

Equalizer

Parallel OpenGL Application Framework

Outline

- Overview
 - High-Performance Visualization
 - Equalizer
 - Competitive Environment
- Equalizer
 - Features
 - Scalability
 - Outlook

HPV

- High-Performance Visualization - like HPC but for interactive 3D applications
- Address the demand to visualize huge data sets using COTS clusters
- Issue is to *scale* rendering performance using multiple GPU's and CPU's

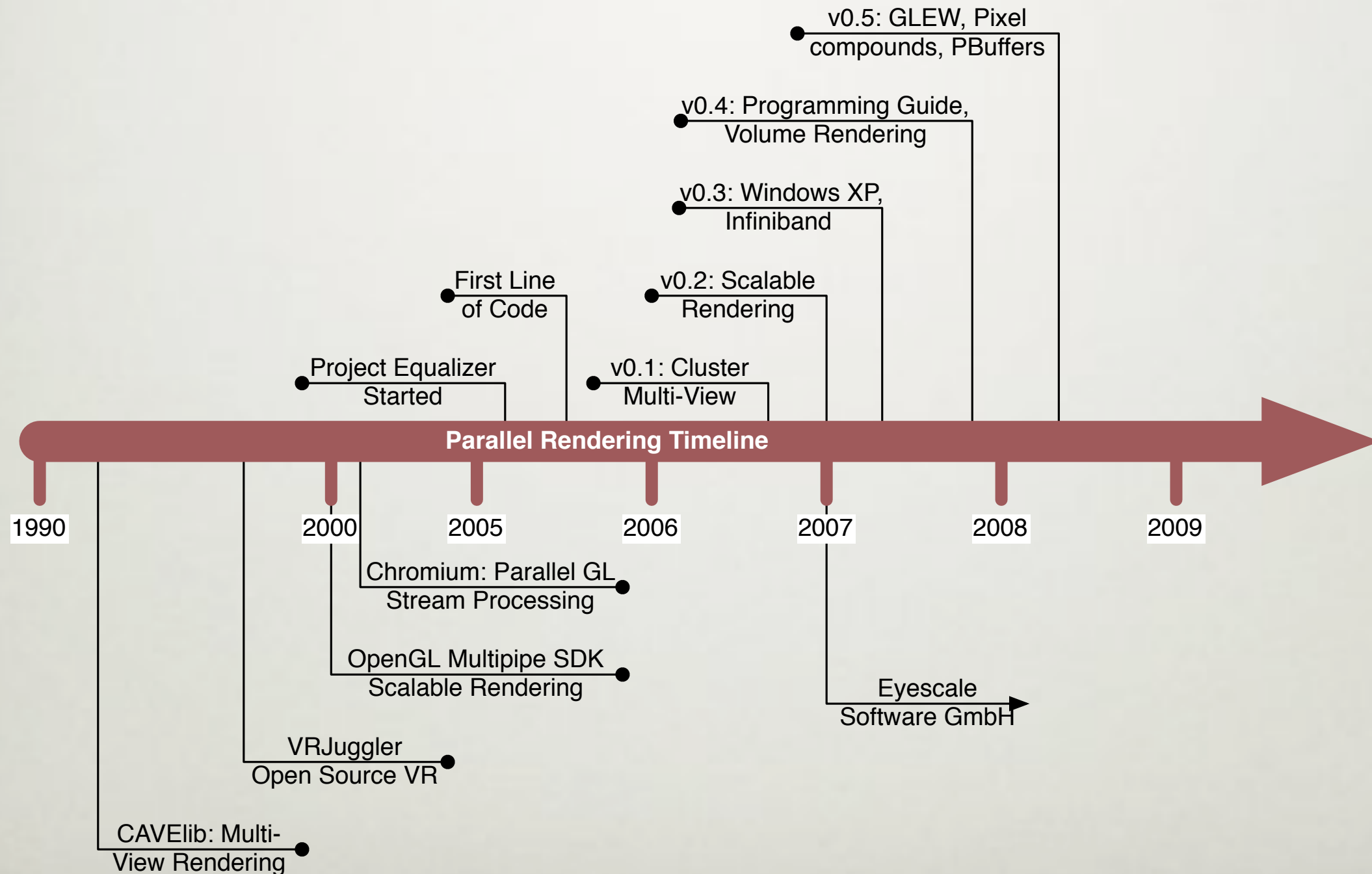
HPC Analogy

	HPC	HPV
What?	Parallel computation across multiple CPU's	Parallel 3D rendering across multiple GPU's and CPU's
How?	Mostly non-interactive batch processing	Highly interactive, real-time rendering
Hardware	Cluster or Supercomputers typically using fast interconnects	Graphics Cluster, Supercomputers, display hardware, input devices

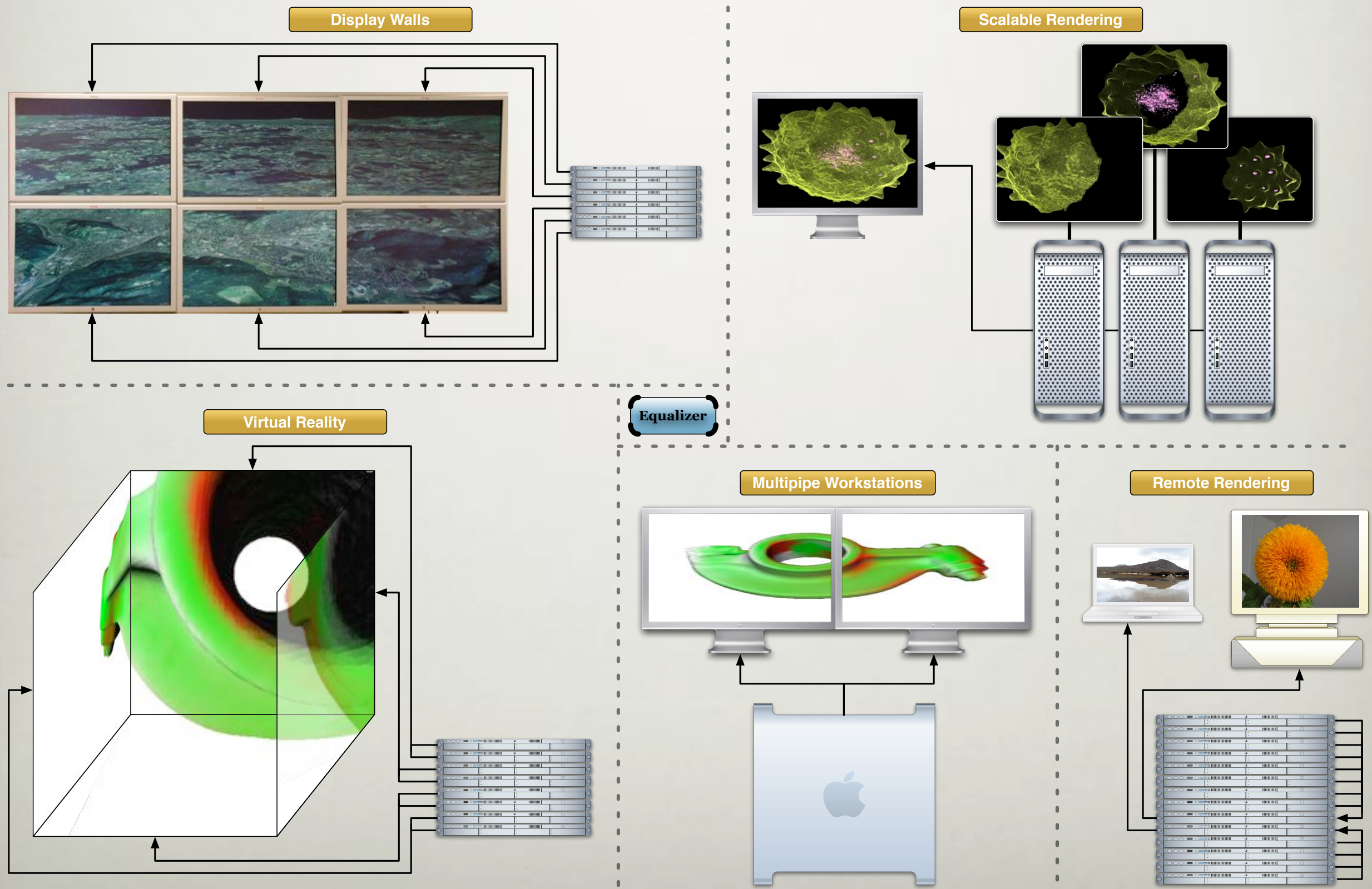
Equalizer

“GLUT for multi-GPU systems and
visualization clusters”

History



Selected Use Cases



Competitive Environment

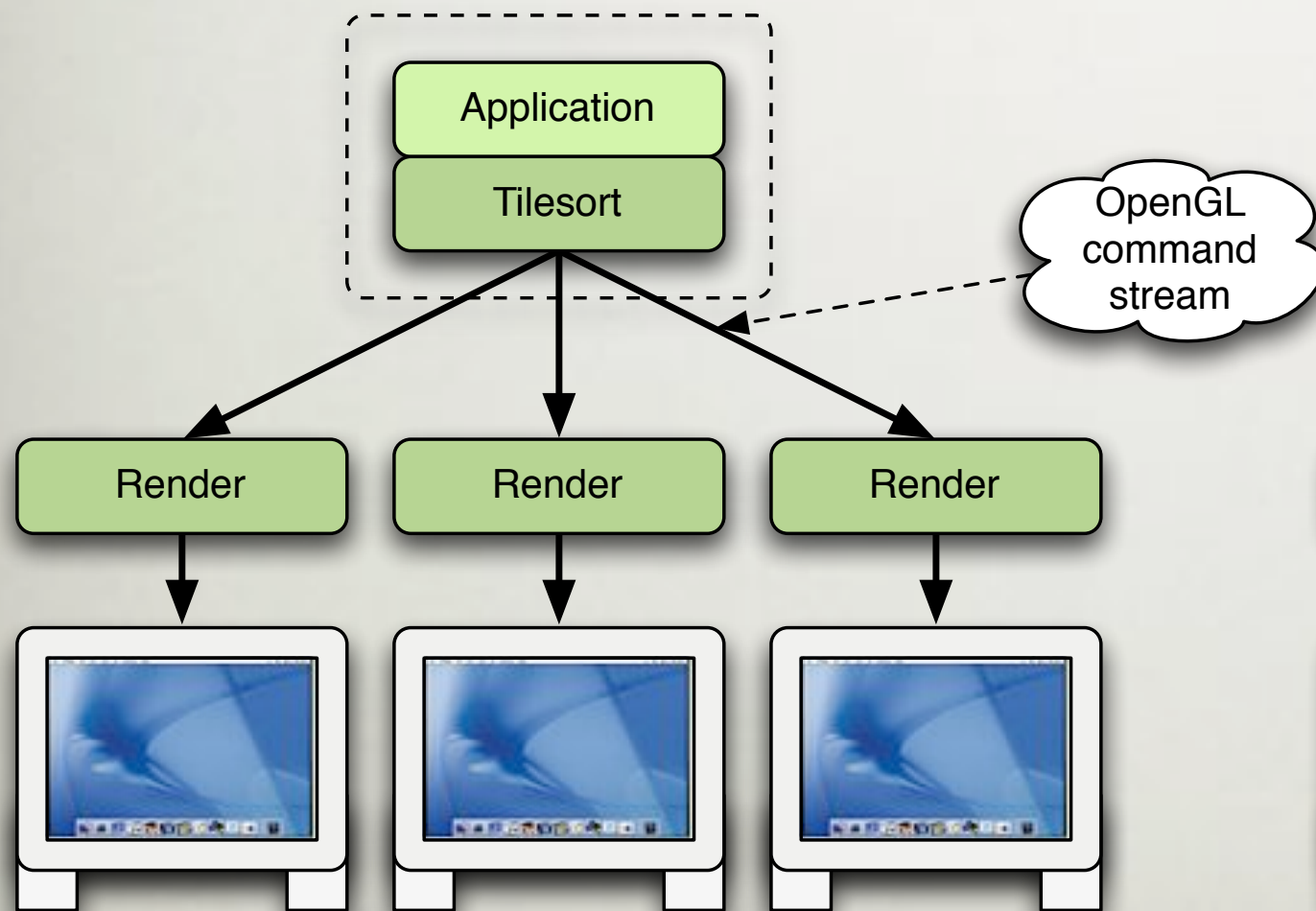
- Transparent solutions
 - Based on OpenGL interception
- Programming interfaces
 - Distributed Scene Graphs
 - Middleware

HPV Transparent Solutions

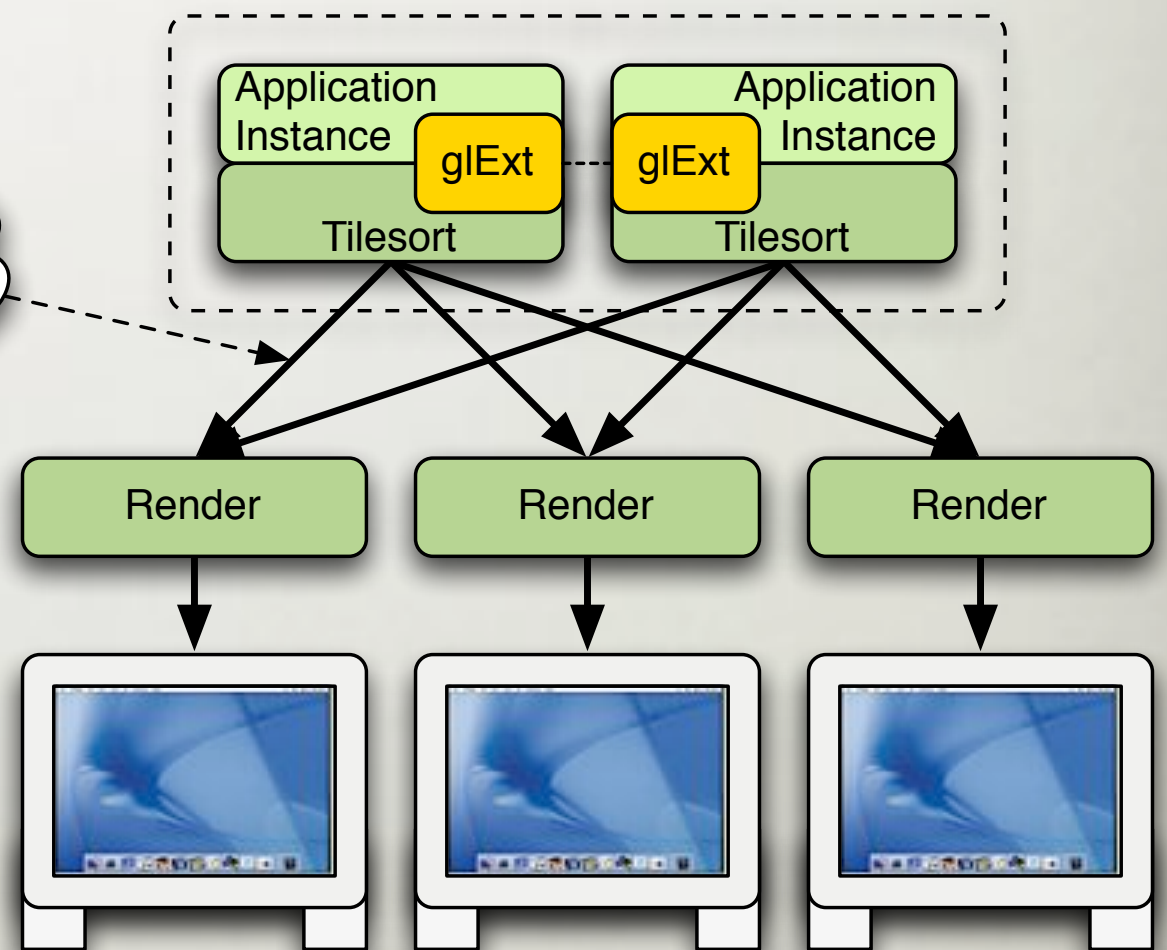
- Chromium, ModViz VGP, OMP, ...
- Operate on OpenGL command stream (HPC analogy: auto-parallelizing compilers)
- Provide programming extensions to improve performance and scalability (semi-transparent)
- Performance and compatibility issues

HPV Transparent Solutions

Transparent



Semi-Transparent



HPV Programming Interfaces

- ScaleViz, Vega Prime, OpenSG
 - Impose invasive programming model and data structure (HPC analogy: CFD codes)
 - Best for developing from scratch
- **Equalizer**, Cavelib, VRJuggler, MPK
 - Limited to HPV-critical areas of the code (HPC analogy: MPI, PVM)
 - Best for porting existing applications

Compositing Libraries

- Paracomp, NVIDIA Multi-GPU SDK
 - Address the backend part of an HPV application
 - Equalizer makes use of these libraries

GPGPU Frameworks

- CUDA, RMDP, CTM
 - HPC tools to use GPUs for data processing
 - Do not address parallel rendering
 - Can be integrated with OpenGL and Equalizer

Equalizer

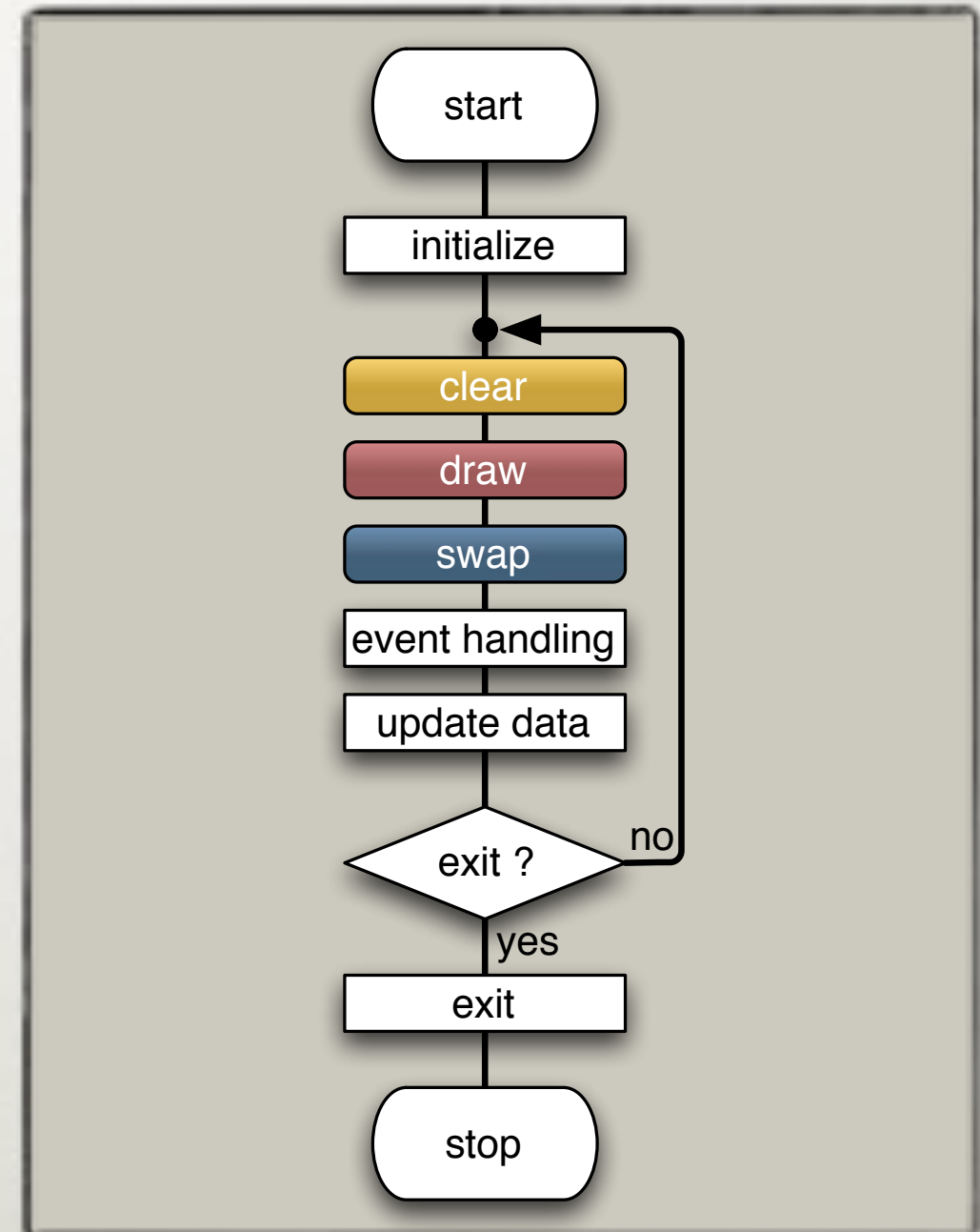
- Minimally invasive
- Runtime configuration
- Runtime scalability
- Asynchronous execution
- Clusters and SSI
- Open Source

Minimally Invasive

- “Make everything as simple as possible, but not simpler.” -- Albert Einstein
- Porting is as easy as possible
- Work is limited to visualization-relevant parts
- Read [Programming Guide](#) or [Parallel Graphics Programming](#) presentation

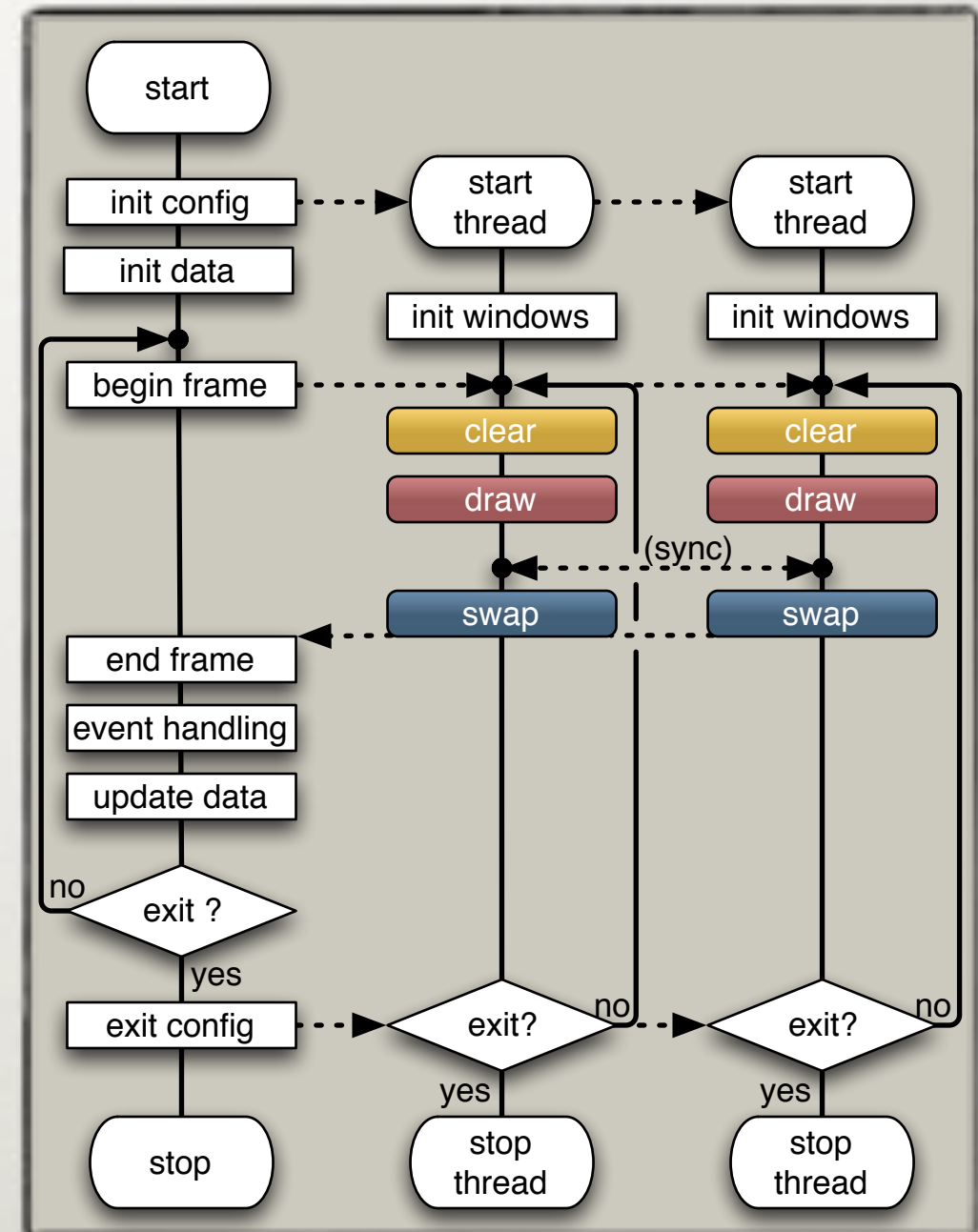
Equalizer Application

- Typical OpenGL application structure
- Separate rendering and application code



Equalizer Application

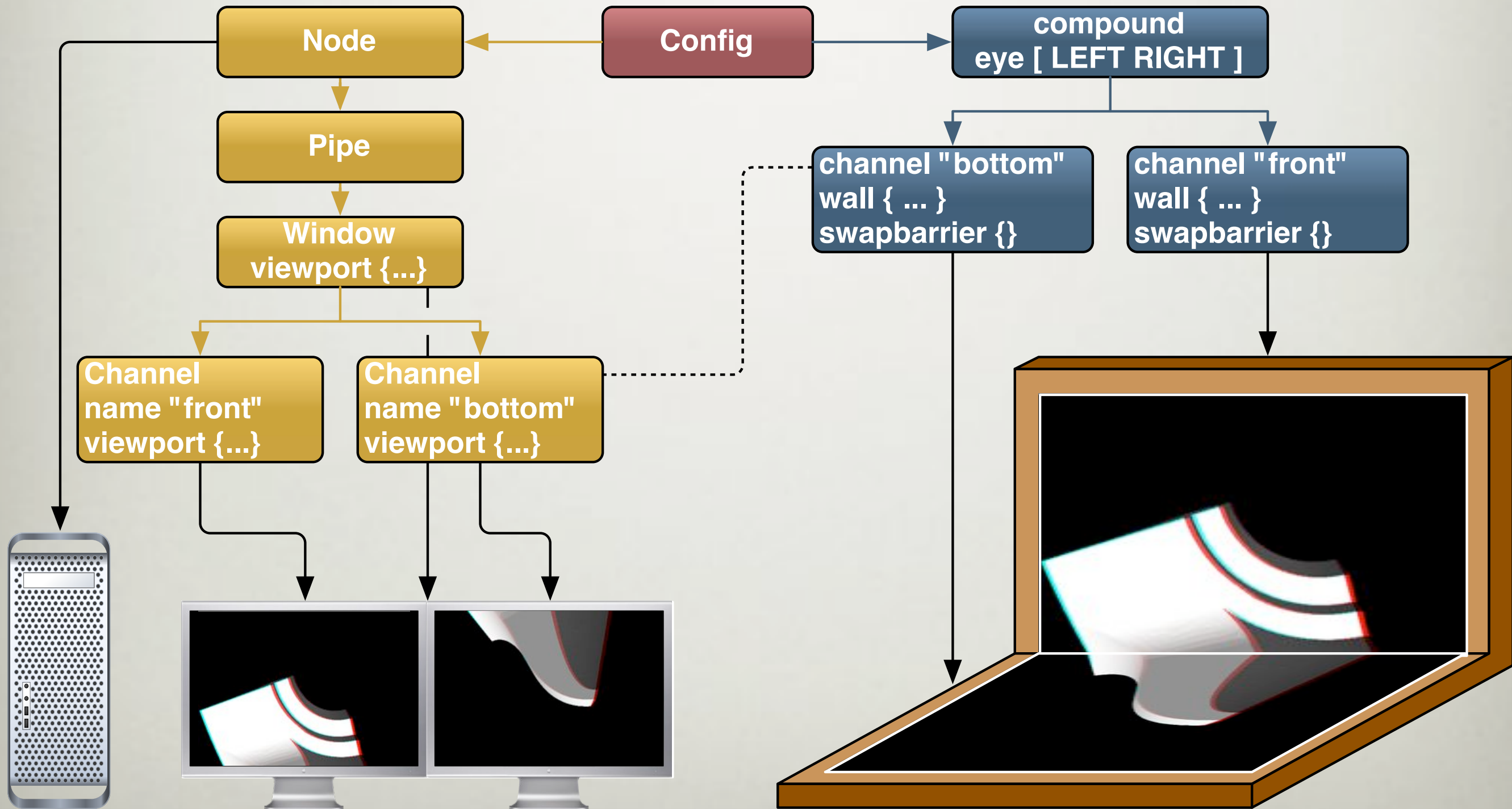
- Instantiate rendering multiple times
- Optional: data distribution for clusters



Runtime Configuration

- Hierarchical resource description:
Node→Pipe→Window→Channel
- Node: single system of the cluster
- Pipe: graphic card
- Window: drawable and context
- Channel: view
- Resource usage: compound tree

Runtime Configuration



Runtime Scalability

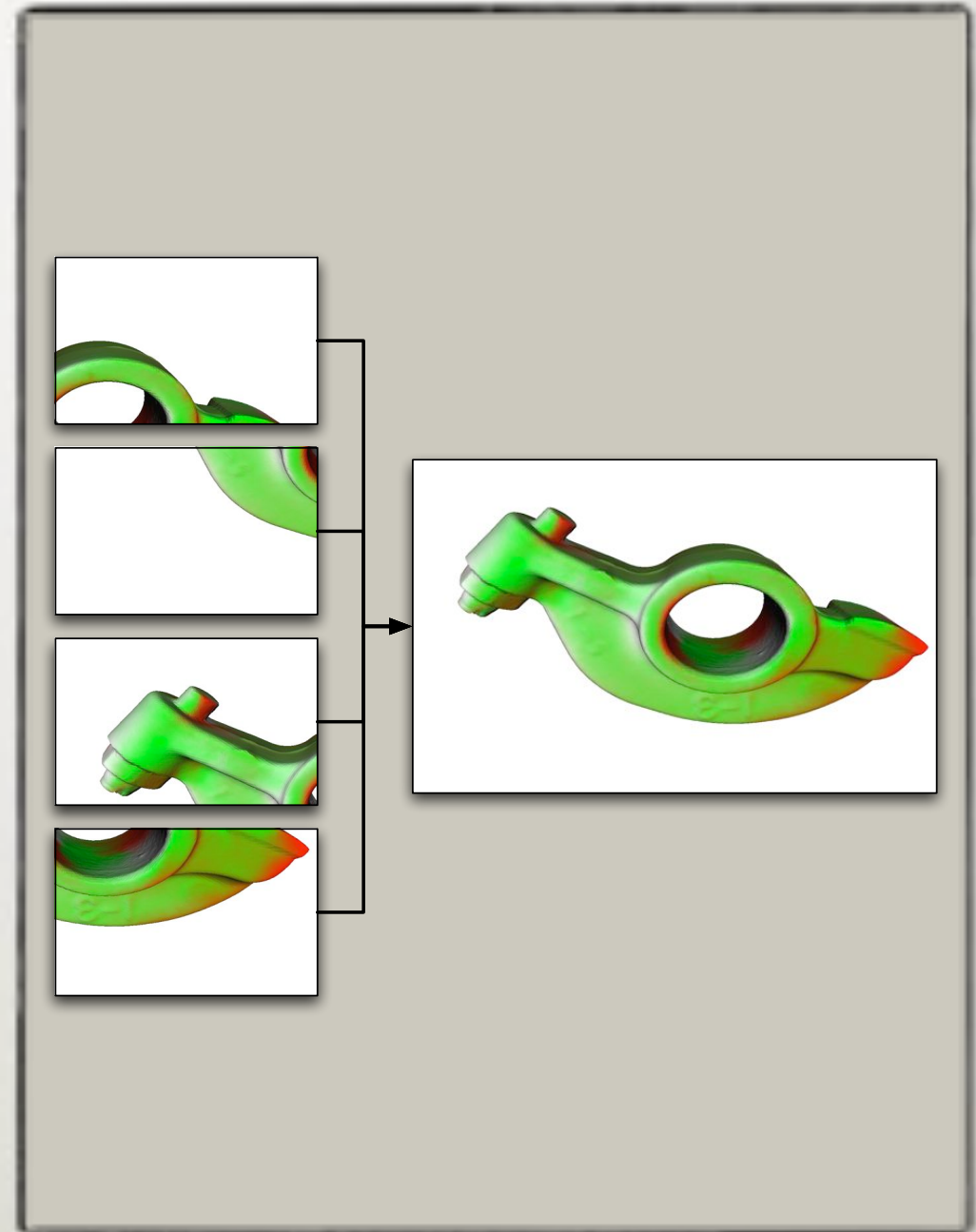
- Parallel execution of the application's rendering code
- One thread per graphics card, one process per node
- Decomposition of rendering for one view

Runtime Scalability

- 2D, DB, Stereo, DPlex, Pixel compounds
- Flexible configuration of decomposition and recomposition
- Compatible with compositing hardware
- Hardware-specific optimizations

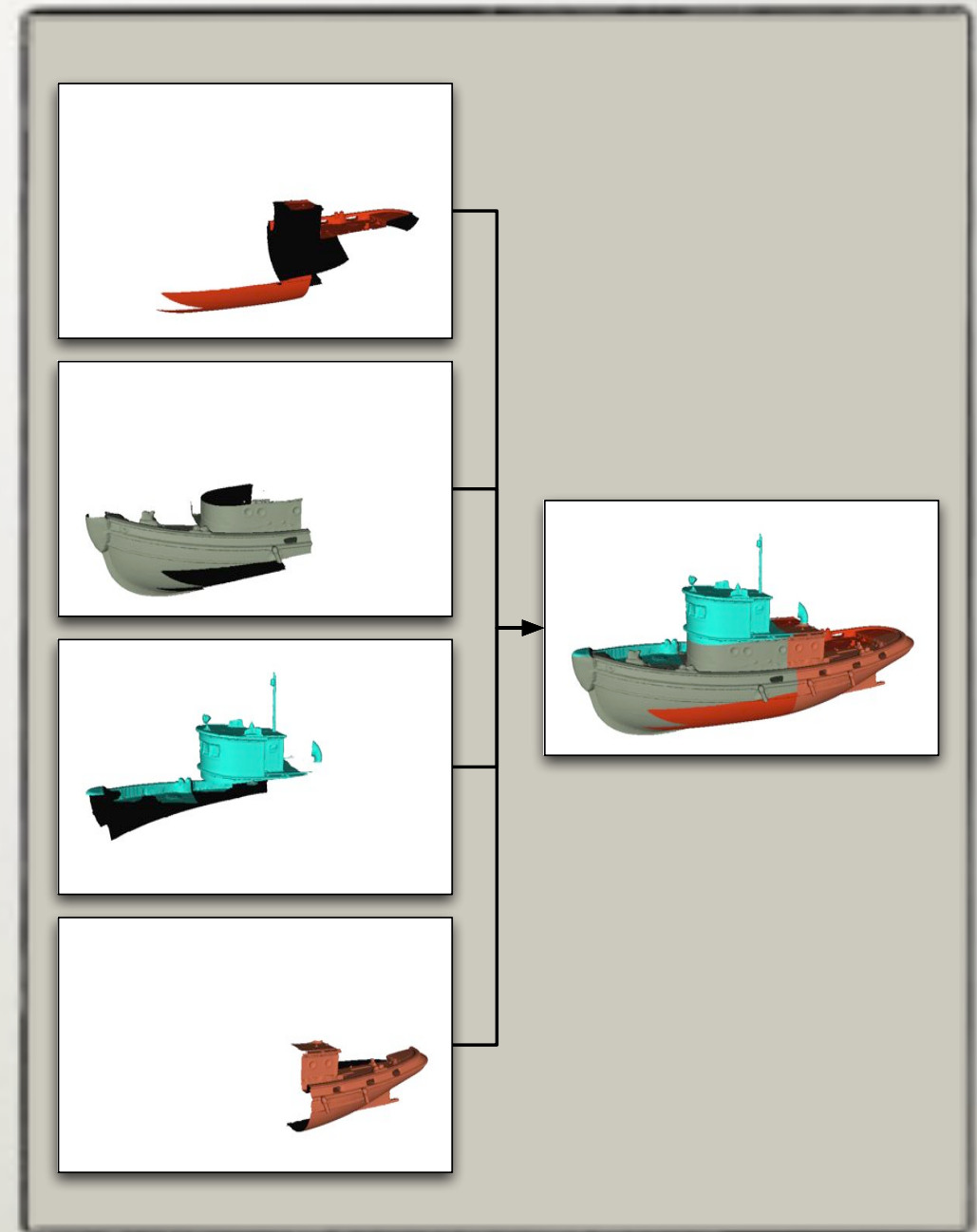
2D / Sort-First

- Scales fillrate
- Scales vertex processing if view frustum culling is efficient
- Parallel overhead due to primitive overlap limits scalability



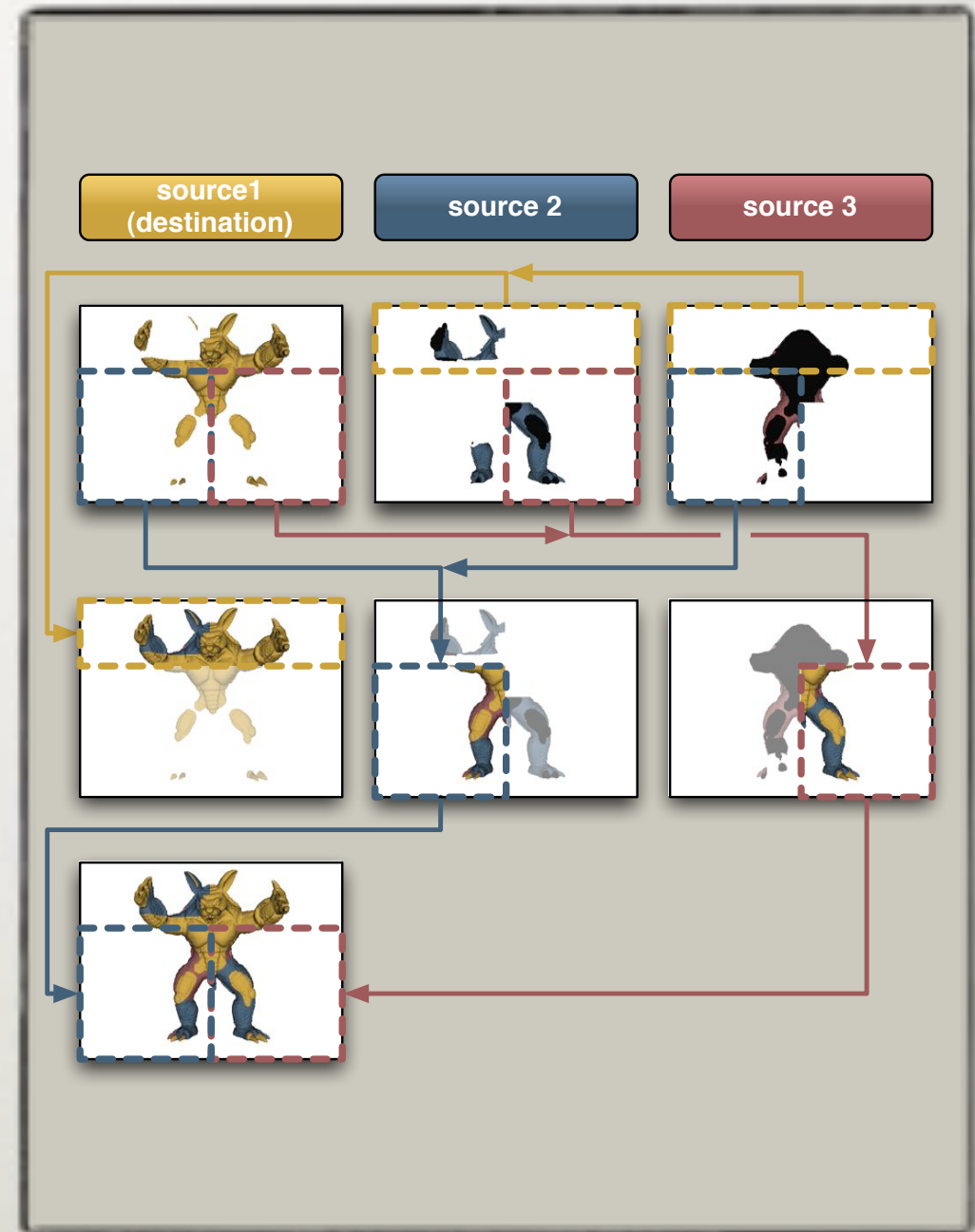
DB / Sort-Last

- Scales all aspects of rendering pipeline
- Application needs to be adapted to render subrange of data
- Recomposition relatively expensive



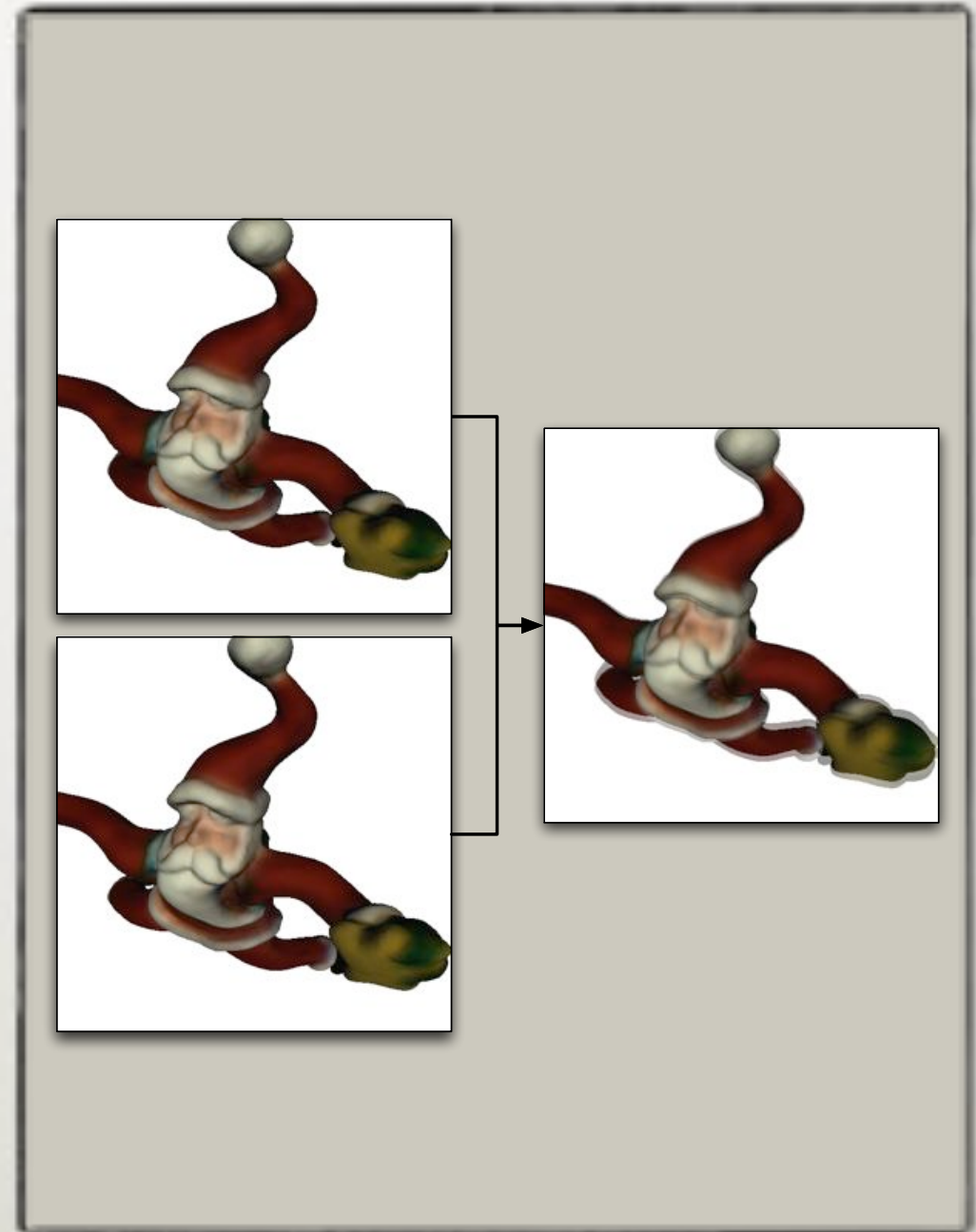
Parallel Compositing

- Compositing cost grows linearly for DB
- Parallelize compositing
- Flexible configuration
- Constant per-node cost
- Details in EGPGV'07 presentation



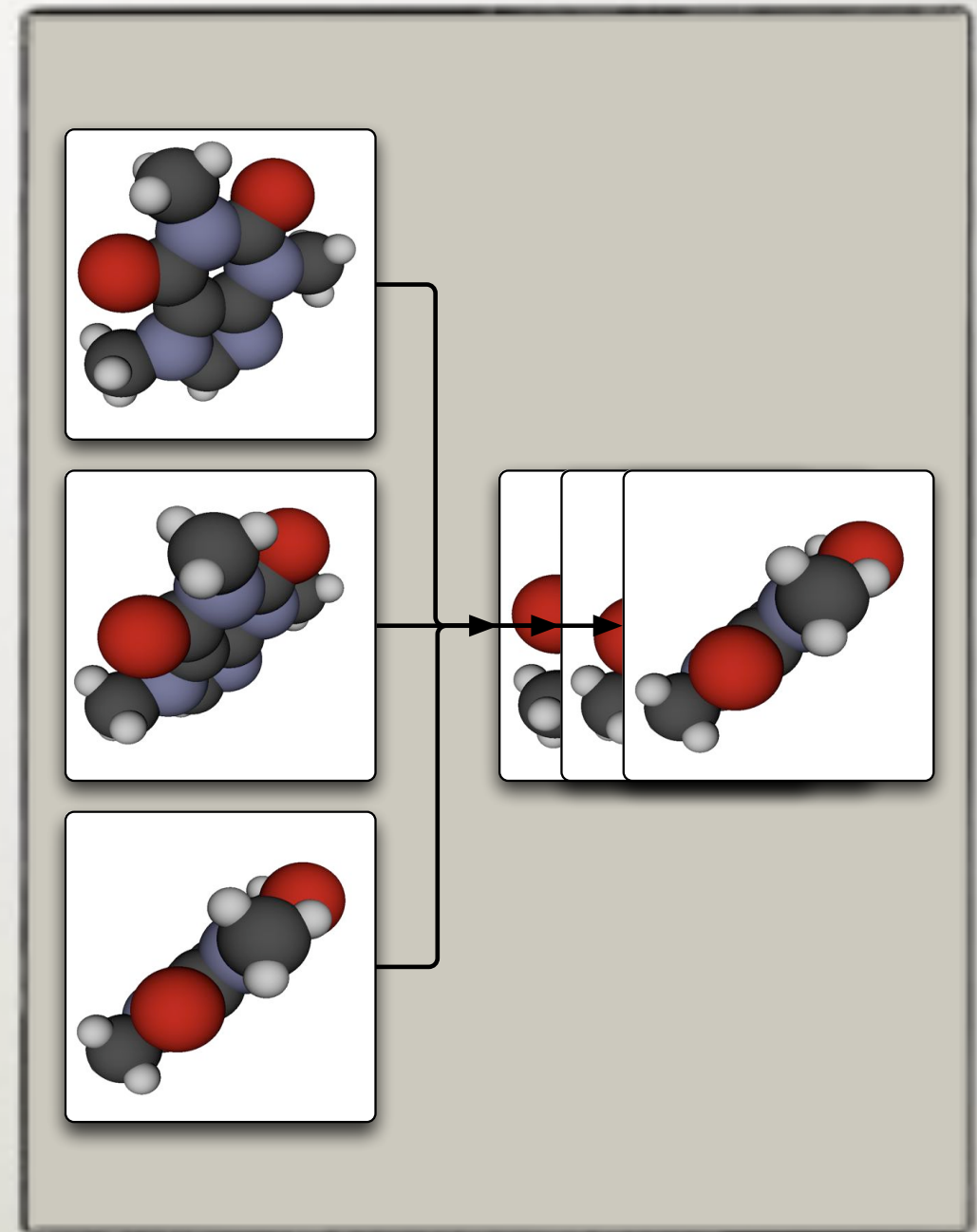
Eye / Stereo

- Stereo rendering
- Active, passive and anaglyphic stereo
- quasi-linear scalability and loadbalancing
- Limited by number of eye views



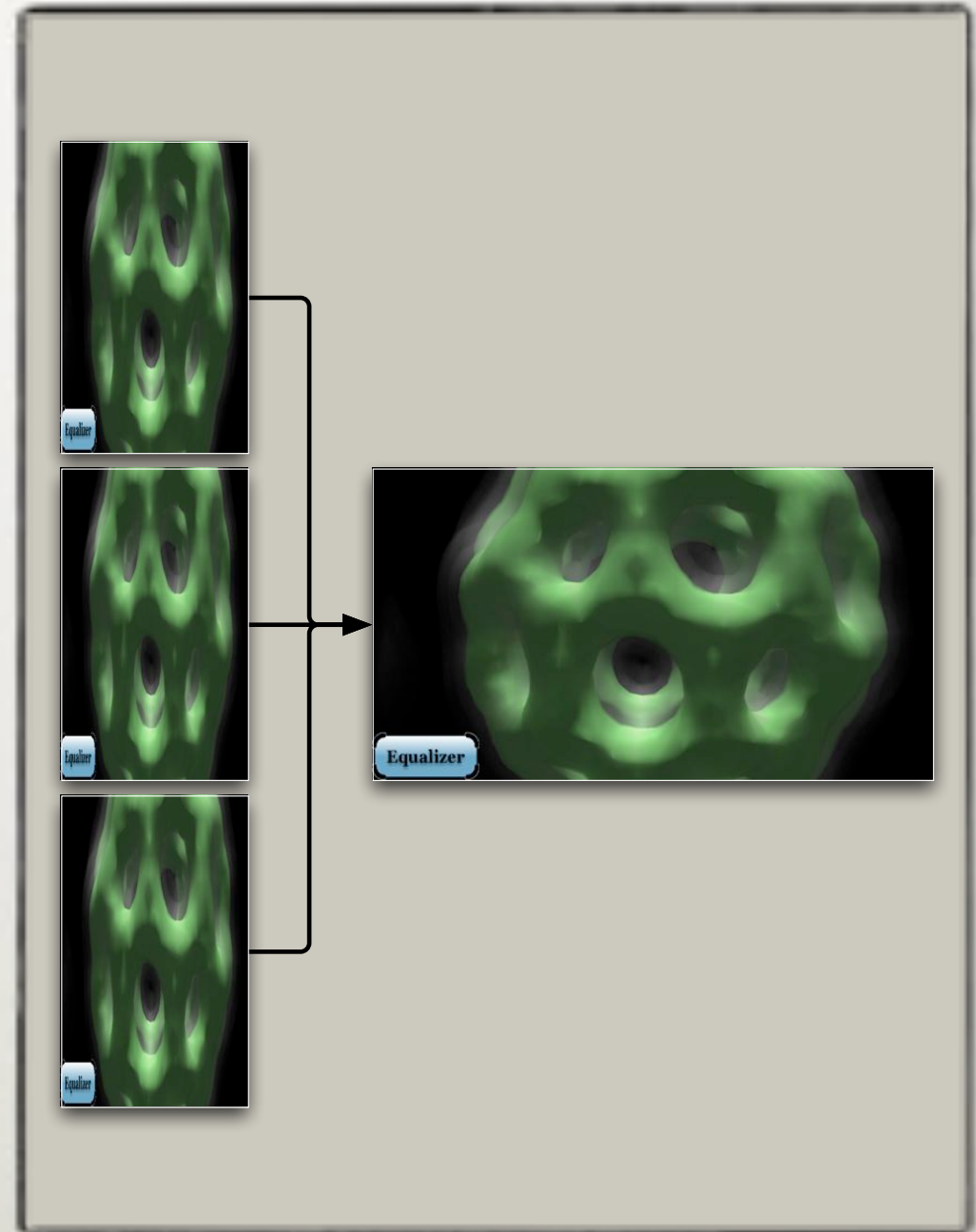
DPlex / Time-Multiplex

- quasi-linear scalability and loadbalancing
- Increased latency may be an issue
- Increased framerate often compensates for latency



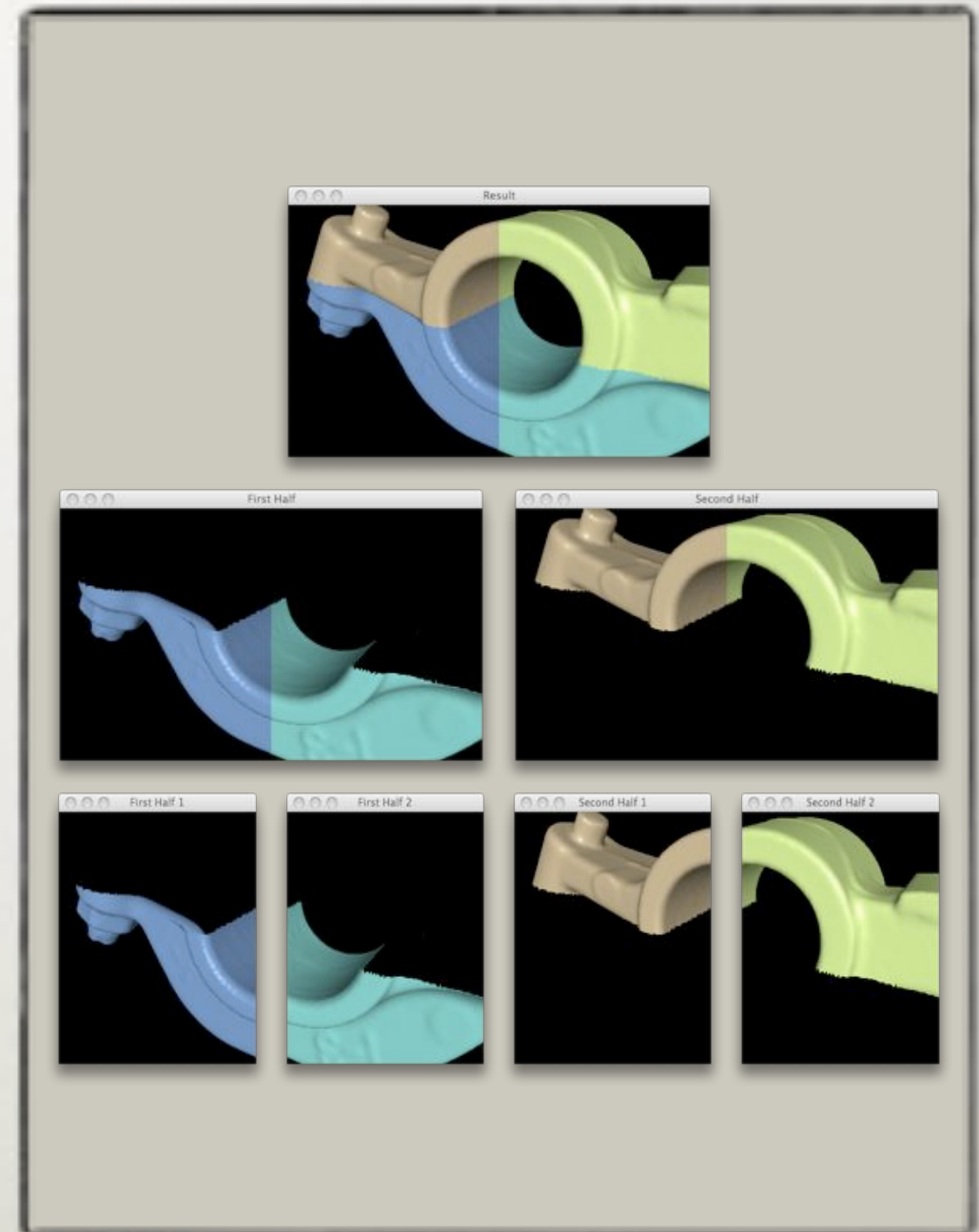
Pixel

- Scales fillrate perfectly
- Similar to 2D
- Raytracing, Volume Rendering



Multilevel Compounds

- Compounds allow any combination of modes
- Combine different algorithm to address and balance bottlenecks
- Example: use DB to fit data on GPU, then use 2D to scale further

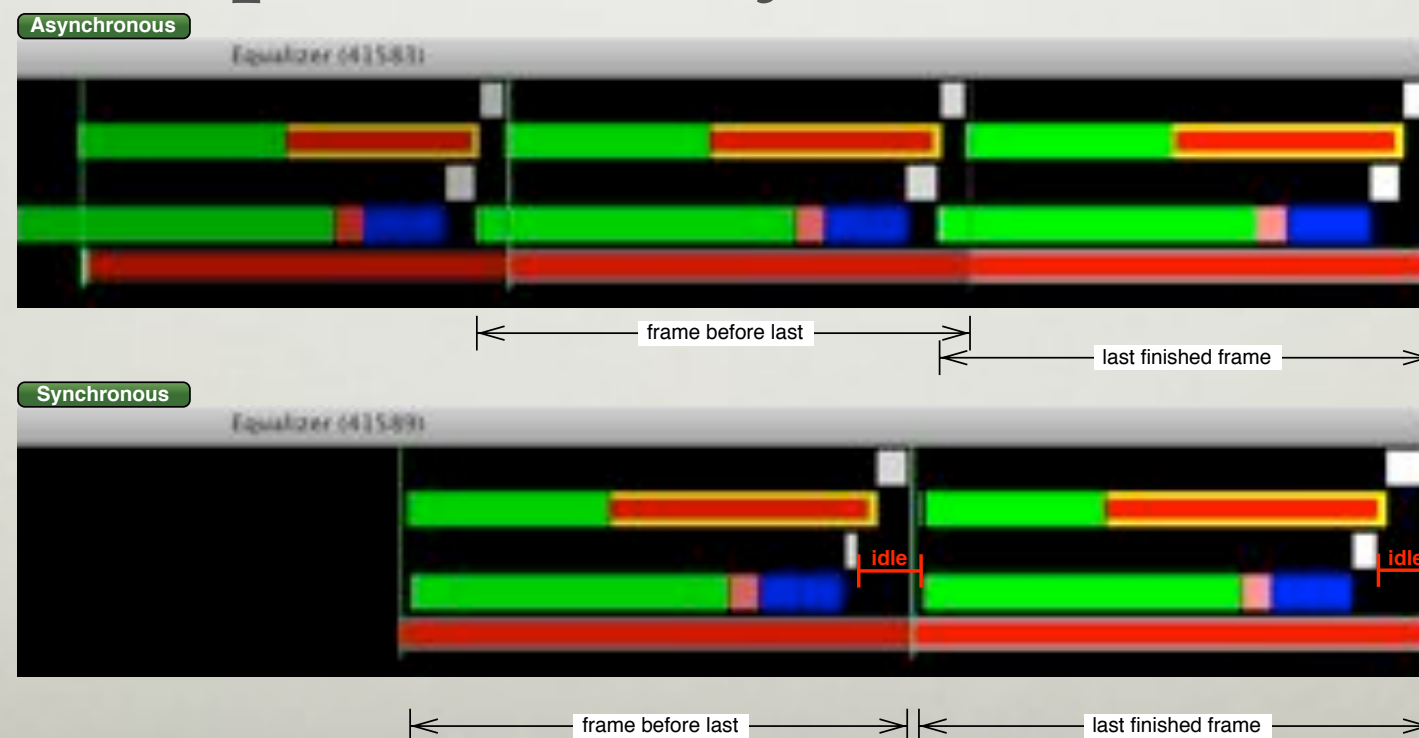


Compounds

- 2D: low IO overhead, limited scalability
 - DB: high IO overhead, great scalability
 - Eye, DPlex: quasi-linear scalability
 - Pixel: linear fill-rate scalability
- ➡ Combine modes
- ➡ DB: use parallel compositing

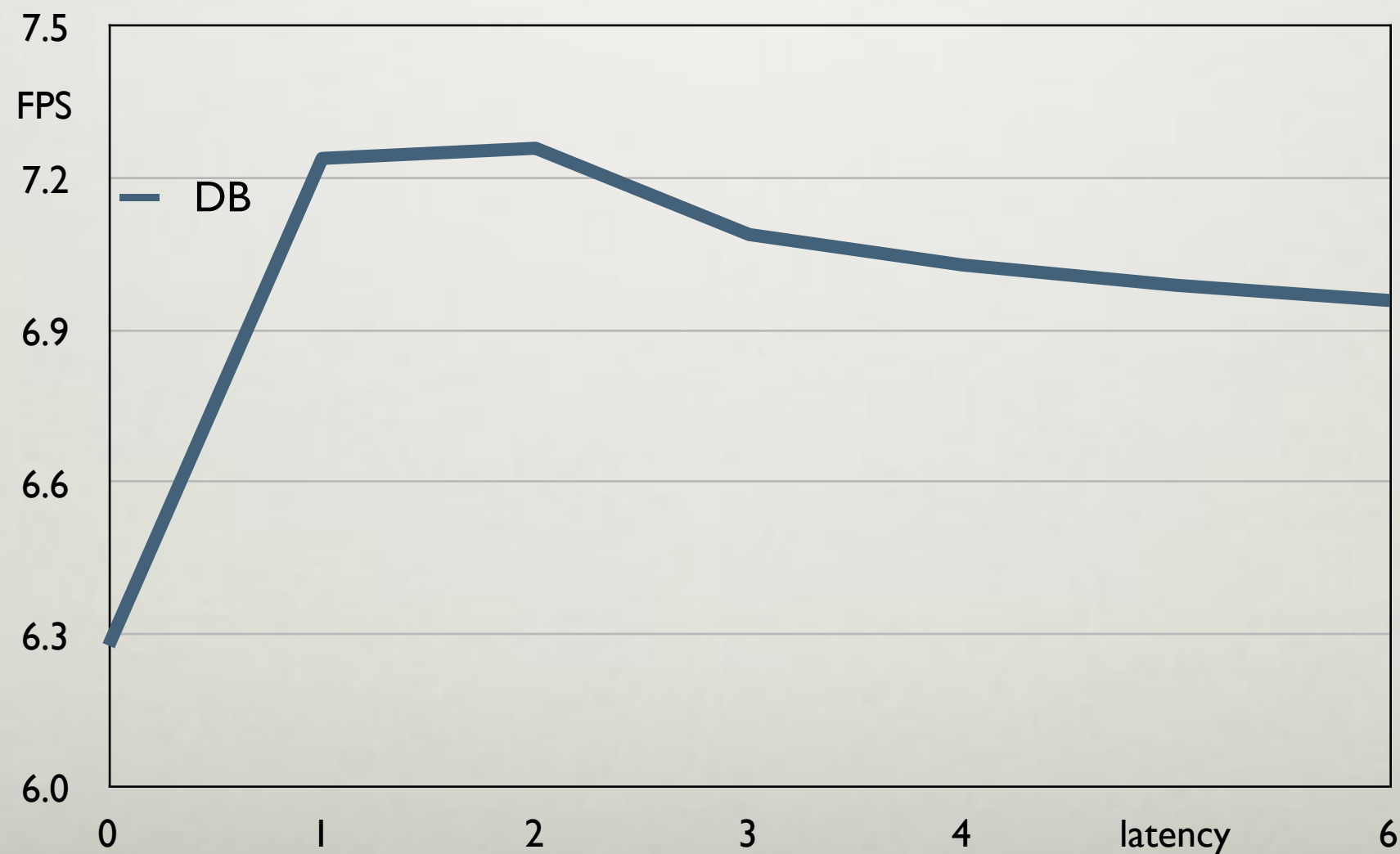
Asynchronous Execution

- Improves scalability on bigger clusters
- Latency between last draw and main
- Hides imbalance in load distribution
- Optional per-node synchronization



Asynchronous Execution

- Example: 5-node sort-last, direct-send
- 15% speedup

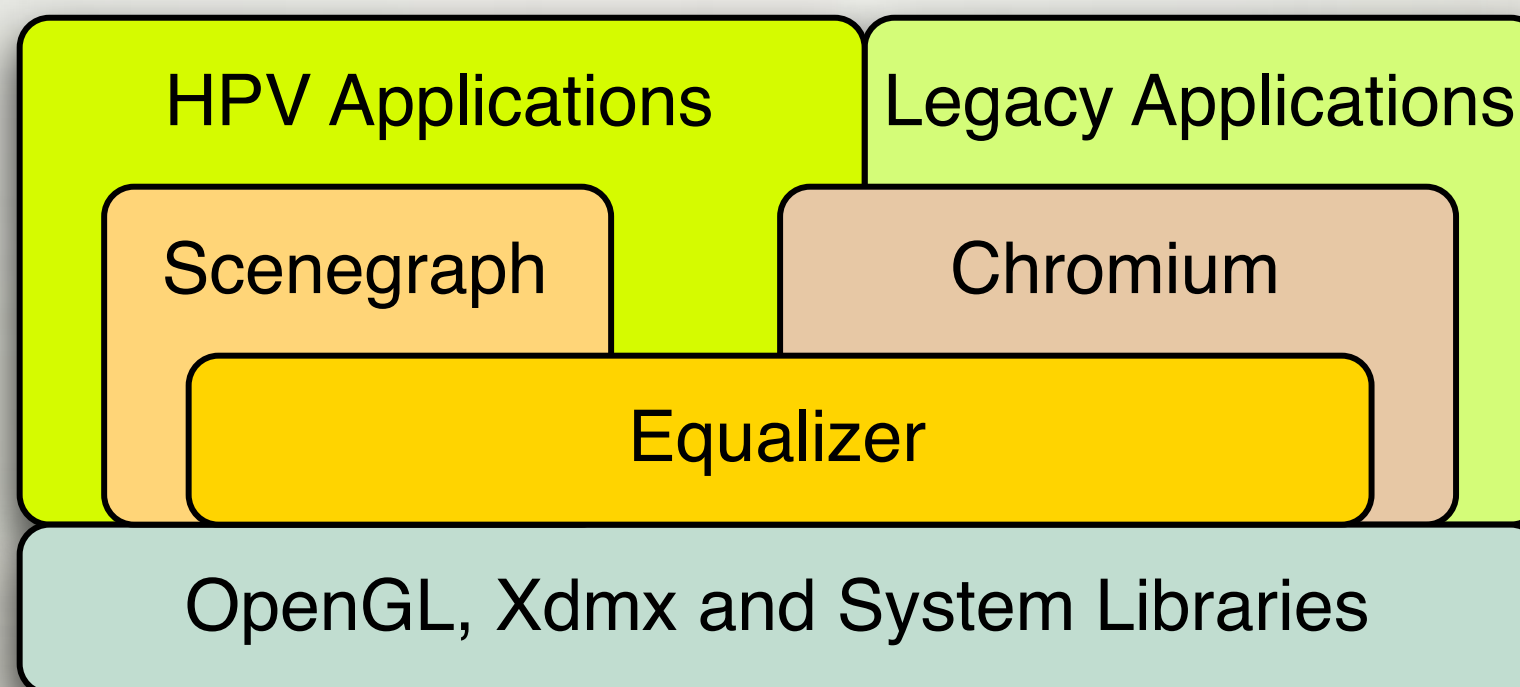


Multi-GPU and Clusters

- Equalizer runs on both architectures
- Execution model is the same
- Shared memory systems allow additional optimisations
- Porting for SSI simpler than full port

Equalizer Vision

- Equalizer: Scalable rendering engine
- Chromium: OpenGL single virtual screen
- Scenegraphs: Distributed data management



Near Future

- DB compositing optimizations
- Examples, demos, applications
- Server extensions
- Failure robustness

Open Source

- LGPL license: commercial use welcome
- Open standard for scalable graphics
- Minimally invasive: easy porting
- Clusters and shared memory systems
- Linux, Windows, Mac OS X
- More on: www.equalizergraphics.com