CMSC 430 - 22 Oct 2026

Tail calls

- Jig - tail calls

CMSC 430 - 22 Oct 2026

Announcements

- Assignment 6 due next ~~Thursday~~ *Tuesday*
- Assignment 7 due Tuesday after next
- Quiz released after class, due by start of Tuesday

Jig

(cont.)

Key idea

Some calls don’t need to be returned to

When a function call doesn’t need to be returned to:

- Pop local environment
- Push arguments
- Jump

When can’t you do this?

When there’s work to do after the call

```
(define (f x)
  (if (zero? x)
      0
      (f (sub1 x))))

(f 100)
```



```
(define (g x)
  (if (zero? x)
      0
      (add1 (g (sub1 x)))))

(g 100)
```



Sometimes you can rewrite to enable tail calls

So there's no work to do after the call

```
(define (g x)
  (if (zero? x)
      0
      (add1 (g (sub1 x)))))

(g 100)
```



```
(define (g x)
  (g/a x 0))

(define (g/a x a)
  (if (zero? x)
      a
      (g/a (sub1 x) (add1 a))))

(g 100)
```



Sometimes you can rewrite to enable tail calls

So there's no work to do after the call

```
(define (reverse xs)
  (match xs
    ['() '()]
    [(cons x xs)
     (append (reverse xs)
             (list x))]))
```



```
(define (reverse xs)
  (append-reverse xs '()))
```



```
;; (append-reverse xs ys)
;;    ≡ (append (reverse xs) ys)
(define (append-reverse xs ys)
  (match xs
    ['() ys]
    [(cons x xs)
     (append-reverse xs (cons x ys))]))
```



DrRacket will show you tail positions

Hover over a function definition

```
#lang racket
(define (reverse xs)
  (append-reverse xs '()))

(define (append-reverse xs ys)
  (match xs
    ['() ys]
    [(cons x xs)
     (append-reverse xs (cons x ys))]))
```

Any function call in these positions is a tail call.

When does a call not need to be returned to?

When it occurs in tail position

```
(define (f x ...) <tail>)

<tail> ::=
  (if e1 <tail> e3)
  (if e1 e2 <tail>)
  (let ((x e)) <tail>)
  (begin e <tail>)
  (f e1 ...)
```

Compile calls in these positions to pop environment, push arguments, and jump.

Compile calls in other positions to push return label, push arguments, jump.

When does a call not need to be returned to?

When it occurs in tail position

```
;; Expr CEnv Boolean -> Asm
(define (compile-e e c t?) ...)
```

t? - is this expression in tail position

```
;; Defn -> Asm
(define (compile-define d)
  (match d
    [(Defn f xs e)
     (seq (Label (symbol->label f))
          (compile-e e (reverse xs) #t)
          (Add rsp (* 8 (length xs))) ; pop args
          (Ret))]))
```

When does a call not need to be returned to?

When it occurs in tail position

```
;; Expr CEnv Boolean -> Asm
(define (compile-e e c t?) ...)
```

t? - is this expression in tail position

```
;; Expr Expr Expr CEnv Boolean -> Asm
(define (compile-if e1 e2 e3 c t?)
  (let ((l1 (gensym 'if))
        (l2 (gensym 'if)))
    (seq (compile-e e1 c #f)
         (Cmp rax (value->bits #f))
         (Je l1)
         (compile-e e2 c t?)
         (Jmp l2)
         (Label l1)
         (compile-e e3 c t?)
         (Label l2))))
```

When does a call not need to be returned to?

When it occurs in tail position

```
;; Expr CEnv Boolean -> Asm
(define (compile-e e c t?) ...)
```

t? - is this expression in tail position

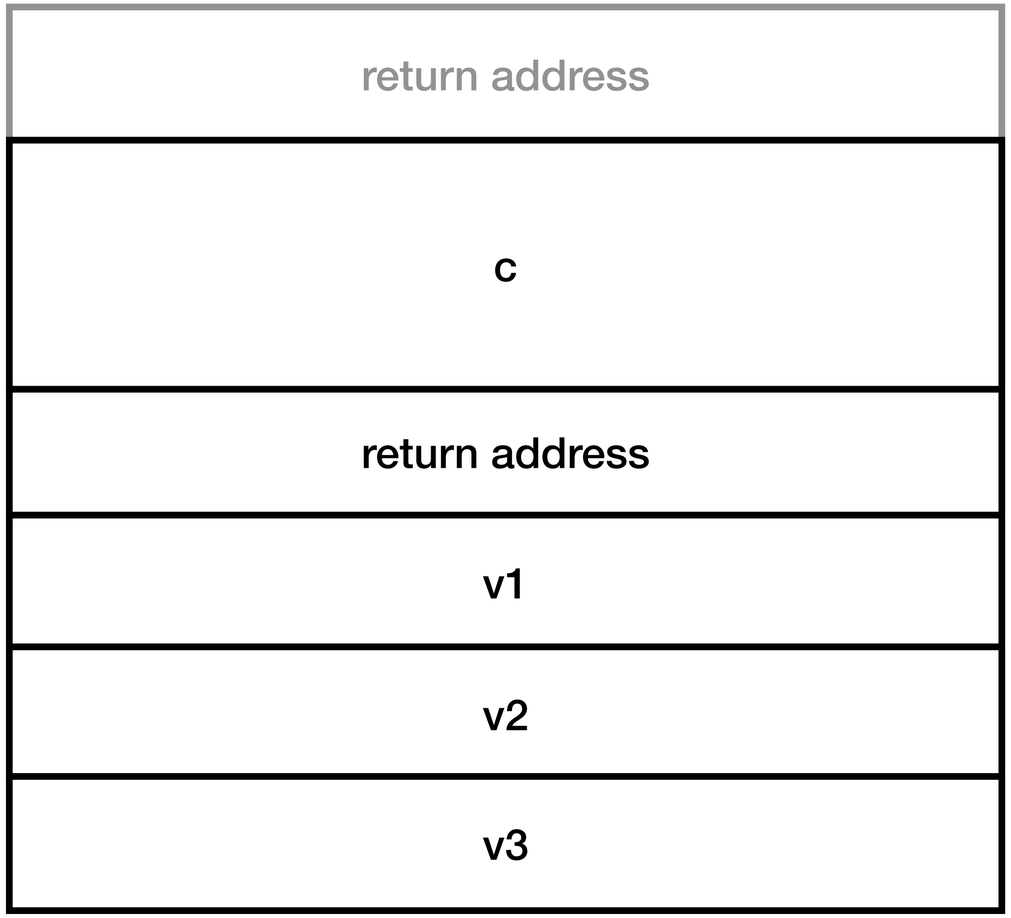
```
;; Id [Listof Expr] CEnv Boolean -> Asm
(define (compile-app f es c t?)
  (if t?
      (compile-app-tail f es c)
      (compile-app-nontail f es c)))
```

Non-tail calls

More work to do after call, so need to return

```
(compile-app-nontail 'f (list e1 e2 e3)) c)


(seq
  (Lea rax (Mem 'return))
  (Push rax)
  (compile-e e1 (cons #f c))
  (Push rax)
  (compile-e e2 (cons #f (cons #f c)))
  (Push rax)
  (compile-e e3 (cons #f (cons #f (cons #f c))))
  (Push rax)
  (Jmp 'f)
  (Label 'return))
```



Non-tail calls

More work to do after call, so need to return

```
(compile-define (Defn 'f (list 'x 'y 'z)) e))

(seq
  (Label 'f)
  (compile-e e (list 'z 'y 'x) #t)
  (Add rsp 24)
  (Ret))
```

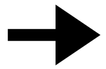


Non-tail calls

More work to do after call, so need to return

```
(compile-define (Defn 'f (list 'x 'y 'z)) e))

(seq
  (Label 'f)
  (compile-e e (list 'z 'y 'x) #t)
  (Add rsp 24)
  (Ret))
```



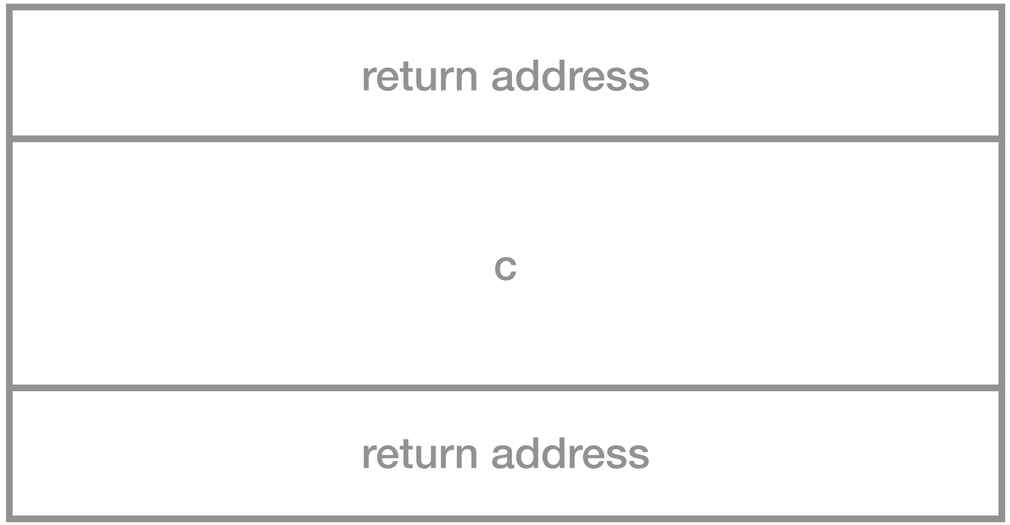


Non-tail calls

More work to do after call, so need to return

```
(compile-define (Defn 'f (list 'x 'y 'z)) e))


(seq
  (Label 'f)
  (compile-e e (list 'z 'y 'x) #t)
  (Add rsp 24)
  (Ret))
```

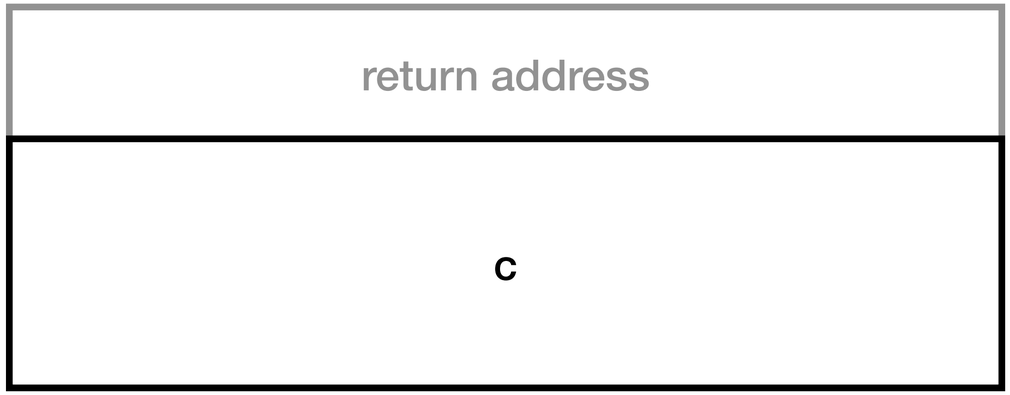


Non-tail calls

More work to do after call, so need to return

```
(compile-app-nontail ‘f (list e1 e2 e3)) c)


(seq
  (Lea rax 'return)
  (Push rax)
  (compile-e e1 (cons #f c))
  (Push rax)
  (compile-e e2 (cons #f (cons #f c)))
  (Push rax)
  (compile-e e3 (cons #f (cons #f (cons #f c))))
  (Push rax)
  (Jmp 'f)
  (Label 'return))
```

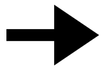


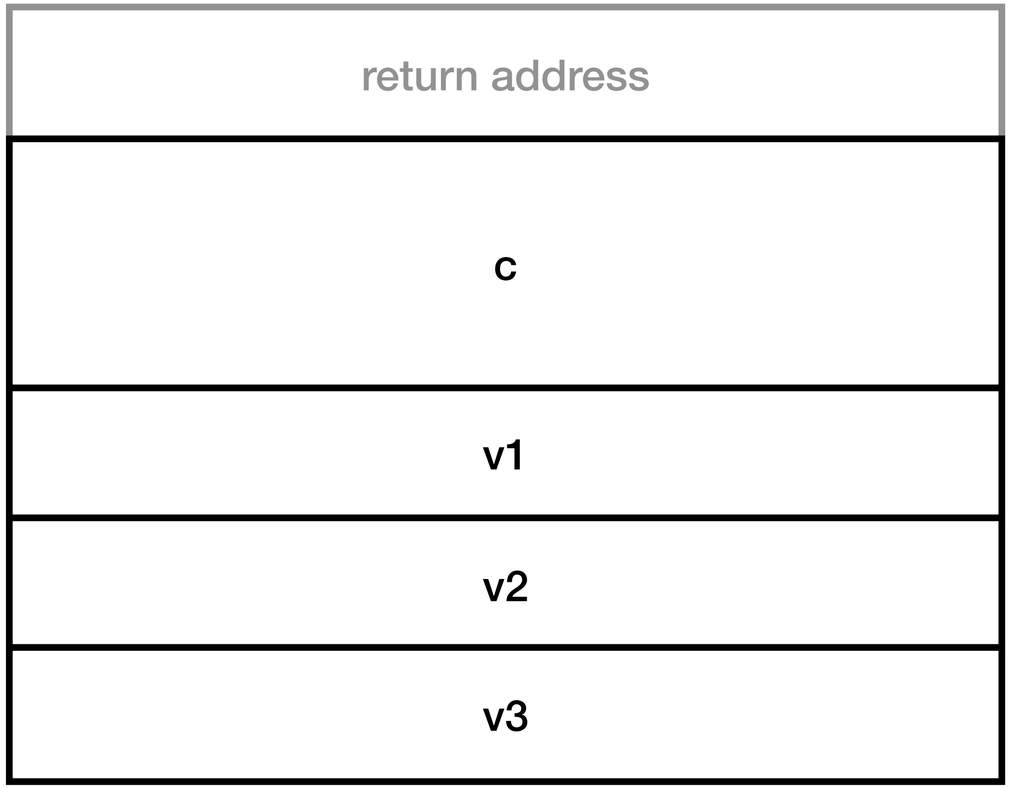
Tail calls (first attempt)

No more work to do after call, so don’t return

```
(compile-app-tail 'f (list e1 e2 e3)) c)

(seq
  (compile-e e1 c)
  (Push 'rax)
  (compile-e e2 (cons #f c))
  (Push 'rax)
  (compile-e e3 (cons #f (cons #f c)))
  (Push 'rax)
  (Jmp 'f))
```



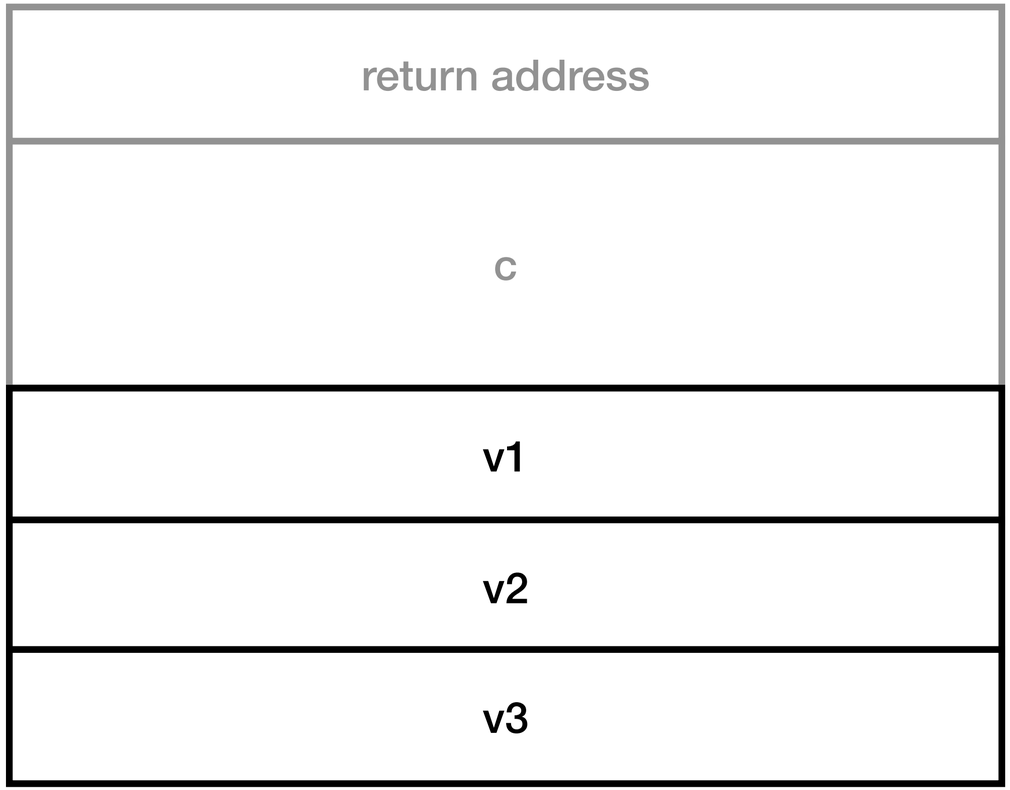


Tail calls (first attempt)

No more work to do after call, so don’t return

```
(compile-define (Defn 'f (list 'x 'y 'z)) e))

(seq
  (Label 'f)
  (compile-e e (list 'z 'y 'x) #t)
  (Add rsp 24)
  (Ret))
```

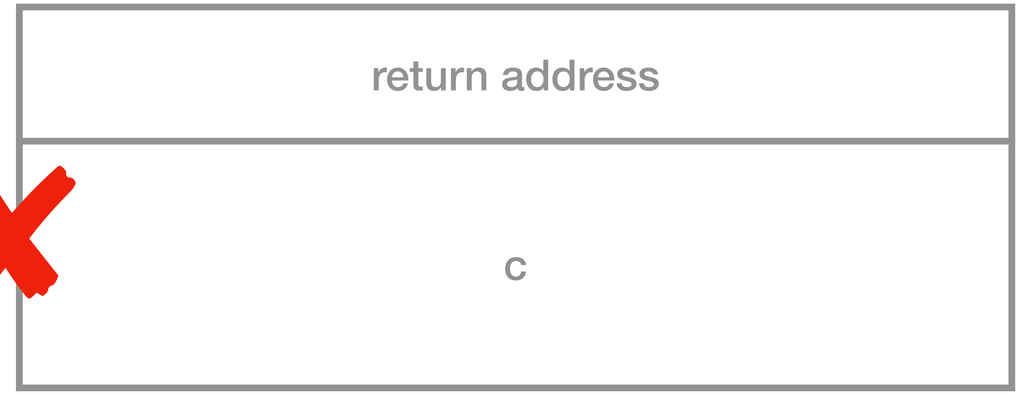


Tail calls (first attempt)

No more work to do after call, so don’t return

```
(compile-define (Defn 'f (list 'x 'y 'z)) e))


(seq
  (Label 'f)
  (compile-e e (list 'z 'y 'x) #t)
  (Add rsp 24)
  (Ret))
```

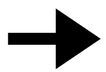


Tail calls (second attempt)

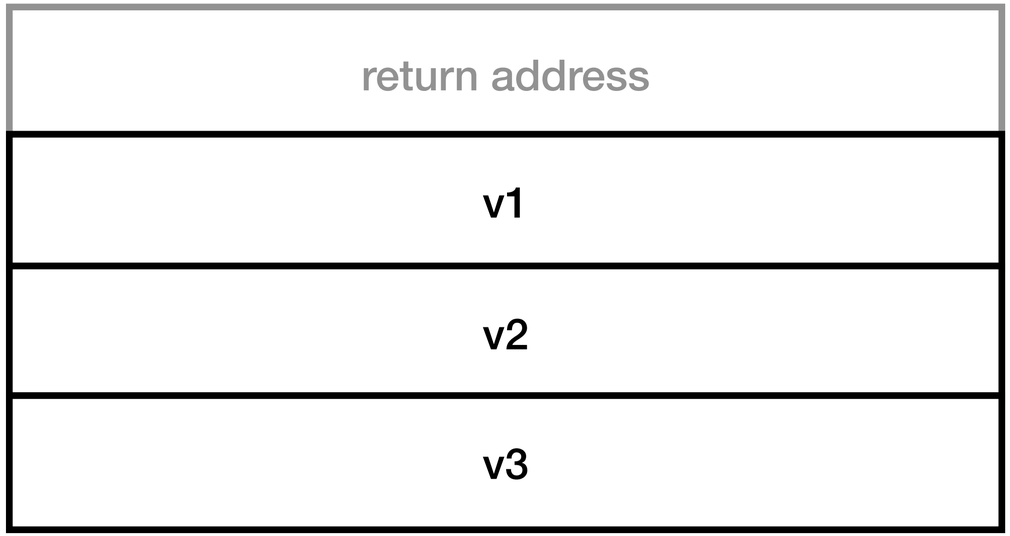
No more work to do after call, so don't return

```
(compile-app-tail 'f (list e1 e2 e3)) c)

(seq
  (Add rsp (* 8 (length c)))
  (compile-e e1 c)
  (Push rax)
  (compile-e e2 (cons #f c))
  (Push rax)
  (compile-e e3 (cons #f (cons #f c)))
  (Push rax)
  (Jmp 'f))
```





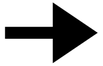


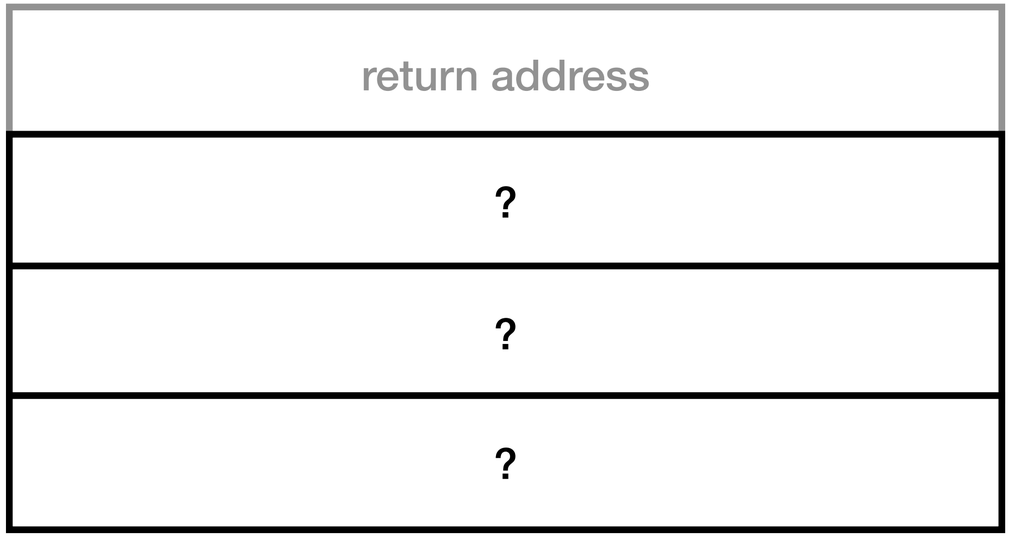
Tail calls (third attempt)

No more work to do after call, so don't return

```
(compile-app-tail 'f (list e1 e2 e3)) c)

(seq
  (compile-e e1 c)
  (Push rax)
  (compile-e e2 (cons #f c))
  (Push rax)
  (compile-e e3 (cons #f (cons #f c)))
  (Push rax)
  (Add rsp (* 8 (length c)))
  (Jmp 'f))
```



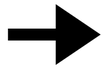


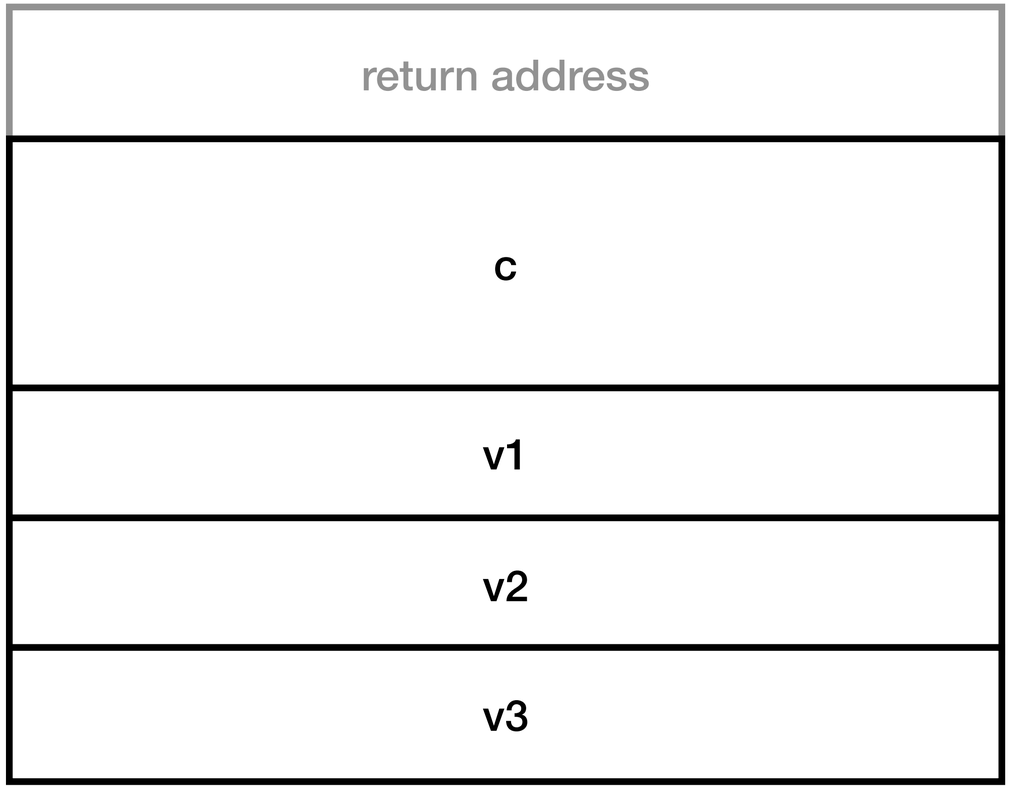
Tail calls (final attempt)

No more work to do after call, so don't return

```
(compile-app-tail 'f (list e1 e2 e3)) c)

(seq
  (compile-e e1 c)
  (Push rax)
  (compile-e e2 (cons #f c))
  (Push rax)
  (compile-e e3 (cons #f (cons #f c)))
  (Push rax)
  (move-args (length es) (length c))
  (Add rsp (* 8 (length c)))
  (Jmp 'f))
```



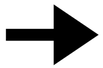


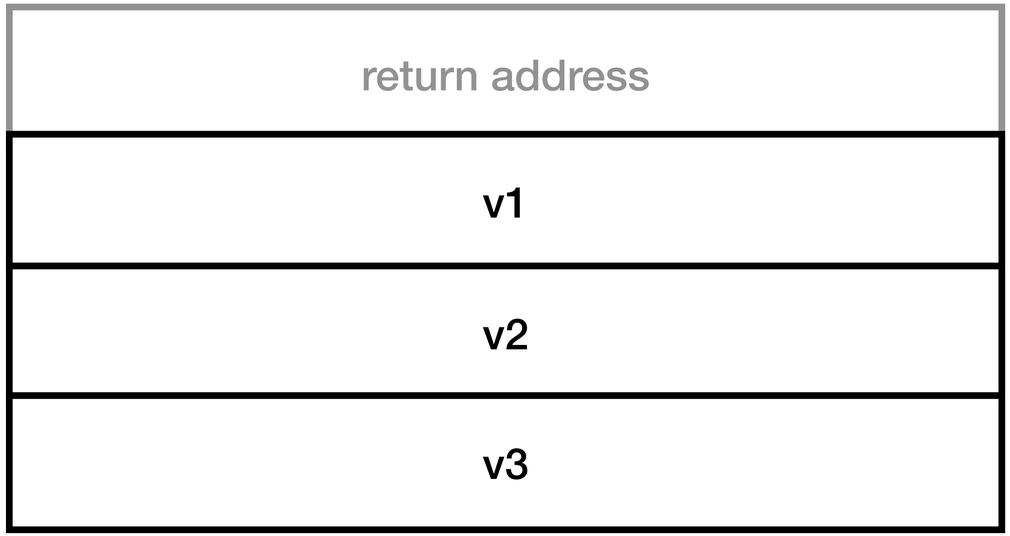
Tail calls (final attempt)

No more work to do after call, so don't return

```
(compile-app-tail 'f (list e1 e2 e3)) c)

(seq
  (compile-e e1 c)
  (Push rax)
  (compile-e e2 (cons #f c))
  (Push rax)
  (compile-e e3 (cons #f (cons #f c)))
  (Push rax)
  (move-args 3 (length c))
  (Add rsp (* 8 (length c)))
  (Jmp 'f))
```



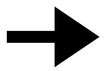


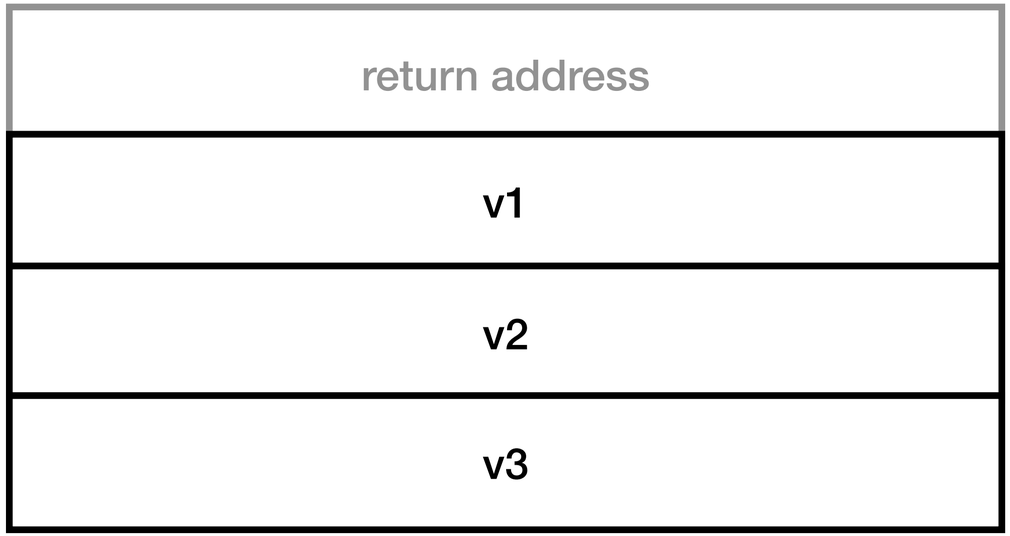
Tail calls (final attempt)

No more work to do after call, so don’t return

```
(compile-define (Defn 'f (list 'x 'y 'z)) e))

(seq
  (Label 'f)
  (compile-e e (list 'z 'y 'x) #t)
  (Add rsp 24)
  (Ret))
```





Tail calls (final attempt)

No more work to do after call, so don’t return

```
(compile-define (Defn 'f (list 'x 'y 'z)) e))
```

```
(seq
  (Label 'f)
  (compile-e e (list 'z 'y 'x) #t)
  (Add rsp 24)
  (Ret))
```

