Evaluating Call-by-need on the Control Stack

Stephen Chang, David Van Horn, Matthias Felleisen

Northeastern University

Sharing implemented with:
heap

Sharing implemented with:

~~heap~~

stack operations

(alternative approach)

Sharing implemented with:
~~heap~~
stack operations
(alternative approach)

[Garcia et al. 2009]

Our Paper

- New way to resolve variable references in the stack

Our Paper

- New way to resolve variable references in the stack
- Reorganize stack structure to allow indexing

Call-by-need λ-Calculus

[Ariola et al. 1995]
[Ariola and Felleisen 1997]

Call-by-need λ-Calculus

[Ariola et al. 1995]
[Ariola and Felleisen 1997]

- Delay evaluation of argument until needed

Call-by-need λ-Calculus

[Ariola et al. 1995]
[Ariola and Felleisen 1997]

- Delay evaluation of argument until needed
- Evaluate each argument only once

Call-by-need λ-Calculus

[Ariola et al. 1995]
[Ariola and Felleisen 1997]

- Delay evaluation of argument until needed
- Evaluate each argument only once

```
M = x | M M | λx.M
```

Call-by-need λ-Calculus

[Ariola et al. 1995]
[Ariola and Felleisen 1997]

- Delay evaluation of argument until needed
- Evaluate each argument only once

```
M = x | M M | λx.M

E = [ ] | E M | (λx.E) M | (λx.E[x]) E
```

Call-by-need λ-Calculus

[Ariola et al. 1995]
[Ariola and Felleisen 1997]

- Delay evaluation of argument until needed
- Evaluate each argument only once

```
M = x | M M | λx.M
```

```
E = [ ] | E M | (λx.E) M | (λx.E[x]) E
```

Call-by-need λ-Calculus

[Ariola et al. 1995]
[Ariola and Felleisen 1997]

- Delay evaluation of argument until needed
- Evaluate each argument only once

```
M = x | M M | λx.M
```

```
E = [ ] | E M | (λx.E) M | (λx.E[x]) E
```

Call-by-need λ-Calculus

[Ariola et al. 1995]
[Ariola and Felleisen 1997]

- Delay evaluation of argument until needed
- Evaluate each argument only once

```
M = x | M M | λx.M
```

E = [ ] | E M | (λx.E) M | **(λx.E[x]) E**

Call-by-need λ-Calculus

[Ariola et al. 1995]
[Ariola and Felleisen 1997]

- Delay evaluation of argument until needed
- Evaluate each argument only once

```
M = x | M M | λx.M
```

```
E = [ ] | E M | (λx.E) M | (λx.E[x]) E
```

deref (β alternative):

$$(\lambda x.E[x])\ V \longrightarrow (\lambda x.E[\mathbf{V}])\ V$$

Call-by-need λ-Calculus

[Ariola et al. 1995]
[Ariola and Felleisen 1997]

- Delay evaluation of argument until needed

- Evaluate each argument only once

```
M = x | M M | λx.M

E = [ ] | E M | (λx.E) M | (λx.E[x]) E
```

deref (β alternative):

$$(\lambda x.E[x])\ V \longrightarrow (\lambda x.E[\mathbf{V}])\ V$$

- One-at-a-time substitution (only when needed)

Call-by-need λ-Calculus

[Ariola et al. 1995]
[Ariola and Felleisen 1997]

- Delay evaluation of argument until needed
- Evaluate each argument only once

```
M = x | M M | λx.M

E = [ ] | E M | (λx.E) M | (λx.E[x]) E
```

deref (β alternative):

```
(λx.E[x]) V ⟶ (λx.E[V]) V
```

- One-at-a-time substitution (only when needed)
- Argument not removed (may need it again)

An Initial Abstract Machine

An Initial Abstract Machine

Standard Reduction = abstract machine

$$\texttt{E[M]} \xrightarrow{\texttt{SR}} \texttt{E[N]}$$

$$\text{if } \texttt{M} \longrightarrow \texttt{N}$$

An Initial Abstract Machine

Standard Reduction = abstract machine

$$\texttt{E[M]} \xrightarrow{\texttt{SR}} \texttt{E[N]}$$

$$\text{if } \texttt{M} \longrightarrow \texttt{N}$$

- Re-partition into E and M after every reduction

CK Machine

[Felleisen 1986]

(For by-value λ calculus)

- Separate program into two registers:
  - `C` = Current subterm being evaluated
  - `K` = Continuation (equiv. to eval. context)

CK Machine

[Felleisen 1986]

(For by-value λ calculus)

- Separate program into two registers:
  - `C` = Current subterm being evaluated
  - `K` = Continuation (equiv. to eval. context)

Don't need to re-partition program after every reduction

CK Machine

[Felleisen 1986]

(For by-value λ calculus)

- Separate program into two registers:
  - `C` = Current subterm being evaluated
  - `K` = Continuation (equiv. to eval. context)

Don't need to re-partition program after every reduction

[Garcia et al. 2009]: lazy CK machine

Evaluation Contexts (E) vs Continuations (K)

```
[ ] ~ mt

E[[ ] M] ~ (arg M K)
      E ~ K

E[(λx.[ ]) M] ~ (bind x M K)
      E ~ K

E[(λx.E'[x]) [ ]] ~ (op x K' K)
      K' ~ E', K ~ E
```

Evaluation Contexts (E) vs Continuations (K)

```
[ ] ~ mt
```

```
E[[ ] M] ~ (arg M K)
      E ~ K
```

```
E[(λx.[ ]) M] ~ (bind x M K)
          E ~ K
```

```
E[(λx.E'[x]) [ ]] ~ (op x K' K)
      K' ~ E', K ~ E
```

Evaluation Contexts (E) vs Continuations (K)

```
[ ] ~ mt
```

```
E[[ ] M] ~ (arg M K)
     E ~ K
```

```
E[(λx.[ ]) M] ~ (bind x M K)
          E ~ K
```

```
E[(λx.E'[x]) [ ]] ~ (op x K' K)
      K' ~ E', K ~ E
```

Evaluation Contexts (E) vs Continuations (K)

```
[ ] ~ mt
```

```
E[[ ] M] ~ (arg M K)
      E ~ K
```

`E[(λx.[ ]) M] ~ (bind x M K)`

```
E ~ K
```

```
E[(λx.E'[x]) [ ]] ~ (op x K' K)
     K' ~ E', K ~ E
```

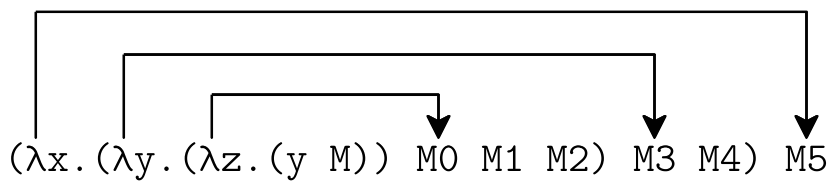
Evaluation Contexts (E) vs Continuations (K)

```
[ ] ~ mt

E[[ ] M] ~ (arg M K)
     E ~ K

E[(λx.[ ]) M] ~ (bind x M K)
            E ~ K

E[(λx.E'[x]) [ ]] ~ (op x K' K)
        K' ~ E', K ~ E
```

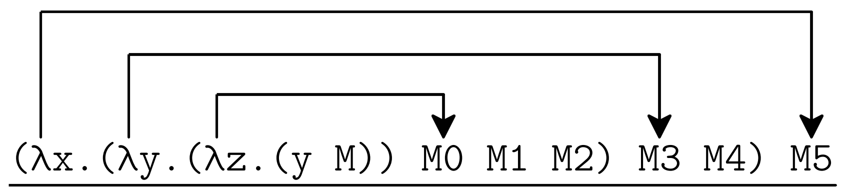
Example (Garcia Machine)

```
(λx.(λy.(λz.(y M)) M0 M1 M2) M3 M4) M5
```

Example (Garcia Machine)

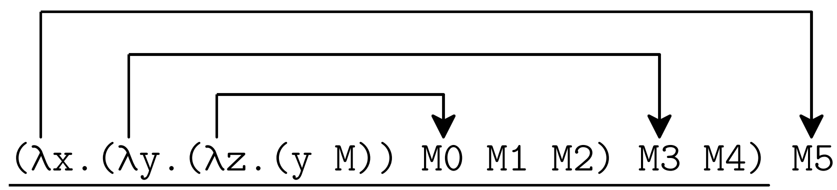


Example (Garcia Machine)



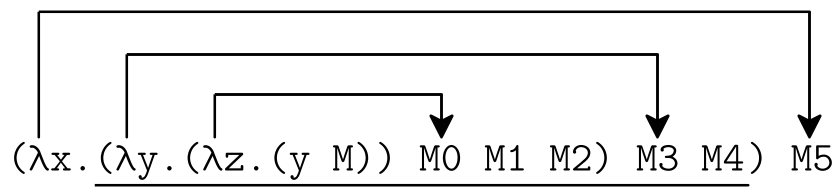

```
C = (λx.(λy.(λz.(y M)) M0 M1 M2) M3 M4) M5
K = mt
```

Example (Garcia Machine)



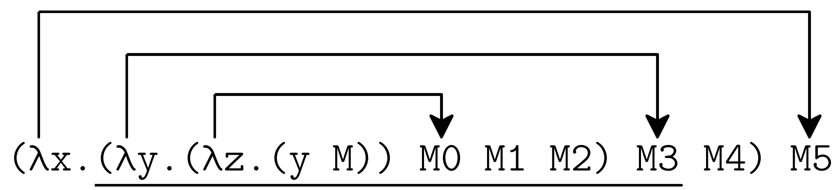
C = (λx.(λy.(λz.(y M)) M0 M1 M2) M3 M4)
K = (arg M5) ➡ mt

Example (Garcia Machine)



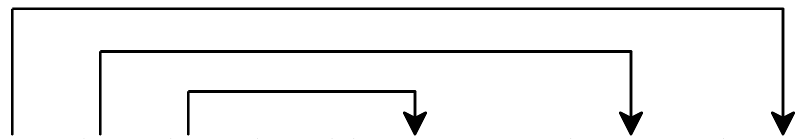

```
C = (λy.(λz.(y M)) M0 M1 M2) M3 M4
K = (bind x M5) ➡ mt
```

Example (Garcia Machine)




```
C = (λy.(λz.(y M)) M0 M1 M2) M3
K = (arg M4) ➡ (bind x M5) ➡ mt
```

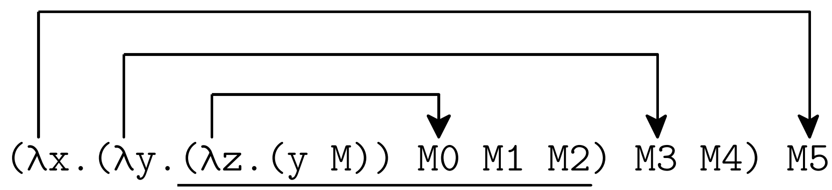
Example (Garcia Machine)



(λx.(λy.(λz.(y M)) M0 M1 M2) M3 M4) M5

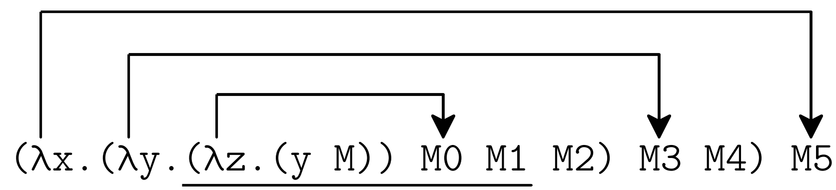
C = (λy.(λz.(y M)) M0 M1 M2)
K = (arg M3) ➡ (arg M4) ➡ (bind x M5) ➡ mt

Example (Garcia Machine)



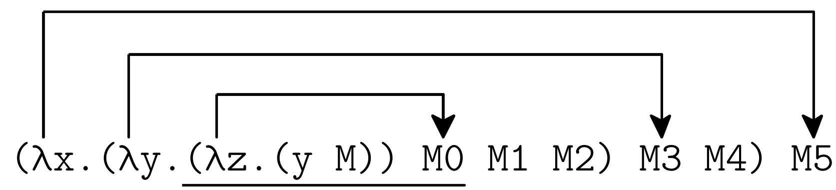

```
C  =  (λz.(y M)) M0 M1 M2
K  =  (bind y M3) ➡ (arg M4) ➡ (bind x M5) ➡ mt
```

Example (Garcia Machine)



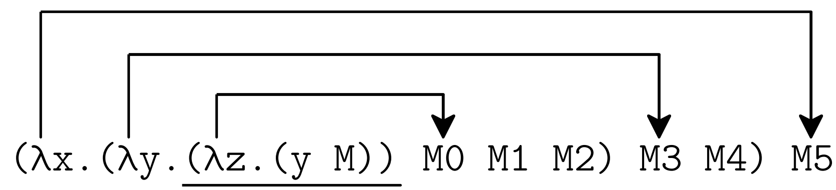

```
C = (λz.(y M)) M0 M1
K = (arg M2) ➡ (bind y M3) ➡ (arg M4) ➡ (bind x M5) ➡ mt
```

Example (Garcia Machine)



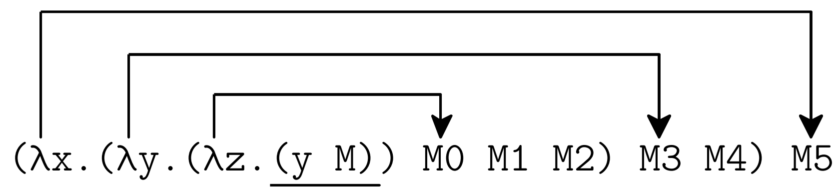

```
C  =  (λz.(y M)) M0
K  =  (arg M1) ➡ (arg M2) ➡ (bind y M3) ➡ (arg M4) ➡ (bind x M5) ➡ mt
```

Example (Garcia Machine)



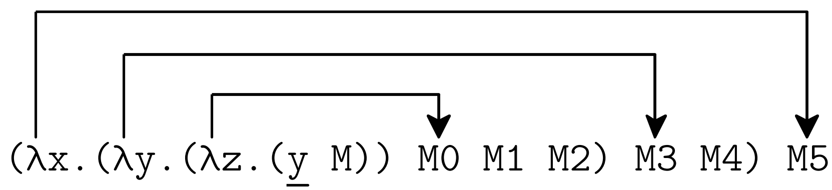

```
C = (λz.(y M))
K = (arg M0) ➡ (arg M1) ➡ (arg M2) ➡ (bind y M3) ➡ (arg M4) ➡ (bind x M5) ➡ mt
```

Example (Garcia Machine)



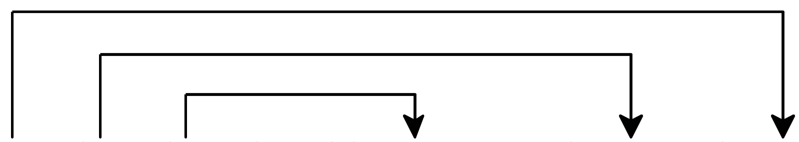
C = (y M)
K = (bind z M0) ➡ (arg M1) ➡ (arg M2) ➡ (bind y M3) ➡ (arg M4) ➡ (bind x M5) ➡ mt

Example (Garcia Machine)



C = y
K = (arg M) ➨ (bind z M0) ➨ (arg M1) ➨ (arg M2) ➨ (bind y M3) ➨ (arg M4) ➨ (bind x M5) ➨ mt

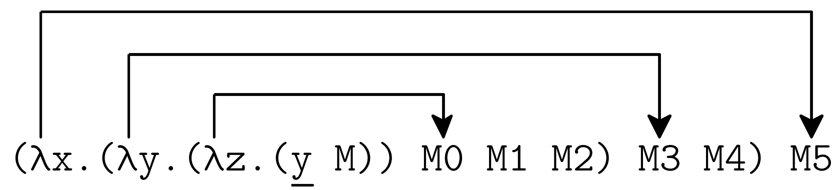
Example (Garcia Machine)



```
(λx.(λy.(λz.(y M)) M0 M1 M2) M3 M4) M5
```

C = y
K = **(arg M)** ➡ (bind z M0) ➡ (arg M1) ➡ (arg M2) ➡ (bind y M3) ➡ (arg M4) ➡ (bind x M5) ➡ mt

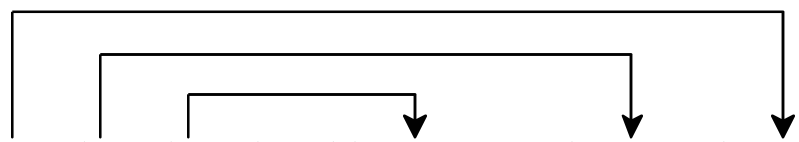
Example (Garcia Machine)



C = y

K = (arg M) ➡ **(bind z M0)** ➡ (arg M1) ➡ (arg M2) ➡ (bind y M3) ➡ (arg M4) ➡ (bind x M5) ➡ mt

Example (Garcia Machine)

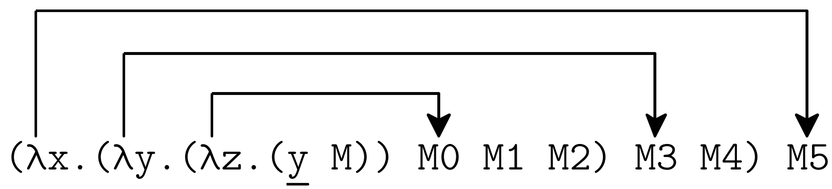


`(λx.(λy.(λz.(y M)) M0 M1 M2) M3 M4) M5`

C = y

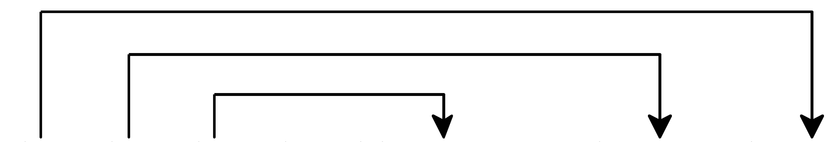
K = (arg M) ➡ (bind z M0) ➡ **(arg M1)** ➡ (arg M2) ➡ (bind y M3) ➡ (arg M4) ➡ (bind x M5) ➡ mt

Example (Garcia Machine)



C = y
K = (arg M) ➡ (bind z M0) ➡ (arg M1) ➡ **(arg M2)** ➡ (bind y M3) ➡ (arg M4) ➡ (bind x M5) ➡ mt

Example (Garcia Machine)

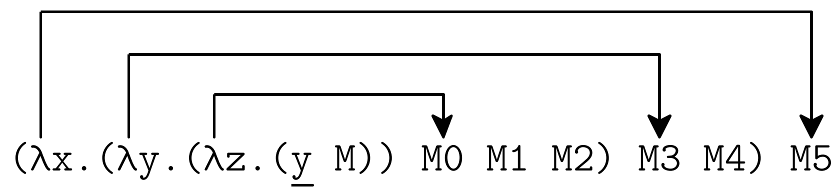


```
(λx.(λy.(λz.(y M)) M0 M1 M2) M3 M4) M5
```

C = y

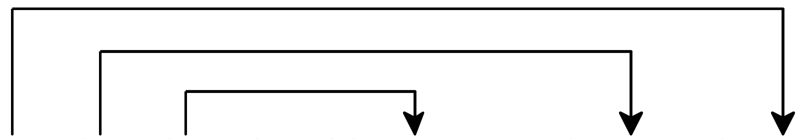
K = (arg M) ➡ (bind z M0) ➡ (arg M1) ➡ (arg M2) ➡ **(bind y M3)** ➡ (arg M4) ➡ (bind x M5) ➡ mt

Example (Garcia Machine)



C = y
K = (arg M) ➡ (bind z M0) ➡ (arg M1) ➡ (arg M2) ➡ **(bind y M3)** ➡ (arg M4) ➡ (bind x M5) ➡ mt

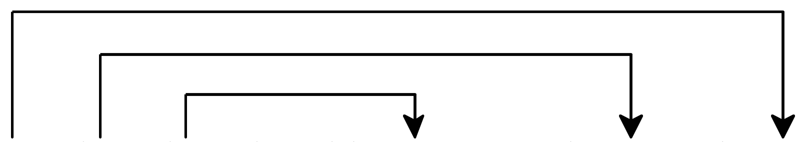
Example (Garcia Machine)



(λx.(λy.(λz.(y M)) M0 M1 M2) M3 M4) M5

C = y
K = (arg M) ➧ (bind z M0) ➧ (arg M1) ➧ (arg M2) ➧ **(bind y M3)** ➧ (arg M4) ➧ (bind x M5) ➧ mt

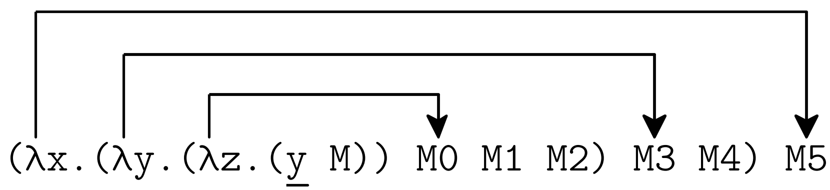
Example (Garcia Machine)



(λx.(λy.(λz.(y M)) M0 M1 M2) M3 M4) M5

C = y
K = (arg M) ➡ (bind z M0) ➡ (arg M1) ➡ (arg M2) ➡ **(bind y M3)** ➡ (arg M4) ➡ (bind x M5) ➡ mt

Example (Garcia Machine)



`C` = y

`K` = (arg M) ➡ (bind z M0) ➡ (arg M1) ➡ (arg M2) ➡ **(bind y M3)** ➡ (arg M4) ➡ (bind x M5) ➡ mt

- Linear search to find argument

CK+ Machine: Stack Structure

- Reorganize stack to be *stack of stacks*
  - `bind` continuations on top

CK+ Machine: Stack Structure

- Reorganize stack to be *stack of stacks*
  - `bind` continuations on top

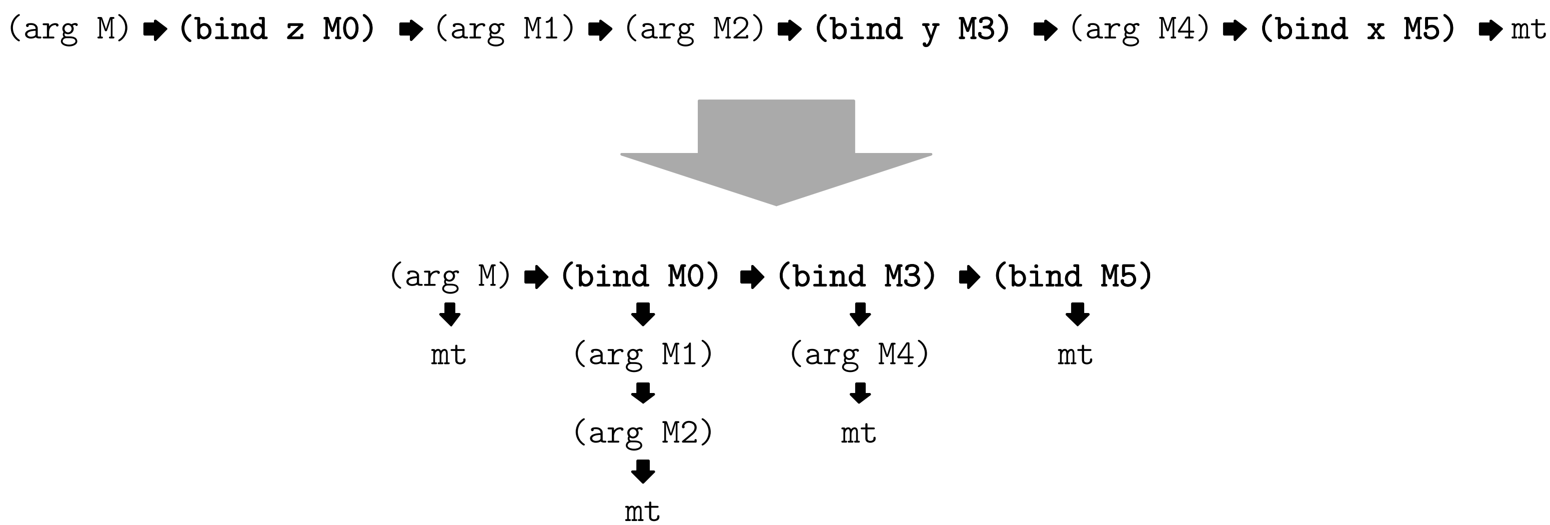
`(arg M)` ➡ `(bind z M0)` ➡ `(arg M1)` ➡ `(arg M2)` ➡ `(bind y M3)` ➡ `(arg M4)` ➡ `(bind x M5)` ➡ `mt`

CK+ Machine: Stack Structure

- Reorganize stack to be *stack of stacks*
  - `bind` continuations on top

`(arg M)` ➡ **`(bind z M0)`** ➡ `(arg M1)` ➡ `(arg M2)` ➡ **`(bind y M3)`** ➡ `(arg M4)` ➡ **`(bind x M5)`** ➡ `mt`

- Reorganize stack to be *stack of stacks*
  - `bind` continuations on top



CK+ Machine: Lexical Addresses

- Replace variables with lexical addresses

[De Bruijn 1972]

CK+ Machine: Lexical Addresses

- Replace variables with lexical addresses

[De Bruijn 1972]

```
M = x | M M | λx.M
```

CK+ Machine: Lexical Addresses

- Replace variables with lexical addresses

[De Bruijn 1972]

```
M = x | M M | λx.M
M = n | M M | λ.M
```

CK+ Machine: Lexical Addresses

- Replace variables with lexical addresses

[De Bruijn 1972]

```
M = x | M M | λx.M
M = n | M M | λ.M
```

```
K = mt | (arg M K) | (bind x M K) | (op x K K)
```

CK+ Machine: Lexical Addresses

- Replace variables with lexical addresses

[De Bruijn 1972]

```
M = x | M M | λx.M
M = n | M M | λ.M

K = mt | (arg M K) | (bind x M K) | (op x K K)
K = mt | (arg M K) | (bind M K) | (op K K)
```

CK+ Machine: Lexical Addresses

- Replace variables with lexical addresses

[De Bruijn 1972]

```
M = x | M M | λx.M
M = n | M M | λ.M

K = mt | (arg M K) | (bind x M K) | (op x K K)
K = mt | (arg M K) | (bind M K) | (op K K)

λx.(x λy.(x y))
```

CK+ Machine: Lexical Addresses

- Replace variables with lexical addresses

[De Bruijn 1972]

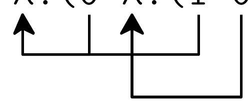
```
M = x | M M | λx.M
M = n | M M | λ.M

K = mt | (arg M K) | (bind x M K) | (op x K K)
K = mt | (arg M K) | (bind M K) | (op K K)

λx.(x λy.(x y))
λ.(0 λ.(1 0))
```

CK+ Machine: Lexical Addresses

- Replace variables with lexical addresses

[De Bruijn 1972]

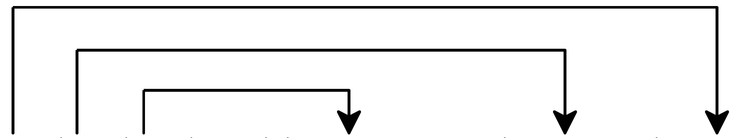
```
M = x | M M | λx.M
M = n | M M | λ.M

K = mt | (arg M K) | (bind x M K) | (op x K K)
K = mt | (arg M K) | (bind M K) | (op K K)

λx.(x λy.(x y))
λ.(0 λ.(1 0))
```



CK+ Machine: Example

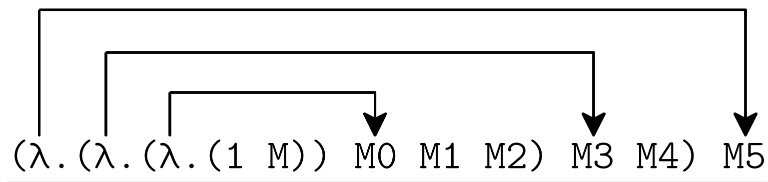
```
(λ.(λ.(λ.(1 M)) M0 M1 M2) M3 M4) M5
```

CK+ Machine: Example



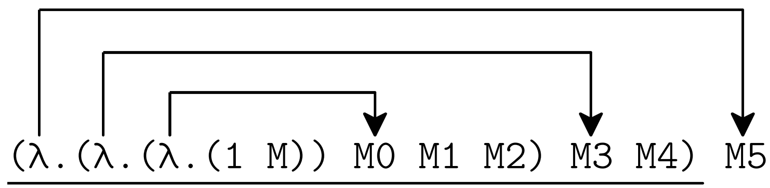
`(λ.(λ.(λ.(1 M)) M0 M1 M2) M3 M4) M5`

CK+ Machine: Example



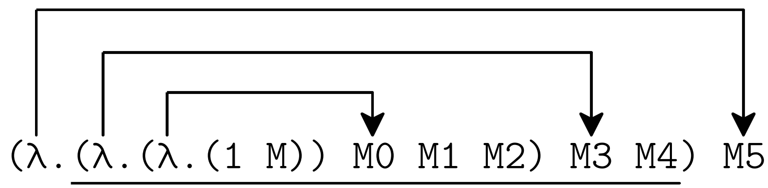
C = (λ.(λ.(λ.(1 M)) M0 M1 M2) M3 M4) M5
K = mt

CK+ Machine: Example



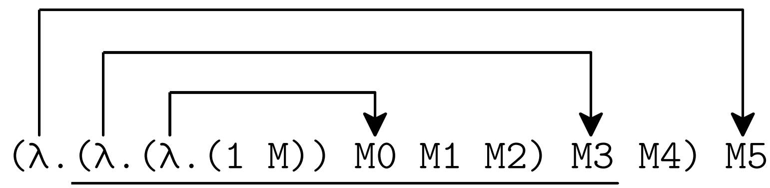

```
C = (λ.(λ.(λ.(1 M)) M0 M1 M2) M3 M4)
K = (arg M5)
        ⬇
        mt
```

CK+ Machine: Example



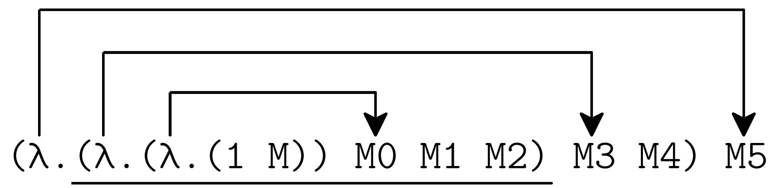

```
C = (λ.(λ.(1 M)) M0 M1 M2) M3 M4
K = mt ➡ (bind M5)
            ⬇
            mt
```

CK+ Machine: Example




```
C = (λ.(λ.(1 M)) M0 M1 M2) M3
K = (arg M4) ➡ (bind M5)
        ⬇           ⬇
        mt          mt
```

CK+ Machine: Example



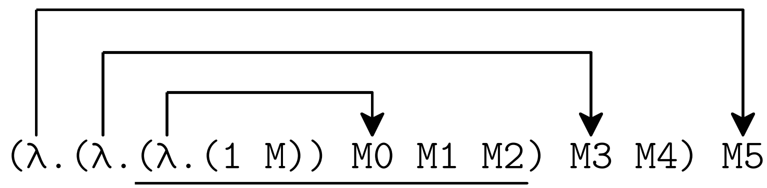
C = (λ.(λ.(1 M)) M0 M1 M2)

K = (arg M3) ➡ (bind M5)

(arg M3) ⬇ (arg M4) ⬇ mt

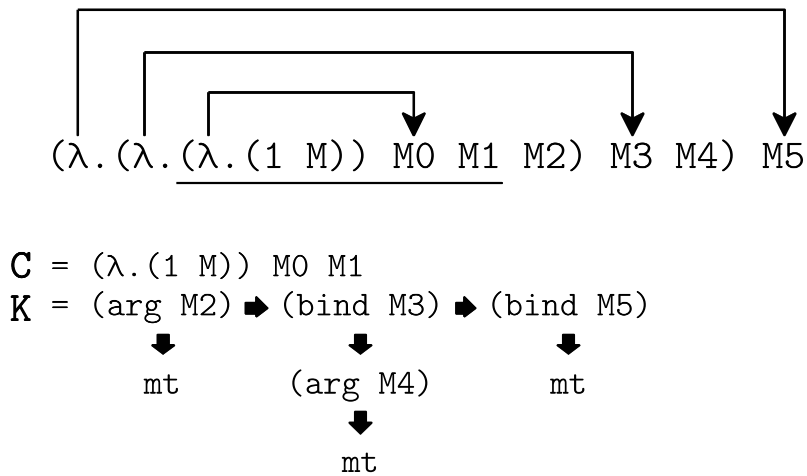
(bind M5) ⬇ mt

CK+ Machine: Example

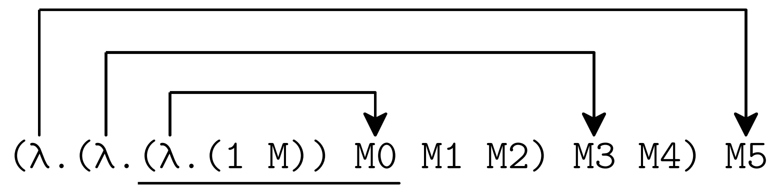



```
C = (λ.(1 M)) M0 M1 M2
K = mt ➡ (bind M3) ➡ (bind M5)
              ⬇             ⬇
          (arg M4)          mt
              ⬇
              mt
```

CK+ Machine: Example

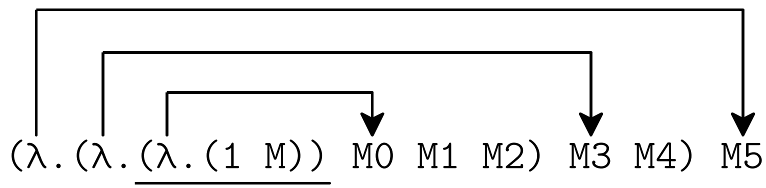


CK+ Machine: Example



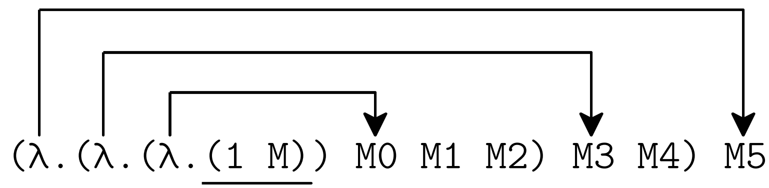

```
C = (λ.(1 M)) M0
K = (arg M1) ➡ (bind M3) ➡ (bind M5)
        ⬇            ⬇            ⬇
     (arg M2)    (arg M4)        mt
        ⬇            ⬇
        mt           mt
```

CK+ Machine: Example




```
C = (λ.(1 M))
K = (arg M0) ➡ (bind M3) ➡ (bind M5)
       ⬇           ⬇           ⬇
    (arg M1)    (arg M4)       mt
       ⬇           ⬇
    (arg M2)       mt
       ⬇
       mt
```

CK+ Machine: Example

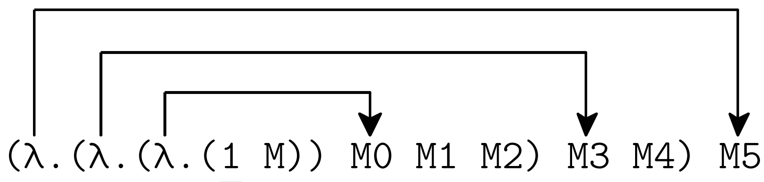


C = (1 M)

K = mt ➡ (bind M0) ➡ (bind M3) ➡ (bind M5)

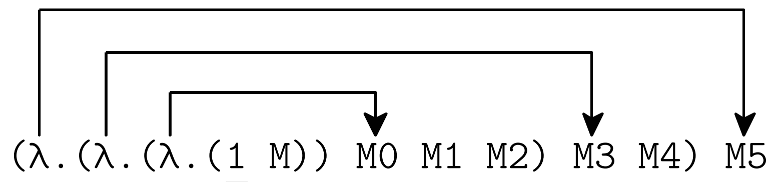
- (bind M0) ⬇ (arg M1) ⬇ (arg M2) ⬇ mt
- (bind M3) ⬇ (arg M4) ⬇ mt
- (bind M5) ⬇ mt

CK+ Machine: Example



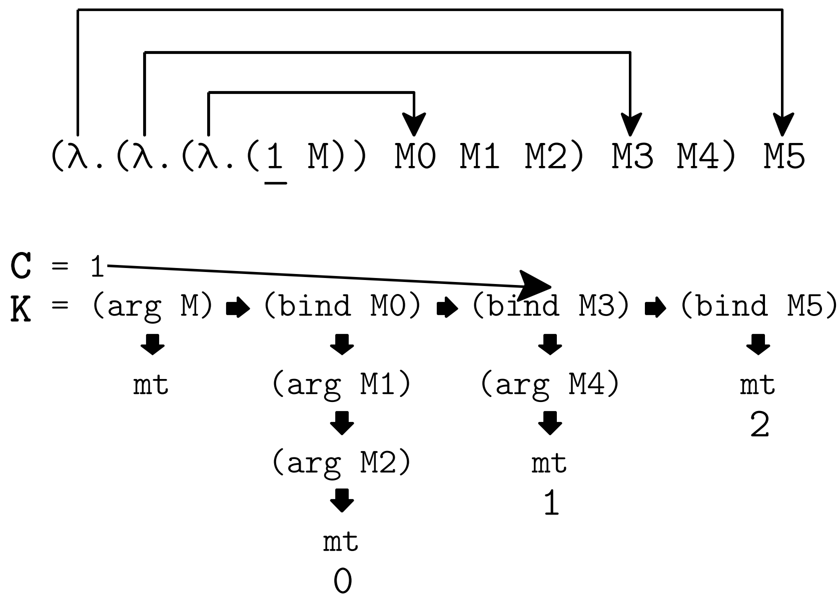

```
C = 1
K = (arg M) ➡ (bind M0) ➡ (bind M3) ➡ (bind M5)
       ⬇          ⬇            ⬇            ⬇
       mt      (arg M1)     (arg M4)        mt
                  ⬇            ⬇
               (arg M2)        mt
                  ⬇
                  mt
```

CK+ Machine: Example

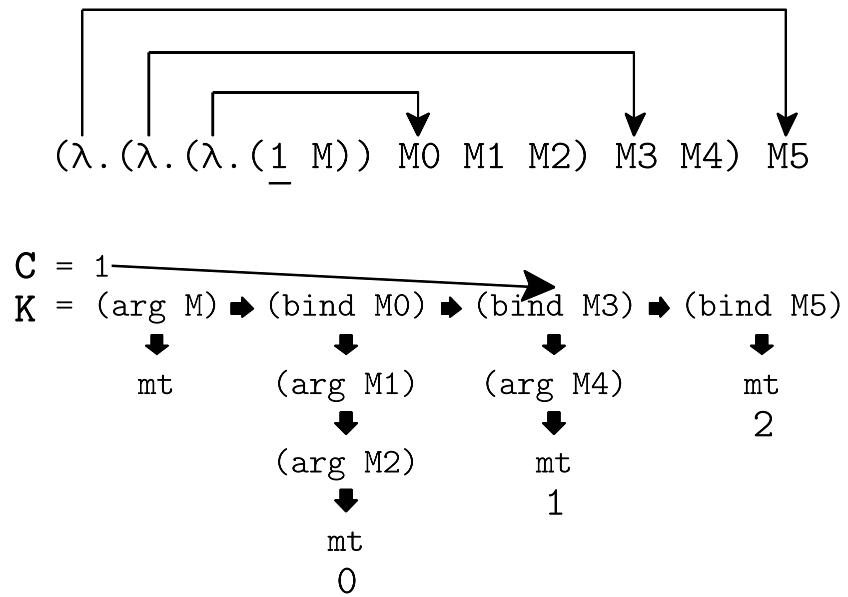



```
C = 1
K = (arg M) ➡ (bind M0) ➡ (bind M3) ➡ (bind M5)
       ⬇          ⬇            ⬇           ⬇
       mt     (arg M1)     (arg M4)        mt
                  ⬇            ⬇           2
              (arg M2)         mt
                  ⬇            1
                  mt
                  0
```

CK+ Machine: Example



CK+ Machine: Example



- Direct index instead of search

Stack Compaction

Stack Compaction

$$\texttt{((}\lambda\texttt{x.M) N)} \longrightarrow \texttt{M}$$

where $\texttt{x} \notin \texttt{FV(M)}$

Stack Compaction

$$\texttt{((}\lambda\texttt{x.M) N)} \longrightarrow \texttt{M}$$

where $\texttt{x} \notin \texttt{FV(M)}$

$\texttt{(}\lambda\texttt{.(}\lambda\texttt{.(}\lambda\texttt{.(1 M)) M0 M1 M2) M3 M4) M5}$

where

`No variables reference M0 or M5`

Stack Compaction

$$((\lambda x.\mathtt{M})\ \mathtt{N}) \longrightarrow \mathtt{M}$$

where $\mathtt{x} \notin \mathtt{FV(M)}$

$$(\lambda.(\lambda.(\lambda.(1\ \mathtt{M}))\ \mathtt{M0}\ \mathtt{M1}\ \mathtt{M2})\ \mathtt{M3}\ \mathtt{M4})\ \mathtt{M5}$$

where

No variables reference M0 or M5

C = 1

K = (arg M) ➡ (bind M0) ➡ (bind M3) ➡ (bind M5)

- (arg M) ⬇ mt
- (bind M0) ⬇ (arg M1) ⬇ (arg M2) ⬇ mt
- (bind M3) ⬇ (arg M4) ⬇ mt
- (bind M5) ⬇ mt

Stack Compaction

((λx.M) N) ⟶ M
where x $\notin$ FV(M)

(λ.(λ.(λ.(1 M)) M0 M1 M2) M3 M4) M5
where
No variables reference M0 or M5

C = 1

K = (arg M) ➡ **(bind M0)** ➡ (bind M3) ➡ (bind M5)

(arg M) ⬇ mt

(bind M0) ⬇ (arg M1) ⬇ (arg M2) ⬇ mt

(bind M3) ⬇ (arg M4) ⬇ mt

(bind M5) ⬇ mt

Stack Compaction

((λx.M) N) ⟶ M

where x ∉ FV(M)

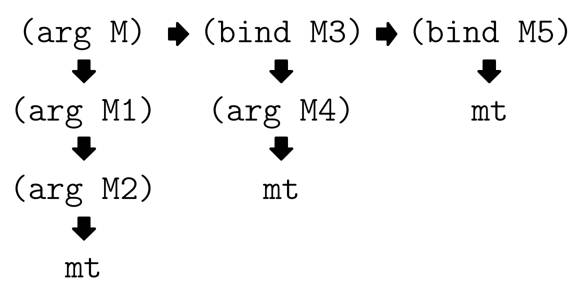
(λ.(λ.(λ.(1 M)) M0 M1 M2) M3 M4) M5

where

No variables reference M0 or M5

C = 1

K =



Stack Compaction

((λx.M) N) ⟶ M

where x ∉ FV(M)

(λ.(λ.(λ.(1 M)) M0 M1 M2) M3 M4) M5

where

No variables reference M0 or M5

C = 1

K =

```
(arg M)  ➡ (bind M3) ➡ (bind M5)
   ⬇          ⬇           ⬇
(arg M1)    (arg M4)       mt
   ⬇          ⬇
(arg M2)       mt
   ⬇
   mt
```

Stack Compaction

$$((\lambda x.M)\ N) \longrightarrow M$$

where $x \notin FV(M)$

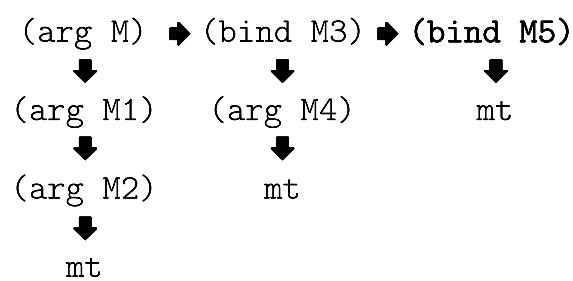
(λ.(λ.(λ.(1 M)) M0 M1 M2) M3 M4) M5

where

No variables reference M0 or M5

C = 1

K =



Stack Compaction

((λx.M) N) ⟶ M

where x ∉ FV(M)

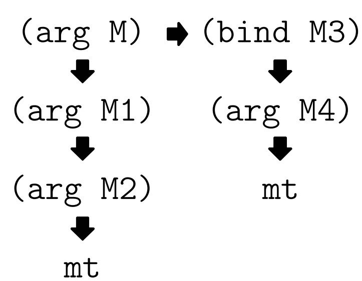
(λ.(λ.(λ.(1 M)) M0 M1 M2) M3 M4) M5

where

No variables reference M0 or M5

C = 1

K =



Thanks!