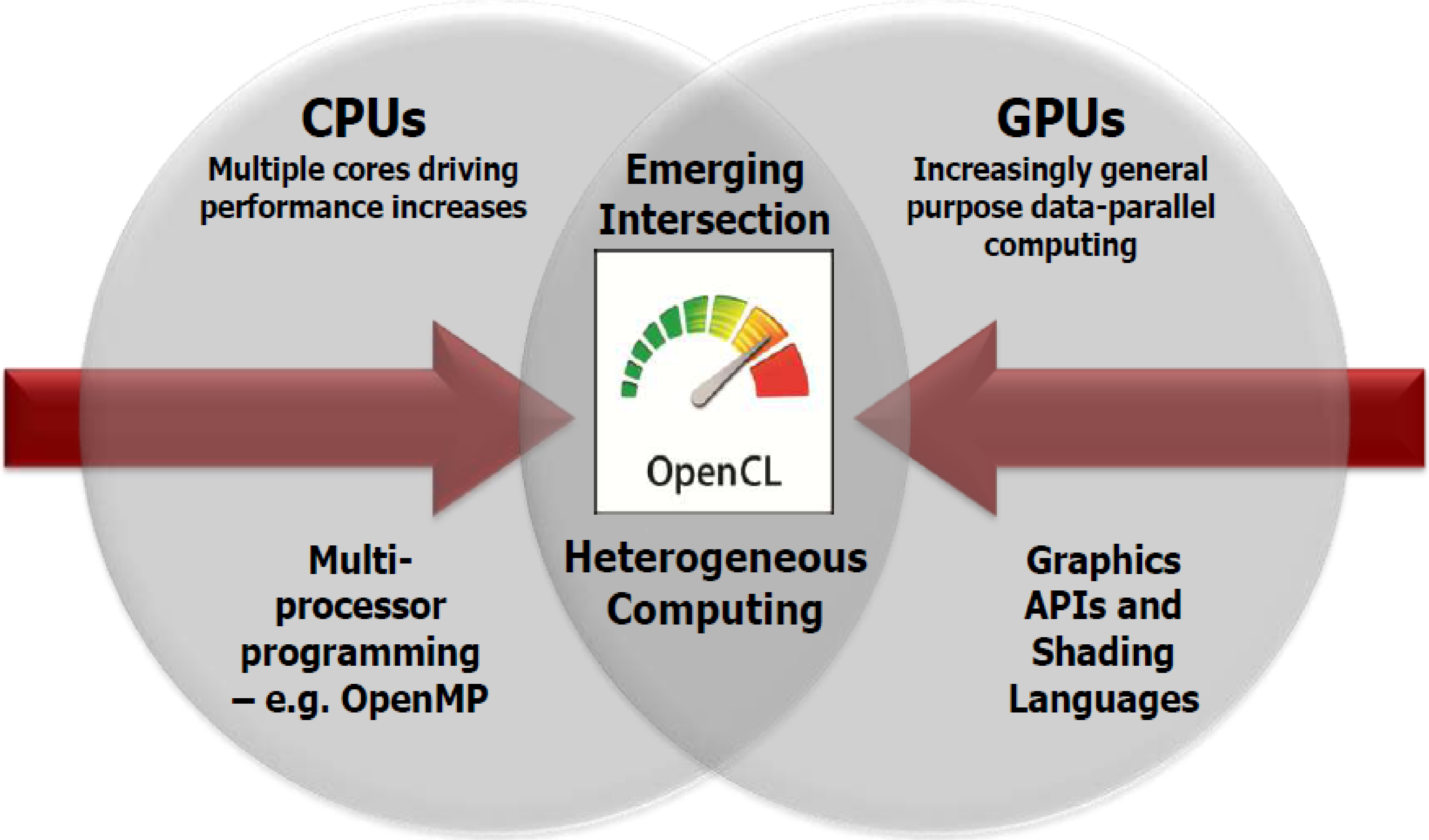
Introduction and Overview

HPC Course

Excerpt of slides © KHRONOS GROUP

Processor Parallelism



CPUs

Multiple cores driving performance increases

Emerging Intersection

GPUs

Increasingly general purpose data-parallel computing

OpenCL

Multi-processor programming – e.g. OpenMP

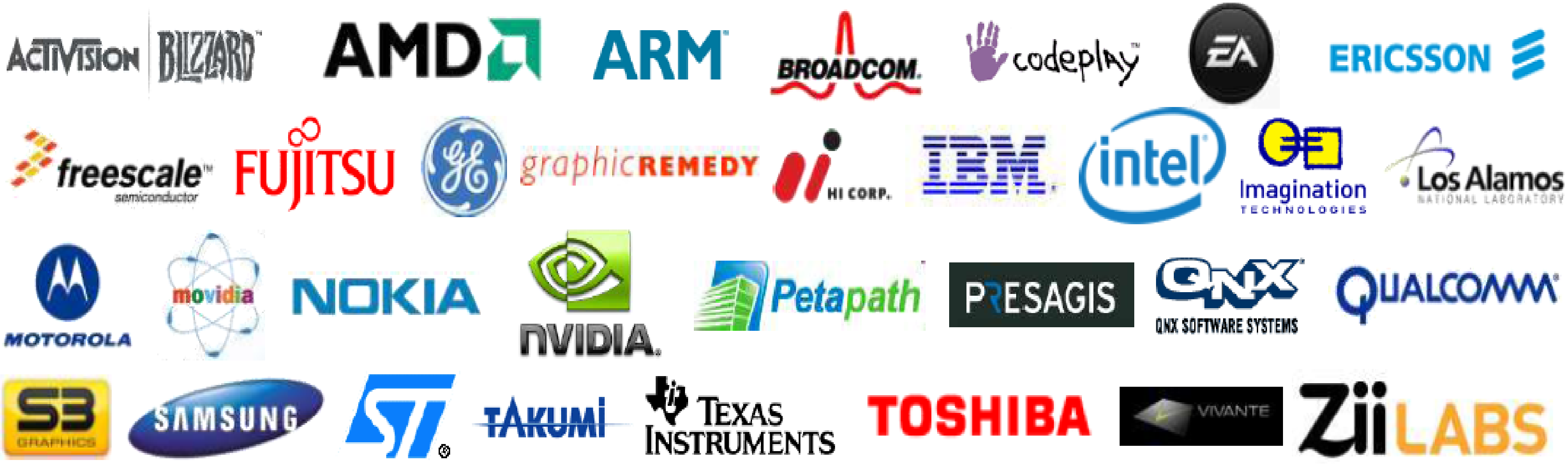
Heterogeneous Computing

Graphics APIs and Shading Languages

OpenCL is a programming framework for heterogeneous compute resources

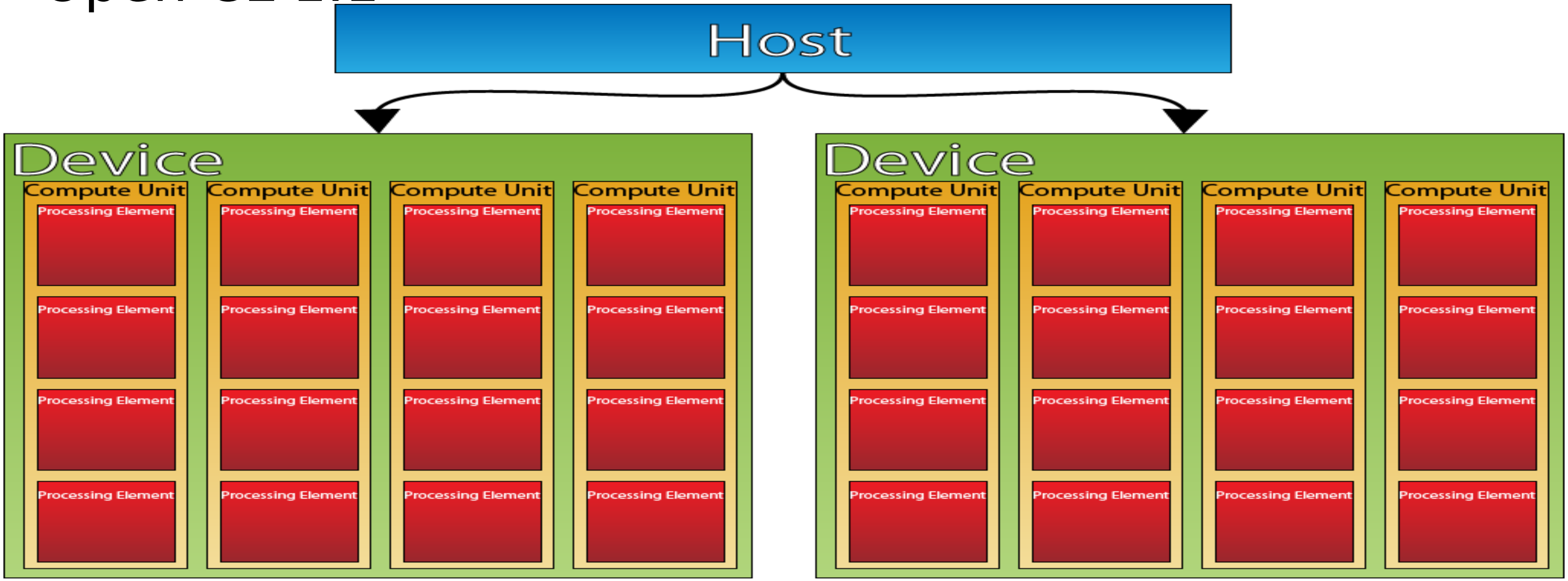
OpenCL Working Group

- **Diverse industry participation**
  - Processor vendors, system OEMs, middleware vendors, application developers
- **Many industry-leading experts involved in OpenCL's design**
  - A healthy diversity of industry perspectives
- **Apple made initial proposal and is very active in the working group**
  - Serving as specification editor



OpenCL Timeline

2008 – Introduction of OpenCL by Apple
2008 – (Dec.) First specification of Open CL 1.0 by Khronos
2010 – Specification release and implementation ship Open CL 1.1

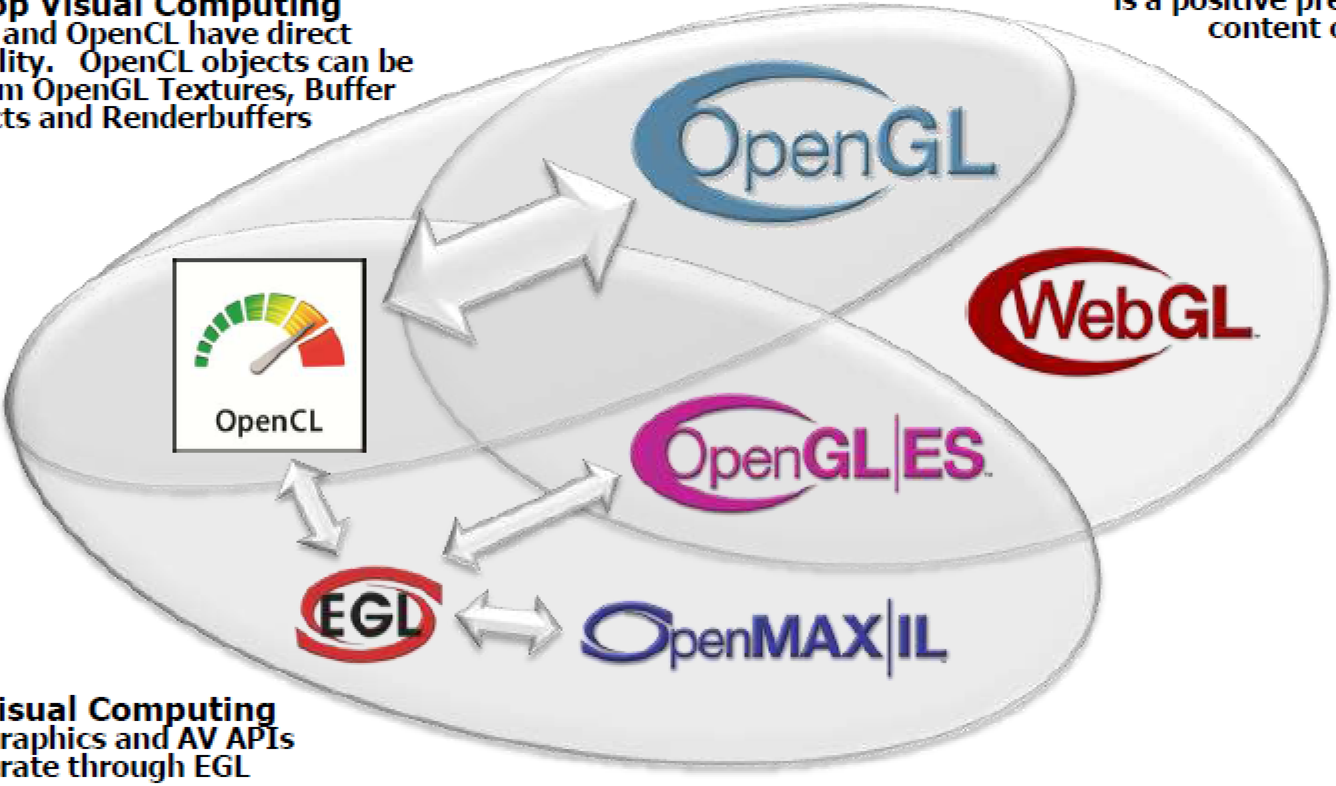


OpenCL

OpenGL-based Ecosystem

Roadmap Convergence
OpenGL 4.0 and OpenGL ES 2.0 are both streamlined, programmable pipelines. GL and ES working groups are working on convergence. WebGL is a positive pressure for portable 3D content on all platforms

Desktop Visual Computing
OpenGL and OpenCL have direct interoperability. OpenCL objects can be created from OpenGL Textures, Buffer Objects and Renderbuffers



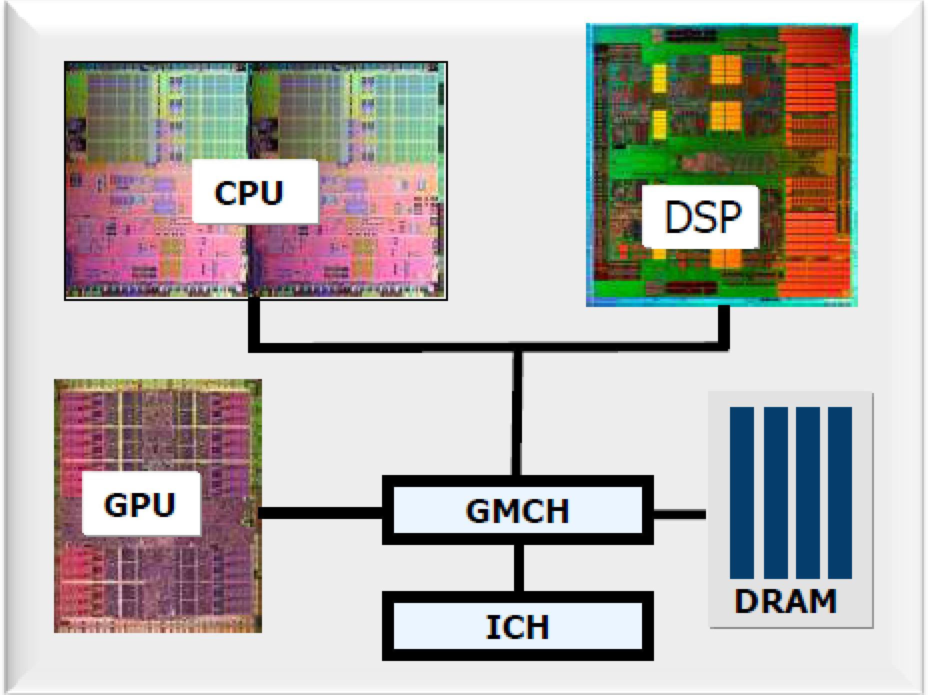
Mobile Visual Computing
Compute, graphics and AV APIs interoperate through EGL

OpenCL Overview

It's a Heterogeneous World

- **A modern platform Includes:**
  - One or more CPUs
  - One or more GPUs
  - DSP processors
  - ... other?

OpenCL lets Programmers write a single portable program that uses ALL resources in the heterogeneous platform



GMCH = graphics memory control hub

ICH = Input/output control hub

The BIG Idea behind OpenCL

- **OpenCL execution model …**
  execute a kernel at each point in a problem domain
  - E.g., process a 1024 x 1024 image with one kernel invocation per pixel or 1024 x 1024 = 1,048,576 kernel executions

Traditional loops

```
void
trad_mul(int n,
         const float *a,
         const float *b,
         float *c)
{
  int i;
  for (i=0; i<n; i++)
    c[i] = a[i] * b[i];          }
```

Data Parallel OpenCL

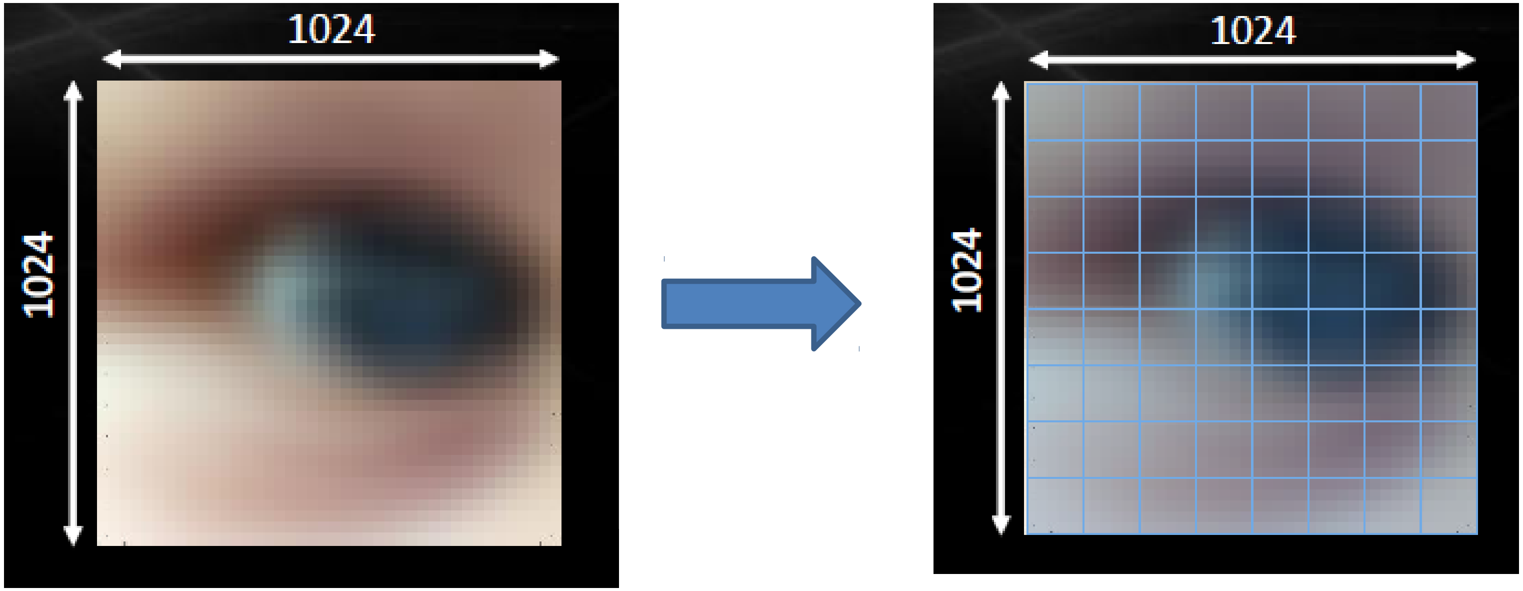
```
kernel void
dp_mul(global const float *a,
       global const float *b,
       global float *c)
{
  int id = get_global_id(0);

  c[id] = a[id] * b[id];

} // execute over "n" work-items
```

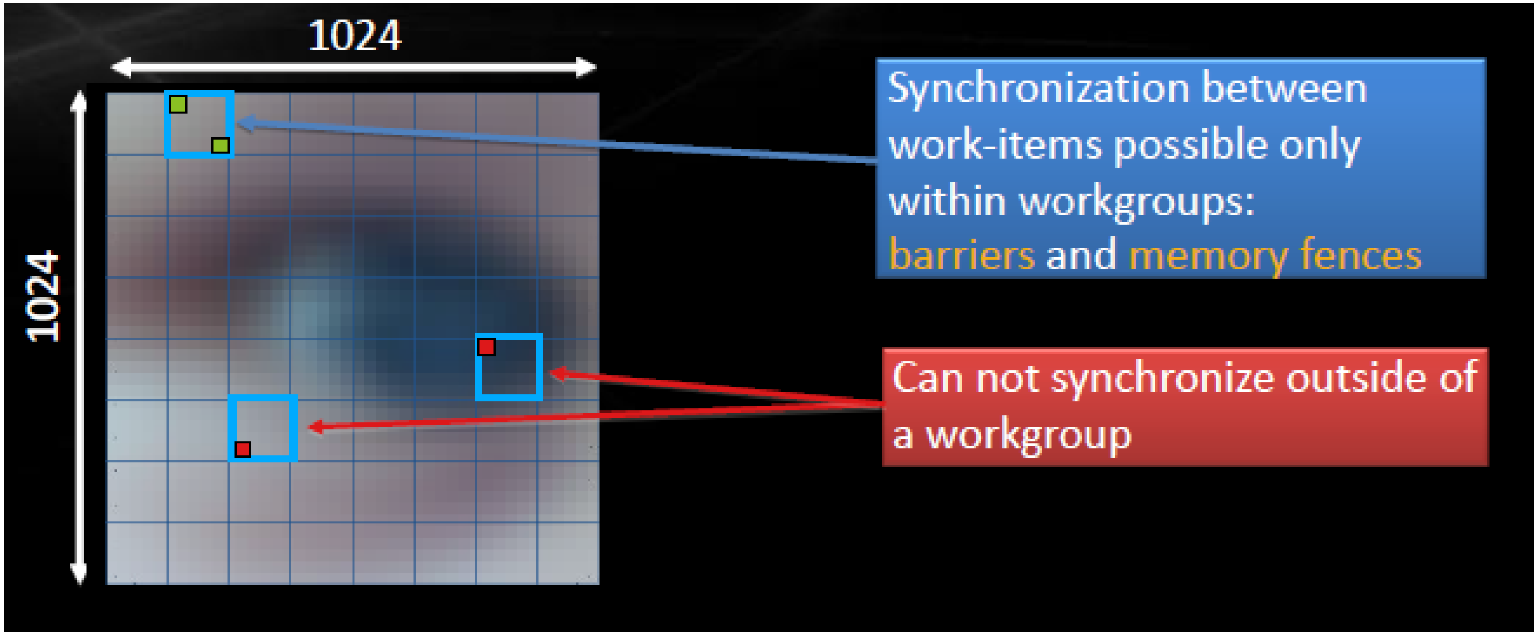
An N-dimension domain of work-items

- **Define the "best" N-dimensioned index space for your algorithm**
  - Global Dimensions: 1024 x 1024 (whole problem space)
  - Local Dimensions: 128 x 128 (work group … executes together)



An N-dimension domain of work-items

- **Define the "best" N-dimensioned index space for your algorithm**
  - Global Dimensions: 1024 x 1024 (whole problem space)
  - Local Dimensions: 128 x 128 (work group ... executes together)



Synchronization between work-items possible only within workgroups: barriers and memory fences

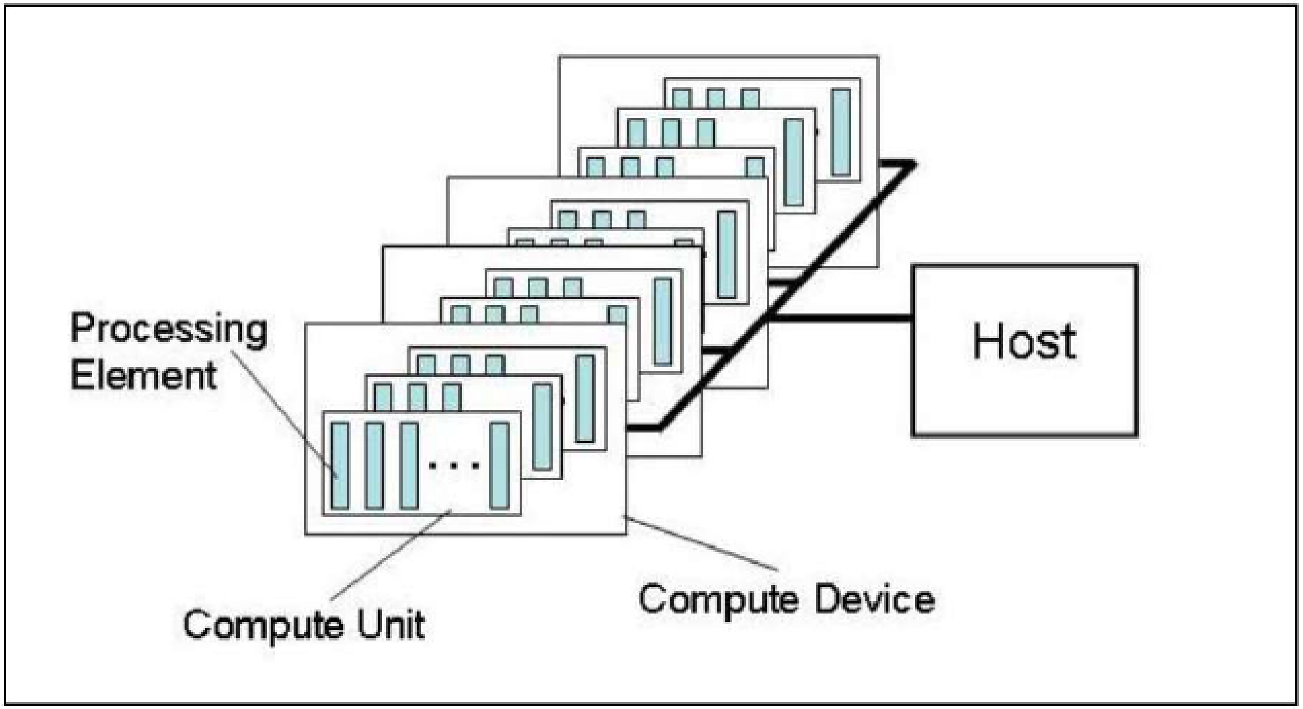
Can not synchronize outside of a workgroup

To use OpenCL, you must

- Define the platform
- Execute code on the platform
- Move data around in memory
- Write (and build) programs

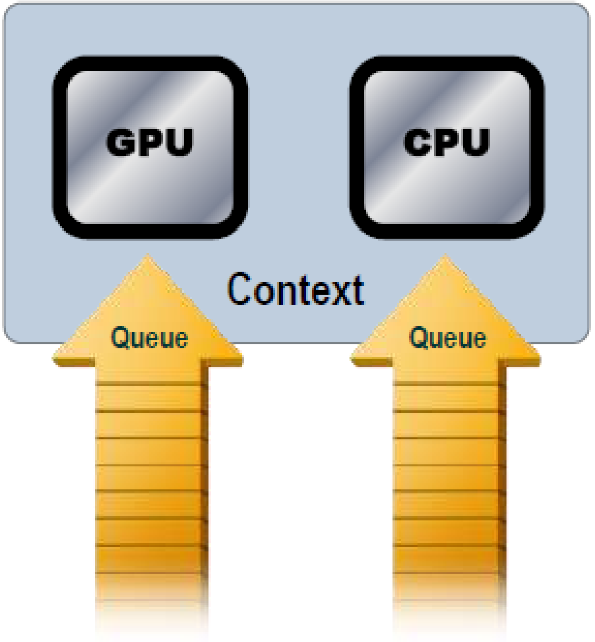
OpenCL Platform Model

- **One Host + one or more Compute Devices**
  - Each Compute Device is composed of one or more Compute Units
    - Each Compute Unit is further divided into one or more Processing Elements



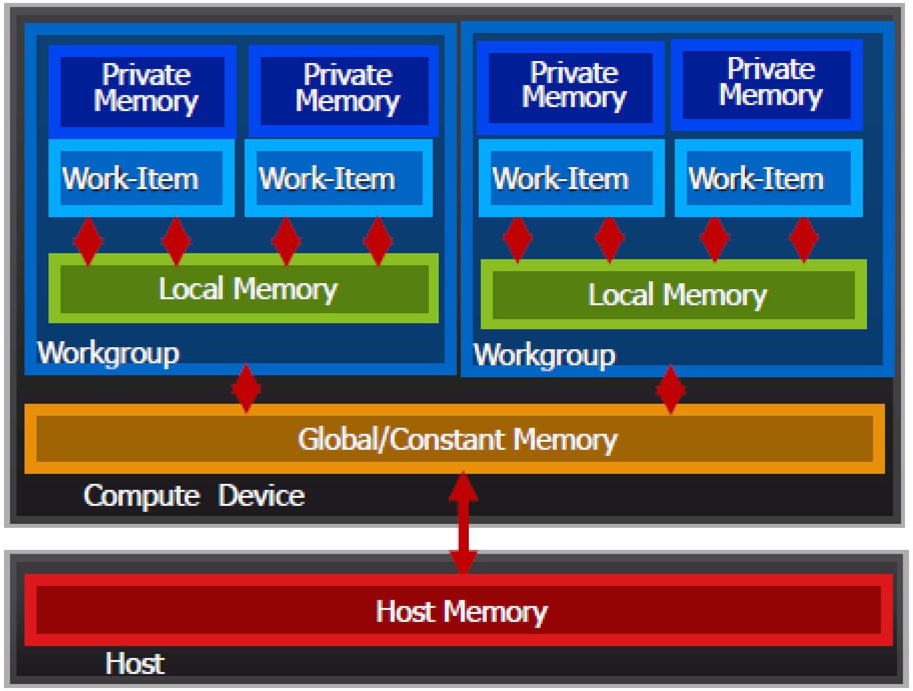
OpenCL Execution Model

- **An OpenCL application runs on a host which submits work to the compute devices**
  - **Work item**: the basic unit of work on an OpenCL device
  - **Kernel**: the code for a work item. Basically a C function
  - **Program**: Collection of kernels and other functions (Analogous to a dynamic library)
  - **Context**: The environment within which work-items executes … includes devices and their memories and command queues
- **Applications queue kernel execution instances**
  - Queued in-order … one queue to a device
  - Executed in-order or out-of-order



OpenCL Memory Model

- **Private Memory**
  - –Per work-item
- **Local Memory**
  - –Shared within a workgroup
- **Global/Constant Memory**
  - –Visible to all workgroups
- **Host Memory**
  - –On the CPU



Memory management is Explicit

You must move data from host -> global -> local … *and* back

Programming kernels: OpenCL C Language

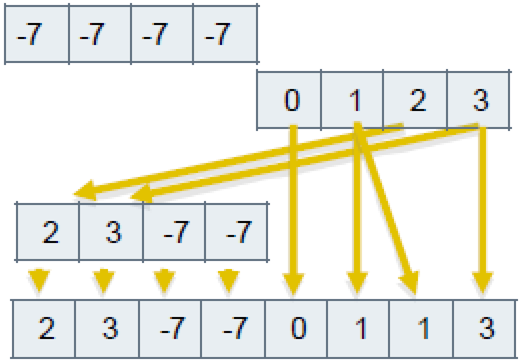
- **A subset of ISO C99**
  - But without some C99 features such as standard C99 headers, function pointers, recursion, variable length arrays, and bit fields
- **A superset of ISO C99 with additions for:**
  - Work-items and workgroups
  - Vector types
  - Synchronization
  - Address space qualifiers
- **Also includes a large set of built-in functions**
  - Image manipulation
  - Work-item manipulation,
  - Specialized math routines, etc.

Programming Kernels: Data Types

- **Scalar data types**
  - char , uchar, short, ushort, int, uint, long, ulong, float
  - bool, intptr_t, ptrdiff_t, size_t, uintptr_t, void, half (storage)
- **Image types**
  - image2d_t, image3d_t, sampler_t
- **Vector data types**
  - Vector lengths 2, 4, 8, & 16 (char2, ushort4, int8, float16, double2, ...)
  - Endian safe
  - Aligned at vector length
  - Vector operations
  - Built-in functions

Double is an optional type in OpenCL 1.0

```
int4 vi0 = (int4) -7;
int4 vi1 = (int4)(0, 1, 2, 3);
vi0.lo = vi1.hi;
int8 v8 = (int8)(vi0, vi1.s01, vi1.odd);
```



-7 -7 -7 -7

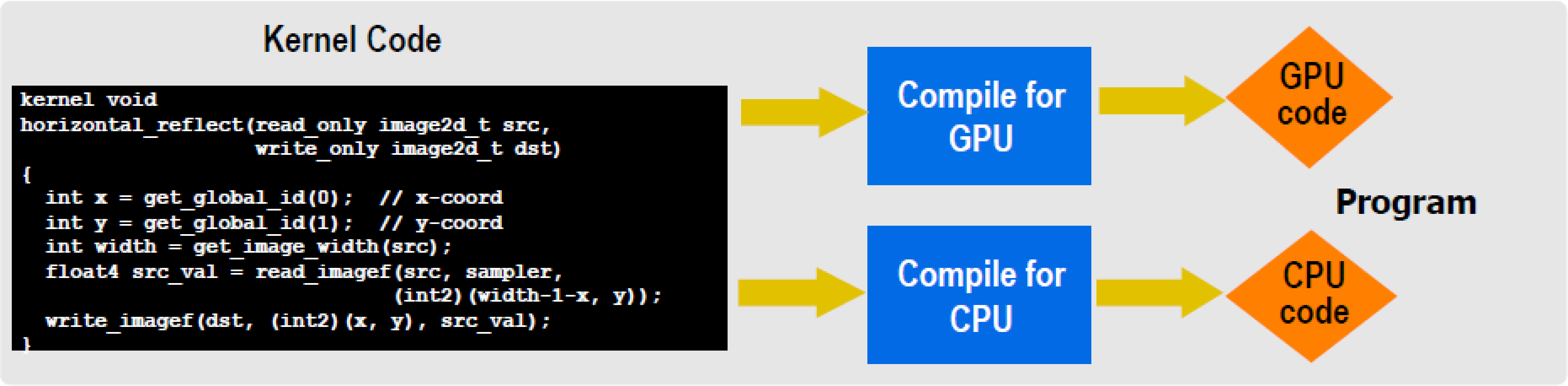
0 1 2 3

2 3 -7 -7

2 3 -7 -7 0 1 1 3

Building Program Objects

- **The program object encapsulates:**
  - A context
  - The program source/binary
  - List of target devices and build options
- **The Build process … to create a program object**
  - clCreateProgramWithSource()
  - clCreateProgramWithBinary()



Kernel Code

```
kernel void
horizontal_reflect(read_only image2d_t src,
                   write_only image2d_t dst)
{
  int x = get_global_id(0);  // x-coord
  int y = get_global_id(1);  // y-coord
  int width = get_image_width(src);
  float4 src_val = read_imagef(src, sampler,
                               (int2)(width-1-x, y));
  write_imagef(dst, (int2)(x, y), src_val);
}
```

Compile for GPU

GPU code

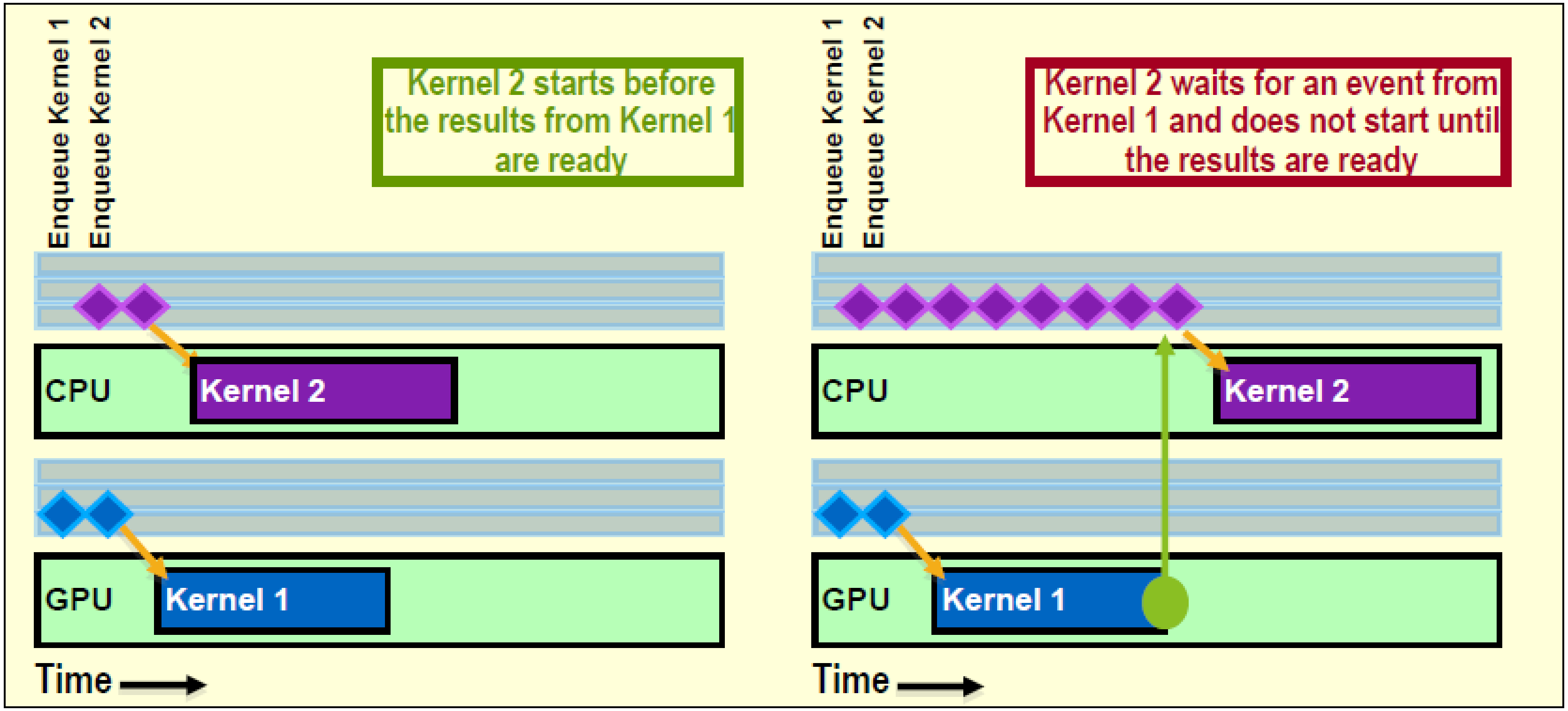
Program

Compile for CPU

CPU code

OpenCL Synch: Queues & Events

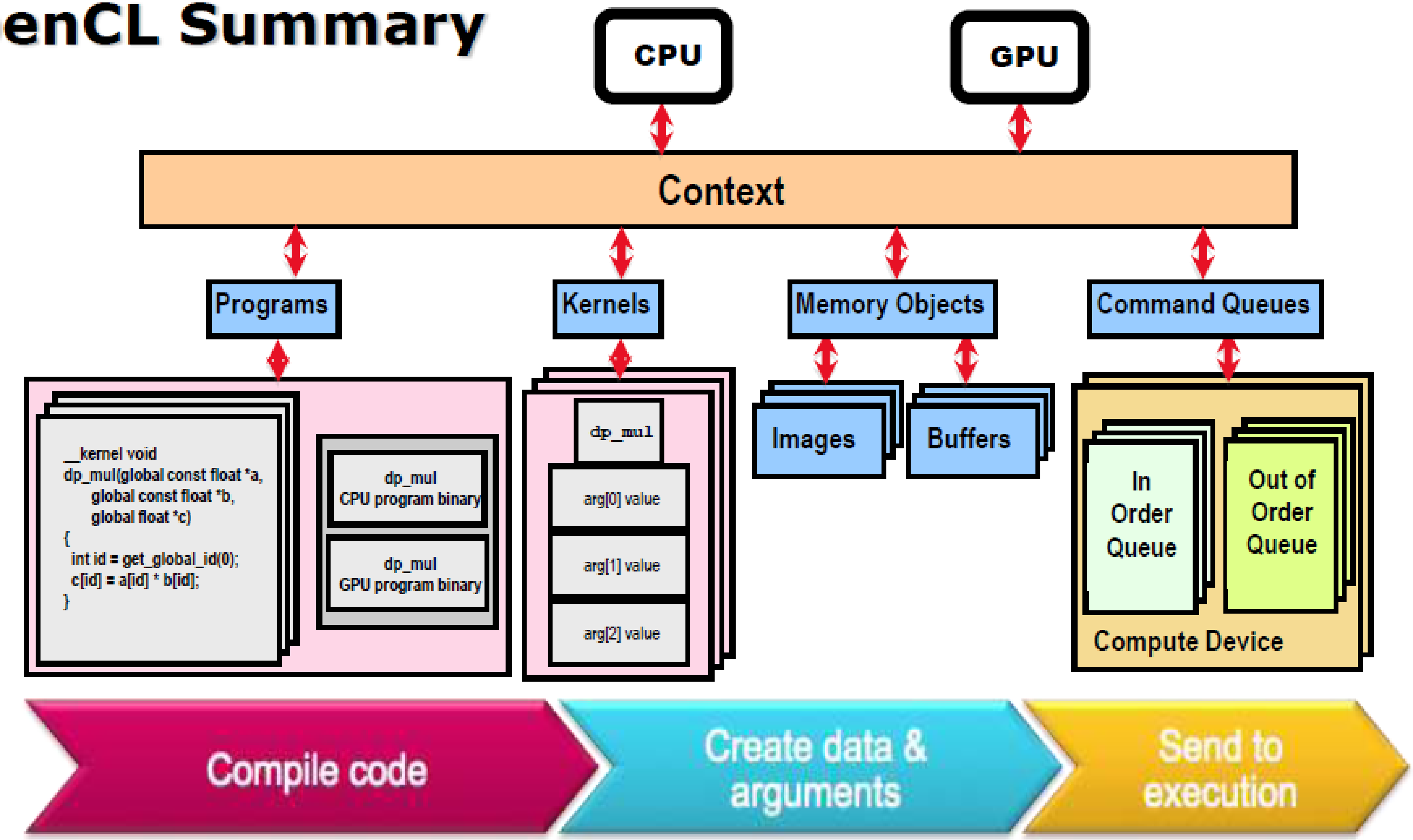
- Events can be used to synchronize kernel executions between queues
- Example: 2 queues with 2 devices



Enqueue Kernel 1
Enqueue Kernel 2
Kernel 2 starts before the results from Kernel 1 are ready
CPU
Kernel 2
GPU
Kernel 1
Time
Enqueue Kernel 1
Enqueue Kernel 2
Kernel 2 waits for an event from Kernel 1 and does not start until the results are ready
CPU
Kernel 2
GPU
Kernel 1
Time

OpenCL

OpenCL Summary



```
__kernel void
dp_mul(global const float *a,
      global const float *b,
      global float *c)
{
  int id = get_global_id(0);
  c[id] = a[id] * b[id];
}
```