Typed Continuations

A Basis for Stack-switching in Wasm

Daniel Hillerström
daniel.hillerstrom@ed.ac.uk

Recall: Critical Use-cases

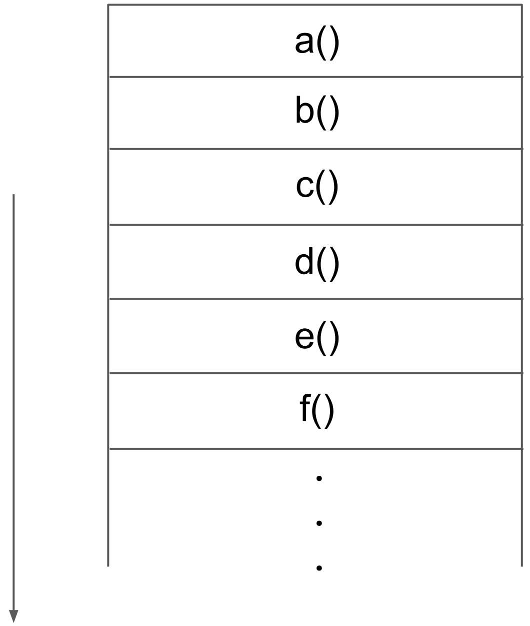
- Async/await
- Green/lightweight threads
- First-class continuations

(https://github.com/WebAssembly/stack-switching/blob/main/proposals/stack-switching/requirements.md)

Stacks & Continuations

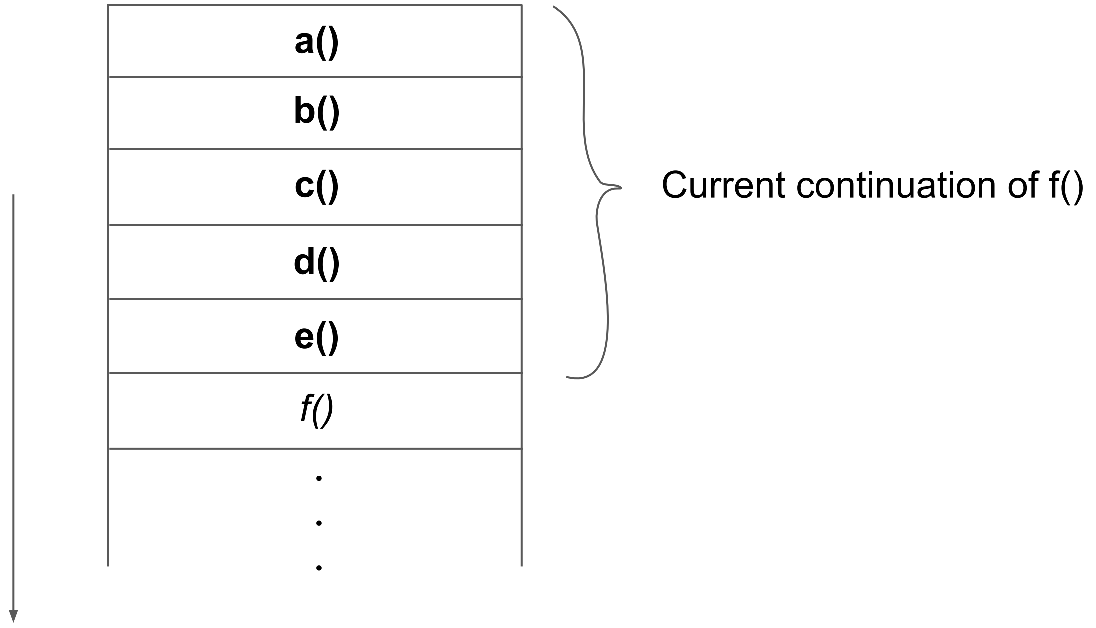
Operational Interpretation, Continuations ~ Stacks

A continuation represents the “remainder of computation”



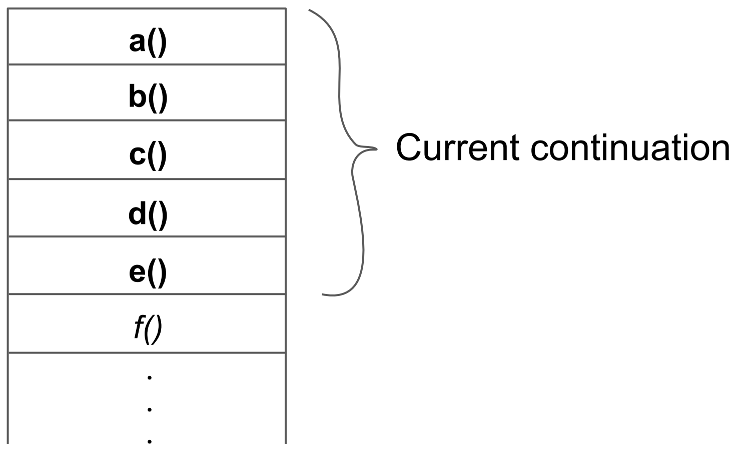
Operational Interpretation, Continuations ~ Stacks

A continuation represents the “remainder of computation”



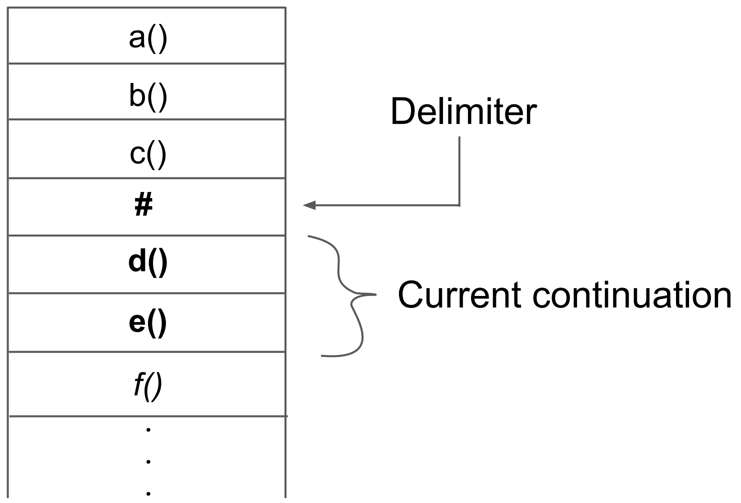
Undelimited vs Delimited Continuations

Undelimited continuations are *global*



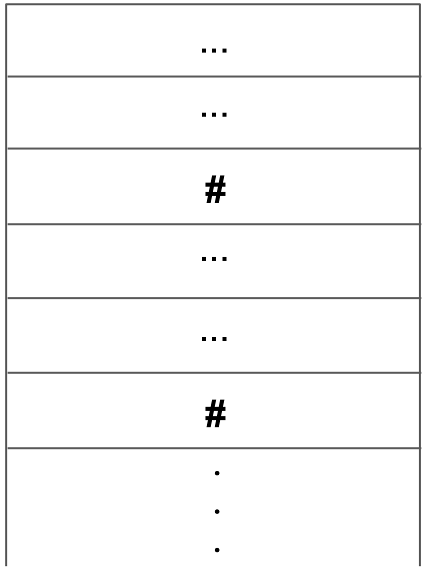
(anti-modular, non-compositional, inexpressive)

Delimited continuations are *local*

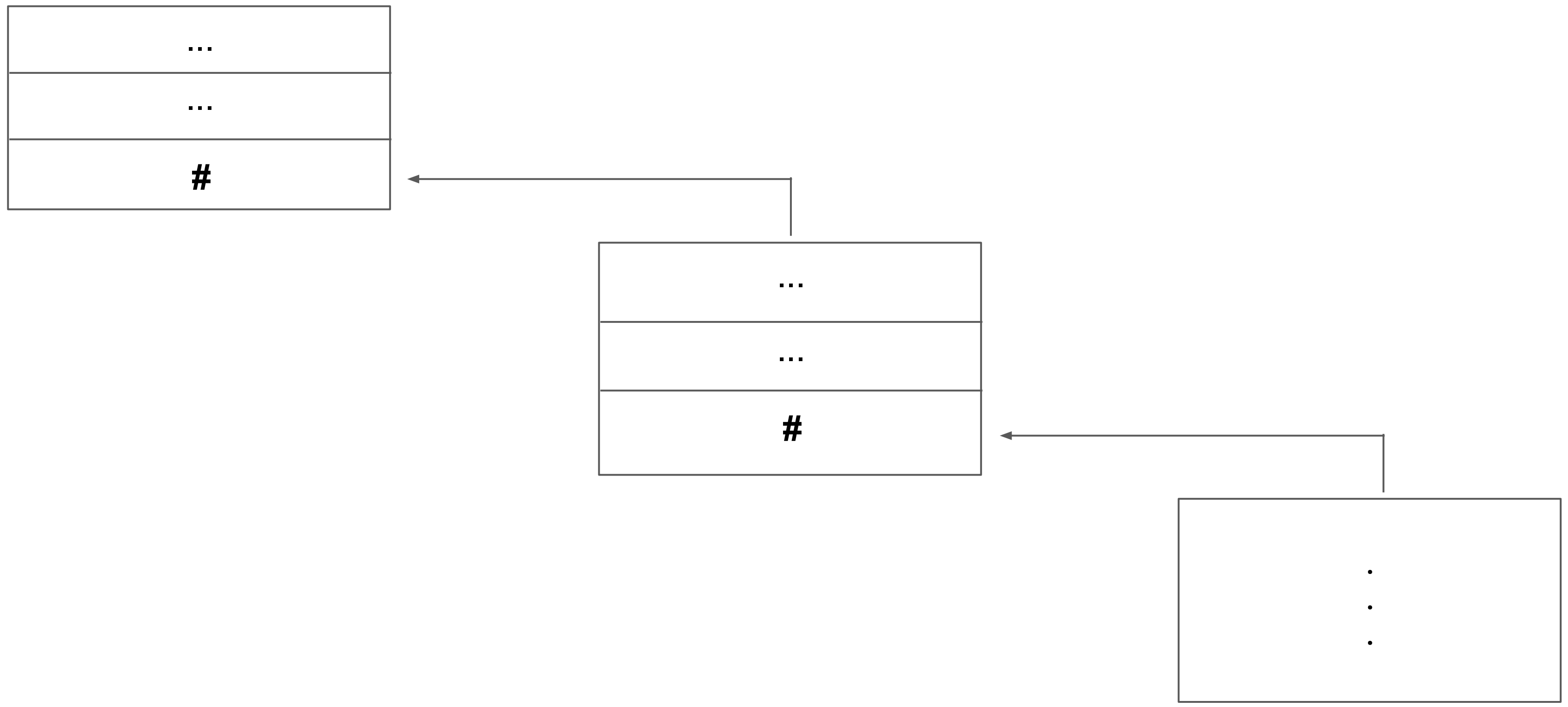


(modular, compositional, expressive)

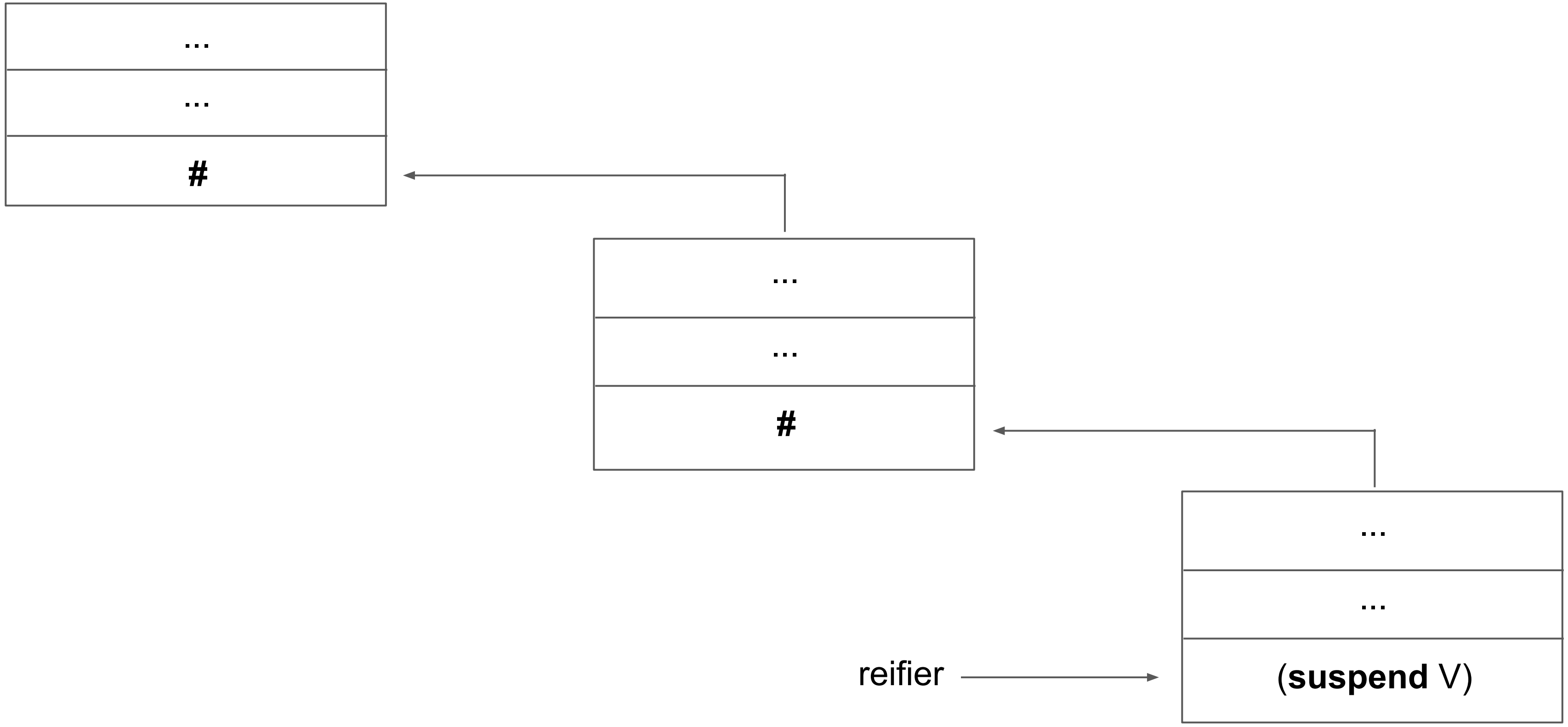
Delimited Continuations as Segmented Stacks [Bruggeman et al. ‘96]



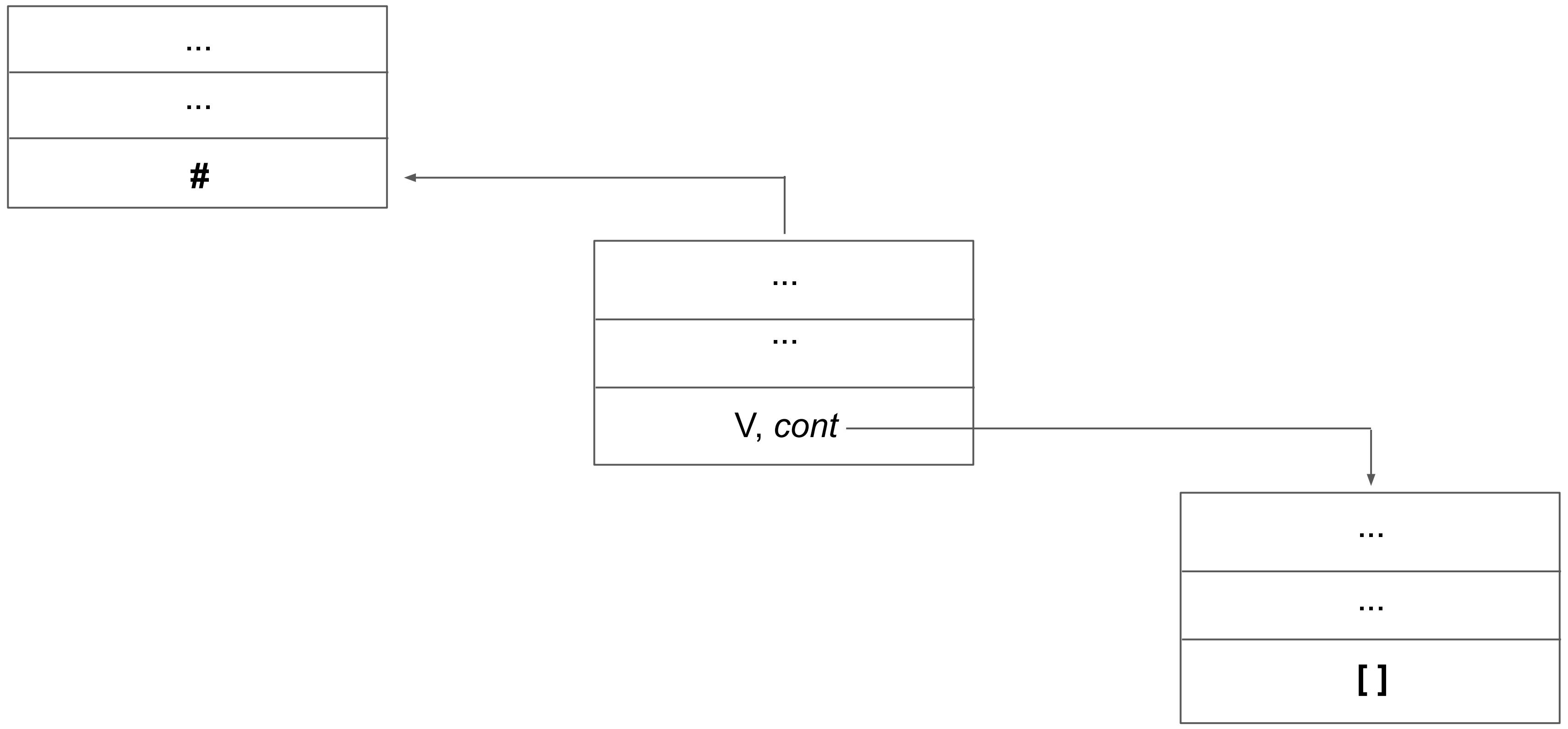
Delimited Continuations as Segmented Stacks [Bruggeman et al. '96]



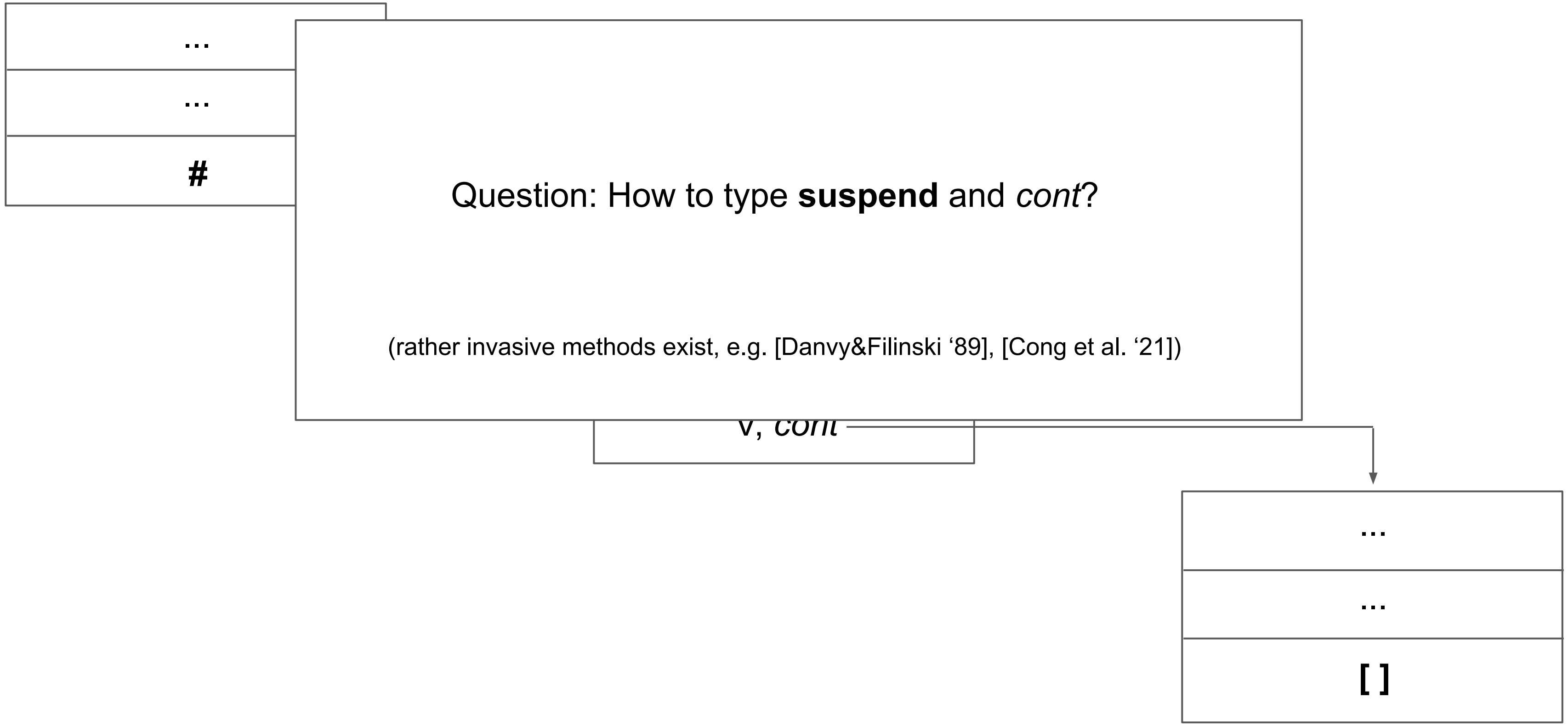
Stack Reification



Stack Reification



Stack Reification



Question: How to type **suspend** and *cont*?

(rather invasive methods exist, e.g. [Danvy&Filinski ‘89], [Cong et al. ‘21])

Typed Continuations at Glance

- Inspired by Pretnar & Plotkin’s effect handlers [Pretnar&Plotkin ‘09]
- Key insight: Multiple *labelled* reifiers (called ‘control tags’)
- Each delimiter *captures* a particular set of control tags
- Continuations are one-shot
- Intuition: exceptions with the ability to resume

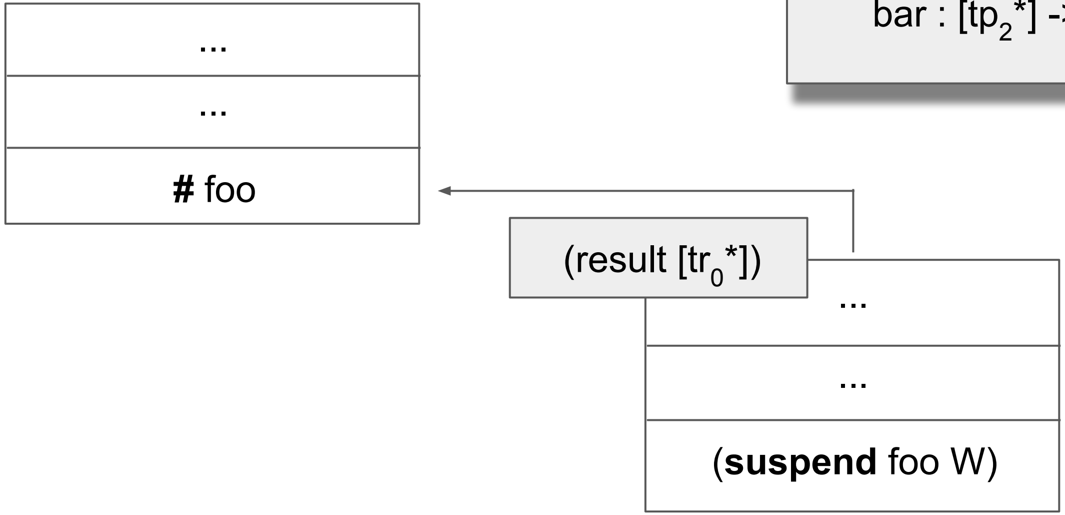
Proposal Dependencies

- Exception handling
- Reftypes
- Funcref

Typed Continuations, the Idea

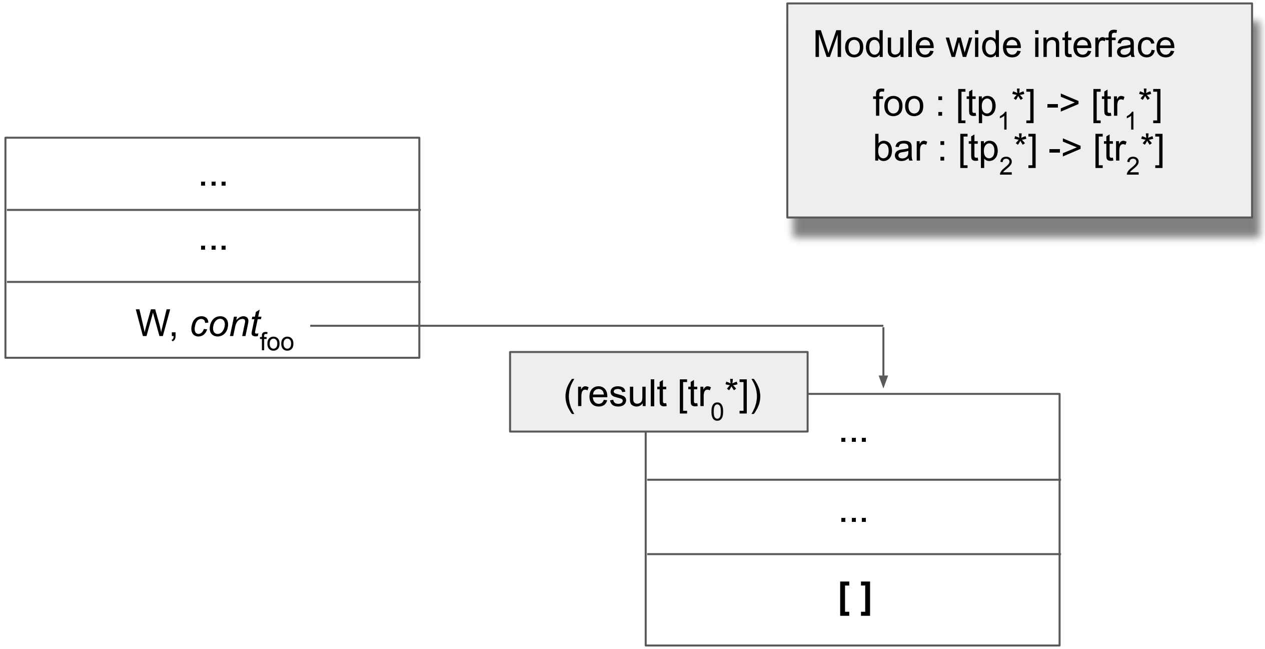
Module wide interface

foo : [tp_1*] -> [tr_1*]
bar : [tp_2*] -> [tr_2*]



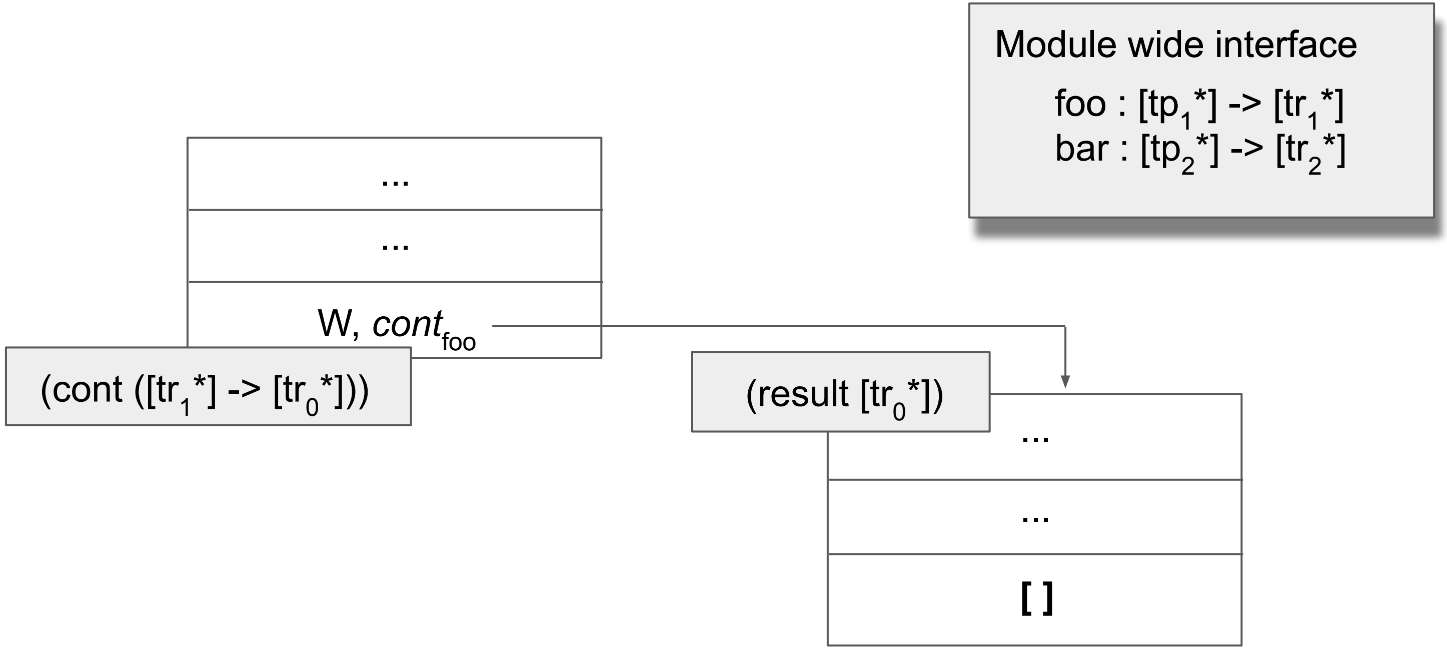
(intuition: **suspend** *t* V ‘sends’ data V to the (typed) channel *t*)

Typed Continuations, the Idea



(intuition: **suspend** *t* V ‘sends’ data V to the (typed) channel *t*)

Typed Continuations, the Idea



Module wide interface
foo : [tp$_1$*] -> [tr$_1$*]
bar : [tp$_2$*] -> [tr$_2$*]
...
...
W, *cont*$_{foo}$
(cont ([tr$_1$*] -> [tr$_0$*]))
(result [tr$_0$*])
...
...
[ ]

(intuition: **suspend** *t* V ‘sends’ data V to the (typed) channel *t*)

Wasm Instructions

Control Tags

- An extension of exception handling tags

```
(tag $tag (param tp*) (result tr*))
```

Creating Continuations

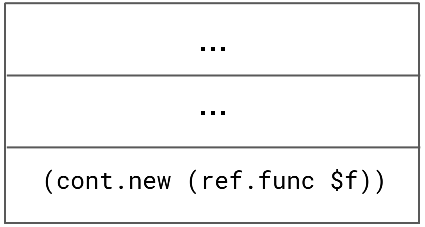
- Continuation type; a new reference type

```
(cont ([t1*] -> [t2*]))
```

- Fresh continuation

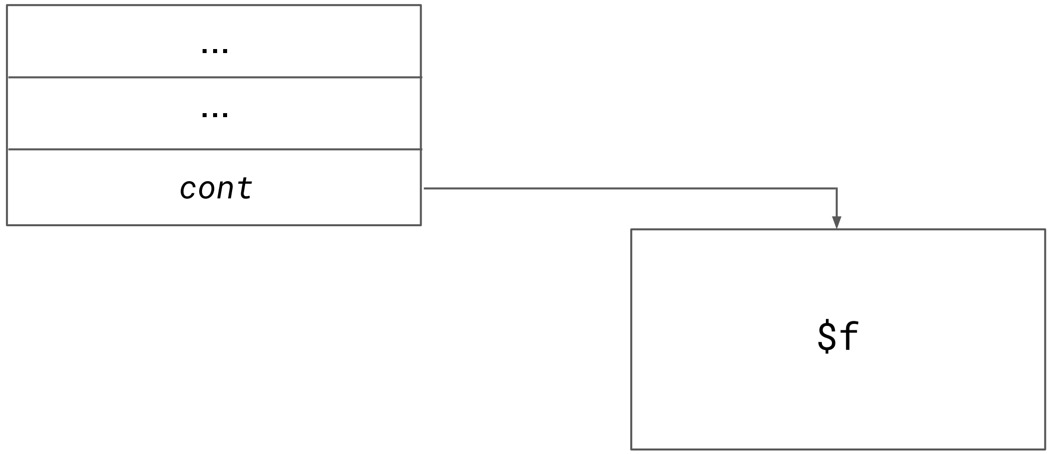
```
cont.new : [(ref ([t1*] -> [t2*])] -> [(cont ([t1*] -> [t2*]))]
```

Cont.new, Operationally



```
(cont.new (ref.func $f))
```

Cont.new, Operationally



Resuming Continuations

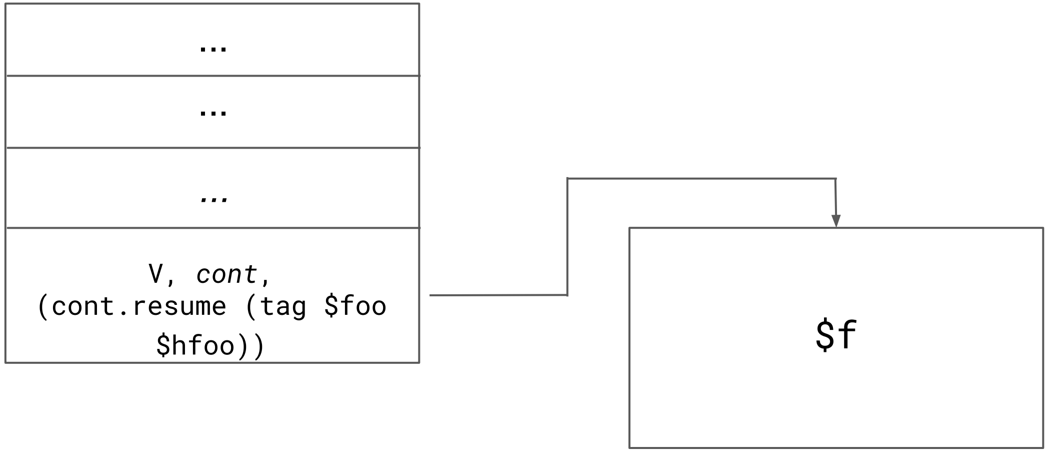
- Ordinary resume

```
cont.resume (tag $tag $handler)* : [tr* (cont ([tr*] -> [t2*]))] -> [t2*]
```

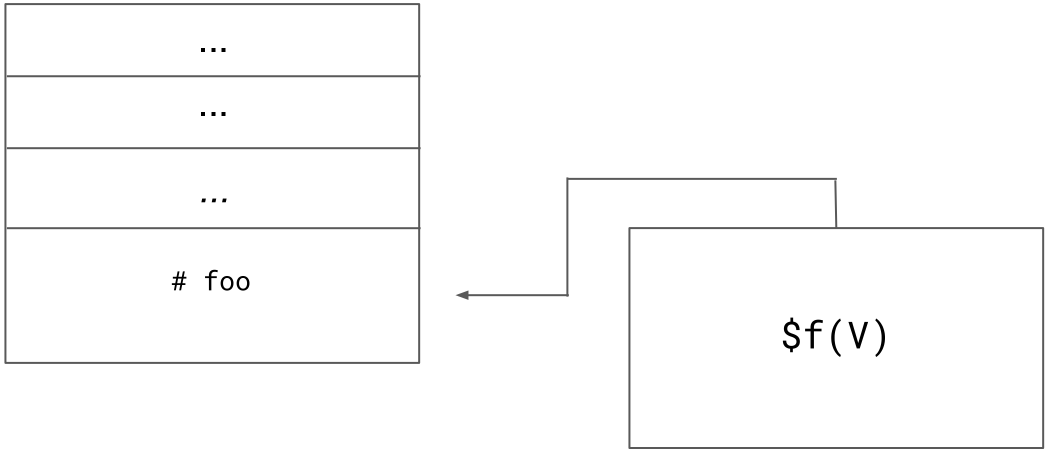
- Abortive resume

```
cont.throw (exception $exn) : [tp* (cont $ft)] -> [t2*]
```

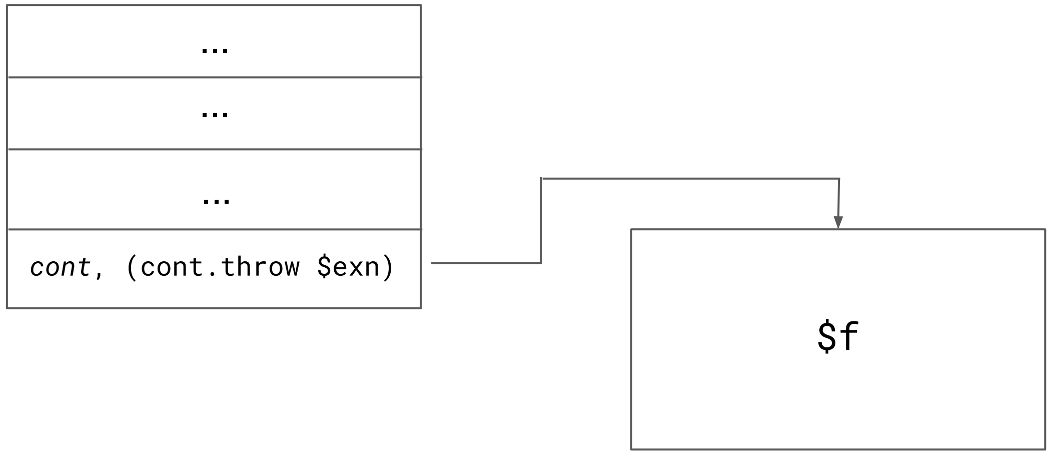
Cont.resume, Operationally



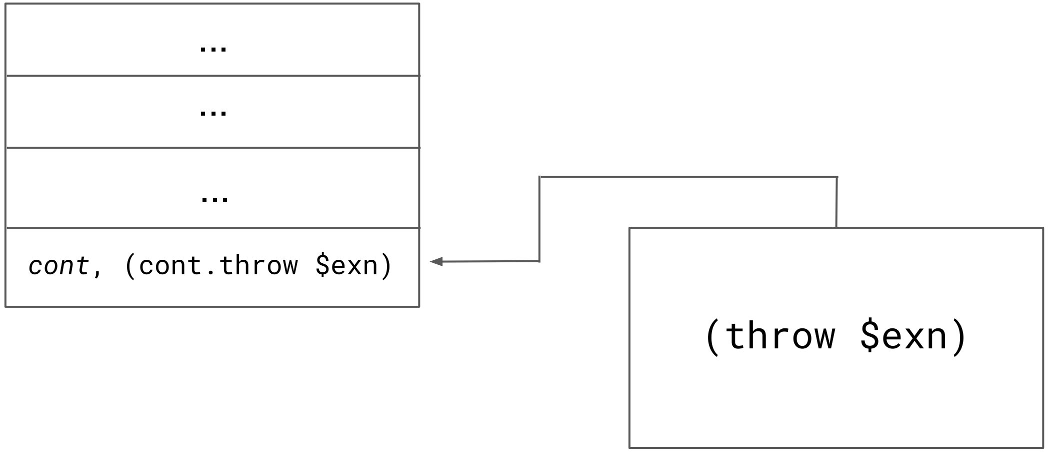
Cont.resume, Operationally



Cont.throw, Operationally



Cont.resume, Operationally



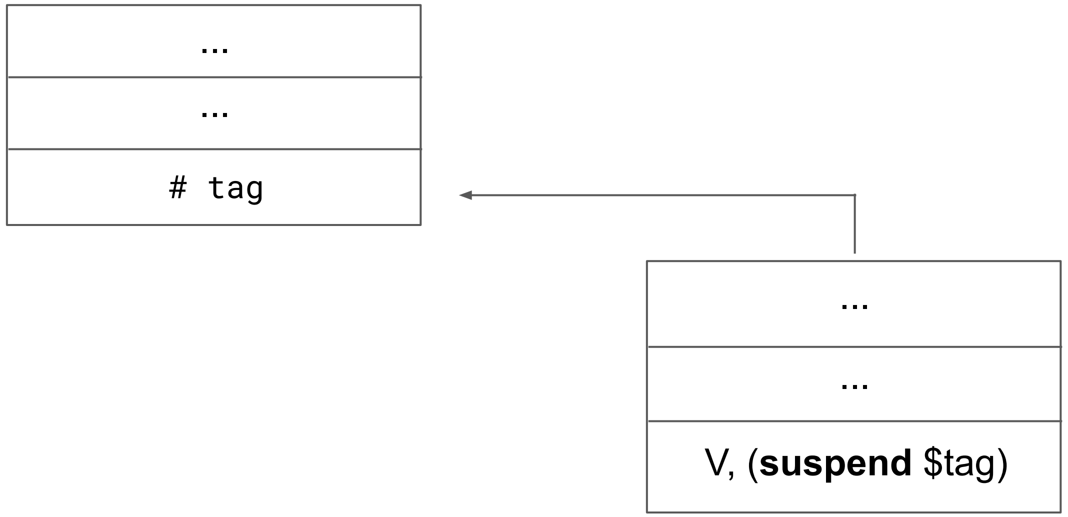
Suspending Continuations

- Labelled suspension

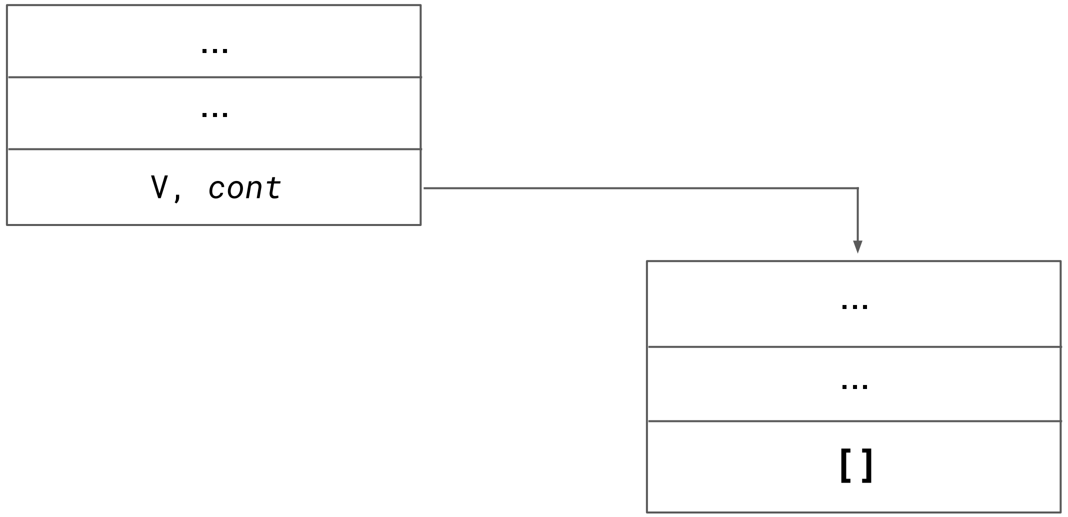
```
cont.suspend $tag : [tp*] -> [tr*]
```

(must match declaration tag : [tp*] -> [tr*])

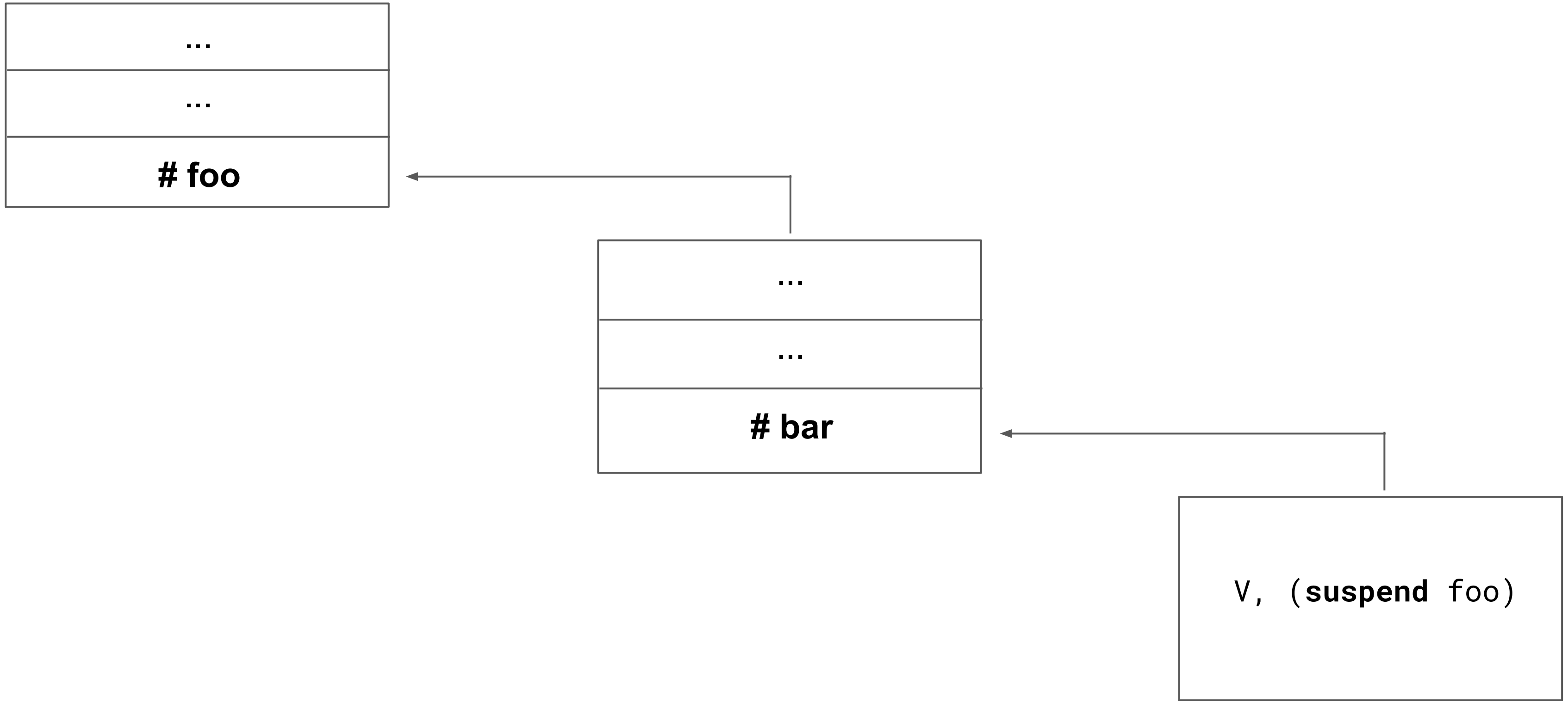
Cont.suspend, Operationally



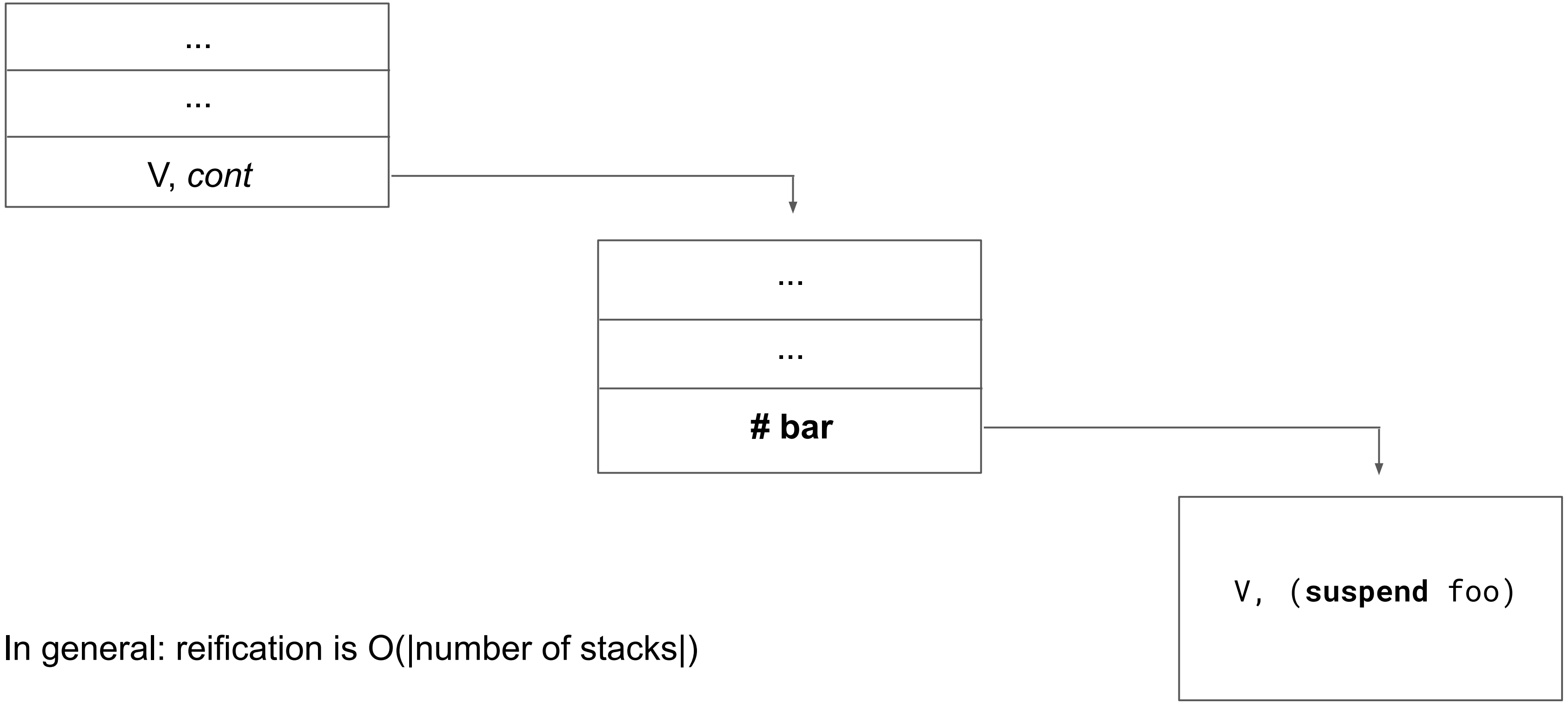
Cont.suspend, Operationally



Cont.suspend does not guarantee O(1) transfer



Cont.suspend does not guarantee O(1) transfer



In general: reification is O(|number of stacks|)

Control Barriers

- Provides a mechanism for ‘trapping’ control

  ```
  (barrier $lbl bt instr*)
  ```

  Ensures that control cannot be captured across language boundaries

Partial Continuation Application

- Provides a way to ‘shrink’ continuation types

```
cont.bind $ct : [tp* (cont ([tp* tp'*] -> [t2*]))] -> [(cont ([tp'*] ->
[t2*]))]
```

(example usage: https://github.com/effect-handlers/wasm-spec/blob/explainer/proposals/continuations/examples/actor.wast#L47)

Extensions

- Static tag dispatch
- Symmetric stack switching
- Tail-resumptive handler clauses
- Multi-shot continuations

Closing Remarks

- This proposal is being developed at

  https://github.com/effect-handlers/wasm-spec
  (feedback & comments are welcome)

- Current status:
  - Operational semantics written down
  - Reference interpreter
  - Explainer document