The Absurdity of Error Handling:

Finding a Purpose for Errors in Safety-Critical SYCL

ERIK TOMUSK

codeplay

Enabling AI & HPC To Be Open, Safe & Accessible To All

Established 2002 in **Edinburgh, Scotland**.

Grown successfully to around 100 employees.

In 2022, we became a **wholly owned subsidiary** of Intel.

Committed to expanding the **open ecosystem** for heterogeneous computing.

Through our involvement in oneAPI and SYCL governance, we help to **maintain and develop** open standards.

Developing at the forefront of **cutting-edge research**.

Currently involved in two research projects - **SYCLOPS** and **AERO**, both funded by the Horizon Europe Project.

- SYCL is an abstraction layer for running C++ code on accelerators like GPUs
- *SC* stands for *safety-critical*, **not** *supercomputing*
  - SC is any domain where software can cause substantial harm
- More on SYCL in room **Cottonwood 2/3**:

Thursday, October 5 • 16:45 - 17:45

Khronos APIs for Heterogeneous Compute and Safety: SYCL and SYCL SC

Disclaimer Slide

This is a personal presentation. Any views, opinions, or statements represented in this presentation are personal and belong solely to the presentation owner and do not represent those of people, institutions, or organizations that the owner may or may not be associated with in a professional or personal capacity, unless explicitly stated

Outline

- Definition of *Safety*
- Definition of *Error Handling*
- Case Study
- Why is this Important?
- Is it Really so bad?
- What does this mean for SYCL SC?
- What does this mean for you?

Definition of *Safety*

Definition of *Safety*

- *Safety* is heavily overloaded

Definition of *Safety* — The Experts

Software Language Designer

I design programming languages and libraries

Functional Safety Practitioner

I write safety-critical software

Definition of *Safety* — The Priorities

Software Language Designer

- Type safety
- Memory safety
- Easy to use correctly; hard to use incorrectly

Functional Safety Practitioner

- Language safety is nice, but …
- I want **determinism** — does the same thing happen in the same way every time?

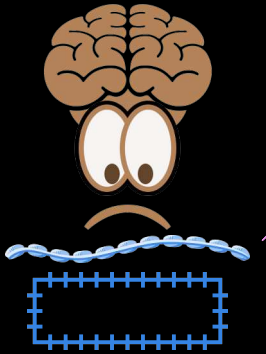
Definition of *Safety* — Abstractions

Assembly language is difficult for humans to reason about

Assembly language requires constantly keeping track of lots of detail

Hardware: Easy to use incorrectly; hard to use correctly

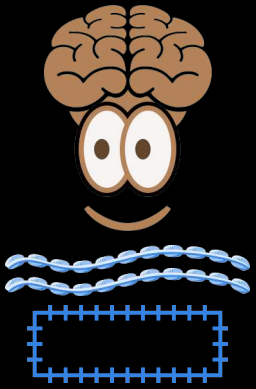
Definition of *Safety* — Abstractions



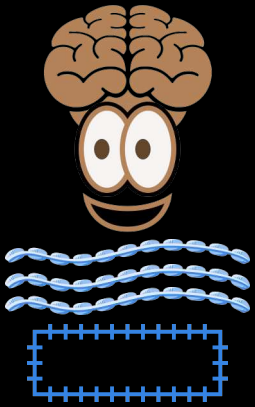
Abstractions separate humans from hardware complexity

Abstractions create **abstract machines** that are easier to reason about than real hardware

Definition of *Safety* — Abstractions



Definition of *Safety* — Abstractions



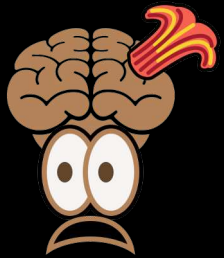
I work hard to design abstractions

Abstractions = language features & guarantees

Definition of *Safety* — The Primary Concern

Software Language Designer

What if the abstract machine breaks?



Functional Safety Practitioner

What if the real machine breaks?

Definition of *Safety* — The Solutions

Software Language Designer

I will put bounds checking on `operator[]`

Undefined behavior is the worst thing ever

I will add abstractions to make C++ less bug-prone

Functional Safety Practitioner

What if bounds checking has unpredictable execution time?

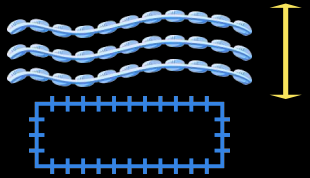
Undefined behavior isn't so bad as long as it does the same thing every time

These abstractions have unpredictable worst-case execution time

Definition of *Safety* — Abstractions

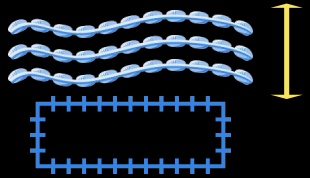
Software Language Designer

Separation from hardware is good



Functional Safety Practitioner

Separation from hardware is a potential liability



Definition of *Safety* — Summary

- Language designers focus on language features
- Functional safety concerned with entire system
- C++ exceptions are archetypical example
  - **Language Design:** Interact well with RAII, object invariants, and abstract machine
  - **Functional Safety:** Unbounded execution time (in mainstream compilers)

Definition of *Error Handling*

Definition of *Error Handling*

Error:
An unintended occurrence

- Various means to communicate the presence of an error
  - Error codes; C++ exceptions; `std::unexpected<E>`

Error Handling:
Returning a software component from an unintended state back into an intended state

- Exceptions and error codes are sometimes used for reporting or information logging
  - Important use cases, but out of scope
- Exceptions and error codes are sometimes used to report non-error information
  - Expedient, and out of scope

Case Study

Case Study: A Trivial Example

- Out-of-bounds write — very bad

Let's add bounds checking to `operator[]`

`operator[]` will throw `std::out_of_range` and [illegible] strong exception guarantee

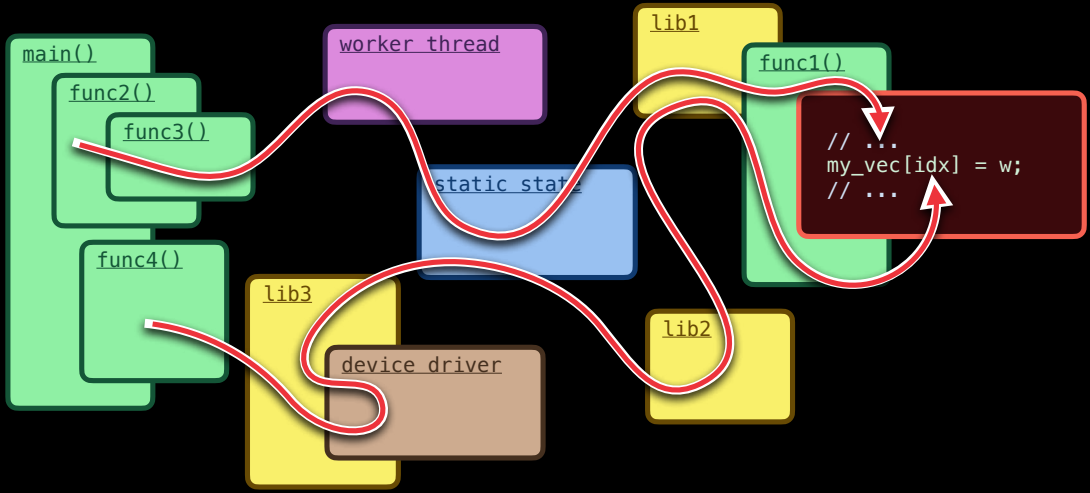

```
// ...
my_vec[idx] = w;
// ...
```

`my_vec.size()` is 3

`idx` is 5

This doesn't help me

Case Study: A Trivial Example



Case Study: A Trivial Example

This is undefined behavior

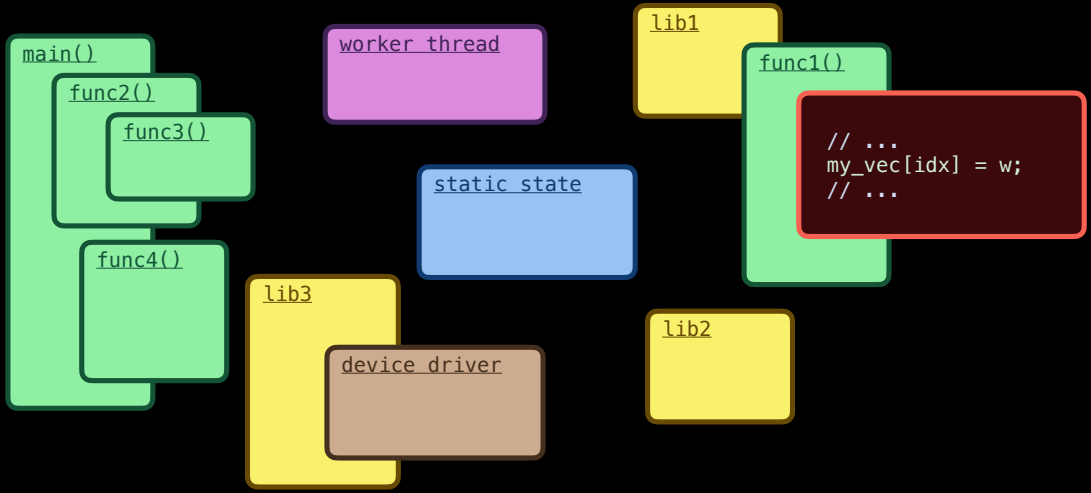
I want to remove undefined behavior

```
// ...
my_vec[idx] = w;
// ...
```

`my_vec.size()` is 3

`idx` is 5

Case Study: A Trivial Example



Case Study: A Trivial Example

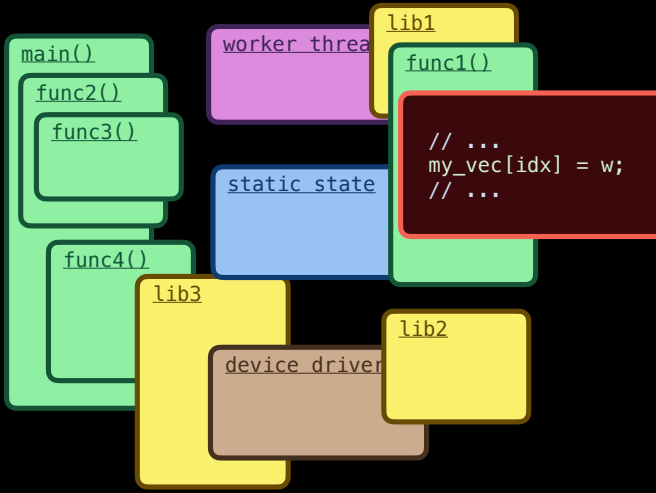
This whole thing is in an inconsistent state

I don't know what effect this will have in the physical world

I don't know where the inconsistency originates

Or how many components are affected

Or what needs to be done to **return all affected components to consistent states** so the application can continue

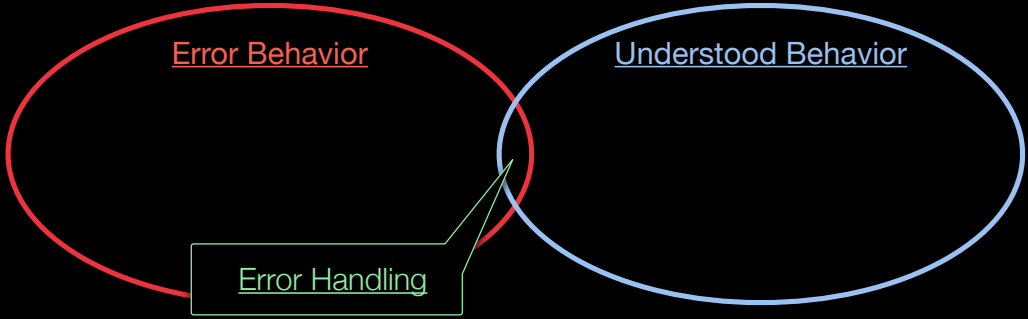


Case Study Summary

- Language design is laser-focused on UB
- Functional safety more concerned with the correct behavior of the entire system

Generalizing the Case Study

- *Error handling* implies that there are situations that are so unusual as to be called “errors” …
- … but are so well-defined, predictable, and well-understood as to allow application-level recovery
- **Absurdity of error handling:** The set of errors that can be “handled” is (nearly) empty
- Is error handling needed?



Why is this Important?

Why is this Important for SYCL SC?

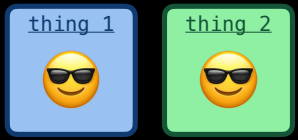
- Safety-critical systems must be resilient in unanticipated situations
- It's often assumed that error handling inside applications is a good way to provides resiliency
  - This needs to be questioned
- Common error handling examples are all either "not really an error" or "not recoverable"
  - If you have an example, please get in touch
- With the error handling API in SYCL SC, are we designing a feature that …
  - Everyone thinks they want, but …
  - There is no way to use it correctly, and …
  - No one really understands it, and …
  - It leads to unnecessary complexity
  - ?

Why is this Important Outside Safety-Critical?

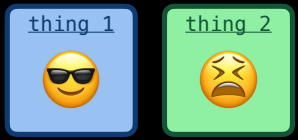
- Role of error handling in (non- safety-critical) C++ similarly unclear
  - What are situations where an application can reliably recover from an error?
- Is error handling just a get-out-of-jail-free card?
  - You've coded yourself into a corner, so you use errors to get out of it
- Futility of error handling: If you can't write correct code, why do you think you can write code to recover from your mistakes?
- Liability: Error handling introduces lots of complexity of questionable value

Is it Really so bad?

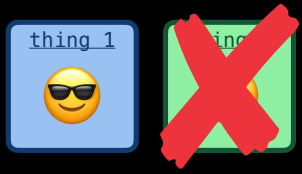
When does Error Handling make Sense?



When does Error Handling make Sense?



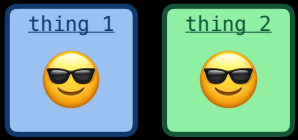
When does Error Handling make Sense?



When does Error Handling make Sense?

When does Error Handling make Sense?

- When one component can be reset without affecting others
- Components could be
  - Different chips
  - Different hypervisors on a CPU
  - Different processes
  - (Perhaps even) `.so` libraries that can be unloaded/reloaded
- C++ objects, C++ threads, etc. too intertwined with application state to be resettable
- *Application envelope*
- *Unit of mitigation*
- Language-level error handling mechanisms might not be appropriate tools for error handling
  - Language boundaries line up with units of mitigation
  - E.g., `bad_alloc`



What does this mean for SYCL SC?

What does this mean for SYCL SC?

- Active area of discussion
  - Exploring things that can fail and how system could recover
- Mine is just one perspective; there are others
- What SYCL SC ends up doing with errors will be influenced by
  - Vulkan SC
  - MISRA
  - SYCL
  - C++
- What SYCL SC ends up doing will likely be evolutionary, not revolutionary
- Some users will use it to good effect; others less so
- Whatever we end up doing, I will probably complain about it
  - But that's OK

What does this mean for you?

If you are an Application Developer

- Be deliberate about what you're doing with errors
- When a library gives you an error code or C++ exception, what are you meant to do with it?
  - Has the library developer told you?
  - Don't assume your application can keep using the library
    - Unless the library has explicitly documented error/exception safety guarantees
  - Is the library using error codes or exceptions to communicate warnings and other info?
    - Then your application might not be on an error path, but might be operating normally
- Easy to introduce lots of `try/catch` and `if()` in an attempt to handle errors
  - Increases application complexity & testing complexity (cost)
  - Are there benefits to your application?
    - Is the extra logic helping to fulfill requirements, or is it just masking bugs?
  - Could you just print an error message and exit?

If you are a Library Developer

- When you report an error code or throw an exception, what is your user meant to do?
  - Is it actually not an error, but just information for the user?
    - Could you communicate this without using an error path?
  - What assumptions can your user make about the state of the library? Is it usable?
  - Have you documented this information?

If you are a C++ Language Designer

- Can we talk about determinism in C++?

- But also …
- C++ exceptions are a monolithic sledgehammer
  - There are no error management primitives that could be used to create domain-appropriate error handling
- C++ exceptions are tightly coupled with object orientation
  - Difficult to mix with generic programming, static state, threads, IPC, external state, etc.
- It can be expedient to use C++ exceptions to communicate non-error information
  - Is there a need for more ways of communicating information between components?

What is “error handling” meant to accomplish?

Erik Tomusk
erik@codeplay.com