Parallel Graphics Programming with Equalizer

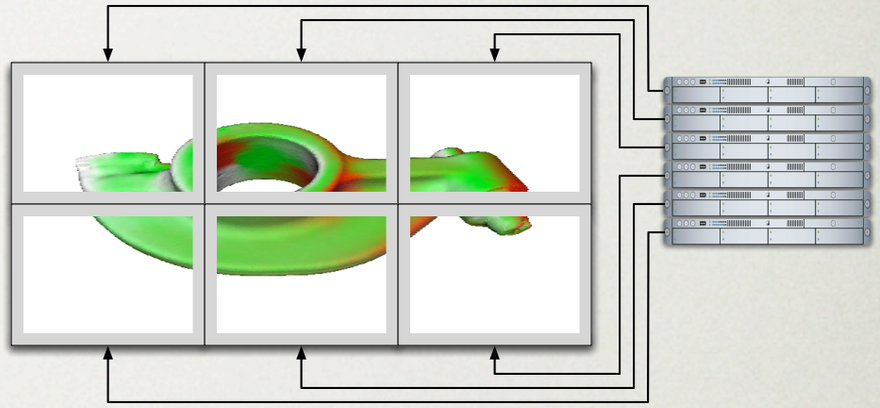
Latest version at http://www.equalizergraphics.com/documents/ParallelGraphicsProgramming.pdf

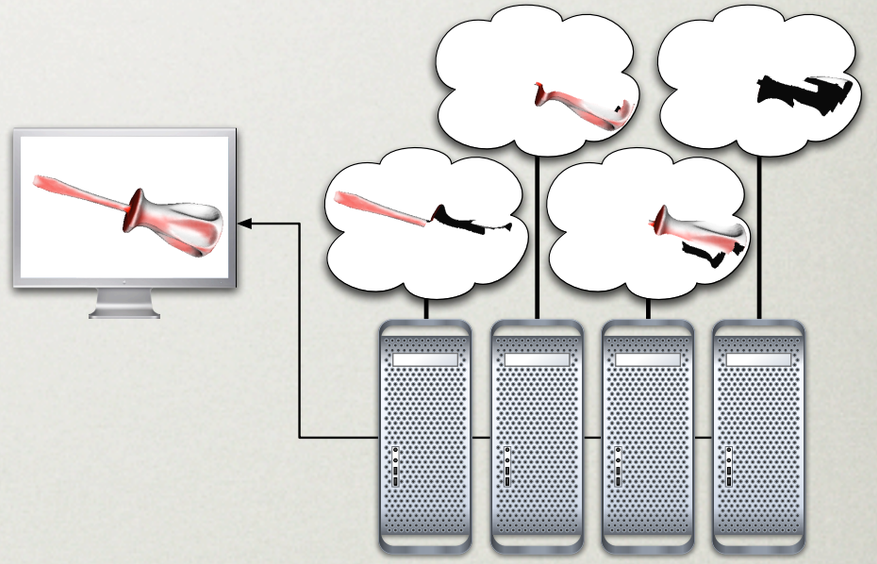
Outline

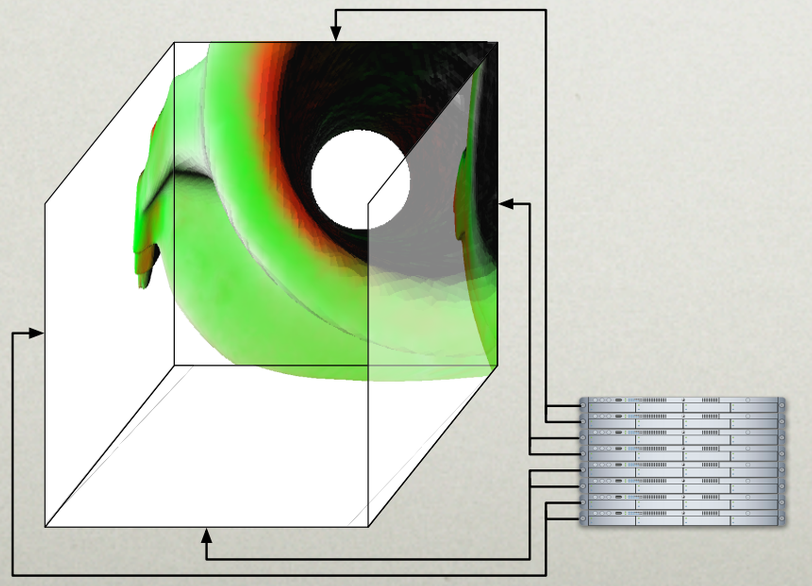
- Environment
- Multipipe Programming
- Equalizer Programming

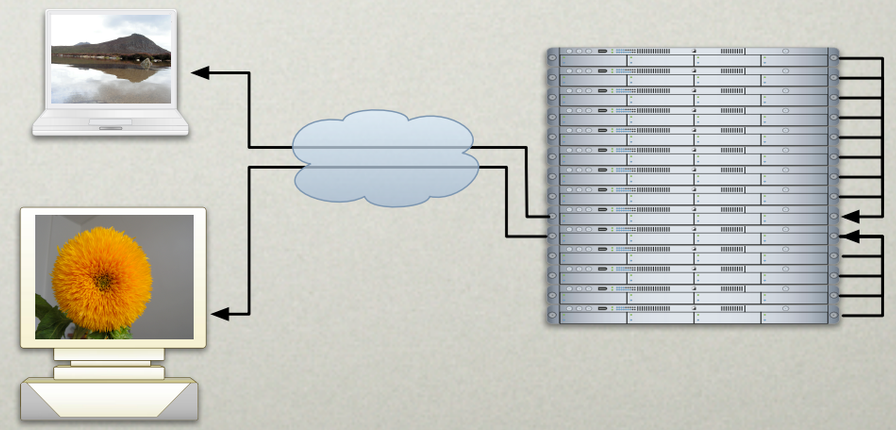
Environment

- Display Walls
- Virtual Reality
- Remote Rendering
- Scalable Rendering



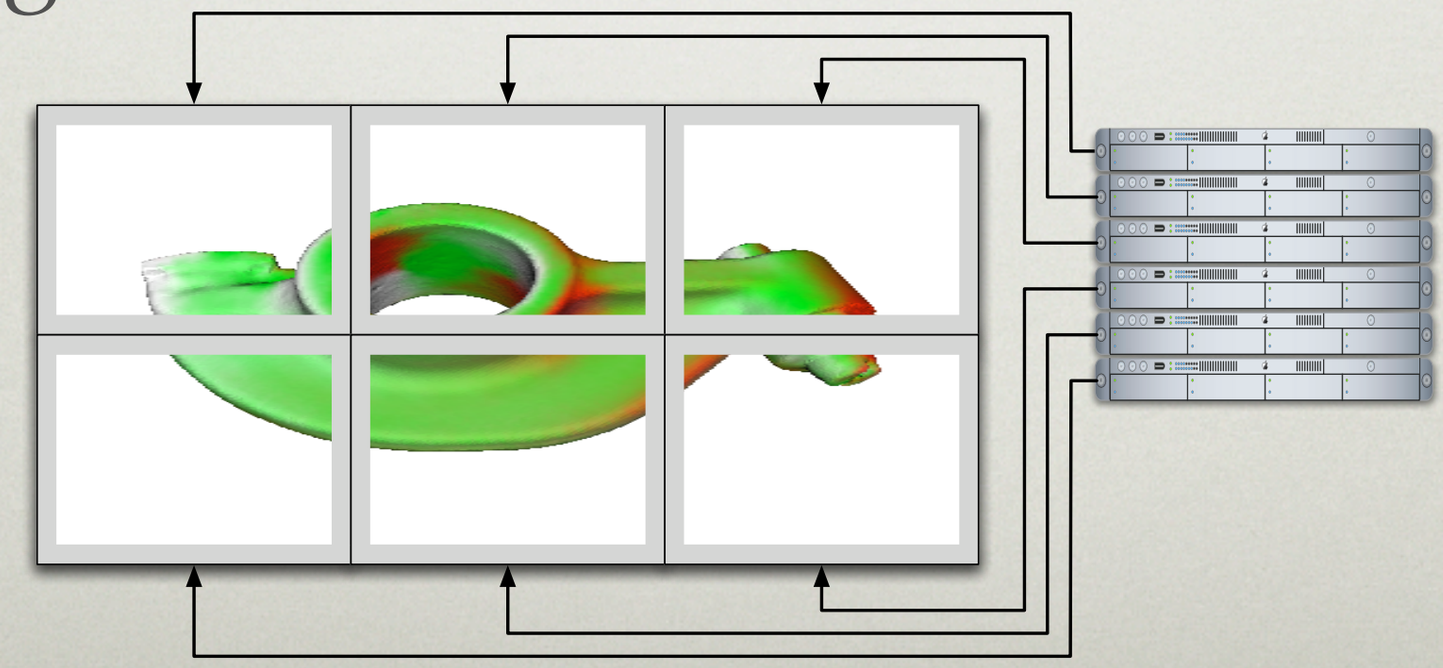






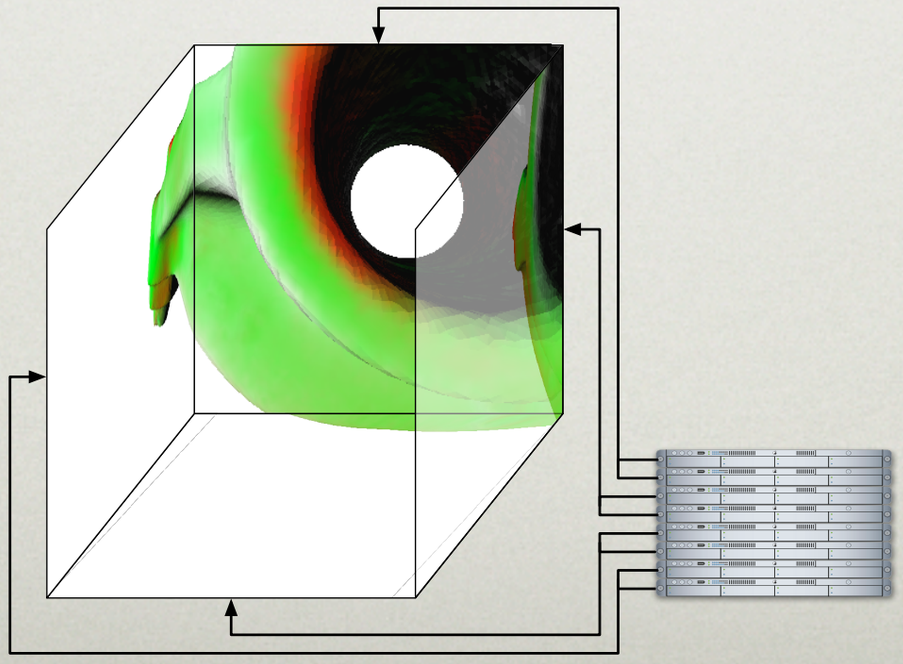
Display Walls

- Group collaboration
- Better data understanding
- One or two displays per computer
- High resolution: 10-100 MPixels



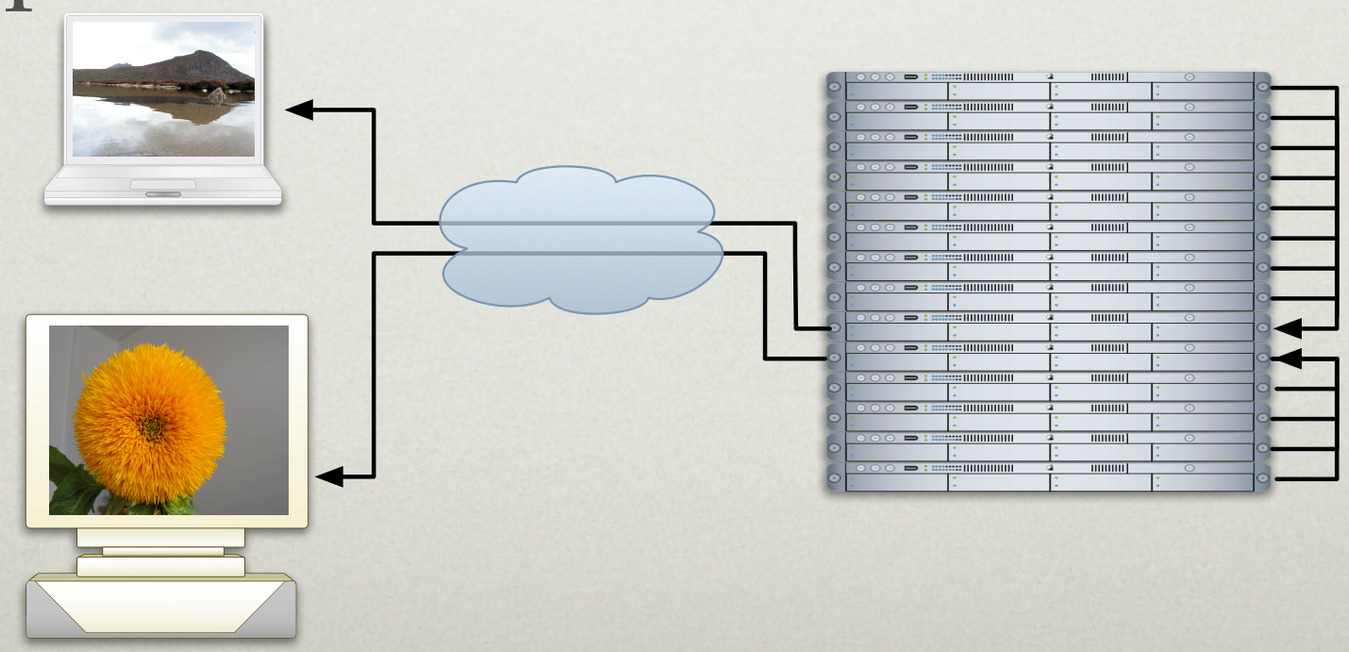
Virtual Reality

- Stereo rendering, head tracking
- High frame rates
- Up to two computers per wall with passive stereo



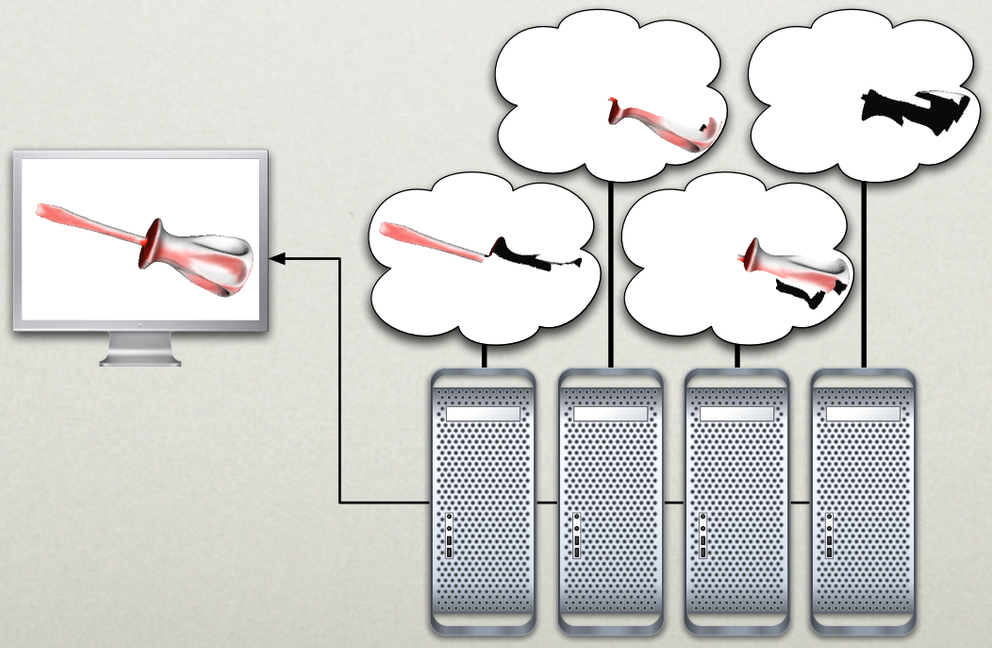
Remote Rendering

- Centralize data, software and hardware
- Can combined with scalable rendering
- Avoids copying of HPC result data
- Simplifies administration



Scalable Rendering

- Render more data faster
- Multiple graphics cars and processors per display
- Different algorithms for parellelization

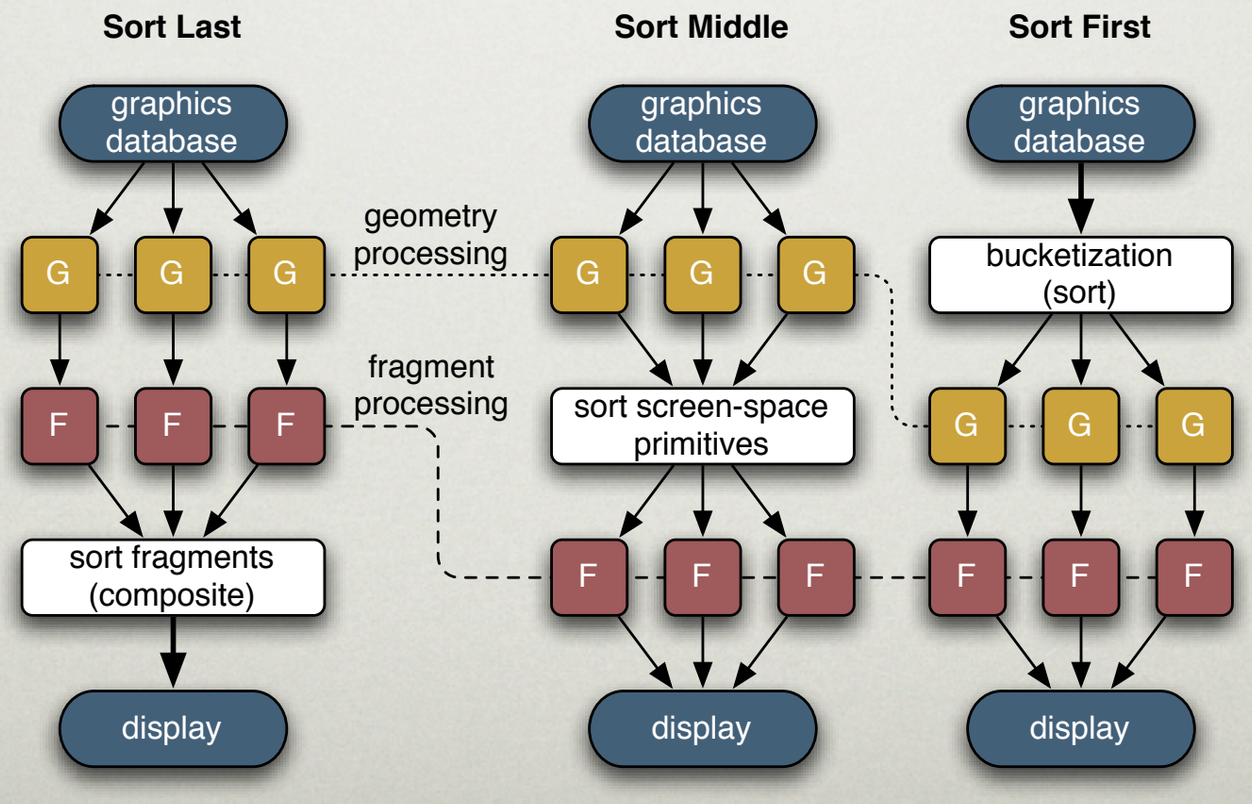


Parallelization

- Single frame decomposition
  - sort-first: screen-space decomposition
  - sort-middle: only practical on GPU
  - sort-last: database decomposition
- Entire frame decomposition
  - DPlex: time-multiplex
  - Eye: stereo passes

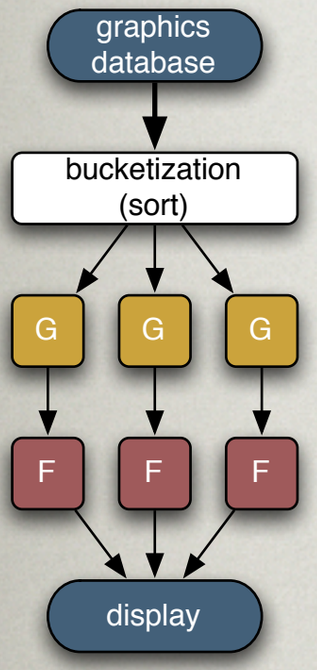
Single frame decomposition

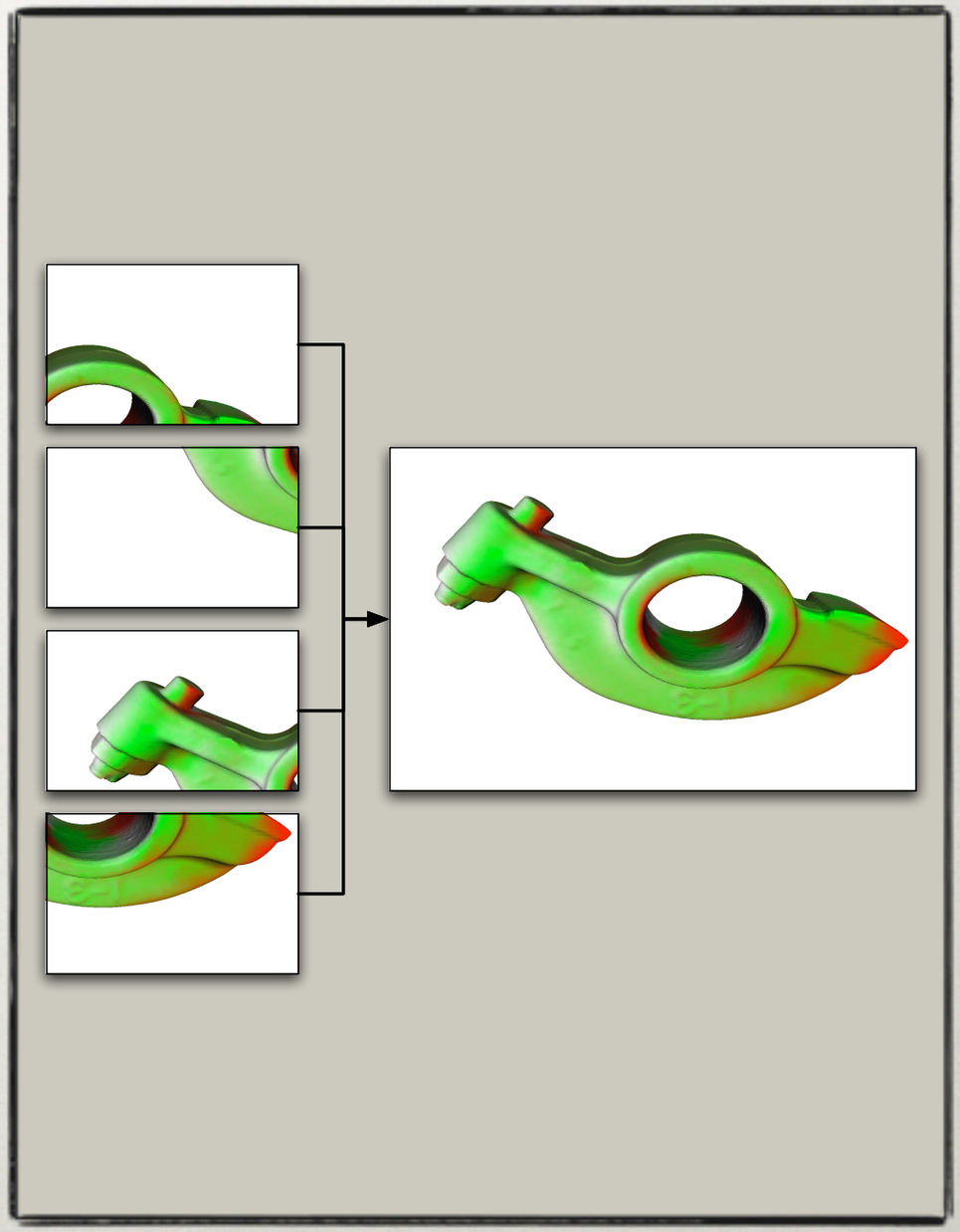
1. Transform primitives into screen space
2. Rasterize primitives into fragments
3. Process fragments into pixels



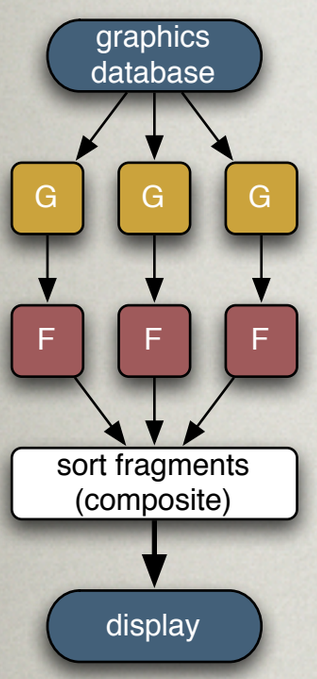
2D / Sort-First

- Scales fillrate / fragment processing
- Scales geometry when used with view frustum culling
- Parallel overhead due to primitive overlap limits scalability

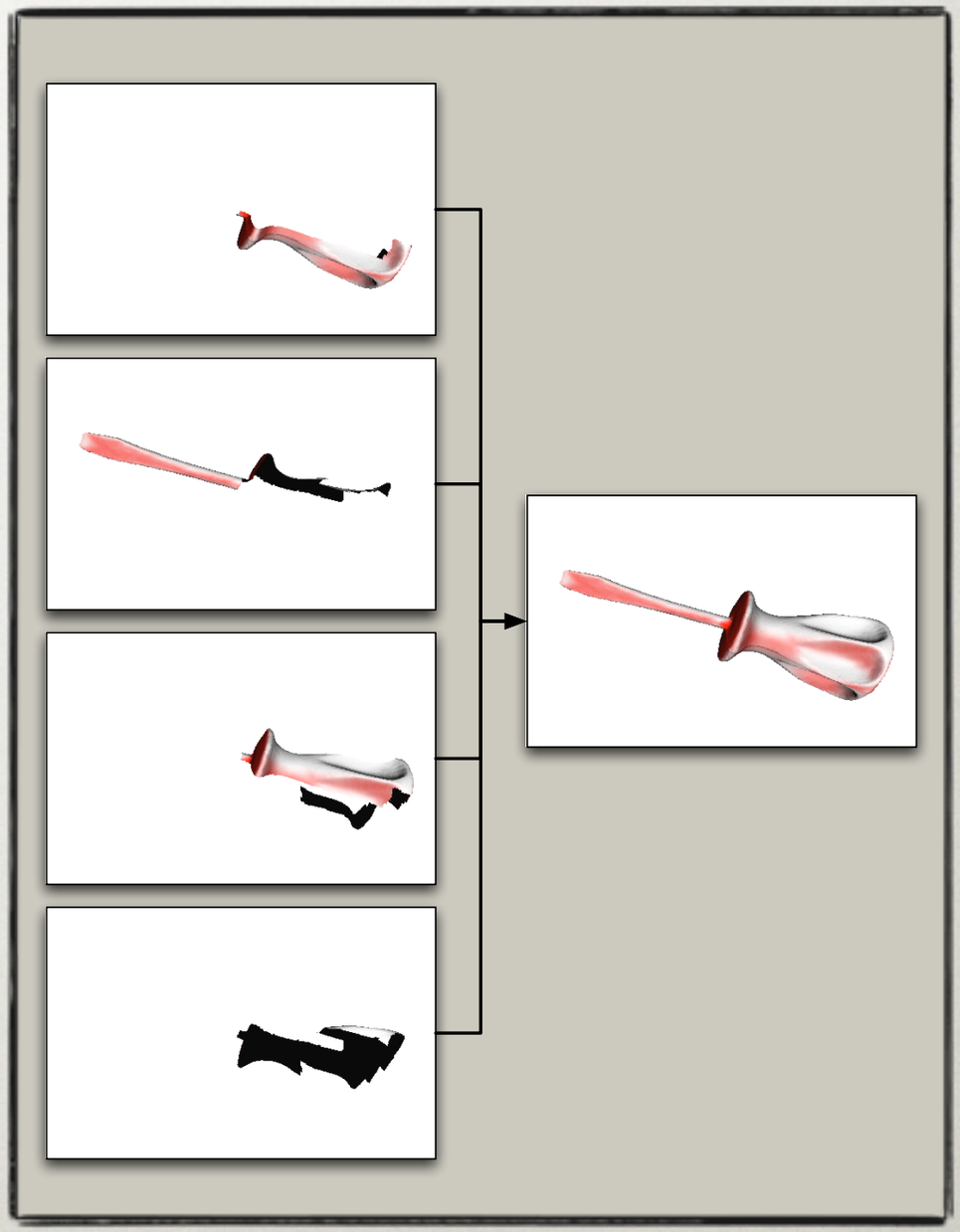




DB/Sort-Last

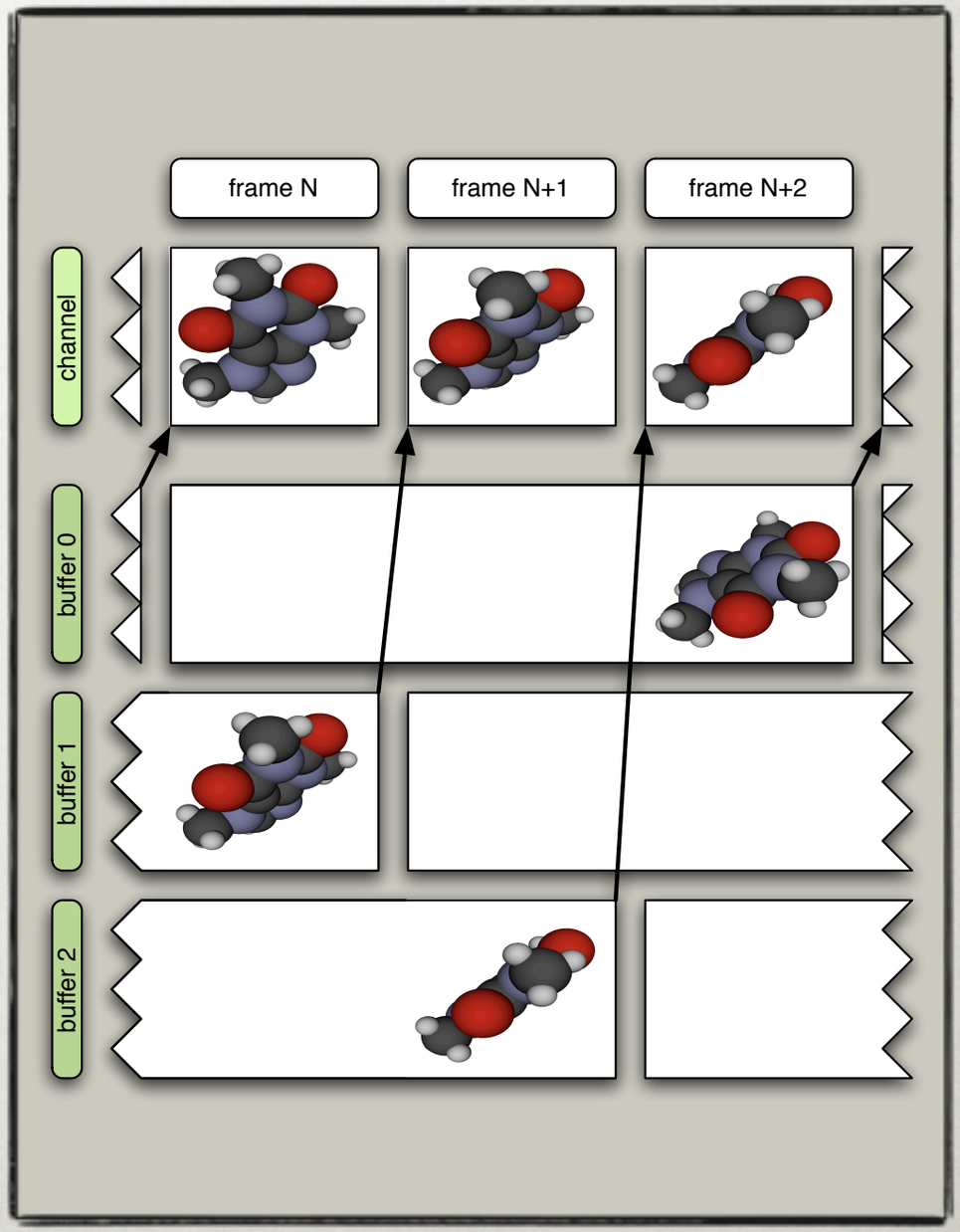


- Scales all aspects of rendering pipeline
- Application needs to be adapted to render subrange of data
- Recomposition relatively expensive



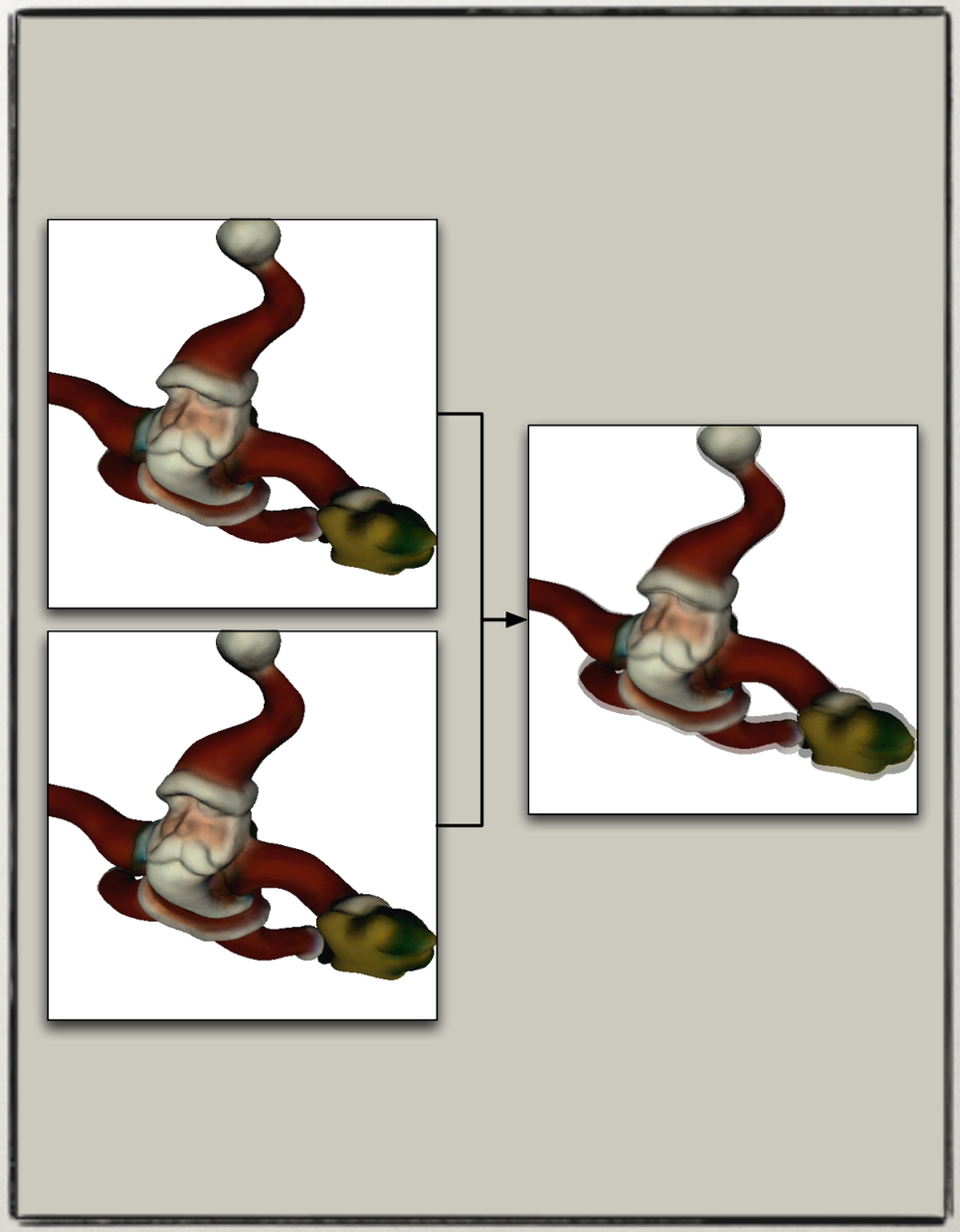
DPlex / Time-Multiplex

- Good scalability and loadbalancing
- Increased latency may be an issue



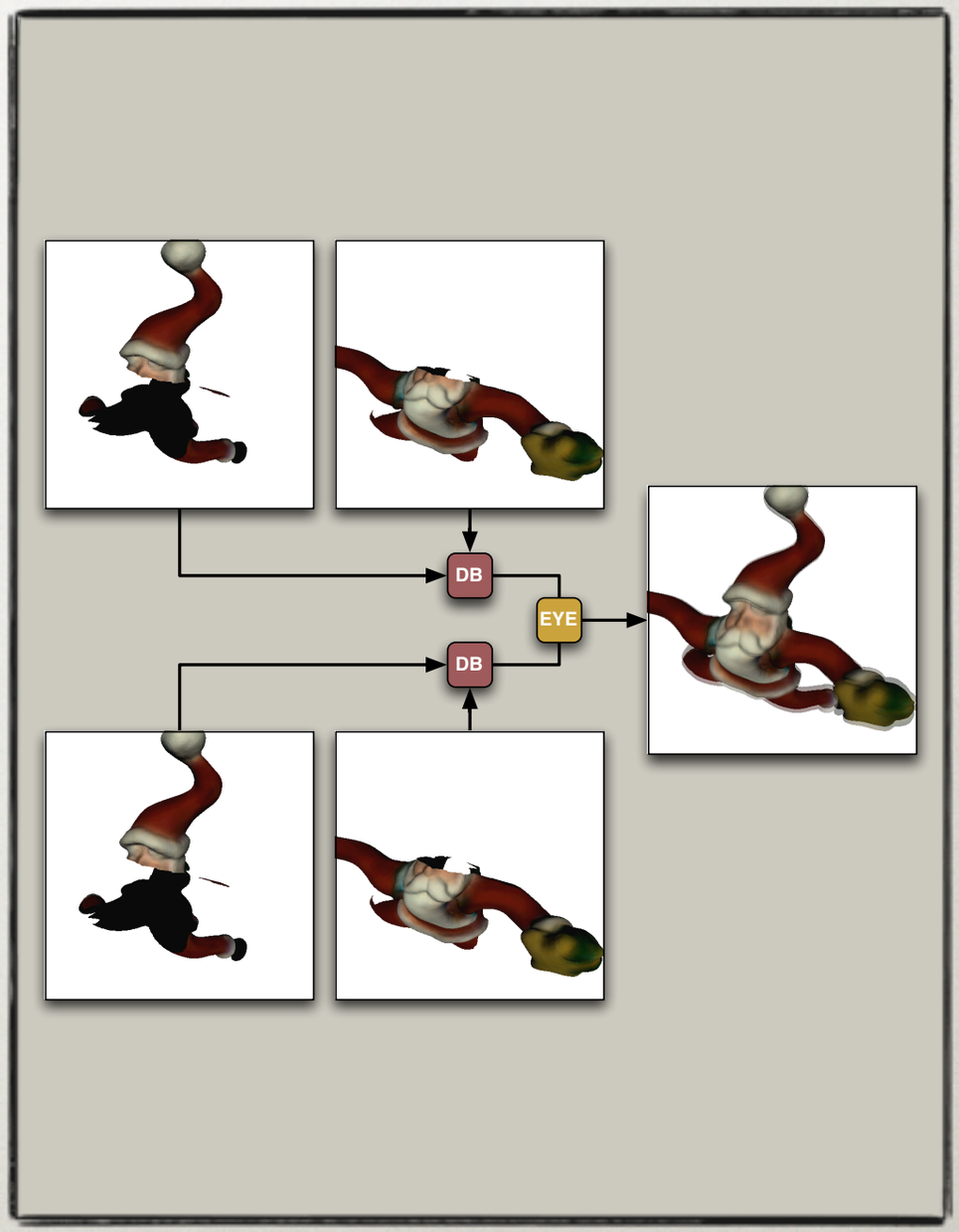
Eye/Stereo

- Stereo rendering
- Excellent loadbalancing
- Limited by number of eye views



Multilevel

- Combine different algorithm to balance bottlenecks
- Flexible configuration of recomposition algorithm

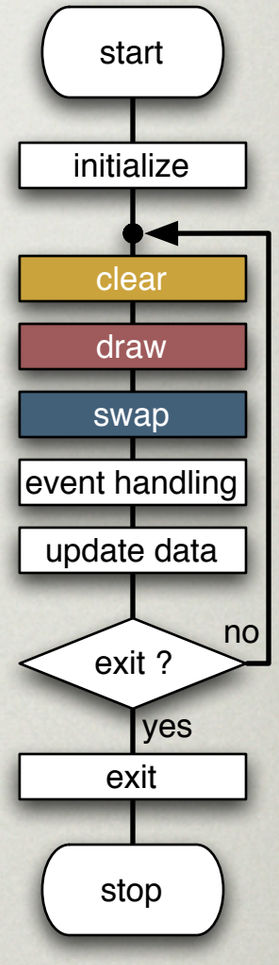


Multipipe Programming

- Single pipe application
- Multipipe porting

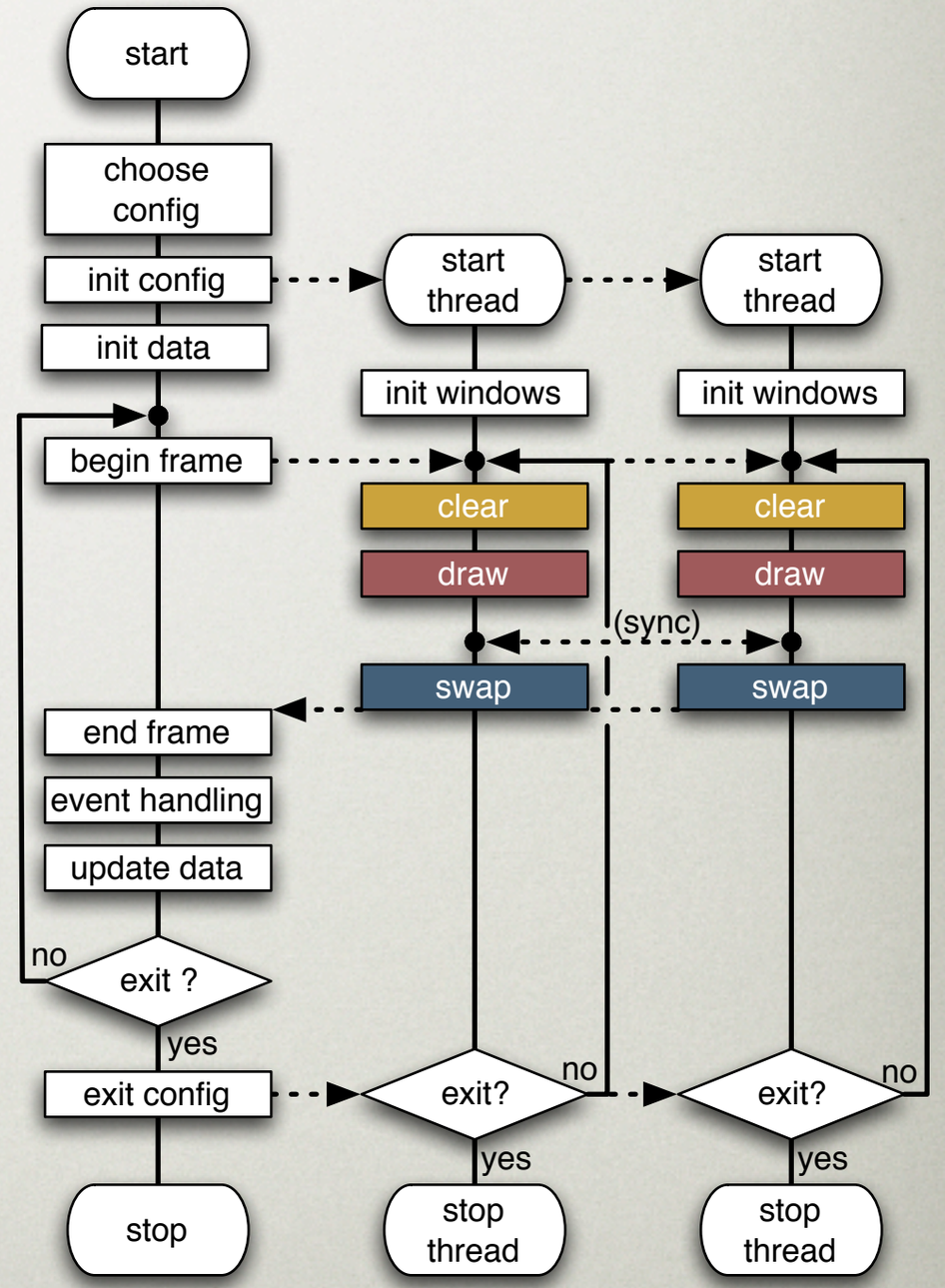
Single Pipe Rendering

- Typical rendering loop
- Stages may not be well separated

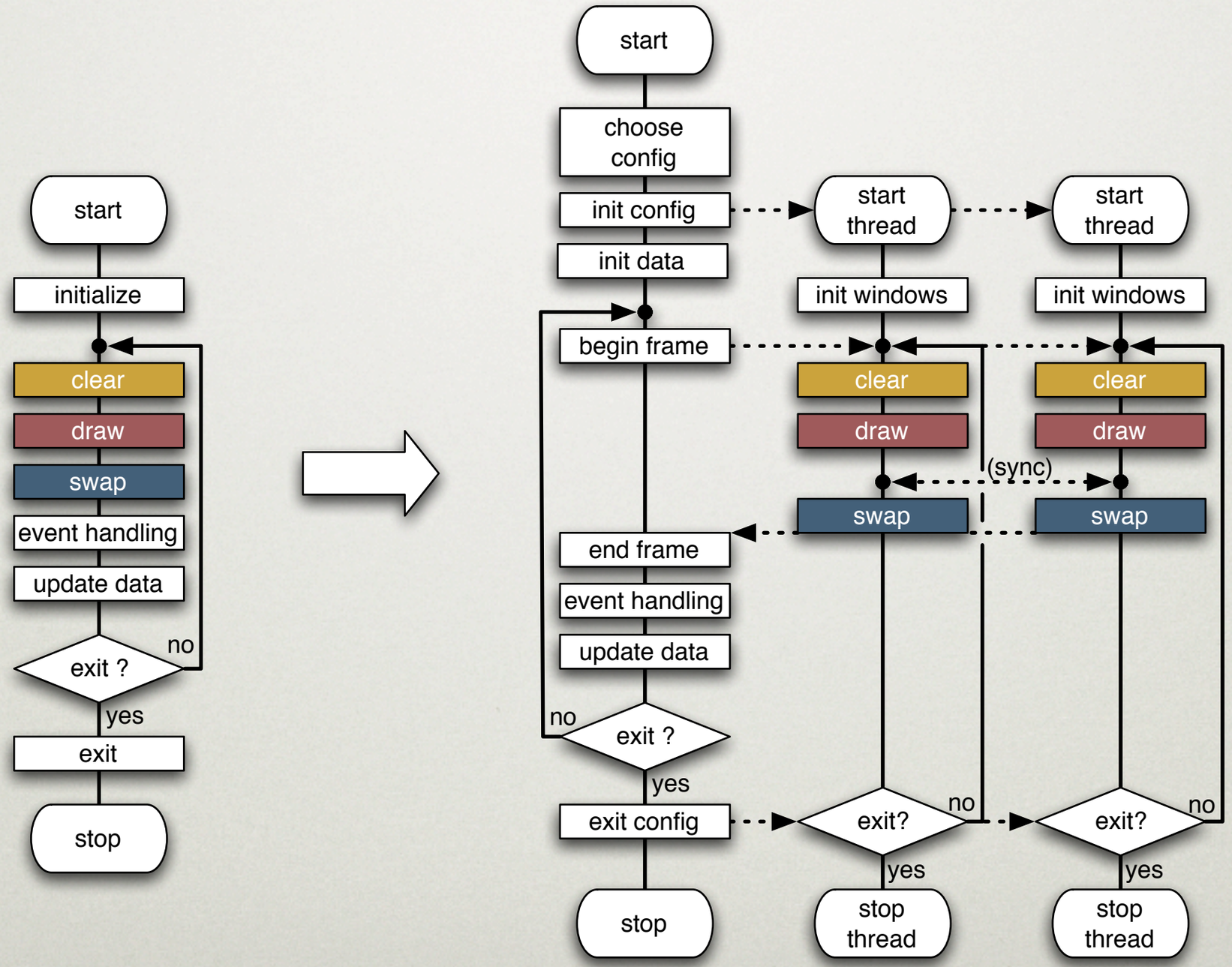


Multipipe Rendering

- Separate rendering and application
- Instantiate rendering multiple times
- Synchronize parallelel execution



Single Pipe to Multipipe



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 transport
- Supports distributed rendering and performs frame compositing

Equalizer Programming Interface

C++ classes for corresponding graphics entities

- **Node** is a single computer in the cluster
- **Pipe** is a graphics card
- **Window** is an OpenGL drawable
- **Channel** is a viewport within a window

Equalizer Programming Interface

Application sub-classes and overrides task methods, e.g.:

- **Channel::draw** to render using the provided frustum, viewport and range
- **Window::init** to init OpenGL drawable and state
- **Pipe::startFrame** to update frame-specific data
- **Node::init** to initialize per node application data

Default methods implement typical use case

Open Source

- LGPL license
- Open standard for scalable graphics
- User-driven development
- Alpha version on www.equalizergraphics.com
- Get in touch!