Equalizer

Parallel OpenGL Application Framework

Stefan Eilemann, Eyescale Software GmbH

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

History

- 1992-?: CAVElib
  - Multi-Frustum
- 2000-2003: SGI OpenGL Multipipe SDK
  - Scalability for SGI Onyx and Prism
- 2005-today: Equalizer
  - Clusters, C++, Open Platform
- 2007-today: Eyescale Software GmbH

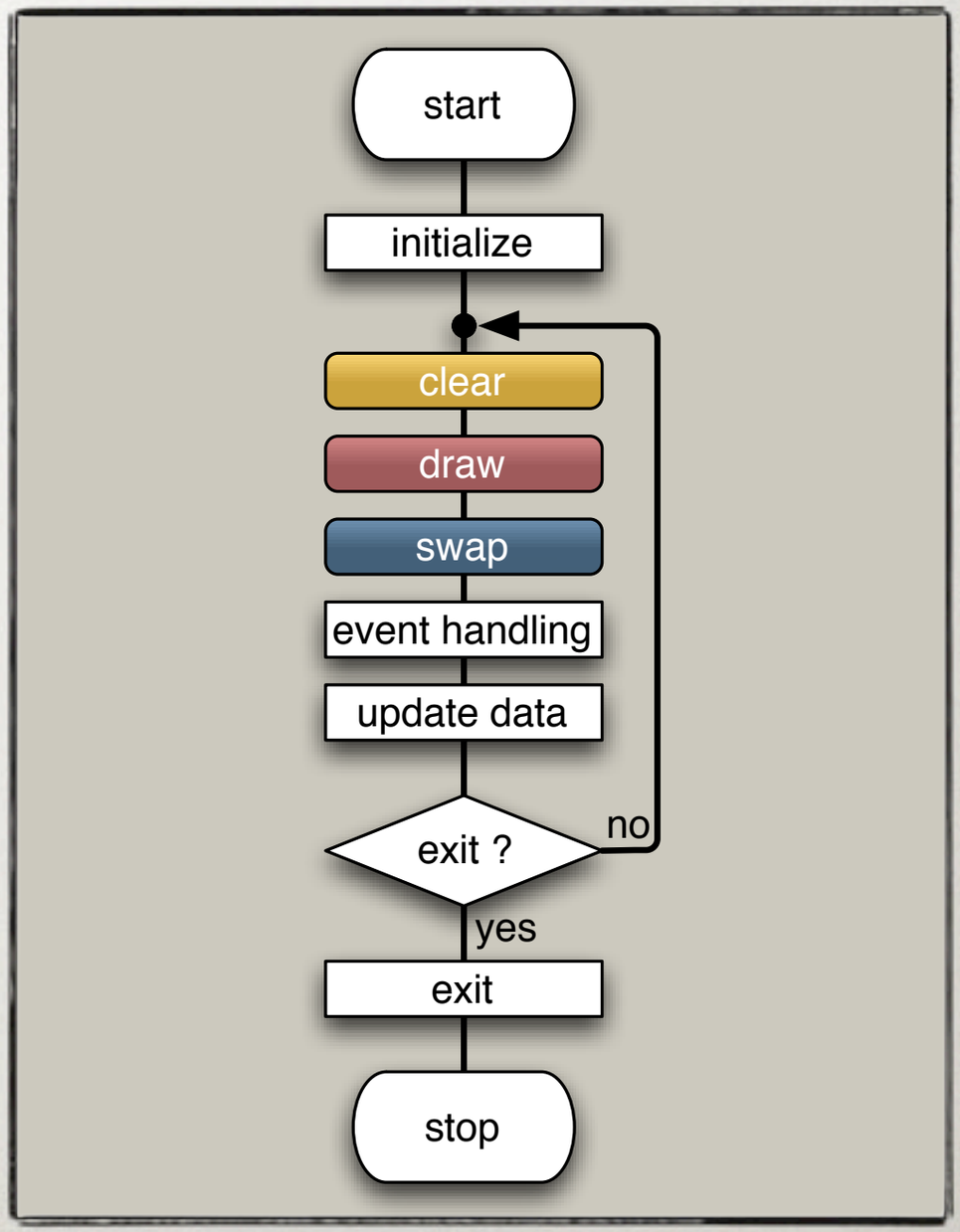
Equalizer

"GLUT for multi-GPU systems and visualization clusters"

- Task-driven: init, exit, clear, draw, (readback, assemble)
- Resource-based: Node, Pipe, Window, Channel

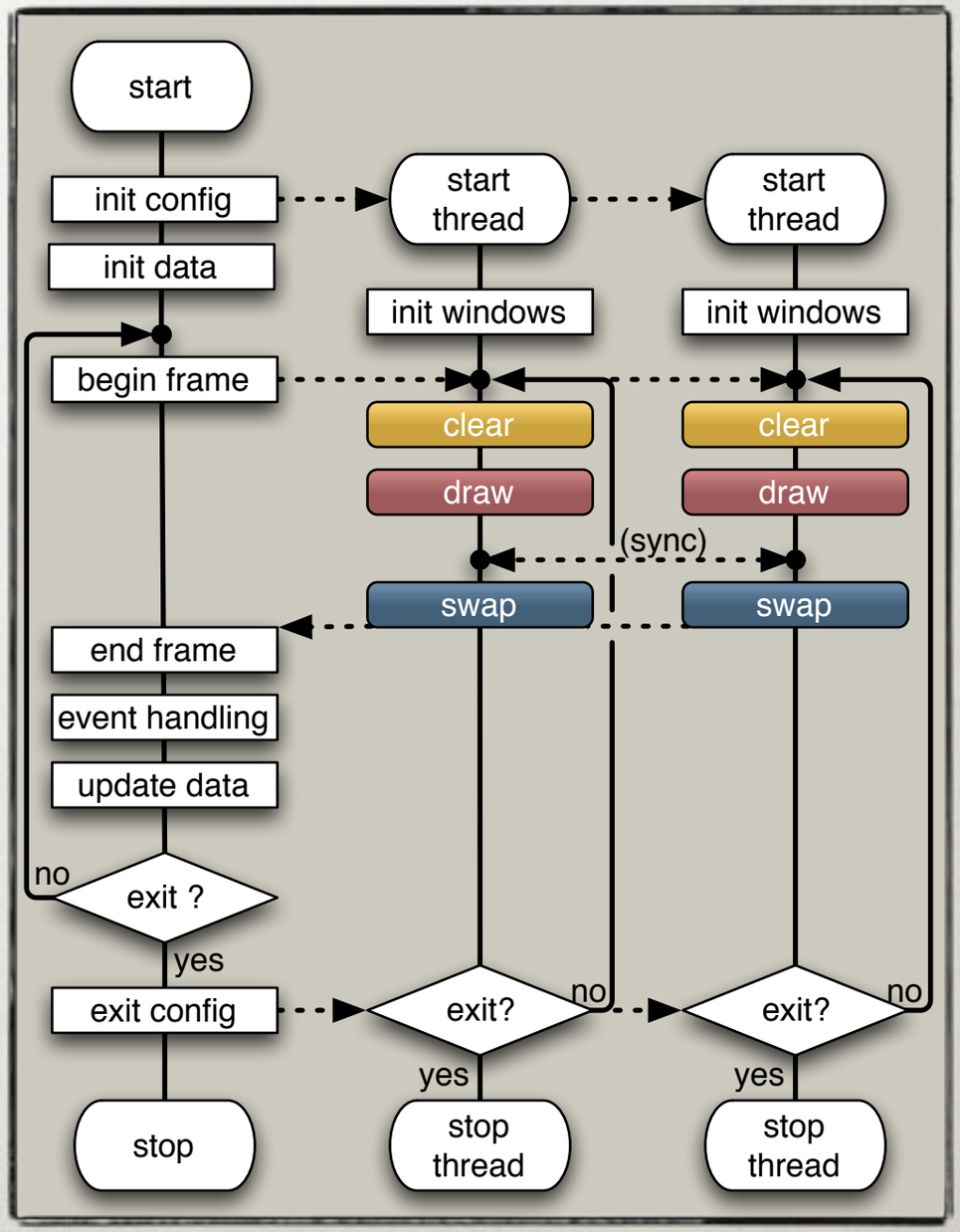
Single Pipe Application

- One thread
- Typical rendering loop
- Stages may not be well separated



Equalizer Application

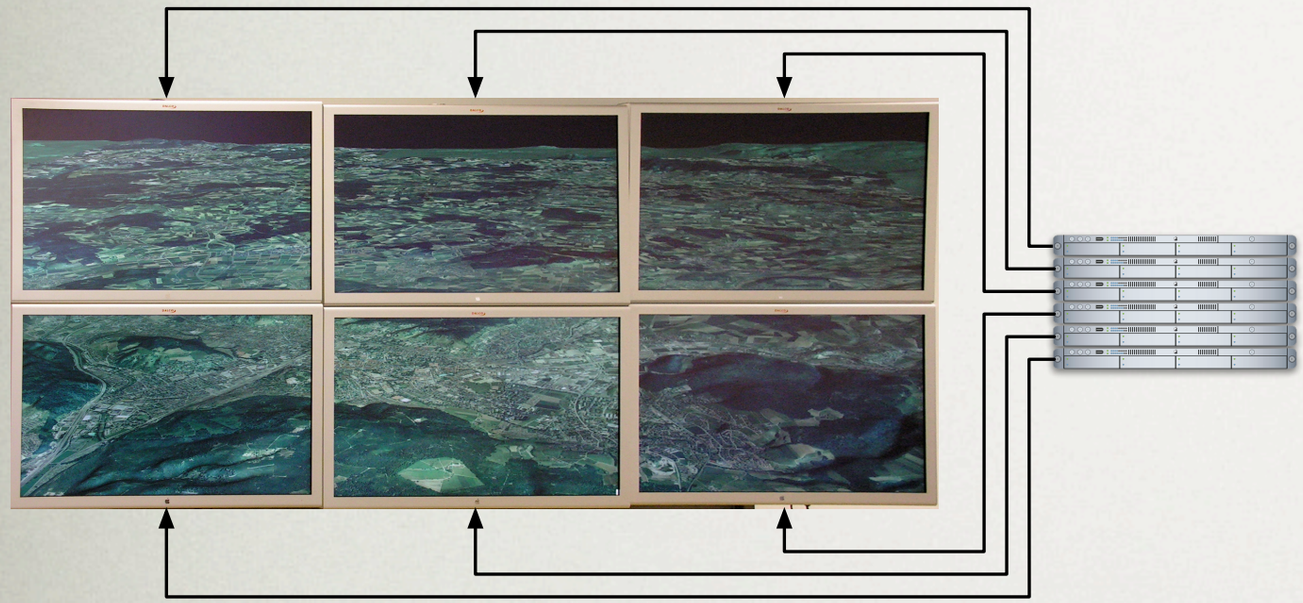
- Separate rendering and application code
- Instantiate rendering multiple times
- Synchronize parallel execution
- Optional: data distribution for clusters



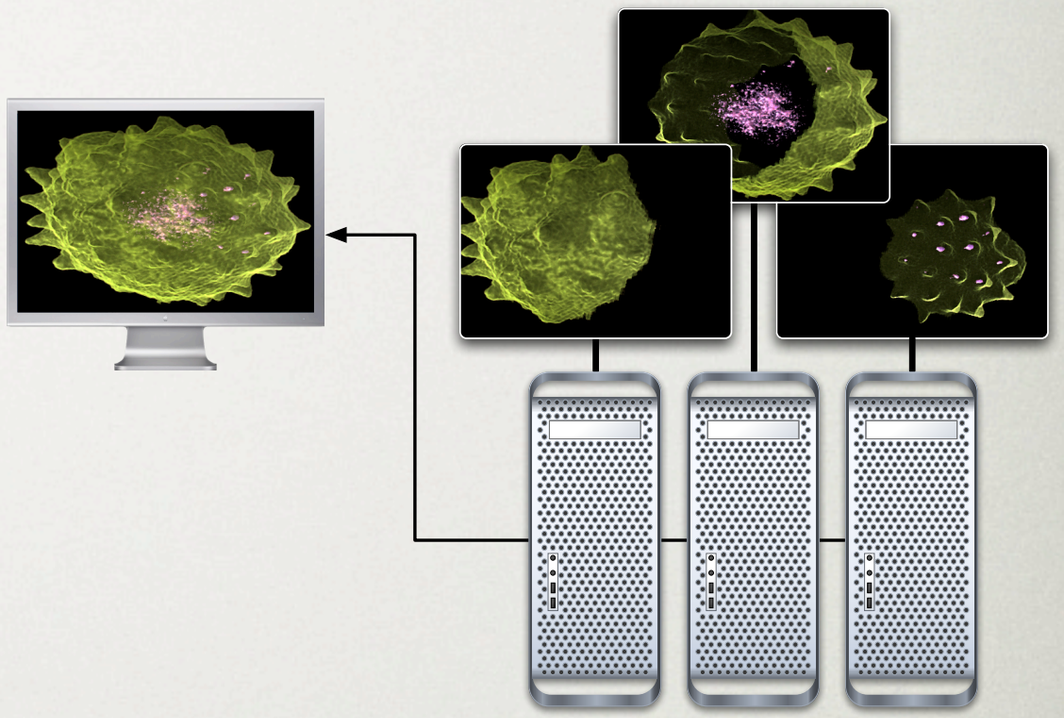
Selected Use Cases

Eyescale

Display Walls

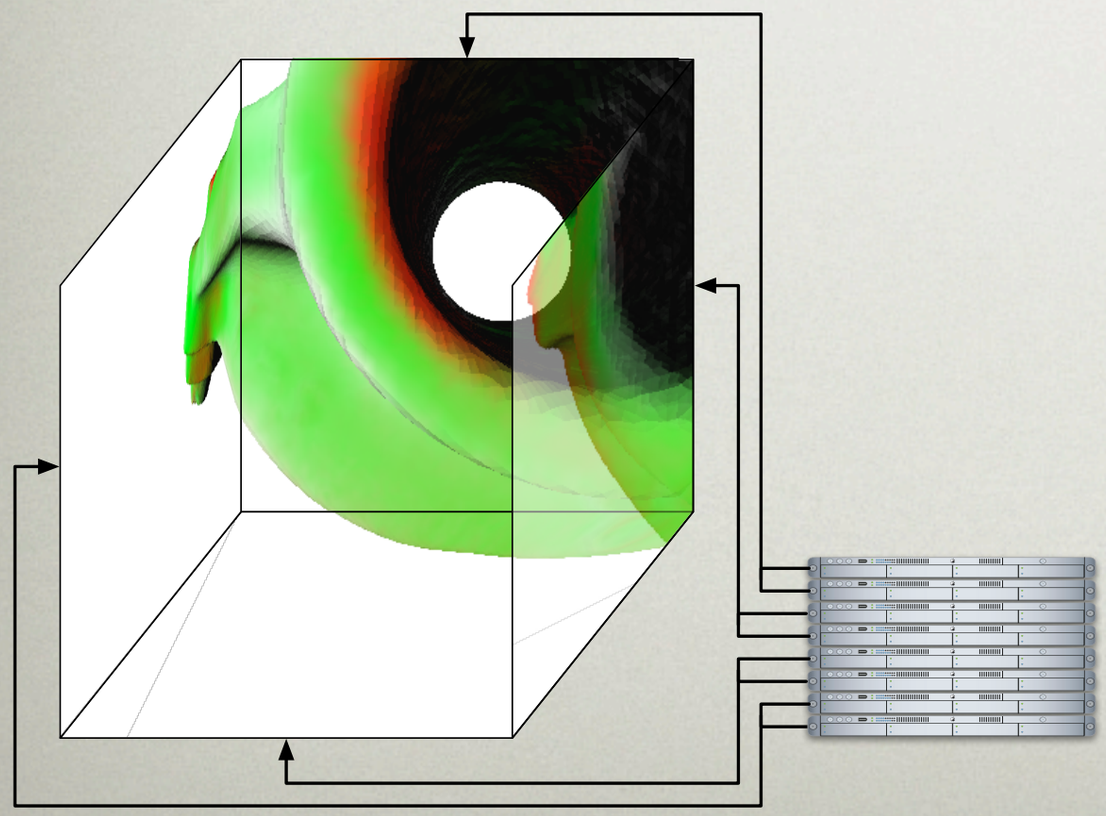


Scalable Rendering

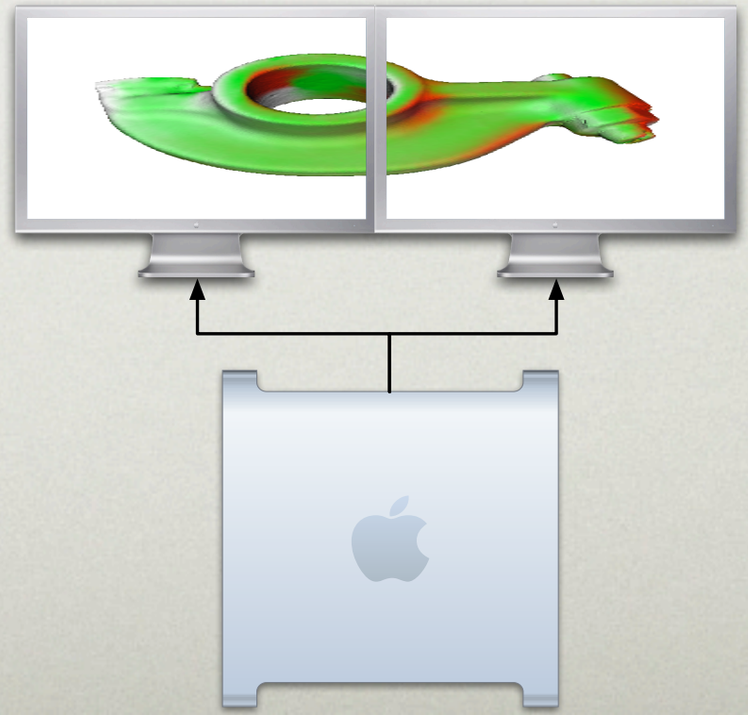


Equalizer

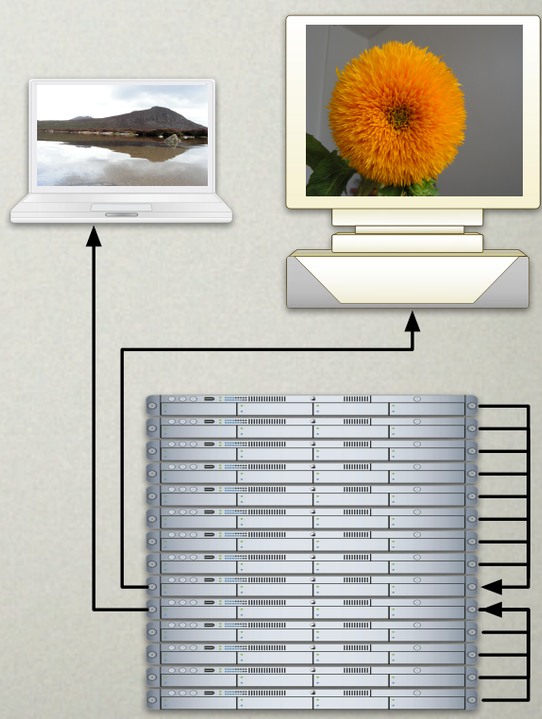
Virtual Reality



Multipipe Workstations



Remote Rendering



Equalizer Programming Interface

Applications are written against a *client library* which abstracts the interface to the execution environment

- Minimally invasive programming approach
- Abstracts multi-processing, synchronisation and data distribution
- Supports distributed rendering and performs result re-assembly

Resource Management System

Applications are deployed by a *server* which balances the resource usage across the system

- Centralizes the setup for all applications
- Configures application and deploys render clients
- Dynamic load-balancing of the cluster resources

Competitive Environment

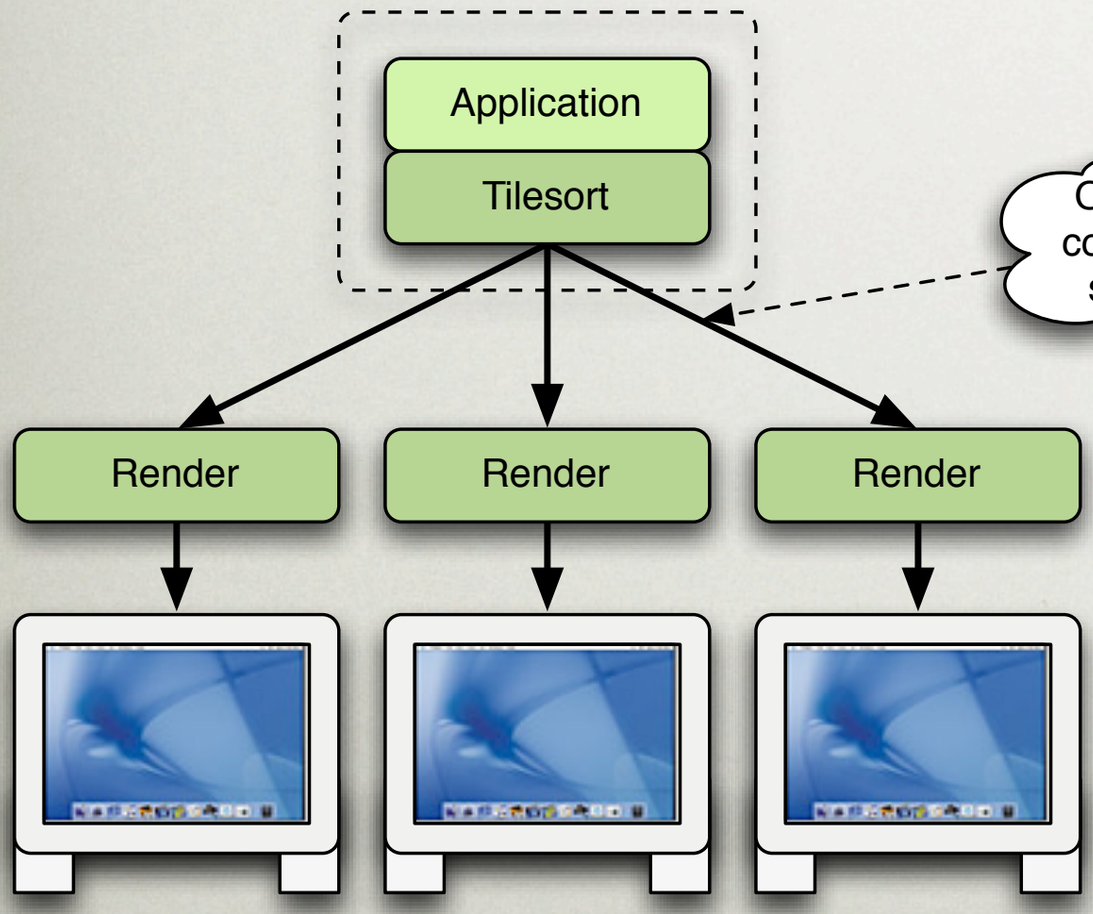
- Transparent solutions
  - Based on OpenGL interception
- Programming interfaces
  - Distributed Scene Graphs
  - Middleware
- GPGPU frameworks

HPV Transparent Solutions

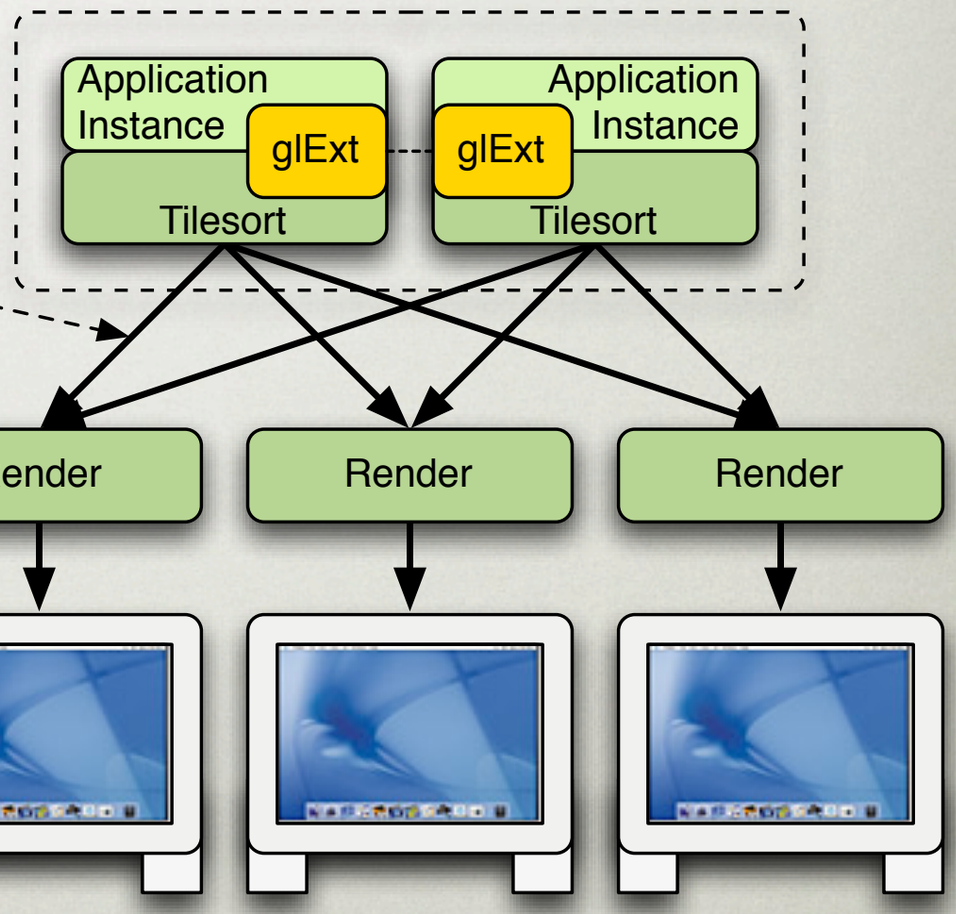
- Chromium, ModViz VGP, OMP, ...
- Operate on OpenGL command stream (HPC analogy: auto-parallelizing compilers)
- Provide programming extensions for better performance and scalability
- Performance and compatibility issues

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

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

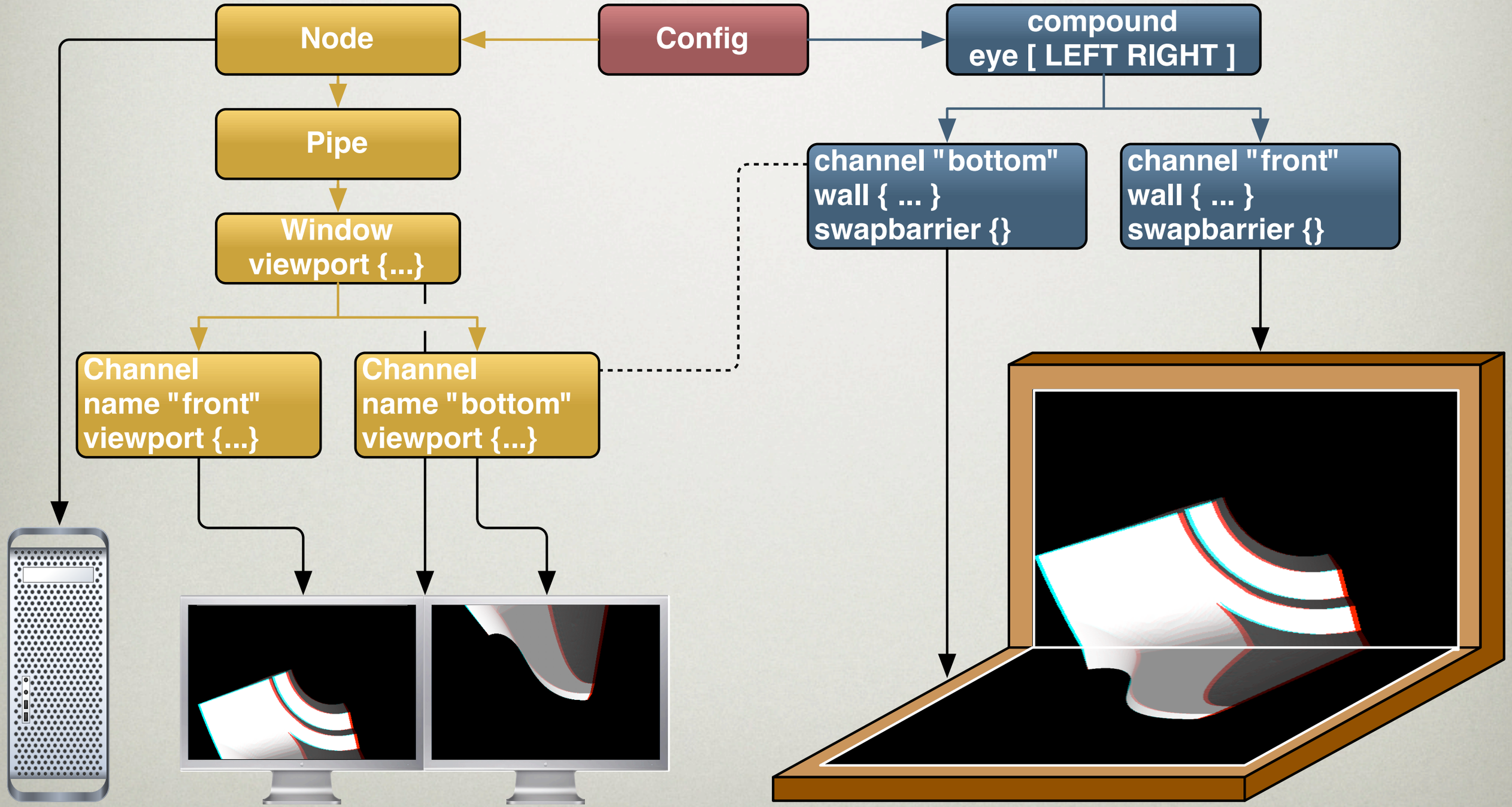
- “Make everything as simple as possible, but not simpler." -- Albert Einstein

- Porting is as easy as possible
- Work is limited to visualization-relevant parts
- Read Programming Guide or Parallel Graphics Programming presentation

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

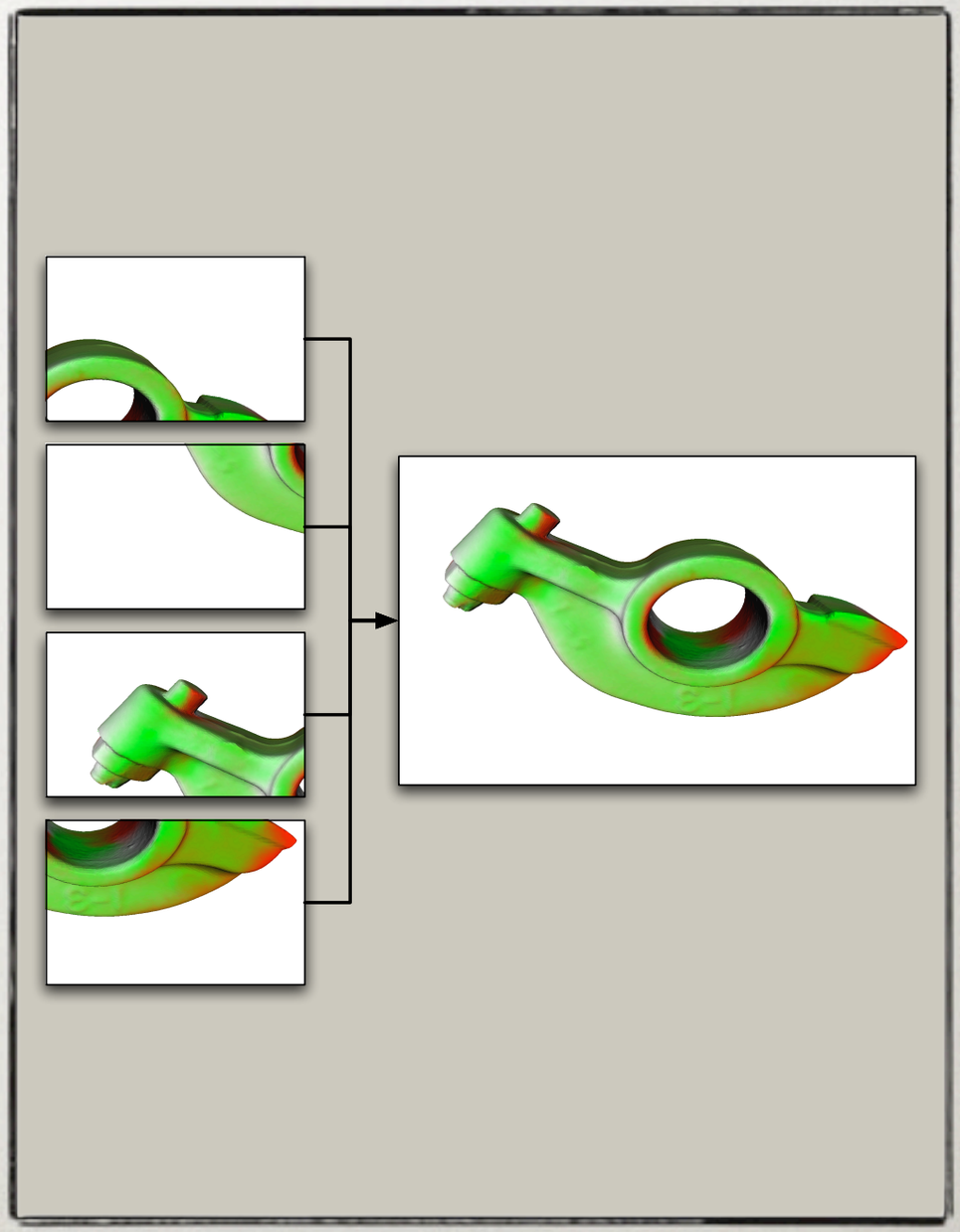
- Parallel execution of the application's rendering code
- One thread per graphics card, one process per node
- Decomposition of rendering for one view

Runtime Scalability

- Sort-first (2D), sort-last (DB), stereo (eye) and pixel compounds
- Flexible configuration of decomposition and recomposition
- Compatible with compositing hardware
- Hardware-specific optimizations

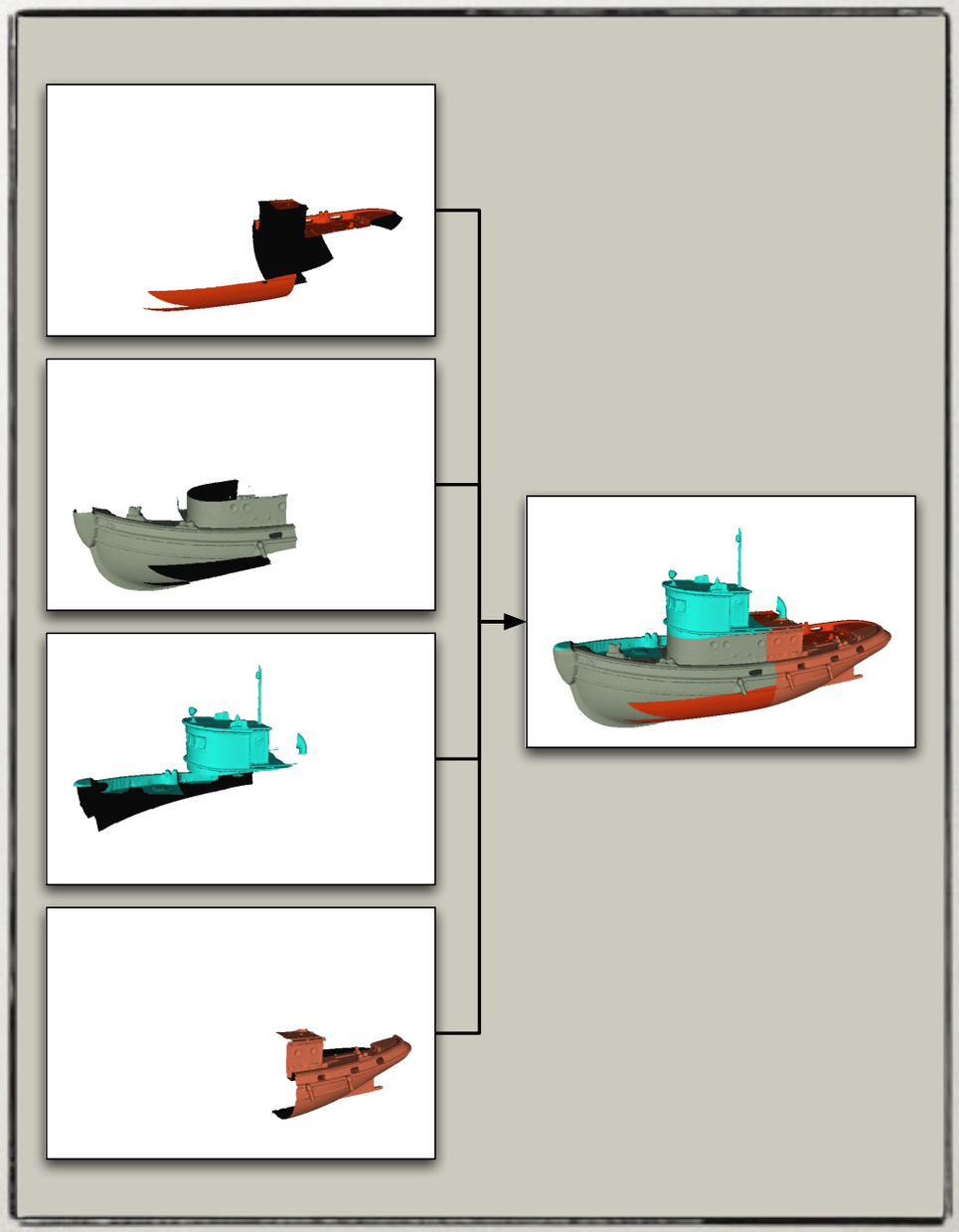
2D / Sort-First

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



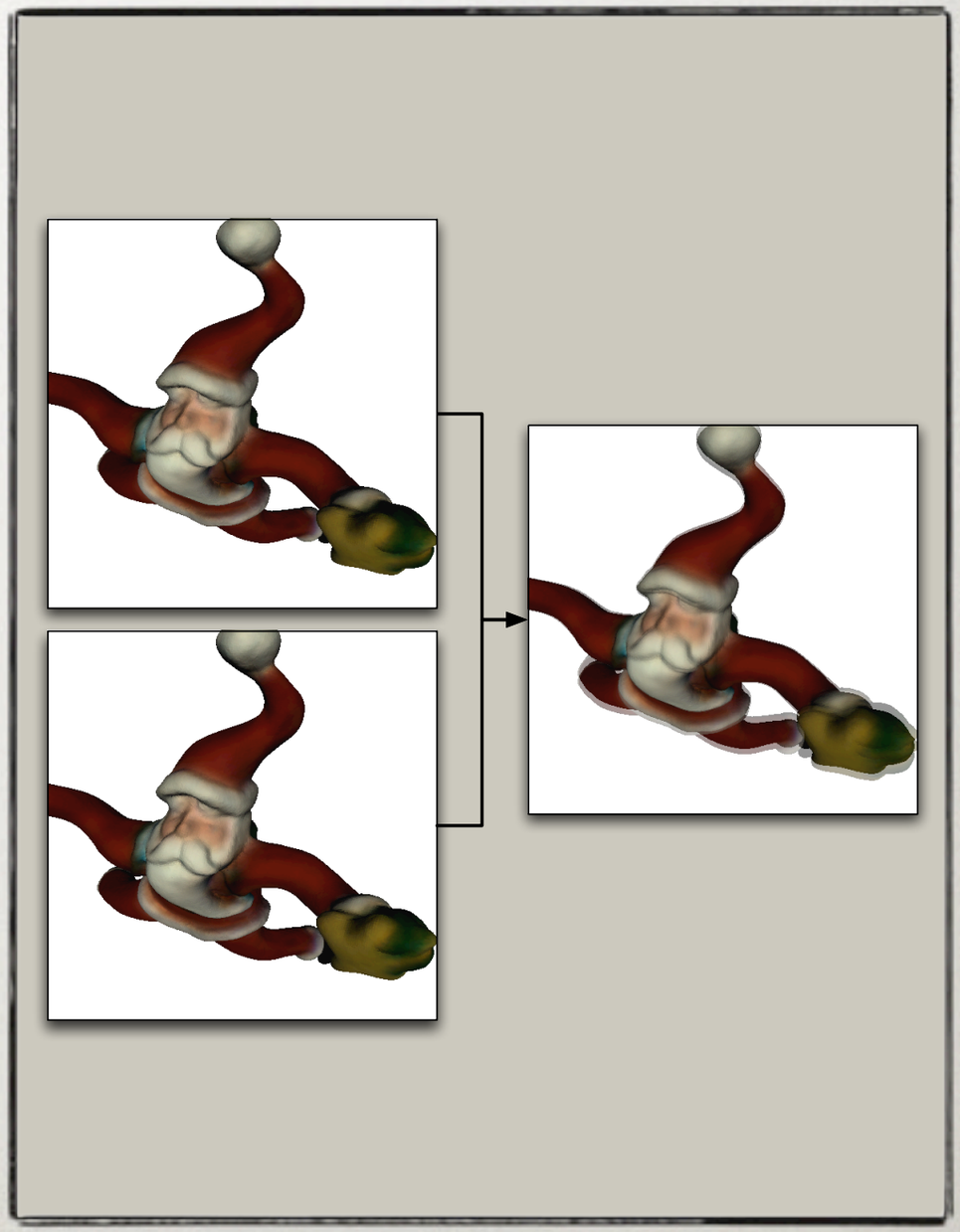
DB/Sort-Last

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



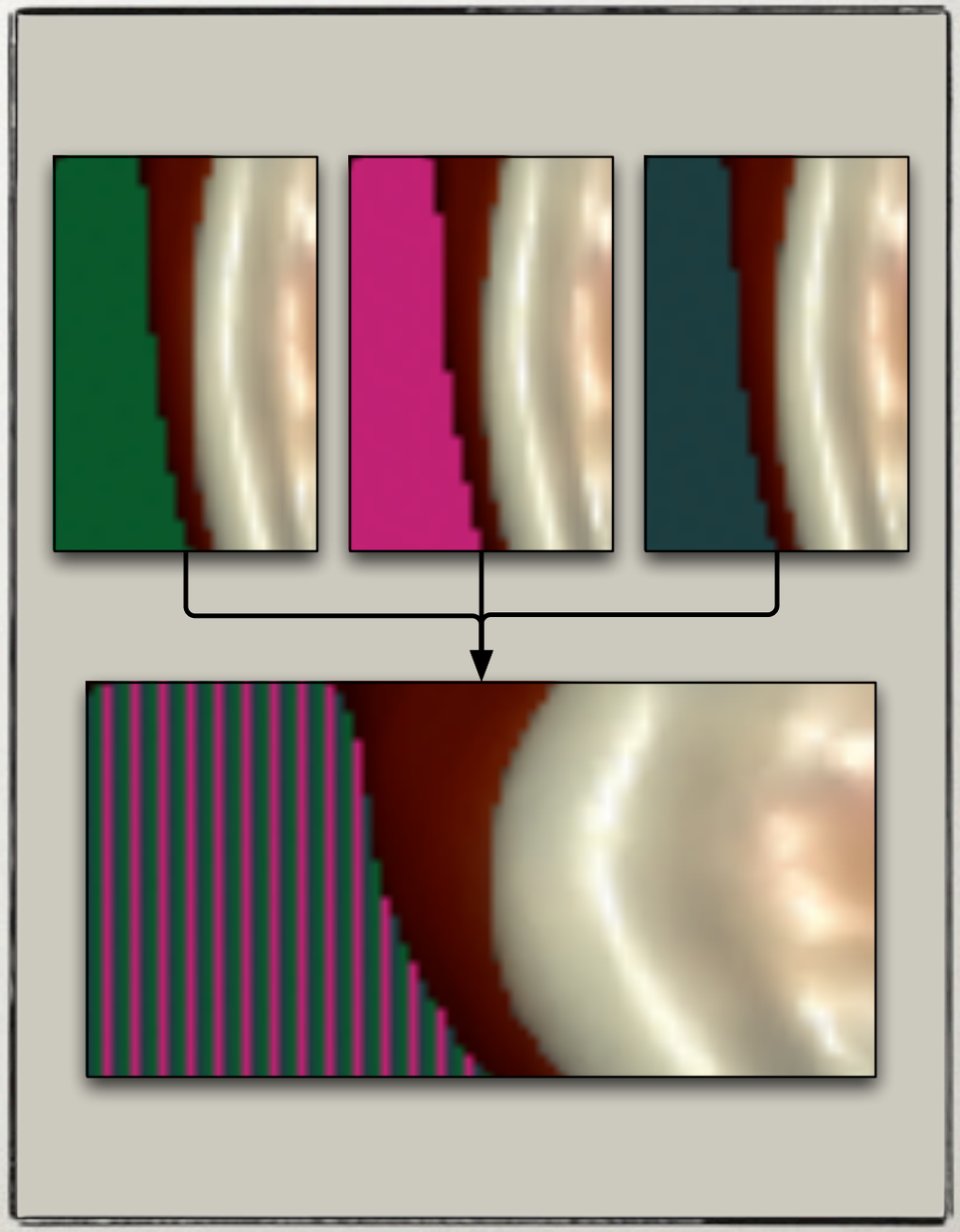
Eye/Stereo

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



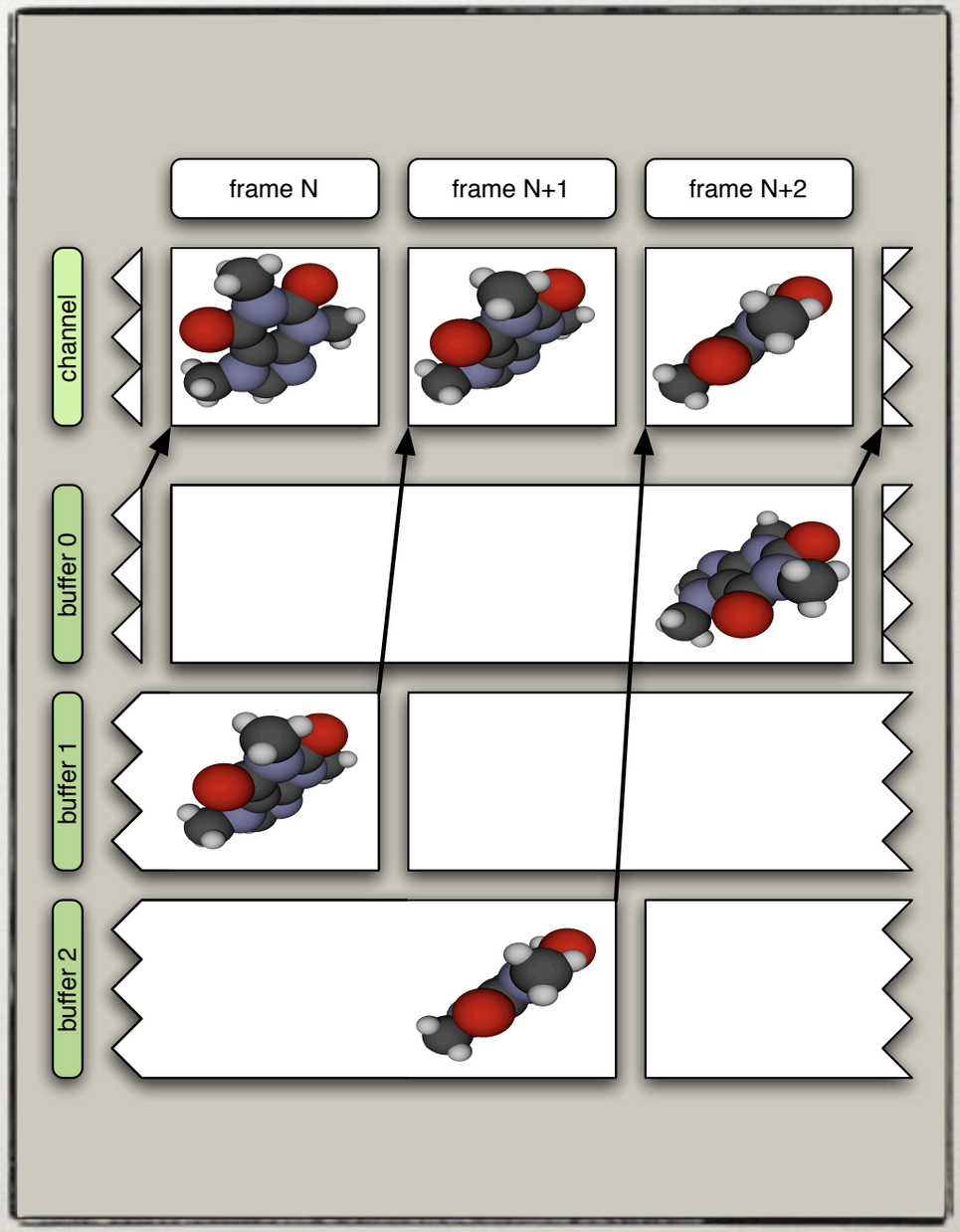
Pixel

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



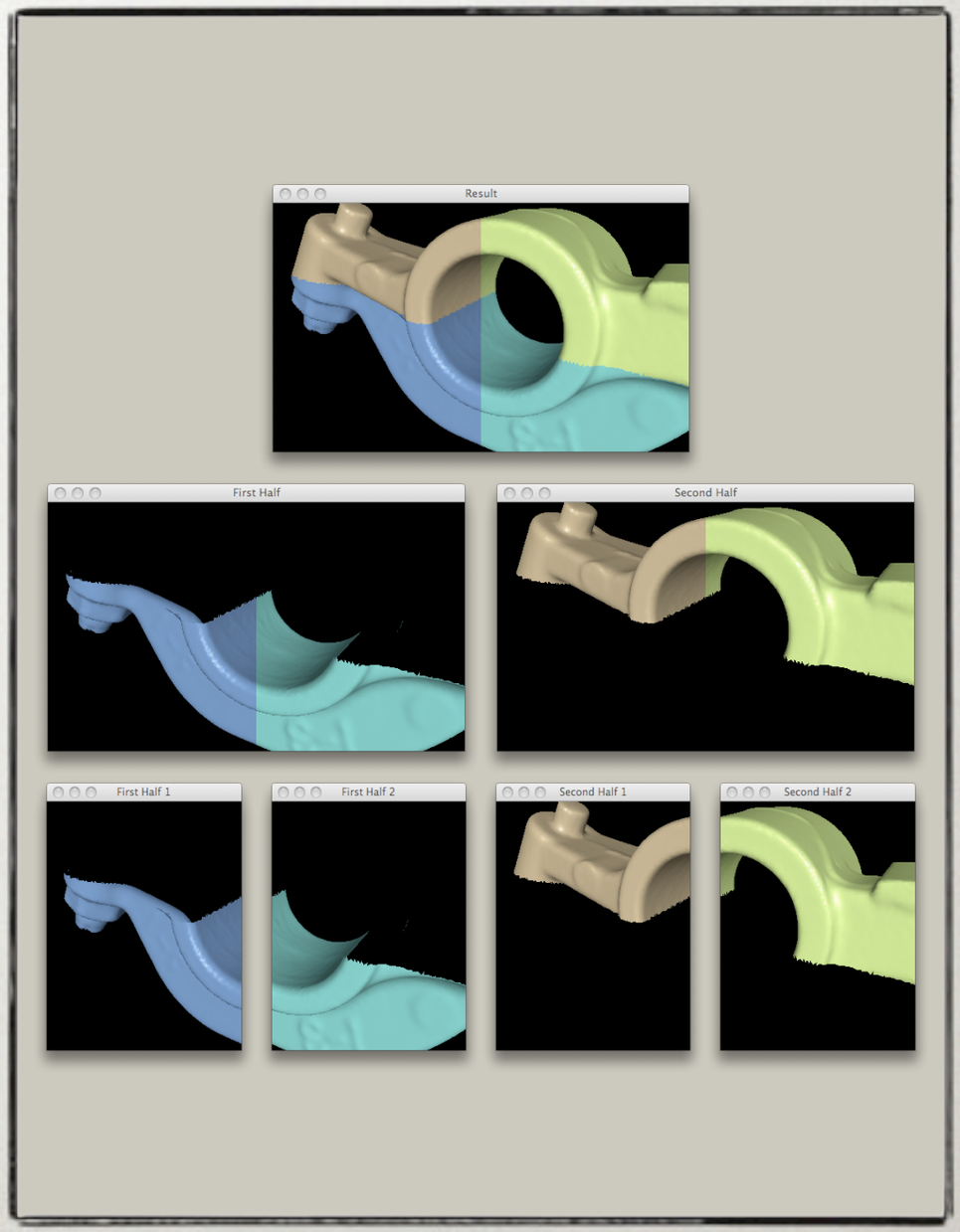
DPlex / Time-Multiplex

- Good scalability and loadbalancing
- Increased latency may be an issue
- Not yet implemented



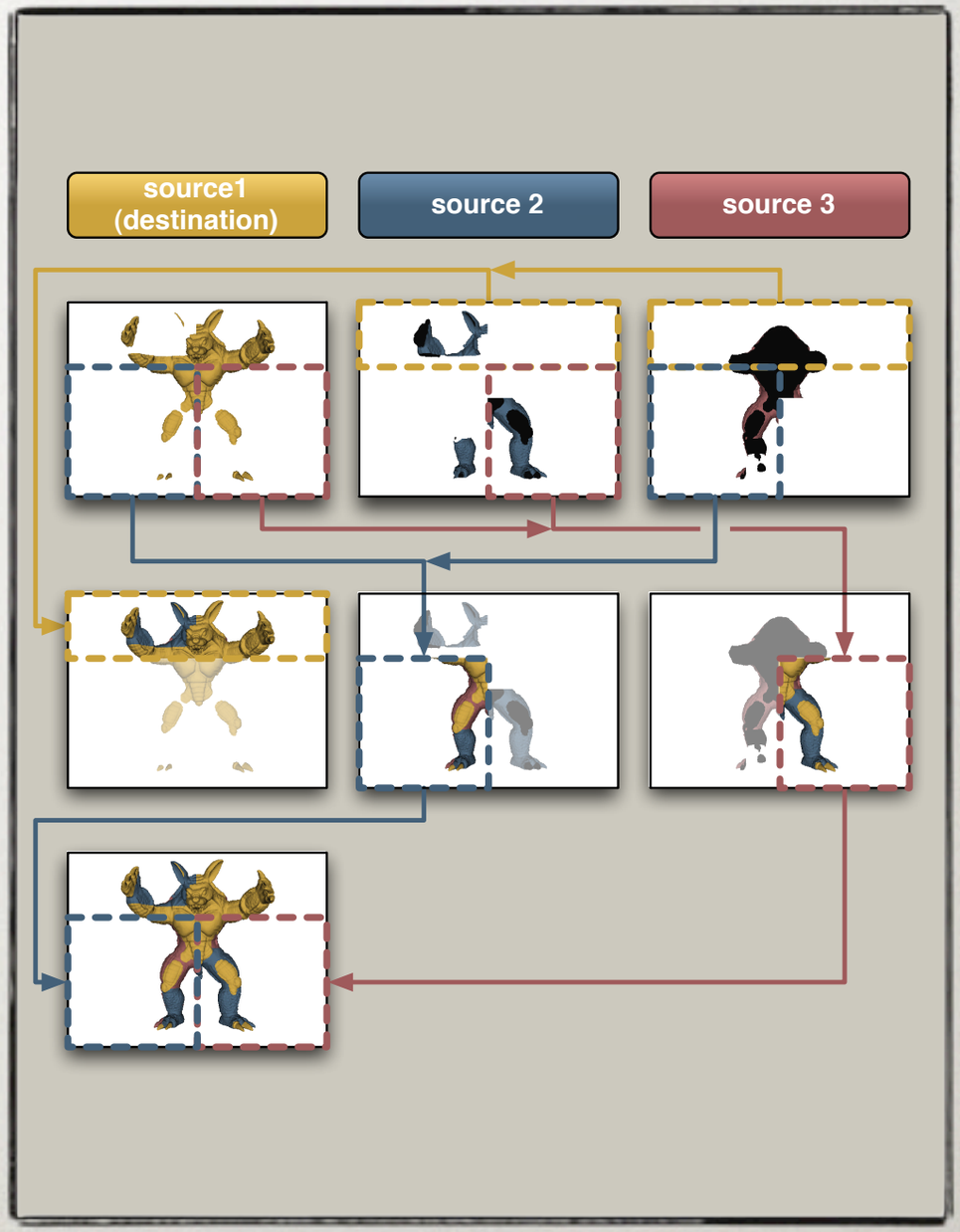
Multilevel Compounds

- 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



Parallel Compositing

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



Compounds

- 2D: low IO overhead, limited scalability
- DB: high IO overhead, great scalability
- Eye: good scalability
- Pixel: linear fill-rate scalability

➡ Combine modes

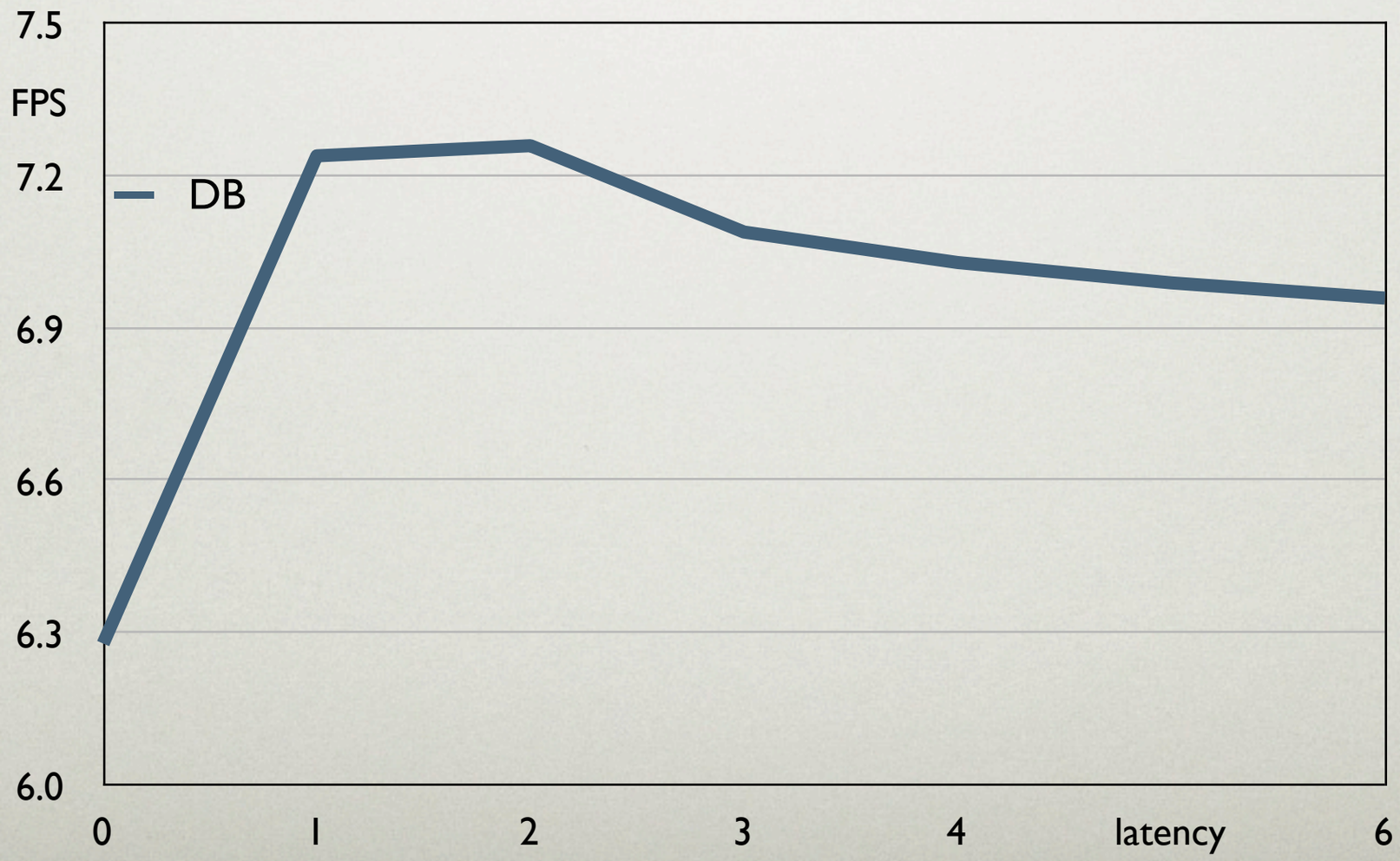
➡ DB: use parallel compositing

Asynchronous Execution

- Latency between last draw and main
- Hides imbalance in load distribution
- Visible channels from the same view are synchronized with swap barrier
- Optional per-node synchronization
- Improves scalability on bigger clusters

Asynchronous Execution

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

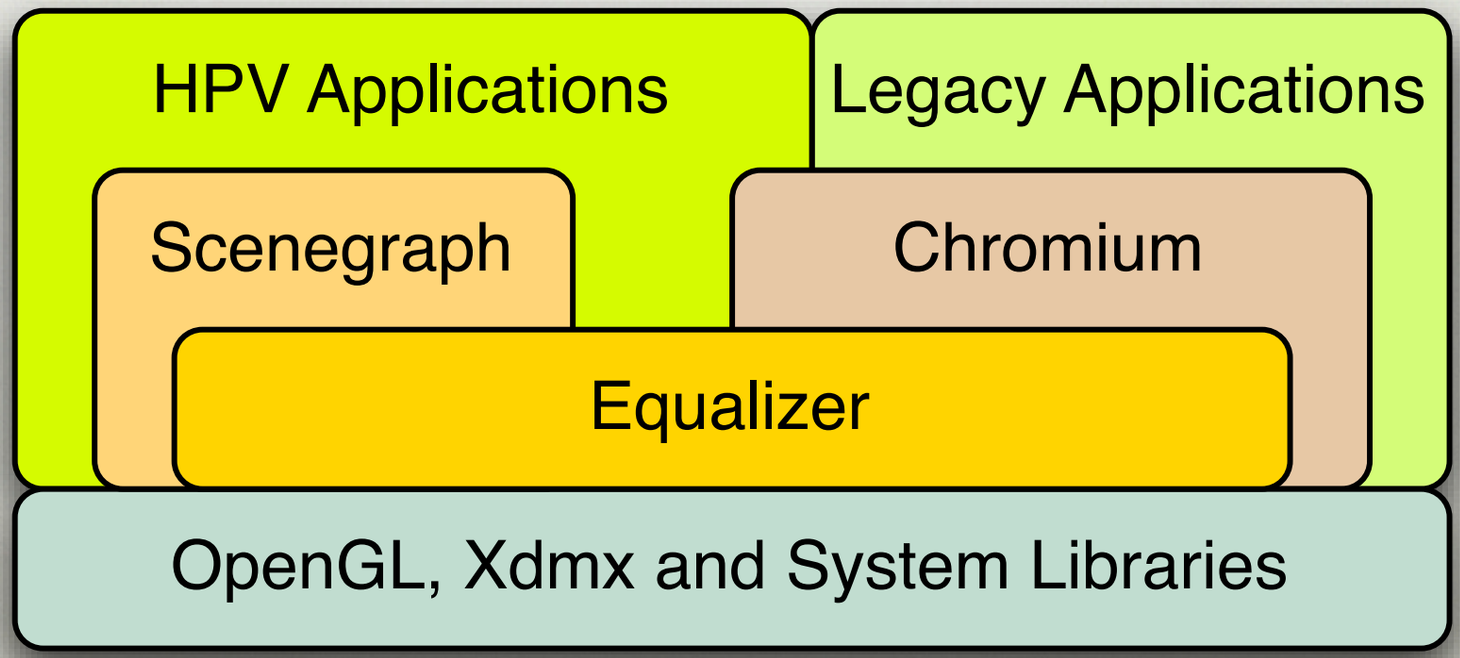


Multi-GPU and Clusters

- 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
- 2D and DB load-balancing
- Data processing
- Examples, demos, applications
- Failure robustness

Last Words

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