What’s New in OpenCL

Session 401

Abe Stephens, PhD
OpenCL Team

What You Will Learn

- Short overview of OpenCL
- Using OpenCL with Grand Central Dispatch
- Compiling OpenCL source in Xcode

What Is OpenCL?

OpenCL

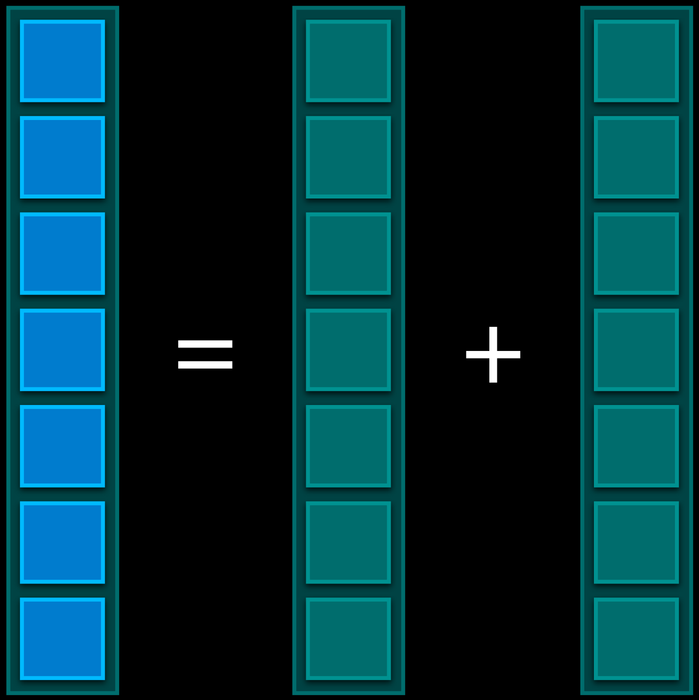
Design goals

- Leverage CPUs and GPUs to accelerate parallel computation
- Get dramatic speedups for computationally intensive applications
- Write accelerated portable code across different devices and architectures

Data Parallel

```
a + b = c
```

```
for (int i=0;i<N;++i) {
  c[i] = a[i] + b[i]
}
```



OpenCL Kernel

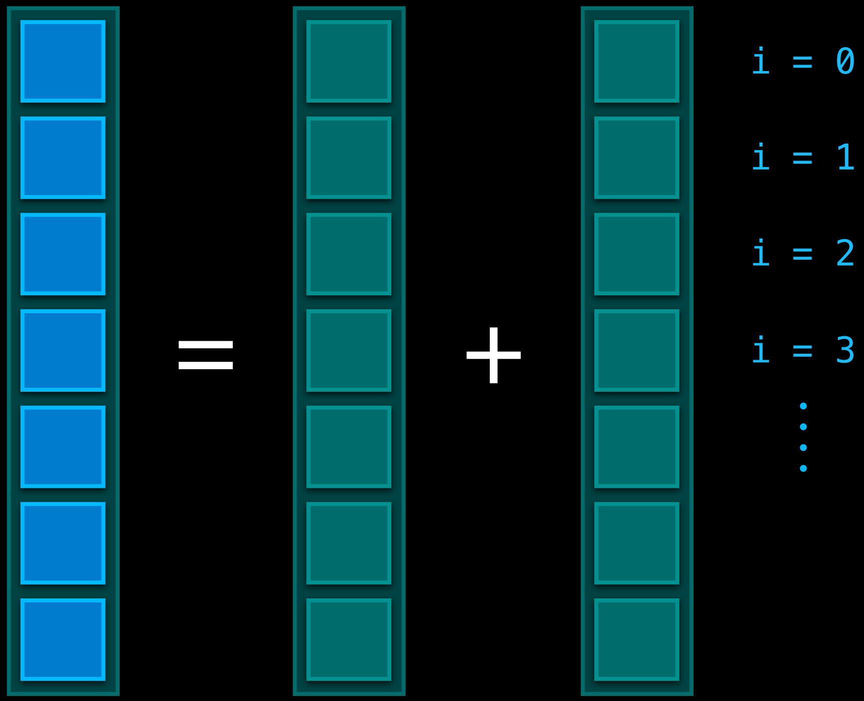
```
for (int i=0;i<N;++i) {
    c[i] = a[i] + b[i]
}
```

```
kernel void add_arrays(global int* a,
                       global int* b,
                       global int* c)
{
  int i = get_global_id(0);
  c[i] = a[i] + b[i];
}
```

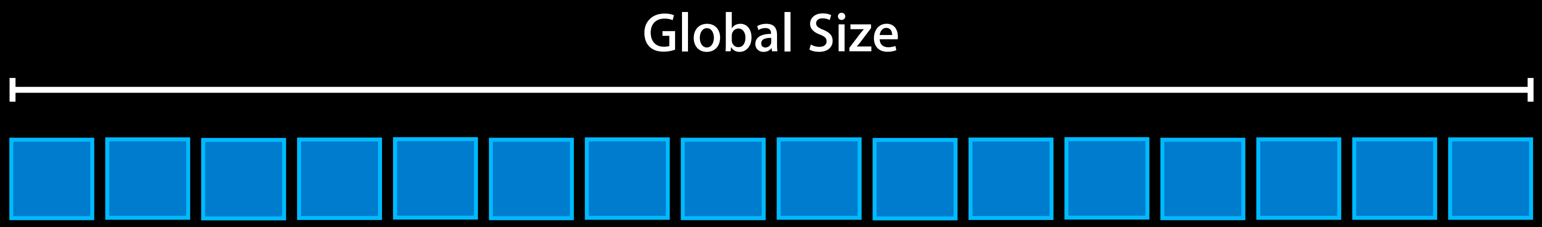
OpenCL Kernel

```
kernel void add_arrays(global int* a,
                       global int* b,
                       global int* c)
{
  int i = get_global_id(0);
  c[i] = a[i] + b[i];
}
```

Work Items

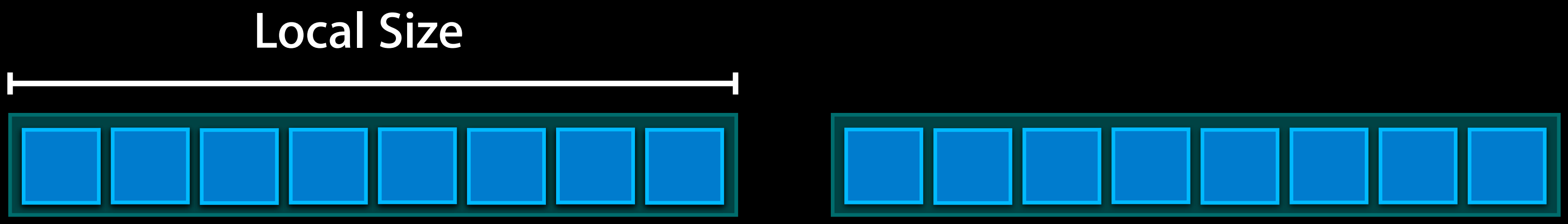


Work Item Dimensions

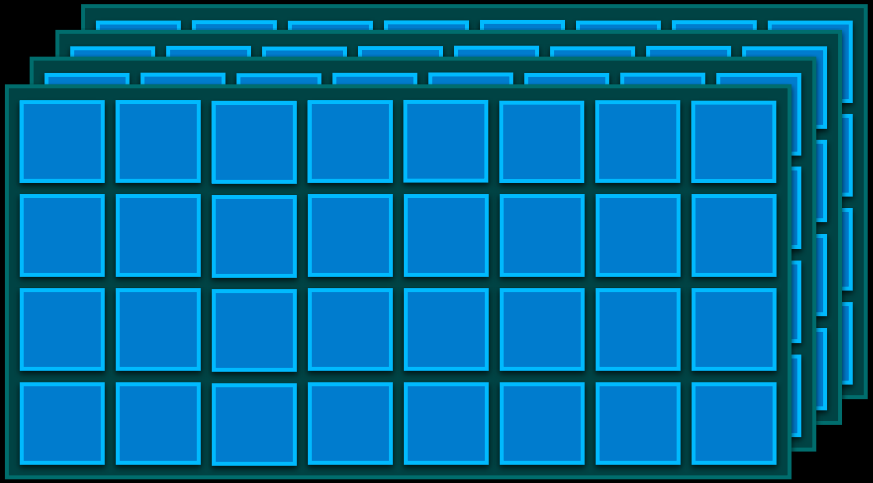


Global Size

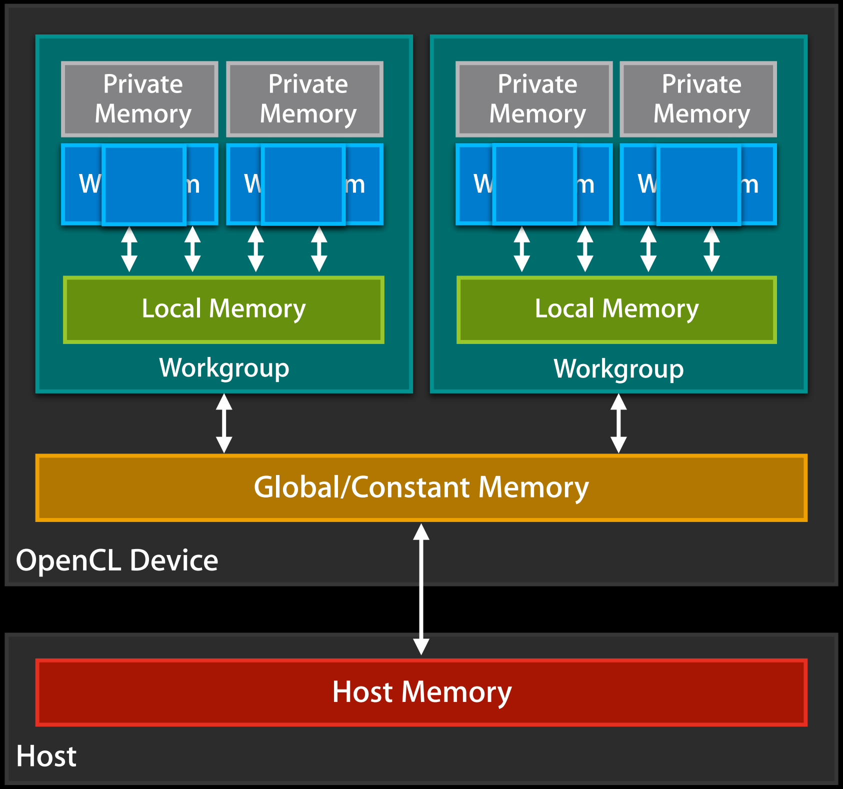
Work Groups



ND-range Dimensions



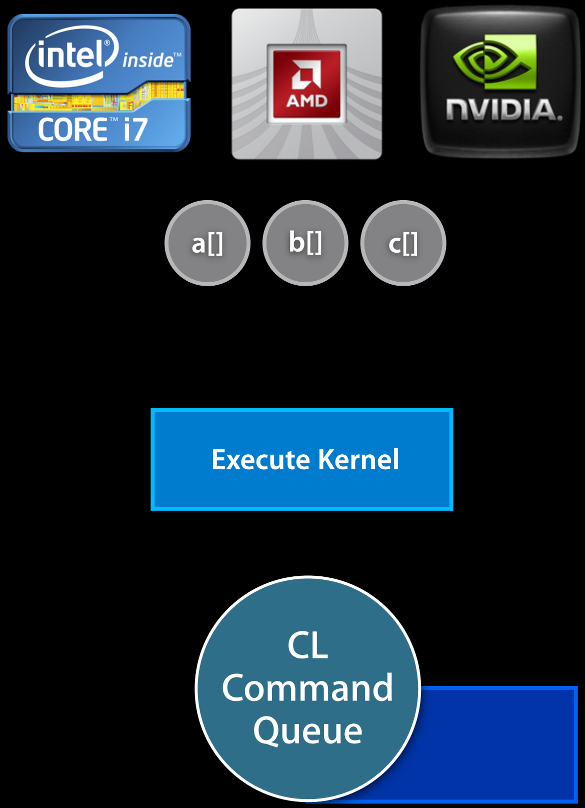
OpenCL Memory Model



Object Model

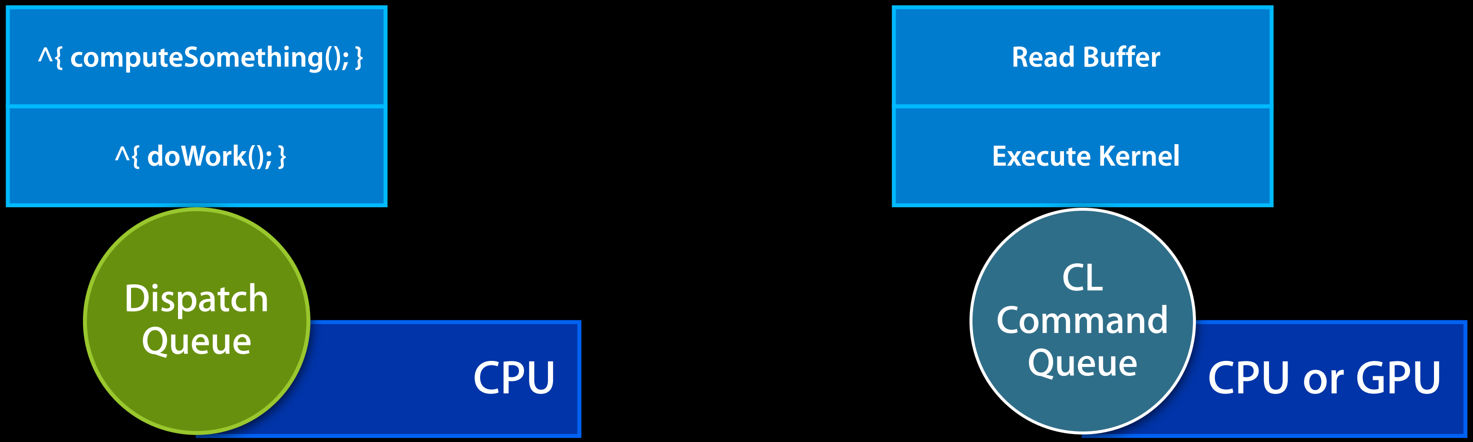
- Devices and contexts
- Command queues
- Memory objects
- Programs and kernels

```
kernel void add_arrays(global int* a,
                       global int* b,
                       global int* c)
{
  int i = get_global_id(0);
  c[i] = a[i] + b[i];
}
```



Integration with Grand Central Dispatch

Different Types of Queues



^{ computeSomething(); }

^{ doWork(); }

Dispatch Queue

CPU

Read Buffer

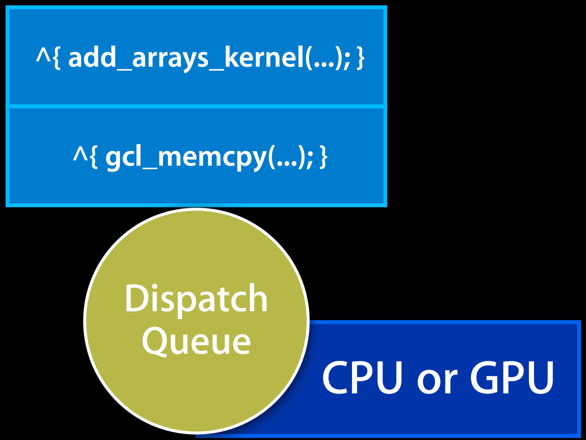
Execute Kernel

CL Command Queue

CPU or GPU

Dispatch Queues in OpenCL

New



^{ add_arrays_kernel(...); }

^{ gcl_memcpy(...); }

Dispatch Queue

CPU or GPU

Sending Kernels to Dispatch Queues

^{ add_arrays_kernel(...); }

```
dispatch_async(q,^{
  cl_ndrange ndrange = { 1, {0}, {N}, {0} };
  add_arrays_kernel(&ndrange, a, b, c);
});
```

What Is a Kernel Block?

```
kernel void add_arrays(global int* a, global int* b, global int *c)
{
  size_t i = get_global_id(0);
  c[i] = a[i] + b[i];
}
```

OpenCL C Compiler

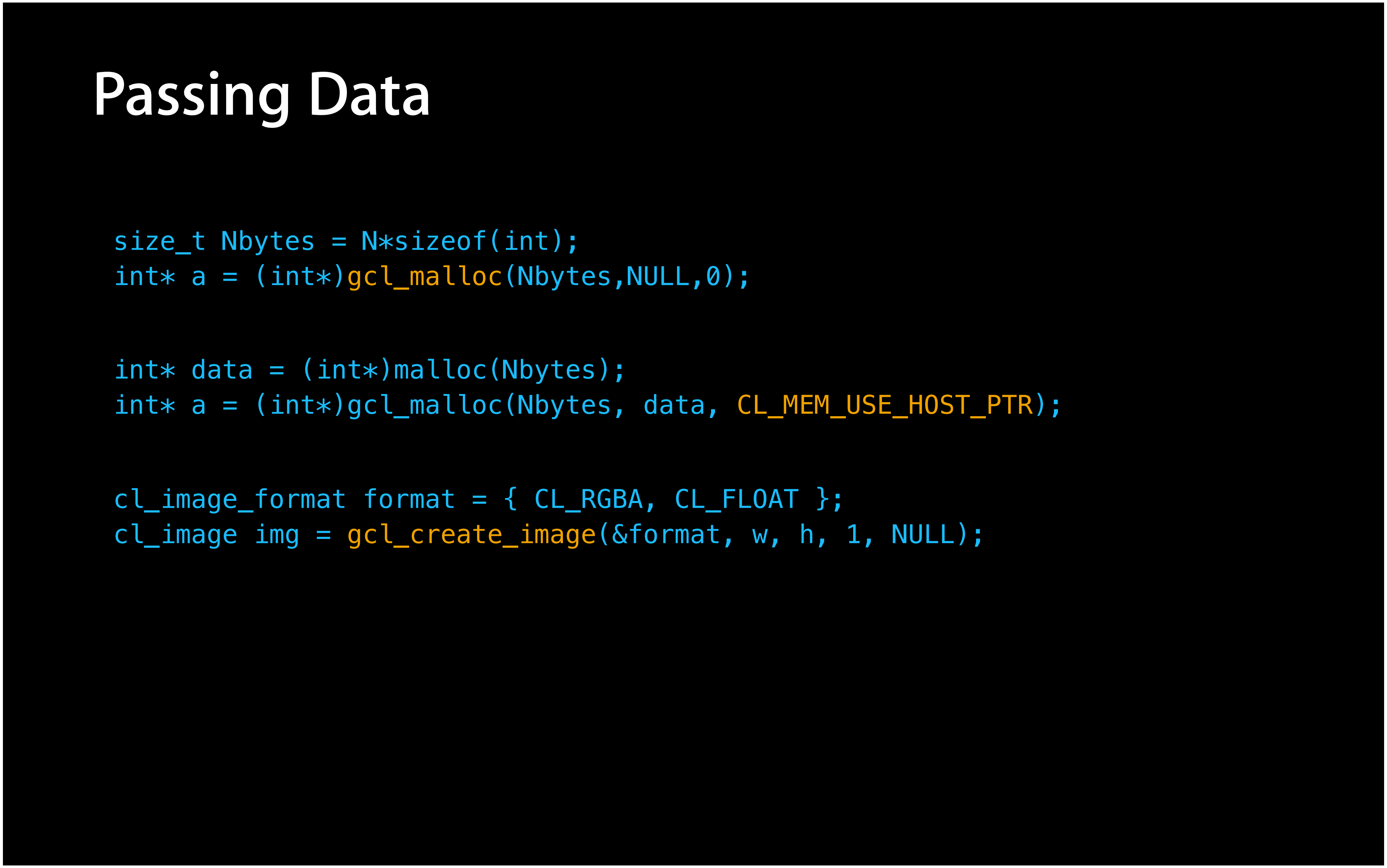
```
extern void (^add_arrays_kernel)(const cl_ndrange *ndrange,
                                 cl_int* a, cl_int* b, cl_int* c);

cl_ndrange nd = { 1, global_offset, global_size, local_size };
```

```
dispatch_async(q,^{
  cl_ndrange ndrange = { 1, {0}, {N}, {0} };
  add_arrays_kernel(&ndrange, a, b, c);
});
```

```
kernel void add_arrays(global int* a, global int* b, global int *c)
```

Creating OpenCL Dispatch Queues

```
dispatch_queue_t q =
  gcl_create_dispatch_queue(CL_DEVICE_TYPE_GPU,NULL);

gcl_create_dispatch_queue(CL_DEVICE_TYPE_GPU|
                          CL_DISPATCH_QUEUE_PRIORITY_HIGH,NULL);

cl_device_id device_id = GetUserDeviceChoice();
gcl_create_dispatch_queue(CL_DEVICE_TYPE_USE_DEVICE_ID, device_id);
```



Passing Data

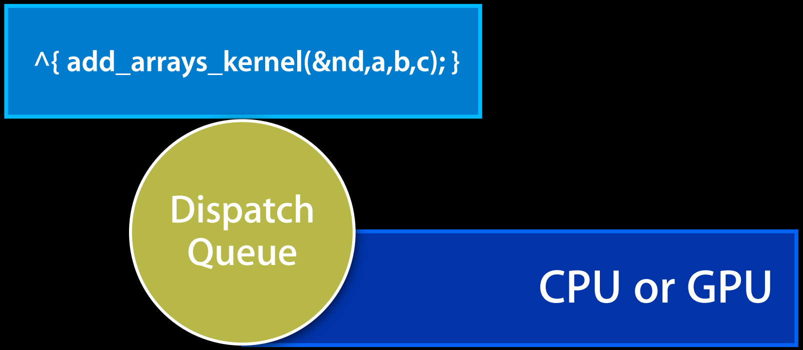
```
size_t Nbytes = N*sizeof(int);
int* a = (int*)gcl_malloc(Nbytes,NULL,0);

int* data = (int*)malloc(Nbytes);
int* a = (int*)gcl_malloc(Nbytes, data, CL_MEM_USE_HOST_PTR);

cl_image_format format = { CL_RGBA, CL_FLOAT };
cl_image img = gcl_create_image(&format, w, h, 1, NULL);
```

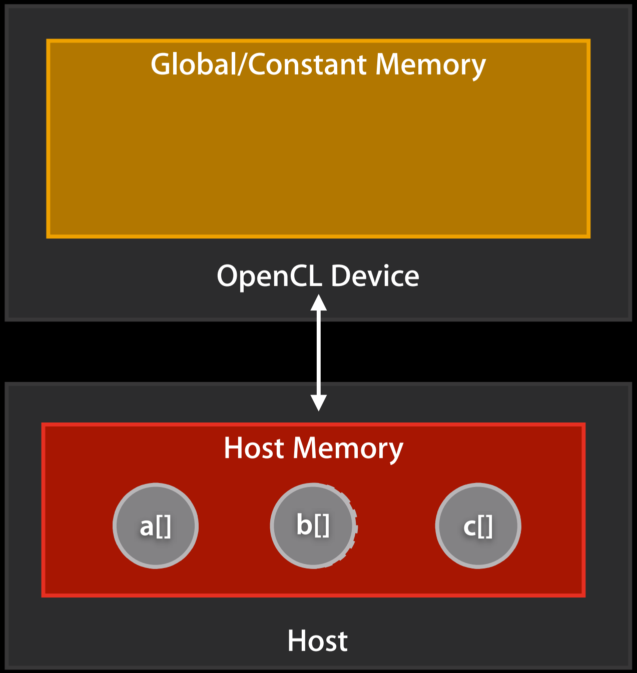
Other Commands

```
int hostData[N];
dispatch_async(q,^{ gcl_memcpy(a, hostData, Nbytes);});

__block void* c_mapped;
dispatch_sync(q,^{ c_mapped = gcl_map_ptr(c, CL_MAP_READ, 0);});
// ...
dispatch_sync(q,^{ gcl_unmap(c_mapped);});
```

Memory Consistency

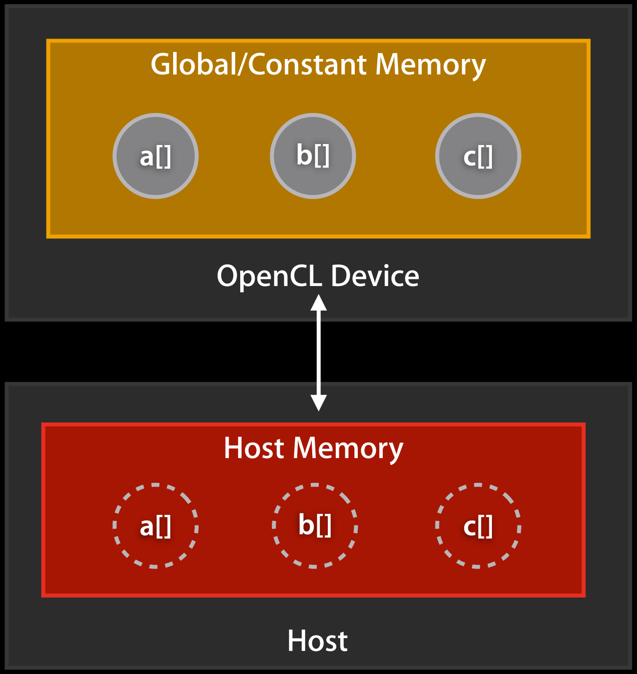
```
int* a = (int*)gcl_malloc(Nbytes,NULL,0);

int* b = (int*)gcl_malloc(Nbytes,NULL,0);

int* c = (int*)gcl_malloc(Nbytes,NULL,0);
```





Memory Consistency

```
__block int* c_map;
dispatch_sync(q,^{
  c_map = gcl_map_ptr(c, CL_MAP_READ, 0);});

// Process results in c_map

dispatch_sync(q,^{ gcl_unmap(c_map);});
```



Waiting in the Application

```
dispatch_group_t group = dispatch_group_create();

dispatch_group_async(group, q0, ^{
  cl_ndrange ndrange = { 1, {0}, {N/2}, {0} };
  add_arrays_kernel(&ndrange, a, b, c);
});

dispatch_group_async(group, q1, ^{
  cl_ndrange ndrange = { 1, {N/2}, {N/2}, {0} };
  add_arrays_kernel(&ndrange, a, b, c);
});

// Perform more work

dispatch_group_wait(group, DISPATCH_TIME_FOREVER);
```

One Block Waiting for Another

```
dispatch_group_t group = dispatch_group_create();
dispatch_group_enter(group);

dispatch_async(q0, ^{
  cl_ndrange ndrange = { 1, {0}, {N/2}, {0} };
  add_arrays_kernel(&ndrange, a, b, c);
  dispatch_group_leave(group);
});

dispatch_async(q1, ^{
  dispatch_group_wait(group, DISPATCH_TIME_FOREVER);
  cl_ndrange ndrange = { 1, {N/2}, {N/2}, {0} };
  add_arrays_kernel(&ndrange, a, b, c);
});

// App does not wait
```

Many Other Functions

- Dispatch callbacks, semaphores
- Memory object finalizers
- Timing functions
- Interoperate with command queues

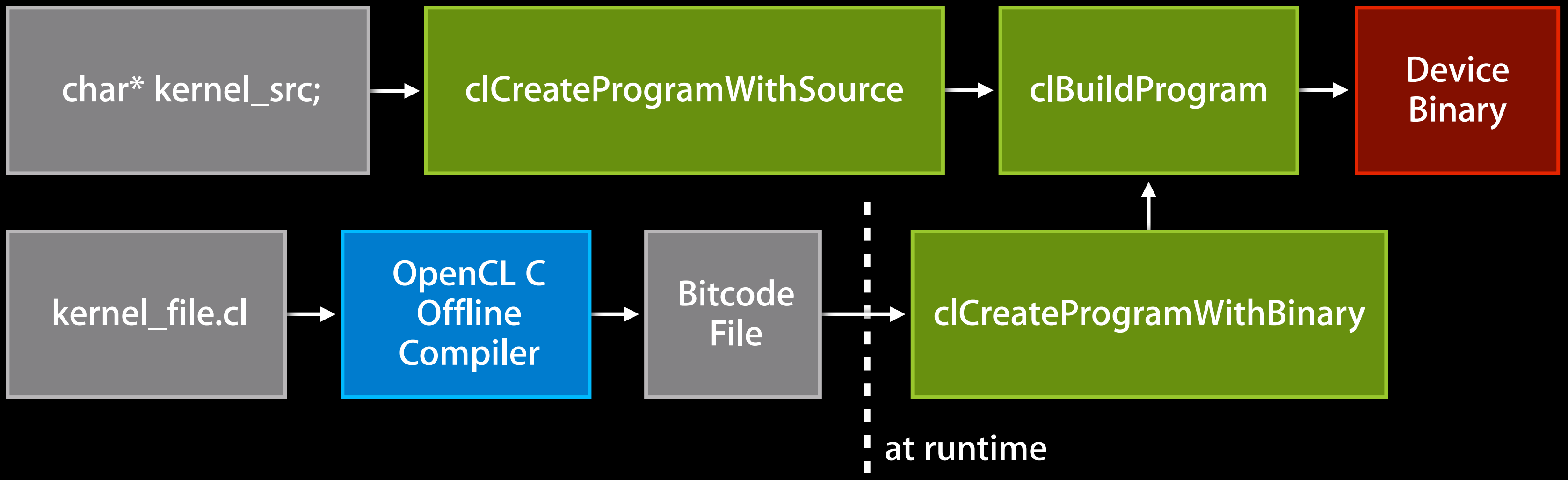
/System/Library/Frameworks/OpenCL.framework/Headers/gcl.h

```
#include <OpenCL/opencl.h>
```

Offline Compilation

Compilation in Lion

New



Compiling to Bitcode

New

/System/Library/Frameworks/OpenCL.framework/Libraries/openclc

```
$ openclc -x cl -triple gpu_32-applecl-darwin -emit-llvm-bc -o kernel_gpu32.bc
kernel.cl
```

Option 1: → | `-triple gpu_32-applecl-darwin`

Option 2: → | `-triple cpu_32-applecl-darwin`

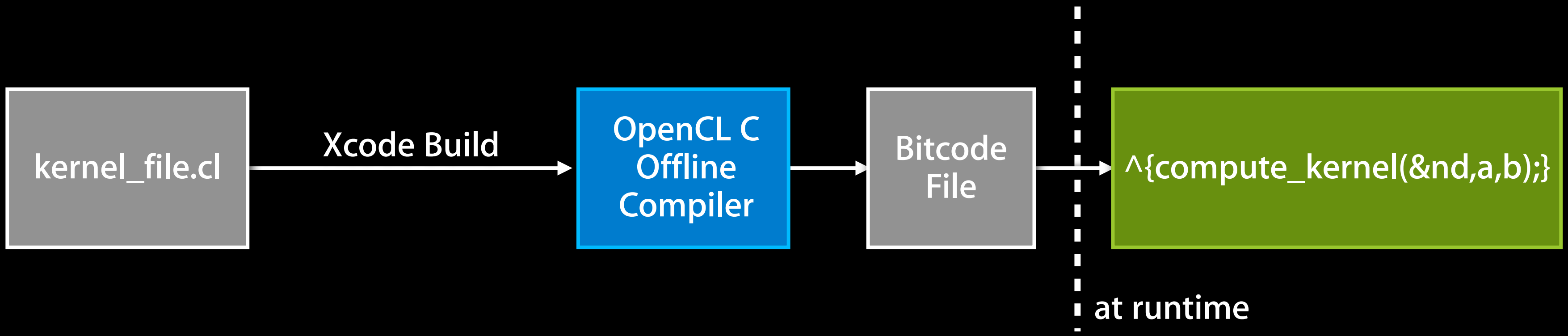
Option 3: → | `-triple cpu_64-applecl-darwin`

Compiling in Xcode 4

New

Automatic compilation of .cl files to bit code

- Generate kernel block declaration headers
- Set compiler options from Xcode project settings



kernel_file.cl

Xcode Build

OpenCL C Offline Compiler

Bitcode File

^{compute_kernel(&nd,a,b);}

at runtime

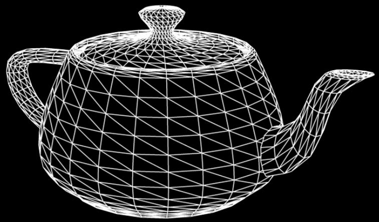
Demo

Sharing with OpenGL

Jim Shearer
OpenCL Engineer

Sharing

Application data

		OpenCL	OpenGL
Geometry		Generate or Modify	Render
Images		Post-process	Render

Sharing

Synchronization primitives

OpenCL

Event Object

OpenGL

Sync Object

OpenCL	OpenGL
Devices	Renderers

GPU CPU

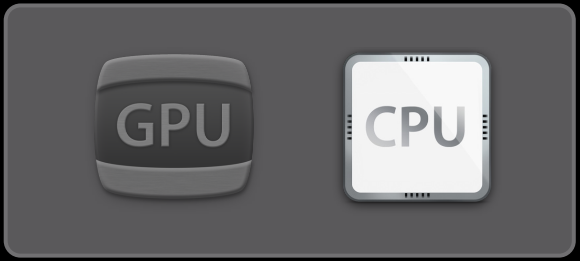
OpenCL

Devices

Device Types

OpenGL

Renderers



`clGetDeviceIds`
`CL_DEVICE_TYPE_CPU`

OpenCL

Devices

Device Types

OpenGL

Renderers

`clGetDeviceIds`

`CL_DEVICE_TYPE_GPU`

OpenCL

Devices

Device Types

OpenGL

Renderers

`clGetDeviceIds`
`CL_DEVICE_TYPE_ALL`

OpenCL

Devices

Device Types

OpenGL

Renderers

Pixel Format

```
NSOpenGLPixelFormatAttribute attr[] =
{
    NSOpenGLPFAOpenGLProfile,
    NSOpenGLProfileVersion3_2Core,
    NSOpenGLColorSize, 24,
    NSOpenGLAlphaSize, 8,
    NSOpenGLPFAAccelerated,
    0
};
```

OpenCL

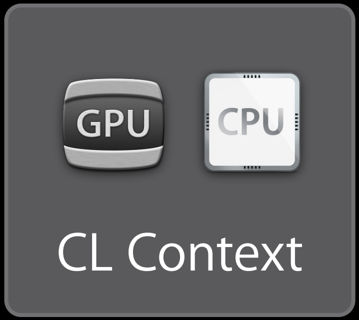
Devices

Device Types

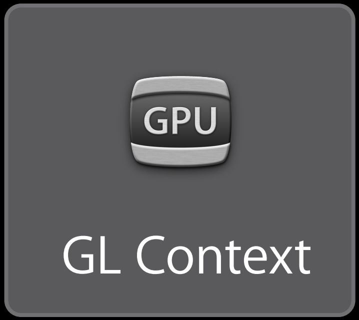
OpenGL

Renderers

Pixel Format



CL Context



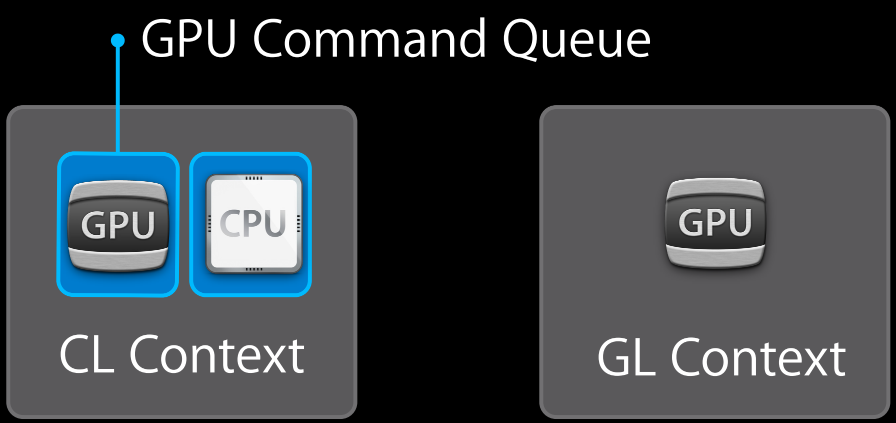
GL Context

OpenCL

- Devices
- Device Types
- Command Queue

OpenGL

- Renderers
- Pixel Format


GPU Command Queue

CL Context

GL Context

OpenCL

Devices

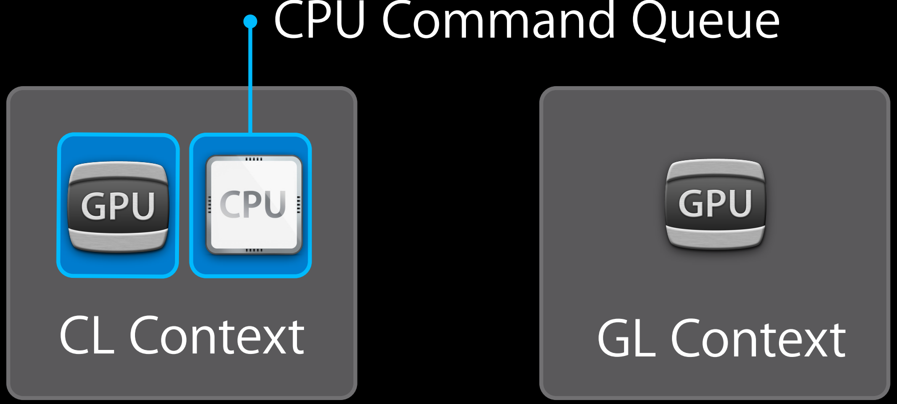
Device Types

Command Queue

OpenGL

Renderers

Pixel Format



CPU Command Queue

OpenCL

Devices

Device Types

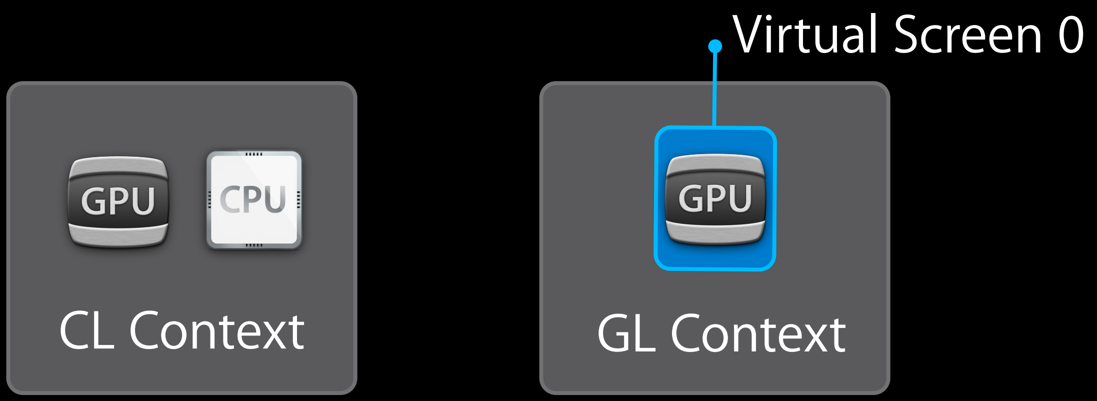
Command Queue

OpenGL

Renderers

Pixel Format

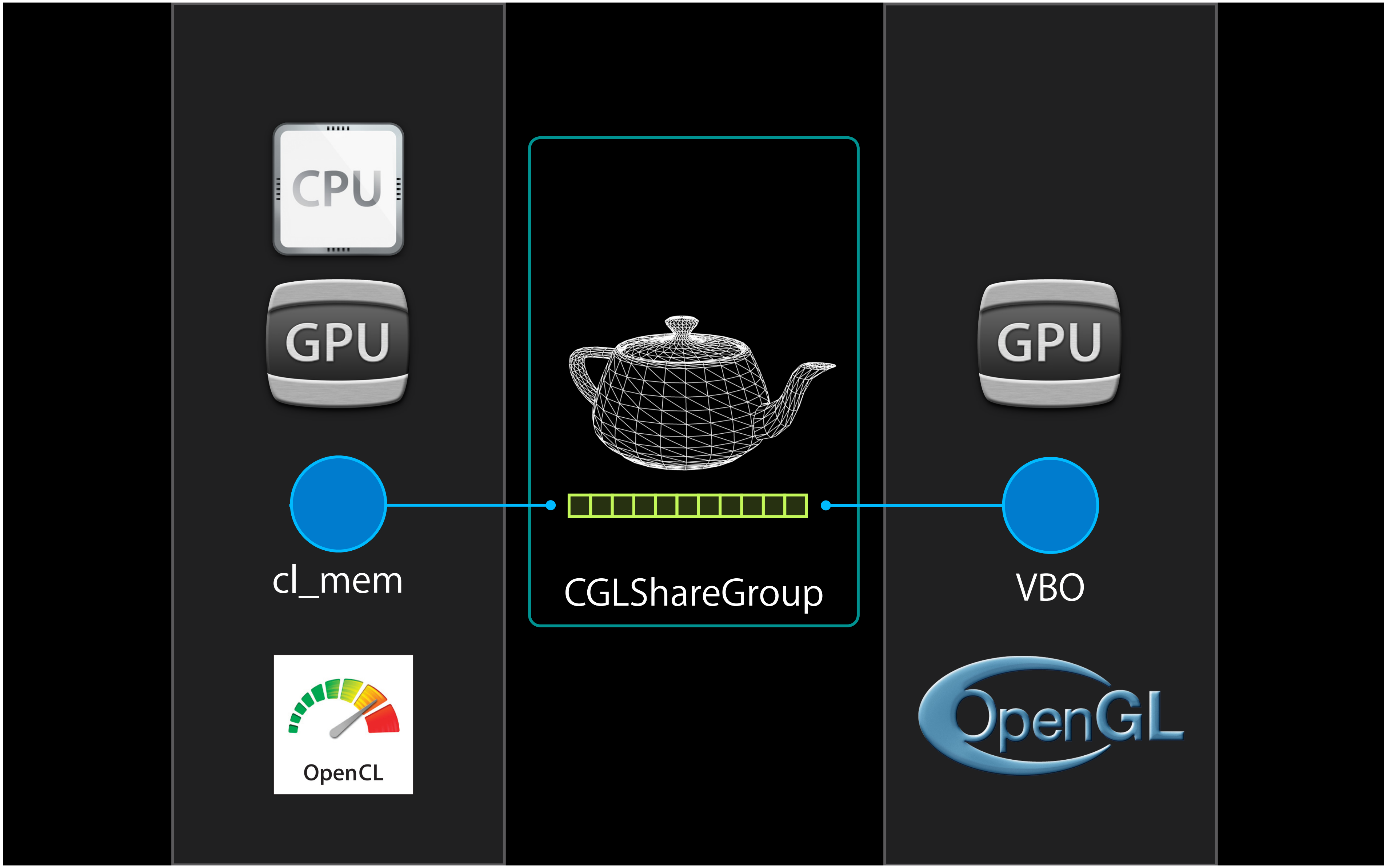
Virtual Screen



Virtual Screen 0

CL Context

GL Context



CPU

GPU

cl_mem

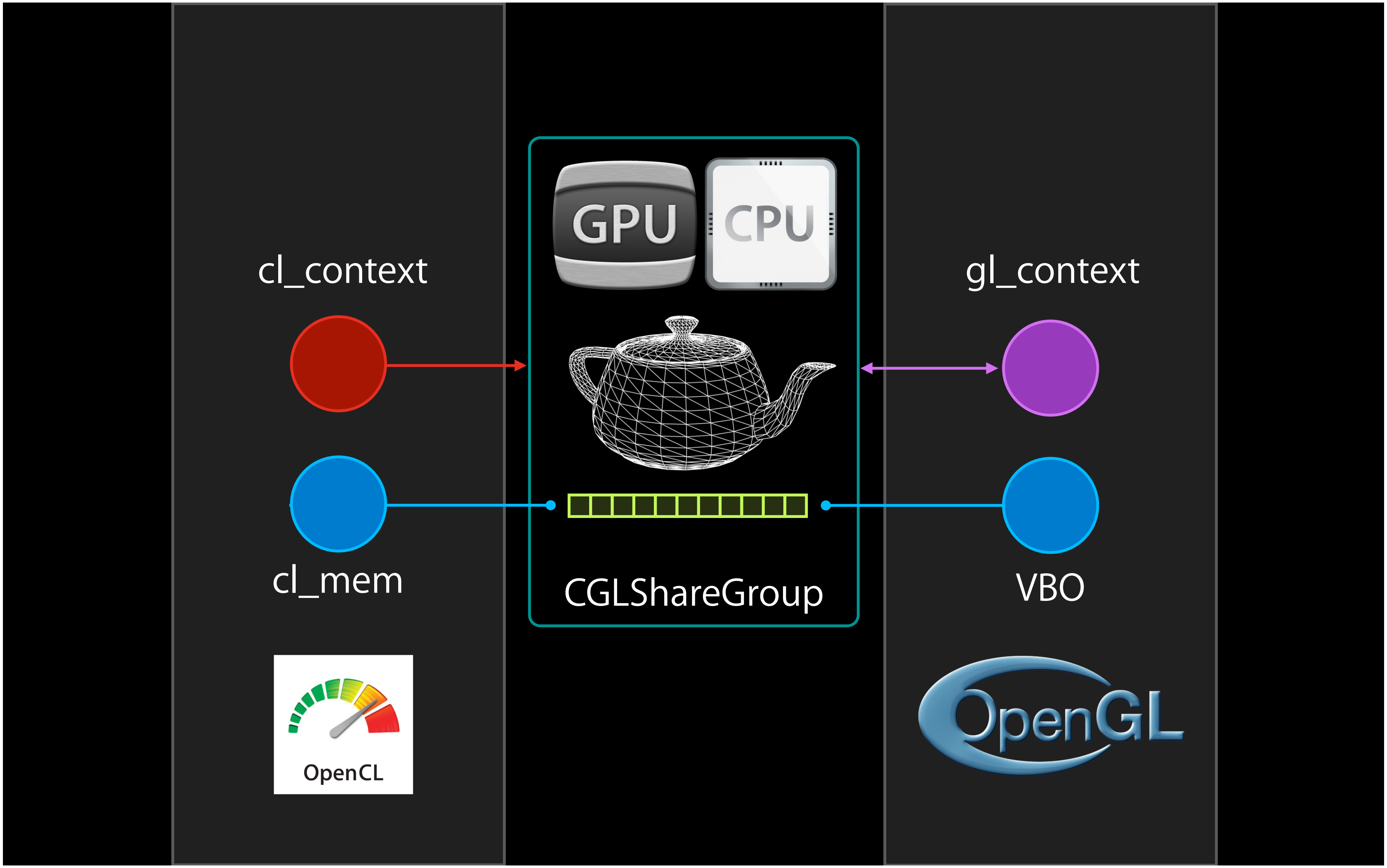
OpenCL

CGLShareGroup

GPU

VBO

OpenGL



cl_context

gl_context

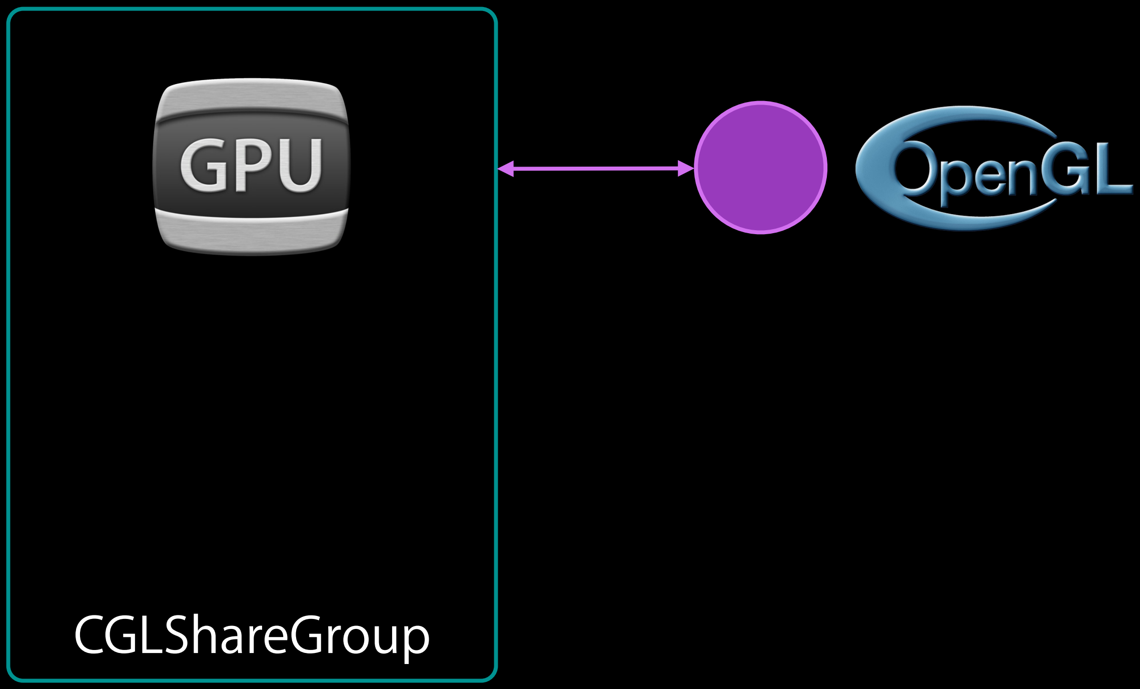
cl_mem

CGLShareGroup

VBO

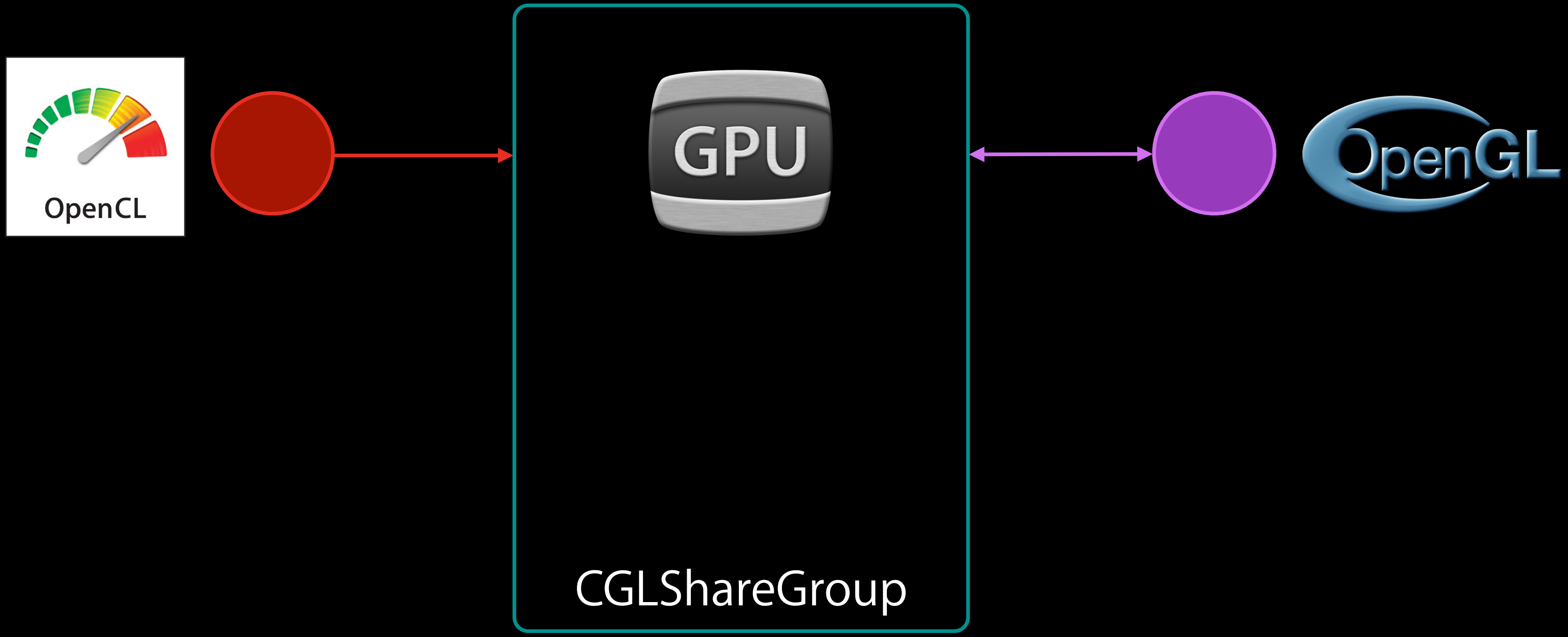
Using Share Groups

You already have one



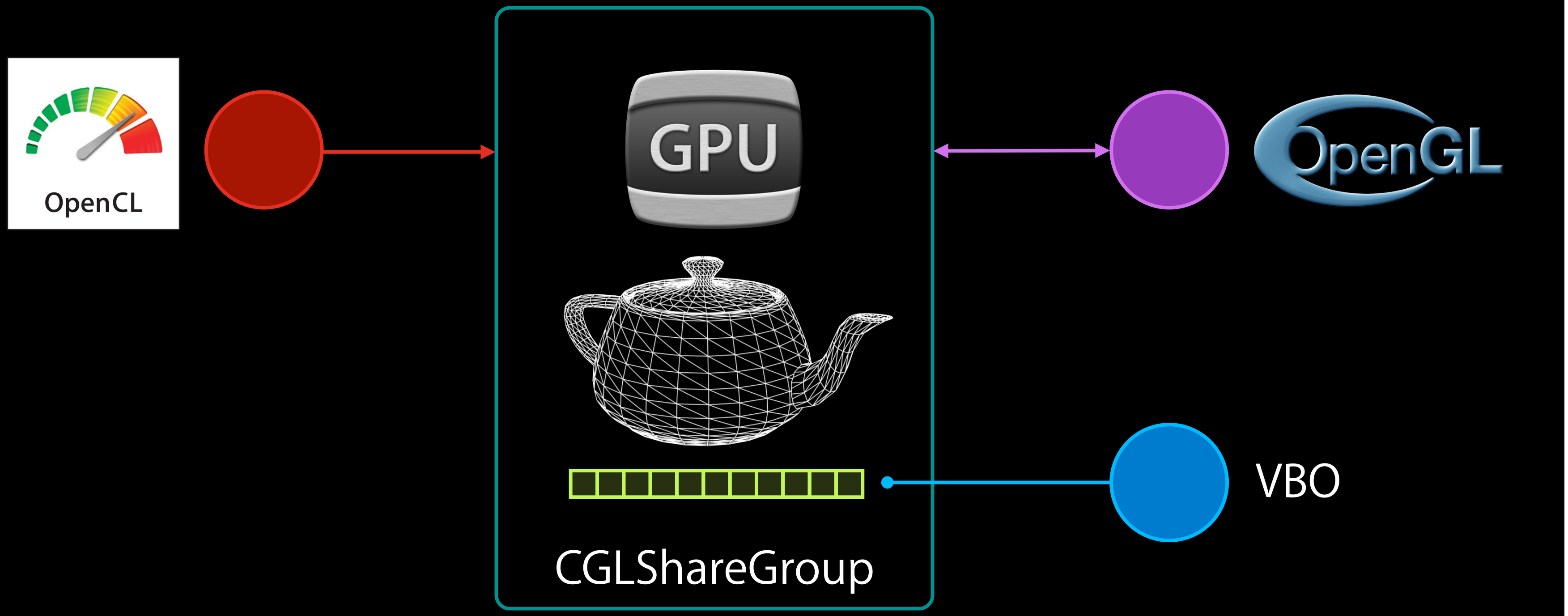
Using Share Groups

Create CL context from the share group



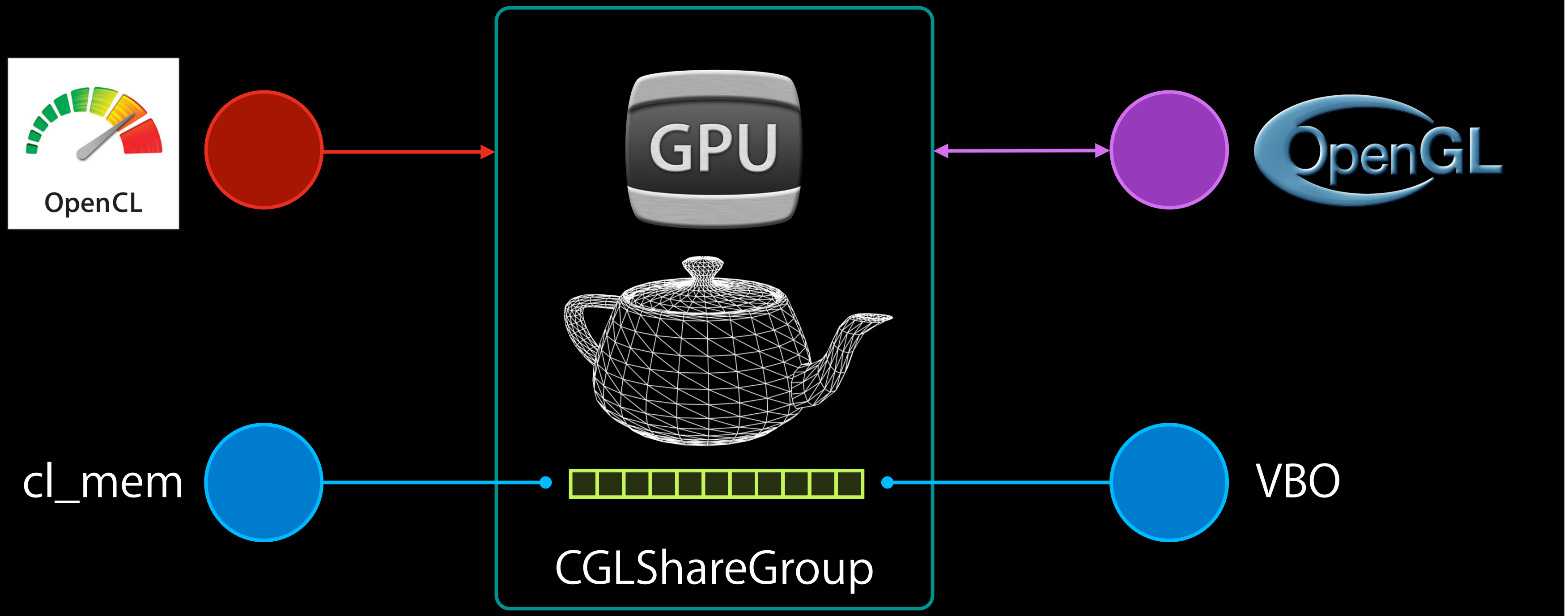
Using Share Groups

Create shared objects in GL first



Using Share Groups

Create CL objects from GL objects



CL-GL Sharing Setup

Creating your CL context

```
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(cgl_ctx),
    0
};

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

CL-GL Sharing Setup

Adding the CPU device

```
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(cgl_ctx),
    0
};
cl_device_id cpu;
clGetDeviceIDs(NULL, CL_DEVICE_TYPE_CPU, 1, &cpu, NULL);

cl_context cl_ctx = clCreateContext(properties, 1, &cpu, NULL, NULL, &err);
```

CL-GL Sharing Setup

Matching the current virtual screen

- The currently in-use GL renderer
- Performance

```
cl_device_id cl_device;

clGetGLContextInfoAPPLE( cl_ctx, cgl_context,
  CL_CGL_DEVICE_FOR_CURRENT_VIRTUAL_SCREEN_APPLE,
  sizeof(cl_device_id), &cl_device, NULL);
```

CL-GL Sharing Setup

Creating OpenCL memory objects

```
// Create cl_mem object from GL VBO

cl_int err;
cl_mem buf = clCreateFromGLBuffer(context, CL_MEM_READ_WRITE, vbo, &err);


clCreateFromGLTexture2D(context,flags,target,miplevel,texture,&err)

clCreateFromGLTexture3D(context,flags,target,miplevel,texture,&err)

clCreateFromGLRenderbuffer(context,flags,renderbuffer,&err)
```

Creating Images from Textures

What do you get?

OpenGL Format	OpenCL Format
GL_RGBA8	CL_RGBA, CL_UNORM_INT8
GL_DEPTH	CL_R, data type
GL_RGBA32F, GL_RGBA32F_ARB	CL_RGBA, CL_FLOAT

... and many more.

CL-GL Sharing

Using shared objects in OpenCL

- Generally, you use the objects as usual
- Flush, acquire, compute, release

```
// Done with previous GL commands
glFlush();

// Update geometry in OpenCL
cl_mem mem_objs[] = { buffer_cl };
clEnqueueAcquireGLObjects(queue, 1, mem_objs, ...);

// Wail away with OpenCL

// Done with CL commands
clEnqueueReleaseGLObjects(queue, 1, mem_objs, ...);
```

Only required if you are on a different thread!

CL-GL Sharing

Using shared objects in OpenCL on the same thread

- Generally, you use the objects as usual
- acquire, compute, release

```
// Update geometry in OpenCL
cl_mem mem_objs[] = { buffer_cl };
clEnqueueAcquireGLObjects(queue, 1, mem_objs, ...);

// Wail away with OpenCL

// Done with CL commands
clEnqueueReleaseGLObjects(queue, 1, mem_objs, ...);
```

CL-GL Sharing

Using shared objects in OpenGL

- OpenCL flushes for you

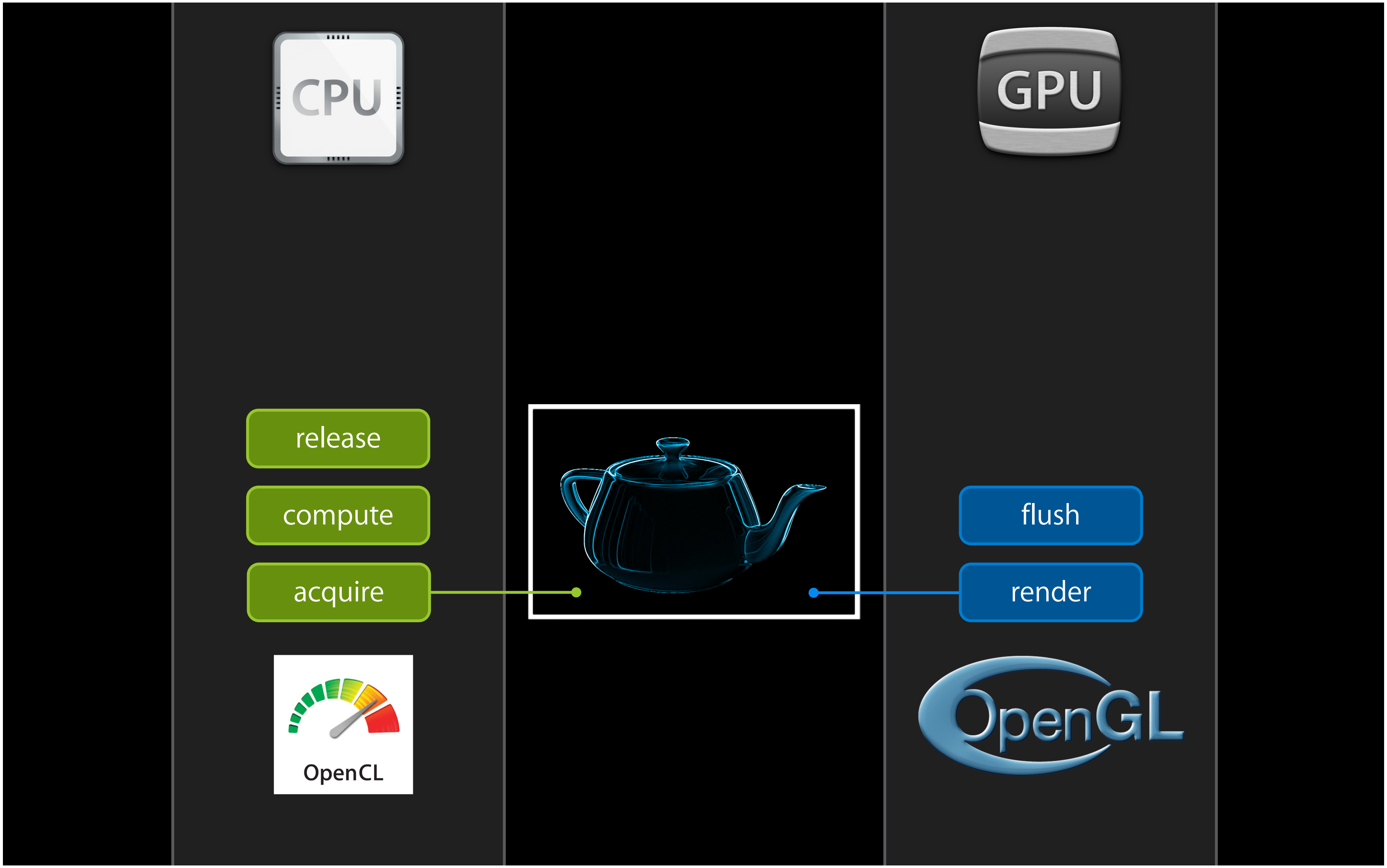
  `clEnqueueReleaseGLObjects()`

- Simply bind and use in OpenGL

CL Events and GL Sync Objects

Fine-grained synchronization

- Wait on some work to complete
- Avoid completely emptying the pipes
- Natural correspondence—use them together



CPU

release

compute

acquire

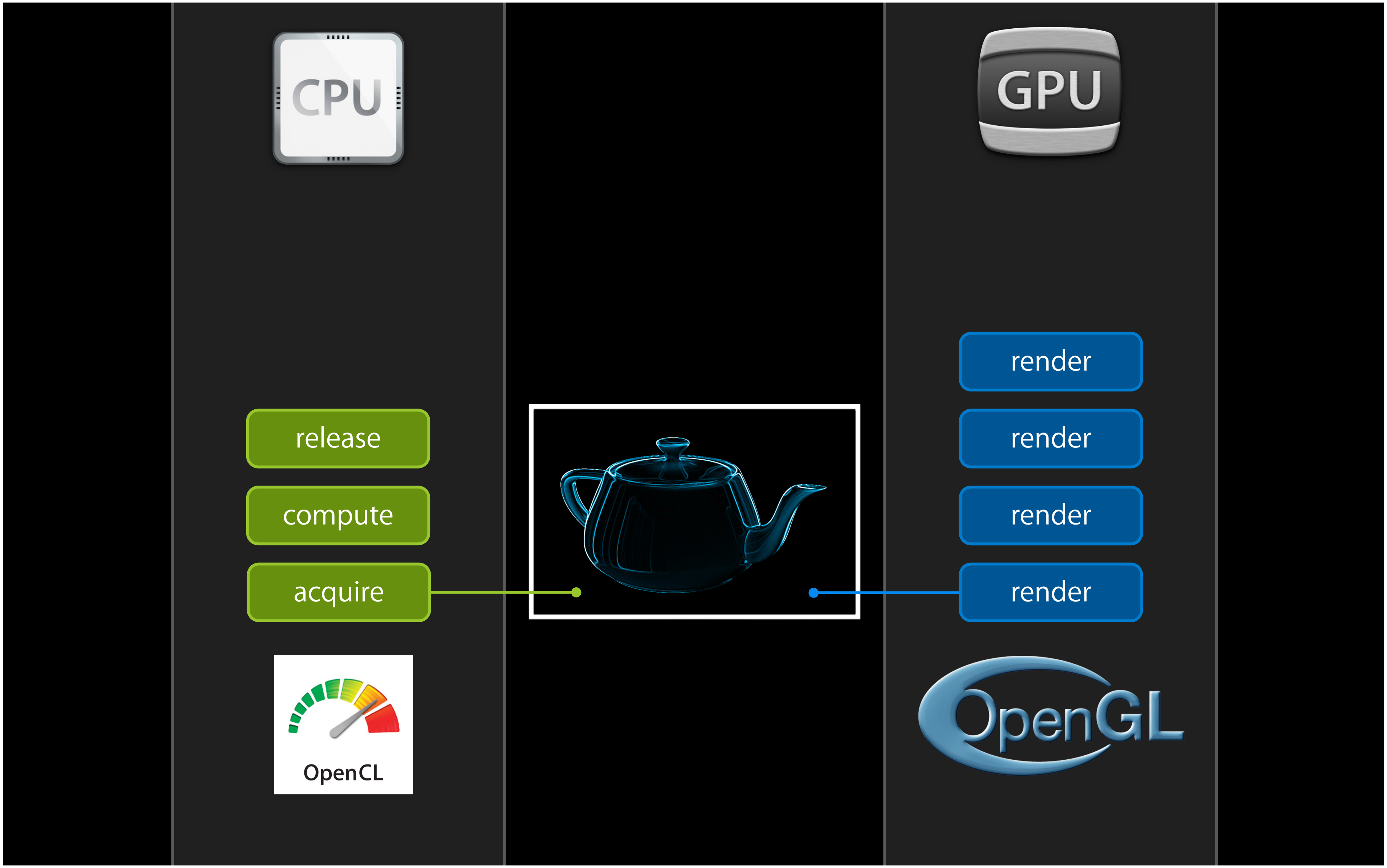
OpenCL

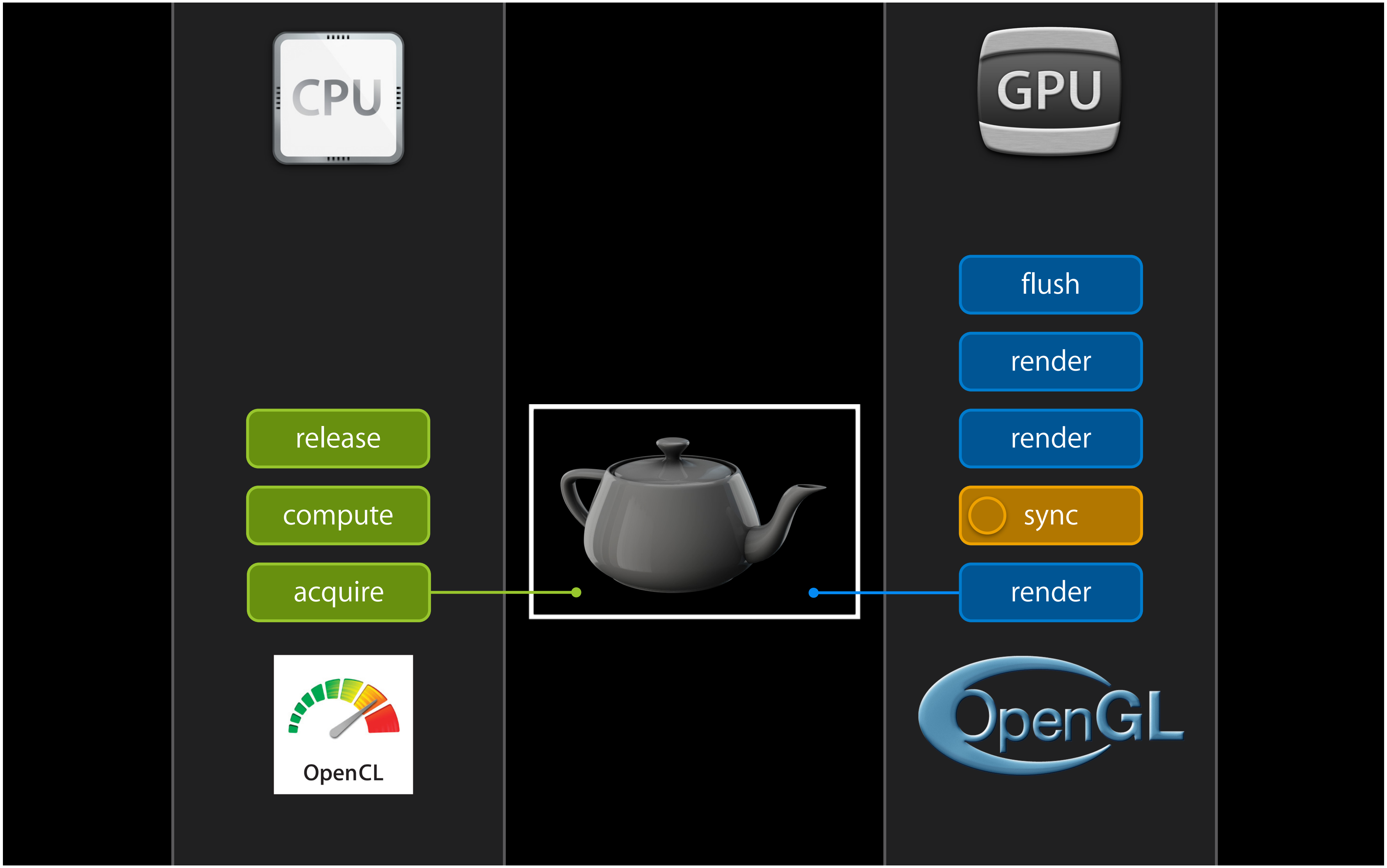
GPU

flush

render

OpenGL


CPU
GPU
render
release
render
compute
render
acquire
render
OpenCL
OpenGL



CPU

release

compute

acquire

OpenCL

GPU

flush

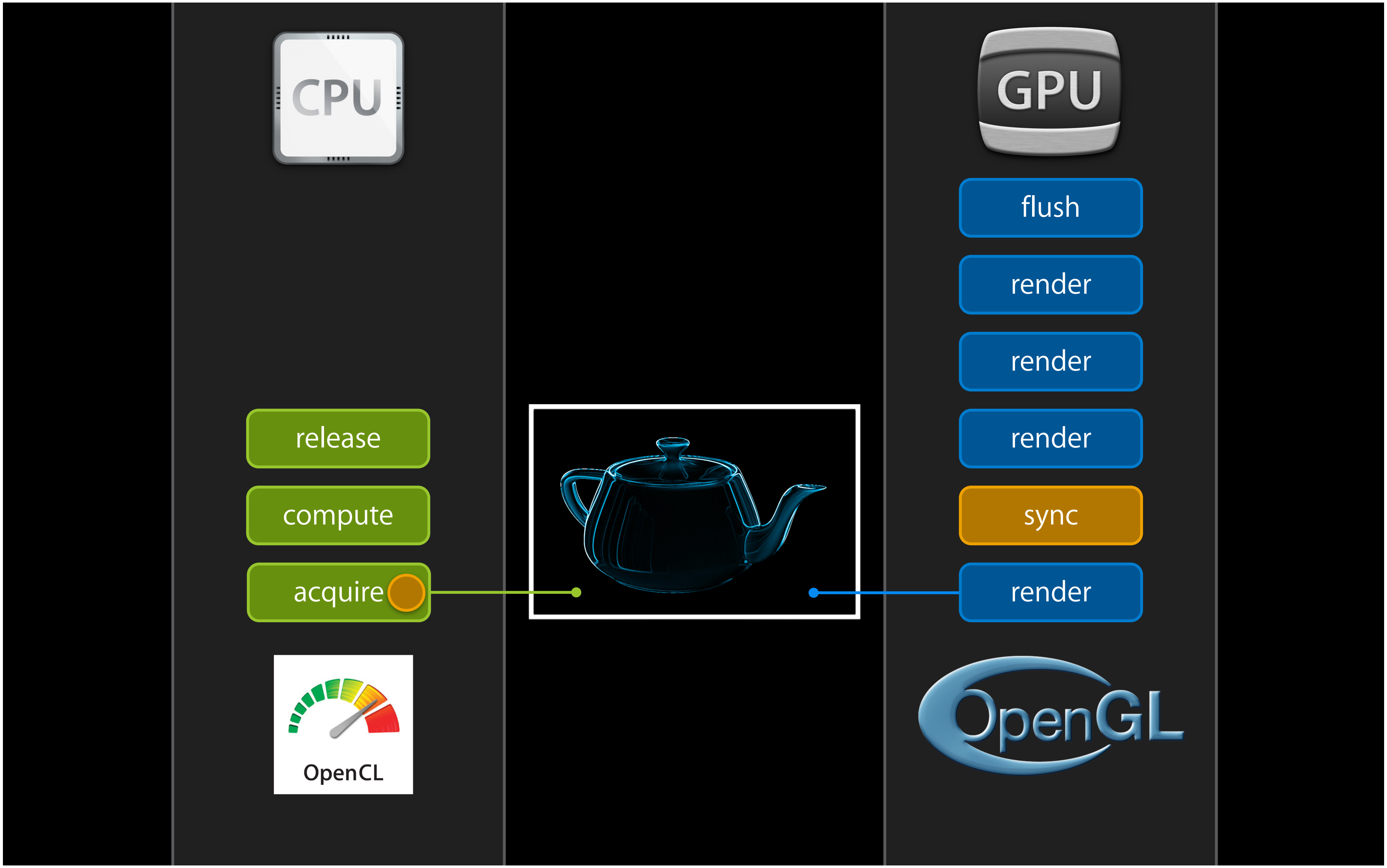
render

render

sync

render

OpenGL



CPU

release

compute

acquire

OpenCL

GPU

flush

render

render

render

sync

render

OpenGL

Synchronization Between CL and GL

New

Render the geometry…

```
drawTeapotToSharedFBO();

GLsync sync = glFenceSync(
    GL_SYNC_GPU_COMMANDS_COMPLETE, 0);

doOtherOpenGLWork();

glFlush();
```

…and then post-process the image

```
event = clCreateEventFromGLsyncKHR(
     cl_context, sync, NULL);
clEnqueueAcquireGLObjects( ..., 1, &event, ... );
```

CL-GL Sharing

Integration with Grand Central Dispatch

- Set the share group first!

```
CGLShareGroupObj sharegroup = CGLGetShareGroup(cgl_context);
gcl_gl_set_sharegroup( sharegroup );
```

- Create your CL objects from GL objects

```
gcl_gl_create_image_from_texture( GL_TEXTURE_2D, 0, my_gl_texture );
gcl_gl_create_image_from_renderbuffer( my_renderbuffer );
gcl_gl_create_ptr_from_buffer( my_gl_buffer );
```

- We handle the acquire/release

Demo

Blue Pony makes some friends

The OpenCL Funhouse

IOSurfaces and OpenCL

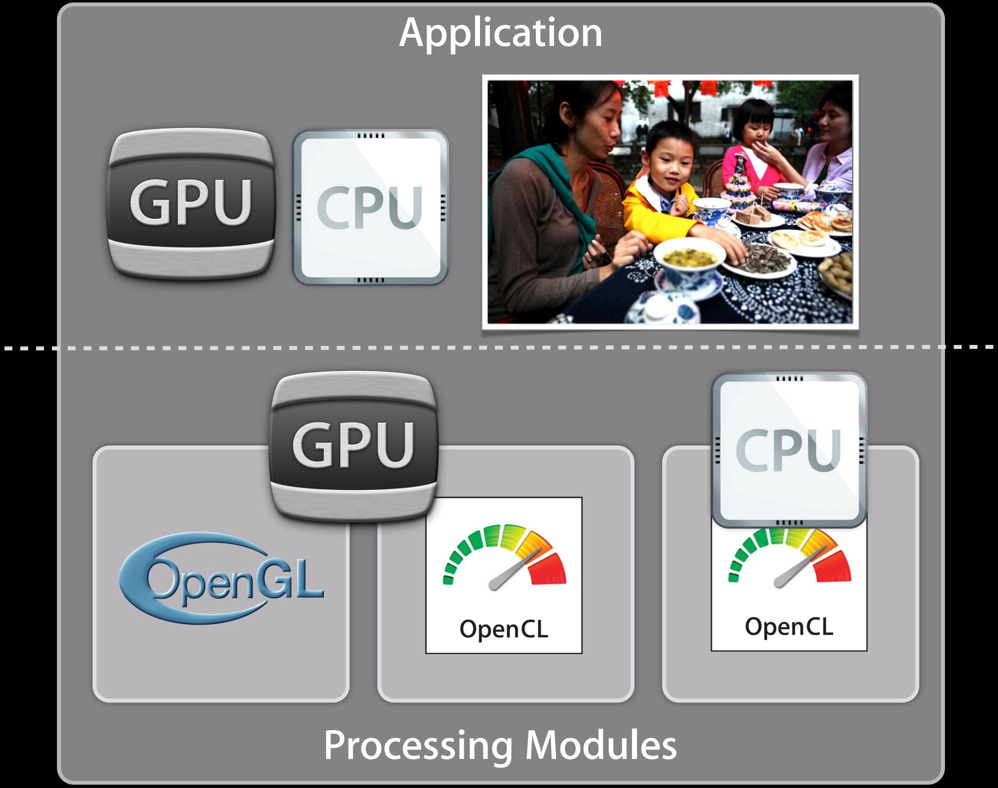
IOSurface Basics

Magic in the machine

- An abstraction for shared image data
- Transcends APIs, architectures, address spaces, and processes

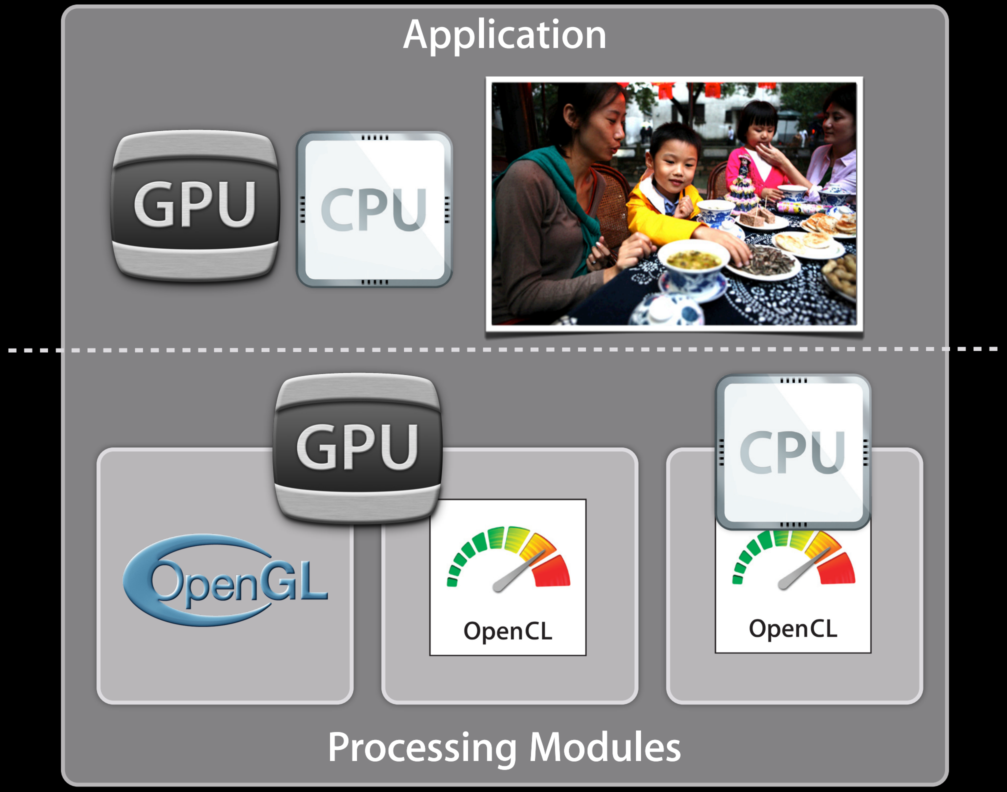
IOSurface

Simplified modularity



IOSurface

Simplified modularity

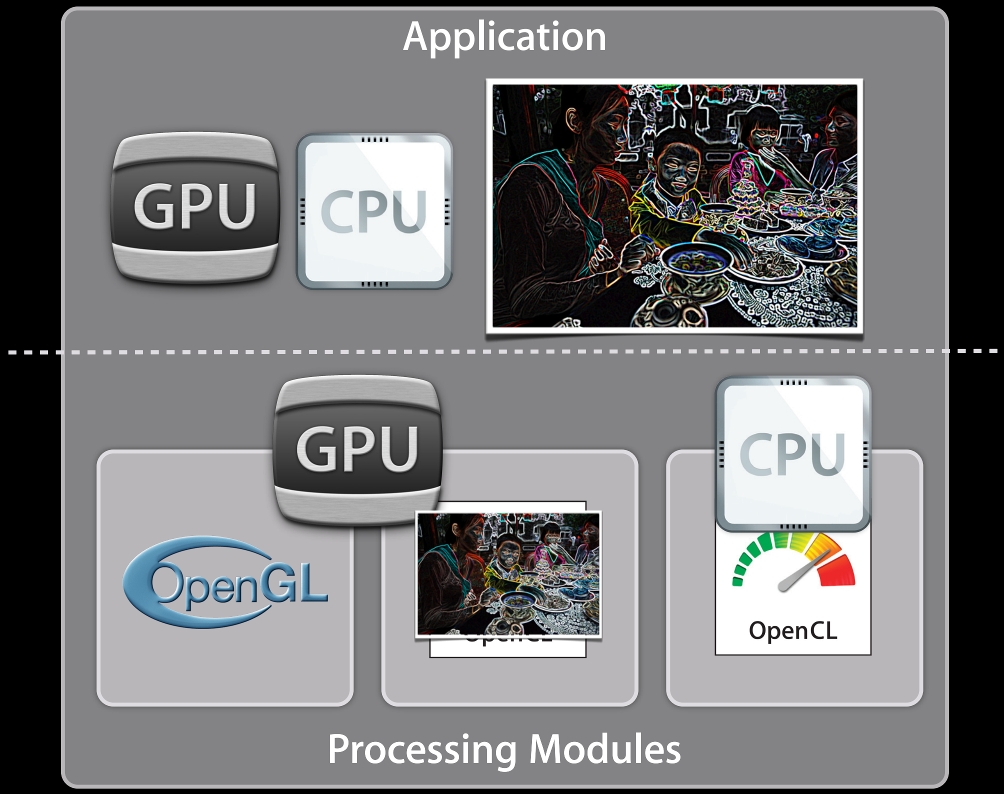


Application

Processing Modules

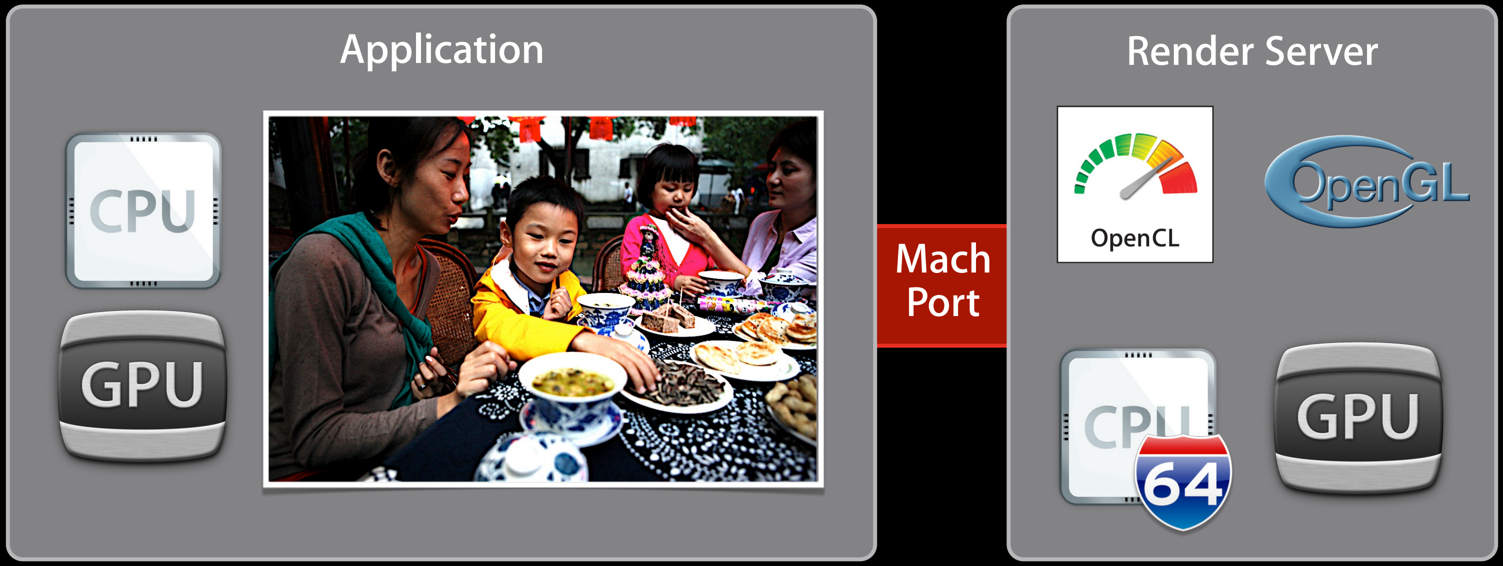
IOSurface

Simplified modularity



IOSurface

Multiple processes



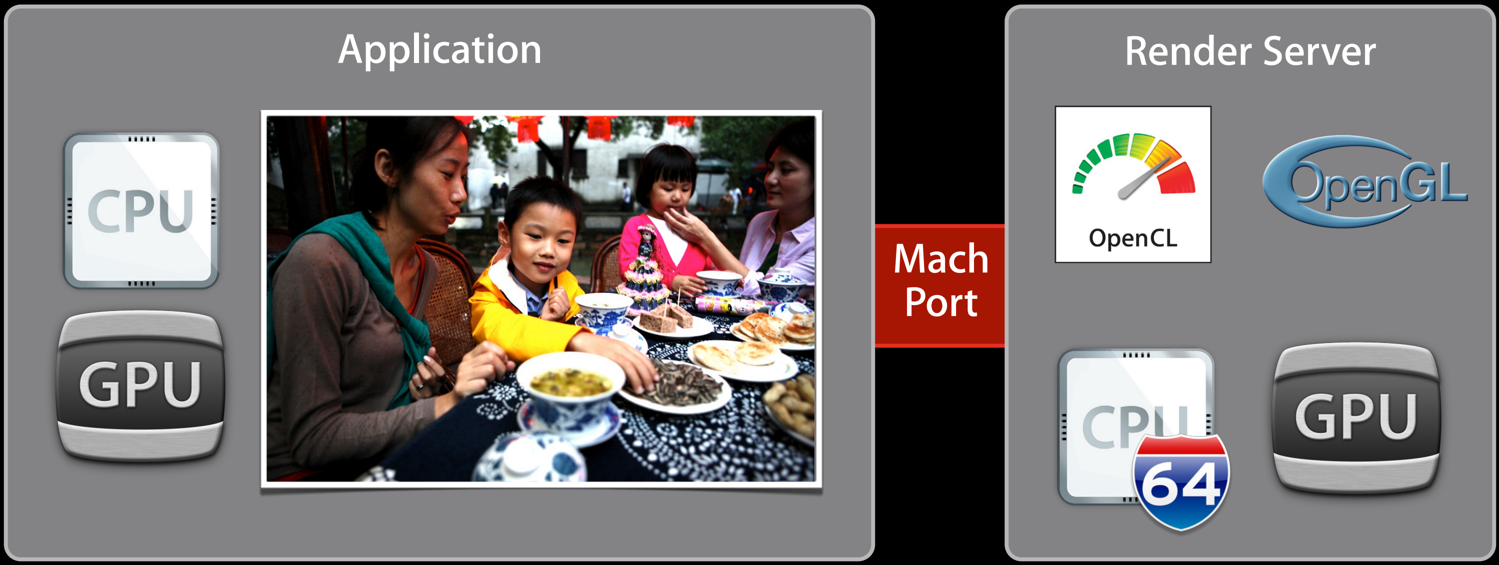
Application

Mach Port

Render Server

IOSurface

Multiple processes



Application

CPU

GPU

Mach Port

Render Server

OpenCL

OpenGL

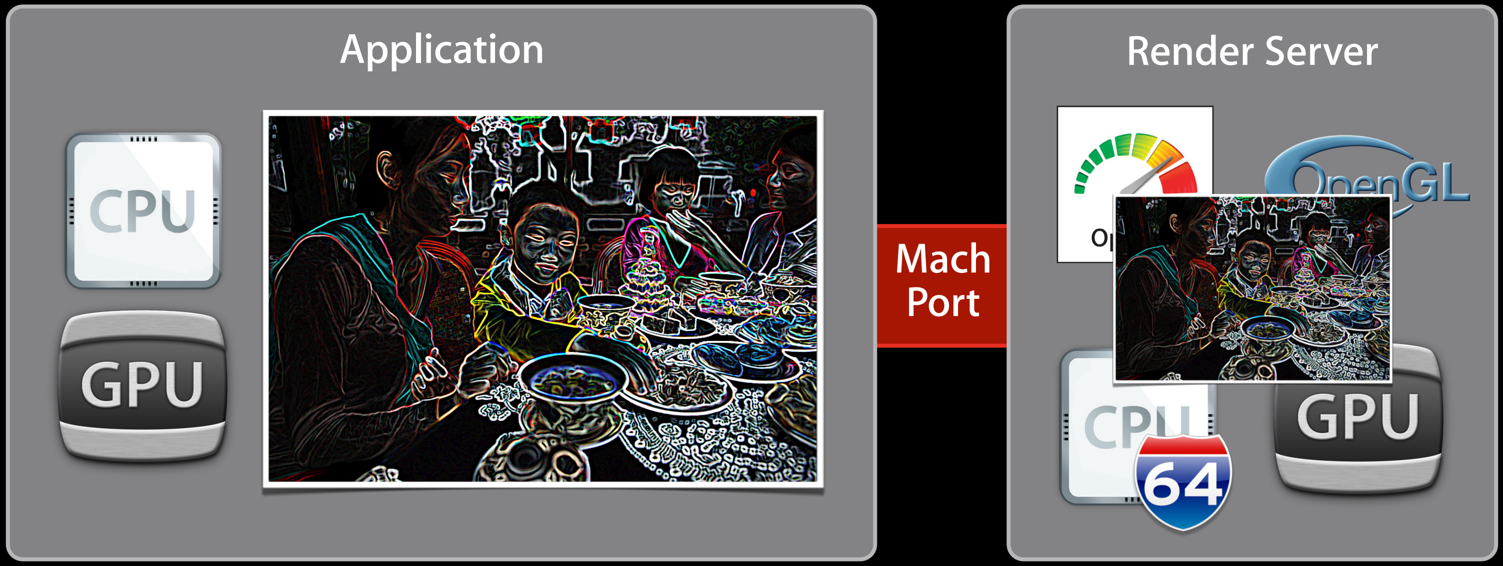
CPU

64

GPU

IOSurface

Multiple processes



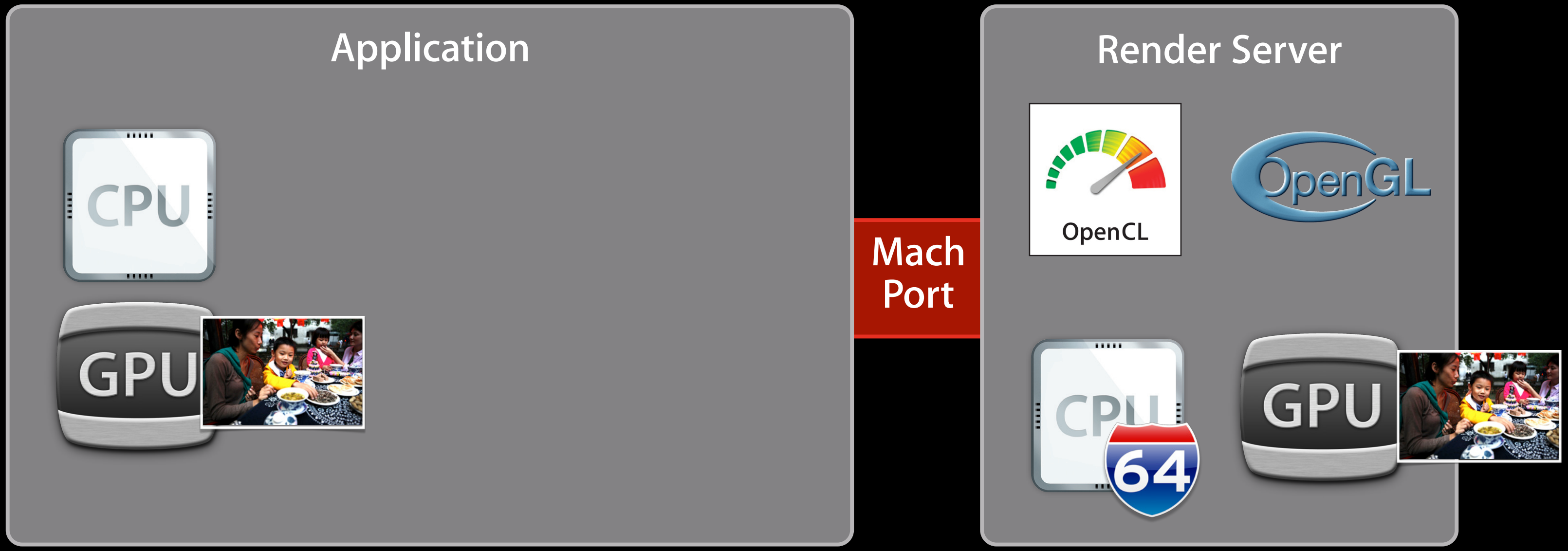
Application

Mach Port

Render Server

IOSurface

Efficiency



Application

Mach Port

Render Server

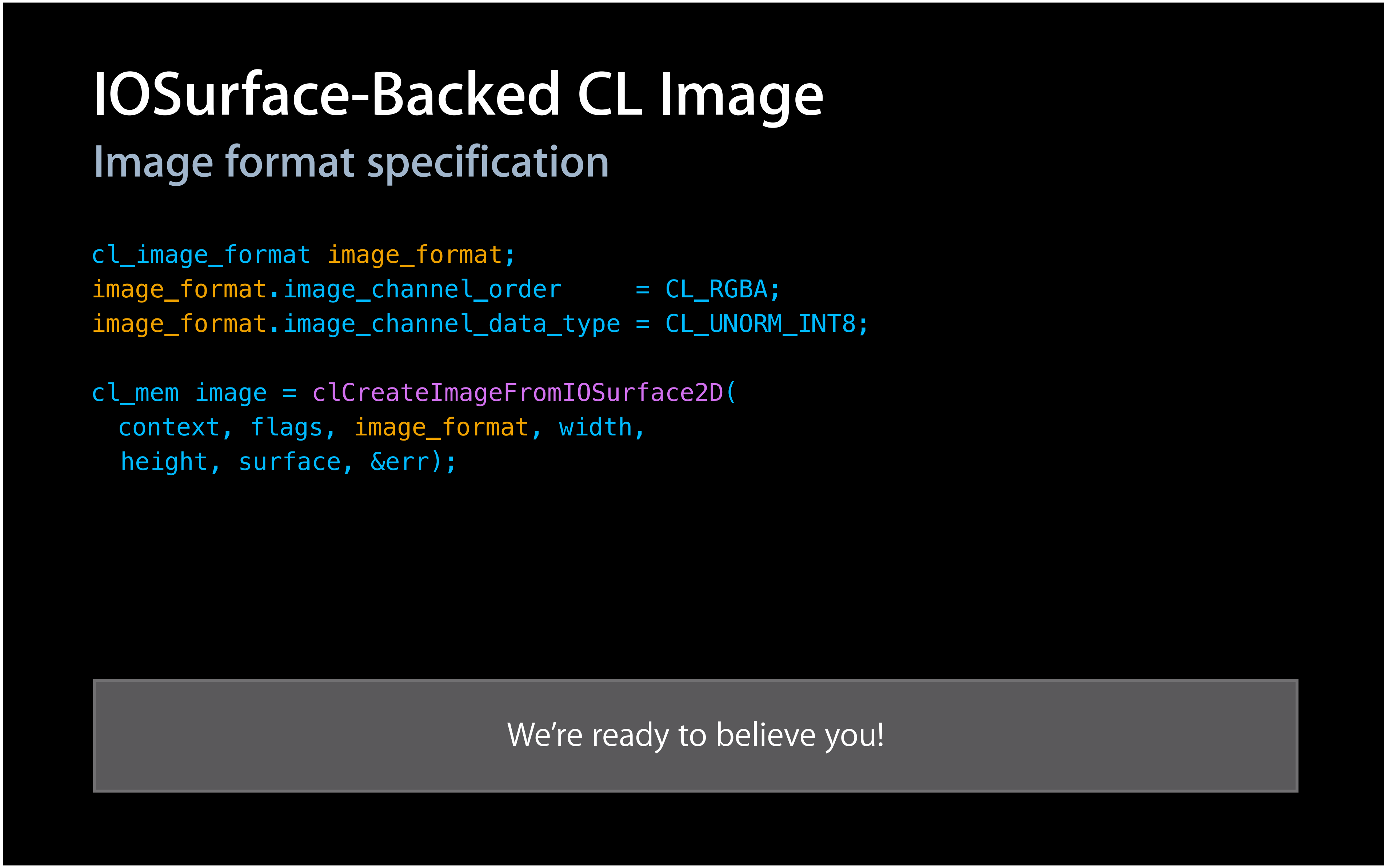
No copy is necessary, no copy is done

IOSurface-Backed CL Image

Creation

New

```
/* An IOSurfaceRef you have created or been given */
IOSurfaceRef surface = ... ;

size_t width  = IOSurfaceGetWidth( surface );
size_t height = IOSurfaceGetHeight( surface );

cl_mem image = clCreateImageFromIOSurface2D(
  context, flags, image_format, width,
  height, surface, &err);
```



IOSurface-Backed CL Image

Image format specification

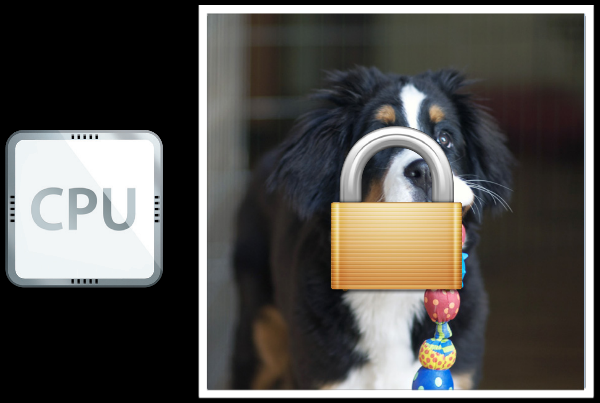
```
cl_image_format image_format;
image_format.image_channel_order      = CL_RGBA;
image_format.image_channel_data_type = CL_UNORM_INT8;

cl_mem image = clCreateImageFromIOSurface2D(
  context, flags, image_format, width,
  height, surface, &err);
```

We’re ready to believe you!

Synchronization Rules

Modifying with the CPU

```
IOSurfaceLock( surface, 0, NULL );
/* do something to surface using the CPU */
IOSurfaceUnlock( surface, 0, NULL );
```



Synchronization Rules

Modifying with the CPU

```
IOSurfaceLock( surface, 0, NULL );
/* do something to surface using the CPU */
IOSurfaceUnlock( surface 0, NULL );

/* We're now free to use the surface-backed image with OpenCL */

clSetKernelArg( 0, &image_from_surface );
clSetKernelArg( 1, &some_other_mem );
clEnqueueNDRangeKernel( ... );
```

Synchronization Rules

Modifying with OpenCL

```
IOSurfaceLock( surface, 0, NULL );
/* do something to surface using the CPU */
IOSurfaceUnlock( surface 0, NULL );


/* We're now free to use the surface-backed image with OpenCL */
~~IOSurfaceLock( surface, 0, NULL );~~
clSetKernelArg( 0, &image_from_surface );
clSetKernelArg( 1, &some_other_mem );
clEnqueueNDRangeKernel( ... );
~~IOSurfaceUnlock( surface, 0, NULL );~~
```

OpenCL

Synchronization Rules

Modifying with OpenCL

```
clSetKernelArg( 0, &image_from_surface );
clEnqueueNDRangeKernel( ... );
clFlush(command_queue);
```

Synchronization Rules

Modifying with OpenCL

```
clSetKernelArg( 0, &image_from_surface );
clEnqueueNDRangeKernel( ... );
clFlush(command_queue);

IOSurfaceLock( surface, kIOSurfaceLockReadOnly, NULL);
/* read as desired using the CPU */
IOSurfaceUnlock( surface kIOSurfaceLockReadOnly, NULL);
```

YUV 4:2:2 IOSurfaces

New

OpenCL image GPU support

- Create as usual

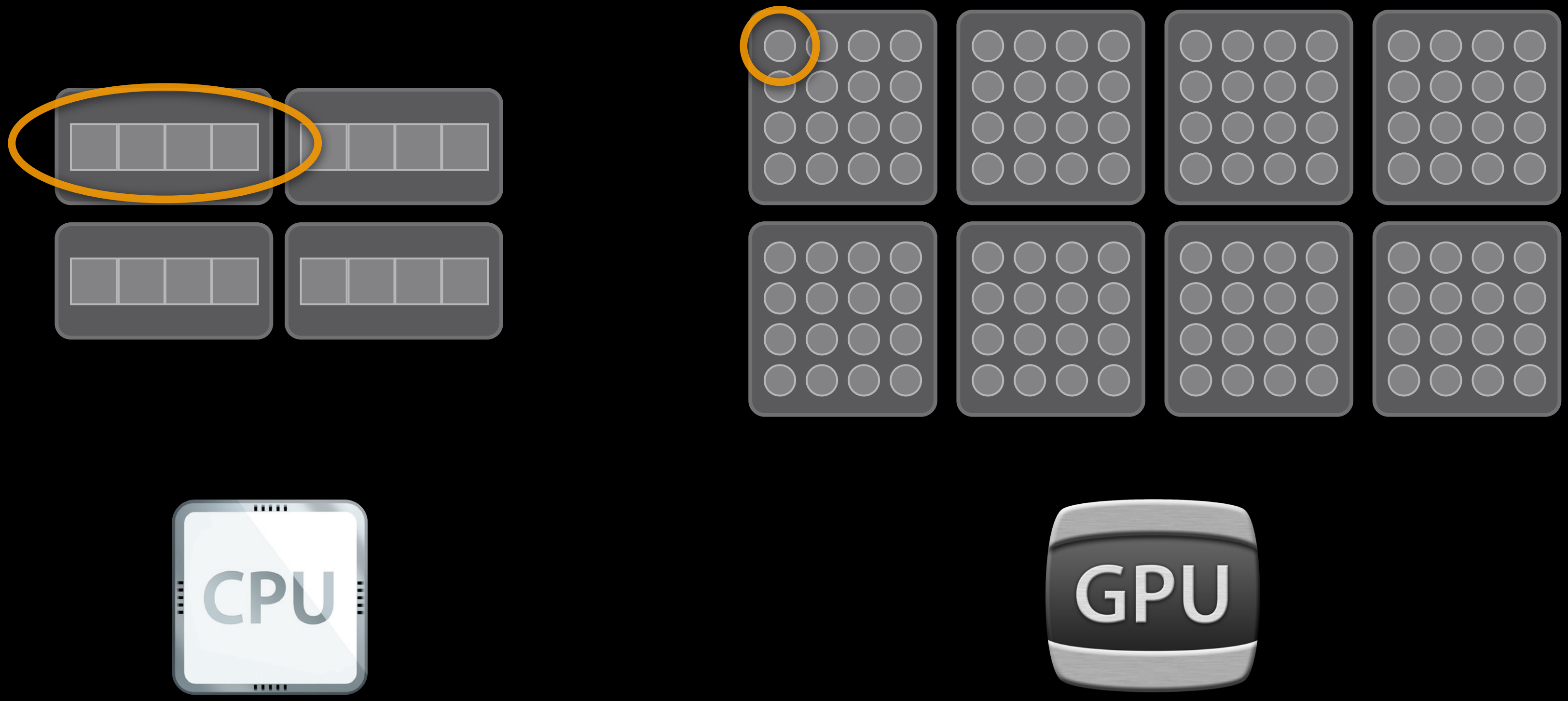
```
cl_image_format format;
format.image_channel_order = CL_YCbYCr_APPLE;
                           = CL_CbYCrY_APPLE;
format.image_data_type     = CL_UNORM_INT8;
                           = CL_UNSIGNED_INT8;
                           = CL_SIGNED_INT8;
```

- Within your CL kernel

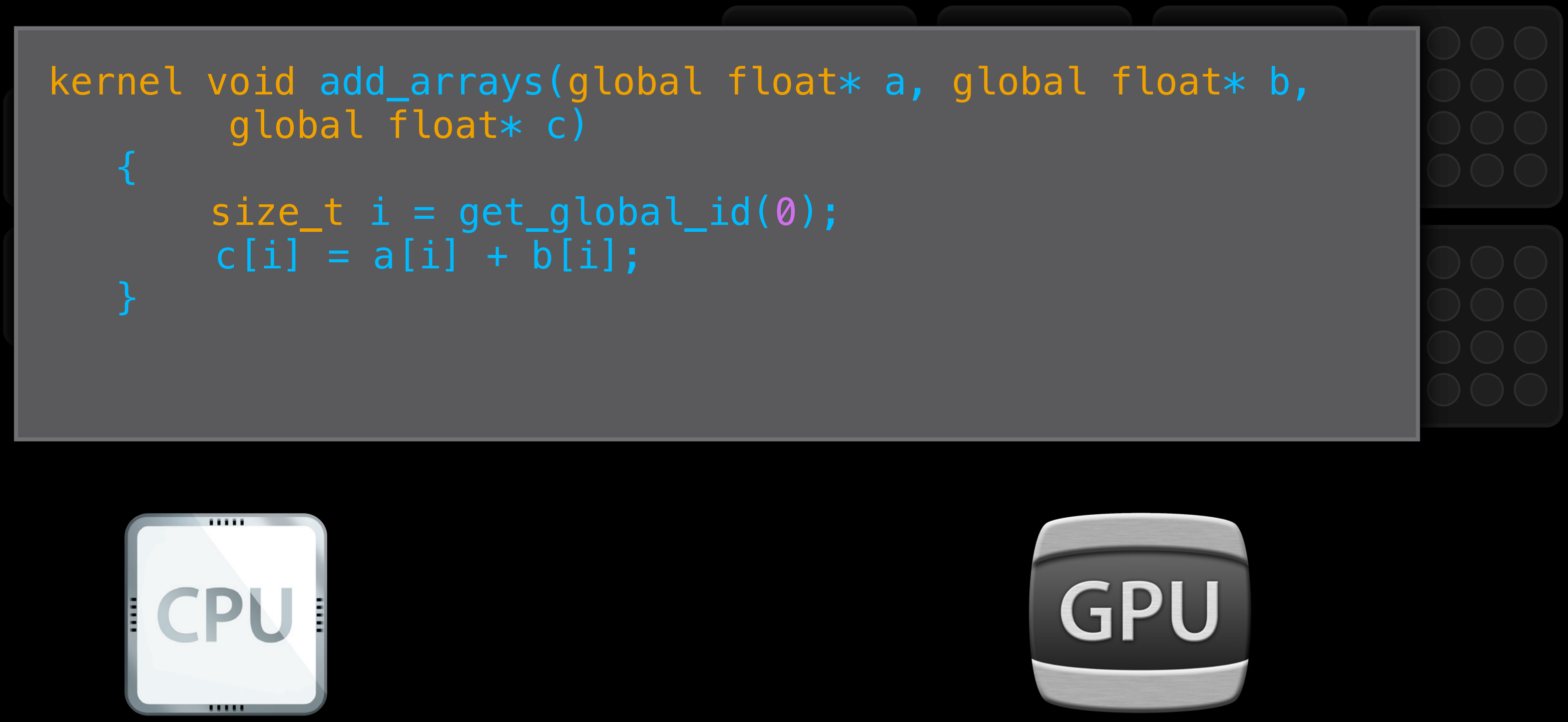
```
pixel = read_image[fui]( image, sampler, coords );
write_image[fui]( image, coords, pixel );
```

Auto-Vectorizer

Parallelism in CPUs vs. GPUs

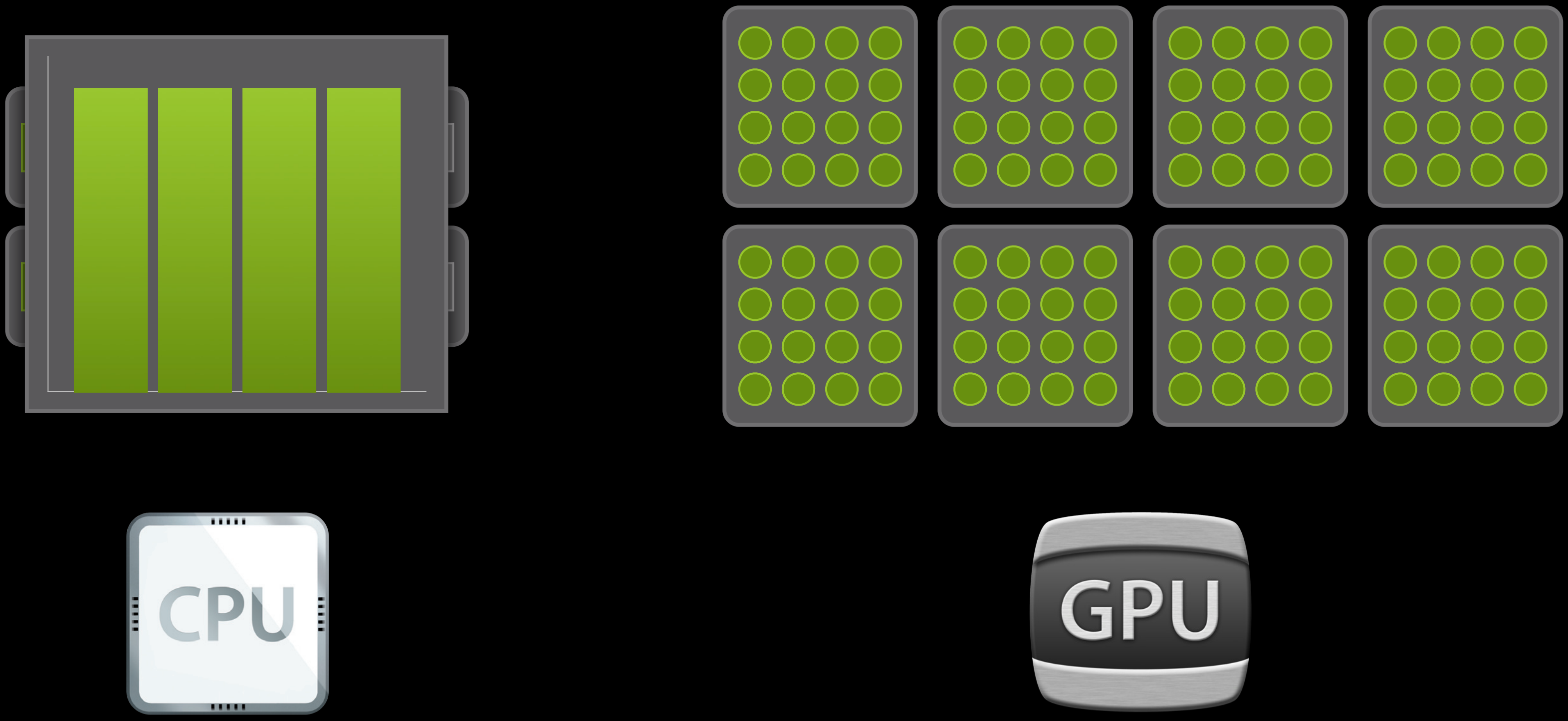


Parallelism in CPUs vs. GPUs

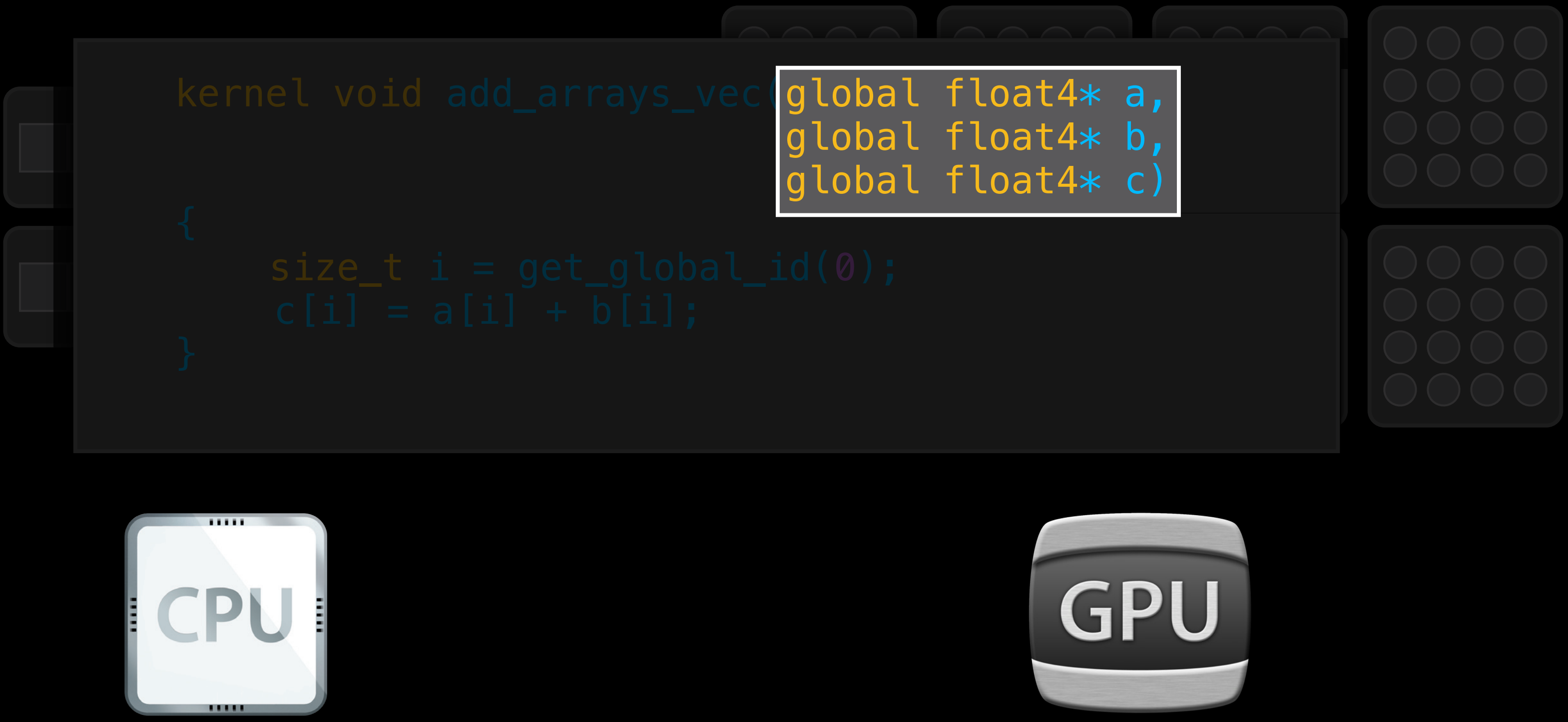


```
kernel void add_arrays(global float* a, global float* b,
        global float* c)
   {
       size_t i = get_global_id(0);
       c[i] = a[i] + b[i];
   }
```

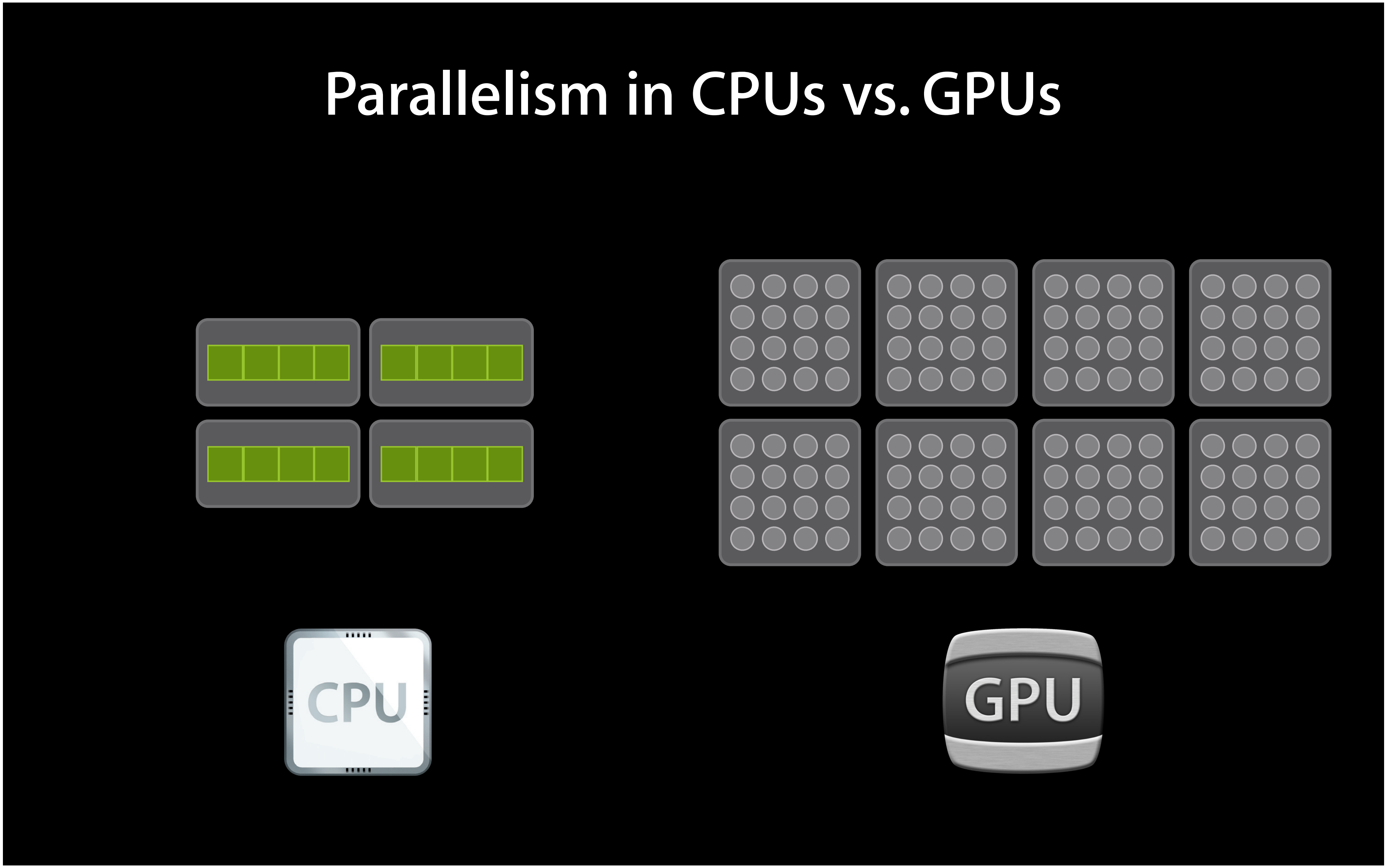
Parallelism in CPUs vs. GPUs



Parallelism in CPUs vs. GPUs



```
kernel void add_arrays_vec(global float4* a,
                           global float4* b,
                           global float4* c)
{
    size_t i = get_global_id(0);
    c[i] = a[i] + b[i];
}
```



Parallelism in CPUs vs. GPUs

Sad, Inescapable Conclusion

We must write multiple kernels

- But I don’t want to write multiple kernels
- But it is easier to write a scalar kernel
- But this should JUST WORK

Intel Auto-Vectorizer

Sion Berkowits
Senior Software Engineer
Intel Corporation

Writing OpenCL Kernels for the CPU

Step 1

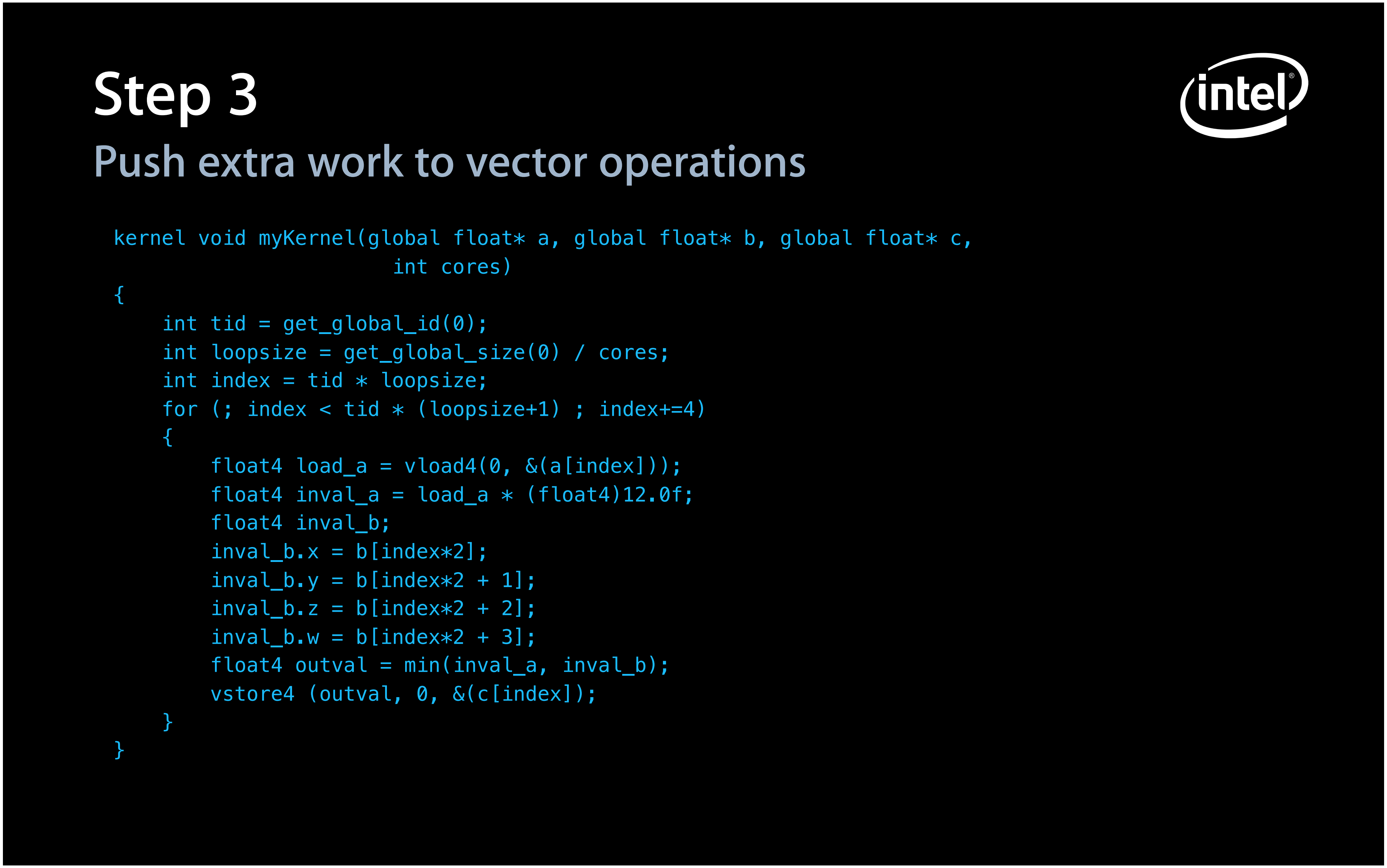
Write a scalar kernel

```
kernel void myKernel(global float* a, global float* b, global float* c)
{
    int tid = get_global_id(0);
    float inval_a = a[tid] * 12.0f;
    float inval_b = b[tid*2];
    float outval = min(inval_a, inval_b);
    c[tid] = outval;
}
```

Step 2

Add loop over several data elements

```
kernel void myKernel(global float* a, global float* b, global float* c,
                       int cores)
{
    int tid = get_global_id(0);
    int loopsize = get_global_size(0) / cores;
    int index = tid * loopsize;
    for (; index < tid * (loopsize+1) ; ++index)
    {
        float inval_a = a[index] * 12.0f;
        float inval_b = b[index*2];
        float outval = min(inval_a, inval_b);
        c[index] = outval;
    }
}
```



Step 3

Push extra work to vector operations

```
kernel void myKernel(global float* a, global float* b, global float* c,
                       int cores)
{
    int tid = get_global_id(0);
    int loopsize = get_global_size(0) / cores;
    int index = tid * loopsize;
    for (; index < tid * (loopsize+1) ; index+=4)
    {
        float4 load_a = vload4(0, &(a[index]));
        float4 inval_a = load_a * (float4)12.0f;
        float4 inval_b;
        inval_b.x = b[index*2];
        inval_b.y = b[index*2 + 1];
        inval_b.z = b[index*2 + 2];
        inval_b.w = b[index*2 + 3];
        float4 outval = min(inval_a, inval_b);
        vstore4 (outval, 0, &(c[index]);
    }
}
```

This Should Be Done Automatically

Intel Auto-Vectorizer

- Runs by default when compiling to CPU
- What it does
  - Packs together work items
  - Generates a loop over entire work group

Achieve speedup of up to 4X without additional effort

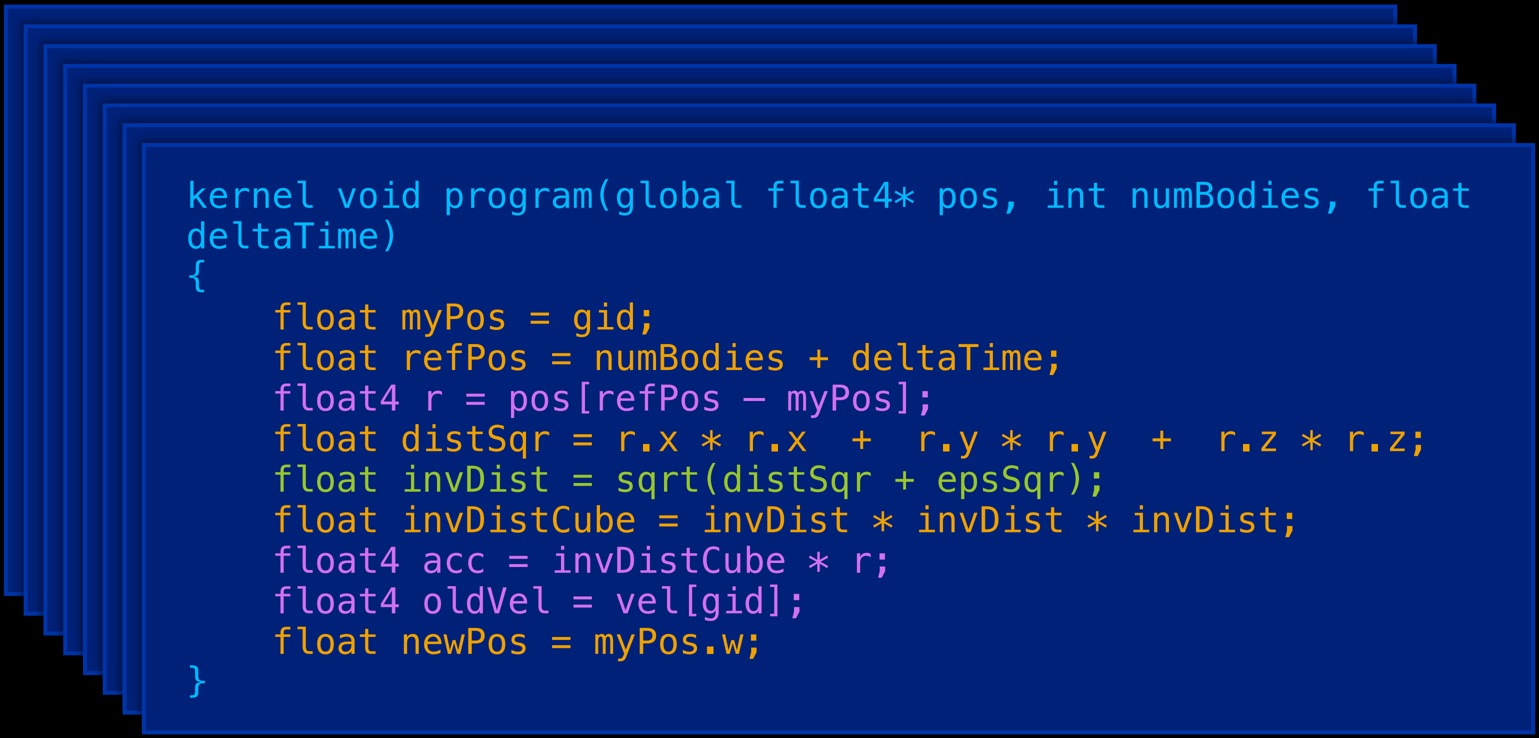
Vectorization Example

OpenCL kernel code

```
kernel void program(global float4* pos, int numBodies, float
deltaTime)
{
    float myPos = gid;
    float refPos = numBodies + deltaTime;
    float4 r = pos[refPos - myPos];
    float distSqr = r.x * r.x  +  r.y * r.y  +  r.z * r.z;
    float invDist = sqrt(distSqr + epsSqr);
    float invDistCube = invDist * invDist * invDist;
    float4 acc = invDistCube * r;
    float4 oldVel = vel[gid];
    float newPos = myPos.w;
}
```

Vectorization Example

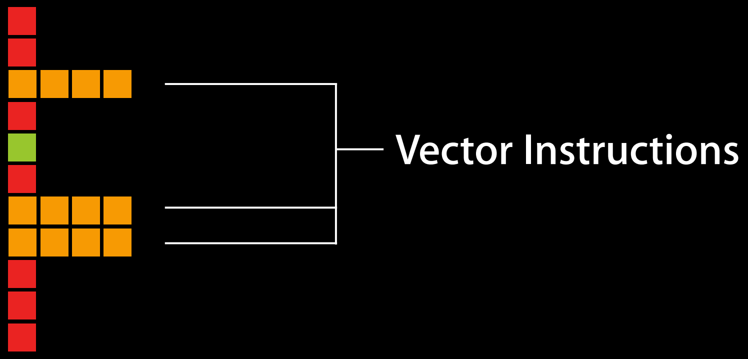
Multiple work items



```
kernel void program(global float4* pos, int numBodies, float
deltaTime)
{
    float myPos = gid;
    float refPos = numBodies + deltaTime;
    float4 r = pos[refPos – myPos];
    float distSqr = r.x * r.x  +  r.y * r.y  +  r.z * r.z;
    float invDist = sqrt(distSqr + epsSqr);
    float invDistCube = invDist * invDist * invDist;
    float4 acc = invDistCube * r;
    float4 oldVel = vel[gid];
    float newPos = myPos.w;
}
```

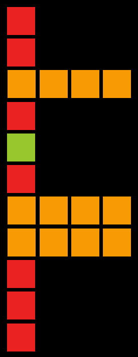
Vectorization Example

Graphic visualization



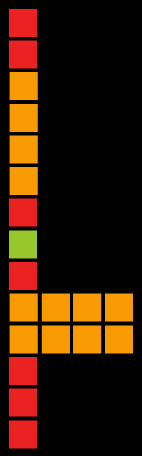
Vectorization Example

Scalarizing code



Vectorization Example

Scalarizing code

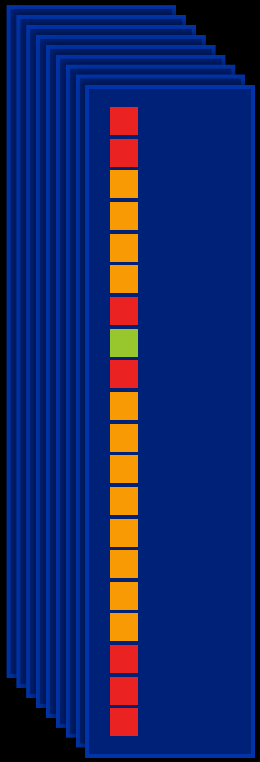


Vectorization Example

Scalarizing code

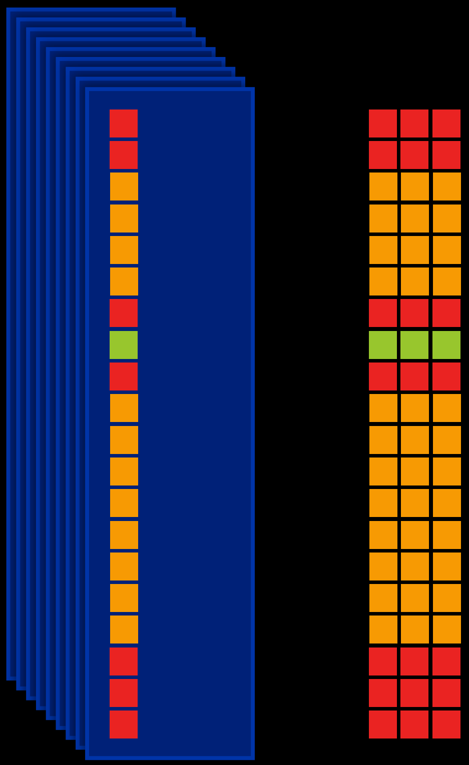
Vectorization Example

Scalarizing code



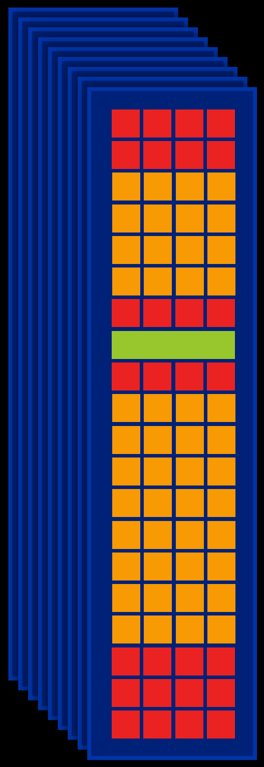
Vectorization Example

Packetizing code



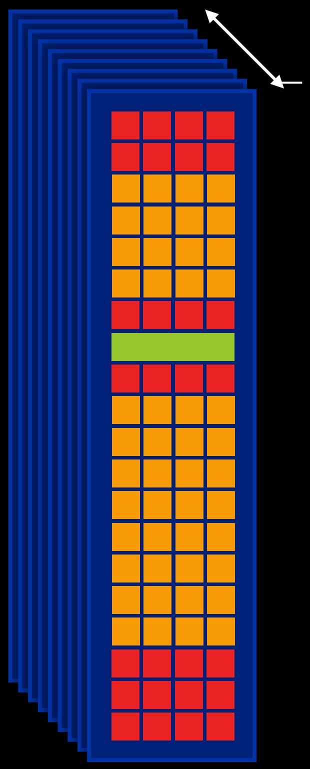
Vectorization Example

Packetizing code



Vectorization Example

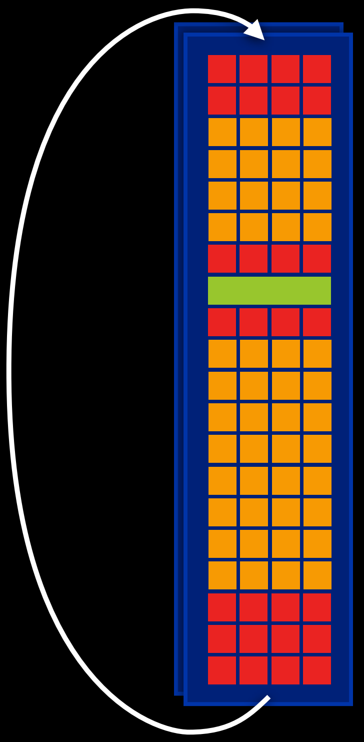
Packetizing code



Reduced amount of invocations

Vectorization Example

Loop over entire work-group



Reduced amount of invocations

Writing Optimal Code for CPU

- Do
  - Write once—run everywhere
    - Same code will be optimal on different CPU architectures (SSE and AVX)
- Don't
  - No need to write device-specific optimizations
  - Minimize work-item ID dependent control flow

Let the Auto-Vectorizer do the work for you!

Demo

Wrapping Up

James Shearer
Apple OpenCL Team

Summary

- Using OpenCL is easier than ever in Lion
    - Integration with Xcode and Grand Central Dispatch
    - Auto-Vectorizer
- Offline compiler
- Easy and efficient sharing
- IOSurface walks through walls

More Information

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

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

Related Sessions

Advances in OpenGL for Mac OS X Lion	Mission Thursday 10:15AM

Labs

OpenCL Lab	Graphics, Media, & Games Lab C Tuesday 2:00PM

WWDC2011