SYCL TOPOLOGY DISCOVERY

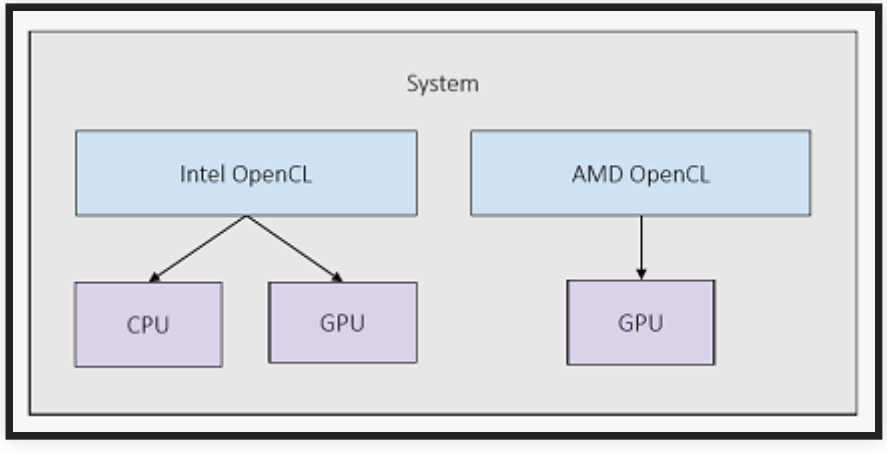
LEARNING OBJECTIVES

- Learn about the SYCL topology discovery
  - Learn how to query information about a platform or device
  - Learn how to select a device using a device selector

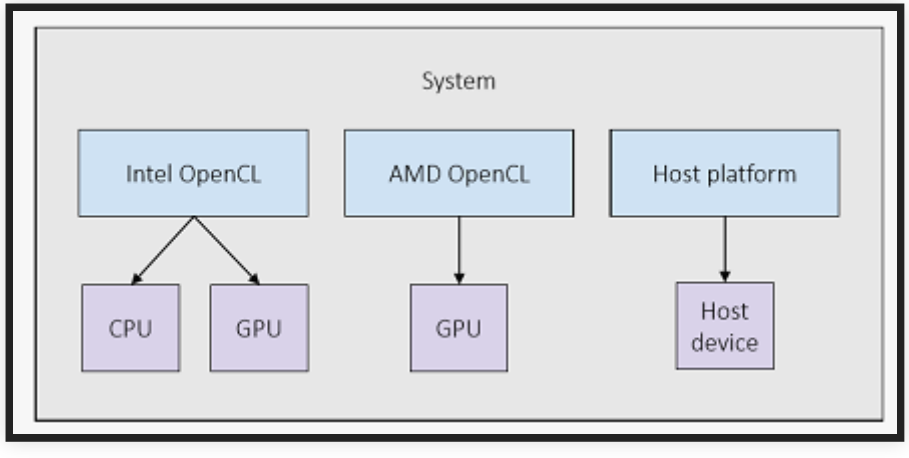
SYCL SYSTEM TOPOLOGY

- A SYCL application can execute work across a range of different heterogeneous devices
- The devices that are available in any given system are determined at runtime through topology discovery

- SYCL finds a set of platforms that are available, and their devices
- CPU, GPU, accelerator...
- Each devices is associated with a single platform

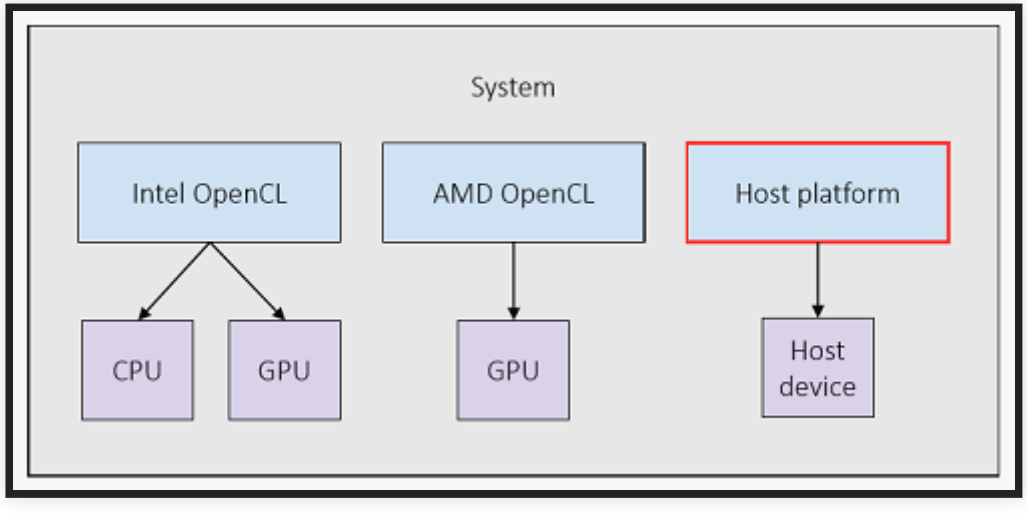


- In SYCL there is also a host device which executes SYCL kernels as native C++
  - The host device emulates the execution and memory model of an OpenCL device
- This is very useful for debugging SYCL kernel
- There is only ever one host device and that device is associated with a host platform
  - It is most likely a CPU



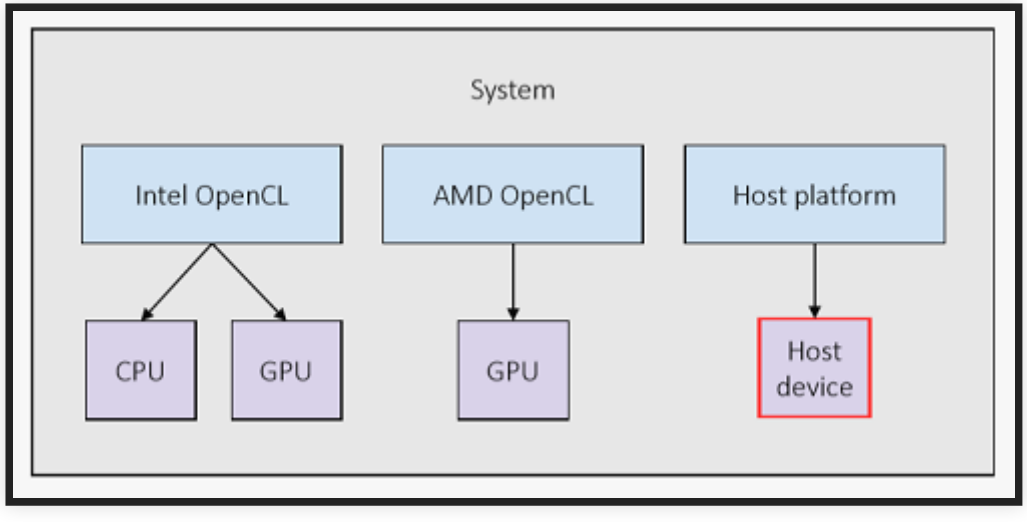
```
auto hostPlatform = platform{};
```

- The **platform** and **device** classes encapsulate these
- A default constructed platform object represents the host platform



```
auto hostDevice = device{};
```

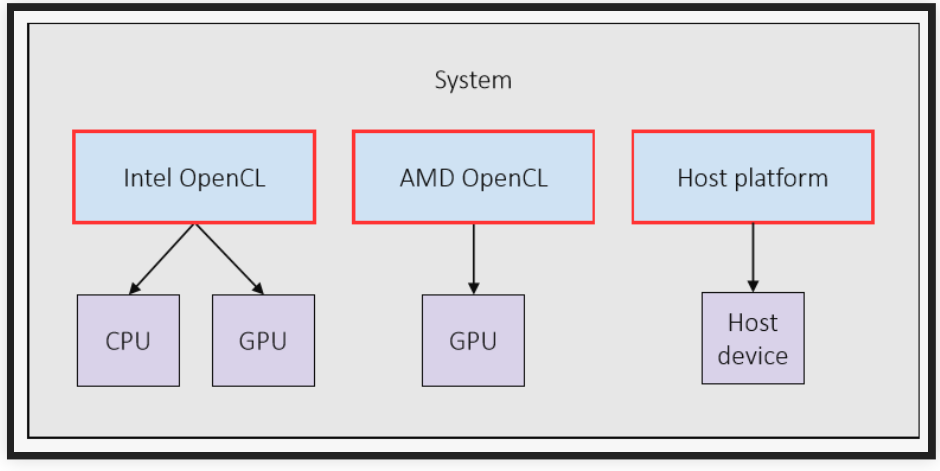
- A default constructed device object represents the host device



- In SYCL there are two ways to query a system's topology
  - The topology can be manually queried and iterated over via APIs of the platform and device classes
  - The topology can be automatically queried and iterated over using a use specified heuristic by a device selector object

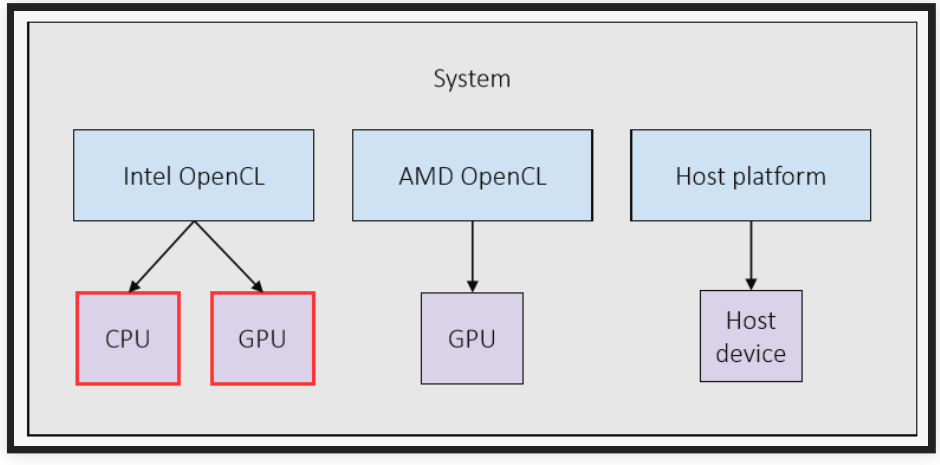
QUERYING THE TOPOLOGY MANUALLY

```
auto platforms = platform::get_platforms();
```



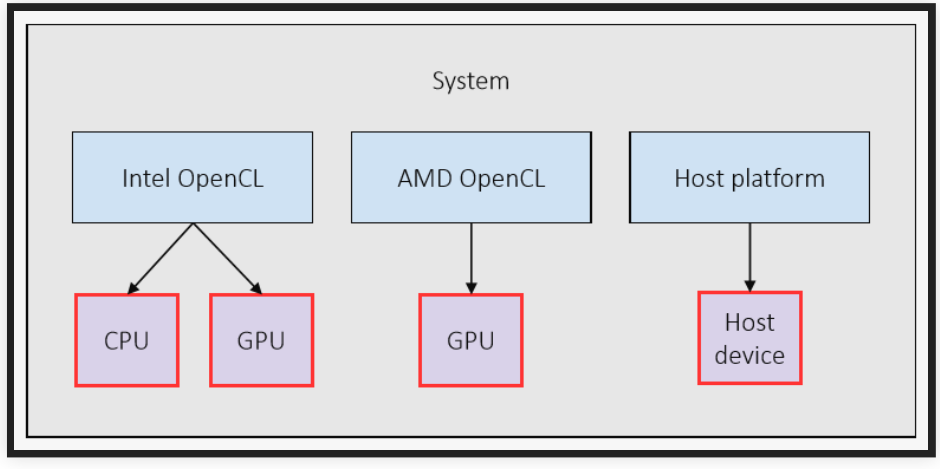
- The platform class provides the static function **get_platforms**
  - It retrieves a vector of all available platforms in the system
- This includes the host platform

```
auto intelDevices = intelPlatform.get_devices();
```



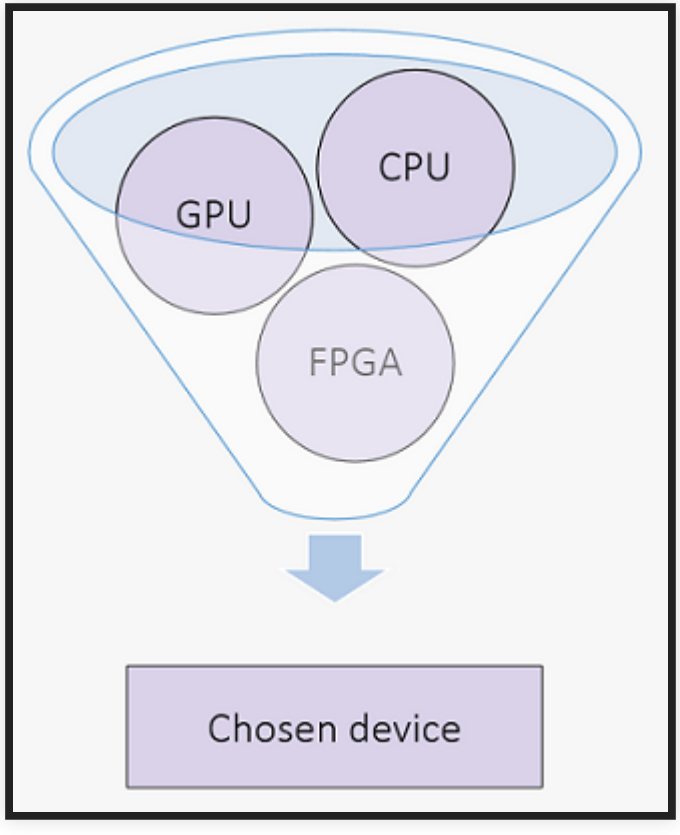
- The platform class provides the member function **get_devices** that
  - It returns a vector of all devices associated with that platform
- This includes the host device if the platform object represents a host platform

```
auto devices = device::get_devices();
```



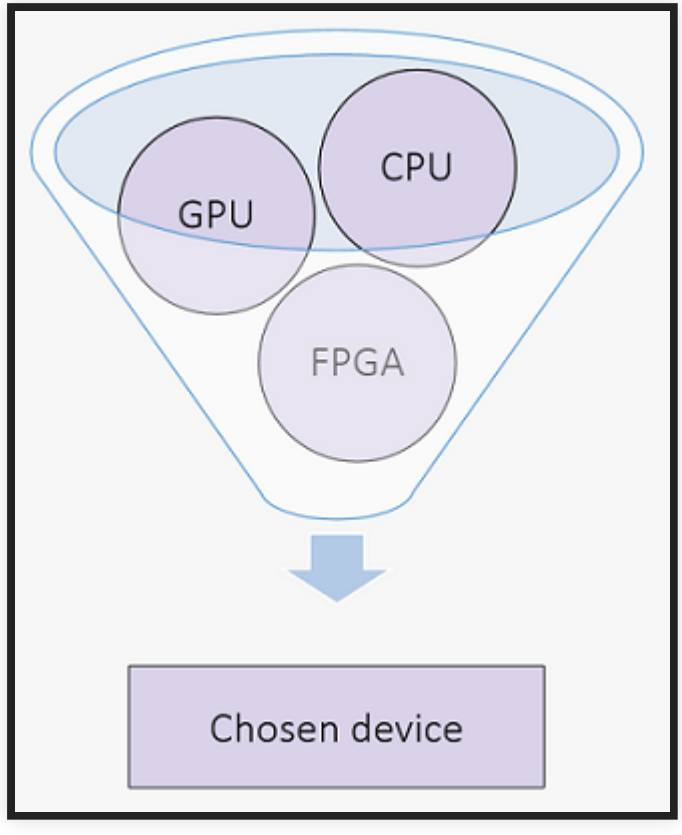
- The device class also provides the static function **get_devices**
  - It retrieves a vector of all available devices in the system
- This includes the host device

QUERYING THE TOPOLOGY USING A DEVICE SELECTOR



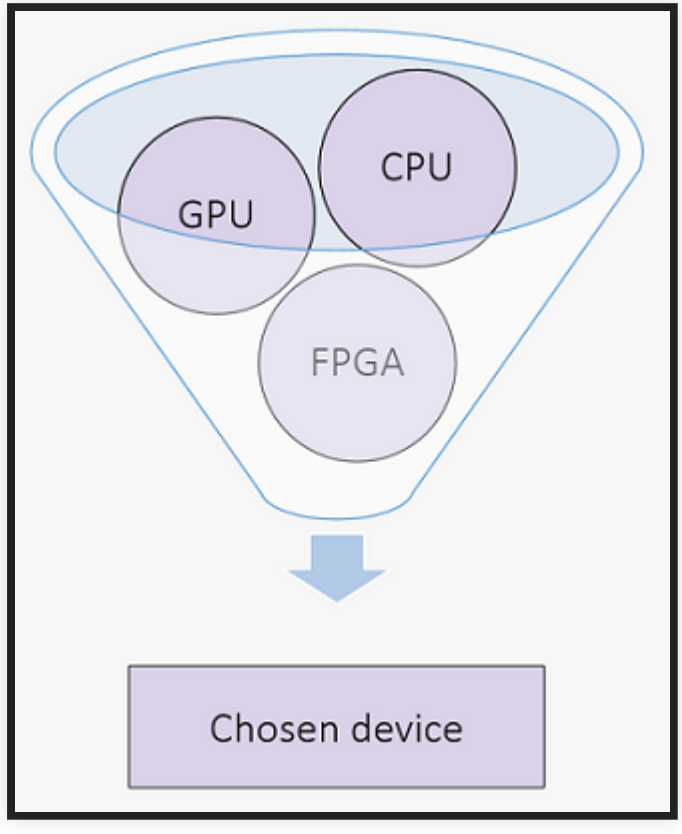
- To simplify the process of traversing the system topology SYCL provides device selectors
- The **device_selector** class device selector is a C++ function object, inherited , which defines a heuristic for scoring devices
- SYCL provides a number of standard device selectors, e.g. **default_selector**, **gpu_selector**, ...
- Users can also create their own device selectors

```
auto gpuSelector = gpu_selector{};
auto gpuDevice = gpuSelector.select_device();
```



- The **device_selector** class provides the member function **select_device**
  - Queries all devices and returns the one with the highest "score"
- A device with a negative score will never be chosen

```
auto defSelector = default_selector{};
auto chosenDevice = defSelector.select_device();
```



- The **default_selector** is a standard device selector type
  - Chooses a device based on an implementation defined heuristic

CUSTOM DEVICE SELECTORS

- It is possible to set rules for device selection
- For example only GPUs by a specific manufacturer

```
#include <CL/sycl.hpp>
using namespace cl::sycl;

struct gpu_selector : public device_selector {

  int operator()(const device& dev) const override {




  }
};

int main(int argc, char *argv[]) {


}
```

- A device selector must inherit from the default selector
- A device selector must have a function call operator which takes a reference to a device

```
#include <CL/sycl.hpp>
using namespace cl::sycl;

struct gpu_selector : public device_selector {

  int operator()(const device& dev) const override {
    if (dev.is_gpu()){
          return 1;
        }
        else {
          return -1;
        }


  }
};

int main(int argc, char *argv[]) {


}
```

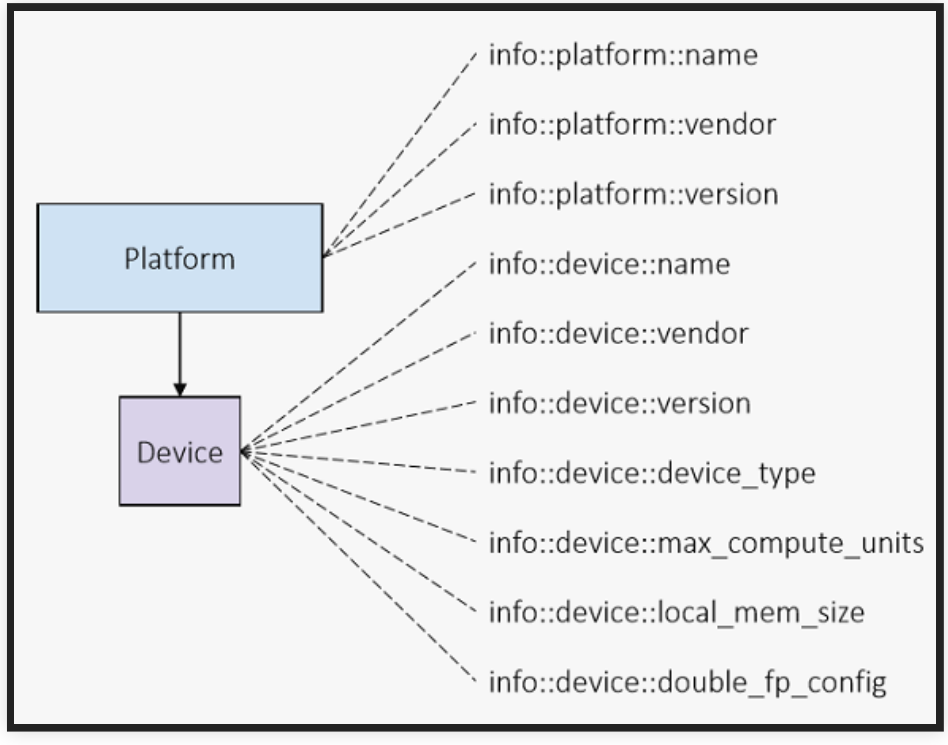
- The body of the function call operator defines the heuristic for selecting devices
- This is where you write the logic for scoring each device

```
#include <CL/sycl.hpp>
using namespace cl::sycl;

struct gpu_selector : public device_selector {

  int operator()(const device& dev) const override {
    if (dev.is_gpu()){
          return 1;
        }
        else {
          return -1;
        }



  }
};

int main(int argc, char *argv[]) {
  auto gpuQueue = queue{gpu_selector{}};

}
```

- Now that there is a device selector that chooses a specific device we can use that to construct a queue

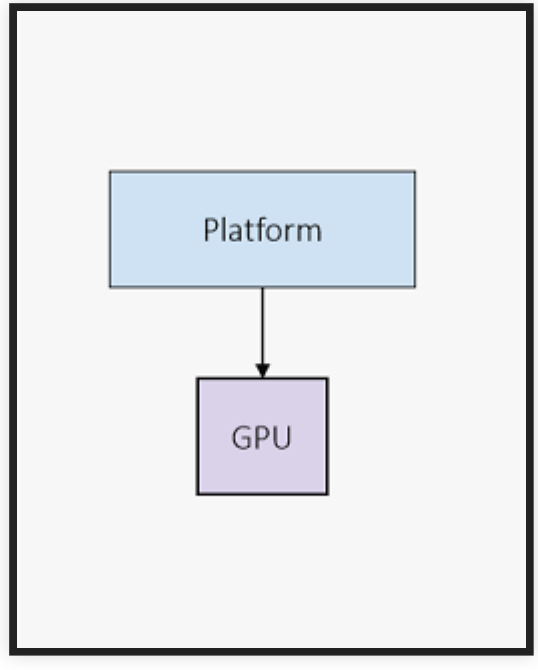
SELECTING A SYCL DEVICE

```
auto plt = dev.get_platform();
auto platformName = dev.get_info<info::device::name
```



- Information about platforms and devices can be queried using the template member function **get_info**
- The info that you are querying is specified by the template parameter
- You can also query a device for its associated platform with the get_platformmember function

```
auto platformSupportsSpir = plt.has_extension("cl_khr_sp
```



- A platform or device can also be queried for supported OpenCL extensions
  - Use the **has_extension** member function
- The extension is specified as a string parameter
- This checks for both KHR and vendor extensions