Racket's Intermediate Language for Control

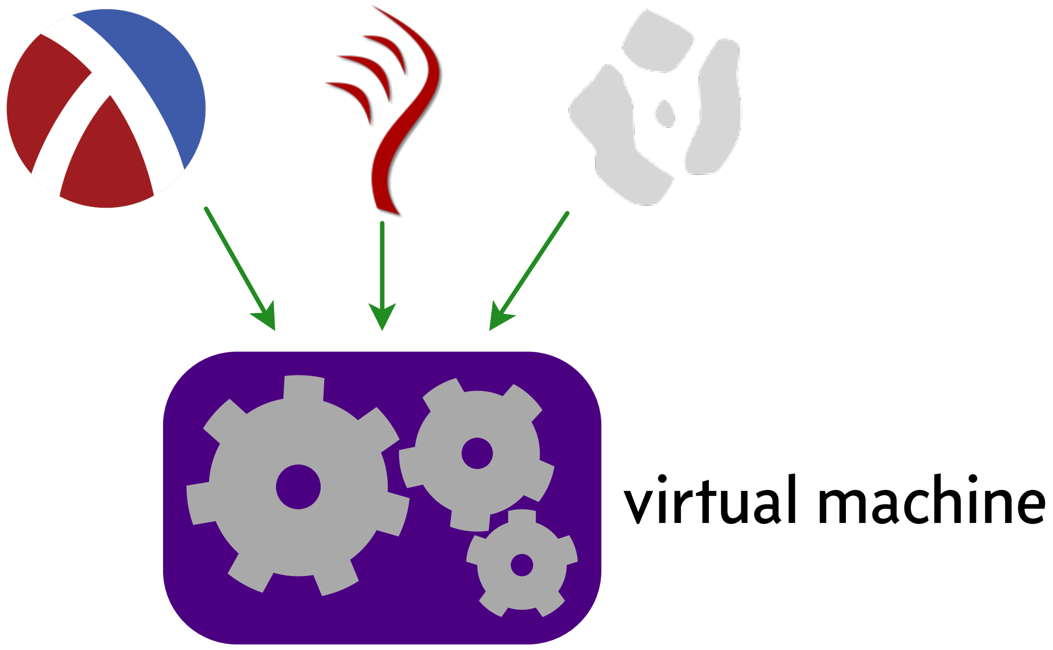
based on talks at VMIL'20 and PLDI'20

Matthew Flatt

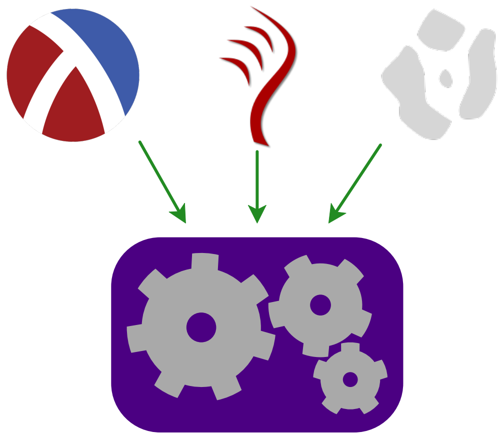
University of Utah

including work with R. Kent Dybvig, John Clements, Gang Yu, Robby Findler, and Matthias Felleisen

Language Building Blocks



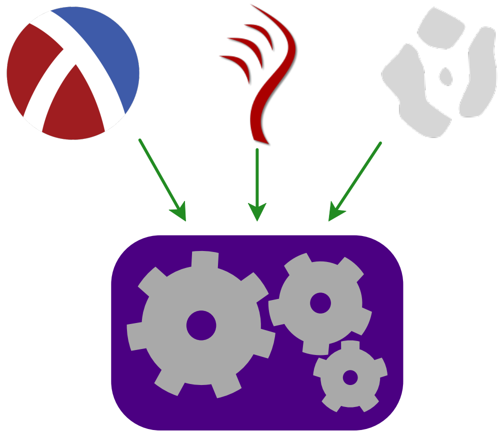
Language Building Blocks



functions

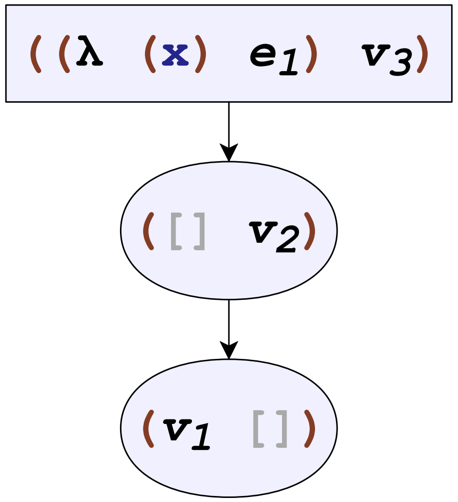
```
(λ (x) e₁)
```

Language Building Blocks

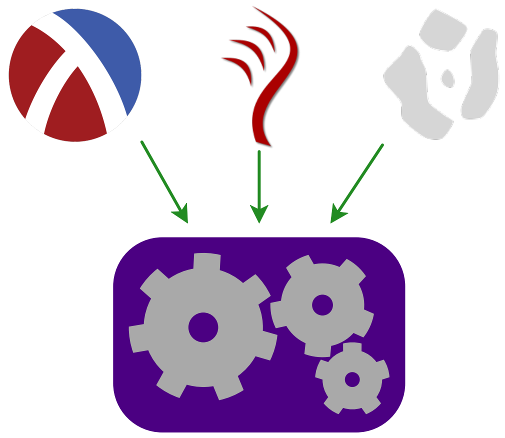


functions

call and return



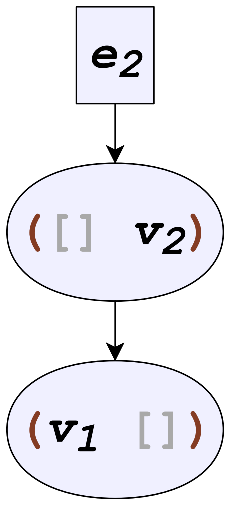
Language Building Blocks



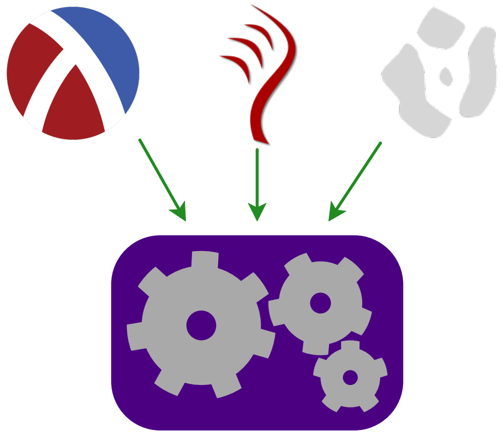
functions

call and return

tail calls



Language Building Blocks

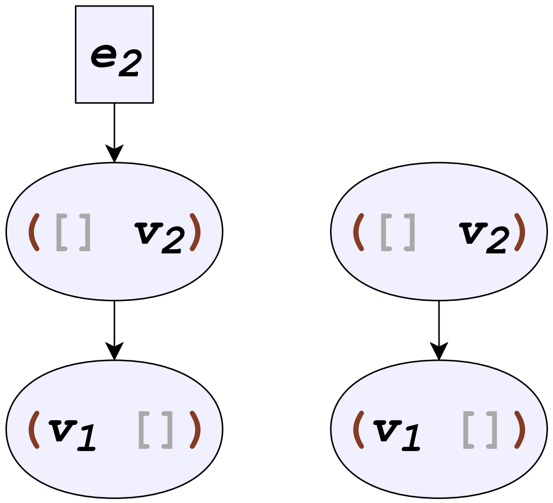


functions

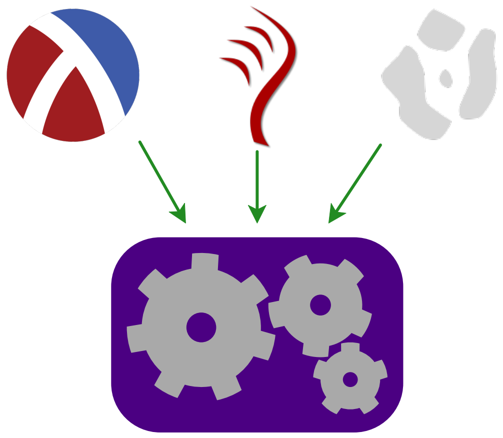
call and return

tail calls

continuations



Language Building Blocks



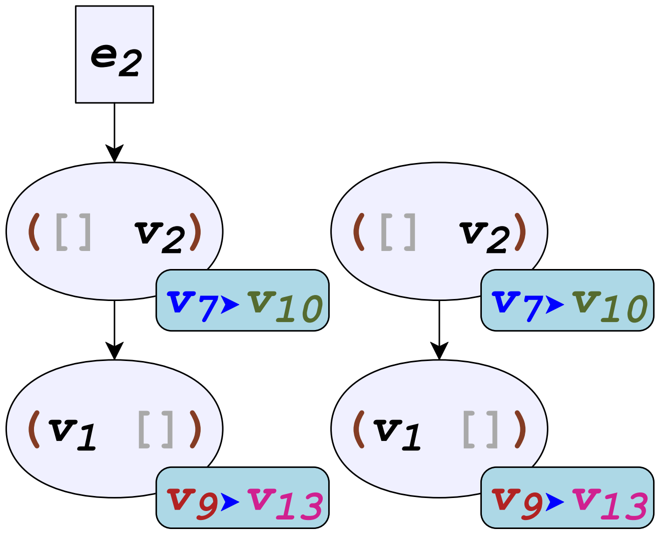
functions

call and return

tail calls

continuations

continuation marks



e_2
$([]\ v_2)$
$v_7 \blacktriangleright v_{10}$
$(v_1\ [])$
$v_9 \blacktriangleright v_{13}$
$([]\ v_2)$
$v_7 \blacktriangleright v_{10}$
$(v_1\ [])$
$v_9 \blacktriangleright v_{13}$

Language Building Blocks

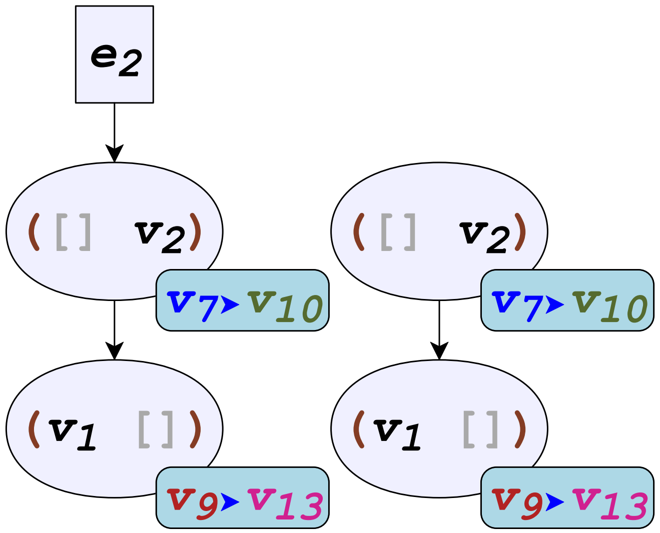
functions

call and return

tail calls

continuations

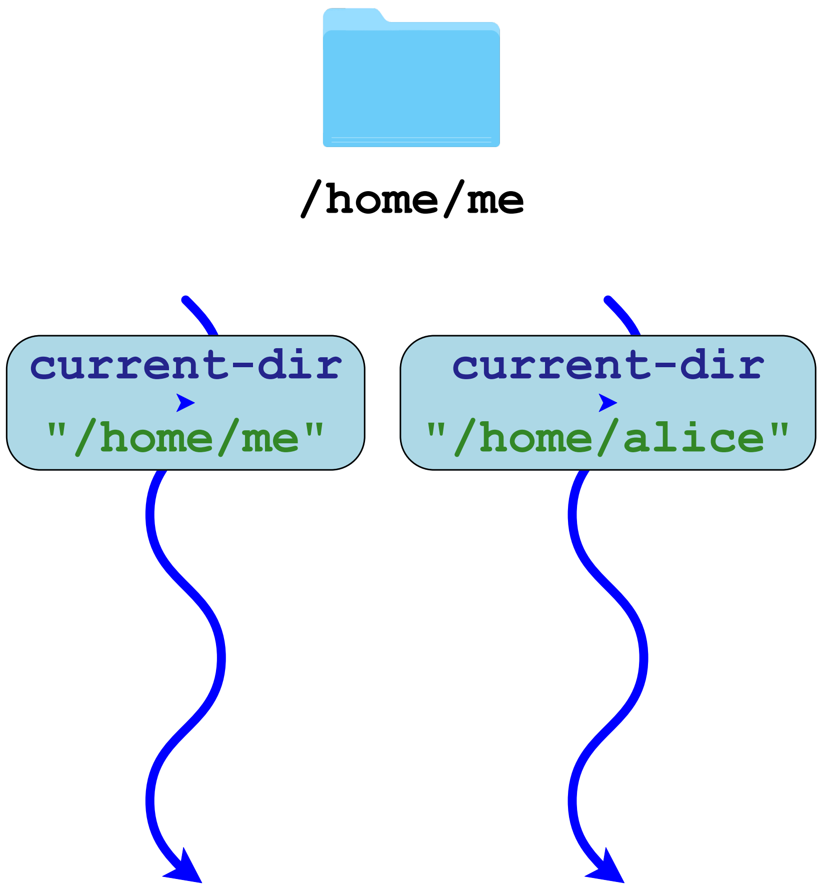
continuation marks



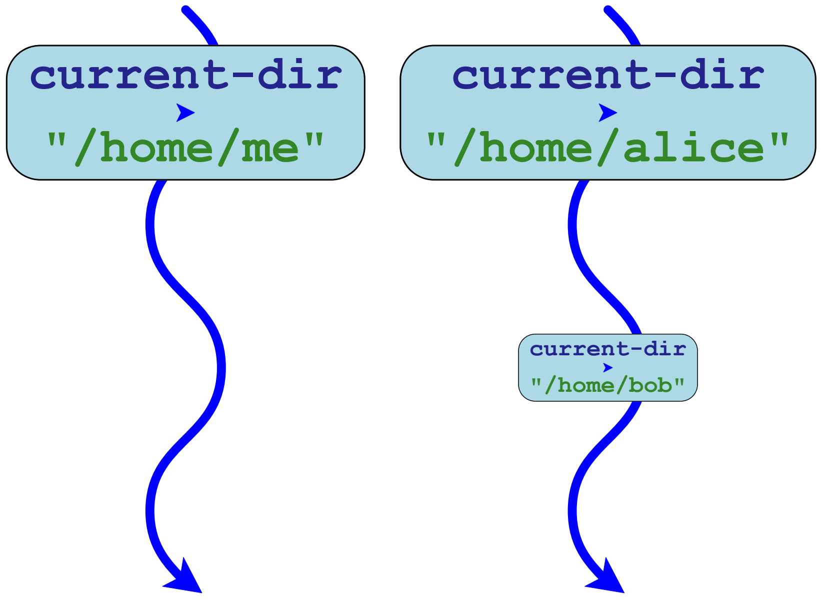
Continuations and Marks

`/home/me`

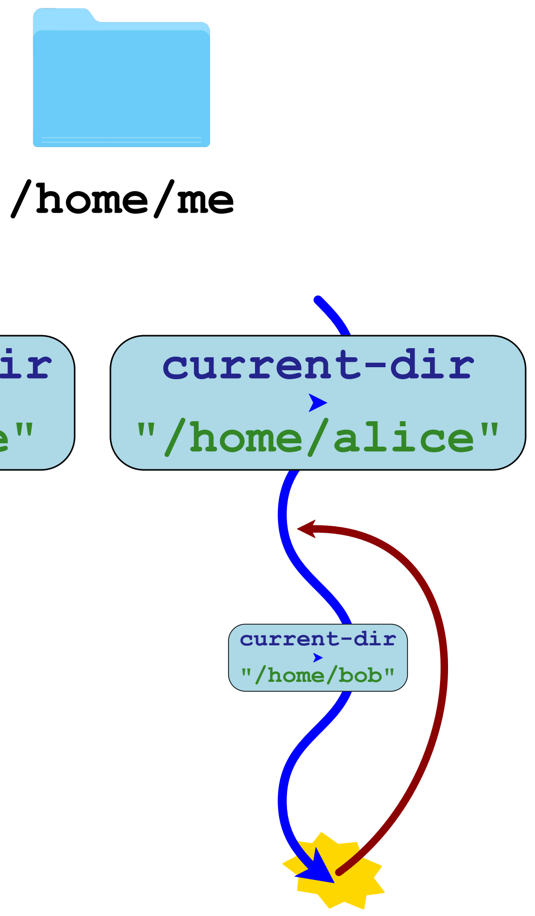
Continuations and Marks



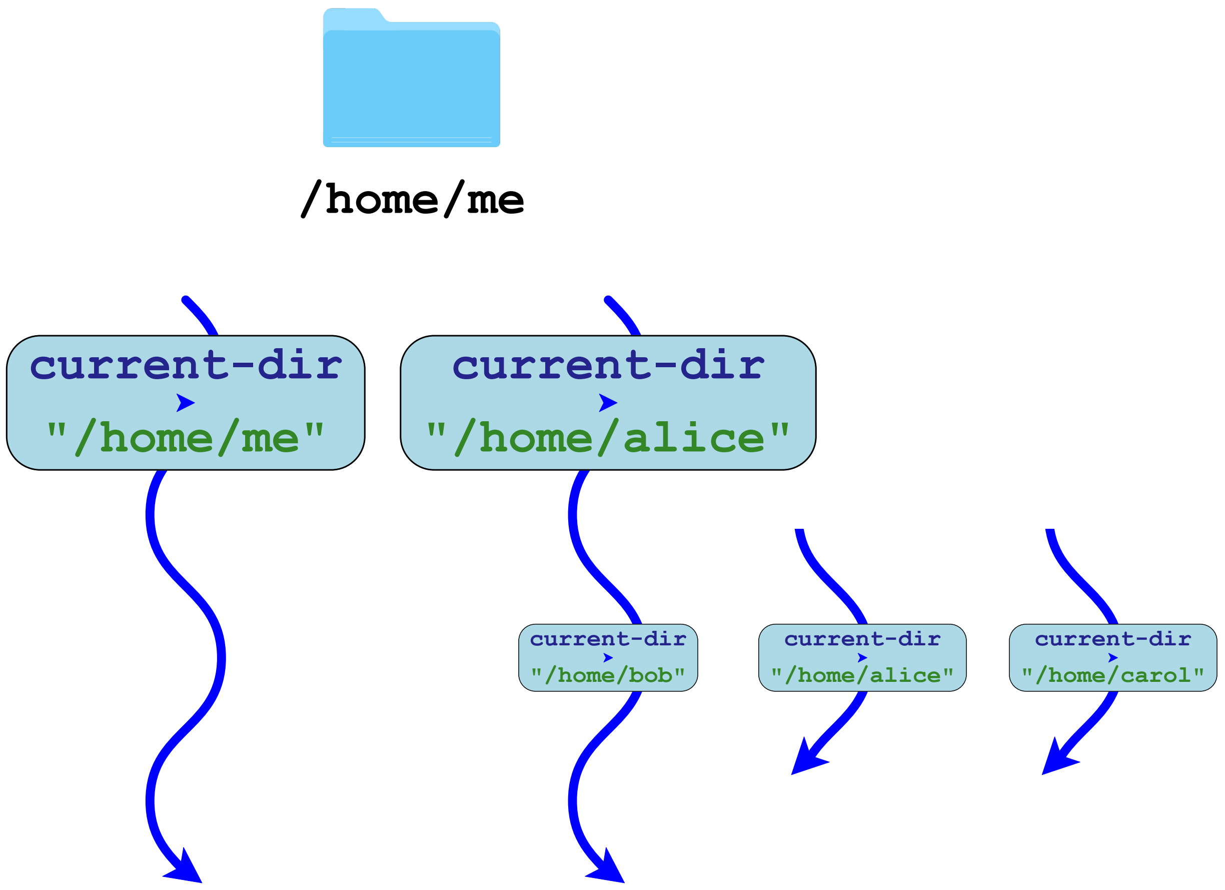
Continuations and Marks



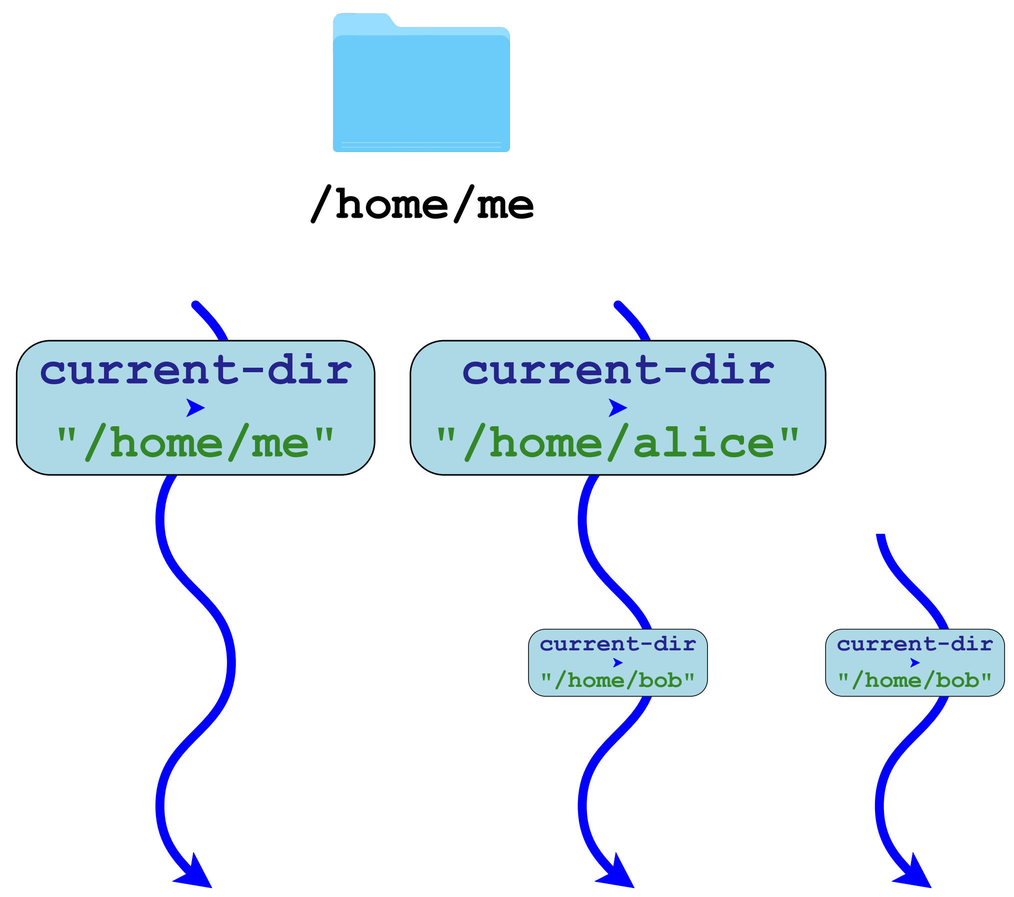
Continuations and Marks



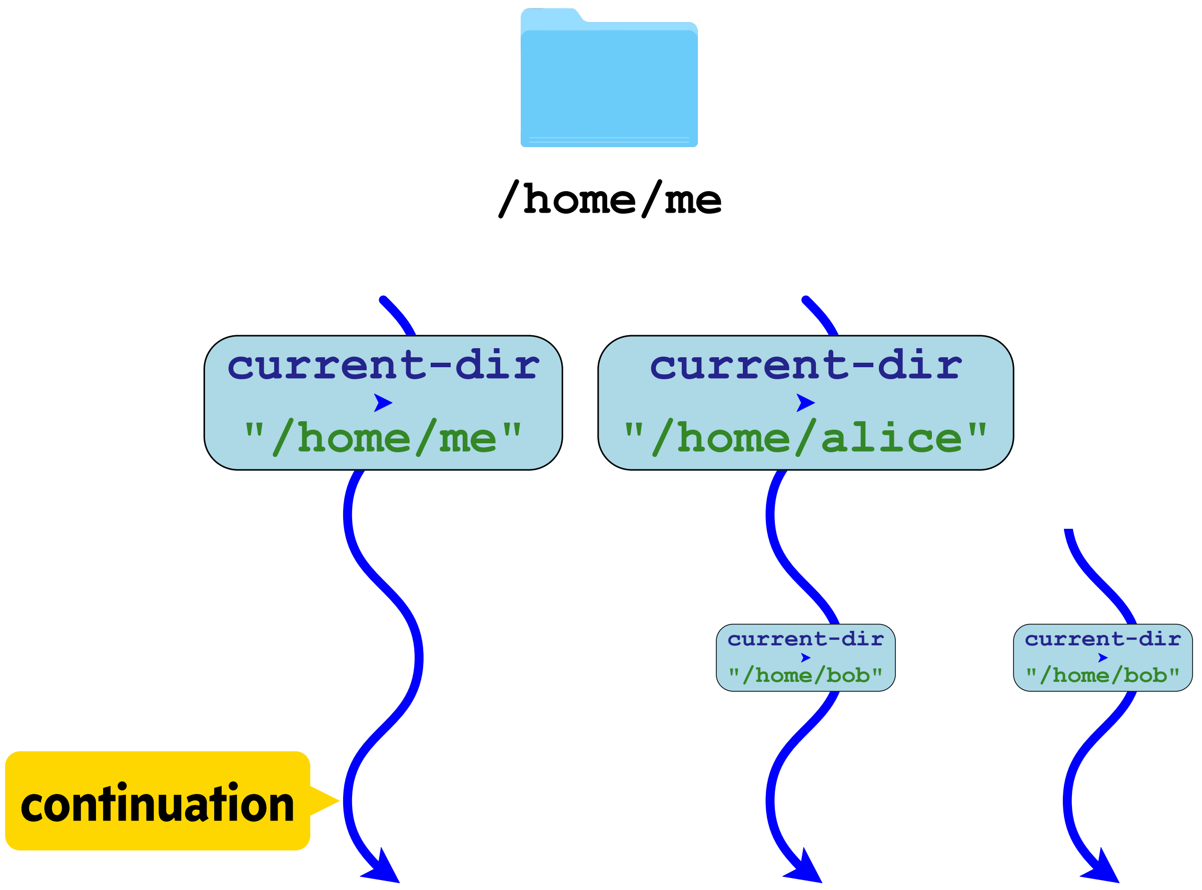
Continuations and Marks



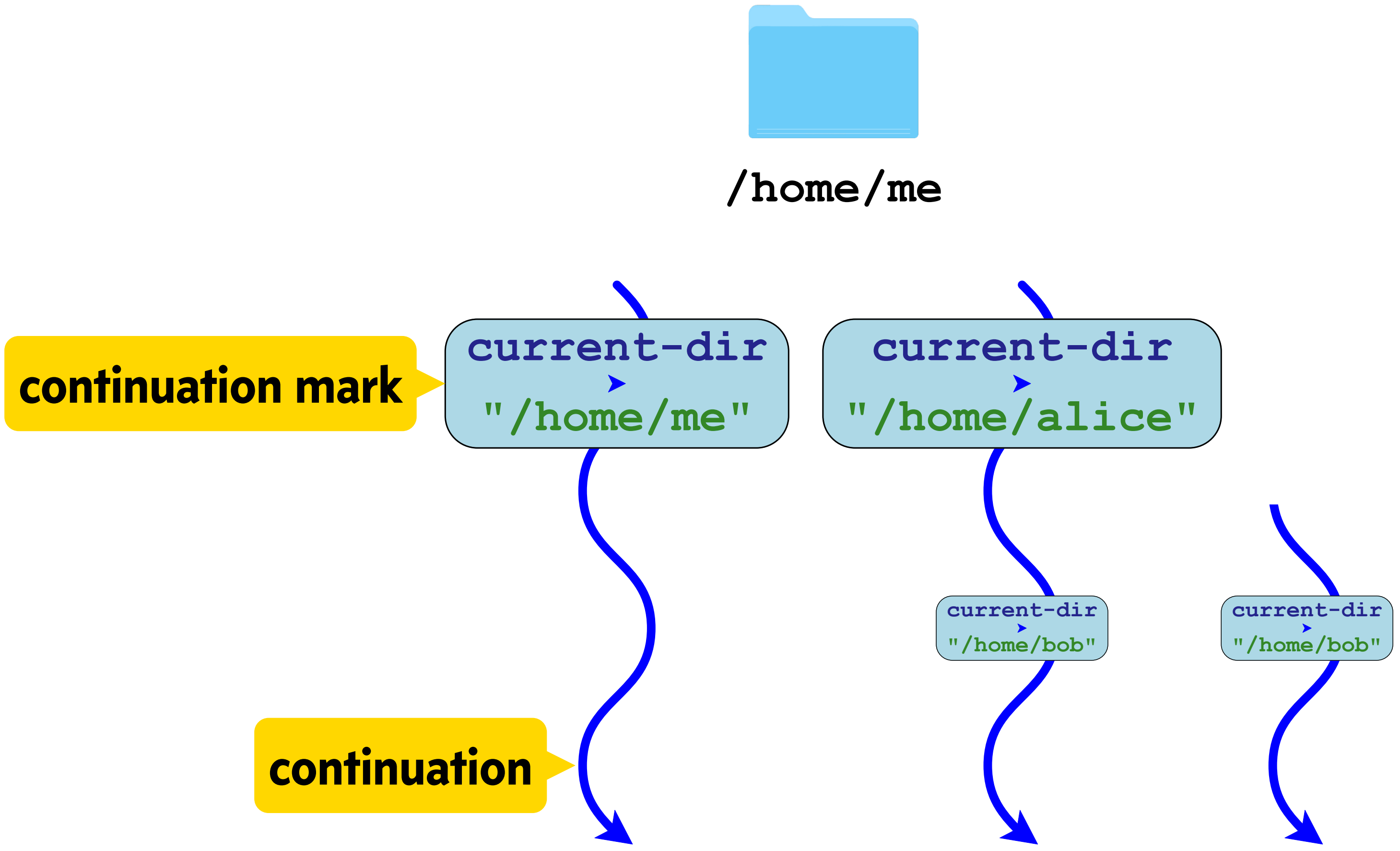
Continuations and Marks



Continuations and Marks



Continuations and Marks



Some Papers on Control

Felleisen ’88
Danvy and Filinski ’90
Hieb and Dybvig ’90
Sitaram and Felleisen ’90
Queinnec and Serpette ’91
Reppy ’91
Sitaram ’93
Queinnec ’93
Moreau and Queinnec ’94
Wadler ’94
deGroote ’94
Rehof and Sørensen ’94
Gunter et al. ’95
Gunter et al. ’95
Thielecke ’97
Flatt et al. ’99
Feeley ’01
Rehof ’01
Clements et al. ’01
Gasbichler and Sperber ’02
Gasbichler and Sperber ’03
Kameyama and Hasegawa ’03
Ariola et al. ’04
Shan ’04
Queinnec ’04
Saurin ’05
Kiselyov et al. ’06
Dybvig et al. ’06
Biernacki et al. ’06
Flatt et al. ’07
Ariola et al. ’09
Plotkin and Pretnar ’09
Garcia et al. ’10
James and Sabry ’11
Kiselyov ’12
Kiselyov et al. ’13
Plotkin and Pretnar ’13
Downen and Ariola ’14
Dolan et al. ’15
Hillerström and Lindley ’16
Philips et al. ’16
Hillerström and Lindley ’18
Flatt and Dybvig ’19
Brachthauser and Leijen ’19
Biernacki ’19

Some Papers on Control

Felleisen '88	Flatt et al. '99	Ariola et al. '09
Danvy and Filinski '90	Feeley '01	Plotkin and Pretnar '09
Hieb and Dybvig '90	Rehof '01	Garcia et al. '10
Sitaram and Felleisen '90	Clements et al. '01	James and Sabry '11
Queinnec and Serpette '91	Gasbichler and Sperber '02	Kiselyov '12
Reppy '91	Gasbichler and Sperber '03	Kiselyov et al. '13
Sitaram '93	Kameyama and Hasegawa '03	Plotkin and Pretnar '13
Queinnec '93	Ariola et al. '04	Downen and Ariola '14
Moreau and Queinnec '94	Shan '04	Dolan et al. '15
Wadler '94	Queinnec '04	Hillerström and Lindley '16
deGroote '94	Saurin '05	Philips et al. '16
Rehof and Sørensen '94	Kiselyov et al. '06	Hillerström and Lindley '18
Gunter et al. '95	Dybvig et al. '06	Flatt and Dybvig '19
Gunter et al. '95	Biernacki et al. '06	Brachthauser and Leijen '19
Thielecke '97	Flatt et al. '07	Biernacki '19

Racket Control API

Continuations
function [tail] calls
`call/cc`
`call-with-composable-continuation`
continuation application
`call-in-continuation`

Prompts
`call-with-continuation-prompt`
`abort-current-continuation`
`continuation-prompt-available?`
`make-continuation-prompt-tag`
`default-continuation-prompt-tag`

Winding
`dynamic-wind`
`call-with-continuation-barrier`

Exceptions
`raise`
`call-with-exception-handler`

Continuation Marks
`with-continuation-mark`
`continuation-marks`
`current-continuation-marks`
`call-with-immediate-continuation-mark`
`continuation-mark-set->iterator`
`continuation-mark-set-first`

Dynamic Binding
`parameterize`
`make-parameter`
parameter application
`current-parameterization`

Threads (Green)
`thread`
`kill-thread`
`make-thread-group`

Thread-Local Storage
`make-thread-cell`
`thread-cell-ref`
`thread-cell-set!`
`current-preserved-thread-cell-values`

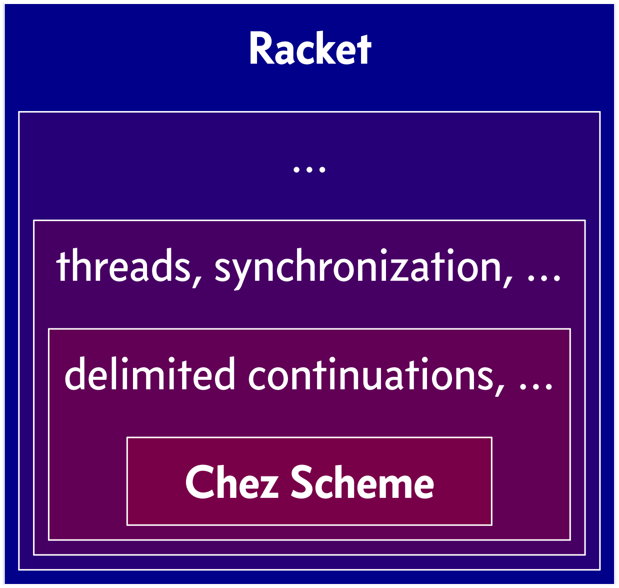
Asynchronous Exceptions
`break-thread`
`break-enabled`
`current-break-parameterization`
`call-with-break-parameterization`

Synchronization
`sync`
`make-channel`
`channel-put-evt`
`channel-get-evt`
`make-semaphore`
`semaphore-post`

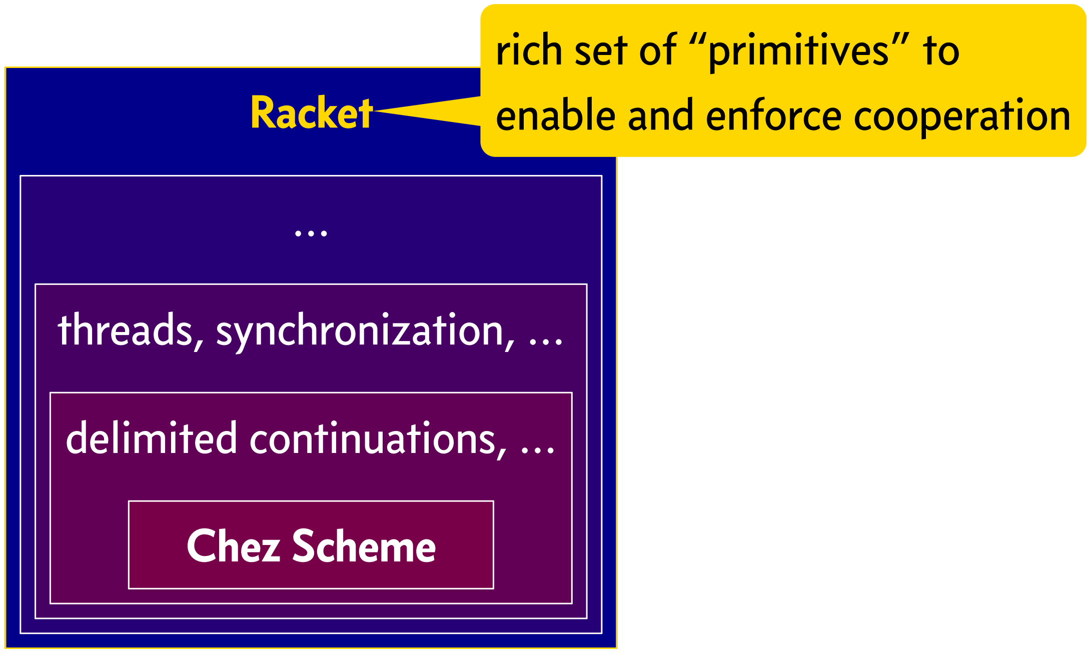
Custodians
`make-custodian`
`custodian-shutdown-all`

Futures
`future`
`touch`

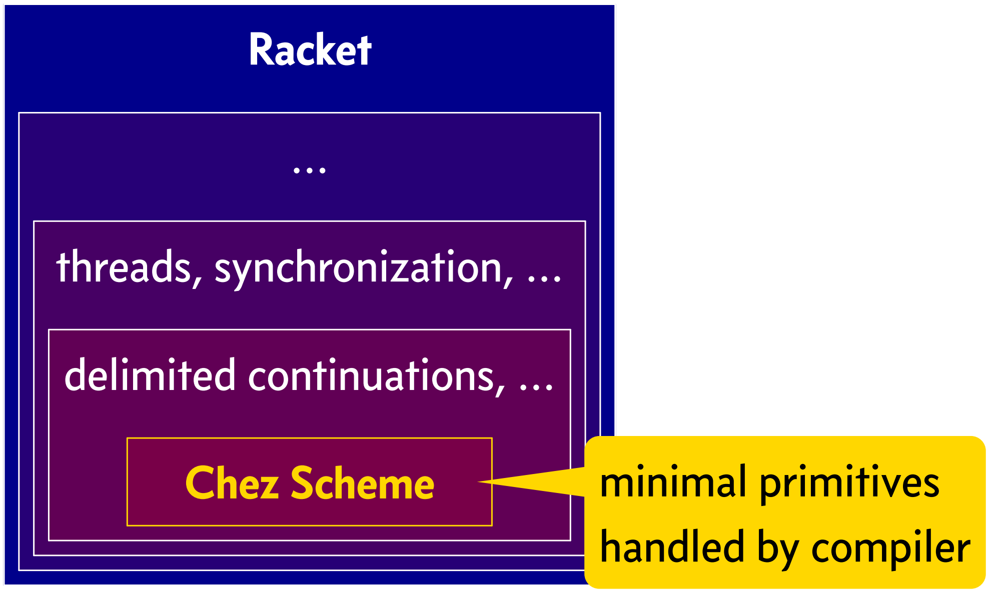
Racket Implementation



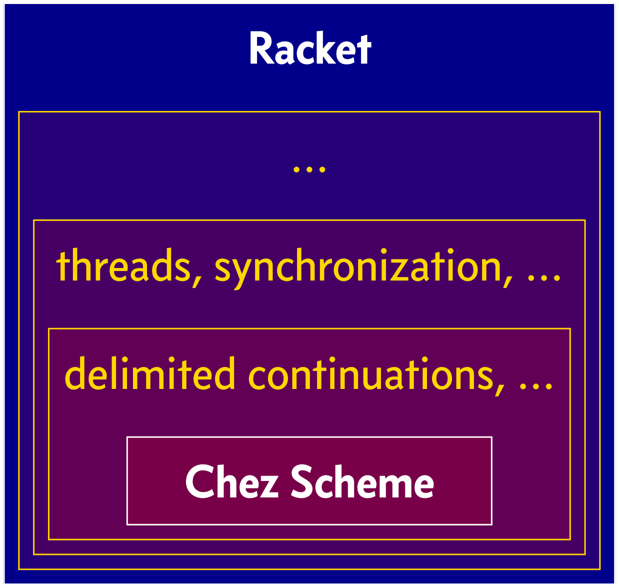
Racket Implementation


Racket
rich set of “primitives” to enable and enforce cooperation
...
threads, synchronization, ...
delimited continuations, ...
Chez Scheme

Racket Implementation


Racket
...
threads, synchronization, ...
delimited continuations, ...
Chez Scheme
minimal primitives handled by compiler

Racket Implementation



Chez Scheme Control API

as used to implement Racket

Continuations

function [tail] calls

`call/cc`

`call-in-continuation`

`null-continuation`

Continuation Marks

`current-continuation-attachments`

`continuation-next-attachments`

`call-getting-continuation-attachment`

`call-setting-continuation-attachment`

`call-consuming-continuation-attachment`

Exceptions

`raise`

`base-exception-handler`

Signals

`start-timer`

`timer-interrupt-handler`

Chez Scheme Control API

as used to implement Racket

globally connect to primitive exceptions

Continuations
function [tail] calls
`call/cc`
`call-in-continuation`
`null-continuation`

Continuation Marks
`current-continuation-attachments`
`continuation-next-attachments`
`call-getting-continuation-attachment`
`call-setting-continuation-attachment`
`call-consuming-continuation-attachment`

Exceptions
`raise`
`base-exception-handler`

Signals
`start-timer`
`timer-interrupt-handler`

Chez Scheme Control API

as used to implement Racket

Continuations
function [tail] calls
`call/cc`
`call-in-continuation`
`null-continuation`

Continuation Marks
`current-continuation-attachments`
`continuation-next-attachments`
`call-getting-continuation-attachment`
`call-setting-continuation-attachment`
`call-consuming-continuation-attachment`

Exceptions
`raise`
`base-exception-handler`

Signals
`start-timer`
`timer-interrupt-handler`

timer signal to implement (green) thread preemption

Chez Scheme Control API

as used to implement Racket

Continuations

function [tail] calls

`call/cc`

`call-in-continuation`

`null-continuation`

Continuation Marks

`current-continuation-attachments`

`continuation-next-attachments`

`call-getting-continuation-attachment`

`call-setting-continuation-attachment`

`call-consuming-continuation-attachment`

Exceptions

`raise`

`base-exception-handler`

Signals

`start-timer`

`timer-interrupt-handler`

Control Layers

Racket

Continuations
function [tail] calls
`call/cc`
`call-with-composable-continuation`
continuation application
`call-in-continuation`

Prompts
`call-with-continuation-prompt`
`abort-current-continuation`
`continuation-prompt-available?`
`make-continuation-prompt-tag`
`default-continuation-prompt-tag`

Winding
`dynamic-wind`
`call-with-continuation-barrier`

Exceptions
`raise`
`call-with-exception-handler`

Continuation Marks
`with-continuation-mark`
`continuation-marks`
`current-continuation-marks`
`call-with-immediate-continuation-mark`
`continuation-mark-set->iterator`
`continuation-mark-set-first`

Dynamic Binding
`parameterize`
`make-parameter`
parameter application
`current-parameterization`

Threads (Green)
`thread`
`kill-thread`
`make-thread-group`

Thread-Local Storage
`make-thread-cell`
`thread-cell-ref`
`thread-cell-set!`
`current-preserved-thread-cell-values`

Asynchronous Exceptions
`break-thread`
`break-enabled`
`current-break-parameterization`
`call-with-break-parameterization`

Synchronization
`sync`
`make-channel`
`channel-put-evt`
`channel-get-evt`
`make-semaphore`
`semaphore-post`

Custodians
`make-custodian`
`custodian-shutdown-all`

Futures
`future`
`touch`

Chez Scheme

Continuations
function [tail] calls
`call/cc`
`call-in-continuation`
`null-continuation`

Continuation Marks
`current-continuation-attachments`
`continuation-next-attachments`
`call-getting-continuation-attachment`
`call-setting-continuation-attachment`
`call-consuming-continuation-attachment`

Exceptions
`raise`
`base-exception-handler`

Signals
`start-timer`
`timer-interrupt-handler`

Part 1 of 2

Language for Control

Continuations
function [tail] calls
`call/cc`
`call-with-composable-continuation`
continuation application
`call-in-continuation`

Prompts
`call-with-continuation-prompt`
`abort-current-continuation`
`continuation-prompt-available?`
`make-continuation-prompt-tag`
`default-continuation-prompt-tag`

Winding
`dynamic-wind`
`call-with-continuation-barrier`

Exceptions
`raise`
`call-with-exception-handler`

Continuation Marks
`with-continuation-mark`
`continuation-marks`
`current-continuation-marks`
`call-with-immediate-continuation-mark`
`continuation-mark-set->iterator`
`continuation-mark-set-first`

Dynamic Binding
`parameterize`
`make-parameter`
parameter application
`current-parameterization`

Threads (Green)
`thread`
`kill-thread`
`make-thread-group`

Thread-Local Storage
`make-thread-cell`
`thread-cell-ref`
`thread-cell-set!`
`current-preserved-thread-cell-values`

Asynchronous Exceptions
`break-thread`
`break-enabled`
`current-break-parameterization`
`call-with-break-parameterization`

Synchronization
`sync`
`make-channel`
`channel-put-evt`
`channel-get-evt`
`make-semaphore`
`semaphore-post`

Custodians
`make-custodian`
`custodian-shutdown-all`

Futures
`future`
`touch`

Notation

```
(+ 1 (* (- 7 3) 2))
```

Notation

```
(+ 1 (* (- 7 3) 2))
```

Notation

```
(- 7 3)
```

```
(+ 1 (* (- 7 3) 2))
```

Notation

```
(- 7 3)
```

```
(+ 1 (* (- 7 3) 2))
```

Notation

```
                          (- 7 3)

(+ 1 (* (- 7 3) 2))       (* [] 2)
```

Notation

```
(- 7 3)
```

```
(+ 1 (* (- 7 3) 2))
```

```
(* [] 2)
```

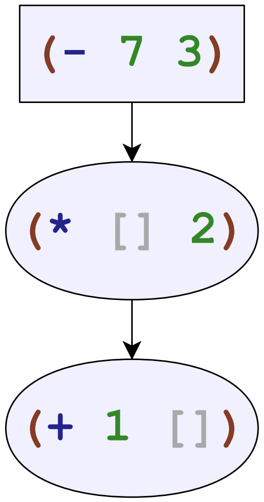
Notation

```
(+ 1 (* (- 7 3) 2))
```

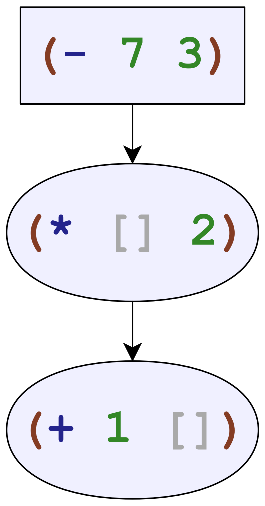
```
(- 7 3)

(* [] 2)

(+ 1 [])
```

Notation

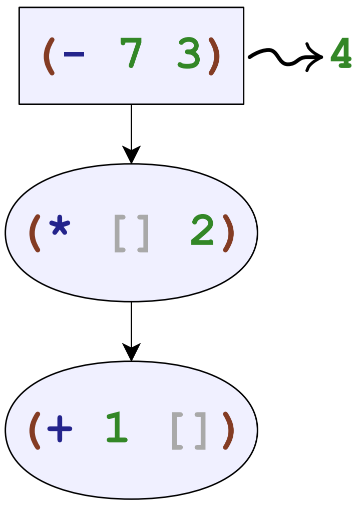
`(+ 1 (* (- 7 3) 2)) =`



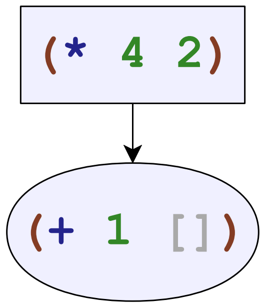
Reductions



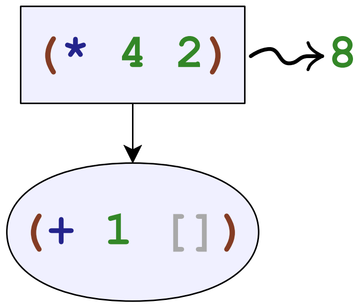
Reductions



Reductions



Reductions



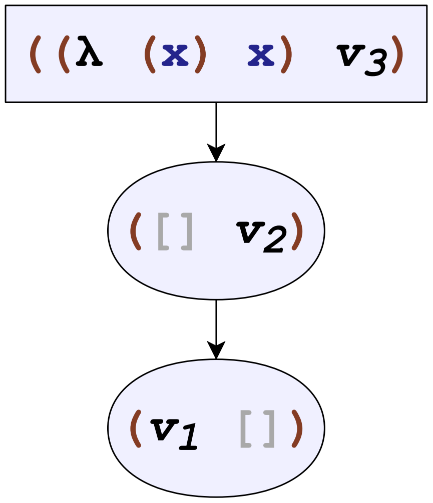
Reductions

```
(+ 1 8)
```

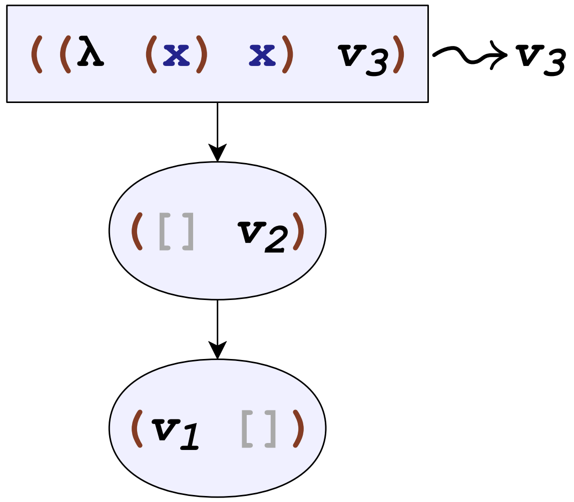
Notation

```
(v₁ (((λ (x) x) v₃) v₂))
```

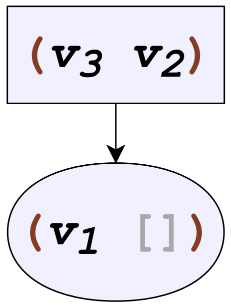
Reductions



Reductions



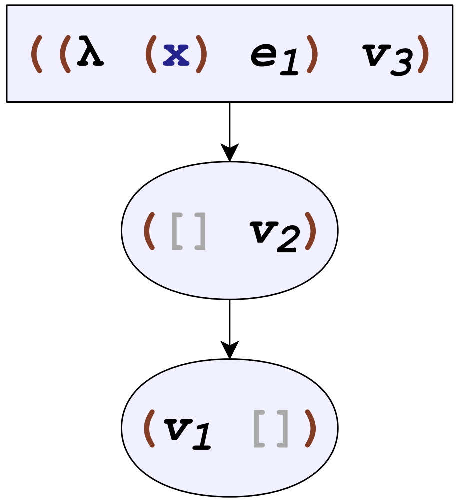
Reductions



Reductions

$$\begin{tikzcd}
\boxed{\texttt{((}\lambda\ \texttt{(x) x)}\ v_3\texttt{)}} \arrow[d] \\
\texttt{([]}\ v_2\texttt{)} \arrow[d] \\
\texttt{(}v_1\ \texttt{[])}
\end{tikzcd}$$

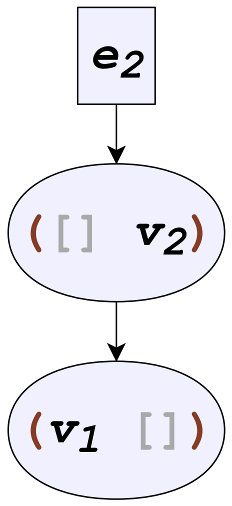
Tail Evaluation



Tail Evaluation

$$\begin{tikzcd} \boxed{((\lambda\ (x)\ e_1)\ v_3)} \arrow[d] \arrow[r, rightsquigarrow] & e_2 \;=\; e_1[x \leftarrow v_3] \\ ([\,]\ \ v_2) \arrow[d] & \\ (v_1\ \ [\,]) & \end{tikzcd}$$

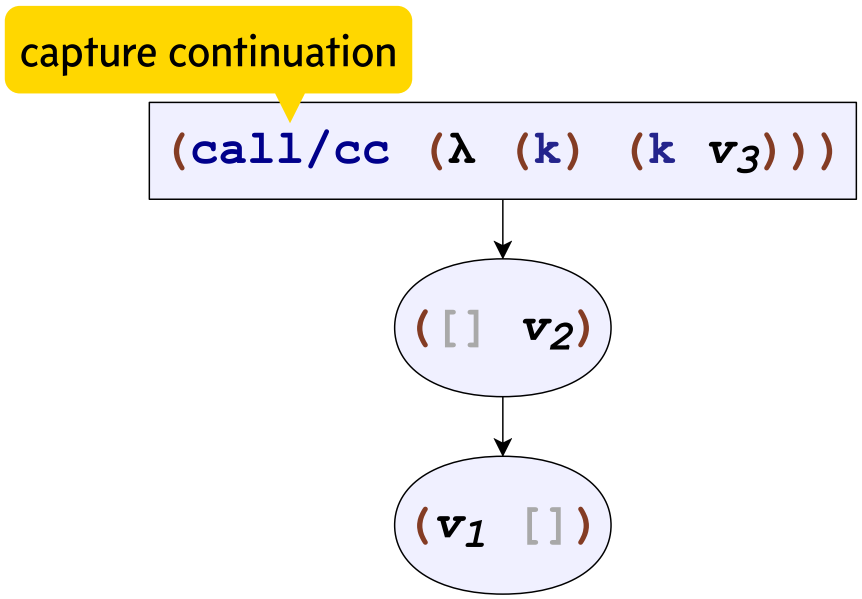
Tail Evaluation



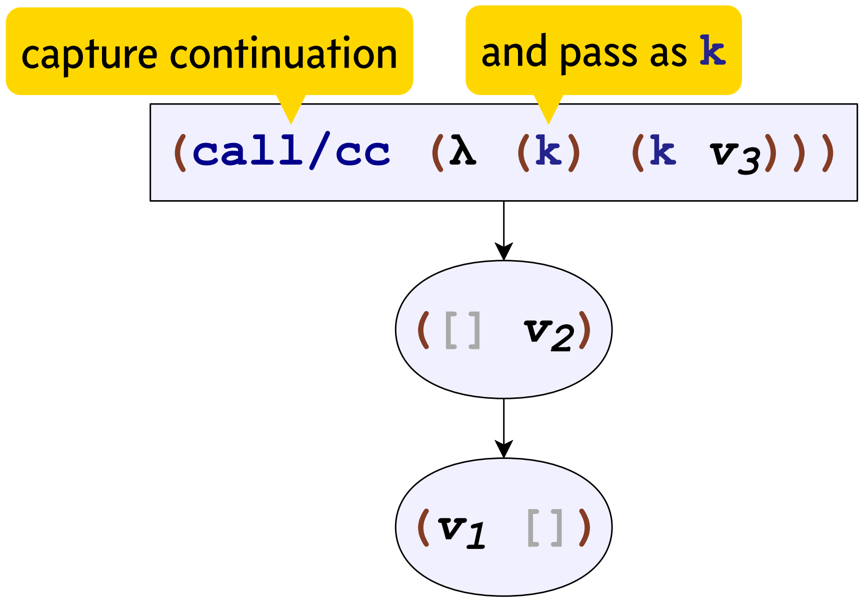
Continuations

$$\begin{tikzcd}
\boxed{\texttt{(call/cc (}\lambda\texttt{ (k) (k }v_3\texttt{)))}} \arrow[d] \\
\texttt{([] }v_2\texttt{)} \arrow[d] \\
\texttt{(}v_1\texttt{ [])}
\end{tikzcd}$$

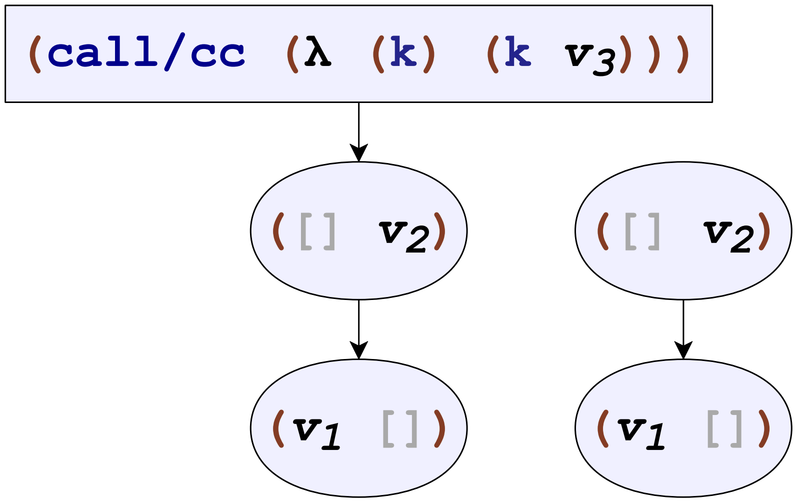
Continuations



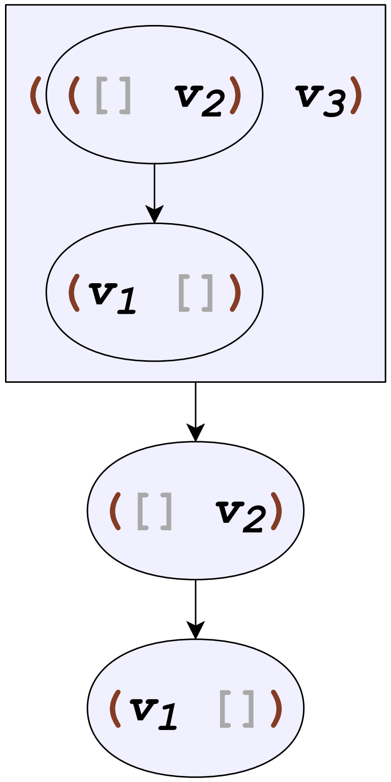
Continuations



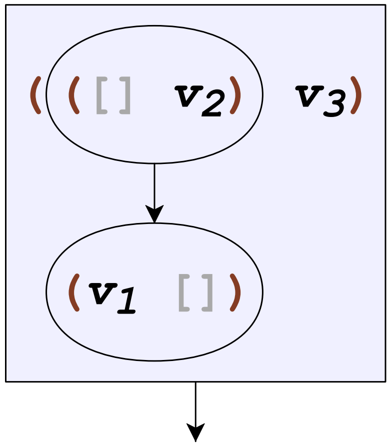
Continuations



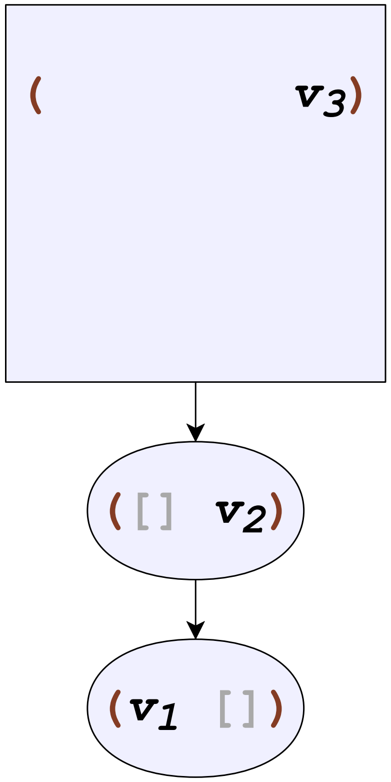
Continuations



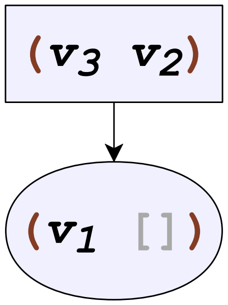
Continuations



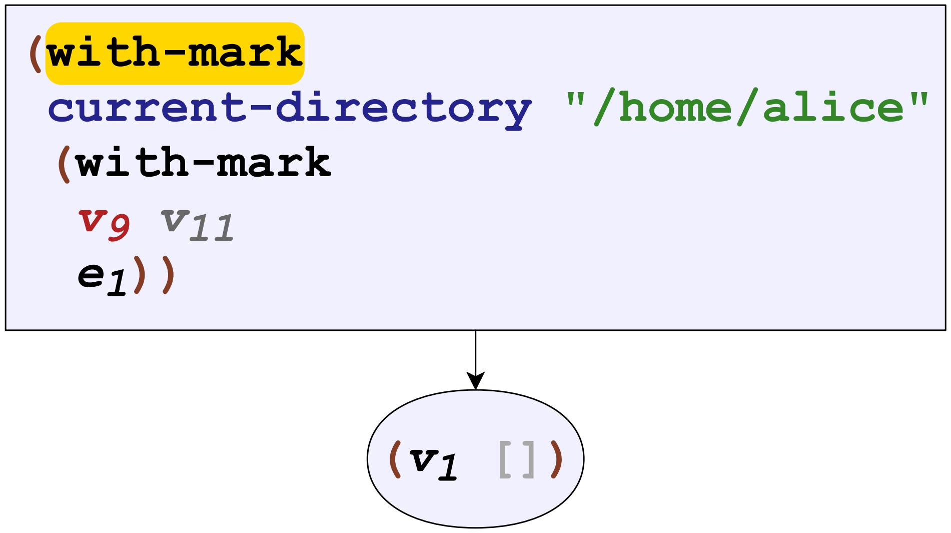
Continuations



Continuations



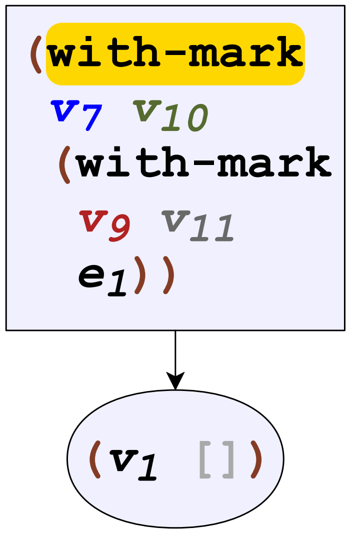
Continuation Marks



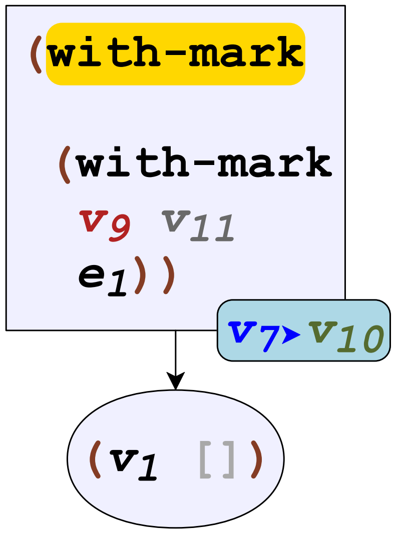
```
(with-mark
 current-directory "/home/alice"
 (with-mark
  v9 v11
  e1))
```

```
(v1 [])
```

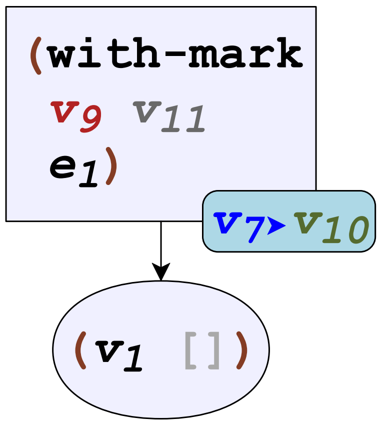
Continuation Marks



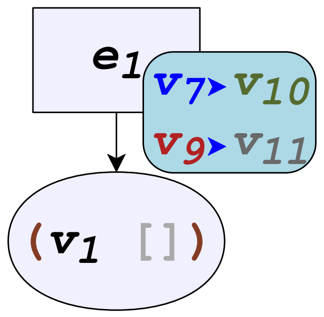
Continuation Marks



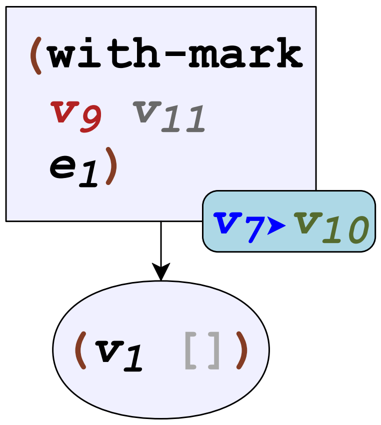
Continuation Marks



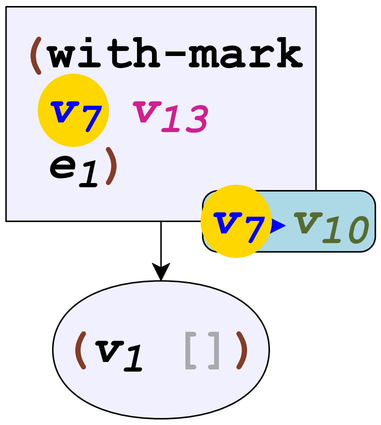
Continuation Marks


e_1
$v_7 \blacktriangleright v_{10}$
$v_9 \blacktriangleright v_{11}$
$(v_1\ [\,])$

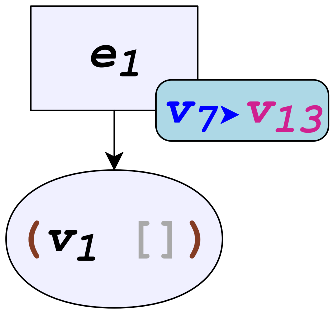
Continuation Marks



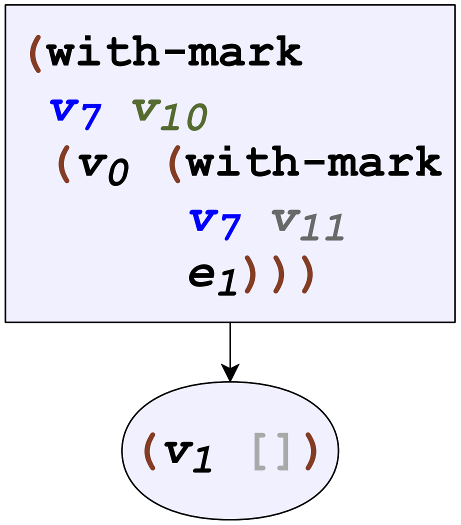
Continuation Marks



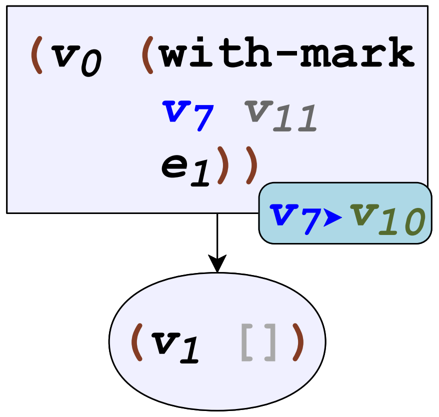
Continuation Marks



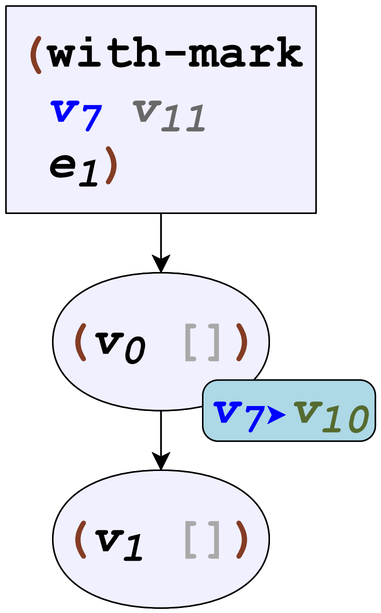
Continuation Marks



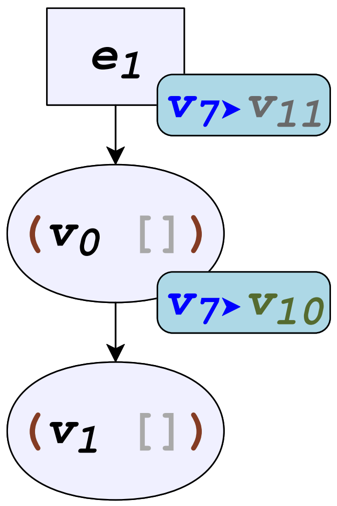
Continuation Marks



Continuation Marks

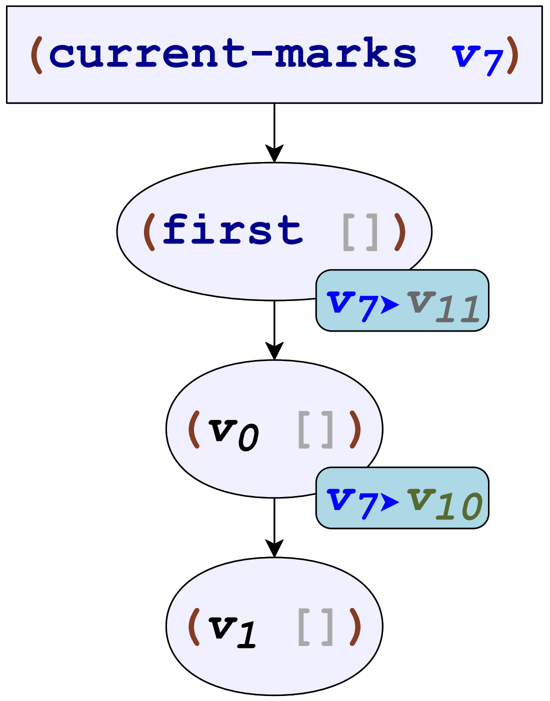


Continuation Marks

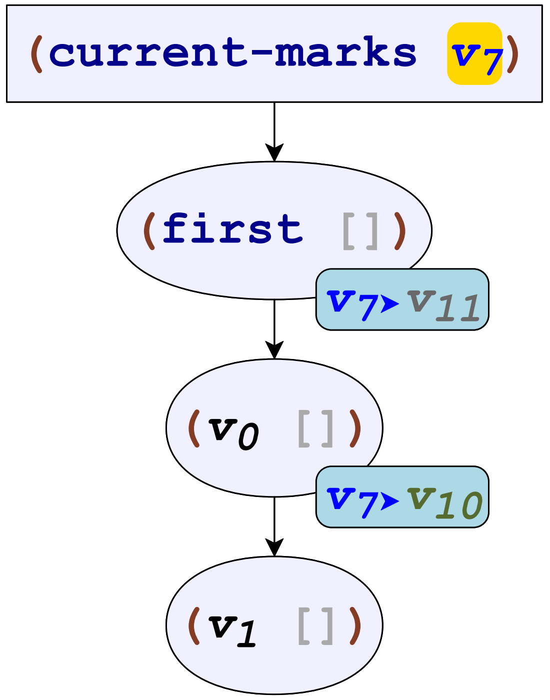


e_1
$v_7 \blacktriangleright v_{11}$
$(v_0\ [\,])$
$v_7 \blacktriangleright v_{10}$
$(v_1\ [\,])$

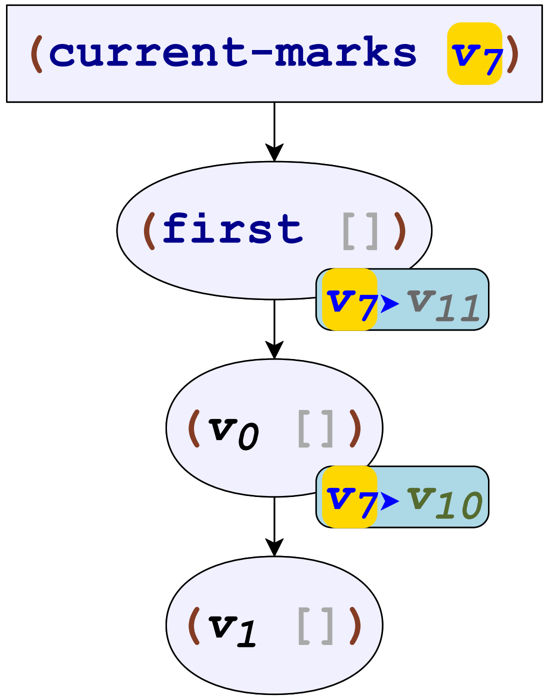
Continuation Marks



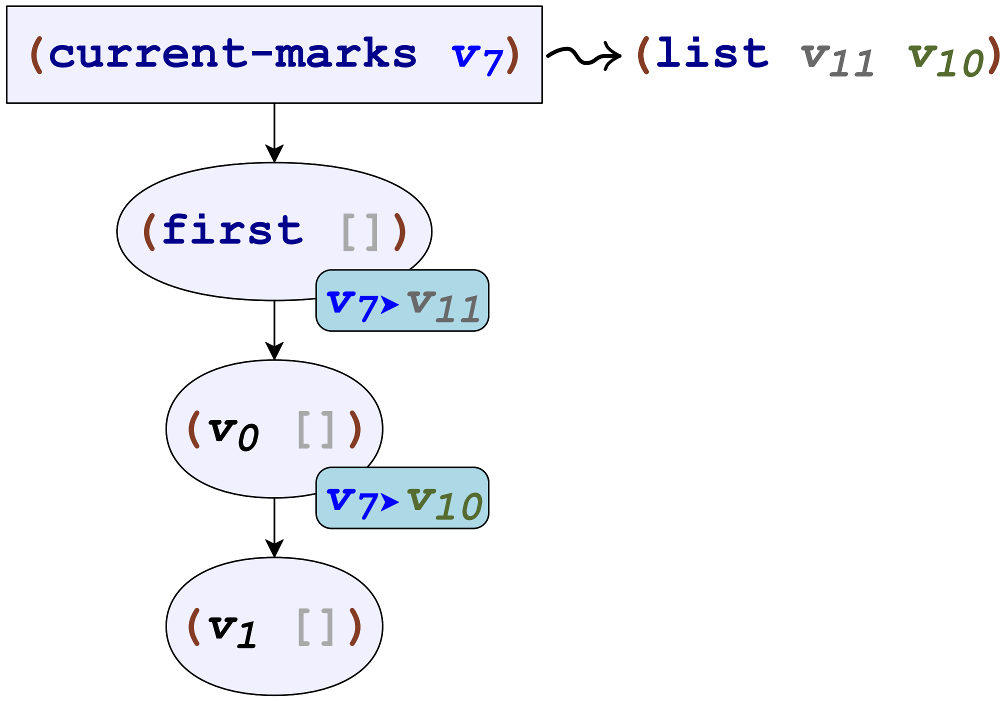
Continuation Marks



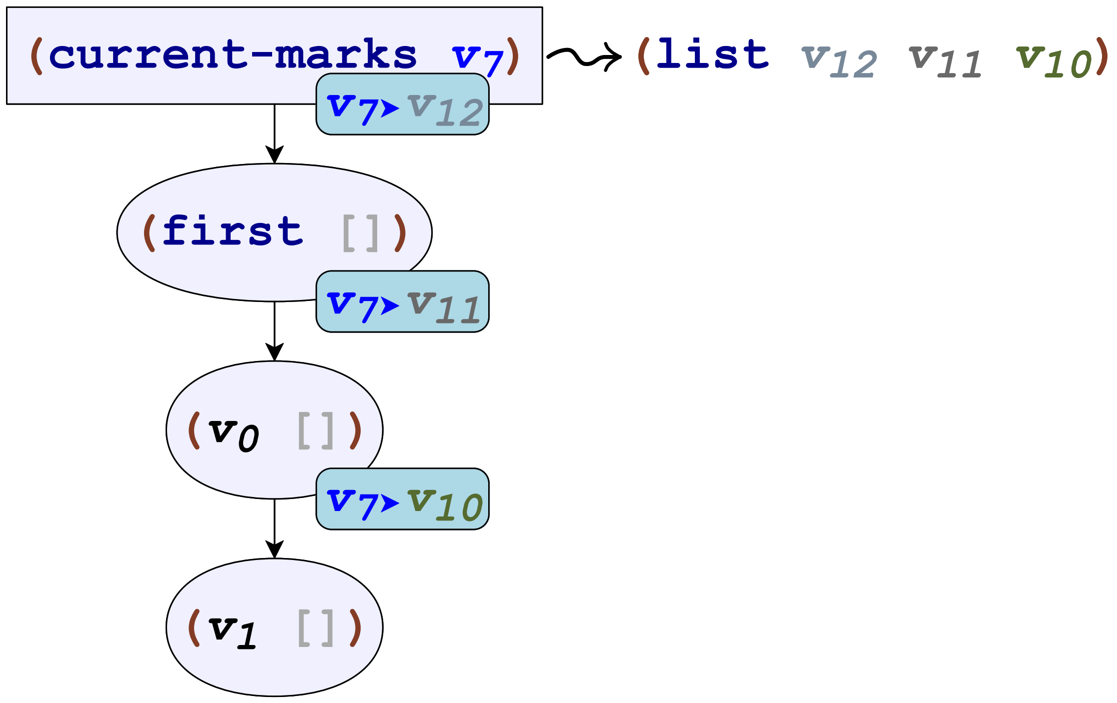
Continuation Marks



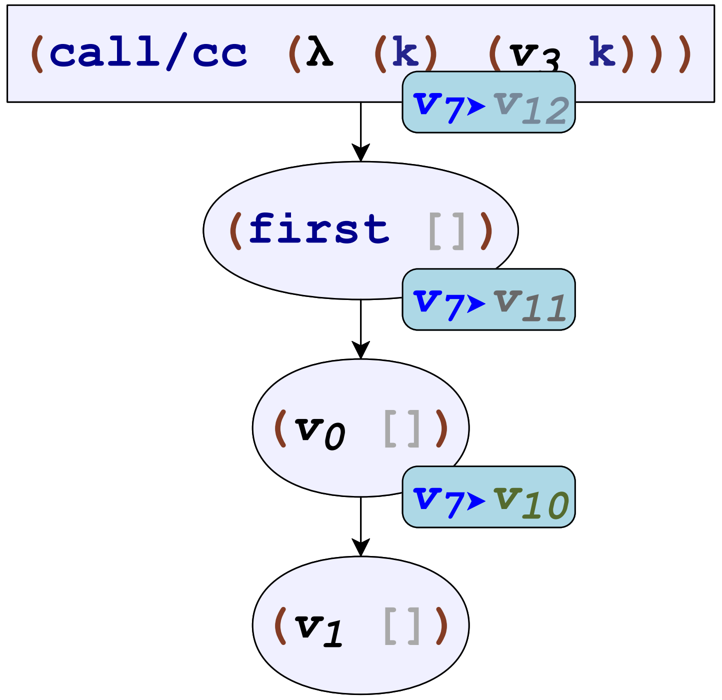
Continuation Marks



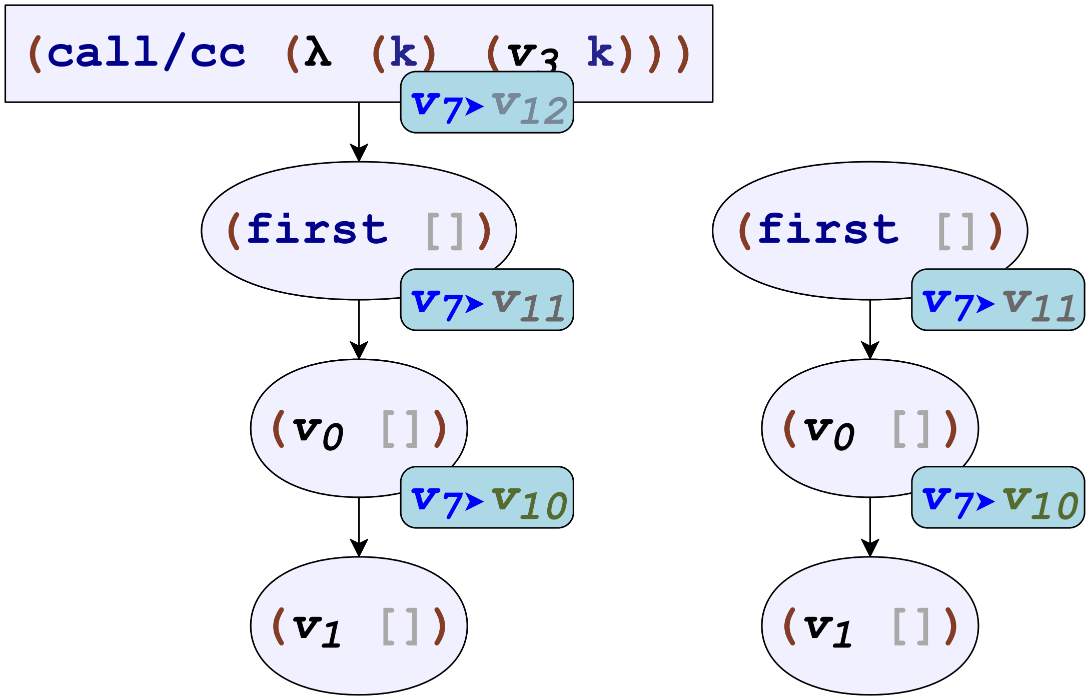
Continuation Marks



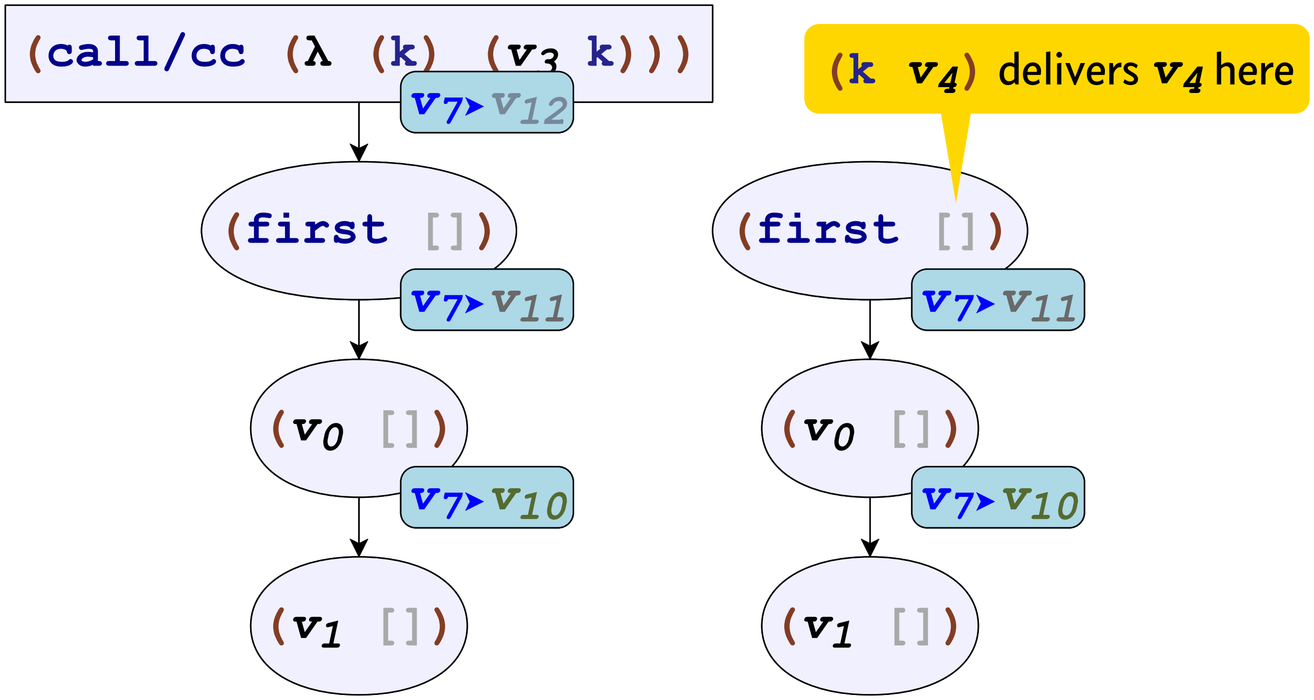
Capturing Marks



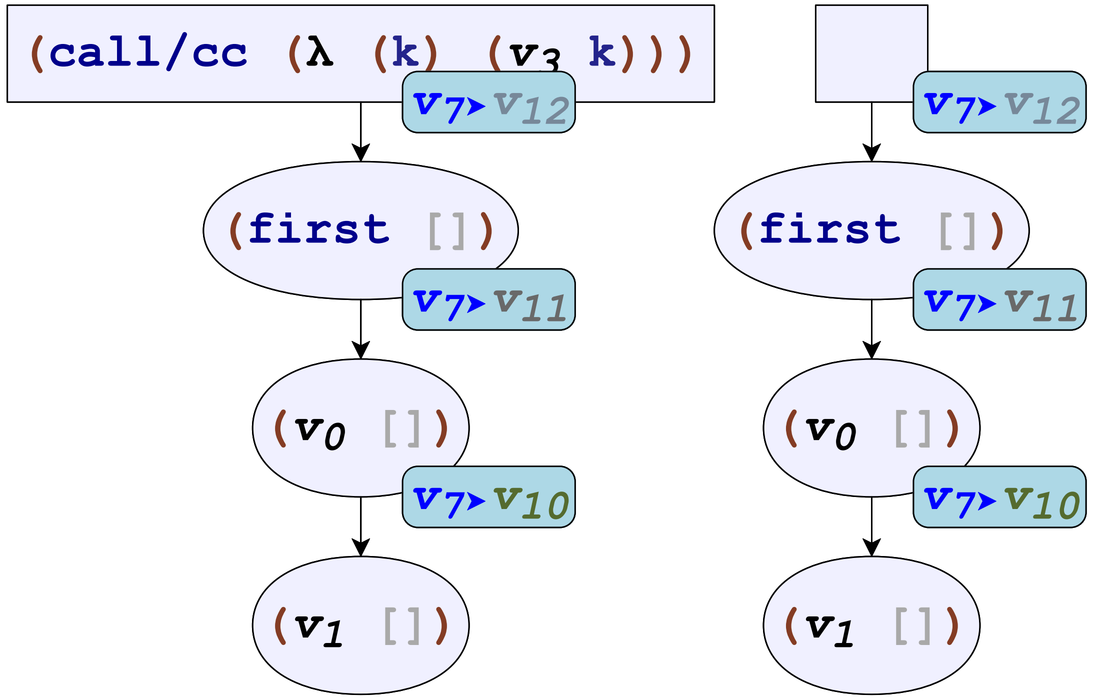
Capturing Marks



Capturing Marks

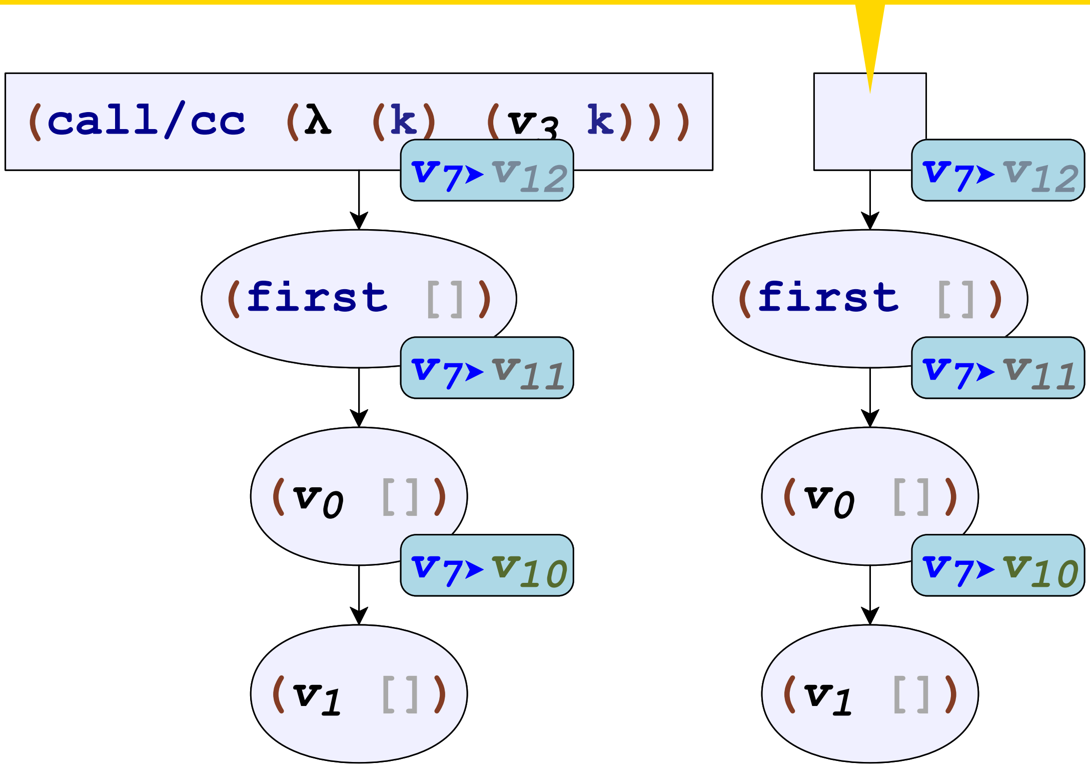


Capturing Marks

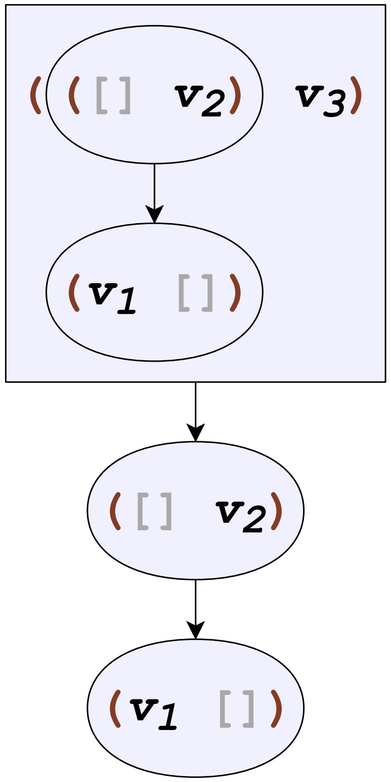


Capturing Marks

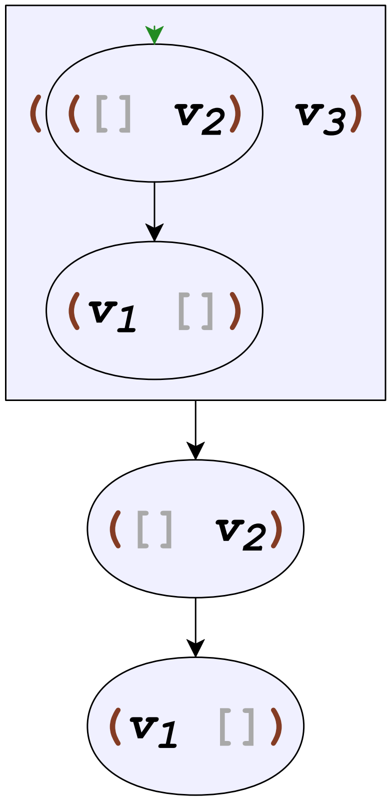
`(call-in-continuation k (λ () e₅))` delivers e_5 here



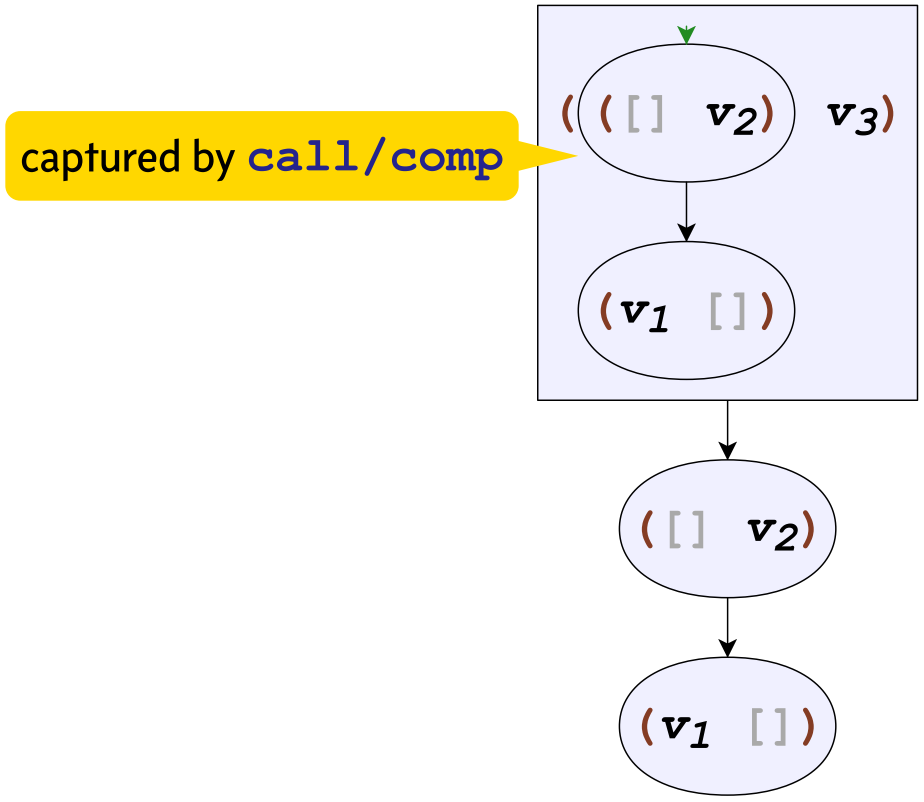
Continuations



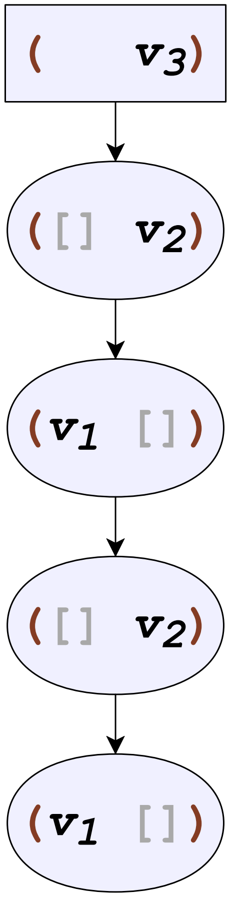
Composable Continuations



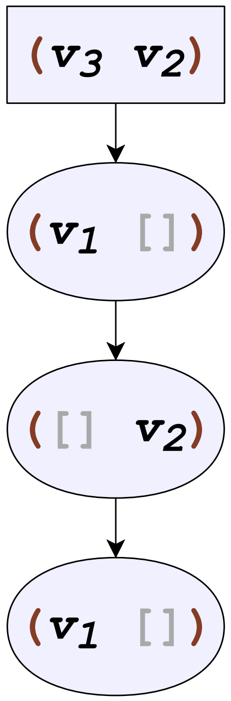
Composable Continuations



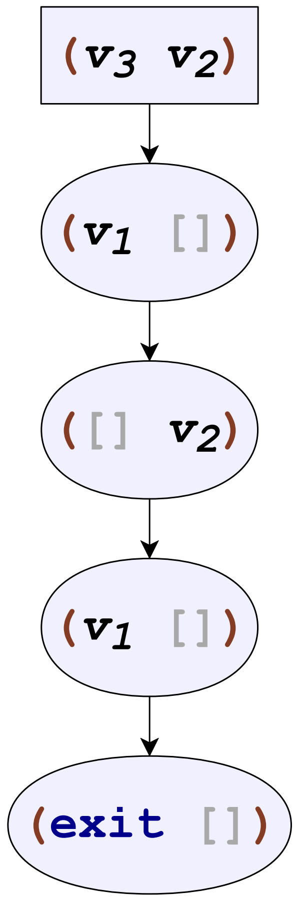
Composable Continuations



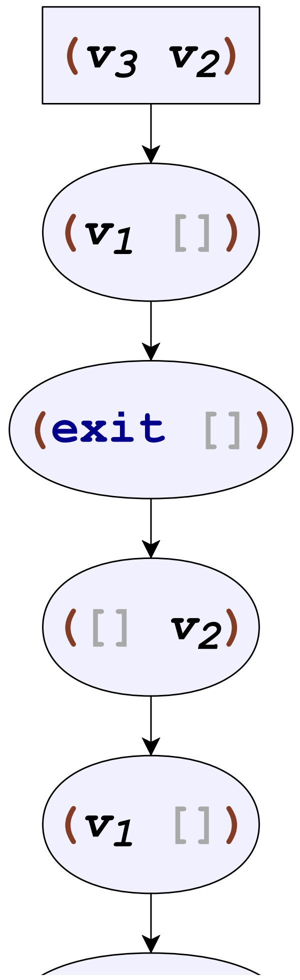
Composable Continuations



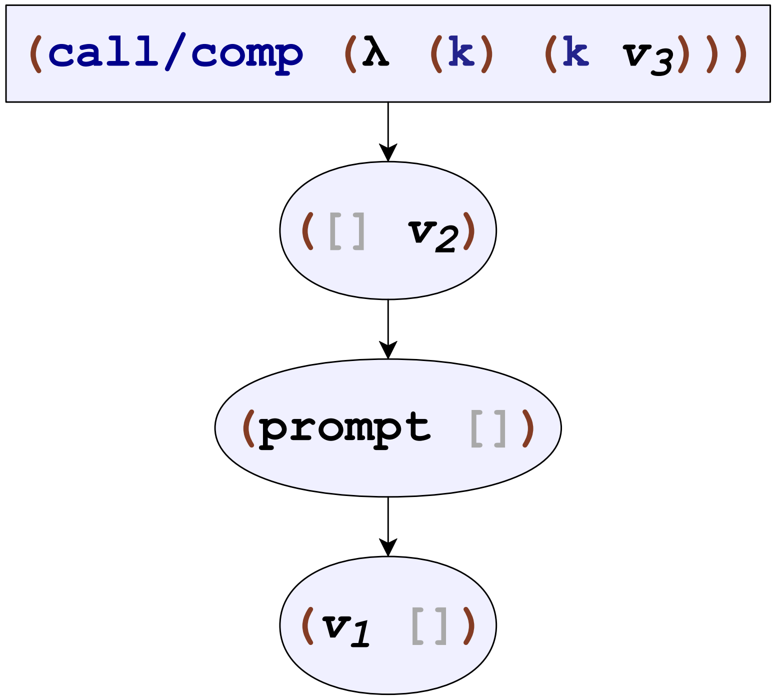
Composable Continuations



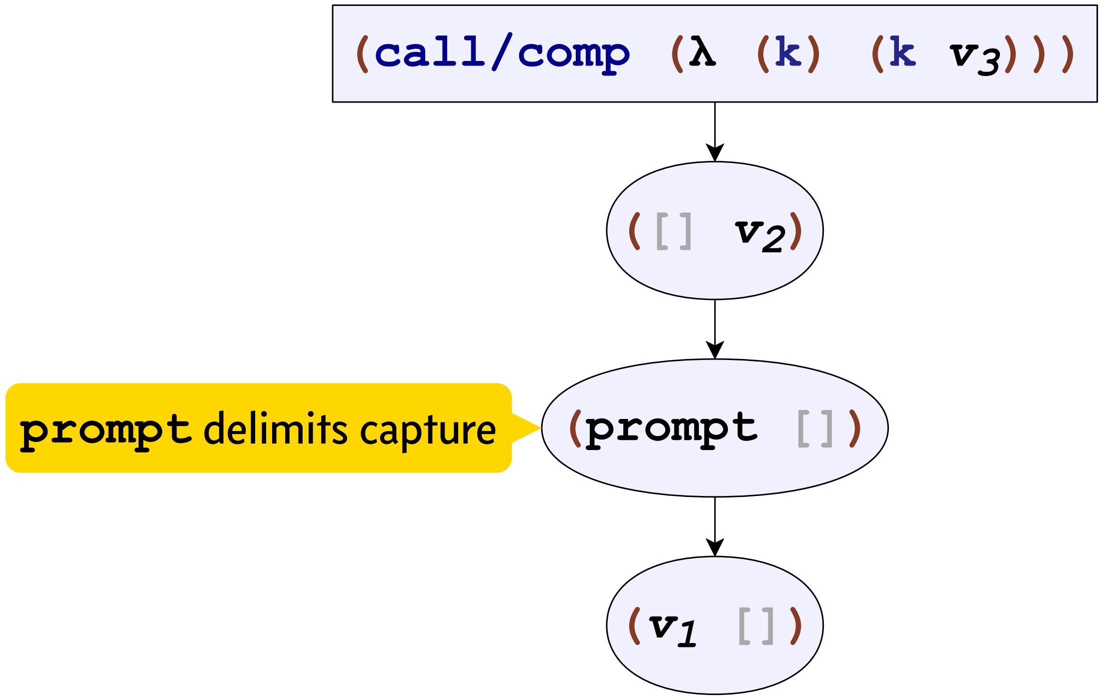
Composable Continuations



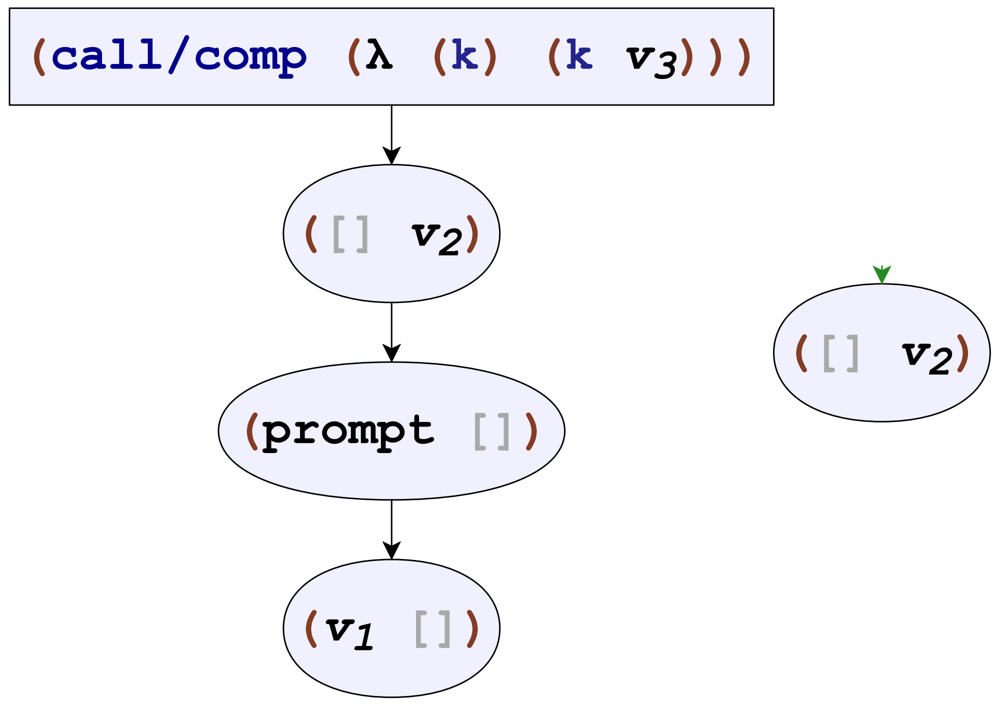
Delimited Capture



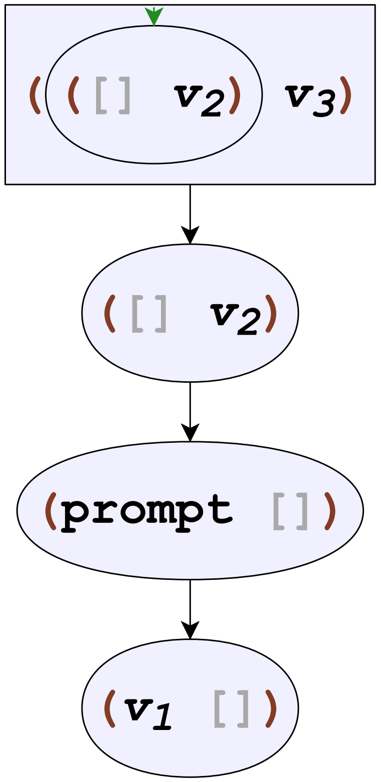
Delimited Capture



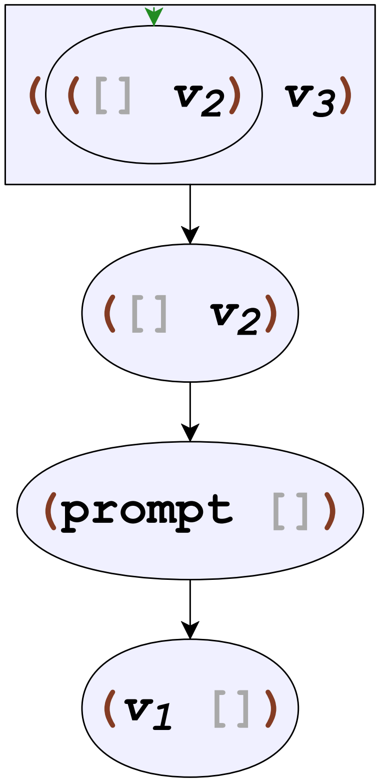
Delimited Capture



Delimited Capture

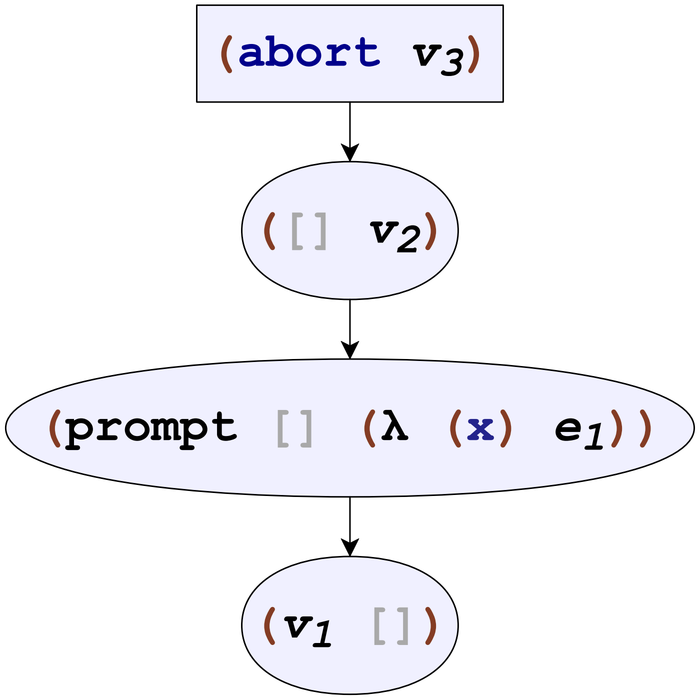


Delimited Capture



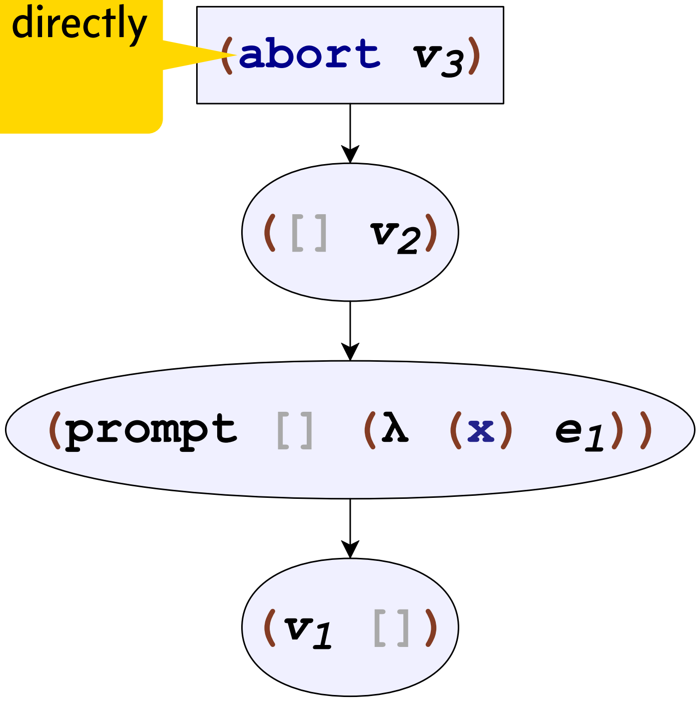
Practically all uses of continuations are better as **delimited**

Abort and Prompt Handlers

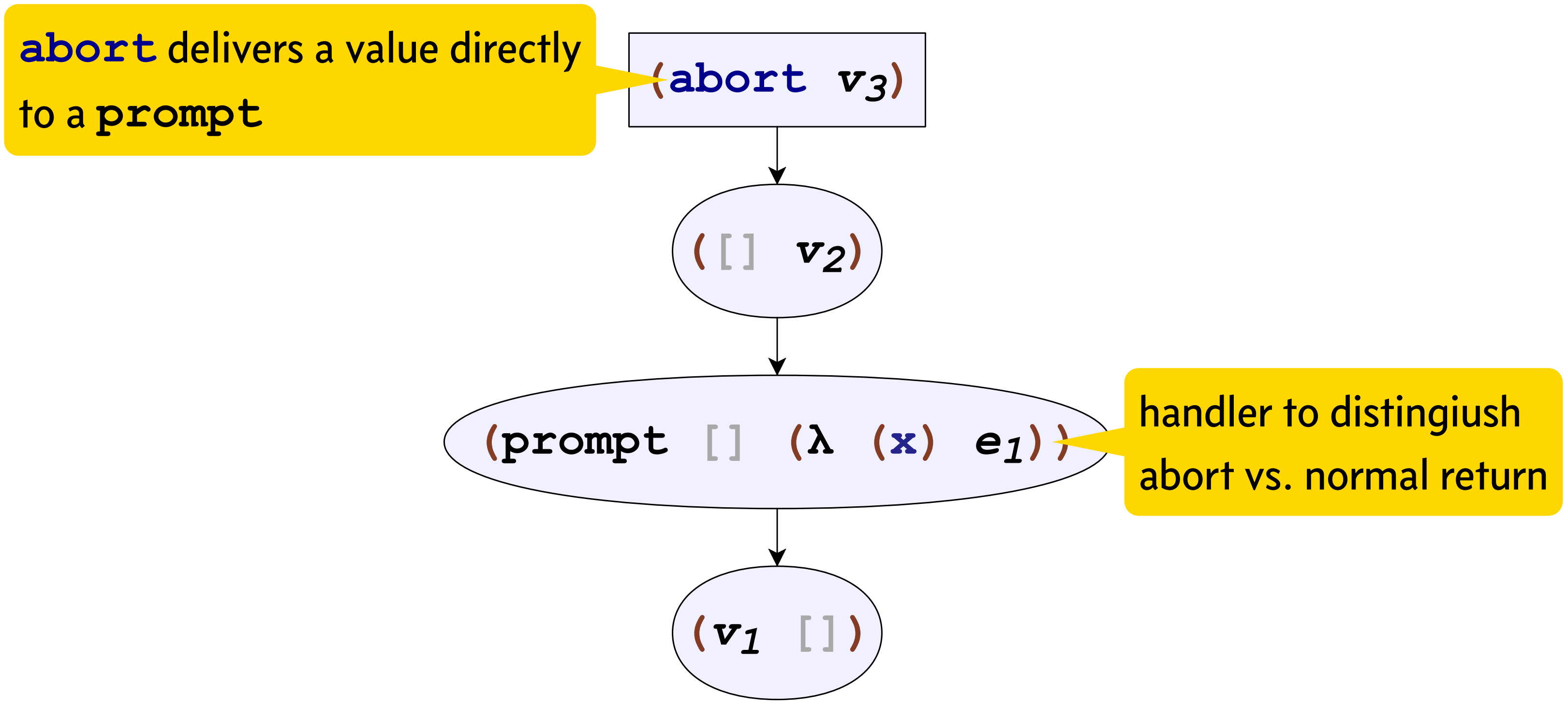


Abort and Prompt Handlers

abort delivers a value directly to a **prompt**



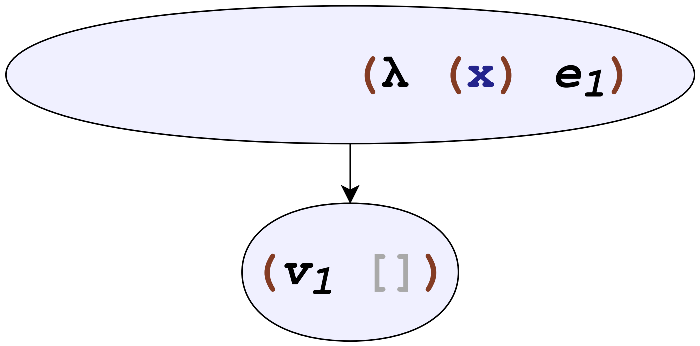
Abort and Prompt Handlers



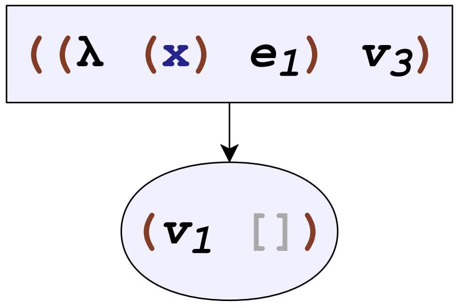
abort delivers a value directly to a prompt
(abort v3)
([] v2)
(prompt [] (λ (x) e1))
handler to distingiush abort vs. normal return
(v1 [])

Abort and Prompt Handlers

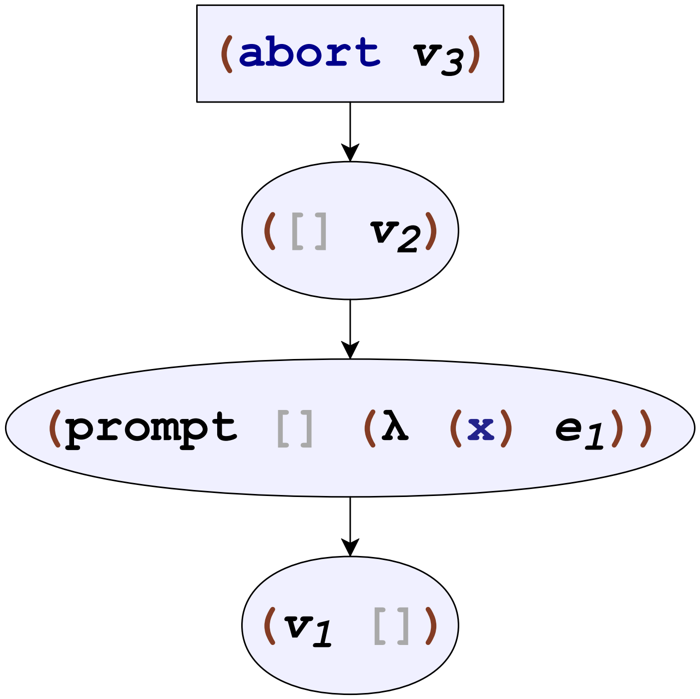
`v3`



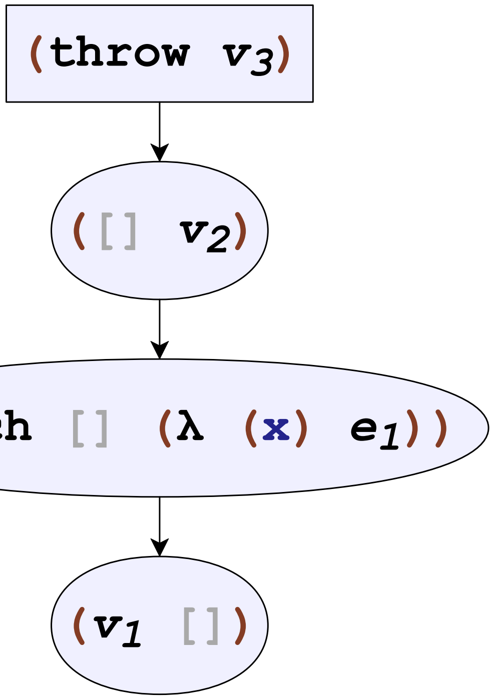
Abort and Prompt Handlers



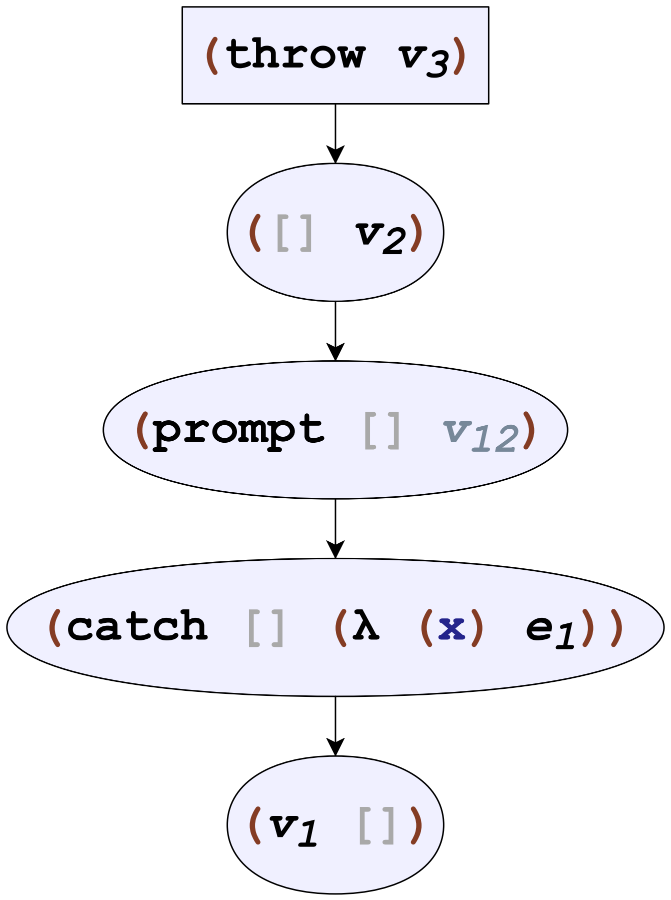
Abort and Prompt Handlers



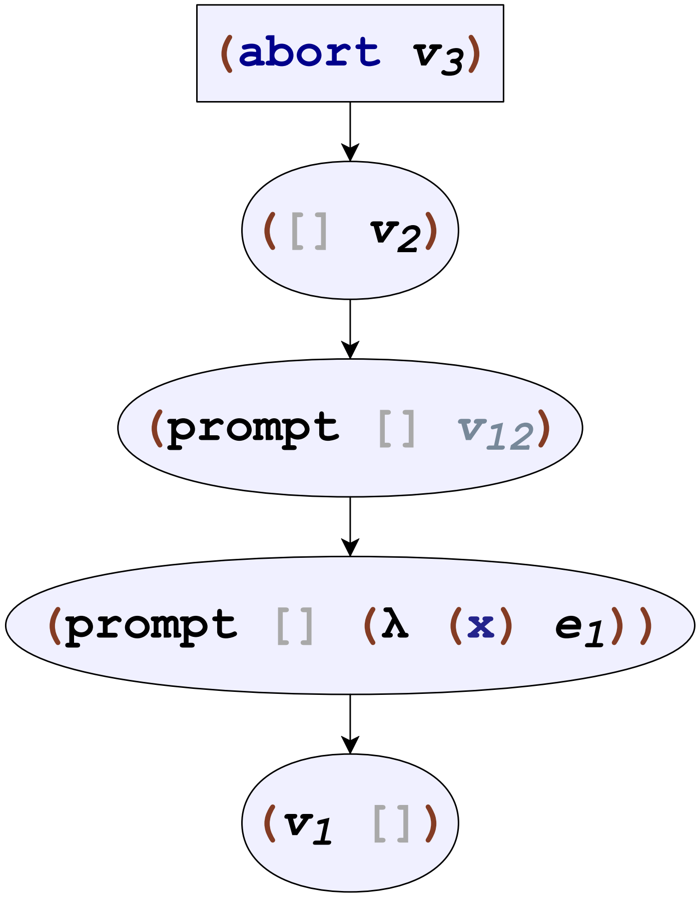
Implementing Catch and Throw



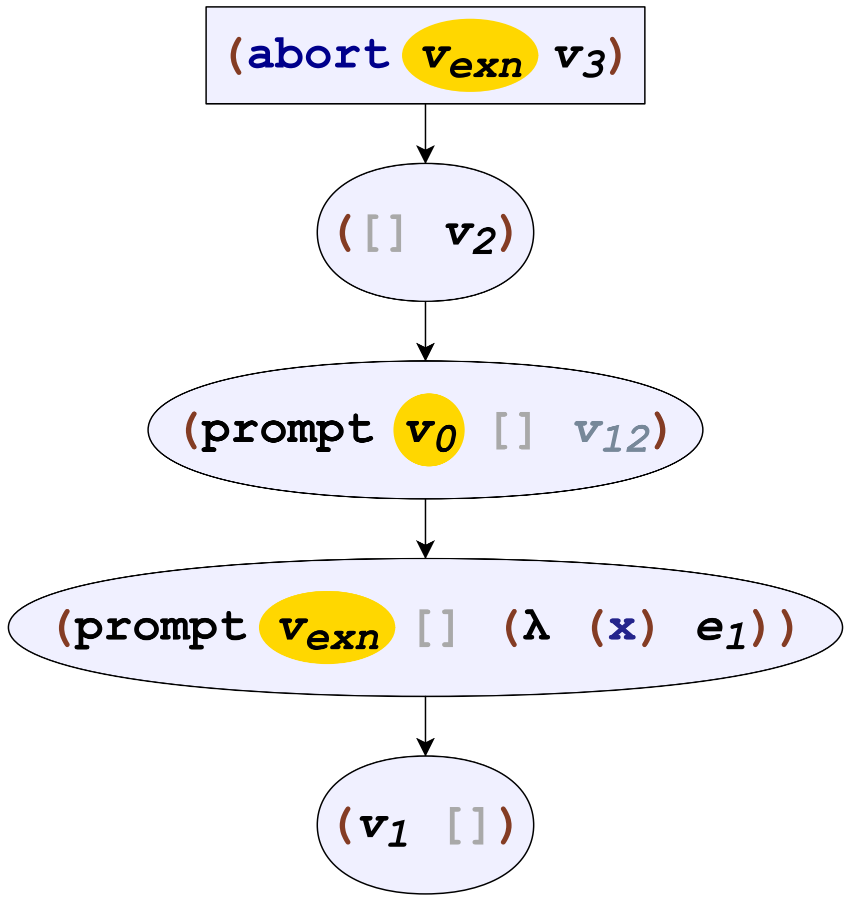
Implementing Catch and Throw



Implementing Catch and Throw

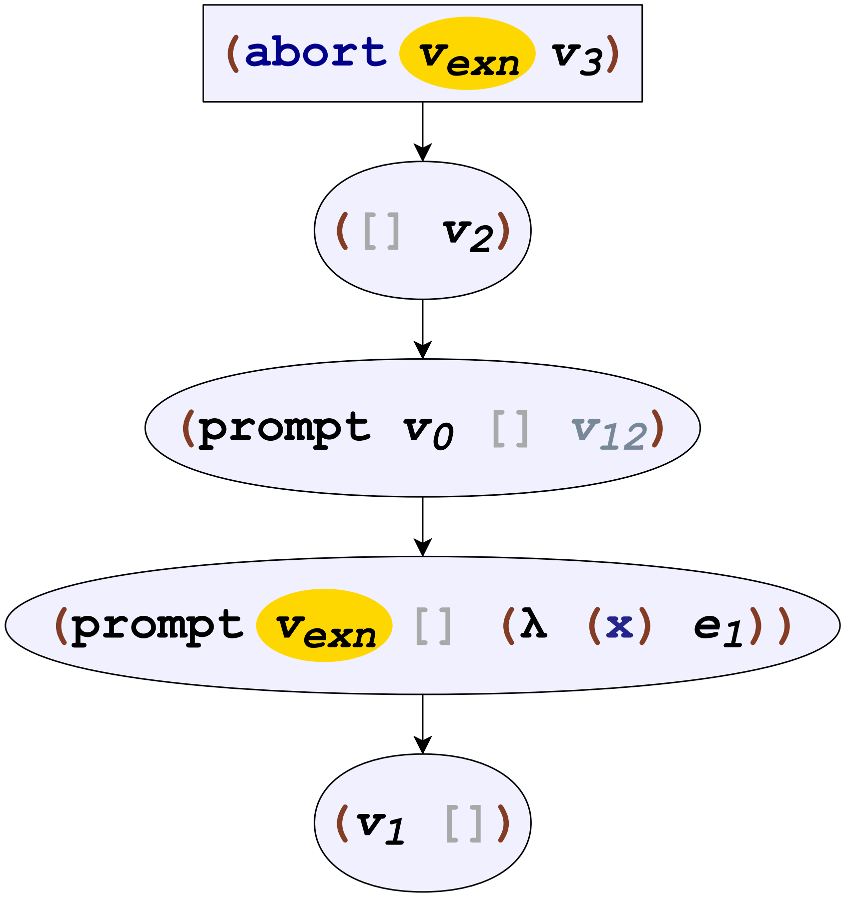


Implementing Catch and Throw

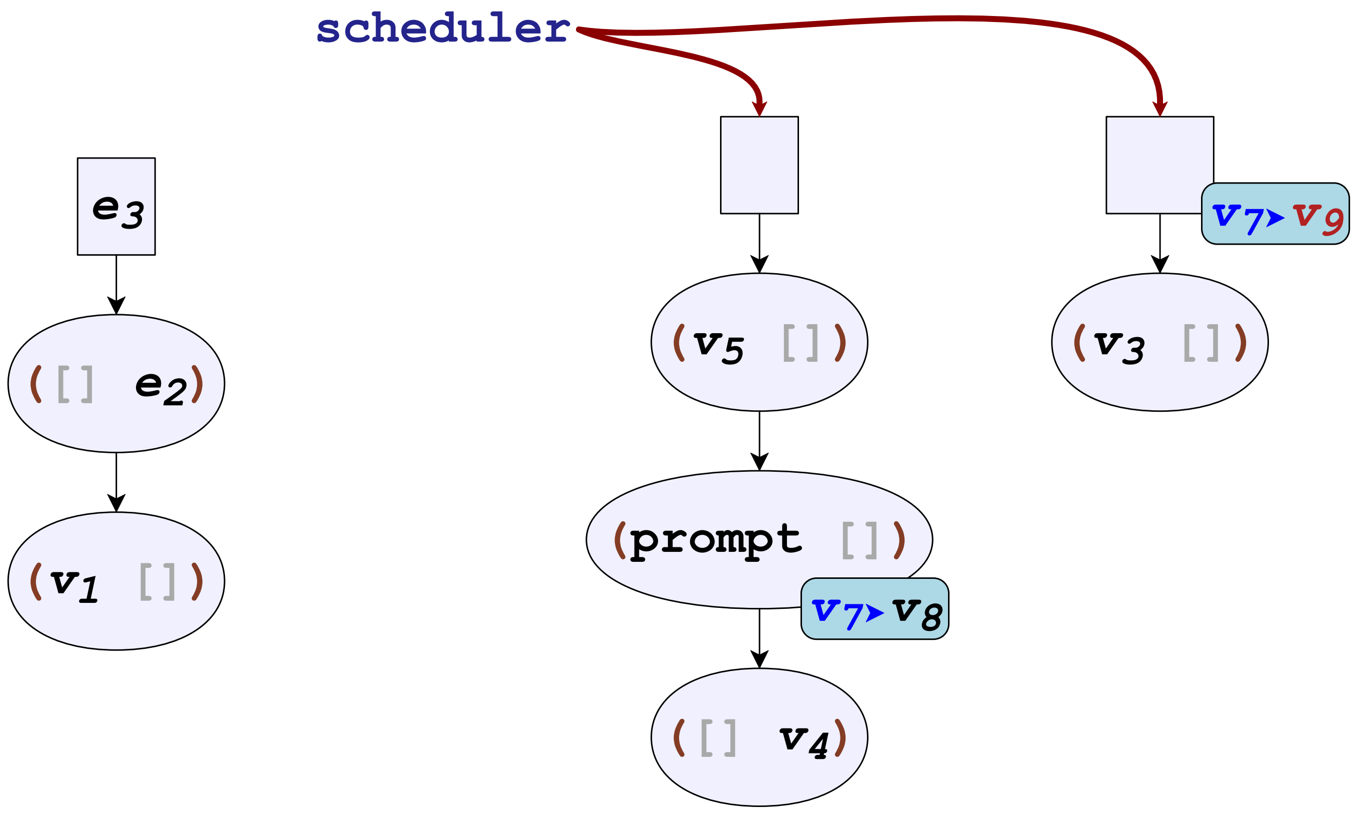

(abort v_{exn} v_3)
([] v_2)
(prompt v_0 [] v_{12})
(prompt v_{exn} [] (λ (x) e_1))
(v_1 [])

prompt **tags**
distinguish uses

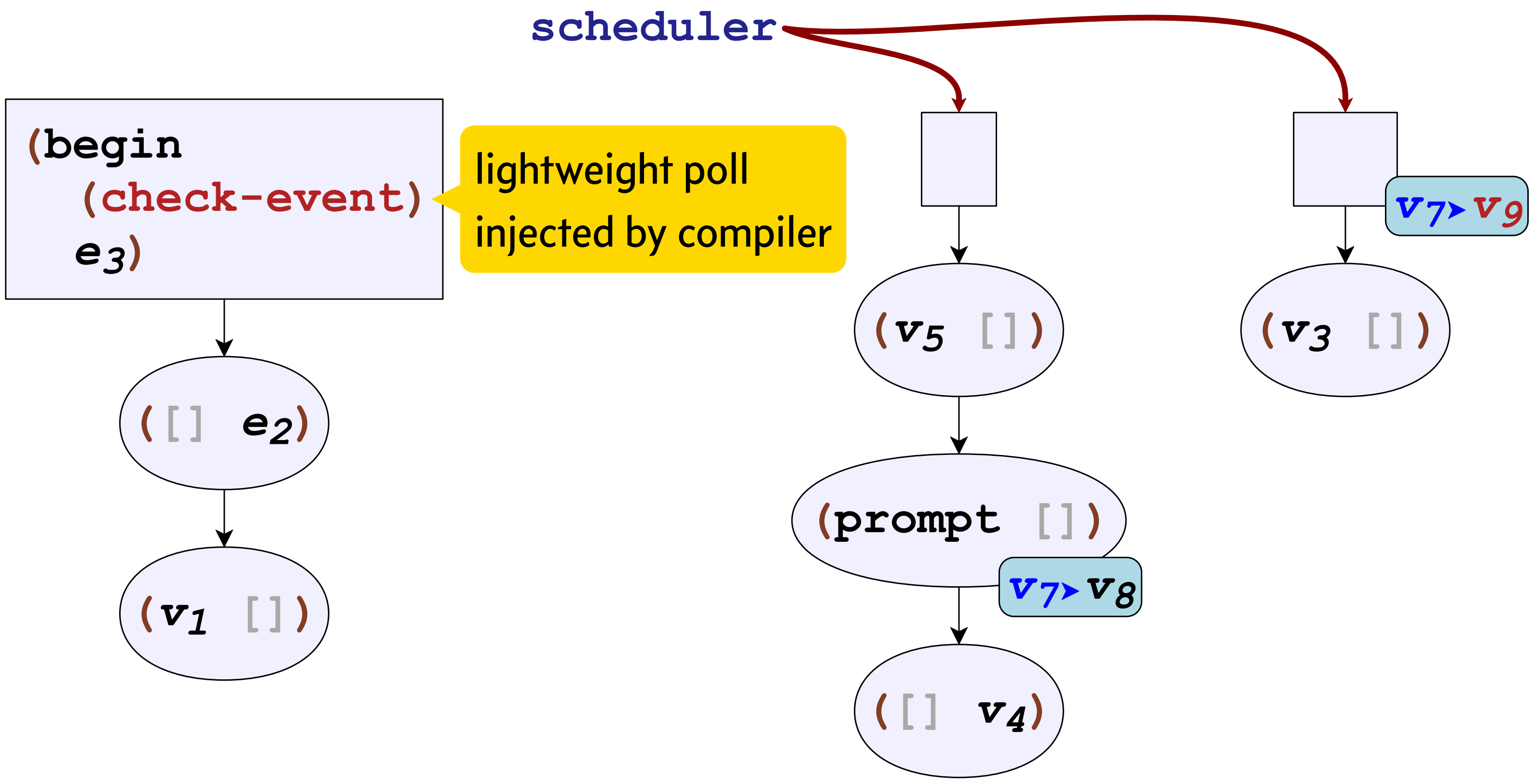
Implementing Catch and Throw


(abort v_{exn} v_3)
([] v_2)
(prompt v_0 [] v_{12})
(prompt v_{exn} [] (λ (x) e_1))
(v_1 [])

Threads



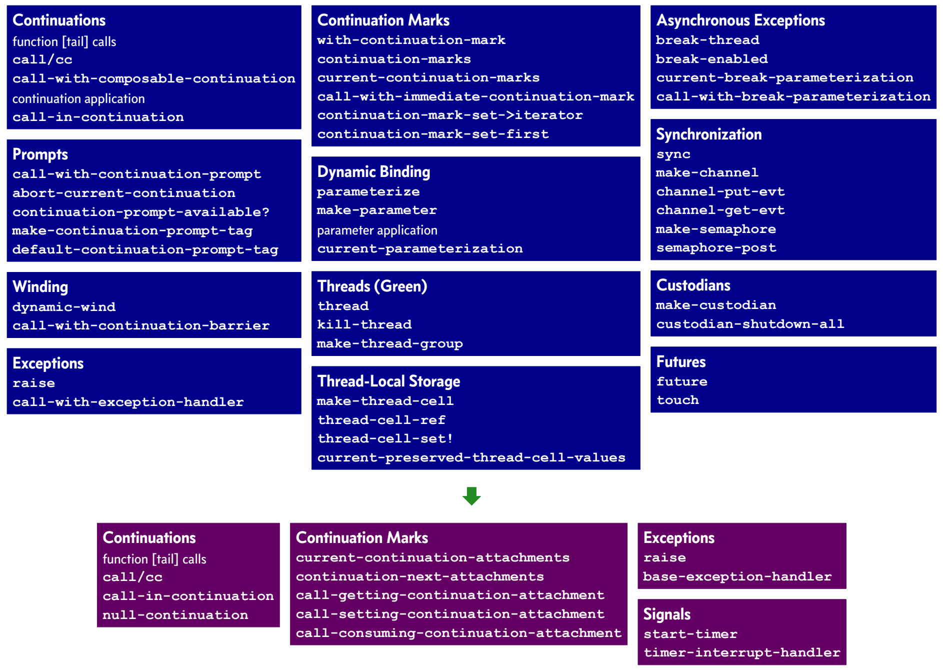
Threads



scheduler
(begin
(check-event)
e3)
lightweight poll
injected by compiler
([] e2)
(v1 [])
(v5 [])
(prompt [])
v7➤v8
([] v4)
v7➤v9
(v3 [])

Part 2 of 2

Implementation



Continuations
function [tail] calls
`call/cc`
`call-with-composable-continuation`
continuation application
`call-in-continuation`

Prompts
`call-with-continuation-prompt`
`abort-current-continuation`
`continuation-prompt-available?`
`make-continuation-prompt-tag`
`default-continuation-prompt-tag`

Winding
`dynamic-wind`
`call-with-continuation-barrier`

Exceptions
`raise`
`call-with-exception-handler`

Continuation Marks
`with-continuation-mark`
`continuation-marks`
`current-continuation-marks`
`call-with-immediate-continuation-mark`
`continuation-mark-set->iterator`
`continuation-mark-set-first`

Dynamic Binding
`parameterize`
`make-parameter`
parameter application
`current-parameterization`

Threads (Green)
`thread`
`kill-thread`
`make-thread-group`

Thread-Local Storage
`make-thread-cell`
`thread-cell-ref`
`thread-cell-set!`
`current-preserved-thread-cell-values`

Asynchronous Exceptions
`break-thread`
`break-enabled`
`current-break-parameterization`
`call-with-break-parameterization`

Synchronization
`sync`
`make-channel`
`channel-put-evt`
`channel-get-evt`
`make-semaphore`
`semaphore-post`

Custodians
`make-custodian`
`custodian-shutdown-all`

Futures
`future`
`touch`

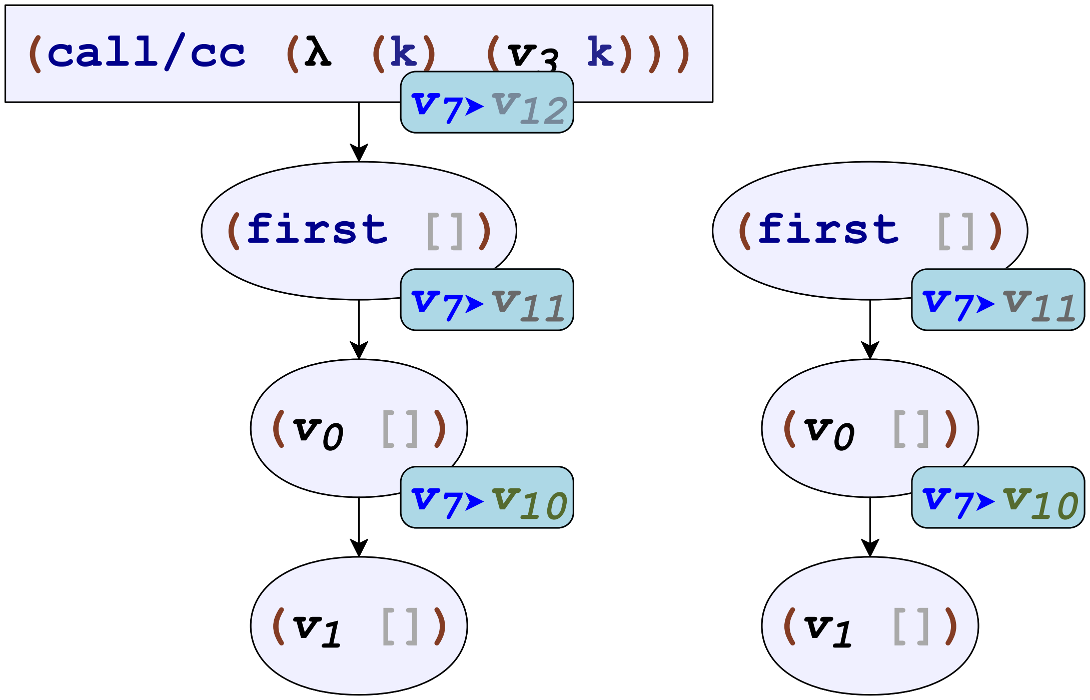
Continuations
function [tail] calls
`call/cc`
`call-in-continuation`
`null-continuation`

Continuation Marks
`current-continuation-attachments`
`continuation-next-attachments`
`call-getting-continuation-attachment`
`call-setting-continuation-attachment`
`call-consuming-continuation-attachment`

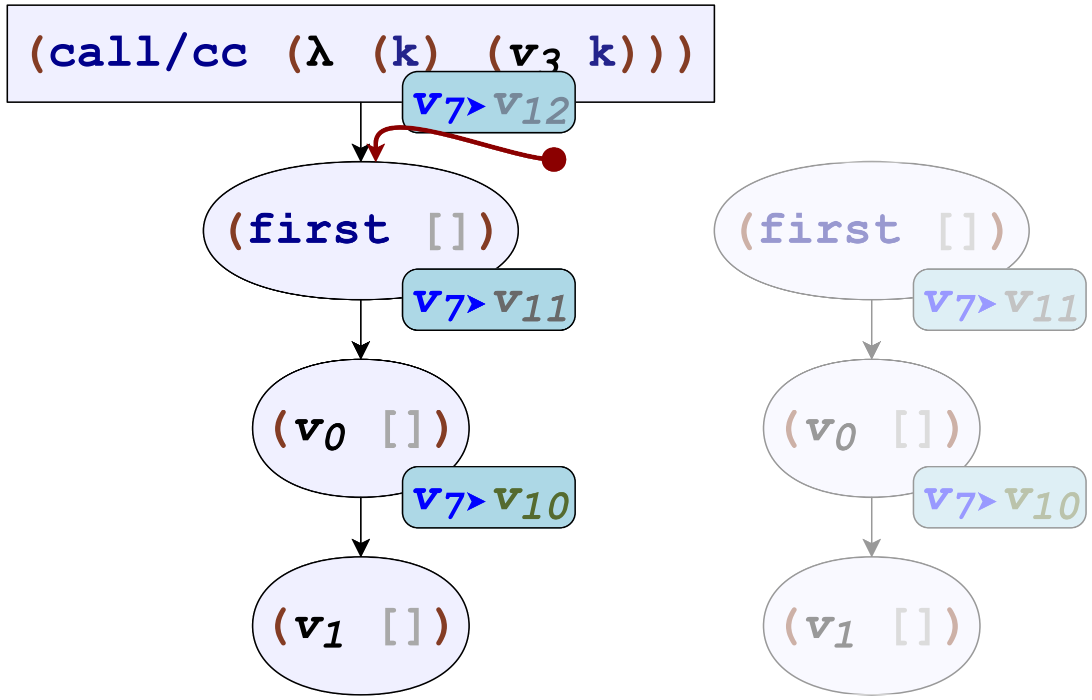
Exceptions
`raise`
`base-exception-handler`

Signals
`start-timer`
`timer-interrupt-handler`

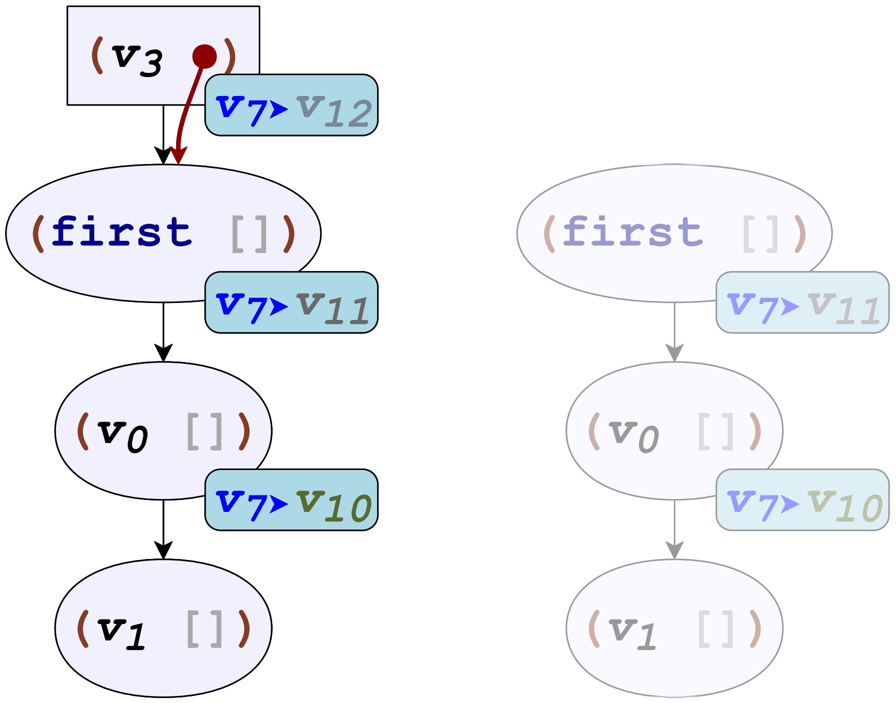
Heap-Allocated Frames



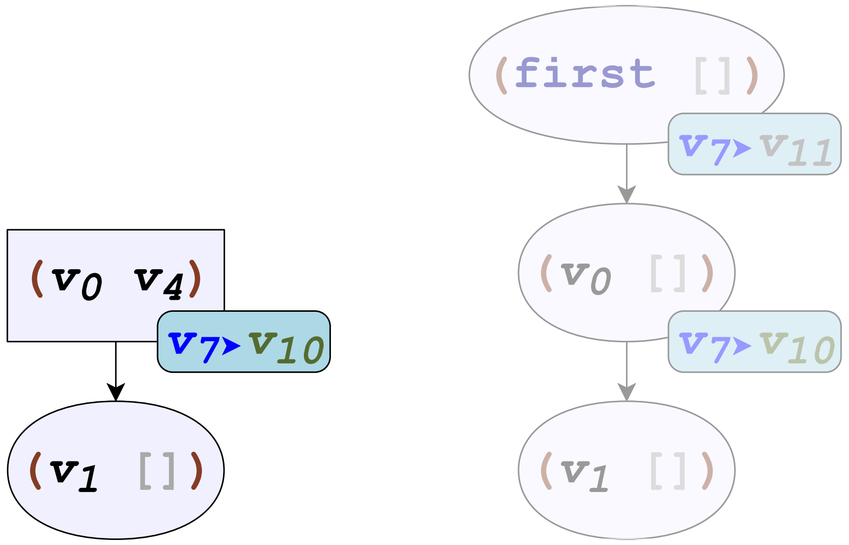
Heap-Allocated Frames



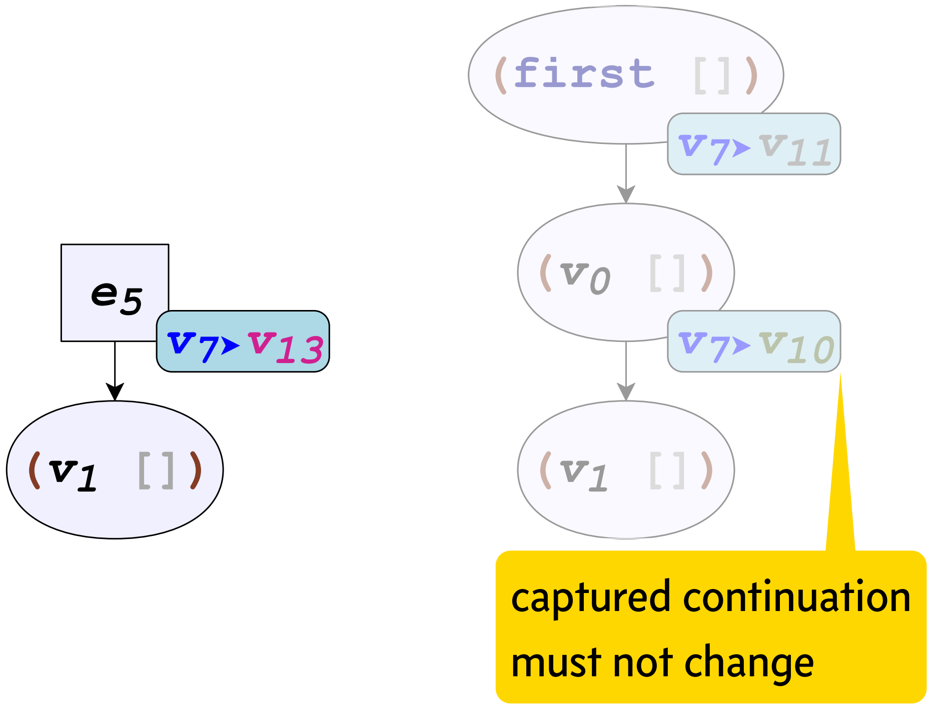
Heap-Allocated Frames



Heap-Allocated Frames

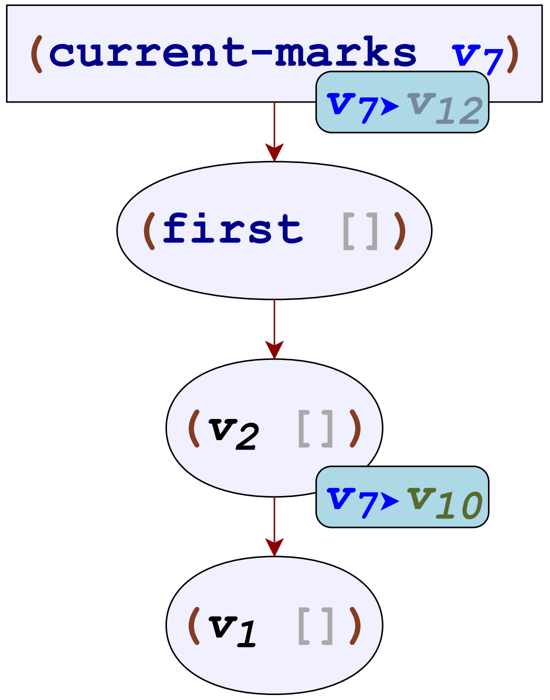


Heap-Allocated Frames

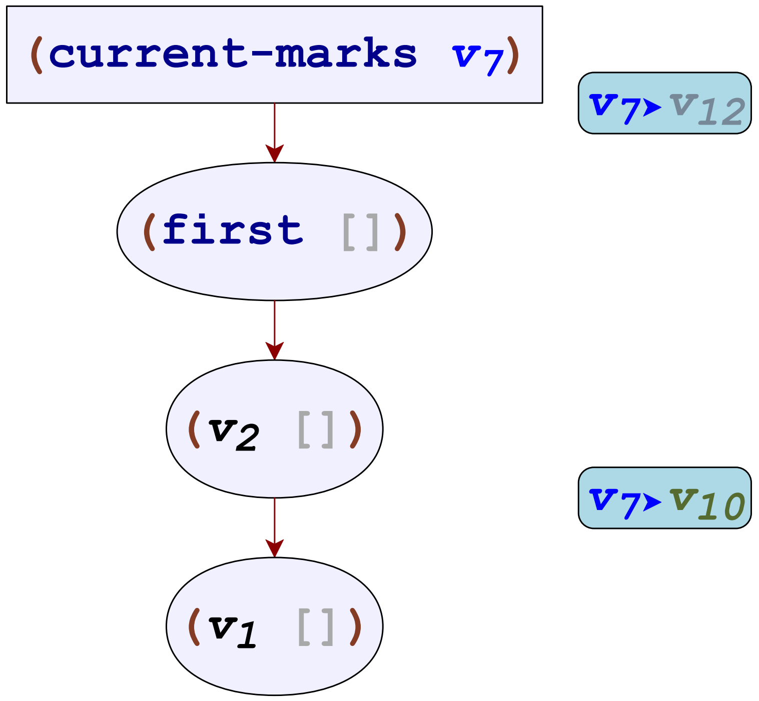


e_5
$v_7 \blacktriangleright v_{13}$
(v1 [])
(first [])
$v_7 \blacktriangleright v_{11}$
(v0 [])
$v_7 \blacktriangleright v_{10}$
(v1 [])
captured continuation must not change

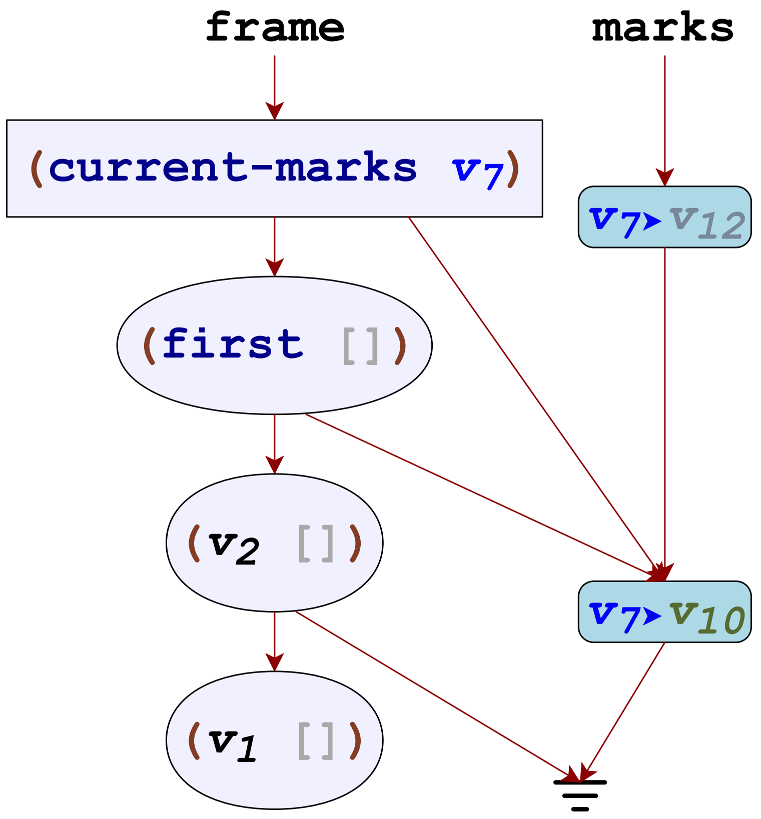
Functional Heap-Allocated Frames



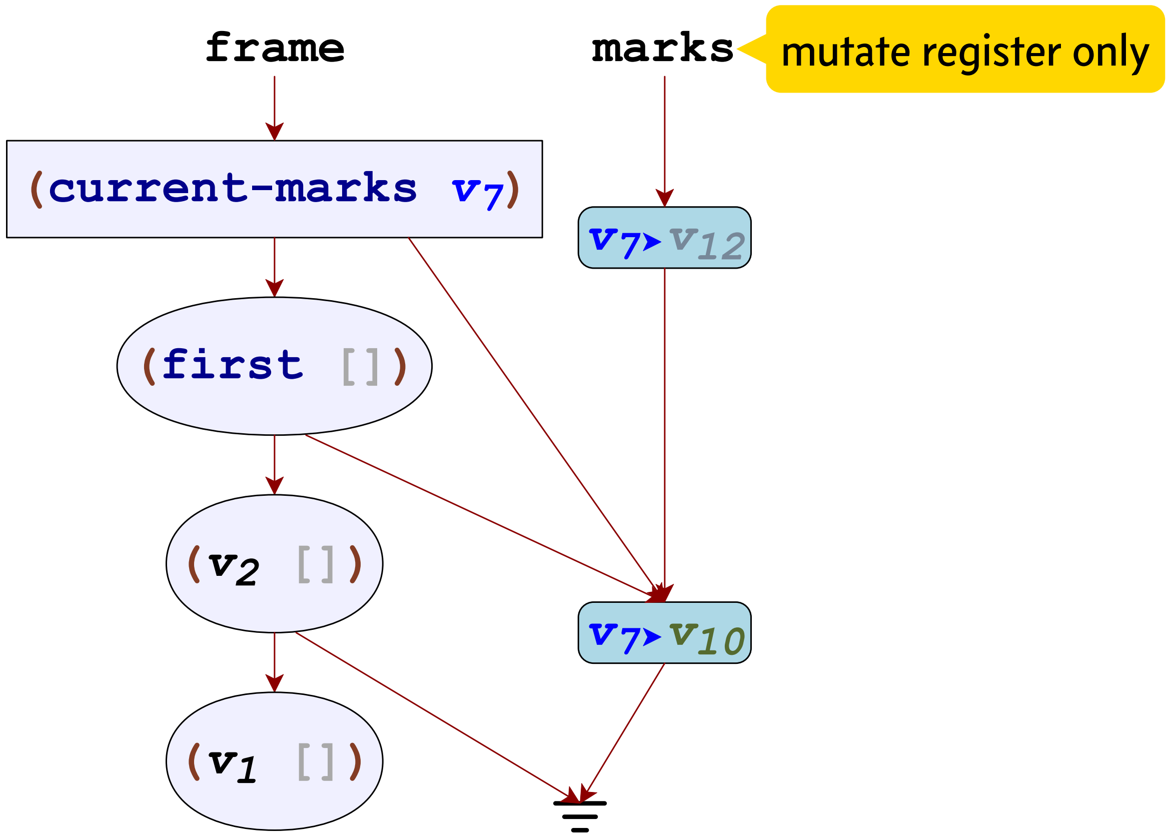
Functional Heap-Allocated Frames



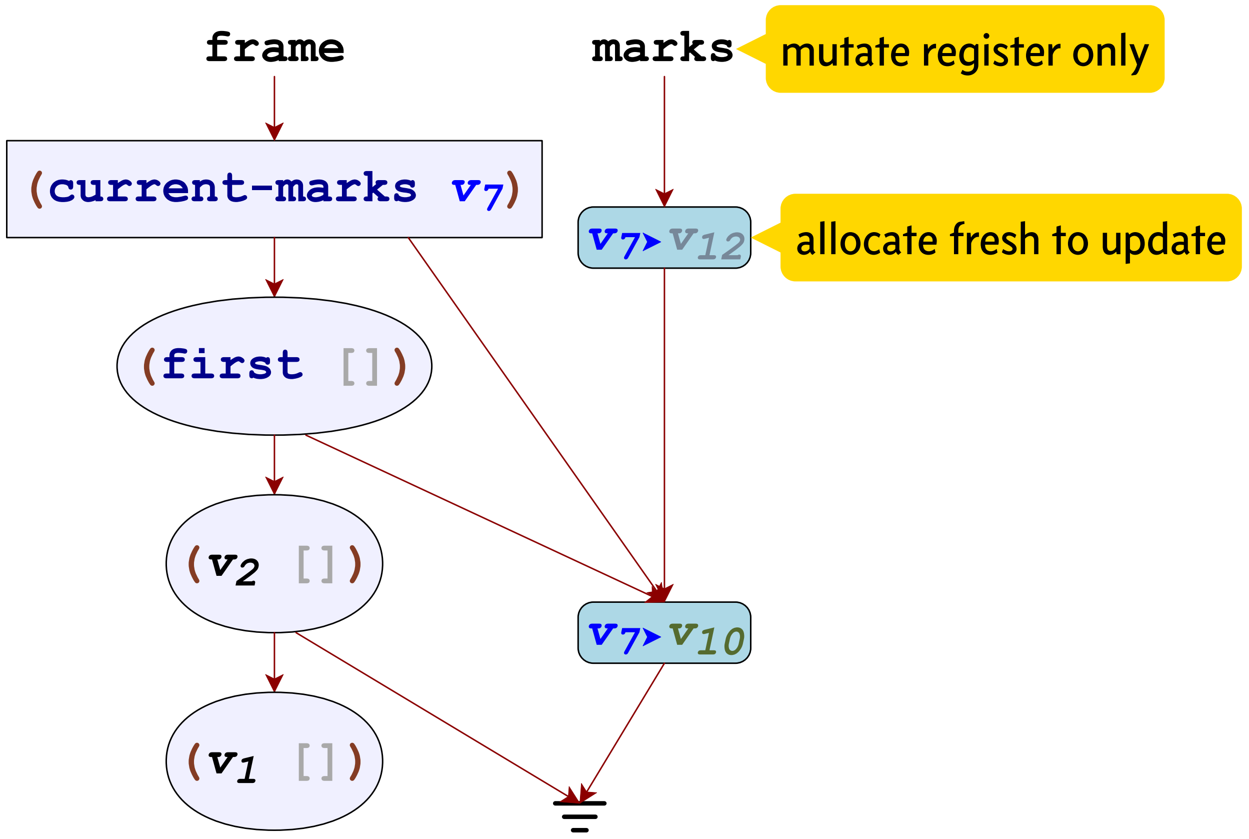
Functional Heap-Allocated Frames



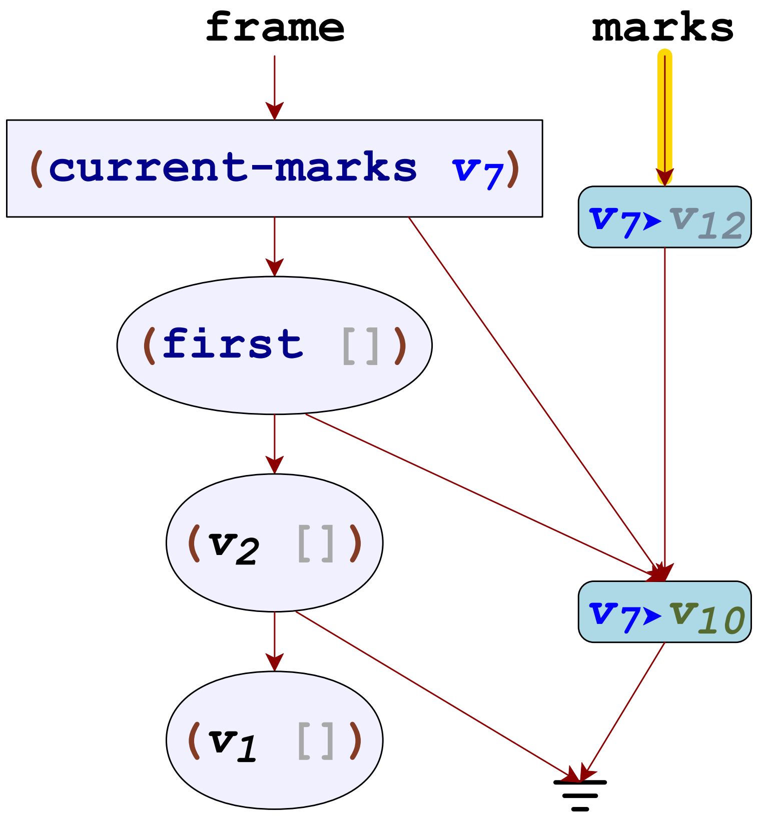
Functional Heap-Allocated Frames



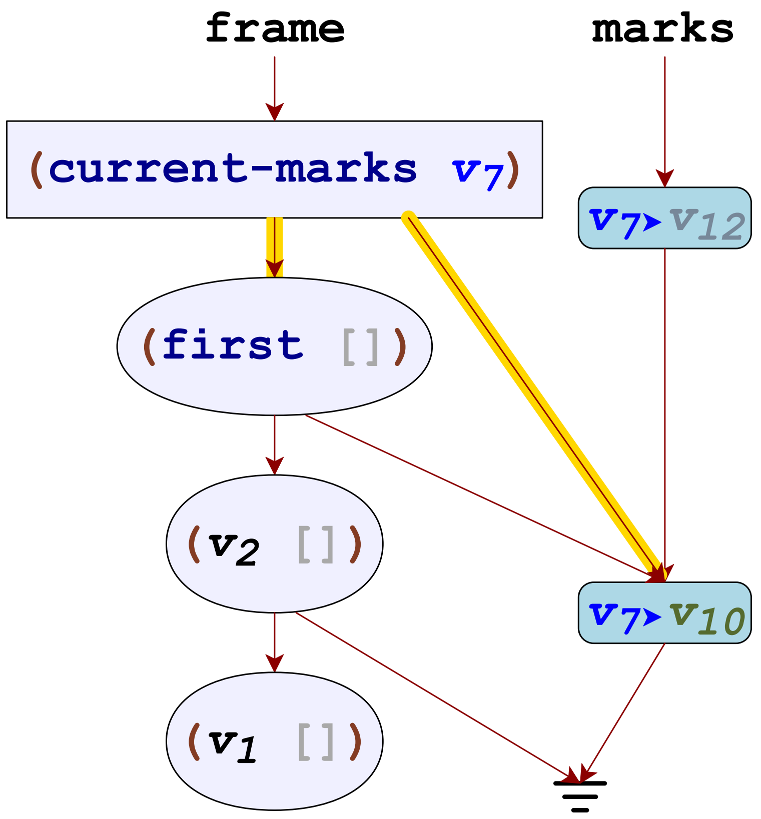
Functional Heap-Allocated Frames



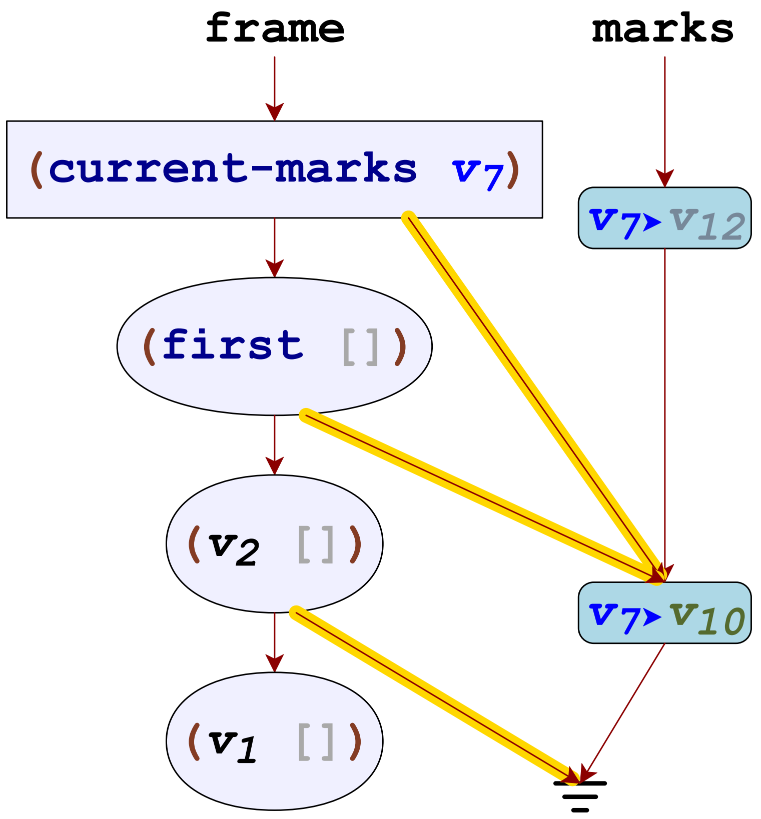
Functional Heap-Allocated Frames



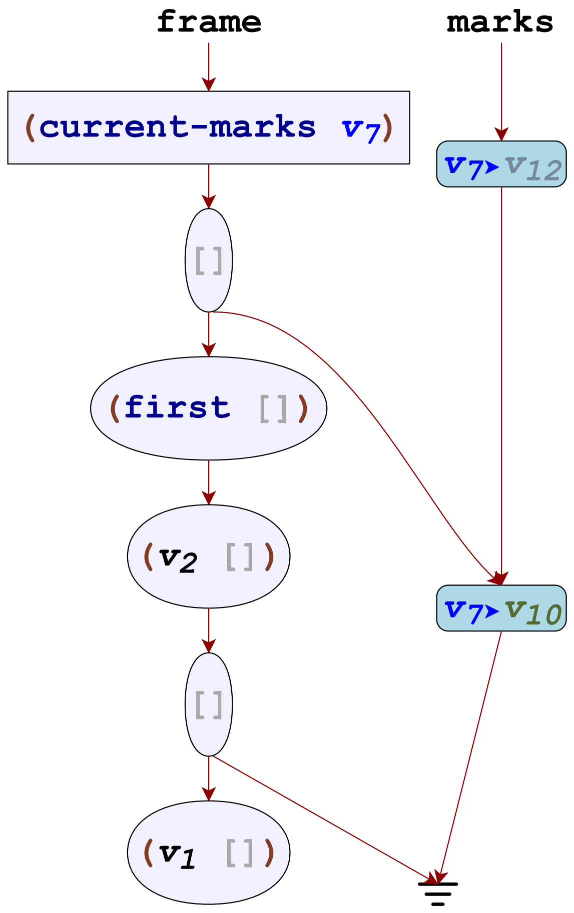
Functional Heap-Allocated Frames



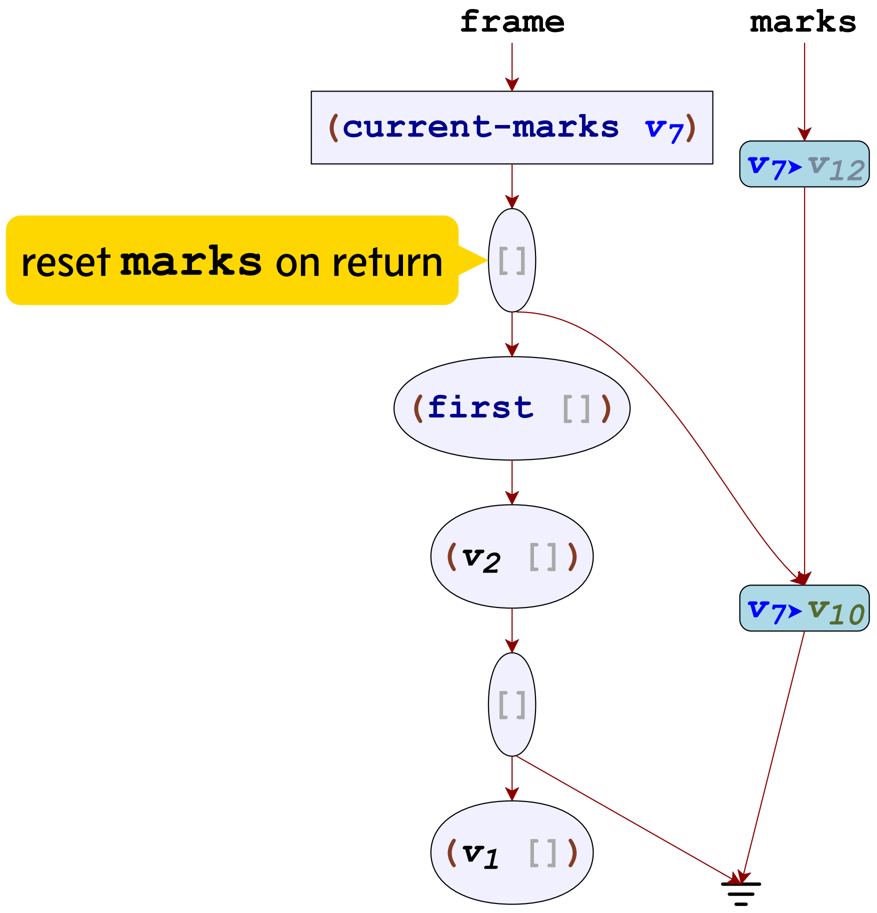
Functional Heap-Allocated Frames



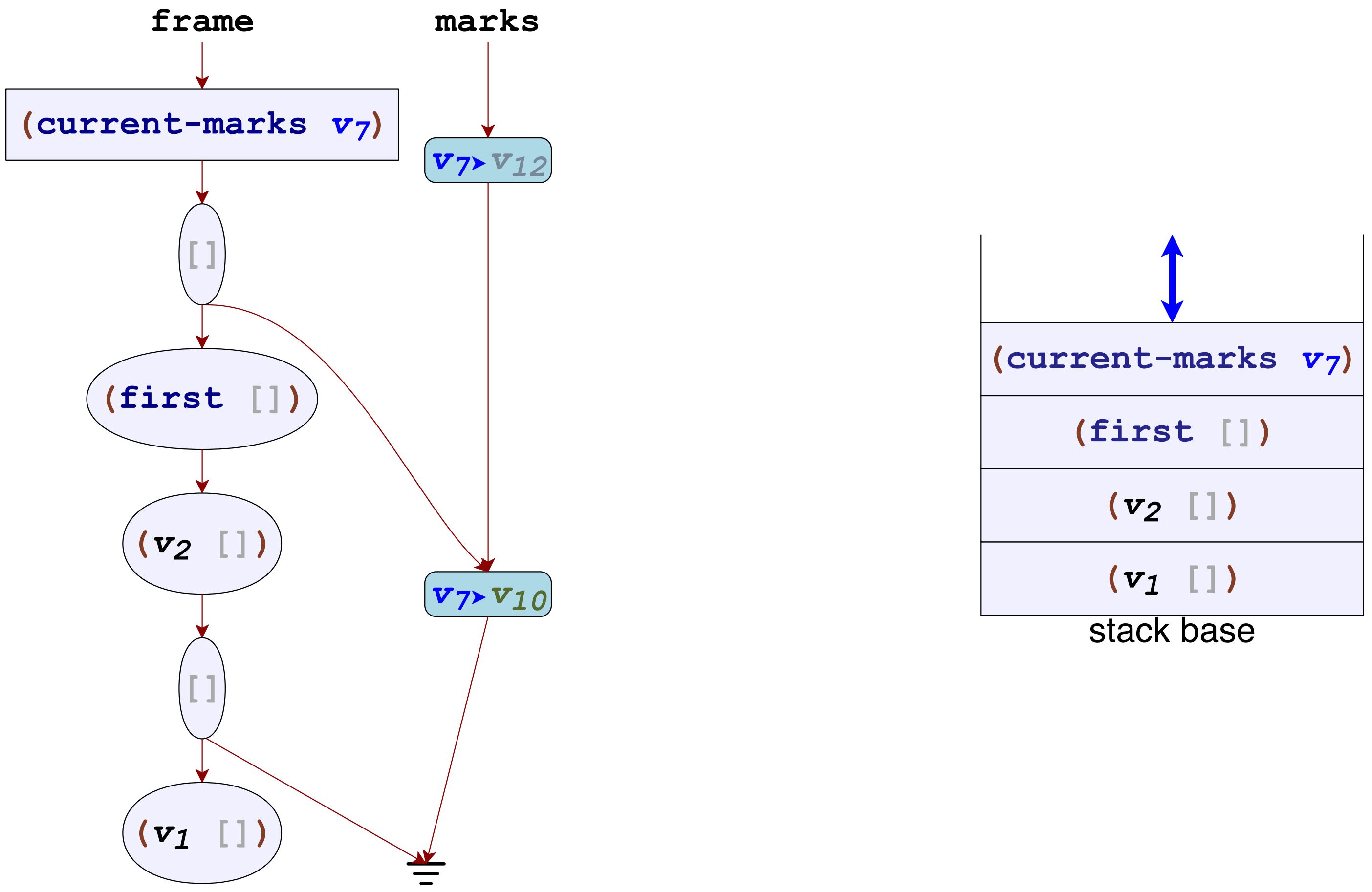
Functional Heap-Allocated Frames



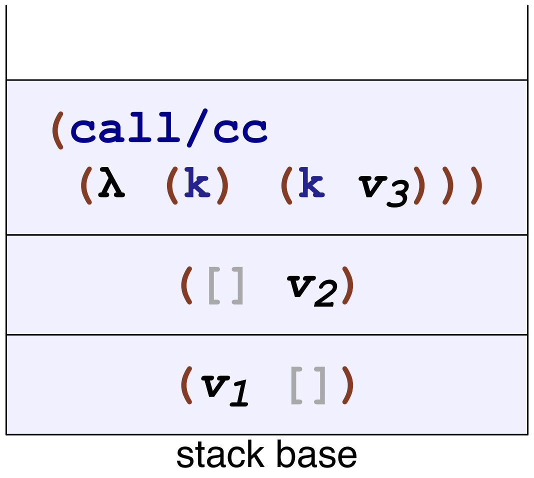
Functional Heap-Allocated Frames



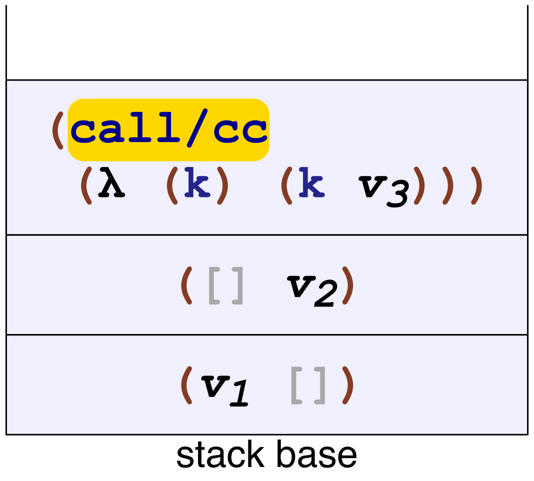
Heap versus Stack



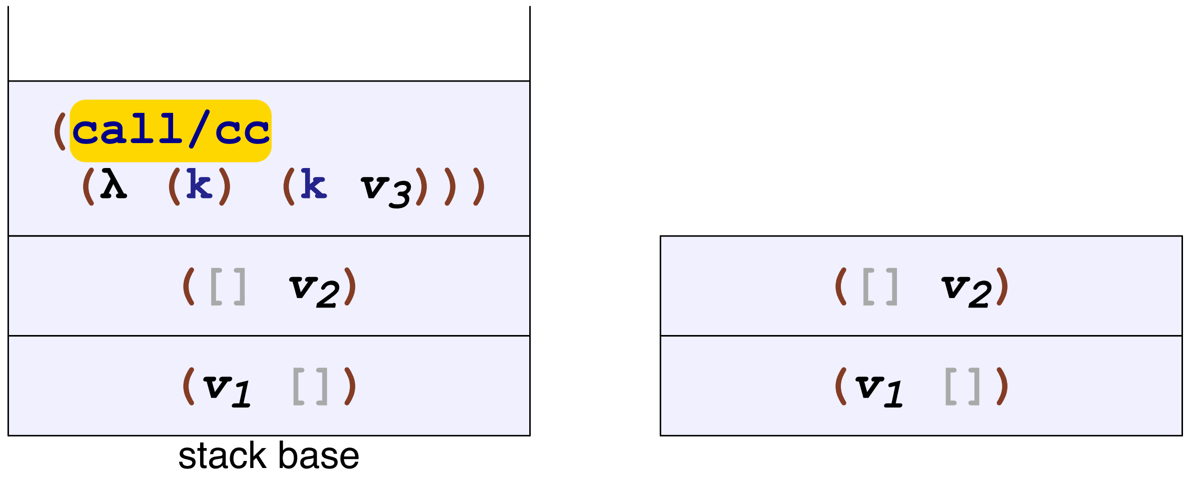
Stack-Allocated Frames



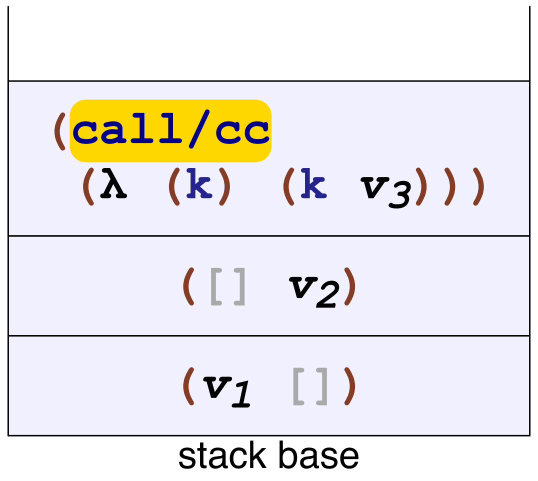
Capturing Stack-Based Continuations



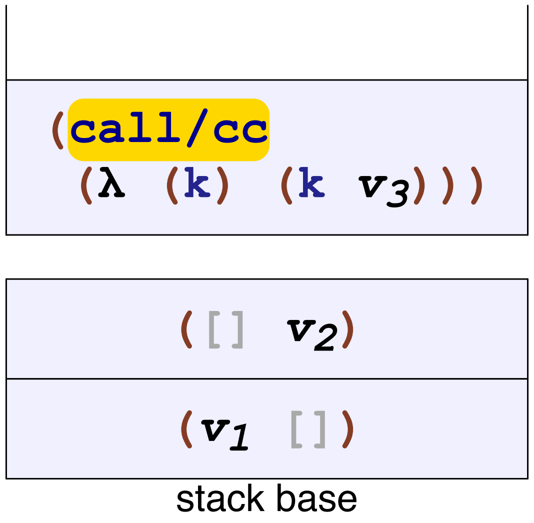
Capturing Stack-Based Continuations



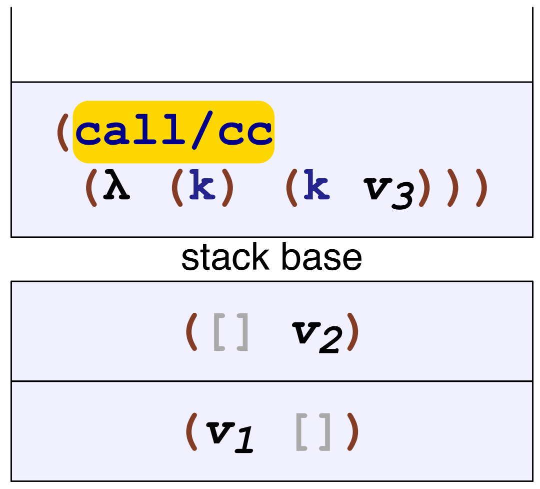
Capturing Stack-Based Continuations



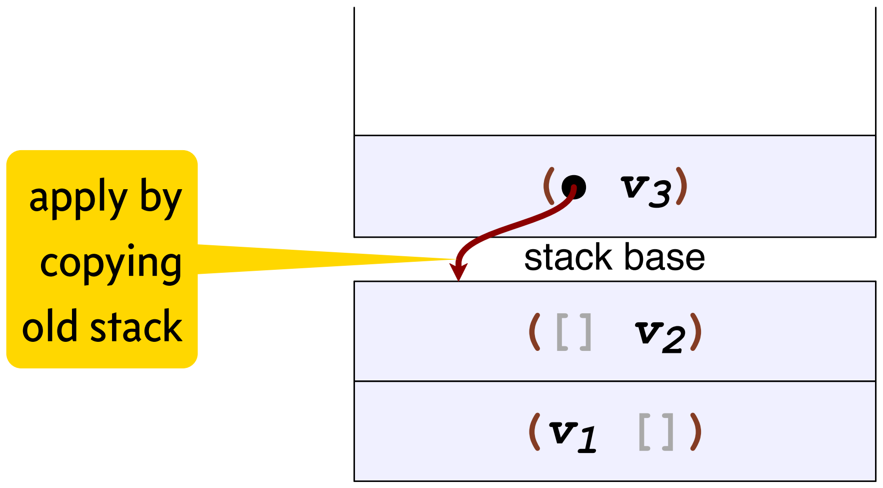
Capturing Stack-Based Continuations



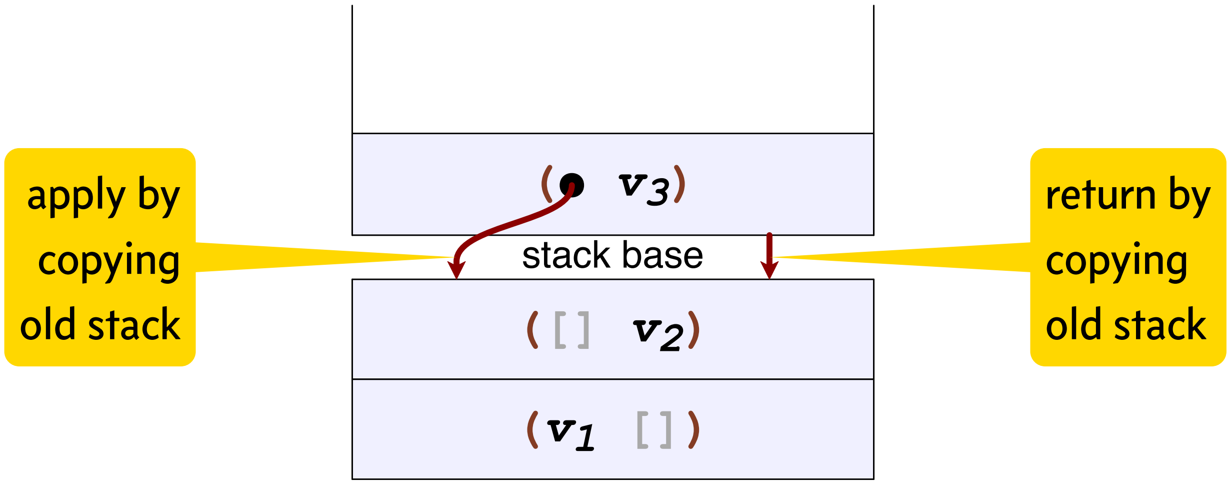
Capturing Stack-Based Continuations



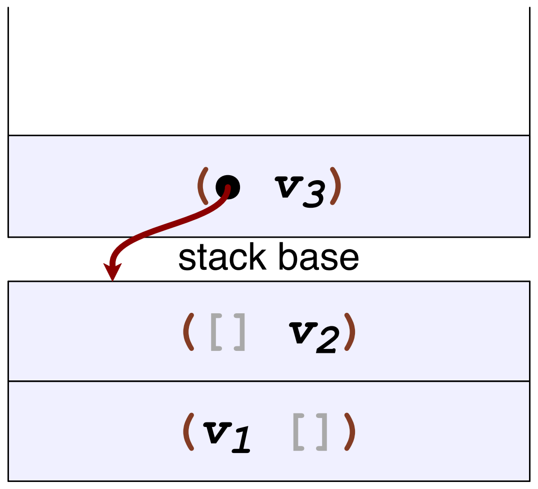
Capturing Stack-Based Continuations



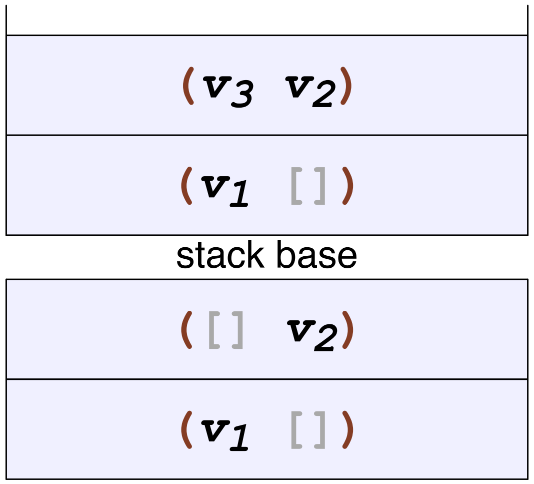
Capturing Stack-Based Continuations



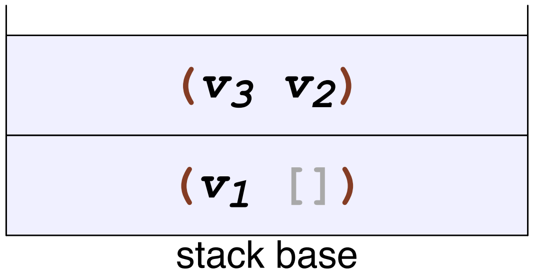
Applying Stack-Based Continuations



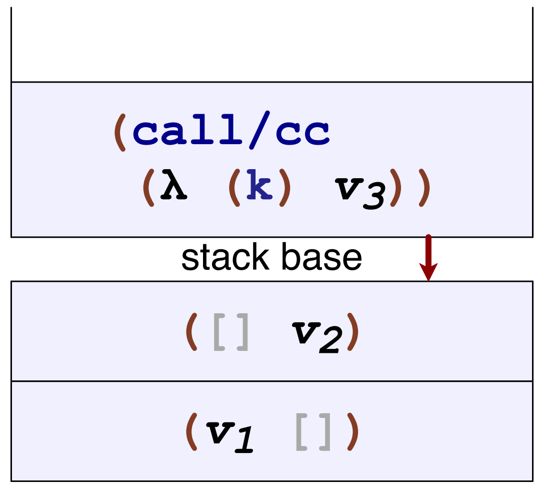
Applying Stack-Based Continuations



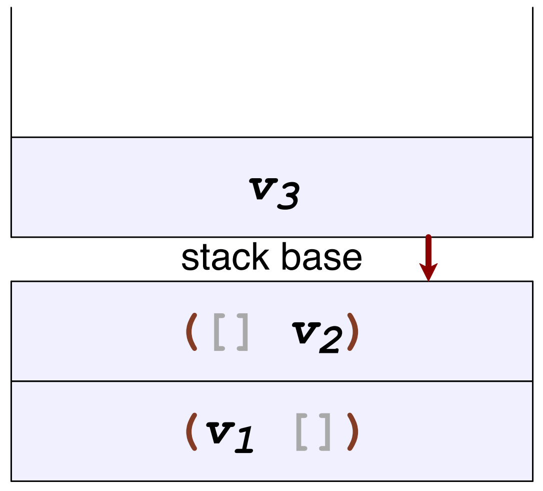
Applying Stack-Based Continuations



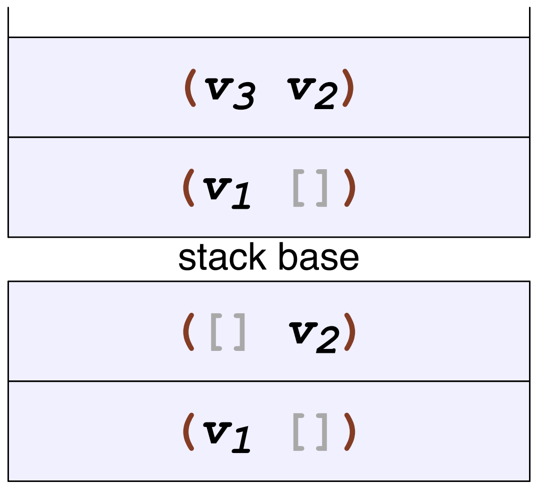
Returning After Capture



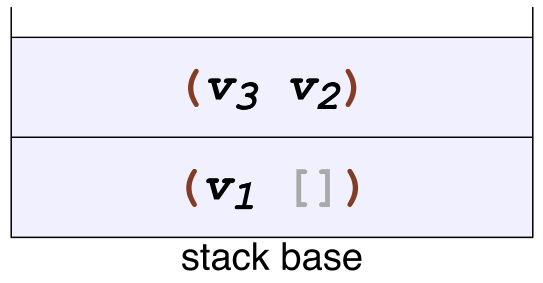
Returning After Capture



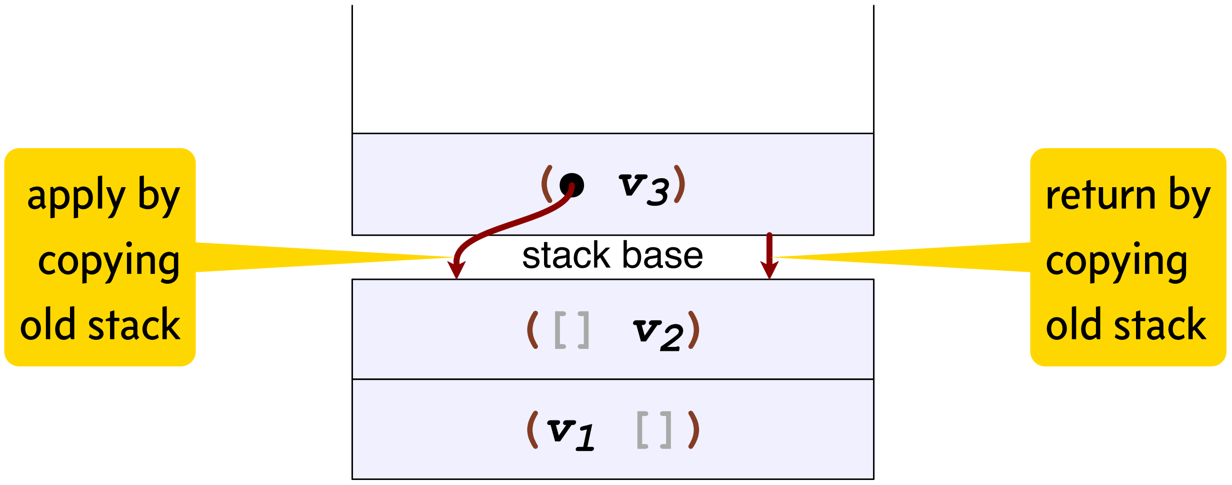
Returning After Capture



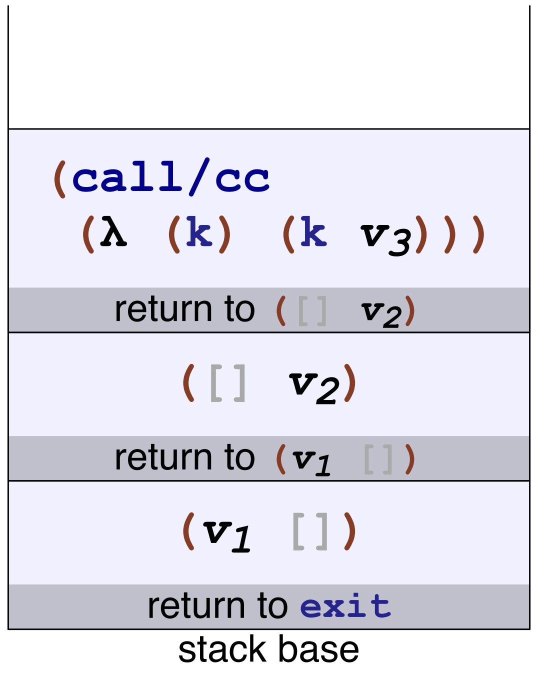
Returning After Capture



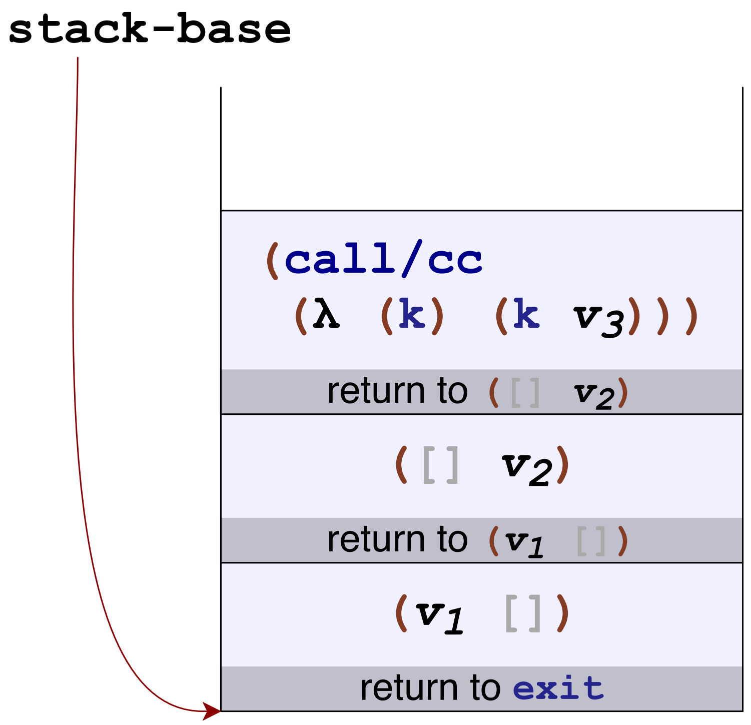
Two Kinds of Continuation Pointers



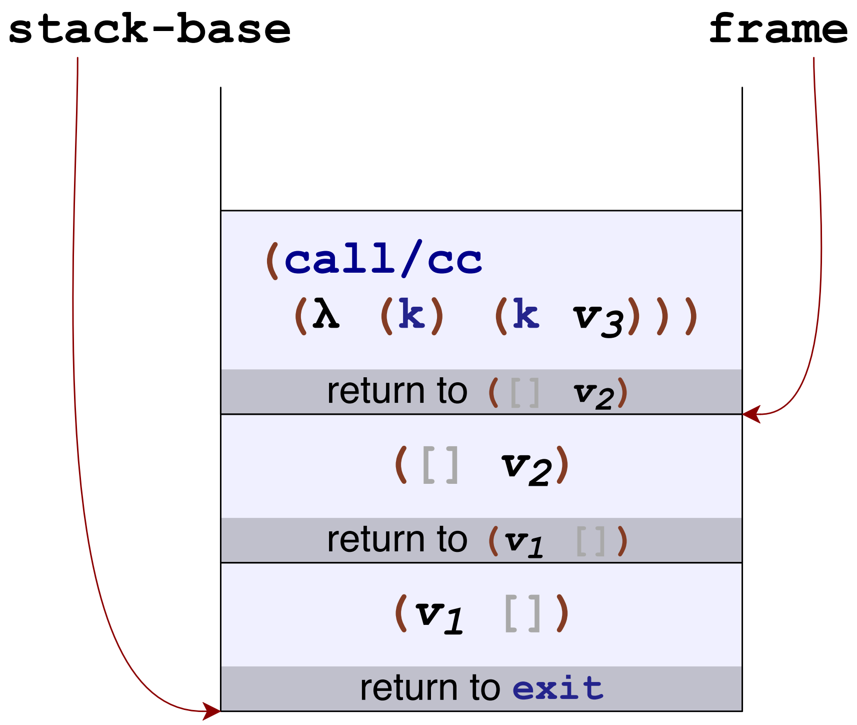
Returns in Stack Frames



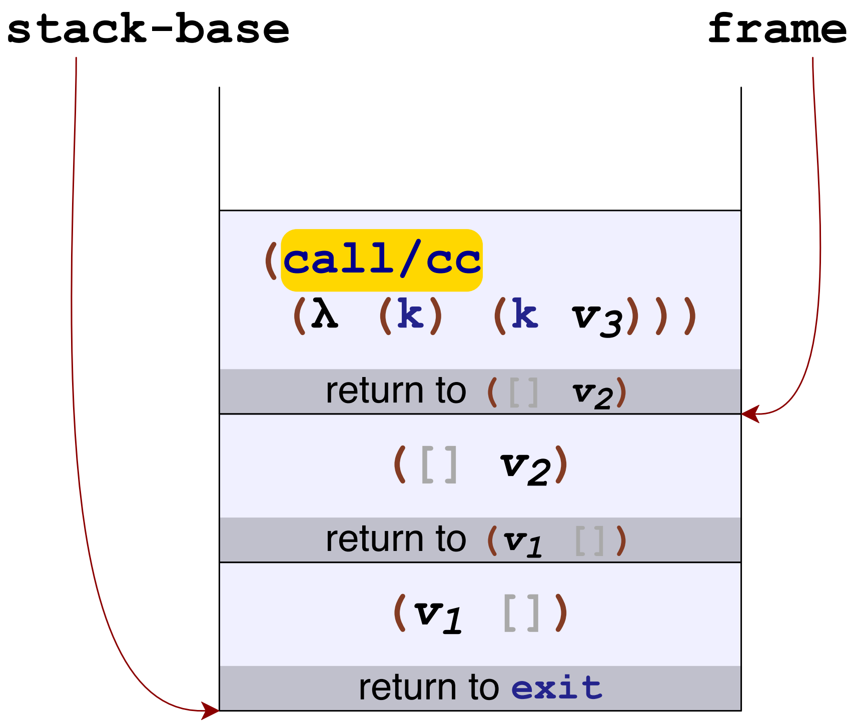
Returns in Stack Frames



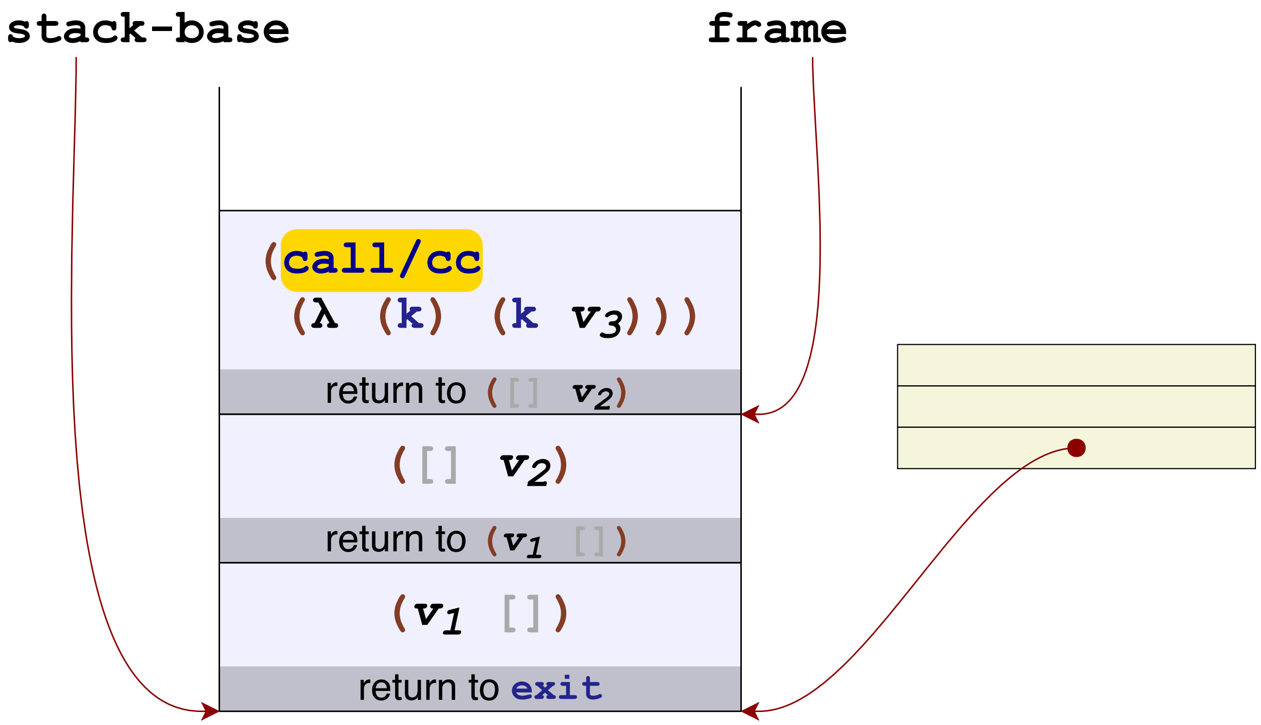
Returns in Stack Frames



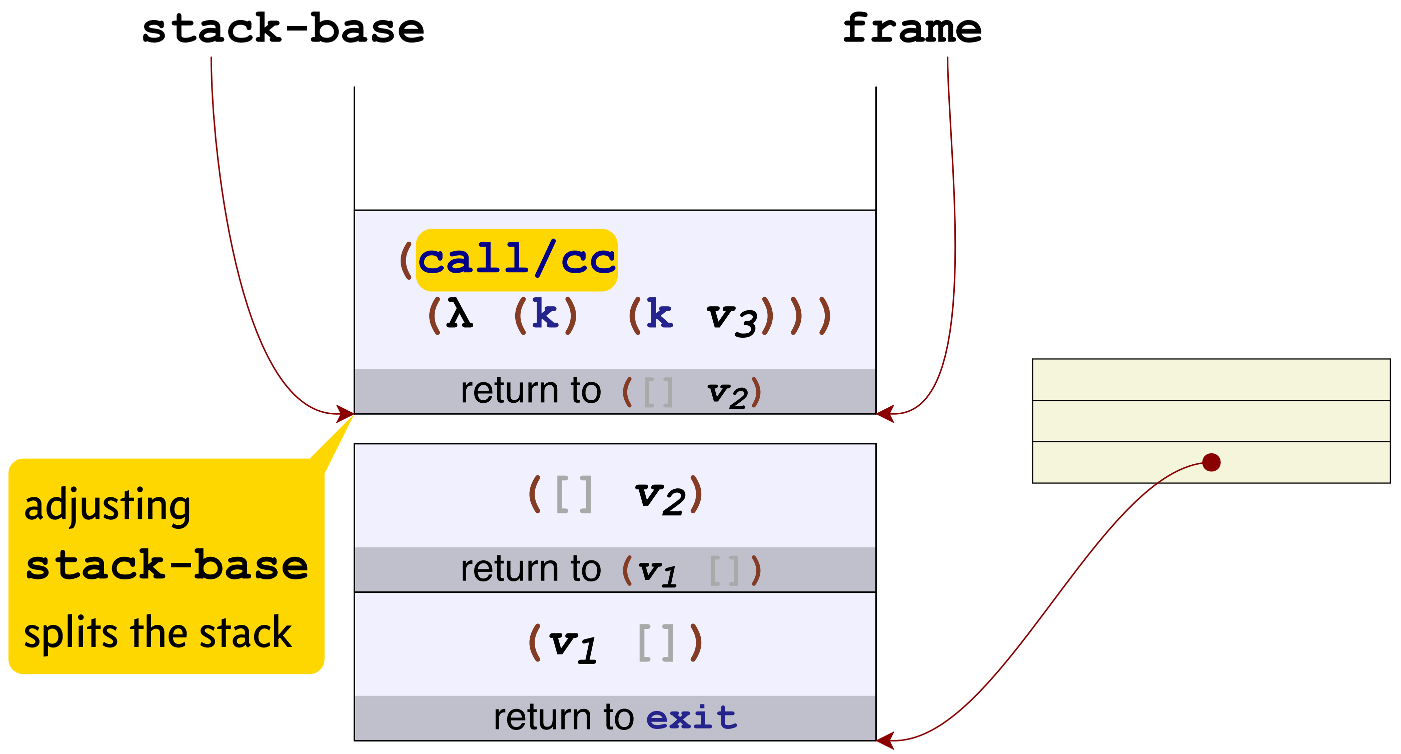
Capturing Stack-Based Continuations



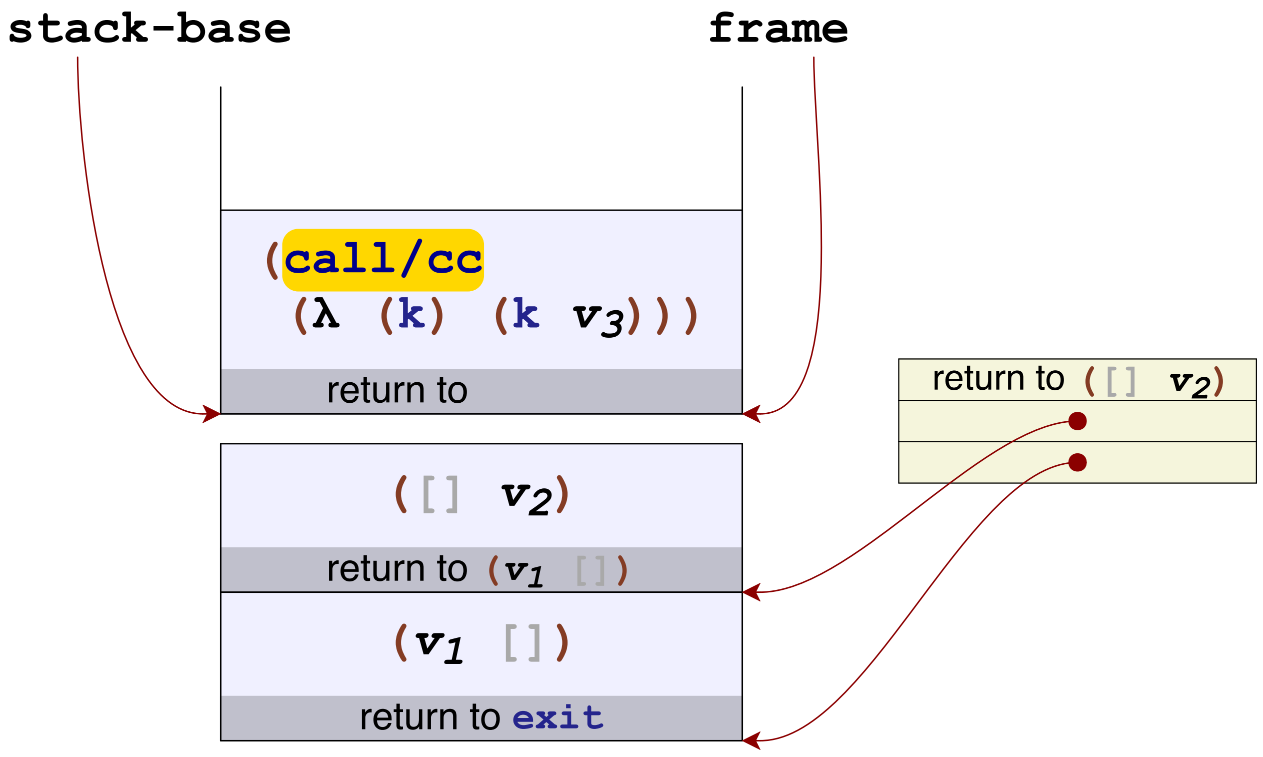
Capturing Stack-Based Continuations



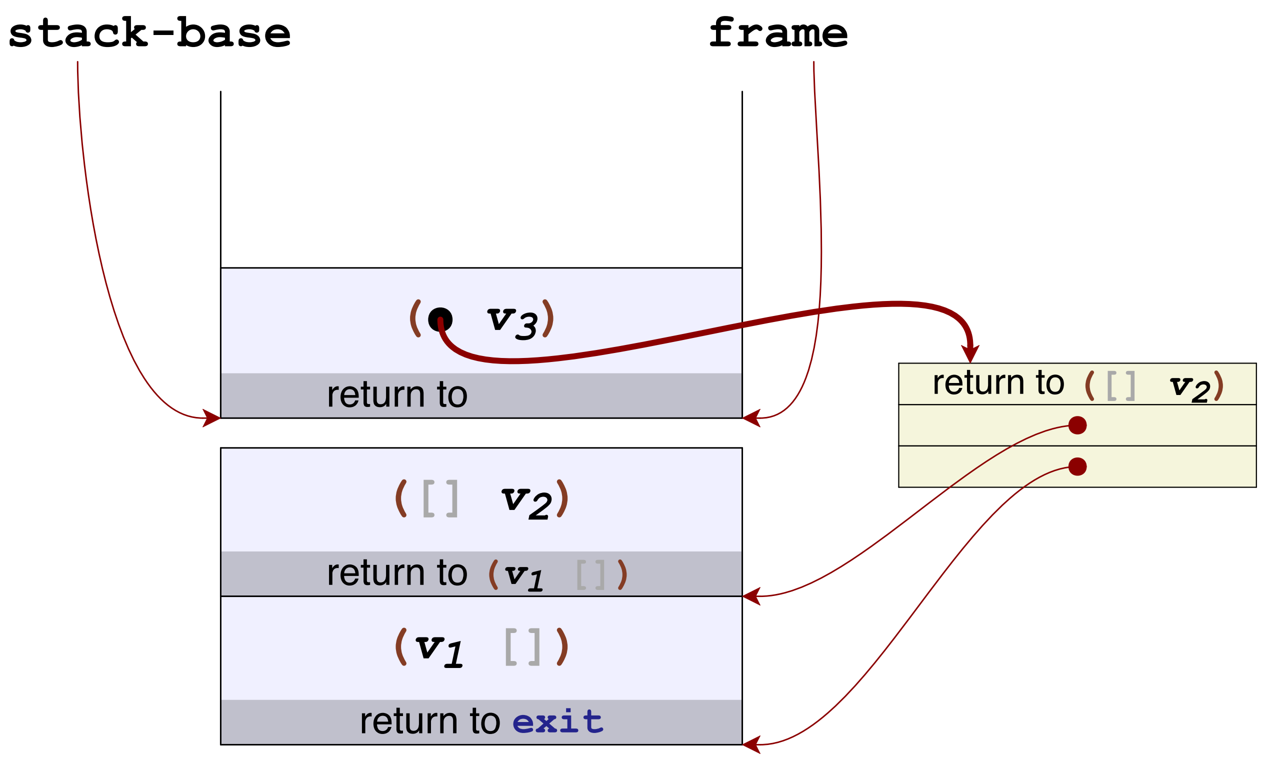
Capturing Stack-Based Continuations



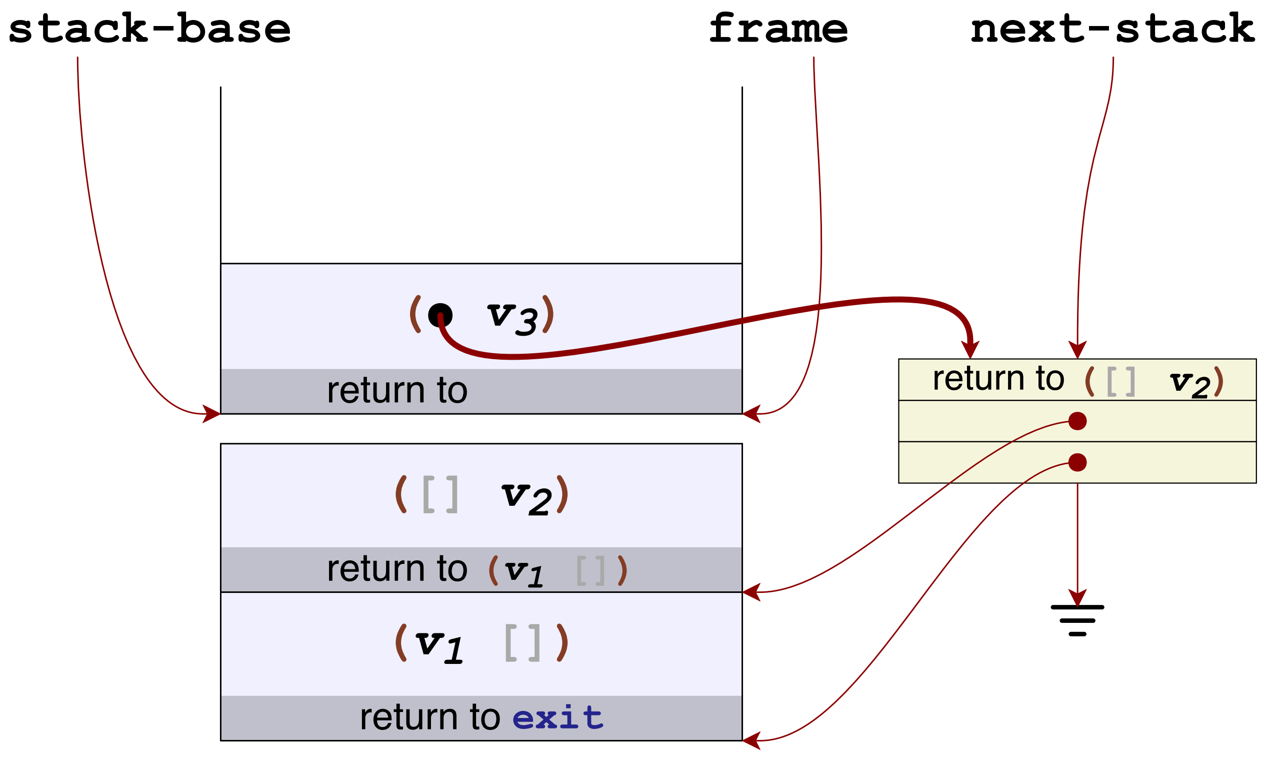
Capturing Stack-Based Continuations



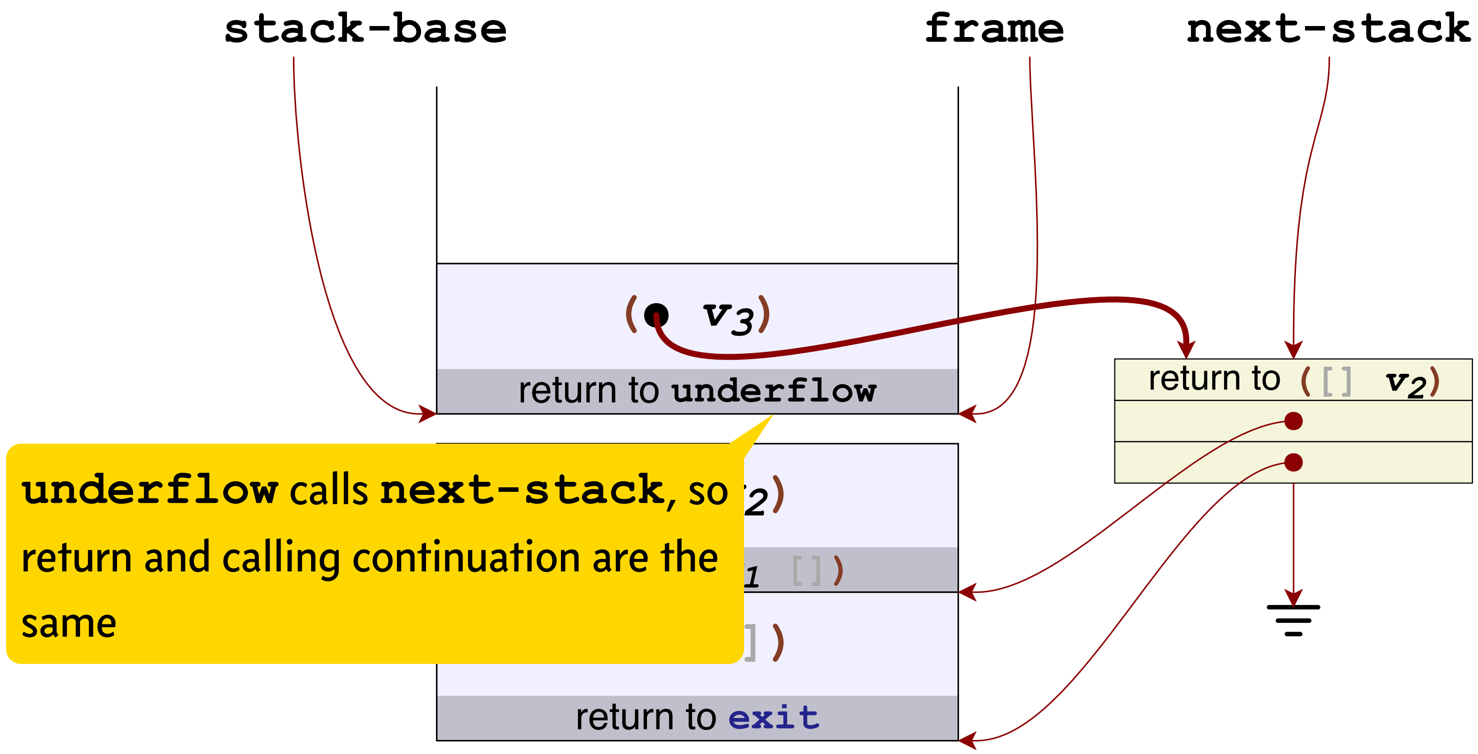
Capturing Stack-Based Continuations



Capturing Stack-Based Continuations

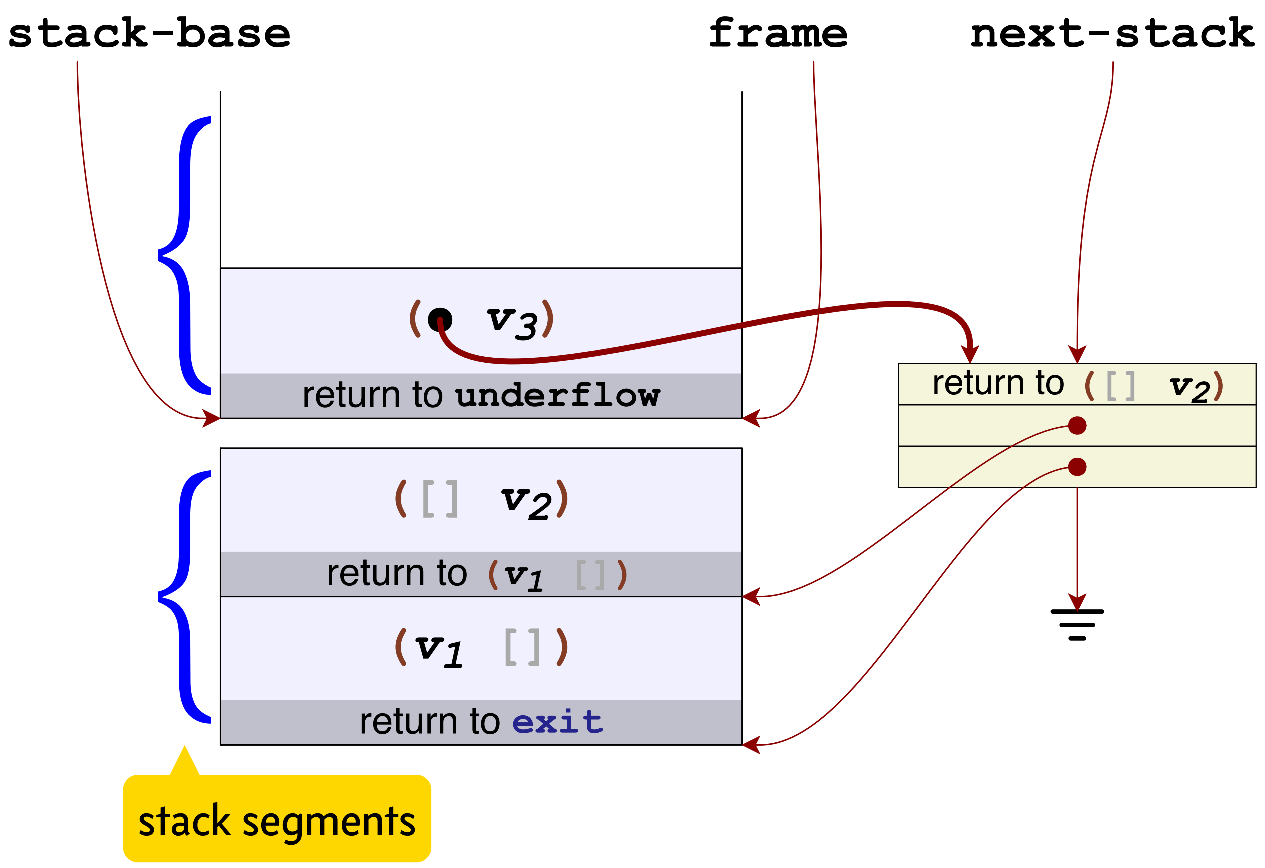


Capturing Stack-Based Continuations

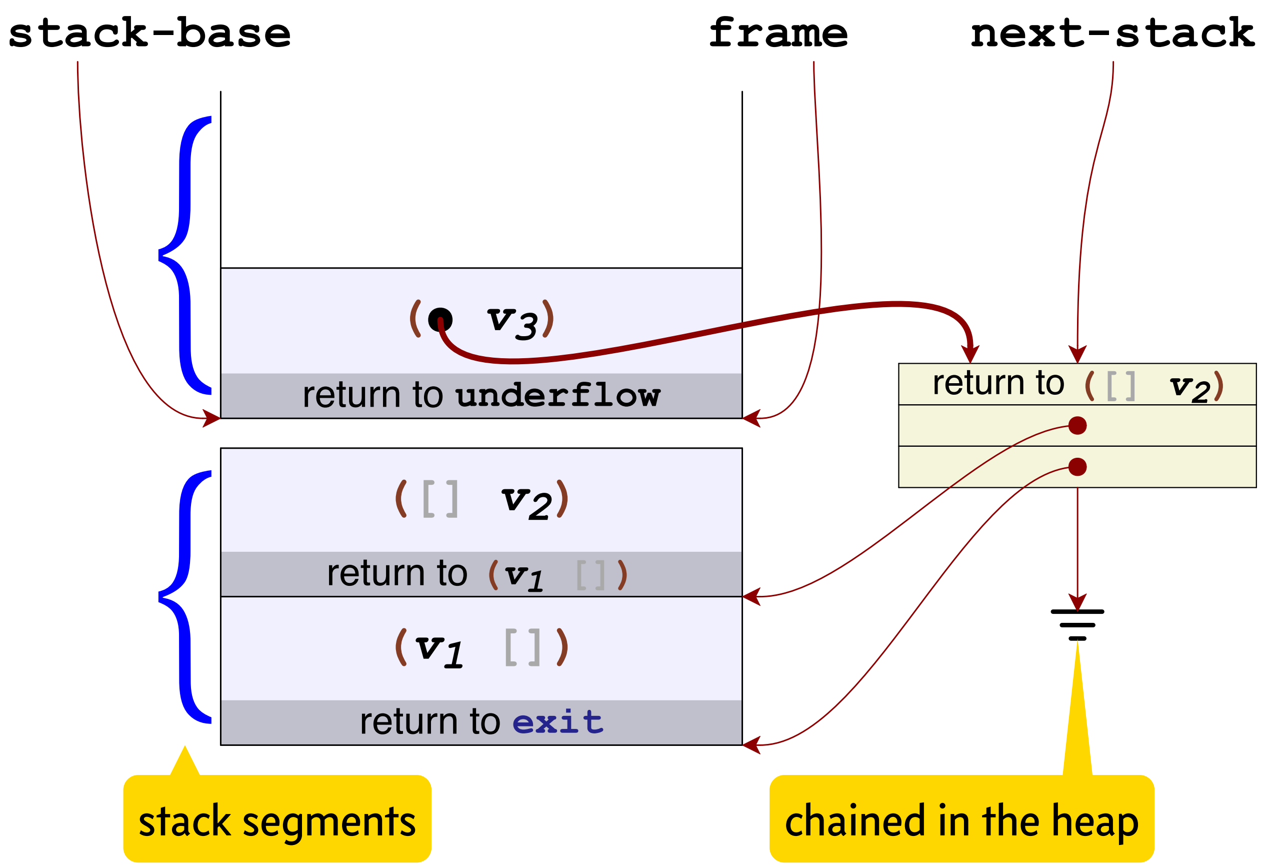


underflow calls **next-stack**, so return and calling continuation are the same

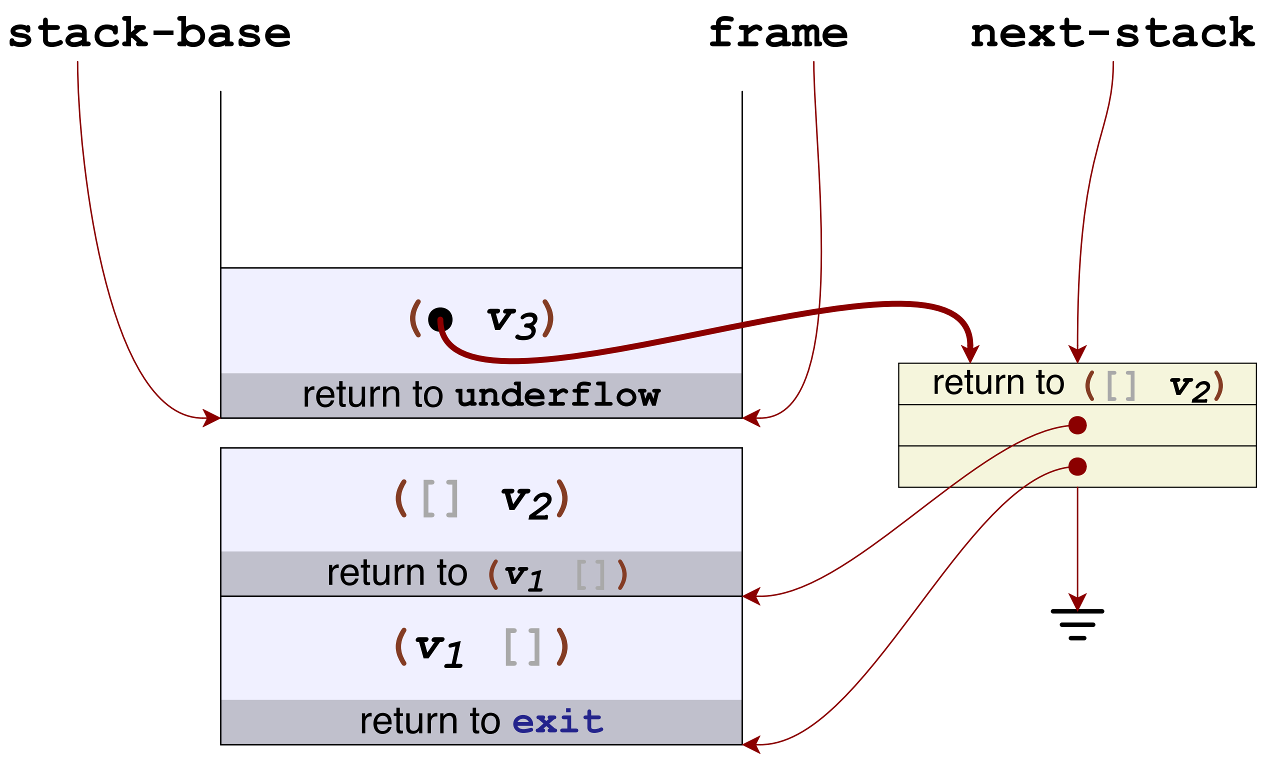
Hybrid Stack–Heap Continuation



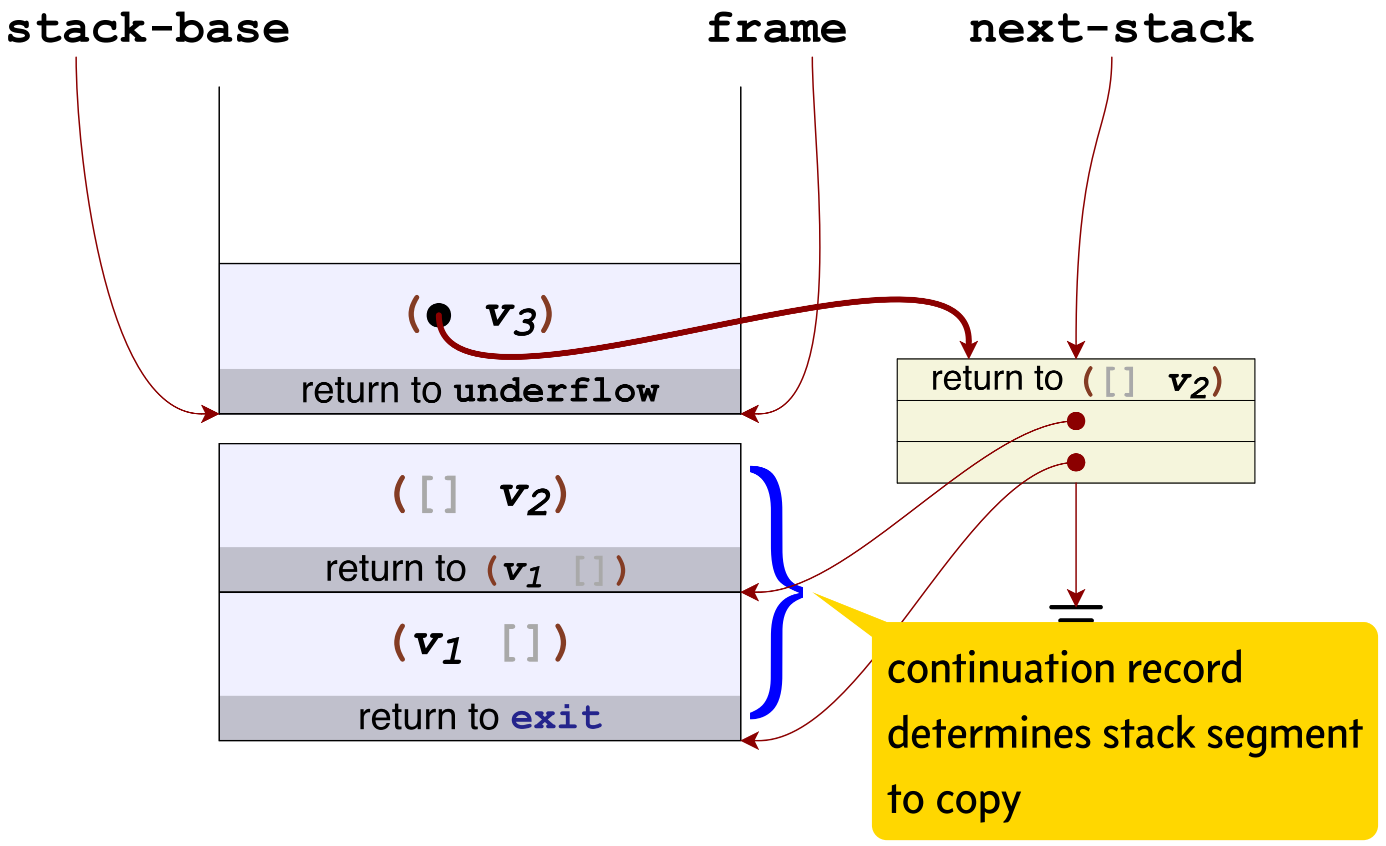
Hybrid Stack–Heap Continuation



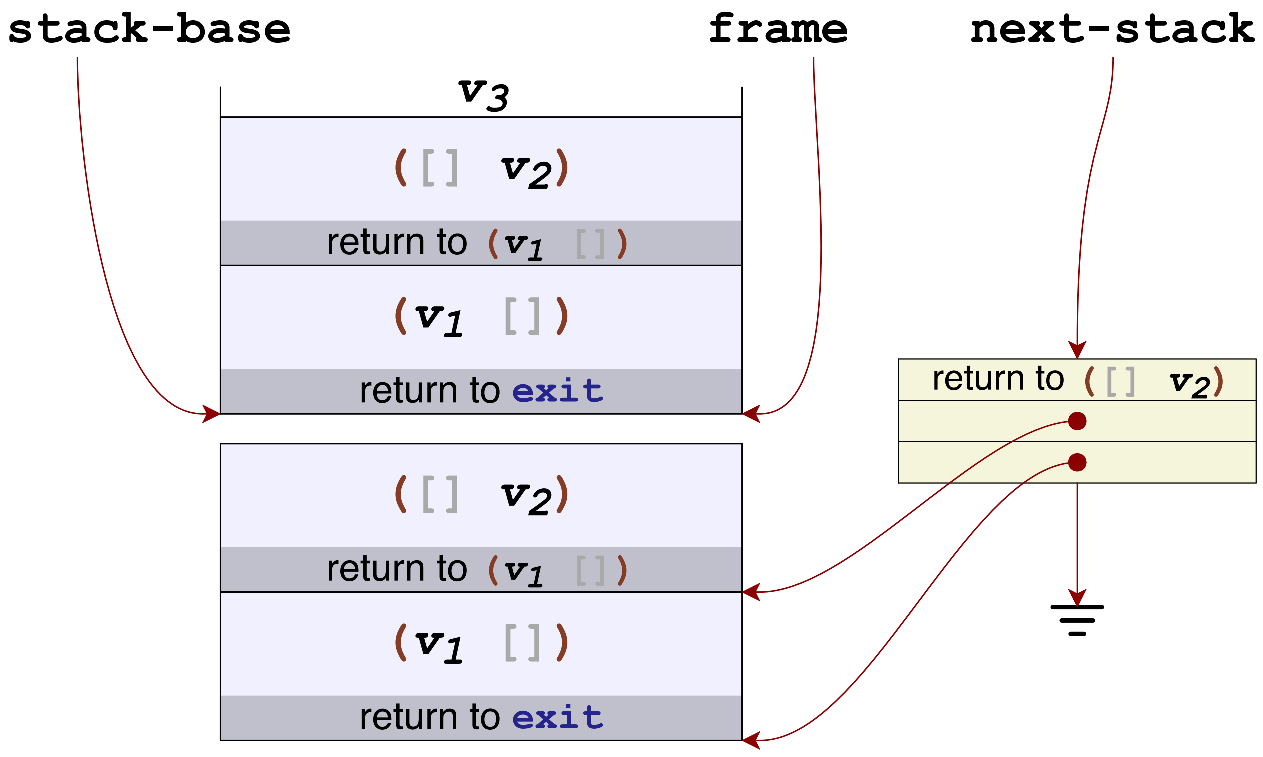
Applying Stack-Based Continuations



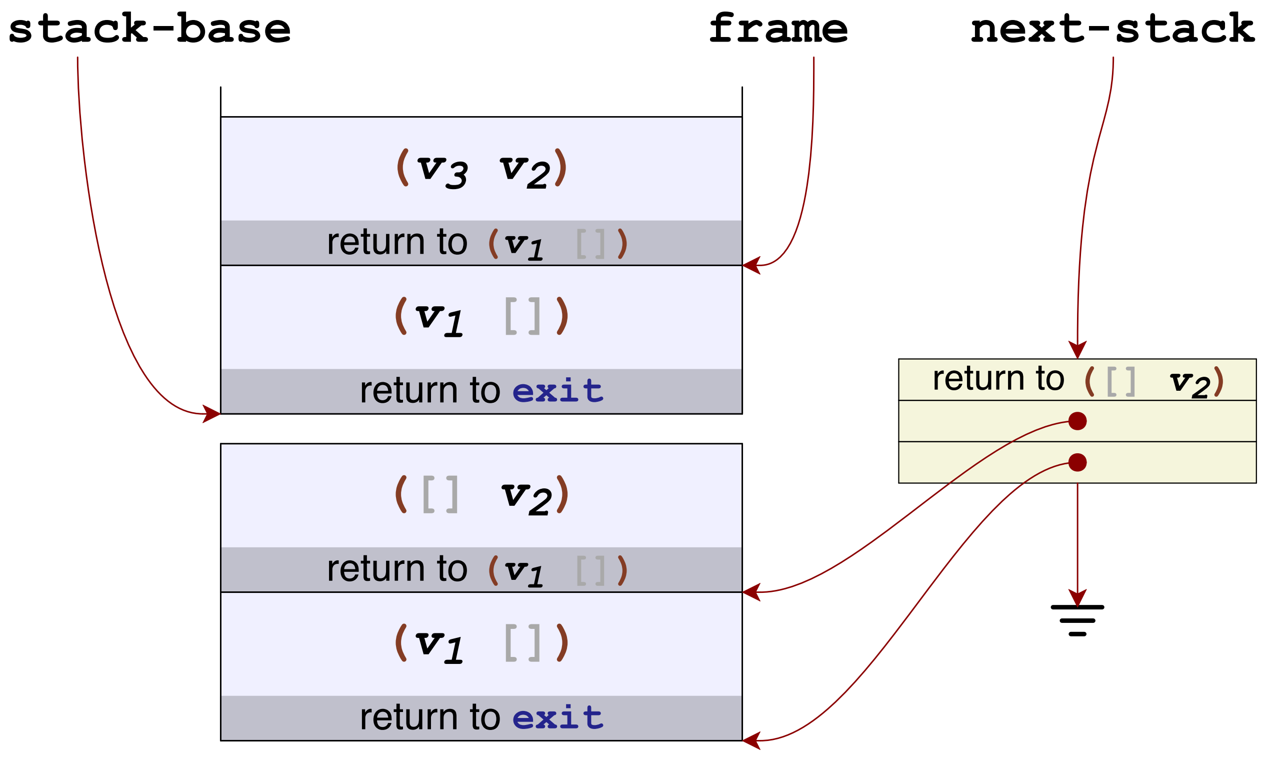
Applying Stack-Based Continuations



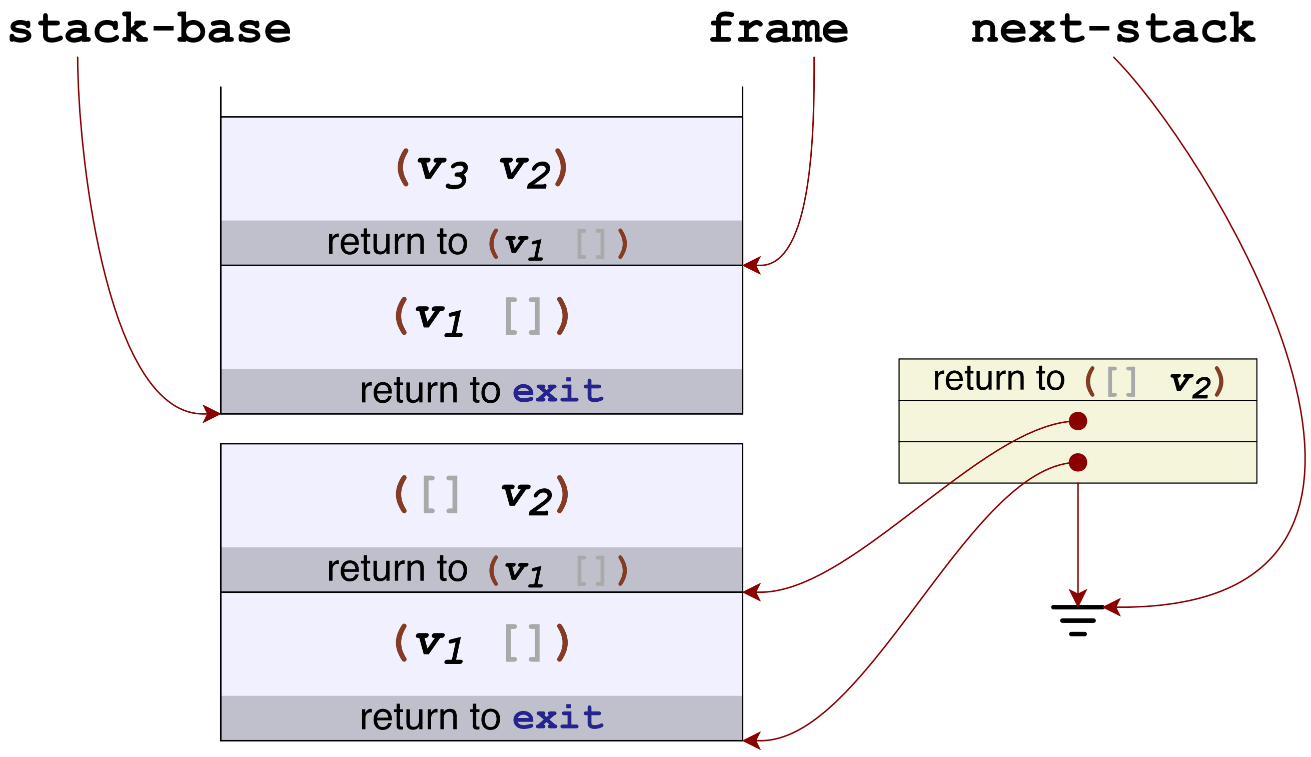
Applying Stack-Based Continuations



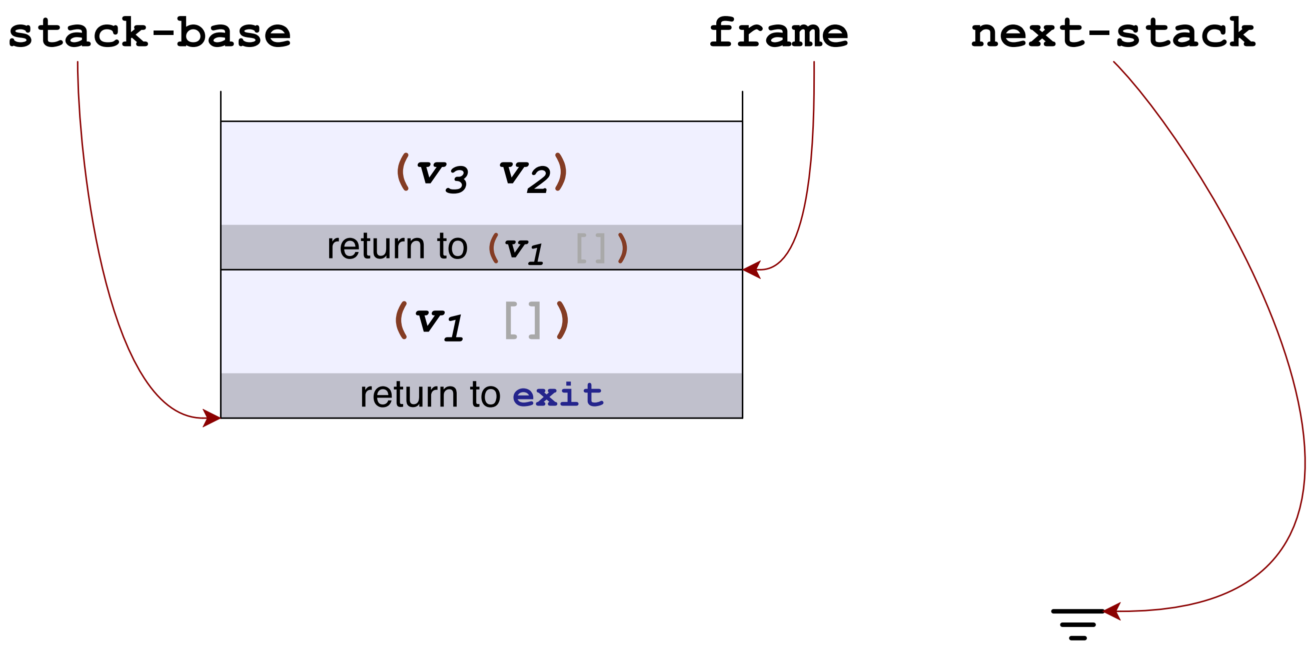
Applying Stack-Based Continuations



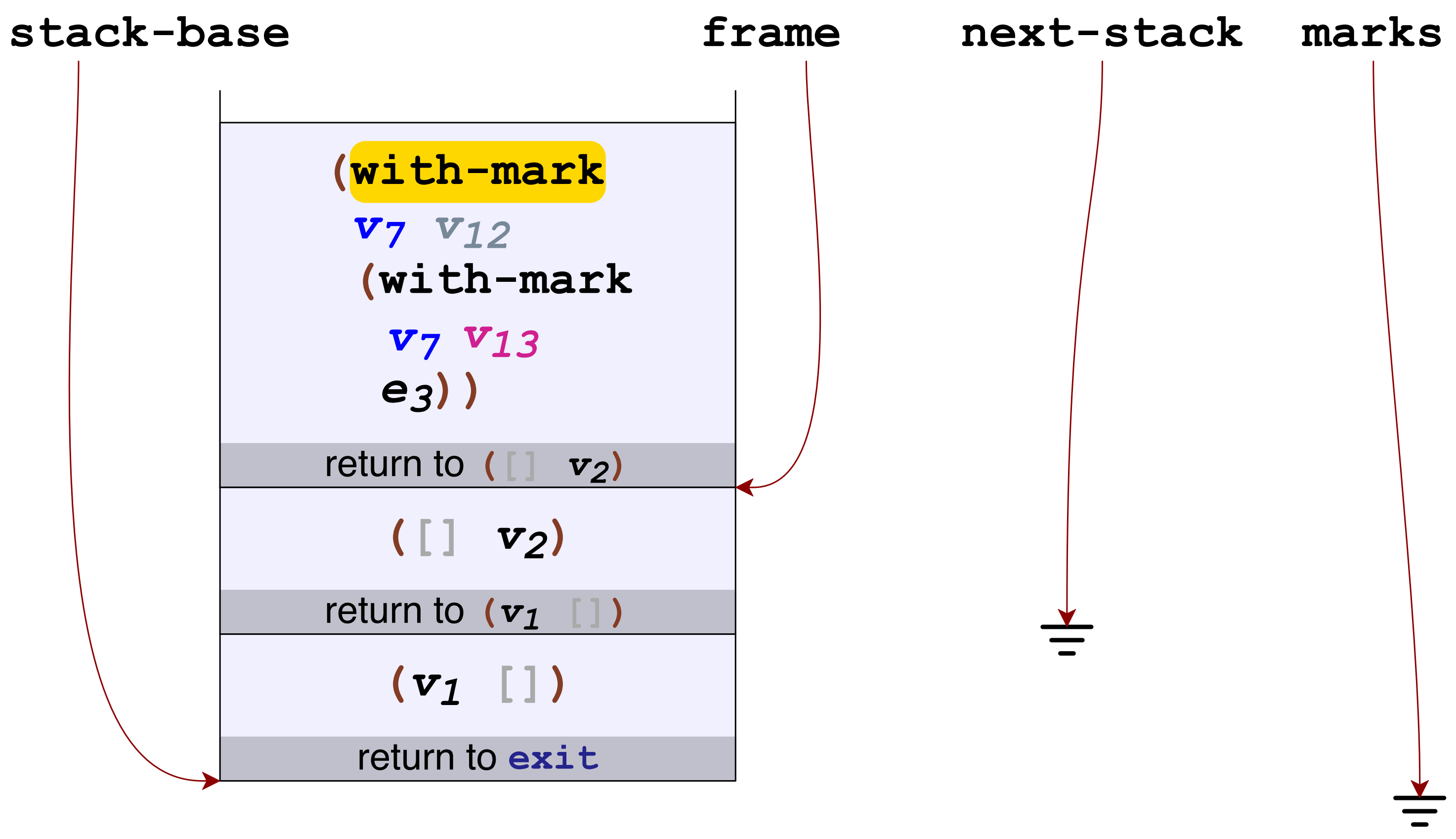
Applying Stack-Based Continuations



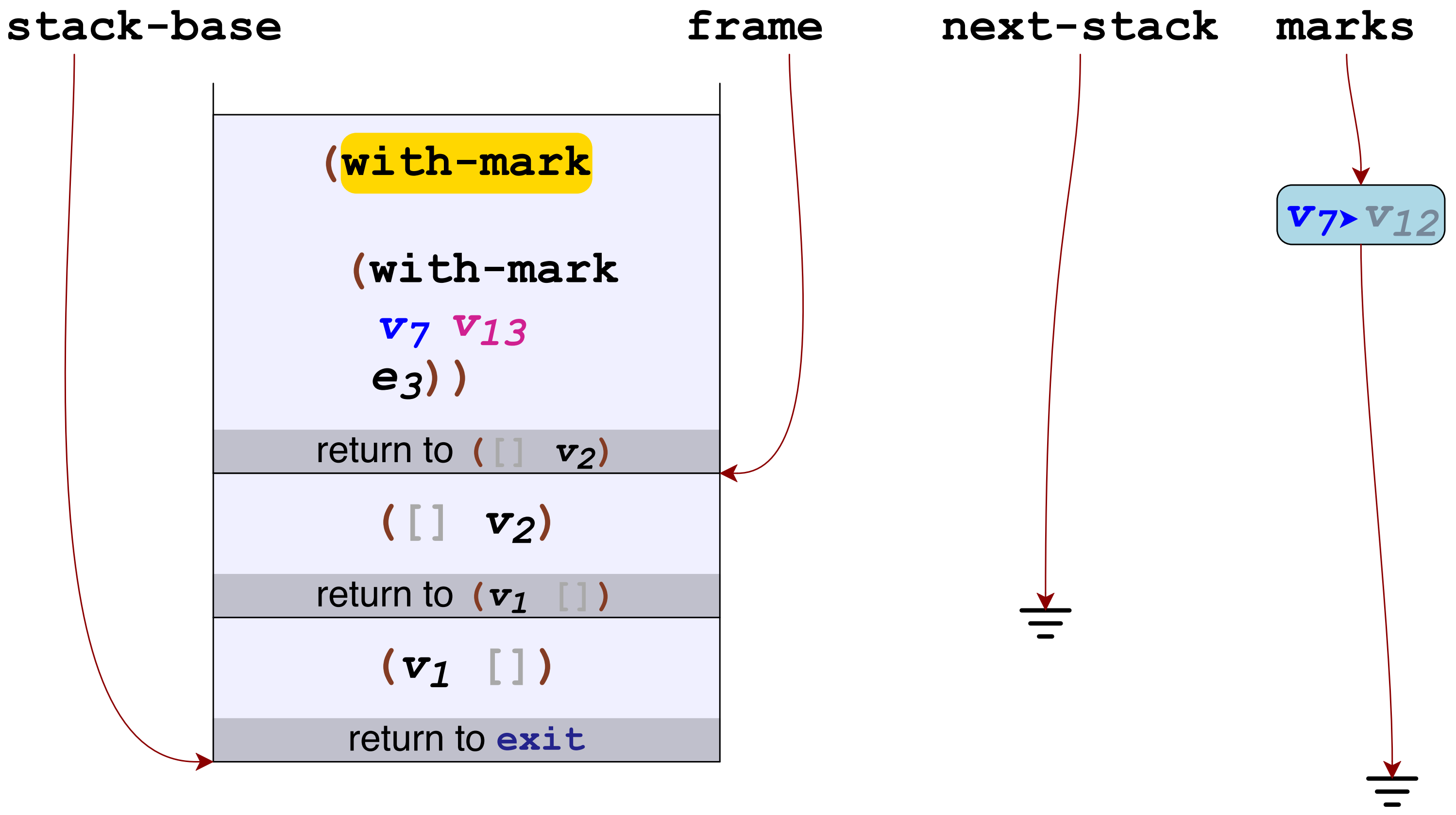
Applying Stack-Based Continuations



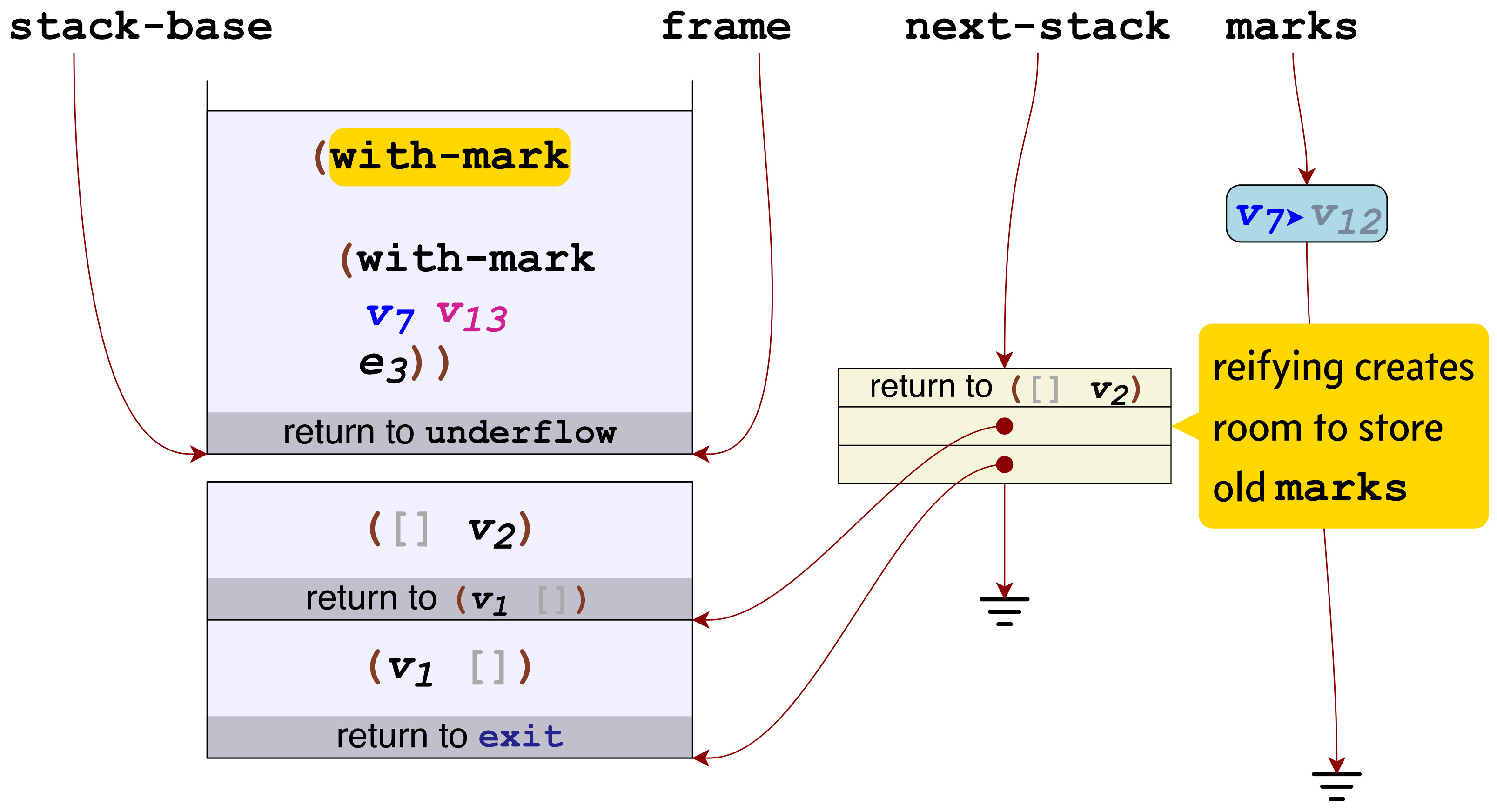
Setting Continuation Marks



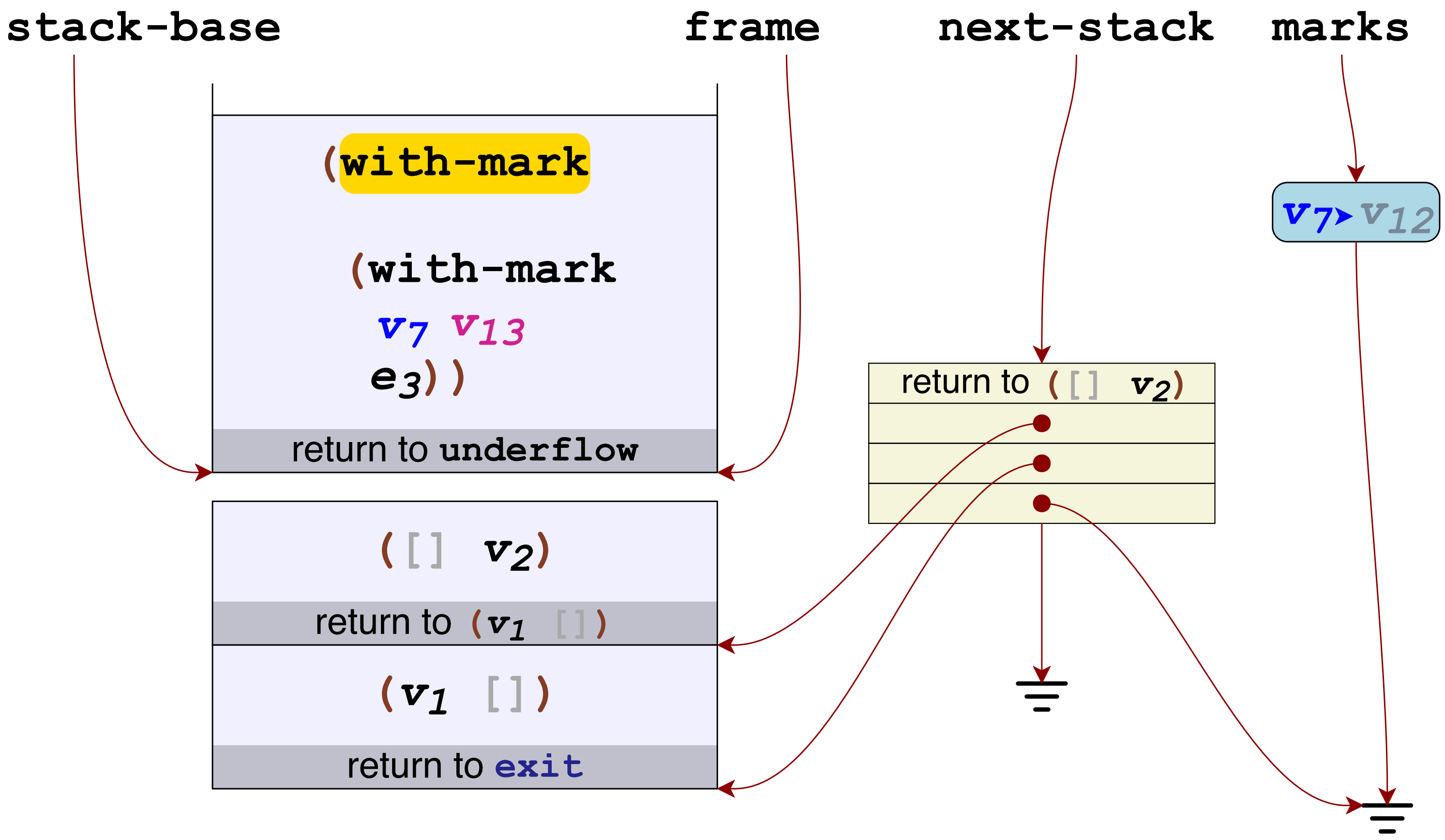
Setting Continuation Marks



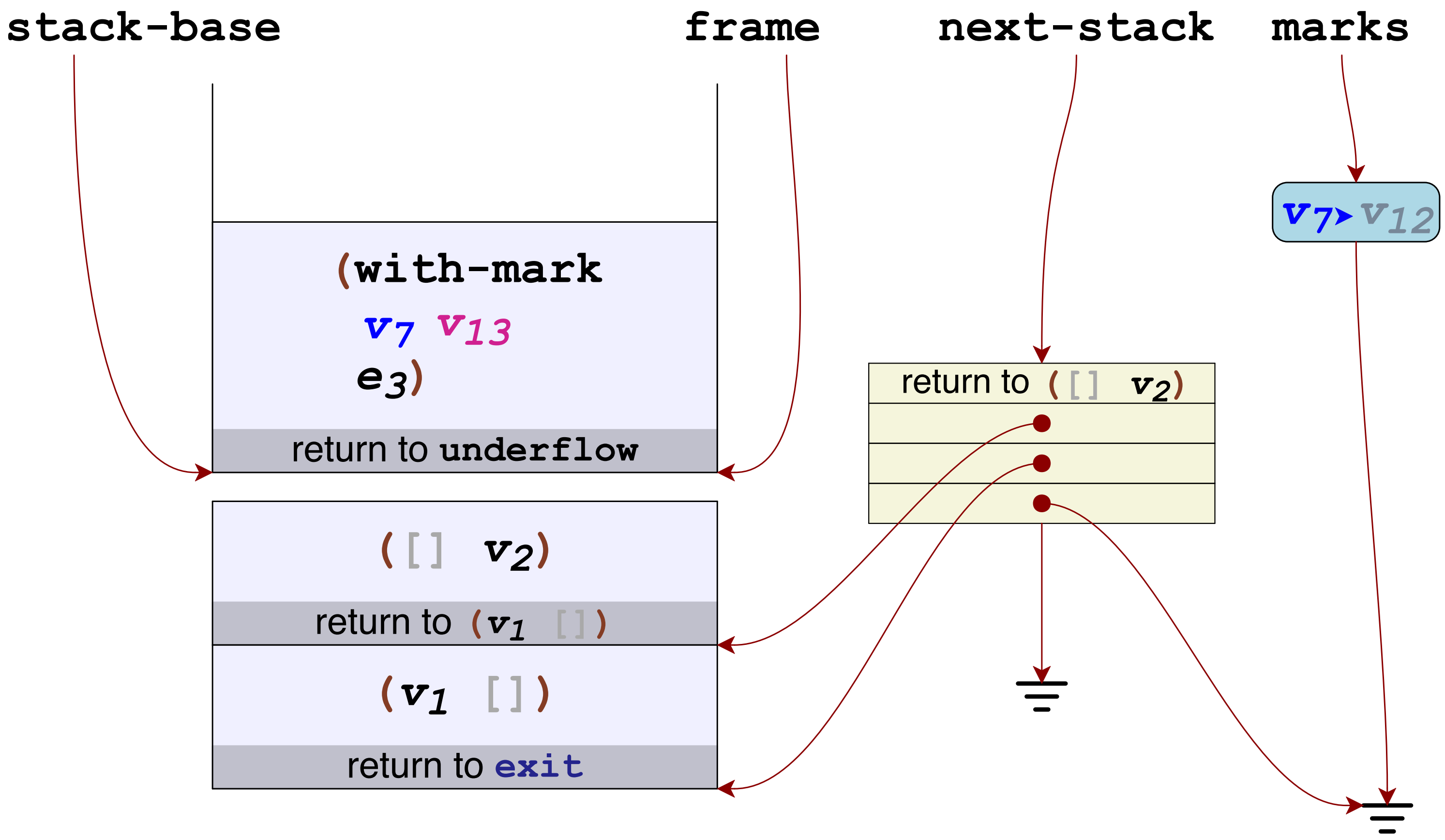
Setting Continuation Marks



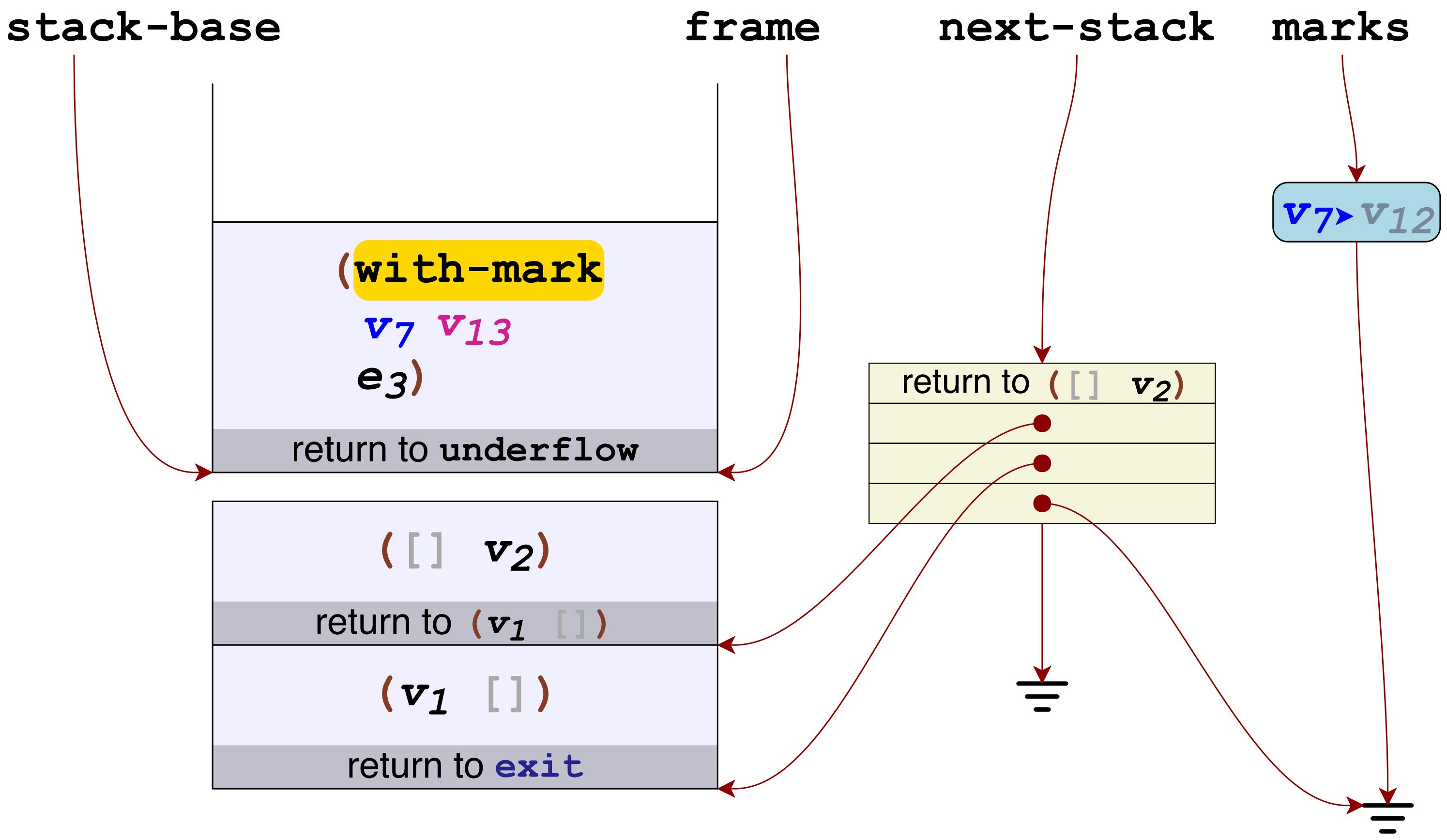
Setting Continuation Marks



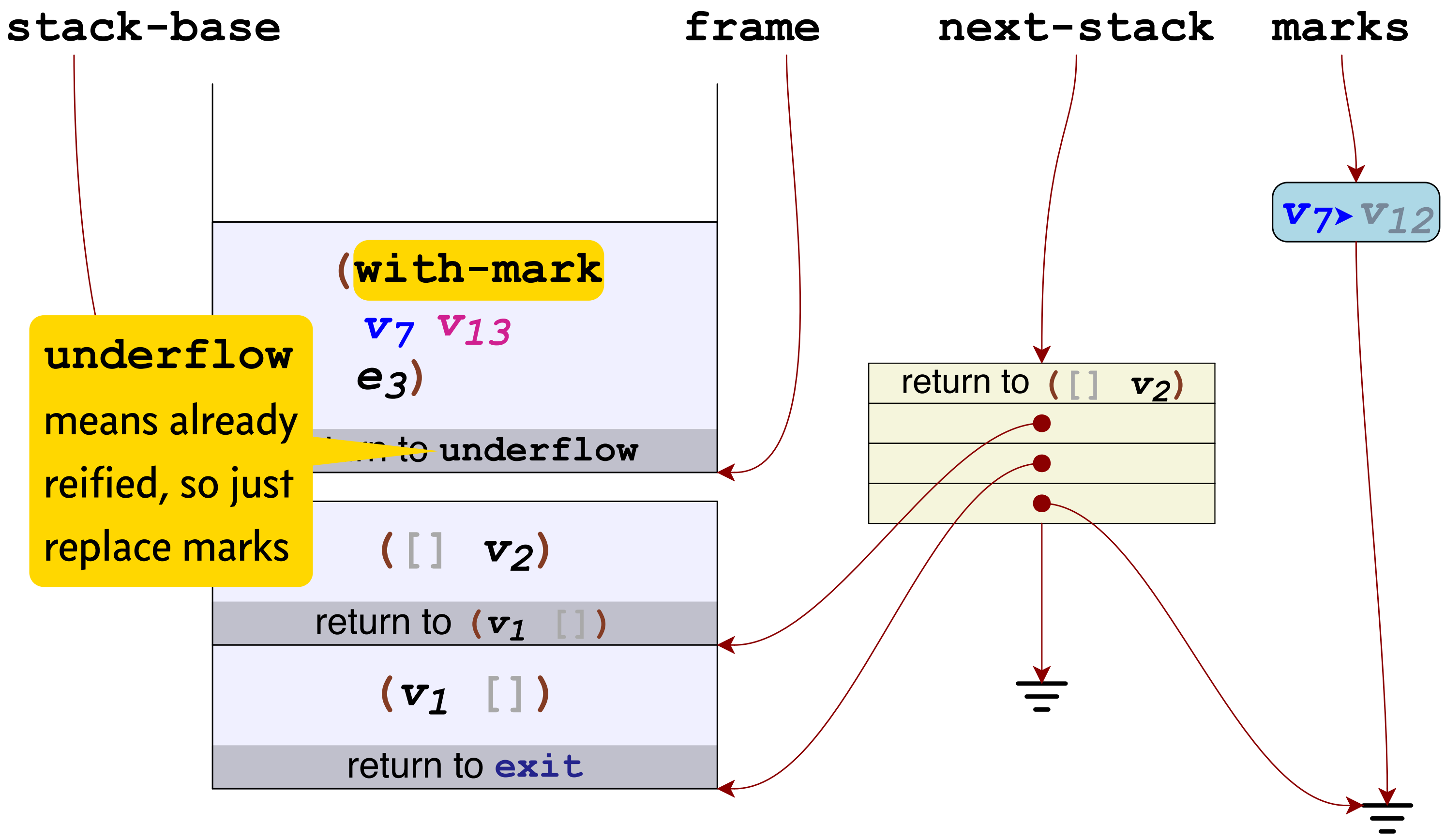
Setting Continuation Marks



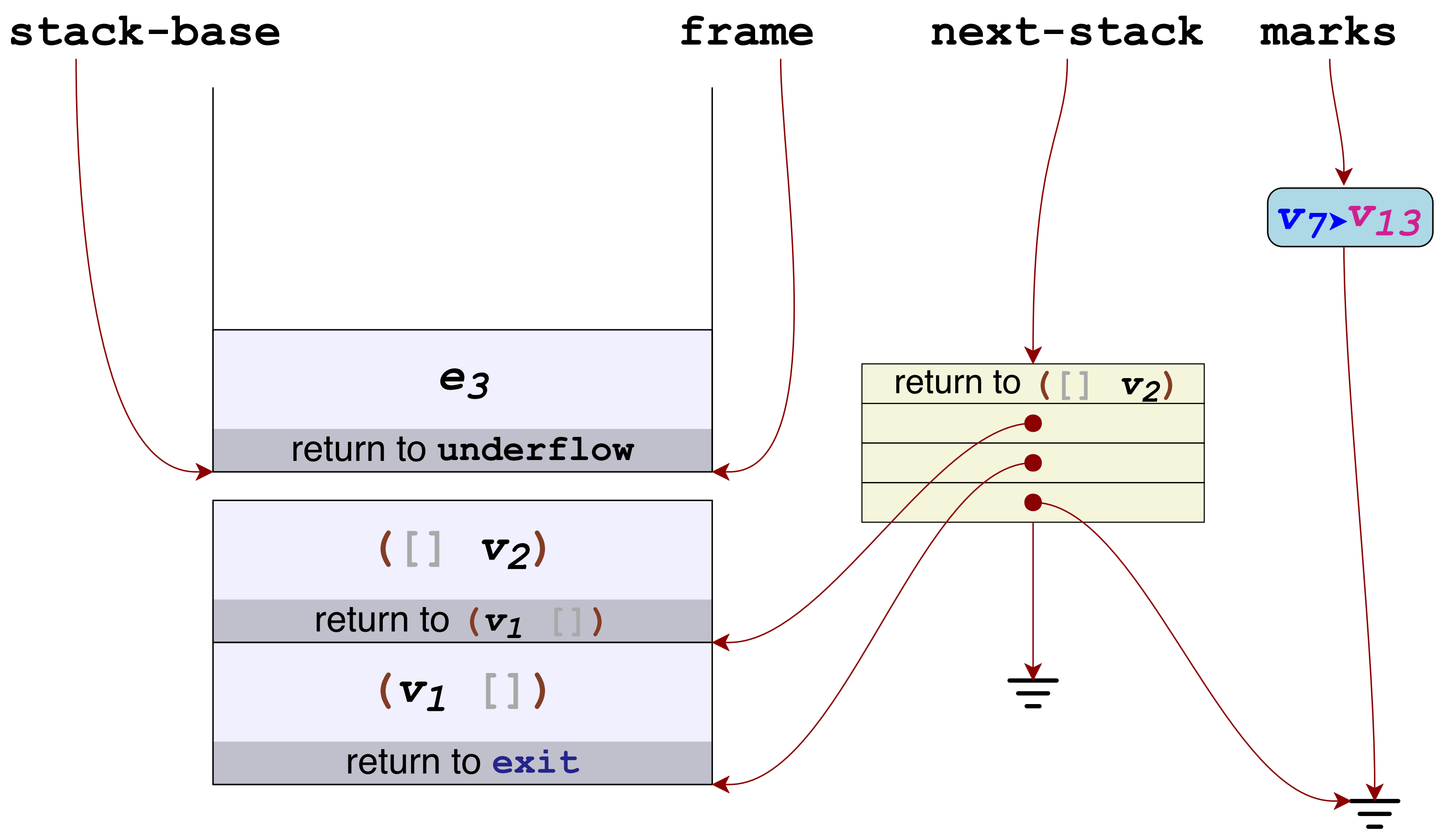
Setting Continuation Marks



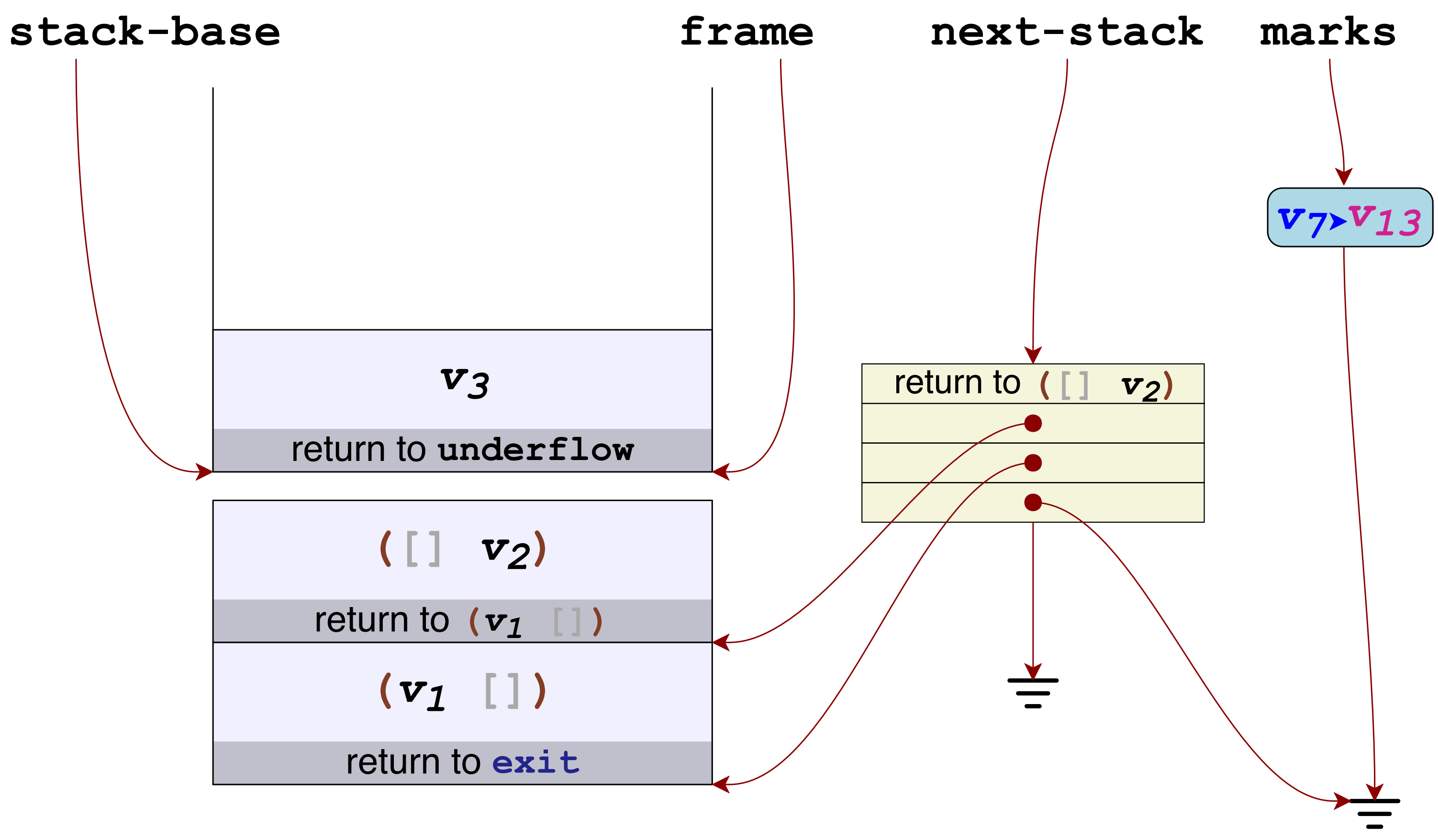
Setting Continuation Marks



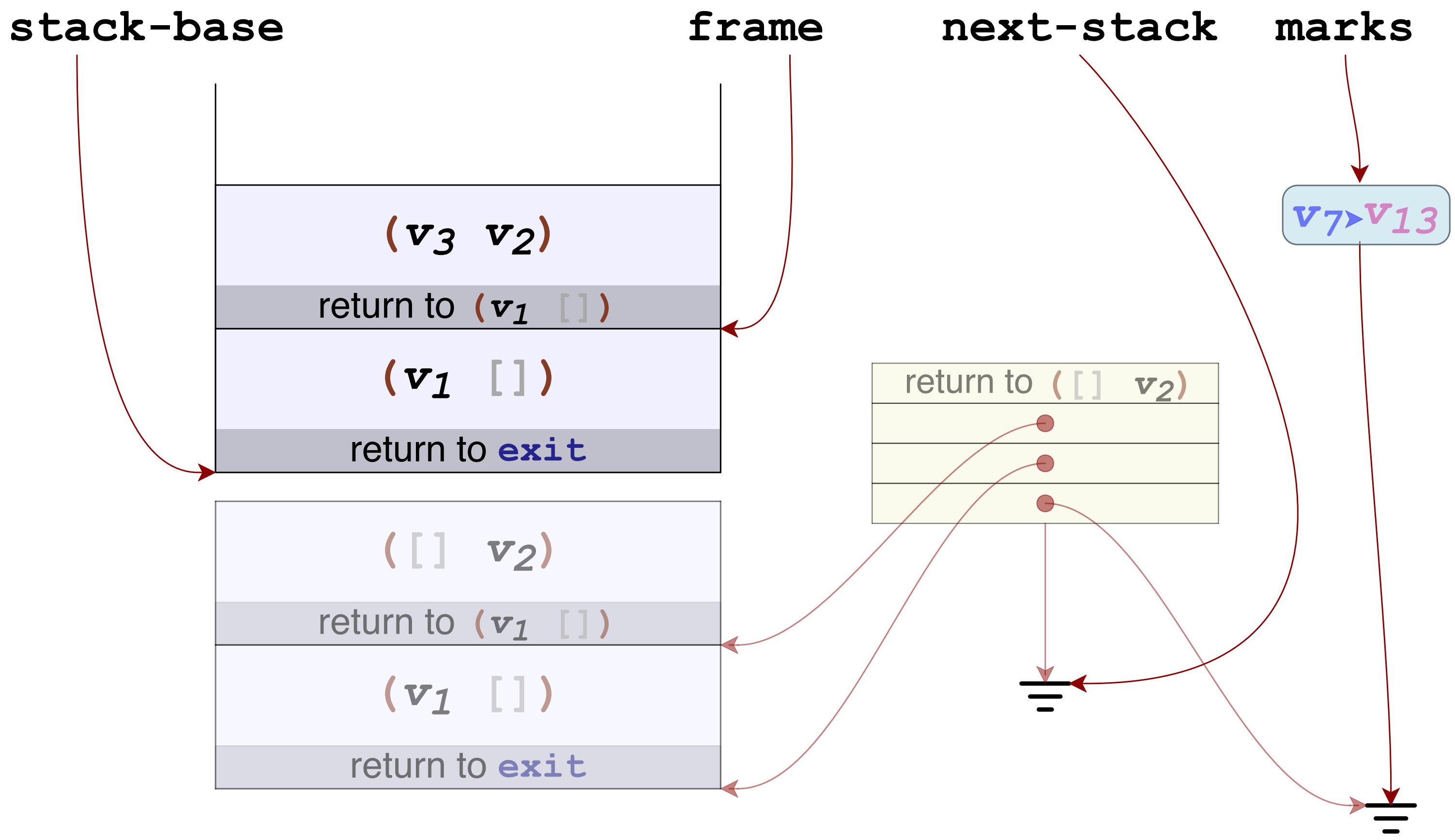
Setting Continuation Marks



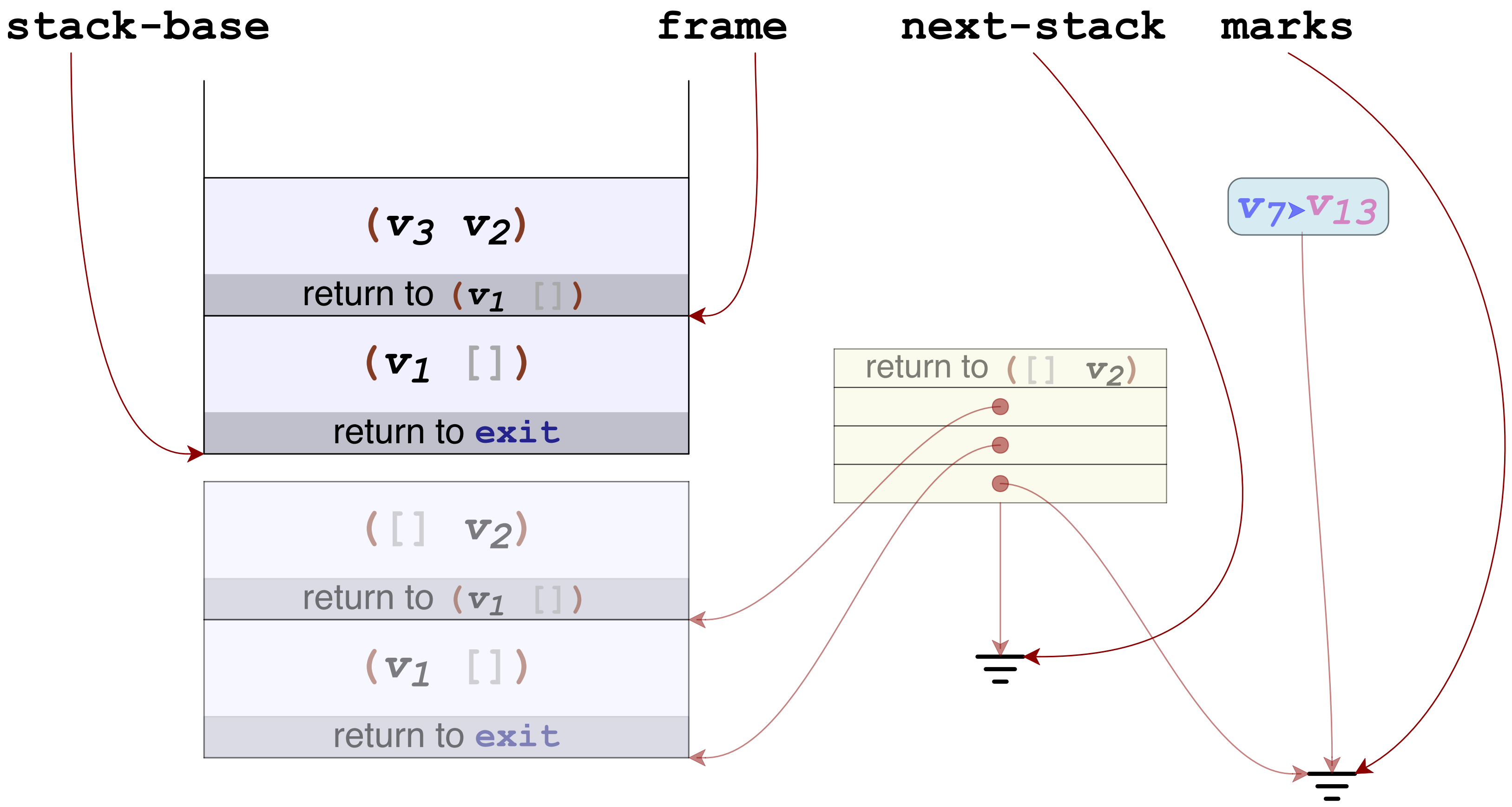
Setting Continuation Marks



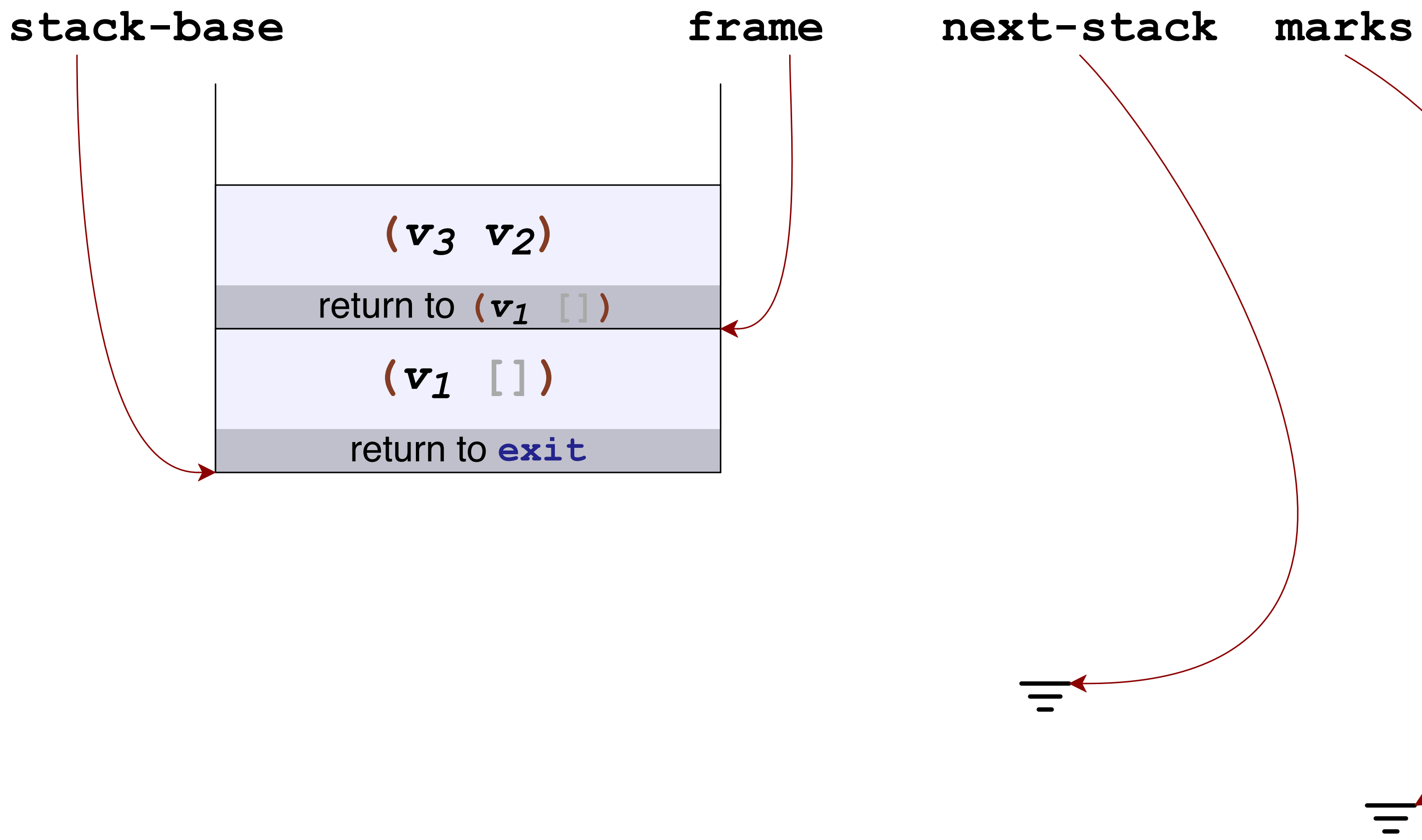
Removing Continuation Marks



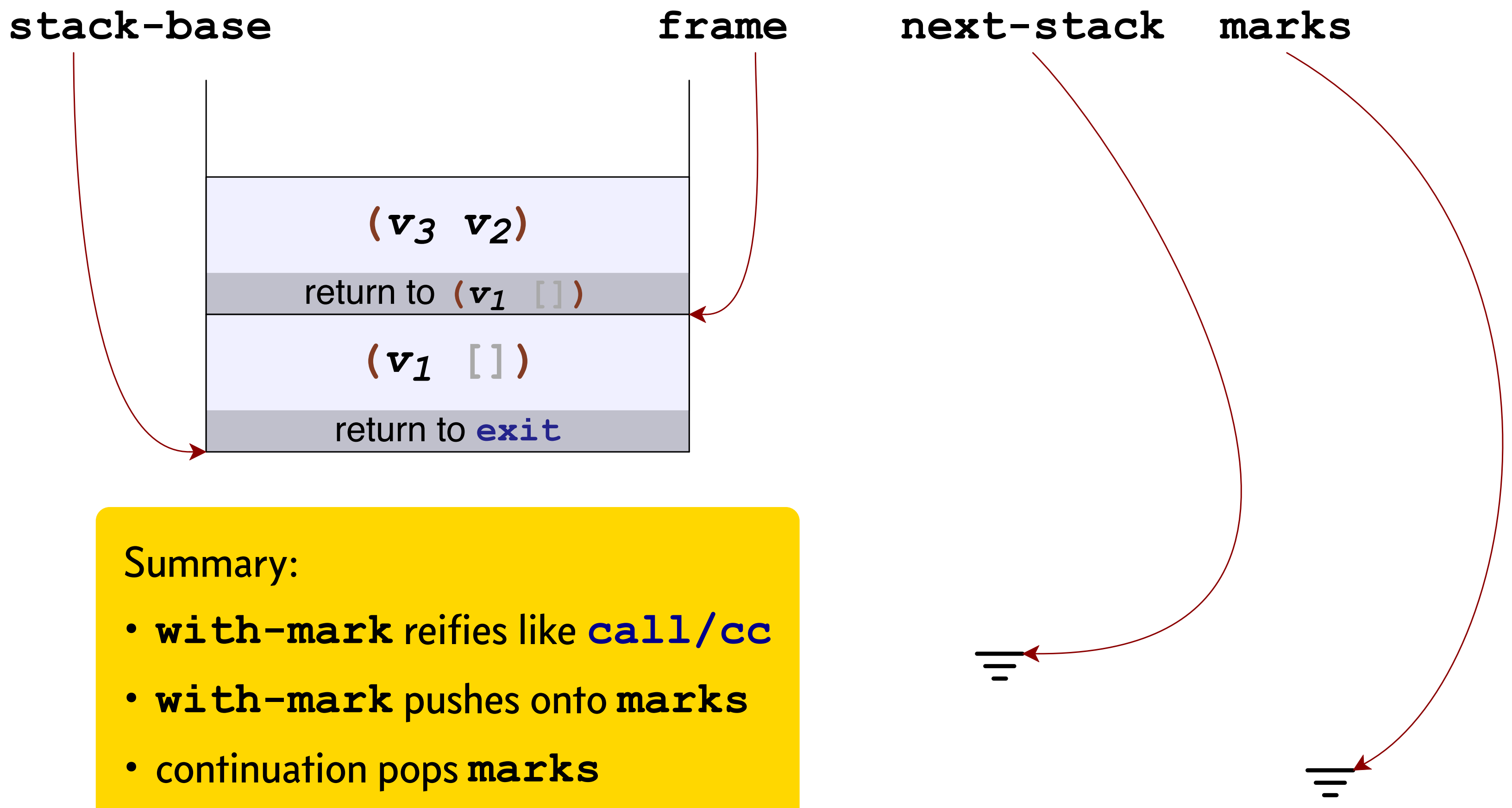
Removing Continuation Marks



Removing Continuation Marks



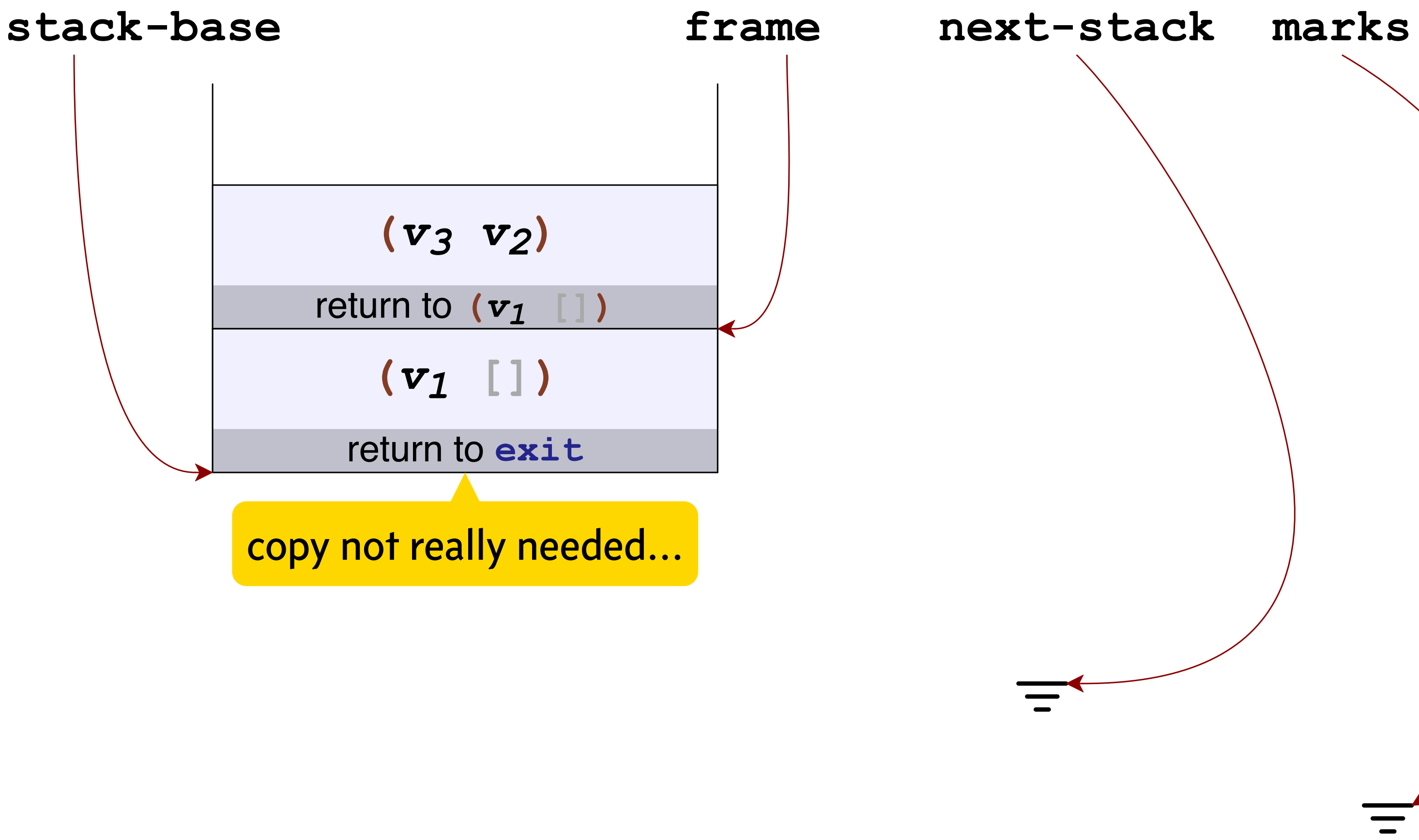
Implementing Continuation Marks



Summary:

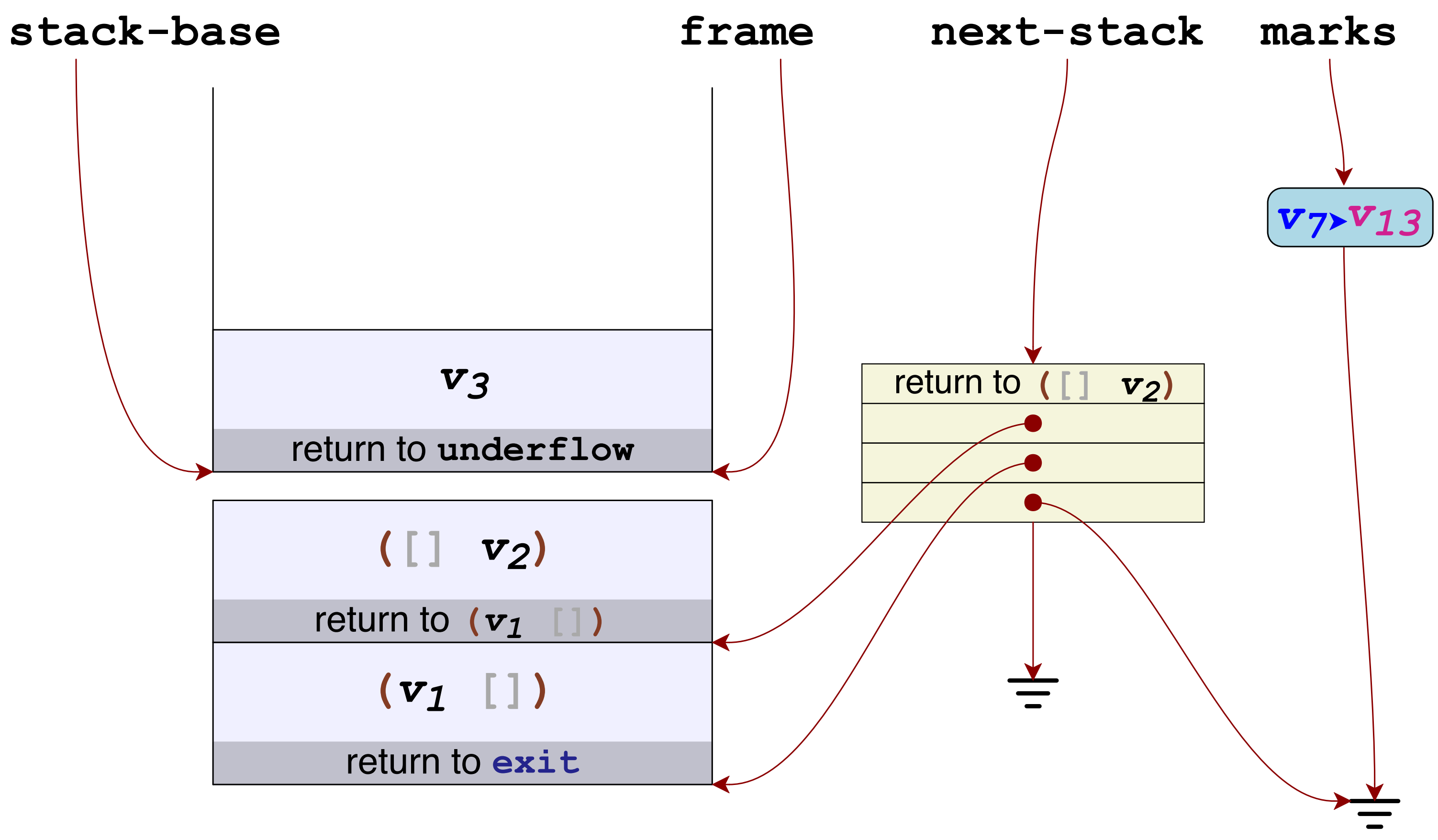
- **with-mark** reifies like **call/cc**
- **with-mark** pushes onto **marks**
- continuation pops **marks**

Implementing Continuation Marks

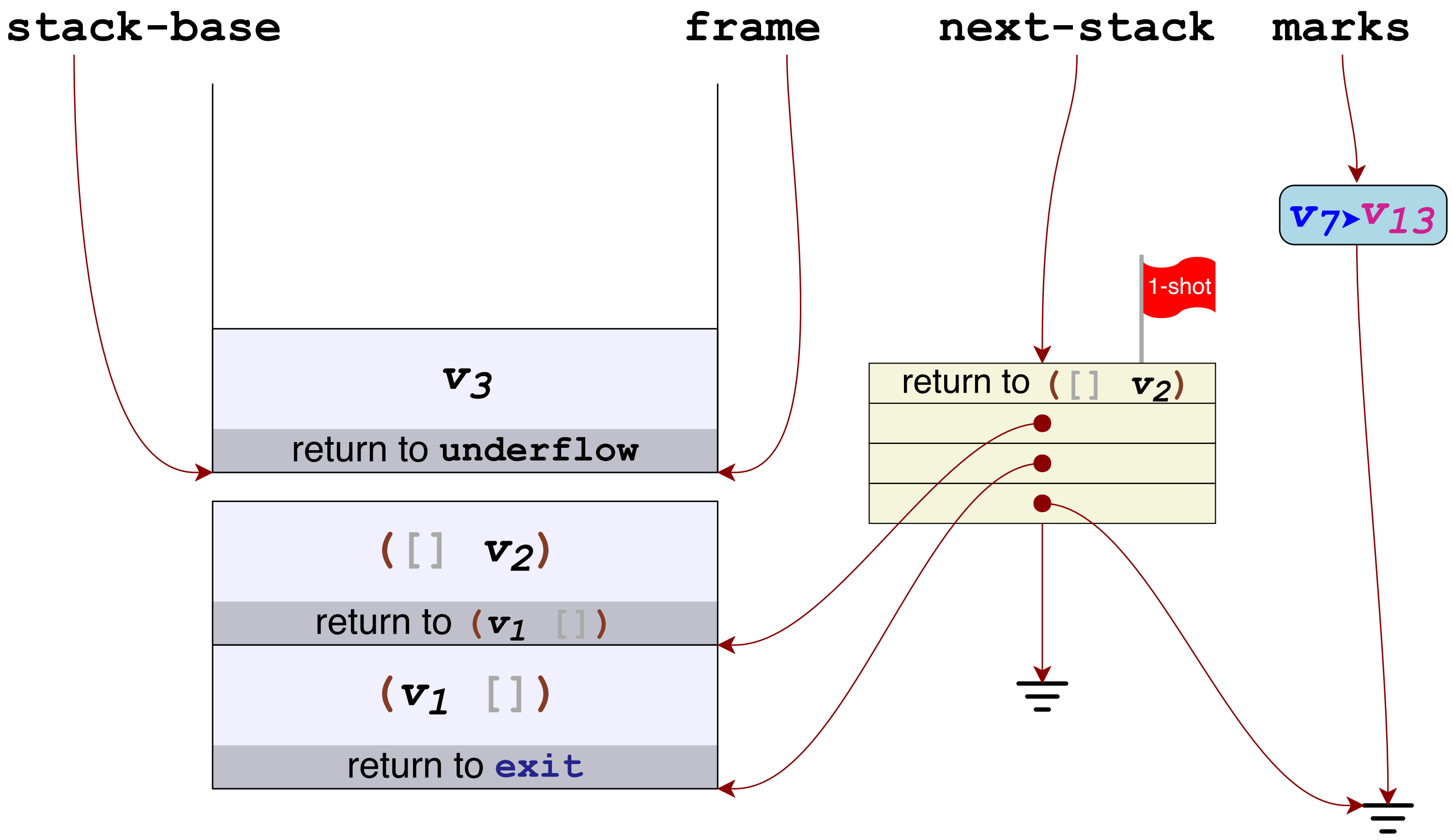


copy not really needed...

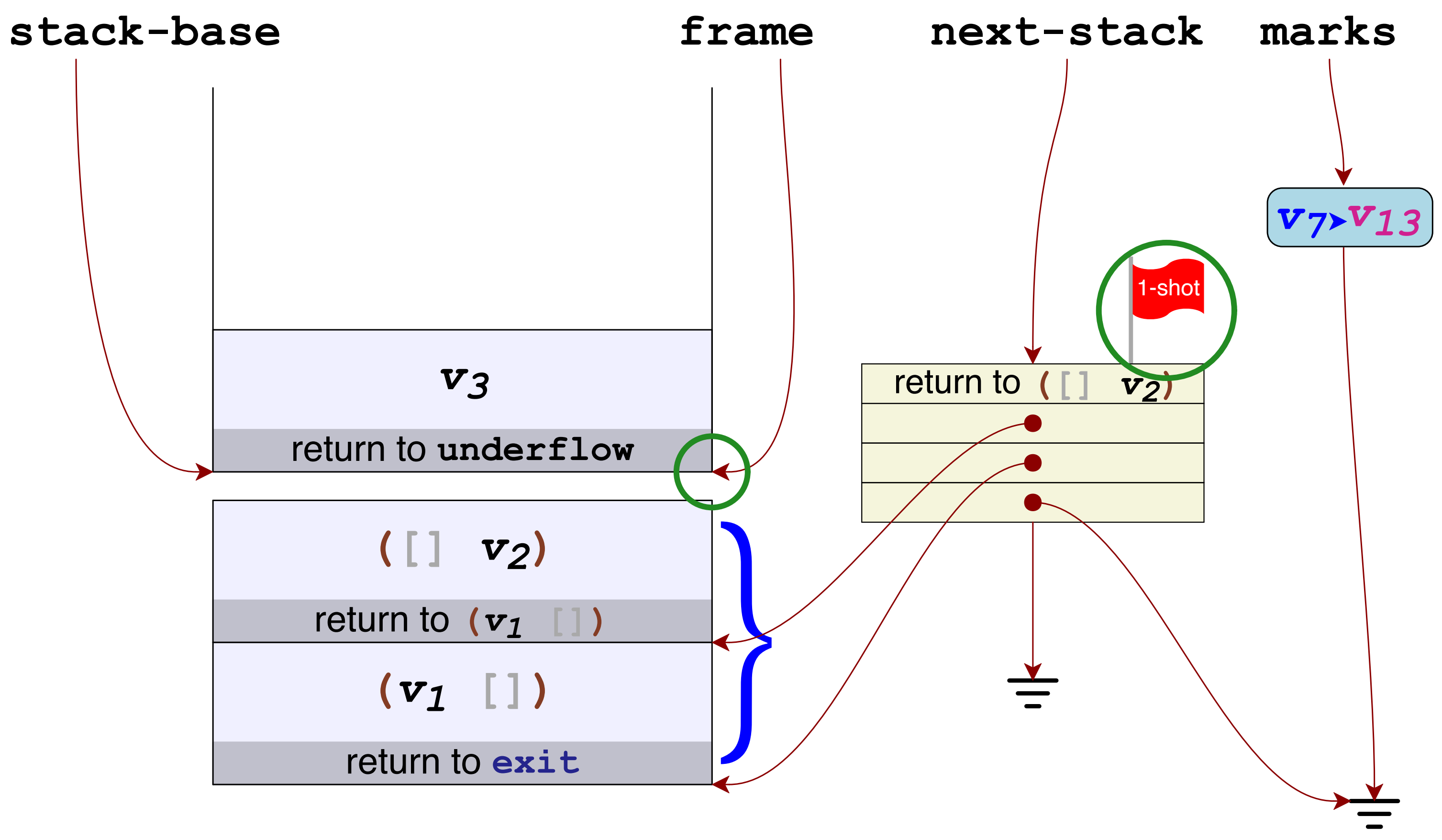
Opportunistic One-Shot Continuations



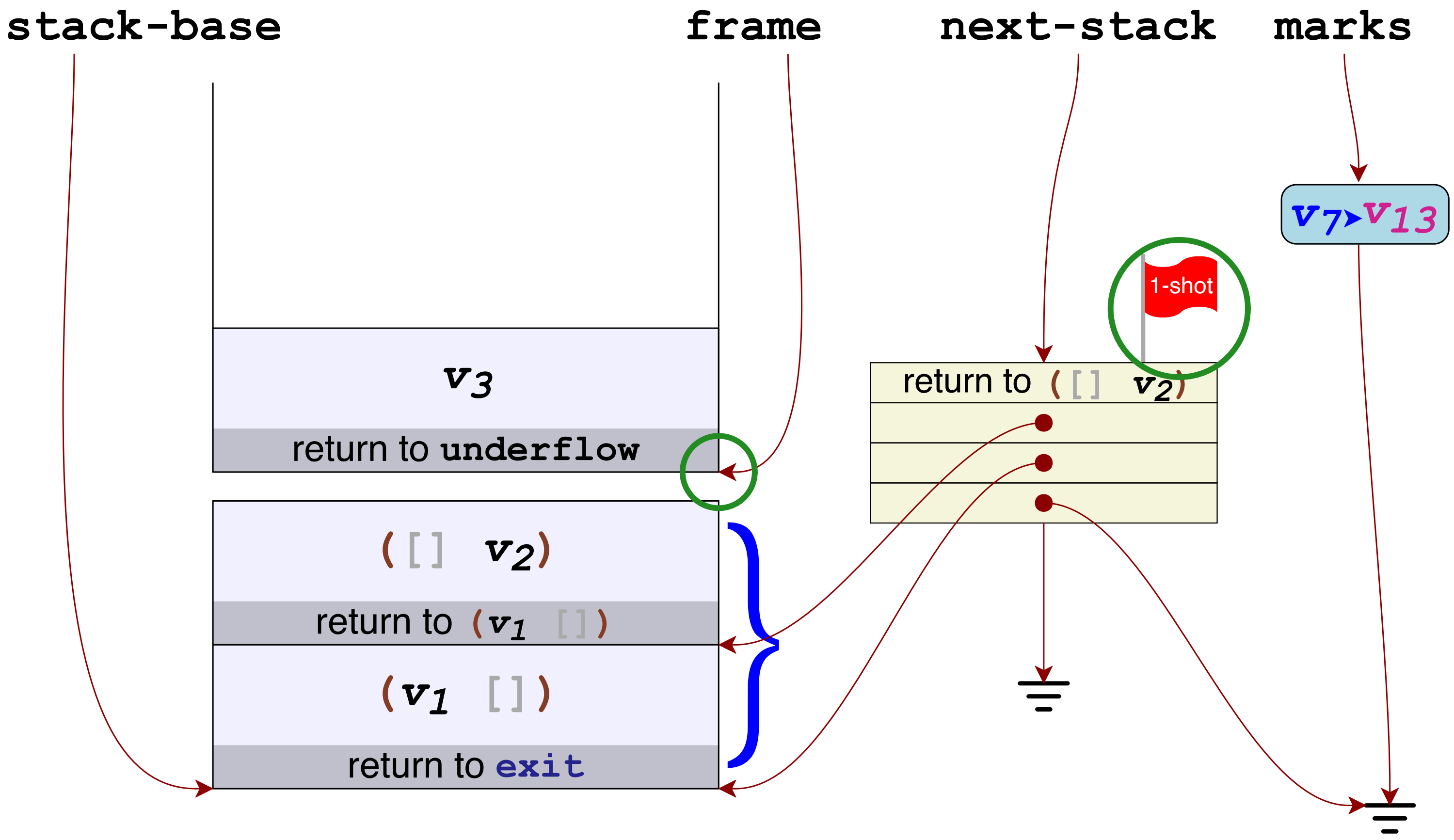
Opportunistic One-Shot Continuations



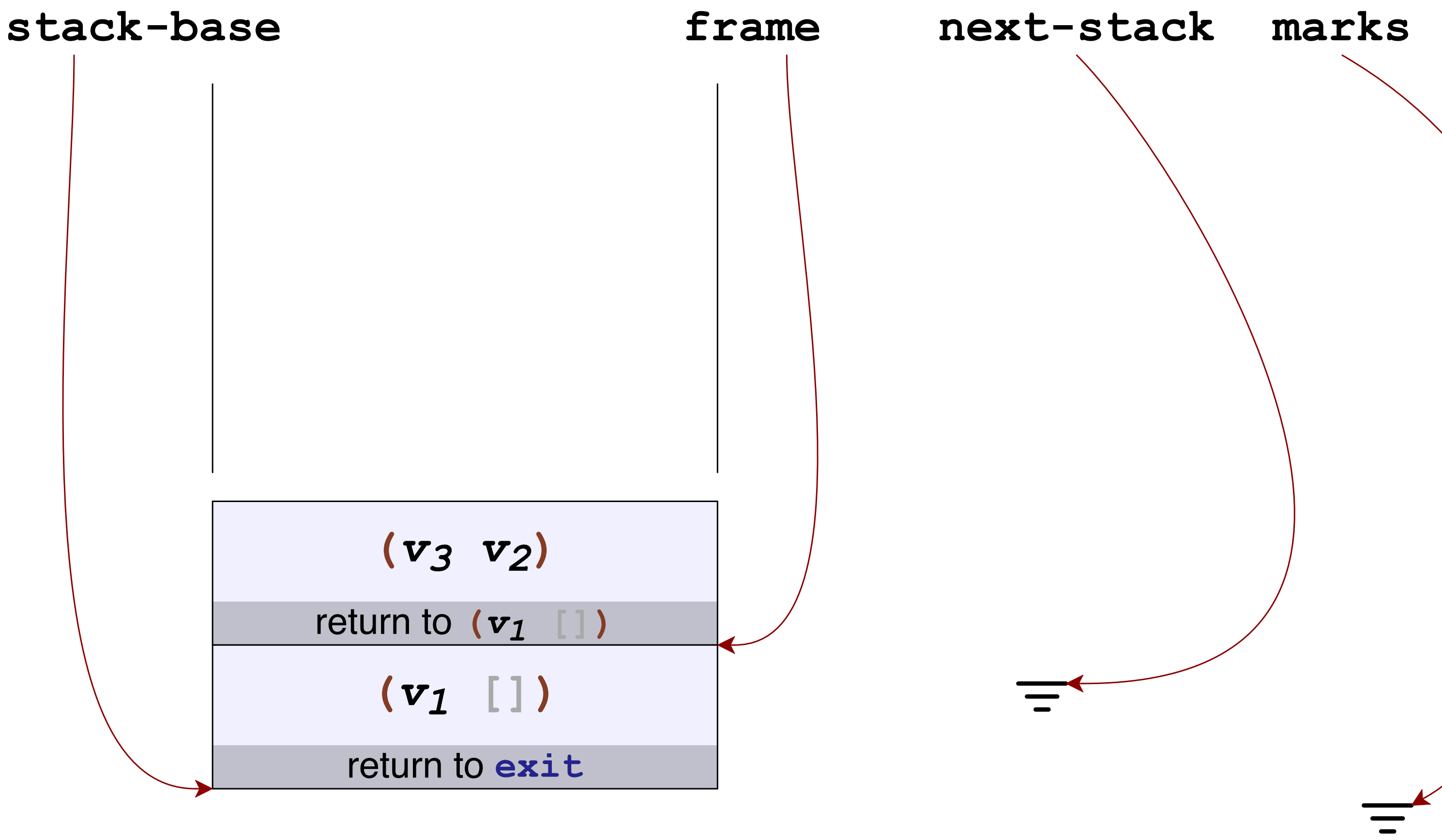
Opportunistic One-Shot Continuations



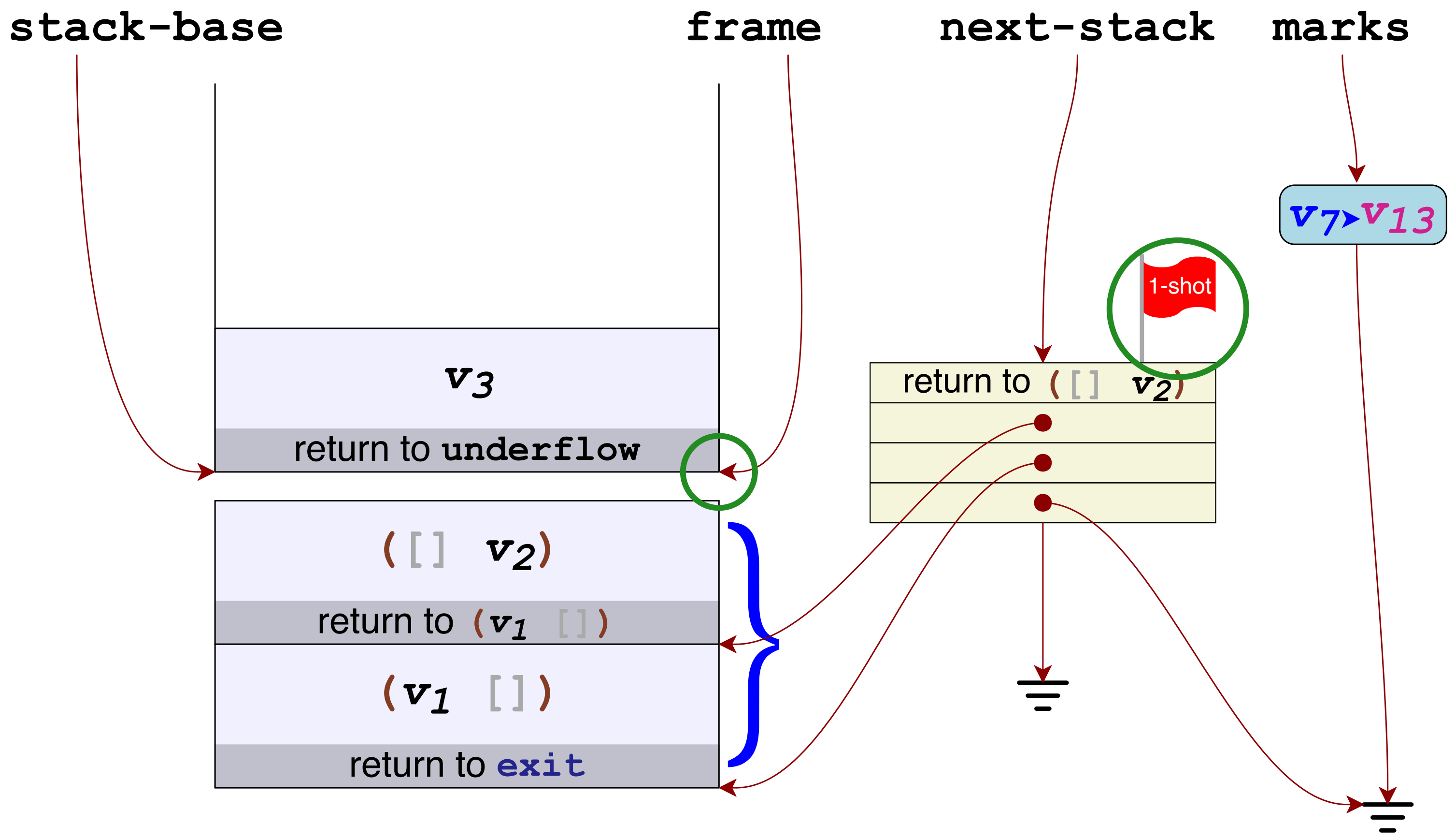
Opportunistic One-Shot Continuations



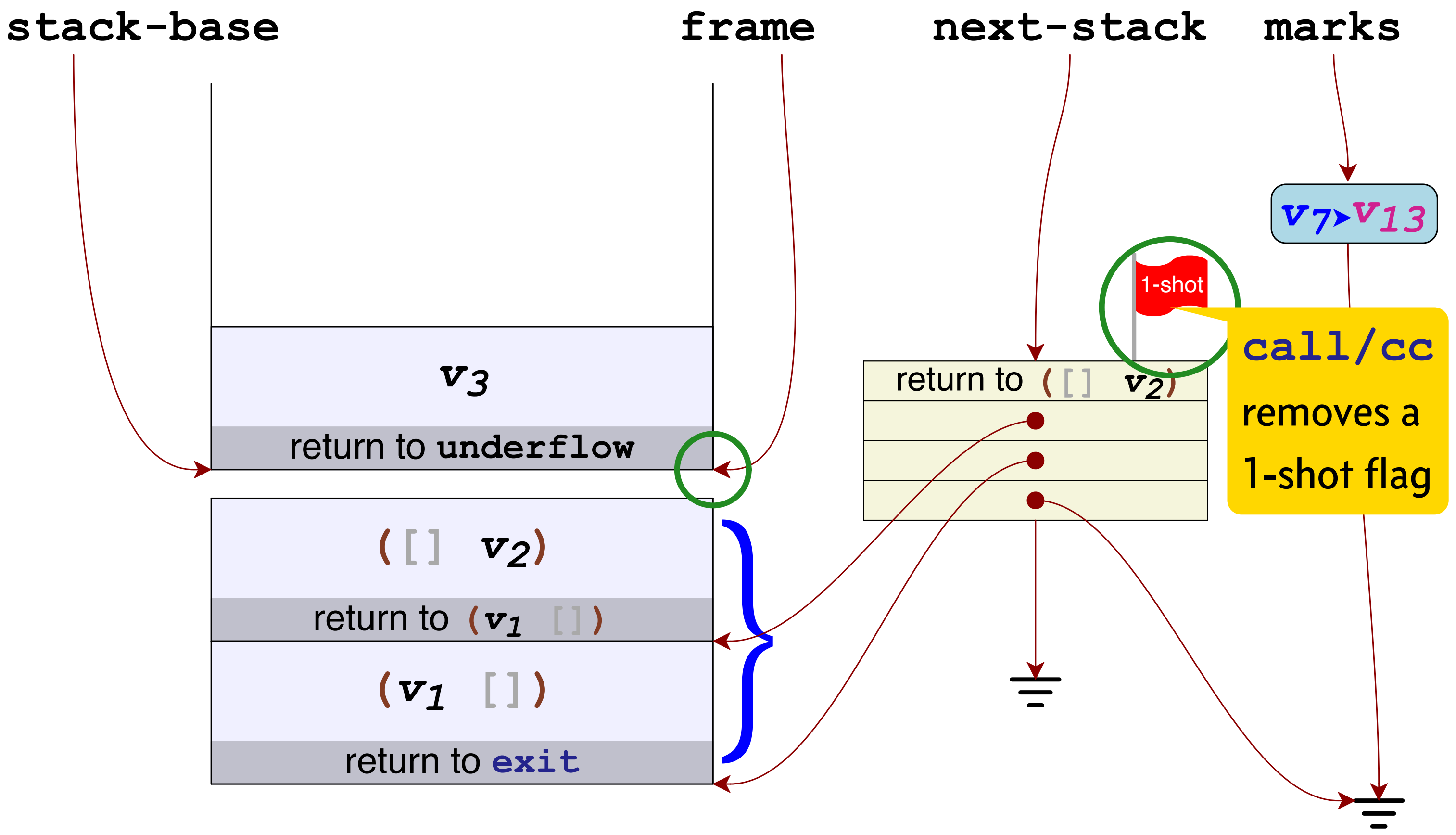
Opportunistic One-Shot Continuations



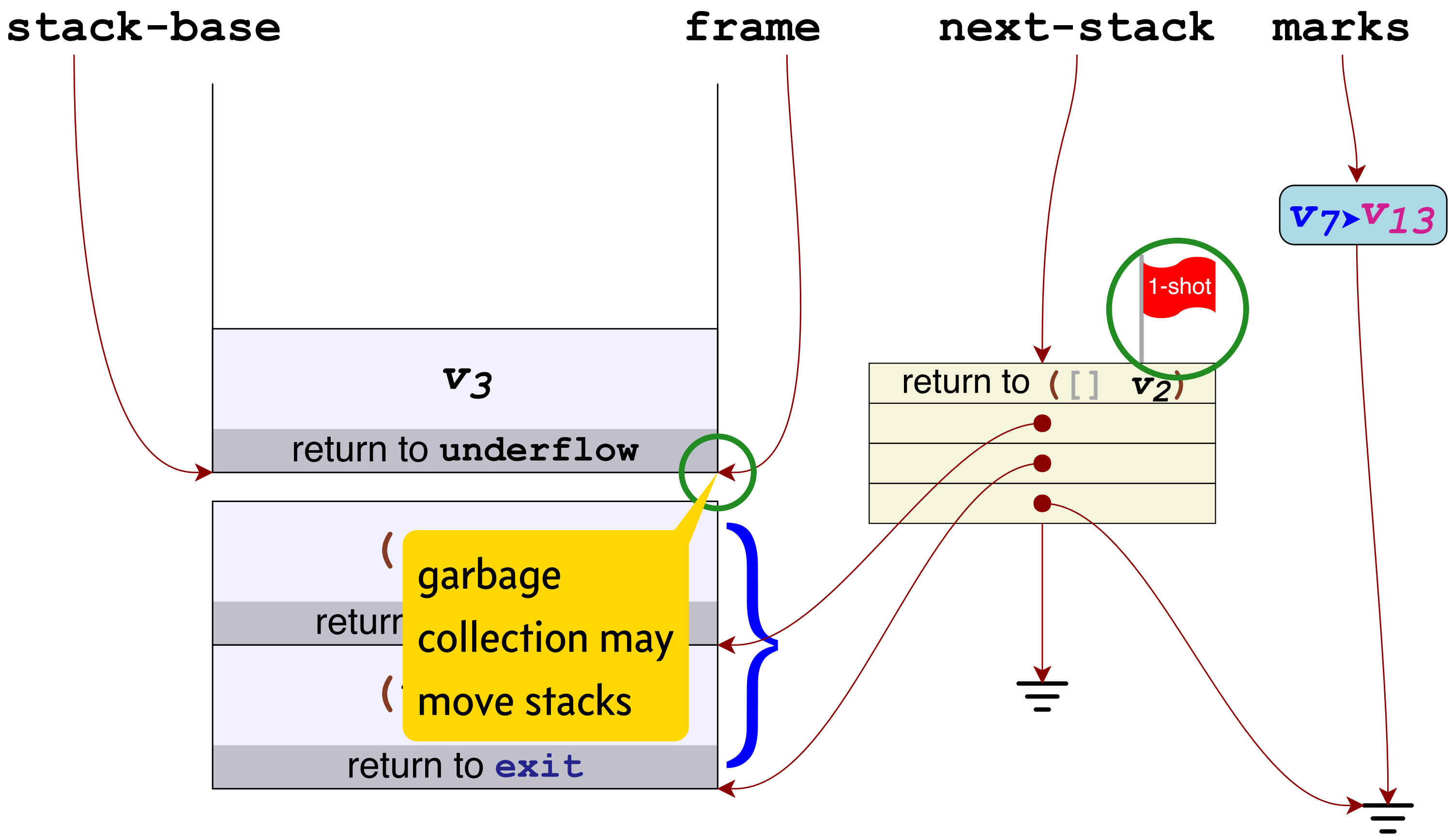
Opportunistic One-Shot Continuations



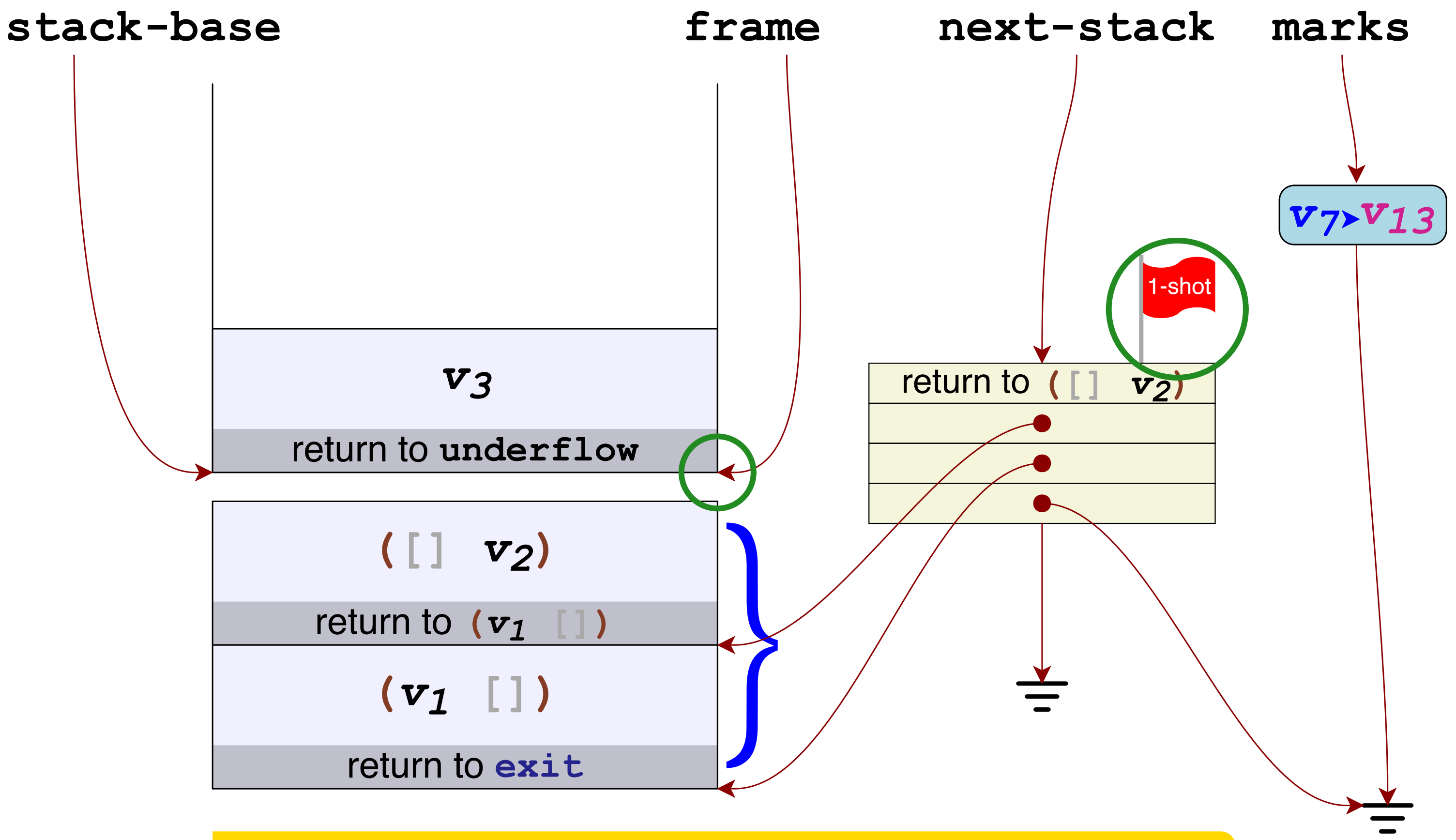
Opportunistic One-Shot Continuations



Opportunistic One-Shot Continuations

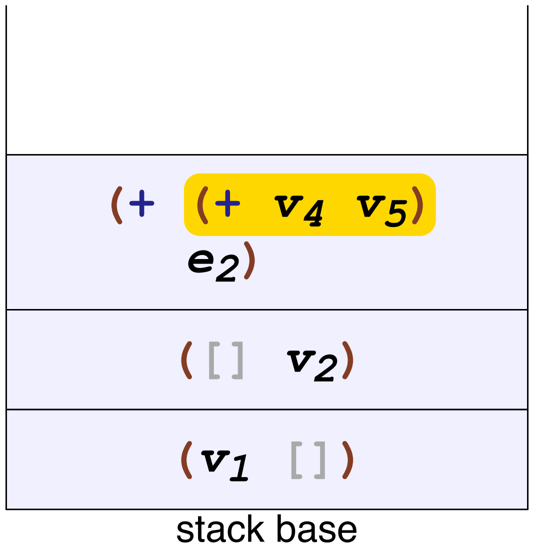


Opportunistic One-Shot Continuations

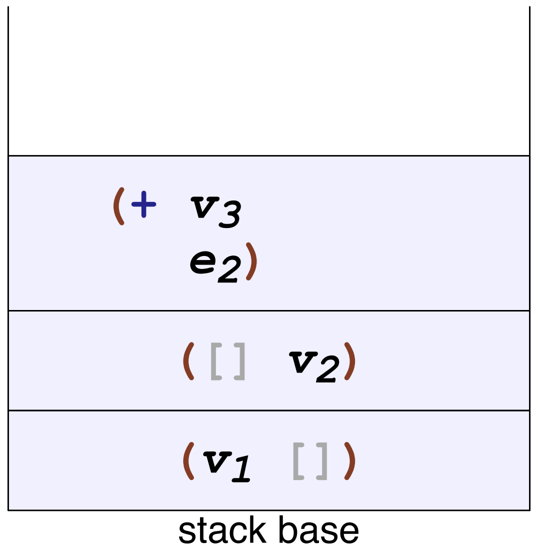


with-mark without **call/cc**: frequent opportunity

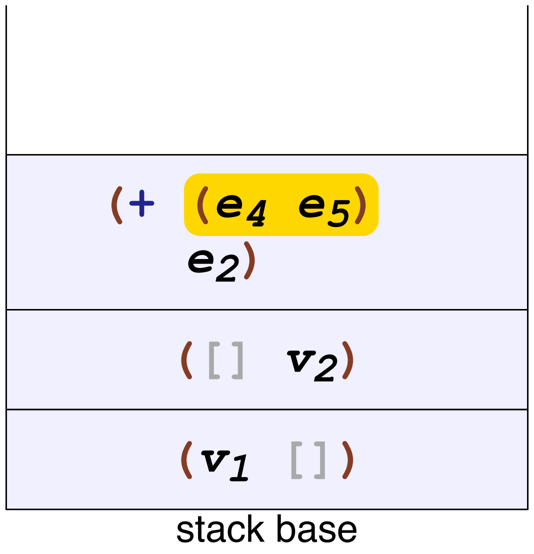
Primitives: Don't Create a Frame



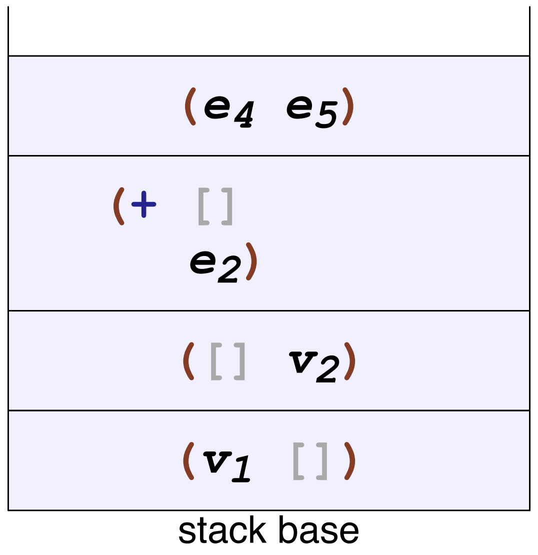
Primitives: Don't Create a Frame



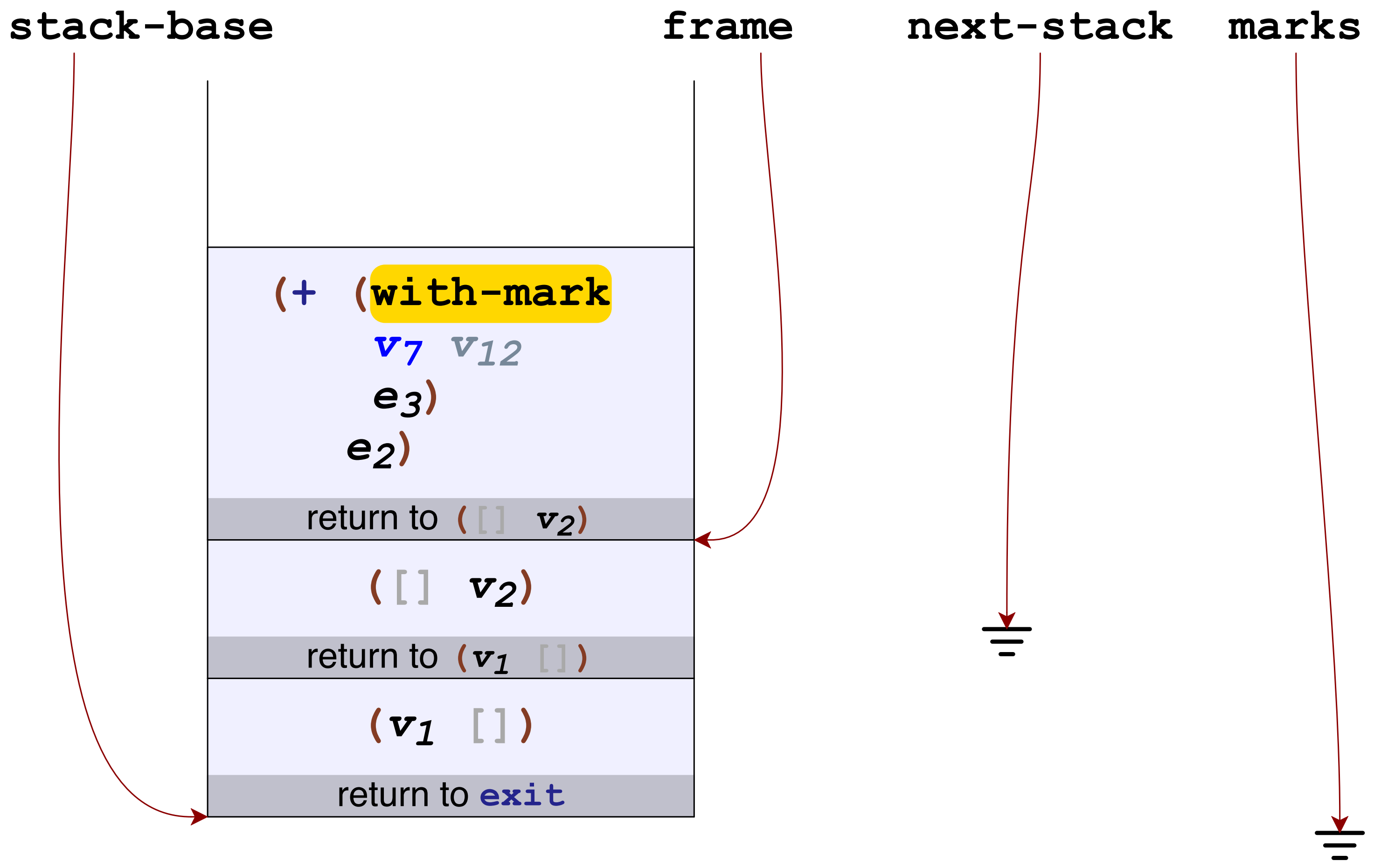
Function Call: Do Create a Frame



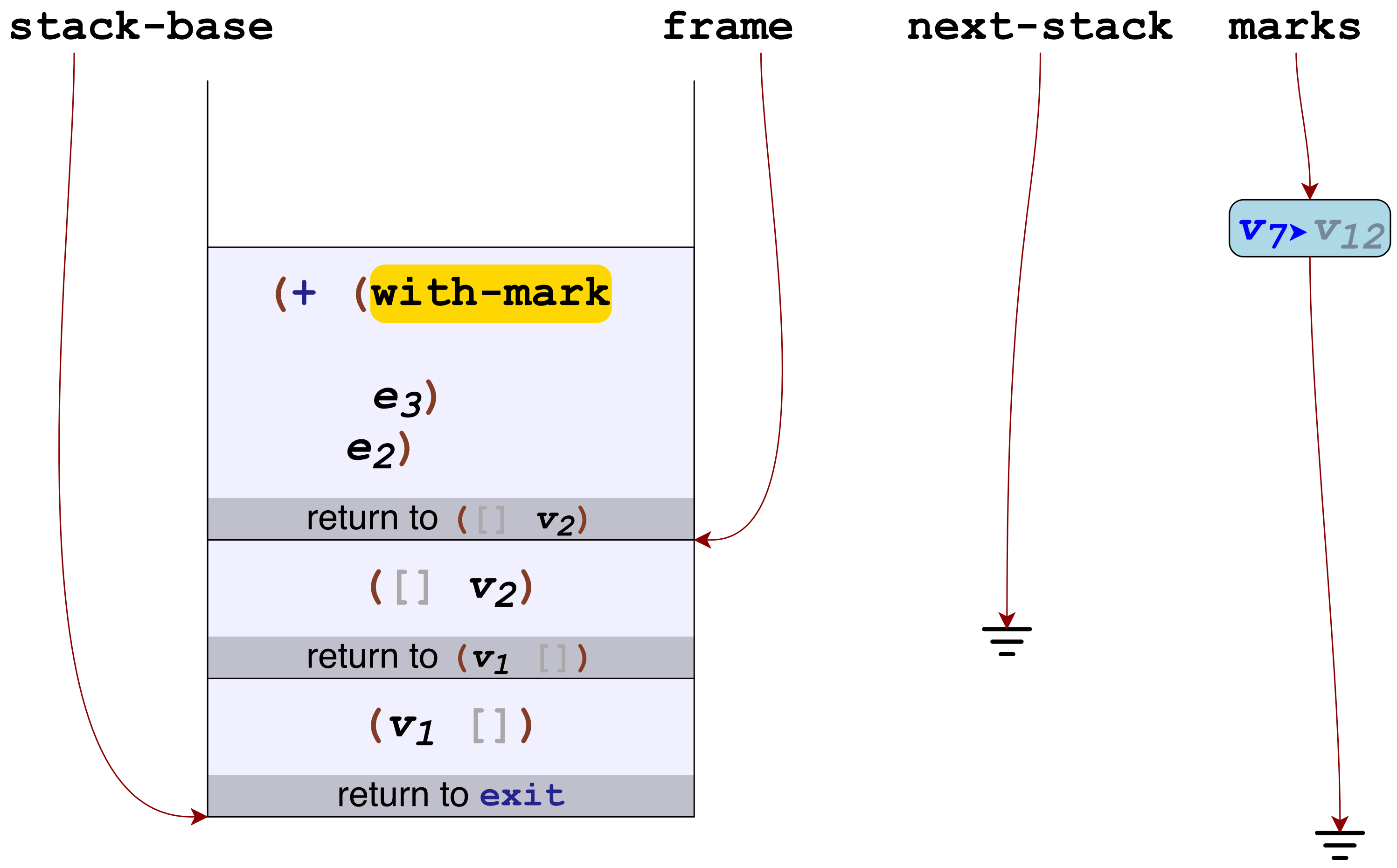
Function Call: Do Create a Frame



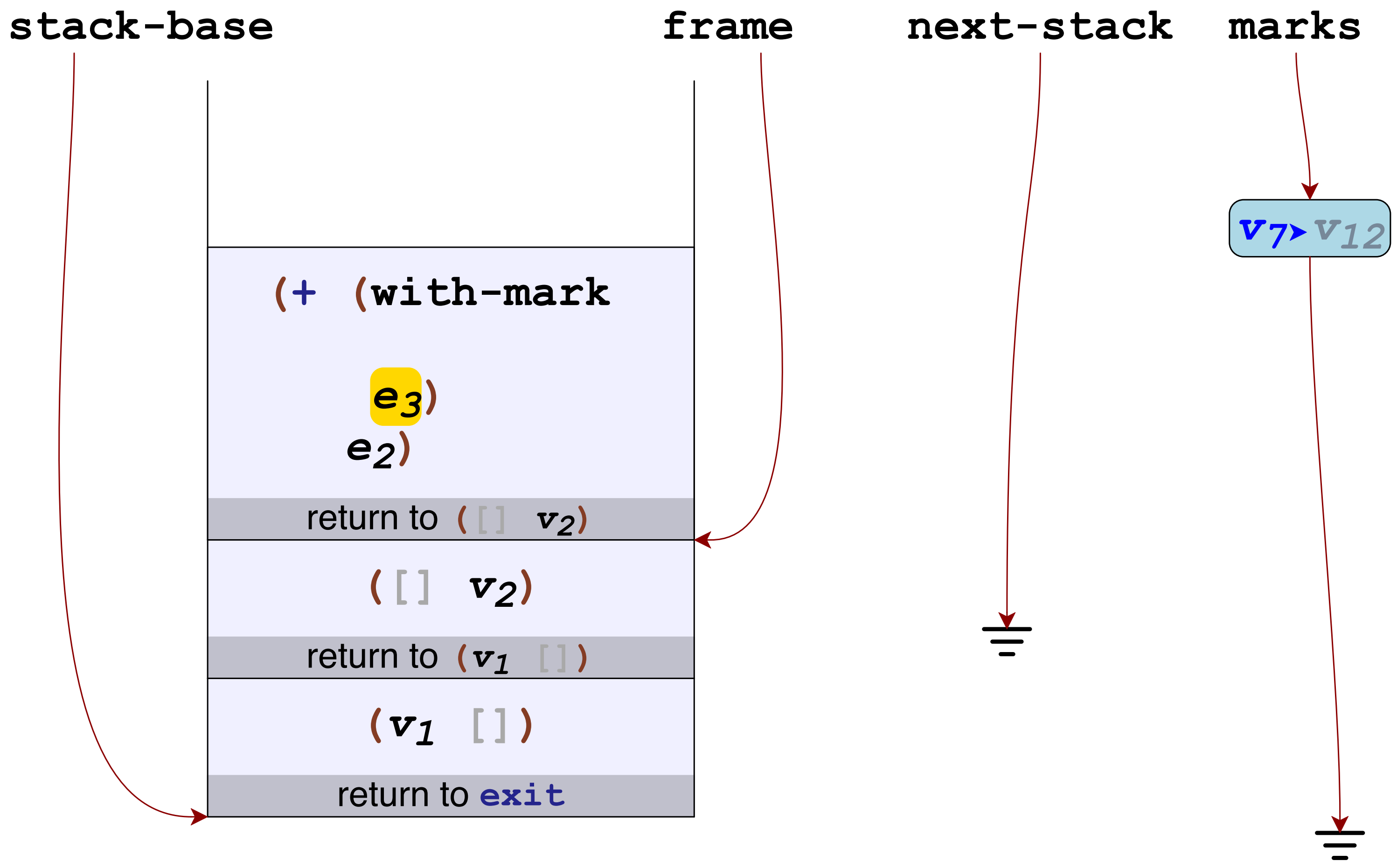
Non-Tail Mark: Don't Create a Frame



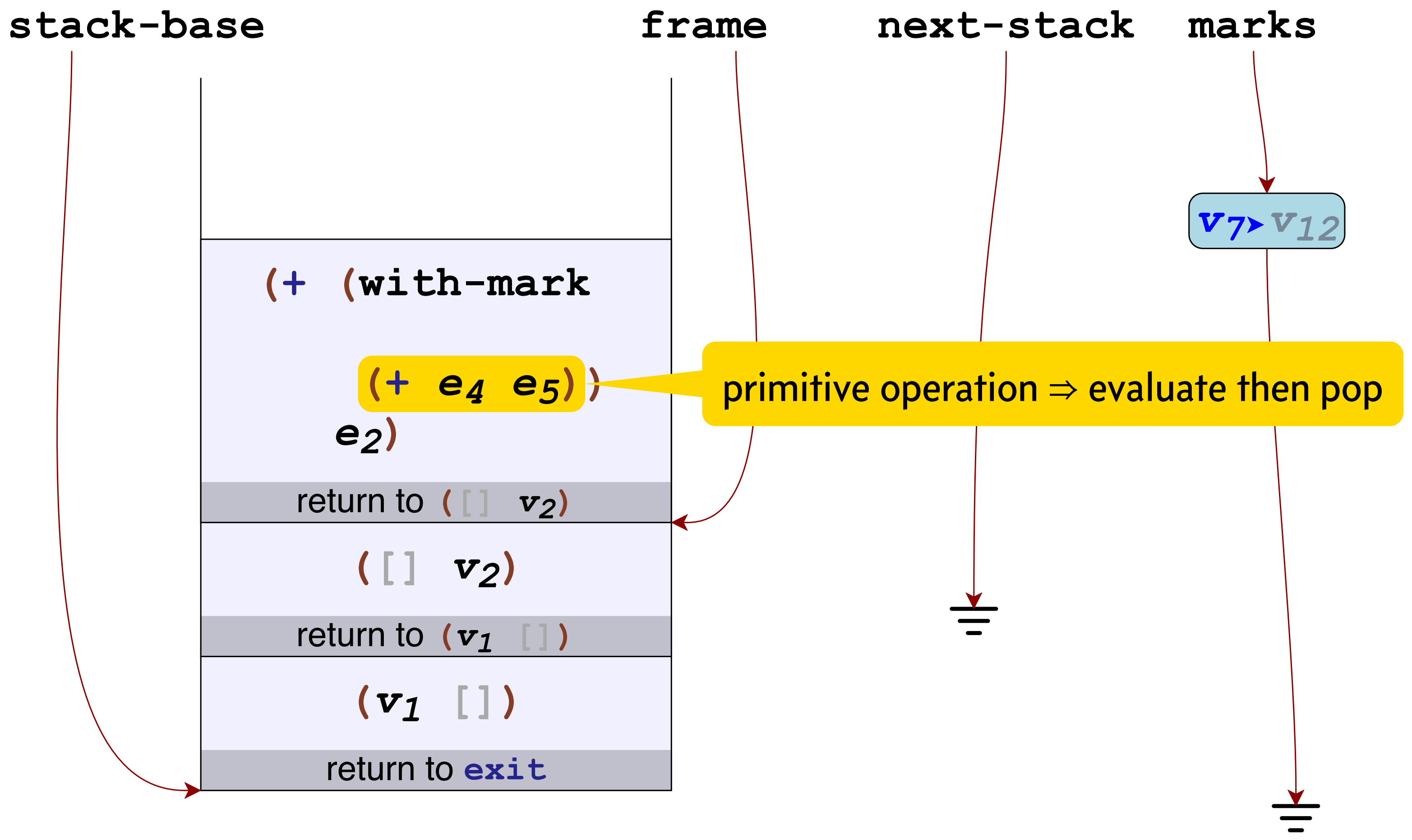
Non-Tail Mark: Don't Create a Frame



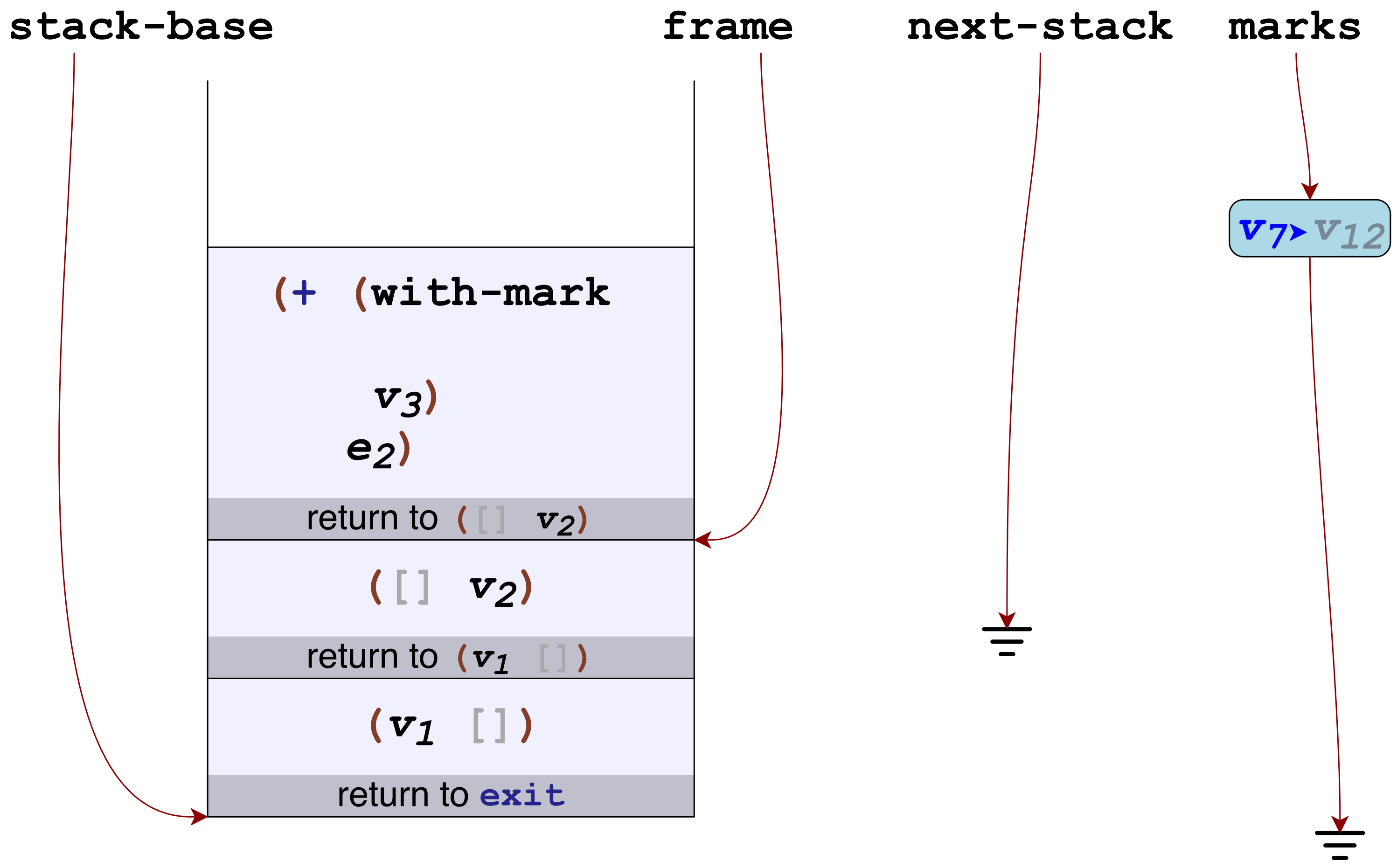
Non-Tail Mark: Don't Create a Frame



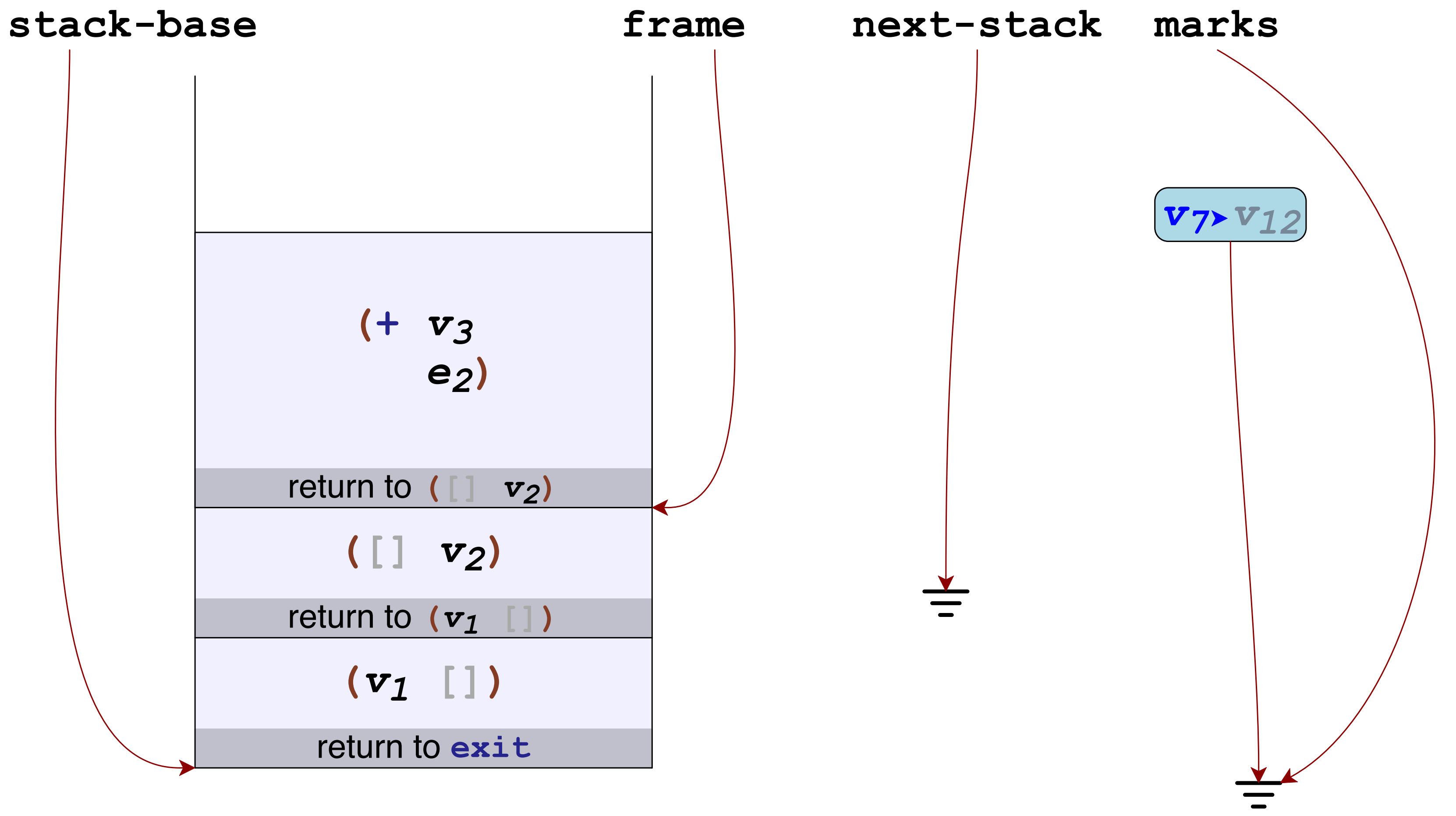
Non-Tail Mark with a Primitive Operation



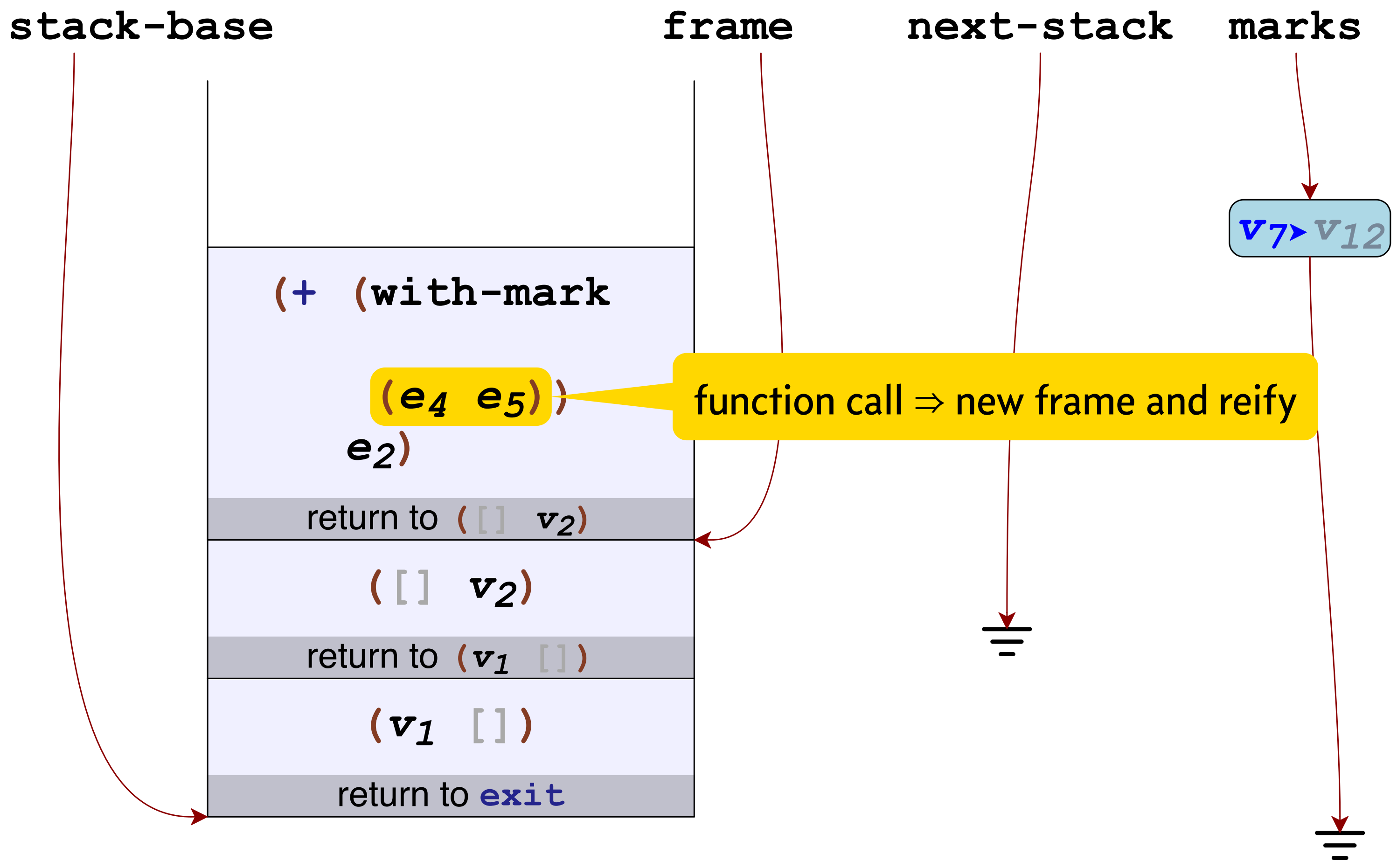
Non-Tail Mark with a Primitive Operation



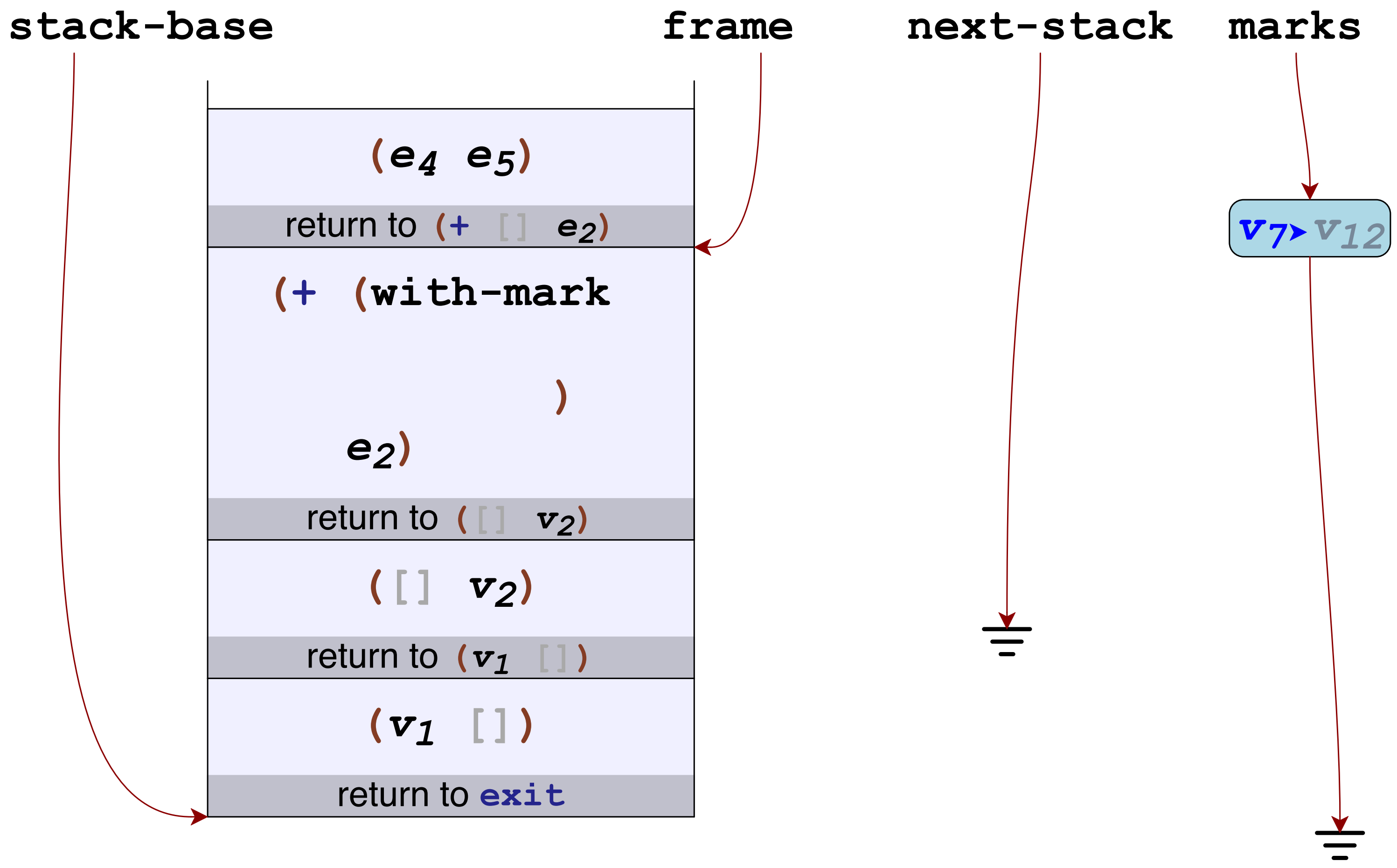
Non-Tail Mark with a Primitive Operation



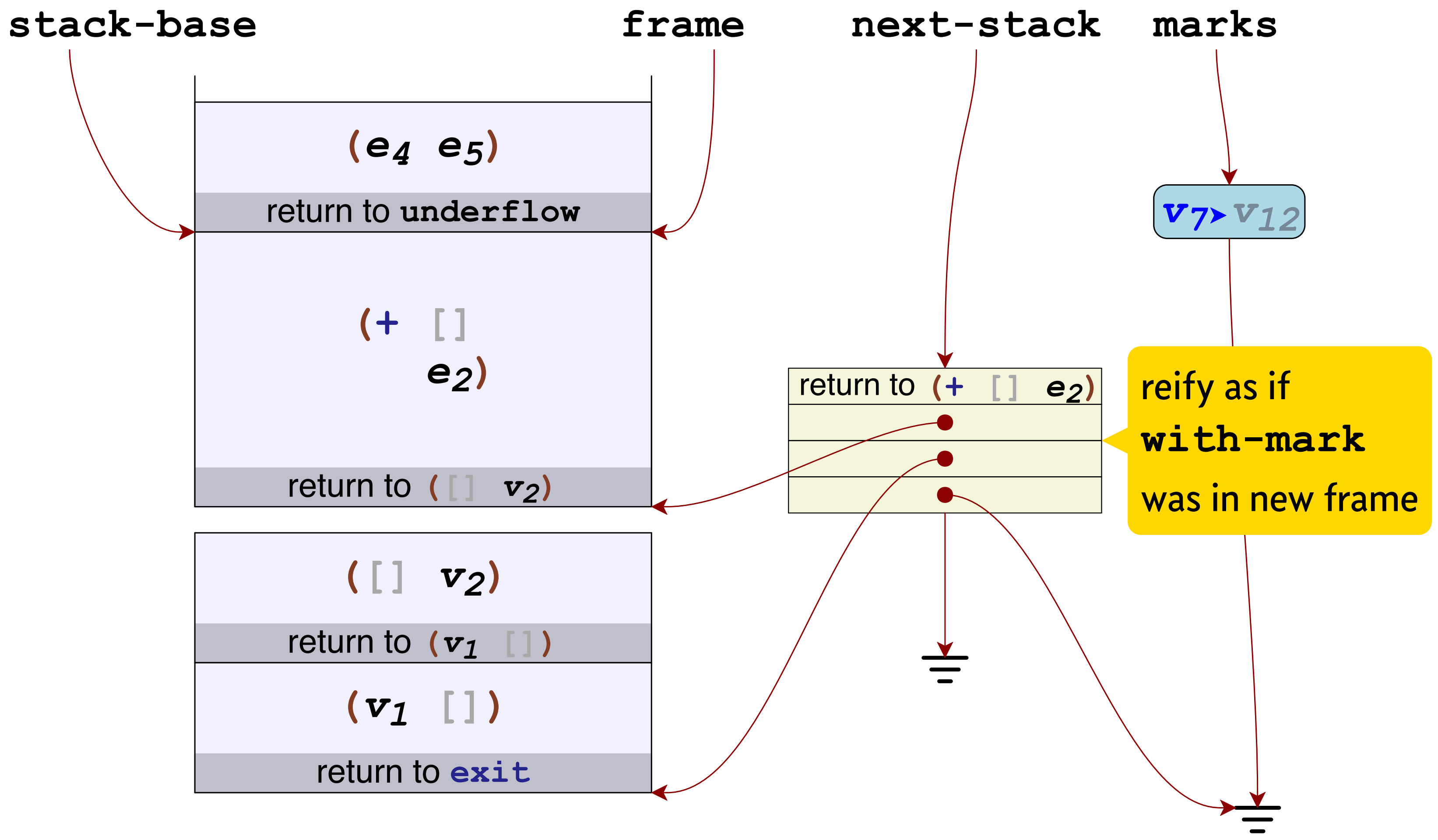
Non-Tail Mark with a Function Call



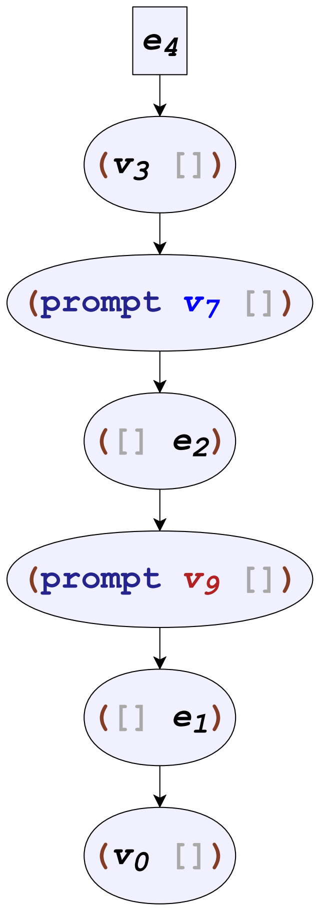
Non-Tail Mark with a Function Call



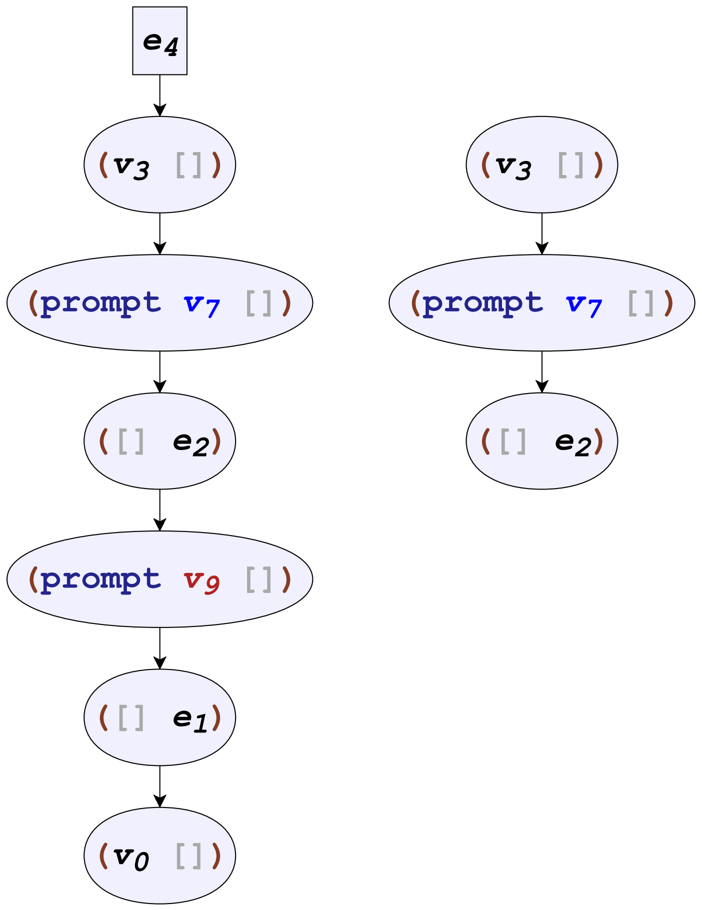
Non-Tail Mark with a Function Call



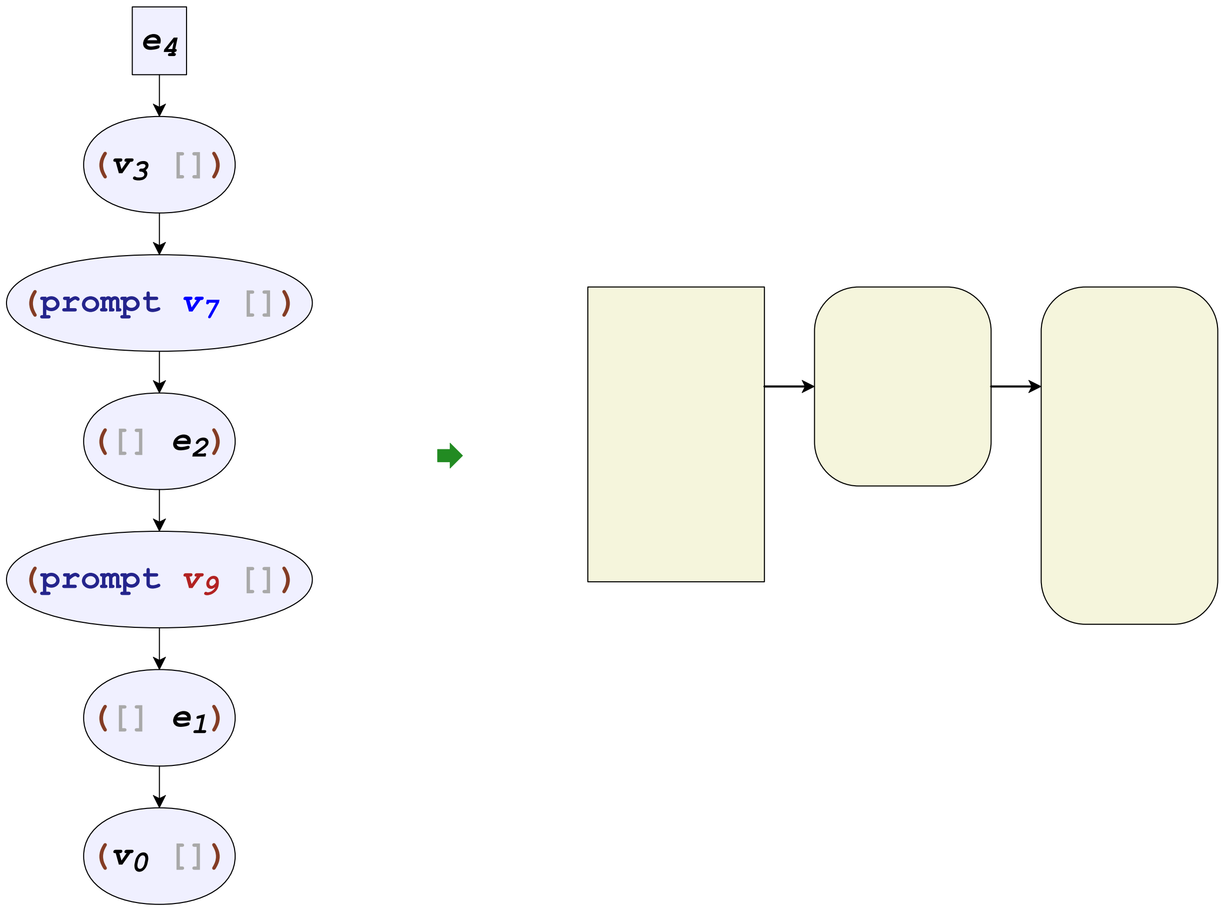
Capturing Delimited Continuations



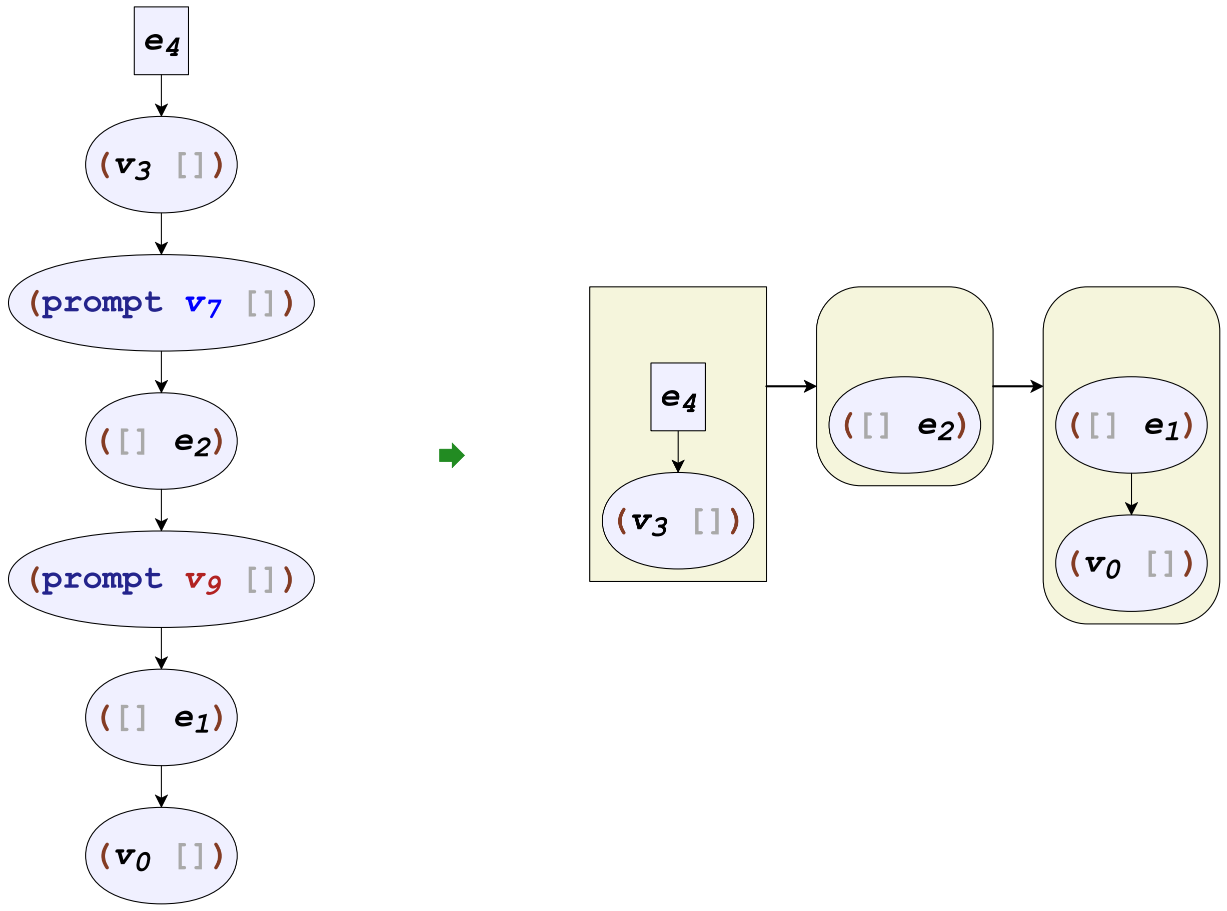
Capturing Delimited Continuations



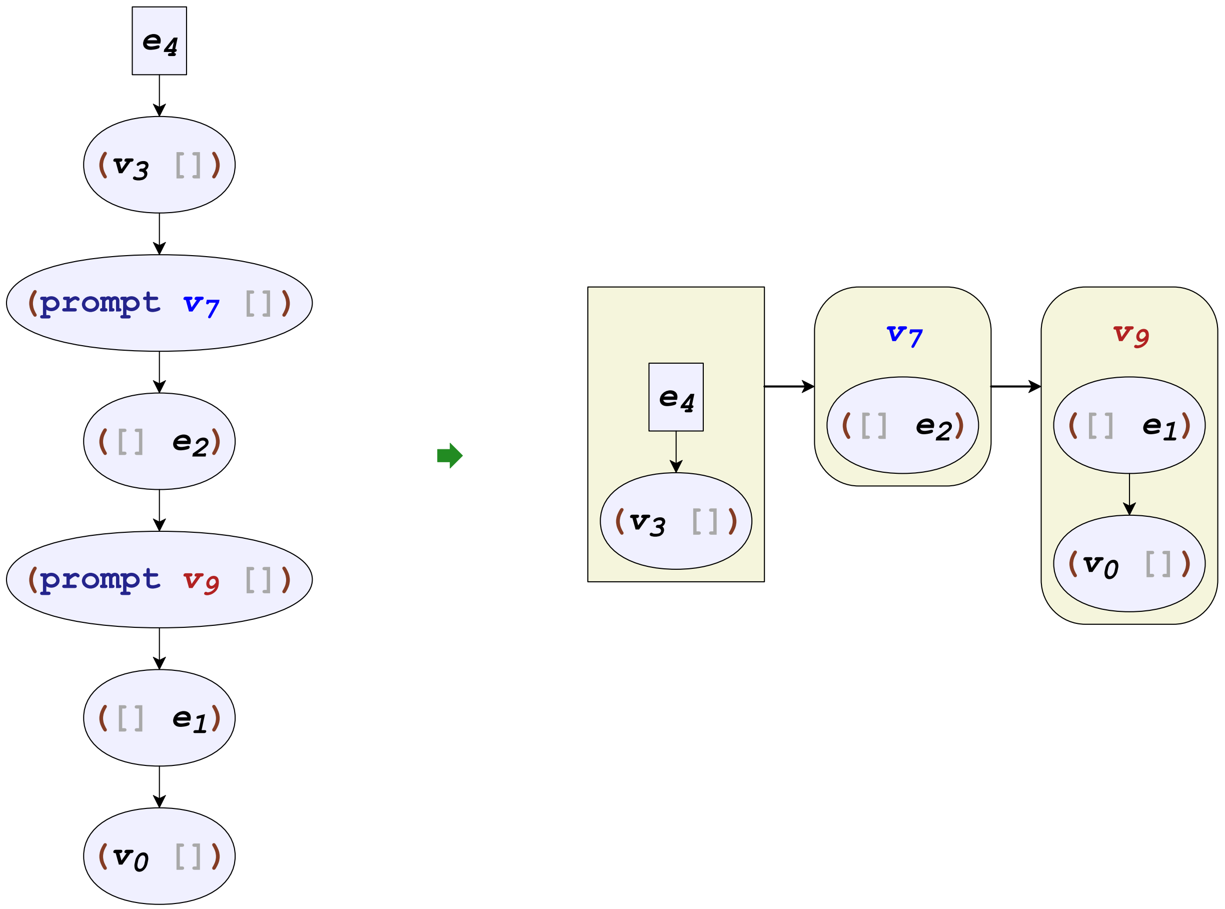
Delimited Continuations via Metacontinuations



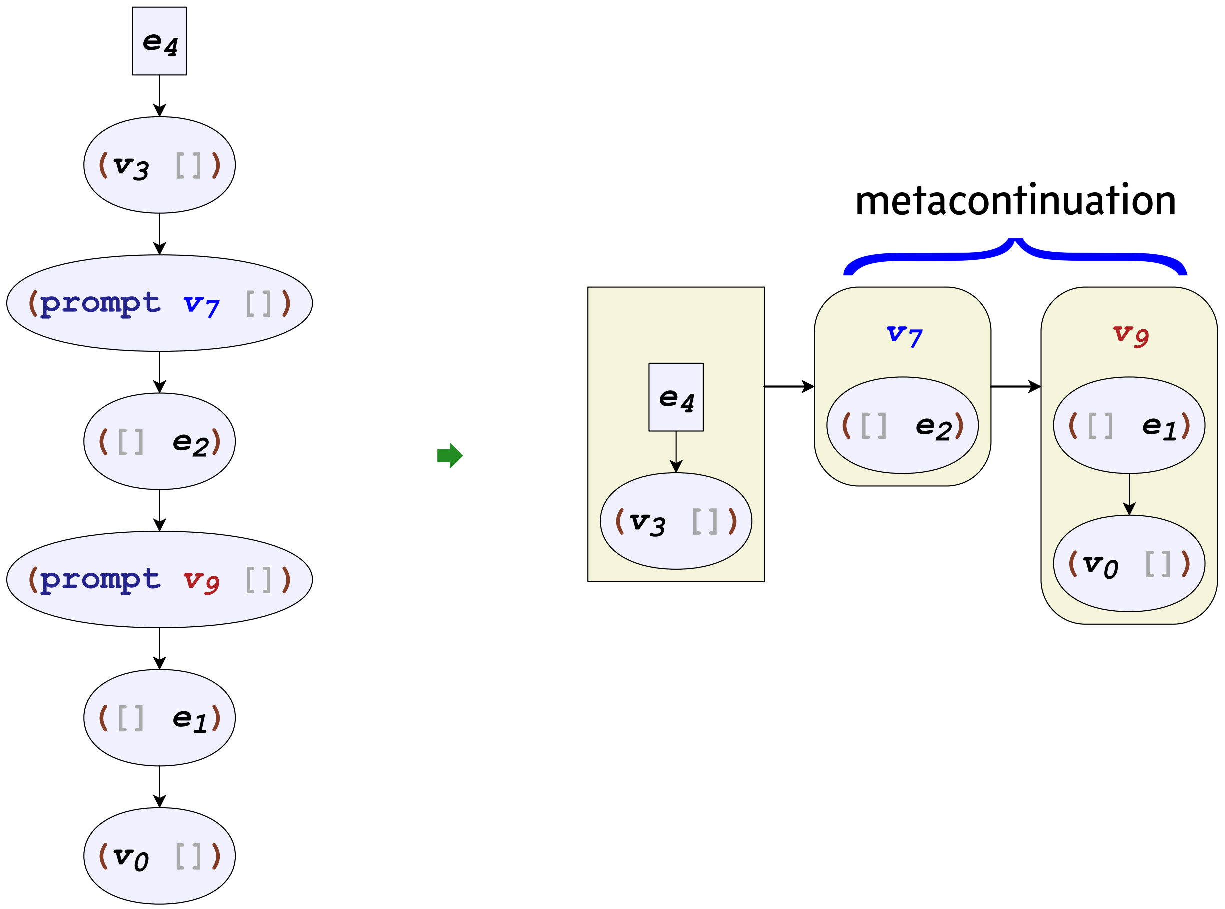
Delimited Continuations via Metacontinuations



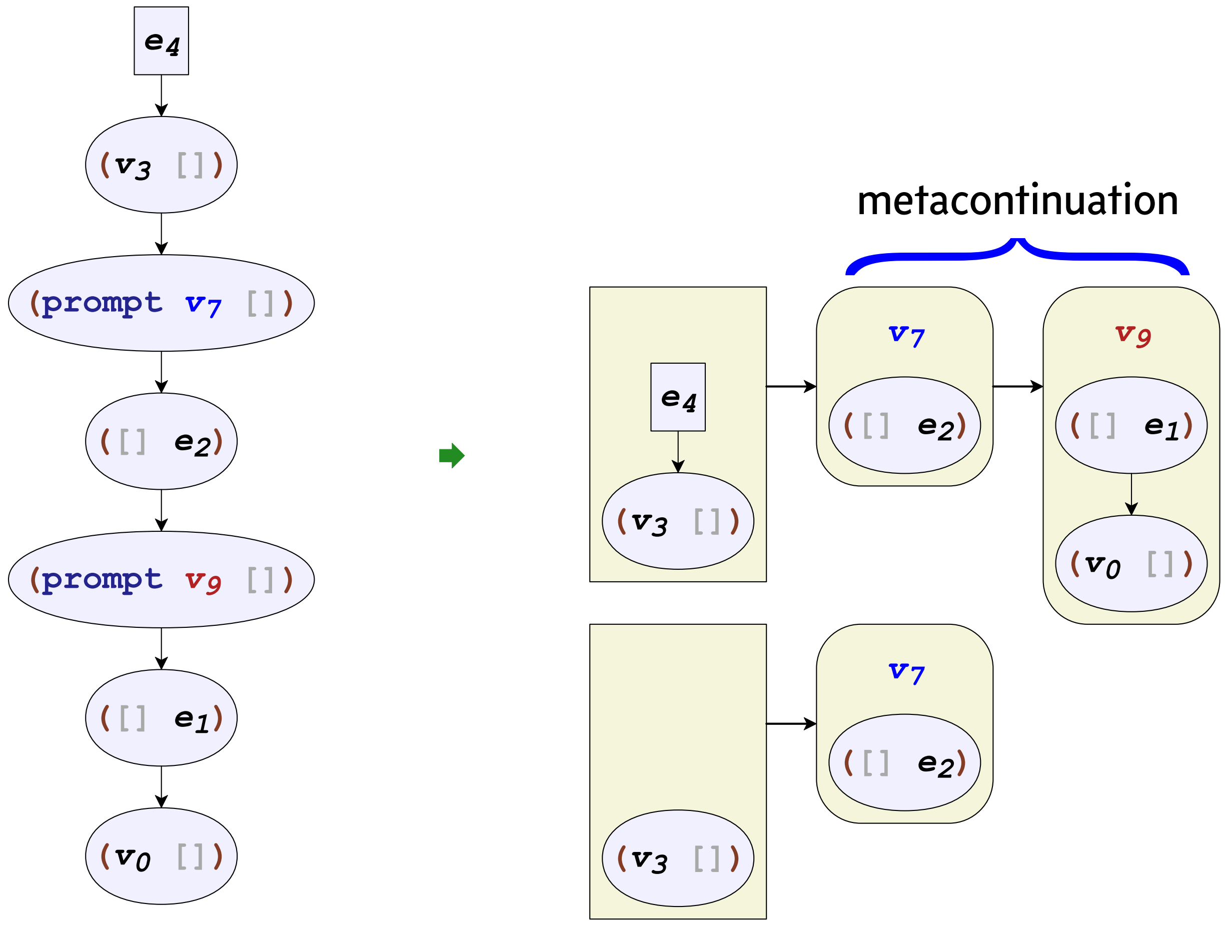
Delimited Continuations via Metacontinuations



Delimited Continuations via Metacontinuations



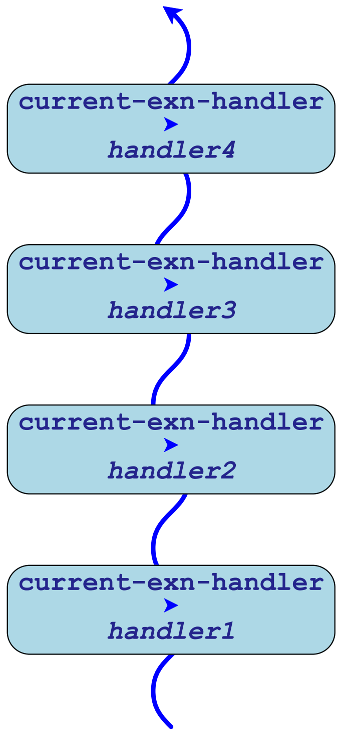
Delimited Continuations via Metacontinuations



Exceptions

`with-handlers` and `raise` implemented with

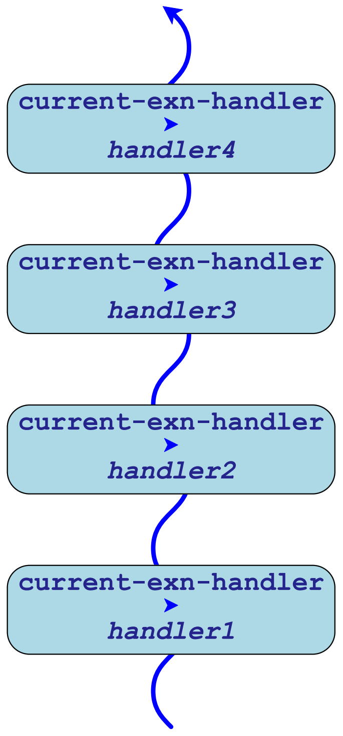
- `with-mark` to register a handler
- `prompt` to escape
- `current-marks` to find a handler



Exceptions

`with-handlers` and **`raise`** implemented with

- **`with-mark`** to register a handler
- **`prompt`** to escape
- ~~**`current-marks`**~~ **`current-marks-iterator`** to find a handler



current-exn-handler ➤ handler4
current-exn-handler ➤ handler3
current-exn-handler ➤ handler2
current-exn-handler ➤ handler1

Engines (proto-threads)

An *engine* runs a thunk for a while:

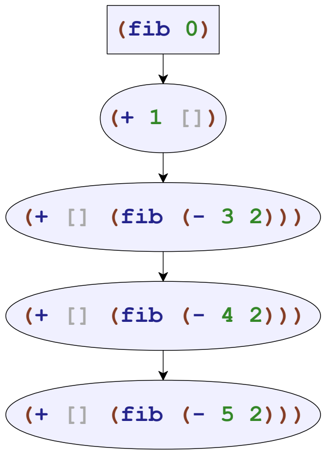
- Completes or suspends after a given timeout
- Can resume suspended with a new timeout

```
(fib 5)
```

Engines (proto-threads)

An *engine* runs a thunk for a while:

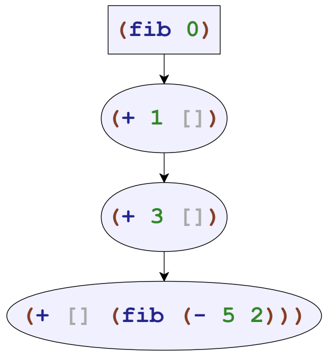
- Completes or suspends after a given timeout
- Can resume suspended with a new timeout



Engines (proto-threads)

An *engine* runs a thunk for a while:

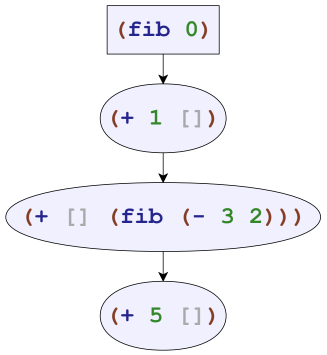
- Completes or suspends after a given timeout
- Can resume suspended with a new timeout



Engines (proto-threads)

An *engine* runs a thunk for a while:

- Completes or suspends after a given timeout
- Can resume suspended with a new timeout



Engines (proto-threads)

An *engine* runs a thunk for a while:

- Completes or suspends after a given timeout
- Can resume suspended with a new timeout

8

Engines (proto-threads)

`make-engine` implemented with

- **`start-timer`** to interrupt
- **`timer-interrupt-handler`** to handle interrupt
- **`call/cc`** to capture during interrupt
- **`call-in-continuation/no-wind`** to suspend/resume

`start-timer` and **`timer-interrupt-handler`** enabled by **`(check-event)`** calls injected by compiler

Threads (green)

`thread` and events implemented with

- **`make-engine`** to run threads
- **`with-cont-mark`** to enable/disable breaks (Ctl-C)

Initial **`prompt`** in a thread ensures portable continuations

Racket's Intermediate Language for Control

Racket

Continuations
function [tail] calls
`call/cc`
`call-with-composable-continuation`
continuation application
`call-in-continuation`

Prompts
`call-with-continuation-prompt`
`abort-current-continuation`
`continuation-prompt-available?`
`make-continuation-prompt-tag`
`default-continuation-prompt-tag`

Winding
`dynamic-wind`
`call-with-continuation-barrier`

Exceptions
`raise`
`call-with-exception-handler`

Continuation Marks
`with-continuation-mark`
`continuation-marks`
`current-continuation-marks`
`call-with-immediate-continuation-mark`
`continuation-mark-set->iterator`
`continuation-mark-set-first`

Dynamic Binding
`parameterize`
`make-parameter`
parameter application
`current-parameterization`

Threads (Green)
`thread`
`kill-thread`
`make-thread-group`

Thread-Local Storage
`make-thread-cell`
`thread-cell-ref`
`thread-cell-set!`
`current-preserved-thread-cell-values`

Asynchronous Exceptions
`break-thread`
`break-enabled`
`current-break-parameterization`
`call-with-break-parameterization`

Synchronization
`sync`
`make-channel`
`channel-put-evt`
`channel-get-evt`
`make-semaphore`
`semaphore-post`

Custodians
`make-custodian`
`custodian-shutdown-all`

Futures
`future`
`touch`

Chez Scheme

Continuations
function [tail] calls
`call/cc`
`call-in-continuation`
`null-continuation`

Continuation Marks
`current-continuation-attachments`
`continuation-next-attachments`
`call-getting-continuation-attachment`
`call-setting-continuation-attachment`
`call-consuming-continuation-attachment`

Exceptions
`raise`
`base-exception-handler`

Signals
`start-timer`
`timer-interrupt-handler`

Racket's Intermediate Language for Control

Racket

Continuations
function [tail] calls
`call/cc`
`call-with-composable-continuation`
continuation application
`call-in-continuation`

Prompts
`call-with-continuation-prompt`
`abort-current-continuation`
`continuation-prompt-available?`
`make-continuation-prompt-tag`
`default-continuation-prompt-tag`

Winding
`dynamic-wind`
`call-with-continuation-barrier`

Exceptions
`raise`
`call-with-exception-handler`

Continuation Marks
`with-continuation-mark`
`continuation-marks`
`current-continuation-marks`
`call-with-immediate-continuation-mark`
`continuation-mark-set->iterator`
`continuation-mark-set-first`

Dynamic Binding
`parameterize`
`make-parameter`
parameter application
`current-parameterization`

Threads (Green)
`thread`
`kill-thread`
`make-thread-group`

Thread-Local Storage
`make-thread-cell`
`thread-cell-ref`
`thread-cell-set!`
`current-preserved-thread-cell-values`

Asynchronous Exceptions
`break-thread`
`break-enabled`
`current-break-parameterization`
`call-with-break-parameterization`

Synchronization
`sync`
`make-channel`
`channel-put-evt`
`channel-get-evt`
`make-semaphore`
`semaphore-post`

Custodians
`make-custodian`
`custodian-shutdown-all`

Futures
`future`
`touch`

⬇

Chez Scheme

Continuations
function [tail] calls
`call/cc`
`call-in-continuation`
`null-continuation`

Continuation Marks
`current-continuation-attachments`
`continuation-next-attachments`
`call-getting-continuation-attachment`
`call-setting-continuation-attachment`
`call-consuming-continuation-attachment`

Exceptions
`raise`
`base-exception-handler`

Signals
`start-timer`
`timer-interrupt-handler`

Racket’s Intermediate Language for Control

Racket

Continuations
function [tail] calls
```
call/cc
call-with-composable-continuation
```
continuation application
```
call-in-continuation
```

Prompts
```
call-with-continuation-prompt
abort-current-continuation
continuation-prompt-available?
make-continuation-prompt-tag
default-continuation-prompt-tag
```

Winding
```
dynamic-wind
call-with-continuation-barrier
```

Exceptions
```
raise
call-with-exception-handler
```

Continuation Marks
```
with-continuation-mark
continuation-marks
current-continuation-marks
call-with-immediate-continuation-mark
continuation-mark-set->iterator
continuation-mark-set-first
```

Dynamic Binding
```
parameterize
make-parameter
```
parameter application
```
current-parameterization
```

Threads (Green)
```
thread
kill-thread
make-thread-group
```

Thread-Local Storage
```
make-thread-cell
thread-cell-ref
thread-cell-set!
current-preserved-thread-cell-values
```

Asynchronous Exceptions
```
break-thread
break-enabled
current-break-parameterization
call-with-break-parameterization
```

Synchronization
```
sync
make-channel
channel-put-evt
channel-get-evt
make-semaphore
semaphore-post
```

Custodians
```
make-custodian
custodian-shutdown-all
```

Futures
```
future
touch
```

rich set of constructs to support cooperating languages and libaries

Chez Scheme

Continuations
function [tail] calls
```
call/cc
call-in-continuation
null-continuation
```

Continuation Marks
```
current-continuation-attachments
continuation-next-attachments
call-getting-continuation-attachment
call-setting-continuation-attachment
call-consuming-continuation-attachment
```

Exceptions
```
raise
base-exception-handler
```

Signals
```
start-timer
timer-interrupt-handler
```

Racket’s Intermediate Language for Control

Racket

Continuations
function [tail] calls
`call/cc`
`call-with-composable-continuation`
continuation application
`call-in-continuation`

Prompts
`call-with-continuation-prompt`
`abort-current-continuation`
`continuation-prompt-available?`
`make-continuation-prompt-tag`
`default-continuation-prompt-tag`

Winding
`dynamic-wind`
`call-with-continuation-barrier`

Exceptions
`raise`
`call-with-exception-handler`

Continuation Marks
`with-continuation-mark`
`continuation-marks`
`current-continuation-marks`
`call-with-immediate-continuation-mark`
`continuation-mark-set->iterator`
`continuation-mark-set-first`

Dynamic Binding
`parameterize`
`make-parameter`
parameter application
`current-parameterization`

Threads (Green)
`thread`
`kill-thread`
`make-thread-group`

Thread-Local Storage
`make-thread-cell`
`thread-cell-ref`
`thread-cell-set!`
`current-preserved-thread-cell-values`

Asynchronous Exceptions
`break-thread`
`break-enabled`
`current-break-parameterization`
`call-with-break-parameterization`

Synchronization
`sync`
`make-channel`
`channel-put-evt`
`channel-get-evt`
`make-semaphore`
`semaphore-post`

Custodians
`make-custodian`
`custodian-shutdown-all`

Futures
`future`
`touch`

rich set of constructs to support cooperating languages and libaries

Chez Scheme

Continuations
function [tail] calls
`call/cc`
`call-in-continuation`
`null-continuation`

Continuation Marks
`current-continuation-attachments`
`continuation-next-attachments`
`call-getting-continuation-attachment`
`call-setting-continuation-attachment`
`call-consuming-continuation-attachment`

Exceptions
`raise`
`base-exception-ha`

Signals
`start-timer`
`timer-interrupt-handler`

efficient implementation

Racket's Intermediate Language for Control

Racket

Continuations
function [tail] calls
`call/cc`
`call-with-composable-continuation`
continuation application
`call-in-continuation`

Prompts
`call-with-continuation-prompt`
`abort-current-continuation`
`continuation-prompt-available?`
`make-continuation-prompt-tag`
`default-continuation-prompt-tag`

Winding
`dynamic-wind`
`call-with-continuation-barrier`

Exceptions
`raise`
`call-with-exception-handler`

Continuation Marks
`with-continuation-mark`
`continuation-marks`
`current-continuation-marks`
`call-with-immediate-continuation-mark`
`continuation-mark-set->iterator`
`continuation-mark-set-first`

Dynamic Binding
`parameterize`
`make-parameter`
parameter application
`current-parameterization`

Threads (Green)
`thread`
`kill-thread`
`make-thread-group`

Thread-Local Storage
`make-thread-cell`
`thread-cell-ref`
`thread-cell-set!`
`current-preserved-thread-cell-values`

Asynchronous Exceptions
`break-thread`
`break-enabled`
`current-break-parameterization`
`call-with-break-parameterization`

Synchronization
`sync`
`make-channel`
`channel-put-evt`
`channel-get-evt`
`make-semaphore`
`semaphore-post`

Custodians
`make-custodian`
`custodian-shutdown-all`

Futures
`future`
`touch`

rich set of constructs to support cooperating languages and libaries

Chez Scheme

Continuations
function [tail] calls
`call/cc`
`call-in-continuation`
`null-continuation`

Continuation Marks
`current-continuation-attachments`
`continuation-next-attachments`
`call-getting-continuation-attachment`
`call-setting-continuation-attachment`
`call-consuming-continuation-attachment`

Exceptions
`raise`
`base-exception-ha`

Signals
`start-timer`
`timer-interrupt-handler`

efficient implementation