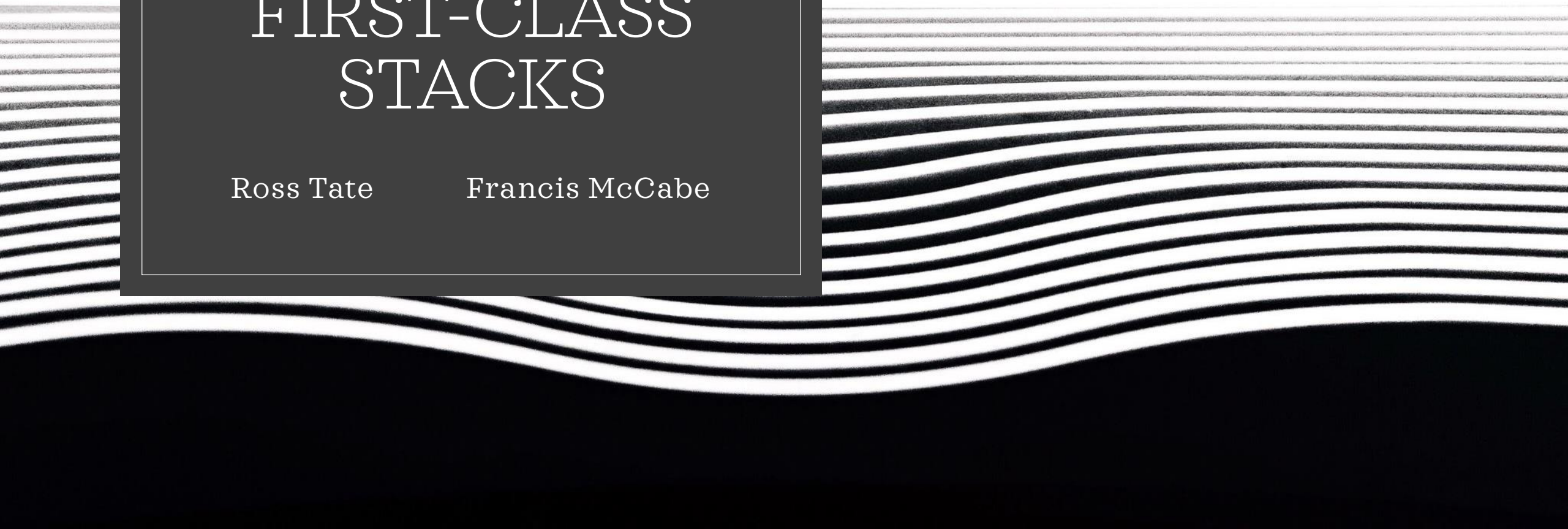
FIRST-CLASS STACKS

Ross Tate Francis McCabe



Applications

- Async/await
- Concurrency (as opposed to parallelism)
  - Lightweight threads
  - Communicating processes
- Continuations (including delimited ones)

Key Components

Values

- Stacks

Operations

- Stack switching
- Stack building
- Stack-walk redirection
- Stack inspection
  - (separable)

Types

- stackref
  - a (unique) reference to a stack that is in a suspended state
    - awaiting an event encoded as an exception (from the EH proposal)
    - can be awaiting multiple kinds of events
    - handlers can be statically known and dynamically determined
  - no type parameters
    - more flexible stack-switching patterns
    - difference in performance seems likely miniscule

Stack Switching

- stack.switch $exn
  - [t* stackref] -> unreachable
  - where (exception $exn t* stackref)
- (stack.switch $exn arg* s)
  - switches control to *resuming* stack s
  - payload of event is arg* and *yielding* stack
  - leaves yielding stack in suspended state
  - throws resuming event as exception

```
(event $yielding (param stackref))
(event $resuming (param))

(global $manager stackref)

(func $yield_current_threadlet

  (block $resumed
    (try

      (stack.switch $yielding
                    (global.get $manager))

    catch $resuming  $resumed)
  ) ;; $resumed : []

)
```

Advanced Stack Switching

- stack.switch_call $f $exn?
  - [ti* stackref] -> unreachable
  - where func $f : [ti* stackref] -> [to*]
  - and event $exn : [to*] (if provided)
- (stack.switch_call $f $exn? arg* s)
  - switches control to resuming stack s
  - calls $f on s with arg* and yielding stack
  - upon $f returning with val*
    - if $exn is specified, throws $exn exception on s with val* as payload
    - otherwise, traps

```
(func $save_manager (param $s stackref)
  (global.set $manager (local.get $s))
)

(func $resume_threadlet_from_manager
        (param $s stackref) (result stackref)
  (block $yielded
    (try
      (stack.switch_call $save_manager
                         $resuming
                         (local.get $s))
    catch $yielding $yielded)
  ) ;; $yielded : [stackref]
)
```

ASYNC/AWAIT (SINGLE STACK)

Pedagogical Example

Program Setup

(func $main (param $input i32) (result i32) …)

- calls $await whenever it needs to wait for promises

(func $await (param $promise externref) (result externref) …)

- To be defined shortly

(func (export “main_async”) (param $input i32) (result externref) …)

- Returns the i32 value (as an externref) or a promise for the value
- To be defined shortly

High-Level Program Architecture

(global $host_stack stackref)

- stores the stack of whoever called into the module

(global $main_stack stackref)

- stores the module's internal stack

(event $finishing (param i32))

- used to pass the final result from the main stack

(event $awaiting (param externref))

- used to suspend the main stack to wait for the specified promise

(event $resolving (param externref))

- used to resume the main stack with the resolution of a promise

$await

```
(func $await (param $promise externref) (result externref)
  (block $resolved
    (try
      (stack.switch_call $save_main_stack $awaiting
                         (local.get $promise) (global.get $host_stack))
    catch $resolving $resolved)
  ) ;; $resolved : [externref]
)
(func $save_main_stack (param $promise externref) (param $s stackref) (result externref)
  (global.set $main_stack (local.get $s))
  (local.get $promise)
)
```

“main_async”

```
(func (export “main_async”) (param $input i32) (result externref)
  (block $finished
    (block $awaited
      (try
        (stack.switch_call $main_root (local.get $input) (call $new_stack)) ;; $main_root defined later
      catch $awaiting $awaited
      catch $finishing $finished)
    ) ;; awaited : [externref]
    (return (call $await_promise)) ;; imported [externref] -> [externref]
  ) ;; $finished : [i32]
  (call $i32_to_externref) ;; imported [i32] -> [externref]
)
(import $new_stack (func (result stackref)))
```

$await_promise

(import $await_promise (func (param externref) (param externref)))

- (p) => p.then((x) => instance.resolve(x), (e) => instance.reject(e))

```
(func (export “resolve”) (param $resolution externref) (result externref)
  (block $finished
    (block $awaited
      (try
        stack.switch_call $save_host_stack $resolving (local.get $resolution) (global.get $main_stack)
      catch $awaiting $awaited
      catch $finishing $finished)
    ) ;; awaited : [externref]
    (return (call $await_promise)) ;; imported [externref] -> [externref]
  ) ;; $finished : [i32]
  (call $i32_to_externref) ;; imported [i32] -> [externref]
)
```

$main_root

```
(func $main_root (param $input i32) (param $s stackref)
  (global.set $host_stack (local.get $s))
  (local.set $input (call $main (local.get $input)))
  (stack.switch_call $drop $finishing
                     (local.get $input) (global.get $host_stack))
)
(func $drop (param $output i32) (param $s stackref) (result i32)
  (local.get $output)
)
```

$main_root

```
(func $main_root (param $input i32) (param $s stackref)
  (global.set $host_stack (local.get $s))
  (local.set $input (call $main (local.get $input)))
  (stack.switch_drop $finishing
                          (local.get $input) (global.get $host_stack))
)
(func $drop (param $output i32) (param $s stackref) (result i32)
  (local.get $output)
)
```

ASYNCHRONOUS PROGRAMMING!

Almost…

STACK-WALK REDIRECTION

Composability

$await_promise, part 2

(import $await_promise (func (param externref) (param externref)))
- (p) => p.then((x) => instance.resolve(x), (e) => instance.reject(e))

(func (export “reject”) (param $error externref) (result externref)

```
  (block $finished
    (block $awaited
      (try
        (stack.switch_call $save_host_stack $externexn (local.get $error) (global.get $main_stack))
      catch $awaiting $awaited
      catch $finishing $finished)
    ) ;; awaited : [externref]
    (return (call $await_promise)) ;; imported [externref] -> [externref]
  ) ;; $finished : [i32]
  (call $i32_to_externref) ;; imported [i32] -> [externref]
)
```

(import $externexn (event (param externref)))

$main_root, revised

```
(func $main_root (param $input i32) (param $s stackref)
  (global.set $host_stack (local.get $s))
  (stack.redirect_to
    (global.get $host_stack)
  then
    (global.set $host_stack)
  within
    (local.set $input (call $main (local.get $input)))
  )
  (stack.switch_drop $finishing
                          (local.get $input) (global.get $host_stack))
)
```

Redirects stack walk to the stackref in $host_stack

Restores $host_stack after the stack walk is done

COMPLEMENTARY FEATURES

Stack Extension and Stack Inspection

Complementary Features

Stack Extension

- Add call frames to stacks without switching to them
- Useful for initializing stacks
  - E.g. creating new threadlets
- Useful for building stacks
  - E.g. multi-shot continuations

Stack Inspection

- Search stack for marks that can inspect and modify frames on the stack
- Many uses outside of first-class stacks
- Useful for delimited continuations and deep/shallow algebraic effects
  - E.g. can implement alternative proposal