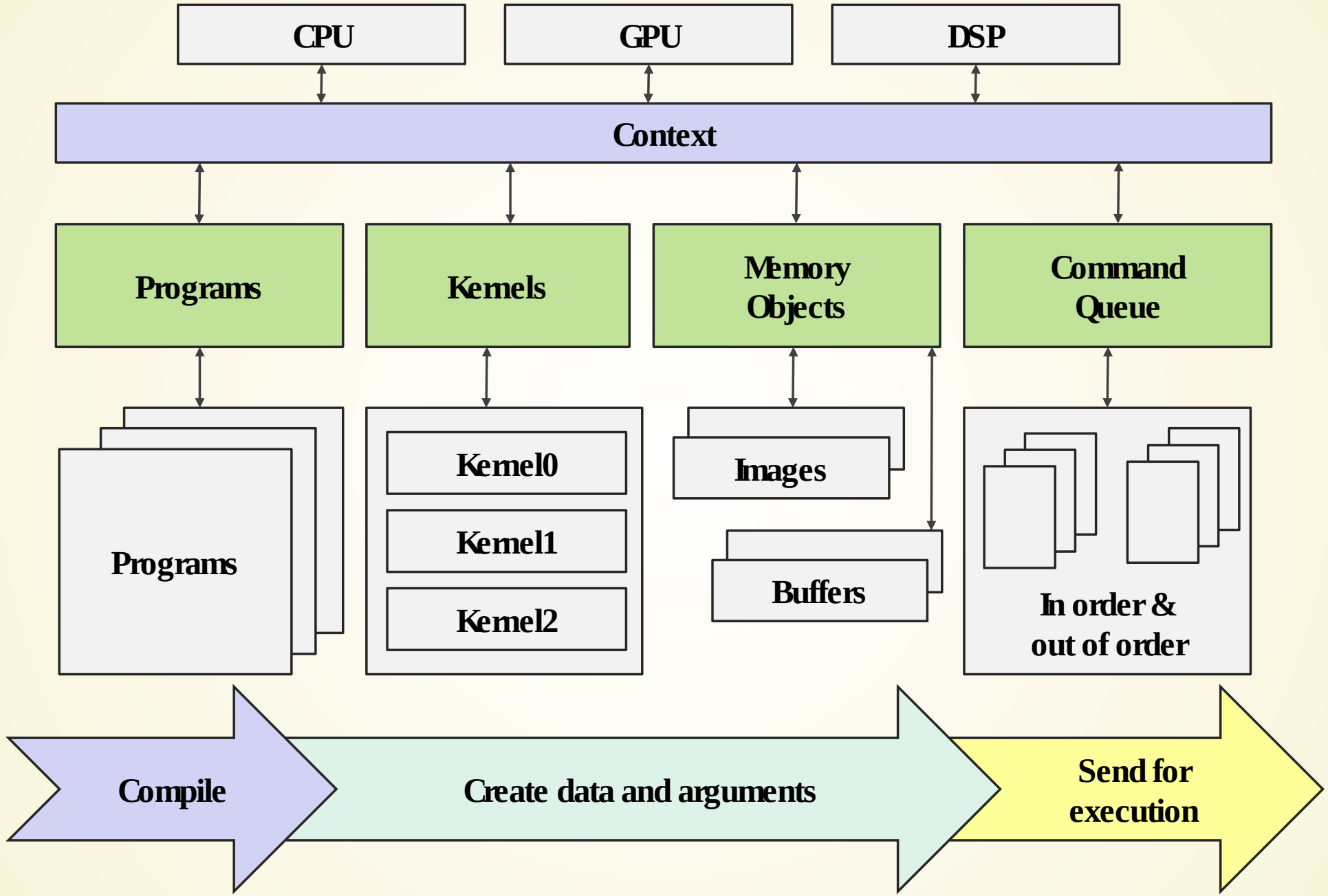
ACCELERATOR DESIGN WITH OPENCL

(ATHENS WEEK 19-24 MARCH, 2018)


CPU
GPU
DSP
Context
Programs
Kernels
Memory Objects
Command Queue
Programs
Kernel0
Kernel1
Kernel2
Images
Buffers
In order & out of order
Compile
Create data and arguments
Send for execution

Hello World

```c
#include <stdio.h>
#include <stdlib.h>
#include <CL/cl.h>
#include <CL/cl_ext.h>
#define STRING_BUFFER_LEN 1024

const char *opencl =
     "__kernel void hello()\n"
     "{\n"
     " printf(\"Hello, World!\\n\");\n"
     "}\n";



void callback(const char *buffer, size_t length, size_t final, void *user_data)
{
     fwrite(buffer, 1, length, stdout);
}

int main()
{
     char char_buffer[STRING_BUFFER_LEN];
     cl_platform_id platform;
```

clGetPlatformIDs

```
cl_int clGetPlatformIDs(cl_uint num_entries,
                        cl_platform_id *platforms,
                        cl_uint *num_platforms)

//num_entries The number of cl_platform_id entries that can be added to
//platforms. If platforms is not NULL, the num_entries must be greater than zero.

//platforms Returns a list of OpenCL platforms found. The cl_platform_id values
//returned in platforms can be used to identify a specific OpenCL platform. If
//platforms argument is NULL, this argument is ignored. The number of OpenCL
//platforms returned is the mininum of the value specified by num_entries or the
//number of OpenCL platforms available.

//num_platforms Returns the number of OpenCL platforms available. If
//num_platforms is NULL, this argument is ignored.
```

Hello World

```
#include <stdio.h>
#include <stdlib.h>
#include <CL/cl.h>
#include <CL/cl_ext.h>
#define STRING_BUFFER_LEN 1024

const char *opencl =
     "__kernel void hello()\n"
     "{\n"
     " printf(\"Hello, World!\\n\");\n"
     "}\n";



void callback(const char *buffer, size_t length, size_t final, void *user_data)
{
     fwrite(buffer, 1, length, stdout);
}

int main()
{
     char char_buffer[STRING_BUFFER_LEN];
     cl_platform_id platform;
```

clGetPlatformInfo

```
cl_int clGetPlatformInfo(    cl_platform_id platform,
                             cl_platform_info param_name,
                             size_t param_value_size,
                             void *param_value,
                             size_t *param_value_size_ret)
//platform The platform ID returned by clGetPlatformIDs or can be NULL. If
//platform is NULL, the behavior is implementation-defined.

//param_name: CL_PLATFORM_PROFILE CL_PLATFORM_VERSION CL_PLATFORM_NAME
//               CL_PLATFORM_VENDOR  CL_PLATFORM_EXTENSIONS

//param_value_size Specifies the size in bytes of memory pointed to by
//param_value. This size in bytes must be greater than or equal to size of
//return type specified in the table below.

//param_value A pointer to memory location where appropriate values for a given
//param_value will be returned. Acceptable param_value values are listed in the
//table below. If param_value is NULL, it is ignored.

//param_value_size_ret Returns the actual size in bytes of data being queried
//by param_value. If param_value_size_ret is NULL, it is ignored
```

Hello World

```
#include <stdio.h>
#include <stdlib.h>
#include <CL/cl.h>
#include <CL/cl_ext.h>
#define STRING_BUFFER_LEN 1024

const char *opencl =
     "__kernel void hello()\n"
     "{\n"
     " printf(\"Hello, World!\\n\");\n"
     "}\n";



void callback(const char *buffer, size_t length, size_t final, void *user_data)
{
     fwrite(buffer, 1, length, stdout);
}

int main()
{
     char char_buffer[STRING_BUFFER_LEN];
     cl_platform_id platform;
```

clGetDeviceIDs

```
cl_int clGetDeviceIDs (    cl_platform_id  platform ,
     cl_device_type  device_type ,
     cl_uint  num_entries ,
     cl_device_id  *devices ,
     cl_uint  *num_devices )

//platform
//Refers to the platform ID returned by clGetPlatformIDs or can be NULL. If platform is NUL

//device_type
//CL_DEVICE_TYPE_CPU
//CL_DEVICE_TYPE_GPU
//CL_DEVICE_TYPE_ACCELERATOR
//CL_DEVICE_TYPE_CUSTOM
//CL_DEVICE_TYPE_DEFAULT
//CL_DEVICE_TYPE_ALL

//num_entries
//The number of cl_device_id entries that can be added to devices. If devices is not NULL,

//devices
//A list of OpenCL devices found. The cl_device_id values returned in devices can be used t
```

Hello World

```
#include <stdio.h>
#include <stdlib.h>
#include <CL/cl.h>
#include <CL/cl_ext.h>
#define STRING_BUFFER_LEN 1024

const char *opencl =
     "__kernel void hello()\n"
     "{\n"
     " printf(\"Hello, World!\\n\");\n"
     "}\n";



void callback(const char *buffer, size_t length, size_t final, void *user_data)
{
     fwrite(buffer, 1, length, stdout);
}

int main()
{
     char char_buffer[STRING_BUFFER_LEN];
     cl_platform_id platform;
```

clCreateContext

```
cl_context clCreateContext(    cl_context_properties *properties,
                           cl_uint num_devices,
                           const cl_device_id *devices,
                           void *pfn_notify (
                               const char *errinfo,
                               const void *private_info,
                               size_t cb,
                               void *user_data),
                               void *user_data,
                           cl_int *errcode_ret)

//An OpenCL context is created with one or more devices. Contexts are used by the
//OpenCL runtime for managing objects such as command-queues, memory, program and
//kernel objects and for executing kernels on one or more devices specified in
//the context.
```

Hello World

```c
#include <stdio.h>
#include <stdlib.h>
#include <CL/cl.h>
#include <CL/cl_ext.h>
#define STRING_BUFFER_LEN 1024

const char *opencl =
     "__kernel void hello()\n"
     "{\n"
     " printf(\"Hello, World!\\n\");\n"
     "}\n";



void callback(const char *buffer, size_t length, size_t final, void *user_data)
{
     fwrite(buffer, 1, length, stdout);
}

int main()
{
     char char_buffer[STRING_BUFFER_LEN];
     cl_platform_id platform;
```

clCreateCommandQueue

```
cl_command_queue clCreateCommandQueue(    cl_context context,
                                          cl_device_id device,
                                          cl_command_queue_properties properties,
                                          cl_int *errcode_ret)

//context
//Must be a valid OpenCL context.

//device Must be a device associated with context. It can either be in the list
//of devices specified when context is created using clCreateContext or have the
//same device type as the device type specified when the context is created using
//clCreateContextFromType.

//properties Specifies a list of properties for the command-queue. This is a
//bit-field. Only command-queue properties specified in the table below can be
//set in properties; otherwise the value specified in properties is considered to
//be not valid.

//CL_QUEUE_OUT_OF_ORDER_EXEC_MODE_ENABLE CL_QUEUE_PROFILING_ENABLE

//errcode_ret Returns an appropriate error code. If errcode_ret is NULL, no error
//code is returned.
```

Hello World

```
#include <stdio.h>
#include <stdlib.h>
#include <CL/cl.h>
#include <CL/cl_ext.h>
#define STRING_BUFFER_LEN 1024

const char *opencl =
     "__kernel void hello()\n"
     "{\n"
     " printf(\"Hello, World!\\n\");\n"
     "}\n";



void callback(const char *buffer, size_t length, size_t final, void *user_data)
{
     fwrite(buffer, 1, length, stdout);
}

int main()
{
     char char_buffer[STRING_BUFFER_LEN];
     cl_platform_id platform;
```

clCreateProgramWithSource

```
cl_program clCreateProgramWithSource (    cl_context context,
                                          cl_uint count,
                                          const char **strings,
                                          const size_t *lengths,
                                          cl_int *errcode_ret)
//context
//Must be a valid OpenCL context.

//strings
//An array of count pointers to optionally null-terminated character strings that
//make up the source code.

//lengths An array with the number of chars in each string (the string length).
//If an element in lengths is zero, its accompanying string is null-terminated.
//If lengths is NULL, all strings in the strings argument are considered
//null-terminated. Any length value passed in that is greater than zero excludes
//the null terminator in its count.

//errcode_ret Returns an appropriate error code. If errcode_ret is NULL, no error
//code is returned.
```

Hello World

```
#include <stdio.h>
#include <stdlib.h>
#include <CL/cl.h>
#include <CL/cl_ext.h>
#define STRING_BUFFER_LEN 1024

const char *opencl =
     "__kernel void hello()\n"
     "{\n"
     " printf(\"Hello, World!\\n\");\n"
     "}\n";



void callback(const char *buffer, size_t length, size_t final, void *user_data)
{
     fwrite(buffer, 1, length, stdout);
}

int main()
{
     char char_buffer[STRING_BUFFER_LEN];
     cl_platform_id platform;
```

clBuildProgram

```
cl_int clBuildProgram (    cl_program program,
                        cl_uint num_devices,
                        const cl_device_id *device_list,
                        const char *options,
                        void (*pfn_notify)(cl_program, void *user_data),
                        void *user_data)

//program
//The program object

//device_list
//A pointer to a list of devices that are in program. If device_list is NULL
//value, the program executable is built for all devices associated with program
//for which a source or binary has been loaded. If device_list is a non-NULL
//value, the program executable is built for devices specified in this list for
//which a source or binary has been loaded.

//num_devices The number of devices listed in device_list.

//options A pointer to a string that describes the build options to be used for
//building the program executable. The list of supported options is described in
//"Build Options" below.
```

Hello World

```
#include <stdio.h>
#include <stdlib.h>
#include <CL/cl.h>
#include <CL/cl_ext.h>
#define STRING_BUFFER_LEN 1024

const char *opencl =
     "__kernel void hello()\n"
     "{\n"
     " printf(\"Hello, World!\\n\");\n"
     "}\n";



void callback(const char *buffer, size_t length, size_t final, void *user_data)
{
     fwrite(buffer, 1, length, stdout);
}

int main()
{
     char char_buffer[STRING_BUFFER_LEN];
     cl_platform_id platform;
```

clCreateKernel

```
cl_kernel clCreateKernel (    cl_program  program,
                              const char *kernel_name,
                              cl_int *errcode_ret)
//program
//A program object with a successfully built executable.

//kernel_name
//A function name in the program declared with the __kernel qualifier

//errcode_ret
//Returns an appropriate error code. If errcode_ret is NULL, no error code is
//returned.

//CL_INVALID_PROGRAM if program is not a valid program object.

//CL_INVALID_PROGRAM_EXECUTABLE if there is no successfully built executable for
//program.

//CL_INVALID_KERNEL_NAME if kernel_name is not found in program.

//CL_INVALID_KERNEL_DEFINITION if the function definition for __kernel function
//given by kernel_name such as the number of arguments, the argument types are
//not the same for all devices for which the program executable has been built.
```

Hello World

```
#include <stdio.h>
#include <stdlib.h>
#include <CL/cl.h>
#include <CL/cl_ext.h>
#define STRING_BUFFER_LEN 1024

const char *opencl =
     "__kernel void hello()\n"
     "{\n"
     " printf(\"Hello, World!\\n\");\n"
     "}\n";



void callback(const char *buffer, size_t length, size_t final, void *user_data)
{
     fwrite(buffer, 1, length, stdout);
}

int main()
{
     char char_buffer[STRING_BUFFER_LEN];
     cl_platform_id platform;
```

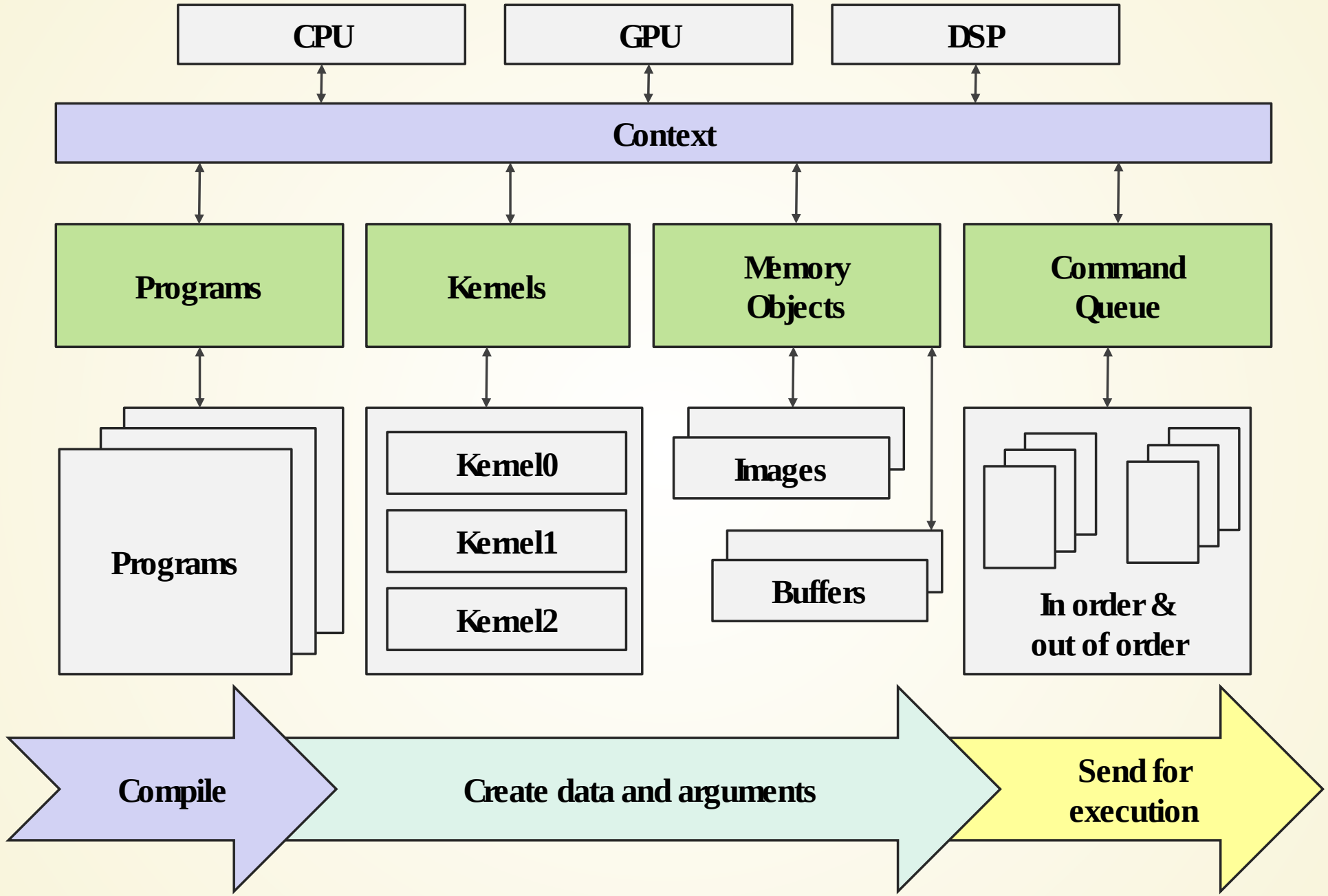
clEnqueueTask

```
cl_int clEnqueueTask (    cl_command_queue  command_queue ,
                          cl_kernel  kernel ,
                          cl_uint  num_events_in_wait_list ,
                          const cl_event  *event_wait_list ,
                          cl_event  *event )

//command_queue A valid command-queue. The kernel will be queued for execution on
//the device associated with command_queue.

//kernel A valid kernel object. The OpenCL context associated with kernel and
//command_queue must be the same.

//num_events_in_wait_list , event_wait_list Sspecify events that need to complete
//before this particular command can be executed. If event_wait_list is NULL,
//then this particular command does not wait on any event to complete. If
//event_wait_list is NULL, num_events_in_wait_list must be 0. If event_wait_list
//is not NULL, the list of events pointed to by event_wait_list must be valid and
//num_events_in_wait_list must be greater than 0. The events specified in
//event_wait_list act as synchronization points. The context associated with
//events in event_wait_list and command_queue must be the same. The memory
//associated with event_wait_list can be reused or freed after the function
//returns.
```

Hello World

```
#include <stdio.h>
#include <stdlib.h>
#include <CL/cl.h>
#include <CL/cl_ext.h>
#define STRING_BUFFER_LEN 1024

const char *opencl =
     "__kernel void hello()\n"
     "{\n"
     " printf(\"Hello, World!\\n\");\n"
     "}\n";



void callback(const char *buffer, size_t length, size_t final, void *user_data)
{
     fwrite(buffer, 1, length, stdout);
}

int main()
{
     char char_buffer[STRING_BUFFER_LEN];
     cl_platform_id platform;
```

https://www.khronos.org/registry/OpenCL/sdk/1.2/


CPU
GPU
DSP
Context
Programs
Kernels
Memory Objects
Command Queue
Programs
Kernel0
Kernel1
Kernel2
Images
Buffers
In order & out of order
Compile
Create data and arguments
Send for execution

Vector Addition

```
#include <stdio.h>
#include <stdlib.h>
#include <iostream> // for standard I/O
#include <math.h>
#include <time.h>
#include <CL/cl.h>
#include <CL/cl_ext.h>
#define STRING_BUFFER_LEN 1024
using namespace std;



void print_clbuild_errors(cl_program program,cl_device_id device)
    {
        cout<<"Program Build failed\n";
        size_t length;
        char buffer[2048];
        clGetProgramBuildInfo(program, device, CL_PROGRAM_BUILD_LOG, sizeof(buffer), buffer
        cout<<"--- Build log ---\n "<<buffer<<endl;
        exit(1);
    }
unsigned char ** read_file(const char *name) {
```

Creating CL Buffers

```
#include <stdio.h>
#include <stdlib.h>
#include <iostream> // for standard I/O
#include <math.h>
#include <time.h>
#include <CL/cl.h>
#include <CL/cl_ext.h>
#define STRING_BUFFER_LEN 1024
using namespace std;



void print_clbuild_errors(cl_program program,cl_device_id device)
    {
        cout<<"Program Build failed\n";
        size_t length;
        char buffer[2048];
        clGetProgramBuildInfo(program, device, CL_PROGRAM_BUILD_LOG, sizeof(buffer), buffer
        cout<<"--- Build log ---\n "<<buffer<<endl;
        exit(1);
    }
unsigned char ** read_file(const char *name) {
```

Copying Host Buffers to CL buffers

```
#include <stdio.h>
#include <stdlib.h>
#include <iostream> // for standard I/O
#include <math.h>
#include <time.h>
#include <CL/cl.h>
#include <CL/cl_ext.h>
#define STRING_BUFFER_LEN 1024
using namespace std;



void print_clbuild_errors(cl_program program,cl_device_id device)
    {
        cout<<"Program Build failed\n";
        size_t length;
        char buffer[2048];
        clGetProgramBuildInfo(program, device, CL_PROGRAM_BUILD_LOG, sizeof(buffer), buffer
        cout<<"--- Build log ---\n "<<buffer<<endl;
        exit(1);
    }
unsigned char ** read_file(const char *name) {
```

Setting Kernel Arguments

```
#include <stdio.h>
#include <stdlib.h>
#include <iostream> // for standard I/O
#include <math.h>
#include <time.h>
#include <CL/cl.h>
#include <CL/cl_ext.h>
#define STRING_BUFFER_LEN 1024
using namespace std;



void print_clbuild_errors(cl_program program,cl_device_id device)
    {
        cout<<"Program Build failed\n";
        size_t length;
        char buffer[2048];
        clGetProgramBuildInfo(program, device, CL_PROGRAM_BUILD_LOG, sizeof(buffer), buffer
        cout<<"--- Build log ---\n "<<buffer<<endl;
        exit(1);
    }
unsigned char ** read_file(const char *name) {
```

Launching Job

```
#include <stdio.h>
#include <stdlib.h>
#include <iostream> // for standard I/O
#include <math.h>
#include <time.h>
#include <CL/cl.h>
#include <CL/cl_ext.h>
#define STRING_BUFFER_LEN 1024
using namespace std;



void print_clbuild_errors(cl_program program,cl_device_id device)
    {
        cout<<"Program Build failed\n";
        size_t length;
        char buffer[2048];
        clGetProgramBuildInfo(program, device, CL_PROGRAM_BUILD_LOG, sizeof(buffer), buffer
        cout<<"--- Build log ---\n "<<buffer<<endl;
        exit(1);
    }
unsigned char ** read_file(const char *name) {
```

Reading back the result.

```
#include <stdio.h>
#include <stdlib.h>
#include <iostream> // for standard I/O
#include <math.h>
#include <time.h>
#include <CL/cl.h>
#include <CL/cl_ext.h>
#define STRING_BUFFER_LEN 1024
using namespace std;



void print_clbuild_errors(cl_program program,cl_device_id device)
    {
        cout<<"Program Build failed\n";
        size_t length;
        char buffer[2048];
        clGetProgramBuildInfo(program, device, CL_PROGRAM_BUILD_LOG, sizeof(buffer), buffer
        cout<<"--- Build log ---\n "<<buffer<<endl;
        exit(1);
    }
unsigned char ** read_file(const char *name) {
```

Reading back the result.

```
#include <stdio.h>
#include <stdlib.h>
#include <iostream> // for standard I/O
#include <math.h>
#include <time.h>
#include <CL/cl.h>
#include <CL/cl_ext.h>
#define STRING_BUFFER_LEN 1024
using namespace std;



void print_clbuild_errors(cl_program program,cl_device_id device)
    {
        cout<<"Program Build failed\n";
        size_t length;
        char buffer[2048];
        clGetProgramBuildInfo(program, device, CL_PROGRAM_BUILD_LOG, sizeof(buffer), buffer
        cout<<"--- Build log ---\n "<<buffer<<endl;
        exit(1);
    }
unsigned char ** read_file(const char *name) {
```

REFERENCES

https://www.khronos.org/assets/uploads/developers/library/20
pan-pacific-road-show-June/OpenCL-Details-
Taiwan_June-2012.pdf
https://www.khronos.org/registry/OpenCL/sdk/1.2/