```
let rec compile_expr e si env (tc : bool) =
  match e with
    | EApp(fname, arg) -> (* single argument *)
      let argis = compile_expr arg si env false
      if tc then
        (* tail calling convention *)
      else
        (* normal calling convention *)
    | EIf(cond, thn, els) ->
      let condis = compile_expr cond si env
      let thnis = compile_expr thn si env
      let elsis = compile_expr els si env
      (* check tags, condition, jmp, etc *)
    | EPrim1(arg) ->
      let argis = compile_expr arg si env
      (* do prim1 op *)
```

false
tc
tc
false

What should the tc argument be?

A: false
B: true
C: tc

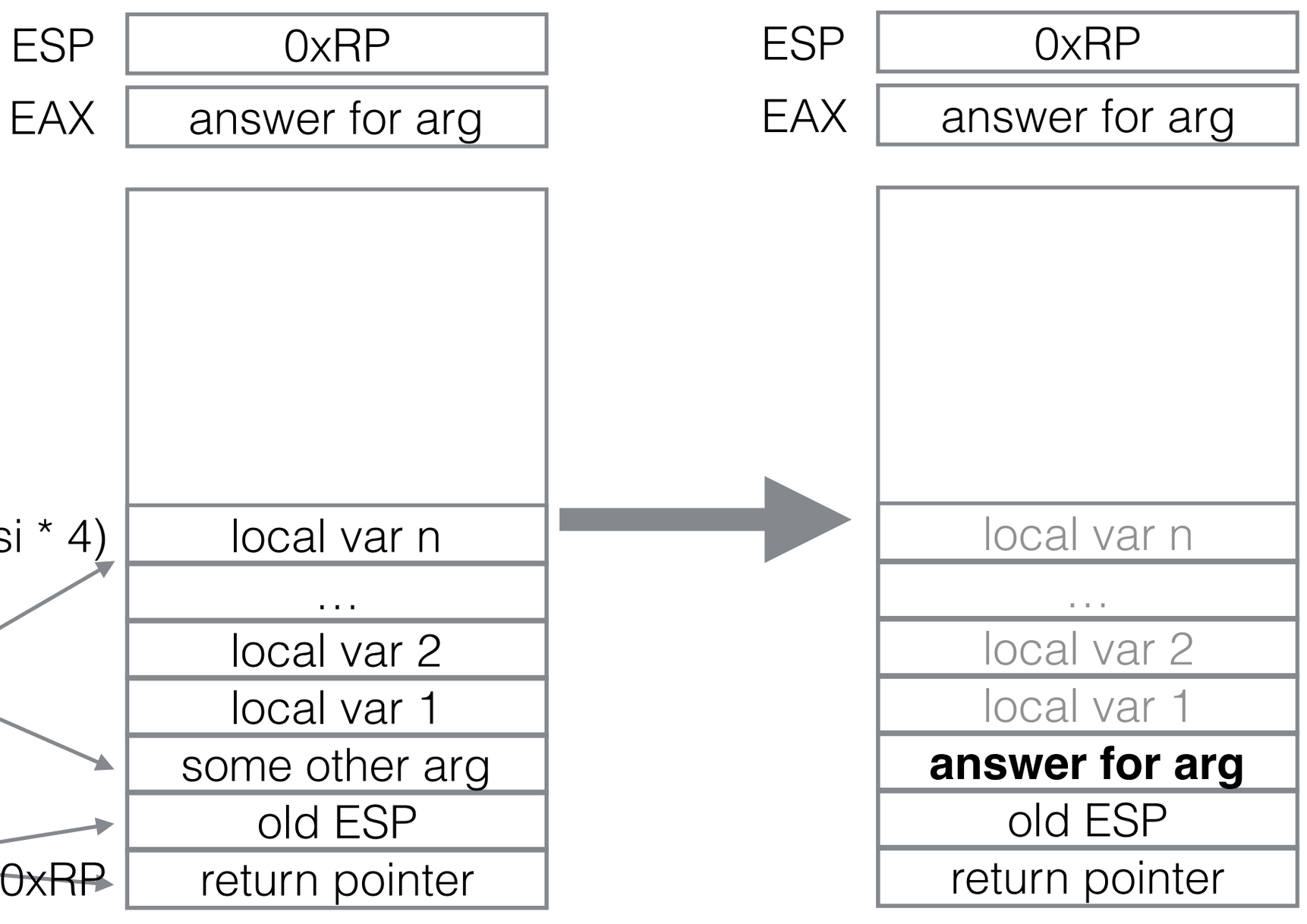
```
let rec compile_decl d =
  match d with
    | Decl(name, args, body) ->
    … compile_expr body env si true …
```

tc vs true

```
def f1(x):
  add1(if x: g() else: h())

def f2(x):
  if x: g() else: h()
```

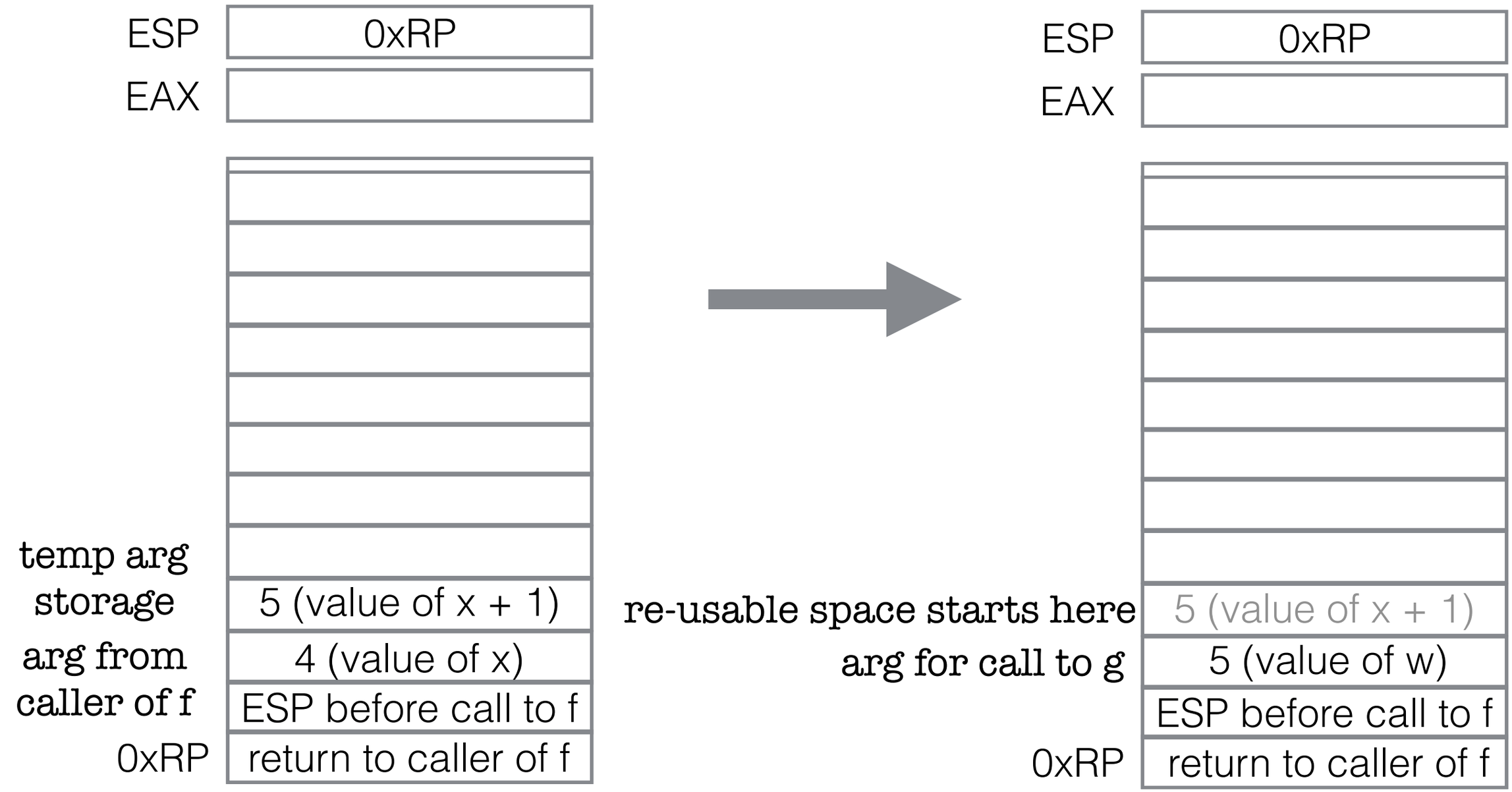
```
type expr =
  | ENumber of int
  | EBool of bool
  | EIf of expr * expr * expr
  | EApp of string * expr
  | EPrim1 of prim1 * expr
```

```
let rec compile_expr e si env (tc : bool) =
  match e with
    | EApp(fname, arg) ->
      let argis = compile_expr arg si env false
      if tc then
        [ IMov(RegOffset(-8, ESP), Reg(EAX))
        ; IJmp(fname)]
```

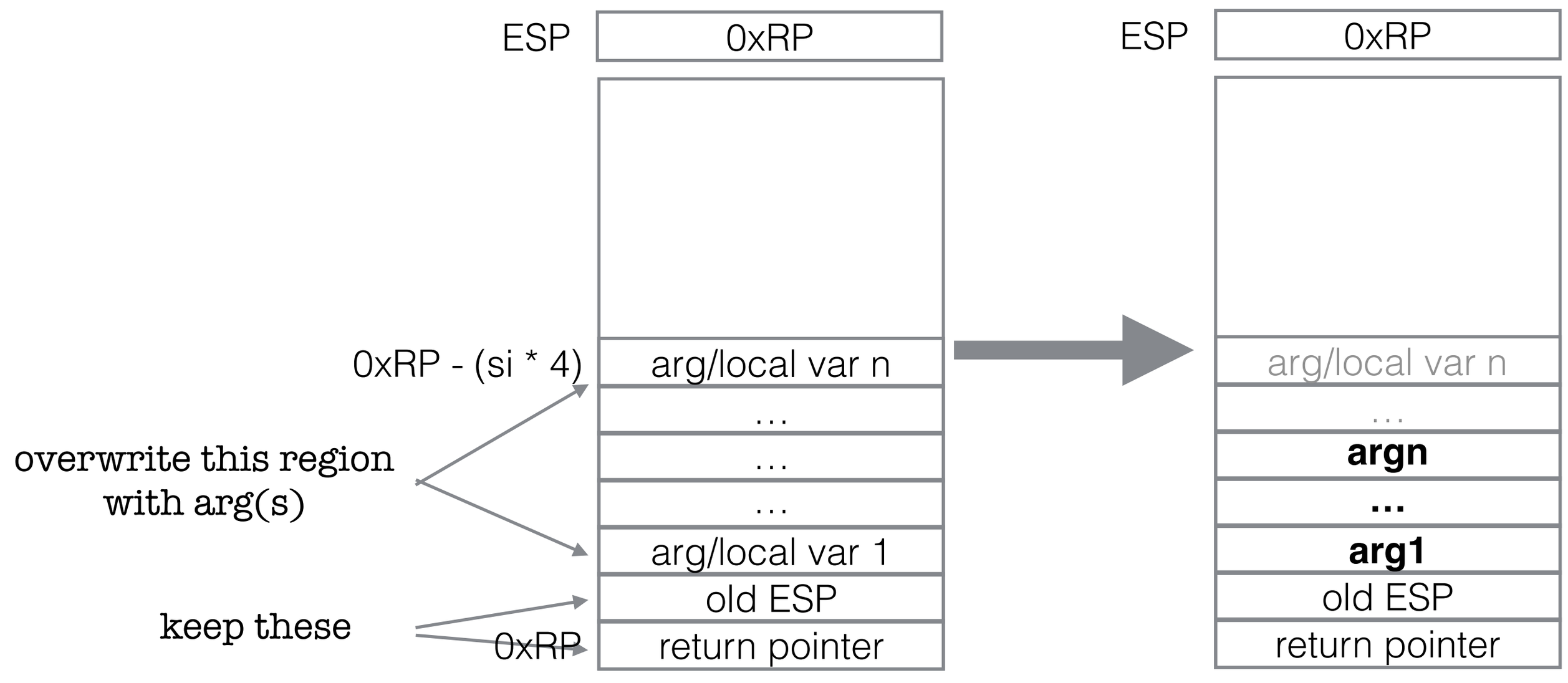


```
let rec compile_expr e si env (tc : bool) =
  match e with
    | EApp(fname, arg) ->
      let argis = compile_expr arg si env false
      if tc then
        [ IMov(RegOffset(-8, ESP), Reg(EAX))
        ; IJmp(fname)]
```

```
def f(x):
  g(x + 1)

def g(w):
  w * 2

f(4)
```

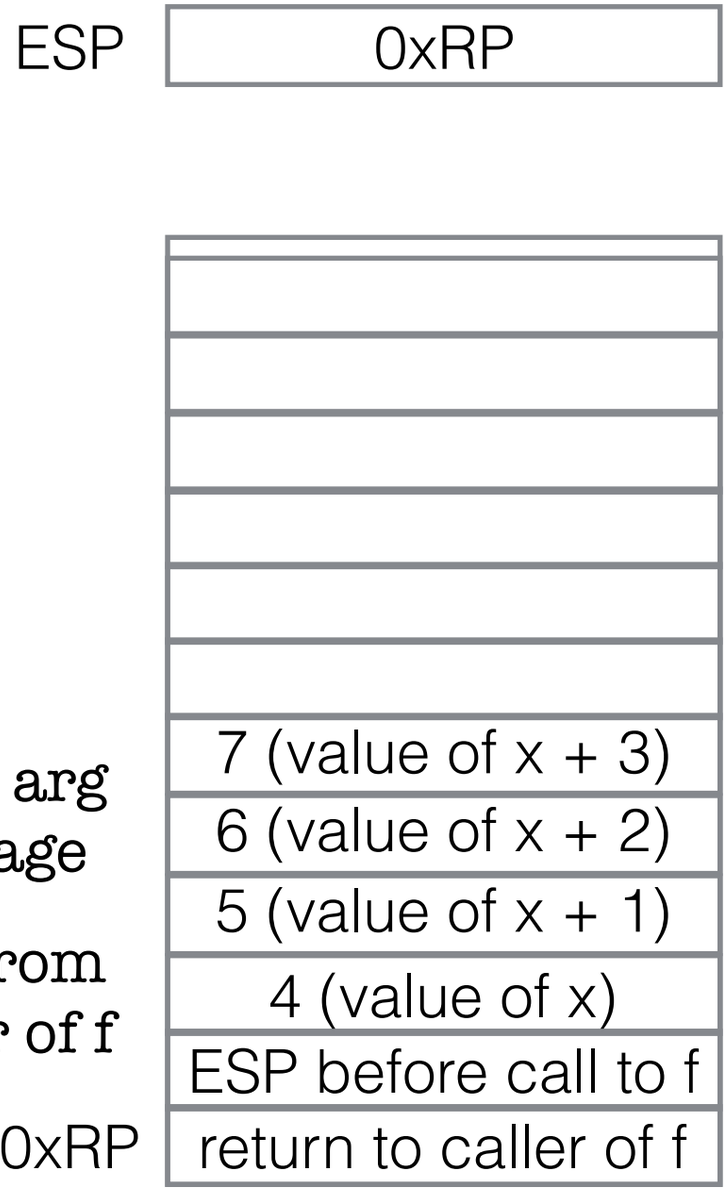


```
let rec compile_expr e si env (tc : bool) =
  match e with
    | EApp(fname, args) ->
      let argis = (* compile all args *) in
      if tc then
        [ (* move each arg answer to space above old ESP *)
        ; IJmp(fname)]
```



```
def f(x):
  g(x + 1, x + 2, x + 3)

def g(w, y, z):
  w * y * z

f(4)
```

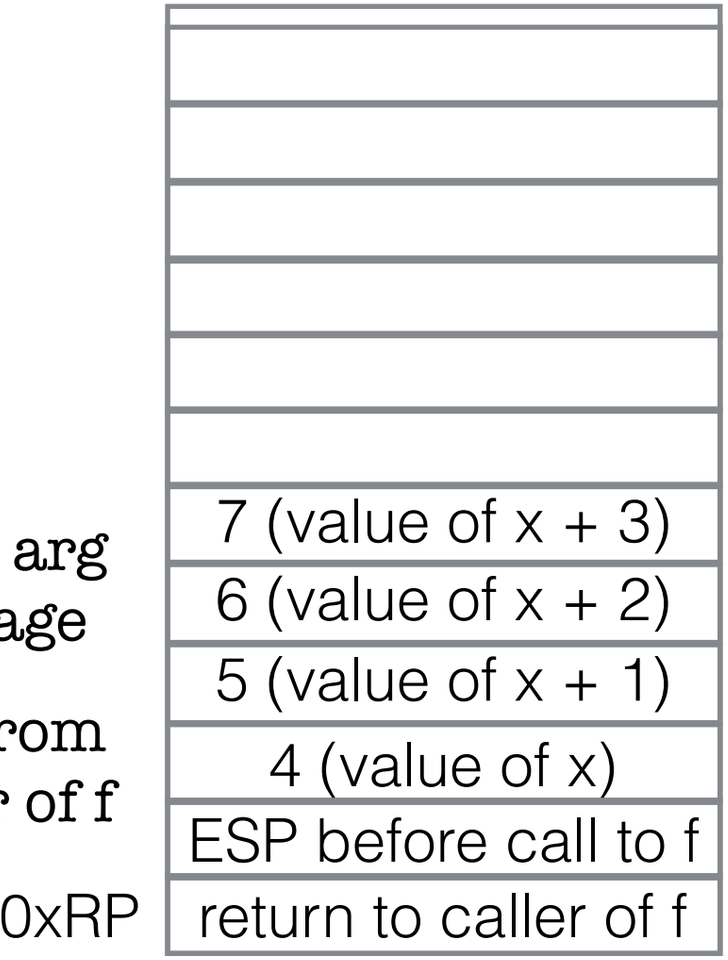
Before call to g



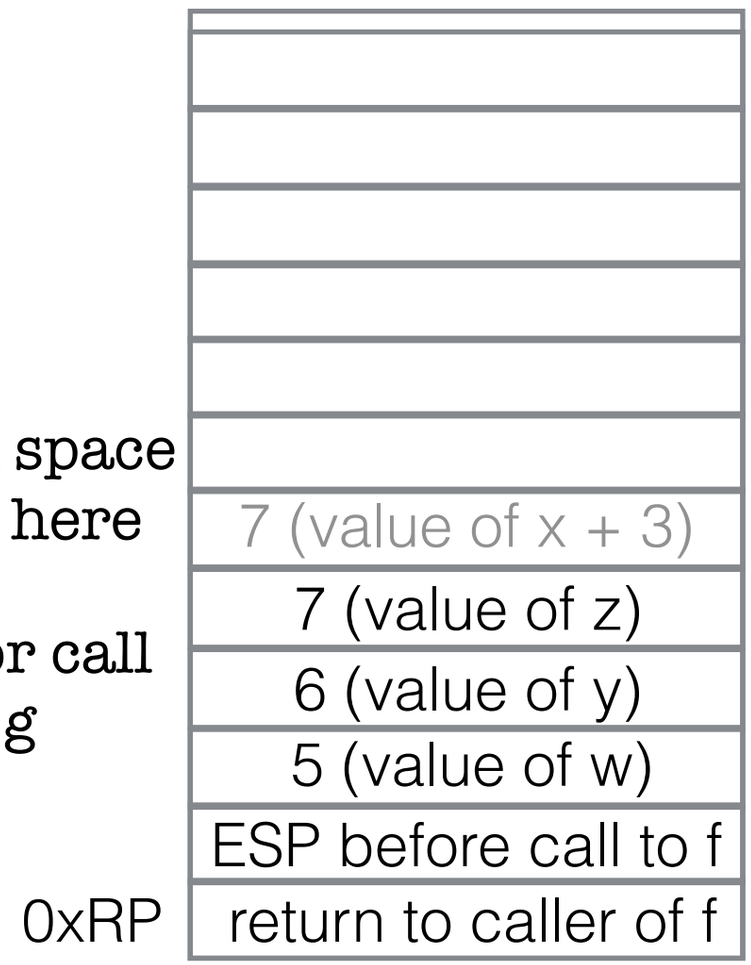
When starting g

```
def f(x):
  g(x + 1, x + 2, x + 3)

def g(w, y, z):
  w * y * z

f(4)
```

Before call to g

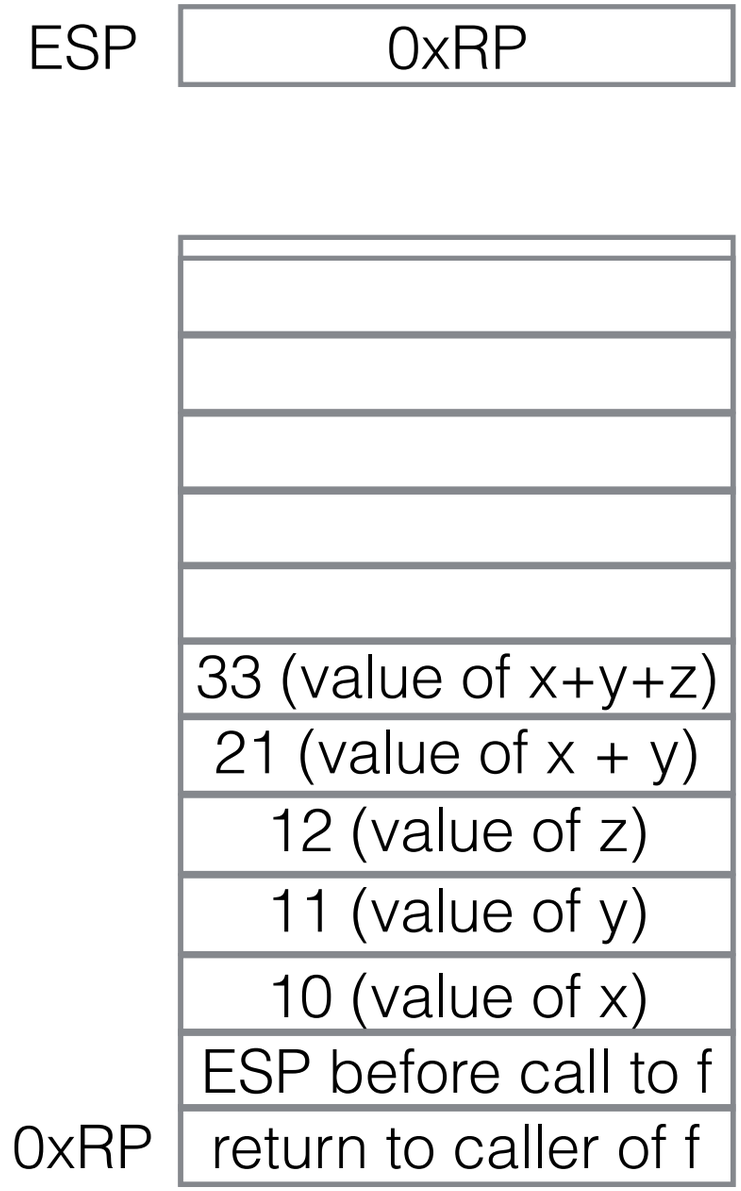


When starting g



```
def f(x, y, z):
  g(x + y + z)

def g(w):
  w * 2

f(10, 11, 12)
```

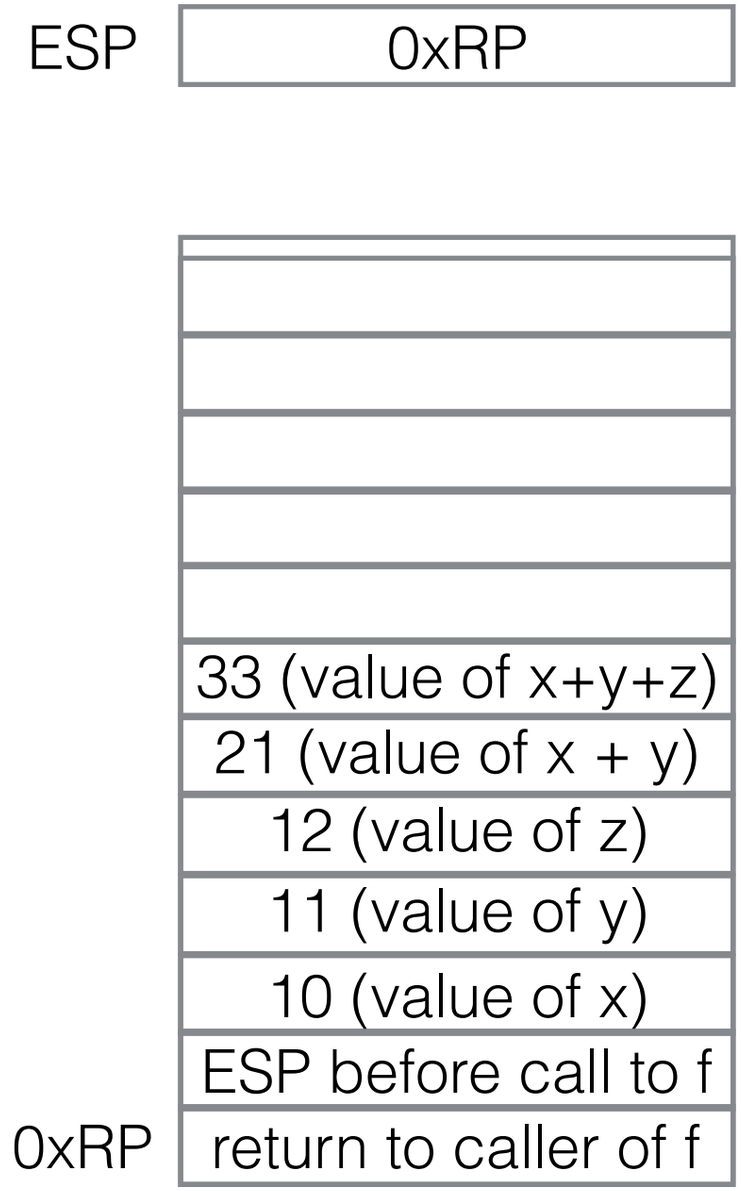
Before call to g



When starting g

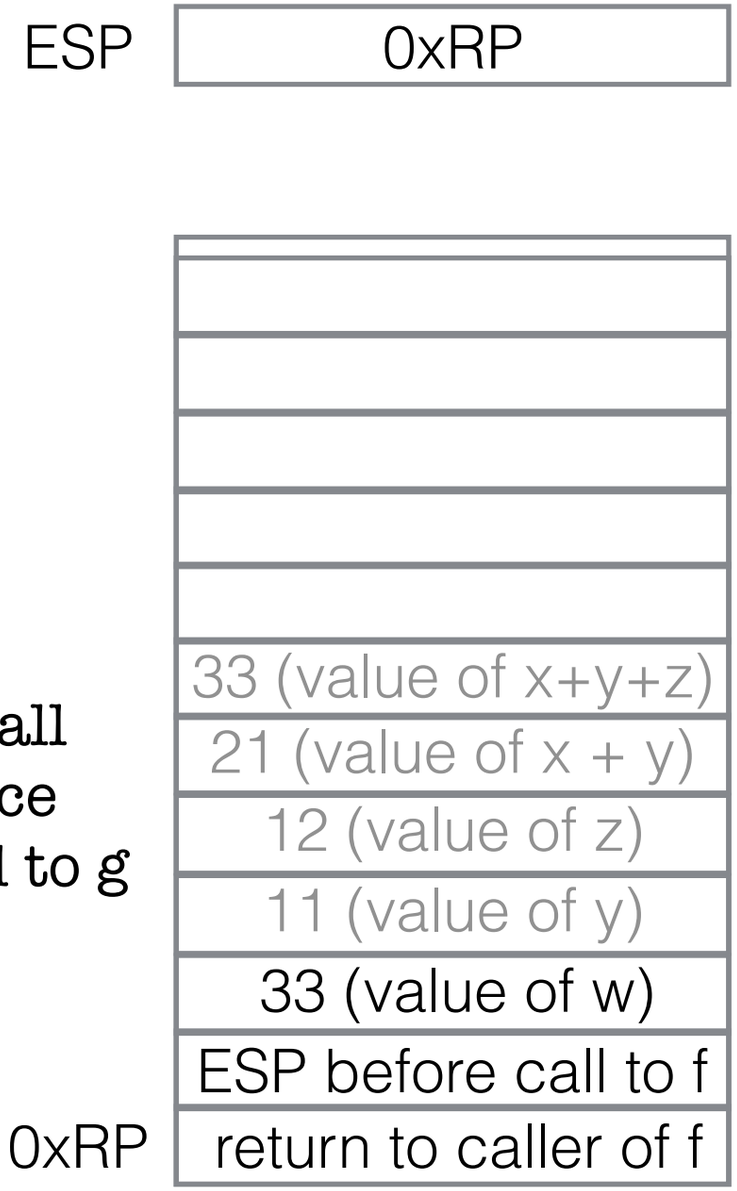
```
def f(x, y, z):
  g(x + y + z)

def g(w):
  w * 2

f(10, 11, 12)
```

Before call to g

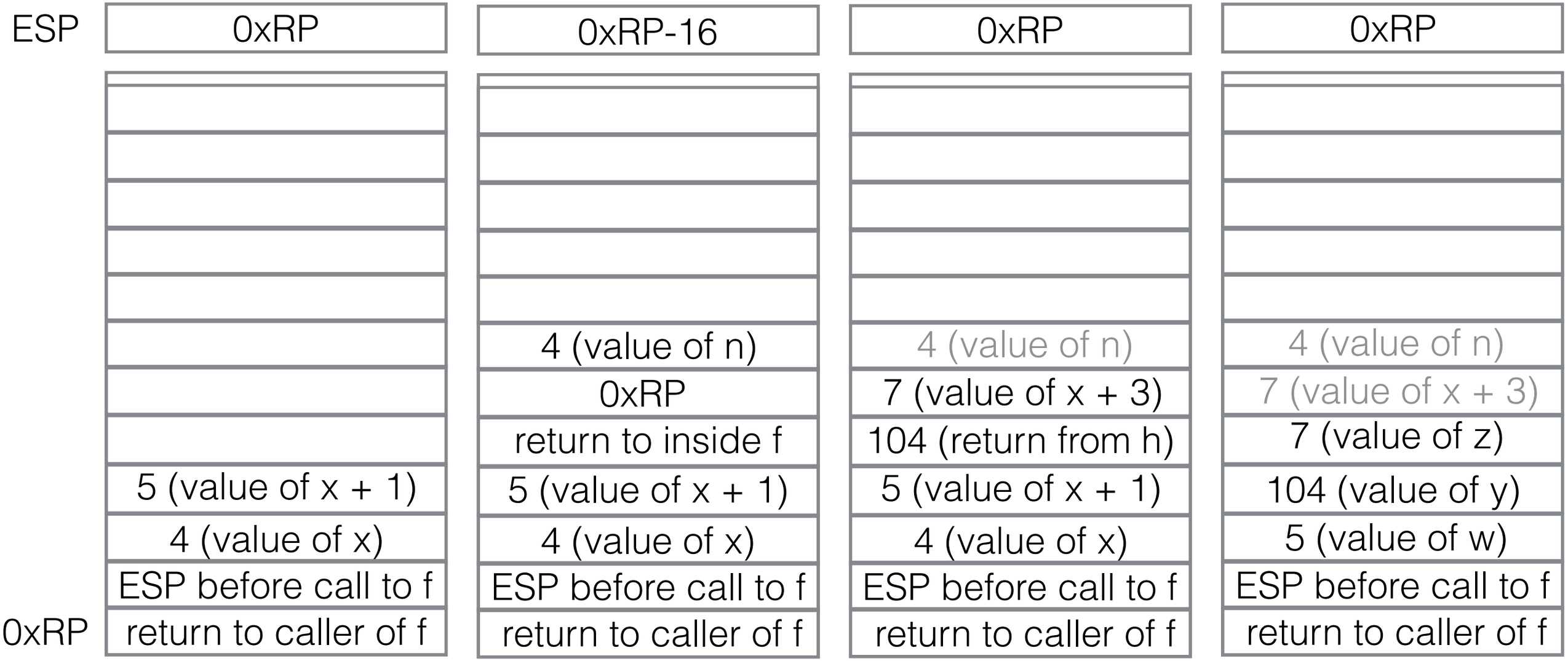


reclaim all this space during call to g

When starting g

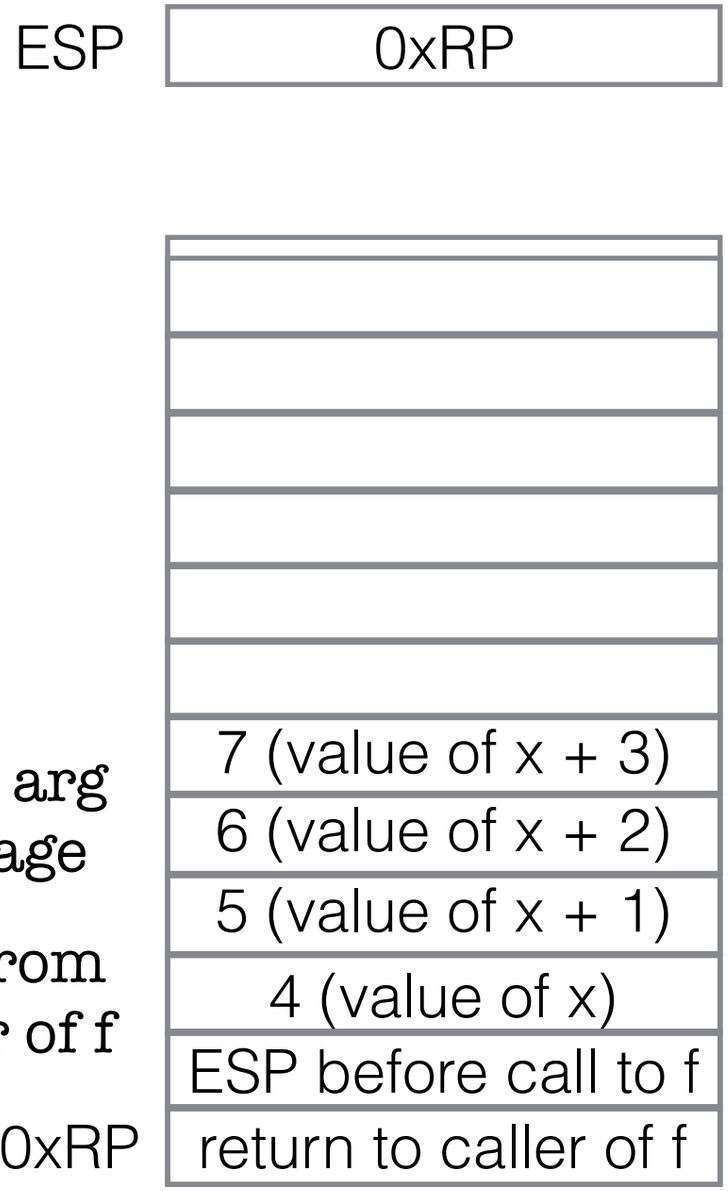


```
def f(x):
  g(x + 1, h(x), x + 3)

def g(w, y, z):
  w * y * z

def h(n):
  n + 100

f(4)
```



```
def f(x):
  g(x + 1, x + 2, x + 3)

def g(w, y, z):
  w * y * z

f(4)
```

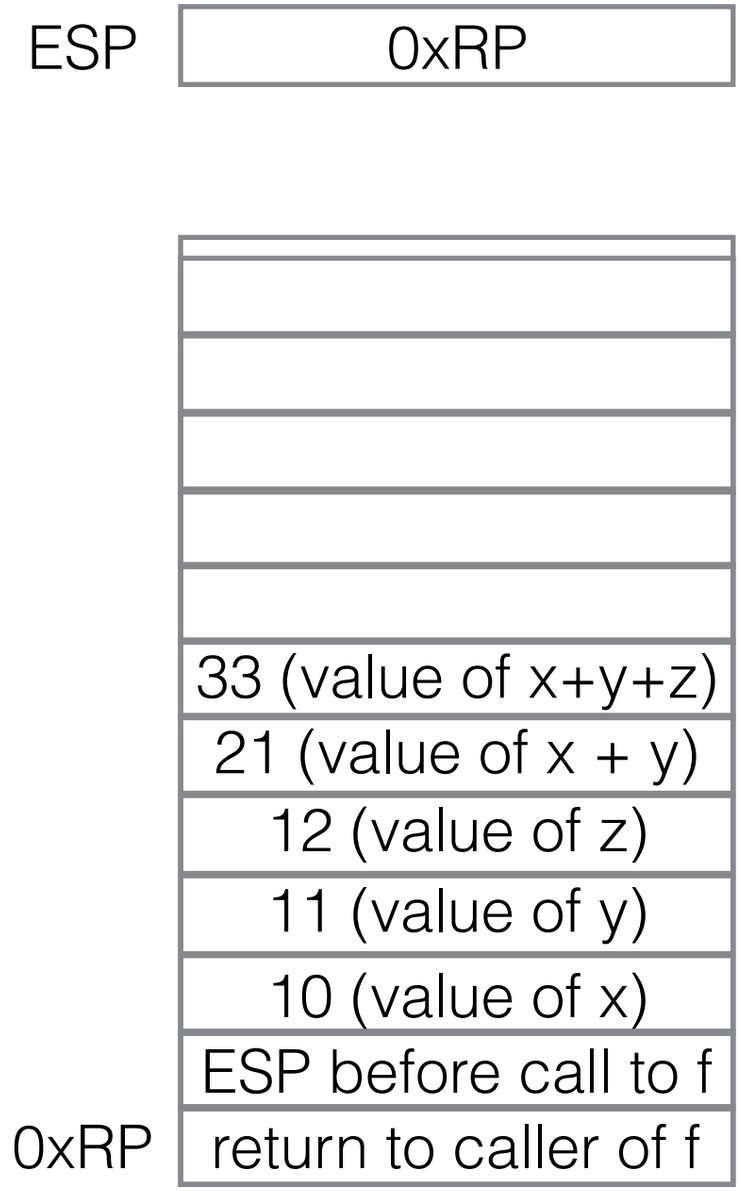
Before call to g



When starting g

```
def f(x, y, z):
  g(x + y + z)

def g(w):
  w * 2

f(10, 11, 12)
```

Before call to g



When starting g