Experiments in Pure Parallelism

Dave Pugmire, ORNL
Hank Childs, LBNL/ UC Davis
Brad Whitlock, LLNL
Mark Howison, LBNL
Prabhat, LBNL
Sean Ahern, ORNL
Gunther Weber, LBNL
Wes Bethel LBNL

The story behind this work.....

Deliver **production** visualization tools to researchers

- Multidiscipline
- Multi-institutional
- Petascale applications
- Running on the largest machines in the world

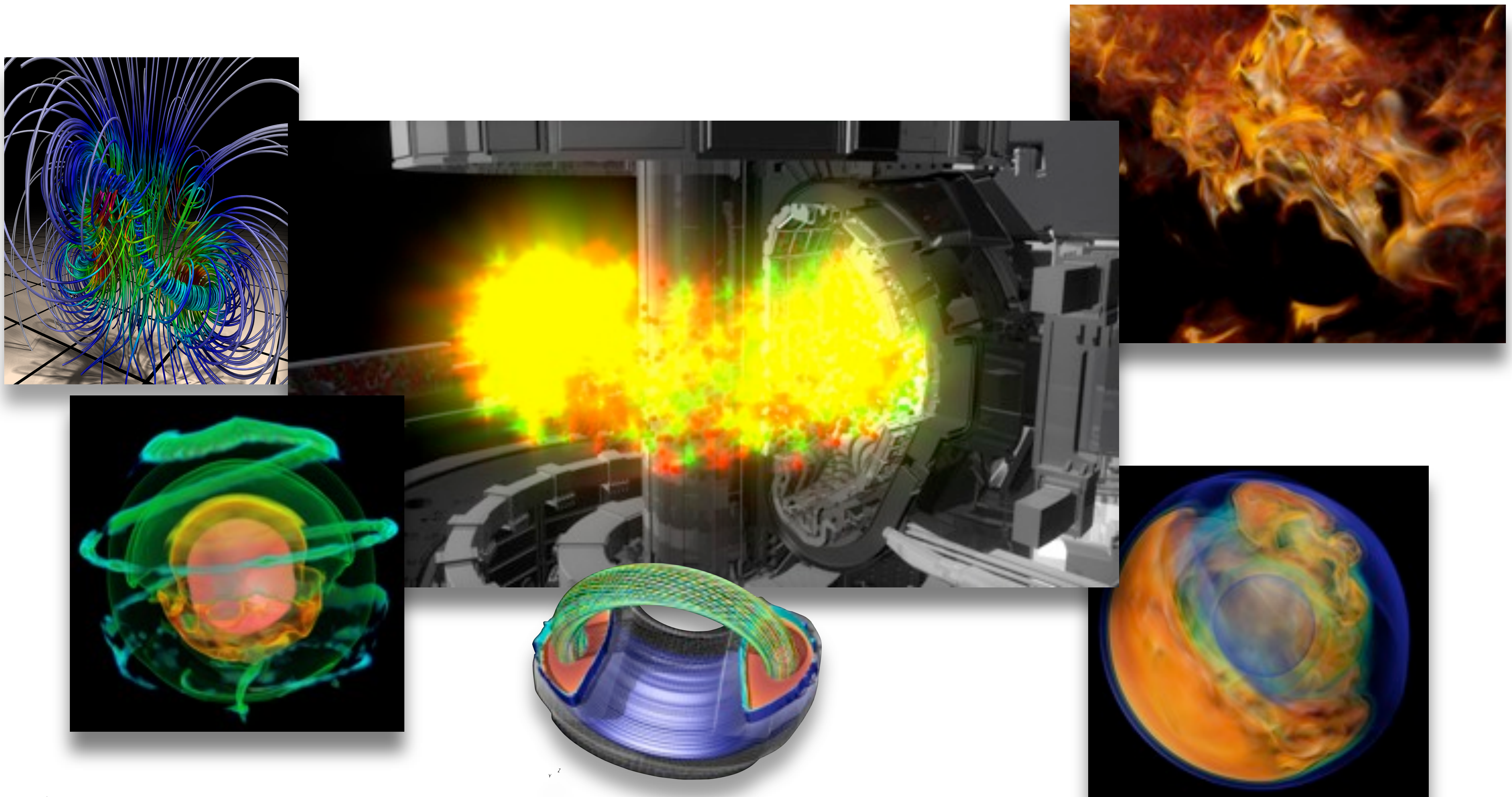
Outline

➡ **Motivation and Introduction**

- Weak scaling study: The Joule Metric
- Scalability on exascale datasets
- Final thoughts

Motivation

- The landscape is going to dramatically change

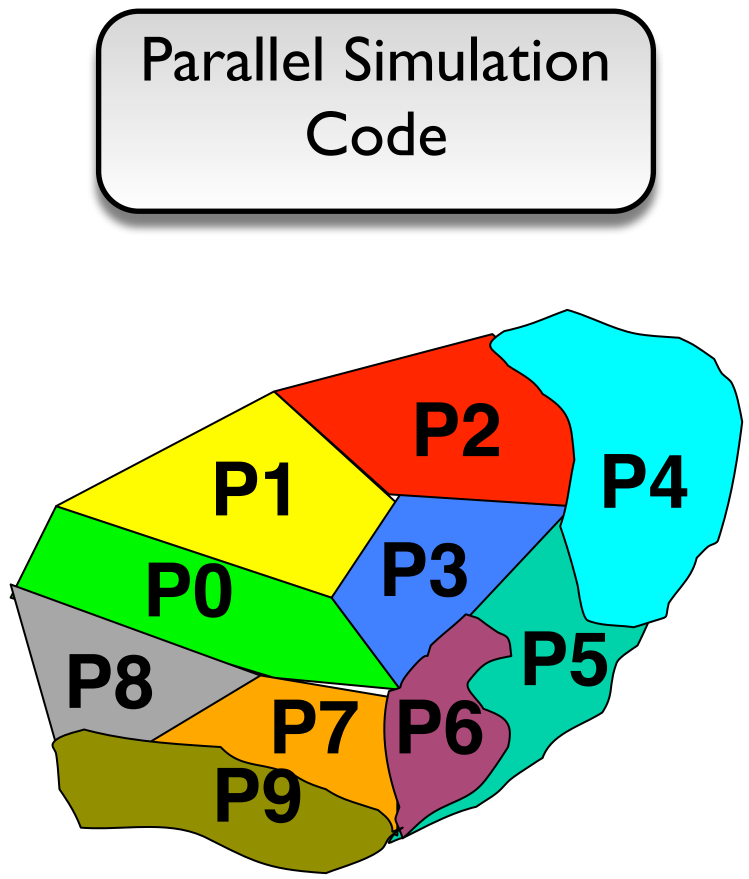


- Questions:
  - Are we ready for exascale simulations?
  - How far can we push production visualization tools of today?
  - What bottlenecks will we encounter?

Pure parallelism

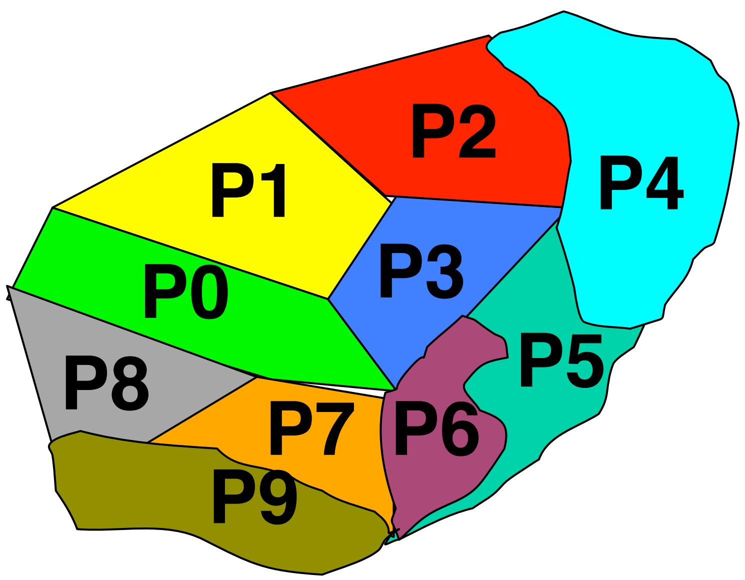
- Pure parallelism is data-level parallelism, but…
  - Multi-resolution can be data-level parallelism
  - Out-of-core can be data-level parallelism
- Pure parallelism: "brute force" … processing full resolution data using data-level parallelism
- Pros:
  - Easy to implement
- Cons:
  - Requires large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

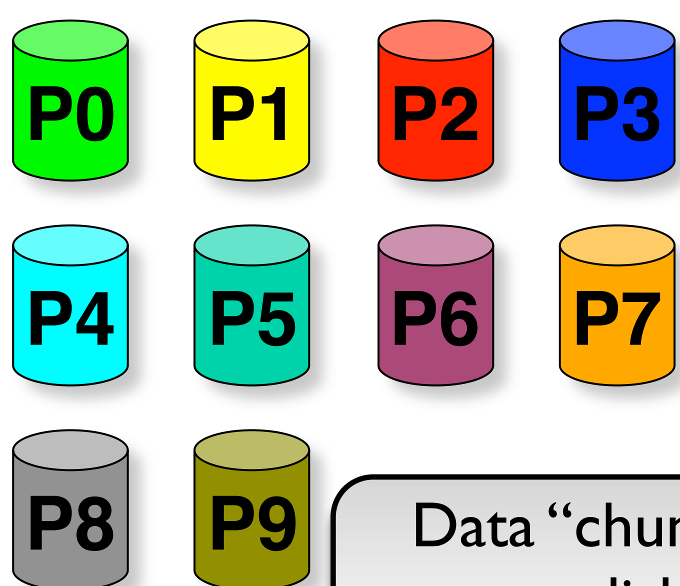
Pure parallelism



Pure parallelism

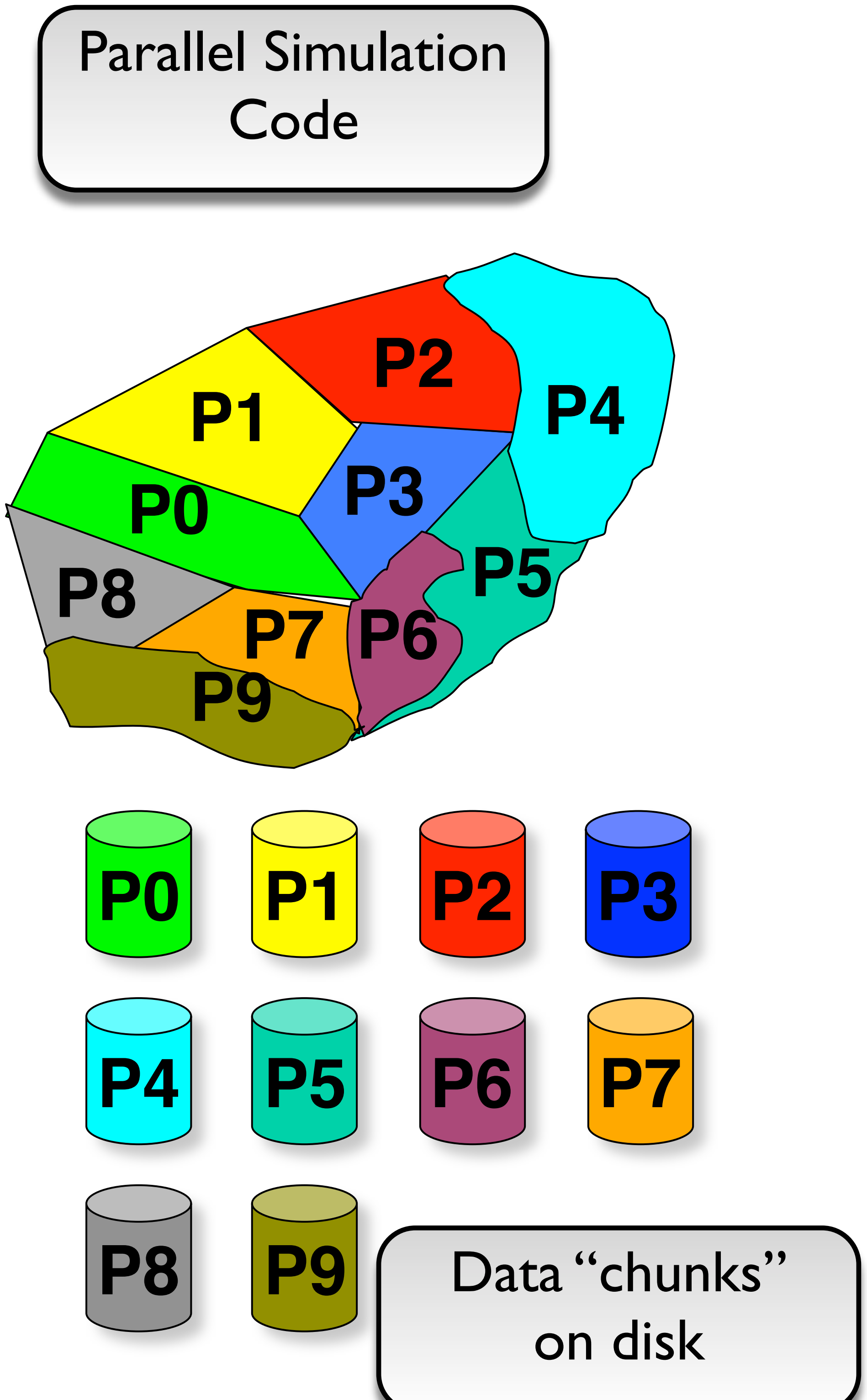
Parallel Simulation Code

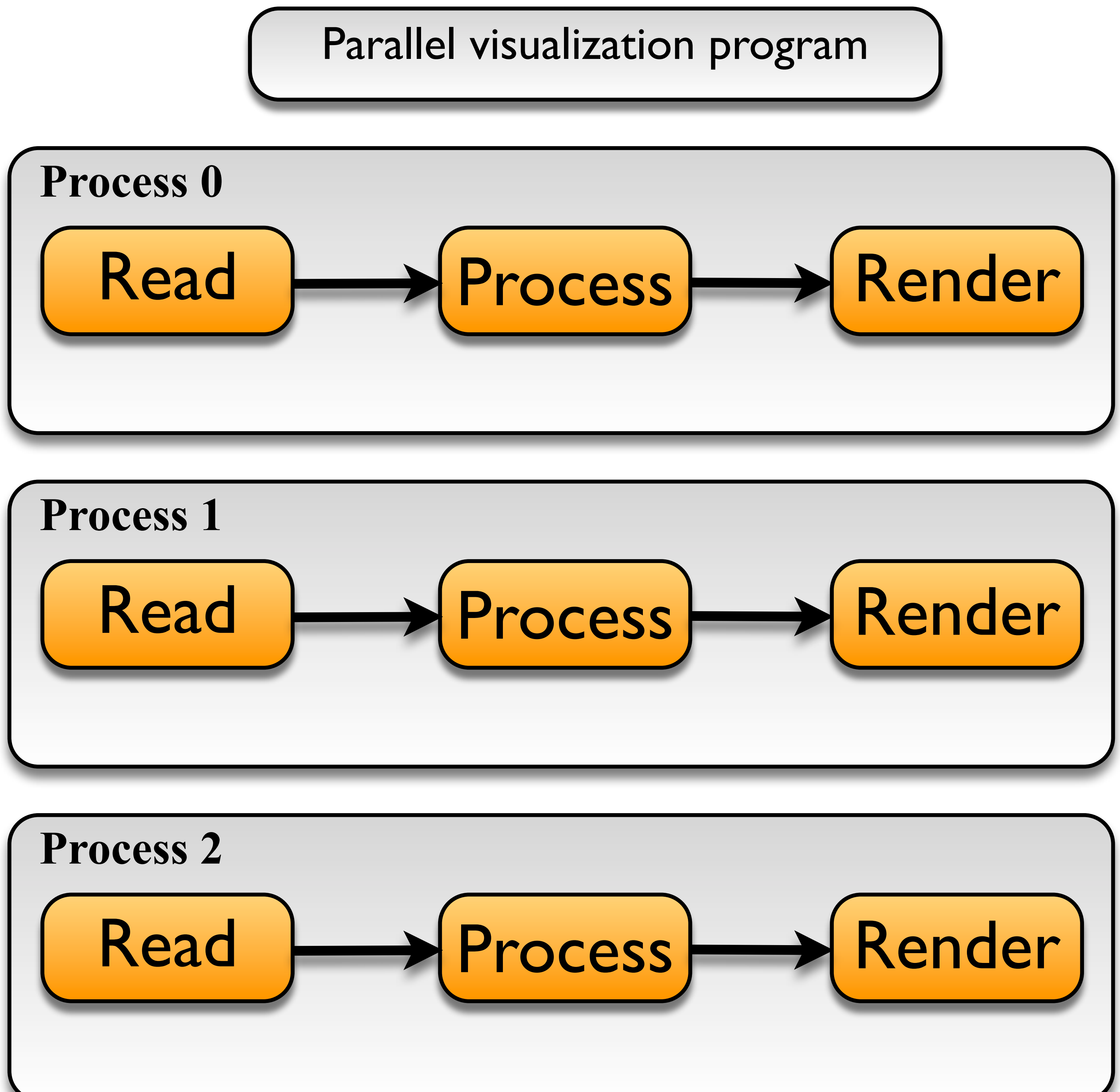




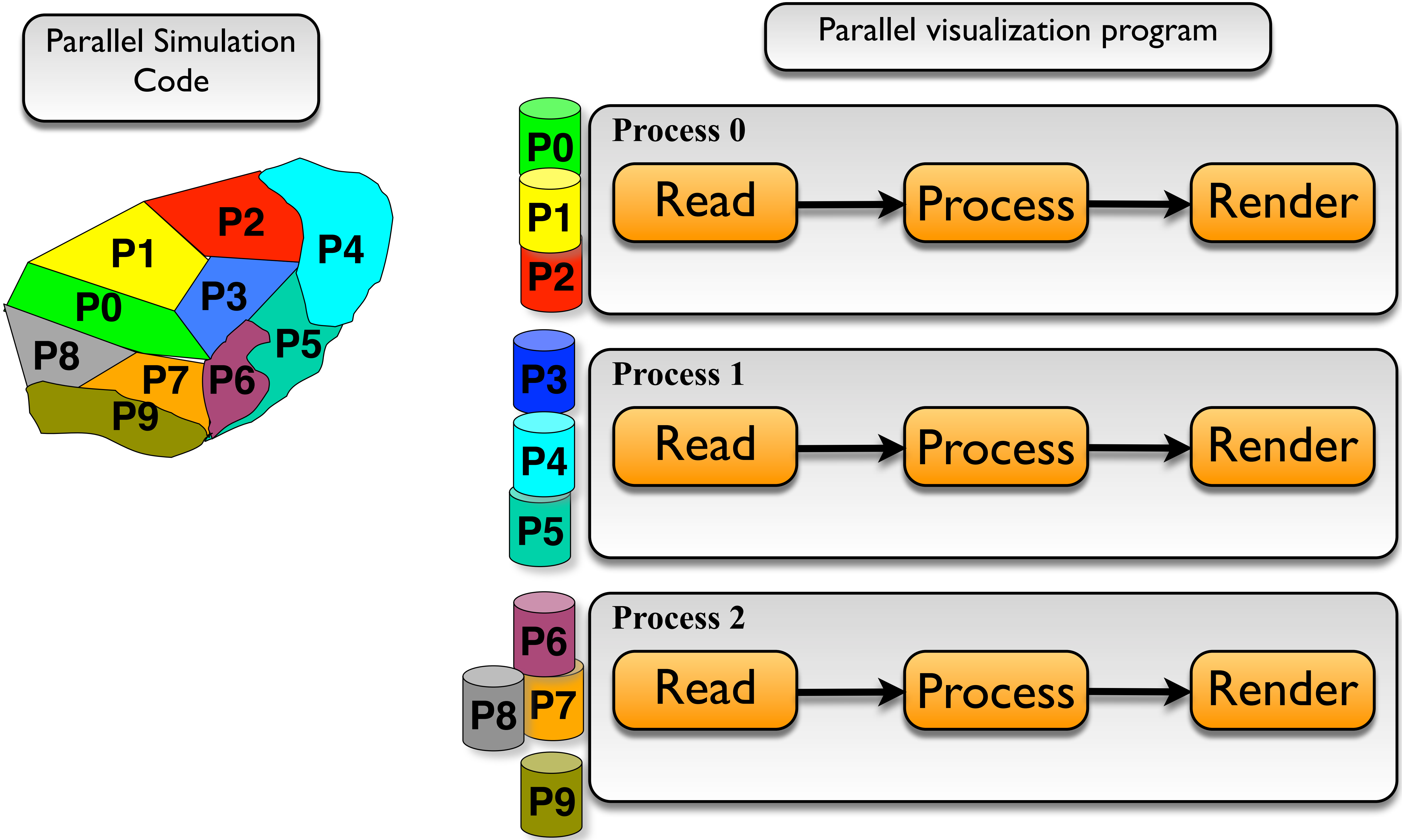
Data "chunks" on disk

Pure parallelism

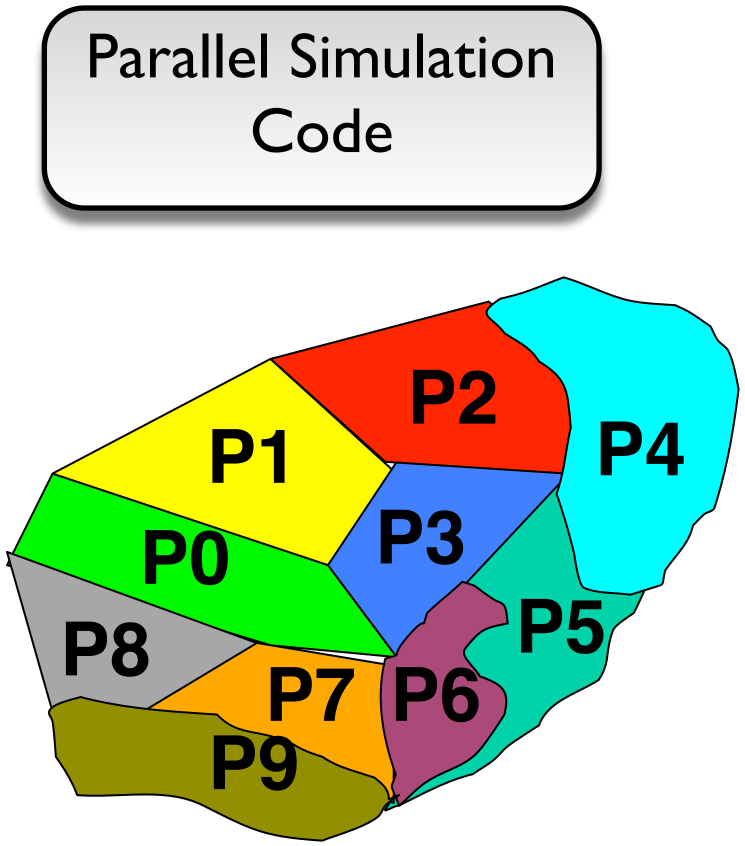


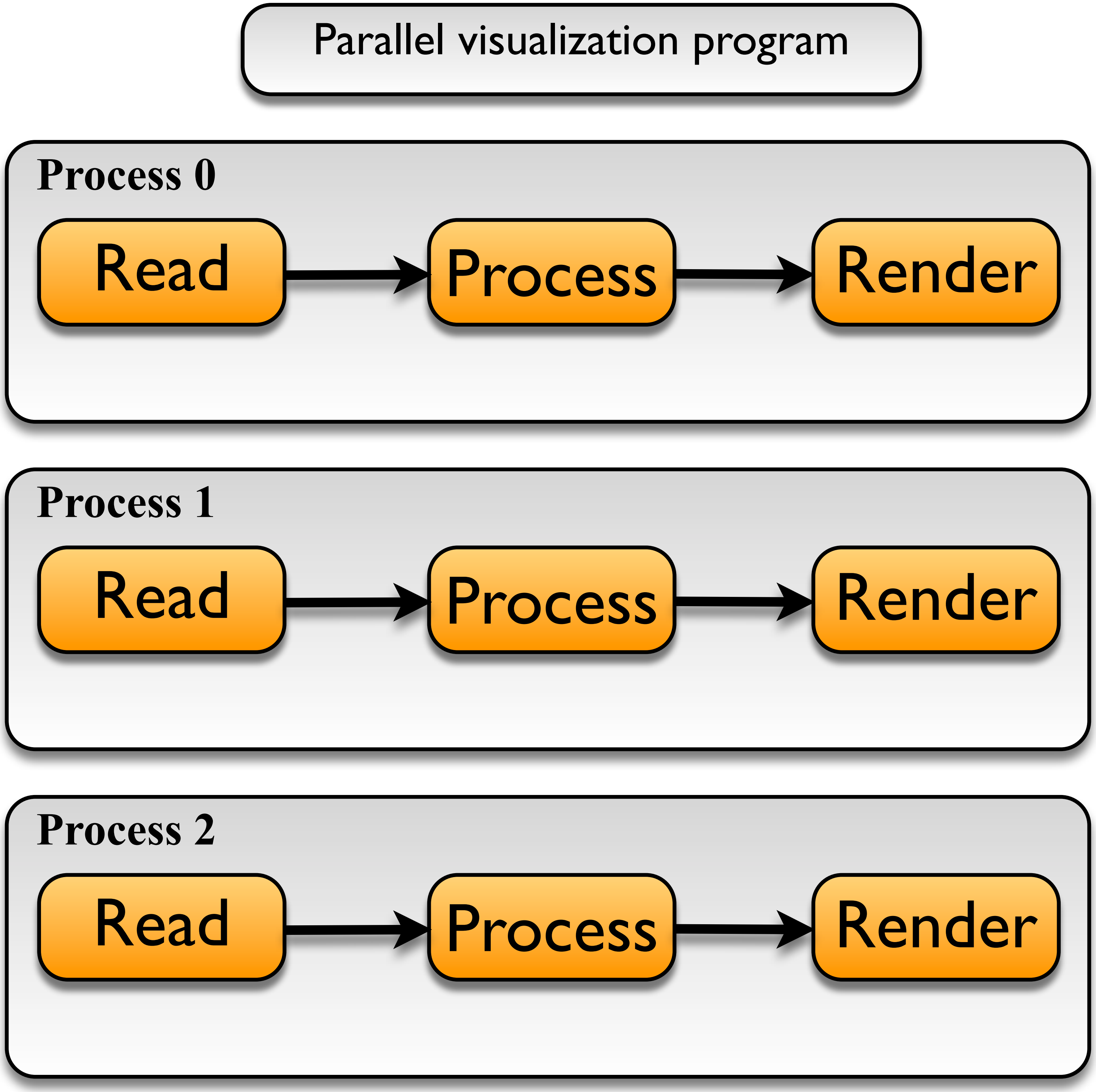


Pure parallelism

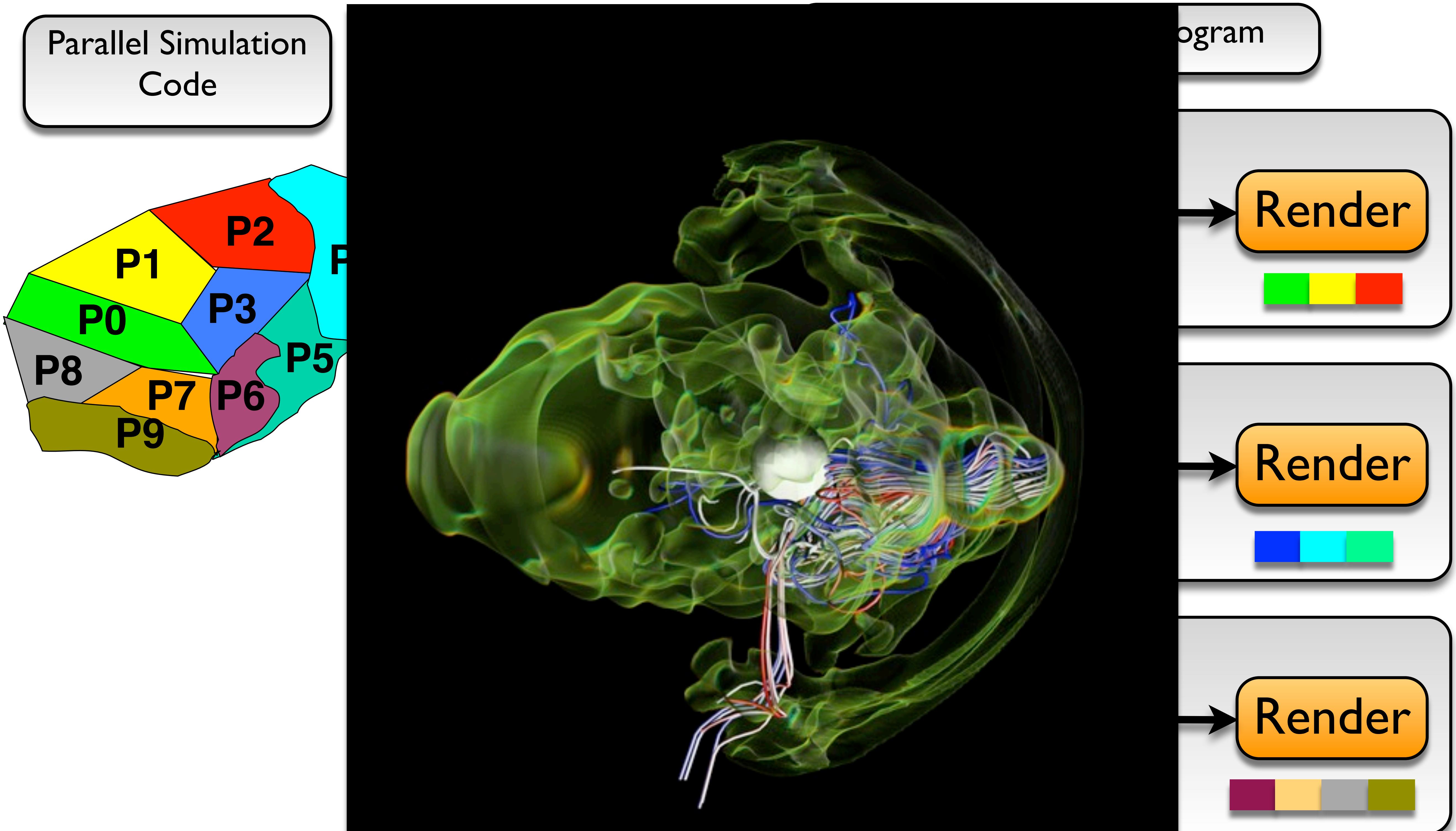


Pure parallelism





Pure parallelism



Outline

- ✓ Motivation and Introduction
- ➡ **Weak scaling study: The Joule Metric**
- Scalability on exascale datasets
- Final thoughts

Scalability of scalar field visualization algorithms

The **Joule** Metric:

DOE Office of Science metric for tracking the efficiency of use of HPC resources

Demonstrate weak scaling by Q4 of a Q2 baseline problem

2009 Joule codes:

Scalability of scalar field visualization algorithms

The **Joule** Metric:

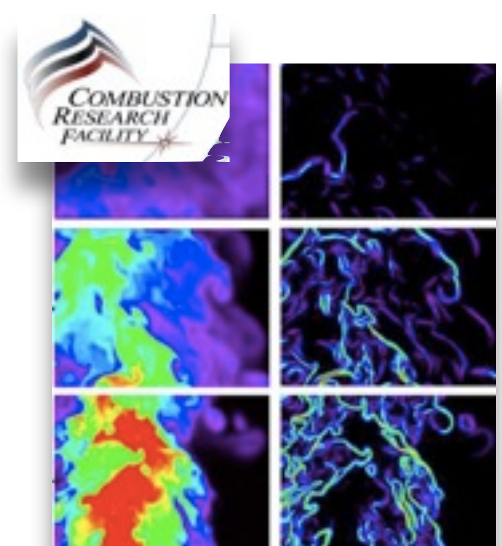
DOE Office of Science metric for tracking the efficiency of use of HPC resources

Demonstrate weak scaling by Q4 of a Q2 baseline problem

2009 Joule codes:

XGC1

CAM

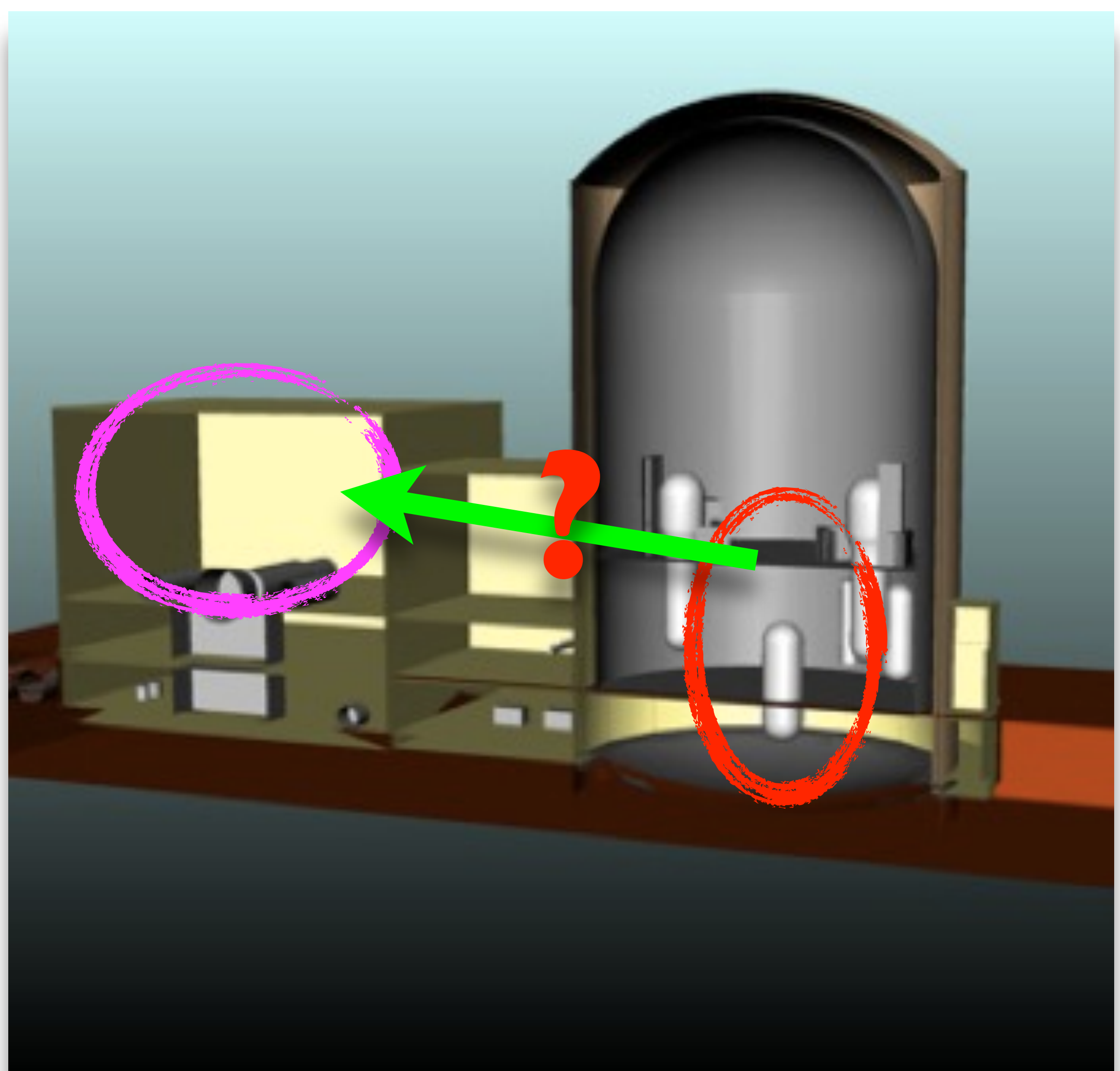


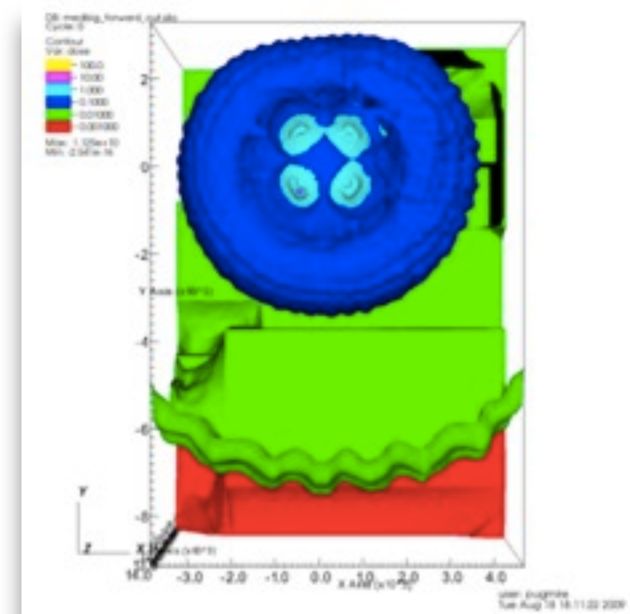
Raptor

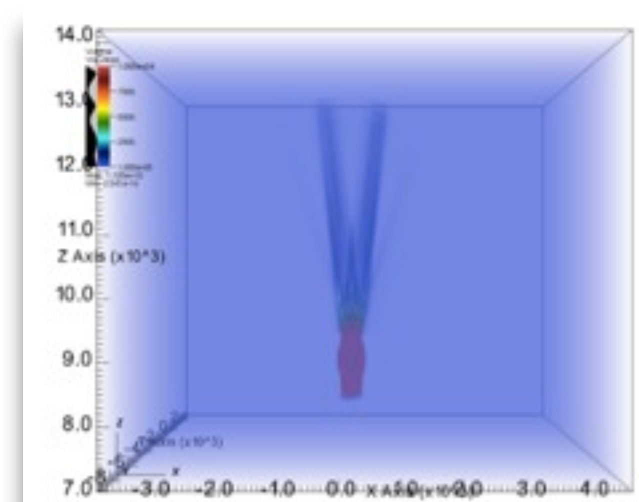
VisIt

Joule and VisIt

Isocontouring and volume rendering of a radiation transport code, Denovo

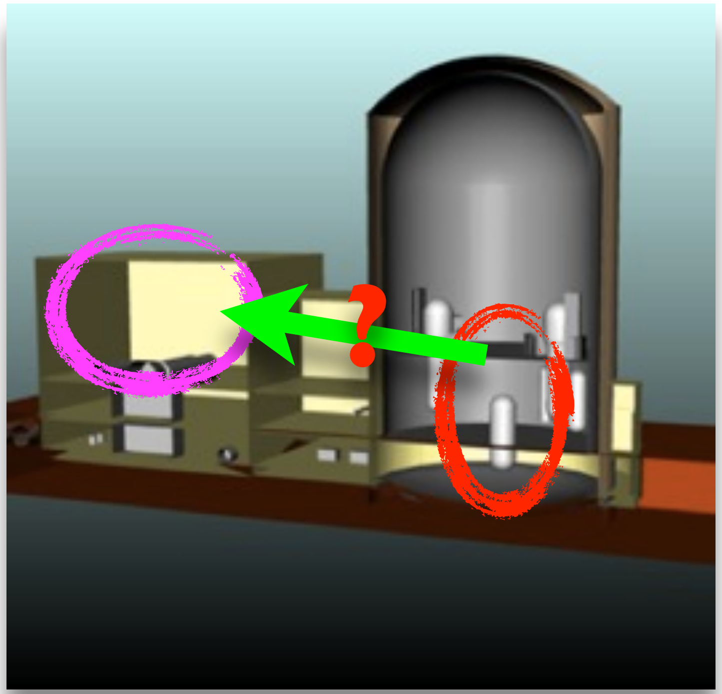




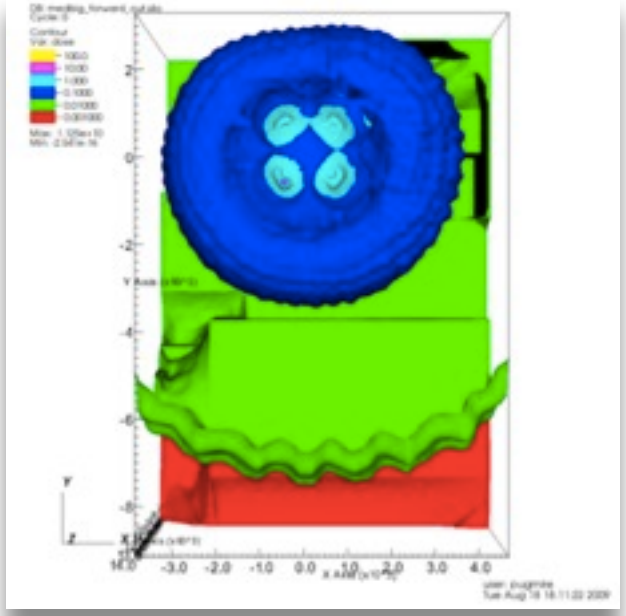


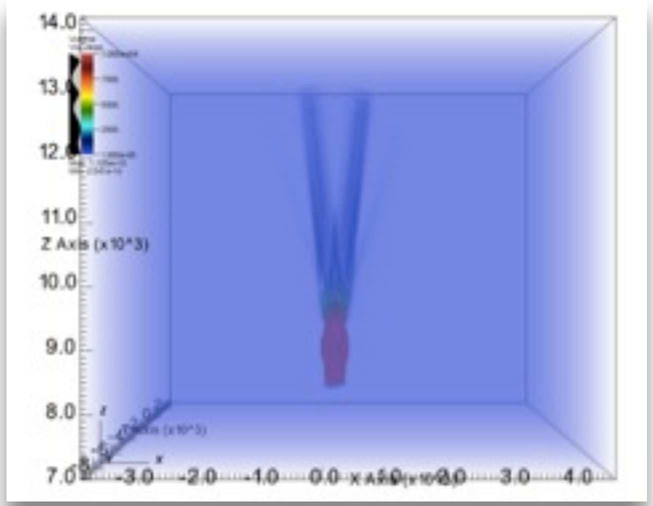
Joule and VisIt

Isocontouring and volume rendering of a radiation transport code, Denovo



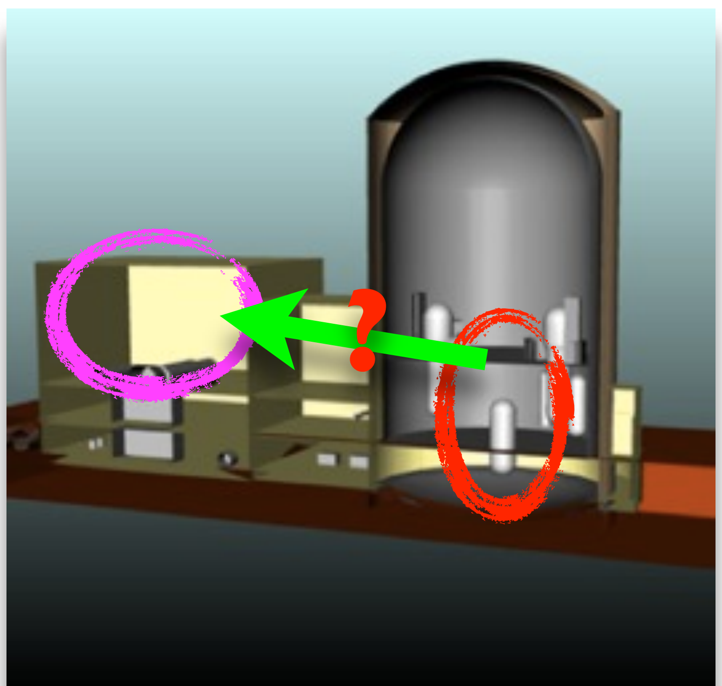
Q2 baseline:
103M cells on 4K cores





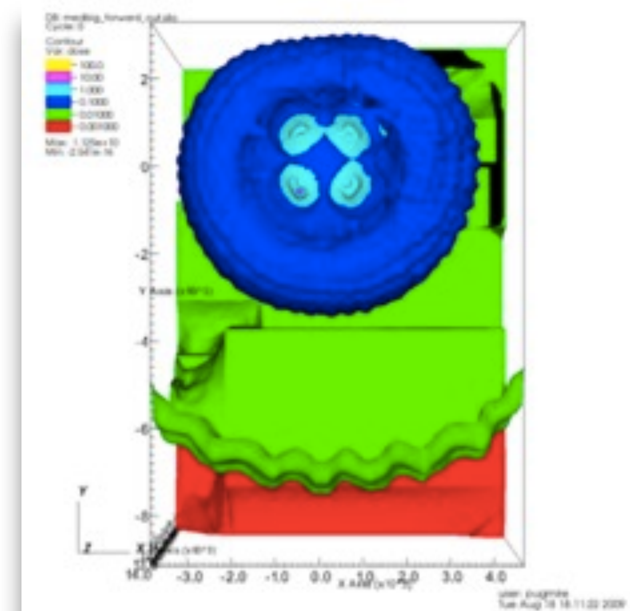
Joule and VisIt

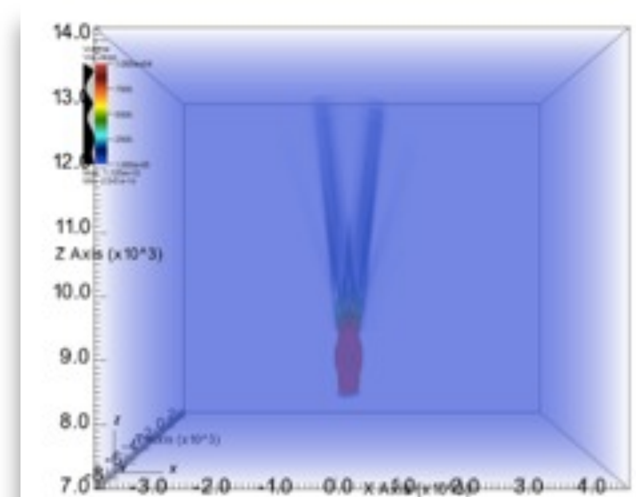
Isocontouring and volume rendering of a radiation transport code, Denovo



Q2 baseline:
103M cells on 4K cores

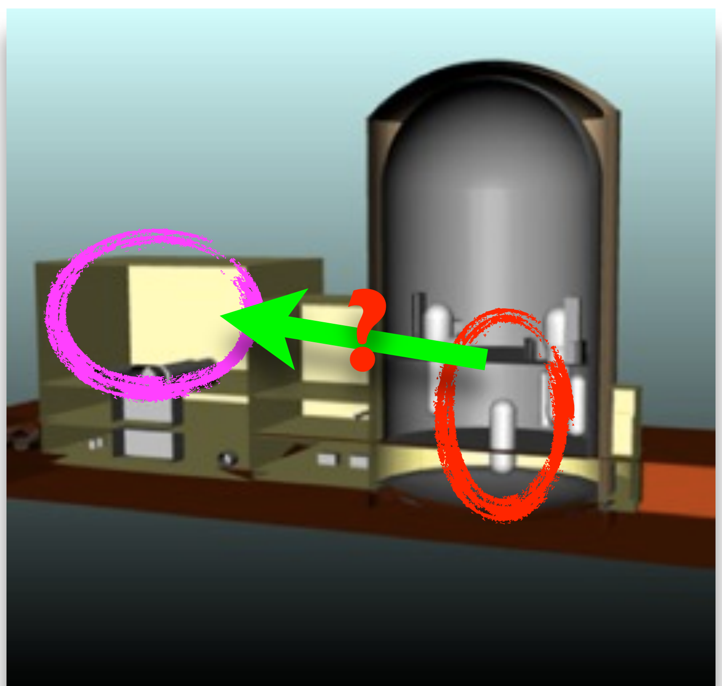
Q4 benchmark:
321M cells on 12K cores





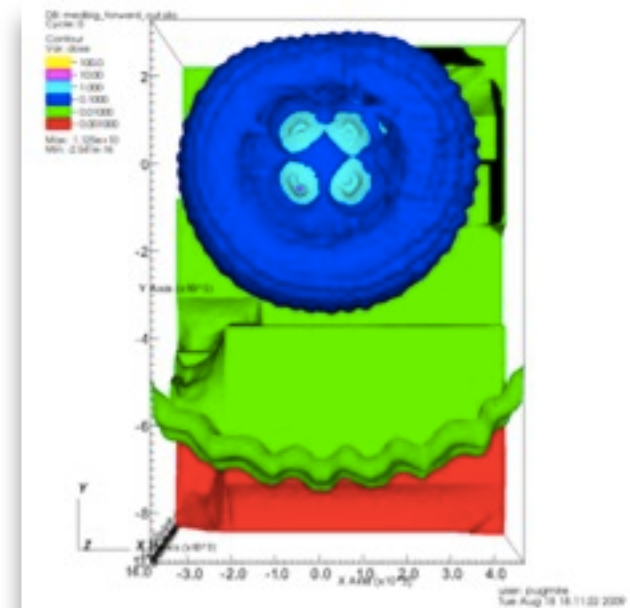
Joule and VisIt

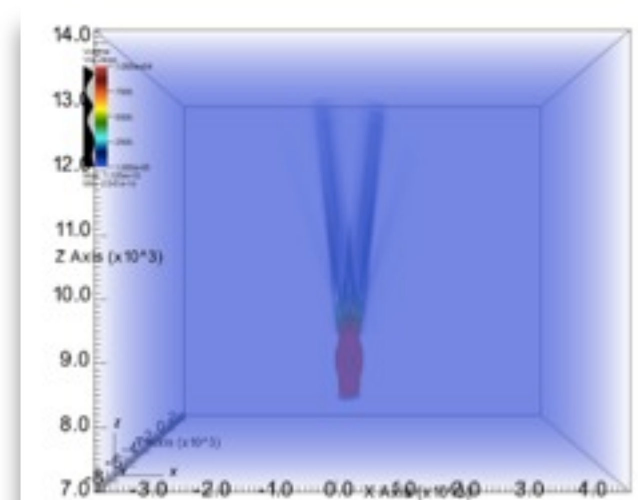
Isocontouring and volume rendering of a radiation transport code, Denovo



Q2 baseline:
103M cells on 4K cores

Q4 benchmark:
321M cells on 12K cores

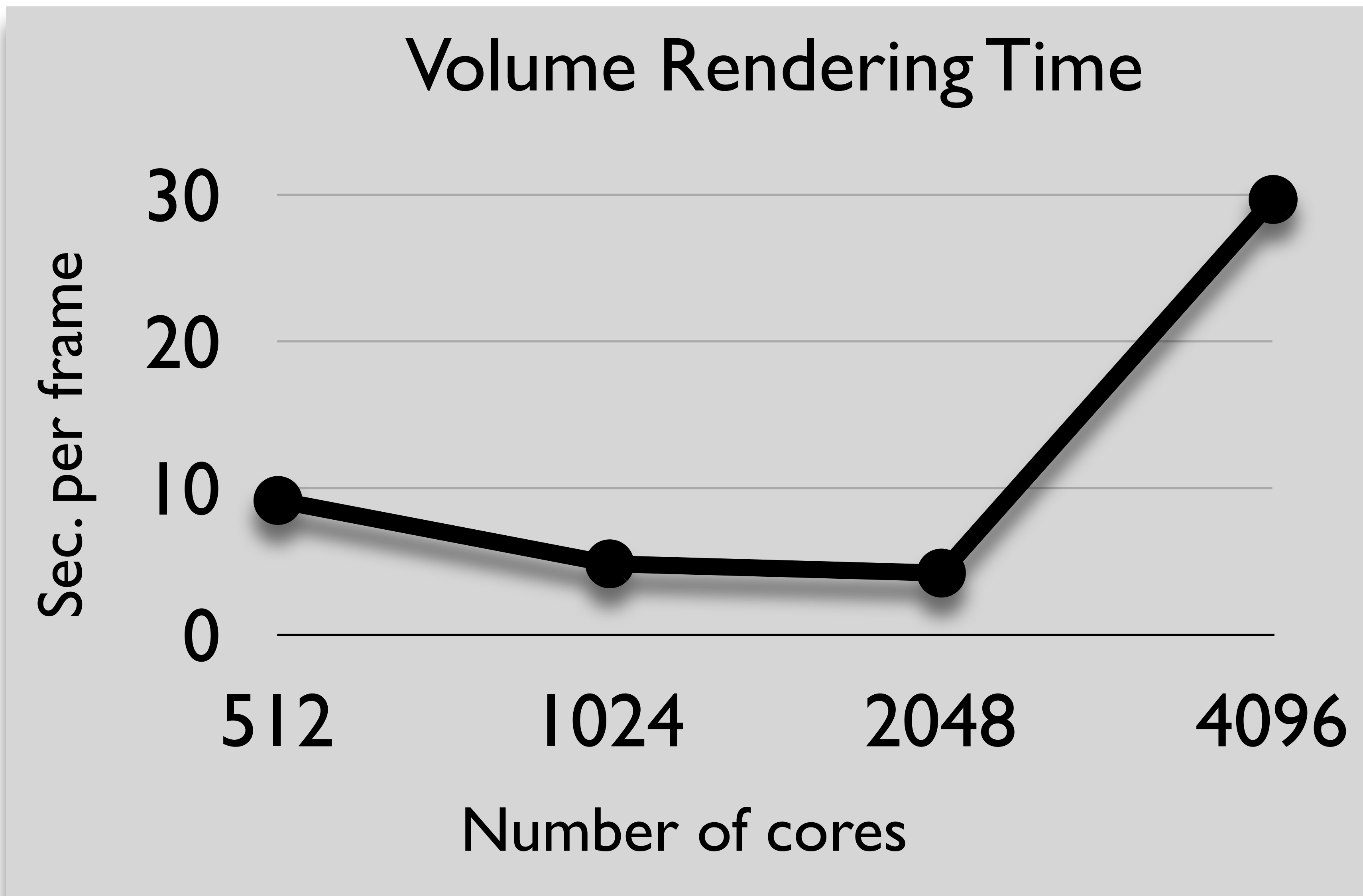




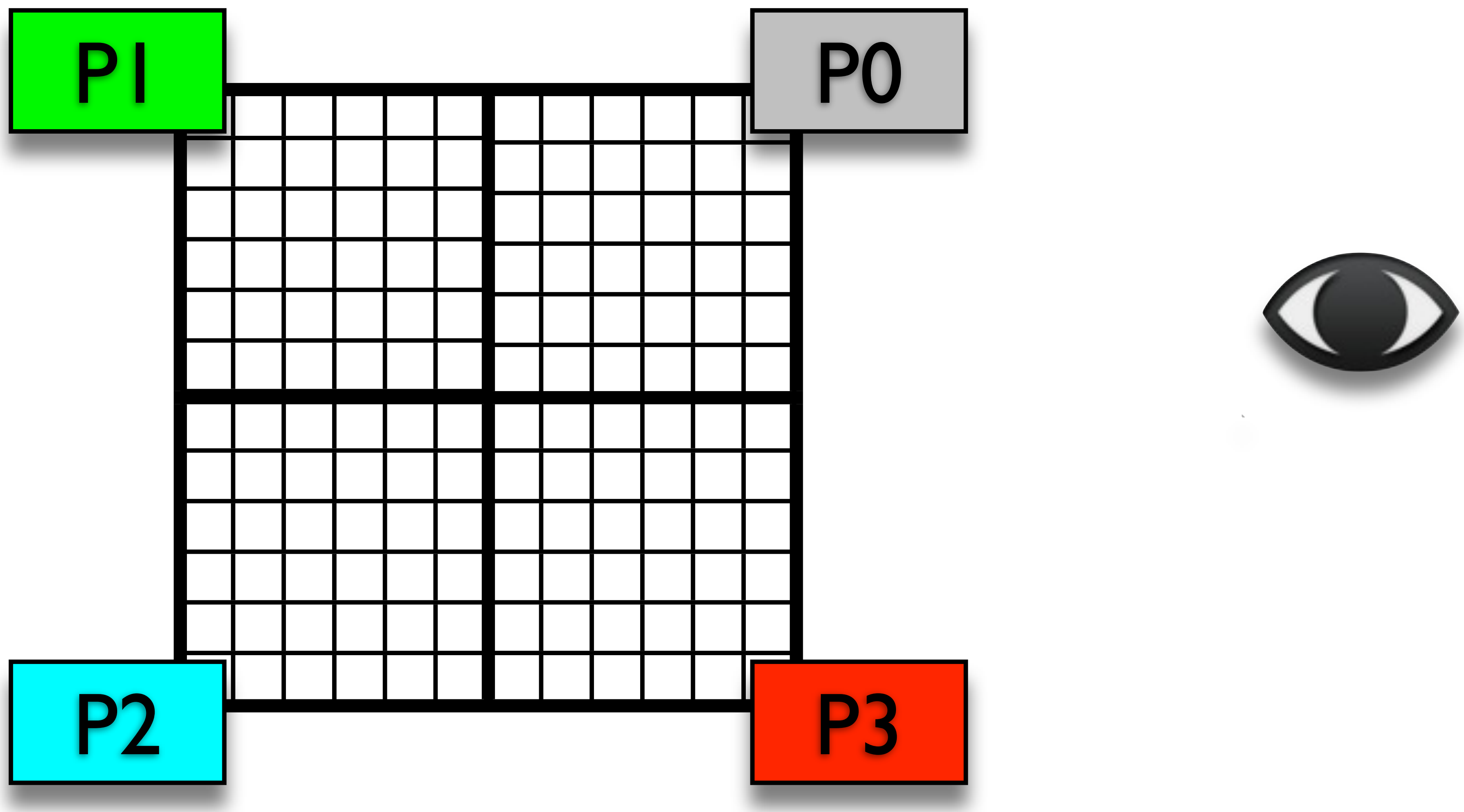
As expected, isocontouring exhibited nearly perfect scaling

Scalability of volume rendering

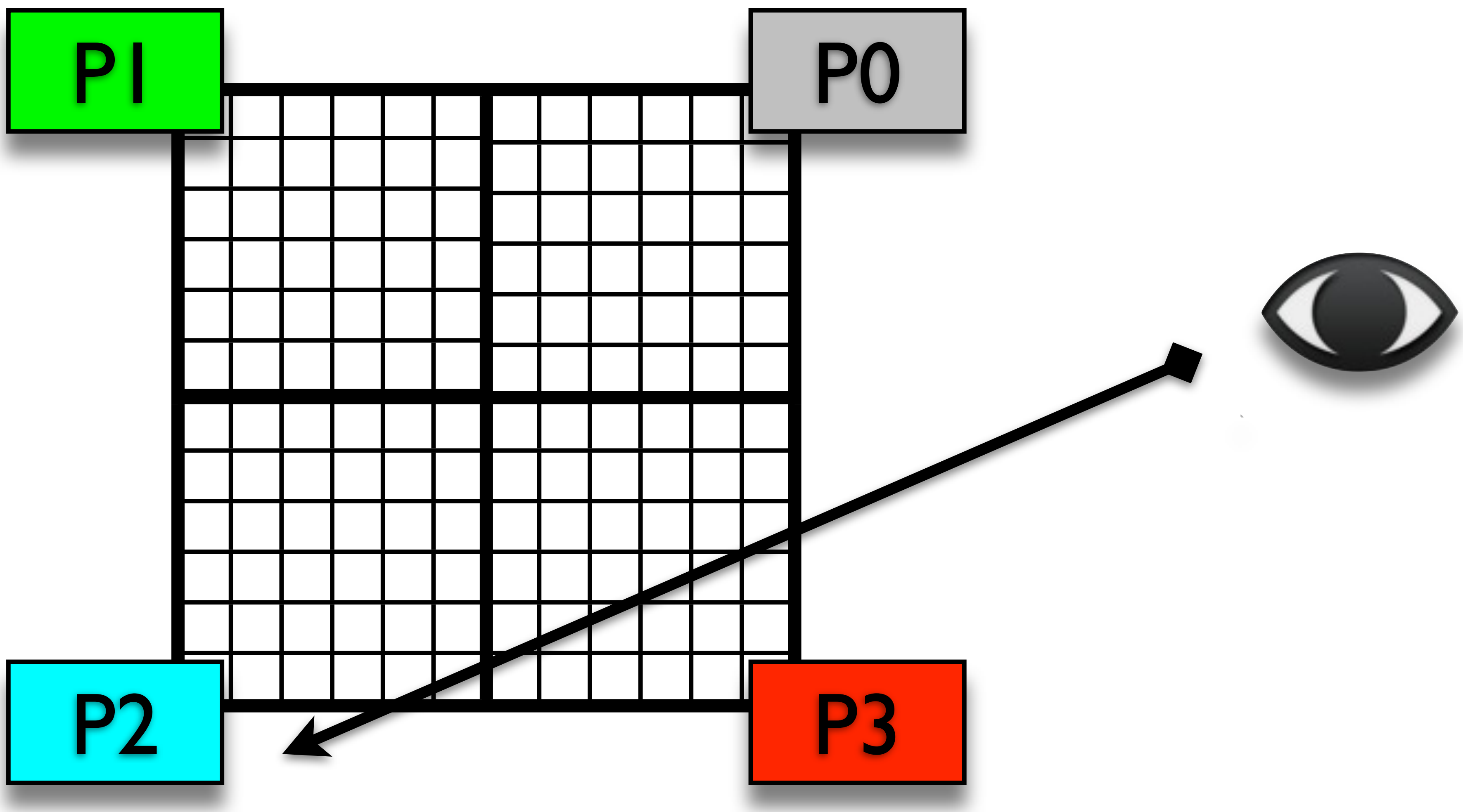
Ray casted volume rendering:
4000 samples per ray, 103M cells



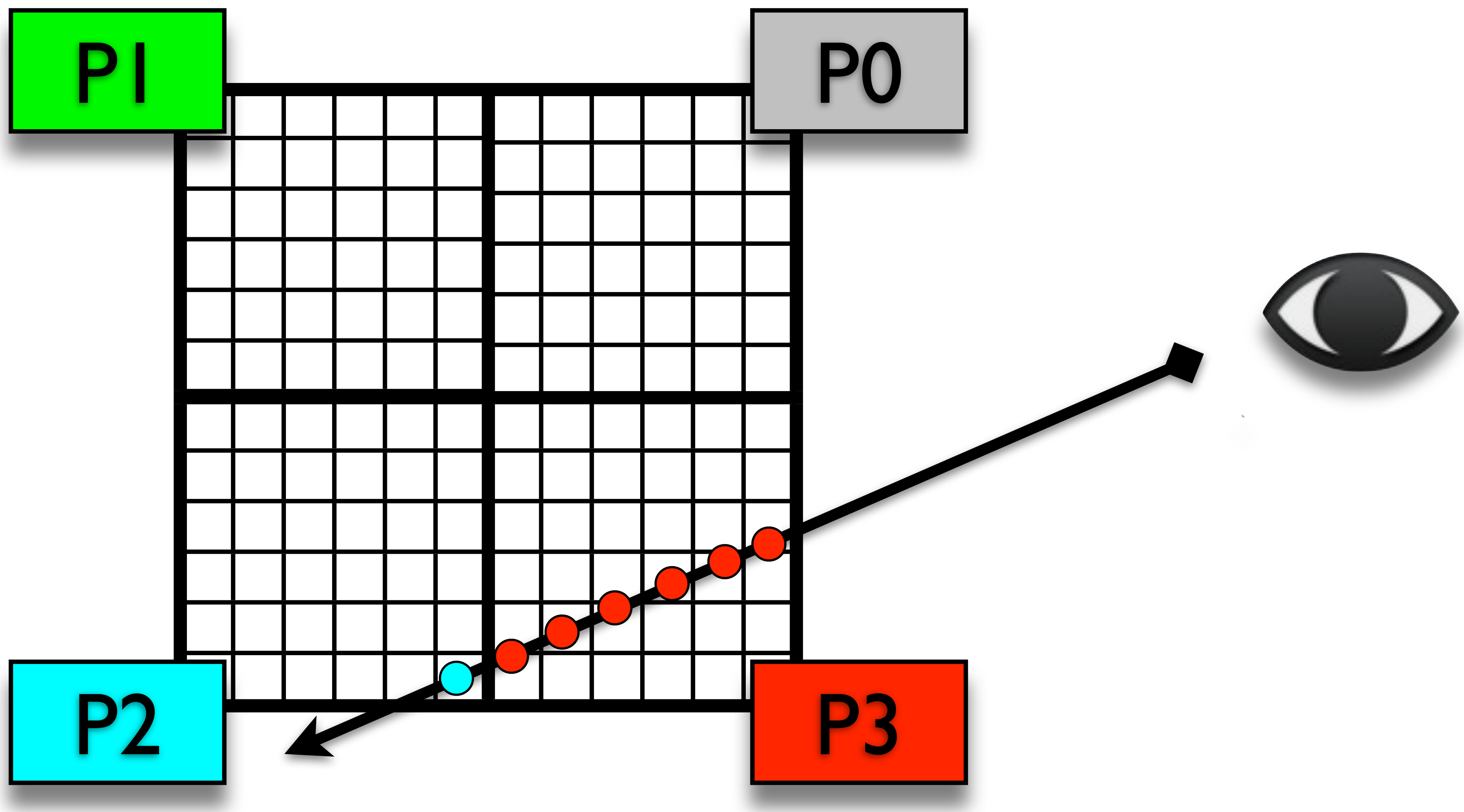
Barrier to volume rendering scalability



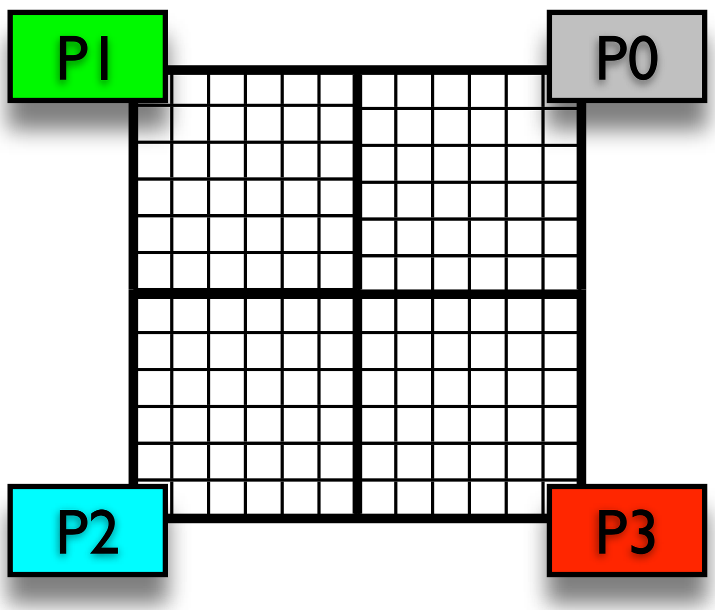
Barrier to volume rendering scalability

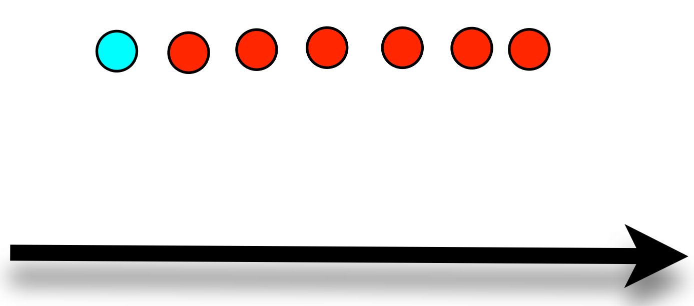


Barrier to volume rendering scalability



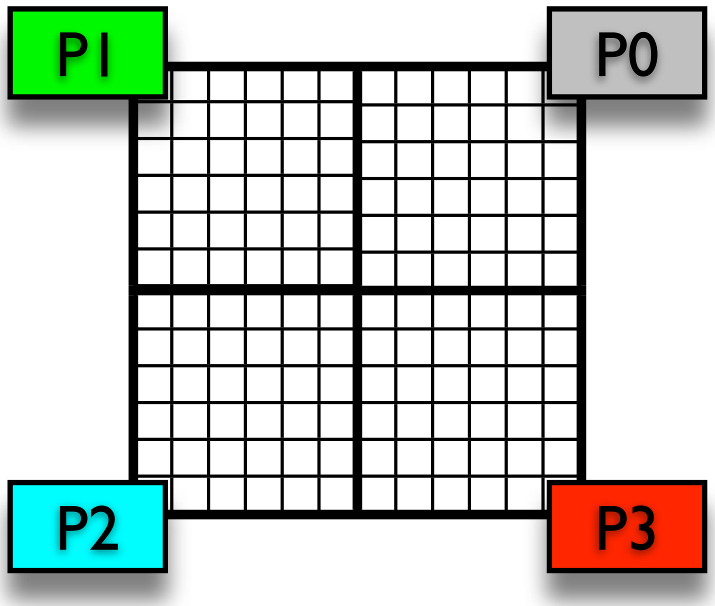
Barrier to volume rendering scalability

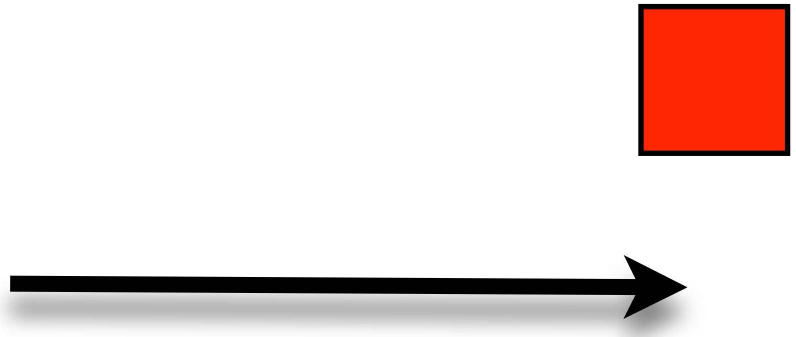




Back to front compositing

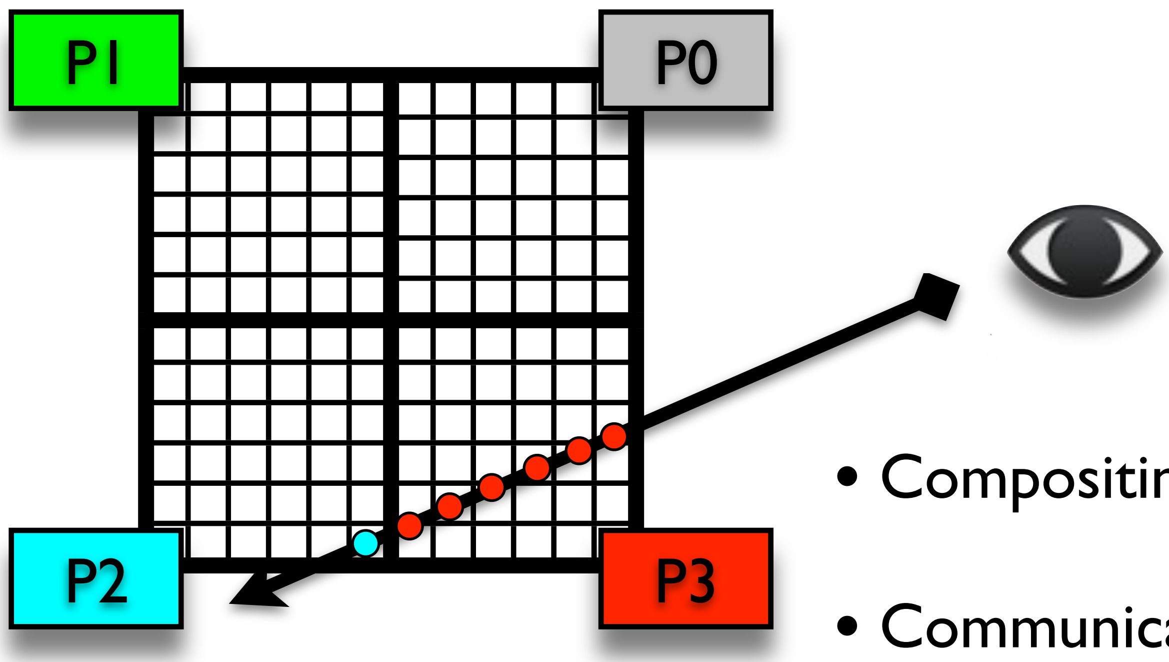
Barrier to volume rendering scalability





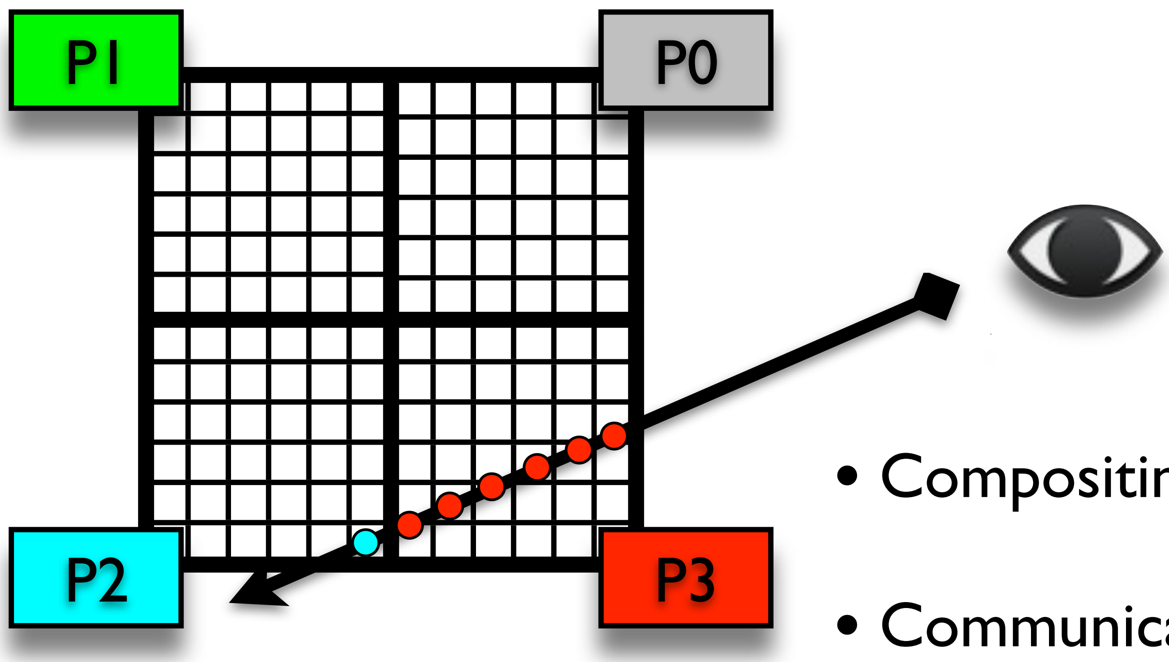
Back to front compositing

Volume rendering bottleneck



- Compositing puts all samples on a single processor.
- Communication minimized if **P2** sends samples to **P3**
- VisIt determines a communication minimizing assignment
- **However,** this optimization requires all-to-all communication

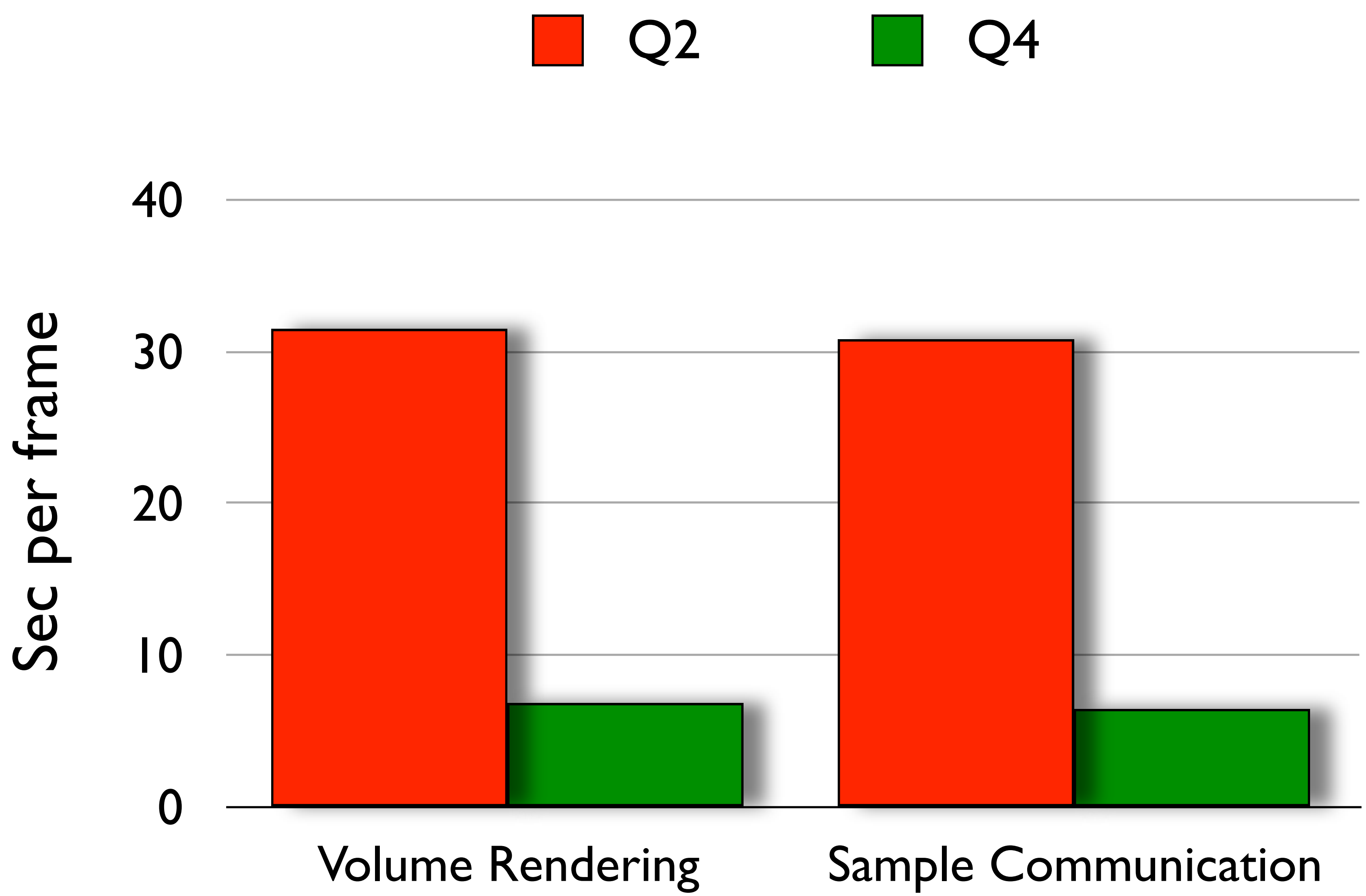
Volume rendering bottleneck



- Compositing puts all samples on a single processor.
- Communication minimized if **P2** sends samples to **P3**
- VisIt determines a communication minimizing assignment
- **However,** this optimization requires all-to-all communication

All to All communication does not scale

Avoiding **all-to-all** communication dramatically improved scalability



Q2 to Q4 weak scaling of **~5X** achieved

Outline

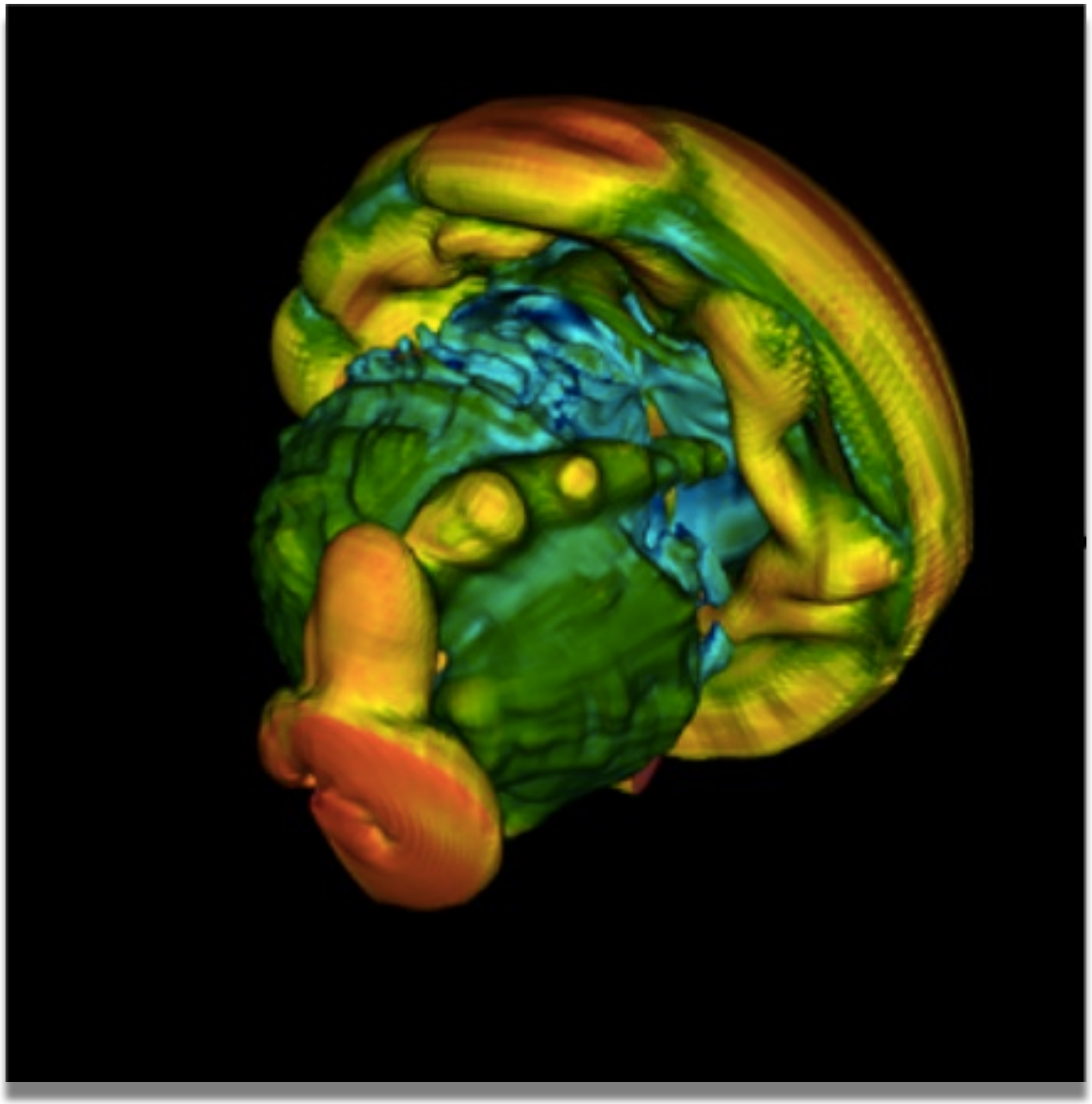
- ✓ Motivation and Introduction
- ✓ Weak scaling study: The Joule Metric
- ➡ **Scalability on exascale datasets**
- \- Final thoughts

Experiments at the exascale

- Is pure parallelism ready for exascale?
- Vary over:
  - Dataset size
  - Data generation
  - Supercomputer architectures

Experiment methodology

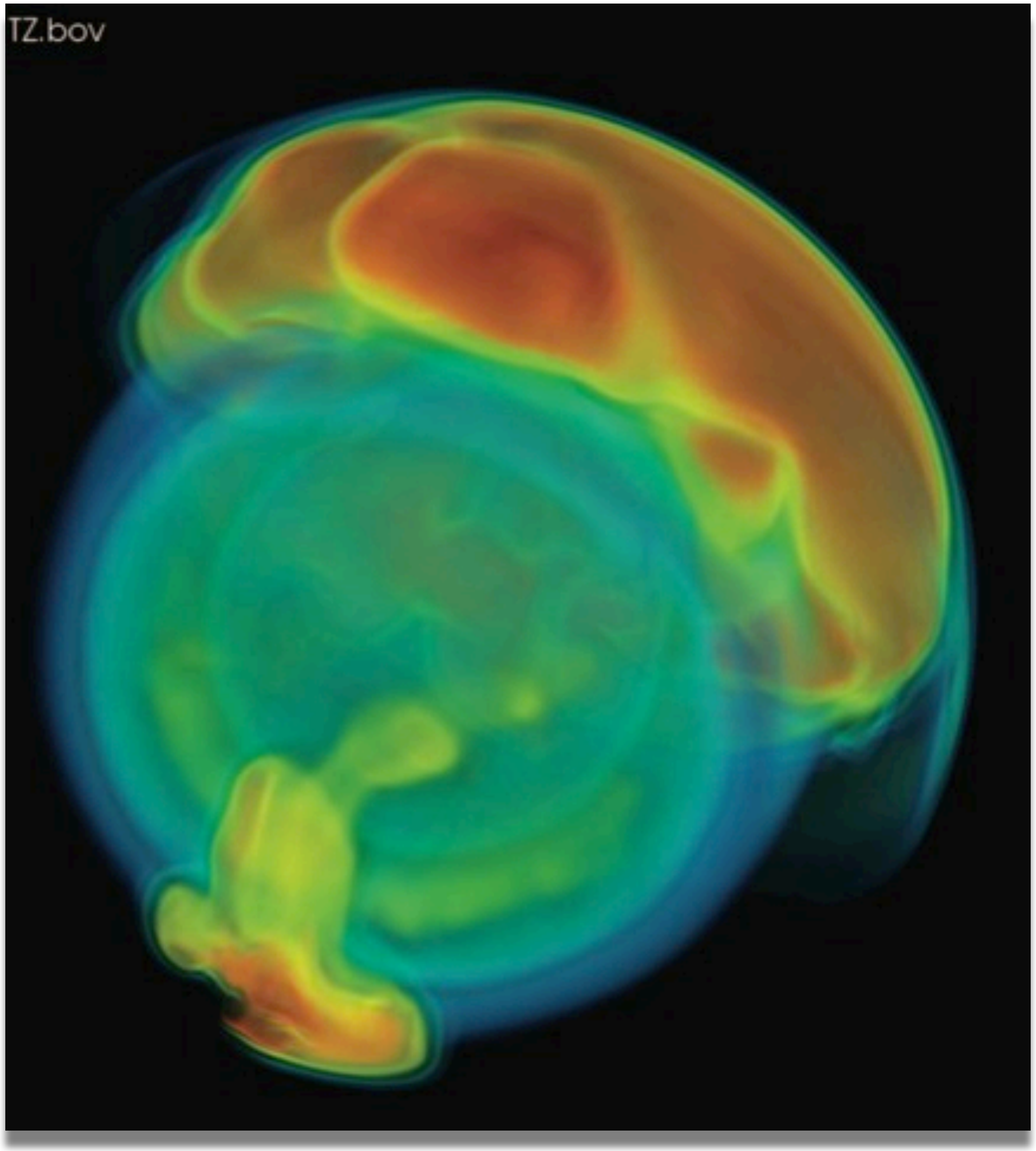
1. Dataset generation:
    - Tomorrow's data not available yet
    - Synthetic data should be reasonable surrogate
2. Read dataset from disk
3. Extract isosurface
4. Render @ 1024x1024



Isosurface of 2 trillion cells, visualized with VisIt on Jaguarpf using 32,000 cores.

Experiment methodology

- Bottlenecks found in volume rendering
- Fixed, but not in time for study
  - Viz runs are opportunistic
  - Hard to get a second chance
  - Render time ~ 5 sec per frame
- Contouring exercises much of the infrastructure



Volume rendering of 2 trillion cells, visualized with VisIt on Jaguarpf using 32,000 cores.

Experiment methodology

- 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 sub-setting
- Pure parallelism is what production visualization tools use right now

Vary over supercomputer

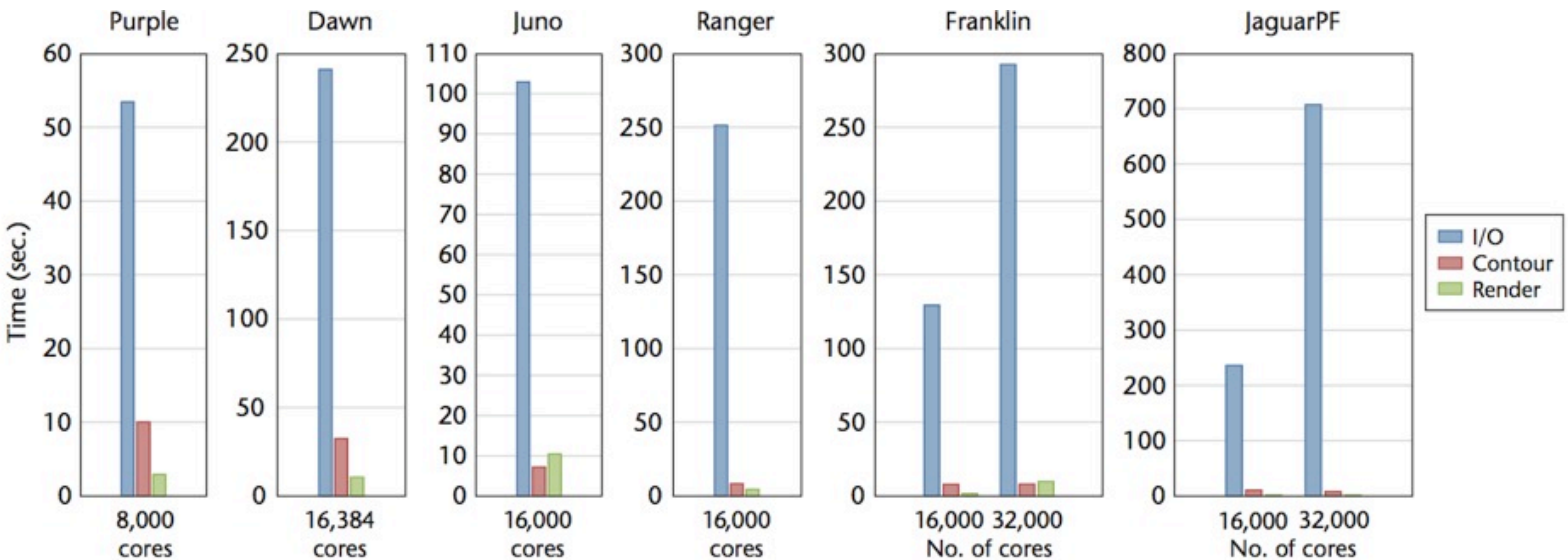
- Goals:
  - Ensure results aren't tied to a single machine.
  - Understand results from different architectures.
- Experiment details
  - 1 trillion cells per 16,000 cores
  - 10*NCores "Brick-of-float" files, gzipped
  - Upsampled data

Vary over supercomputer

- Goals:
  - Ensure results aren't tied to a single machine.
  - Understand results from different architectures.
- Experiment details
  - 1 trillion cells per 16,000 cores
  - 10*NCores "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 (*A*dvanced *I*nteractive E*x*ecutive)	12,208	3.5	Power5	1.9 GHz	92.8 Tflops	66

Results



Runtimes for I/O, contouring, and rendering.

Results

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

Results

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

Lustre striping of 2 vs. 4

Results

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

Lustre striping of 2 vs. 4

BG/L has 850 MHz clock

Results

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

Lustre striping of 2 vs. 4

BG/L has 850 MHz clock

7-10 network links failed.
Required re-routing

Results

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

Lustre striping of 2 vs. 4

BG/L has 850 MHz clock

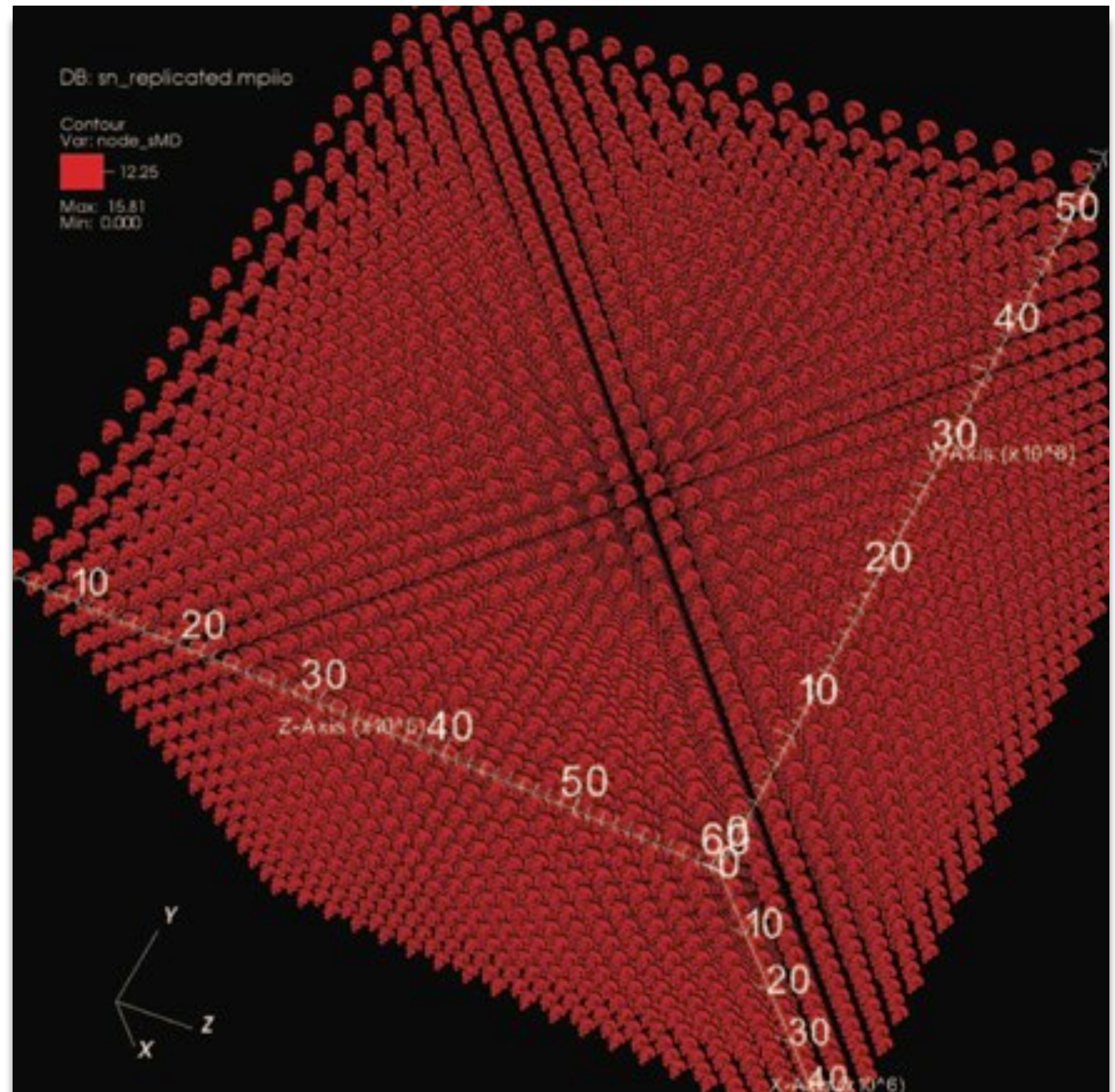
?????????

7-10 network links failed.
Required re-routing

Varying over data generation

- Concern:
  - Does upsampling produce unrepresentatively smooth surfaces?

- Alternative: replication



Isosurface 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

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

More triangles generated, but loading 1trillion zones **dominates**

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

More triangles generated, but loading 1trillion zones **dominates**

Replicated data contains more geometry

Pitfalls at scale

- Volume rendering
  - Algorithms scales well up to 2048 processors
- 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
    - Added in VisIt 2.0
    - Still need to demonstrate at scale

Pitfalls at scale

All to one communication

- Each MPI task needs to report high level information
  - Errors, data, spatial extents, etc
- Previous implementation:
  - Every MPI task sends a direct message to MPI task 0.
- New implementation (Mark Miller, LLNL):
  - Tree communication

Pitfalls at scale

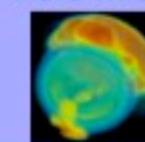
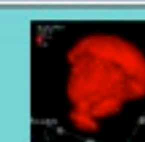
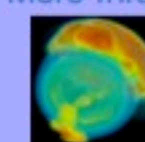
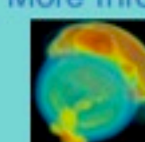
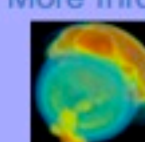
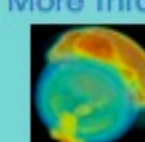
All-to-one?	No. of cores	Data set size (TCells)	Total I/O time (sec.)	Contour time (sec.)	Total pipeline execution time (sec.)	Pipeline minus contour & I/O (sec.)	Date run
Yes	16,384	1	88.0	32.2	368.7	248.5	June 2009
Yes	65,536	4	95.3	38.6	425.9	294.0	June 2009
No	16,384	1	240.9	32.4	277.6	4.3	Aug. 2009

Pitfalls at scale

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

Debugging runs at scale critical to resolving these issues

Continued study

Date	Number of zones per timestep	Number of timesteps	Total Size Zones & Bytes	Mesh+Vars	Platform	Visioneers (Visualization-Engineers)	More Info...
October 2009	$20{,}001^3$ (8 Tz)	1		rect+1	12000 cpus of Graph (Linux)	Cyrus Harrison	More Info 
June 2009	$15{,}871^3$ (4 Tz)	1		rect+1	65536 cpus of Dawn (BlueGene/P) Dawn Details	Brad Whitlock	
May 2009	$12{,}596^3$ (2 Tz)	1		rect+1	32000 cpus of Jaguar (Cray)	David Pugmire Sean Ahern	More Info 
June 2009	$12{,}596^3$ (2 Tz)	1		rect+1	32000 cpus of Franklin (Cray)	Mark Howison Prabhat Hank Childs	More Info 
April 2009	$10{,}000^3$ (1 Tz)	1		rect+1	16000 cpus of Jaguar (Cray)	David Pugmire Sean Ahern	More Info 
May 2009	$10{,}000^3$ (1 Tz)	1		rect+1	16000 cpus of Ranger (Sun Linux)	Hank Childs	More Info 

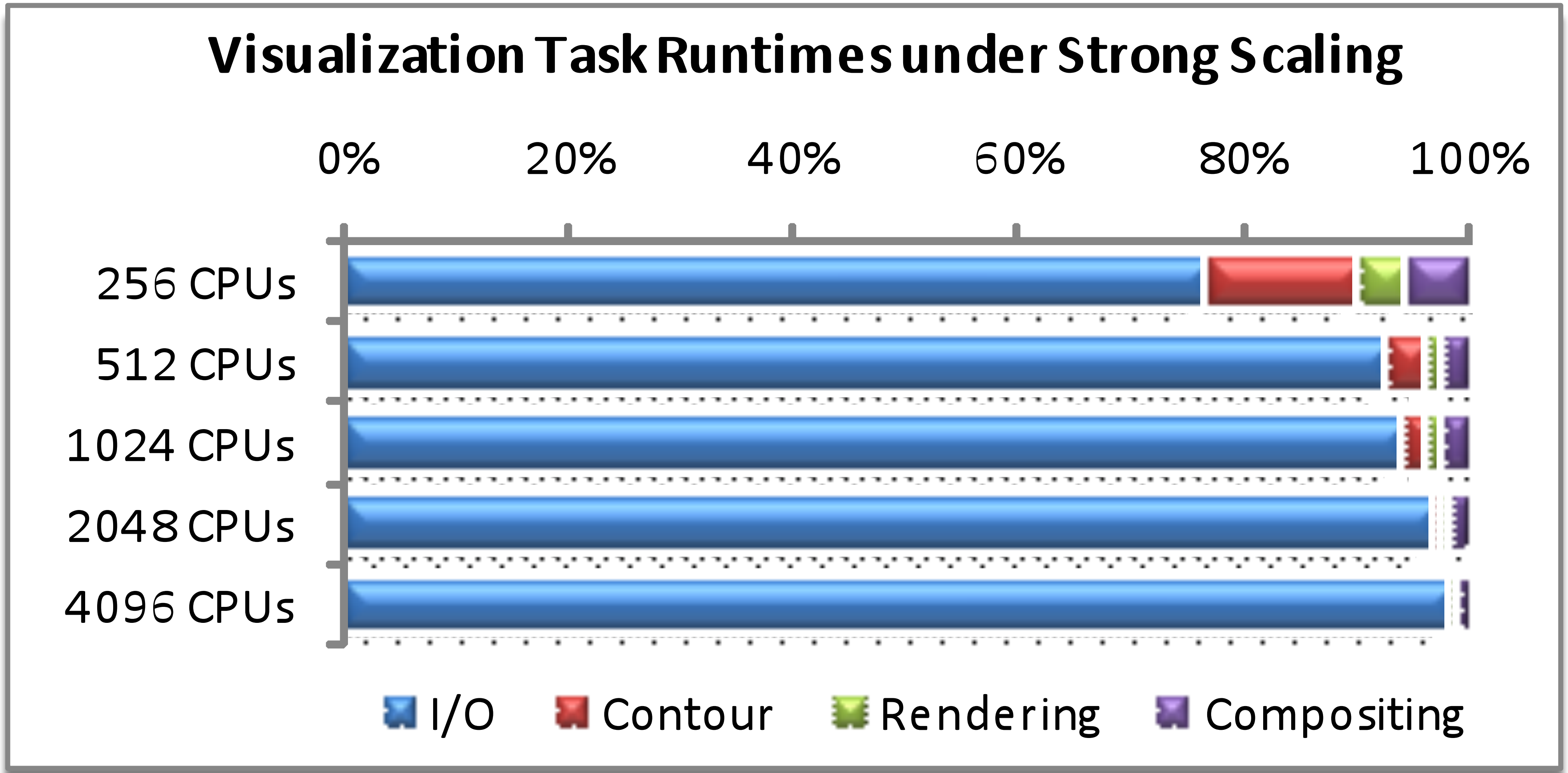
Outline

- ✓ Motivation and Introduction
- ✓ Weak scaling study: The Joule Metric
- ✓ Scalability on exascale datasets
- ➡ **Final thoughts**

Should more tools have been used?

- Could have performed this study with VisIt, ParaView, EnSight, etc.
- Successful test with VisIt validates pure parallelism.
- Of course, I/O is a big problem … but ParaView, EnSight, etc, are doing the same **fread()**

Trends in I/O



- I/O dominates pure parallelism architectures
  - Usually > 50%
  - Sometimes 98% !
- Amount of data to visualize is typically O(total mem)
- Relative I/O is key: $\frac{Mem}{IO}$

Trends in I/O

Machine	Year	$\frac{GB/s}{GFLOPS}$	Time to write memory
ASCI Red	1997	0.30%	300 sec
ASCI Blue Pacific	1998	0.15%	400 sec
ASCI White	2001	0.07%	660 sec
ASCI Red Storm	2004	0.14%	660 sec
ASCI Purple	2005	0.10%	500 sec
Jaguar XT4	2007	0.02%	1400 sec
Roadrunner	2008	0.01%	1600 sec
Jaguar XT5	2008	0.02%	1250 sec
Exascale	~2020	**????**	**????**

Why is relative I/O getting slower?

- "I/O doesn't pay the bills"
  - And I/O is becoming a dominant cost in the overall supercomputer procurement.
- Simulation codes are less vulnerable
  - But will be more exposed with proposed future architectures.

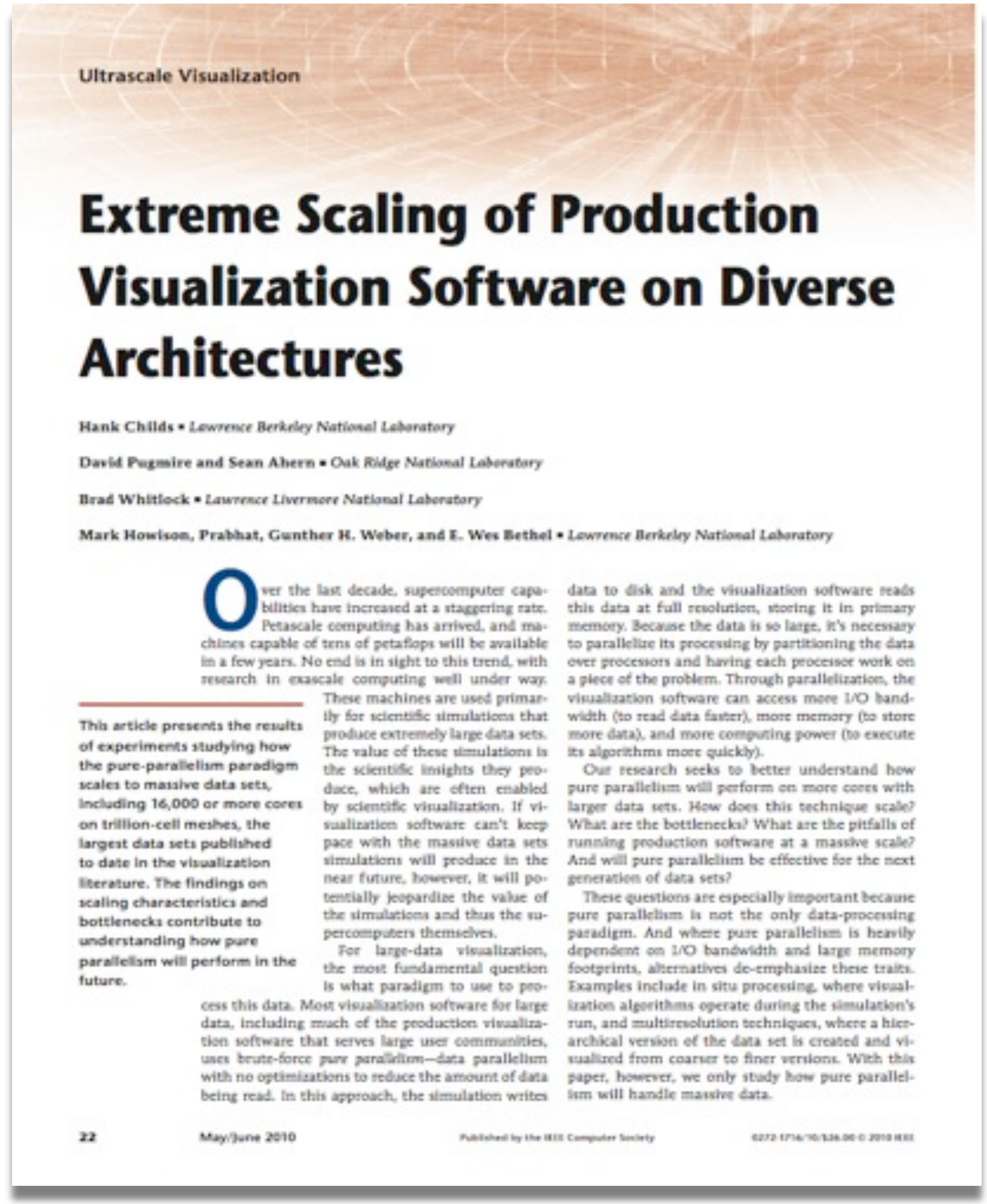
Why is relative I/O getting slower?

- "I/O doesn't pay the bills"
  - And I/O is becoming a dominant cost in the overall supercomputer procurement.
- Simulation codes are less vulnerable
  - But will be more exposed with proposed future architectures.

We **MUST** de-emphasize I/O in our visualization and analysis tools.

Conclusions

- Pure parallelism works, but is only as good as the underlying I/O infrastructure
  - – I/O future looks grim
  - – In situ, multi-resolution **needed**
- Full results available in special issue of *Computer Graphics & Applications, special issue 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*

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.

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. 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.

22 May/June 2010 Published by the IEEE Computer Society 0272-1716/10/$26.00 © 2010 IEEE

Questions ?