# Introduction to Visualization on Stampede
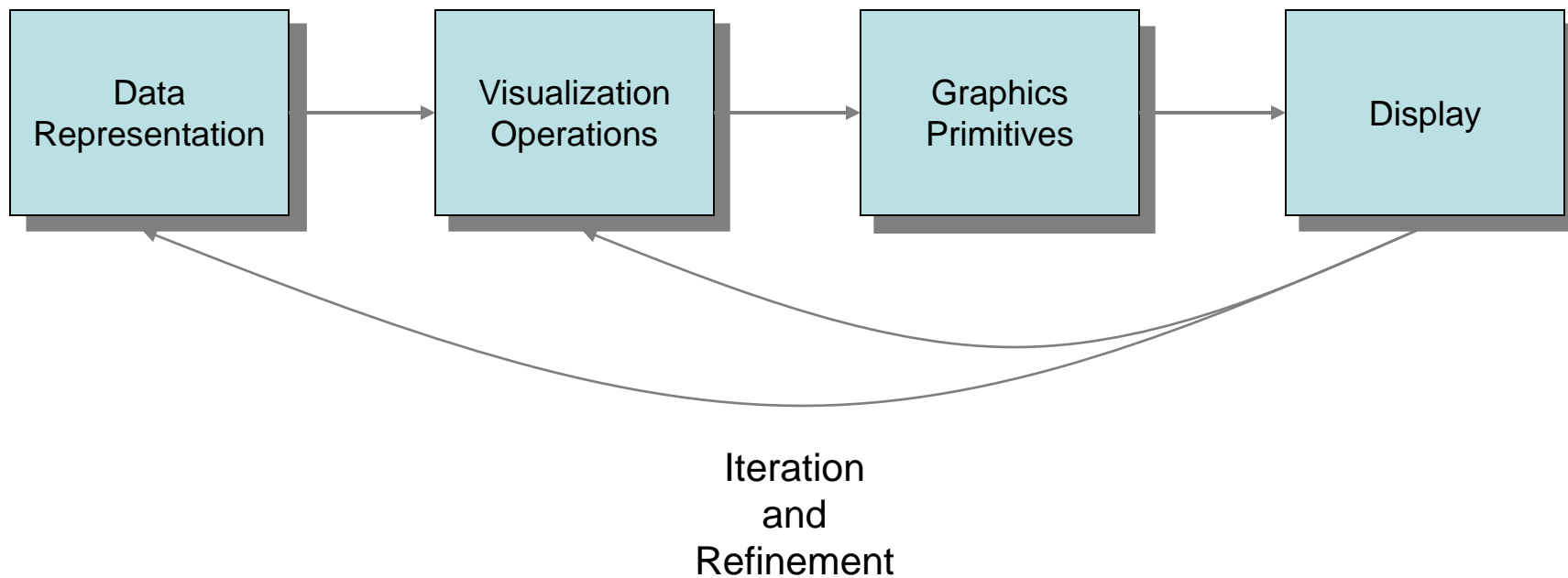
Aaron Birkland

Cornell CAC

With contributions from TACC visualization training materials

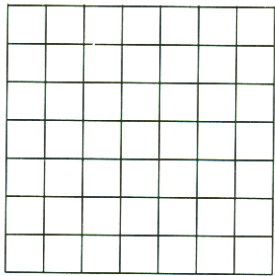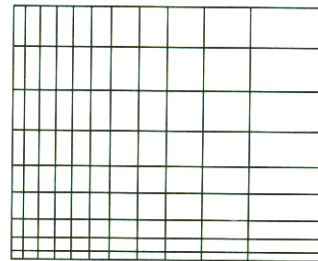*Parallel Computing on Stampede*

*June 11, 2013*

# From data to Insight

```
Data            Visualization      Graphics          Display
Representation   Operations        Primitives
```

Iteration
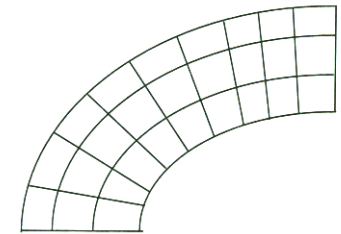and
Refinement

# Points, Meshes, and Coordinates

Medical scan
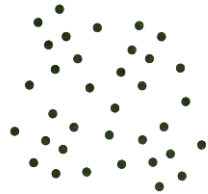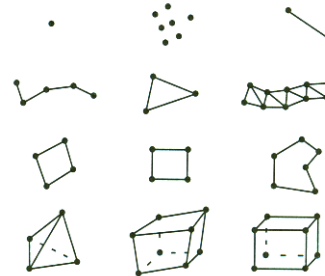
(a) Structured Points

(b) Rectilinear Grid

(c) Structured Grid

Engineering Model

N-body simulation

(d) Unstructured Points

(e) Polygonal Data
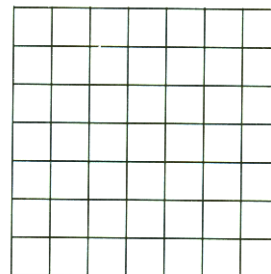
(f) Unstructured Grid

Extracted Surface
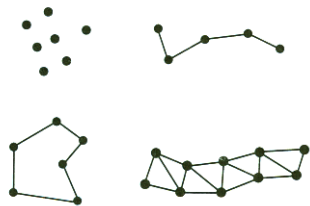
From *The Visualization Toolkit* by Schroeder et al.

# Data

- Values at each point
- Type and nature will determine applicable techniques
  - Scalar, Vector, Tensor?
  - Discrete? Continuous?
  - Nominal, Ordinal, Interval, Ratio?

- Now what do want to show about your data?

(a) Structured Points
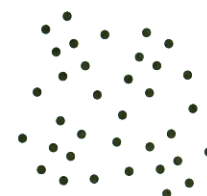
(d) Unstructured Points
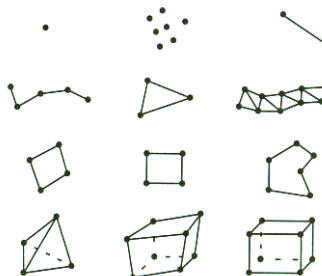
(f) Unstructured Grid

## Surface Shading (Pseudocolor)

Given a scalar value at a point on the surface and a color map, find the corresponding color (and/or opacity) and apply it to the surface point.
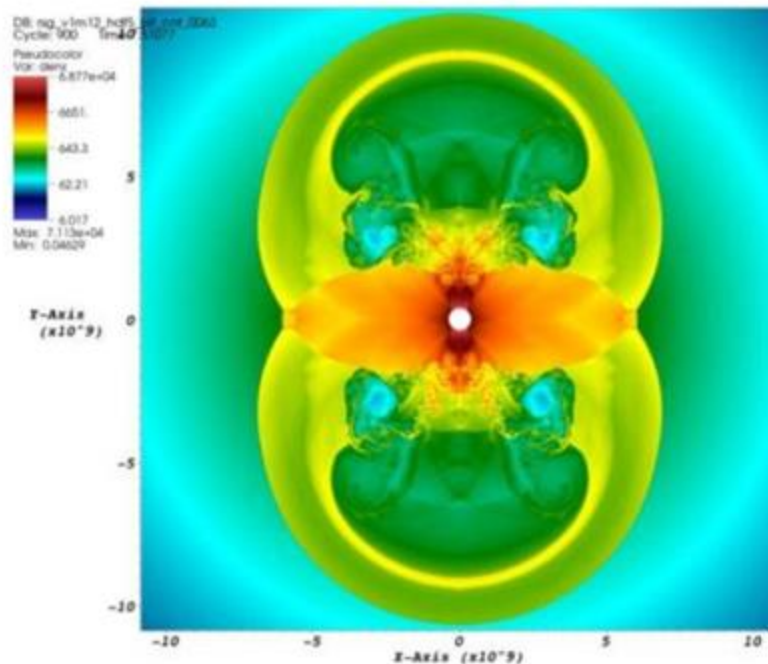
# Isosurfaces (Contours)

- Surface that represents points of constant value with a volume

- Plot the surface for a given scalar value.

- Good for showing known values of interest

- Good for sampling through a data range
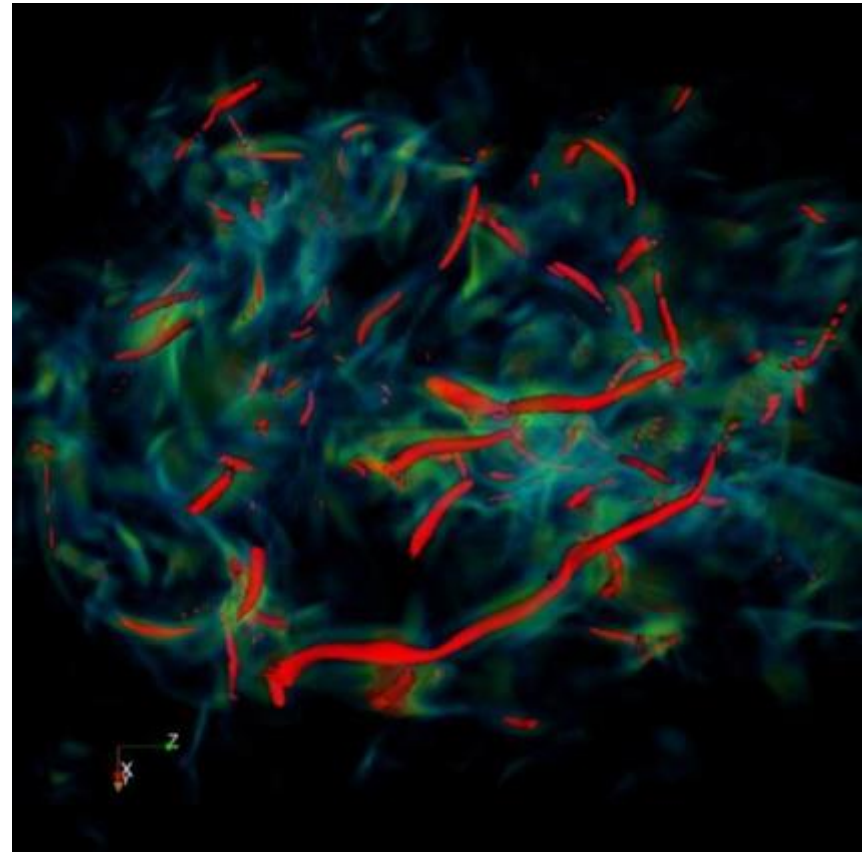
# Volume Rendering

Expresses how light travels through a volume
Color and opacity controlled by transfer function
Smoother transitions than isosurfaces



www.cac.cornell.edu
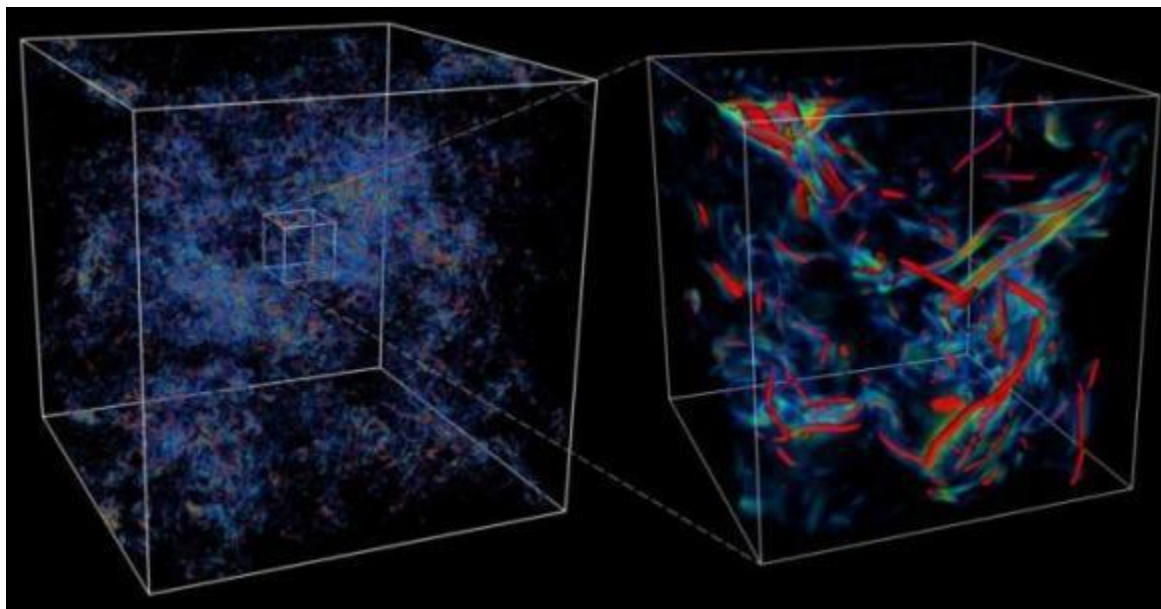
# Particle Traces (Streamlines)

Given a vector field, extract a trace that follows that trajectory defined by the vector.

$$P_{new} = P_{current} + V_P \Delta t$$

Streamlines – trace in space
Pathlines – trace in time
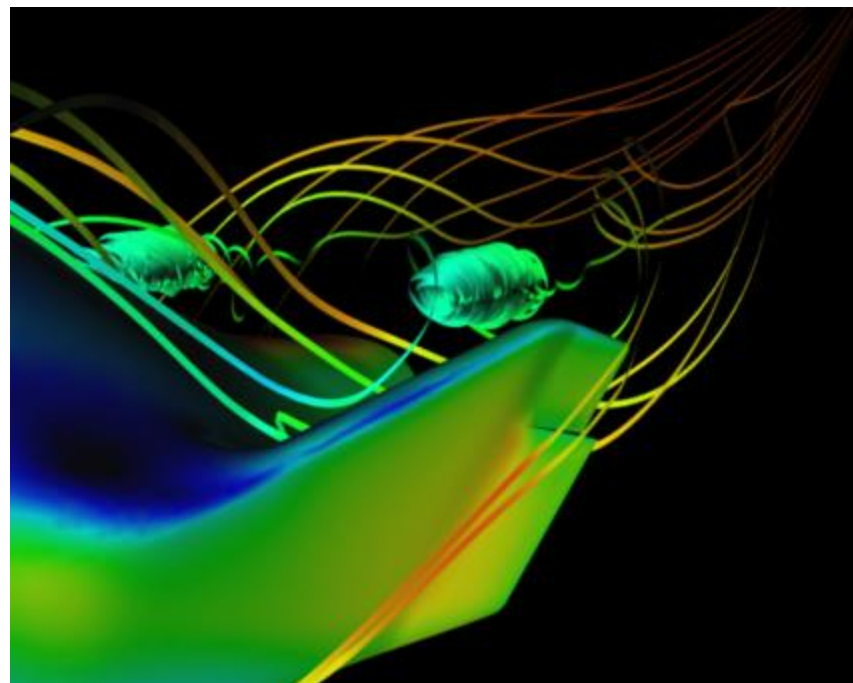
# Graphics Primitives

- Basic unit: Polygons, Colors, Textures, Opacity
  - Flat surface formed between points
  - This surface may have an associated color or texture, or opacity
- Complex surfaces composed of several polygons
- A dataset in and of itself!

# Graphics Pipeline

```
Modeling          Trivial Rejection    Illumination         Viewing
Transformation                                              Transformation
```

```
Clipping          Projection           Rasterization        Display
```
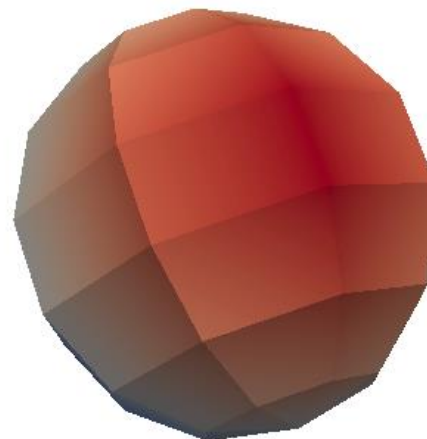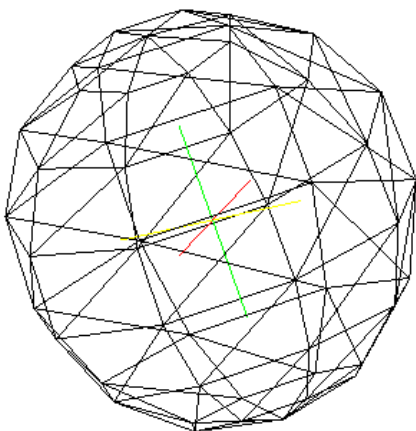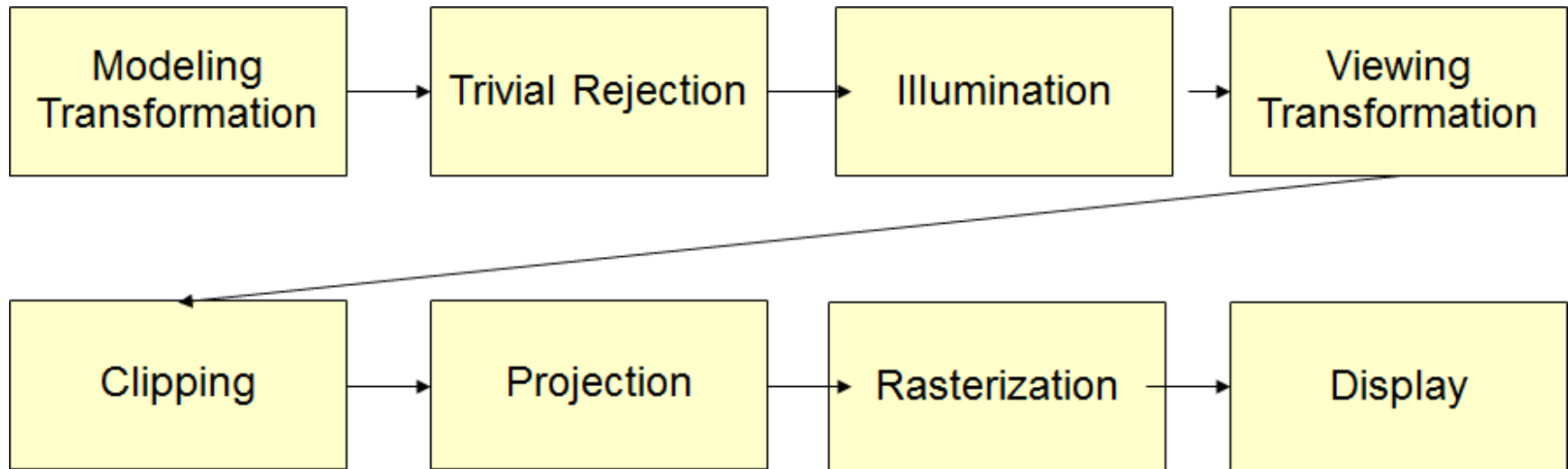
# Visualization applications

- Support for many input formats
    - Some may be better than others for certain tasks
- Aim for realtime point and click image manipulation
    - Data exploration
- Pipeline of data refinement or visualization operations
- Lots of tweakable parameters
- Parallel rendering (more on this later)
- Ease of use and suitability for certain tasks can vary
    - Definitely not one-size-fits-all

# ParaView

- http://www.paraview.org/
- Open-source, multi-platform parallel data analysis and visualization application
- Mature, feature-rich interface
- Good for general-purpose, rapid visualization
- Built upon the Visualization ToolKit (VTK) library
- Primary contributors:
    - Kitware, Inc.
    - Sandia National Laboratory
    - Los Alamos National Laboratory
    - Army Research Laboratory

# ParaView

- Supports derived variables
  - New scalar / vector variables that are functions of existing variables in your data set
- Scriptable via Python
- Saves animations
- Can run in parallel / distributed mode for large data visualization

# ParaView

- All processing operations (filters) produce data sets
- Can further process the result of every operation to build complex visualizations
  - e.g. can extract a cutting plane, and apply glyphs (i.e. vector arrows) to the result
    - Gives a plane of glyphs through your 3D volume

# VisIt
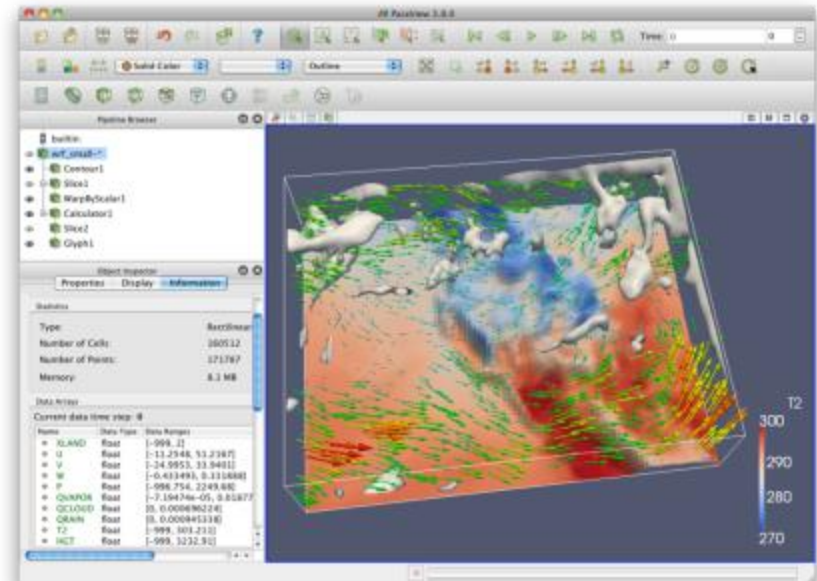
- https://wci.llnl.gov/codes/visit/
- Open Source, Multiplatform, interactive parallel visualization and graphical analysis tool
- Developed by the Department of Energy (DOE) Advanced Simulation and Computing Initiative (ASCI)
- Although VisIt was developed for visualizing terascale data, it is also well suited typical desktop simulations
- Can run in parallel/distributed mode for large-scale visualization

# VisIt

- VisIt's visualization capabilities are grouped into two categories:

  – Plots are used to visualize data and include boundary, contour, label, mesh, pseudocolor,

  – Operators consist of operations that can be performed on the data prior to visualization. (Examples include slice, isosurface, threshold among others)
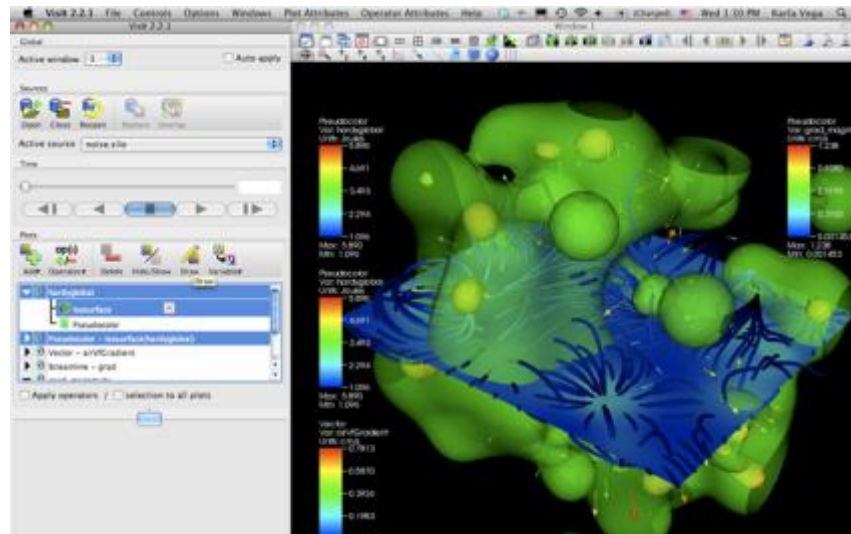
# Parallel Visualization

- Why? Performance
  - Processing may be too slow on one CPU
    - Interactive visualization requires real-time frame rates
    - Use lots of CPUs
    - Shared-memory/multicore *or* distributed

  - Data may be too big for available node
    - Virtual memory works, but paging is slow
    - Use lots of nodes to increase physical memory size
    - Big shared-memory/multicore scaling is costly ($/CPU)

## Increase interactivity or feasibility

# Memory Utilization

- Some visualization techniques cause memory use to skyrocket!
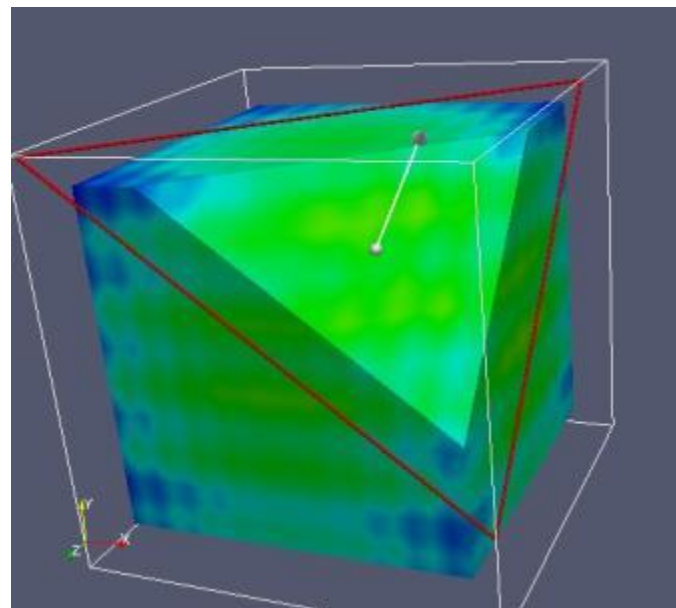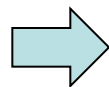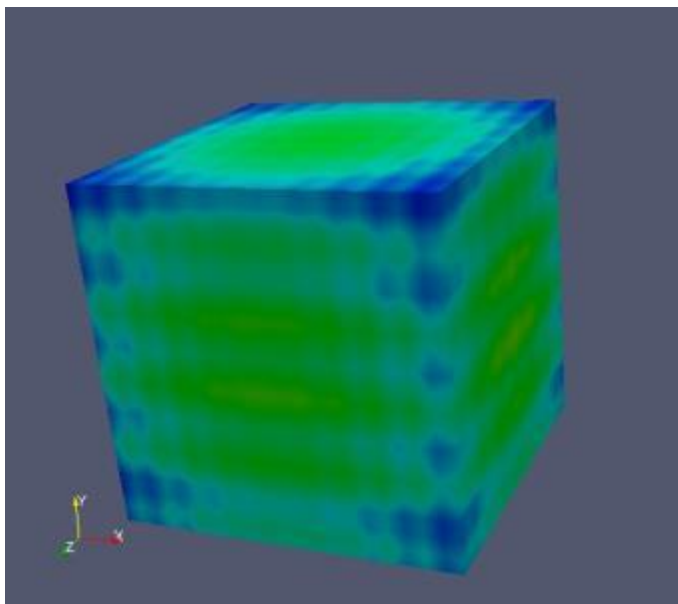
# Memory Utilization: Regular Grids

- Specified by:
  - (x,y,z) origin
  - (nx, ny, nz) counts
  - Data array
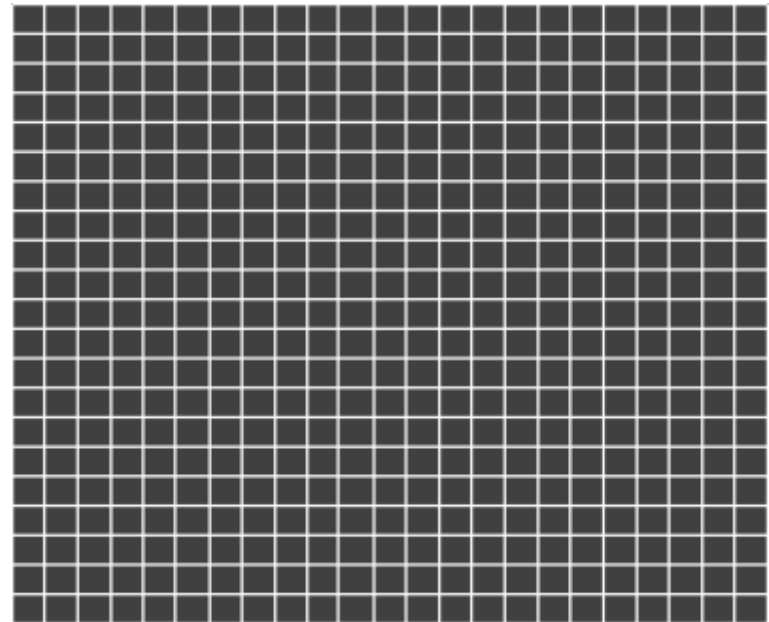- Requires very little memory

# Memory Utilization: Regular Grids

- Chop off corner -> need an unstructured grid to represent data points
- Specified by
  - Explicit list of vertices
  - Explicit list of triangles
- Memory use can go up *many* times
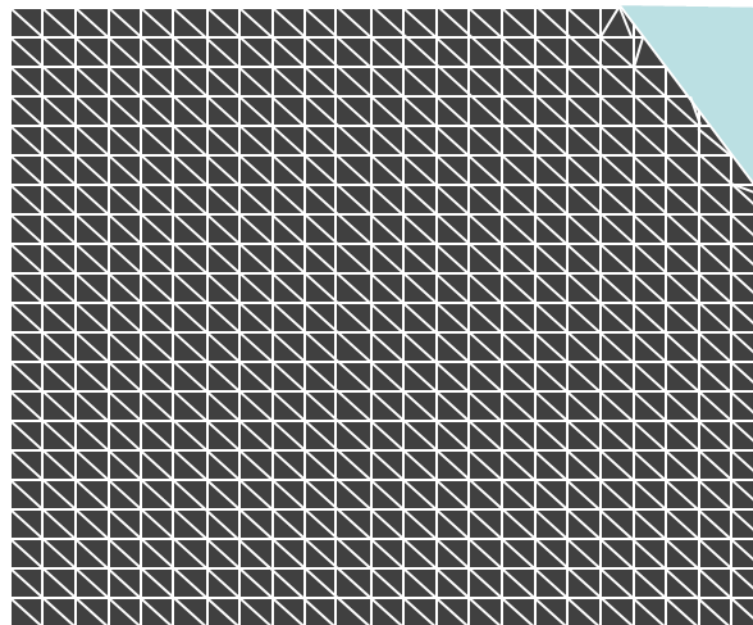
# Memory Utilization: examples

- Mummy.vtk:
  - Structured Grid
  - (128x128x128)
  - 2MB raw data
- Contour: 7MB
  - Polygonal Mesh
- Slice of Contour: .1MB
- Tetrahedralize: 520MB!!
  - Unstructured Grid
  - Data points ->
    Tetrahedrons
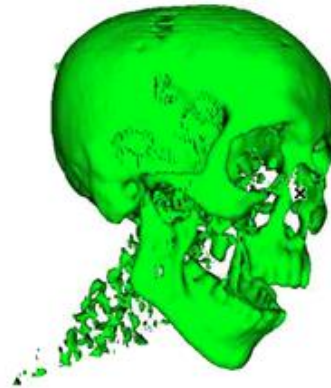
# Visualization scales with HPC

- Large data produced by large simulations require large visualization machines and produce large visualization results
- Data and all derivations in memory,

cumulative!
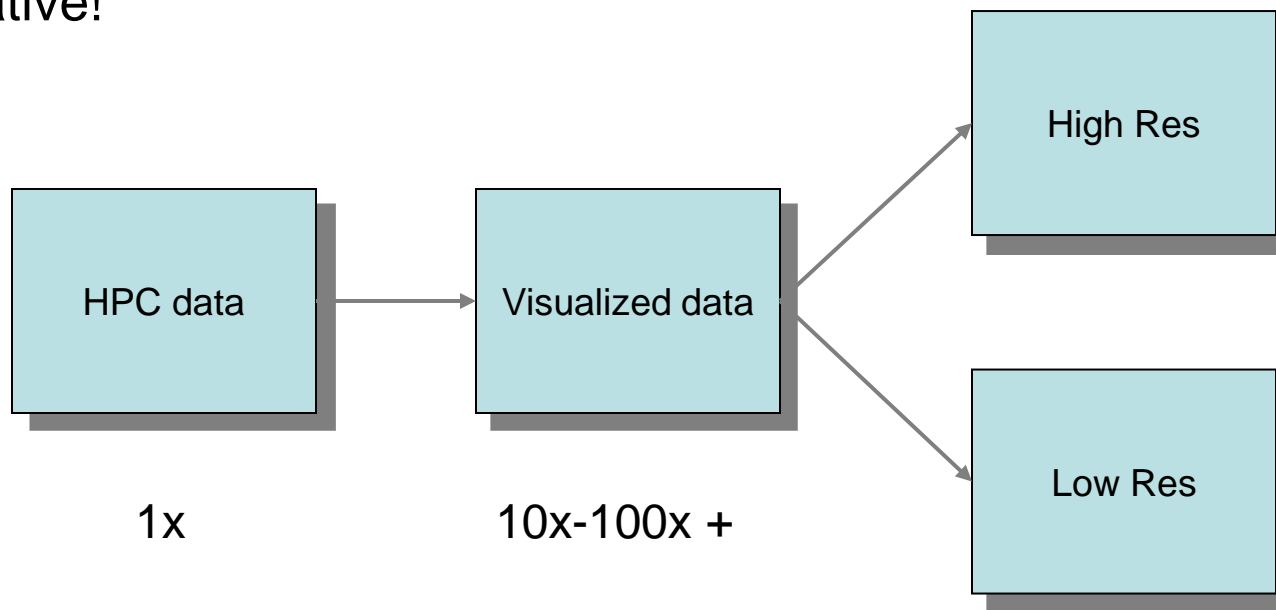
# Large data, Remote Systems
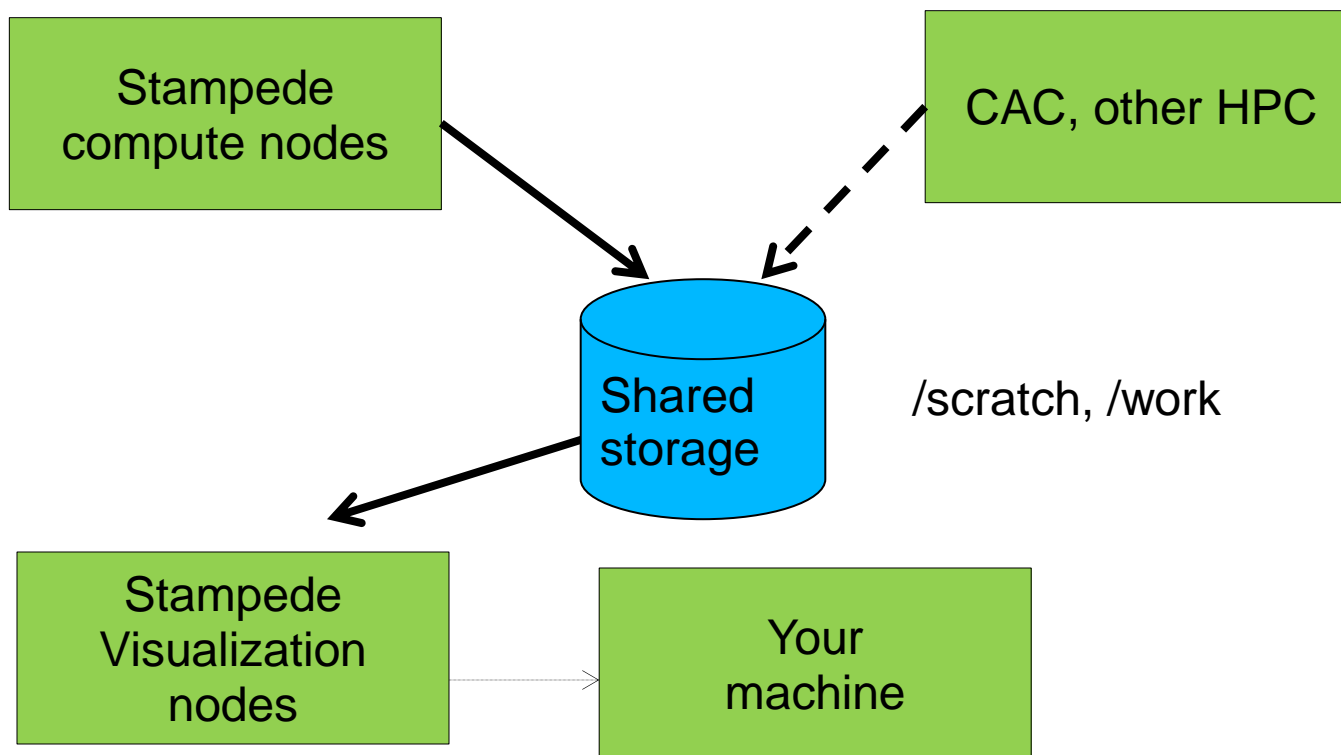
# Visualization nodes

- Stampede:
  - 128 nodes
  - K20 GPU in each node, 8GB GDDR5 RAM
  - viz, gpu queues
  - 32 GB RAM, 16 cores
  - Share Stampede's lustre filesystems
- Longhorn
  - 256 nodes
  - 48GB-144GB RAM, 8 cores per node
  - Dual Nvidia Quadro FX5800 GPUs in each node, 4GB RAM
  - Lustre filesystems separate from Stampede

# Parallel Algorithms: Data Parallelism

## Data parallelism

Data set is partitioned among the processes and all processes execute same operations on the data.

Scales well as long as the data and operations can be decomposed.

### Timesteps

| Processes | 1 | 2 | 3 |
|---|---|---|---|
| 1 | Read partition 1 | Isosurface partition 1 | Render partition 1 |
| 2 | Read partition 2 | Isosurface partition 2 | Render partition 2 |
| 3 | Read partition 3 | Isosurface partition 2 | Render partition 3 |

# Parallel algorithms: What doesn't work

- Streamlines!
    - Not data-parallel
    - Partial streamlines must be passed from processor to processor as the streamline moves from partition to partition
    - No more parallelism available than the number of streamlines!
    - If >1 streamlines pass through the same partition, you may not even get that
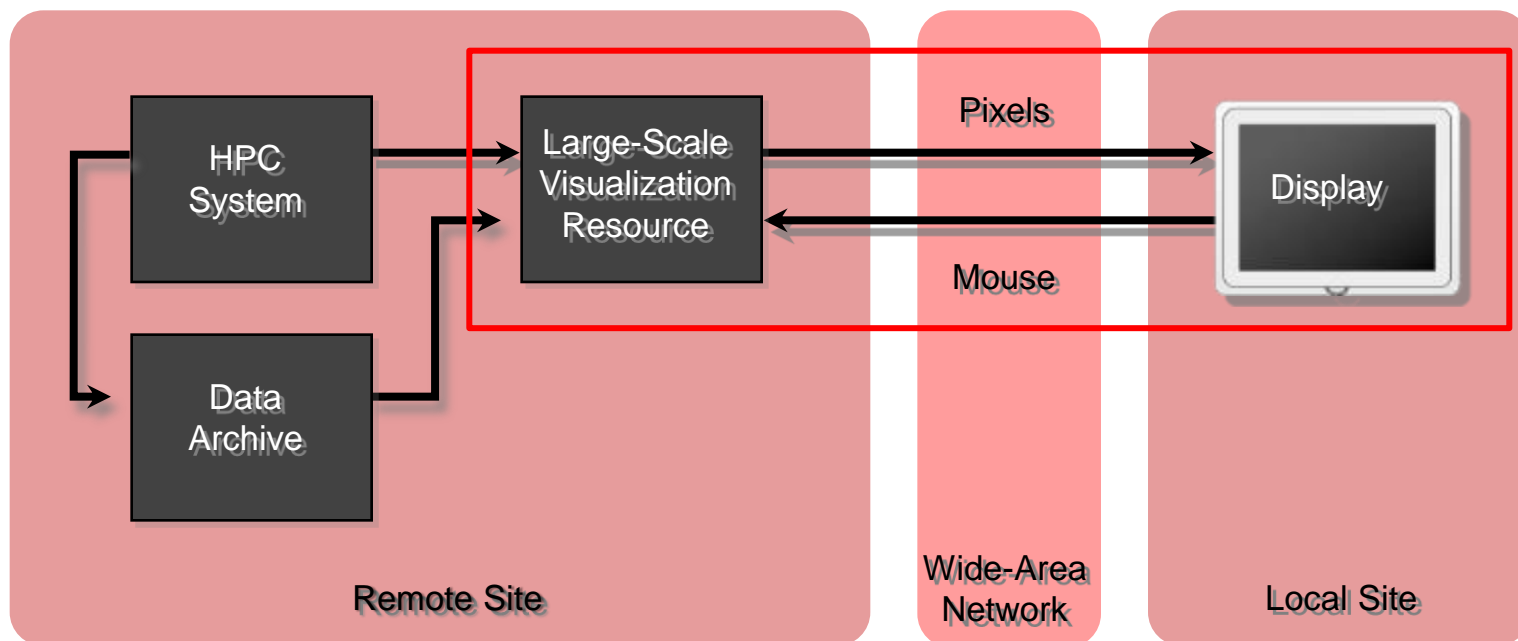
# Rendering

- Many graphics primitives spread out over nodes

- Rendering solutions
  - 1. Gather triangles onto one node, render there
    - Best when there's not a lot of data to render
  - 2. Render triangles in place, gather and Z-composite the results
    - Best when there *is* a lot of data to render
    - Overhead is *almost* independent of data size

- VisIt and ParaView both do it both ways
  - User controls threshold, but both apps aim for reasonable defaults

- Now how do we get rendered graphics to the user?

# Remote Visualization Model

# VNC

- Desktop process runs on remote *server*: vnc session
  - Windows, applications, mouse position
- Rendering occurs on server
  - Render on remote GPU.  Send pixels to client
- Collaboration
  - Many can join vnc session, share control of mouse.
- VNC password to protect *session*
  - Share passwd with collaborators!  Don't use login passwd!!
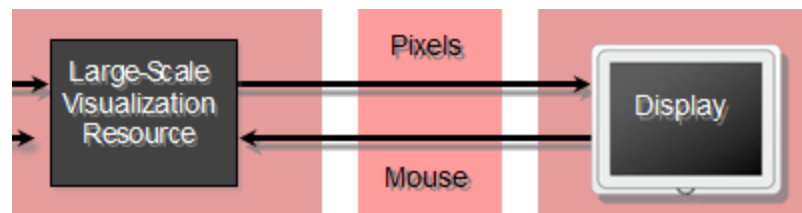
# Visualization Session

1. Allocate set of nodes on visualization system.  This will start a VNC
   server one one node, which you will connect
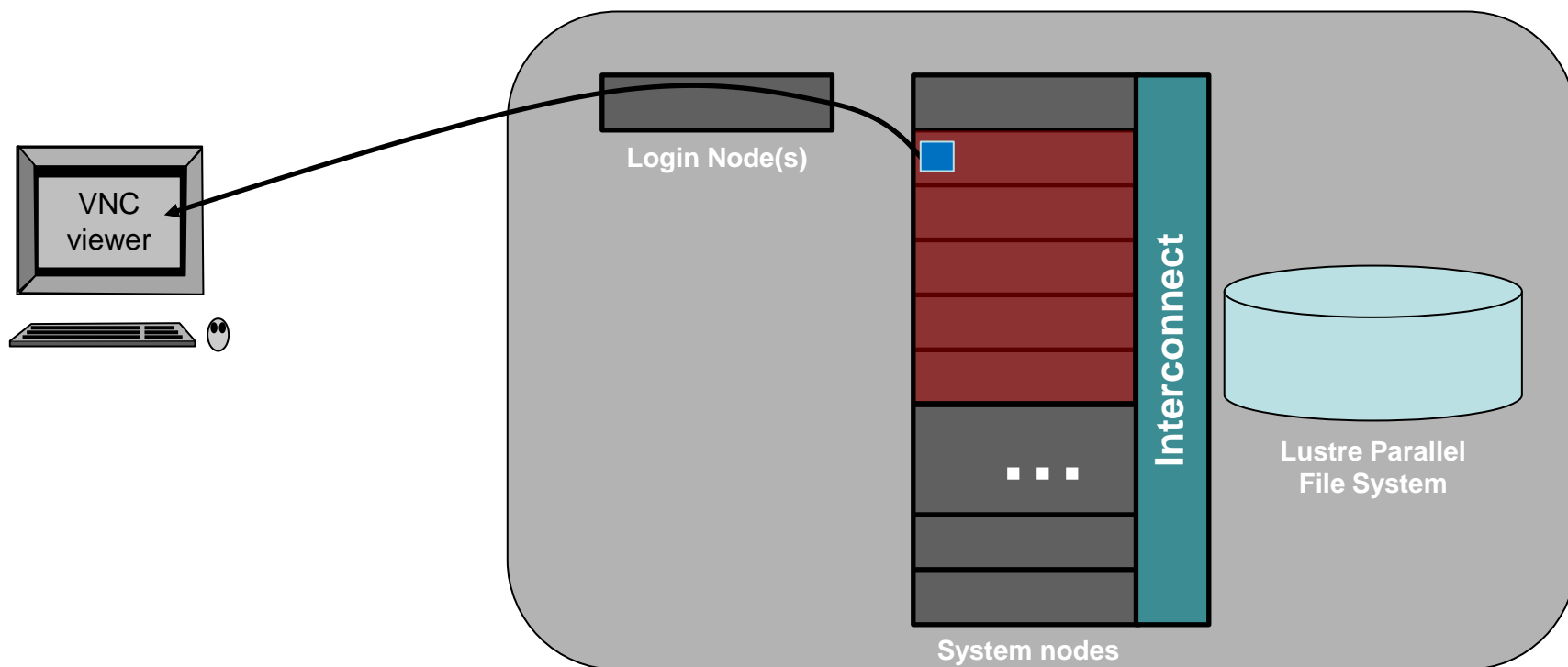
# Start VNC session on Stampede

- Stampede provides convenient job script
  - `sbatch /share/doc/slurm/job.vnc`

- Control node allocation via SLURM params
  - `sbatch -N 8 -A MY_ACCOUNT /share/doc/slurm/job.vnc`

- VNC desktop runs on compute node, private vnc port opened on login nodes just for you

```
local (compute node) VNC port is 5901
got login node VNC port 15754
Created reverse ports on Stampede logins
Your VNC server is now running!
To connect via VNC client:
SSH tunnel port 15754 to stampede.tacc.utexas.edu:15754
Then connect to localhost::15754
```

# Visualization Session

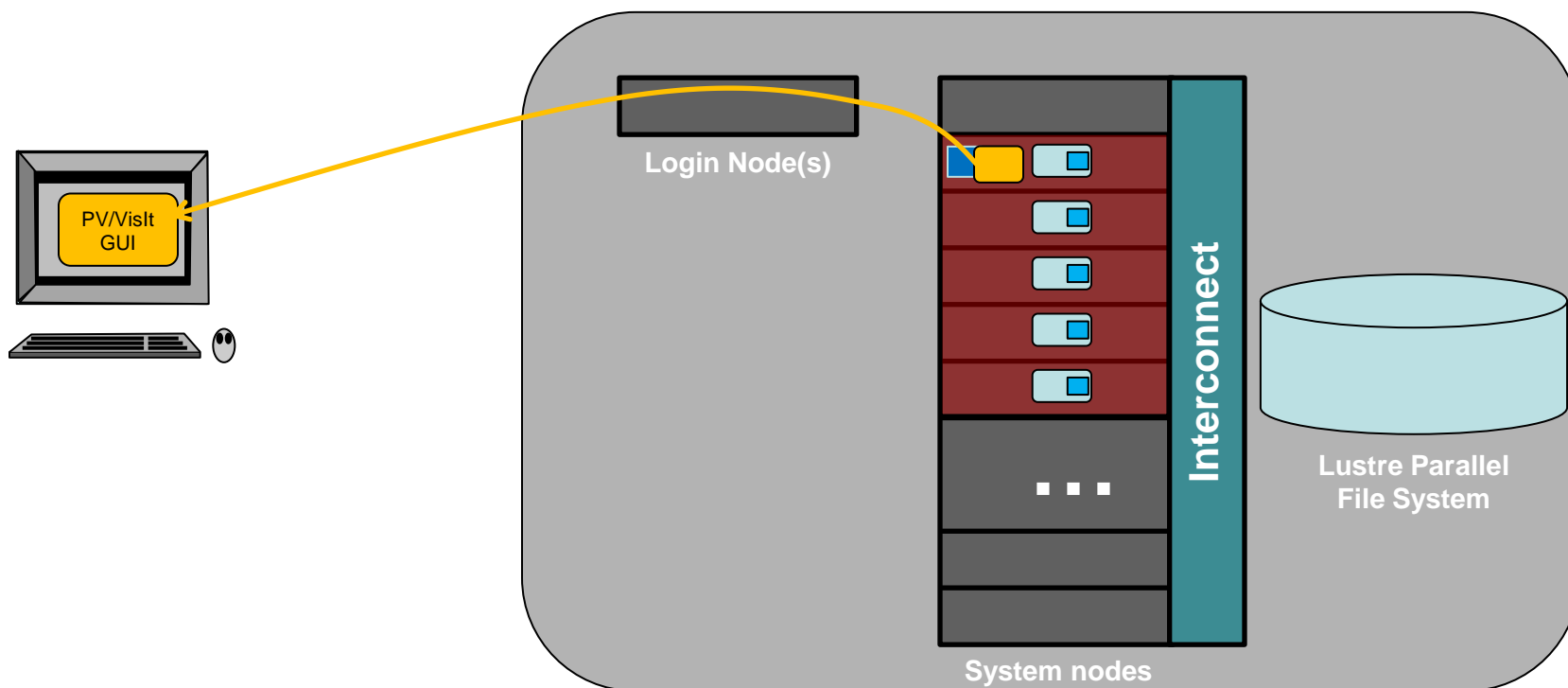2. From that desktop, launch the PV or VisIt Client App
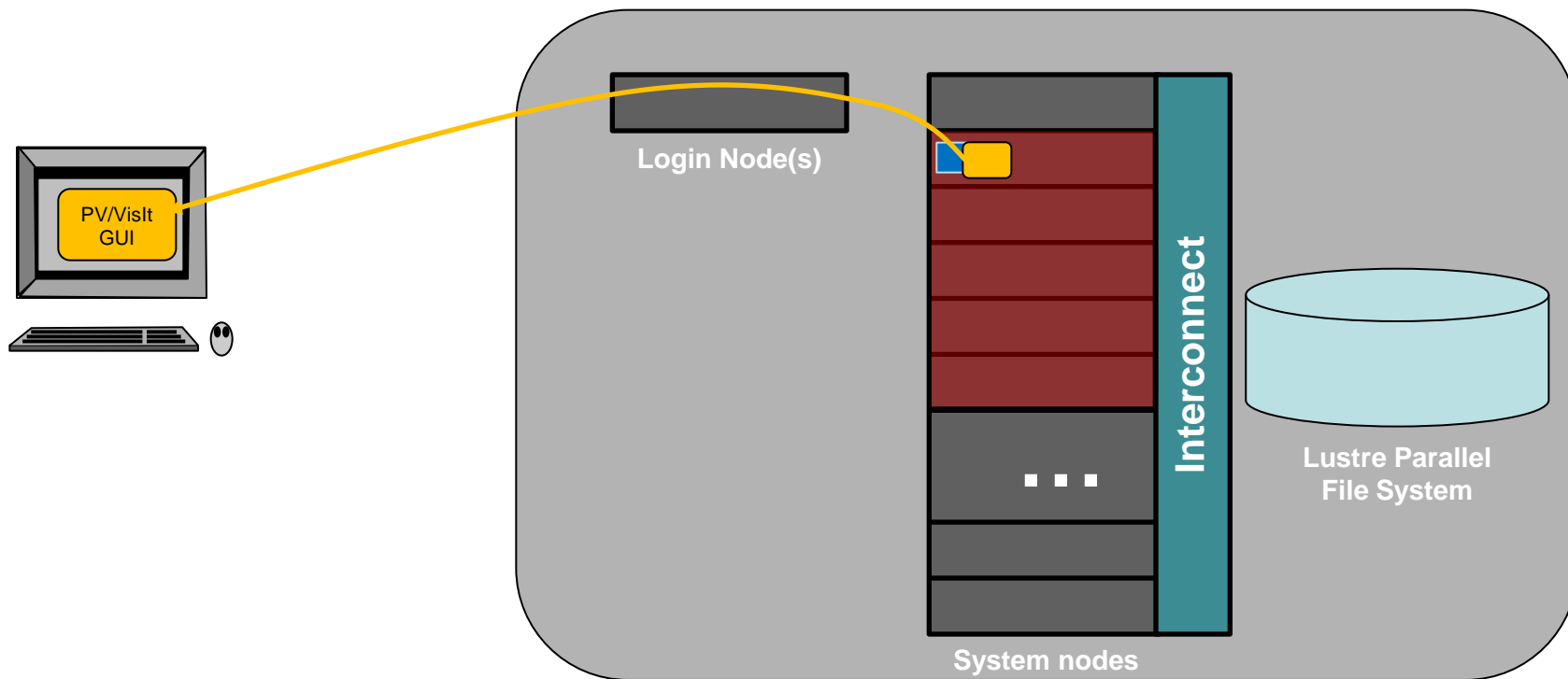
# Visualization Session

3. Start Paraview or VisIt Server Processes
   (Paraview and Visit can do this automatically, with the right configuration)

# Launching Vis Applications

- Applications using openGL (i.e. all visualization apps) need to be wrapped with `vglrun <app>`
  - This is a workaround for the fact that vnc servers do not support openGL natively
- This starts the visualization GUI only.
  - Parallel backends are launched on-demand by visualization app, or manually by user
  - If not using parallel mode, then you're done!
- Visit simply asks if you want parallel or serial mode
  - Params automatically determined by session params
- Paraview needs to be told to run backend processes via `ibrun`

# Visualization Session

4. Multiple processes/node to take advantage of multiple cores/node -- wayness