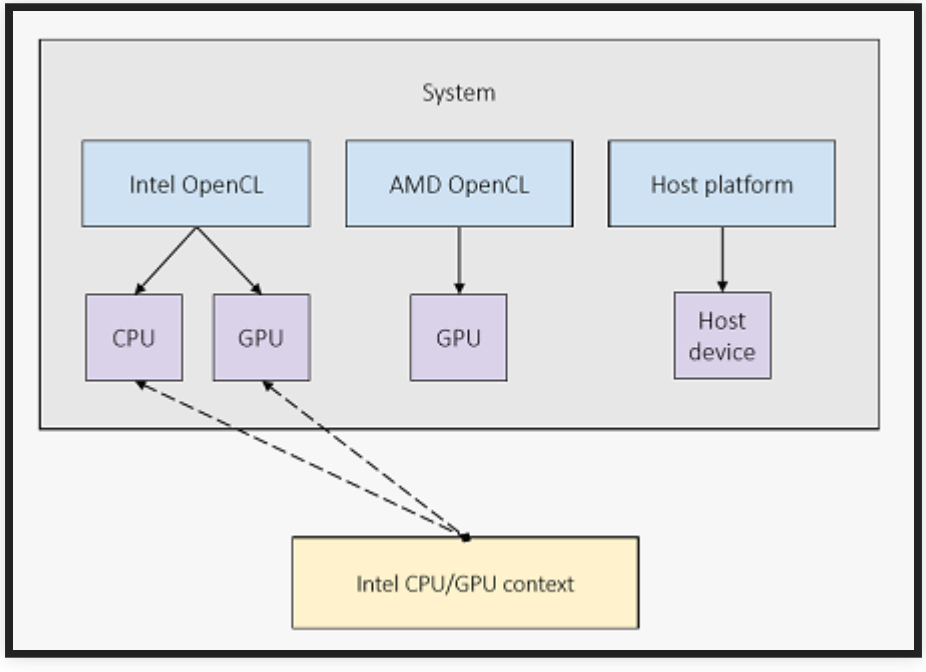
CONFIGURING A SYCL QUEUE

LEARNING OBJECTIVES

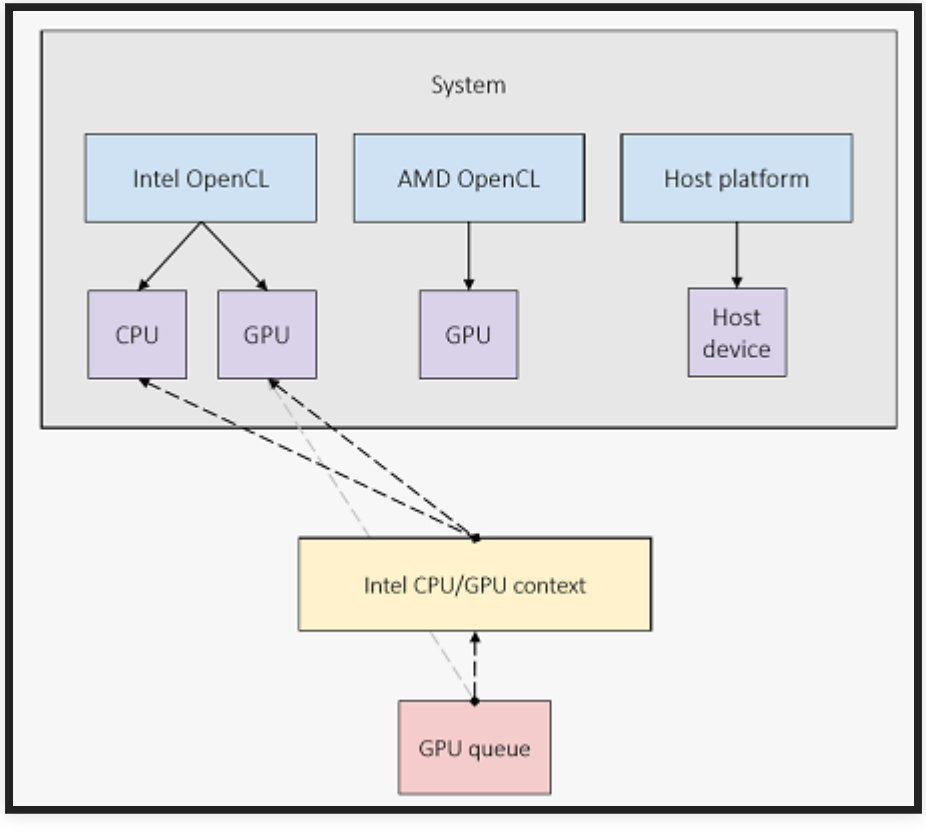
- Learn about the SYCL queue and what it does
- Learn about command groups and the command group handler
- Learn about the SYCL scheduler model

WHAT IS A SYCL QUEUE

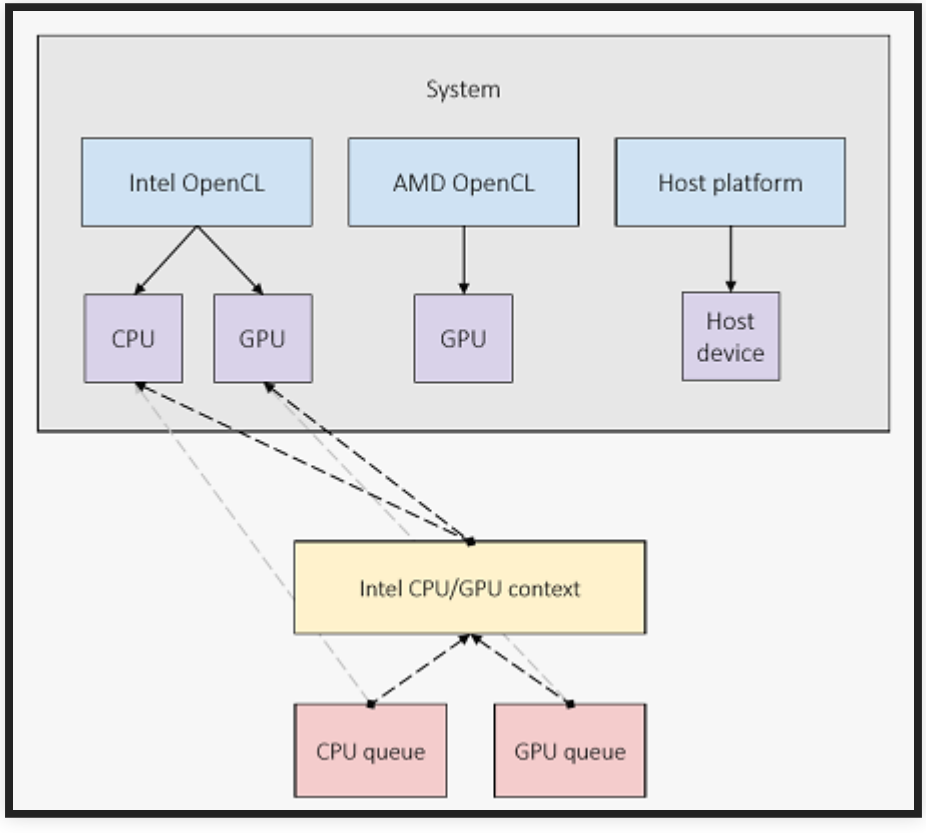
- Manages the execution of commands on your device(s)
- Jobs are submitted to the queue to be executed on the devices



- In SYCL the underlying execution and memory resources of a platform and its devices is managed by creating a context
- A context represents one or more devices, but all devices must be associated with the same platform



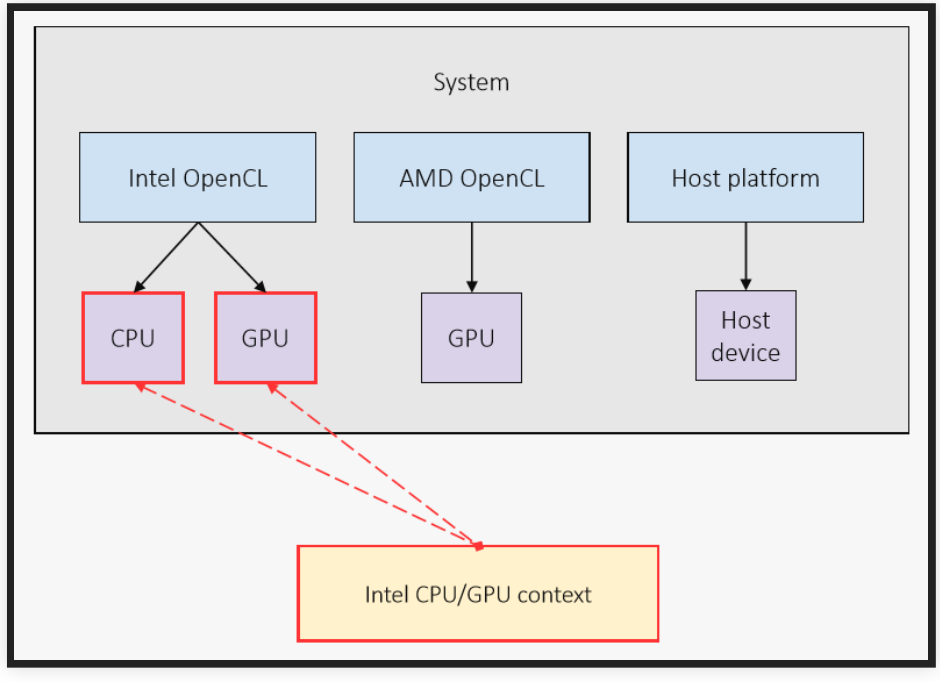
- In SYCL the object which is used to submit work is the queue
- A queue is associated with a context and a specific device



- A single SYCL application will often want to target several different devices
- This can be useful for task level parallelism and load balancing

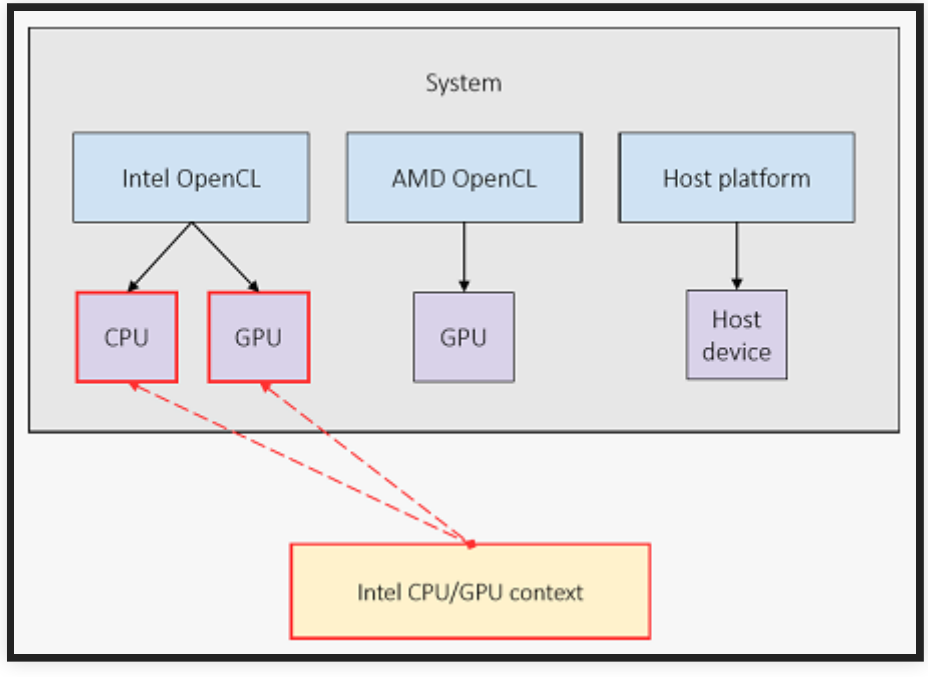
CREATING A CONTEXT

```
auto defaultContext = context{};
```



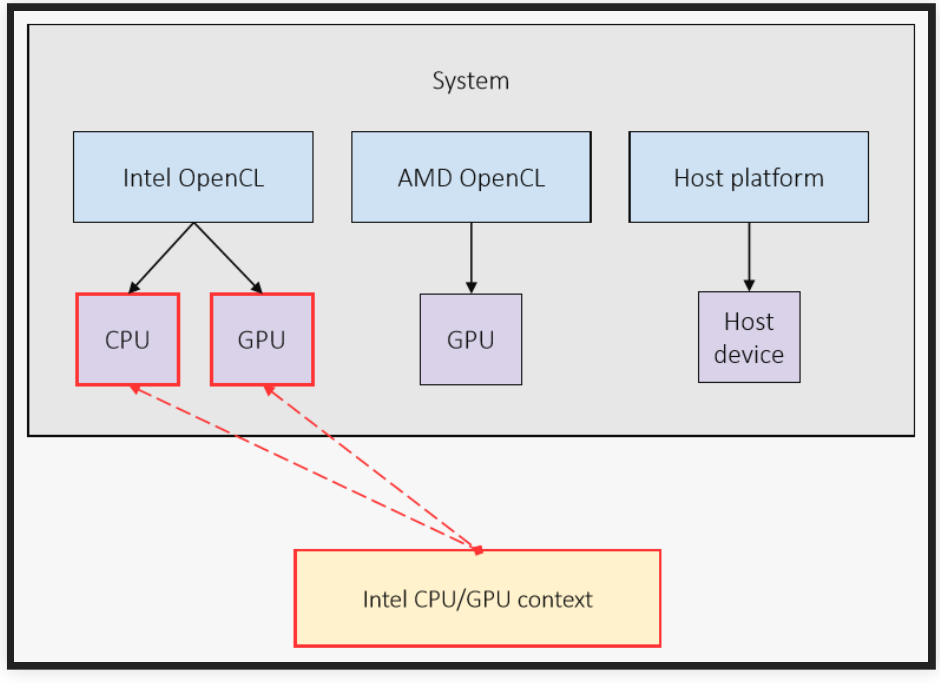
- A default constructed context object will use the **default_selector** to choose a device
- It will then create a context for all devices that share a platform with the chosen device

```
auto intelSelector = intel_selector{};
auto intelContext = context{intelSelector};
```



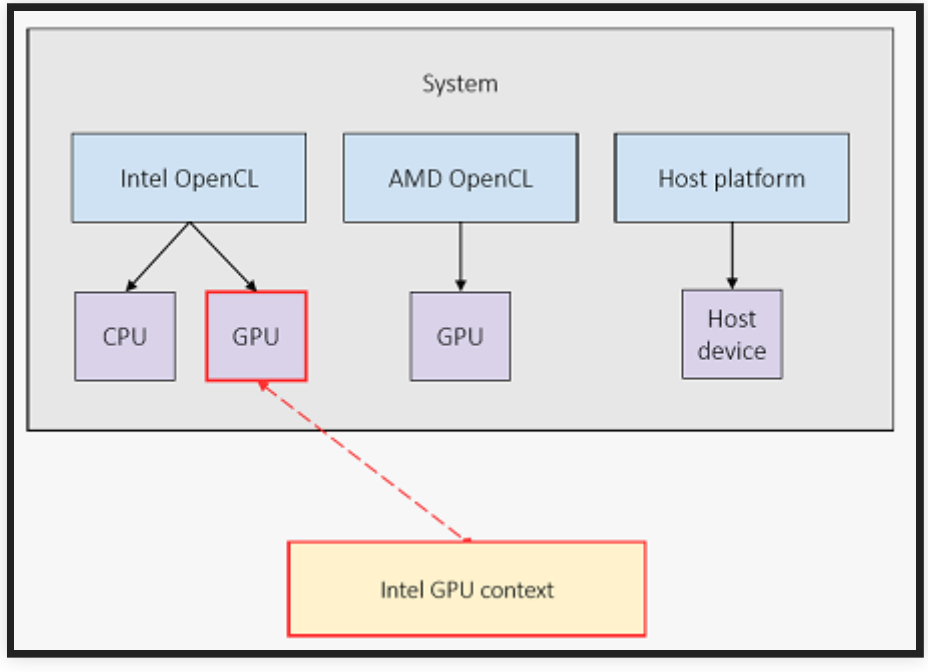
- A context object can be constructed from a device selector
- This will use the device selector to choose a device and then create a context for all devices that share a platform with the chosen device

```
auto intelContext = context{intelPlatform};
```



- A context object can also be constructed from a platform
- This will create a context for all devices associated with that platform

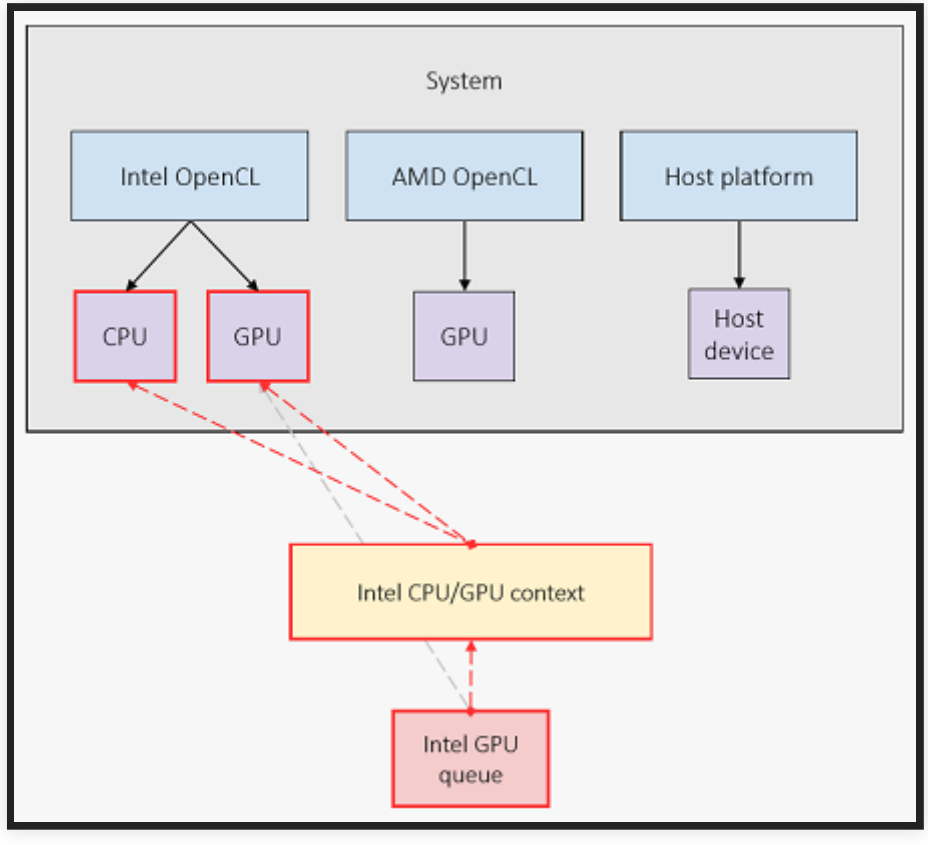
```
auto intelGPUContext = context{intelGPUDevice};
```



- Further, a context object can be constructed from a device
- This will create a context for only that device

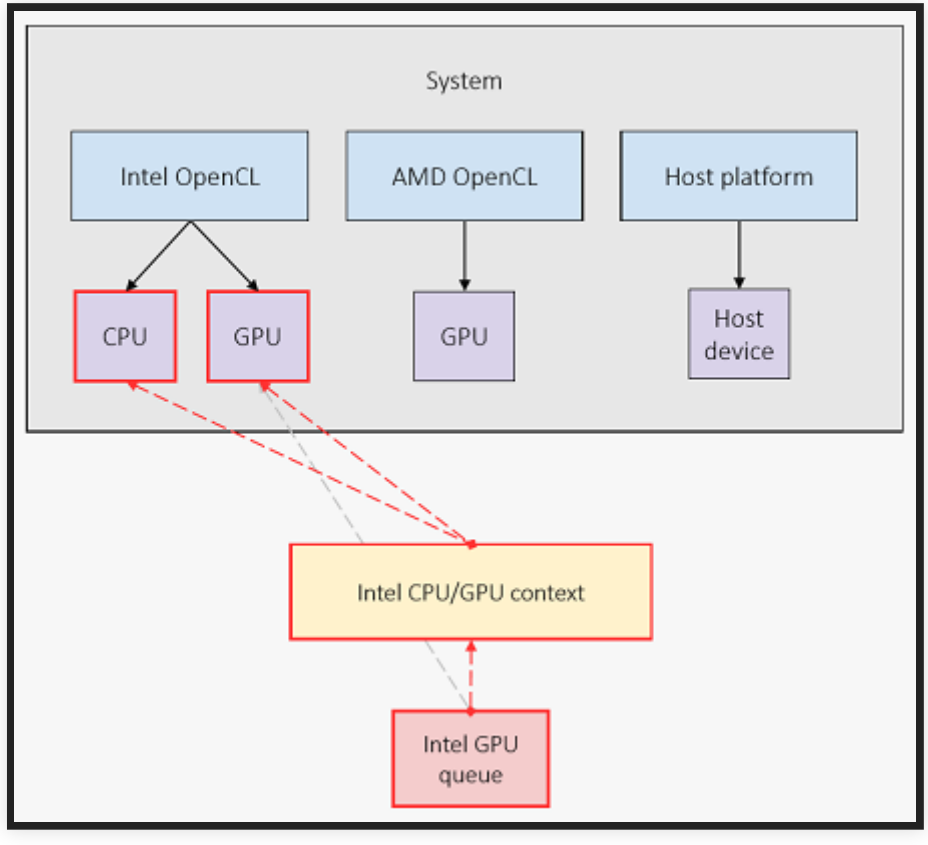
CREATING A QUEUE

```
auto defaultQueue = queue{};
```



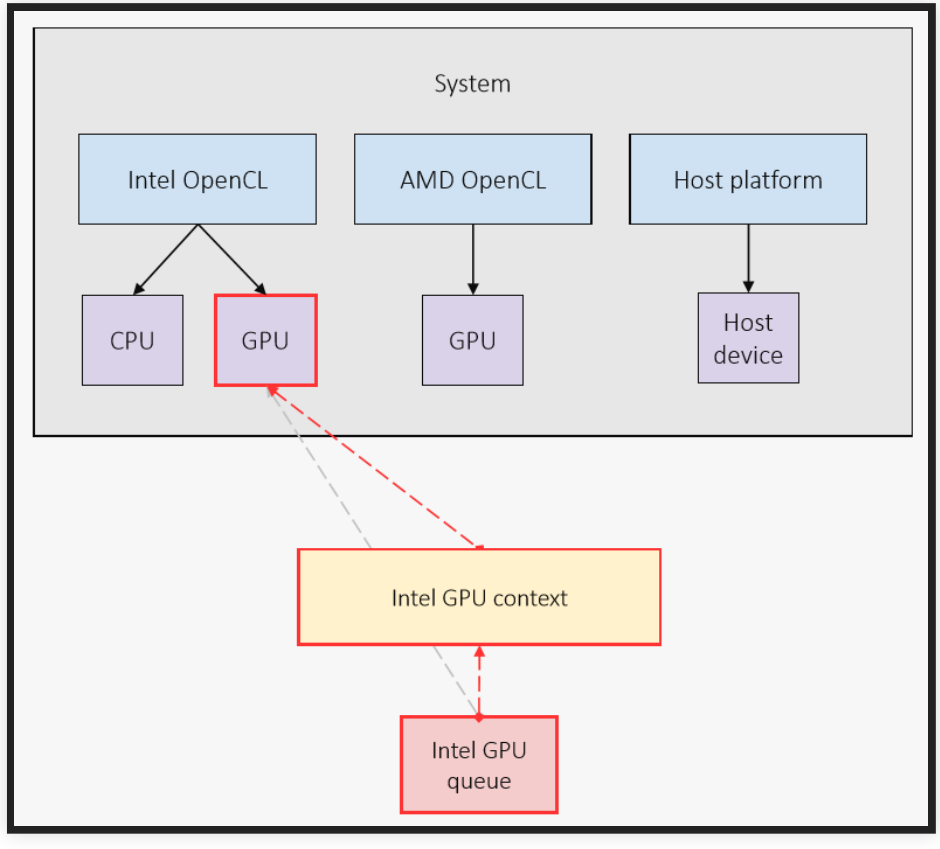
- A default constructed queue object will use the **default_selector** to choose a device
- It will then create a queue for the chosen device, creating an implicit context for all devices that share a platform with the chosen device

```
auto intelGPUSelector = intel_gpu_selector{};
auto intelGPUQeuue = queue{intelGPUSelector};
```



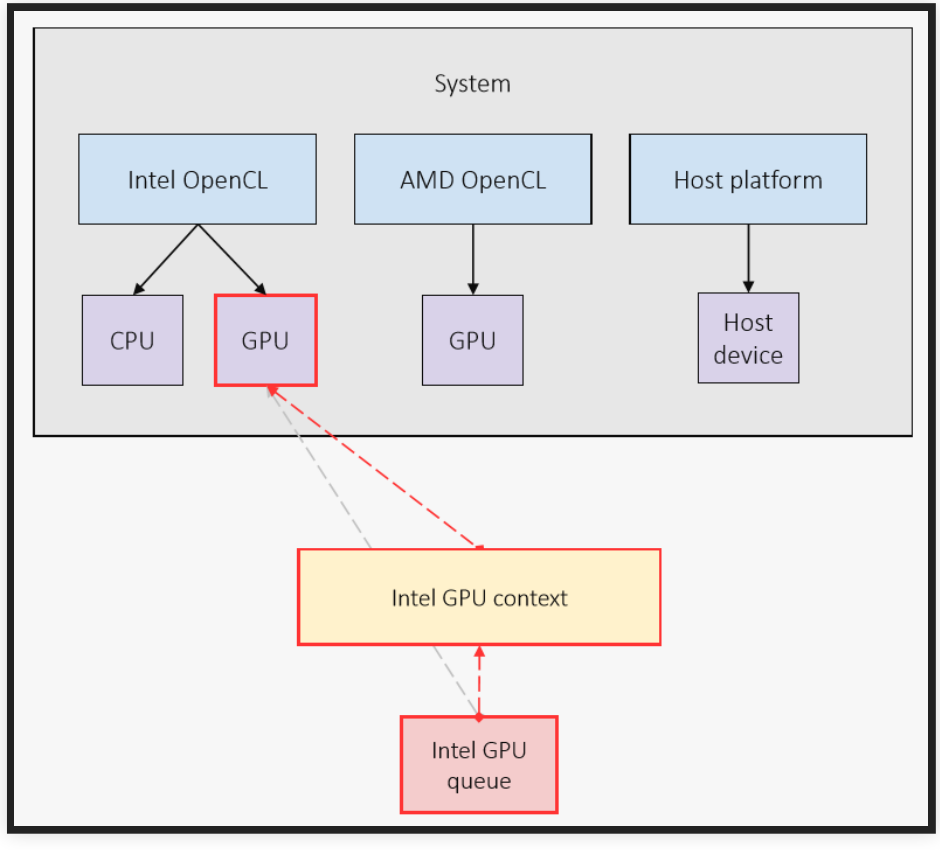
- A queue object can be constructed from a device selector which is used to choose a device
- It will then create a queue for the chosen device, creating an implicit context for all devices that share a platform with the chosen device

```
auto intelGPUSelector = intel_gpu_selector{};
auto intelGPUQeuue = queue{intelContext, intelGPUSel
```



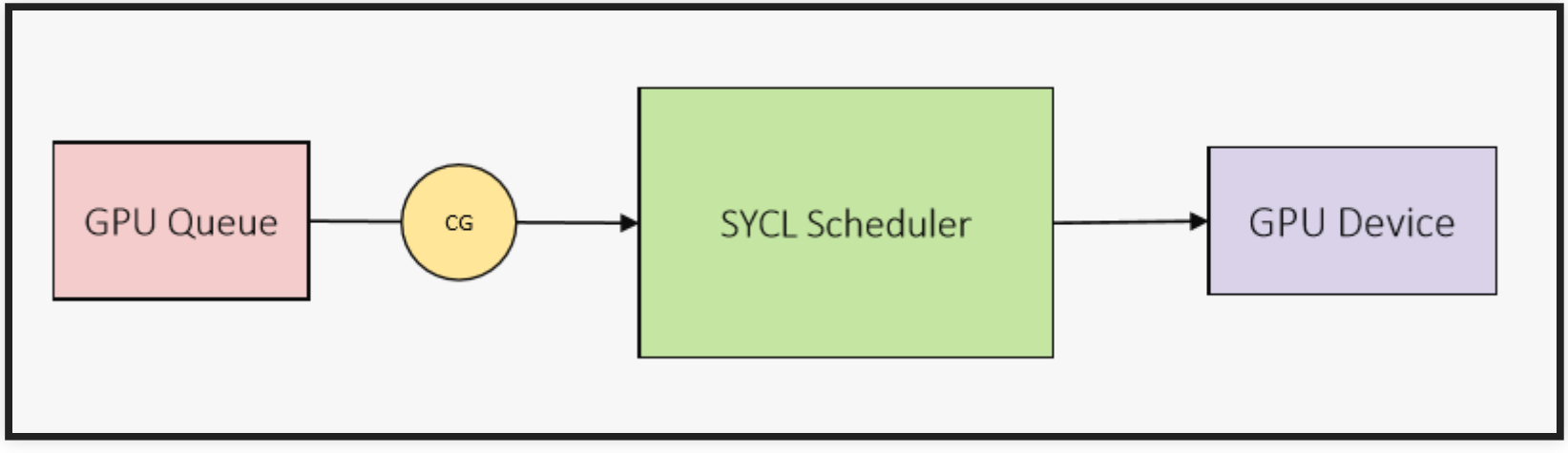
- A queue object can be constructed from a context
- A device selector must also be provided which is used to choose a device
- It will then create a queue for the chosen device, using the context provided

```
auto intelGPUSel = intel_gpu_selector{};
auto intelGPUDevice = intelGPUSel.select_device();
auto intelGPUQeuue = queue{intelGPUDevice};
```

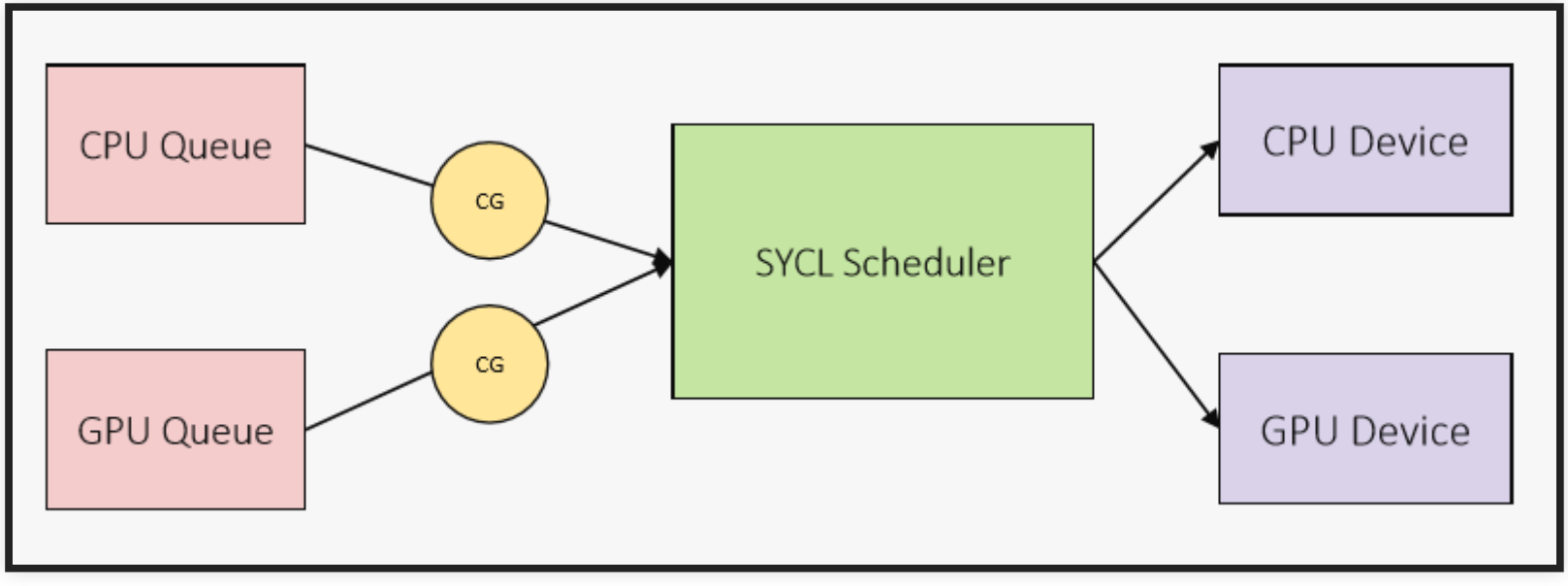


- A queue object can be constructed from a device
- This will create a queue for only that device

SUBMITTING WORK TO A QUEUE



- In SYCL work is submitted via a queue object
- This is done using the **submit** member function
- This will submit a command group to the SYCL scheduler for execution on the device associated with the queue



- The same scheduler is used for all queues in order to share dependency information

```
#include <CL/sycl.hpp> using namespace cl::sycl;
int main(int argc, char *argv[]) {
  queue gpuQueue(gpu_selector{});
  gpuQueue.submit([&](handler &cgh){

  // Command group

  });


}
```

- The **submit** member function takes a C++ function object, which takes a reference to a **handler** object
- The function object can be a lambda or a class with a function call operator
- The body of the function object represents the command group that is being submitted
- The handler object is created by the SYCL runtime and is used to link commands and requirements declared inside the command group

```
#include <CL/sycl.hpp> using namespace cl::sycl;
int main(int argc, char *argv[]) {
  queue gpuQueue(gpu_selector{});
  gpuQueue.submit([&](handler &cgh){

  // Command group

  });


}
```

- The command group is processed exactly once when **submit** is called
- At this point all the commands and requirements declared inside the command group are collected together, processed and passed on to the scheduler
- The work is then enqueued to the device asynchronously by the SYCL scheduler, potentially in another thread

```
#include <CL/sycl.hpp> using namespace cl::sycl;
int main(int argc, char *argv[]) {
  queue gpuQueue(gpu_selector{});
  gpuQueue.submit([&](handler &cgh){

  // Command group

  });

  gpuQueue.wait();

}
```

- The queue object will not wait for work to complete on destruction
- You must wait on the queue to complete if there are no data dependencies