Lecture 2
Host programming

Outline

- **Introduction to OpenCL**
- **OpenCL Framework**
  - Platform layer
  - Compiler for OpenCL C
  - Runtime
- **The Flow of Host Program**
  - Example: Hello world

Outline

- Introduction to OpenCL
- **OpenCL Framework**
  - Platform layer
  - Compiler for OpenCL C
  - Runtime
- **The Flow of Host Program**
  - Example: Hello world

What is OpenCL

- **OpenCL (Open Computing Language)**
  - is a framework suited for parallel programming of heterogeneous systems.

- **The framework supports heterogeneous accelerated processing units**
  - execution on CPU, DSPs, FPGAs, GPUs, and other.

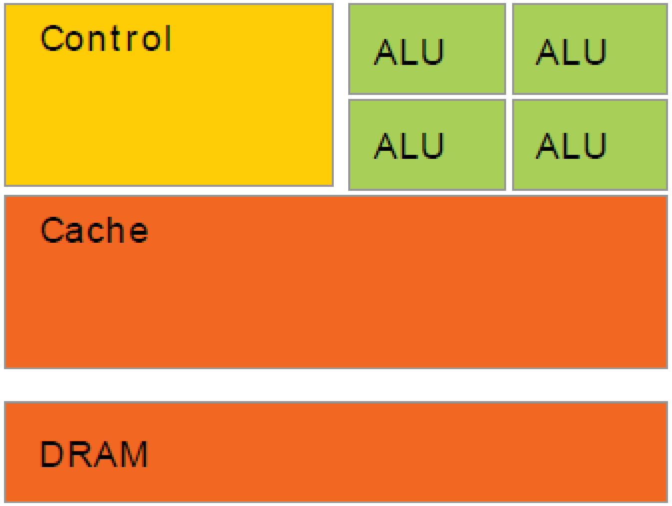
- **OpenCL provides parallel computing**
  - task-based parallelism
  - data-based parallelism

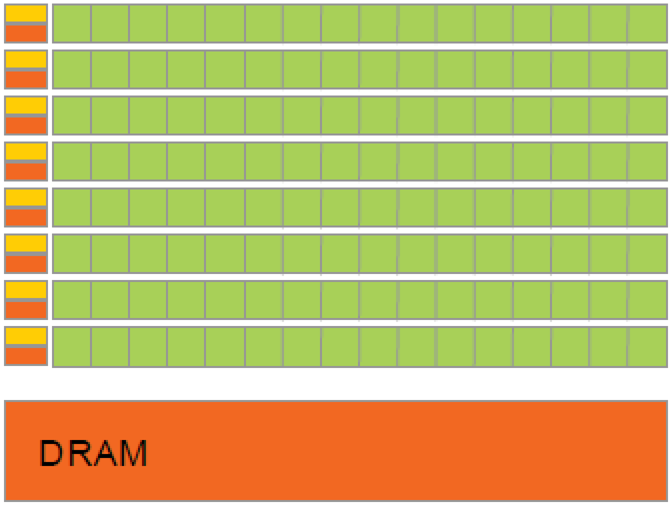
Design Goals of OpenCL

- **Use all computational resources in the system**
  - CPUs, GPUs and other processors as peers
- **Efficient parallel programming model**
  - Based on C99
  - Data- and task-parallel computational model
  - Abstract the specifics of underlying hardware
  - Specify accuracy of floating-point computations
- **Desktop and Handheld Profiles**

CPU-GPU Cooperation (1/2)

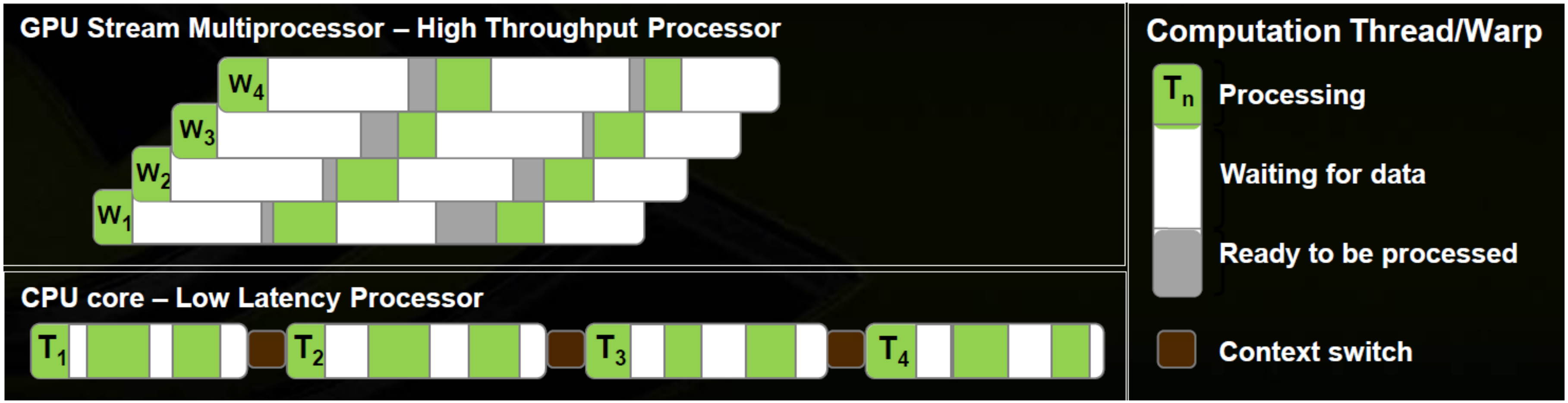
- **CPU**
  - Optimized for low-latency access to cached data sets
  - Control logic for out-of-order and speculative execution
- **GPU**
  - Optimized for data-parallel, throughput computation
  - Architecture tolerant of memory latency
  - More transistors dedicated to computation





CPU-GPU Cooperation (2/2)

- CPU architecture must minimize latency within each thread
- GPU architecture hides latency with computation from other thread warps



http://www.cc.gatech.edu/~vetter/keeneland/tutorial-2011-04-14/02-cuda-overview.pdf

OpenCL Specification

1. **OpenCL application**
2. **Platform model**
   – Define the host and devices.
3. **Execution model**
   – Defines the OpenCL environment on the host and how kernels are executed on the device.
4. **Memory model**
   – Defines the abstract memory hierarchy that kernels use
5. **Programming model**
   – Defines how the concurrency model is mapped to physical hardware.

OpenCL Application (1/2)

OpenCL Code Consists of Two Part:

Host code:

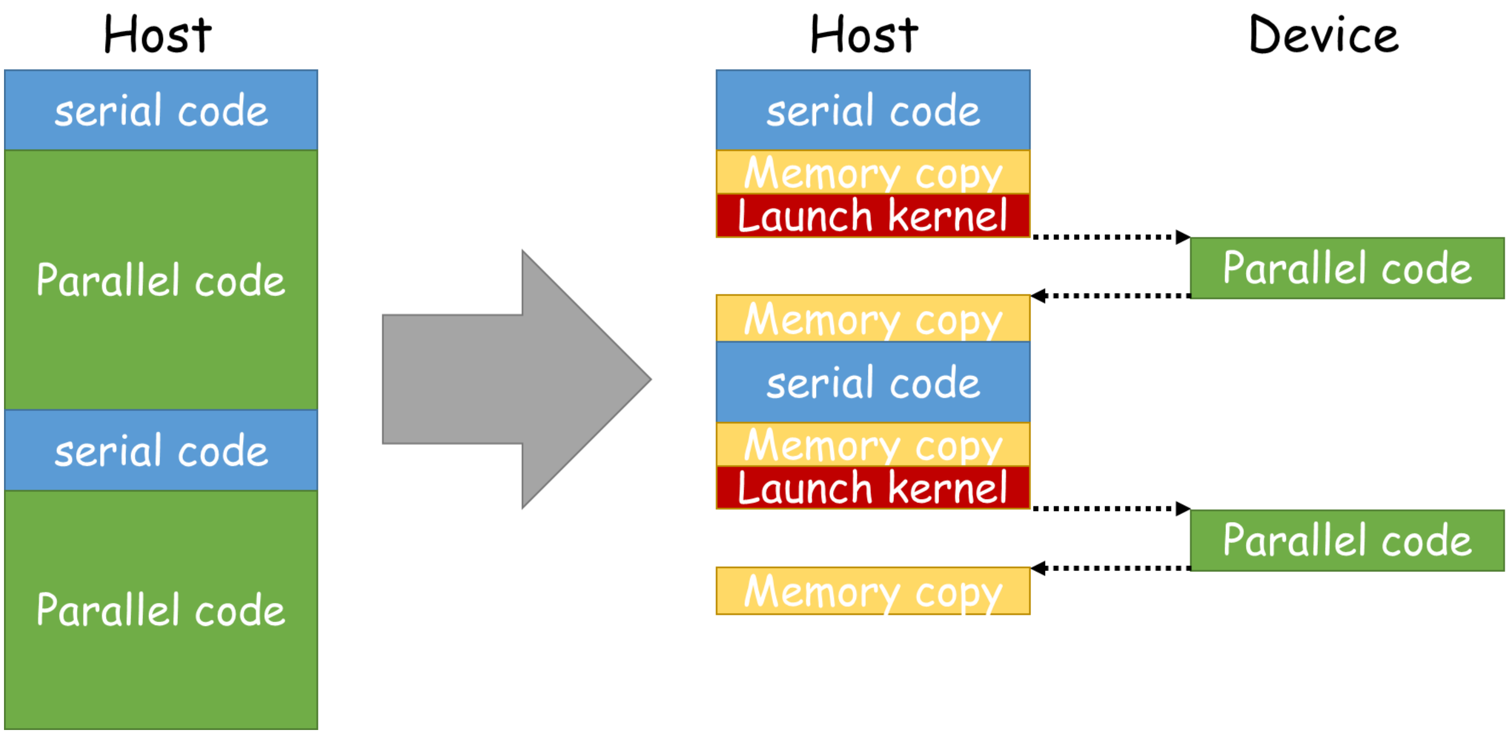
- Written in regular C or C++
- Creating the data structures that manage the host-device communication
- Run on the host (like CPU)

Kernel code:

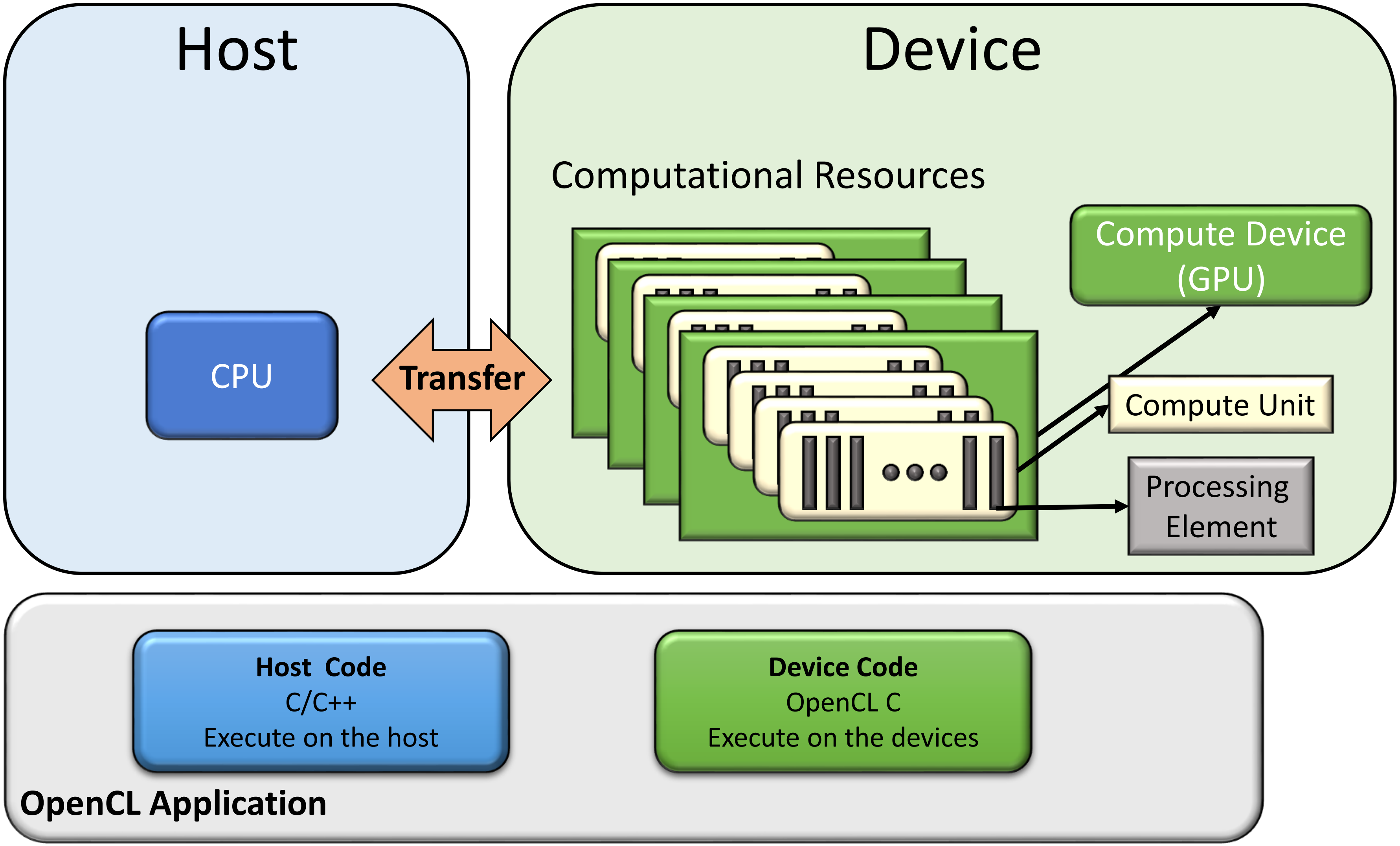
- Using the high-performance capabilities defined in the OpenCL standard.
- Run on the devices (like GPU)

OpenCL Application (2/2)

- Serial code executes in a Host (CPU) thread.
- Parallel code executes in many Device (GPU) threads across multiple processing elements.



OpenCL Platform Model



OpenCL Execution Model (1/3)

Decompose Task Into *work-items*

- **Define N-dimensional computation domain**
- **Execute a *kernel* at each point in computation domain**

• Traditional loop as function in C

```
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];
}
```

• OpenCL C code

```
__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"
```

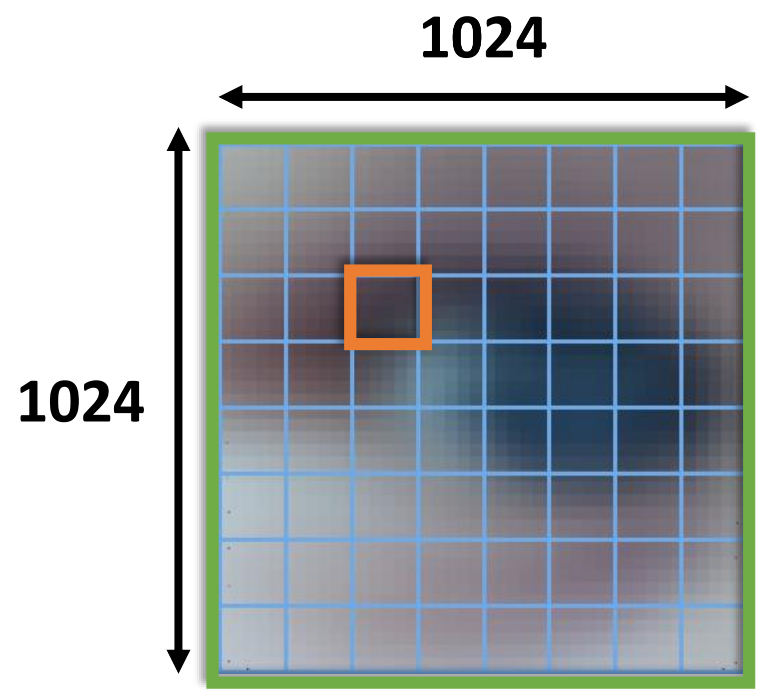
OpenCL Execution Model (2/3)

An N-dimension Domain of Work-items

- Kernels are executed across a global domain of *work-items*
- Work-items are grouped into local domain of *work-groups*

Global dimensions: 1024 x 1024

Local dimensions: 128 x 128

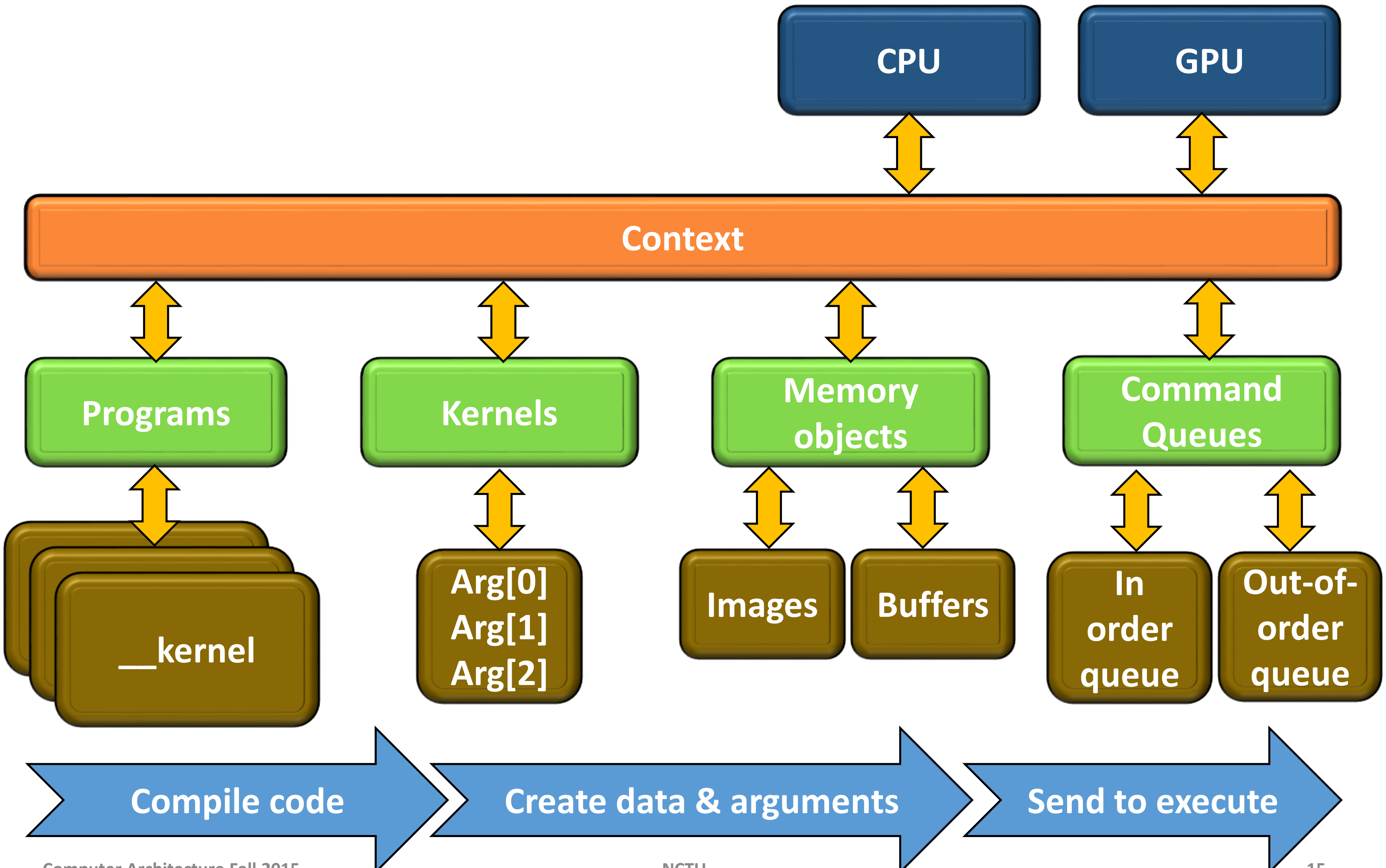


OpenCL Execution Model (3/3)

The application runs on the host which submits task to the devices

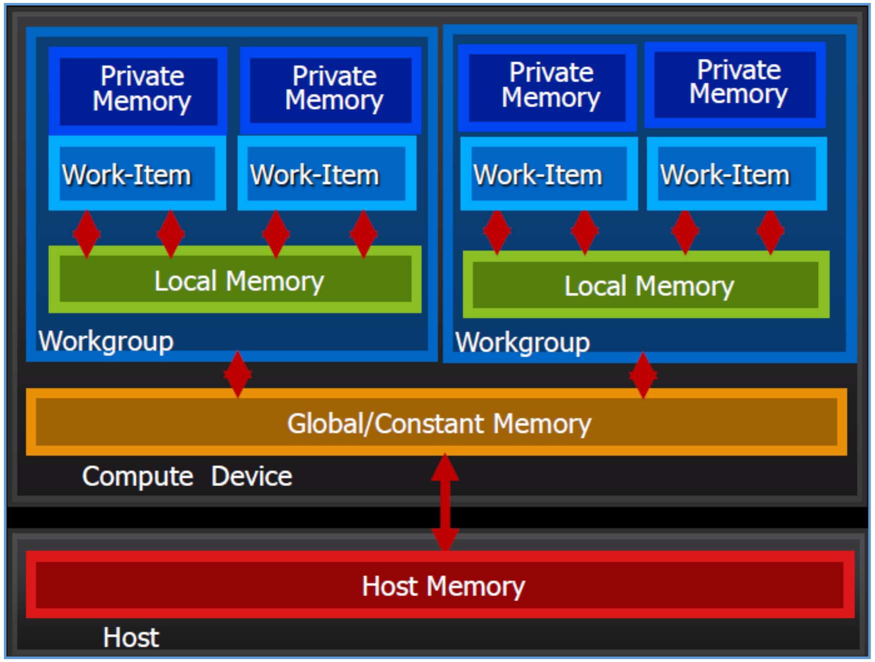
- Work-item:
  the basic unit of work on the OpenCL device
- Kernel:
  the code for a work-item (basically a C function)
- Program:
  Collection of kernels and other functions

OpenCL Framework



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



OpenCL Programming Flow (1/5)

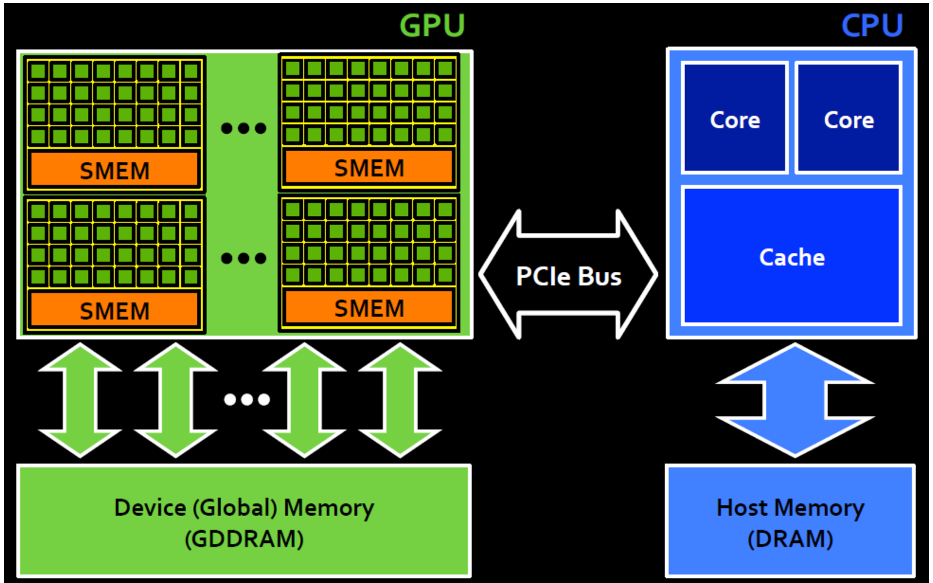
Standard Execute Flow

Step 1: Copy data to GPU memory

Step 2: Launch the kernels on GPU

Step 3: Execute kernels on GPU

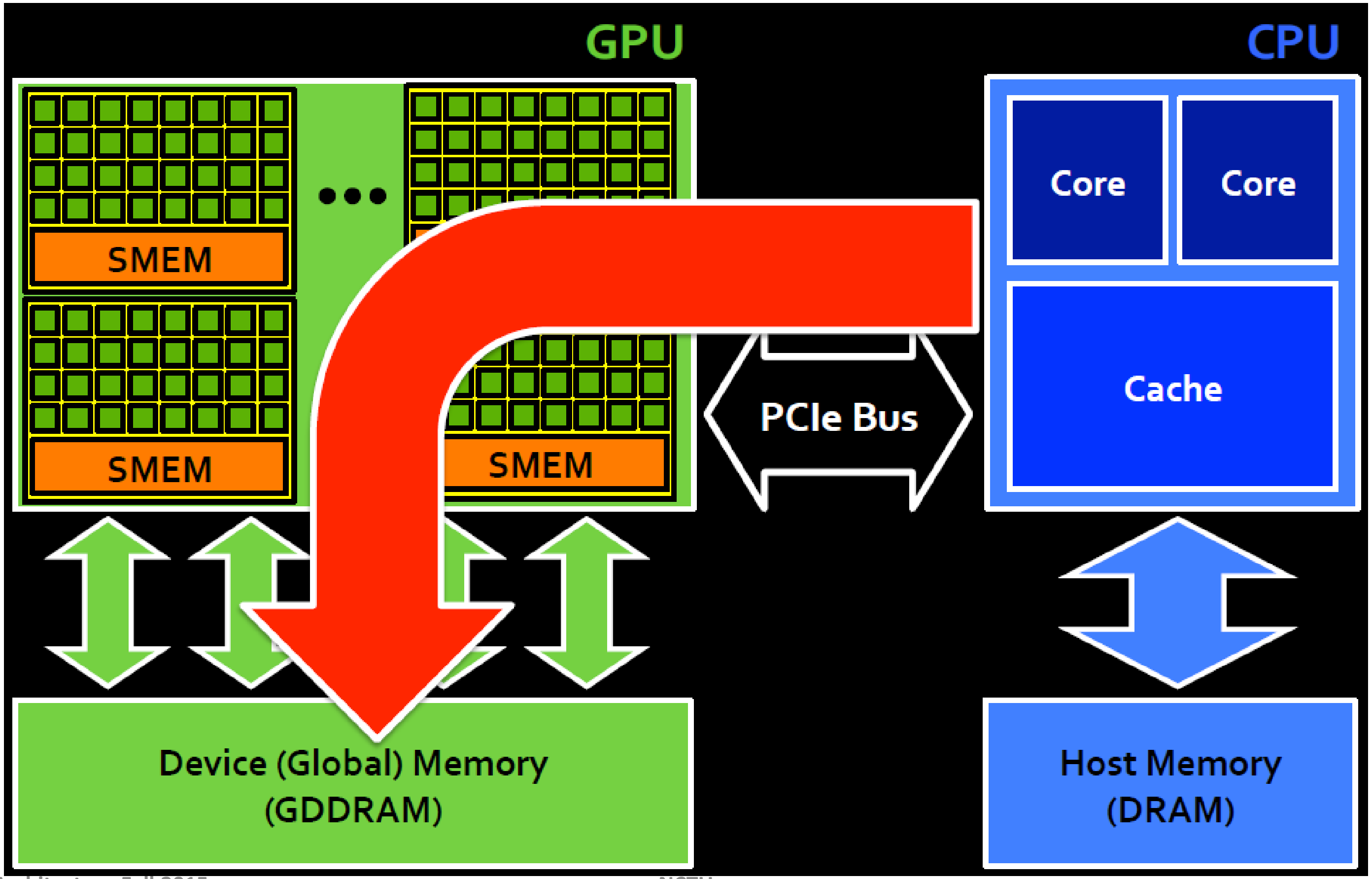
Step 4: Copy data to CPU memory



http://mc.stanford.edu/cgi-bin/images/f/f7/Darve_cme343_cuda_1.pdf

OpenCL Programming Flow (2/5)

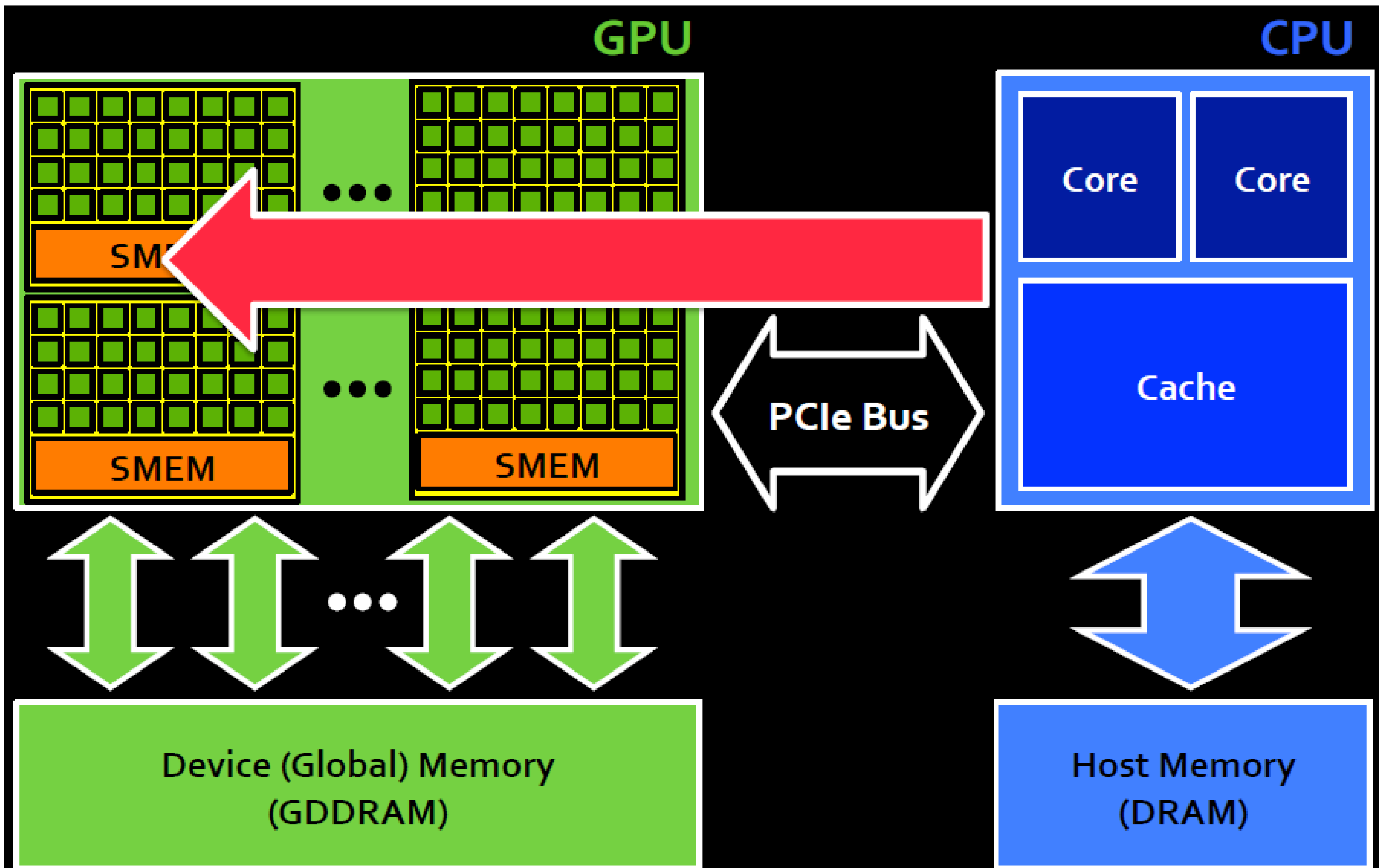
Step 1: Copy Data to GPU Memory



http://mc.stanford.edu/cgi-bin/images/f/f7/Darve_cme343_cuda_1.pdf

OpenCL Programming Flow (3/5)

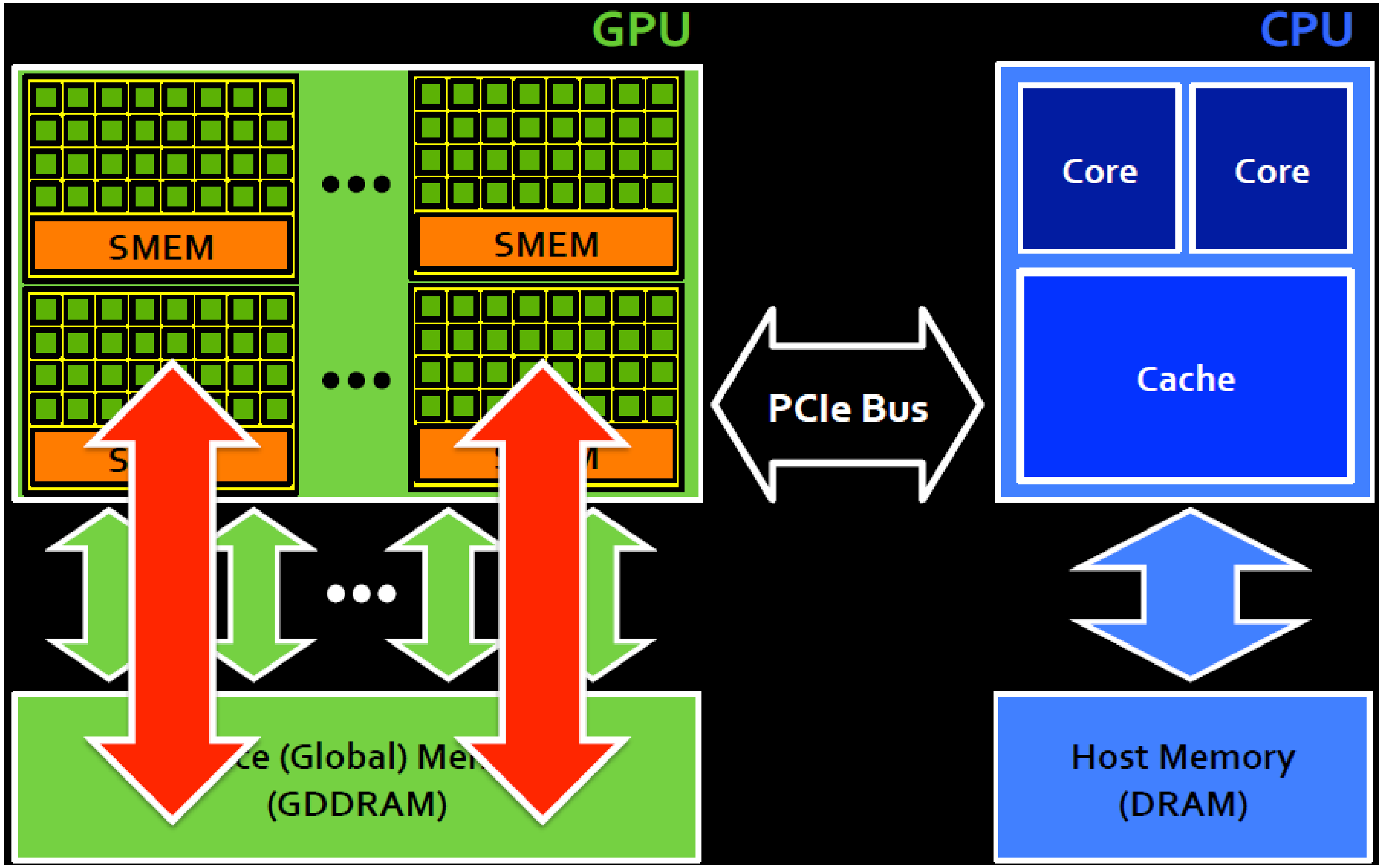
Step 2: Launch The Kernels on GPU



http://mc.stanford.edu/cgi-bin/images/f/f7/Darve_cme343_cuda_1.pdf

OpenCL Programming Flow (4/5)

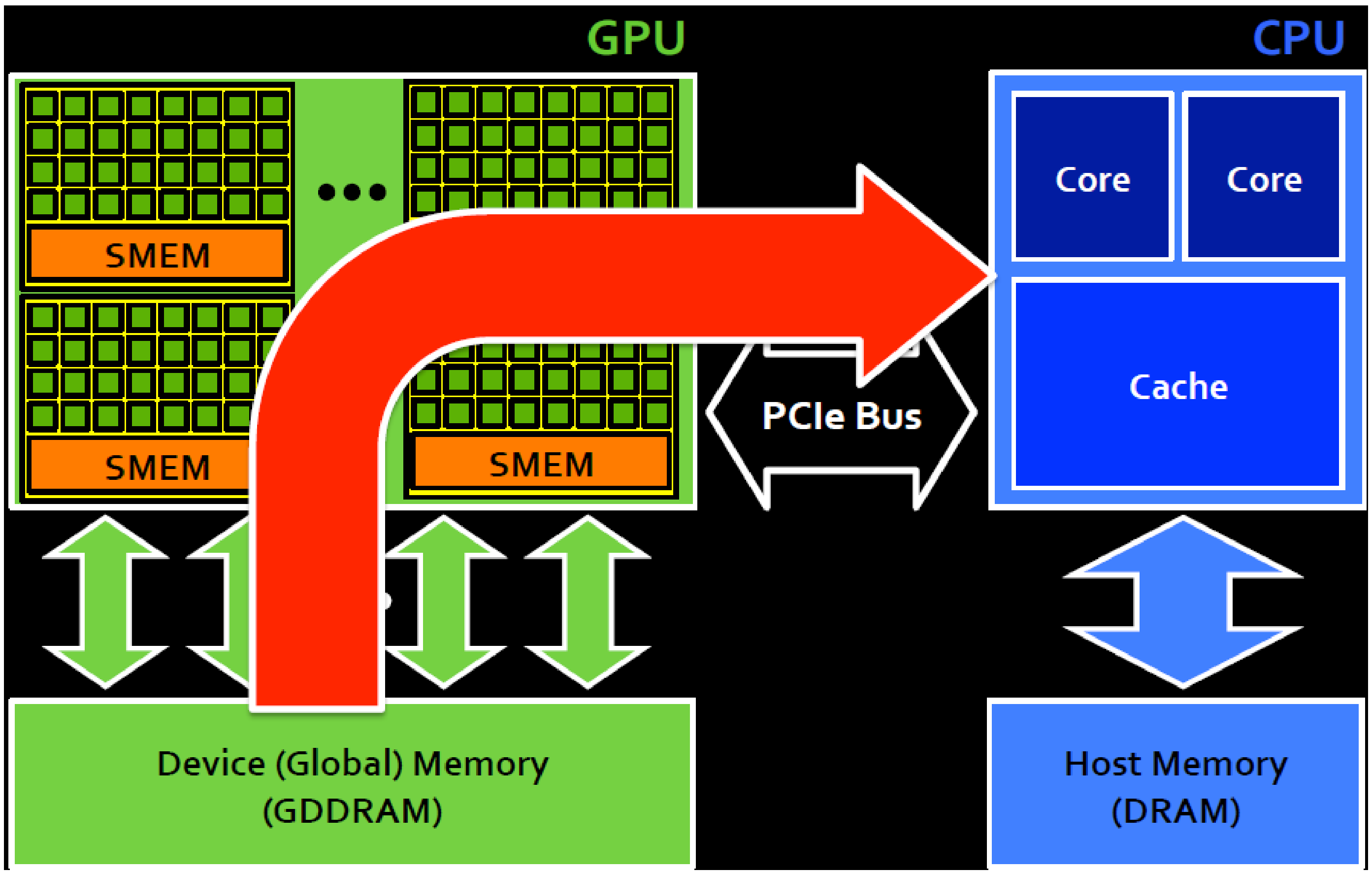
Step 3: Execute Kernels on GPU



http://mc.stanford.edu/cgi-bin/images/f/f7/Darve_cme343_cuda_1.pdf

OpenCL Programming Flow (5/5)

Step 4: Copy Data to CPU Memory



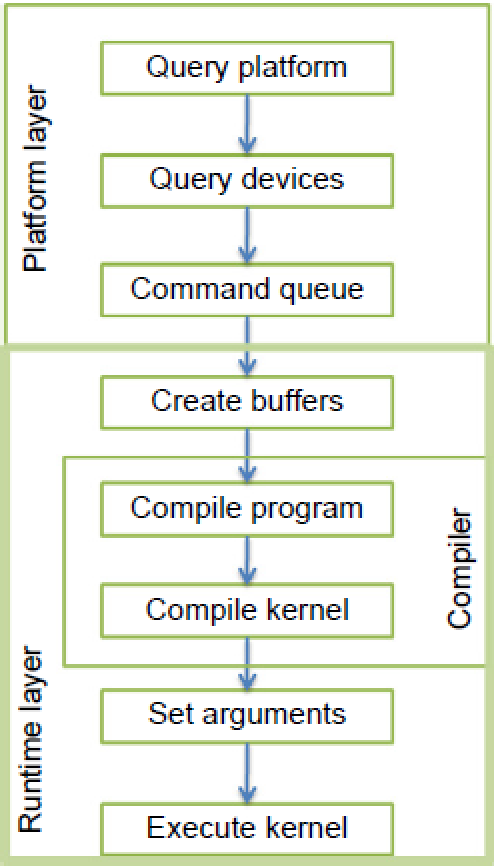
http://mc.stanford.edu/cgi-bin/images/f/f7/Darve_cme343_cuda_1.pdf

Outline

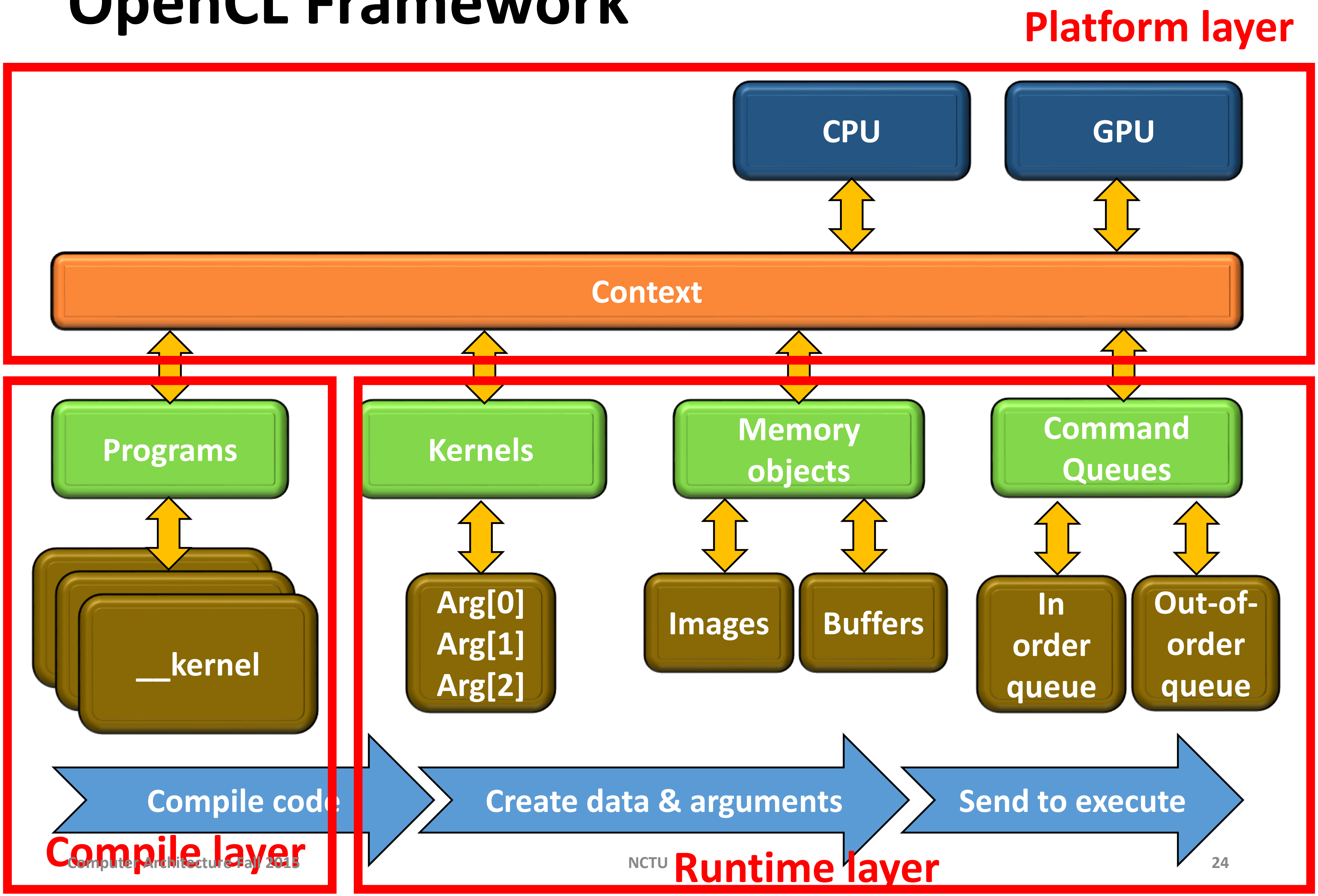
- **Introduction to OpenCL**
- **OpenCL Framework**
  - Platform layer
  - Compiler for OpenCL C
  - Runtime
- **The Flow of Host Program**
  - Example: Hello world

OpenCL Framework

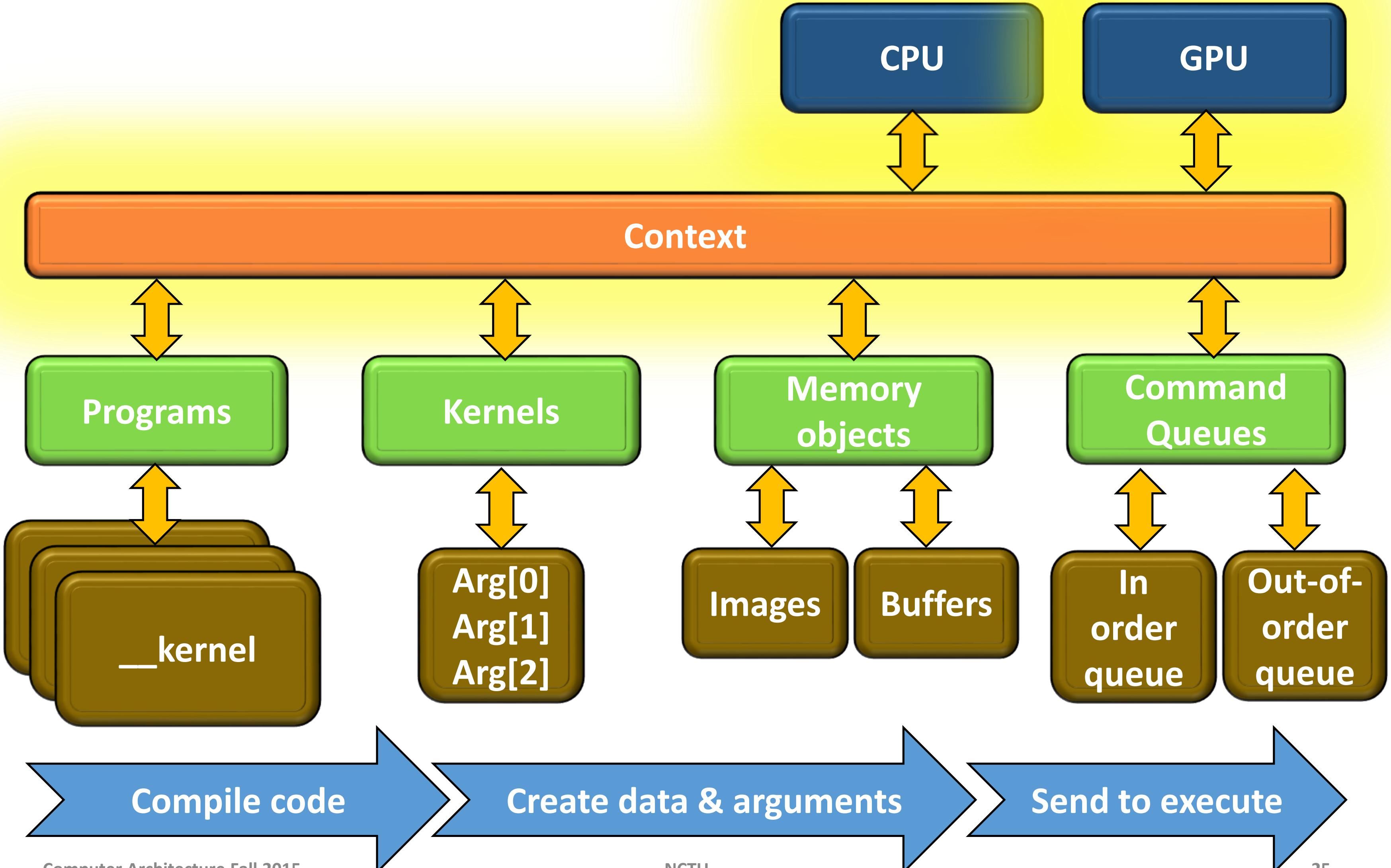
- **Platform layer**
  - Platform query and context creation
- **Compiler for OpenCL C**
- **Runtime**
  - Memory management and command execution within a context



OpenCL Framework



OpenCL Framework
Platform layer



OpenCL Framework
Platform layer

Create Platform

- *clGetPlatformIDs():* **Obtain the list of platforms available.**
- *clGetPlatformInfo():* **Name, version, vendor, extensions**
- *clGetDeviceIDs():*
  Obtain the list of devices available on a platform.
- *clGetDeviceInfo():* **Name, type, capabilities**

OpenCL Framework
Platform layer

Create Context

clCreateContext(): **Creates an OpenCL context.**

Context =

cl_context

One or more devices

cl_dvice_id

Memory and device code

cl_mem cl_program

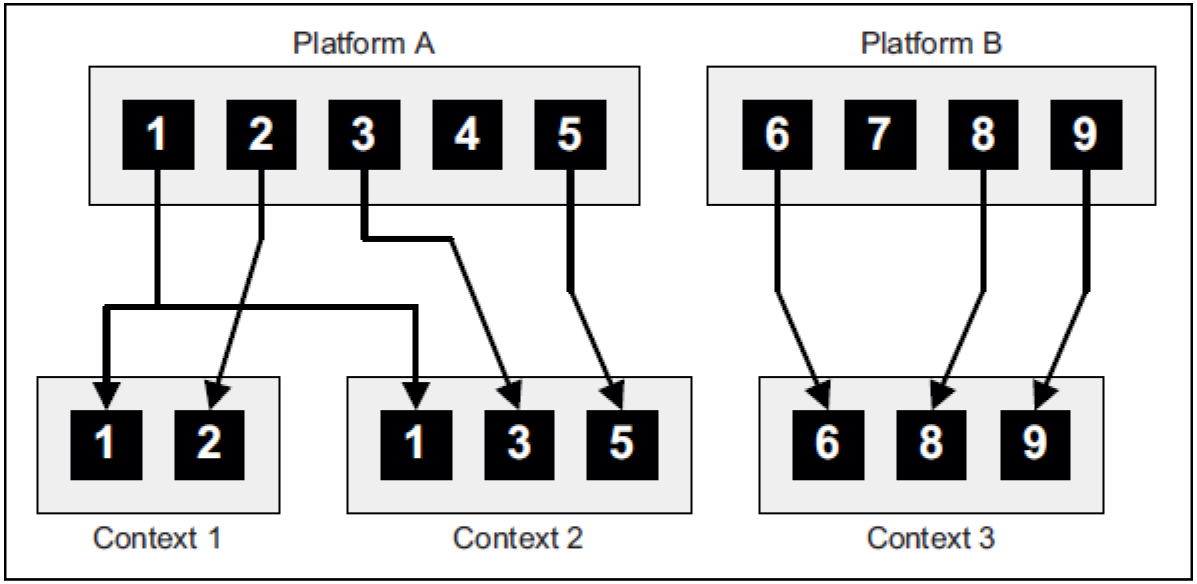
Command queue to send command to devices

cl_command_queue

OpenCL Framework
Platform layer

Create Context

- The devices in a context must be provided by the same platform.

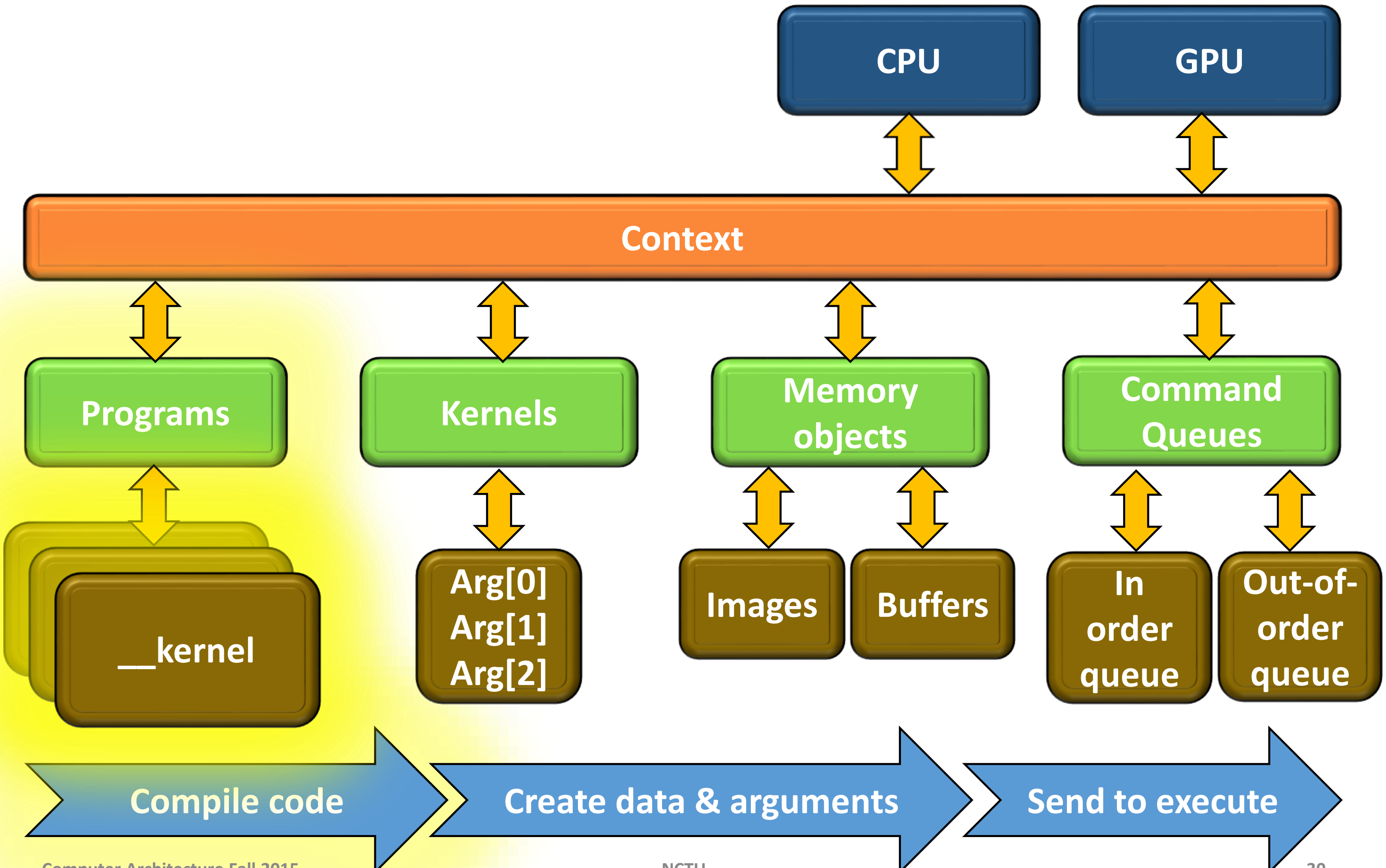


OpenCL Framework
Platform layer

Simple Example: Context Creation

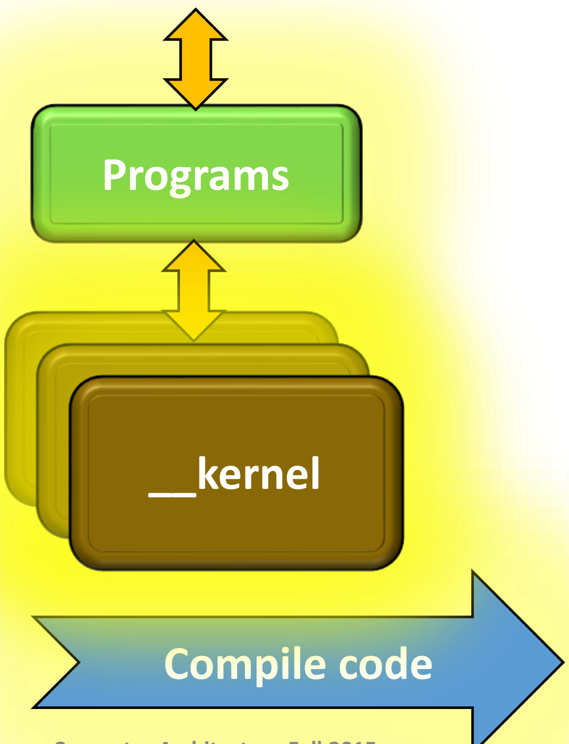
```
// Get the platform ID
cl_platform_id platforms;
clGetPlatformIDs(1, &platforms, NULL);
// Get the first GPU device associated with the platform.
cl_device_id device;
clGetDeviceIDs(platforms, CL_DEVICE_TYPE_ALL, 1, &device, NULL);
// Create an OpenCL context for GPU device
cl_context context;
context=clCreateContext(NULL, 1, devices, NULL, NULL, NULL);
```

OpenCL Framework
Compile OpenCL C



OpenCL Framework
Compile OpenCL C

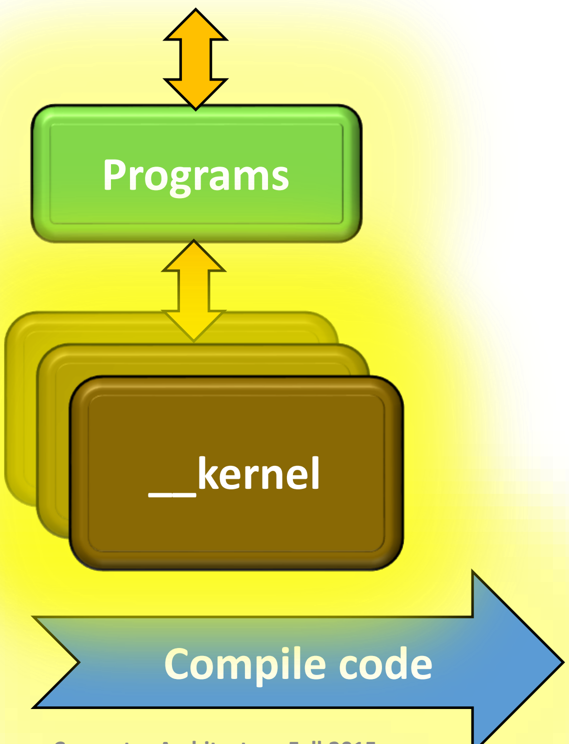
OpenCL C



- **Derived from ISO C99 (with some restrictions)**
- **Language Features Added**
  - Work-items and work-groups
  - Vector types
  - Synchronization
  - Address space qualifiers
- **Includes some built-in functions**
  - Image manipulation
  - Work-item manipulation
  - Math functions

OpenCL Framework
Compile OpenCL C

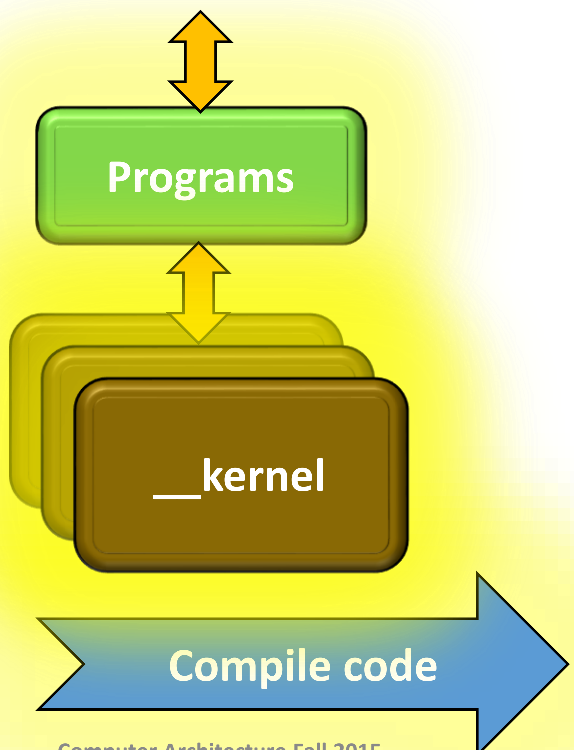
OpenCL C Optional Extensions



- **Extensions are optional features exposed through OpenCL**
  - Double precision
  - Atomic functions
  - Byte-addressable stores
  - Print functions

OpenCL Framework
Compile OpenCL C

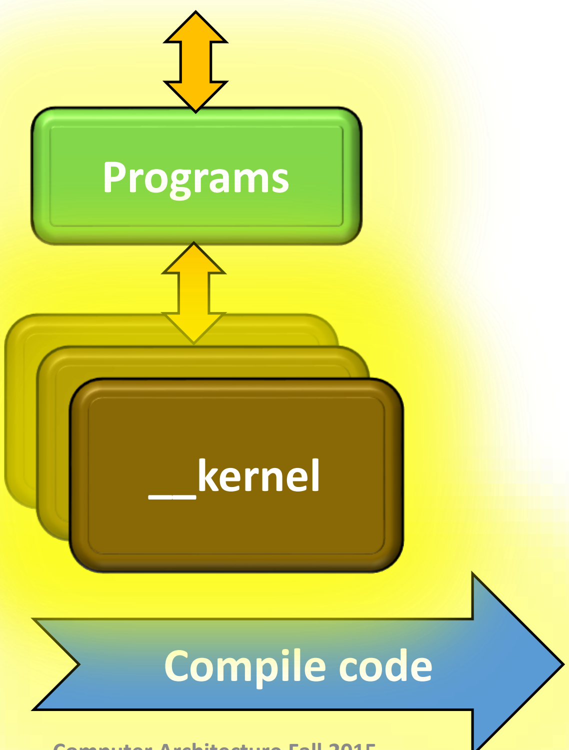
Language Features



- Work-items and work-groups
- Vector types
- Address space qualifiers
- Synchronization

OpenCL Framework
Compile OpenCL C

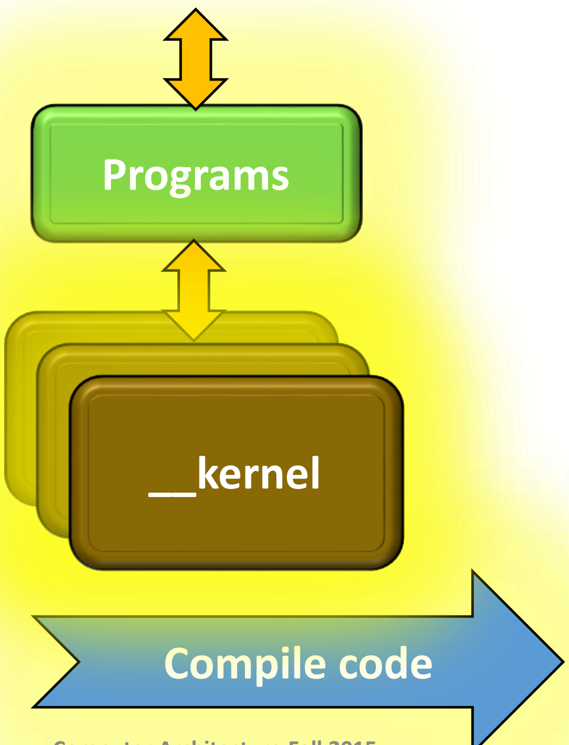
OpenCL C Language Restrictions



- Pointers to functions are not allowed
- Pointers to pointers allowed within a kernel, but not as an argument
- Bit-fields are not supported
- Variable-length arrays and structures are not supported
- Recursion is not supported
- Double types are not supported, but reserved

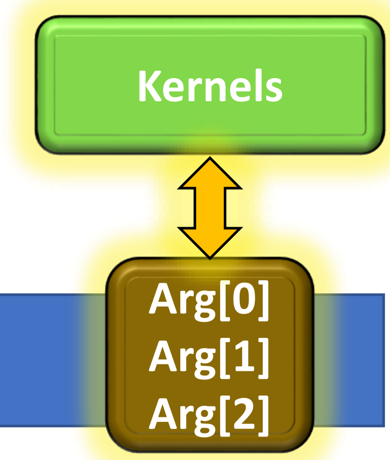
OpenCL Framework
Compile OpenCL C

OpenCL Program Structure



- OpenCL C code is called a program *(cl_program).*
- A program is a collection of functions called kernels *(cl_kernel)*.
- Programs are compiled at runtime through a series of API calls.
- This runtime compilation gives the system an opportunity to optimize for a specific device.

OpenCL Framework
Compile OpenCL C



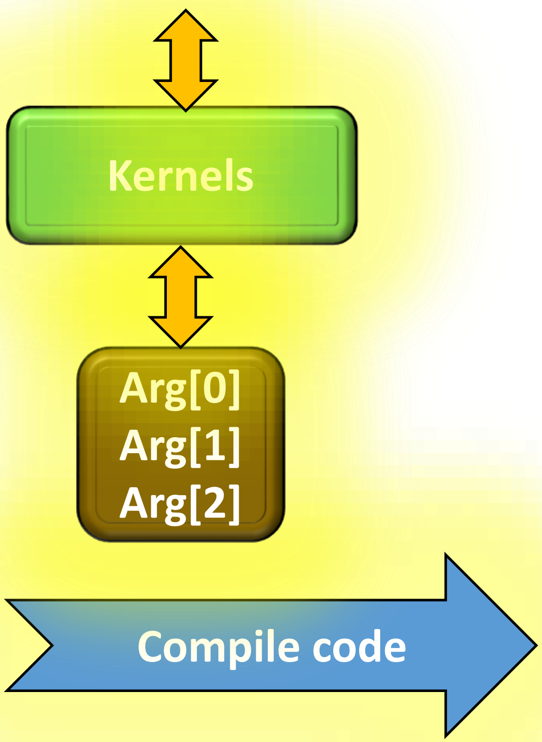
Device Code Compilation And Execution

- **A *cl_program* object encapsulates some source code (kernel functions) and its last successful build.**
  - *clCreateProgramWithSource()* // Create program from source
  - *clBuildProgram()* // Compile program

- **A *cl_kernel* object encapsulates the values of the kernel's arguments used when the kernel is executed**
  - *clCreateKernel()* // Create kernel from successfully compiled program
  - *clSetKernelArg()* // Set values of kernel's arguments

OpenCL Framework
Compile OpenCL C

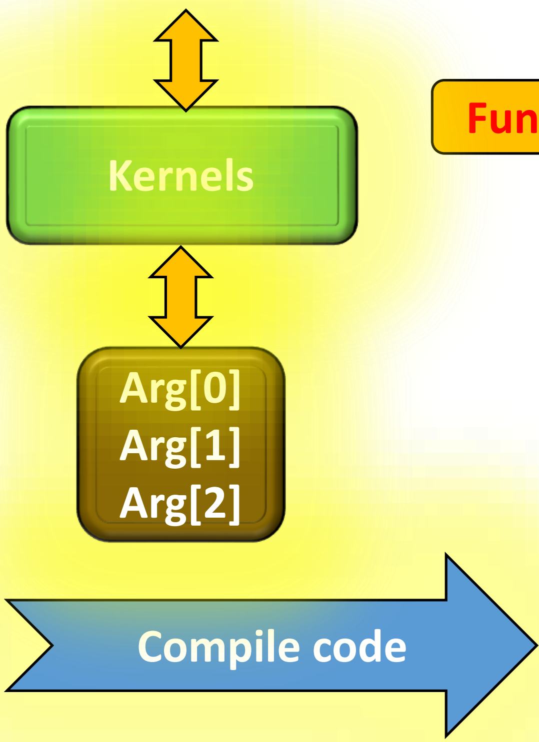
The Process of Creating a Kernel



1. The OpenCL C source code is stored in a character string.
2. The source code is turned into a program object, *cl_program*, by calling *clCreateProgramWithSource().*
3. The program object is then compiled with *clBuildProgram().* If there are compile errors, they will be reported here.

OpenCL Framework
Compile OpenCL C

Simple Example: Language Features



- OpenCL C code

```
__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”
```

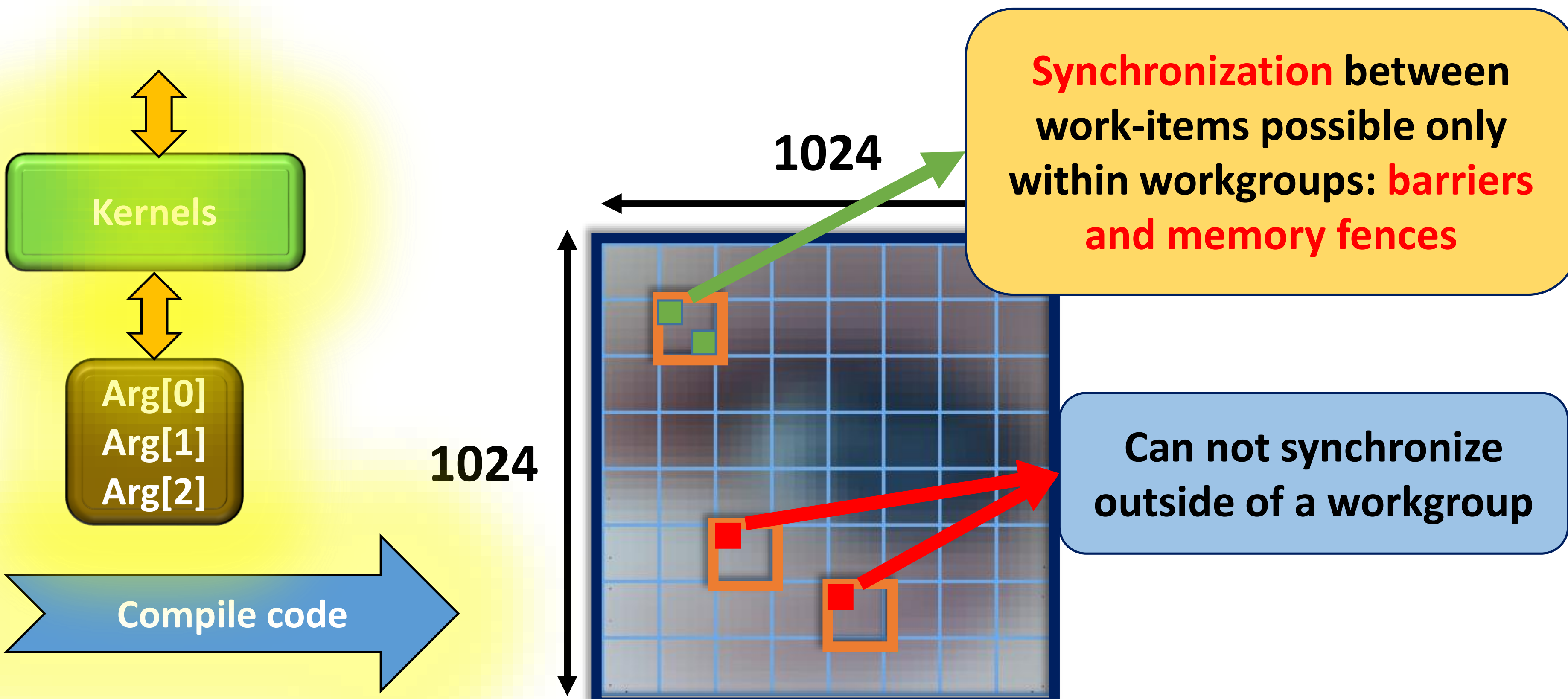
Function qualifier

Address space qualifiers

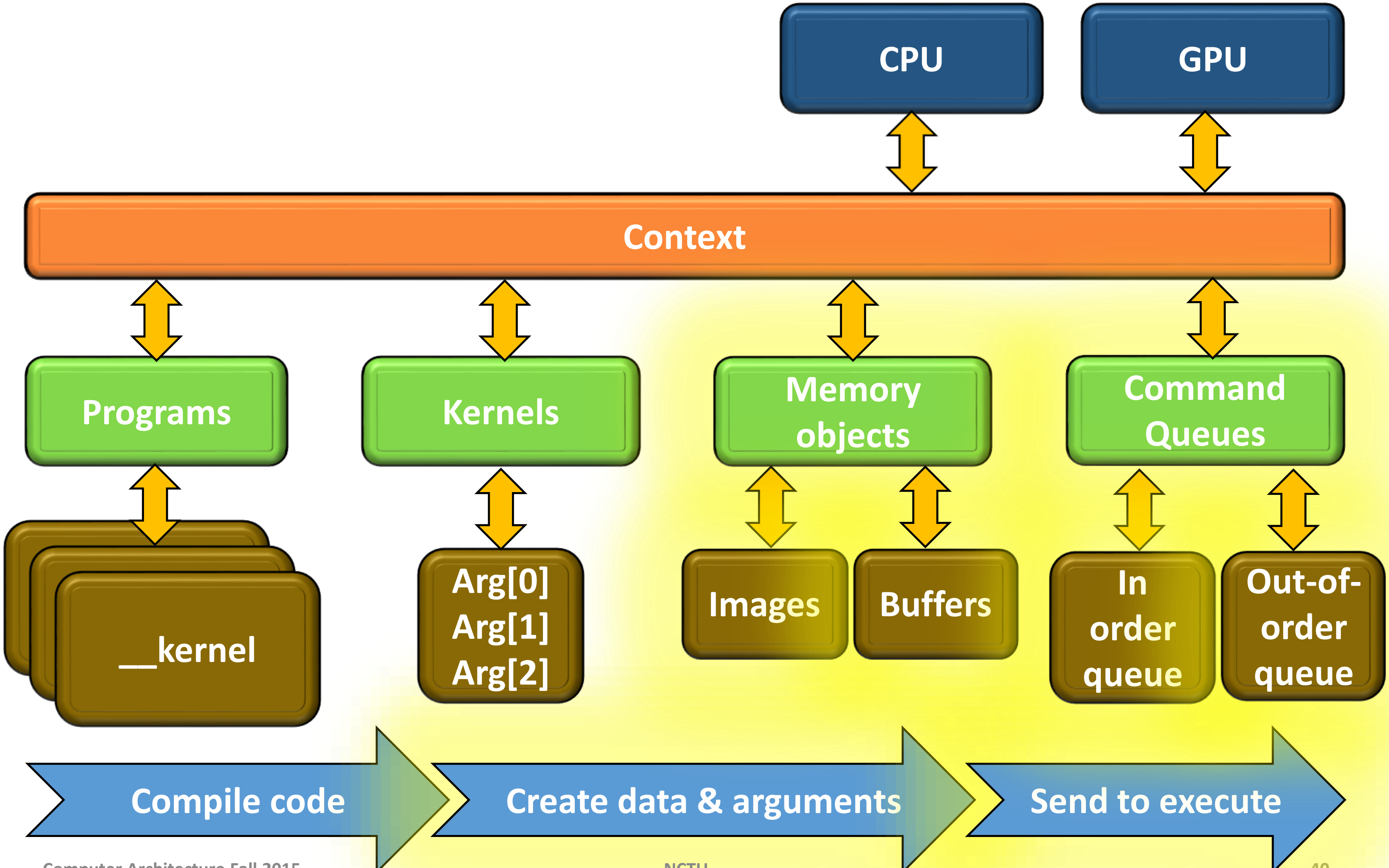
Build-in function
Work items

OpenCL Framework
Compile OpenCL C

Simple Example: Language Features

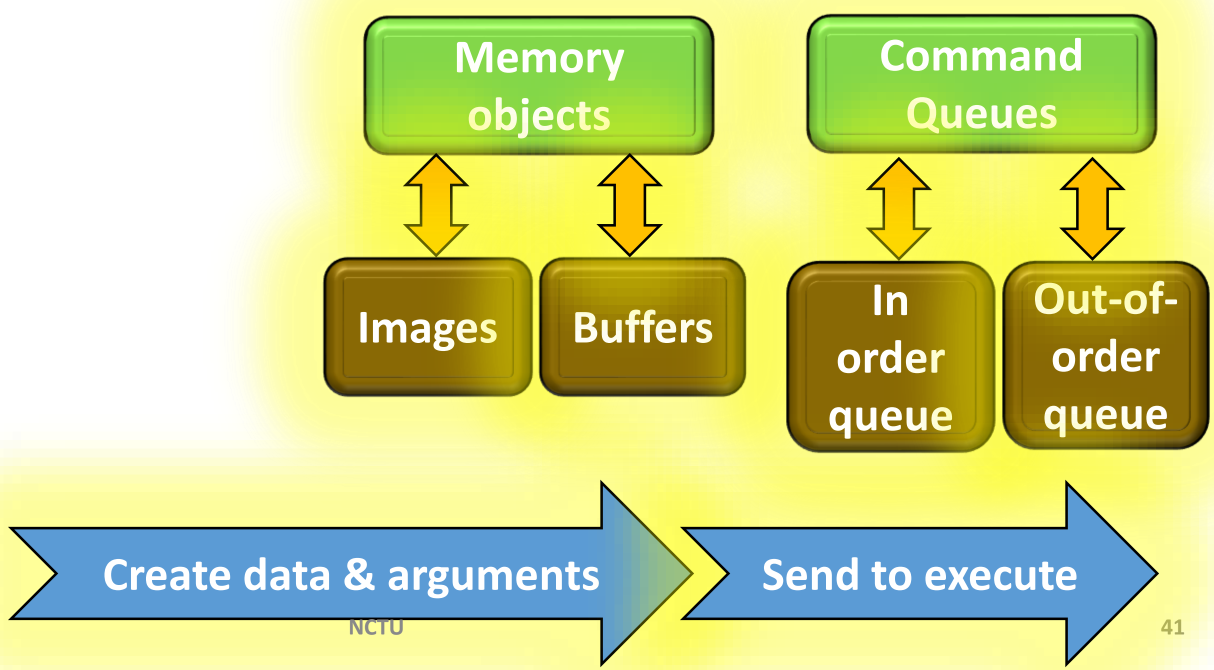


OpenCL Framework
Runtime layer



OpenCL Framework
Runtime layer

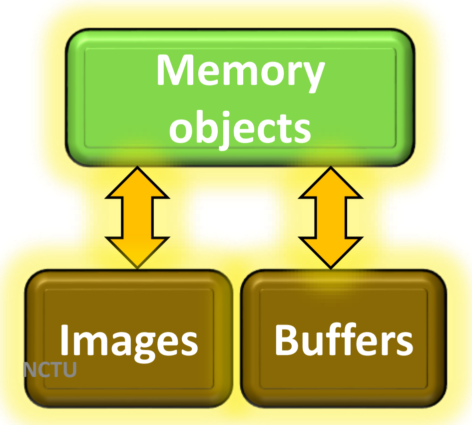
- Device memory allocation and management
- Command queues creation and management
- Device code compilation and execution
- Event creation and management (synchronization, profiling)



OpenCL Framework
Runtime layer

Device Memory Allocation And Management

- When the device performs a task, you have to provide at least three pieces of information:
    1. The instructions to be executed,
    2. A buffer containing data to be processed
    3. A buffer where processed data should be stored
- In OpenCL, memory objects serve as standard packages for transfer data between a host and device.

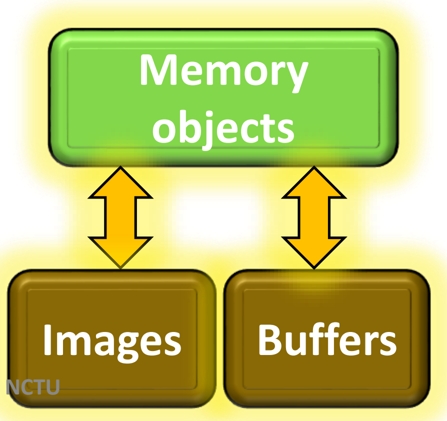


OpenCL Framework
Runtime layer

Device Memory Allocation And Management

- **OpenCL defines two types of memory objects(cl_mem):**
  - **Buffer objects:**
    One-dimensional array
    clCreateBuffer(cl_context context,
    cl_mem_flags options,
    size_t size,
    *void *host_ptr,*
    *cl_int *error)*

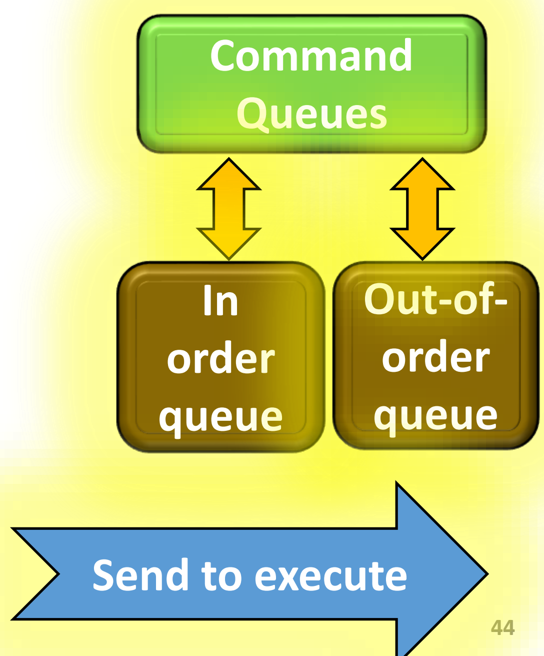
    CL_MEM_READ_WRITE
    CL_MEM_WRITE_ONLY
    CL_MEM_READ_ONLY
    CL_MEM_USE_HOST_PTR
    CL_MEM_COPY_HOST_PTR
    CL_MEM_ALLOC_HOST_PTR
  - **Image objects:**
    clCreateImage2D()
    clCreateImage3D()



OpenCL Framework
Runtime layer

Command Queue Creation

- **Command queue is mechanism of communication between host and devices.**
- **One command queue needs to be created per device.**
- **Two modes of execution:**
  - In-order
  - Out-of-order



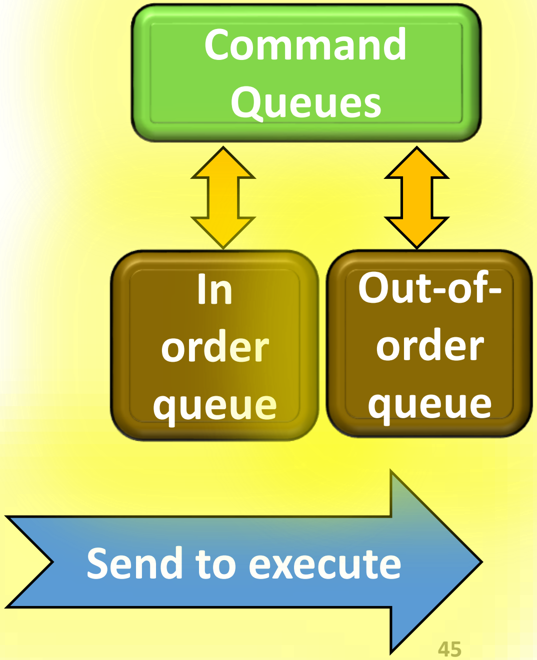
- ***clCreateCommandQueue()***

OpenCL Framework
Runtime layer

How to Send Task to Devices

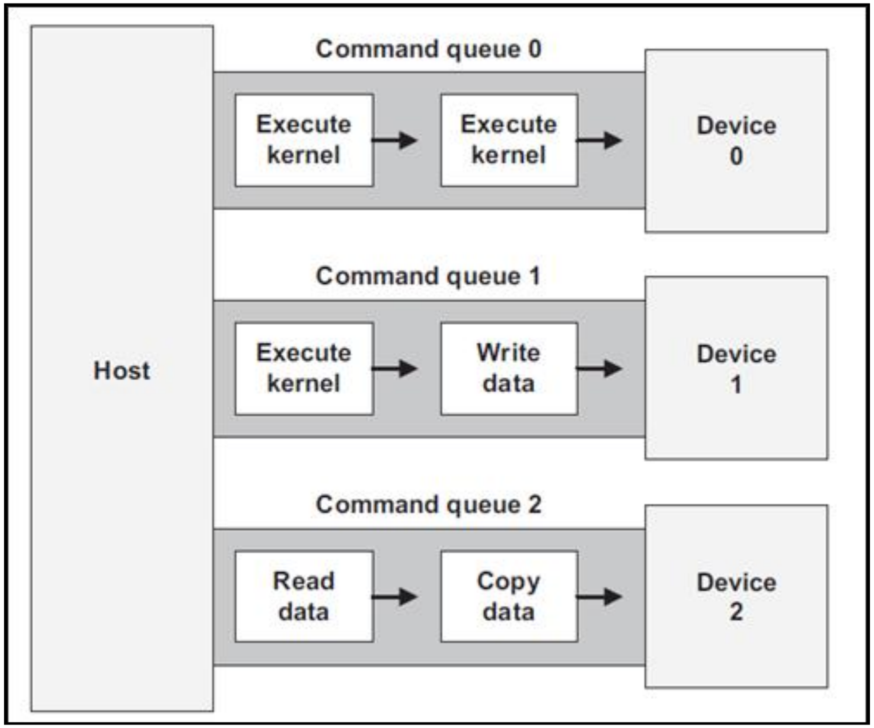
- OpenCL provides many functions that start with "*clEnqueue**", and each of them dispatched a command to a device through a command queue.

- Function:
  clEnqueueTask
  clEnqueueReadBuffer
  clEnqueueWriteBuffer
  clEnqueueNDrangeKernel
  …



OpenCL Framework
Runtime layer

Command Queue Creation



OpenCL Framework
Runtime layer

How to Send Task to Devices

```
cl_int clEnqueueNDRangeKernel (cl_command_queue command_queue,
                               cl_kernel kernel,
                               cl_uint work_dim,
                               const size_t *global_work_offset,
                               const size_t *global_work_size,
                               const size_t *local_work_size,
                               cl_uint num_events_in_wait_list,
                               const cl_event *event_wait_list,
                               cl_event *event)
```

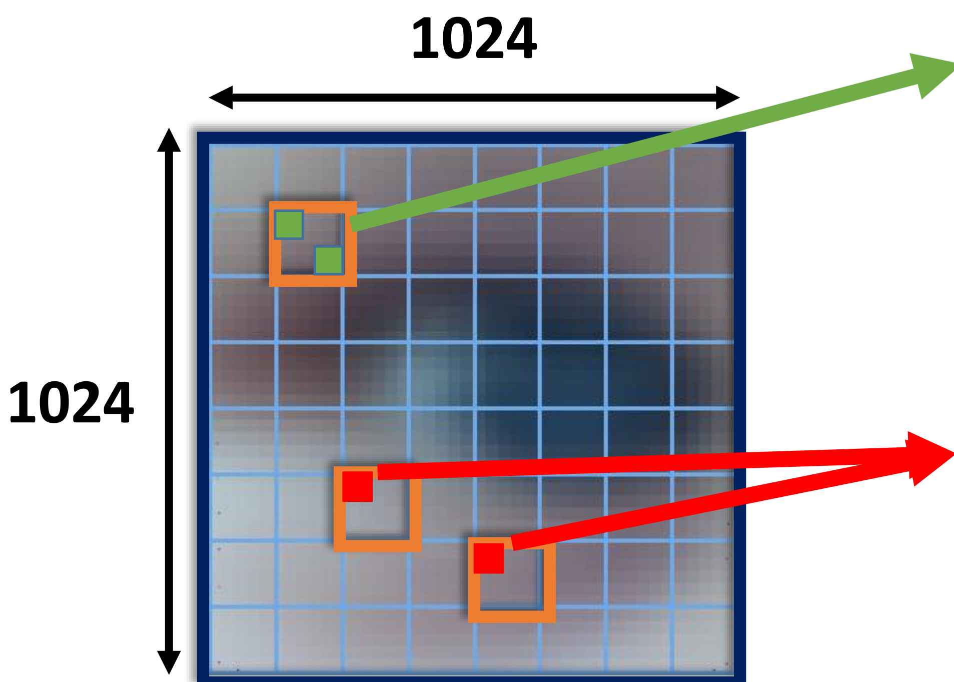
NDRange can be 1-, 2-, 3-dimemsions

- **Global_work_size**
  Define the number of global work-items in work_dim dimensions
- **Local_work_size**
  Define the number of work-items in work_dim dimensions make up a work-group

OpenCL Framework
Runtime layer

Example: 2-dimensions

- **Global dimensions: 1024 x 1024**
- **Local dimensions: 128 x 128**



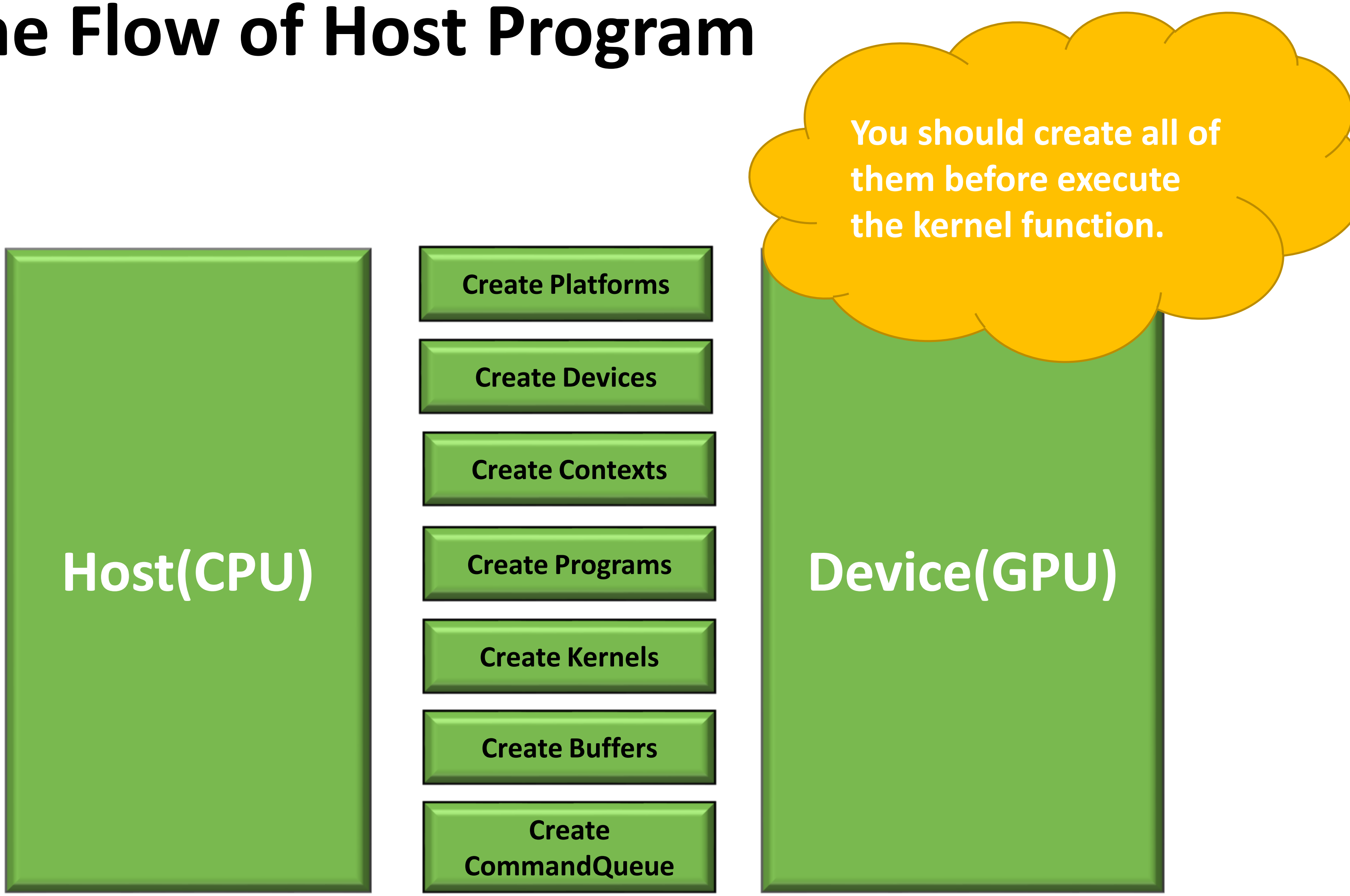
Synchronization between work-items possible only within workgroups: barriers and memory fences

Can not synchronize outside of a workgroup

Outline

- **Introduction to OpenCL**
- **OpenCL Framework**
  - Platform layer
  - Compiler for OpenCL C
  - Runtime
- **The Flow of Host Program**
  - Example: Hello world

The Flow of Host Program

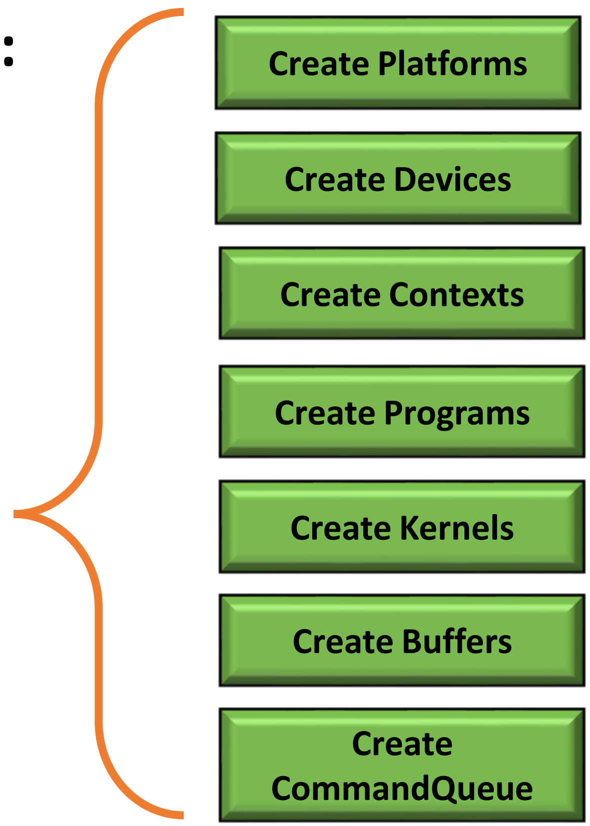


First program

Hello World

- **Preparing two source code:**
  - Hello_world.c
  - Hello_world.cl

- **Program flow**
  1. Read Hello_world.cl file
  2. Create OpenCL structure
  3. Send data to GPU
  4. Send the task to GPU
  5. Get the result from GPU
  6. Done!



1. Read Hello_world.cl file

```
/* Create program from a file */
char *read_cl_file(const char* filename){
  FILE *program_handle;
  char *program_buffer;
  size_t program_size;

  /* Read program file and place content into buffer */
  program_handle = fopen(filename, "r");
  if(program_handle == NULL) {
    perror("Couldn't find the program file");
    exit(1);
  }
  fseek(program_handle, 0, SEEK_END);
  program_size = ftell(program_handle);
  rewind(program_handle);
  program_buffer = (char*)malloc(program_size + 1);
  program_buffer[program_size] = '\0';
  fread(program_buffer, sizeof(char), program_size, program_handle);
  fclose(program_handle);

  return program_buffer;
}
```

2. Create OpenCL structure (1/2)

```
int main() {

  /* OpenCL data structures */
  cl_platform_id platform;
  cl_device_id device;
  cl_context context;
  cl_command_queue queue;
  cl_program program;
  cl_kernel kernel;
  cl_int err;

  /* Data and buffers */
  char msg[16];
  cl_mem msg_buffer;
```

2. Create OpenCL structure (2/2)

```
/* Create a platform*/
err = clGetPlatformIDs(1, &platform, NULL);
 /* Create a device */
err = clGetDeviceIDs(platform, CL_DEVICE_TYPE_ALL, 1, &device, NULL);
 /* Create a context */
context = clCreateContext(NULL, 1, &device, NULL, NULL, &err);
/* Create a program */
char *programSource;
programSource = read_cl_file("Hello_world.cl");
program = clCreateProgramWithSource(context, 1, (const char**)&programSource, NULL, &err);
/* Compile a program */
clBuildProgram(program, 1, &device, NULL, NULL, NULL);
/* create a kernel */
kernel = clCreateKernel(program, "hello_kernel", &err);
```

- Create Platforms
- Create Devices
- Create Contexts
- Create Programs
- Create Kernels
- Create Buffers
- Create CommandQueue

3. Send data to GPU

```
/* Create a buffer to hold the output data */
msg_buffer = clCreateBuffer(context, CL_MEM_WRITE_ONLY, sizeof(msg), NULL, &err);
/* Create kernel argument */
err = clSetKernelArg(kernel, 0, sizeof(cl_mem), &msg_buffer);
/* Create a command queue */
queue = clCreateCommandQueue(context, device, 0, &err);

…
…
…
```

- Create Platforms
- Create Devices
- Create Contexts
- Create Programs
- Create Kernels
- **Create Buffers**
- **Create CommandQueue**

4. Send the task to GPU
5. Get the result from GPU
6. Done! (Release structure)

```c
/* Enqueue kernel */
  size_t globalWorkSize[1];
  globalWorkSize[0]=2;
  err = clEnqueueNDRangeKernel(queue, kernel, 1, NULL, globalWorkSize, NULL, 0, NULL, NULL);
  /* Read and print the result */
  err = clEnqueueReadBuffer(queue, msg_buffer, CL_TRUE, 0, sizeof(msg), &msg, 0, NULL, NULL);
  printf("Kernel output: %s\n", msg);
  /* Deallocate resources */
  clReleaseMemObject(msg_buffer);
  clReleaseKernel(kernel);
  clReleaseCommandQueue(queue);
  clReleaseProgram(program);
  clReleaseContext(context);
  return 0;
}
```

Hello_world.cl

```
__kernel
void hello_kernel(__global char *msg)
{
    *msg = (char)('H', 'e', 'l', 'l', 'o', ' ', 'k', 'e', 'r', 'n', 'e', 'l', '!', '!', '!', '\0');
    printf("Thread[%d]: Hello, welcome come to Opencl world.\n", get_global_id(0));
}
```

Next lecture will discuss

- OpenCL kernel code
- Memory model
- Synchronization

References

- Introduction to OpenCL.
  Cliff Woolley, NVIDIA
  http://www.cc.gatech.edu/~vetter/keeneland/tutorial-2011-04-14/06-intro_to_opencl.pdf
- CUDA C Programming Guide
  http://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#axzz3g8riFfwE
- CUDA Overview.
  Cliff Woolley, NVIDIA
  http://www.cc.gatech.edu/~vetter/keeneland/tutorial-2011-04-14/02-cuda-overview.pdf
- Introduction to CUDA
  James Balfour, NVIDIA Research
  http://mc.stanford.edu/cgi-bin/images/f/f7/Darve_cme343_cuda_1.pdf
- The OpenCL Programming Book - Free HTML version
  Publisher: Fixstars
  Author: Ryoji Tsuchiyama, Takashi Nakamura, Takuro Iizuka, Akihiro Asahara, Jeongdo Son, Satoshi Miki
- OpenCL in action
  Author: Matthew Scarpino