Wasm Exception Handling

Current State = **Phase 1**

Involved Parties

- Champion: Heejin Ahn (Google)
- Google WebAssembly Toolchain TLM: Derek Schuff
- Google V8 Implementation: Michael Starzinger

Proposal Goals

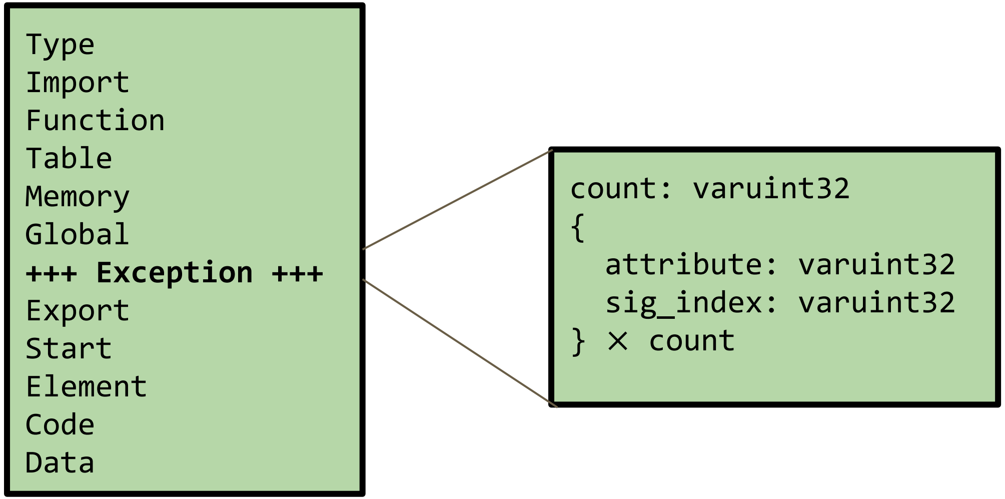
- Provide a primitive for exception handling for Wasm programs
    - Zero-cost: no runtime cost until exceptions are thrown or caught
    - Structured: composes properly with existing control flow constructs
    - Safe: fast, single-pass verification
- Low-cost C++ exception handling
    - Emscripten currently indirects through JS for try/catch (not production quality)
    - WASM solution should be nearly as efficient as a native target
- Low-impedance for other languages
- Preserve forward compatibility for effect handlers

Why this is hard

- One of the first proposals were multi-lingualism hits hard
  - C++ vs JavaScript vs Java compiled to WASM
  - What does it mean to catch an exception from JavaScript in WASM?
- Exceptions in JavaScript engines are host references, but WASM has no reference types (yet)
  - First proposal was to hide all exception values
  - Second proposal allows them as new value type, but still questions about lifetime
- LLVM intermediate representation of exception edges
  - C++ ctor/dtors impl a try...finally around all constructors
  - Leads to a lot of chained control flow for cleanup, can also be shared by non-excepting path ⇒ **`rethrow`** within a function to accomplish proper chaining of cleanup

Exception signatures

- A new module section is proposed to declare exceptions.



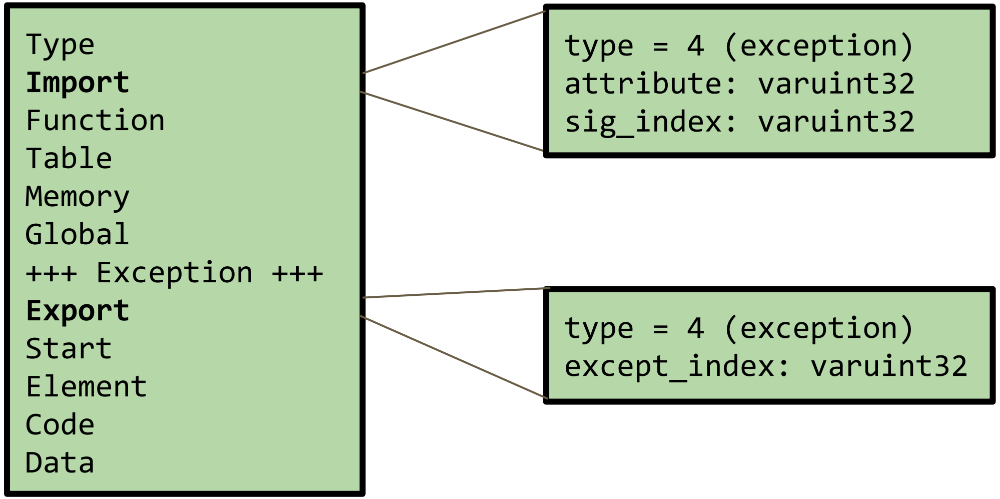
```
Type
Import
Function
Table
Memory
Global
+++ Exception +++
Export
Start
Element
Code
Data
```

```
count: varuint32
{
  attribute: varuint32
  sig_index: varuint32
} × count
```

- Exception section declares a list of exception types
- `attribute`: application-specified number
- `sig_index`: index into types section of function signature

Exception import/export

- Exceptions can be *imported* and *exported* from a module.
- Each instantiation of a module produces *new* exception tags.



Type
Import
Function
Table
Memory
Global
+++ Exception +++
Export
Start
Element
Code
Data

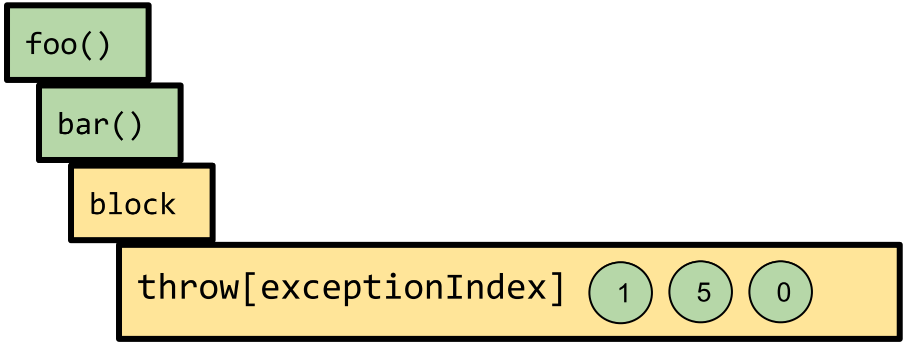
```
type = 4 (exception)
attribute: varuint32
sig_index: varuint32
```

```
type = 4 (exception)
except_index: varuint32
```

- Imported exceptions specific an expected signature
- Exported exceptions specify an exported exception index

The `throw` instruction

- A new instruction **`throw`** *creates* an exception and begins searching the *control* and *call* stacks for a handler.



- **`throw`** accepts its arguments on the stack and stores them into the exception. Like **`br`**, **`throw`** *ends* control.

The `try/catch` construct

- A new control flow structure **`try ... catch`** introduces a new block kind.
- Two flavors have been explored. **We must choose one**.

```
try[blockType]
  ...
catch[exceptionIndex]
  ...
catch[exceptionIndex]
  ...
catch_all
  ...
end
```

Flavor #1

```
try[blockType]
  ...
catch
  ...
end
```

Flavor #2

Flavor #1 `try/catch`

```
try[blockType]
  …
catch[exceptionIndex]
  …
catch[exceptionIndex]
  …
catch_all
  …
end
```

match+catch

- Exception values ***are not*** first class.
- Multiple catch clauses per try.
- Each catch clause specifies an exception index and automatically *unpacks* the values from that exception onto the stack.
- A **`catch_all`** can specify a default case and catch host language exceptions.
- **`rethrow`** (covered later) is allowed in the **`catch`**[**`_all`**] blocks.

Flavor #2 - `try/catch`

```
try[blockType]
  ...
catch
  ...
end
```

catchall only

```
if_except[exceptionIndex]
  ...
else
  if_except[exceptionIndex]
    ...
  end
end
```

explicit pattern matching

- Exception values ***are*** first class.
- One catch clause per **`try`**.
- Pattern matching can be done manually with a new control flow construct, **`if_except`**.
- This allows factoring into helper routines and sharing within a function.

The `rethrow` instruction

- In either flavor, **`rethrow`** can be used to continue exception handling from the point of the rethrow.

```
try[blockType]
  …
catch_all
  rethrow[catch,depth]
end
```

Flavor #1

- **`rethrow`** legal in **`catch[_all]`**
- Rethrows an exception to a specific enclosing outer catch (or to caller).

```
rethrow x
```

Flavor #2

- **`rethrow`** can occur anywhere
- Accepts exception to rethrow on the top of the operand stack

Implementation Status

- V8 implemented Flavor #1 exceptions
    - Available with --wasm-prototype-eh
    - Passed all handwritten test cases so far
    - Built on optimizing JIT support for JavaScript exceptions
    - Happy to switch to Flavor #2 if community prefers this direction
- Toolchain implements Flavor #1 exceptions
    - Backend compilation part is mostly done
    - Object file format + linker support is in progress
    - libcxxabi / libunwind modification is needed
    - Needs to be tested with bigger programs

Next Steps

- High priority: decide if we should do first-class exceptions (Flavor #2)
    - Pro: more expressive
    - Pro: forward-compatible with effect handlers (needs validation)
    - Con: Potential lifetime issues for engines with no backing GC
    - Con: Potentially less efficient if engine has to stop at every catch (no filtering)