Graphics and Games

#WWDC14

# Harnessing the Power of the Mac Pro with OpenGL and OpenCL

Session 601

Abe Stephens, PhD
GPU Software

# Mac Pro

---

# Graphics and Compute APIs

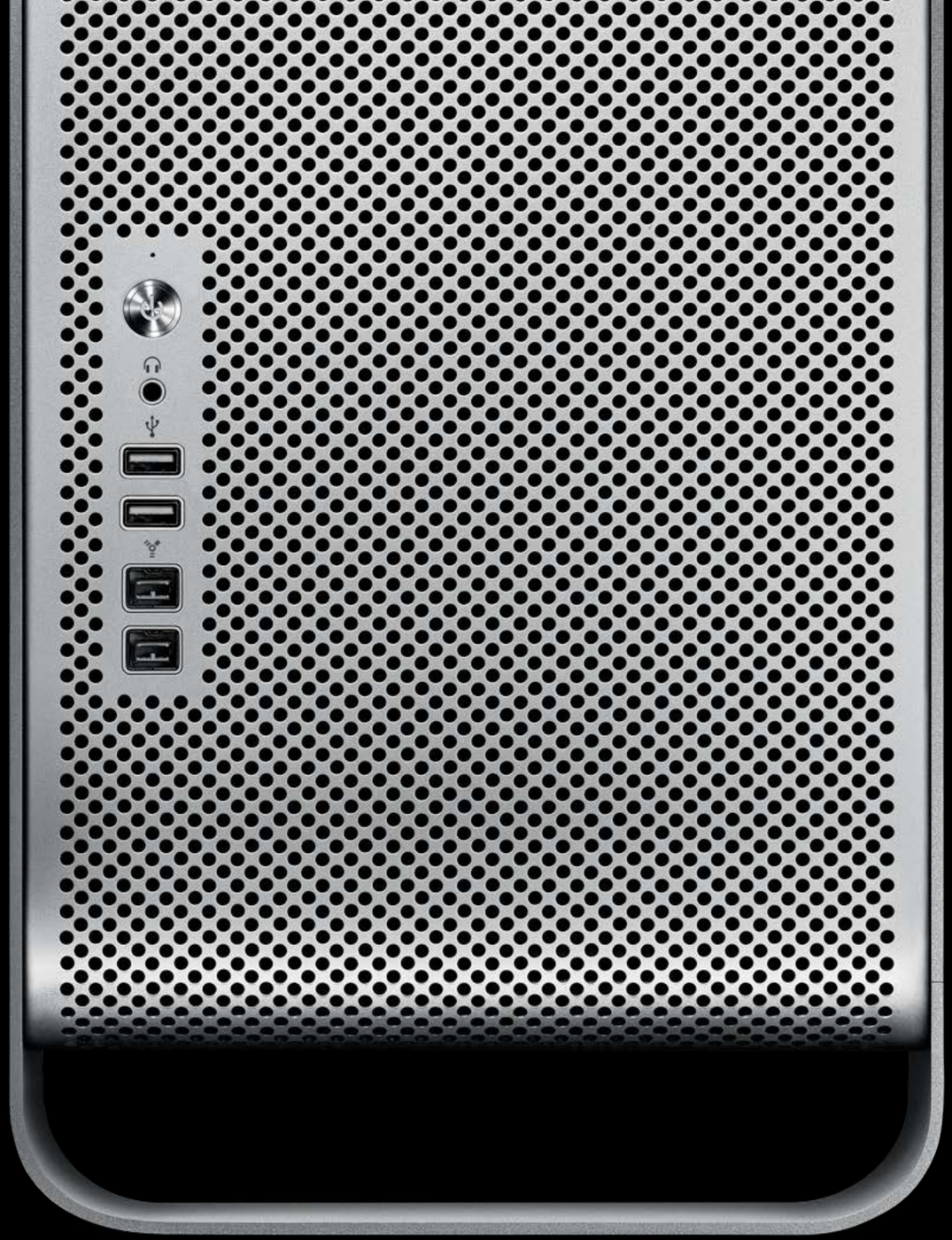
---

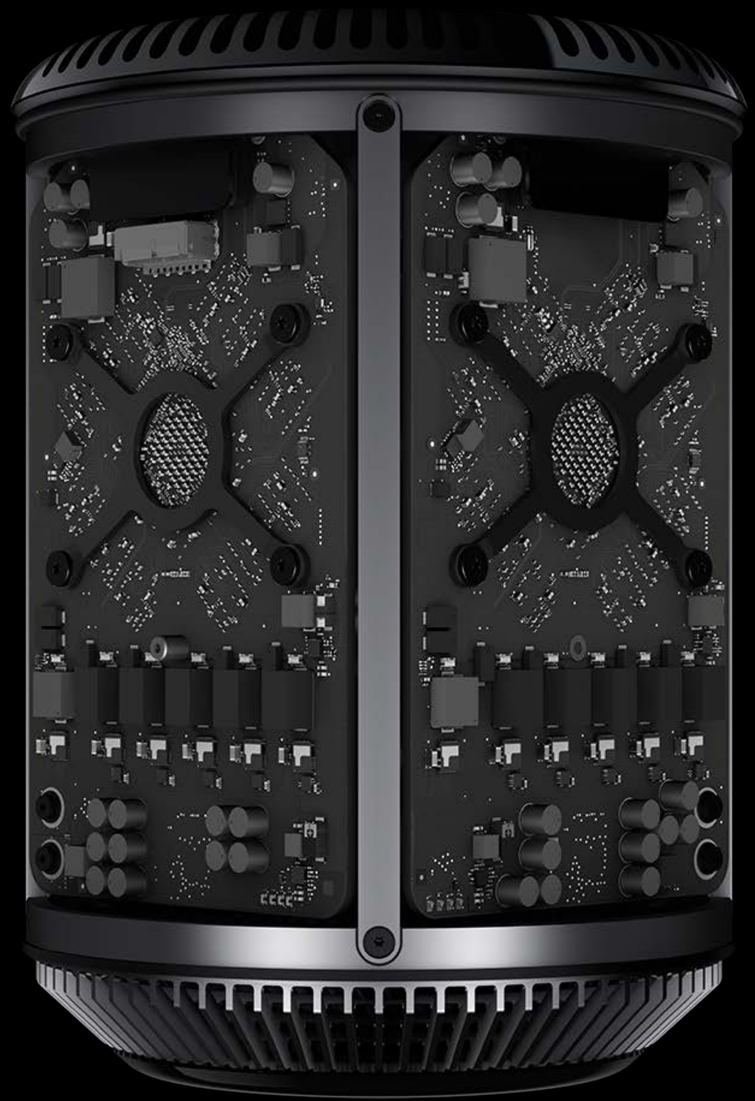
# Programming Patterns

# Mac Pro

## Graphics and Compute APIs

## Programming Patterns

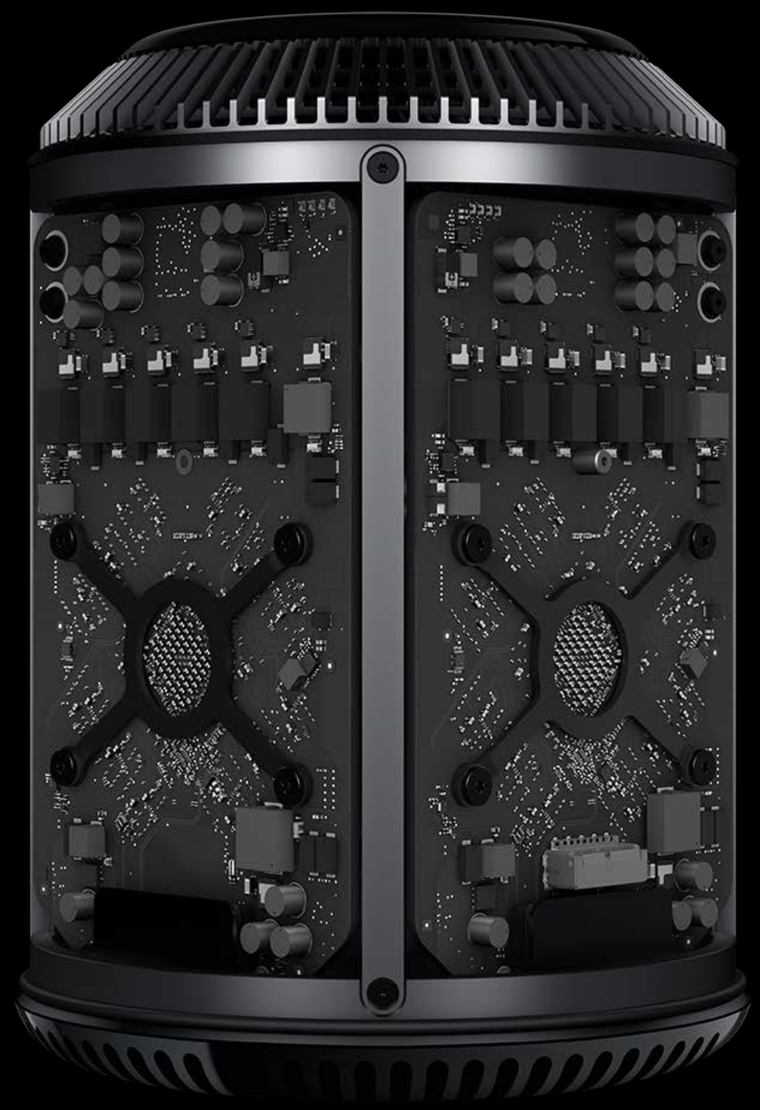




# Closer Look at the Mac Pro

Each GPU has:

- 2,048 stream processors
- 6GB of VRAM
- 3.5 teraflops peak



# Mac Pro

## Graphics and Compute APIs

## Programming Patterns

Mac Pro

# Graphics and Compute APIs

Programming Patterns

# Software Stack

# Software Stack

Application

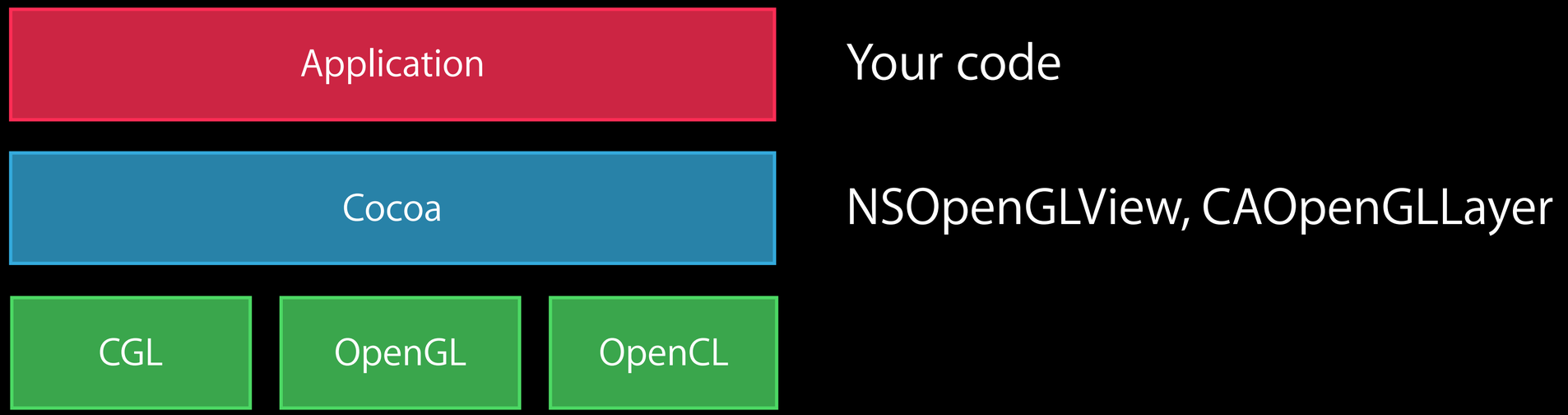
Your code

# Software Stack

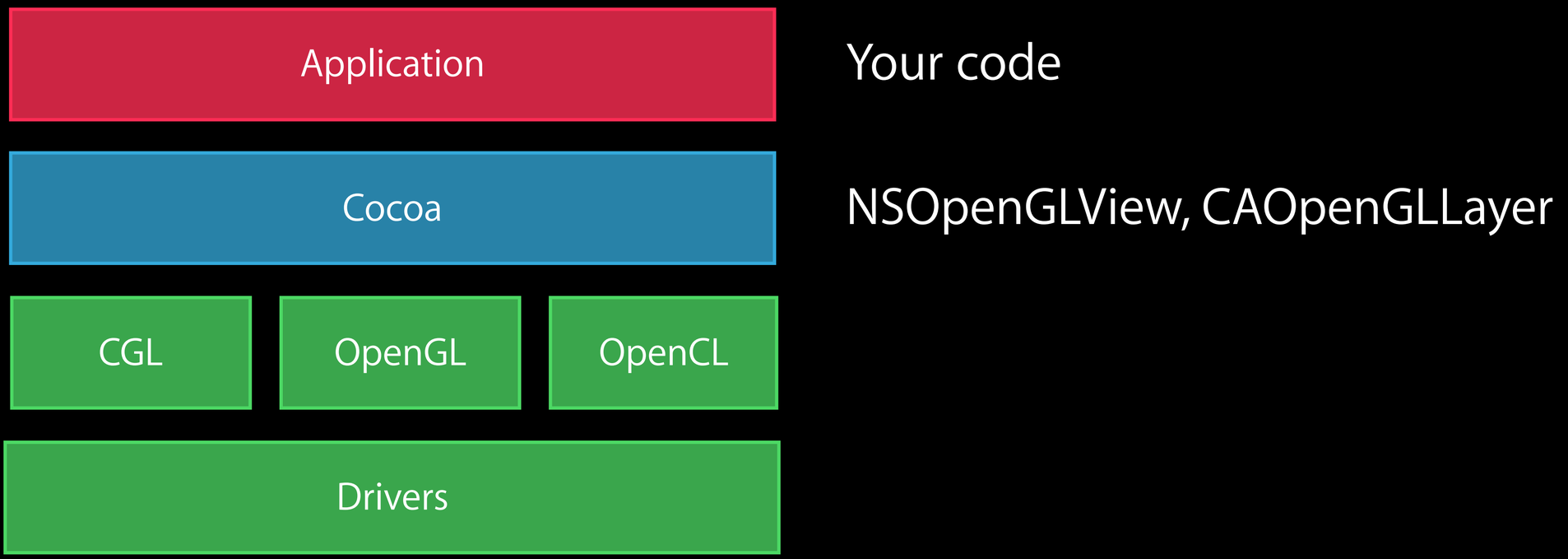

Your code

NSOpenGLView, CAOpenGLLayer

# Software Stack



# Software Stack



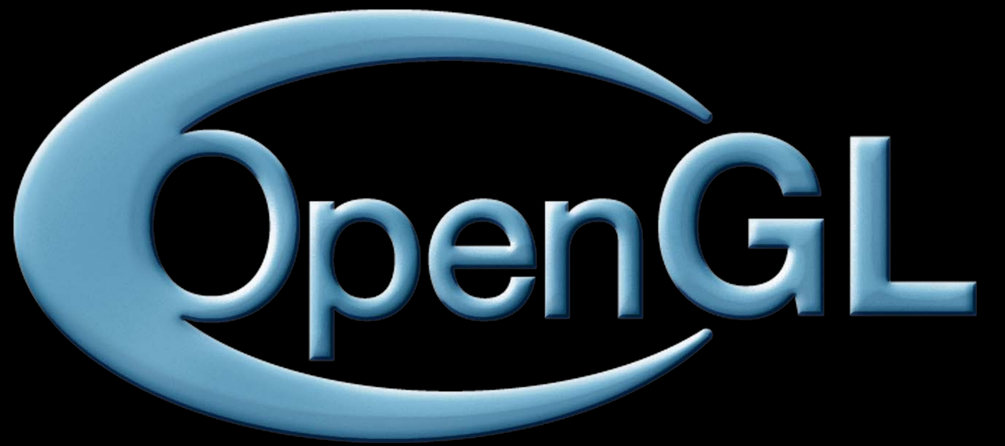
# Software Stack

# OpenGL for Graphics

GPU-accelerated 2D/3D API

OpenGL 4.1 Core Profile

GLSL 4.10



# OpenCL for Compute

Data parallel compute API

OpenCL 1.2

OpenCL C kernel language

Supports CPU and GPUs

Mac Pro GPU supports double precision

Queue priority flags

# What Can You Do?

Use the primary GPU for display

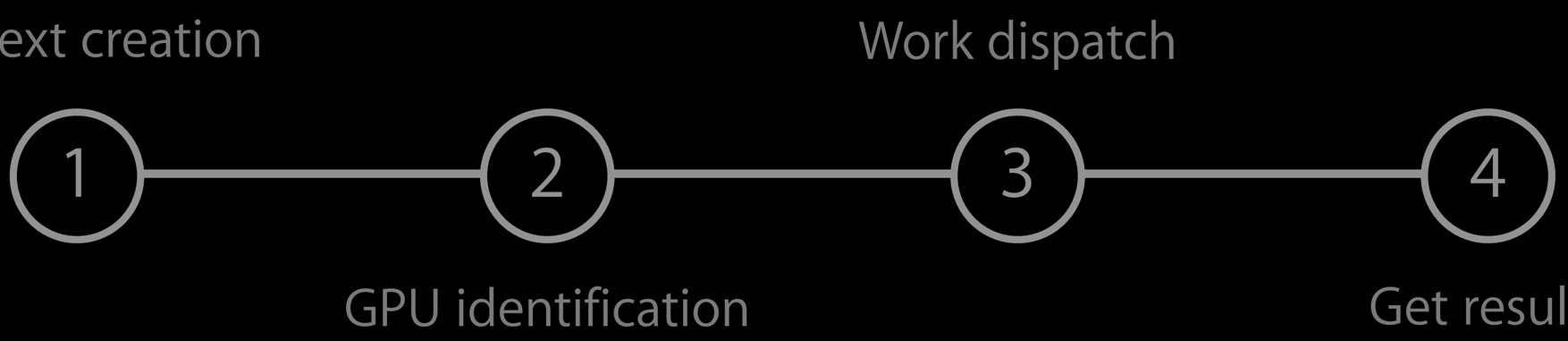
Add support for the secondary GPU

- OpenCL
- Off-screen rendering

# Setting Up for Dual GPUs

# Programming Steps

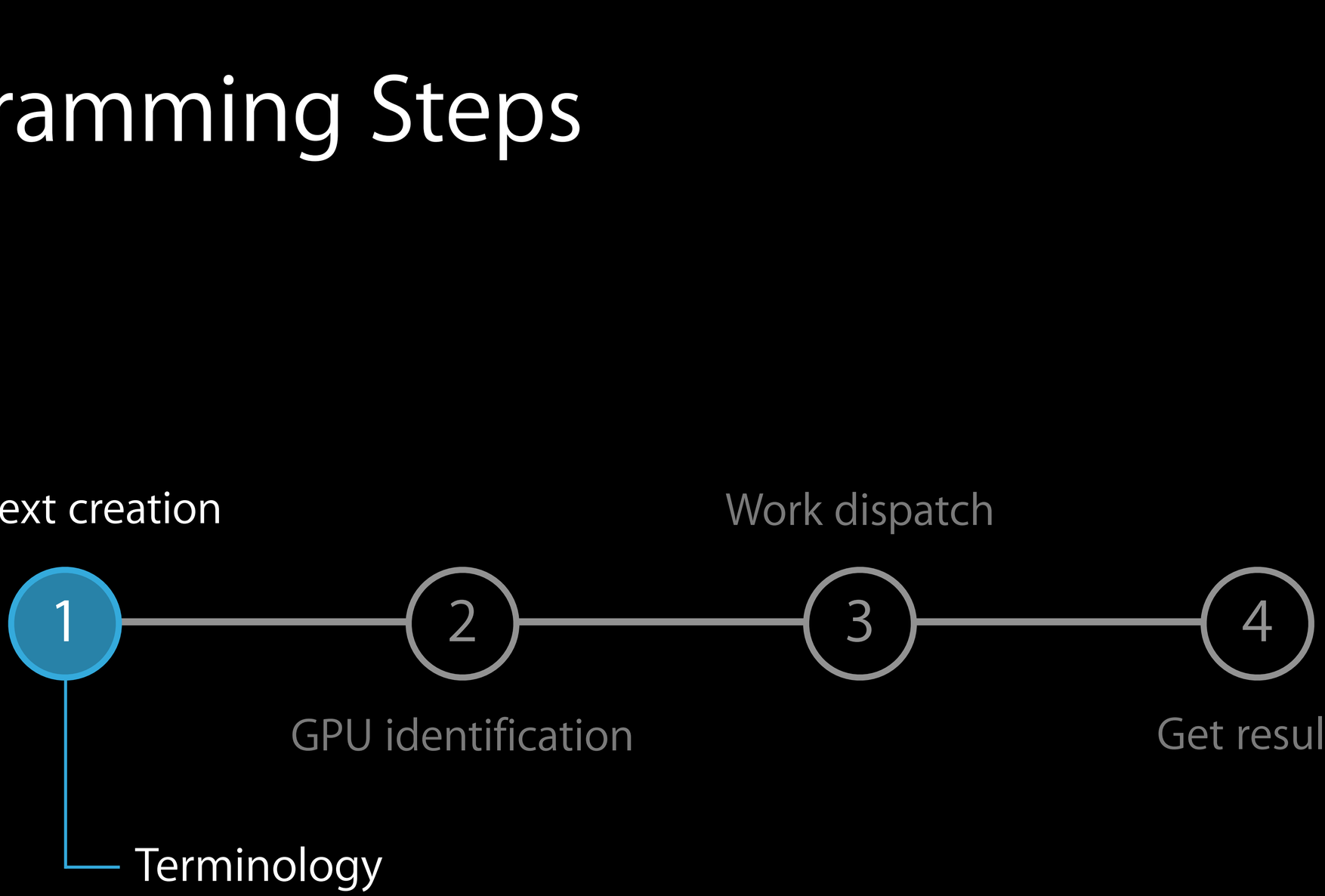
# Programming Steps



# Programming Steps

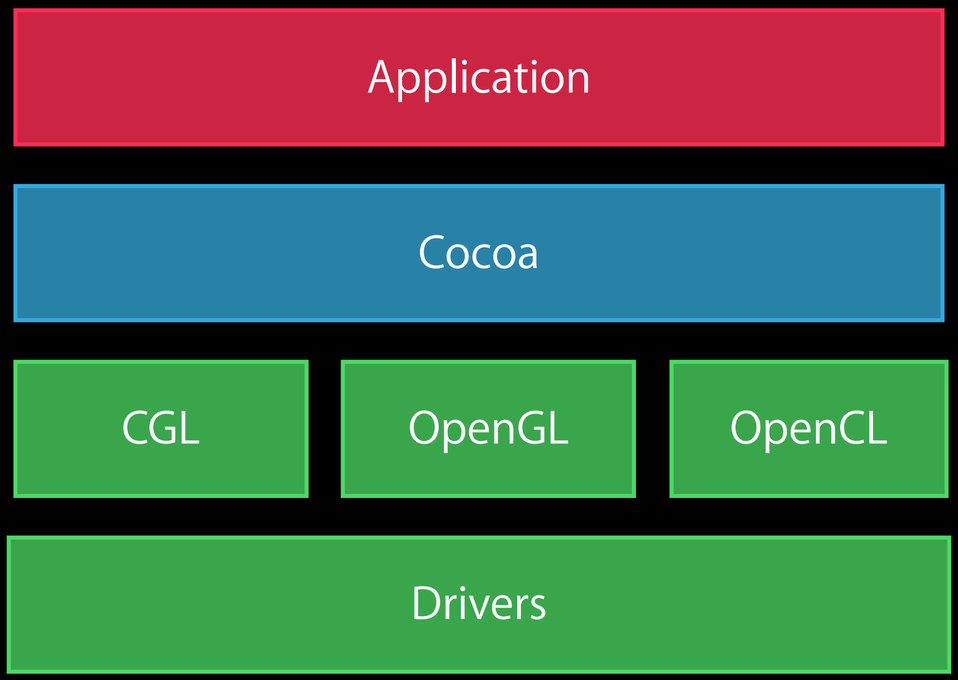
1. Context creation
2. GPU identification
3. Work dispatch
4. Get results

# Programming Steps



# Context Creation

## Software stack



# Context Creation

## Software stack

| | |
|---|---|
| OpenGL | Graphics API |
| CGL | Used to set up OpenGL contexts, select devices |
| OpenCL | Compute API, encompasses device selection |

# Context Creation

## OpenGL terminology

Renderer/Renderer ID

# Context Creation

## OpenGL terminology

# Context Creation

## OpenGL terminology

Pixel format attributes

Double Buffered
Offline Renderers
Core Profile

# Context Creation

## OpenGL terminology

Pixel format attributes

Renderer/Renderer ID


Software
Renderer

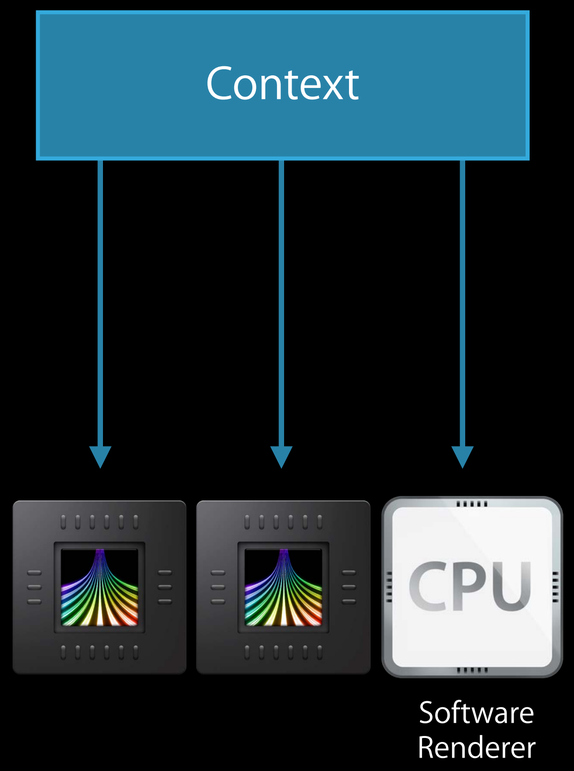
# Context Creation

## OpenGL terminology

Pixel format attributes

Renderer/Renderer ID

Context



# Context Creation

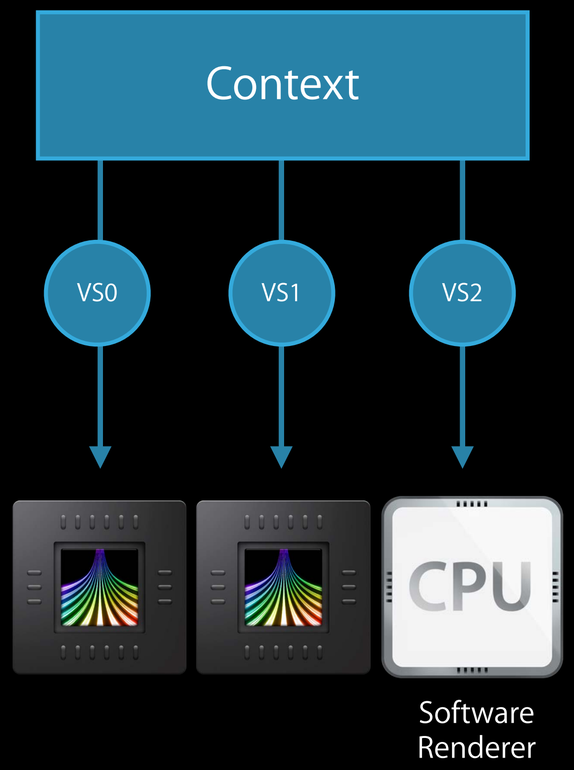
## OpenGL terminology

Pixel format attributes

Renderer/Renderer ID

Context

Virtual screen number



# Context Creation

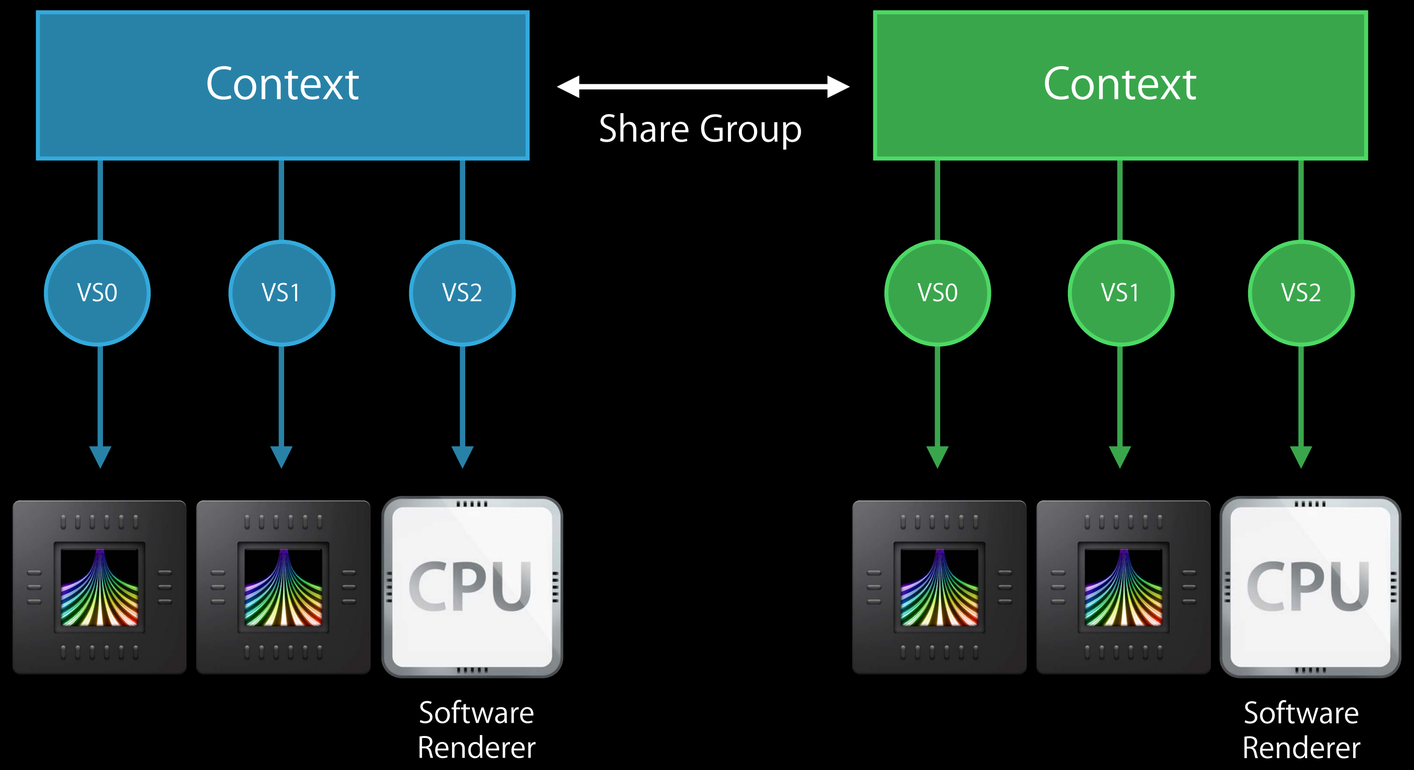
## OpenGL terminology

Pixel format attributes

Renderer/Renderer ID

Context

Virtual screen number



# Context Creation

## OpenCL terminology

```
cl_device_id

cl_context

cl_command_queue
```

# Context Creation

## OpenCL terminology

`cl_device_id` ← Renderer

`cl_context`

`cl_command_queue`

# Context Creation

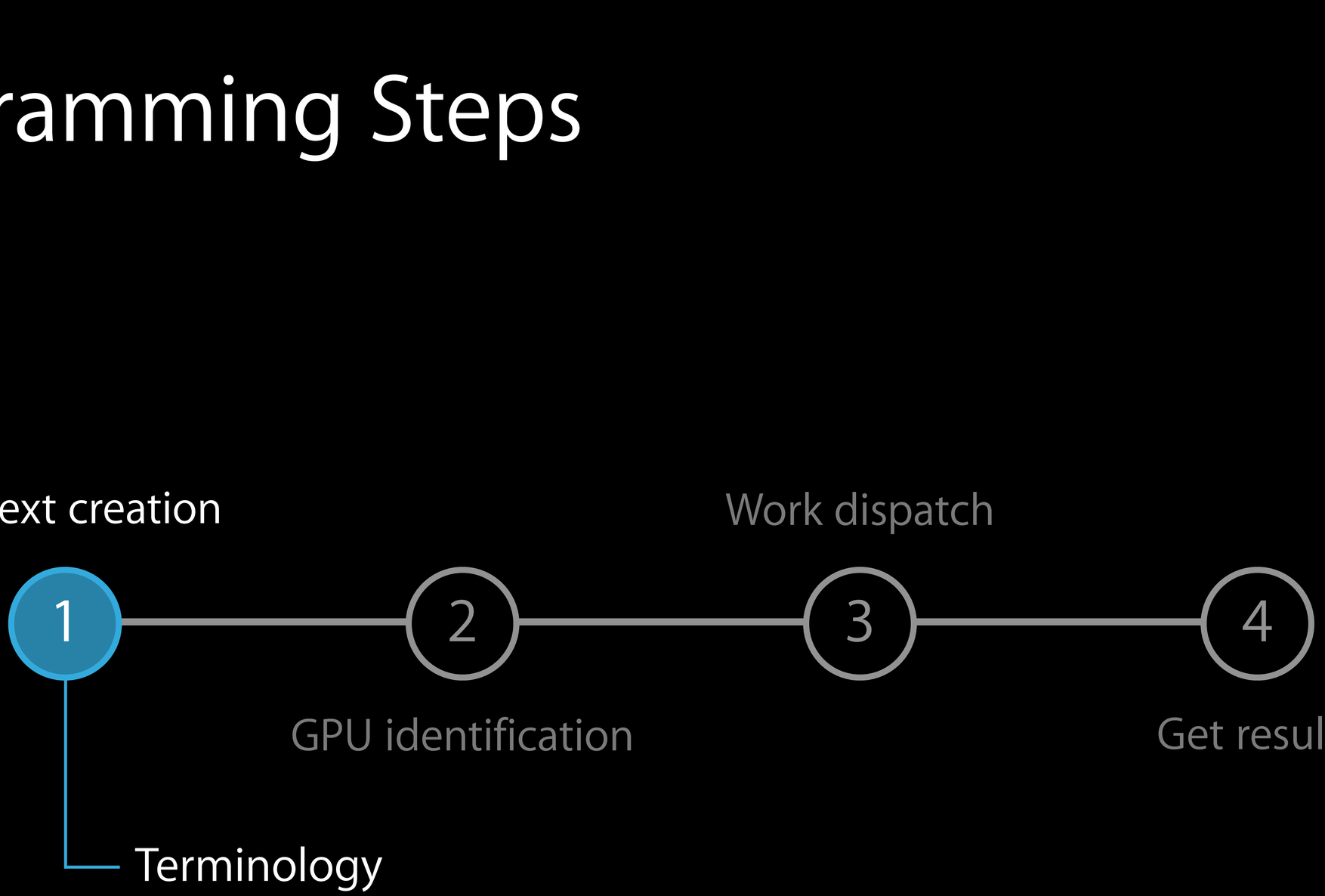
## OpenCL terminology

`cl_device_id` ← Renderer

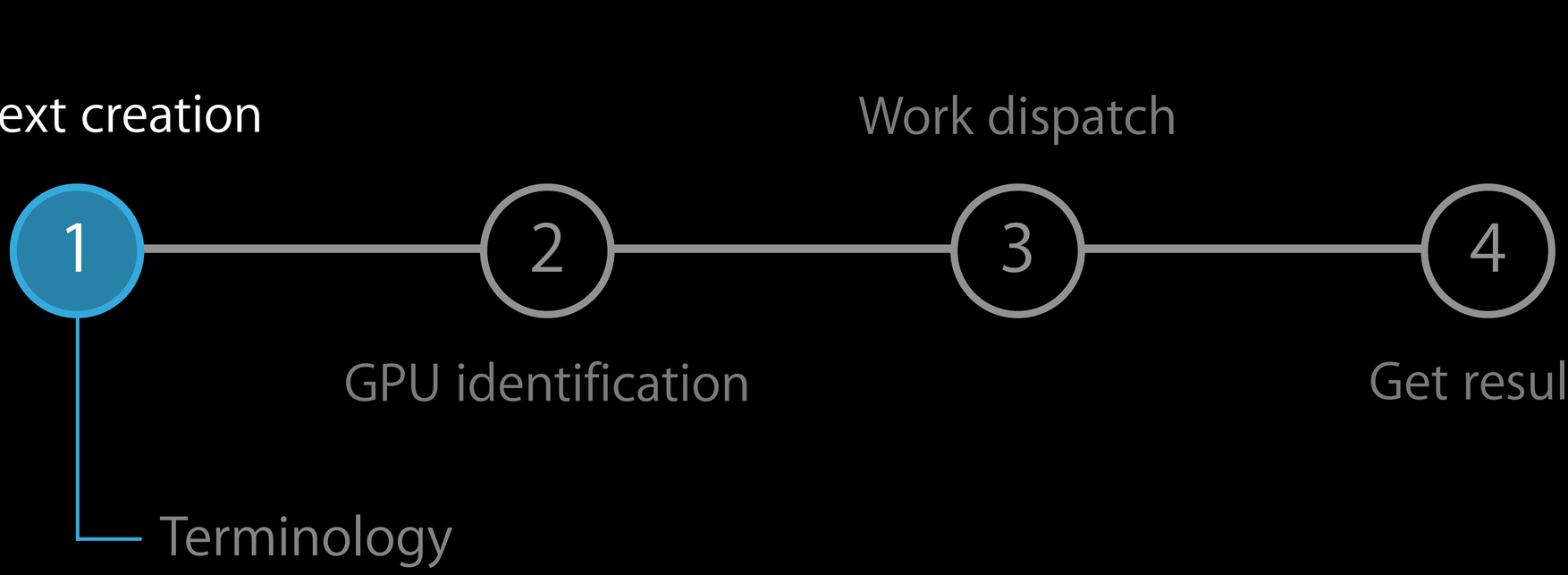
`cl_context` ← Share group

`cl_command_queue`

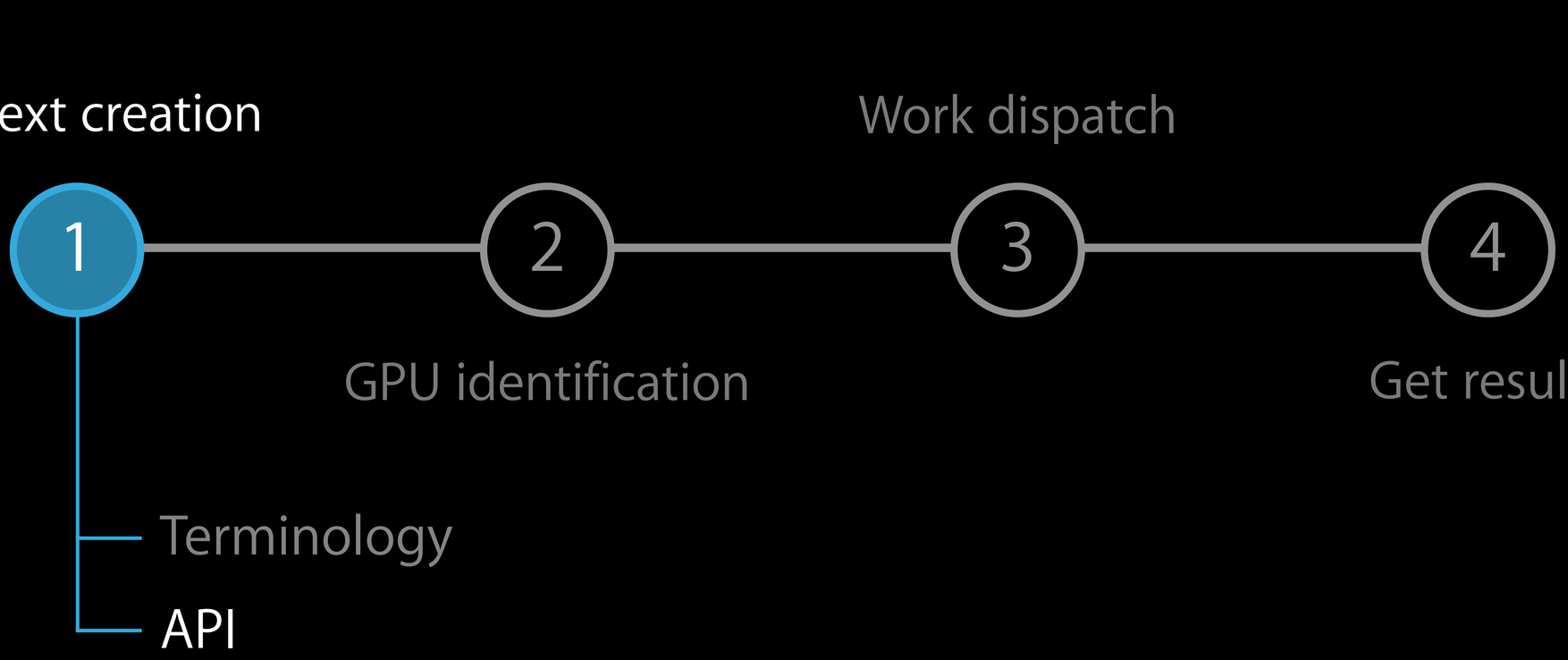
# Programming Steps



# Programming Steps



# Programming Steps



# Context Creation

## Using an NSOpenGLView

# Context Creation

## Using an NSOpenGLView

```
@implementation MyGLView // Derived from NSOpenGLView
```

# Context Creation

## Using an NSOpenGLView

```
@implementation MyGLView // Derived from NSOpenGLView

- (id) initWithFrame:(NSRect)frame {
```

# Context Creation

## Using an NSOpenGLView

```
@implementation MyGLView // Derived from NSOpenGLView

- (id) initWithFrame:(NSRect)frame {
  NSOpenGLPixelFormatAttribute attribs[] = {
    NSOpenGLPFAOpenGLProfile, NSOpenGLProfileVersion3_2Core,
    NSOpenGLPFADoubleBuffer,
    NSOpenGLPFAAllowOfflineRenderers,
    0 };
NSOpenGLPixelFormat* fmt = [NSOpenGLPixelFormat alloc] initWithAttributes:attribs];
```

# Context Creation

## Using an NSOpenGLView

```
@implementation MyGLView // Derived from NSOpenGLView

- (id) initWithFrame:(NSRect)frame {
  NSOpenGLPixelFormatAttribute attribs[] = {
    NSOpenGLPFAOpenGLProfile, NSOpenGLProfileVersion3_2Core,
    NSOpenGLPFADoubleBuffer,
    NSOpenGLPFAAllowOfflineRenderers,
    0 };
  NSOpenGLPixelFormat* fmt = [NSOpenGLPixelFormat alloc] initWithAttributes:attribs];

  self = [super initWithFrame:frame pixelFormat:fmt];
 ...
}
```

# Context Creation

## OpenCL

```
// Create a context with all GPUs
cl_context c = clCreateContextWithType(NULL,CL_DEVICE_TYPE_GPU,NULL,NULL,NULL);
```

# Context Creation

## CL/GL Sharing

```
CGLContextObj cgl_ctx = …;
CGLShareGroupObj sharegroup = CGLGetShareGroup(cgl_ctx);
cl_context_properties props[] = {
  CL_CONTEXT_PROPERTY_USE_CGL_SHAREGROUP_APPLE,
  (cl_context_properties)sharegroup,
  0
};
c = clCreateContext(props,0,NULL,NULL,NULL,NULL);
```

# Programming Steps

1. Context creation
2. GPU identification
3. Work dispatch
4. Get results

# Programming Steps

1. Context creation
2. GPU identification
3. Work dispatch
4. Get results

# GPU Identification

Want to determine which GPU is online and which is offline

Find its virtual screen or `cl_device_id`

The first virtual screen might not be the online display

# GPU Identification

## Finding the Offline Renderer ID

```
CGLRendererInfoObj rend;
GLint nrend = 0;
GLint secondaryGPURendererID = 0x0;
CGLQueryRendererInfo(0xffffffff, &rend, &nrend);
for(GLint i=0; i<nrend; ++i) {
  GLint online = 1;
  CGLDescribeRenderer(rend, i, kCGLRPOnline, &online);
  if(!online) {
    GLint accelerated = 0;
    CGLDescribeRenderer(rend, i, kCGLRPAcceleratedCompute, &accelerated);
    if(accelerated) {
      CGLDescribeRenderer(rend, i, kCGLRPRendererID,
                          &secondaryGPURendererID);
      break;
      …
```

# GPU Identification

## Finding the Offline Renderer ID

```
CGLRendererInfoObj rend;
GLint nrend = 0;
GLint secondaryGPURendererID = 0x0;
CGLQueryRendererInfo(0xffffffff, &rend, &nrend);
for(GLint i=0; i<nrend; ++i) {
  GLint online = 1;
  CGLDescribeRenderer(rend, i, kCGLRPOnline, &online);
  if(!online) {
    GLint accelerated = 0;
    CGLDescribeRenderer(rend, i, kCGLRPAcceleratedCompute, &accelerated);
    if(accelerated) {
      CGLDescribeRenderer(rend, i, kCGLRPRendererID,
                          &secondaryGPURendererID);
      break;
      …
```

# GPU Identification

## Finding the Offline Renderer ID

```
CGLRendererInfoObj rend;
GLint nrend = 0;
GLint secondaryGPURendererID = 0x0;
CGLQueryRendererInfo(0xffffffff, &rend, &nrend);
for(GLint i=0; i<nrend; ++i) {
  GLint online = 1;
  CGLDescribeRenderer(rend, i, kCGLRPOnline, &online);
  if(!online) {
    GLint accelerated = 0;
    CGLDescribeRenderer(rend, i, kCGLRPAcceleratedCompute, &accelerated);
    if(accelerated) {
      CGLDescribeRenderer(rend, i, kCGLRPRendererID,
                          &secondaryGPURendererID);
      break;
      …
```

# GPU Identification

## Finding the Offline Renderer ID

```
CGLRendererInfoObj rend;
GLint nrend = 0;
GLint secondaryGPURendererID = 0x0;
CGLQueryRendererInfo(0xffffffff, &rend, &nrend);
for(GLint i=0; i<nrend; ++i) {
  GLint online = 1;
  CGLDescribeRenderer(rend, i, kCGLRPOnline, &online);
  if(!online) {
    GLint accelerated = 0;
    CGLDescribeRenderer(rend, i, kCGLRPAcceleratedCompute, &accelerated);
    if(accelerated) {
      CGLDescribeRenderer(rend, i, kCGLRPRendererID,
                          &secondaryGPURendererID);
      break;
      …
```

# GPU Identification

## Finding the Offline Renderer ID

```
CGLRendererInfoObj rend;
GLint nrend = 0;
GLint secondaryGPURendererID = 0x0;
CGLQueryRendererInfo(0xffffffff, &rend, &nrend);
for(GLint i=0; i<nrend; ++i) {
  GLint online = 1;
  CGLDescribeRenderer(rend, i, kCGLRPOnline, &online);
  if(!online) {
    GLint accelerated = 0;
    CGLDescribeRenderer(rend, i, kCGLRPAcceleratedCompute, &accelerated);
    if(accelerated) {
      CGLDescribeRenderer(rend, i, kCGLRPRendererID,
                          &secondaryGPURendererID);
      break;
      …
```

# GPU Identification

## Renderer ID to virtual screen

```
CGLContextObj cgl_context = self.openGLContext.CGLContextObj;
CGLPixelFormatObj fmt     = self.pixelFormat.CGLPixelFormatObj;
GLint secondaryVirtualScreen = -1;

GLint count = 0;
CGLDescribePixelFormat(fmt,0,kCGLPFAVirtualScreenCount,&count);
for (GLint i=0;i!=count;++i) {
  CGLSetVirtualScreen(cgl_context, i);
  GLint r;
  CGLGetParameter(cgl_context, kCGLCPCurrentRendererID, &r);
  if (r == secondaryGPURendererID) {
    secondaryVirtualScreen = i; break;
  }
}
```

# GPU Identification

## Renderer ID to virtual screen

```
CGLContextObj cgl_context = self.openGLContext.CGLContextObj;
CGLPixelFormatObj fmt     = self.pixelFormat.CGLPixelFormatObj;
GLint secondaryVirtualScreen = -1;

GLint count = 0;
CGLDescribePixelFormat(fmt,0,kCGLPFAVirtualScreenCount,&count);
for (GLint i=0;i!=count;++i) {
  CGLSetVirtualScreen(cgl_context, i);
  GLint r;
  CGLGetParameter(cgl_context, kCGLCPCurrentRendererID, &r);
  if (r == secondaryGPURendererID) {
    secondaryVirtualScreen = i; break;
  }
}
```

# GPU Identification

## Renderer ID to virtual screen

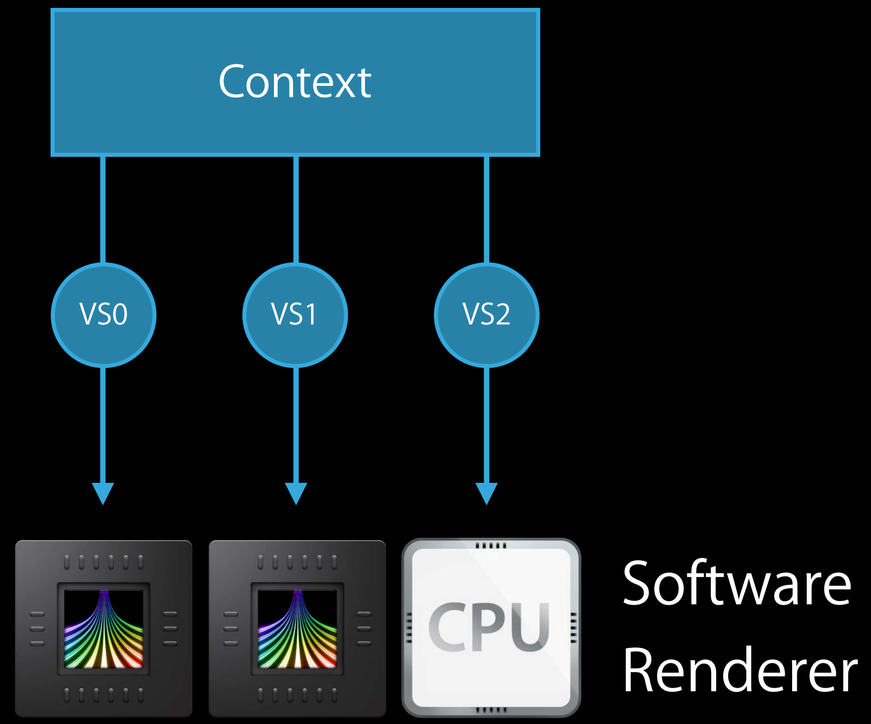
```
CGLContextObj cgl_context = self.openGLContext.CGLContextObj;
CGLPixelFormatObj fmt     = self.pixelFormat.CGLPixelFormatObj;
GLint secondaryVirtualScreen = -1;

GLint count = 0;
CGLDescribePixelFormat(fmt,0,kCGLPFAVirtualScreenCount,&count);
for (GLint i=0;i!=count;++i) {
  CGLSetVirtualScreen(cgl_context, i);
  GLint r;
  CGLGetParameter(cgl_context, kCGLCPCurrentRendererID, &r);
  if (r == secondaryGPURendererID) {
    secondaryVirtualScreen = i; break;
  }
}
```

# GPU Identification

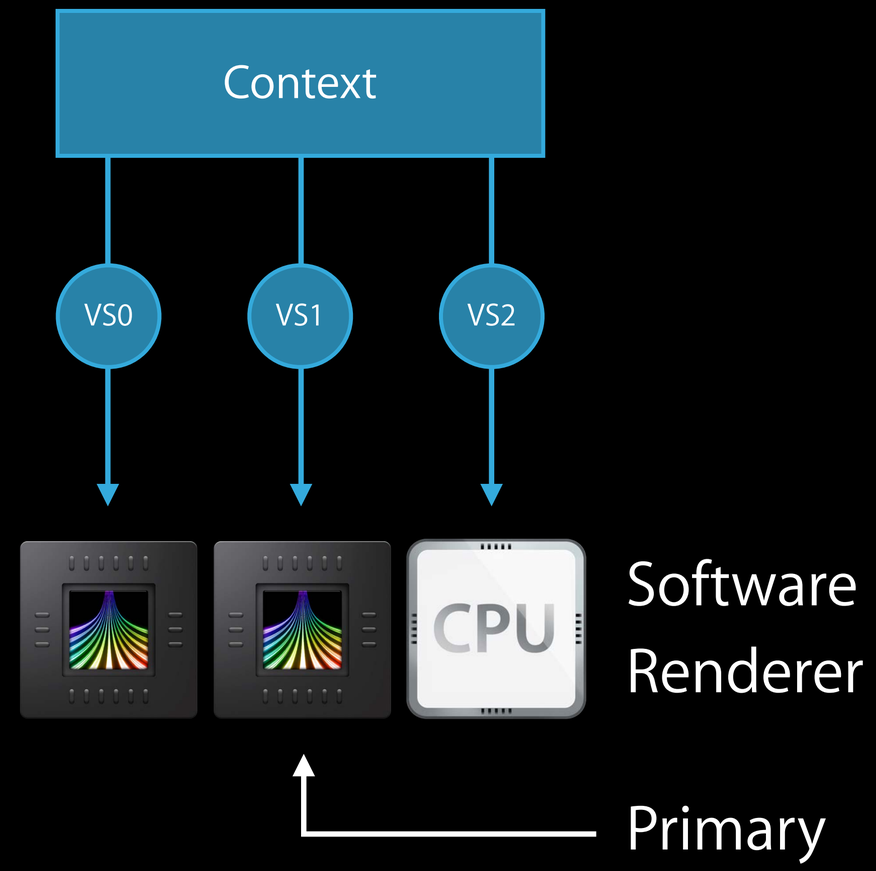
## Renderer ID to virtual screen

```
CGLContextObj cgl_context = self.openGLContext.CGLContextObj;
CGLPixelFormatObj fmt     = self.pixelFormat.CGLPixelFormatObj;
GLint secondaryVirtualScreen = -1;

GLint count = 0;
CGLDescribePixelFormat(fmt,0,kCGLPFAVirtualScreenCount,&count);
for (GLint i=0;i!=count;++i) {
  CGLSetVirtualScreen(cgl_context, i);
  GLint r;
  CGLGetParameter(cgl_context, kCGLCPCurrentRendererID, &r);
  if (r == secondaryGPURendererID) {
    secondaryVirtualScreen = i; break;
  }
}
```

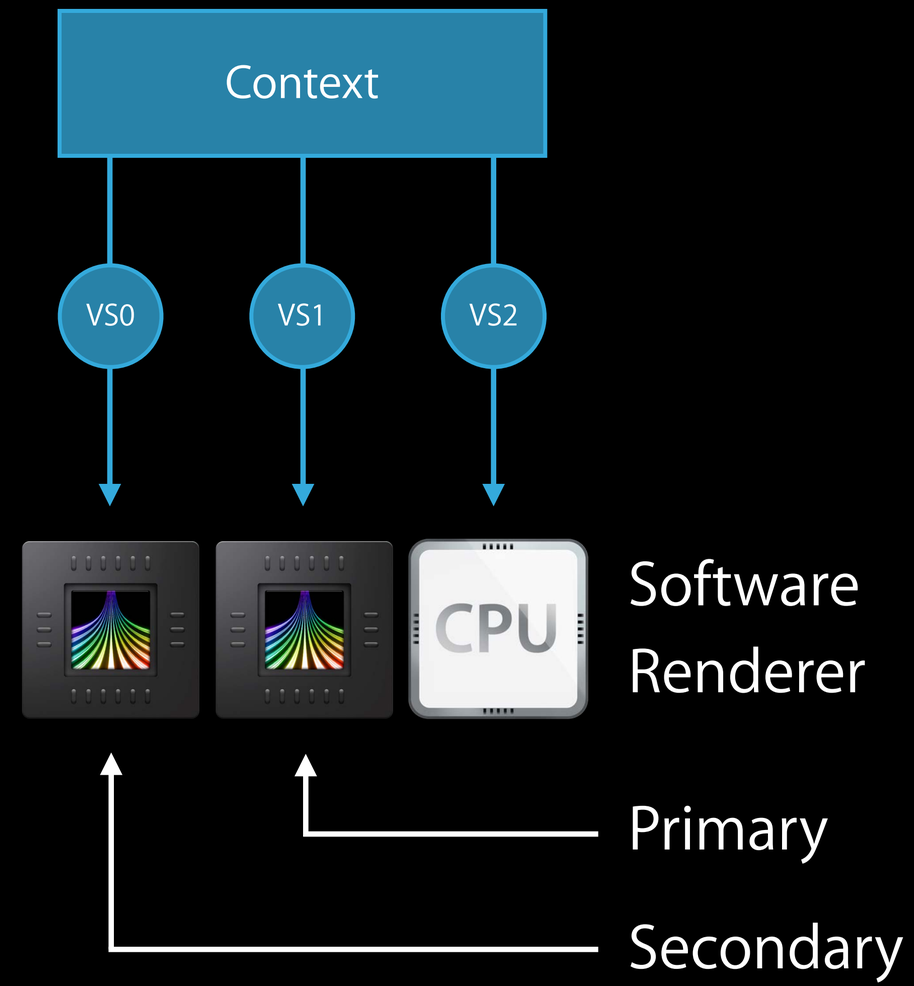
# Always Check Virtual Screen Numbers



# Always Check Virtual Screen Numbers



# Always Check Virtual Screen Numbers



# GPU Identification

## Renderer ID to cl_device_id

The `cl_device_id` is a static global value, not a context property

```
cl_device_id secondaryCLDeviceId =
  CGLGetDeviceFromGLRenderer(secondaryGPURendererID);

cl_command_queue q = clCreateCommandQueue(c,secondaryCLDeviceId,0,NULL);
```

# GPU Identification

## Renderer ID to cl_device_id

The `cl_device_id` is a static global value, not a context property

```
cl_device_id secondaryCLDeviceId =
  CGLGetDeviceFromGLRenderer(secondaryGPURendererID);

cl_command_queue q = clCreateCommandQueue(c,secondaryCLDeviceId,0,NULL);
```

# Programming Steps

1. Context creation
2. GPU identification
3. Work dispatch
4. Get results

# Programming Steps

1. Context creation
2. GPU identification
3. Work dispatch
4. Get results

# Work Dispatch

## OpenGL

```
CGLSetCurrentContext(self.view.openGLContext.CGLContextObj);
CGLSetVirtualScreen(secondaryVirtualScreen);

glBindVertexArray(…);
glDrawElements(…);
```

# Work Dispatch

## OpenCL

```
cl_command_queue q = clCreateCommandQueue(c,secondaryCLDeviceId,0,NULL);

clSetKernelArg(k,0,…);
clSetKernelArg(k,1,…);
clEnqueueNDRangeKernel(q,k,…);
```

# Programming Steps

1. Context creation
2. GPU identification
3. Work dispatch
4. Get results

# Programming Steps

Context creation

1

2

GPU identification

3

Work dispatch

4

Get results

# Get Results

## CL/GL sharing

Runtime will copy data automatically

Must follow certain rules

# Get Results

## Switching GPUs

# Get Results

## Switching GPUs

```
CGLSetVirtualScreen(secondary);
```

# Get Results

## Switching GPUs

```
CGLSetVirtualScreen(secondary);
…
glBindTexture(…);
glDrawElements(…);
glFlushRenderAPPLE();
```

# Get Results

## Switching GPUs

```
CGLSetVirtualScreen(secondary);
…
glBindTexture(…);
glDrawElements(…);
glFlushRenderAPPLE();
…
CGLSetVirtualScreen(primary);
```

# Get Results

## Switching GPUs

```
CGLSetVirtualScreen(secondary);
…
glBindTexture(…);
glDrawElements(…);
glFlushRenderAPPLE();
```

```
…
CGLSetVirtualScreen(primary);
…
glDrawElements(…);
…
```

Primary GPU

Secondary GPU

Primary GPU

Secondary GPU

glFlushRenderAPPLE

Primary GPU

Secondary GPU

CGLSetVirtualScreen

glFlushRenderAPPLE

Primary GPU

Secondary GPU

CGLSetVirtualScreen

glFlushRenderAPPLE

Host Engine Copy

Page Off

Page On

Primary GPU

Secondary GPU

CGLSetVirtualScreen

glFlushRenderAPPLE

Host Engine Copy

Page Off

Page On

glDrawElements

# Get Results

## Switching GPUs

# Get Results

## Switching GPUs

```
cl_command_queue primary_q = …
cl_command_queue secondary_q = …
```

# Get Results

## Switching GPUs

```
cl_command_queue primary_q = …
cl_command_queue secondary_q = …

clEnqueueNDRangeKernel(primary_q, k0, … );
```

# Get Results

## Switching GPUs

```
cl_command_queue primary_q = …
cl_command_queue secondary_q = …

clEnqueueNDRangeKernel(primary_q, k0, … );
clFlush(primary_q);
```

# Get Results

## Switching GPUs

```
cl_command_queue primary_q = …
cl_command_queue secondary_q = …

clEnqueueNDRangeKernel(primary_q, k0, … );
clFlush(primary_q);

clEnqueueNDRangeKernel(secondary_q, k1, …);
…
```

# Get Results

## Flush and bind

# Get Results

## Flush and bind

```
clEnqueueNDRangeKernel(secondary_q, k0, … );
```

# Get Results

## Flush and bind

```
clEnqueueNDRangeKernel(secondary_q, k0, … );
clFlush(primary_q);
…
```

# Get Results

## Flush and bind

```
clEnqueueNDRangeKernel(secondary_q, k0, … );
clFlush(primary_q);
…

CGLSetVirtualScreen(secondary);
```

# Get Results

## Flush and bind

```
clEnqueueNDRangeKernel(secondary_q, k0, … );
clFlush(primary_q);
…

CGLSetVirtualScreen(secondary);
…
glBindBuffer(…);
glDrawElements(…);
…
```

# Contexts Should Include All GPUs

Mac Pro

# Graphics and Compute APIs

Programming Patterns

Mac Pro

Graphics and Compute APIs

# Programming Patterns

# Offline Tasks

# Offline Tasks

Apply the operation once

# Offline Tasks

Apply the operation once

Takes longer than a frame

# Offline Tasks

Apply the operation once

Takes longer than a frame

Results saved to main memory or disk

# Offline Tasks

Apply the operation once

Takes longer than a frame

Results saved to main memory or disk

Could be done off the main thread

Primary GPU

Secondary GPU

Primary GPU

Secondary GPU

clEnqueueNDRangeKernel

clEnqueueNDRangeKernel

…

clFlush

glBindBuffer

glDrawElements

…

glFlushRenderAPPLE

Primary GPU

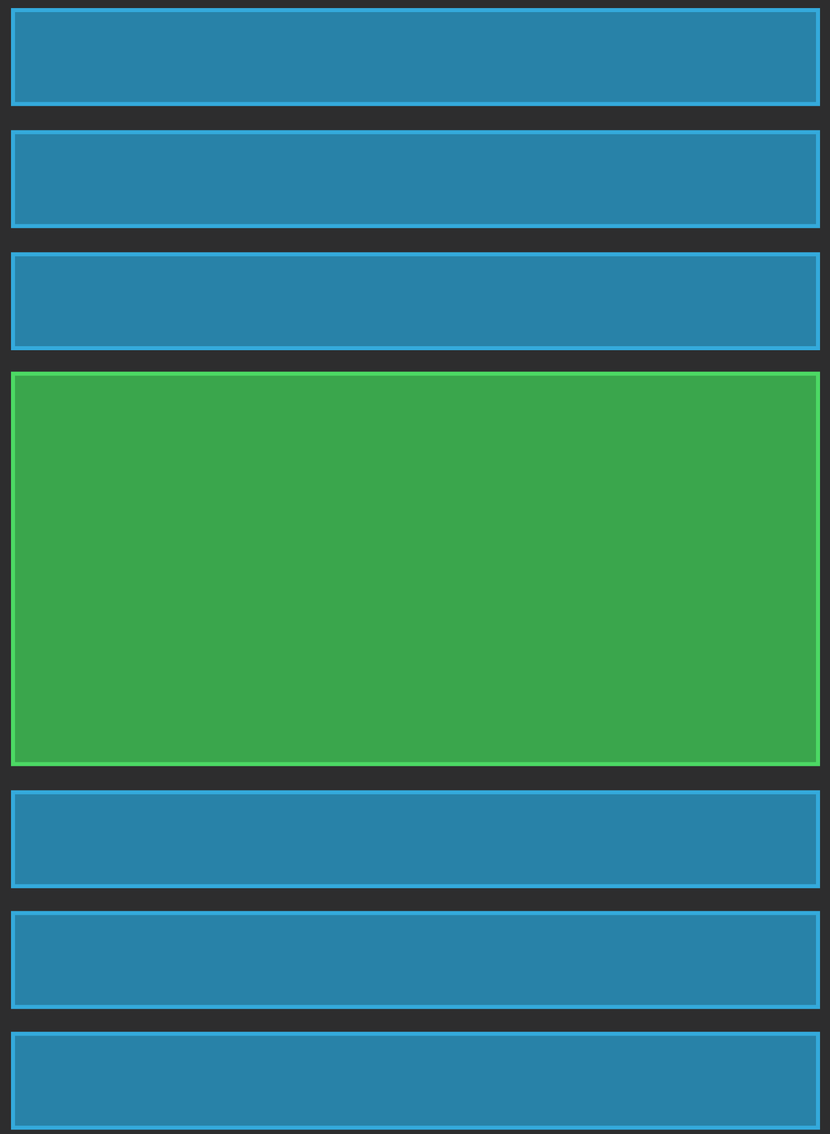
Secondary GPU

Primary GPU

Secondary GPU

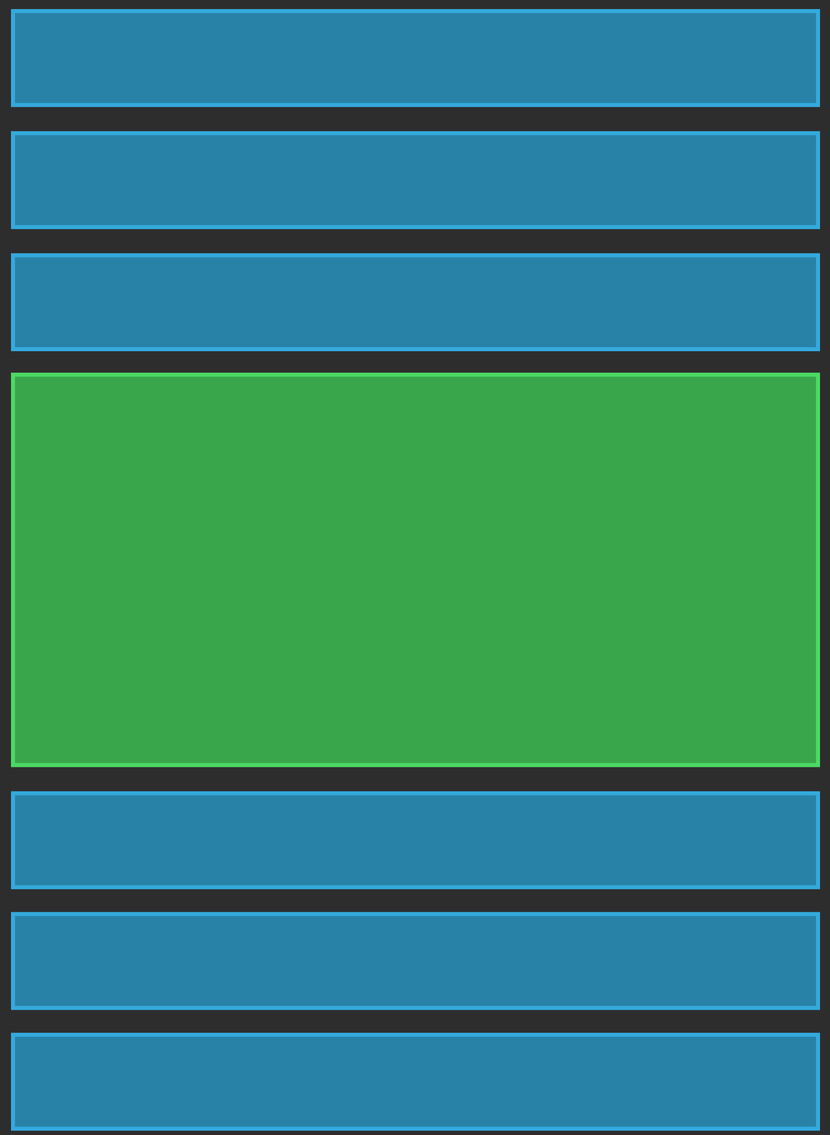
Primary GPU

Secondary GPU



Primary GPU

Secondary GPU



Primary GPU

Secondary GPU

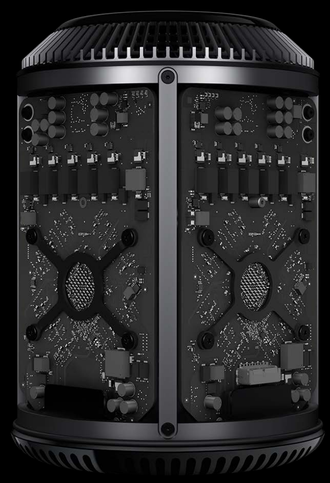
```
-(IBAction)applyEffect:(id)sender
{
  // Apply the effect filter to the image
  clSetKernelArg(…);
  clSetKernelArg(…);

  for (unsigned i=0;i!=numIters;++i) {
    clSetKernelArg(…, &rows[i]);
    clEnqueueNDRangeKernel(qPrimary, …);
    // ...
  }

  // Redraw the NSOpenGLView
  [self.view setNeedsDisplay:YES];
}
```

```
-(IBAction)applyEffect:(id)sender
{
  // Apply the effect filter to the image
  clSetKernelArg(…);
  clSetKernelArg(…);

  for (unsigned i=0;i!=numIters;++i) {
    clSetKernelArg(…, &rows[i]);
    clEnqueueNDRangeKernel(qSecondary, …);
    // ...
  }

  // Redraw the NSOpenGLView
  [self.view setNeedsDisplay:YES];
}
```

# Graphics on Both GPUs

Divide work between GPUs

Window server will copy from secondary



App Thread

Primary GPU

Secondary GPU

## App Thread

```
CGLSetCurrentContext(ctx)
CGLSetVirtualScreen(primary)
drawScene()
glFlushRenderAPPLE
```

## Primary GPU

glBindVertexArray

glDrawElements

glDrawElements

glFlushRenderAPPLE

flushBuffer

## Secondary GPU

App Thread

```
CGLSetCurrentContext(ctx)
CGLSetVirtualScreen(primary)
drawScene()
glFlushRenderAPPLE
```

```
CGLSetCurrentContext(ctx)
CGLSetVirtualScreen(secondary)
drawScene()
glFlushRenderAPPLE
```

Primary GPU

- glBindVertexArray
- glDrawElements
- glDrawElements
- glFlushRenderAPPLE
- flushBuffer

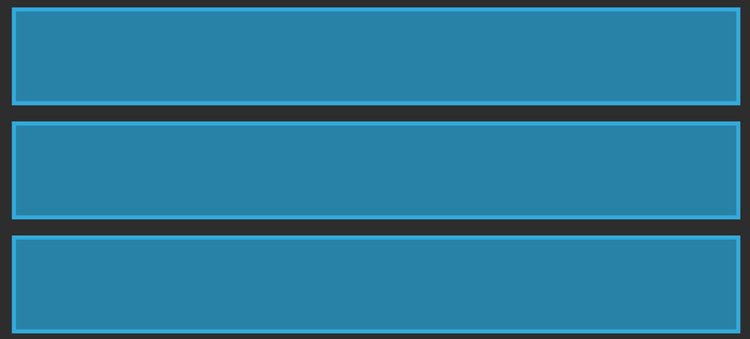
Secondary GPU

- glBindVertexArray
- glDrawElements
- glDrawElements
- glFlushRenderAPPLE
- flushBuffer

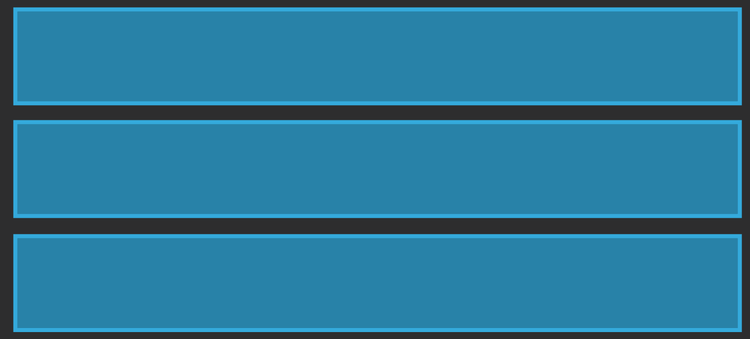
Primary GPU

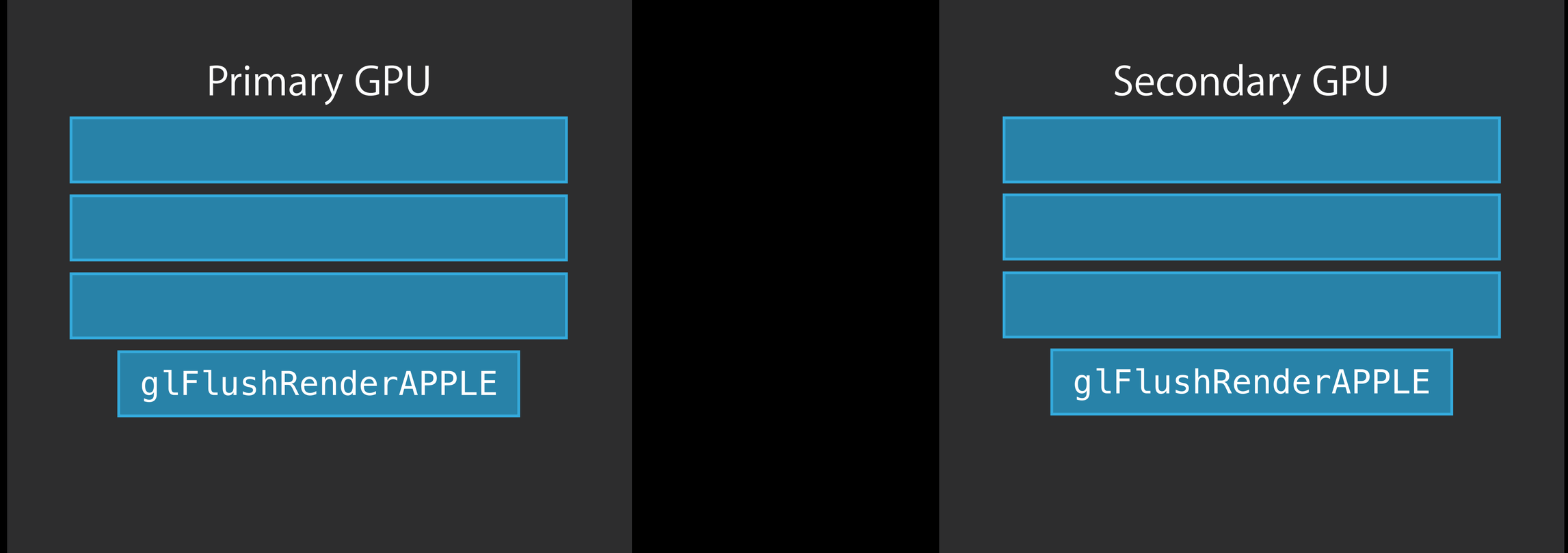
Secondary GPU

Primary GPU



Secondary GPU




Primary GPU
glFlushRenderAPPLE
Secondary GPU
glFlushRenderAPPLE

Primary GPU

glFlushRenderAPPLE

flushBuffer

Secondary GPU

glFlushRenderAPPLE

flushBuffer

Primary GPU

glFlushRenderAPPLE

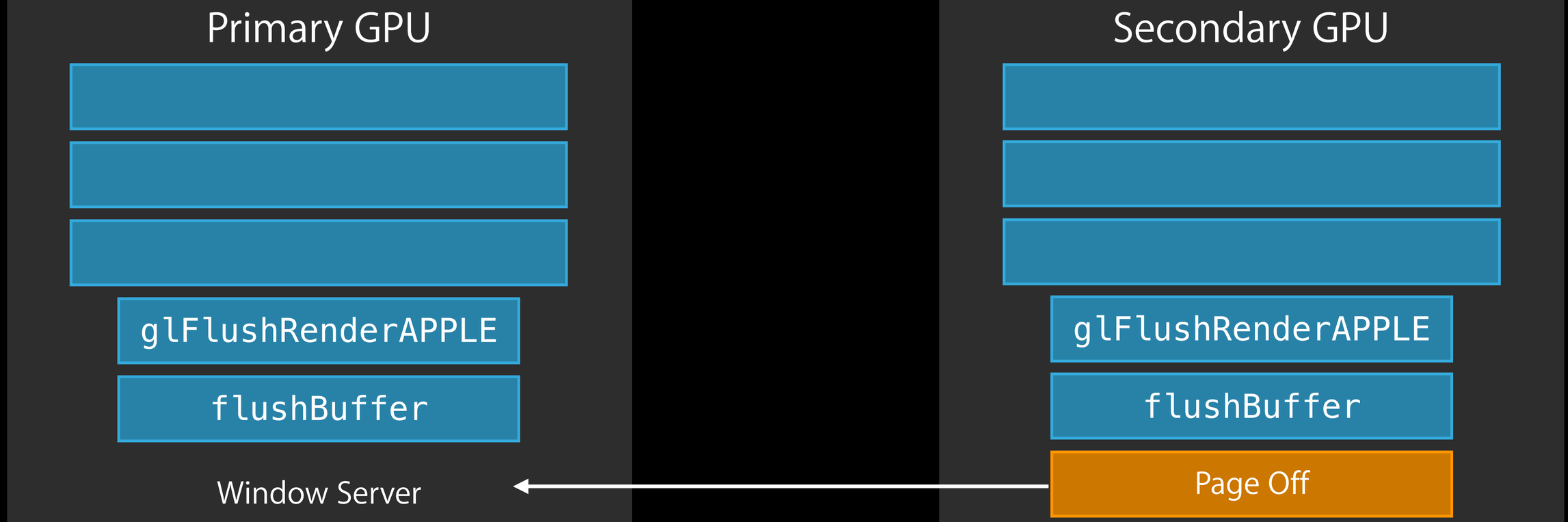
flushBuffer

Window Server

Secondary GPU

glFlushRenderAPPLE

flushBuffer


Primary GPU
glFlushRenderAPPLE
flushBuffer
Window Server
Secondary GPU
glFlushRenderAPPLE
flushBuffer
Page Off

Primary GPU

Secondary GPU

glFlushRenderAPPLE

flushBuffer

glFlushRenderAPPLE

flushBuffer

Page Off

Window Server

Page On

Composite

Primary GPU

Secondary GPU

glFlushRenderAPPLE

flushBuffer

glFlushRenderAPPLE

flushBuffer

Page Off

Window Server

Page On

Composite

Displayed

Primary GPU

Secondary GPU

glFlushRenderAPPLE

flushBuffer

glFlushRenderAPPLE

flushBuffer

Page Off

Window Server

Page On

Composite

Displayed

Primary GPU

glFlushRenderAPPLE

flushBuffer

Window Server

Page On

Composite

Displayed

Secondary GPU

glFlushRenderAPPLE

flushBuffer

Page Off

# Design Challenge

# Design Challenge

Have to divide the work

# Design Challenge

Have to divide the work

Display connected to primary GPU only

# Handling Other Multi-GPU Situations

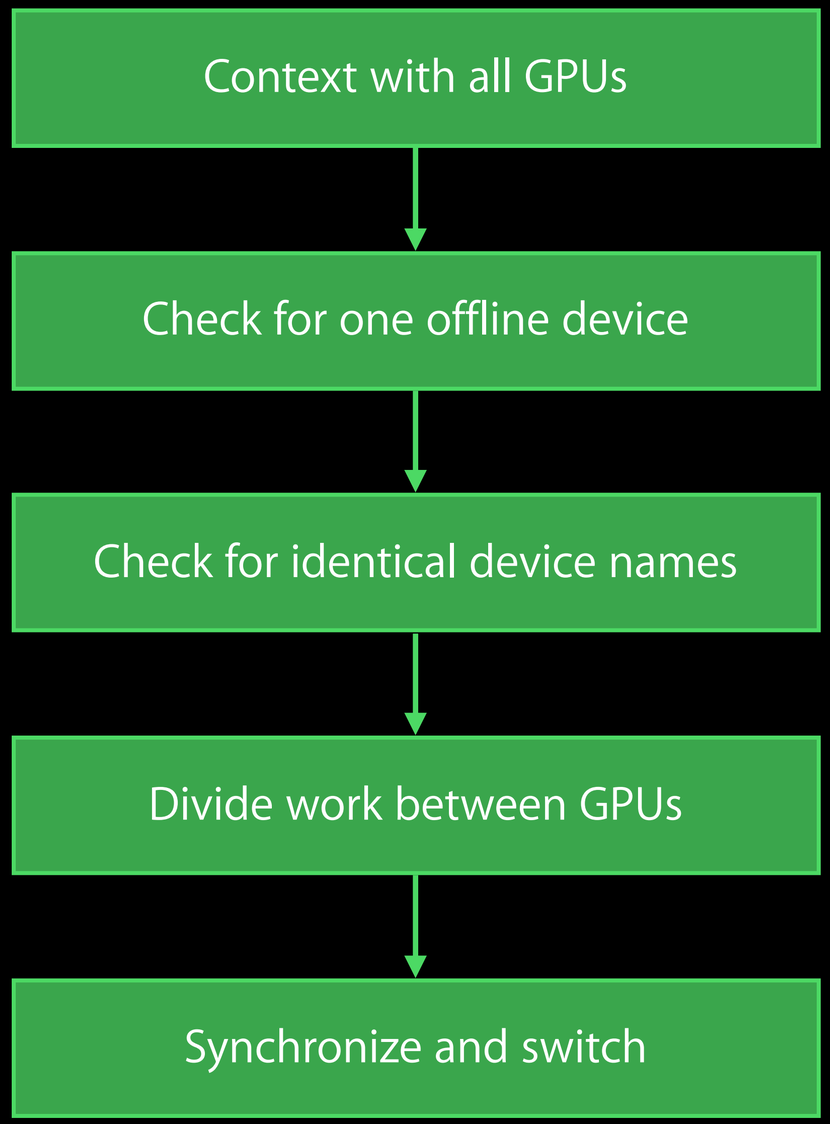
Laptop with automatic graphics switching

Multiple displays connected to multiple GPUs

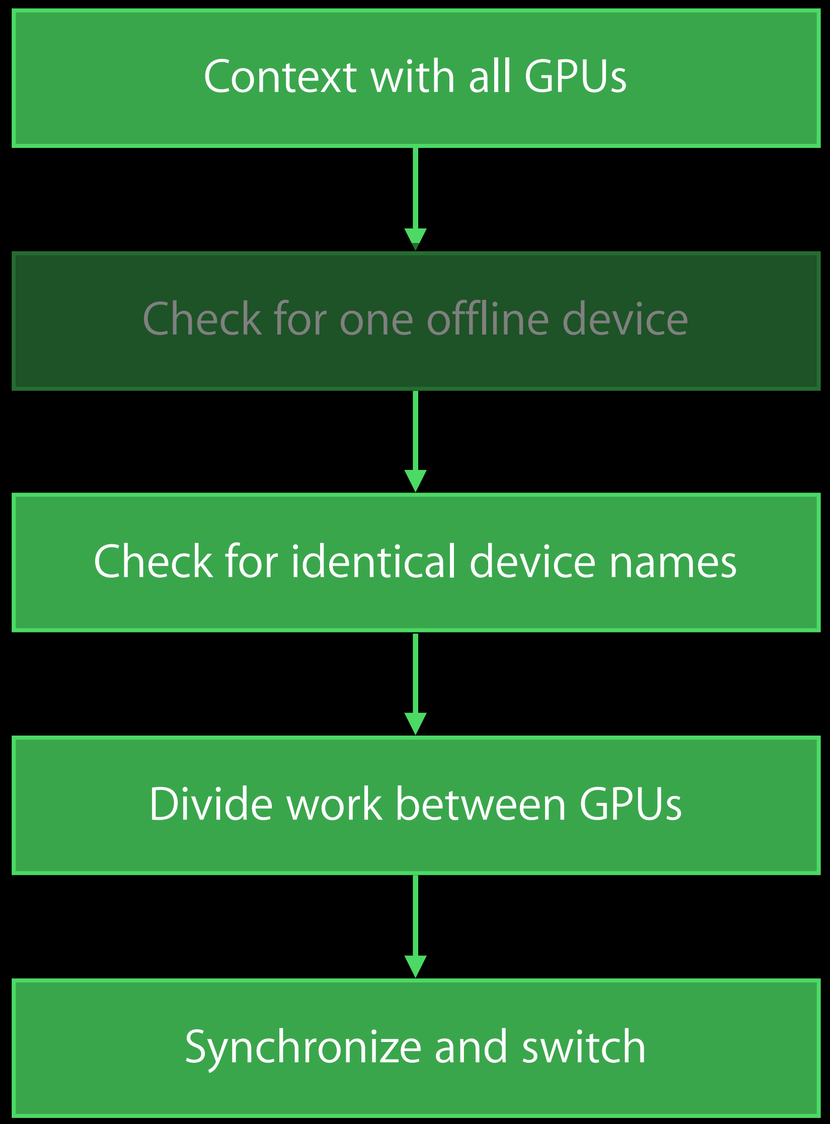
# Complete Example

Context with all GPUs

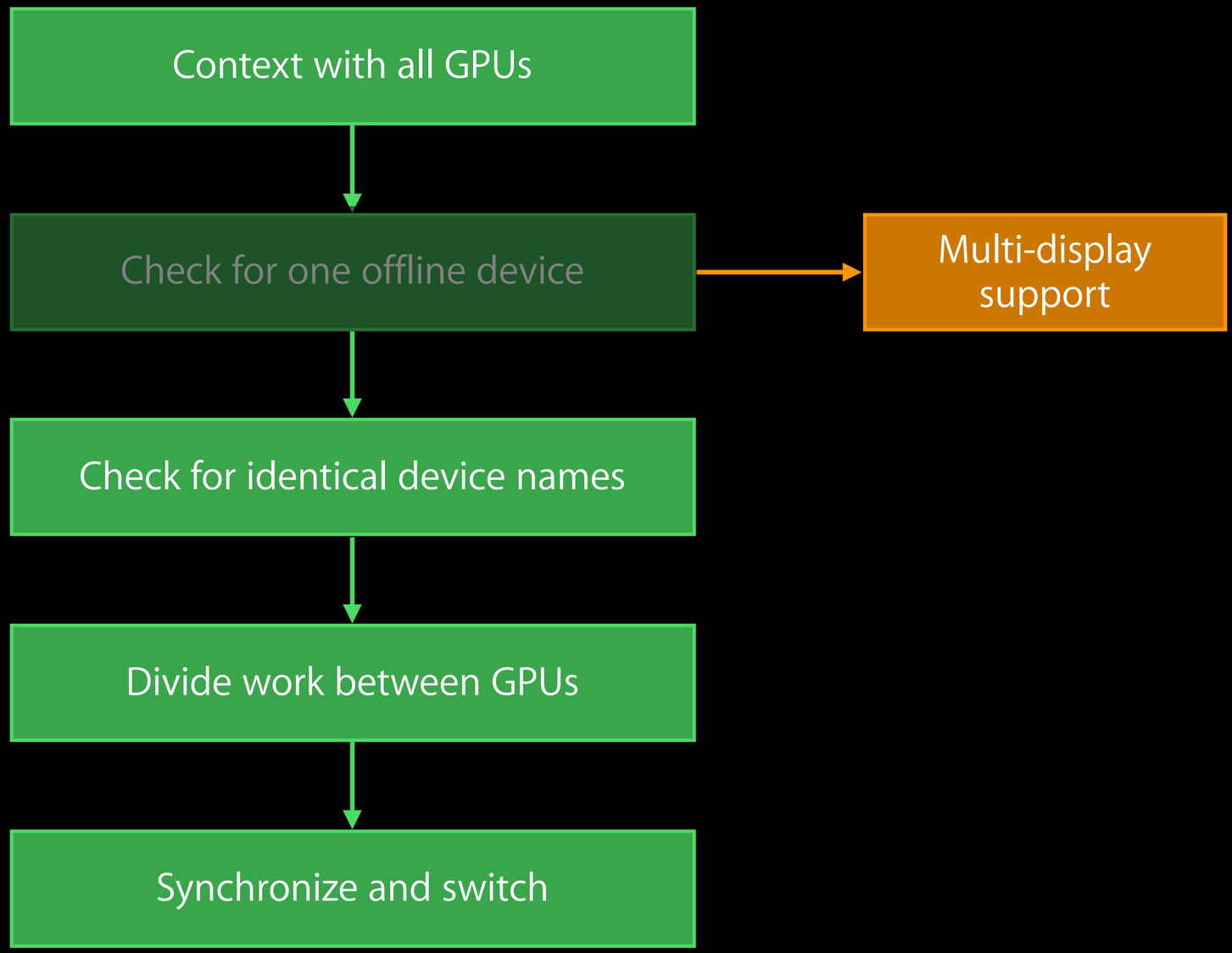
# Complete Example



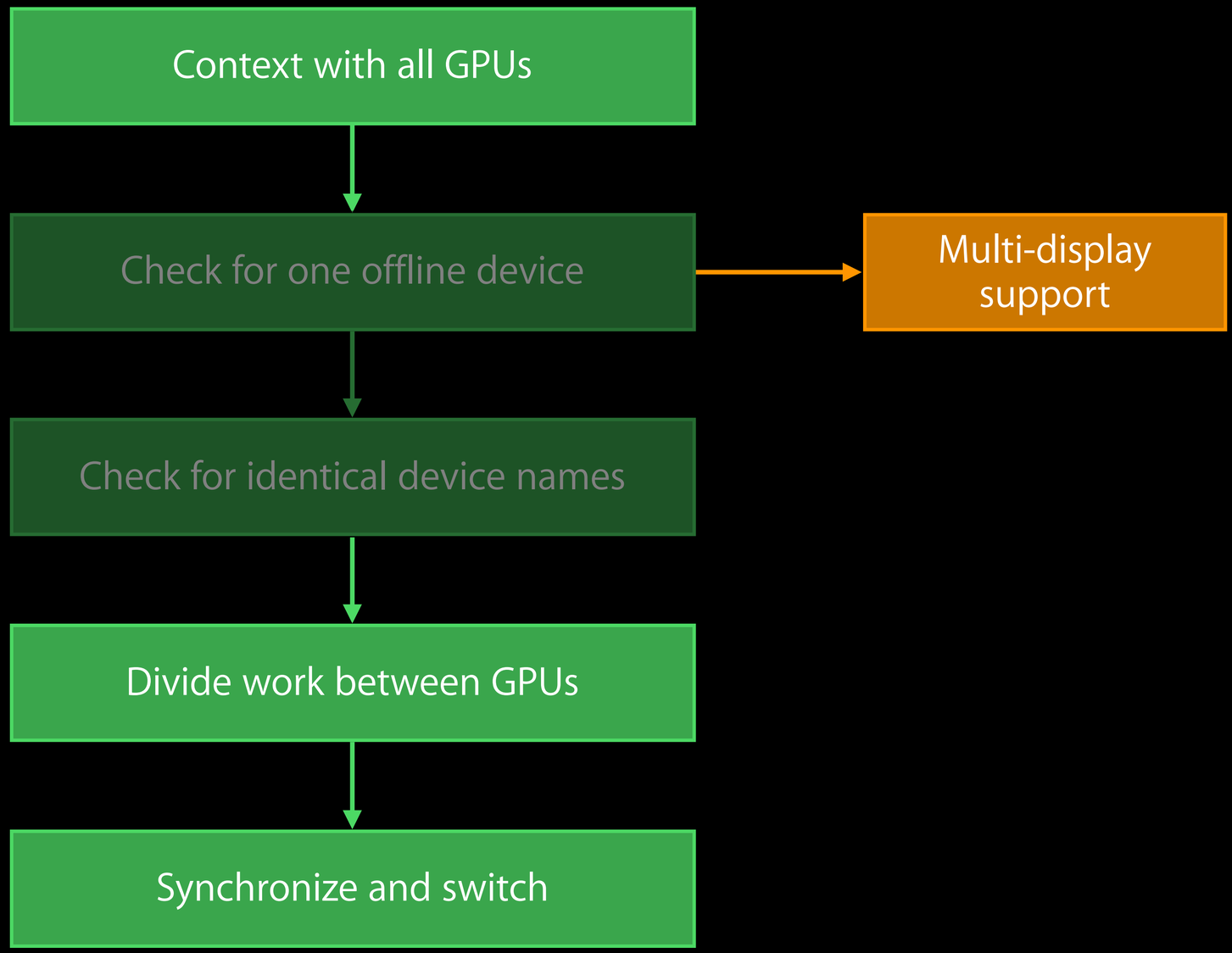
# Complete Example



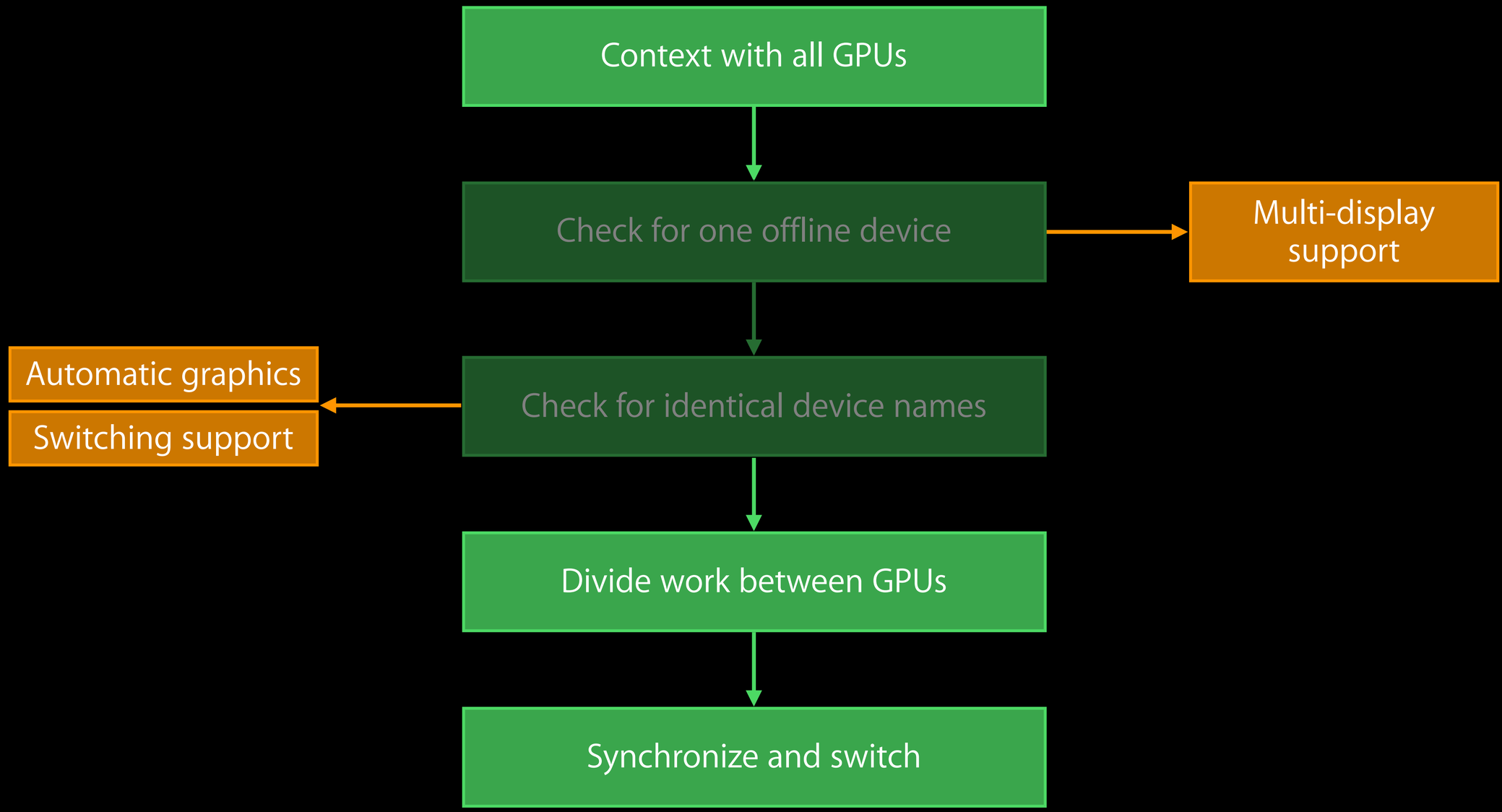
# Complete Example



# Complete Example



# Complete Example



# *Demo*

# More Information

Filip Iliescu
Graphics and Game Technologies Evangelist
filiescu@apple.com

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

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

# Labs

| | | |
|---|---|---|
| • OpenCL Lab | Graphics and Games Lab A | Tuesday 10:15AM |
| • OpenGL Lab | Graphics and Games Lab B | Tuesday 10:15AM |
| • OpenGL ES Lab | Graphics and Games Lab B | Wednesday 2:00PM |

WWDC14