SYCL Academy

HANDLING ERRORS AND DEBUGGING

NHR SW

LEARNING OBJECTIVES

- Learn about how SYCL handles errors
- Learn about the difference between synchronous and asynchronous exceptions
- Learn how to handle exceptions and retrieve further information
- Learn about the host device and how to use it

SYCL EXCEPTIONS

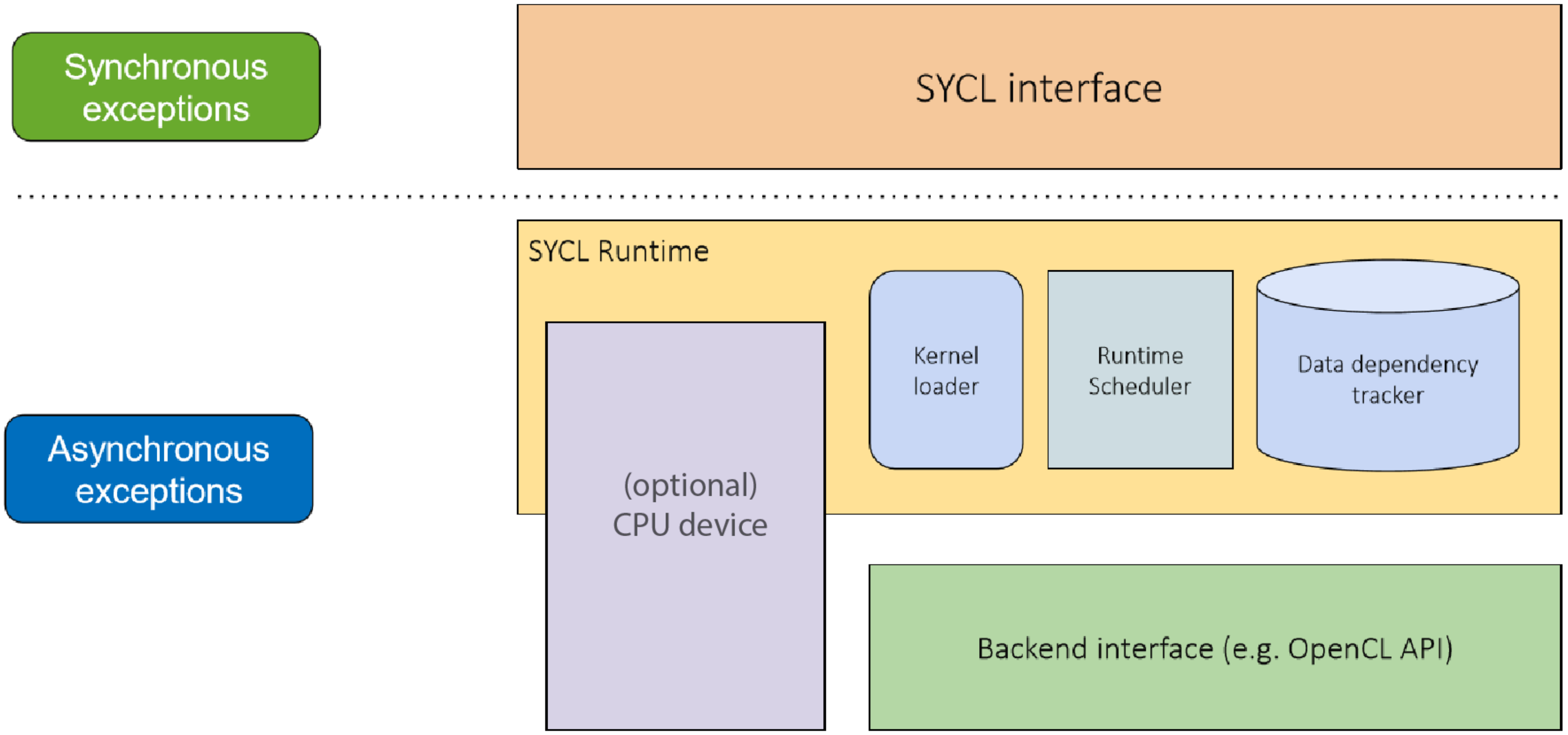
- In SYCL errors are handled by throwing exceptions.
- It is crucial that these errors are handled, otherwise your application could fail in unpredictable ways.
- In SYCL there are two kinds of error:
    - Synchronous errors (thrown in user thread) .
    - Asynchronous errors (thrown by the SYCL scheduler).

HANDLING ERRORS

```
int main() {
  queue q();

  /* Synchronous code */

  q.parallel_for(buf0.get_range(), [=](id<1> i) {

      /* Asynchronous code */

    });
  });
}
```

- Kernels run asynchronously on the device, and will throw asynchronous errors.
- Everything else runs synchronously on the host, and will throw synchronous errors.

SYCL EXCEPTIONS



HANDLING ERRORS

```
int main() {
  queue q();

  /* Synchronous code */

  q.submit([&](handler &cgh) {
    /* Synchronous code */

    cgh.single_task([=](id<1> i) {
      /* Asynchronous code */
    });
  }).wait();
}
```

- Code on the device runs asynchronously
- If errors are not handled, the application can fail:
  - SYCL 1.2.1 application will fail silently.
  - SYCL 2020 provides a default async handler that will call `std::terminate` when an asynchronous error is thrown.

```
int main() {
  std::vector<float> dA{ 7, 5, 16, 8 }, dB{ 8, 16, 5, 7 }, dO{ 0, 0, 0, 0 };
  try {
    queue gpuQueue(gpu_selector{});

    buffer bufA{dA};
    buffer bufB{dB};
    buffer bufO{dO};

    gpuQueue.submit([&](handler &cgh) {
      auto inA = accessor{bufA, cgh, read_only};
      auto inB = accessor{bufB, cgh, read_only};
      auto out = accessor{bufO, cgh, write_only};

      cgh.parallel_for(bufO.get_range(), [=](id<1> i) {
        out[i] = inA[i] + inB[i];
      });
    }).wait();

  } catch (...) { /* handle errors */ }
}
```

- Synchronous errors are typically thrown by SYCL API functions.

to handle all SYCL errors you must wrap everything in a try-catch block.

```cpp
int main() {
  std::vector<float> dA{ 7, 5, 16, 8 }, dB{ 8, 16, 5, 7 }, dO{ 0, 0, 0, 0 };
  try{
    queue gpuQueue(gpu_selector{}, async_handler{});
    buffer bufA{dA};
    buffer bufB{dB};
    buffer bufO{dO};

    gpuQueue.submit([&](handler &cgh) {
      auto inA = accessor{bufA, cgh, read_only};
      auto inB = accessor{bufB, cgh, read_only};
      auto out = accessor{bufO, cgh, write_only};

      cgh.parallel_for(bufO.get_range(), [=](id<1> i) {
        out[i] = inA[i] + inB[i];
      });
    }).wait();

    gpuQueue.throw_asynchronous();
  } catch (...) { /* handle errors */
}
```

- Asynchronous errors errors that may have occurred will be thrown after a nd group has been submitted to a queue.

```
int main() {
  std::vector<float> dA{ 7, 5, 16, 8 }, dB{ 8, 16, 5, 7 }, dO{ 0, 0, 0, 0 };
  try{
    queue gpuQueue(gpu_selector{}, [=](exception_list eL) {
      for (auto e : eL) { std::rethrow_exception(e); }
    });

    buffer bufA{dA};
    buffer bufB{dB};
    buffer bufO{dO};

    gpuQueue.submit([&](handler &cgh) {
      auto inA = accessor{bufA, cgh, read_only};
      auto inB = accessor{bufB, cgh, read_only};
      auto out = accessor{bufO, cgh, write_only};

      cgh.parallel_for(bufO.get_range(), [=](id<1> i) {
        out[i] = inA[i] + inB[i];
      });
    }).wait();

    gpuQueue.throw_asynchronous();
  } catch (...) { /* handle errors */ }
}
```

NHR SW

The async handler is a C++ lambda or function object that takes as a parameter

SYCL Academy

```
int main() {
  std::vector<float> dA{ 7, 5, 16, 8 }, dB{ 8, 16, 5, 7 }, dO{ 0,
0, 0, 0 };
  try {
    queue gpuQueue(gpu_selector{}, [=](exception_list eL) {
      for (auto e : eL) { std::rethrow_exception(e); }
    });

    ...

    gpuQueue.throw_asynchronous();
  } catch (const std::exception& e) {
    std::cout << "Exception caught: " << e.what()
     << std::endl;
  }
}
```

- Once rethrown and caught, a SYCL exception can provide information about the error
- The `what` member function will return a string with more details

NHR SW

SYCL Academy

EXCEPTION TYPES

- In SYCL 1.2.1 there are a number of different exception types that inherit from `std::exception`
  - E.g. `runtime_error`, `kernel_error`
- SYCL 2020 only has a single `sycl::exception` type which provides different error codes
  - E.g. `errc::runtime`, `errc::kernel`

SYCL Academy

DEBUGGING SYCL KERNEL FUNCTIONS

NHR
SW

- SYCL requires: a device must always be available
- → Implementations provide a CPU device
  - In AdaptiveCpp, this is the OpenMP backend
  - ⇒ Debug SYCL kernels: use the CPU device + a standard C++ debugger (gdb/lldb)

- In general, users can query the `host_debuggable` device aspect to check whether they can use the same functionality

```
int main() {
  std::vector<float> dA{ 7, 5, 16, 8 }, dB{ 8, 16, 5, 7 }, dO{ 0, 0, 0, 0 };
  try{
    queue hostQueue(aspect_selector<aspect::host_debuggable>(), async_handler{});

    buffer bufA{dA};
    buffer bufB{dB};
    buffer bufO{dO};

    hostQueue.submit([&](handler &cgh) {
      auto inA = accessor{bufA, cgh, read_only};
      auto inB = accessor{bufB, cgh, read_only};
      auto out = accessor{bufO, cgh, write_only};

      cgh.parallel_for(bufO.get_range(), [=](id<1> i) {
        out[i] = inA[i] + inB[i];
      });
    });
    hostQueue.wait_and_throw();
  } catch (...) { /* handle errors */ }
}
```

- Any SYCL application can be debugged on the host device by switching the queue for a host queue

SYCL Academy

QUESTIONS

EXERCISE

Code_Exercises/Exercise_4_Section_4_Handling_Errors/source

Add error handling to a SYCL application for both synchronous and asynchronous errors.