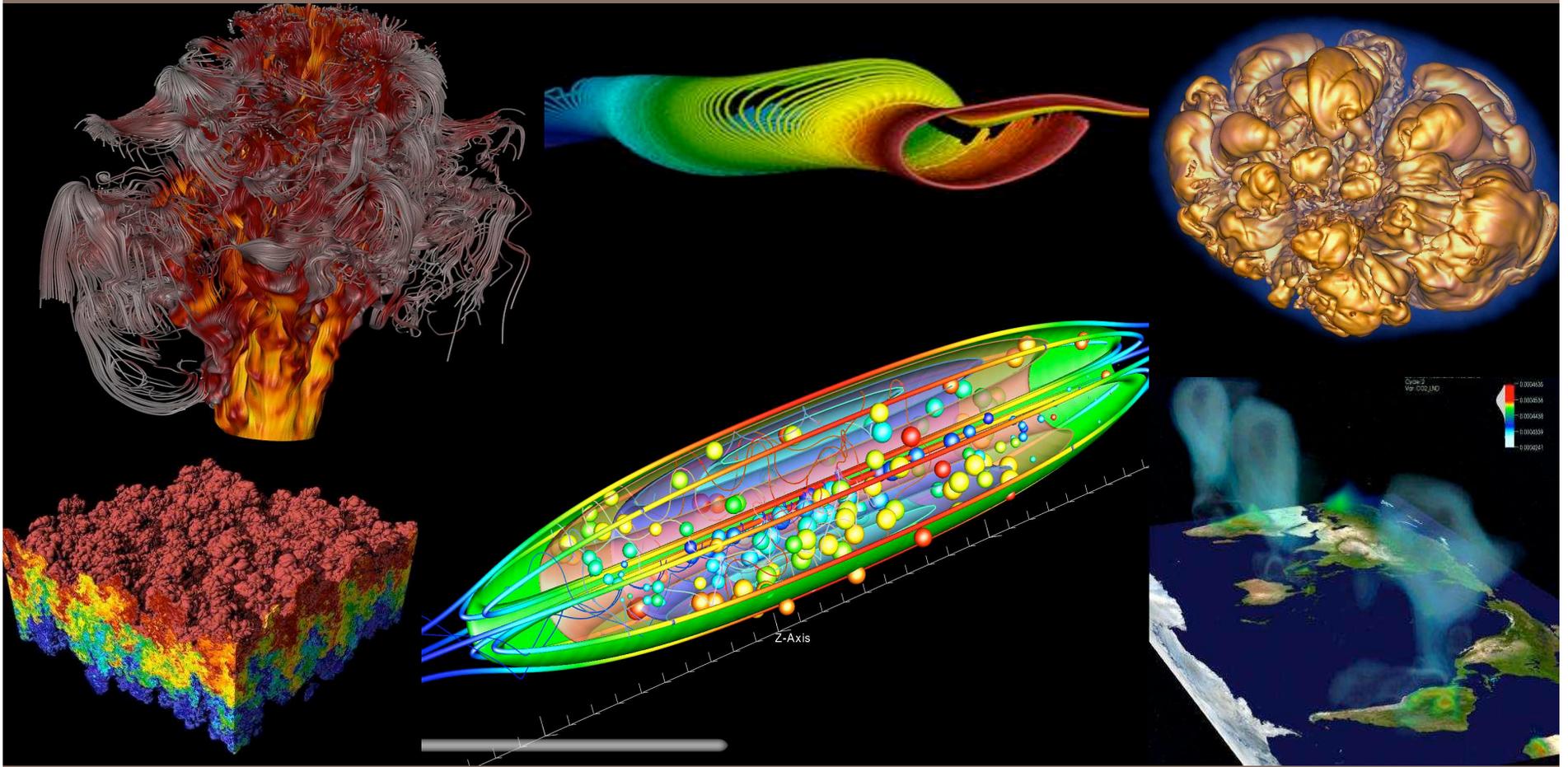


How to build a visualization application for very large data

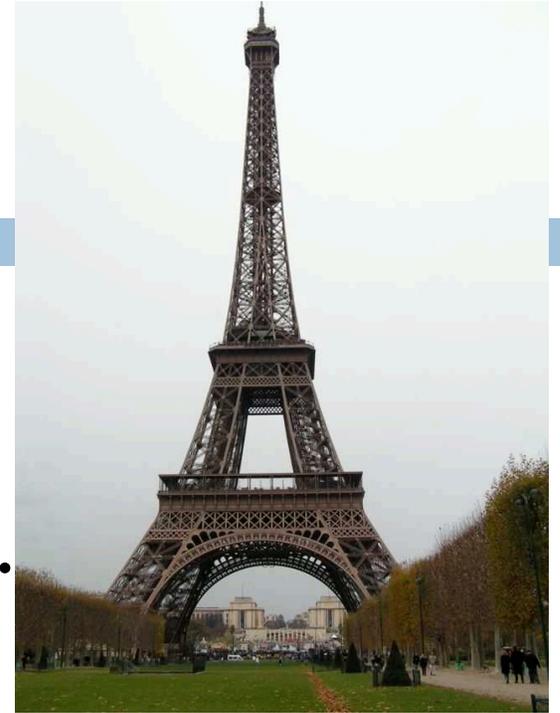


June 13, 2011

Hank Childs, Lawrence Berkeley Lab & UC Davis

Thank you!

- Thank you to Thierry, Laurent, and the organizers...
- Thank you to all of you for coming...
- ... and my family thanks you too!



Why are visualization tools/ frameworks important?

- Visualization and analysis is an enabling technology: it is important that we have products that we can deploy to users.
- Visualization is often explorative
 - ▣ Users want to employ a variety of rich techniques to better understand their data. Visualization tools enable this.
- Visualization tools can be written for many application areas simultaneously.
 - ▣ The large investment it takes to make a tool can pay off by benefiting many groups.

Why are visualization tools/ frameworks important?

- Visualization and analysis is an enabling technology: it is important that we have products that we can deploy to users.

This course is primarily about visualization tools/
frameworks: what techniques they use and how to
make them be successful.

- Visualization tools can be written for many application areas simultaneously.
 - ▣ The large investment it takes to make a tool can pay off by benefiting many groups.

Overview of this course

- Monday, June 13th, 2011
 - Lecture #1 (right now!): 90 minutes
 - Course overview
 - Three Strategies for Three Epochs
 - Lecture #2: 60 minutes
 - Data flow networks
 - VTK
 - OpenDX
 - Lecture #3: 30 minutes
 - MPI overview
 - Overview of Hands-On Session #1
 - Hands-On Session #1
 - Read + Process + Render

Overview of this course



- Tuesday June 14th, 2011
 - Lecture #4: 90 minutes
 - Parallel visualization
 - Architecture
 - “Contracts”
 - Rendering
 - IceT
 - Performance study
 - Overview of Hands-On Session #2
 - Hands-On Session #2
 - Parallelizing your program

Overview of this course



- Wednesday June 15th, 2011
 - Lecture #5: 60 minutes
 - Non-embarrassingly parallel algorithms
 - Lecture #6: 30 minutes
 - Hybrid parallelism
 - Overview of Hands-On Session #3
 - Hands-On Session #3 (for half of you)
 - Accelerating your processing
 - A non-embarrassingly parallel algorithm

Overview of this course



- Thursday June 16th, 2011
 - Lecture #7: 120 minutes
 - Smart techniques:
 - In situ visualization
 - Multi-resolution visualization
 - Query driven visualization
 - Overview of Hands-On Session #4
 - Hands-On Session #3 (for half of you)
 - Accelerating your processing
 - A non-embarrassingly parallel algorithm
 - Hands-On Session #4 (for half of you)
 - Performance analysis

Overview of this course



- Friday June 17th, 2011

- Lecture #8: 120 minutes

- Putting it all together:

- What it takes to deploy a visualization tool

- “Tutorial”

- Demonstrate UI

- Demonstrate power of data flow networks

- Hands-On Session #4 (for half of you)

- Performance analysis

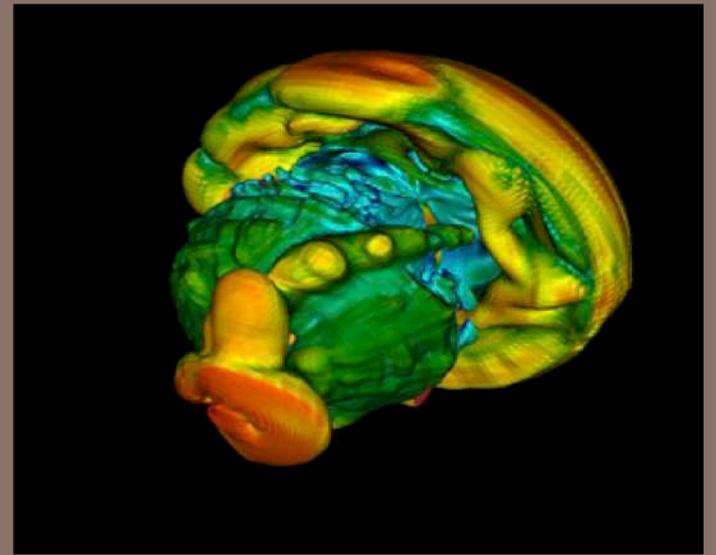
- (I will not attend this session.)

Bonus mini-lecture



- How can we better understand data?
 - High resolution data?
 - Ensembles of data?

{TERA | PETA | EXA}-SCALE VISUALIZATION: Three Strategies For Three Epochs



June 13, 2011

Hank Childs, Lawrence Berkeley Lab & UC Davis

Supercomputing 101

- Why simulation?
 - ▣ Simulations are sometimes more cost effective than experiments.
 - ▣ New model for science has three legs: theory, experiment, and *simulation*.
- FLOPs: How supercomputers are measured
 - ▣ 1 FLOP = 1 Floating point operation per second
 - ▣ 1 GigaFLOP = 1 billion FLOPs
 - ▣ 1 TeraFLOP = 1 000 GigaFLOPs, 1 PetaFlop = 1,000,000 GigaFLOPs, 1 ExaFLOP = 1,000,000,000 GigaFLOPs

Supercomputing 101

- What are the three epochs?
 - ▣ Terascale = TeraFLOP + TeraBytes of memory + Tera/PetaBytes on disk
 - ▣ Petascale = PetaFLOPs + petabytes on disk + petabytes of memory
 - ▣ Exascale = ExaFLOPs + exabyte disk + petabytes of memory
- Why terascale/petascale/exascale?
 - ▣ More compute cycles, more memory, etc, lead for faster and/or more accurate simulations.

Petascale computing arrived in 2009.



An order of magnitude jump in computing in the next year.

- Two ~20PFlop machines will be online in 2011/2012



A really big change in computing happened last year...



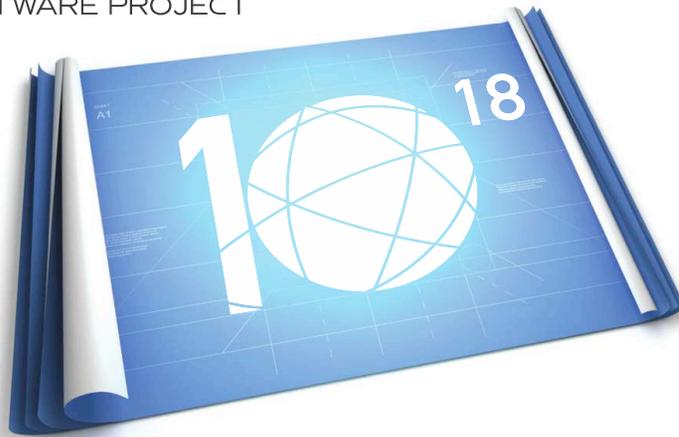
China's Tianhe machine

... and this change is the springboard to exascale computing.

International Exascale Software Project

www.exascale.org

INTERNATIONAL **EXASCALE** ROADMAP 1.0 SOFTWARE PROJECT



The International Exascale
Software Roadmap,
J. Dongarra, P. Beckman, et al.,
*International Journal of High
Performance Computer
Applications* **25**(1), 2011, ISSN
1094-3420. (Publ. 6 Jan 2011)

Jack Dongarra
Pete Beckman
Terry Moore
Patrick Aerts
Giovanni Aloisio
Jean-Claude Andre
David Barkai
Jean-Yves Berthou
Taisuke Boku
Bertrand Braunschweig
Franck Cappello
Barbara Chapman
Xuebin Chi

Alok Choudhary
Sudip Dosanjh
Thom Dunning
Sandro Fiore
Al Geist
Bill Gropp
Robert Harrison
Mark Herold
Michael Heroux
Adolfy Hoisie
Koh Hotta
Yutaka Ishikawa
Fred Johnson

Sanjay Kale
Richard Kenway
David Keyes
Bill Kramer
Jesus Labarta
Alain Lichniewsky
Thomas Lippert
Bob Lucas
Barney Maccabe
Satoshi Matsuoka
Paul Messina
Peter Michielse
Bernd Mohr

Matthias Mueller
Wolfgang Nagel
Hiroshi Nakashima
Michael E. Papka
Dan Reed
Mitsuhsa Sato
Ed Seidel
John Shalf
David Skinner
Marc Snir
Thomas Sterling
Rick Stevens
Fred Streitz

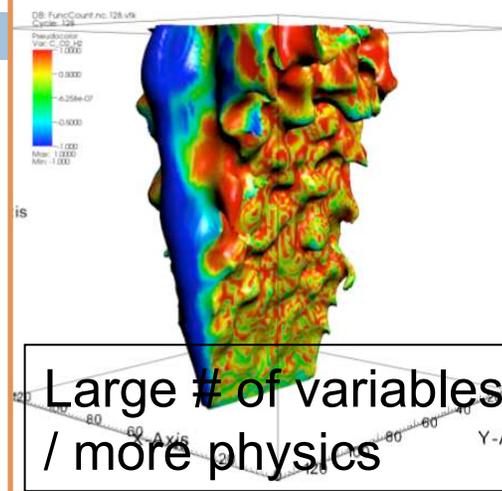
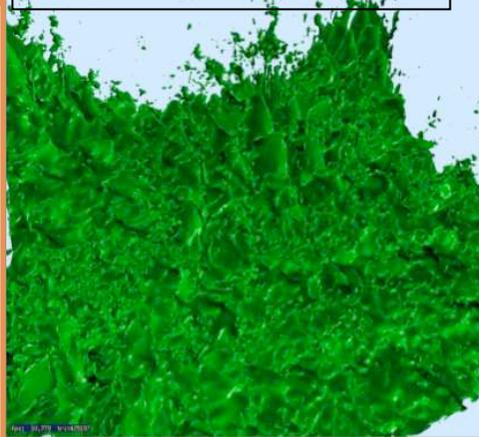
Bob Sugar
Shinji Sumimoto
William Tang
John Taylor
Rajeev Thakur
Anne Trefethen
Mateo Valero
Aad van der Steen
Jeffrey Vetter
Peg Williams
Robert Wsniewski
Kathy Yelick

SPONSORS



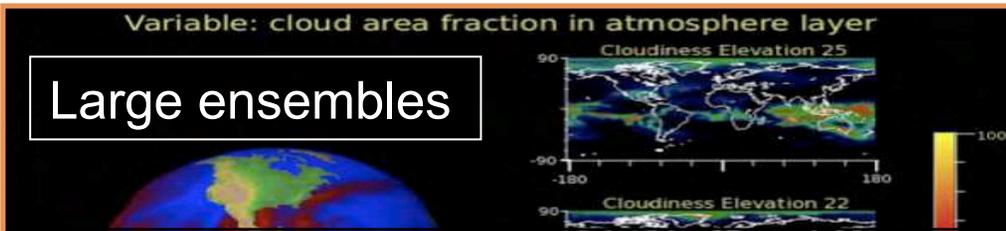
How does increased computing power affect the data to be visualized?

High-res meshes

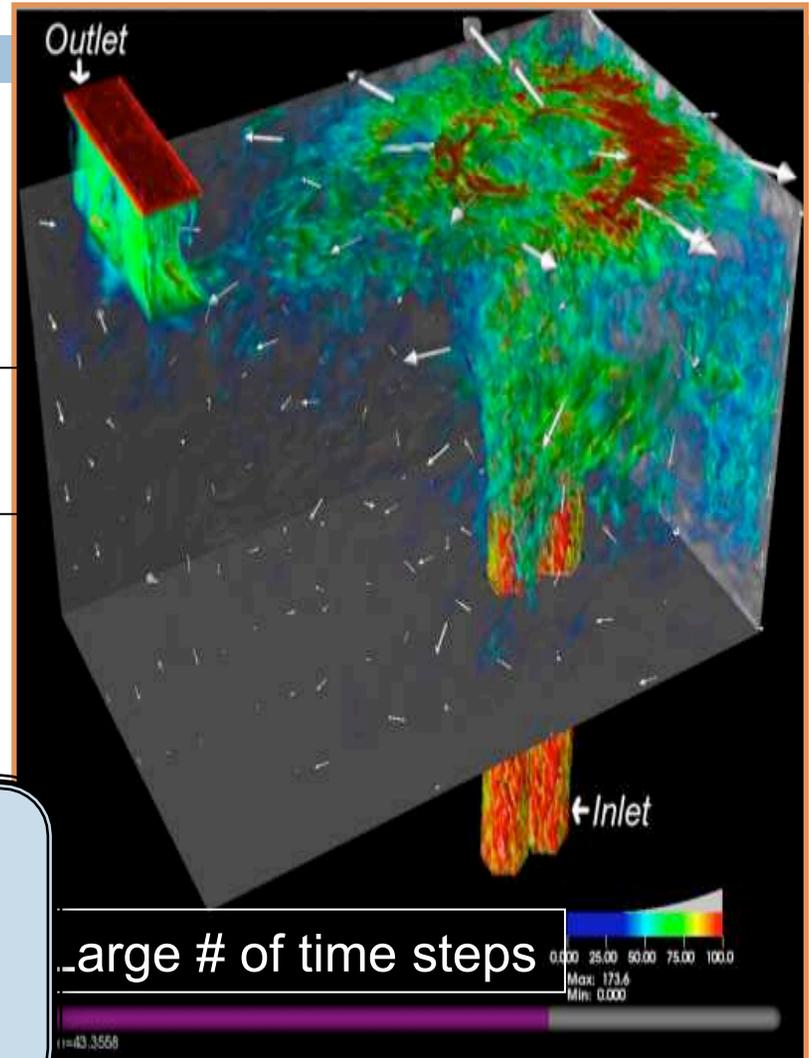


Large # of variables / more physics

Large ensembles



Your mileage may vary; some simulations produce a lot of data and some don't.



Large # of time steps

Thanks!: Sean Ahern & Ken Joy

Some history behind this presentation...

SciDAC 2007



- ▣ “Architectural Problems and Solutions for Petascale Visualization and Analysis”



Some history behind this presentation...



- “Why Petascale Visualization Will Changes The Rules”



Some history behind this presentation...



- “Why Petascale Visualization Will Changes The Rules”



Eurographics Symposium on
Parallel Graphics and Visualization

“Exascale Visualization: Get Ready For a Whole New World”

Fable: The Boy Who Cried Wolf

- Once there was a shepherd boy who had to look after a flock of sheep. One day, he felt bored and decided to play a trick on the villagers. He shouted, “Help! Wolf! Wolf!” The villagers heard his cries and rushed out of the village to help the shepherd boy. When they reached him, they asked, “Where is the wolf?” The shepherd boy laughed loudly, “Ha, Ha, Ha! I fooled all of you. I was only playing a trick on you.”

Fable: The Boy Who Cried Wolf

- Once there was a **viz expert** who had to look after **customers**. One day, he **needed funding** and decided to play a trick on **his funders**. He shouted, “Help! **Big Big Data!**” The **funders** heard his cries and **sent lots of money** to help the **viz expert**. When **petascale arrived**, they asked, “Where is the **problem?**” The **viz expert shrugged and said**, “The **problem isn’t quite here yet, but it will be soon.**”

This is NOT the story of this presentation.

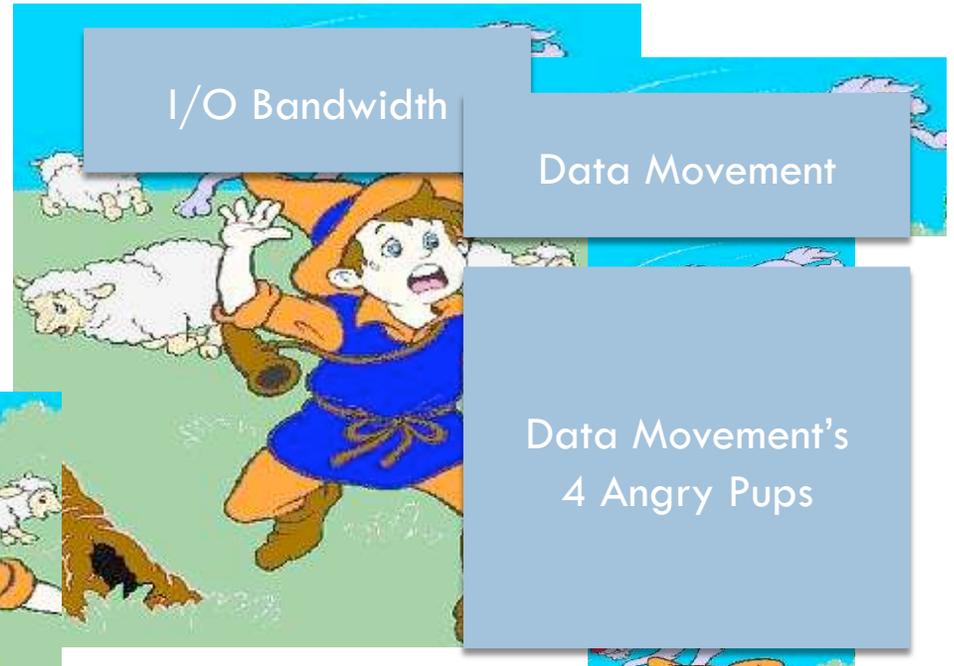
The message from this presentation...



Petascale Visualization



Terascale Visualization



Exascale Visualization

Outline



- The Terascale Strategy
- The I/O Wolf & Petascale Visualization
- An Overview of the Exascale Machine
- The Data Movement Wolf and Its 4 Angry Pups
- Under-represented topics
- Conclusions

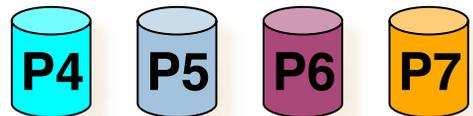
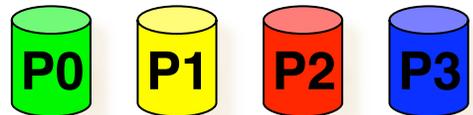
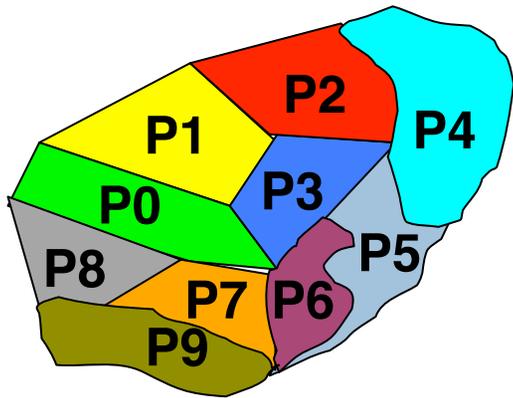
Outline



- **The Terascale Strategy**
- The I/O Wolf & Petascale Visualization
- An Overview of the Exascale Machine
- The Data Movement Wolf and Its 4 Angry Pups
- Under-represented topics
- Conclusions

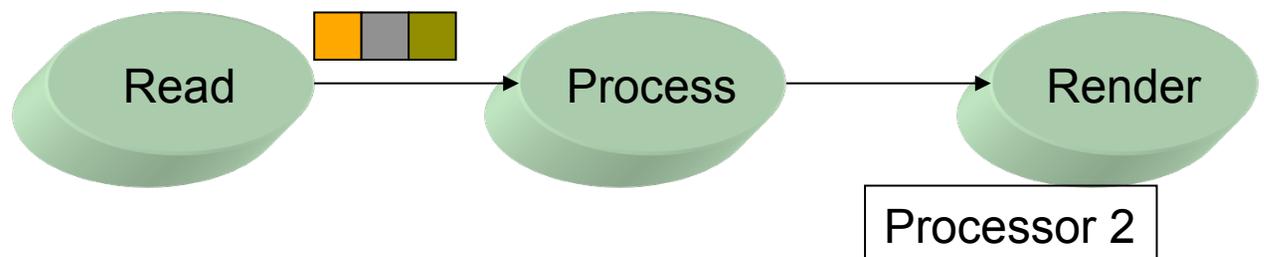
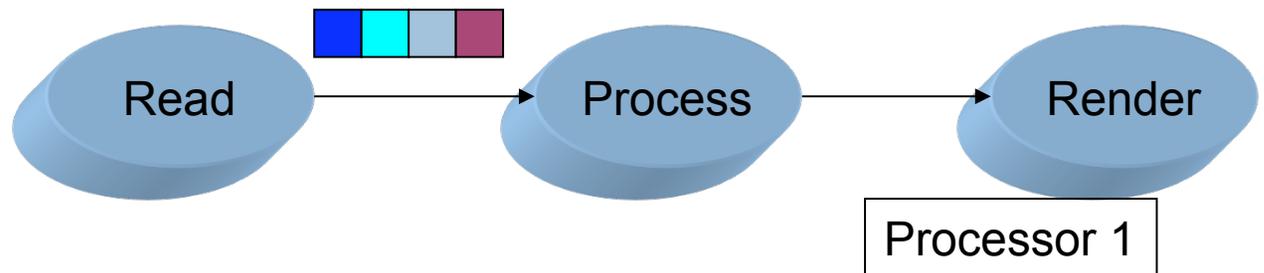
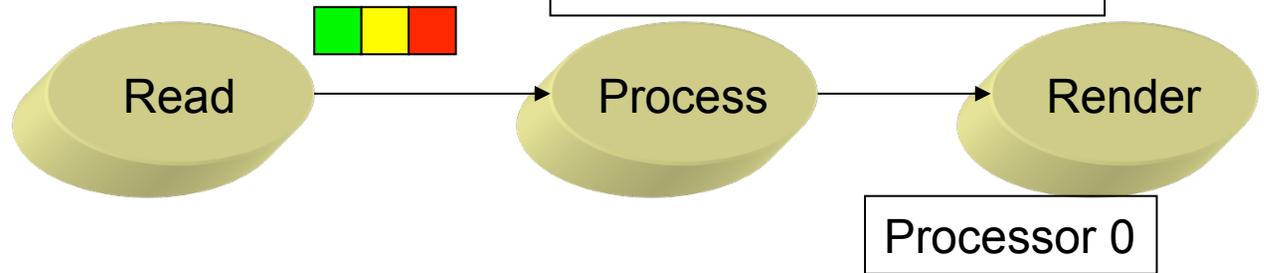
Production visualization tools use “pure parallelism” to process data.

Parallel Simulation Code



Pieces of data
(on disk)

Parallelized visualization data flow network



Pure parallelism



- Pure parallelism is data-level parallelism, but...
 - Multi-resolution can be data-level parallelism
 - Out-of-core can be data-level parallelism
- Pure parallelism: “brute force” ... processing full resolution data using data-level parallelism
- Pros:
 - Easy to implement
- Cons:
 - Requires large I/O capabilities
 - Requires large amount of primary memory

Pure parallelism and today's tools



- Three of the most popular end user visualization tools -- VisIt, ParaView, & EnSight -- primarily employ a pure parallelism + client-server strategy.
 - ▣ All tools working on advanced techniques as well
- Of course, there's lots more technology out there besides those three tools...

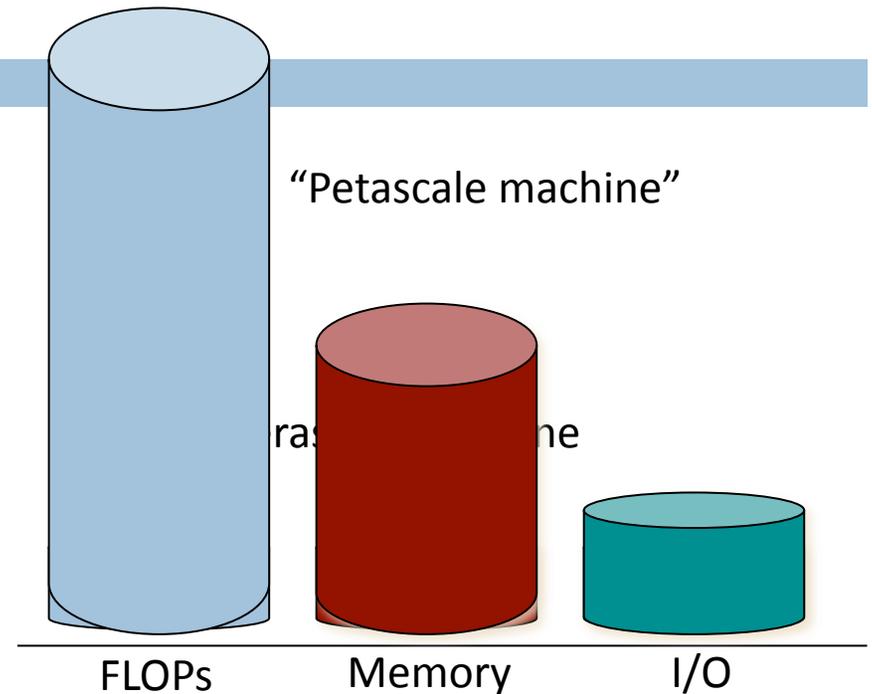
Outline



- The Terascale Strategy
- **The I/O Wolf & Petascale Visualization**
- An Overview of the Exascale Machine
- The Data Movement Wolf and Its 4 Angry Pups
- Under-represented topics
- Conclusions

I/O and visualization

- Pure parallelism is almost always $>50\%$ I/O and sometimes 98% I/O
- Amount of data to visualize is typically $O(\text{total mem})$
- Two big factors:
 - ① how much data you have to read
 - ② how fast you can read it
- \rightarrow Relative I/O (ratio of total memory and I/O) is key



Trends in I/O

Machine	Year	Time to write memory
ASCI Red	1997	300 sec
ASCI Blue Pacific	1998	400 sec
ASCI White	2001	660 sec
ASCI Red Storm	2004	660 sec
ASCI Purple	2005	500 sec
Jaguar XT4	2007	1400 sec
Roadrunner	2008	1600 sec
Jaguar XT5	2008	1250 sec

c/o David Pugmire, ORNL

Why is relative I/O getting slower?



- I/O is quickly becoming a dominant cost in the overall supercomputer procurement.
 - ▣ And I/O doesn't pay the bills.
- Simulation codes aren't as exposed.

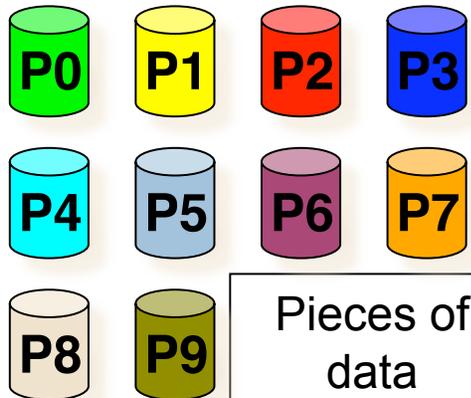
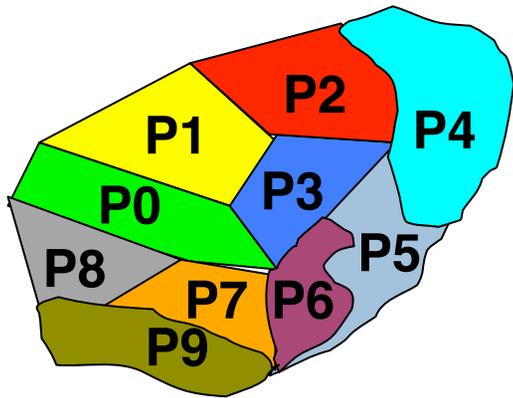
We need to de-emphasize I/O in our visualization and analysis techniques.

There are “smart techniques” that de-emphasize memory and I/O.

- Out of core
- Data subsetting
- Multi-resolution
- In situ

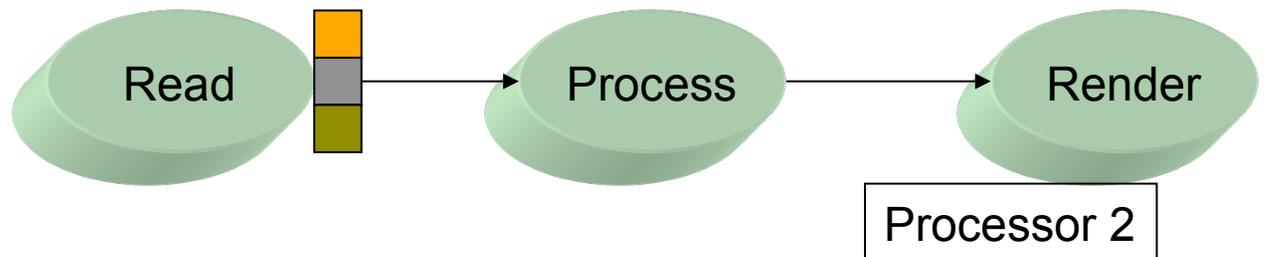
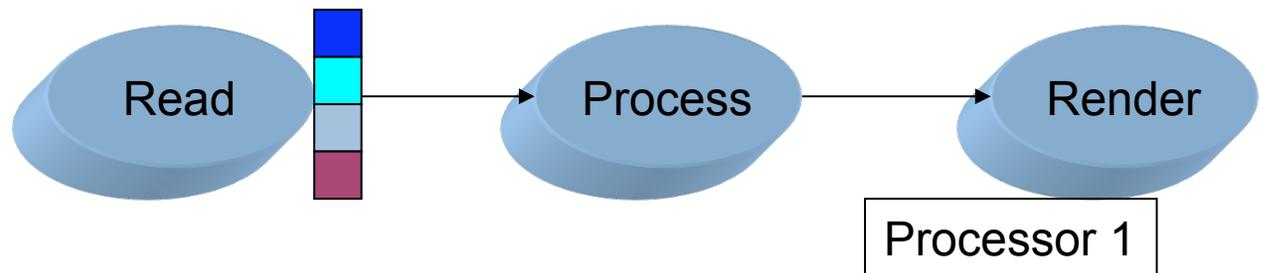
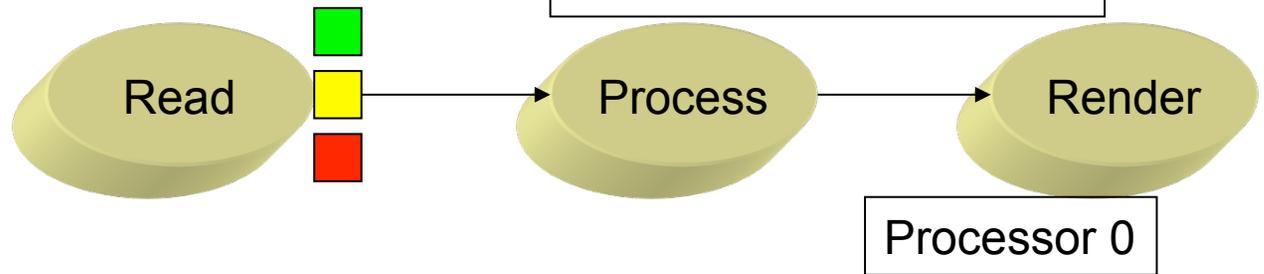
Out-of-core iterates pieces of data through the pipeline one at a time.

Parallel Simulation Code



Pieces of data
(on disk)

Parallelized visualization
data flow network



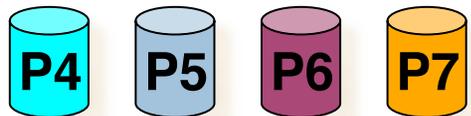
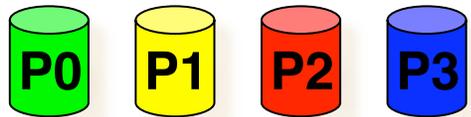
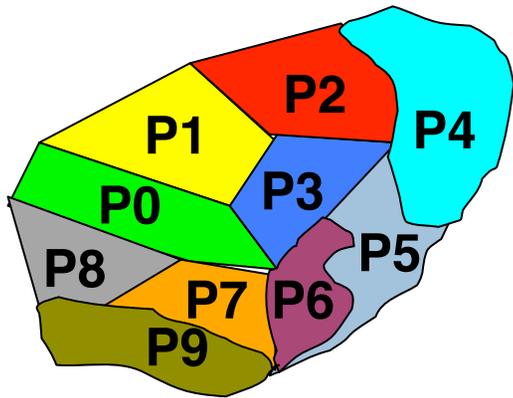
Out-of-core: pros and cons



- Pros:
 - Lower requirement for primary memory
 - Doesn't require big machines
- Cons:
 - Still paying large I/O costs
 - Slow

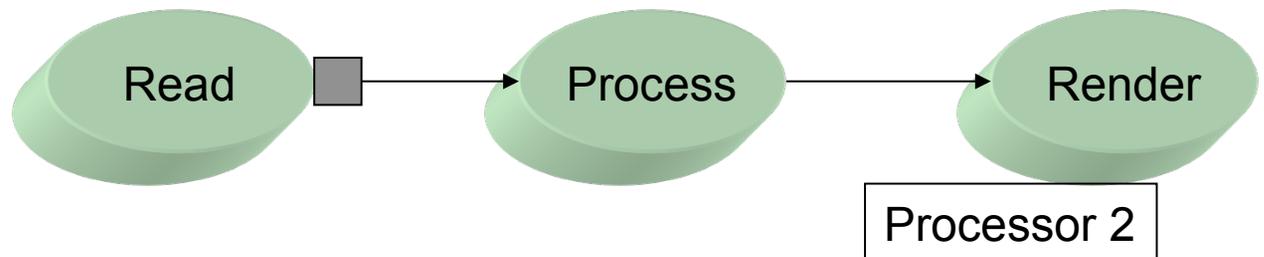
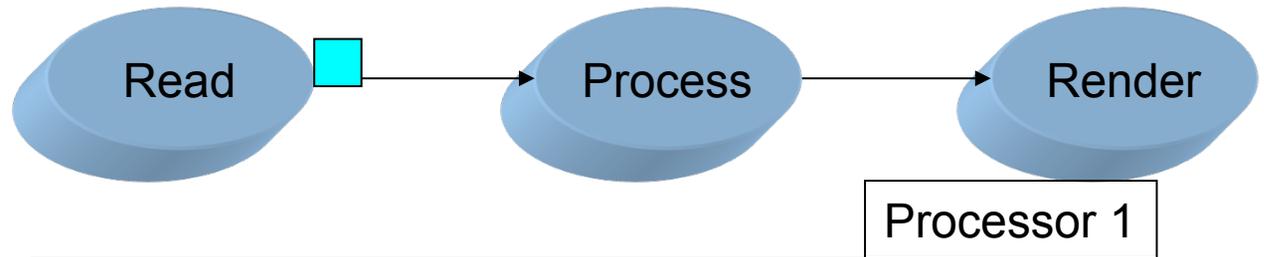
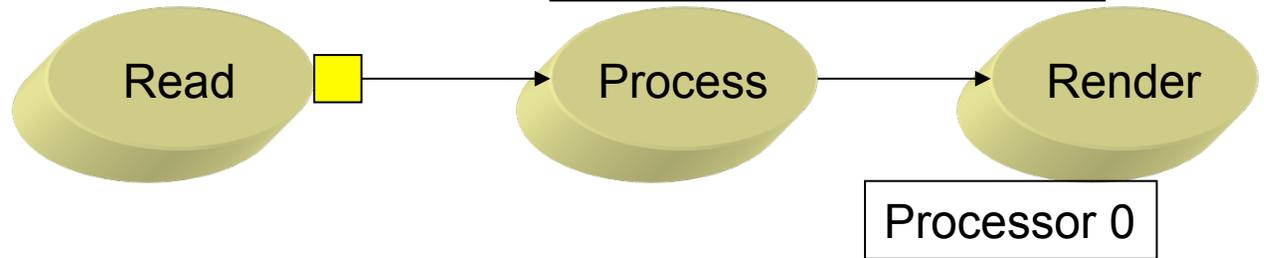
Data subsetting eliminates pieces that don't contribute to the final picture.

Parallel Simulation Code



Pieces of data
(on disk)

Parallelized visualization
data flow network



Data Subsetting: pros and cons



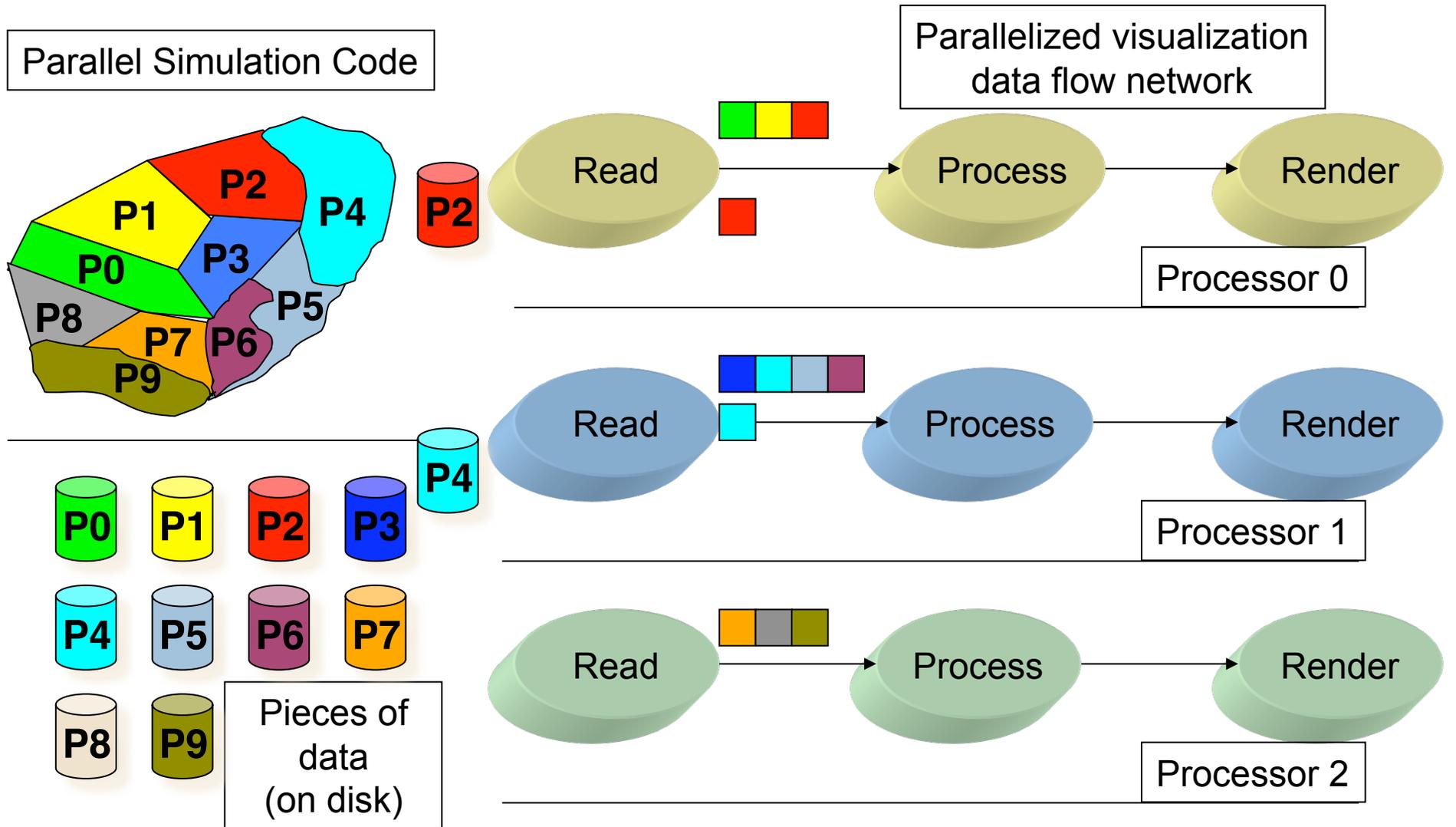
- Pros:

- Less data to process (less I/O, less memory)

- Cons:

- Only applicable to some algorithms

Multi-resolution techniques use coarse representations then refine.



Multi-resolution techniques

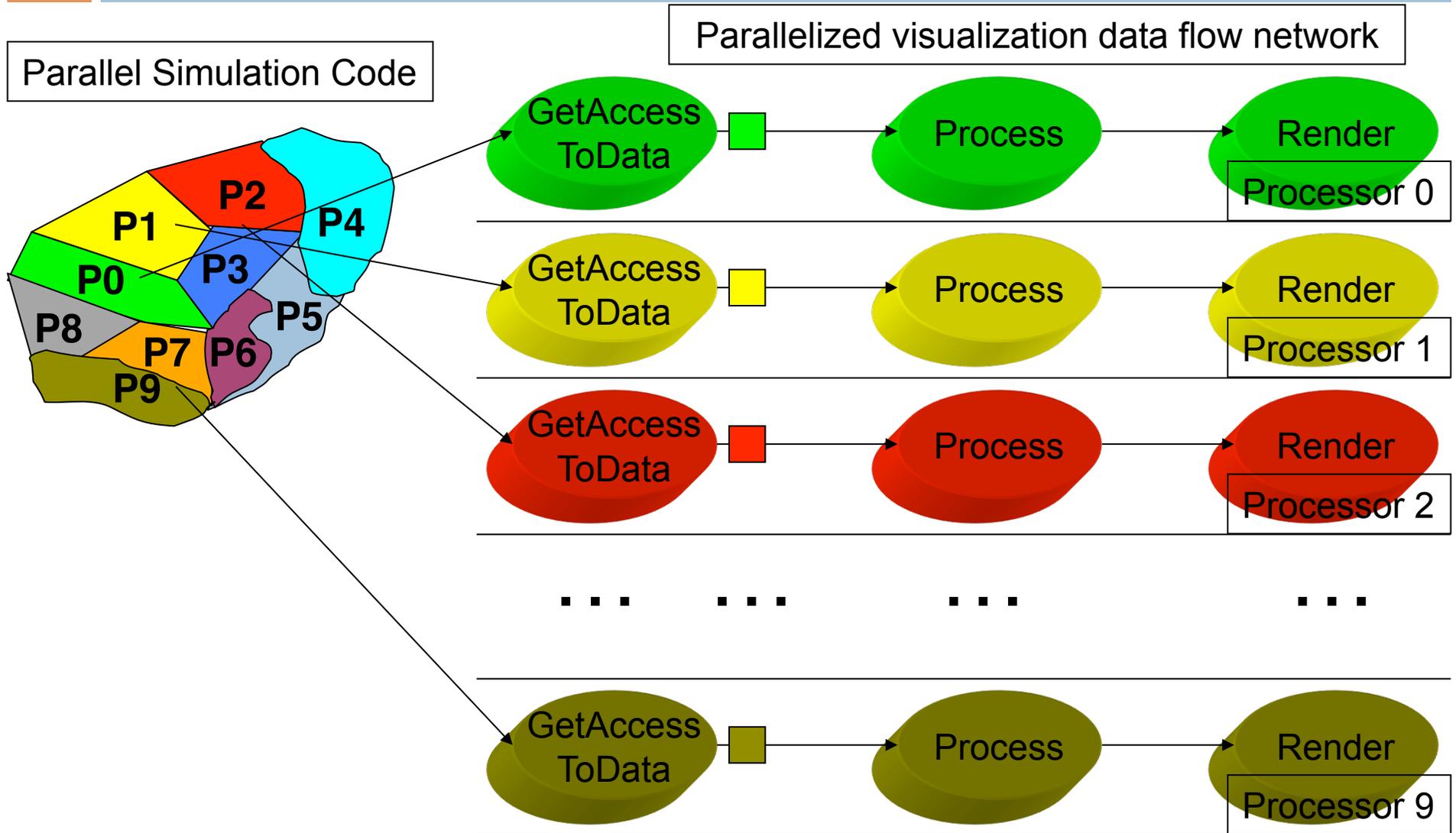
- Pros

- Drastically reduce I/O & memory requirements
- Confidence in pictures; multi-res hierarchy addresses “many cells to one pixel issue”

- Cons

- Not always meaningful to process simplified version of the data.
- How do we generate hierarchical representations during dump? What costs do they incur (data movement costs, storage costs)?

In situ processing does visualization as part of the simulation.

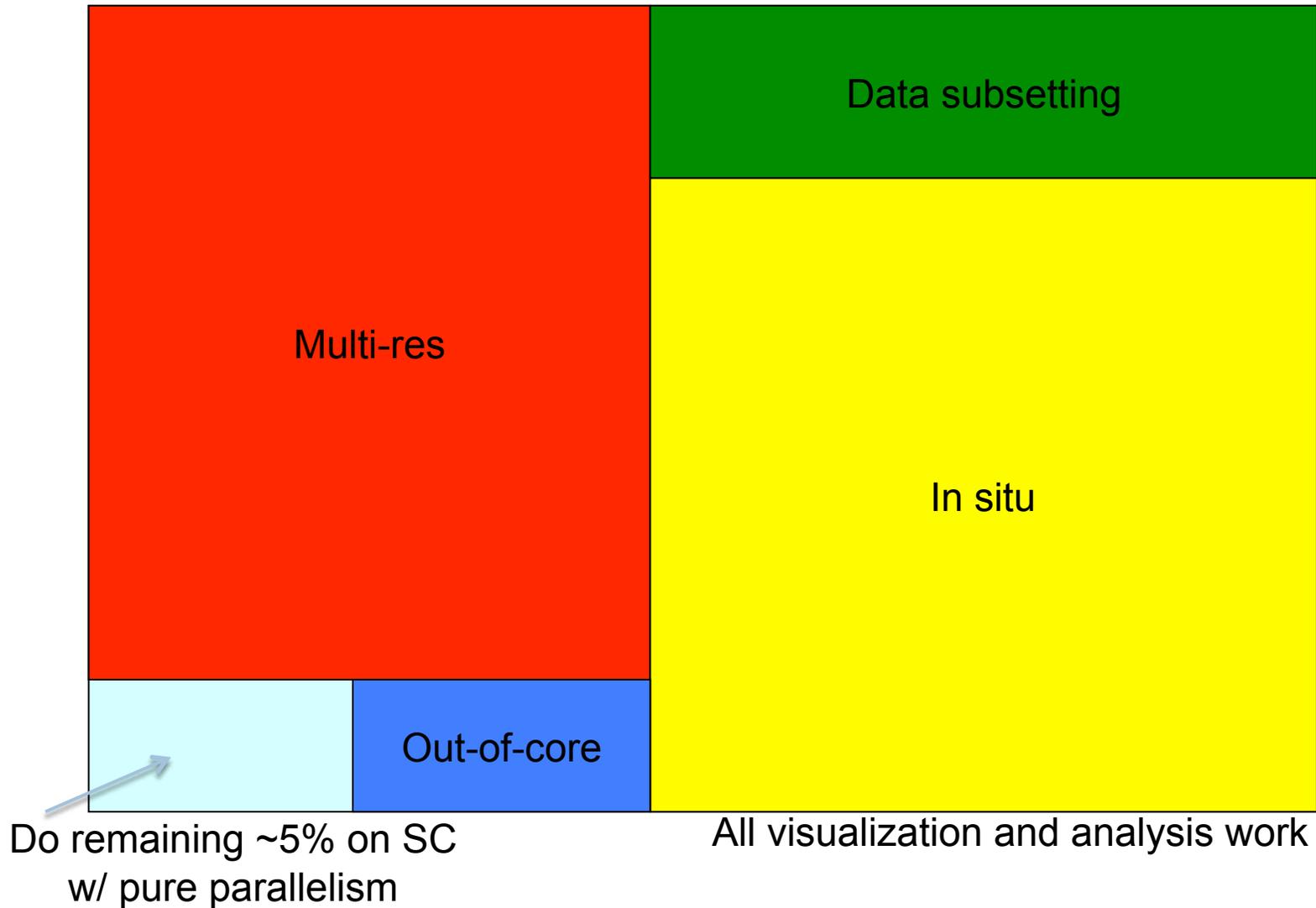


In situ



- In situ processing can mean multiple things
 - Will discuss this more later in the talk
- Common perceptions of in situ
 - Pros:
 - No I/O & plenty of compute
 - Cons:
 - Very memory constrained
 - Some operations not possible
 - Once the simulation has advanced, you cannot go back and analyze it
 - User must know what to look a priori

Petascale visualization will likely require a lot of solutions.



Outline



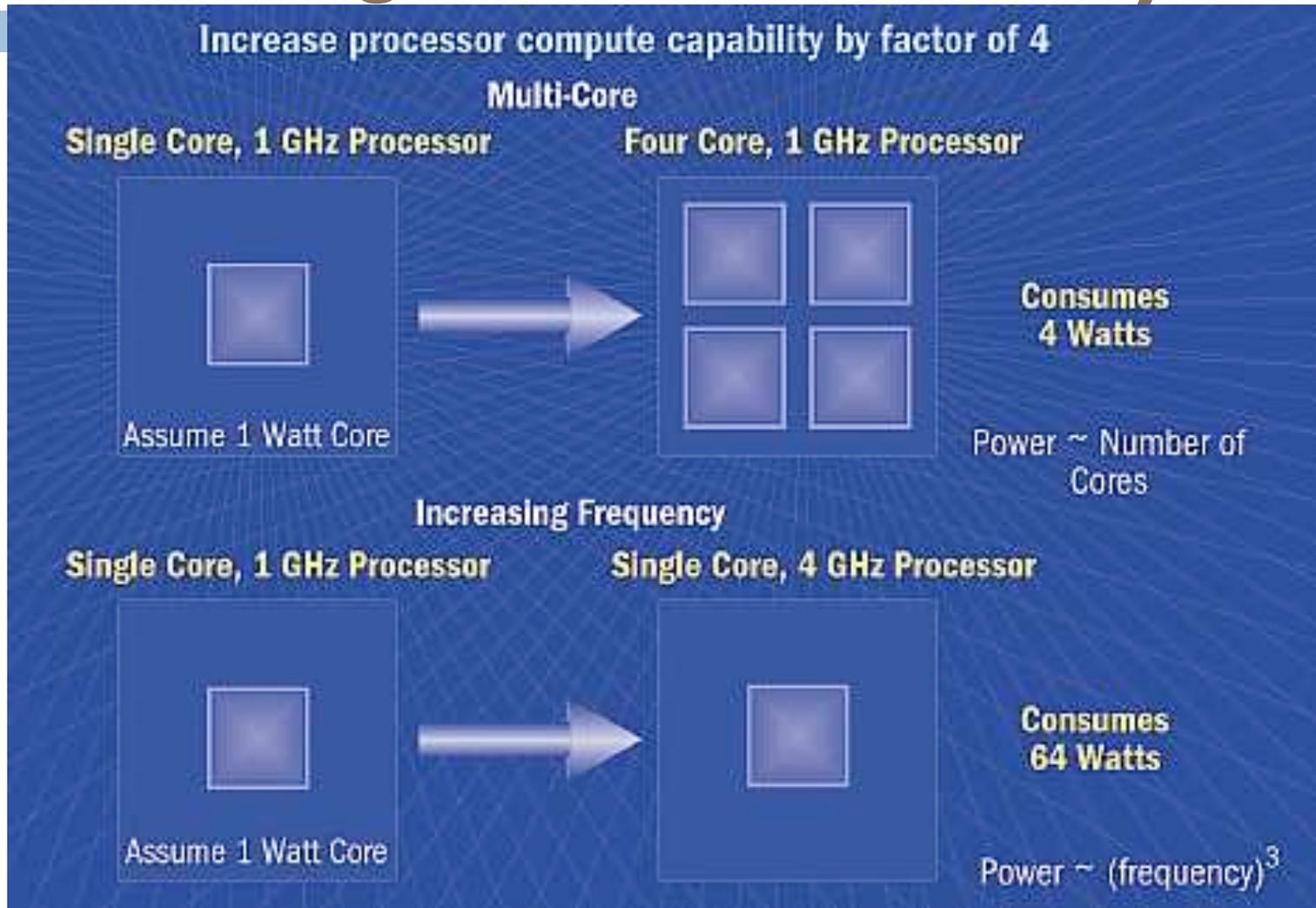
- The Terascale Strategy
- The I/O Wolf & Petascale Visualization
- **An Overview of the Exascale Machine**
- The Data Movement Wolf and Its 4 Angry Pups
- Under-represented topics
- Conclusions

Exascale assumptions



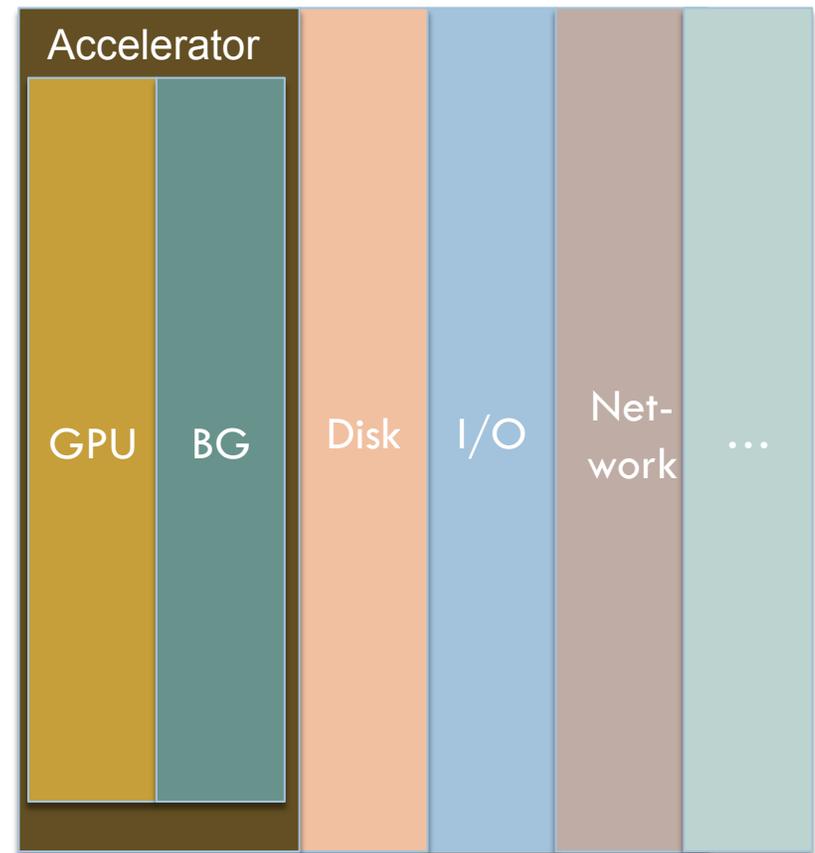
- The machine will be capable of one exaflop.
- The machine will cost $< \$200\text{M}$.
- The machine will use $< 20\text{MW}$.
- The machine may arrive as early as 2018.

Hurdle #1: power requires slower clocks and greater concurrency

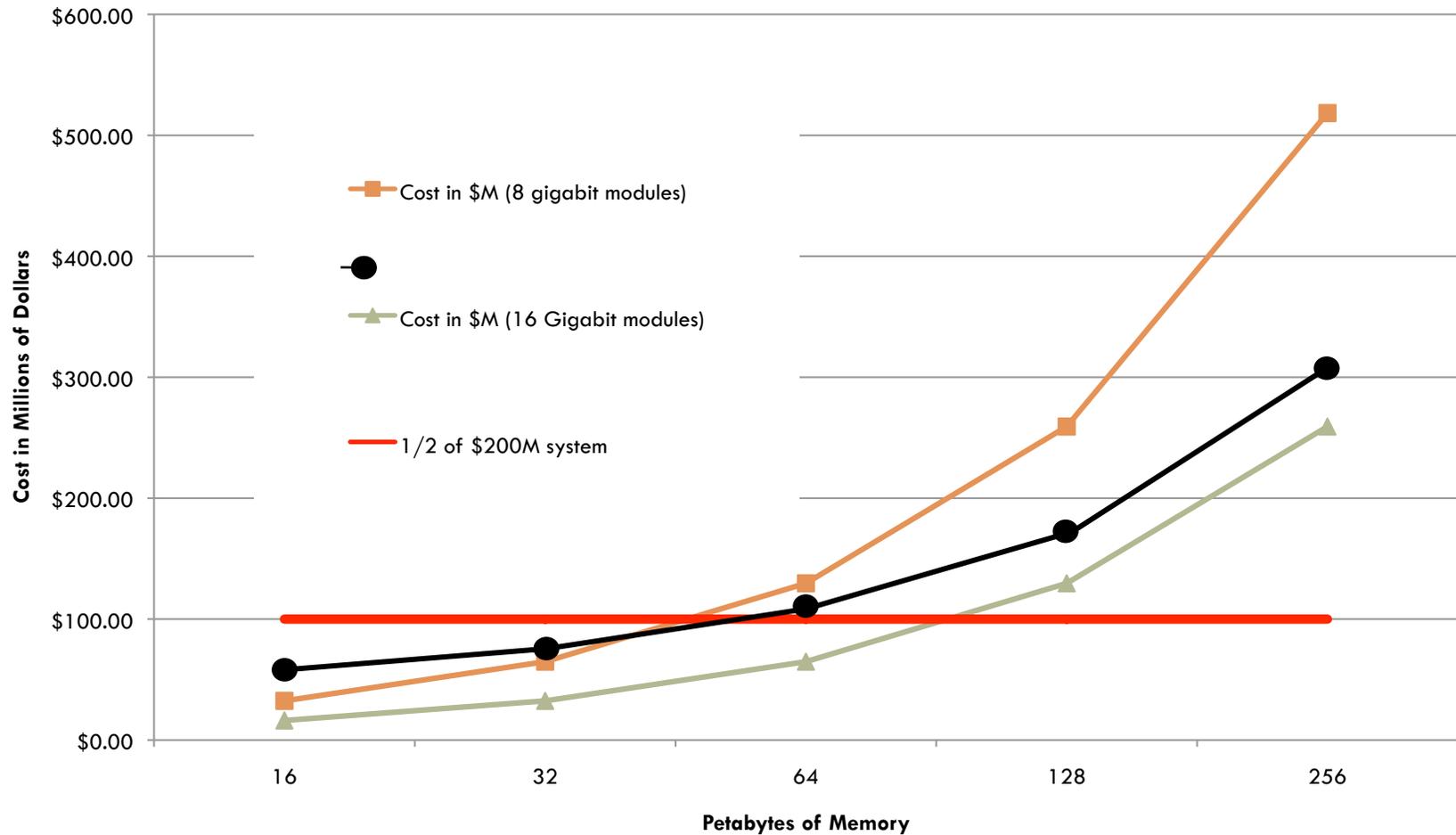


Accelerator technologies

- Currently simultaneously thinking about two different accelerator technologies:
 - IBM BlueGene's successor – some architectural merger of BlueGene, Power, and Cell
 - GPU / GPU evolution
- Referred to as “swim lanes”: a visual element used in process flow diagrams, or flowcharts, that visually distinguishes responsibilities for sub-processes of a business process.

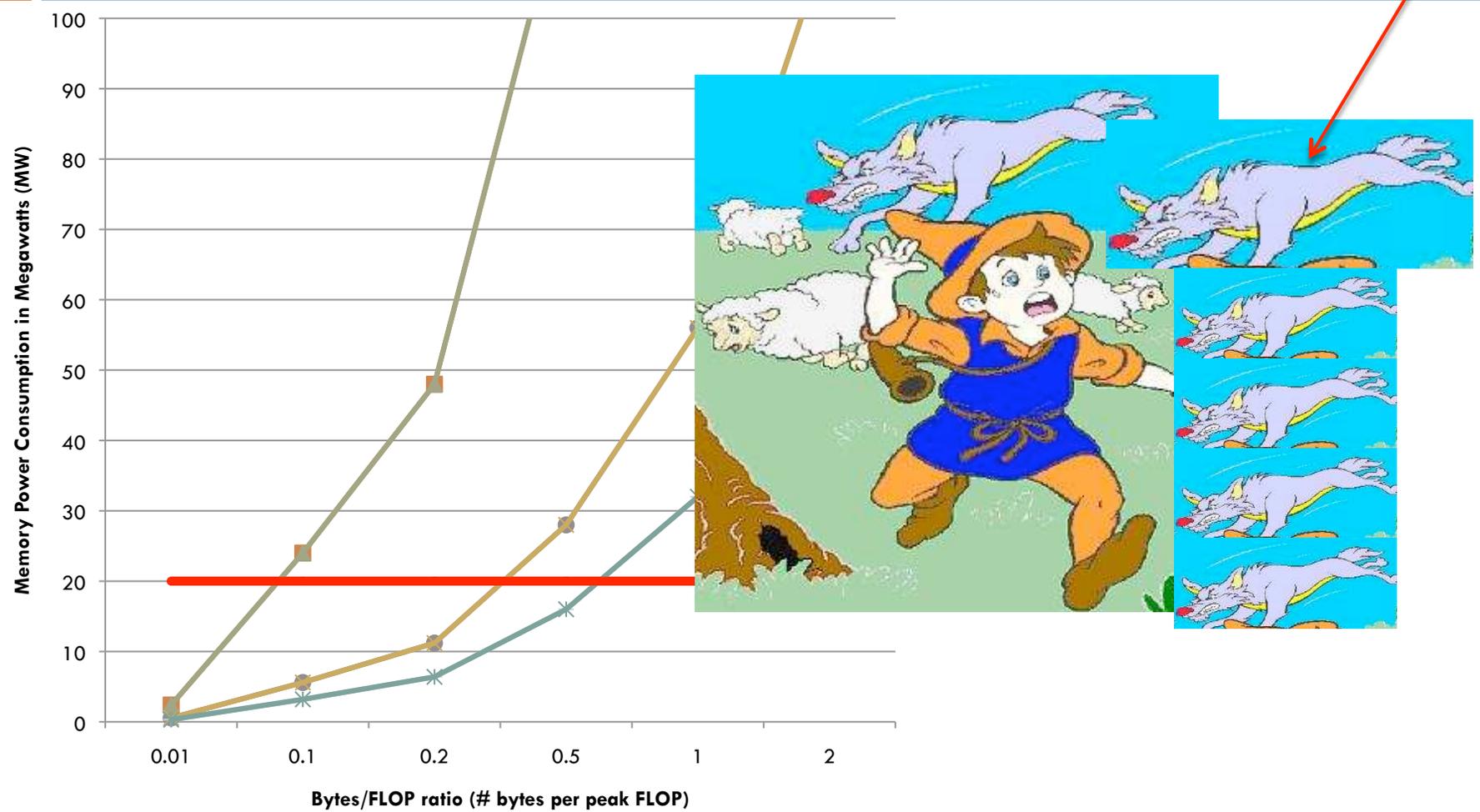


Hurdle #2: memory capacity eats up the entire fiscal budget



c/o John Shalf, LBNL

Hurdle #3: memory bandwidth eats up the entire power budget



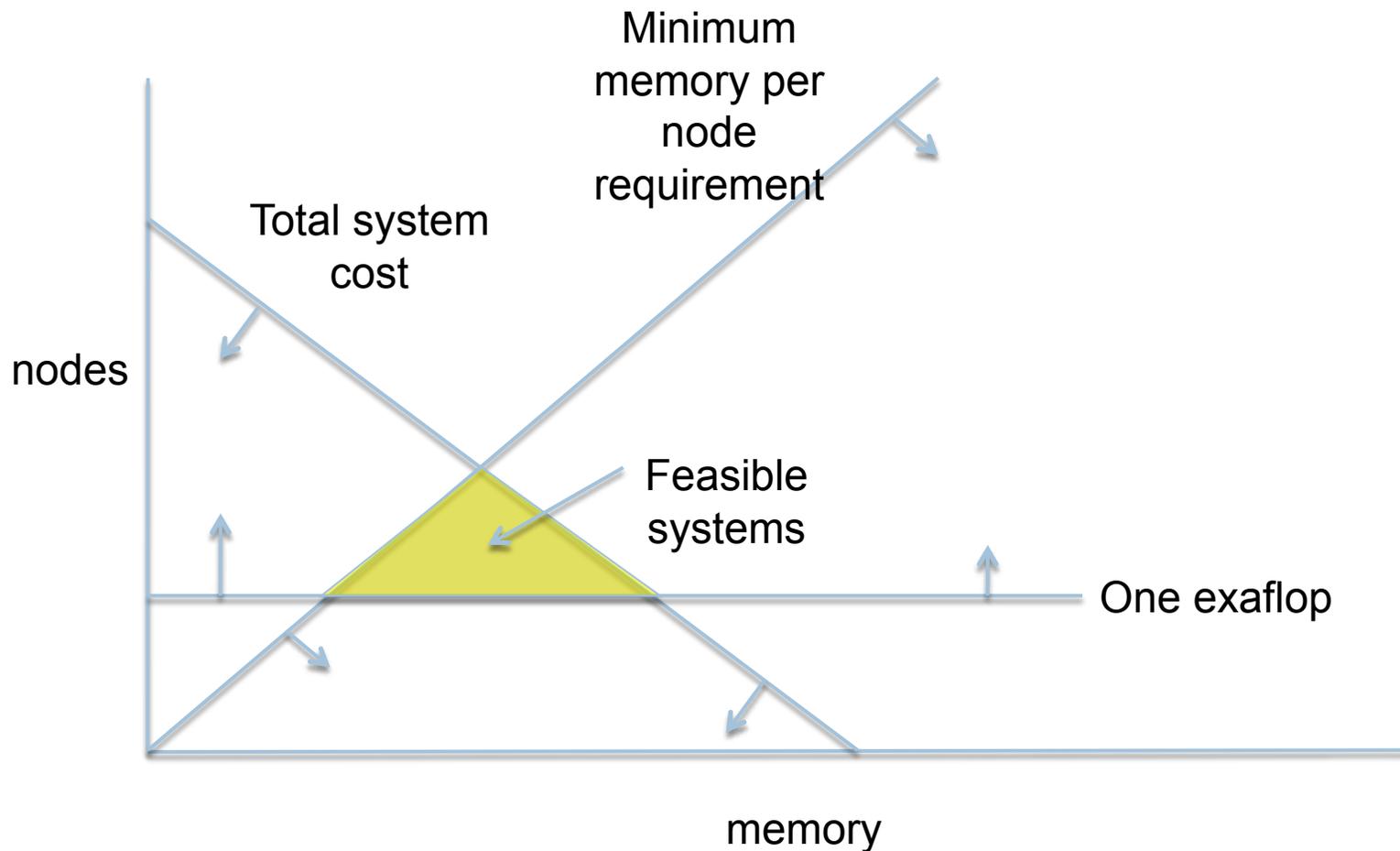
c/o John Shalf, LBNL

The change in memory bandwidth to compute ratio will lead to new approaches.

□ Example: linear solvers

- They start with a rough approximation and converge through an iterative process.
 - $1.125 \rightarrow 1.1251 \rightarrow 1.125087 \rightarrow 1.12508365$
- Each iteration requires sending some numbers to neighboring processors to account for neighborhoods split over multiple nodes.
- Proposed exascale technique: devote some threads of the accelerator to calculating the difference from the previous iteration and just sending the difference.
 - Takes advantage of “free” compute and minimizes expensive memory movement.

The trade space for exascale is very complex.



c/o A. White, LANL

Exascale: a heterogeneous, distributed memory GigaHz KiloCore MegaNode system

Systems	2009	2018	Difference Today & 2018
System peak	2 Pflop/s	1 Eflop/s	O(1000)
Power	6 MW	~20 MW	~3
System memory	0.3 PB	32 - 64 PB [.03 Bytes/Flop]	O(100)
Node performance	125 GF	1,2 or 15TF	O(10) - O(100)
Node memory BW	25 GB/s	2 - 4TB/s [.002 Bytes/Flop]	O(100)
Node concurrency	12	O(1k) or 10k	O(100) - O(1000)
Total Node Interconnect BW	3.5 GB/s	200-400GB/s (1:4 or 1:8 from memory BW)	O(100)
System size (nodes)	18,700	O(100,000) or O(1M)	O(10) - O(100)
Total concurrency	225,000	O(billion) [O(10) to O(100) for latency hiding]	O(10,000)
Storage	15 PB	500-1000 PB (>10x system memory is min)	O(10) - O(100)
IO	0.2 TB	60 TB/s (how long to drain the machine)	O(100)
MTTI	days	O(1 day)	- O(10)

c/o P. Beckman, Argonne

Architectural changes will make writing fast and reading slow.

- Great idea: put SSDs on the node
 - Great idea for the simulations ...
 - ... scary world for visualization and analysis
 - We have lost our biggest ally in lobbying the HPC procurement folks
 - We are unique as data consumers.
- \$200M is not enough...
 - The quote: “1 /3 memory, 1 /3 I/O, 1 /3 networking ... and the flops are free”
 - Budget stretched to its limit and won't spend more on I/O.

Summarizing exascale visualization



- Hard to get data off the machine.
 - And we can't read it in if we do get it off.
- Hard to even move it around the machine.

- → Beneficial to process the data in situ.

Outline



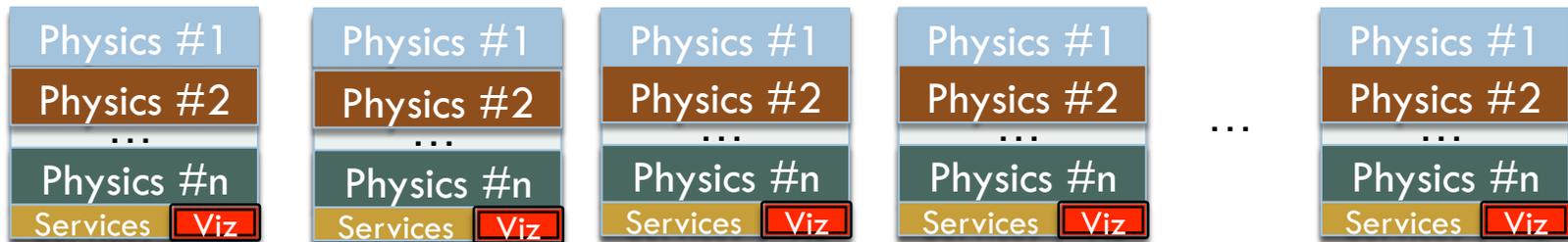
- The Terascale Strategy
- The I/O Wolf & Petascale Visualization
- An Overview of the Exascale Machine
- The Data Movement Wolf and Its 4 Angry Pups
 - ▣ Pup #1: In Situ Systems Research
- Under-represented topics
- Conclusions

Summarizing flavors of in situ

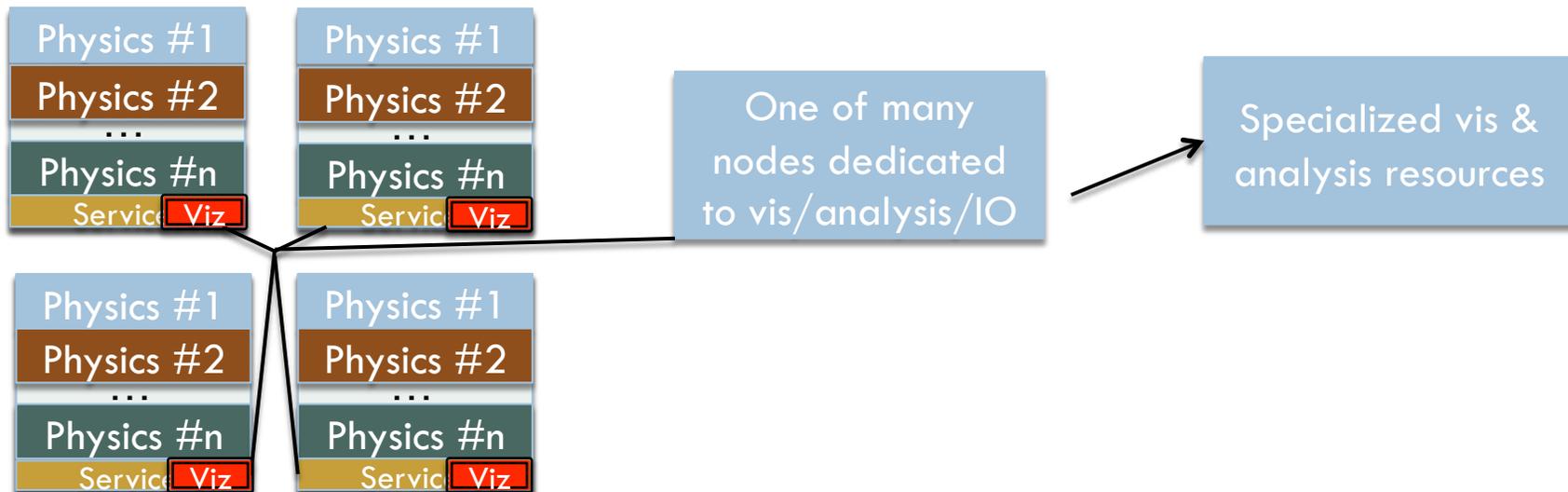
In Situ Technique	Aliases	Description	Negative Aspects
Tightly coupled	Synchronous, co-processing	Visualization and analysis have direct access to memory of simulation code	1) Very memory constrained 2) Large potential impact (performance, crashes)
Loosely coupled	Asynchronous, concurrent	Visualization and analysis run on concurrent resources and access data over network	1) Data movement costs 2) Requires separate resources
Hybrid		Data is reduced in a tightly coupled setting and sent to a concurrent resource	1) Complex 2) Shares negative aspects (to a lesser extent) of others

Possible in situ visualization scenarios

Visualization could be a service in this system (tightly coupled)...

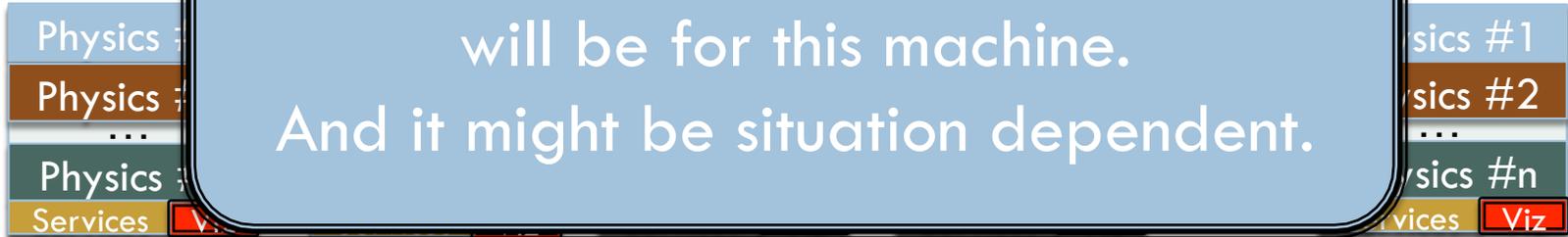


... or visualization could be done on a separate node located nearby dedicated to visualization/analysis/IO/etc. (loosely coupled)



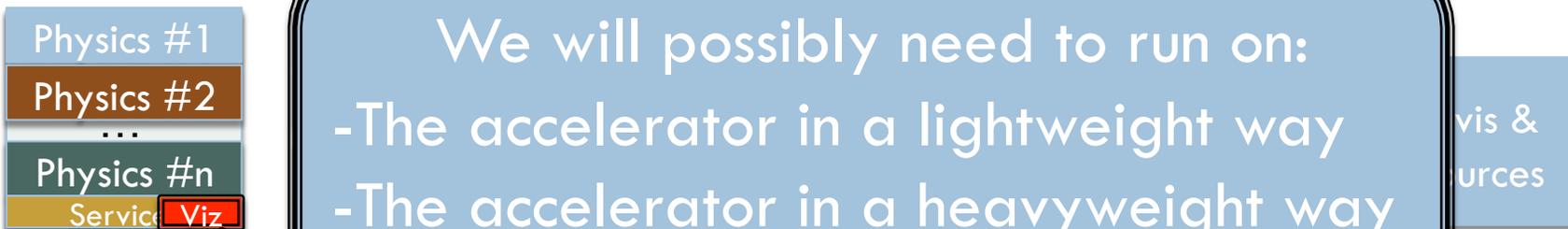
Possible in situ visualization scenarios

Visualization

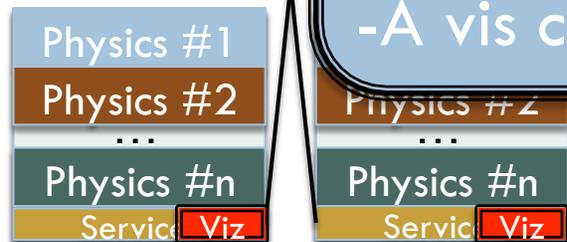


We don't know what the best technique will be for this machine. And it might be situation dependent.

... or visualization could be done on a separate node located nearby dedicated to visualization/analysis/IO/etc. (loosely coupled)

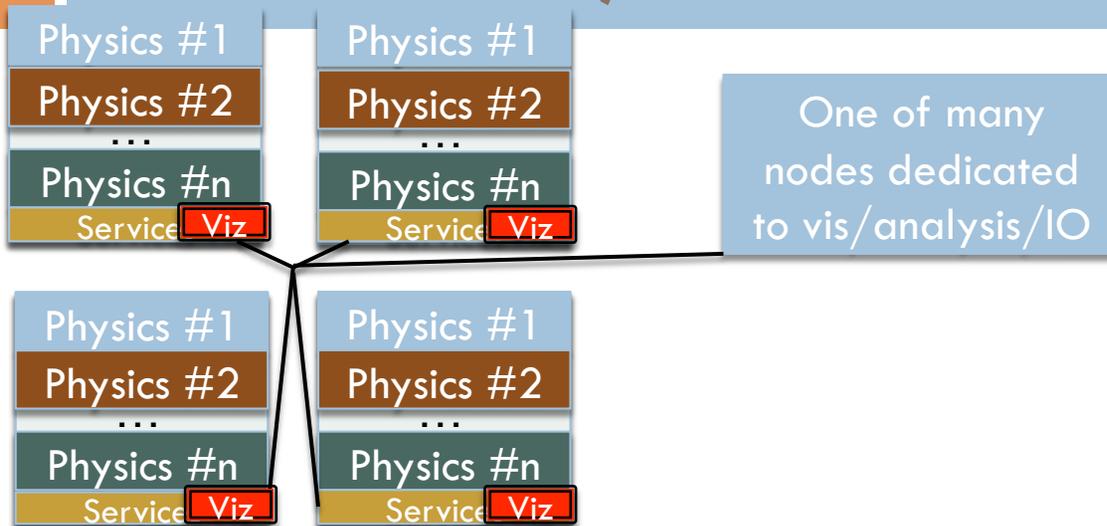


We will possibly need to run on:
-The accelerator in a lightweight way
-The accelerator in a heavyweight way
-A vis cluster (?)



the data is reduced and sent to dedicated resources off machine!
to high performance machine (e.g. GPU)
... And likely many more configurations

Reducing data to results (e.g. pixels or numbers) can be hard.



- Must to reduce data every step of the way.
 - ▣ Example: contour + normals + render
 - Important that you have less data in pixels than you had in cells. (*)
 - Could contouring and sending triangles be a better alternative?
 - ▣ Easier example: synthetic diagnostics

Outline



- The Terascale Strategy
- The I/O Wolf & Petascale Visualization
- An Overview of the Exascale Machine
- The Data Movement Wolf and Its 4 Angry Pups
 - ▣ Pup #2: Programming Languages
- Under-represented topics
- Conclusions

Angry Pup #2: Programming Language

- VTK: enables the community to develop diverse algorithms for diverse execution models for diverse data models
 - ▣ Important benefit: “write once, use many”
 - ▣ Substantial investment
- We need something like this for exascale.
 - ▣ Will also be a substantial investment
- Must be:
 - ▣ Lightweight
 - ▣ Efficient
 - ▣ Able to run in a many core environment

OK, what language is this in?
OpenCL? DSL?
... not even clear how to start

Message-passing remains important at the exascale, but we lose its universality

MPI will be combined with other paradigms within a shared memory node (OpenMP, OpenCL, CUDA, etc.)



Codes will not be hardware-universal again, until a lengthy evolutionary period passes

Outline



- The Terascale Strategy
- The I/O Wolf & Petascale Visualization
- An Overview of the Exascale Machine
- The Data Movement Wolf and Its 4 Angry Pups
 - ▣ Pup #3: Memory Footprint
- Under-represented topics
- Conclusions

Memory efficiency

- 64 PB of memory for 1 billion cores means 64MB per core
 - (May be 10 billion cores and 6.4MB per core)
- Memory will be the 2nd most precious resource on the machine.
 - There won't be a lot left over for visualization and analysis.
- Zero copy in situ is an obvious start
 - Templates? Virtual functions?
- Ensure fixed limits for memory footprints (Streaming?)

Outline



- The Terascale Strategy
- The I/O Wolf & Petascale Visualization
- An Overview of the Exascale Machine
- The Data Movement Wolf and Its 4 Angry Pups
 - ▣ Pup #4: In Situ-Fueled Exploration
- Under-represented topics
- Conclusions

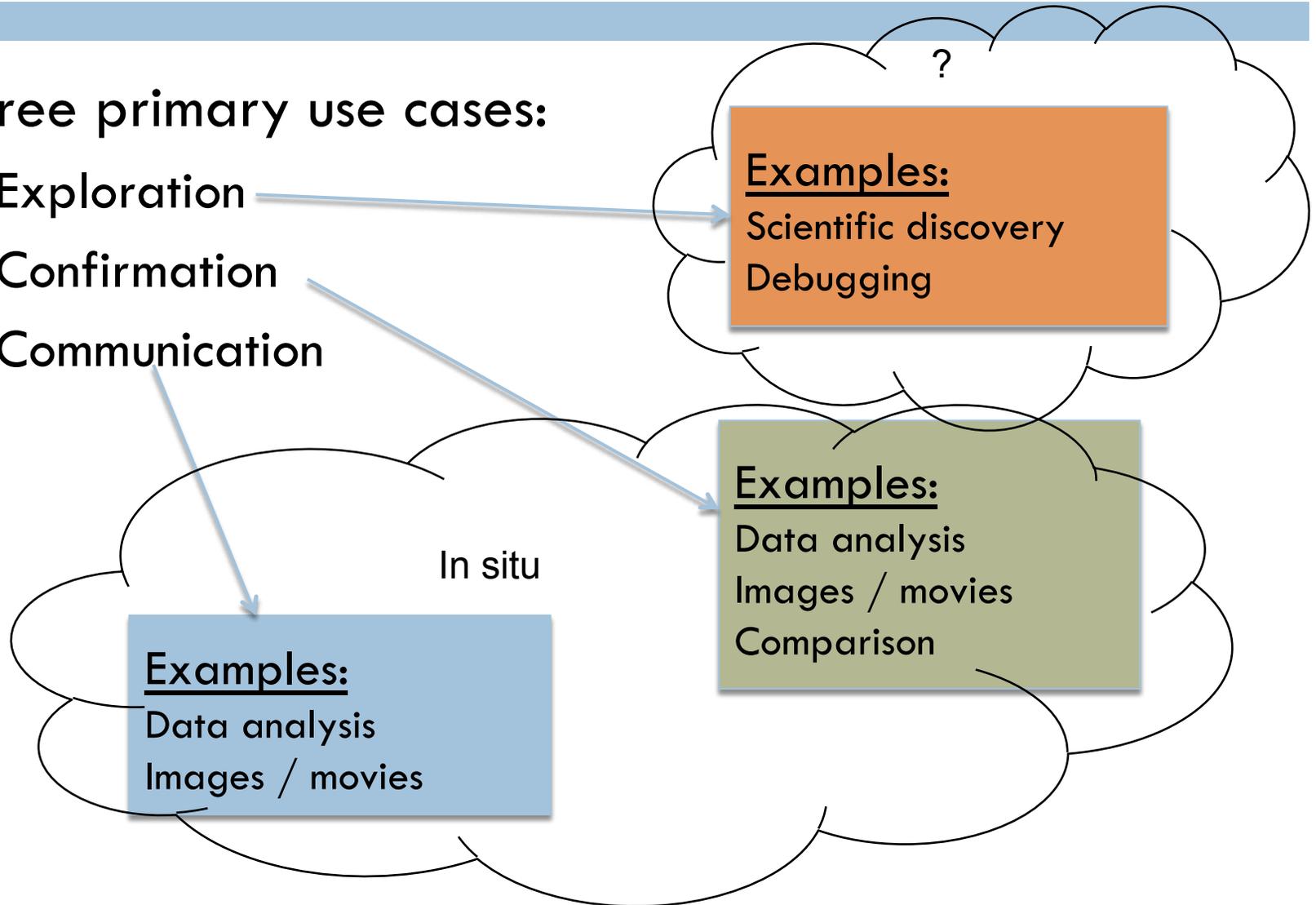
Do we have our use cases covered?

- Three primary use cases:

- Exploration

- Confirmation

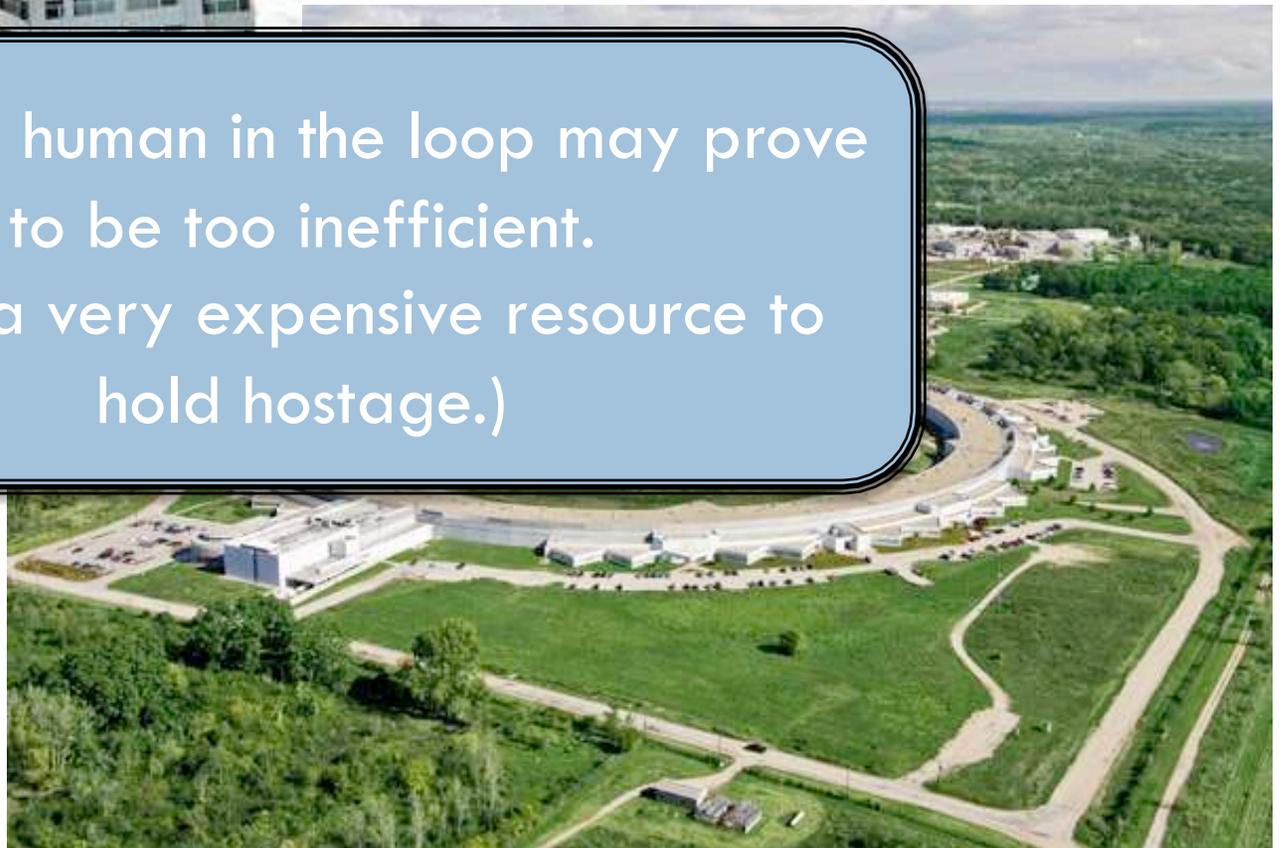
- Communication



Can we do exploration in situ?



Having a human in the loop may prove to be too inefficient.
(This is a very expensive resource to hold hostage.)



Enabling exploration via in situ processing

- Requirement: must transform the data in a way that both reduces and enables meaningful exploration.
- Subsetting
 - ▣ Exemplar subsetting approach: query-driven visualization
 - User applies repeated queries to better understand data
 - New model: produce set of subsets in situ, explore it with postprocessing
- Multi-resolution
 - ▣ Old model: user looks at coarse data, but can dive down to original data.
 - ▣ New model: branches of the multi-res tree are pruned if they are very similar. (compression!)

Enabling exploration via in situ processing

- Requirement: must transform the data in a way that both reduces and enables meaningful exploration.
- Subsetting
 - It is not clear what the best way is to use in situ processing to enable exploration with post-processing ... it is only clear that we need to do it.
- Multi-resolution
 - Old model: user looks at coarse data, but can dive down to original data.
 - New model: branches of the multi-res tree are pruned if they are very similar. (compression!)

Outline



- The Terascale Strategy
- The I/O Wolf & Petascale Visualization
- An Overview of the Exascale Machine
- The Data Movement Wolf and Its 4 Angry Pups
- **Under-represented topics**
- Conclusions

Under-represented topics in this talk.



- Two topics we will discuss later:
 - ▣ We will have quintillions of data points ... how do we meaningfully represent that with millions of pixels?
 - ▣ Data is going to be different at the exascale: ensembles, multi-physics, etc.
 - The outputs of visualization software will be different.
- Accelerators on exascale machine are likely not to have cache coherency
 - ▣ How well do our algorithms work in a GPU-type setting?
 - ▣ We have a huge investment in CPU-SW. What now?
- What do we have to do to support resiliency issue?

Outline



- The Terascale Strategy
- The I/O Wolf & Petascale Visualization
- An Overview of the Exascale Machine
- The Data Movement Wolf and Its 4 Angry Pups
- Under-represented topics
- **Conclusions**

It is funny how this happens...

- All petascale processing techniques still are very relevant at the exascale.
 - In situ: data movement wolf
 - Out-of-core: Pup #3: memory efficiency
 - Multi-res: Pup #4: exploration
 - Data subsetting: Pup #4: exploration
 - Pure parallelism: experiences at massive concurrency will be critical

Exascale Summary



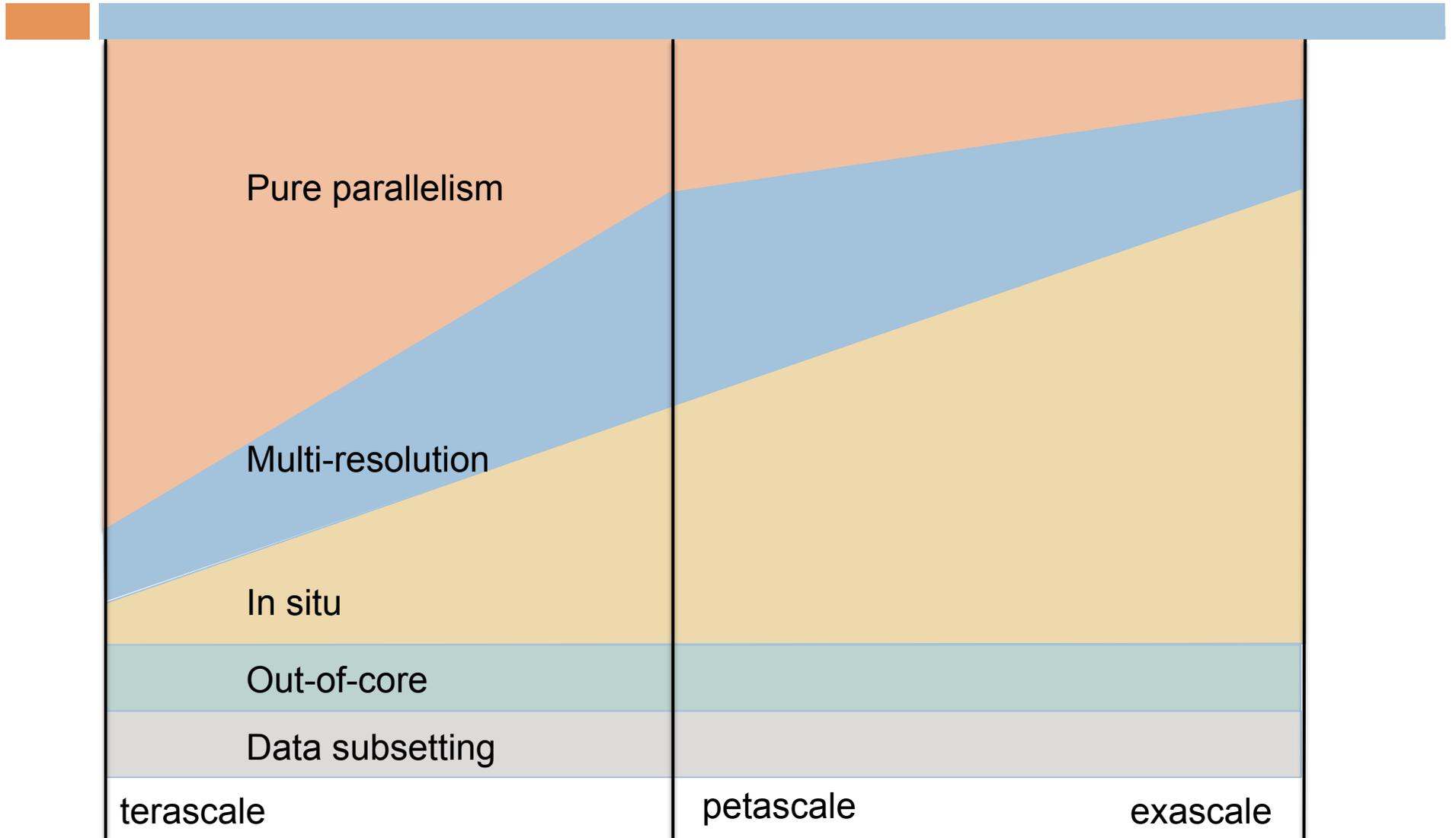
- We are unusual: we are data consumers, not data producers, and the exascale machine is being designed for data producers
- So the exascale machine will almost certainly lead to a paradigm shift in the way visualization programs process data.
 - ▣ Where to process data and what data to move will be a central issue.

Exascale Summary



- In addition to the I/O “wolf”, we will now have to deal with a data movement “wolf”, plus its 4 pups:
 - 1) In Situ System
 - 2) Programming Language
 - 3) Memory Efficiency
 - 4) In Situ-Fueled Exploration

Three Strategies for Three Epochs

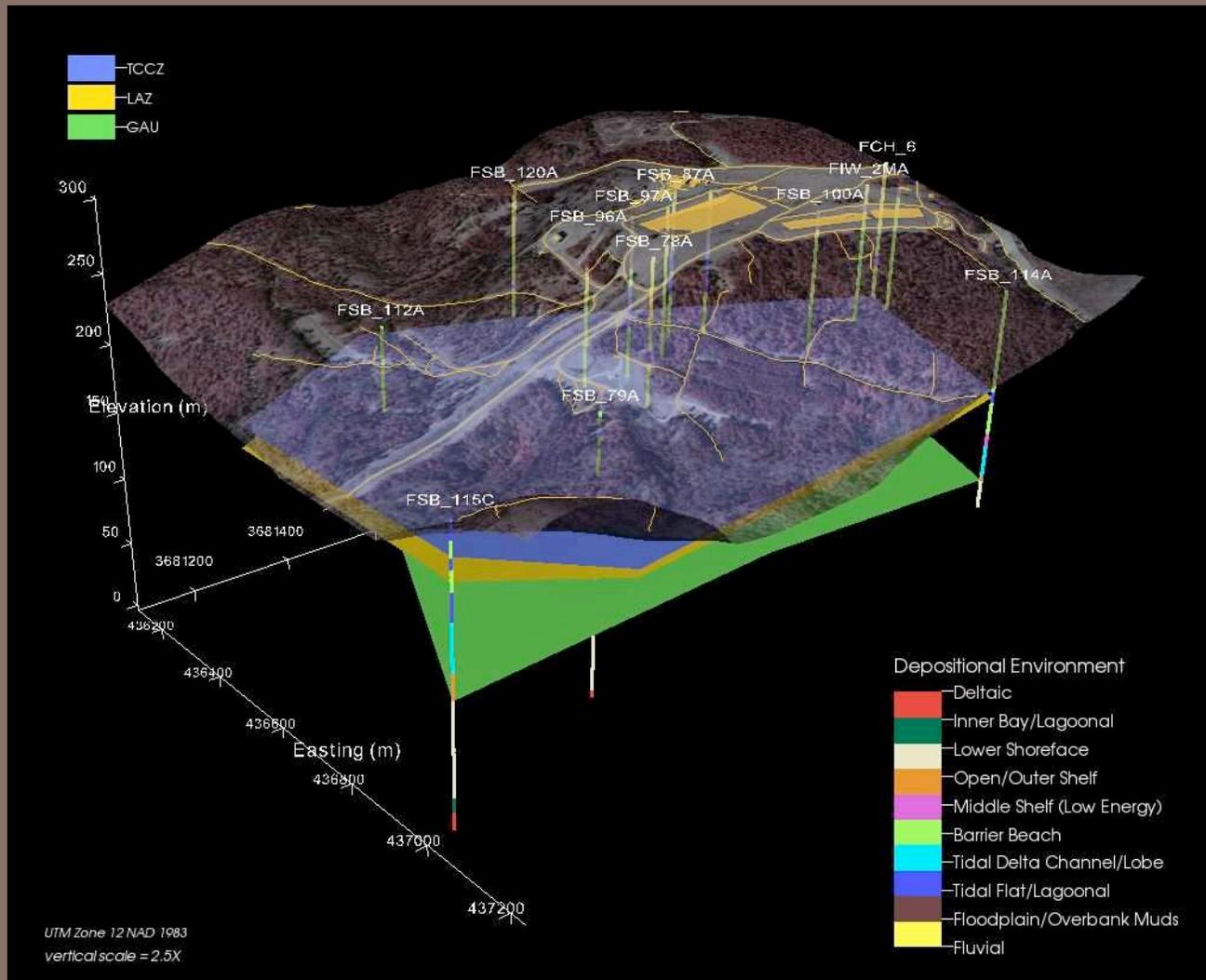


Summary



- There are three distinct strategies for the terascale, petascale, and exascale supercomputing
 - Visualization researchers are interested in different processing techniques depending on what scale of supercomputing they are planning for.
- Lecture #2: can we write visualization software that will work in lots of different processing environments?

Data Flow Networks



June 13, 2011

Hank Childs, Lawrence Berkeley Lab & UC Davis

Outline



- Data flow networks overview
- Data flow networks implementations
 - VTK
 - OpenDX

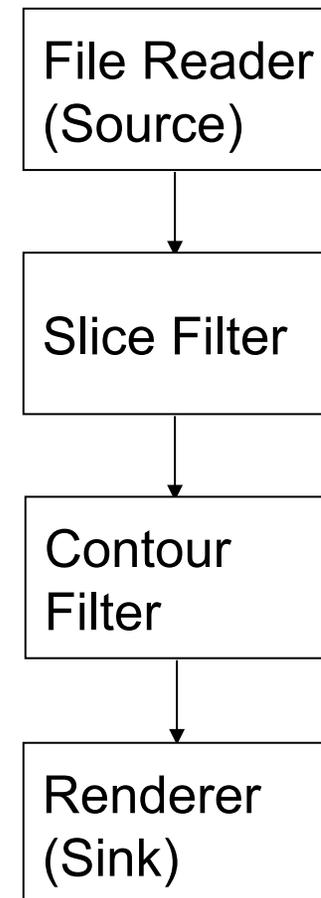
Outline



- Data flow networks overview
- Data flow networks implementations
 - VTK
 - OpenDX

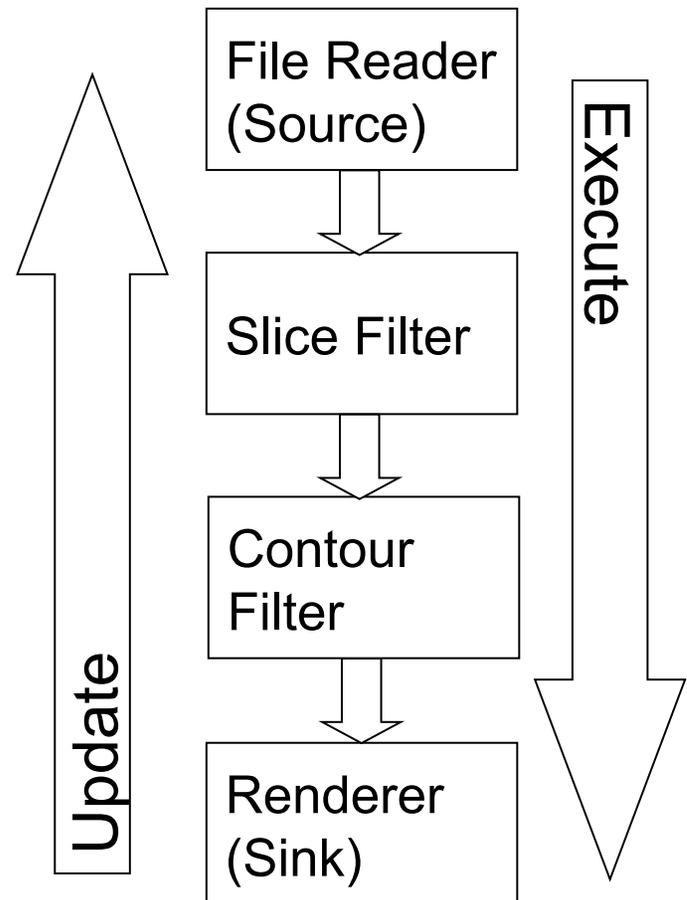
Data flow networks 101

- Work is performed by a pipeline
- A pipeline consists of data objects and components (sources, filters, and sinks)
 - ▣ Data objects: contain data
 - Typically “problem-sized” data.
 - ▣ Sources: source of data
 - Examples: file readers, geometry generators
 - ▣ Sinks: sinks for data
 - Examples: file writers, rendering modules
 - ▣ Filters: both sink and source
 - Purpose: manipulate input data object to create new output



Data flow networks 101

- Work is performed by a pipeline
- A pipeline consists of data objects and components (sources, filters, and sinks)
- Pipeline execution begins with a “pull”, which starts Update phase
- Data flows from component to component during the Execute phase



Data flow networks: pluses & minuses



□ Pluses

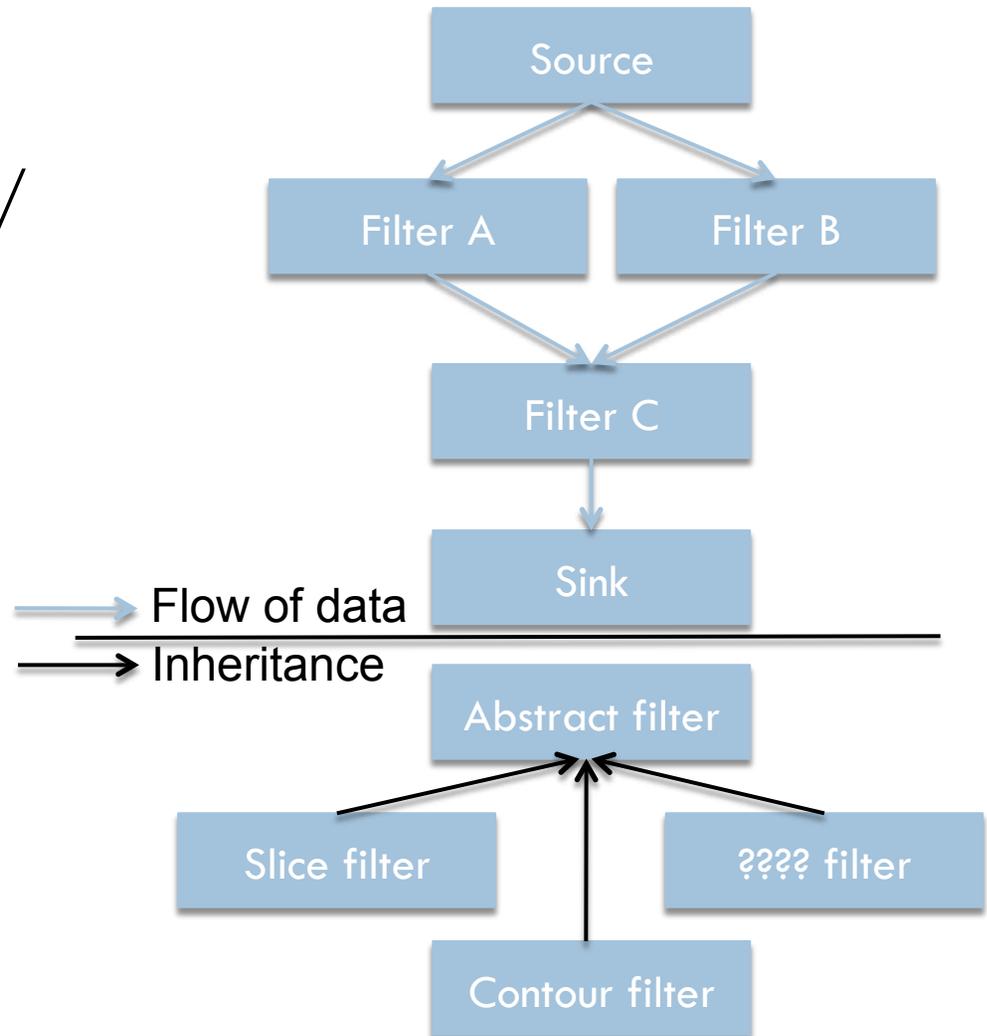
- Interoperability / Flexibility
- Extensible

□ Minuses

- Memory efficiency
- Performance efficiency
- Easy to add new algorithms, but hard to extend the data model

Data flow networks: strengths

- Flexible usage
 - ▣ Networks can be multi-input / multi-output
- Interoperability of modules
- Embarrassingly parallel algorithms handled by base infrastructure
- Easy to extend
 - ▣ New derived types of filters



Data flow networks: weaknesses



- Execution of modules happens in stages
 - Algorithms are executed at one time
 - Cache inefficient
 - Memory footprint concerns
- Some implementations fix the data model.

Outline



- Data flow networks overview
- Data flow networks implementations
 - VTK
 - OpenDX

Visualization with VTK



Content from: Erik Vidholm, Univ of Uppsala, Sweden
David Gobbi, Robarts Research Institute, London, Ontario, Canada

VTK – The Visualization ToolKit



- Open source, freely available software for 3D computer graphics, image processing, and visualization
- Managed by Kitware Inc.
- Use C++, Tcl/Tk, Python, Java

True visualization system



- Visualization techniques for visualizing
 - ▣ Scalar fields
 - ▣ Vector fields
 - ▣ Tensor fields
- Polygon reduction
- Mesh smoothing
- Image processing
- Your own algorithms

Additional features



- Parallel support (message passing, multithreading)
- Stereo support
- Integrates easily with Motif, Qt, Tcl/Tk, Python/Tk, X11, Windows, ...
- Event handling
- 3D widgets

3D graphics



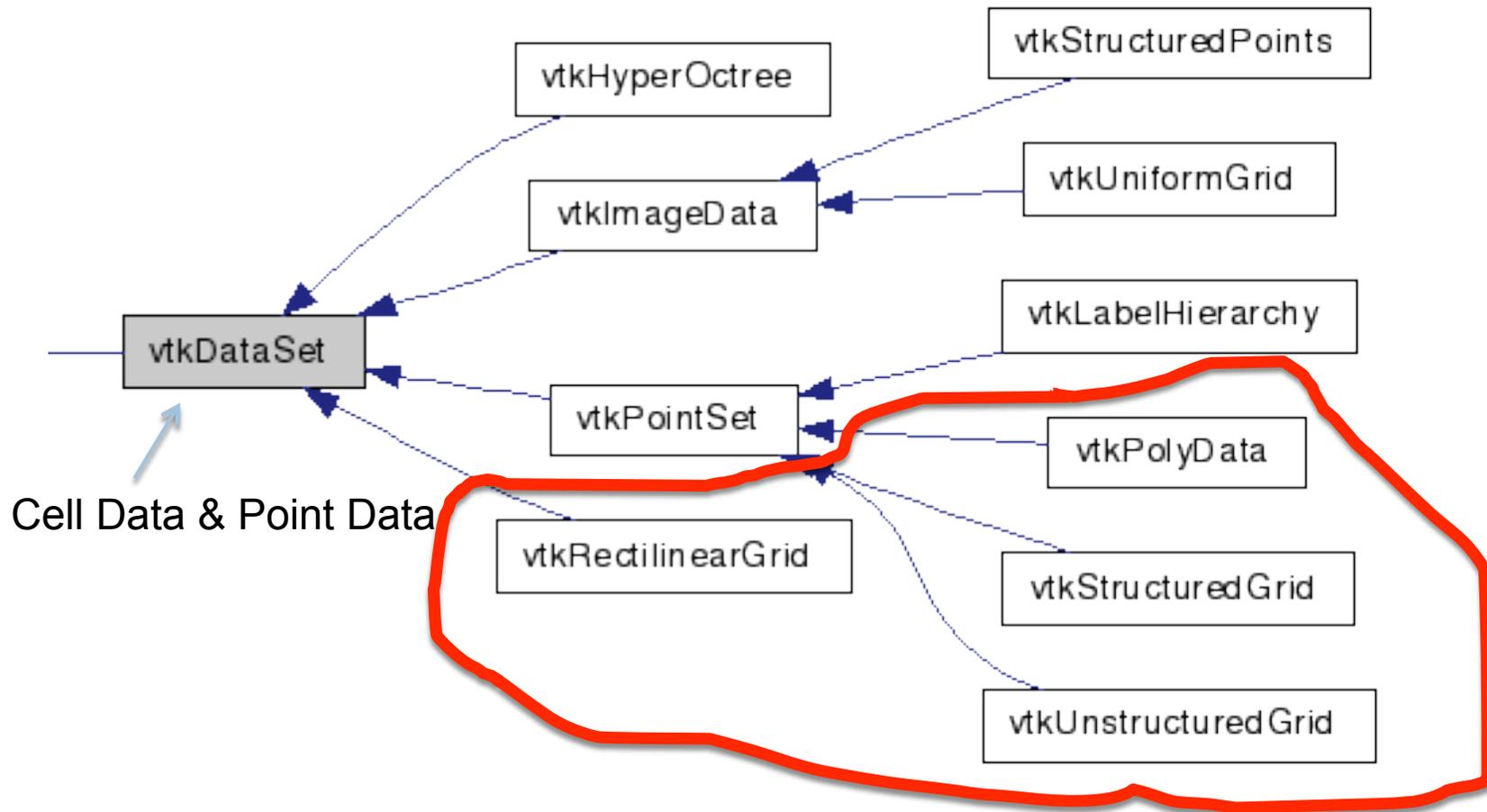
- Surface rendering
- Volume rendering
 - ▣ Ray casting
 - ▣ Texture mapping (2D)
 - ▣ Volume pro support
- Lights and cameras
- Textures
- Save render window to .png, .jpg, ...
(useful for movie creation)

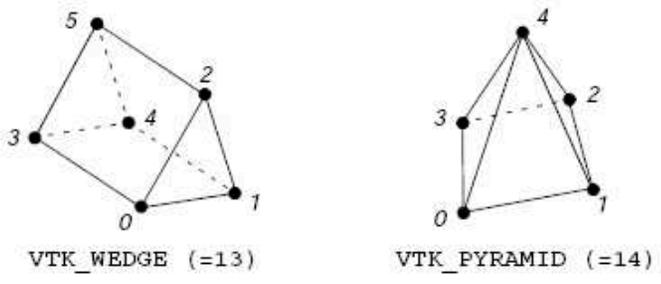
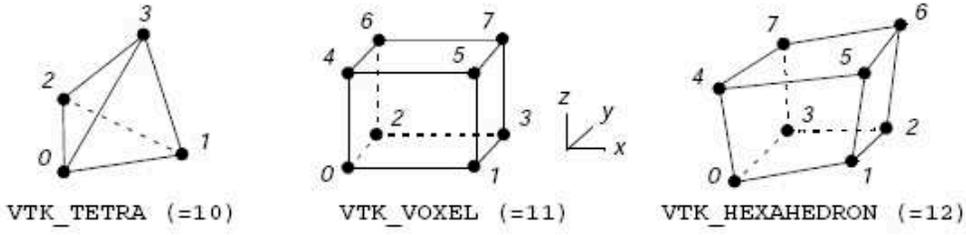
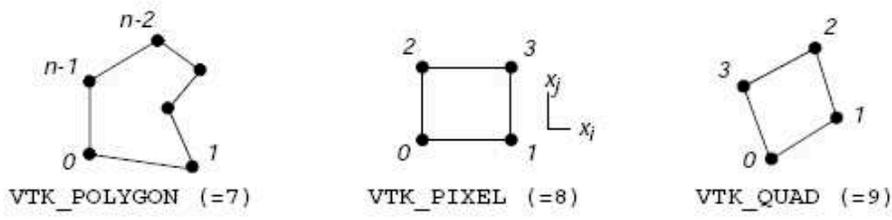
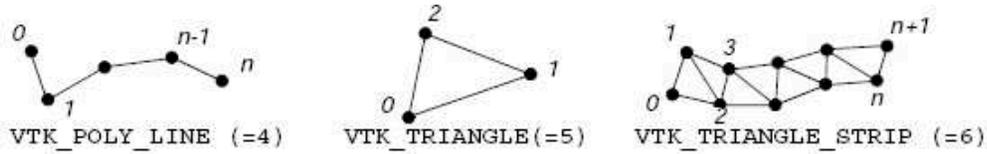
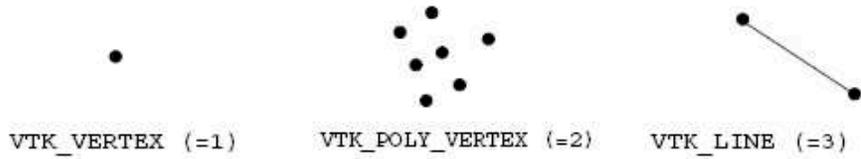
Objects

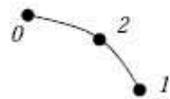


- Data objects
 - Next slide
- Process objects
 - Source objects (vtkReader, vtkSphereSource)
 - Filter objects (vtkContourFilter)
 - Mapper objects (vtkPolyDataMapper)

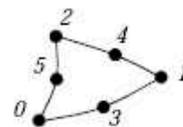
Data model



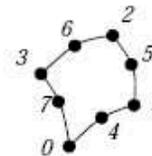




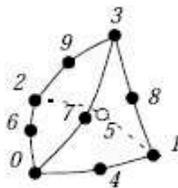
VTK_QUADRATIC_EDGE
(=21)



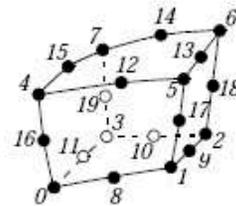
VTK_QUADRATIC_TRIANGLE
(=22)



VTK_QUADRATIC_QUAD
(=23)



VTK_QUADRATIC_TETRA
(=24)



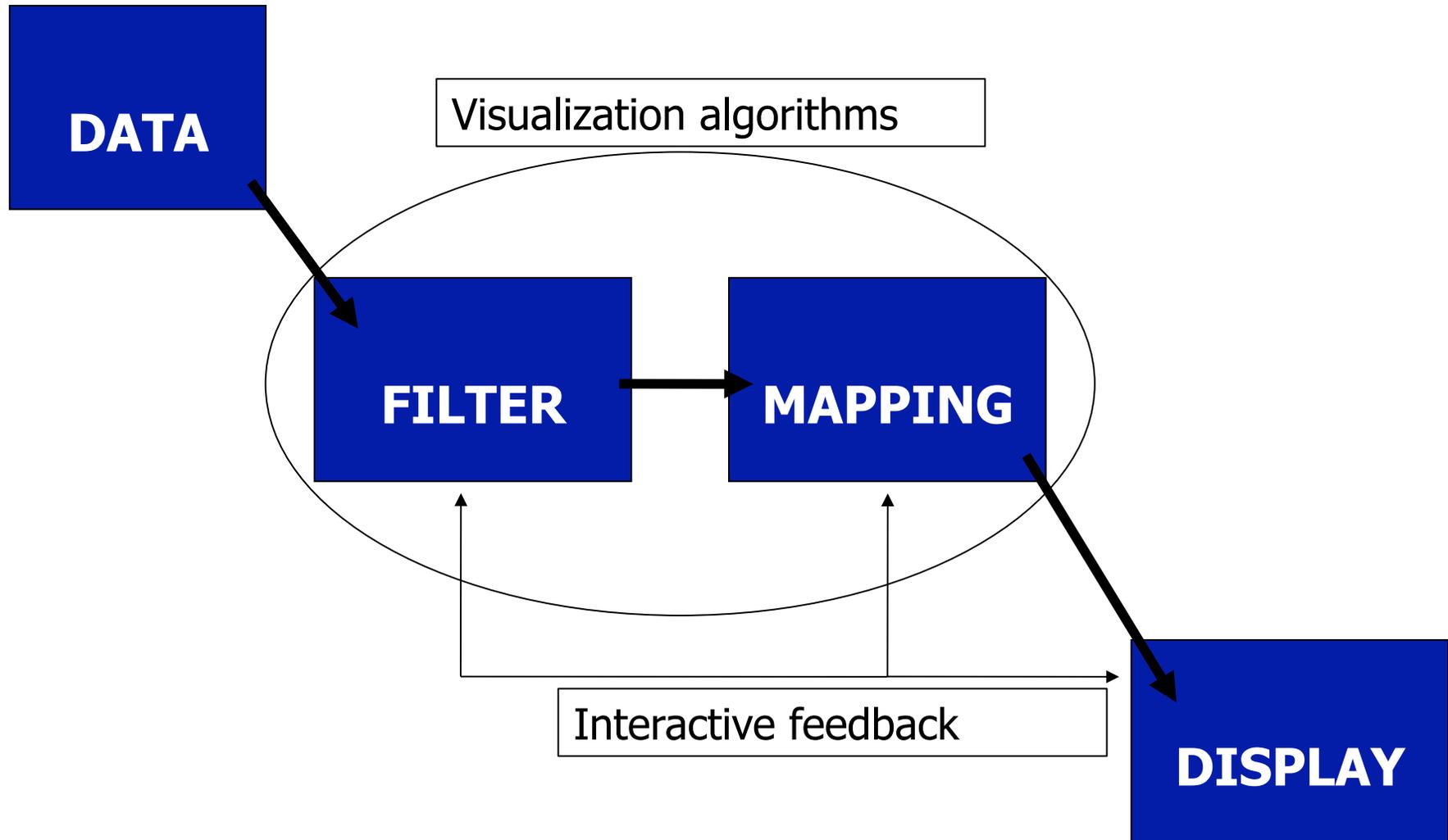
VTK_QUADRATIC_HEXAHEDRON
(=25)

Visualization continued

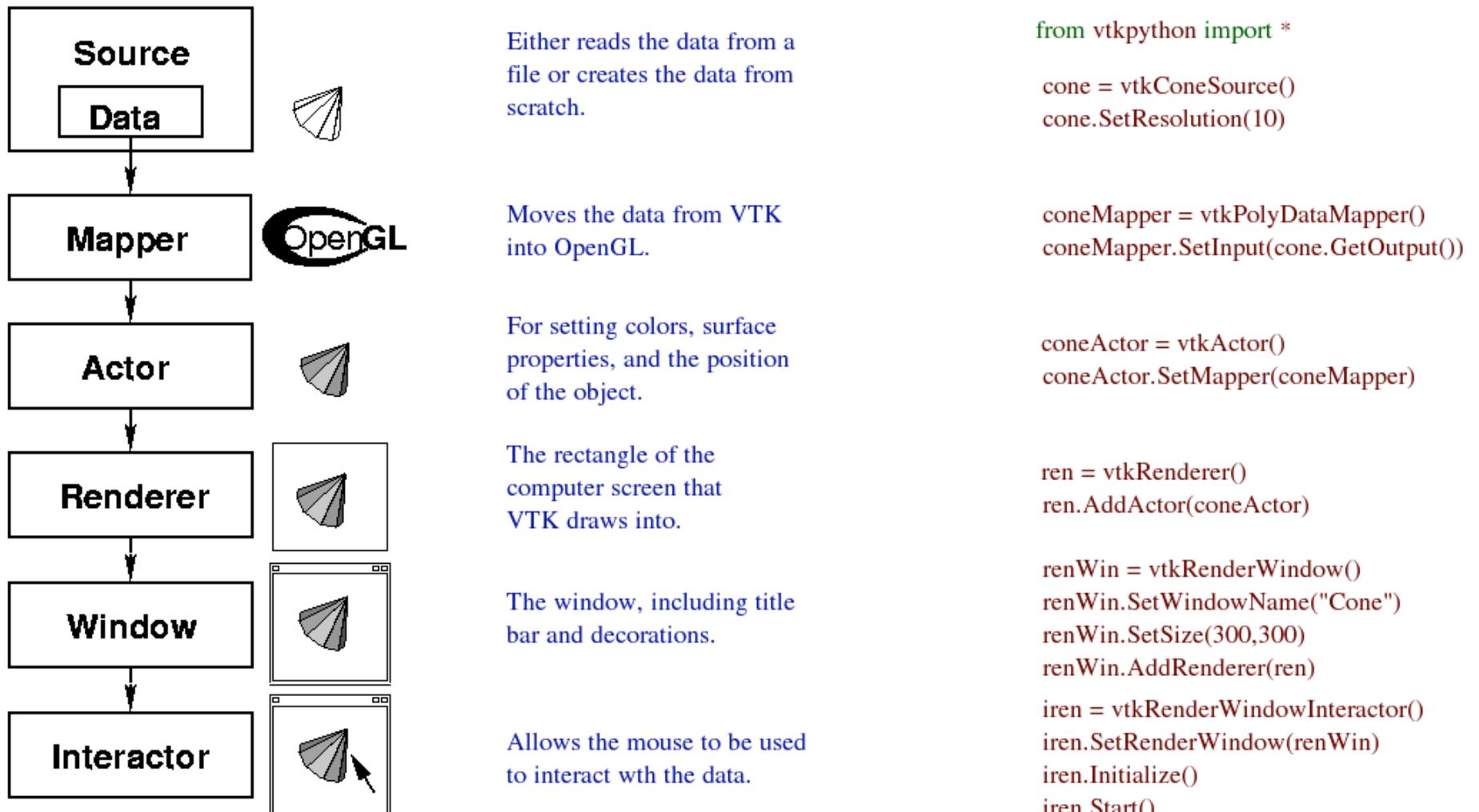


- Scalar algorithms
 - Iso-contouring
 - Color mapping
- Vector algorithms
 - Hedgehogs
 - Streamlines / streamtubes
- Tensor algorithms
 - Tensor ellipsoids

The visualization pipeline



Cone.py Pipeline Diagram (type "python Cone.py" to run)



Imaging

- Supports streaming => huge datasets
- `vtkImageToImageFilter`
 - Diffusion
 - High-pass / Low-pass (Fourier)
 - Convolution
 - Gradient (magnitude)
 - Distance map
 - Morphology
 - Skeletons

Summary +

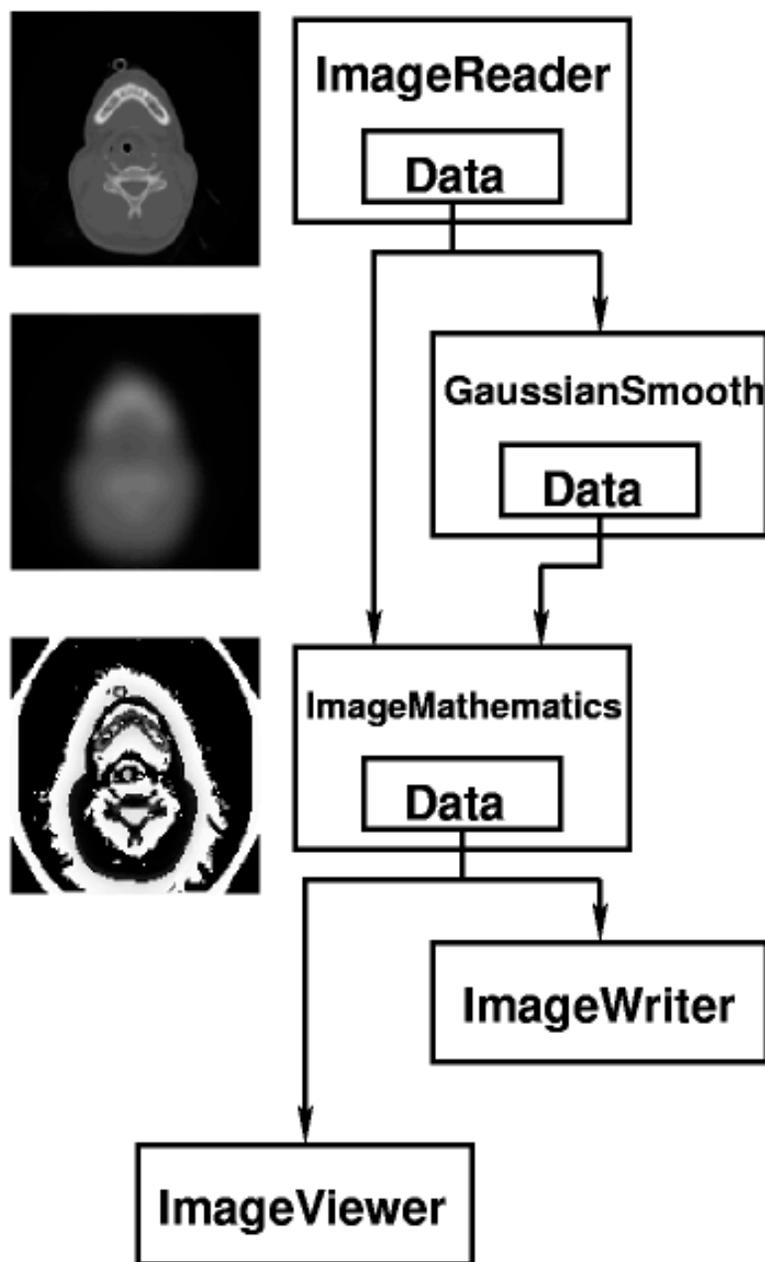


- Free and open source
- Create graphics/visualization applications fairly fast
- Object oriented - easy to derive new classes
- Build applications using "interpretive" languages Tcl, Python, and Java
- Many (state of the art) algorithms
- Heavily tested in real-world applications
- Large user base provides decent support
- Commercial support and consulting available

Summary -



- Not a super-fast graphics engine due to portability and C++ dynamic binding – you need a decent workstation
- Very large class hierarchy => learning threshold might be steep
- Many subtleties in usage
 - ▣ Pipeline execution model
 - ▣ Memory management



```

reader = vtkBMPReader()
reader.SetFileName("image.bmp")

```

```

blur = vtkImageGuassianSmooth()
blur.SetInput(reader.GetOutput())
blur.SetDimensionality(2)
blur.SetStandardDeviations(5.0, 5.0)
blur.SetRadiusFactors(10.0, 10.0)

```

```

subtract = vtkImageMathematics()
subtract.SetOperationToSubtract()
subtract.SetInput1(reader.GetOutput())
subtract.SetInput2(blur.GetOutput())

```

```

writer = vtkBMPWriter()
writer.SetInput(subtract.GetOutput())
writer.SetFileName("image2.bmp")
writer.Write()

```

```

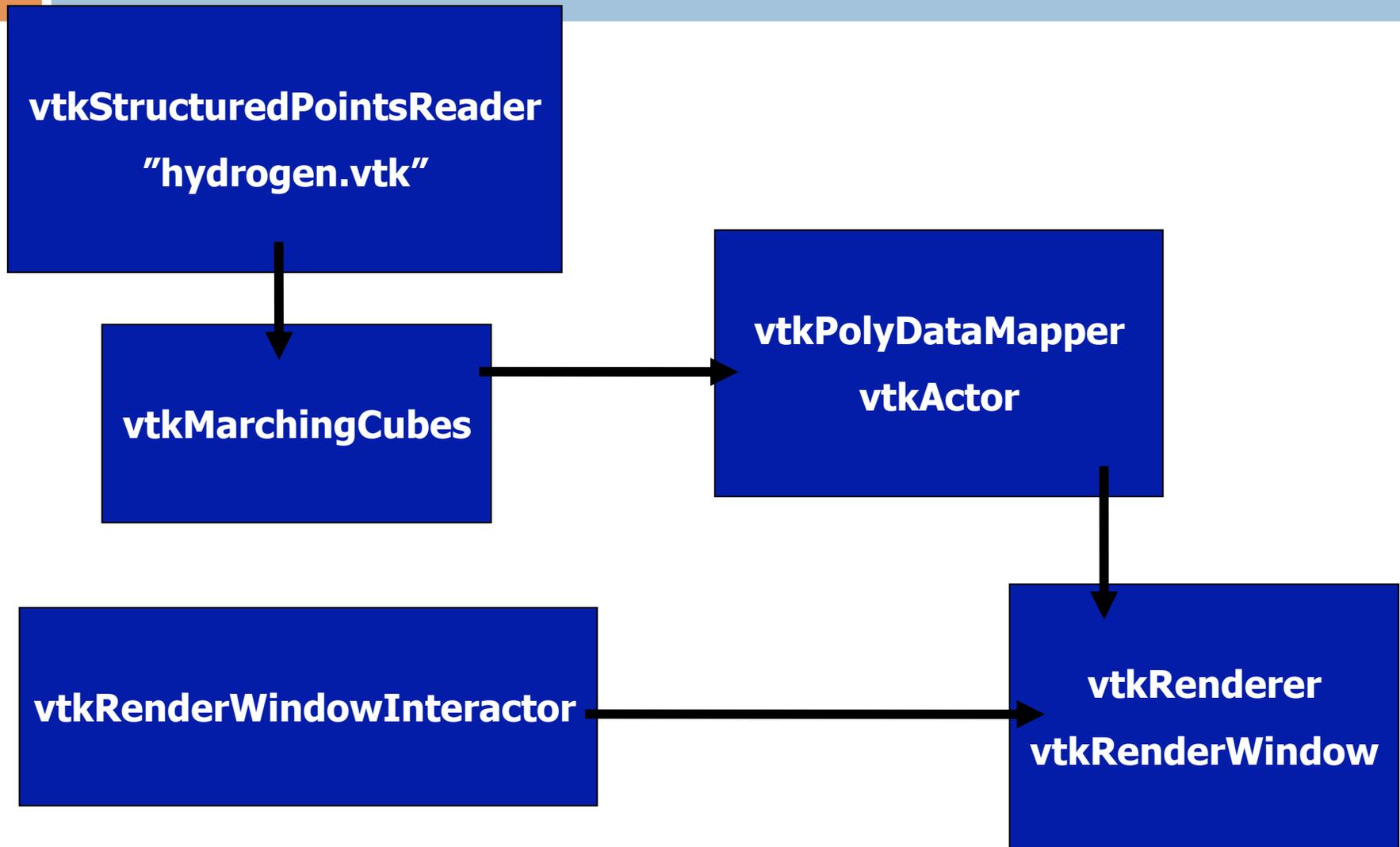
viewer = vtkImageViewer()
viewer.SetInput(subtract.GetOutput())
viewer.SetColorWindow(255)
viewer.SetColorLevel(127.5)
viewer.Render()

```

Example – Vector field visualization

```
vtkStructuredGridReader reader  
reader SetFileName "office.binary.vtk"  
  
# Create source for streamtubes  
vtkPointSource seeds  
seeds SetRadius 0.15  
eval seeds SetCenter 0.1 2.1 0.5  
seeds SetNumberOfPoints 6  
vtkRungeKutta4 integ  
vtkStreamLine streamer  
streamer SetInput [reader GetOutput]  
streamer SetSource [seeds GetOutput]  
streamer SetMaximumPropagationTime 500  
streamer SetStepLength 0.5  
streamer SetIntegrationStepLength 0.05  
streamer SetIntegrationDirectionToIntegrateBothDirections  
streamer SetIntegrator integ  
  
...
```

The visualization pipeline - example



Python example: visualization hydrogen molecule

**Must call
update to
read!**

**Pipeline
connections**

```
# File: isosurface.py
import vtk

# image reader
reader = vtk.vtkStructuredPointsReader()
reader.SetFileName("hydrogen.vtk")
reader.Update()

# bounding box
outline = vtk.vtkOutlineFilter()
outline.SetInput( reader.GetOutput() )
outlineMapper = vtk.vtkPolyDataMapper()
outlineMapper.SetInput( outline.GetOutput() )
outlineActor = vtk.vtkActor()
outlineActor.SetMapper( outlineMapper )
outlineActor.GetProperty().SetColor(0.0,0.0,1.0)
```

Example continued

vtkContourFilter
chooses the
appropriate
method for the
data set

```
# iso surface
isosurface = vtk.vtkContourFilter()
isosurface.SetInput( reader.GetOutput() )
isosurface.SetValue( 0, .2 )
isosurfaceMapper = vtk.vtkPolyDataMapper()
isosurfaceMapper.SetInput( isosurface.GetOutput() )
isosurfaceMapper.SetColorModeToMapScalars()
isosurfaceActor = vtk.vtkActor()
isosurfaceActor.SetMapper( isosurfaceMapper )

# slice plane
plane = vtk.vtkImageDataGeometryFilter()
plane.SetInput( reader.GetOutput() )
planeMapper = vtk.vtkPolyDataMapper()
planeMapper.SetInput( plane.GetOutput() )
planeActor = vtk.vtkActor()
planeActor.SetMapper( planeMapper )
```

Example continued

Creates a legend from the data and a lookup table

```
# a colorbar
scalarBar = vtk.vtkScalarBarActor()
scalarBar.SetTitle("Iso value")

# renderer and render window
ren = vtk.vtkRenderer()
ren.SetBackground(.8, .8, .8)
renWin = vtk.vtkRenderWindow()
renWin.SetSize( 400, 400 )
renWin.AddRenderer( ren )
```

Example continued

**The
RenderWindowInteractor
contains functions for
mouse/keyboard
interaction**

```
# render window interactor  
iren = vtk.vtkRenderWindowInteractor()  
iren.SetRenderWindow( renWin )
```

```
# add the actors  
ren.AddActor( outlineActor )  
ren.AddActor( isosurfaceActor )  
ren.AddActor( planeActor )  
ren.AddActor( scalarBar )
```

**The renWin.Render()
calls Update() on the
renderer, which calls
Update() for all its
actors, which calls...**

```
# this causes the pipeline to "execute"  
renWin.Render()
```

```
# initialize and start the interactor  
iren.Initialize()  
iren.Start()
```

The VTK file format

- Many modules to write VTK files

```
# vtk DataFile Version 2.0
Hydrogen orbital
ASCII
DATASET STRUCTURED_POINTS
DIMENSIONS 64 64 64
ORIGIN 32.5 32.5 32.5
SPACING 1.0 1.0 1.0
POINT_DATA 262144

SCALARS probability float
LOOKUP_TABLE default
0.0 0.0 0.01 0.01 .....
```

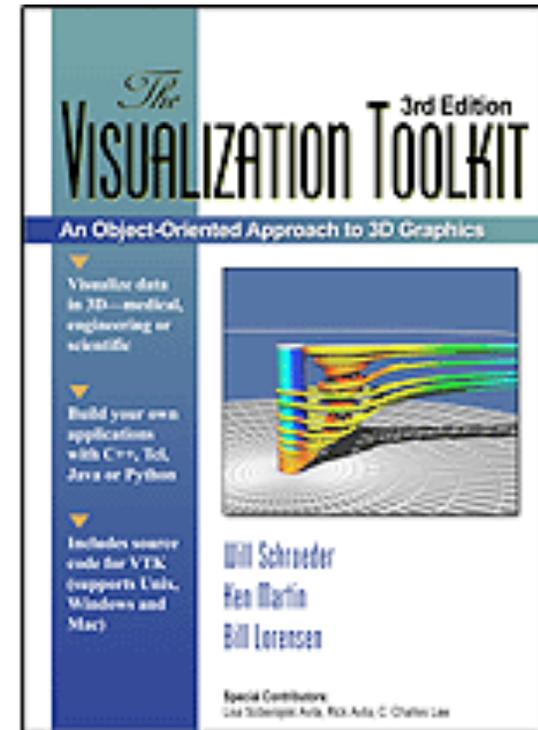
VTK and C++



- Build with CMake and your favorite compiler
- CMake generates makefiles or project files for your environment
- Use the resulting file(s) to build your executable
- With C++ you have full control and can derive own classes, but you need to write many lines of code...

VTK resources

- www.vtk.org
 - Download (source and binaries)
 - Documentation
 - Mailing lists
 - Links
 - FAQ, Search
- www.kitware.com
 - VTK Textbook
 - VTK User's guide
 - Mastering CMake



Outline



- Data flow networks overview
- Data flow networks implementations
 - VTK
 - **OpenDX** ← courtesy Greg Abram!!

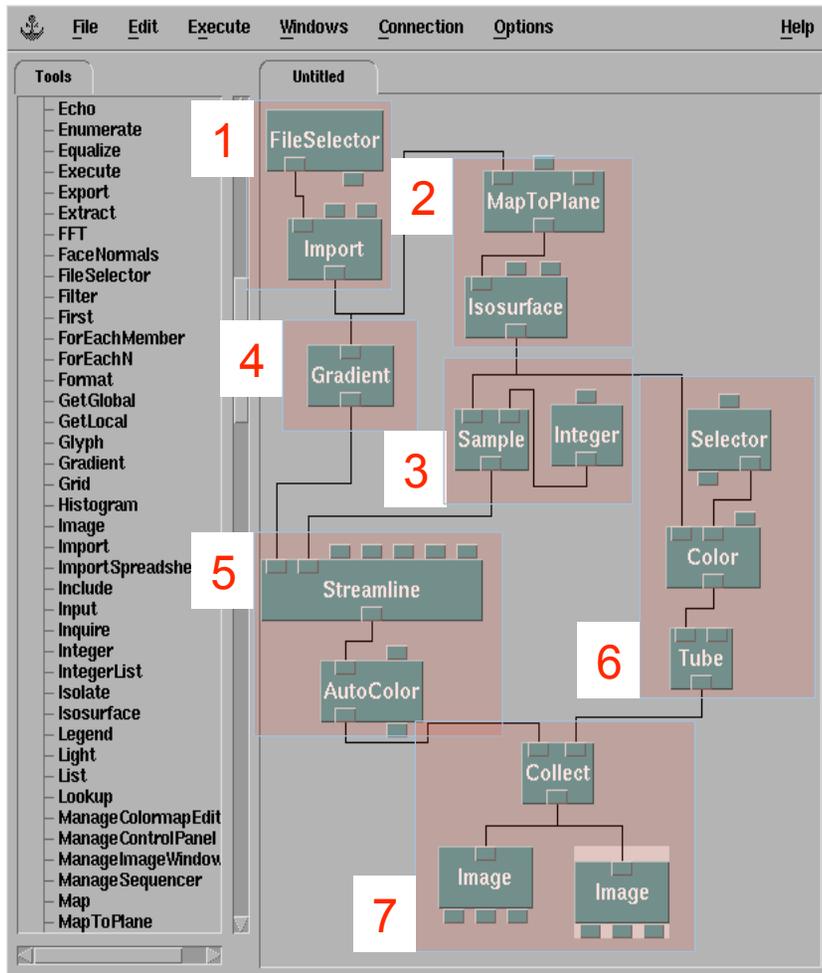
OpenDX History

- 1988 Work begins at IBM TJ Watson Research Center to develop a data-parallel “Visualization Supercomputer”
 - Goals:
 - Extreme performance
 - Accessibility to non-sophisticated users
 - Technology:
 - Hardware: 32-way cache-coherent SMP NUMA system based on Intel i860 processors – the IBM PVS
 - Software: Scientific visualization environment based on a data-flow-like execution model, a visual programming paradigm and a comprehensive data model
- 1991 IBM Visualization Data Explorer software released on wide range of workstation-level platforms including SMP support
- 1996 Open-sourced as OpenDX

Architecture

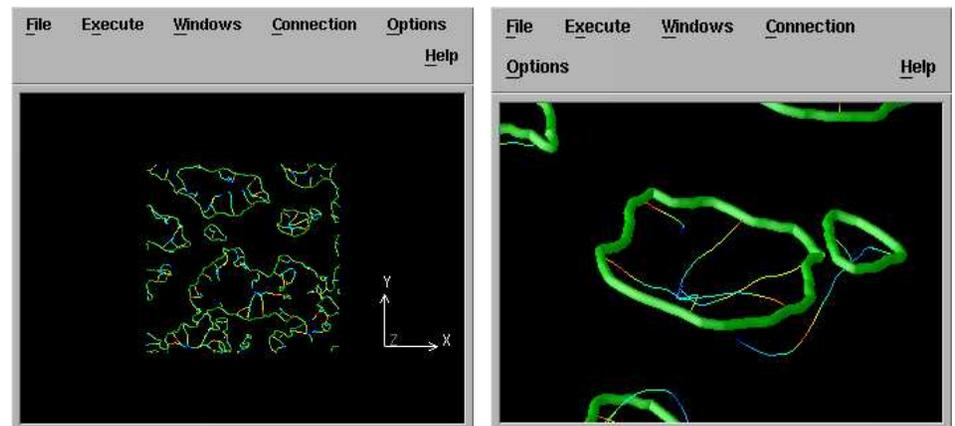
- Client / Server: GUI is client, Executive is server, communication via sockets and X protocol
- GUI:
 - ▣ Presents “data flow” programming environment to application programmer;
 - ▣ Presents application GUI (e.g. control panels, image windows) to application user;
 - ▣ Sends program “network” to Executive and receives images for display
- Executive:
 - ▣ Implements an object-oriented data model for the representation of data;
 - ▣ Manages cache of previously computed results;
 - ▣ Analyzes networks to determine what needs to be done for a given execution based on network connectivity and availability of prior results; and
 - ▣ Calls module code to perform required computation

General Directed Acyclic Graphs

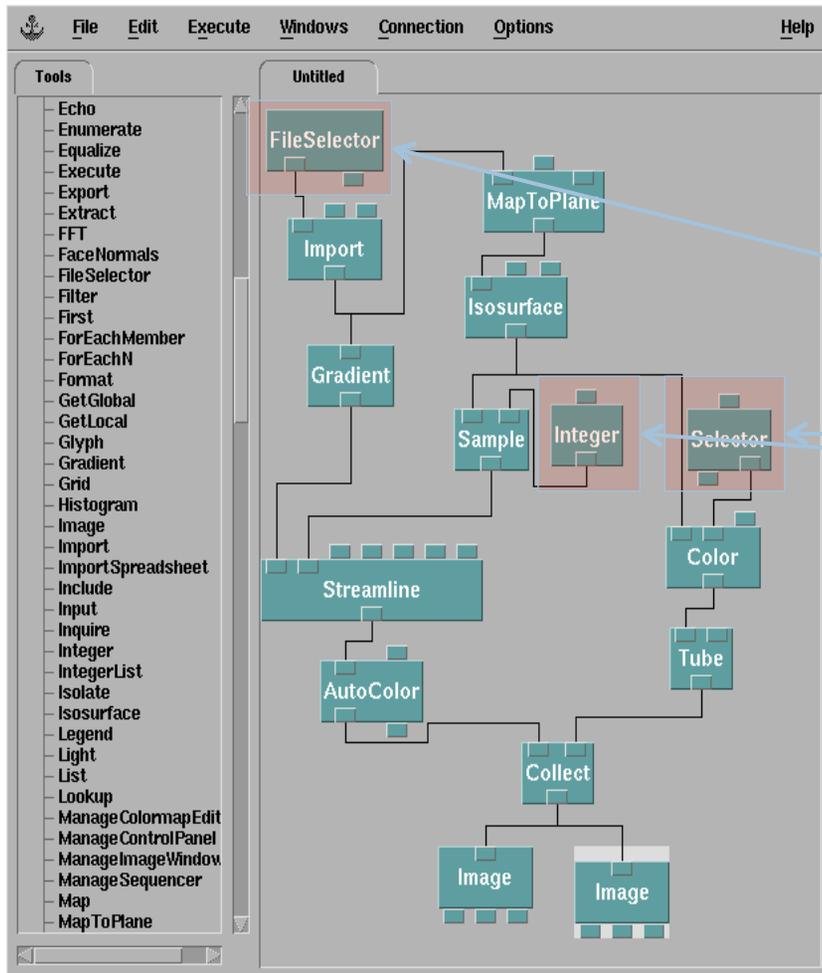


Enables user to compute multiple inputs to a module: here, the vector field and starting point set inputs of streamline module

1. Import data
2. compute the 0-contour of a scalar field in an arbitrary plane
3. Sample contours for streamline starting points
4. Compute gradient of scalar field
5. Compute streamlines in gradient field
6. Color and tube-ify contours
7. Show streamlines, contours from 2 different viewpoints



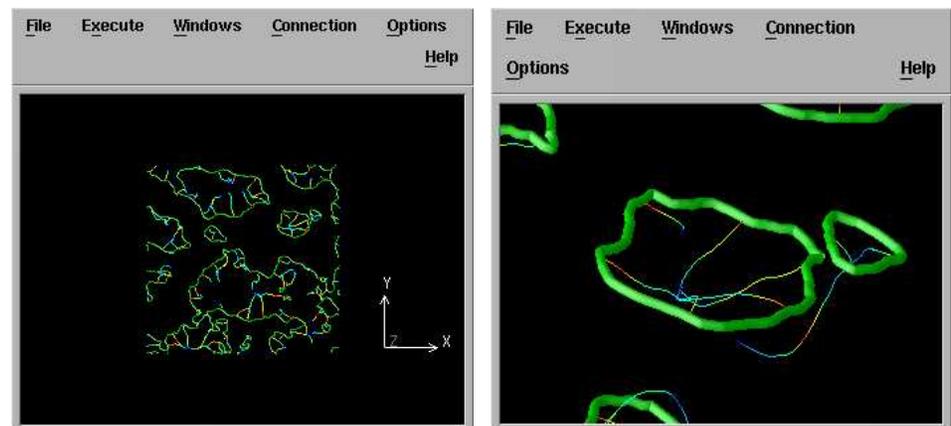
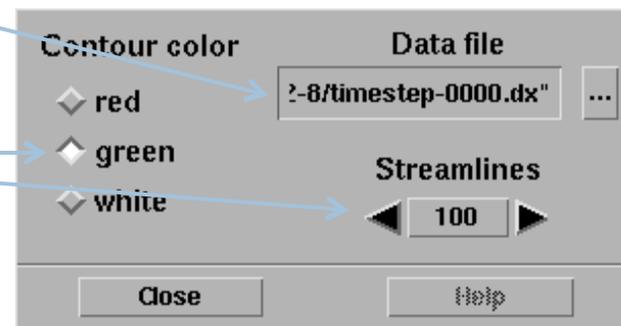
Programming- vs. Application-Interface



VPE Programming UI

- Interactors corresponding to network inputs are placed into control panels
- Users need never see visual programming environment

Application UI



Data Caching



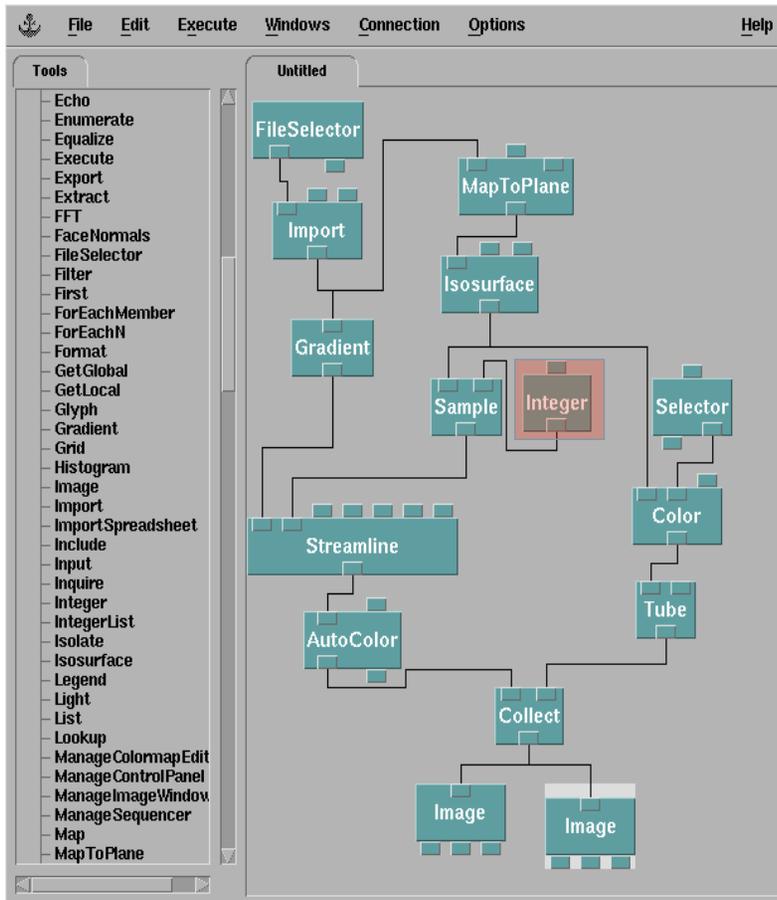
- Module results are cached for re-use in interactive and iterative executions of network
- Module are pure functions of their inputs
 - No side-effects
 - Modules run completely; no partial execution of modules based on downstream factors
- Executive maintains object cache:
 - Cache tags formed from producing module name and the cache tags of the producing module's inputs
 - Executive can determine whether a result is available without calling the module code

“Data Flow” By Network Analysis

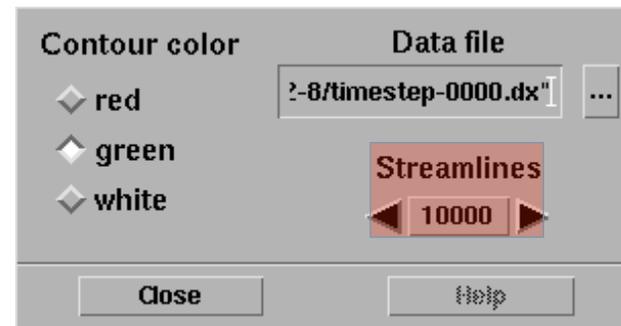


1. Identify data sinks – e.g. render windows, exporters
2. Traverse graph *upward* from sinks to identify connected modules
3. Traverse resulting subgraph *downward* from connected inputs to assign cache tags
4. Traverse *upward* to find sub-subgraph containing uncached results
5. Traverse sub-subgraph *downward* to perform required computations

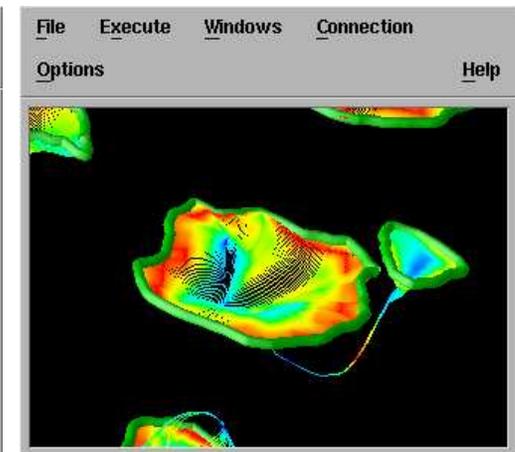
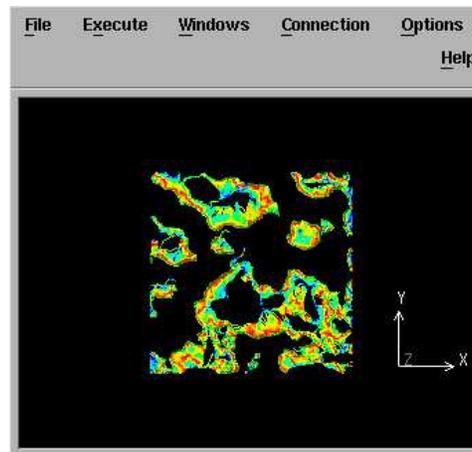
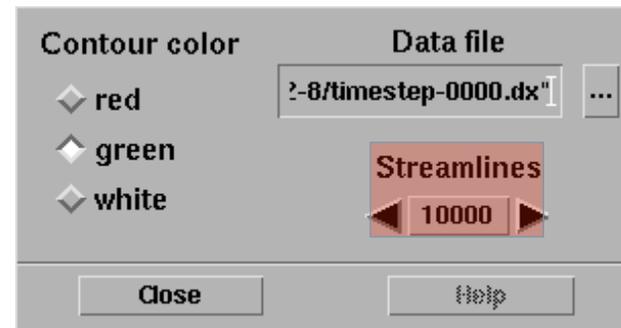
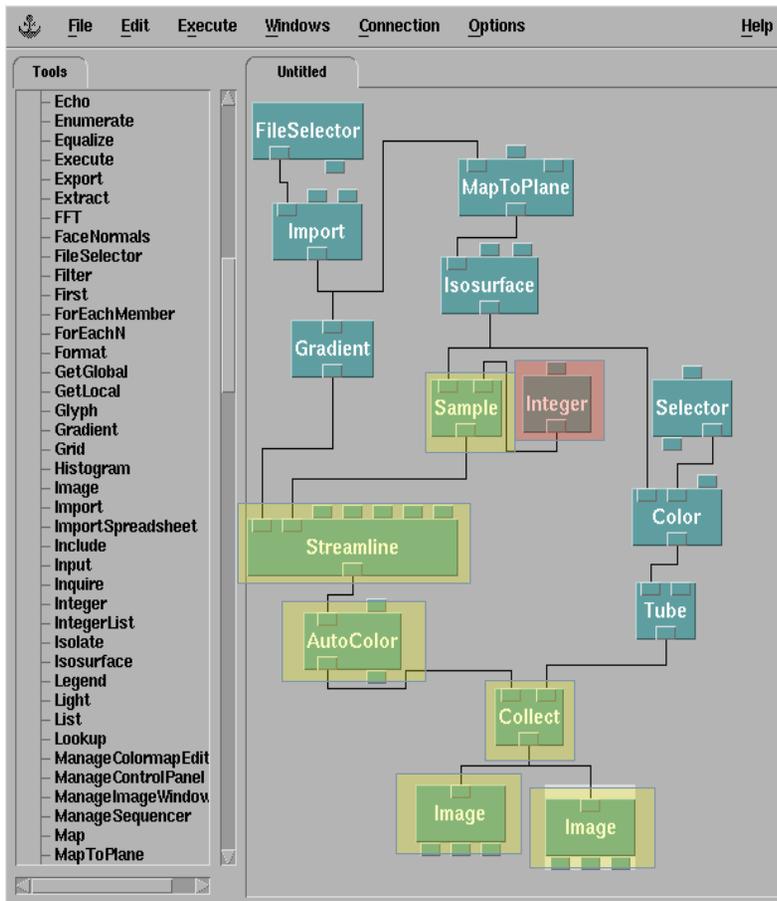
Interaction causes re-execution



- User decides streamlines are too sparse, uses the interactor in the control panel to increase the number of samples to be taken in the iso-contours
- Poor-man's stream-surfaces?



Executive determines dependent modules and runs them



OpenDX Summary



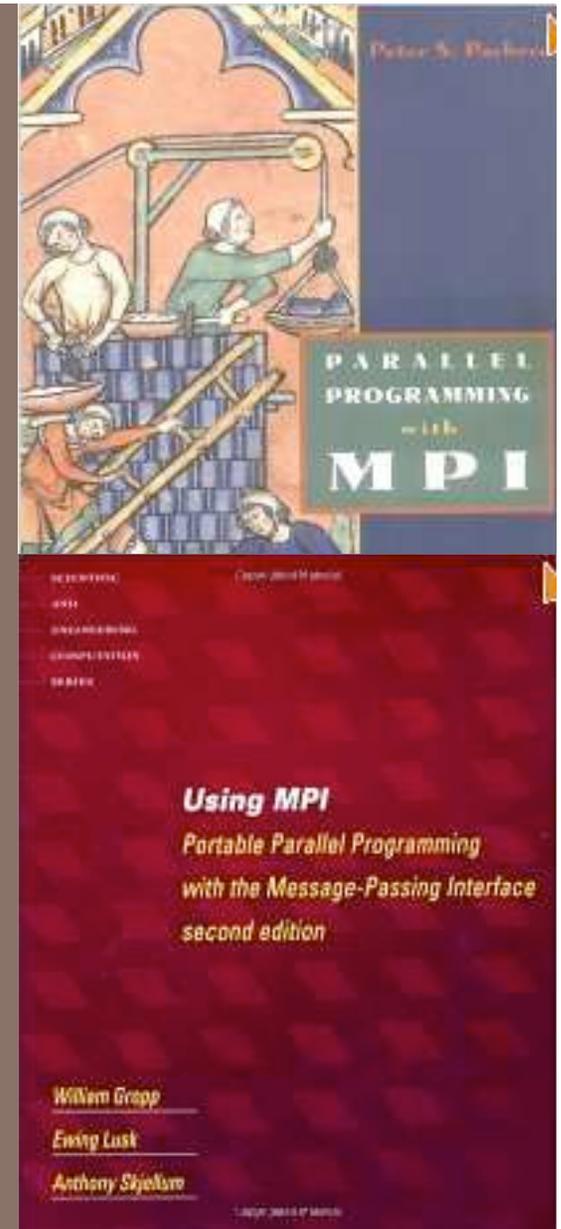
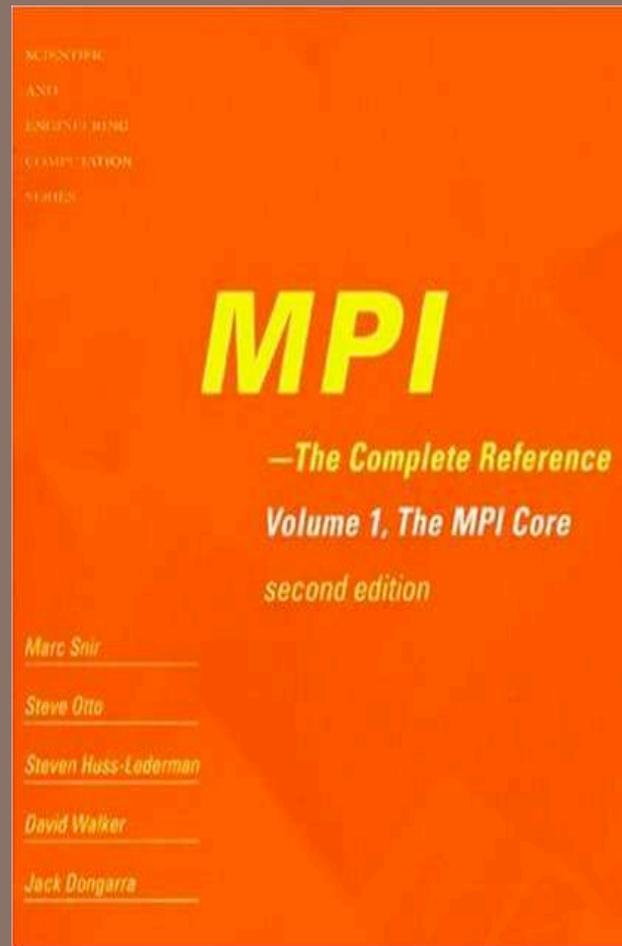
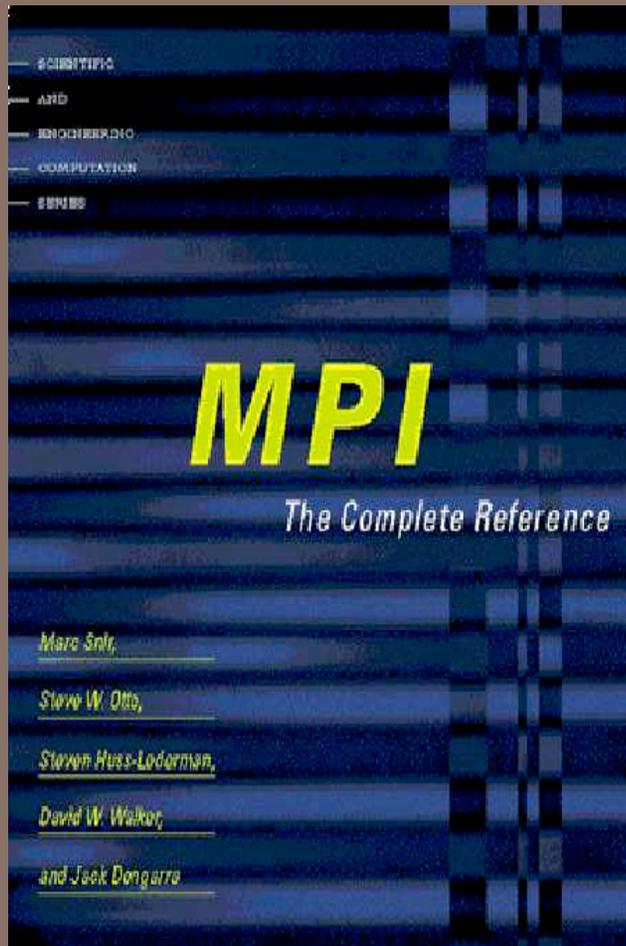
- Visualization programs are generalized DAG networks rather than pipelines
 - ▣ Aimed at developing “visualization applications”, rather than visualizations themselves
 - ▣ Some programs consisted of hundreds or thousands of modules
- Control flow is determined by program analysis rather than true push- or pull-model dataflow
- Extensions to execution model support conditional execution and looping

Data flow networks: summary



- Data flow networks is a design that:
 - allows for users to explore data in dynamic and unforeseen ways
 - is highly extensible
 - is very popular in visualization software
 - Tomorrow: can be used with many processing techniques

MPI Overview

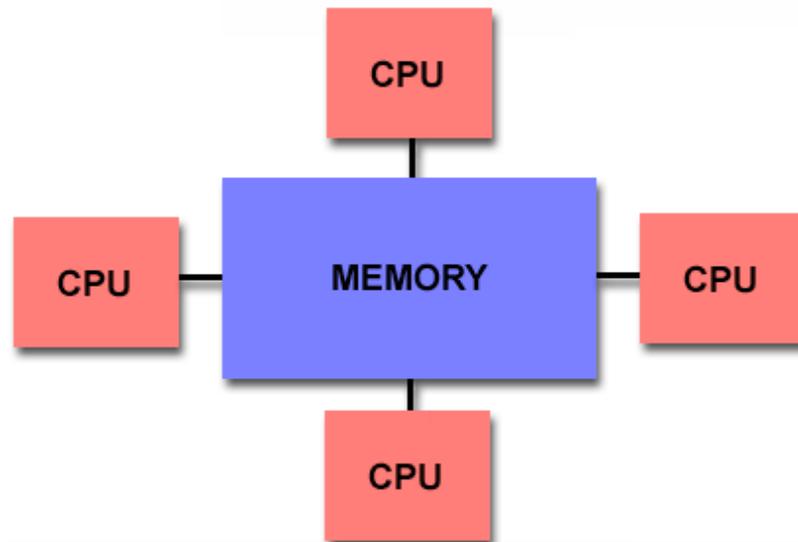


June 14, 2011

Hank Childs, Lawrence Berkeley Lab & UC Davis

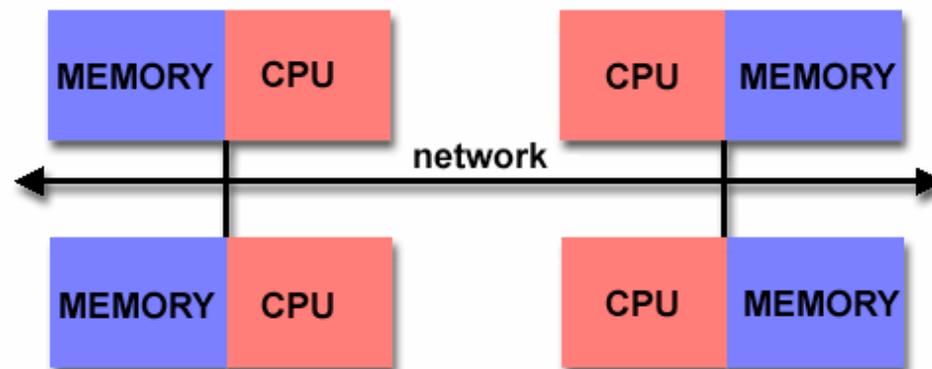
Parallel Computer Memory Architectures

- **Shared Memory**
 - **Uniform Memory Access (UMA)**
 - **Non-Uniform Memory Access (NUMA)**



Parallel Computer Memory Architectures ...contd.

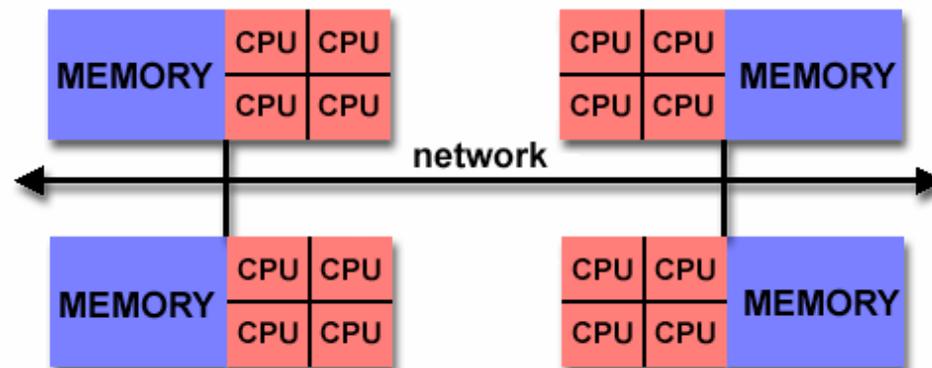
□ Distributed Memory



Parallel Computer Memory Architectures ...contd.

□ Hybrid Distributed-Shared Memory

- The largest and fastest computers in the world today employ both shared and distributed memory architectures.



Comparison of Shared and Distributed Memory Architectures

Architecture	UMA	NUMA	Distributed
Communications	MPI Threads OpenMP shmem	MPI Threads OpenMP Shmem	MPI (or hybrid)
Scalability	to 10s of processors	to 100s of processors	to 100000s of processors
Draw Backs	Memory-CPU bandwidth	Memory-CPU bandwidth Non-uniform access times	System administration Programming is hard to develop and maintain

What is MPI?



- MPI = message passing interface
- De facto standard for programming distributed memory machines
- API that enables coordination of parallel programs via message passing (send & receive)
- Enables portability. Programmers make MPI calls to coordinate processing, MPI translates to network calls.

Historical Development of MPI

- 1980-early 1990: distributed memory parallel computing application develops and calls for a standard
- 1992: MPI Forum established
- 1993: draft MPI standard presented at SC'93
- May, 1994: MPI-1 final version released, 115 routines defined
- 1996; MPI-2 finalized, which picked up “difficult” issues that MPI-1 intentionally left off.
- Most vendors have full implementation of MPI-1, but partial implementation of MPI-2

Using MPI

- Two ways to compile:
 - Manually tell compiler library location
 - `#include <mpi.h>`
 - `gcc -I/path/to/mpi/include file.c -L/path/to/mpi/lib -lmpi`
 - Special compiler that manages environment
 - `mpicc file.c`
- Special way to invoke MPI programs:
 - `mpirun -np 8 my_program`
 - `qsub / aprun`
 - Many more...

Sustainable High Performance Computing

HoS : MPI

Mauro Bianco
Texas A&M University

Thierry Carrard
CEA / DIF

Sustainable High Performance Computing *HoS* : MPI

Mauro Bianco
Texas A&M University

Thierry Carrard
CEA / DIF

Outline



- Opening questions / remarks
- Examples fast forward
- Examples fast forward
 - Hellow World
 - String Search
 - Pi computation
 - Matrix multiply
- Getting started
- Exercises
- Questions

Opening questions / remarks



- Questions about the morning course
- Remarks / special requests for this HoS

Examples fast forward



- Hello World
 - Initialization of the runtime
 - Ensure everyone is ready to compile and execute
- String search
 - Divide and conquer
- Pi computation
 - Scalable problem, no communication bottleneck
- Matrix multiply
 - A first practical numerical problem

Getting started: MPI

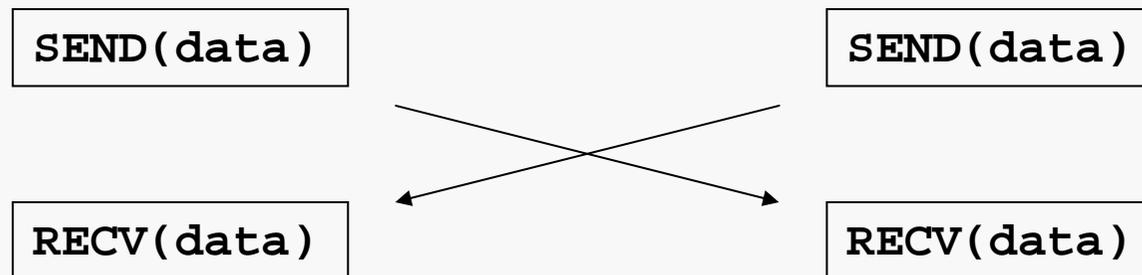


- `#include <mpi.h>`
- The first line of `main(argc, argv)` must be
 - `MPI_Init(&argc, &argv);`
- `MPI_COMM_WORLD` is the initial communicator
- Important initial information:
 - `int id, nprocs;`
 - `MPI_Comm_rank(MPI_COMM_WORLD, &id);`
 - `MPI_Comm_size(MPI_COMM_WORLD, &nprocs);`
- Last Statement is
 - `MPI_Finalize()`

MPI Execution



- Different processes each of them with an integer ID from 0 to P-1
- Communicate through send/receive
- Beware of low level protocol implementation



- This can cause a deadlock depending on size of `data`

MPI send

```
MPI_Send(BUFF, /* address of storage */  
          SIZE, /* elements to send */  
          MPI_TYPE, /* type of data */  
          DEST, /* destination process */  
          TAG, /* message identifier */  
          COMMUNICATOR); /* id space */
```

MPI Send is a blocking operation: when function returns **BUFF** has been sent to destination and can be reused.

MPI Receive



```
MPI_Recv(BUFF, /* address of storage */  
          SIZE, /* elements to send */  
          MPI_TYPE, /* type of data */  
          DEST, /* destination process */  
          TAG, /* message identifier */  
          COMMUNICATOR, /* id space */  
          &status); /* info about data */
```

MPI Recv is a blocking operation: when function returns **BUFF** contains payload data from a message matching **TAG**.

MPI Isend

```
MPI_Isend(BUFF, /* address of storage */  
          SIZE, /* elements to send */  
          MPI_TYPE, /* type of data */  
          DEST, /* destination process */  
          TAG, /* message identifier */  
          COMMUNICATOR, /* id space */  
          REQUEST); /* to check status */
```

MPI Isend is a NON blocking operation: when function returns **BUFF** may be still unsent and can NOT be safely reused.

MPI Irecv



```
MPI_Irecv(BUFF, /* address of storage */  
          SIZE, /* elements to send */  
          MPI_TYPE, /* type of data */  
          DEST, /* destination process */  
          TAG, /* message identifier */  
          COMMUNICATOR, /* id space */  
          REQUEST); /* to check status */
```

MPI Irecv is a NON blocking operation: when function returns **BUFF** may not contain payload data from a message matching **TAG**.

MPI Test&Wait



- Used to check status of non blocking operations like `MPI_Isend` and `MPI_Irecv`
- **`MPI_Wait(&request, &status);`**
 - Returns when `request` is completed
 - `status` variable as in `MPI_Recv`
- **`MPI_Test(&request, &flag, &status);`**
 - `int flag` matches true if `request` succeeded

MPI Others



- **MPI_Barrier (COMMUNICATOR)**
 - All processes in COMMUNICATOR must reach barrier before proceeding
- **MPI_Reduce (&var, &res, size, MPI_TYPE, MPI_OP, proc_id, MPI_COMMUNICATOR);**
 - `var` is the input value in each processor
 - `res` is the variable containing the result in `proc_id` processor
 - `size` is the size of the buffer
 - `MPI_OP` identify the operation (e.g., `MPI_SUM`)
- **MPI_Bcast (BUF, SIZE, TYPE, ROOT_ID, COMM)**
 - Broadcast the content of `BUF` from `ROOT_ID` to all `COMM`

Compiling MPI



- `mpicc` is a front end to gcc compiler
- Same options as gcc apply to `mpicc`
- To run an MPI program
 - `mpirun -nprocs <n> programname`
 - `mpirun -nprocs <n> -machinefile
-file progname`
 - `machinefile` contains the names of
computing nodes to use

Hello world



- Write an “Hello world I’m process # of # processes” in MPI

Hello world: MPI



```
#include <stdio>
#include <mpi.h>

int main(int argc, char** argv)
{
    int id, nprocs;

    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &id);
    MPI_Comm_size(MPI_COMM_WORLD, &nprocs);

    printf("Hello from process %u of %u\n", id, nprocs);

    MPI_Finalize();
    return 0;
}
```

String Matching



- Count the number of occurrences of a pattern in a (random) text
 - MPI
 - TBB
 - OpenMP
- In MPI text is distributed across the processes and the pattern is known to everyone

MPI: String Matching 1/4



```
MPI_Init(&argc, &argv);
int N=atoi(argv[1]);
int P;
MPI_Comm_size(MPI_COMM_WORLD, &P);

N=N/P;

std::vector<char> V(N);
//Initialize data.
int pid;
MPI_Comm_rank(MPI_COMM_WORLD, &pid);

std::for_each(V.begin(), V.end(), init(pid));

std::string S(argv[2]);
int M=S.length();
```

MPI: String Matching 2/4



```
//Count matches on local data.
int cnt=0;
for (int i=0; i <= N-M+1; ++i) {
    bool match = true;
    for (int j=0; j < M; ++j) {
        if (V[i+j] != S[j])
            match = false;
    }
    if (match) ++cnt;
}
```

MPI: String Matching 3/4



```
if (pid>0) {
    MPI_Send(&V[0], M-1, MPI_CHAR, pid-1, 1,
            MPI_COMM_WORLD);
}
if (pid<P-1) {
    std::vector<char> BUFF(2*(M-1));
    std::copy(V.begin()+N-M+1, V.end(), BUFF.begin());
    MPI_Status status;
    MPI_Recv( &BUFF[M-1], M-1, MPI_CHAR, pid+1, 1,
            MPI_COMM_WORLD, &status );
    for (int i=0; i <= M-1; ++i) {
        bool match=true;
        for (int j=0; j < M; ++j) {
            if (BUFF[i+j]!=S[j])
                match=false;
        }
        if (match) ++cnt;
    }
}
```

MPI: String Matching 4/4



```
double match_time = match_timer.stop();
reduce_timer.start();

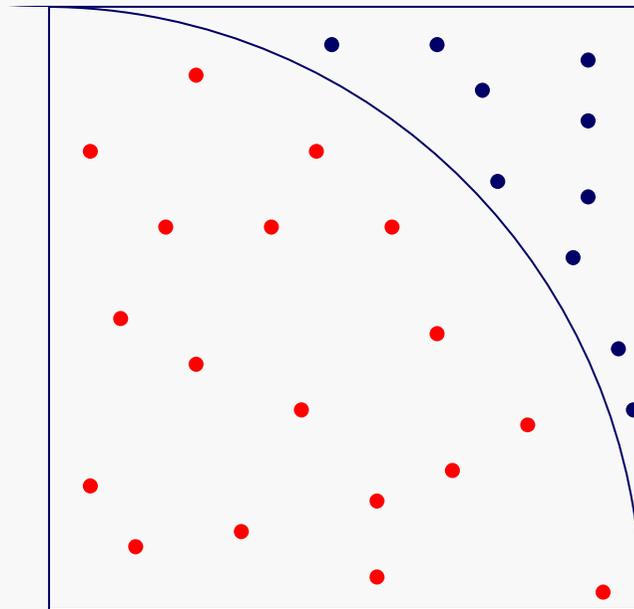
int res;
MPI_Reduce ( &cnt, &res, 1, MPI_INT, MPI_SUM, 0,
MPI_COMM_WORLD );

MPI_Finalize();
if (pid==0) {
    printf("RESULT    ==>    %d\n",res);
    printf("Overall %f Initialize %f Match %f Reduce
%f\n",
        overall_time, init_time, match_time,
        reduce_time);
}

return 0;
```

PI Example

- Compute Pi with a Monte Carlo simulation
- Generate N random points in the unit square
- Count how many falls in the unit circle: **M**
- $4 * M / N = \text{Pi}$



MPI: Pi 1/3



```
MPI_Init(&argc, &argv);
```

```
int N=atoi(argv[1]);
```

```
int P;
```

```
MPI_Comm_size(MPI_COMM_WORLD, &P);
```

```
N=N/P;
```

```
int pid;
```

```
MPI_Comm_rank(MPI_COMM_WORLD, &pid);
```

```
srand48(pid);
```

MPI: Pi 2/3



```
int cnt(0);
for (int i=0; i<N; ++i) {
    double xcoord = drand48();
    double ycoord = drand48();

    double dist = std::sqrt(std::pow(xcoord, 2.0) +
                             std::pow(ycoord, 2.0));
    if (dist < 1.0)
        ++cnt;
}
//Collect the number of matches from all processes.
int res;
MPI_Reduce ( &cnt, &res, 1, MPI_INT, MPI_SUM, 0,
             MPI_COMM_WORLD );

if (pid==0) {
    double pi = 4*(res/(double)(N*P));
    printf("RESULT    ==>    %f\n",pi);
}
```

Matrix Multiplication



- Implement a dense matrix multiplication algorithm in
 - TBB
 - OpenMP
 - MPI
- $C += A * B$, A is $N \times L$, B is $L \times M$, C is $N \times M$
- Input are three integers: N,L,M
- Initialize matrices and perform computation checking correctness

Basic sequential loop



```
void SMM(float A[][SIZEZ],
         float B[][SIZEZ],
         float C[][SIZEZ],
         size_t Arows, size_t Acols , size_t Bcols) {
    for( size_t i=0; i<Arows; ++i ) {
        for( size_t j=0; j<Bcols; ++j ) {
            for( size_t k=0; k<Acols; ++k ) {
                C[i][j] += A[i][k]*B[k][j];
            }
        }
    }
}
```

TRY SWAPPING THE TWO INNER
LOOPS AND MEASURE TIME – with
optimizations on

MPI version



- Each Processor holds N/P rows of A and C , and M/P columns of B
- Each processor computes $A*B$ (on its portion)
- Then sends B to the previous *or next) processor
- Repeat for P times
- Assume P divides N , L , and M

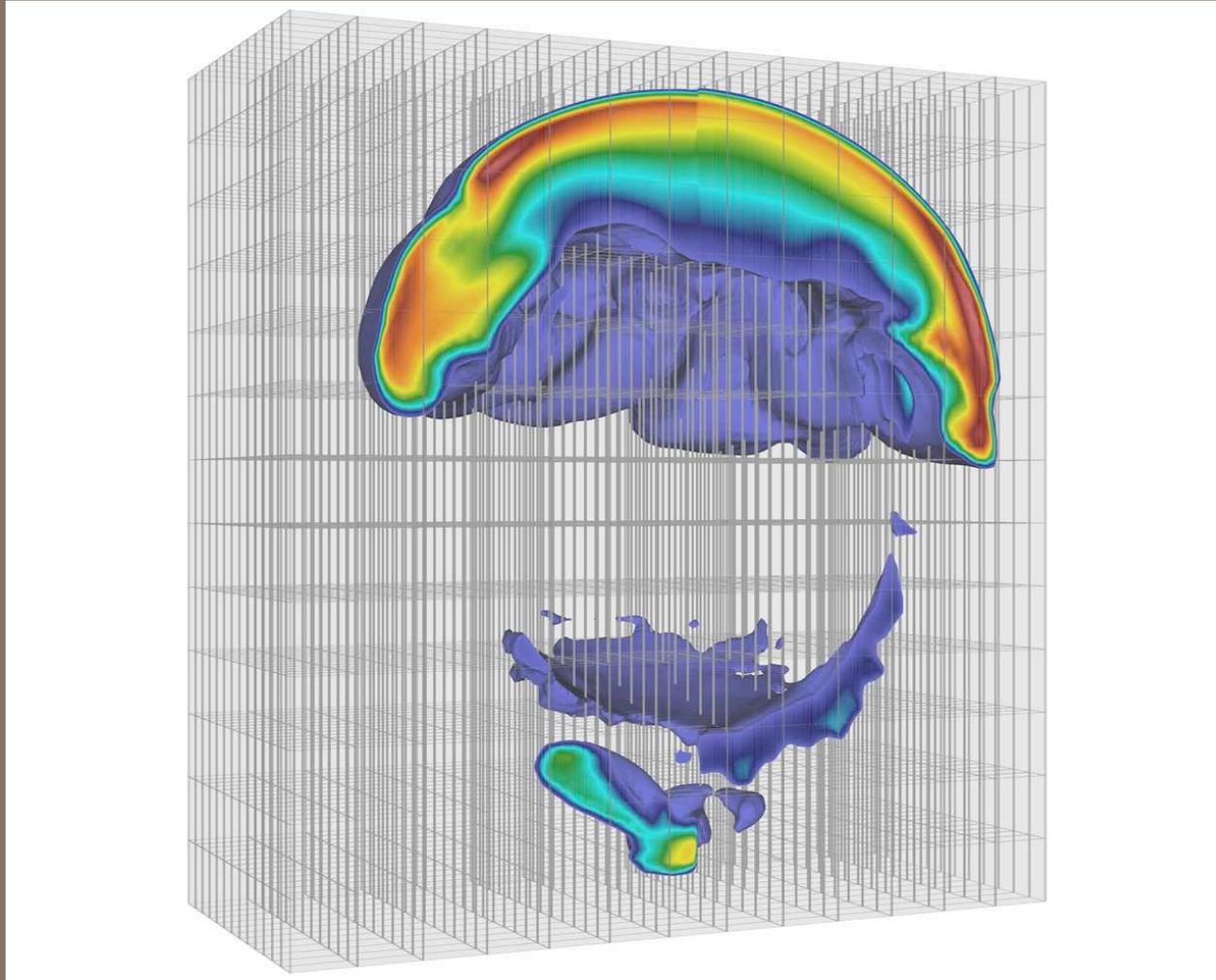
MPI matrix multiplication



```
MPI_Init(&argc, &argv);
const size_t Arows=SIZEZ, Acols=SIZEZ, Bcols=SIZEZ;
int P;    MPI_Comm_size(MPI_COMM_WORLD, &P);
int pid;  MPI_Comm_rank(MPI_COMM_WORLD, &pid);
MPI_Status status;
MPI_Request r;
// Each processor has Arows/P rows of A and all of B, C
// stored like A
float *A = new float[SIZEZ*SIZEZ/P];
float *B = new float[SIZEZ*SIZEZ/P];
float *tmp = new float[SIZEZ*SIZEZ/P];
float *C = new float[SIZEZ*SIZEZ/P];

CreateData(A, B, pid, Arows/P, Acols, Bcols/P);
InitC(C0,Arows,Bcols/P);
```

Parallel visualization



June 14, 2011

Hank Childs, Lawrence Berkeley Lab & UC Davis

Outline



- Parallel visualization basics
- Smart techniques and data flow
- Contracts
- Parallel Rendering
- IceT
- Performance study

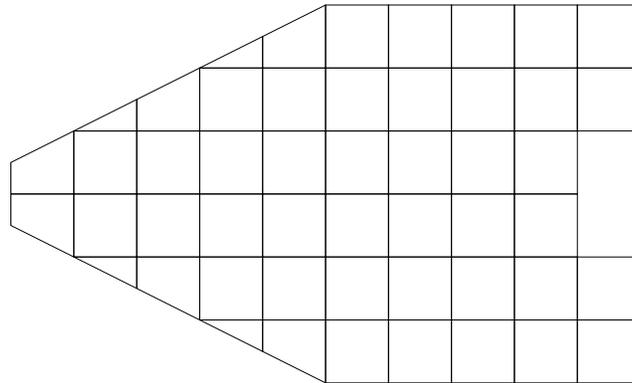
Outline



- **Parallel visualization basics**
- Smart techniques and data flow
- Contracts
- Parallel Rendering
- IceT
- Performance study

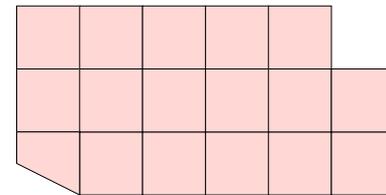
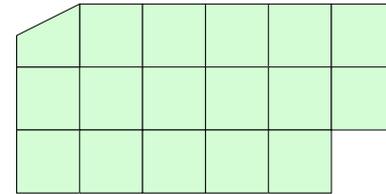
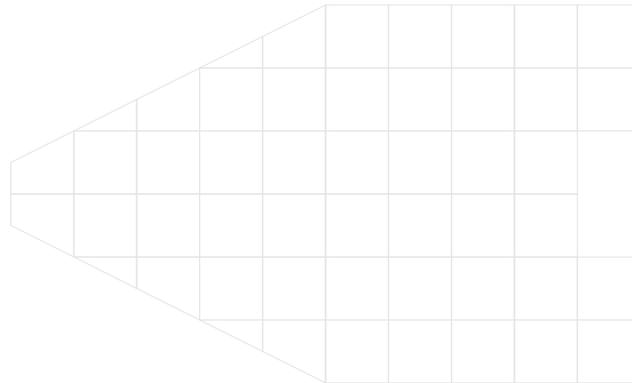
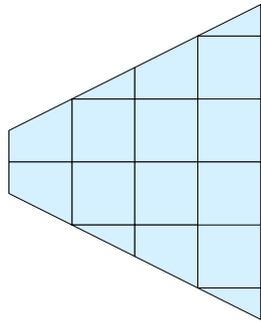
Data Parallel Pipelines

- Duplicate pipelines run independently on different partitions of data.



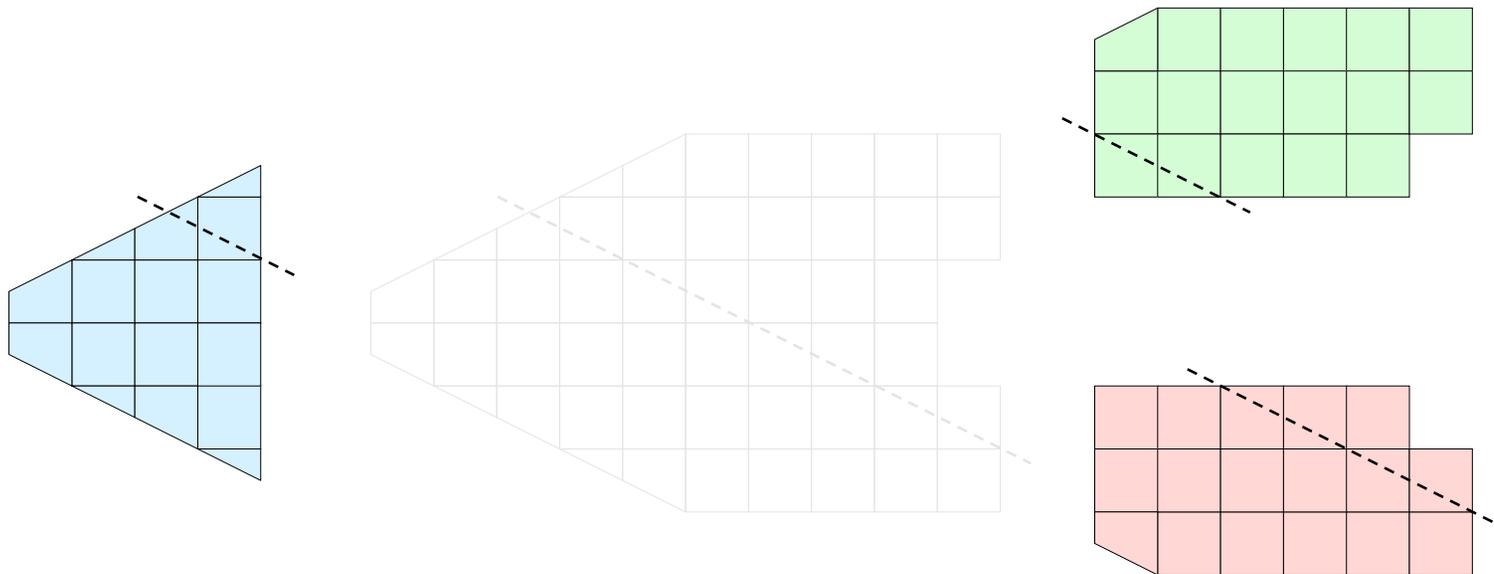
Data Parallel Pipelines

- Duplicate pipelines run independently on different partitions of data.



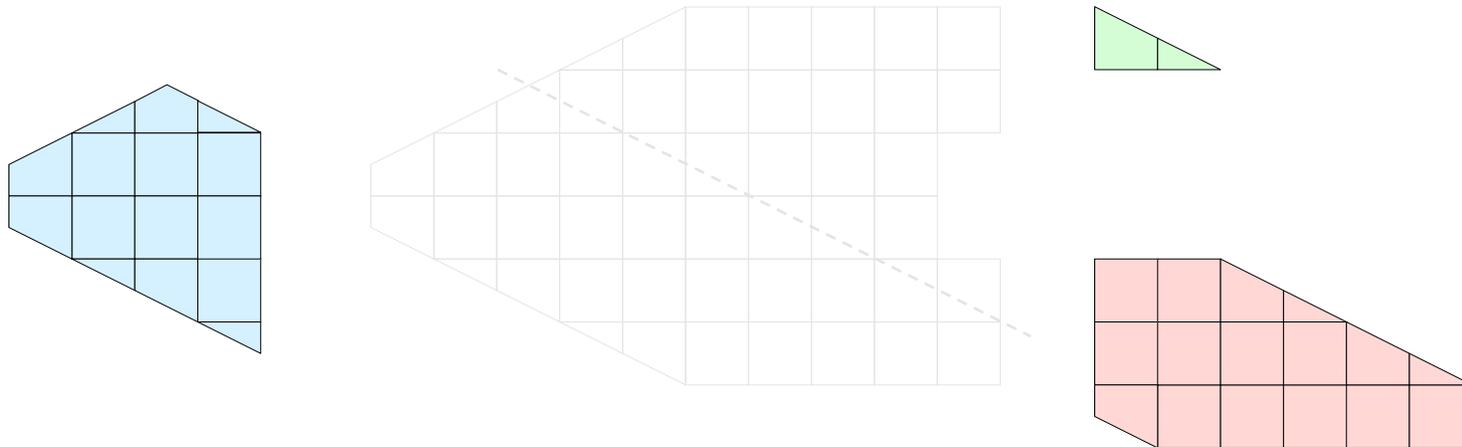
Data Parallel Pipelines

- Some operations will work regardless.
 - Example: Clipping.



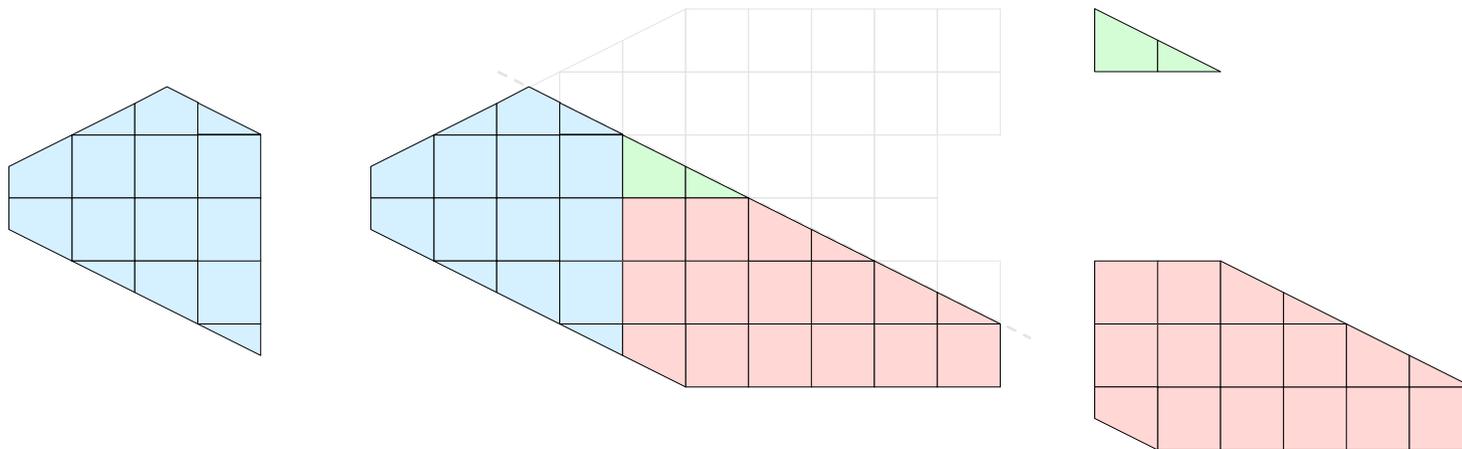
Data Parallel Pipelines

- Some operations will work regardless.
 - Example: Clipping.



Data Parallel Pipelines

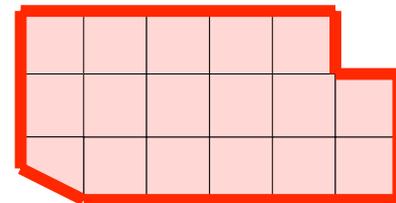
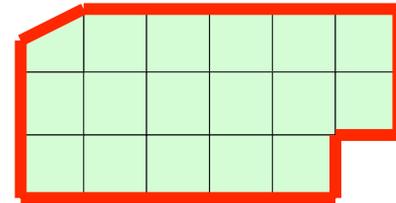
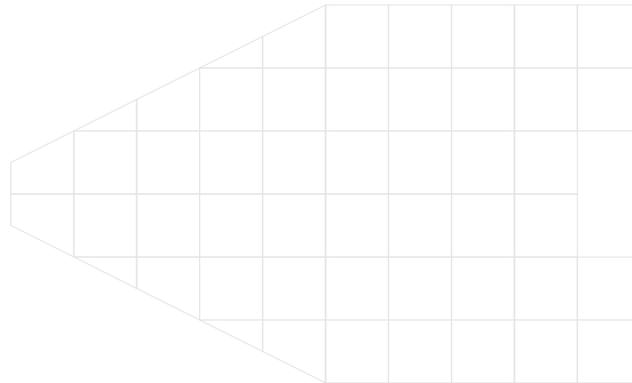
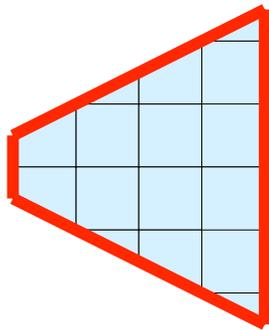
- Some operations will work regardless.
 - Example: Clipping.



Slide courtesy of Ken Moreland, Sandia Lab

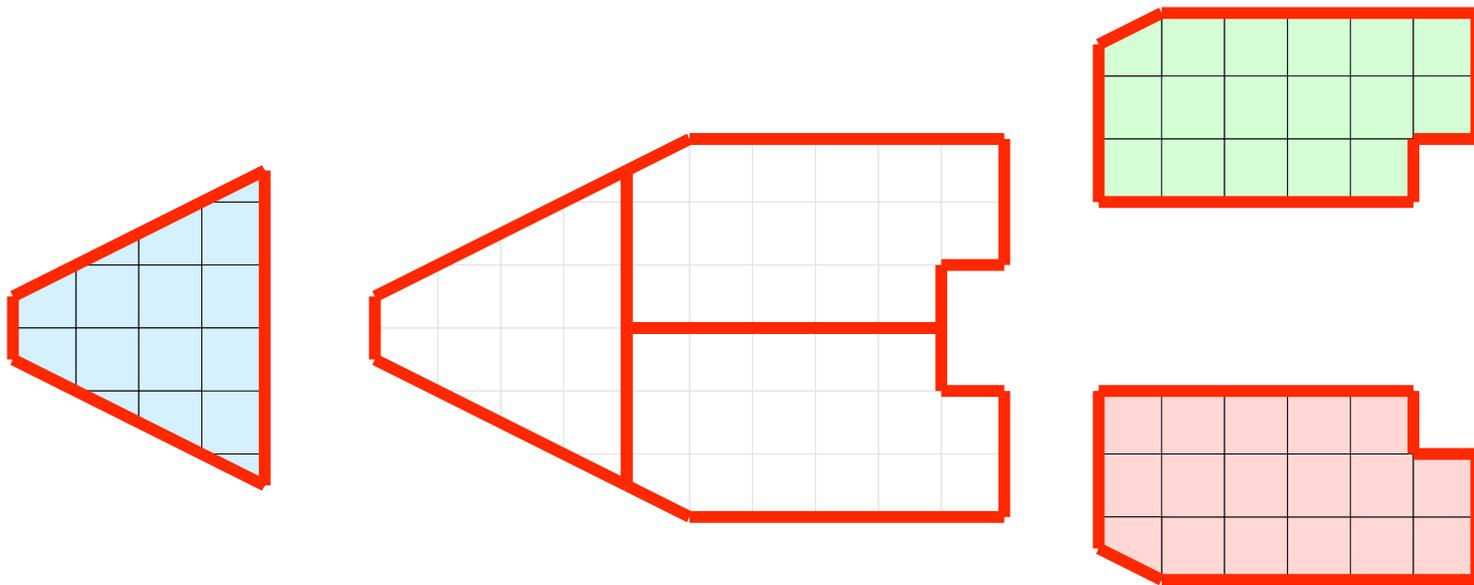
Data Parallel Pipelines

- Some operations will have problems.
 - Example: External Faces



Data Parallel Pipelines

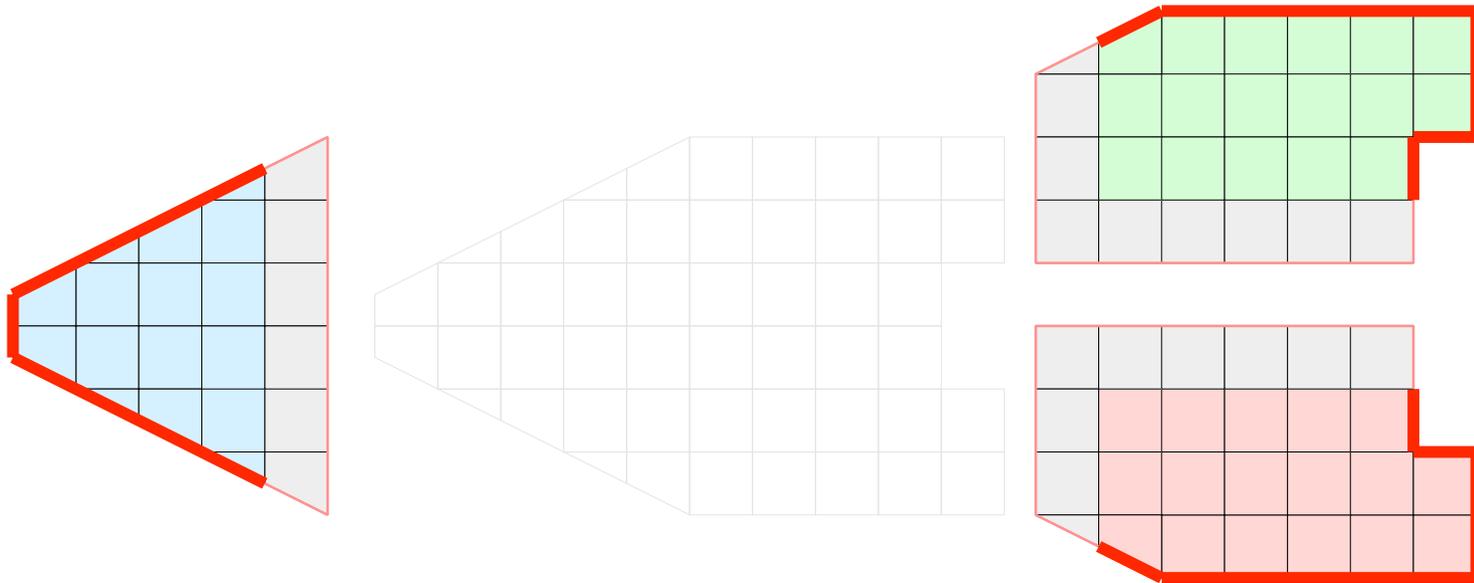
- Some operations will have problems.
 - Example: External Faces



Slide courtesy of Ken Moreland, Sandia Lab

Data Parallel Pipelines

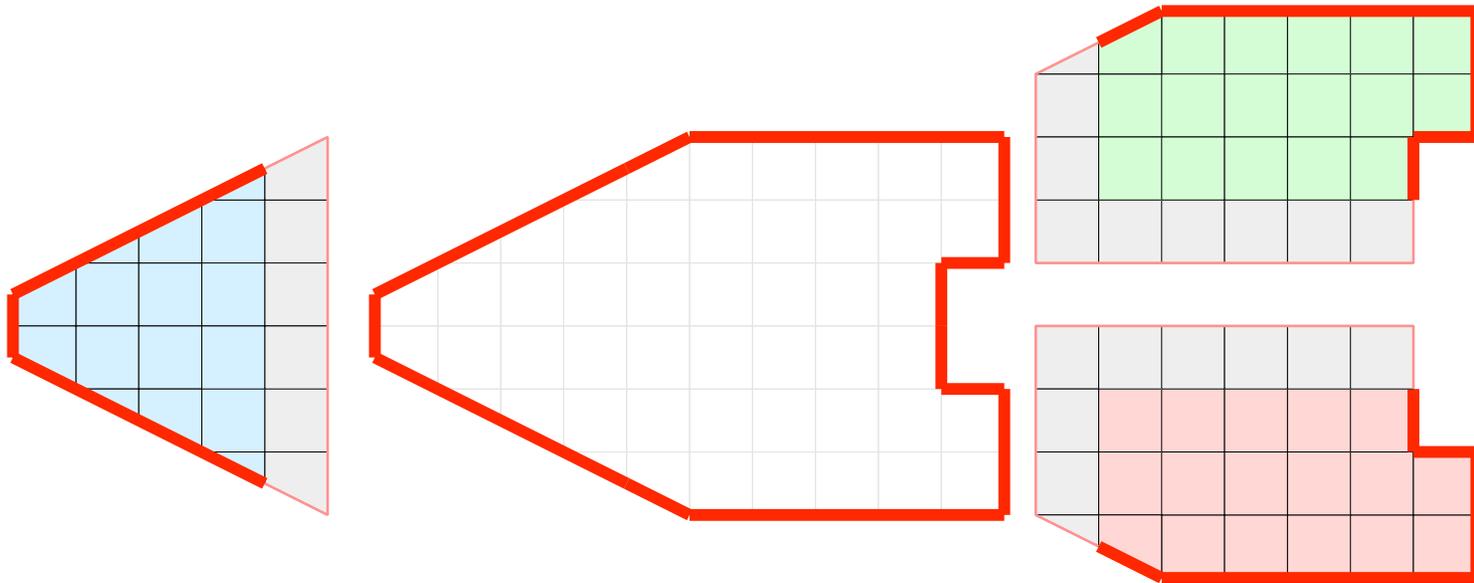
- Ghost cells can solve most of these problems.



Slide courtesy of Ken Moreland, Sandia Lab

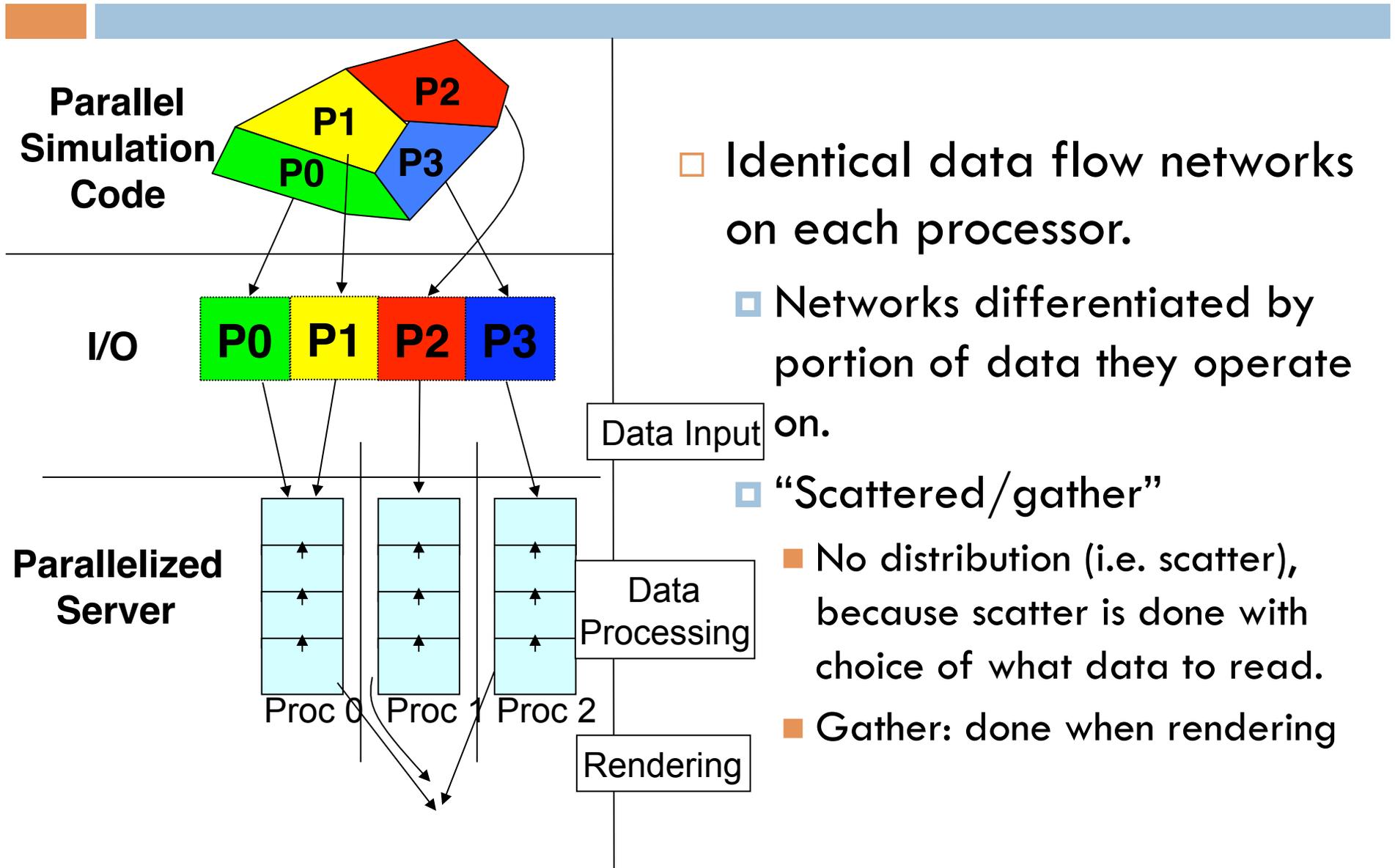
Data Parallel Pipelines

- Ghost cells can solve most of these problems.



Slide courtesy of Ken Moreland, Sandia Lab

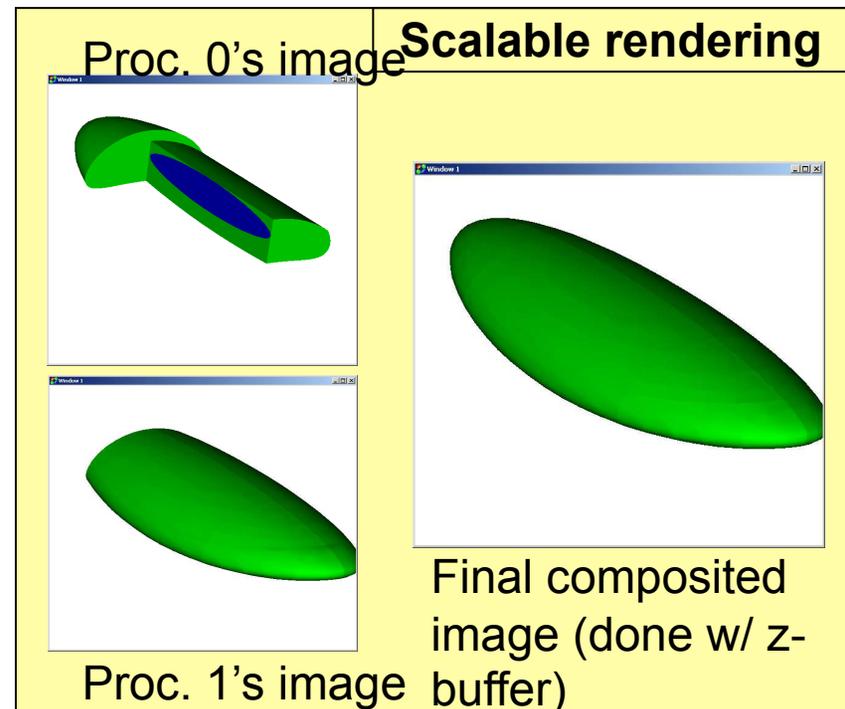
Parallelization covers data input, data processing, and rendering.



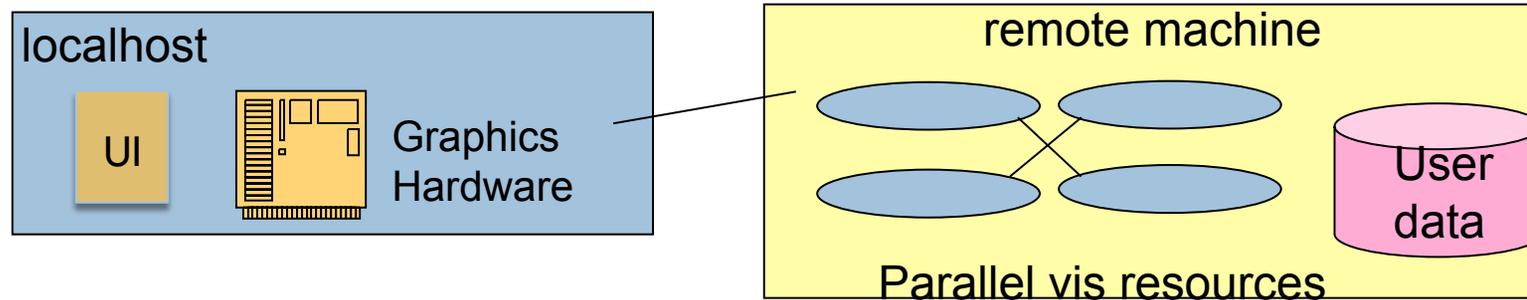
- Identical data flow networks on each processor.
- Networks differentiated by portion of data they operate on.
- “Scattered/gather”
 - No distribution (i.e. scatter), because scatter is done with choice of what data to read.
 - Gather: done when rendering

Parallel rendering (basic “sort last” version)

- Parallel rendering
 - Every processor renders local geometry
 - Z-buffer is used to compare depth of images
 - Communication between processor for final image
- Harder:
 - Shadows, transparency, ray-casting
- More on this topic later...



The standard architecture for data flow network-based tools.



□ Observations:

- Good for remote visualization
- Leverages available resources
- Scales well
- No need to move data

Outline



- Parallel visualization basics
- Smart techniques and data flow
- Contracts
- Parallel Rendering
- IceT
- Performance study

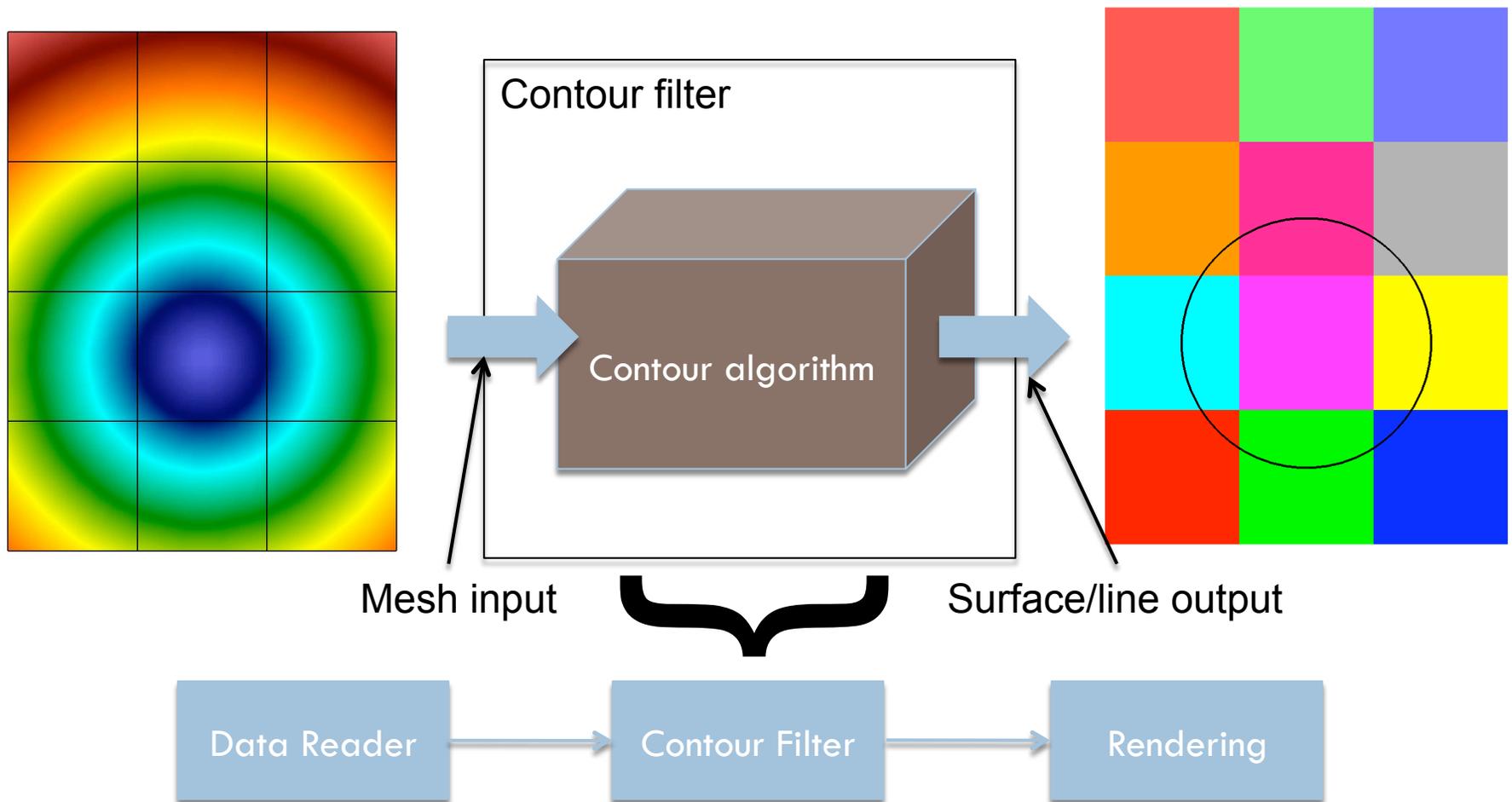
Data flow networks: observations



- Source for managing flow of data is small and in one place
- Majority of code investment is in algorithms (derived types of filters), not in base classes (which manage data flow).

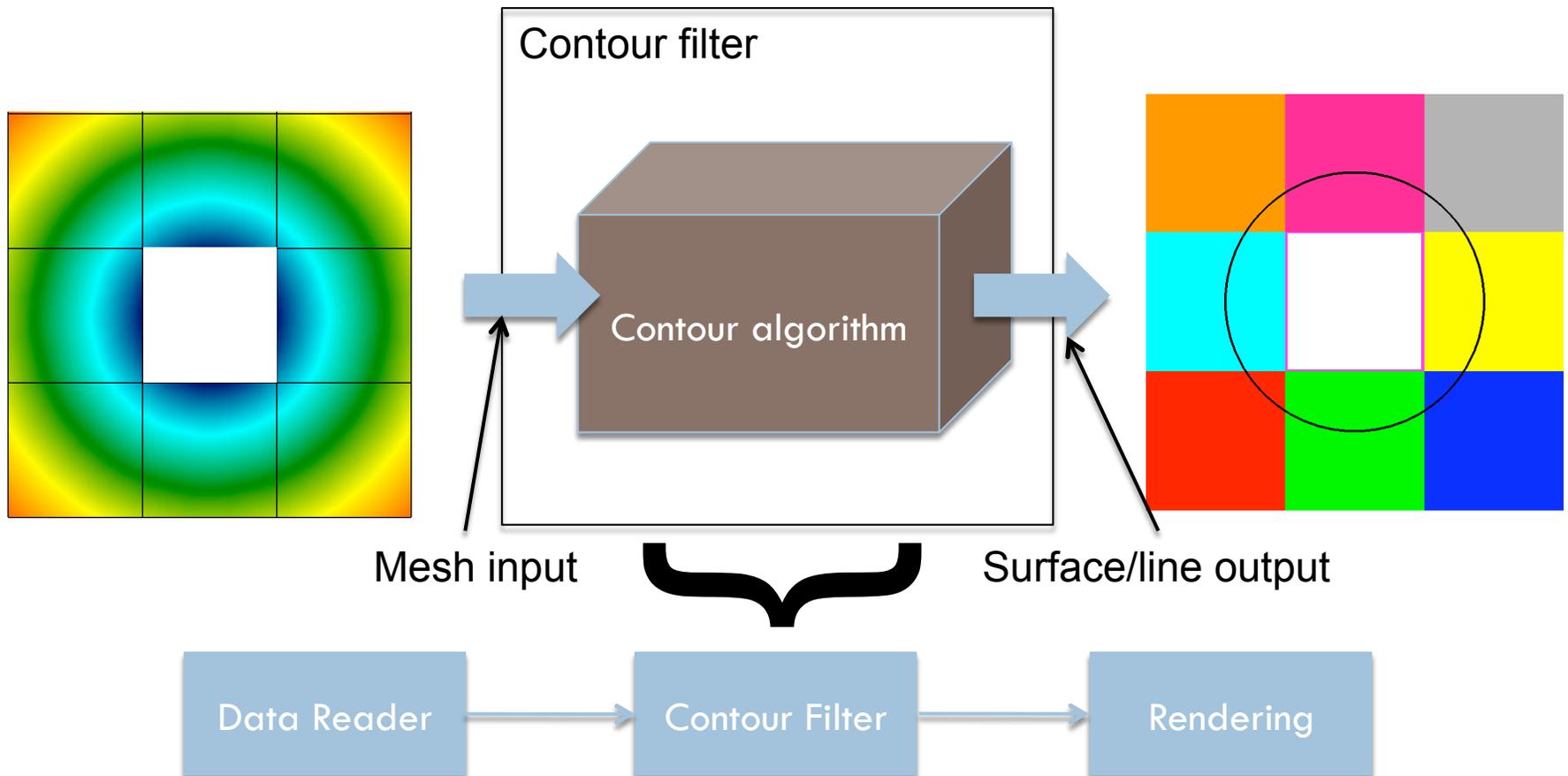
Algorithms don't care about data processing paradigm ... they only care about operating on inputs and outputs.

Example filter: contouring



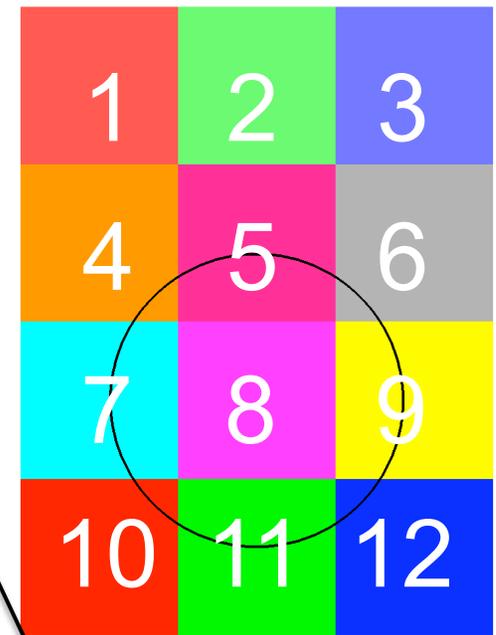
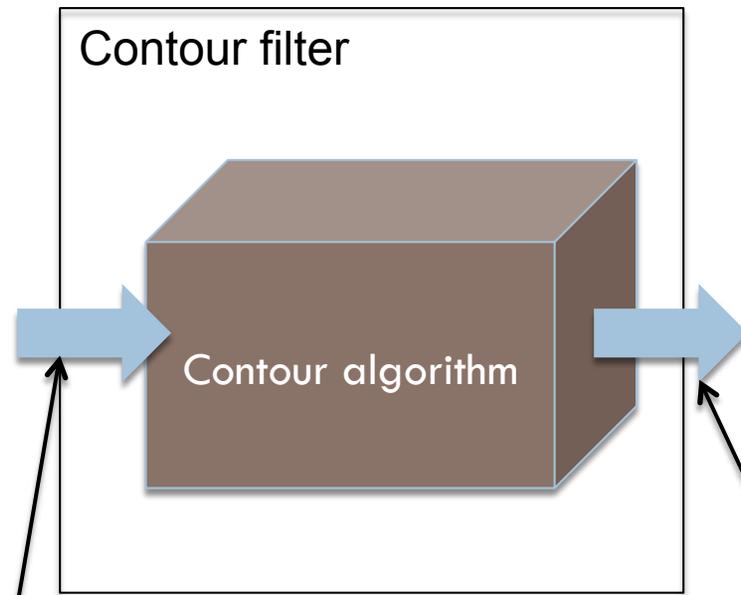
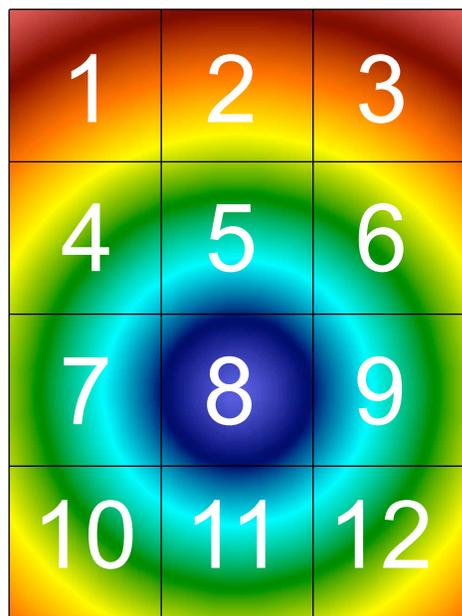
Example filter: contouring with data subsetting

Communicate
with executive to
discard domains



Example filter: contouring with out-of-core

Algorithm called
12 times

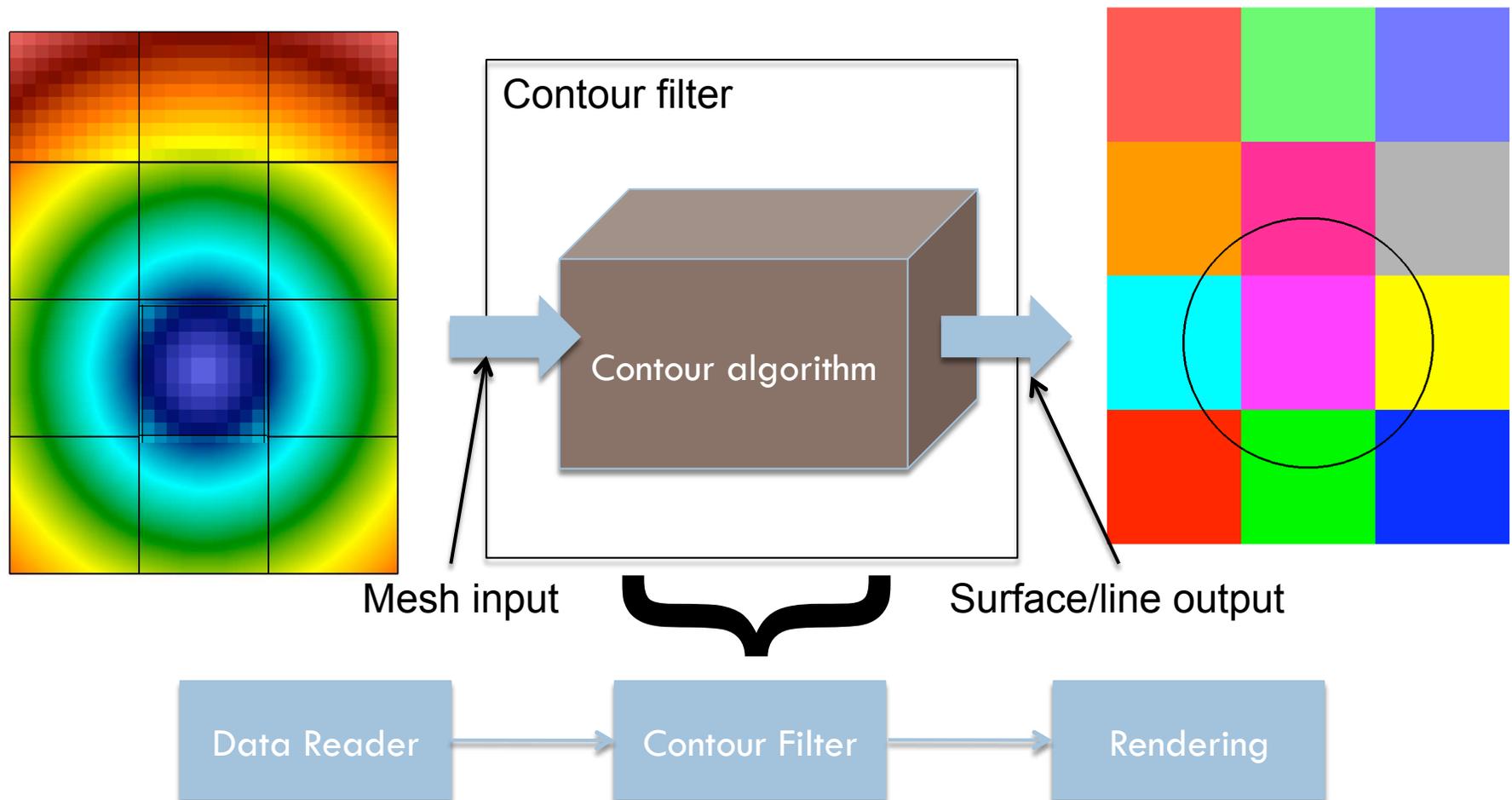


Mesh input

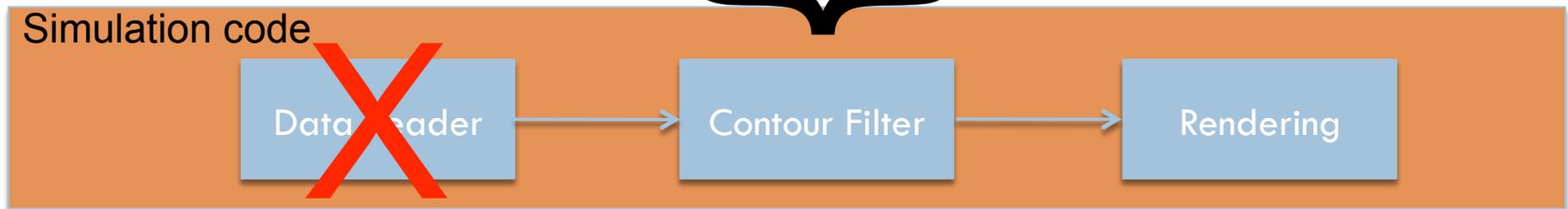
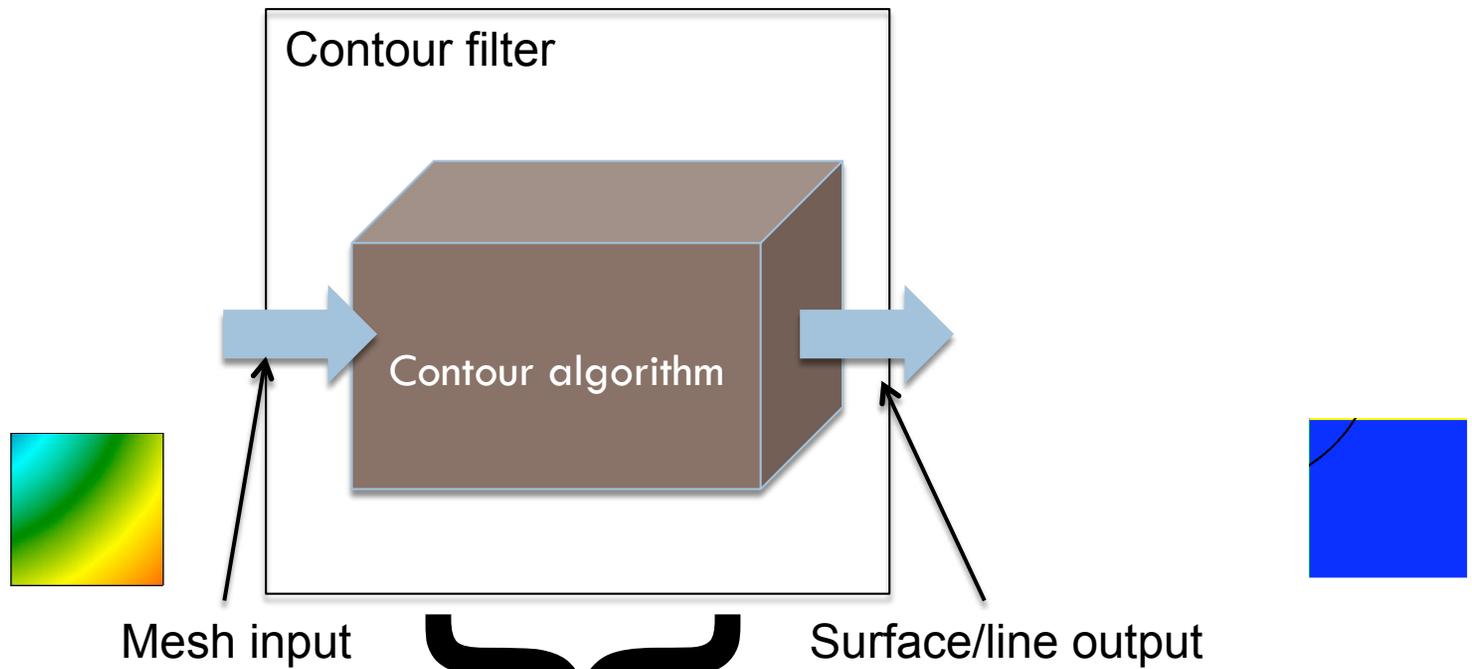
Surface/line output



Example filter: contouring with multi-resolution techniques



Example filter: contouring with in situ



How Petascale Changes the Rules



- We can't use pure parallelism alone any more
- We will need many techniques to work in many processing paradigms
- Data flow networks are a good fit: write algorithm once, use anywhere
 - ▣ Only core infrastructure has to worry about the processing paradigm

Outline

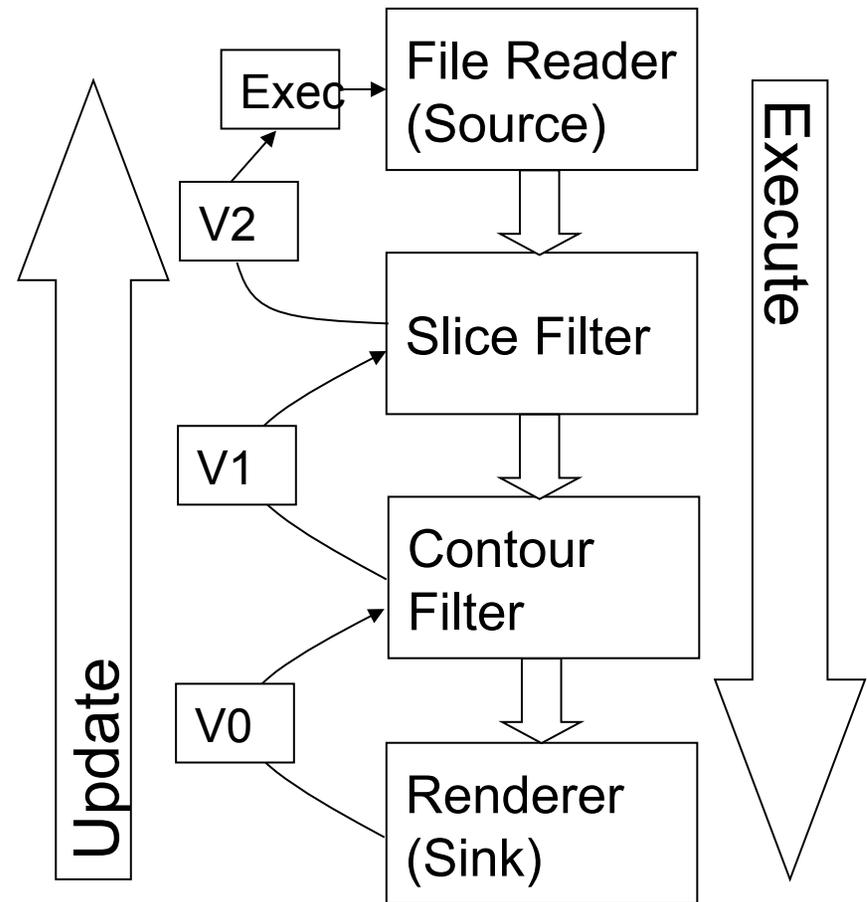


- Parallel visualization basics
- Smart techniques and data flow
- **Contracts**
- Parallel Rendering
- IceT
- Performance study

Contracts are an extension to the standard data flow network design.

Data Flow Networks “101”:

- Work is performed by a pipeline
- A pipeline consists of data objects and components (sources, filters, and sinks)
- Pipeline execution begins with a “pull”, which starts Update phase
- Data flows from component to component during the Execute phase



Extension:

- Contracts are coupled with the Update phase

Initial observations about contracts.

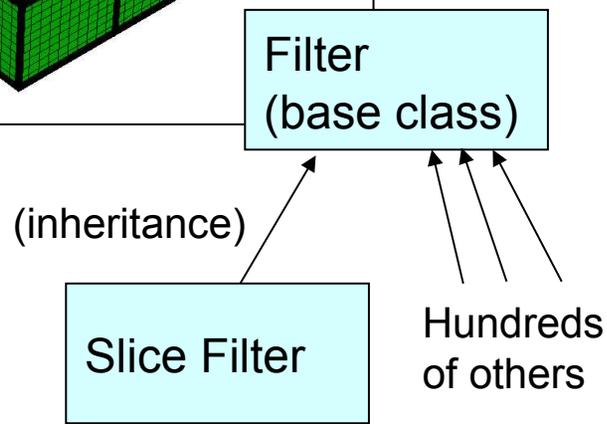
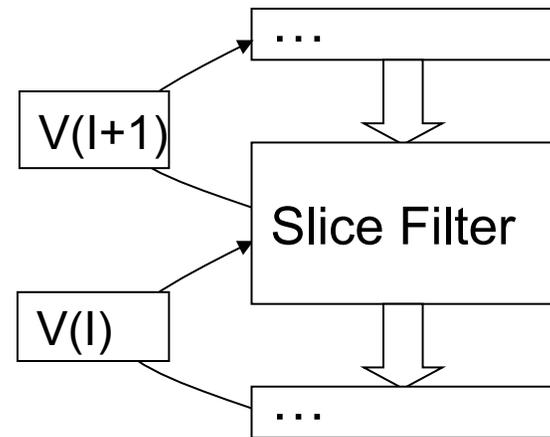
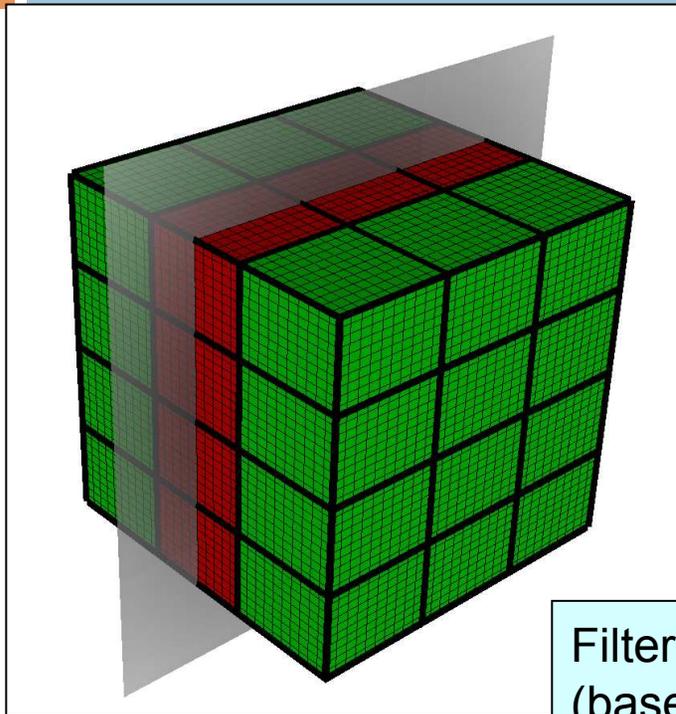
- A contract is simply a data structure
 - ▣ The members of the data structure reflect the optimizations
- Optimizations are adaptively applied based on the final contract
- Each component describes its impact on the pipeline
 - ▣ Allows for effective management of hundreds of components
 - ▣ Allows for new, unforeseen components to be added
- Combining contracts with the Update phase
 - seamlessly integrated into data flow networks
 - every component has a chance to modify the contract

Why are contracts important?



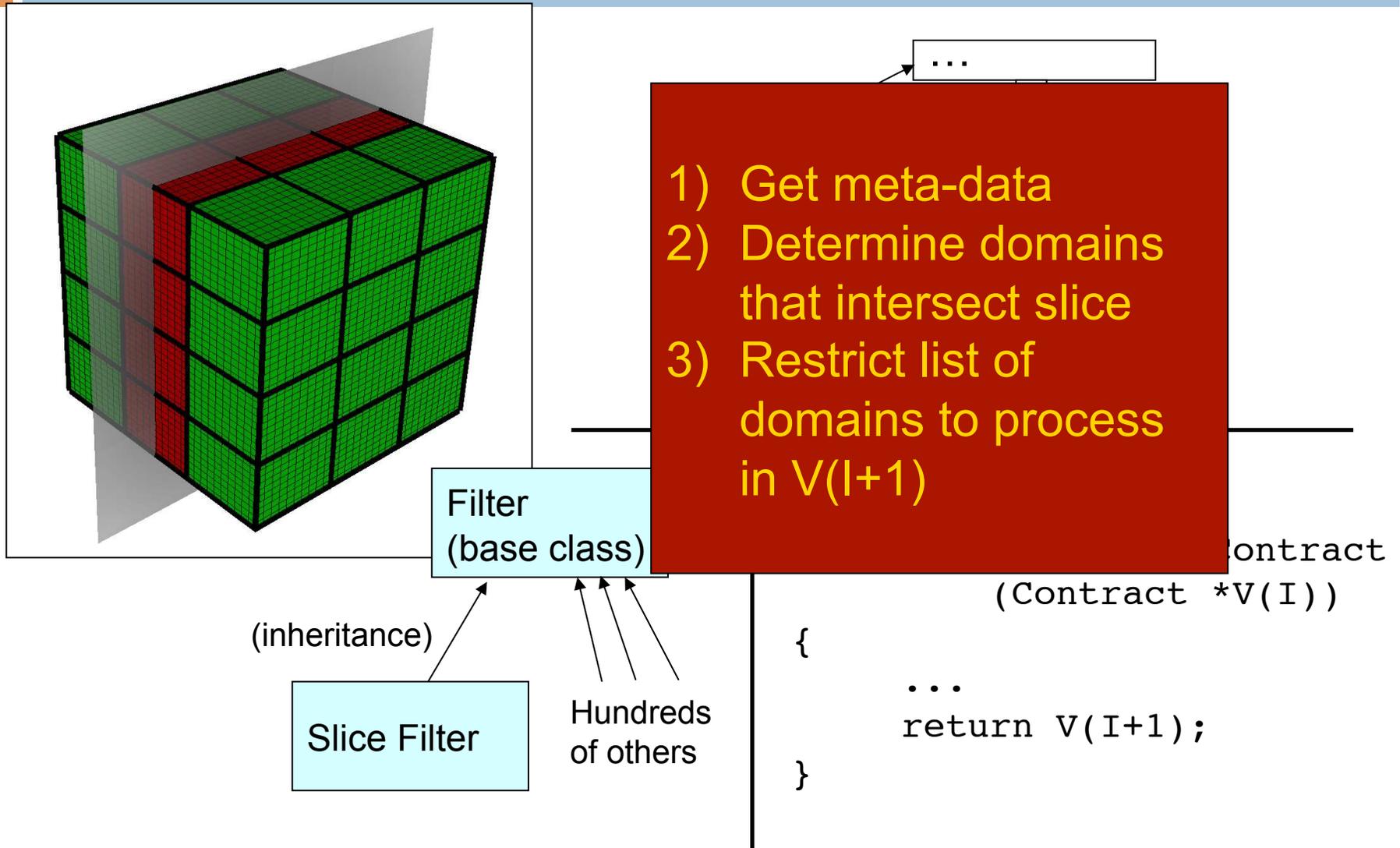
- Contracts are important for managing which optimizations can be utilized in a richly featured system
- We will look at the impact of these optimizations to better understand their importance.

Operating on Optimal Subset of Data

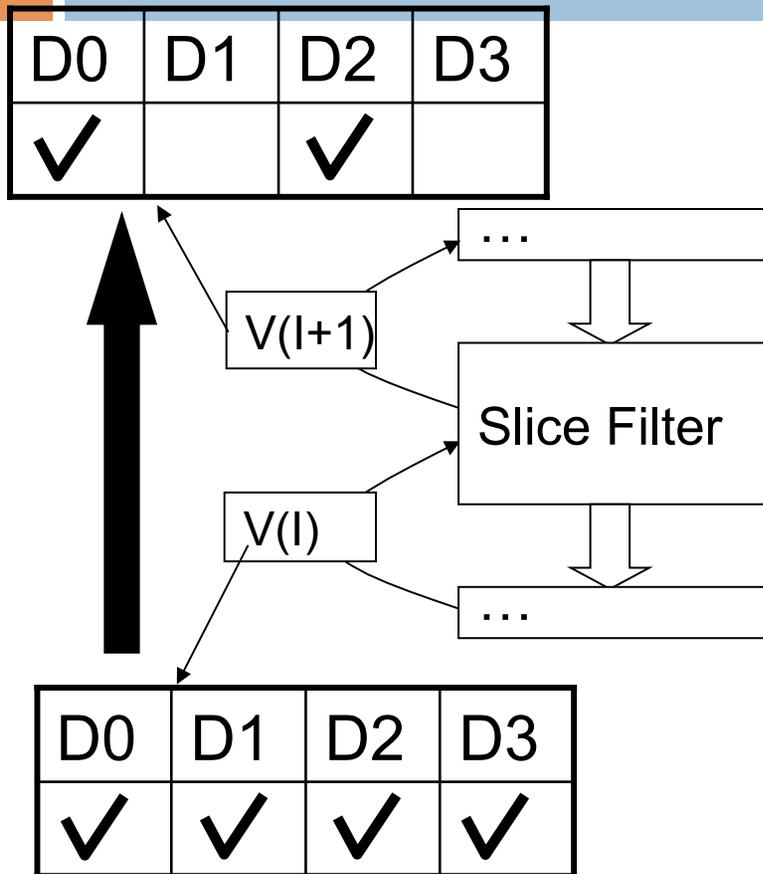


```
Contract *
SliceFilter::ModifyContract
    (Contract *V(I))
{
    ...
    return V(I+1);
}
```

Operating on Optimal Subset of Data

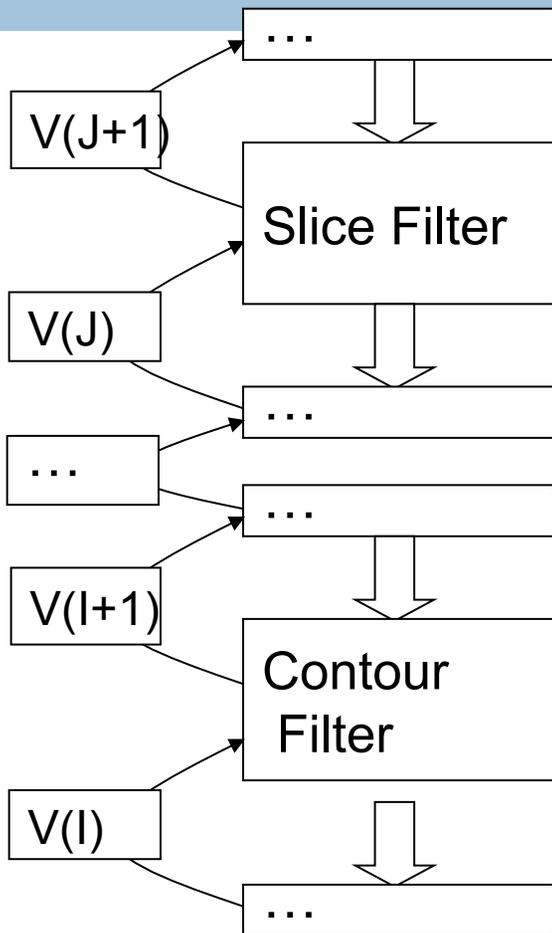


Operating on Optimal Subset of Data

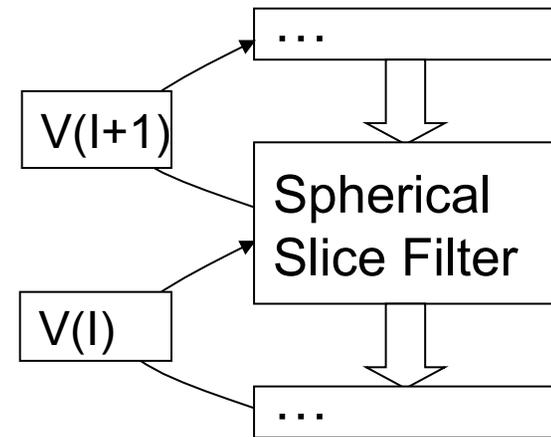


- 1) Get meta-data
- 2) Determine domains that intersect slice
- 3) Restrict list of domains to process in $V(I+1)$

The contract-based system provides high flexibility for this optimization.



Multiple filters can use the same optimizations



A new, plugin filter can use this optimization without any modification to system

We studied performance using a simulation of a Rayleigh-Taylor

Instability.

- RTI: heavy and light fluids mixing
 - ▣ 1.5B elements
 - ▣ 729 domains
- LLNL's thunder
 - ▣ Top 500's #7
(We only got a little bit of it)
 - ▣ 1.4GHz Intel Itanium2

We studied performance using a simulation of a Rayleigh-Taylor

Instability.

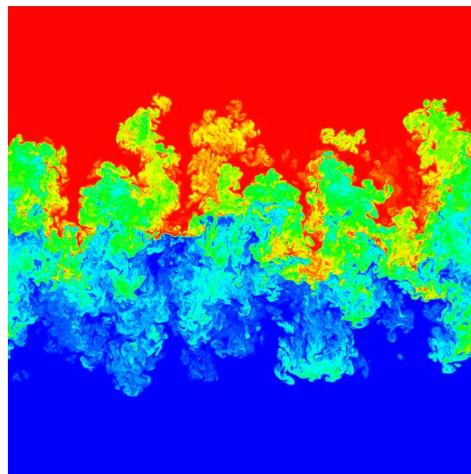
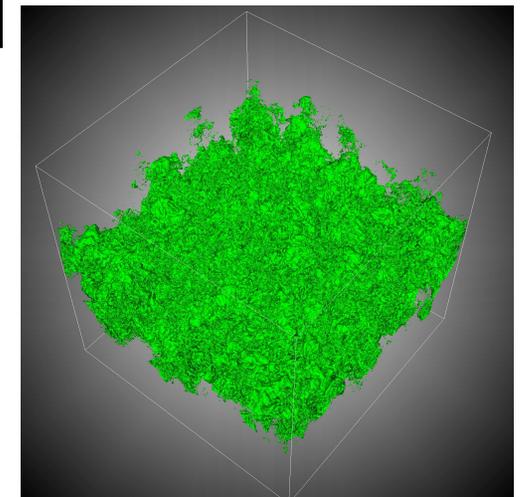
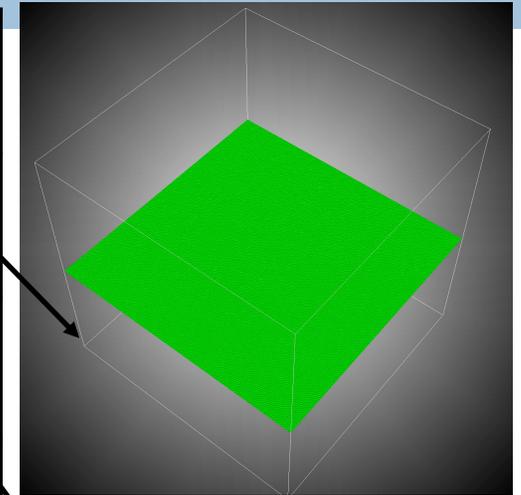
The techniques shown are not new

The performance increase motivates the importance of optimizations

This, in turn, motivates the importance of contracts

Processing only the necessary domains is a lucrative optimization.

Algorithm	Processors	Without Contracts	With Contracts	Speedup
Contouring (early)	32	41.1s	5.8s	7.1X
Contouring (late)	32	185.0s	97.2s	1.9X
Slicing	32	25.3s	3.2s	7.9X



What is the right technique for distributing domains across processors?

- Two ways:
 - ▣ Statically: make assignments before Execute phase
 - ▣ Dynamically: adaptively during Execute phase
- Performance:
 - ▣ Static: good chance of load imbalance
 - As fast as slowest processor
 - ▣ Dynamic: adaptively balancing load
 - Obtains near optimal parallel efficiency
- Communication:
 - ▣ Static: collective communication okay
 - ▣ Dynamic: no collective communication

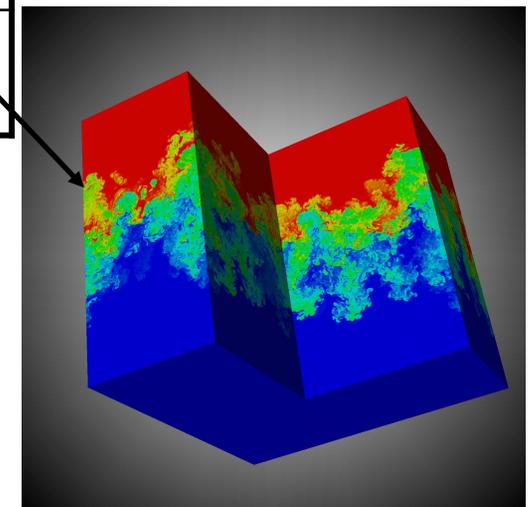
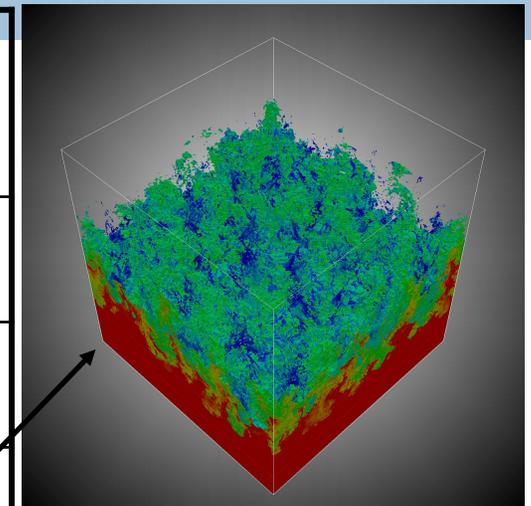
Contracts steer what load balancing technique we use.

- What load balancing technique should we use?
- If we need collective communication → static
- Otherwise, we want performance → dynamic

- Contracts enable this
 - ▣ During Update phase:
 - Every filter can modify the contract to state whether or not it needs collective communication
 - ▣ Before Execute phase:
 - Executive examines contract and decides which load balancing technique to use.

Employing dynamic load balancing is a lucrative optimization.

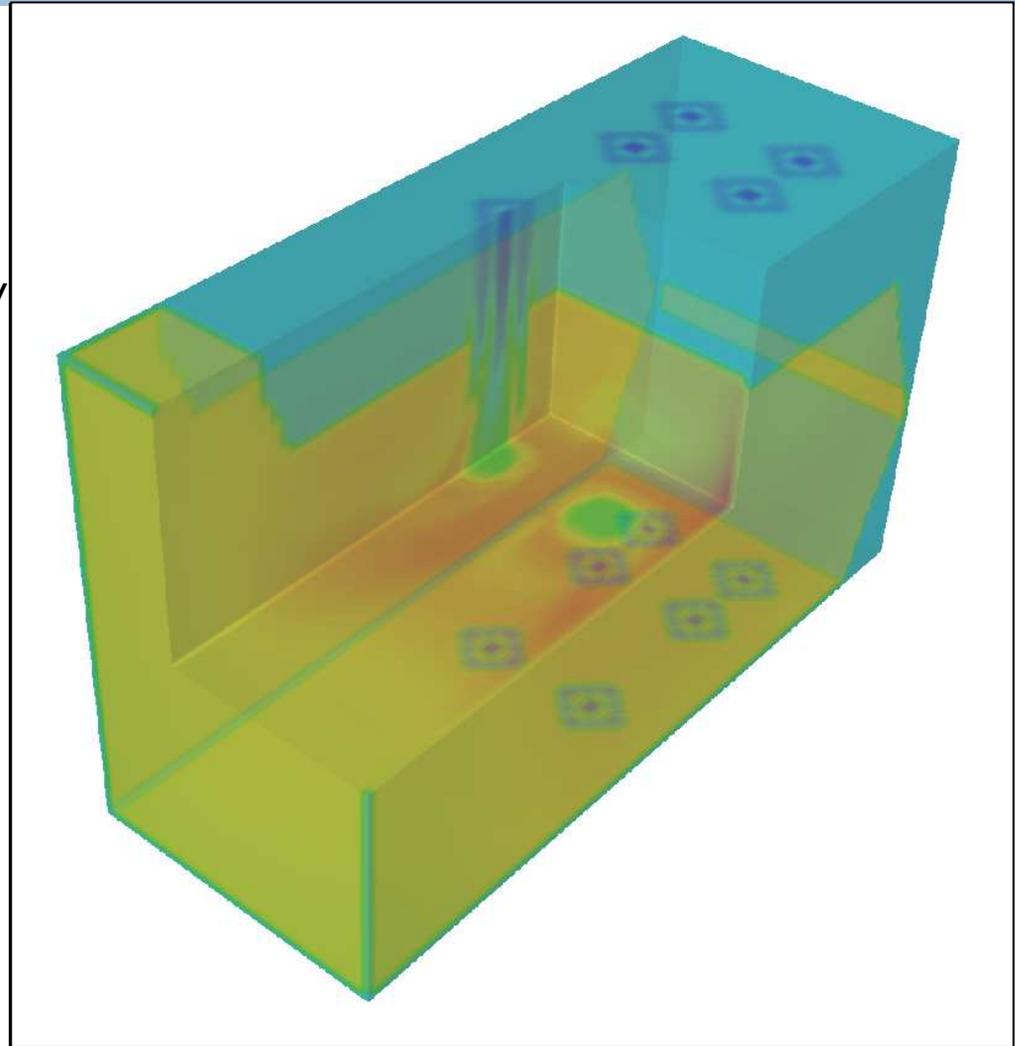
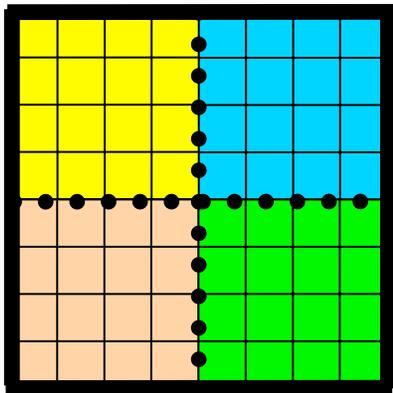
Algorithm *	Processors	Static Load Balancing	Dynamic Load Balancing	Speedup
Slicing	32	3.2s	4.0s	0.8X
Contouring	32	97.2s	65.1s	1.5X
Thresholding	64	181.3s	64.1s	2.8X
Clipping	64	59.0s	30.7s	1.9X



** = All of these operations have no collective communication*

Artifacts occur along the boundaries of domains.

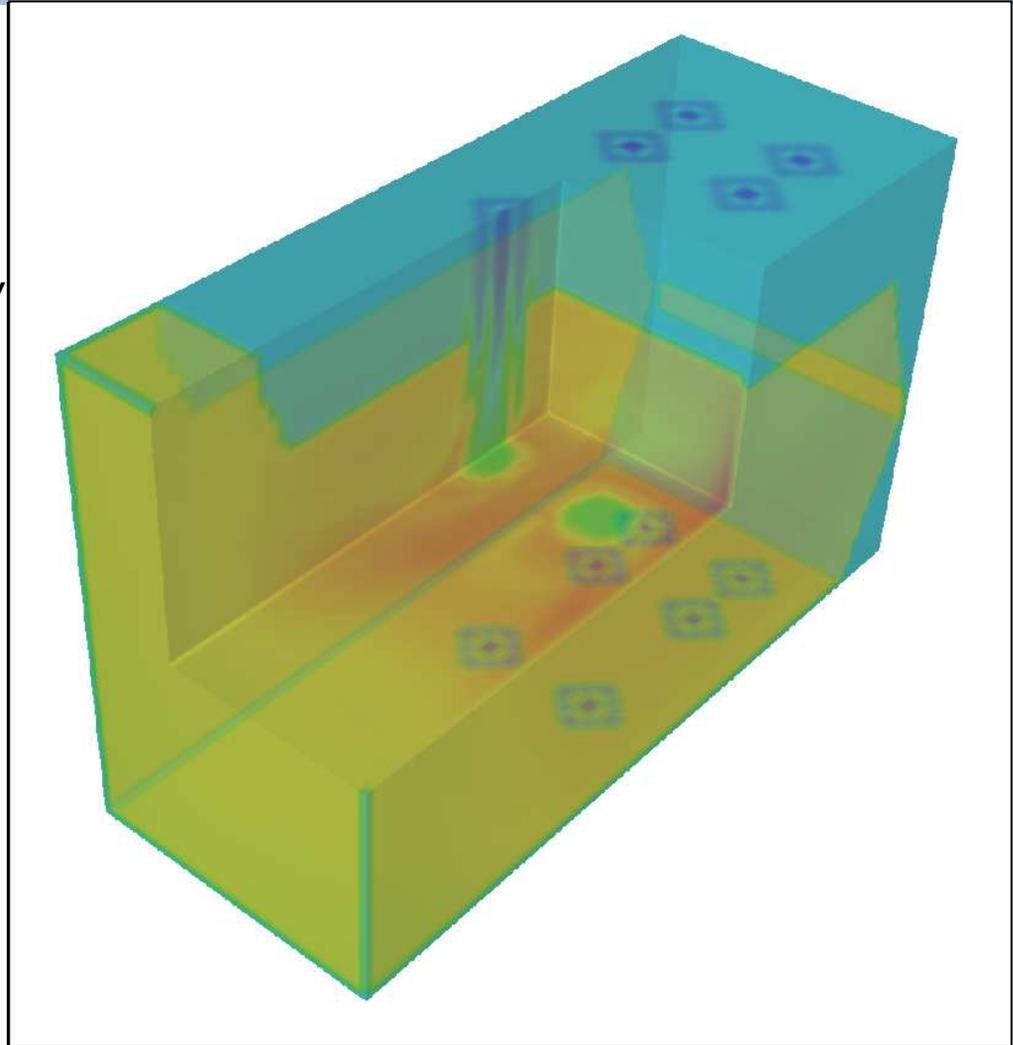
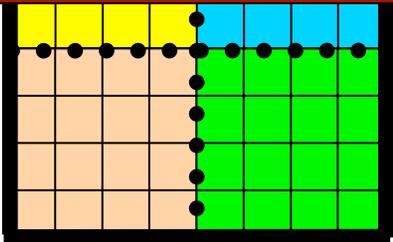
- Looking at external faces
 - ▣ Faces external to a domain can be internal to the data set
 - many extra, unneeded faces
 - wrong picture with transparency



Artifacts occur along the boundaries of domains.

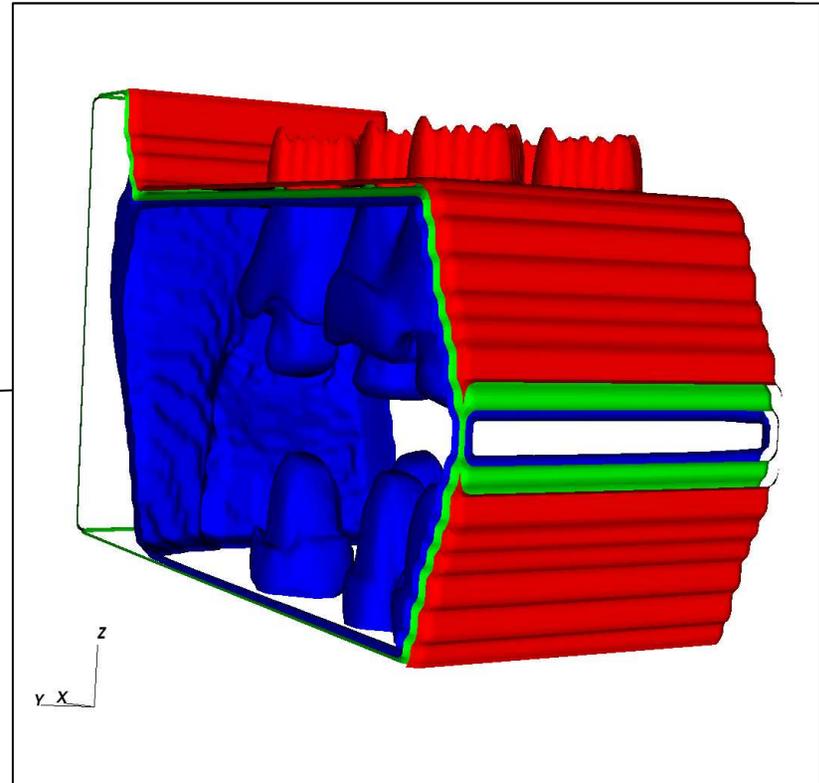
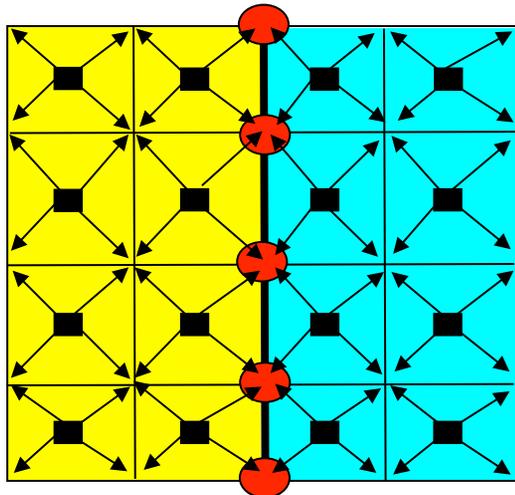
- Looking at external faces
 - ▣ Faces external to a domain can be internal to the data set
 - many extra, unneeded faces
 - wrong picture with transparency

Solution: mark unwanted faces as "ghost"



Artifacts occur along the boundaries of domains.

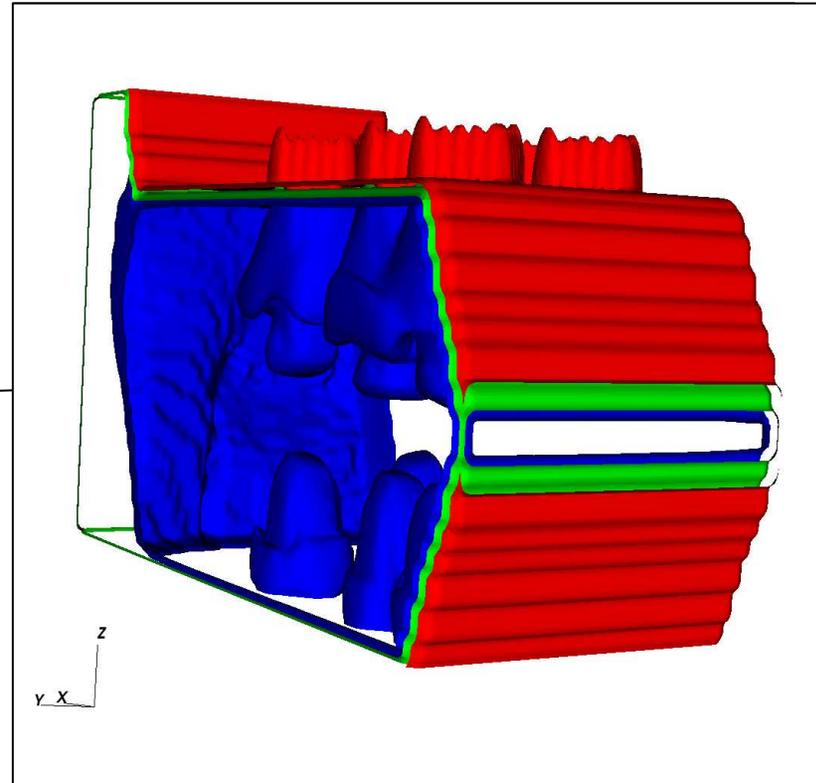
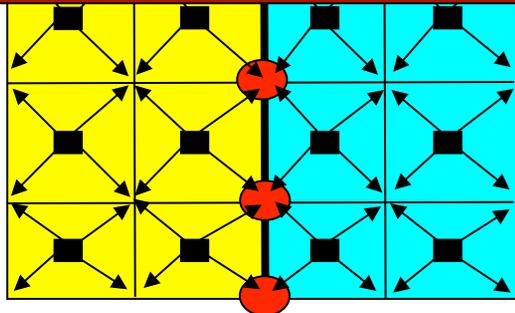
- Interpolation
 - Inconsistent values at nodes along boundary
 - broken contour surfaces



Artifacts occur along the boundaries of domains.

- Interpolation
 - Inconsistent values at nodes along boundary

Solution: make redundant layer of "ghost" elements



Ghost data fixes artifacts along domain boundaries.

- Solution: generate ghost data on the fly
- Through contracts, system determines necessary type of ghost data
- There are different costs for ghost data:
 - Ghost faces: memory
 - Ghost elements: memory, collective communication

Ghost data fixes artifacts along domain boundaries.

- Solution: generate ghost data on the fly
- Through contracts, system determines necessary type of ghost data

*Always get the right picture,
and do it with the minimum cost*

- there are different costs for ghost data:
 - Ghost faces: memory
 - Ghost elements: memory, collective communication

Contracts are a simple idea that have a large impact.

- **Contracts:**
 - ▣ Just a data structure
 - ▣ Describe what impact a component has on the pipeline



Name	Type	Default Value
domains	vector<bool>	all true
hasColl-Commun.	bool	false
ghostType	enum {None, Face, Element}	None
much more...

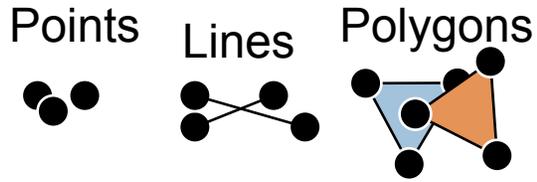
- Contracts enable us to **avoid** the following “dumb” (conservative) strategies:
 - Read all data
 - Always assume collective communication
 - Always create ghost elements

Outline



- Parallel visualization basics
- Smart techniques and data flow
- Contracts
- **Parallel Rendering** ← **courtesy Ken Moreland**
- IceT
- Performance study

The Graphics Pipeline



Rendering Hardware

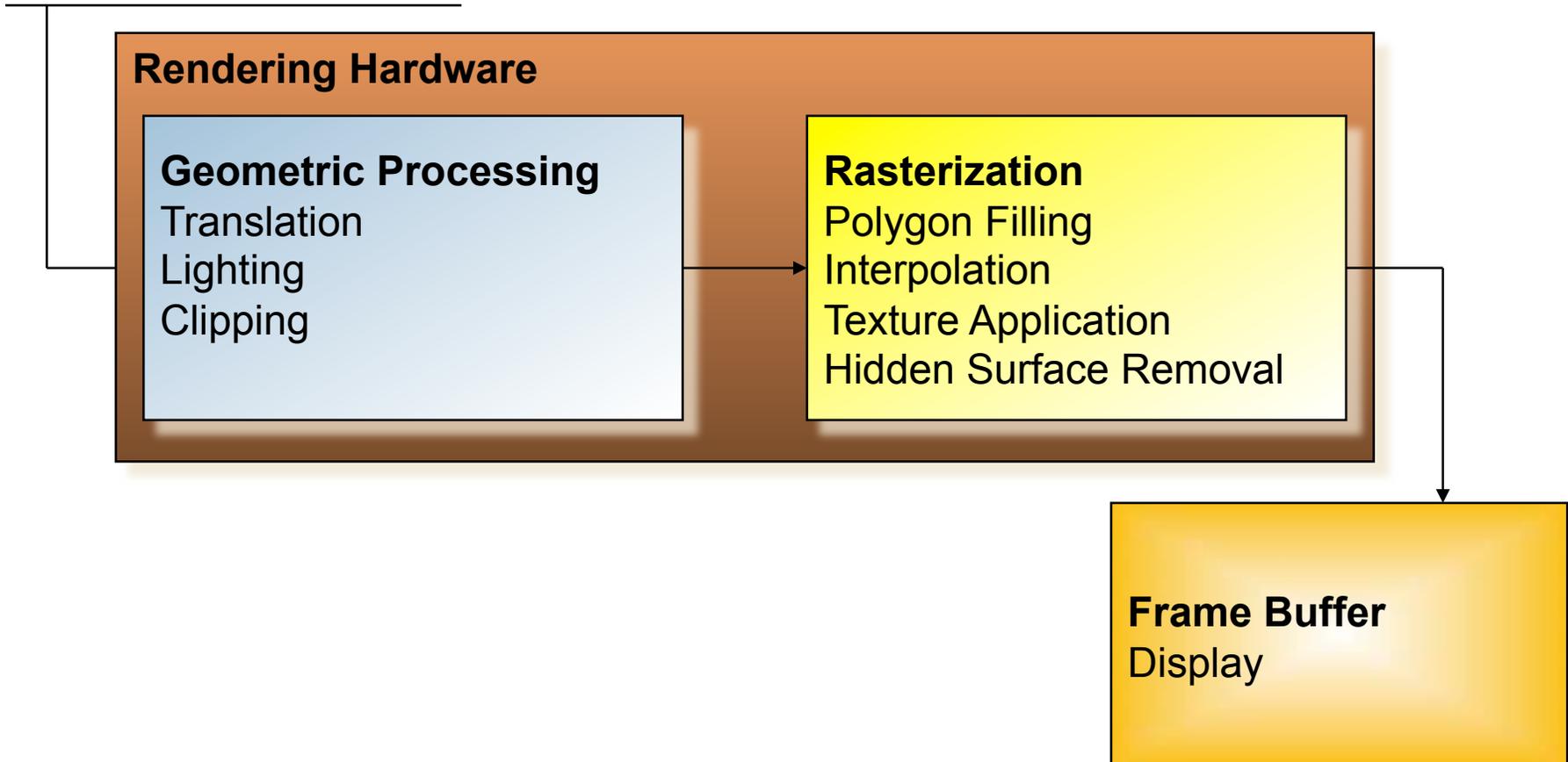
Geometric Processing

Translation
Lighting
Clipping

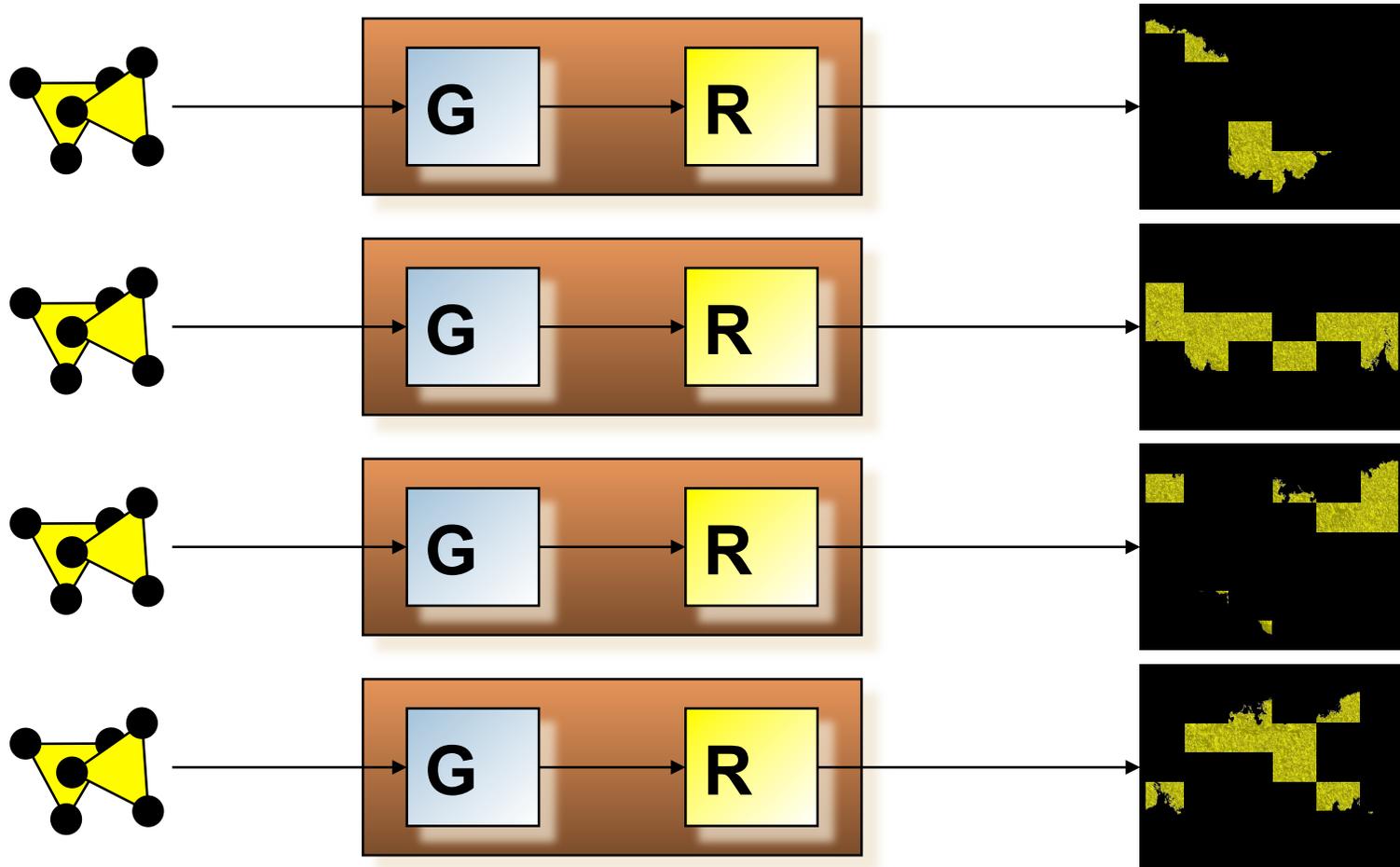
Rasterization

Polygon Filling
Interpolation
Texture Application
Hidden Surface Removal

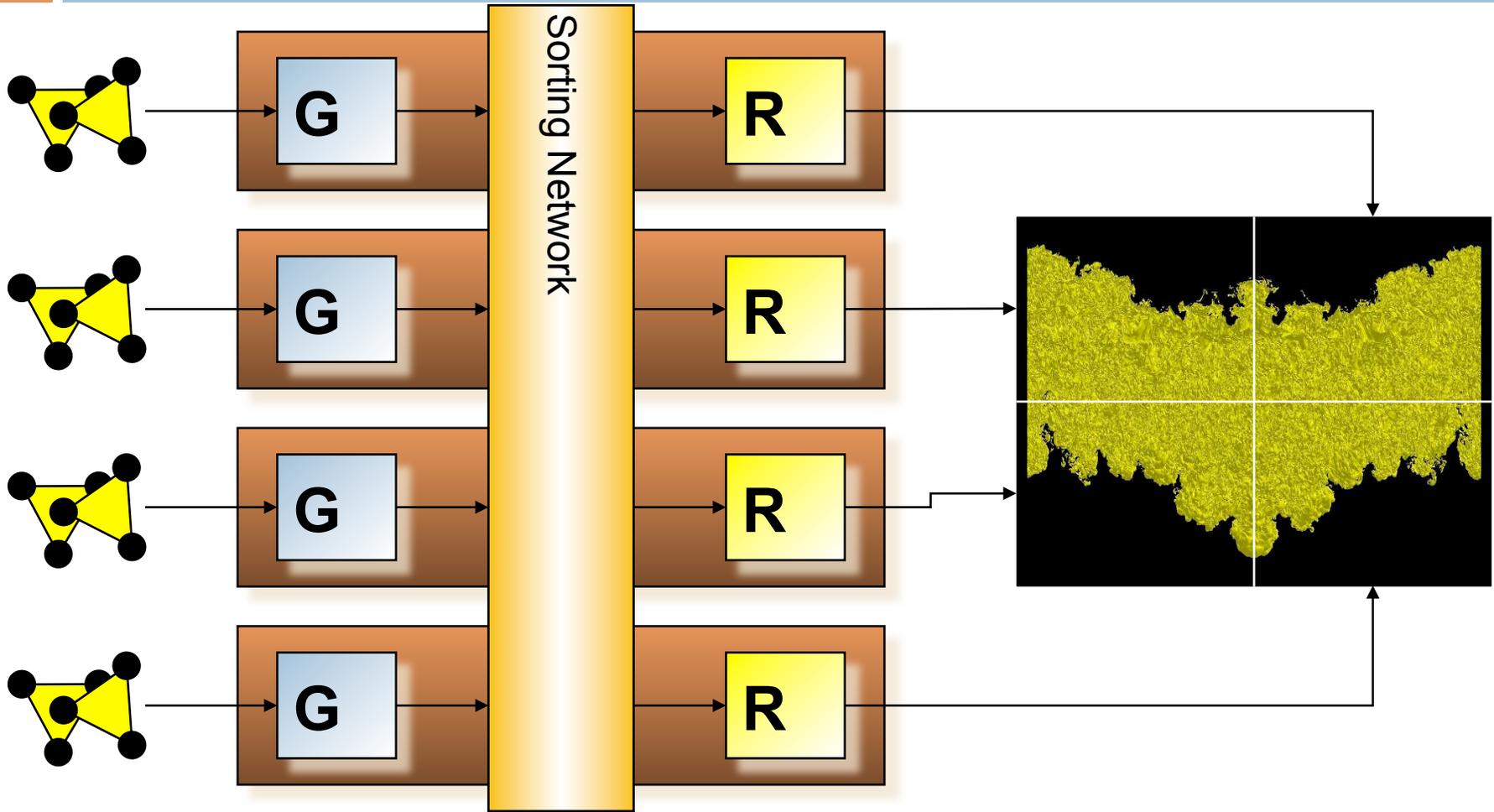
Frame Buffer
Display



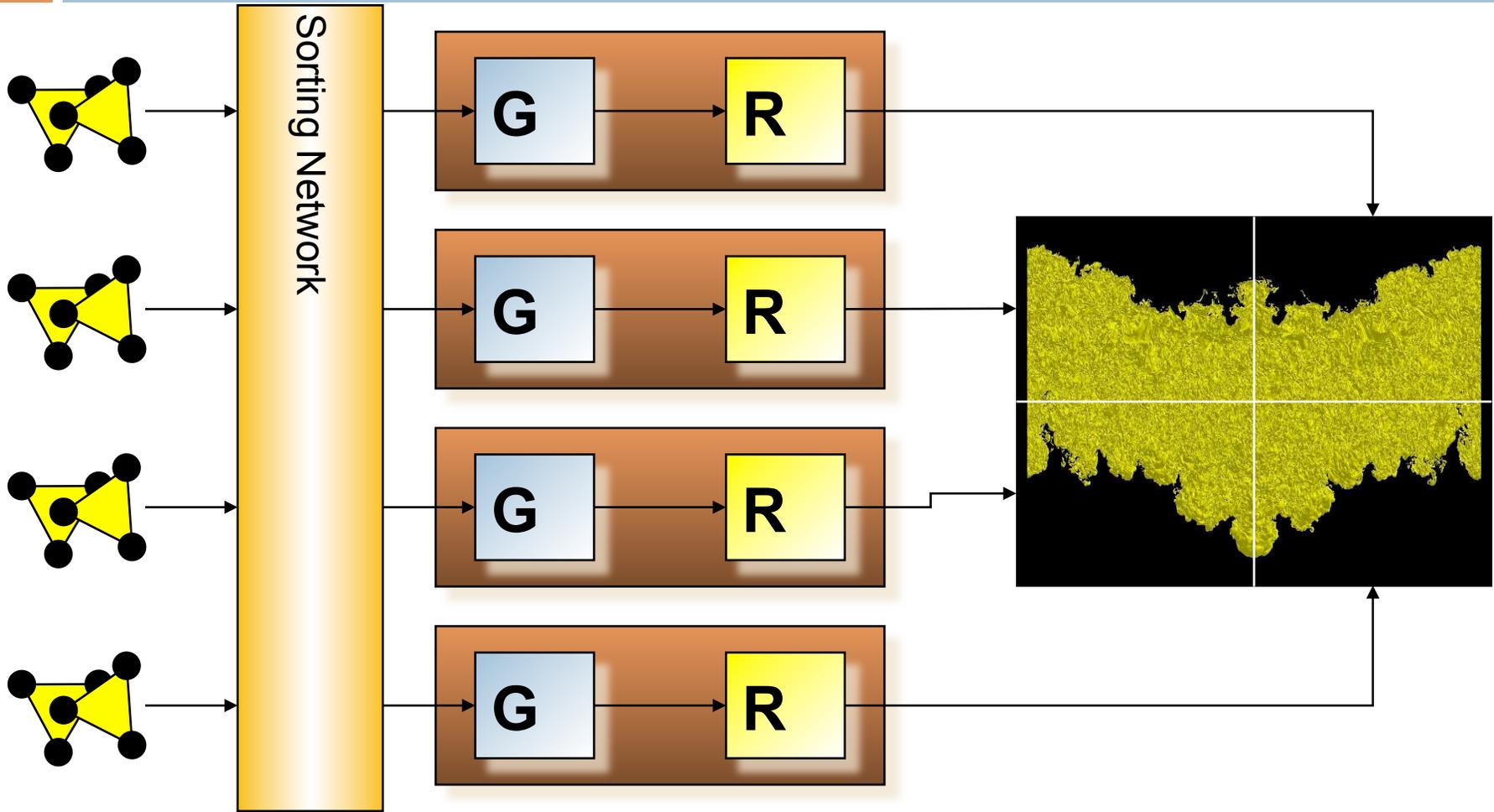
Parallel Graphics Pipelines



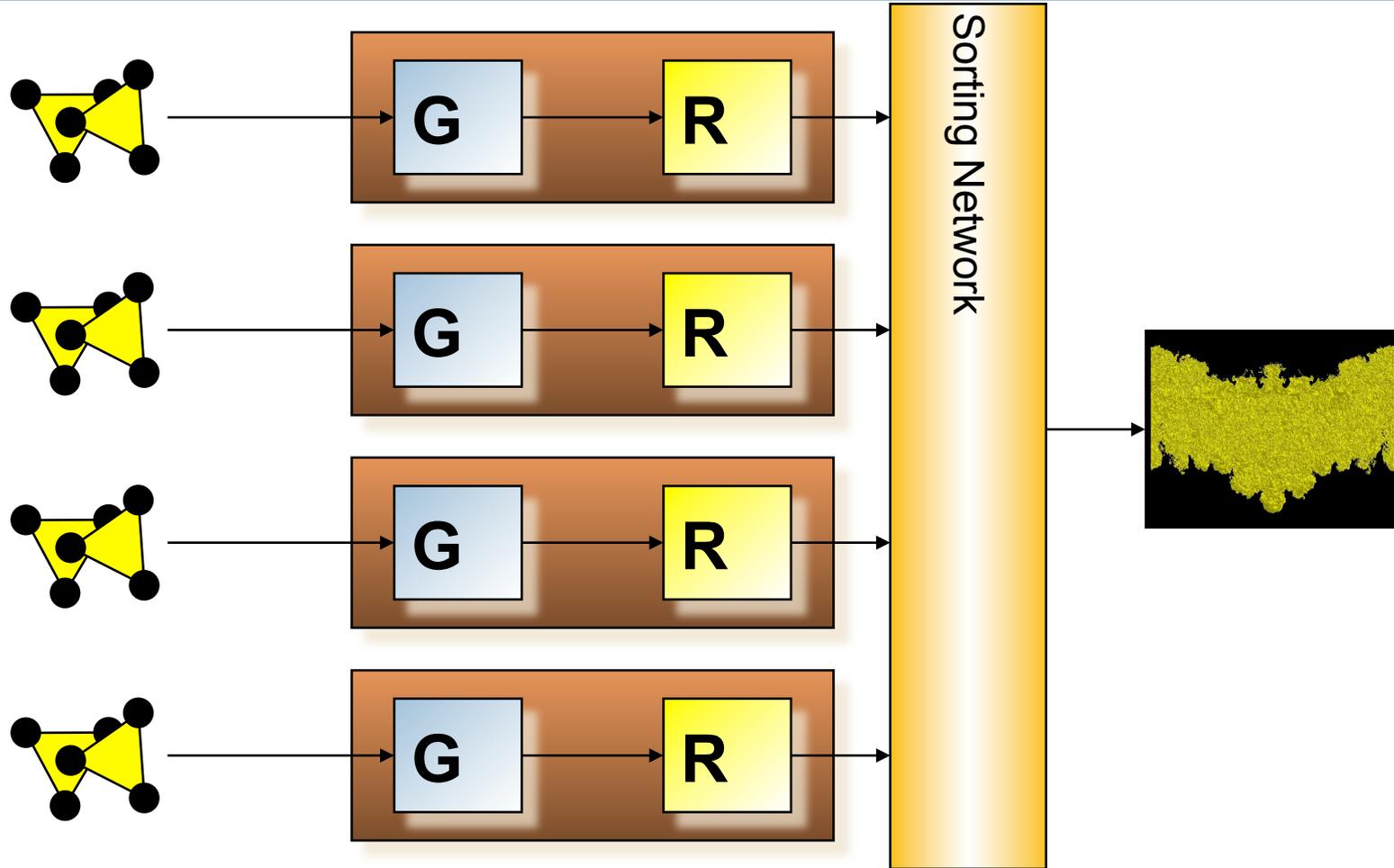
Sort Middle Parallel Rendering



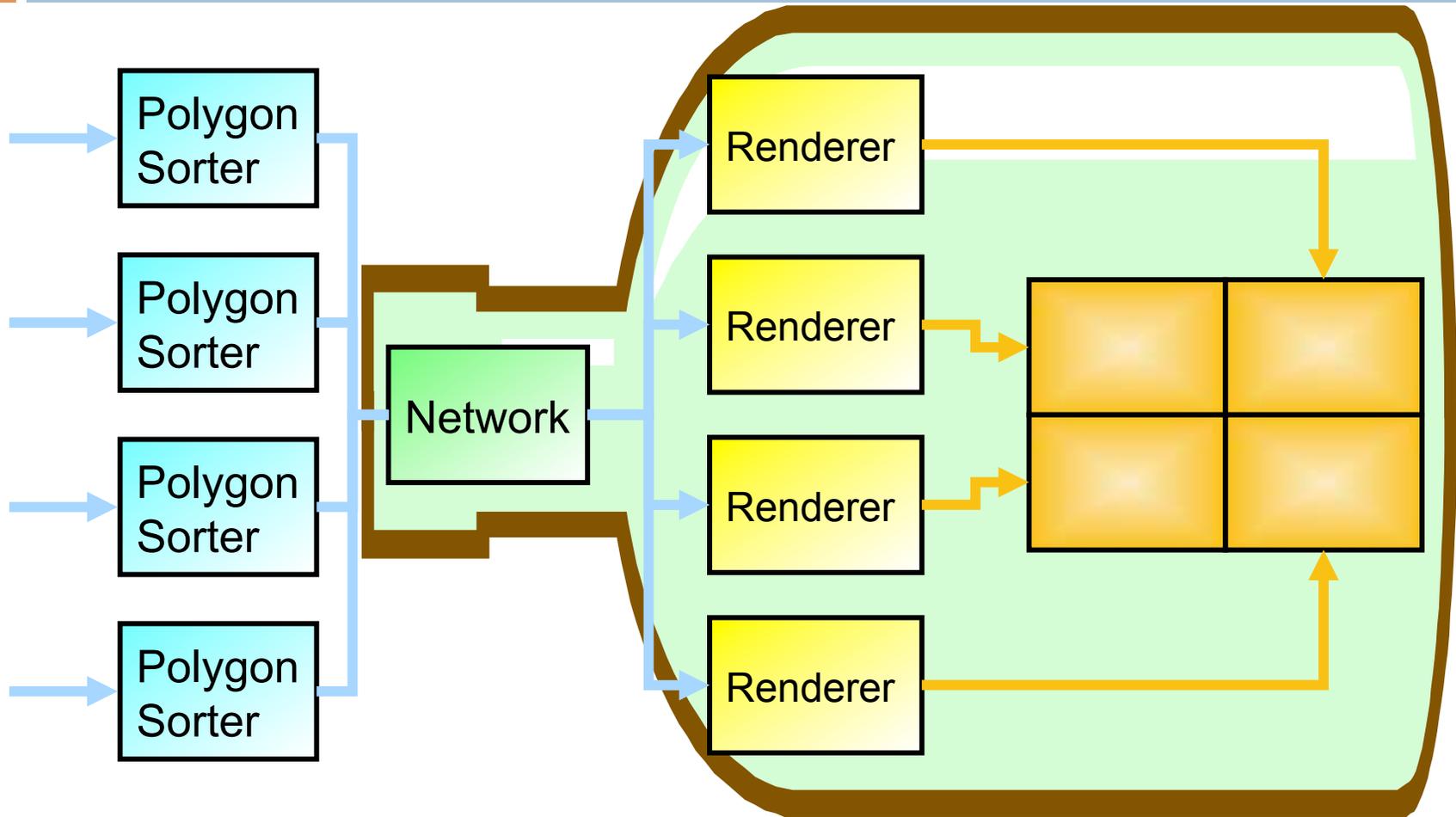
Sort First Parallel Rendering



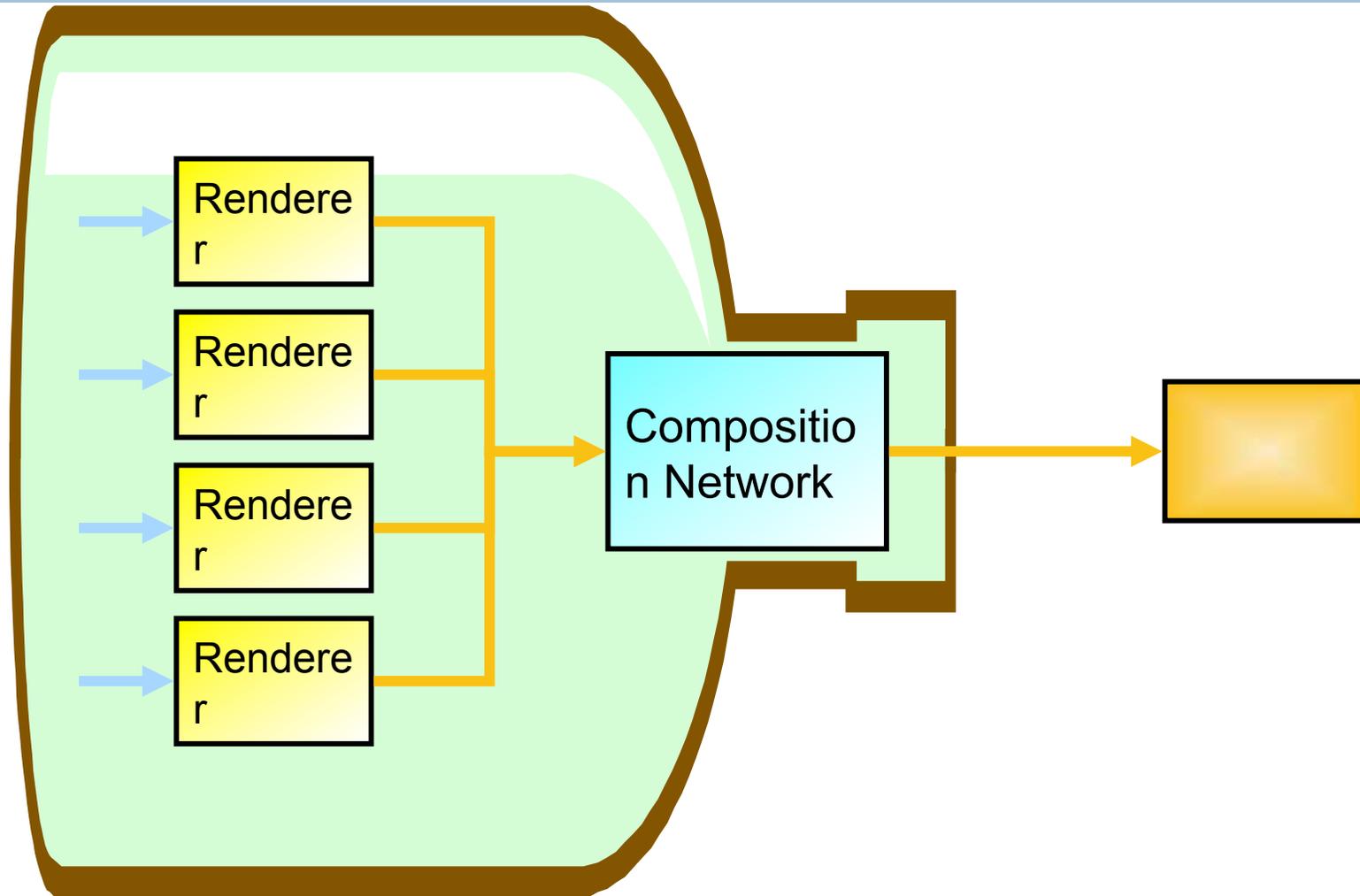
Sort Last Parallel Rendering



Sort-First Bottleneck



Sort-Last Bottleneck



Outline



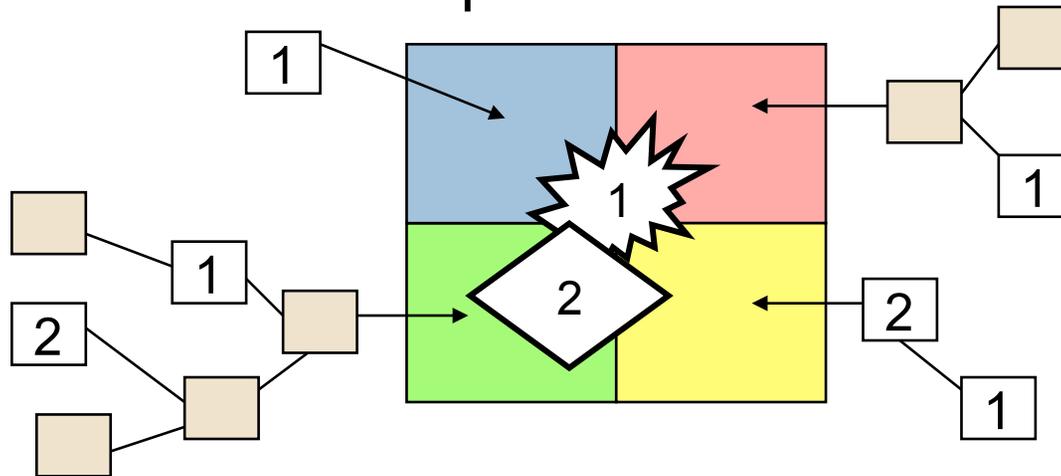
- Parallel visualization basics
- Smart techniques and data flow
- Contracts
- Parallel Rendering
- IceT ← courtesy Ken Moreland!
- Performance study

IceT: parallel rendering library

- IceT = Image Compositing Engine for Tiles
- Developed and maintained by Sandia Nat'l Lab.
- Popular library, used by VisIt and ParaView
- Used most often for non-tiled displays.
 - ▣ Heavily optimized
 - “Active pixels”
 - ▣ Employs multiple strategies based on user settings, problem setup
- <http://icet.sandia.gov/>

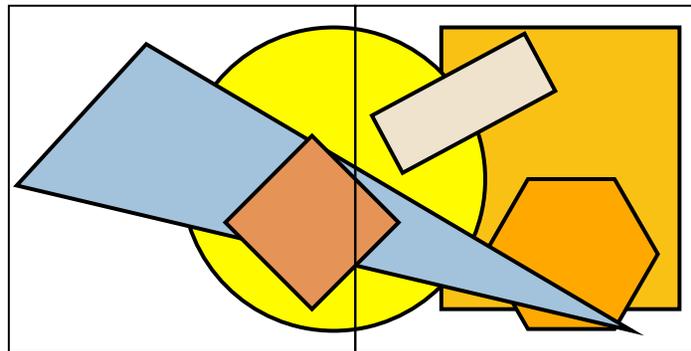
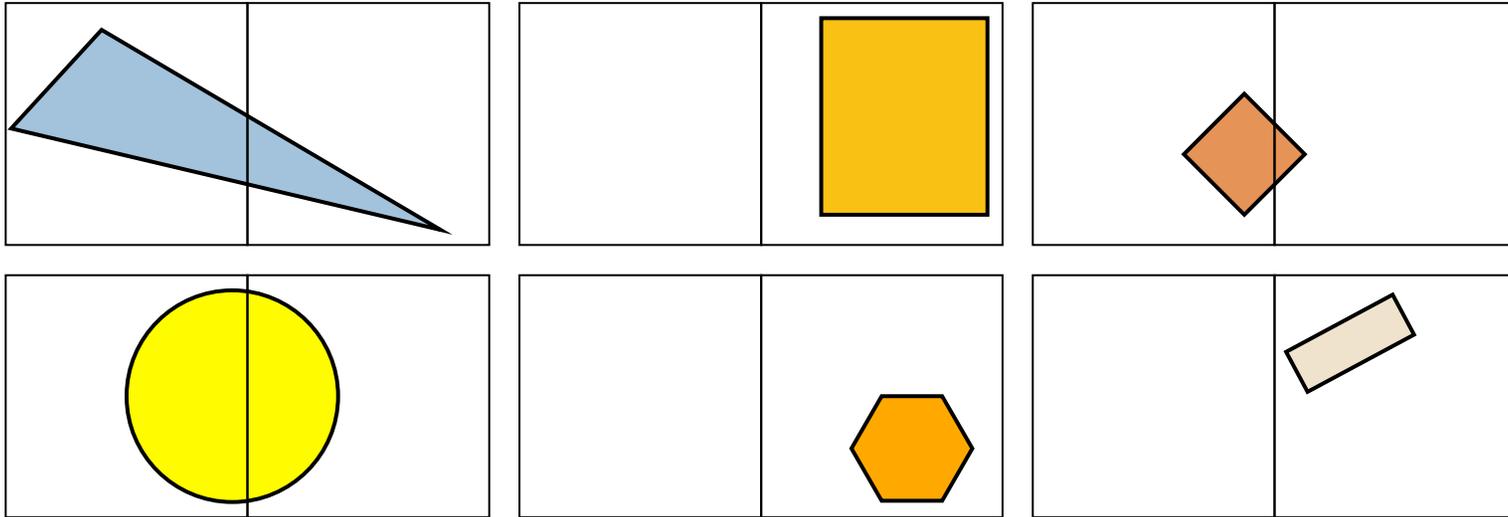
Image Composition Engine for Tiles (ICE-T)

- Render/compose individual tiles.
- Take advantage of spatial decomposition.
 - ▣ Throw away blank images.
 - ▣ Build “virtual” composition networks.

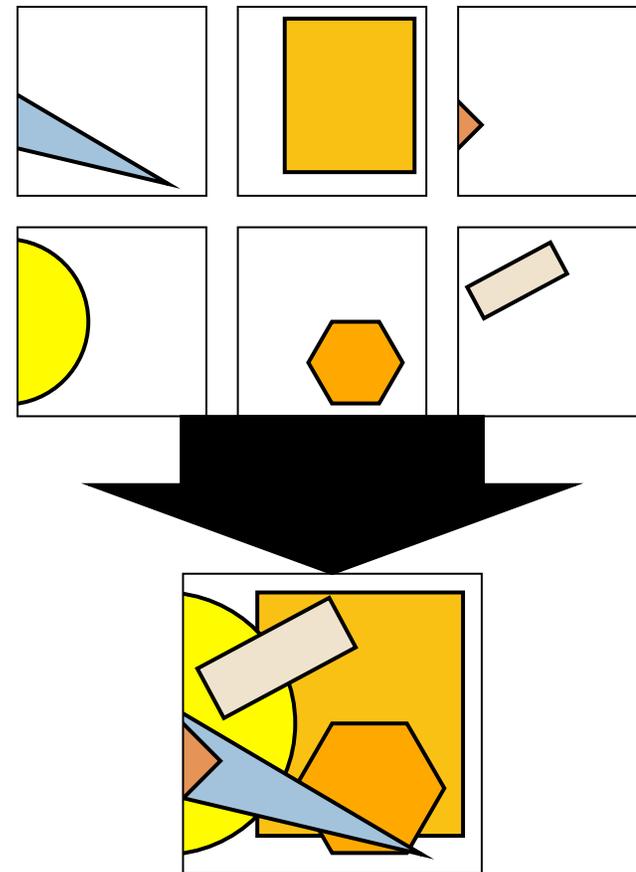
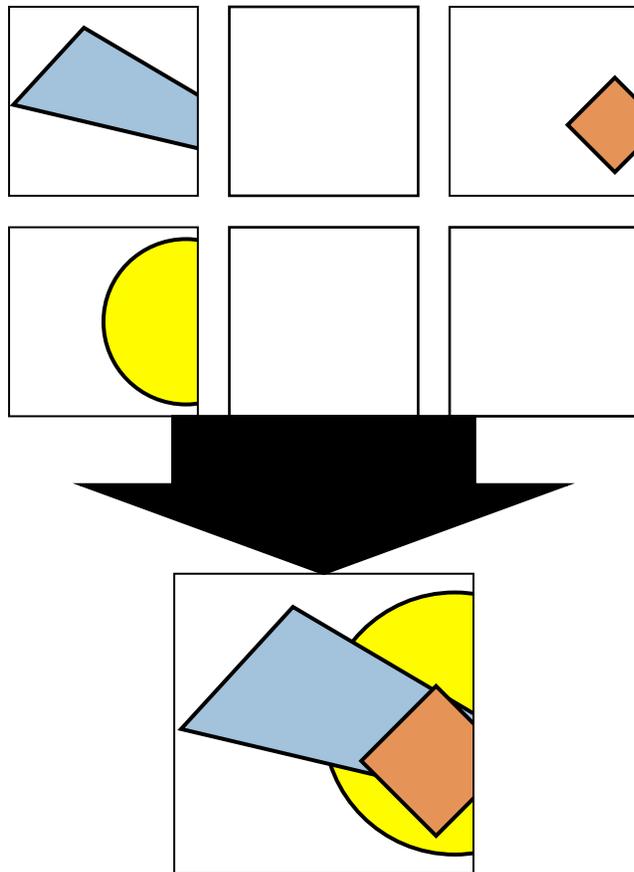


- Challenge: perform all compositions in parallel and maintain good load balancing.

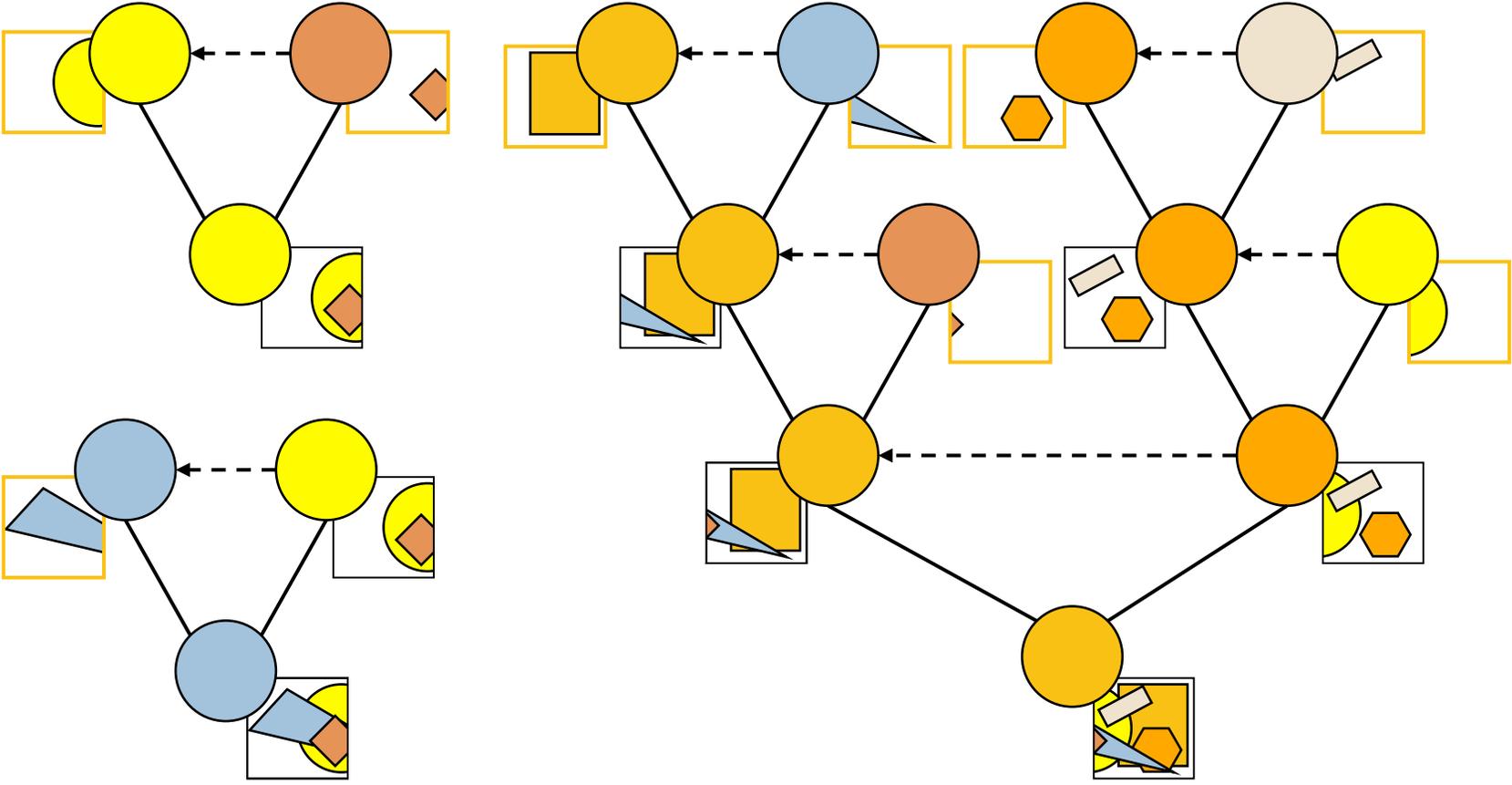
Example: 6 Nodes, 2 Tiles



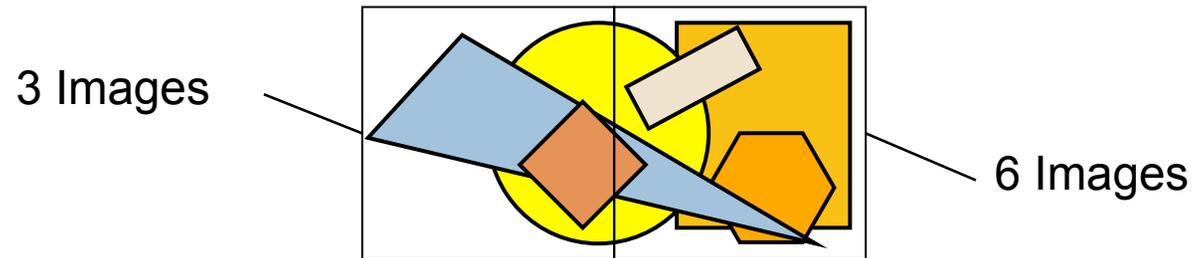
Serial Strategy



Virtual Trees Strategy



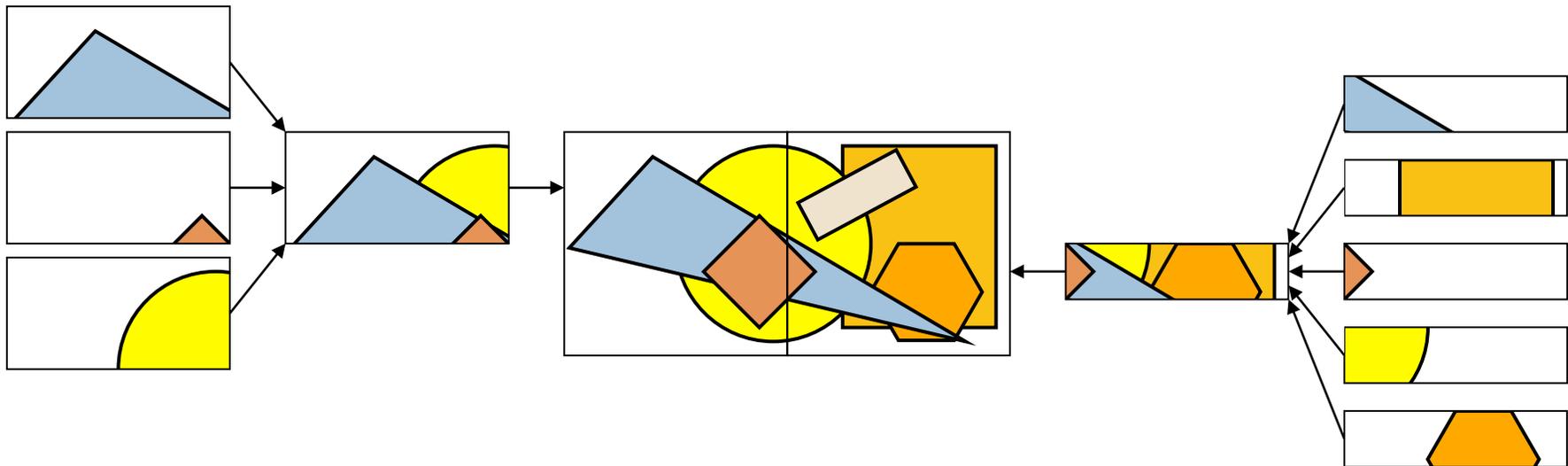
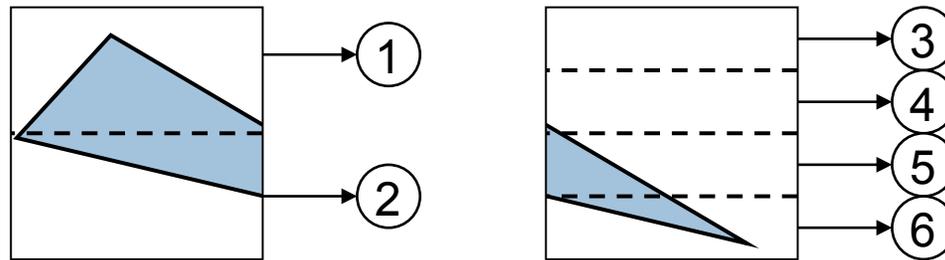
Tile Split and Delegate Strategy



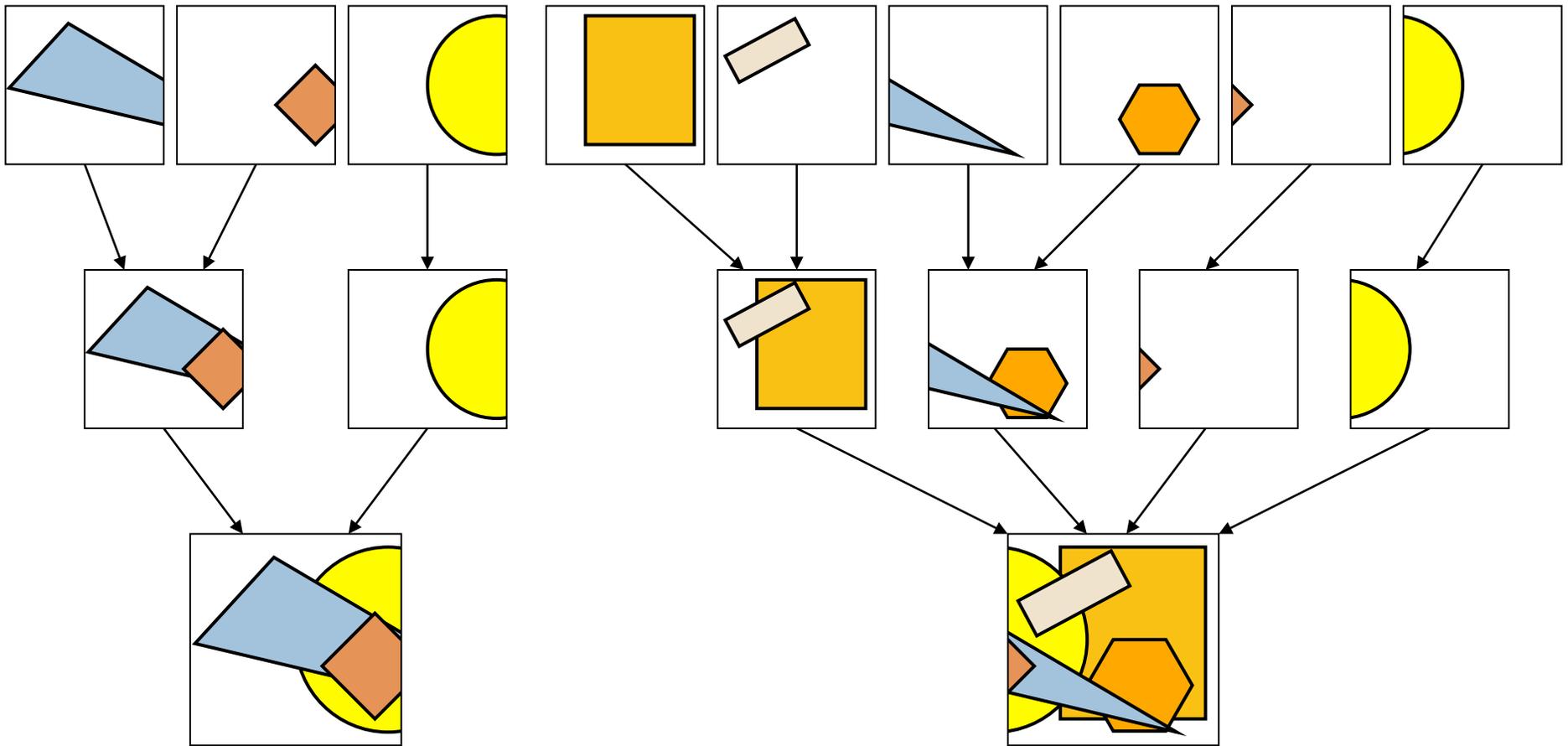
Assign Processors 1, 2	Assign Processors 3, 4, 5, 6
------------------------------	------------------------------------

1	3
	4
2	5
	6

Tile Split and Delegate Strategy

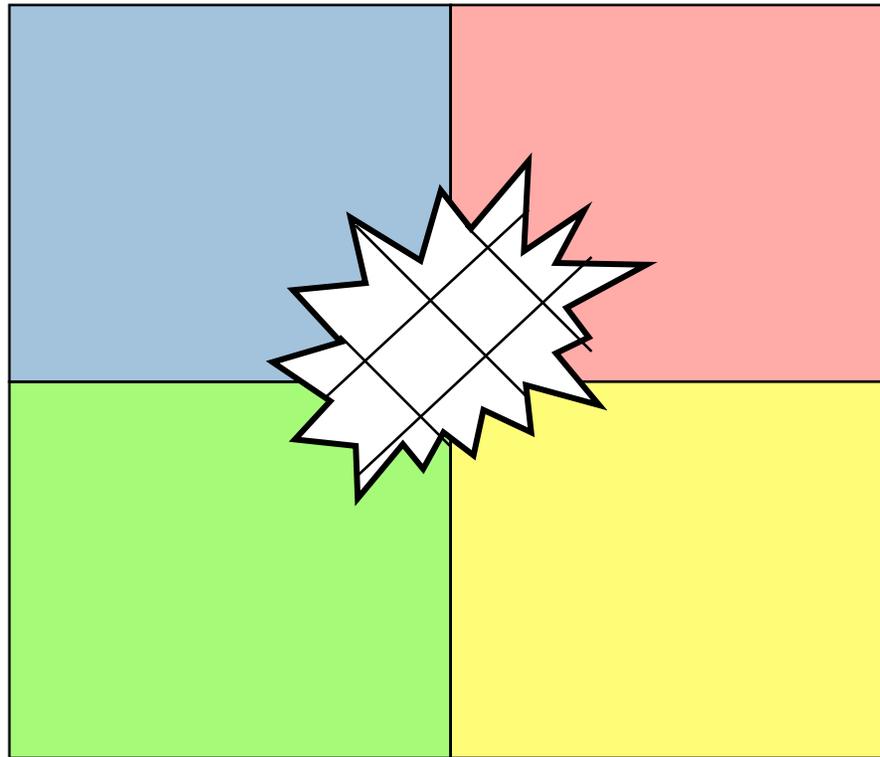


Reduce to Single Tile Strategy



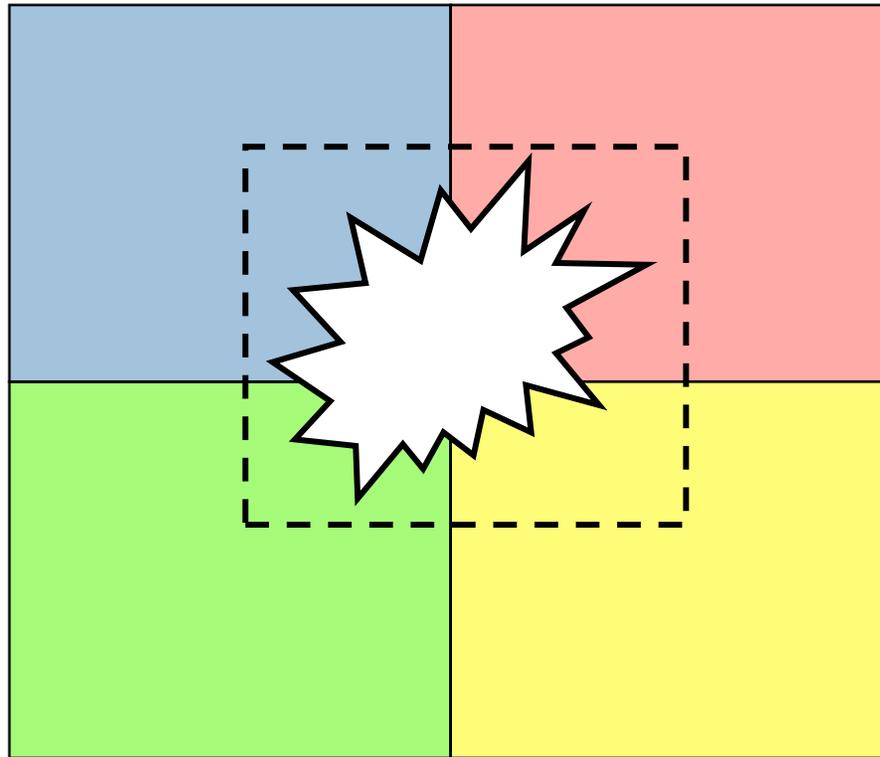
Bucketing

Using buckets to reduce number of polygons re-rendered.



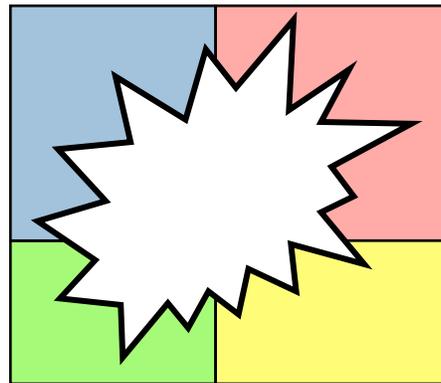
Floating Viewport

Object fits within a tile, but is translated so that it straddles up to four tiles.



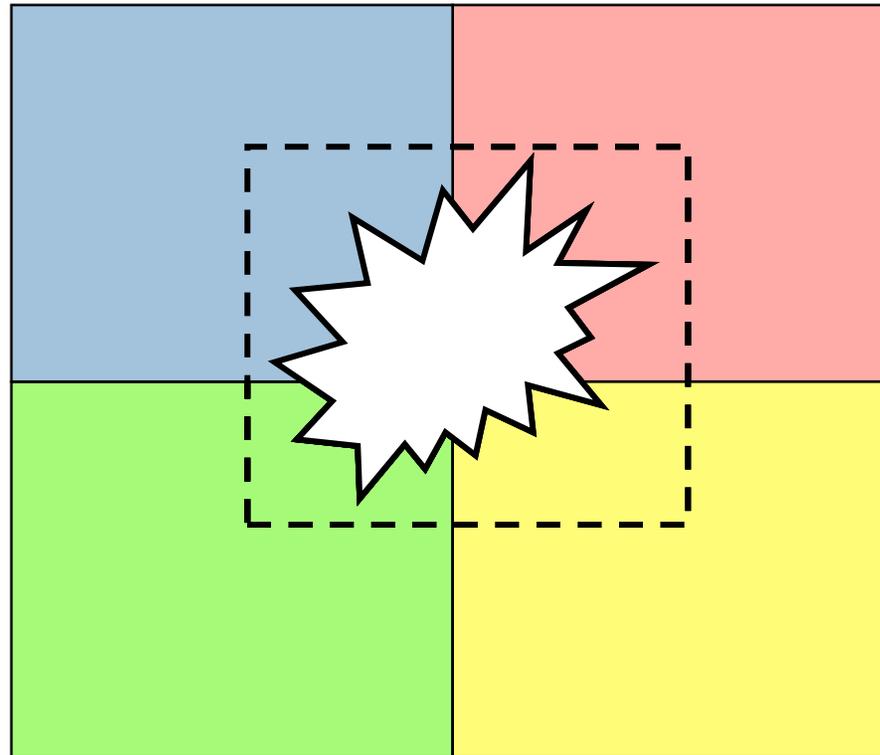
Floating Viewport

Rather than render four times, render once in a viewport that completely contains the object.



Floating Viewport

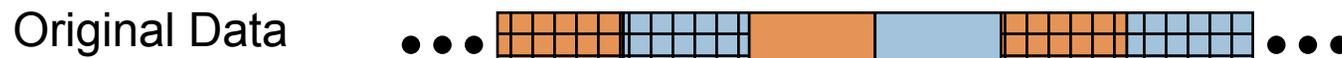
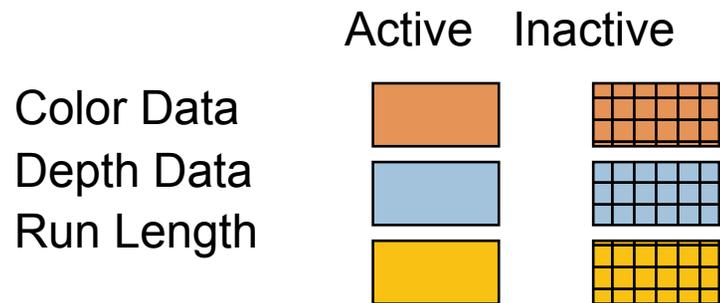
Break the image into four pieces and pad each piece to form an image for each tile.



Active Pixel Encoding

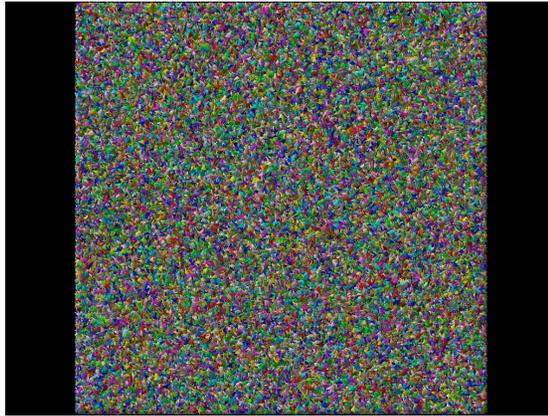
- Record run lengths of “active” pixels and “inactive” pixels.
- Throw away data for “inactive” pixels.
- Fast encoding.
 - ▣ Three operations per pixel.
- Free decoding. Faster depth compare.
- Effective compression.
 - ▣ Encoded $1/5$ full image at beginning.
- Good worst case behavior.
 - ▣ Encoded image can only grow a few bytes.

Active Pixel Encoding, Worst Case

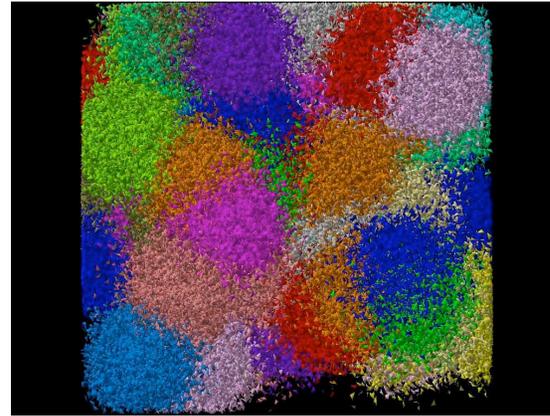


Test Data Distributions

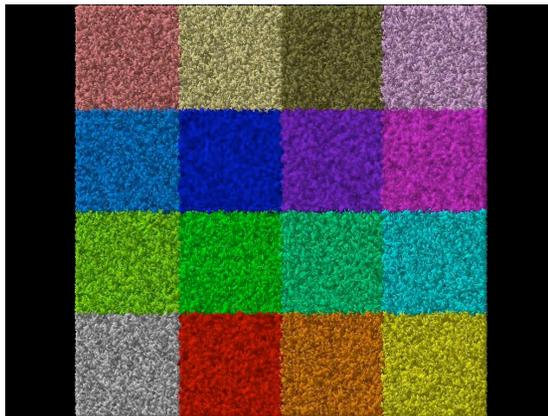
Linear
Distribution



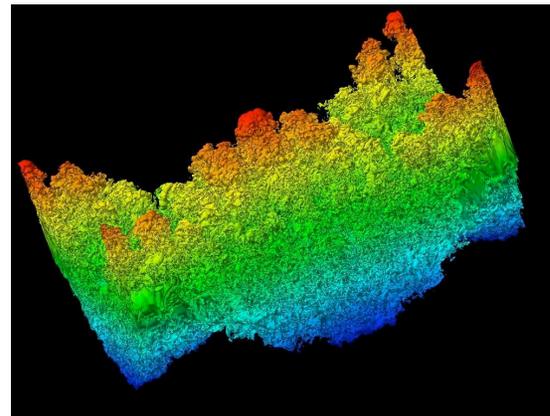
Gaussian
Distribution



Perfect
Separation



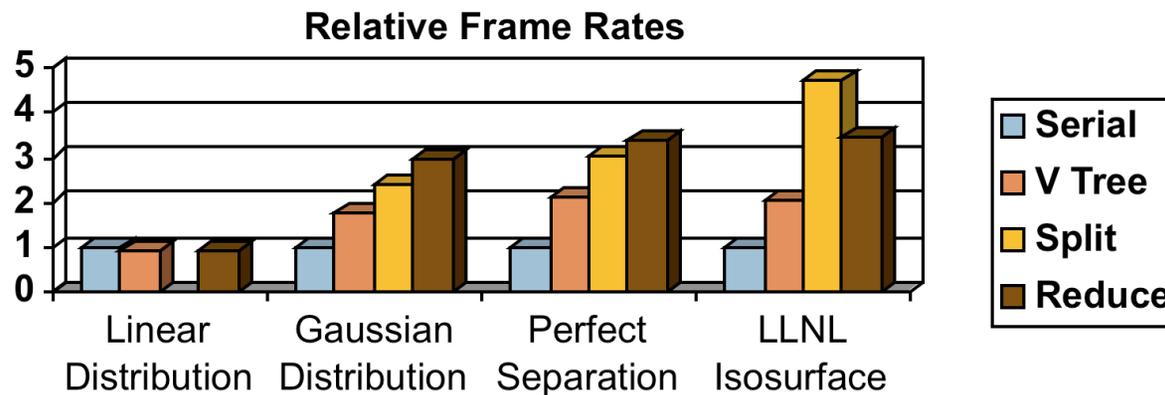
469 Mtri
Isosurface*



*Image covered by Lawrence Livermore National Laboratories: UCRL-MI-142527 Rev 1

Experimental Results

		Data Set			
		Linear Distribution	Gaussian Distribution	Perfect Separation	LLNL Isosurface
Strategy	Serial	0.269 Hz	0.370 Hz	0.309 Hz	0.076 Hz
	V Tree	0.241 Hz	0.652 Hz	0.657 Hz	0.155 Hz
	Split		0.888 Hz	0.930 Hz	0.361 Hz
	Reduce	0.247 Hz	1.093 Hz	1.040 Hz	0.262 Hz



IceT: Conclusions

□ Pluses

- Renders extremely large sets of polygons at fast rates.
- Good performance on tile displays.
- Runs on clusters (\$\$\$).
- Good scalability.
- Maintains good load balancing with unstructured data.

□ Minuses

- Slow frame rates.
 - Large constant overhead.
 - Frame buffer read back a huge bottleneck.
- Requires multi-pass rendering.
- Relies on spatial decomposition for good performance.

Outline



- Parallel visualization basics
- Smart techniques and data flow
- Contracts
- Parallel Rendering
- IceT
- Performance study

Pure parallelism and tomorrow's data

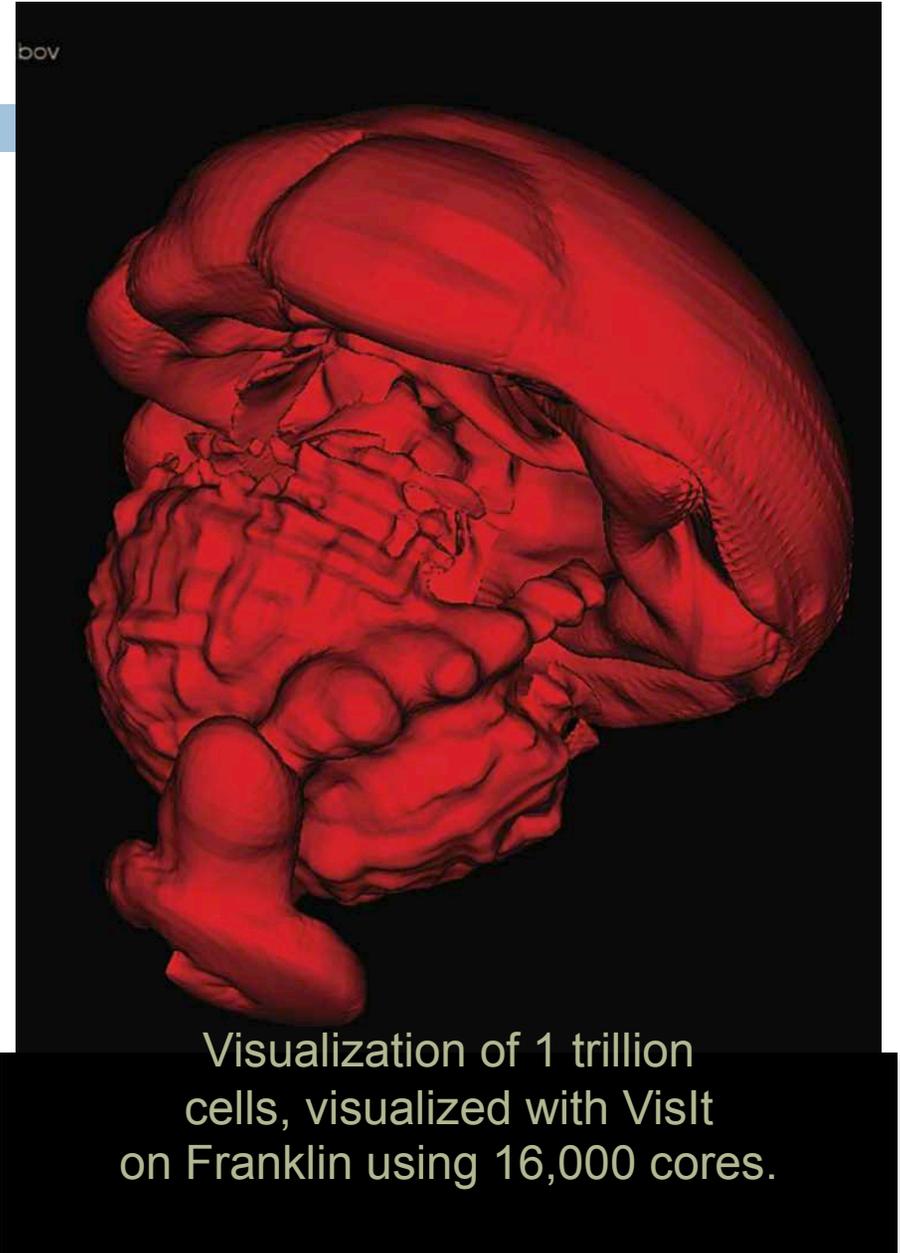
- Research questions:

- Is it possible/feasible to run production-quality visual data analysis s/w on large machines and on large data sets?
 - Are the tools we use right now ready for tomorrow's data?
- What obstacles/bottlenecks do we encounter at massive data?

Experiment methodology

- Preprocess step: generate large data set
- Read it
- Contour
- Render @ 1024x1024

- Synthetic data:
 - Wanted to look at tomorrow's data; not available yet
 - Synthetic data should be reasonable surrogate for real data.



Experiment methodology, continued

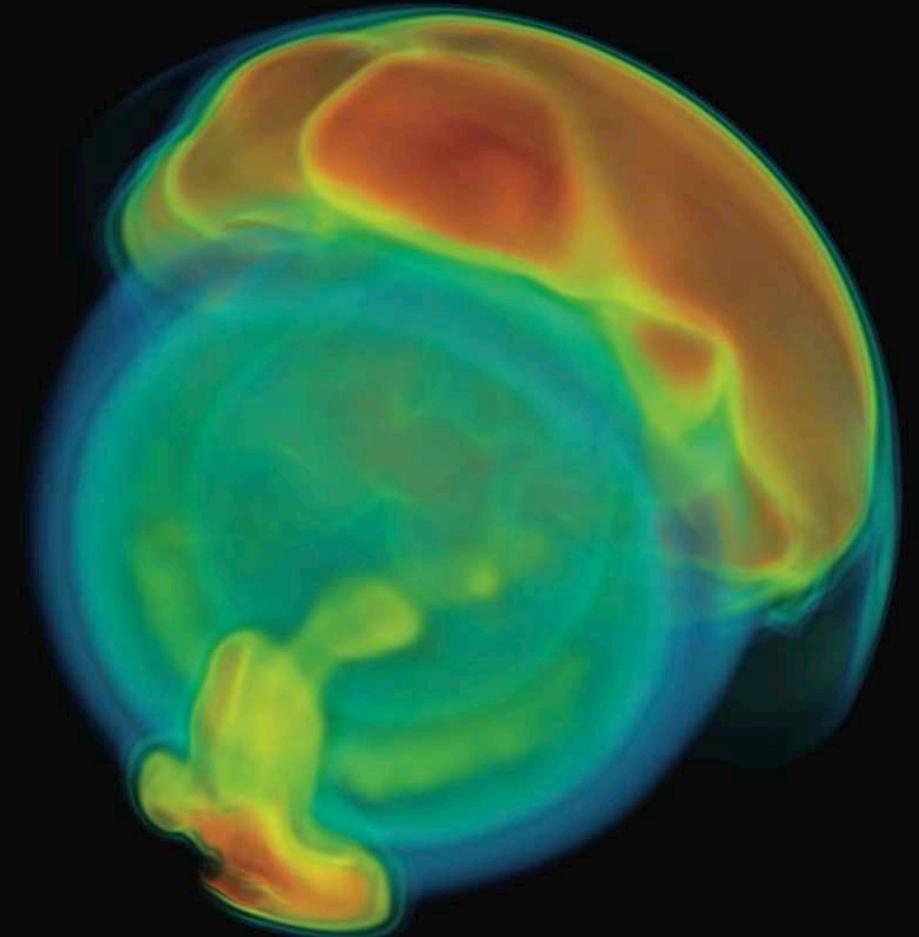


- Only used pure parallelism
 - ▣ This experiment was about testing the limits of pure parallelism
 - ▣ Purposely did not use in situ, multi-resolution, out-of-core, data subsetting
- Pure parallelism is what the production visualization tools use right now (*).

Volume rendering

- Ran into problems with volume rendering.
- Problem eventually fixed, but not in time for study
 - ▣ Runs on these big machines are opportunistic and it's hard to get a second chance
 - ▣ Approximately five seconds per render
- Contouring exercises much of the infrastructure (read, process, render)

TZ.bov



Visualization of 2 trillion cells, visualized with VisIt on JaguarPF using 32,000 cores.

Experiment methodology, continued



- Three basic variations
 - Vary over supercomputing environment
 - Vary over data generation
 - Vary over I/O pattern

Varying over supercomputer environment

- **Goals:**
 - Ensure results aren't tied to a single machine.
 - Understand differences from different architectures.
- **Experiment details**
 - 1 trillion cells per 16,000 cores
 - $10 \times N_{\text{Cores}}$ "Brick-of-float" files, gzipped
 - Upsampled data

Machine name	Machine type or OS	Total no. of cores	Memory per core (Gbytes)	System type	Clock speed	Peak flops	Top 500 rank (as of Nov. 2009)
JaguarPF	Cray	224,162	2.0	XT5	2.6 GHz	2.33 Pflops	1
Ranger	Sun Linux	62,976	2.0	Opteron Quad	2.0 GHz	503.8 Tflops	9
Dawn	Blue Gene/P	147,456	1.0	PowerPC	850.0 MHz	415.7 Tflops	11
Franklin	Cray	38,128	1.0	XT4	2.6 GHz	352 Tflops	15
Juno	Commodity (Linux)	18,402	2.0	Opteron Quad	2.2 GHz	131.6 Tflops	27
Purple	AIX (Advanced Interactive Executive)	12,208	3.5	Power5	1.9 GHz	92.8 Tflops	66

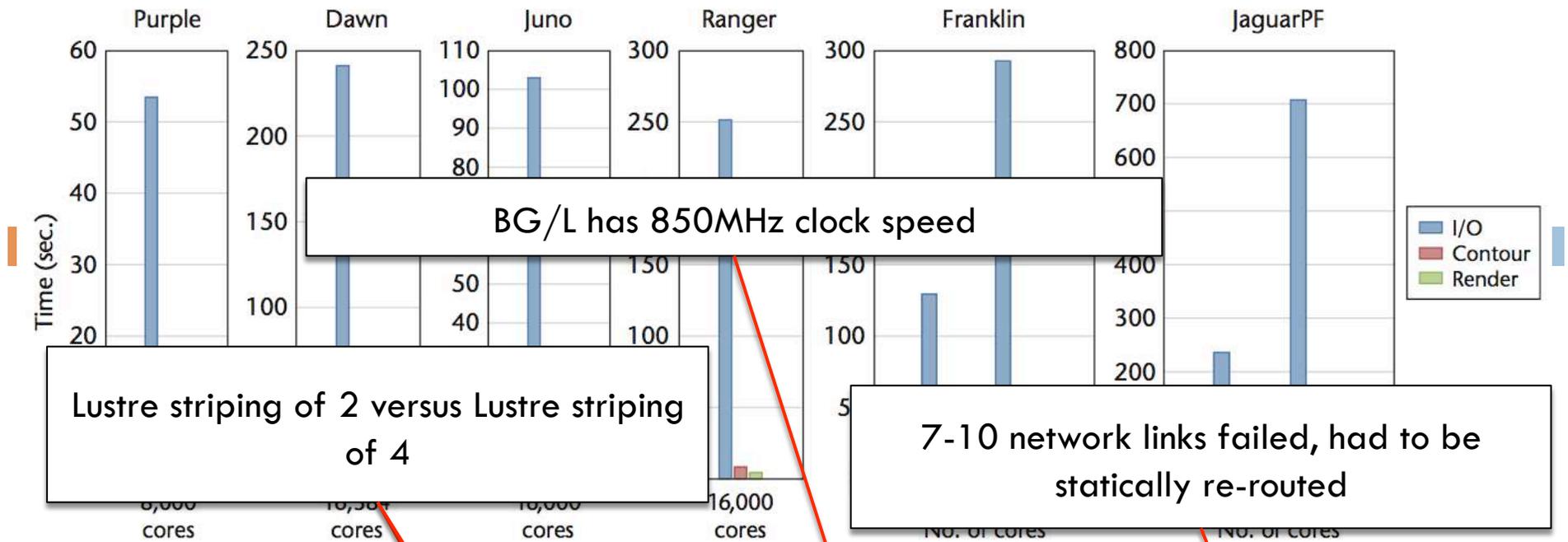


Figure 2. Runtimes for I/O, contouring, and rendering. These results show that, although there is variation across the supercomputers, I/O is the slowest phase.

Table 2. Performance across diverse architectures.

Machine	No. of cores	Data set size (TCells)	Total I/O time (sec.)	Contour time (sec.)	Total pipeline execution time (sec.) [†]	Rendering time (sec.)
Purple	8,000	0.5	53.4	10.0	63.7	2.9
Dawn	16,384*	1.0	240.9	32.4	277.6	10.6
Juno	16,000	1.0	102.9	7.2	110.4	10.4
Ranger	16,000	1.0	251.2	8.3	259.7	4.4
Franklin	16,000	1.0	129.3	7.9	137.3	1.6
JaguarPF	16,000	1.0	236.1	10.4	246.7	1.5
Franklin	32,000	2.0	292.4	8.0	300.6	9.7
JaguarPF	32,000	2.0	707.2	7.7	715.2	1.5

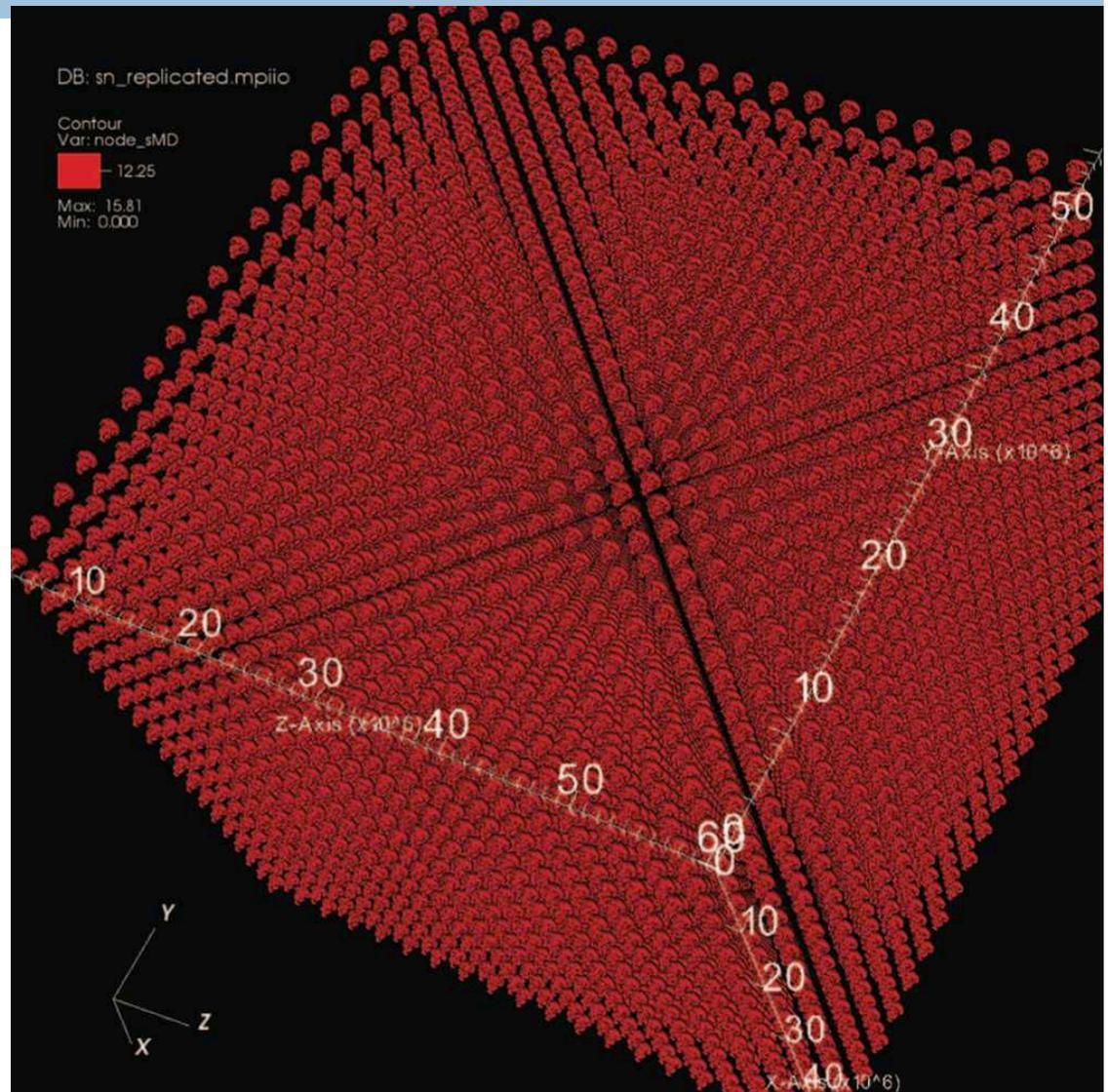
* Dawn requires that the number of cores be a power of two.

† This measure indicates the time to produce the surface.

Varying over data generation pattern

- Concern: does upsampling produce unrepresentatively smooth surfaces?
- Alternative: replication

Visualization of 1 trillion cells, visualized with VisIt on Franklin using 16,000 cores.



Results from data generation test

- Test on franklin, using 16,000 cores with unzipped data

Data generation	Total I/O time (sec.)	Contour time (sec.)	Total pipeline execution time (sec.)	Rendering time (sec.)
Upsampled	478.3	7.6	486.0	2.8
Replicated	493.0	7.6	500.7	4.9

Contouring time is the same because case where a triangle is generated is rare.

Rendering time is different because replicated pattern has more geometry.

Varying over I/O pattern

- Previous tests: uncoordinated I/O, doing 10 “fread”s per core.
- Can collective communication help?

I/O pattern	No. of cores	Data set size (TCells)	Total I/O time (sec.)	Data read (Gbytes)	Read bandwidth (Gbytes per second)
Collective	16,016	1	478.3	3,725.3	7.8
Noncollective	16,000	1	129.3	954.2	7.4

Franklin I/O maximum: 12GB/s

Pitfalls at scale



- Volume rendering
- Startup time
 - Loading plugins overwhelmed file system
 - Took ~5 minutes
 - Solution #1: Read plugin information on MPI task 0 and broadcast. (90% speedup)
 - Solution #2: static linking
 - Still need to demonstrate at scale

Pitfalls at scale #2: All to one communication

- Each MPI task needs to report high level information
 - Was there an error in execution for that task?
 - Data extents? Spatial Extents?
- Previous implementation:
 - Every MPI task sends a direct message to MPI task 0.
- New implementation (Miller, LLNL):
 - Tree communication

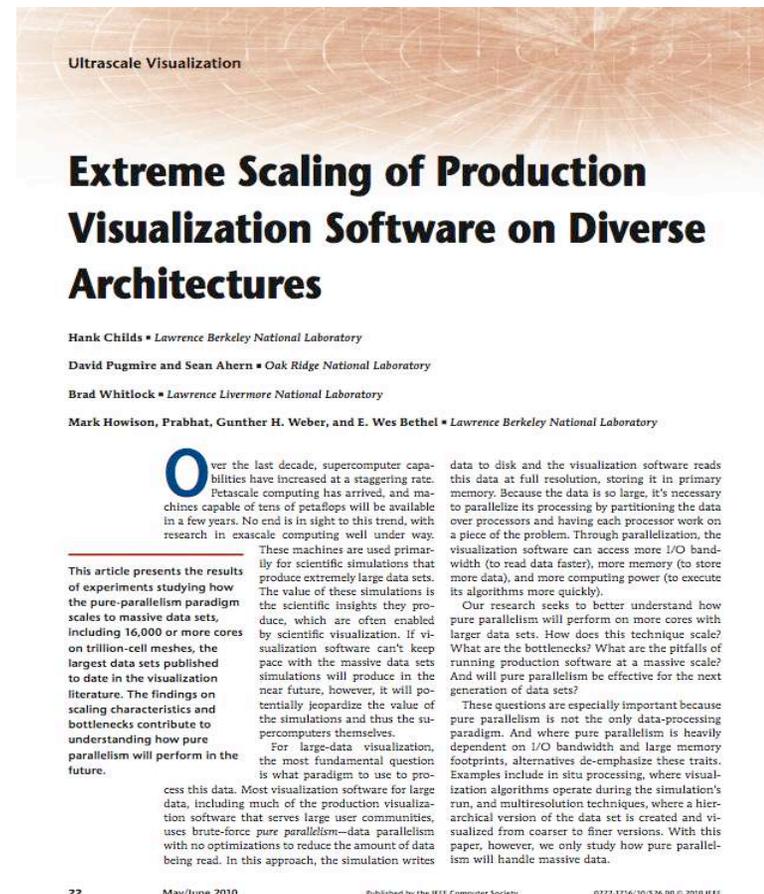
Pitfalls at scale #3: reproducible results

All-to-one?	No. of cores	Data set size (TCells)	Total I/O time (sec.)	Contour time (sec.)	Total pipeline execution time (sec.)	Pipeline minus contour & I/O (sec.)	Date run
Yes	16,384	1	88.0	32.2	368.7	248.5	June 2009
Yes	65,536	4	95.3	38.6	425.9	294.0	June 2009
No	16,384	1	240.9	32.4	277.6	4.3	Aug. 2009

Repeated debugging runs at scale are critical to resolving issues like these.

Conclusions

- Pure parallelism works, but is only as good as the underlying I/O infrastructure
 - ▣ and the I/O future looks grim
- Full results available in special issue of *Computer Graphics & Applications on Ultrascale Visualization*.



Ultrascale Visualization

Extreme Scaling of Production Visualization Software on Diverse Architectures

Hank Childs • Lawrence Berkeley National Laboratory

David Pugmire and Sean Ahern • Oak Ridge National Laboratory

Brad Whitlock • Lawrence Livermore National Laboratory

Mark Howison, Prabhat, Gunther H. Weber, and E. Wes Bethel • Lawrence Berkeley National Laboratory

Over the last decade, supercomputer capabilities have increased at a staggering rate. Petascale computing has arrived, and machines capable of tens of petaflops will be available in a few years. No end is in sight to this trend, with research in exascale computing well under way.

This article presents the results of experiments studying how the pure-parallelism paradigm scales to massive data sets, including 16,000 or more cores on trillion-cell meshes, the largest data sets published to date in the visualization literature. The findings on scaling characteristics and bottlenecks contribute to understanding how pure parallelism will perform in the future.

These machines are used primarily for scientific simulations that produce extremely large data sets. The value of these simulations is the scientific insights they produce, which are often enabled by scientific visualization. If visualization software can't keep pace with the massive data sets simulations will produce in the near future, however, it will potentially jeopardize the value of the simulations and thus the supercomputers themselves.

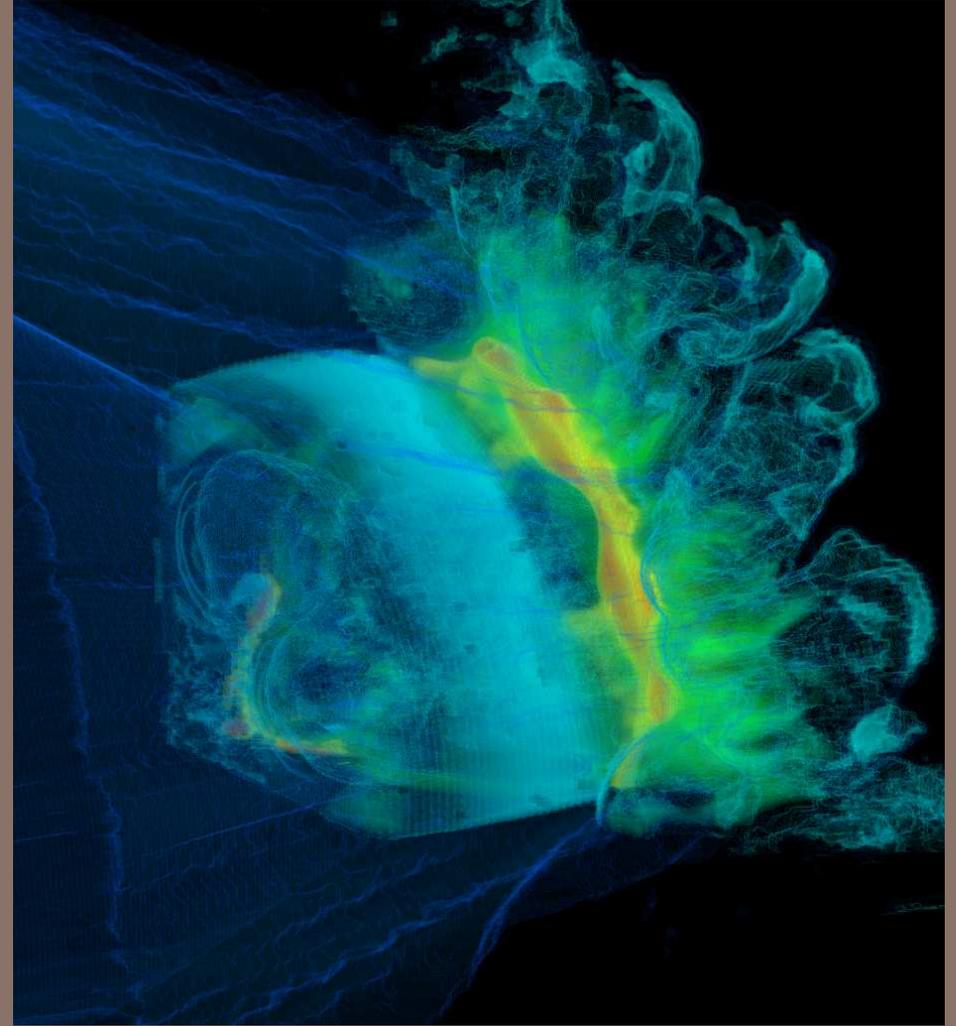
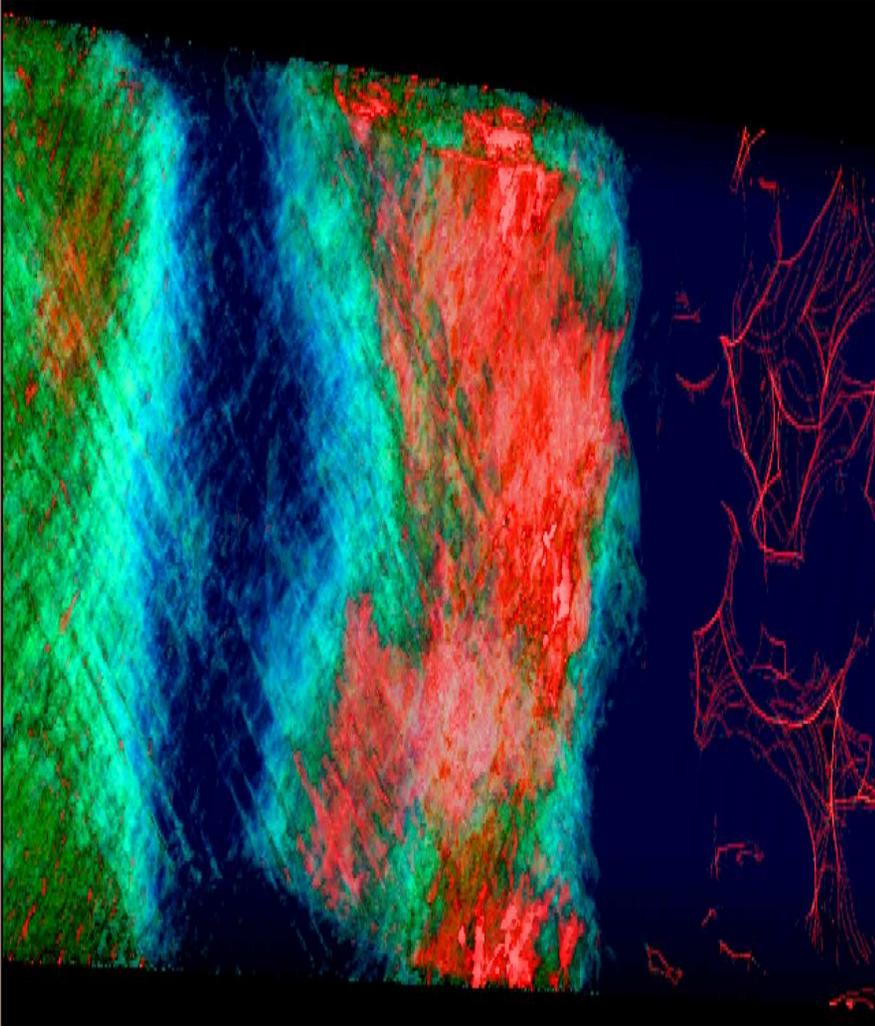
For large-data visualization, the most fundamental question is what paradigm to use to process this data. Most visualization software for large data, including much of the production visualization software that serves large user communities, uses brute-force pure parallelism—data parallelism with no optimizations to reduce the amount of data being read. In this approach, the simulation writes

data to disk and the visualization software reads this data at full resolution, storing it in primary memory. Because the data is so large, it's necessary to parallelize its processing by partitioning the data over processors and having each processor work on a piece of the problem. Through parallelization, the visualization software can access more I/O bandwidth (to read data faster), more memory (to store more data), and more computing power (to execute its algorithms more quickly).

Our research seeks to better understand how pure parallelism will perform on more cores with larger data sets. How does this technique scale? What are the bottlenecks? What are the pitfalls of running production software at a massive scale? And will pure parallelism be effective for the next generation of data sets?

These questions are especially important because pure parallelism is not the only data-processing paradigm. And where pure parallelism is heavily dependent on I/O bandwidth and large memory footprints, alternatives de-emphasize these traits. Examples include in situ processing, where visualization algorithms operate during the simulation's run, and multiresolution techniques, where a hierarchical version of the data set is created and visualized from coarser to finer versions. With this paper, however, we only study how pure parallelism will handle massive data.

Non-embarrassingly Parallel Algorithms



June 13, 2011

Hank Childs, Lawrence Berkeley Lab & UC Davis

Outline



- Volume rendering
- Particle advection
- Connected components & line scans

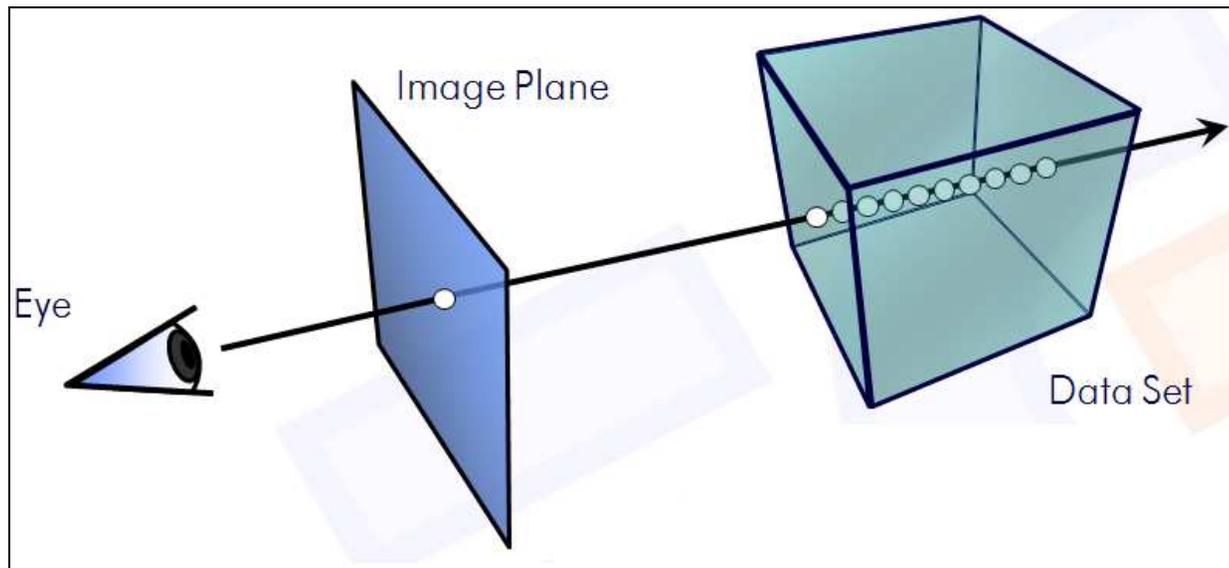
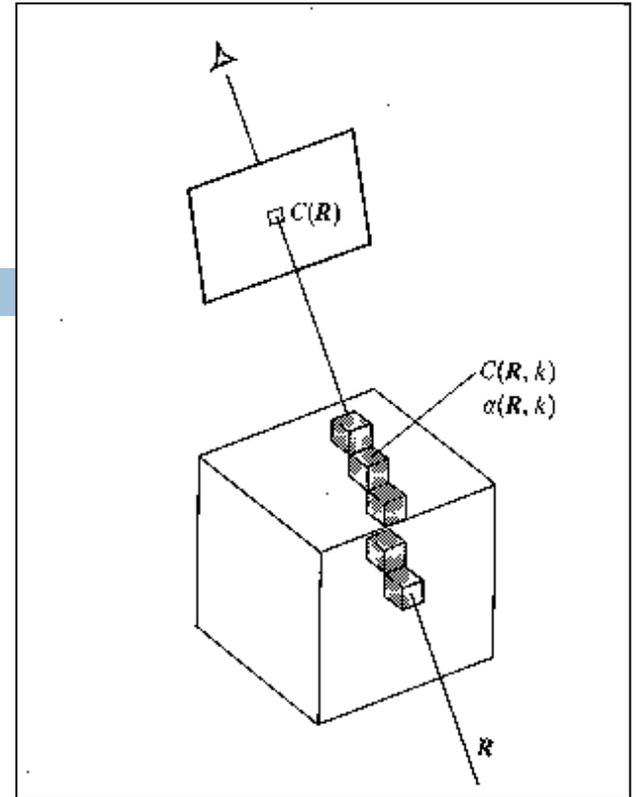
Outline



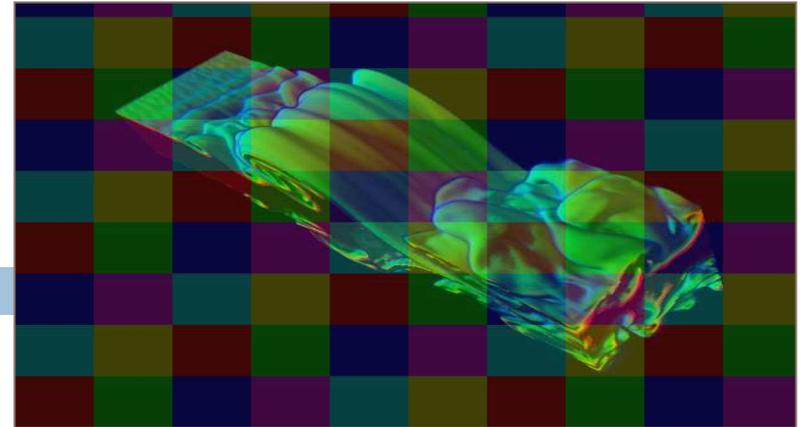
- Volume rendering
- Particle advection
- Connected components & line scans

Algorithm Studied: Raycasting VR

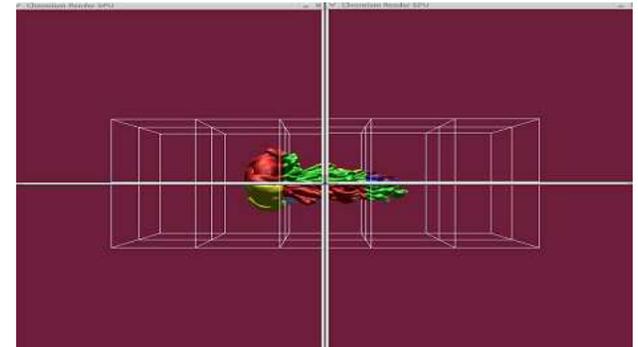
- Overview of Levoy's method
 - For each pixel in image plane:
 - Find intersection of ray and volume
 - Sample data (RGBa) along ray, integrate samples to compute final image pixel color



Parallelizing Volume Rendering



- Image-space decomposition.
 - ▣ Each process works on a disjoint subset of the final image (in parallel)
 - ▣ Processes may access source voxels more than once, will access a given output pixel only once.
 - ▣ Great for shared memory parallelism.
- Object-space decomposition.
 - ▣ Each process works on a disjoint subset data (in parallel).
 - ▣ Processes may access output pixels more than once.
 - ▣ Output requires image composition (ordering semantics).



Parallel volume rendering pitfalls



- Both work-decomposition schemes suffer from load balance issues:
 - Object-space: what if some processor's portion of the data set dominates the view frustum?
 - Image-space: what if some processor's portion of the view frustum contains a large fraction of the data set?
- Object-space pitfall: what if the blocks can not be ordered properly?

Volume rendering pitfalls

- Image-space decomposition:
 - Load balance issue: what if some processor's portion of the view frustum contains a large fraction of the data set?

1M cells	Proc 0's pixels
1M cells	Proc 1's pixels
20M cells	Proc 2's pixels
1M cells	Proc 3's pixels
1M cells	Proc 4's pixels
1M cells	Proc 5's pixels

Image (side view)

This assumes an a priori decomposition of pixels. Dynamic decomposition of pixels entails an entirely different set of problems.

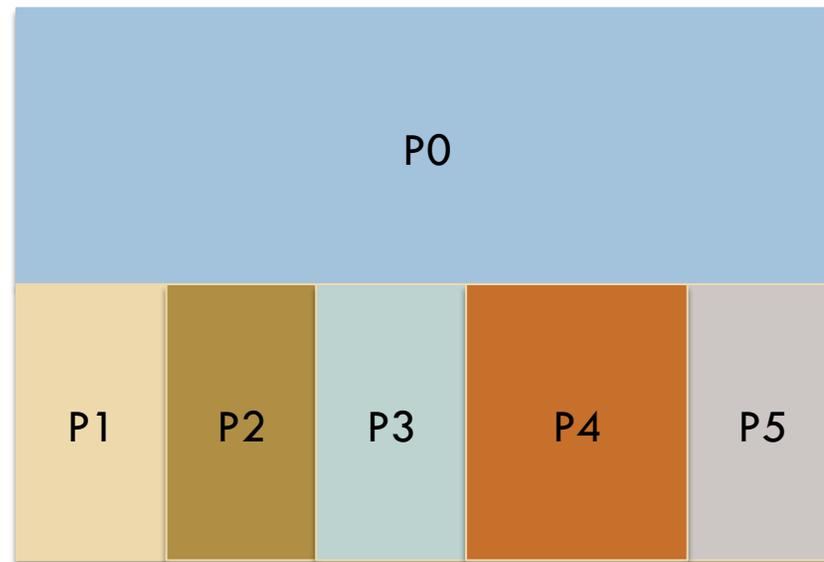
Parallel volume rendering pitfalls

- Image-space decomposition:
 - Performance issue: list of cells to consider changes every render.
 - Solutions:
 - Go to disk?
 - (probably bad)
 - Data redistribution amongst processors?
 - Could work, but load balance issues from last slide are still in play...

Image-space decomposition is well suited for shared memory parallelism. Load balance becomes easy since pixels can be dynamically assigned and every processor can access every cell.

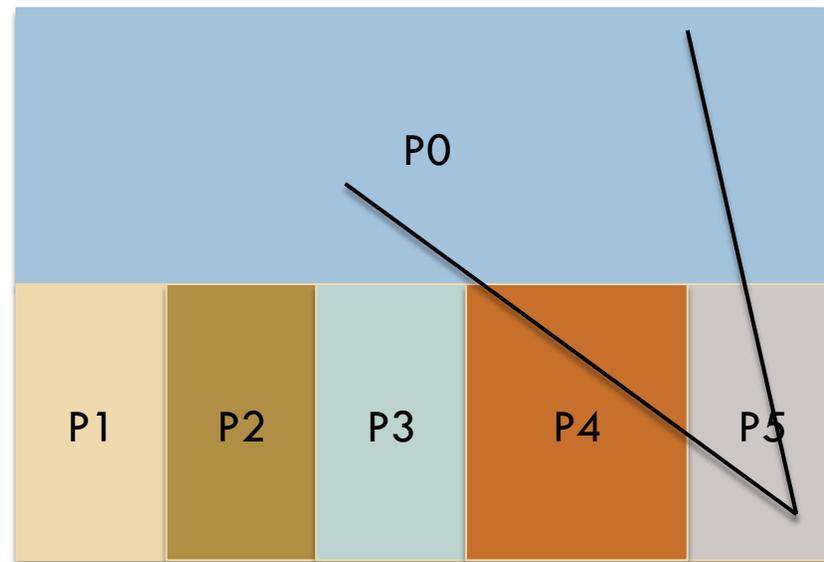
Parallel volume rendering pitfalls

- Object-space decomposition:
 - Load balance issue: what if some processor's portion of the view frustum contains a large fraction of the data set?



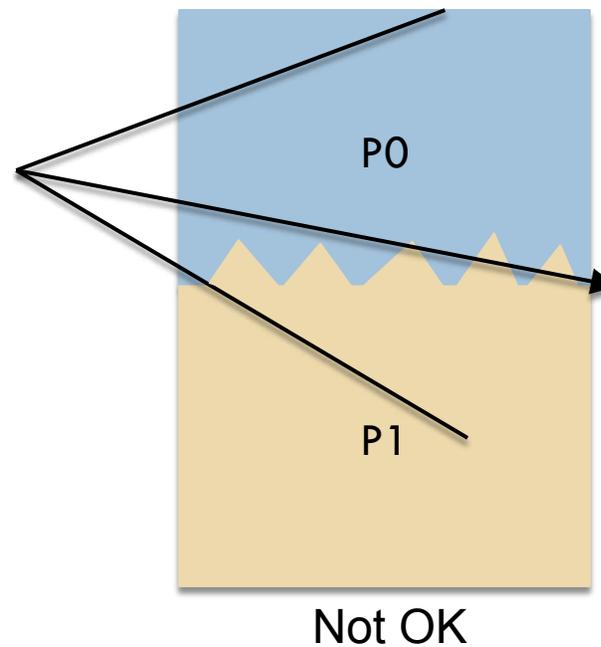
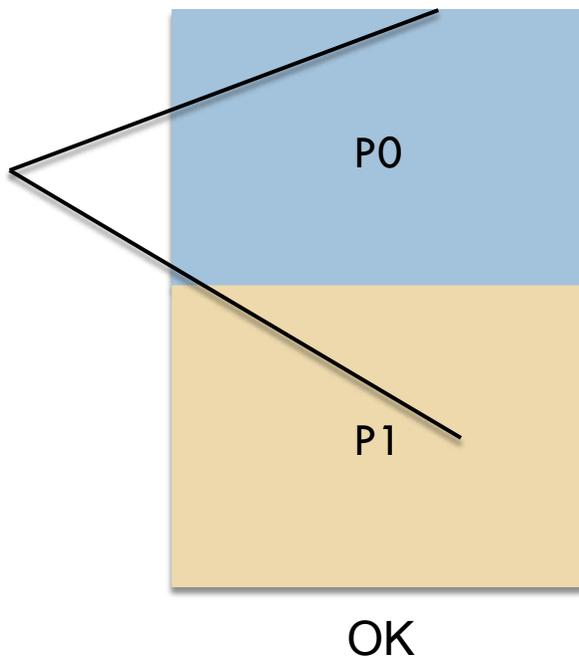
Parallel volume rendering pitfalls

- Object-space decomposition:
 - Load balance issue: what if some processor's portion of the view frustum contains a large fraction of the data set?



Parallel volume rendering pitfalls

- Object-space decomposition:
 - Applicability issue: what if there is no possible ordering of sub-images?



Hybrid Volume Rendering

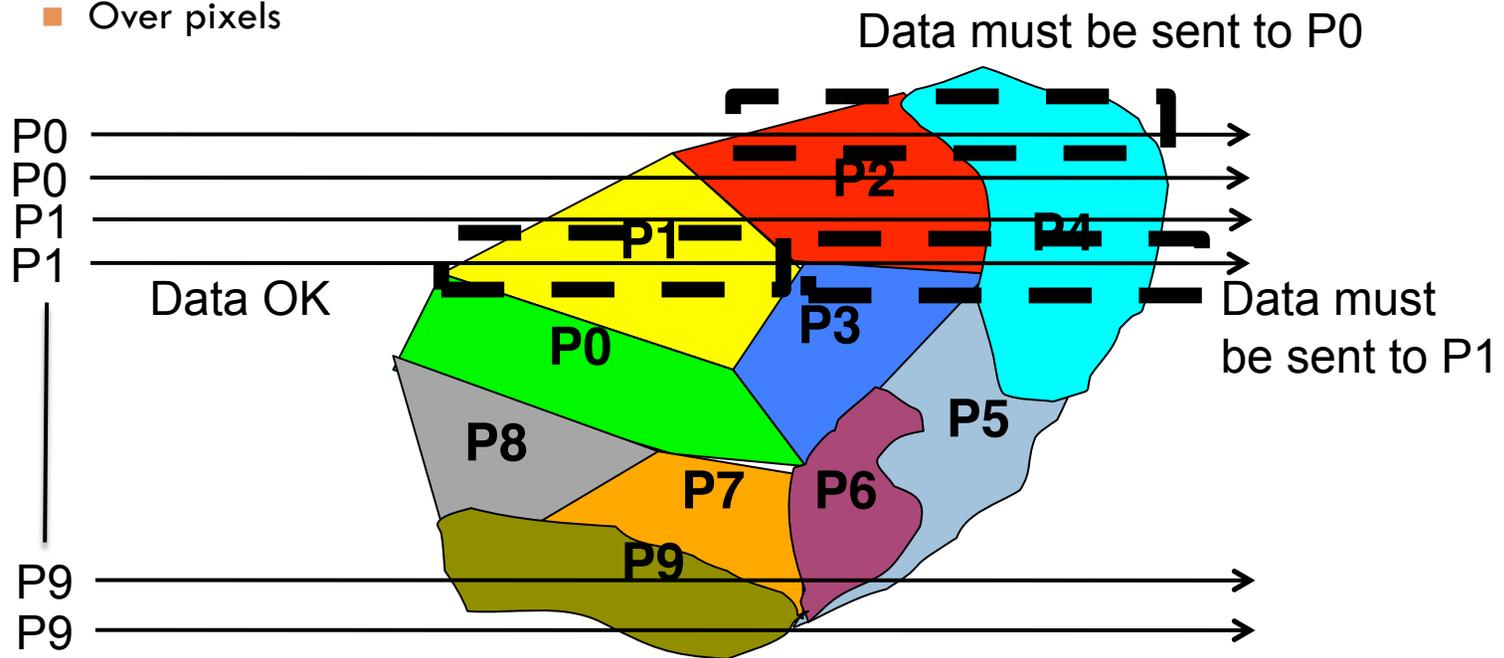
- Hybrid volume rendering:
 - Refers to mixture of object- and image-order techniques to do volume rendering.
 - Most contemporary parallel volume rendering projects are hybrid volume renderers:
 - Object order – divide data into disjoint chunks, each processor works on its chunk of data.
 - Image order – parallel compositing algorithm divides work over final image, each composites over its portion of the final image.
 - A two-stage algorithm, heavy communication load between stages.

Hybrid Volume Rendering

- Hybrid volume rendering:

- Dual partition scheme:

- Over data
 - Over pixels



Reconsidering pitfalls

- Image-space decomposition:
 - Load balance issue: what if some processor's portion of the view frustum contains a large fraction of the data set?

1M cells	Proc 0's pixels
1M cells	Proc 1's pixels
20M cells	Proc 2's pixels
1M cells	Proc 3's pixels
1M cells	Proc 4's pixels
1M cells	Proc 5's pixels

Image (side view)

Non-issue for hybrid parallel ... all pixels have approximately the same amount of data.

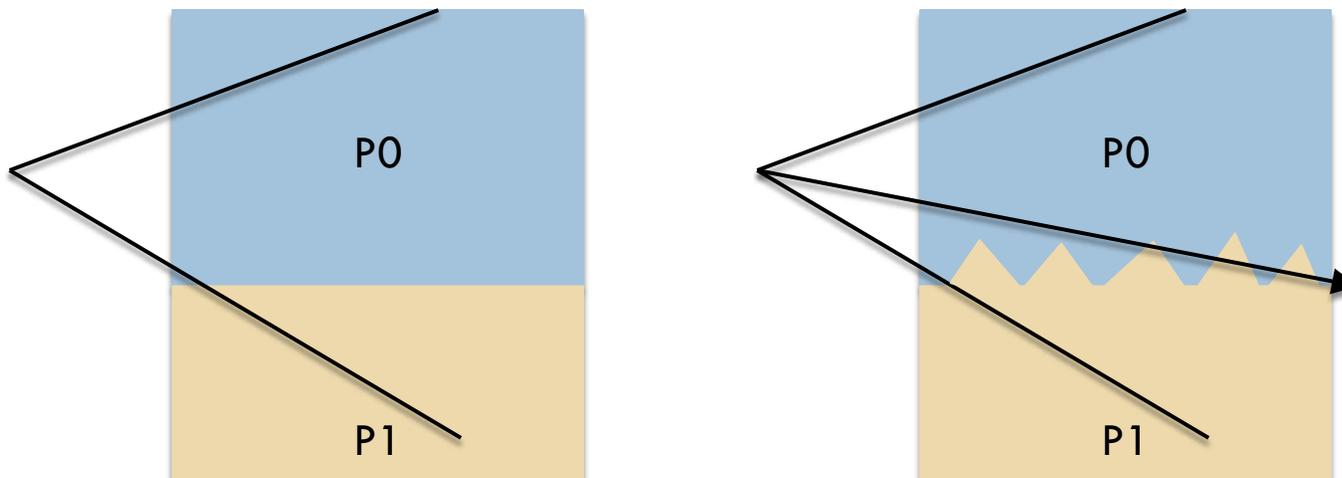
Reconsidering pitfalls

- Image-space decomposition:
 - Performance issue: list of cells to consider changes every render.
 - Solutions:
 - Go to disk?
 - (probably bad)
 - Data redistribution amongst processors?
 - Could work, but load balance issues from last slide are still in play...

Not an issue for hybrid volume rendering ... cell list is the same for every view.

Parallel volume rendering pitfalls

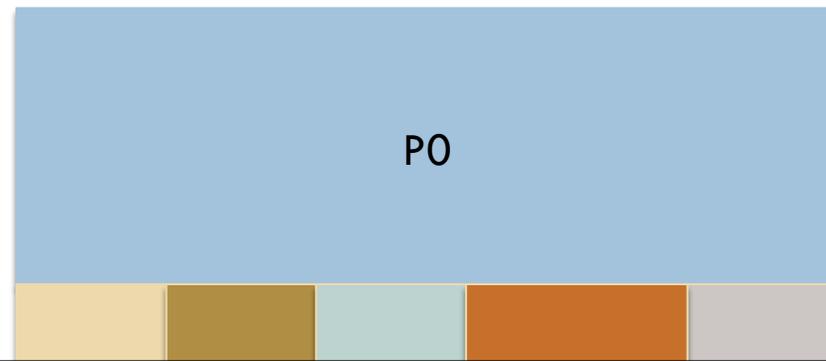
- Object-space decomposition:
 - Applicability issue: what if there is no possible ordering of sub-images?



Not an issue for hybrid volume rendering ... data can be ordered as it is composited.

Parallel volume rendering pitfalls

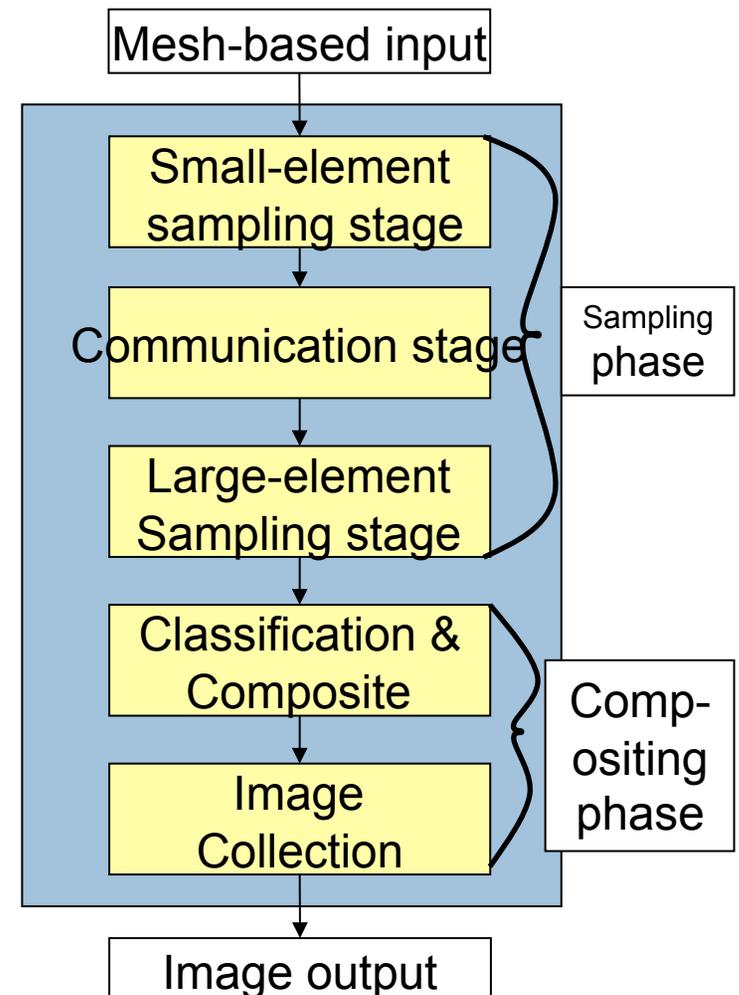
- Object-space decomposition:
 - Load balance issue: what if some processor's portion of the view frustum contains a large fraction of the data set?



This one still is a problem for hybrid parallel volume rendering.

Optimization: only sample “small” cells in first phase.

- **Small-element sampling stage:**
 - ▣ Parallelizes over object decomposition
 - ▣ Samples small elements, defers large elements
 - ▣ Outputs partially populated (G, F) , plus untouched large elements
- **Communication stage**
 - ▣ Large all-to-all communication to go from object to image decomposition
- **Large-element sampling stage:**
 - ▣ Parallelizes over image decomposition
 - ▣ Samples large elements, but only the portions within portion of image-space
 - ▣ Each processor outputs fully populated (G, F) for its portion of image space



This algorithm was demonstrated to be strongly scalable.

Scaling study:

- 100M element unstructured grid, 1024x1024 pixel image
- Cluster of 2.4GHz Opterons, connected by InfiniBand
- Run within real world application (VisIt)
- Variations:
 - 3D rasterization versus Kernel-based sampling
 - Camera inside data set versus outside data set

Procs	3D Rast. / outside	3D Rast. / Inside	Kernel / Outside	Kernel / inside
25	12.0s	21.9s	12.1s	63.7s
50	5.8s	12.1s	5.8s	30.5s
100	3.0s	7.0s	3.1s	15.3s
200	1.6s	3.6s	1.3s	7.6s
400	0.9s	2.1s	0.7s ⁹	4.1s

Outline



- Volume rendering
- Particle advection
 - Motivating more than streamlines
 - Parallelization
- Connected components
- Line scans

Particle advection basics

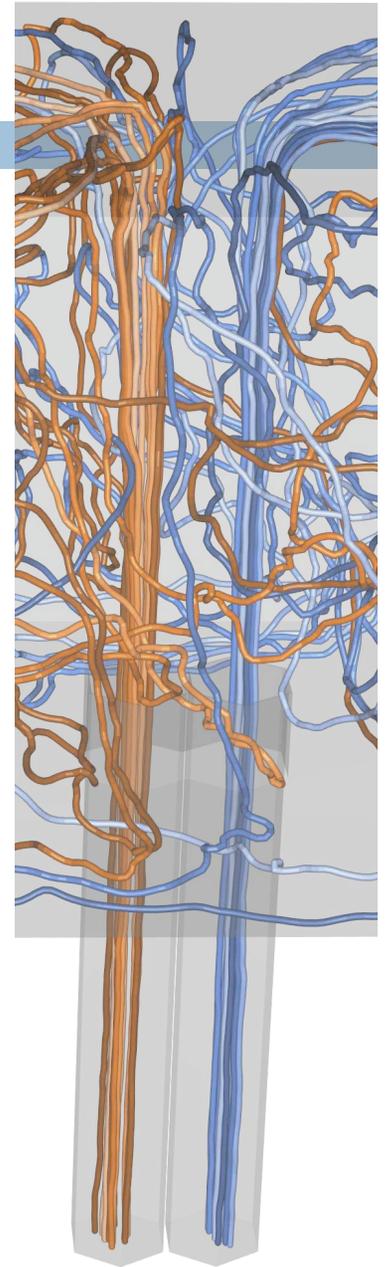
Advecting particles create integral curves

$$S'(t) = v(t, S(t)) \quad S(t_0) := x_0$$

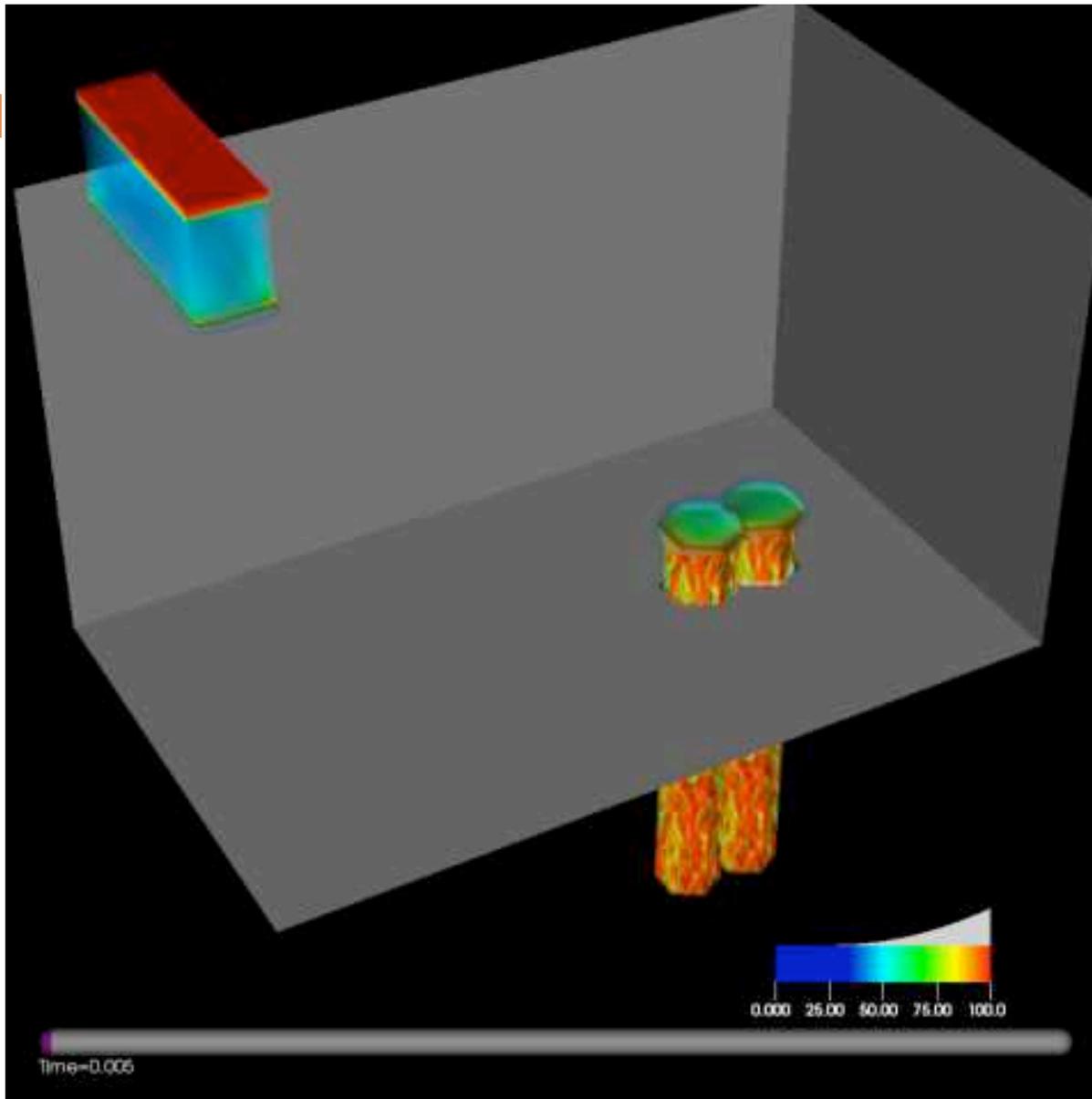


Streamlines: display particle path
(instantaneous velocities)

Pathlines: display particle path (velocity
field evolves as particle moves)

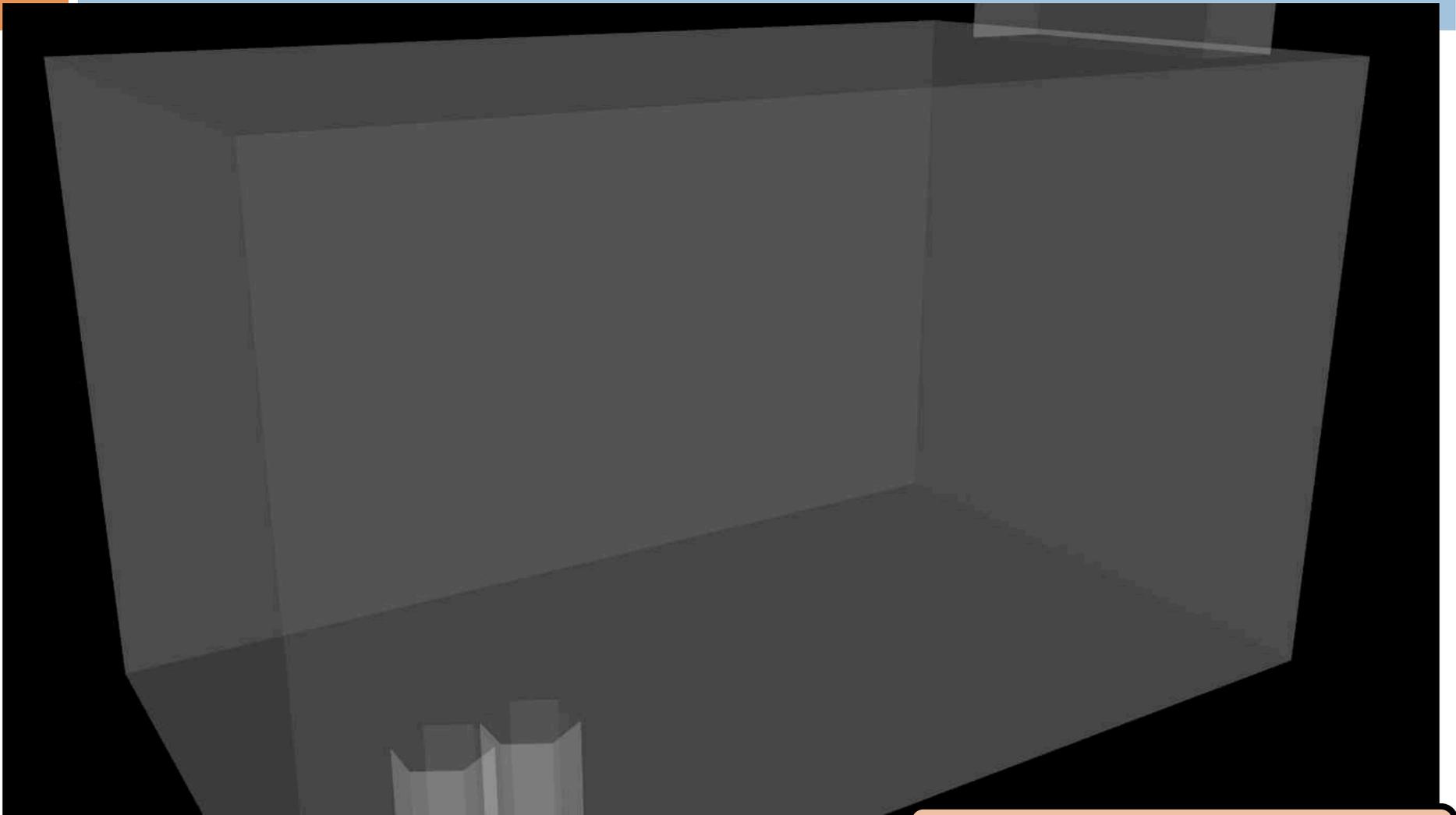


“The Fish Tank”

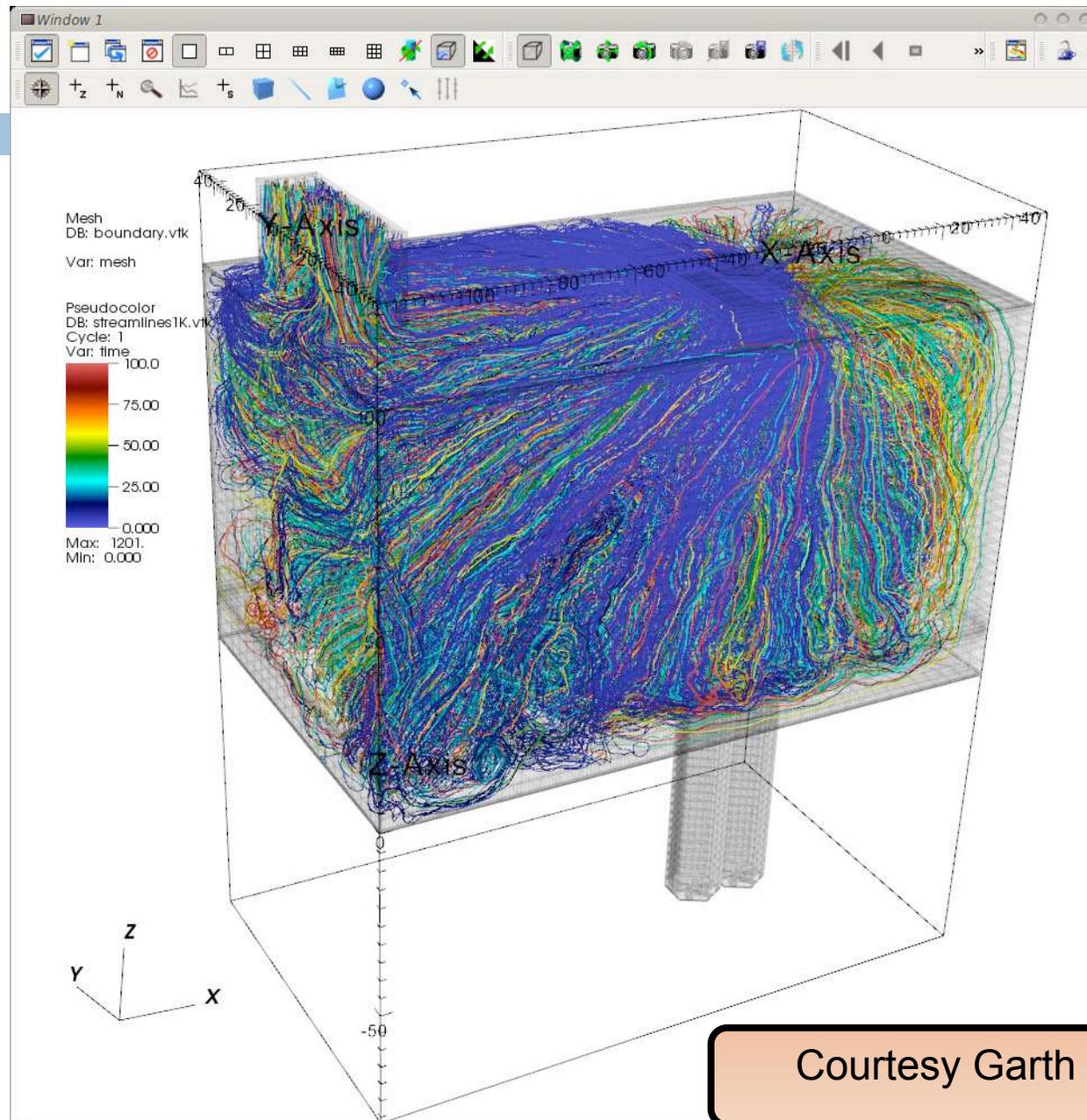


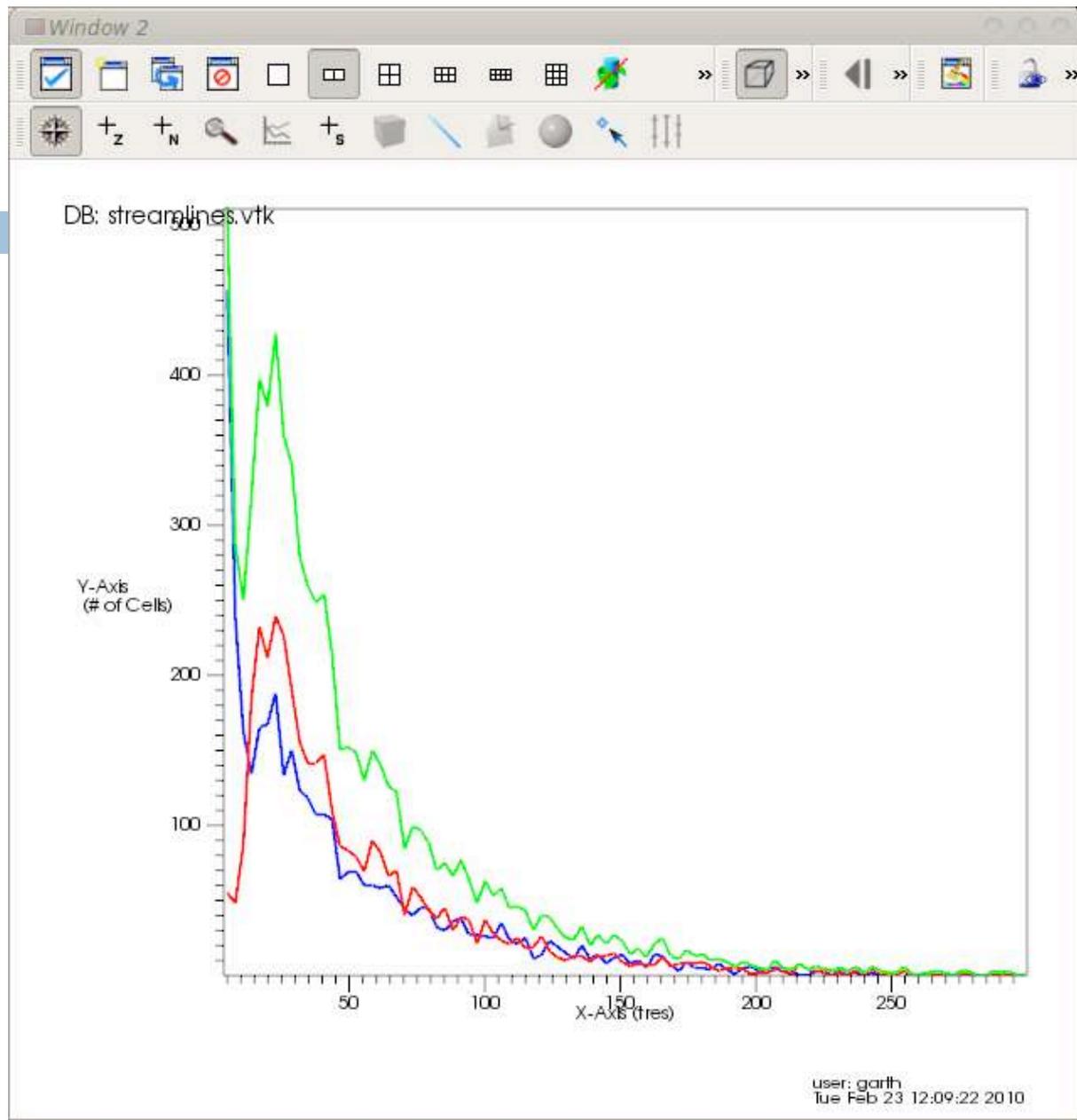
“Simulation of the Turbulent Flow of Coolant in an Advanced Recycling Nuclear Reactor.” Movie credits to Childs, Fischer, Obabko, Pointer, and Siegel

Particles Moving Through the “Fish Tank”



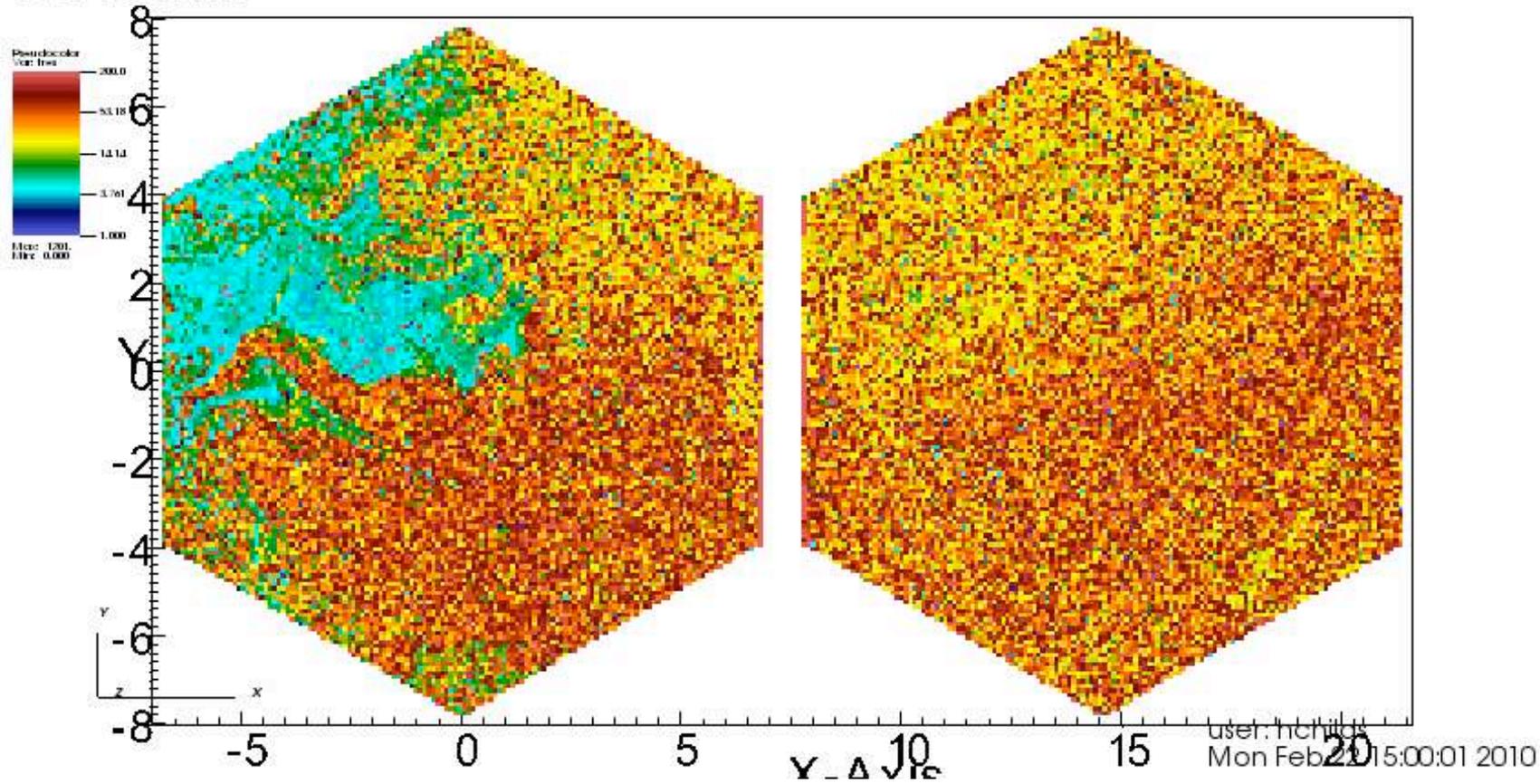
Courtesy Garth & Childs



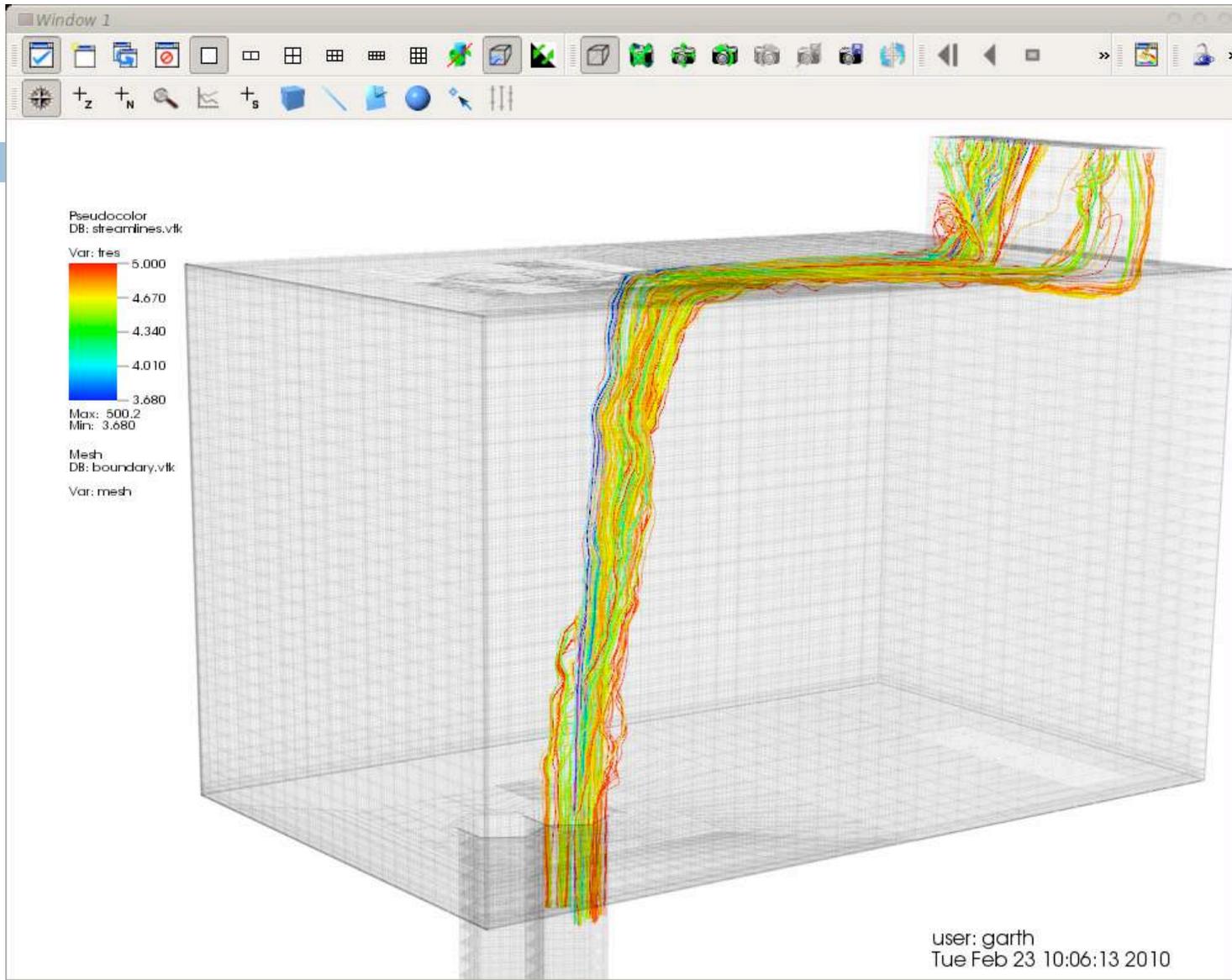


Courtesy Garth & Childs

DB: test.vtk



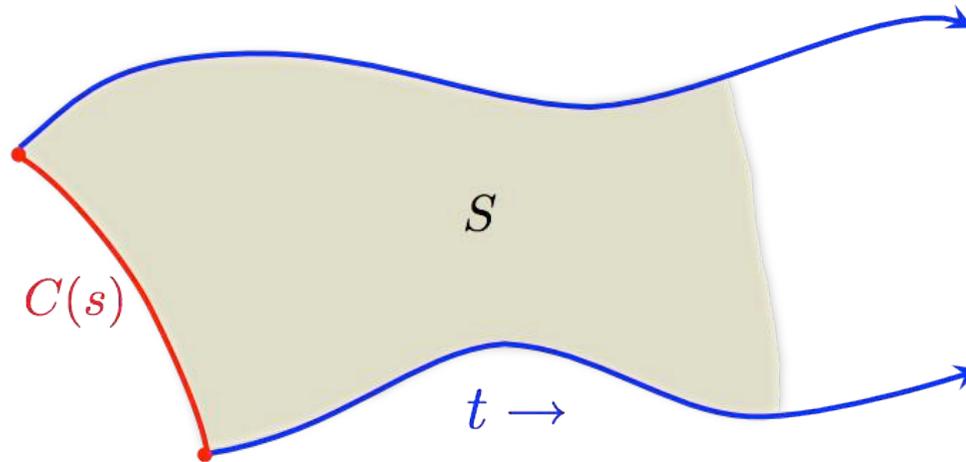
Courtesy Garth & Childs



Courtesy Garth & Childs

Sets of Streamlines

- Visualizing all integral curves...
 - ... starting from a seed curve:
Stream Surface or Path Surface



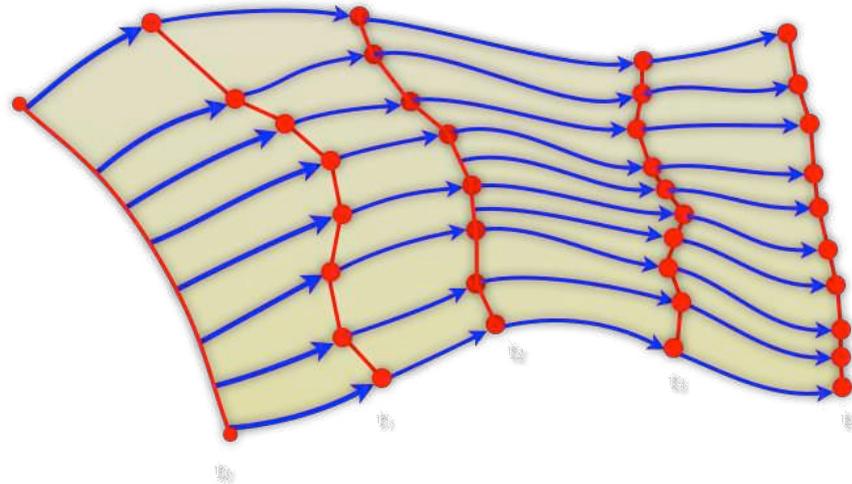
$$\frac{d}{dt}S(s, t) = \vec{v}(t, S(s, t))$$

$$S(s, 0) := C(s)$$

Courtesy Garth

Sets of Streamlines

- Stream surface computation:

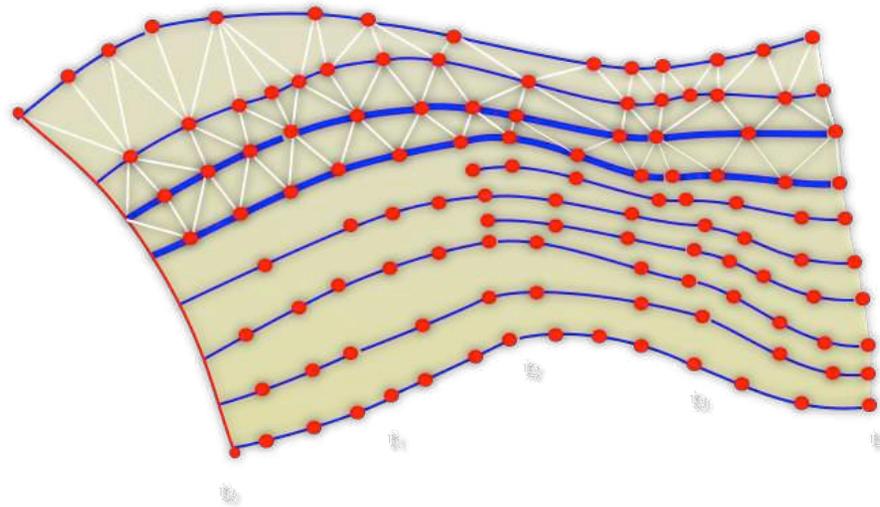


- Skeleton from Integral Curves + Timelines

Courtesy Garth

Sets of Streamlines

□ Stream surface computation:



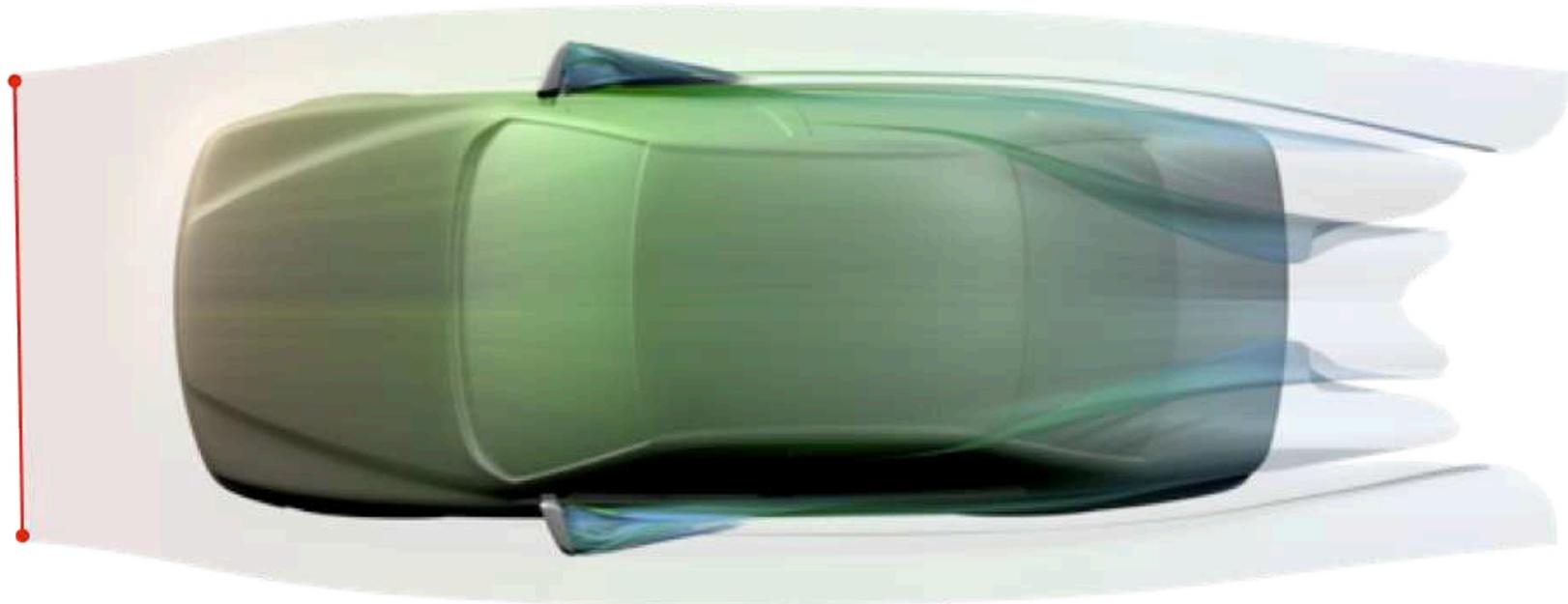
- Skeleton from Integral Curves + Timelines
- Triangulation

Generation of Accurate Integral Surfaces in Time-Dependent Vector Fields. C. Garth, H. Krishnan, X. Tricoche, T. Bobach, K. I. Joy. In IEEE TVCG, 14(6):1404–1411, 2007

Courtesy Garth

Sets of Streamlines

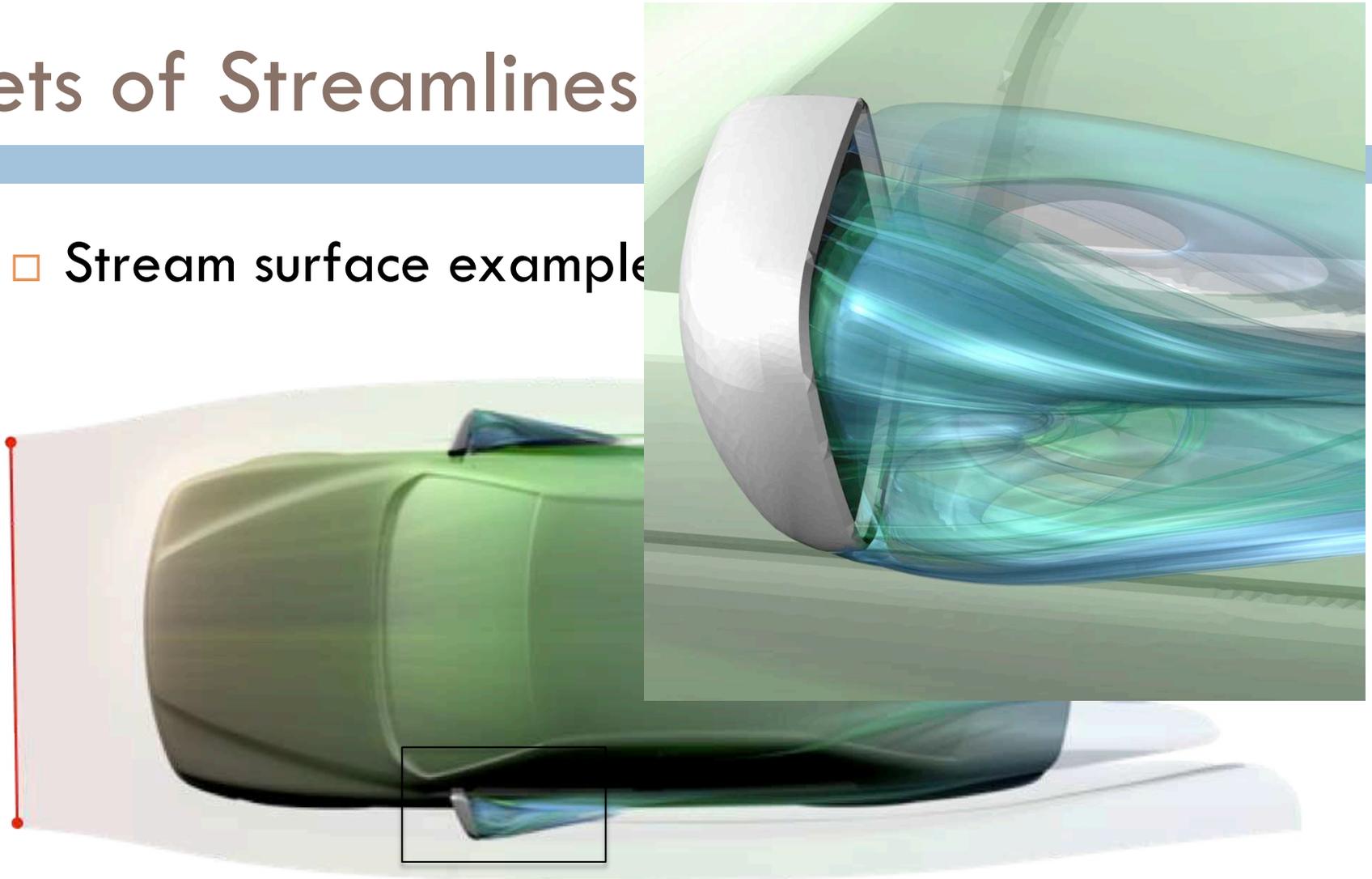
- Visualizing all integral curves...
 - ... starting from a seed curve:
Stream Surface or Path Surface



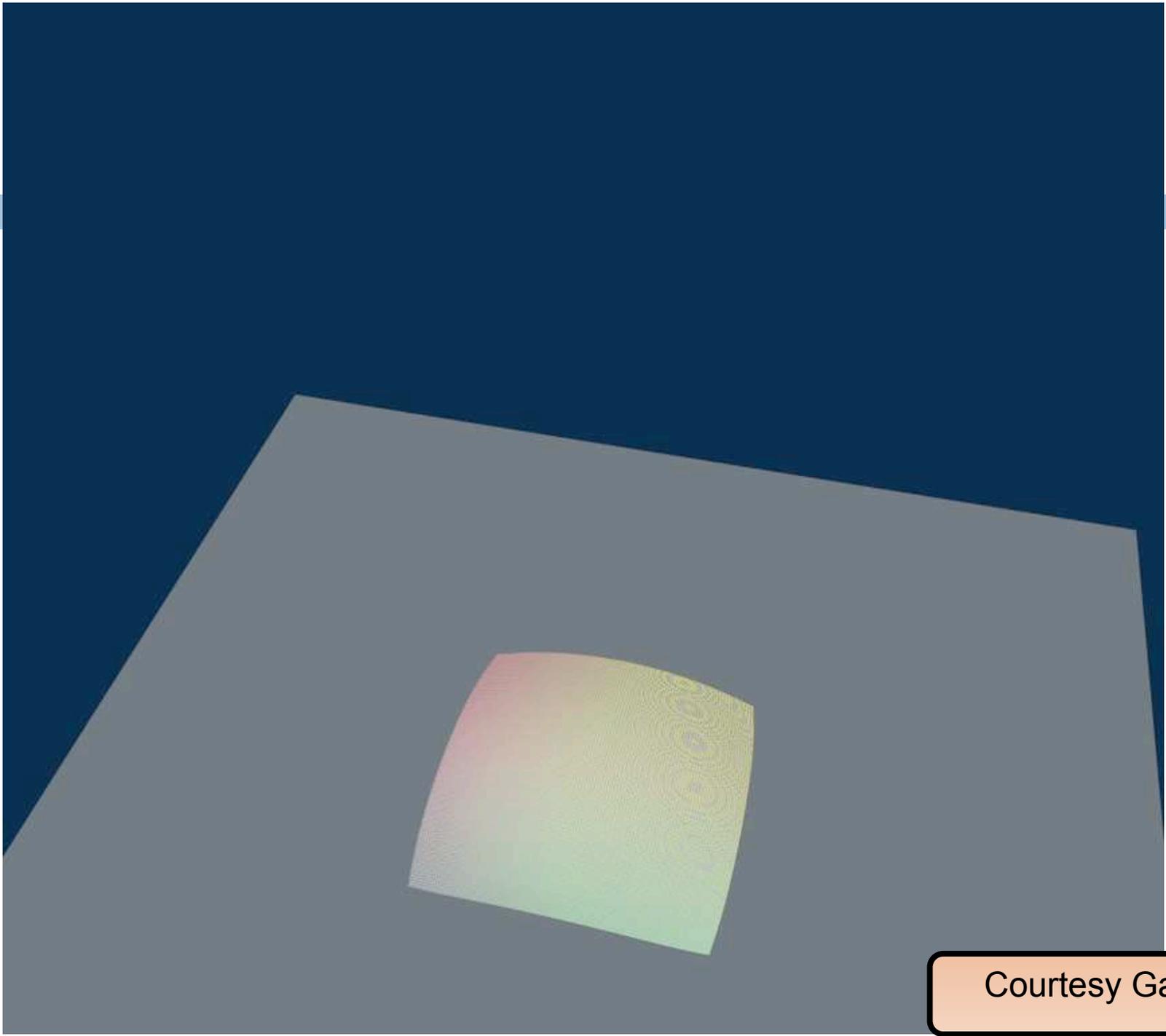
Courtesy Garth

Sets of Streamlines

- Stream surface example



Courtesy Garth



Courtesy Garth

Lagrangian Methods

- Visualize manifolds of maximal stretching in a flow, as indicated by dense particles



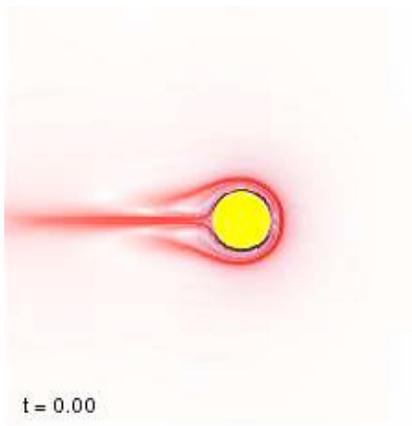
- Finite-Time Lyapunov Exponent (FTLE)

$$\sigma_{\Delta t}(t, x) := \frac{1}{\Delta t} \ln \sqrt{\lambda_{max}(D_x \phi_{\Delta t}(t, x))}$$

Courtesy Garth

Lagrangian Methods

- Visualize manifolds of maximal stretching in a flow, as indicated by dense particles
 - Forward in time: **FTLE+** indicates divergence
 - Backward in time: **FTLE-** indicates convergence



www.vacet.org

time-varying

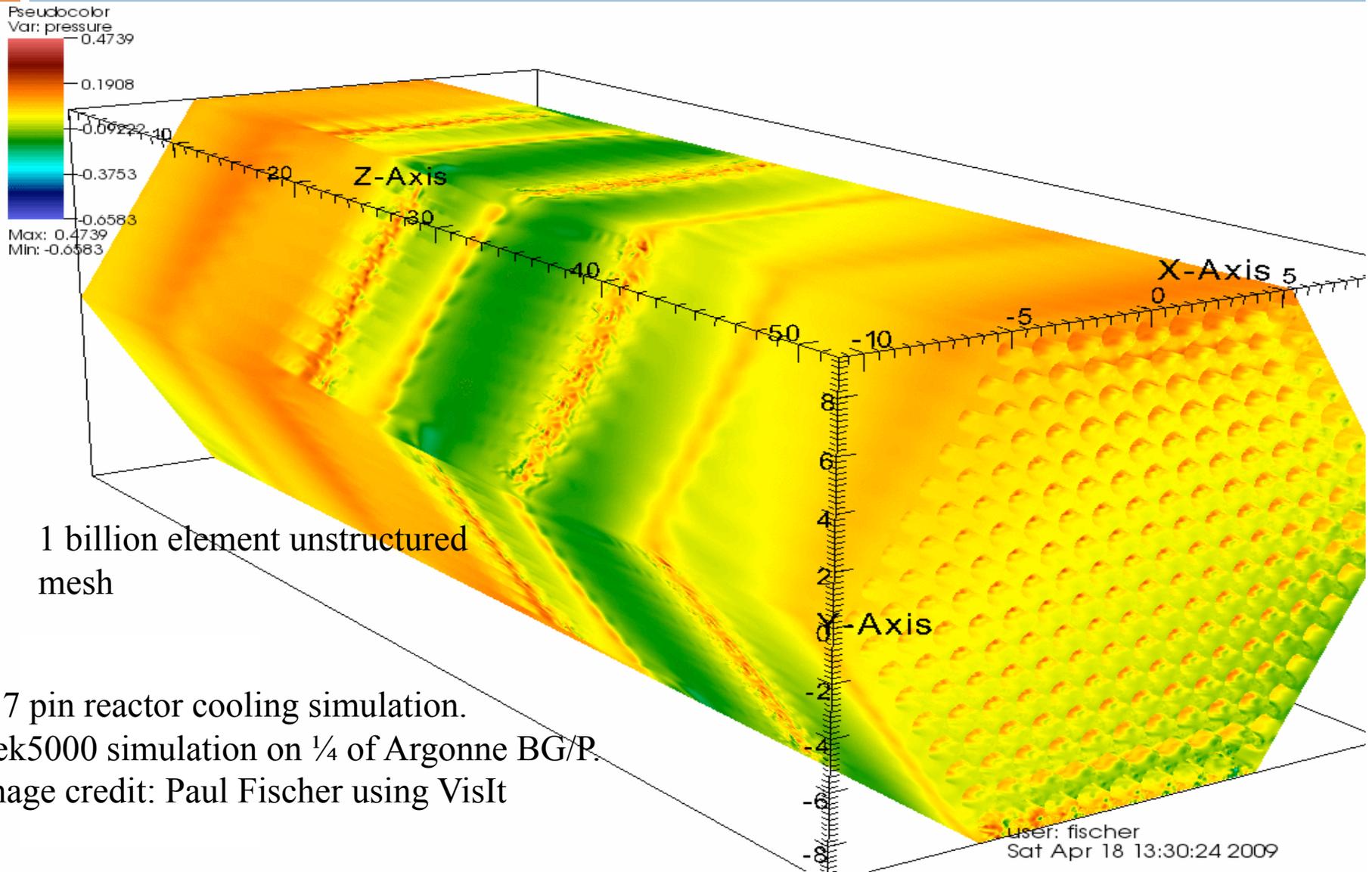
Courtesy Garth

Outline

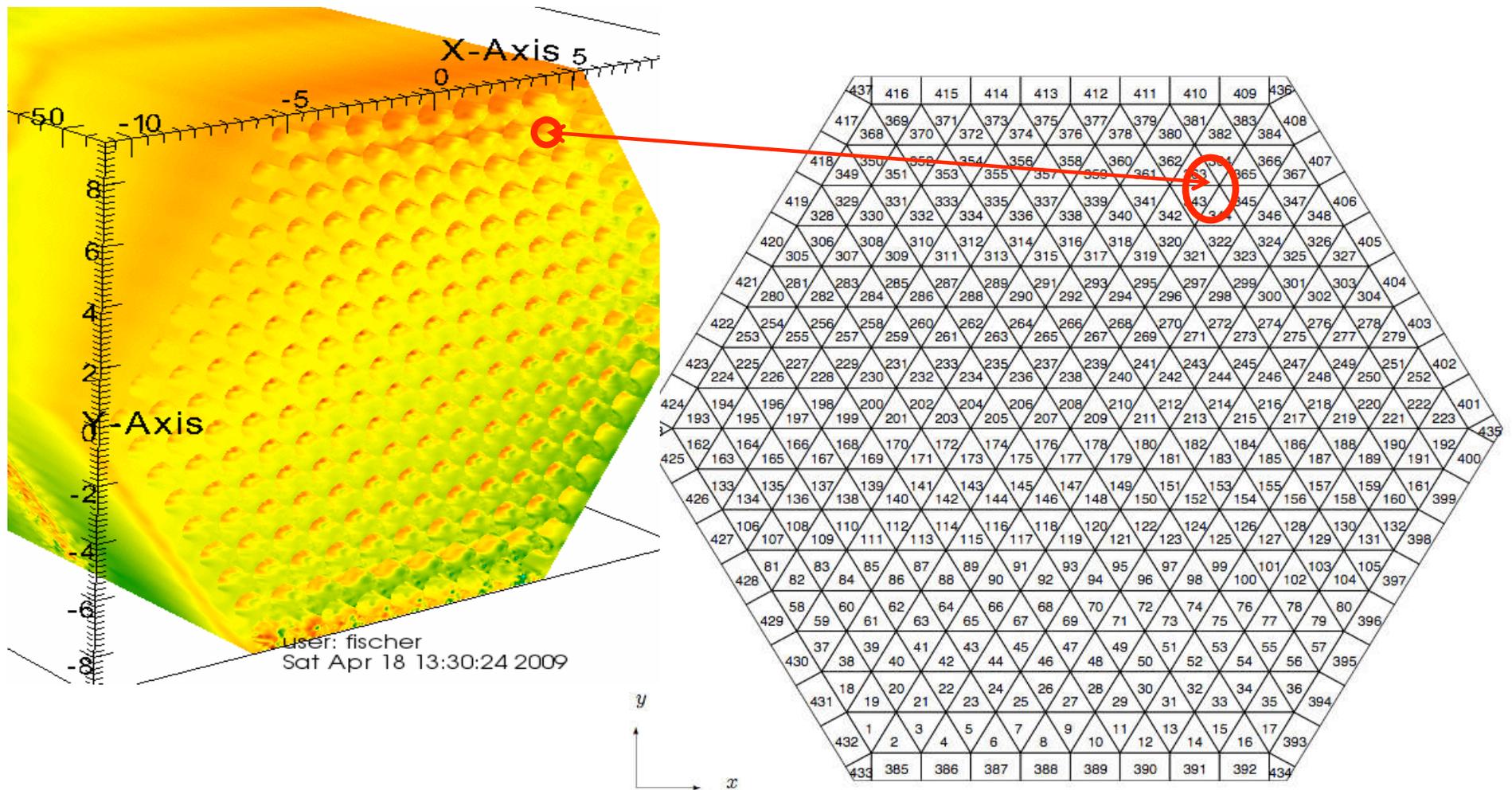


- Volume rendering
- Particle advection
 - Motivating more than streamlines
 - Parallelization
- Connected components & line scans

Supercomputers are generating large data sets that often require parallelized postprocessing.



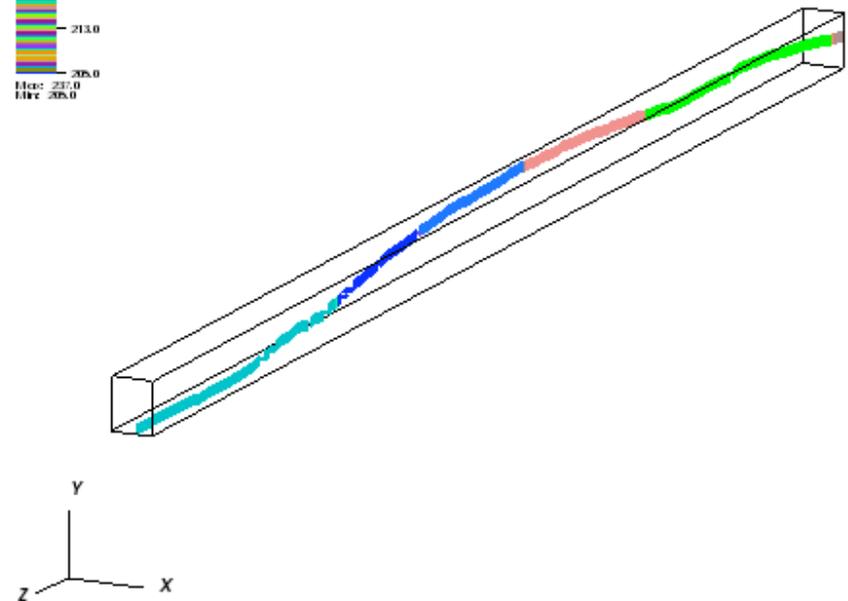
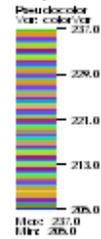
Communication between “channels” are a key factor in effective cooling.



Particle advection can be used to study communication properties.

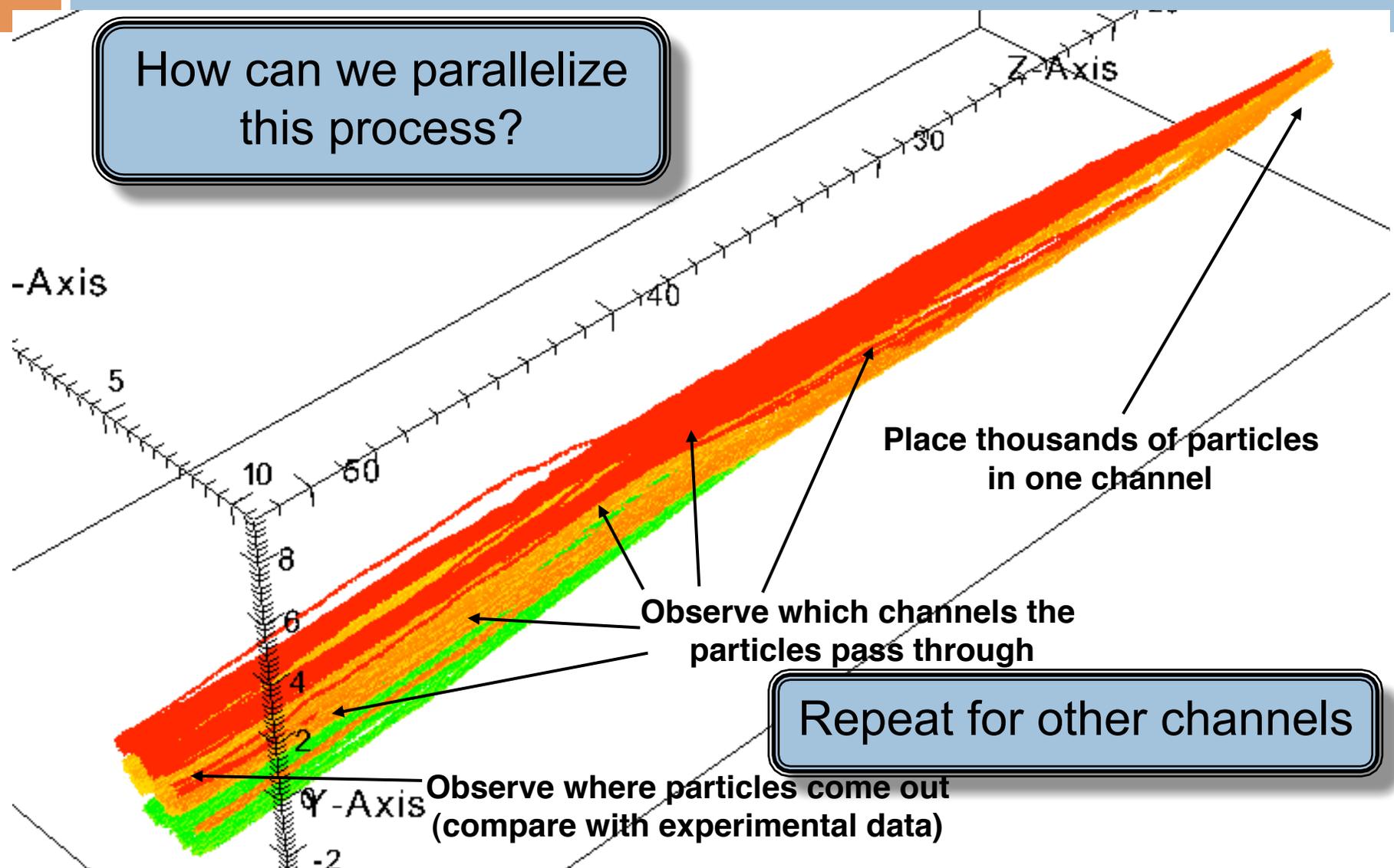


DB: visit0001.vtk
Cycle: 1

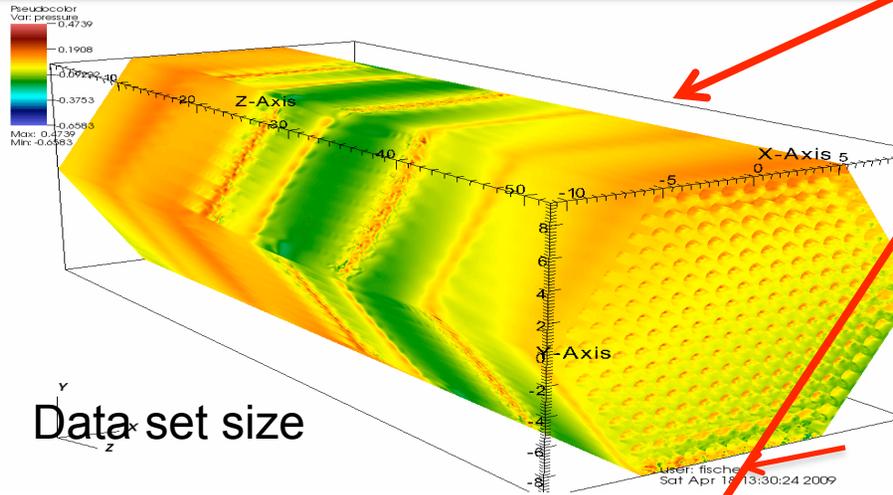
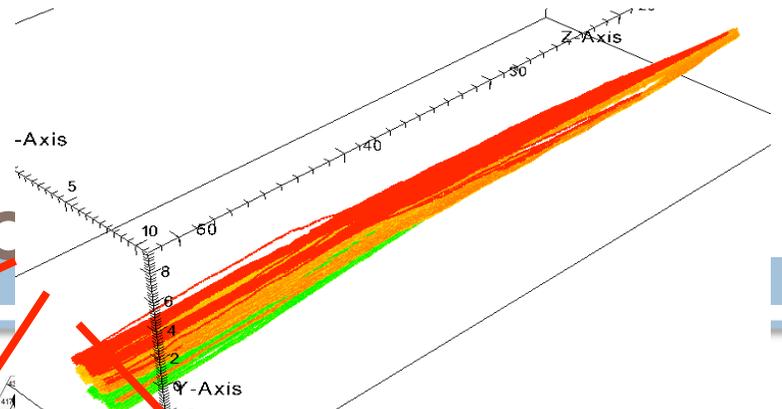


This sort of analysis requires many particles to be statistically significant.

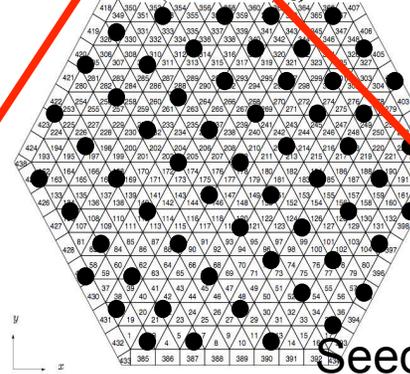
How can we parallelize this process?



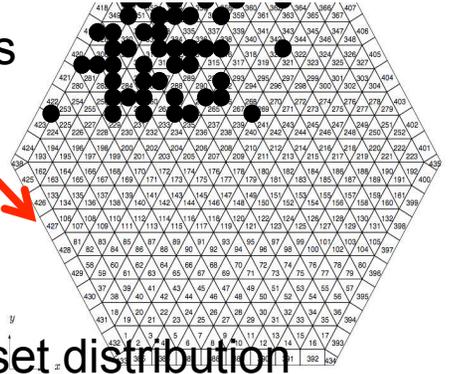
Particle advection: Four dimensions of cc



Data set size



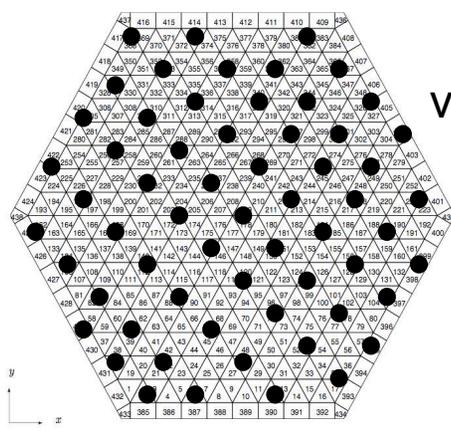
VS



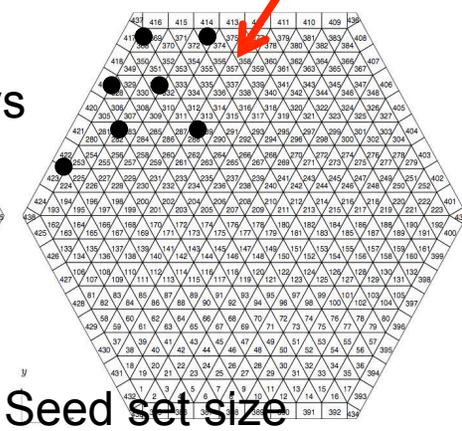
Seed set distribution

Figure 1: cells only:ps

Figure 1: cells only:ps



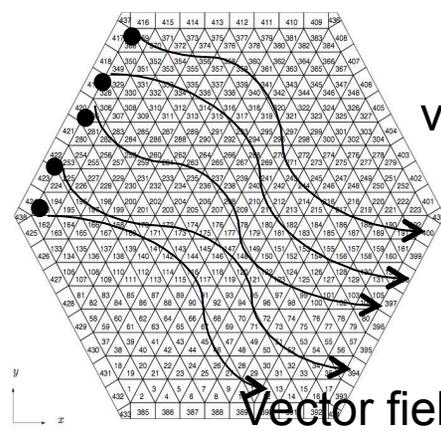
VS



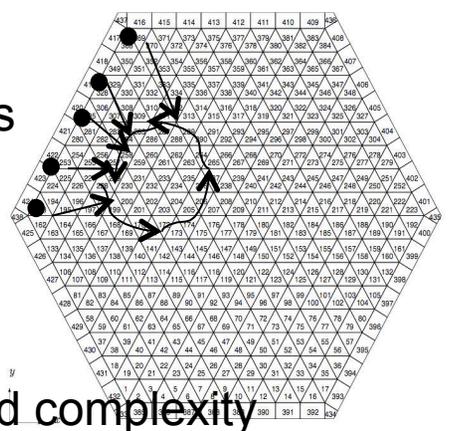
Seed set size

Figure 1: cells only:ps

Figure 1: cells only:ps



VS



Vector field complexity

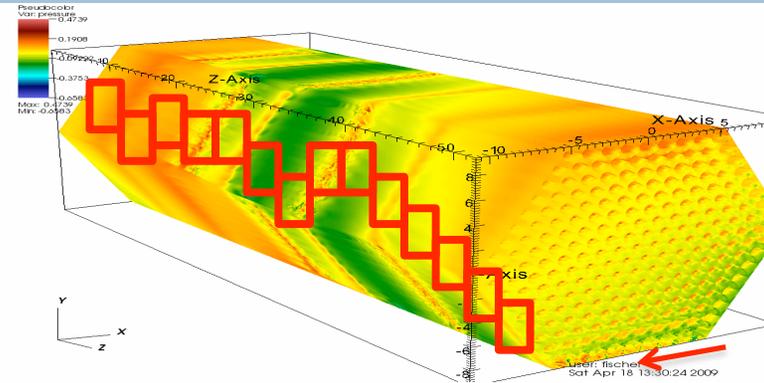
Figure 1: cells only:ps

Figure 1: cells only:ps

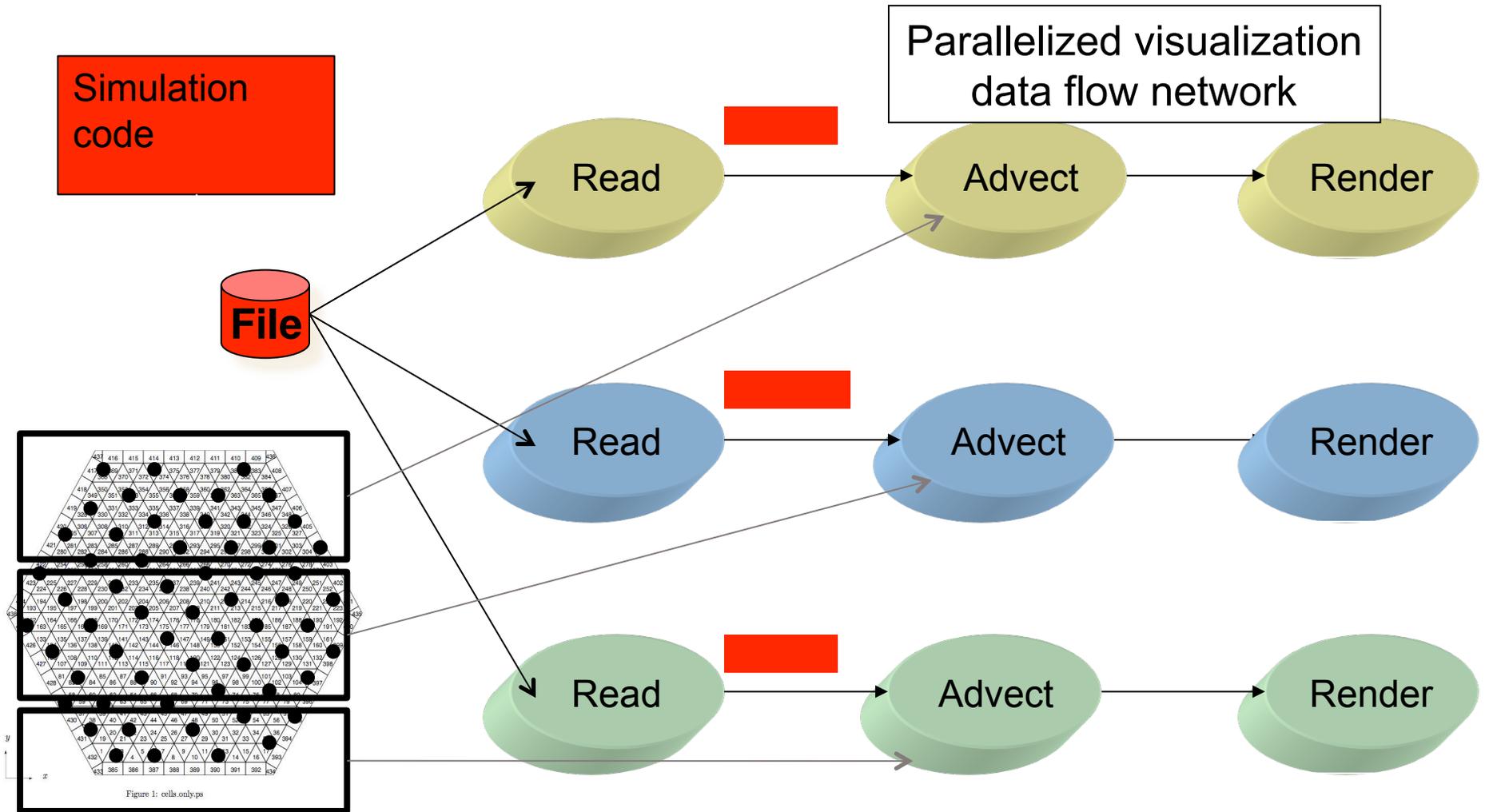
Do we need parallel processing?

When? How complex?

- Data set size?
 - Not enough!
- Large #'s of particles?

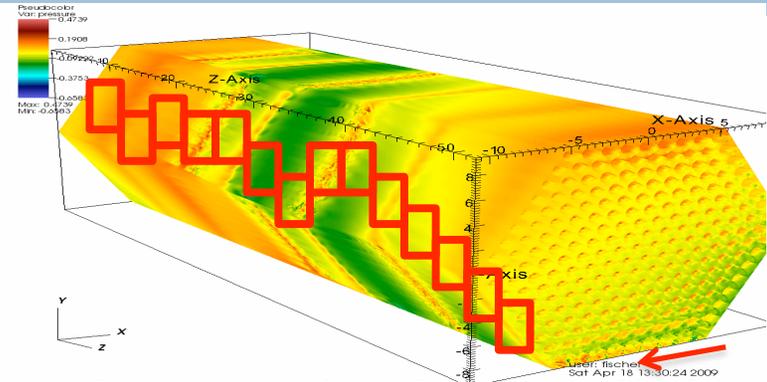


Parallelization for small data and a large number of particles.



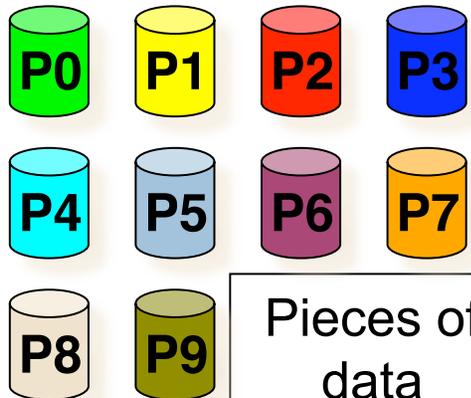
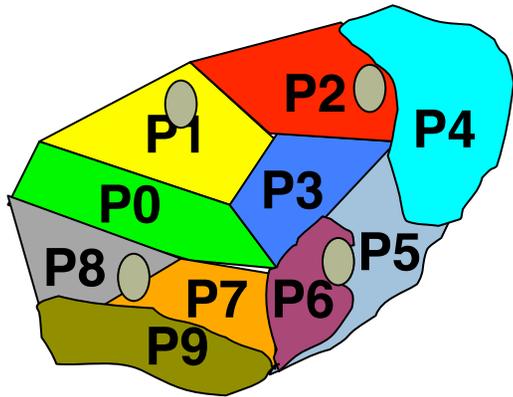
Do we need advanced parallelization techniques? When?

- Data set size?
 - Not enough!
- Large #'s of particles?
 - Need to parallelize, but embarrassingly parallel OK
- Large #'s of particles + large data set sizes



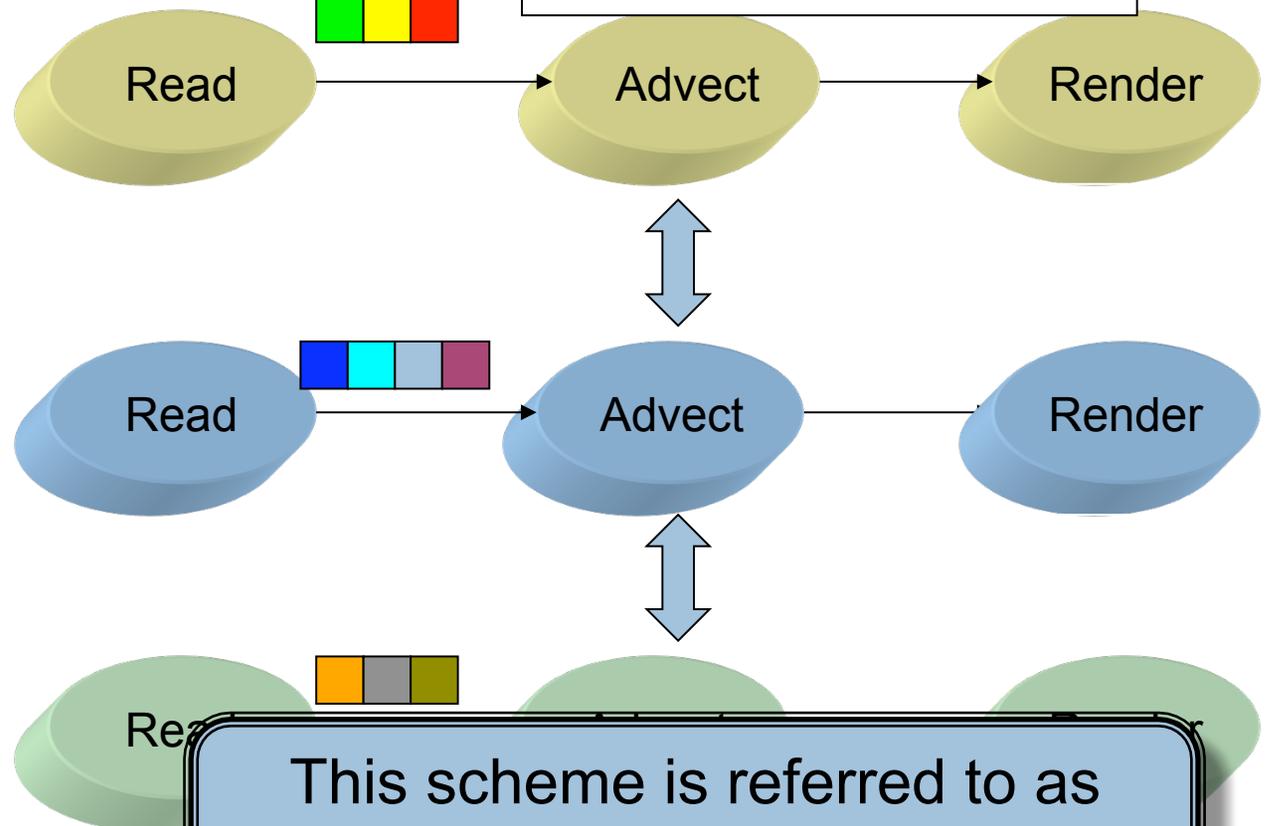
Parallelization for large data with good “distribution”.

Parallel Simulation Code



Pieces of data
(on disk)

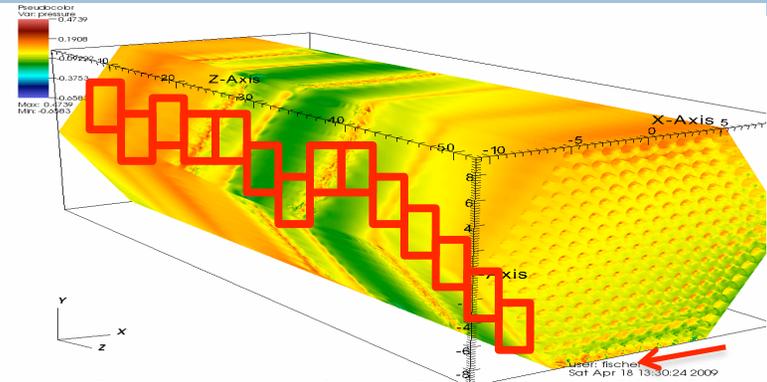
Parallelized visualization
data flow network



This scheme is referred to as
parallelizing-over-data.

Do we need advanced parallelization techniques? When?

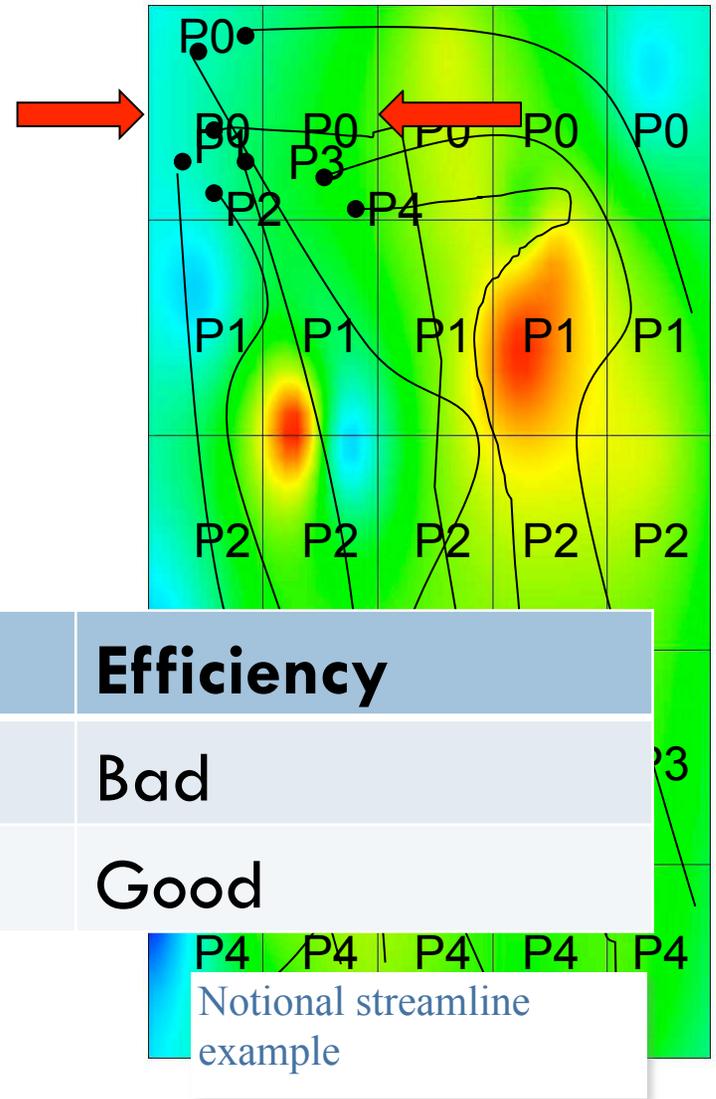
- Data set size?
 - ▣ Not enough!
- Large #'s of particles?
 - ▣ Need to parallelize, but embarrassingly parallel OK
- Large #'s of particles + large data set sizes
 - ▣ Need to parallelize, simple schemes may be OK
- Large #'s of particles + large data set sizes + (bad distribution OR complex vector field)
 - ▣ Need smart algorithm for parallelization



Parallelization with big data & lots of seed points & bad

distribution

- Two extremes:
 - Partition data over processors and pass particles amongst processors
 - Parallel inefficiency!
 - Partition seed points over processors and process necessary data for advection
 - Redundant I/O!

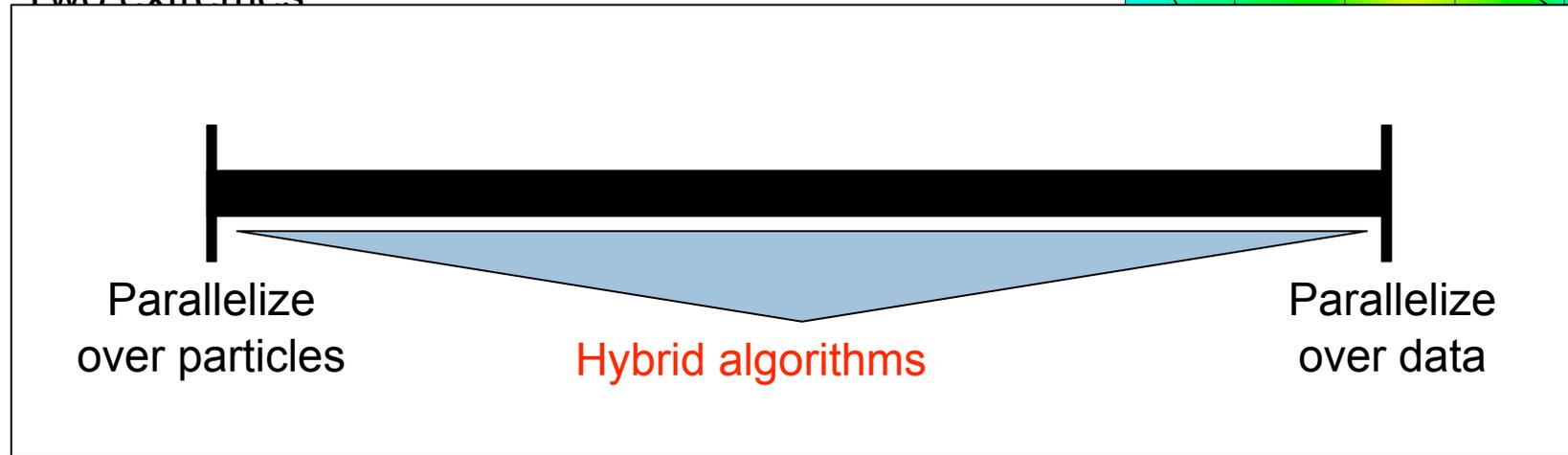


Parallelizing Over	I/O	Efficiency
Data	Good	Bad
Particles	Bad	Good

Parallelization with big data & lots of seed points & bad

distribution

- Two extremes:



Parallelizing Over	I/O	Efficiency
Data	Good	Bad
Particles	Bad	Good

P4 P4 P4 P4 P4

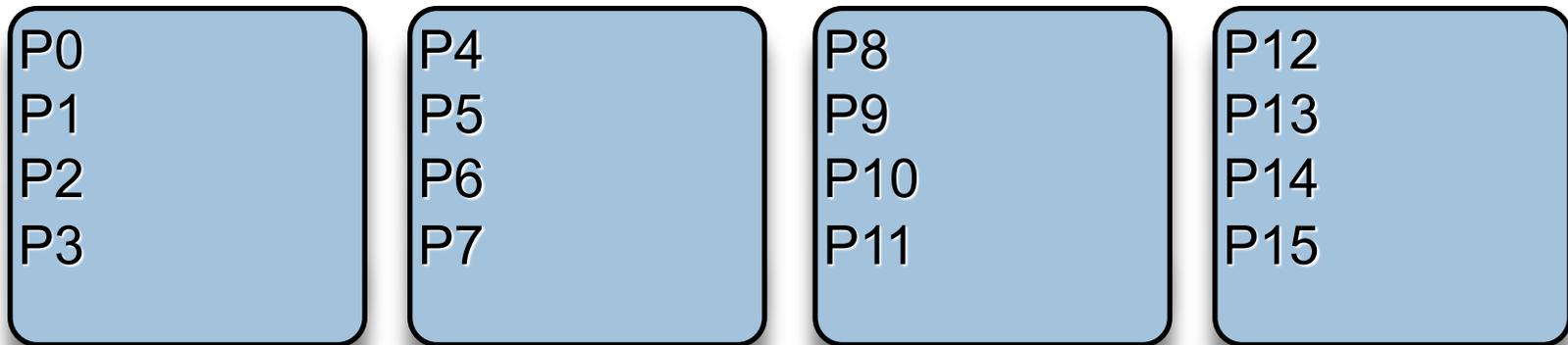
Notional streamline example

The master-slave algorithm is an example of a hybrid technique.

- “Scalable Computation of Streamlines on Very Large Datasets”, Dave Pugmire, et al, SC09
 - ▣ Many of the following slides compliments of Dave Pugmire.
- Algorithm adapts during runtime to avoid pitfalls of parallelize-over-data and parallelize-over-particles.
 - ▣ Nice property for production visualization tools.
- Implemented inside VisIt visualization and analysis package.

Master-Slave Hybrid Algorithm

- Divide processors into groups of N
- Uniformly distribute seed points to each group



Master:

- Monitor workload
- Make decisions to optimize resource utilization

Slaves:

- Respond to commands from Master
- Report status when work complete

Master Process Pseudocode

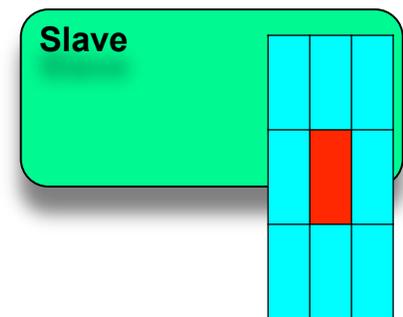
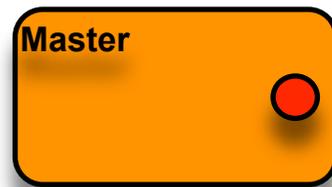
```
Master()
{
  while ( ! done )
  {
    if ( NewStatusFromAnySlave() )
    {
      commands = DetermineMostEfficientCommand()

      for cmd in commands
        SendCommandToSlaves( cmd )
    }
  }
}
```

What are the possible commands?

Commands that can be issued by master

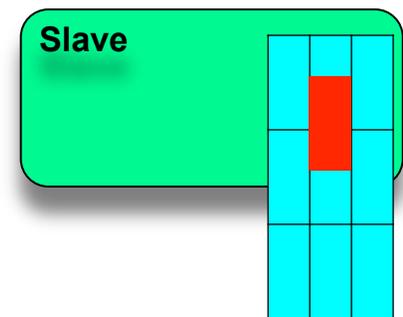
1. **Assign / Loaded Block**
 2. Assign / Unloaded Block
 3. Handle OOB / Load
 4. Handle OOB / Send
- OOB = out of bounds



Slave is given a streamline that is contained in a block that is already loaded

Commands that can be issued by master

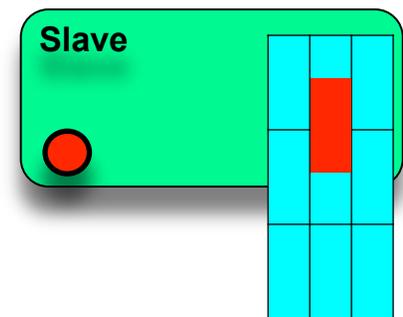
1. Assign / Loaded Block
 - 2. Assign / Unloaded Block**
 3. Handle OOB / Load
 4. Handle OOB / Send
- OOB = out of bounds



Slave is given a streamline
and loads the block

Commands that can be issued by master

1. Assign / Loaded Block
 2. Assign / Unloaded Block
 - 3. Handle OOB / Load**
 4. Handle OOB / Send
- OOB = out of bounds

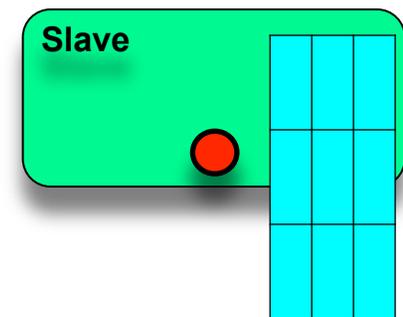


Slave is instructed to load a block. The streamline in that block can then be computed.

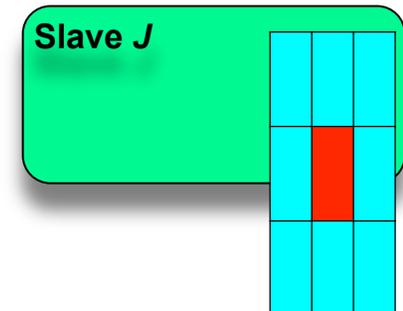
Commands that can be issued by master

1. Assign / Loaded Block
2. Assign / Unloaded Block
3. Handle OOB / Load
- 4. Handle OOB / Send**

OOB = out of bounds



Slave is instructed to send a streamline to another slave that has loaded the block



Master Process Pseudocode

```
Master()
```

```
{
```

```
  while ( ! done )
```

```
  {
```

```
    if ( NewStatusFromAnySlave() )
```

```
    {
```

```
      commands = DetermineMostEfficientCommand()
```

```
      for cmd in commands
```

```
        SendCommandToSlaves( cmd )
```

```
    }
```

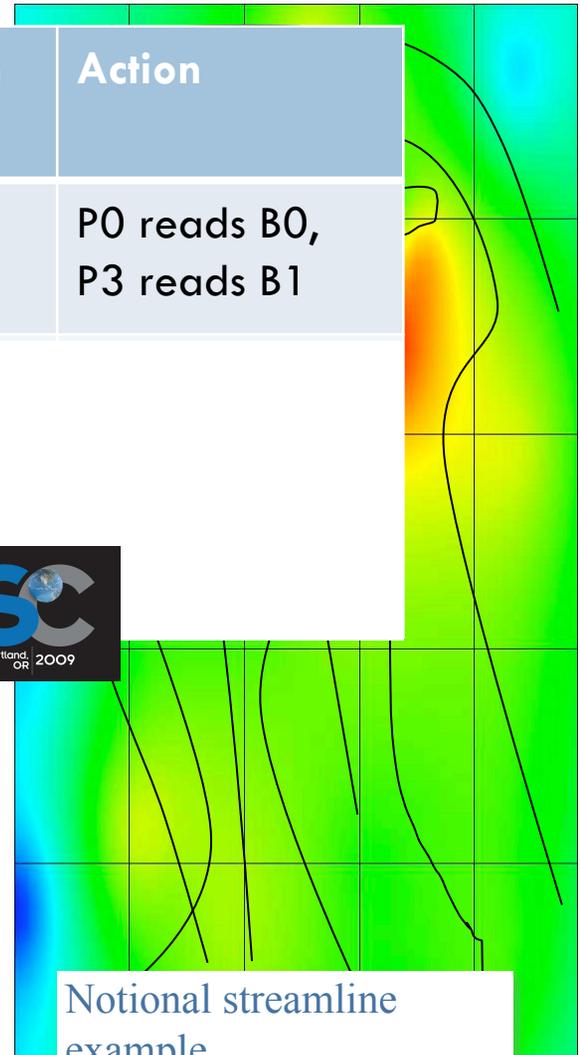
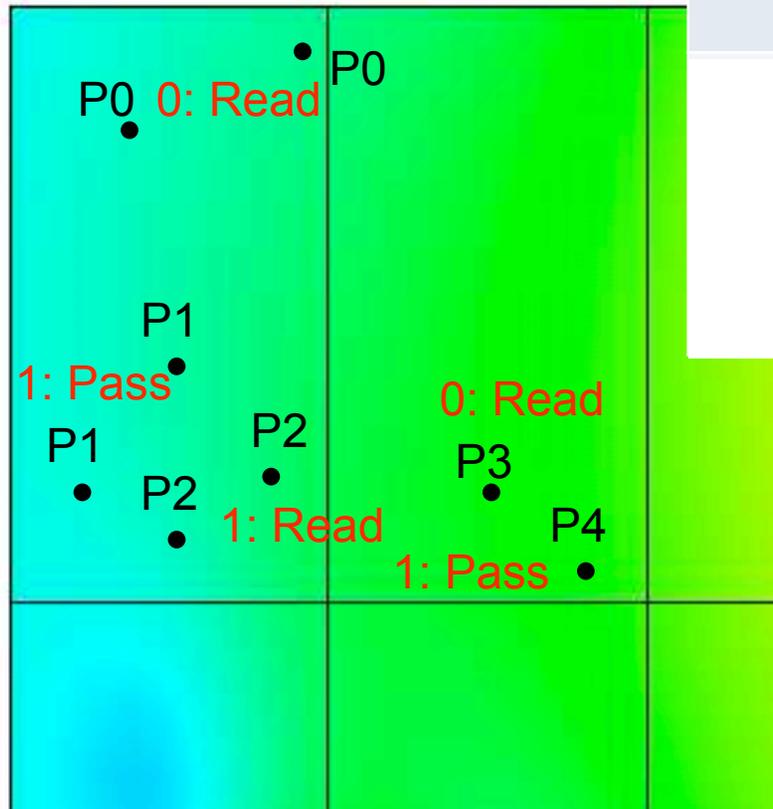
```
  }
```

```
}
```

*** See SC 09 paper
for details**

Master-slave in action

Iteration	Action
0	P0 reads B0, P3 reads B1



Notional streamline example

Master-slave in action

Iteration	Action
0	P0 reads B0, P3 reads B1

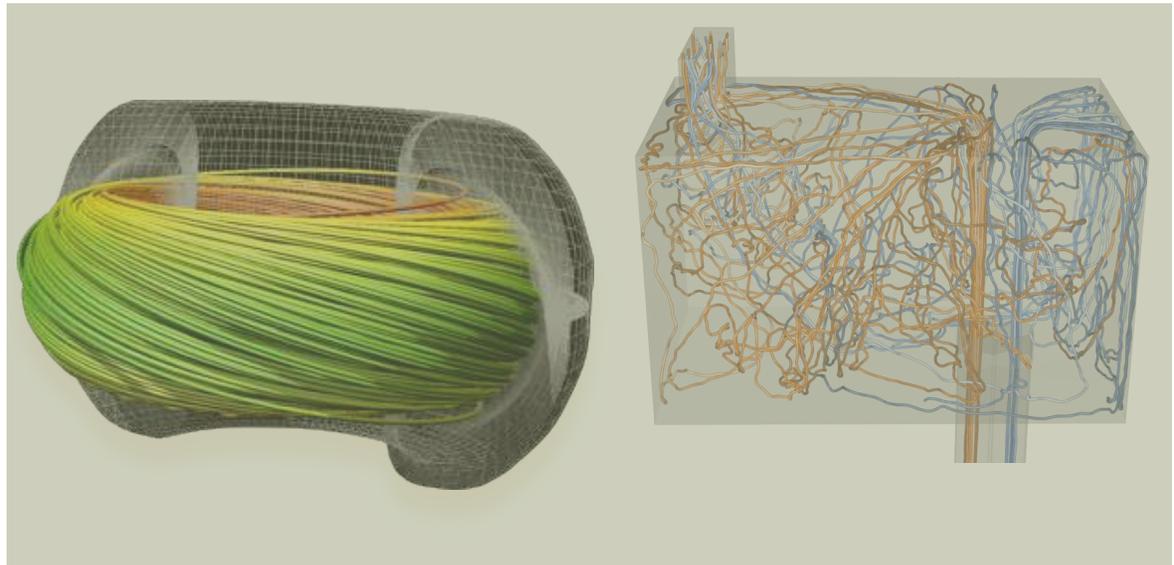
- When to pass and when to read?
- How to coordinate communication? Status? Efficiently?



Notional streamline example

Algorithm Test Cases

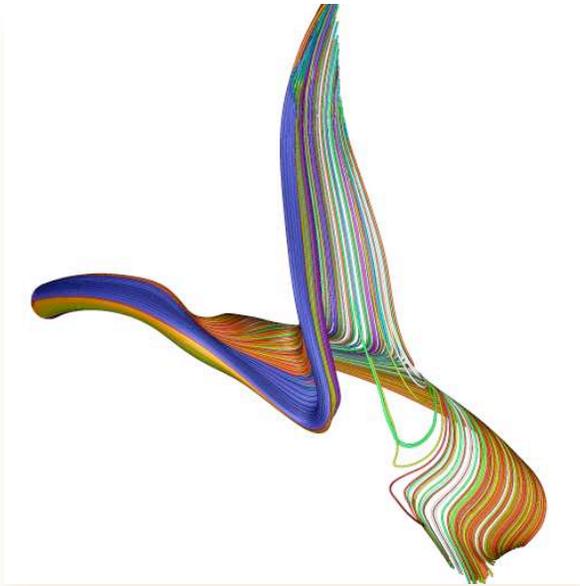
- Core collapse supernova simulation
- Magnetic confinement fusion simulation
- Hydraulic flow simulation



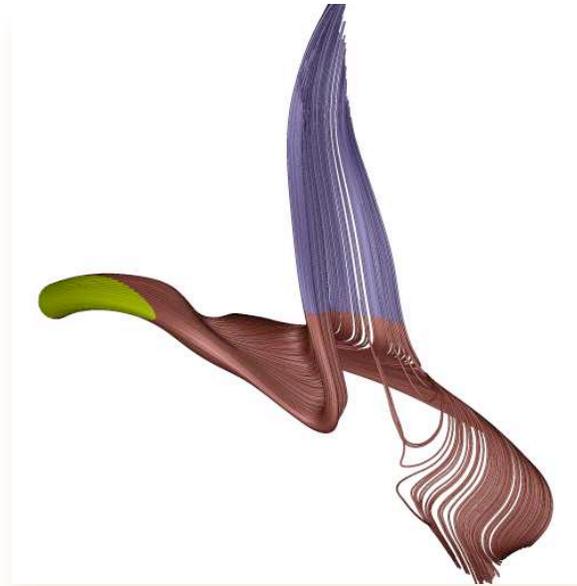
Workload distribution in supernova simulation

Parallelization by:

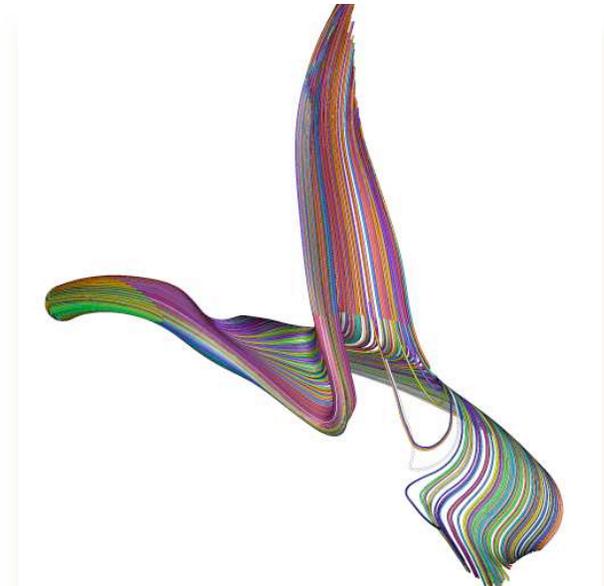
Particles



Data



Hybrid

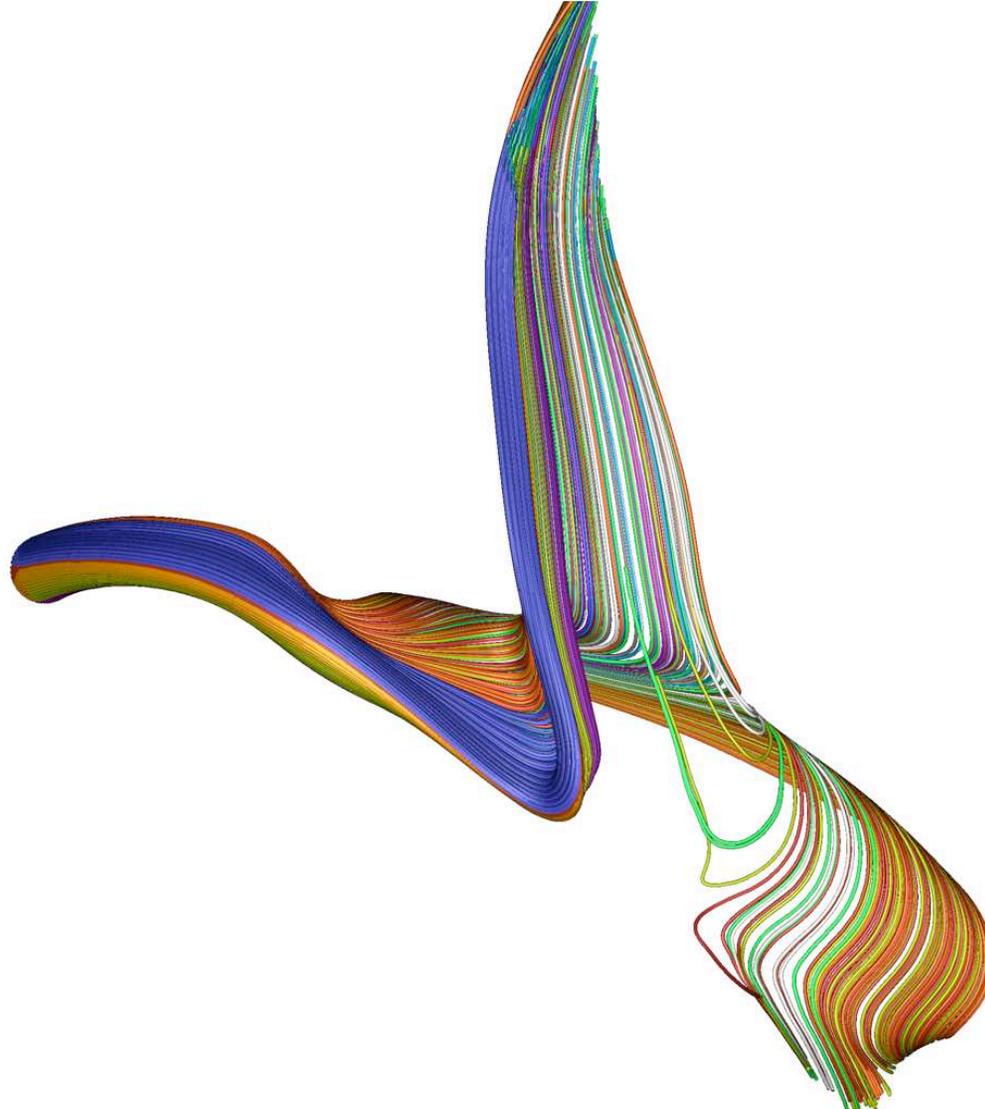


Colored by processor doing integration

Workload distribution in parallelize-over-particles



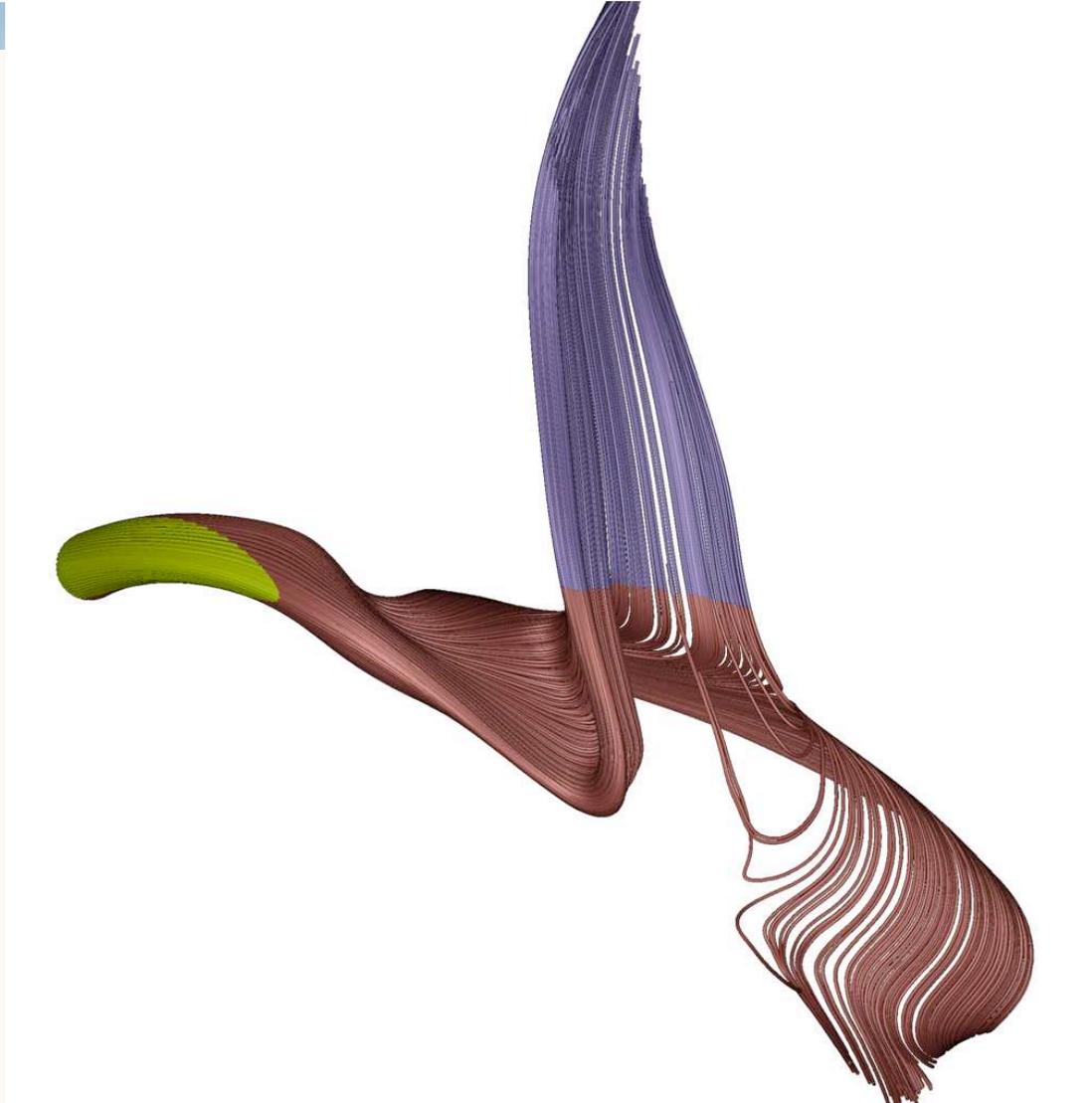
Too much I/O



Workload distribution in parallelize-over-data

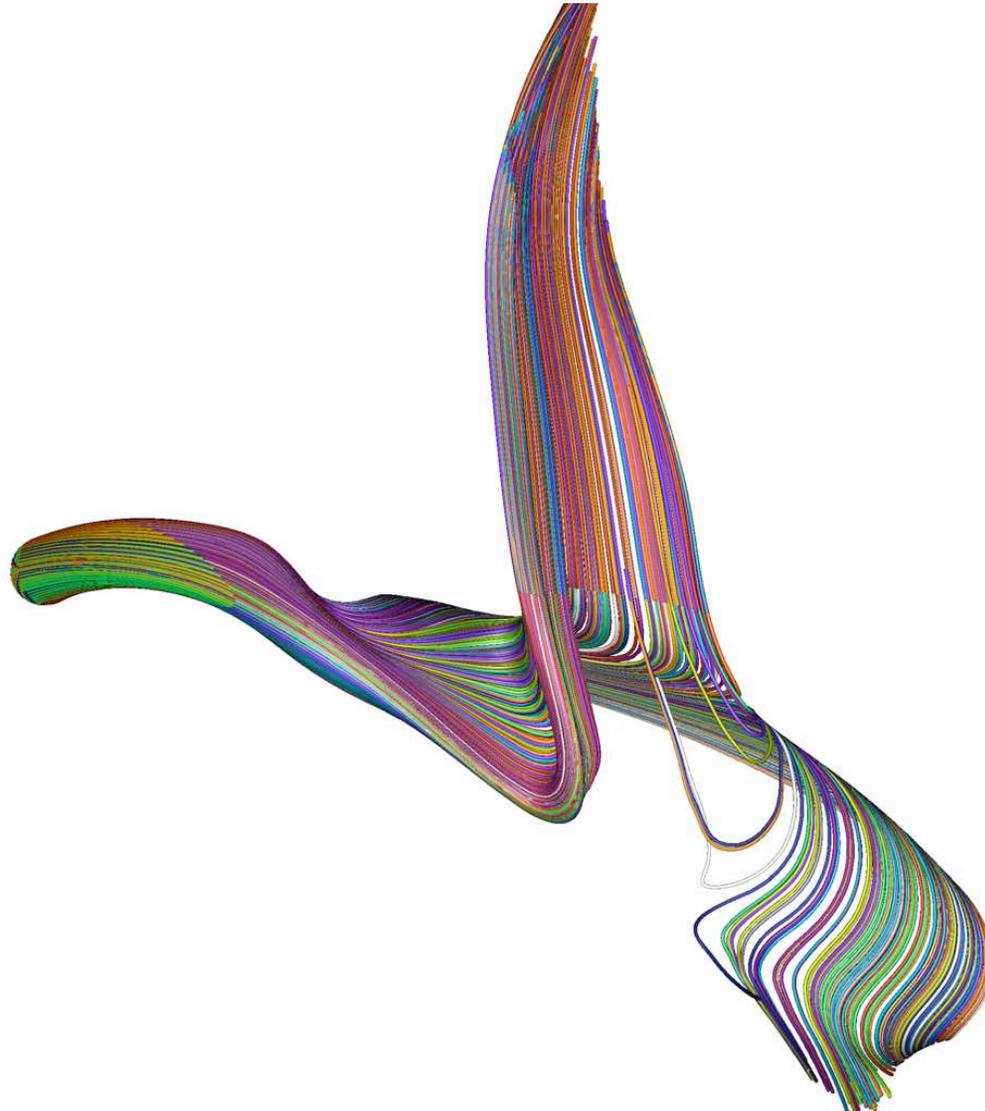


Starvation

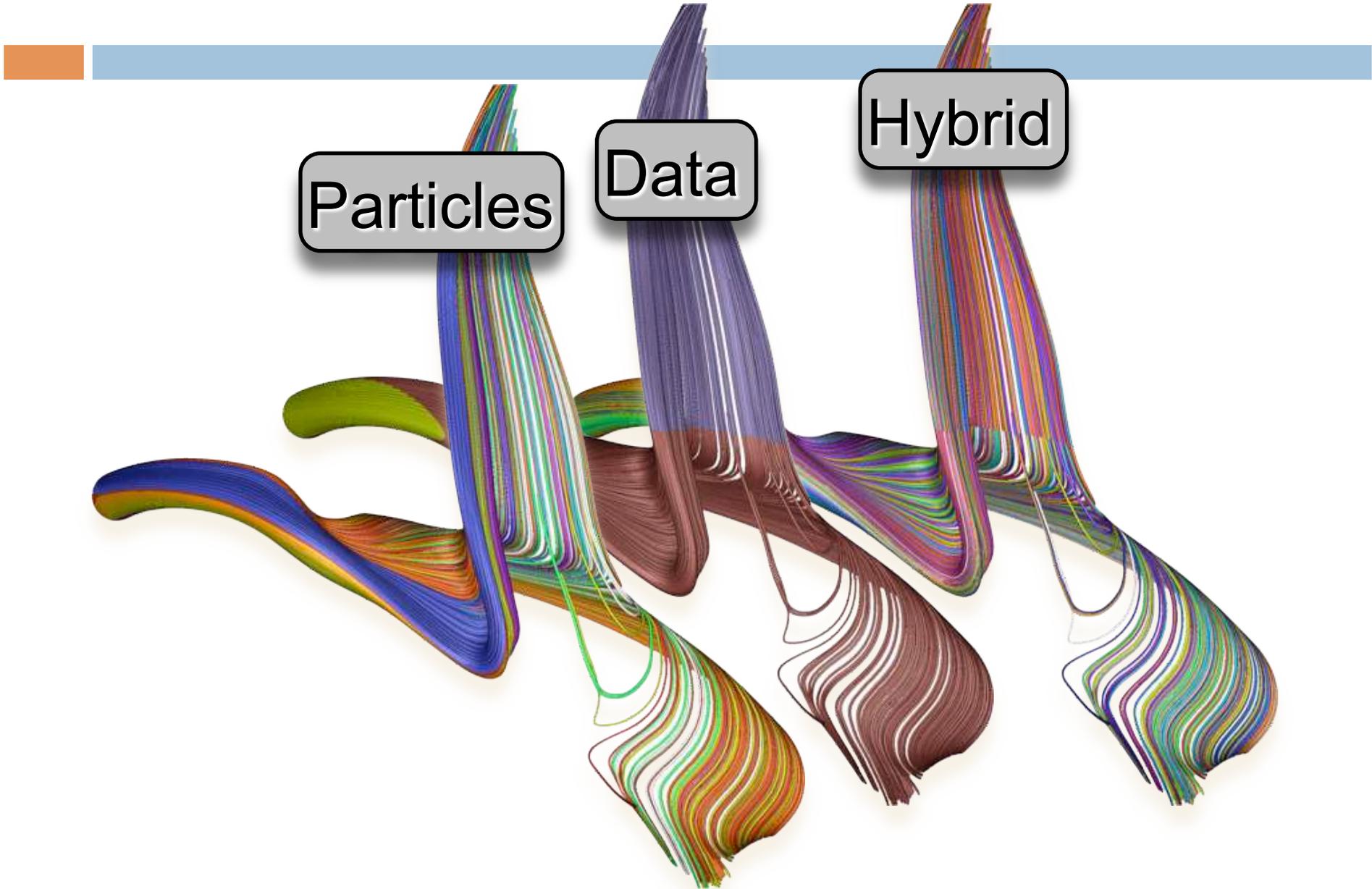


Workload distribution in hybrid algorithm

Just right

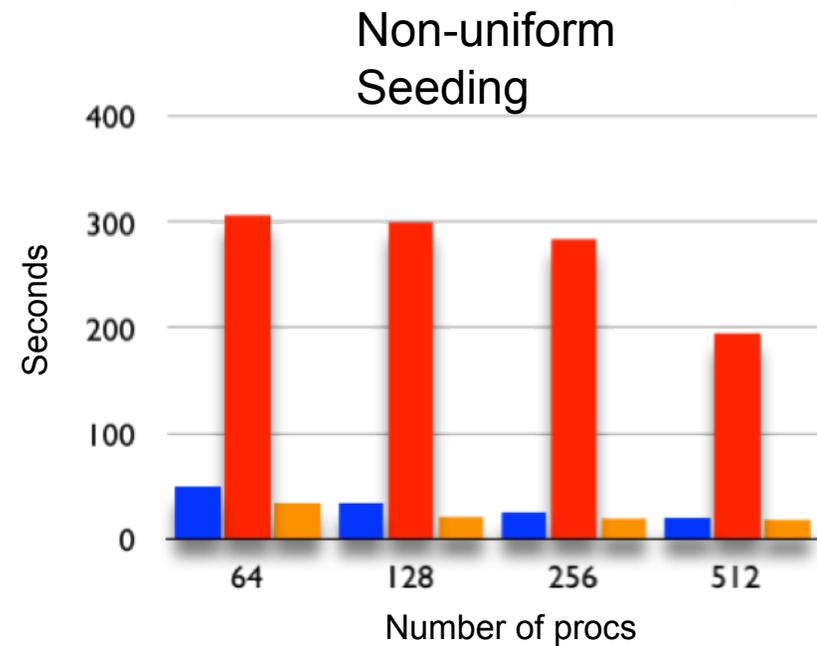
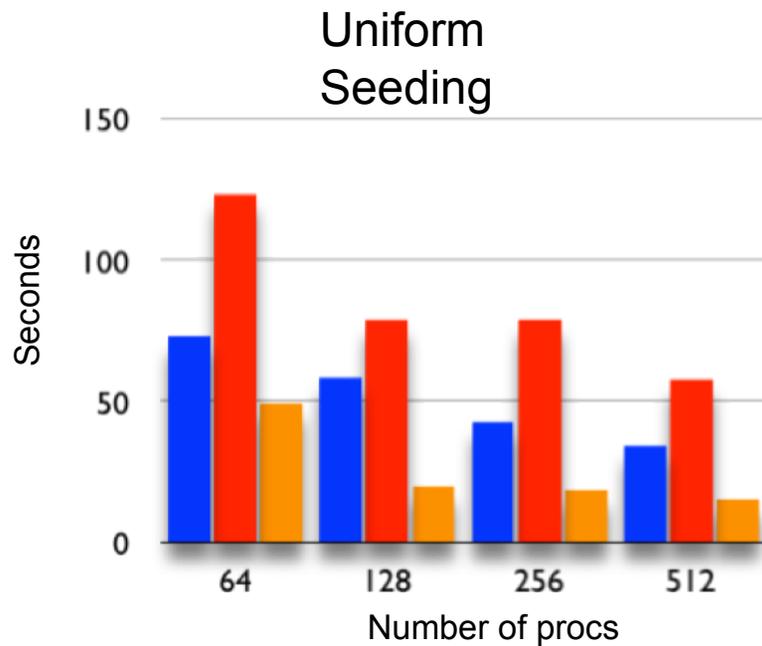
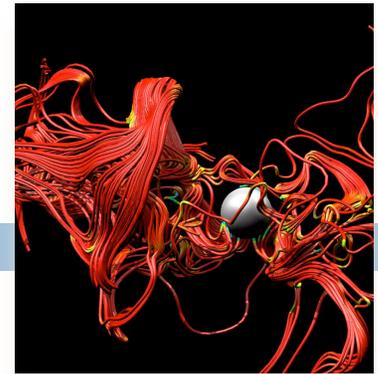


Comparison of workload distribution



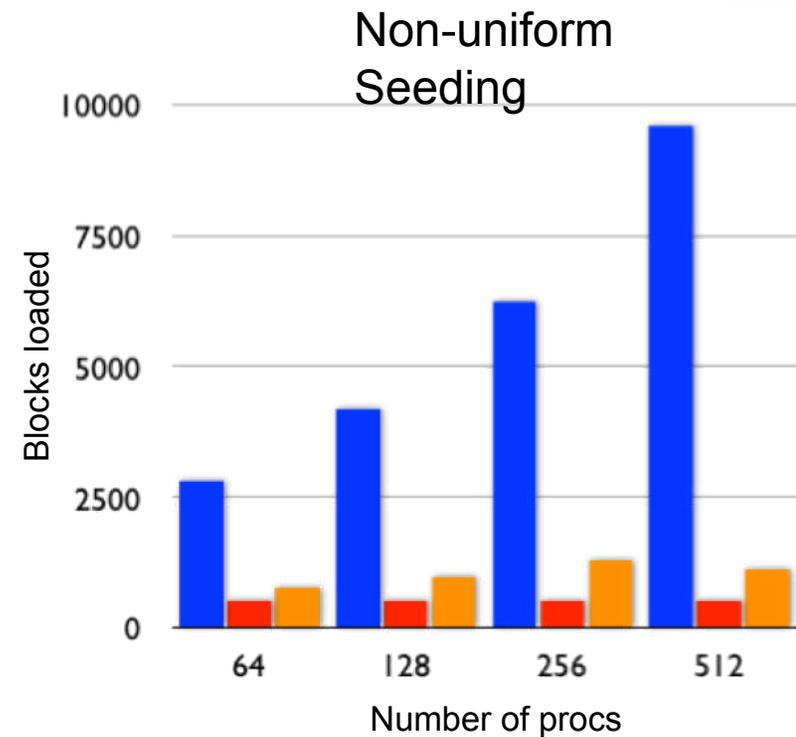
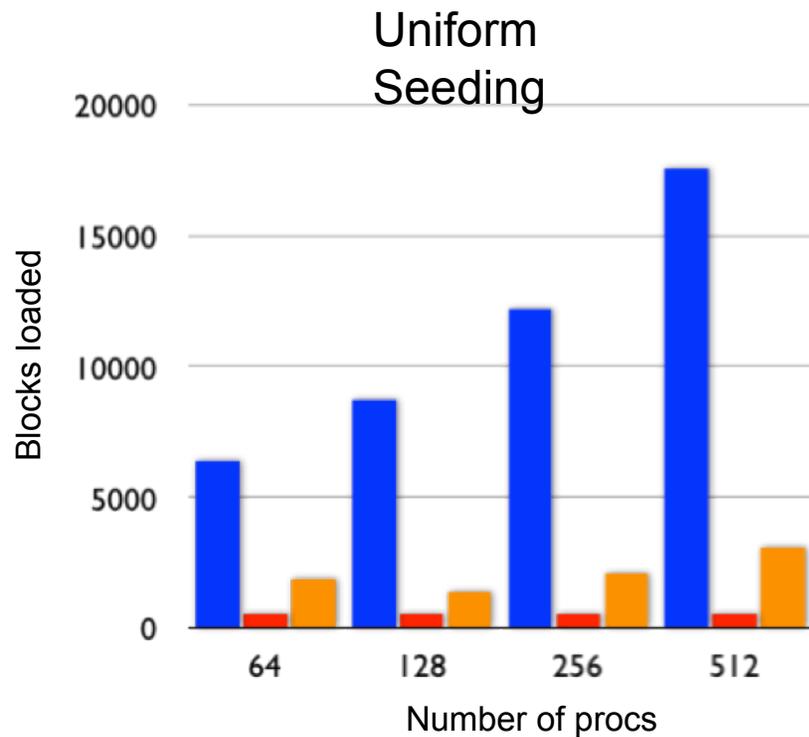
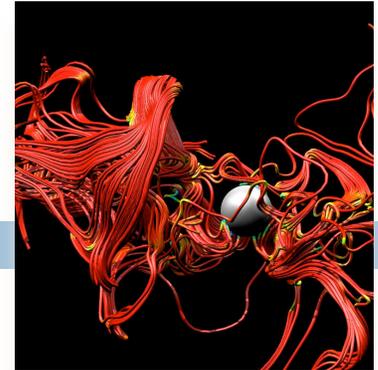
Astrophysics Test Case:

Total time to compute 20,000 Streamlines



■ Particles ■ Data ■ Hybrid

Astrophysics Test Case: Number of blocks loaded



■ Particles ■ Data ■ Hybrid

Summary for Large Data and Parallelization

- The type of parallelization required will vary based on data set size, number of seeds, seed locations, and vector field complexity
- Parallelization may occur via parallelization-over-data, parallelization-over-particles, or somewhere in between (master-slave). Hybrid algorithms have the opportunity to de-emphasize the pitfalls of the traditional techniques.
- Note that I said nothing about time-varying data...

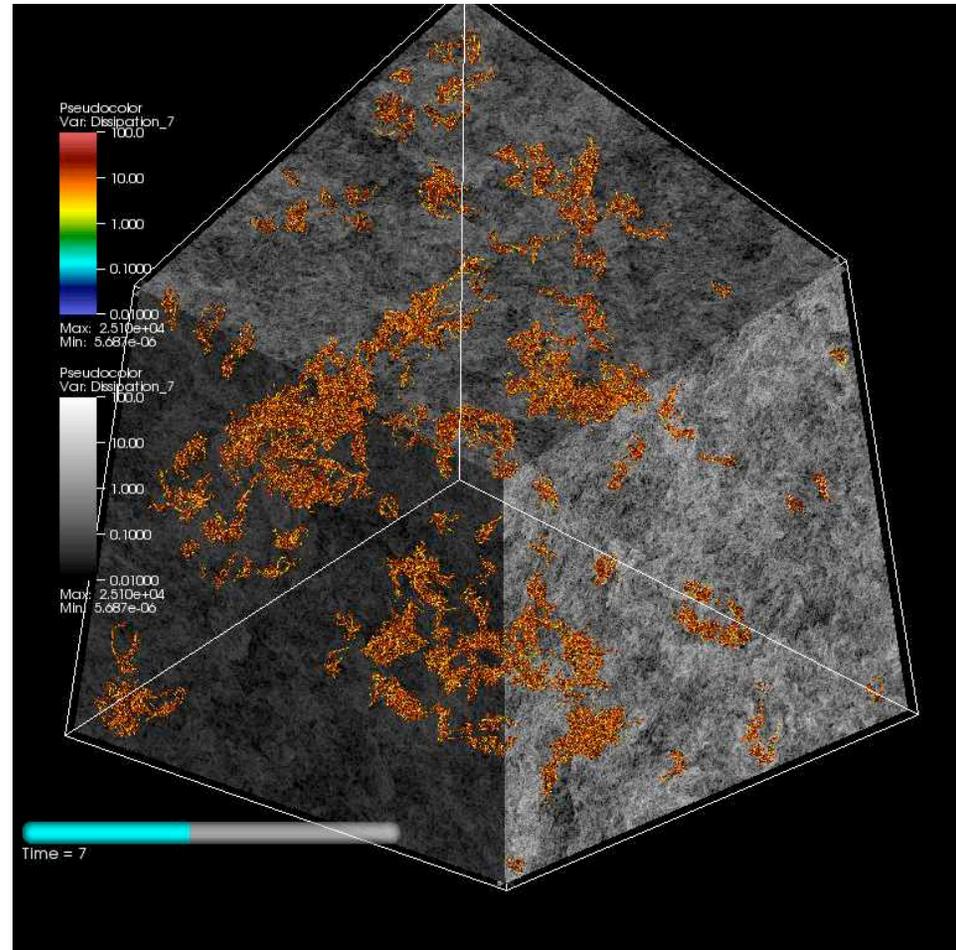
Outline



- Volume rendering
- Particle advection
- Connected components & line scans

Visualizing and Analyzing Large-Scale Turbulent Flow

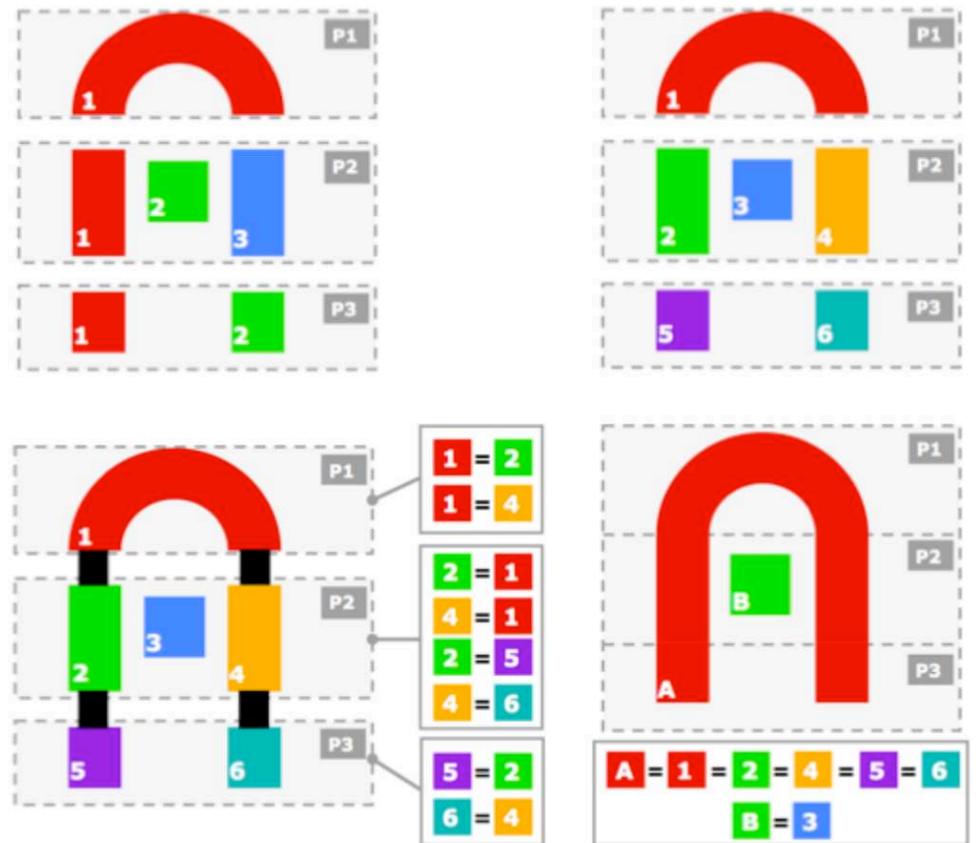
- Detect, track, classify, and visualize features in large-scale turbulent flow.
- Analysis effort by Kelly Gaither (TACC), Hank Childs (LBNL), & more...
- Stresses two algorithms that are difficult in a distributed memory parallel setting:
 1. Can we identify connected components?
 2. Can we characterize their shape?



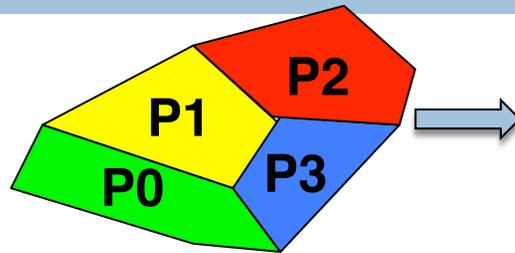
VisIt calculated connected components on a $4K^3$ turbulence data in parallel using TACC's Longhorn machine. 2 million components were initially identified and then the map expression was used to select only the components that had total volume greater than 15. Data courtesy of P.K. Yeung & and Diego Donzis

Identifying connected components in parallel is difficult.

- Hard to do efficiently
- Tremendous bookkeeping problem.
- 4 stage algorithm that finds local connectivity and then merges globally.



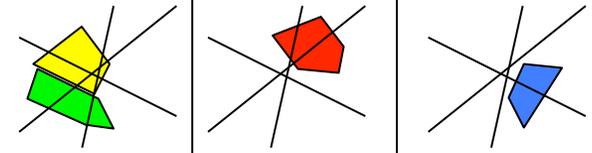
We used shape characterization to assist our feature tracking.



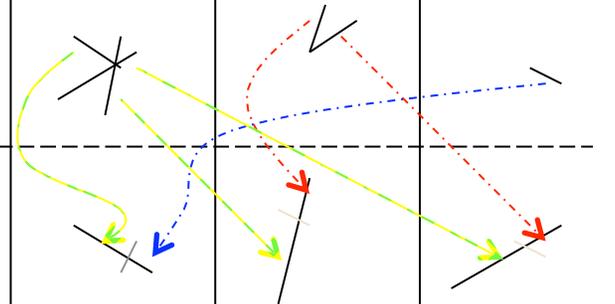
- Shape characterization metric: chord length distribution
- Difficult to perform efficiently in a distributed memory setting

Line Scan Filter

1) Choose Lines



2) Calculate Intersections

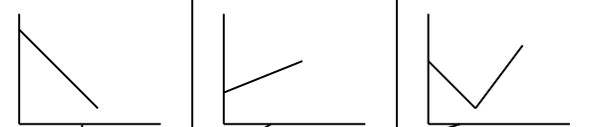


3) Segment redistribution



Line Scan Analysis Sink

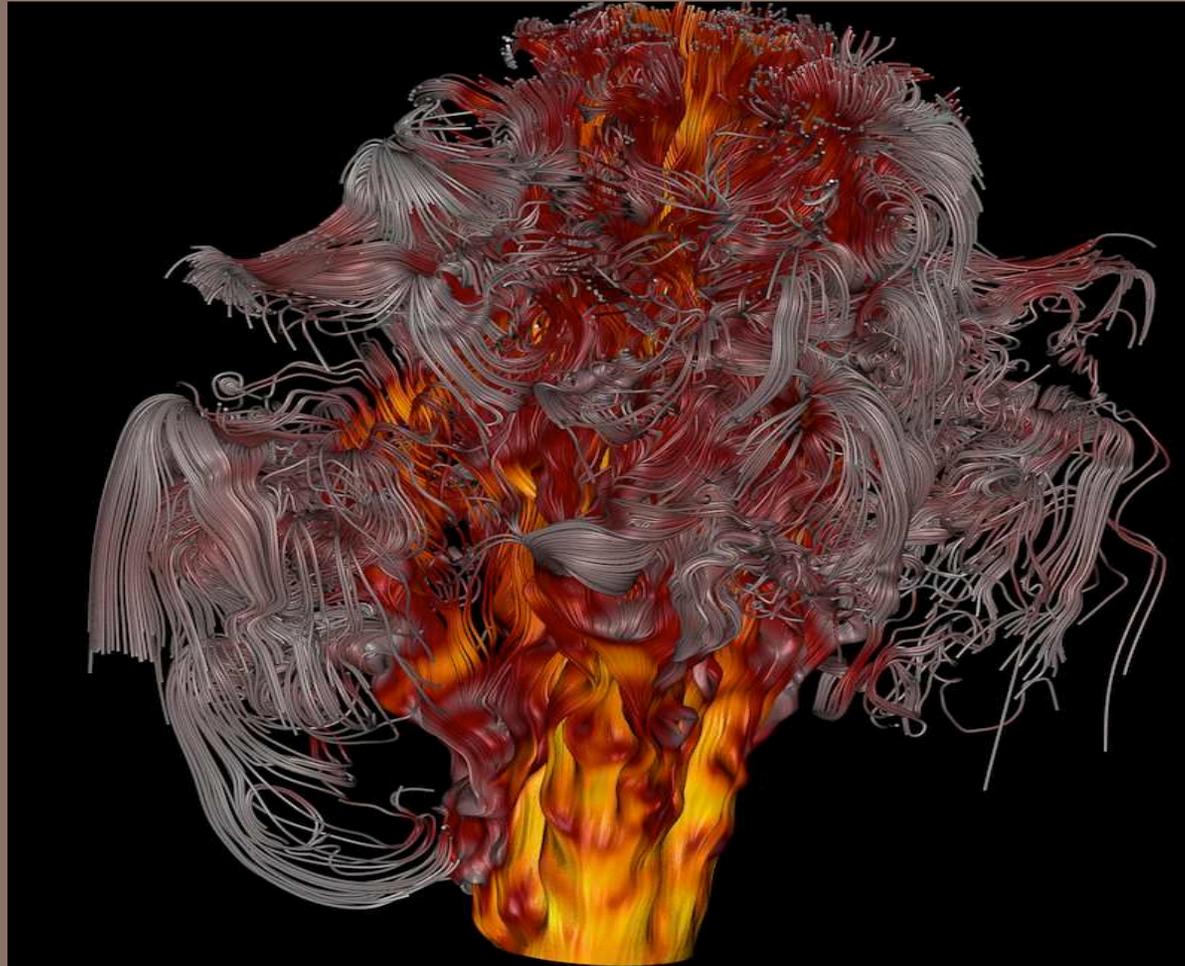
4) Analyze lines



5) Collect results



Hybrid Parallelism



June 13, 2011

Hank Childs, Lawrence Berkeley Lab & UC Davis

Outline



- Overview of Hybrid Parallelism
- Examples
 - ▣ Volume rendering
 - ▣ Streamlines

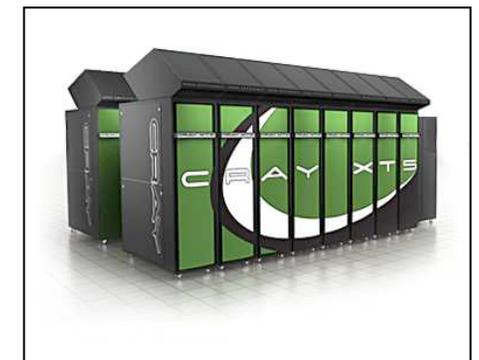
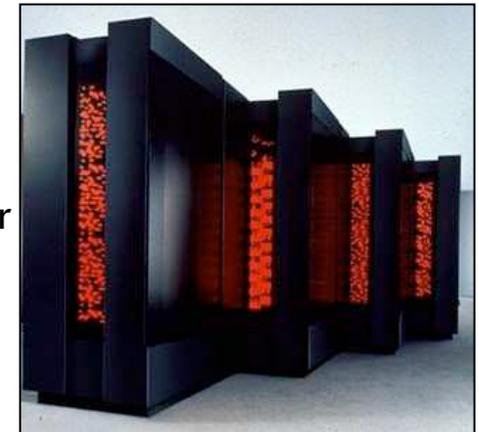
Outline



- Overview of Hybrid Parallelism
- Examples
 - ▣ Volume rendering
 - ▣ Streamlines

History of Parallelism

- Mid 1970s-Early 1990s:
 - ▣ Vector machines: Cray 1 ... NEC SX
 - ▣ Vectorizing Fortran compilers help optimize $a[i]=b[i]*x+c$.
- Early 1990s-present:
 - ▣ The rise of the MPP based on the commodity microprocessor. Cray T3D, TM CM1, CM2, CM5, etc.
 - ▣ Message Passing Interface (MPI) becomes the gold standard for building/running parallel codes on MPPs.
- Early 1990s-Early 2000s:
 - Shared memory parallelism (e.g. SGI)
- Mid 2000s-present:
 - ▣ Rise of the multi-core CPU, GPU. AMD Opteron, Intel Nehalem, Sony Cell BE, NVIDIA G80, etc.
 - ▣ Large supercomputers comprised of lots of multi-core CPUs.
 - ▣ Shared memory programming on a node: pthreads, OpenMP; data parallel languages (CUDA); global shared memory languages (UPC) and utilities (CAF).



Hybrid parallelism: MPI + ?

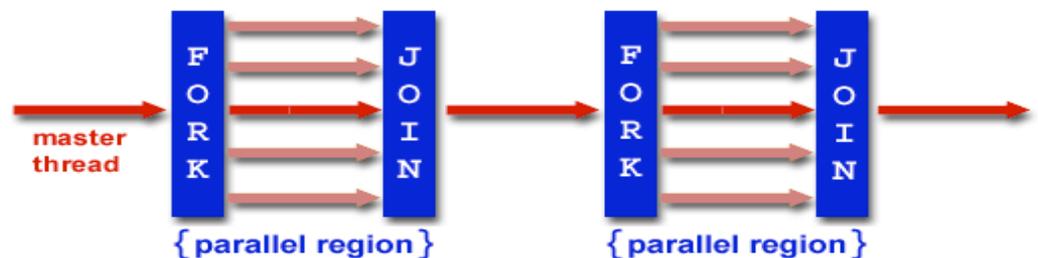
- Hybrid-parallelism blends distributed- and shared-memory parallelism concepts.
 - ▣ Use distributed memory techniques across nodes & shared memory techniques within a node.
- Distributed memory parallelism
 - ▣ MPI is the gold standard
- Shared memory parallelism
 - ▣ Pthreads
 - ▣ OpenMP
 - ▣ CUDA / OpenCL
 - ▣ More...

Pthreads = POSIX threads

- Pthreads: standard, portable library available with C programming on UNIX
- Thread = “independent stream of instructions that can be scheduled to run by the operating system”
- 4 major groups of subroutines in Pthreads API:
 - ▣ Thread management, mutexes, condition variables, synchronization
- Threads created and destroyed dynamically
- Memory shared between the threads
- Each thread may execute a totally different subroutine.

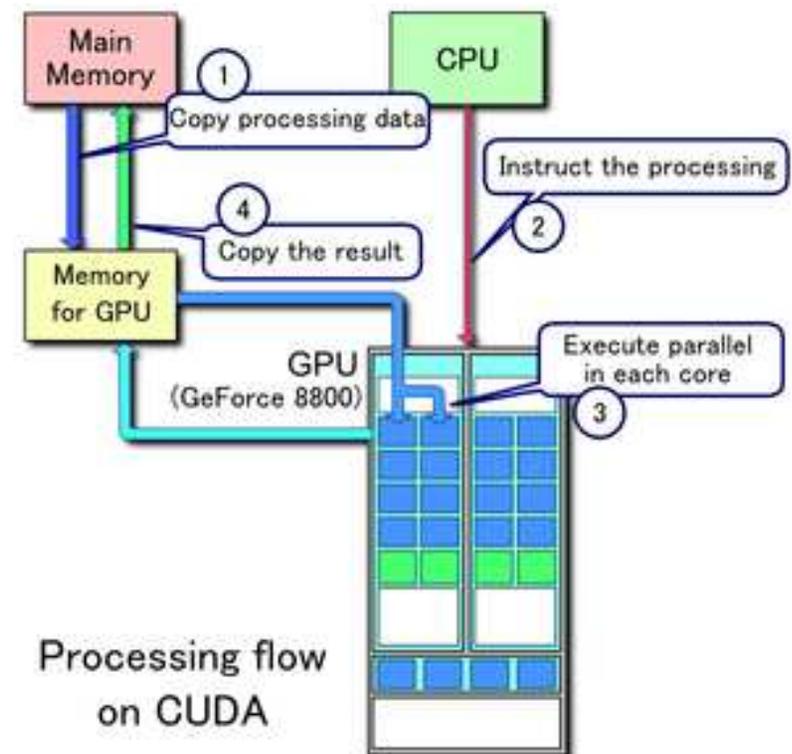
OpenMP = Open Multi-Processing

- OpenMP: shared-memory parallel programming in C/C++/Fortran on Unix, Windows NT, and more.
- Defined by a group of major computer hardware and software vendors.
- Portable, scalable model that gives shared-memory parallel programmers a simple and flexible interface.
- Realized through compiler directives.
- Follows a fork/join model.



CUDA = Compute Unified Device Architecture

- Allows developers to program NVIDIA GPUs by giving them access to its virtual instruction set and memory of the parallel computational elements.
- Recursion-free, function-pointer-free subset of the C language.



OpenCL = Open Computing Language



- Developed by Apple, AMD, IBM, Intel, and Nvidia, and transferred to the Khronos Group.
- Programming is similar to CUDA, although widely regarded to be a less mature environment.
- Capable of supporting x86, Nvidia, and ATI cards.

Hybrid Parallelism on Large, Multi-core Platforms

- Why hybrid parallelism?
 - ▣ MPI-only approaches for parallel visualization may not work well in future: 100-1000 cores per node.
 - ▣ Exascale machines will likely have $O(1M)$ nodes
- Questions when considering hybrid parallelism:
 - ▣ Will MPI-only work?
 - ▣ Will hybrid work?
 - ▣ Are there performance gains with hybrid? Losses?

Research in Hybrid Parallelism

□ Caveats

- Relatively new research area, not a great deal of published work.
- Studies focus on “solvers,” not vis/graphics.
- State of hybrid parallel visualization: lots of work to do

□ Fundamental questions:

- How to map algorithm onto a complex memory, communication hierarchy?
- What is the right balance of distributed- vs. shared-memory parallelism? How does balance impact performance?

Research in Hybrid Parallelism

- Conclusions of these previous works:
 - What is best? Answer: it depends.
 - Many factors influence performance/scalability:
 - Synchronization overhead.
 - Load balance (intra- and inter-node).
 - Communication overhead and patterns.
 - Memory access patterns.
 - Fixed costs of initialization.
 - Number of runtime threads.

Outline



- Overview of Hybrid Parallelism
- Examples
 - ▣ Volume rendering
 - ▣ Streamlines

Hybrid Parallelism for Volume Rendering on Large, Multi-core Platforms

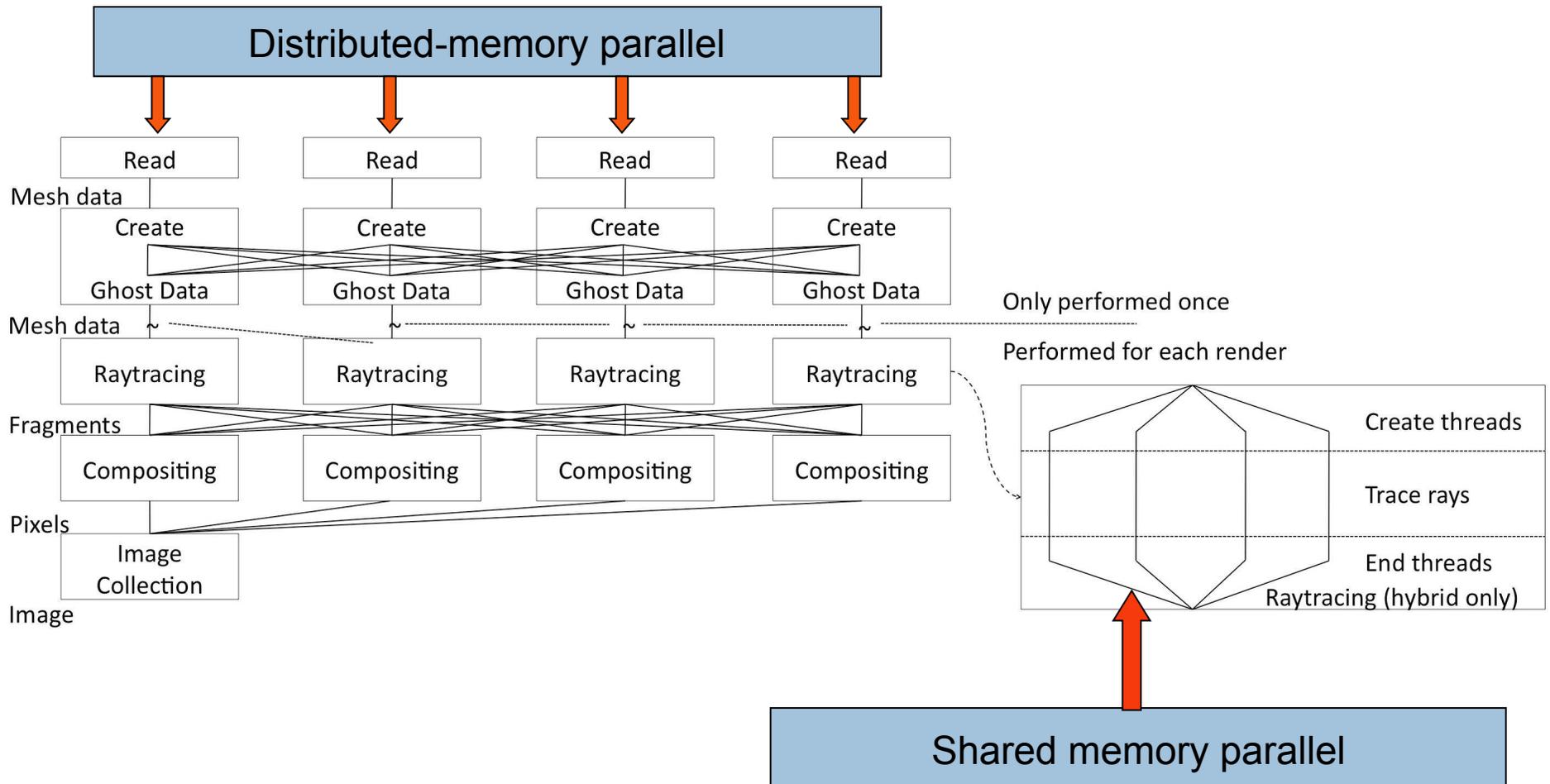
- Does hybrid-parallelism work for ray casted volume rendering at extreme concurrency? If so, how well?
 - ▣ Ask same questions the HPC folks do:
 - How to map algorithm to hybrid parallel space?
 - How does performance compare with MPI-only implementation?
- Study:
 - ▣ Compare MPI-only, MPI+threads, MPI+OpenMP at 216K concurrency
- Results:
 - ▣ Experiment to compare performance shows favorable characteristics of hybrid-parallel, especially at very high concurrency.

Hybrid Parallelism Versus Hybrid Volume Rendering

- Hybrid volume rendering:
 - Refers to mixture of object- and image-order techniques to do volume rendering.
 - A two-stage algorithm, heavy communication load between stages.
- Hybrid parallelism:
 - Refers to mixture of shared and distributed memory approaches.
- (We are doing both.)

Hybrid Parallel Volume Rendering

□ Our hybrid-parallel architecture:

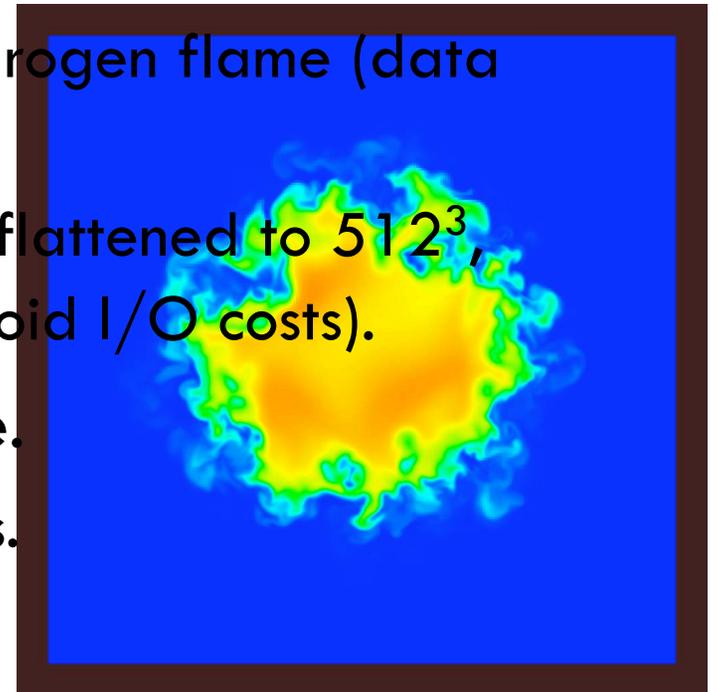


Experiment Overview

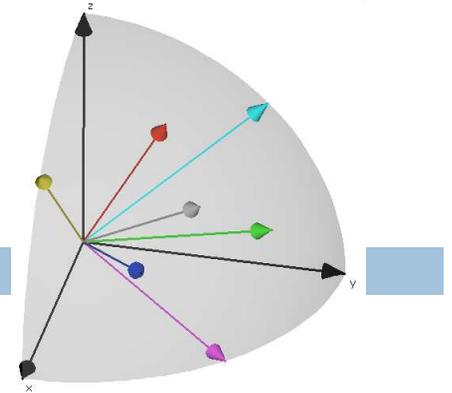
- Thesis: hybrid-parallel will exhibit favorable performance, resource utilization characteristics compared to traditional approach.
- Strong scaling study: hold problem size constant, vary amount of resources.
 - ▣ As we increase the number of procs/cores, each proc/core works on a smaller-sized problem.
 - ▣ Time-to-solution should drop.

Experiment: Platform and Source Data

- Platform: JaguarPF, a Cray XT5 system at ORNL
 - 18,688 nodes, dual-socket, six-core AMD Opteron (224K cores)
- Source data:
 - Combustion simulation results, hydrogen flame (data courtesy J. Bell, CCSE, LBNL)
 - Effective AMR resolution: 1024^3 , flattened to 512^3 , runtime upscaled to 4608^3 (to avoid I/O costs).
- Target image size: 4608^2 image.
 - Want approx 1:1 voxels to pixels.



Experiment – The Unit Test

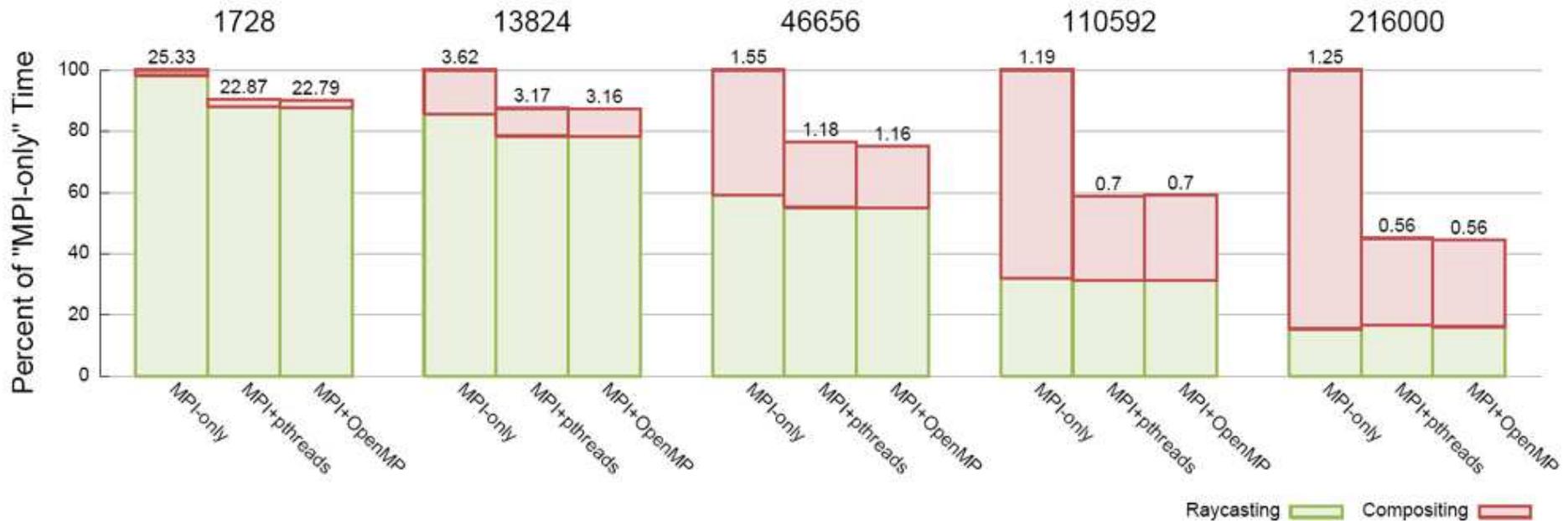


Raycasting time: view/data dependent

- ▣ Execute from 10 different prescribed views: forces with- and cross-grained memory access patterns.
- ▣ Execute 10 times, result is average of all.
- ▣ Compositing
 - ▣ Five different ratios of compositing PEs to rendering PEs.
- ▣ How/what to measure?
 - ▣ Memory footprint
 - ▣ right after initialization.
 - ▣ for data blocks and halo exchange
 - ▣ Absolute runtime and scalability of raycasting and compositing
 - ▣ All across a wide range of concurrencies.
 - ▣ Remember: we're concerned about what happens at extreme

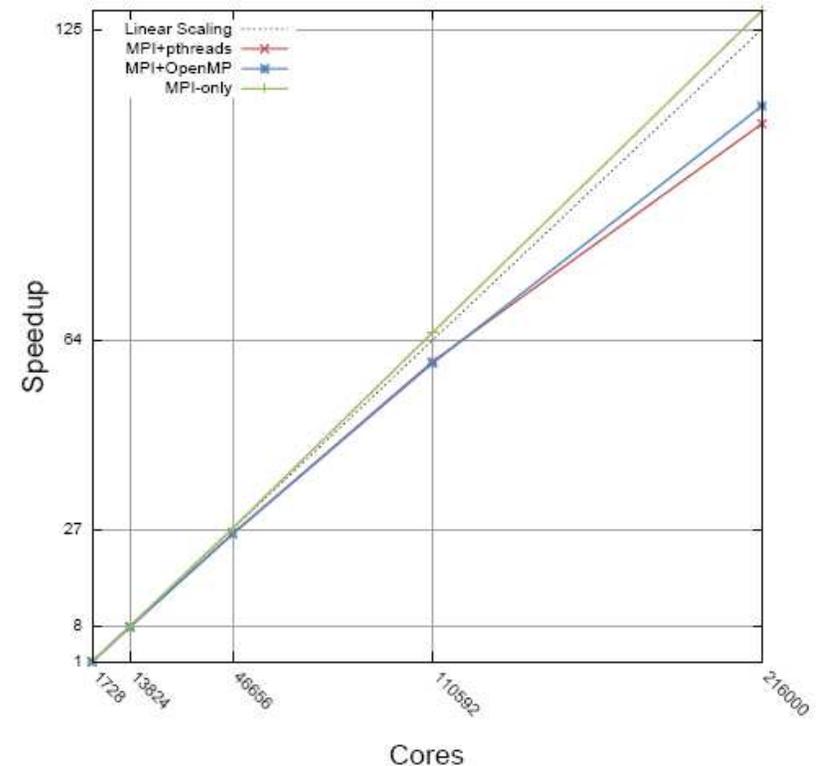
Absolute Runtime

- -hybrid outperforms -only at every concurrency level.
 - ▣ At 216K-way parallel, -hybrid is more than twice as fast as -only.



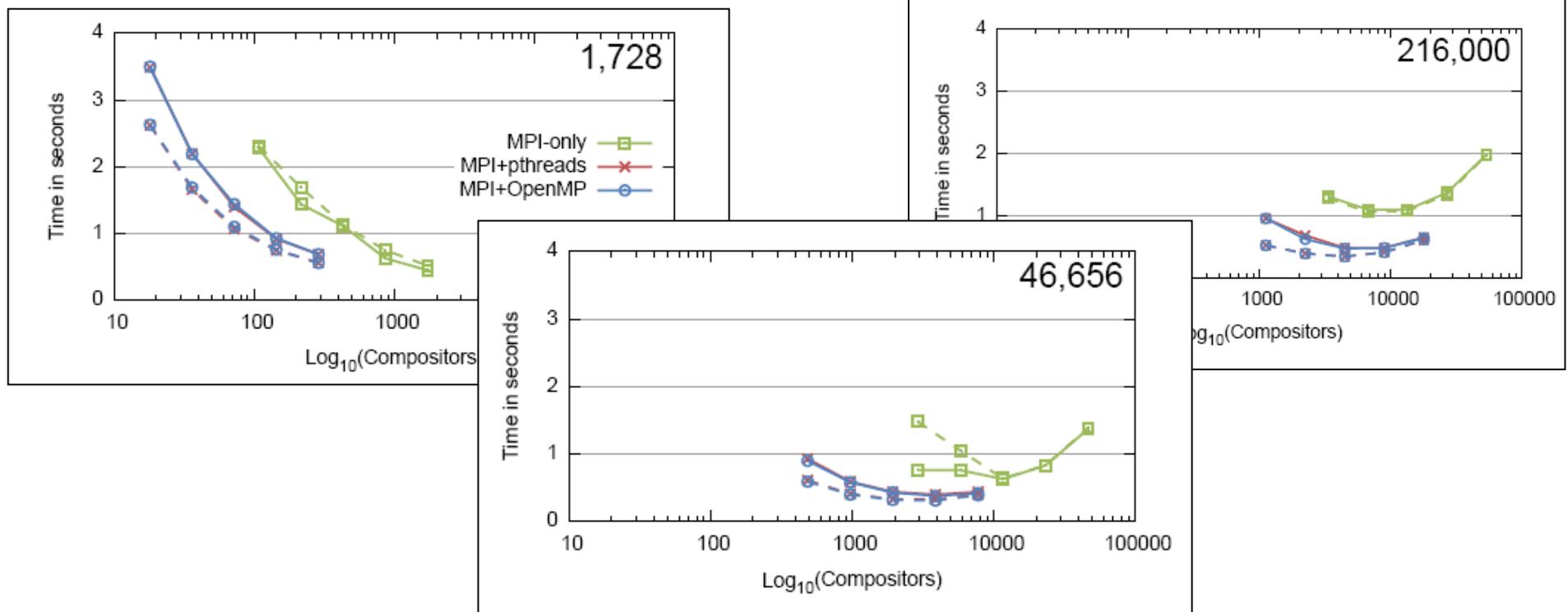
Scalability – Raycasting Phase

- Near linear scaling since no interprocess communication.
- -hybrid shows sublinear scaling due to oblong block shape.
- -only shows slightly better than linear due to reduced work caused by perspective foreshortening.



Scalability – Compositing

- How many compositors to use?
 - Previous work: 1K to 2K for 32K renderers (Peterka, 2009).
 - Our work: above $\sim 46K$ renderers, 4K to 8K works better.
 - -hybrid cases always performs better: fewer messages.



Memory Use – Data Decomposition

- 16GB RAM per node
 - Sets lower bound on concurrency for this problem size: 1728-way parallel (no virtual memory!).
- Source data (1x), gradient field (3x)
- Want cubic decomposition.
 - 1x2x3 block configuration per socket for –only.
- -hybrid has ~6x data per socket than –only
 - Would prefer to run study on 8-core CPUs to maintain cubic shape

MPI-only		MPI-hybrid		Memory Per Node
MPI PEs	Block Dimensions	MPI PEs	Block Dimensions	
$12^3=1728$	$384 \times 384 \times 384$	288	$384 \times 768 \times 1152$	10368MB
$24^3=13824$	$192 \times 192 \times 192$	2304	$192 \times 384 \times 576$	1296MB
$36^3=46656$	$128 \times 128 \times 128$	7776	$128 \times 256 \times 384$	384MB
$48^3=110592$	$96 \times 96 \times 96$	18432	$96 \times 192 \times 288$	162MB
$60^3=216000$	$76 \times 76 \times 76$	36000	$76 \times 153 \times 230$	80.4MB / 81.6MB

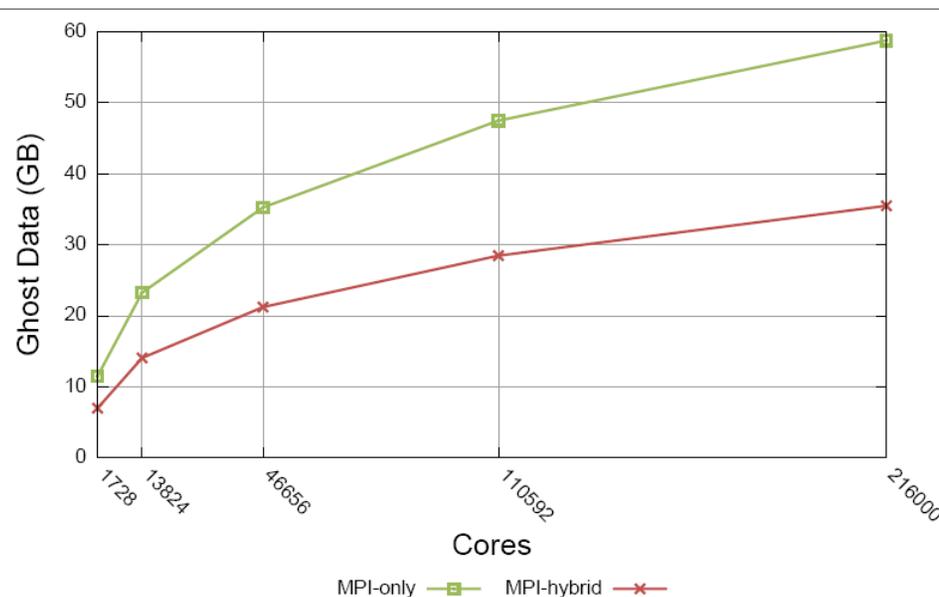
Memory Use – MPI_Init()

- Per PE memory:
 - ▣ About the same at 1728, over 2x at 216000.
- Aggregate memory use:
 - ▣ About 6x at 1728, about 12x at 216000.

Cores	Mode	MPI PEs	MPI Runtime Memory Usage		
			Per PE (MB)	Per Node (MB)	Aggregate (GB)
1728	MPI-hybrid	288	67	133	19
1728	MPI-only	1728	67	807	113
13824	MPI-hybrid	2304	67	134	151
13824	MPI-only	13824	71	857	965
46656	MPI-hybrid	7776	68	136	518
46656	MPI-only	46656	88	1055	4007
110592	MPI-hybrid	18432	73	146	1318
110592	MPI-only	110592	121	1453	13078
216000	MPI-hybrid	36000	82	165	2892
216000	MPI-only	216000	176	2106	37023

Memory Use – Ghost Data

- Two layers of ghost cells required for this problem:
 - One for trilinear interpolation during ray integration loop.
 - Another for computing a gradient field (central differences) for shading.
- Hybrid approach uses fewer, but larger data blocks.
 - ~40% less memory required for ghost data (smaller surface area)
 - Reduced communication costs

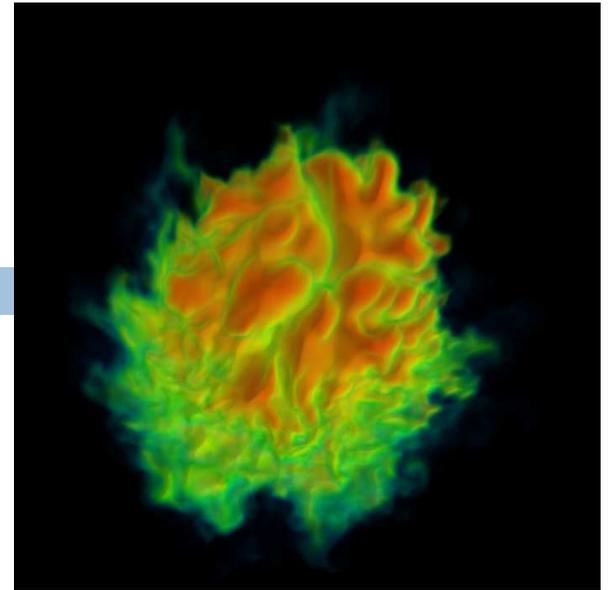


Comparing our results to classic hybrid parallel factors

- Factors in hybrid parallelism performance
 - Synchronization overhead.
 - Had two MPI tasks per node, not one, to prevent work spreading across CPU.
 - Load balance (intra- and inter-node).
 - Studied extensively, comes down to communication
 - Communication overhead and patterns.
 - Hybrid implementation naturally lends itself to superior communication pattern
 - Memory access patterns.
 - Not presented
 - Fixed costs of initialization.
 - Ghost data generation cost reduced with hybrid parallelism
 - MPI initialization cost reduced with hybrid parallelism
 - Number of runtime threads.
 - Not studied

Summary of Results

- Absolute runtime: -hybrid twice as fast as -only at 216K-way parallel.
- Memory footprint: -only requires 12x more memory for MPI initialization than -hybrid
 - ▣ Factor of 6x due to 6x more MPI PEs.
 - ▣ Additional factor of 2x at high concurrency, likely a vendor MPI implementation (an N^2 effect).
- Communication traffic:
 - ▣ -hybrid performs 40% less communication than -only for ghost data setup.
 - ▣ -only requires 6x the number of messages for compositing.
- Image: 4608^2 image of a $\sim 4500^3$ dataset generated using 216,000 cores on JaguarPF in ~ 0.5 s (not counting I/O time).



Outline



- Overview of Hybrid Parallelism
- Examples
 - Volume rendering
 - **Streamlines**

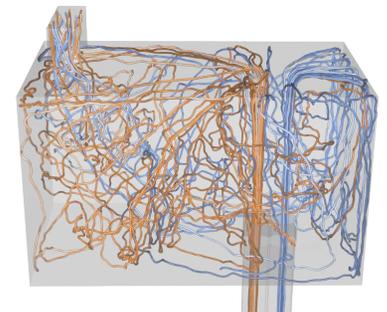
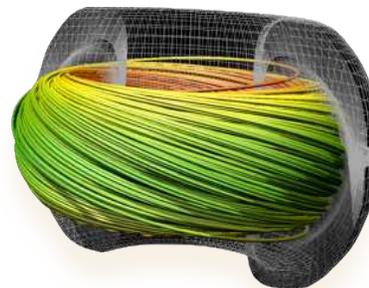
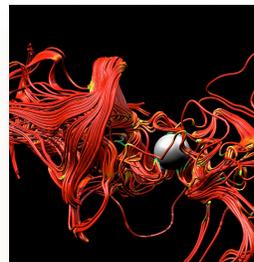
Once again, the word “hybrid” is being used in two contexts...

- The master-slave algorithm is a **hybrid algorithm**, sharing concepts from both parallelization-over-data and parallelization-over-seeds.
- **Hybrid parallelism** involves using a mix of shared and distributed memory techniques, e.g. MPI + pthreads or MPI+CUDA.
- One could think about implement a **hybrid particle advection algorithm** in a **hybrid parallel** setting.

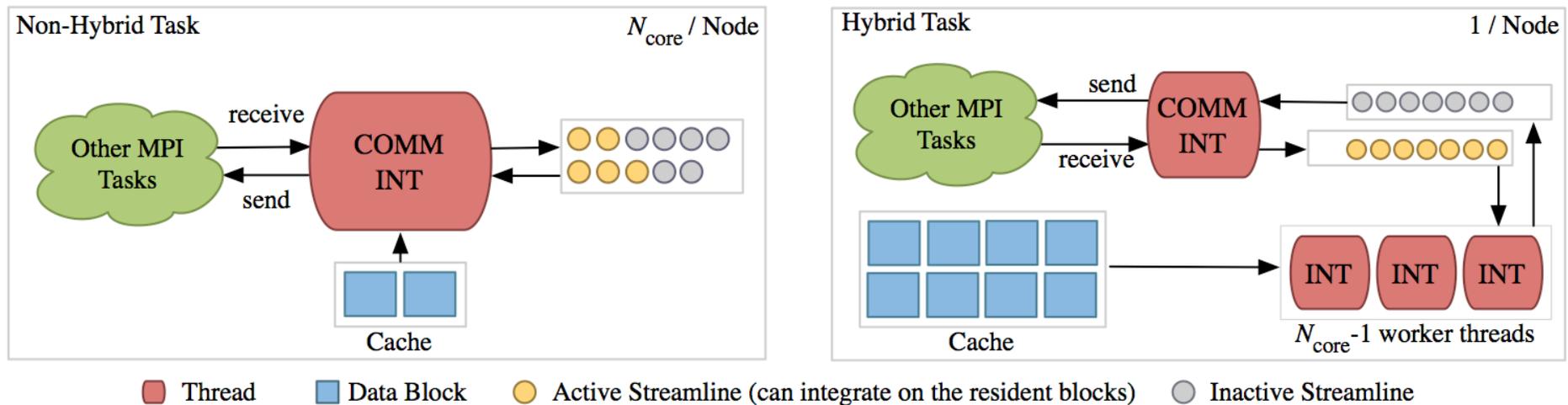
Streamline integration using MPI-hybrid parallelism on a large multi-core architecture

- Implement parallelize-over-data and parallelize-over-particles in a hybrid parallel setting (MPI + pthreads)
 - Did not study the master-slave algorithm
- Run series of tests on NERSC Franklin machine (Cray)
- Compare 128 MPI tasks (non-hybrid)
vs 32 MPI tasks / 4 cores per task (hybrid)
- 12 test cases: large vs small # of seeds
uniform vs non-uniform seed locations

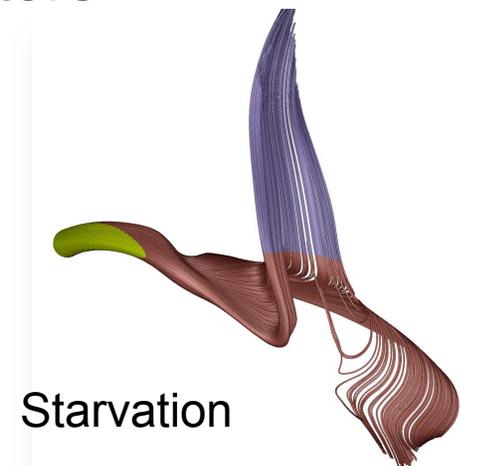
3 data sets



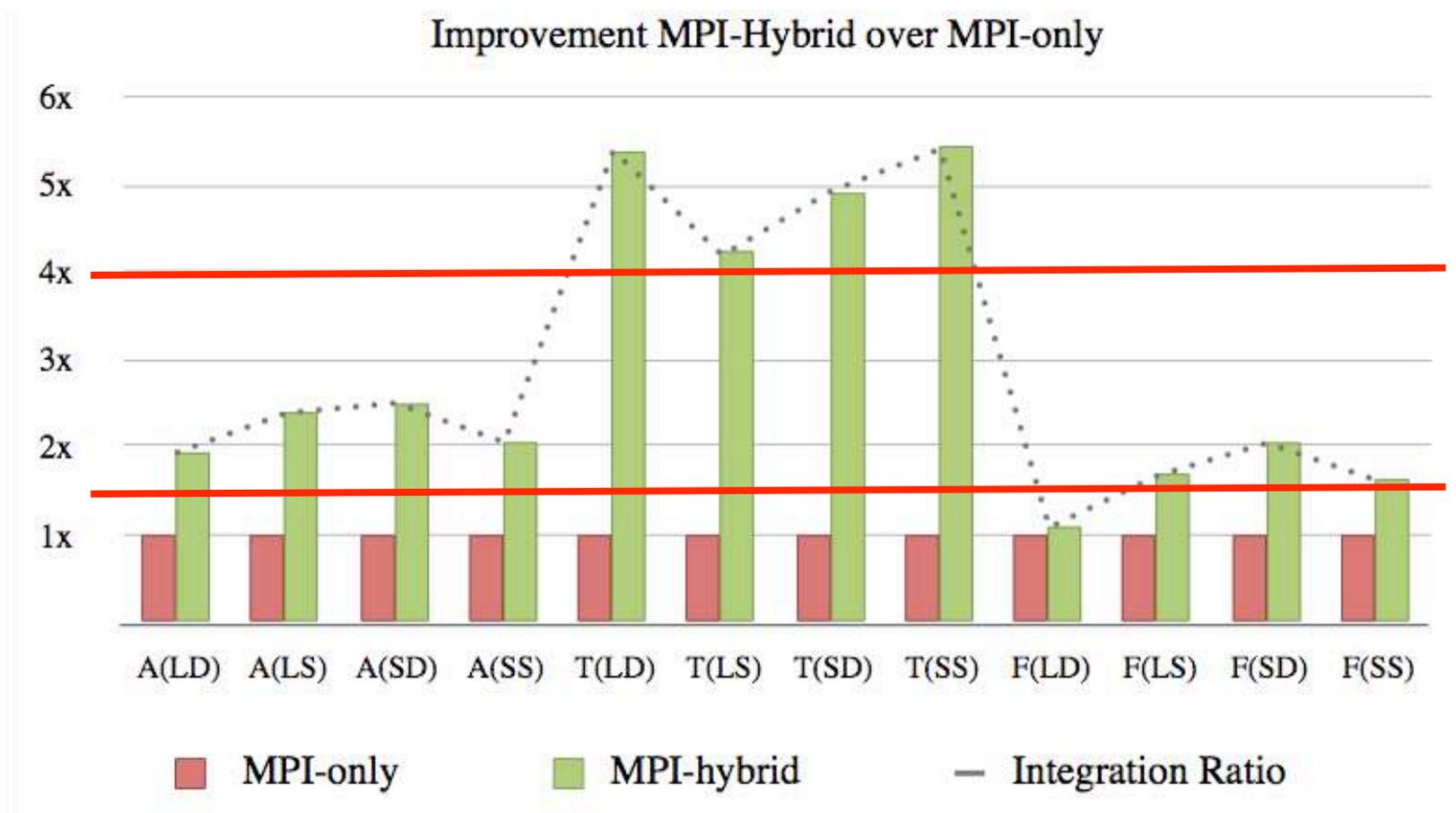
Hybrid parallelism for parallelize-over-data



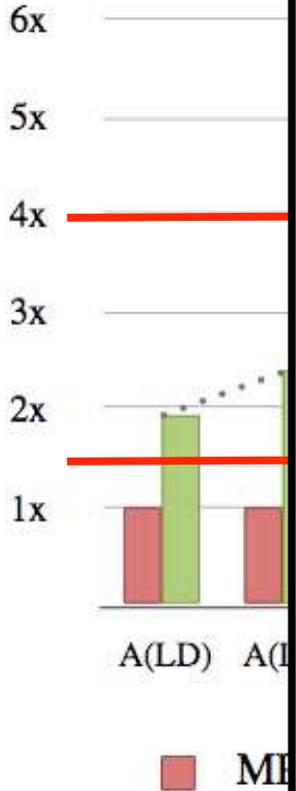
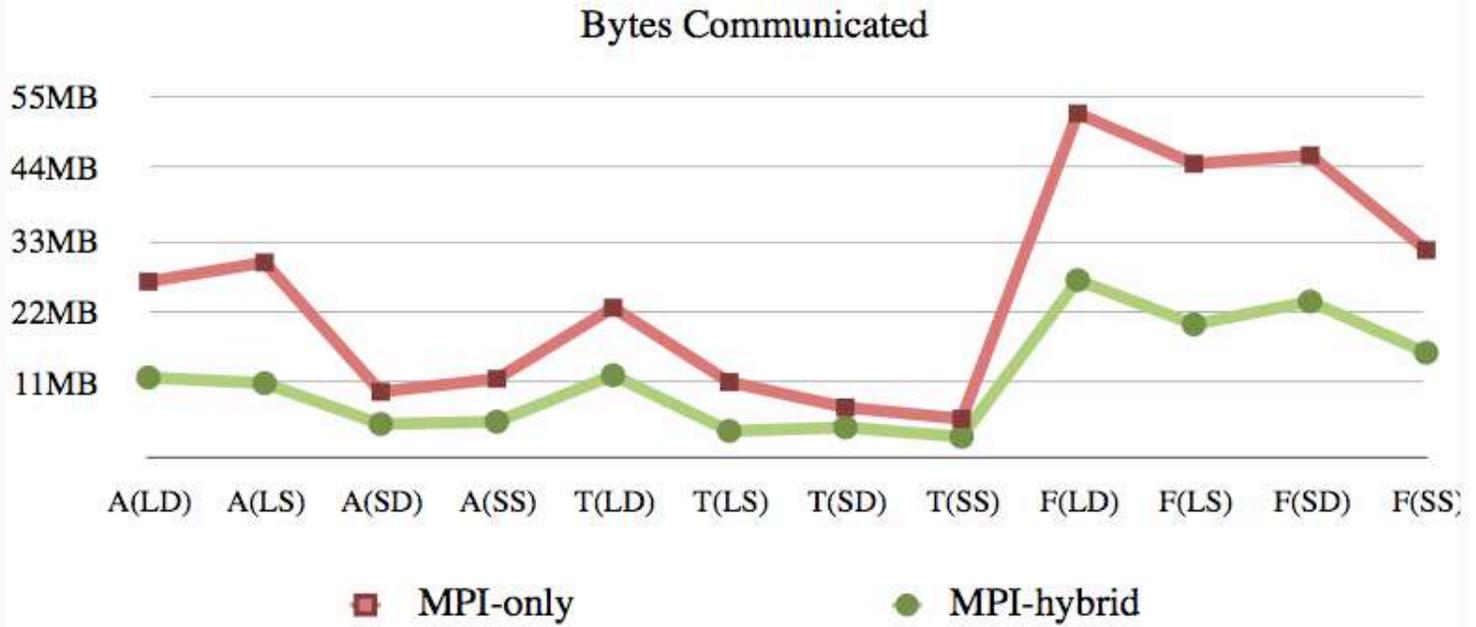
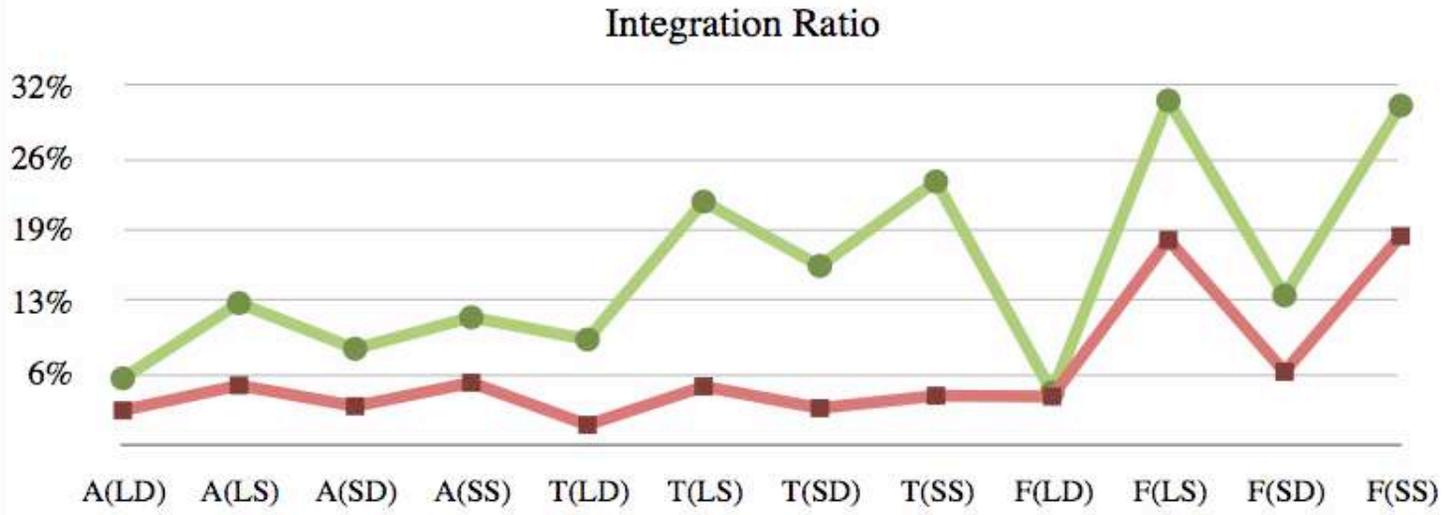
- Expected benefits:
 - Less communication and communicators
 - Should be able to avoid starvation by sharing data within a group.



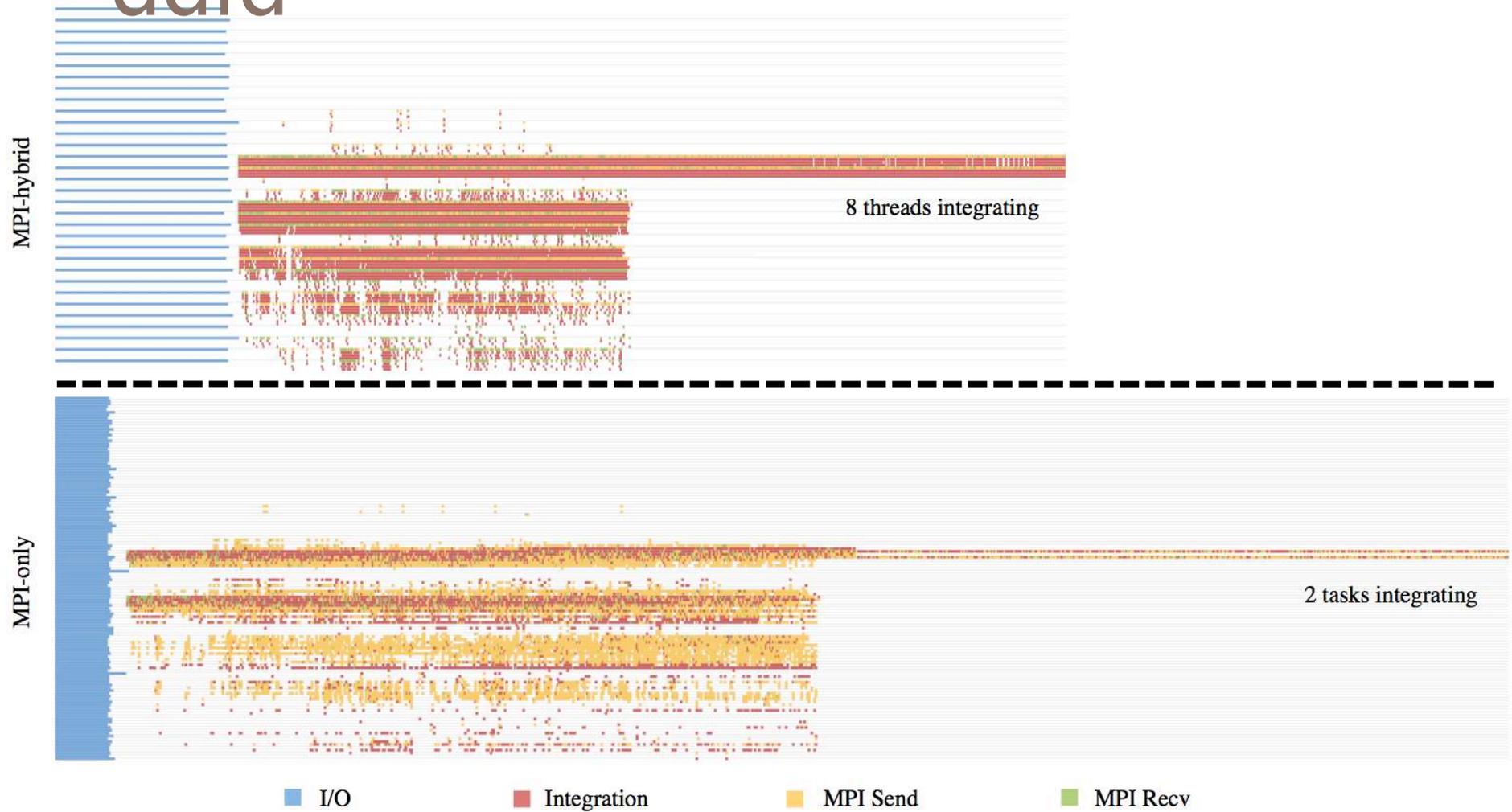
Measuring the benefits of hybrid parallelism for parallelize-over-data



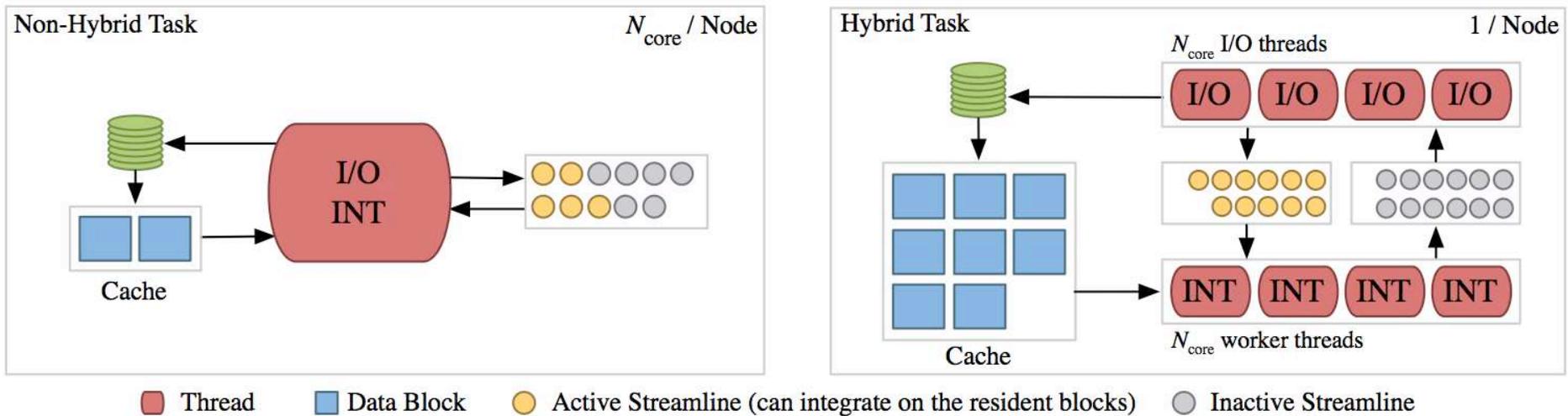
Measure parallel



Gantt chart for parallelize-over-data

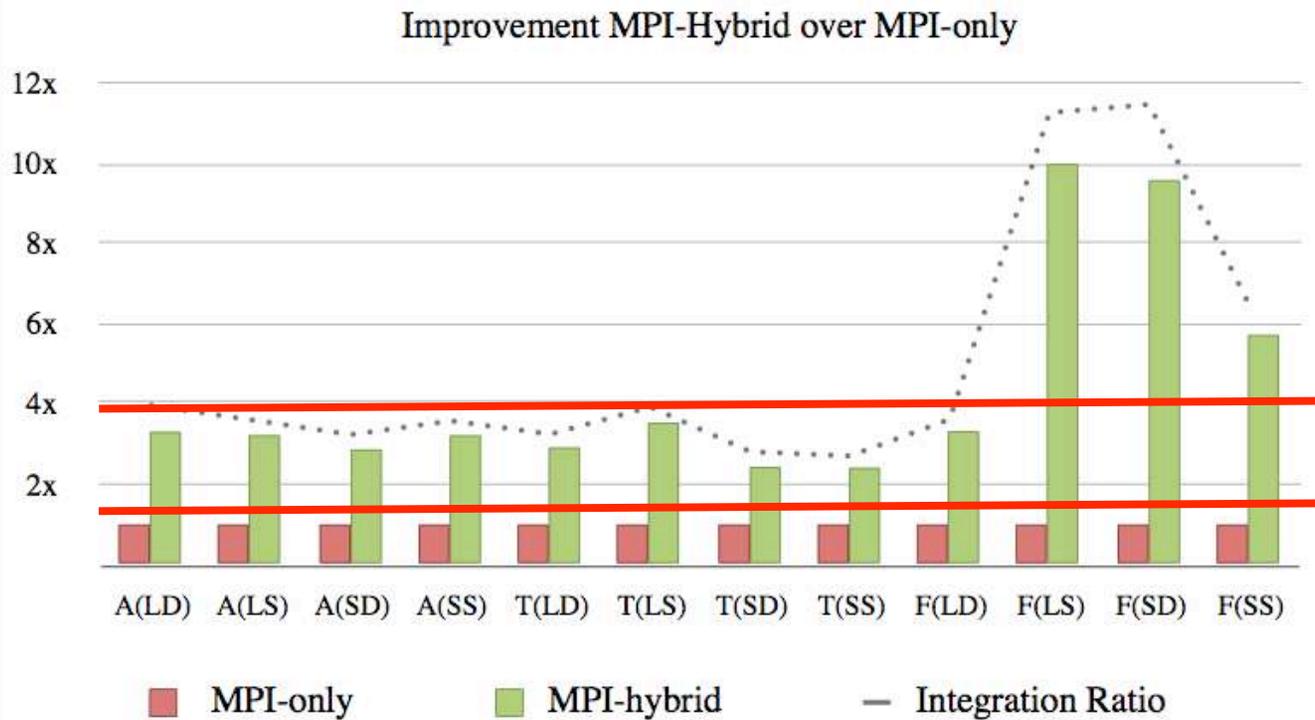


Hybrid parallelism for parallelize-over-particles

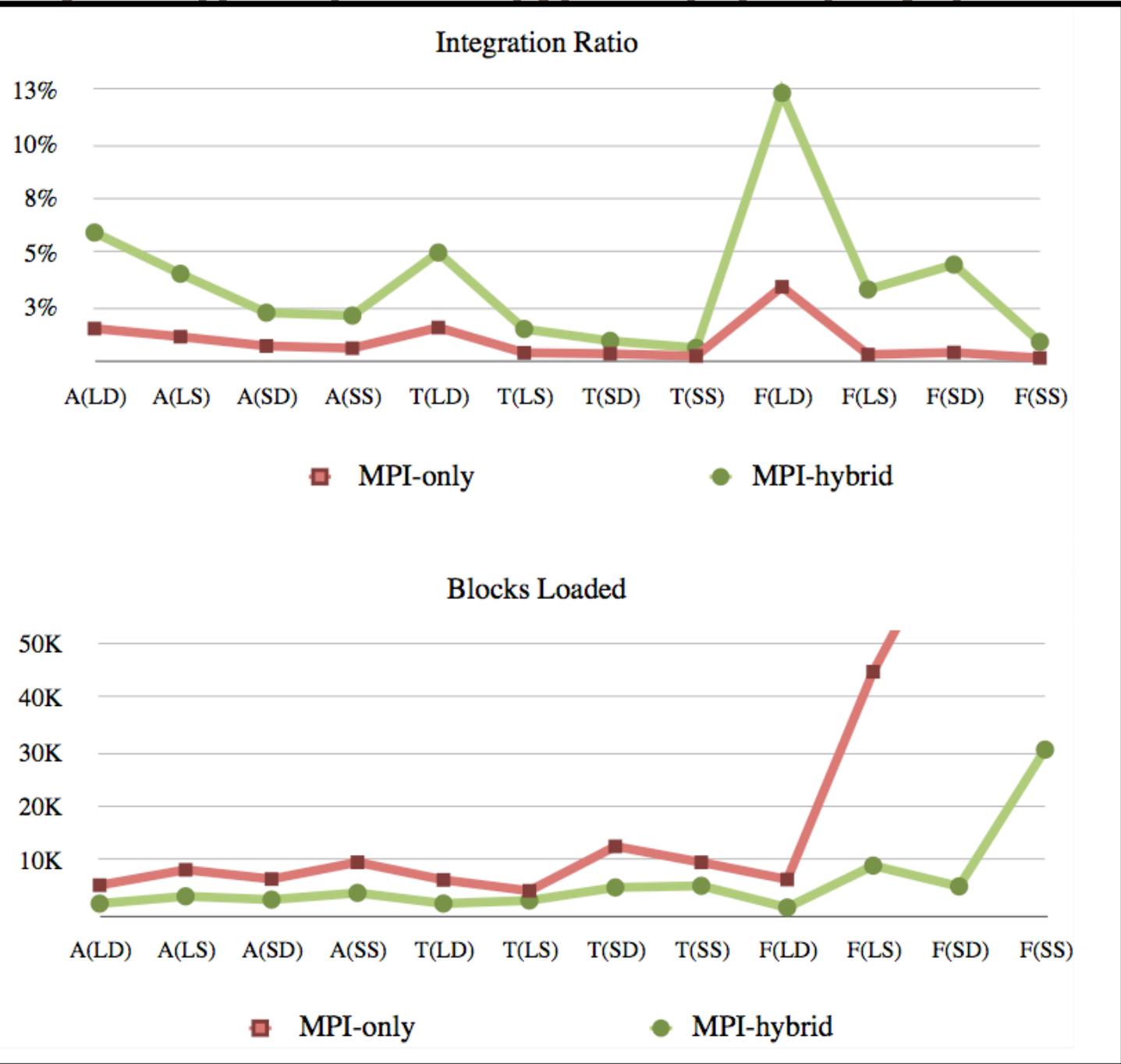
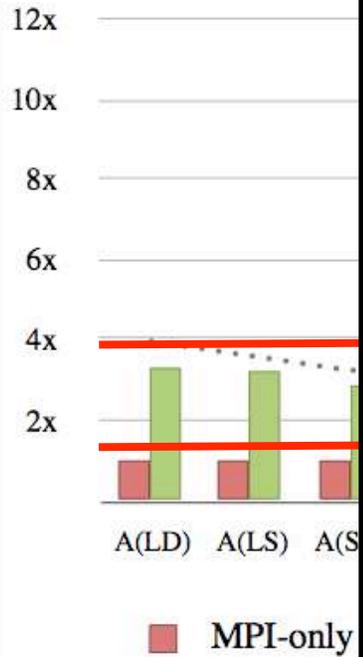


- Expected benefits:
 - Only need to read blocks once for node, instead of once for core.
 - Larger cache allows for reduced reads
 - “Long” paths automatically shared among cores on node

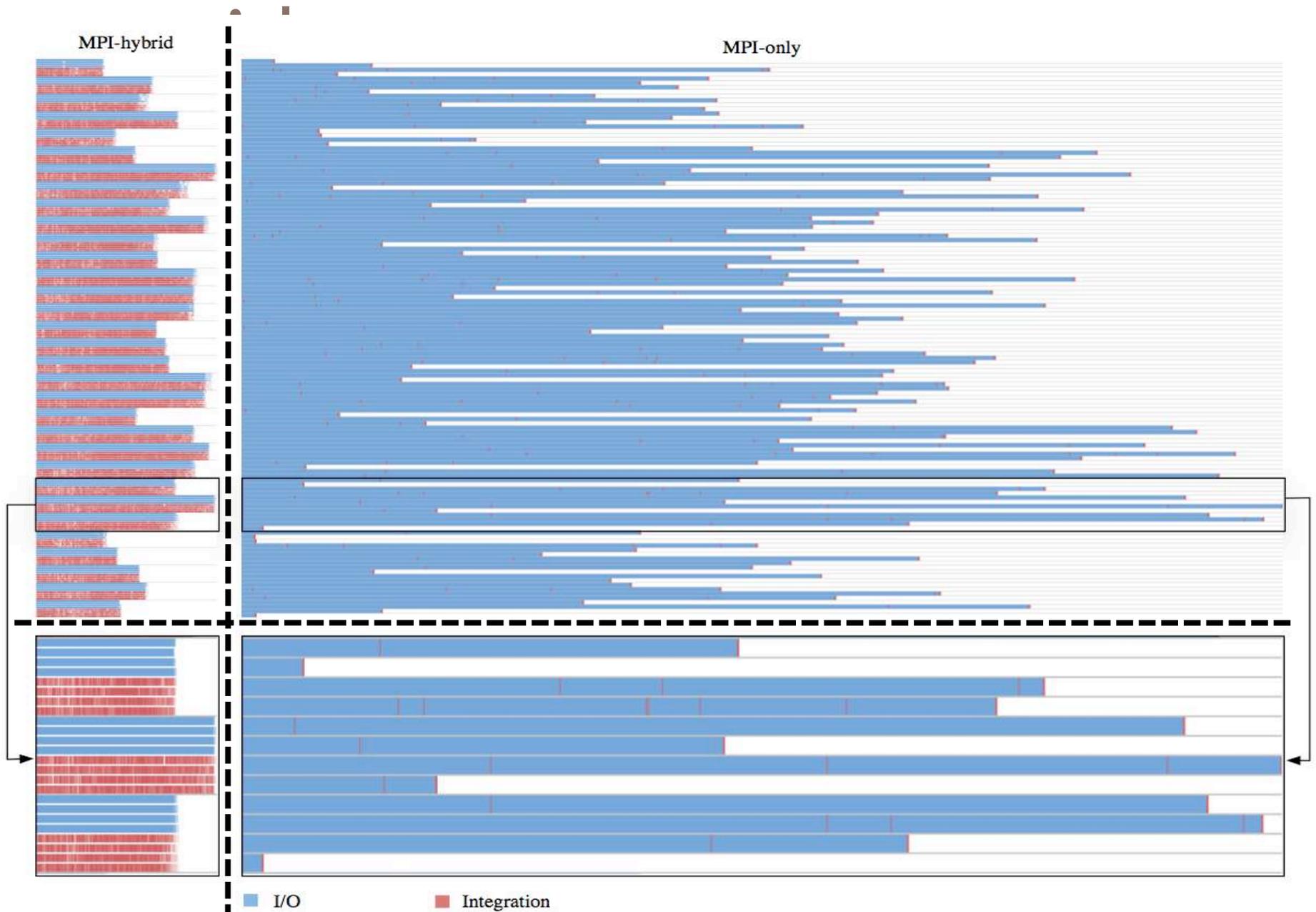
Measuring the benefits of hybrid parallelism for parallelize-over-particles



Meas para partic

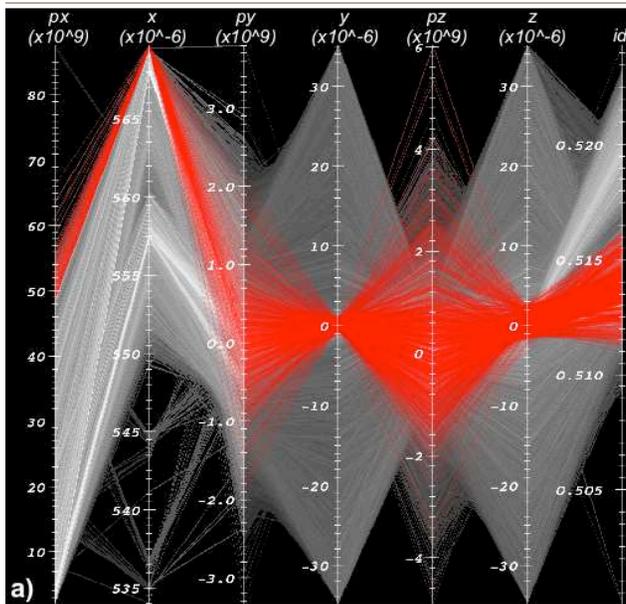


Gantt chart for parallelize-over-

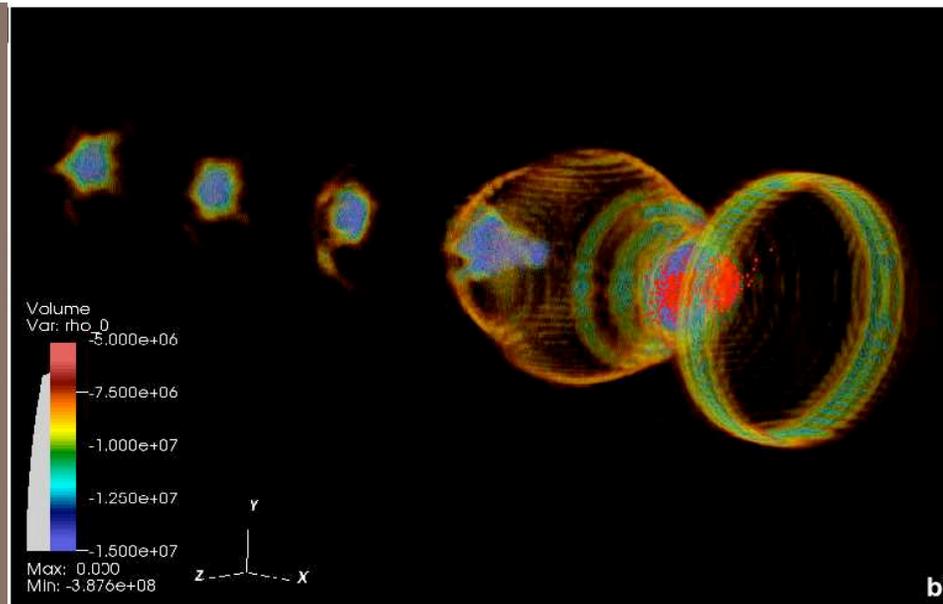


Summary of Hybrid Parallelism Study

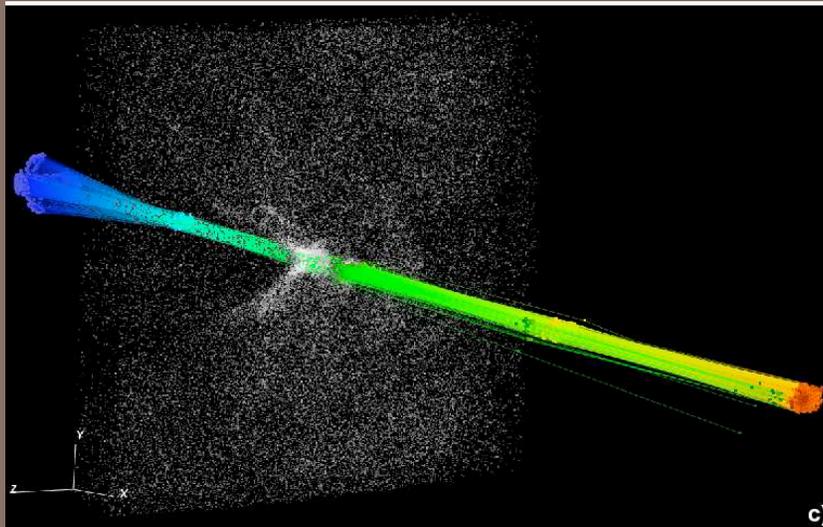
- Hybrid parallelism appears to be extremely beneficial to particle advection.
- Didn't implement the master-slave algorithm
 - ▣ ... but benefits shown at the spectrum extremes provide hope that hybrid algorithms will also benefit.



a) Selecting particles of interest



b) Selected particles (red) and volume rendering of the plasma density



c) Traces of the the selected particle-bunch

Smart Processing Techniques

June 16, 2011

Hank Childs, Lawrence Berkeley Lab & UC Davis

Outline



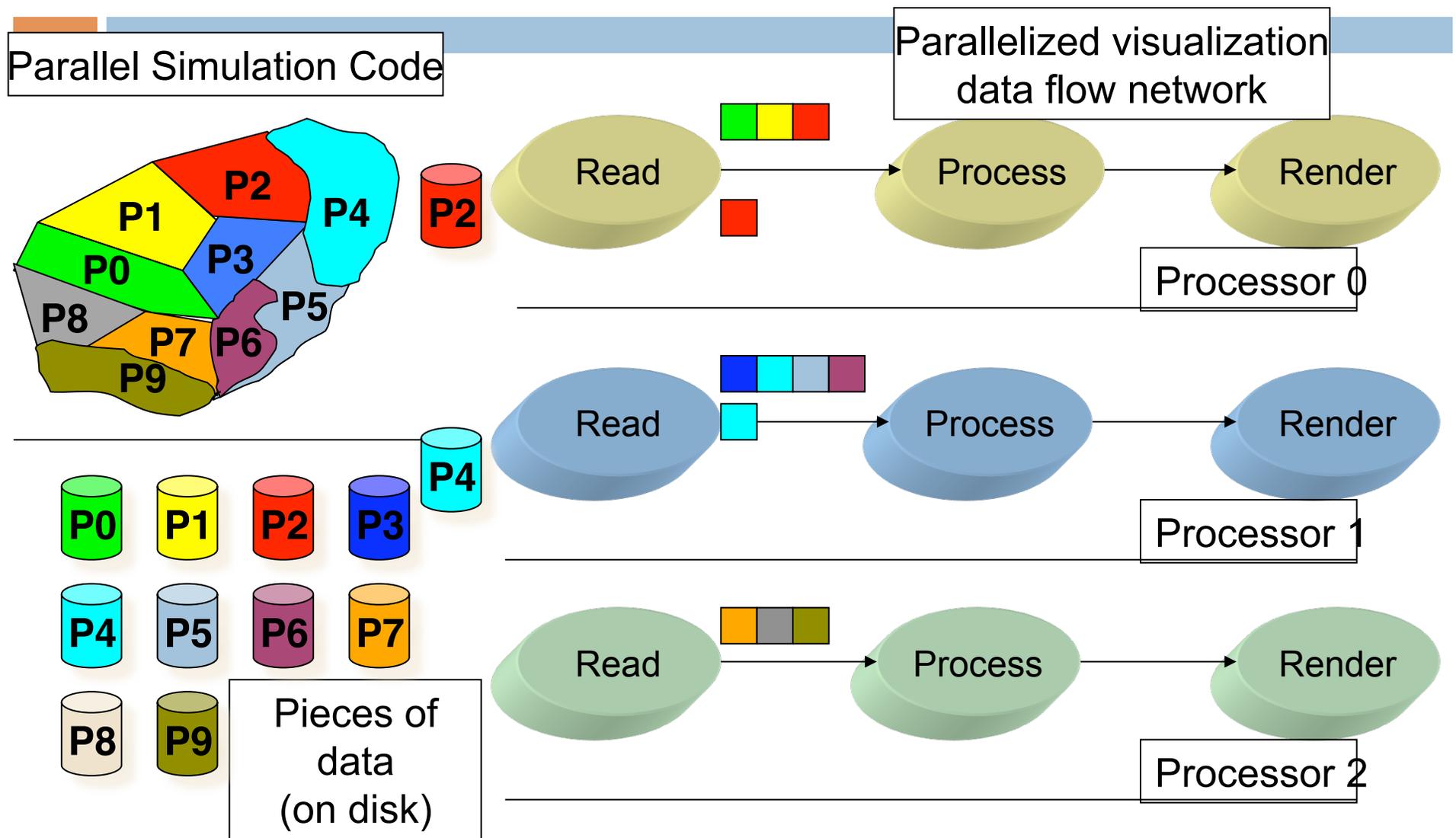
- Multi-resolution processing
 - Space filling curves (from Valerio Pascucci)
 - Wavelet compression (from John Clyne)
- In situ processing
 - System overview (from Brad Whitlock)
 - Example in action (from Jean Favre)
- Query-driven visualization
 - Overview (from Wes Bethel)
 - FastBit (from John Wu)
 - Example in action (from Oliver Ruebel)

Outline



- **Multi-resolution processing**
 - Space filling curves (from Valerio Pascucci)
 - Wavelet compression (from John Clyne)
- **In situ processing**
 - System overview (from Brad Whitlock)
 - Example in action (from Jean Favre)
- **Query-driven visualization**
 - Overview (from Wes Bethel)
 - FastBit (from John Wu)
 - Example in action (from Oliver Ruebel)

Multi-resolution techniques use coarse representations then refine.



Multi-resolution: pros and cons



- Pros

- Drastically reduce I/O & memory requirements

- Cons

- Is it meaningful to process simplified version of the data?
- How do we generate hierarchical representations?
What costs do they incur?

Difficult conversations in the future.



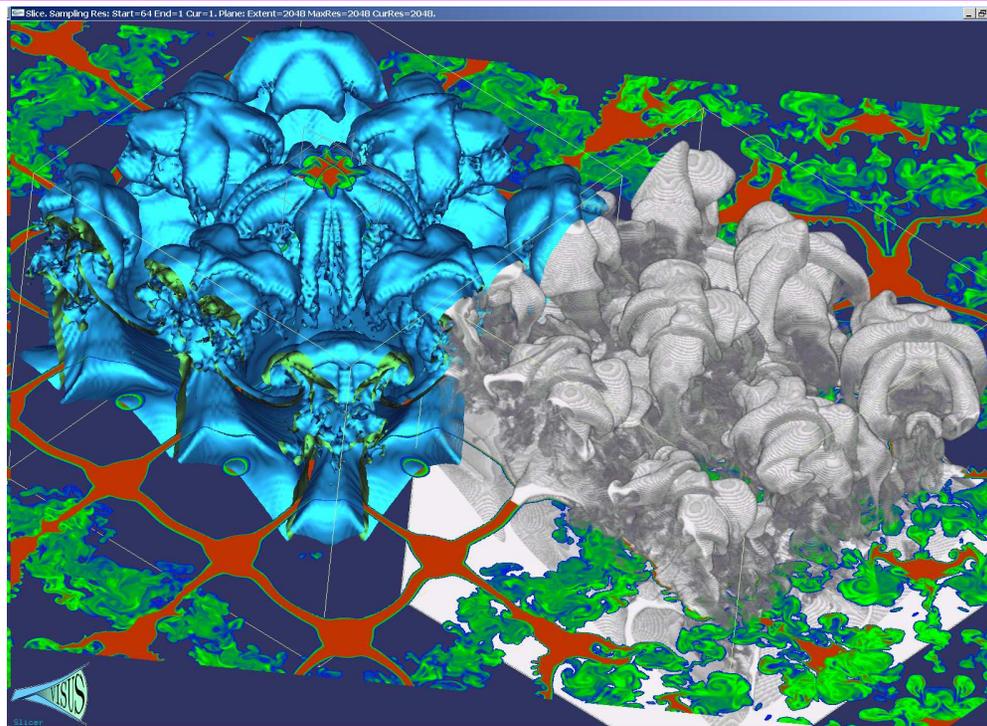
- Multi-resolution questions we should be asking our customers:
 - Do you understand what a multi-resolution hierarchy should look like for your data?
 - Who do you trust to generate it?
 - Are you comfortable with your I/O routines generating these hierarchies while they write?
 - How much overhead are you willing to tolerate on your dumps? 33+%?
 - Willing to accept that your visualizations are not the “real” data?

Outline

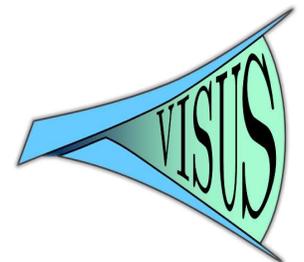


- Multi-resolution processing
 - Space filling curves (from Valerio Pascucci)
 - Wavelet compression (from John Clyne)
- In situ processing
 - System overview (from Brad Whitlock)
 - Example in action (from Jean Favre)
- Query-driven visualization
 - Overview (from Wes Bethel)
 - FastBit (from John Wu)
 - Example in action (from Oliver Ruebel)

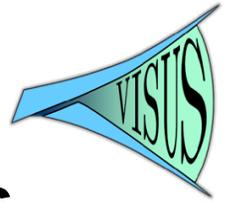
Cache Oblivious Progressive Methods for Regular Data



Valerio Pascucci
University of Utah

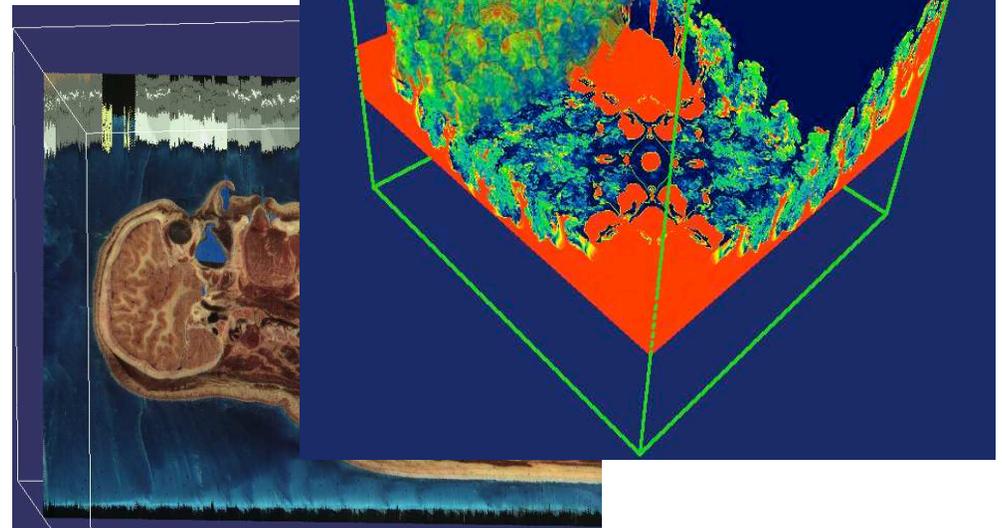
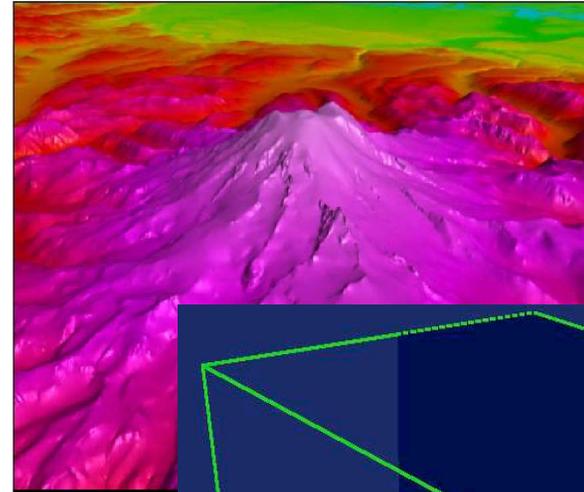


We must achieve real-time interaction with large datasets on a wide variety of platforms

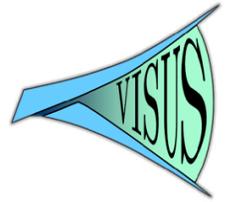


The problem

- Large datasets of different type: terrains, satellite images, 8GB/ timestep ($2k^3$ grids +time).
- Interactive rendering for real-time data exploration.
- Target platforms: desktop, parallel server, cluster.

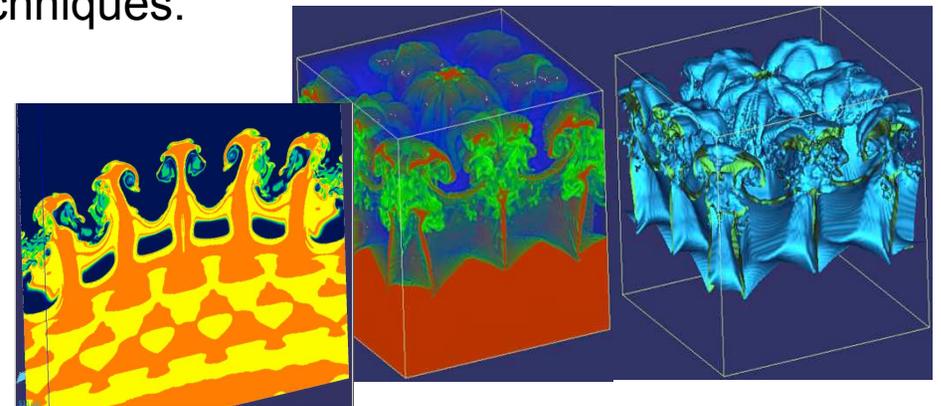
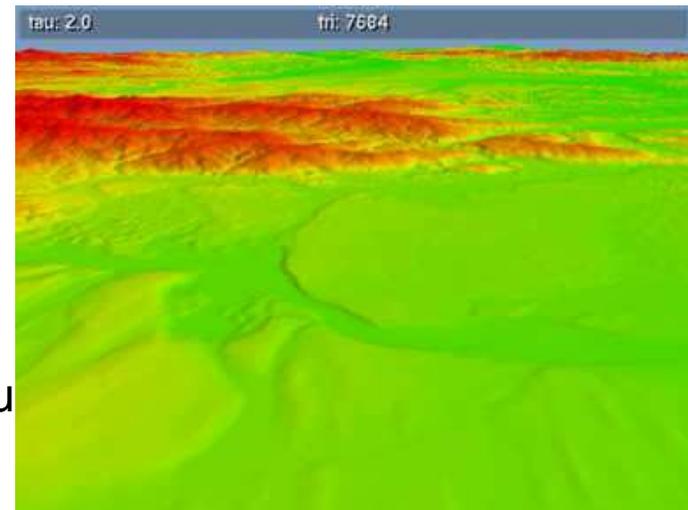


We apply three fundamental techniques to the visualization of large simulation data

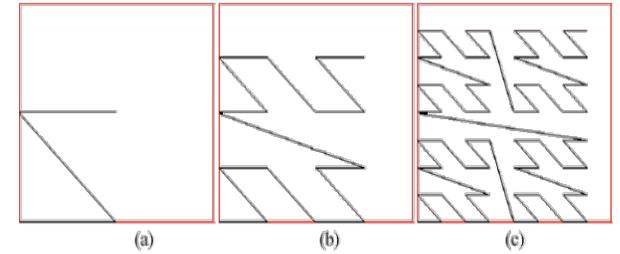
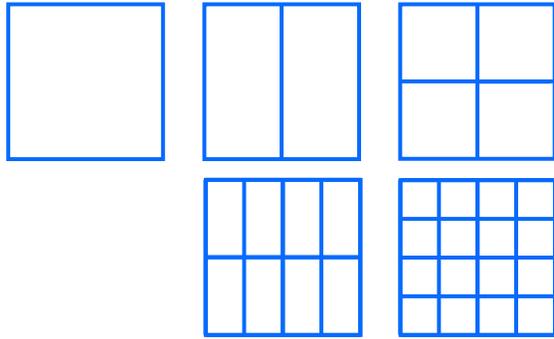


The general approach

- Multi-resolution geometric representation:
 - adaptive view-dependent refinement;
 - minimal geometric output for selected error tolerance.
- Cache oblivious external memory data layout
 - exploit spatial and resolution coherency;
 - no need for complicated paging techniques.
- Progressive processing:
 - continuously improved rendering;
 - scalability with the resources without budgeting.

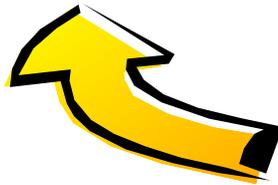
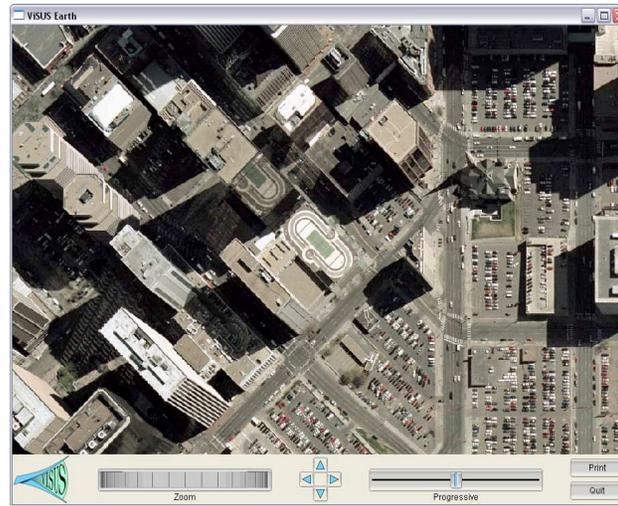


The General Infrastructure is Structured into Three Main Components



Processing Network
(Data Access Path)

Data Layout
(Cache Oblivious)

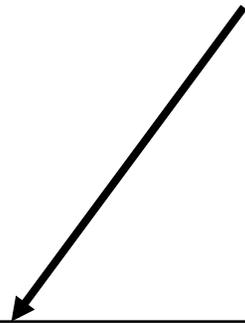


Algorithm Design
(Progressive Processing)

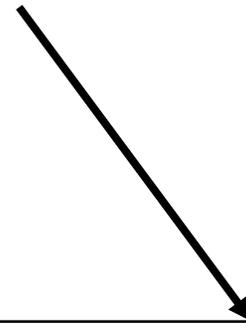


General Data Layout

Data coherent Progressive refinement of a hierarchical geometric data-structure



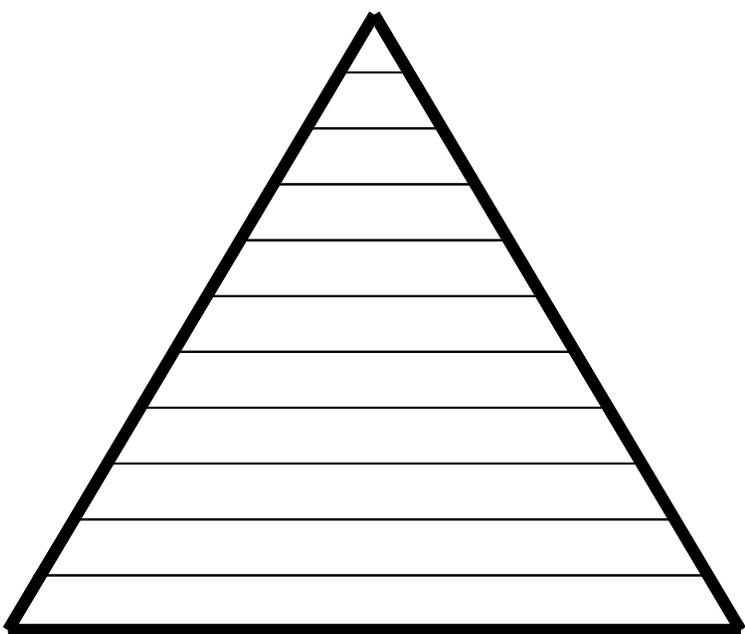
Grouping the data by level of resolution



Grouping the data by geometric proximity

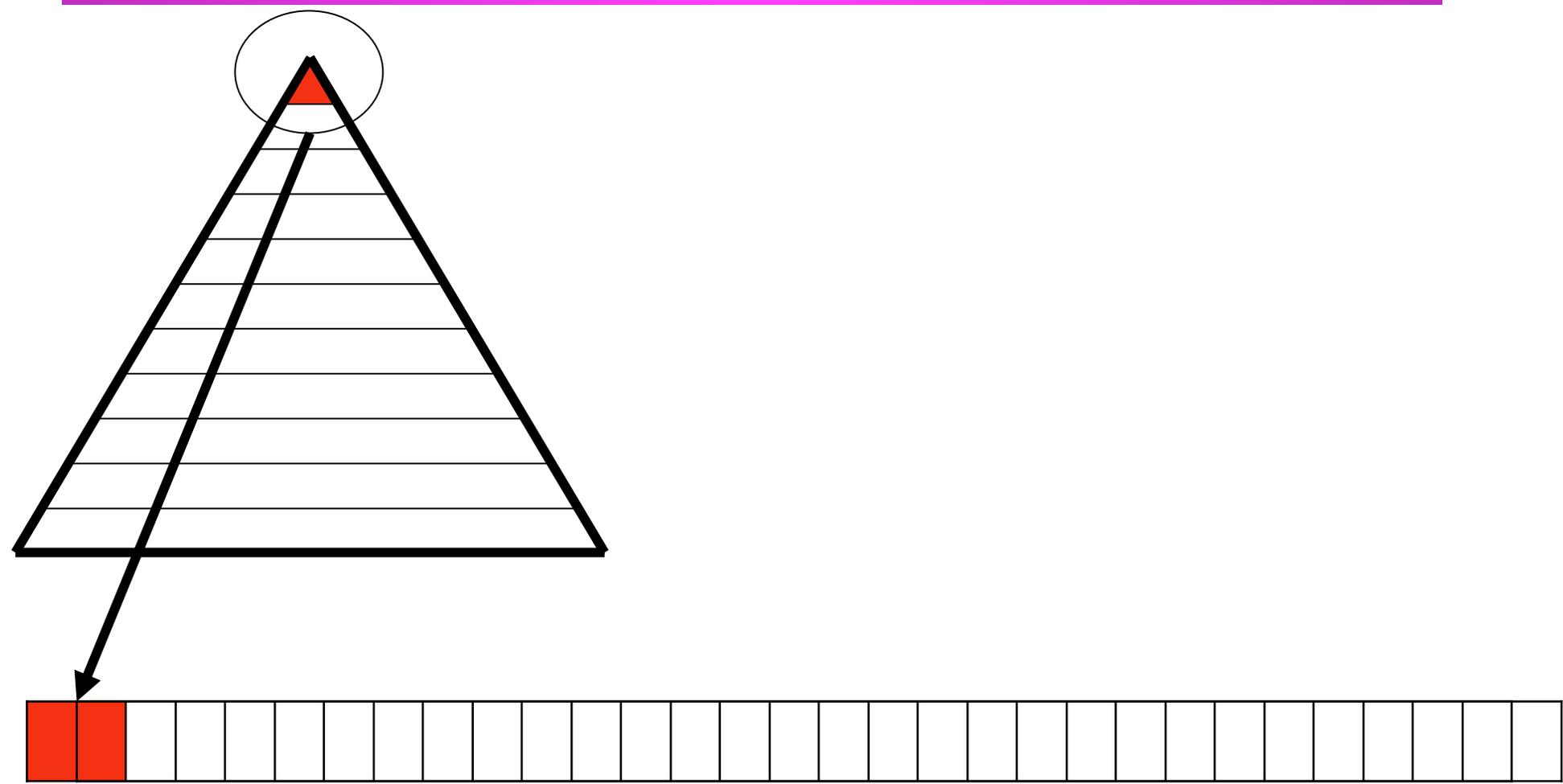


General Data Layout



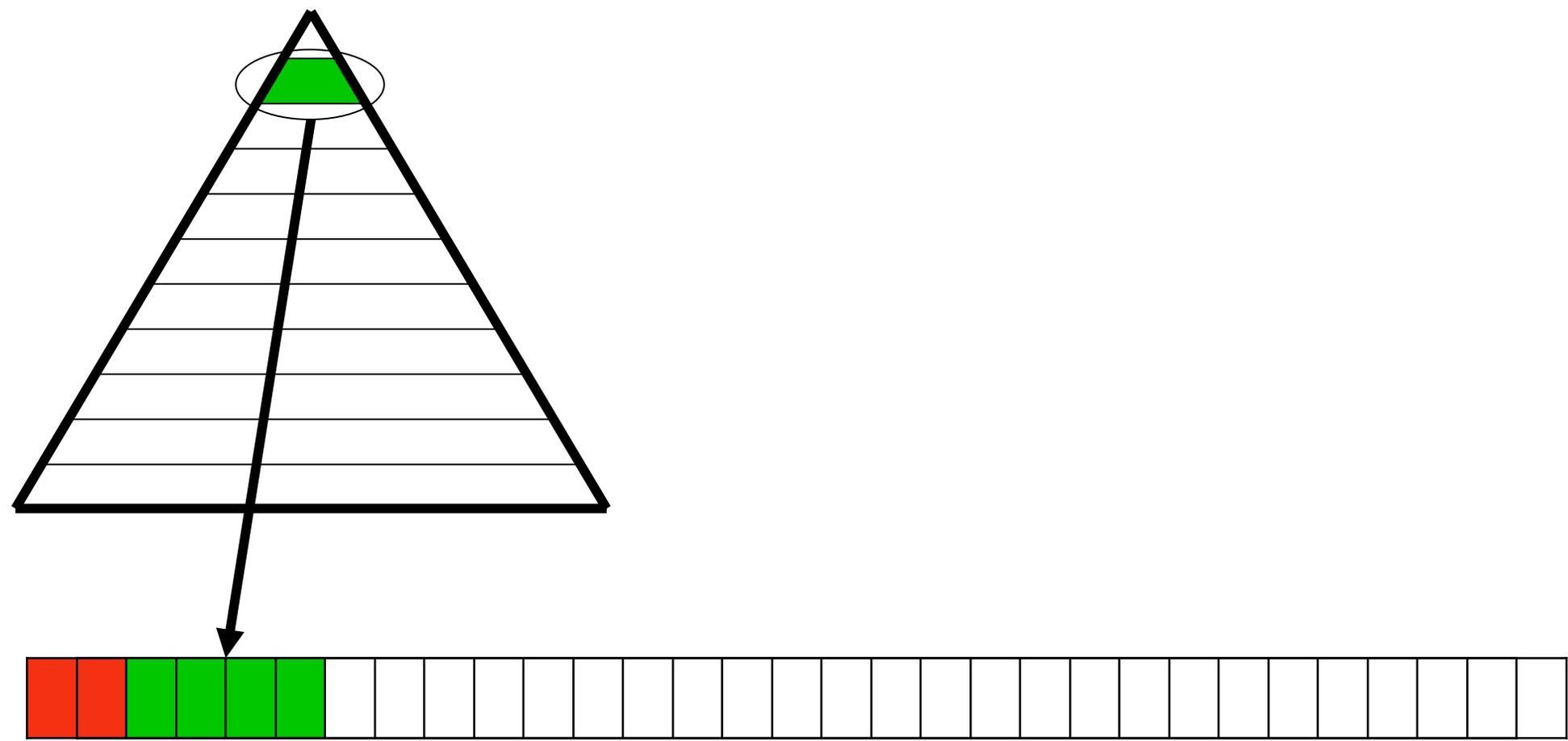


General Data Layout



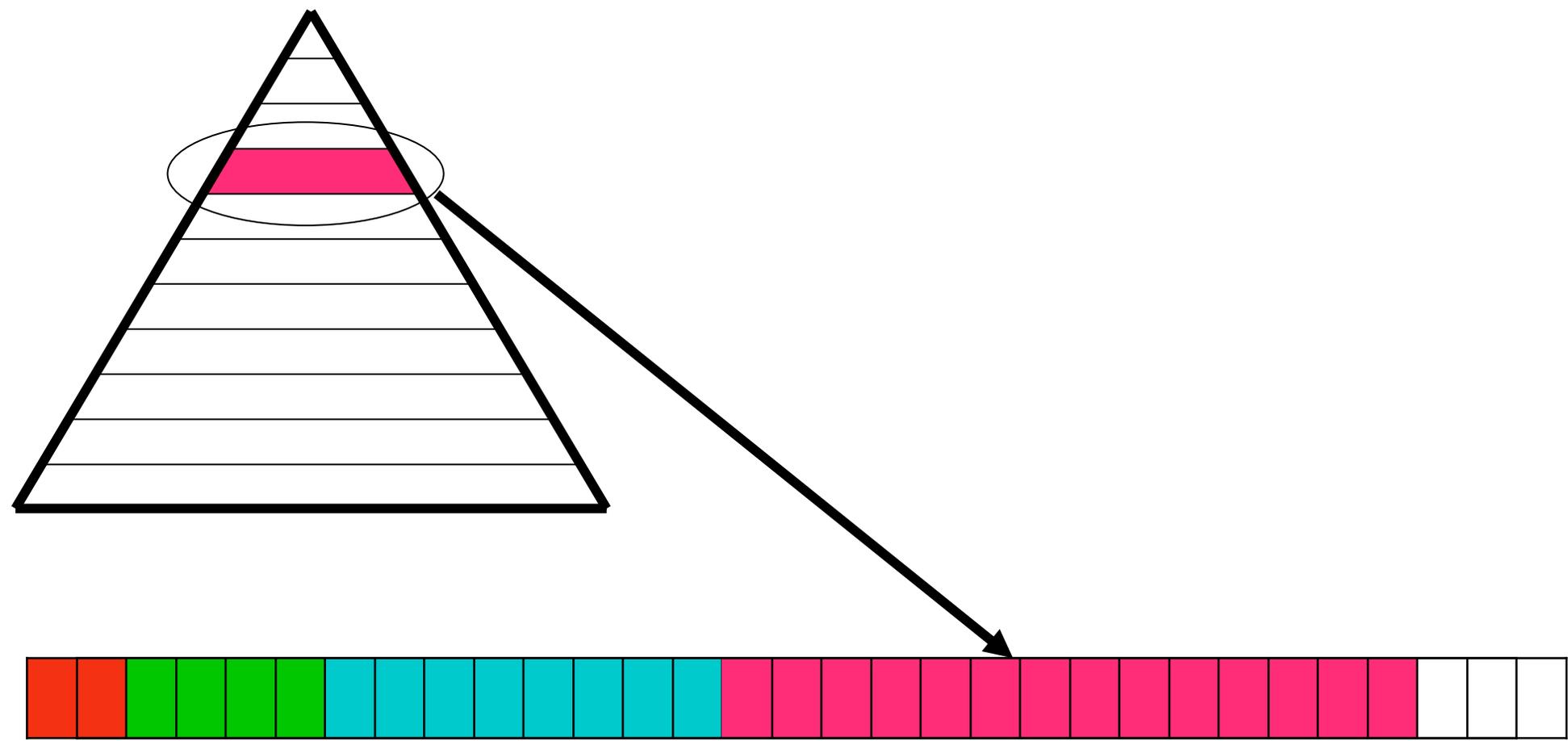


General Data Layout



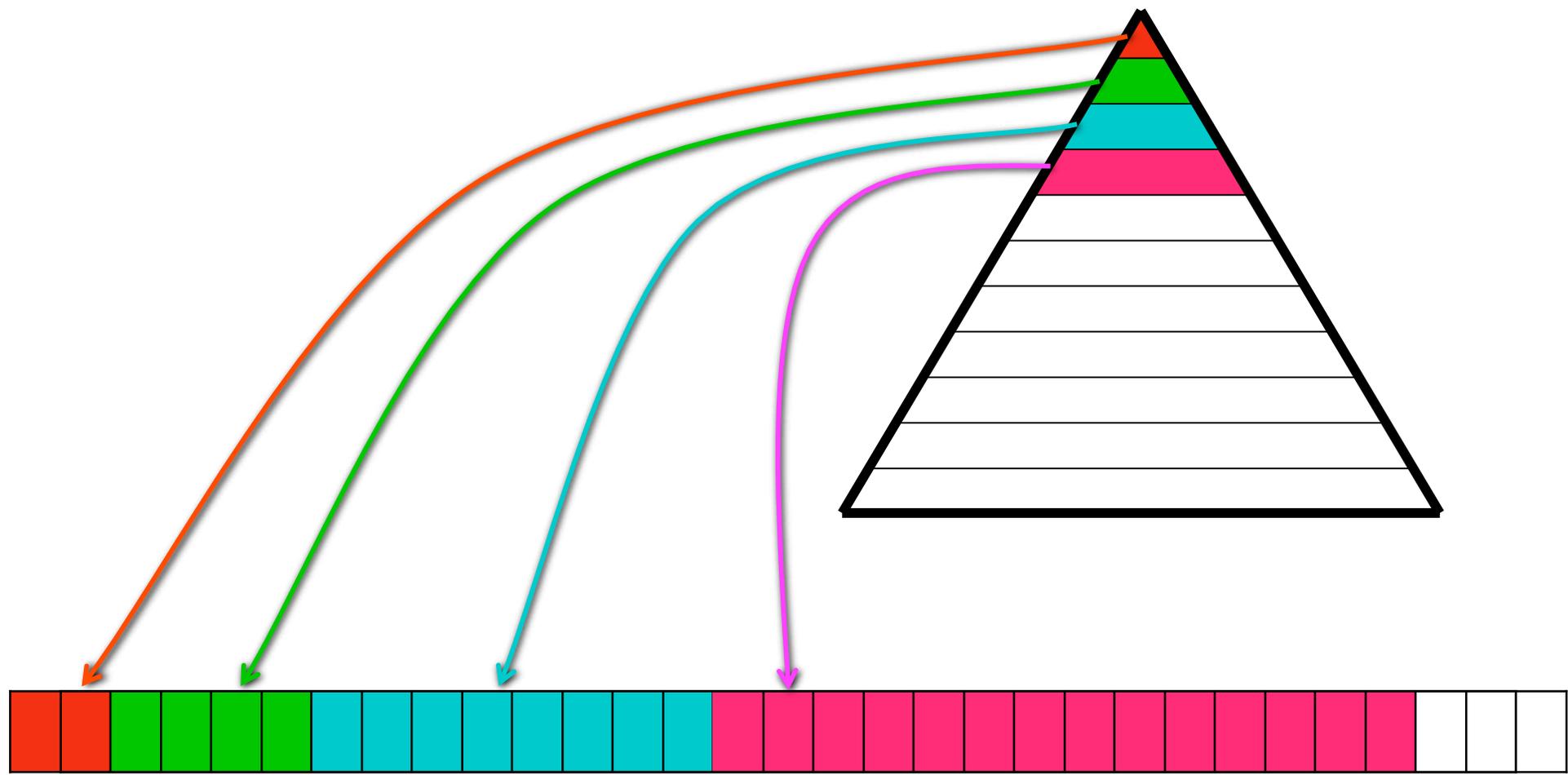


General Data Layout

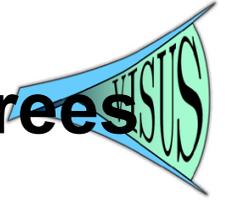




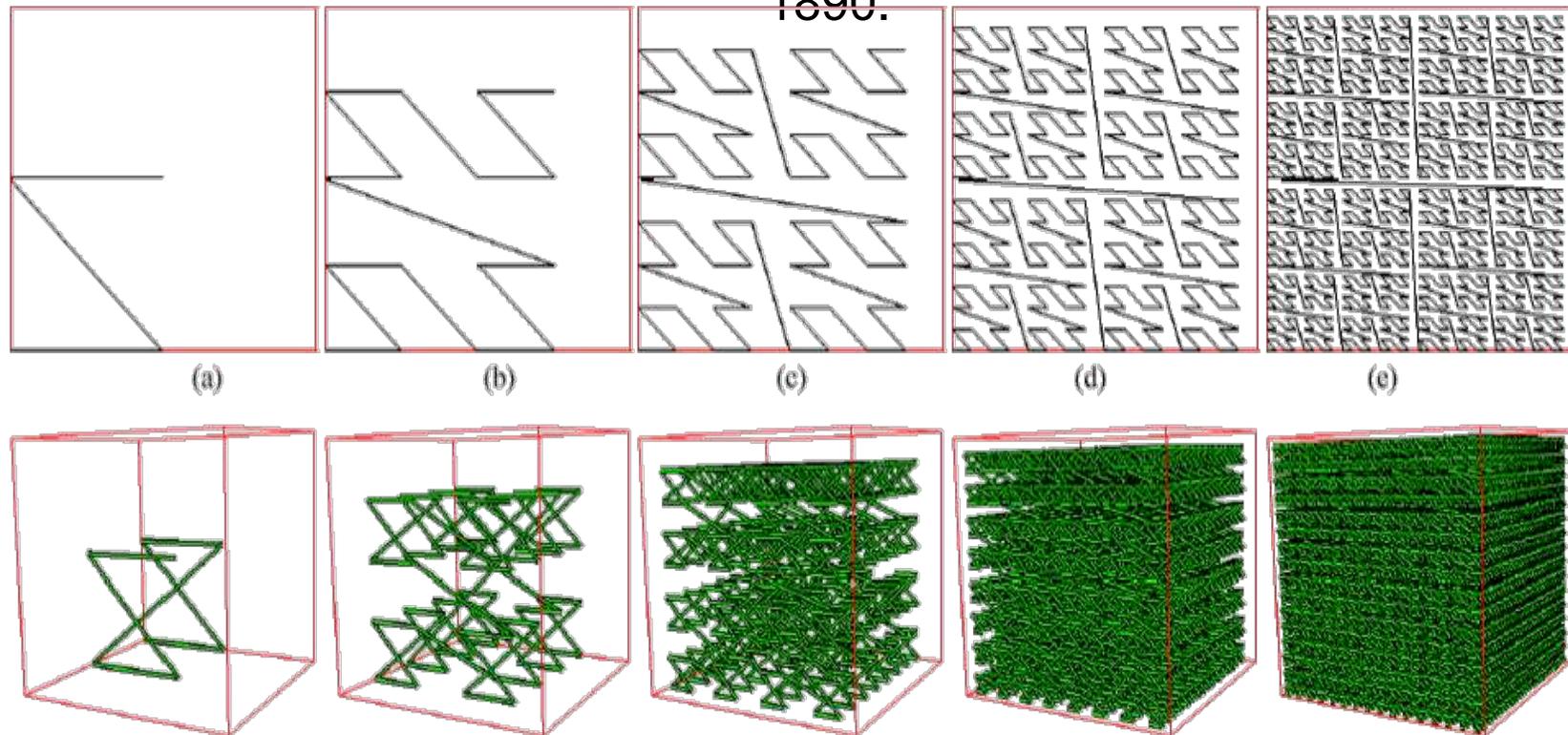
General Data Layout



We exploit the correlation of bin/quad/oct-trees with the Lebesgue space-filling curves



The Lebesgue curve is also known as Z-order, Morton, Curve.
Special case of the general definition introduced by Guiseppe Peano in 1890.

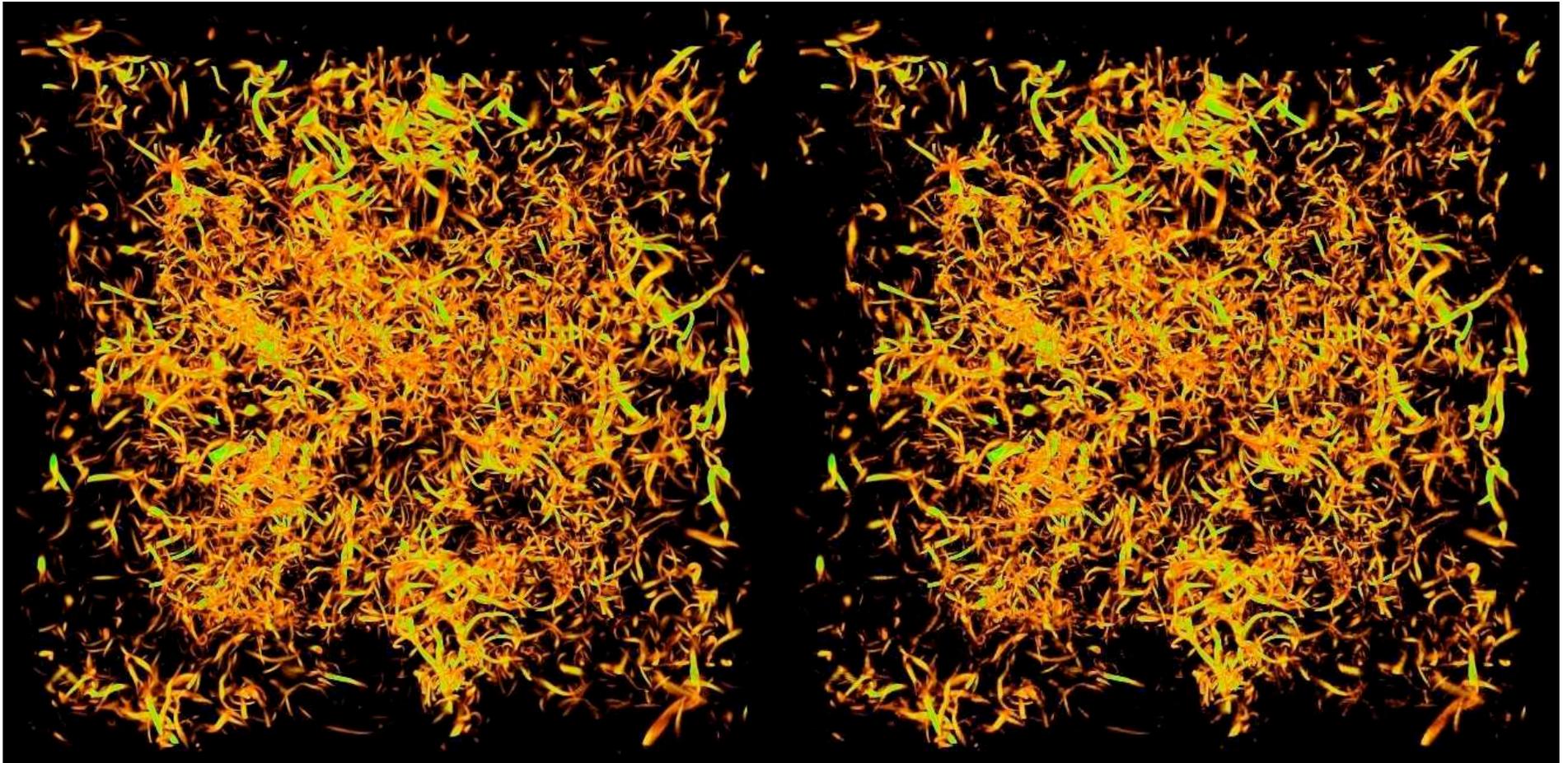


Outline



- Multi-resolution processing
 - Space filling curves (from Valerio Pascucci)
 - Wavelet compression (from John Clyne)
- In situ processing
 - System overview (from Brad Whitlock)
 - Example in action (from Jean Favre)
- Query-driven visualization
 - Overview (from Wes Bethel)
 - FastBit (from John Wu)
 - Example in action (from Oliver Ruebel)

Wavelet Compression
John Clyne
National Center for Atmospheric Research (NCAR)



No compression

Coefficient prioritization (VDC2)

Compression of scientific data with wavelets

- Many popular multi-media formats employ wavelet based compression techniques
 - E.g. JPEG 2000 (still images), Ogg (A/V), Dirac (video)
- Extending wavelet techniques to multi-dimensional, gridded scientific data is relatively straight forward
- Advantages of Wavelet based compression strategies for scientific data include:
 - Reduced storage capacity
 - Reduced IO bandwidth
 - Reduced computation and memory needs
 - Wavelet based compression readily supports hierarchical data representation
 - Coarsened approximations in hierarchy have fewer grid points leading to less processing and less memory
 - Progressive refinement
 - Data may be delivered with progressively increasing detail, providing coarse approximations that may be selectively refined all the way up to the original data



Wavelet transforms in a nutshell

- Similar to Fourier transforms a Wavelet transform expresses a signal $f(t)$ as linear expansion :

$$f(t) = \sum_l a_l \psi_l(t)$$

where a_l are real-valued coefficients, and ψ_l are basis functions

- For many wavelet functions the transform coefficients, a_l , are simply given by the inner product:

$$a_l = \langle f(t), \psi_l(t) \rangle = \int f(t) \psi_l(t) dt$$

- Wavelet transforms have several key differences from Fourier transforms
 - Basis function, ψ , is a wavelet, not a complex exponential
 - Wavelets have compact support (zero value outside of a narrow interval)
 - transforms can localize signal details (frequencies) in time (space). This tells us not just *what* frequencies are present but *when* they occur
 - Forward and inverse transforms are computationally efficient: $O(N)$ compared to $O(N \log N)$ for Fourier



Discrete Wavelet Transforms

The wavelet basis function is constructed from a *scaling* function, ϕ :

$$\psi(t) = \sum_k h_\psi(k) \sqrt{2} \phi(2t - k), \quad k \in \mathbb{Z} \quad \text{wavelet function}$$

which is recursively constructed from scaled, dyadic translates of itself:

$$\phi(t) = \sum_k h_\phi(k) \sqrt{2} \phi(2t - k), \quad k \in \mathbb{Z} \quad \text{scaling function}$$

which leads to a more general representation of a wavelet expansion of f :

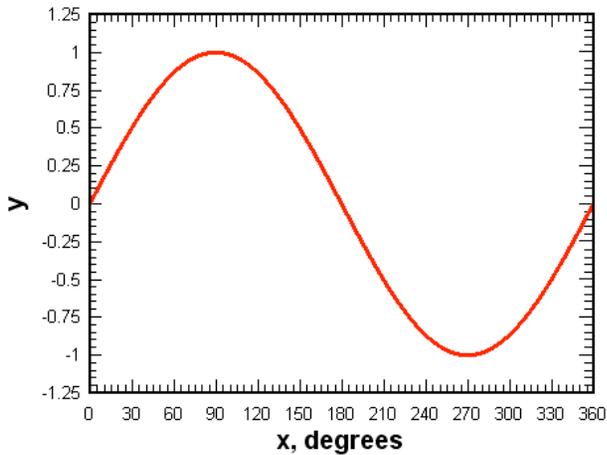
$$f(t) = \sum_k c(k) \phi_k(t) + \sum_k \sum_{j=0}^{\log_2 N} d_j(k) \psi_{j,k}(t)$$

Scaling term: low resolution approximation of $f(t)$

Detail term: high frequency components of signal missing from scaling term



$$y = \sin(x)$$



Fourier transform basis function: sine, cosine

Compact support of wavelets enables efficient transforms and ability of wavelet transforms to isolate signal components in time (space)

A **very** small sampling of wavelet basis functions

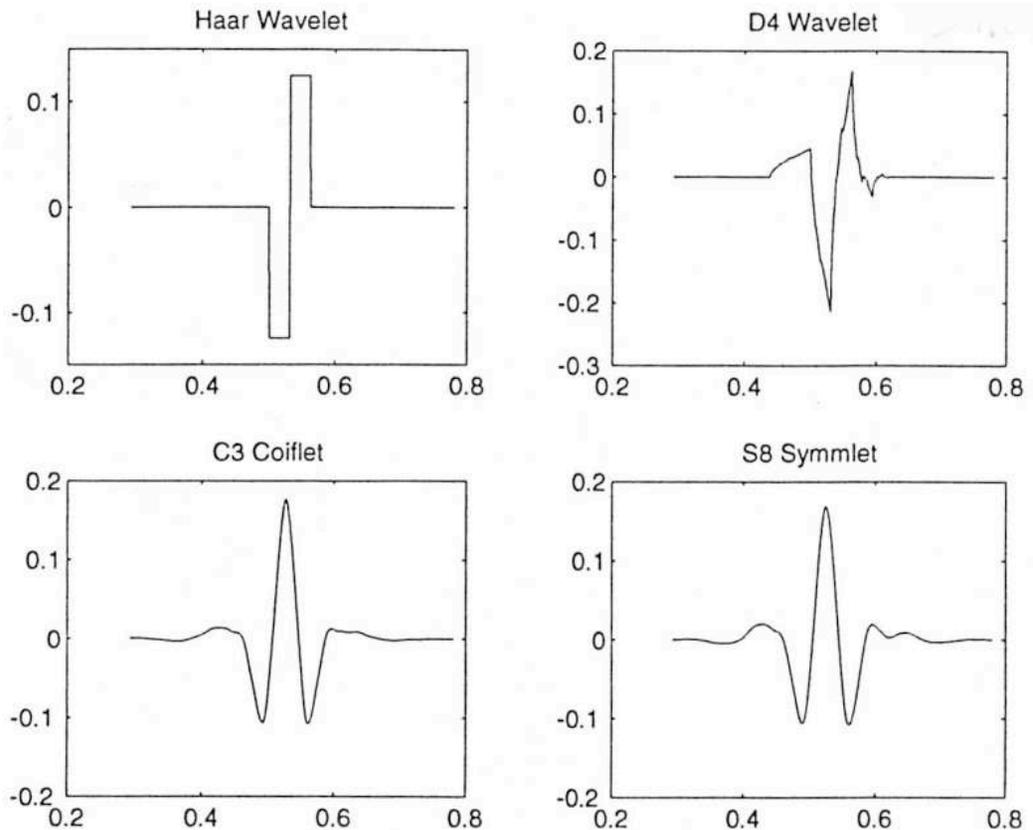


Image credit: K.H. Parker

Wavelet compression and progressive access (1)

Frequency truncation method

- Truncate “ j ” parameter in expansion:

$$f(t) = \sum_k c(k)\phi_k(t) + \sum_k \sum_{j=0}^{\log_2 N} d_j(k)\psi_{j,k}(t)$$

- Provides coarsened approximation of signal at power-of-two increments
- Good
 - Simple to implement
 - Extremely computationally efficient (much faster than reading data)
 - Retained coefficients are implicitly addressed => no additional storage required
 - Progressive refinement readily provided
- Not so good:
 - Limited to power-of-two grid reductions
 - Quality of approximations



Wavelet compression and progressive access (2)

Coefficient prioritization method

Goal: prioritize coefficients used in linear expansion based on contribution to signal

$$f(t) = \sum_{n=0}^{N-1} a_n u(t), \quad \text{original } f(t) \qquad \hat{f}(t) = \sum_{m=0}^{M-1} a_{\pi(m)} u(t), \quad (M < N), \quad \text{compressed } f(t)$$

L^2 error given by: $L^2 = \left\| f(t) - \hat{f}(t) \right\|_2^2$

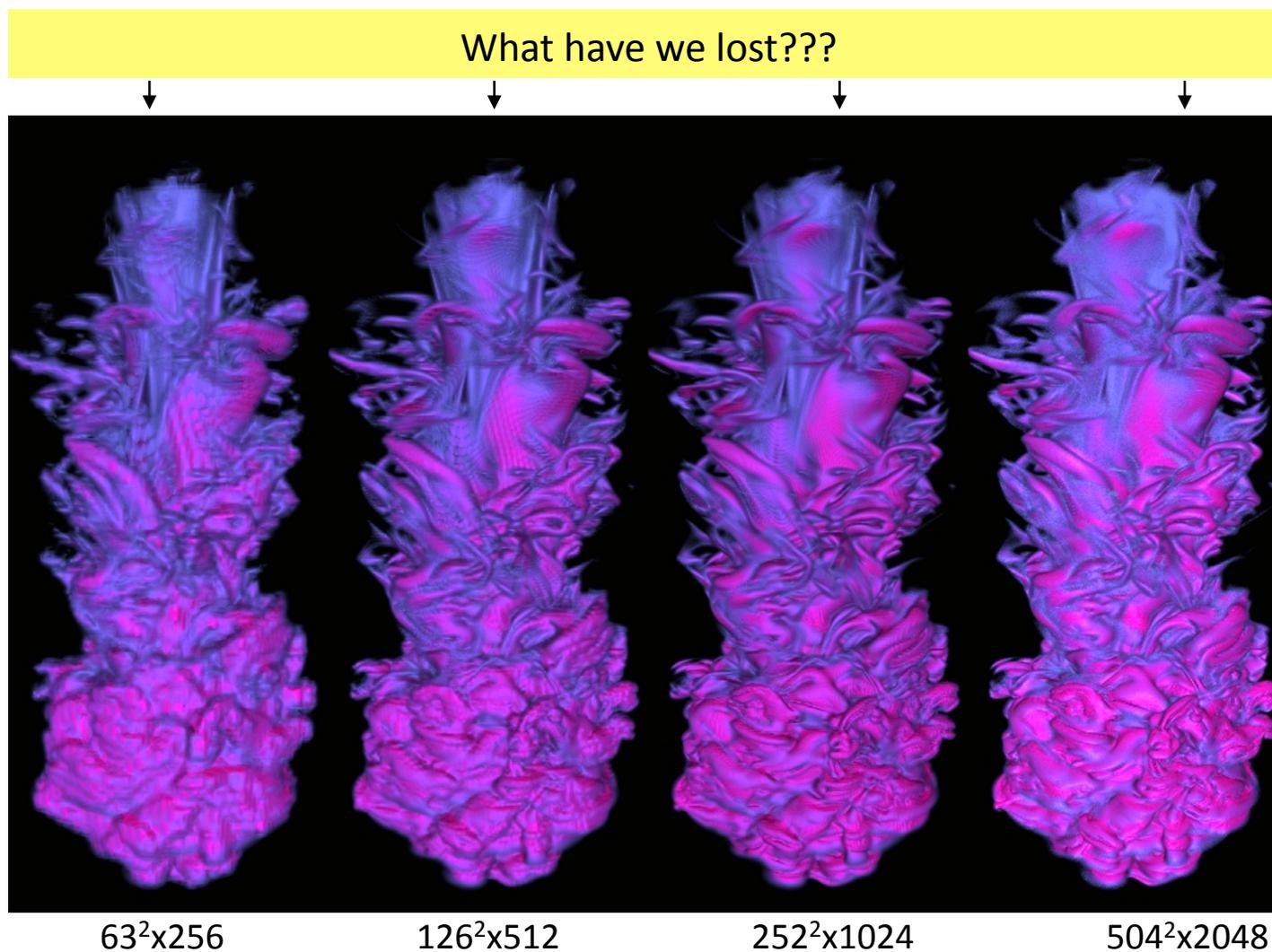
If $u(t)$ are *orthonormal*, then:

$$\text{orthonormal: } \int u_k(t) u_l(t) dt = \begin{cases} 0, & k \neq l \\ 1, & k = l \end{cases}$$

$$L^2 = \sum_{i=M}^{N-1} (a_{\pi(i)})^2 = \left\| f(t) - \hat{f}(t) \right\|_2^2, \quad \text{where } a_{\pi(i)} \text{ are discarded coefficients}$$

- The L^2 error is the sum of the squares of the coefficients we leave out!
- So to minimize the L^2 error, we simply **discard** (or **delay** transfer) the smallest coefficients!
- If discarded coefficients are zero, there is no information loss!

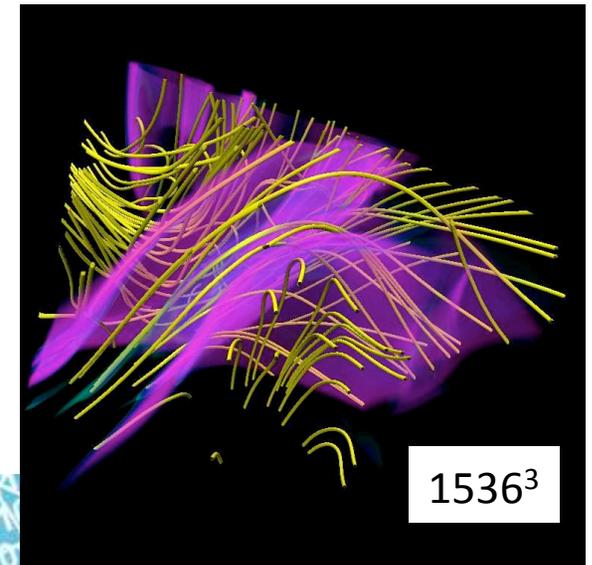
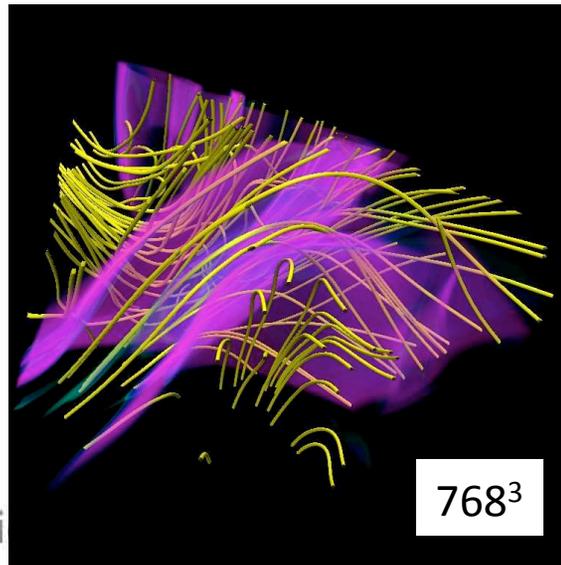
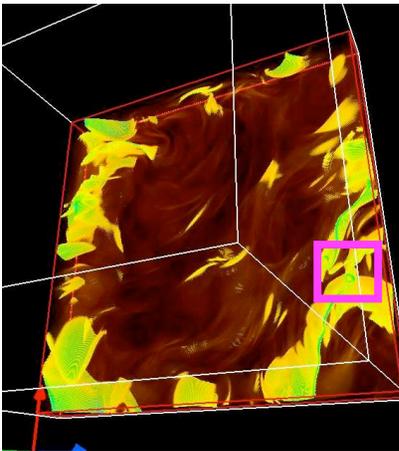
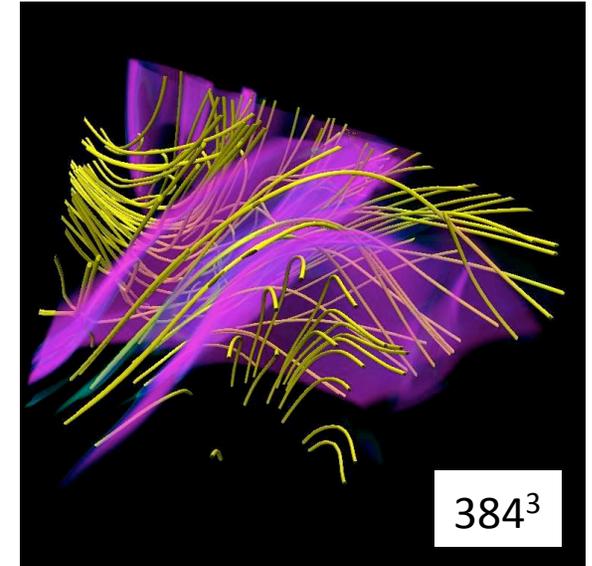
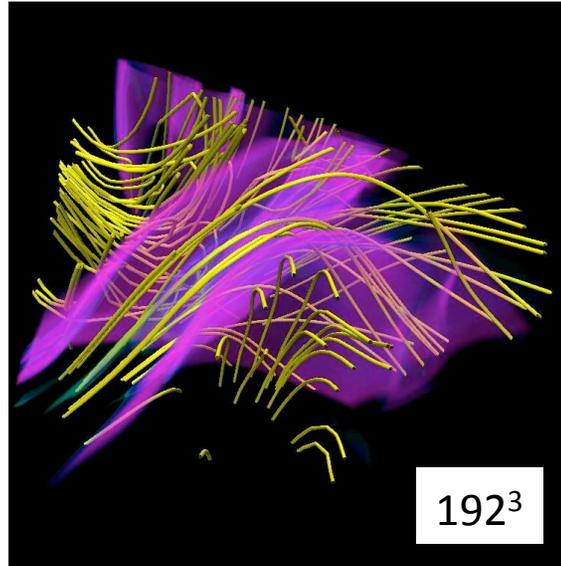
A simulated solar thermal plume at varying grid resolutions provided
Compression: *Frequency Truncation*



Integration of magnetic field lines

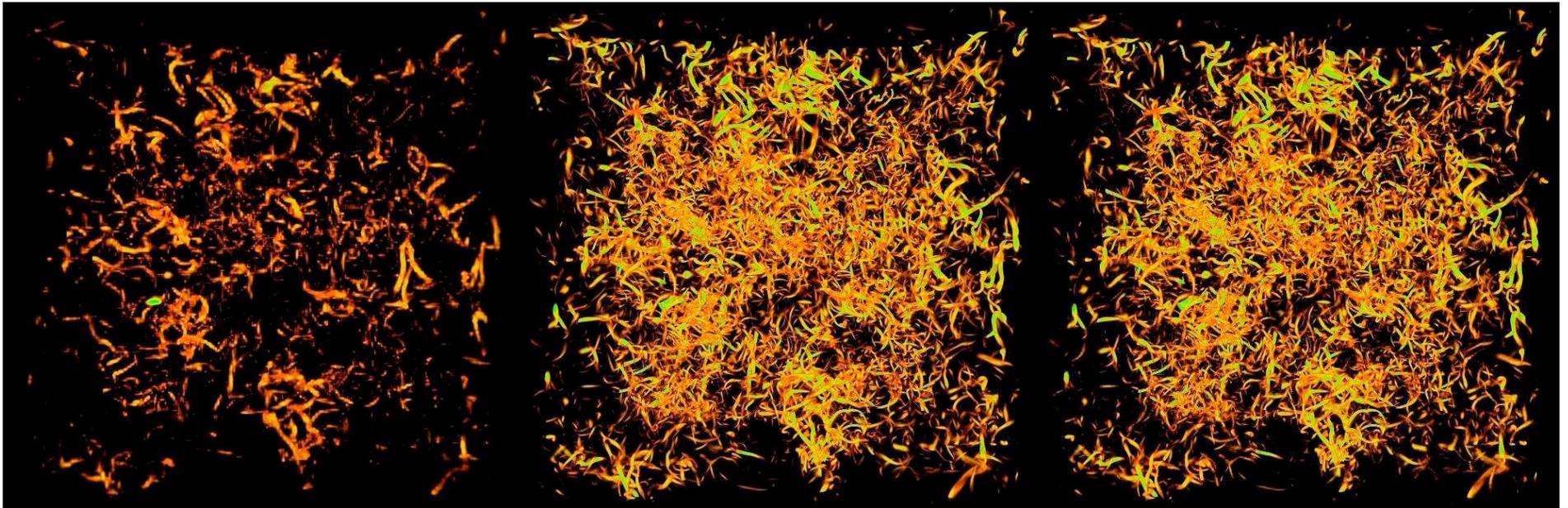
Compression: frequency truncation

- 1536^3 MHD Simulation
- 4th order Runge-Kutte
- Mininni et al. (2007)



Frequency vs coefficient prioritization

64:1 Compression



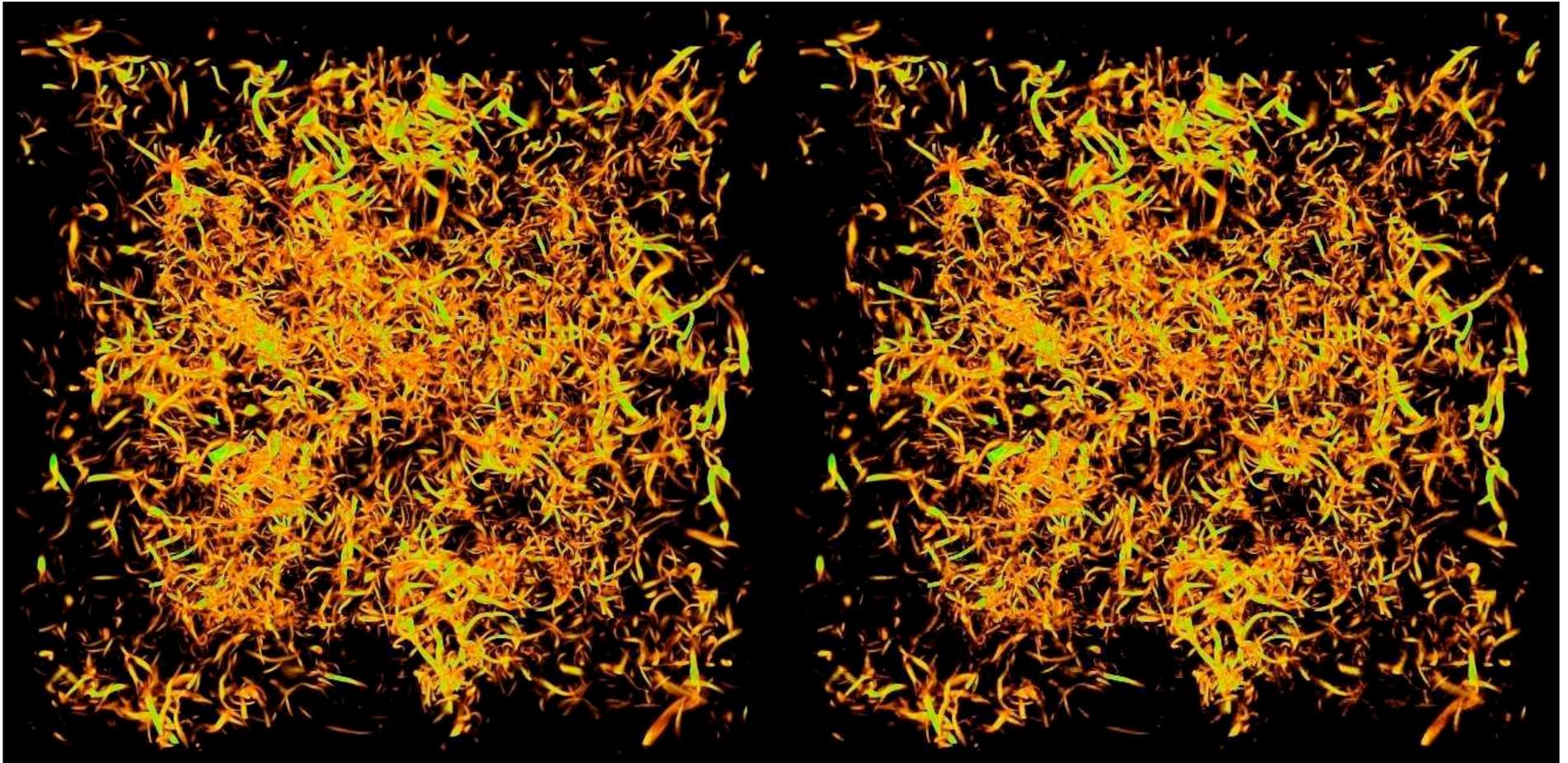
Frequency truncation

No compression

Coefficient prioritization

1024³ Taylor-Green turbulence (enstrophy field) [P. Mininni, 2006]

100:1 Compression with coefficient prioritization
1024³ Taylor-Green turbulence (enstrophy field) [P. Mininni, 2006]



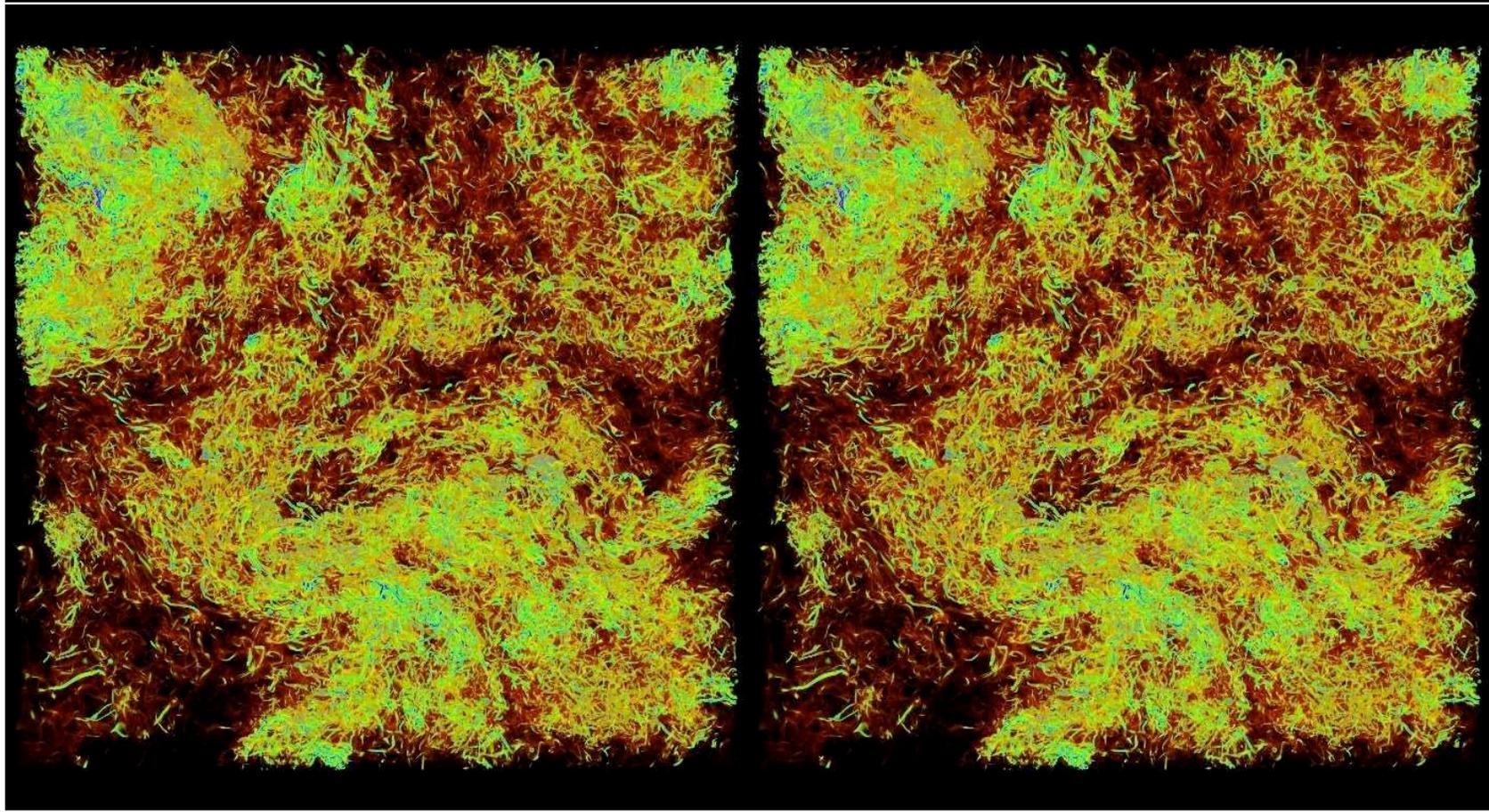
No compression

Coefficient prioritization (VDC2)

4096³ Homogenous turbulence simulation
Volume rendering of original enstrophy field and 800:1 compressed field

Original: 275GBs/field

800:1 compressed: 0.34GBs/field



Data provided by P.K. Yeung at Georgia Tech and Diego Donzis at Texas A&M

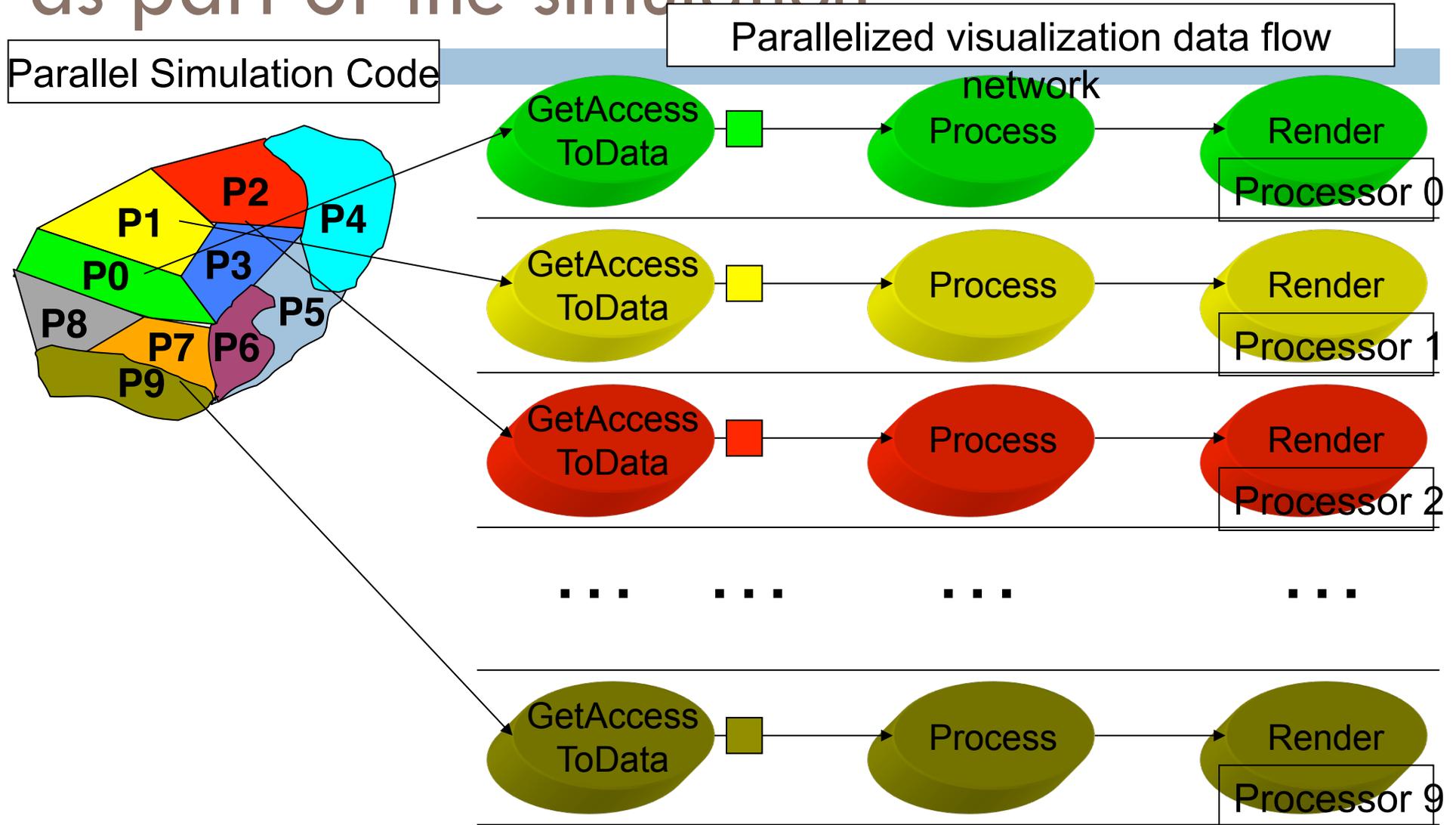


Outline



- Multi-resolution processing
 - Space filling curves (from Valerio Pascucci)
 - Wavelet compression (from John Clyne)
- **In situ processing**
 - System overview (from Brad Whitlock)
 - Example in action (from Jean Favre)
- Query-driven visualization
 - Overview (from Wes Bethel)
 - FastBit (from John Wu)
 - Example in action (from Oliver Ruebel)

In situ processing does visualization as part of the simulation



In situ: pros and cons



- Pros:

- No I/O!
- Lots of compute power available

- Cons:

- Very memory constrained
- Many operations not possible
 - Once the simulation has advanced, you cannot go back and analyze it
- User must know what to look a priori
 - Expensive resource to hold hostage!

Difficult conversations in the future.



- Conversations we should be having with our customers...
 - How much memory are you willing to give up for visualization?
 - Will you be angry if the vis algorithms crash?
 - Do you know what you want to generate a priori?
 - Can you re-run simulations if necessary?

Outline



- Multi-resolution processing
 - Space filling curves (from Valerio Pascucci)
 - Wavelet compression (from John Clyne)
- In situ processing
 - System overview (from Brad Whitlock)
 - Example in action (from Jean Favre)
- Query-driven visualization
 - Overview (from Wes Bethel)
 - FastBit (from John Wu)
 - Example in action (from Oliver Ruebel)

Lawrence Livermore National Laboratory

Parallel In Situ Coupling of Simulation with a Fully Featured Visualization System

Brad Whitlock

Lawrence Livermore National Laboratory



Jean M. Favre

Swiss National Supercomputing Centre



CSCS

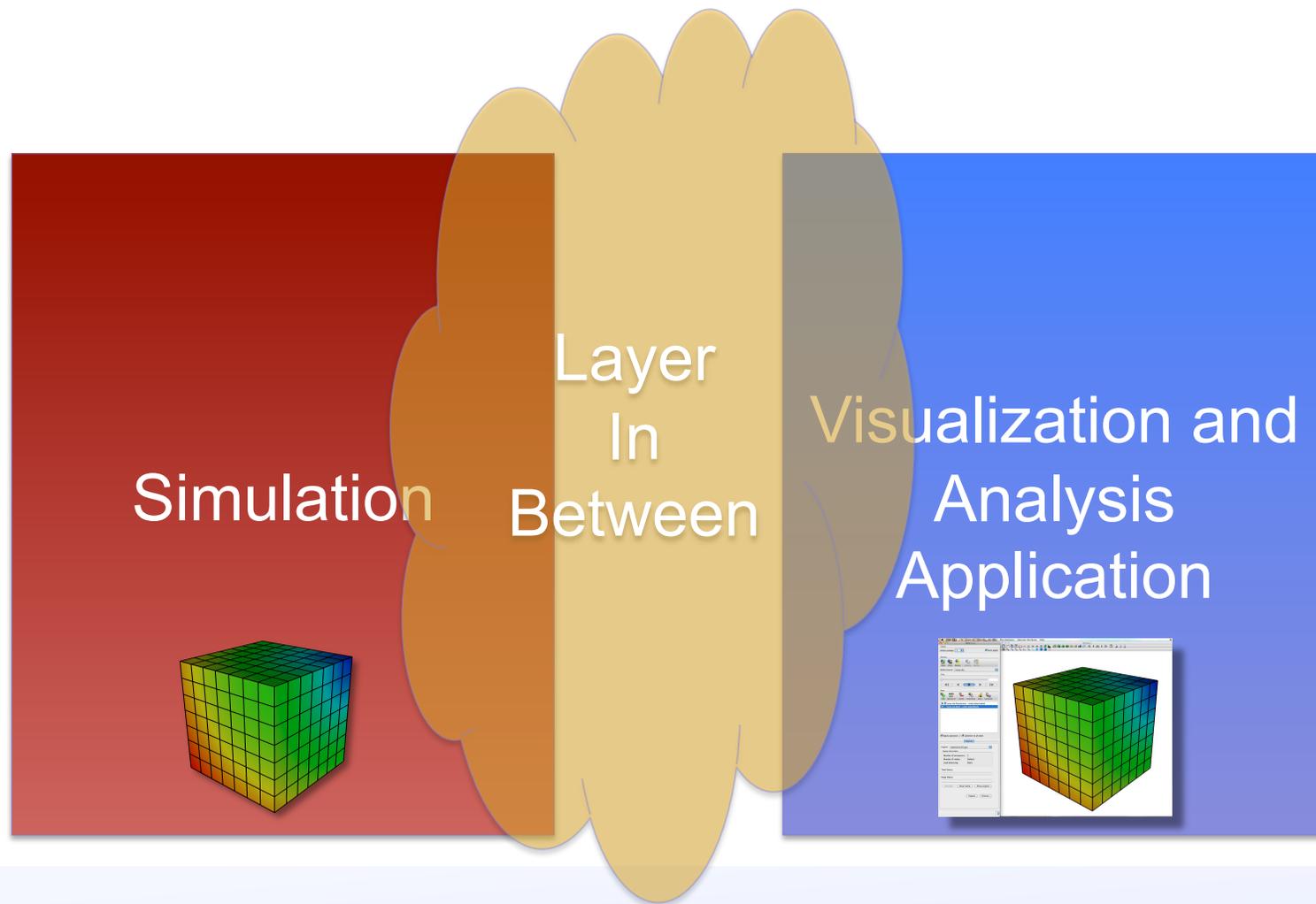
Swiss National Supercomputing Centre

Jeremy S. Meredith

Oak Ridge National Laboratory



A Marriage Between Two Fairly Inflexible Partners...



In Situ Processing Strategies

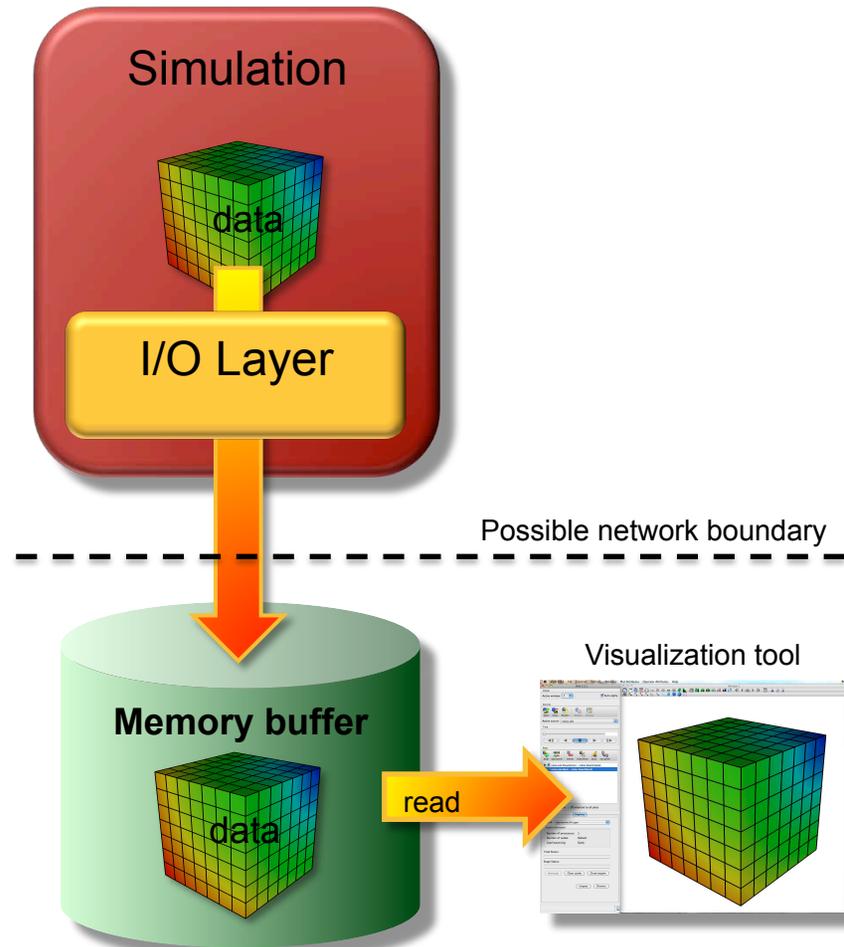
We find 3 main strategies for in situ processing:

In Situ Strategy	Description	Negative Aspects
Loosely coupled	Visualization and analysis run on concurrent resources and access data over network	<ol style="list-style-type: none"> 1) Data movement costs 2) Requires separate resources
Tightly coupled	Visualization and analysis have direct access to memory of simulation code	<ol style="list-style-type: none"> 1) Very memory constrained 2) Large potential impact (performance, crashes)
Hybrid	Data is reduced in a tightly coupled setting and sent to a concurrent resource	<ol style="list-style-type: none"> 1) Complex 2) Shares negative aspects (to a lesser extent) of others



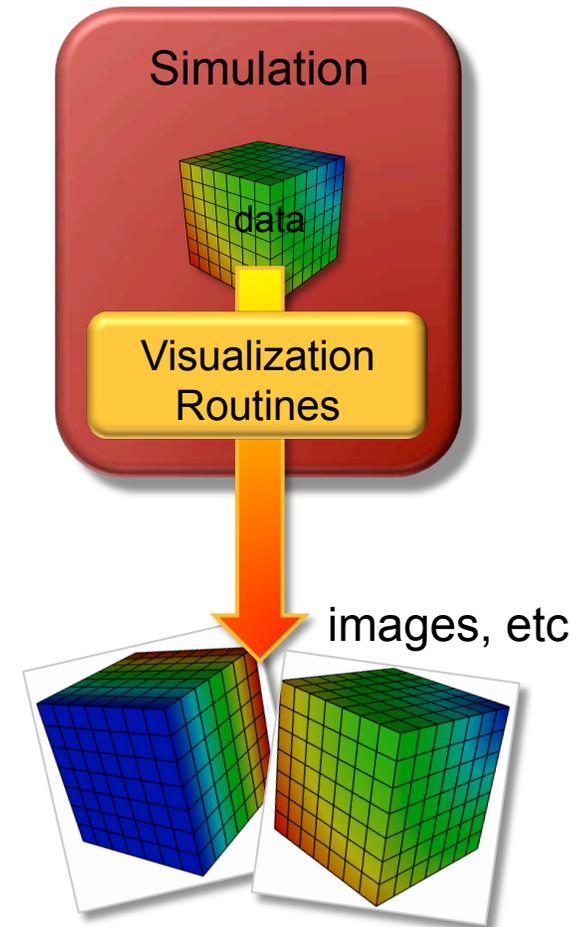
Loosely Coupled In Situ Processing

- I/O layer stages data into secondary memory buffers, possibly on other compute nodes
- Visualization applications access the buffers and obtain data
- Separates visualization processing from simulation processing
- Copies and moves data



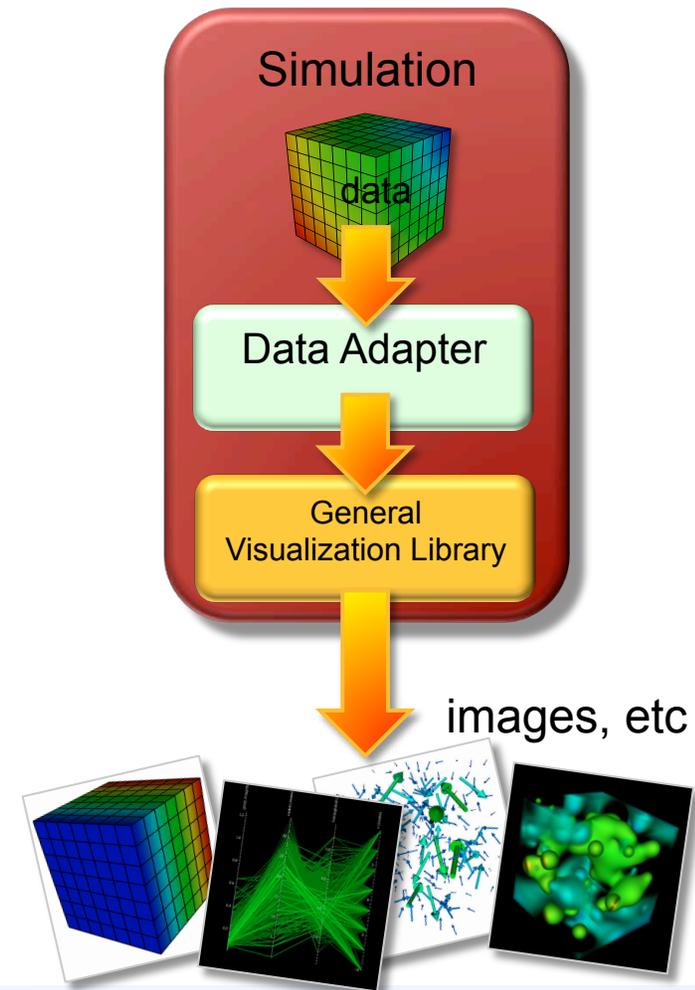
Tightly Coupled Custom In Situ Processing

- Custom visualization routines are developed specifically for the simulation and are called as subroutines
 - Create best visual representation
 - Optimized for data layout
- Tendency to concentrate on very specific visualization scenarios
- *Write once, use once*



Tightly Coupled General In Situ Processing

- Simulation uses data adapter layer to make data suitable for general purpose visualization library
- Rich feature set can be called by the simulation
- Operate directly on the simulation's data arrays when possible
- *Write once, use many times*



Which Strategy is Appropriate?

There have been many excellent papers and systems in this space. Different circumstances often merit different solutions.

	Tightly Coupled	Loosely Coupled	Hybrid
Custom	✘		
General	✘	✘	



Design Philosophy

- Visualization and analysis will be done in the same memory space as the simulation on native data to avoid duplication
- Maximize features and capabilities
- Minimize code modifications to simulations
- Minimize impact to simulation codes
- Allow users to start an in situ session on demand instead of deciding before running a simulation
 - Debugging
 - Computational steering



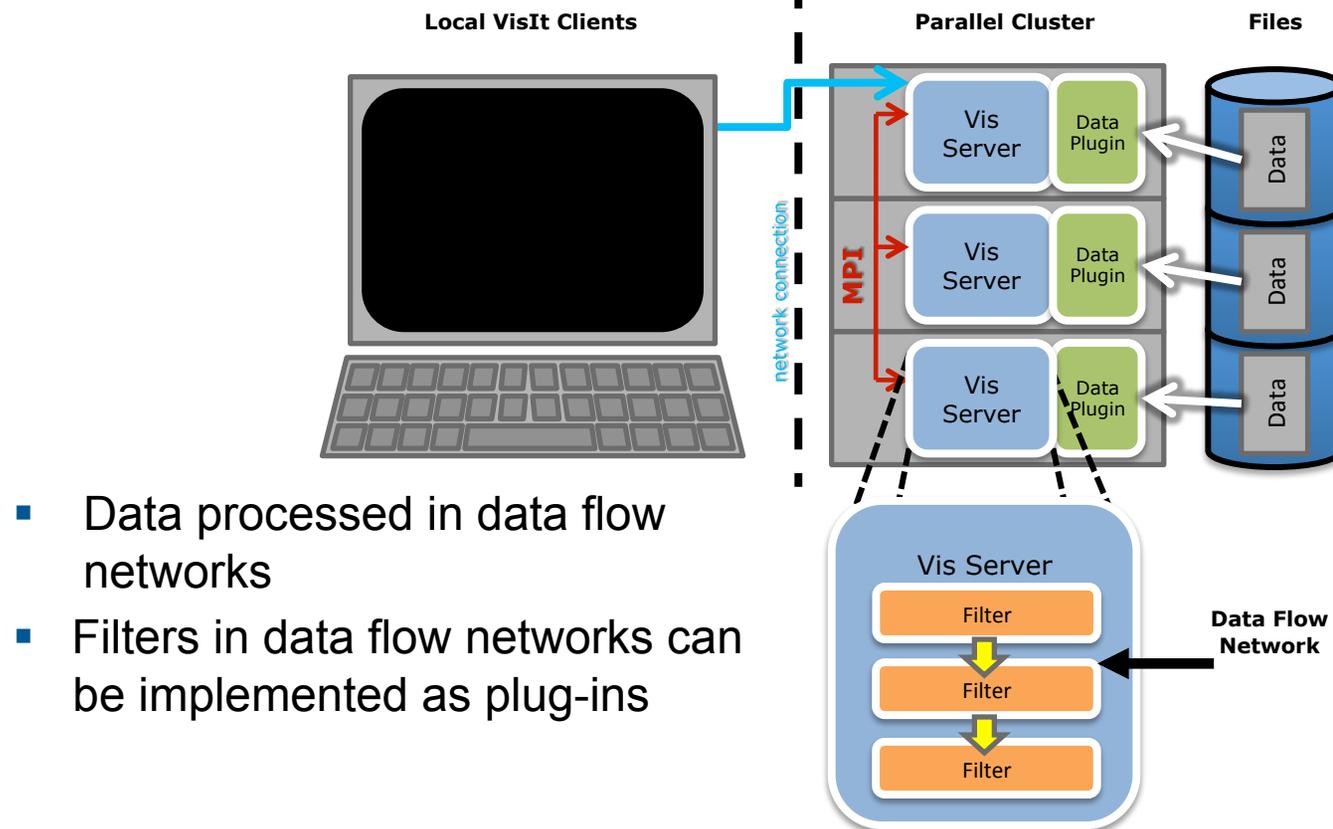
Selecting an In Situ Strategy

- Our strategy is tightly coupled, yet general
- Fully featured visualization code connects interactively to running simulation
 - Allows live exploration of data for when we don't know visualization setup a priori
 - Opportunities for steering
- We chose VisIt as the visualization code
 - VisIt runs on several HPC platforms
 - VisIt has been used at many levels of concurrency
 - We know how to develop for VisIt



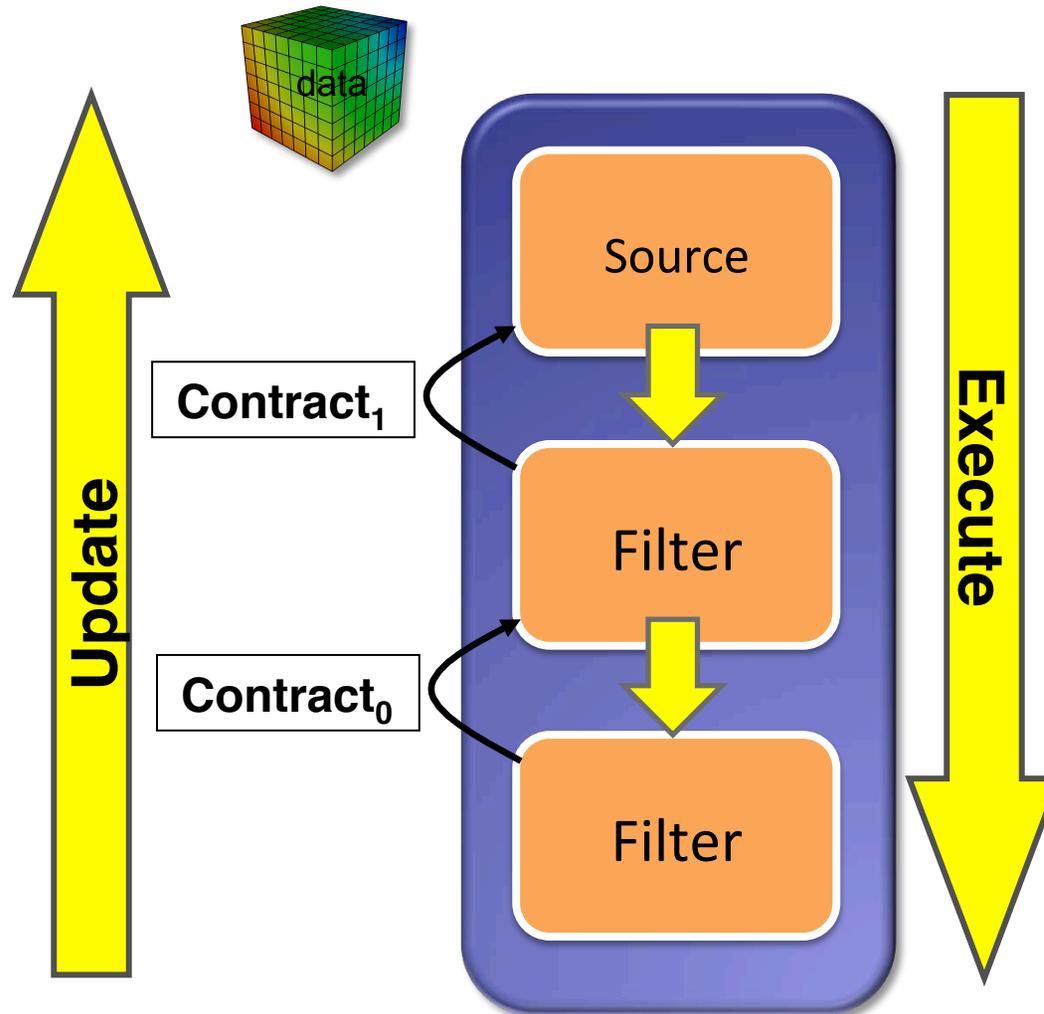
Visualization Tool Architecture

- Clients runs locally and display results computed on the server
- Server runs remotely in parallel, handling data processing for client



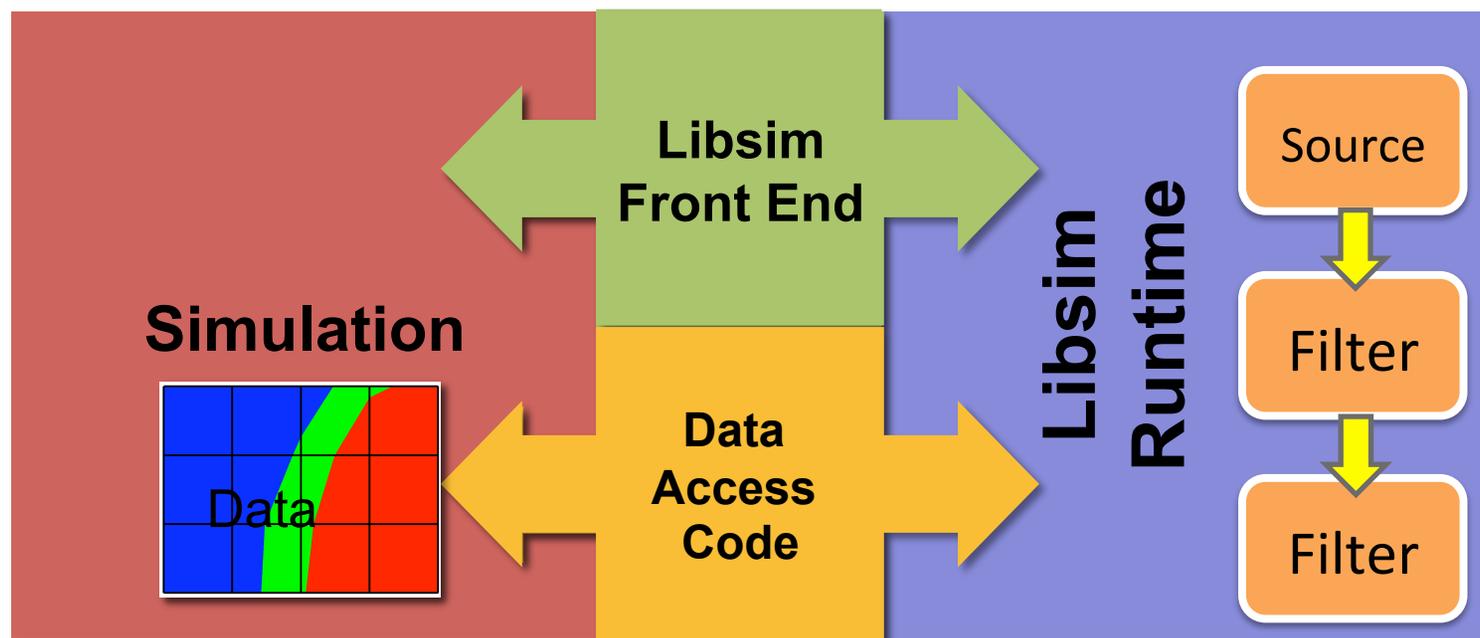
- Data processed in data flow networks
- Filters in data flow networks can be implemented as plug-ins

Coordination Among Filters Using Contracts



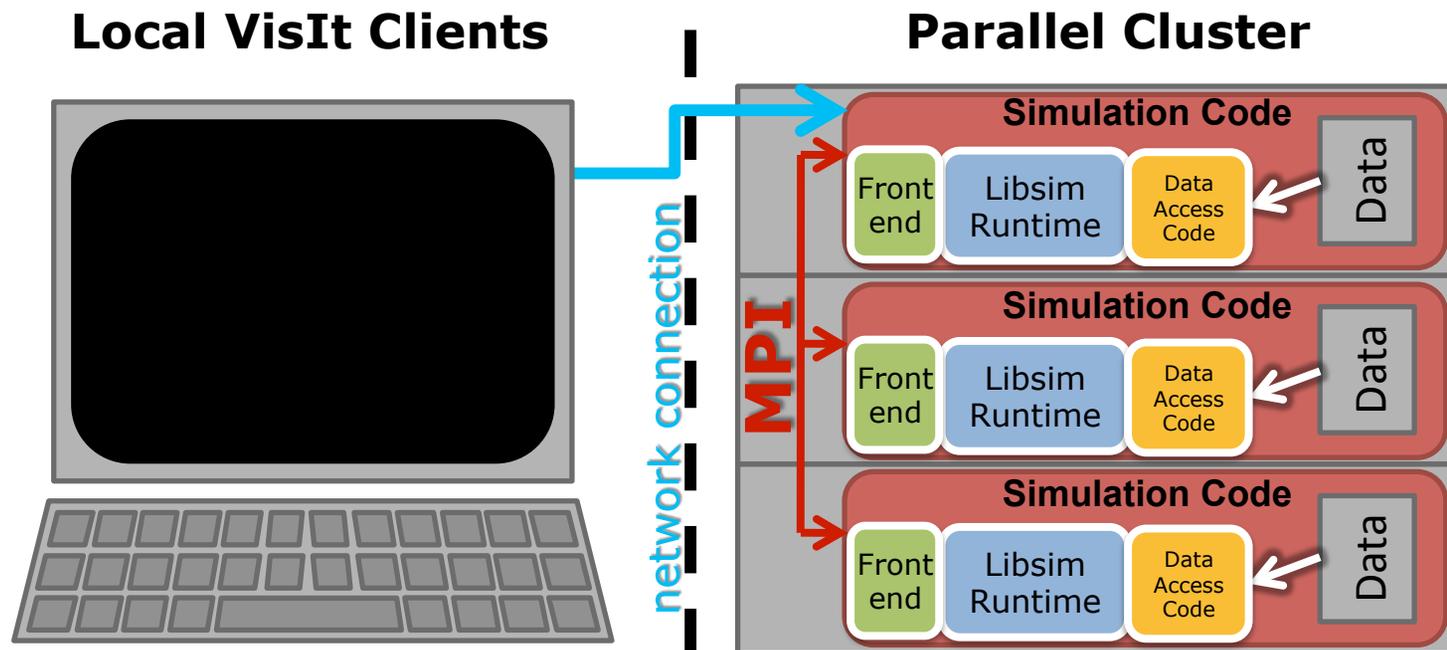
Coupling of Simulations and VisIt

- We created Libsim, a library that simulations use to let VisIt connect and access their data



A Simulation Using Libsim

- Front end library lets VisIt connect
- Runtime library processes the simulation's data
- Runtime library obtains data on demand through user-supplied *Data Access Code* callback functions



In Situ Processing Workflow

1. The simulation code launches and starts execution
2. The simulation regularly checks for connection attempts from visualization tool
3. The visualization tool connects to the visualization
4. The simulation provides a description of its meshes and data types
5. Visualization operations are handled via Libsim and result in data requests to the simulation



Instrumenting a Simulation

Additions to the source code are usually minimal, and follow three incremental steps:

Initialize Libsim and alter the simulation's main iterative loop to listen for connections from VisIt.

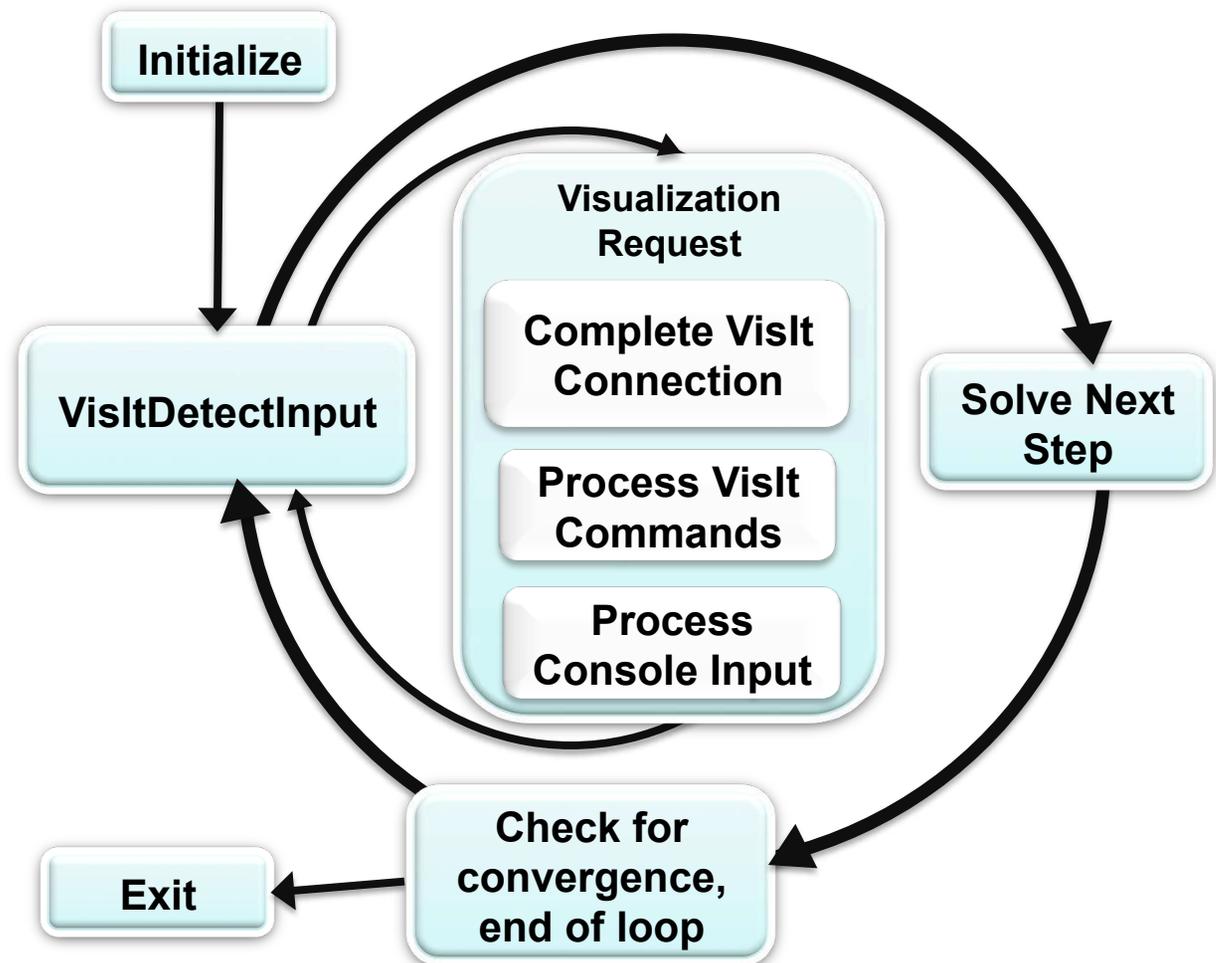
Create *data access callback* functions so simulation can share data with Libsim.

Add control functions that let VisIt steer the simulation.



Adapting the Main Loop

- Libsim opens a socket and writes out connection parameters
- VisItDetectInput checks for:
 - Connection request
 - VisIt commands
 - Console input



Sharing Data

- Visit requests data on demand through data access callback functions
 - Return actual pointers to your simulation's data (nearly zero-copy)
 - Return alternate representation that Libsim can free
 - Written in C, C++, Fortran, Python

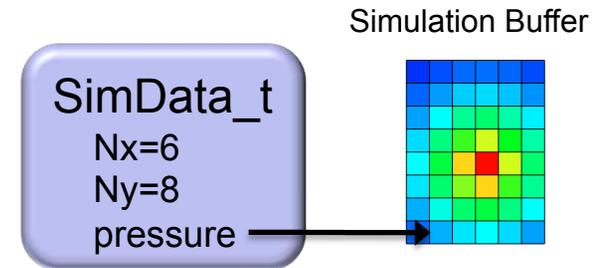


Sharing Data Example

```

// Example Data Access Callback
visit_handle
GetVariable(int domain, char *name,
void *cbdata)
{
    visit_handle h = VISIT_INVALID_HANDLE;
    SimData_t *sim = (SimData_t *)cbdata;
    if(strcmp(name, "pressure") == 0)
    {
        VisIt_VariableData_alloc(&h);
        VisIt_VariableData_setDataD(h,
            VISIT_OWNER_SIM,
            1, sim->nx*sim->ny,
            sim->pressure);
    }
    return h;
}

```

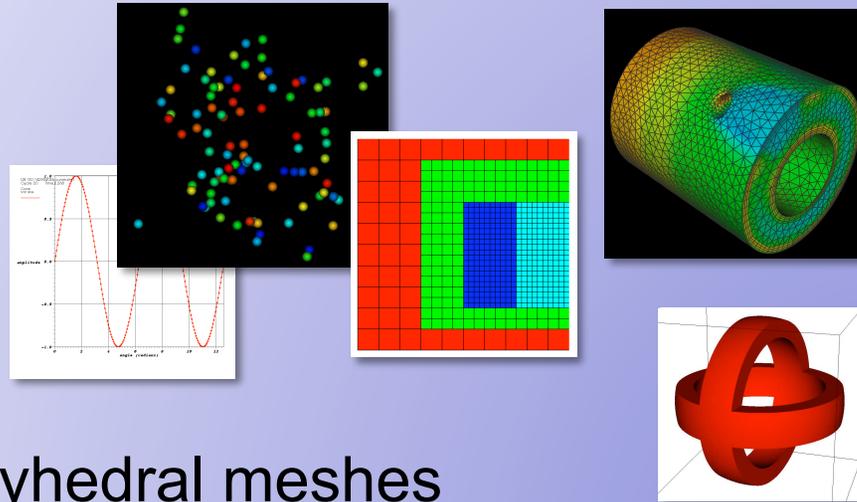


Pass simulation
buffer to Libsim

Supported Data Model

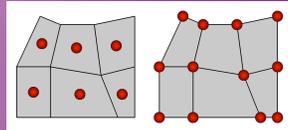
■ Mesh Types

- Structured meshes
- Point meshes
- CSG meshes
- AMR meshes
- Unstructured & Polyhedral meshes



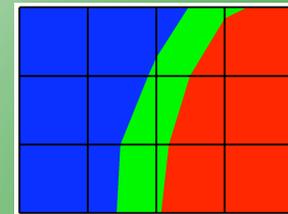
■ Variables

- 1 to N components
- Zonal and Nodal



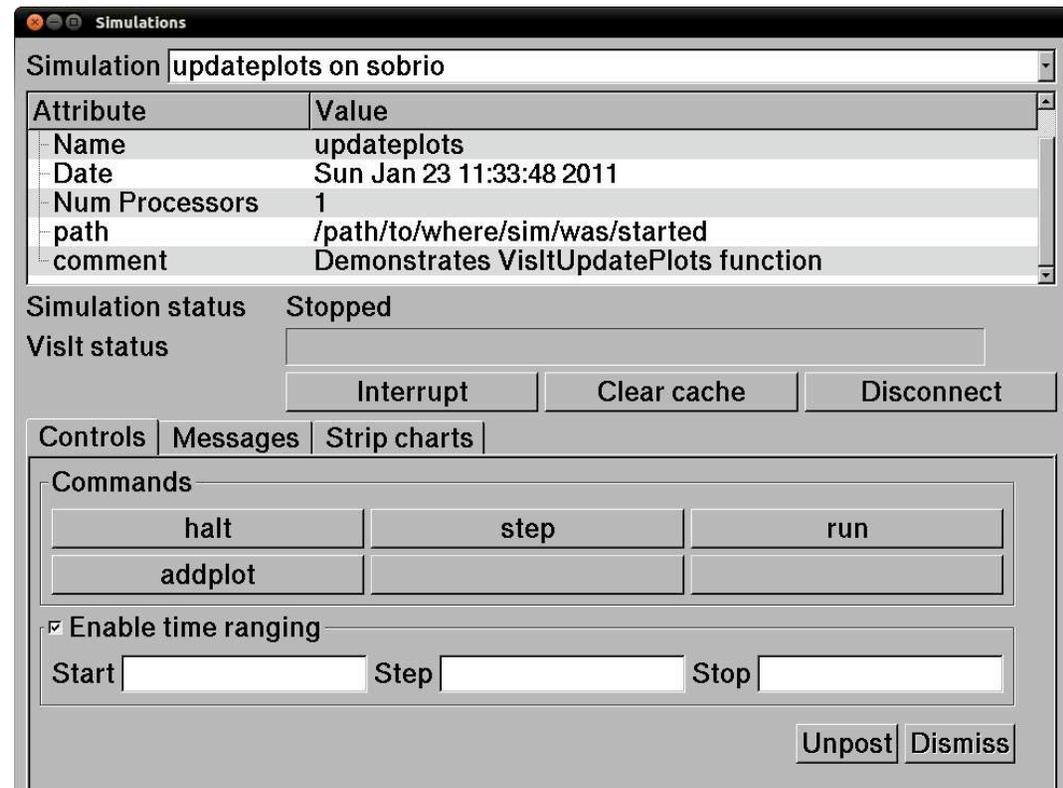
■ Materials

■ Species



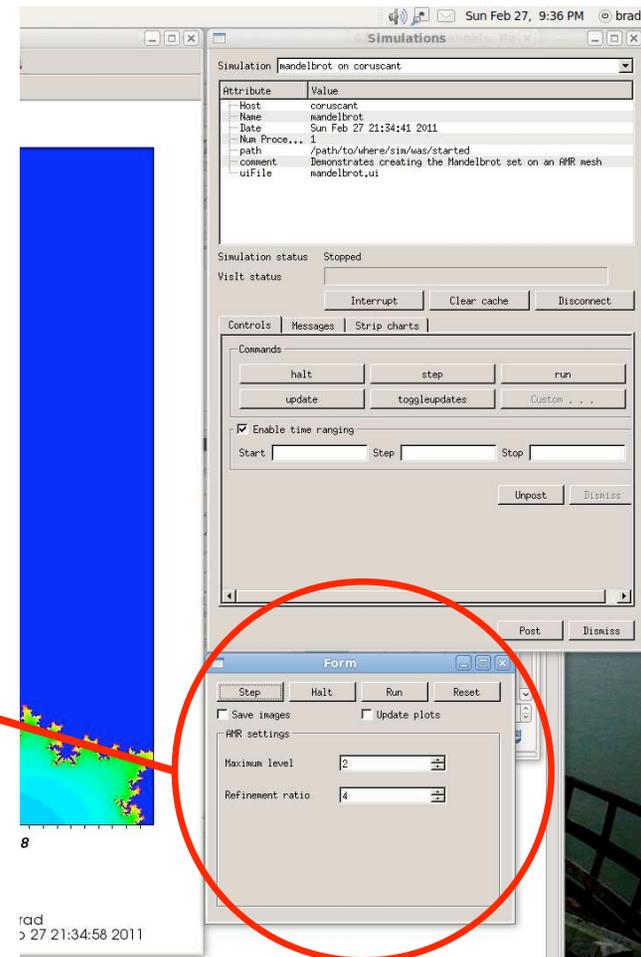
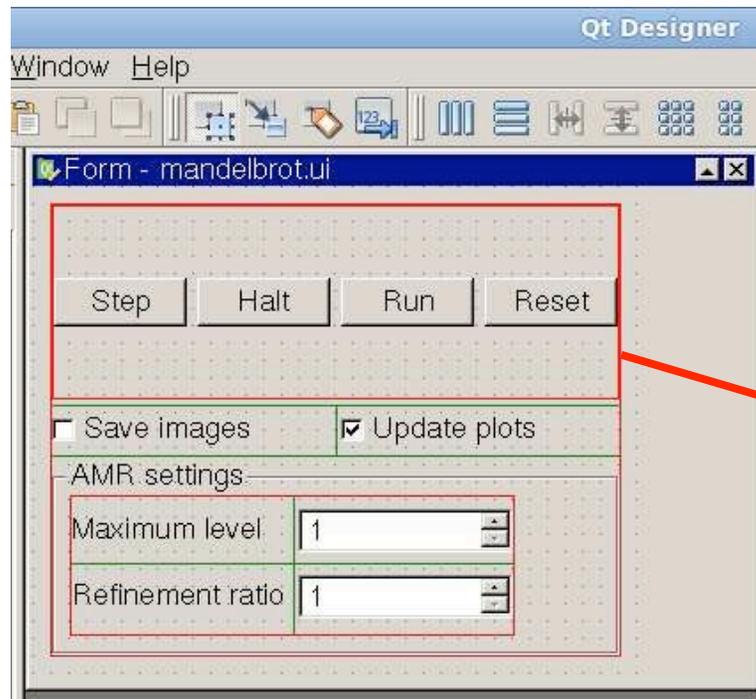
Adding Control Functions

- The simulation provides commands to which it will respond
- Commands generate user interface controls in Simulations Window



Custom User Interfaces

- Simulation can provide UI description for more advanced computational steering



Libsim in Practice

- We conducted our experiments on a 216 node visualization cluster
 - Two, 6 core 2.8GHz Intel Xeon 5660 processors
 - 96Gb of memory per node
 - InfiniBandQDR high-speed interconnect
 - Lustre parallel file system
- We measured the impact of Libsim on a simulation's main loop, without connecting to VisIt
- We measured memory usage after loading VisIt
- We instrumented GADGET-2, a popular cosmology code, with Libsim and measured performance



Impact on the Main Loop

- Measure cost of calling Libsim in the main loop
- Instrumenting the main loop for a parallel simulation requires calling *VisitDetectInput* and *MPI_Bcast*
 - We timed how long it took to call both using 512 cores
 - 10K main loop iterations

Cores	VisitDetectInput overhead	MPI_Bcast overhead	Overhead loading Visit runtime libraries
512	2 μ s	8 μ s	1s (1 time cost)



Impact on Memory Usage

- Measure memory used before and after VisIt is connected
- Measured our updateplots example program
- Read values from `/proc/<pid>/smaps`

Event	Size	Resident Set Size
Simulation startup	8.75 Mb	512 Kb
After Libsim Initialization	8.75 Mb	614 Kb
After Loading VisIt	222 Mb	43.5 Mb



Impact on GADGET-2 Simulation

- GADGET-2, a distributed-memory parallel code for cosmological simulations of structure formation
- We measured in situ performance versus I/O performance at 3 levels of concurrency
 - Render 2048*2048 pixel image
 - Collective I/O and file-per-process I/O
 - 16 million particles and 100 million particles
- Results show that in situ using a fully featured visualization system can provide performance advantages over I/O
 - We relied on VisIt's ability to scale (*runs up to 65,356 cores have been demonstrated*)



Libsim/GADGET-2 Timing Results

16M particles	32 cores	256 cores
I/O 1 file	2.76s	4.72s
I/O N files	0.74s	0.31s
In situ	0.77s	0.34s

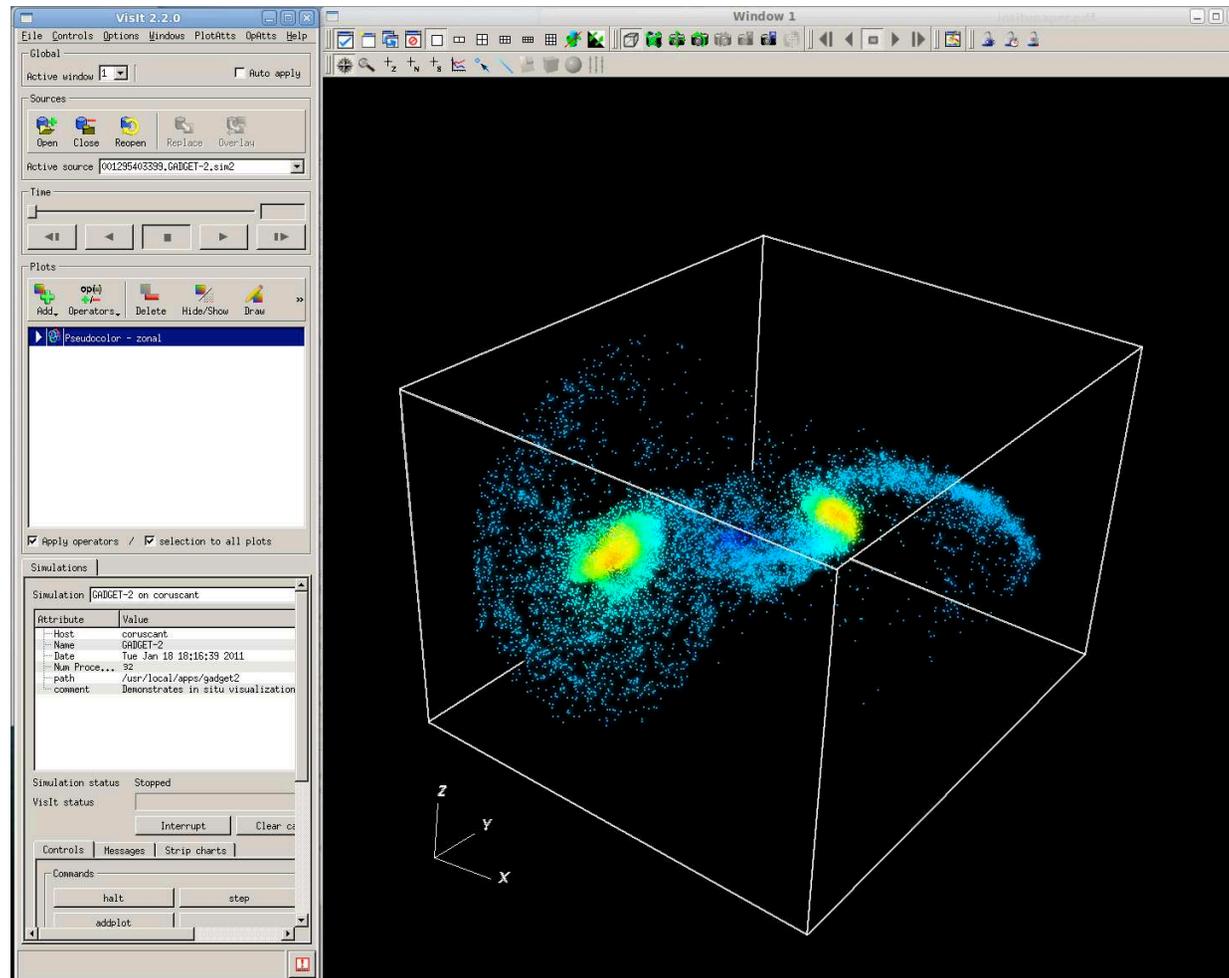
100M particles	32 cores	256 cores	512 cores
I/O 1 file	24.45s	26.7s	25.27s
I/O N files	0.69s	1.43s	2.29s
In situ	1.70s	0.46s	0.64s

- I/O results are the average of 5 runs per test case
- In Situ results are averaged from timing logs for multiple cores

- In situ competitive or faster than single file I/O with increasing cores
- It should be possible to do several in situ operations in the time needed for I/O
- Time savings compared to simulation followed by post processing

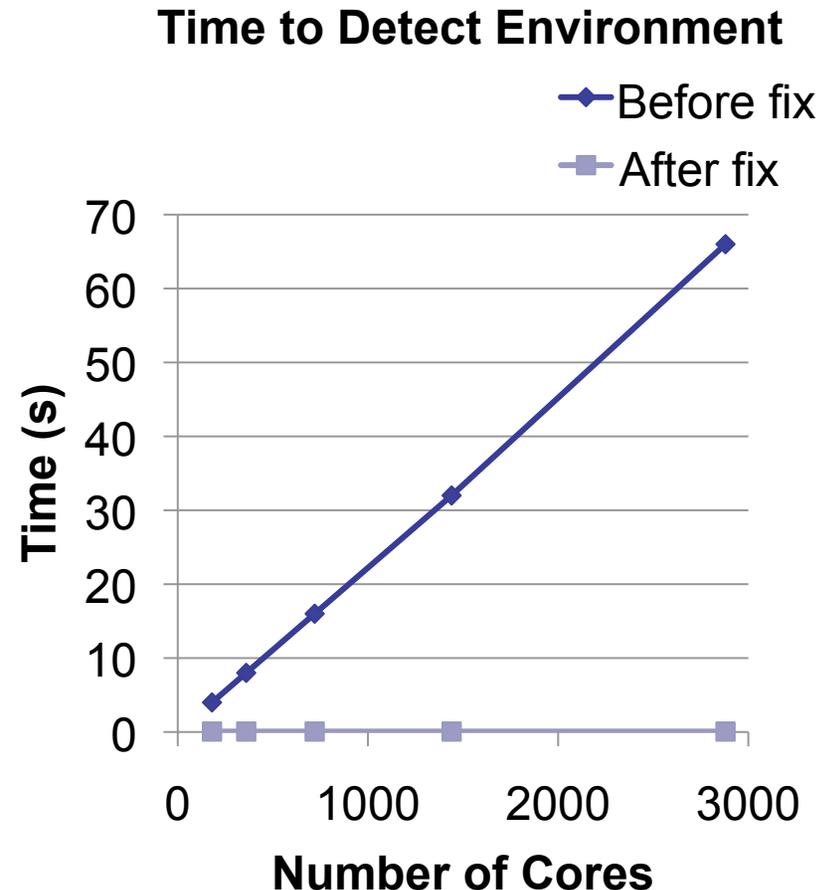


Visit Connected Interactively to GADGET-2



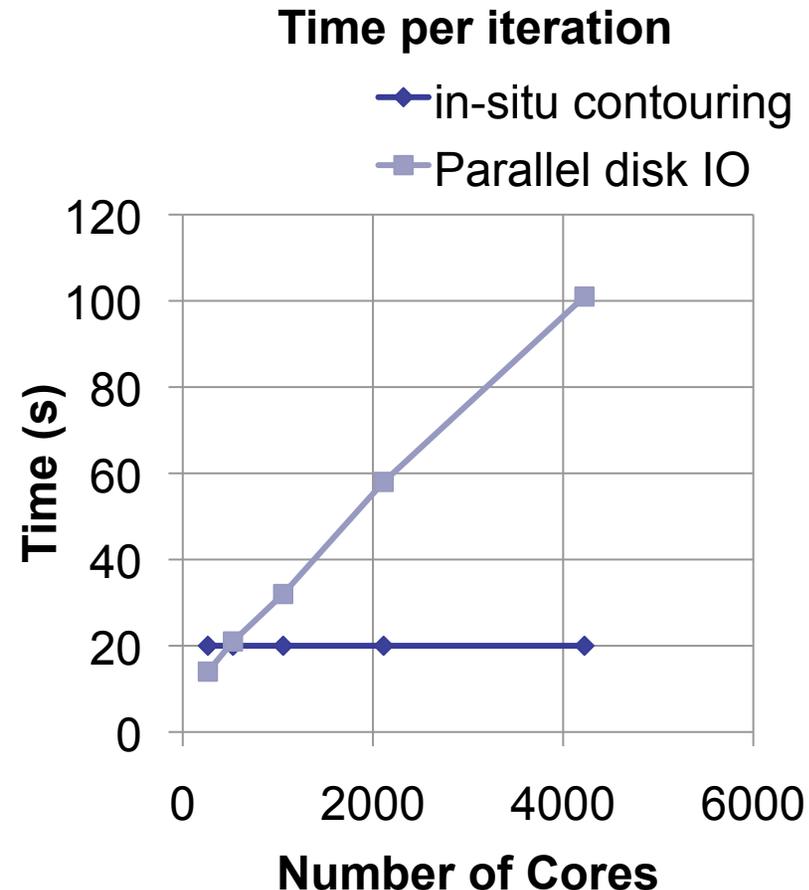
Additional Results

- We have recently instrumented additional simulations to investigate Libsim's scaling properties on a Cray XE6 using up to 4224 cores
- We identified and corrected a performance bottleneck in Libsim's environment detection functions



Additional Results

- Simulation was run on 11, 22, 44, 88 and 176 nodes (24 cores/node)
- Each MPI task had a 512x512x100 block of data to isocontour at 10 different thresholds
- Parallel I/O to disk was done with netCDF-4, in files of size 27, 55, 110, 221, and 442 Gb per iteration



Limitations of Implementation

- Memory intensive
 - Runtime library cost is larger than with static-linking since we use the whole feature set
 - Filters may use intermediate memory
 - Zero-copy is not fully implemented
- We currently require an interactive session though with some changes we could avoid this



Future Work

- Provide additional functions for setting up visualization so in situ processing can be less user-driven
- Further limit resources consumed by the VisIt runtime libraries in order to lessen the impact that in situ analysis has on the simulation
- Characterize performance costs of using shared libraries on larger scale runs



Conclusion

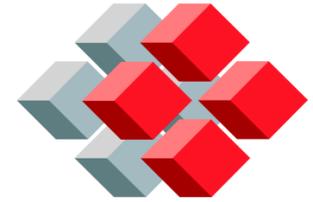
- We have implemented Libsim, an easy to use library that enables in situ computations
 - Provides access to a fully featured, parallel visualization and analysis tool that excels at scale
 - Minimizes impact to simulation performance
 - Minimizes the amount of new code that must be written
 - Fully integrated with the open-source distribution of VisIt



Outline



- Multi-resolution processing
 - Space filling curves (from Valerio Pascucci)
 - Wavelet compression (from John Clyne)
- In situ processing
 - System overview (from Brad Whitlock)
 - Example in action (from Jean Favre)
- Query-driven visualization
 - Overview (from Wes Bethel)
 - FastBit (from John Wu)
 - Example in action (from Oliver Ruebel)



In-situ Visualization Examples

Dr. Jean M. Favre
Scientific Computing Research Group

20-05-2011

Outline

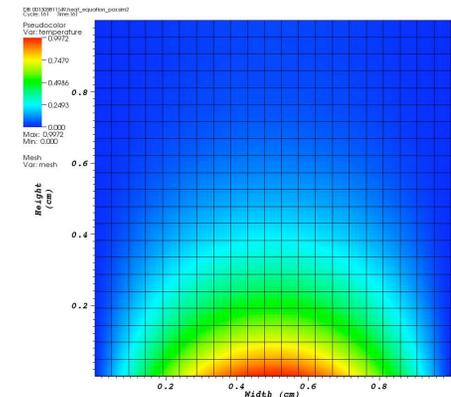
- Problem description
- A 2D solver with parallel partitioning
 - Need for ghost-nodes
- *In-situ* visualization
 - Source code instrumentation
 - Specify ghost-nodes
 - Single stepping through the execution



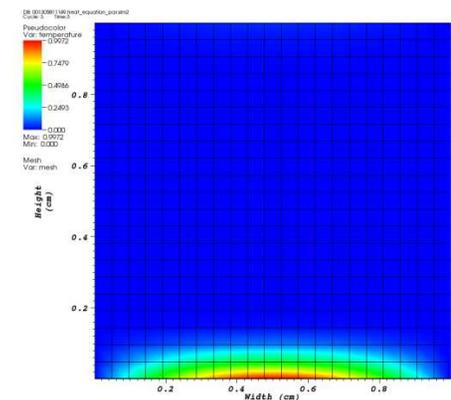
Solving a PDE and visualizing the execution

Full source code solution is given here:

- <http://portal.nersc.gov/svn/visit/trunk/src/tools/DataManualExamples/Simulations/contrib/pjacobi/>
- C, F90 and Python subdirectories



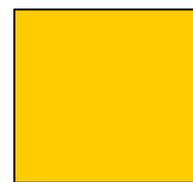
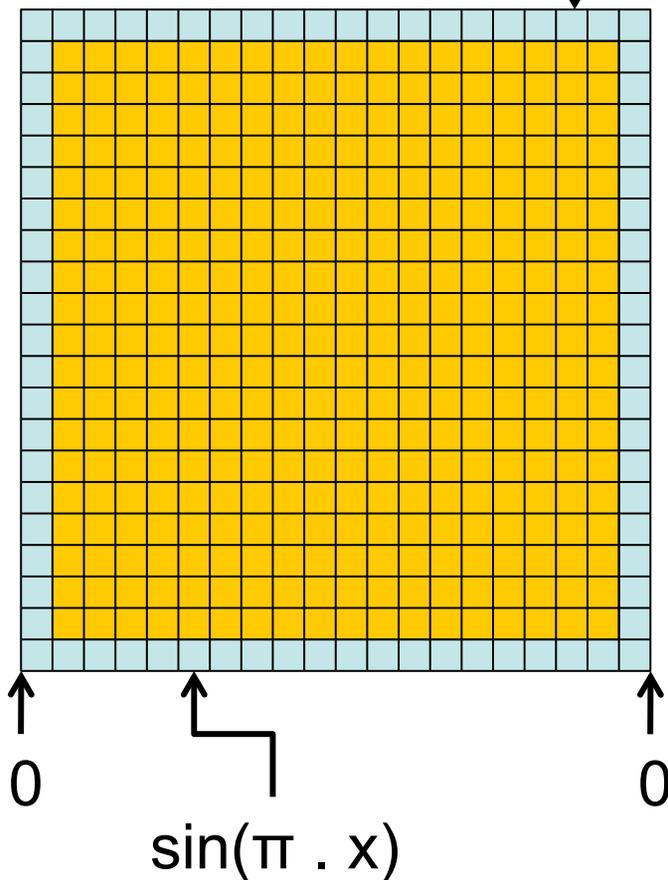
user: fjavie
Thu May 19 15:25:54 2011



user: fjavie
Thu May 19 15:24:58 2011

A PDE with fixed boundary conditions

$$\sin(\pi \cdot x) \cdot \sin(-\pi \cdot y)$$



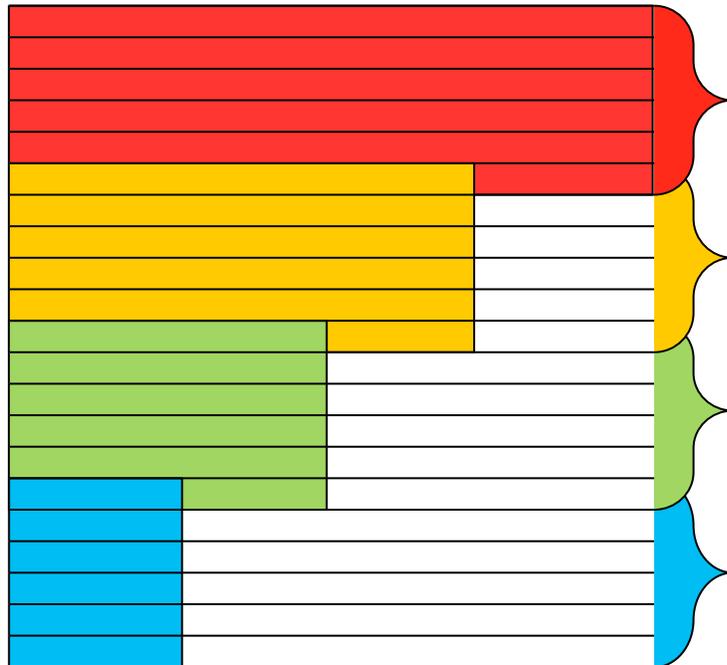
Update grid with solver



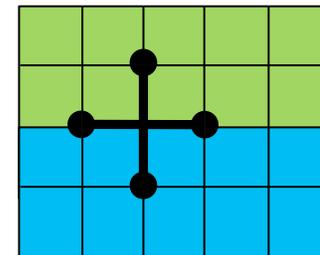
Fixed boundary conditions

Laplace equation: $\Delta u = 0$

2D grid partitioning and initialization

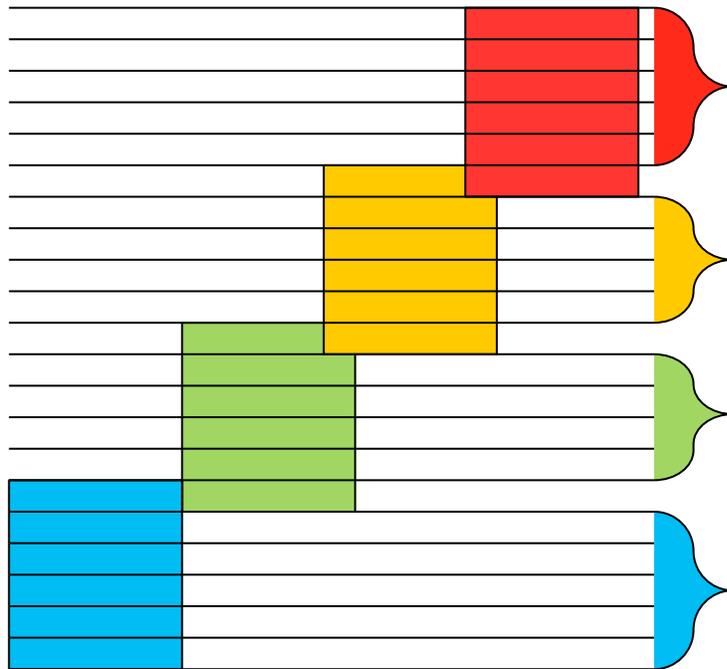


- The grid is partitioned along the Y direction
- Boundary conditions are set
- A single line of ghost-nodes insure that the 5-point stencil is continuous across MPI task boundaries



I/O patterns

Check-pointing and restart



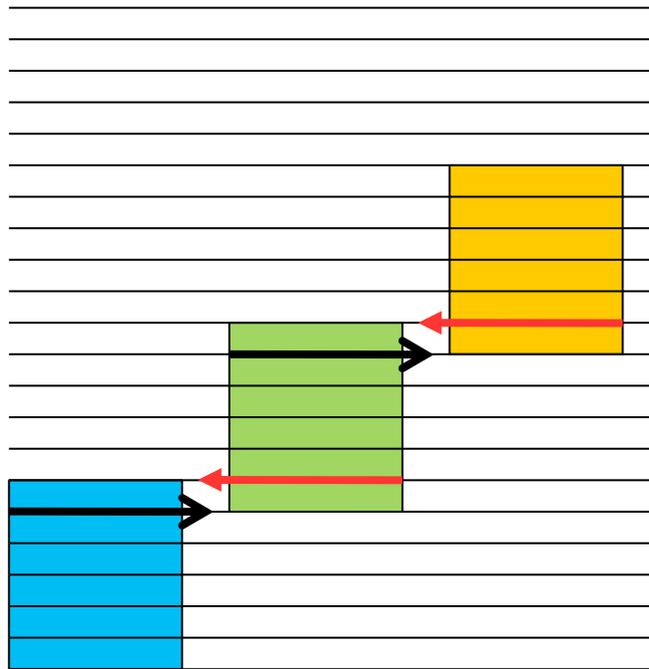
($mp+1$) grid lines to read/write

(mp) grid lines to read/write

(mp) grid lines to read/write

($mp+1$) grid lines to read/write

Ghost data exchange



Overlap Send and Receive

Proc. 0 does not receive from “below”

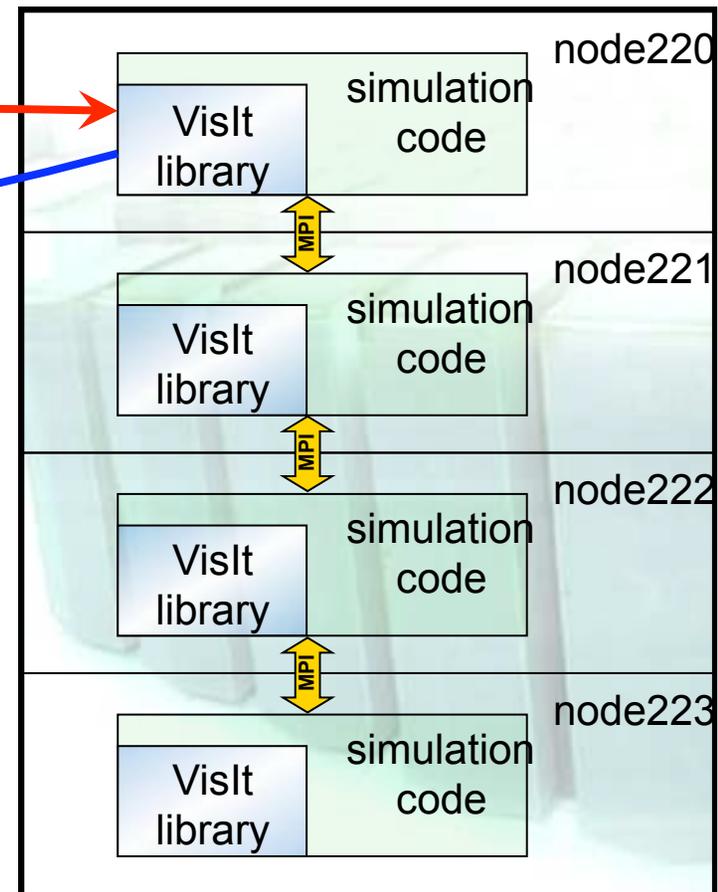
Proc. (N-1) does not send “above”

VisIt's libsim implements a tight coupling

Desktop Machine



Parallel Supercomputer



commands

images

- Link simulation source code with visualization library.
- Data is shared via pointers.

The source code needs to be instrumented

1. The execution flow needs to check for Visualization Requests
2. Once connected, the simulation code needs to advertize what data/meshes are available, and
3. Provide pointers to data, or wrap data into the expected form/shape

Source code examples are instrumented with:

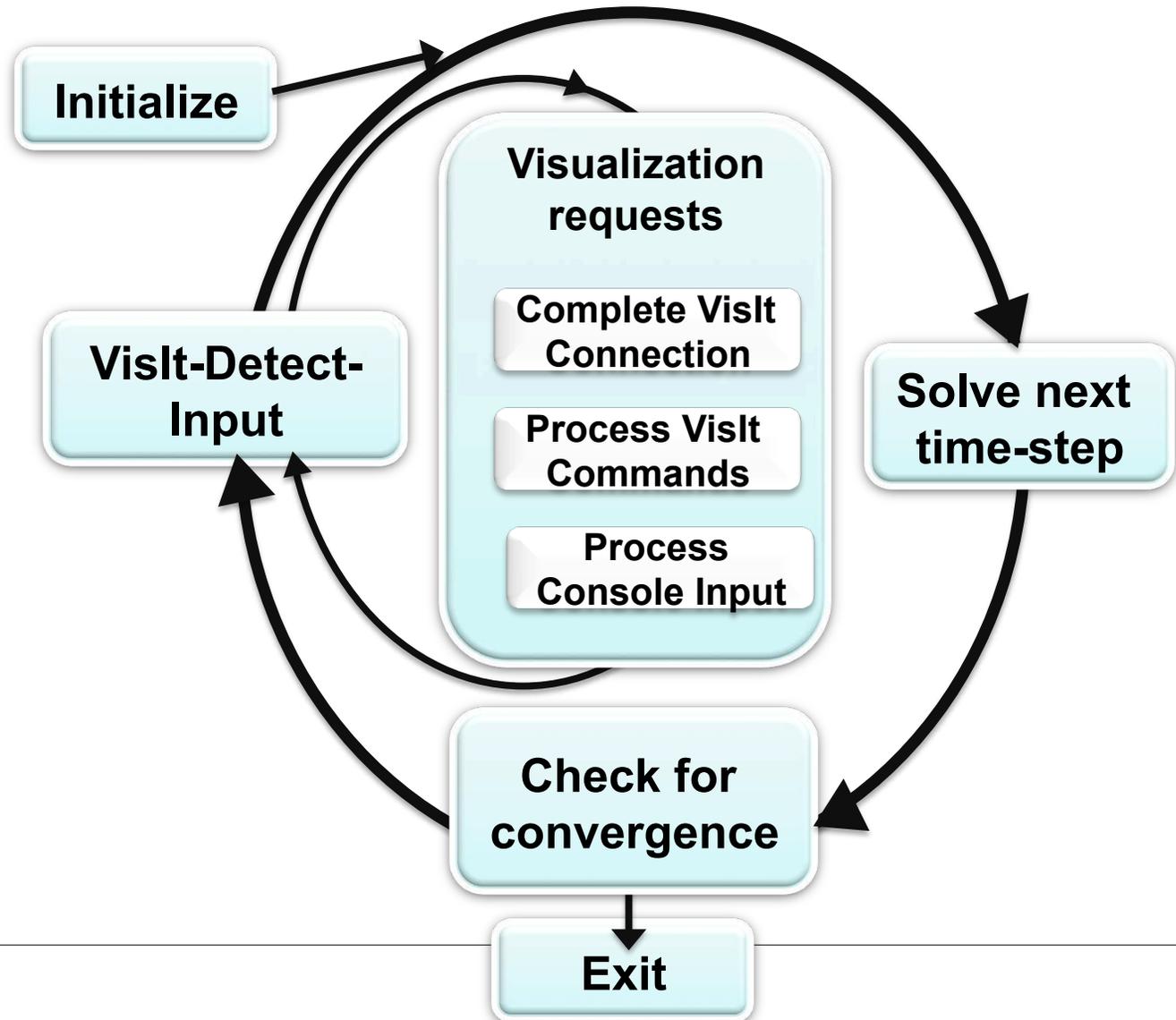
```
#ifdef _VISIT_  
#endif
```

Application's flow diagram (before and after)

Connection to the visualization library is optional

Execution is *step-by-step* or in *continuous* mode

Live connection can be closed and re-opened at later time



Step-by-step or continuous execution

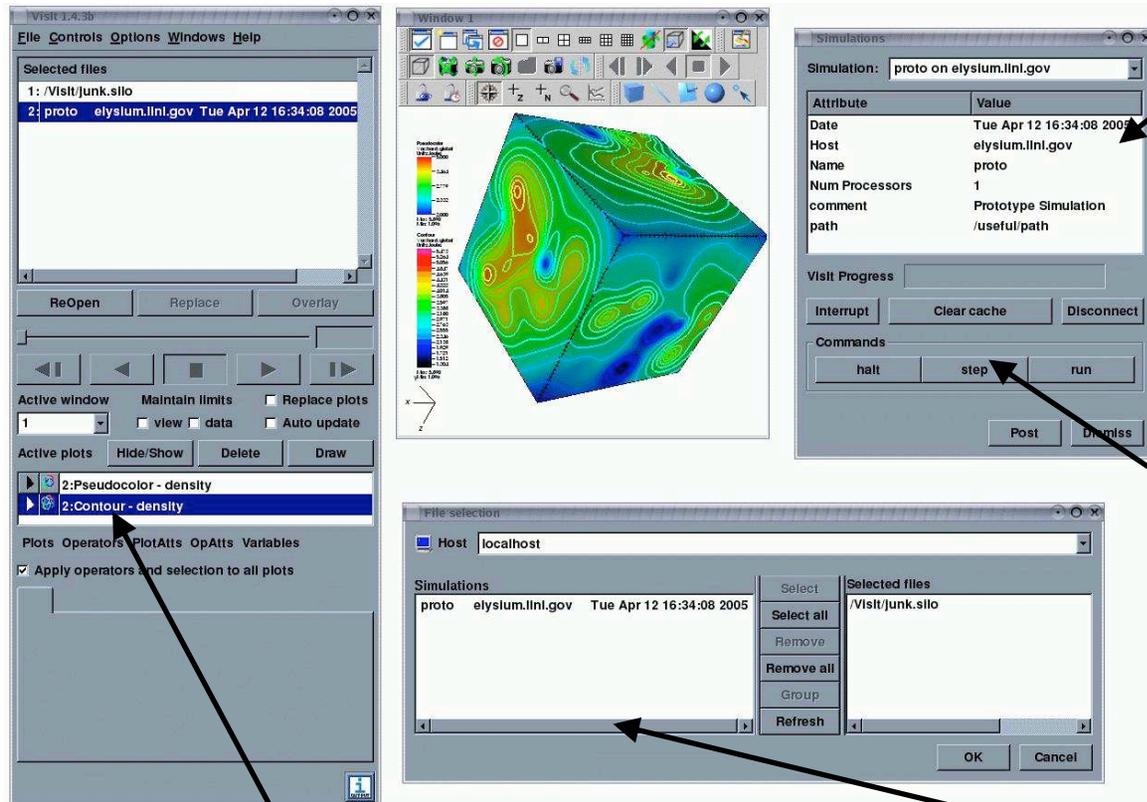
- A simulation would normally not wait for a connection and execute as fast as possible.

These examples however, pause immediately, so they won't finish before you have time to connect!

The call `visitdetectinput(bool blocking, -1)` instructs the simulation to wait for a connection at init time.

The examples also block after each timestep so you have time to request multiple plots.

Use VisIt <https://wci.llnl.gov/codes/visit>



The Simulation's window shows meta-data about the running code

Control commands exposed by the code are available here

All of VisIt's existing functionality is accessible

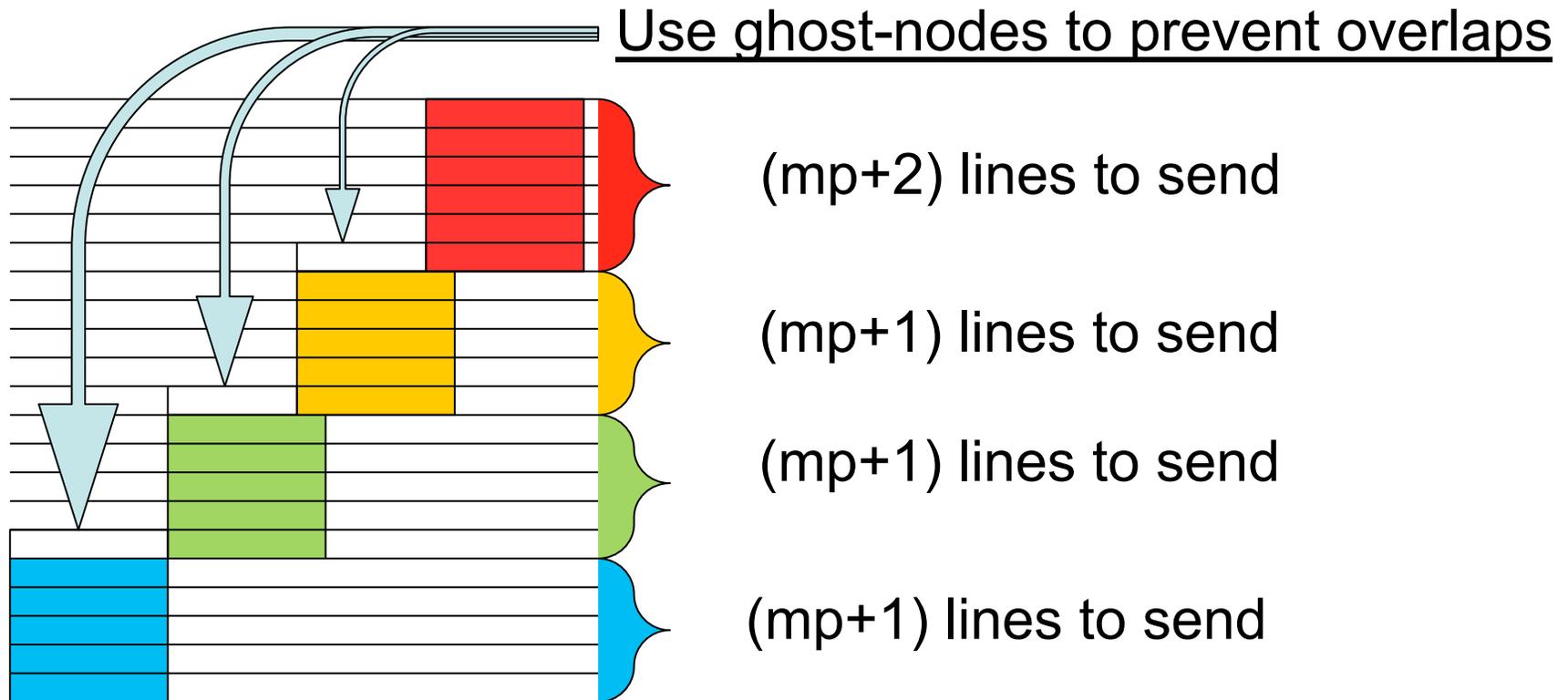
Users select simulations to open as if they were files

Data sharing

- The VisIt Data API has just a few callbacks
 - GetMetaData()
 - GetMesh()
 - GetScalar(), GetVector()

 - Each MPI rank provides the full mesh/data (with ghost regions) marked in a way similar to HDF5 hyperslabs or `MPI_Type_create_subarray()`.

grid mesh for *in situ* graphics

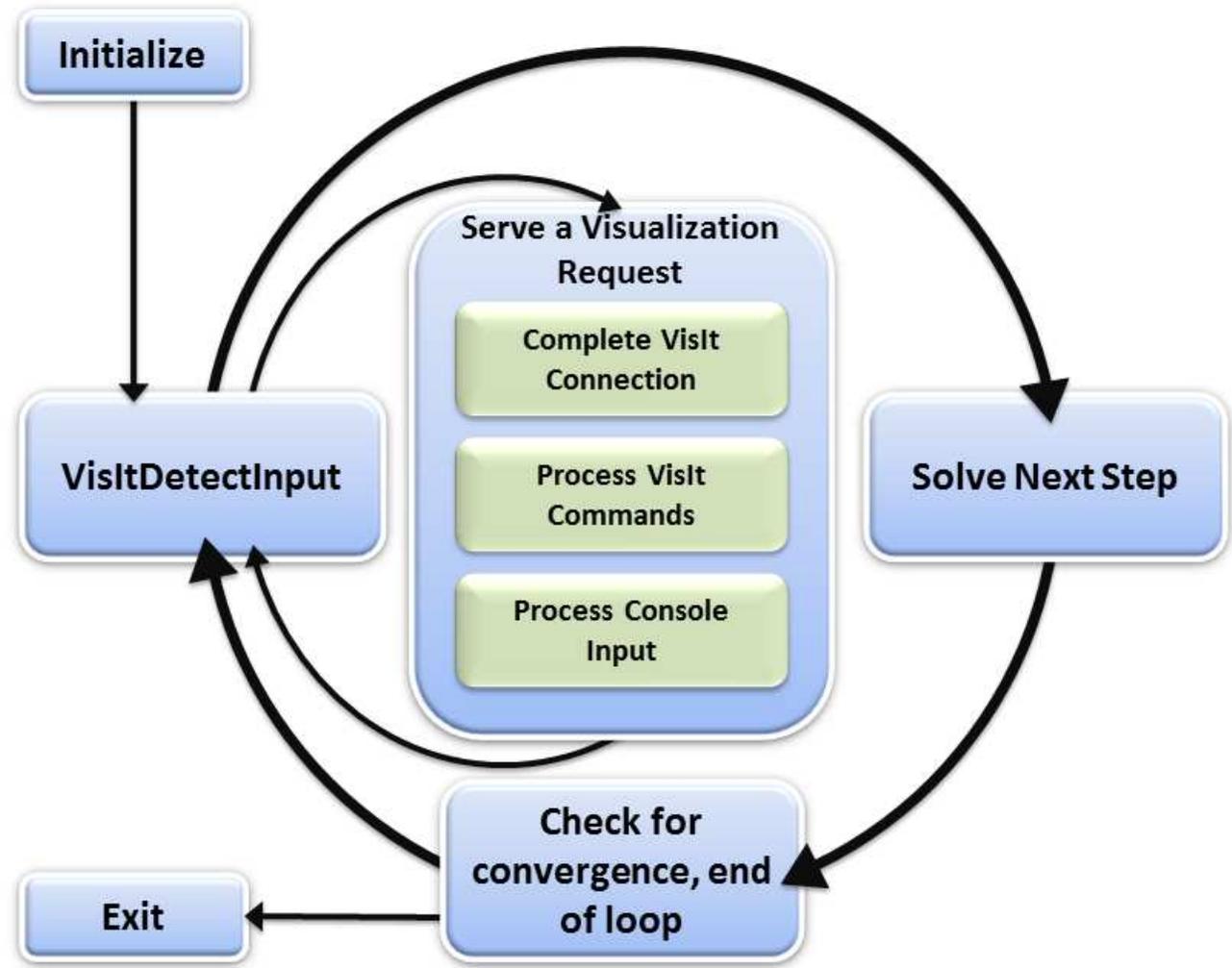


VisitRectMeshSetRealIndices(h , minRealIndex, maxRealIndex)

At least two entry points to the execution

Execution of the next step can be triggered by:

- normal program flow,
- on-demand single-stepping from the GUI,
- console input.



How much impact in the source code?

The **best suited** simulations are those allocating large (contiguous) memory arrays to store mesh coordinates, connectivity, and variables.

Memory pointers are used, and the simulation (or the visualization) can be assigned the responsibility to de-allocate the memory when done.

F90 example:

```
allocate ( v(0:m+1,0:mp+1) )
```

```
visitvardatasetd(h, VISIT_OWNER_SIM, 1, (m+2)*(mp+2), v)
```

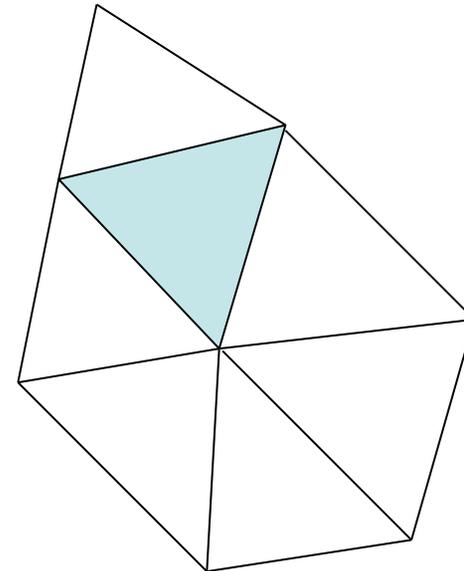
How much impact in the source code?

The **least suited** are those pushing the Object Oriented philosophy to a maximum.

Example: Finite Element code handling a triangular mesh:

```
TYPE Element
  REAL(r8) :: x(3)
  REAL(r8) :: y(3)

  REAL(r8) :: h
  REAL(r8) :: u
  REAL(r8) :: zb(3)
END TYPE Element
```



How much impact in the source code?

When data points are spread across many objects, there must be a new memory allocation and a gathering done before passing the data to the Vis Engine

```

REAL, DIMENSION(:), ALLOCATABLE :: cx

ALLOCATE( cx(numNodes) , stat=ierr)

DO iElem = 1, numElems+numHalos
  DO i = 1, 3
    cx(ElementList(iElem)%lclNodeIDs(i)) = ElementList(iElem)%x(i)
  END DO
END DO

err = visitvardatasetf(x, VISIT_OWNER_COPY, 1, numNodes, cx)

```

The *in-situ* library provides many features

- Access to scalar, vector, tensor arrays, and label
- CSG meshes, multi-block meshes, AMR meshes
- Polyhedra
- Material species
- Ability to save images directly from the simulation
- Interleaved XY, XYZ coordinate arrays

- Connecting in-situ does not mean you cannot do I/O to files anymore.

The merits of libsim

- The greatest bottleneck (disk I/O) can be eliminated
- Not restricted by limitations of any file format
- No need to reconstruct ghost-cells from archived data
- All time steps are potentially accessible
- All problem variables can be visualized
- Internal data arrays can be exposed or used
- Step-by-step execution will help you debug your code and your communication patterns

- The simulation can watch for a particular event and trigger the update of the VisIt plots

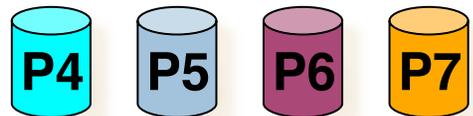
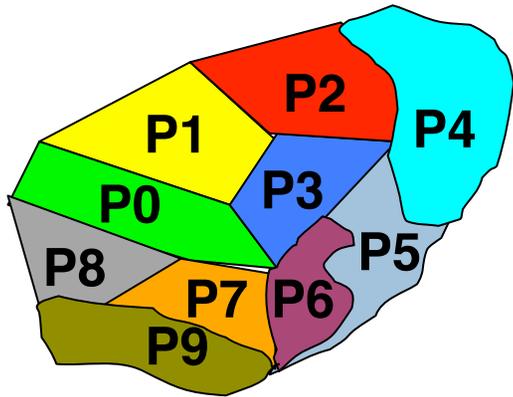
Outline



- Multi-resolution processing
 - Space filling curves (from Valerio Pascucci)
 - Wavelet compression (from John Clyne)
- In situ processing
 - System overview (from Brad Whitlock)
 - Example in action (from Jean Favre)
- **Query-driven visualization**
 - Overview (from Wes Bethel)
 - FastBit (from John Wu)
 - Example in action (from Oliver Ruebel)

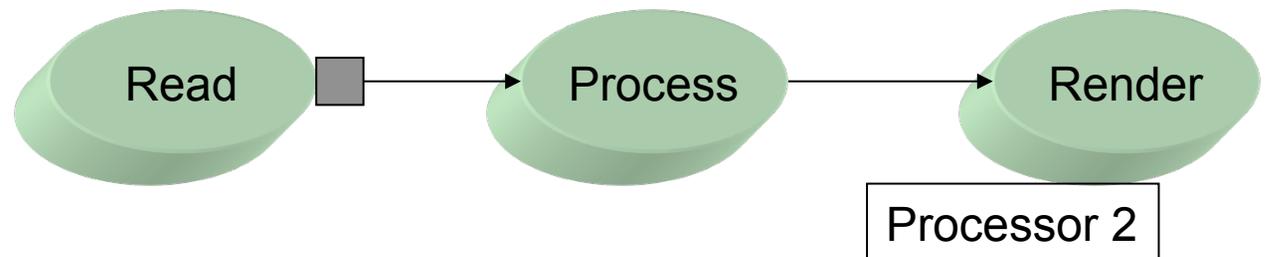
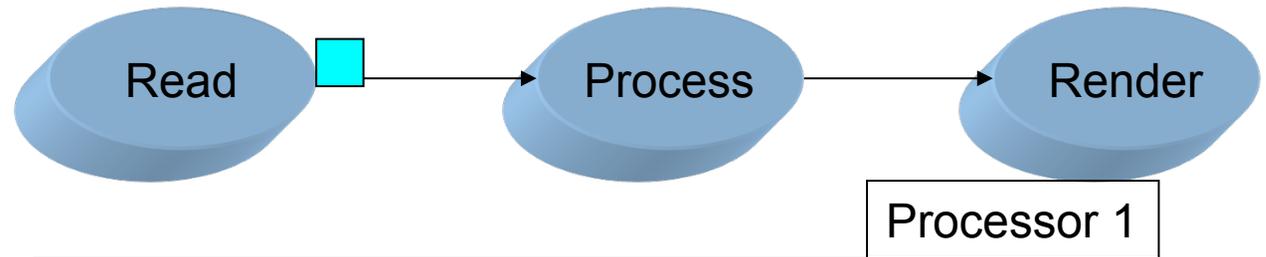
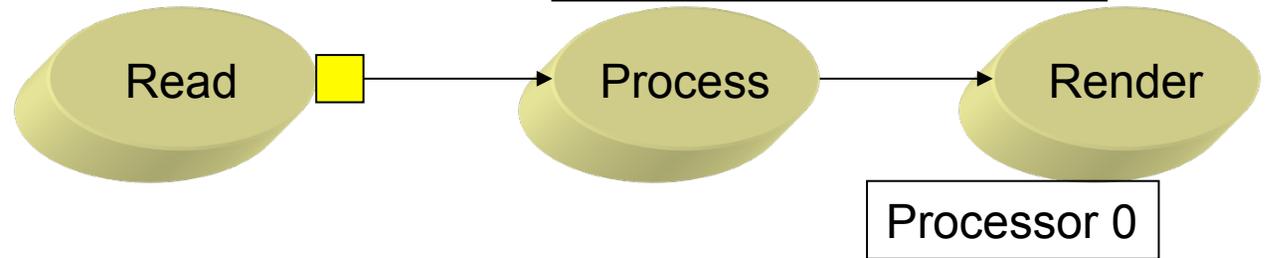
Data subsetting eliminates pieces that don't contribute to the final picture.

Parallel Simulation Code



Pieces of data
(on disk)

Parallelized visualization
data flow network



Data Subsetting: pros and cons



- Pros:

- Less data to process (less I/O, less memory)

- Cons:

- Only applicable to some algorithms

Outline



- Multi-resolution processing
 - Space filling curves (from Valerio Pascucci)
 - Wavelet compression (from John Clyne)
- In situ processing
 - System overview (from Brad Whitlock)
 - Example in action (from Jean Favre)
- Query-driven visualization
 - Overview (from Wes Bethel)
 - FastBit (from John Wu)
 - Example in action (from Oliver Ruebel)

Query-Driven Visualization



*Kurt Stockinger, Kesheng (John) Wu,
John Shalf, and Wes Bethel*

Computational Research Division
Lawrence Berkeley National Laboratory
November 2005



Motivation and Problem Statement



- Too much data.
- Visualization “meat grinders” not especially responsive to needs of scientific research community.
- What scientific users want:
 - Scientific Insight
 - Quantitative results
 - Feature detection, tracking, characterization
 - (lots of bullets here omitted)
- See:
 - <http://vis.lbl.gov/Publications/2002/VisGreenFindings-LBNL-51699.pdf>
 - <http://www-user.slac.stanford.edu/rmount/dm-workshop-04/Final-report.pdf>





Motivation and Problem Statement



- Too much data.
- Visualization “meat grinders” not especially responsive to needs of scientific research community.
- What scientific users want:
 - Scientific Insight
 - Quantitative results
 - Feature detection, tracking, characterization
 - (lots of bullets here omitted)
- See:
 - <http://vis.lbl.gov/Publications/2002/VisGreenFindings-LBNL-51699.pdf>
 - <http://www-user.slac.stanford.edu/rmount/dm-workshop-04/Final-report.pdf>





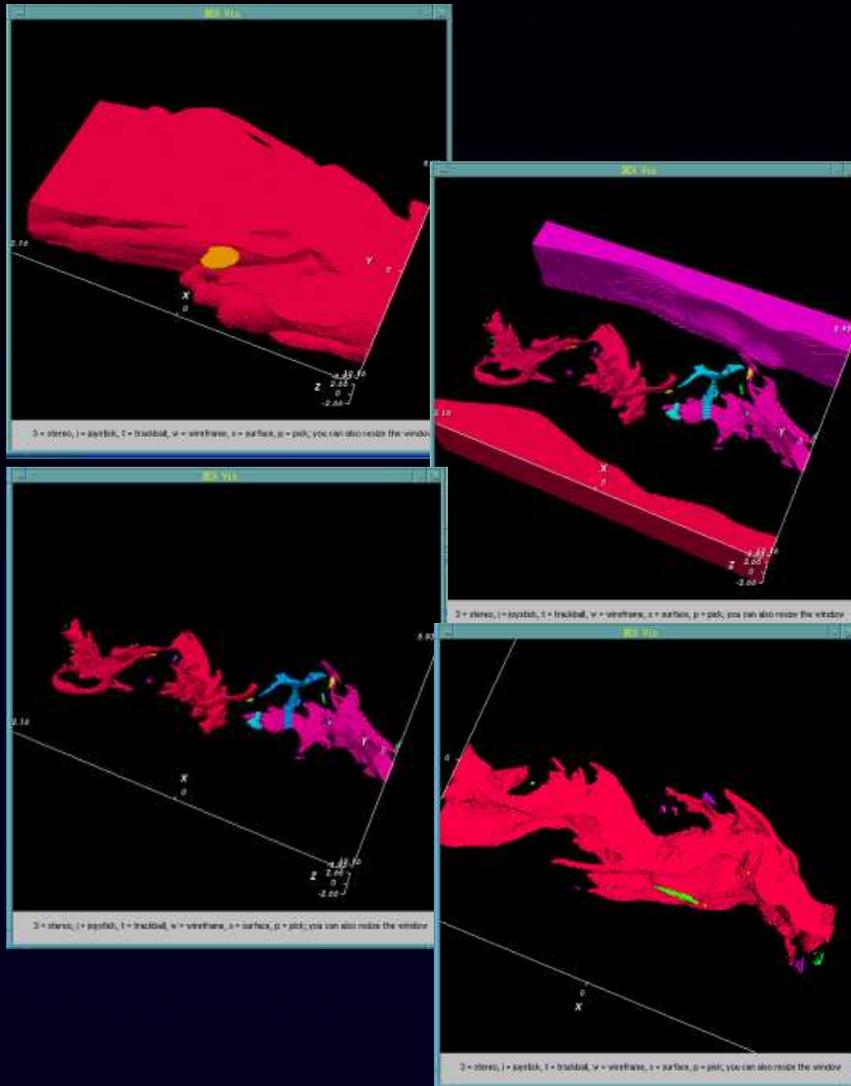
Today's Main Message

- Visualization stands to benefit in a huge way by leveraging technology from the field of scientific data management.
- An introduction to compressed bitmap indexing using reference points familiar to the visualization community.
- Compressed bitmap indexing:
 - Has low storage overhead.
 - Has low computational complexity (theoretically optimal).
 - Accommodates n -dimensional queries.
- Topics for another day:
 - Assisted/guided query posing.
 - Effective visualization of n -dimensional data.





Query-Driven Visualization: Visual Example



➤ $\text{CH}_4 > 0.3$

➤ $\text{Temp} < T_1$

➤ $\text{CH}_4 > 0.3$ AND $\text{temp} < T_1$

➤ $\text{CH}_4 > 0.3$ AND $\text{temp} < T_2$
• $T_1 < T_2$



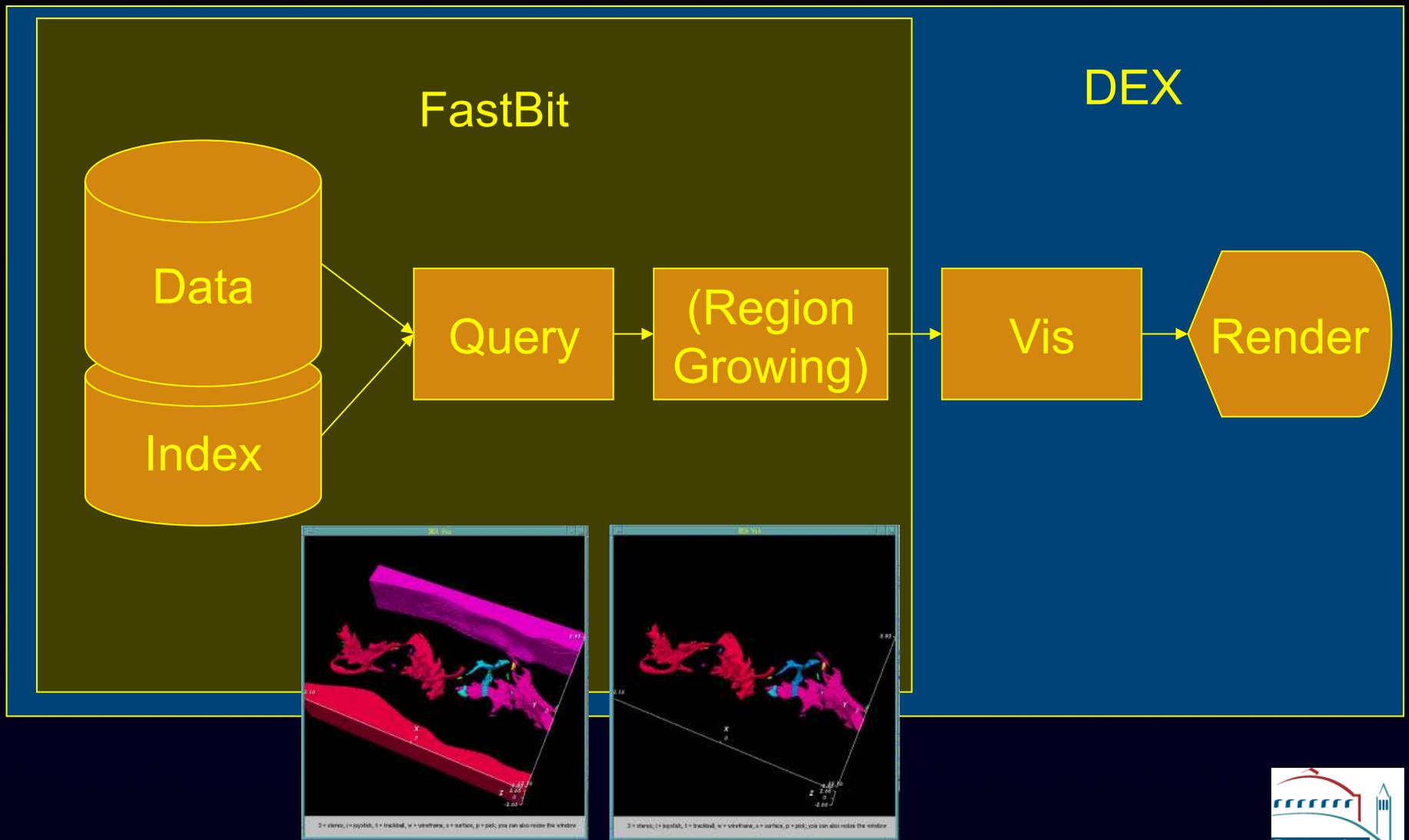


Architecture Overview: Generic Vis Pipeline





Architecture Overview: Query-Driven Pipeline





What is Query-Driven Visualization?

- Focus visualization processing on subsets of data deemed to be “interesting.”
 - “Interesting” is something the user needs to define.
- Challenges
 - How to define “interesting.”
 - Formulation of definition (domain-specific).
 - Expression of definition (semantic).
 - Find interesting data quickly (data management).
 - Effective visual presentation of “interesting data” (visualization).
 - Architectures/deployment that complements existing visualization algorithms and applications (computer science).





Value of Multi-dimensional Queries

- New opportunities for scientific insight: N -dimensional queries are the basis for complex analysis and hypothesis testing.
 - What are the characteristics of a flame front?
 - How are two (or n) Supernovae explosions similar/different?
 - Will this vaccine work against the Bird Flu?
 - Temporal-based queries and analysis.
- Reducing processing and interpretation load.
 - 100TB datasets being queued up now.
 - Increased spatial resolution.
 - Lots more variables per cell.
 - Can't expect a user to visually process 100TB of data.





Finding Data Quickly: Why Bitmap Indices

- In the data management community, the bitmap indices have supplanted trees for “heavy lifting” queries.
- Bitmap indices do not suffer from curse of dimensionality.
- Bitmap indices used in all major commercial database systems.
- Caveat: Bitmap indexing is not the panacea for everything:
 - Spatial vs. Data-value partitioning: visibility culling.



What is a Bitmap Index?

Data values	b_0	b_1	b_2	b_3	b_4	b_5
0	1	0	0	0	0	0
1	0	1	0	0	0	0
5	0	0	0	0	0	1
3	0	0	0	1	0	0
1	0	1	0	0	0	0
2	0	0	1	0	0	0
0	1	0	0	0	0	0
4	0	0	0	0	1	0
1	0	1	0	0	0	0

- Compact: one bit per distinct value per object.
- Easy and fast to build: $O(n)$ vs. $O(n \log n)$ for trees.
- Efficient to query: use bitwise logical operations.
($0.0 < \text{H}_2\text{O} < 0.1$) AND ($1000 < \text{temp} < 2000$)
- Efficient for multidimensional queries.
 - No “curse of dimensionality”
- What about floating-point data?
 - Binning strategies.



Bitmap Index Query Complexity and Space Requirements

- How Fast are Queries Answered?
 - Let N denote the **number of objects** and H denote the **number of hits** of a condition.
 - Using **uncompressed** bitmap indices, search time is $O(N)$
 - With a good **compression** scheme, the search time is $O(H)$ – the theoretical **optimum**.
- How Big are the Indices?
 - In the worst case (completely random data), the bitmap index requires about 2x in data size (typically 0.3x).
 - In contrast, 4x space requirement not uncommon for tree-based methods.
 - Curse of dimensionality: for N points in D dimensions:
 - Bitmap index size: $O(N \cdot D)$
 - Tree-based method: $O(N^{**}D)$

Outline



- Multi-resolution processing
 - Space filling curves (from Valerio Pascucci)
 - Wavelet compression (from John Clyne)
- In situ processing
 - System overview (from Brad Whitlock)
 - Example in action (from Jean Favre)
- Query-driven visualization
 - Overview (from Wes Bethel)
 - **FastBit (from John Wu)**
 - Example in action (from Oliver Ruebel)



FASTBIT TUTORIAL

Outline

- Overview
- Basic functions
- Top-level interface
- Application example

John Wu

Scientific Data Management
Berkeley Lab

<http://sdm.lbl.gov/fastbit>

What FastBit Is Not

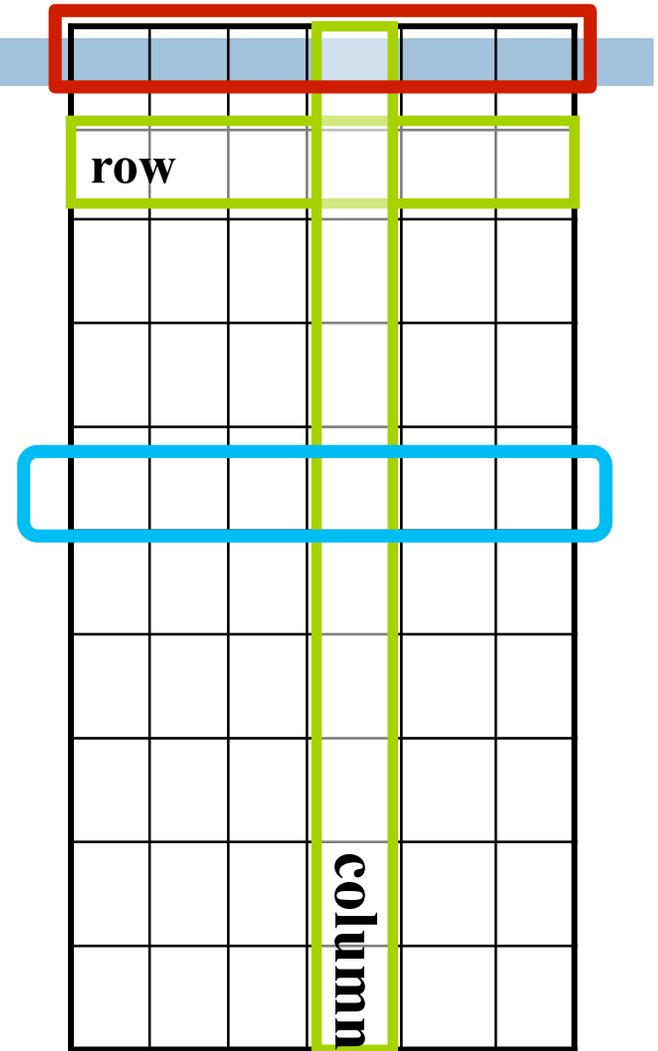
- ✘ Not a database management system (DBMS)
 - ▣ It is much closer to BigTable than to ORACLE
 - ▣ Most SQL commands are not supported
- ✘ Not a plug-in for a DBMS
 - ▣ It is a stand-alone data processing tool
 - ▣ No DBMS is needed in order to use FastBit
- ✘ Not an internet search engine
 - ▣ FastBit is primarily for structured data; internet search engines are for text (unstructured) data
- ✘ Not a client-server system
 - ▣ FastBit has been used in server programs, but by itself, it is not a client-server system

How Do I Use FastBit

- Command-line tools
 - ▣ A handful of command-line tools are available to load data, build indexes, and query data
- Write your own program using FastBit as a library
 - ▣ Two levels of API:
 - Class table
 - Class part + query
 - ▣ FastBit is written in C++
 - Other languages may access FastBit through C API

FastBit Data Model

- FastBit is designed to search multi-dimensional append-only data
 - Conceptually in table format
 - rows → objects
 - columns → attributes
- FastBit uses vertical (column-oriented) data organization
 - Efficient for searching
- Physical data layout
 - A data table is split into “partitions”
 - Each partition is a directory in a file system
 - Each directory has a metadata file describing the data partition
 - Each column is represented by a file



Basic Bitmap Index

<i>Data values</i>	b_0 =0	b_1 =1	b_2 =2	b_3 =3	b_4 =4	b_5 =5
0	1	0	0	0	0	0
1	0	1	0	0	0	0
5	0	0	0	0	0	1
3	0	0	0	1	0	0
1	0	1	0	0	0	0
2	0	0	1	0	0	0
0	1	0	0	0	0	0
4	0	0	0	0	1	0
1	0	1	0	0	0	0

$\swarrow \quad \searrow$ $A < 2$ $\swarrow \quad \searrow$ $2 < A$

- First commercial version
 - Model 204, P. O'Neil, 1987
- Easy to build: faster than building B-trees
- Efficient for querying: only bitwise logical operations
 - $A < 2 \rightarrow b_0 \text{ OR } b_1$
 - $A > 2 \rightarrow b_3 \text{ OR } b_4 \text{ OR } b_5$
- Efficient for multi-dimensional queries
 - Use bitwise operations to combine the partial results
- Size: one bit per distinct value per row
 - Definition: **Cardinality** == number of distinct values
 - Compact for low cardinality attributes, say, cardinality < 100
 - Worst case: cardinality = N, number of rows; index size: $N*N$ bits

Strategies to Improve Bitmap Index

□ Compression

- Reduce the size of each individual bitmap
- Best known compression method: Byte-aligned Bitmap Code [Antoshenkov 1994], used in Oracle bitmap index
- **Word-Aligned Hybrid (WAH)** code trades some disk space for much more efficient query processing

□ Encoding

- Basic equality encoding, in Model 204
- Multi-component encoding [[Chan and Ioannidis 1998](#)]
- **Multi-level encoding**

□ Binning

- Equal-width binning, equal-depth binning, ...
- Has to perform candidate check to rule out false positives, time for candidate check dominates the total query response time
- **Order-preserving Bin-based Clustering (OrBiC)**

Outline



- Multi-resolution processing
 - Space filling curves (from Valerio Pascucci)
 - Wavelet compression (from John Clyne)
- In situ processing
 - System overview (from Brad Whitlock)
 - Example in action (from Jean Favre)
- Query-driven visualization
 - Overview (from Wes Bethel)
 - FastBit (from John Wu)
 - Example in action (from Oliver Ruebel)

High Performance Multivariate Visual Data Exploration for Extremely Large Data

Oliver Rübel^{1,2,3}, Prabhat¹, Kesheng Wu¹, Hank Childs⁴, Jeremy Meredith⁵,
Cameron G.R. Geddes⁶, Estelle Cormier-Michel⁶, Sean Ahern⁵, Gunther H. Weber^{1,2,3},
Peter Messmer⁷, Hans Hagen², Bernd Hamann^{3,2,1} and E. Wes Bethel^{1,3}

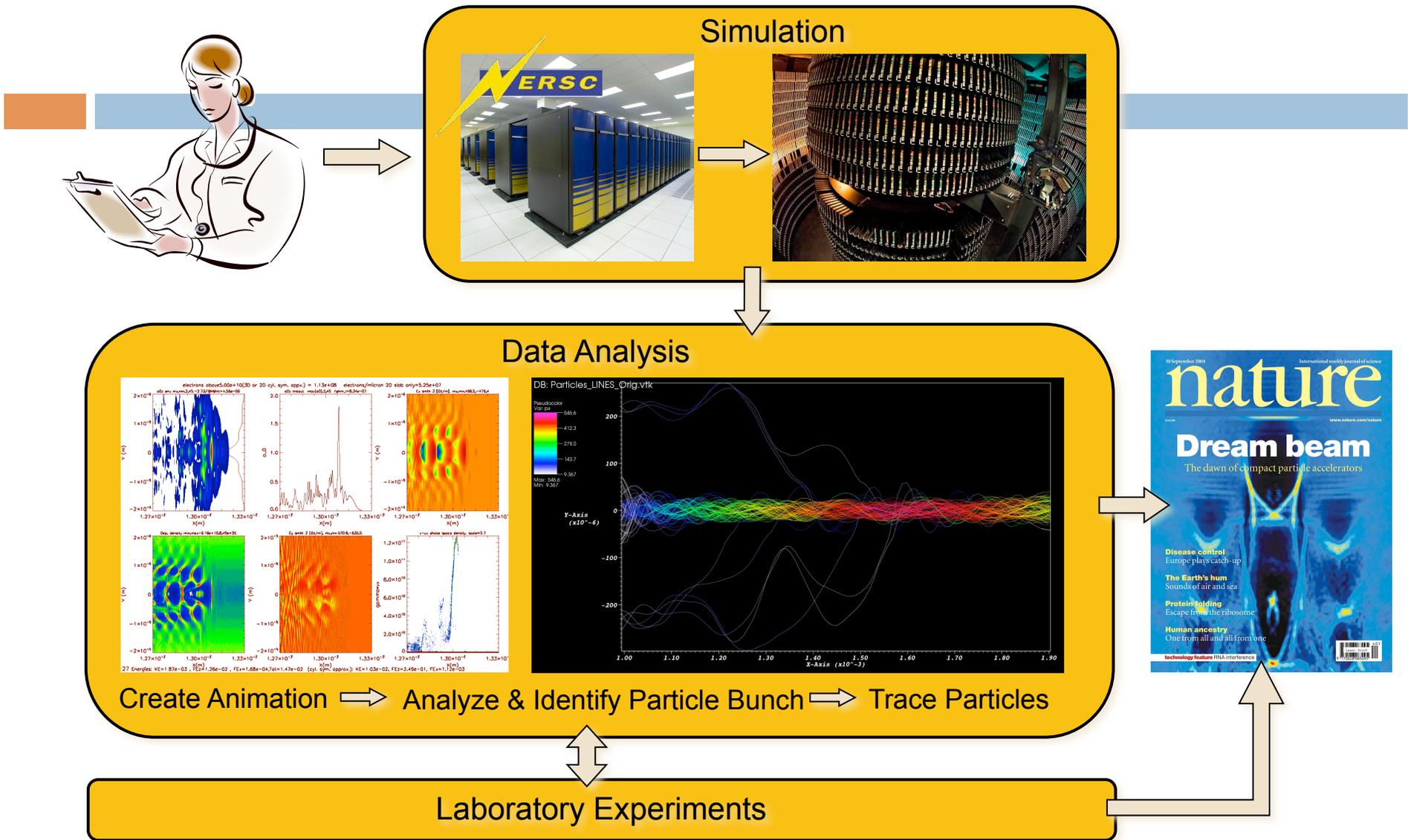
1. Computational Research Division, Lawrence Berkeley National Laboratory (LBNL)
2. International Research Training Group 1131, University of Kaiserslautern, Germany
3. Institute for Data Analysis and Visualization, University of California, Davis
4. Lawrence Livermore National Laboratory (LLNL)
5. Oak Ridge National Laboratory (ORNL)
6. LOASIS program of Lawrence Berkeley National Laboratory
7. Tech-X Corporation

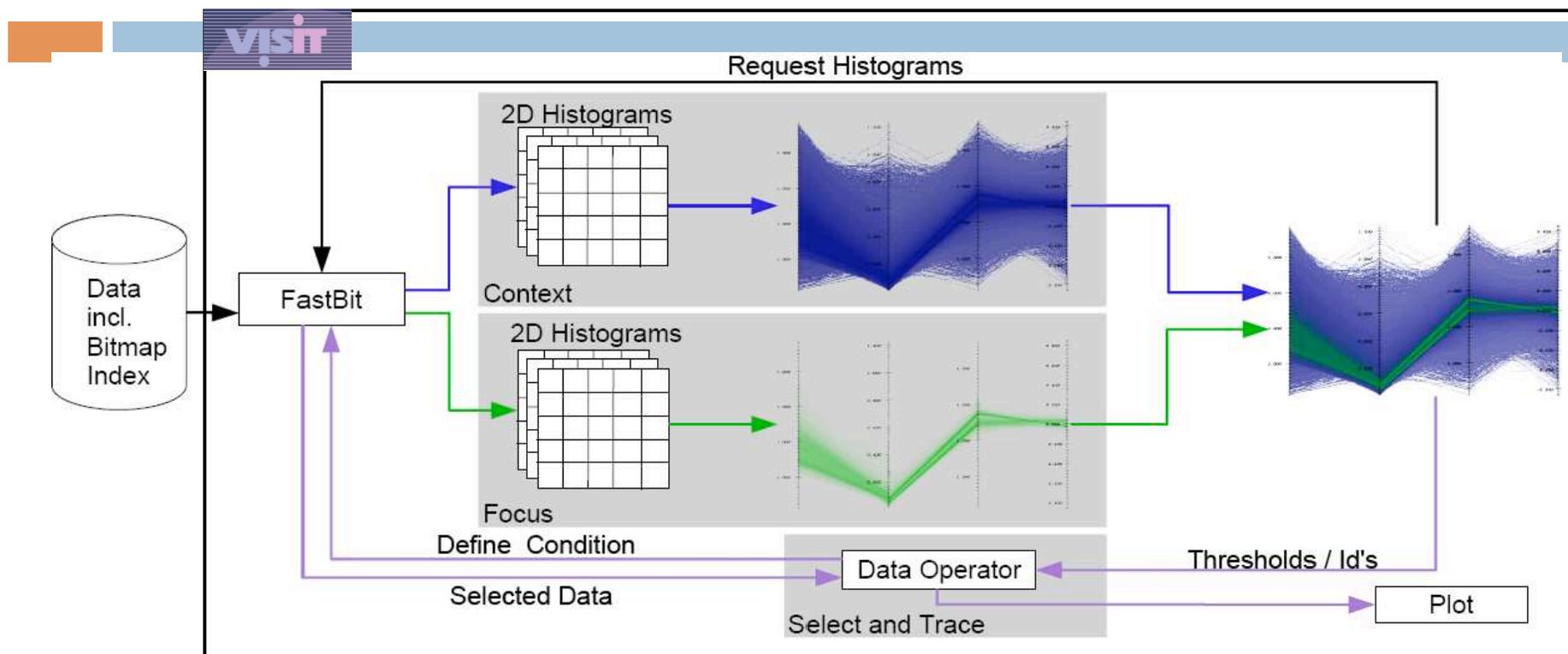


Overview:

- Enable rapid knowledge discovery from large, complex, multivariate, time-varying data
- Illustrate in a case study how our system can be used to effectively analyze laser wakefield particle acceleration data
- Analyze the performance of our system

Motivation





References:

- VisIt is available from <https://wci.llnl.gov/codes/visit/>
- FastBit is available from <https://codeforge.lbl.gov/projects/fastbit>

Why FastBit:

- Query response time is a linear function of the number of hits $O(H)$
 - Particle tracking: Naïve approach: $O(n^2)$
With FastBit: $O(H*t)$
 - Conditional histogram: Naïve approach: $O(n)$
FastBit: $O(H)$
- Linear spatial and computational complexity with respect to number of data dimension
- FastBit indices can be constructed fast
- FastBit indices are small compared to, e.g., B-trees [2 Fig.7]
- FastBit implements fastest known bitmap compression technique [2]
- R&D100 award [1]

We use FastBit to accelerate:

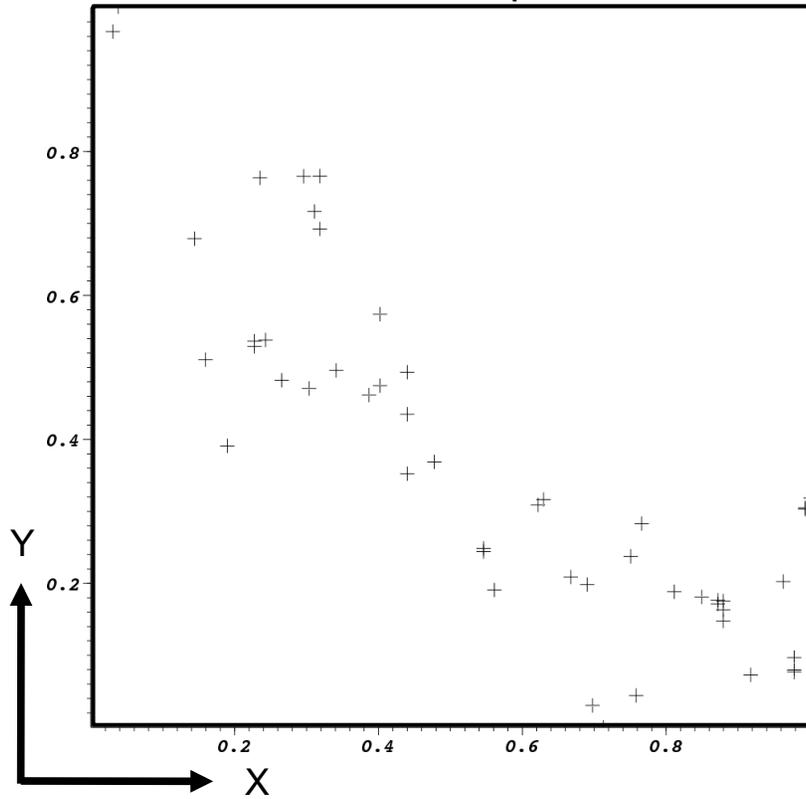
- Computation of conditional histograms used for rendering of parallel coordinates
- Multi-dimensional threshold queries used for identification of particles of interest
- ID-queries used for tracing of particles over time

References:

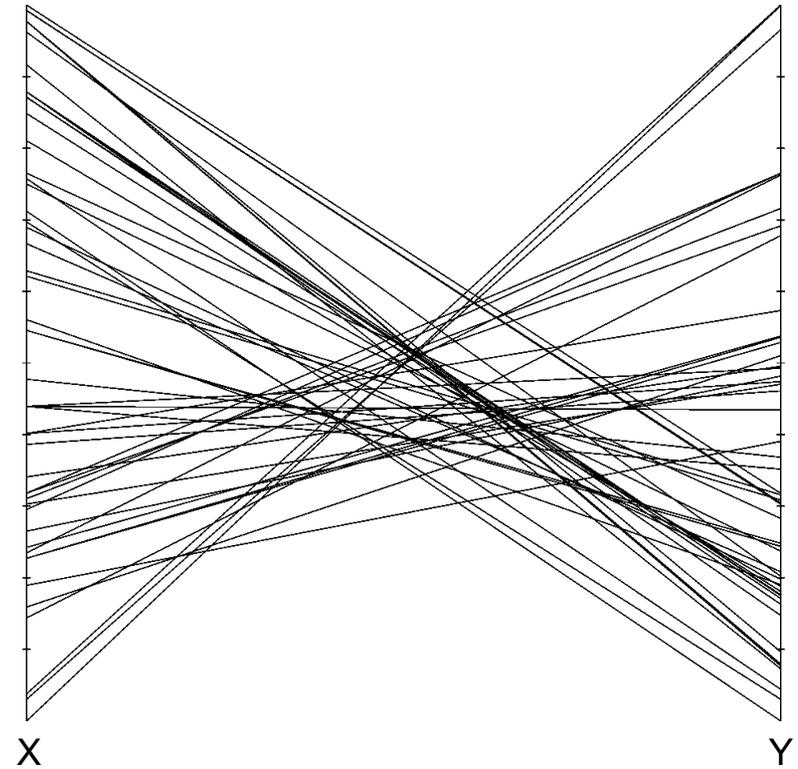
[1] FastBit is available from <https://codeforge.lbl.gov/projects/fastbit>

[2] K. Wu, E. Otoo, and A. Shoshani, "Compressing bitmap indexes for faster search operations", ACM Transactions on Database Systems, vol 31, pp. 1-38, 2006

2D Scatter-plot

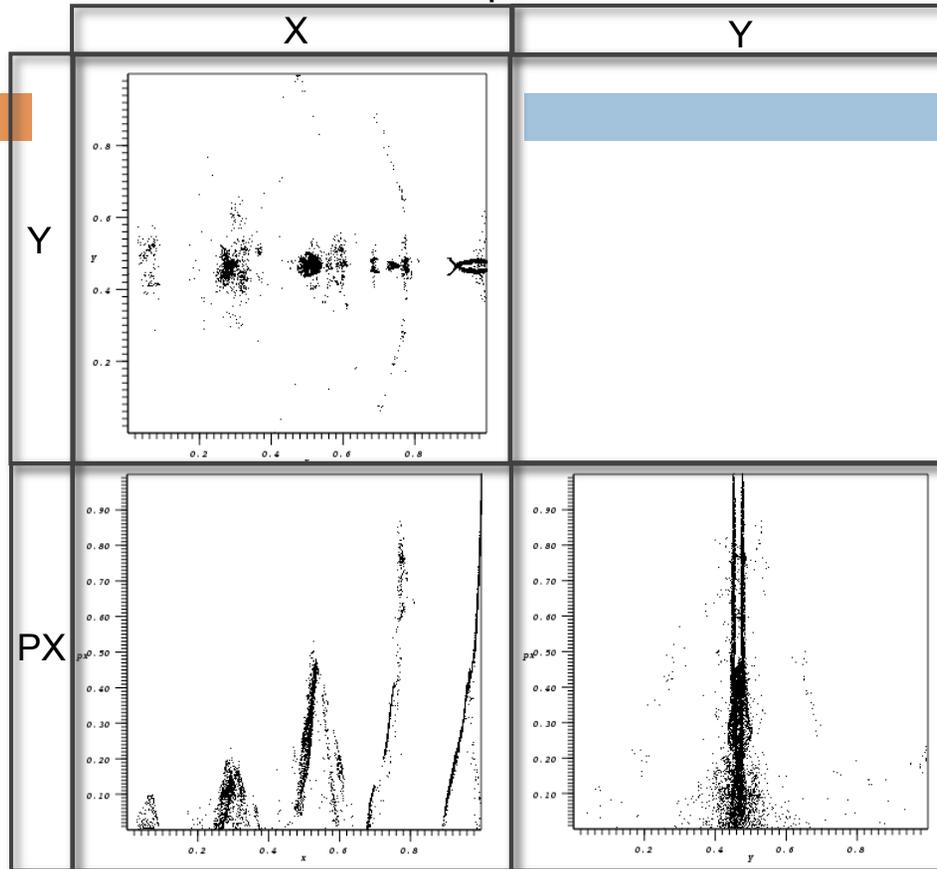


Parallel Coordinates

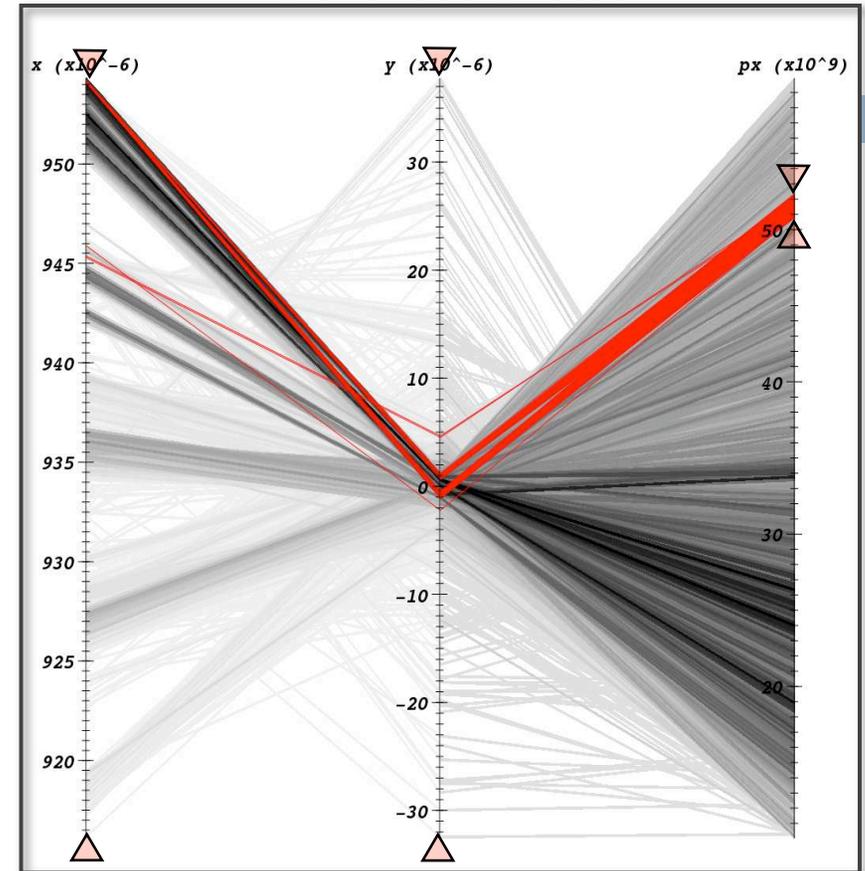


Introduction to Parallel Coordinates

Scatter-plot Matrix



Parallel Coordinates



Advantages:

- Enable display of many data dimensions in parallel
- Easy to use interface for defining data selections
- Immediate feedback is provided while performing data selection

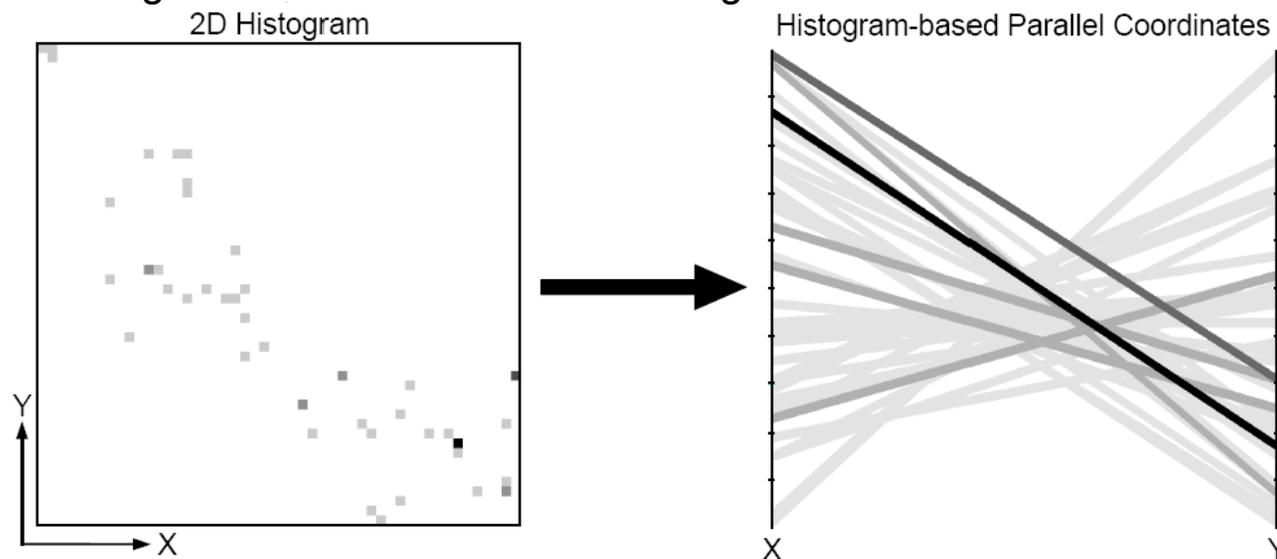
Histogram-based Parallel Coordinates

Disadvantages:

- Rendering and computational complexity is directly proportional to the size of the displayed data
- Clutter
- Occlusion
- Order dependent visualization

Solution:

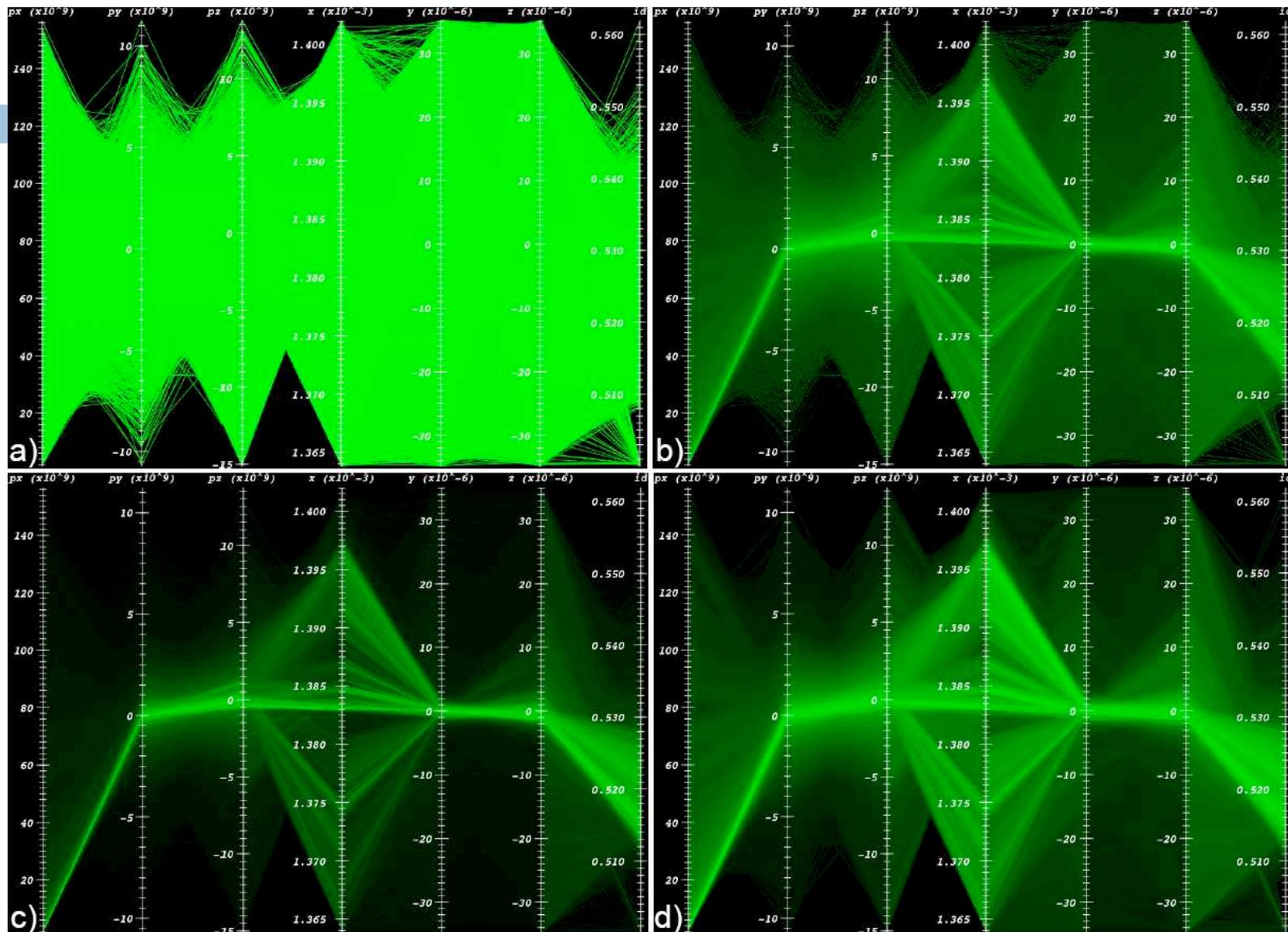
- Use 2D-histograms as basis for the rendering not the raw data



References:

- M. Novotny and H. Hauser, "Outlier-preserving focus+context visualization in parallel coordinates," *IEEE Transactions on Visualization and Computer Graphics*, vol. 12, no. 5, pp. 893-900. 2006.

Histogram-based Parallel Coordinates cont.

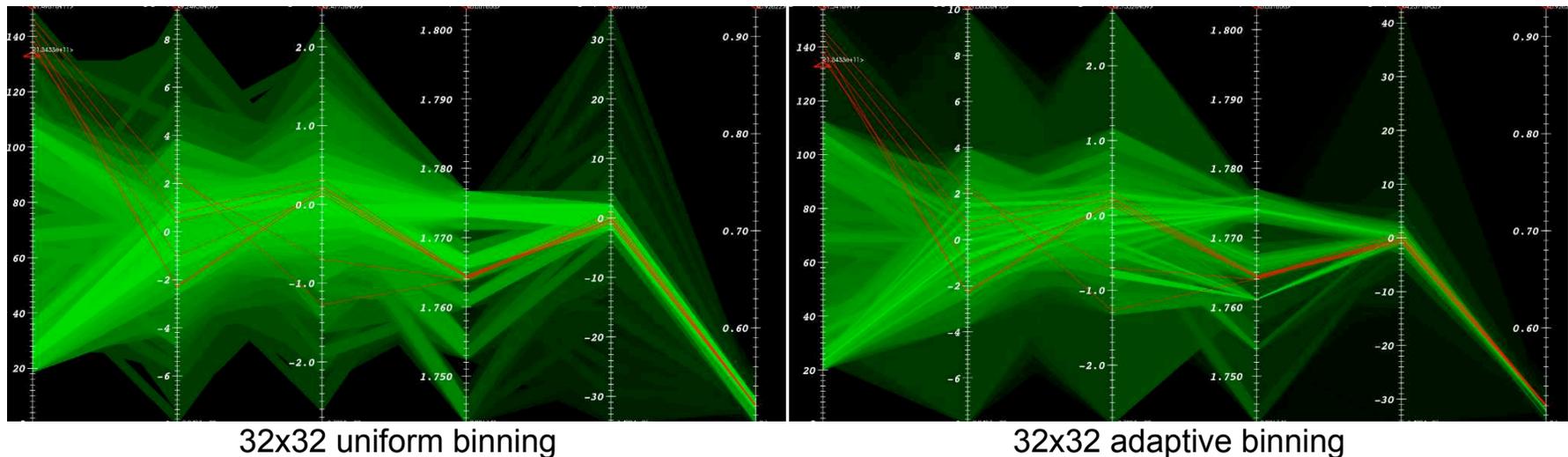


Histograms are computed on request:

- Enable rendering also of data subsets using histogram-based parallel coordinates
- Enable rendering with arbitrary number of bins
- Enable close zoom-ins and smooth drill-downs into the data

Allow use of adaptively binned histograms:

- Enable more accurate representation of the data in lower-level-of-detail views



Laser Wakefield Particle Acceleration



Advantages:

- Can achieve electric fields thousands of times stronger than in conventional accelerators
 - ➔ Can achieve high acceleration in very short distance.

References:

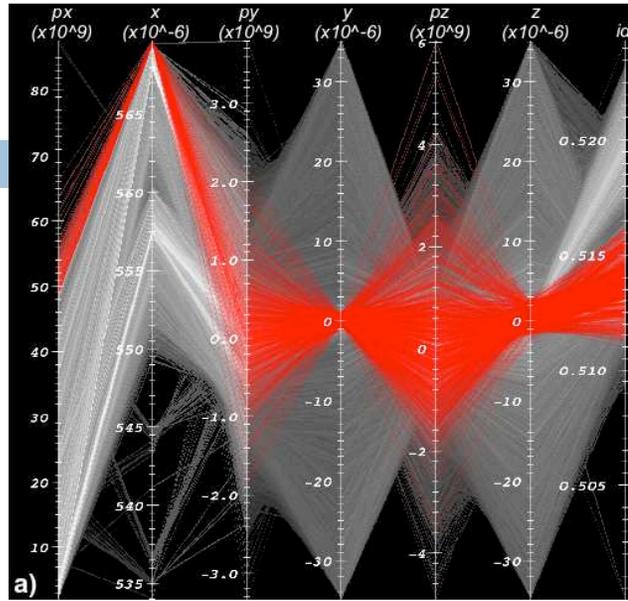
- C.G.R. Geddes, C. Toth, J. van Tilborg, E. Esarey, C. Schroeder, D. Bruhwiler, C. Nieter, J. Cary, and W. Leemans, "High-Quality Electron Beams from a Laser Wakefield Accelerator using Plasma-Channel Guiding," *Nature*, vol. 438, pp. 538-541, 2004

- **Simulation:** VORPAL, 2D and 3D
- **Particle data:**
 - Scattered data
 - x,y,z (location), px, py, pz (momentum), id (particle identifier)
 - No. of particles per timestep:
 - $\sim 0.4 \cdot 10^6 - 30 \cdot 10^6$ (in 2D)
 - $\sim 80 \cdot 10^6 - \mathbf{200 \cdot 10^6}$ (in 3D)
 - Total size:
 - $\sim 1.5\text{GB} - >30\text{GB}$ (in 2D)
 - $\sim 100\text{GB} - \mathbf{>1\text{TB}}$ (in 3D)
- **Field data:**
 - Defined on regular grid
 - Electric field, magnetic field, and RhoJ
 - Resolution: Typically $\sim 0.02\text{-}0.03\mu\text{m}$ longitudinally, and $\sim 0.1\text{-}0.2\mu\text{m}$ transversely
 - Total size:
 - $\sim 3.5\text{GB} - >70\text{GB}$ (in 2D)
 - $\sim 200\text{GB} - \mathbf{>2\text{TB}}$ (in 3D)

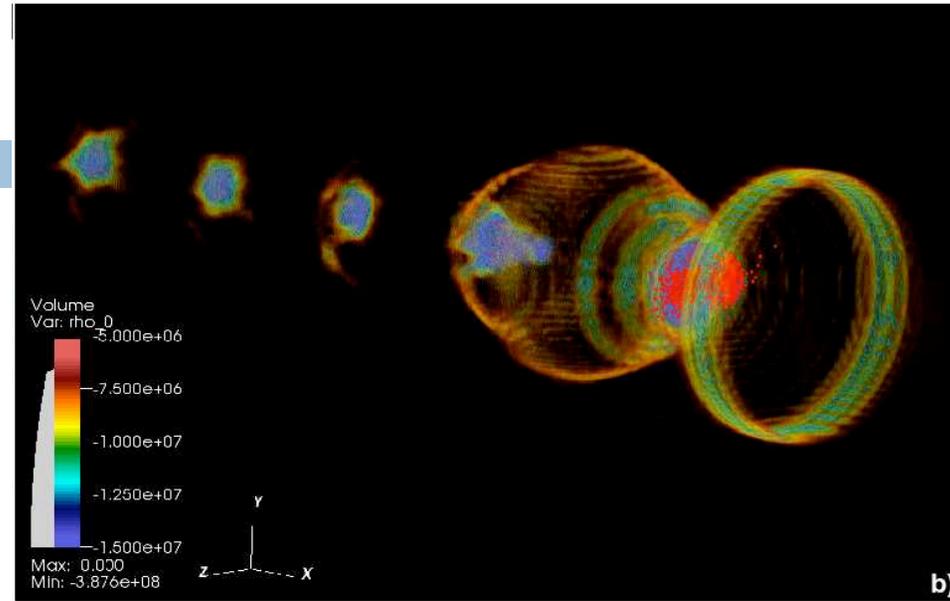
References:

- Cameron G.R. Geddes, "Plasma Channel Guided Laser Wakefield Accelerator," Phd-thesis, University of California, Berkeley, CA, 2005
- C. Nieter and J. R. Cary, "VORPAL: A Versatile Plasma Simulation Code," J. Comput. Phys., vol. 196, no. 2, pp. 448–473, 2004.

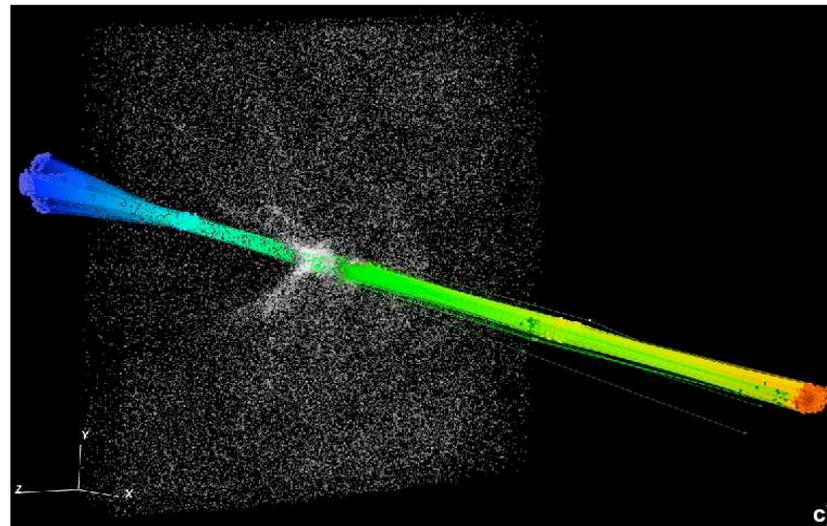
3D Analysis Example



a) Selecting particles of interest



b) Selected particles (red) and volume rendering of the plasma density



c) Traces of the the selected particle-bunch

Performance tests

□ **Serial Performance Tests:**

- Unconditional histograms:
 - Characterize the effect of varying the number of bins on the histogram computation
- Conditional histograms:
 - Characterize the effect of the number of hits on the computation of conditional histograms
- Particle Selection:
 - Characterize the performance of ID-queries

□ **Parallel Performance Tests:**

- Characterize scalability of:
 - Computation of histograms
 - Particle tracking

□ **Setup:**

- Compare FastBit-enabled application to Custom application performing a sequential scan

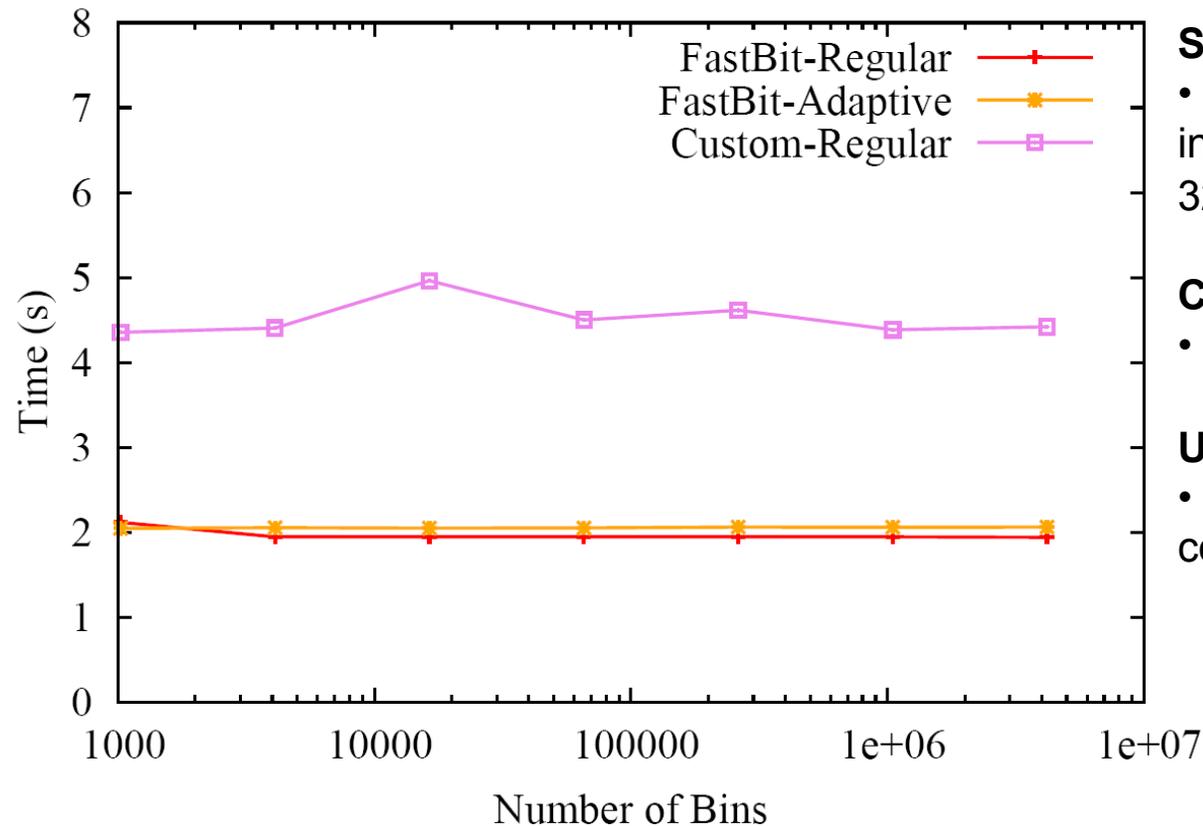
Serial Performance I: Unconditional Histograms

Dataset:

- 3D dataset consisting of 30 timesteps
- ~90 million particles per timestep
- ~7GB per timestep (including ~2GB for the index)
- ~210GB total size

Test platform:

- Workstation
- CPU: 2.2GHz AMD Opteron
- Memory: 4GB RAM
- OS: SuSE Linux



Setup:

- Test performance with increasing bin counts: 32x32 to 2048x2048

Custom:

- Perform sequential scan

Use Case:

- Initial computation of the context of parallel coordinates

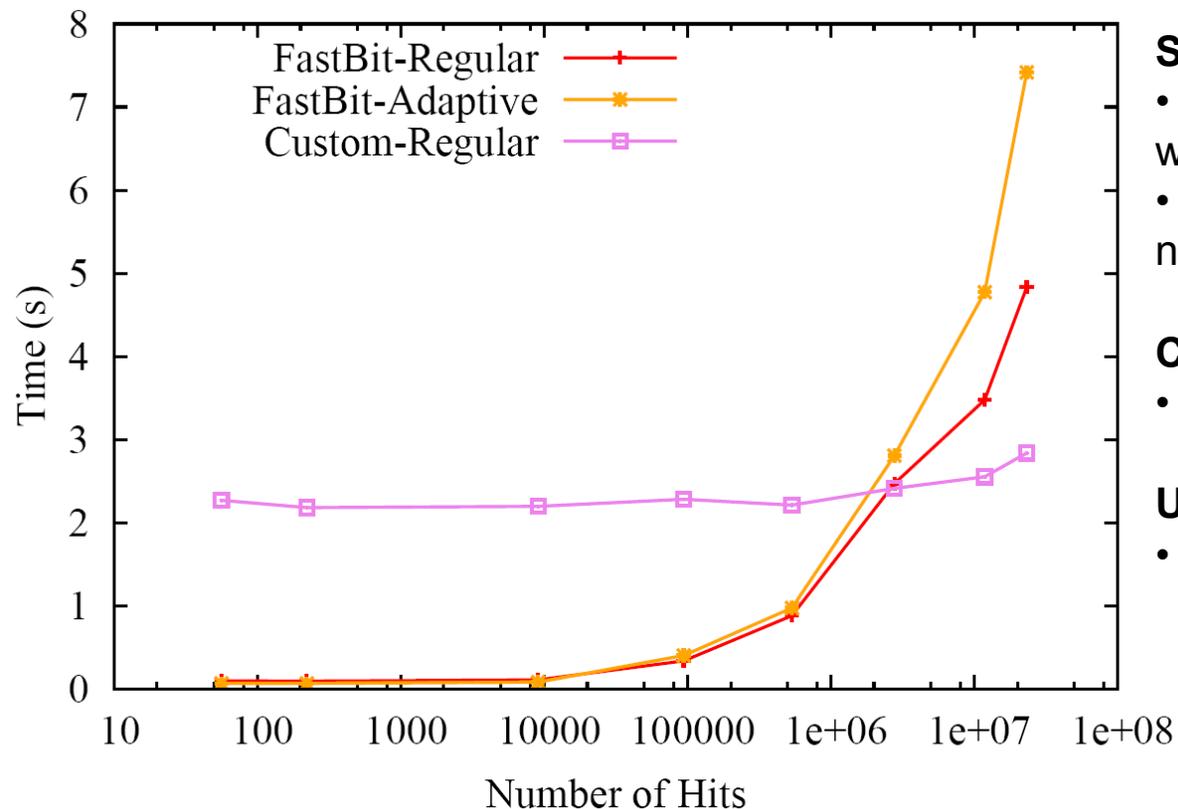
Serial Performance II: Conditional Histograms

Dataset:

- 3D dataset consisting of 30 timesteps
- ~90 million particles per timestep
- ~7GB per timestep (including ~2GB for the index)
- ~210GB total size

Test platform:

- Workstation
- CPU: 2.2GHz AMD Opteron
- Memory: 4GB RAM
- OS: SuSE Linux



Setup:

- Compute 1024x1024 histogram with varying condition ($px > \dots$)
- By increasing the threshold the number of hits decreases

Custom:

- Perform sequential scan

Use Case:

- Focus of parallel coordinates

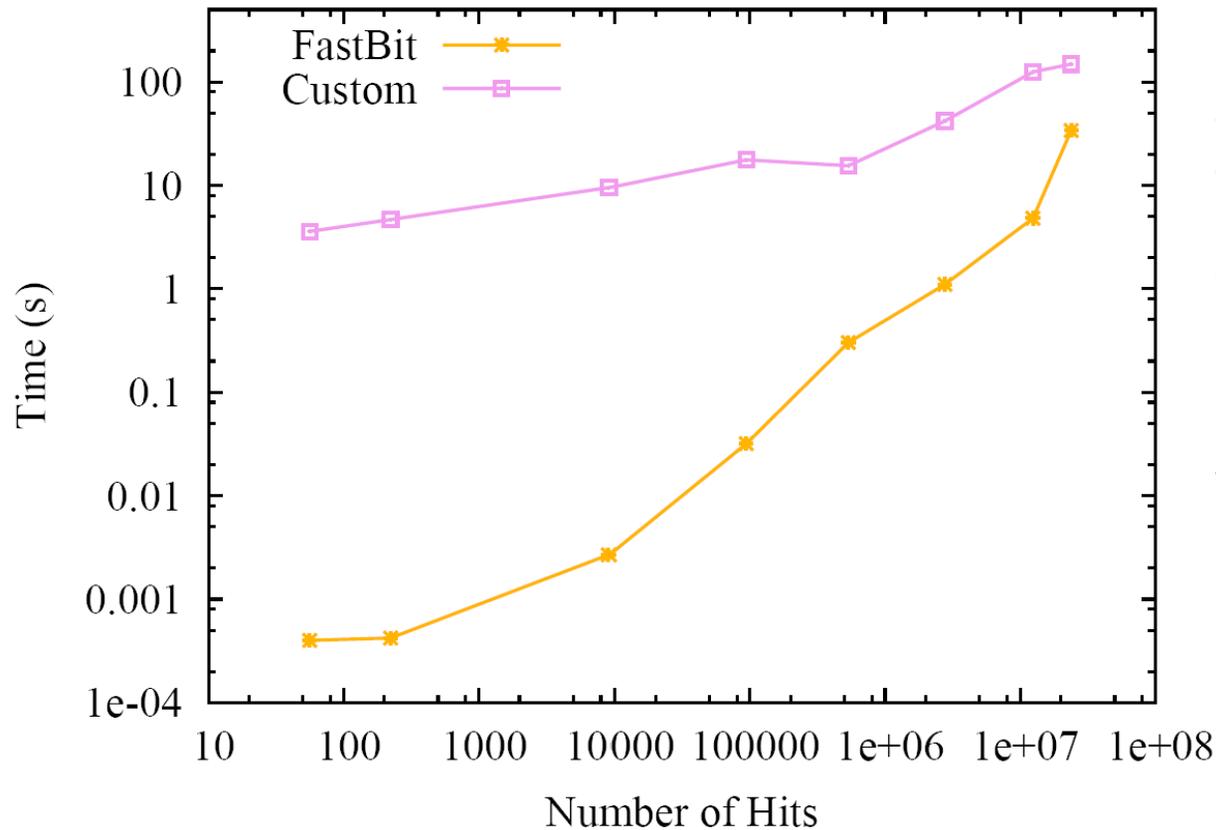
Serial Performance III: Particle Selection

Dataset:

- 3D dataset consisting of 30 timesteps
- ~90 million particles per timestep
- ~7GB per timestep (including ~2GB for the index)
- ~210GB total size

Test platform:

- Workstation
- CPU: 2.2GHz AMD Opteron
- Memory: 4GB RAM
- OS: SuSE Linux



Setup:

- Perform ID query at a single timestep and vary the size of the search set S

Custom:

- Compare particle ID of each data record to the search set
- Use efficient search algorithm with $O(\log(S))$ complexity

Use Case:

- Trace particle subset

Parallel Performance I: Histograms

Dataset:

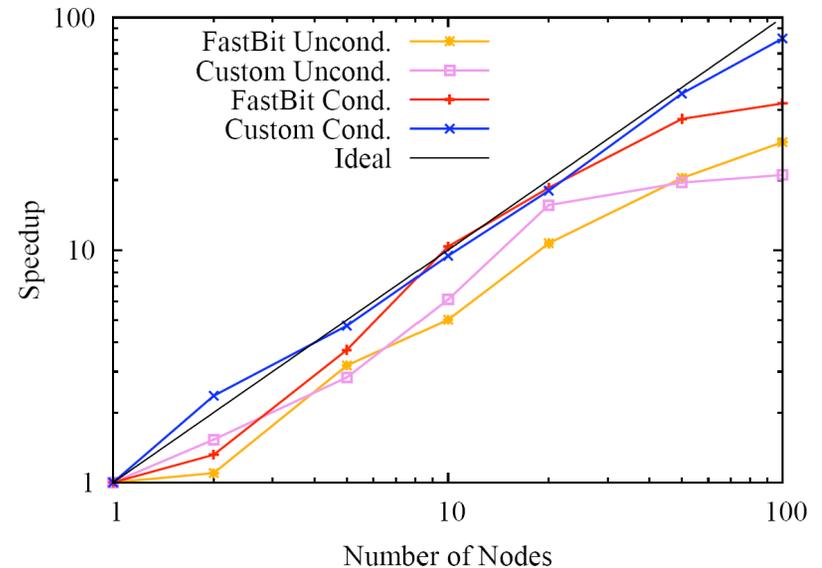
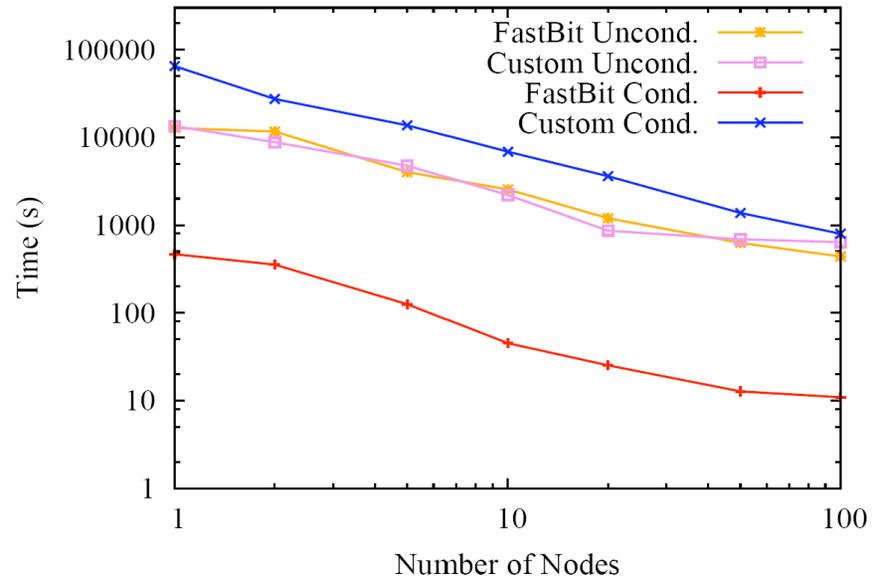
- 3D dataset consisting of 100 timesteps
- ~177 million particles per timestep
- ~10 GB per timestep
- ~1TB total size

Test platform: (as of July.2008)

- franklin.nersc.gov
- 9,660 nodes, 19K cores Cray XT4 system
- Filesystem: Lustre Parallel Filesystem
- Each node consists of:
 - CPU: 2.6 GHz, dual-core AMD Opteron
 - Memory: 4GB
 - OS: Compute Node Linux

Test setup:

- Restrict operations to a single core of each node to maximize I/O bandwidth available to each process
- Assign data subsets corresponding to individual timesteps to individual nodes for processing
- Generate five 1024x1024 histograms for position and momentum fields at each timestep
- Conditon: $px > 7 \cdot 10^{10}$
- Levels of parallelism: 1, 2, 5, 10, 20, 50, 100



Parallel Performance II: Particle Tracking

Test setup:

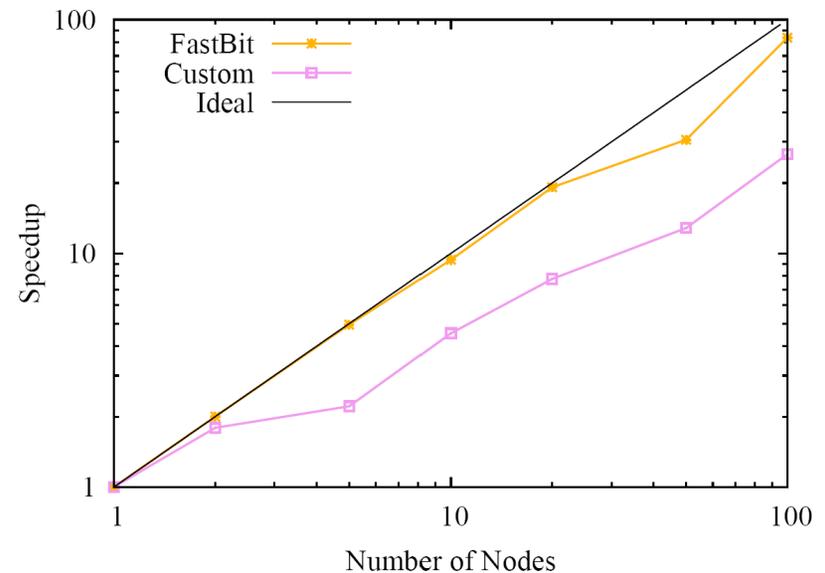
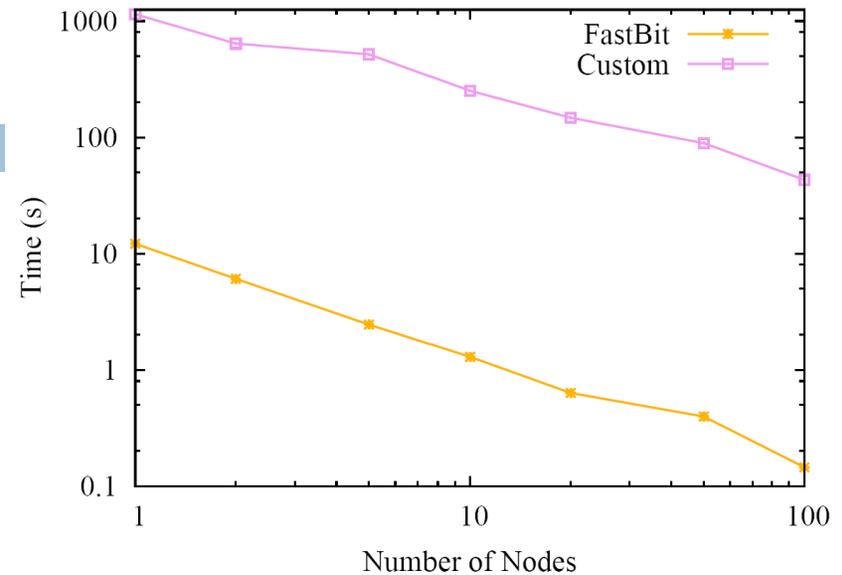
- Same as for histogram computation
- Track 500 particles (Condition: $px > 10^{11}$) over 100 timesteps

Results:

- FastBit is able to track 500 particles over 1.5TB of data in 0.15 seconds

Performance of original IDL scripts:

- ~2.5 hours to track 250 particles in small 5GB dataset

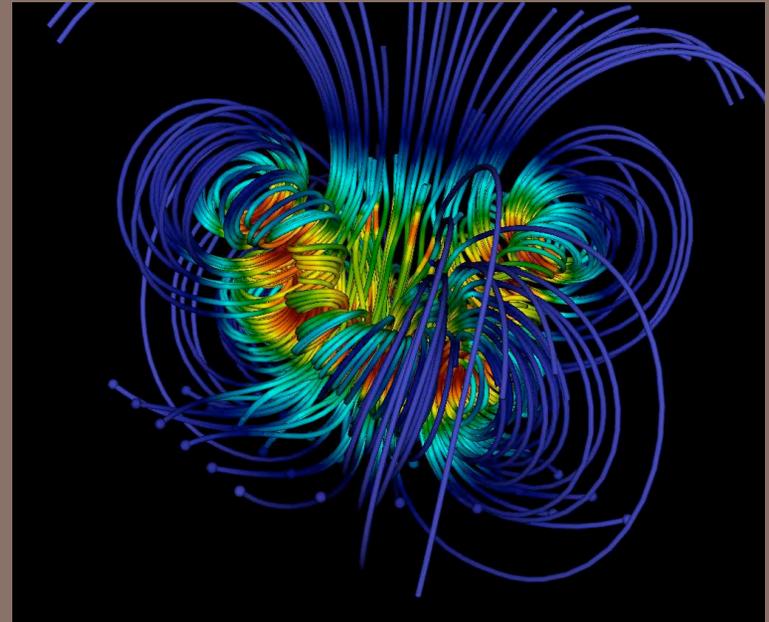


Summary



- Query-driven visualization allows for rapid exploration of data (in some scenarios).
- The FastBit library is an example technology for accelerating loading of subsets.
- These techniques fit well within the data flow framework.

Putting it all together: a visualization application for very large data



June 17, 2011

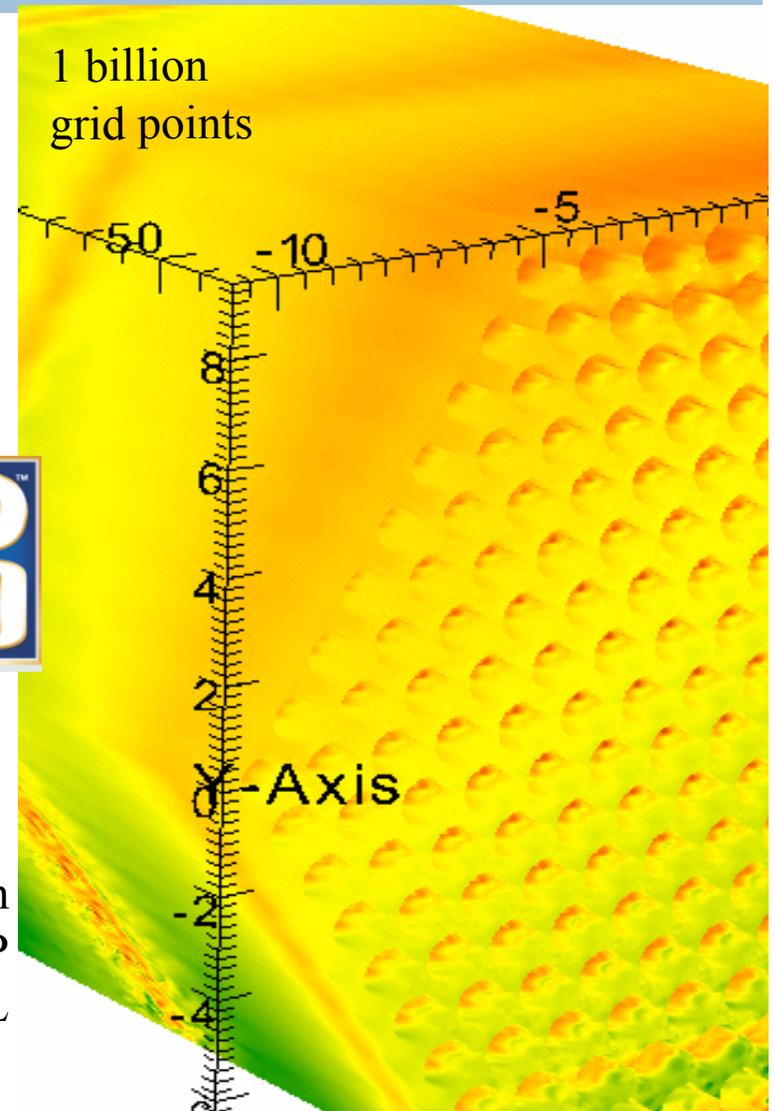
Hank Childs, Lawrence Berkeley Lab & UC Davis

VisIt is an open source, richly featured, turn-key application for large data.

- Used by:
 - Visualization experts
 - Simulation code developers
 - Simulation code consumers
- Popular
 - R&D 100 award in 2005
 - Used on many of the Top500
 - >>>100K downloads

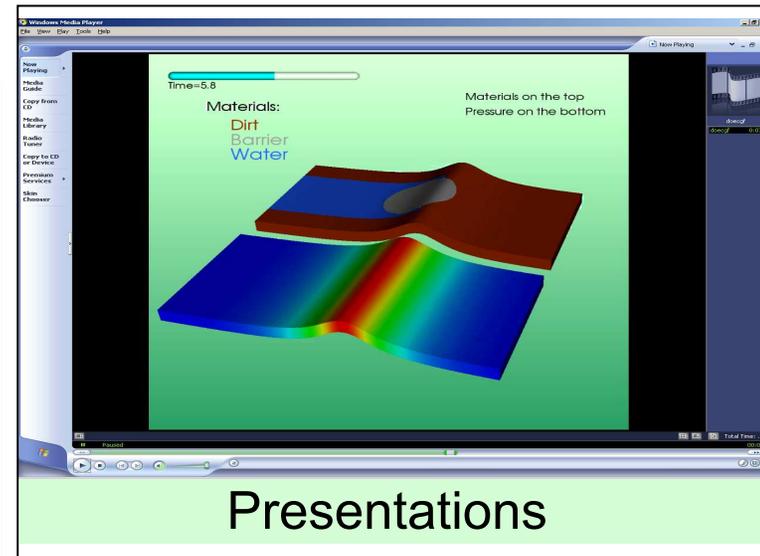
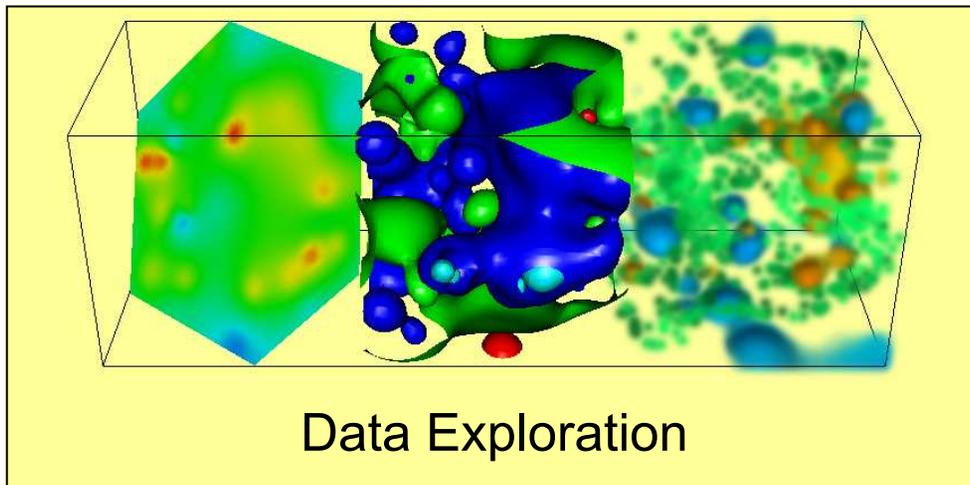
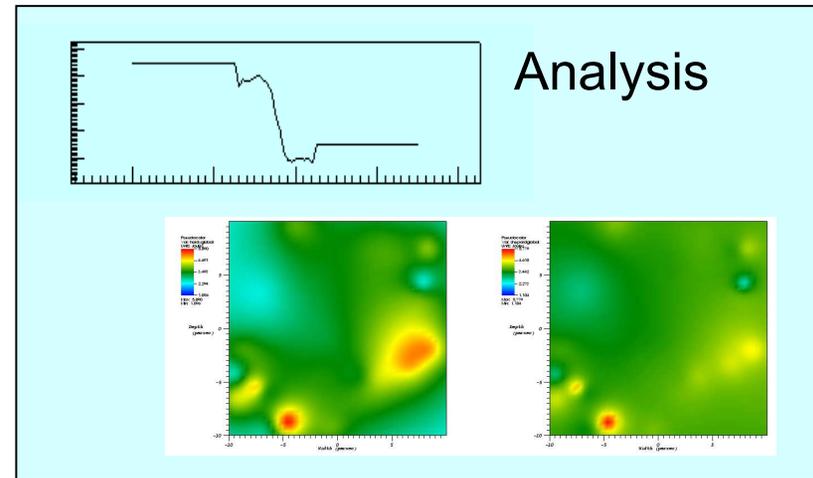
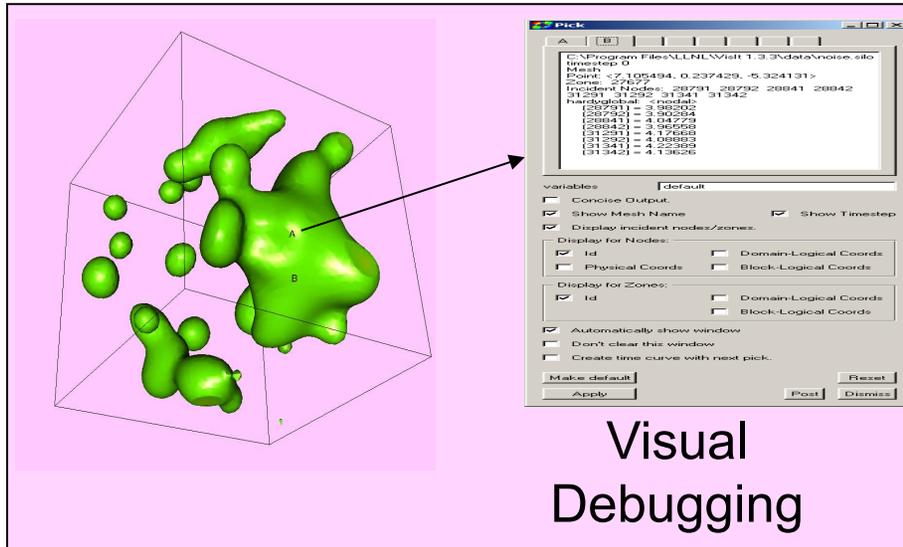


217 pin reactor cooling simulation
Run on 1/4 of Argonne BG/P
Image credit: Paul Fischer, ANL



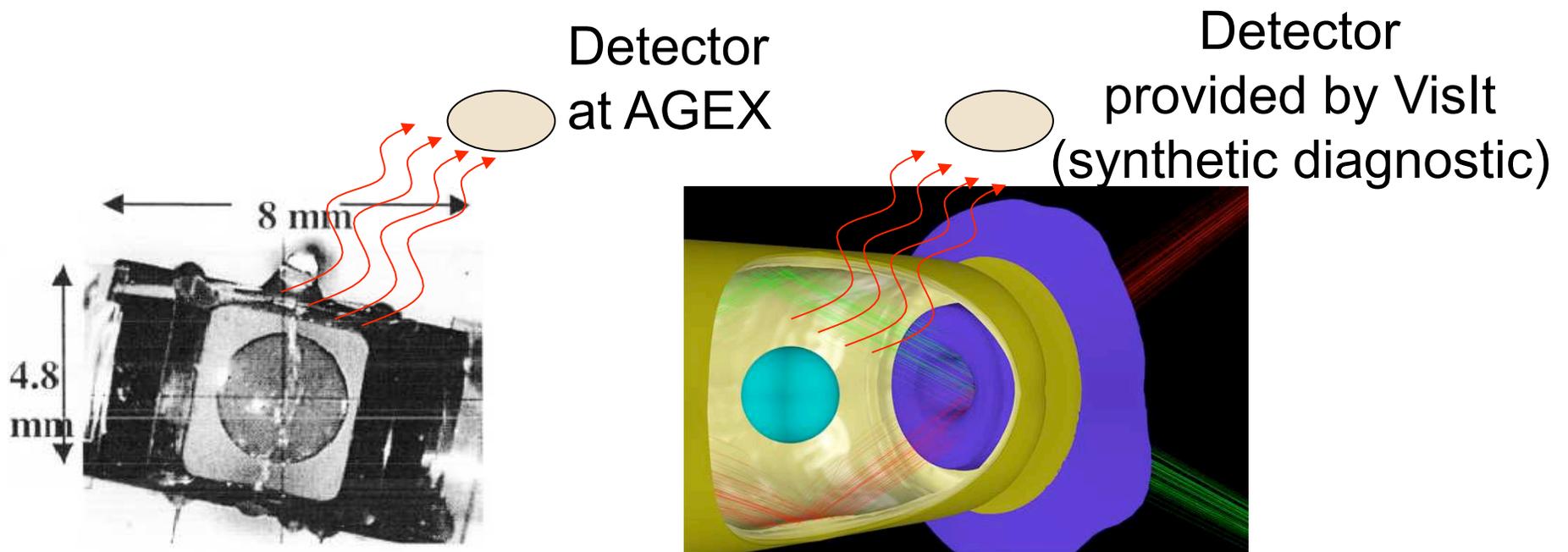
Terribly Named!!!

... intended for much more than just visualization



What sort of analysis is appropriate for VisIt?

- Techniques that span scientific domains (e.g. integration, volumes, surface areas, fluxes, connected components, chord length distributions)
- Specialized analysis (e.g. hohlraum flux at AGEX)



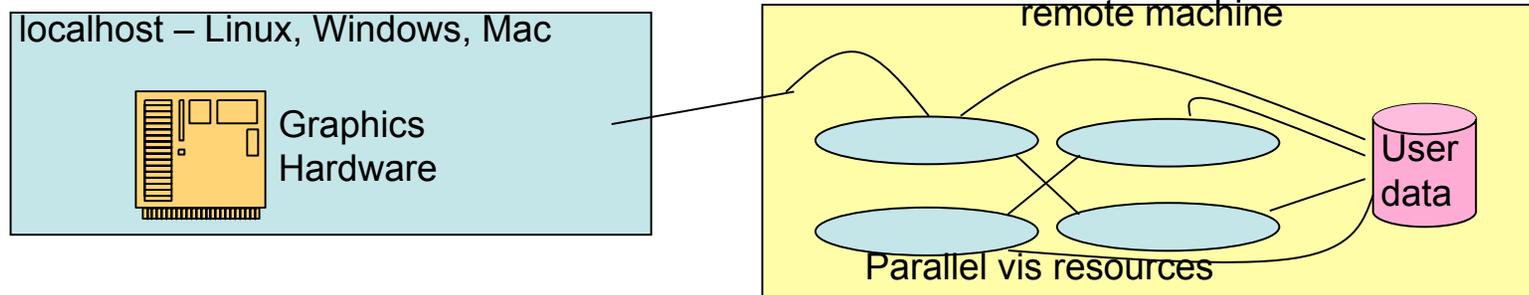
Aside: Why address so many postprocessing problems with a single program?

- Answer: It is economical to do so.
- For developers: shared assets / code re-use
 - ▣ Data infrastructure (esp. for large data)
 - ▣ File readers
 - ▣ Proper interpretation of data (ie material interface reconstruction)
 - ▣ Shared algorithms (multiple ways to slice & dice data)
 - ▣ High flexibility and extensibility
- For users:
 - ▣ Learn a single tool
 - ▣ Divisions between areas of postprocessing often blur

VisIt has a rich feature set.

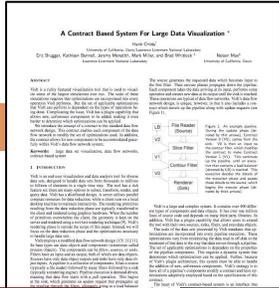
- Meshes: rectilinear, curvilinear, unstructured, point, AMR
- Data: scalar, vector, tensor, material, species
- Dimension: 1D, 2D, 3D, time varying
- Rendering (~15): pseudocolor, volume rendering, hedgehogs, glyphs, mesh lines, etc...
- Data manipulation (~40): slicing, contouring, clipping, thresholding, restrict to box, reflect, project, revolve, ...
- File formats (~115)
- Derived quantities: >100 interoperable building blocks
 - ▣ +, -, *, /, gradient, mesh quality, if-then-else, and, or, not
- Many general features: position lights, make movie, etc
- Queries (~50): ways to pull out quantitative information, debugging, comparative analysis

VisIt employs a parallelized client-server architecture.

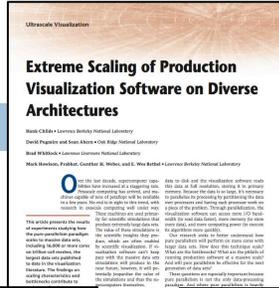


- Client-server observations:
 - ▣ Good for remote visualization
 - ▣ Leverages available resources
 - ▣ Scales well
 - ▣ No need to move data
- Additional design considerations:
 - ▣ Plugins
 - ▣ Heavy use of VTK
 - ▣ Multiple UIs: GUI (Qt), CLI (Python), more...

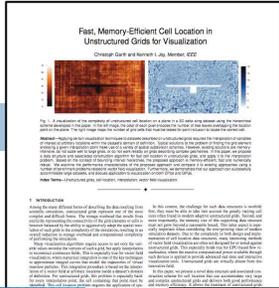
It takes a lot of research to make VisIt work



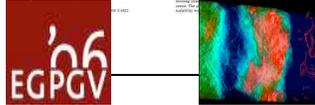
Systems research:
Adaptively applying algorithms in a production env.



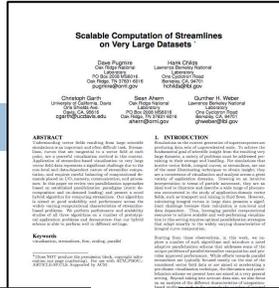
Scaling research:
Scaling to 10Ks of cores and trillions of cells.



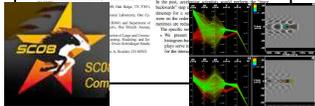
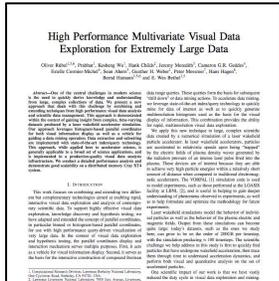
Algorithms research:
Accelerating field evaluation of huge unstructured grids



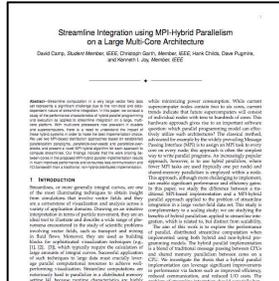
Algorithms research:
How to volume render efficiently in parallel.



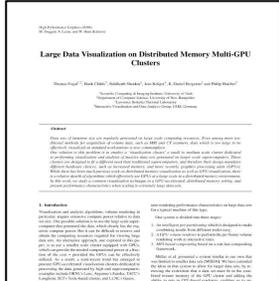
Algorithms research:
How to efficiently calculate particle paths in parallel.



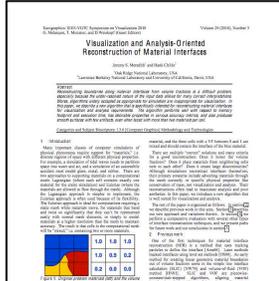
Systems research:
Using smart DB technology to accelerate processing



Architectural research:
Hybrid parallelism + particle advection



Architectural research:
Parallel GPU volume rendering



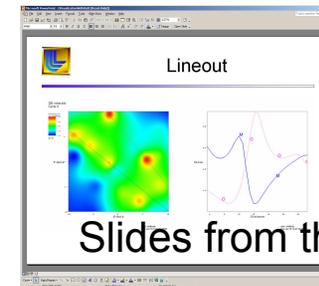
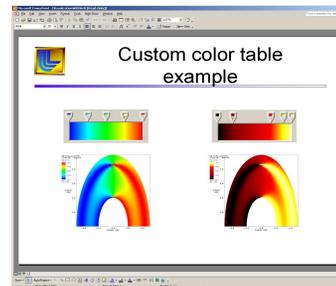
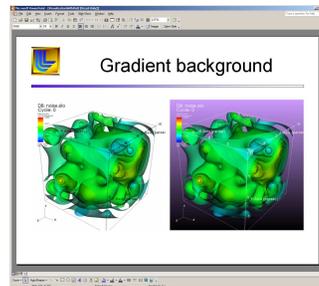
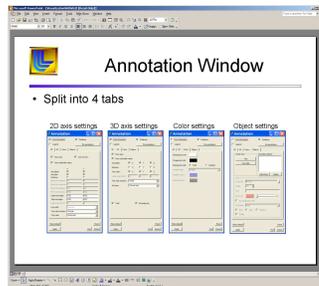
Algorithms research:
Reconstructing material interfaces for visualization



Methods research:
How to incorporate statistics into visualization.

The VisIt team focuses on making a robust, usable product for end users.

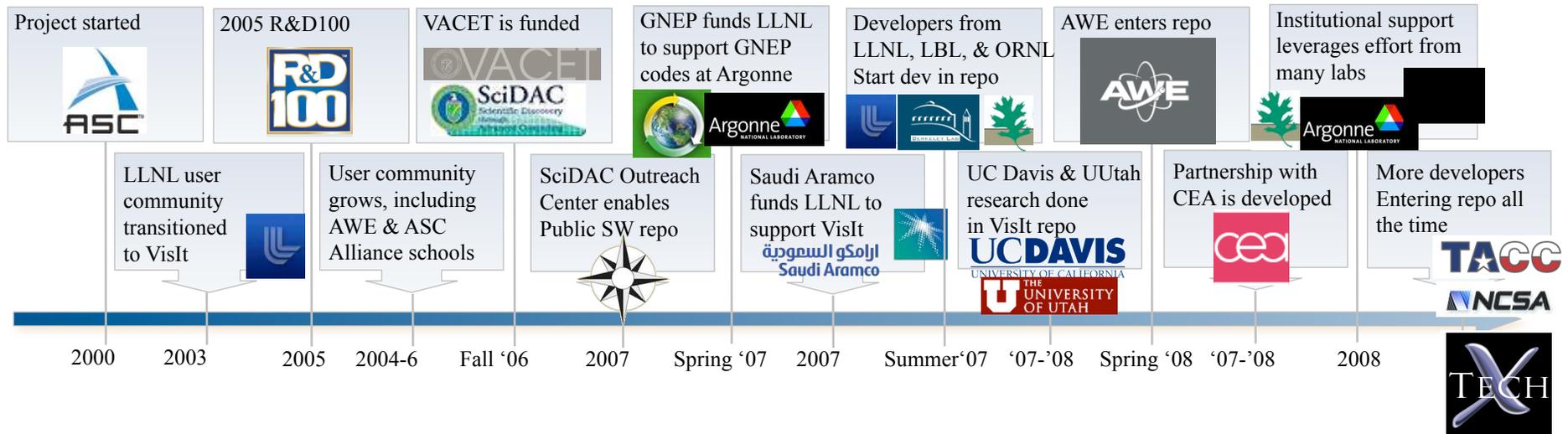
- Manuals
 - 300 page user manual
 - 200 page command line interface manual
 - “Getting your data into VisIt” manual
- Wiki for users (and developers)
- Revision control, nightly regression testing, etc
- Executables for all major platforms
- Tutorial & day long class, complete with exercises



Slides from the VisIt class

VisIt is a vibrant project with many participants.

- Over 75 person-years of effort
- Over 1.5 million lines of code
- Partnership between: Department of Energy's Office of Science, National Nuclear Security Agency, and Office of Nuclear Energy, the National Science Foundation XD centers (Longhorn XD and RDAV), and more....



VisIt: What's the Big Deal?

- Everything works at scale
- Robust, usable tool
- Features that span the “power of visualization”:
 - ▣ Data exploration
 - ▣ Confirmation
 - ▣ Communication
- Features for different kinds of users:
 - ▣ Vis experts
 - ▣ Code developers
 - ▣ Code consumers
- Healthy future: vibrant developer and user communities

“How to make VisIt work after you get home”

- How to get VisIt running on your machine
 - ▣ Downloading and installing VisIt
 - ▣ Building VisIt from scratch
- How to get VisIt to read your data
- How to get help when you run into trouble
- I like the power of VisIt, but I hate the interface
- How to run client-server

“How to make VisIt work after you get home”

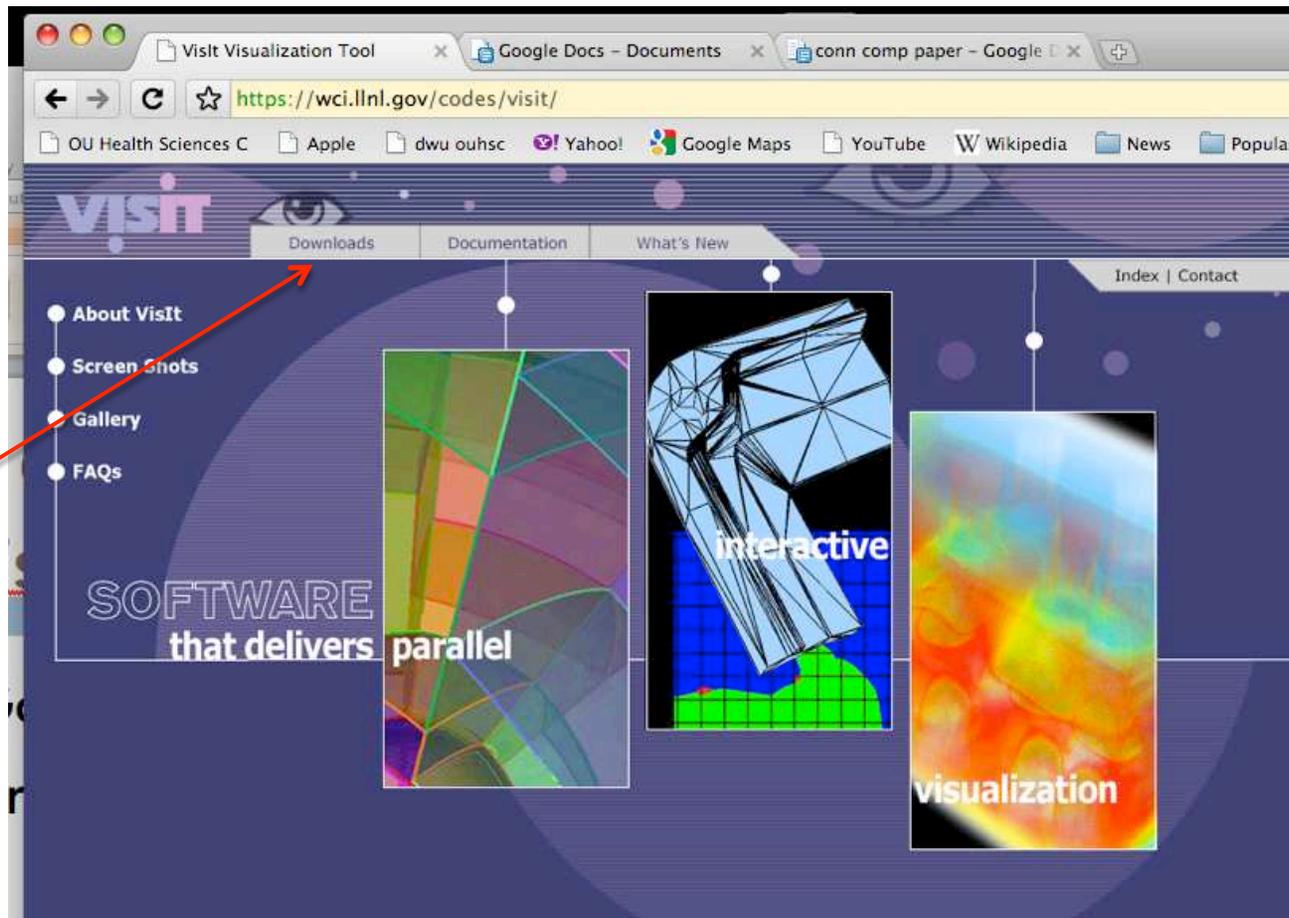
- How to get VisIt running on your machine
 - Downloading and installing VisIt
 - Building VisIt from scratch
- How to get VisIt to read your data
- How to get help when you run into trouble
- I like the power of VisIt, but I hate the interface
- How to run client-server

Can I use a pre-built VisIt binary or do I need to build it myself?

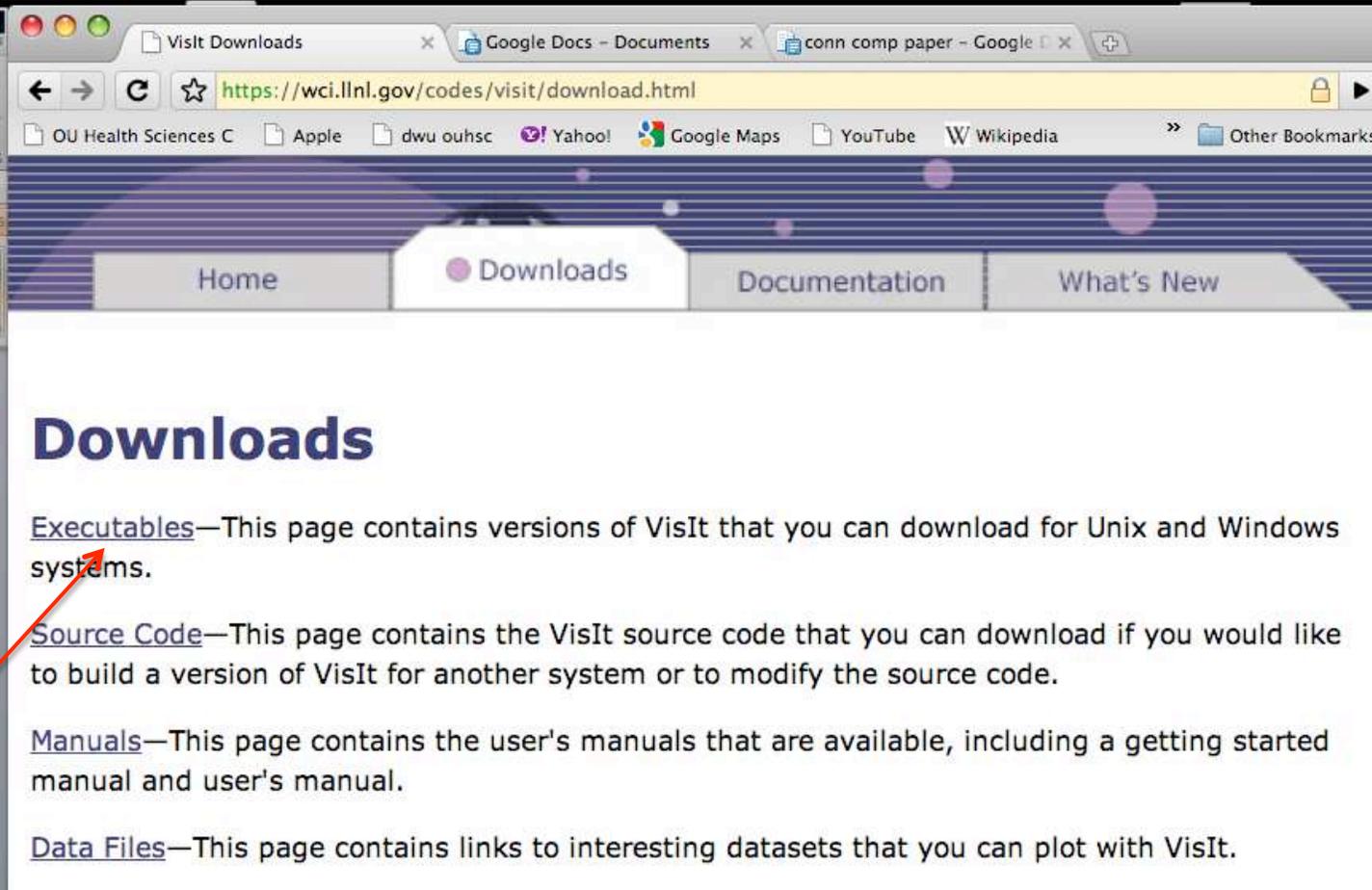
- Pre-built binaries work on most modern machines.
- ... but pre-built binaries are serial only.
 - ▣ Why the VisIt team can't offer parallel binaries:
Your MPI libraries, networking libraries are unlikely to match ours
- Recommendation: try to use the pre-builts first and build VisIt yourself if they don't work.
- Also: all VisIt clients run serial-only. If you want to install VisIt on your desktop to connect to a remote parallel machine, serial is OK.

How do I use pre-built VisIt binaries?

- A: Go to <http://www.llnl.gov/visit>



How do I use pre-built VisIt binaries?



The screenshot shows a web browser window with the URL <https://wci.llnl.gov/codes/visit/download.html>. The browser tabs include "Visit Downloads", "Google Docs - Documents", and "conn comp paper - Google". The address bar shows the URL and a lock icon. Below the address bar is a bookmark bar with entries for "OU Health Sciences C", "Apple", "dwu ouhsc", "Yahoo!", "Google Maps", "YouTube", "Wikipedia", and "Other Bookmarks". The page content features a navigation menu with "Home", "Downloads" (selected), "Documentation", and "What's New". The main heading is "Downloads". Below the heading are four sections: "Executables", "Source Code", "Manuals", and "Data Files", each with a brief description. A red arrow points from the left side of the slide to the "Executables" section.

Downloads

[Executables](#)—This page contains versions of VisIt that you can download for Unix and Windows systems.

[Source Code](#)—This page contains the VisIt source code that you can download if you would like to build a version of VisIt for another system or to modify the source code.

[Manuals](#)—This page contains the user's manuals that are available, including a getting started manual and user's manual.

[Data Files](#)—This page contains links to interesting datasets that you can plot with VisIt.

How do I use pre-built VisIt binaries?

VisIt Executables

This page contains links to download VisIt executables for Unix, Windows, and Mac OS X systems. The page contains several versions of VisIt, organized from the most recent to the oldest. The unix and Mac OS X executables require downloading an install script along with the file containing the executable. The Windows executables are packaged in a self contained installer. Instructions for installing VisIt can be found in the install notes. Md5 and sha1 checksums, as well as file sizes are provided for checking that the files were properly downloaded if corruption of the files is suspected during the download process.

VisIt 2.1.0

- [VisIt release notes](#)
- [VisIt install script](#)
- [VisIt install notes](#)
- [VisIt md5 checksums](#)
- [VisIt sha1 checksums](#)
- [VisIt file sizes](#)

Important

platform	executable
Linux - x86 32 bit Redhat Enterprise Linux 3, hoth.llnl.gov 2.4.21-27.0.2c.ELsmp, gcc 3.2.3 Will work on most Linux x86 systems.	 download
Linux - x86_64 64 bit Ubuntu 8.04, pion.ornl.gov 2.6.24-19, gcc 4.2.4	 download
Linux - x86_64 64 bit Redhat Enterprise Linux 4, photon.ornl.gov 2.6.9-89.0.20.ELsmp, gcc 3.4.6 Will work on most Linux x86_64 systems.	 download

How do I use pre-built VisIt binaries?

Linux - x86_64 64 bit Ubuntu 8.04, pion.ornl.gov 2.6.24-19, gcc 4.2.4	
Linux - x86_64 64 bit Redhat Enterprise Linux 4, photon.ornl.gov 2.6.9-89.0.20.ELsmp, gcc 3.4.6 Will work on most Linux x86_64 systems.	
Linux - x86_64 64 bit Redhat Enterprise Linux 5, yana.llnl.gov 2.6.18-76chaos, gcc 4.1.2 Will work on most Linux x86_64 systems.	
Linux - x86_64 64 bit Scientific Linux SL release 5.4, euclid.nersc.gov 2.6.18-164.9.1.el5-bsd3, gcc 4.1.2	
Windows (Xp / Vista / 7) 32 bit MSVC8, Visual Studio 2005	
Mac OS X - Intel Darwin 10.5, Darwin Kernel Version 9.7.0, gcc 4.0.1, OpenMPI <i>(Includes parallel VisIt compatible with MacOS X 10.5's default MPI)</i>	
Mac OS X - Intel 64 bit Darwin 10.6.3, Darwin Kernel Version 10.3.0, gcc 4.2.1, OpenMPI <i>(Includes parallel VisIt compatible with MacOS X 10.6's default MPI)</i>	
Mac OS X - Intel Darwin 10.4	
AIX - 32 bit AIX 5.3, up.llnl.gov 00C5D6DD4C00, xlc	
AIX - 64 bit AIX 5.3, up.llnl.gov 00C5D6DD4C00, xlc	
Java client library (jar file, compiled classes, source code, examples)	

How do I use the pre-built VisIt binaries?

□ Unix & Mac:

- Download install script
- Download binary
- Run install script
- --or—
- Download binary
- Untar



Good for host profiles,
maintaining multiple versions,
multiple OSs



Quick & easy

□ Windows:

- Download installer program & run

□ Full install notes:

- https://wci.llnl.gov/codes/visit/2.1.0/INSTALL_NOTES

Important step: choosing host profiles

- Many supercomputing sites have set up “host profiles”.
 - ▣ These files contain all the information about how to connect to their supercomputers and how to launch parallel jobs there.
- You select which profiles to install when you install VisIt.
- Profiles that come with VisIt:
 - ▣ NERSC, LLNL Open, LLNL Closed, ORNL, Argonne, TACC, LBNL desktop network, Princeton, UMich CAC
- Other sites maintain profiles outside of VisIt repository.
 - ▣ If you know folks running VisIt in parallel at a site not listed above, ask them for their profiles.

“How to make VisIt work after you get home”

- How to get VisIt running on your machine
 - Downloading and installing VisIt
 - Building VisIt from scratch
- How to get VisIt to read your data
- How to get help when you run into trouble
- I like the power of VisIt, but I hate the interface
- How to run client-server

Building VisIt from scratch

- Building VisIt from scratch on your own is very difficult.
- ... but the “build_visit” script is fairly reliable.

Automatically build VisIt with the *build_visit* script!

[Download build_visit script here.](#)

VisIt can now be built automatically using the [build_visit](#) script on many Linux, MacOS X, and AIX platforms (*more to come*). The [build_visit](#) script takes care of downloading relevant VisIt and 3rd party source code, configuring, and building it all using your C++ compiler. We encourage users to build VisIt using the [build_visit](#) script when our binary distributions have trouble running on some systems. We also recommend using the [build_visit](#) script on your system if you plan to:

- Modify the VisIt source code.
- Run a parallel compute engine. Building a parallel version of VisIt on your system allows you to configure VisIt so it uses your MPI library, avoiding incompatibilities.
- Create your own VisIt plugins. Building VisIt on your system ensures that it is built with the same C++ compiler that you will use to develop your plugin, minimizing the chance for runtime library incompatibilities.



(build_visit screen shot)

What “build_visit” does



- Downloads third party libraries
- Patches them to accommodate OS quirks
- Builds them
- Creates “config-site” file, which communicates information about where 3rd party libraries live to VisIt’s build system.
- Downloads VisIt source code
- Builds VisIt

“build_visit” details



- There are two ways to use build_visit:
 - Curses-style GUI
 - Command line options through `–console`
 - Developers all use `–console` and it shows!!
- Tips:
 - Don't build every third party library unless you really need to.
 - Set up a “`—thirdparty-path`”.

“build_visit” details



- Q: How long does build_visit take? A: hours
- Q: I have my own Qt/VTK/Python, can I use those?
 - ▣ Hank highly recommends against
- Q: What happens after build_visit finishes?
 - ▣ A1: you can run directly in the build location
 - ▣ A2: you can make a package and do an install like you would with the pre-built binaries

“build_visit” details



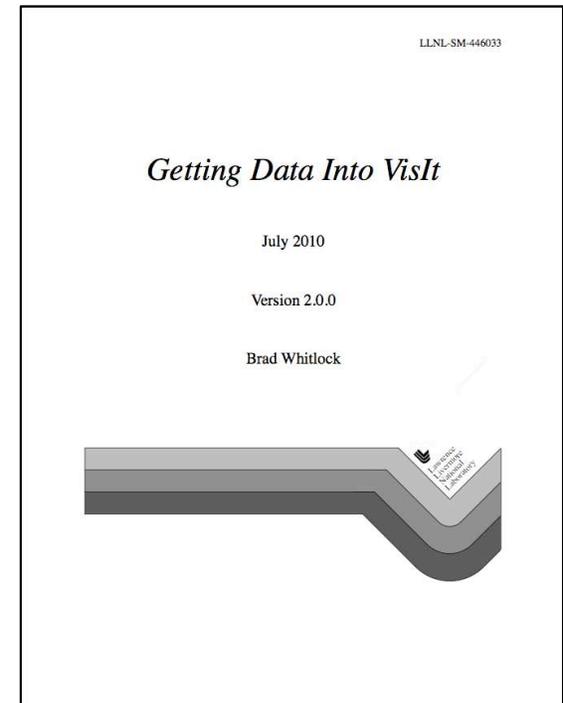
- Most common build_visit failures:
 - gcc is not installed
 - X11 development package is not installed
 - OpenGL development package is not installed
- Most common VisIt runtime failure: really antique OpenGL drivers.
 - Hank runs SUSE 9.1 (from 2005) at home.
- Build process for Windows is very different. Rarely a need to build on Windows, aside from VisIt development.

“How to make VisIt work after you get home”

- How to get VisIt running on your machine
 - Downloading and installing VisIt
 - Building VisIt from scratch
- How to get VisIt to read your data
- How to get help when you run into trouble
- I like the power of VisIt, but I hate the interface
- How to run client-server

How to get your data into VisIt

- There is an extensive (and up-to-date!) manual on this topic: “Getting Your Data Into VisIt”
- Three ways:
 - Use a known format
 - Write a file format reader
 - In situ processing
 - Latter two covered in afternoon course

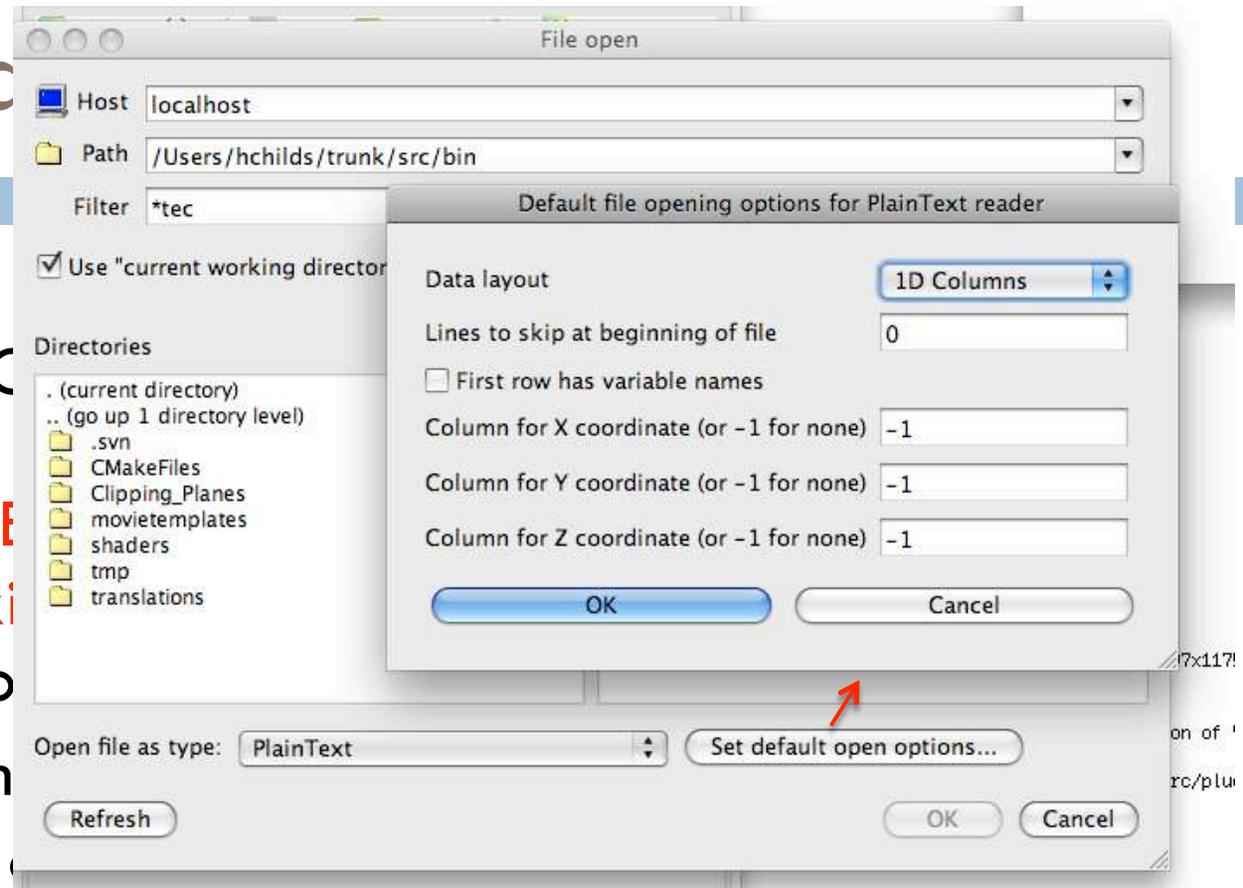


File formats that VisIt supports

- ADIOS, BOV, Boxlib, CCM, CGNS, Chombo, CLAW, EnSight, ENZO, Exodus, FLASH, Fluent, GDAL, Gadget, Images (TIFF, PNG, etc), ITAPS/MOAB, LAMMPS, NASTRAN, NETCDF, Nek5000, OpenFOAM, PLOT3D, PlainText, Pixie, Shapefile, Silo, Tecplot, VTK, Xdmf, Vs, and many more
 - 113 total readers
- Some readers are more robust than others.
 - For some formats, support is limited to flavors of a file a VisIt developer has encountered previously (e.g. Tecplot).

File format

- ADIOS, BOV, EnSight, ENZO, Images (TIFF, NASTRAN, NE, PlainText, Pixi and many more
- BOV: raw binary
 - → you have a file that describes dimensions
- PlainText: reads space delimited columns.
 - Controls for specifying column purposes



File formats that VisIt supports

- ADIOS, **BOV**, Boxlib, CCM, CGNS, Chombo, CLAW, EnSight, ENZO, Exodus, FLASH, Fluent, GDAL, Gadget, Images (TIFF, PNG, etc), ITAPS/MOAB, LAMMPS, NASTRAN, **NETCDF**, Nek5000, OpenFOAM, PLOT3D, **PlainText**, **Pixie**, Shapefile, **Silo**, Tecplot, **VTK**, **Xdmf**, **Vs**, and many more
- NETCDF: VisIt reader understands many (but not all) conventions
- Pixie: most general HDF5 reader
 - Many other HDF5 readers

File formats that VisIt supports

- ADIOS, **BOV**, Boxlib, CCM, CGNS, Chombo, CLAW, EnSight, ENZO, Exodus, FLASH, Fluent, GDAL, Gadget, Images (TIFF, PNG, etc), ITAPS/MOAB, LAMMPS, NASTRAN, **NETCDF**, Nek5000, OpenFOAM, PLOT3D, **PlainText**, **Pixie**, Shapefile, **Silo**, Tecplot, **VTK**, **Xdmf**, **Vs**, and many more
- Xdmf: specify an XML file that describes semantics of arrays in HDF5 file
- VizSchema (Vs): add attributes to your HDF5 file that describes semantics of the arrays.

File formats that VisIt supports

- ADIOS, **BOV**, Boxlib, CCM, CGNS, Chombo, CLAW, EnSight, ENZO, Exodus, FLASH, Fluent, GDAL, Gadget, Images (TIFF, PNG, etc), ITAPS/MOAB, LAMMPS, NASTRAN, **NETCDF**, Nek5000, OpenFOAM, PLOT3D, **PlainText**, **Pixie**, Shapefile, **Silo**, Tecplot, **VTK**, **Xdmf**, **Vs**, and many more
- VTK: not built for performance, but it is great for getting into VisIt quickly
- Silo: higher barriers to entry, but performs well and fairly mature

VTK File Format

- The VTK file format has both ASCII and binary variants.

- Great documentation at

<http://www.vtk.org/VTK/img/file-formats.pdf>

- Easiest way to write VTK files: use VTK modules

- ... but this creates a dependence on the VTK library

- You can also try to write them yourself, but this is an error prone process.

- Third option: visit_writer



File Formats

for VTK Version 4.2

(Taken from The VTK User's Guide
Contact Kitware www.kitware.com to purchase)

VTK File Formats

The *Visualization Toolkit* provides a number of source and writer objects to read and write popular data file formats. The *Visualization Toolkit* also provides some of its own file formats. The main reason for creating yet another data file format is to provide a consistent data representation for a variety of dataset types, and to provide a simple method to communicate it between software. Whenever possible, we recommend that you use formats that are more widely used. But if this is not possible, the *Visualization Toolkit* formats described here can be used instead. Note that these formats may not be supported by many other tools.

There are two different styles of file formats available in VTK. The simplest are the legacy, serial formats that are easy to read and write either by hand or programmatically. However, these formats are less flexible than the XML based formats described later in this section. The XML formats support random access, parallel I/O, and portable data compression so are preferred to the serial VTK file formats whenever possible.

Simple Legacy Formats

The legacy VTK file formats consist of five basic parts.

1. The first part is the file version and identifier. This part contains the single line: `# vtk DataFile Version x.x`. This line must be exactly followed with the version of the version number `x.x`, which will vary with different releases of VTK. Note: files created with version 3.0 version 4.0 files are compatible with version 3.0
2. The second part is the header. The header consists of a character string terminated by end-of-line character `\n`. The header is 256 characters maximum. The header can be used to describe the data and include any other pertinent information.
3. The next part is the file format. The file format describes the type of file, either ASCII or binary. On this line the string `ASCII` or `BINARY` must appear.
4. The fourth part is the geometry. The geometry describes the geometry of the dataset. This part begins with a keyword, `POINTS` or `CELLS`, followed by the keyword `ARRAYS` and the type of dataset. Then, depending upon the type of dataset, other keyword/data combinations define the actual data.
5. The final part describes the dataset attributes. This part begins with the keywords `POINT_DATA` or `CELL_DATA`, followed by an integer number specifying the number of points or cells, respectively. (It doesn't matter whether `POINT_DATA` or `CELL_DATA` comes first.) Other keyword/data combinations then define the actual dataset attribute values (i.e., scalars, vectors, tensors, normals, texture coordinates, or field data).

An overview of the file format is shown in **Figure 1**. The first three parts are mandatory, but the other two are optional. Thus you have the flexibility of mixing and matching dataset attributes and geometry, either by operating system file manipulation or using VTK filters to merge data. Keywords are case insensitive, and may be separated by whitespace.

Before describing the data file formats please note the following.

- * *dataType* is one of the types `bit`, `unsigned_char`, `char`, `unsigned_short`, `short`, `unsigned_int`, `int`, `unsigned_long`, `long`, `float`, or `double`. These keywords are used to describe the form of the data, both for reading from file, as well as constructing the appropriate internal objects. Not all data types are supported for all classes.

VisItWriter writes VTK files

- It is a “library” (actually a single C file) that writes VTK-compliant files.
 - ▣ The typical path is to link `visit_writer` into your code and write VTK files
- There is also Python binding for `visit_writer`.
 - ▣ The typical path is to write a Python program that converts from your format to VTK
- Both options are short term: they allow you to play with VisIt on your data. If you like VisIt, then you typically formulate a long term file format strategy.
- More information on `visit_writer`:
 - ▣ <http://visitusers.org/index.php?title=VisItWriter>

Python VisitWriter in action

```
import visit_writer
import math
import sys

nX = 20
nY = 20
conn = []
for i in range(nX-1):
    for j in range(nY-1):
        pt1 = j*(nX) + i;
        pt2 = j*(nX) + i+1;
        pt3 = (j+1)*(nX) + i+1;
        pt4 = (j+1)*(nX) + i;
        conn.append([ "quad", pt1, pt2, pt3, pt4 ])

pts = []
rad = []
for i in range(nX):
    for j in range(nY):
        pts.extend([ float(i), float(j), 0 ])
        rad.append( math.sqrt(i*i + j*j) )

var_datum = [ "radius", 1, 1, rad ]
vars = [ var_datum ]
visit_writer.WriteUnstructuredMesh("ugrid.vtk", 0, pts, conn, vars)

sys.exit()
```

Silo file format

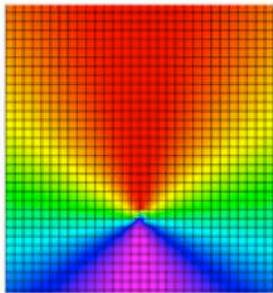


- Silo is a mature, self-describing file format that deals with multi-block data.
- It has drivers on top of HDF5, NetCDF, and “PDB”.
- Fairly rich data model
- More information:
 - ▣ <https://wci.llnl.gov/codes/silo/>

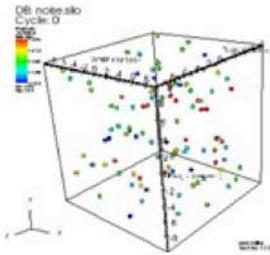
Silo features

Welcome to Silo

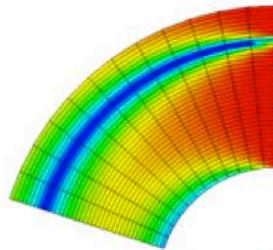
A mesh and field I/O library and scientific database



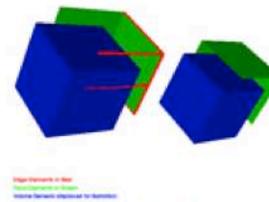
Structured Rectilinear Mesh



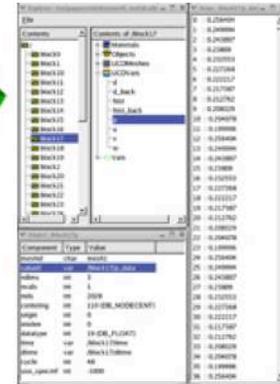
Gridless Point Mesh



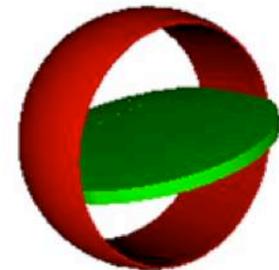
Structured (Curvilinear) Mesh



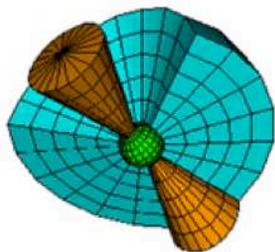
Arbitrary Subsets



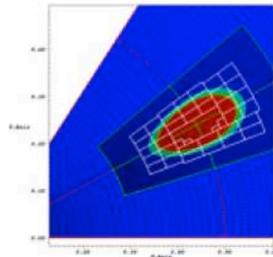
Silex browser for Silo files



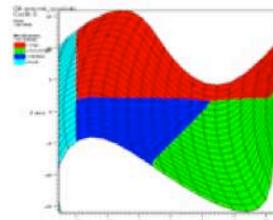
Constructive Solid Geometry (CSG) Mesh



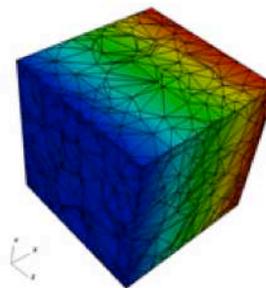
Unstructured Zoo (UCD) Mesh



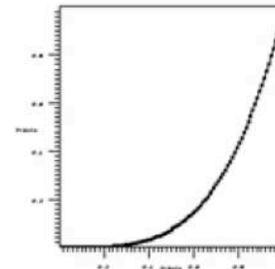
Adaptive Mesh Refinement (AMR) Mesh



Mixing Materials



Arbitrary Polyhedral Mesh



XY Curve

“How to make VisIt work after you get home”

- How to get VisIt running on your machine
 - Downloading and installing VisIt
 - Building VisIt from scratch
- How to get VisIt to read your data
- How to get help when you run into trouble
- I like the power of VisIt, but I hate the interface
- How to run client-server

How to get help when you run into trouble

□ Six options:

□ FAQ

- <http://visit.llnl.gov/FAQ.html>

□ Documentation

- <https://wci.llnl.gov/codes/visit/doc.html>
- <http://www.visitusers.org>

□ VisIt-users mailing list

□ VisIt-users archives

□ VisIt users forum

□ VisIt-help-XYZ mailing list



Manuals & documentation



- Getting started manual
- Users manual (old, but still useful)
- Python interface (to be updated in two weeks)
- Getting Data Into VisIt
- VisIt Class Slides
- VisIt Class Exercises
- This Tutorial

Visitusers.org

- Users section has lots of practical tips:
 - “I solved this problem with this technique”
 - “Here’s my script to do this functionality”
- In practical terms, this is a staging area for formal documentation in the future.

Misc

[edit]

- Using VisIt in an mxterm
- Using derived data functions (DDFs)
- Using the command line interface
- How volume rendering works in VisIt
- Using cross-mesh field evaluations ... how to do differences, access other time slices, etc
- Keyframing example
- Exporting databases
- Directions for specific machines
- Using the VisIt Python API with a standard Python interpreter
- Pages that contain instructions specific to certain user groups and needs
- Issues related to running VisIt on Windows under cygwin
- VisIt's Camera model
- Using VisIt's mpeg2encode
- Molecular data features
- Extracting alpha
- (Very) High resolution rendering
- Elevating shapefiles
- Raytracing your visualizations with POV-Ray and a tutorial POV-Ray exporting example

FAQ: <http://visit.llnl.gov/FAQ.html>



Frequently Asked Questions

1. [Contact information](#)
2. [Supported platforms](#)
3. [Optimal hardware/software](#)
4. [Debugging problems starting VisIt or opening files](#)
5. [Stereo rendering](#)
6. [VisIt won't run on Linux](#)
7. [Slow performance on Linux](#)
8. [Slow performance Using SSH](#)
9. [No output in visualization window](#)
10. [Accessing data on remote machine](#)
11. [Running VisIt in parallel](#)
12. [Supported data file formats](#)
13. [Getting your data into VisIt](#)
14. [Making a movie of your data](#)
15. [Setting your user name to connect to a remote machine](#)
16. [Cannot connect to a remote computer](#)
17. [Building VisIt on a Windows computer](#)
18. [Installing VisIt on a MacOS X computer](#)
19. [Hanging at 12% on Windows computers](#)
20. [Getting the Plugin Developer's Guide](#)
21. [Writing a plugin for VisIt](#)
22. [When new versions of VisIt are released](#)
23. [What is new in the latest version of VisIt](#)
24. [Compilers that can be used to build VisIt](#)
25. [VisIt's licensing agreement](#)
26. [Slow performance with ATI cards on Linux](#)
27. [Custom plugins with a downloaded VisIt binary](#)
28. [Getting HDF5 data into VisIt](#)
29. [Getting NETCDF data into VisIt](#)
30. [When I run VisIt on my Linux machine, I get a black screen](#)
31. [I get the message 'Publisher cannot be verified' when installing VisIt on Windows](#)
32. [Which libraries should I enable in build_visit?](#)

[visit-users] Building Parallel Visit: Issue w/ Qt

Vedran Coralic vcoralic@caltech.edu

Wed Nov 3 00:21:37 EDT 2010

- Previous message: [\[visit-users\] Building Parallel Visit: Issue w/ Qt](#)
- Next message: [\[visit-users\] makemili](#)
- Messages sorted by: [\[date \]](#) [\[thread \]](#) [\[subject \]](#) [\[author \]](#)

Thank you very much Jeremy! That seemed to do the trick. I have now finished successfully building VisIt.

2010/11/2 Meredith, Jeremy S. <jsmeredith@ornl.gov>

> Here's what I did to work around this problem:

> - when the Qt build fails, cd into the Qt directory and type "make install"

> - this appears to immediately start putting the libraries in the installation location even though the build "failed"

> - as soon as it's put the libQCLucene stuff into the installation location, kill the build

> - now type "make", and it will finish building successfully

> - and when it's done, type "make install" and it will finish installing

Archives by thread

[\[subject \]](#) [\[author \]](#) [\[date \]](#)

EDT 2010

ST 2010

[Building Parallel Visit: Issue w/ Qt](#) Vedran Coralic
[Building Parallel Visit: Issue w/ Qt](#) Meredith, Jeremy S.
[Building Parallel Visit: Issue w/ Qt](#) Vedran Coralic
Daniel, James L ERDC-GSL-MS
[makemili](#) Seipel, William F NWO
[makemili](#) Seipel, William F NWO
[Eclipse/RCP](#) Leguay Romain
[Eclipse/RCP](#) Hank Childs
[of recorded macros?](#) Cyrus Harrison
[of hardware acceleration problems](#) Patrick Shinpaugh
[on a CentOS server](#) Katie Boyle

- [\[visit-users\] Running Visit on a CentOS server](#) Cyrus Harrison
- [\[visit-users\] Running Visit on a CentOS server](#) J.S. van Bethlehem
- [\[visit-users\] Controlling Annotation objects through cli](#) Shriram Jagannathan
- [\[visit-users\] Controlling Annotation objects through cli](#) Cyrus Harrison
 - [\[visit-users\] Controlling Annotation objects through cli](#) Shriram Jagannathan
 - [\[visit-users\] Controlling Annotation objects through cli](#) J.S. van Bethlehem

question from Australia to be answered by a European air while I'm asleep

- ❑ List: visit-users@ornl.gov
- ❑ More information: <https://email.ornl.gov/mailman/listinfo/visit-users>
- ❑ Archive: <https://email.ornl.gov/pipermail/visit-users/>

Board Topics								
		CGNS OversetHoles « Pages 1 2 »		tpg2114	16	160	👉 11/11/10 at 22:39:11 By: cean	<input type="checkbox"/>
		3d vector on 2d mesh?		tsch	2	34	👉 11/10/10 at 09:26:36 By: Jeremy Meredith	<input type="checkbox"/>
		Image messed up when save		Pinpin	13	150	👉 11/09/10 at 12:14:29 By: BradWhitlock	<input type="checkbox"/>
		pseudocolor plot legend attributes in python		Jennifer	2	17	👉 11/07/10 at 22:11:27 By: Jennifer	<input type="checkbox"/>
		graph along 2D cur						
		threshold variable depe						
		Python compatibility iss						
		python interface ... sca						
		Mesa: 'make' for M						
		failed						
		applyOperator in py						
		how to get cycle on tim						
		annotation? « Pages 1 2 »						
		Averaging 2D slices						
		databases						
		smooth operator						
		Appearance of lines in i						
		Add and read par						
		No image was save						

Members viewing this topic (1): **Hank Childs.**

pseudocolor plot legend attributes in python (Read 18 times)

Jennifer
YaBB Newbies

Posts: 4
Fort Collins, CO

pseudocolor plot legend attributes in python
11/07/10 at 19:06:30

Quote Modify Delete

Hello. I want to set the attributes for a pseudocolor plot legend in a python script such as the location of the legend (turn off Let VisIt manage legend position), the X-scale & Y-scale, the number of Tic Marks, and the label appearance (number format, font height). Is it possible to set these properties in a python script? If so, how can I do this?

I tried to use the Command Control to record these changes, but the output states:
"# Logging for AddAnnotationObject is not implemented yet.
Logging for SetAnnotationObjectOptions is not implemented yet."

Thanks,
Jennifer

Back to top PM IP Logged

Hank Childs
YaBB Moderator

I use VisIt and I develop VisIt

Posts: 135
Davis, CA

Re: pseudocolor plot legend attributes in python
Reply #1 - 11/07/10 at 19:47:03

Quote Modify Split Delete

Hello Jennifer,

Each plot has an index and the plot's legend is referred to through that same index.

```
>>> GetAnnotationObjectNames()
('Plot0003',)
>>> a = GetAnnotationObject("Plot0003")
>>> a
active = 1
managePosition = 1
position = (0.05, 0.9)
xScale = 1
yScale = 1
```

Visit-help-xyz



- Some customer groups pay for Visit funding and get direct support.
 - ▣ These customers can post directly to visit-help-xyz without being a subscriber
 - ▣ The messages are received by all Visit developers and supported communally
- Lists:
 - ▣ Visit-help-asc, visit-help-scidac, visit-help-gnep, visit-help-ascem

“How to make VisIt work after you get home”

- How to get VisIt running on your machine
 - Downloading and installing VisIt
 - Building VisIt from scratch
- How to get VisIt to read your data
- How to get help when you run into trouble
- I like the power of VisIt, but I hate the interface
- How to run client-server

It is possible (although non-trivial) to write a custom user interface to VisIt

The screenshot displays the VORPAL Composer interface, which is divided into several panels:

- CONTROLS:** A sidebar on the left containing a vertical menu with icons for Welcome, Input, Run, Output, Visualize, and Help. Below this is a list of variables categorized into Scalars, Vectors, and Meshes. The 'Slice' dropdown is set to 'YeeElecField_1 (YeeElecField)'. The 'Align slice/clip to axis' buttons for X, Y, and Z are visible. The 'Annotation Level' is set to '4 - all' and the 'Primary Window' is set to '3d view'. Buttons for 'Reset Views' and 'Save Image' are at the bottom.
- VISUALIZATION:** The main central area showing a 3D view of a simulation. It features a 3D coordinate system with X, Y, and Z axes. A central red arrow points downwards, labeled 'Normal <0 0 1>'. Three green, rounded rectangular volumes are arranged horizontally along the X-axis. A red box highlights the 'Origin <0 0 0>'.
- 2-d view:** A smaller window on the right showing a 2D projection of the simulation data.
- Line-out:** A window on the right showing a line plot of simulation data over time.

At the bottom right of the visualization window, the text reads: 'user: mdlurant Fri Nov 12 17:14:20 2010'.

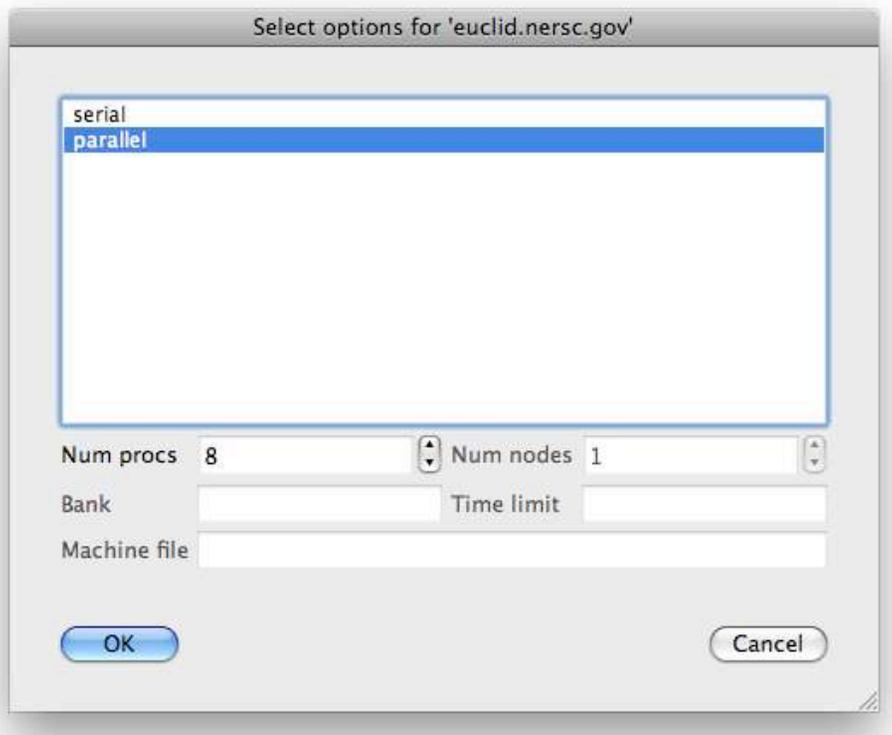
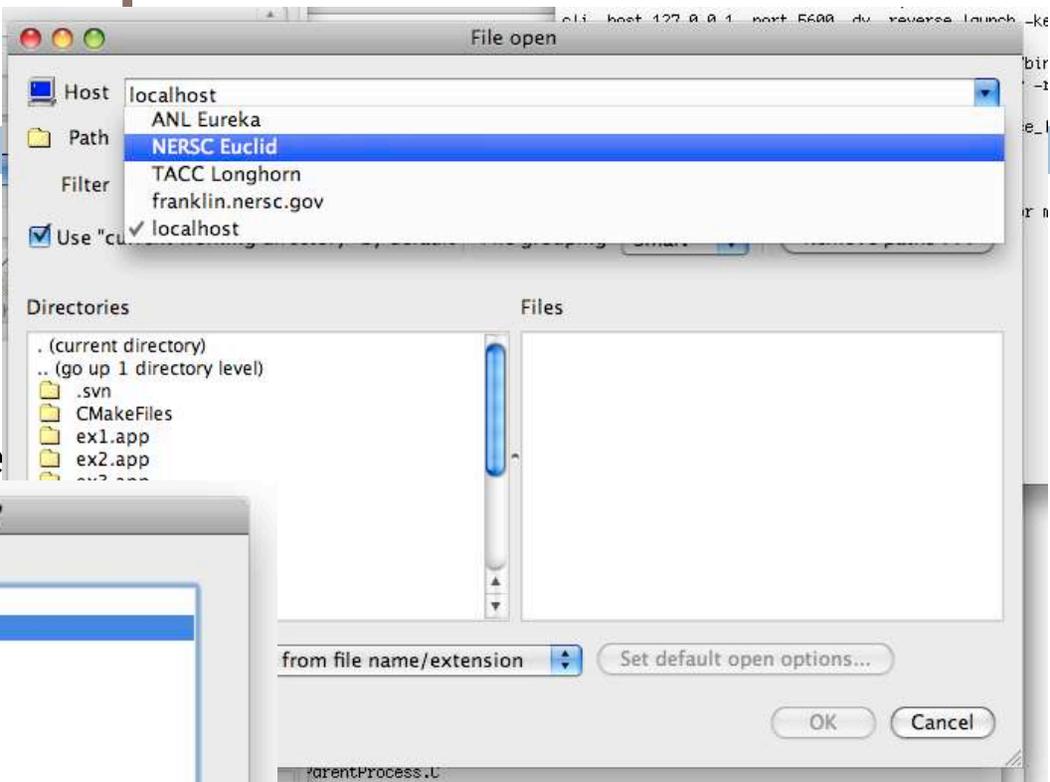


“How to make VisIt work after you get home”

- How to get VisIt running on your machine
 - ▣ Downloading and installing VisIt
 - ▣ Building VisIt from scratch
- How to get VisIt to read your data
- How to get help when you run into trouble
- I like the power of VisIt, but I hate the interface
- **How to run client-server**

How to run cli

- There are two critical steps:
 - Connecting to the host
 - Getting an engine



VisIt's Data Model



- A very rich data model
 - Closer to the “computational model”
- Internally implemented with VTK
- Many conventions built on top of VTK

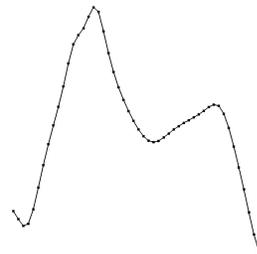
Meshes



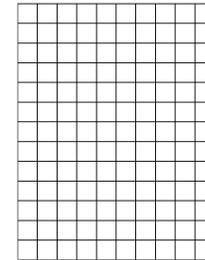
- All data in VisIt lives on a mesh
- Discretizes space into points and cells
 - ▣ 1D, 2D, 3D
 - ▣ All of these over time (up to 4D)
 - ▣ Can have lower-dimensional meshes in a higher-dimensional space (e.g. 2D surface in 3D space)
- Provides a place for data to be located
- Defines how data is interpolated

Mesh types

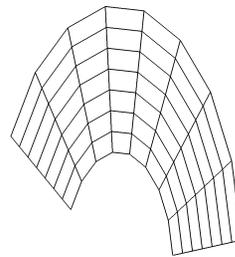
- 1D Curves
- 2D/3D meshes
 - Rectilinear
 - Curvilinear
 - Unstructured
 - Points
 - Molecular



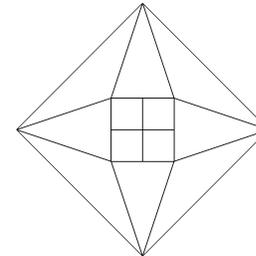
Curve



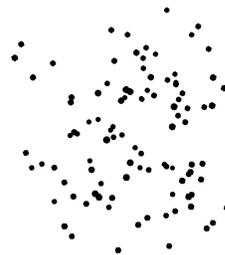
Rectilinear



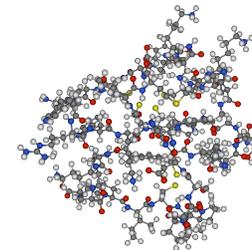
Curvilinear



Unstructured



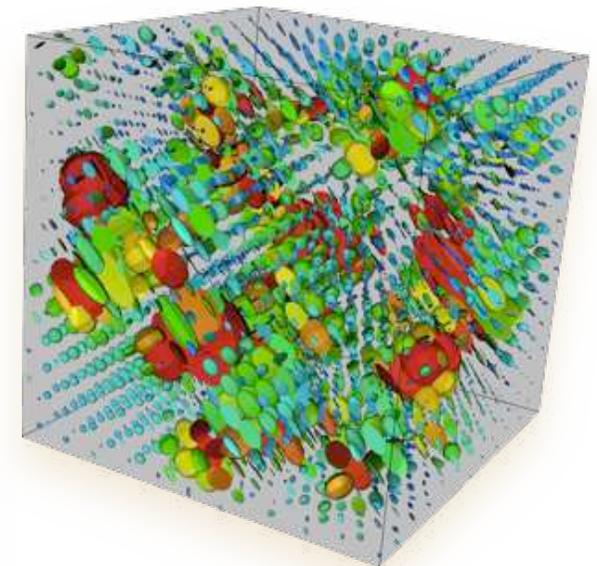
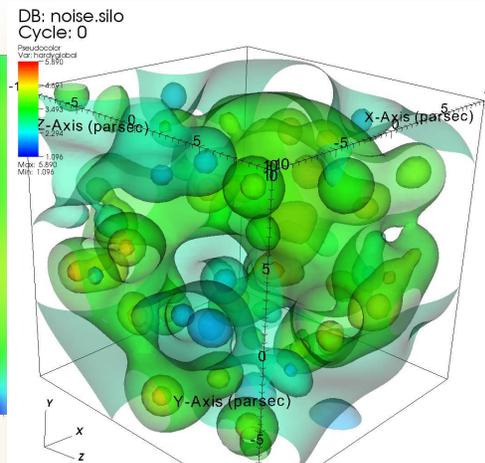
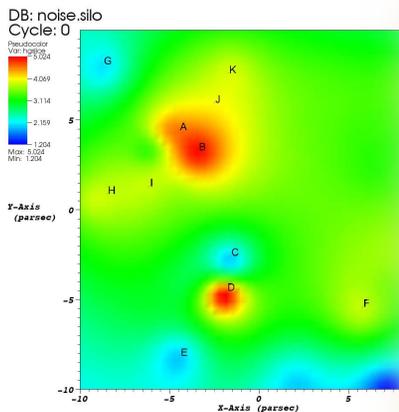
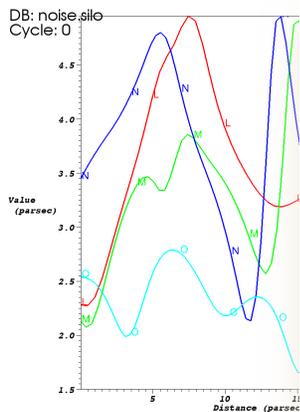
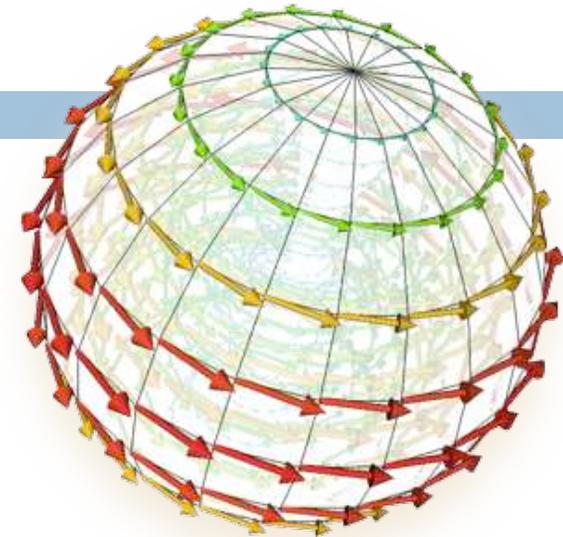
Points



Molecular

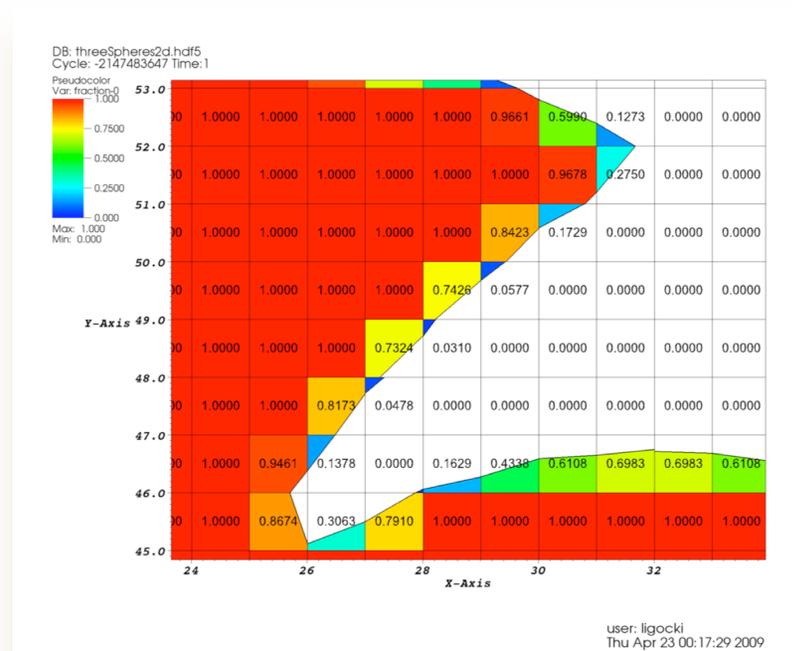
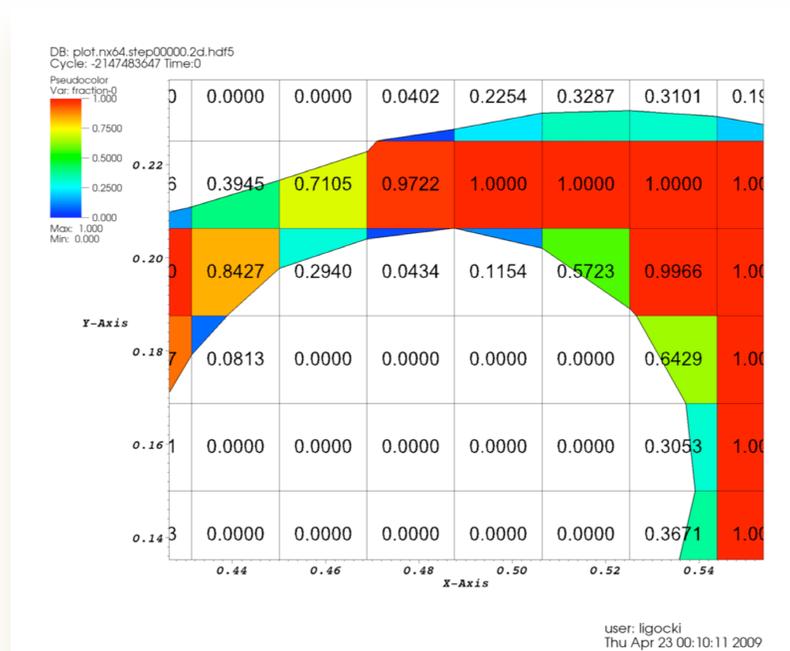
Variables

- Scalars, Vectors, Tensors
- Sits on points or cells of a mesh
 - ▣ Points: linear interpolation
 - ▣ Cells: piecewise constant
- Can have different dimensionality than the mesh (e.g. 3D vector data on a 2D mesh)



Materials

- Describes disjoint spatial regions at a sub-grid level
- Volume/area fractions
- VisIt will do high-quality sub-grid material interface



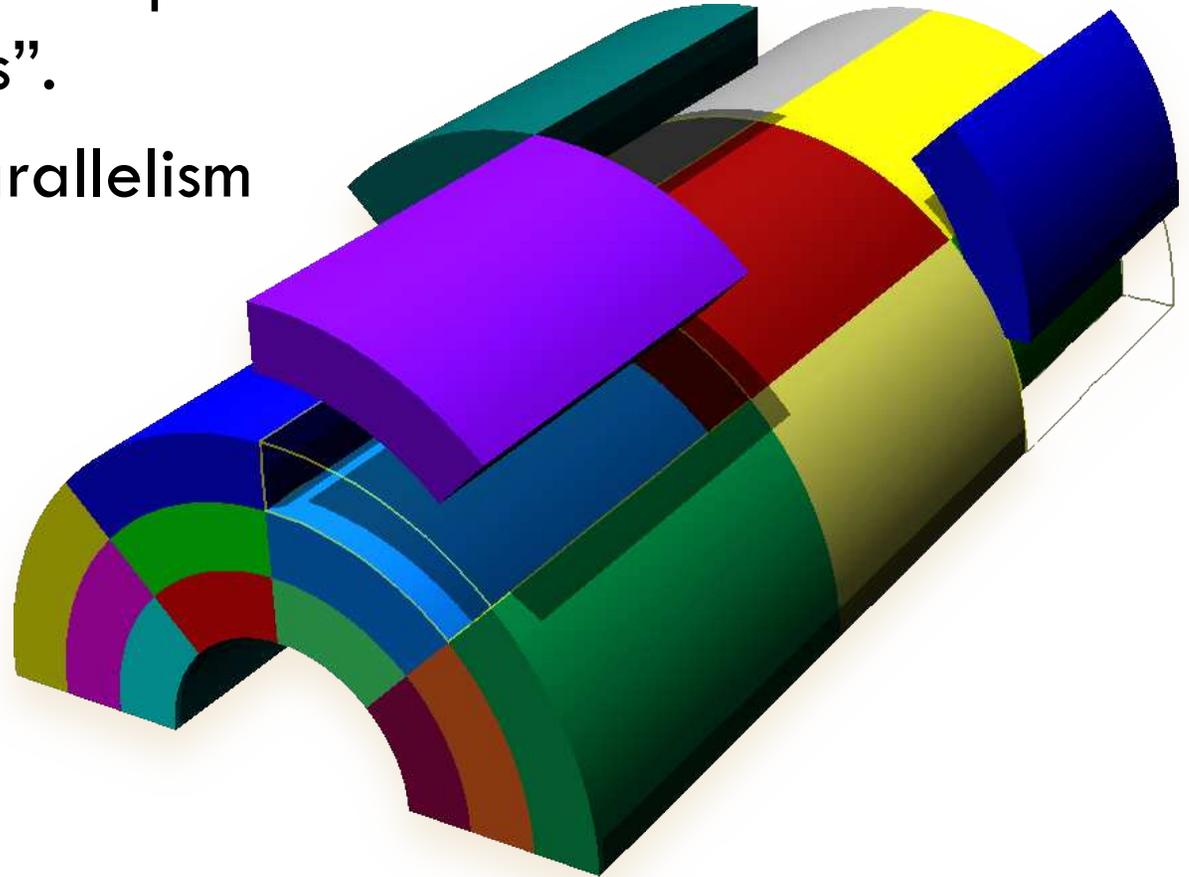
Species



- Similar to materials, describes sub-grid variable composition
- Used for mass fractions
- Generally weights other scalars (e.g. partial pressure)

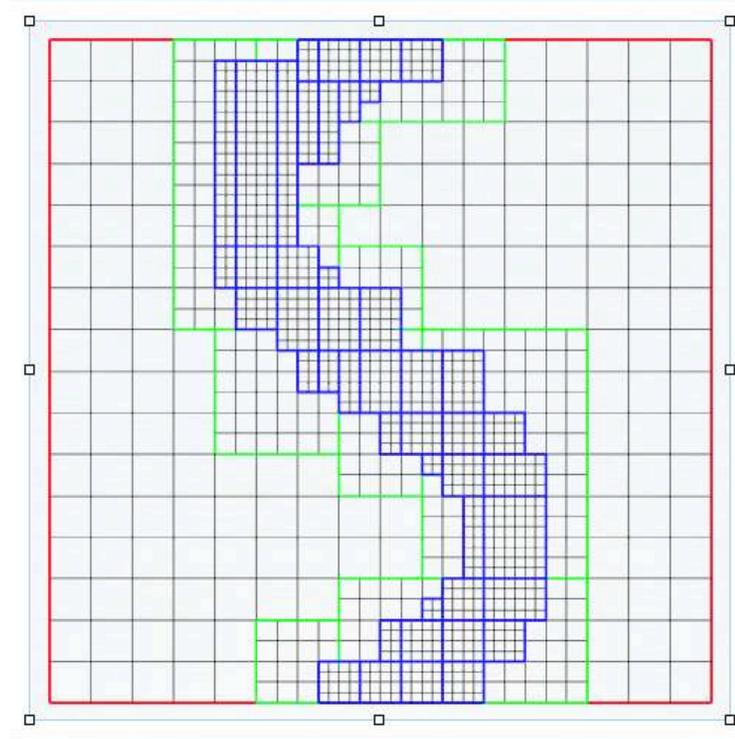
Parallel meshes

- Provides aggregation for meshes
- A mesh may be composed of hundreds of thousands of mesh “blocks”.
- Allows data parallelism



AMR meshes

- Mesh blocks can be associated with patches and levels.
- Allows for aggregation of meshes into AMR hierarchy levels.



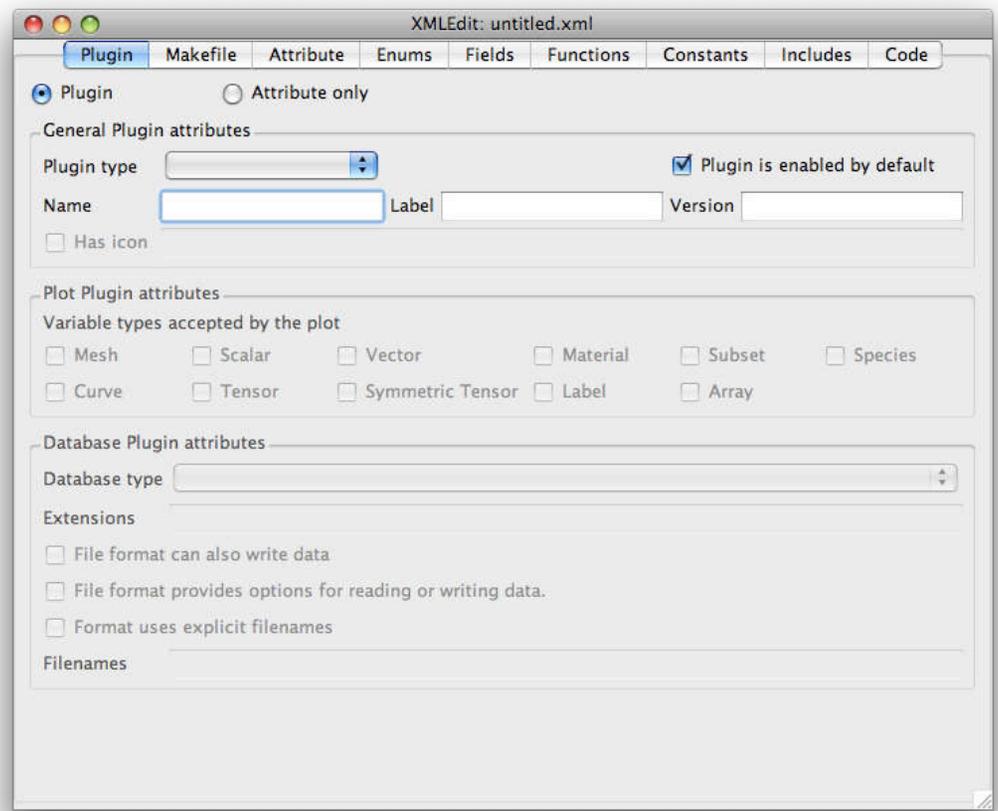
Developing a Database plugin



- Three basic steps:
 - Use xmledit tool to describe basics of reader
 - Use xml2plugin tool to generate Visit “glue” code.
 - Fill in required class methods

xmledit

- GUI tool to edit all information related to plugins
- Used to define type of database, filename extensions, etc.
- Creates XML file that describes your reader



Types of database plugins

- Two axes: domains and time
- Moving down and to the right adds complexity

	Single Domain	Multiple Domain
Single Timestep	STSD	STMD
Multiple Timesteps	MTSD	MTMD

We will make an “IEEE” plugin

- Reads text file of points with data
- Simple STSD database

The image shows a screenshot of the XMLEdit application interface for configuring a plugin named 'IEEE'. The 'Plugin' tab is selected, and the 'Plugin' radio button is chosen. The configuration is as follows:

- General Plugin attributes:**
 - Plugin type: Database
 - Plugin is enabled by default:
 - Name: ieee
 - Label: IEEE Vis
 - Version: (empty)
 - Has icon:
- Plot Plugin attributes:**
 - Variable types accepted by the plot:
 - Mesh:
 - Scalar:
 - Vector:
 - Material:
 - Subset:
 - Species:
 - Curve:
 - Tensor:
 - Symmetric Tensor:
 - Label:
 - Array:
- Database Plugin attributes:**
 - Database type: STSD - Generic single time single domain
 - Extensions: ieee
 - File format can also write data:
 - File format provides options for reading or writing data:
 - Format uses explicit filenames:
 - Filenames: (empty)

In the background, a window titled 'data0000.ieee' displays a text file with the following content:

```
# x y v
4.23753518051 -6.28432838795 0.521763945345
-2.27336977284 1.23918450369 1.42646054811
-4.03918487303 -0.286620182048 0.522498245438
0.620222855503 2.06408008731 0.00514247344
5733223819
46270708
474912551
1343808748
99229709
3437056273
00790499
023275219
634724902132
599040687
114782698
438175926
9648857654
0.892558277144
04051844594
2772994571
9976798792
1480018933
82836195
9612071
```

Generate glue code

- xml2plugin takes your XML file and generates many

```
xterm — xterm
% xml2plugin IEEE.xml
***** running: xmltest IEEE.xml *****
Running: xmltest IEEE.xml
-----
Parsed document of type Plugin
-----
Plugin: ieee ("IEEE Vis", type=database) -- version
Attribute: ()

***** running: xml2atts IEEE.xml *****
Running: xml2atts -noprint IEEE.xml
No attributes to generate for database plugins
***** running: xml2window IEEE.xml *****
Running: xml2window -noprint IEEE.xml
No window to generate for database plugins
***** running: xml2info IEEE.xml *****
Running: xml2info -noprint IEEE.xml
***** running: xml2makefile IEEE.xml *****
Running: xml2makefile -noprint IEEE.xml
***** running: xml2zavt IEEE.xml *****
Running: xml2zavt -noprint IEEE.xml
***** running: xml2python IEEE.xml *****
Running: xml2python -noprint IEEE.xml
No python to generate for database plugins
***** running: xml2java IEEE.xml *****
Running: xml2java -noprint IEEE.xml
No java to generate for database plugins
Mac 11366 2009/IEEE Vis/DatabasePlugin
% /bin/ls -l
total 120
-rw-rw-r-- 1 ahern ahern 440 Oct 11 17:41 IEEE.xml
-rw-rw-r-- 1 ahern ahern 4405 Oct 11 17:41 Makefile
-rw-rw-r-- 1 ahern ahern 11466 Oct 11 17:41 avtieeeFileFormat.C
-rw-rw-r-- 1 ahern ahern 4164 Oct 11 17:41 avtieeeFileFormat.h
-rw-rw-r-- 1 ahern ahern 3879 Oct 11 17:41 ieeeCommonPluginInfo.C
-rw-rw-r-- 1 ahern ahern 3136 Oct 11 17:41 ieeeEnginePluginInfo.C
-rw-rw-r-- 1 ahern ahern 2830 Oct 11 17:41 ieeeMDServerPluginInfo.C
-rw-rw-r-- 1 ahern ahern 5981 Oct 11 17:41 ieeePluginInfo.C
-rw-rw-r-- 1 ahern ahern 4125 Oct 11 17:41 ieeePluginInfo.h
Mac 11367 2009/IEEE Vis/DatabasePlugin
% █
```

files

General Info

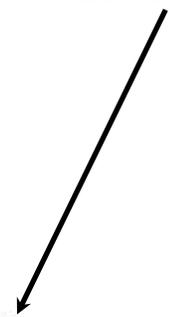
Makefile

“Glue”

Attribute: ()

```
***** running: xml2atts IEEE.xml *****
Running: xml2atts -noprint IEEE.xml
No attributes to generate for database plugins
***** running: xml2window IEEE.xml *****
Running: xml2window -noprint IEEE.xml
No window to generate for database plugins
***** running: xml2info IEEE.xml *****
Running: xml2info -noprint IEEE.xml
***** running: xml2makefile IEEE.xml *****
Running: xml2makefile -noprint IEEE.xml
***** running: xml2avt IEEE.xml *****
Running: xml2avt -noprint IEEE.xml
***** running: xml2python IEEE.xml *****
Running: xml2python -noprint IEEE.xml
No python to generate for database plugins
***** running: xml2java IEEE.xml *****
Running: xml2java -noprint IEEE.xml
No java to generate for database plugins
Mac 11366 2009/IEEE Vis/DatabasePlugin
% /bin/ls -l
total 120
-rw-rw-r-- 1 ahern ahern 440 Oct 11 17:41 IEEE.xml
-rw-rw-r-- 1 ahern ahern 4405 Oct 11 17:41 Makefile
-rw-rw-r-- 1 ahern ahern 11466 Oct 11 17:41 avtieeeFileFormat.C
-rw-rw-r-- 1 ahern ahern 4164 Oct 11 17:41 avtieeeFileFormat.h
-rw-rw-r-- 1 ahern ahern 3879 Oct 11 17:41 ieeeCommonPluginInfo.C
-rw-rw-r-- 1 ahern ahern 3136 Oct 11 17:41 ieeeEnginePluginInfo.C
-rw-rw-r-- 1 ahern ahern 2830 Oct 11 17:41 ieeeMDServerPluginInfo.C
-rw-rw-r-- 1 ahern ahern 5981 Oct 11 17:41 ieeePluginInfo.C
-rw-rw-r-- 1 ahern ahern 4125 Oct 11 17:41 ieeePluginInfo.h
Mac 11367 2009/IEEE Vis/DatabasePlugin
% █
```

“Glue”



```
avtieceeFileFormat.C
avtieceeFileFormat.C:1
// *****
// Method: avtieceeFileFormat con
//
// Programmer: ahern -- generate
// Creation: Sun Oct 11 17:41:
//
// *****
avtieceeFileFormat::avtieceeFileFormat(const char *filename)
: avtSTSDFileFormat(filename)
{ ... }

// *****
// Method: avtieceeFileFormat::FreeUpResources
//
// Purpose:
// When VisIt is done focusing on a particular timestep, it asks that
// timestep to free up any resources (memory, file descriptors) that
// it has associated with it. This method is the mechanism for doing
// that.
//
// Programmer: ahern -- generated by xml2avt
// Creation: Sun Oct 11 17:41:51 PST 2009
//
// *****
void
avtieceeFileFormat::FreeUpResources(void)
{ ... }

// *****
// Method: avtieceeFileFormat::PopulateDatabaseMetaData
//
// Purpose:
// This database meta-data object is like a table of contents for the
// file. By populating it, you are telling the rest of VisIt what
// information it can request from you.
//
// Programmer: ahern -- generated by xml2avt
// Creation: Sun Oct 11 17:41:51 PST 2009
//
// *****
void
avtieceeFileFormat::PopulateDatabaseMetaData(avtDatabaseMetaData *md)
{ ... }

// *****
// Method: avtieceeFileFormat::GetMesh
//
// Purpose:
```

- avtieceeFileFormat::avtieceeFileFormat(const char *filename)
- avtieceeFileFormat::FreeUpResources(void)
- avtieceeFileFormat::PopulateDatabaseMetaData(avtDatabaseMetaData *md)
- avtieceeFileFormat::GetMesh(const char *meshname)
- avtieceeFileFormat::GetVar(const char *varname)
- avtieceeFileFormat::GetVectorVar(const char *varname)

```
avtieceeFileFormat.C
avtieceeFileFormat.C:71  <No selected symbol>

//
// Programmer: ahern -- generated by xml2avt
// Creation:   Sun Oct 11 17:41:51 PST 2009
//
// *****

void
avtieceeFileFormat::PopulateDatabaseMetaData(avtDatabaseMetaData *md)
{
    //
    // CODE TO ADD A MESH
    //
    // string meshname = ...
    //
    // AVT_RECTILINEAR_MESH, AVT_CURVILINEAR_MESH, AVT_UNSTRUCTURED_MESH,
    // AVT_POINT_MESH, AVT_SURFACE_MESH, AVT_UNKNOWN_MESH
    // avtMeshType mt = AVT_RECTILINEAR_MESH;
    //
    // int nblocks = 1; <-- this must be 1 for STSD
    // int block_origin = 0;
    // int spatial_dimension = 2;
    // int topological_dimension = 2;
    // double *extents = NULL;
    //
    // Here's the call that tells the meta-data object that we have a mesh:
    //
    // AddMeshToMetaData(md, meshname, mt, extents, nblocks, block_origin,
    //                   spatial_dimension, topological_dimension);
    //
    //
    // CODE TO ADD A SCALAR VARIABLE
    //
    // string mesh_for_this_var = meshname; // ??? -- could be multiple meshes
    // string varname = ...
    //
    // AVT_NODECENT, AVT_ZONECENT, AVT_UNKNOWN_CENT
    // avtCentering cent = AVT_NODECENT;
    //
    // Here's the call that tells the meta-data object that we have a var:
    //
    // AddScalarVarToMetaData(md, varname, mesh_for_this_var, cent);
    //
    //
    // CODE TO ADD A VECTOR VARIABLE
    //
    // string mesh_for_this_var = meshname; // ??? -- could be multiple meshes
    // string varname = ...
    // int vector_dim = 2;
    //
    // AVT_NODECENT, AVT_ZONECENT, AVT_UNKNOWN_CENT
    // avtCentering cent = AVT_NODECENT;
    //
    //

```

Implement your par

- You need to fill in:
 - Constructor/Destructor
 - PopulateDatabaseMeta
 - GetMesh
 - GetVar (can be empty, k
 - GetVectorVar (can be er
- You need to write:
 - Code to read the file
- Add class members as nece

```
void
avtieceeFileFormat::ReadData()
{
    if (fileRead)
        return;

    // Read the header if we haven't already;
    ReadHeader();

    // Process all lines, pulling out the coordinate fields.
    vector<loat> x;
    vector<loat> y;
    vector<loat> v;
}

// Method: avtieceeFileFormat::PopulateDatabaseMetaData
// Purpose:
// This database meta-data object is like a table of contents for the
// file. By populating it, you are telling the rest of VisIt what
// information it can request from you.
// Programmer: ohern -- generated by xml2avt
// Creation: Sun Oct 11 16:42:45 PST 2009
// *****

void
avtieceeFileFormat::PopulateDatabaseMetaData(avtDatabaseMetaData *md)
{
    ReadHeader();

    avtMeshMetaData *mesh = new avtMeshMetaData;
    mesh->name = "mesh";
    mesh->meshType = AVT_POINT_MESH;
    mesh->blockOrigin = 0;
    mesh->spatialDimension = 2;
    mesh->topologicalDimension = 0;
    mesh->hasSpatialExtents = false;
    md->Add(mesh);

    if (varFound)
    {
        AddScalarVarToMetaData(md, varname, "mesh", AVT_NODECENT, NULL);
    }
}

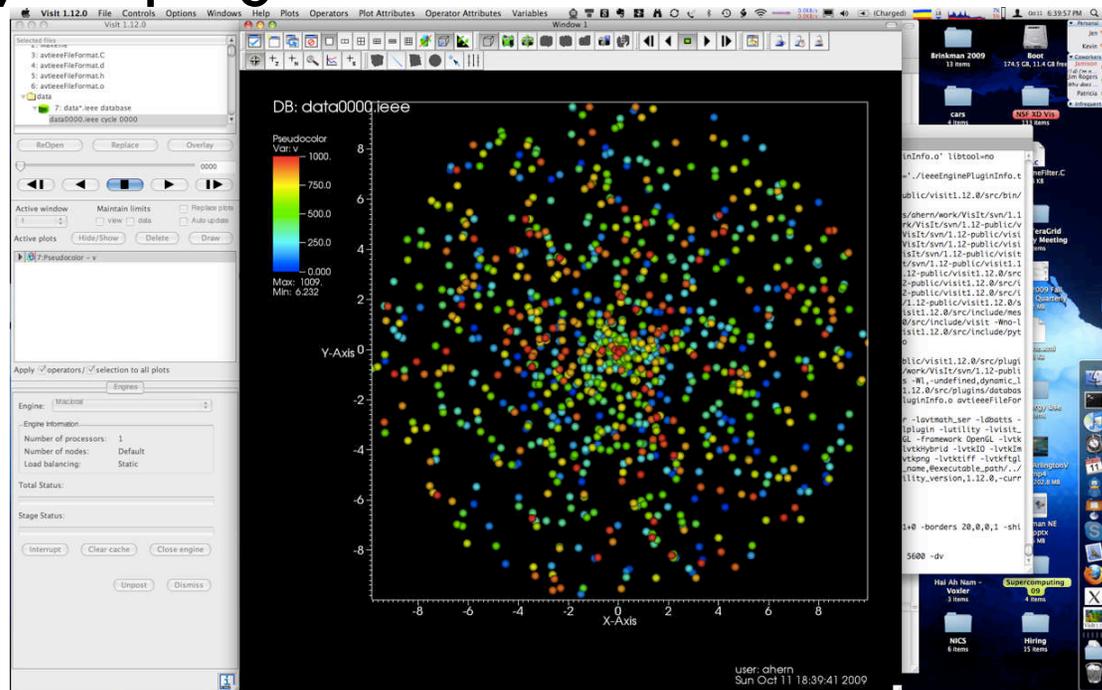
if (varFound)
{
    var = vtkFloatArray::New();
    var->SetNumberOfTuples(npts);
    for (int i = 0; i < npts; i++)
        var->SetTuple1(i, v[i]);
}

// Success!
//cerr << x.size() << " points of data read and grid created." << endl;
fileRead = true;
}

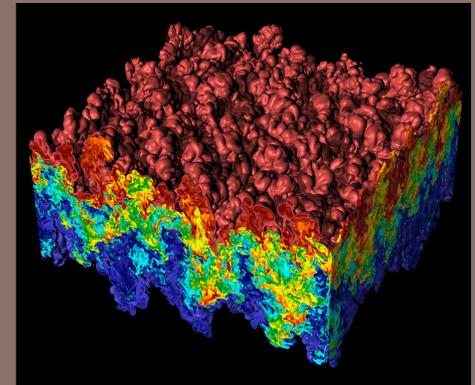
// *****
```

Build, install, and run

- Makefile will automatically put plugin in your $\sim/.visit/plugins$ directory or in the public location if you desire.
- VisIt will load your plugin at launch



How can we better understand data?

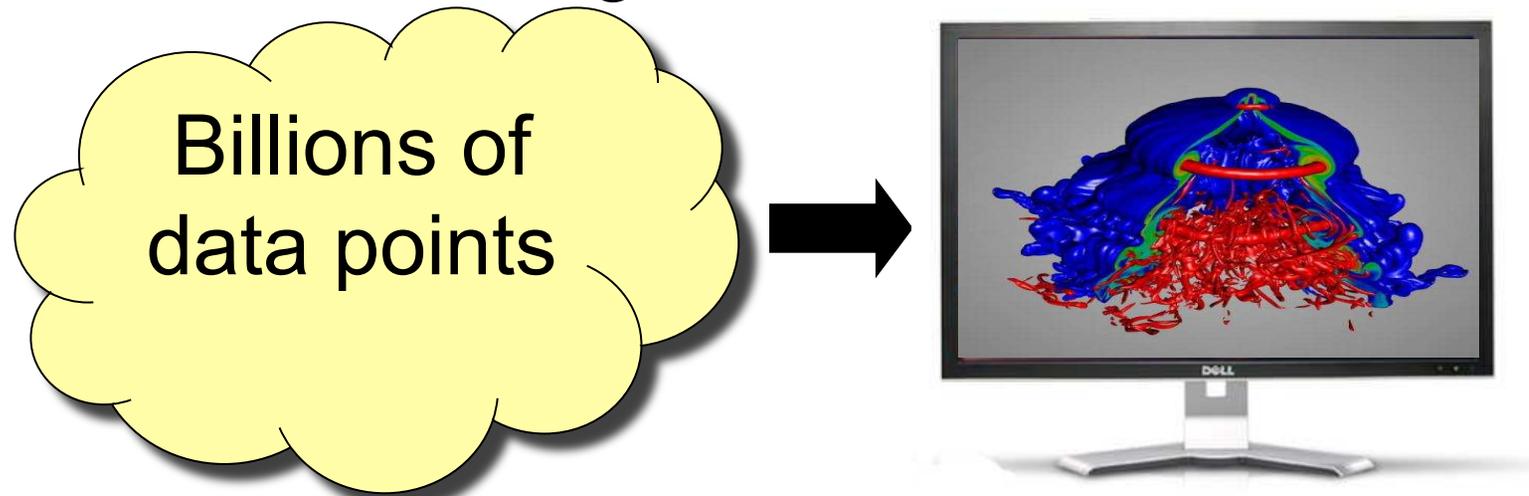


June 13, 2011

Hank Childs, Lawrence Berkeley Lab & UC Davis

Visualization works because it uses the brain's highly effective visual processing system.

But is this still a good idea at extreme scale?

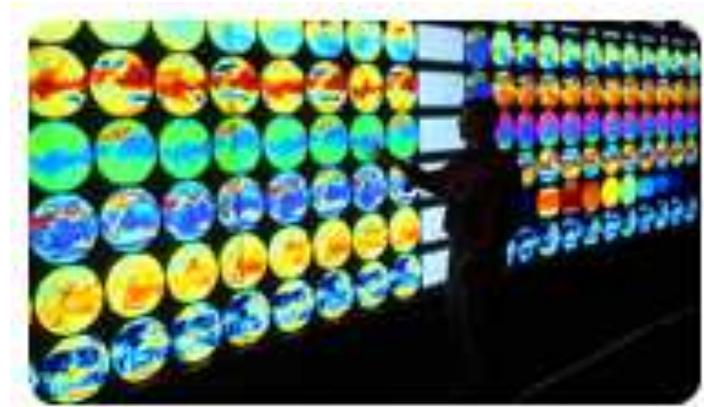
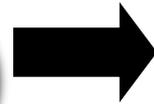
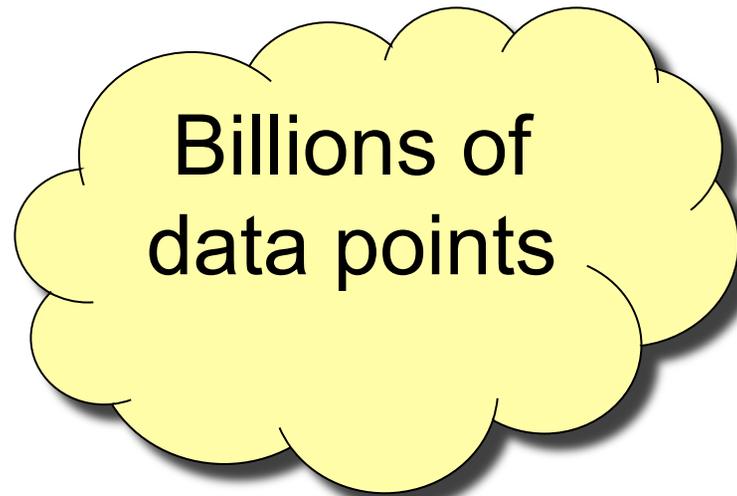


- (Note that visualization is often reducing the data ... so we are frequently **not** trying to render all of the data points.)

Millions of pixels

Visualization works because it uses the brain's highly effective visual processing system.

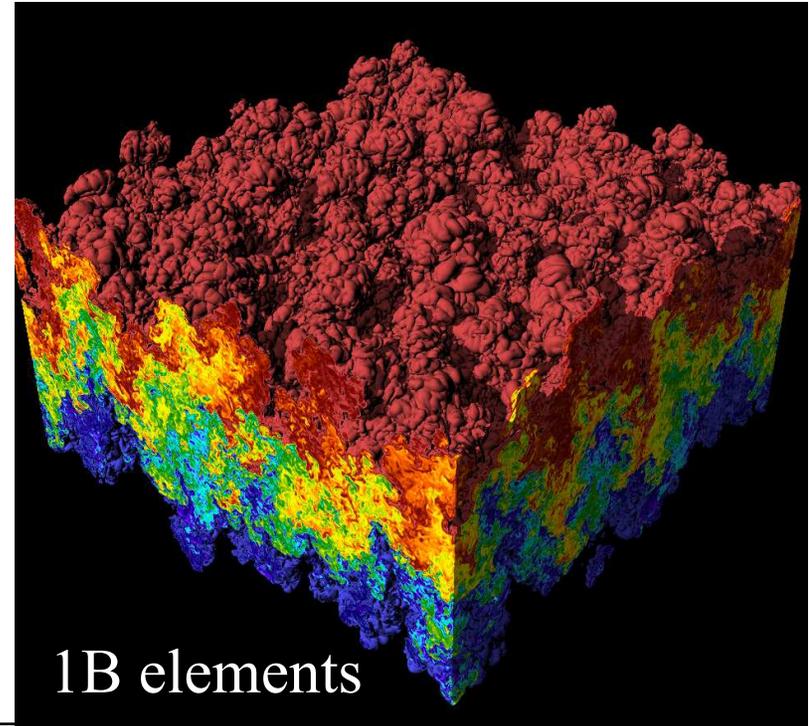
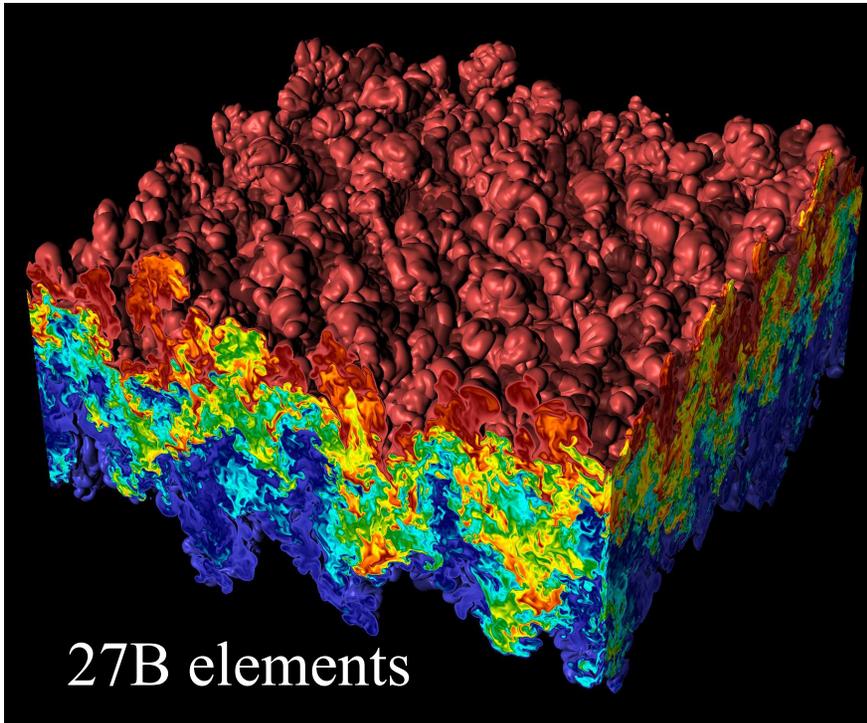
One idea: add more pixels!



35M pixel powerwall

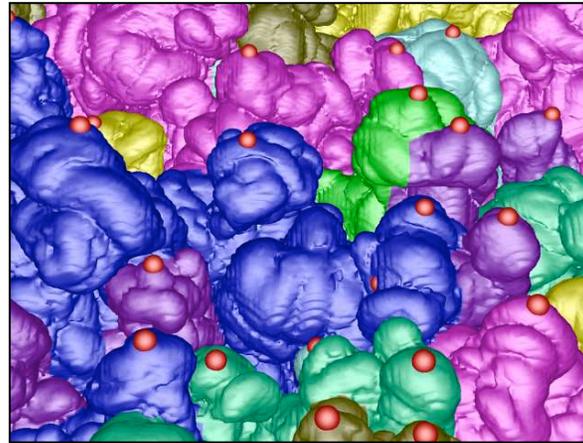
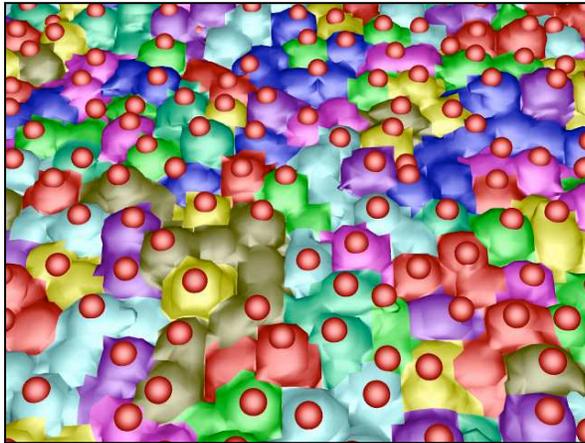
- Bonus: big displays act as collaboration centers.
- But rendering so many pixels is hard ... cannot simply use a GPU

Increased resolution often leads to small but important differences

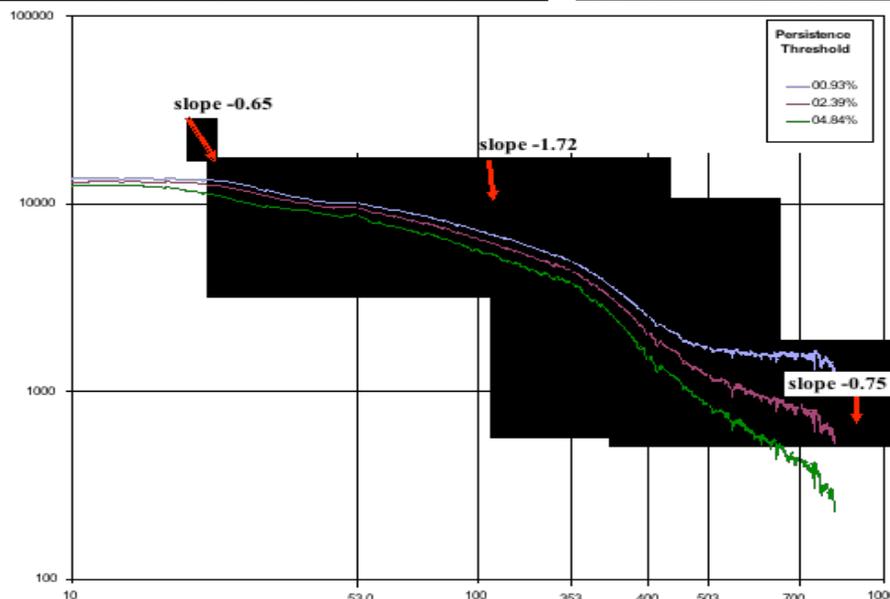


**These differences are normally hard to see.
The best solution is often to add quantitative
metrics.**

Topological analysis was used to count and compare the number of bubbles.

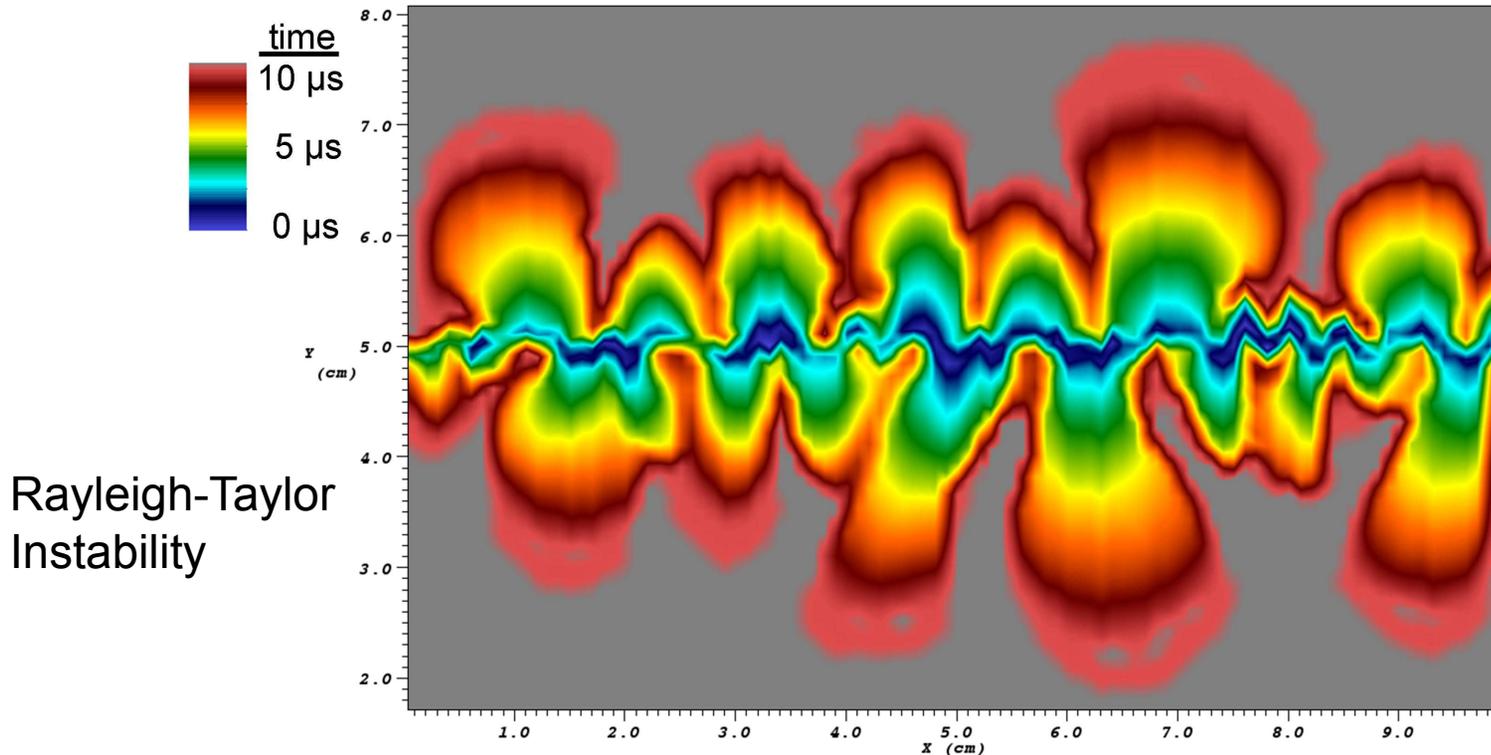


Credit:
Daniel Laney, Timo
Bremer, Valerio
Pascucci, et al.

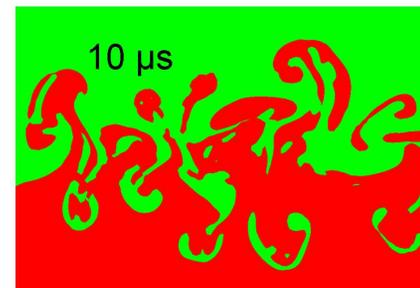
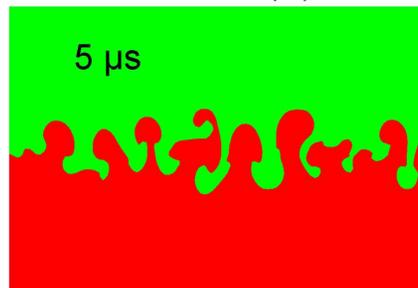
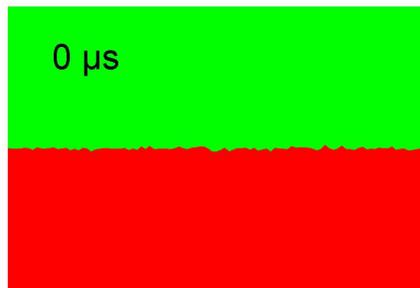


Can we build metrics
like this for the many
scientific problems we
are interested in?

Comparative techniques have applications to better visualization of time-varying data.



Rayleigh-Taylor Instability



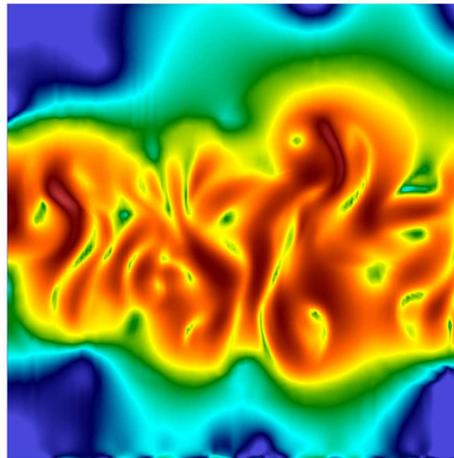
Comparative techniques have applications with parameter studies/ensembles

Studying 25 Rayleigh-Taylor Instability calculations (all at 10 μ s)

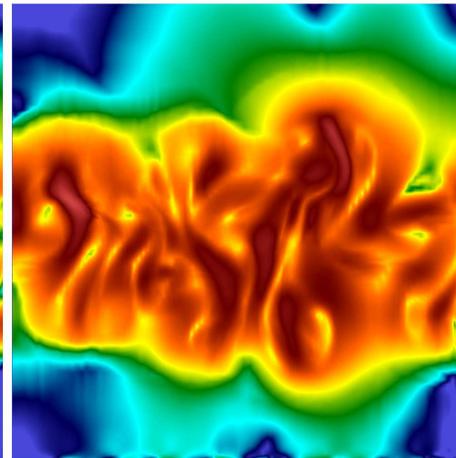
Two “knobs”: turbulent viscosity coefficient, buoyancy coefficient

Five values for each knob, 25 pairs total

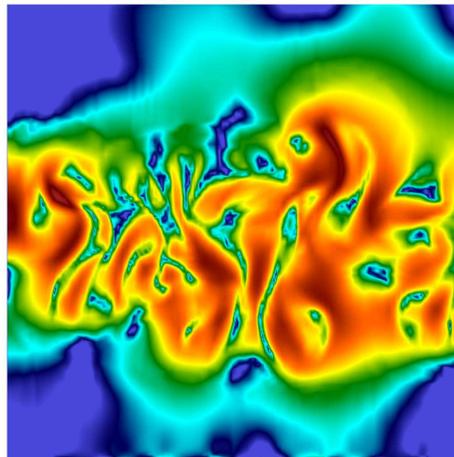
Average Speed
over all 25



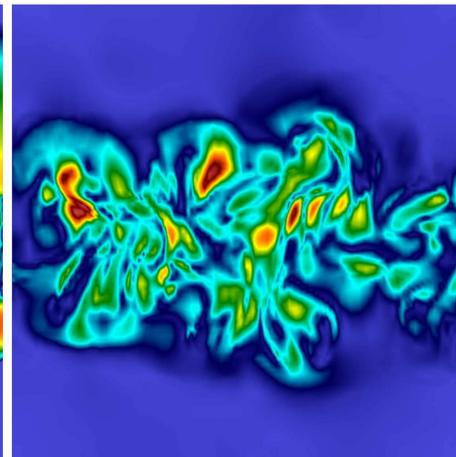
Max Speed
over all 25



Min Speed
over all 25

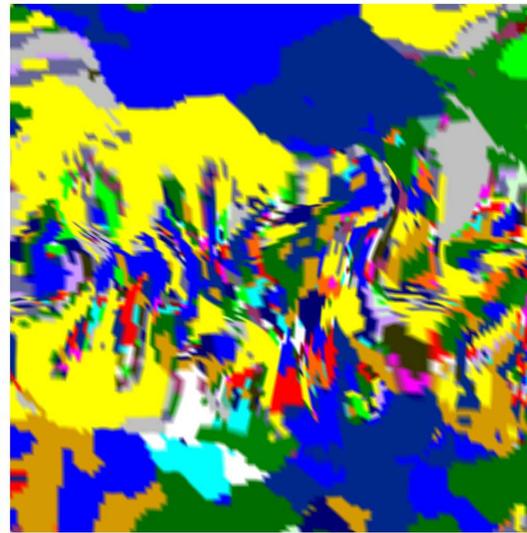
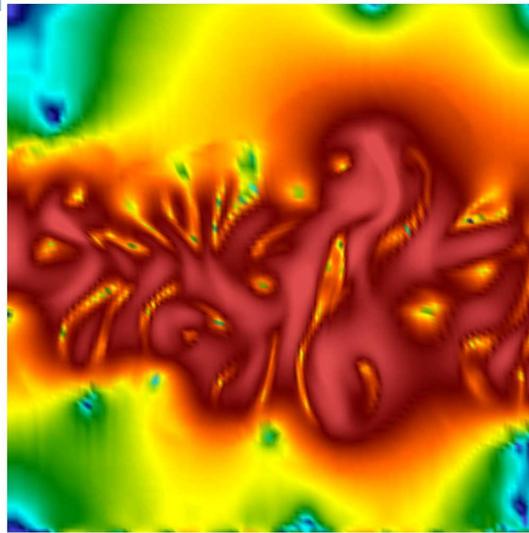


Biggest
difference
over all 25
(is this uncertainty
quantification?)



Comparative techniques have applications with parameter studies/ensembles

Speed for one simulation

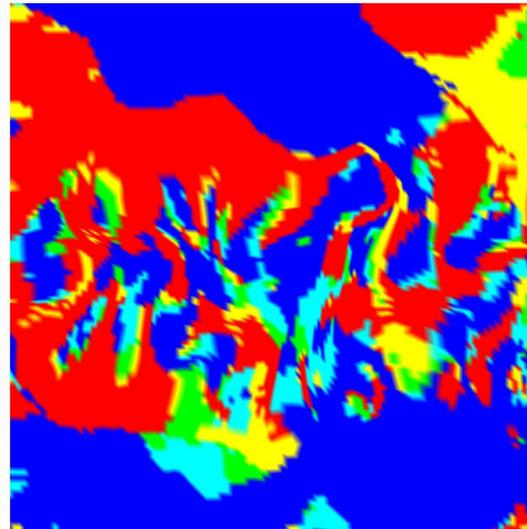
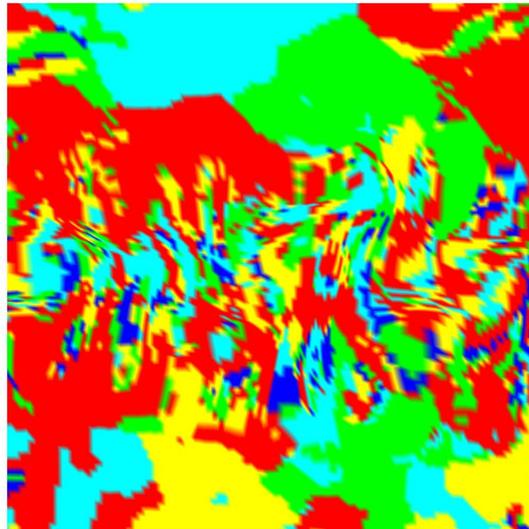


Coloring by Simulation ID with maximum speed

- $K0=V0, K1=V0 \rightarrow$
- $K0=V0, K1=V1 \rightarrow$
- ...
- $K0=V4, K1=V4 \rightarrow$

Coloring by "Knob 0" (buoyancy) with maximum speed

- $K0=V0 \rightarrow$
- $K0=V1 \rightarrow$
- $K0=V2 \rightarrow$
- $K0=V3 \rightarrow$
- $K0=V4 \rightarrow$



Coloring by "Knob 1" (viscosity) with maximum speed

- $K1=V0 \rightarrow$
- $K1=V1 \rightarrow$
- $K1=V2 \rightarrow$
- $K1=V3 \rightarrow$
- $K1=V4 \rightarrow$

Summary



- The purpose of these slides was to show more than parallel isosurfacing...