OpenCL *1.2*

OpenCL Overview

Ofer Rosenberg, AMD
November 2011

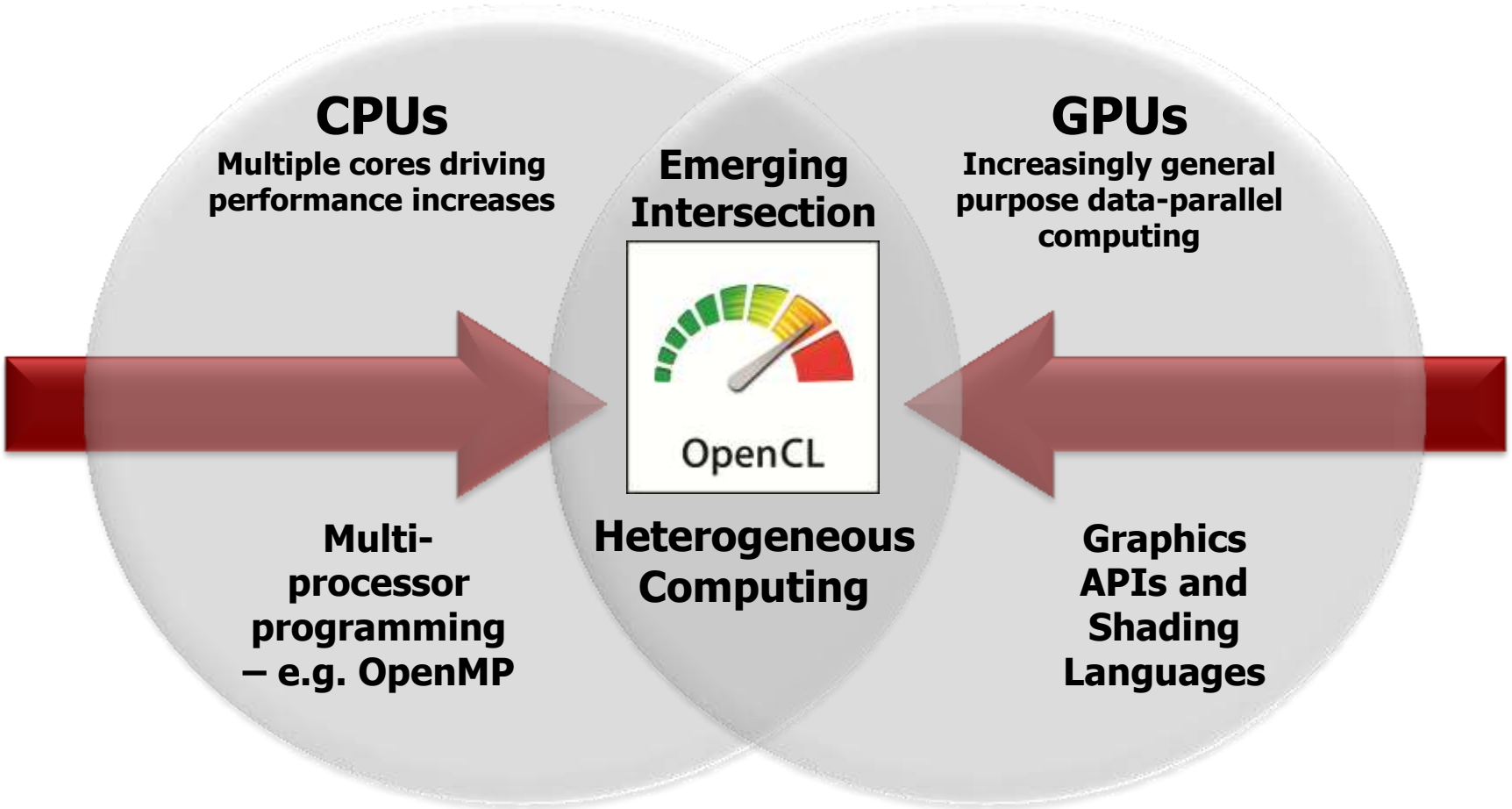
KHRONOS GROUP

Agenda

- OpenCL in context
- OpenCL Overview
- Next steps for OpenCL

KHRONOS GROUP

Processor Parallelism



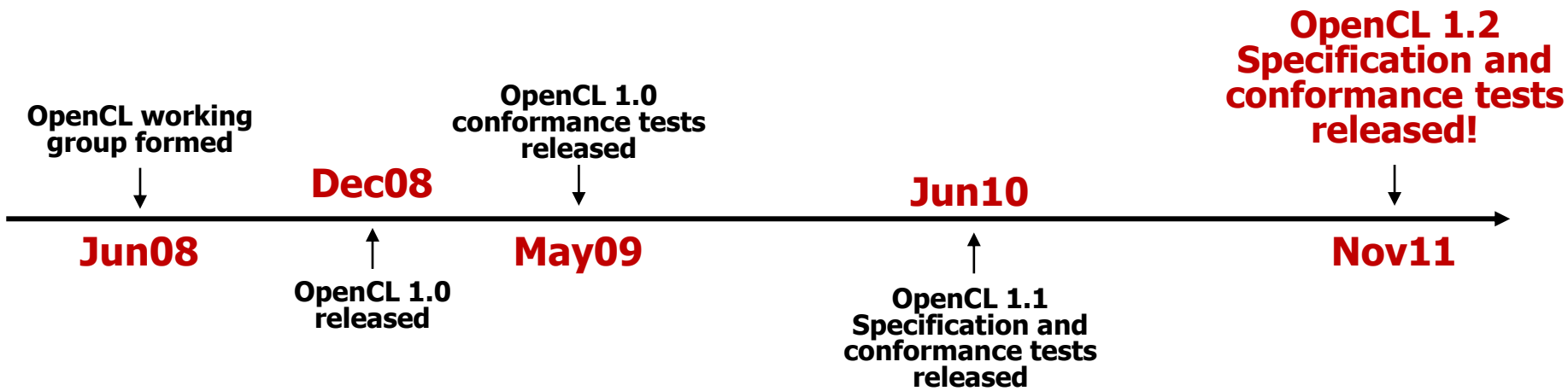
OpenCL is a programming framework for heterogeneous compute resources

OpenCL Working Group Members

- **Diverse industry participation – many industry experts**
  - Processor vendors, system OEMs, middleware vendors, application developers
  - Academia and research labs, FPGA vendors
- **NVIDIA is chair, Apple is specification editor**

OpenCL Milestones

- **Six months from proposal to released OpenCL 1.0 specification**
  - Due to a strong initial proposal and a shared commercial incentive
- **Multiple conformant implementations shipping**
  - For CPUs and GPUs on multiple OS
- **18 month cadence between OpenCL 1.0, OpenCL 1.1 and now OpenCL 1.2**
  - Backwards compatibility protect software investment



Khronos OpenCL Resources

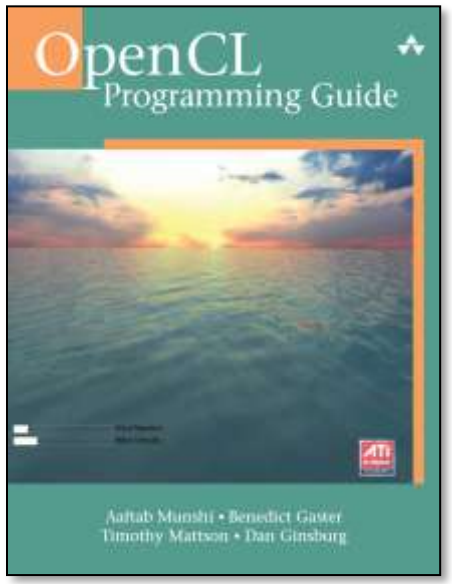
- **OpenCL is 100% free for developers**
  - Download drivers from your silicon vendor
- **OpenCL Registry**
  - www.khronos.org/registry/cl/
- **OpenCL 1.2 Reference Card**
  - PDF version
  - http://www.khronos.org/files/opencl-1-2-quick-reference-card.pdf
- **Online Reference pages**
  - http://www.khronos.org/registry/cl/sdk/1.2/docs/man/xhtml/
- **OpenCL Developer Forums**
  - Give us your feedback!
  - www.khronos.org/message_boards/

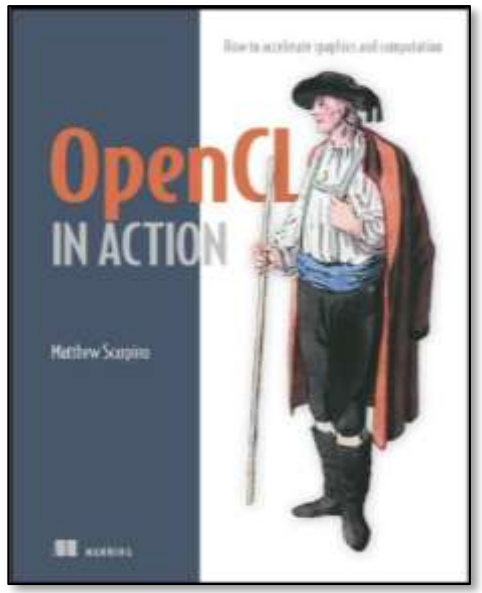
OpenCL Desktop Implementations

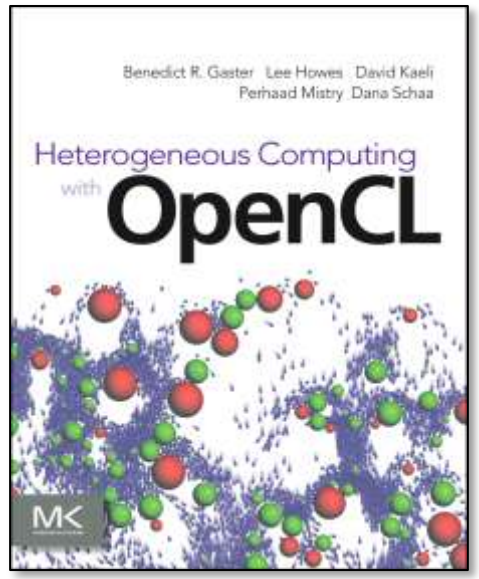
- http://developer.amd.com/zones/OpenCLZone/
- http://software.intel.com/en-us/articles/opencl-sdk/
- http://developer.nvidia.com/opencl

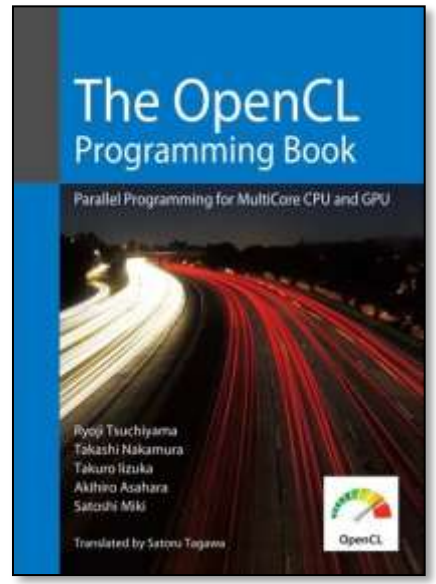
OpenCL Books – Available Now!

- **OpenCL Programming Guide - The "Red Book" of OpenCL**
  - http://www.amazon.com/OpenCL-Programming-Guide-Aaftab-Munshi/dp/0321749642
- **OpenCL in Action**
  - http://www.amazon.com/OpenCL-Action-Accelerate-Graphics-Computations/dp/1617290173/
- **Heterogeneous Computing with OpenCL**
  - http://www.amazon.com/Heterogeneous-Computing-with-OpenCL-ebook/dp/B005JRHYUS
- **The OpenCL Programming Book**
  - http://www.fixstars.com/en/opencl/book/









Agenda

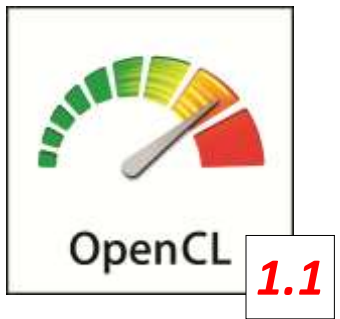
- OpenCL in context
- OpenCL Overview
- Next steps for OpenCL

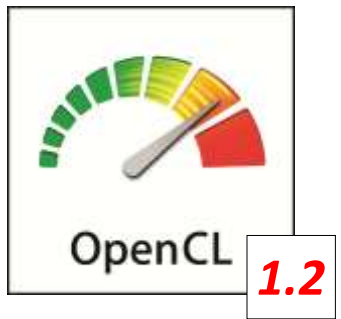
Welcome to OpenCL

- With OpenCL you can…

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

- This presentation covers OpenCL 1.0 – 1.2. Features which are supported in OpenCL 1.1 or 1.2 are marked in the following way:





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
```

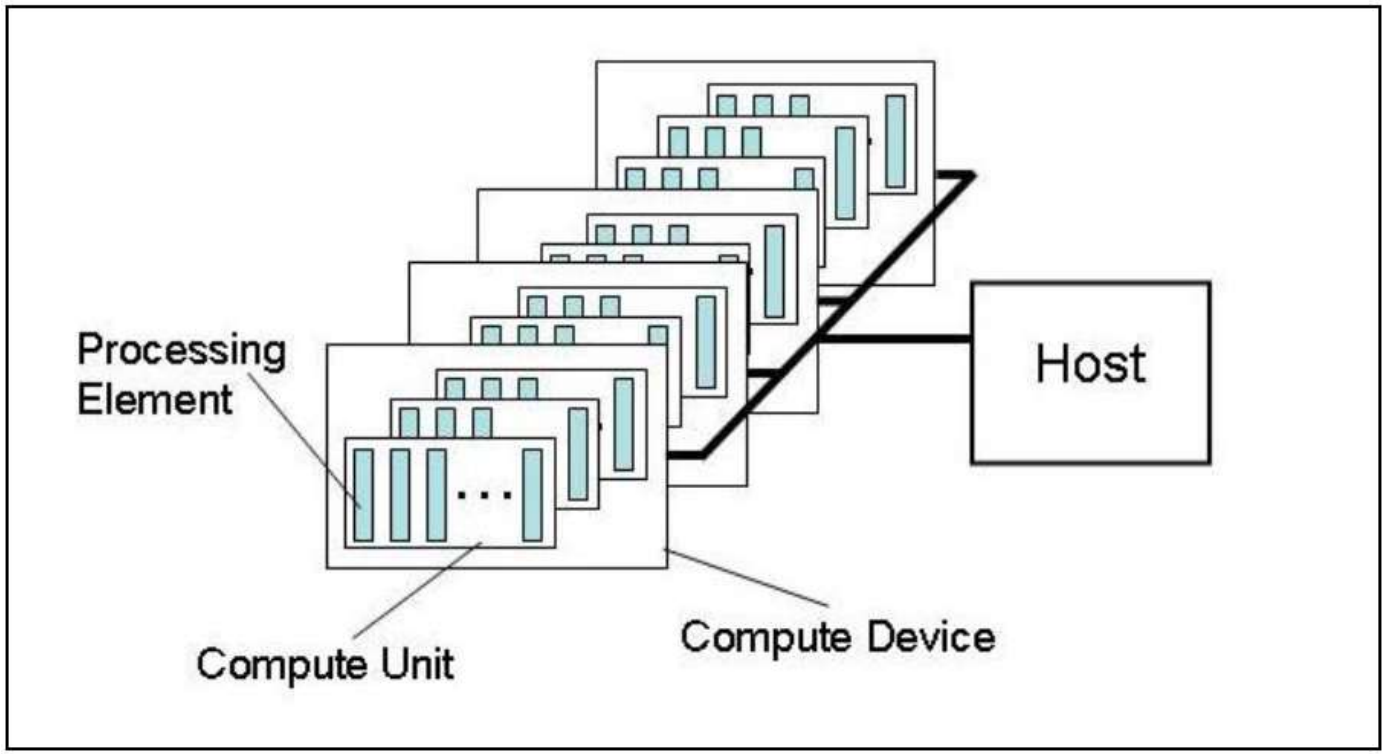
Anatomy of OpenCL

- **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
- **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

KHRONOS GROUP

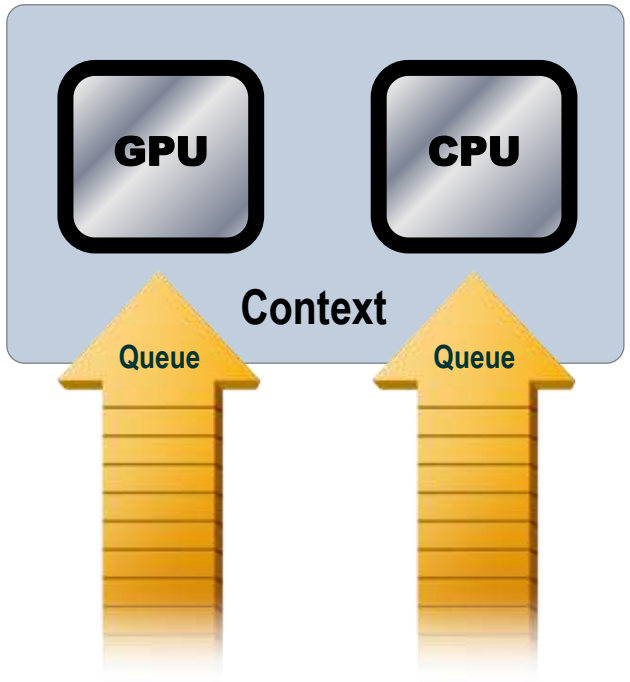
OpenCL Platform Model

- **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



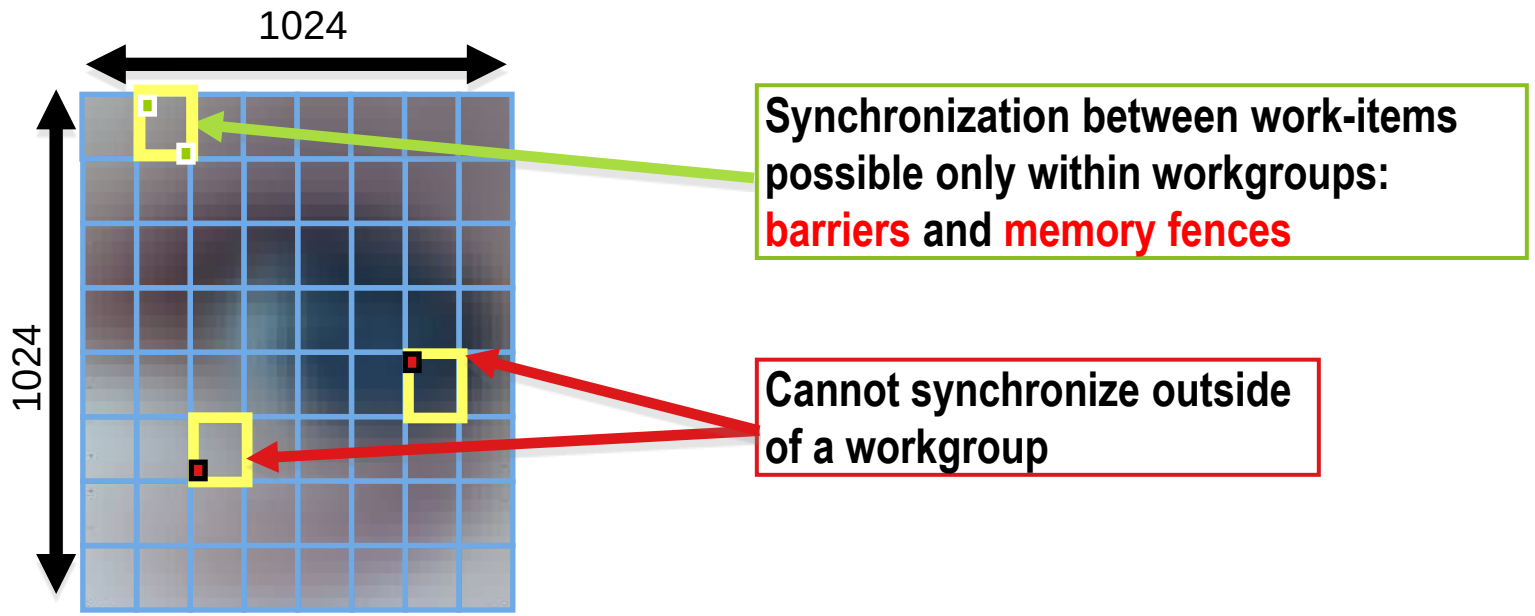
OpenCL Execution Model

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



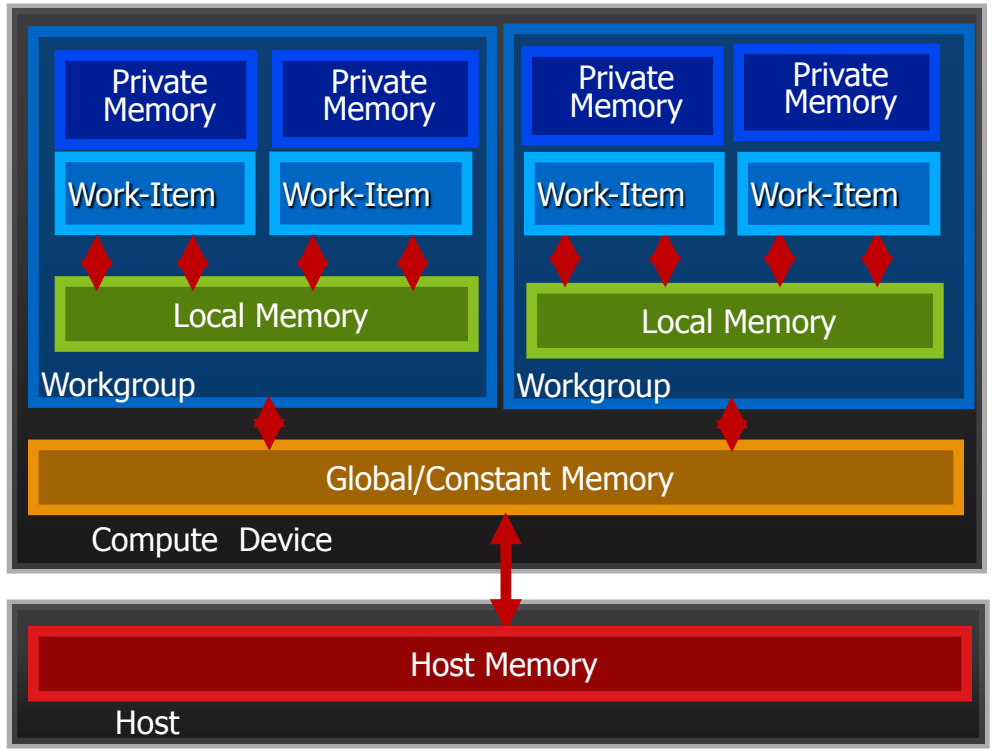
An N-dimension domain of work-items

- **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

Compilation Model

- **OpenCL™ uses Dynamic/Runtime compilation model (like OpenGL®):**
  1. The code is complied to an Intermediate Representation (IR)
     - Usually an assembler or a virtual machine
     - Known as offline compilation
  2. The IR is compiled to a machine code for execution.
     - This step is much shorter.
     - It is known as online compilation.

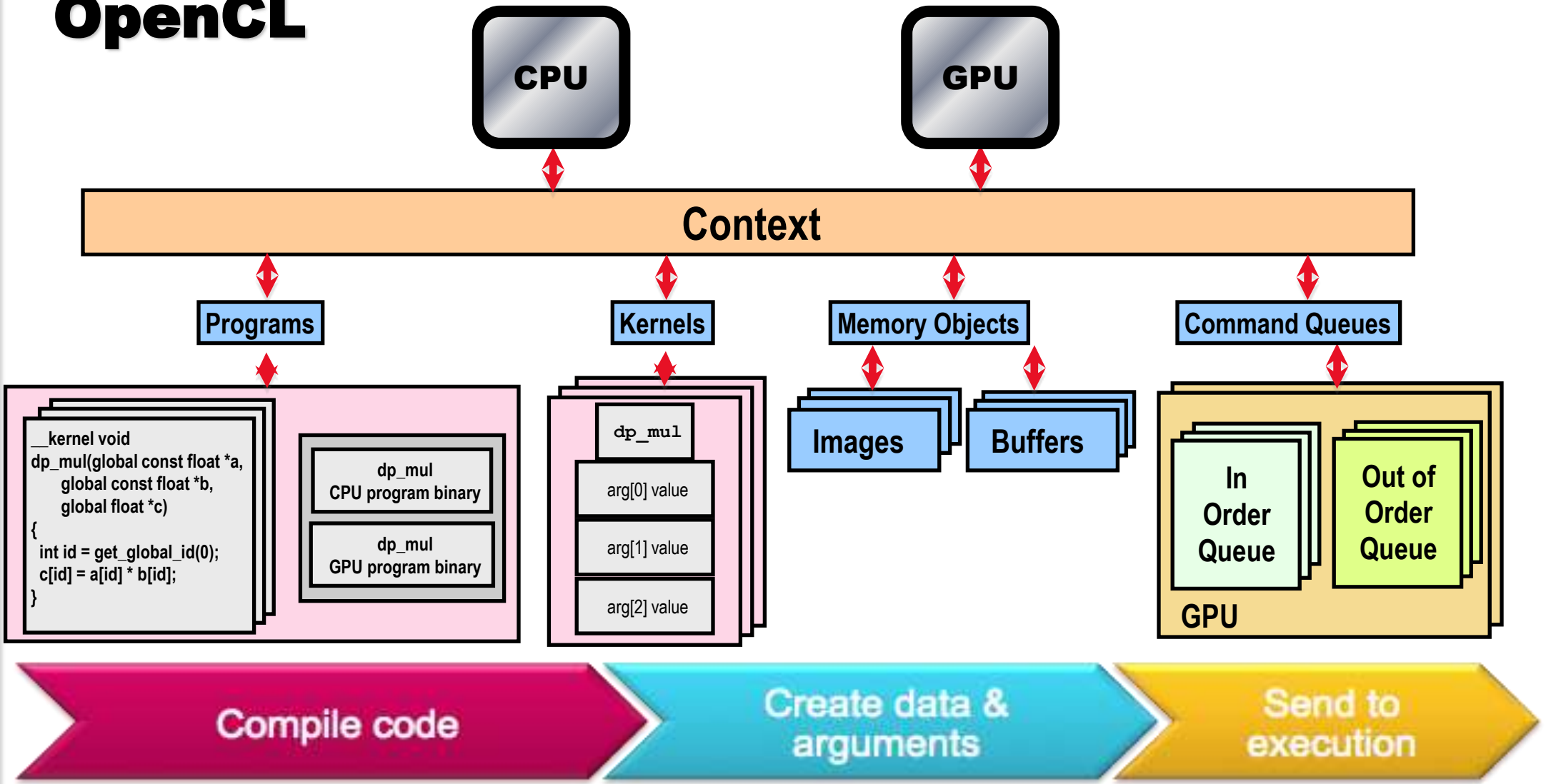
- **In dynamic compilation, step 1 is done usually only once, and the IR is stored.**
- **The App loads the IR and performs step 2 during the App's runtime (hence the term…)**

Using OpenCL

OpenCL Objects

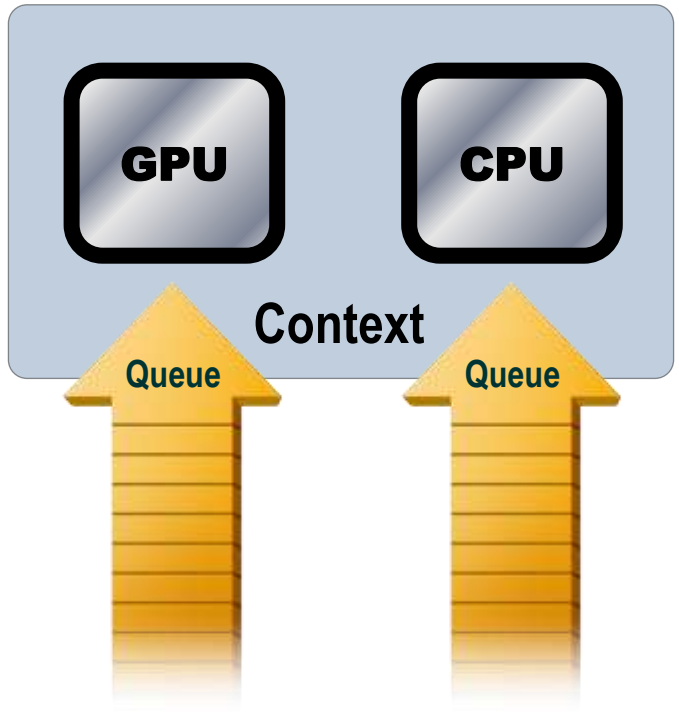
- **Setup**
  - Devices — GPU, CPU, Cell/B.E.
  - Contexts — Collection of devices
  - Queues — Submit work to the device
- **Memory**
  - Buffers — Blocks of memory
  - Images — 2D or 3D formatted images
- **Execution**
  - Programs — Collections of kernels
  - Kernels — Argument/execution instances
- **Synchronization/profiling**
  - Events

OpenCL



Setup

1. Get the device(s)
2. Create a context
3. Create command queue(s)




```
cl_uint num_devices_returned;
cl_device_id devices[2];
err = clGetDeviceIDs(NULL, CL_DEVICE_TYPE_GPU, 1,
                     &devices[0], num_devices_returned);
err = clGetDeviceIDs(NULL, CL_DEVICE_TYPE_CPU, 1,
                     &devices[1], &num_devices_returned);

cl_context context;
context = clCreateContext(0, 2, devices, NULL, NULL, &err);

cl_command_queue queue_gpu, queue_cpu;
queue_gpu = clCreateCommandQueue(context, devices[0], 0, &err);
queue_cpu = clCreateCommandQueue(context, devices[1], 0, &err);
```

Setup: Notes

- **Devices**
  - Multiple cores on CPU or GPU together are a single device
  - OpenCL executes kernels across all cores in a data-parallel manner
- **Contexts**
  - Enable sharing of memory between devices
  - To share between devices, both devices must be in the same context
- **Queues**
  - All work submitted through queues
  - Each device must have a queue

Choosing Devices

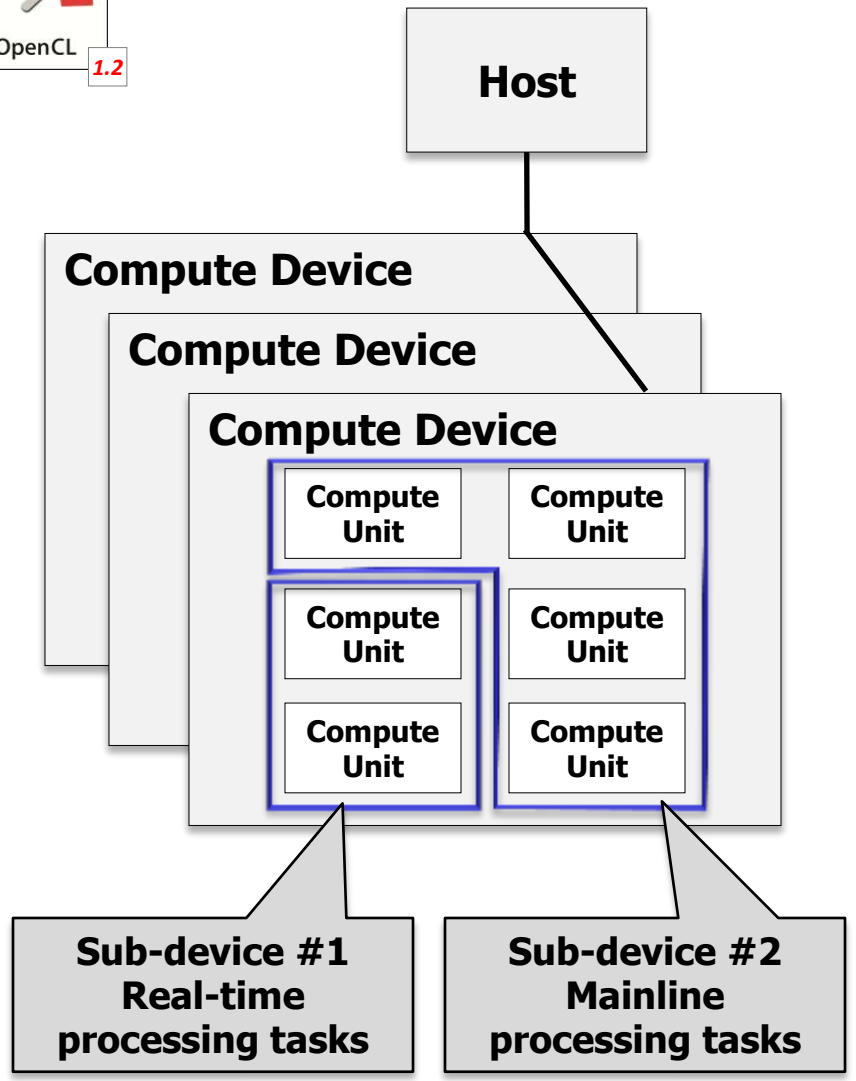
- A system may have several devices—which is best?
- The "best" device is algorithm- and hardware-dependent

- Query device info with: `clGetDeviceInfo(device, param_name, *value)`
  - Number of compute units `CL_DEVICE_MAX_COMPUTE_UNITS`
  - Clock frequency `CL_DEVICE_MAX_CLOCK_FREQUENCY`
  - Memory size `CL_DEVICE_GLOBAL_MEM_SIZE`
  - Extensions (double precision, atomics, etc.)

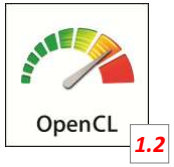
- Pick the best device for your algorithm

Partitioning Devices

- **Devices can be partitioned into sub-devices**
  - More control over how computation is assigned to compute units
- **Sub-devices may be used just like a normal device**
  - Create contexts, building programs, further partitioning and creating command-queues
- **Three ways to partition a device**
  - Split into equal-size groups
  - Provide list of group sizes
  - Group devices sharing a part of a cache hierarchy



Custom Devices and Built-in Kernels

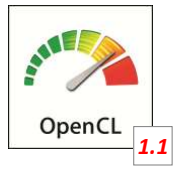


- **Embedded platforms often contain specialized hardware and firmware**
  - That cannot support OpenCL C
- **Built-in kernels can represent these hardware and firmware capabilities**
  - Such as video encode/decode
- **Hardware can be integrated and controlled from the OpenCL framework**
  - Can enqueue built-in kernels to custom devices alongside OpenCL kernels
- **OpenCL becomes a powerful coordinating framework for diverse resources**
  - Programmable and non-programmable devices controlled by one run-time

KHRONOS GROUP

Memory Resources

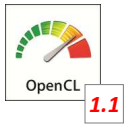
- **Buffers**
  - Simple chunks of memory
  - Kernels can access however they like (array, pointers, structs)
  - Kernels can read and write buffers
- **Sub-Buffers**
  - Added in OpenCL 1.1
  - Created from regions of OpenCL Buffers
  - Enables distribution of buffers & compute to multiple devices



- **Images**
  - Opaque 2D or 3D formatted data structures
  - OpenCL 1.2 added 1D, 1D from Buffer, 2D array & 3D array
  - Kernels access only via read_image() and write_image()
  - Each image can be read or written in a kernel, but not both

Image Formats and Samplers

- **Formats**
  - Channel orders: `CL_A`, `CL_RG`, `CL_RGB`, `CL_RGBA`, etc.
    - OpenCL 1.1: `CL_Rx`, `CL_RGx`, `CL_RGBx`
  - Channel data type: `CL_UNORM_INT8`, `CL_FLOAT`, etc.
  - `clGetSupportedImageFormats()` returns supported formats
- **Samplers (for reading images)**
  - Filter mode: linear or nearest
  - Addressing: clamp, clamp-to-edge, repeat or none
    - OpenCL 1.1: CL_ADDRESS_MIRRORED_REPEAT
  - Normalized: true or false
- **Benefit from image access hardware on GPUs**



Allocating Images and Buffers

```
cl_image_format format;
format.image_channel_data_type = CL_FLOAT;
format.image_channel_order = CL_RGBA;

cl_mem input_image;
input_image  = clCreateImage2D(context, CL_MEM_READ_ONLY, &format,
                    image_width, image_height, 0, NULL, &err);
cl_mem output_image;
output_image = clCreateImage2D(context, CL_MEM_WRITE_ONLY, &format,
                    image_width, image_height, 0, NULL, &err);

cl_mem input_buffer, output_buffer, input_subbuffer ;
input_buffer = clCreateBuffer(context, CL_MEM_READ_ONLY,
                    sizeof(cl_float)*4*image_width*image_height, NULL, &err);

output_buffer = clCreateBuffer(context, CL_MEM_WRITE_ONLY,
                    sizeof(cl_float)*4*image_width*image_height, NULL, &err);

cl_buffer_region sub1_region;
sub1_region.origin = 0;
Sub1_region.size = 4096;
input_subbuffer = clCreateSubBuffer(input_buffer, CL_MEM_READ_ONLY,
                      CL_BUFFER_CREATE_TYPE_REGION, &sub1_region, &err);
```

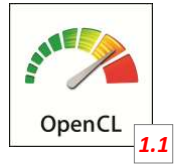
Reading / Writing Memory Object Data

- **Explicit commands to access memory object data**
  - OpenCL 1.1 added the "BufferRect" commands: 2D & 3D region access

- **Read from a region in memory object to host memory**
  - `clEnqueueReadBuffer(queue, object, blocking, offset, size, *ptr, ...)`
  - `clEnqueueReadBufferRect(queue, object, blocking, buffer_origin, ...)`
- **Write to a region in memory object from host memory**
  - `clEnqueueWriteBuffer(queue, object, blocking, offset, size, *ptr, ...)`
  - `clEnqueueWriteBufferRect(queue, object, blocking, buffer_origin, ...)`
- **Map a region in memory object to host address space**
  - `clEnqueueMapBuffer(queue, object, blocking, flags, offset, size, ...)`
  - `clEnqueueMapBufferRect(queue, object, blocking, buffer_origin, ...)`
- **Copy regions of memory objects**
  - `clEnqueueCopyBuffer(queue, srcobj, dstobj, src_offset, dst_offset, ...)`
  - `clEnqueueCopyBufferRect(queue, object, blocking, buffer_origin, ...)`
- **Operate synchronously (`blocking = CL_TRUE`) or asynchronously**

KHRONOS GROUP

Memory Object Callbacks



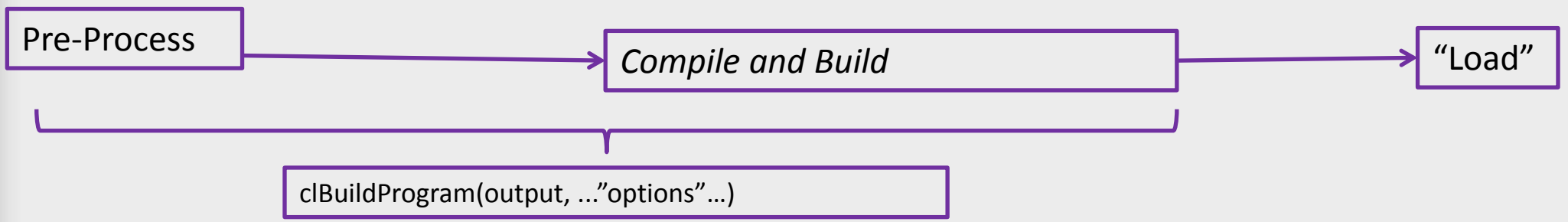
- **Memory Object Destructor Callback**
  - For **cl_mem** objects created with **CL_MEM_USE_HOST_PTR** need a way to determine when it is safe to free or reuse the **host_ptr**
  - Lazy deallocation of **cl_mem** objects make this a little difficult
  - **clSetMemObjectDestructorCallback**
    - Registers 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.

Compilation and Execution of Kernels

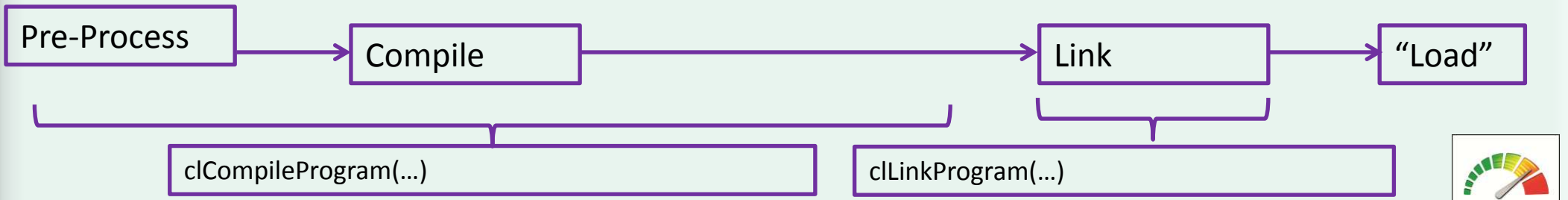
Program and Kernel Objects

- **Program objects encapsulate …**
  - a program source or binary
  - list of devices and latest successfully built executable for each device
  - a list of kernel objects
- **Kernel objects encapsulate …**
  - a specific kernel function in a program - declared with the kernel qualifier
  - argument values
  - kernel objects created after the program executable has been built

Building OpenCL Programs



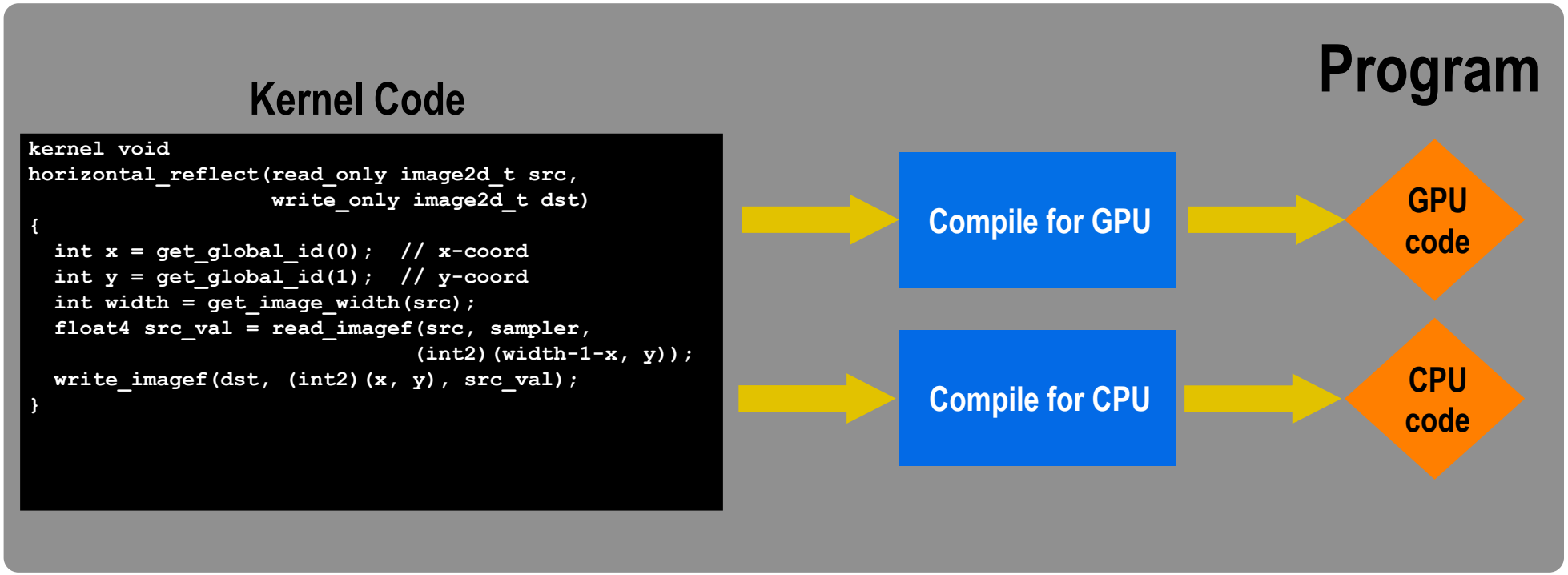
OpenCL 1.0 and 1.1: Compile and Build from a single source



OpenCL 1.2: Compile and linke separated. Supports modular software development

Executing Code

- Programs build executable code for multiple devices
- Execute the same code on different devices



Executing Kernels

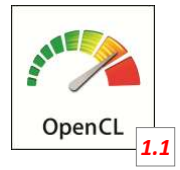
1. Set the kernel arguments
2. Enqueue the kernel

```
err = clSetKernelArg(kernel, 0, sizeof(input), &input);
err = clSetKernelArg(kernel, 1, sizeof(output), &output);

size_t global[3] = {image_width, image_height, 0};
err = clEnqueueNDRangeKernel(queue, kernel, 2, NULL, global, NULL, 0, NULL, NULL;
```

- **Note: Your kernel is executed asynchronously**
  - Nothing may happen — you have just enqueued your kernel
  - Use a blocking read `clEnqueueRead*(... CL_TRUE ...)`
  - Use events to track the execution status

- **OpenCL 1.1 added the ability to specify initial offset**
  - Range starts from a specific number
  - Split work across multiple devices, each executing a range



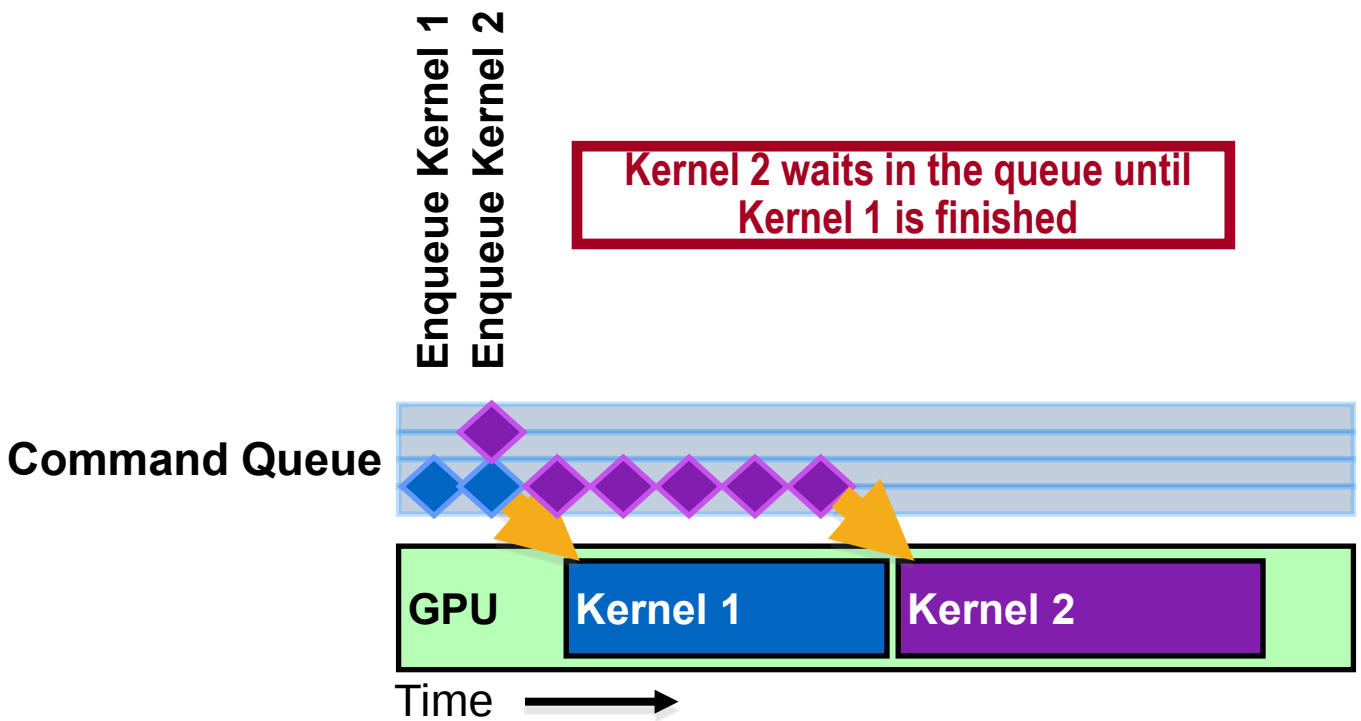
Synchronization Between Commands

- **Each *individual* queue can execute in order or out of order**
  - For in-order queue, all commands execute in order
  - Behaves as expected (as long as you're enqueuing from one thread)
- **You must *explicitly synchronize between queues***
  - Multiple devices each have their own queue
  - Use events to synchronize
- **Events**
  - Commands *return events* and *obey waitlists*
  - `clEnqueue*(..., num_events_in_waitlist, *event_waitlist, *event_out)`
- **User Events**
  - Allow developers to enqueue commands that wait on an external event
  - `clCreateUserEvent (context, errcode_ret)`
  - `clSetUserEventStatus (event, execution_status)`

KHRONOS GROUP

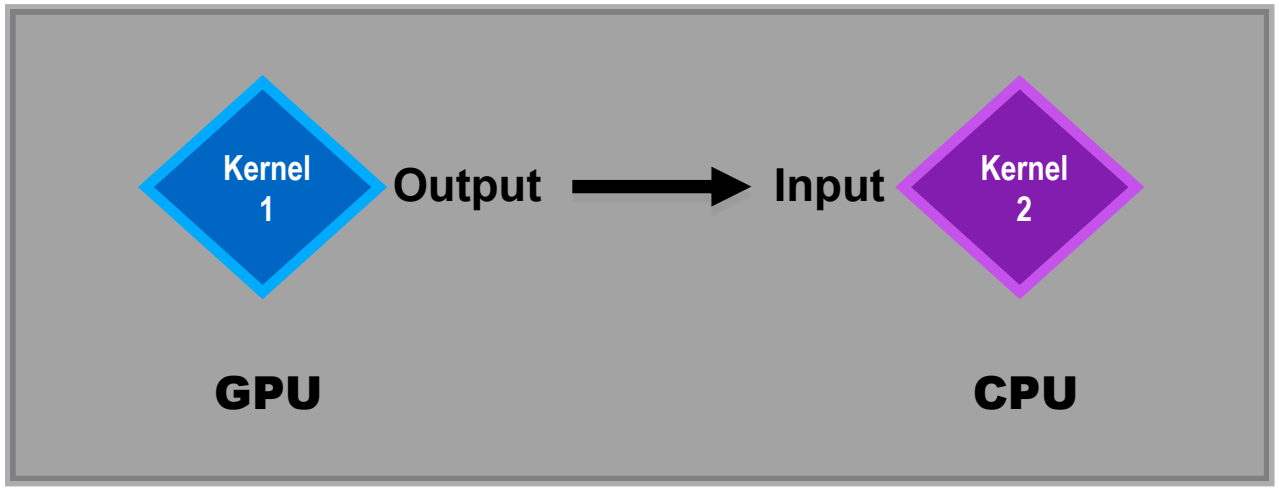
Synchronization: One Device/Queue

- Example: Kernel 2 uses the results of Kernel 1

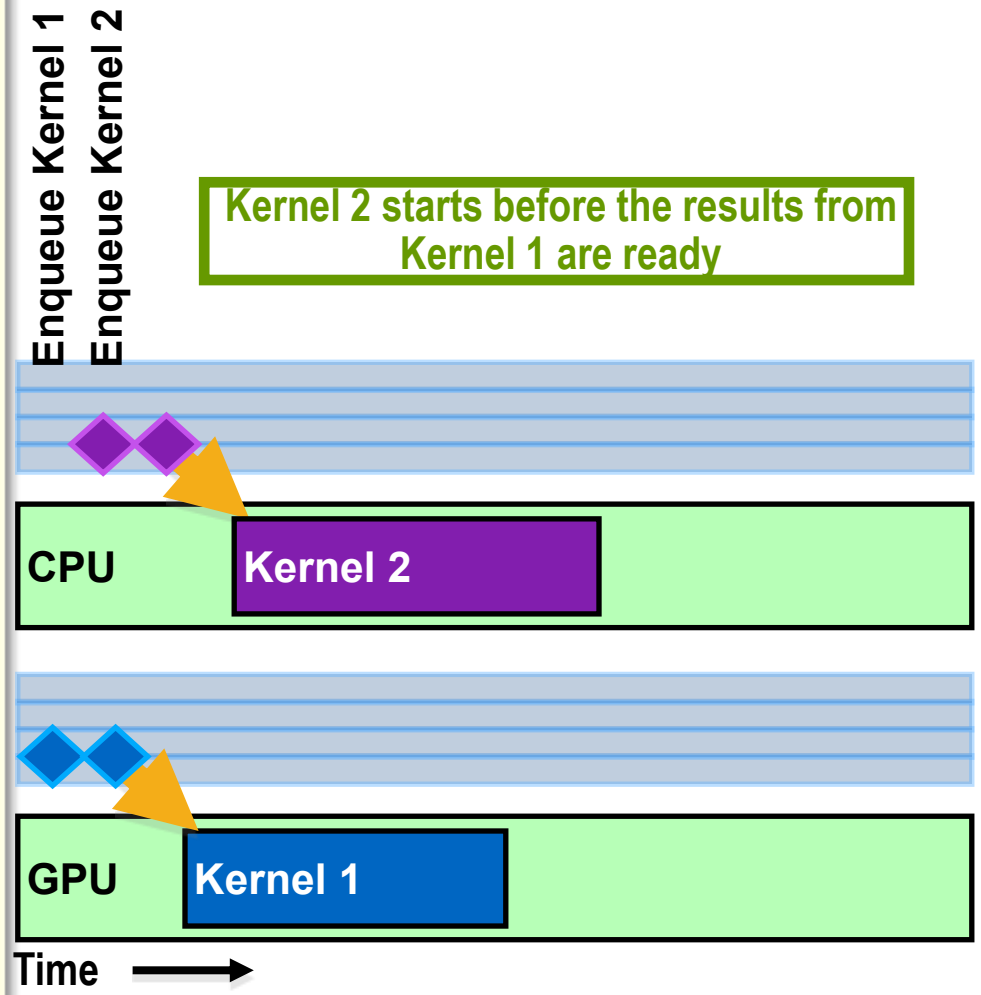


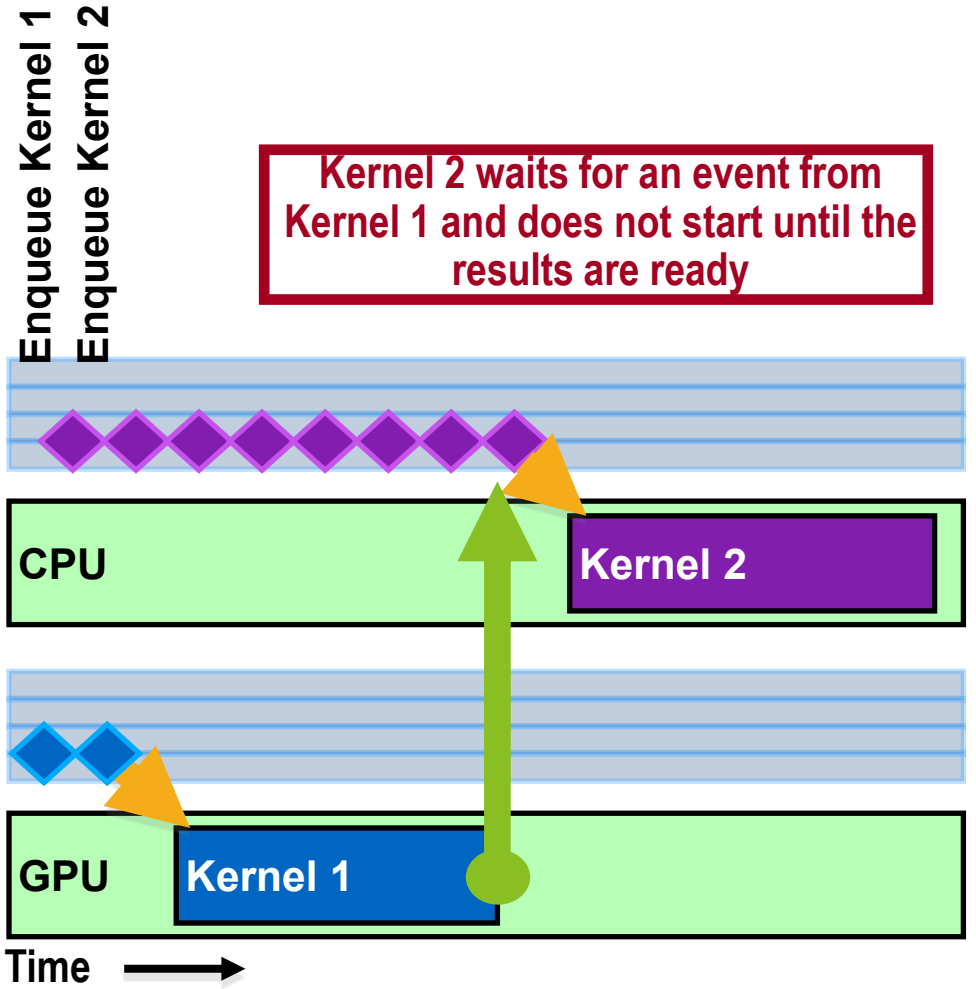
Synchronization: Two Devices/Queues

- Explicit dependency: Kernel 1 must finish before Kernel 2 starts



Synchronization: Two Devices/Queues

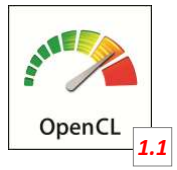




Using Events on the Host

- `clWaitForEvents(num_events, *event_list)`
  - Blocks until events are complete
- `clEnqueueMarker(queue, *event)`
  - Returns an event for a marker that moves through the queue
- `clEnqueueWaitForEvents(queue, num_events, *event_list)`
  - Inserts a “WaitForEvents” into the queue
- `clGetEventInfo()`
  - Command type and status

    `CL_QUEUED, CL_SUBMITTED, CL_RUNNING, CL_COMPLETE`, or error code
- `clGetEventProfilingInfo()`
  - Command queue, submit, start, and end times
- `clSetEventCallback()`
  - Called when command identified by event has completed



OpenCL C

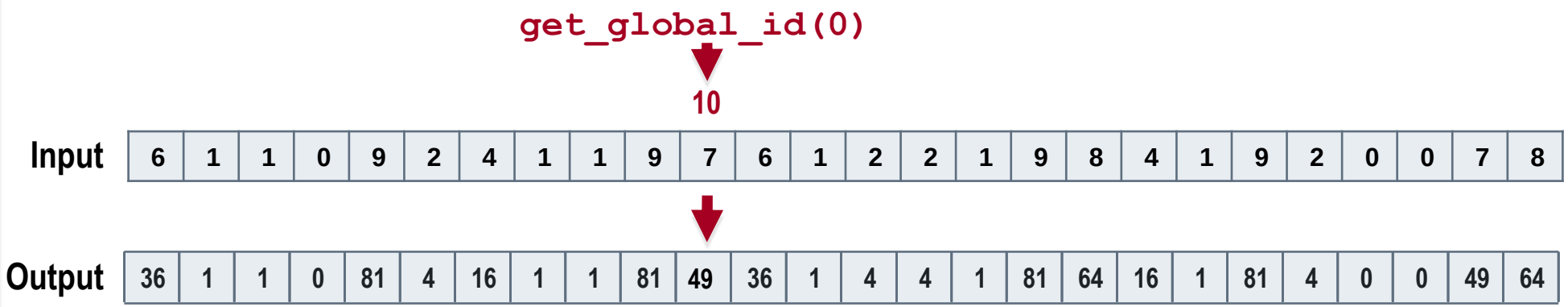
OpenCL C Language

- **Derived from ISO C99**
  - No standard C99 headers, function pointers, recursion, variable length arrays, and bit fields
- **Additions to the language for parallelism**
  - Work-items and workgroups
  - Vector types
  - Synchronization
- **Address space qualifiers**
- **Optimized image access**
- **Built-in functions**

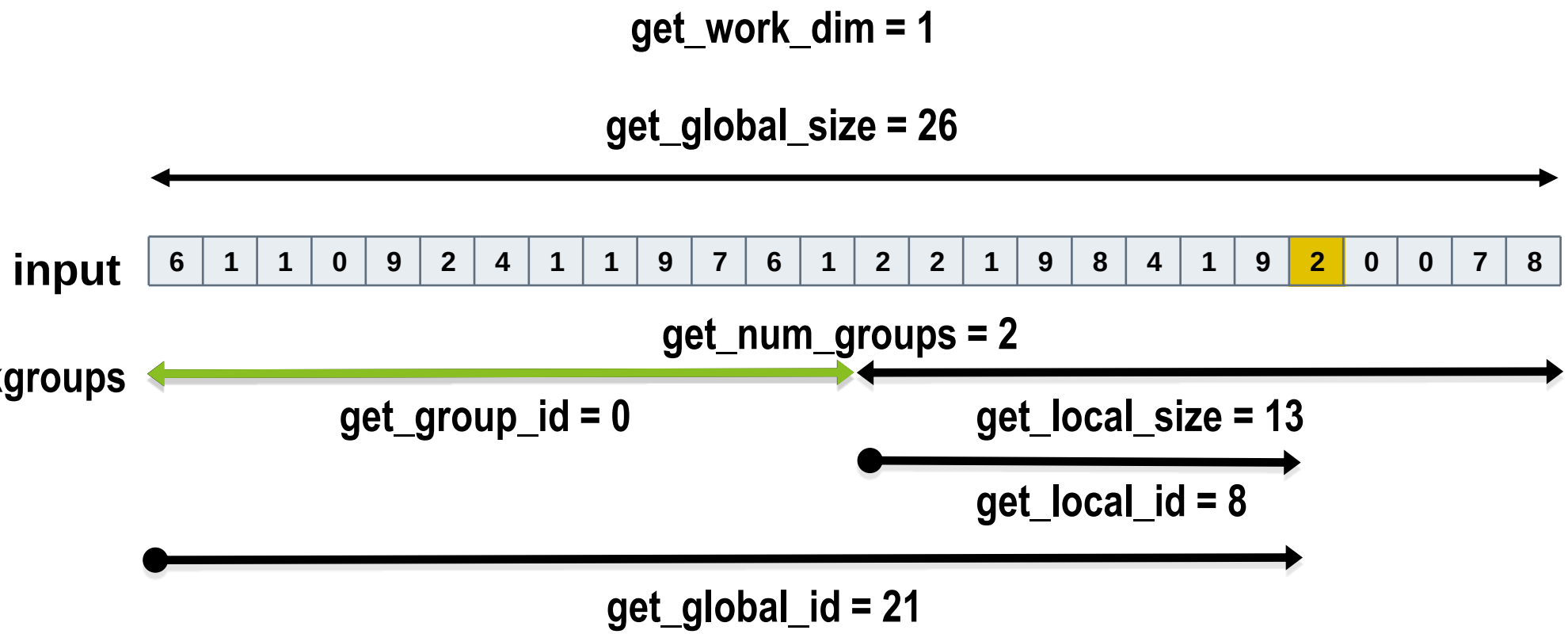
Kernel

- A data-parallel function executed for each work-item

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



Work-Items and Workgroup Functions



Data Types

- **Scalar data types**
  - char , uchar, short, ushort, int, uint, long, ulong
  - bool, intptr_t, ptrdiff_t, size_t, uintptr_t, void, half (storage)
- **Image types**
  - image2d_t, image3d_t, sampler_t
  - OpenCL 1.2 added image1d_t, image1d_buffer_t, image1d_array_t, image2d_array_t

- **Vector data types**
  - Portable
  - Vector length of 2, 4, 8, and 16
  - OpenCL 1.1 added vector length of 3 (aligned to 4)
  - char2, ushort4, int8, float16, double2, …
  - Endian safe
  - Aligned at vector length
  - Vector operations and built-in functions

Vector Operations

- Vector literal

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

- Vector components

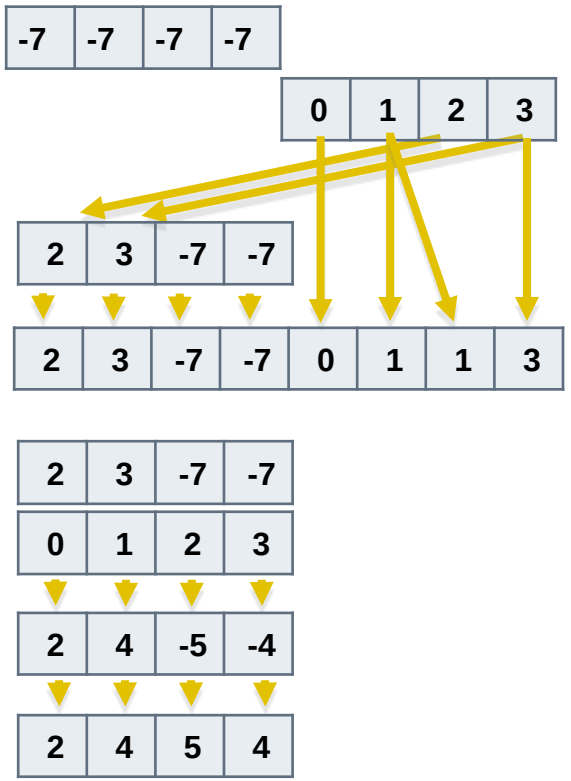
```
vi0.lo = vi1.hi;

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

- Vector ops

```
vi0 += vi1;

vi0 = abs(vi0);
```



Address Spaces

- Kernel pointer arguments must use global, local or constant

```
kernel void distance(global float8* stars, local float8* local_stars)
kernel void sum(private int* p)   // Illegal because is uses private
```

- Default address space for arguments and local variables is private

```
kernel void smooth(global float* io) {
  float temp;
  ...
}
```

- image2d_t and image3d_t are always in global address space

```
kernel void average(read_only global image_t in, write_only image2d_t out)
```

Address Spaces

- Program (global) variables must be in constant address space

```
constant float bigG = 6.67428E-11;
global   float time;              // Illegal non constant
kernel void force(global float4 mass) { time = 1.7643E18f; }
```

- Casting between different address spaces is undefined

```
kernel void calcEMF(global float4* particles) {
  global float* particle_ptr = (global float*) particles;
  float* private_ptr = (float*) particles; // Undefined behavior -
  float particle = * private_ptr;          //  different address
}
```

Conversions

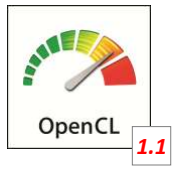
- Scalar and pointer conversions follow C99 rules
- No implicit conversions for vector types

```
float4 f4 = int4_vec;            // Illegal implicit conversion
```

- No casts for vector types (different semantics for vectors)

```
float4 f4 = (float4) int4_vec;  // Illegal cast
```

- Implicit Widening
  - OpenCL 1.0 requires widening for arithmetic operators
  - OpenCL 1.1 extends this feature to all operators
    - relational, equality, bitwise, logical and ternary




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

Conversions

- **Explict conversions: convert_*destType*<_sat><_roundingMode>**
  - Scalar and vector types
  - No ambiguity

```
uchar4 c4 = convert_uchar4_sat_rte(f4);
```

f4	-5.0f	254.5f	254.6	1.2E9f
c4	0	254	255	255

Saturate to 0

Round down to nearest even

Round up to nearest value

Saturated to 255

Reinterpret Data: *as_typen*

- Reinterpret the bits to another type
- Types must be the same size
- OpenCL provides a select built-in

```
// f[i] = f[i] < g[i] ? f[i] : 0.0f
float4 f, g;
int4 is_less = f < g;
f = as_float4(as_int4(f) & is_less);
```

f	-5.0f	254.5f	254.6f	1.2E9f
g	254.6f	254.6f	254.6f	254.6f
is_less	ffffffff	ffffffff	00000000	00000000
as_int	c0a00000	42fe0000	437e8000	4e8f0d18
&	c0a00000	42fe0000	00000000	00000000
f	-5.0f	254.5f	0.0f	0.0f

Built-in Math Functions

- IEEE 754 compatible rounding behavior for single precision floating-point
- IEEE 754 compliant behavior for double precision floating-point
- Defines maximum error of math functions as ULP values
- Handle ambiguous C99 library edge cases
- Commonly used single precision math functions come in three flavors
  - eg. log(x)
    - Full precision <= 3ulps
    - Half precision/faster. half_log—minimum 11 bits of accuracy, <= 8192 ulps
    - Native precision/fastest. native_log: accuracy is implementation defined
  - Choose between accuracy and performance

Built-in Work-group Functions

- **Synchronization**
  - `Barrier`
- **Work-group functions**
  - Encountered by all work-items in the work-group
  - With the same argument values

```
kernel read(global int* g, local int* shared) {
  if (get_global_id(0) < 5)
    barrier(CLK_GLOBAL_MEM_FENCE);
  else
    k = array[0];
}
```

← work-item 0

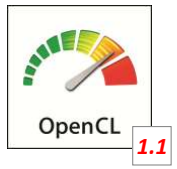
← work-item 6

Illegal since not all work-items encounter barrier

Built-in Work-group Functions

- `async_work_group_copy`
  - Copy from global to local or local to global memory
  - Use DMA engine or do a memcpy across work-items in work-group
  - Returns an event object

- `async_work_group_strided_copy`
  - Specify a stride on access

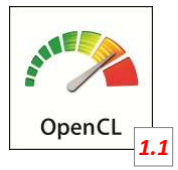


- `wait_group_events`
  - wait for events that identify `async_work_group_copy` operations to complete

Built-in Functions

- Integer functions
  - `abs, abs_diff, add_sat, hadd, rhadd, clz, mad_hi, mad_sat, max, min, mul_hi, rotate, sub_sat, upsample, clamp (OpenCL 1.1)`
- Image functions
  - `read_image[f | i | ui]`
  - `write_image[f | i | ui]`
  - `get_image_[width | height | depth]`
- Common, Geometric and Relational Functions
- Vector Data Load and Store Functions
  - eg. `vload_half, vstore_half, vload_halfn, vstore_halfn`, ...

- Vector shuffle
  - Construct a runtime permutation of elements from 1 or 2 vectors and a mask
- 32bit Atomic functions to global and local memory
  - add, sub, xchg, inc, dec, cmp_xchg, min, max, and, or, xor



Built-in Functions

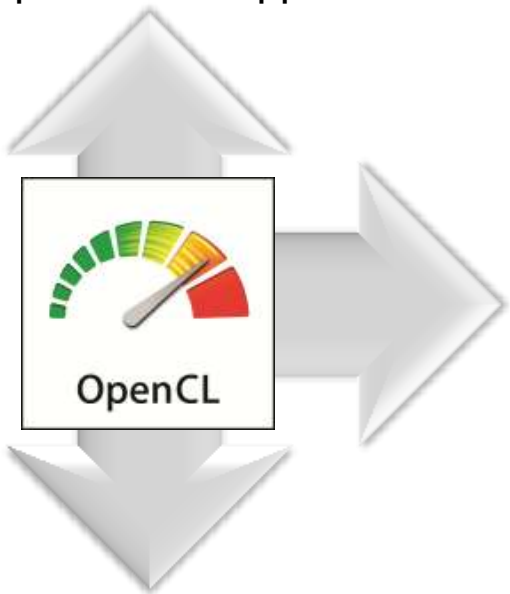
Math Functions
gentype acos (gentype)
gentype acosh (gentype)
gentype acospi (gentype x)
gentype asin (gentype)
gentype asinh (gentype)
gentype asinpi (gentype x)
gentype atan (gentype y_over_x)
gentype atan2 (gentype y, gentype x)
gentype atanh (gentype)
gentype atanpi (gentype x)
gentype atan2pi (gentype y, gentype x)
gentype cbrt (gentype)
gentype ceil (gentype)
gentype copysign (gentype x, gentype y)
gentype cos (gentype)
gentype cosh (gentype)
gentype cospi (gentype x)
gentype erfc (gentype)
gentype erf (gentype)
gentype exp (gentype x)
gentype exp2 (gentype)
gentype exp10 (gentype)
gentype expm1 (gentype x)
gentype fabs (gentype)
gentype fdim (gentype x, gentype y)
gentype floor (gentype)
gentype fma (gentype a, gentype b, gentype c)
gentype fmax (gentype x, gentype y)
gentype fmax (gentype x, float y)
gentype fmin (gentype x, gentype y)
gentype fmin (gentype x, float y)
gentype fmod (gentype x, gentype y)
gentype fract (gentype x, gentype *iptr)
gentype frexp (gentype x, intn *exp)
gentype hypot (gentype x, gentype y)
intn ilogb (gentype x)
gentype ldexp (gentype x, intn n)
gentype ldexp (gentype x, int n)
gentype lgamma (gentype x)
gentype lgamma_r (gentype x, intn *signp)
gentype log (gentype)
gentype log2 (gentype)
gentype log10 (gentype)
gentype log1p (gentype x)
gentype logb (gentype x)
gentype mad (gentype a, gentype b, gentype c)
gentype modf (gentype x, gentype *iptr)
gentype nan (uintn nancode)
gentype nextafter (gentype x, gentype y)
gentype pow (gentype x, gentype y)
gentype pown (gentype x, intn y)
gentype powr (gentype x, gentype y)
gentype remainder (gentype x, gentype y)
gentype remquo (gentype x, gentype y, intn *quo)
gentype rint (gentype)
gentype rootn (gentype x, intn y)
gentype round (gentype x)
gentype rsqrt (gentype)
gentype sin (gentype)
gentype sincos (gentype x, gentype *cosval)
gentype sinh (gentype)
gentype sinpi (gentype x)
gentype sqrt (gentype)
gentype tan (gentype)
gentype tanh (gentype)
gentype tanpi (gentype x)
gentype tgamma (gentype)
gentype trunc (gentype)
Integer Ops
ugentype abs (gentype x)
ugentype abs_diff (gentype x, gentype y)
gentype add_sat (gentype x, gentype y)
gentype hadd (gentype x, gentype y)
gentype rhadd (gentype x, gentype y)
gentype clz (gentype x)
gentype mad_hi (gentype a, gentype b, gentype c)
gentype mad_sat (gentype a, gentype b, gentype c)
gentype max (gentype x, gentype y)
gentype min (gentype x, gentype y)
gentype mul_hi (gentype x, gentype y)
gentype rotate (gentype v, gentype i)
gentype sub_sat (gentype x, gentype y)
shortn upsample (intn hi, uintn lo)
ushortn upsample (uintn hi, uintn lo)
intn upsample (intn hi, uintn lo)
uintn upsample (uintn hi, uintn lo)
longn upsample (intn hi, uintn lo)
ulongnn upsample (uintn hi, uintn lo)
gentype mad24 (gentype x, gentype y, gentype z)
gentype mul24 (gentype x, gentype y)
Common Functions
gentype clamp (gentype x, gentype minval, gentype maxval)
gentype clamp (gentype x, float minval, float maxval)
gentype degrees (gentype radians)
gentype max (gentype x, gentype y)
gentype max (gentype x, float y)
gentype min (gentype x, gentype y)
gentype min (gentype x, float y)
gentype mix (gentype x, gentype y, gentype a)
gentype mix (gentype x, gentype y, float a)
gentype radians (gentype degrees)
gentype sign (gentype x)
Geometric Functions
float4 cross (float4 p0, float4 p1)
float dot (gentype p0, gentype p1)
float distance (gentype p0, gentype p1)
float length (gentype p)
float fast_distance (gentype p0, gentype p1)
float fast_length (gentype p)
gentype fast_normalize (gentype p)
Relational Ops
int isequal (float x, float y)
intn isequal (floatn x, floatn y)
int isnotequal (float x, float y)
intn isnotequal (floatn x, floatn y)
int isgreater (float x, float y)
intn isgreater (floatn x, floatn y)
int isgreaterequal (float x, float y)
intn isgreaterequal (floatn x, floatn y)
int isless (float x, float y)
intn isless (floatn x, floatn y)
int islessequal (float x, float y)
intn islessequal (floatn x, floatn y)
int islessgreater (float x, float y)
intn islessgreater (floatn x, floatn y)
int isfinite (float)
intn isfinite (floatn)
int isnan (float)
intn isnan (floatn)
int isnormal (float)
intn isnormal (floatn)
int isordered (float x, float y)
intn isordered (floatn x, floatn y)
int isunordered (float x, float y)
intn isunordered (floatn x, floatn y)
int signbit (float)
intn signbit (floatn)
int any (igentype x)
int all (igentype x)
gentype bitselect (gentype a, gentype b, gentype c)
gentype select (gentype a, gentype b,igentype c)
gentype select (gentype a, gentype b,ugentype c)
Vector Loads/Store Functions
gentypen vloadn (size_t offset, const global gentype *p)
gentypen vloadn (size_t offset, const __local gentype *p)
gentypen vloadn (size_t offset, const __constant gentype *p)
gentypen vloadn (size_t offset, const __private gentype *p)
void vstoren (gentypen data, size_t offset, global gentype *p)
void vstoren (gentypen data, size_t offset, __local gentype *p)
void vstoren (gentypen data, size_t offset, __private gentype *p)
void vstore_half (float data, size_t offset, global half *p)
void vstore_half_rte (float data, size_t offset, global half *p)
void vstore_half_rtz (float data, size_t offset, global half *p)
void vstore_half_rtp (float data, size_t offset, global half *p)
void vstore_half_rtn (float data, size_t offset, global half *p)
void vstore_half (float data, size_t offset, __local half *p)
void vstore_half_rte (float data, size_t offset, __local half *p)
void vstore_half_rtz (float data, size_t offset, __local half *p)
void vstore_half_rtp (float data, size_t offset, __local half *p)
void vstore_half_rtn (float data, size_t offset, __local half *p)
void vstore_half (float data, size_t offset, __private half *p)
void vstore_half_rte (float data, size_t offset, __private half *p)
void vstore_half_rtz (float data, size_t offset, __private half *p)
void vstore_half_rtp (float data, size_t offset, __private half *p)
void vstore_half_rtn (float data, size_t offset, global half *p)
void vstore_halfn (floatn data, size_t offset, global half *p)
void vstore_halfn_rte (floatn data, size_t offset, global half *p)
void vstore_halfn_rtz (floatn data, size_t offset, global half *p)
void vstore_halfn_rtp (floatn data, size_t offset, global half *p)
void vstore_halfn_rtn (floatn data, size_t offset, global half *p)
void vstore_halfn (floatn data, size_t offset, __local half *p)
void vstore_halfn_rte (floatn data, size_t offset, __local half *p)
void vstore_halfn_rtz (floatn data, size_t offset, __local half *p)
void vstore_halfn_rtp (floatn data, size_t offset, __local half *p)
void vstore_halfn_rtn (floatn data, size_t offset, __local half *p)
void vstore_halfn (floatn data, size_t offset, __private half *p)
void vstore_halfn_rte (floatn data, size_t offset, __private half *p)
void vstore_halfn_rtz (floatn data, size_t offset, __private half *p)
void vstore_halfn_rtp (floatn data, size_t offset, __private half *p)
void vstore_halfn_rtn (floatn data, size_t offset, __private half *p)
void vstorea_halfn (floatn data, size_t offset, global half *p)
void vstorea_halfn_rte (floatn data, size_t offset, global half *p)
void vstorea_halfn_rtz (floatn data, size_t offset, global half *p)
void vstorea_halfn_rtp (floatn data, size_t offset, global half *p)
void vstorea_halfn_rtn (floatn data, size_t offset, global half *p)
void vstorea_halfn (floatn data, size_t offset, __local half *p)
void vstorea_halfn_rte (floatn data, size_t offset, __local half *p)
void vstorea_halfn_rtz (floatn data, size_t offset, __local half *p)
void vstorea_halfn_rtp (floatn data, size_t offset, __local half *p)
void vstorea_halfn_rtn (floatn data, size_t offset, __local half *p)
void vstorea_halfn (floatn data, size_t offset, __private half *p)
void vstorea_halfn_rte (floatn data, size_t offset, __private half *p)
void vstorea_halfn_rtz (floatn data, size_t offset, __private half *p)
void vstorea_halfn_rtp (floatn data, size_t offset, __private half *p)
void vstorea_halfn_rtn (floatn data, size_t offset, __private half *p)

Agenda

- OpenCL in context
- OpenCL Overview
- Next steps for OpenCL

Looking Forward

OpenCL-HLM
Exploring high-level programming model, unifying host and device execution environments through language syntax for increased usability and broader optimization opportunities



Long-term Core Roadmap
Exploring enhanced memory and execution model flexibility expose emerging hardware capabilities

WebCL
Bring parallel computation to the Web through a JavaScript binding to OpenCL

WebCL

OpenCL-SPIR
Exploring low-level Intermediate Representation for code obfuscation/security and to provide target back-end for alternative high-level languages