Stack switching for Wasm

Sam Lindley

The University of Edinburgh

12th February 2025

Motivation

Non-local control flow features are pervasive

- Async/await (e.g. C++, C#, Dart, JavaScript, Rust, Swift)
- Coroutines (e.g. C++, Kotlin, Python, Swift)
- Lightweight threads (e.g. Erlang, Go, Haskell, Java, Swift)
- Generators and iterators (e.g. C#, Dart, Haskell, JavaScript, Kotlin, Python)
- First-class continuations (e.g. Haskell, Java, OCaml, Scheme)

Motivation

Non-local control flow features are pervasive

- Async/await (e.g. C++, C#, Dart, JavaScript, Rust, Swift)
- Coroutines (e.g. C++, Kotlin, Python, Swift)
- Lightweight threads (e.g. Erlang, Go, Haskell, Java, Swift)
- Generators and iterators (e.g. C#, Dart, Haskell, JavaScript, Kotlin, Python)
- First-class continuations (e.g. Haskell, Java, OCaml, Scheme)

The stack switching instructions are sufficiently general to support all of these features

Current status

Moved to Phase 2 in August 2024 ✓

Reference interpreter implementation ✓

Wizard implementation ✓

Wasmtime implemention ✓

- PR submitted for upstreaming
- x64 & ARM64 backends

Binaryen support ✓

Wasm-tools support ✓

Formal specification ✓

- SpecTec
- WasmCert — fully mechanised soundness proof

Unified design

Founded on effect handlers — **asymmetric** switching
parent-child relationship between stacks

OOPSLA 2023 paper "Continuing WebAssembly with Effect Handlers"
(WasmFX / typed continuations)
https://arxiv.org/abs/2308.08347

Additional `switch` instruction to optimise performance of **symmetric** switching
(Bag of stacks)

Instruction set

Module-level definitions

- Control tags generalise exception tags
  `(tag $yield (param i32) (result i32))`
- Heap type for continuations
  `(type $ct (cont $ft))`

Instruction set

Module-level definitions

- Control tags generalise exception tags
  `(tag $yield (param i32) (result i32))`
- Heap type for continuations
  `(type $ct (cont $ft))`

Core instructions

- Create new suspended continuation (from function reference)
  `(cont.new $ct)`
- Resume continuation under a handler
  `(resume $ct (on $yield $handler_block))`
- Suspend with tag up to nearest handler
  `(suspend $yield)`
- Switch directly to target continuation
  `(switch $ct $yield)`

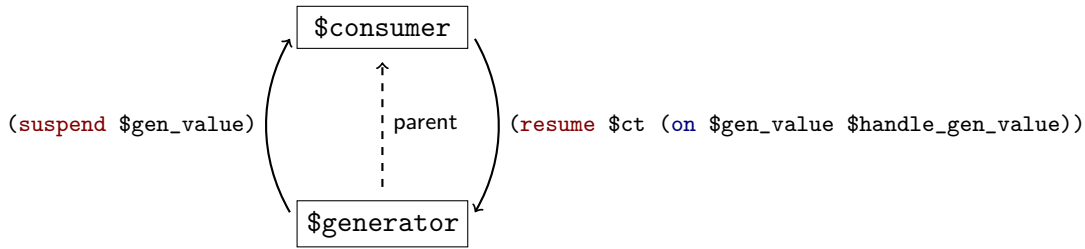
Instruction set

Additional instructions

- Cancel a continuation by raising an exception
  `(resume_throw $ct $exn)`
- Partially apply a continuation
  `(cont.bind $ct1 $ct2)`

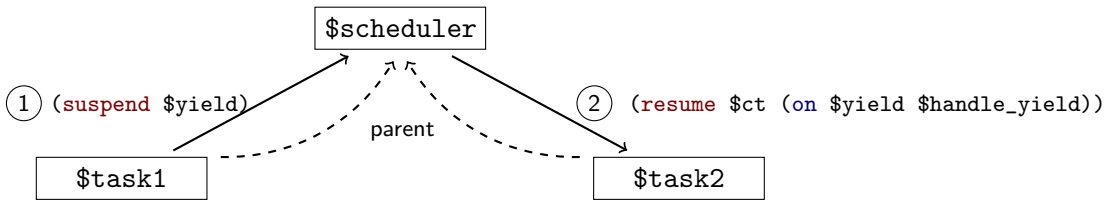
Example: asymmetric switching

Generator and consumer



Motivation for symmetric switching — scheduling lightweight threads

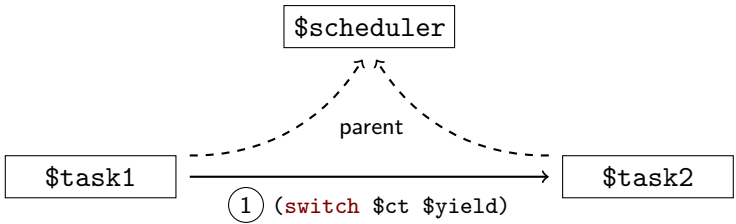
Asymmetric scheduler



Task switching takes two stack switches

Example: symmetric switching

Symmetric scheduler



Task switching takes a single stack switch

Current status

Moved to Phase 2 in August 2024 ✓

Reference interpreter implementation ✓

Wizard implementation ✓

Wasmtime implemention ✓

- PR submitted for upstreaming
- x64 & ARM64 backends

Binaryen support ✓

Wasm-tools support ✓

Formal specification ✓

- SpecTec
- WasmCert — fully mechanised soundness proof

Next steps

Phase 3 vote after PR is upstreamed to Wasmtime

Work with producers to target stack switching instructions

Browser implementations (can adapt existing JSPI infrastructure)

Post MVP: experiment with named handlers variation

Resources

Stack switching proposal (explainer, examples, spec, reference interpreter)
(https://github.com/WebAssembly/stack-switching)

Wizard implementation
(https://github.com/titzer/wizard-engine)

Wasmtime implementation
(https://github.com/bytecodealliance/wasmtime/pull/10177)

Binaryen implementation
(https://github.com/WebAssembly/binaryen)

Wasm-tools implementation
(https://github.com/bytecodealliance/wasm-tools)

OOPSLA 2023 paper "Continuing WebAssembly with Effect Handlers"
(https://arxiv.org/abs/2308.08347)