GPU TECHNOLOGY CONFERENCE

# SCIENTIFIC VISUALIZATION ON GPU CLUSTERS

PETER MESSMER, NVIDIA

# Visualization ≠ Rendering

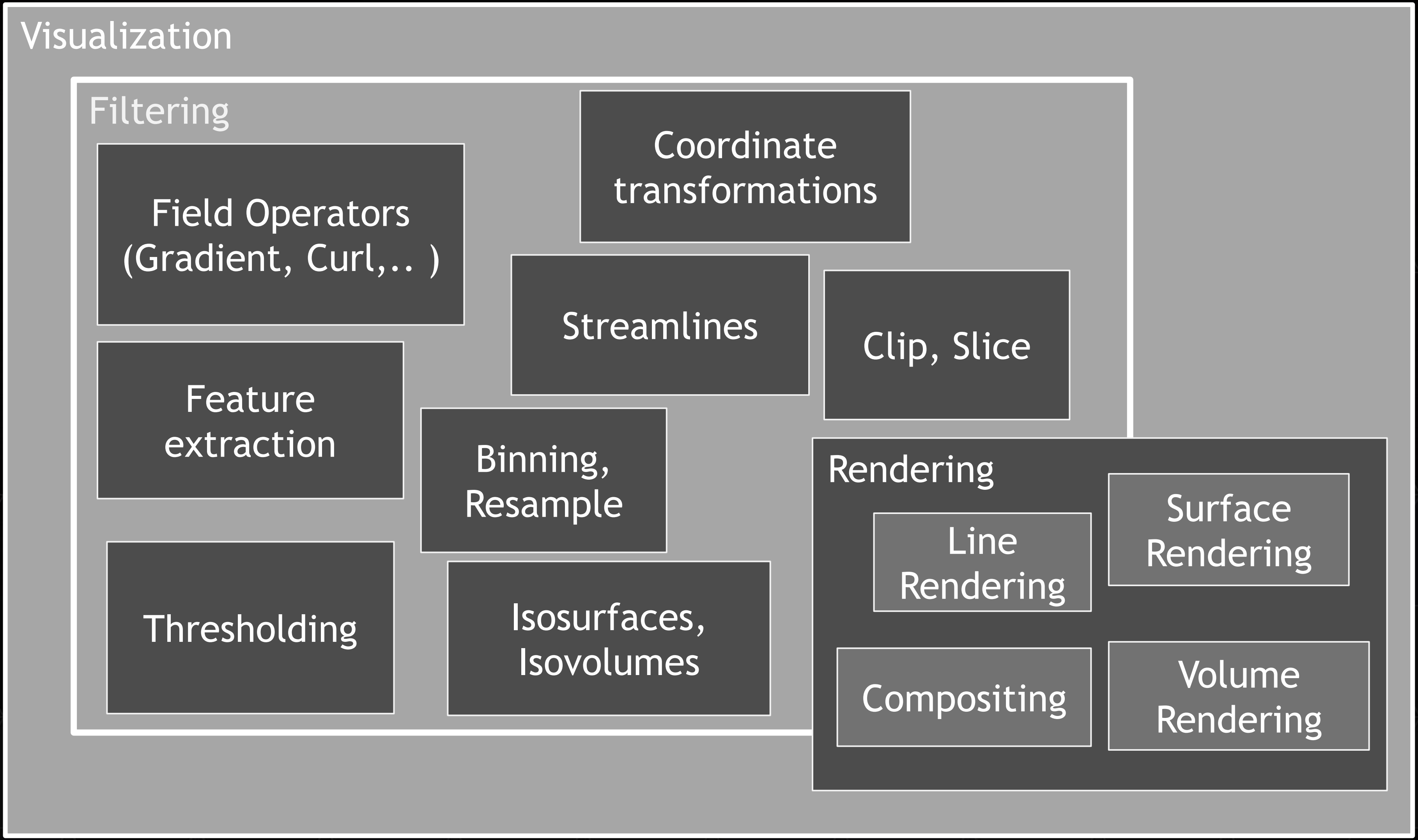
GPU TECHNOLOGY CONFERENCE

## Visualization

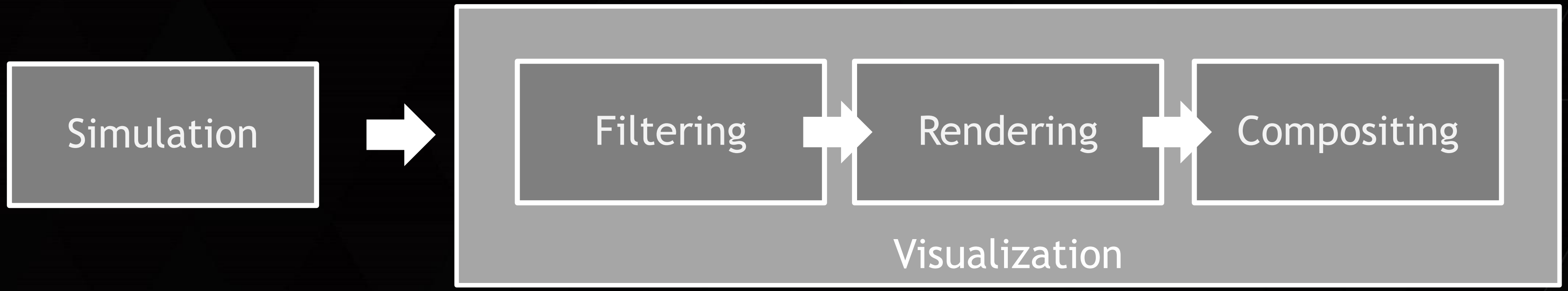
- Isosurfaces, Isovolumes
- Field Operators (Gradient, Curl,.. )
- Coordinate transformations
- Feature extraction
- Clip, Slice
- Streamlines
- Binning, Resample
- Thresholding

### Rendering

- Surface Rendering
- Line Rendering
- Compositing
- Volume Rendering


Visualization
Filtering
Field Operators (Gradient, Curl,.. )
Coordinate transformations
Streamlines
Clip, Slice
Feature extraction
Binning, Resample
Thresholding
Isosurfaces, Isovolumes
Rendering
Line Rendering
Surface Rendering
Compositing
Volume Rendering

# THE VISUALIZATION PIPELINE




- Simulation: Data as needed in numerical algorithm
- Filtering: Conversion of simulation data into data ready for rendering
  - Typical operations: binning, down/up-sampling, iso-surface extraction, interpolation, coordinate transformation, sub-selection, ..
  - Sometimes embedded in simulation
- Rendering: Conversion of shapes to pixels (Fragment processing)
- Compositing: Combination of independently generated pixels into final frame

# PARTICULARITIES IN HPC VIZ

Parallelism

Remoteness


Heterogeneity

# OUTLINE

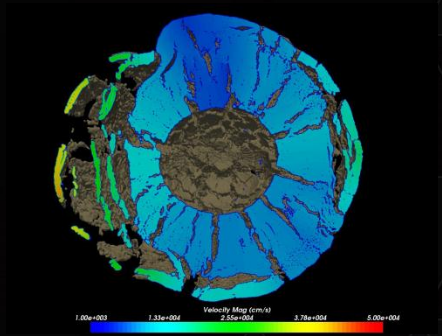
- Tools
  - Paraview, Visit, others
- Rendering
  - Enable HW rendering on Tesla
  - Remote rendering
  - Compositing
  - Delivery
- In-situ visualization

High-level overview. Some parts platform dependent. Check with your sysadmin.

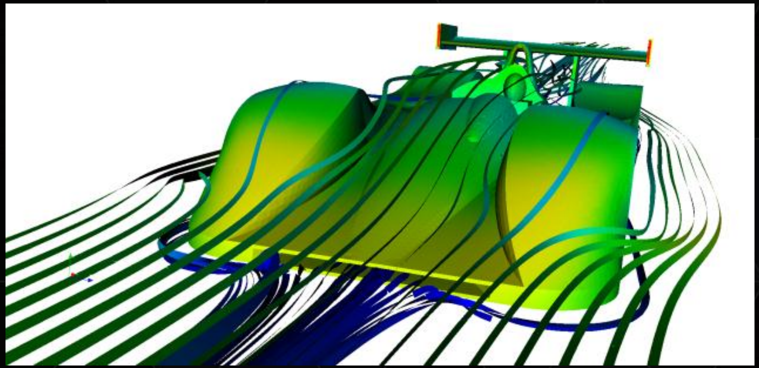
# VISUALIZATION APPLICATIONS

# PARAVIEW

- Scalar, vector and tensor field data features
  - Plots: contour, curve, mesh, pseudocolor, volume,..
  - Operators: slice, iso-surface, threshold, binning,..



- Quantitative and qualitative analysis/vis
  - Derived fields, dimension reduction, line-outs
  - Pick & query



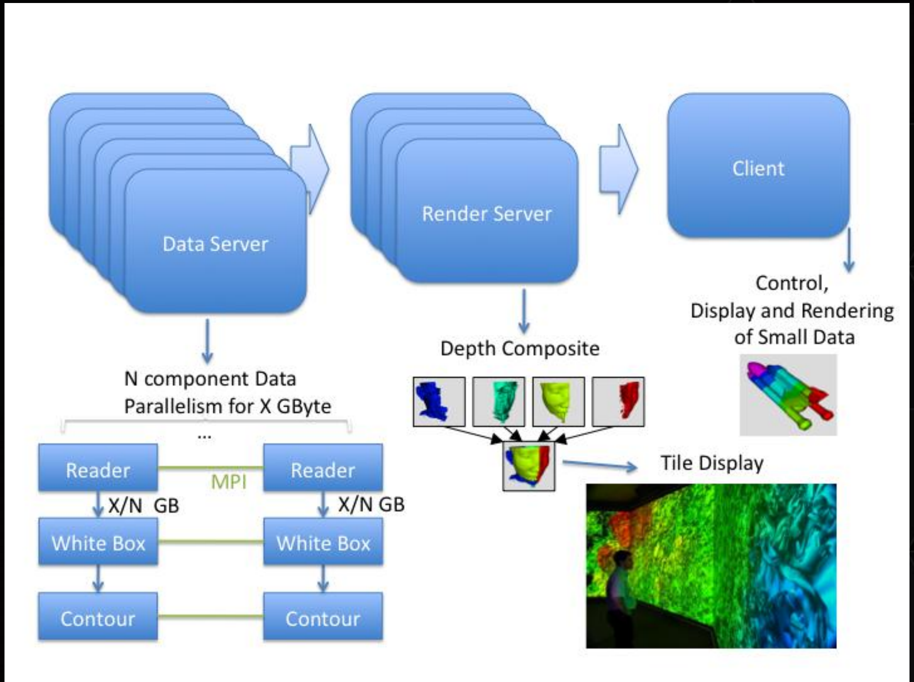
- Scalable architecture
- Developed by Kitware, open source

http://www.paraview.org

# PARAVIEW'S SCALABLE ARCHITECTURE

- Client-server-server architecture
- Server MPI parallel
- Distributed filtering
- GPU accelerated, parallel rendering*

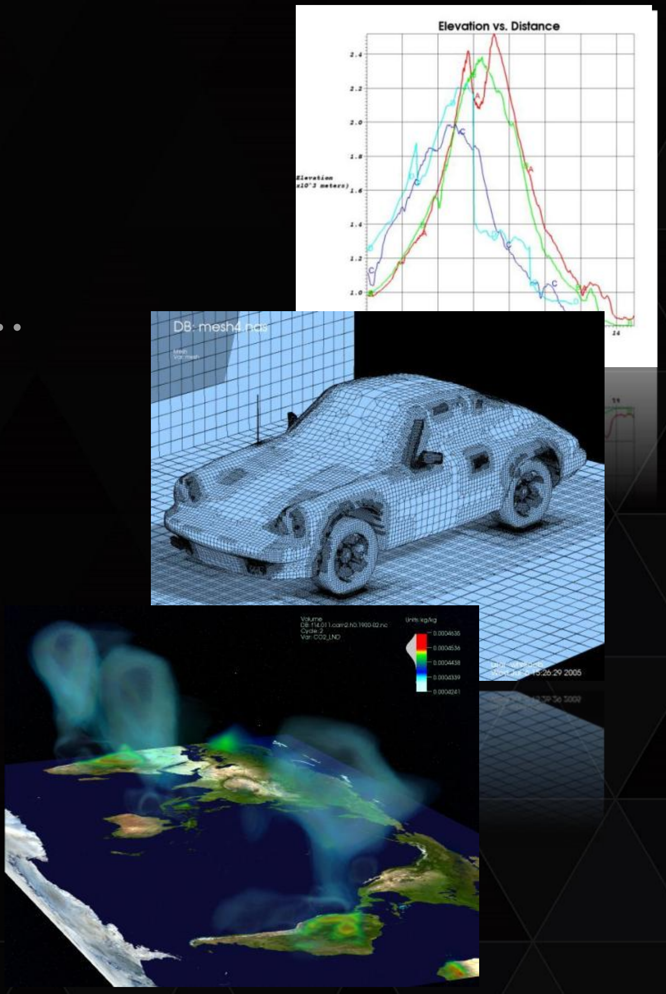
* requires X server on each node



# VISIT

- Scalar, vector and tensor field data features
  - Plots: contour, curve, mesh, pseudo-color, volume,..
  - Operators: slice, iso-surface, threshold, binning,..
- Quantitative and qualitative analysis/vis
  - Derived fields, dimension reduction, line-outs
  - Pick & query
- Scalable architecture
- Open source

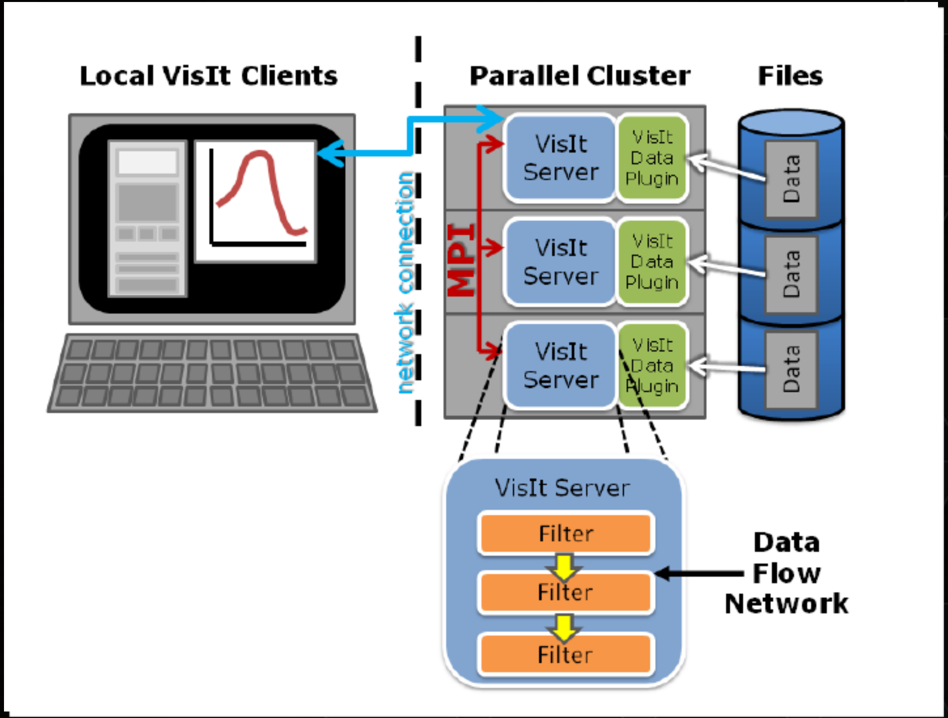
**http://wci.llnl.gov/codes/visit/**



# VISIT'S SCALABLE ARCHITECTURE

- Client-server architecture
- Server MPI parallel
- Distributed filtering
- (multi-)GPU accelerated, parallel rendering*

* requires X server on each node



# SOME OTHER TOOLS

- Wide range of visualization tools
- Often emerged from specialized application domain
  - Tecplot, EnSight: structural analysis, CFD
  - IndeX: seismic data processing & visualization
  - IDL: image processing
- Early adopters of visual programming
  - AVS/Express, OpenDX

# FURTHER READING

- Paraview Tutorial:

http://www.paraview.org/Wiki/The_ParaView_Tutorial

- VisIt Manuals/Tutorials:

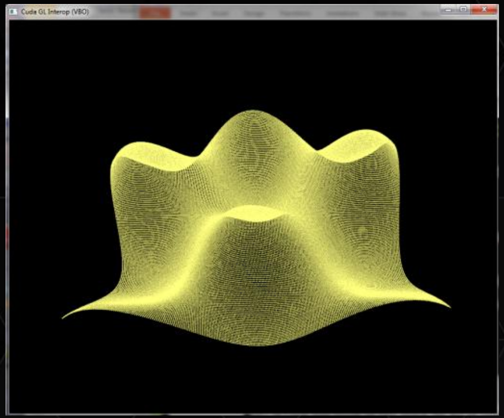
http://wci.llnl.gov/codes/visit/manuals.html

# RENDERING

# OPENGL: API FOR GPU ACCELERATED RENDERING

- Primitives: points, lines, polygons
- Properties: colors, lighting, textures, ..
- View: camera position and perspective
- Shaders: Rendering to screen/framebuffer

- C-style functions, enums



See e.g. “What Every CUDA Programmer Should Know About OpenGL”

(http://www.nvidia.com/content/GTC/documents/1055_GTC09.pdf)

# A SIMPLE OPENGL EXAMPLE

```
glColor3f(1.0f,0,0);

glBegin(GL_QUADS);
    glVertex3f(-1.0f, -1.0f, 0.0f); // The bottom left corner
    glVertex3f(-1.0f, 1.0f, 0.0f);  // The top left corne
    glVertex3f(1.0f, 1.0f, 0.0f);   // The top right corne
    glVertex3f(1.0f, -1.0f, 0.0f);  // The bottom right corner
glEnd();

glFlush();
```

State-based API (sticky attributes)

Drawing

Render to screen

```
glColor3f(1.0f,0,0);

glBegin(GL_QUADS);
    glVertex3f(-1.0f, -1.0f, 0.0f);
    glVertex3f(-1.0f, 1.0f, 0.0f);
    glVertex3f(1.0f, 1.0f, 0.0f);
    glVertex3f(1.0f, -1.0f, 0.0f);
glEnd();

glFlush();
```

?

```
float* vert={-1.0f, -1.0f, ..};
float* d_vert;


cudaMalloc(&d_vert, n);
cudaMemcpy(d_vert, vert, n,
        cudaMemcpyHostToDevice);

renderQuad<<<N/128, N>>>(d_vert);

flushToScreen<<<..>>>();
```

# CUDA-OPENGL INTEROP: MAPPING MEMORY

- OpenGL: Opaque data buffer object (Mention here that this is mainly in "legacy" OpenGL and more control available in modern OpenGL)
  - Vertex Buffer Object (VBO)
  - User has very limited control

- CUDA: C-style memory management
  - User has full control

- CUDA-OpenGL Interop:

  Map/Unmap OpenGL buffers into CUDA memory space

```
cudaGraphicsGLRegisterBuffer(cuda_vbo, *vbo, flags);
cudaGraphicsMapResources(1, cuda_vbo, 0);
```

# CAN ALL GPUS SUPPORT OPENGL?

- GeForce : standard feature set including OpenGL 4.5
- Quadro : + certain highly accelerated features (e.g. CAD)

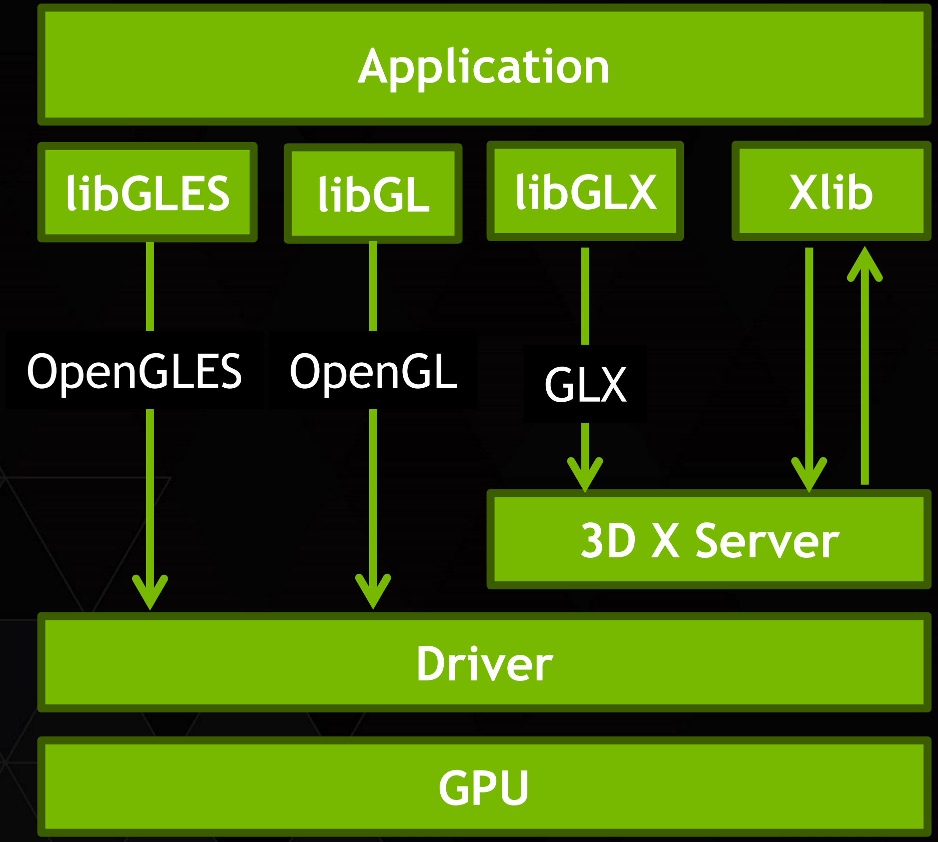
- Tesla, K20X,m: Requires setting GPU Operation Mode (GOM) to “All on”
- Tesla K40, K80: GOM “All on” by default

```
nvidia-smi --query-gpu=gom.current
nvidia-smi -q
```

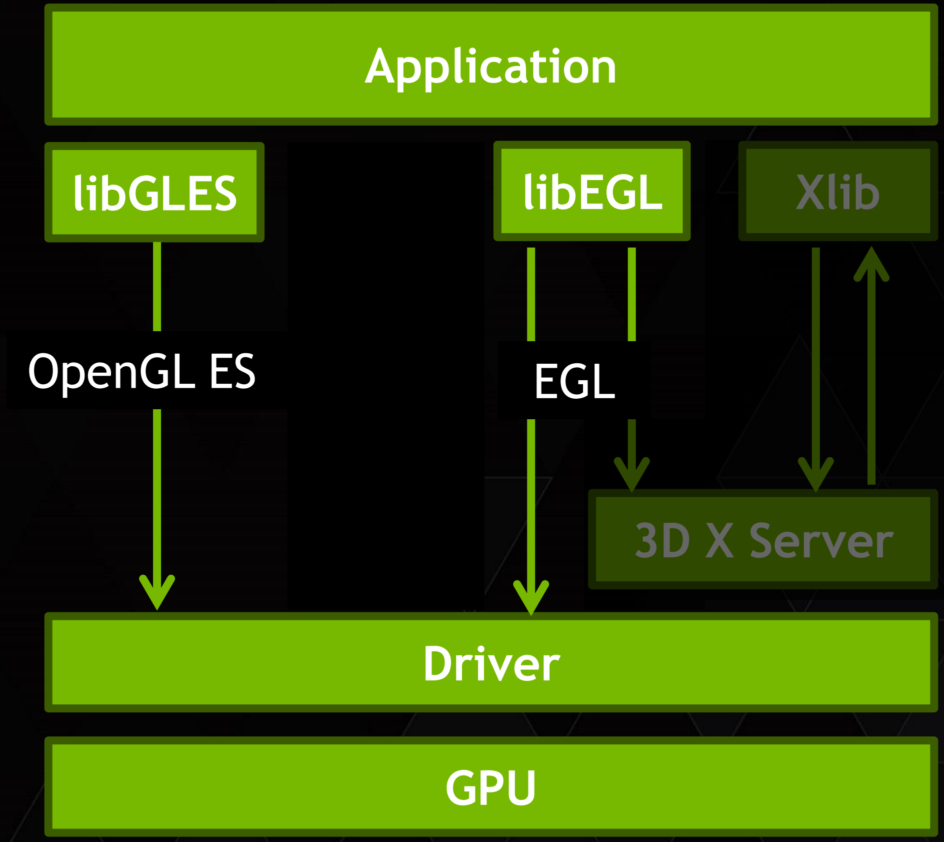
# OPENGL CONTEXT

- State of an OpenGL instance
  - Incl. viewable surface
  - Interface to windowing system
- Context creation: platform specific
  - Not part of OpenGL
  - Handled by Xserver in Linux/Unix-like systems
  - New alternative: EGL (currently: OpenGL ES only)
- GLX: Interaction X<-> OpenGL

# OPENGL ON HEADLESS SERVERS



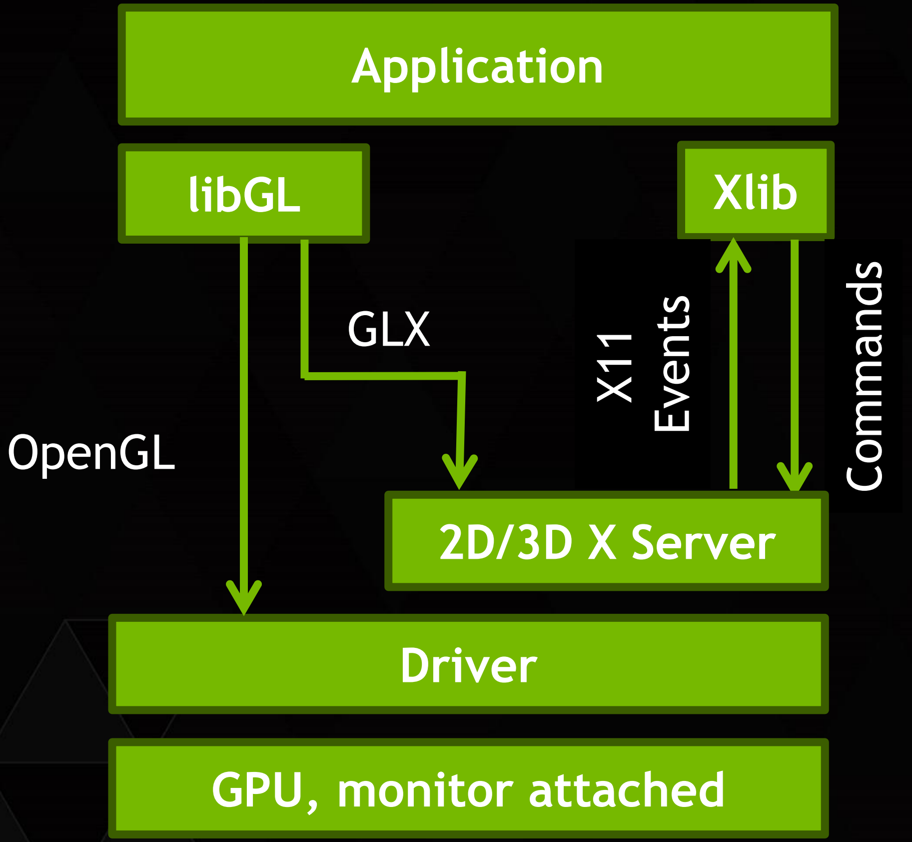

Current State



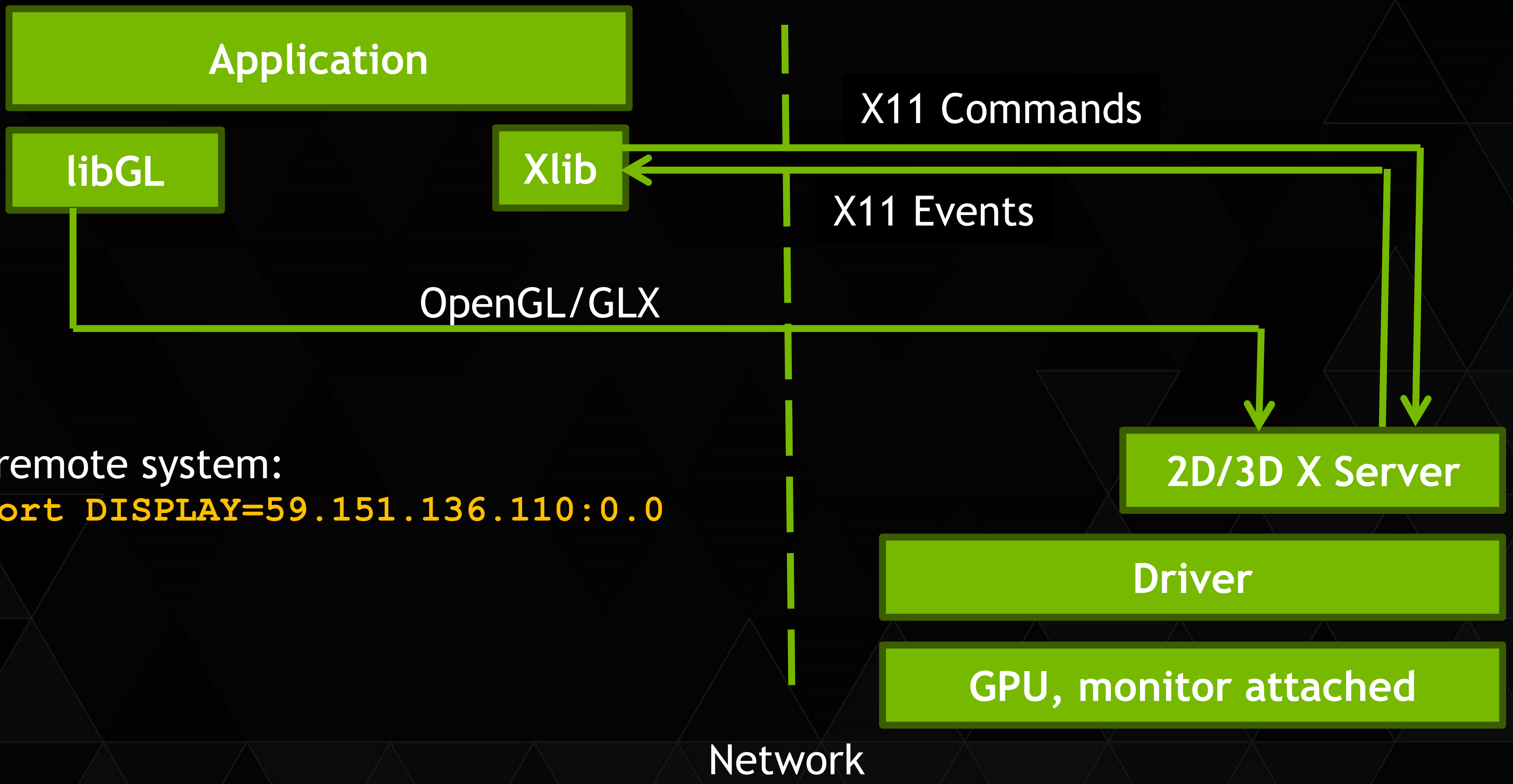

Current State for OpenGL ES applications

# REMOTE RENDERING

# LOCAL RENDERING



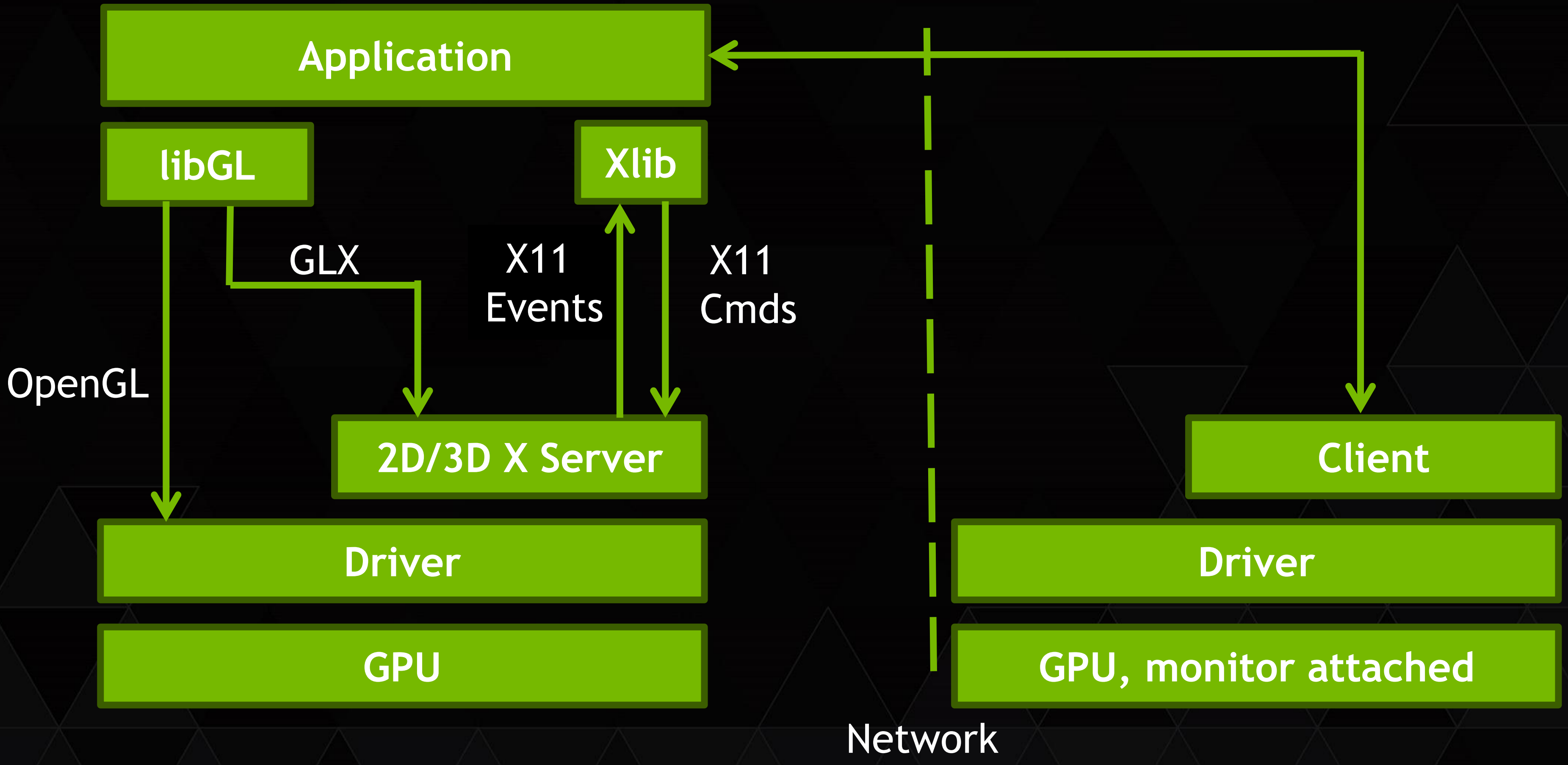
# X-FORWARDING: THE SIMPLEST FORM OF "REMOTE" RENDERING



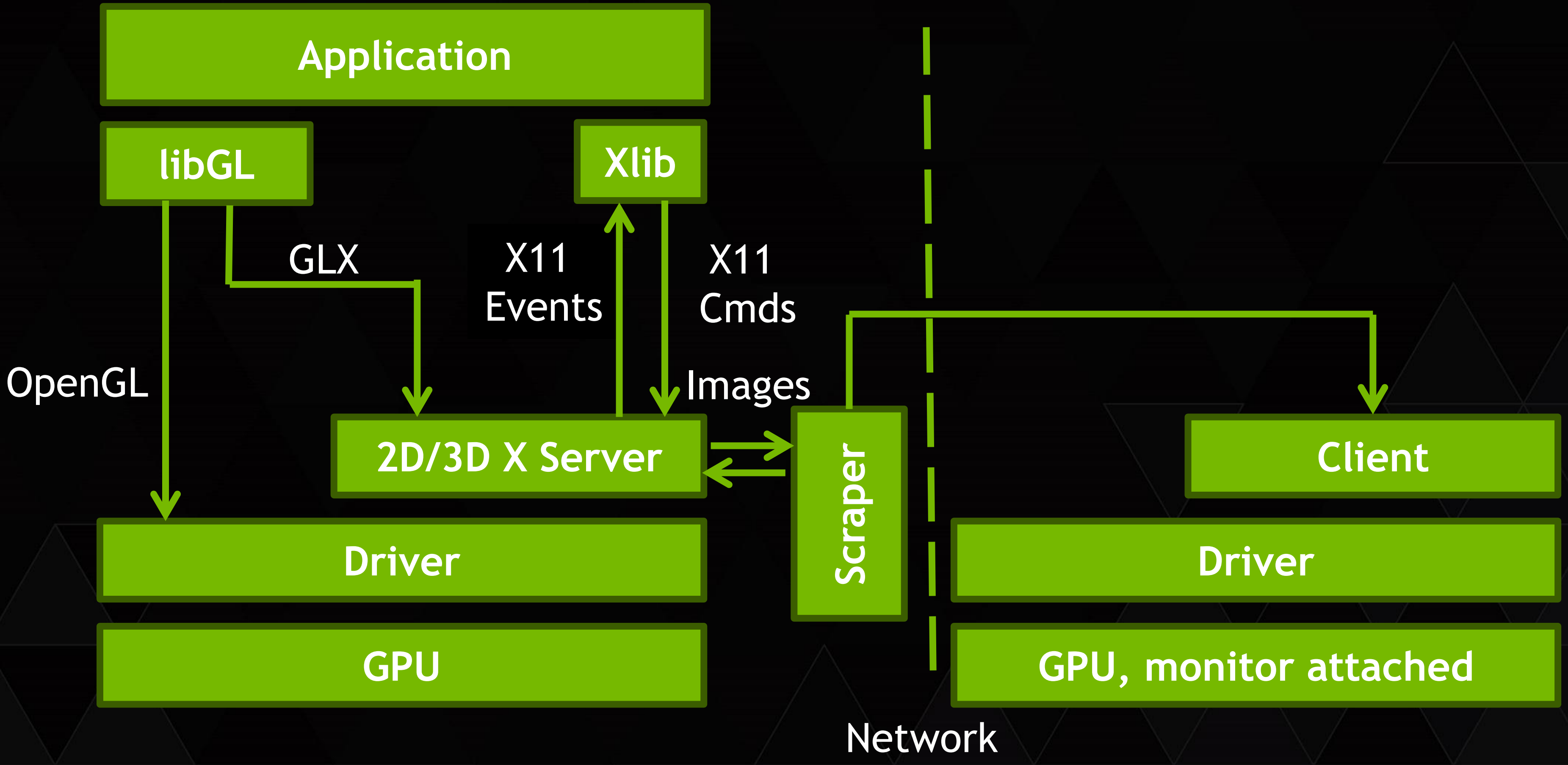

On remote system:

```
export DISPLAY=59.151.136.110:0.0
```

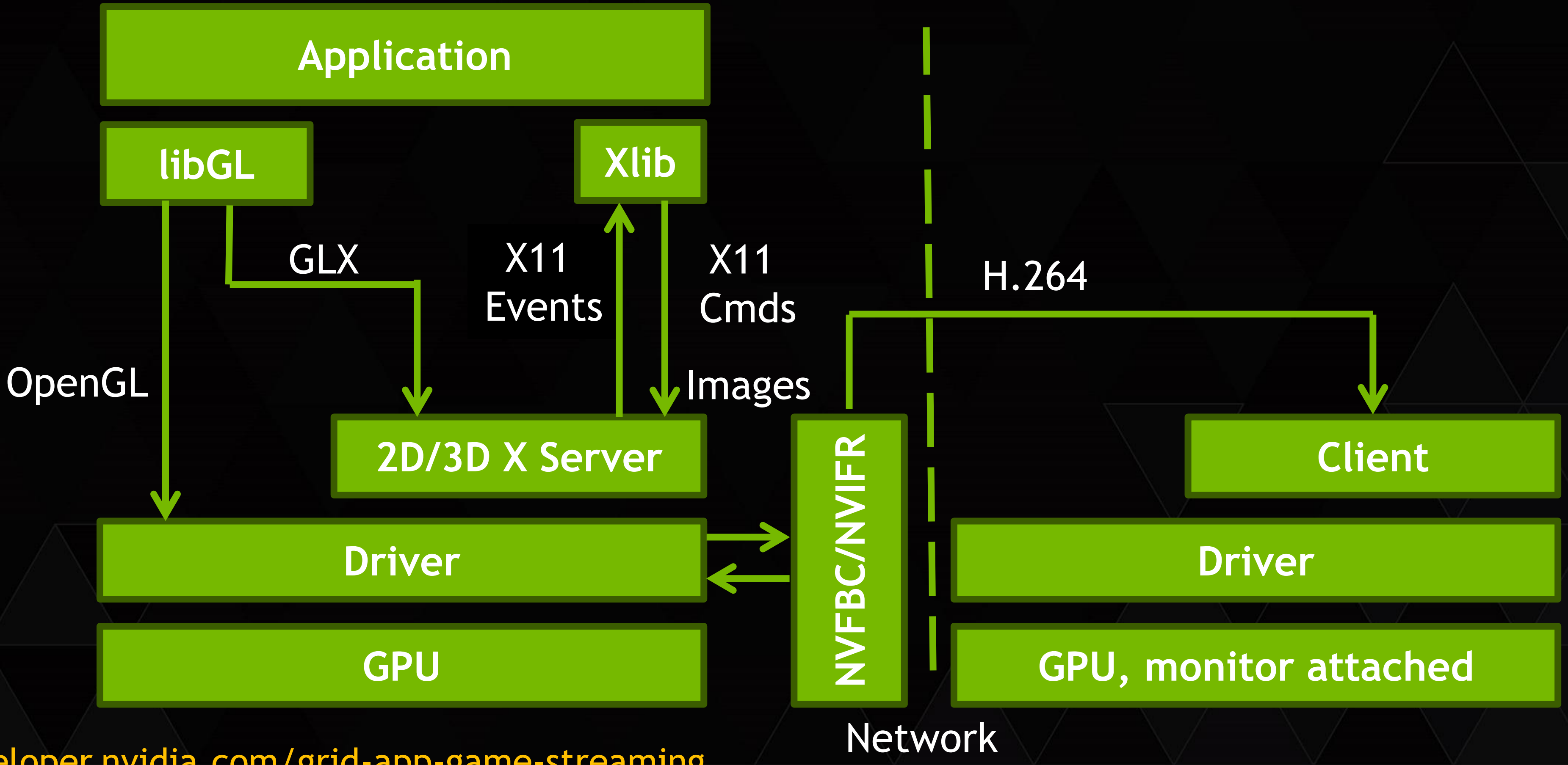
# SERVER-SIDE RENDERING + REMOTE VIZ APPLICATION



# SERVER-SIDE RENDERING + SCRAPING

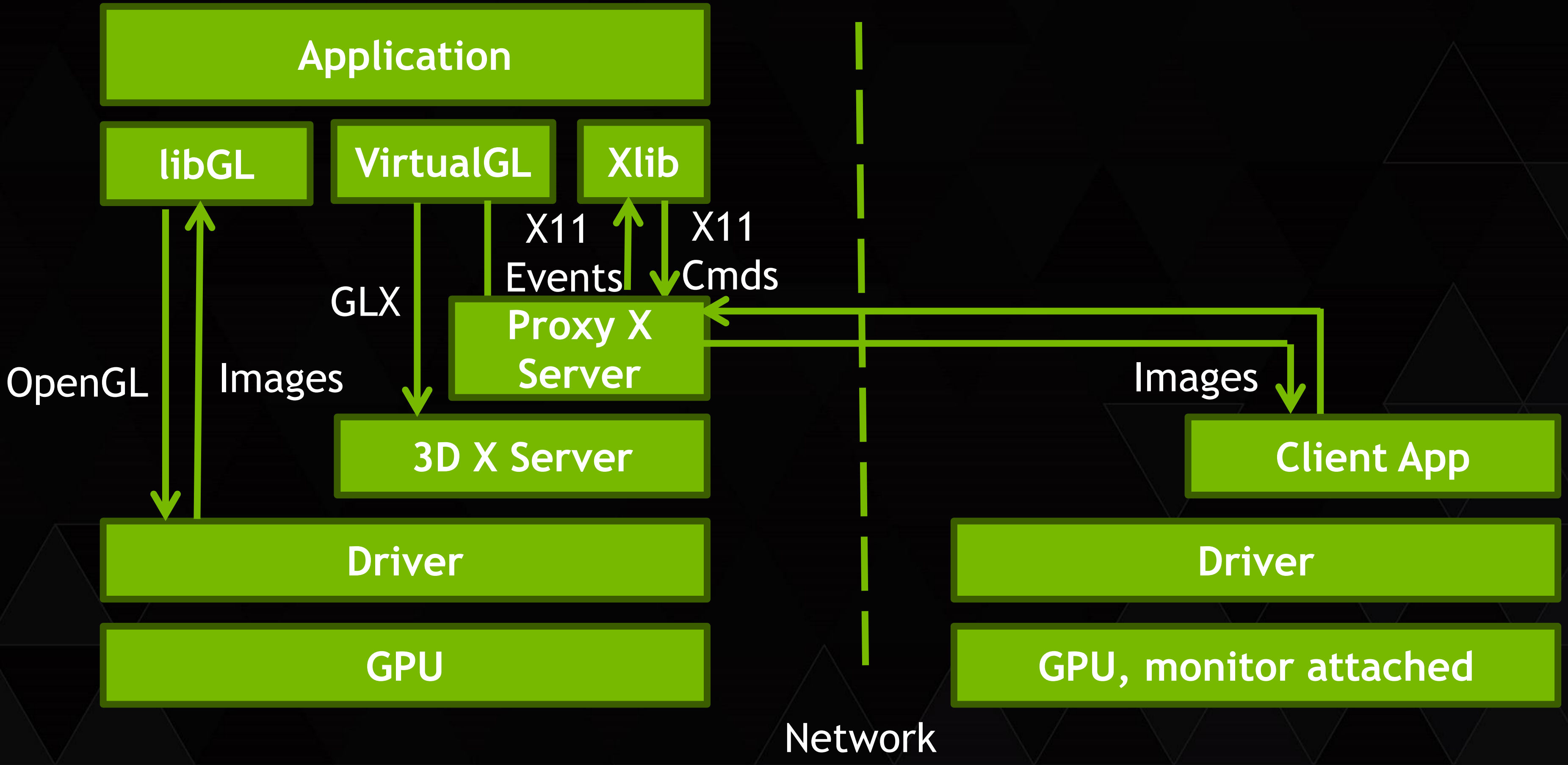


# CAPTURE/ENCODING WITH NVFBC/NVIFR




https://developer.nvidia.com/grid-app-game-streaming
https://developer.nvidia.com/nvidia-video-codec-sdk

# GLX FORKING WITH INTERPOSER LIBRARY



# EXAMPLE OF REMOTING SOLUTIONS

TurboVNC + VirtualGL

+ Open source solution
+ Compressed image transport
+ Remote GPU accelerates OpenGL

http://www.virtualgl.org

NICE DCV

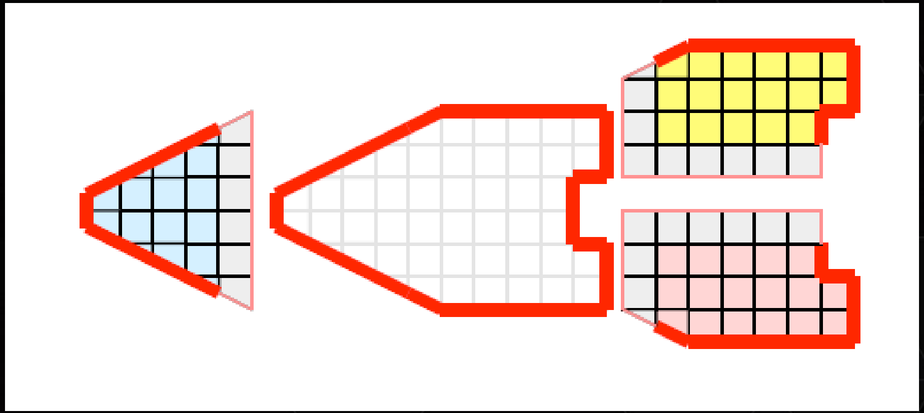
+ Commercial grade product
+ H264 encoded video stream

http://www.nice-software.com/products/dcv

# PARALLEL VISUALIZATION

# PARALLEL VISUALIZATION

- Domain decomposition
- Parallelism at multiple levels
  - Filtering
  - Rendering



- Both supported by VisIt & Paraview
  - Heavy lifting already done!

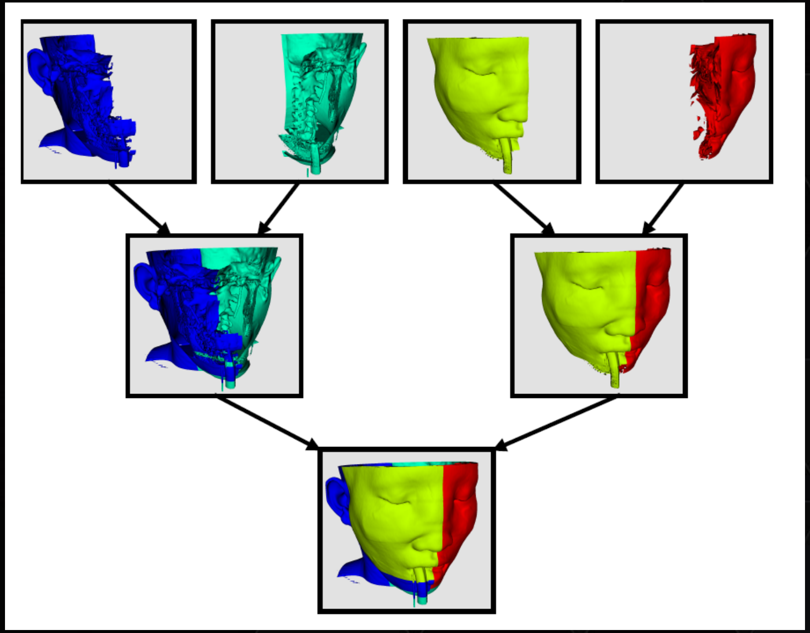
- Typically biggest challenge: Setup in parallel environment
  - Both tools provide support for most common cases
  - Both VisIt & Paraview MPI parallel -> need custom build

# PARALLEL COMPOSITING WITH ICET

- Each node renders fraction of image
- Sort last compositing
- Highly scalable

- Widely used (Paraview, VisIt .. )

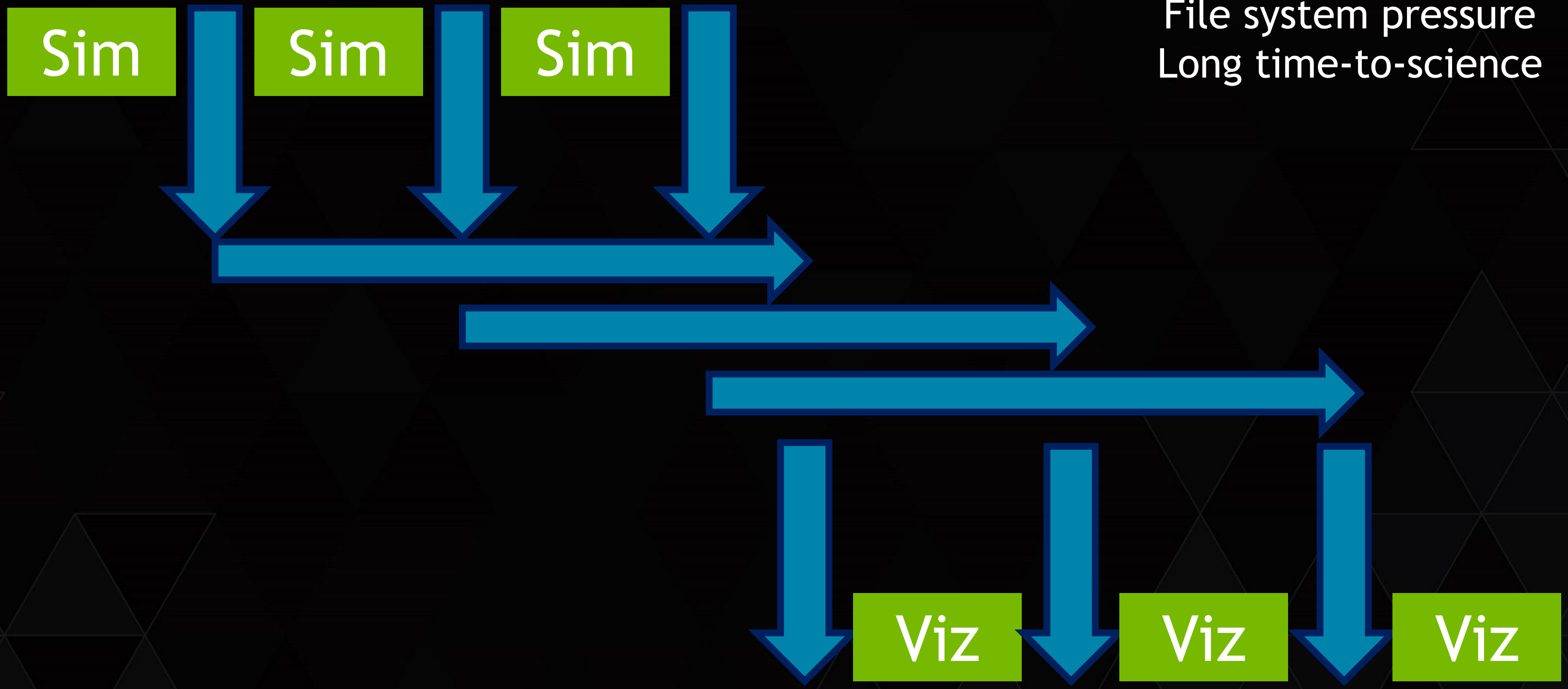
http://icet.sandia.gov



GPU TECHNOLOGY CONFERENCE

# IN-SITU VISUALIZATION

# LEGACY WORKFLOW

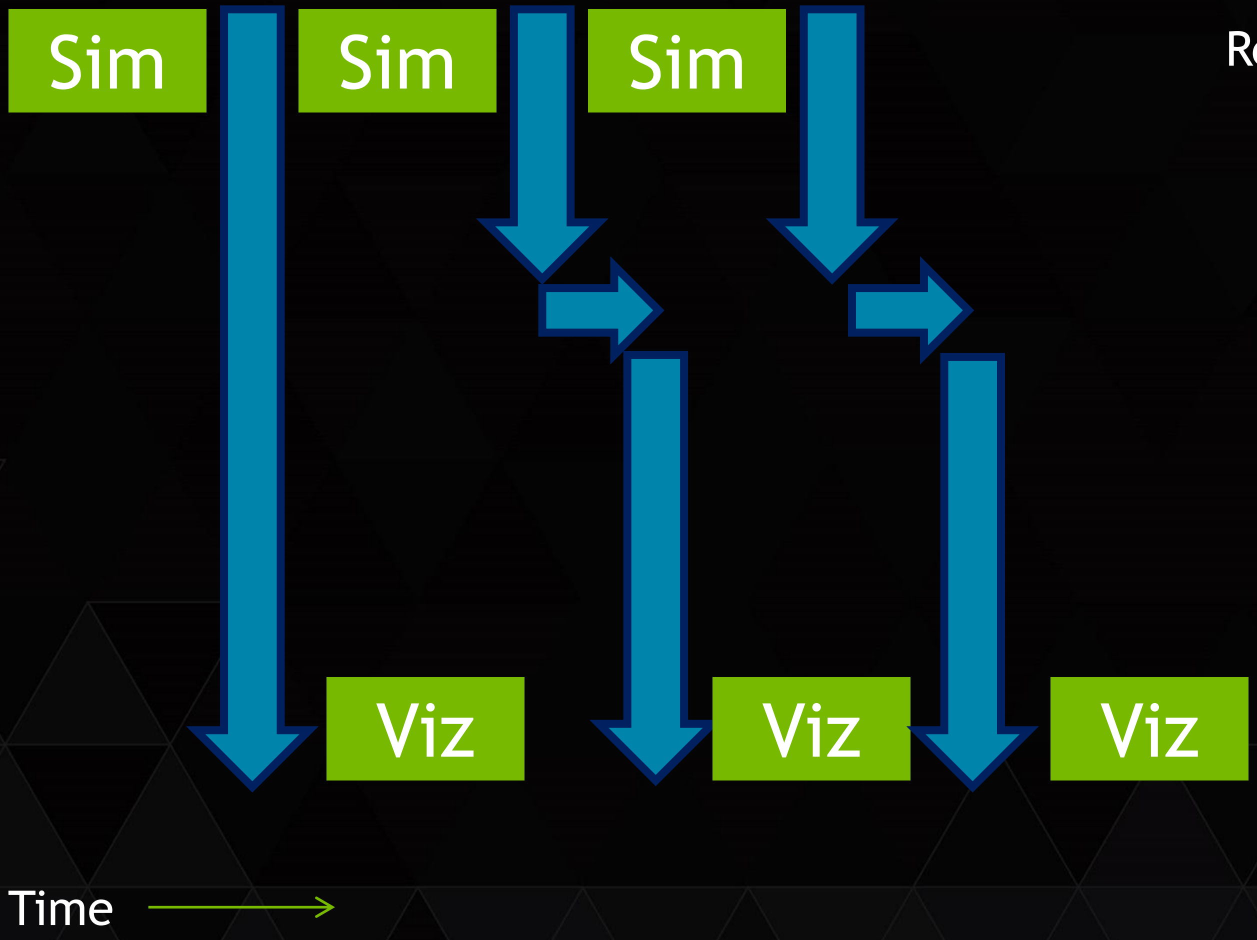



File system pressure
Long time-to-science

Time →

GPU TECHNOLOGY CONFERENCE

# PIPELINED IN-SITU ANALYSIS

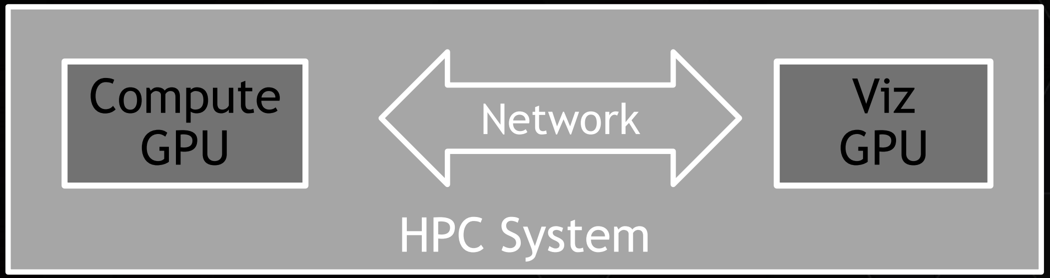



Reduced file system consumption
Early detection of faulty runs
Reduced

# DIFFERENT VISUALIZATION SCENARIOS

- 1) Legacy workflow
  - Separate compute & viz system
  - Communication via filesystem


- 2) Partitioned HPC system
  - Different nodes for viz & compute
  - Communication via High Perf Network



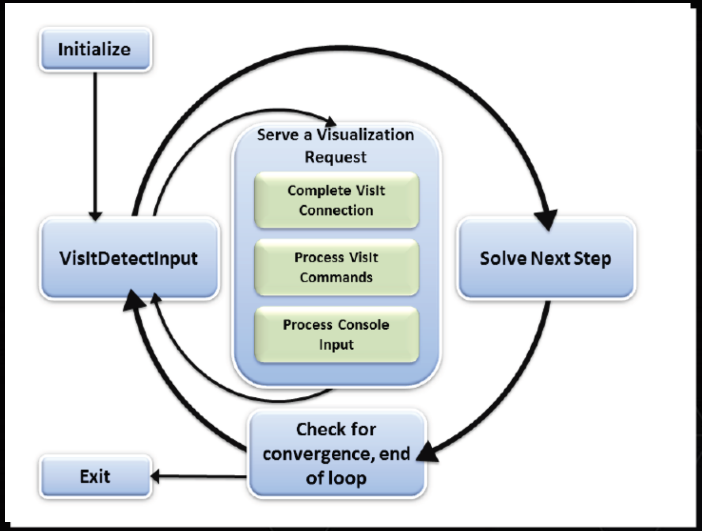

- 3) Co-Processing
  - Compute on GPU, viz on CPU
  - Compute and visualization on same GPU

# SUPPORT FOR IN-SITU VISUALIZATION

- Paraview: Catalyst
- VisIt: LibSim

- Instrument application to expose data
  - Paraview: Adaptor
  - LibSim: Data access callbacks

- Potential for interactive steering




S5815, today, 3:30: Programming Pointers to Optimize your In-Situ Visualization Pipeline
S5710, Fri 9:30: In-Situ Data Analysis and Visualization: ParaView, Calalyst and VTK-m

# SUMMARY

- Visualization is more than rendering
  - Filtering often expensive component
  - Fast rendering can enable new applications (interactive supercomputing)
- Tesla systems can be used for rendering
  - On some systems requires GOM=All ON
  - Xserver or EGL
- Remote visualization
  - Supported by common tools
  - Take advantage of available hardware (H264 encoder)
- Parallel Rendering
  - Use IceT for compositing
- In-Situ Visualization
  - Supported by common tools

GPU TECHNOLOGY CONFERENCE

# THANK YOU

JOIN THE CONVERSATION

#GTC15