CPSC/ECE 4780/6780

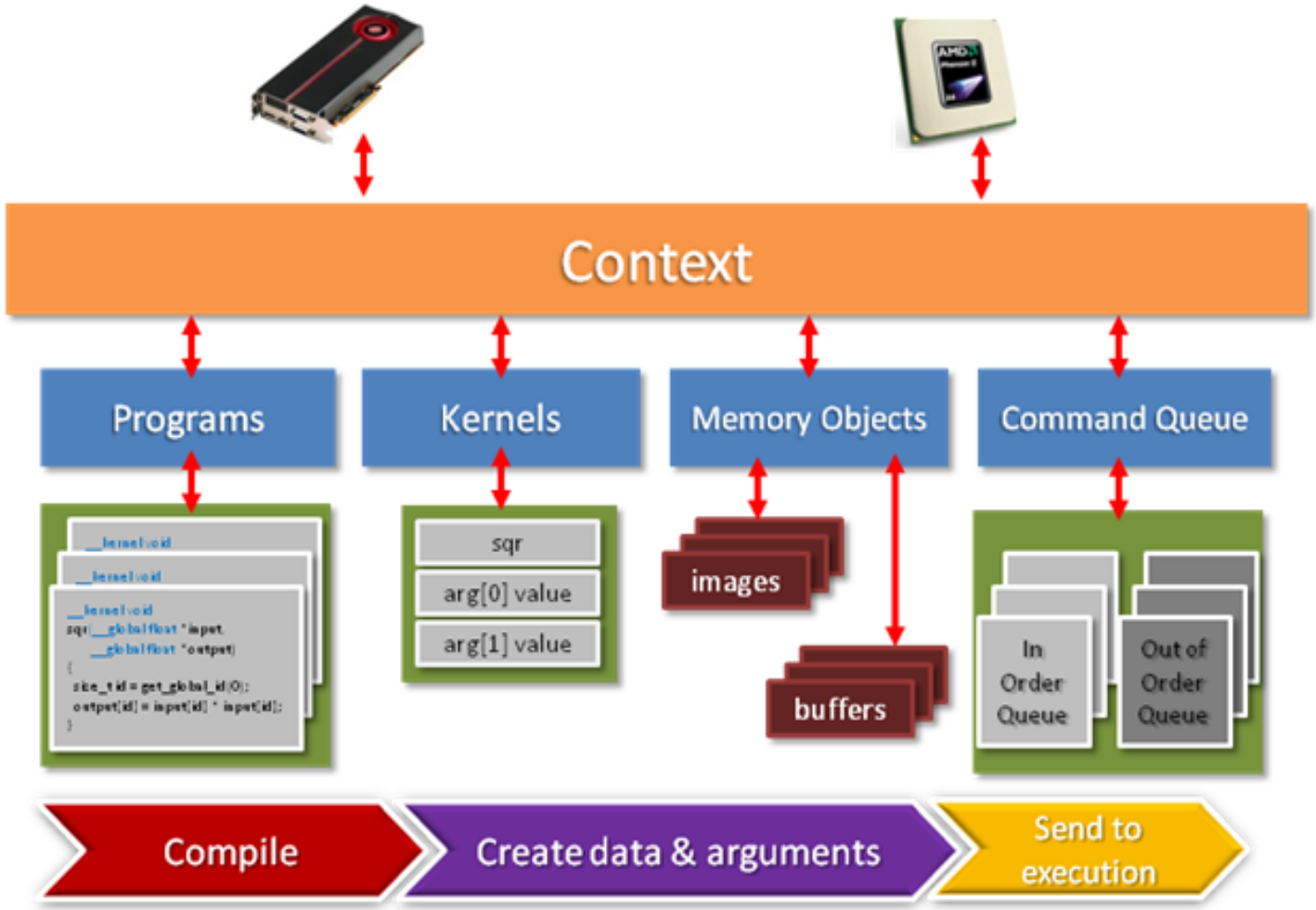
General-Purpose Computation on Graphical Processing Units (GPGPU)

Lecture 14: (OpenCL) Programming Details

Recap of Last Lecture

- What is OpenCL? Why do we use OpenCL?
- What is the Anatomy of OpenCL?
- What does OpenCL architecture look like?
- What functions are involved in OpenCL program flow?

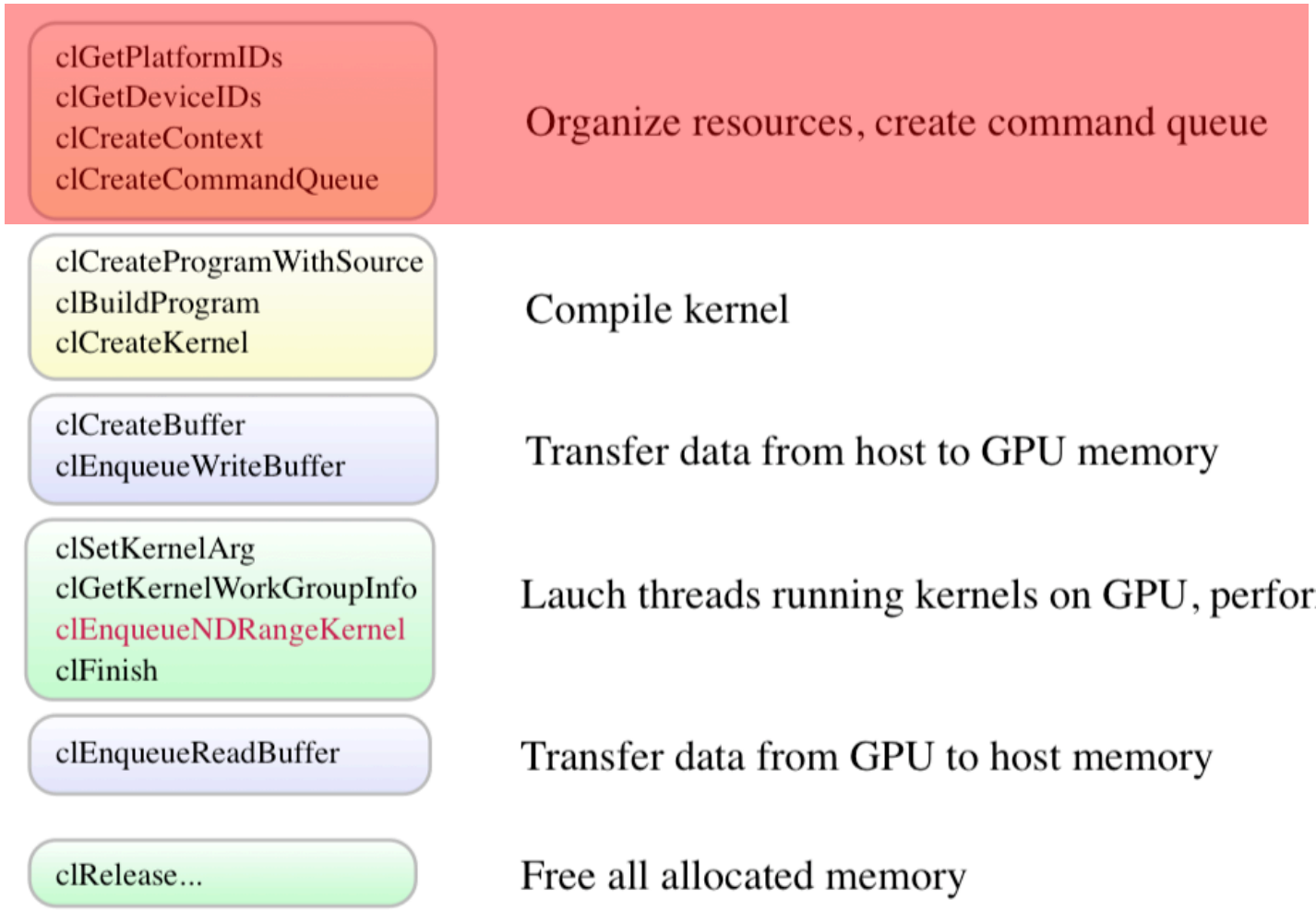
Structure of an OpenCL Program



To execute an OpenCL program:

1. Query the host system for OpenCL devices
2. Create a context to associate the OpenCL devices
3. Create programs that will run on one or more associated devices
4. From the programs, select kernels to execute
5. Create memory objects on the host or on the device
6. Copy memory data to the device as needed
7. Provide arguments for the kernels.
8. Submit the kernels to the command queue for execution
9. Copy the results from the device to the host

OpenCL Program Flow (Resource Setup)



Query for Platform Information

```
cl_platform_id platforms;
cl_uint num_platforms;
// query for 1 available platform
cl_int err = clGetPlatfromIDs(
        1,                  // the number of entries that can be added to platforms
        &platforms,         // list of OpenCL platform found
        &num_platforms);    // the number of OpenCL platforms found
```

`clGetPlatformIDs` return values are:

- **CL_INVALID_VALUE** — Platforms and num_platforms is NULL or the number of entries is 0.
- **CL_SUCCESS** — The function executed successfully.

Query for Platform Information (Cont.)

- Get specific information about the OpenCL platform by clGetPlatformInfo()

```
for(i=0;i<num_platforms;i++) {
   char buffer[10240];
   clGetPlatformInfo(clPlatformIDs[i], CL_PLATFORM_PROFILE, 10240, buffer, NULL);
   printf("  PROFILE = %s\n", buffer);
   clGetPlatformInfo(clPlatformIDs[i], CL_PLATFORM_VERSION, 10240, buffer, NULL);
   printf("  VERSION = %s\n", buffer);
   clGetPlatformInfo(clPlatformIDs[i], CL_PLATFORM_NAME, 10240, buffer, NULL);
   printf("  NAME = %s\n", buffer);
   clGetPlatformInfo(clPlatformIDs[i], CL_PLATFORM_VENDOR, 10240, buffer, NULL);
   printf("  VENDOR = %s\n", buffer);
   clGetPlatformInfo(clPlatformIDs[i], CL_PLATFORM_EXTENSIONS, 10240, buffer, NULL);
   printf("  EXTENSIONS = %s\n", buffer);
}
```

Output:

```
Number of platforms: 1
  PROFILE = FULL_PROFILE
  VERSION = OpenCL 1.2 CUDA 8.0.0
  NAME = NVIDIA CUDA
  VENDOR = NVIDIA Corporation
  EXTENSIONS = cl_khr_global_int32_base_atomics cl_khr_global_int32_extended_atomics cl_kh
r_local_int32_base_atomics cl_khr_local_int32_extended_atomics cl_khr_fp64 cl_khr_byte_add
ressable_store cl_khr_icd cl_khr_gl_sharing cl_nv_compiler_options cl_nv_device_attribute_
query cl_nv_pragma_unroll cl_nv_copy_opts cl_khr_gl_event cl_nv_create_buffer
```

Query for OpenCL Device

```
cl_device_id device_id;
cl_uint num_of_devices;
cl_int err;
err = clGetDeviceIDs(
    platform_id,          // the platform_id retrieved from clGetPlatformIDs
    CL_DEVICE_TYPE_GPU,   // the device type to search for
    1,                    // the number of id add to device_id list
    &device_id,           // the list of device ids
    &num_of_devices);     // the number of compute devices found
```

OpenCL device types:

- CL_DEVICE_TYPE_CPU
- CL_DEVICE_TYPE_GPU
- CL_DEVICE_TYPE_ACCELERATOR
- CL_DEVICE_TYPE_DEFAULT
- CL_DEVICE_TYPE_ALL

`clGetDeviceIDs()` return values are:

- **CL_INVALID_PLATFORM** — Platform is not valid.
- **CL_INVALID_DEVICE_TYPE** — The device is not a valid value.
- **CL_INVALID_VALUE** — `num_of_devices` and devices are NULL.
- **CL_DEVICE_NOT_FOUND** — No matching OpenCL of `device_type` was found.
- **CL_SUCCESS** — The function executed successfully.

Query for OpenCL Device (Cont.)

- Get specific capabilities about the OpenCL devices by clGetDeviceInfo()

```
for (int i=0; i<gpudevcount; i++) {
        char buffer[10240];
        cl_uint buf_uint;
        cl_ulong buf_ulong;
        clGetDeviceInfo(mydevice[i], CL_DEVICE_NAME, sizeof(buffer), buffer, NULL);
        printf("  DEVICE_NAME = %s\n", buffer);
        clGetDeviceInfo(mydevice[i], CL_DEVICE_VENDOR, sizeof(buffer), buffer, NULL);
        printf("  DEVICE_VENDOR = %s\n", buffer);
        clGetDeviceInfo(mydevice[i], CL_DEVICE_VERSION, sizeof(buffer), buffer, NULL);
        printf("  DEVICE_VERSION = %s\n", buffer);
        clGetDeviceInfo(mydevice[i], CL_DRIVER_VERSION, sizeof(buffer), buffer, NULL);
        printf("  DRIVER_VERSION = %s\n", buffer);
        clGetDeviceInfo(mydevice[i], CL_DEVICE_MAX_COMPUTE_UNITS, sizeof(buf_uint), &buf_uint, NULL);
        printf("  DEVICE_MAX_COMPUTE_UNITS = %u\n", (unsigned int)buf_uint);
        clGetDeviceInfo(mydevice[i], CL_DEVICE_MAX_CLOCK_FREQUENCY, sizeof(buf_uint), &buf_uint, NULL);
        printf("  DEVICE_MAX_CLOCK_FREQUENCY = %u\n", (unsigned int)buf_uint);
        clGetDeviceInfo(mydevice[i], CL_DEVICE_GLOBAL_MEM_SIZE, sizeof(buf_ulong), &buf_ulong, NULL);
        printf("  DEVICE_GLOBAL_MEM_SIZE = %llu\n", (unsigned long long)buf_ulong);
}
```

Output:

```
DEVICE_NAME = GeForce GTX TITAN Black
DEVICE_VENDOR = NVIDIA Corporation
DEVICE_VERSION = OpenCL 1.2 CUDA
DRIVER_VERSION = 381.22
DEVICE_MAX_COMPUTE_UNITS = 15
DEVICE_MAX_CLOCK_FREQUENCY = 980
DEVICE_GLOBAL_MEM_SIZE = 6376390656
```

Create a Context

```
cl_context context;

// Context properties list (must be terminated with 0)
properties[0] = CL_CONTEXT_PLATFORM;
properties[1] = (cl_context_properties) platform_id;
properties[2] = 0;

// Create a context with the GPU device
context = clCreateContext(
   properties,          // list of context properties
   1,                   // num of devices in the device_id list
   &device_id,          // the device id list
   NULL,                // pointer to the error callback function (if required)
   NULL,                // the argument data to pass to the callback function
   &err);               // the return code
```

Return values when the context is not created successfully:

- **CL_INVALID_PLATFORM** — The property list is NULL or the platform value is not valid.
- **CL_INVALID_VALUE** — Either:
  - The property name in the properties list is not valid.
  - The number of devices is 0.
  - The `device_id` list is null.
  - The device in the `device_id` list is invalid or not associated with the platform.
- **CL_DEVICE_NOT_AVAILABLE** — The device in the `device_id` list is currently unavailable.
- **CL_OUT_OF_HOST_MEMORY** — The host is unable to allocate OpenCL resources.

Create the Command Queue

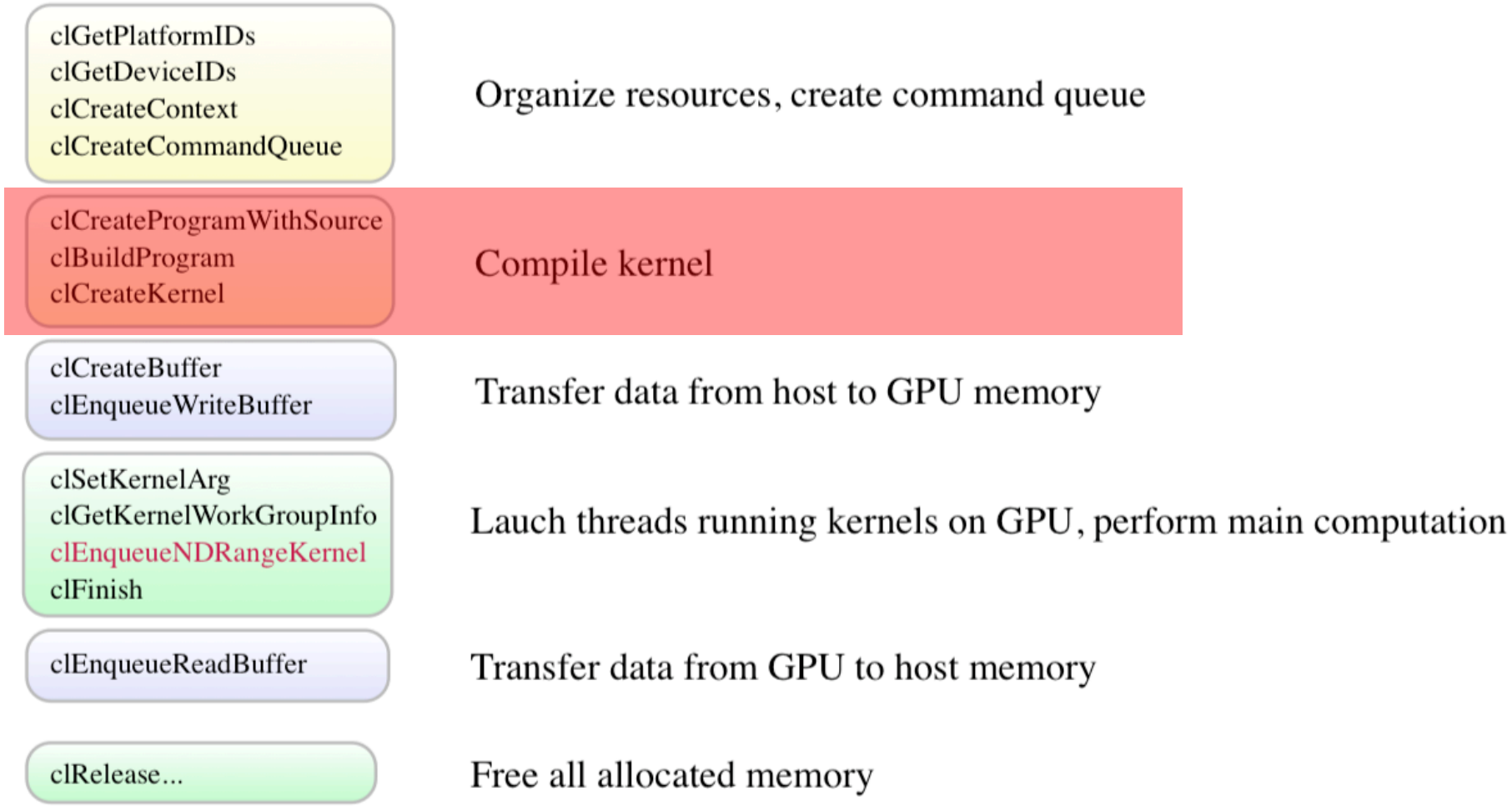
```
cl_command_queue command_queue;

// Create a command queue using the context and device
command_queue = clCreateCommandQueue(
  context,              // a valid context
  device_id,            // a valid device associated with the context
  0,                    // properties for the queue - not used here
  &err);                // the return code
```

Return values when the command queue is not created successfully:

- **CL_INVALID_CONTEXT** — The context is not valid.
- **CL_INVALID_DEVICE** — Either the device is not valid or it is not associated with the context.
- **CL_INVALID_VALUE** — The properties list is not valid.
- **CL_INVALID_QUEUE_PROPERTIES** — The device does not support the properties specified in the properties list.
- **CL_OUT_OF_HOST_MEMORY** — The host is unable to allocate OpenCL resources.

OpenCL Program Flow (Kernel Programming and Compiling)



Create the Program Object

```
const char *kernelSource =
"__kernel void hello(__global float *input, __global float *output)\n"\
"{\n"\
"  size_t id = get_global_id(0);\n"\
"  output[id] = input[id] * input[id];\n"\
"}\n"\
"\n";

cl_program program;

// Create a program from the kernel source code
program = clCreateProgramWithSource(
   context,                              // a valid conterxt
   1,                                    // the number of strings in the next parameter
   (const char **) &kernelSource,        // the array of strings
   NULL,                                 // the length of each string or can be NULL terminated
   &err);                                // the error return code
```

Return values when the program object is not created successfully:

- **CL_INVALID_CONTEXT** — The context is not valid.
- **CL_INVALID_VALUE** — The string count is 0 (zero) or the string array contains a NULL string.
- **CL_OUT_OF_HOST_MEMORY** — The host is unable to allocate OpenCL resources.

Build Program Executables

```
// Compile the program
err = clBuildProgram(
      program,     // a valid program object
      0,           // number of devices in the device list
      NULL,        // device list -- NULL means for all devices
      NULL,        // a string of build options
      NULL,        // callback function when executable has been built
      NULL);       // data arguments for the callback function
```

Return values when the compilation is unsuccessfully:

- **CL_INVALID_VALUE** — The number of devices is greater than zero, but the device list is empty.
- **CL_INVALID_VALUE** — The callback function is NULL, but the data argument list is not NULL.
- **CL_INVALID_DEVICE** — The device list does not match the devices associated in the program object.
- **CL_INVALID_BUILD_OPTIONS** — The build options string contains invalid options.
- **CL_OUT_OF_HOST_MEMORY** — The host is unable to allocate OpenCL resources.

Compiling Source Code Error

- Retrieve the latest compilation results embedded in the program object by clGetBuildProgramInfo()

```
// Compile the program
if (clBuildProgram(program, 0, NULL, NULL, NULL, NULL) != CL_SUCCESS) {
   printf("Error building program\n");

   char buffer[4096];
   size_t length;
   clGetProgramBuildInfo(
      program,              // valid program object
      device_id,            // valid device_id that executable was built
      CL_PROGRAM_BUILD_LOG, // indicate to retrieve build log
      sizeof(buffer),       // size of the buffer to write log to
      buffer,               // the actual buffer to write log to
      &length               // the actual size in bytes of data copied to buffer
   );
   printf("%s\n",buffer);
   exit(1);
}
```

Compiling Source Code Error (Cont.)

- For example, we have a buggy kernel source:

```
const char *kernelSource =
"__kernel void hello(__global float *input, __global float *output)\n"\
"{\n"\
"  size_t id = get_global_id(0);\n"\
"  output[id] = input[id] * input[id];\n"\
"  error;\n"\
"}\n"\
"\n";
```

- Sample build log output:

```
Error building program
<kernel>:5:3: error: use of undeclared identifier 'error'
  error;
  ^
```

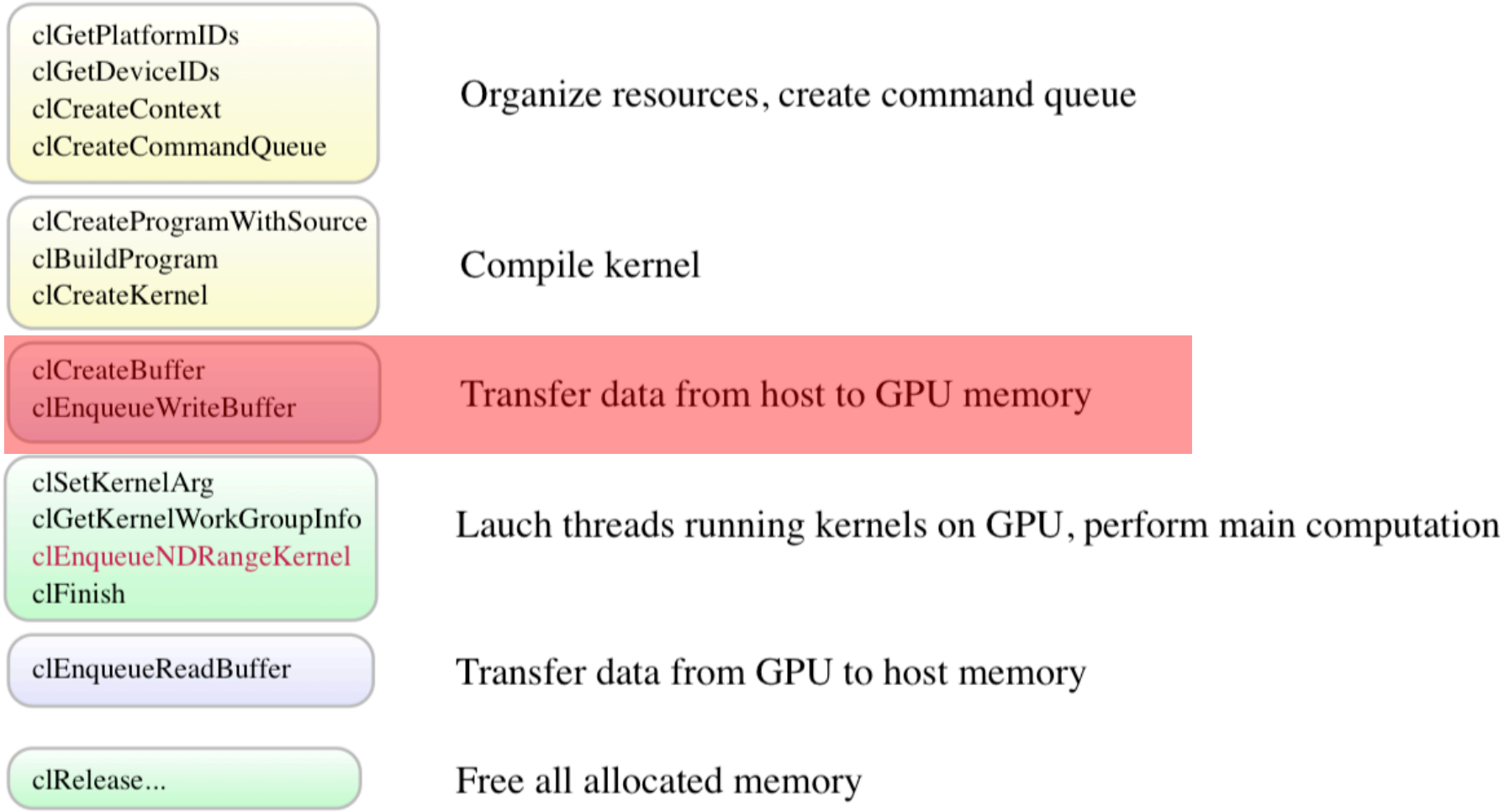
Create Kernel Objects

```
cl_kernel kernel;
kernel = clCreateKernel(
   program,  // a valid program object that has been successfully built
   "hello",  // the name of the kernel declared with __kernel
   &err);    // error return code
```

Return values when the kernel object is not created successfully:

- **CL_INVALID_PROGRAM** — The program is not a valid program object.
- **CL_INVALID_PROGRAM_EXECUTABLE** — The program does not contain a successfully built executable.
- **CL_INVALID_KERNEL_NAME** — The kernel name is not found in the program object.
- **CL_INVALID_VALUE** — The kernel name is NULL.
- **CL_OUT_OF_HOST_MEMORY** — The host is unable to allocate OpenCL resources.

OpenCL Program Flow (Host to GPU Memory Transfer)



Create Buffer Objects

```
cl_mem input;
// Create buffers for the input
input = clCreateBuffer(
     context,                                  // a valid context
     CL_MEM_READ_ONLY | CL_MEM_COPY_HOST_PTR,  // bit-field flag to specify the usage of memory
     sizeof(float) * DATA_SIZE,                // size in bytes of the buffer to allocated
     inputsrc,                                 // pointer to buffer data to be copied from host
     &err);                                    // returned error code
```

Return values when the buffer object is not created successfully:

- **CL_INVALID_CONTEXT** — The context is not valid.
- **CL_INVALID_VALUE** — The value in cl_mem_flag is not valid (see table above for supported flags).
- **CL_INVALID_BUFFER_SIZE** — The buffer size is 0 (zero) or exceeds the range supported by the compute devices associated with the context.
- **CL_INVALID_HOST_PTR** — Either: The host_ptr is NULL, but CL_MEM_USE_HOST_PTR, CL_MEM_COPY_HOST_PTR, and CL_MEM_ALLOC_HOST_PTR are set; or host_ptr is not NULL, but the CL_MEM_USE_HOST_PTR, CL_MEM_COPY_HOST_PTR, and CL_MEM_ALLOC_HOST_PTR are not set.
- **CL_INVALID_OBJECT_ALLOCATION_FAILURE** — Unable to allocate memory for the memory object.
- **CL_OUT_OF_HOST_MEMORY** — The host is unable to allocate OpenCL resources.

CL_MEM Flags	Description
CL_MEM_READ_WRITE	Kernel can read and write to the memory object.
CL_MEM_WRITE_ONLY	Kernel can write to memory object. Read from the memory object is undefined.
CL_MEM_READ_ONLY	Kernel can only read from the memory object. Write from the memory object is undefined.
CL_MEM_USE_HOST_PTR	Specifies to OpenCL implementation to use memory reference by host_ptr (4th arg) as storage for memory object.
CL_MEM_COPY_HOST_PTR	Specifies to OpenCL to allocate the memory and copy data pointed by host_ptr (4th arg) to the memory object.
CL_MEM_ALLOC_HOST_PTR	Specifies to OpenCL to allocate memory from host accessible memory.

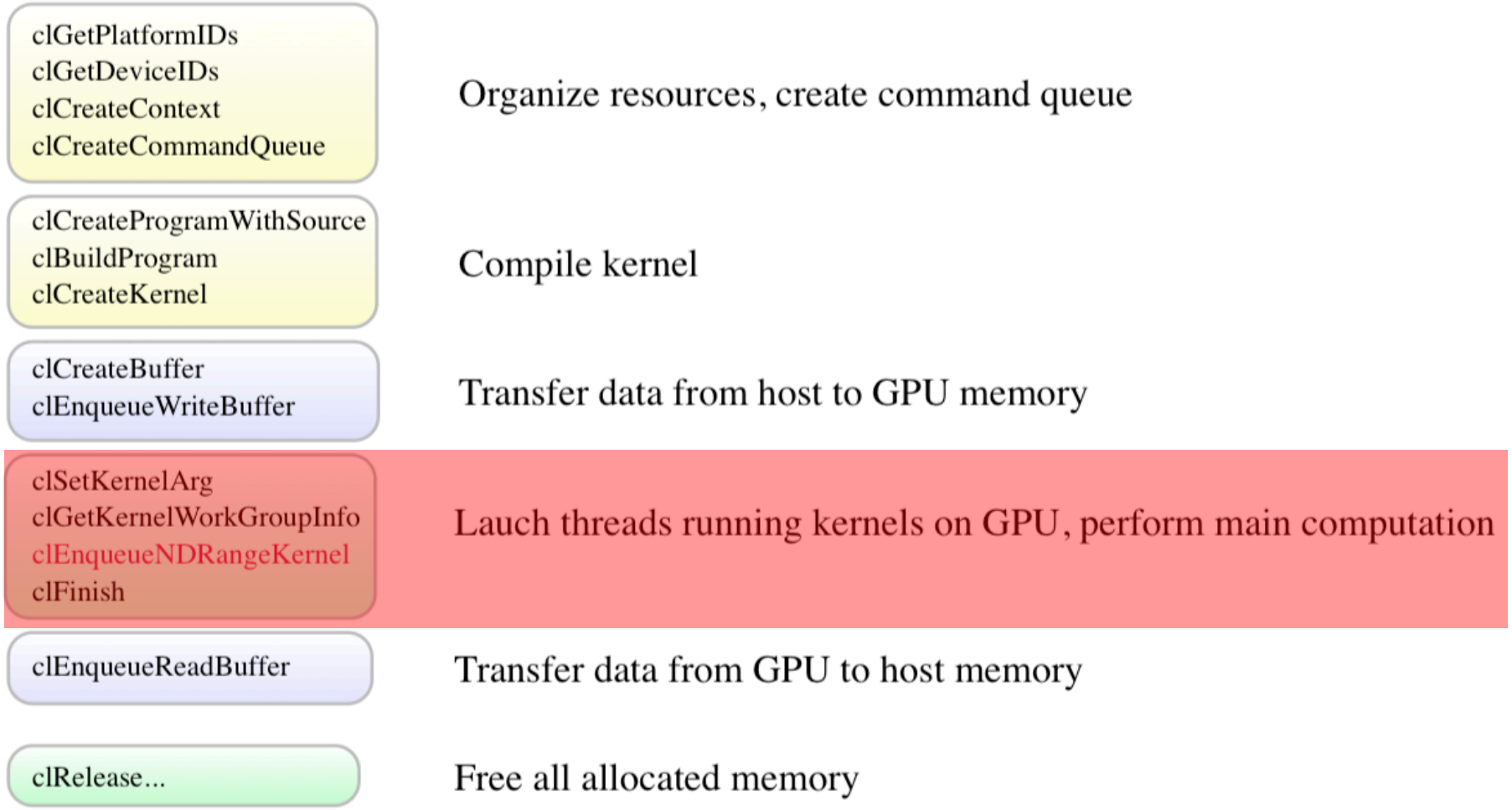
Write Buffer Objects

```
err = clEnqueueWriteBuffer(
    command_queue,              // valid command queue
    input,                      // memory buffer to write to
    CL_TRUE,                    // indicate blocking write
    0,                          // the offset in the buffer object to write to
    sizeof(float) *DATA_SIZE,   // size in bytes of data to write
    host_ptr,                   // pointer to buffer in host mem to read data from
    0,                          // number of event in the event list
    NULL,                       // list of events that needs to complete before this executes
    NULL);                      // event object to return on completion
```

Return values when the function is not executed successfully:

- **CL_INVALID_COMMAND_QUEUE** — The command queue is not valid
- **CL_INVALID_CONTEXT** — The command queue buffer object is not associated with the same context.
- **CL_INVALID_VALUE** — The region being read/write specified by the offset is out of bounds or the host pointer is NULL.
- **CL_INVALID_EVENT_WAIT_LIST** — Either: The events list is empty (NULL), but the number of events argument is greater than 0; or number of events is 0, but the event list is not NULL; or ;the events list contains invalid event objects.
- **CL_OUT_OF_HOST_MEMORY** — The host is unable to allocate OpenCL resources.

OpenCL Program Flow (Kernel Execution)



Set Kernel Arguments

```
err = clSetKernelArg(
      kernel,           // valid kernel object
      0,                // the specific argument index of a kernel
      sizeof(cl_mem),   // the size of the argument data
      &input_data);     // a pointer of data used as the argument
```

Return values when the function is not executed successfully:

- **CL_INVALID_KERNEL** — The kernel is not a valid kernel object.
- **CL_INVALID_ARG_INDEX** — The index value is not a valid argument index.
- **CL_INVALID_MEMORY_OBJECT** — The argument is declared as a memory object, but the argument value is not a valid memory object.
- **CL_INVALID_ARG_SIZE** — The argument size does not match the size of the data type of the declared argument.

For example, we have kernel function:

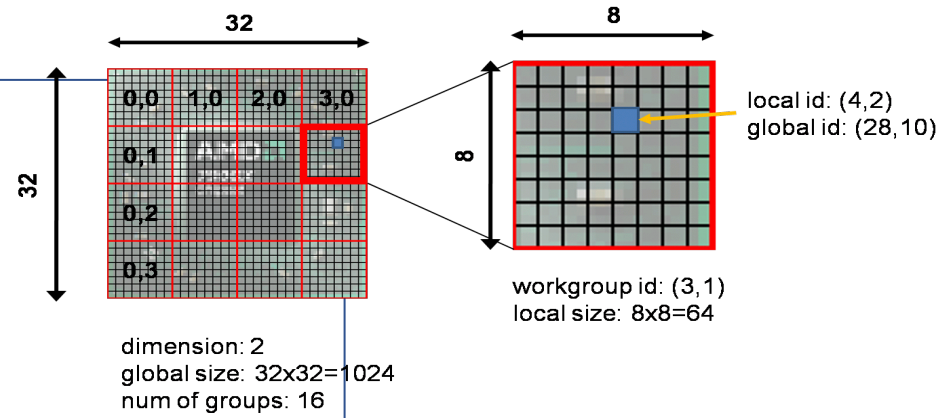
```
'__kernel void hello(__global float *input, __global float *output)
```

To set arguments:

```
// Set the argument list for the kernel command
clSetKernelArg(kernel, 0, sizeof(cl_mem), &input);
clSetKernelArg(kernel, 1, sizeof(cl_mem), &output);
```

Determine the Work Group

```
global = DATA_SIZE;
// Enqueue the kernel command for execution
err = clEnqueueNDRangeKernel(
    command_queue, // valid command queue
    kernel,        // valid kernel object
    1,             // the work problem dimensions
    NULL,          // reserved for future revision - must be NULL
    &global,       // work-items for each dimension
    NULL,          // work-group size for each dimension
    0,             // number of event in the event list
    NULL,          // list of events that needs to complete before this executes
    NULL);         // event object to return on completion
```



- Work-item size for each dimension
  - For example, if an image of 512x512 pixels is to be processed, an array must be provided that points to the number of work-item for each dimension as: size_t global[2]={512,512};
- Work-group size for each dimension
  - For example, if 64 work-items were grouped into an 8x8 work-group, the work-group size for each dimension would be specified as array: size_t local[2]={8,8};
- **clGetKernelWorkGroupInfo**(): returns information about the kernel object that may be specific to a device, such as CL_KERNEL_WORK_GROUP_SIZE, CL_KERNELL_LOCAL_MEM_SIZE, etc.

Determine the Work Group (Cont.)

Return values when the function is not executed successfully:

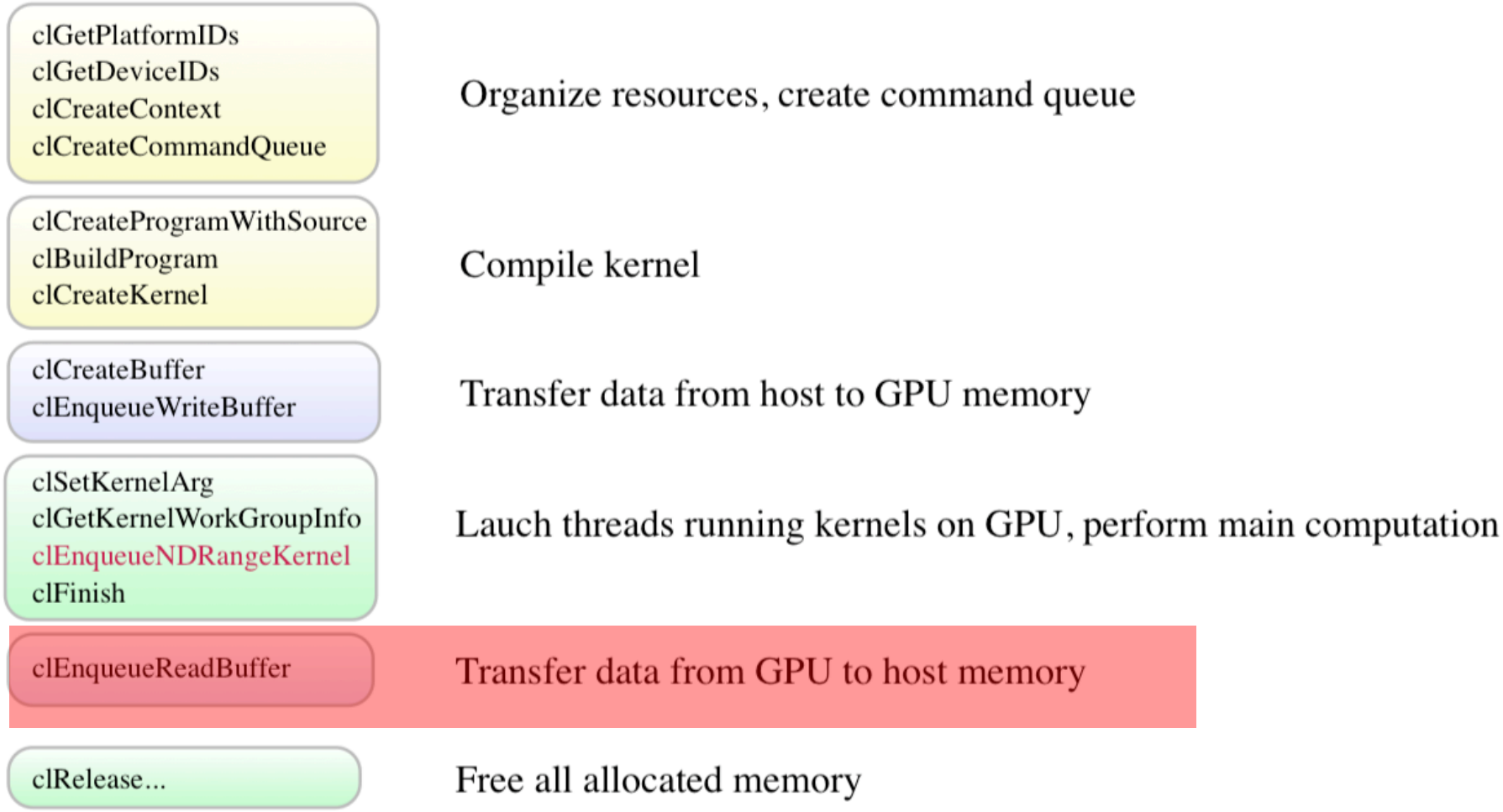
- **CL_INVALID_PROGRAM_EXECUTABLE** — No executable has been built in the program object for the device associated with the command queue.
- **CL_INVALID_COMMAND_QUEUE** — The command queue is not valid.
- **CL_INVALID_KERNEL** — The kernel object is not valid.
- **CL_INVALID_CONTEXT** — The command queue and kernel are not associated with the same context.
- **CL_INVALID_KERNEL_ARGS** — Kernel arguments have not been set.
- **CL_INVALID_WORK_DIMENSION** — The dimension is not between 1 and 3.
- **CL_INVALID_GLOBAL_WORK_SIZE** — The global work size is NULL or exceeds the range supported by the compute device.
- **CL_INVALID_WORK_GROUP_SIZE** — The local work size is not evenly divisible with the global work size or the value specified exceeds the range supported by the compute device.
- **CL_INVALID_GLOBAL_OFFSET** — The reserved global offset parameter is not set to NULL.
- **CL_INVALID_EVENT_WAIT_LIST** — The events list is empty (NULL) but the number of events arguments is greater than 0; or number of events is 0 but the event list is not NULL; or the events list contains invalid event objects.
- **CL_OUT_OF_HOST_MEMORY** — The host is unable to allocate OpenCL resources.

Synchronization Point

```
clFinish(command_queue);
```

- Blocks until all previously queued OpenCL commands in *command_queue* are issued to the associated device and have completed
- Returns CL_SUCCESS if the function call was executed successfully
- Returns CL_INVALID_COMMAND_QUEUE if *command_queue* is not a valid command-queue
- Returns CL_OUT_OF_HOST_MEMORY if there is a failure to allocate resources required by the OpenCL implementation on the host

OpenCL Program Flow (Host to GPU Memory Transfer)

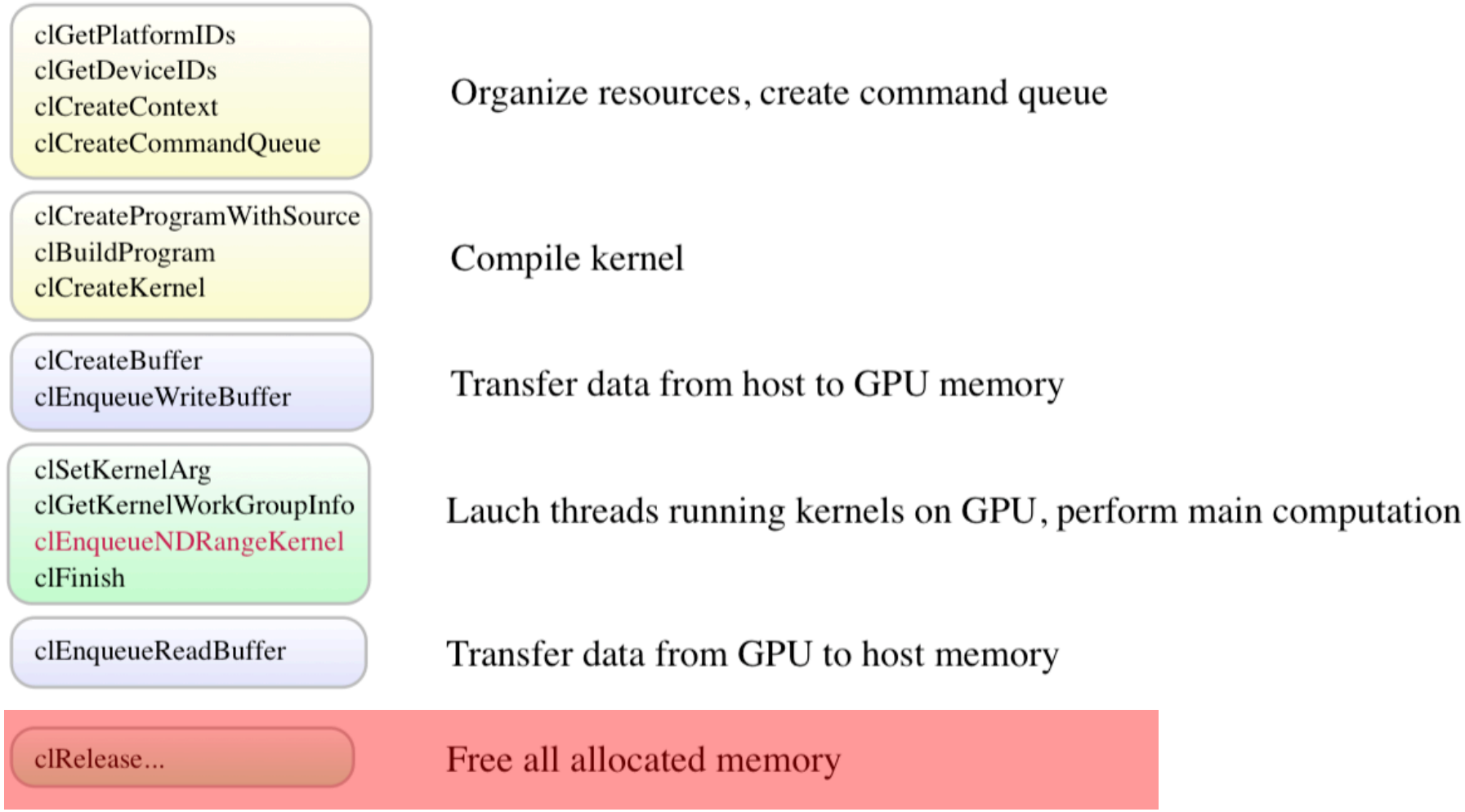


Read Buffer Objects

```
err = clEnqueueReadBuffer(
    command_queue,              // valid command queue
    output,                     // memory buffer to read from
    CL_TRUE,                    // indicate blocking read
    0,                          // the offset in the buffer object to read from
    sizeof(float) *DATA_SIZE, // size in bytes of data being read
    results,                    // pointer to buffer in host mem to store read data
    0,                          // number of event in the event list
    NULL,                       // list of events that needs to complete before this executes
    NULL);                      // event object to return on completion
```

Similar to clEnqueueWriteBuffer()

OpenCL Program Flow (Clean Up)



Release Resources

```
// Cleanup (release OpenCL resources)
clReleaseContext(context);
clReleaseCommandQueue(command_queue);
clReleaseProgram(program);
clReleaseKernel(kernel);
clReleaseMemObject(input);
clReleaseMemObject(output);
```

- All clRelease<object>() functions decrement and return a reference count to the object
  - If all resources are correctly released, the reference count should be zero
  - Otherwise, the returned reference counts can be used to track down memory leaks

Sample Code

- Two versions of implementation for squaring the values of a vector
  - vecSquare_1.cpp: create program object in online mode (with an embedded source code)
  - vecSquare_2.cpp and vecSquare_2.cl: create program object in offline mode (with a separate .cl file)
  - Compile with **oclc** command after gpu nodes have been requested:
    - ./oclc vecSquare_1
    - ./oclc vecSquare_2
- Serves as a blueprint for contrasting an OpenCL program with basic program flow