

```
(call/cc (lambda (k)
          (set! start-k k)))
```

# Adding Delimited and Composible Control to a Production Programming Environment



**Matthew Flatt** University of Utah

**Gang Yu** Institute of Software, Chinese Academy of Sciences

**Robert Bruce Findler** University of Chicago

**Matthias Felleisen** Northeastern University



Name

**a programming language**

with

first-class continuations...

**(start-k)**

# Adding Delimited and Composible Control to a Production Programming Environment



**Matthew Flatt** University of Utah

**Gang Yu** Institute of Software, Chinese Academy of Sciences

**Robert Bruce Findler** University of Chicago

**Matthias Felleisen** Northeastern University



Name

**a programming language**

with

**first-class continuations...**



Name

**a programming language**

with

first-class continuations...

**Scheme**



Name

**an application**

with

**first-class continuations...**



Name

**an application**

with

first-class continuations...

**web servers**

# Web Servlet with Continuations

```
(define (paper-search-servlet)
  (let ([terms (get-search-terms)])
    (find-paper terms)))
```

# Web Servlet with Continuations

send back HTML form, then  
wait for answer as new request

```
(define (paper-search-servlet)
  (let ([terms (get-search-terms)])
    (find-paper terms)))
```

# Implementing a Web Server

```
(serve out1 in1)
```

# Implementing a Web Server

```
(reply out1 (generate-html in1))
```

# Implementing a Web Server

`(reply out1 [])`

```
graph TD; A("reply out1 []") --> B[generate-html in1];
```

`(generate-html in1)`

# Implementing a Web Server

`(reply out1 [])`

The diagram consists of an oval at the top containing the code `(reply out1 [])`. A downward-pointing arrow connects the bottom of this oval to a rectangular box below it. The box contains the code `(call/cc (λ (esc) (dispatch in1)))`. This illustrates the implementation of the `reply` function using `call/cc`.

```
(call/cc  
  (λ (esc) (dispatch in1)))
```

# Implementing a Web Server

`esc = (reply out1 [])`

`(reply out1 [])`

`(call/cc  
 (λ (esc) (dispatch in1)))`

# Implementing a Web Server

`esc = (reply out1 [])`

`(reply out1 [])`



`(dispatch in1)`

# Implementing a Web Server

esc = (reply out<sub>1</sub> [])

(reply out<sub>1</sub> [])



```
(let ([terms (get-search-terms)])  
  (find-paper terms))
```

# Implementing a Web Server

esc = (reply out<sub>1</sub> [])

(reply out<sub>1</sub> [])

(let ([terms []])  
 (find-paper terms))

(get-search-terms)

# Implementing a Web Server

esc = (reply out<sub>1</sub> [])

(reply out<sub>1</sub> [])

(let ([terms []])  
 (find-paper terms))

```
(call/cc  
  (λ (web-k)  
    (esc (build-form (cont->url web-k))))))
```

# Implementing a Web Server

esc = (reply out<sub>1</sub> [])

web-k = (reply out<sub>1</sub> [])

(reply out<sub>1</sub> [])

(let ([terms []])  
 (find-paper terms))

(let ([terms []])  
 (find-paper terms))

```
(call/cc  
  (λ (web-k)  
    (esc (build-form (cont->url web-k))))))
```

# Implementing a Web Server

esc = (reply out<sub>1</sub> [])

web-k = (reply out<sub>1</sub> [])

(reply out<sub>1</sub> [])

(let ([terms []])  
 (find-paper terms))

(let ([terms []])  
 (find-paper terms))

(esc (build-form (cont->url web-k)))

# Implementing a Web Server

esc = (reply out<sub>1</sub> [])

web-k = (reply out<sub>1</sub> [])

(reply out<sub>1</sub> [])

(let ([terms []])  
 (find-paper terms))

(let ([terms []])  
 (find-paper terms))

(esc <html/>)

# Implementing a Web Server

esc = (reply out<sub>1</sub> [])

(reply out<sub>1</sub> <html/>)

web-k = (reply out<sub>1</sub> [])

(let ([terms []])  
 (find-paper terms))

# Implementing a Web Server

esc = (reply out<sub>1</sub> [])

(reply out<sub>2</sub> [])

(dispatch in<sub>2</sub>)

web-k = (reply out<sub>1</sub> [])

(let ([terms []])  
 (find-paper terms))

# Implementing a Web Server

`esc = (reply out1 [])`

`(reply out2 [])`

`(web-k form-answers)`

`web-k = (reply out1 [])`

`(let ([terms []])  
 (find-paper terms))`

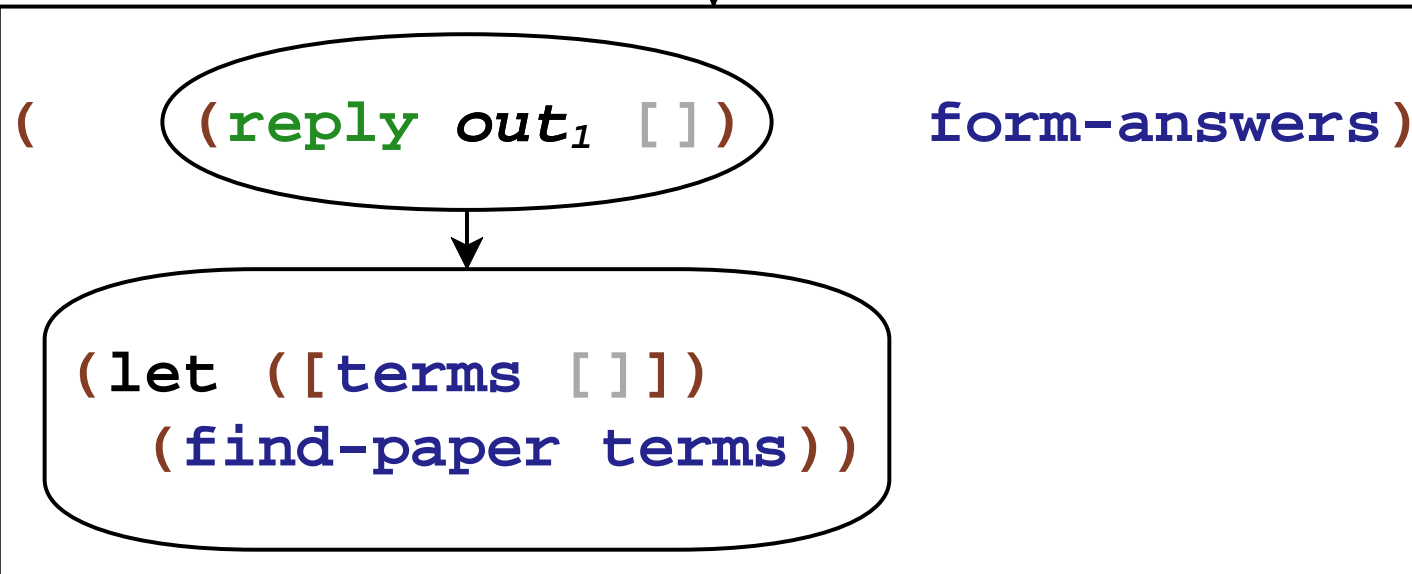
# Implementing a Web Server

esc = (reply out<sub>1</sub> [])

web-k = (reply out<sub>1</sub> [])

(reply out<sub>2</sub> [])

(let ([terms []])  
 (find-paper terms))



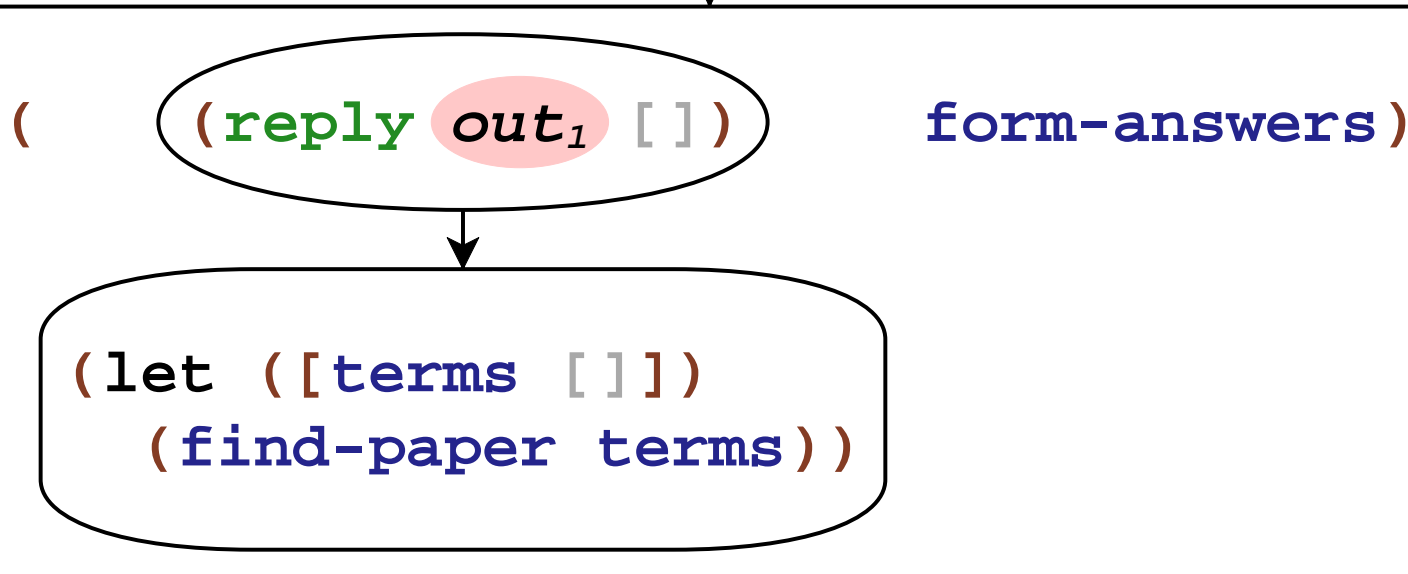
# Implementing a Web Server

esc = (reply out<sub>1</sub> [])

web-k = (reply out<sub>1</sub> [])

(reply out<sub>2</sub> [])

(let ([terms []])  
 (find-paper terms))



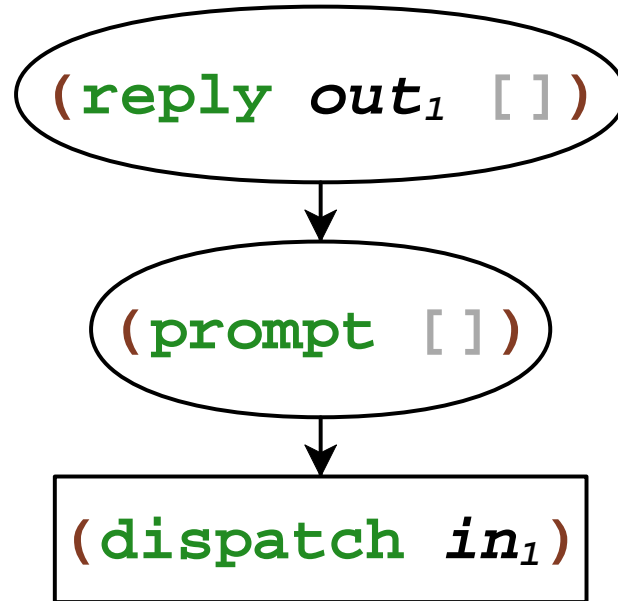
# Web Server with Prompt

`(reply out1 [])`

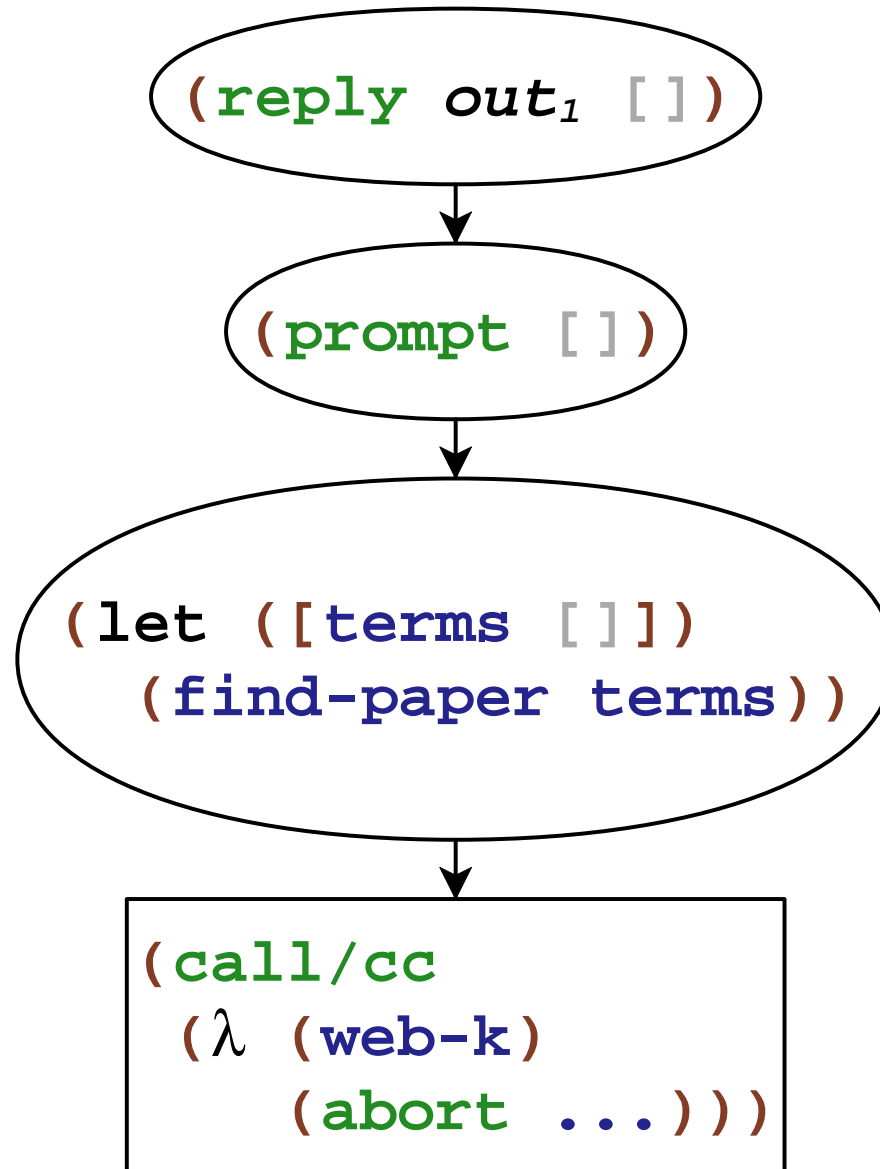


`(prompt (dispatch in1))`

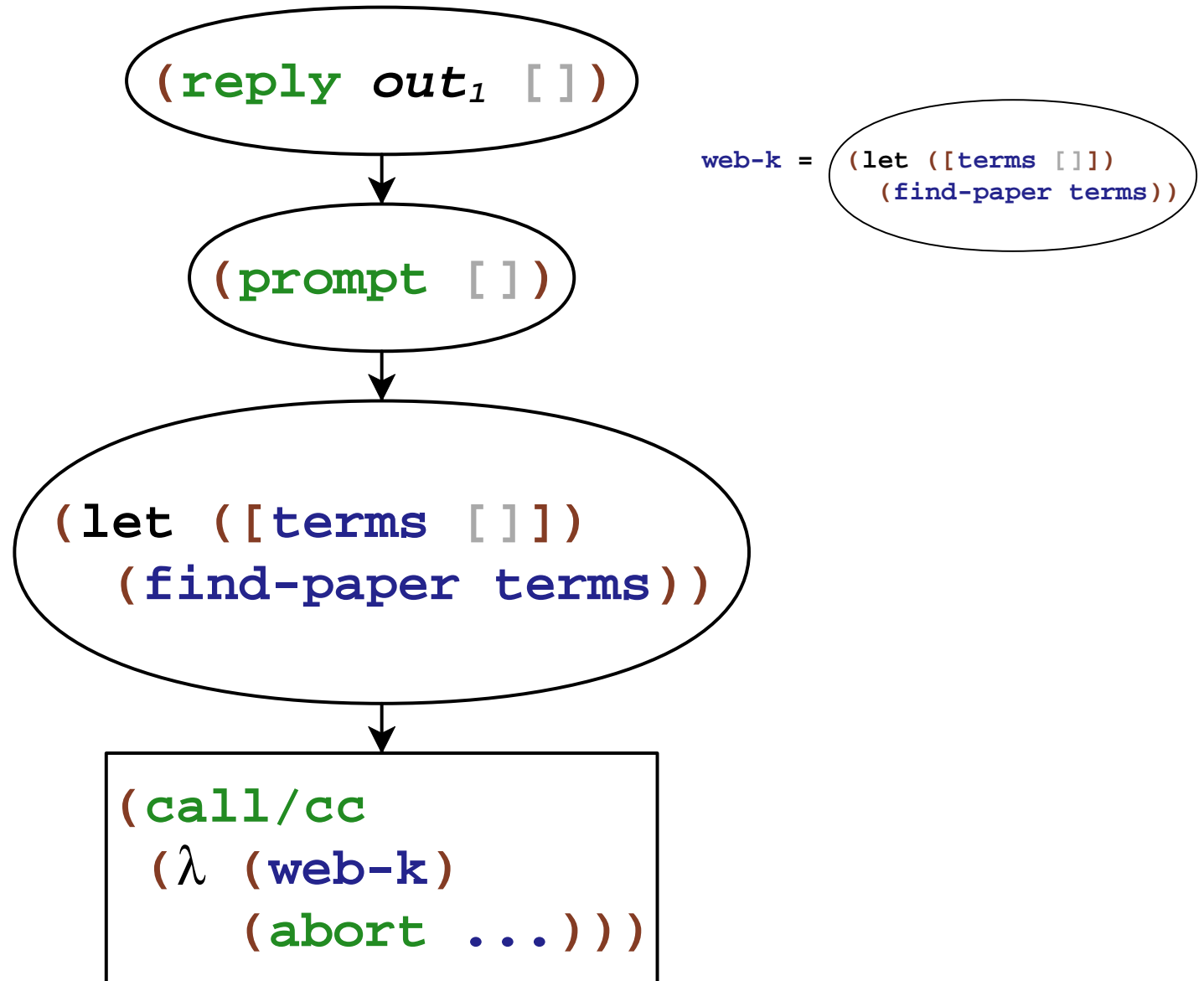
# Web Server with Prompt



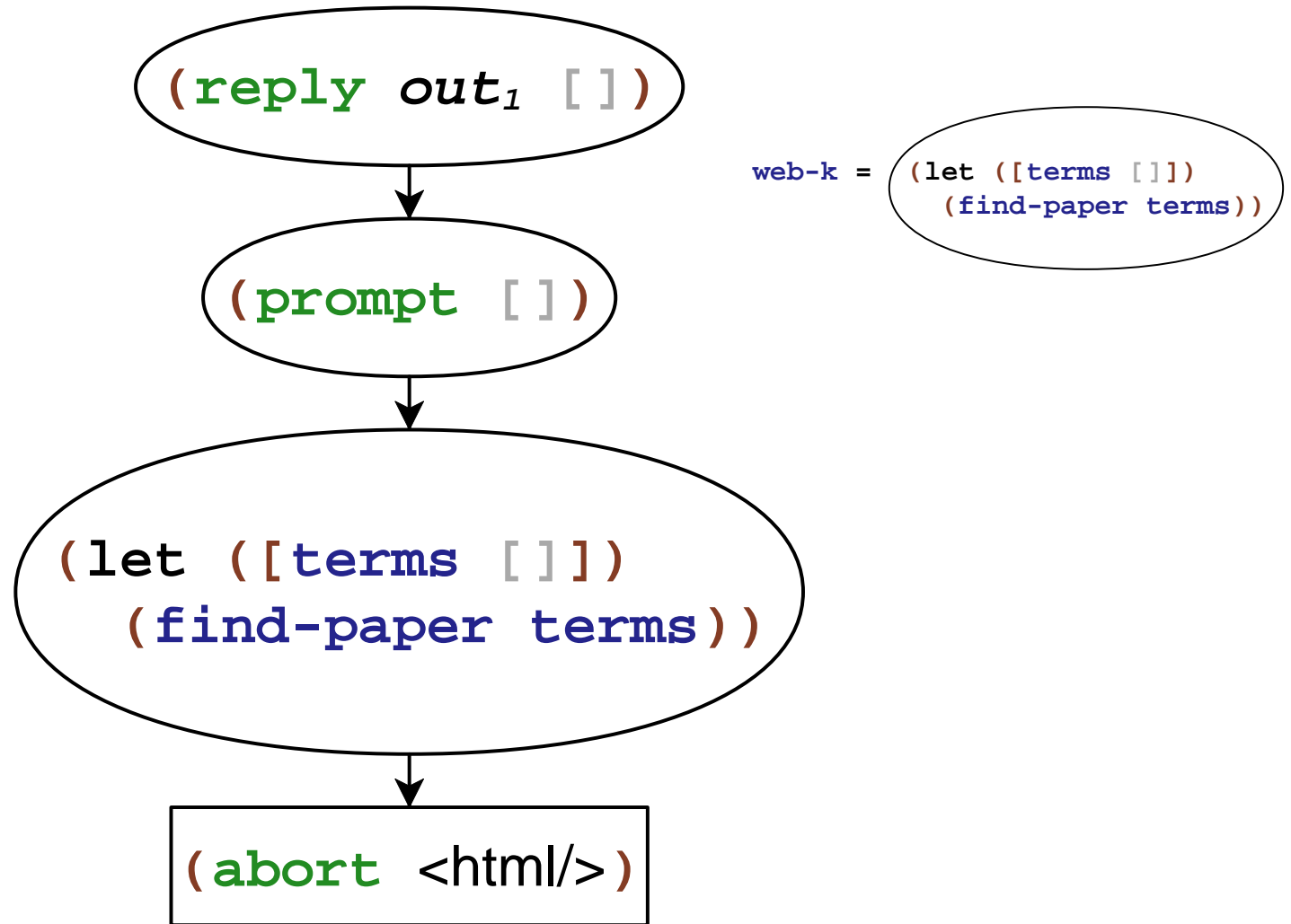
# Web Server with Prompt



# Web Server with Prompt



# Web Server with Prompt

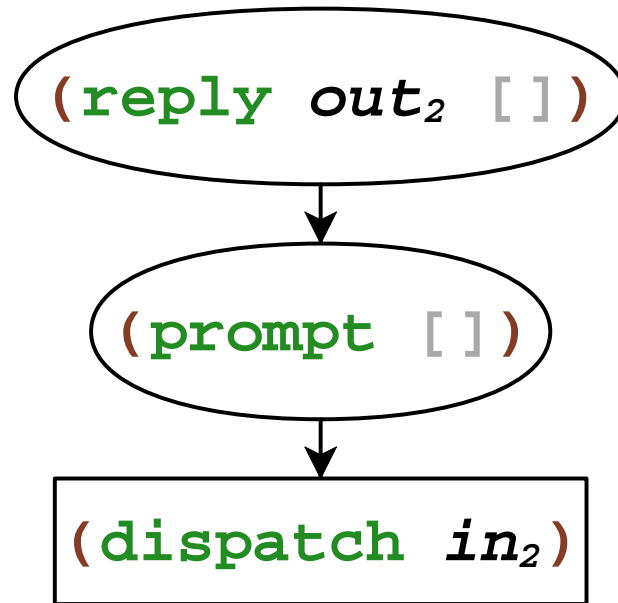


# Web Server with Prompt

```
(reply out1 <html/>)
```

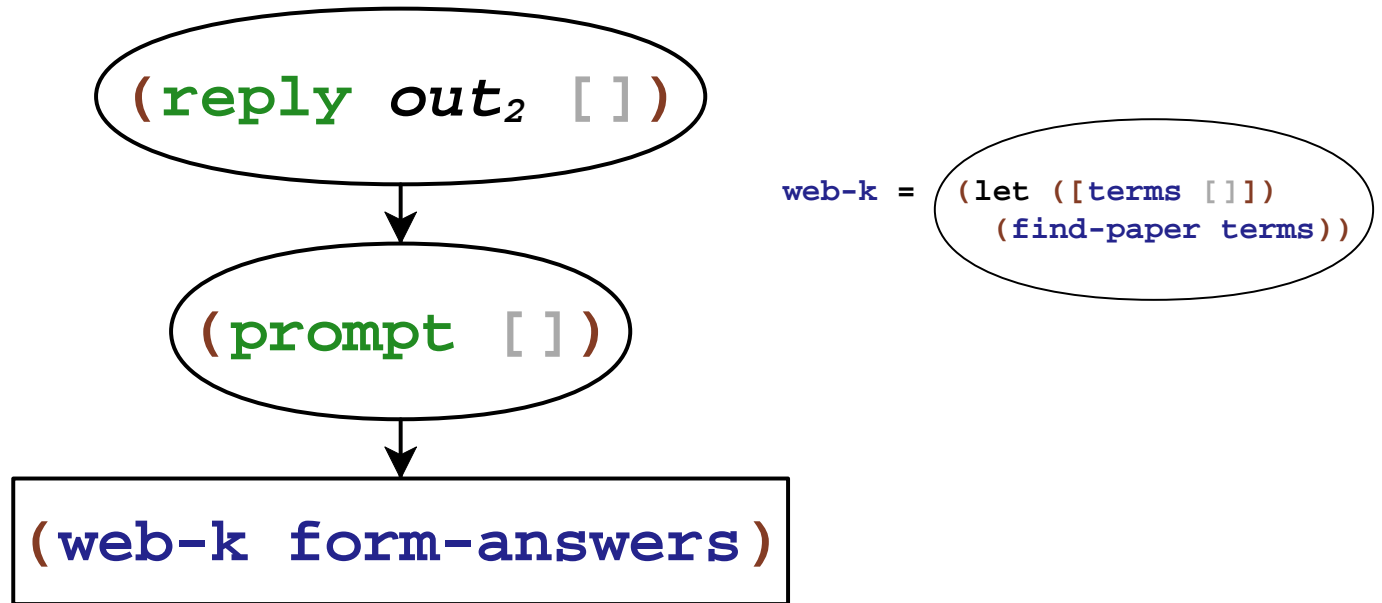
```
web-k = (let ([terms []])  
         (find-paper terms))
```

# Web Server with Prompt

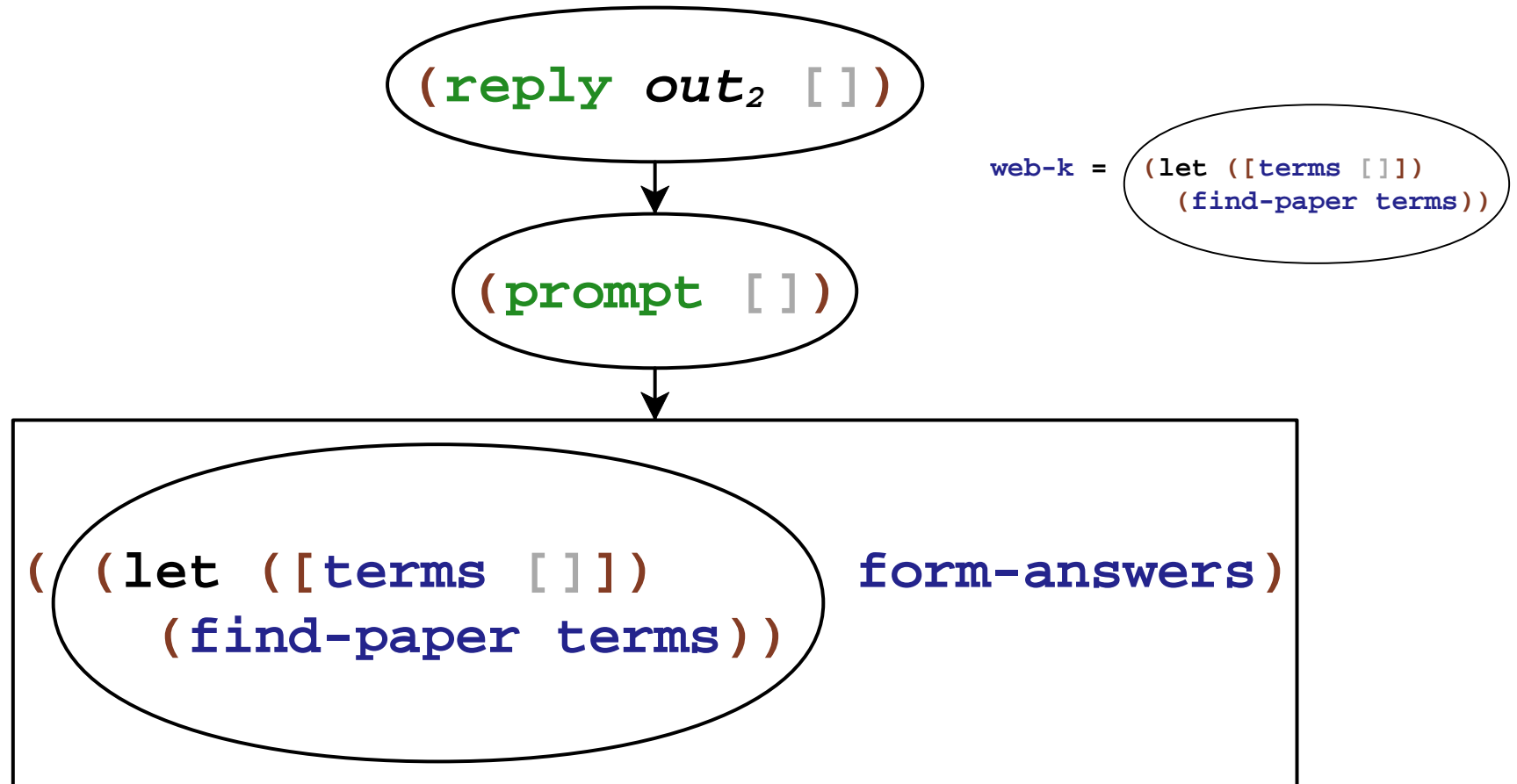


web-k = (`let` ([`terms` []])  
(`find-paper terms`))

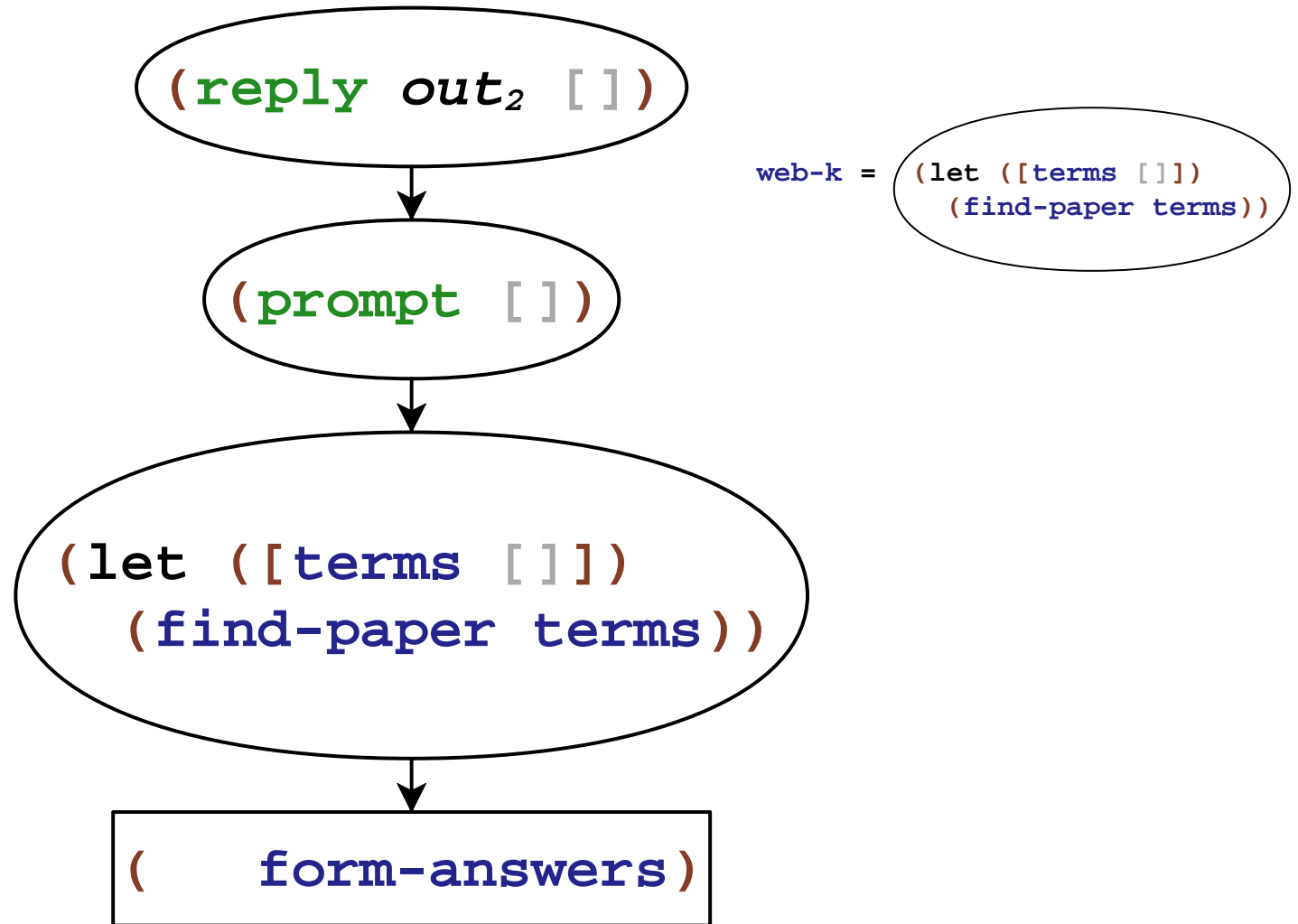
# Web Server with Prompt



# Web Server with Prompt



# Web Server with Prompt



# Web Server with Prompt

`(reply out2 [])`



`(prompt [])`



```
(let ([terms form-answers])  
  (find-paper terms))
```

web-k = `(let ([terms []])  
 (find-paper terms))`

# Rolling Your Own vs. Language Extension

Delimited control via `call/cc` doesn't work right with

- exceptions
- dynamic binding
- `dynamic-wind`

**Our goal:** delimited control integrated with existing constructs

## Papers on Delimited Continuations

Felleisen 88

Hieb and Dybvig 90

Queinnec and Serpette 91

Queinnec 93

Wadler 94

Rehof and Sørensen 94

Gunter et al. 95

Rehof 01

Kameyama and Hasegawa 03

Shan 04

Kiselyov et al. 06

Biernacki et al. 06

Danvy and Filinski 90

Sitaram and Felleisen 90

Sitaram 93

Moreau and Queinnec 94

deGroote 94

Gunter et al. 95

Thielecke 97

Gasbichler and Sperber 02

Ariola et al. 04

Saurin 05

Dybvig et al. 06

# Papers on Design

Felleisen 88

Danvy and Filinski 90

Hieb and Dybvig 90

Sitaram and Felleisen 90

Queinnec and Serpette 91

Sitaram 93

Moreau and Queinnec 94

Gunter et al. 95

Kiselyov et al. 06

Dybvig et al. 06

# Papers on Implementation

Gasbichler and Sperber 02

Dybvig et al. 06

# Implementations

# Implementations (Now)



# Balance

exceptions

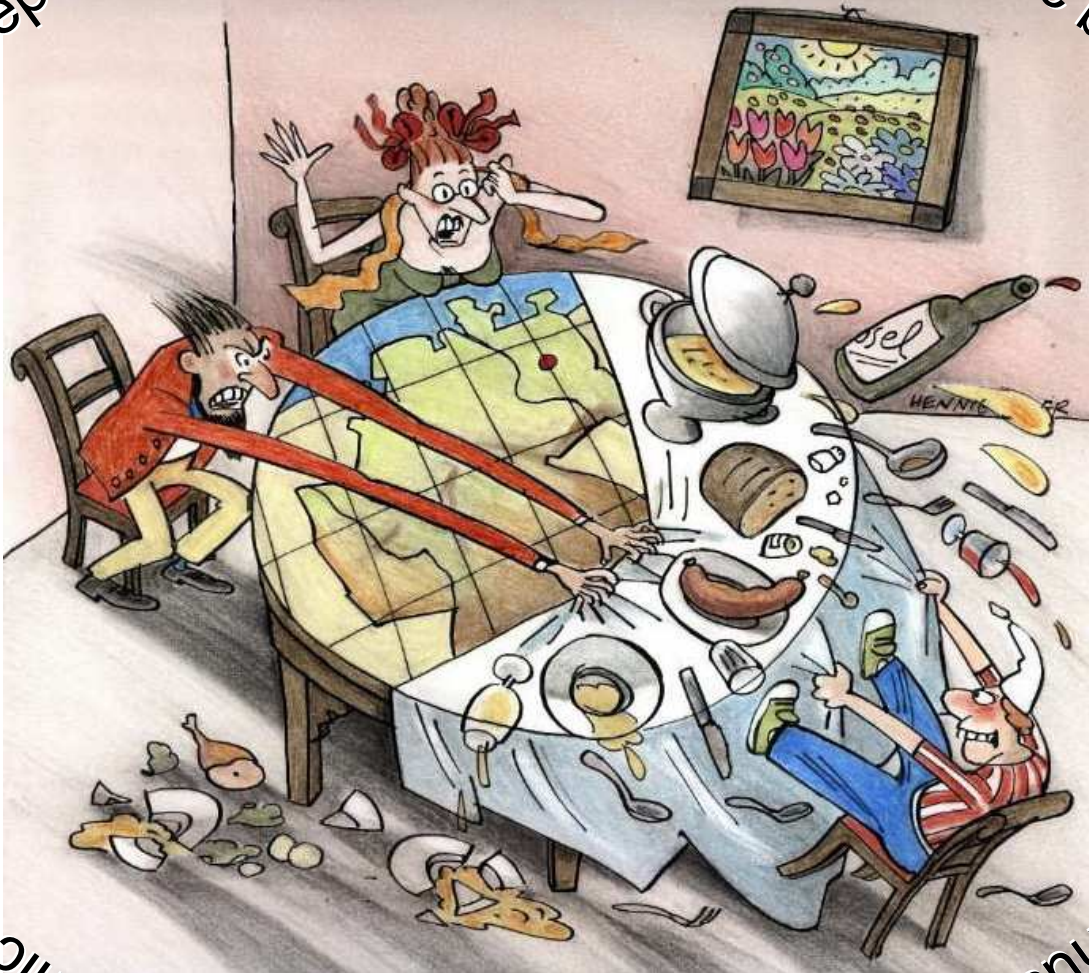
dynamic binding

prompts

abort

dynamic-wind

continuations



# Contributions

- Comprehensive design
- Formal model
- Implementation

# Contributions

- Comprehensive design

  
*graphical intuition*  


- Formal model

- Implementation

**(start-k)**

# Adding Delimited and Composible Control to a Production Programming Environment



**Matthew Flatt** University of Utah

**Gang Yu** Institute of Software, Chinese Academy of Sciences

**Robert Bruce Findler** University of Chicago

**Matthias Felleisen** Northeastern University

# Notation

$$(\mathbf{v}_1 \ ((\lambda \ (\mathbf{x}) \ \mathbf{x}) \ \mathbf{v}_3) \ \mathbf{v}_2))$$

# Notation

$$(\mathbf{v}_1 \ ((\lambda \ (\mathbf{x}) \ \mathbf{x}) \ \mathbf{v}_3) \ \mathbf{v}_2))$$

# Notation

$(\mathbf{v}_1 ((\lambda (\mathbf{x}) \mathbf{x}) \mathbf{v}_3) \mathbf{v}_2))$

$((\lambda (\mathbf{x}) \mathbf{x}) \mathbf{v}_3)$

# Notation

$(\mathbf{v}_1 \ ((\lambda \ (\mathbf{x}) \ \mathbf{x}) \ \mathbf{v}_3) \ \mathbf{v}_2))$

$((\lambda \ (\mathbf{x}) \ \mathbf{x}) \ \mathbf{v}_3)$

# Notation

$(\mathbf{v}_1 ((\lambda (\mathbf{x}) \mathbf{x}) \mathbf{v}_3) \mathbf{v}_2))$

$([\ ] \mathbf{v}_2)$

$((\lambda (\mathbf{x}) \mathbf{x}) \mathbf{v}_3)$

# Notation

$(\mathbf{v}_1 ((\lambda (\mathbf{x}) \mathbf{x}) \mathbf{v}_3) \mathbf{v}_2))$

$([\ ] \mathbf{v}_2)$

$((\lambda (\mathbf{x}) \mathbf{x}) \mathbf{v}_3)$

# Notation

$(\mathbf{v}_1 ((\lambda (\mathbf{x}) \mathbf{x}) \mathbf{v}_3) \mathbf{v}_2))$

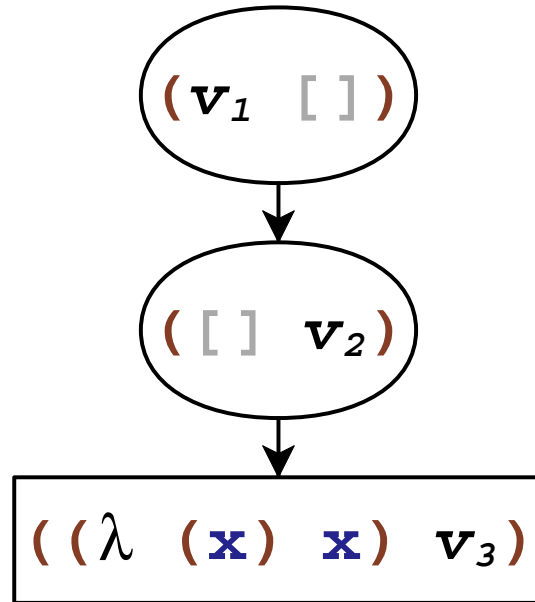
$(\mathbf{v}_1 [])$

$([] \mathbf{v}_2)$

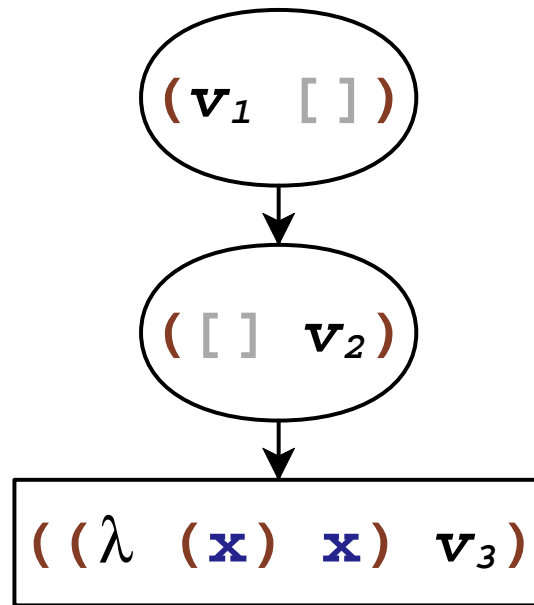
$((\lambda (\mathbf{x}) \mathbf{x}) \mathbf{v}_3)$

# Notation

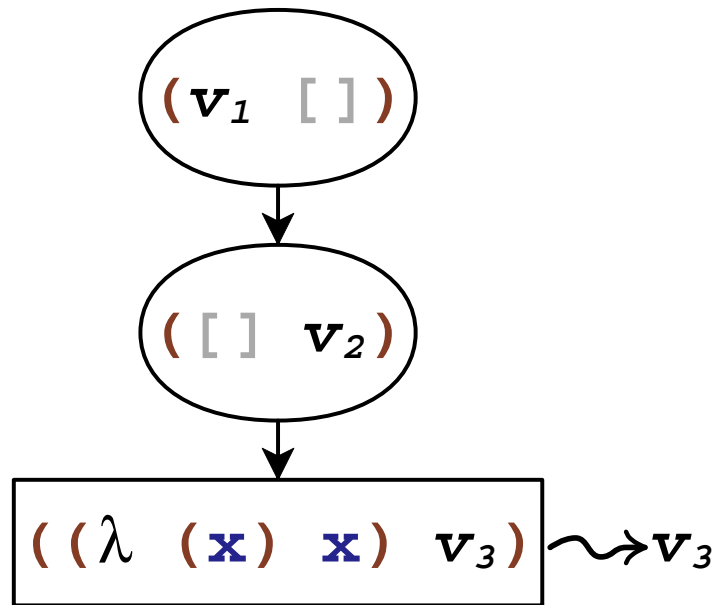
$$(\mathbf{v}_1 \ ((\lambda \ (\mathbf{x}) \ \mathbf{x}) \ \mathbf{v}_3) \ \mathbf{v}_2)) =$$



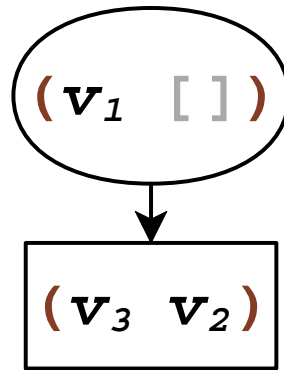
# Reductions



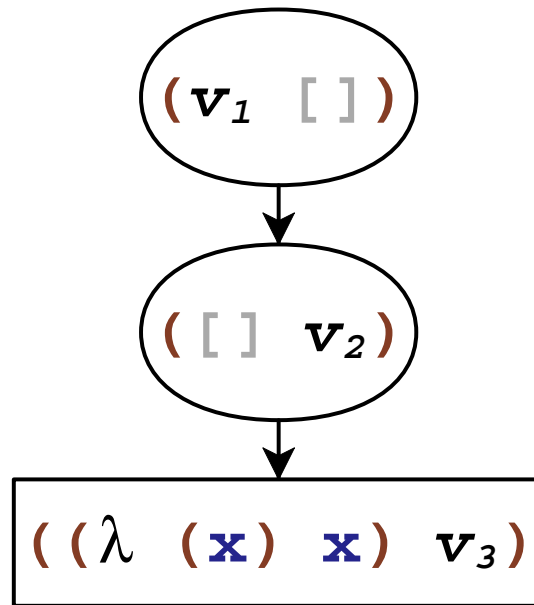
# Reductions



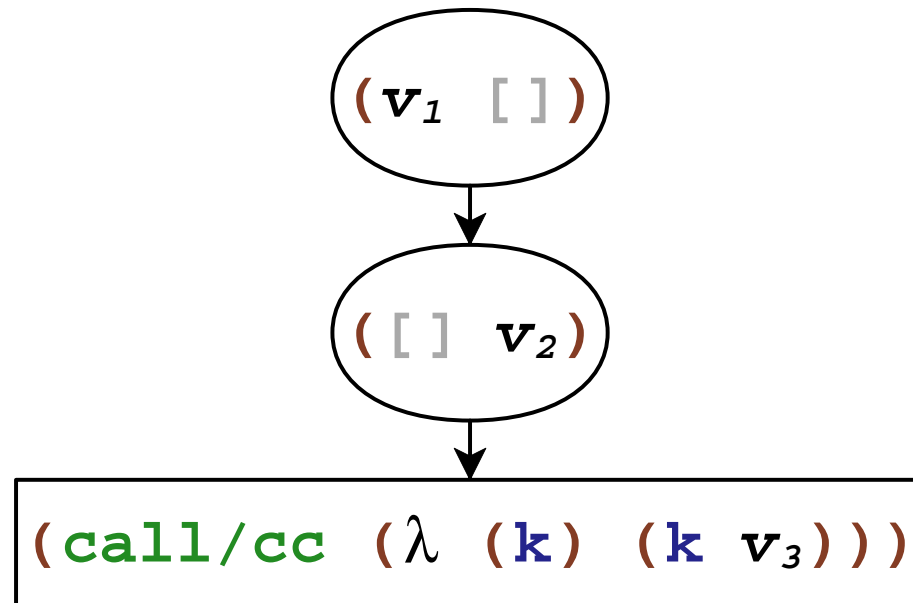
# Reductions



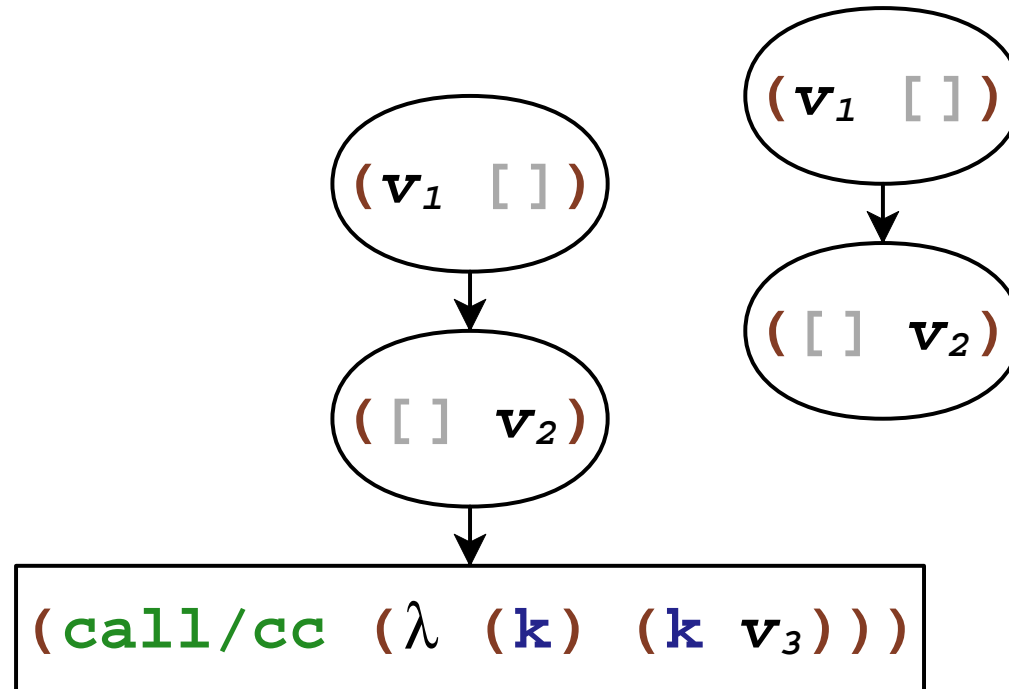
# Reductions



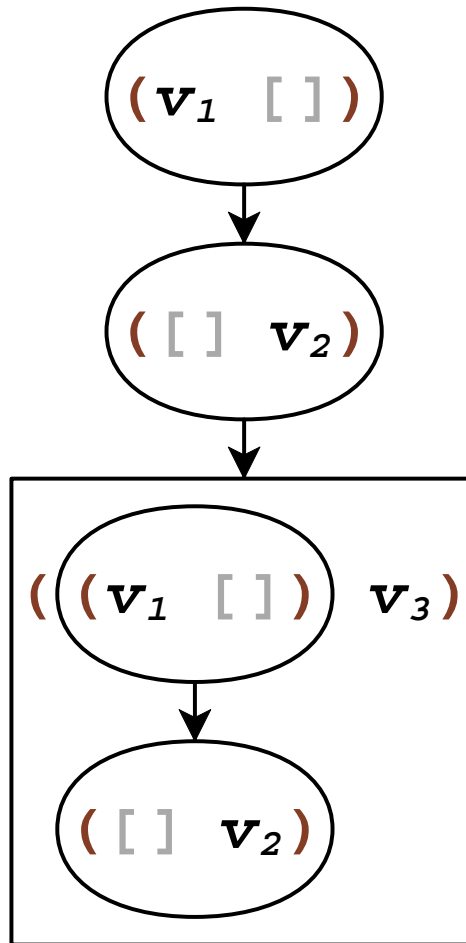
# Continuations



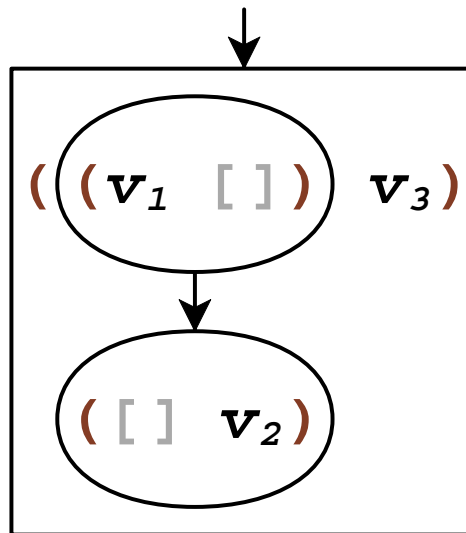
# Continuations



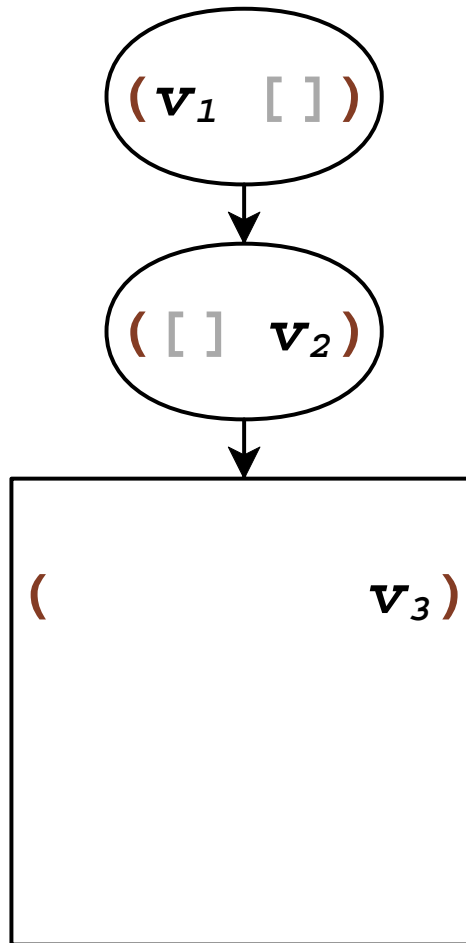
# Continuations



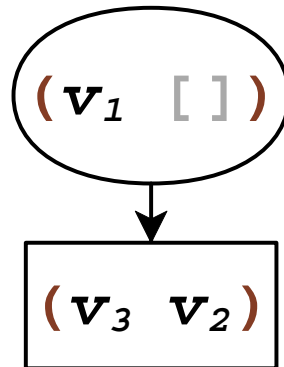
# Continuations



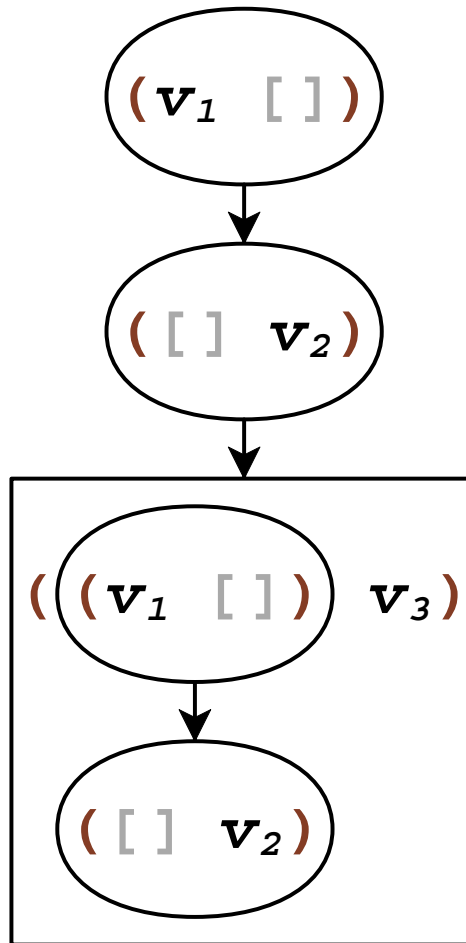
# Continuations



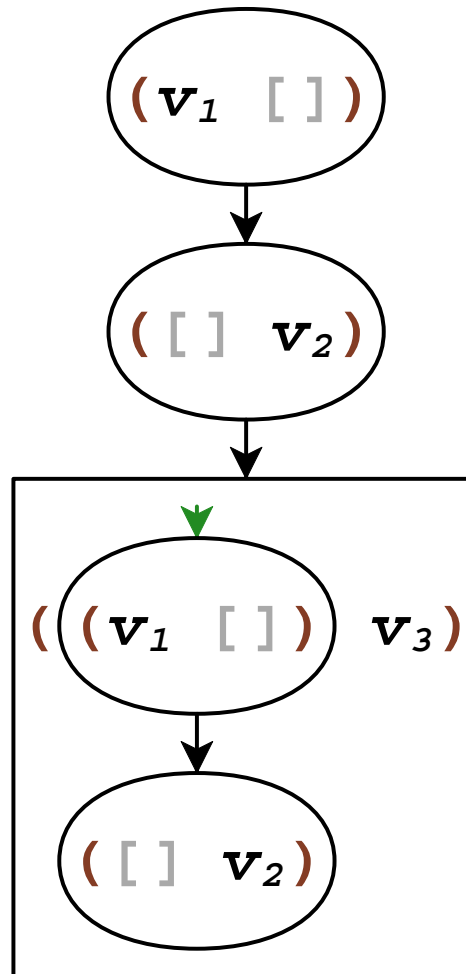
# Continuations



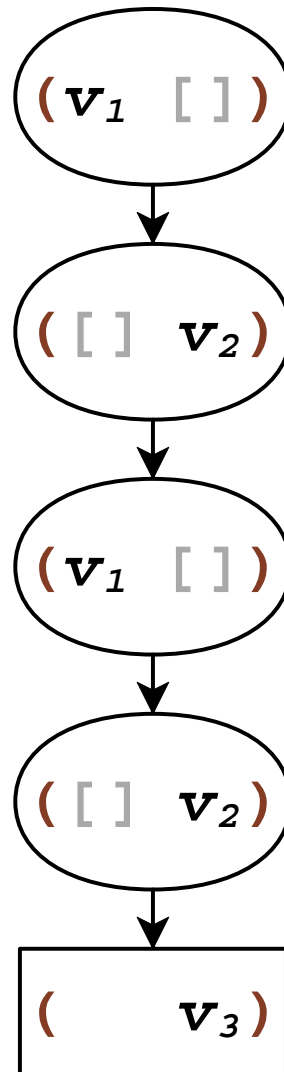
# Continuations



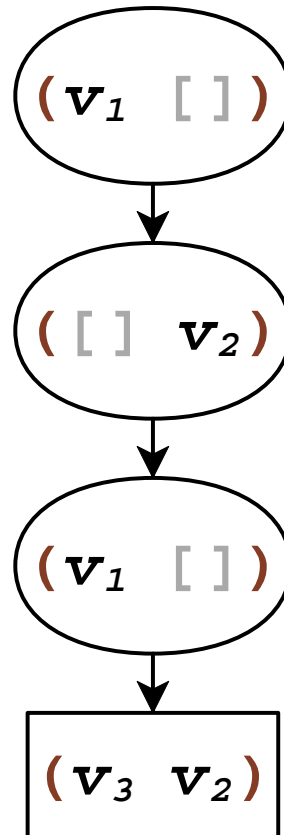
# Composable Continuations



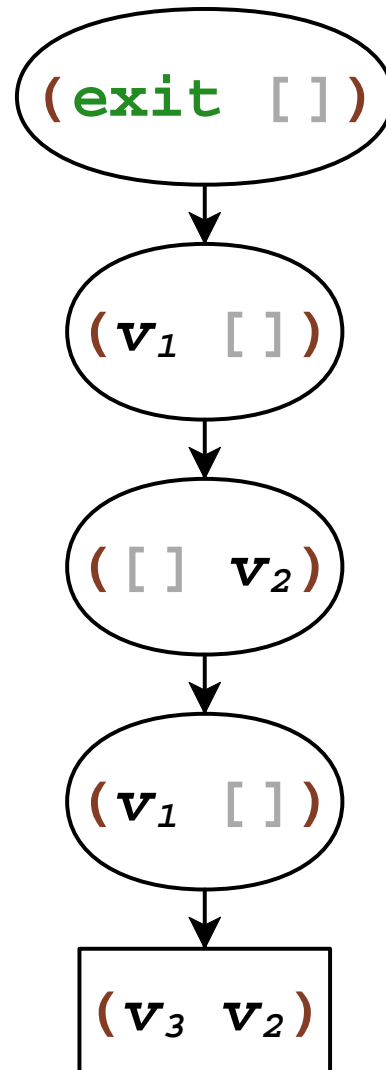
# Composable Continuations



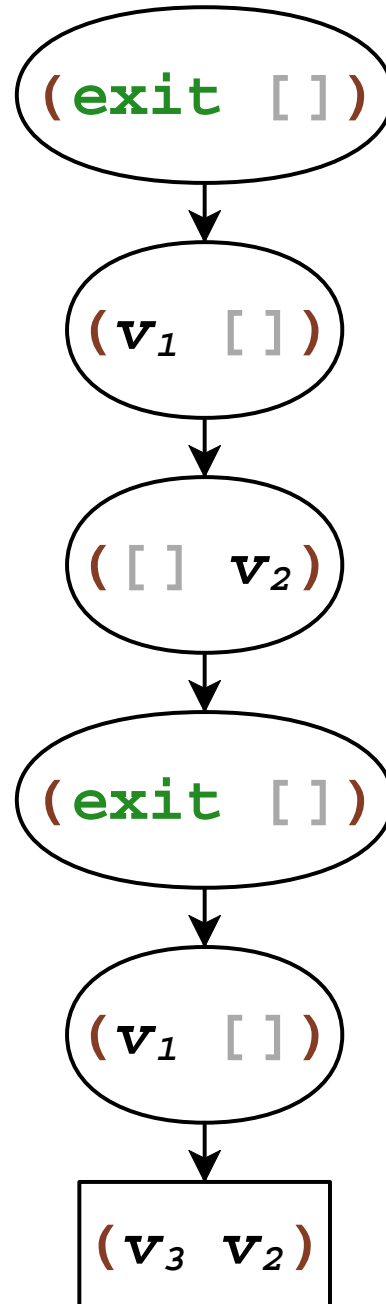
# Composable Continuations



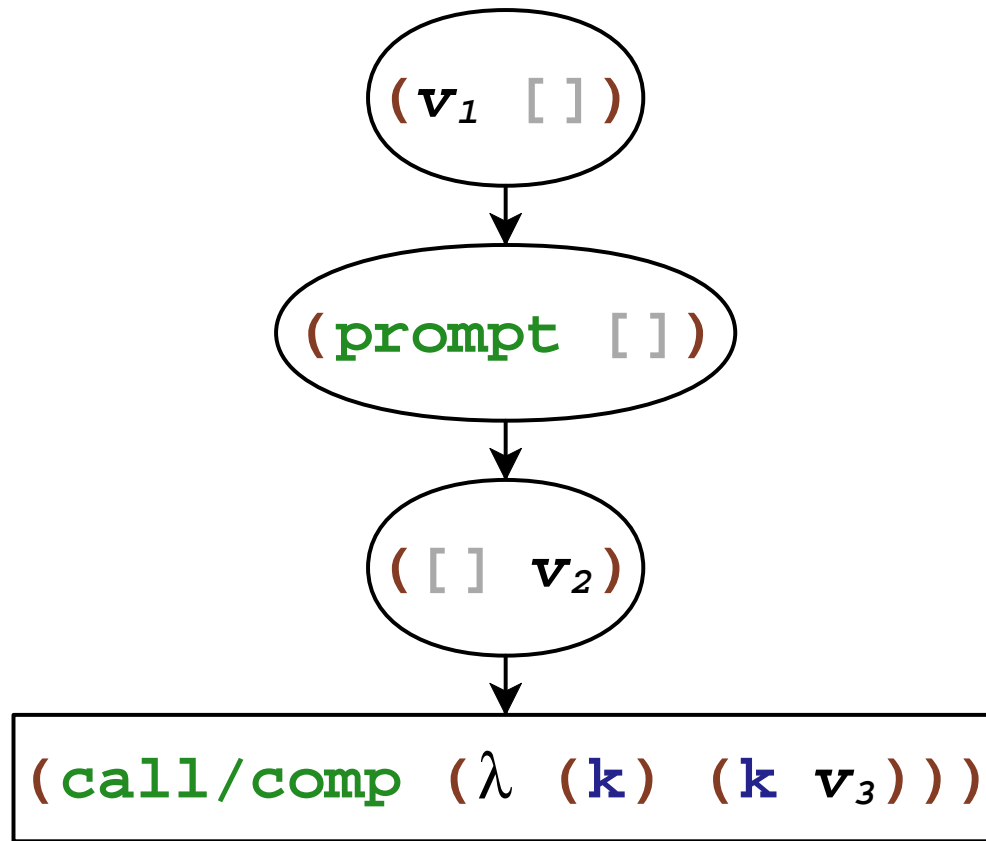
# Composable Continuations



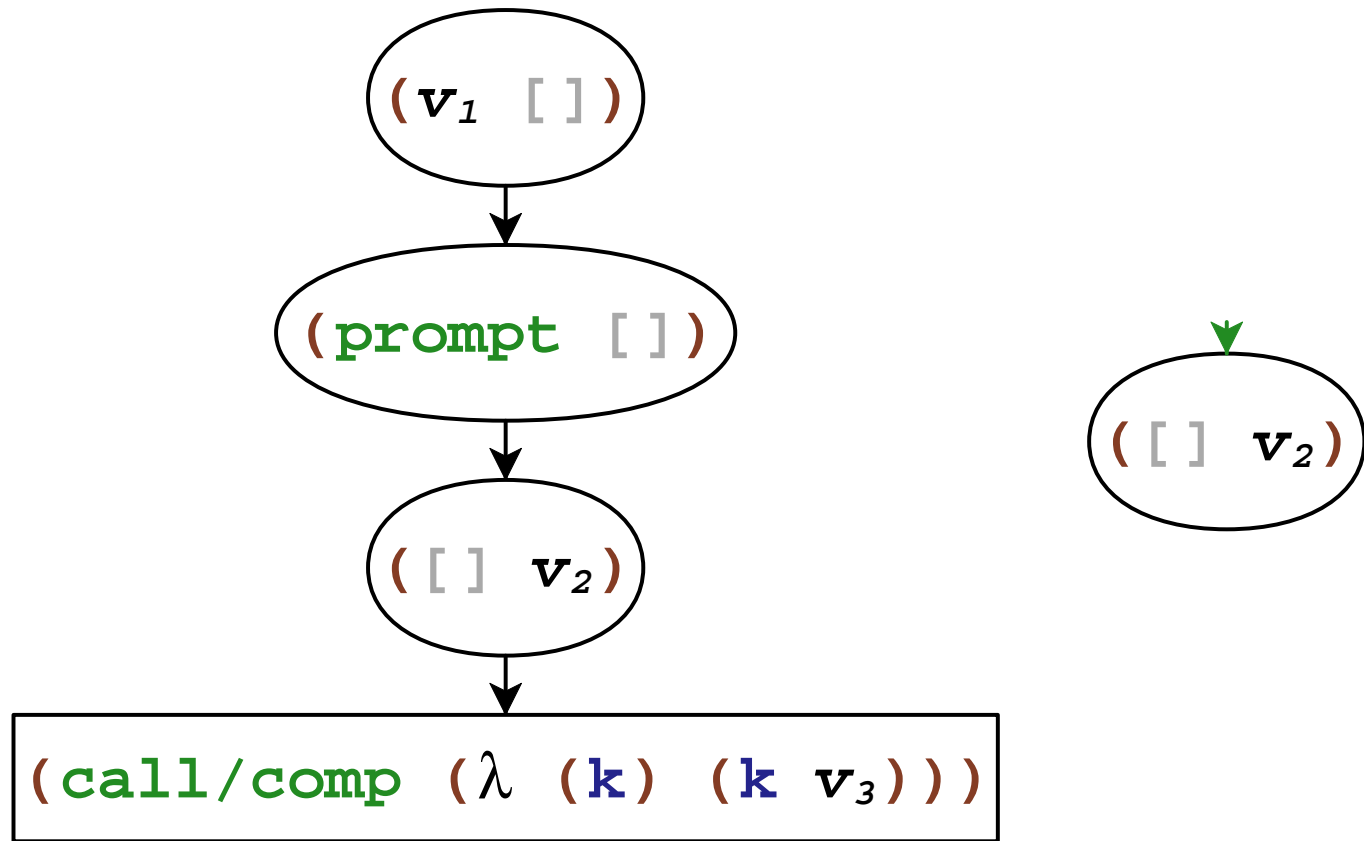
# Composable Continuations



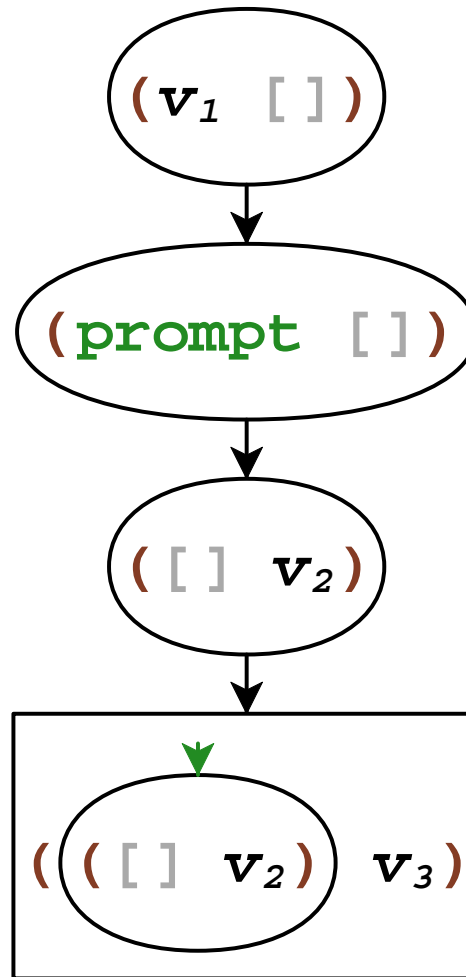
# Delimited Capture



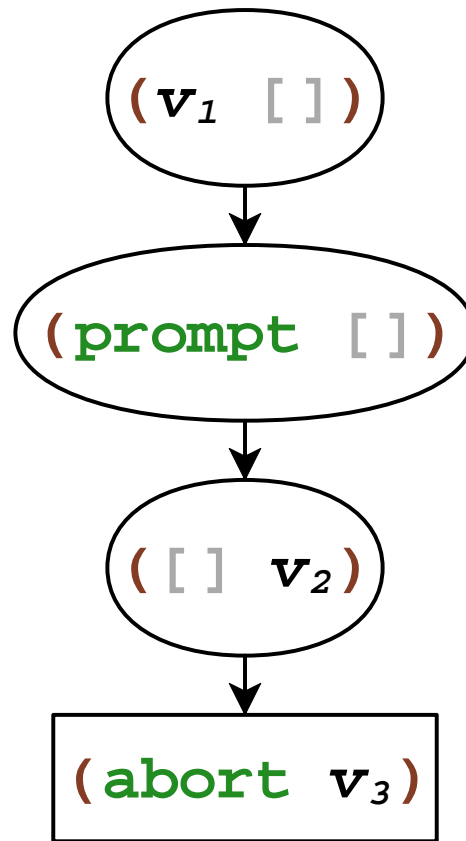
# Delimited Capture



# Delimited Capture



# Delimited Abort



# Delimited Abort

$(\mathbf{v}_1 \quad [ \ ])$

$\mathbf{v}_3$

# Delimited Abort

$$(\mathbf{v}_1 \ \mathbf{v}_3)$$

# Splitting Capture and Abort

- **call/comp** : capture current continuation
- **abort** : abort current continuation
- $\mathcal{F}$  : capture and abort current continuation

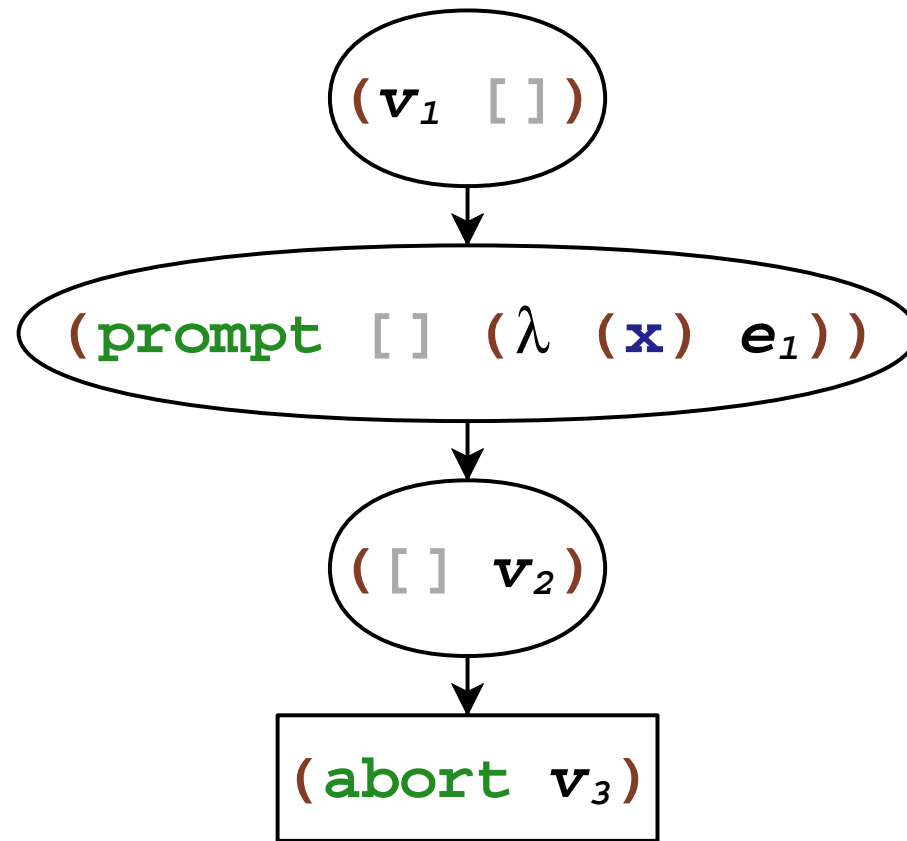
# Splitting Capture and Abort

- **call/comp** : capture current continuation
- **abort** : abort current continuation
- $\mathcal{F}$  : capture and abort current continuation

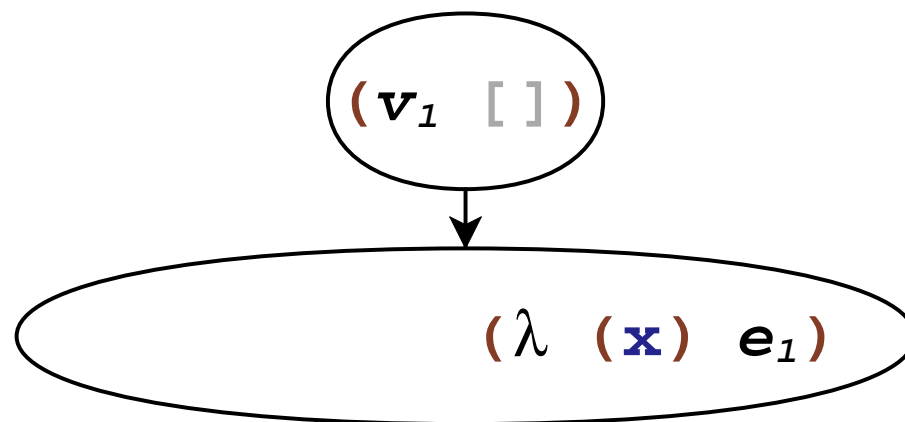
$$\mathcal{F} = (\lambda (f) (\text{call/comp} (\lambda (k) (\text{abort} (\lambda () (f k))))))$$



# Prompt with Handler

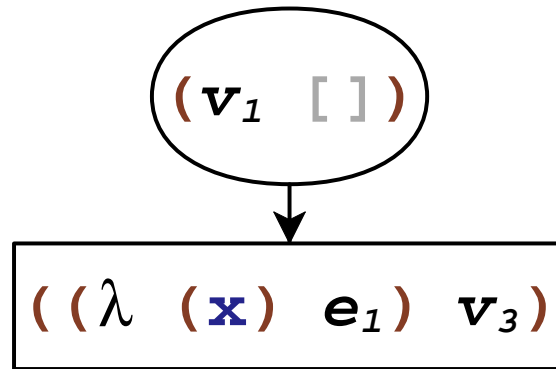


# Prompt with Handler

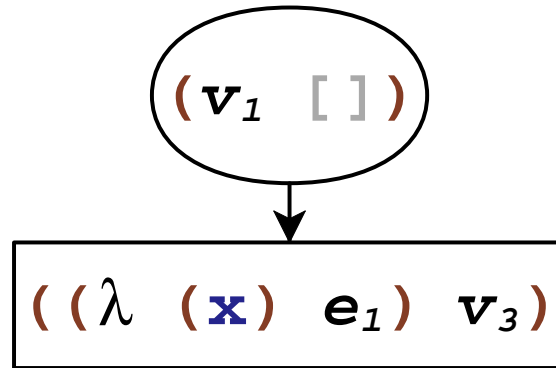


$\mathbf{v}_3$

# Prompt with Handler



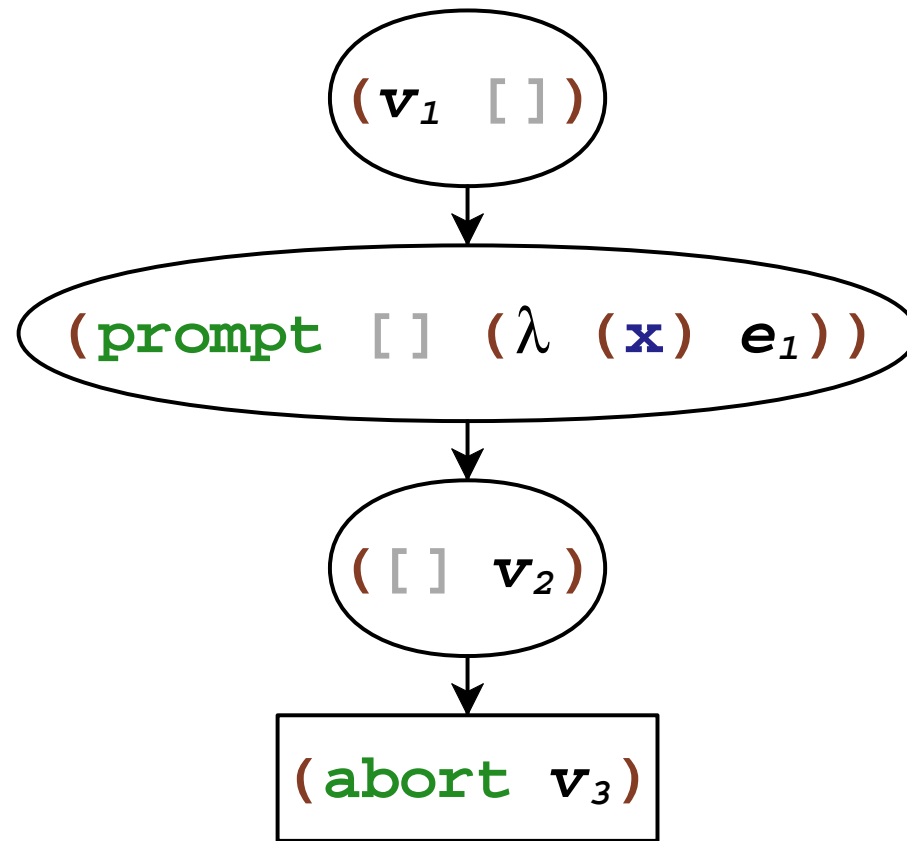
# Prompt with Handler



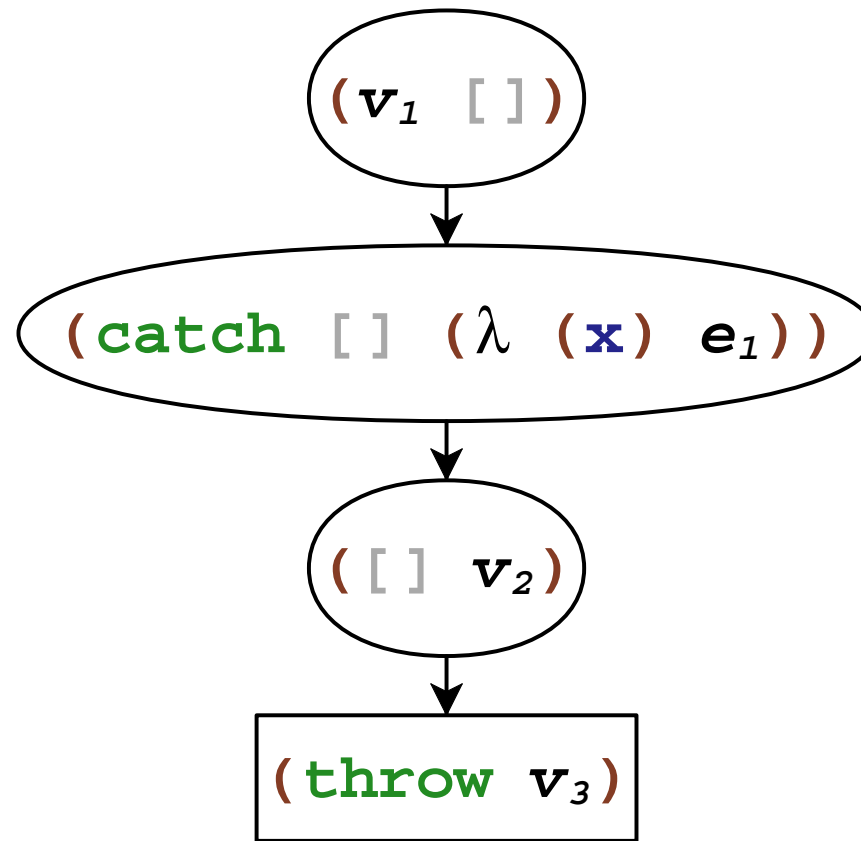
$(\text{prompt } e_1) = (\text{prompt } e_1 \ (\lambda \ (t) \ (t)))$

$\mathcal{F} = (\lambda \ (f) \ (\text{call/comp} \ (\lambda \ (k) \ (\text{abort } (\lambda \ () \ (f \ k))))))$

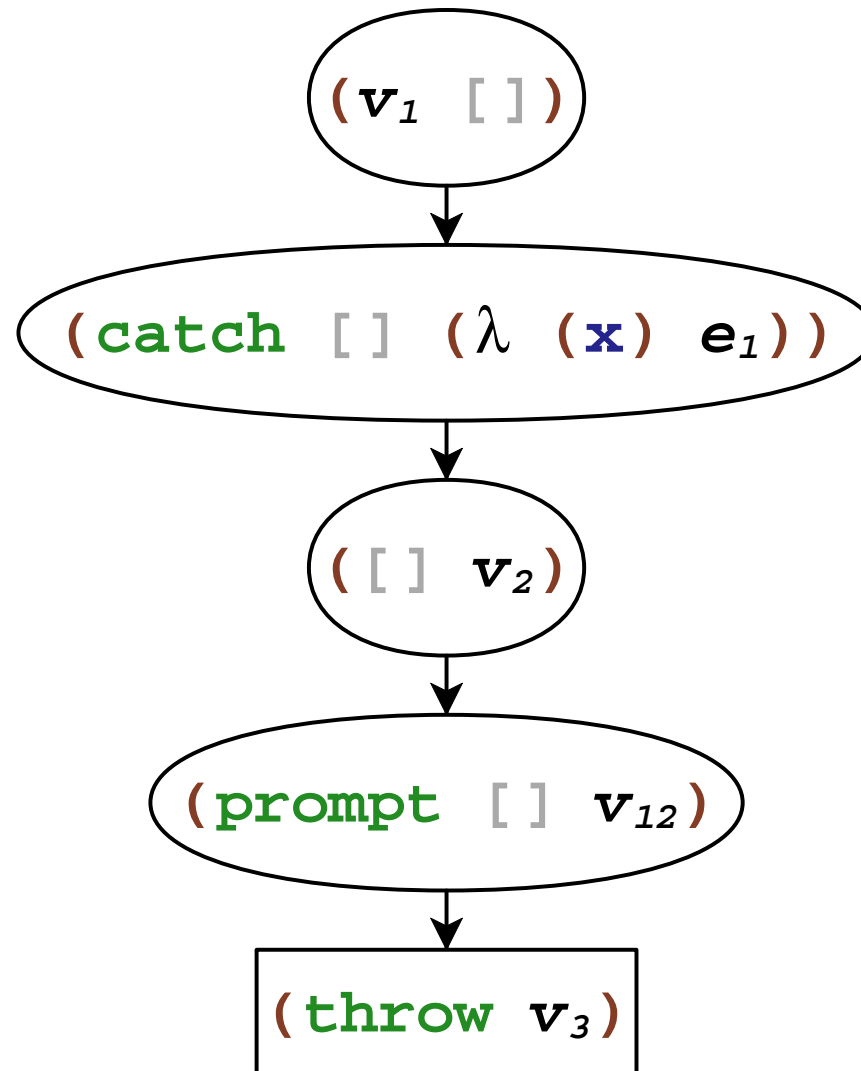
# Prompt with Handler



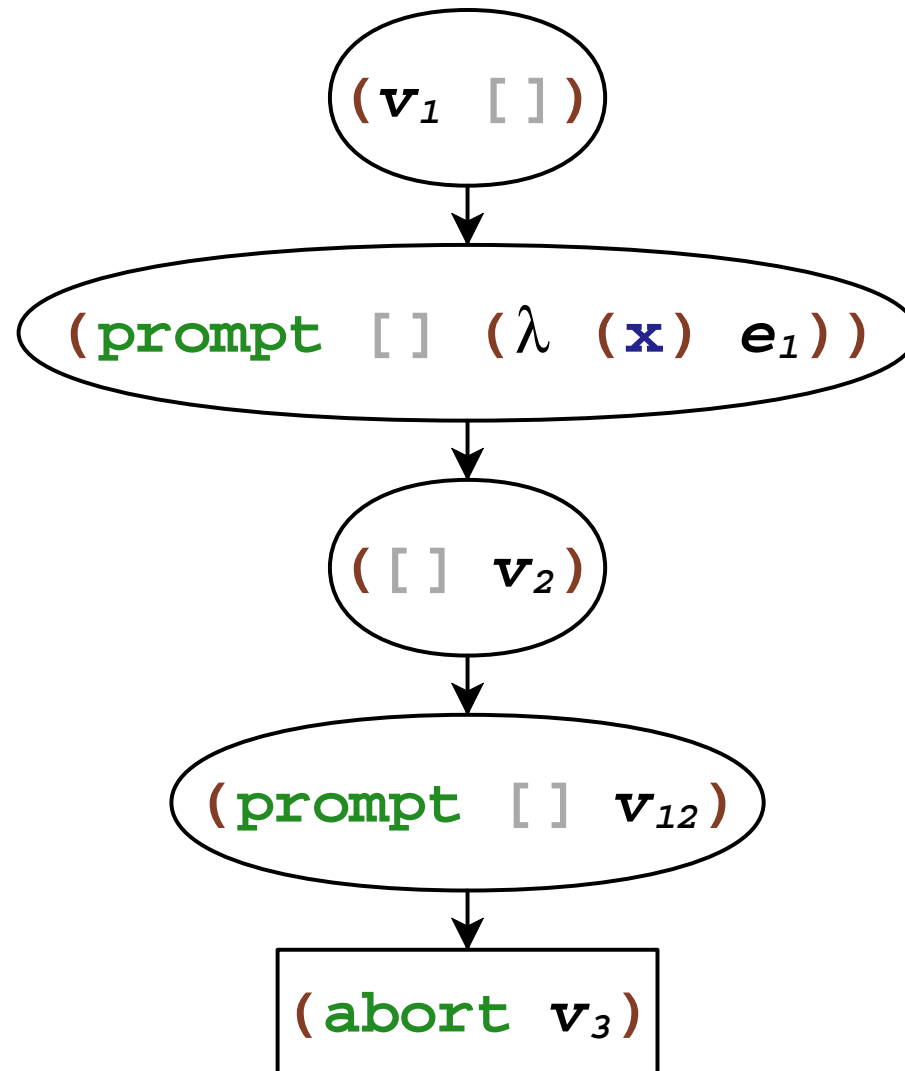
# Catch and Throw



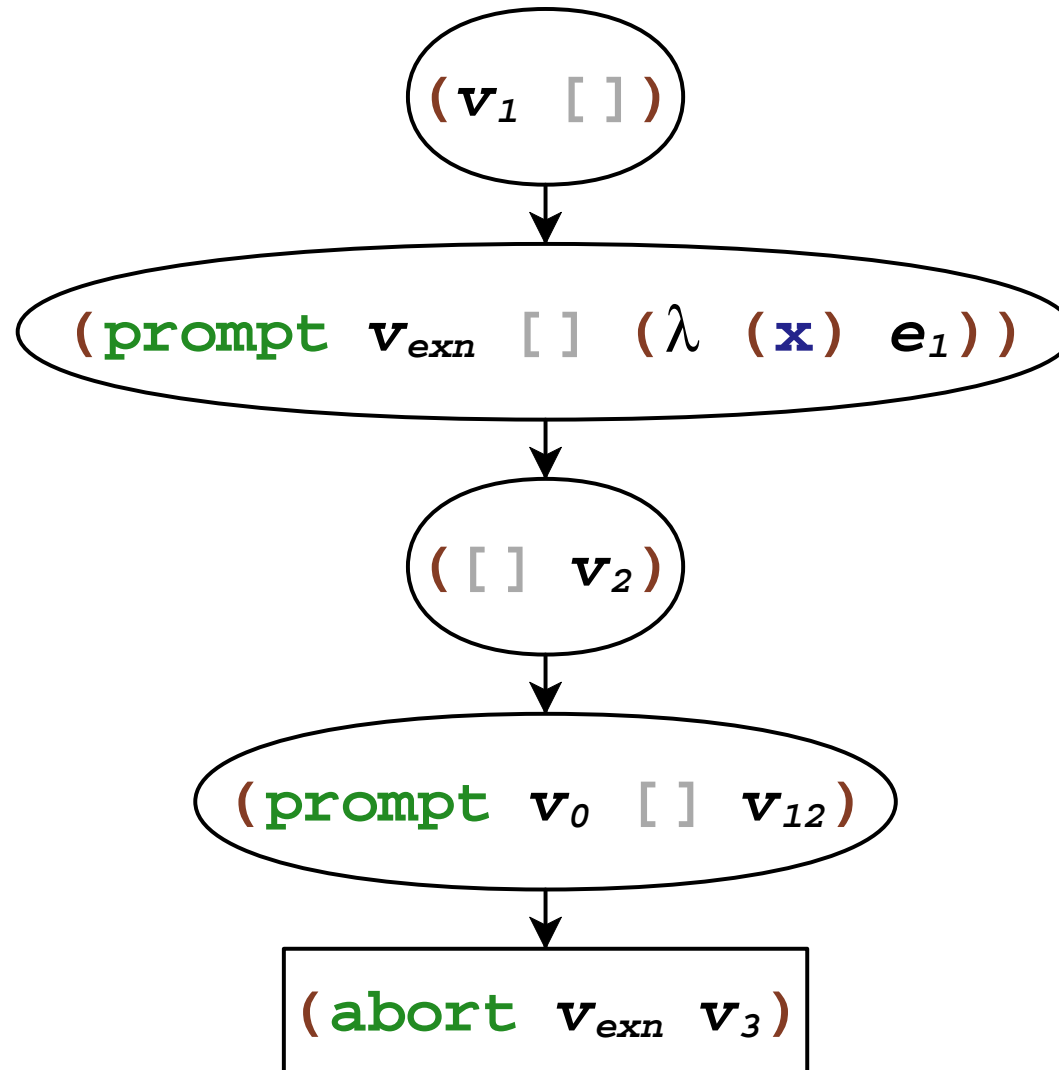
# Catch and Throw



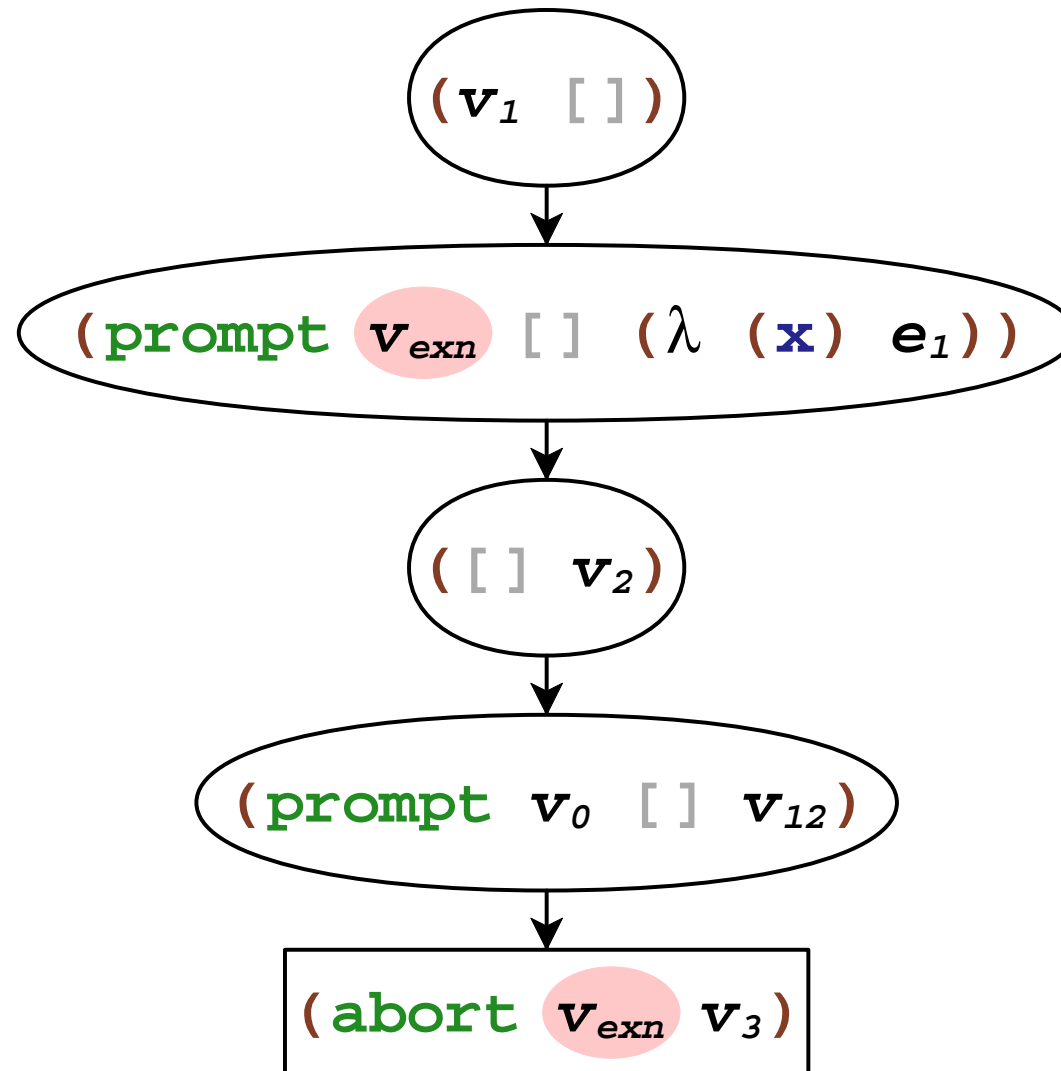
# Catch and Throw



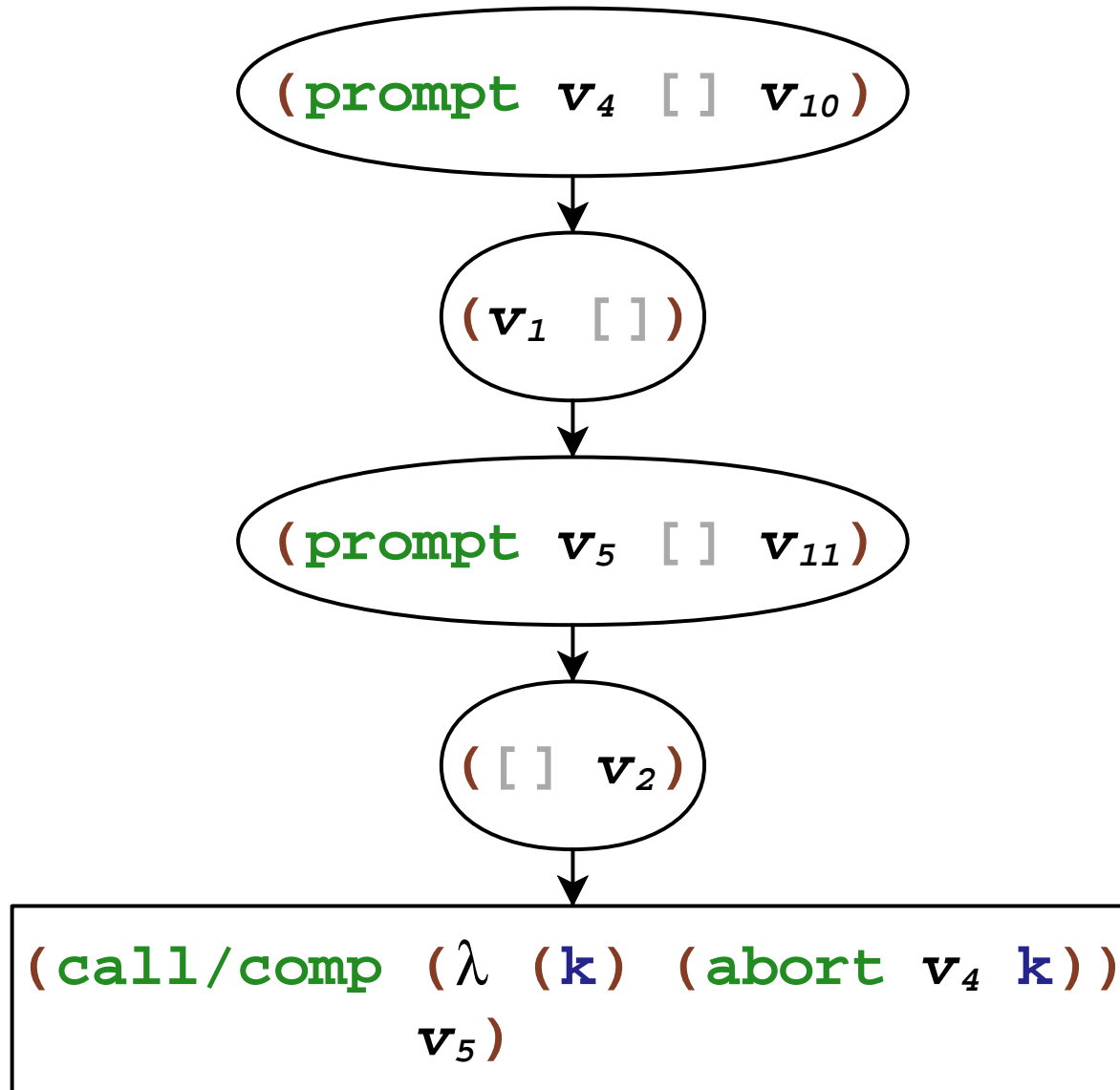
# Catch and Throw



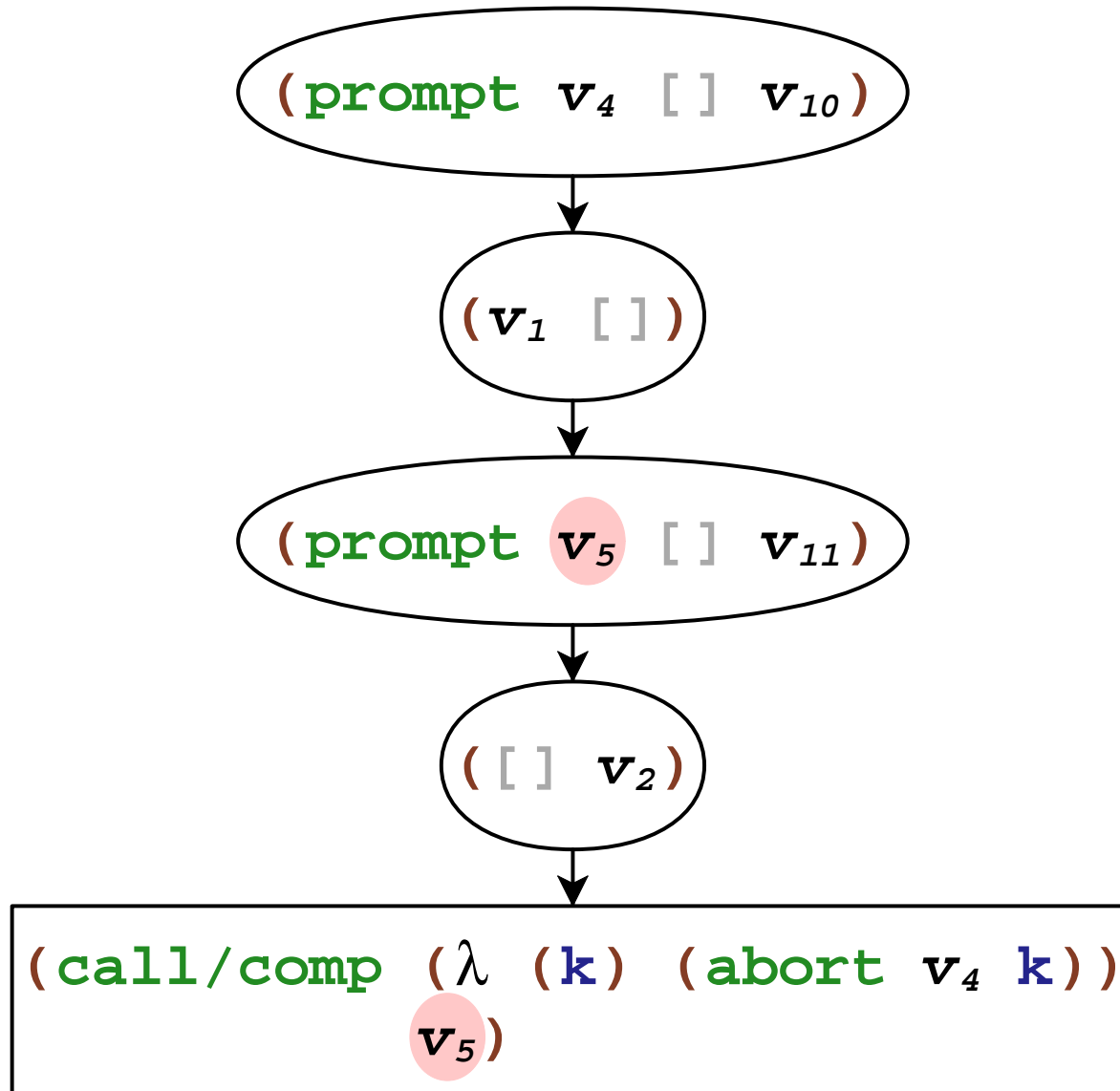
# Tagged Prompts



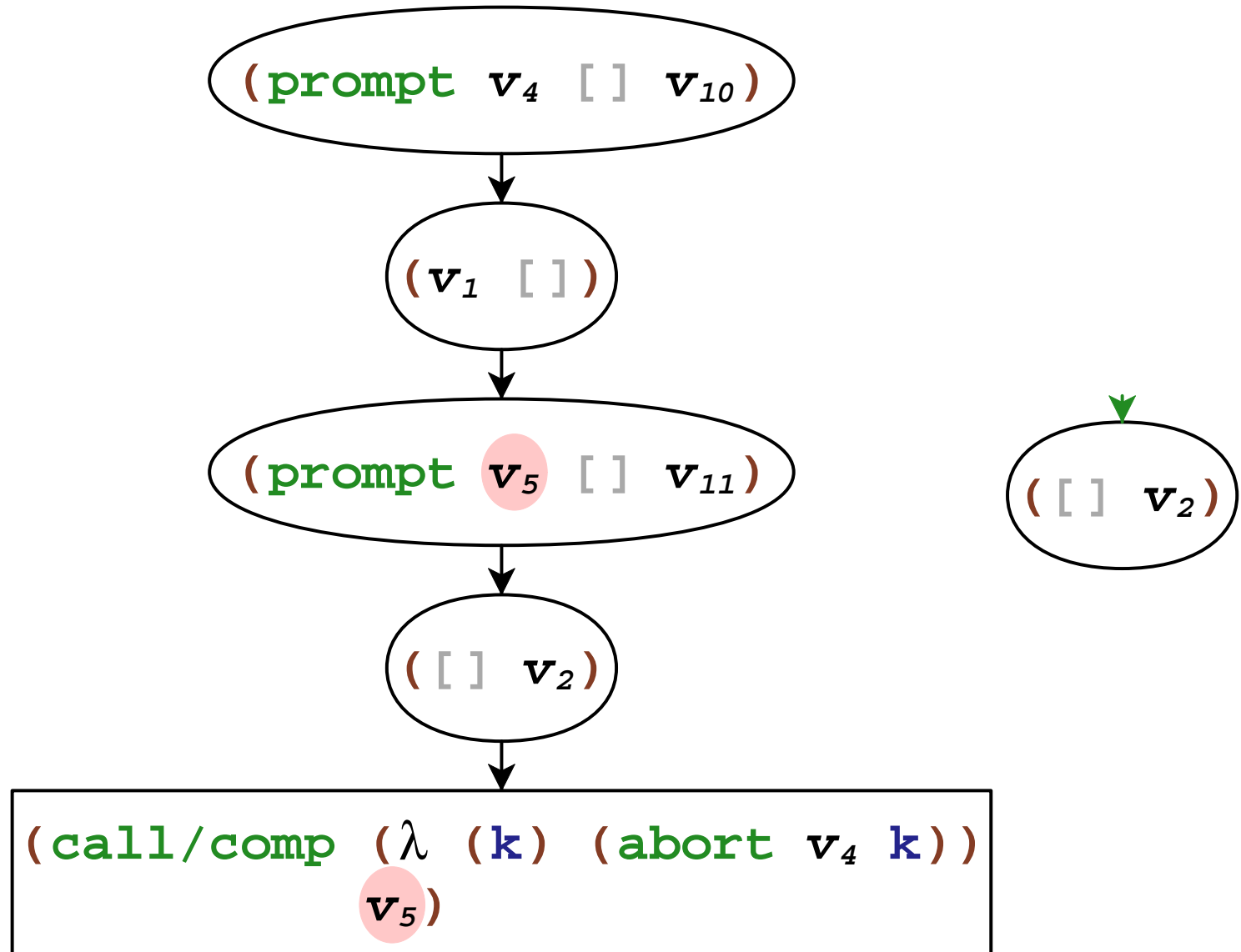
# Tagged Prompts



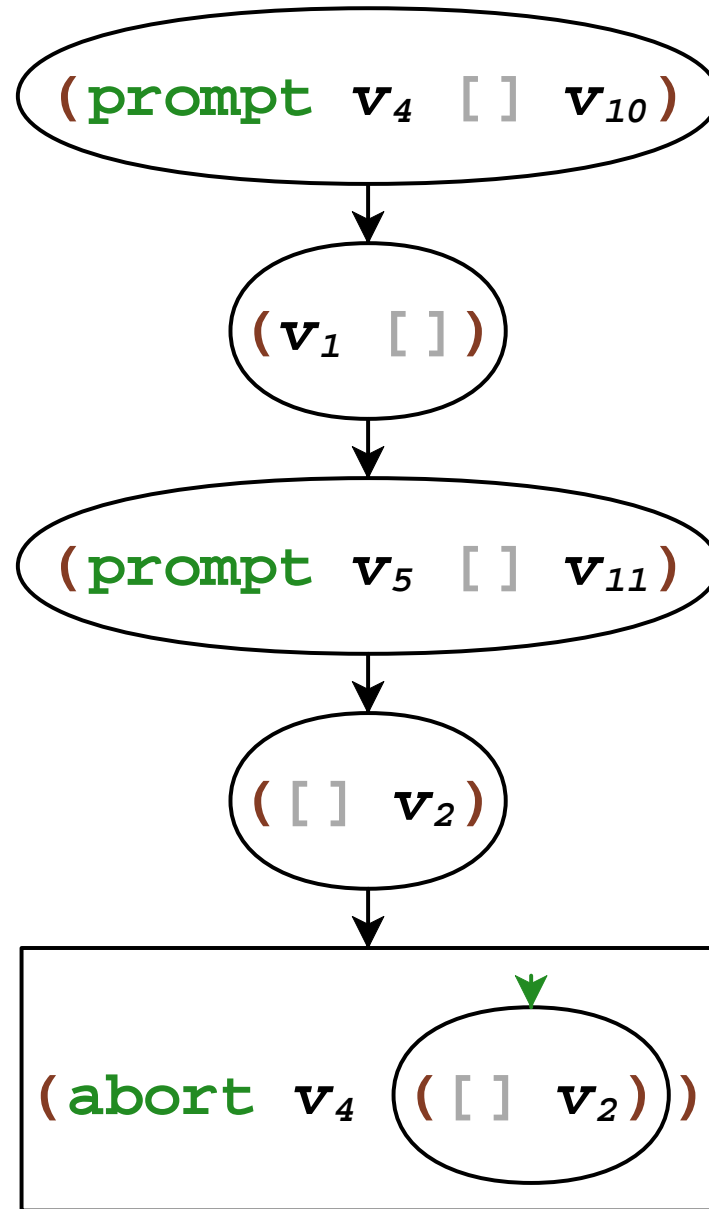
# Tagged Prompts



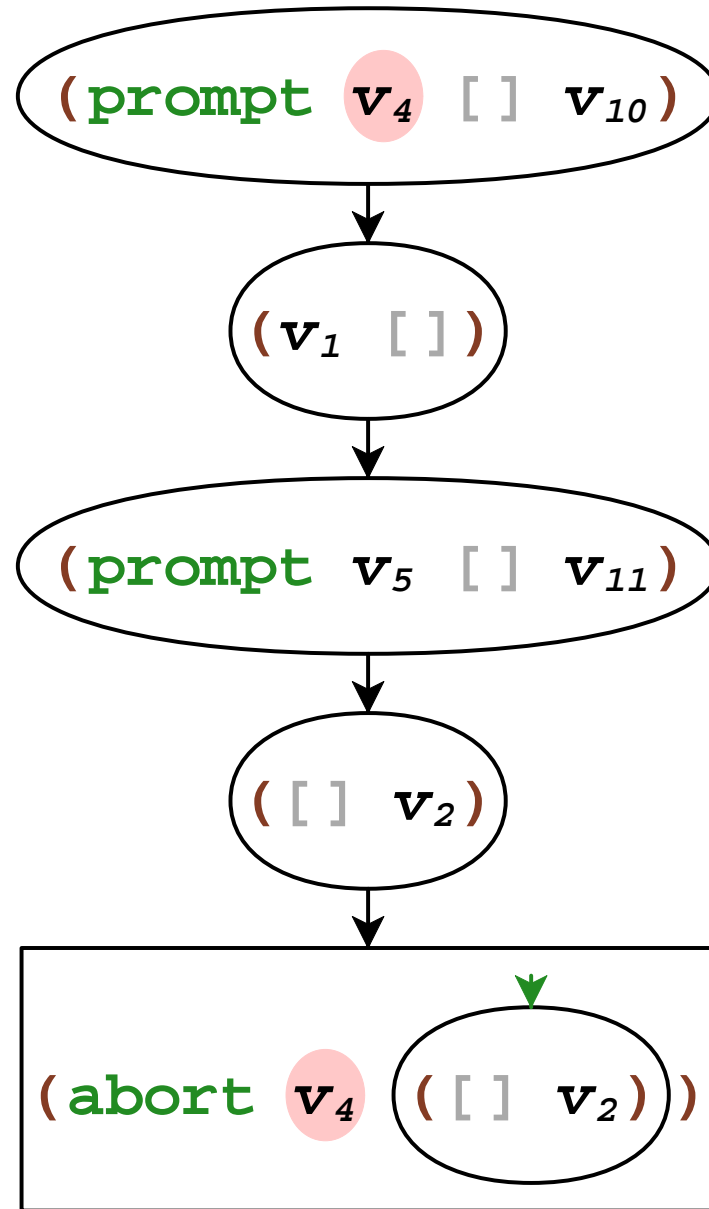
# Tagged Prompts



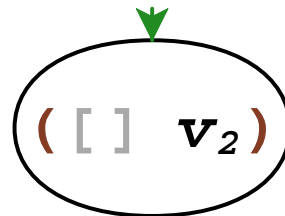
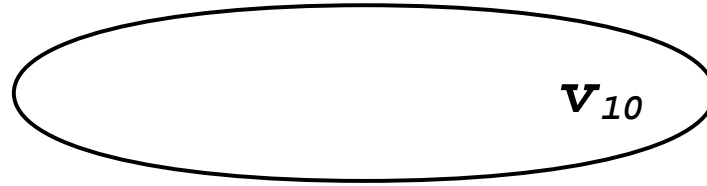
# Tagged Prompts



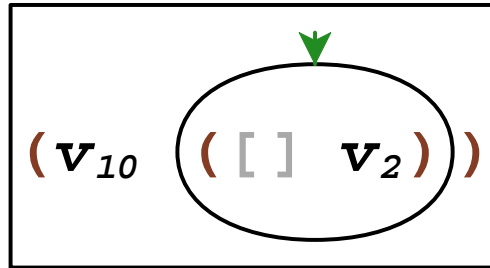
# Tagged Prompts



# Tagged Prompts



# Tagged Prompts

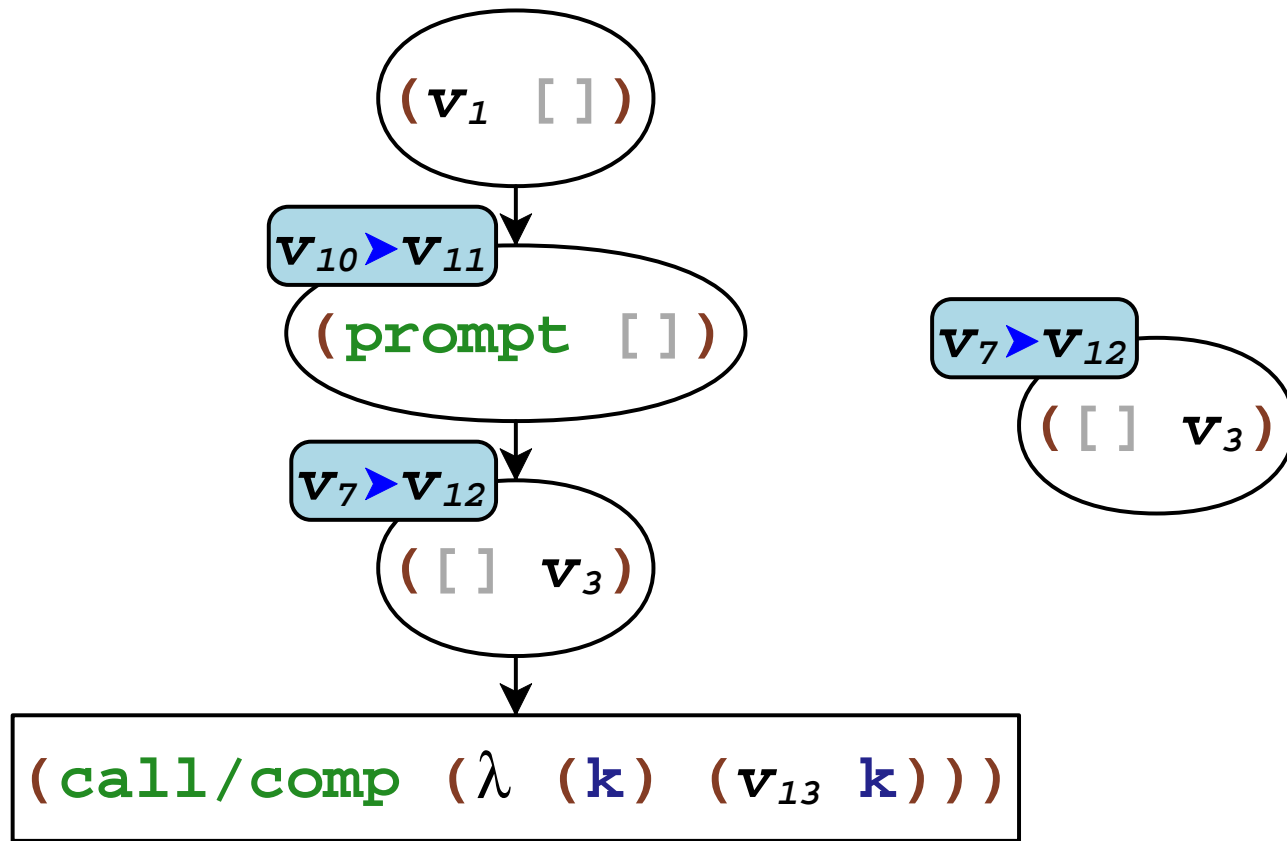


# Continuations Summary

- `prompt` with handler and tag
- `abort` with tag
- `call/comp` with tag
- continuation composition
- plain `call/cc` uses default tag

[Sitaram PLDI'93]

# Continuation Marks



# Dynamic Binding Summary

- `call/cm` to add marks
- `current-marks` to get marks, up to a tag
- capture marks in `call/comp`
- splice marks in continuation composition

# Side Effects and Control

```
(define (with-resource work-thunk)
  (begin
    (grab-resource)
    (work-thunk)
    (release-with-resource)))
```

# Dynamic Wind

```
(define (with-resource work-thunk)
  (dynamic-wind
    (λ () (grab-resource))
    (λ () (work-thunk))
    (λ () (release-with-resource))))
```

# Dynamic Wind

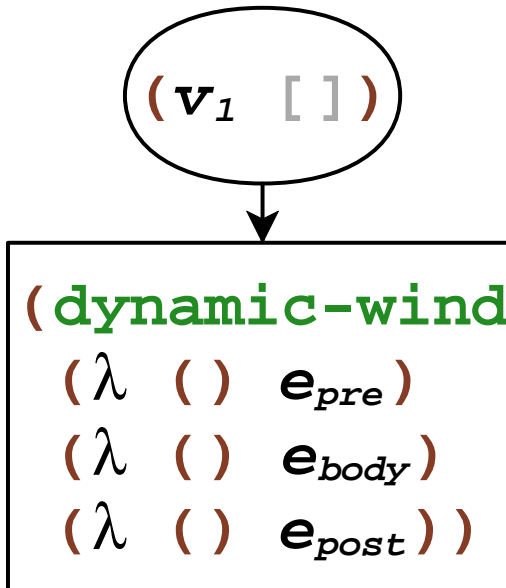
(dynamic-wind

( $\lambda$  ()  $e_{pre}$ )

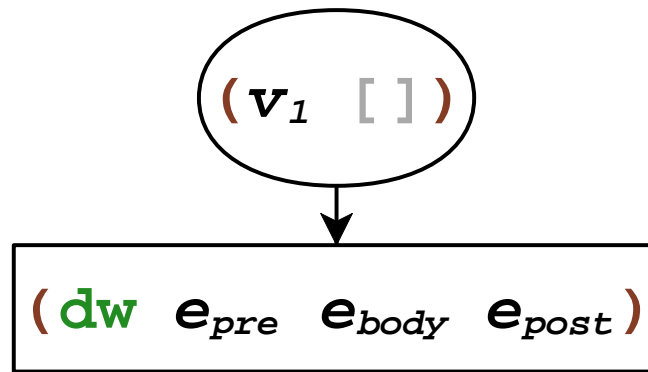
( $\lambda$  ()  $e_{body}$ )

( $\lambda$  ()  $e_{post}$ ))

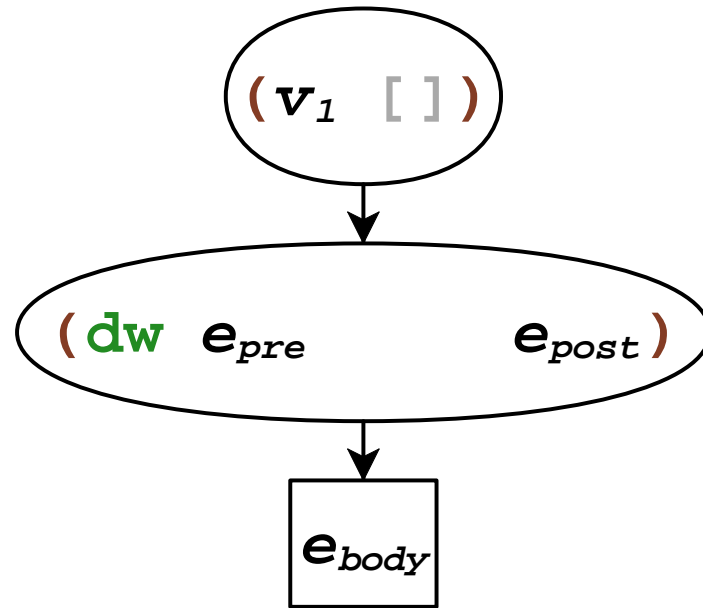
# Dynamic Wind



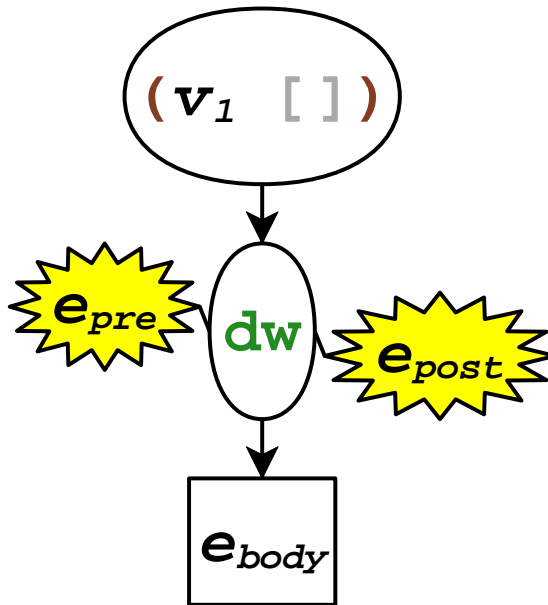
# Dynamic Wind



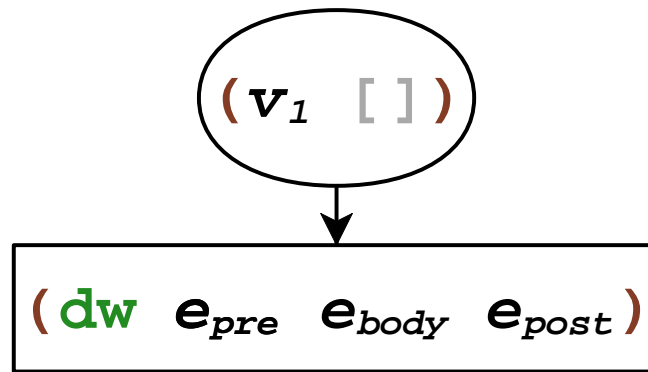
# Dynamic Wind



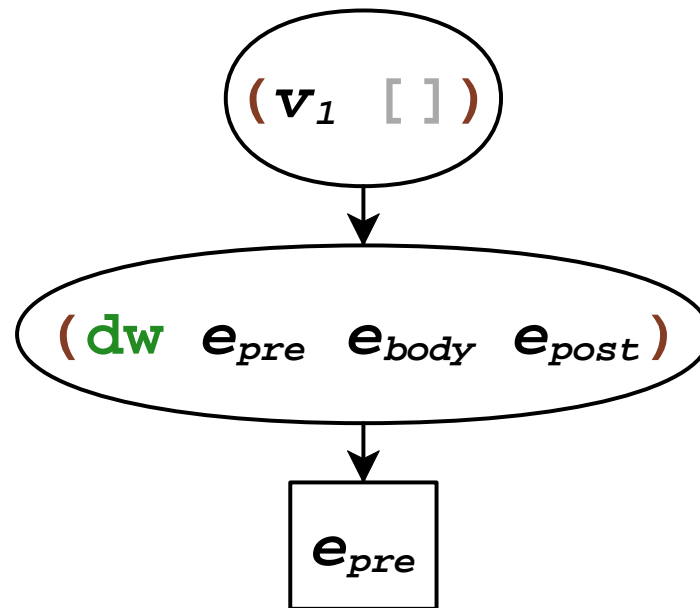
# Dynamic Wind



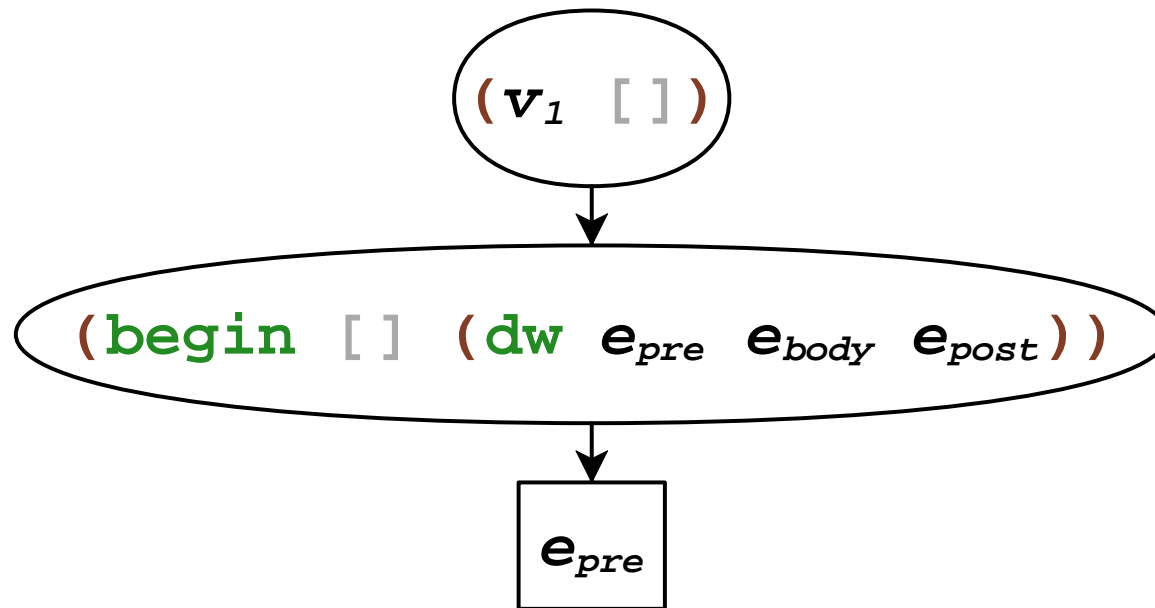
# Dynamic Wind Evaluation



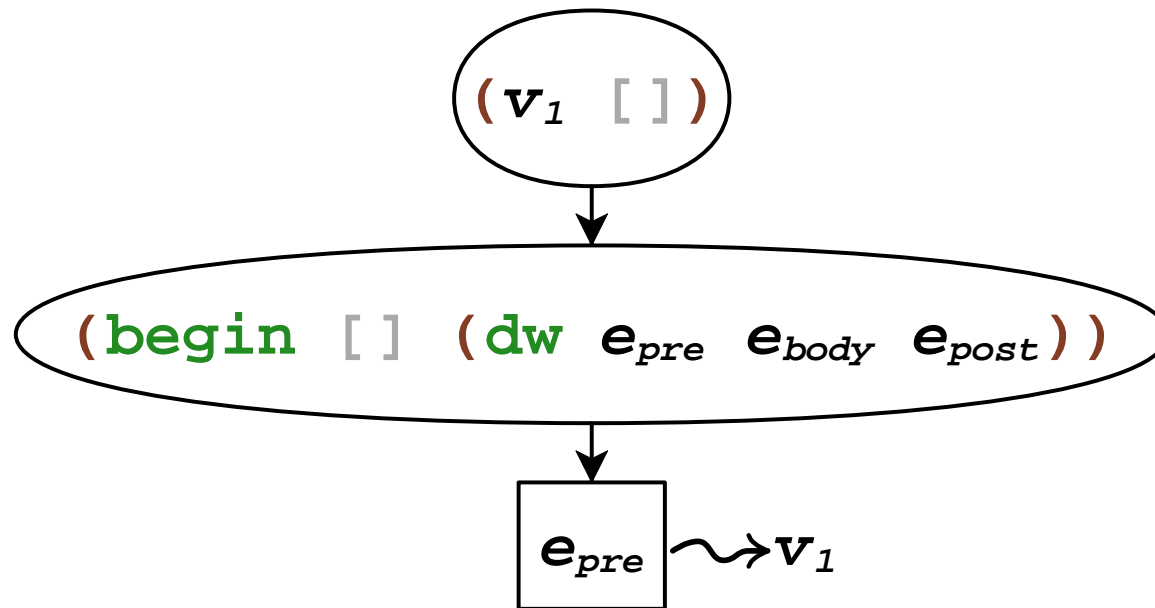
# Dynamic Wind Evaluation



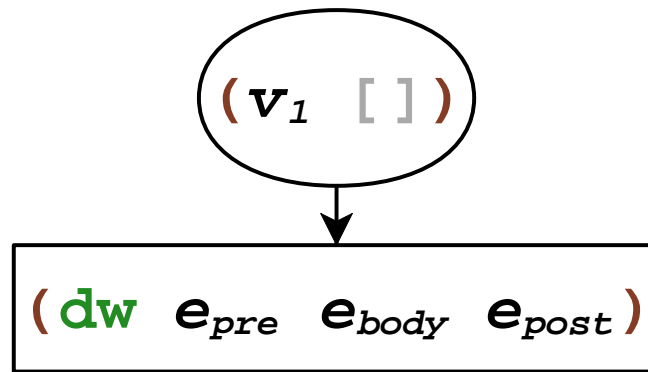
# Dynamic Wind Evaluation



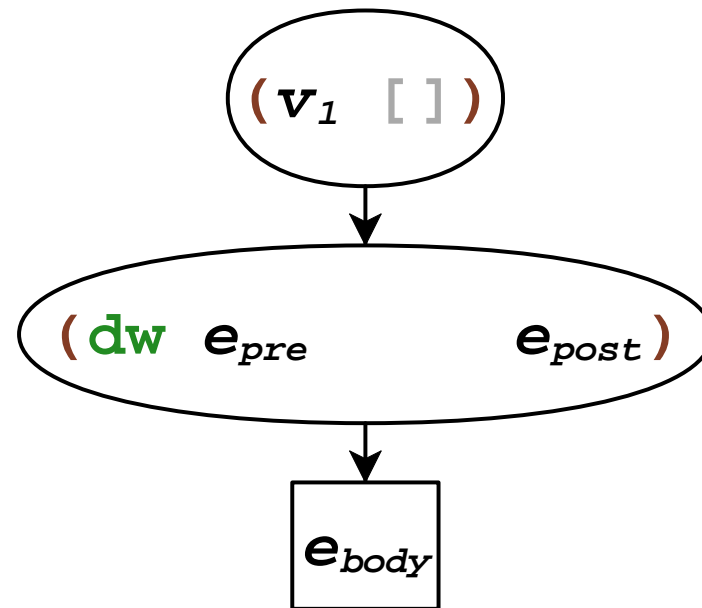
# Dynamic Wind Evaluation



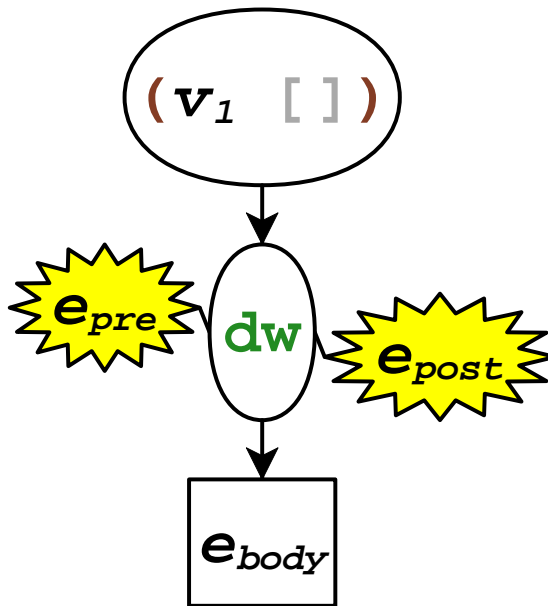
# Dynamic Wind Evaluation



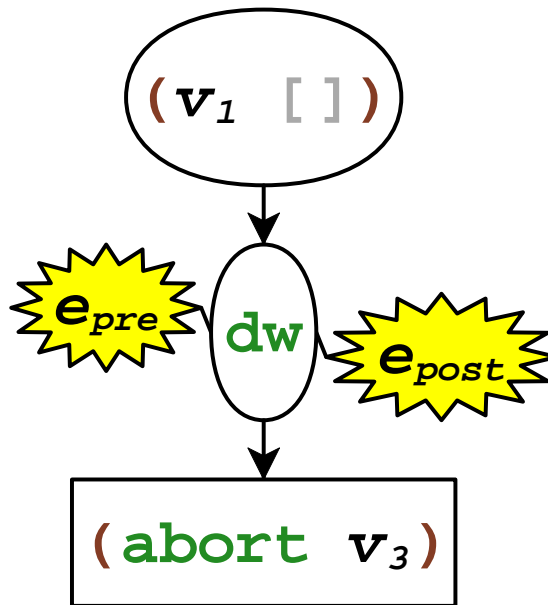
# Dynamic Wind Evaluation



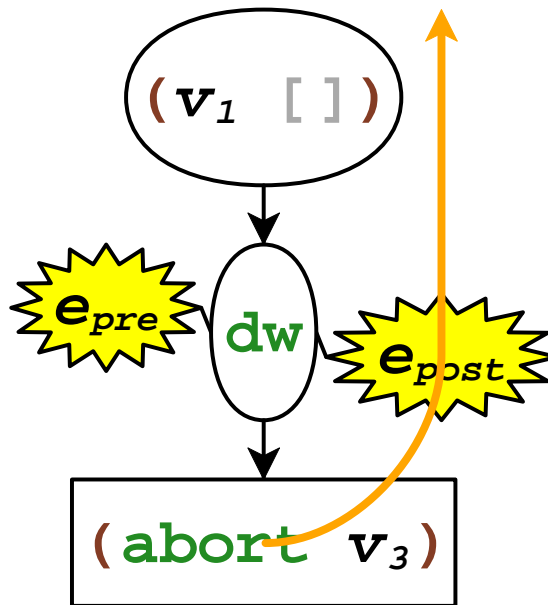
# Dynamic Wind and Jumps



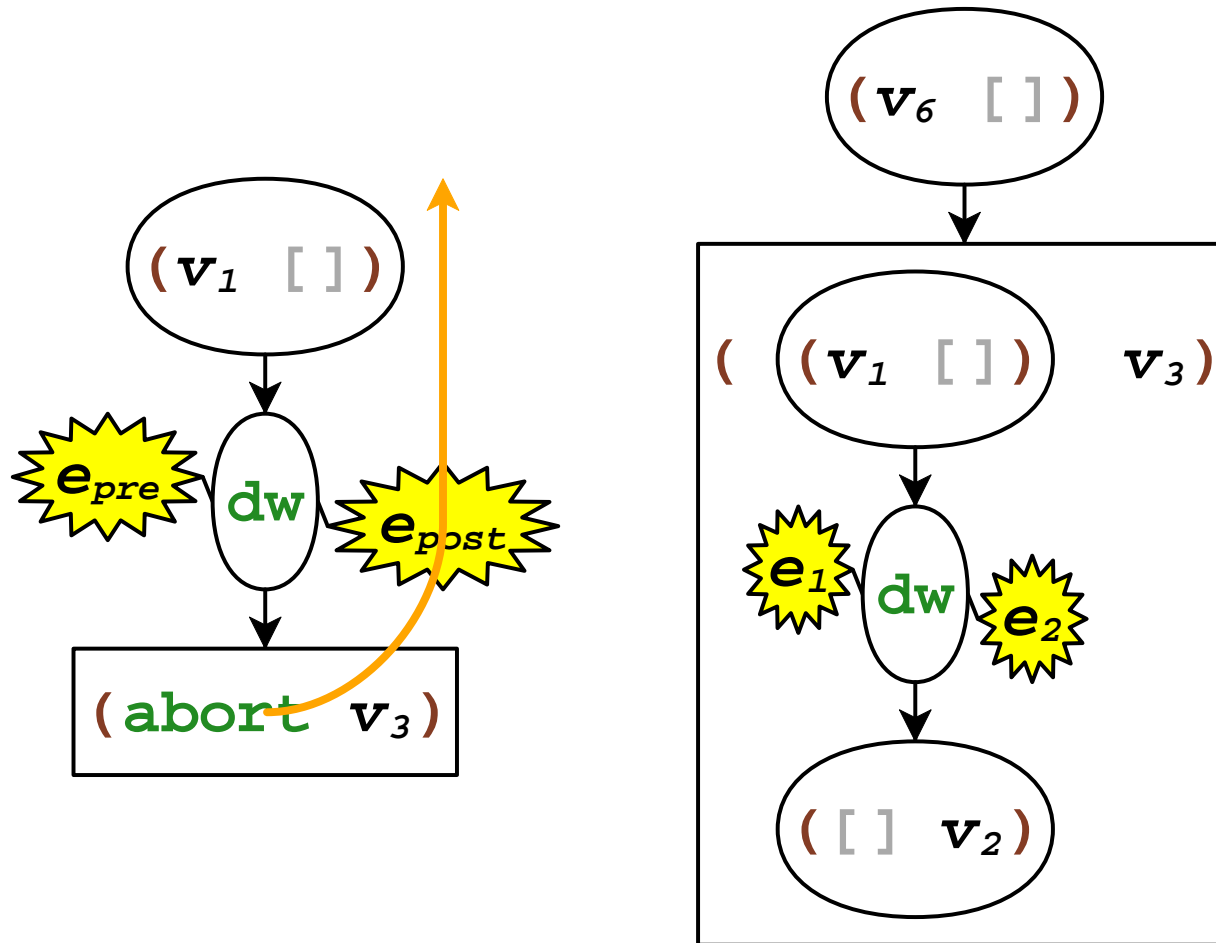
# Dynamic Wind and Jumps



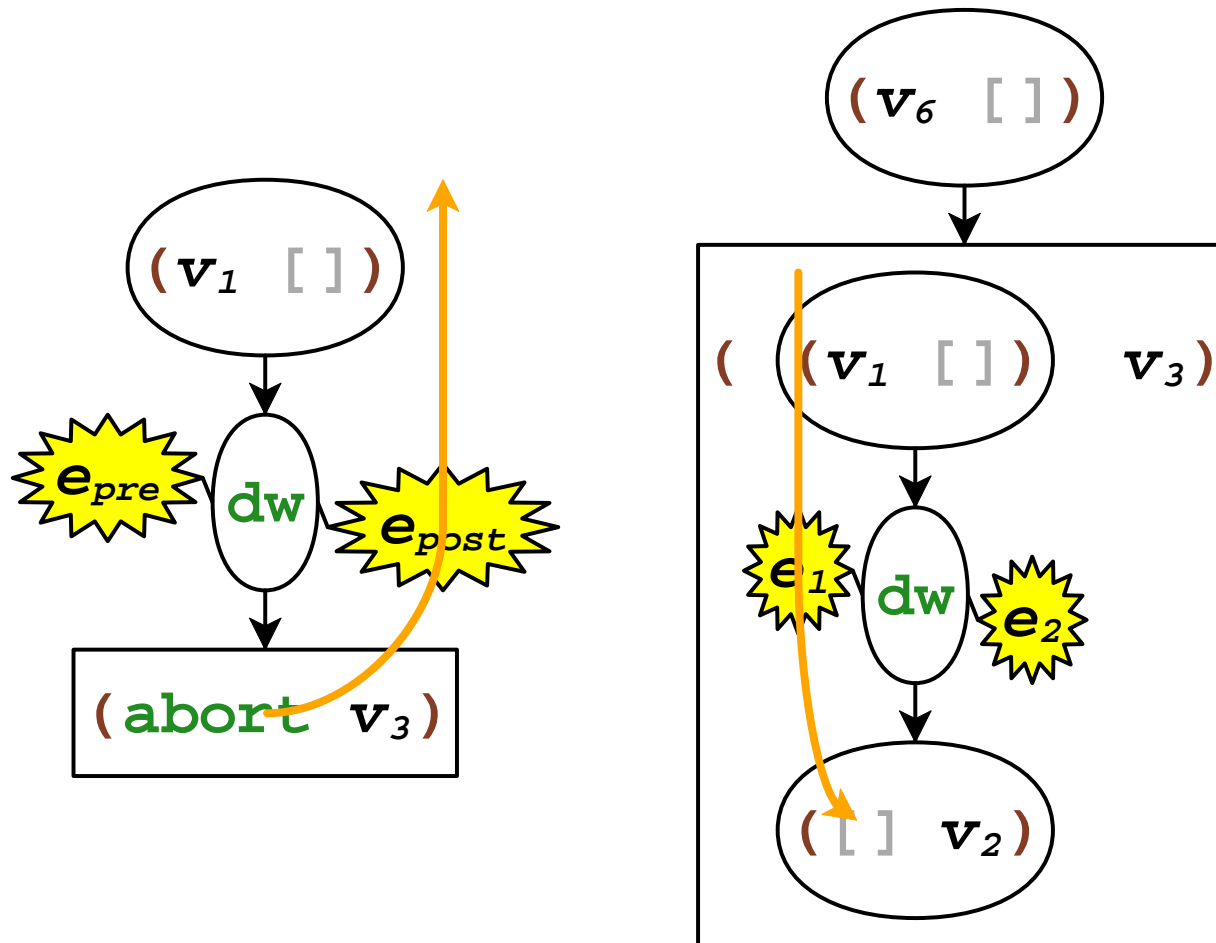
# Dynamic Wind and Jumps



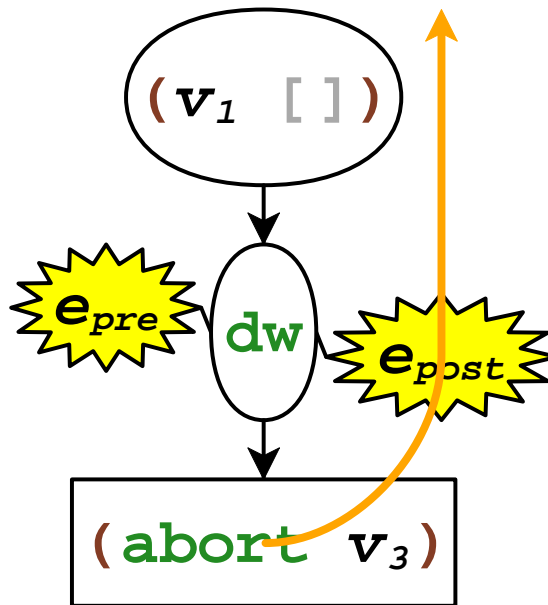
# Dynamic Wind and Jumps



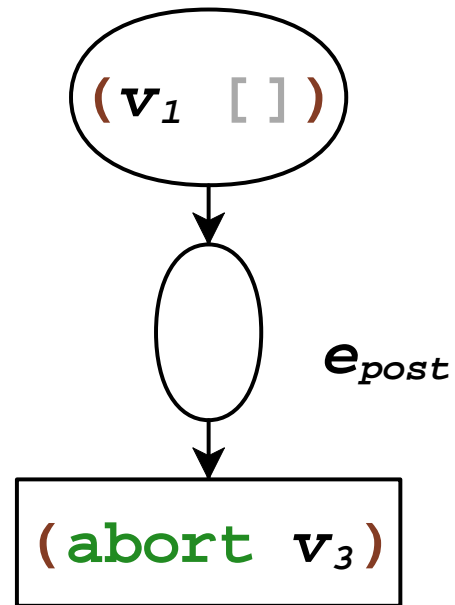
# Dynamic Wind and Jumps



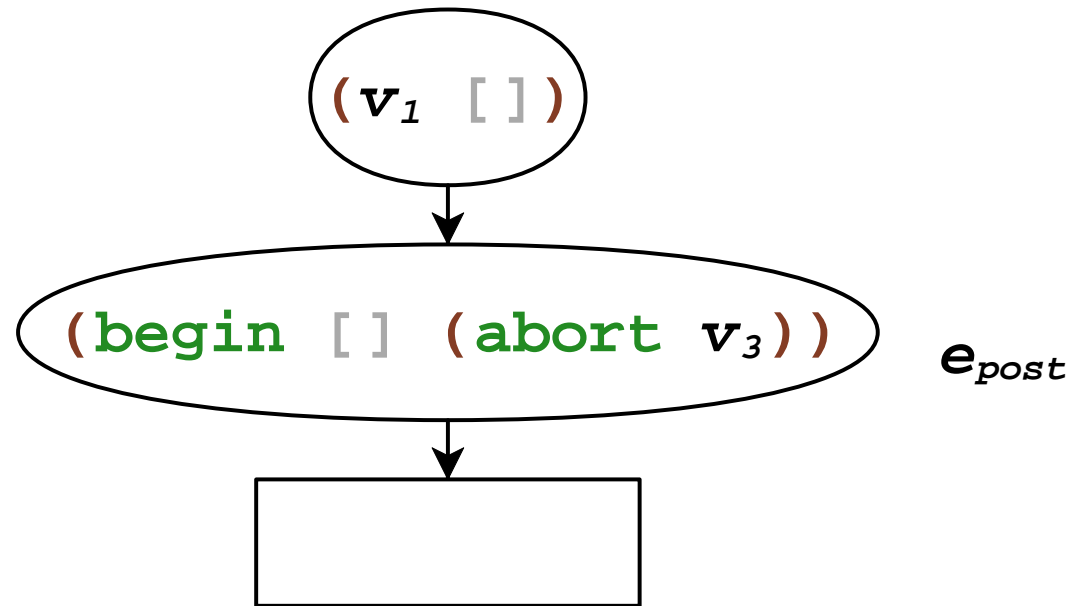
# Dynamic Wind and Abort Evaluation



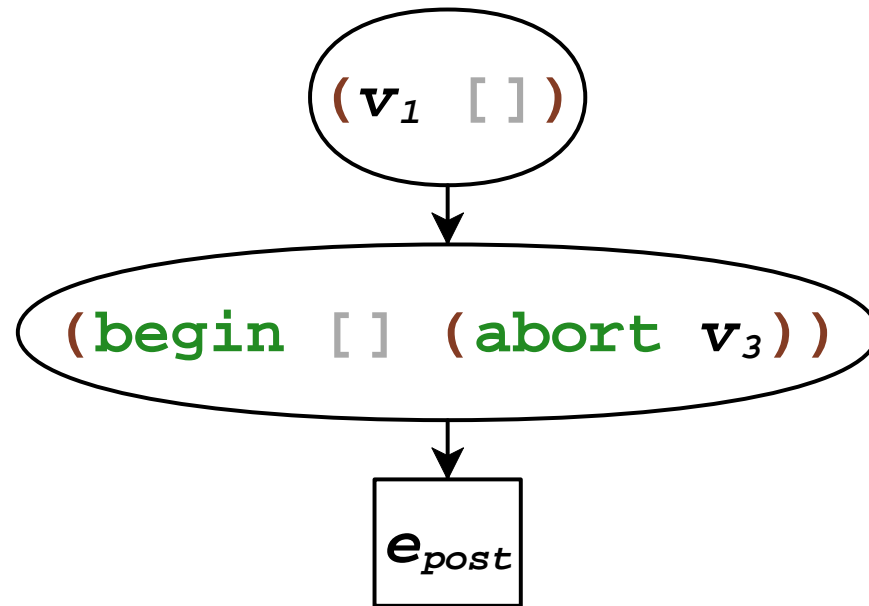
# Dynamic Wind and Abort Evaluation



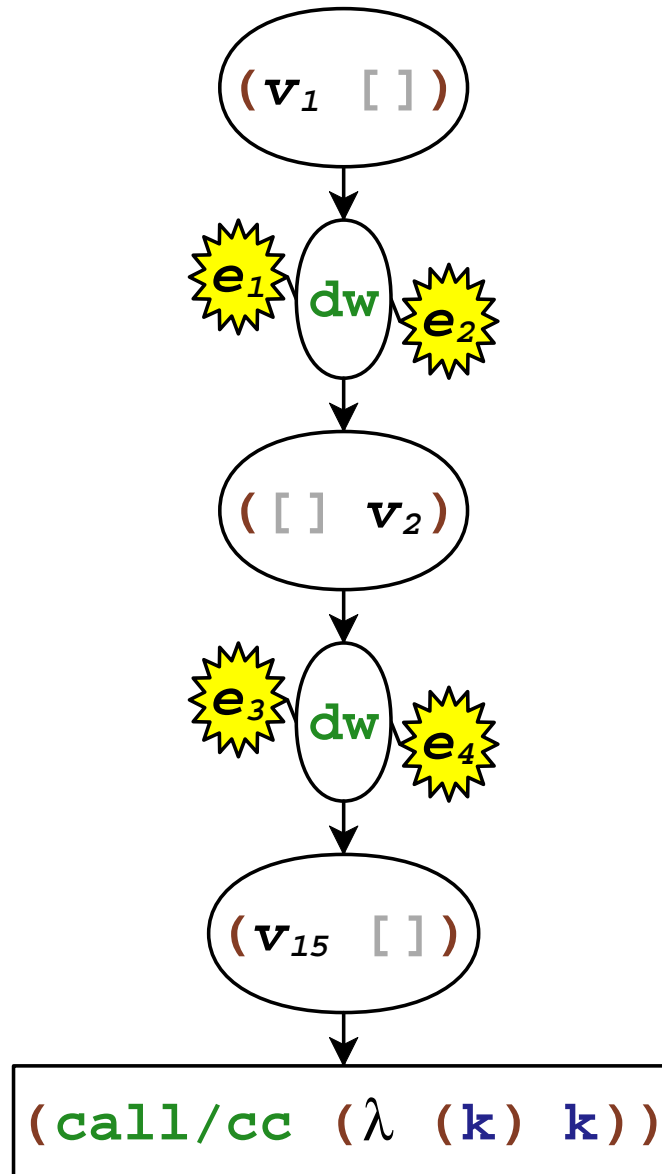
# Dynamic Wind and Abort Evaluation



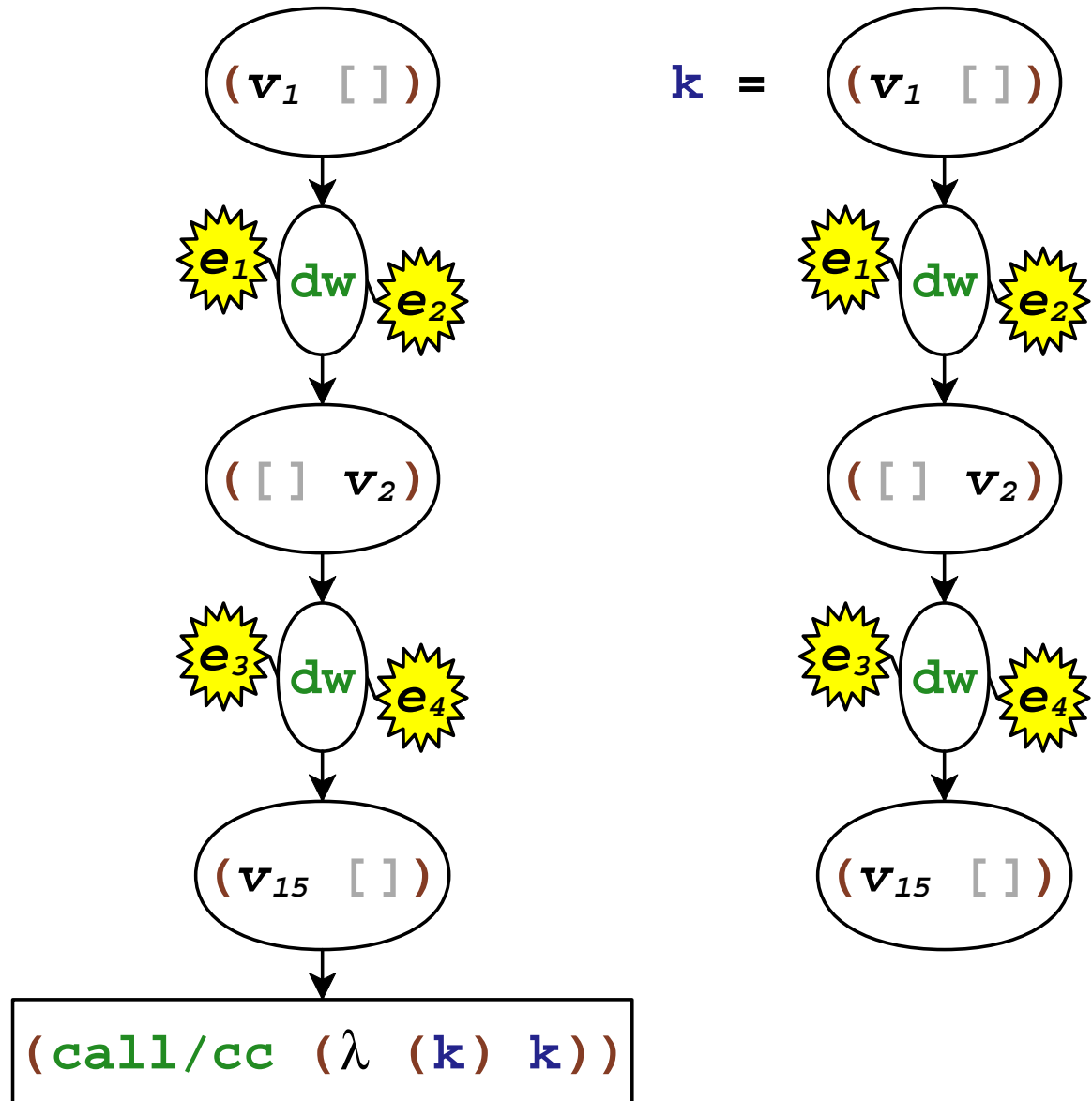
# Dynamic Wind and Abort Evaluation



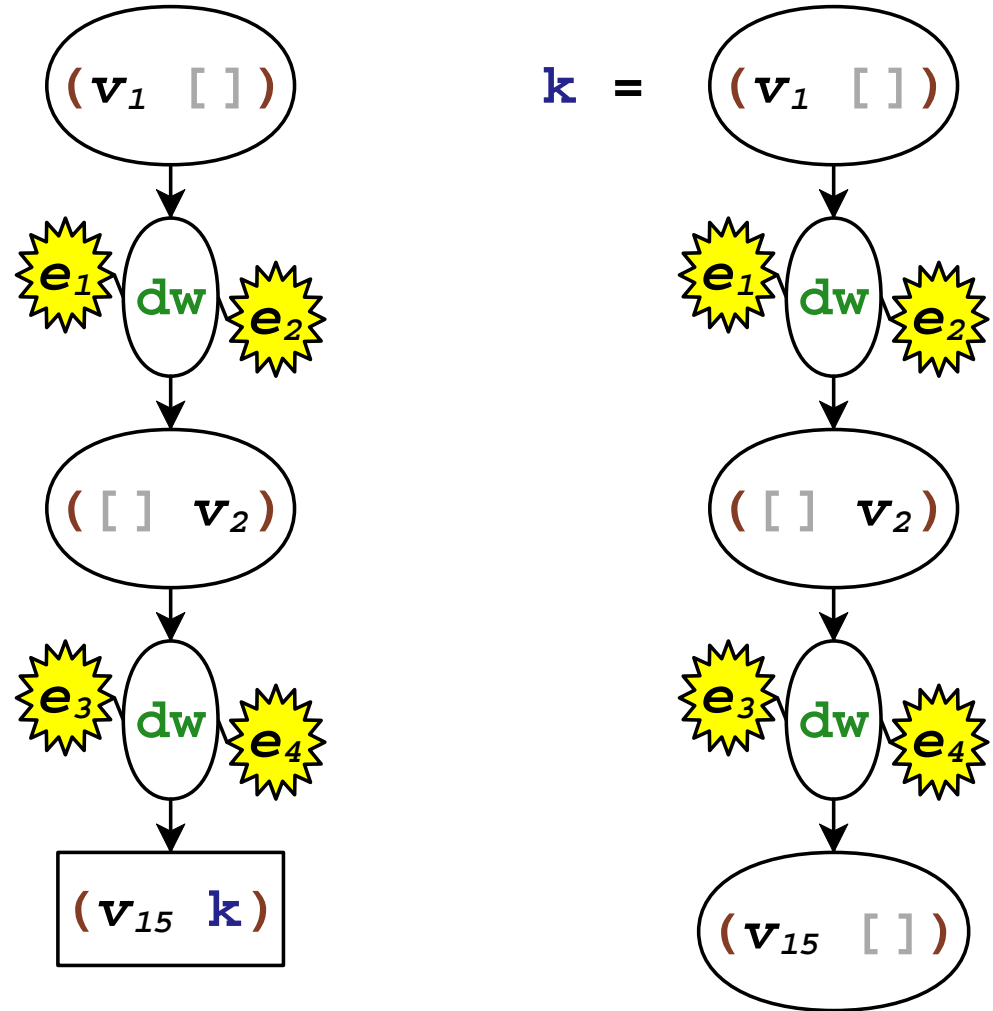
# Dynamic Wind and Call/cc



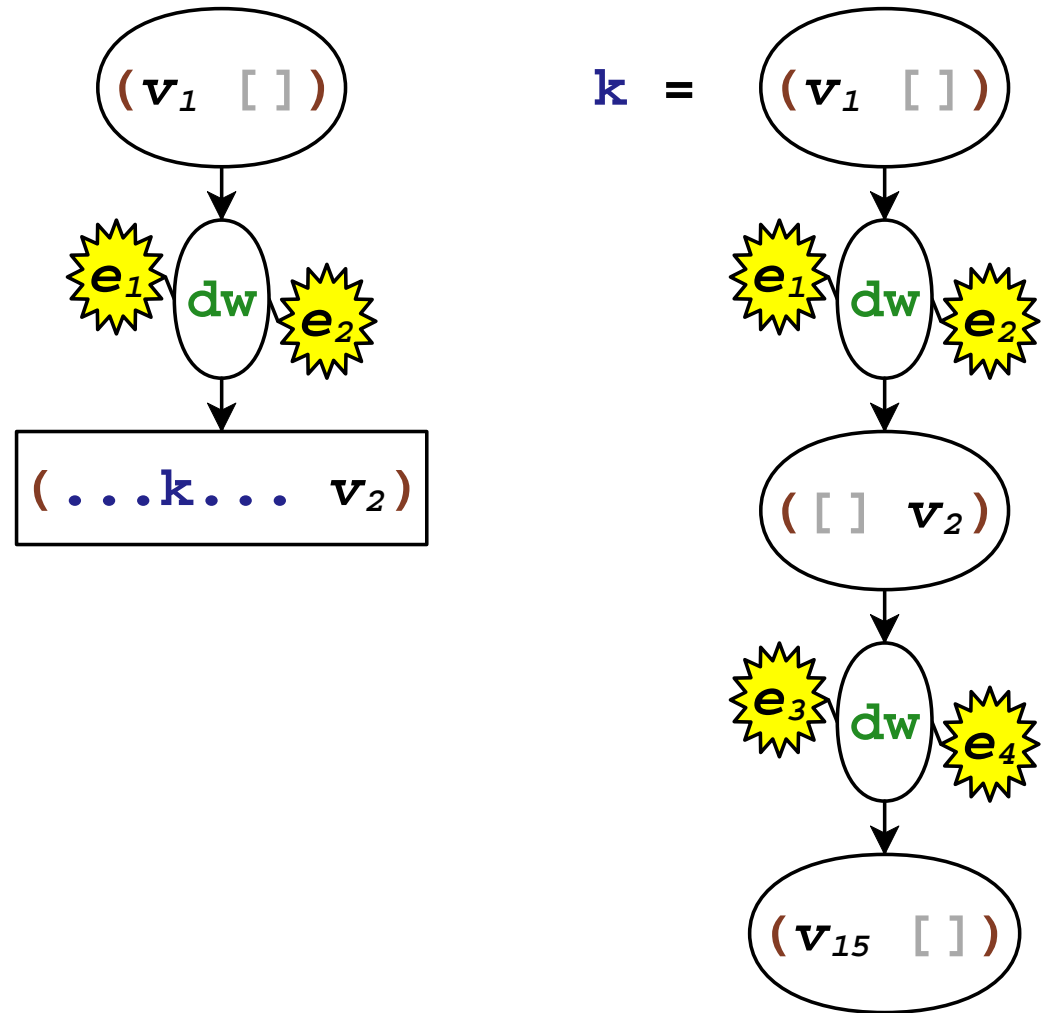
# Dynamic Wind and Call/cc



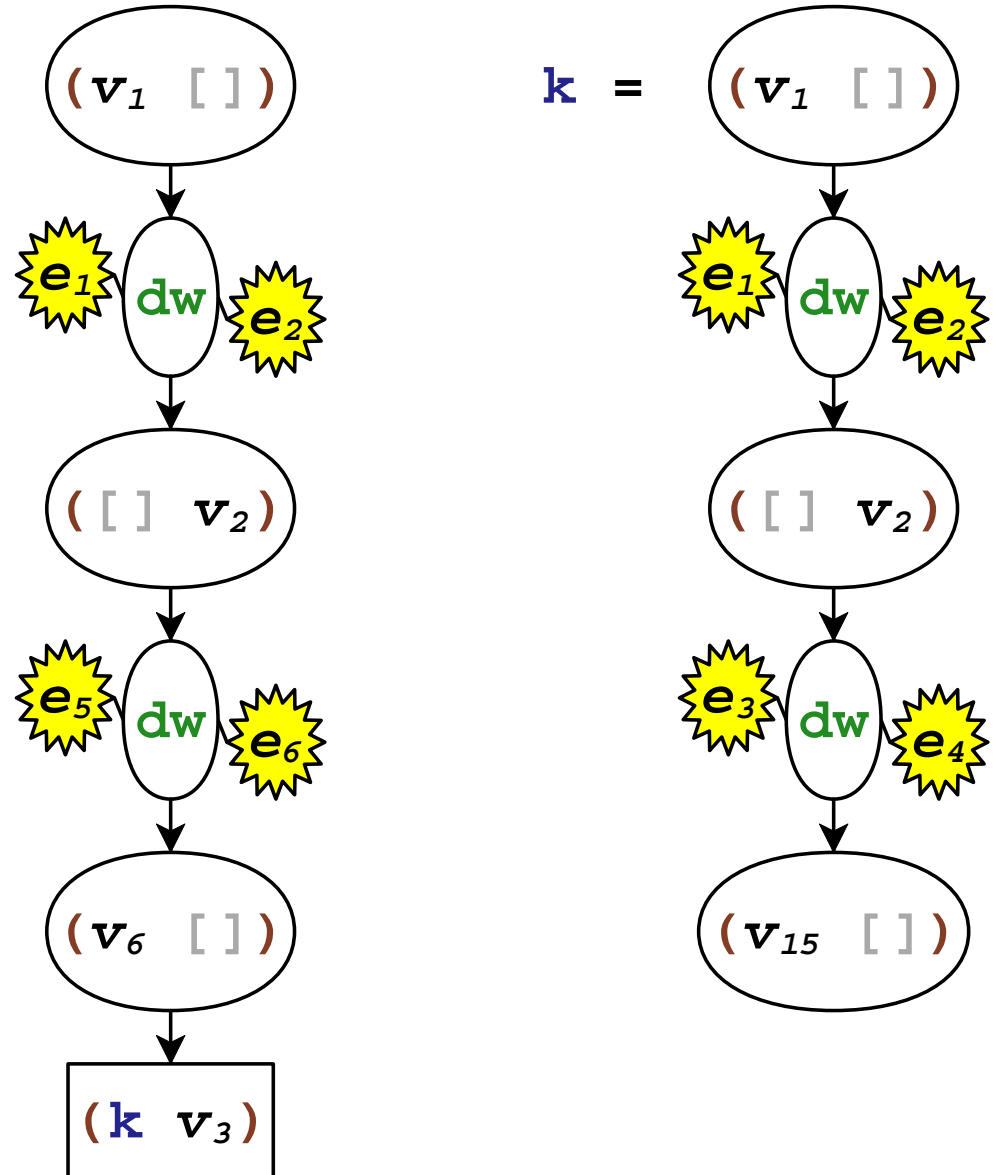
# Dynamic Wind and Call/cc



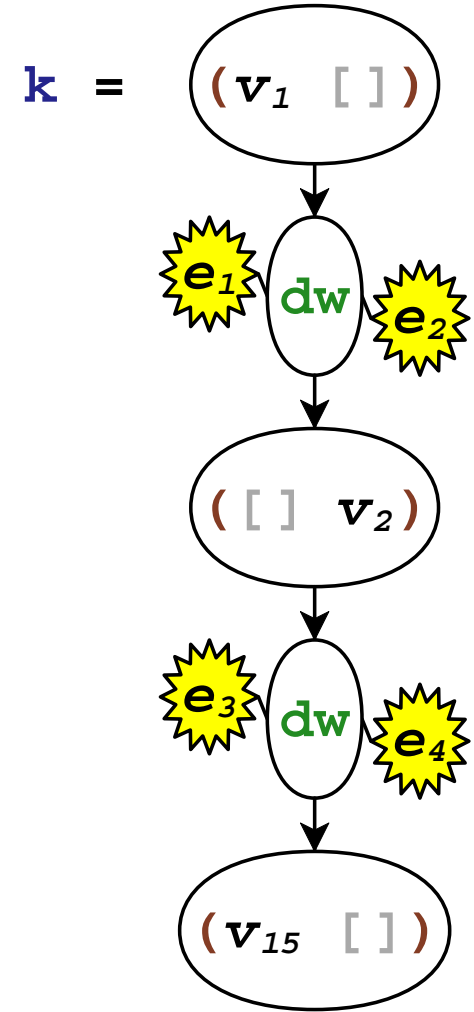
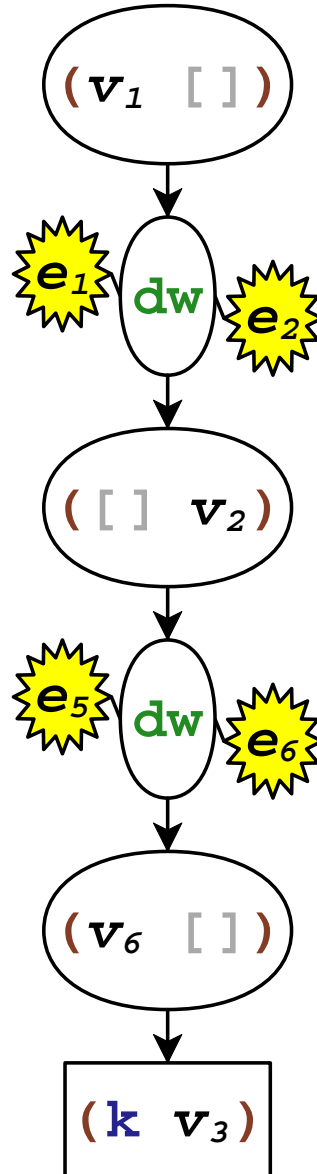
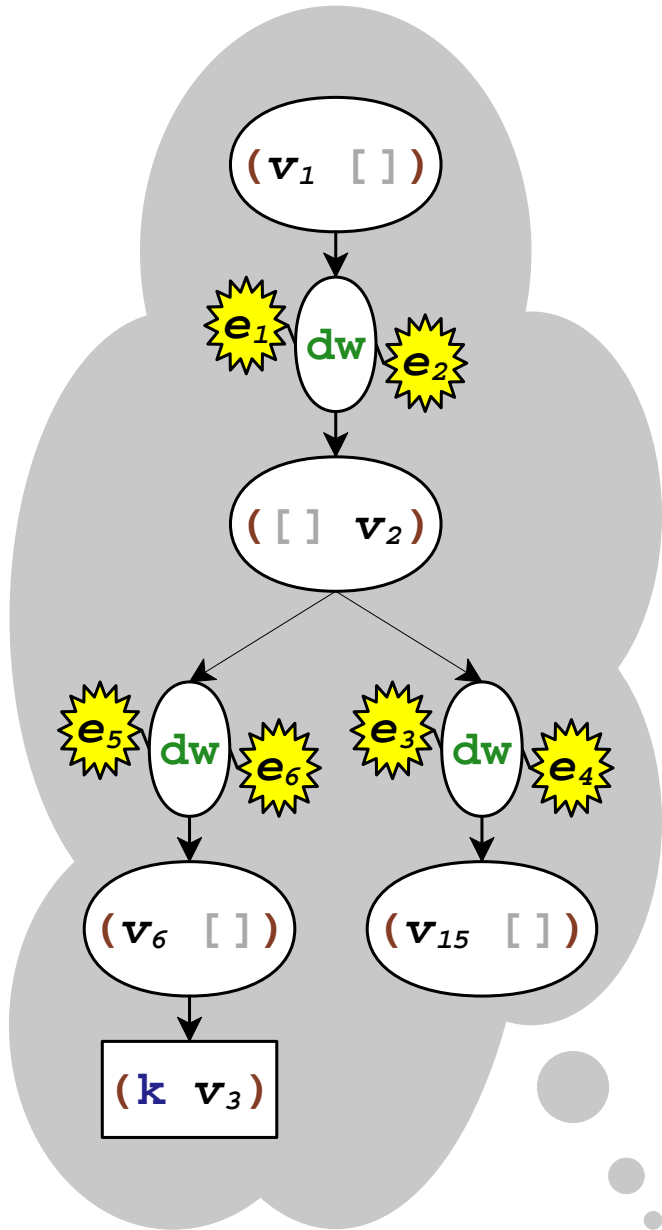
# Dynamic Wind and Call/cc



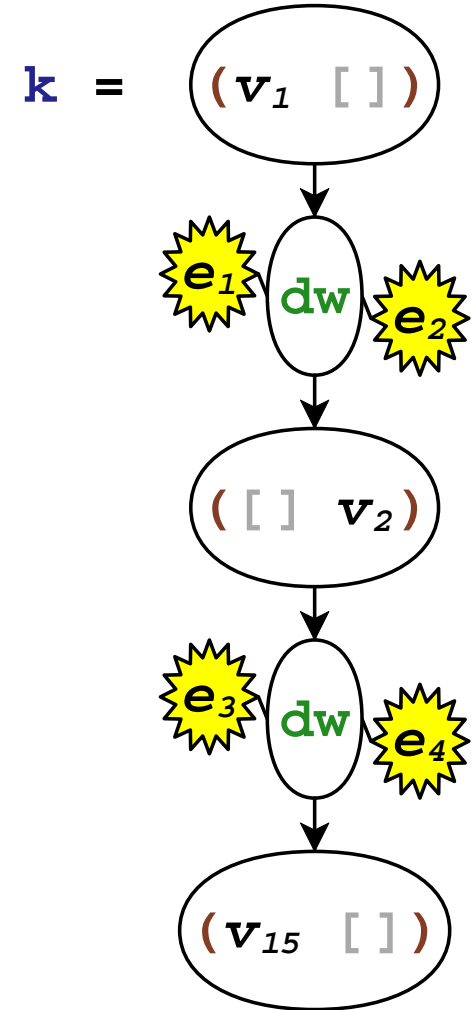
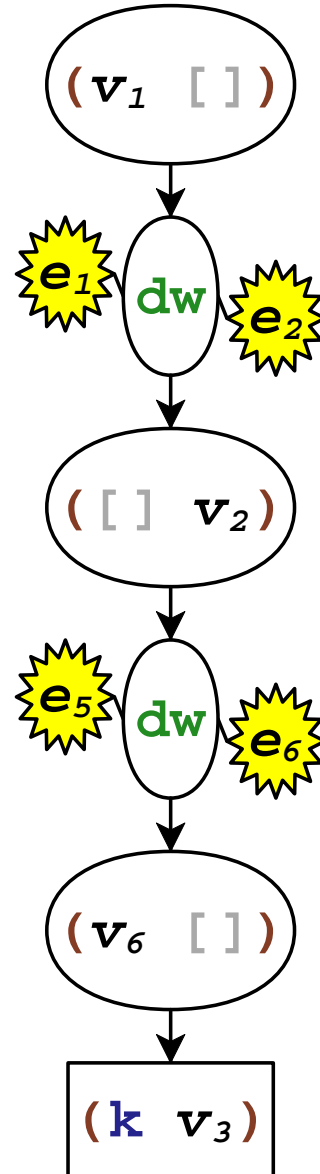
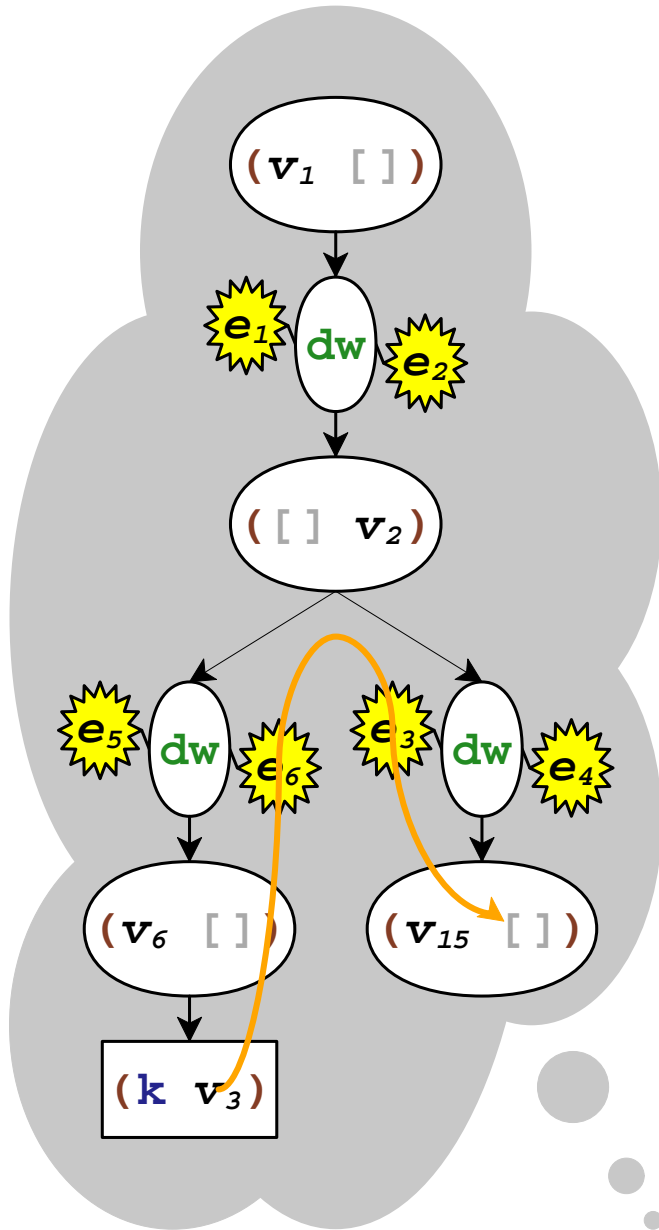
# Dynamic Wind and Call/cc



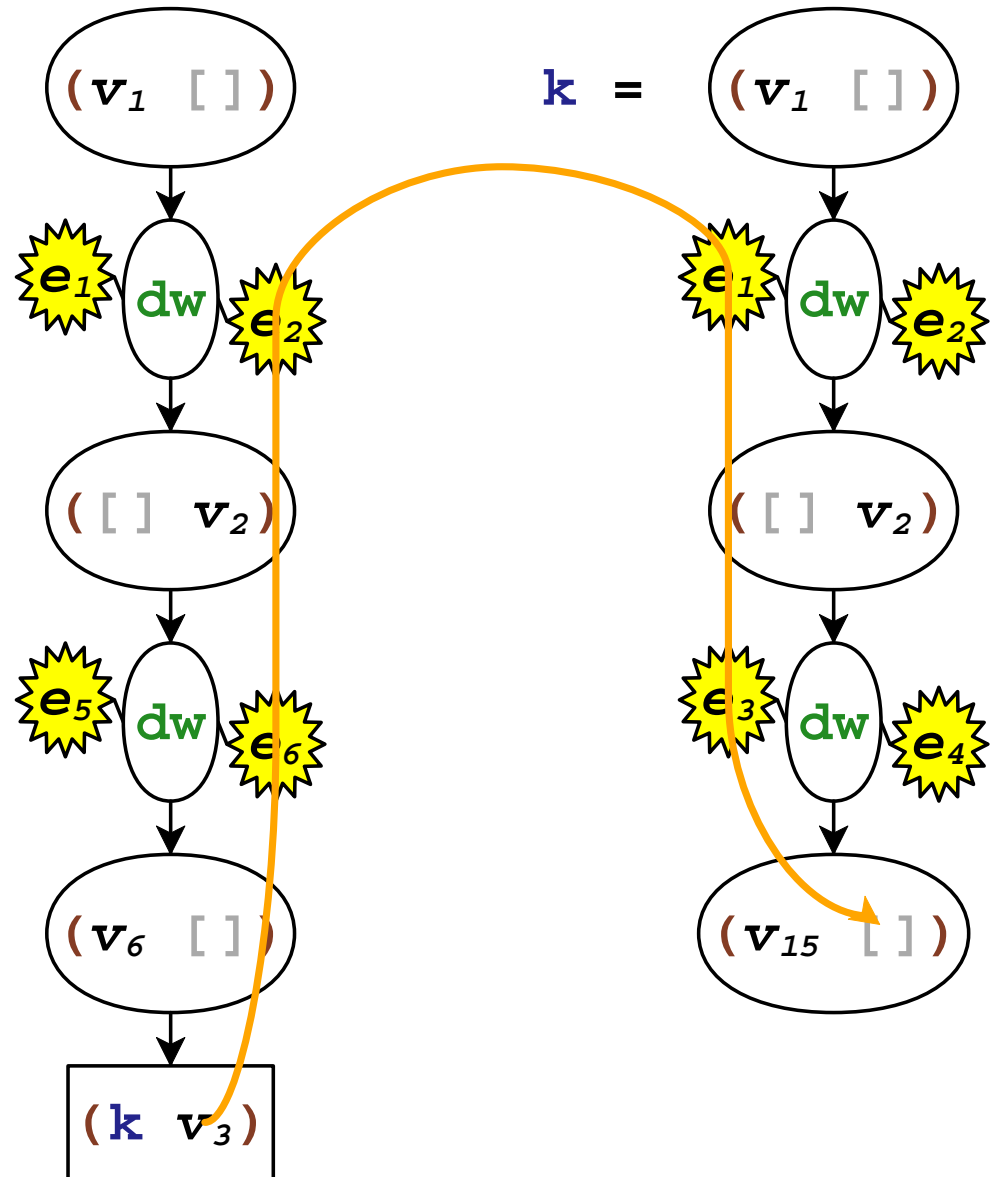
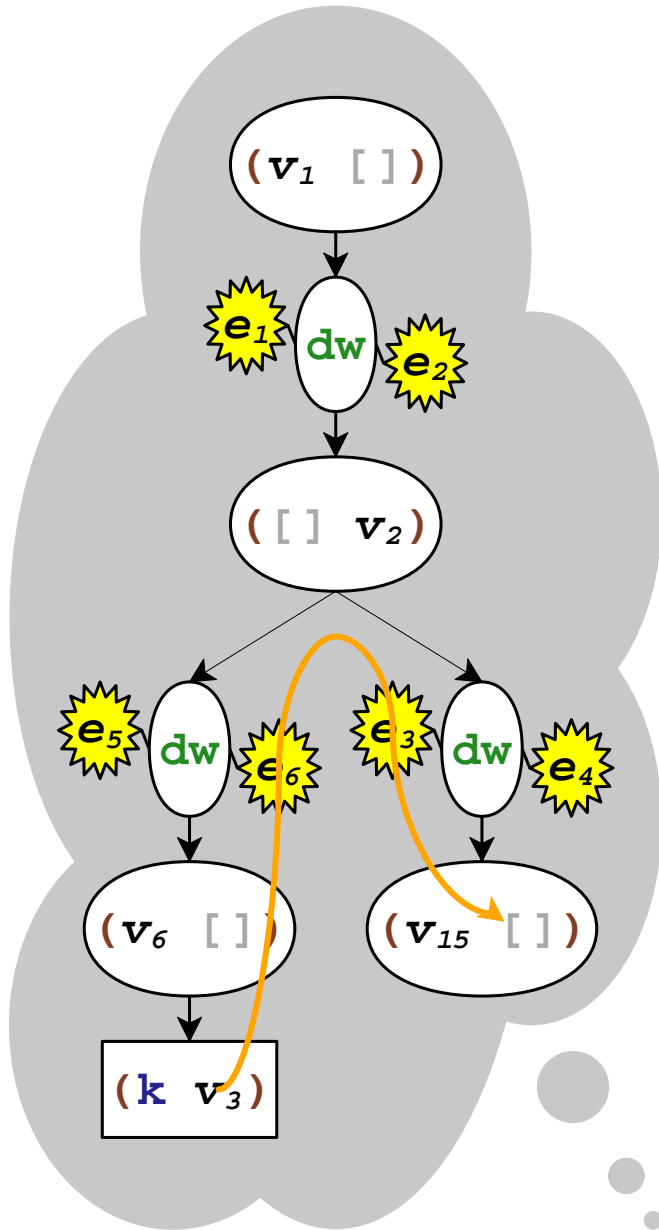
# Dynamic Wind and Call/cc



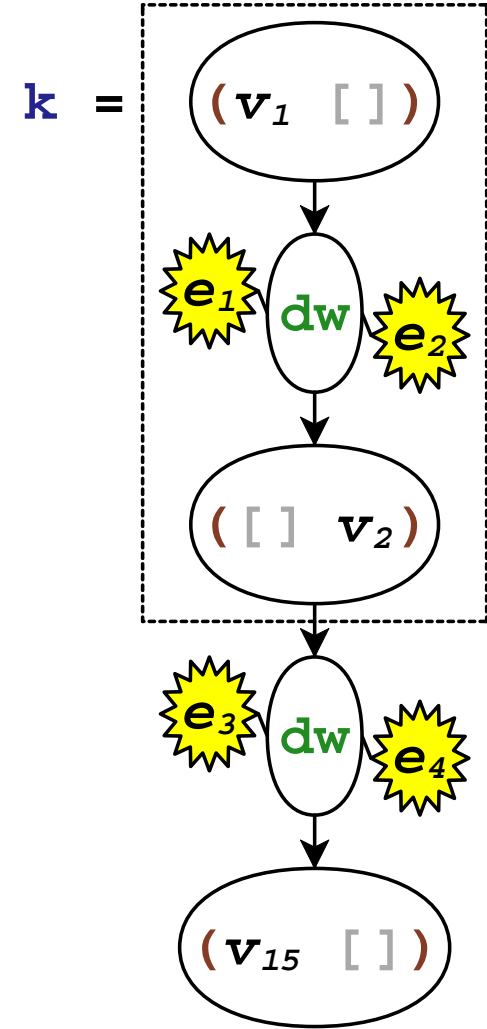
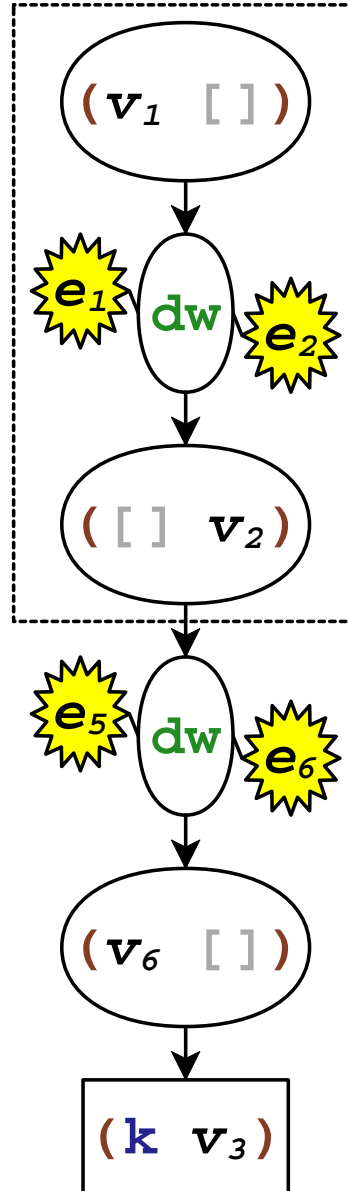
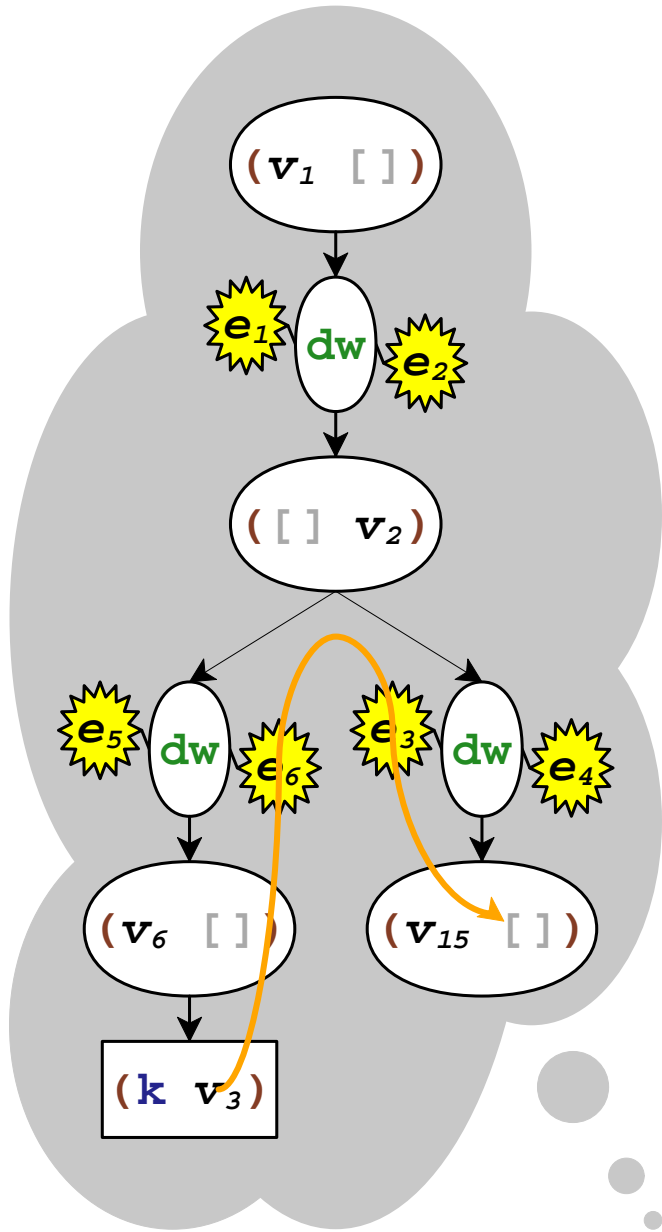
# Dynamic Wind and Call/cc



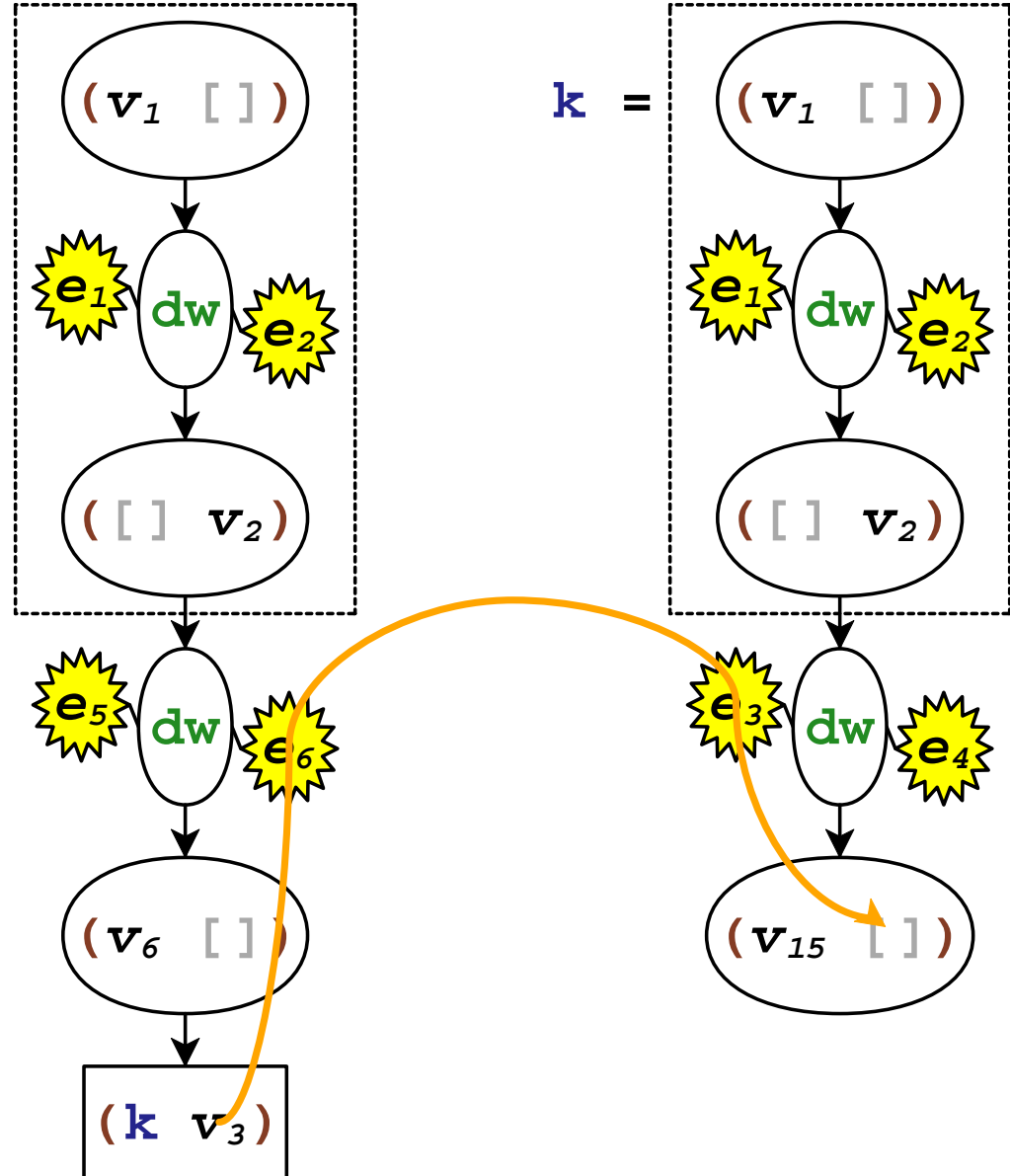
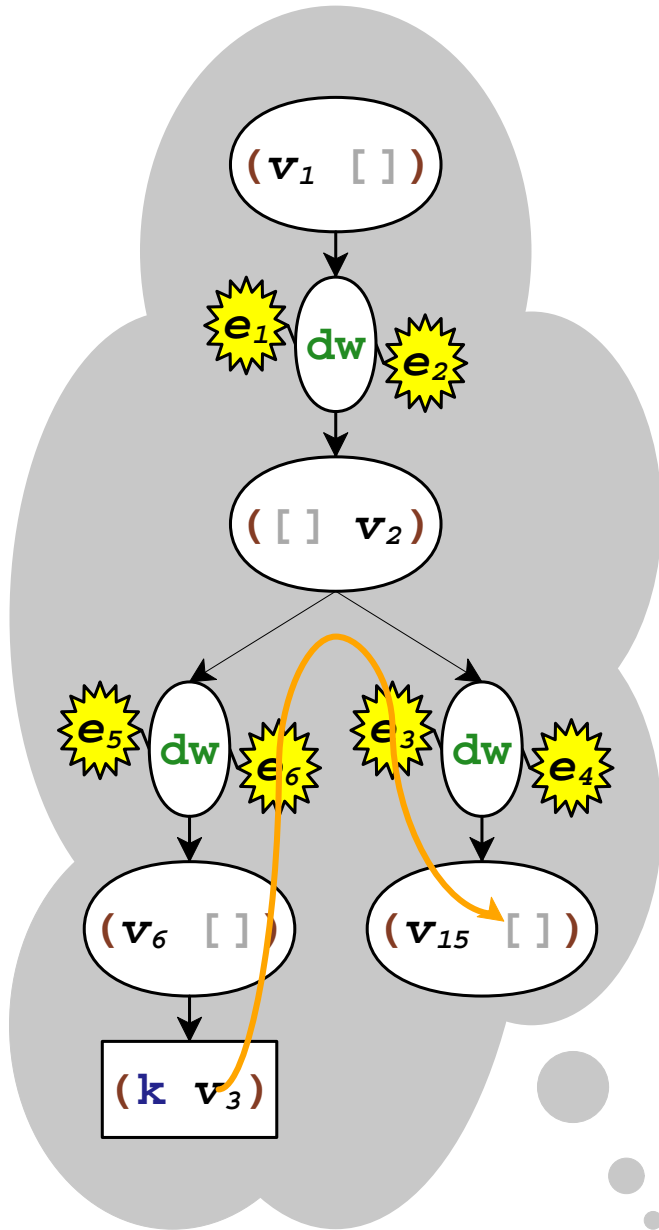
# Dynamic Wind and Call/cc



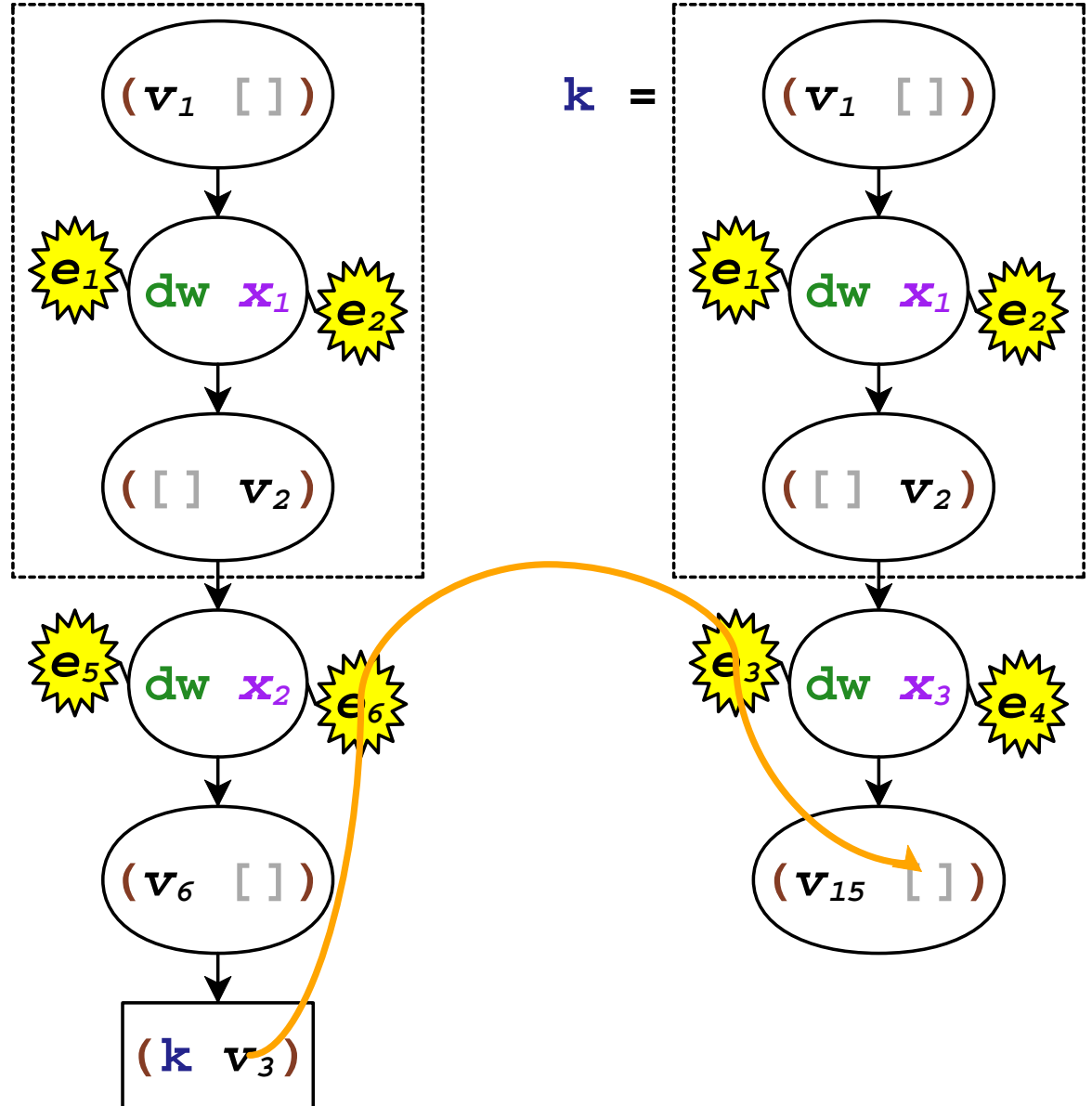
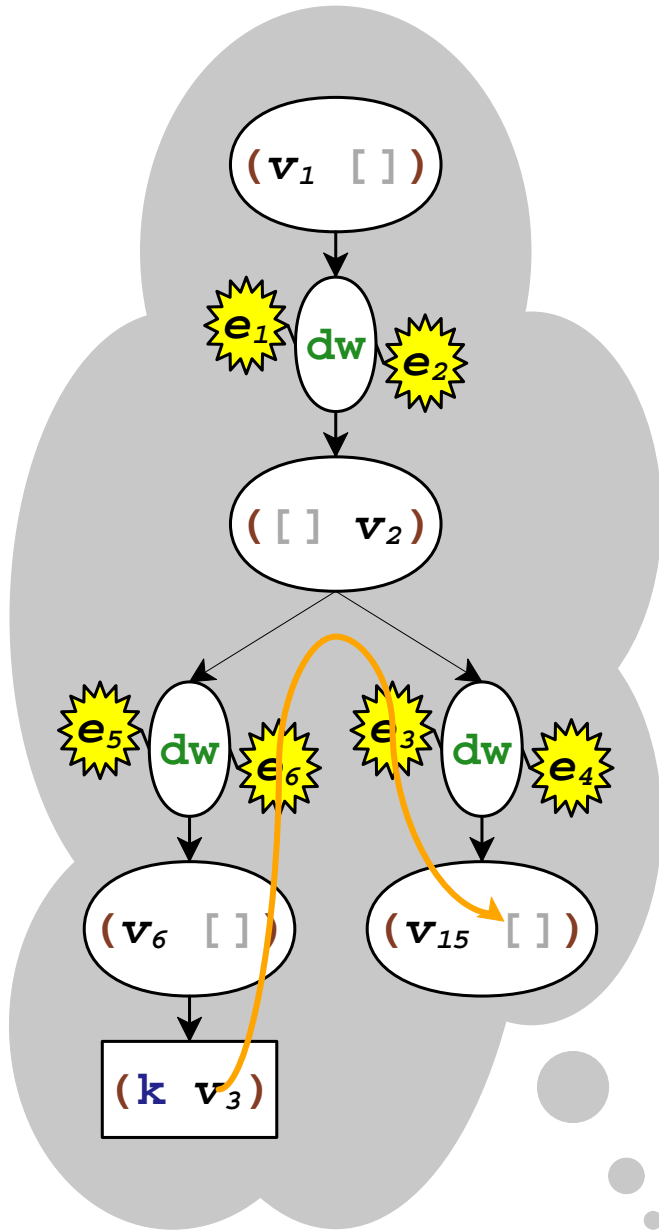
# Dynamic Wind and Call/cc



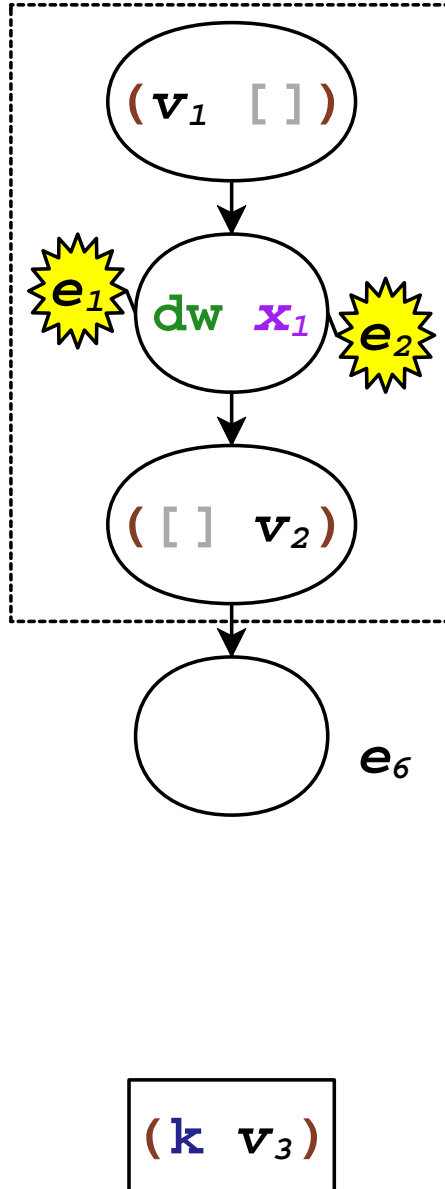
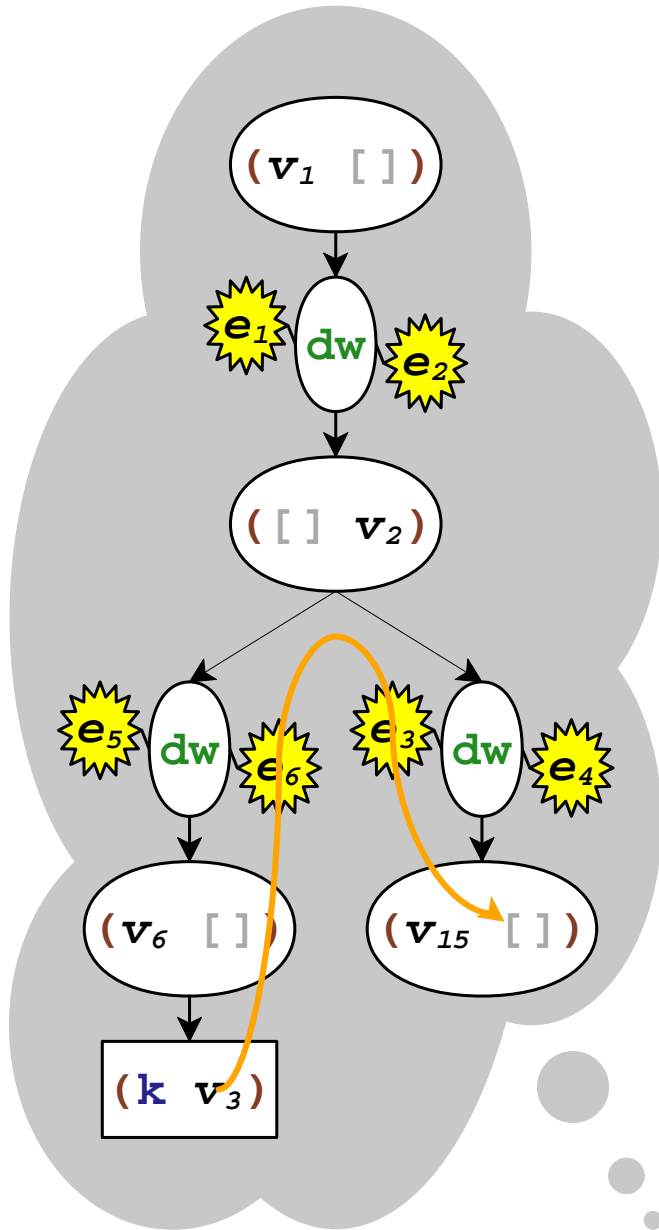
# Dynamic Wind and Call/cc



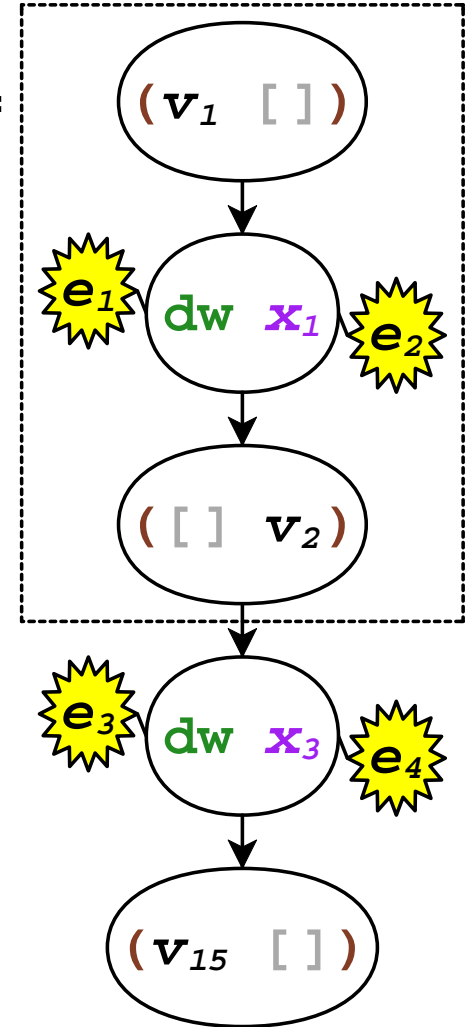
# Dynamic Wind and Call/cc



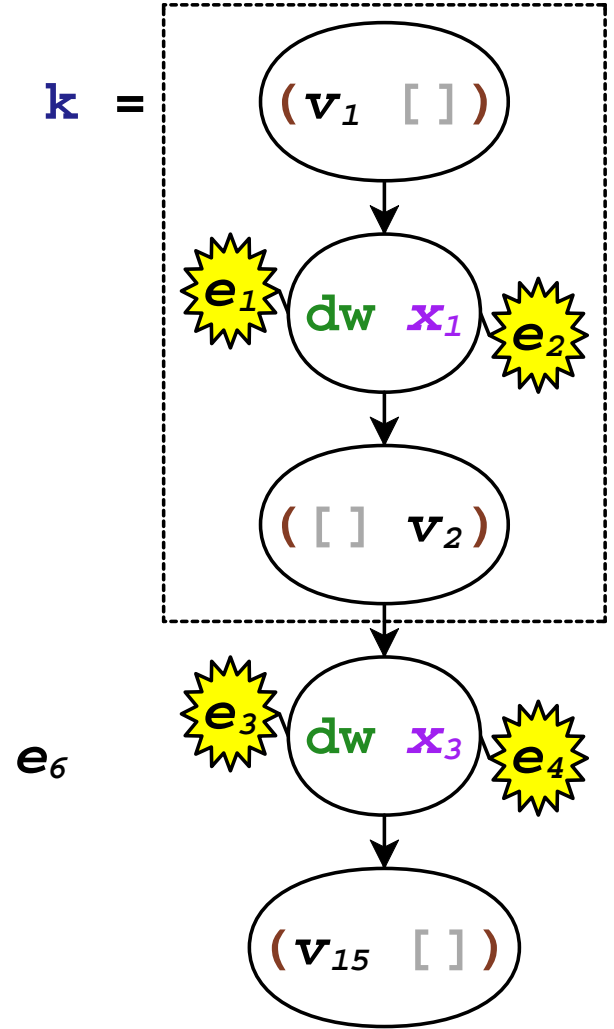
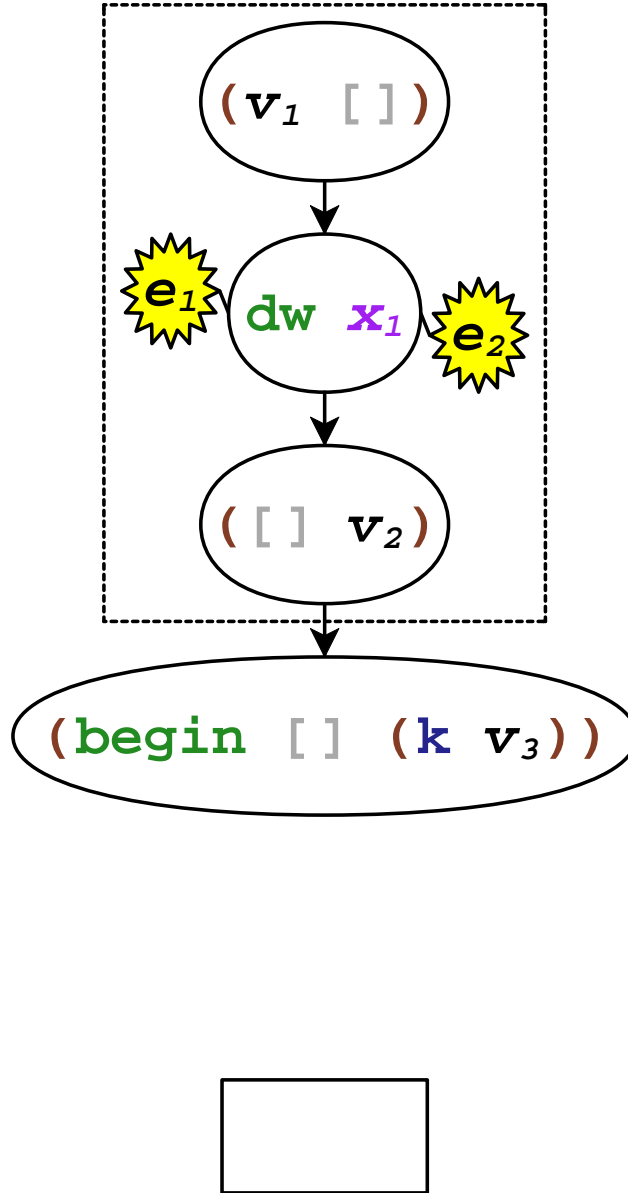
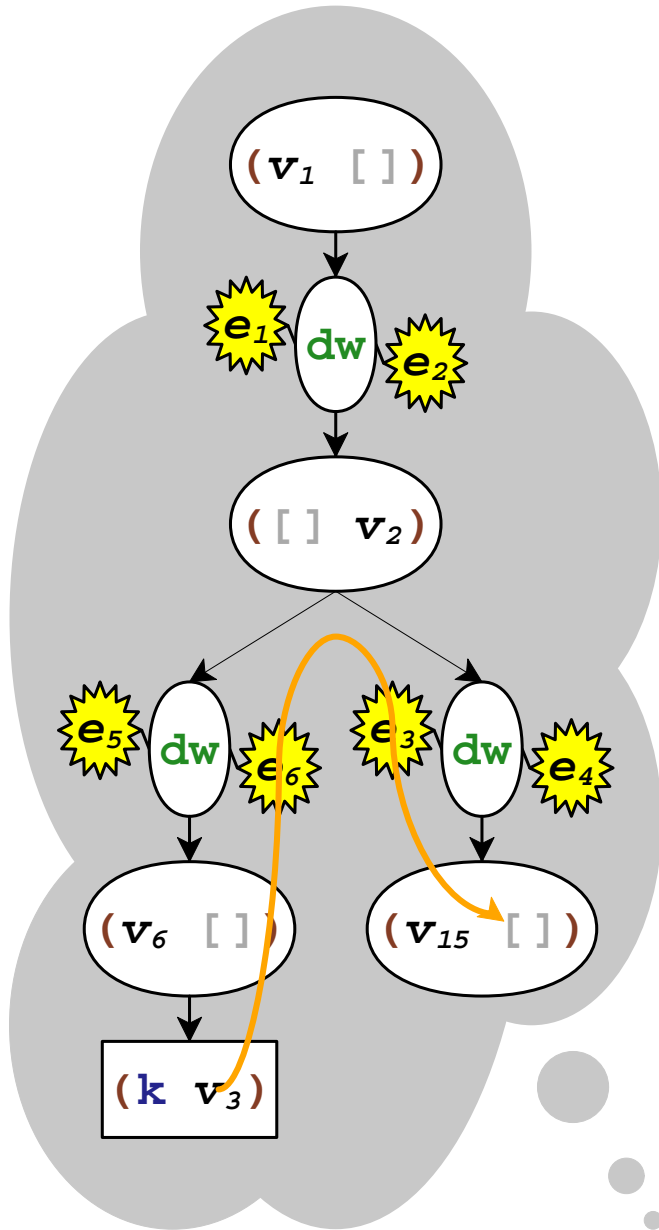
# Dynamic Wind and Call/cc



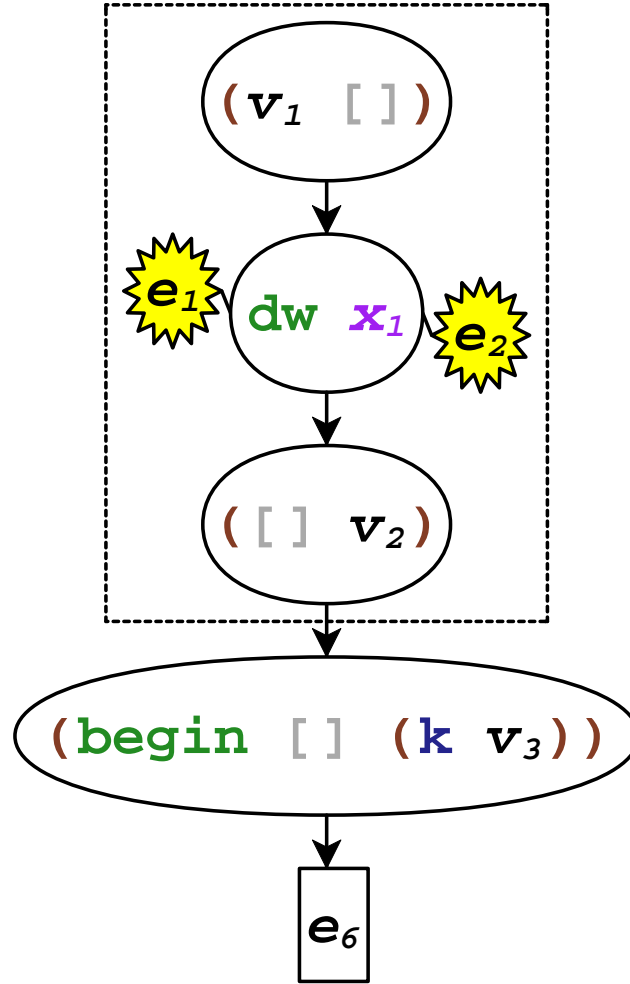
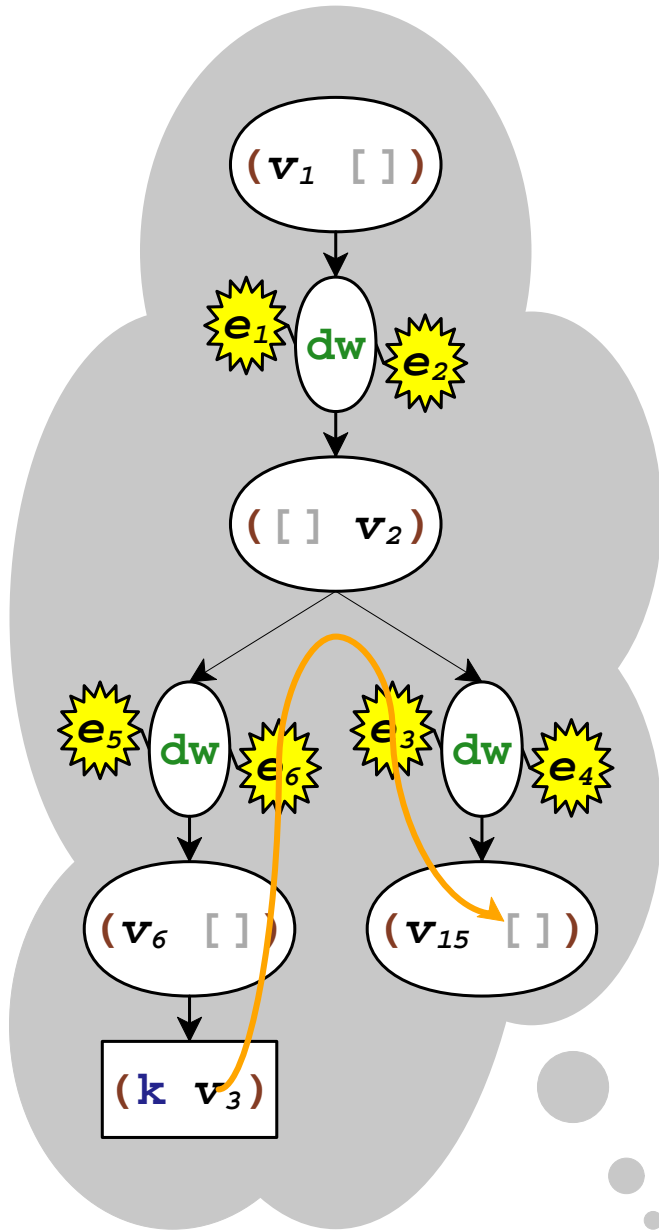
$k =$



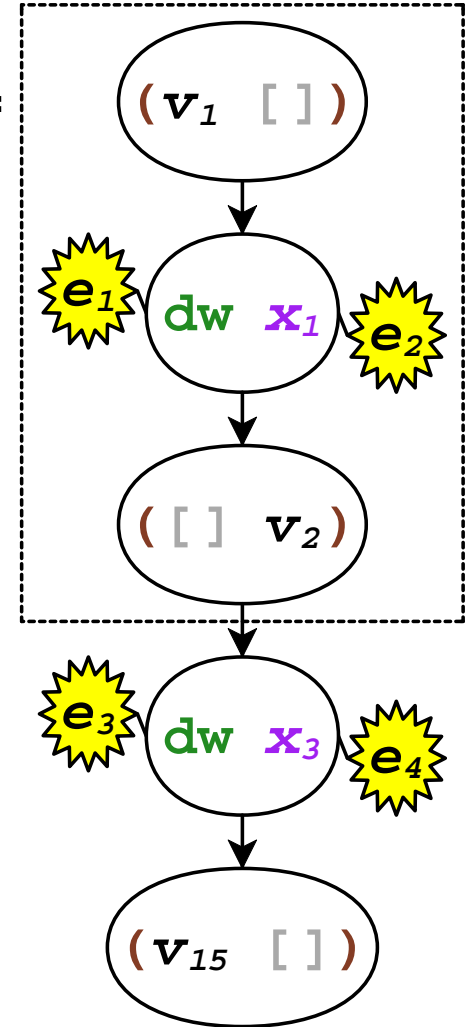
# Dynamic Wind and Call/cc



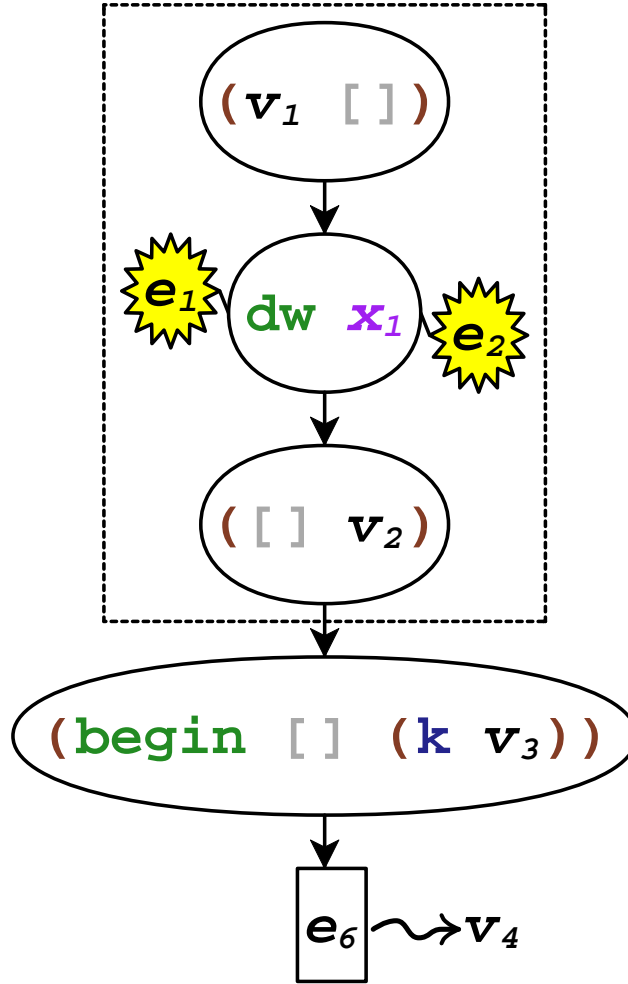
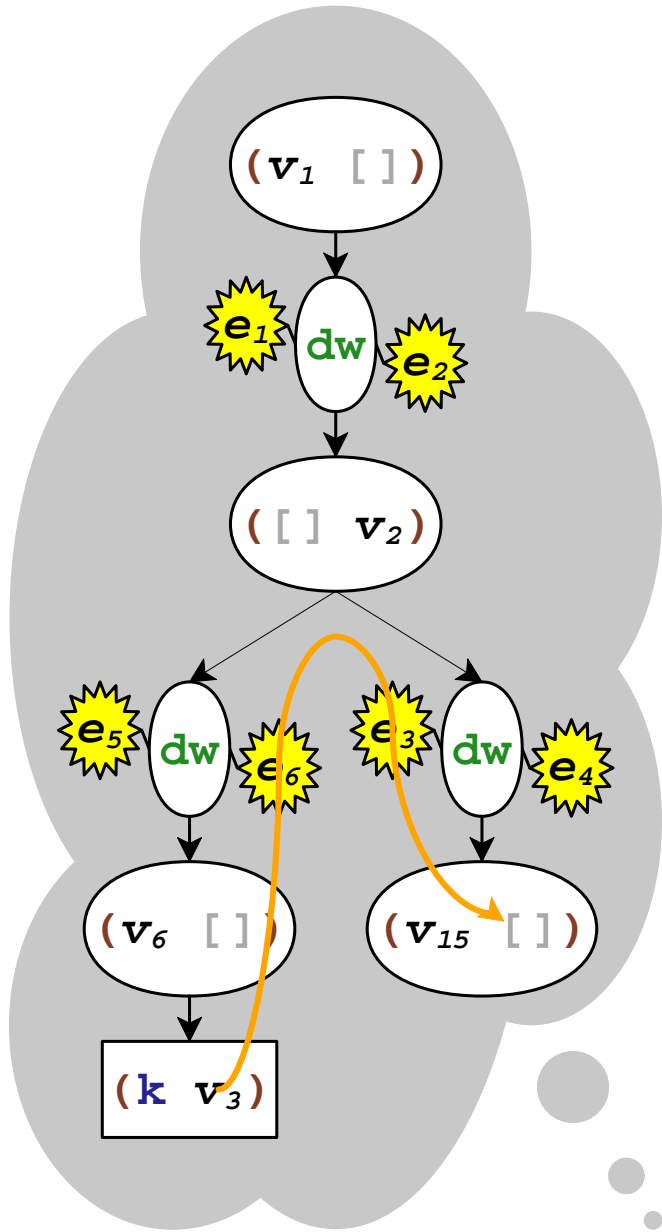
# Dynamic Wind and Call/cc



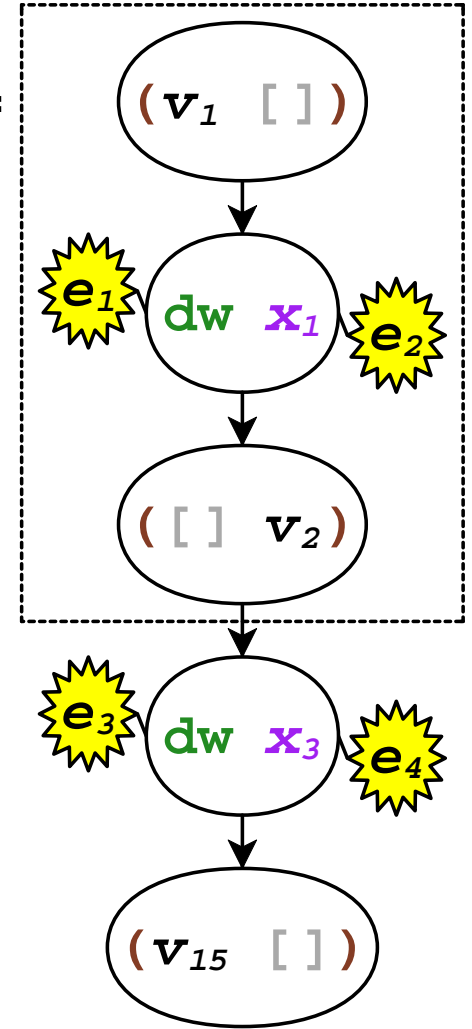
$k =$



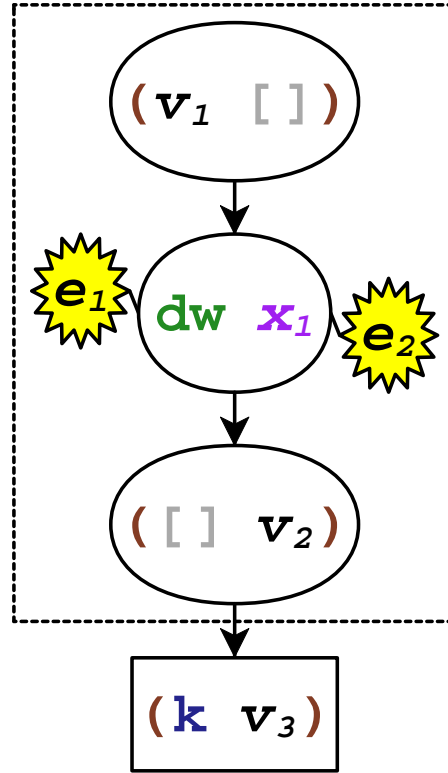
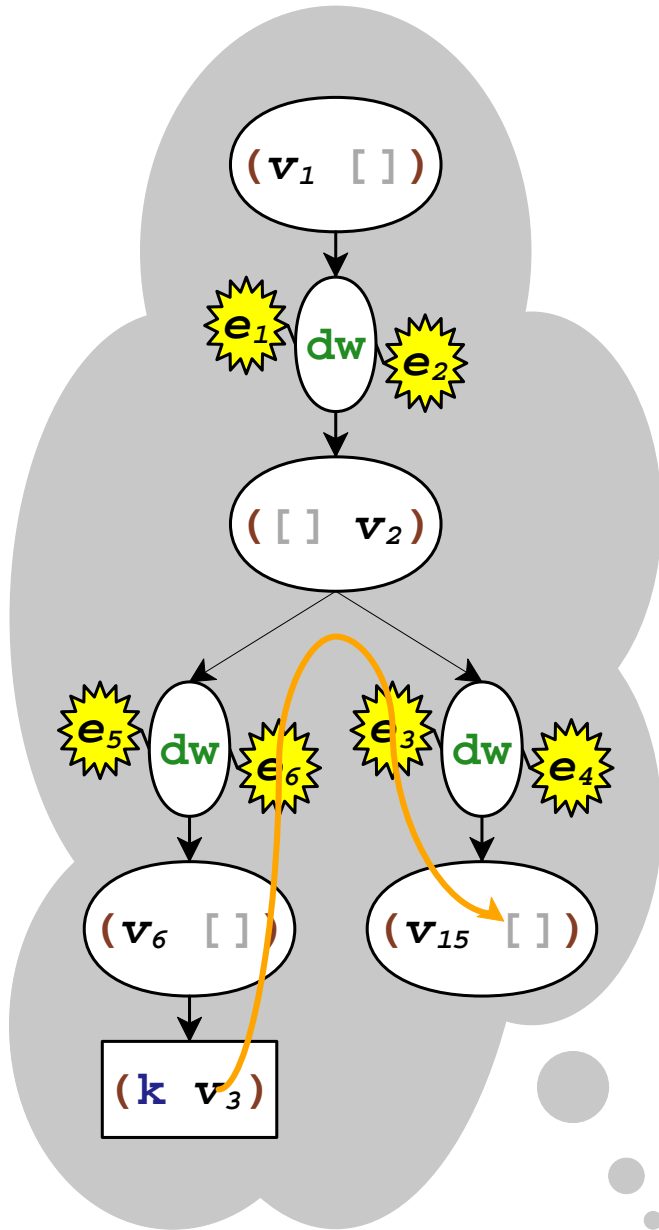
# Dynamic Wind and Call/cc



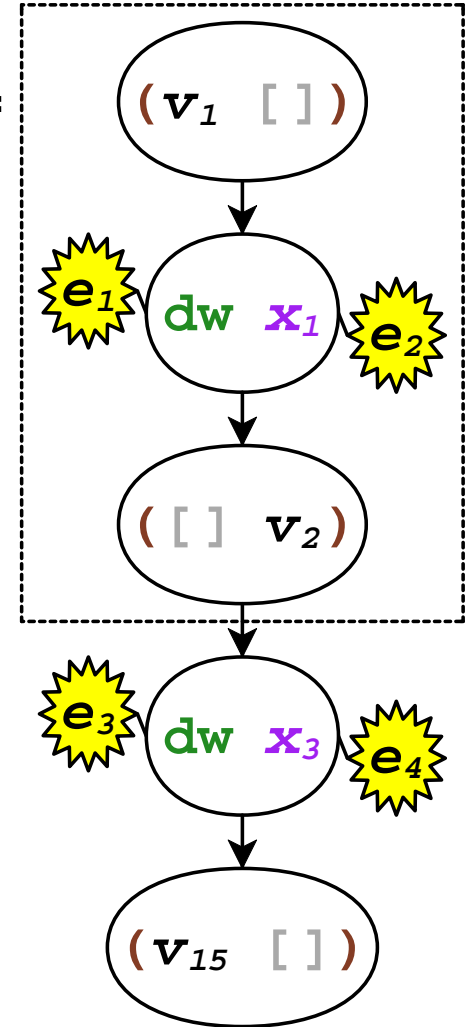
$k =$



# Dynamic Wind and Call/cc



$k =$



# Dynamic Wind Summary

- `dynamic-wind` generates `dw`
- `call/cc` detects sharing in continuation jumps
- capture `dw` thunks in `call/comp` and `call/cc`
- run post thunks in `abort`
- run pre thunks in continuation composition

## To Continue...

- Read the paper

- Run the Redex model:

`http://www.cs.utah.edu/plt/delim-cont/`

- Download the implementation:

`http://www.plt-scheme.org/`

```
(web-k ' ("control"  
         "production"  
         "environment" ) )
```

# Web Server with Prompt

`(reply out2 [])`



`(prompt [])`



```
(let ([terms '("control"
               "production"
               "environment")])
  (find-paper terms))
```

web-k = `(let ([terms []])  
 (find-paper terms))`