FOX

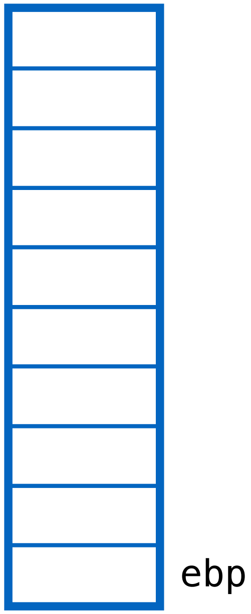
Garbage Collection

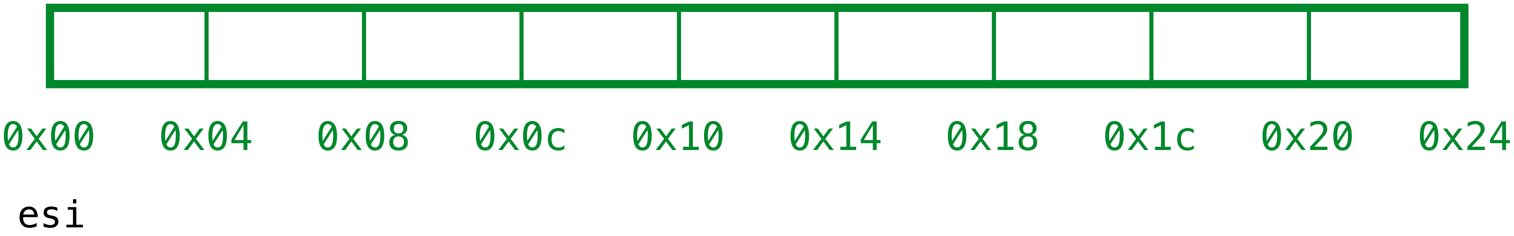
FOX / GC

Example 1

ex1: garbage at end

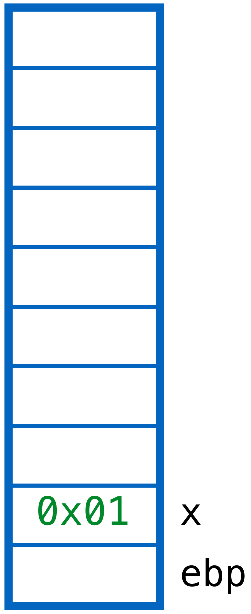
```
let x  = (1, 2)
  , y  = let tmp = (10, 20)
         in tmp[0] + tmp[1]
  , p0 = x[0] + y
  , p1 = x[1] + y
in
  (p0, p1)
```

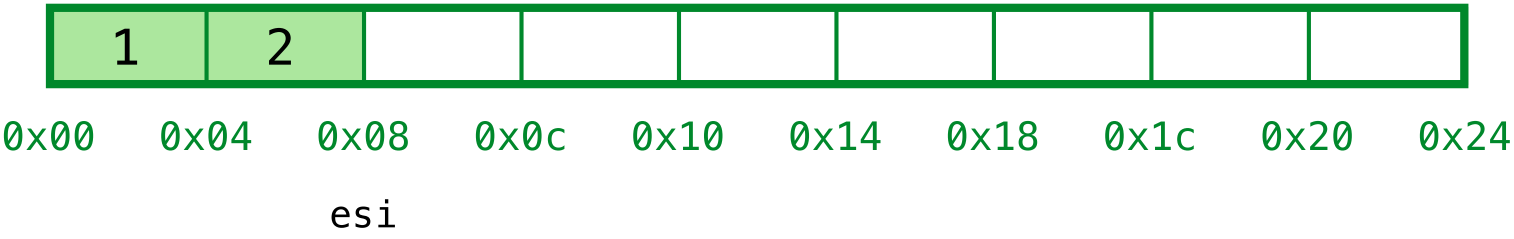




ex1: garbage at end

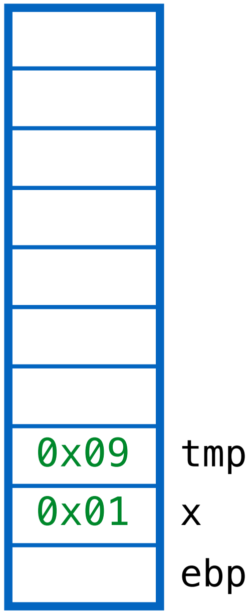
```
let x  = (1, 2)
  , y  = let tmp = (10, 20)
         in tmp[0] + tmp[1]
  , p0 = x[0] + y
  , p1 = x[1] + y
in
  (p0, p1)
```





ex1: garbage at end

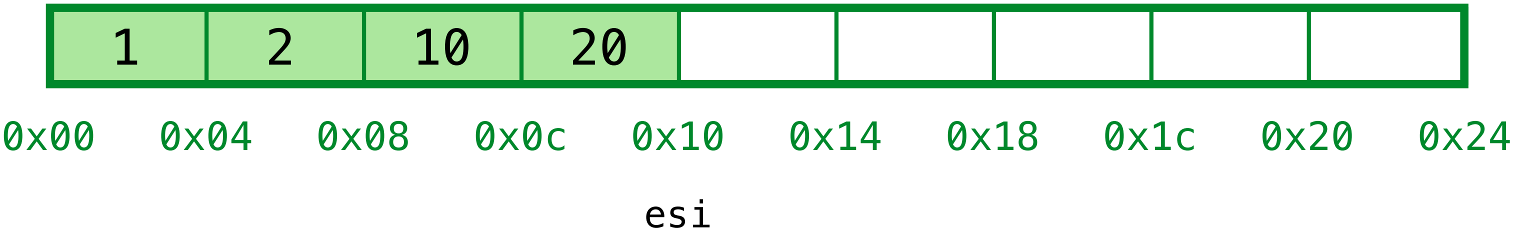
```
let x  = (1, 2)
  , y  = let tmp = (10, 20)
         in tmp[0] + tmp[1]
  , p0 = x[0] + y
  , p1 = x[1] + y
in
  (p0, p1)
```



0x09 tmp

0x01 x

ebp



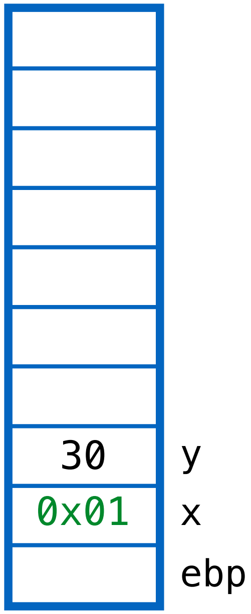
1 2 10 20

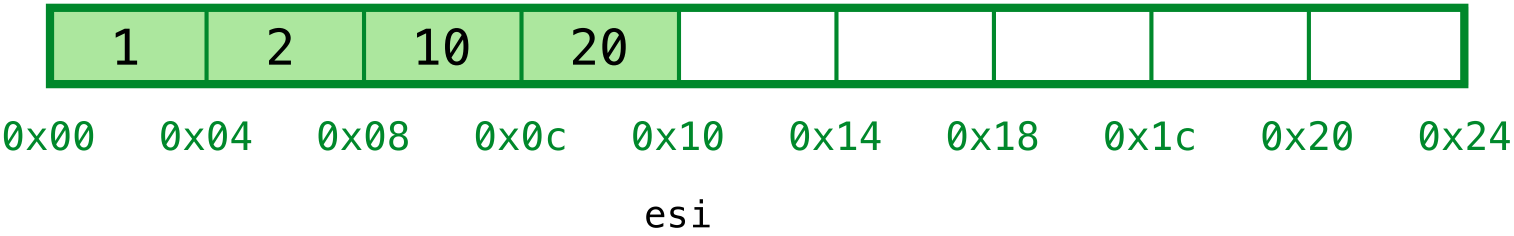
0x00 0x04 0x08 0x0c 0x10 0x14 0x18 0x1c 0x20 0x24

esi

ex1: garbage at end

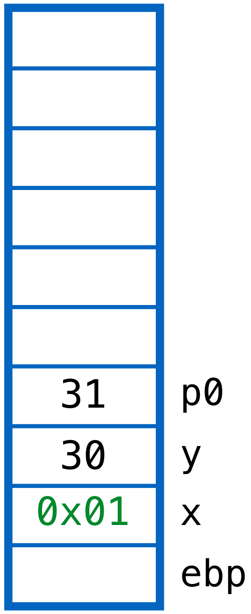
```
let x  = (1, 2)
  , y  = let tmp = (10, 20)
         in tmp[0] + tmp[1]
  , p0 = x[0] + y
  , p1 = x[1] + y
in
  (p0, p1)
```



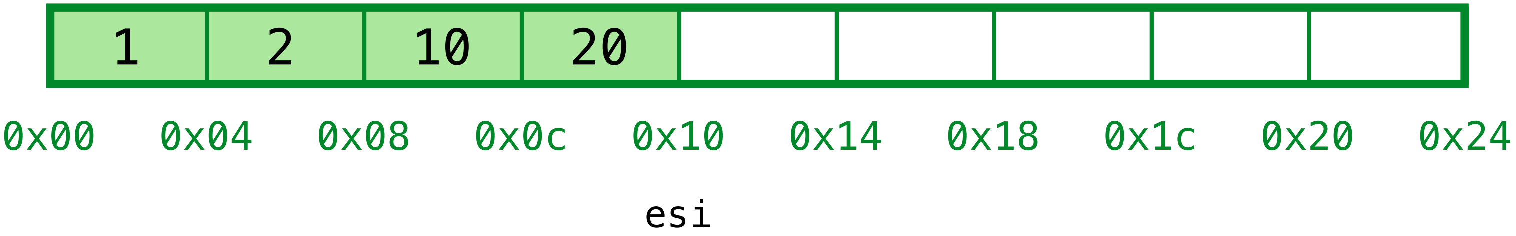


ex1: garbage at end

```
let x  = (1, 2)
  , y  = let tmp = (10, 20)
         in tmp[0] + tmp[1]
  , p0 = x[0] + y
  , p1 = x[1] + y
in
  (p0, p1)
```



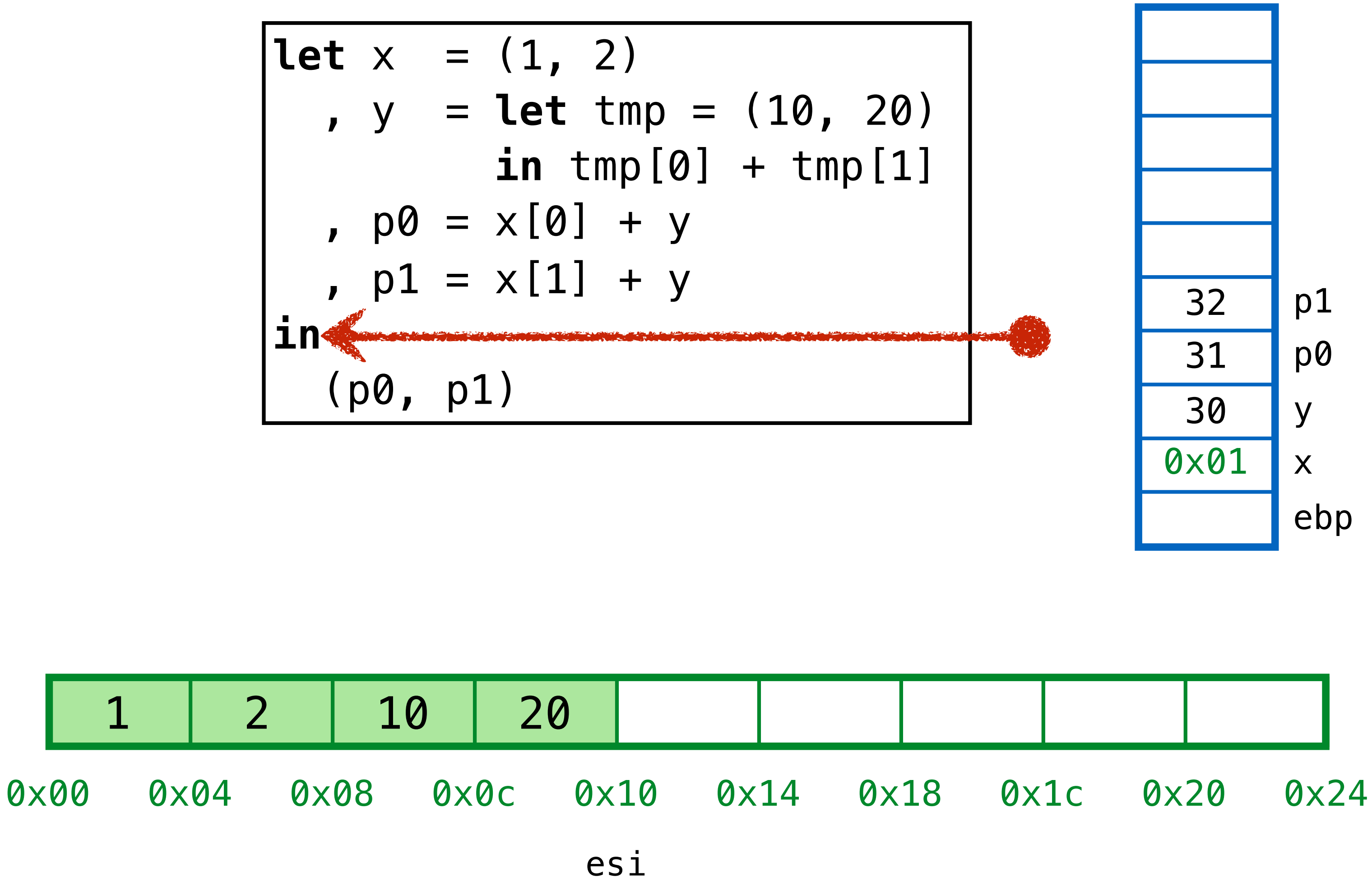
31	p0
30	y
0x01	x
	ebp



0x00	0x04	0x08	0x0c	0x10	0x14	0x18	0x1c	0x20	0x24
1	2	10	20						
				esi					

ex1: garbage at end

```
let x  = (1, 2)
  , y  = let tmp = (10, 20)
         in tmp[0] + tmp[1]
  , p0 = x[0] + y
  , p1 = x[1] + y
in
  (p0, p1)
```



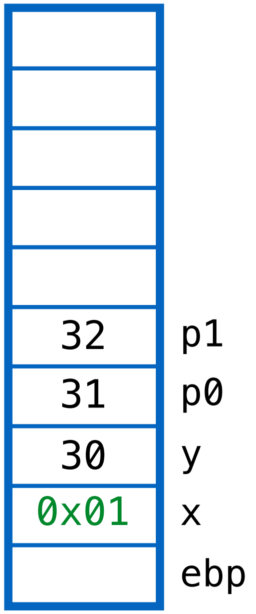
32	p1
31	p0
30	y
0x01	x
	ebp

0x00	0x04	0x08	0x0c	0x10	0x14	0x18	0x1c	0x20	0x24
1	2	10	20						
				esi					

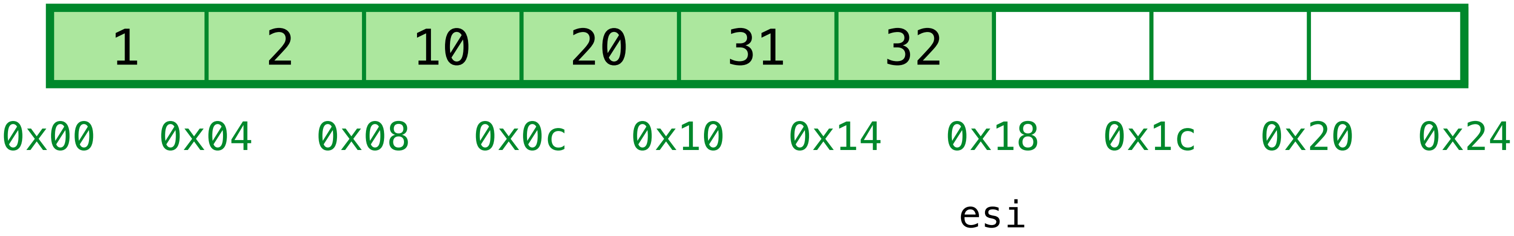
ex1: garbage at end

```
let x  = (1, 2)
  , y  = let tmp = (10, 20)
         in tmp[0] + tmp[1]
  , p0 = x[0] + y
  , p1 = x[1] + y
in
  (p0, p1)
```

Result (eax) = 0x11



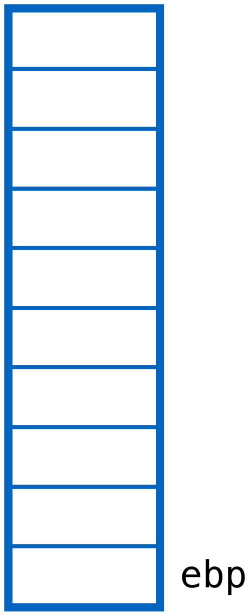
32	p1
31	p0
30	y
0x01	x
	ebp

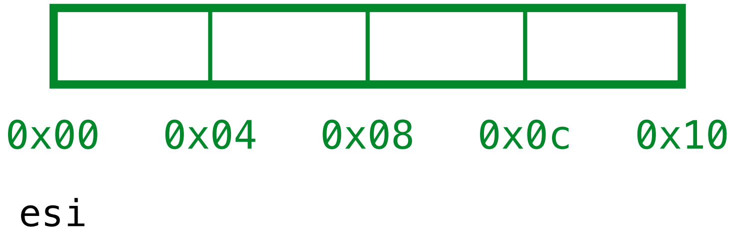


0x00	0x04	0x08	0x0c	0x10	0x14	0x18	0x1c	0x20
1	2	10	20	31	32			

ex1: garbage at end

```
let x  = (1, 2)
  , y  = let tmp = (10, 20)
         in tmp[0] + tmp[1]
  , p0 = x[0] + y
  , p1 = x[1] + y
in
  (p0, p1)
```

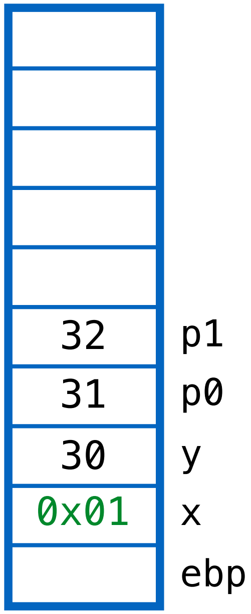




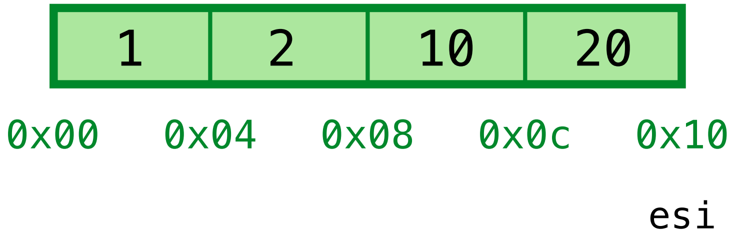
Suppose we had a smaller, 4-word heap

ex1: garbage at end

```
let x  = (1, 2)
  , y  = let tmp = (10, 20)
         in tmp[0] + tmp[1]
  , p0 = x[0] + y
  , p1 = x[1] + y
in
  (p0, p1)
```



32	p1
31	p0
30	y
0x01	x
	ebp



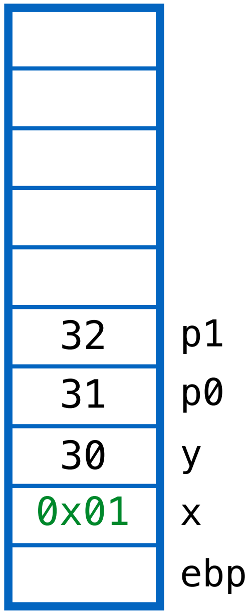
1	2	10	20

0x00 0x04 0x08 0x0c 0x10

esi

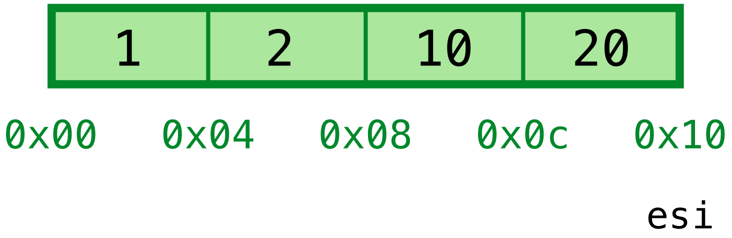
ex1: garbage at end

```
let x  = (1, 2)
  , y  = let tmp = (10, 20)
         in tmp[0] + tmp[1]
  , p0 = x[0] + y
  , p1 = x[1] + y
in
  (p0, p1)
```



32	p1
31	p0
30	y
0x01	x
	ebp

Out of memory!
Can't allocate **`(p0, p1)`**



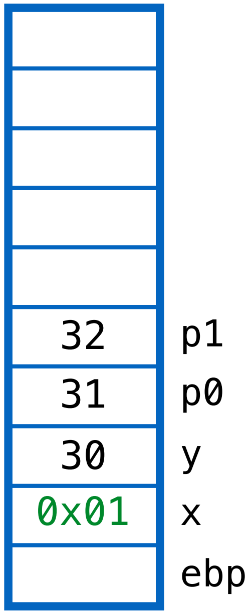
1	2	10	20

0x00 0x04 0x08 0x0c 0x10

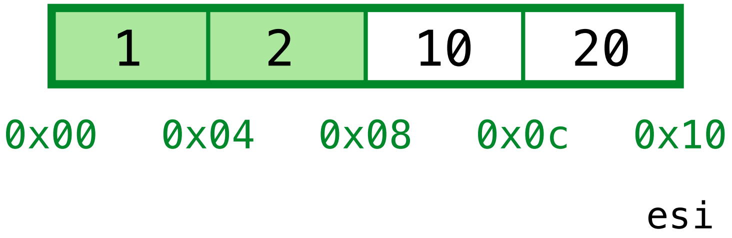
esi

ex1: garbage at end

```
let x  = (1, 2)
  , y  = let tmp = (10, 20)
         in tmp[0] + tmp[1]
  , p0 = x[0] + y
  , p1 = x[1] + y
in
  (p0, p1)
```



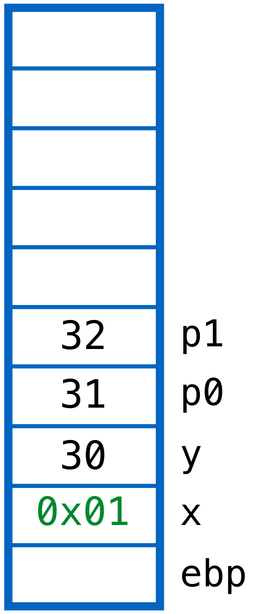
`(10, 20)` is “garbage”



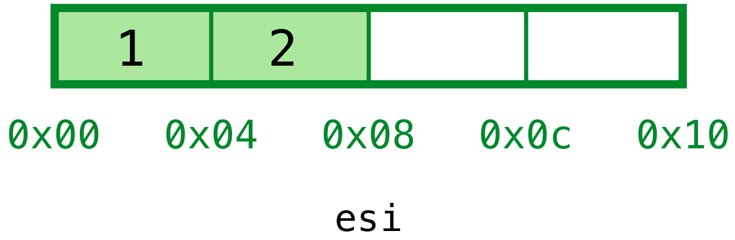
Q: How to determine if cell is garbage?

ex1: garbage at end

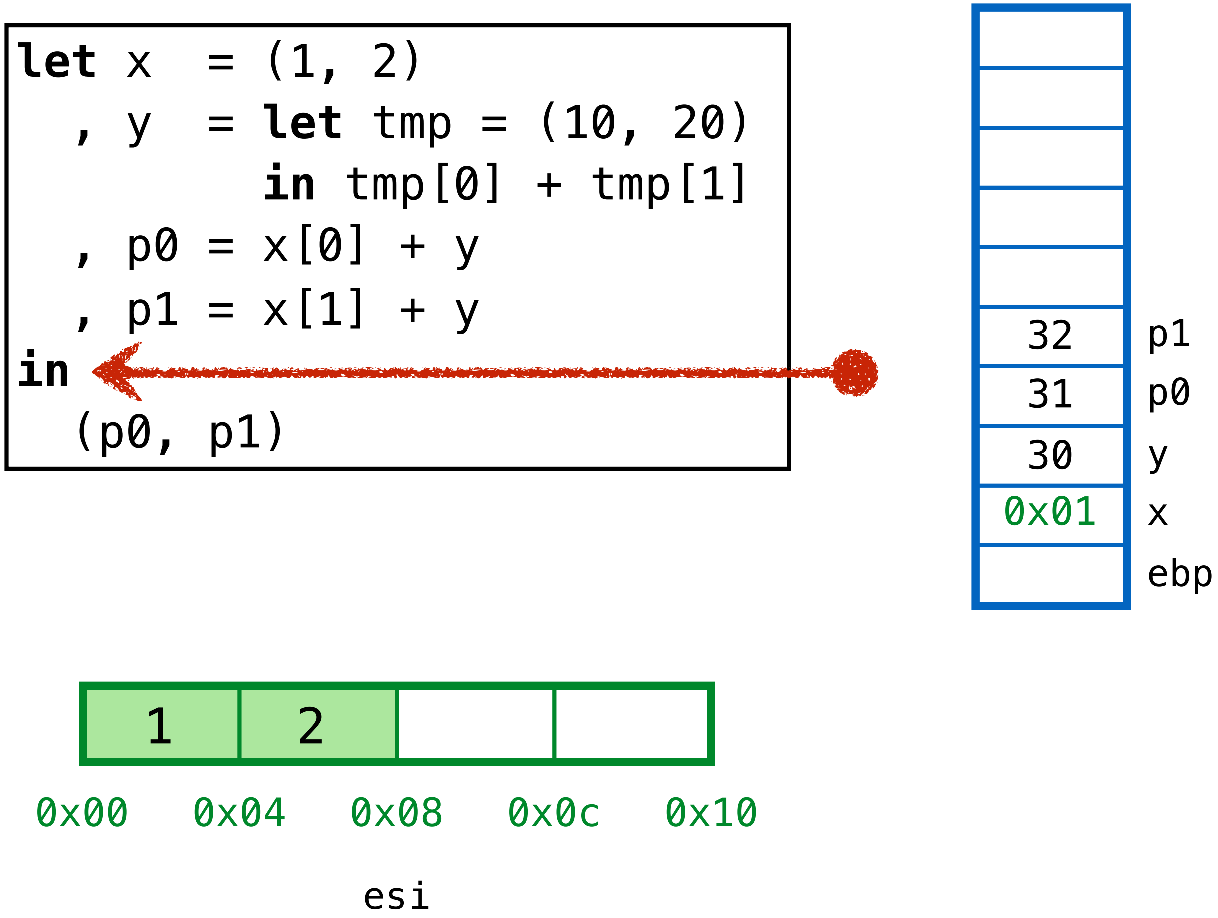
```
let x  = (1, 2)
  , y  = let tmp = (10, 20)
         in tmp[0] + tmp[1]
  , p0 = x[0] + y
  , p1 = x[1] + y
in
  (p0, p1)
```



(10, 20) is “garbage”

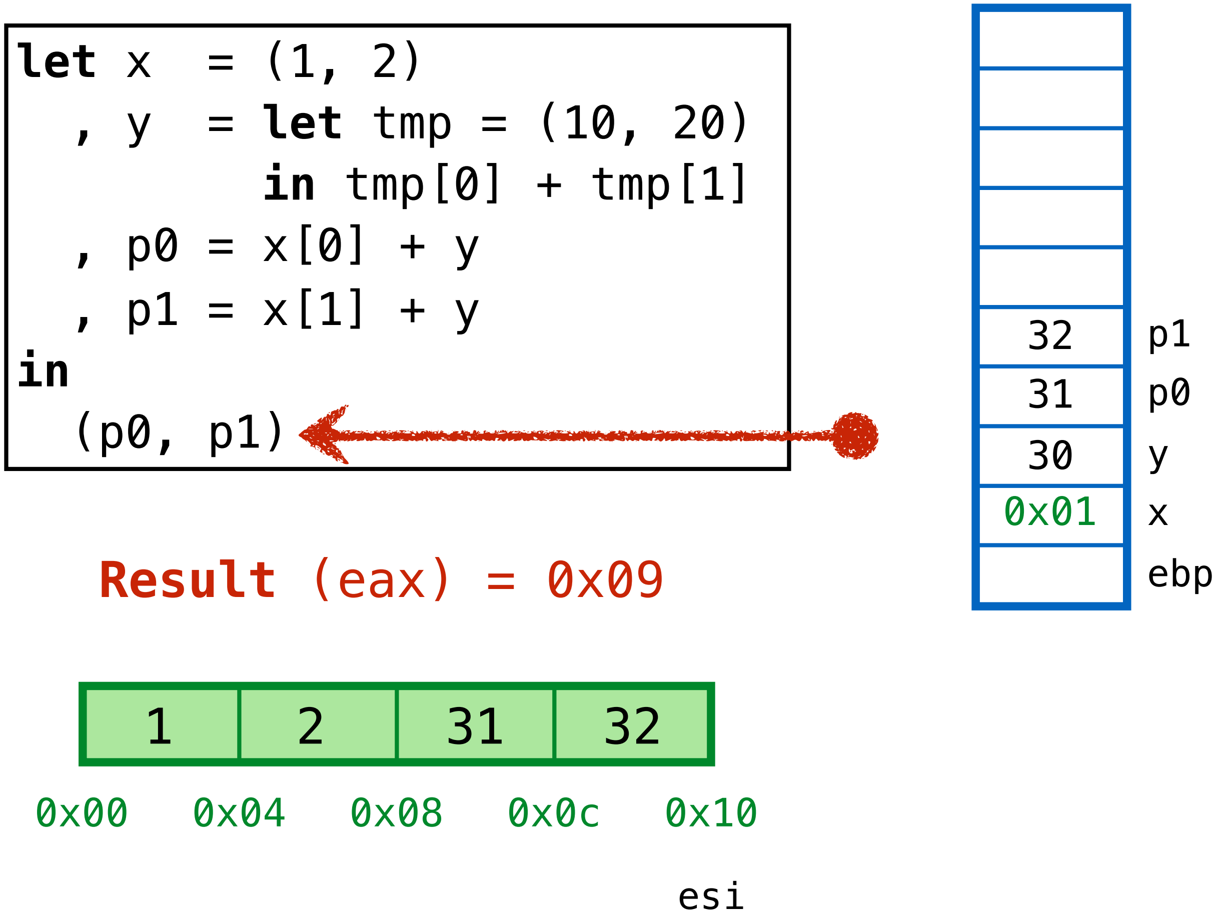


ex1: garbage at end



```
let x  = (1, 2)
  , y  = let tmp = (10, 20)
         in tmp[0] + tmp[1]
  , p0 = x[0] + y
  , p1 = x[1] + y
in
  (p0, p1)
```

ex1: garbage at end



```
let x  = (1, 2)
  , y  = let tmp = (10, 20)
         in tmp[0] + tmp[1]
  , p0 = x[0] + y
  , p1 = x[1] + y
in
  (p0, p1)
```

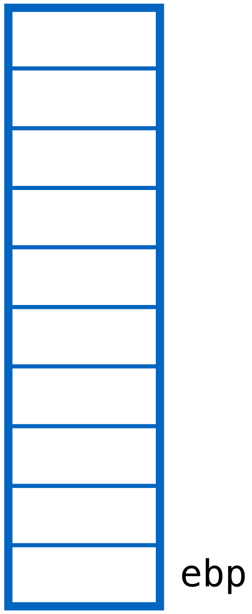
Result (eax) = 0x09

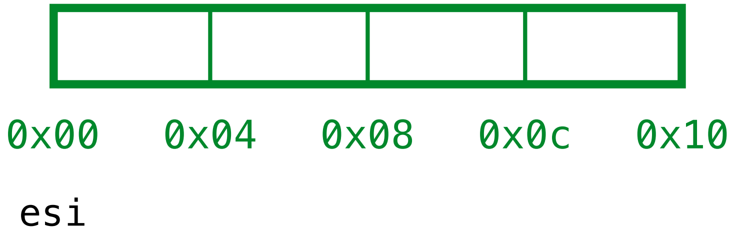
FOX / GC

Example 2

ex2: garbage in the middle

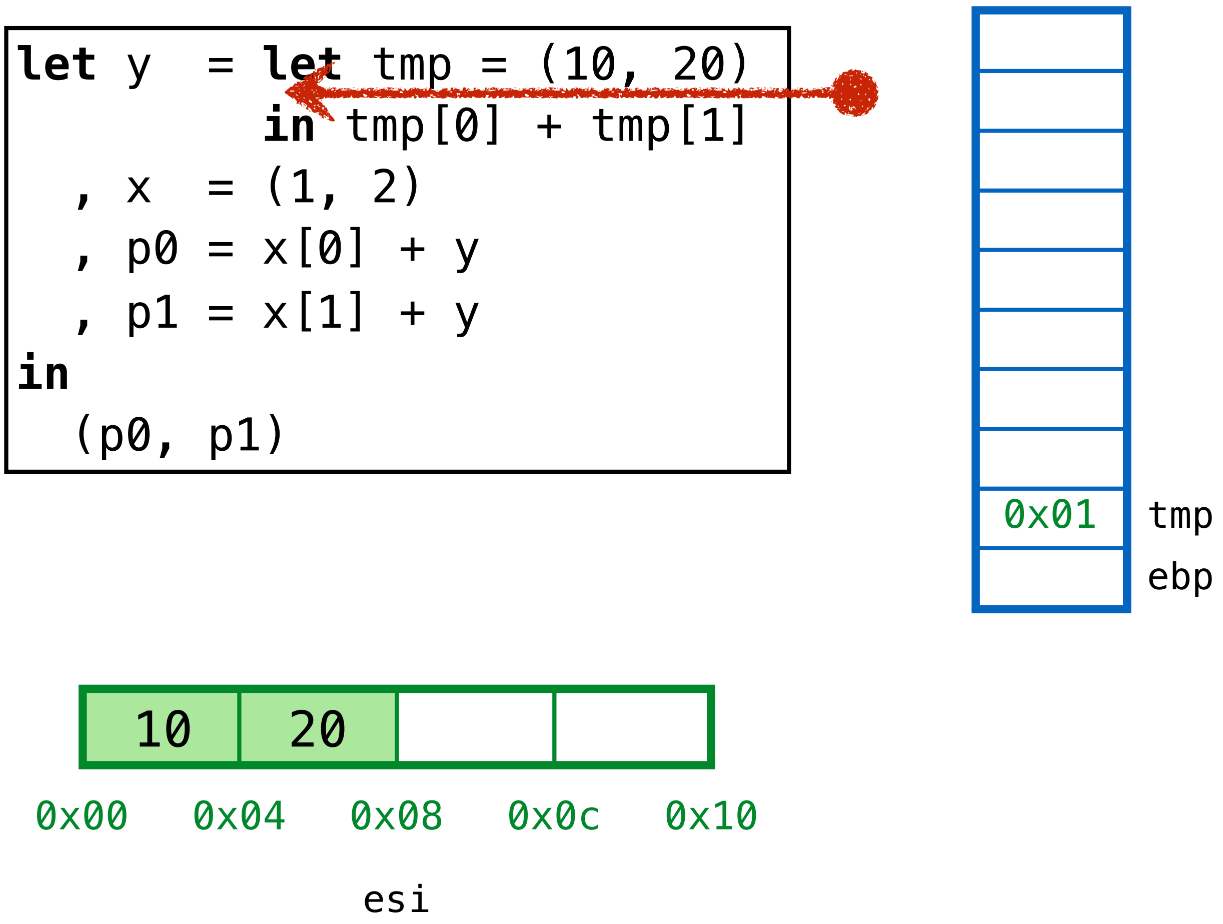
```
let y  = let tmp = (10, 20)
         in tmp[0] + tmp[1]
  , x  = (1, 2)
  , p0 = x[0] + y
  , p1 = x[1] + y
in
  (p0, p1)
```





Start with a 4-word heap

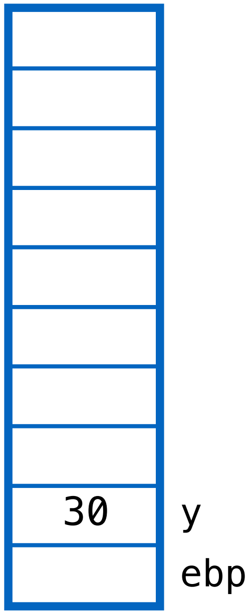
ex2: garbage in the middle

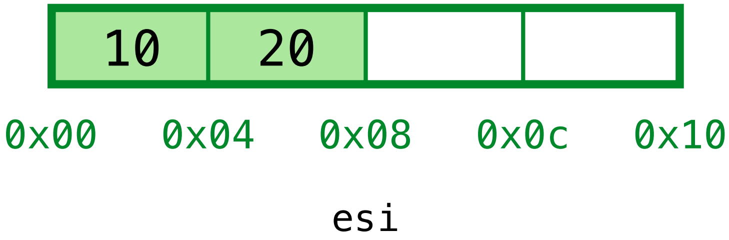


```
let y  = let tmp = (10, 20)
         in tmp[0] + tmp[1]
  , x  = (1, 2)
  , p0 = x[0] + y
  , p1 = x[1] + y
in
  (p0, p1)
```

ex2: garbage in the middle

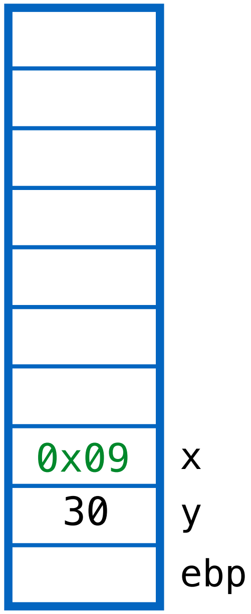
```
let y  = let tmp = (10, 20)
         in tmp[0] + tmp[1]
  , x  = (1, 2)
  , p0 = x[0] + y
  , p1 = x[1] + y
in
  (p0, p1)
```



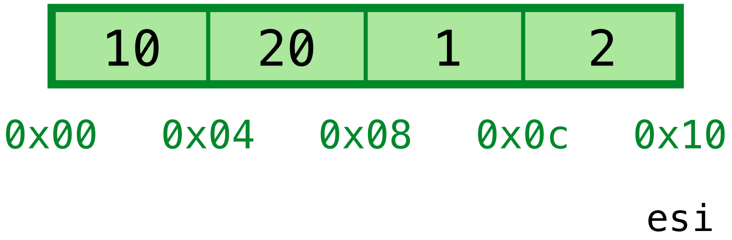


ex2: garbage in the middle

```
let y  = let tmp = (10, 20)
         in tmp[0] + tmp[1]
  , x  = (1, 2)
  , p0 = x[0] + y
  , p1 = x[1] + y
in
  (p0, p1)
```



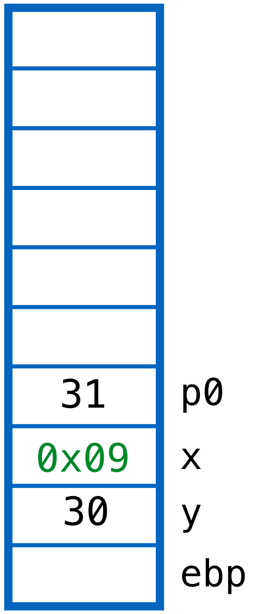
0x09	x
30	y
	ebp



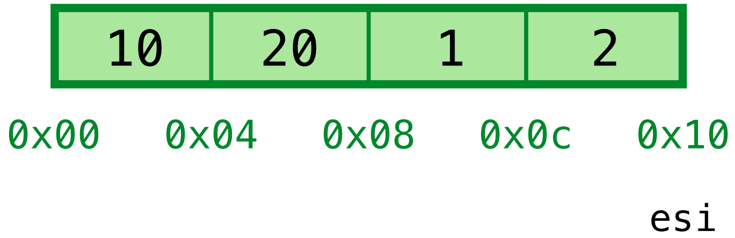
10	20	1	2	
0x00	0x04	0x08	0x0c	0x10
				esi

ex2: garbage in the middle

```
let y  = let tmp = (10, 20)
         in tmp[0] + tmp[1]
  , x  = (1, 2)
  , p0 = x[0] + y
  , p1 = x[1] + y
in
  (p0, p1)
```



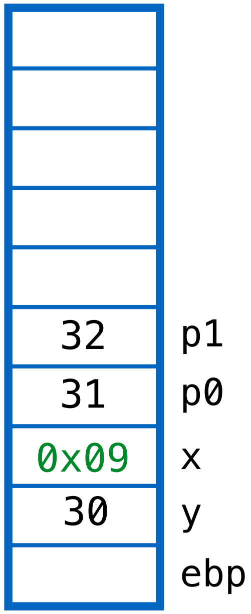
31	p0
0x09	x
30	y
	ebp



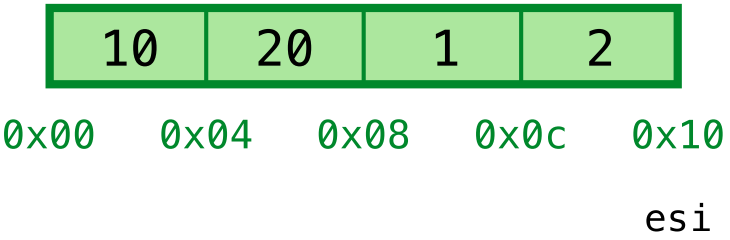
10	20	1	2	
0x00	0x04	0x08	0x0c	0x10
				esi

ex2: garbage in the middle

```
let y  = let tmp = (10, 20)
         in tmp[0] + tmp[1]
  , x  = (1, 2)
  , p0 = x[0] + y
  , p1 = x[1] + y
in
  (p0, p1)
```



32	p1
31	p0
0x09	x
30	y
	ebp

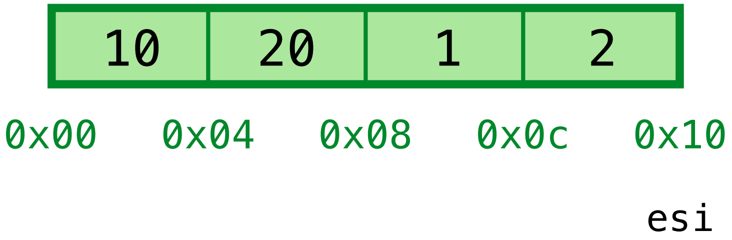


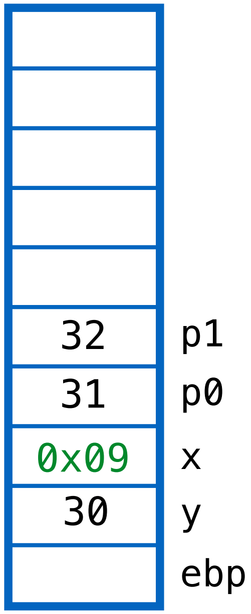
10	20	1	2	
0x00	0x04	0x08	0x0c	0x10
				esi

ex2: garbage in the middle

```
let y  = let tmp = (10, 20)
         in tmp[0] + tmp[1]
  , x  = (1, 2)
  , p0 = x[0] + y
  , p1 = x[1] + y
in
  (p0, p1)
```

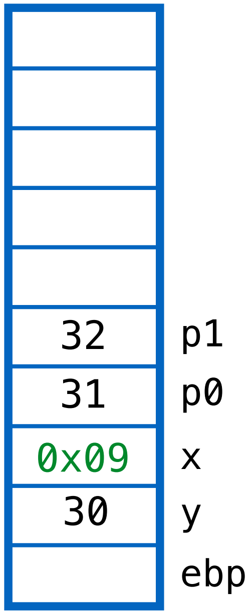
Out of memory!
Can’t allocate **`(p0, p1)`**



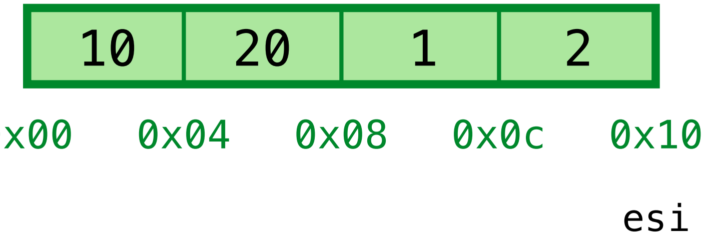


ex2: garbage in the middle

```
let y  = let tmp = (10, 20)
         in tmp[0] + tmp[1]
  , x  = (1, 2)
  , p0 = x[0] + y
  , p1 = x[1] + y
in
  (p0, p1)
```

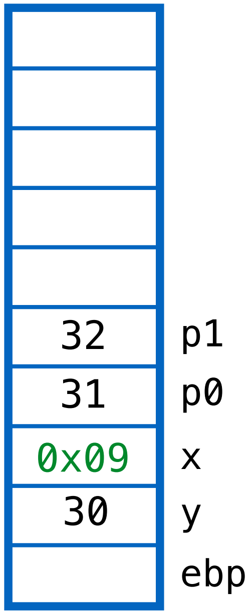


Lets reclaim & recycle garbage!



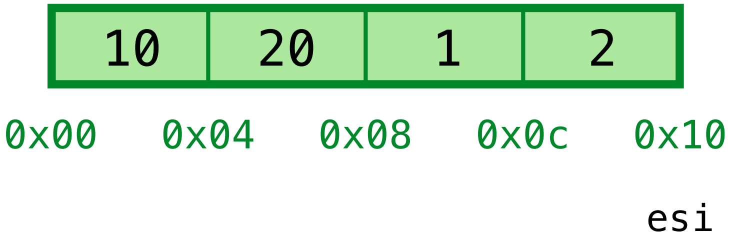
ex2: garbage in the middle

```
let y  = let tmp = (10, 20)
         in tmp[0] + tmp[1]
  , x  = (1, 2)
  , p0 = x[0] + y
  , p1 = x[1] + y
in
  (p0, p1)
```



32	p1
31	p0
0x09	x
30	y
	ebp

Lets reclaim & recycle garbage!



10	20	1	2

0x00 0x04 0x08 0x0c 0x10

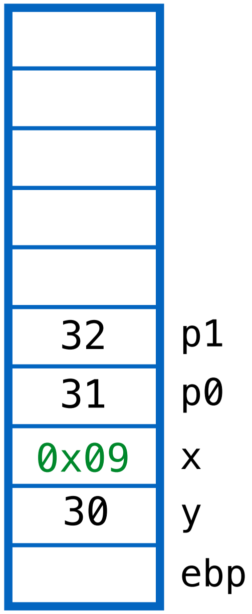
esi

QUIZ: Which cells are garbage?

(A) 0x00, 0x04 (B) 0x04, 0x08 (C) 0x08, 0x0c (D) None (E) All

ex2: garbage in the middle

```
let y  = let tmp = (10, 20)
         in tmp[0] + tmp[1]
  , x  = (1, 2)
  , p0 = x[0] + y
  , p1 = x[1] + y
in
  (p0, p1)
```



32 p1

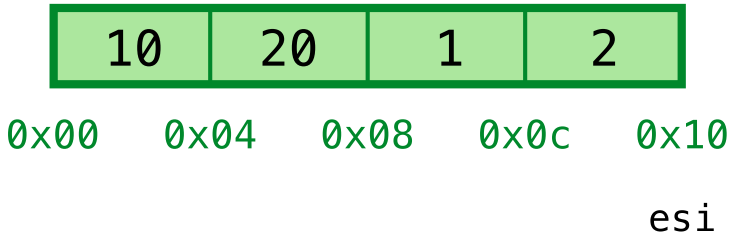
31 p0

0x09 x

30 y

ebp

Lets reclaim & recycle garbage!



10	20	1	2

0x00 0x04 0x08 0x0c 0x10

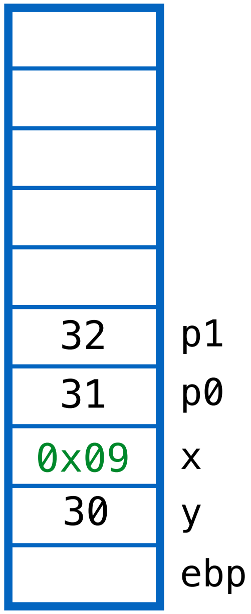
esi

QUIZ: Which cells are garbage?

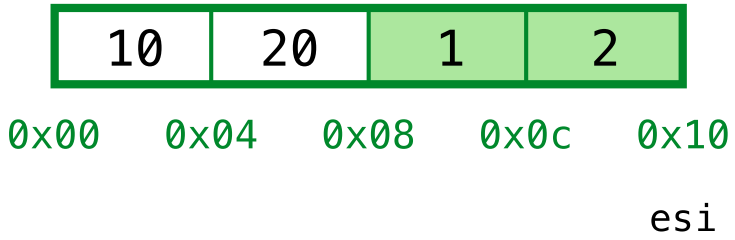
Those that are *not reachable from stack*

ex2: garbage in the middle

```
let y  = let tmp = (10, 20)
         in tmp[0] + tmp[1]
  , x  = (1, 2)
  , p0 = x[0] + y
  , p1 = x[1] + y
in
  (p0, p1)
```



Lets reclaim & recycle garbage!

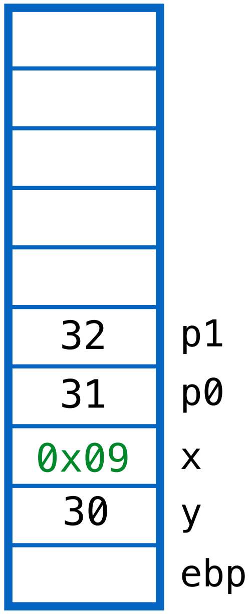


QUIZ: Which cells are garbage?

Those that are *not reachable from stack*

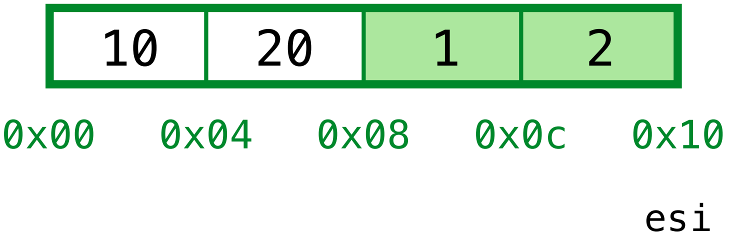
ex2: garbage in the middle

```
let y  = let tmp = (10, 20)
         in tmp[0] + tmp[1]
  , x  = (1, 2)
  , p0 = x[0] + y
  , p1 = x[1] + y
in
  (p0, p1)
```



32	p1
31	p0
0x09	x
30	y
	ebp

Lets reclaim & recycle garbage!

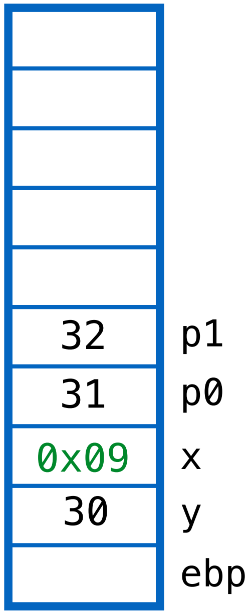


Q: How to reclaim space?

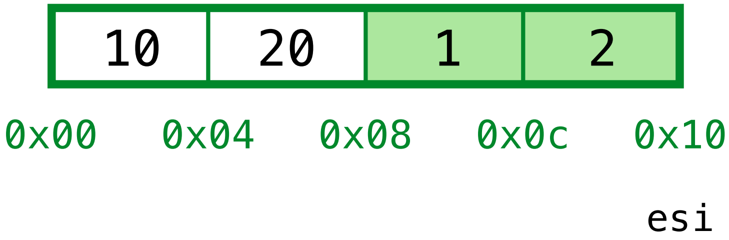
Why is it not enough to rewind `esi`?

ex2: garbage in the middle

```
let y  = let tmp = (10, 20)
         in tmp[0] + tmp[1]
  , x  = (1, 2)
  , p0 = x[0] + y
  , p1 = x[1] + y
in
  (p0, p1)
```



Lets reclaim & recycle garbage!

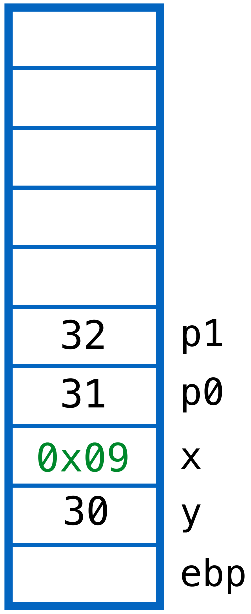


Why is it not enough to rewind `esi`?

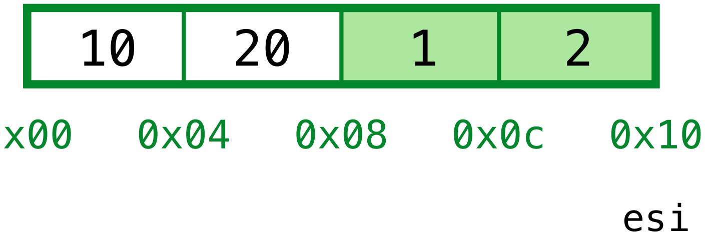
Want free space to be *contiguous* (i.e. go to end of heap)

ex2: garbage in the middle

```
let y  = let tmp = (10, 20)
         in tmp[0] + tmp[1]
  , x  = (1, 2)
  , p0 = x[0] + y
  , p1 = x[1] + y
in
  (p0, p1)
```



Lets reclaim & recycle garbage!

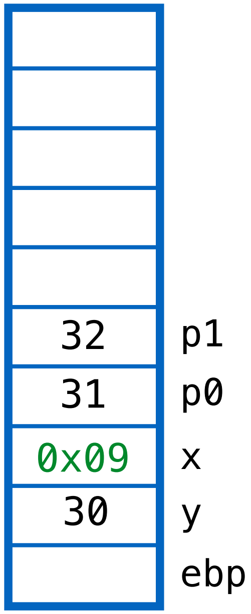


Solution: Compaction

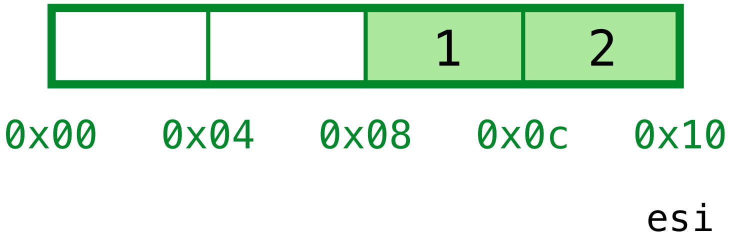
Copy “live” cells into “garbage” …

ex2: garbage in the middle

```
let y  = let tmp = (10, 20)
         in tmp[0] + tmp[1]
  , x  = (1, 2)
  , p0 = x[0] + y
  , p1 = x[1] + y
in
  (p0, p1)
```



Lets reclaim & recycle garbage!

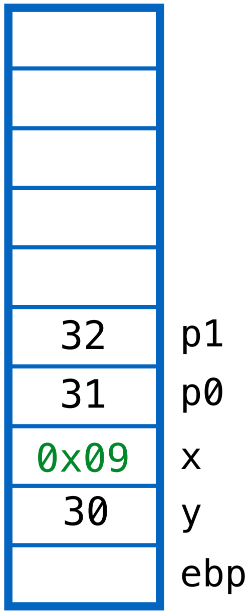


Solution: Compaction

Copy “live” cells into “garbage” …

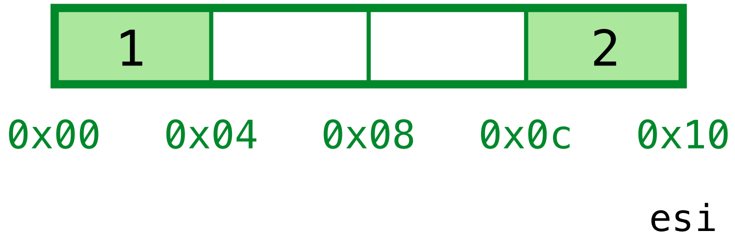
ex2: garbage in the middle

```
let y  = let tmp = (10, 20)
         in tmp[0] + tmp[1]
  , x  = (1, 2)
  , p0 = x[0] + y
  , p1 = x[1] + y
in
  (p0, p1)
```



32	p1
31	p0
0x09	x
30	y
	ebp

Lets reclaim & recycle garbage!

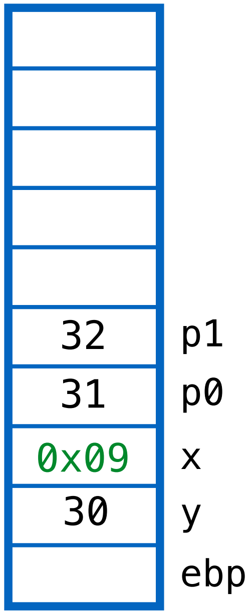


Solution: Compaction

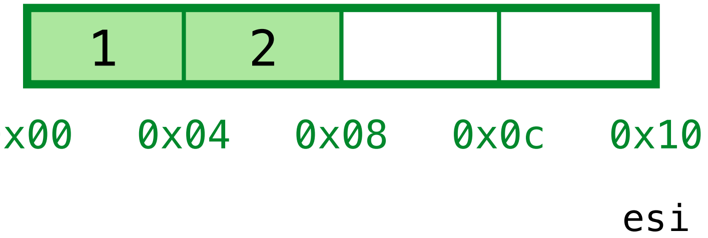
Copy “live” cells into “garbage” ...

ex2: garbage in the middle

```
let y  = let tmp = (10, 20)
         in tmp[0] + tmp[1]
  , x  = (1, 2)
  , p0 = x[0] + y
  , p1 = x[1] + y
in
  (p0, p1)
```



Lets reclaim & recycle garbage!

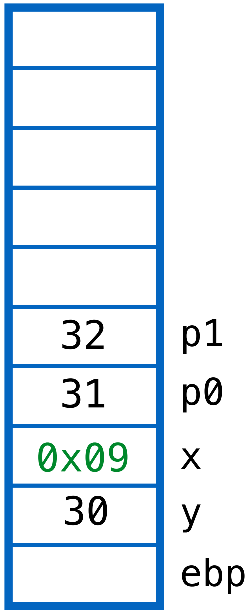


Solution: Compaction

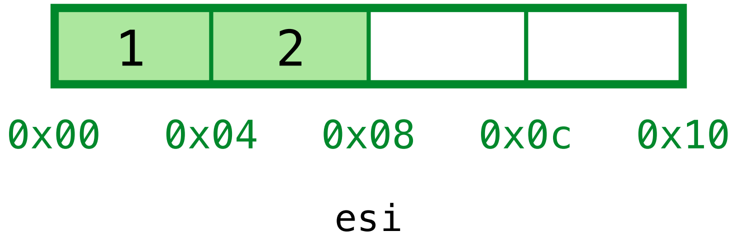
Copy “live” cells into “garbage” …

ex2: garbage in the middle

```
let y  = let tmp = (10, 20)
         in tmp[0] + tmp[1]
  , x  = (1, 2)
  , p0 = x[0] + y
  , p1 = x[1] + y
in
  (p0, p1)
```



Lets reclaim & recycle garbage!



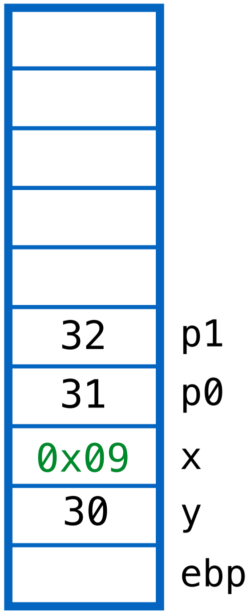
Solution: Compaction

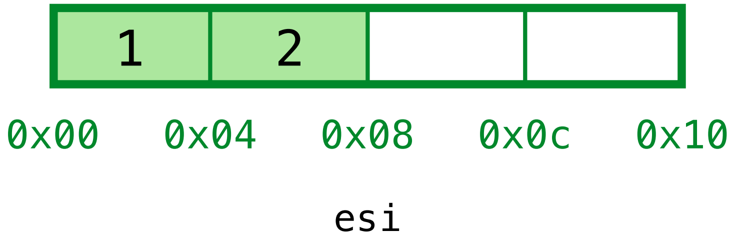
Copy “live” cells into “garbage” … then rewind `esi`!

ex2: garbage in the middle

```
let y  = let tmp = (10, 20)
         in tmp[0] + tmp[1]
  , x  = (1, 2)
  , p0 = x[0] + y
  , p1 = x[1] + y
in
  (p0, p1)
```

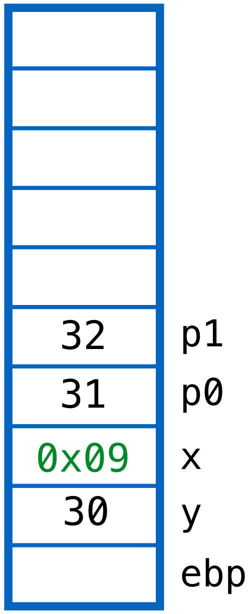
Yay! Have space for `(p0, p1)`





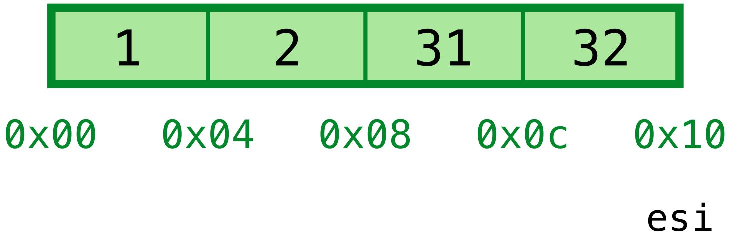
ex2: garbage in the middle

```
let y  = let tmp = (10, 20)
         in tmp[0] + tmp[1]
  , x  = (1, 2)
  , p0 = x[0] + y
  , p1 = x[1] + y
in
  (p0, p1)
```



32	p1
31	p0
0x09	x
30	y
	ebp

Yay! Have space for `(p0, p1)`



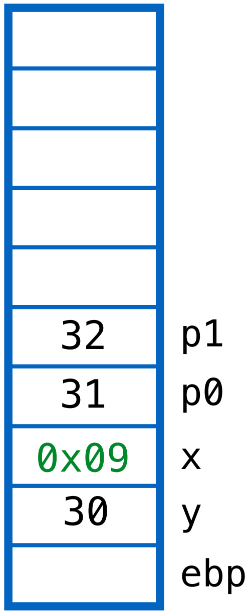
1	2	31	32

0x00 0x04 0x08 0x0c 0x10

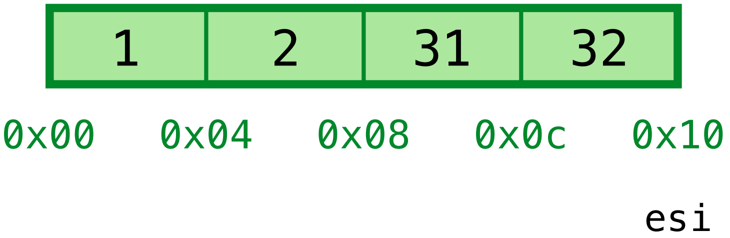
esi

ex2: garbage in the middle

```
let y  = let tmp = (10, 20)
         in tmp[0] + tmp[1]
  , x  = (1, 2)
  , p0 = x[0] + y
  , p1 = x[1] + y
in
  (p0, p1)
```



Result (eax) = 0x09

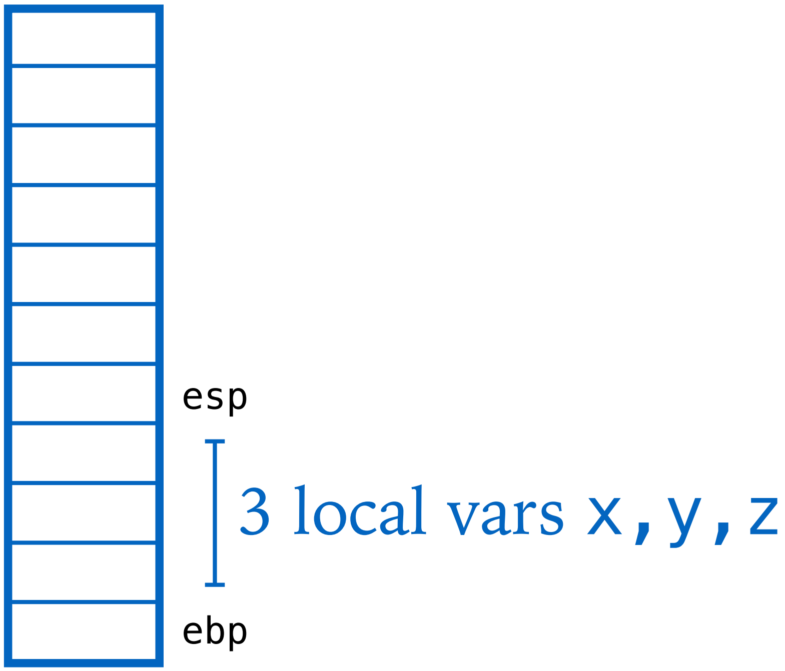


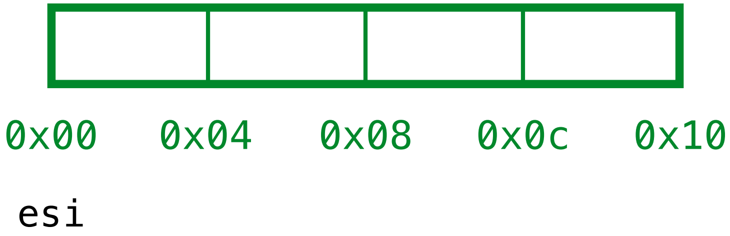
FOX / GC

Example 3

ex3: garbage in the middle (with stack)

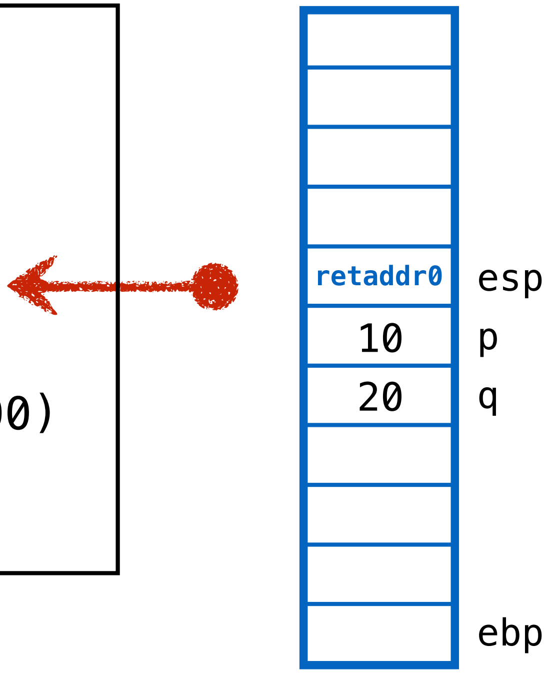
```
def foo(p, q):
  let tmp = (p, q)
  in tmp[0] + tmp[1]

let y  = foo(10, 20)
  , x  = (y, y + 1)
  , z  = foo(100, 200)
in
  x[0] + y + z
```

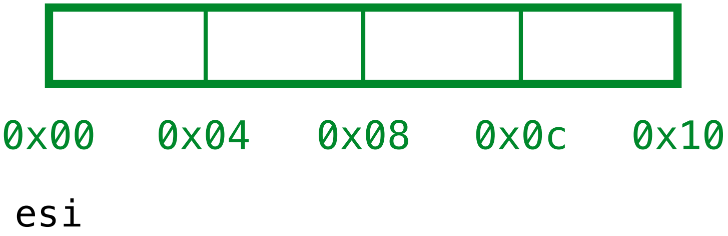




ex3: garbage in the middle (with stack)

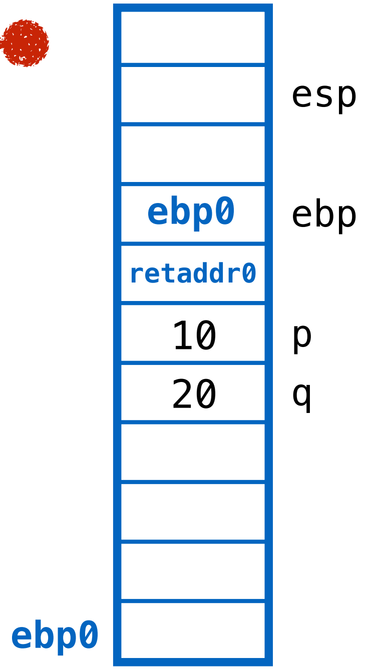
```
def foo(p, q):
  let tmp = (p, q)
  in tmp[0] + tmp[1]

let y  = foo(10, 20)
  , x  = (y, y + 1)
  , z  = foo(100, 200)
in
  x[0] + y + z
```

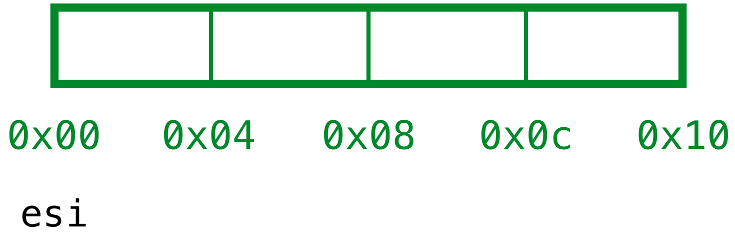




ex3: garbage in the middle (with stack)

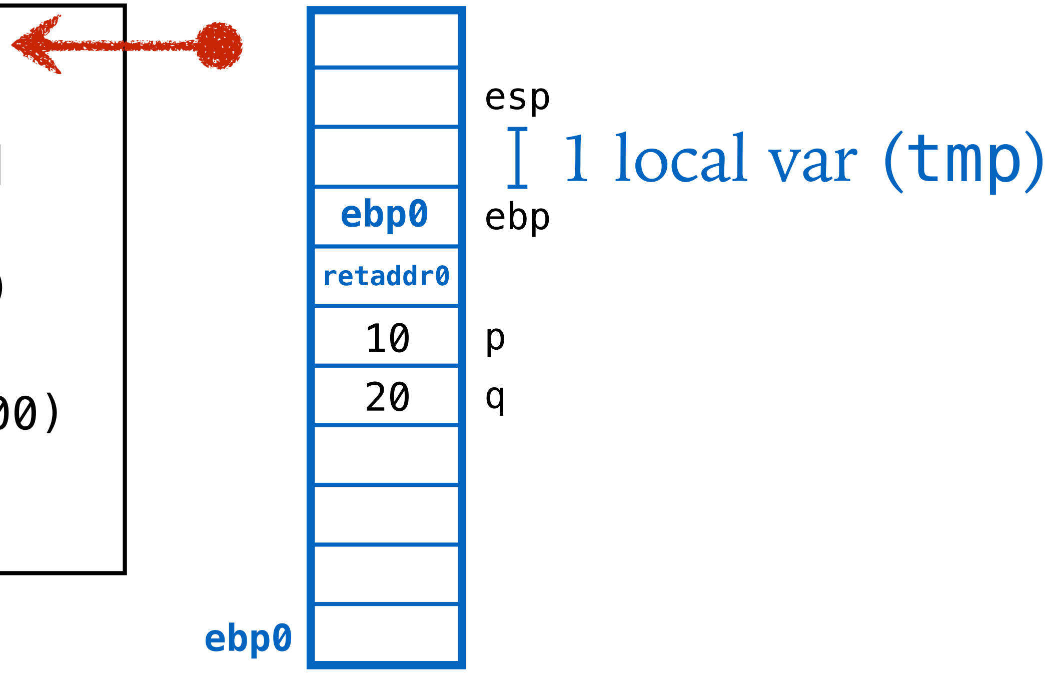
```
def foo(p, q):
  let tmp = (p, q)
  in tmp[0] + tmp[1]

let y  = foo(10, 20)
  , x  = (y, y + 1)
  , z  = foo(100, 200)
in
  x[0] + y + z
```

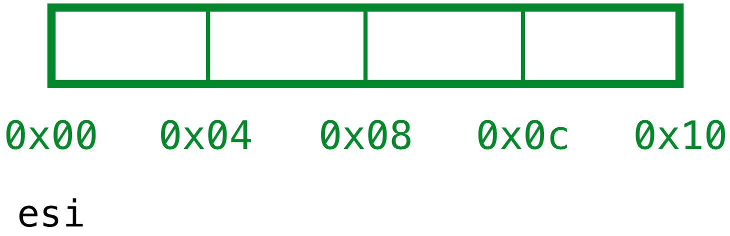




ex3: garbage in the middle (with stack)

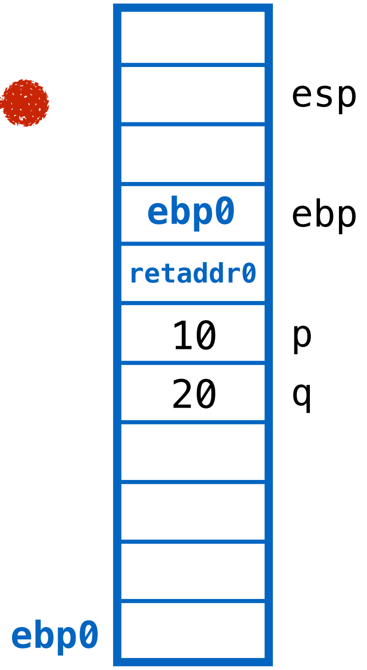
```
def foo(p, q):
  let tmp = (p, q)
  in tmp[0] + tmp[1]

let y  = foo(10, 20)
  , x  = (y, y + 1)
  , z  = foo(100, 200)
in
  x[0] + y + z
```

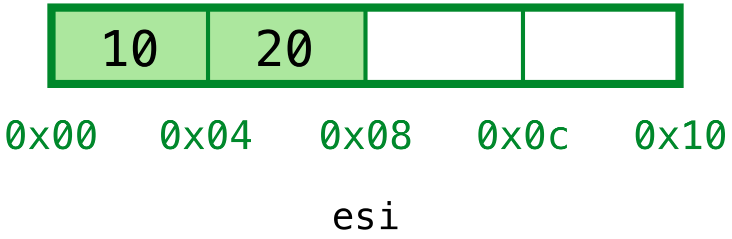




ex3: garbage in the middle (with stack)

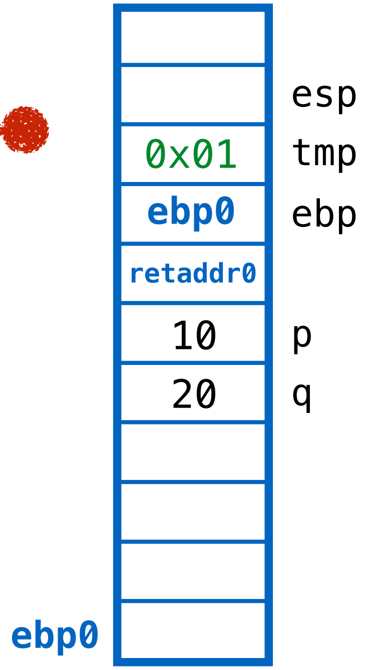
```
def foo(p, q):
  let tmp = (p, q)
  in tmp[0] + tmp[1]

let y  = foo(10, 20)
  , x  = (y, y + 1)
  , z  = foo(100, 200)
in
  x[0] + y + z
```

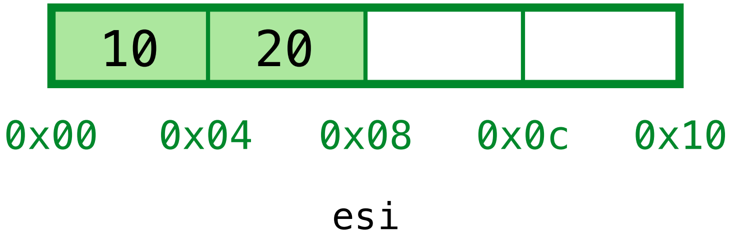




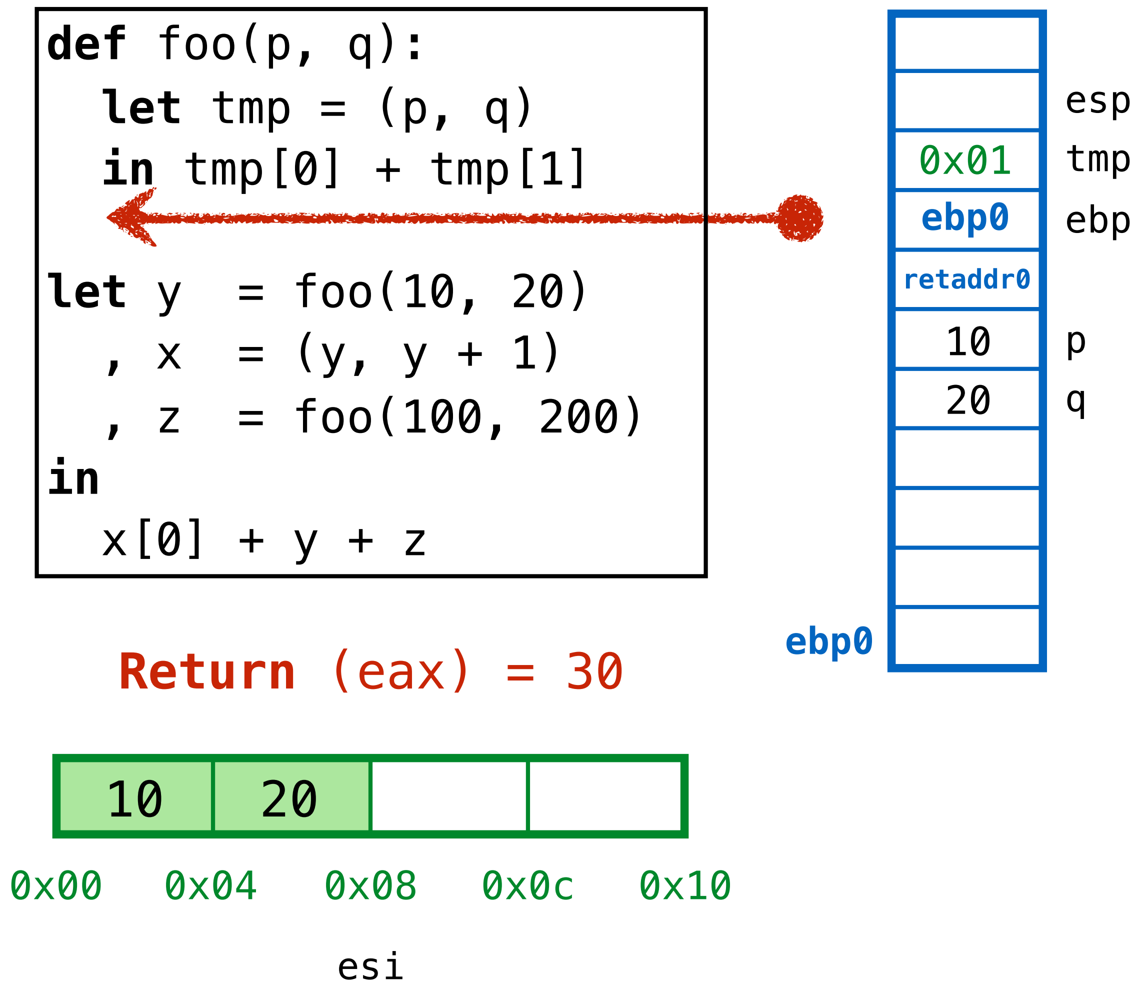
ex3: garbage in the middle (with stack)

```
def foo(p, q):
   let tmp = (p, q)
   in tmp[0] + tmp[1]

let y  = foo(10, 20)
  , x  = (y, y + 1)
  , z  = foo(100, 200)
in
  x[0] + y + z
```





ex3: garbage in the middle (with stack)

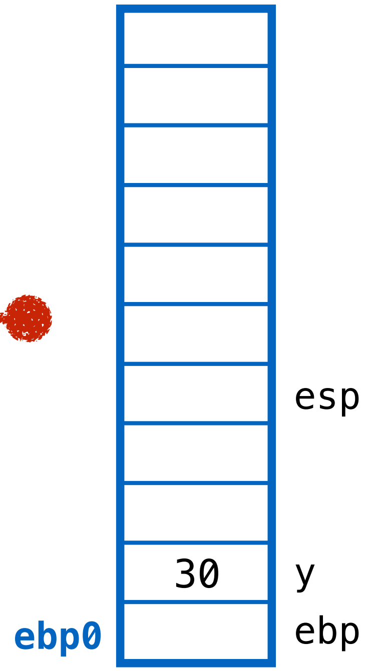


```
def foo(p, q):
  let tmp = (p, q)
  in tmp[0] + tmp[1]

let y  = foo(10, 20)
  , x  = (y, y + 1)
  , z  = foo(100, 200)
in
  x[0] + y + z
```

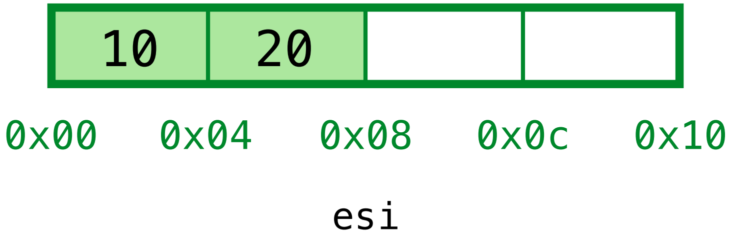
Return (eax) = 30

ex3: garbage in the middle (with stack)

```
def foo(p, q):
  let tmp = (p, q)
  in tmp[0] + tmp[1]

let y  = foo(10, 20)
  , x  = (y, y + 1)
  , z  = foo(100, 200)
in
  x[0] + z
```

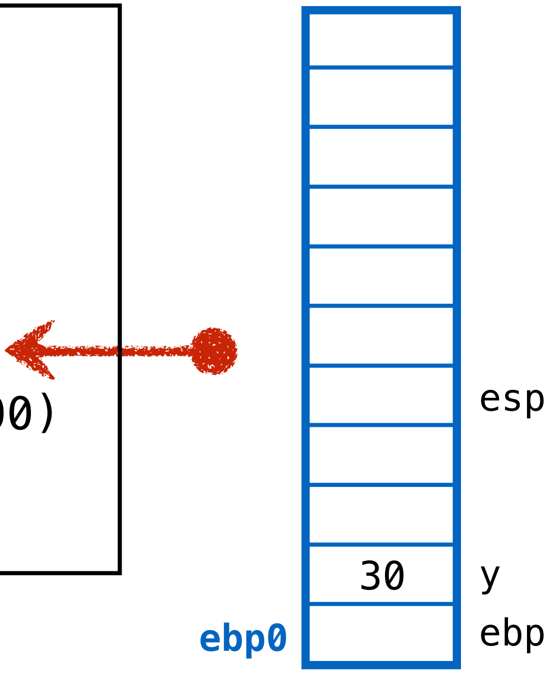


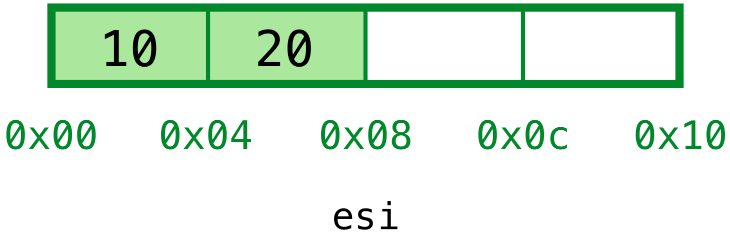
Return (eax) = 30



ex3: garbage in the middle (with stack)

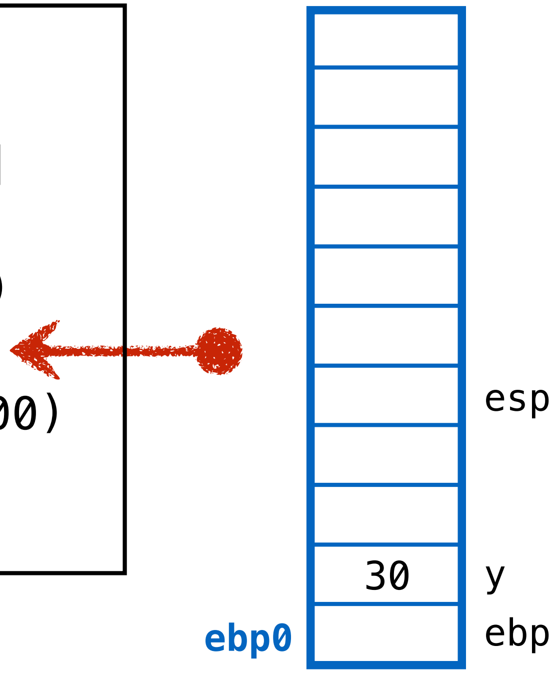
```
def foo(p, q):
  let tmp = (p, q)
  in tmp[0] + tmp[1]

let y  = foo(10, 20)
  , x  = (y, y + 1)
  , z  = foo(100, 200)
in
  x[0] + z
```

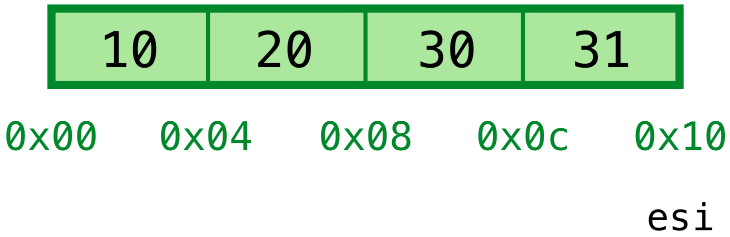




ex3: garbage in the middle (with stack)

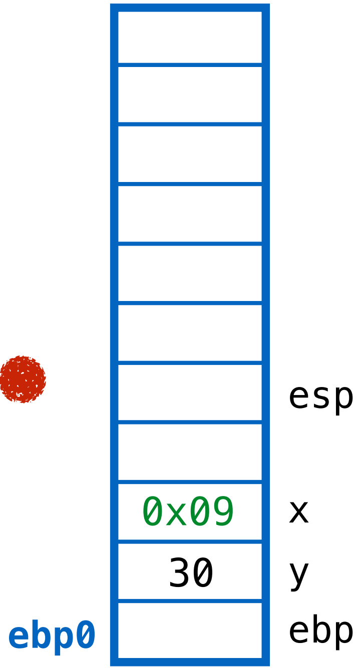
```
def foo(p, q):
  let tmp = (p, q)
  in tmp[0] + tmp[1]

let y  = foo(10, 20)
  , x  = (y, y + 1)
  , z  = foo(100, 200)
in
  x[0] + z
```

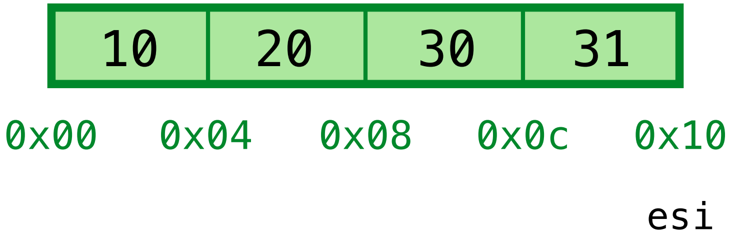




ex3: garbage in the middle (with stack)

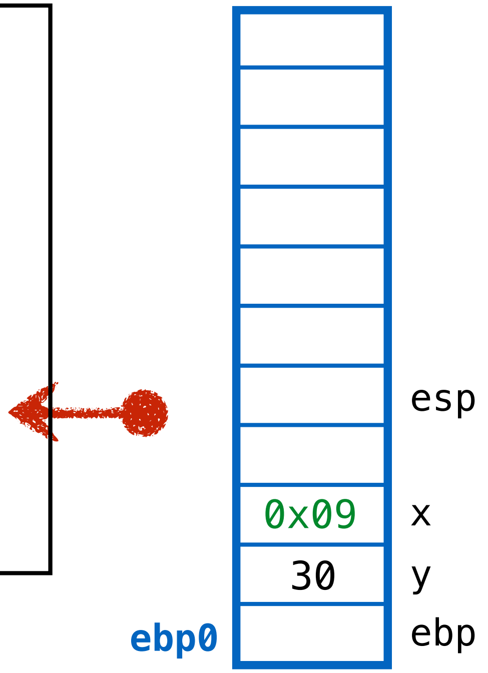
```
def foo(p, q):
  let tmp = (p, q)
  in tmp[0] + tmp[1]

let y  = foo(10, 20)
  , x  = (y, y + 1)
  , z  = foo(100, 200)
in
  x[0] + z
```

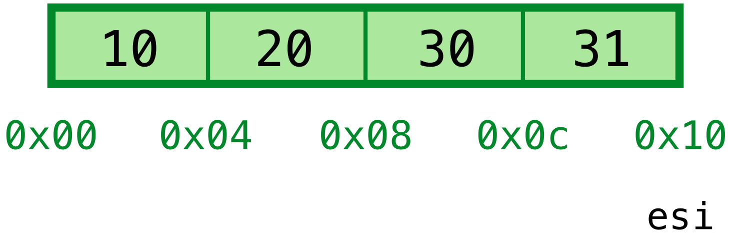




ex3: garbage in the middle (with stack)

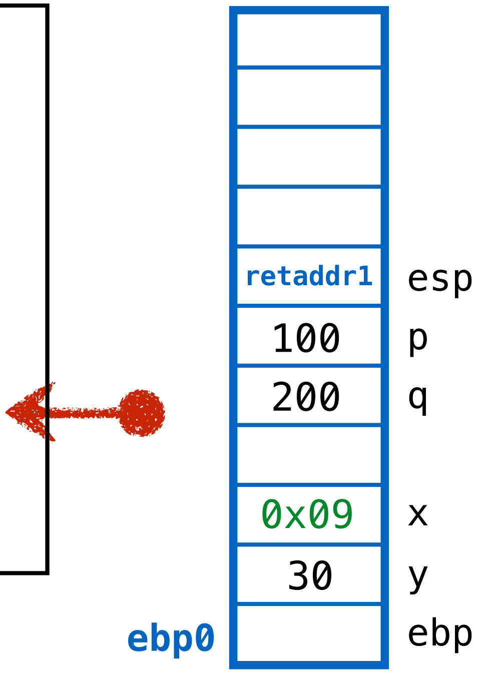
```
def foo(p, q):
  let tmp = (p, q)
  in tmp[0] + tmp[1]

let y  = foo(10, 20)
  , x  = (y, y + 1)
  , z  = foo(100, 200)
in
  x[0] + z
```



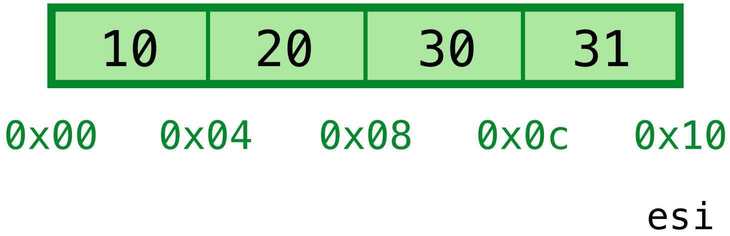


ex3: garbage in the middle (with stack)

```
def foo(p, q):
  let tmp = (p, q)
  in tmp[0] + tmp[1]

let y  = foo(10, 20)
  , x  = (y, y + 1)
  , z  = foo(100, 200)
in
  x[0] + z
```



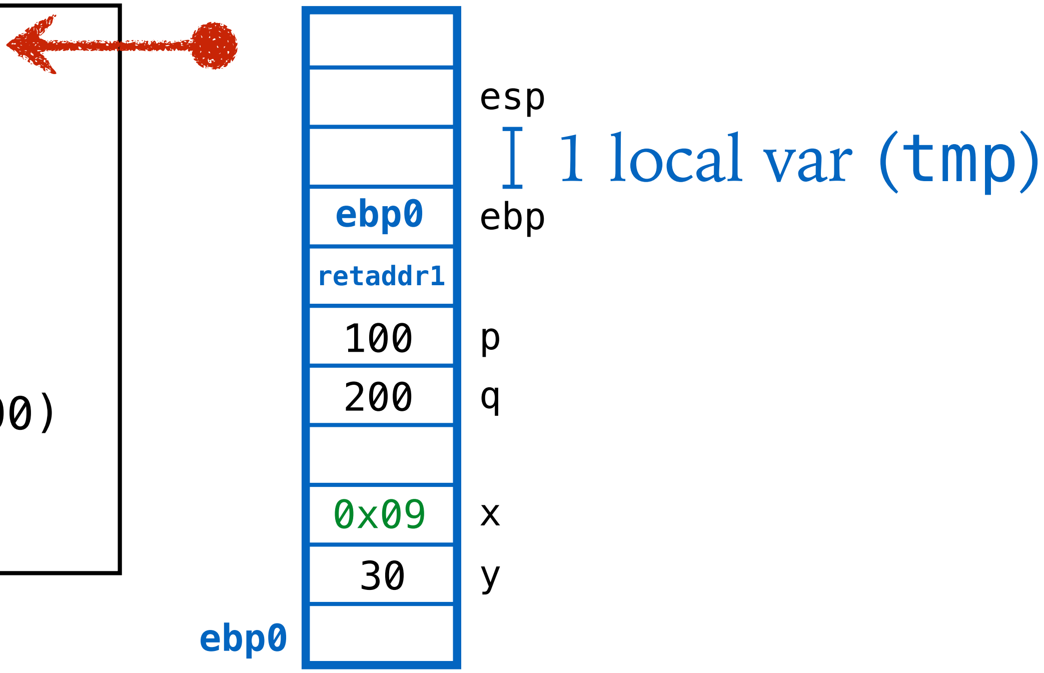
	retaddr1	esp
	100	p
	200	q
	0x09	x
	30	y
ebp0		ebp

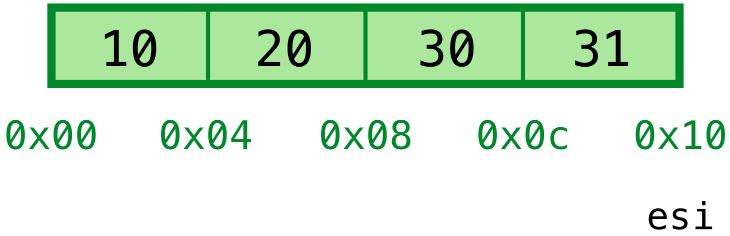


10	20	30	31

ex3: garbage in the middle (with stack)

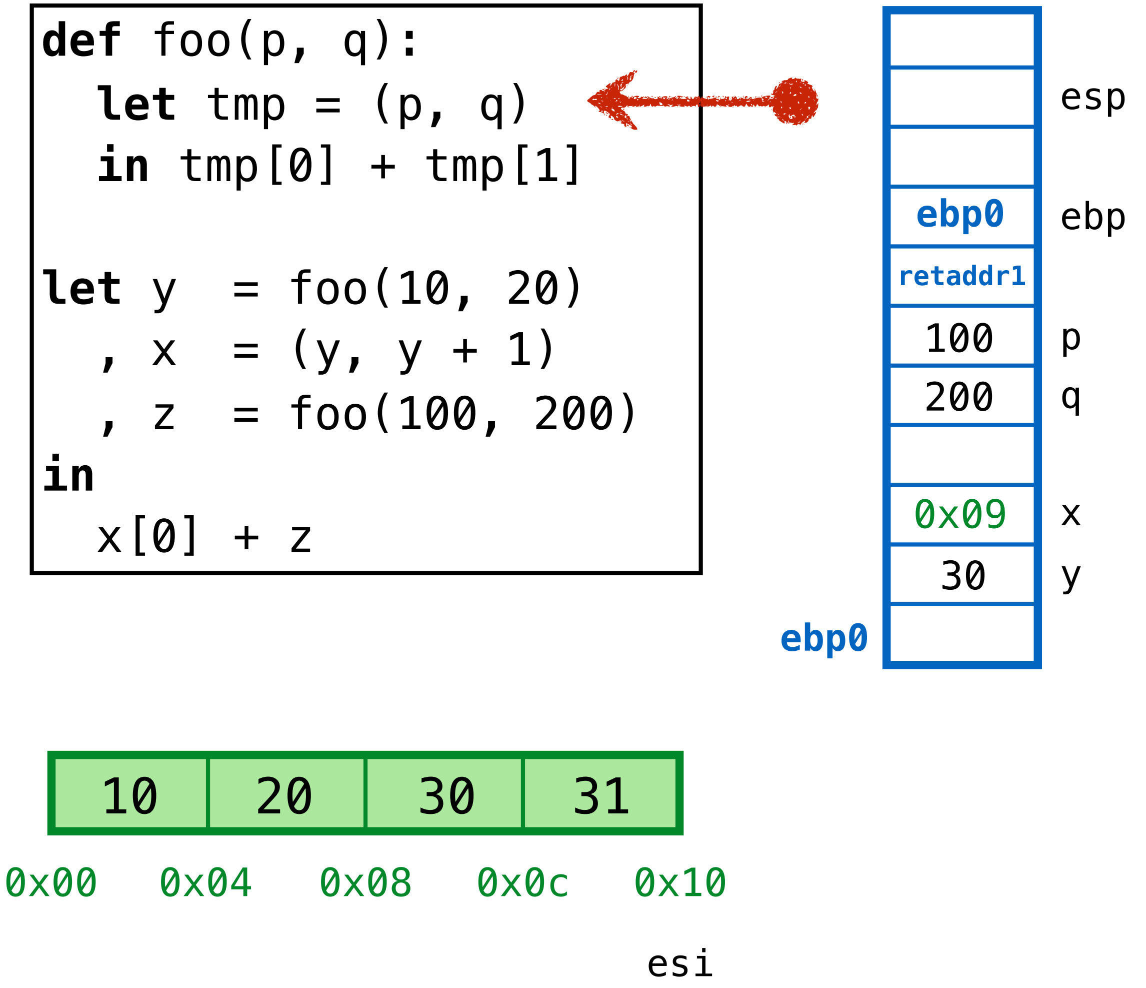
```
def foo(p, q):
  let tmp = (p, q)
  in tmp[0] + tmp[1]

let y  = foo(10, 20)
  , x  = (y, y + 1)
  , z  = foo(100, 200)
in
  x[0] + z
```





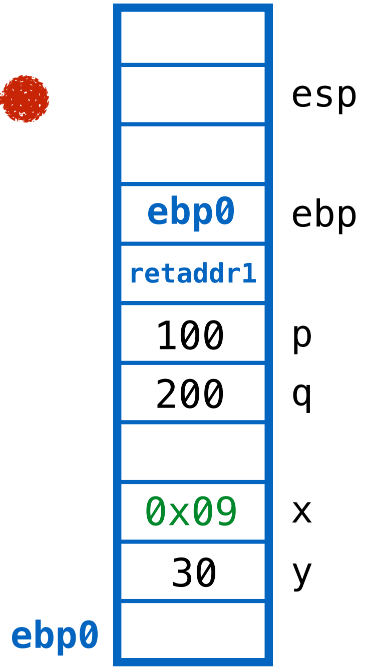
ex3: garbage in the middle (with stack)

```
def foo(p, q):
  let tmp = (p, q)
  in tmp[0] + tmp[1]

let y  = foo(10, 20)
  , x  = (y, y + 1)
  , z  = foo(100, 200)
in
  x[0] + z
```

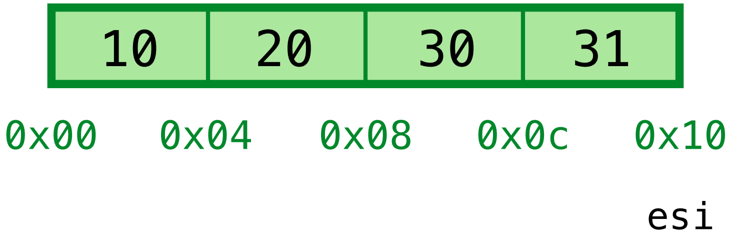


ex3: garbage in the middle (with stack)

```
def foo(p, q):
  let tmp = (p, q)
  in tmp[0] + tmp[1]

let y  = foo(10, 20)
  , x  = (y, y + 1)
  , z  = foo(100, 200)
in
  x[0] + z
```

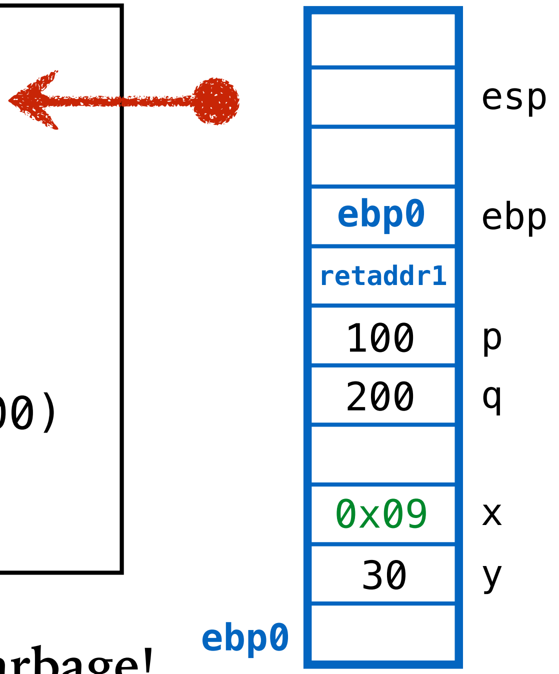


Lets reclaim & recycle garbage!



ex3: garbage in the middle (with stack)

```
def foo(p, q):
  let tmp = (p, q)
  in tmp[0] + tmp[1]

let y  = foo(10, 20)
  , x  = (y, y + 1)
  , z  = foo(100, 200)
in
  x[0] + z
```



Lets reclaim & recycle garbage!

10	20	30	31

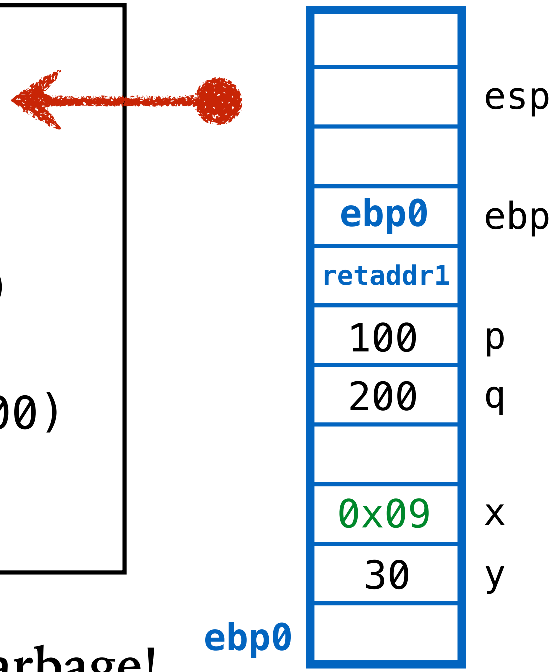
0x00 0x04 0x08 0x0c 0x10

QUIZ: Which cells are garbage?

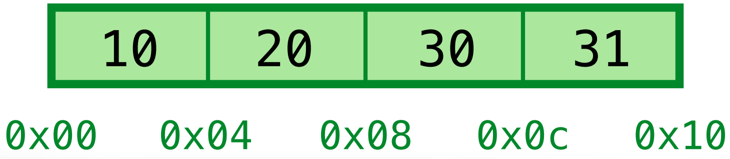
(A) 0x00, 0x04 (B) 0x04, 0x08 (C) 0x08, 0x0c (D) None (E) All

ex3: garbage in the middle (with stack)

```
def foo(p, q):
  let tmp = (p, q)
  in tmp[0] + tmp[1]

let y  = foo(10, 20)
  , x  = (y, y + 1)
  , z  = foo(100, 200)
in
  x[0] + z
```



Lets reclaim & recycle garbage!

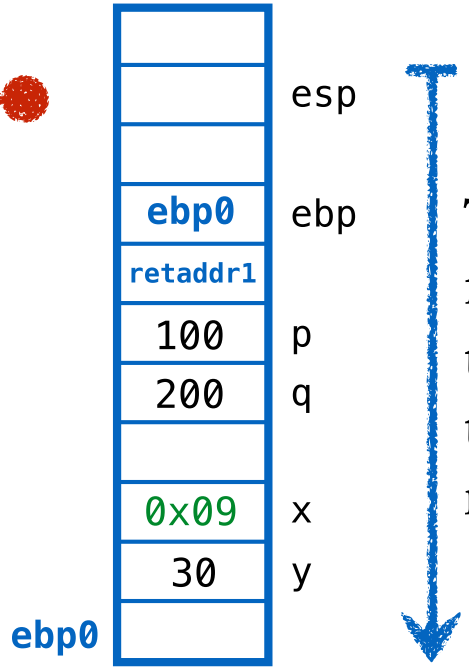


QUIZ: Which cells are garbage?

Those that are *not reachable from any stack frame*

ex3: garbage in the middle (with stack)

```
def foo(p, q):
  let tmp = (p, q)
  in tmp[0] + tmp[1]

let y  = foo(10, 20)
  , x  = (y, y + 1)
  , z  = foo(100, 200)
in
  x[0] + z
```



Traverse Stack from top (`esp`) to bottom (`ebp0`) to mark reachable cells.

Lets reclaim & recycle garbage!

10	20	30	31

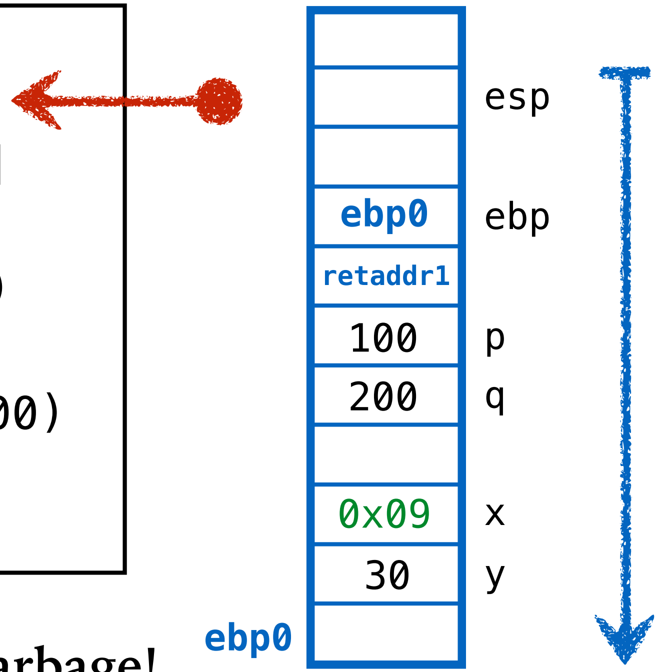
`0x00` `0x04` `0x08` `0x0c` `0x10`

QUIZ: Which cells are garbage?

Those that are *not reachable from any stack frame*

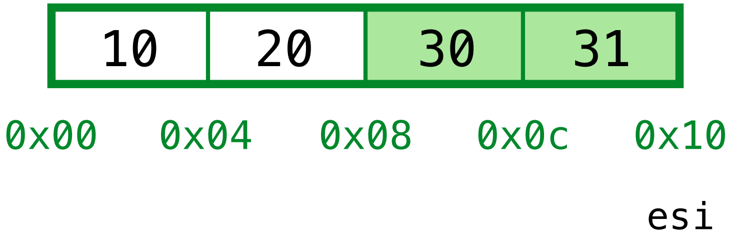
ex3: garbage in the middle (with stack)

```
def foo(p, q):
  let tmp = (p, q)
  in tmp[0] + tmp[1]

let y  = foo(10, 20)
  , x  = (y, y + 1)
  , z  = foo(100, 200)
in
  x[0] + z
```



Traverse Stack from top (`esp`) to bottom (`ebp0`) to mark reachable cells.

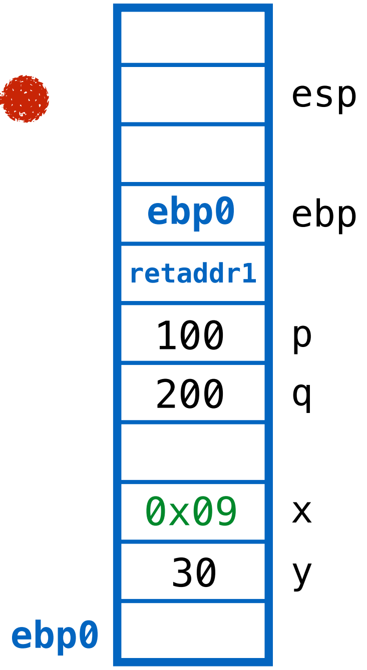
Lets reclaim & recycle garbage!

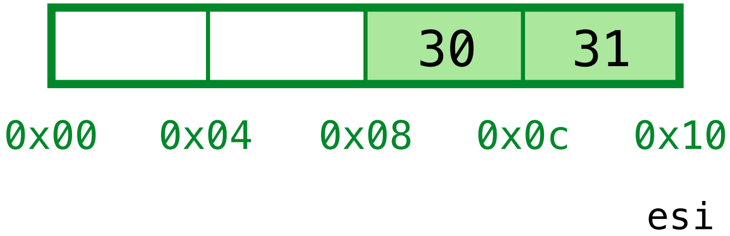


Which cells are garbage?

ex3: garbage in the middle (with stack)

```
def foo(p, q):
  let tmp = (p, q)
  in tmp[0] + tmp[1]

let y  = foo(10, 20)
  , x  = (y, y + 1)
  , z  = foo(100, 200)
in
  x[0] + z
```

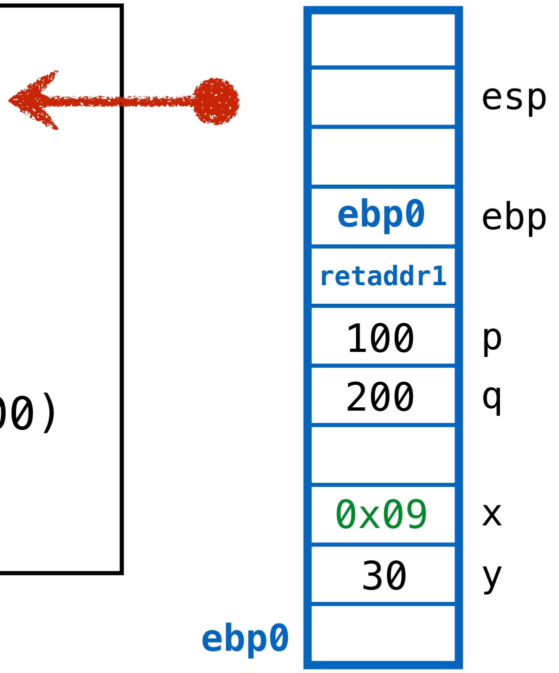


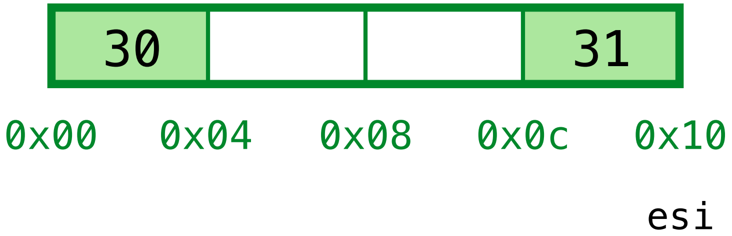


Compact the live cells

ex3: garbage in the middle (with stack)

```
def foo(p, q):
  let tmp = (p, q)
  in tmp[0] + tmp[1]

let y  = foo(10, 20)
  , x  = (y, y + 1)
  , z  = foo(100, 200)
in
  x[0] + z
```

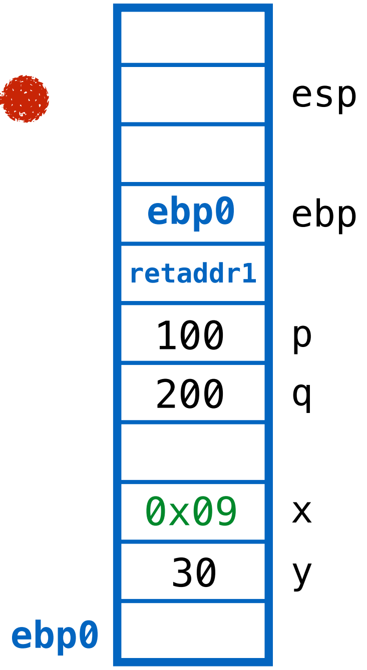


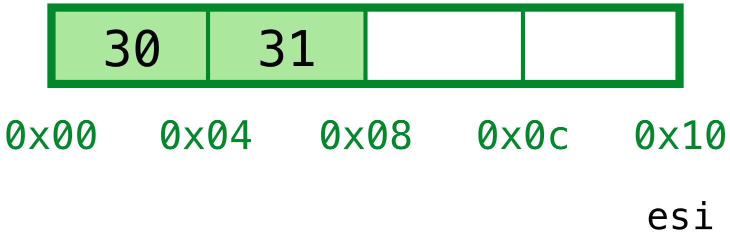


Compact the live cells

ex3: garbage in the middle (with stack)

```
def foo(p, q):
  let tmp = (p, q)
  in tmp[0] + tmp[1]

let y  = foo(10, 20)
  , x  = (y, y + 1)
  , z  = foo(100, 200)
in
  x[0] + z
```

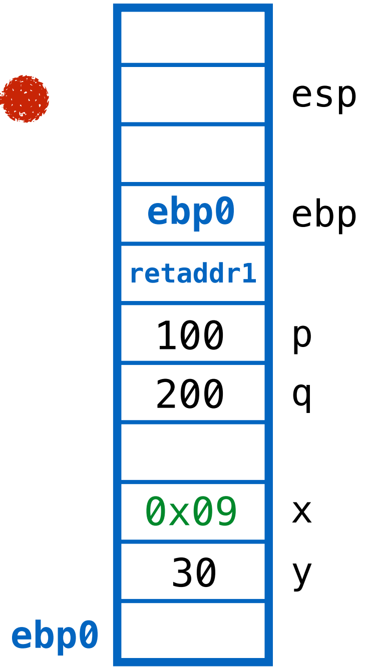


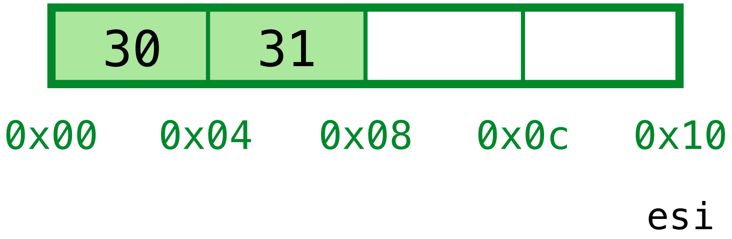


Compact the live cells

ex3: garbage in the middle (with stack)

```
def foo(p, q):
  let tmp = (p, q)
  in tmp[0] + tmp[1]

let y  = foo(10, 20)
  , x  = (y, y + 1)
  , z  = foo(100, 200)
in
  x[0] + z
```

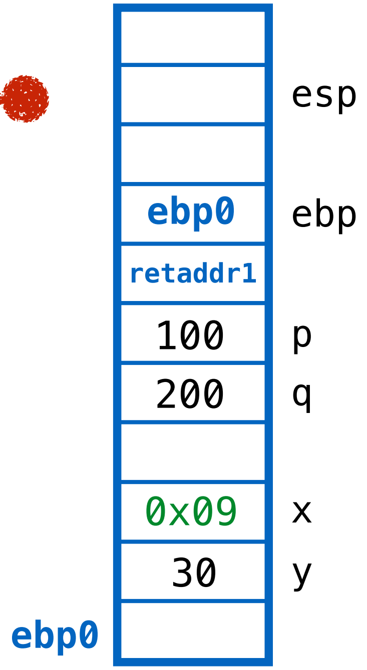


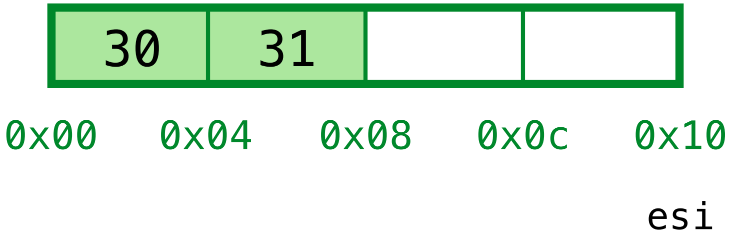


Compact the live cells

ex3: garbage in the middle (with stack)

```
def foo(p, q):
  let tmp = (p, q)
  in tmp[0] + tmp[1]

let y  = foo(10, 20)
  , x  = (y, y + 1)
  , z  = foo(100, 200)
in
  x[0] + z
```

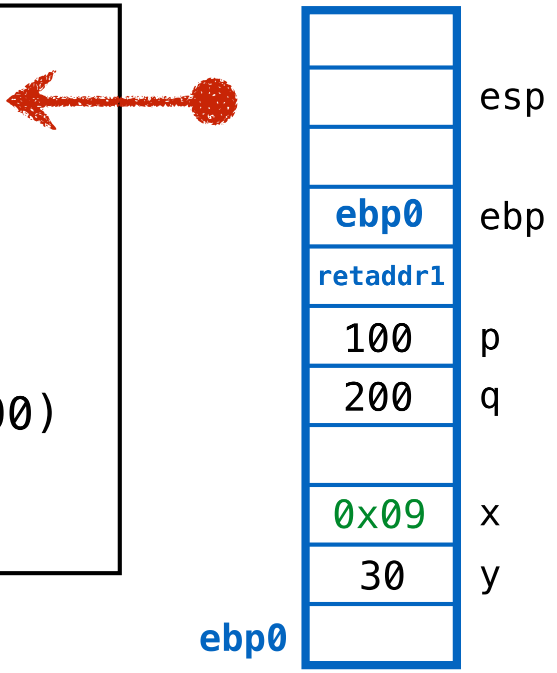


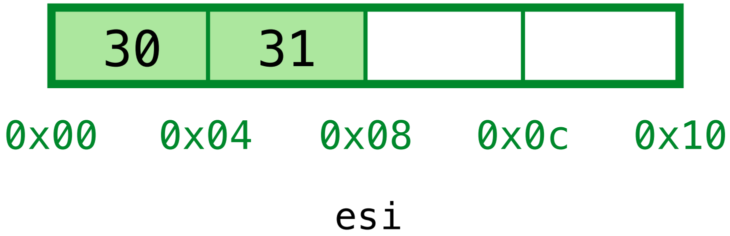


Compact the live cells … then rewind `esi`

ex3: garbage in the middle (with stack)

```
def foo(p, q):
  let tmp = (p, q)
  in tmp[0] + tmp[1]

let y  = foo(10, 20)
  , x  = (y, y + 1)
  , z  = foo(100, 200)
in
  x[0] + z
```

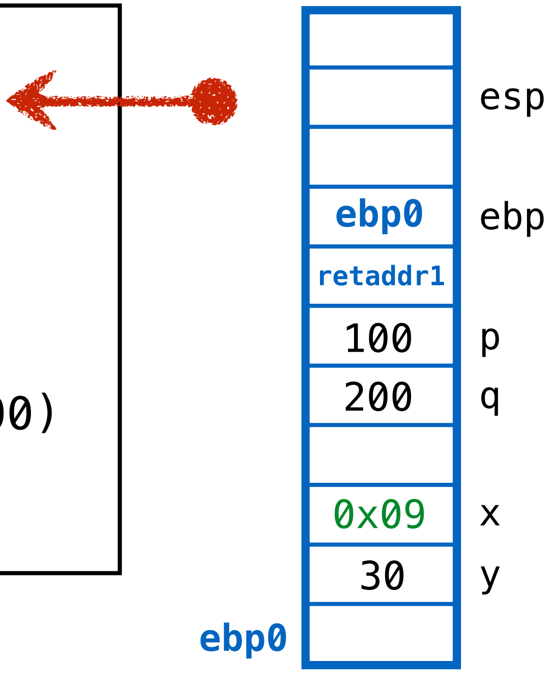


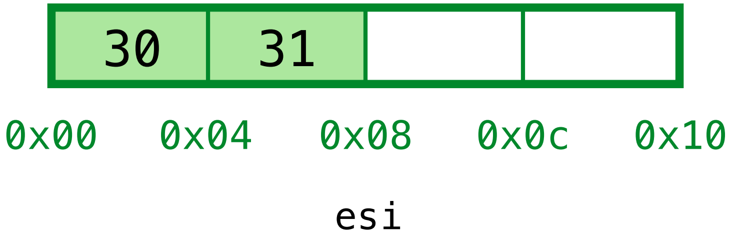


Compact the live cells … then rewind `esi`

ex3: garbage in the middle (with stack)

```
def foo(p, q):
  let tmp = (p, q)
  in tmp[0] + tmp[1]

let y  = foo(10, 20)
  , x  = (y, y + 1)
  , z  = foo(100, 200)
in
  x[0] + z
```

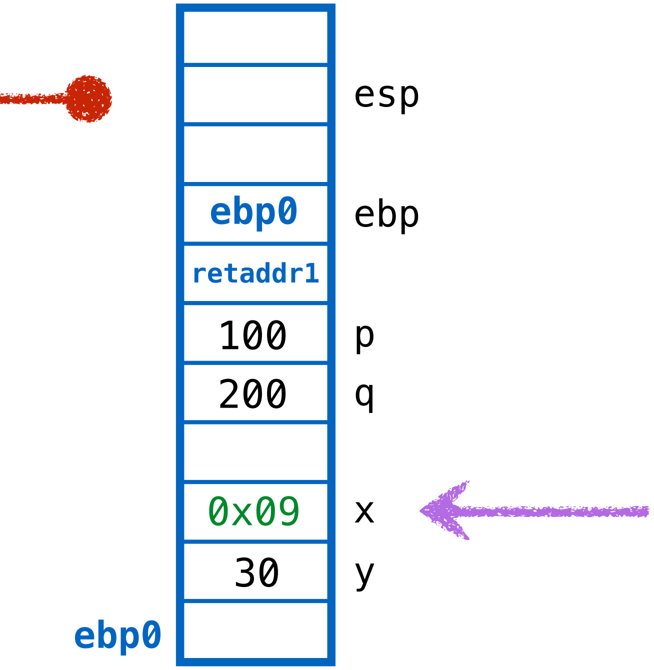


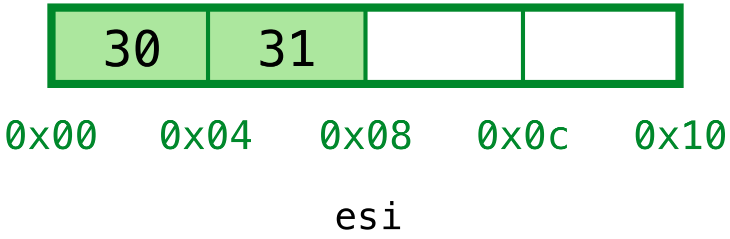


Problem???

ex3: garbage in the middle (with stack)

```
def foo(p, q):
  let tmp = (p, q)
  in tmp[0] + tmp[1]

let y  = foo(10, 20)
  , x  = (y, y + 1)
  , z  = foo(100, 200)
in
  x[0] + z
```

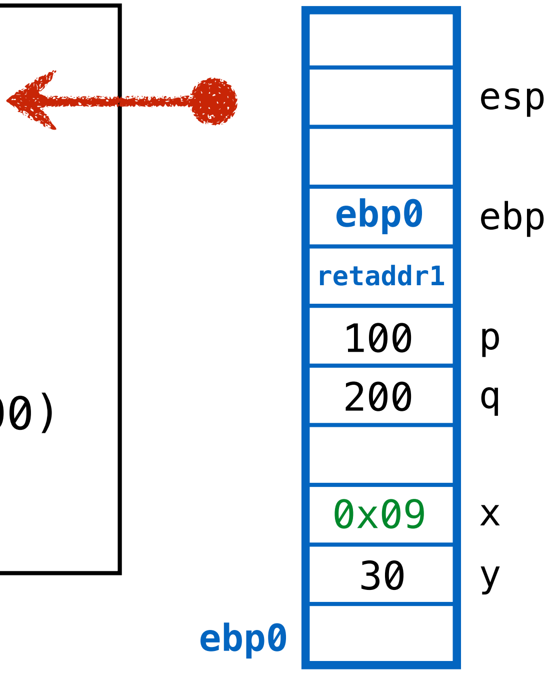


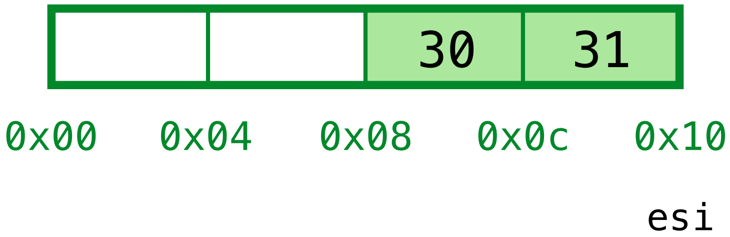


Problem! Have to REDIRECT existing pointers

ex3: garbage in the middle (with stack)

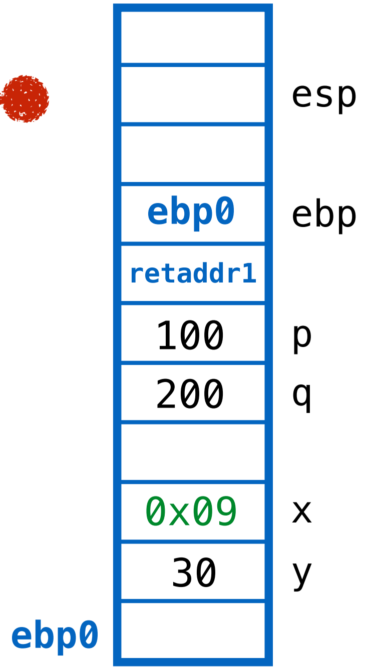
```
def foo(p, q):
  let tmp = (p, q)
  in tmp[0] + tmp[1]

let y  = foo(10, 20)
  , x  = (y, y + 1)
  , z  = foo(100, 200)
in
  x[0] + z
```

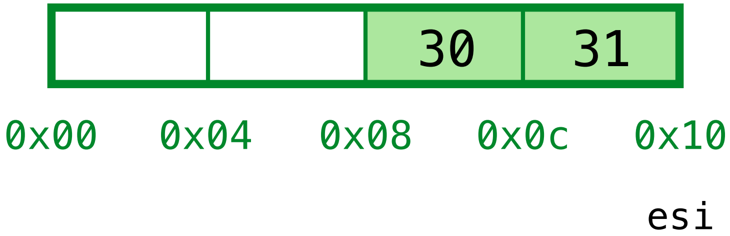




ex3: garbage in the middle (with stack)

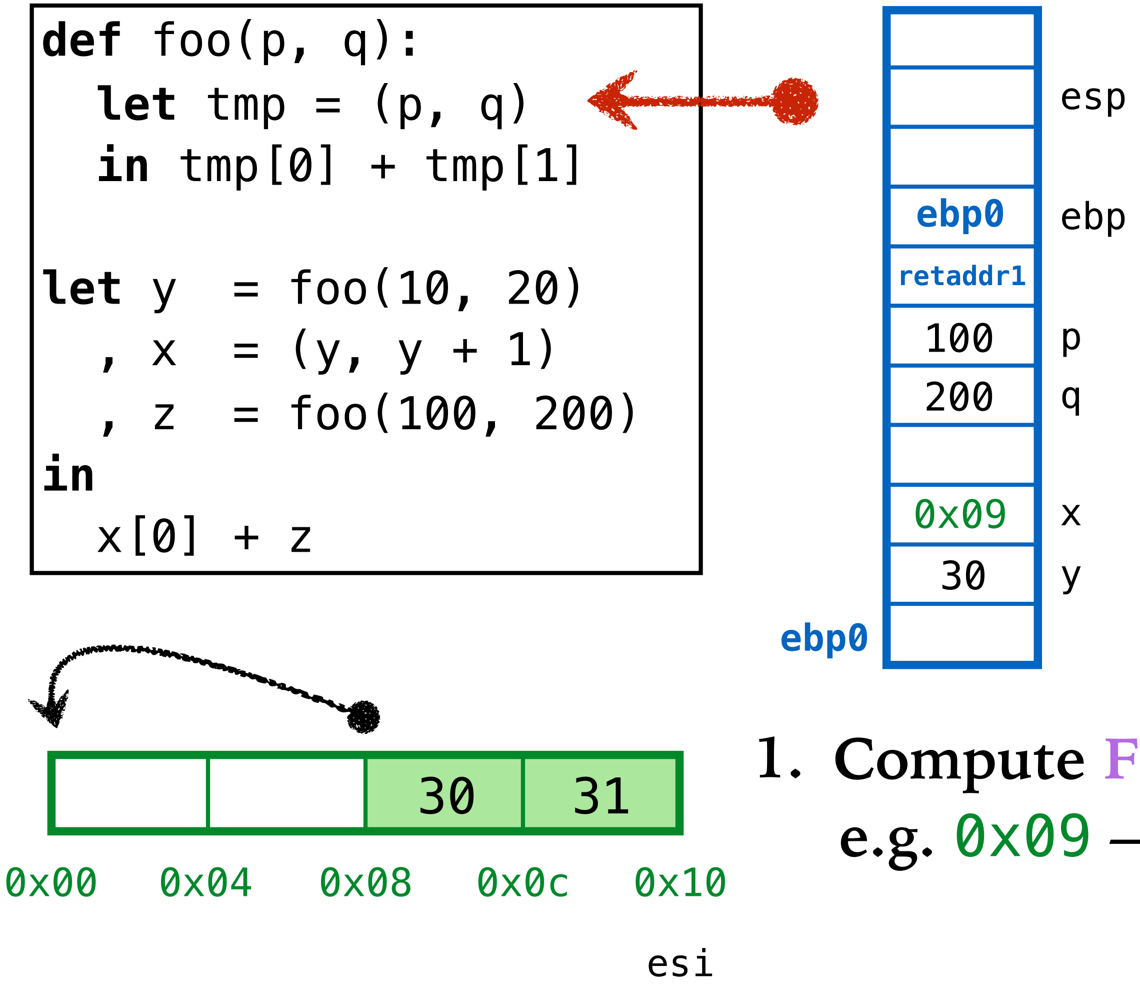
```
def foo(p, q):
  let tmp = (p, q)
  in tmp[0] + tmp[1]

let y  = foo(10, 20)
  , x  = (y, y + 1)
  , z  = foo(100, 200)
in
  x[0] + z
```





1. Compute FORWARD addrs (i.e. new compacted addrs)

ex3: garbage in the middle (with stack)

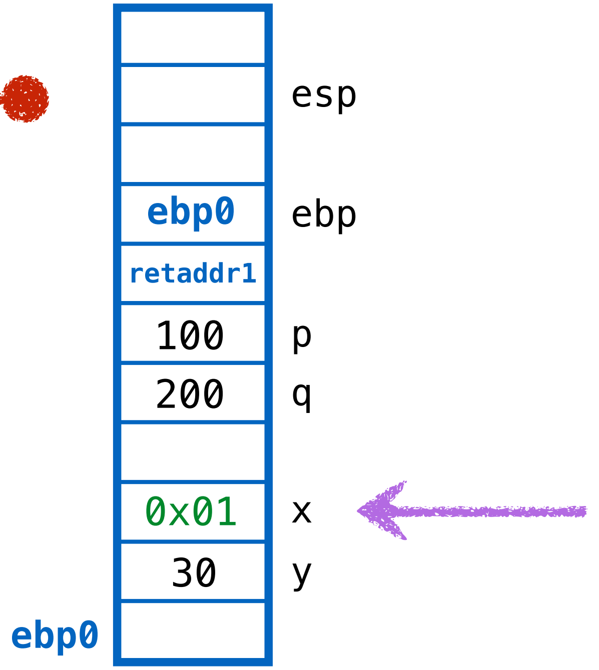


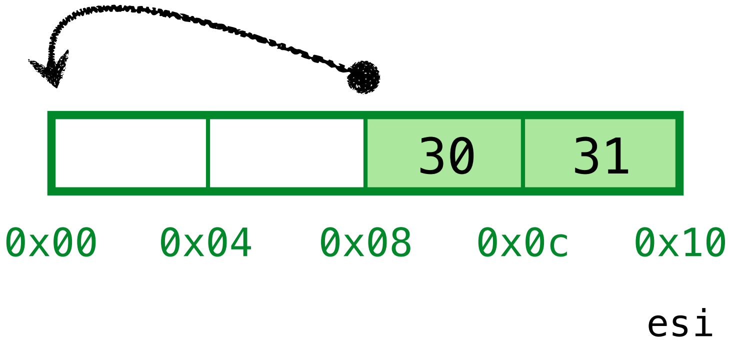
```
def foo(p, q):
  let tmp = (p, q)
  in tmp[0] + tmp[1]

let y  = foo(10, 20)
  , x  = (y, y + 1)
  , z  = foo(100, 200)
in
  x[0] + z
```

1. Compute FORWARD addrs
   e.g. 0x09 —> 0x01

ex3: garbage in the middle (with stack)

```
def foo(p, q):
  let tmp = (p, q)
  in tmp[0] + tmp[1]

let y  = foo(10, 20)
  , x  = (y, y + 1)
  , z  = foo(100, 200)
in
  x[0] + z
```



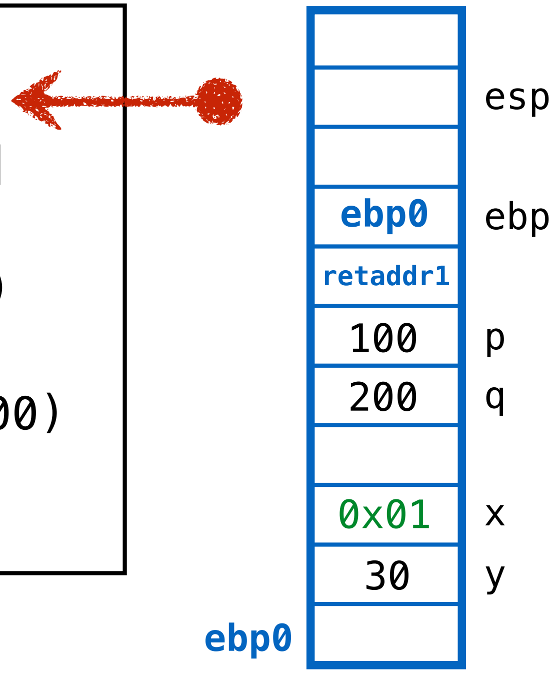


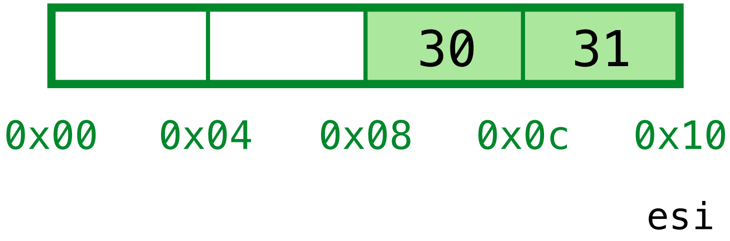
1. Compute FORWARD addrs
   e.g. 0x09 —> 0x01

2. REDIRECT addrs on stack

ex3: garbage in the middle (with stack)

```
def foo(p, q):
  let tmp = (p, q)
  in tmp[0] + tmp[1]

let y  = foo(10, 20)
  , x  = (y, y + 1)
  , z  = foo(100, 200)
in
  x[0] + z
```





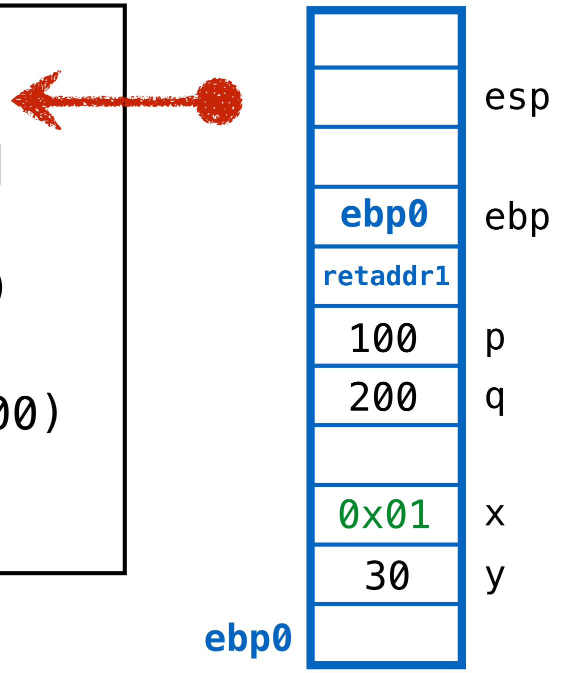
1. Compute FORWARD addrs
   e.g. 0x09 —> 0x01

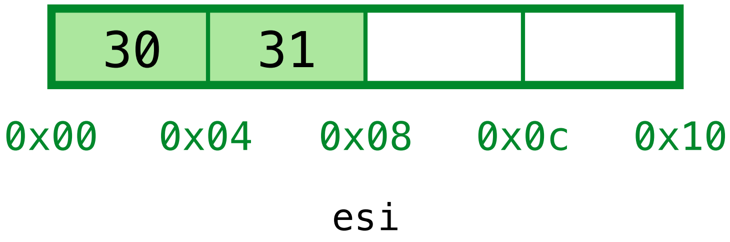
2. REDIRECT addrs on stack

3. COMPACT cells on heap

ex3: garbage in the middle (with stack)

```
def foo(p, q):
  let tmp = (p, q)
  in tmp[0] + tmp[1]

let y  = foo(10, 20)
  , x  = (y, y + 1)
  , z  = foo(100, 200)
in
  x[0] + z
```





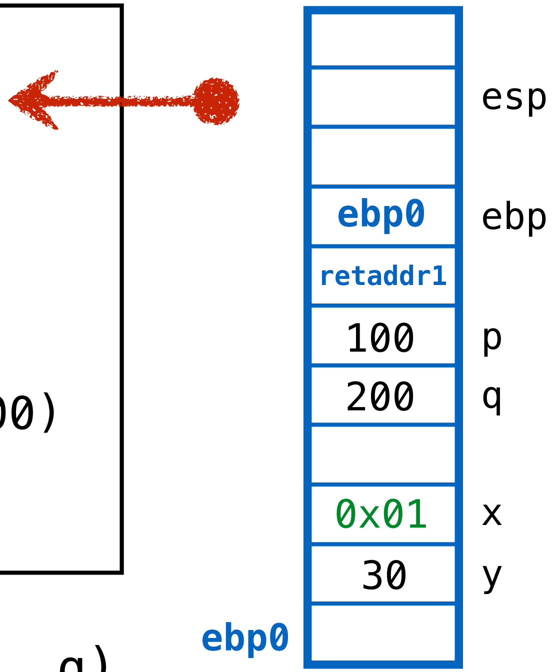
1. Compute FORWARD addrs
   e.g. `0x09` —> `0x01`

2. REDIRECT addrs on stack

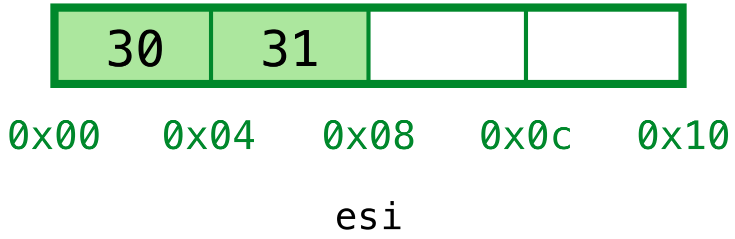
3. COMPACT cells on heap

ex3: garbage in the middle (with stack)

```
def foo(p, q):
  let tmp = (p, q)
  in tmp[0] + tmp[1]

let y  = foo(10, 20)
  , x  = (y, y + 1)
  , z  = foo(100, 200)
in
  x[0] + z
```

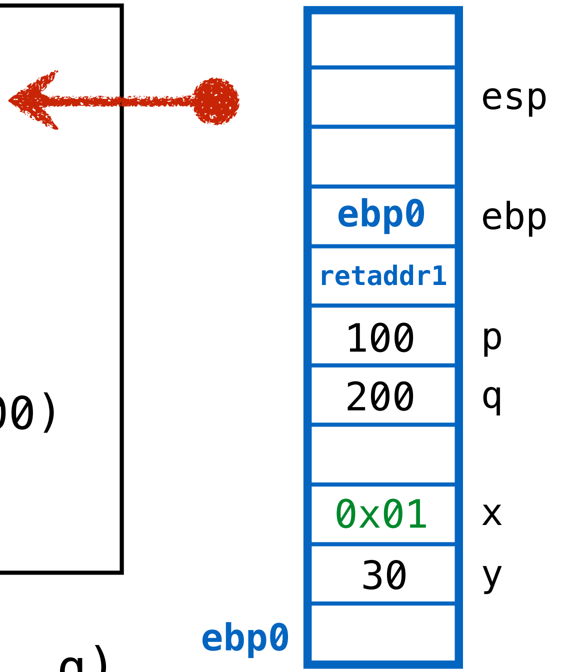


Yay! Have space for `(p, q)`

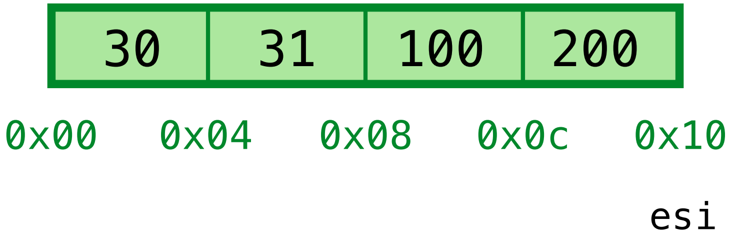


ex3: garbage in the middle (with stack)

```
def foo(p, q):
  let tmp = (p, q)
  in tmp[0] + tmp[1]

let y  = foo(10, 20)
  , x  = (y, y + 1)
  , z  = foo(100, 200)
in
  x[0] + z
```

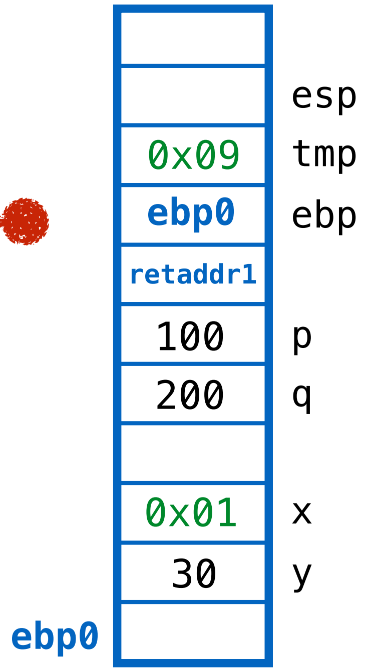


Yay! Have space for `(p, q)`



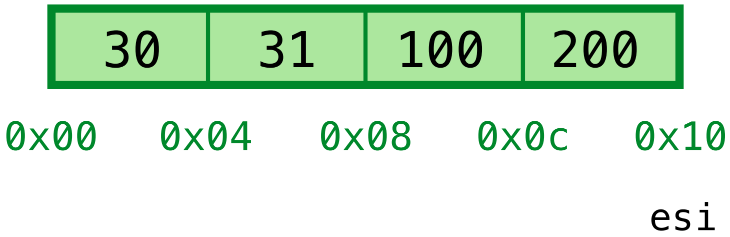
ex3: garbage in the middle (with stack)

```
def foo(p, q):
   let tmp = (p, q)
   in tmp[0] + tmp[1]

let y  = foo(10, 20)
  , x  = (y, y + 1)
  , z  = foo(100, 200)
in
  x[0] + z
```



		esp
	0x09	tmp
	ebp0	ebp
	retaddr1	
	100	p
	200	q
	0x01	x
	30	y
ebp0		

Return (eax) = 300



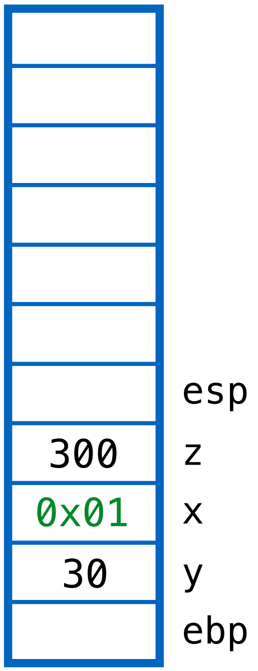
30	31	100	200

0x00 0x04 0x08 0x0c 0x10

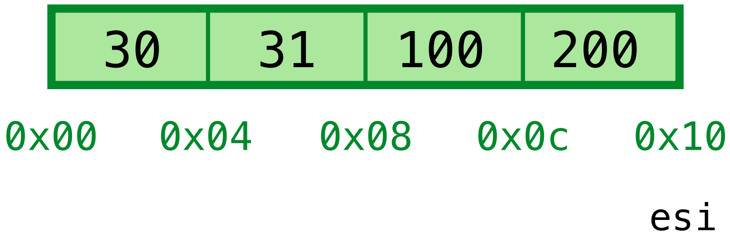
esi

ex3: garbage in the middle (with stack)

```
def foo(p, q):
   let tmp = (p, q)
   in tmp[0] + tmp[1]

let y  = foo(10, 20)
  , x  = (y, y + 1)
  , z  = foo(100, 200)
in
  x[0] + z
```

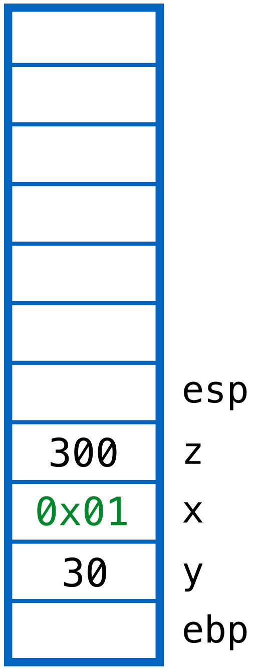


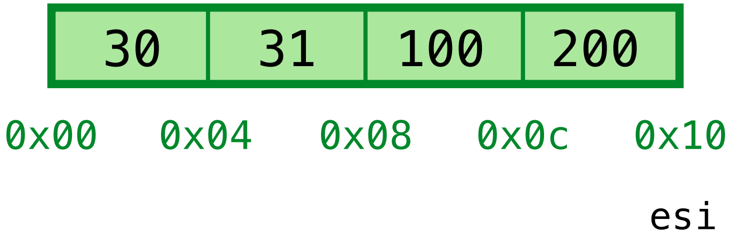
Return (eax) = 300



ex3: garbage in the middle (with stack)

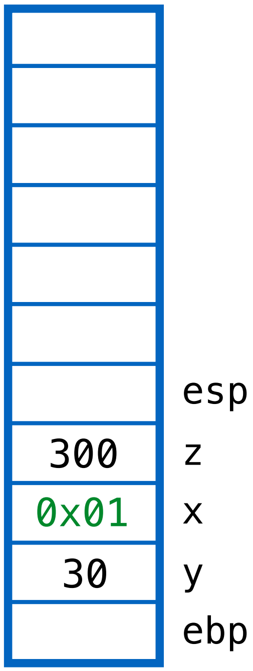
```
def foo(p, q):
  let tmp = (p, q)
  in tmp[0] + tmp[1]

let y  = foo(10, 20)
  , x  = (y, y + 1)
  , z  = foo(100, 200)
in
  x[0] + z
```



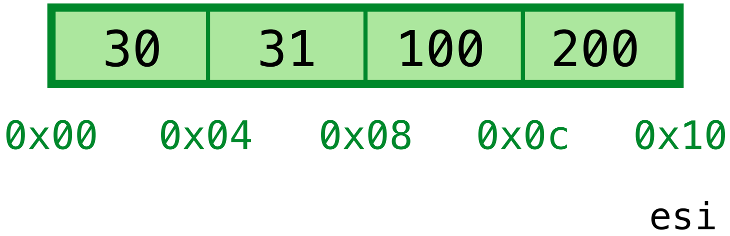


ex3: garbage in the middle (with stack)

```
def foo(p, q):
  let tmp = (p, q)
  in tmp[0] + tmp[1]

let y  = foo(10, 20)
  , x  = (y, y + 1)
  , z  = foo(100, 200)
in
  x[0] + z
```



Return (eax) = 30+300 = 330

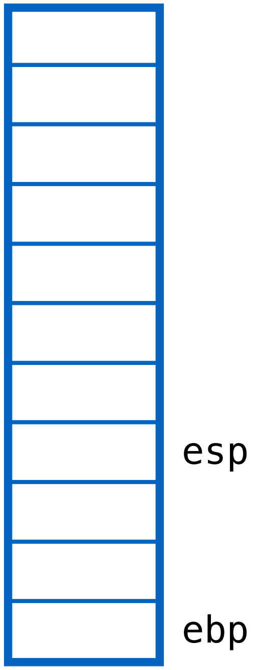


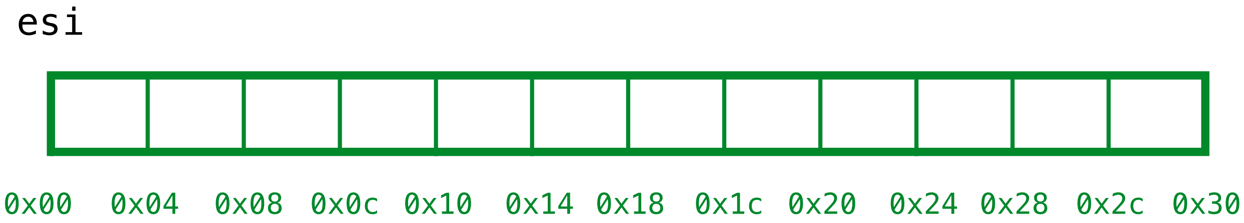
FOX / GC

Example 4

ex4: recursive data

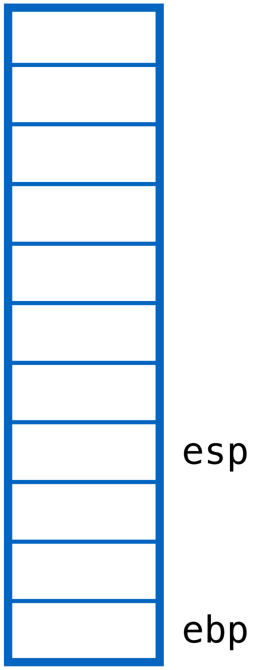
```
def range(i, j):
  if (j <= i): false else: (i,range(i+1, j))

def sum(l):
  if l == false: 0 else: l[0] + sum(l[1])

let t1 =
         let l1 = range(0, 3)
         in sum(l1)
  , l  = range(t1, t1 + 3)
in
  (1000, l)
```



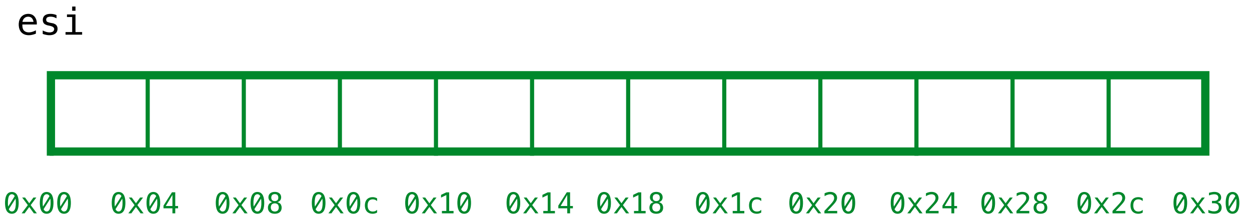


ex4: recursive data

```
def range(i, j):
  if (j <= i): false else: (i,range(i+1, j))

def sum(l):
  if l == false: 0 else: l[0] + sum(l[1])

let t1 =
         let l1 = range(0, 3)
         in sum(l1)
  , l  = range(t1, t1 + 3)
in
  (1000, l)
```

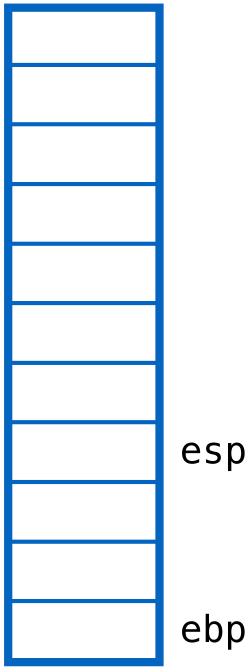


call range(0, 3)



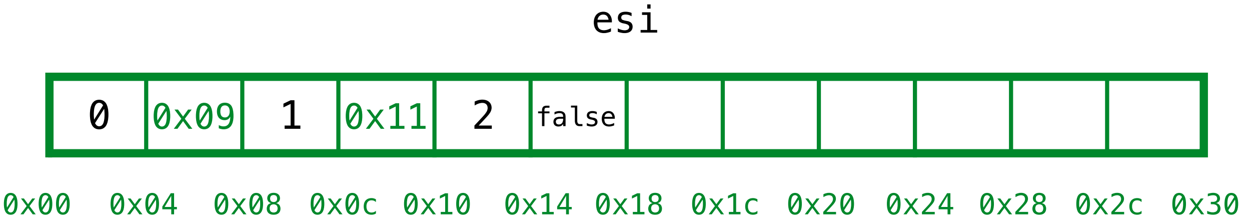
ex4: recursive data

```
def range(i, j):
  if (j <= i): false else: (i,range(i+1, j))

def sum(l):
  if l == false: 0 else: l[0] + sum(l[1])

let t1 =
         let l1 = range(0, 3)
         in sum(l1)
  , l  = range(t1, t1 + 3)
in
  (1000, l)
```



QUIZ: What is heap when `range(0,3)` returns?

(A)

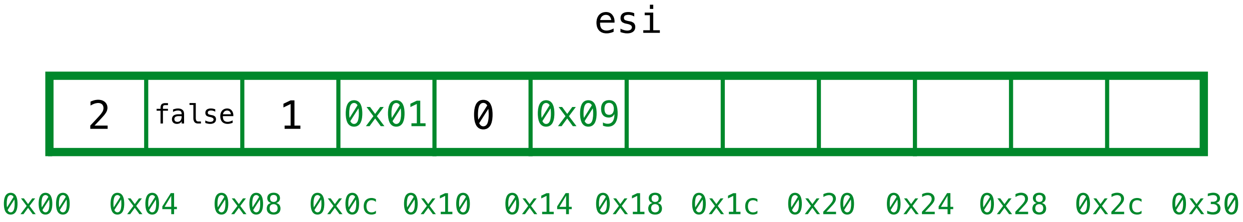


esi

0	0x09	1	0x11	2	false						

0x00 0x04 0x08 0x0c 0x10 0x14 0x18 0x1c 0x20 0x24 0x28 0x2c 0x30

(B)



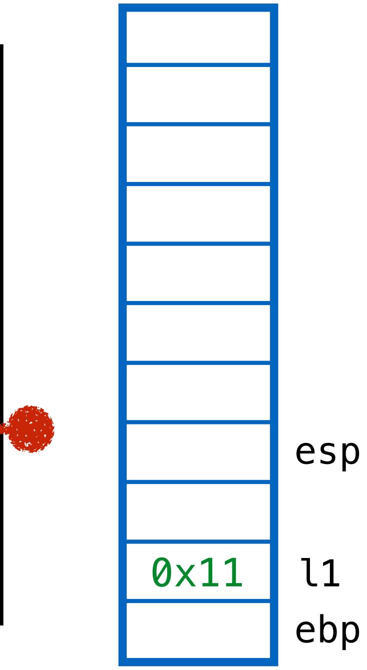
esi

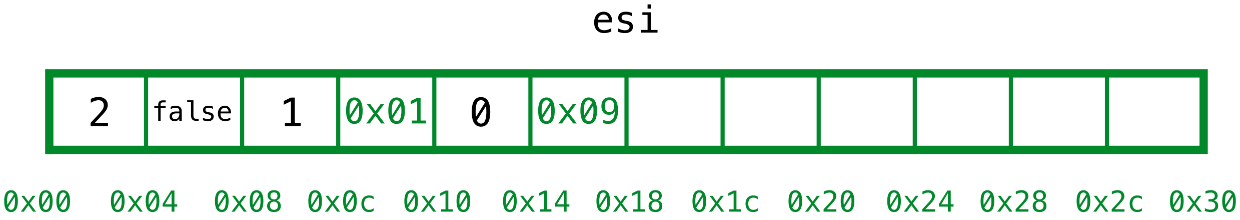
2	false	1	0x01	0	0x09						

0x00 0x04 0x08 0x0c 0x10 0x14 0x18 0x1c 0x20 0x24 0x28 0x2c 0x30

ex4: recursive data

```
def range(i, j):
  if (j <= i): false else: (i,range(i+1, j))

def sum(l):
  if l == false: 0 else: l[0] + sum(l[1])

let t1 =
         let l1 = range(0, 3)
         in sum(l1)
  , l  = range(t1, t1 + 3)
in
  (1000, l)
```

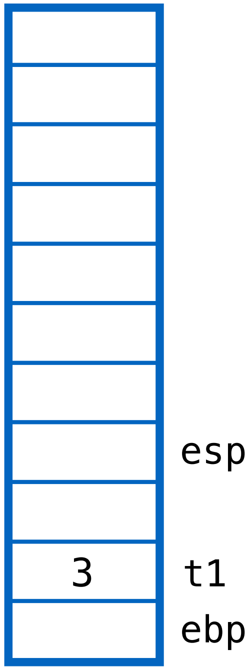




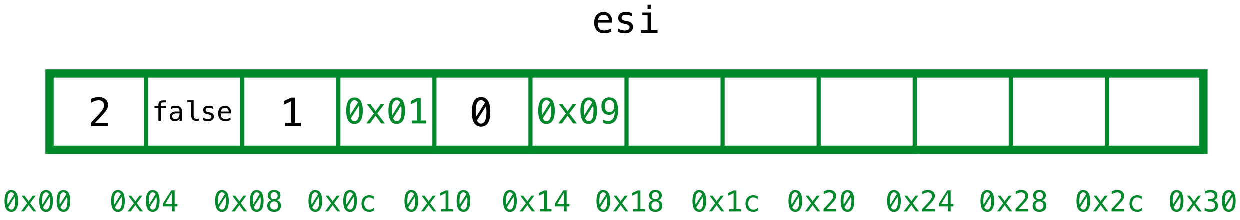
0x00	0x04	0x08	0x0c	0x10	0x14	0x18	0x1c	0x20	0x24	0x28	0x2c	0x30
2	false	1	0x01	0	0x09							

ex4: recursive data

```
def range(i, j):
  if (j <= i): false else: (i,range(i+1, j))

def sum(l):
  if l == false: 0 else: l[0] + sum(l[1])

let t1 =
         let l1 = range(0, 3)
         in sum(l1)
  , l  = range(t1, t1 + 3)
in
  (1000, l)
```



Result sum(0x11) = 3

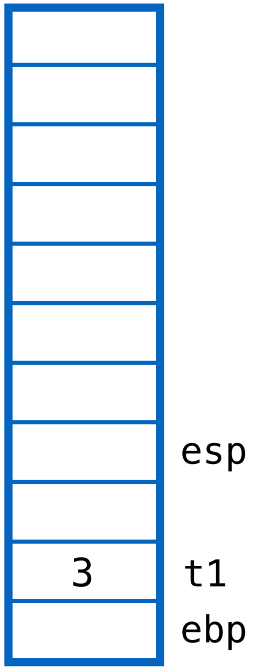


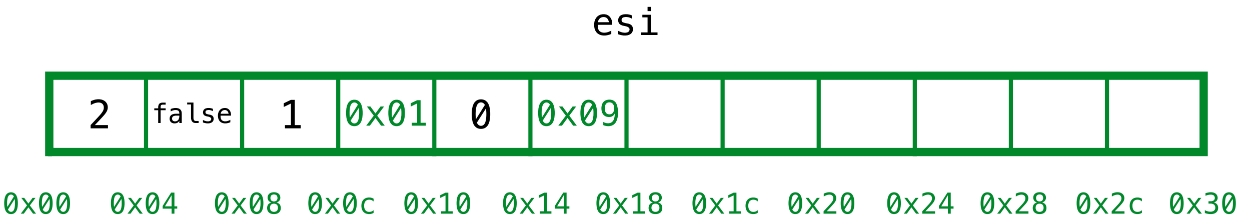
2	false	1	0x01	0	0x09						

0x00 0x04 0x08 0x0c 0x10 0x14 0x18 0x1c 0x20 0x24 0x28 0x2c 0x30

ex4: recursive data

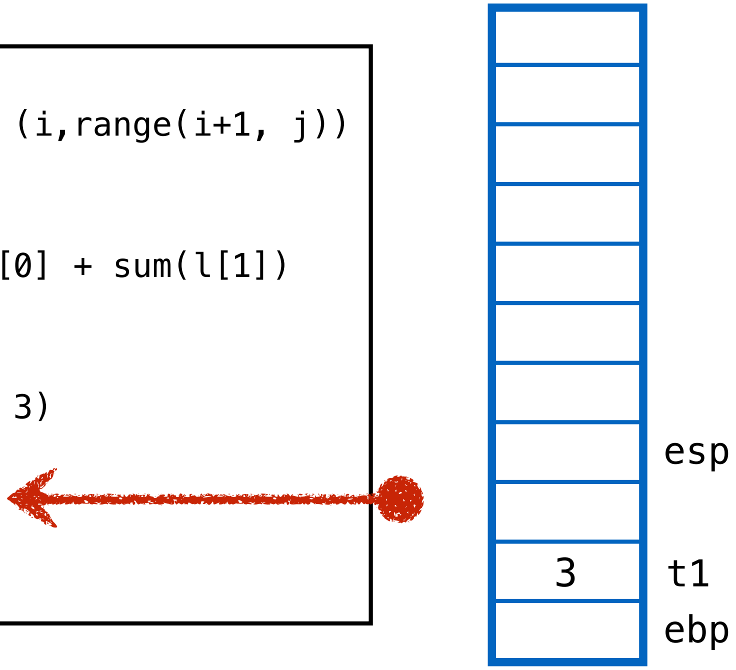
```
def range(i, j):
  if (j <= i): false else: (i,range(i+1, j))

def sum(l):
  if l == false: 0 else: l[0] + sum(l[1])

let t1 =
         let l1 = range(0, 3)
         in sum(l1)
  , l  = range(t1, t1 + 3)
in
  (1000, l)
```



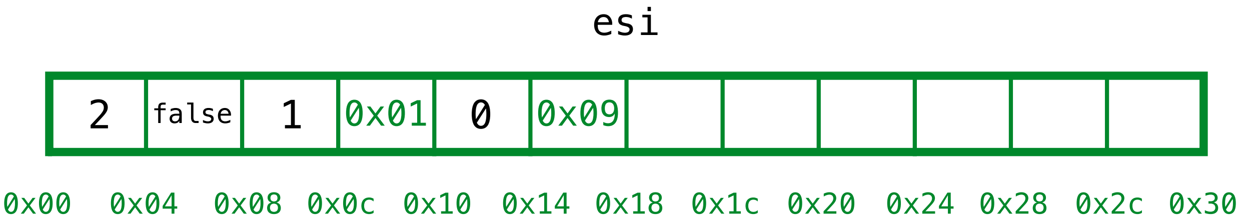


ex4: recursive data

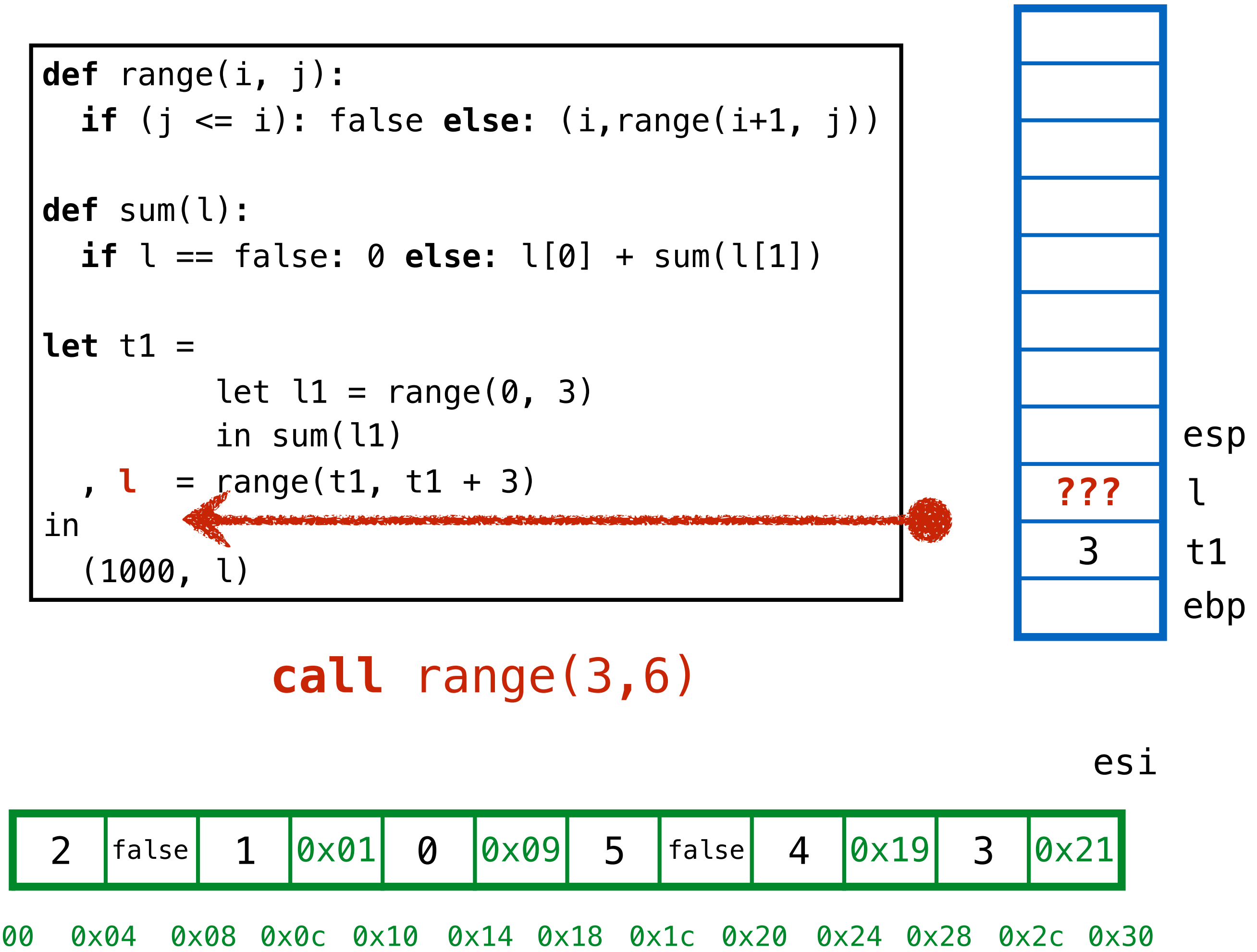
```
def range(i, j):
  if (j <= i): false else: (i,range(i+1, j))

def sum(l):
  if l == false: 0 else: l[0] + sum(l[1])

let t1 =
         let l1 = range(0, 3)
         in sum(l1)
  , l  = range(t1, t1 + 3)
in
  (1000, l)
```



call range(3,6)



ex4: recursive data



```
def range(i, j):
  if (j <= i): false else: (i,range(i+1, j))

def sum(l):
  if l == false: 0 else: l[0] + sum(l[1])

let t1 =
         let l1 = range(0, 3)
         in sum(l1)
  , l  = range(t1, t1 + 3)
in
  (1000, l)
```

	esp
???	l
3	t1
	ebp

call range(3,6)

esi

2	false	1	0x01	0	0x09	5	false	4	0x19	3	0x21

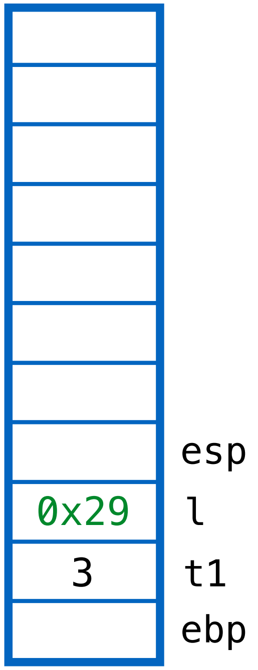
0x00 0x04 0x08 0x0c 0x10 0x14 0x18 0x1c 0x20 0x24 0x28 0x2c 0x30

QUIZ: What is the value of l?

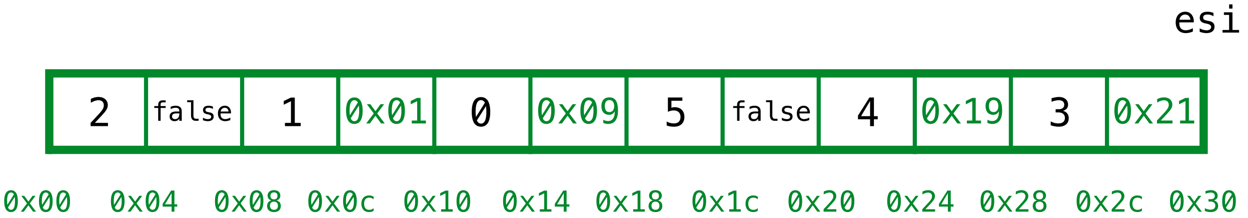
(A) 0x18 (B) 0x19 (C) 0x28 (D) 0x29 (E) 0x30

ex4: recursive data

```
def range(i, j):
  if (j <= i): false else: (i,range(i+1, j))

def sum(l):
  if l == false: 0 else: l[0] + sum(l[1])

let t1 =
         let l1 = range(0, 3)
         in sum(l1)
  , l  = range(t1, t1 + 3)
in
  (1000, l)
```



Yikes! Out of Memory!



ex4: recursive data

QUIZ: Which cells are “live” on the heap?

(A) 0x00

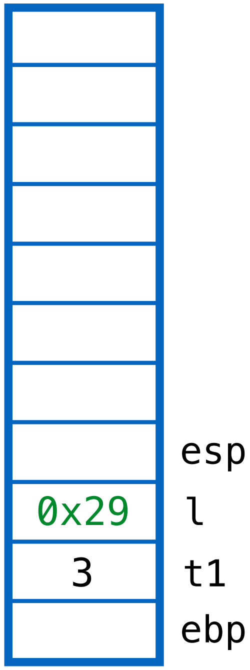
(B) 0x08

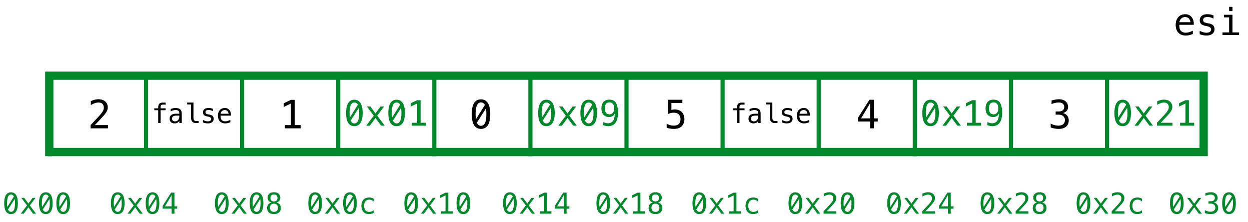
(C) 0x10

(D) 0x18

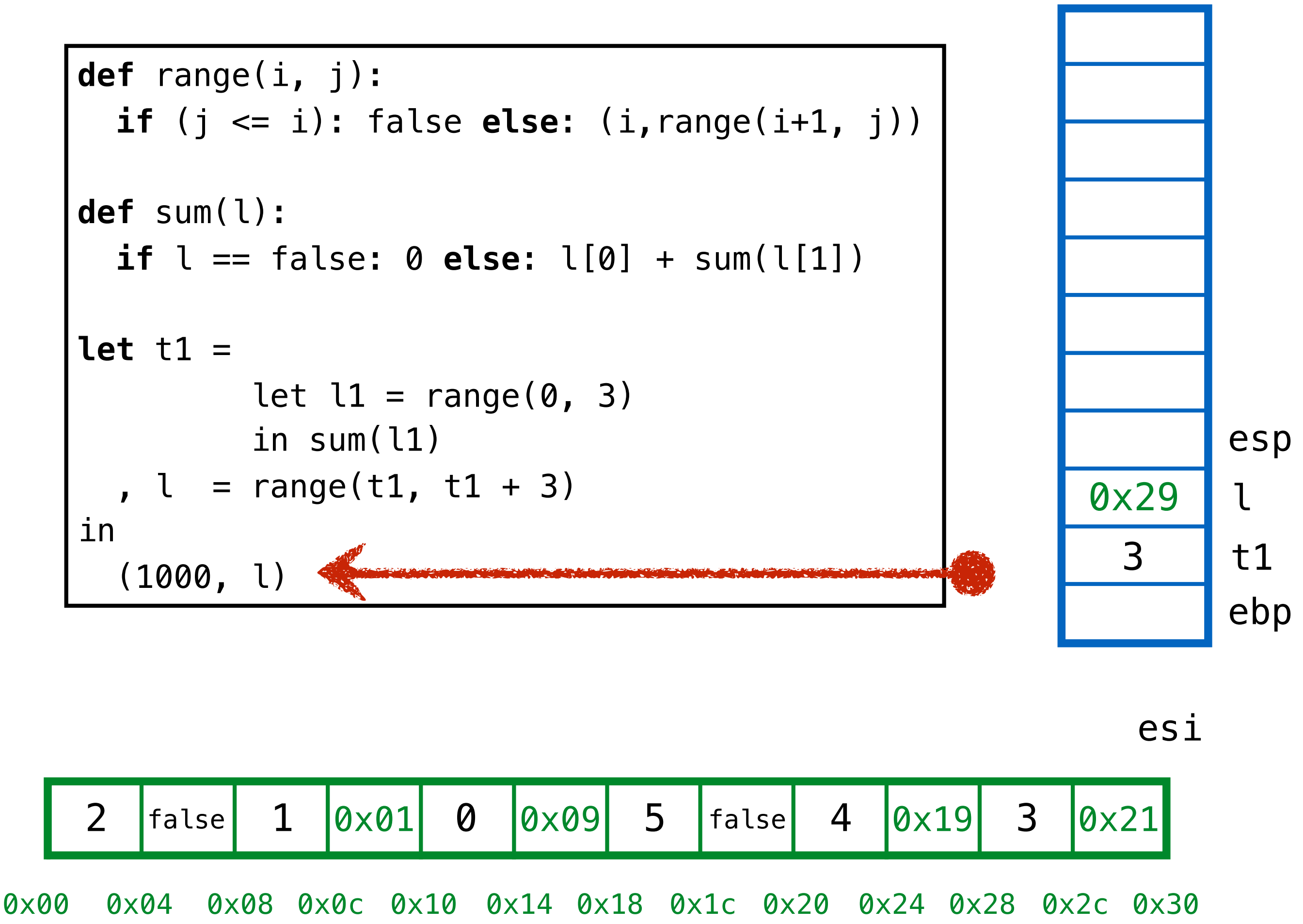
(E) 0x20

(F) 0x28





ex4: recursive data

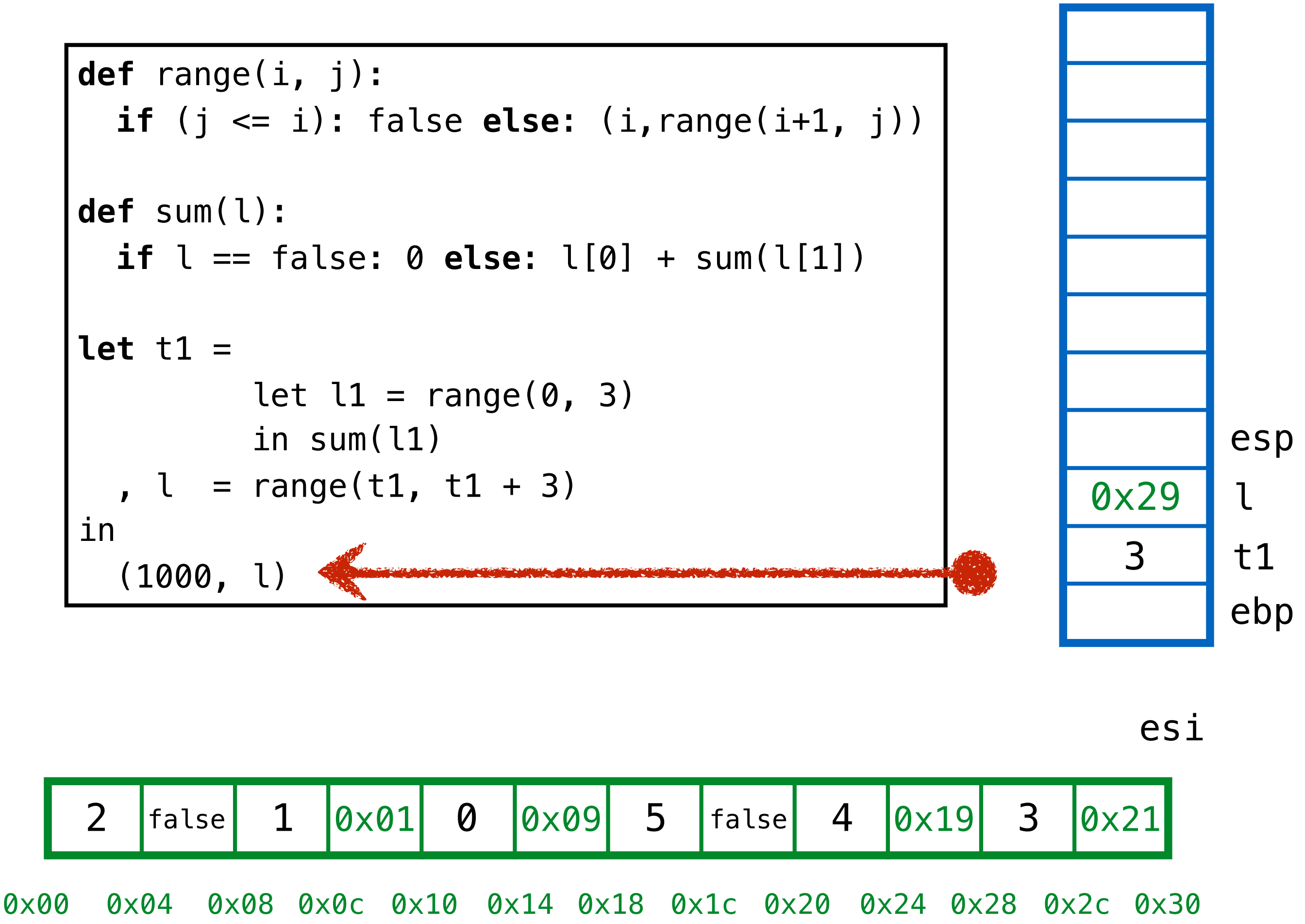


```
def range(i, j):
  if (j <= i): false else: (i,range(i+1, j))

def sum(l):
  if l == false: 0 else: l[0] + sum(l[1])

let t1 =
         let l1 = range(0, 3)
         in sum(l1)
  , l  = range(t1, t1 + 3)
in
  (1000, l)
```

2	false	1	0x01	0	0x09	5	false	4	0x19	3	0x21

1. MARK live addrs
2. Compute FORWARD addrs
3. REDIRECT addrs on stack
4. COMPACT cells on heap

ex4: recursive data

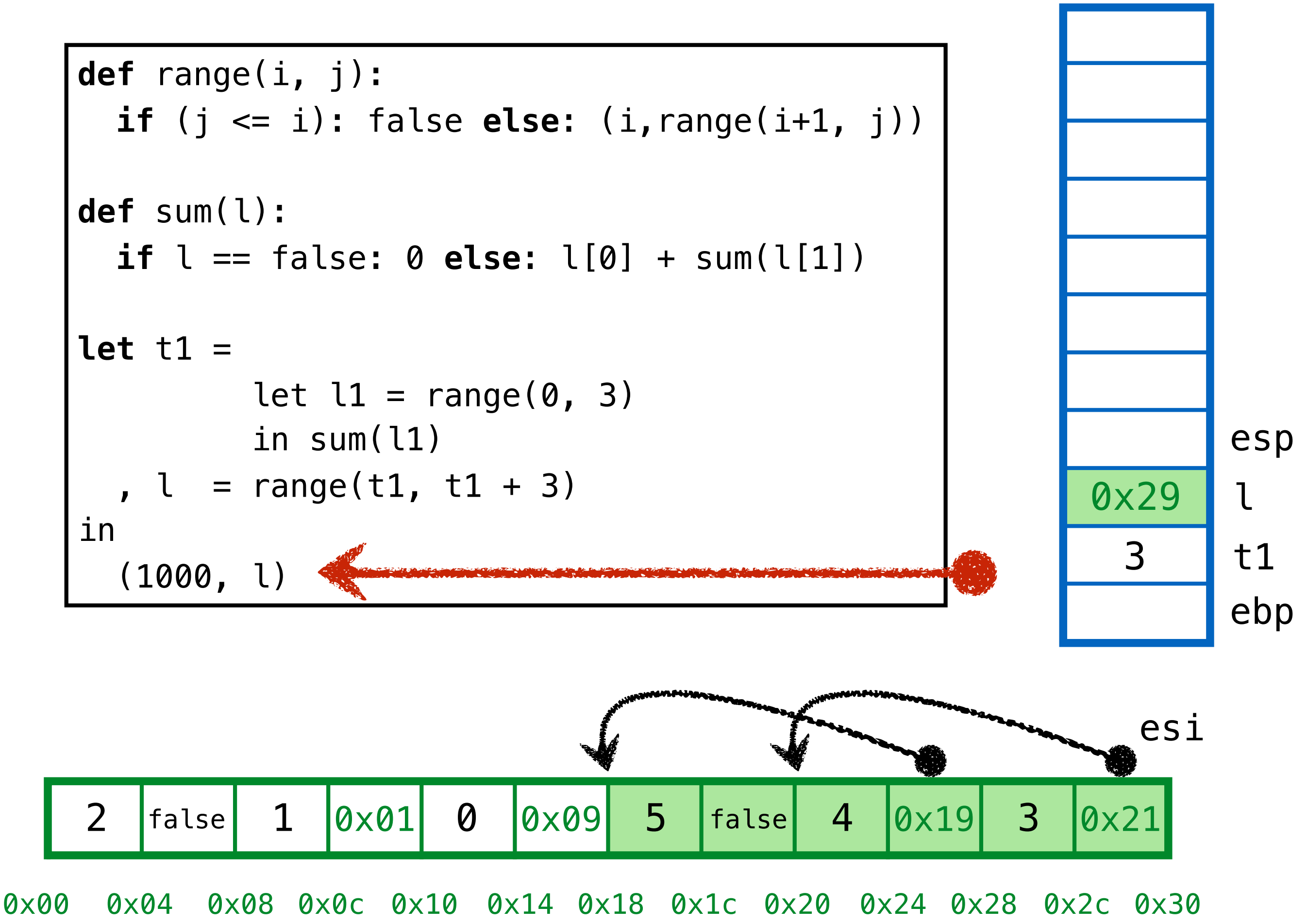


```
def range(i, j):
  if (j <= i): false else: (i,range(i+1, j))

def sum(l):
  if l == false: 0 else: l[0] + sum(l[1])

let t1 =
         let l1 = range(0, 3)
         in sum(l1)
  , l  = range(t1, t1 + 3)
in
  (1000, l)
```

1. MARK live addrs

reachable from stack

ex4: recursive data

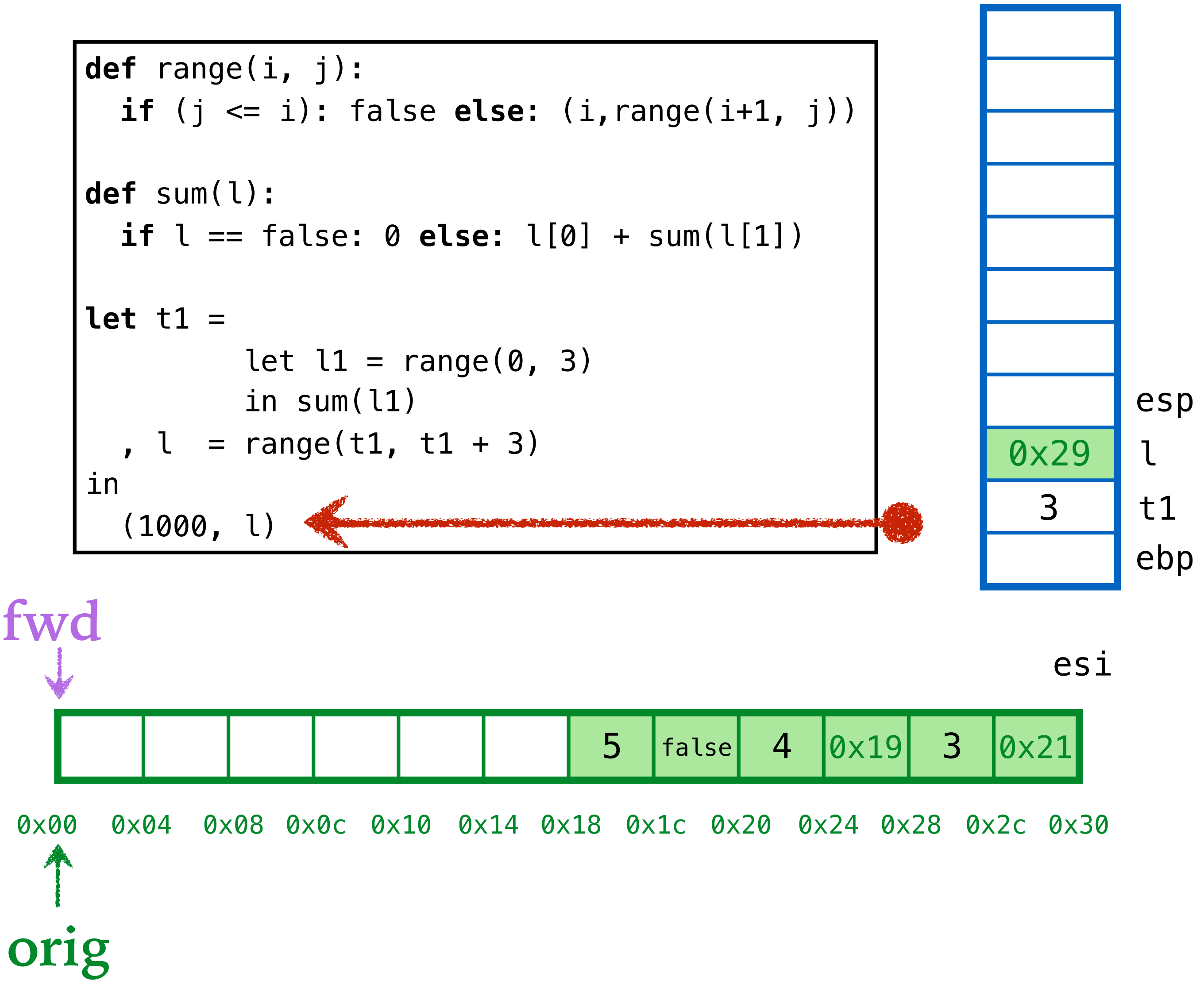


```
def range(i, j):
  if (j <= i): false else: (i,range(i+1, j))

def sum(l):
  if l == false: 0 else: l[0] + sum(l[1])

let t1 =
          let l1 = range(0, 3)
          in sum(l1)
  , l  = range(t1, t1 + 3)
in
  (1000, l)
```

1. MARK live addrs

reachable from stack

ex4: recursive data

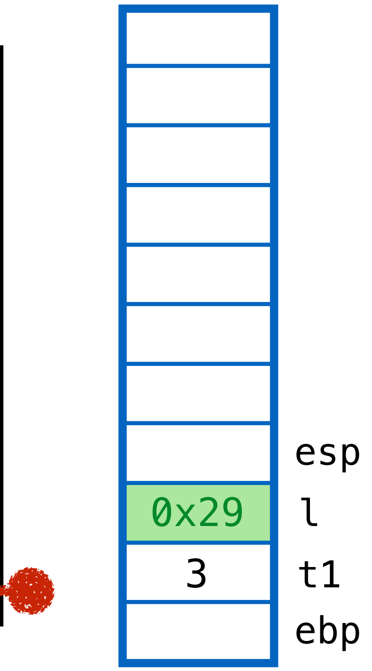


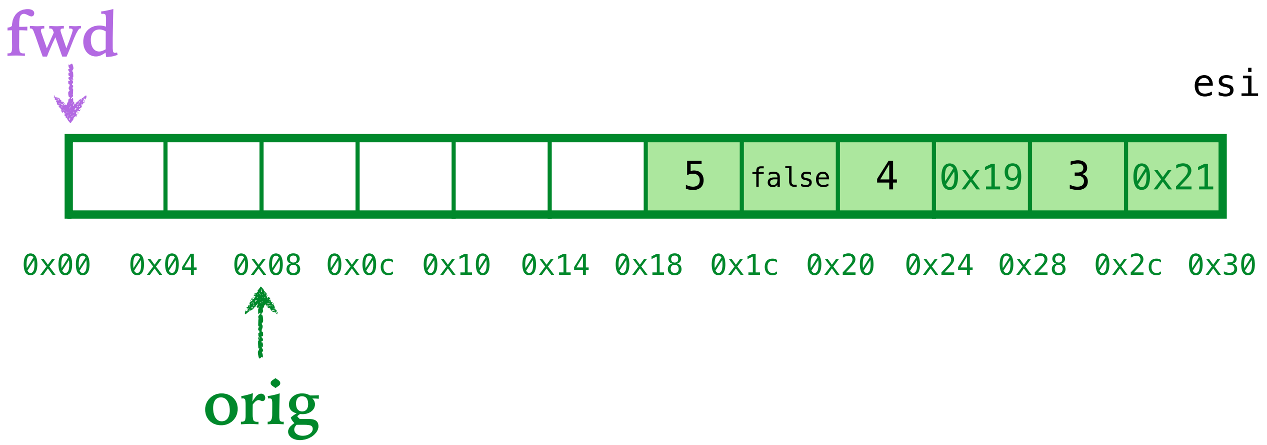
```
def range(i, j):
  if (j <= i): false else: (i,range(i+1, j))

def sum(l):
  if l == false: 0 else: l[0] + sum(l[1])

let t1 =
         let l1 = range(0, 3)
         in sum(l1)
  , l  = range(t1, t1 + 3)
in
  (1000, l)
```

2. Compute FORWARD addrs

ex4: recursive data

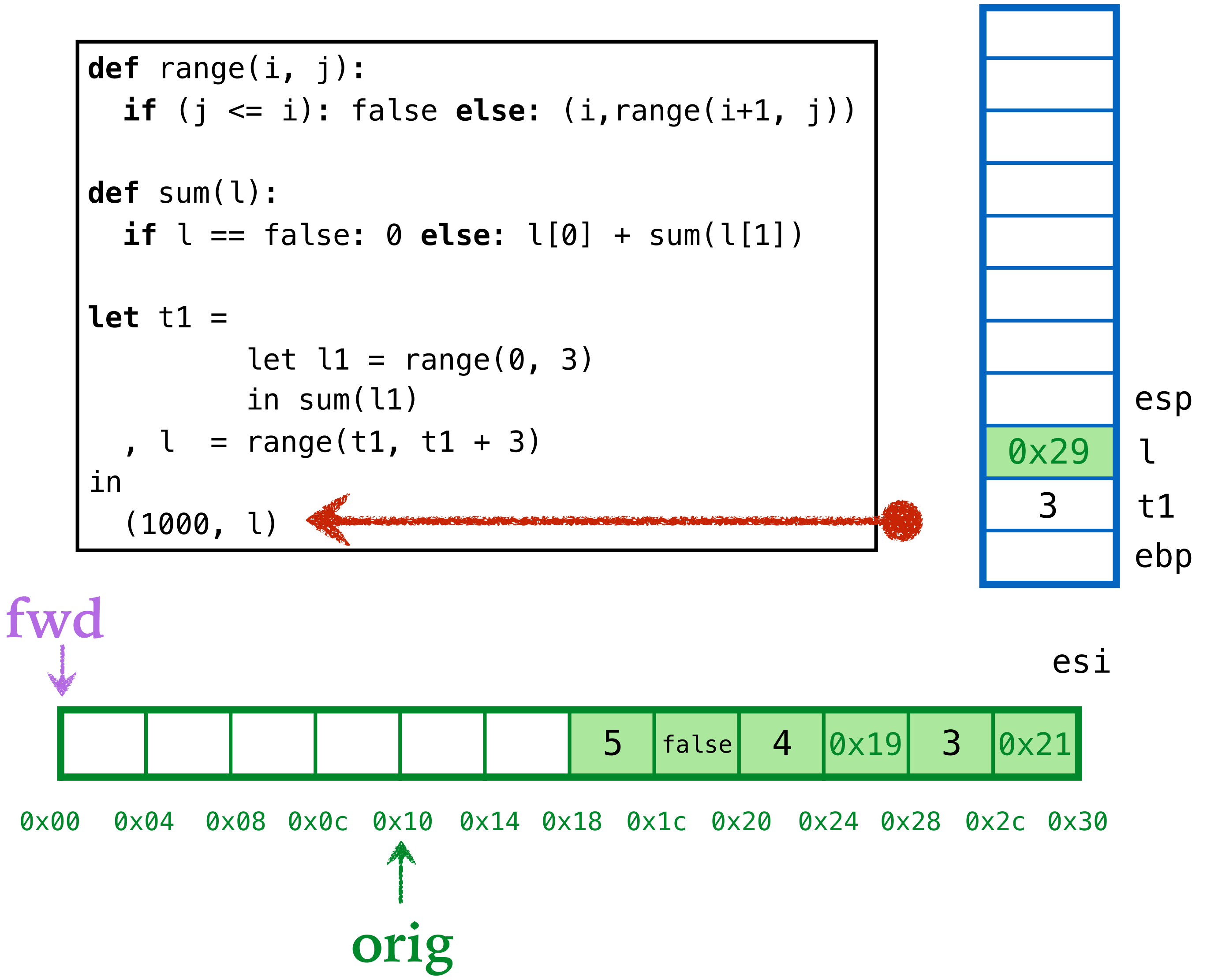
```
def range(i, j):
  if (j <= i): false else: (i,range(i+1, j))

def sum(l):
  if l == false: 0 else: l[0] + sum(l[1])

let t1 =
         let l1 = range(0, 3)
         in sum(l1)
  , l  = range(t1, t1 + 3)
in
  (1000, l)
```





2. Compute FORWARD addrs

ex4: recursive data

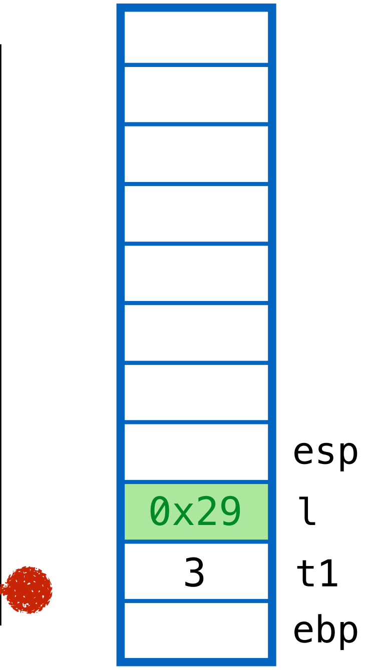


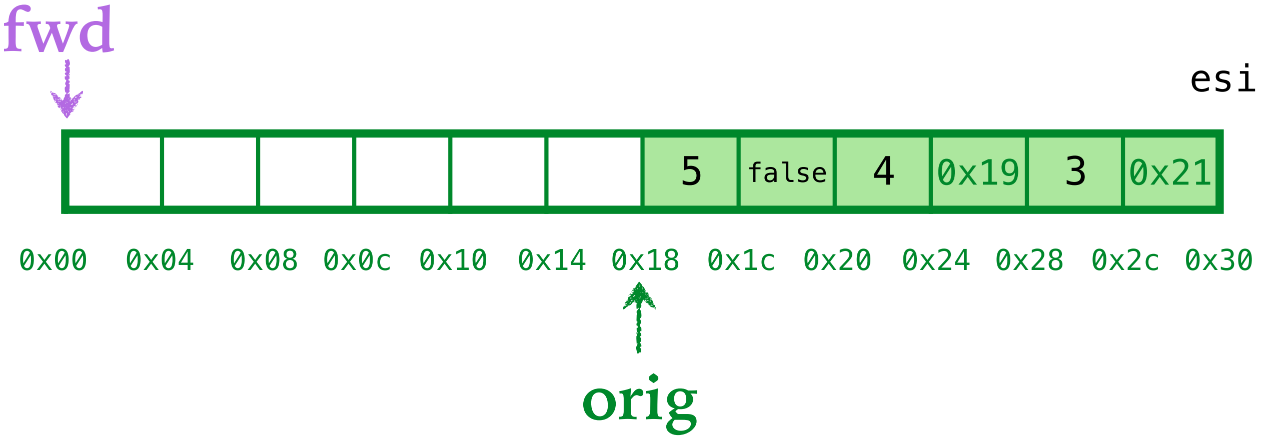
```
def range(i, j):
  if (j <= i): false else: (i,range(i+1, j))

def sum(l):
  if l == false: 0 else: l[0] + sum(l[1])

let t1 =
         let l1 = range(0, 3)
         in sum(l1)
  , l  = range(t1, t1 + 3)
in
  (1000, l)
```

2. Compute FORWARD addrs

ex4: recursive data

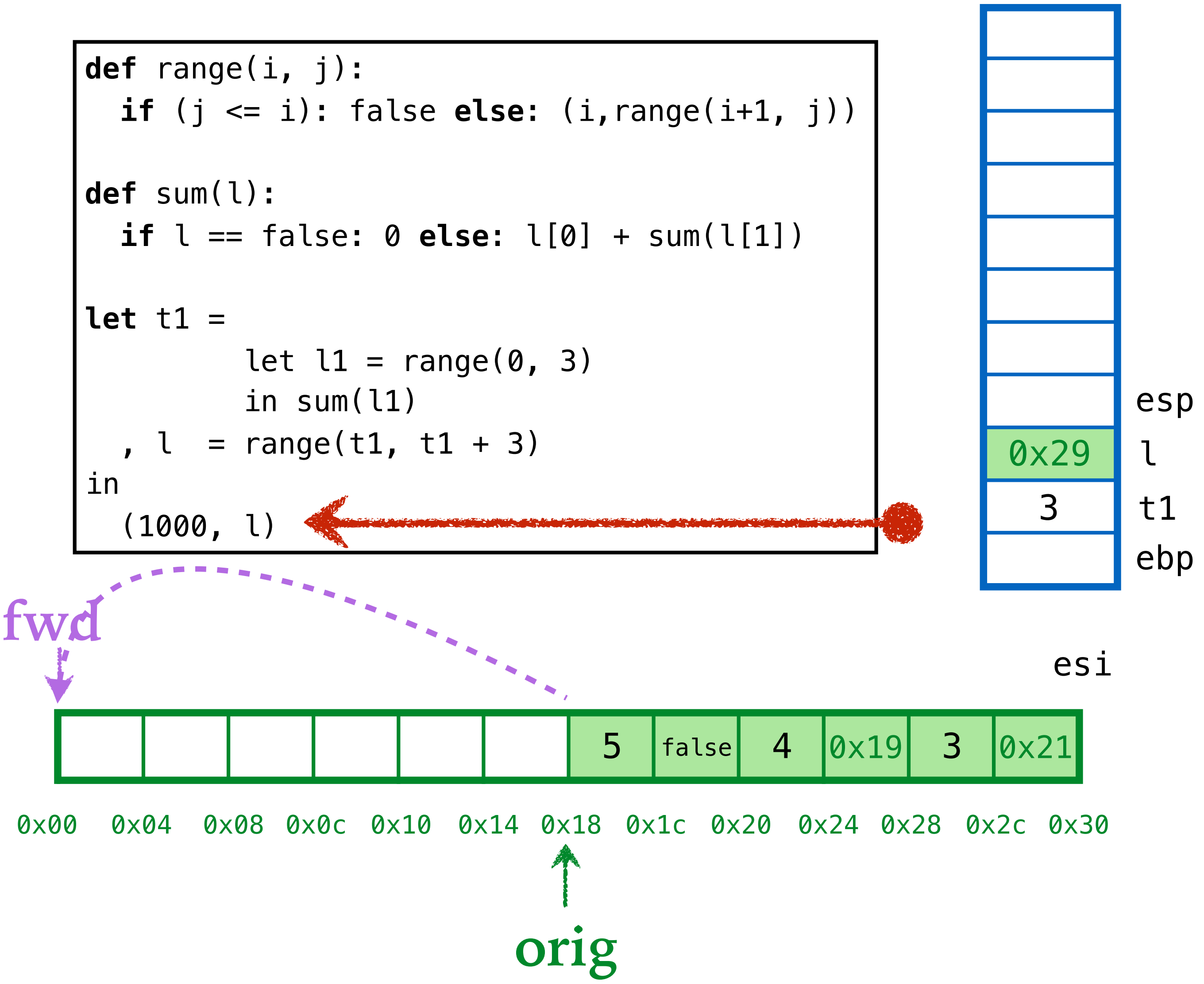
```
def range(i, j):
  if (j <= i): false else: (i,range(i+1, j))

def sum(l):
  if l == false: 0 else: l[0] + sum(l[1])

let t1 =
         let l1 = range(0, 3)
         in sum(l1)
  , l  = range(t1, t1 + 3)
in
  (1000, l)
```





2. Compute FORWARD addrs

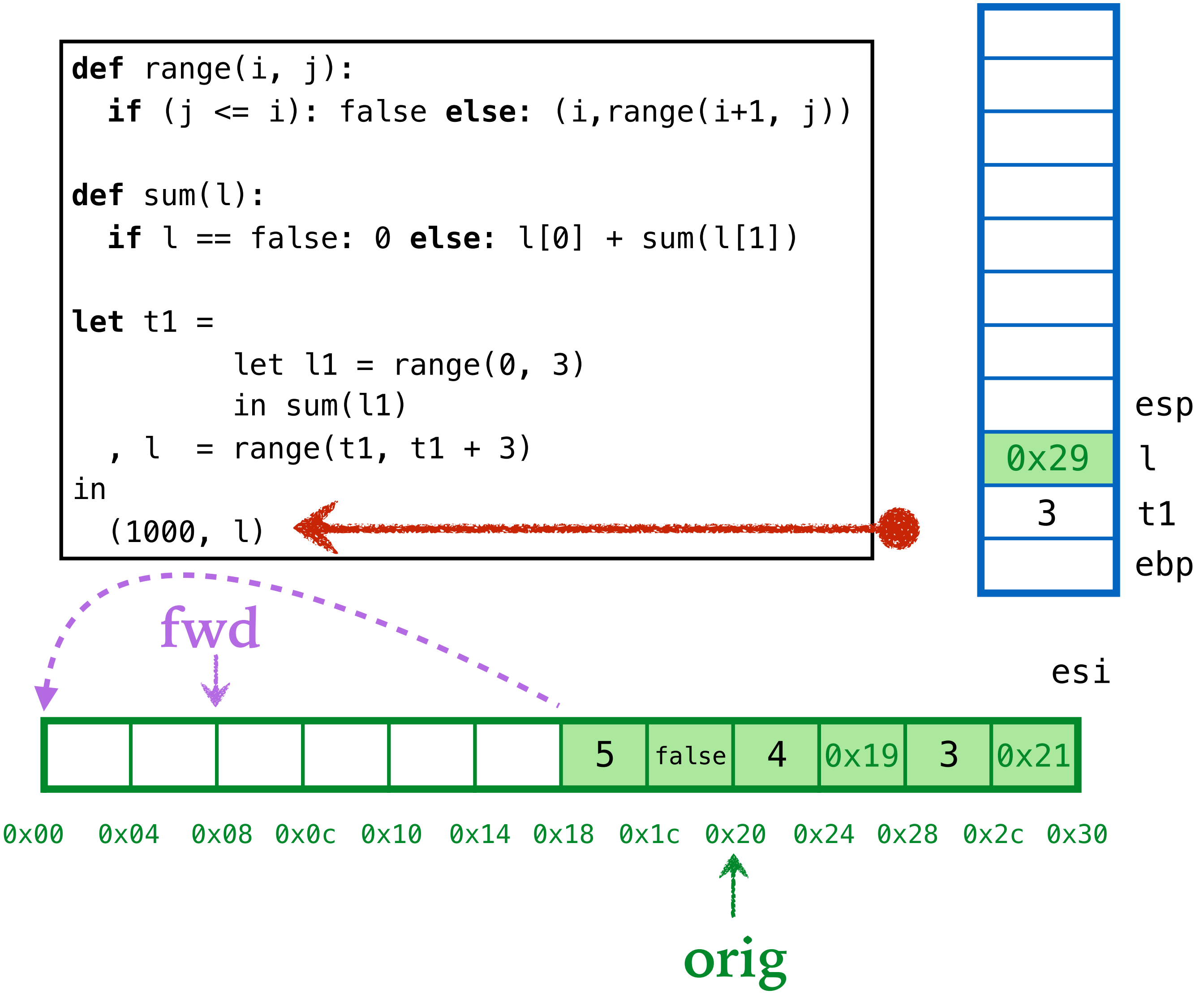
ex4: recursive data



```
def range(i, j):
  if (j <= i): false else: (i,range(i+1, j))

def sum(l):
  if l == false: 0 else: l[0] + sum(l[1])

let t1 =
         let l1 = range(0, 3)
         in sum(l1)
  , l  = range(t1, t1 + 3)
in
  (1000, l)
```

2. Compute FORWARD addrs

ex4: recursive data



```
def range(i, j):
  if (j <= i): false else: (i,range(i+1, j))

def sum(l):
  if l == false: 0 else: l[0] + sum(l[1])

let t1 =
        let l1 = range(0, 3)
        in sum(l1)
  , l  = range(t1, t1 + 3)
in
  (1000, l)
```

	esp
0x29	l
3	t1
	ebp

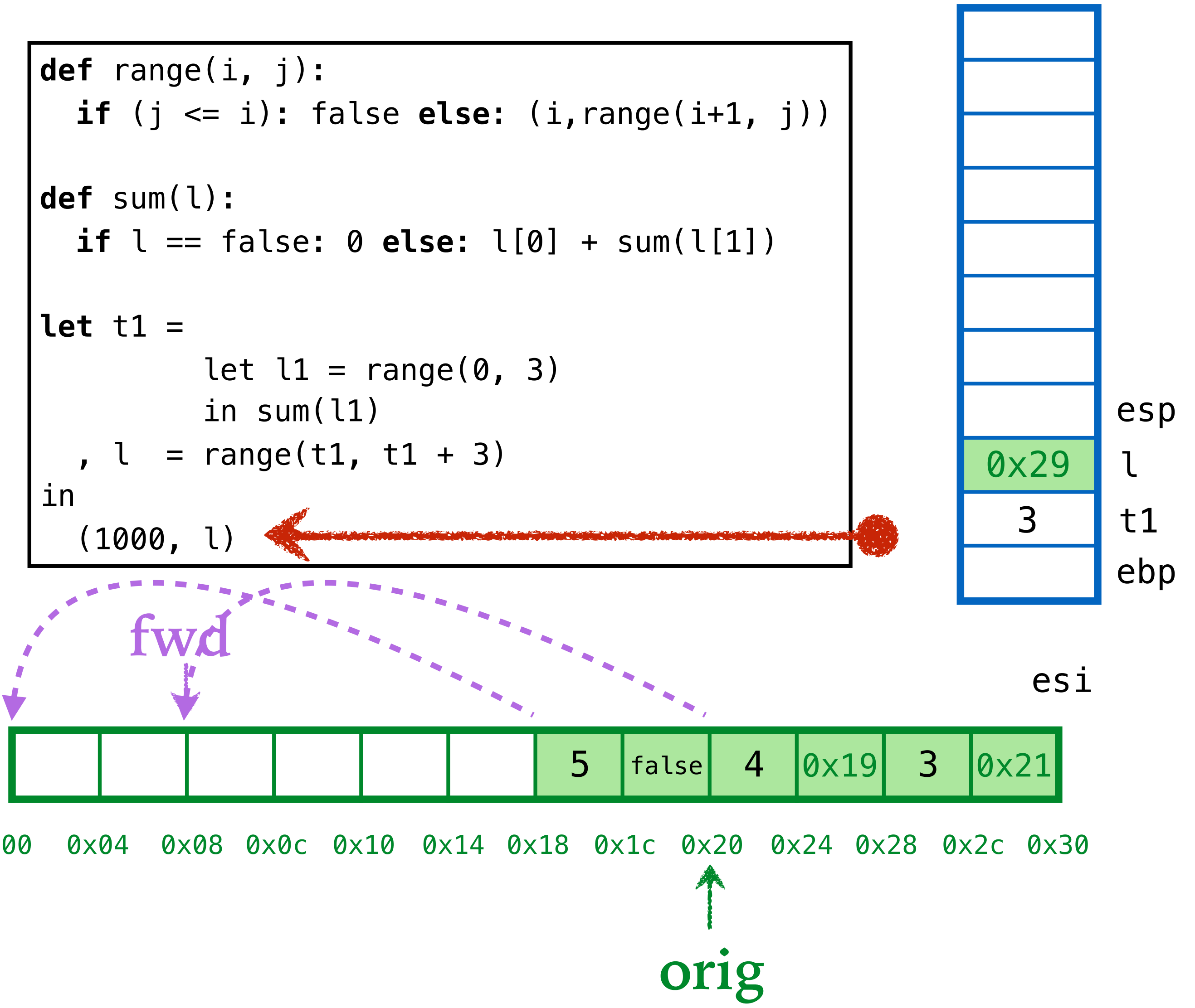
esi

						5	false	4	0x19	3	0x21

0x00 0x04 0x08 0x0c 0x10 0x14 0x18 0x1c 0x20 0x24 0x28 0x2c 0x30

2. Compute FORWARD addrs

ex4: recursive data

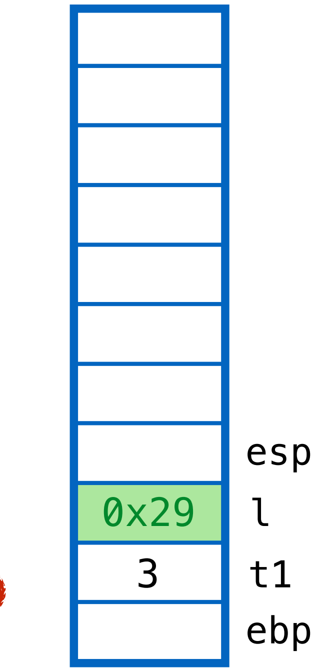


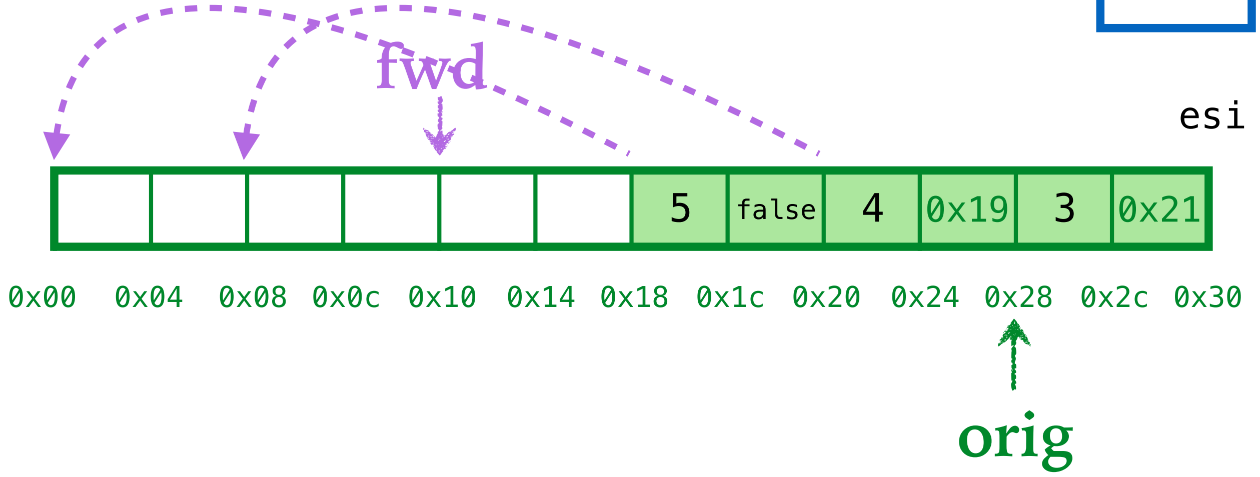
```
def range(i, j):
  if (j <= i): false else: (i,range(i+1, j))

def sum(l):
  if l == false: 0 else: l[0] + sum(l[1])

let t1 =
         let l1 = range(0, 3)
         in sum(l1)
  , l  = range(t1, t1 + 3)
in
  (1000, l)
```

2. Compute FORWARD addrs

ex4: recursive data

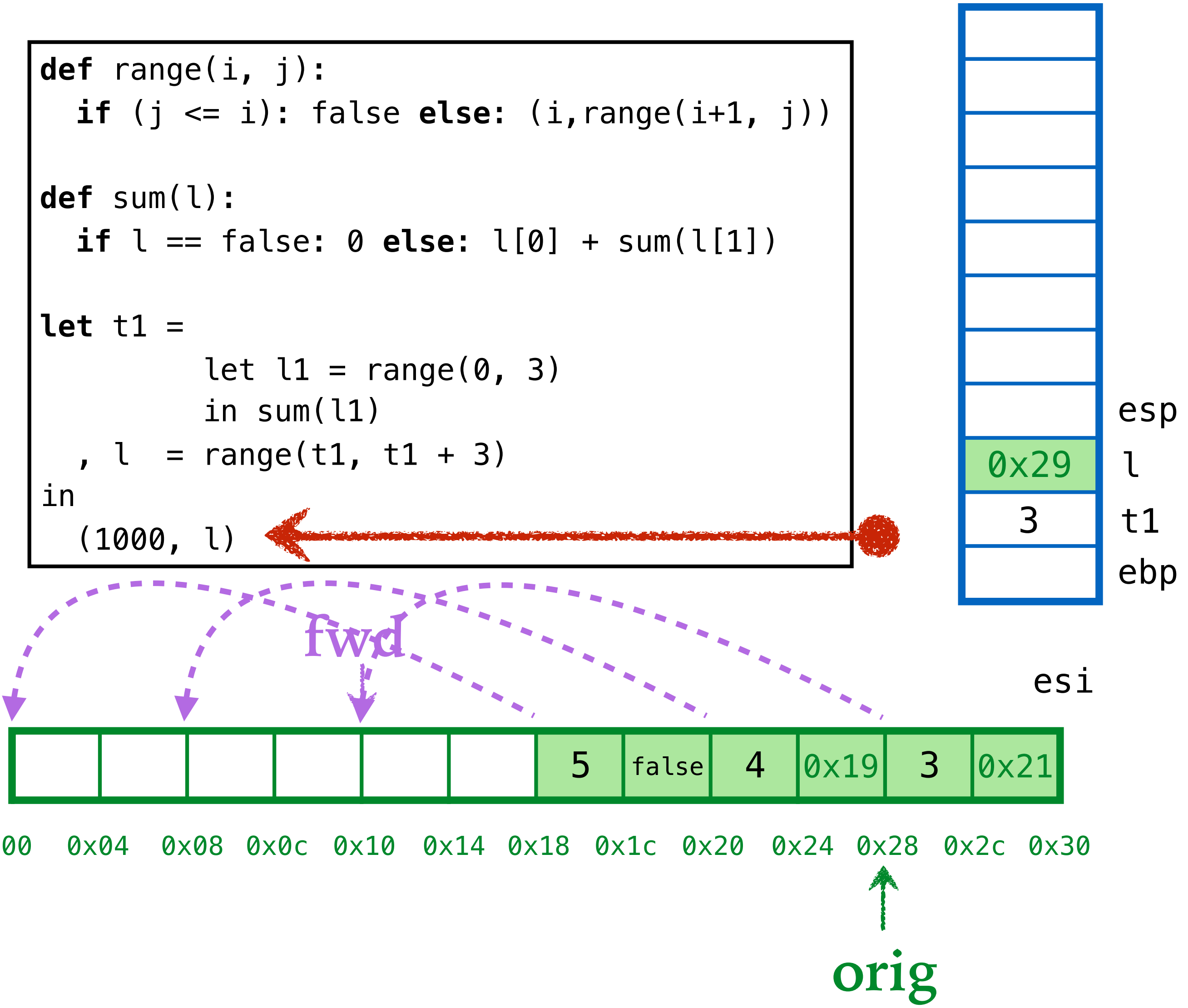
```
def range(i, j):
  if (j <= i): false else: (i,range(i+1, j))

def sum(l):
  if l == false: 0 else: l[0] + sum(l[1])

let t1 =
         let l1 = range(0, 3)
         in sum(l1)
  , l  = range(t1, t1 + 3)
in
  (1000, l)
```





2.Compute FORWARD addrs

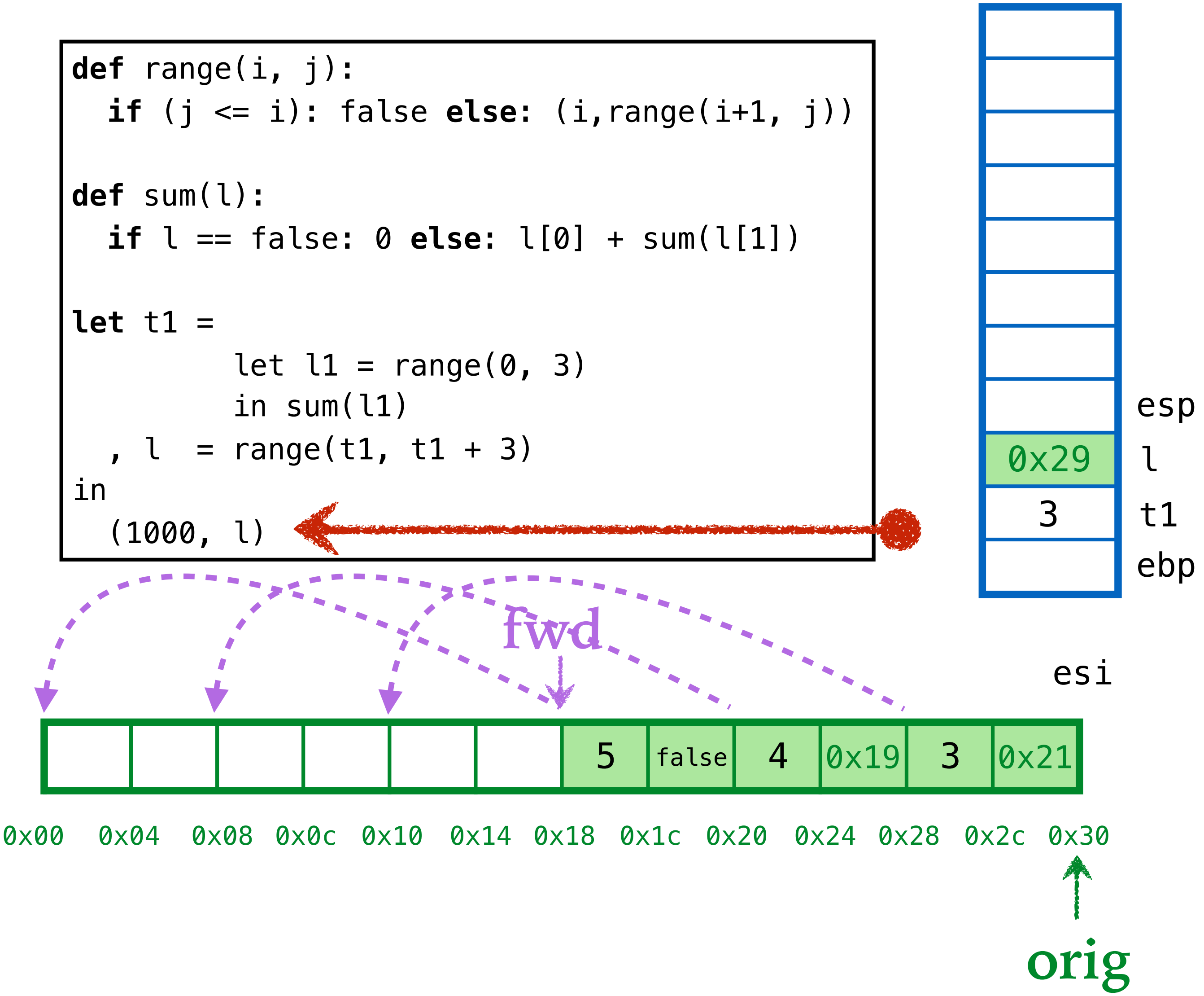
ex4: recursive data



```
def range(i, j):
  if (j <= i): false else: (i,range(i+1, j))

def sum(l):
  if l == false: 0 else: l[0] + sum(l[1])

let t1 =
         let l1 = range(0, 3)
         in sum(l1)
  , l  = range(t1, t1 + 3)
in
  (1000, l)
```

2. Compute FORWARD addrs

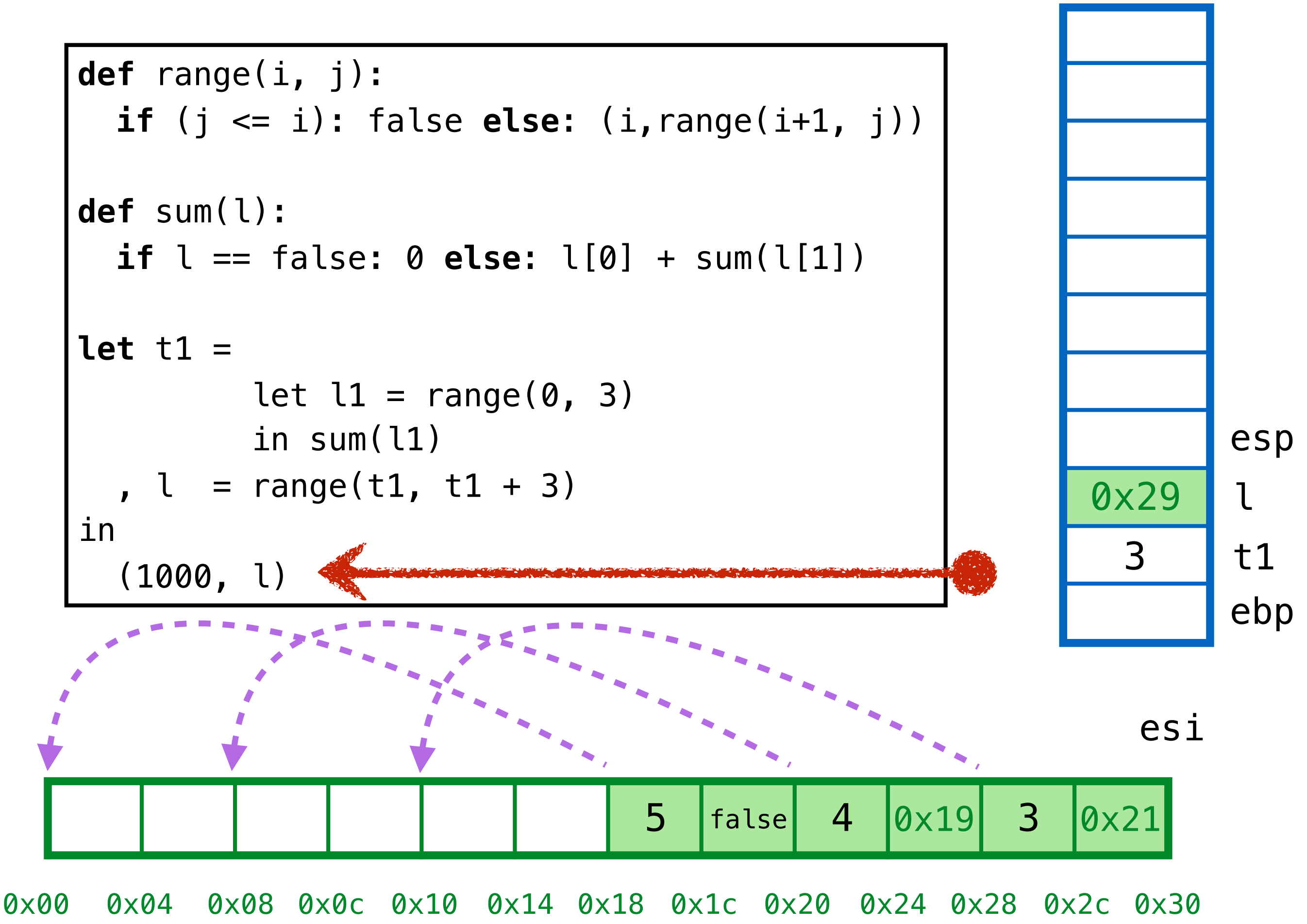
ex4: recursive data



```
def range(i, j):
  if (j <= i): false else: (i,range(i+1, j))

def sum(l):
  if l == false: 0 else: l[0] + sum(l[1])

let t1 =
         let l1 = range(0, 3)
         in sum(l1)
  , l  = range(t1, t1 + 3)
in
  (1000, l)
```

2. Compute FORWARD addrs

ex4: recursive data

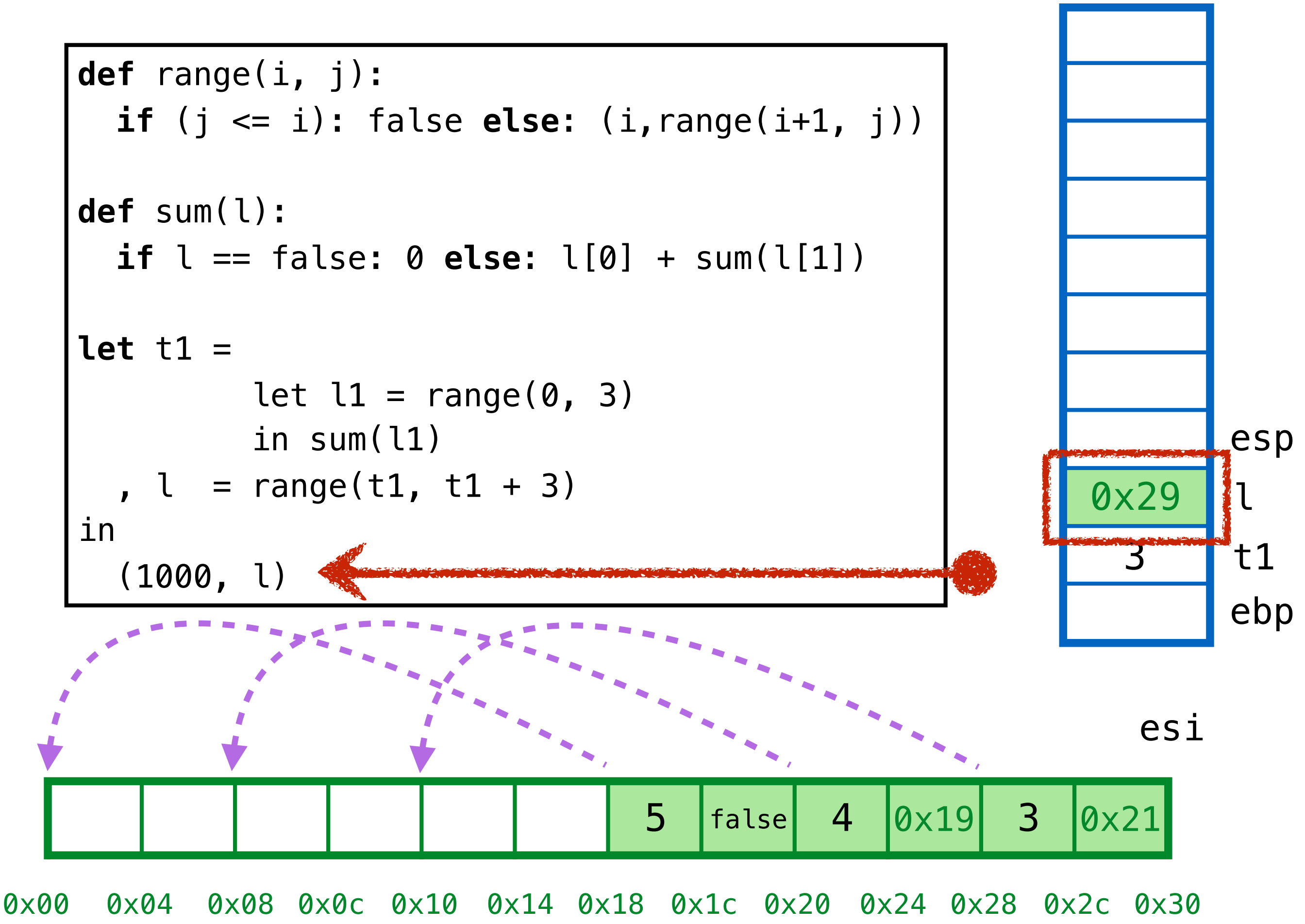


```
def range(i, j):
  if (j <= i): false else: (i,range(i+1, j))

def sum(l):
  if l == false: 0 else: l[0] + sum(l[1])

let t1 =
          let l1 = range(0, 3)
          in sum(l1)
  , l  = range(t1, t1 + 3)
in
  (1000, l)
```

2. Compute FORWARD addrs

Where should we store the forward addrs?

ex4: recursive data

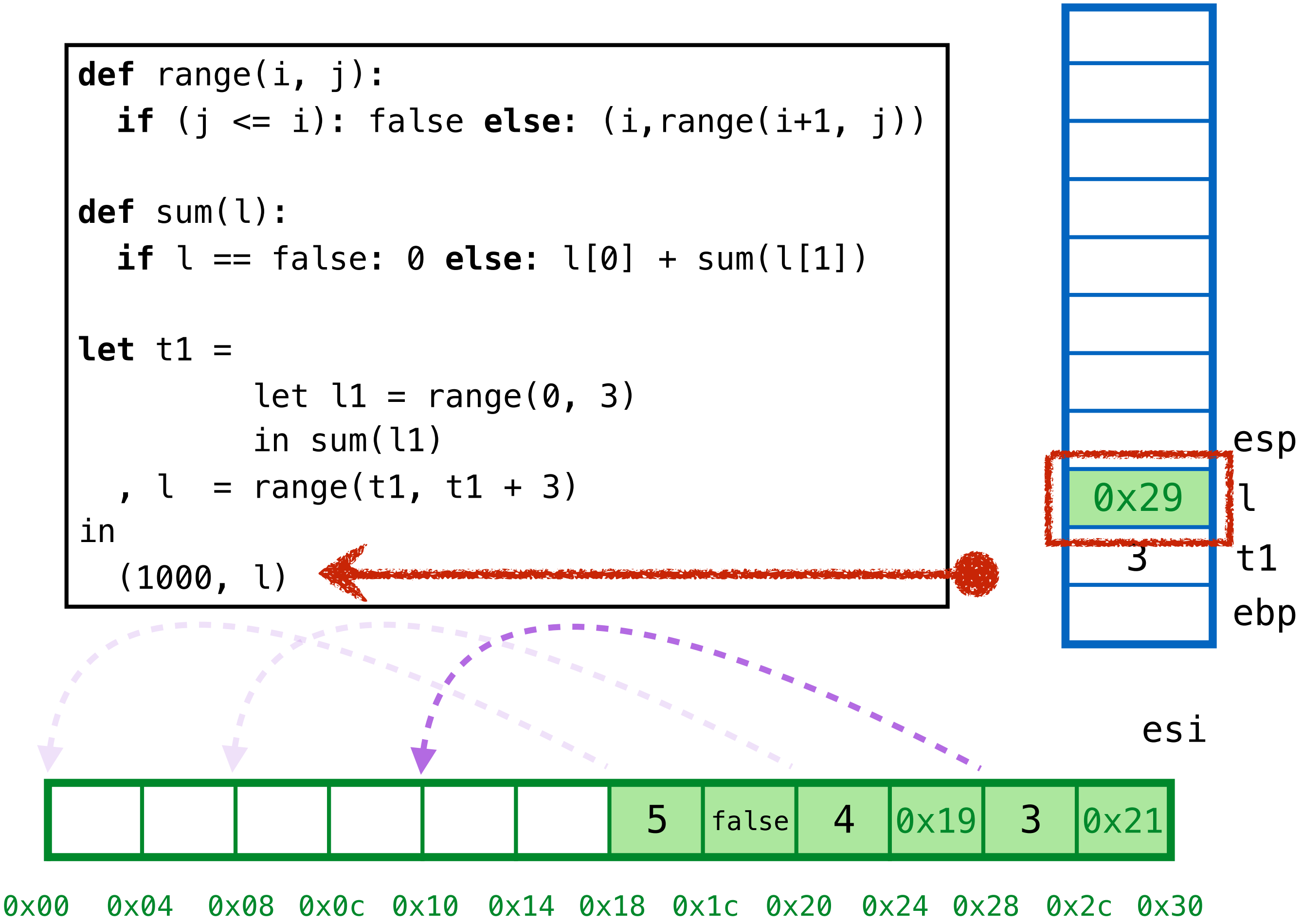


```
def range(i, j):
  if (j <= i): false else: (i,range(i+1, j))

def sum(l):
  if l == false: 0 else: l[0] + sum(l[1])

let t1 =
         let l1 = range(0, 3)
         in sum(l1)
  , l  = range(t1, t1 + 3)
in
  (1000, l)
```

0x00	0x04	0x08	0x0c	0x10	0x14	0x18	0x1c	0x20	0x24	0x28	0x2c
						5	false	4	0x19	3	0x21

3. REDIRECT addrs on stack

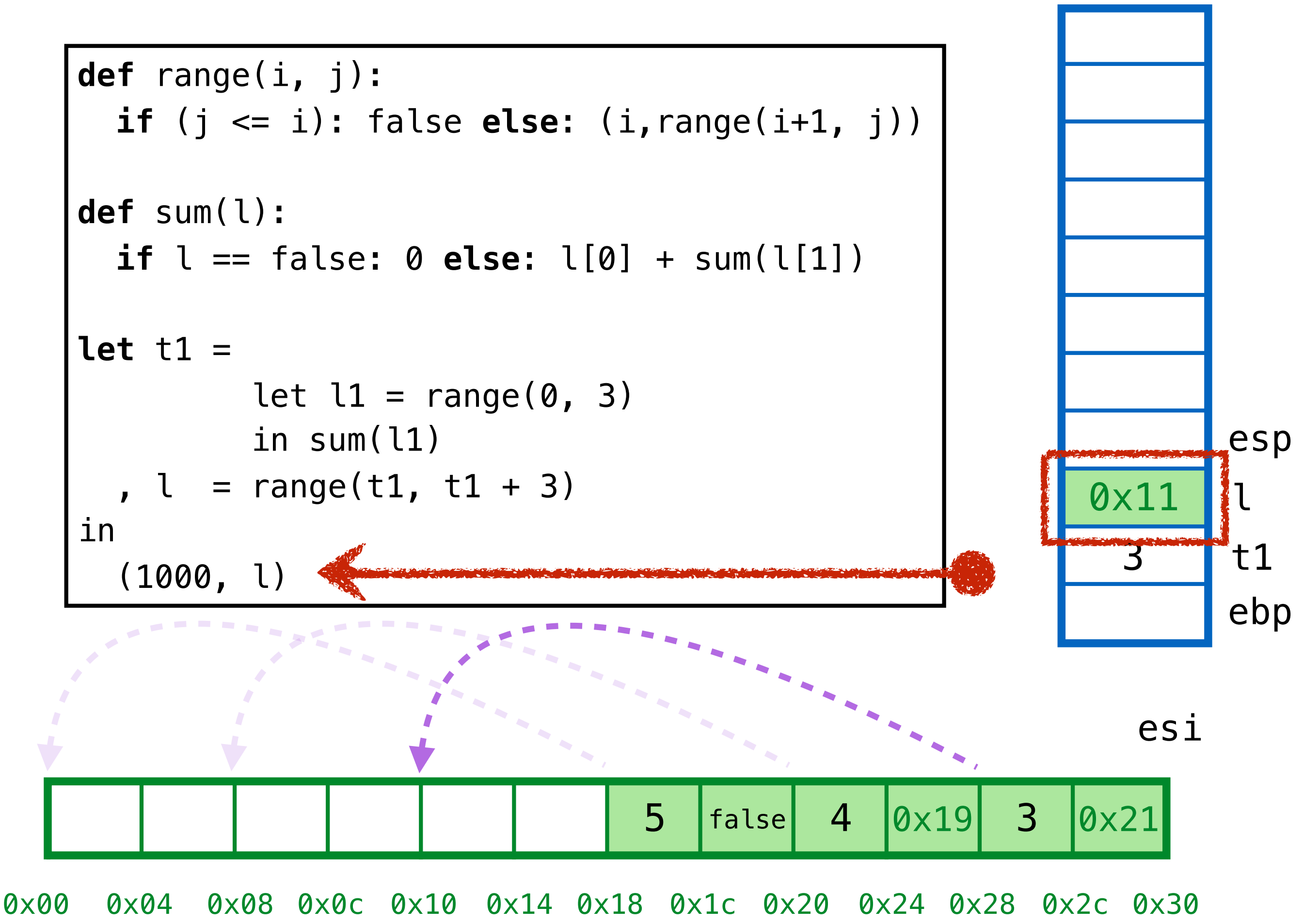
ex4: recursive data



```
def range(i, j):
  if (j <= i): false else: (i,range(i+1, j))

def sum(l):
  if l == false: 0 else: l[0] + sum(l[1])

let t1 =
         let l1 = range(0, 3)
         in sum(l1)
  , l  = range(t1, t1 + 3)
in
  (1000, l)
```

3. REDIRECT addrs on stack

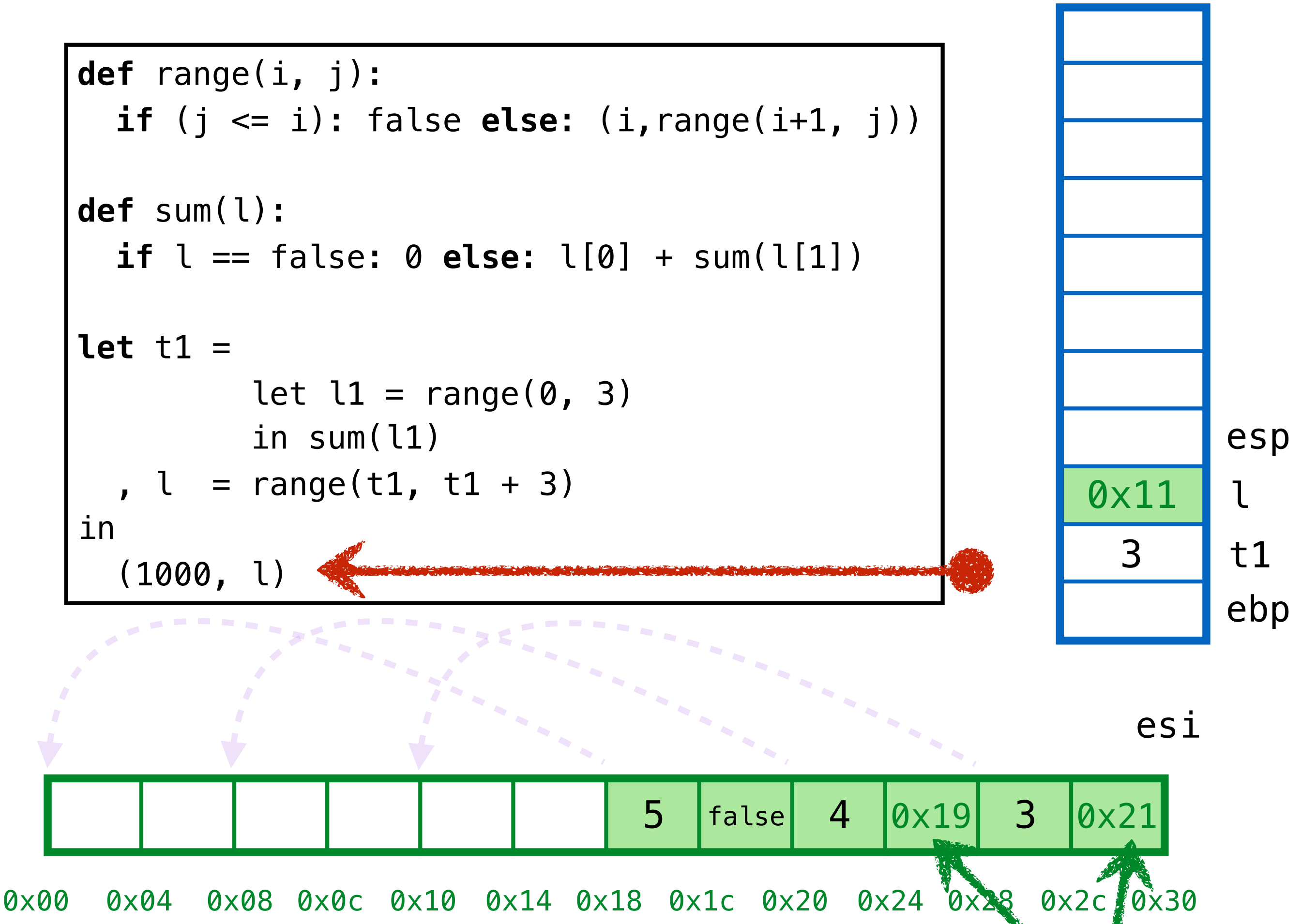
ex4: recursive data



```
def range(i, j):
  if (j <= i): false else: (i,range(i+1, j))

def sum(l):
  if l == false: 0 else: l[0] + sum(l[1])

let t1 =
         let l1 = range(0, 3)
         in sum(l1)
  , l  = range(t1, t1 + 3)
in
  (1000, l)
```

3. REDIRECT addrs on stack

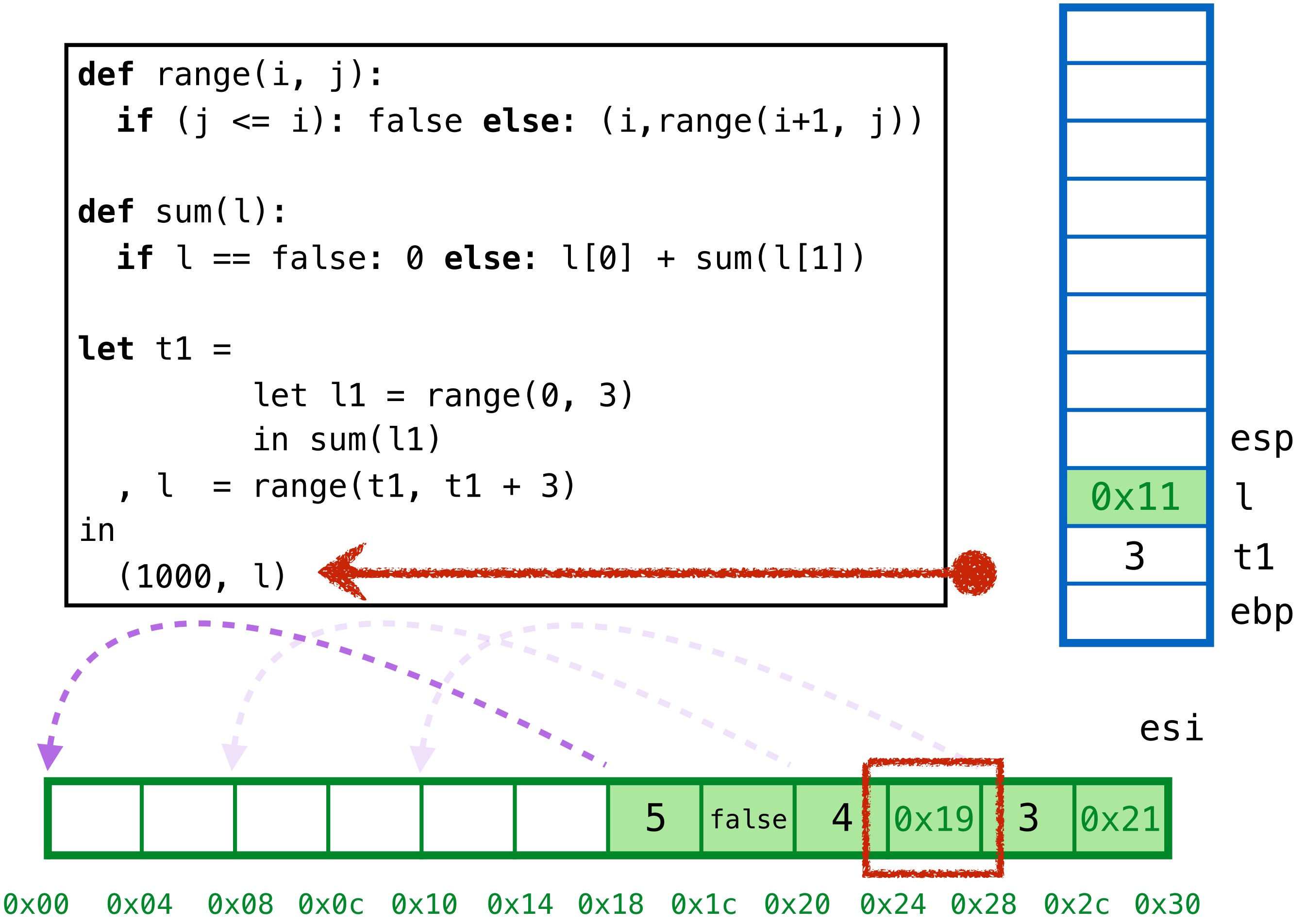
ex4: recursive data



```
def range(i, j):
  if (j <= i): false else: (i,range(i+1, j))

def sum(l):
  if l == false: 0 else: l[0] + sum(l[1])

let t1 =
         let l1 = range(0, 3)
         in sum(l1)
  , l  = range(t1, t1 + 3)
in
  (1000, l)
```

3. REDIRECT addrs on stack and heap!

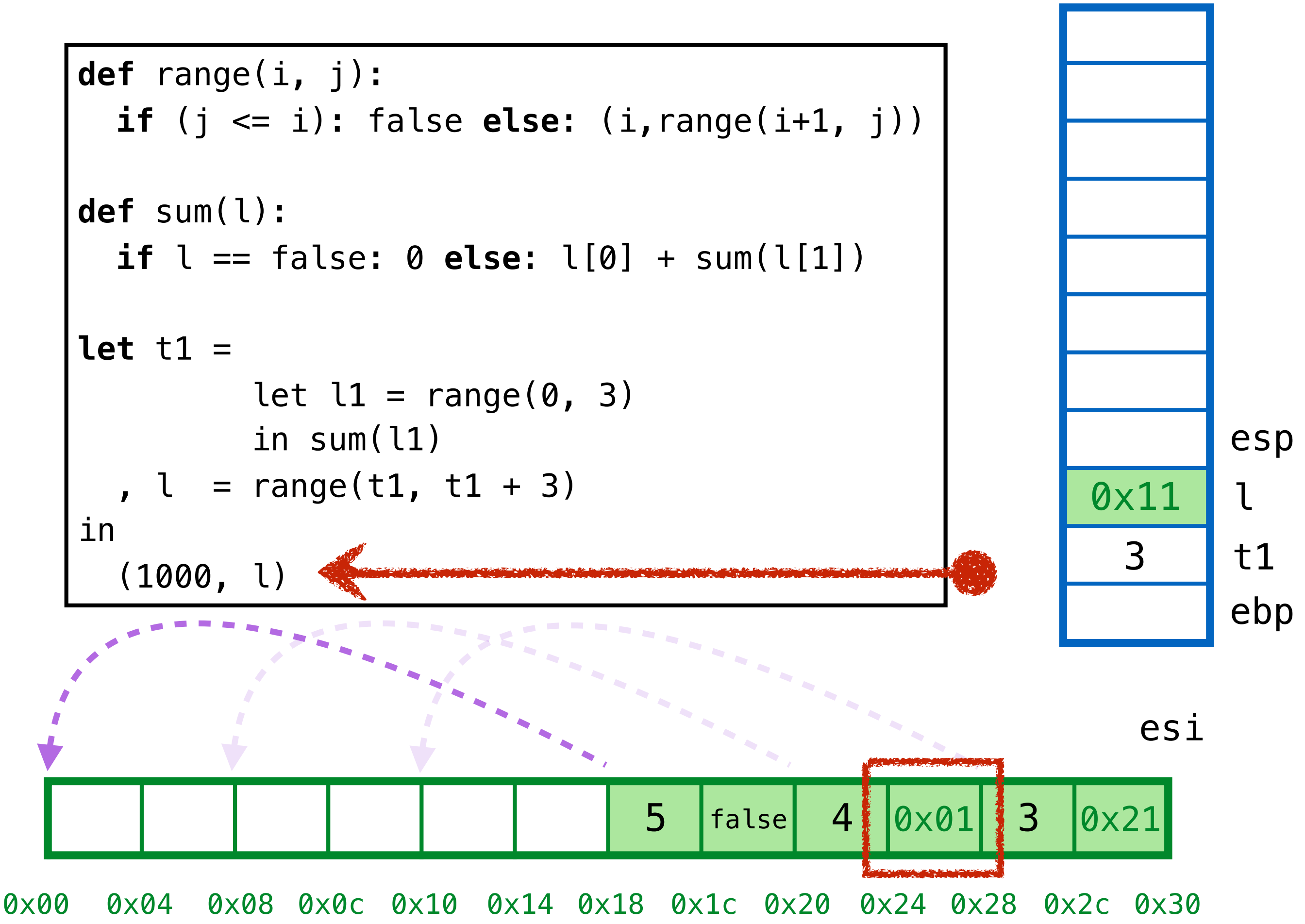
ex4: recursive data



```
def range(i, j):
  if (j <= i): false else: (i,range(i+1, j))

def sum(l):
  if l == false: 0 else: l[0] + sum(l[1])

let t1 =
         let l1 = range(0, 3)
         in sum(l1)
  , l  = range(t1, t1 + 3)
in
  (1000, l)
```

3. REDIRECT addrs on stack and heap!

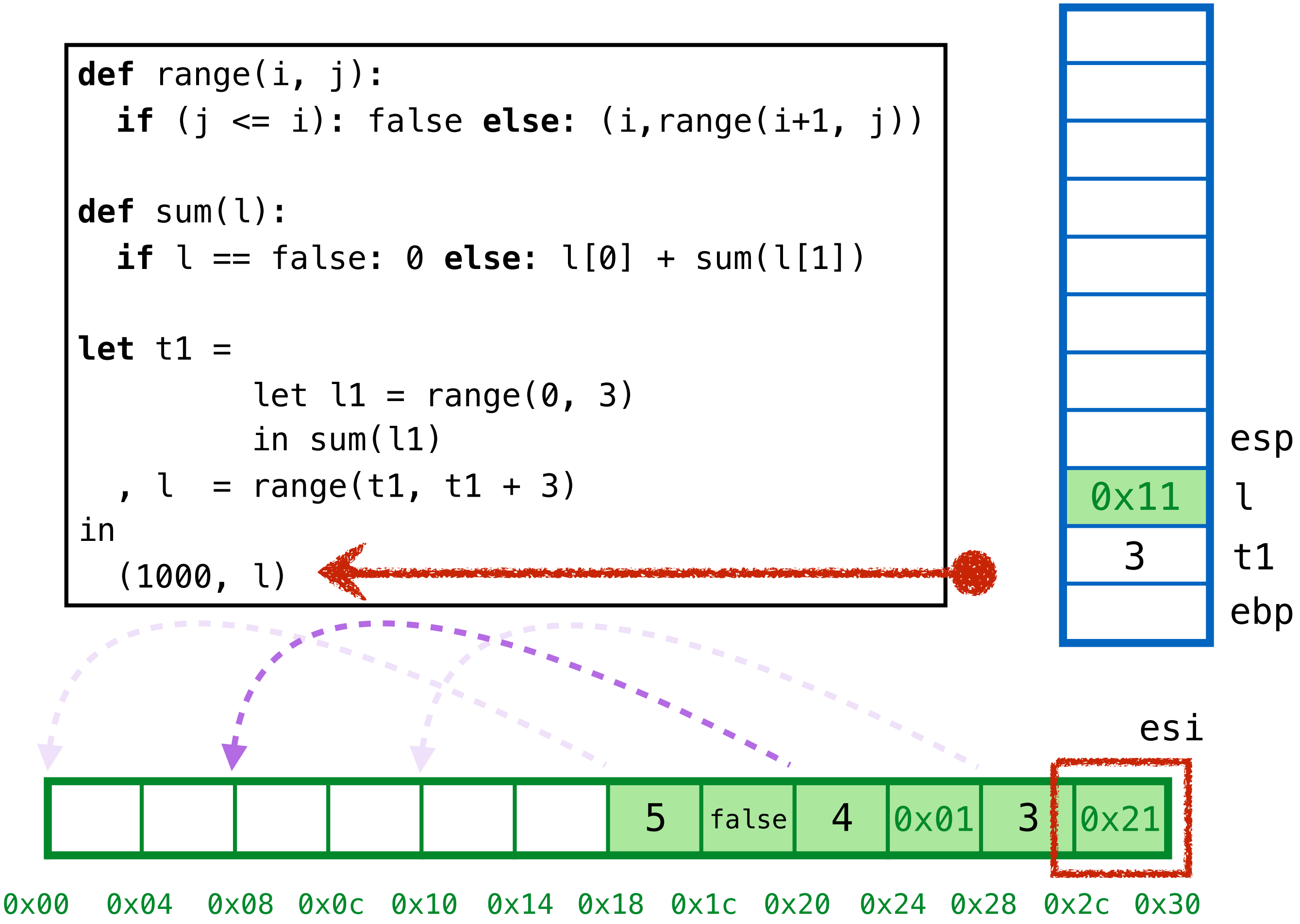
ex4: recursive data



```
def range(i, j):
  if (j <= i): false else: (i,range(i+1, j))

def sum(l):
  if l == false: 0 else: l[0] + sum(l[1])

let t1 =
         let l1 = range(0, 3)
         in sum(l1)
  , l  = range(t1, t1 + 3)
in
  (1000, l)
```

3. REDIRECT addrs on stack and heap!

ex4: recursive data

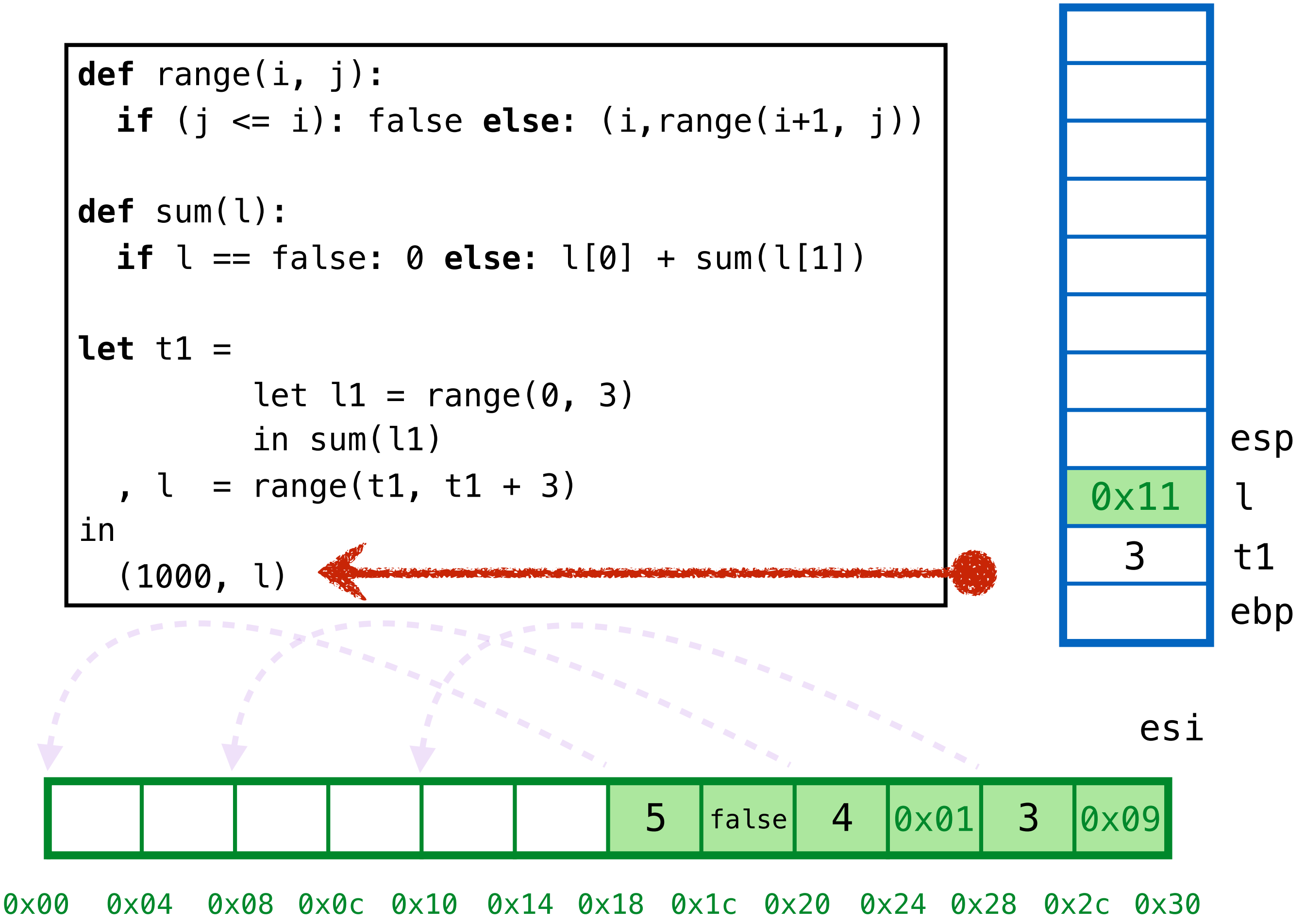


```
def range(i, j):
  if (j <= i): false else: (i,range(i+1, j))

def sum(l):
  if l == false: 0 else: l[0] + sum(l[1])

let t1 =
         let l1 = range(0, 3)
         in sum(l1)
  , l  = range(t1, t1 + 3)
in
  (1000, l)
```

3. REDIRECT addrs on stack and heap!

ex4: recursive data

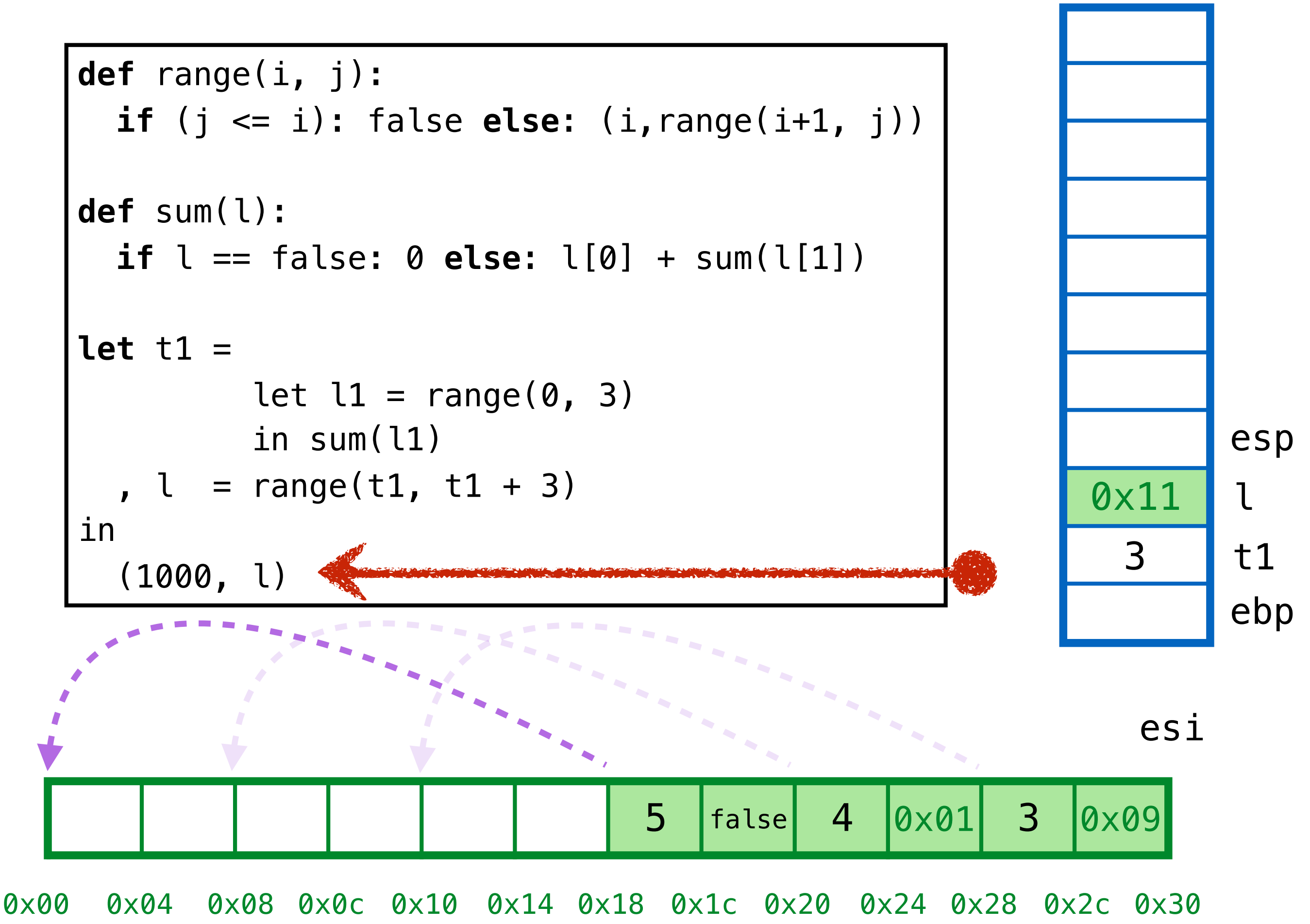
```
def range(i, j):
  if (j <= i): false else: (i,range(i+1, j))

def sum(l):
  if l == false: 0 else: l[0] + sum(l[1])

let t1 =
         let l1 = range(0, 3)
         in sum(l1)
  , l  = range(t1, t1 + 3)
in
  (1000, l)
```



4. COMPACT cells on heap

Copy cell to forward addr!

ex4: recursive data

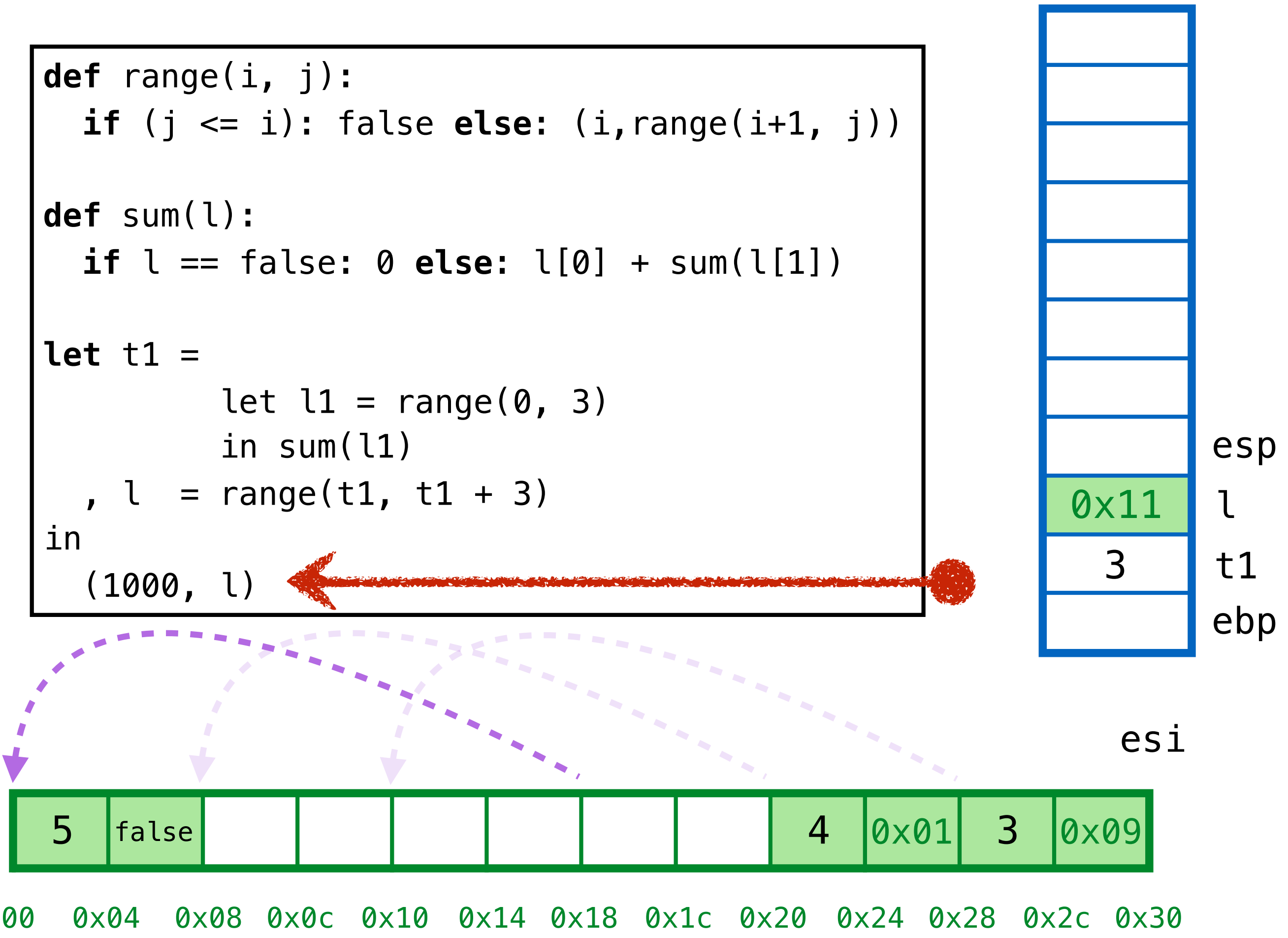


```
def range(i, j):
  if (j <= i): false else: (i,range(i+1, j))

def sum(l):
  if l == false: 0 else: l[0] + sum(l[1])

let t1 =
         let l1 = range(0, 3)
         in sum(l1)
  , l  = range(t1, t1 + 3)
in
  (1000, l)
```

4. COMPACT cells on heap

Copy cell to forward addr!

ex4: recursive data

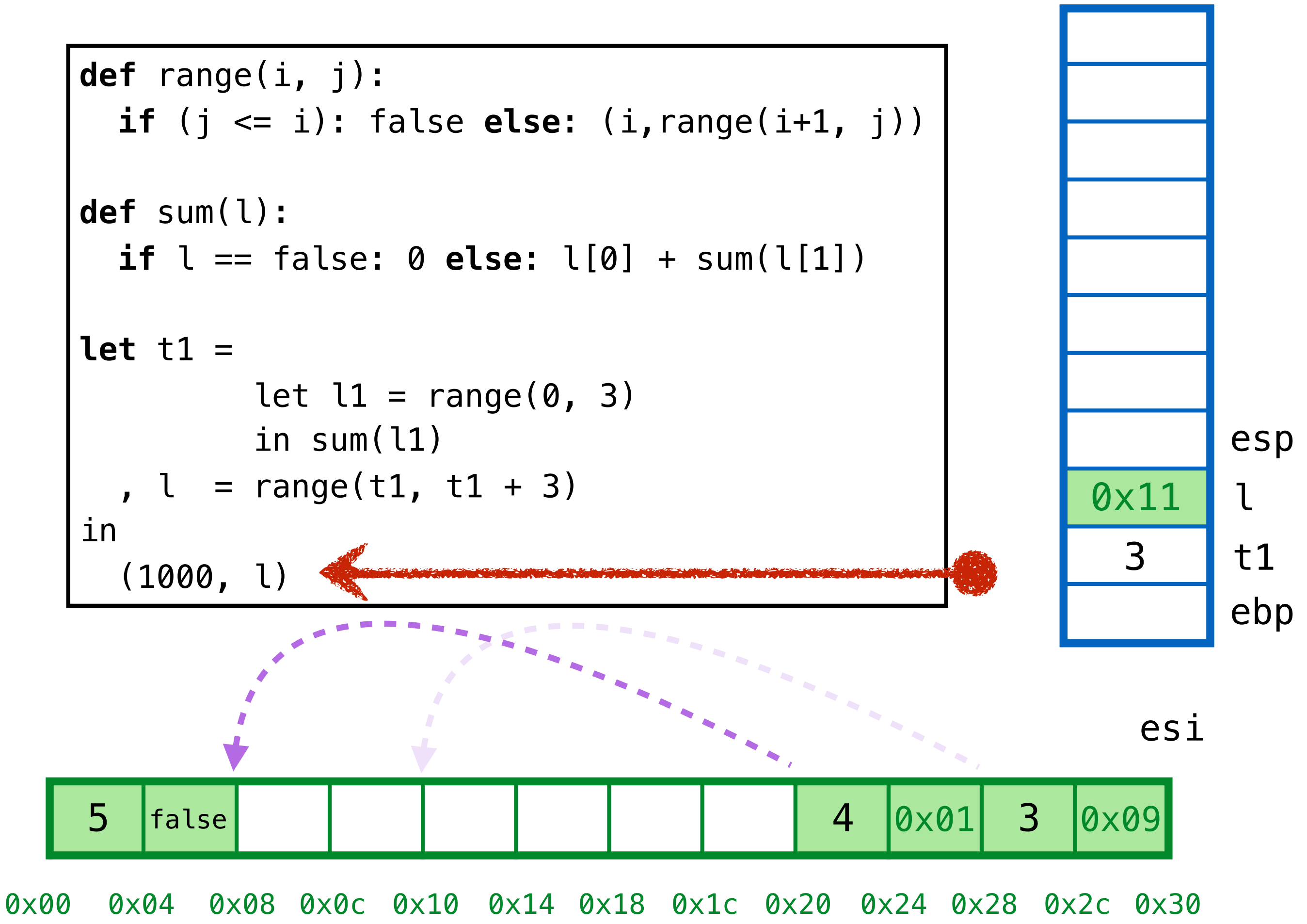


```
def range(i, j):
  if (j <= i): false else: (i,range(i+1, j))

def sum(l):
  if l == false: 0 else: l[0] + sum(l[1])

let t1 =
         let l1 = range(0, 3)
         in sum(l1)
  , l  = range(t1, t1 + 3)
in
  (1000, l)
```

4. COMPACT cells on heap

Copy cell to forward addr!

ex4: recursive data

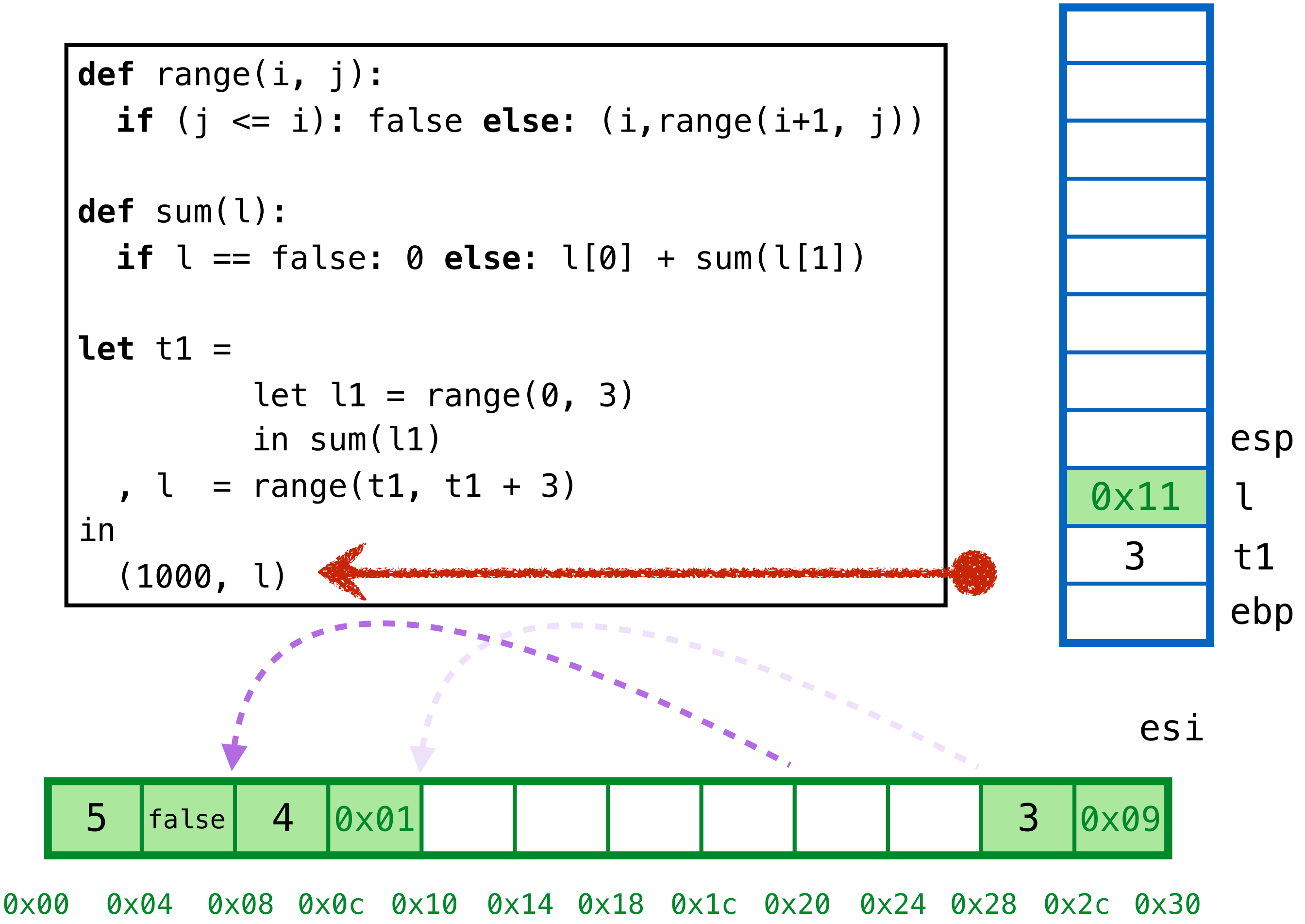


```
def range(i, j):
  if (j <= i): false else: (i,range(i+1, j))

def sum(l):
  if l == false: 0 else: l[0] + sum(l[1])

let t1 =
         let l1 = range(0, 3)
         in sum(l1)
  , l  = range(t1, t1 + 3)
in
  (1000, l)
```

4. COMPACT cells on heap

Copy cell to forward addr!

ex4: recursive data

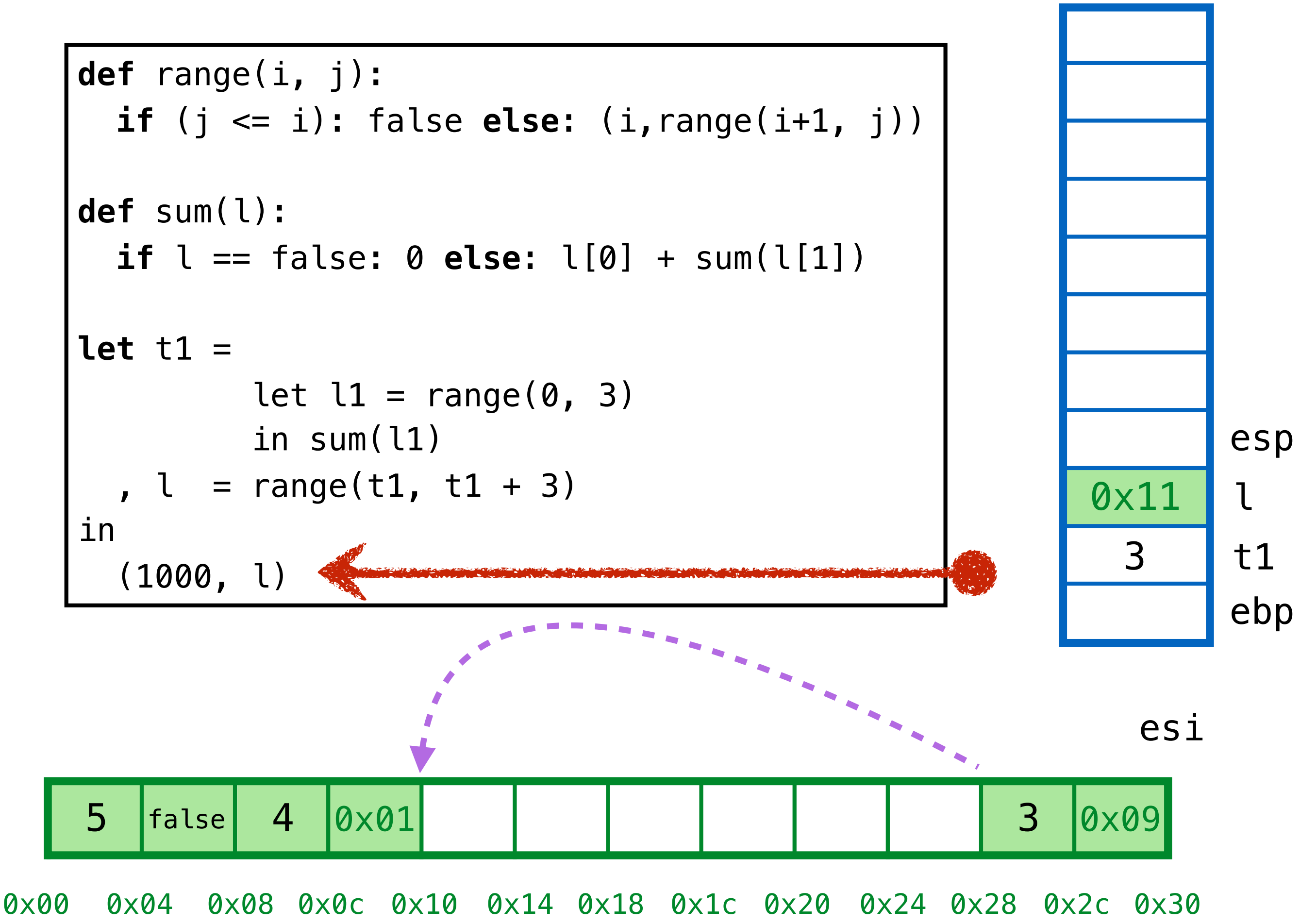


```
def range(i, j):
  if (j <= i): false else: (i,range(i+1, j))

def sum(l):
  if l == false: 0 else: l[0] + sum(l[1])

let t1 =
         let l1 = range(0, 3)
         in sum(l1)
  , l  = range(t1, t1 + 3)
in
  (1000, l)
```

4. COMPACT cells on heap

Copy cell to forward addr!

ex4: recursive data

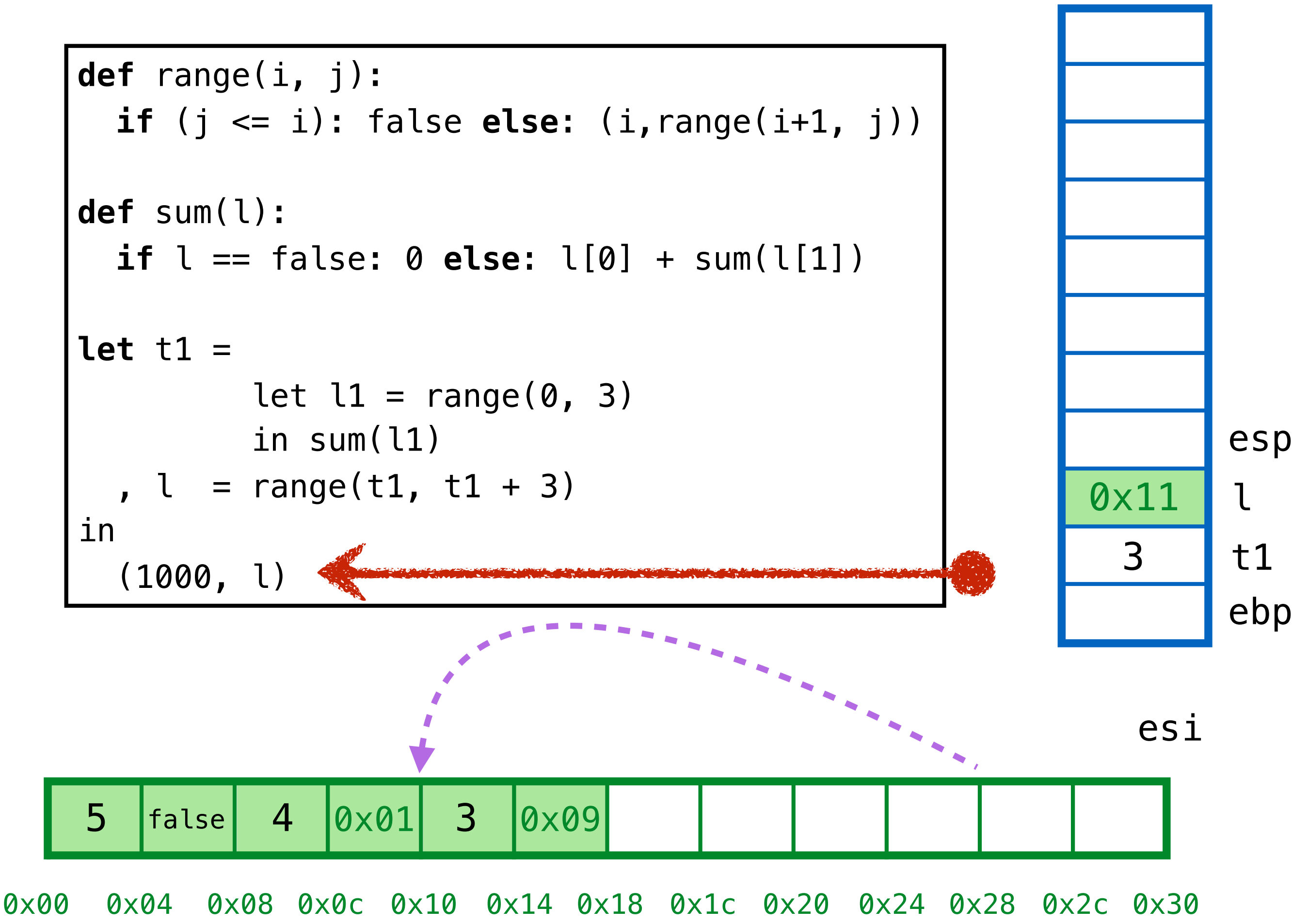


```
def range(i, j):
  if (j <= i): false else: (i,range(i+1, j))

def sum(l):
  if l == false: 0 else: l[0] + sum(l[1])

let t1 =
          let l1 = range(0, 3)
          in sum(l1)
  , l  = range(t1, t1 + 3)
in
  (1000, l)
```

4. COMPACT cells on heap

Copy cell to forward addr!

ex4: recursive data

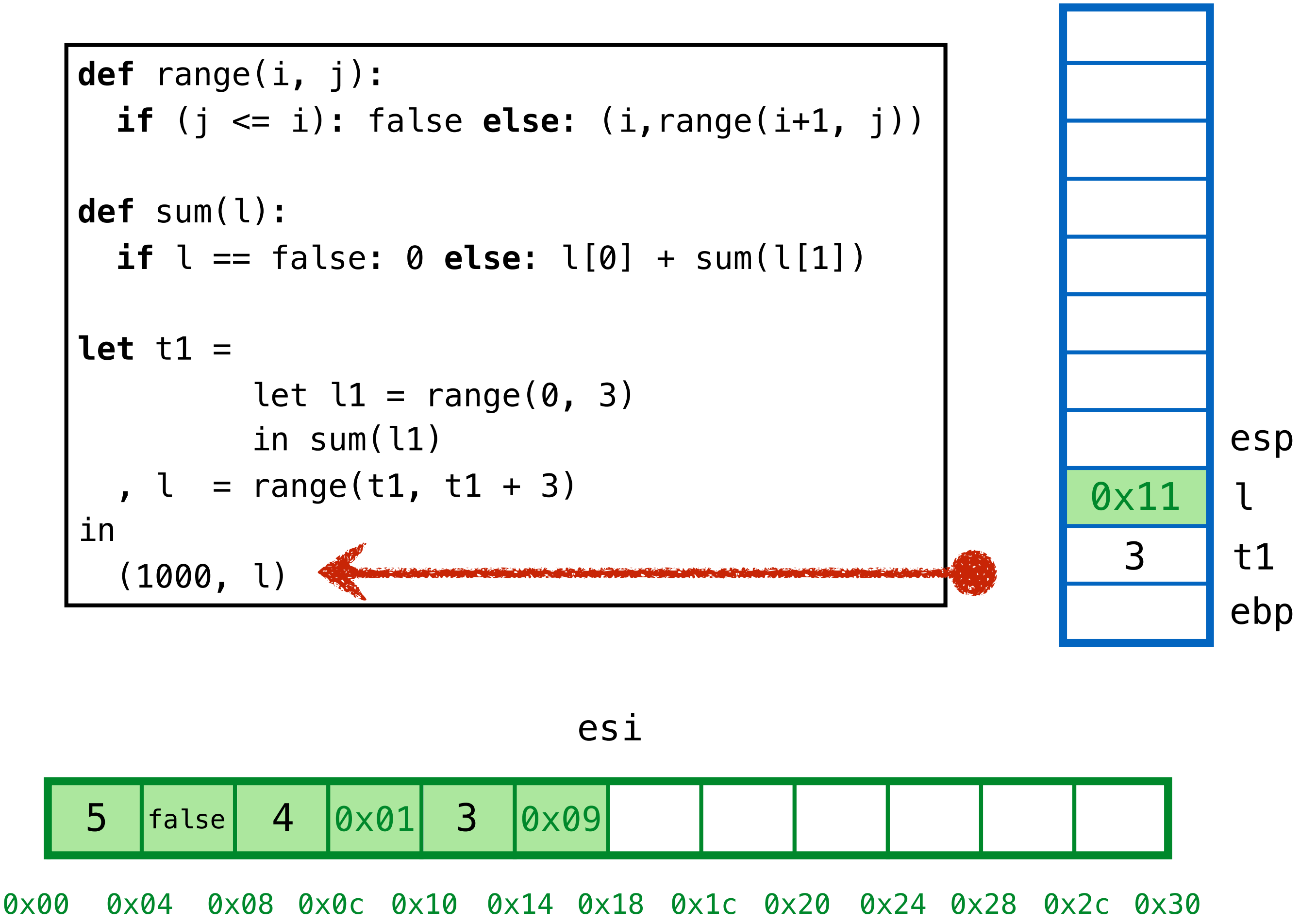


```
def range(i, j):
  if (j <= i): false else: (i,range(i+1, j))

def sum(l):
  if l == false: 0 else: l[0] + sum(l[1])

let t1 =
         let l1 = range(0, 3)
         in sum(l1)
  , l  = range(t1, t1 + 3)
in
  (1000, l)
```

4. COMPACT cells on heap

Copy cell to forward addr!

ex4: recursive data



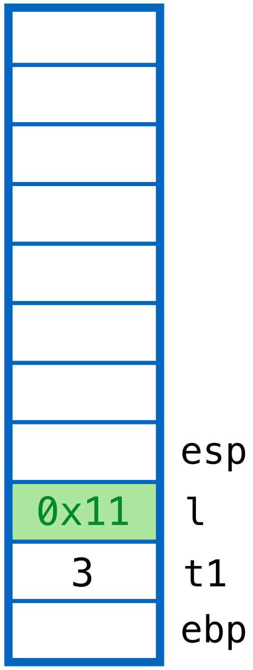
```
def range(i, j):
  if (j <= i): false else: (i,range(i+1, j))

def sum(l):
  if l == false: 0 else: l[0] + sum(l[1])

let t1 =
         let l1 = range(0, 3)
         in sum(l1)
  , l  = range(t1, t1 + 3)
in
  (1000, l)
```

GC Complete!

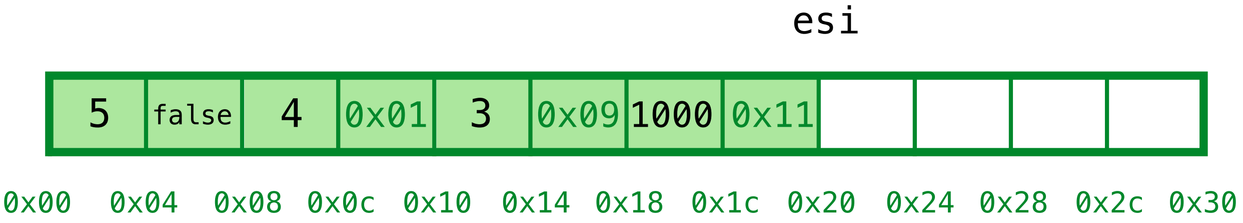
Have space for `(1000, l)`

ex4: recursive data

```
def range(i, j):
  if (j <= i): false else: (i,range(i+1, j))

def sum(l):
  if l == false: 0 else: l[0] + sum(l[1])

let t1 =
         let l1 = range(0, 3)
         in sum(l1)
  , l  = range(t1, t1 + 3)
in
  (1000, l)
```



	esp
0x11	l
3	t1
	ebp



0x00	0x04	0x08	0x0c	0x10	0x14	0x18	0x1c	0x20	0x24	0x28	0x2c	0x30
5	false	4	0x01	3	0x09	1000	0x11	esi				

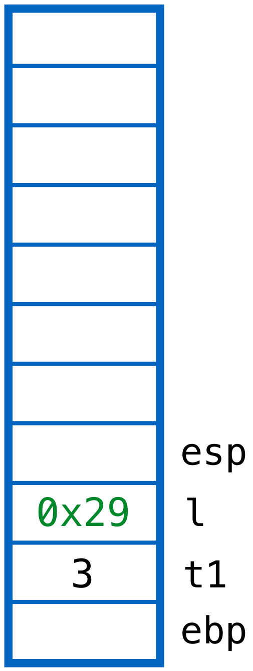
GC Complete!

Have space for `(1000, l)`

ex4: recursive data

QUIZ: What should `print(0x11)` show?

```
(A) (0, (1, (2, false)))
(B) (3, (4, (5, false)))
(C) (0, (1, (2, (3, (4, (5, false))))))
(D) (3, (4, (5, (0, (1, (2, false))))))
(E) (2, (1, (0, (3, (4, (5, false))))))
```

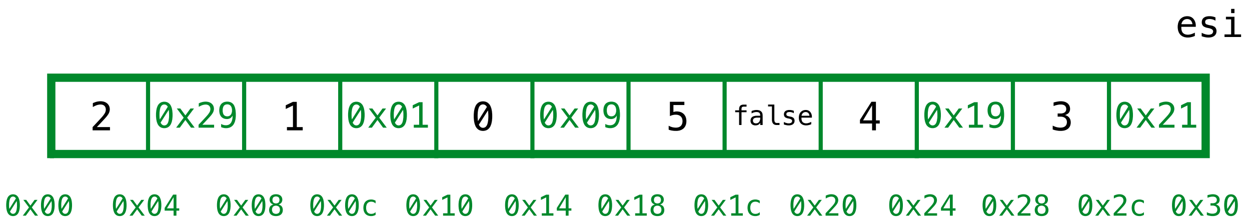


esp

0x29 l

3 t1

ebp



esi

2	0x29	1	0x01	0	0x09	5	false	4	0x19	3	0x21

0x00 0x04 0x08 0x0c 0x10 0x14 0x18 0x1c 0x20 0x24 0x28 0x2c 0x30

ex4: recursive data

QUIZ: Which cells are "live" on the heap?

(A) 0x00

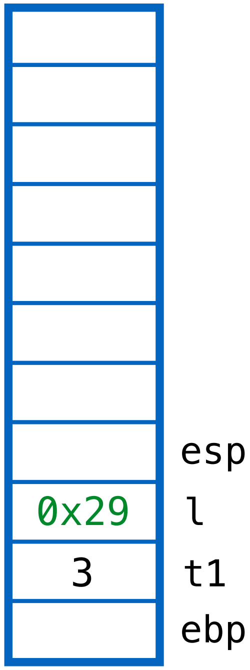
(B) 0x08

(C) 0x10

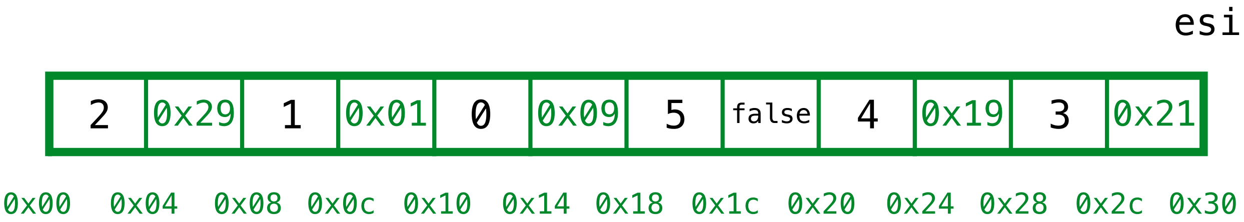
(D) 0x18

(E) 0x20

(F) 0x28



	esp
0x29	l
3	t1
	ebp



esi

2	0x29	1	0x01	0	0x09	5	false	4	0x19	3	0x21	
0x00	0x04	0x08	0x0c	0x10	0x14	0x18	0x1c	0x20	0x24	0x28	0x2c	0x30