Taking Advantage of Multiple GPUs

Kenneth Dyke
Sr. Engineer, Graphics and Compute Architecture

What You'll Learn

- Supporting multiple GPUs in your application
  - Finding all renderers and devices
  - Responding to renderer changes
- Making use of multiple GPUs at the same time
  - Shared contexts
  - Resource management and synchronization
  - Performance tips
- Using IOSurface with multiple GPUs

Supporting Multiple GPUs in Your App

Motivation

- Systems with multiple GPUs are shipping today!
  - Mac Pros can have up to four GPUs
  - Some recent MacBook Pros also have multiple GPUs
- Better user experience on systems with multiple GPUs
  - Increased performance
  - Hot-plug support

Basic Multi-GPU Support

Basic Multi-GPU Support in OpenGL

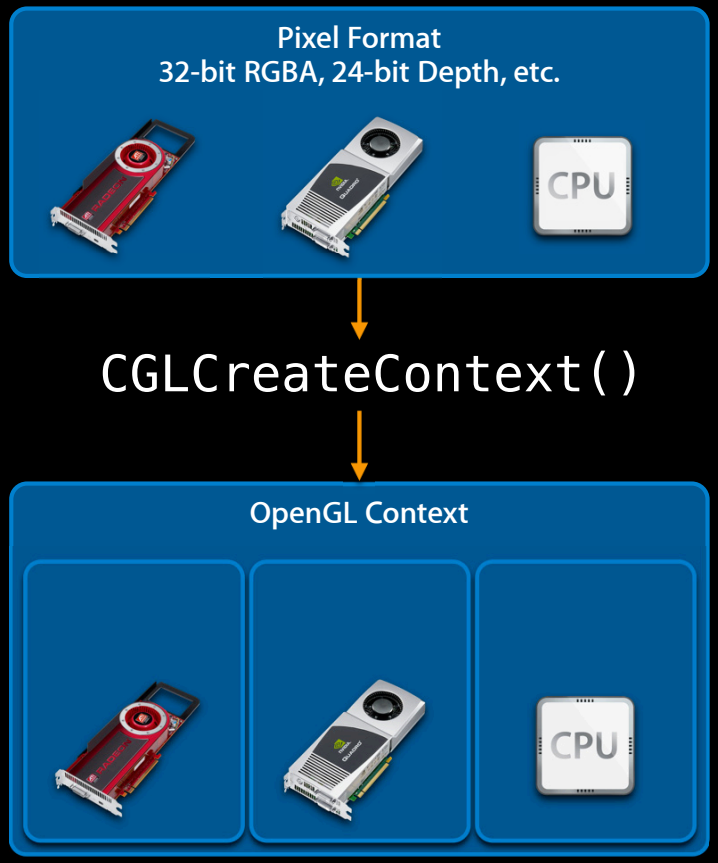
Renderers and pixel formats

- A *renderer* represents a single GPU or the CPU-based software rasterizer
- Each renderer has a unique *renderer id*
- A *pixel format* represents both drawable attributes *and* a specific set of renderers

Basic Multi-GPU Support in OpenGL

Pixel formats and contexts

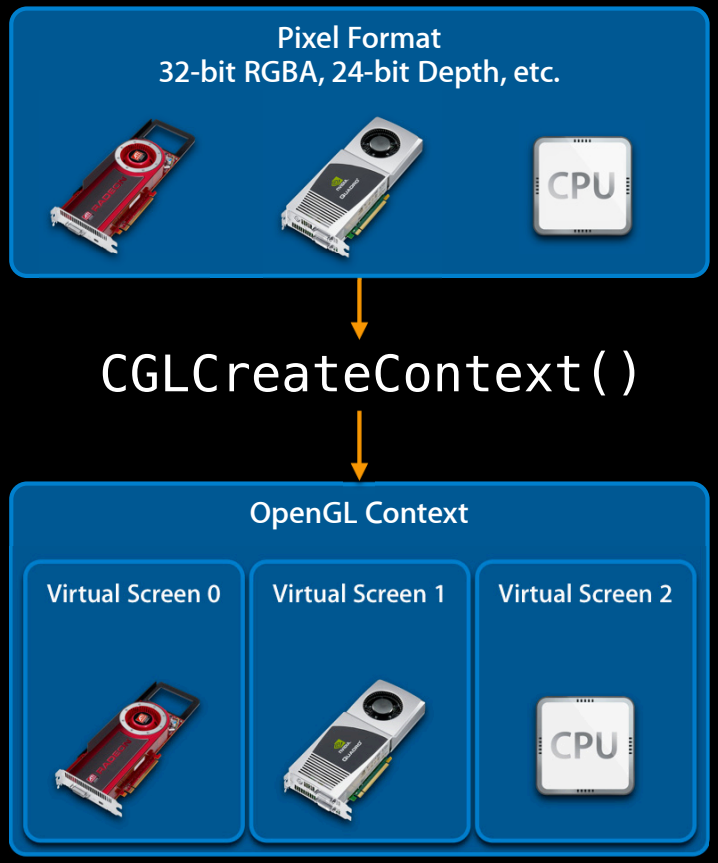
- OpenGL contexts inherit the list of renderers from the pixel format
- Each renderer within an OpenGL context is assigned a *virtual screen*



Basic Multi-GPU Support in OpenGL

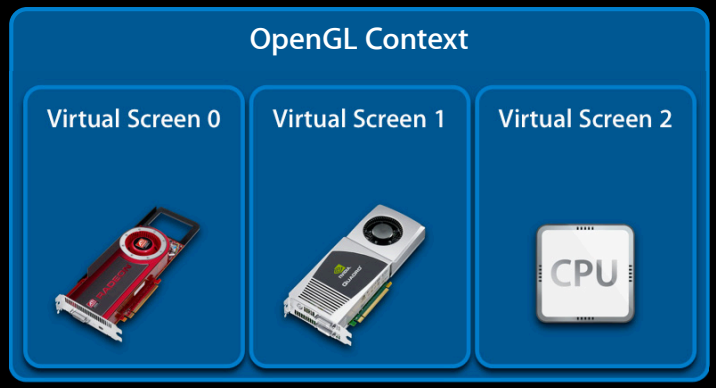
Pixel formats and contexts

- OpenGL contexts inherit the list of renderers from the pixel format
- Each renderer within an OpenGL context is assigned a *virtual screen*



Basic Multi-GPU Support in OpenGL

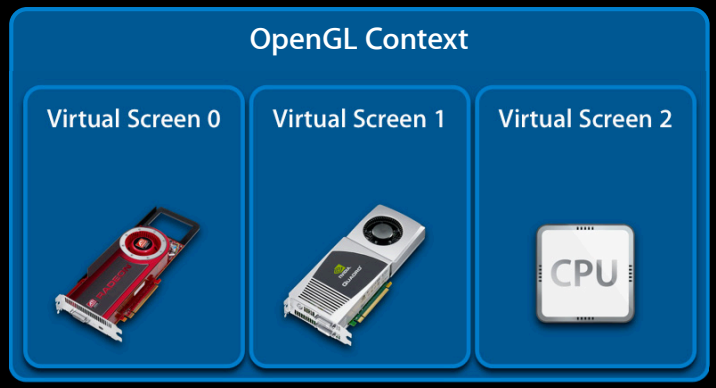
Virtual screens and renderers



Basic Multi-GPU Support in OpenGL

Virtual screens and renderers

- The *current* virtual screen is used to select the context's renderer
- Virtual screen order matches pixel format
- No correlation with physical screens
- Multiple heterogenous renderers is unique to Mac OS X!



Basic Multi-GPU Support in OpenCL

Different APIs, but similar concepts

- `clGetDeviceIDs()` instead of CGLChoosePixelFormat()
- `clCreateContext()` passed list of devices instead of pixel format
- `cl_command_queue` selects the device for a command

Basic Multi-GPU Support

Allowing for multiple renders

- Don't use NSOpenGLPFAScreenMask
  - Guaranteed you'll only get a single hardware renderer
  - Only necessary for legacy full-screen contexts

```
displayMask = CGDisplayIDToOpenGLDisplayMask(kCGDirectMainDisplay);
NSOpenGLPixelFormatAttribute attribs[] =
{
   NSOpenGLPFAAccelerated,
   NSOpenGLPFADoubleBuffer,
   NSOpenGLPFAColorSize, 32,
   NSOpenGLPFAScreenMask, displayMask,
   0
};
```

Basic Multi-GPU Support

Allowing for multiple renders

- Don't use NSOpenGLPFAScreenMask
  - Guaranteed you'll only get a single hardware renderer
  - Only necessary for legacy full-screen contexts

```
NSOpenGLPixelFormatAttribute attribs[] =
{
   NSOpenGLPFAAccelerated,
   NSOpenGLPFADoubleBuffer,
   NSOpenGLPFAColorSize, 32,

   0
};
```

Basic Multi-GPU Support

Allowing for multiple renders

- Do use NSOpenGLPFAAllowOfflineRenderers

```
NSOpenGLPixelFormatAttribute attribs[] =
{
   NSOpenGLPFAAccelerated,
   NSOpenGLPFADoubleBuffer,
   NSOpenGLPFAColorSize, 32,

   0
};
```

Basic Multi-GPU Support

Allowing for multiple renders

- Do use NSOpenGLPFAAllowOfflineRenderers
- This gets you all renderers, even ones without a display
- Important for hot plug

```
NSOpenGLPixelFormatAttribute attribs[] =
{
   NSOpenGLPFAAccelerated,
   NSOpenGLPFADoubleBuffer,
   NSOpenGLPFAColorSize, 32,
   NSOpenGLPFAAllowOfflineRenderers,
   0
};
```

Basic Multi-GPU Support

Triggering renderer changes

- -[NSOpenGLContext update]
  - Best renderer is chosen automatically
  - Usually in response to -[NSOpenGLView update]
  - Or, NSGlobalViewFrameDidChange
- [NSOpenGLContext setVirtualScreen:]
  - Forces a specific renderer
  - Useful for offscreen contexts

```
@implementation MyOpenGLView
...
- (void)update
{
    [[self openGLContext] update];
};
...
@end
```

Basic Multi-GPU Support

Responding to renderer changes

- For some apps, nothing is required
- Otherwise, check for virtual screen changes

```
@implementation MyOpenGLView
...
– (void)update
{
    [[self openGLContext] update];
    /* w00t! I’m done! */
};
...
@end
```

Basic Multi-GPU Support

Responding to renderer changes

- For some apps, nothing is required
- Otherwise, check for virtual screen changes

```
@implementation MyOpenGLView
...
- (void)update
{
    [[self openGLContext] update];
    if(virtualScreen !=
     [[self openGLContext]
       currentVirtualScreen])
       [self gpuChanged];
};
...
```

Basic Multi-GPU Support

Responding to renderer changes

- For some apps, nothing is required
- Otherwise, check for virtual screen changes
  - Check for required extensions

```
@implementation MyOpenGLView
...
- (void)gpuChanged
{




};
...
@end
```

Basic Multi-GPU Support

Responding to renderer changes

- For some apps, nothing is required
- Otherwise, check for virtual screen changes
  - Check for required extensions
  - Verify hardware limits

```
@implementation MyOpenGLView
...
- (void)gpuChanged
{
    GLchar *extensions =
      glGetString(GL_EXTENSIONS);



};
...
@end
```

Basic Multi-GPU Support

Responding to renderer changes

- For some apps, nothing is required
- Otherwise, check for virtual screen changes
  - Check for required extensions
  - Verify hardware limits
  - Synchronize offscreen contexts

```
@implementation MyOpenGLView
...
- (void)gpuChanged
{
    GLchar *extensions =
      glGetString(GL_EXTENSIONS);
    GLsizei maxTextureSize =
      glGetIntegerv(GL_MAX_...);

};
...
@end
```

Basic Multi-GPU Support

Responding to renderer changes

- For some apps, nothing is required
- Otherwise, check for virtual screen changes
  - Check for required extensions
  - Verify hardware limits
  - Synchronize offscreen contexts

```
@implementation MyOpenGLView
...
- (void)gpuChanged
{
    GLchar *extensions =
      glGetString(GL_EXTENSIONS);
    GLsizei maxTextureSize =
      glGetIntegerv(GL_MAX_...);
     [self syncOffscreenContexts];
};
...
@end
```

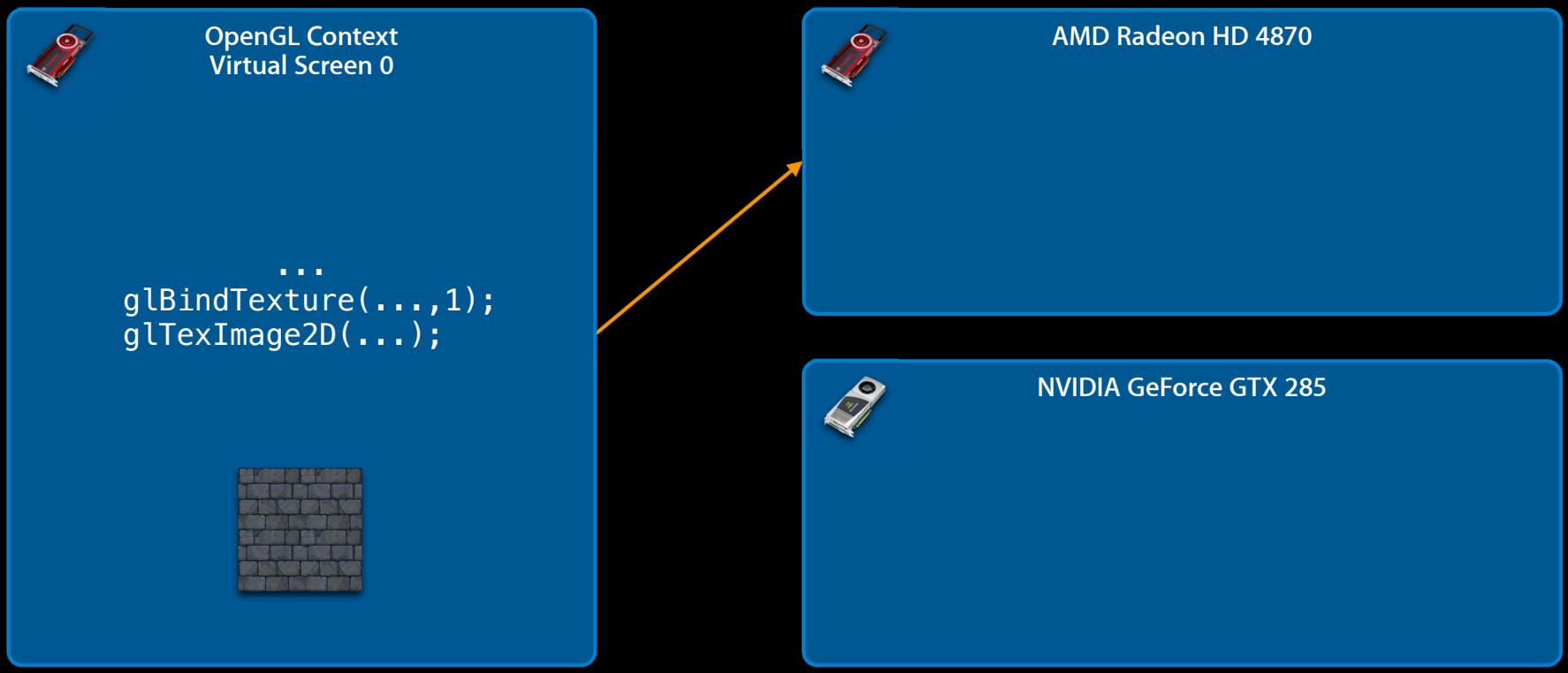
Basic Multi-GPU Support

Resource management during renderer changes

- OpenGL/OpenCL automatically track resources across GPUs
- Unmodified resources will be uploaded as needed
- Modified resources will be copied across via system memory
  - Currently bound objects synchronized at render change
  - Other objects synchronized lazily at bind time

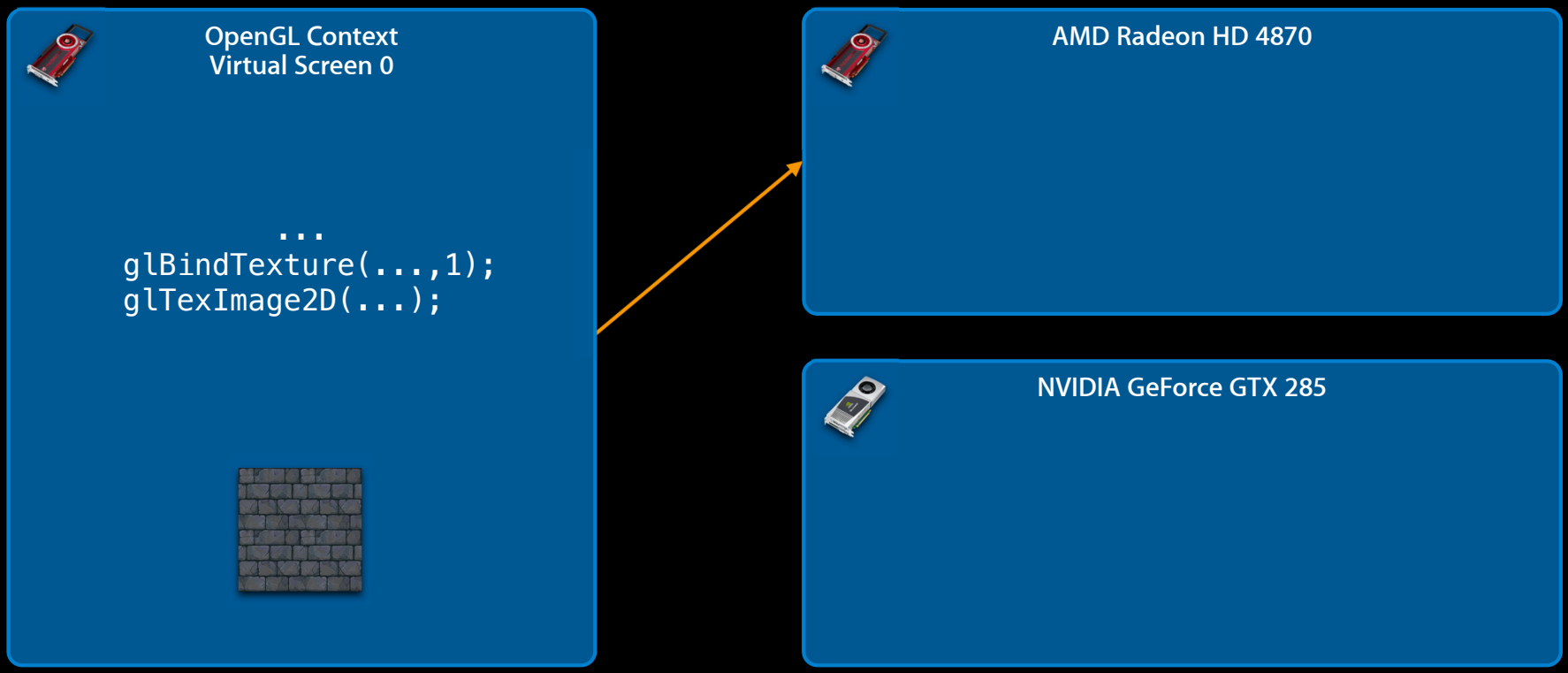
Basic Multi-GPU Support

Resource management during renderer changes



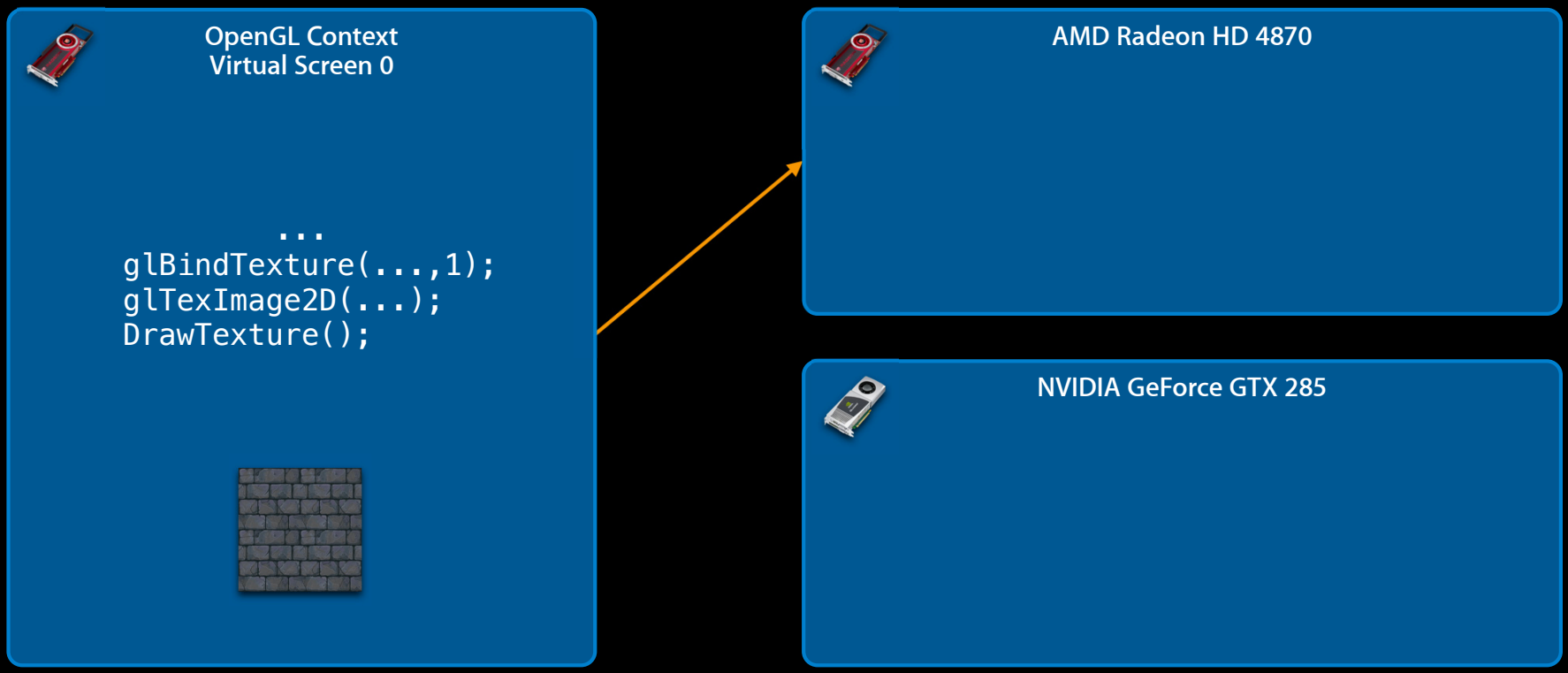
Basic Multi-GPU Support

Resource management during renderer changes



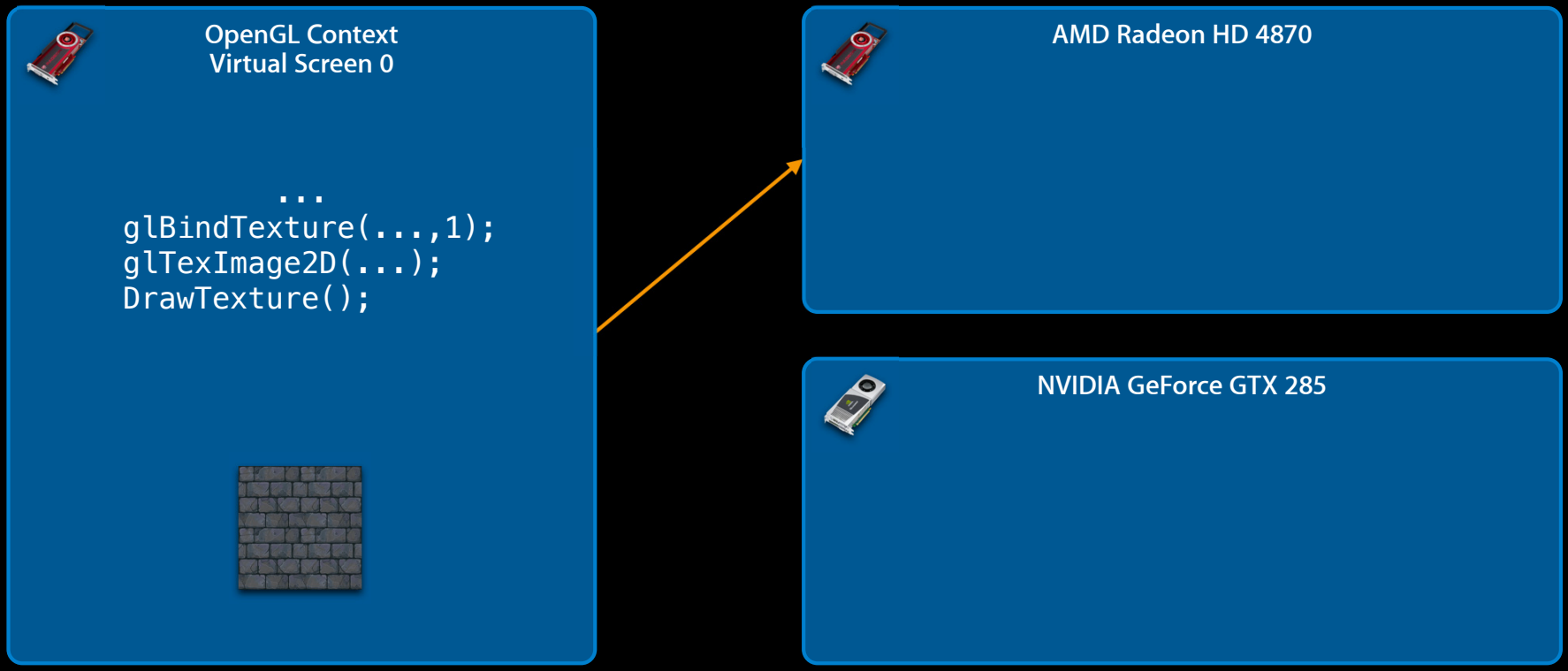
Basic Multi-GPU Support

Resource management during renderer changes



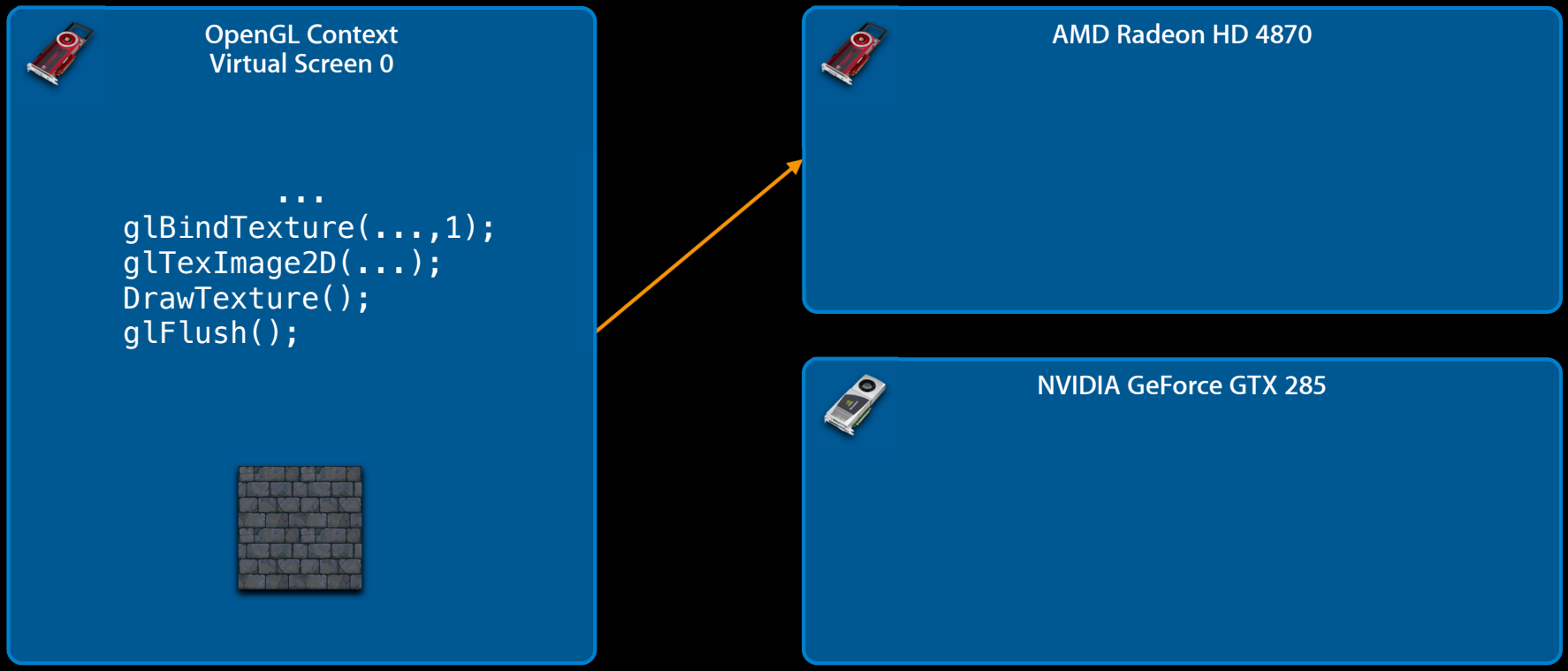
Basic Multi-GPU Support

Resource management during renderer changes



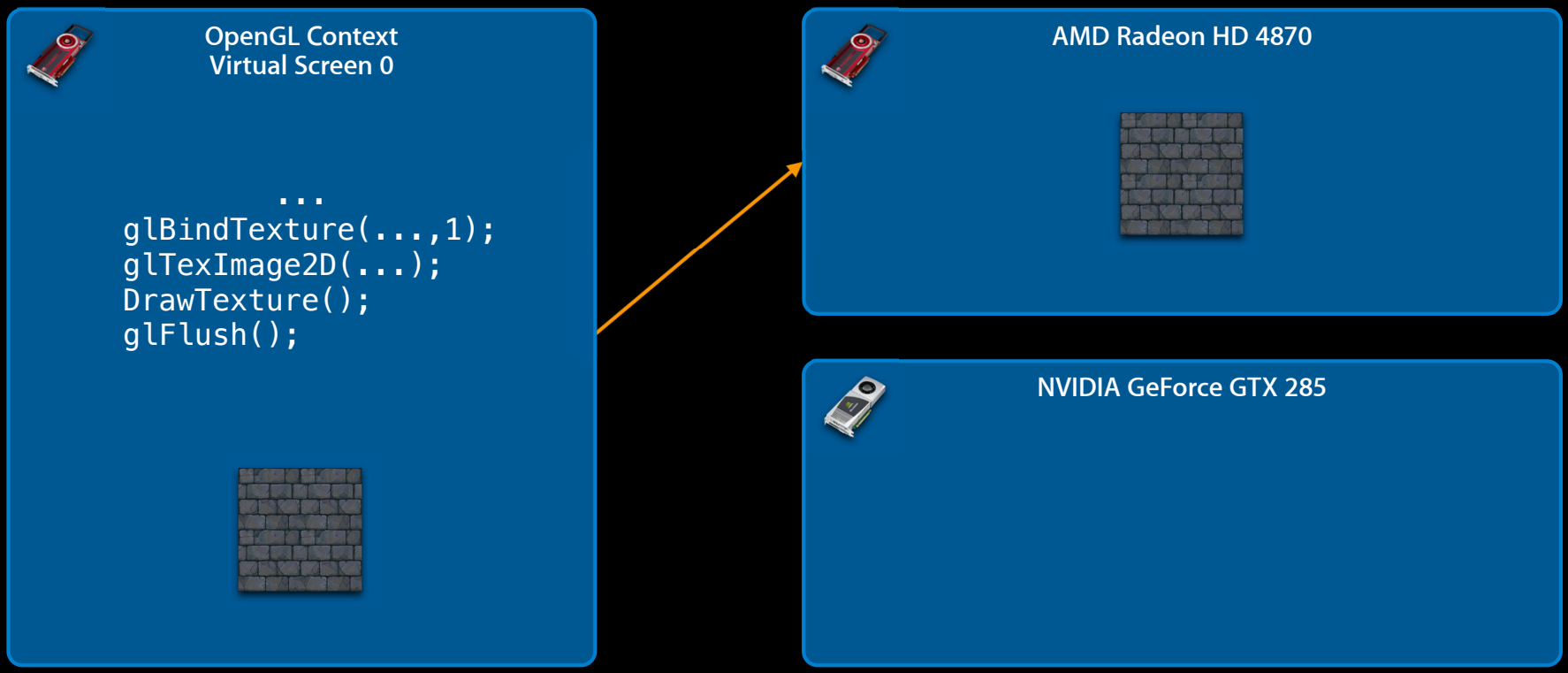
Basic Multi-GPU Support

Resource management during renderer changes



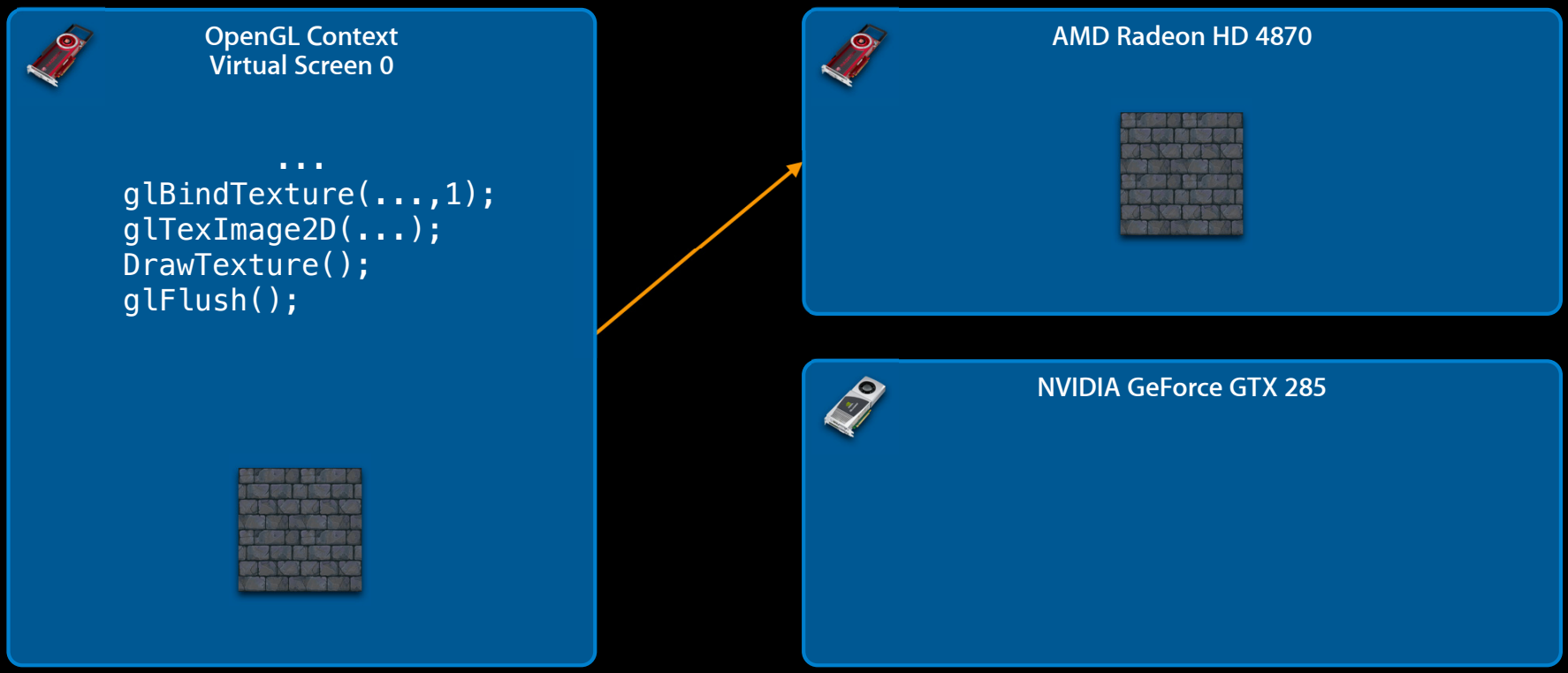
Basic Multi-GPU Support

Resource management during renderer changes



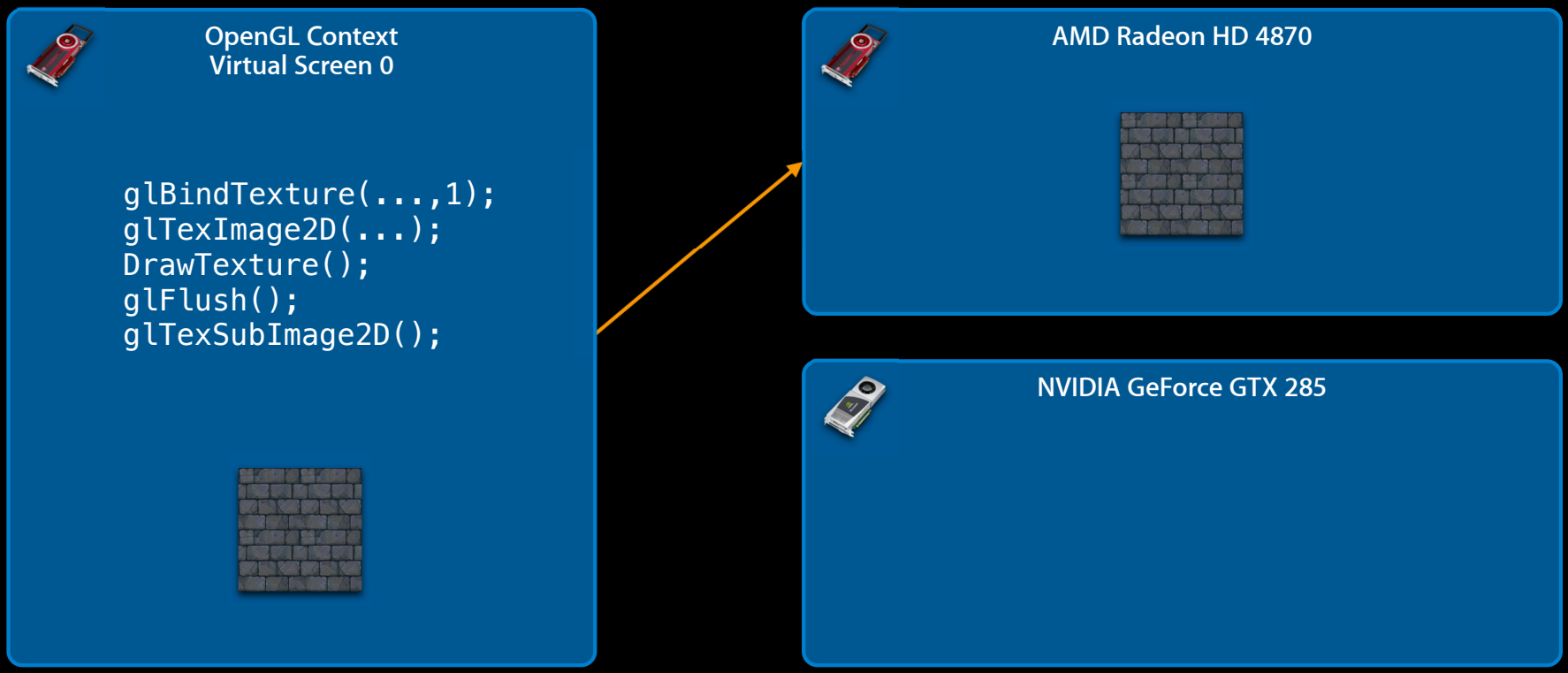
Basic Multi-GPU Support

Resource management during renderer changes



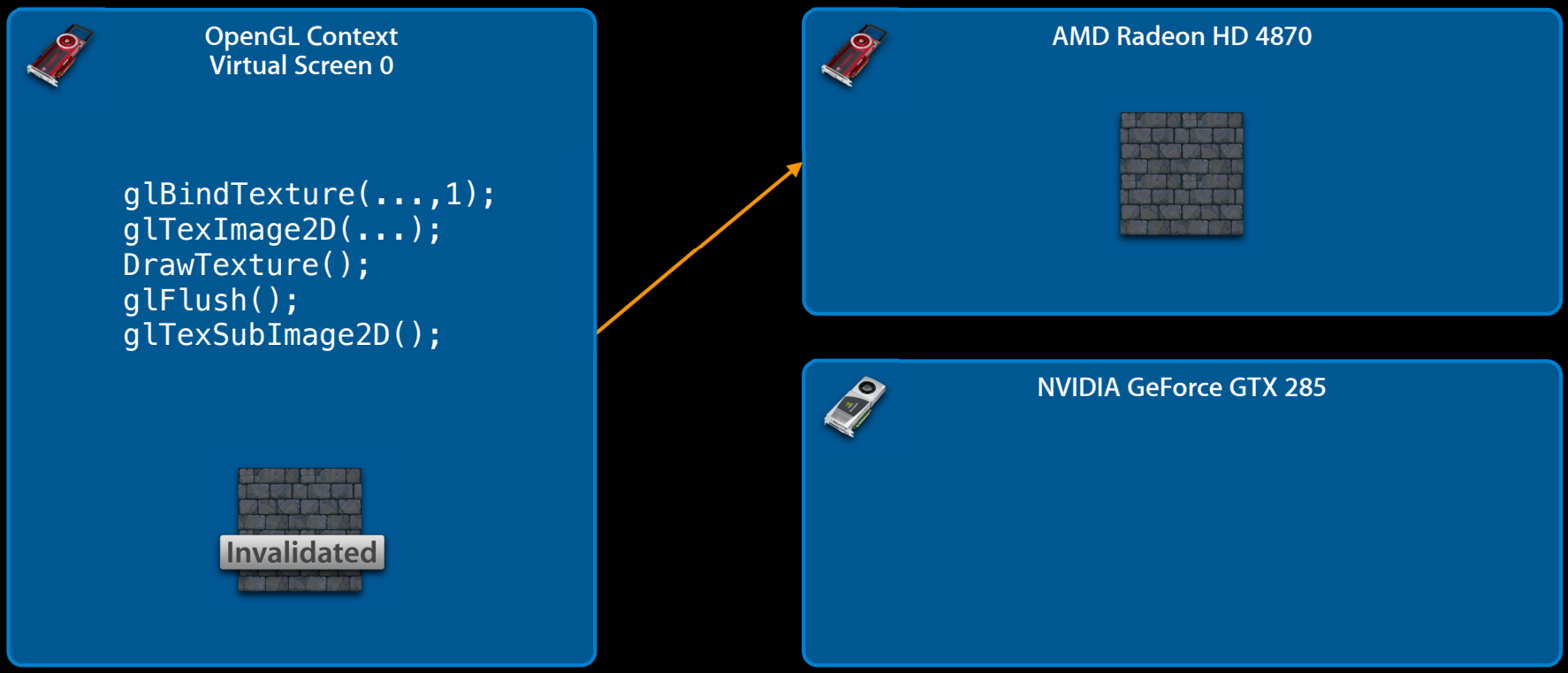
Basic Multi-GPU Support

Resource management during renderer changes



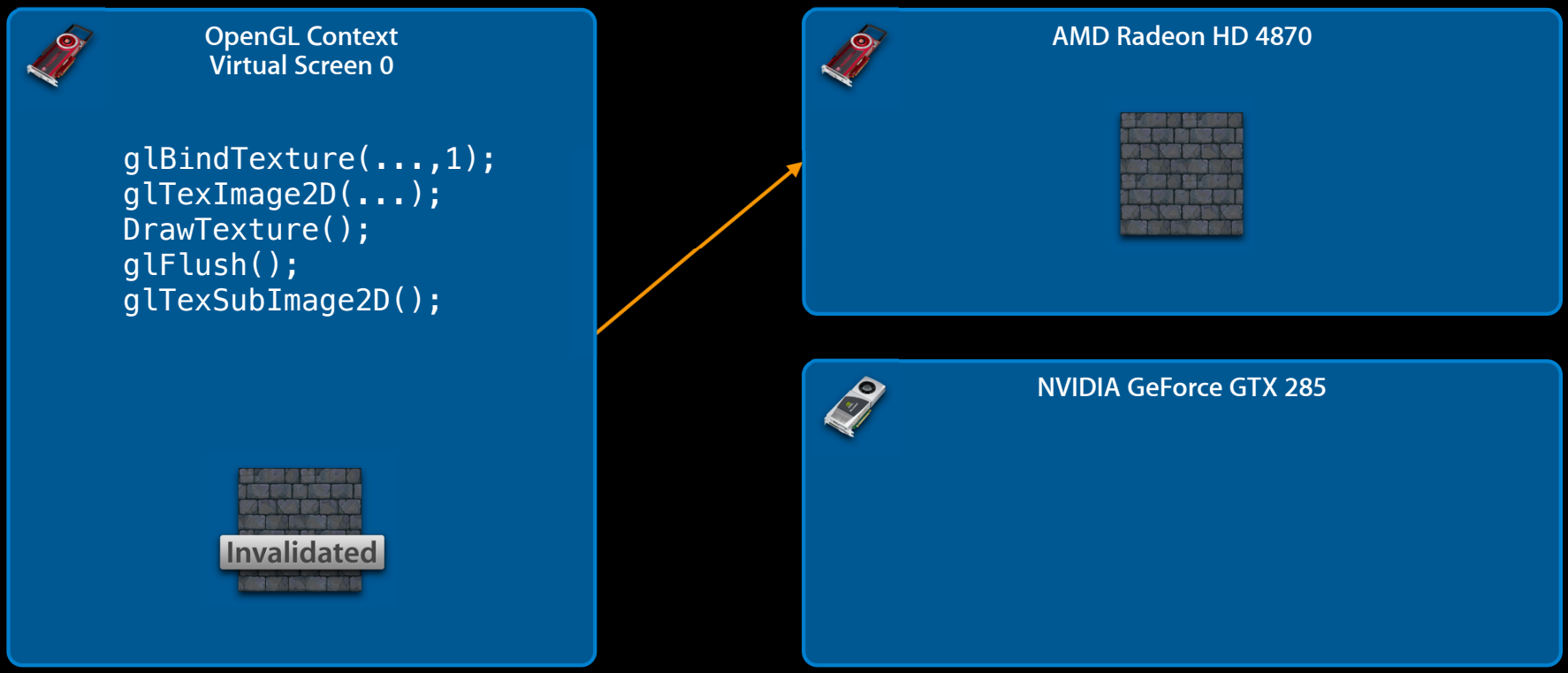
Basic Multi-GPU Support

Resource management during renderer changes



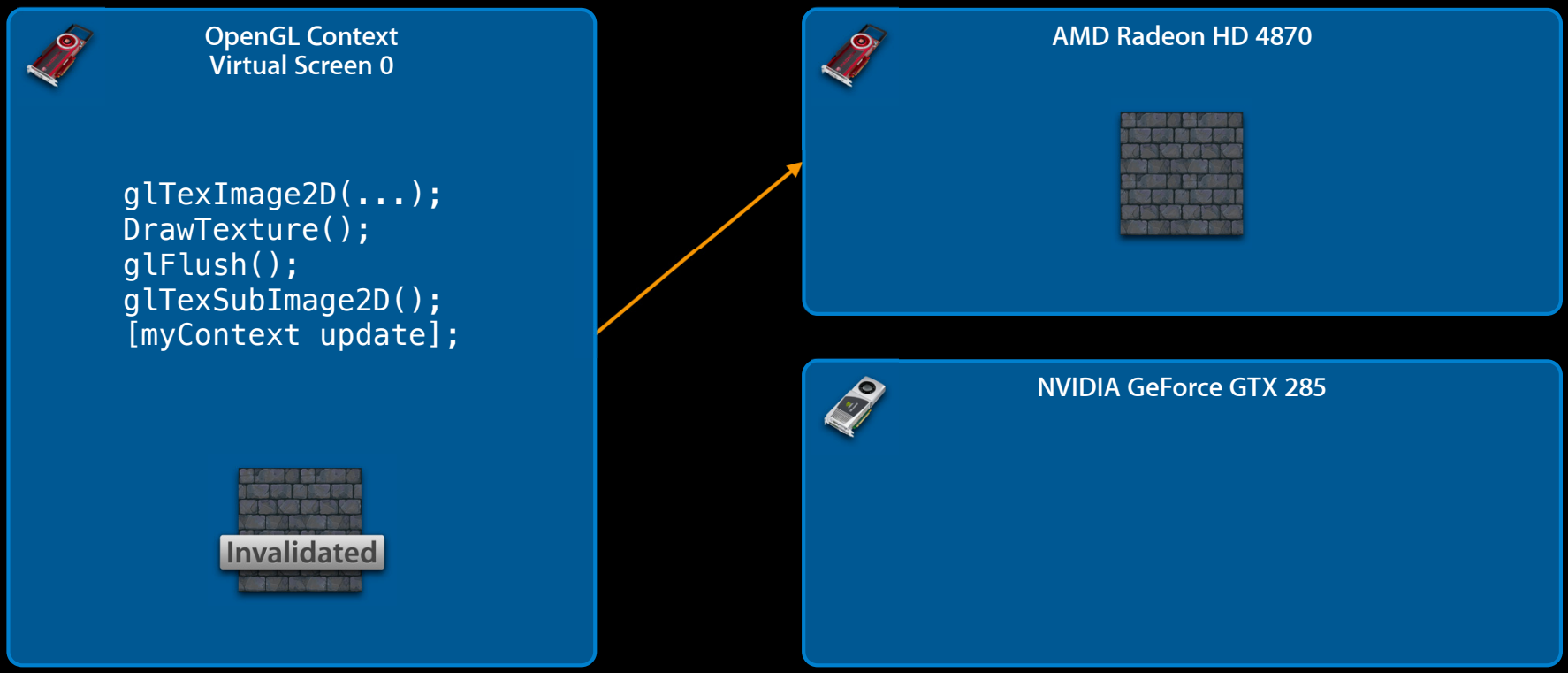
Basic Multi-GPU Support

Resource management during renderer changes



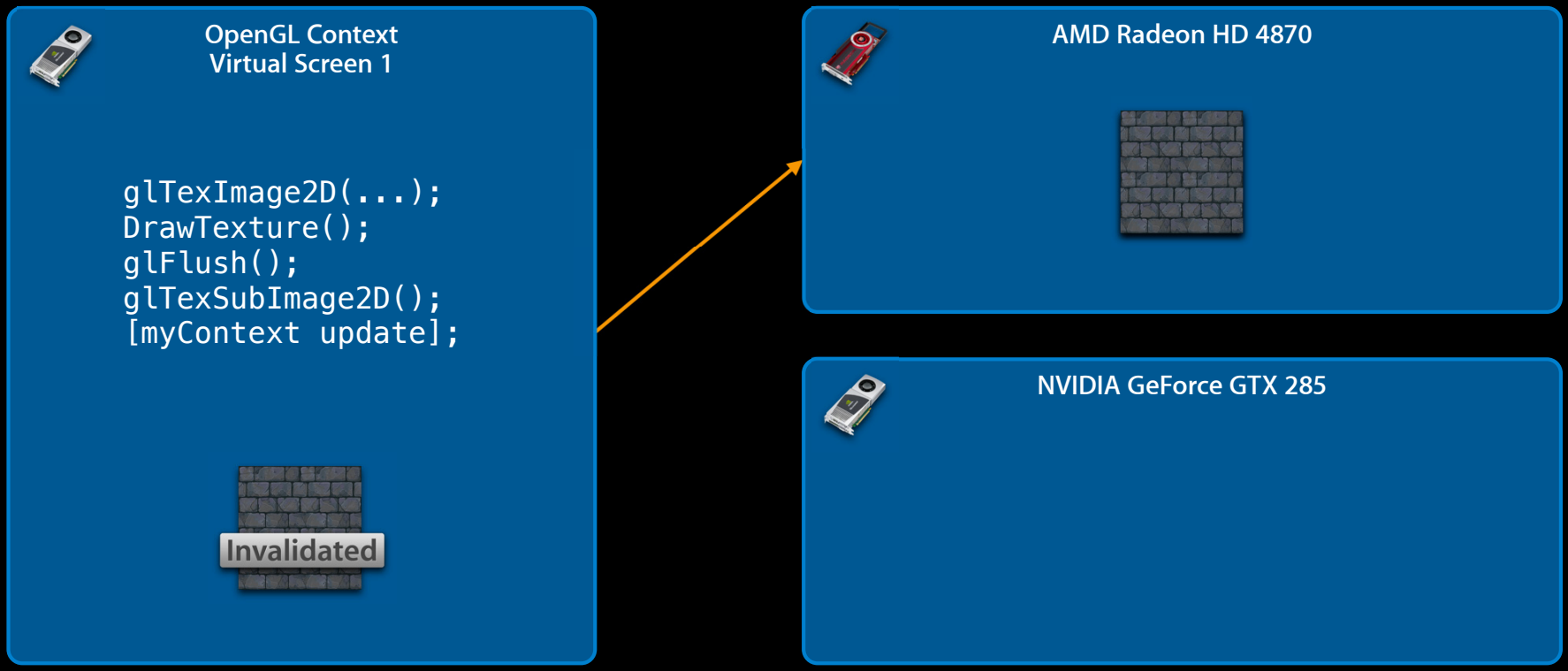
Basic Multi-GPU Support

Resource management during renderer changes



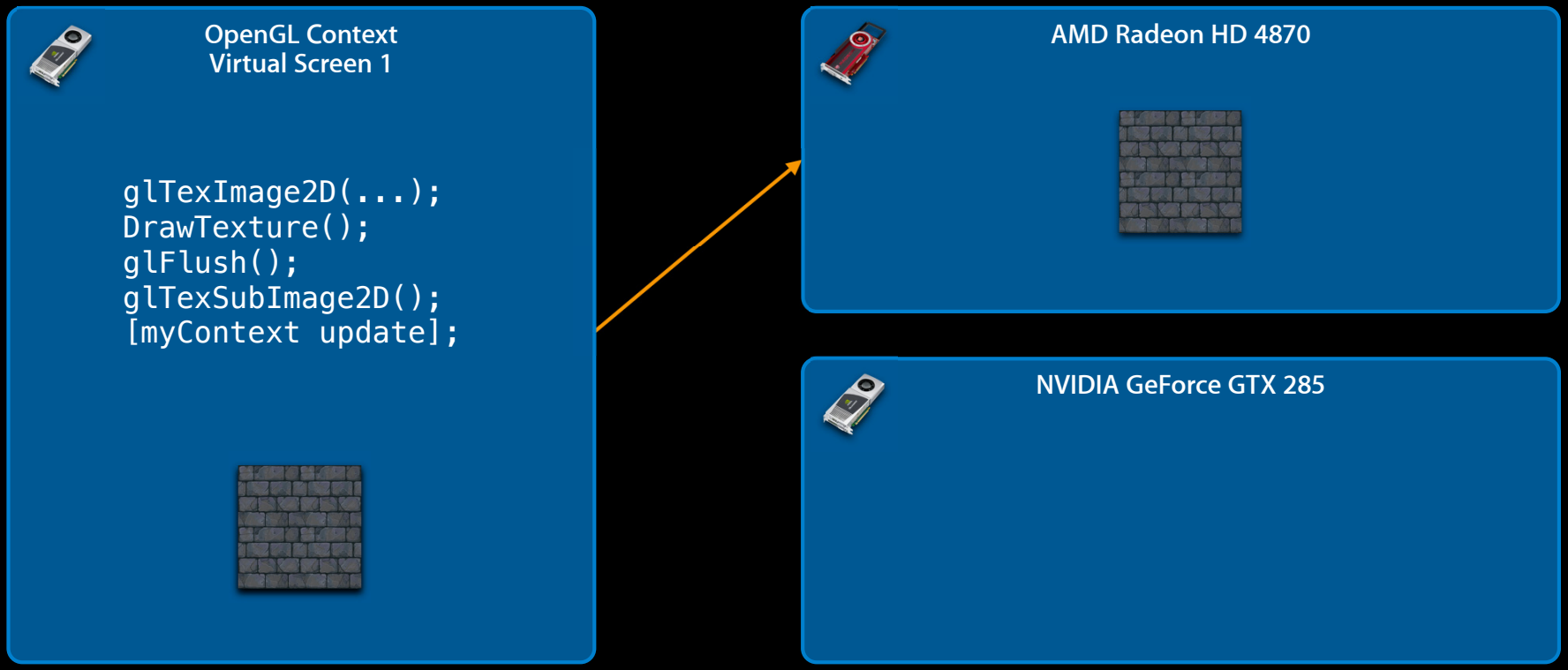
Basic Multi-GPU Support

Resource management during renderer changes



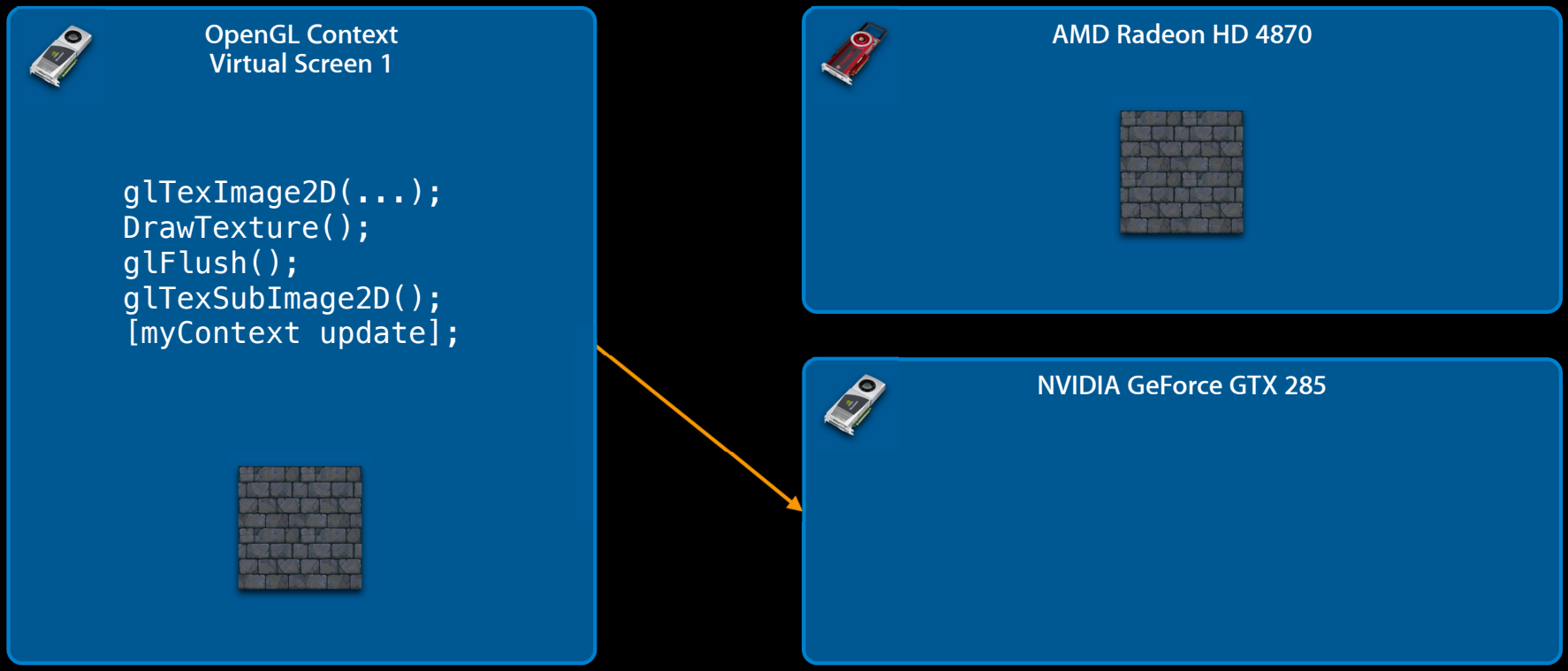
Basic Multi-GPU Support

Resource management during renderer changes



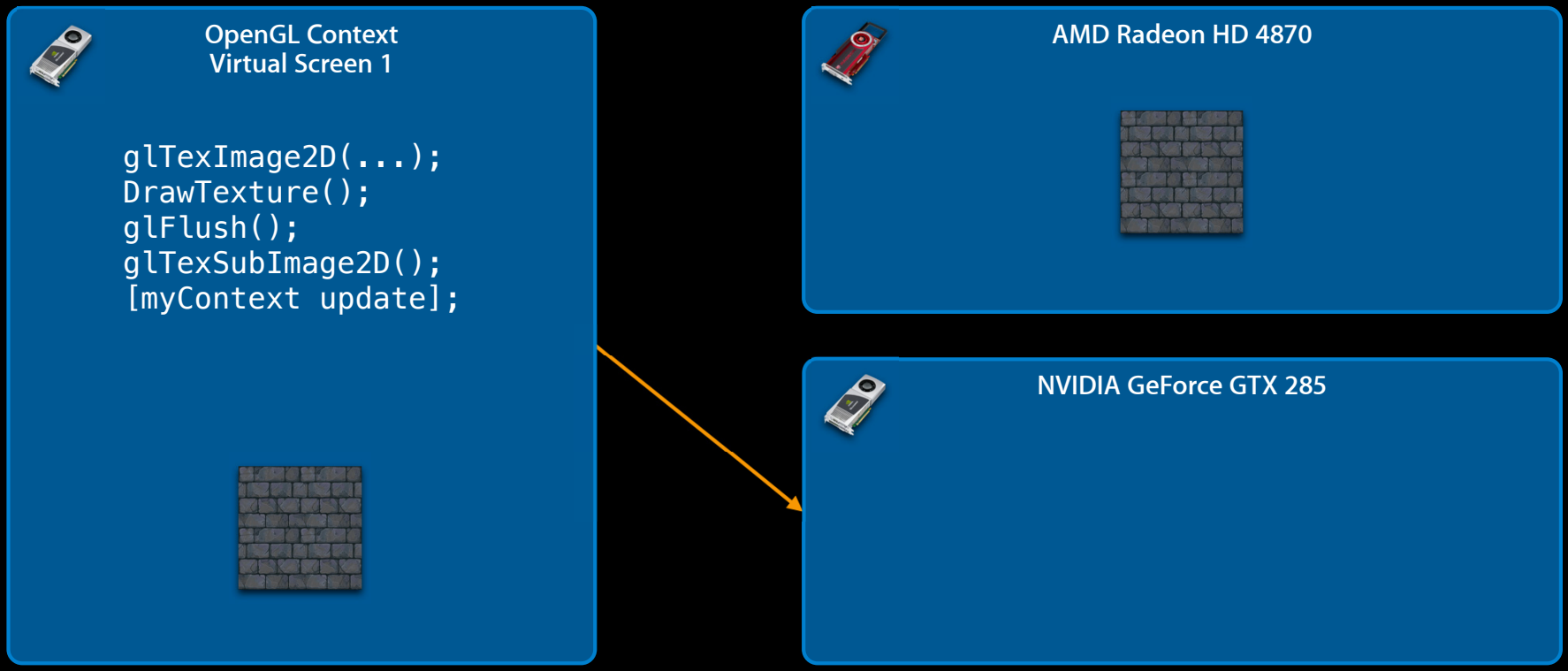
Basic Multi-GPU Support

Resource management during renderer changes



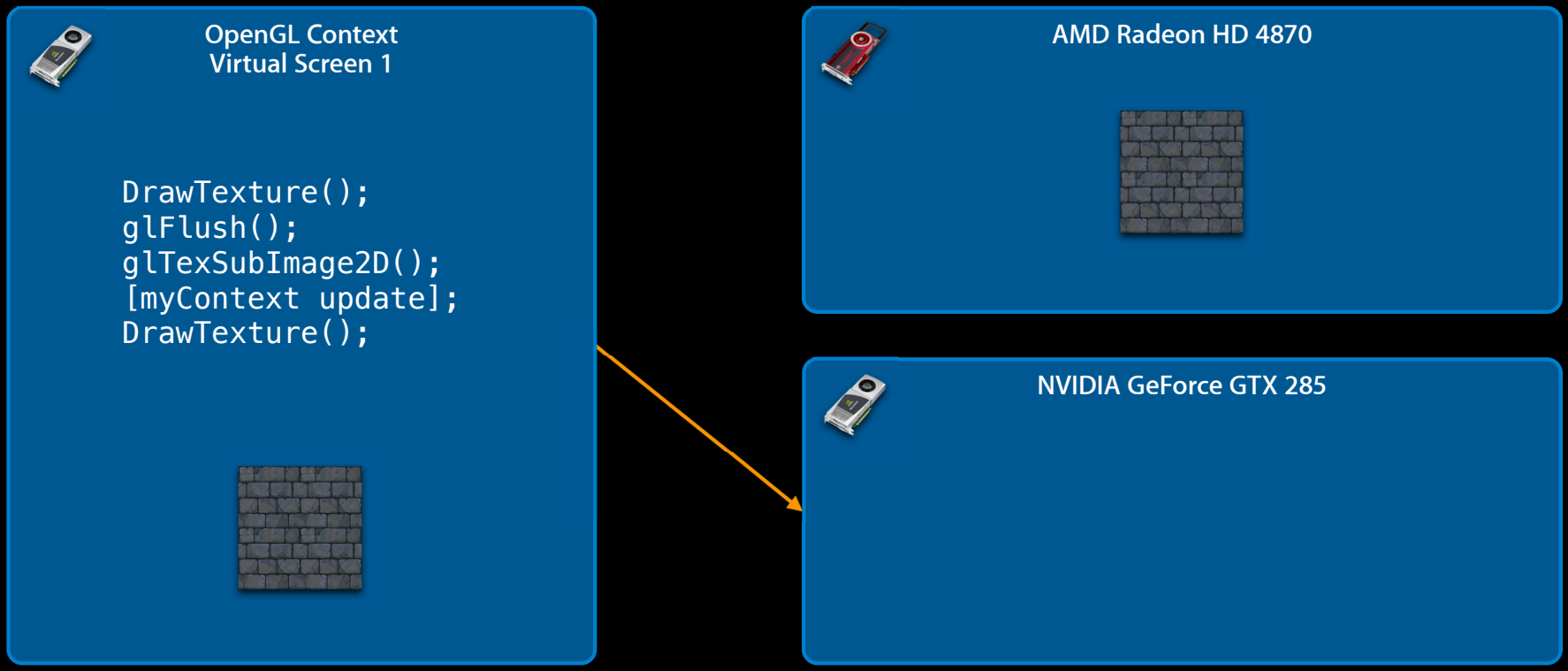
Basic Multi-GPU Support

Resource management during renderer changes



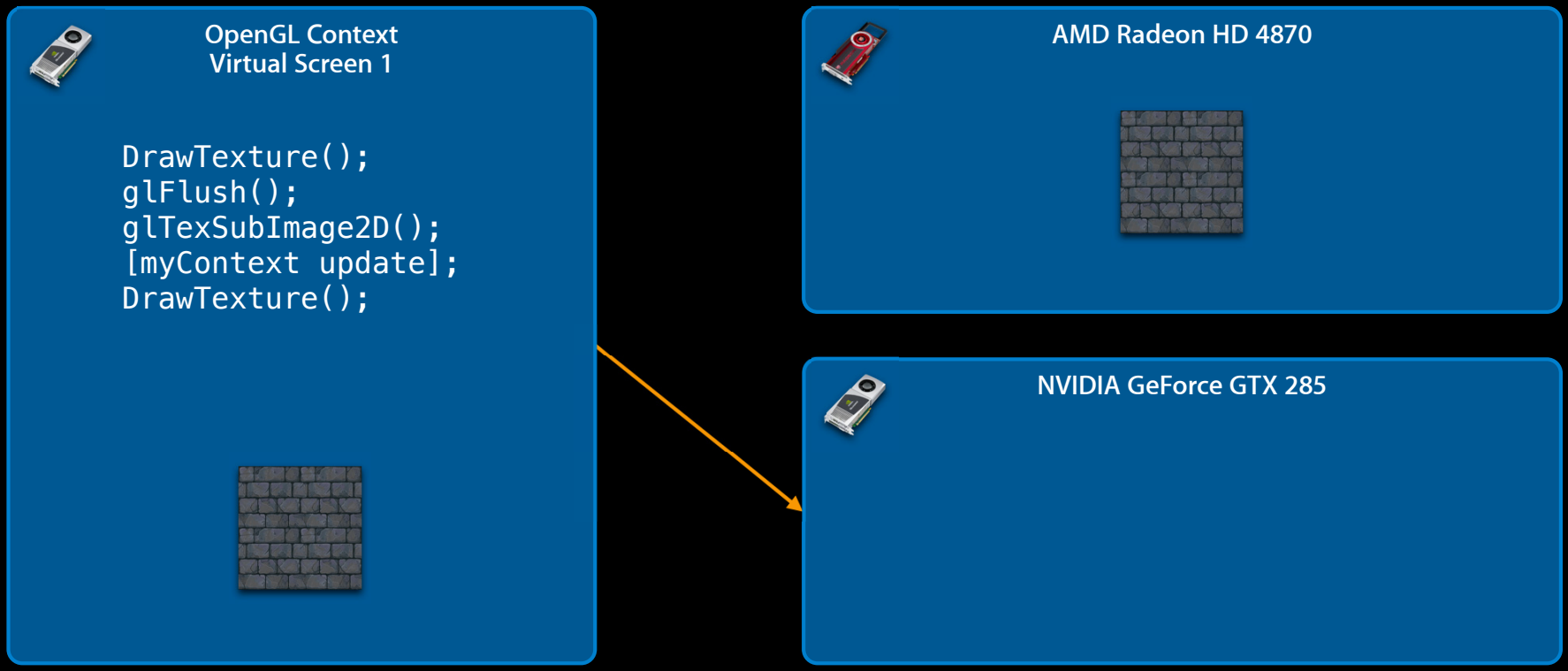
Basic Multi-GPU Support

Resource management during renderer changes



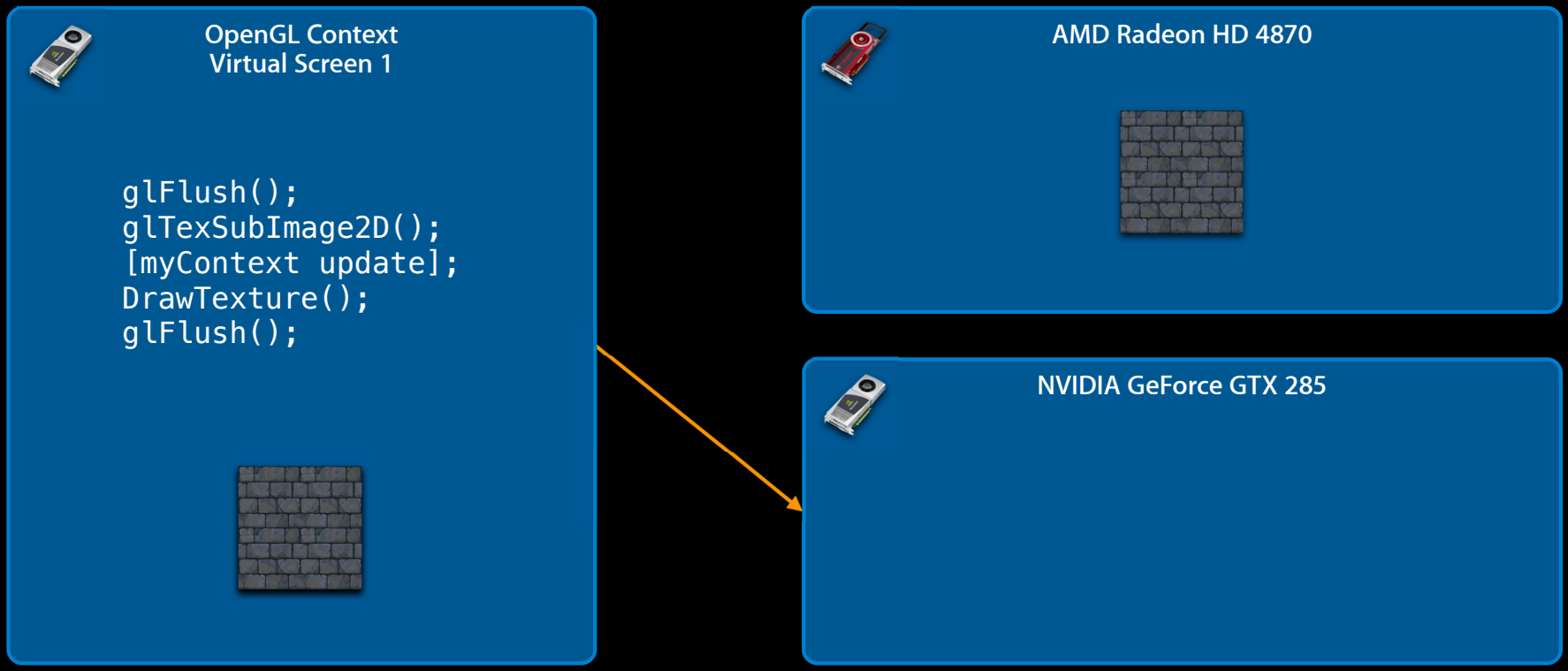
Basic Multi-GPU Support

Resource management during renderer changes



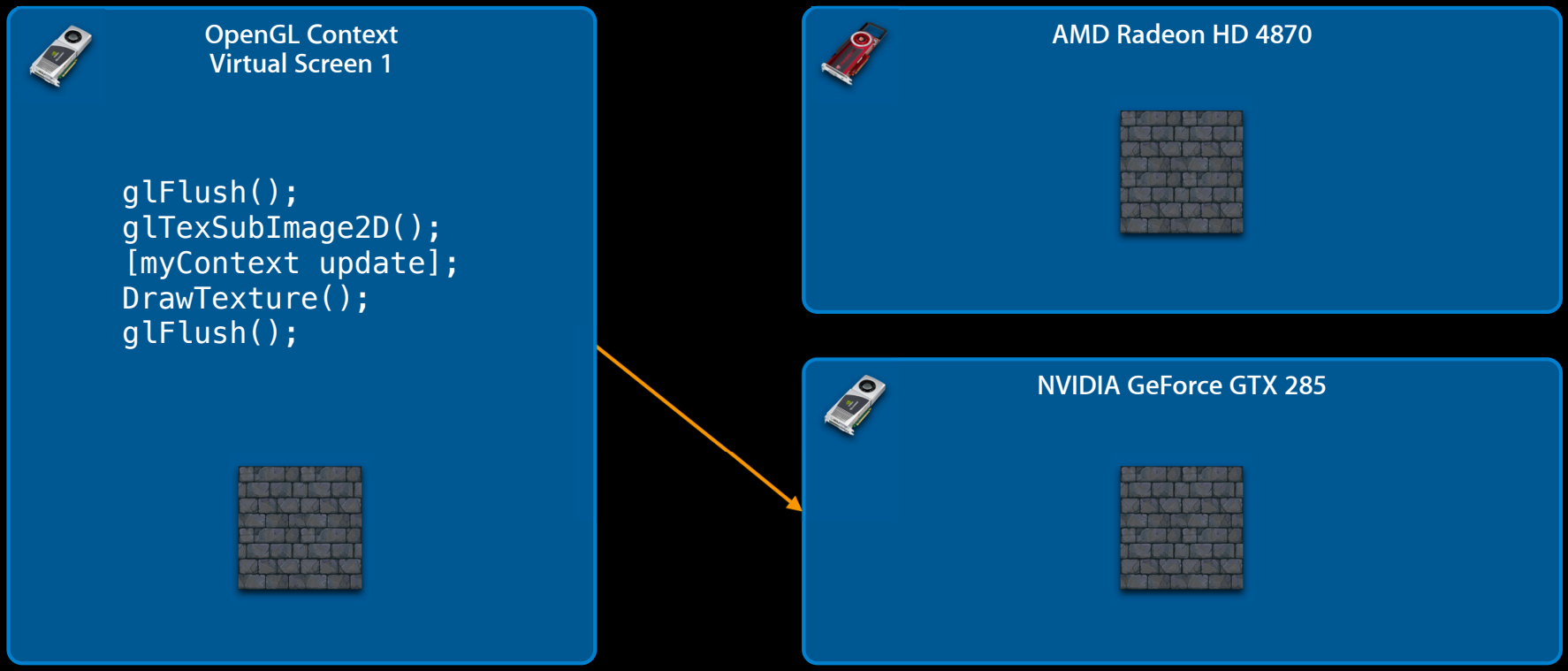
Basic Multi-GPU Support

Resource management during renderer changes



Basic Multi-GPU Support

Resource management during renderer changes



Demo: BoingX 2010

Kenneth Dyke
Sr. Mad Scientist

Advanced Multi-GPU Support

Advanced Multi-GPU Support

Using multiple GPUs simultaneously

- Motivations
  - Increased performance
  - Specific GPU feature requirements
- Issues to consider
  - Context sharing
  - Synchronization and resource management

Advanced Multi-GPU Support

Context sharing with OpenGL

- Multiple OpenGL contexts may share resources

```
CGLContextObj cgl_ctx;

err = CGLCreateContext(pix_fmt, NULL, &cgl_ctx);
```

Advanced Multi-GPU Support

Context sharing with OpenGL

- Multiple OpenGL contexts may share resources
- All contexts in the same *share group* must use the same set of renderers
  - Be very careful when using different pixel format objects
  - Safest way is to use the pixel format from the other context

```
CGLContextObj cgl_ctx;

err = CGLCreateContext(pix_fmt, share_ctx, &cgl_ctx);
```

Advanced Multi-GPU Support

Context sharing with OpenGL

- Multiple OpenGL contexts may share resources
- All contexts in the same *share group* must use the same set of renderers
  - Be very careful when using different pixel format objects
  - Safest way is to use the pixel format from the other context

```
CGLContextObj cgl_ctx;
CGLPixelFormatObj share_pix = CGLGetPixelFormat(share_ctx);

err = CGLCreateContext(share_pix, share_ctx, &cgl_ctx);
```

Advanced Multi-GPU Support

Context sharing in OpenCL

- All queues created from the same OpenCL context will share resources
- Can also create OpenCL contexts that are compatible with OpenGL

```
CGLContextObj cgl_ctx = [[self openGLContext] CGLContextObj];
cl_int err = 0;

cl_context_properties properties[] = {
    CL_CONTEXT_PROPERTY_USE_CGL_SHAREGROUP_APPLE,
    (cl_context_properties)CGLGetShareGroup(context),
    0
};

computeContext = clCreateContext(properties, 0, 0, 0, 0, &err);
```

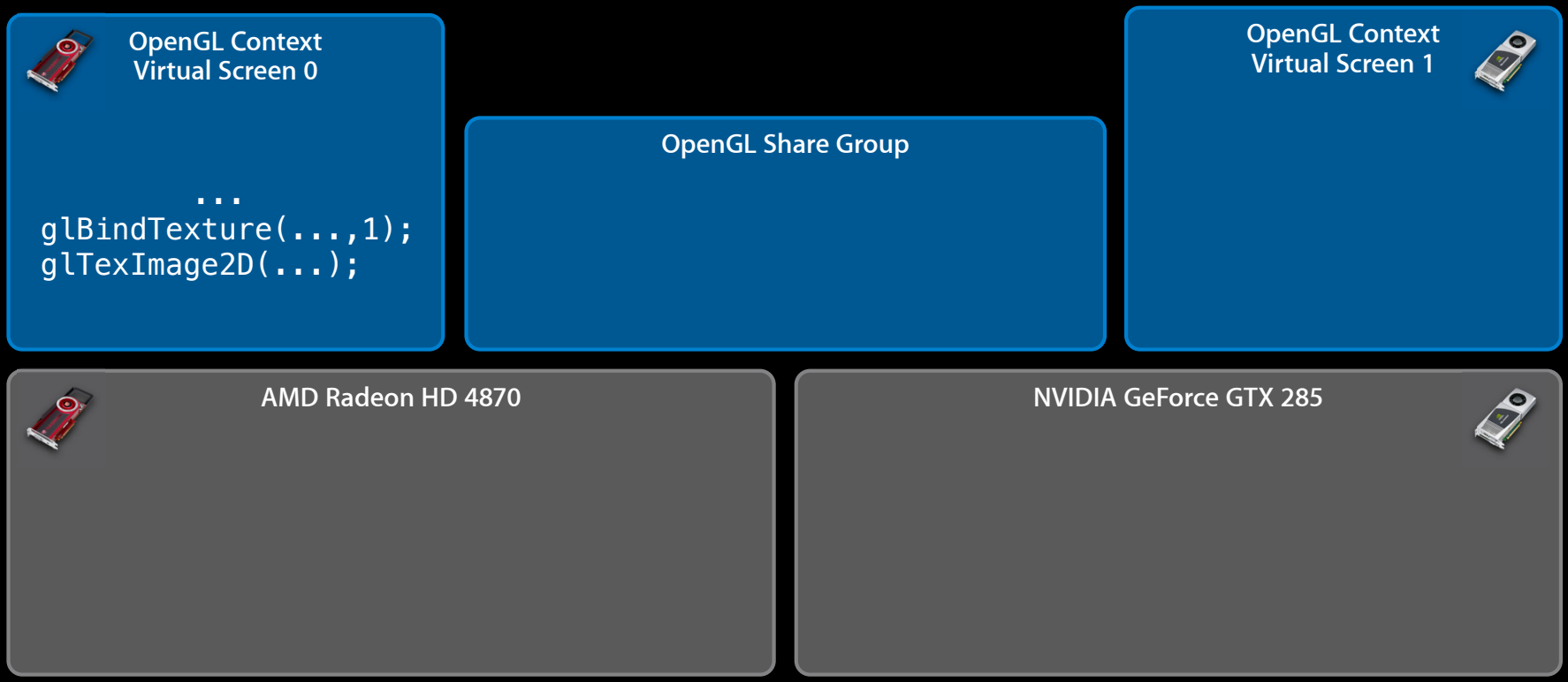
Advanced Multi-GPU Support

Multicontext synchronization

- Maintaining order of operations is very important
- OpenGL uses 'flush' then 'bind' semantics
  - Producer context must *flush* (`glFlush`, `glFinish`, etc.)
  - Consumer context must *bind* (`glBindTexture`, etc.)
- Applies to *both* single and multi-GPU cases
- OpenCL uses events for dependencies
- Data dependencies between GL and CL require `glFlush`/`clAcquire`

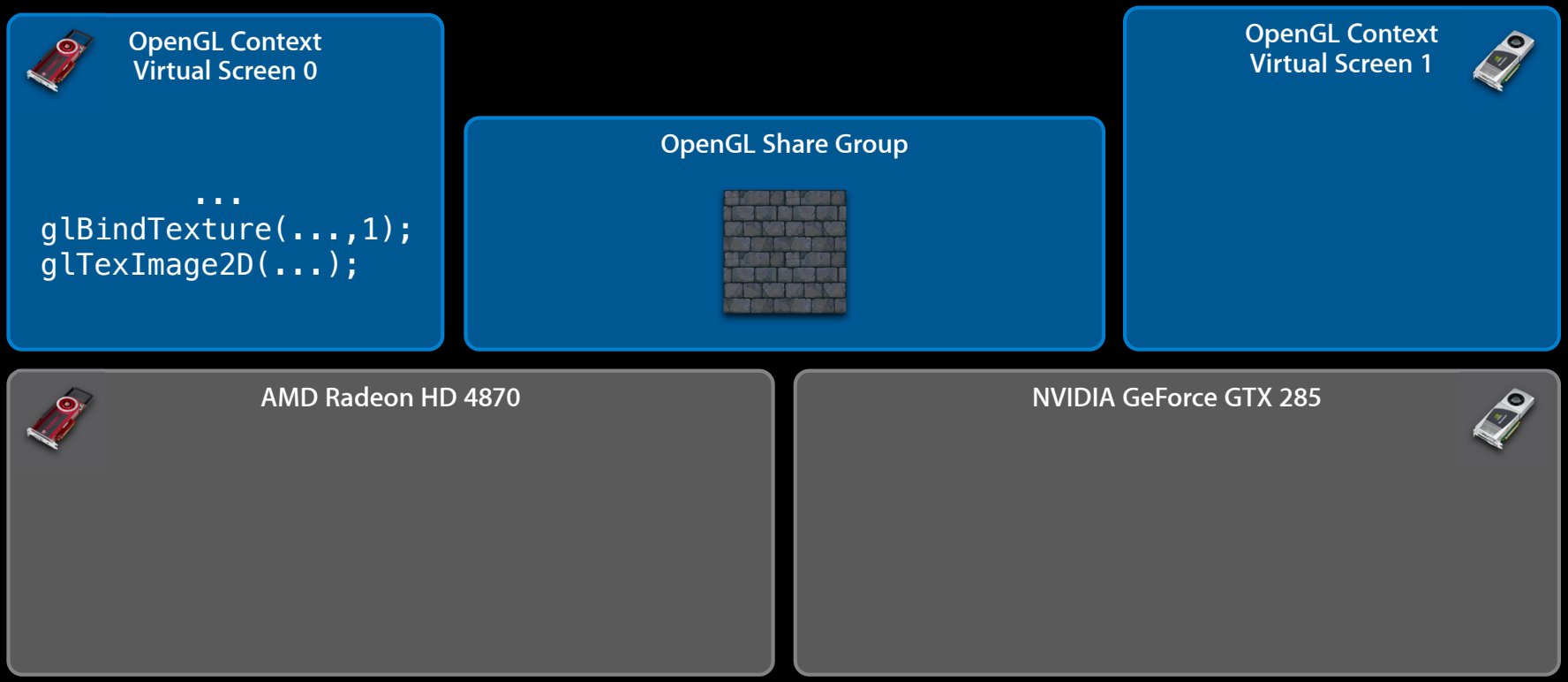
Advanced Multi-GPU Support

Multicontext resource management



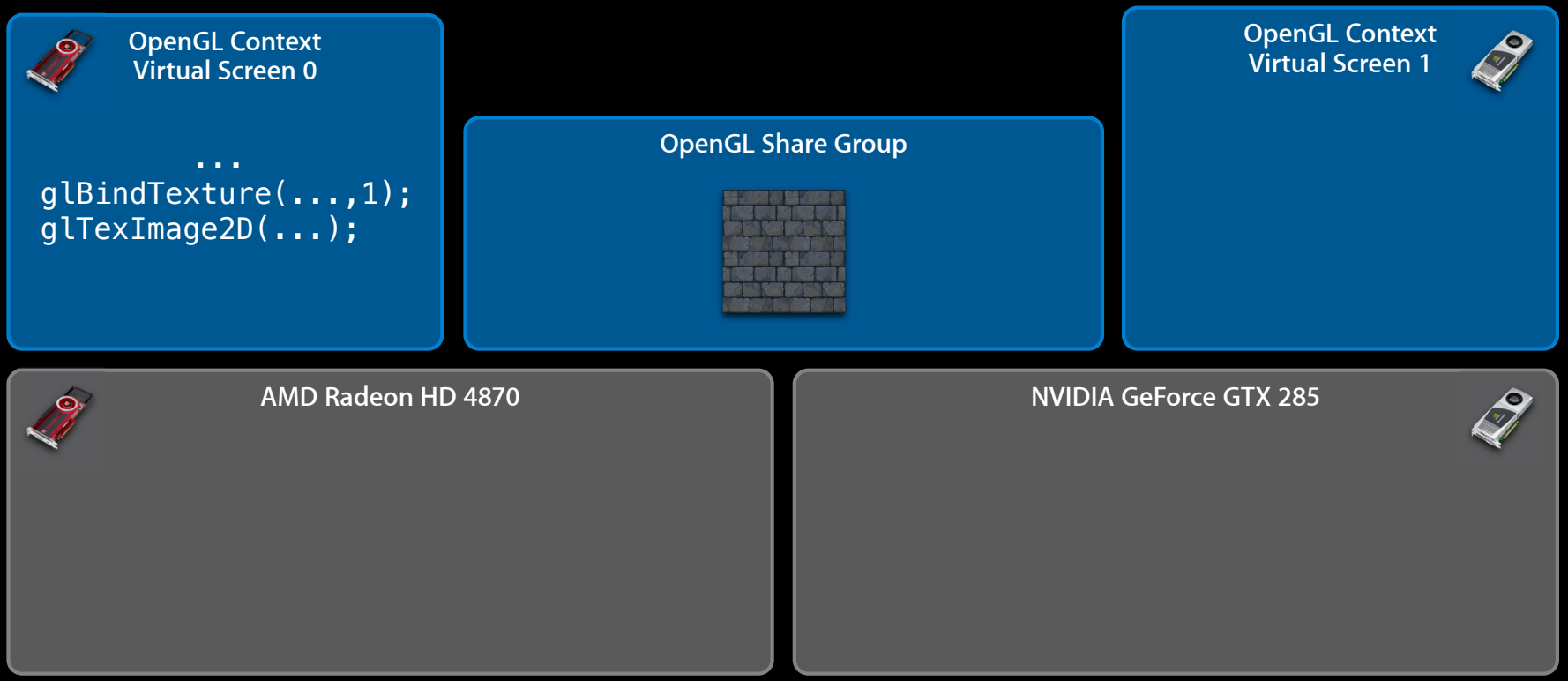
Advanced Multi-GPU Support

Multicontext resource management



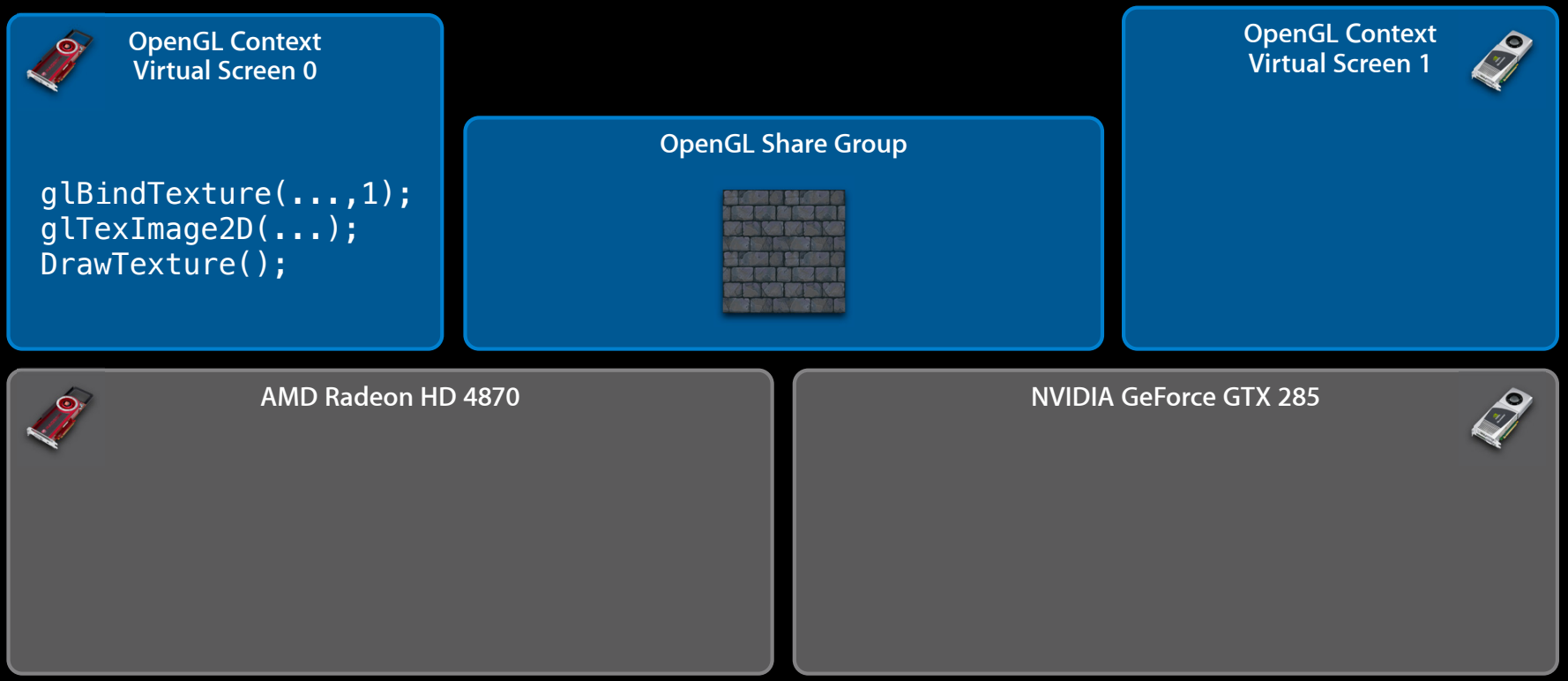
Advanced Multi-GPU Support

Multicontext resource management



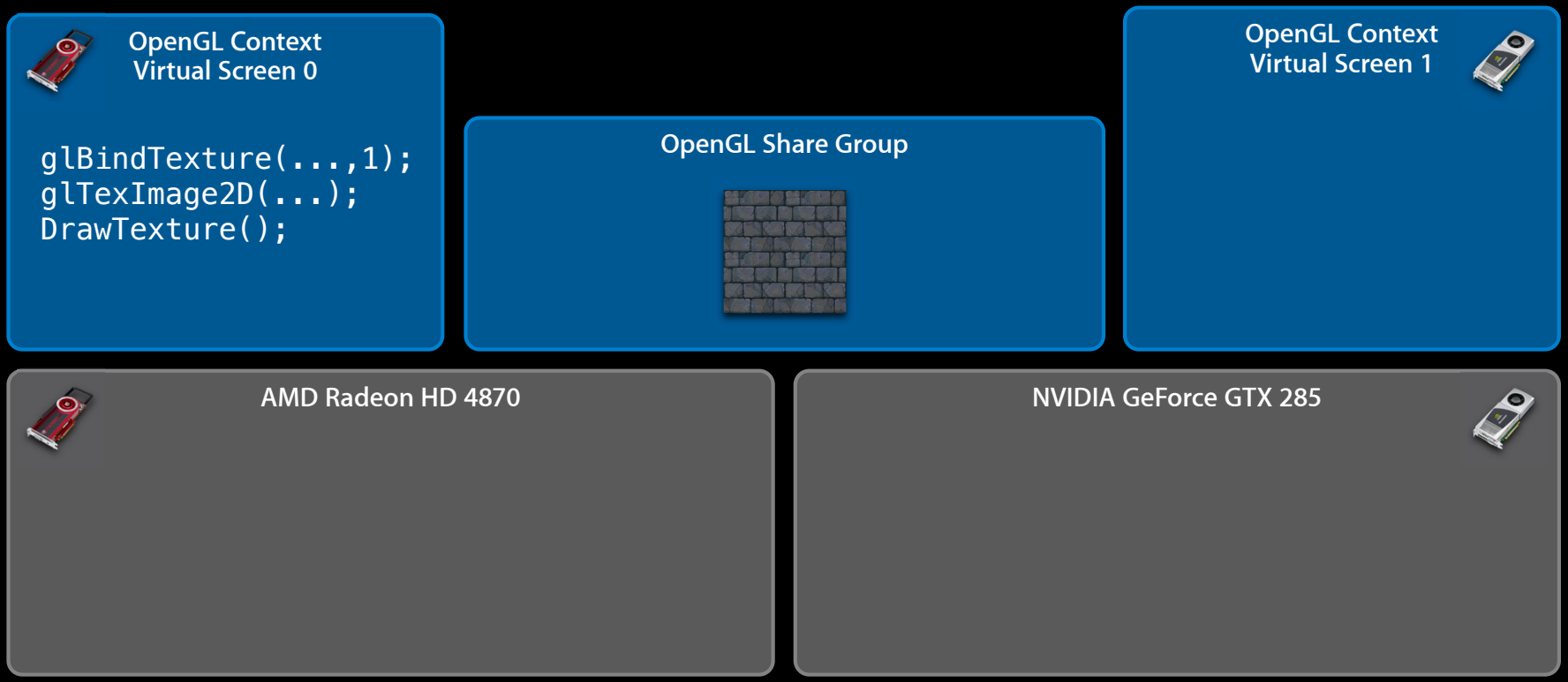
Advanced Multi-GPU Support

Multicontext resource management



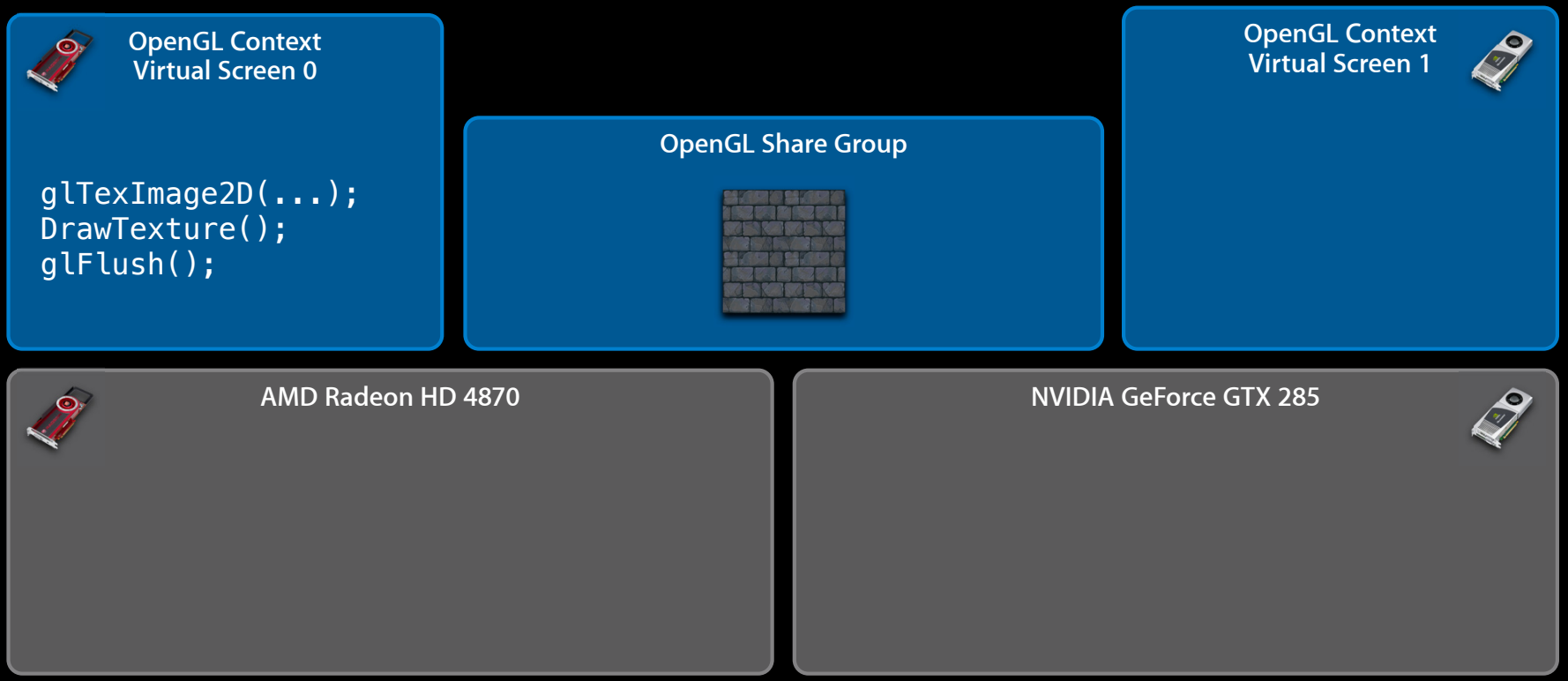
Advanced Multi-GPU Support

Multicontext resource management



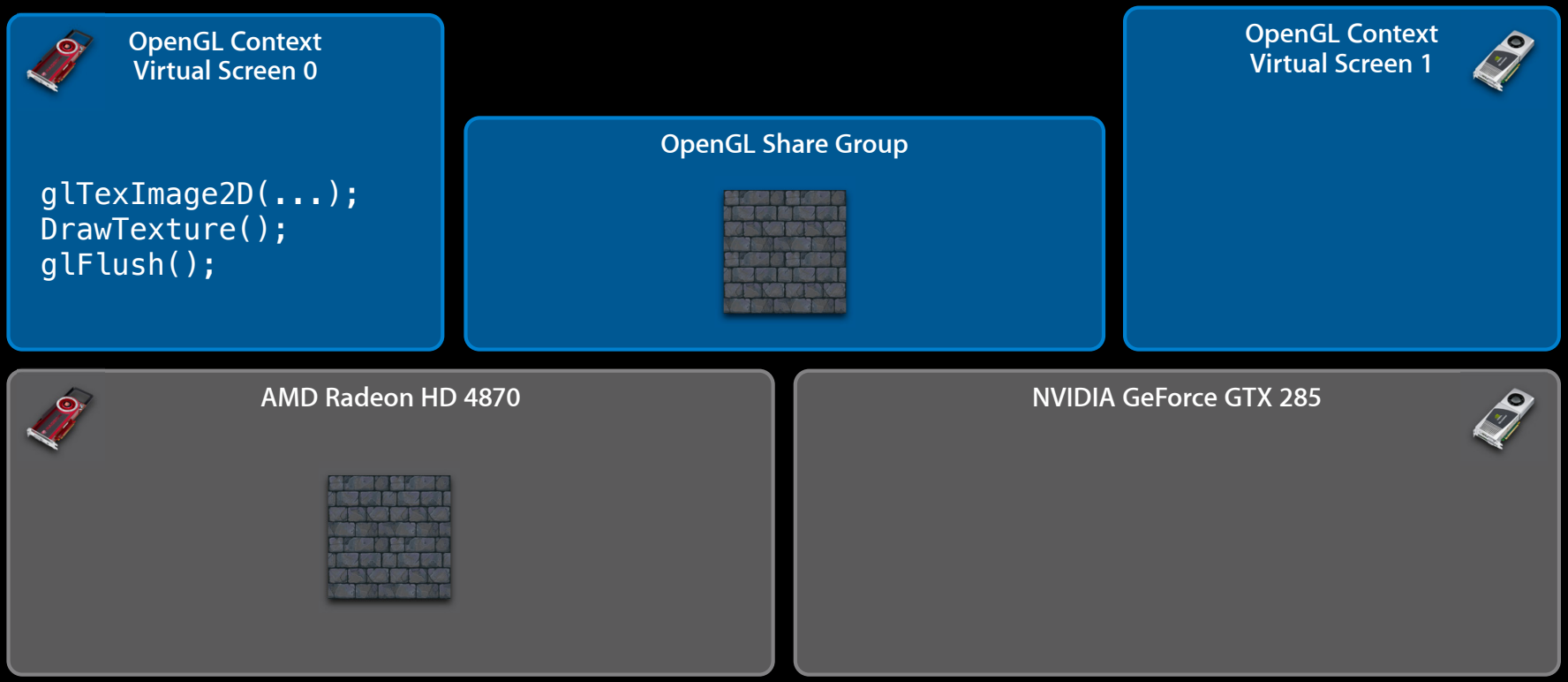
Advanced Multi-GPU Support

Multicontext resource management



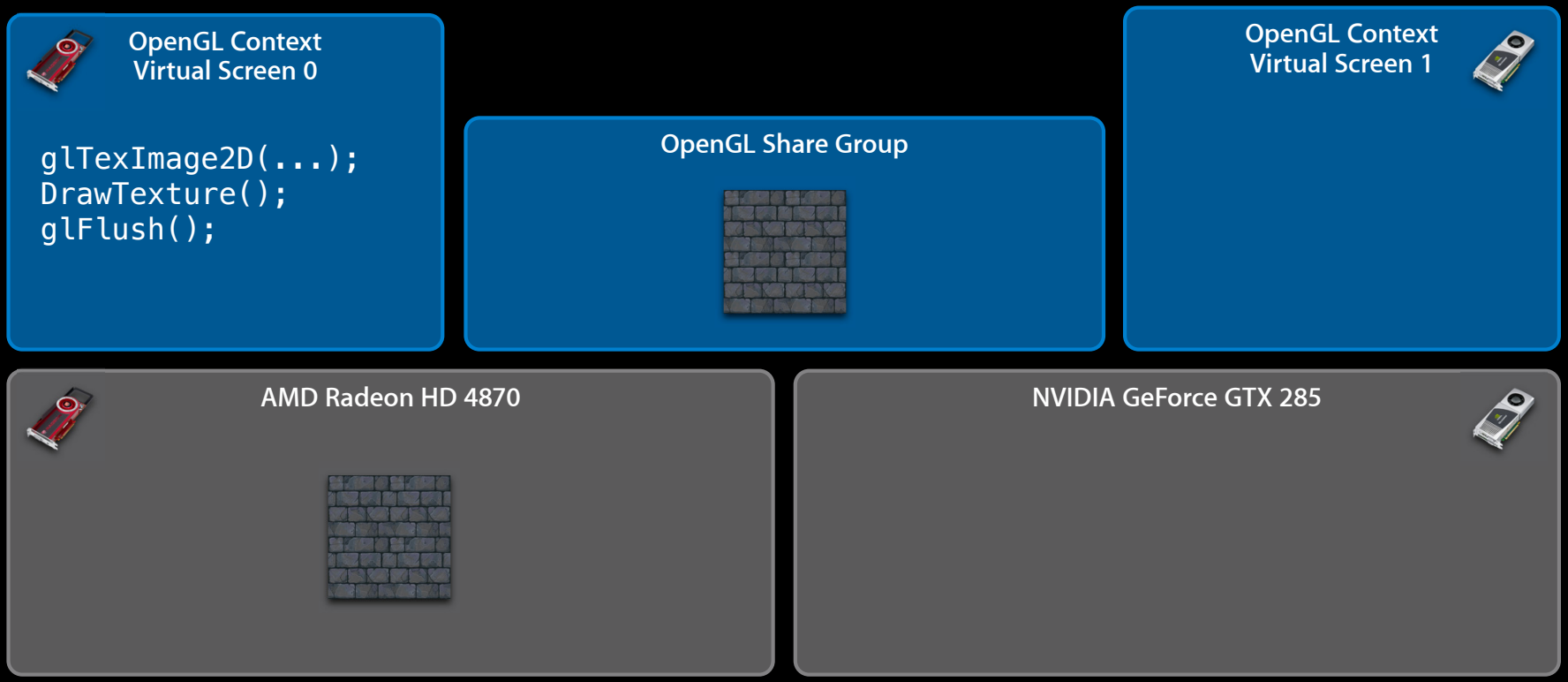
Advanced Multi-GPU Support

Multicontext resource management



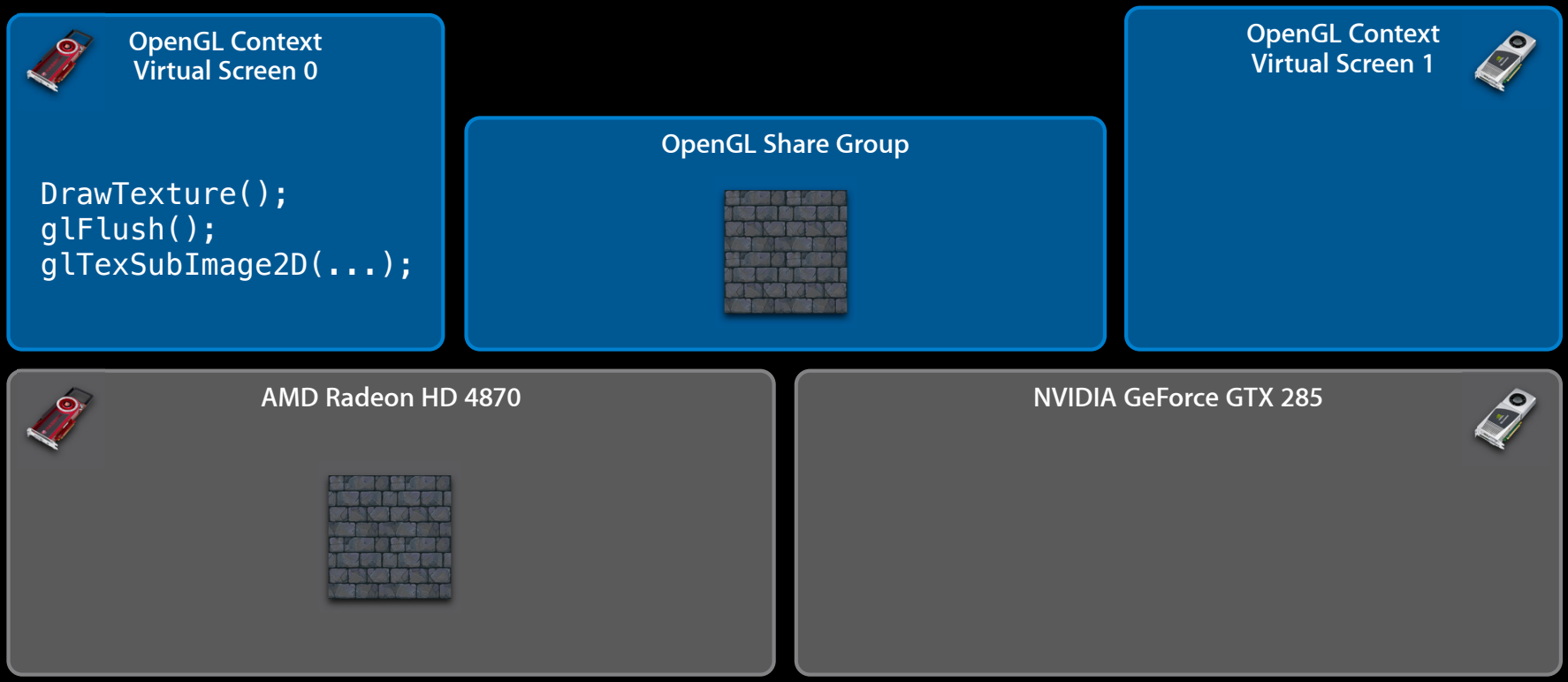
Advanced Multi-GPU Support

Multicontext resource management



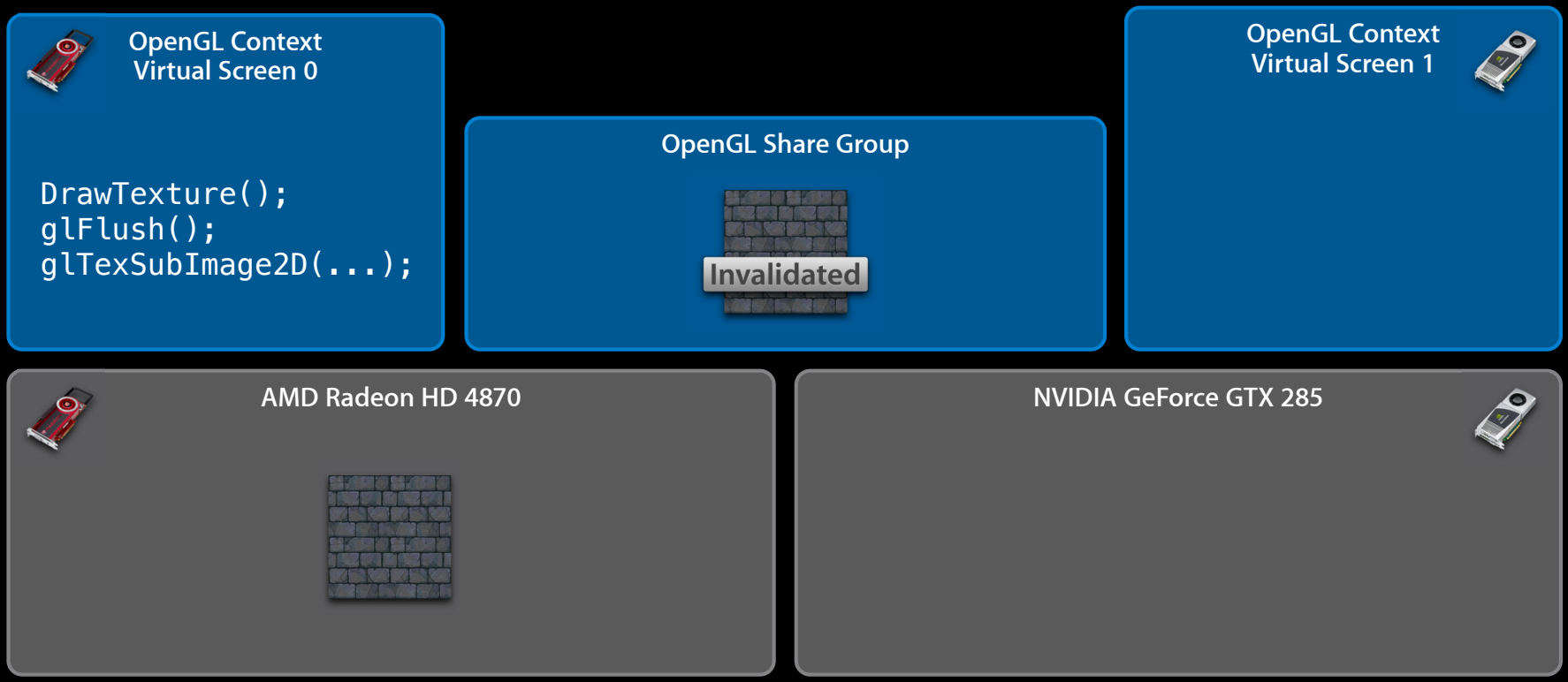
Advanced Multi-GPU Support

Multicontext resource management



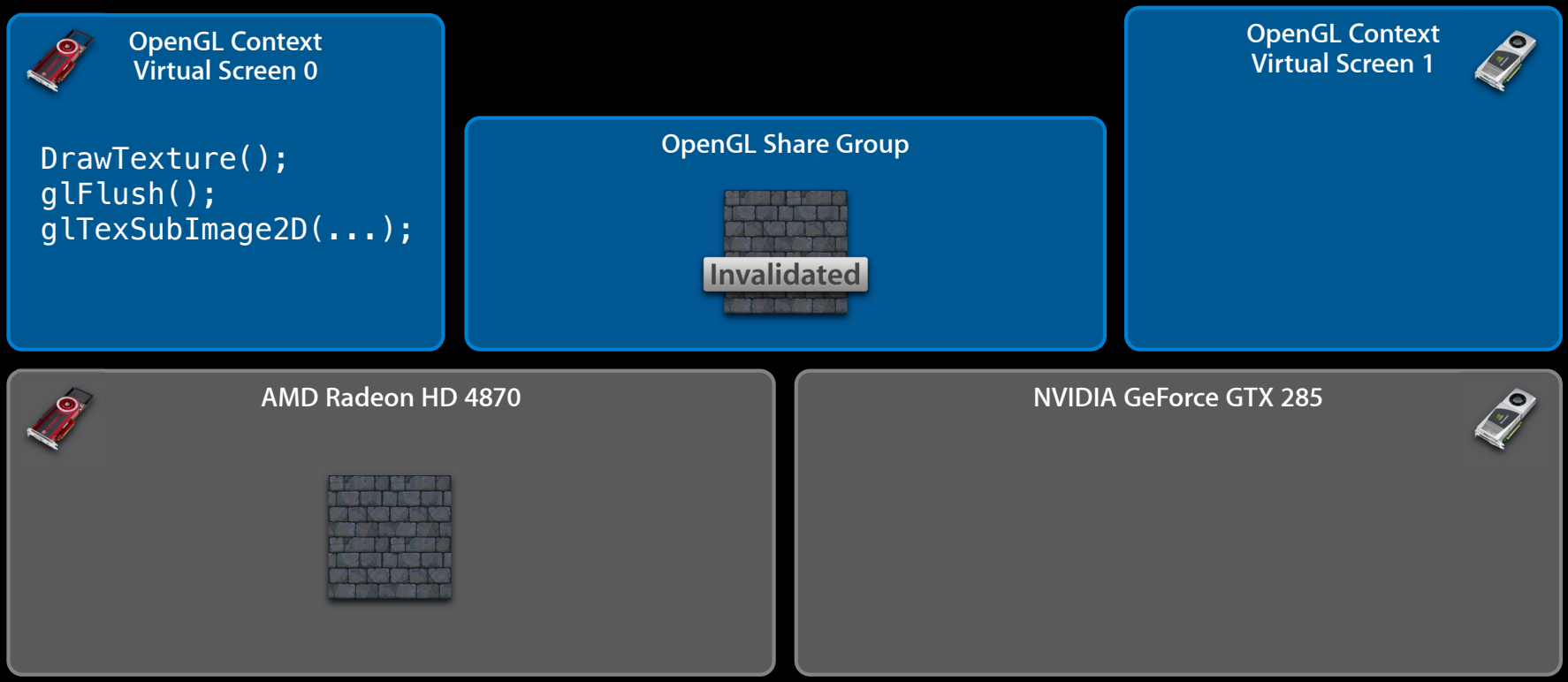
Advanced Multi-GPU Support

Multicontext resource management



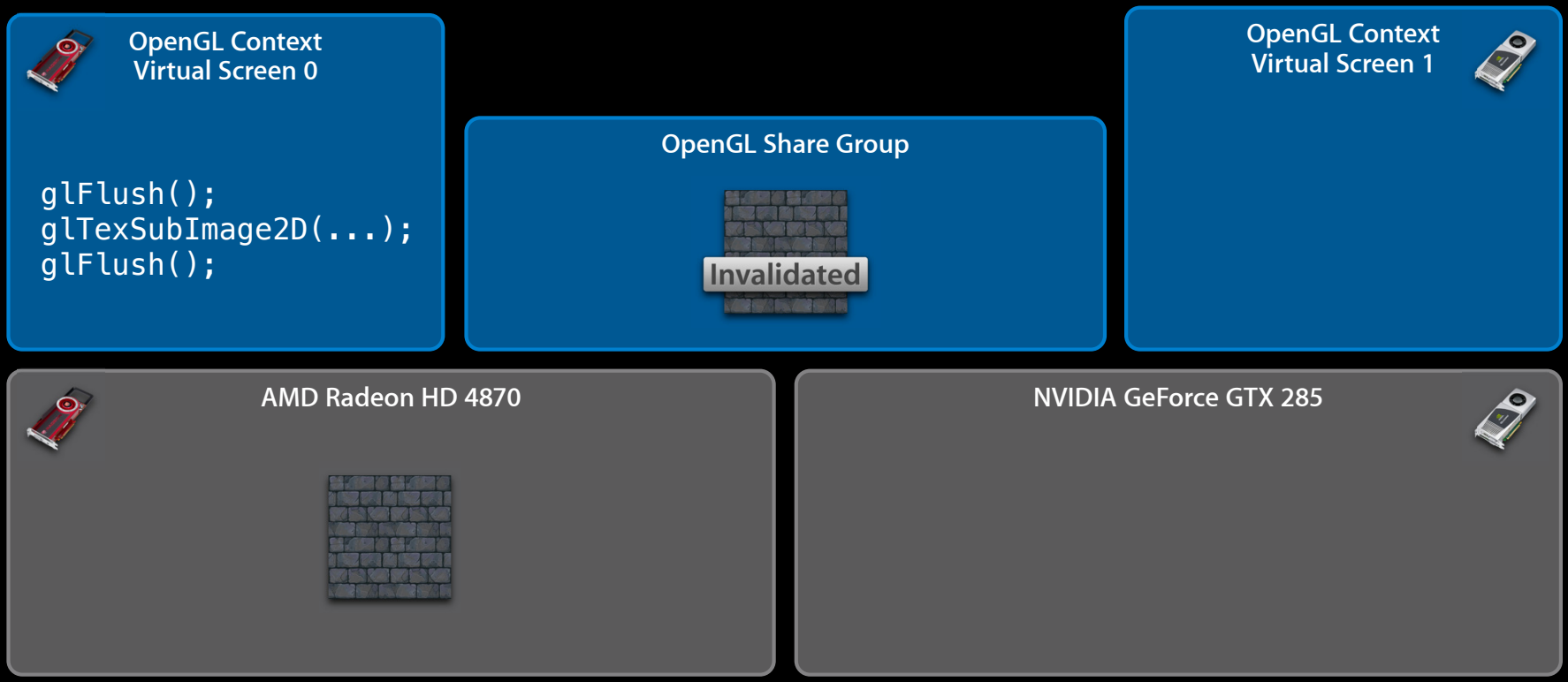
Advanced Multi-GPU Support

Multicontext resource management



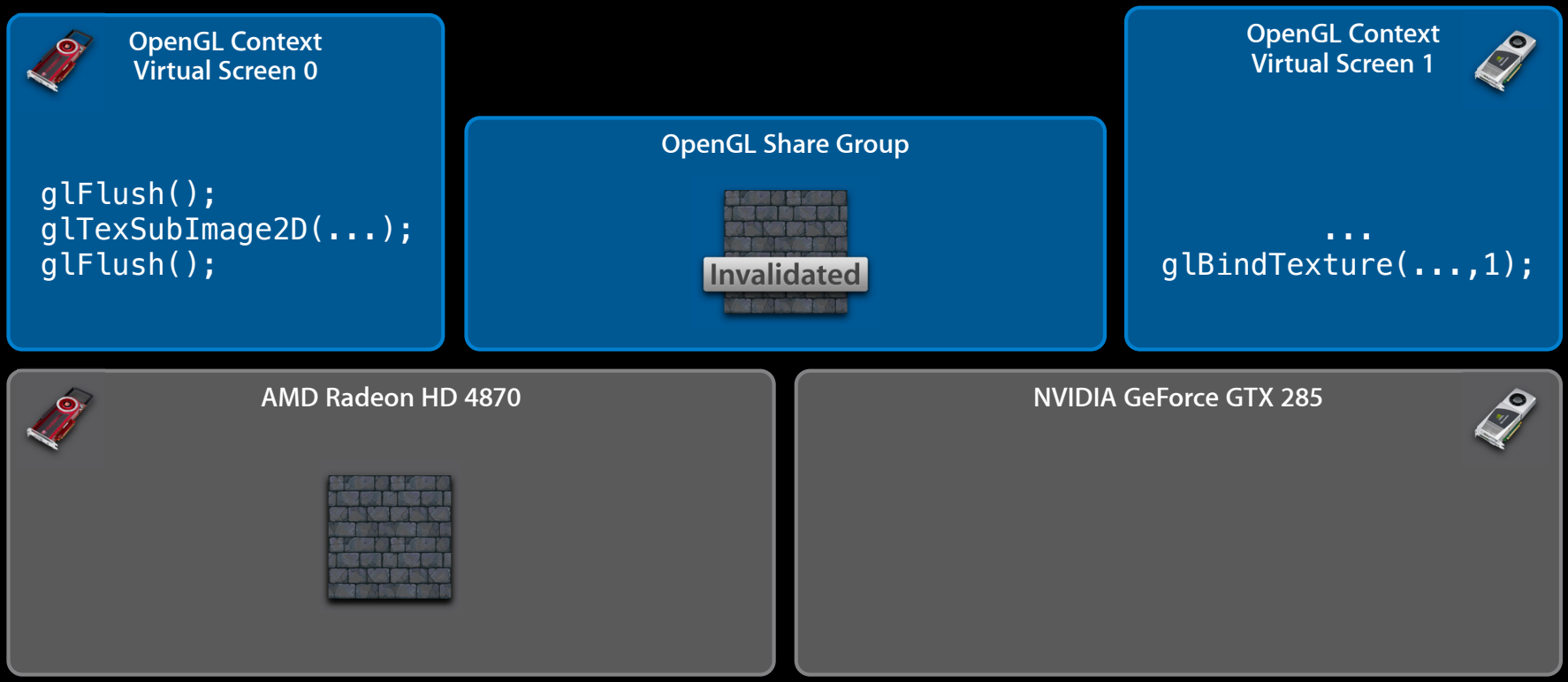
Advanced Multi-GPU Support

Multicontext resource management



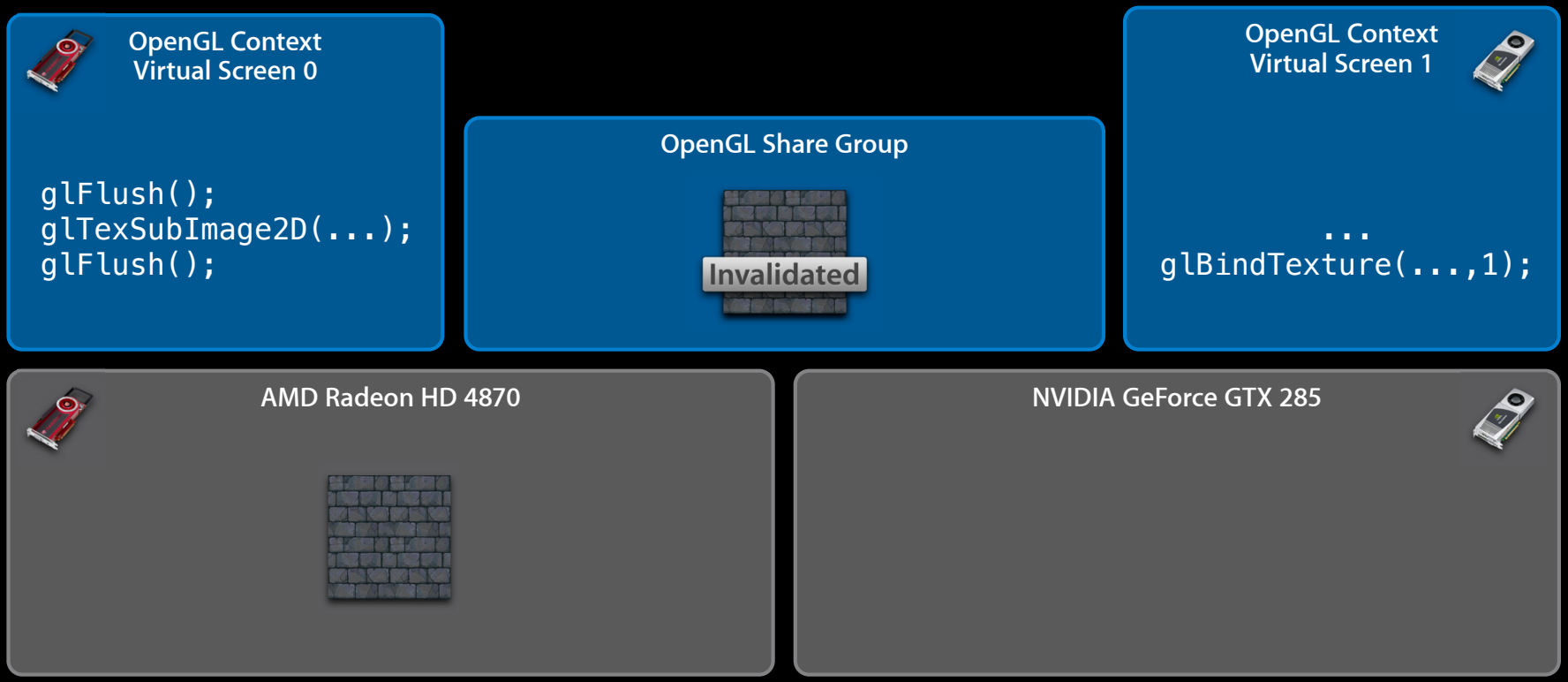
Advanced Multi-GPU Support

Multicontext resource management



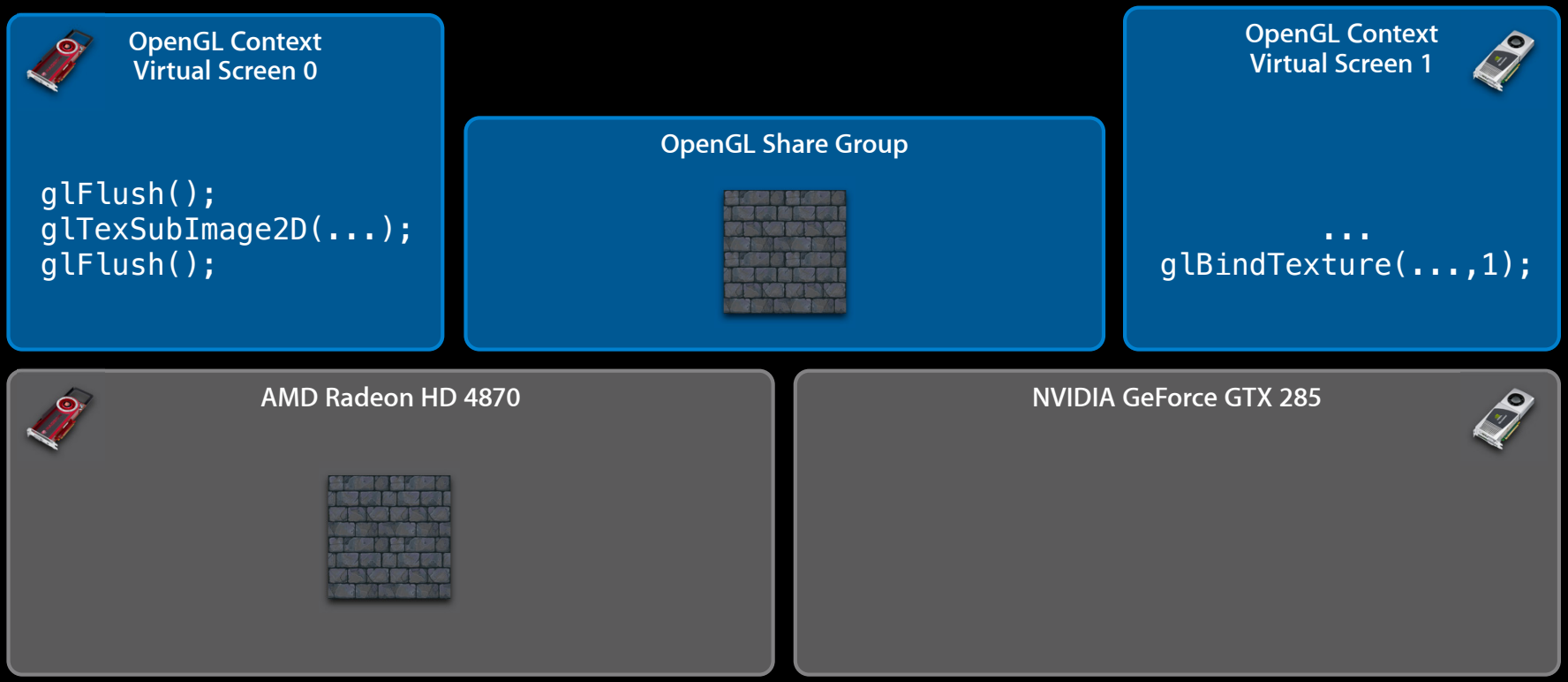
Advanced Multi-GPU Support

Multicontext resource management



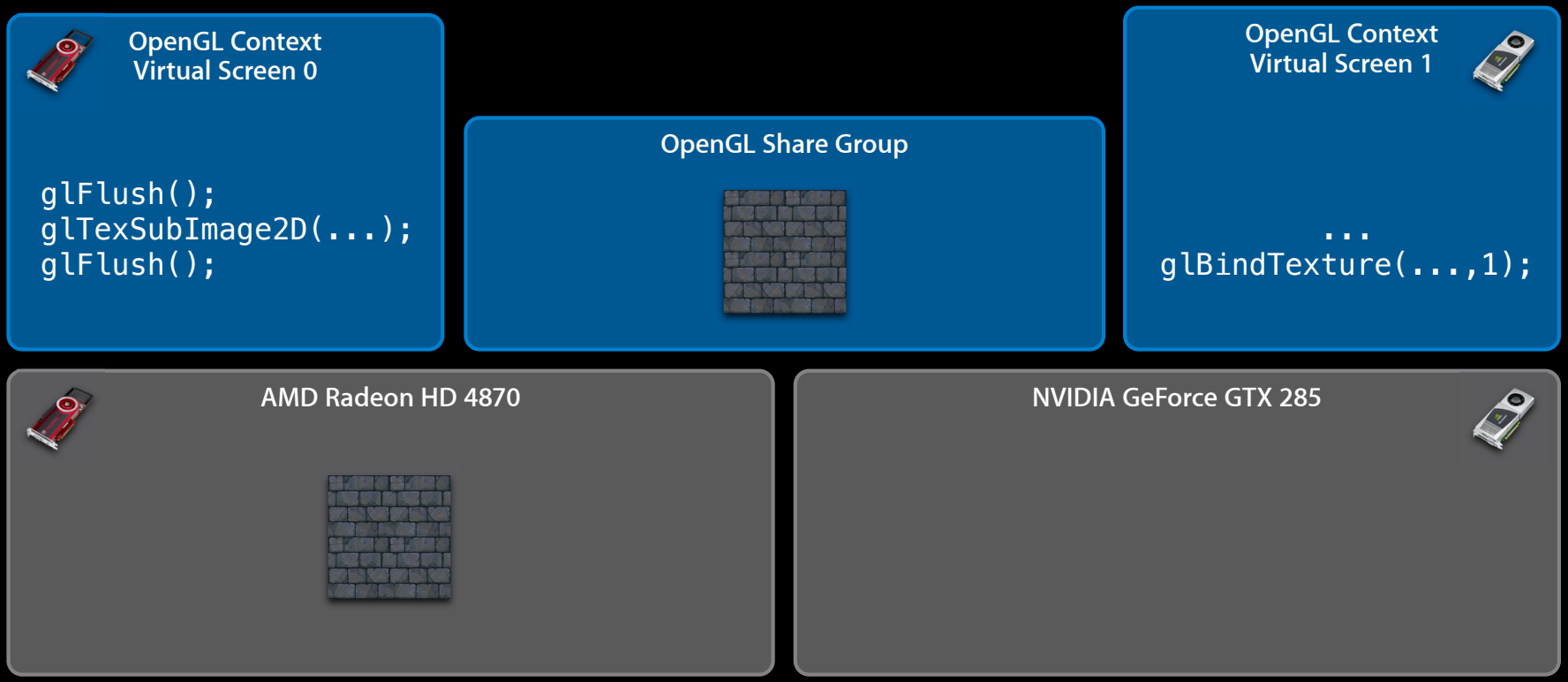
Advanced Multi-GPU Support

Multicontext resource management



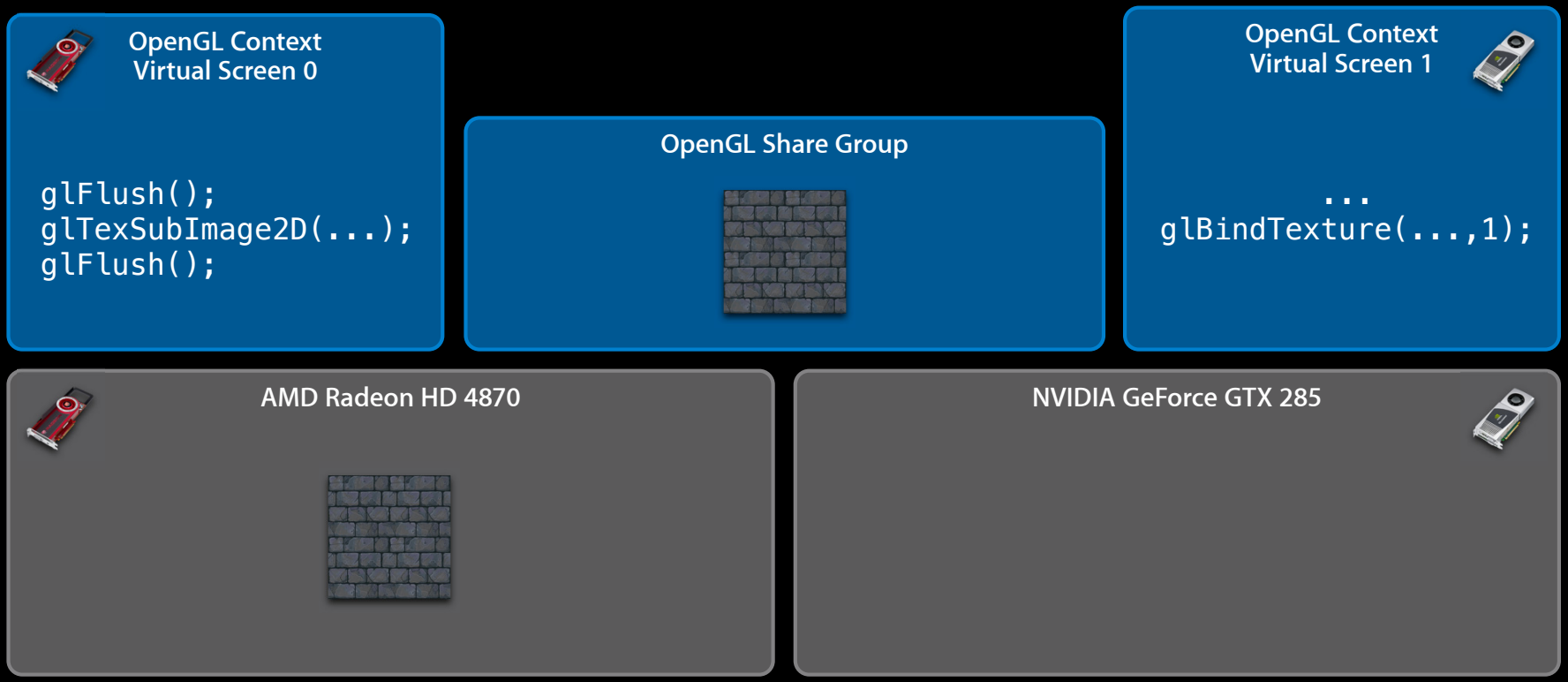
Advanced Multi-GPU Support

Multicontext resource management



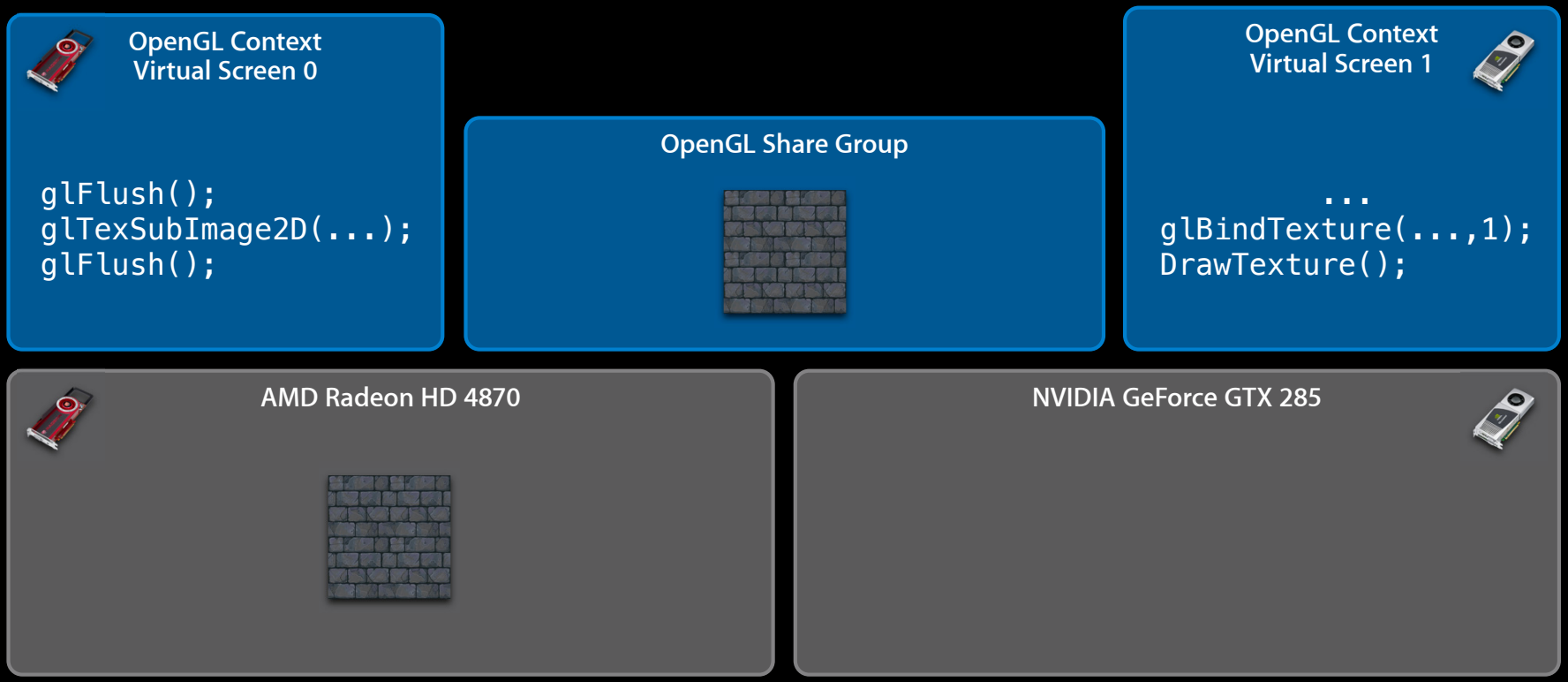
Advanced Multi-GPU Support

Multicontext resource management



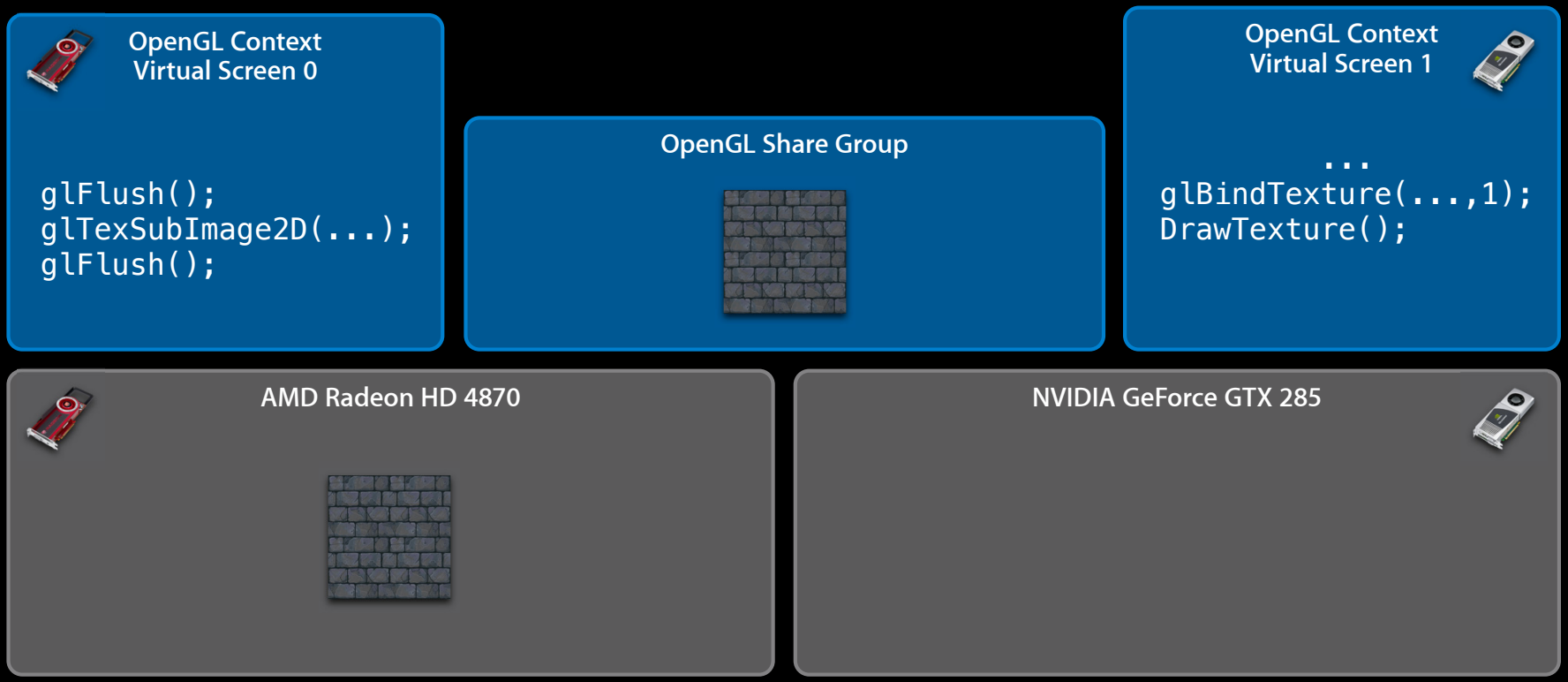
Advanced Multi-GPU Support

Multicontext resource management



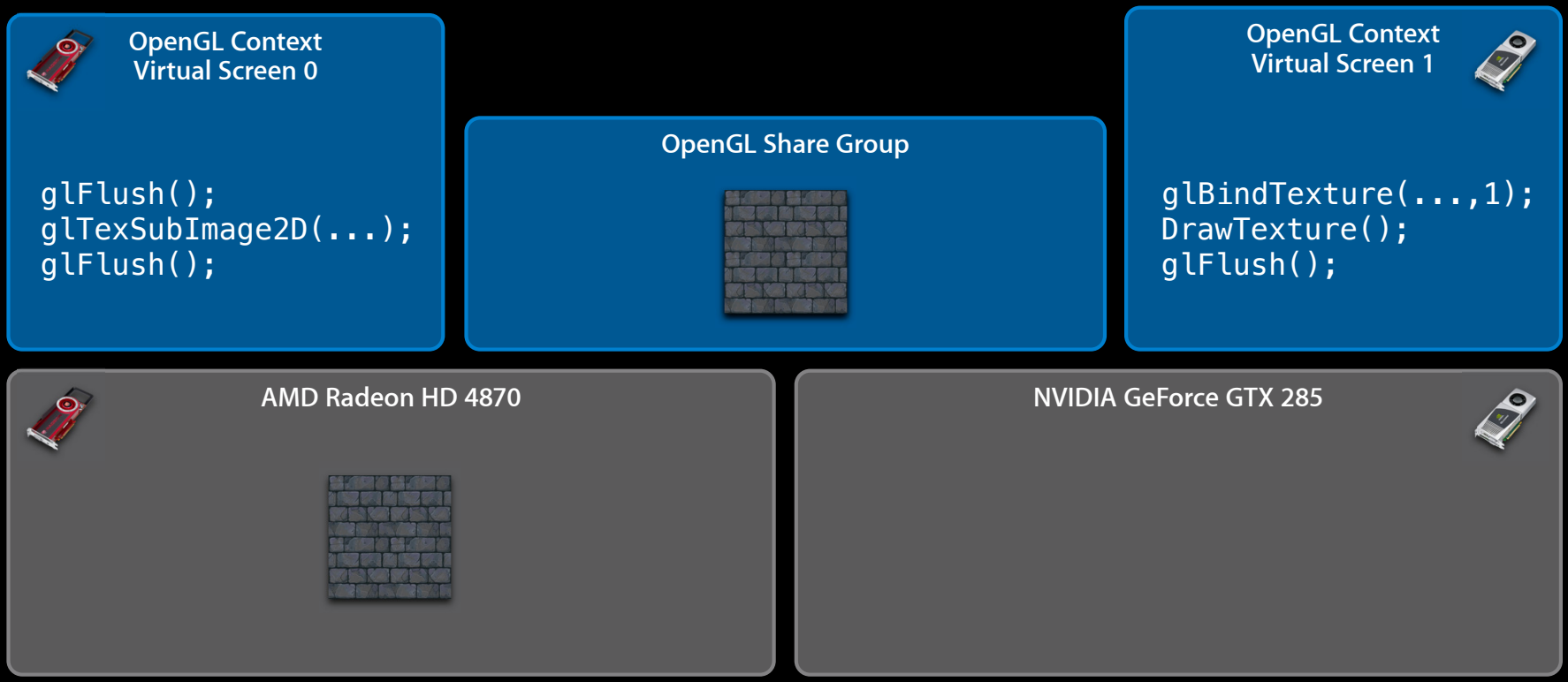
Advanced Multi-GPU Support

Multicontext resource management



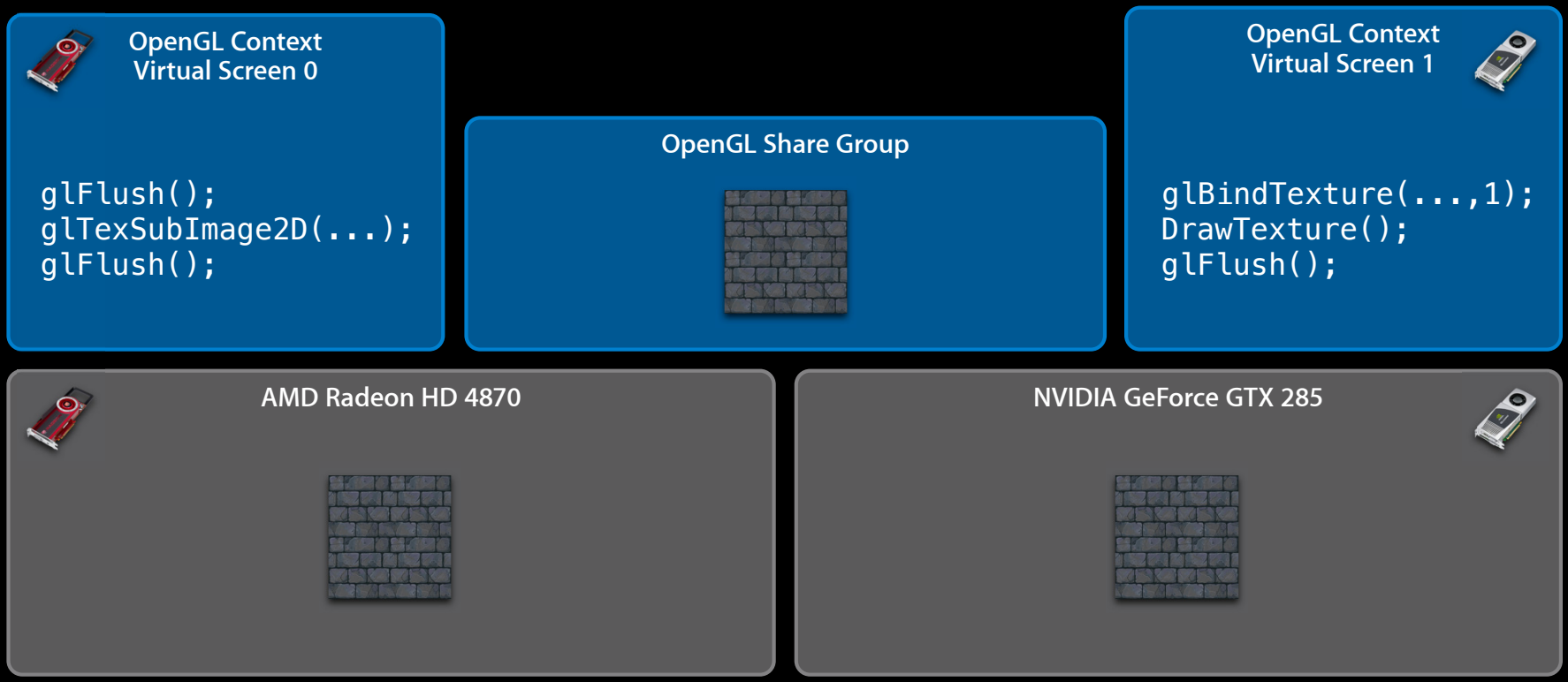
Advanced Multi-GPU Support

Multicontext resource management



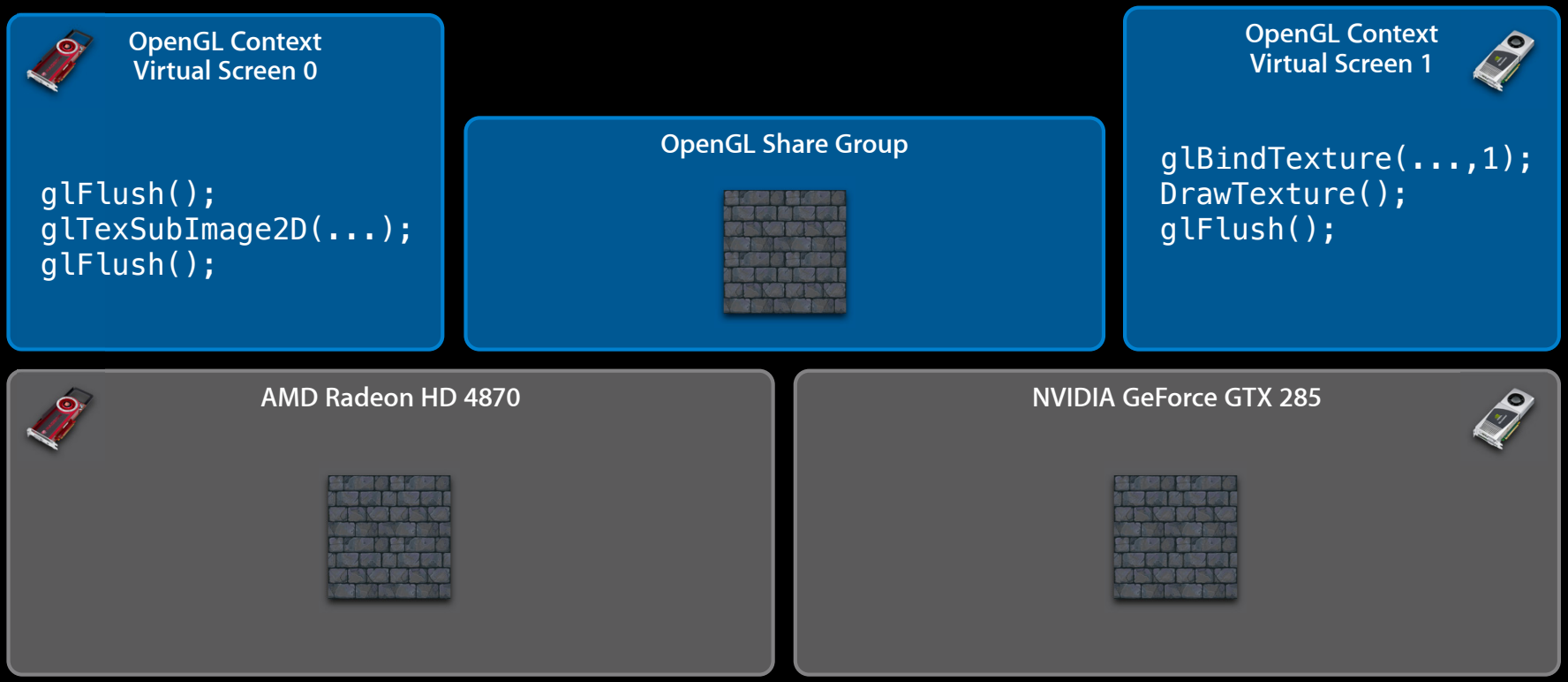
Advanced Multi-GPU Support

Multicontext resource management



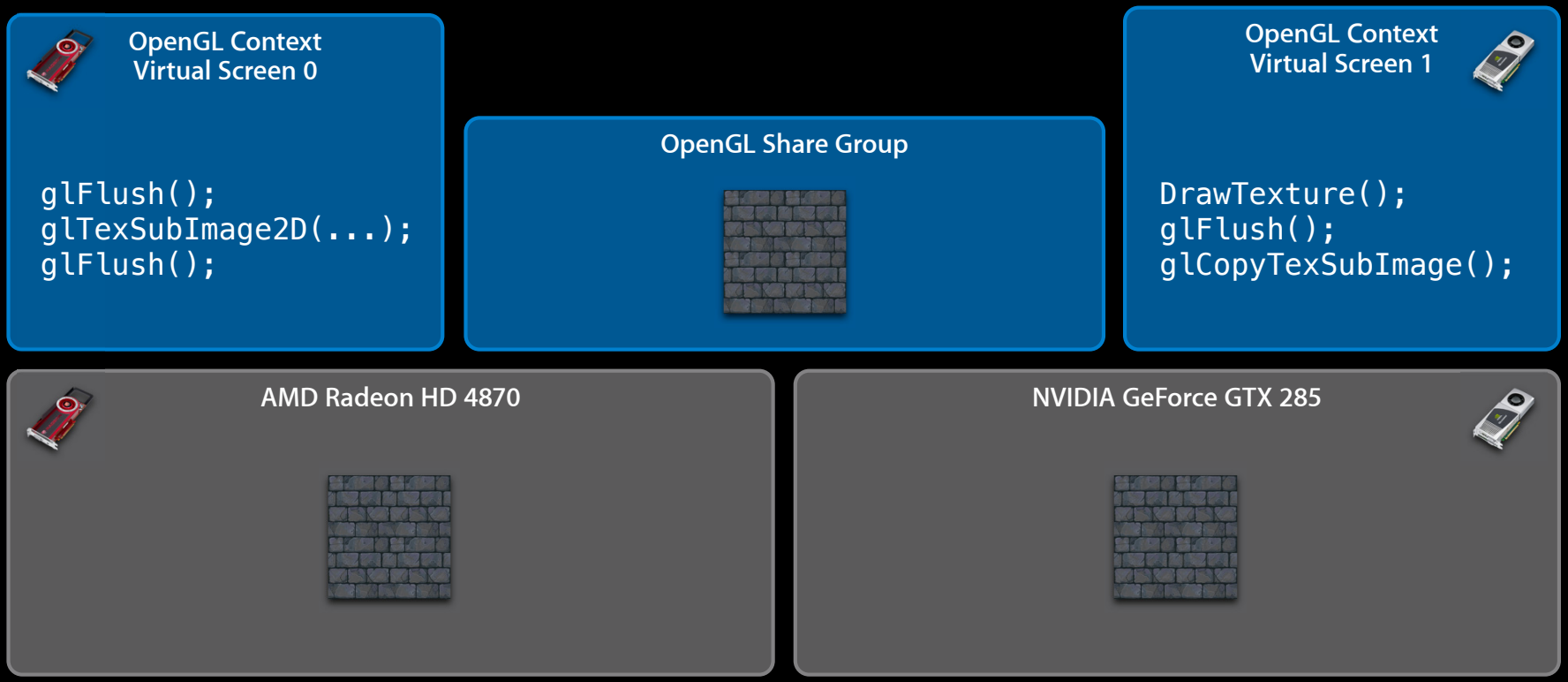
Advanced Multi-GPU Support

Multicontext resource management



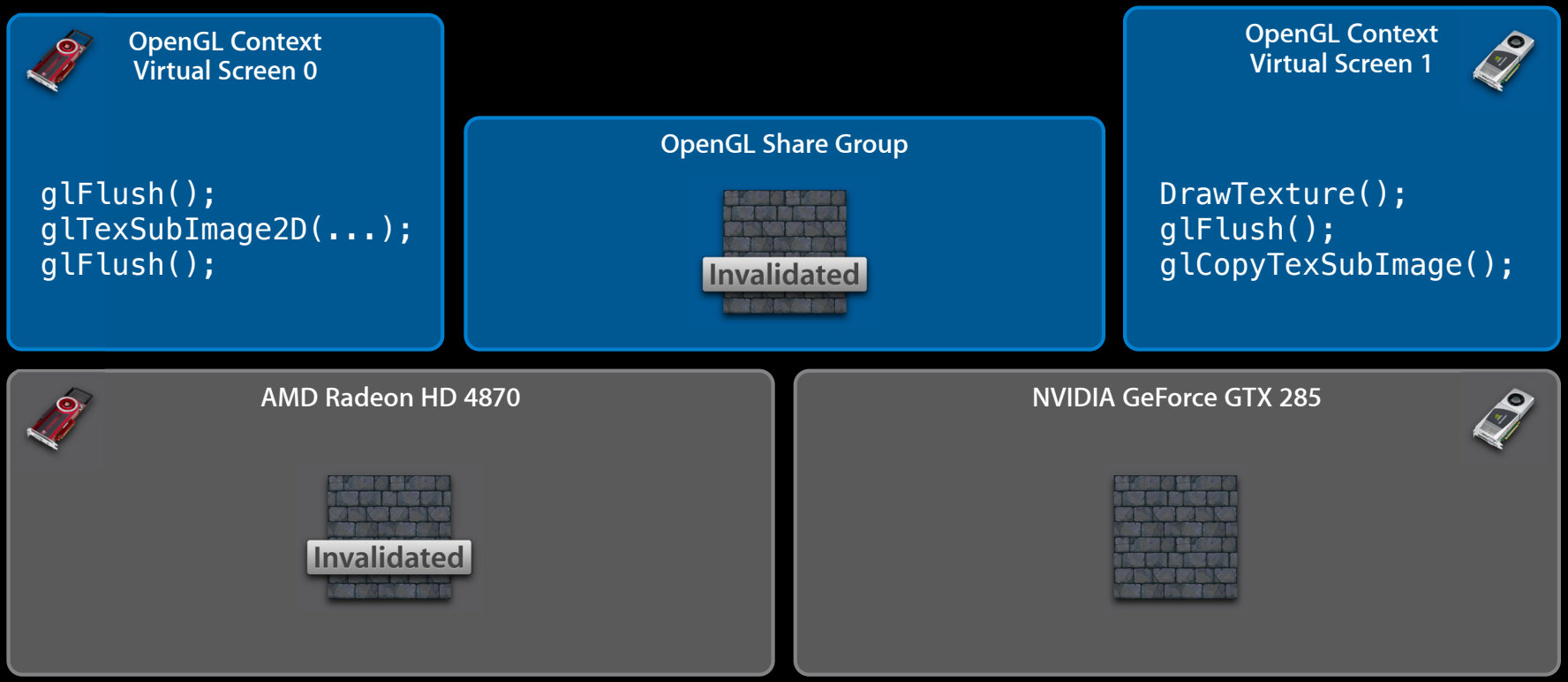
Advanced Multi-GPU Support

Multicontext resource management



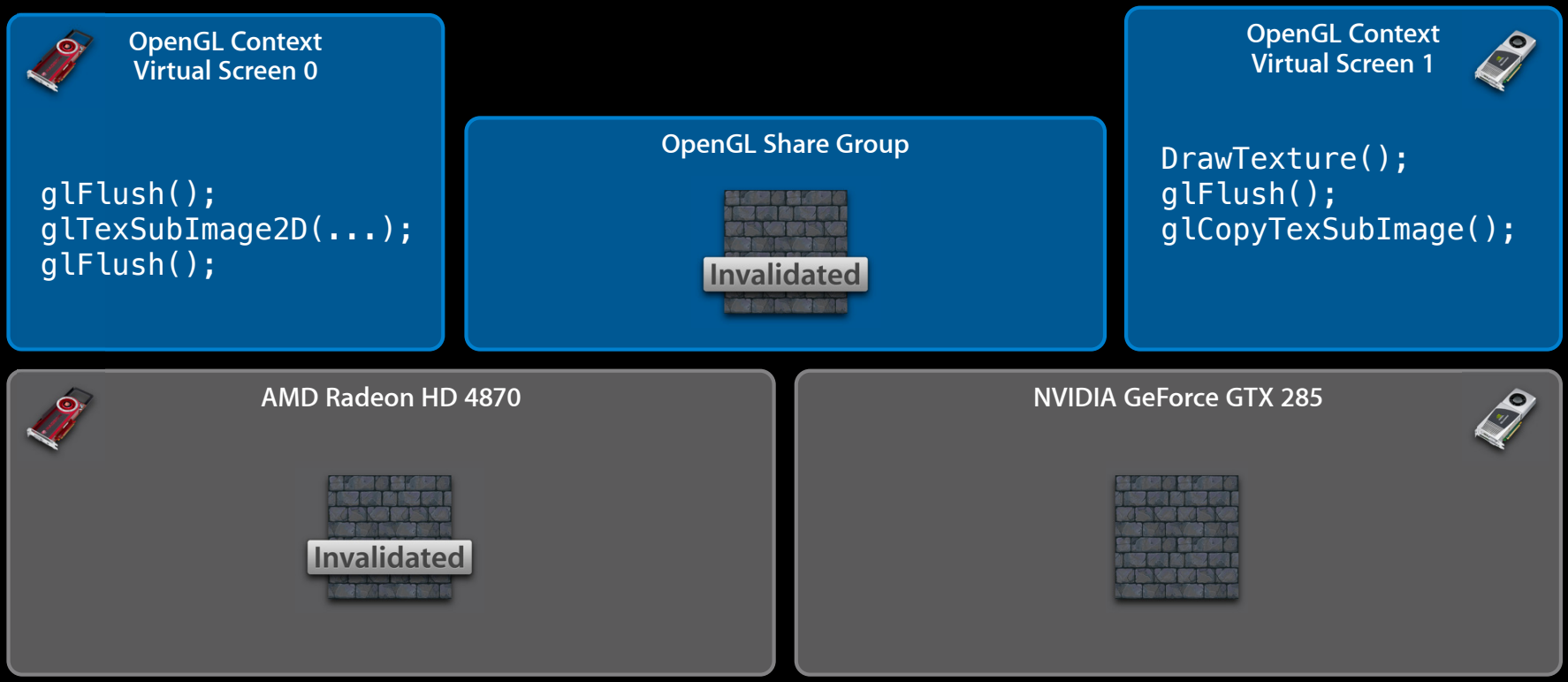
Advanced Multi-GPU Support

Multicontext resource management



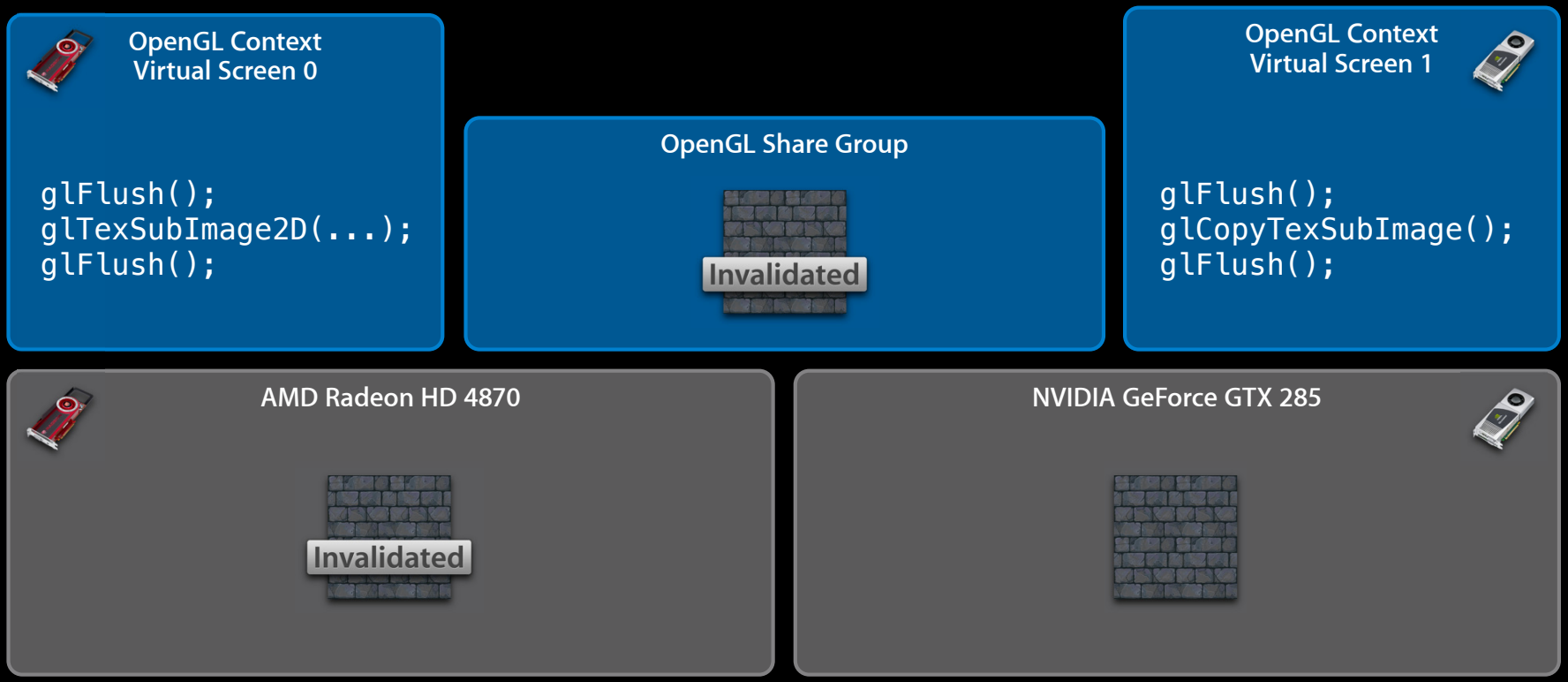
Advanced Multi-GPU Support

Multicontext resource management



Advanced Multi-GPU Support

Multicontext resource management



Advanced Multi-GPU Support

Performance tips

- Do enough work!
  - You must take into account fill/compute rate vs. transfer cost
  - Be mindful of 4x vs. 16x PCI-E slots in Mac Pro
- Decouple GPU workloads as much as possible
  - Don’t rely upon automatic resource synchronization for performance
  - Consider using extra buffering between GPUs to avoid stalls
  - Avoid display bottlenecks

Advanced Multi-GPU Demo

Abe Stephens
OpenCL Engineer

IOSurface

IOSurface

Resource sharing made easy

- High-level abstraction around shared memory
- Very efficient cross-process and cross-API data sharing
- Integrated directly into GPU software stack
- Hides details about moving data between CPU and GPUs

IOSurface

GPU integration

- OpenGL texture can be bound to an IOSurface
- Rendering is done by binding an IOSurface texture to an FBO
- Currently supported in OpenCL via OpenGL context sharing
- All textures bound to an IOSurface share the same video memory
- All GPUs share the same backing memory

IOSurface

OpenGL texture creation example

```
ioSurfaceBuffer = IOSurfaceCreate((CFDictionaryRef)
     [NSDictionary dictionaryWithObjectsAndKeys:
          [NSNumber numberWithInt:256],  (id)kIOSurfaceWidth,
          [NSNumber numberWithInt:256], (id)kIOSurfaceHeight,
          [NSNumber numberWithInt:4], (id)kIOSurfaceBytesPerElement,
          nil]);
glGenTextures(1, &name);
glBindTexture(GL_TEXTURE_RECTANGLE_EXT, name);
err = CGLTexImageIOSurface2D(
          cgl_ctx, GL_TEXTURE_RECTANGLE_EXT,
          GL_RGBA, /* Internal format */
          256, 256, /* width, height */
          GL_BGRA, GL_UNSIGNED_INT_8_8_8_8_REV, /* format/type */
          ioSurfaceBuffer, 0 /* plane */);
```

IOSurface

OpenGL texture creation example

```
ioSurfaceBuffer = IOSurfaceCreate((CFDictionaryRef)
     [NSDictionary dictionaryWithObjectsAndKeys:
          [NSNumber numberWithInt:256],  (id)kIOSurfaceWidth,
          [NSNumber numberWithInt:256], (id)kIOSurfaceHeight,
          [NSNumber numberWithInt:4], (id)kIOSurfaceBytesPerElement,
          nil]);
glGenTextures(1, &name);
glBindTexture(GL_TEXTURE_RECTANGLE_EXT, name);
err = CGLTexImageIOSurface2D(
          cgl_ctx, GL_TEXTURE_RECTANGLE_EXT,
          GL_RGBA, /* Internal format */
          256, 256, /* width, height */
          GL_BGRA, GL_UNSIGNED_INT_8_8_8_8_REV, /* format/type */
          ioSurfaceBuffer, 0 /* plane */);
```

IOSurface

OpenGL texture creation example

```
ioSurfaceBuffer = IOSurfaceCreate((CFDictionaryRef)
     [NSDictionary dictionaryWithObjectsAndKeys:
          [NSNumber numberWithInt:256],  (id)kIOSurfaceWidth,
          [NSNumber numberWithInt:256], (id)kIOSurfaceHeight,
          [NSNumber numberWithInt:4], (id)kIOSurfaceBytesPerElement,
          nil]);
glGenTextures(1, &name);
glBindTexture(GL_TEXTURE_RECTANGLE_EXT, name);
err = CGLTexImageIOSurface2D(
          cgl_ctx, GL_TEXTURE_RECTANGLE_EXT,
          GL_RGBA, /* Internal format */
          256, 256, /* width, height */
          GL_BGRA, GL_UNSIGNED_INT_8_8_8_8_REV, /* format/type */
          ioSurfaceBuffer, 0 /* plane */);
```

IOSurface

OpenGL texture creation example

```
ioSurfaceBuffer = IOSurfaceCreate((CFDictionaryRef)
     [NSDictionary dictionaryWithObjectsAndKeys:
          [NSNumber numberWithInt:256],  (id)kIOSurfaceWidth,
          [NSNumber numberWithInt:256], (id)kIOSurfaceHeight,
          [NSNumber numberWithInt:4], (id)kIOSurfaceBytesPerElement,
          nil]);
glGenTextures(1, &name);
glBindTexture(GL_TEXTURE_RECTANGLE_EXT, name);
err = CGLTexImageIOSurface2D(
          cgl_ctx, GL_TEXTURE_RECTANGLE_EXT,
          GL_RGBA, /* Internal format */
          256, 256, /* width, height */
          GL_BGRA, GL_UNSIGNED_INT_8_8_8_8_REV, /* format/type */
          ioSurfaceBuffer, 0 /* plane */);
```

IOSurface

Synchronization rules

- IOSurface follows the Mac OS X OpenGL flush/bind rule
- CPU writes must be completed before a bind

```
IOSurfaceLock(mySurface, 0, NULL);
/* Write to surface here */
IOSurfaceUnlock(mySurface, 0, NULL);

glBindTexture(GL_TEXTURE_RECTANGLE, mySurfaceTexName);
/* Do something with texture */
```

IOSurface

Synchronization rules

- IOSurface follows the Mac OS X OpenGL flush/bind rule
- CPU writes must be completed before a bind
- GPU flush must happen before any CPU reads or writes

```
/* Render to IOSurface */
glFlush();

IOSurfaceLock(mySurface, kIOSurfaceLockReadOnly, NULL);
/* Read from surface here */
IOSurfaceUnlock(mySurface, kIOSurfaceLockReadOnly, NULL);
```

IOSurface

Performance tips and tricks

- Automatic synchronization is easy, but not asynchronous
- Take advantage of IOSurfaceLock() to implement double buffering
  - Control over when data is written back to main memory
  - Shared backing means no CPU copies
- Use IOSurface to "cast" from one format/type to another
  - Example: Manipulate luminance data as quarter-width RGBA
  - Total data sizes much match

IOSurface

Example usage cases

- Application plug-ins
  - Common abstraction for CPU- or GPU-based plug-ins
- Client/server situations
  - Pass IOSurfaces efficiently between processes
  - Resources already on the GPU stay there
- Combine both
  - Run plug-ins in a different address space, on a different architecture, and even on a different GPU!

Demo: Muti-GPU with IOSurface

Kenneth Dyke
Sr. Mad Scientist

Summary

- Support systems with multiple GPUs whenever possible
- Take advantage of multiple GPUs when beneficial
- Use IOSurface to help
- Read the sample code!

More Information

Allan Schaffer
Graphics and Game Technologies Evangelist
aschaffer@apple.com

Apple Developer Forums
http://devforums.apple.com

Related Sessions

Harnessing OpenCL in Your Application	Russian Hill Wednesday 3:15PM
Maximizing OpenCL Performance	Russian Hill Wednesday 4:30PM
OpenGL for Mac OS X	Nob Hill Thursday 9:00AM

Labs

OpenCL Lab	Graphics and Media Lab C Thursday 9:00AM
OpenGL for Mac OS X Lab	Graphics and Media Lab C Thursday 2:00PM


WWDC10