Continuation-passing Style (CPS)

Assignment-converted/alphatized IR (.alpha)

```
e ::= (let ([x e] ...) e)
    | (lambda (x ...) e)
    | (lambda x e)
    | (apply e e)
    | (e e ...)
    | (prim op e ...)
    | (apply-prim op e)
    | (if e e e)
    | (call/cc e)
    | x
    | (quote dat)
```

Administrative normal form (ANF) (.anf)

```
e ::= (let ([x e]) e)
    | (apply ae ae)
    | (ae ae ...)
    | (prim op ae ...)
    | (apply-prim op ae)
    | (if ae e e)
    | (call/cc ae)
    | ae
ae ::= (lambda (x ...) e)
    | (lambda x e)
    | x
    | (quote dat)
```

Continuation-passing style (CPS) (.cps)

```
e ::= (let ([x (apply-prim op ae)]) e)
    | (let ([x (prim op ae ...)]) e)
    | (apply ae ae)
    | (ae ae ...)
    | (if ae e e)
ae ::= (lambda (x ...) e)
    | (lambda x e)
    | x
    | (quote dat)
```

```
e ::= (let ([x (apply-prim op ae)]) e)
    | (let ([x (prim op ae ...)]) e)
    | (apply ae ae)
    | (ae ae ...)
    | (if ae e e)
ae ::= (lambda (x ...) e)
    | (lambda x e)
    | x
    | (quote dat)
```

- Programs in CPS require no stack and never return.

- Instead, at each application, a continuation (a callback function) is passed forward explicitly.

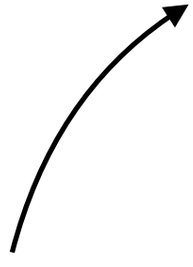
- Points that would otherwise have extended the stack now create a closure (where the environment saves local variables and the stack tail).

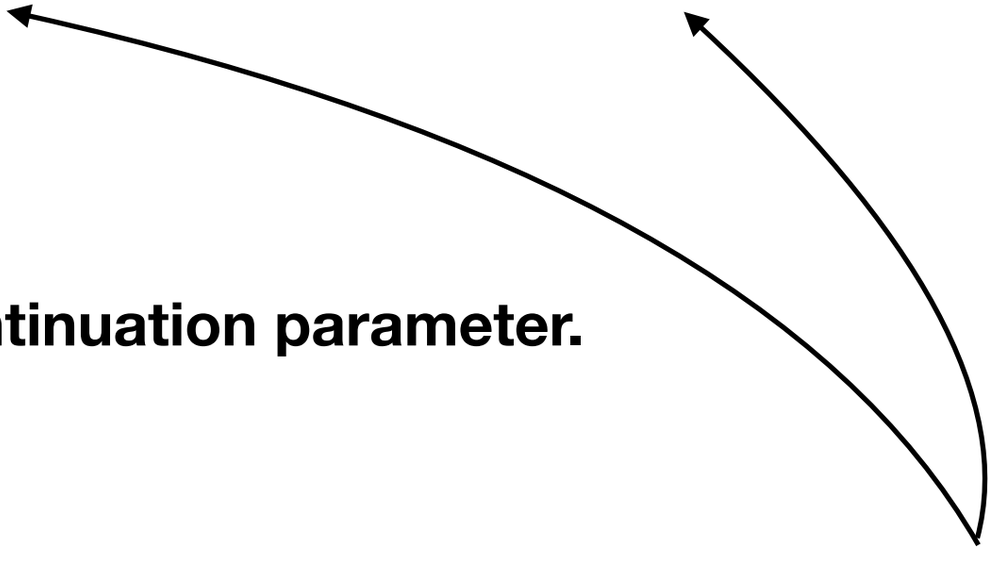
- Return points become invocations of the current continuation.

```
(let ([x ((lambda (y z) z) a b)])
  e)
```

e_{cps} has a free var for e's cont.

```
((lambda (k y z) (k 0 z)) (lambda (k v) e_cps) a b)
```





All functions take an extra continuation parameter.

As call/cc lets us pass continuations as values, so must they (despite not using it).

```
call/cc =
(lambda (k f)
  (f k k))
```

Visualizing CPS (example)

IR

```
(define (fib n)
  (if (<= n 1)
      n
      (+ (fib (- n 1))
         (fib (- n 2)))))
```

```
(fib 4)
```

IR

```
(define (fib n)
  (if (<= n 1)
      n
      (+ (fib (- n 1))
         (fib (- n 2)))))
```

```
(fib 3)
```

```
fn {+} (fib (- n 2)) [n = 4]
```

IR

```
(define (fib n)
  (if (<= n 1)
      n
      (+ (fib (- n 1))
         (fib (- n 2)))))
```

```
(fib 2)
```

```
fn {+} (fib (- n 2)) [n = 3]
fn {+} (fib (- n 2)) [n = 4]
```

IR

```
(define (fib n)
  (if (<= n 1)
      n
      (+ (fib (- n 1))
         (fib (- n 2)))))
```

```
(fib 1)
```

```
fn {+} (fib (- n 2)) [n = 2]
fn {+} (fib (- n 2)) [n = 3]
fn {+} (fib (- n 2)) [n = 4]
```

IR

```
(define (fib n)
  (if (<= n 1)
      n
      (+ (fib (- n 1))
         (fib (- n 2)))))
```

{1}

```
fn {+} (fib (- n 2)) [n = 2]
fn {+} (fib (- n 2)) [n = 3]
fn {+} (fib (- n 2)) [n = 4]
```

IR

```
(define (fib n)
  (if (<= n 1)
      n
      (+ (fib (- n 1))
         (fib (- n 2)))))
```

```
(fib 0)
```

```
fn {+} {1}
```

```
fn {+} (fib (- n 2)) [n = 3]
```

```
fn {+} (fib (- n 2)) [n = 4]
```

IR

```
(define (fib n)
  (if (<= n 1)
      n
      (+ (fib (- n 1))
         (fib (- n 2)))))
```

0

```
fn {+} {1}
fn {+} (fib (- n 2)) [n = 3]
fn {+} (fib (- n 2)) [n = 4]
```

IR

```
(define (fib n)
  (if (<= n 1)
      n
      (+ (fib (- n 1))
         (fib (- n 2)))))
```

1

```
fn {+} (fib (- n 2)) [n = 3]
fn {+} (fib (- n 2)) [n = 4]
```

IR

```
(define (fib n)
  (if (<= n 1)
      n
      (+ (fib (- n 1))
         (fib (- n 2)))))
```

```
(fib 1)
```

```
fn {+} {1}
```

```
fn {+} (fib (- n 2)) [n = 4]
```

IR

```
(define (fib n)
  (if (<= n 1)
      n
      (+ (fib (- n 1))
         (fib (- n 2)))))
```

1

```
fn {+} {1}
fn {+} (fib (- n 2)) [n = 4]
```

IR

```
(define (fib n)
  (if (<= n 1)
      n
      (+ (fib (- n 1))
         (fib (- n 2)))))
```

2

```
fn {+} (fib (- n 2)) [n = 4]
```

IR

```
(define (fib n)
  (if (<= n 1)
      n
      (+ (fib (- n 1))
         (fib (- n 2)))))
```

```
(fib 2)
```

```
fn {+} {2}
```

IR

```
(define (fib n)
  (if (<= n 1)
      n
      (+ (fib (- n 1))
         (fib (- n 2)))))
```

```
(fib 1)
```

```
fn {+} (fib (- n 2)) [n = 2]
```

```
fn {+} {2}
```

IR

```
(define (fib n)
  (if (<= n 1)
      n
      (+ (fib (- n 1))
         (fib (- n 2)))))
```

1

```
fn {+} (fib (- n 2)) [n = 2]
fn {+} {2}
```

IR

```
(define (fib n)
  (if (<= n 1)
      n
      (+ (fib (- n 1))
         (fib (- n 2)))))
```

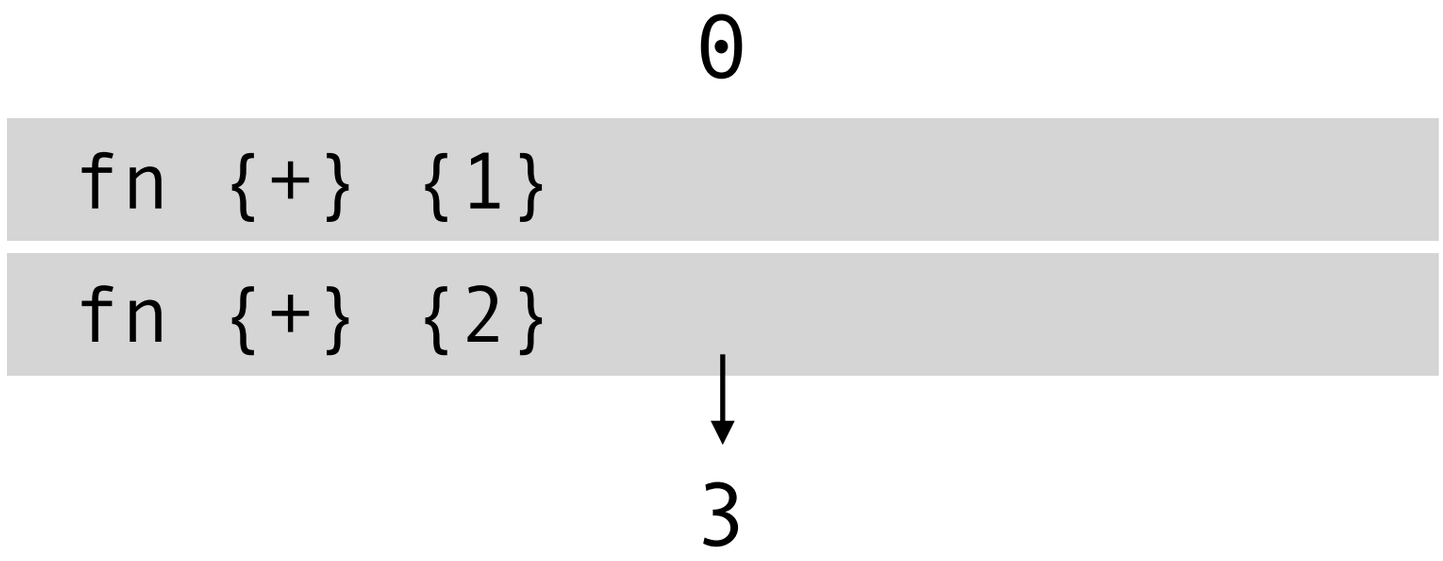
```
(fib 0)
```

```
fn {+} {1}
```

```
fn {+} {2}
```

IR

```
(define (fib n)
  (if (<= n 1)
      n
      (+ (fib (- n 1))
         (fib (- n 2)))))
```



ANF

```
(define (fib n)
  (let ([c (<= n 1)])
    (if c
        n
        0(let ([n-1 (- n 1)])
           1(let ([v0 (fib n-1)])
              2(let ([n-2 (- n 2)])
                 3(let ([v1 (fib n-2)])
                    4(let ([s (+ v0 v1)])
                       5s)))))))))
```

(fib 4)

ANF

```
(define (fib n)
  (let ([c (<= n 1)])
    (if c
        n
        0(let ([n-1 (- n 1)])
           1(let ([v0 (fib n-1)])
              2(let ([n-2 (- n 2)])
                 3(let ([v1 (fib n-2)])
                    4(let ([s (+ v0 v1)])
                       5s)))))))
```

(fib 3)

letk v0 e_2 [n=4,n-1=3,…]

ANF

```
(define (fib n)
  (let ([c (<= n 1)])
    (if c
        n
        0(let ([n-1 (- n 1)])
           1(let ([v0 (fib n-1)])
              2(let ([n-2 (- n 2)])
                 3(let ([v1 (fib n-2)])
                    4(let ([s (+ v0 v1)])
                       5s)))))))
```

(fib 2)

```
letk v0 e2 [n=3,n-1=2,…]
letk v0 e2 [n=4,n-1=3,…]
```

ANF

```
(define (fib n)
  (let ([c (<= n 1)])
    (if c
        n
        0(let ([n-1 (- n 1)])
           1(let ([v0 (fib n-1)])
              2(let ([n-2 (- n 2)])
                 3(let ([v1 (fib n-2)])
                    4(let ([s (+ v0 v1)])
                       5s)))))))
```

(fib 1) -> 1

```
letk v0 e2 [n=2,n-1=1,…]
letk v0 e2 [n=3,n-1=2,…]
letk v0 e2 [n=4,n-1=3,…]
```

ANF

```
(define (fib n)
  (let ([c (<= n 1)])
    (if c
        n
        0(let ([n-1 (- n 1)])
          1(let ([v0 (fib n-1)])
            2(let ([n-2 (- n 2)])
              3(let ([v1 (fib n-2)])
                4(let ([s (+ v0 v1)])
                  5s)))))))
```

(fib 0) -> 0

```
letk v1 e4 [v0=1,n=2,n-1=1,…]
letk v0 e2 [n=3,n-1=2,…]
letk v0 e2 [n=4,n-1=3,…]
```

ANF

```
(define (fib n)
  (let ([c (<= n 1)])
    (if c
        n
        0(let ([n-1 (- n 1)])
          1(let ([v0 (fib n-1)])
            2(let ([n-2 (- n 2)])
              3(let ([v1 (fib n-2)])
                4(let ([s (+ v0 v1)])
                  5s)))))))
```

s [s=1,v0=1,v1=0,…]

letk v0 e_2 [n=3,n-1=2,…]

letk v0 e_2 [n=4,n-1=3,…]

ANF

```
(define (fib n)
  (let ([c (<= n 1)])
    (if c
        n
        0(let ([n-1 (- n 1)])
           1(let ([v0 (fib n-1)])
              2(let ([n-2 (- n 2)])
                 3(let ([v1 (fib n-2)])
                    4(let ([s (+ v0 v1)])
                       5s)))))))
```

(fib 1) -> 1

```
letk v1 e4 [v0=1,n=3,n-1=2,…]
letk v0 e2 [n=4,n-1=3,…]
```

ANF

```
(define (fib n)
  (let ([c (<= n 1)])
    (if c
        n
        0(let ([n-1 (- n 1)])
           1(let ([v0 (fib n-1)])
              2(let ([n-2 (- n 2)])
                 3(let ([v1 (fib n-2)])
                    4(let ([s (+ v0 v1)])
                       5s)))))))
```

s [s=2,v0=1,v1=1,…]

letk v0 e_2 [n=4,n-1=3,…]

ANF

```
(define (fib n)
  (let ([c (<= n 1)])
    (if c
        n
        0(let ([n-1 (- n 1)])
           1(let ([v0 (fib n-1)])
              2(let ([n-2 (- n 2)])
                 3(let ([v1 (fib n-2)])
                    4(let ([s (+ v0 v1)])
                       5s)))))))
```

```
(fib 2)
```

```
letk v1 e4 [v0=2,n=4,n-1=3,…]
```

ANF

```
(define (fib n)
  (let ([c (<= n 1)])
    (if c
        n
        0(let ([n-1 (- n 1)])
          1(let ([v0 (fib n-1)])
            2(let ([n-2 (- n 2)])
              3(let ([v1 (fib n-2)])
                4(let ([s (+ v0 v1)])
                  5s)))))))
```

(fib 1) -> 1

letk v0 e_2 [n=2,n-1=3,…]

letk v1 e_4 [v0=2,n=4,n-1=3,…]

ANF

```
(define (fib n)
  (let ([c (<= n 1)])
    (if c
        n
        0(let ([n-1 (- n 1)])
          1(let ([v0 (fib n-1)])
            2(let ([n-2 (- n 2)])
              3(let ([v1 (fib n-2)])
                4(let ([s (+ v0 v1)])
                  5s)))))))))
```

```
(fib 0) -> 0
```

```
letk v1 e4 [v0=1,n=2,n-1=3,…]
```

```
letk v1 e4 [v0=2,n=4,n-1=3,…]
```

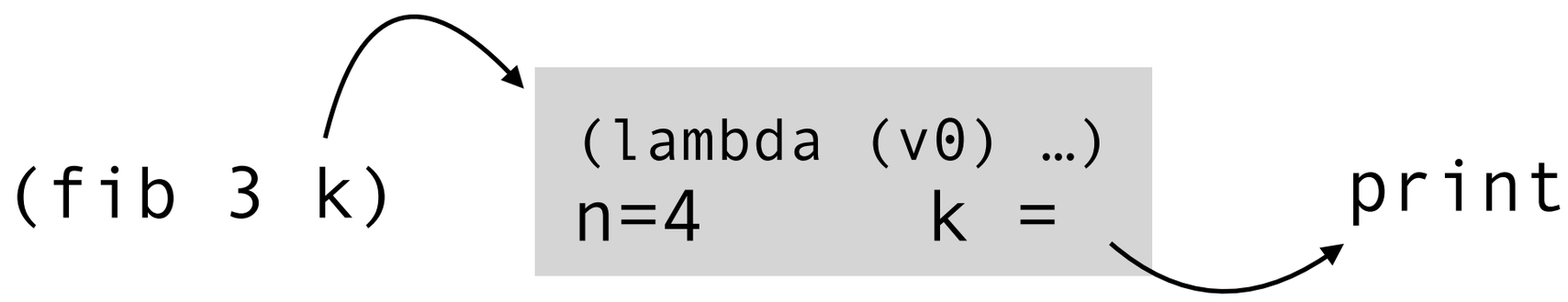
CPS

```
(define (fib n k)
  (let ([c (<= n 1)])
    (if c
        (k n)
        (let ([n-1 (- n 1)])
          (fib n-1
            (lambda (v0)
              (let ([n-2 (- n 2)])
                (fib n-2
                  (lambda (v1)
                    (let ([s (+ v0 v1)])
                      (k s)))))))))))

(fib 4 print)
```

CPS

```
(define (fib n k)
  (let ([c (<= n 1)])
    (if c
        (k n)
        (let ([n-1 (- n 1)])
          (fib n-1
            (lambda (v0)
              (let ([n-2 (- n 2)])
                (fib n-2
                  (lambda (v1)
                    (let ([s (+ v0 v1)])
                      (k s)))))))))))
```



CPS

```
(define (fib n k)
  (let ([c (<= n 1)])
    (if c
        (k n)
        (let ([n-1 (- n 1)])
          (fib n-1
            (lambda (v0)
              (let ([n-2 (- n 2)])
                (fib n-2
                  (lambda (v1)
                    (let ([s (+ v0 v1)])
                      (k s)))))))))))
```

```
(fib 2 k)
```

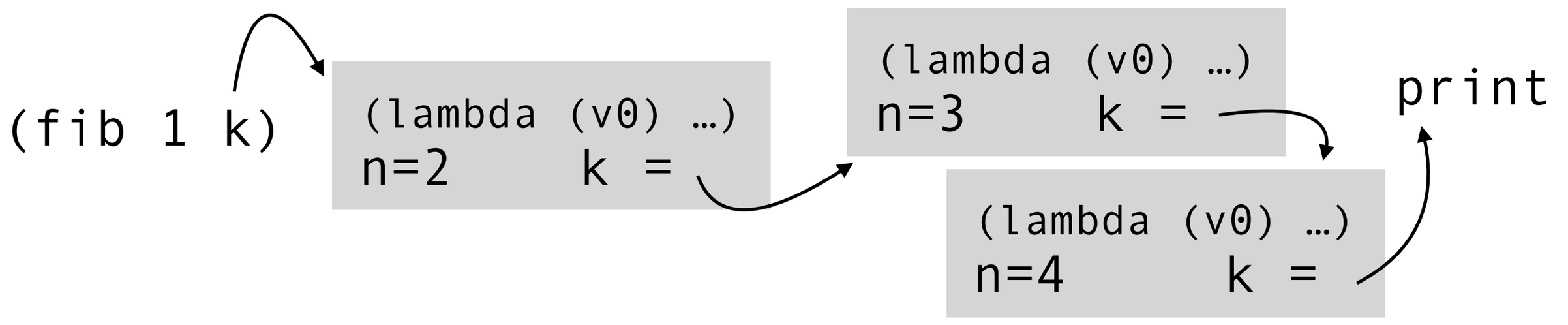
```
(lambda (v0) …)
n=3      k =
```

```
(lambda (v0) …)
n=4      k =
```

```
print
```

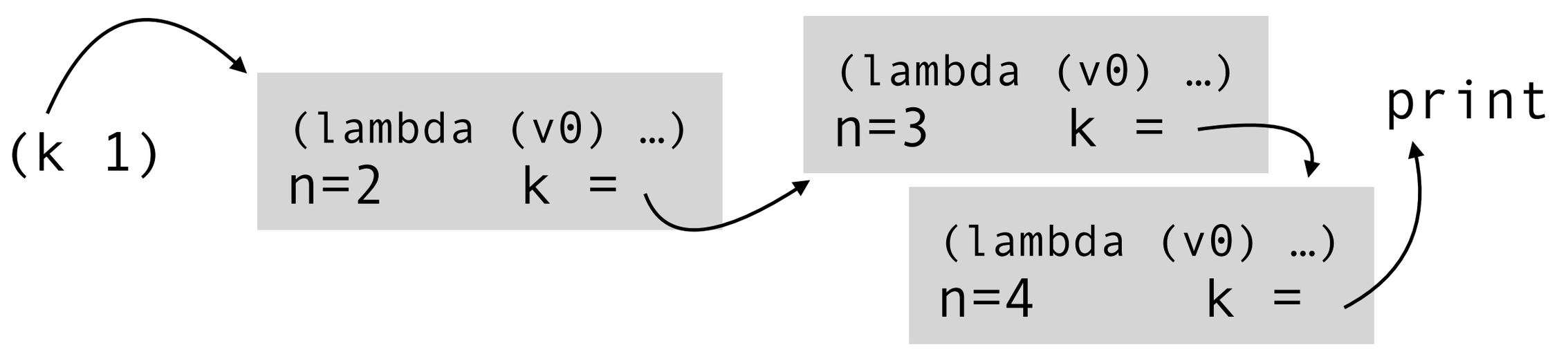
CPS

```
(define (fib n k)
  (let ([c (<= n 1)])
    (if c
        (k n)
        (let ([n-1 (- n 1)])
          (fib n-1
            (lambda (v0)
              (let ([n-2 (- n 2)])
                (fib n-2
                  (lambda (v1)
                    (let ([s (+ v0 v1)])
                      (k s)))))))))))
```



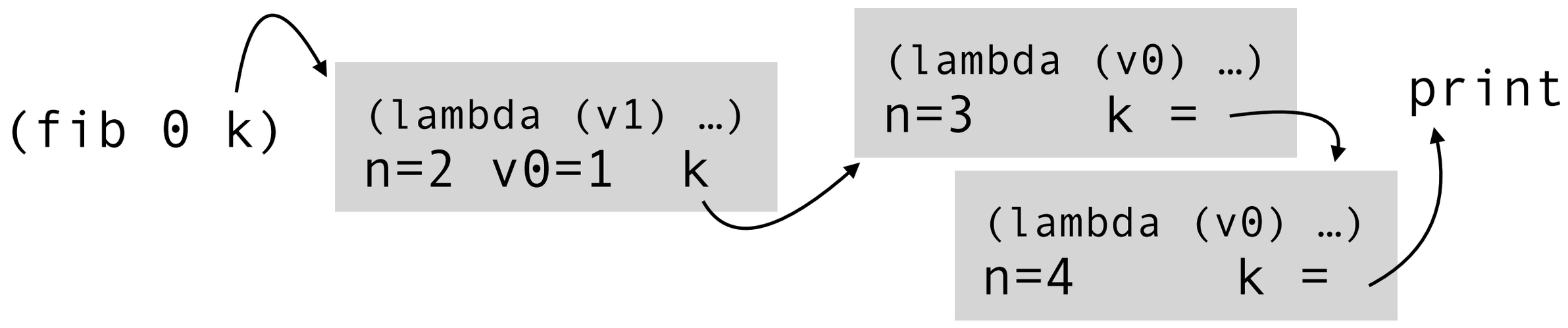
CPS

```
(define (fib n k)
  (let ([c (<= n 1)])
    (if c
        (k n)
        (let ([n-1 (- n 1)])
          (fib n-1
            (lambda (v0)
              (let ([n-2 (- n 2)])
                (fib n-2
                  (lambda (v1)
                    (let ([s (+ v0 v1)])
                      (k s)))))))))))
```



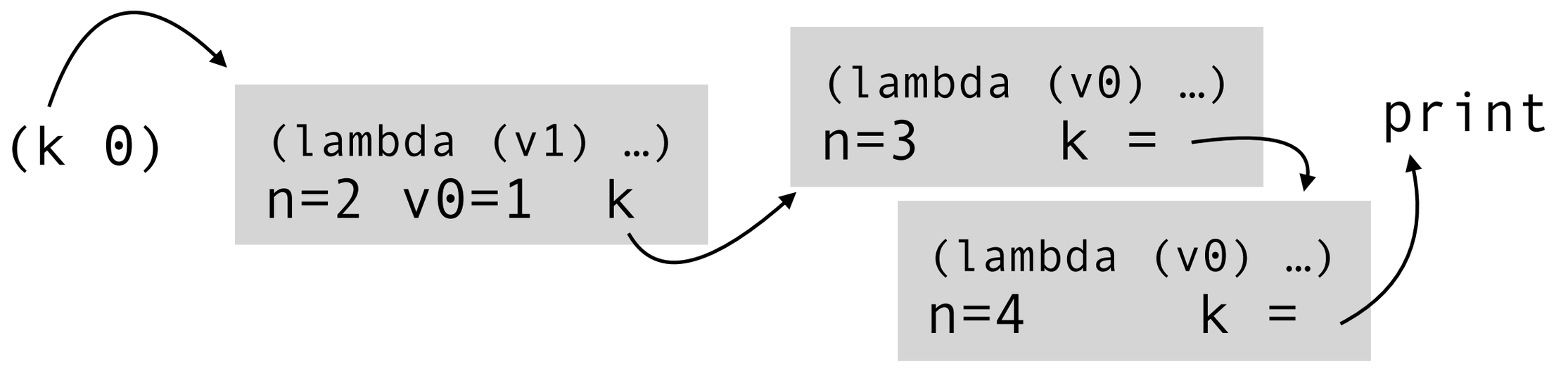
CPS

```
(define (fib n k)
  (let ([c (<= n 1)])
    (if c
        (k n)
        (let ([n-1 (- n 1)])
          (fib n-1
            (lambda (v0)
              (let ([n-2 (- n 2)])
                (fib n-2
                  (lambda (v1)
                    (let ([s (+ v0 v1)])
                      (k s)))))))))))
```



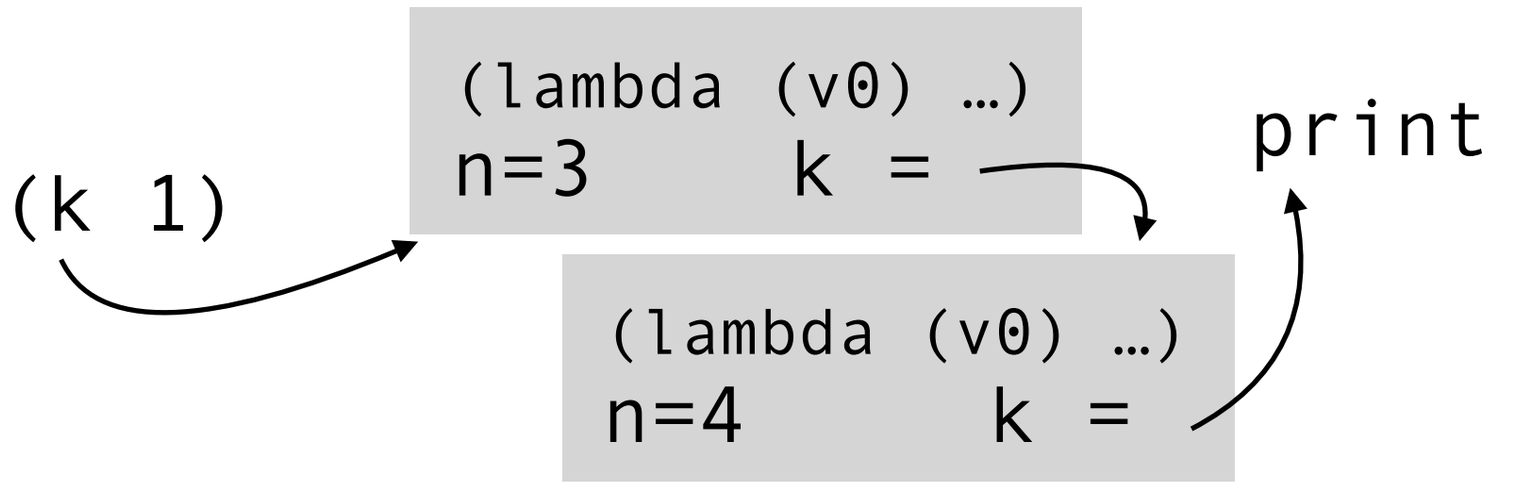
CPS

```
(define (fib n k)
  (let ([c (<= n 1)])
    (if c
        (k n)
        (let ([n-1 (- n 1)])
          (fib n-1
            (lambda (v0)
              (let ([n-2 (- n 2)])
                (fib n-2
                  (lambda (v1)
                    (let ([s (+ v0 v1)])
                      (k s)))))))))))
```



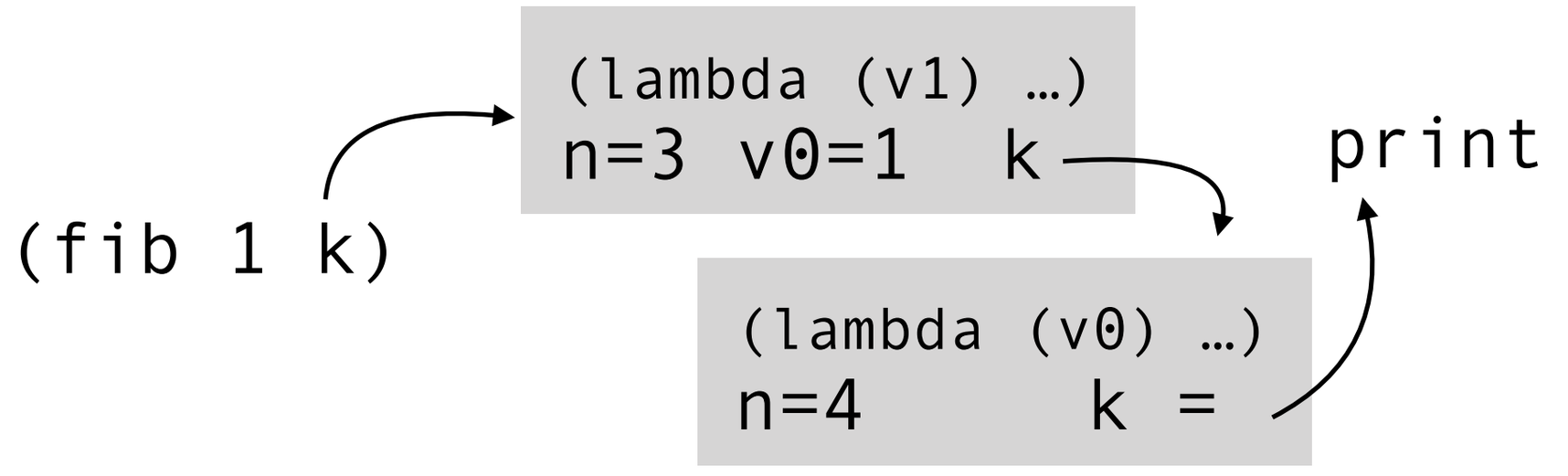
CPS

```
(define (fib n k)
  (let ([c (<= n 1)])
    (if c
        (k n)
        (let ([n-1 (- n 1)])
          (fib n-1
            (lambda (v0)
              (let ([n-2 (- n 2)])
                (fib n-2
                  (lambda (v1)
                    (let ([s (+ v0 v1)])
                      (k s)))))))))))
```



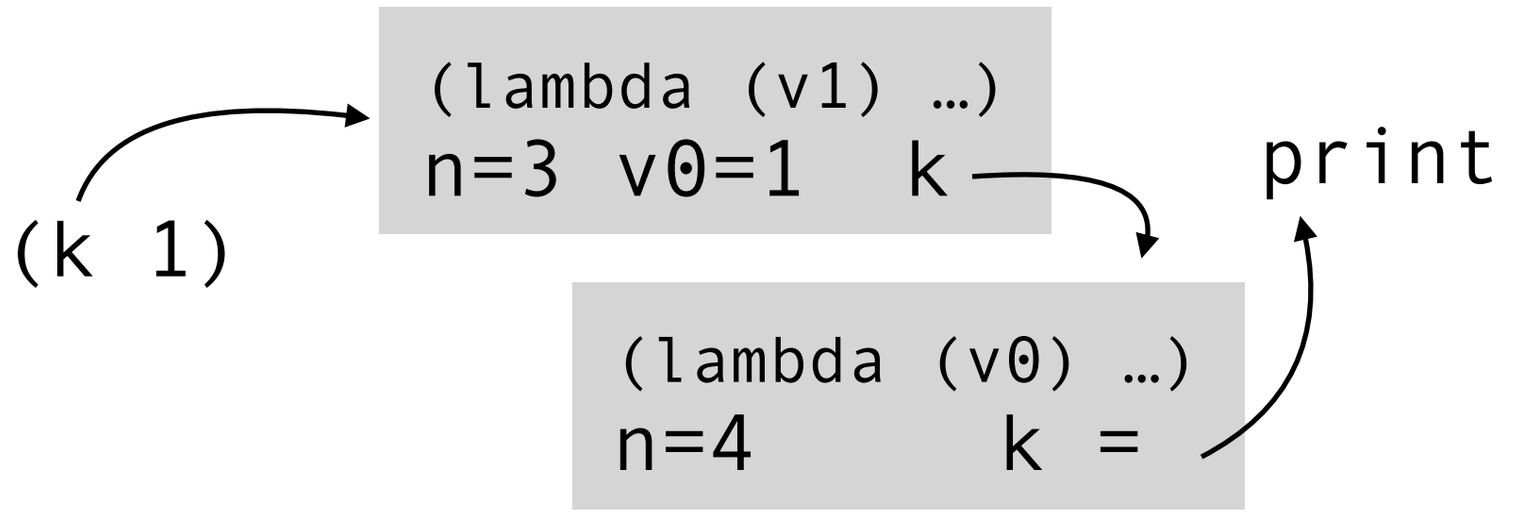
CPS

```
(define (fib n k)
  (let ([c (<= n 1)])
    (if c
        (k n)
        (let ([n-1 (- n 1)])
          (fib n-1
            (lambda (v0)
              (let ([n-2 (- n 2)])
                (fib n-2
                  (lambda (v1)
                    (let ([s (+ v0 v1)])
                      (k s)))))))))))
```



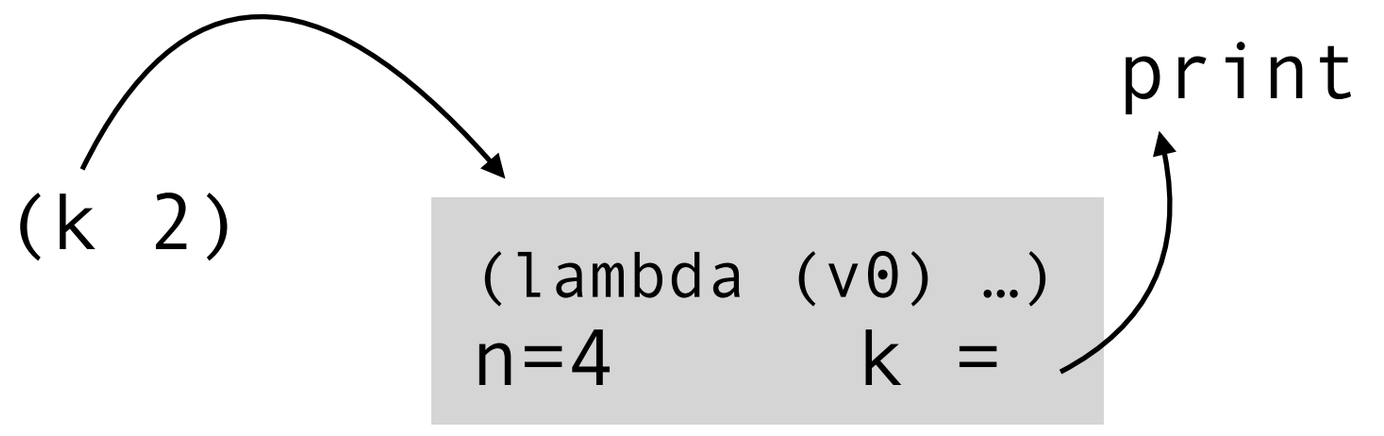
CPS

```
(define (fib n k)
  (let ([c (<= n 1)])
    (if c
        (k n)
        (let ([n-1 (- n 1)])
          (fib n-1
            (lambda (v0)
              (let ([n-2 (- n 2)])
                (fib n-2
                  (lambda (v1)
                    (let ([s (+ v0 v1)])
                      (k s)))))))))))
```



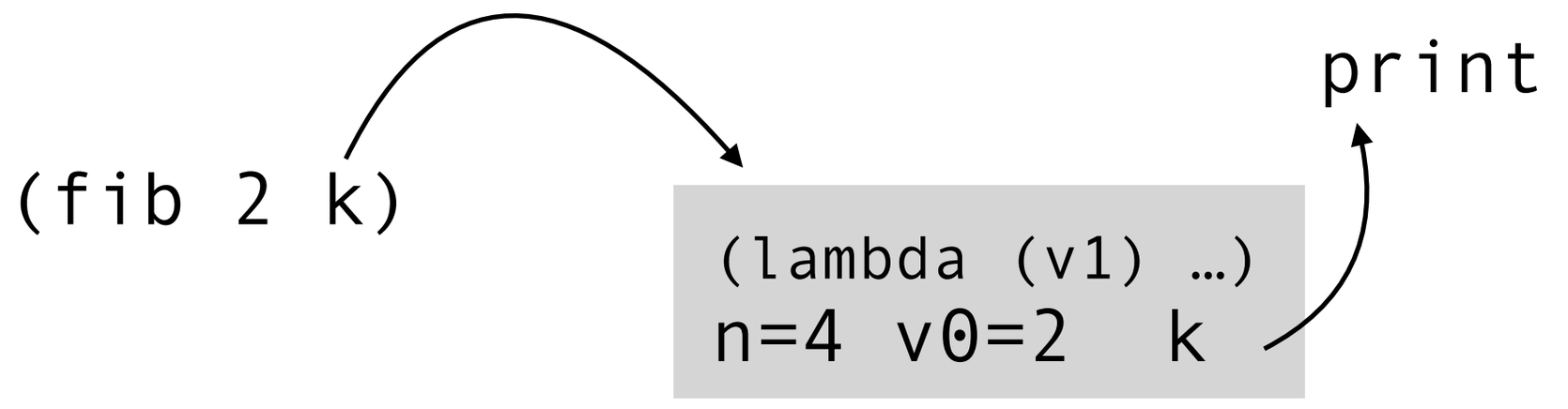
CPS

```
(define (fib n k)
  (let ([c (<= n 1)])
    (if c
        (k n)
        (let ([n-1 (- n 1)])
          (fib n-1
            (lambda (v0)
              (let ([n-2 (- n 2)])
                (fib n-2
                  (lambda (v1)
                    (let ([s (+ v0 v1)])
                      (k s)))))))))))
```



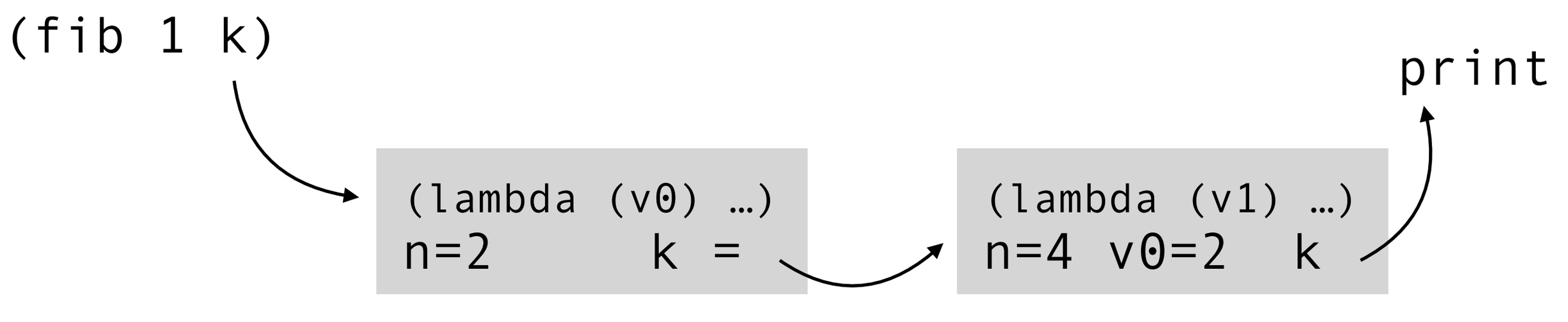
CPS

```
(define (fib n k)
  (let ([c (<= n 1)])
    (if c
        (k n)
        (let ([n-1 (- n 1)])
          (fib n-1
            (lambda (v0)
              (let ([n-2 (- n 2)])
                (fib n-2
                  (lambda (v1)
                    (let ([s (+ v0 v1)])
                      (k s)))))))))))
```



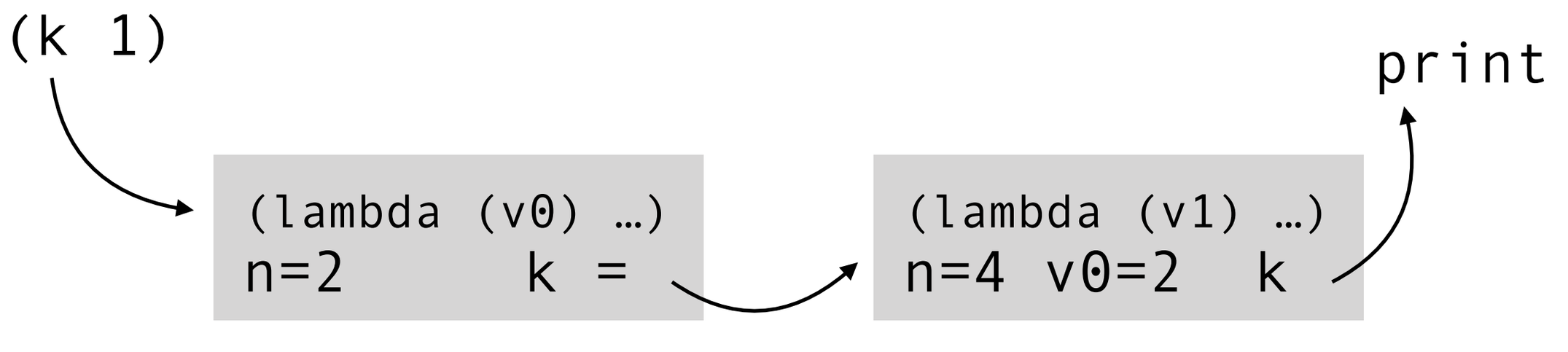
CPS

```
(define (fib n k)
  (let ([c (<= n 1)])
    (if c
        (k n)
        (let ([n-1 (- n 1)])
          (fib n-1
            (lambda (v0)
              (let ([n-2 (- n 2)])
                (fib n-2
                  (lambda (v1)
                    (let ([s (+ v0 v1)])
                      (k s)))))))))))
```



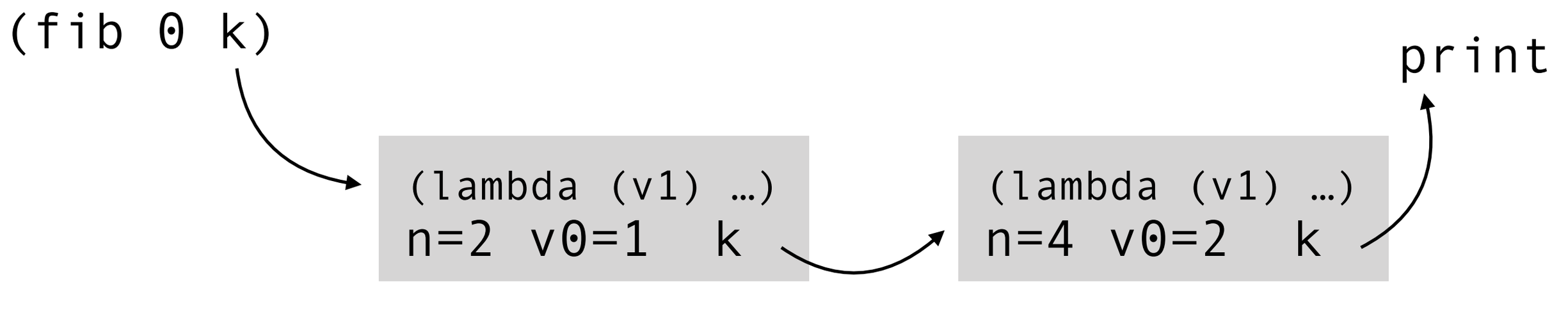
CPS

```
(define (fib n k)
  (let ([c (<= n 1)])
    (if c
        (k n)
        (let ([n-1 (- n 1)])
          (fib n-1
            (lambda (v0)
              (let ([n-2 (- n 2)])
                (fib n-2
                  (lambda (v1)
                    (let ([s (+ v0 v1)])
                      (k s)))))))))))
```



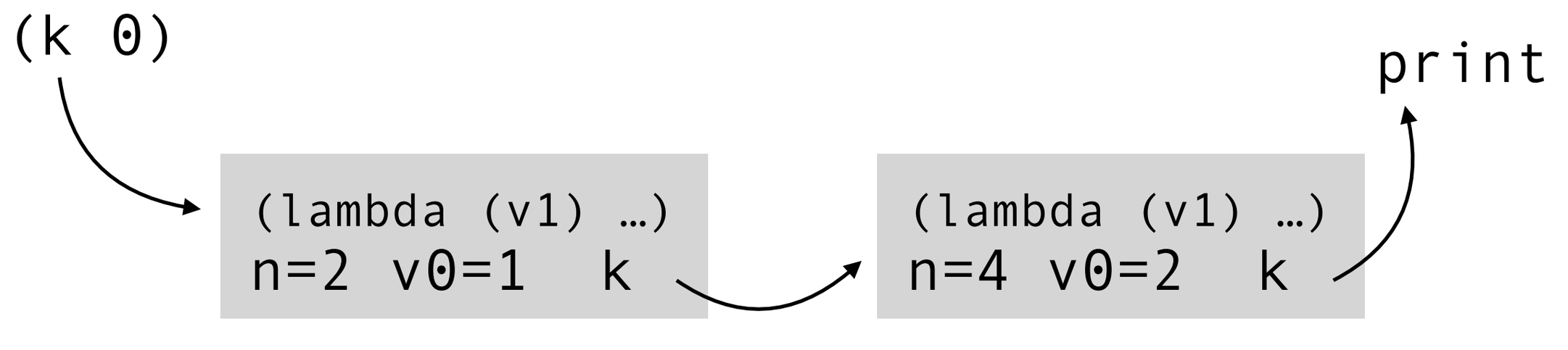
CPS

```
(define (fib n k)
  (let ([c (<= n 1)])
    (if c
        (k n)
        (let ([n-1 (- n 1)])
          (fib n-1
            (lambda (v0)
              (let ([n-2 (- n 2)])
                (fib n-2
                  (lambda (v1)
                    (let ([s (+ v0 v1)])
                      (k s)))))))))))
```



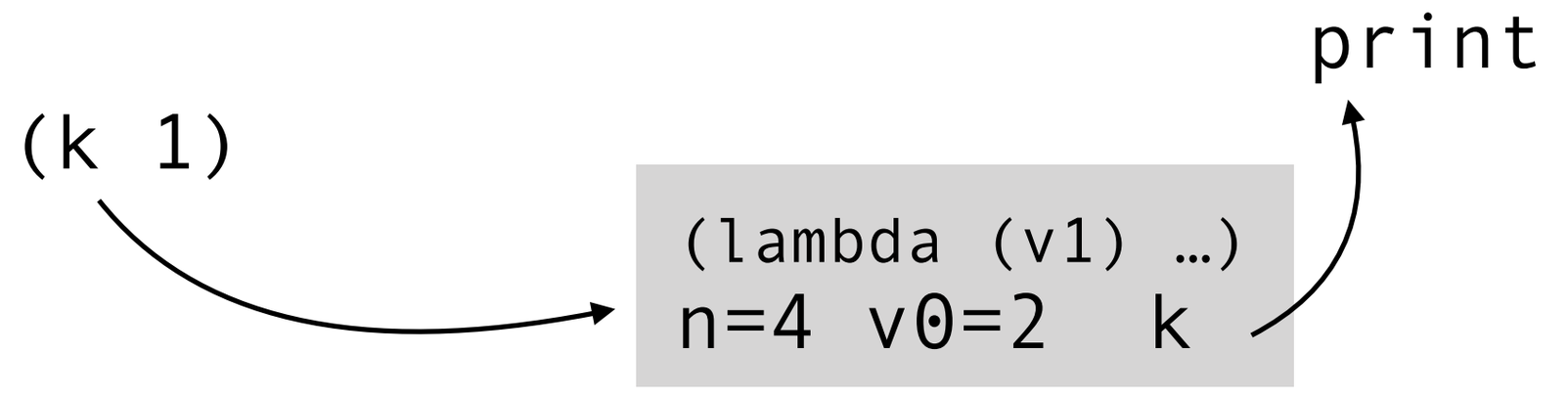
CPS

```
(define (fib n k)
  (let ([c (<= n 1)])
    (if c
        (k n)
        (let ([n-1 (- n 1)])
          (fib n-1
            (lambda (v0)
              (let ([n-2 (- n 2)])
                (fib n-2
                  (lambda (v1)
                    (let ([s (+ v0 v1)])
                      (k s)))))))))))
```



CPS

```
(define (fib n k)
  (let ([c (<= n 1)])
    (if c
        (k n)
        (let ([n-1 (- n 1)])
          (fib n-1
            (lambda (v0)
              (let ([n-2 (- n 2)])
                (fib n-2
                  (lambda (v1)
                    (let ([s (+ v0 v1)])
                      (k s)))))))))))
```



CPS

```
(define (fib n k)
  (let ([c (<= n 1)])
    (if c
        (k n)
        (let ([n-1 (- n 1)])
          (fib n-1
            (lambda (v0)
              (let ([n-2 (- n 2)])
                (fib n-2
                  (lambda (v1)
                    (let ([s (+ v0 v1)])
                      (k s)))))))))))
```