Parallel Rendering using OpenGL Multipipe SDK (MPK)

Praveen Bhaniramka

SGI

Talk Outline

- Overview
- Application Structure
- Configuration Interface
- Parallel Rendering
- Related Projects/Approaches
- More Information

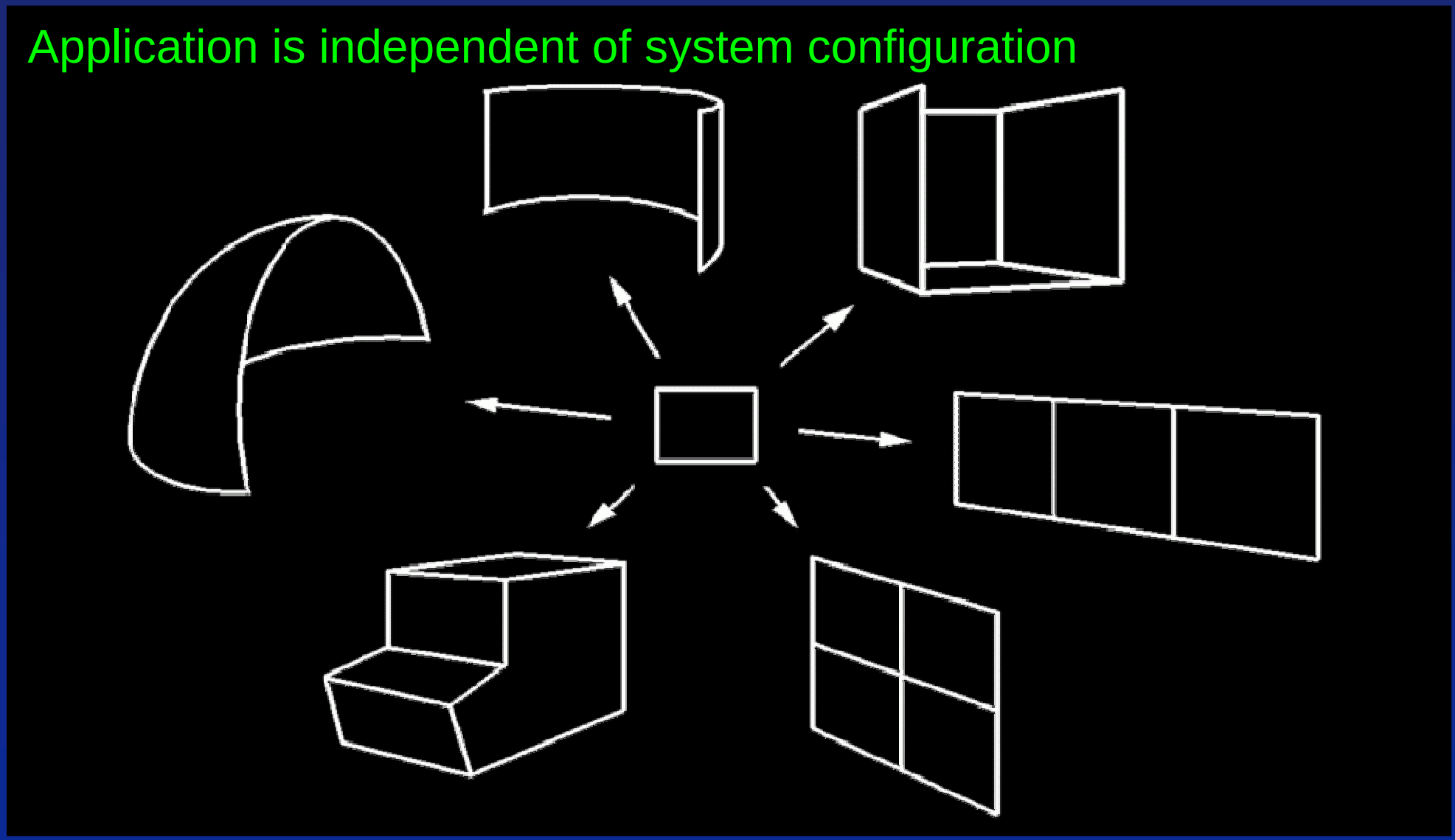
Overview

- Design Objectives/Features
  - Provide an API for development of OpenGL-based Multipipe applications
  - ‘Runtime portability’ from desktop to multi-pipe systems
  - ‘Runtime scalability’
  - Minimize invasiveness

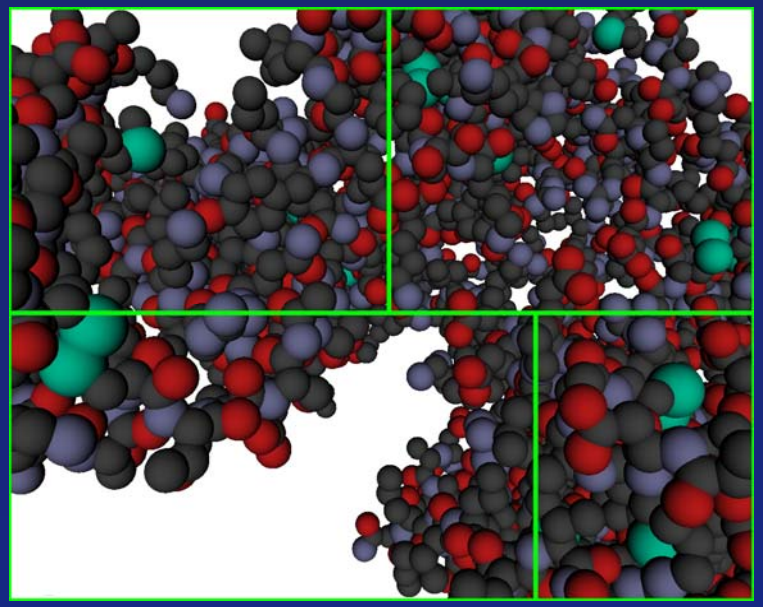
  Sort of like a multipipe GLUT

Runtime Portability

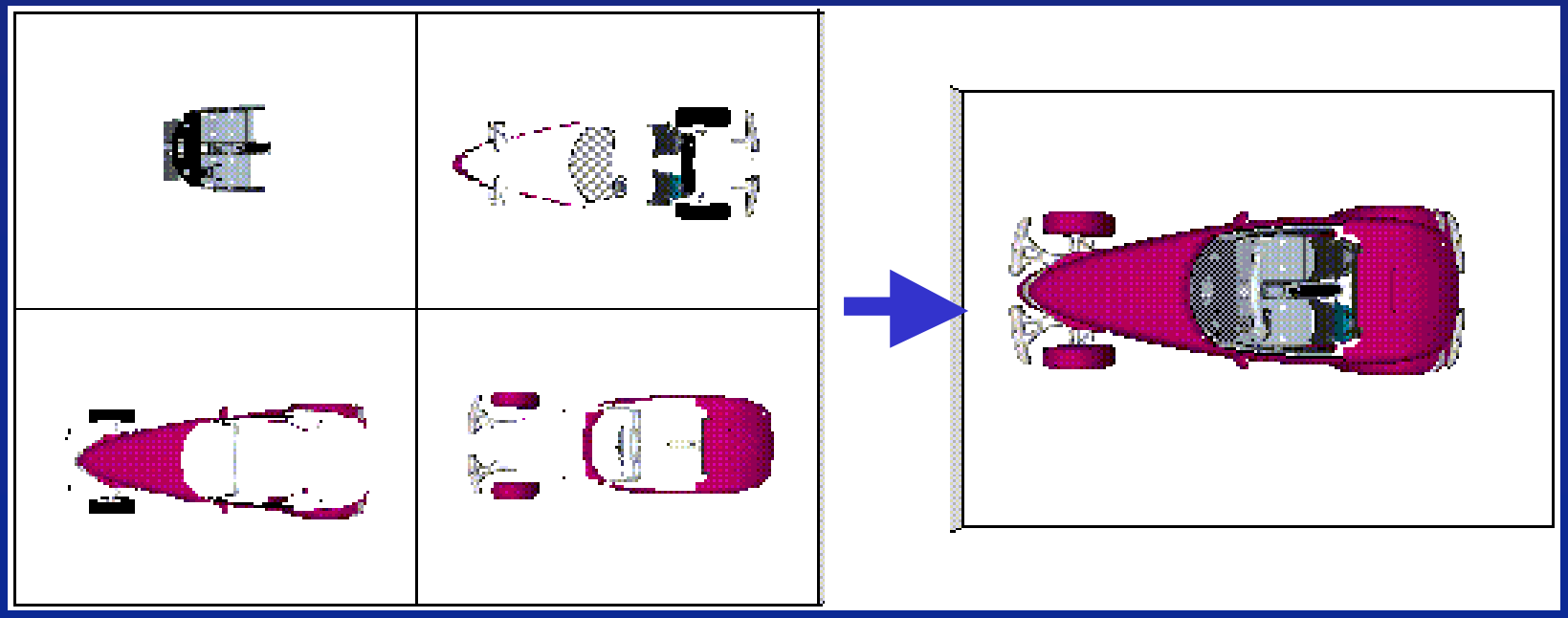
- From Single-pipe systems to Multi-pipe systems



Runtime Scalability



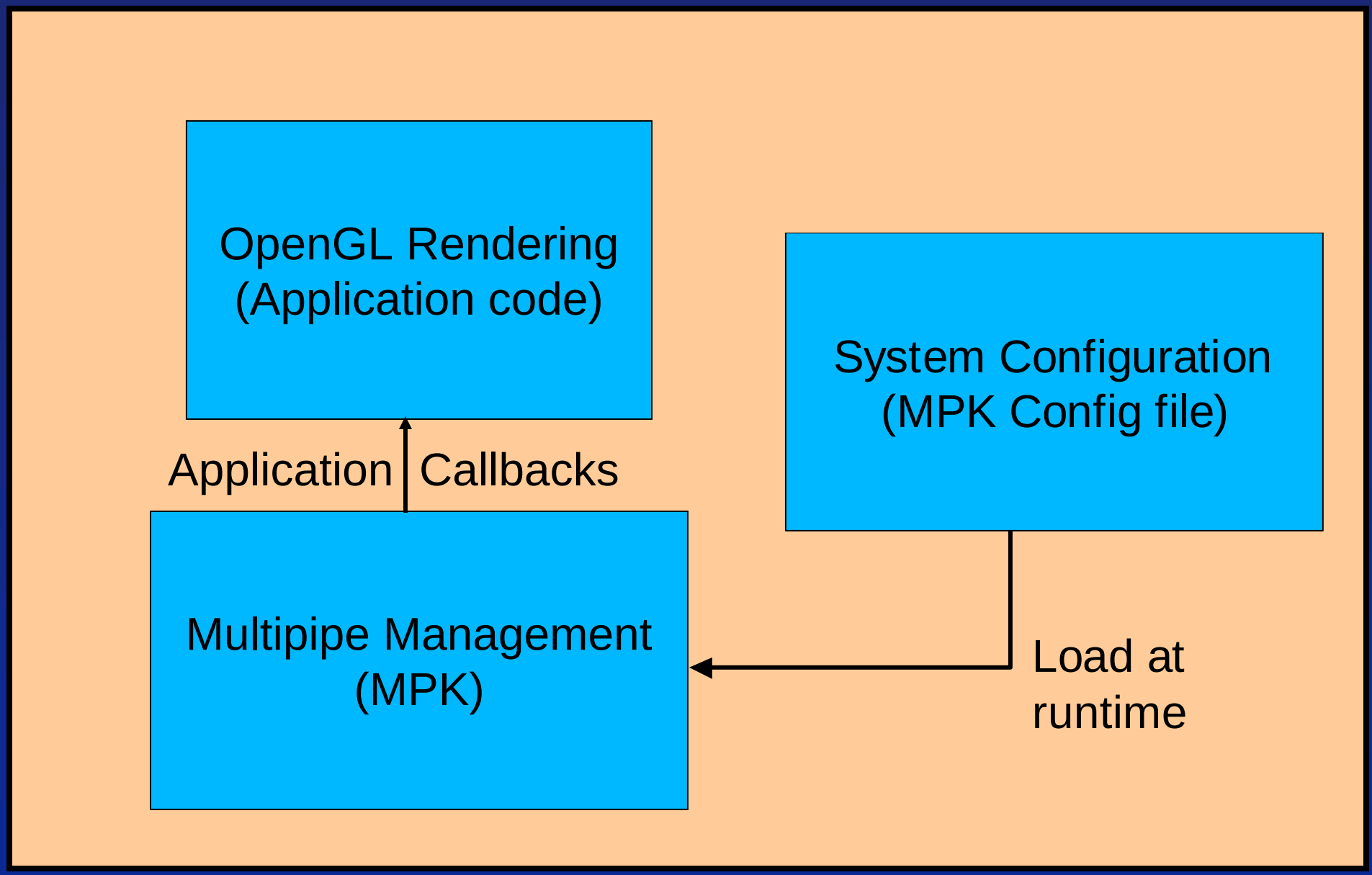
Application scales as hardware resources are added



Minimal Invasiveness

- Callback driven
- Basic OpenGL Framework
- Relatively simple GLUT-like C API
- Channel & Stereo Independent

Application Structure

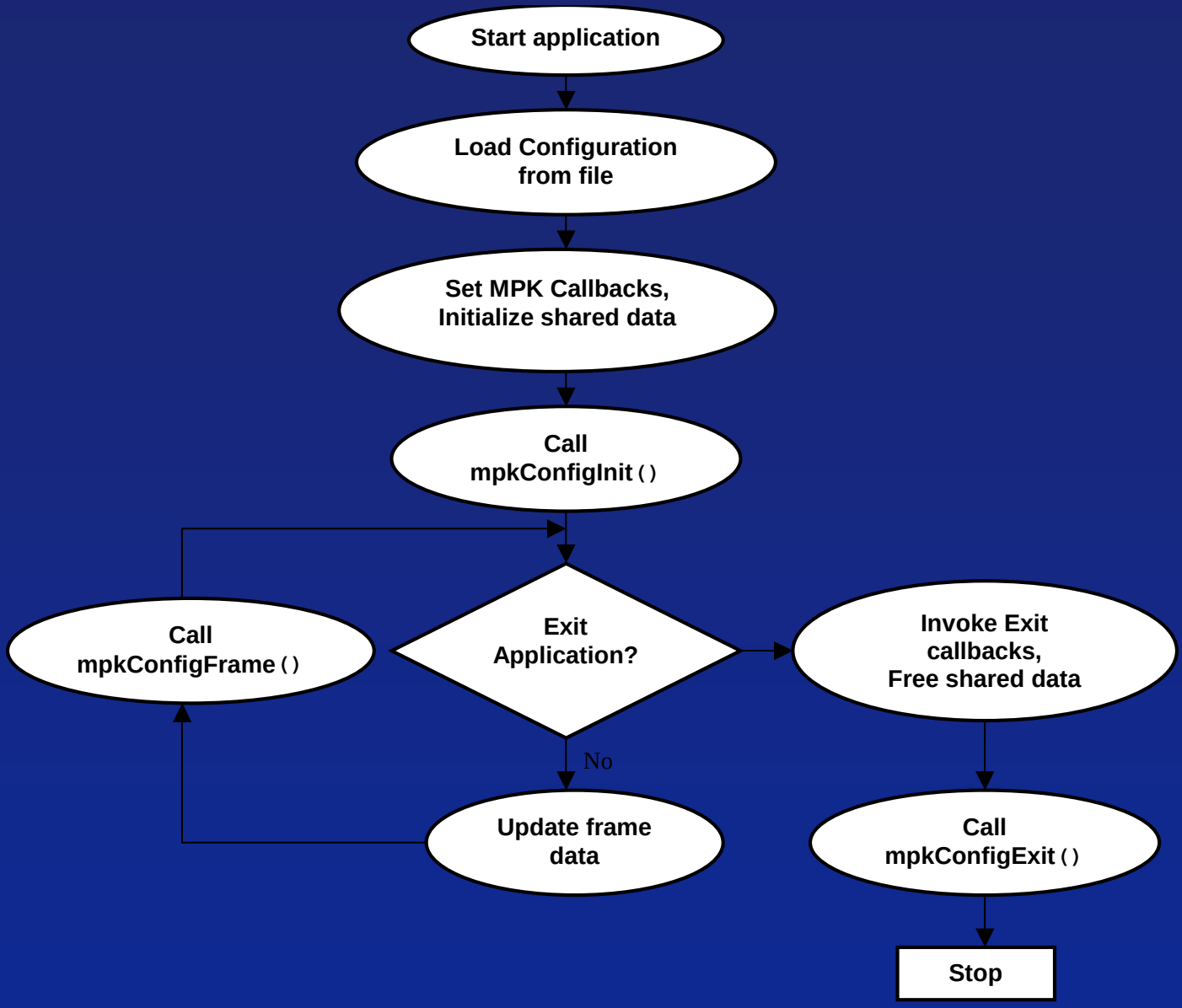


System Configuration

- MPKConfig
  - Hierarchical Description of System Configuration
  - Specifies the relationships between different components
- MPKChannel
  - Basic, display device independent OpenGL rendering unit.
  - These are “framebuffer resources”
- MPKConfig Hierarchy
  - A Config has one or more Pipes
  - A Pipe has one or more Windows*
  - A Window has one or more Channels

    * Each window has a dedicated rendering thread

Application Flow



Application Flow

Render execution flow is controlled by ...

mpkConfigFrame()

... which leads to execution of ...

the draw scene callback (per channel)

- Executes one frame of rendering
- Window threads invoke update callbacks
- Passed framedata is distributed latency-correct

Application

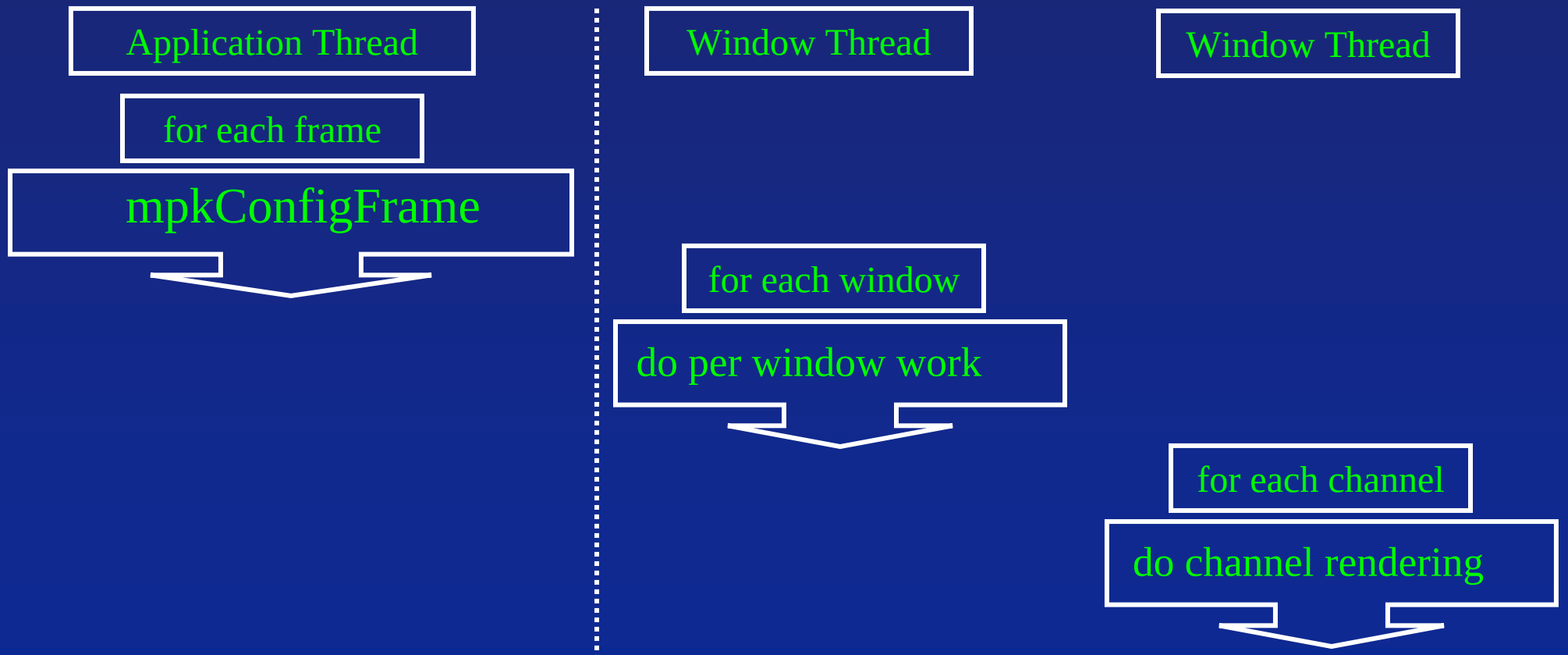
```
#include <mpk/mpk.h>
// main()
mpkInit();
MPKConfig* cfg = mpkConfigLoad(configFileName);
//read config
mpkConfigSetWindowInitCB(cfg,initWindowCB);
// initialize window callbacks
mpkConfigInit(cfg);
// start: spawn one thread per window
// rendering loop
while ( notDone ) {
   // do application work (spin the cow, make sharks
   // swim, etc)
   updateSharedData( &framedata );
   // sync window loop threads, etc.
   mpkConfigFrame( cfg, framedata );
}
mpkConfigExit(cfg);
```

Application

```
// updateChannel()user callback: invoked by
// mpkConfigFrame() => glutDisplayFunc
// clear render area
mpkChannelApplyBuffer( c );
mpkChannelApplyViewport( c );
glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT );
// apply projection matrix
glMatrixMode( GL_PROJECTION );
glLoadIdentity();
mpkChannelApplyFrustum( c );
// apply modelview matrix
glMatrixMode( GL_MODELVIEW );
glLoadIdentity();
mpkChannelApplyTransformation( c );
// cull and/or draw cow, sharks, etc.
drawSharedData( c, framedata );
```

Rendering Flow

Pipe => Window => Thread => GL Context => Channel



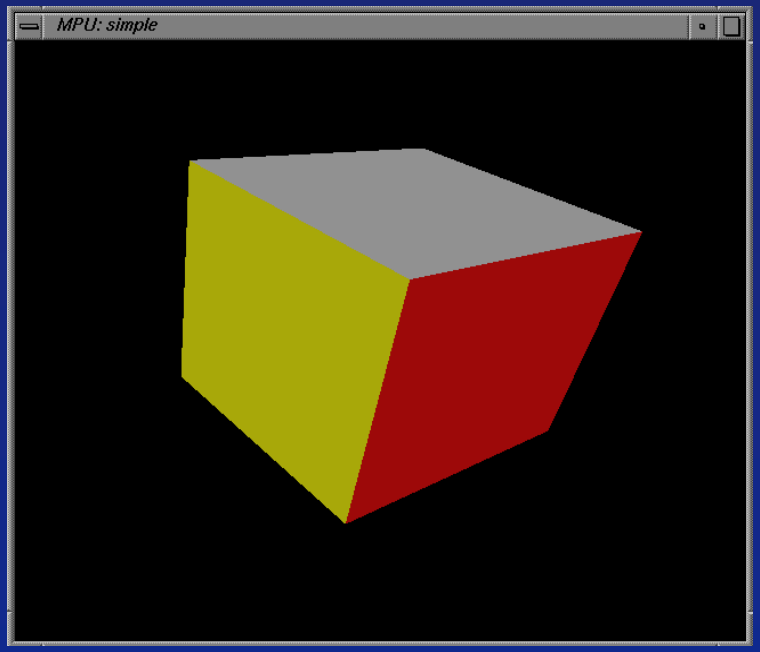
Config File

- Simple ASCII file representation of MPKConfig data structure
- Hierarchical Description for Framebuffer Resources
- Channel Physical Layout
- Channel Decomposition

Simple Config

1 pipe, 1 window

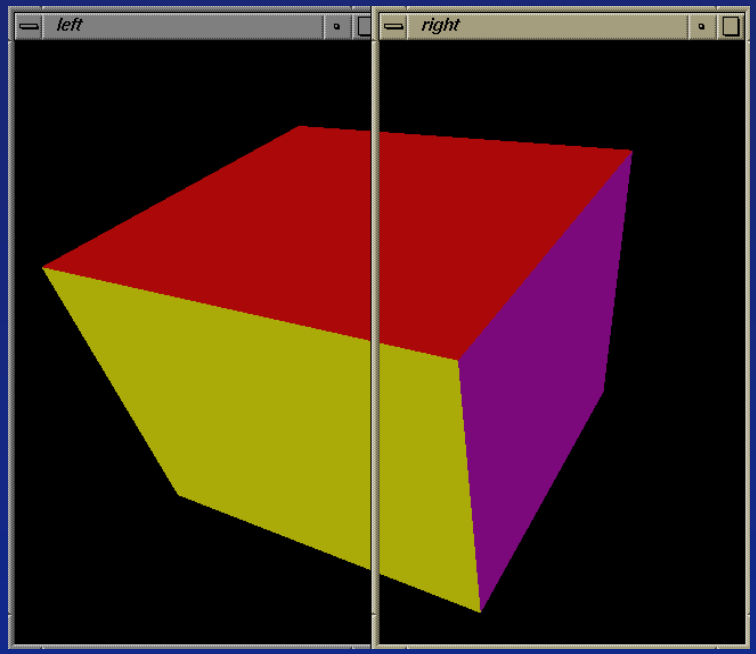
```
config {
    name “1-window”
    pipe {
        window {
        name “MPU:simple”
        viewport [ 0.25 0.25 0.5 0.5]
            channel {
                name “channel”
                wall {
                    bottom_left  [0 0 0]
                    bottom_right [1 0 1]
                    top_left     [1 1 1]
                }
            }
        }
    }
}
```



Simple Config

1 pipe, 2 windows

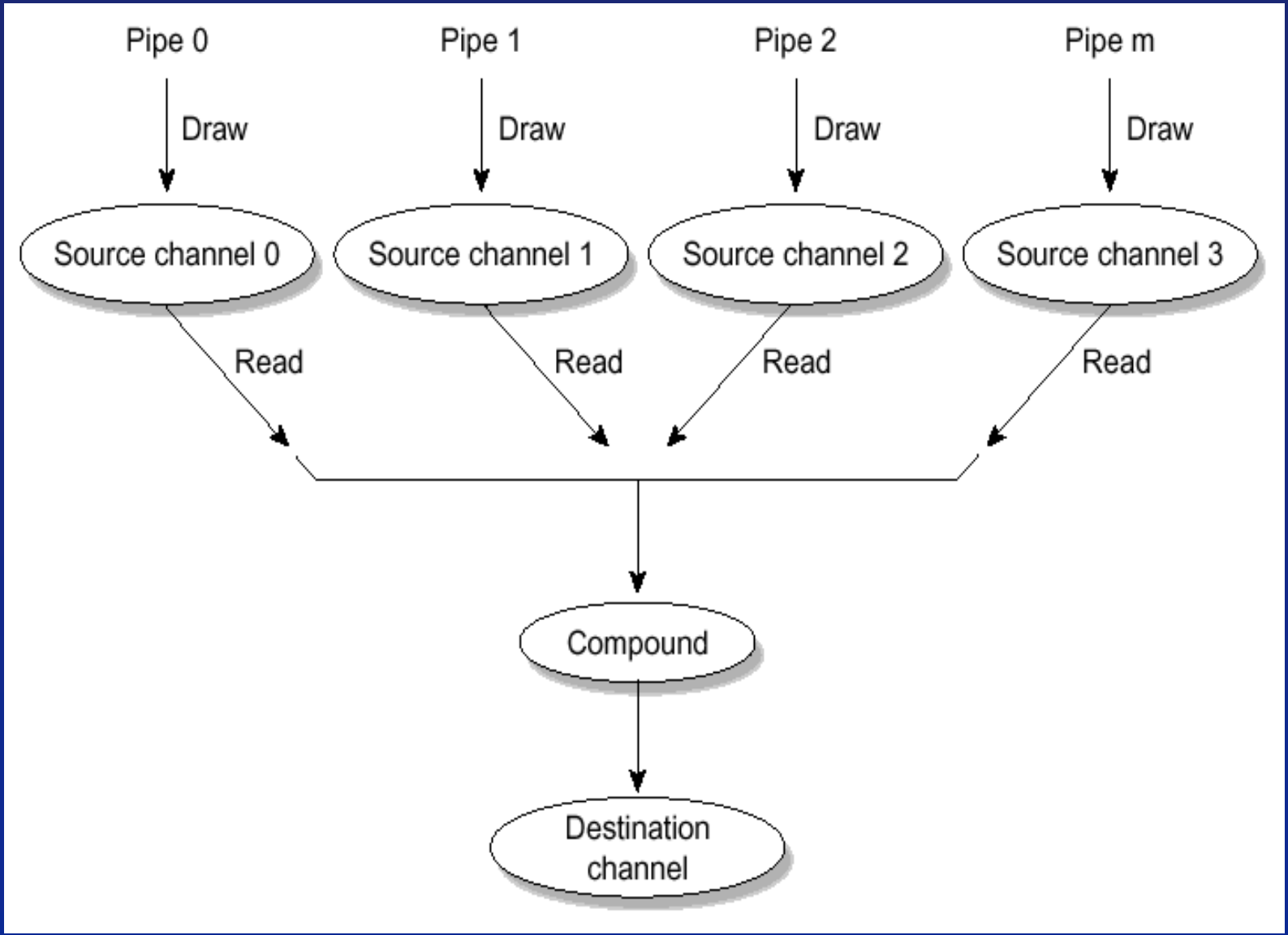
```
config {
    name “2-windows”
    pipe {
        window {
        name “left”
        viewport [ 0.25 0.25 0.25 0.5]
            channel {
                name “channel”
                wall {
                    bottom_left  [0 0 0]
                    bottom_right [1 0 1]
                    top_left     [1 1 1]
                }
            }
        }
        window {
        name “right”
        viewport [ 0.5 0.25 0.25 0.5]
            channel {
            }
        }
    }
}
```



Compounds

- Compounds provide an abstraction for parallel rendering
  - A Config can have one or more compounds
  - Compounds can be hierarchical with a tree-like structure
  - Compounds reference channels as sources and/or destinations
  - SW as well as HW compositing
  - Scaling may require some application awareness

Compounds


Pipe 0
Pipe 1
Pipe 2
Pipe m
Draw
Draw
Draw
Draw
Source channel 0
Source channel 1
Source channel 2
Source channel 3
Read
Read
Read
Read
Compound
Destination channel

Compounds

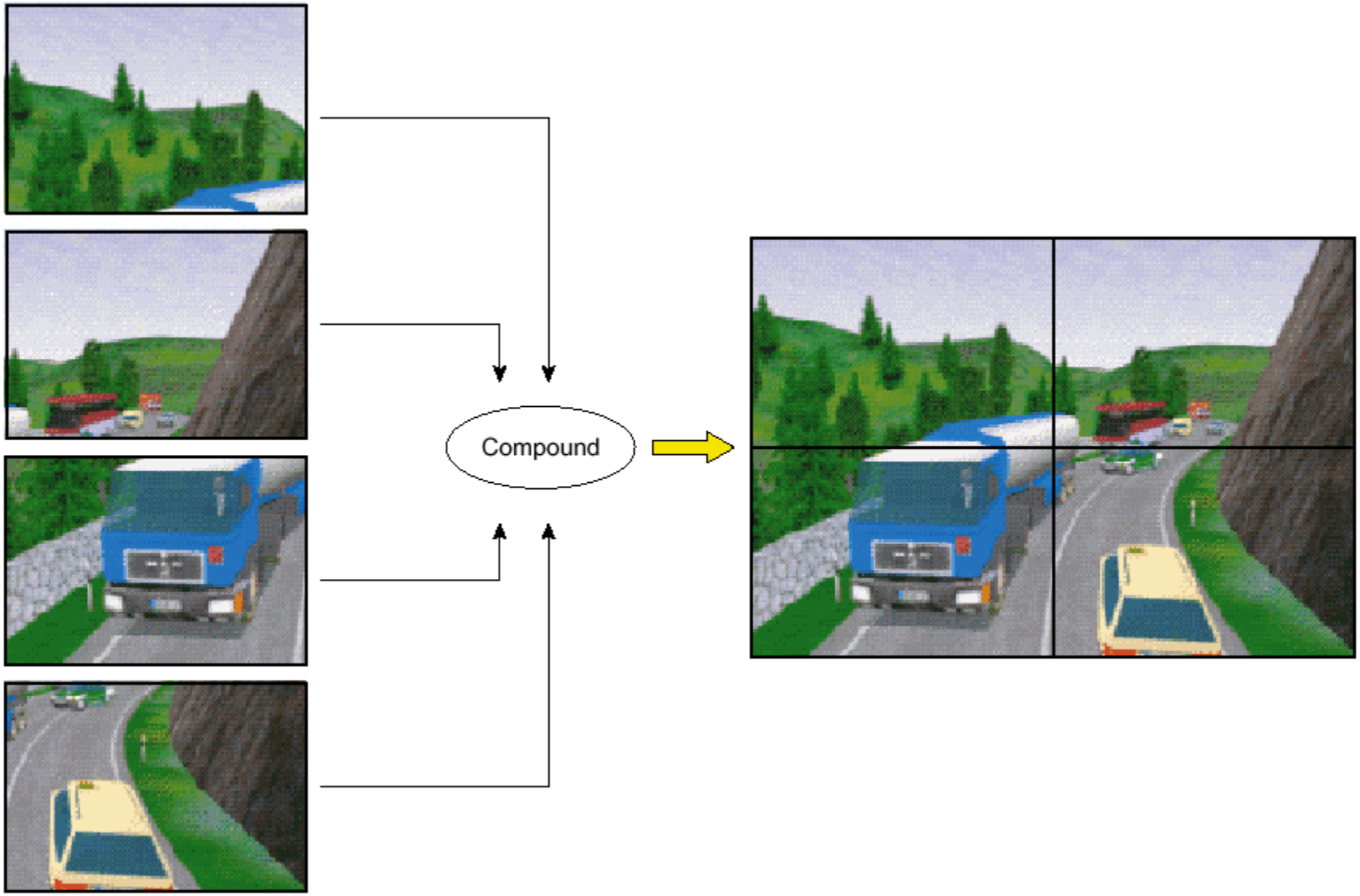
```
config {

   # one or more pipes with windows and channels
   pipe {
   ... ...
   }
   ...

   # compound for the above config
   compound {
        # specify the compound type, format and output channel
        mode [ 2D/DB/DPLEX, HW/NOCOPY, etc ]
        format [ COLOR, DEPTH, etc]
        channel "channel_1“
        # specify one or more source channels and their params
        region {
            ... ...
        }
        ...
     }
}
```

Compounds

- Commonly used Modes
  - 2D (screen tiling)
  - DB (database decomposition)
  - DPLEX (time-slice multiplexing)
  - EYE (stereo decomposition)
  - Others...

2D Compound

Each pipe renders a different viewport

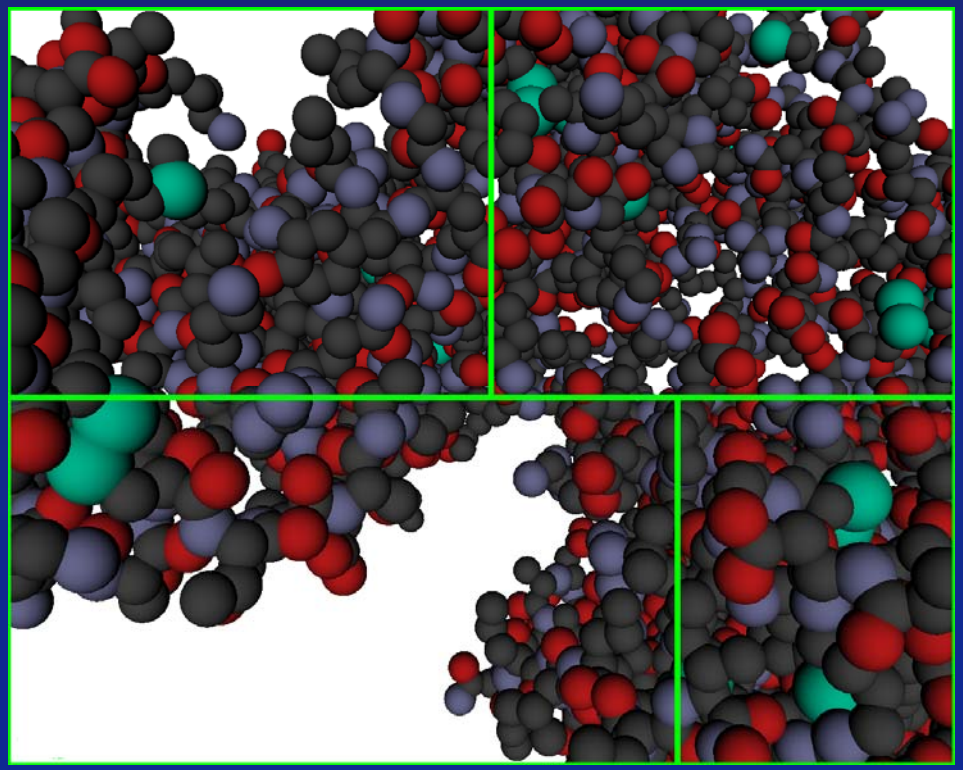


2D Compound

```
compound {
        mode    [ 2D ]
        format [ COLOR ]
        channel "channel_1"
        region {
              viewport [ 0., .0, 1.0, 0.5 ]
              channel “channel_1"
         }
          region {
              viewport [ 0., .5, 1.0, 0.5 ]
              channel "channel_2"
         }
}
```

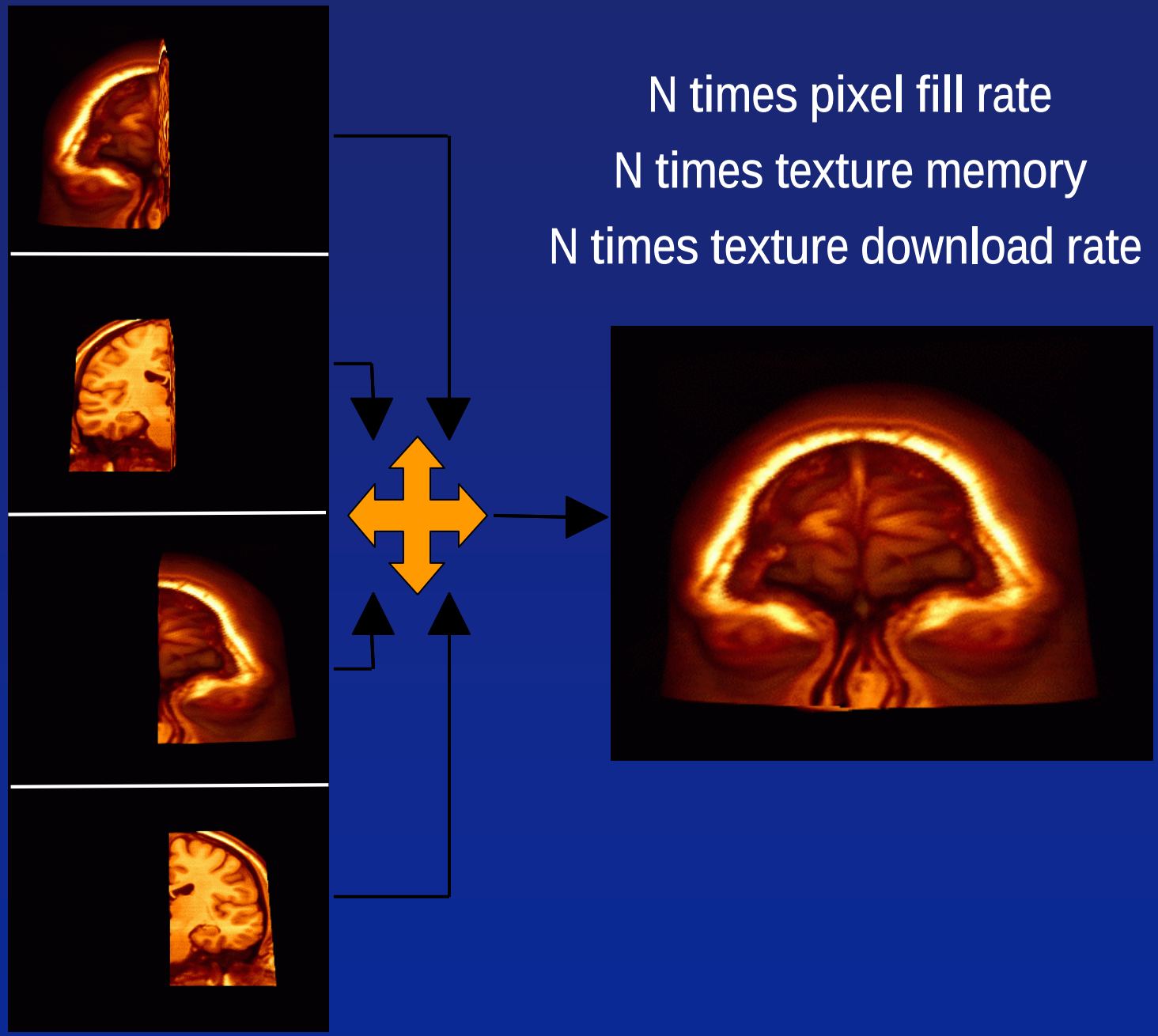
2D Compound

- Automatic Load balancing
  - Based on timing values from last frame
  - Good results for low-latency decompositions



DB Compound

Each pipe renders a different part of the data set



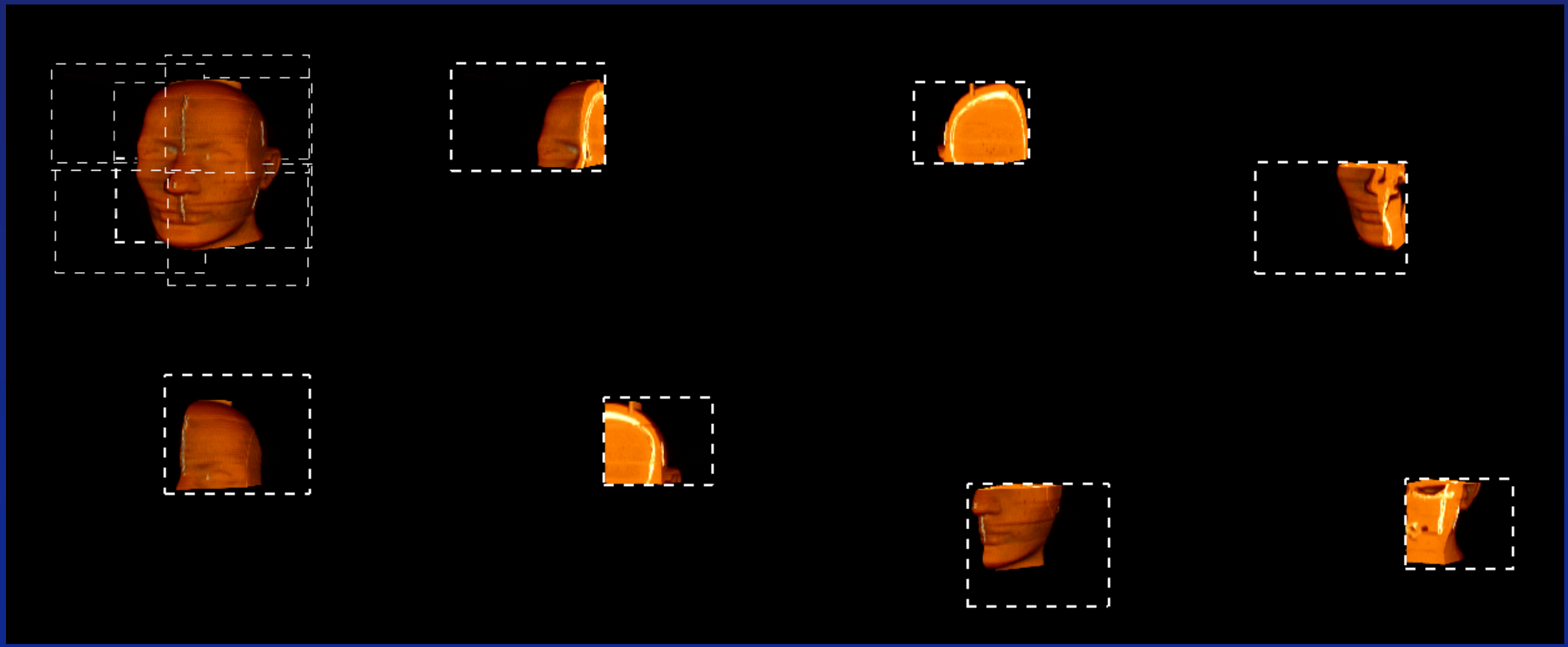
N times pixel fill rate

N times texture memory

N times texture download rate

DB Compound

Compositing order changes with view
Adaptive Readback comes in handy

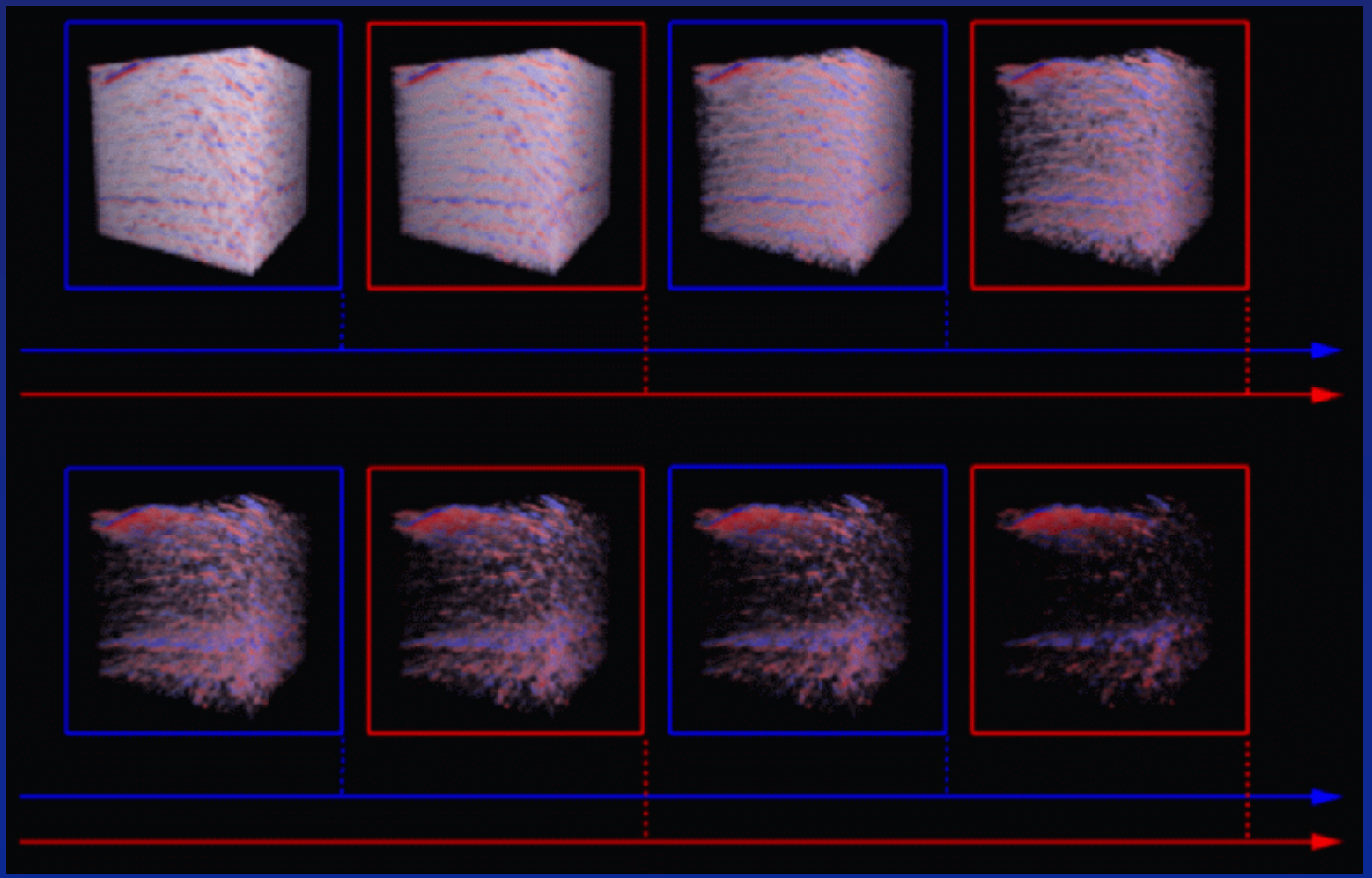


DB Compound

```
compound {
        mode    [ DB ]
        format [ COLOR DEPTH ]
        channel "channel_1"
        region {
              range [ 0., 0.5 ]
              channel “channel_1"
         }
          region {
              range [ 0.5, 1 ]
              channel "channel_2"
          }
}
```

DPLEX Compound

Each pipe renders a different frame

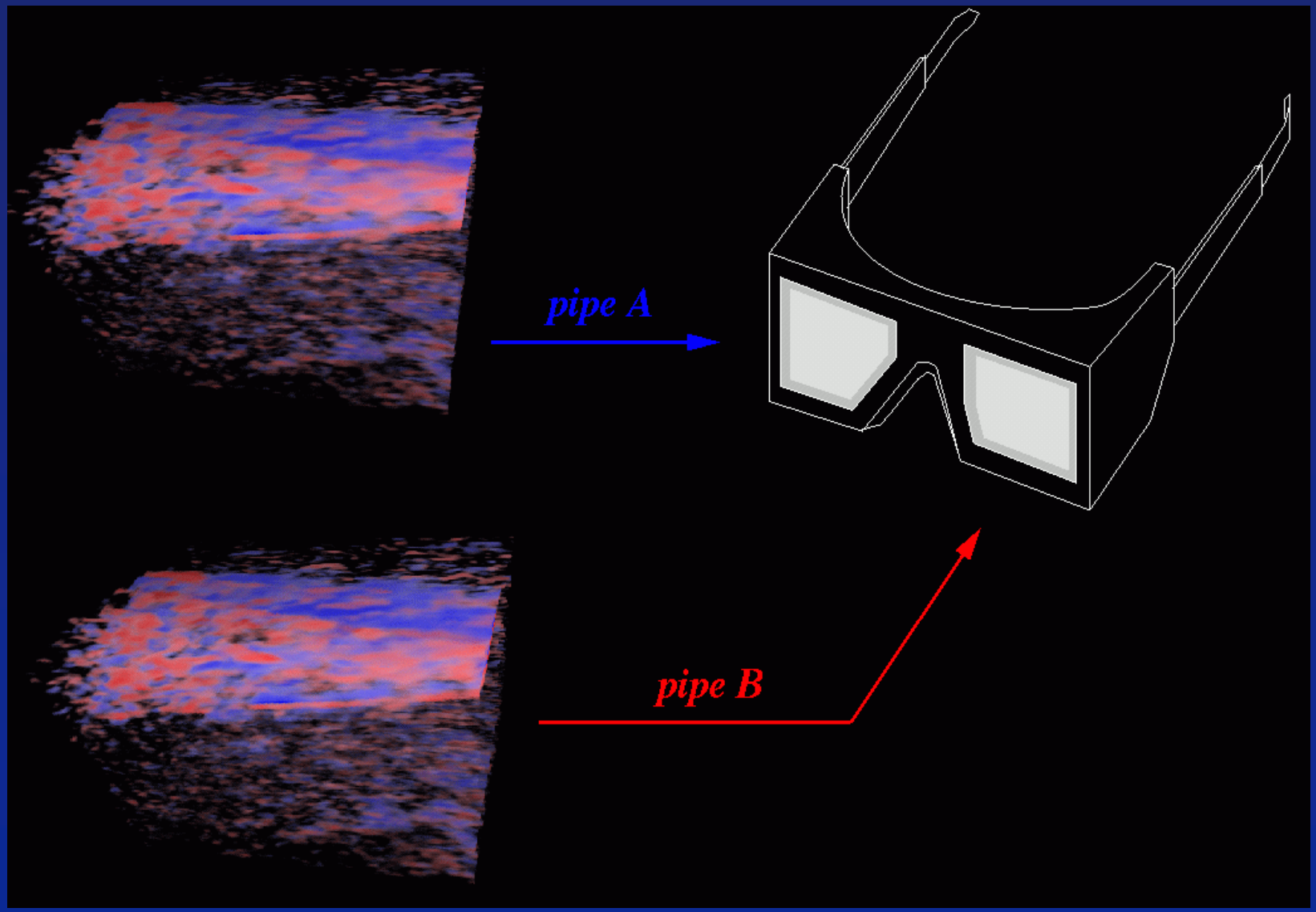


DPLEX Compound

```
compound {
        mode    [ DPLEX ]
        format [ COLOR ]
        channel "channel_1"
        region {
              channel “channel_1"
         }
          region {
              channel "channel_2"
          }
}
```

EYE Compound

Each pipe renders for a different eye position



Compound Configs

- Choosing the right decomposition mode

Mode	Geometry Processing	Pixel Fill	Bandwidth to graphics	Graphics Memory	Application Transparent
2D	Y/N	Y	N	N	Y/N
DPLEX	Y	Y	Y	N	Y
DB	Y	Y	Y	Y	N

Hierarchical Compounds


```
compound {
        mode    [ DPLEX ASYNC ]
        channel "channel"
        region {
            compound {
                mode    [ DPLEX ASYNC ]
                channel "dplex::1"
                region { channel "dplex:a:1" }
                ... ...
        } }
        region {
            compound {
                mode    [ DPLEX ASYNC ]
                channel "dplex::2"
                region { channel "dplex:b:1" }
                ... ...
         } }
        region {
           ... ...
        } }
```

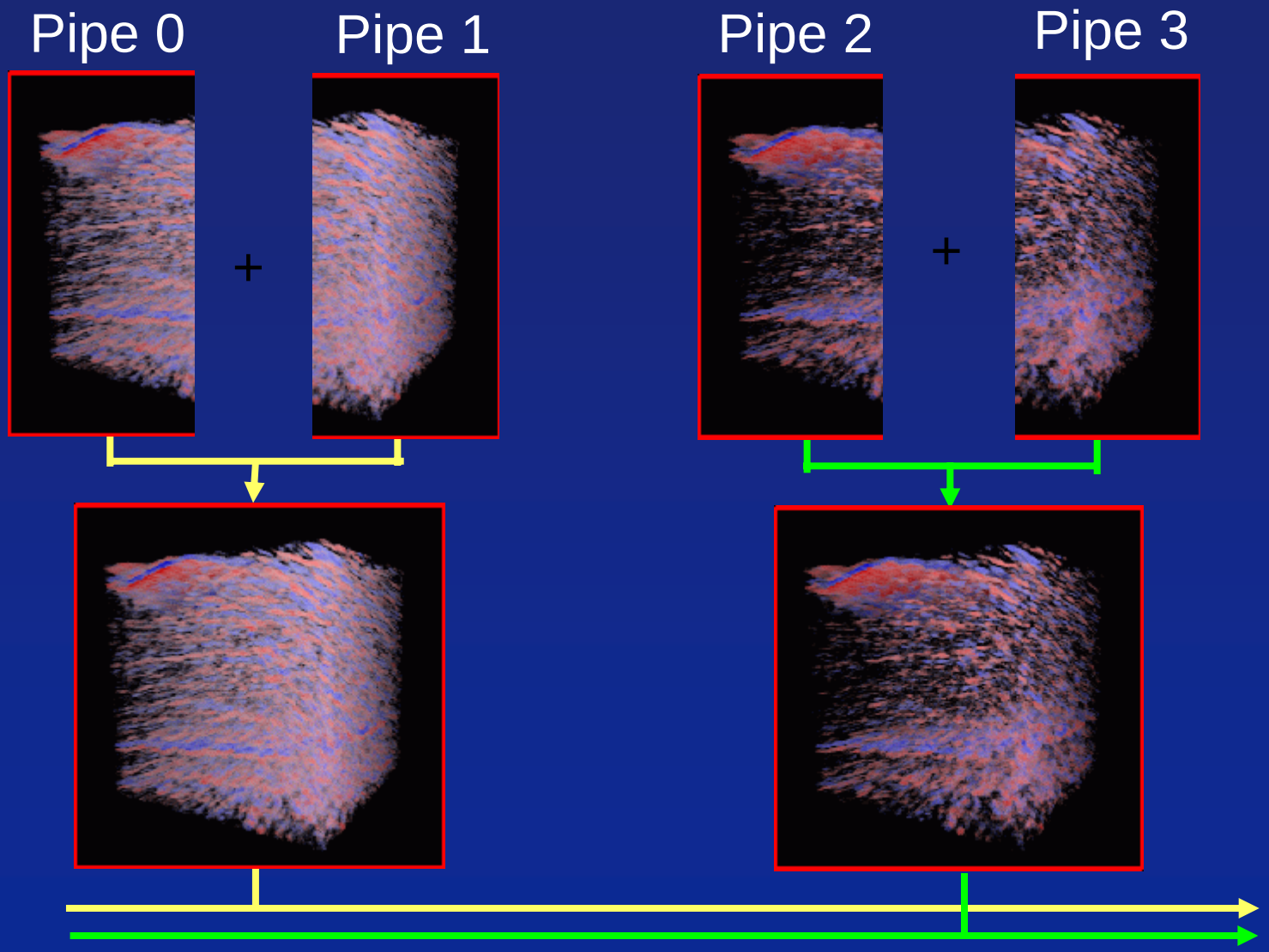
9 => 3 => 1 windows

Hardware Composition

```
compound {
        mode    [ 2D HW NOCOPY ]
        channel "channel0"
        region {
            viewport    [ 0., .75, 1., .25 ]
            channel "channel0"
        }
       ... ...
        region {
            viewport    [ 0., .25, 1., .25 ]
            channel "channel2"
        }
}
```

Hybrid Compounds

- Combined DPLEX - 2D



Related Projects

- Related Projects
  - Chromium

    http://chromium.sourceforge.net
  - OpenGL Multipipe (OMP)
    http://www.sgi.com/software/multipipe
  - CAVELib
  - VRJuggler

Scalability Approaches

- Aware Applications
  - Effort required to port the app to run in MP environment (MPK, PF)
  - Good scalability with app work
  - Immersive environments easier to handle
- Unaware Applications
  - Effort goes into the intercept-dispatch library (CR, OMP)
  - Good/Limited scalability depending on the app
  - Immersive environments not so easy

MPK vs OMP

- Multipipe SDK
  - API for writing MP apps
  - App scales fill, geometry, memory and display
- OpenGL Multipipe
  - Transparent app layer
  - Scales display size, fill, well. Limited geometry, texture, scaling.

Recap: Features

- Ease of Integration
  - fork, sproc, pthread support
  - Event-driven execution model
  - Adaptive readback interface
  - App-created windows support
  - Non-threaded windows support
  - Xinerama integration
  - Custom compositing interface
- Runtime Portability
  - ASCII File Format specification
  - Multi-frustum support
  - Dynamic parallel rendering
- Runtime Scalability
  - Compound class specification
  - 2D, DB, EYE, DPLEX and FSAA compounds
  - RGBA, Z and STENCIL image compositing
  - Latency / ASYNC decomposition
  - Automatic load-balancing
  - SGI Scalable Graphics Hardware integration
- Stereo / Immersion
  - Off-axis frustum computations
  - Stereo / Head-Tracking support
  - Head Mounted Display [HMD]
  - Mirrored Projection Support

MPK: More Information

- MPK 3.0.1 web release available
- Multipipe SDK product web site
  http://www.sgi.com/software/multipipe/sdk
- Engineering mailing list

  mpsdk@els.sgi.com


sgi
TM

MPK/OMP/Chromium

Multipipe API Comparison				
		Chromium	OMP	MPK
Application-transparent		Yes/No	Yes	No
Open source		Yes	No	No
Supported OSes		Windows, Linux, IRIX, etc.	Linux (IA64), IRIX	Linux (IA64), IRIX
Programming model		application-transparent OpenGL + optional Chromium extension(s)	application-transparent OpenGL	callbacks for frame and data management
Runtime configuration method		python launch scripts, interchangeable modular Stream Processing Units (SPUs)	command line flags, environment variables	MPK config file
Architecture		multiprocess	multiprocess	multithreaded/fork
	Node structure	arbitrary directed graph of SPUs (many masters, many slaves)	one master (app), one or more cullers, many slaves	one master, many slaves (optional culler per slave)
Codec		WireGL-like	GLScodec	N/A
Transport among processes		tcp, miranet, MPI, IB?	shared memory (queue) - data must be copied into shm	shared memory (arena) - can pass pointers to data residing in shm
Decomposition modes				
	Sort-first (Tilesort)	Yes	Yes	Yes
	Sort-last (Depth/Alpha)	Yes	No	Yes
	Sort-last w/ HW readback	No	No	No
	Timeslice	??	No	Yes
	Stereo	Yes	No	Yes
Culling (tilesort)				
	BBox computation	Master node	Master node	Application
	BBox transformation	Master	Culler	Application
	GL state management	Master	Slave	Application
Number of transport channels		one per edge in the graph	one per rendering application thread	one per MPK application
Readback/glGet		Not fully supported?	Master queries slaves or local pipe	Native GL

MPK: Configuration