DEVICE DISCOVERY

LEARNING OBJECTIVES

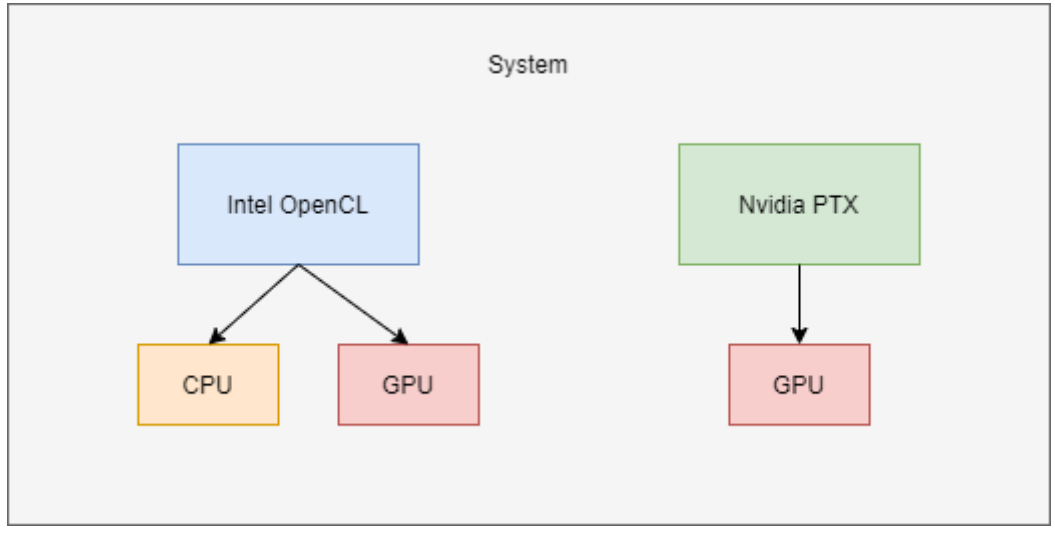
- Learn about the SYCL system topology and how to traverse it
- Learn how to query information about a platform or device
- Learn how to select a device; both manually and using device selectors

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.

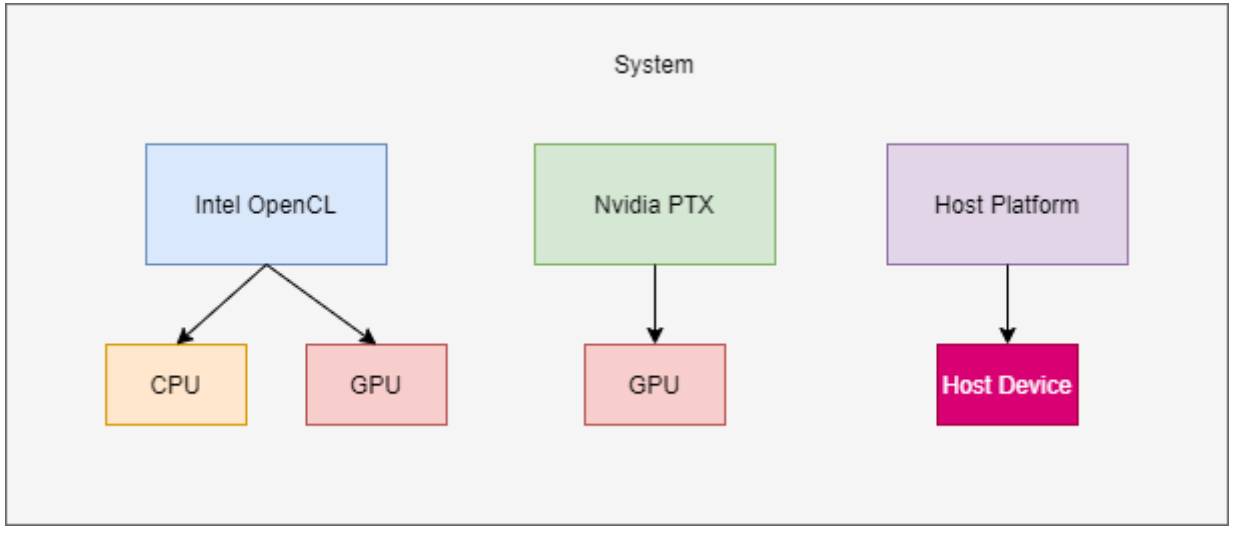
PLATFORMS AND DEVICES

- The SYCL runtime will discover a set of platforms that are available in the system.
  - Each platform represents a backend implementation such as Intel OpenCL or Nvidia PTX.
- The SYCL runtime will also discover all the devices available for each of those platforms.
  - CPU, GPU, FPGA, and other kinds of accelerators.



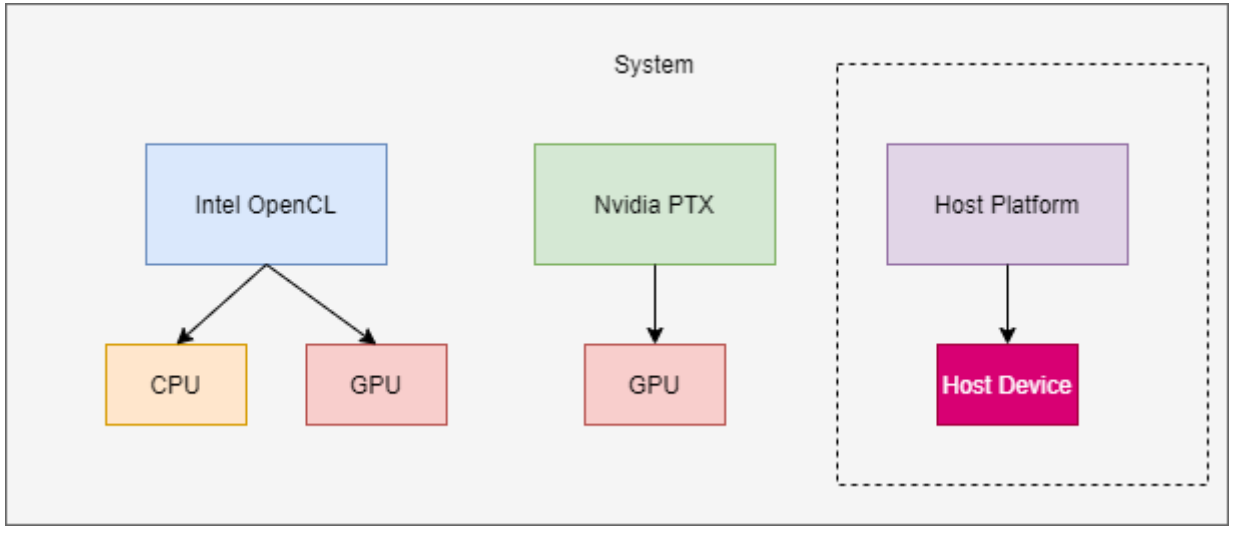
HOST DEVICE

- 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 a SYCL device.
- This is very useful for debugging SYCL kernels.
- There is only ever one host device and that device is associated with a host platform.
  - This is generally a CPU implementation.



PLATFORM AND DEVICE CLASSES

- Platforms and devices are represented by the `platform` and `device` classes respectively.
- A default constructed platform object represents the host platform.
- A default constructed device object represents the host device.

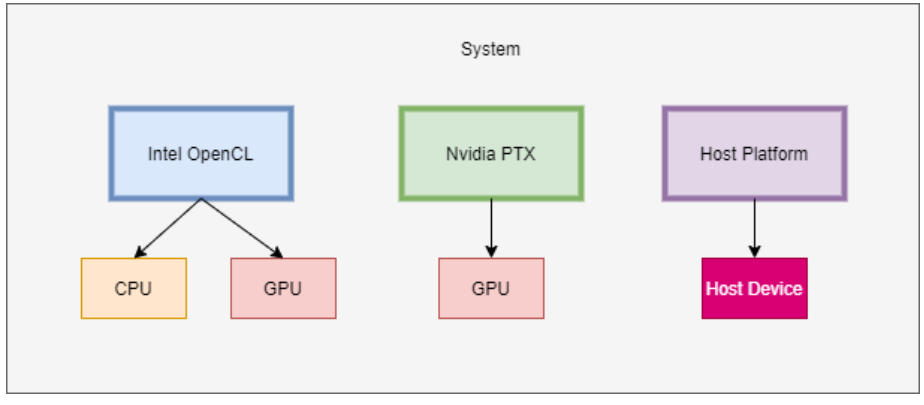


QUERYING THE TOPOLOGY

- 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 MANUALLY

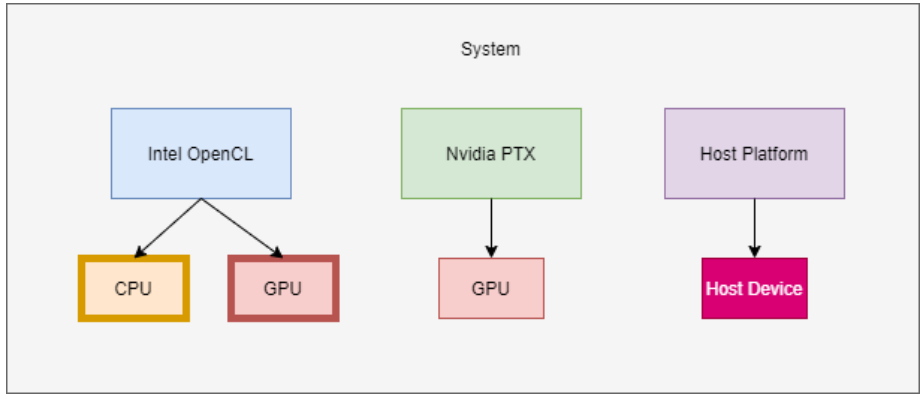
```
auto platforms = platform::get_platforms();
```



- 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.

QUERYING MANUALLY

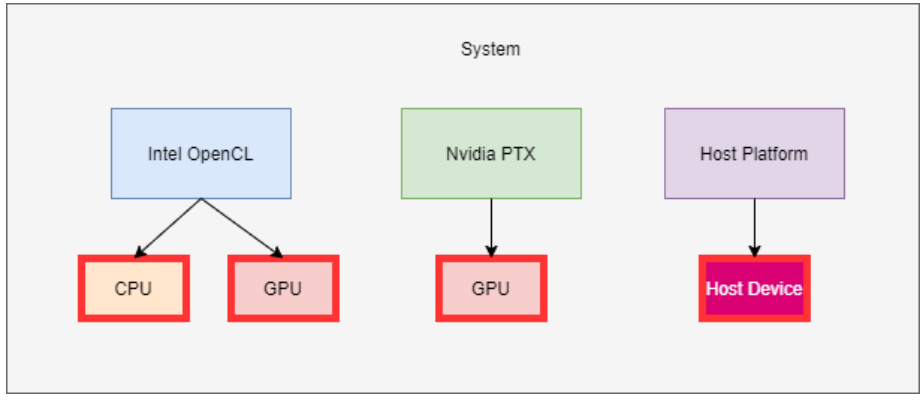
```
auto intelDevices = intelPlatform.get_devices();
```



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

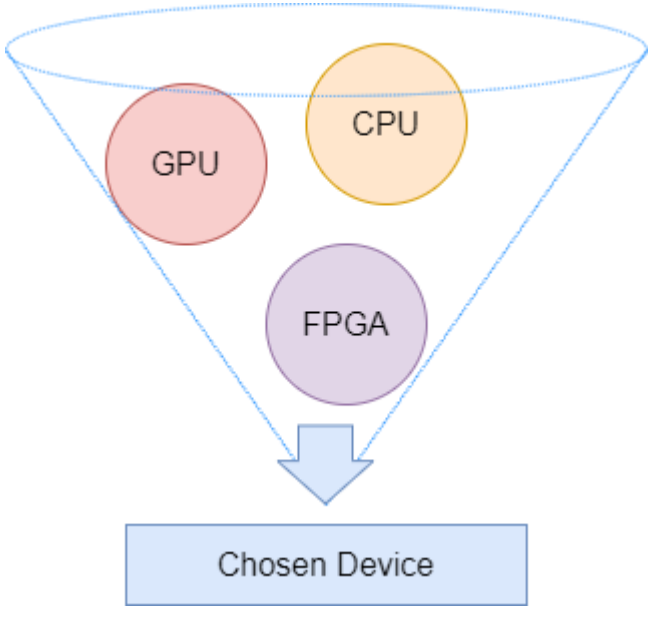
QUERYING MANUALLY

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



- The device class also provides the static function `get_devices` that returns a vector of all available devices in the system.
- This includes the host device.

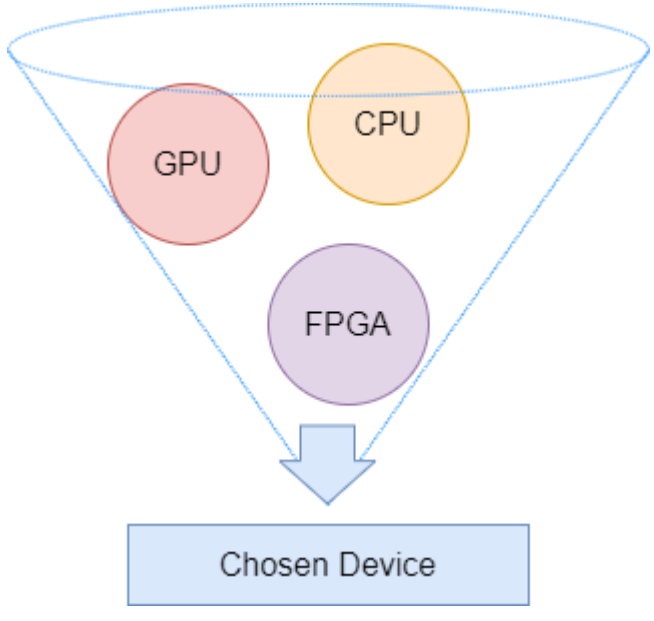
QUERYING WITH A DEVICE SELECTOR



- To simplify the process of traversing the system topology SYCL provides device selectors.
- A device selector is is a C++ function object, derived from the `device_selector` class, which defines a heuristic for scoring devices.
- SYCL provides a number of standard device selectors, e.g. `default_selector_v`, `gpu_selector_v`, etc.
- Users can also create their own device selectors.

QUERYING WITH A DEVICE SELECTOR

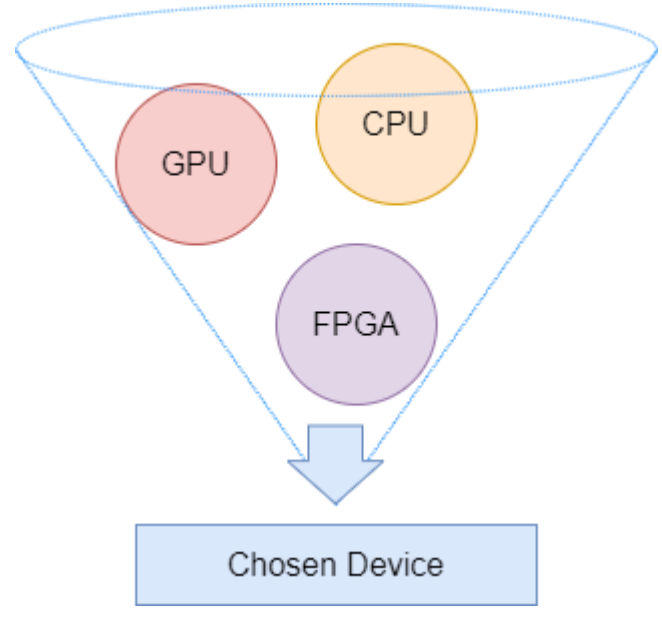
```
auto gpuDevice = gpu_selector_v.select_device();
```



- 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.

QUERYING THE TOPOLOGY USING A DEVICE SELECTOR

```
auto chosenDevice = default_selector_v.select_device
```



- The `default_selector_v` is a standard device selector type.
- Chooses a device based on an implementation defined heuristic.

CREATING A CUSTOM DEVICE SELECTOR

```
struct my_gpu_selector : public device_selector {

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

  }
};
```

- A device selector must inherit from the `device_selector` class.
  - In SYCL 2020 it can be any callable object.
- A device selector must have a function call operator which takes a reference to a device.

CREATING A CUSTOM DEVICE SELECTOR

```
struct my_gpu_selector : public device_selector {

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

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

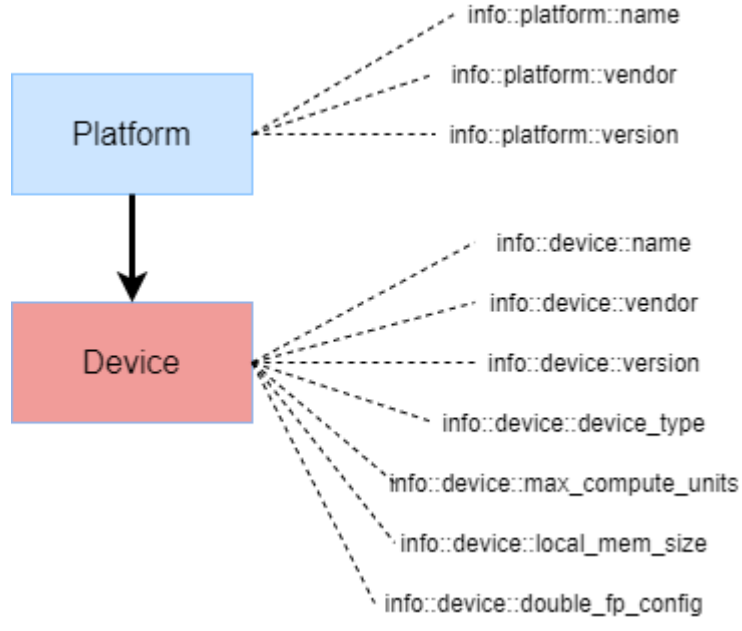
CREATING A CUSTOM DEVICE SELECTOR

```
struct my_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{my_gpu_selector};

}
```

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

PLATFORM/DEVICE INFO

```
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_platform` member function.

ASPECTS

```
bool supportsFp16 = dev.has(aspect::fp16);
```

- Capabilities of a device or platform are represented by aspects.
- These can be queried via the has member function.

QUESTIONS

EXERCISE

Code_Exercises/Exercise_5_Device_Selection/source

Create your own device selector that chooses the device in your system that you would like to target.