OpenCL Overview
Benedict R. Gaster, AMD

March 2010

The BIG Idea behind OpenCL

- **OpenCL execution model …**
  - Define N-dimensional computation domain
  - Execute a kernel at each point in computation domain

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
```

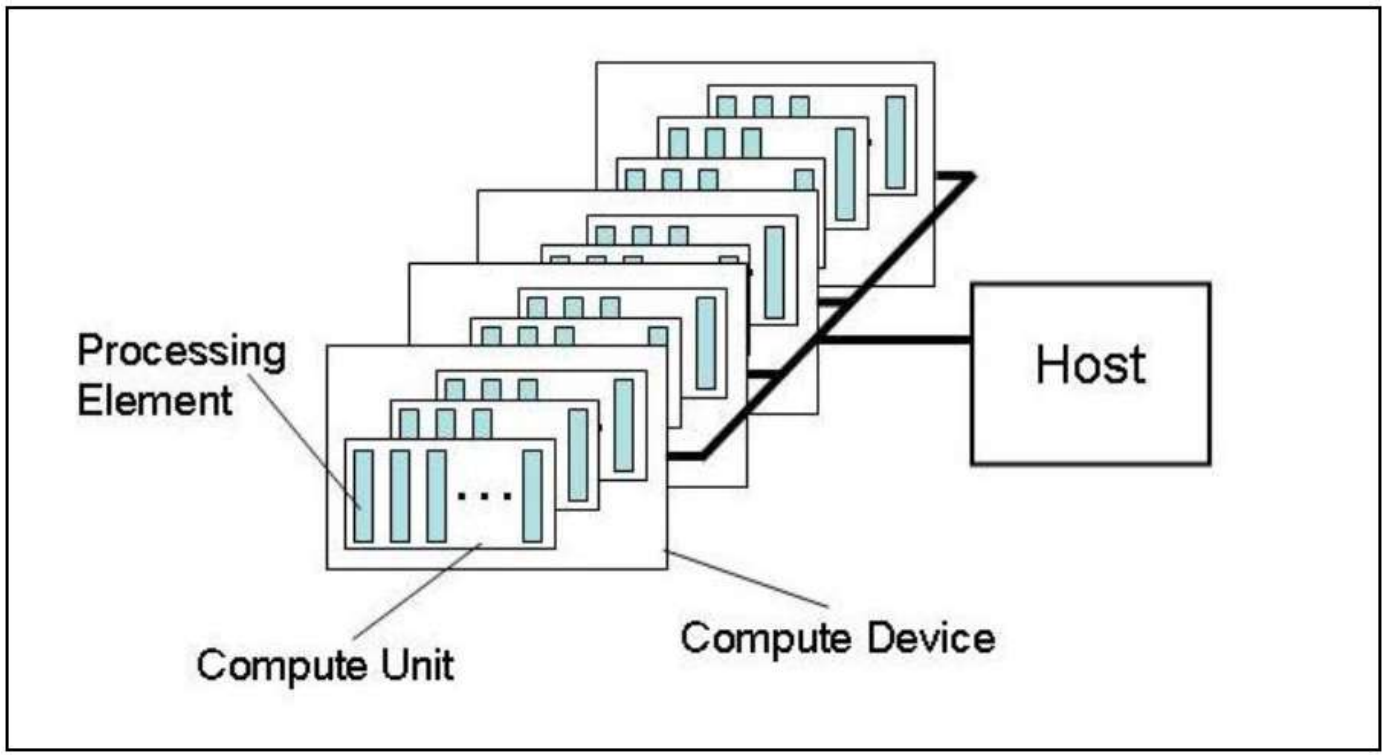
Anatomy of OpenCL

- **Language Specification**
  - C-based cross-platform programming interface
  - Subset of ISO C99 with language extensions - familiar to developers
  - Defined numerical accuracy - IEEE 754 rounding with specified maximum error
  - Online or offline compilation and build of compute kernel executables
  - Rich set of built-in functions
- **Platform Layer API**
  - A hardware abstraction layer over diverse computational resources
  - Query, select and initialize compute devices
  - Create compute contexts and work-queues
- **Runtime API**
  - Execute compute kernels
  - Manage scheduling, compute, and memory resources

KHRONOS GROUP

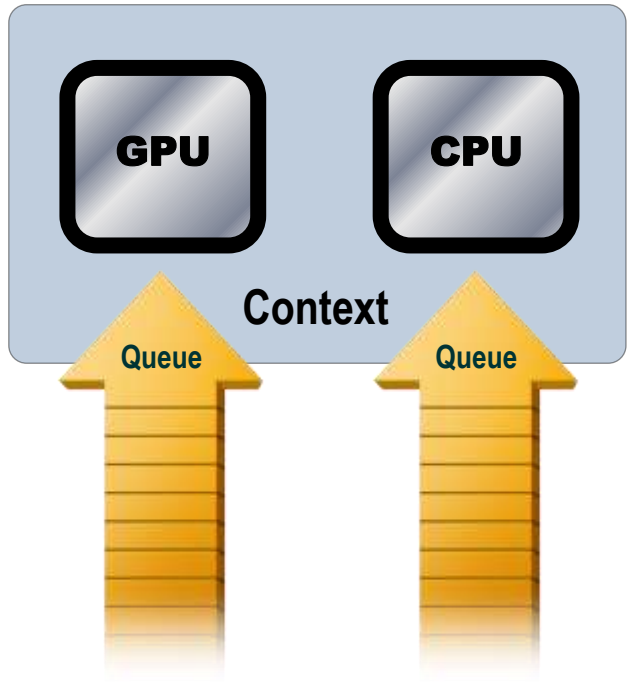
OpenCL Platform Model

- **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



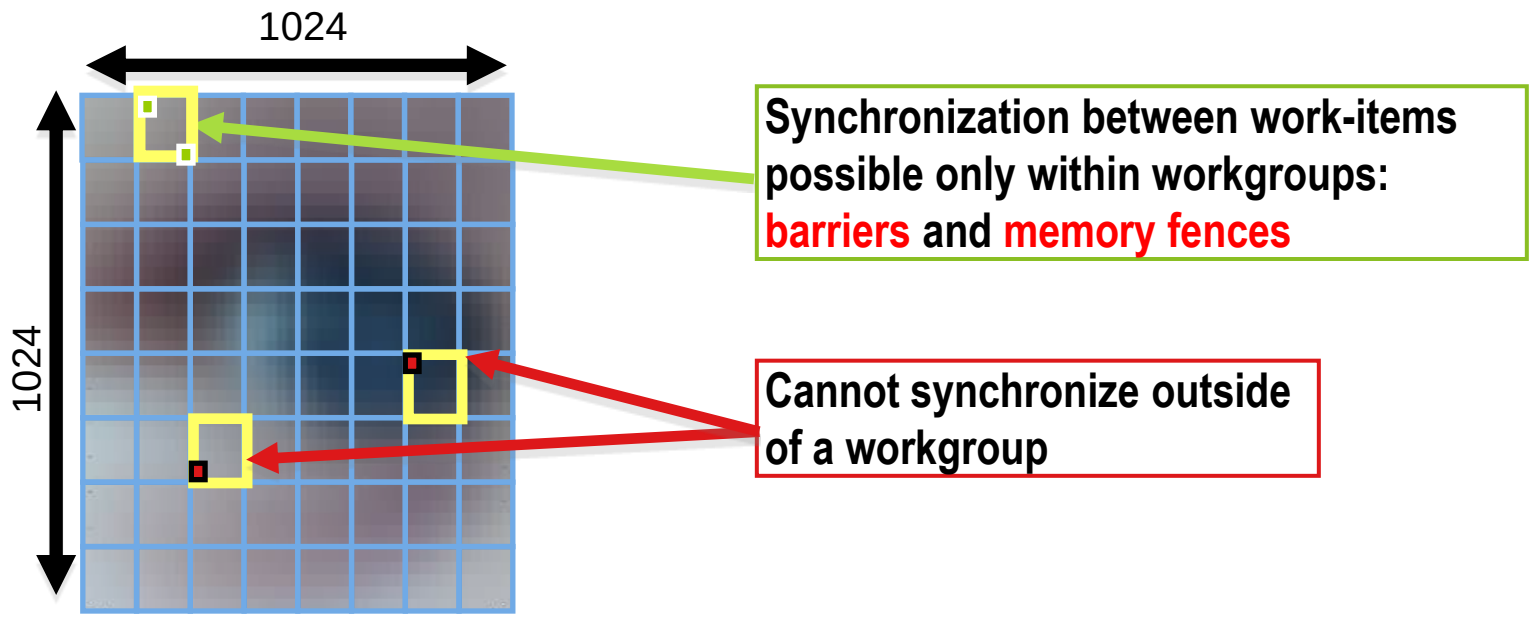
OpenCL Execution Model

- **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**
  - Executed in-order or out-of-order



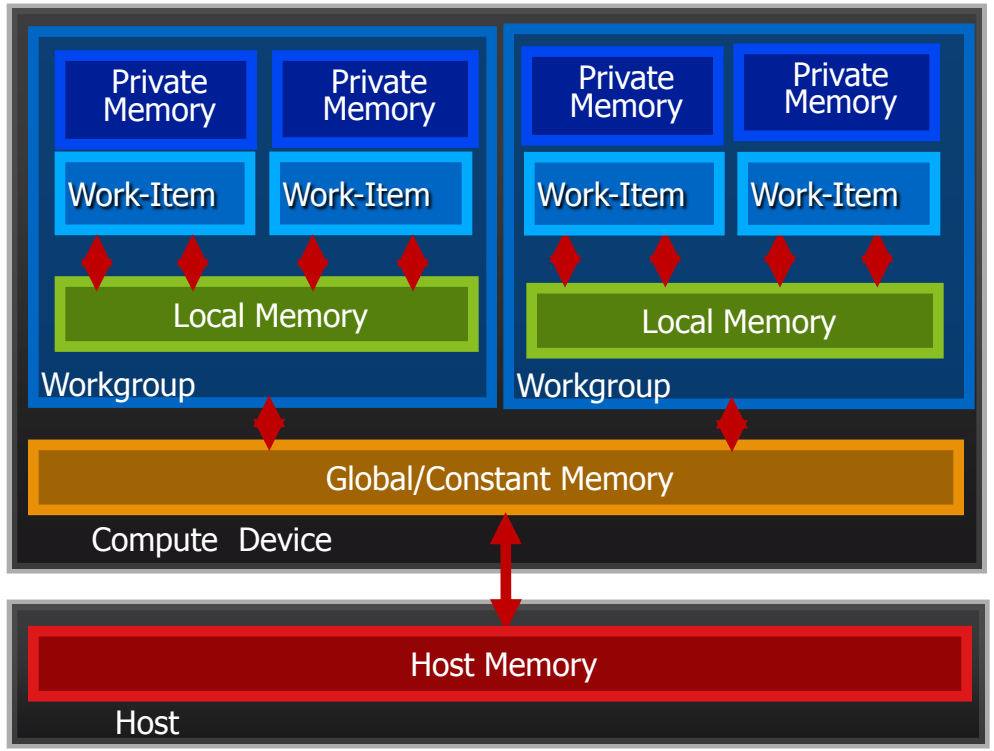
An N-dimension domain of work-items

- **Kernels executed across a global domain of *work-items***
- **Work-items grouped into local *workgroups***
- **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)



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

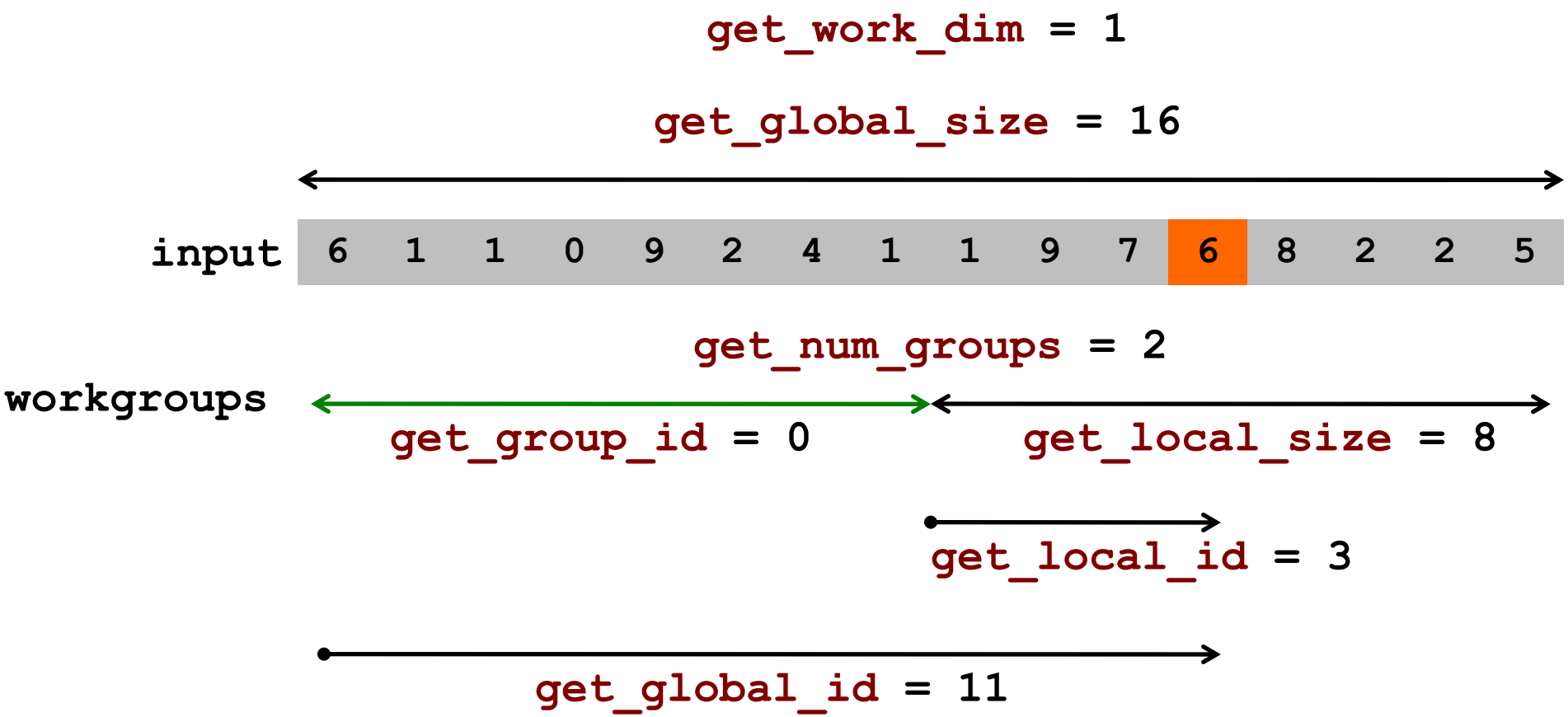
- **Derived from ISO C99**
  - But without some C99 features such as standard C99 headers, function pointers, recursion, variable length arrays, and bit fields
- **Language Features Added**
  - Work-items and workgroups
  - Vector types
  - Synchronization
  - Address space qualifiers
- **Also includes a large set of built-in functions**
  - Image manipulation
  - Work-item manipulation,
  - Math functions, etc.

Programming Kernels: What is a kernel

- **A data-parallel function executed by each work-item**

```
kernel void square(global float* input, global float*
output)
{
  int i = get_global_id(0);
  output[i] = input[i] * input[i];
}
```

Programming Kernels: WorkItems & Groups



Programming Kernels: Data Types

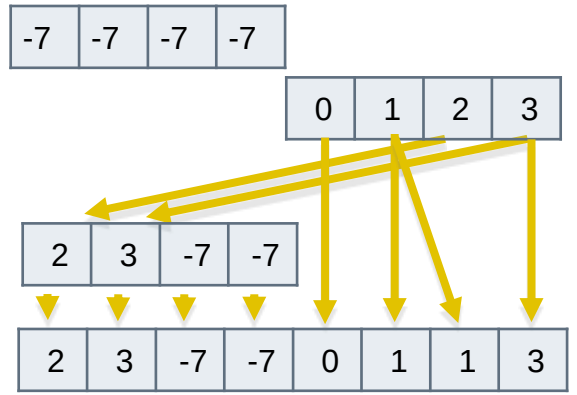
- **Scalar data types**
  - char , uchar, short, ushort, int, uint, long, ulong, float
  - bool, intptr_t, ptrdiff_t, size_t, uintptr_t, void, half (storage)
- **Image types**
  - image2d_t, image3d_t, sampler_t
- **Vector data types**
  - Vector lengths 2, 4, 8, & 16 (char2, ushort4, int8, float16,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

Double is an optional type in OpenCL 1.0

Programming Kernels: Vector Operations

- **Vector Literal**

```
int4 vi0 = (int4) -7;
int4 vi1 = (int4)(0, 1, 2, 3);
```

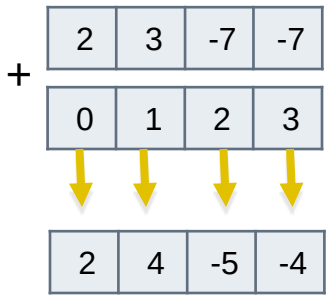


- **Vector Components**

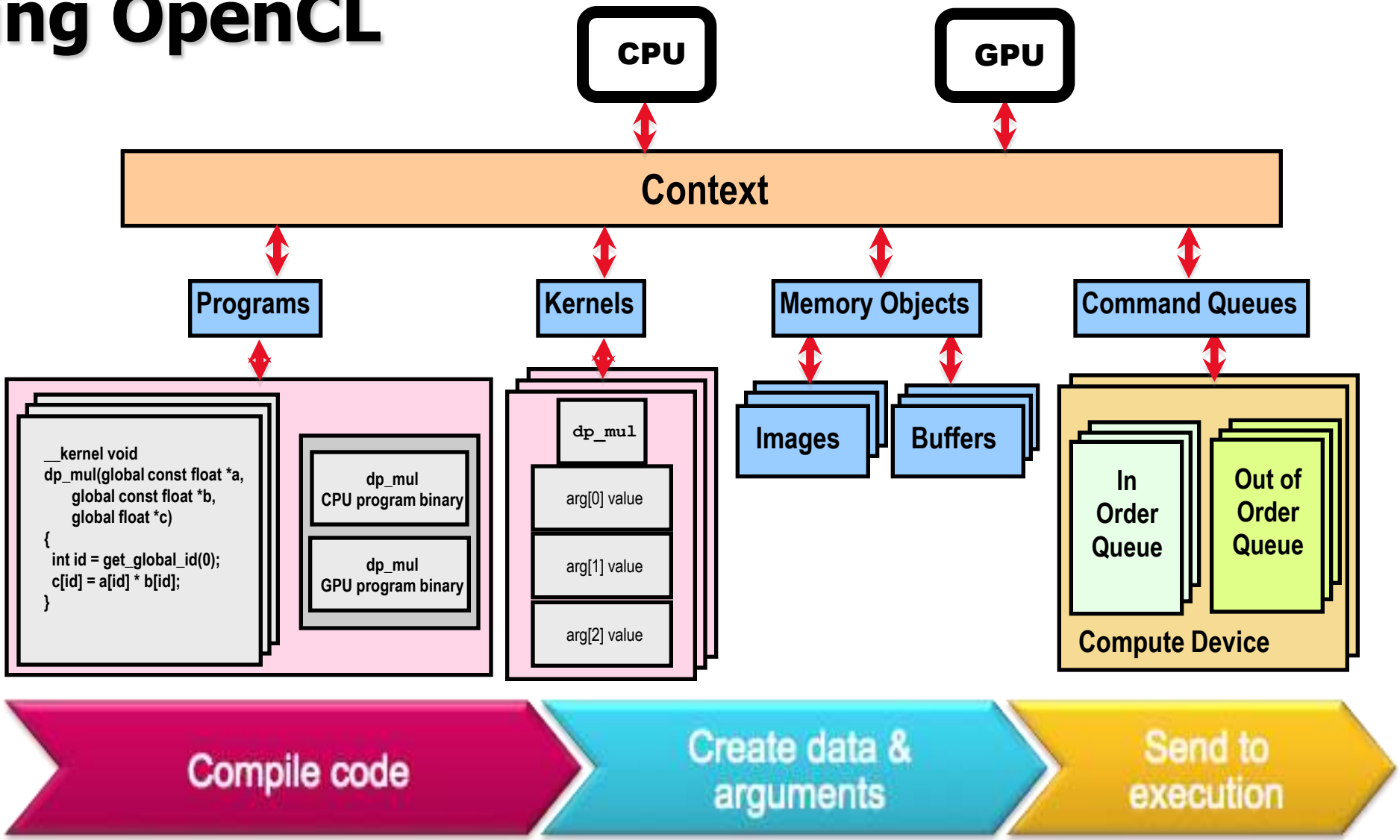
```
vi0.lo = vi1.hi;
int8 v8 = (int8)(vi0, vi1.s01, vi1.odd);
```


- **Vector Operations**

```
vi0 += vi1;
```

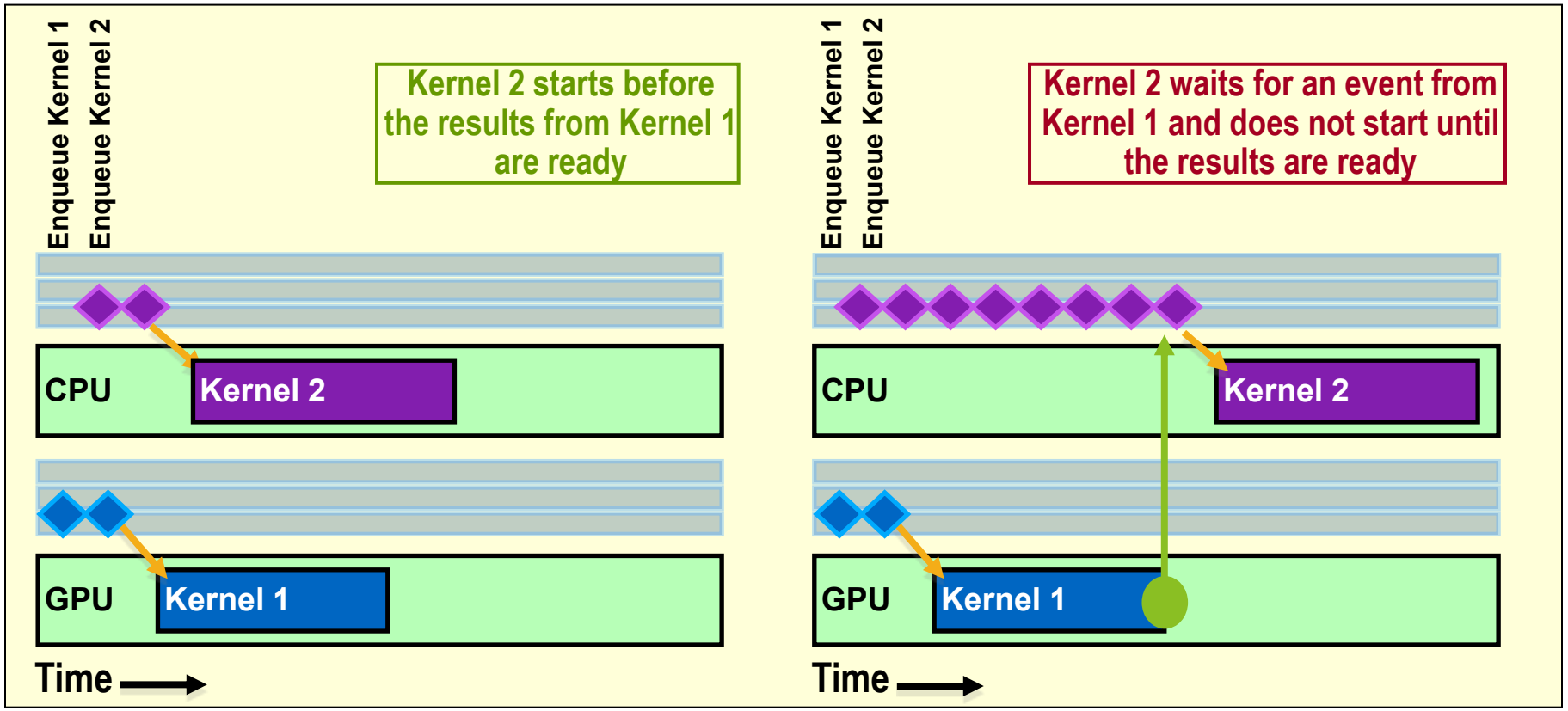


Using OpenCL



Synchronization: Queues & Events

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



OpenCL 1.1 – New API Features

- Thread-safety for all API calls except clSetKernelArg
- Sub-Buffer Objects - to distribute regions of buffer to multiple devices
- Reading, writing & copying rectangular regions in a buffer object
- User Events - all clEnqueue* commands take a list of events to wait on
- Event Callbacks – enqueueing of CL commands on event state changes
- clSetEventCallback to register a user callback function
- Memory Object Destructor Callback
- Device queries
- global_work_offset Enqueue kernels for regions of the ND range
- Link CL events and GL sync objects for faster & finer-grained interop

KHRONOS GROUP

OpenCL 1.1 – New Language Features

- C++ Bindings
- Implicit conversions: relational, equality, bitwise, logical, ternary operators
- 3-component vector data types
- Byte addressability for char, char2, uchar, uchar2, short, ushort and half
- 32-bit atomic operations to local and global memory
- Clamp for integer data types
- Strided copies for scatter/gather from and to global and local memory
- Shuffle to construct runtime permutations from 1 or 2 vectors and a mask
- Image Addressing mode – CL_ADDRESS_MIRRORED_REPEAT
- Optional image formats – CL_Rx, CL_RGx and CL_RGBx

OpenCL 1.1 – Thread-safety & Buffers

- **Thread-safety**
  - All API calls except **clSetKernelArg** are now thread-safe
- **Sub-Buffer Objects**
  - Easy and efficient way to distribute regions of a buffer across multiple devices
  - Modifications to sub-buffer objects reflected in appropriate regions of parent buffer object.
- **Reading, writing & copying rectangular regions in a buffer object**
  - Specify the following
    - Region type – 2D or 3D
    - Row-pitch for a 2D & 3D region and Slice-pitch for a 3D region
  - **clEnqueue{Read|Write|Copy}BufferRect**

OpenCL 1.1 – Events

- **User Events**
  - All **clEnqueue*** commands take a list of events to wait on
  - In OpenCL 1.0, events can only refer to OpenCL commands
  - User events allow developers to enqueue commands that wait on an external event
- **Event Callbacks**
  - Allow applications to enqueue CL commands based on event state changes in a non-blocking manner
  - **clSetEventCallback** to register a user callback function
    - Called when command identified by event has completed
    - Recommend **not calling** expensive system APIs, OpenCL APIs that create objects or enqueue blocking commands in the callback function.

KHRONOS GROUP

OpenCL 1.1 – Memory Object Callbacks

- **Memory Object Destructor Callback**
  - For **cl_mem** objects created with **CL_MEM_USE_HOST_PTR** need a way to determine when it is safe to free or reuse the **host_ptr**
  - Lazy deallocation of **cl_mem** objects make this a little difficult
  - **clSetMemObjectDestructorCallback**
    - Registers a destructor callback function
    - Called when the memory object is ready to be deleted
  - Recommend **not calling** expensive system APIs, OpenCL APIs that create objects or enqueue blocking commands in the callback function.

OpenCL 1.1 – Queries

- **Kernel Queries**
  - **CL_KERNEL_PREFERRED_WORKGROUP_SIZE_MULTIPLE**
    - A performance hint
- **Device Queries**
  - **CL_DEVICE_LOCAL_MEM_SIZE**
    - Increased from 16 KB to 32 KB
  - **CL_DEVICE_MAX_PARAMETER_SIZE**
    - Increased from 256 to 1024 bytes
  - **CL_DEVICE_OPENCL_C_VERSION**
    - Version of OpenCL C supported by device.
  - **CL_DEVICE_HOST_UNIFIED_MEMORY**
    - Whether device & host have a unified memory subsystem

KHRONOS GROUP

OpenCL 1.1 – Additional API Features

- **global_work_offset**
  - Argument to **clEnqueueNDRangeKernel**
  - No longer required to be a NULL value
  - Enqueue kernels that operate on different regions of the N-D range
- **C++ API bindings**
  - A wrapper API
  - Built on top of the OpenCL 1.1 API specification (not a replacement)

OpenCL 1.1 – Language Features

- **Implicit Conversions**
  - OpenCL 1.0 requires widening for arithmetic operators

```
float4  a, b;
float c;

b = a + c; // c is widened to a float4 first
           // and then the + is performed.
```

  - OpenCL 1.1 extends this feature to all operators
    - relational, equality, bitwise, logical and ternary

OpenCL 1.1 – Language Features

- **3-component vector data types**
  - Useful data type for a number of applications such as game physics
  - Aligned to the corresponding 4-component data type
  - vload3 and vstore3 can be used to view a buffer of scalar elements as a packed buffer of 3-component vectors
- **cl_khr_byte_addressable is a core feature**
  - Writes to a pointer or array of `type` **`char, char2, uchar, uchar2, short, ushort`** and **`half`** are now supported
- **32-bit atomic operations to local and global memory is a core feature**

KHRONOS GROUP

OpenCL 1.1 – Built-in Functions

- **get_global_offset**
  - Global offset values specified to **clEnqueueNDRangeKernel**
- **clamp for integer data types**
  - Only floating-point types were supported in OpenCL 1.0
- **async_work_group_strided_copy**
  - gather from global to local memory
  - scatter from local to global memory
- **shuffle**
  - Construct a runtime permutation of elements from 1 or 2 vectors and a mask

```
uint4 mask = (uint4)(3, 2, 1, 0);
float4 a;
float4 r = shuffle(a, mask)
// r.s0123 = a.wzyx
```

OpenCL 1.1 – Images

- **Addressing mode – CL_ADDRESS_MIRRORED_REPEAT**
  - Flip the image coordinate at every integer junction
  - Can only be used with normalized coordinates i.e. **CL_NORMALIZED_COORDS_TRUE** must be set in the sampler
- **Optional image formats – CL_Rx, CL_RGx and CL_RGBx**
  - Similar to CL_R, CL_RG and CL_RGB except *alpha = 0* at *edges*
  - For image processing, *alpha* must always be 0 at edges.