Garter

Garbage Collection

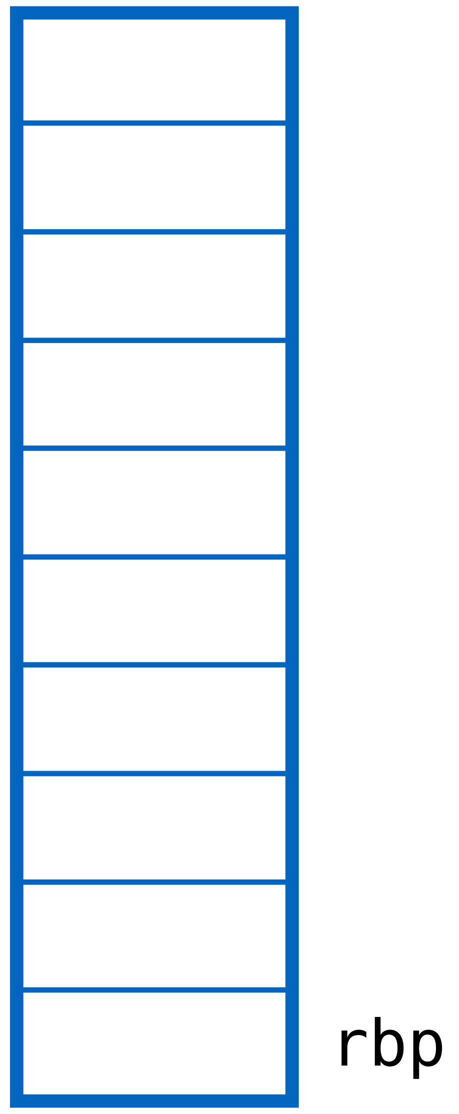
Ranjit Jhala | UCSD

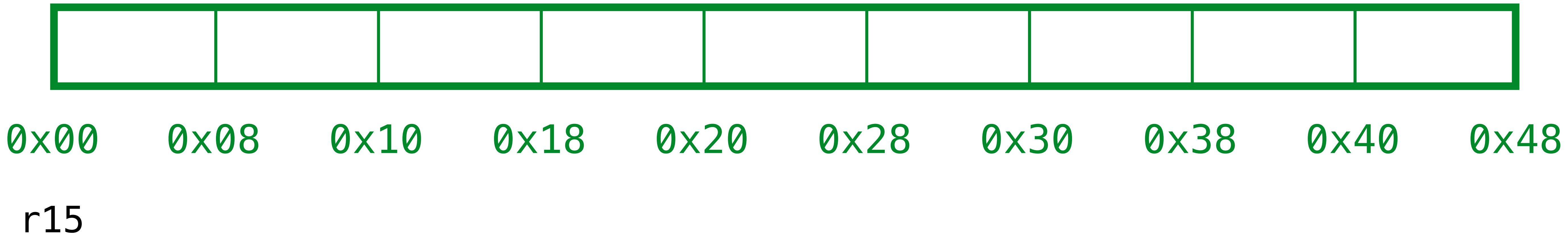
Garter / GC

Example 1

ex1: garbage at end

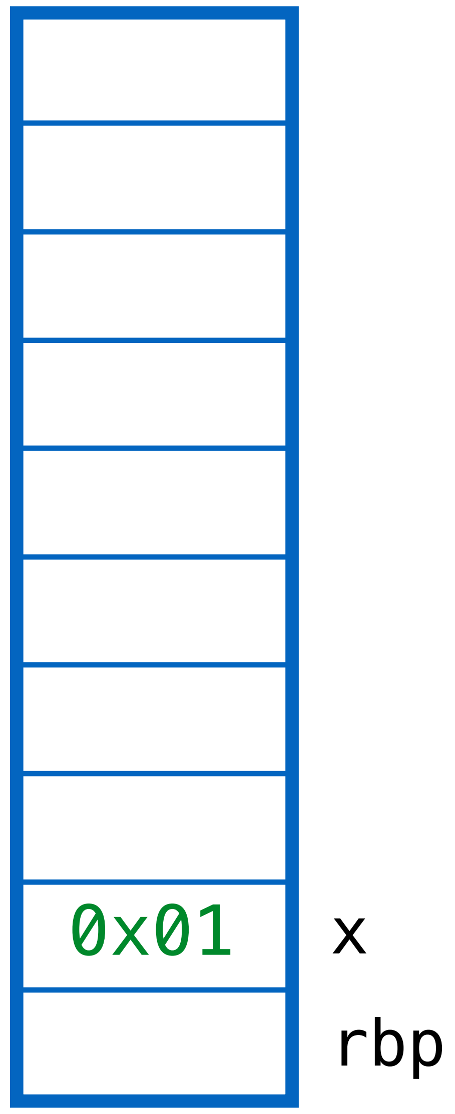
```
let x  = (1, 2)
  , y  = let tmp = (10, 20)
         in tmp[0] + tmp[1]
  , p0 = x[0] + y
  , p1 = x[1] + y
in
  (p0, p1)
```

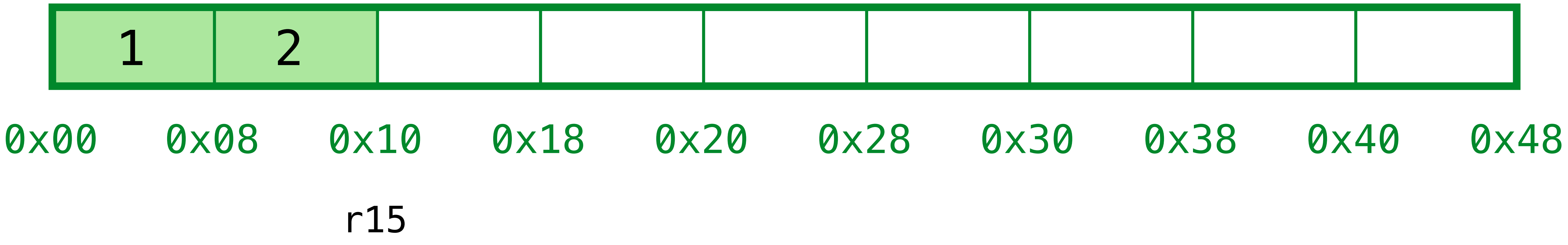




ex1: garbage at end

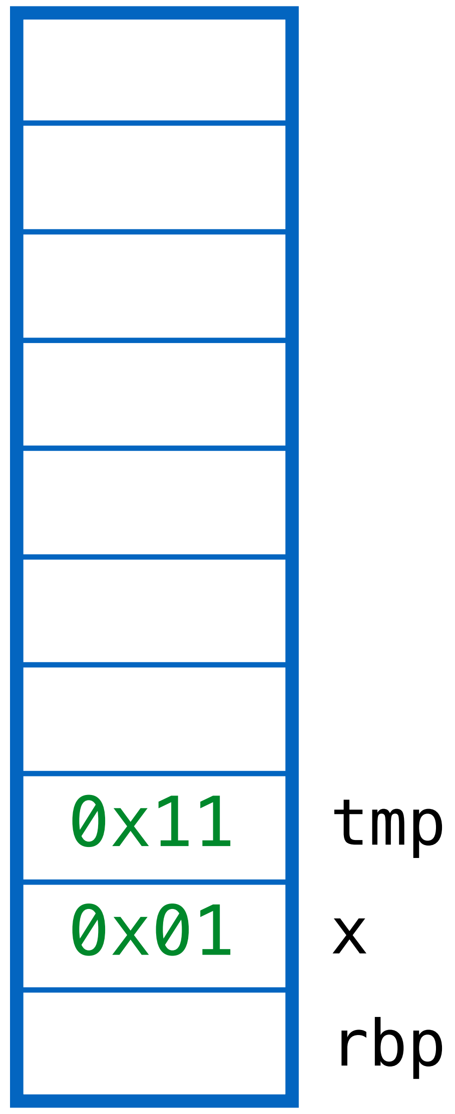
```
let x  = (1, 2)
  , y  = let tmp = (10, 20)
         in tmp[0] + tmp[1]
  , p0 = x[0] + y
  , p1 = x[1] + y
in
  (p0, p1)
```



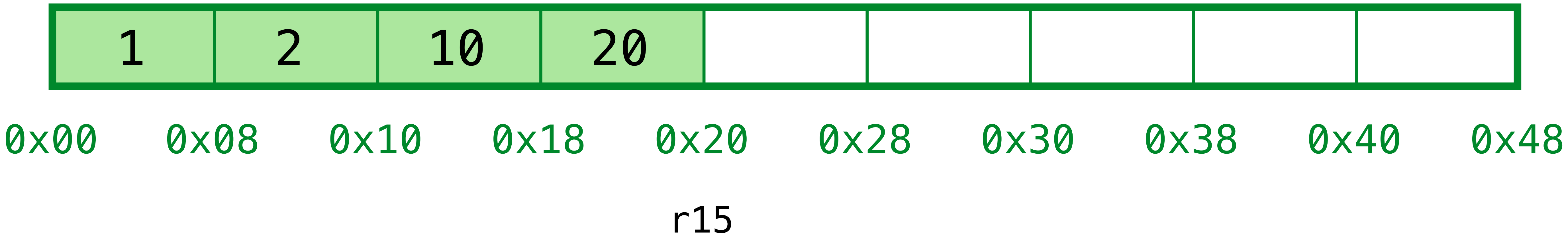


ex1: garbage at end

```
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)
```



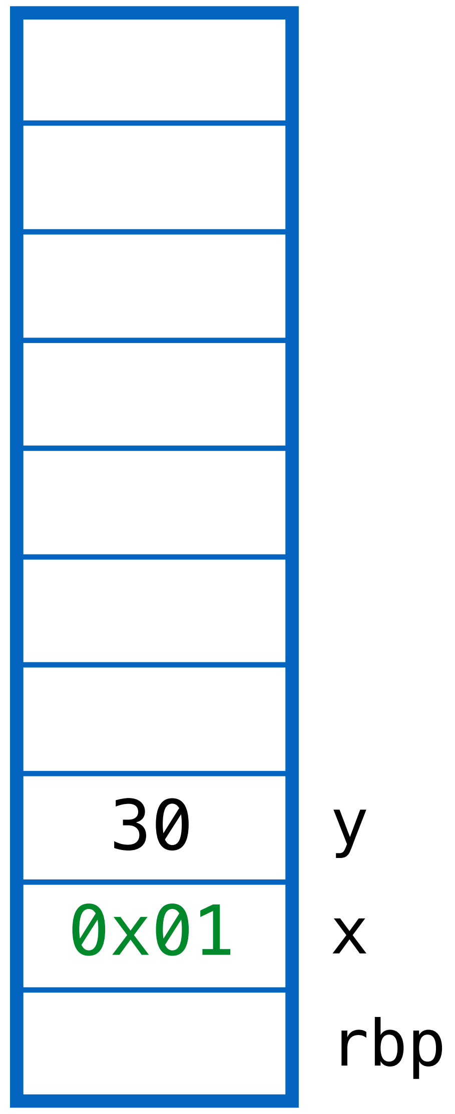
0x11	tmp
0x01	x
	rbp

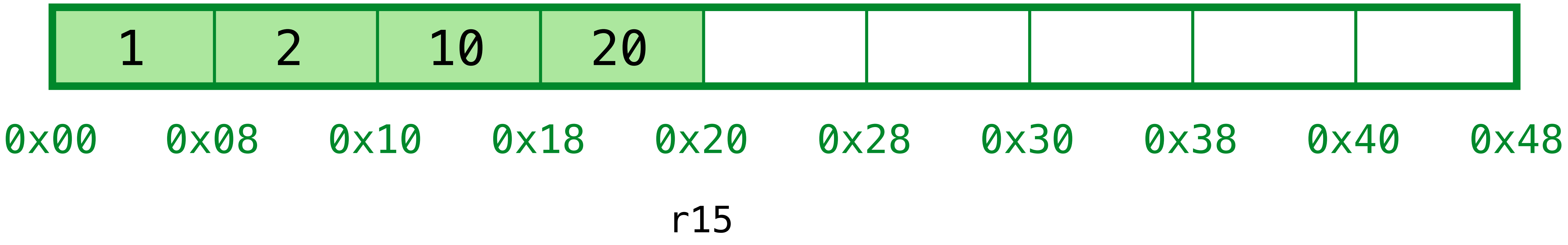


0x00	0x08	0x10	0x18	0x20	0x28	0x30	0x38	0x40
1	2	10	20					

ex1: garbage at end

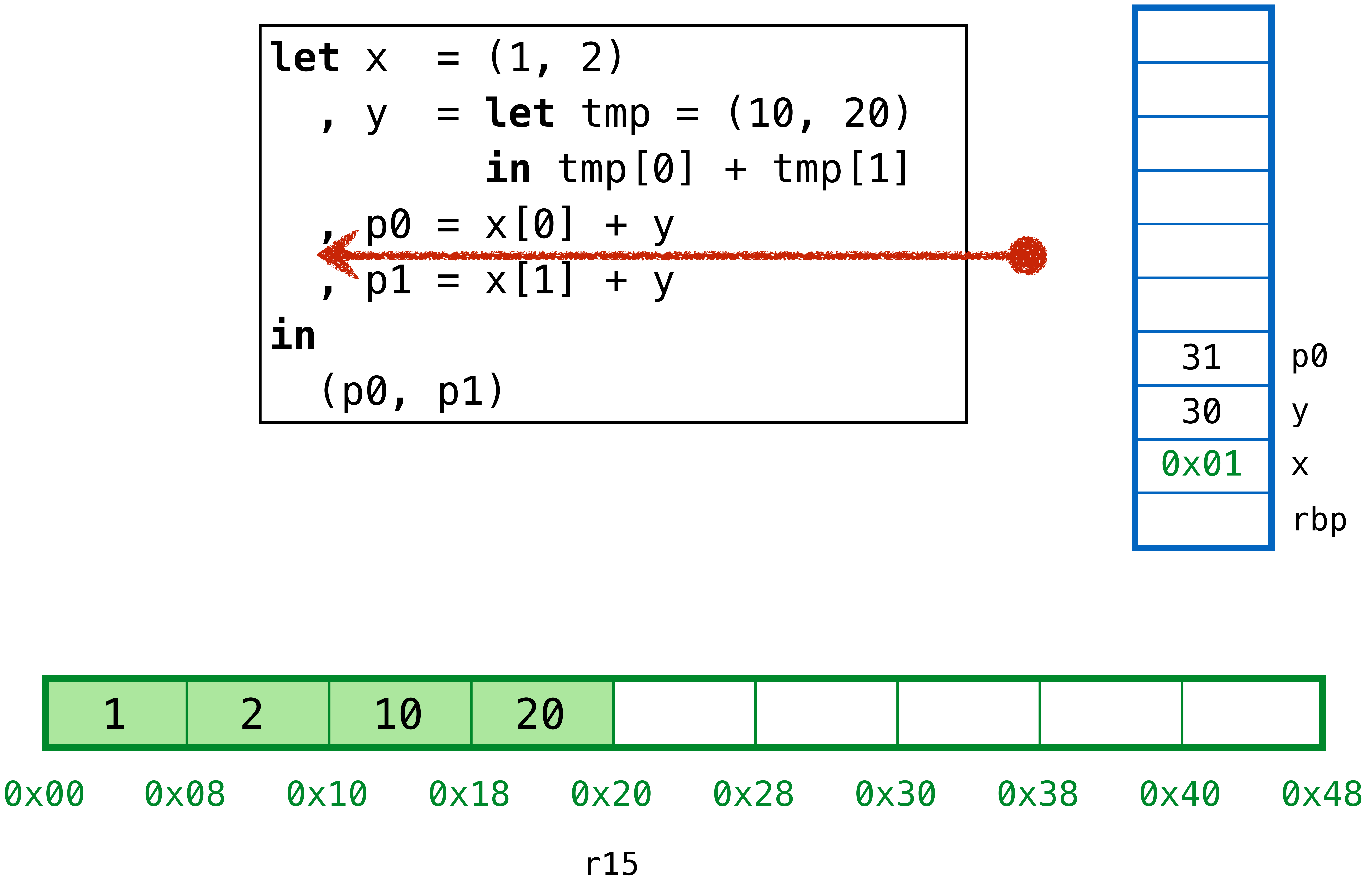
```
let x  = (1, 2)
  , y  = let tmp = (10, 20)
           in tmp[0] + tmp[1]
  , p0 = x[0] + y
  , p1 = x[1] + y
in
  (p0, p1)
```





0x00	0x08	0x10	0x18	0x20	0x28	0x30	0x38	0x40
1	2	10	20					

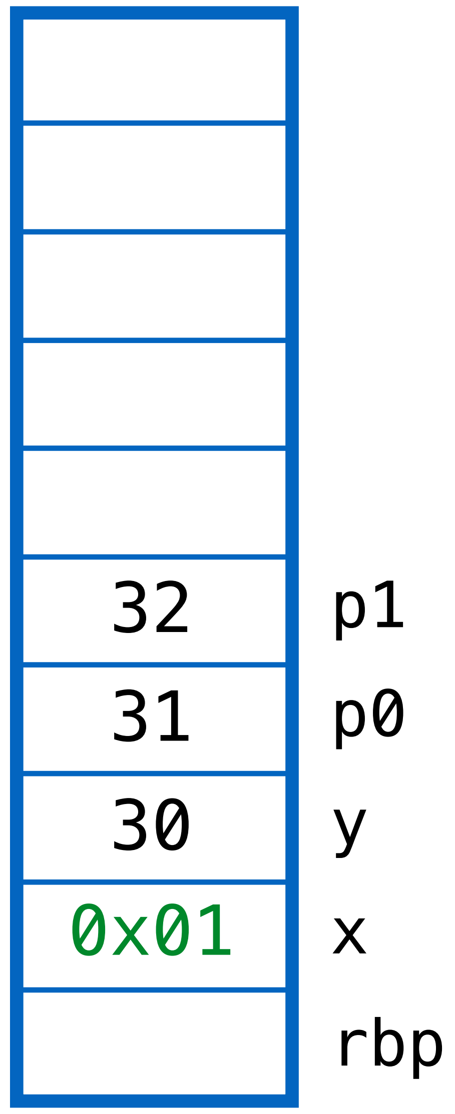
ex1: garbage at end



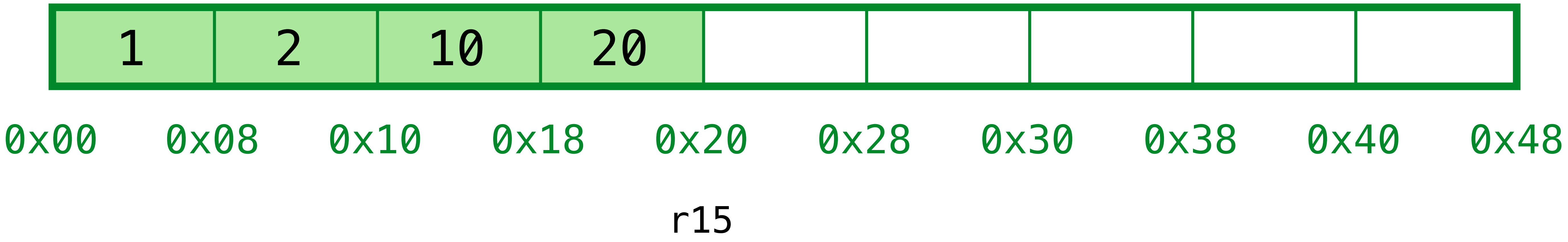
```
let x  = (1, 2)
  , y  = let tmp = (10, 20)
         in tmp[0] + tmp[1]
  , p0 = x[0] + y
  , p1 = x[1] + y
in
  (p0, p1)
```

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)
```



32	p1
31	p0
30	y
0x01	x
	rbp



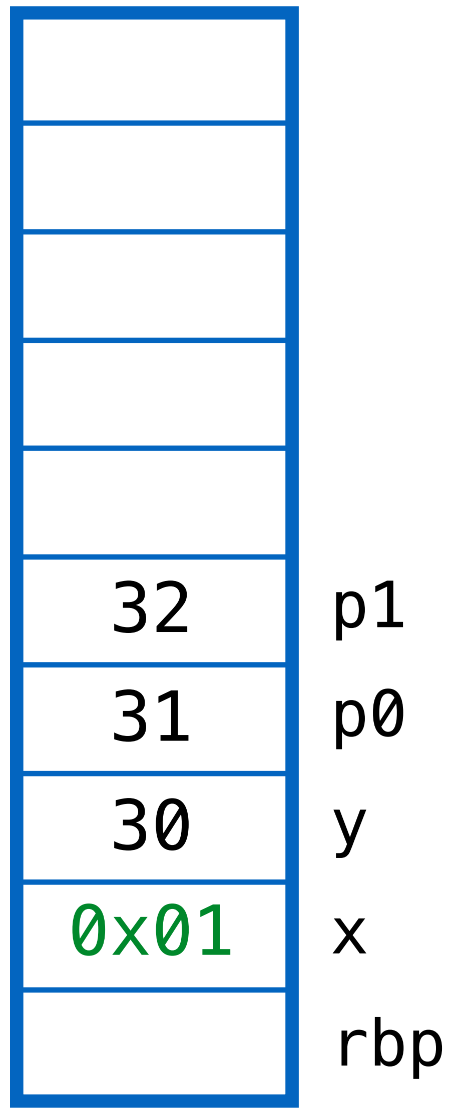
0x00	0x08	0x10	0x18	0x20	0x28	0x30	0x38	0x40
1	2	10	20					

0x48

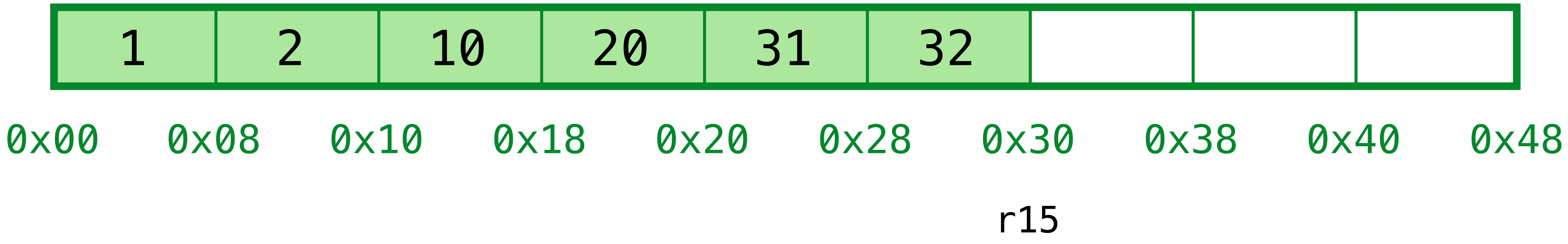
r15

ex1: garbage at end

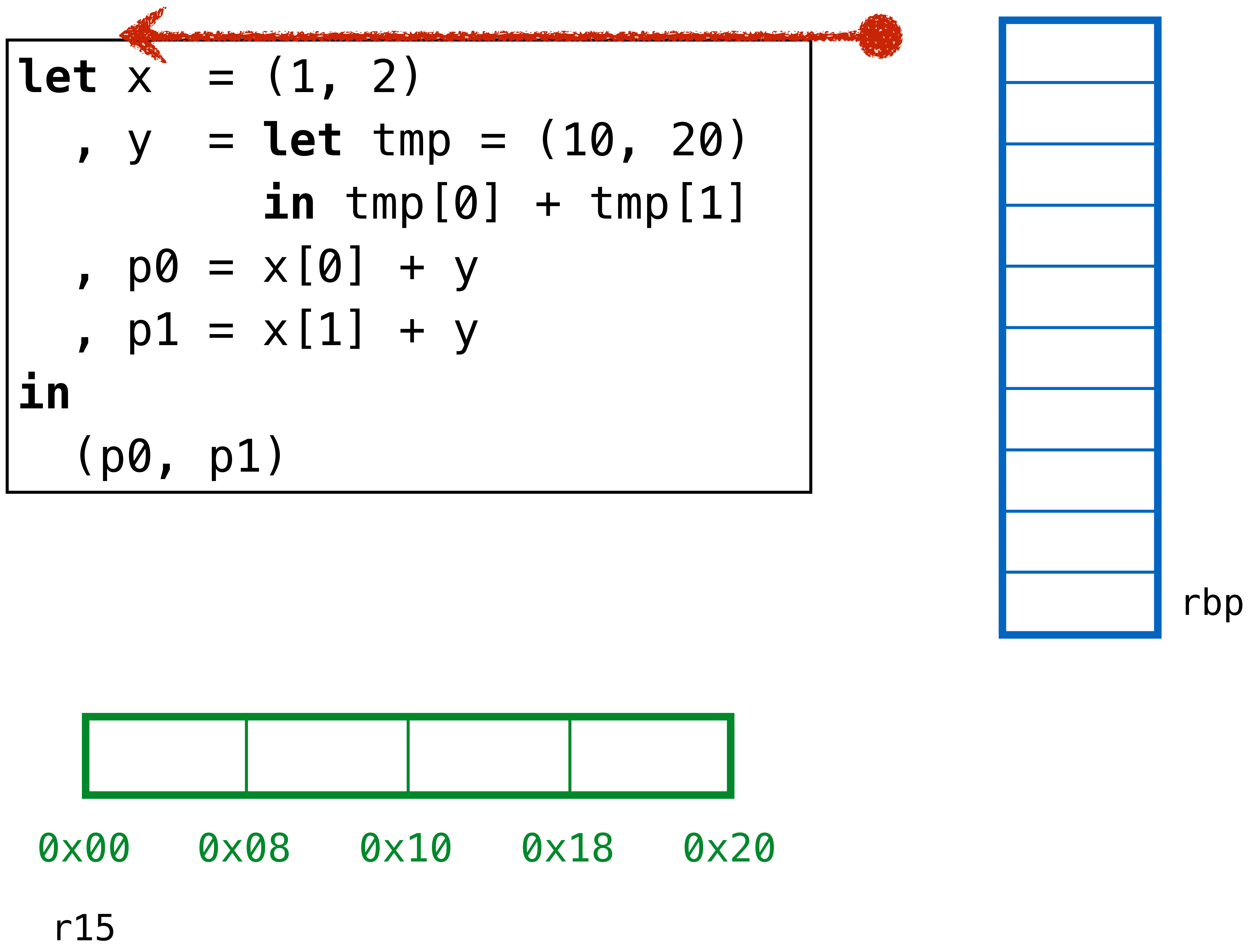
```
let x  = (1, 2)
  , y  = let tmp = (10, 20)
         in tmp[0] + tmp[1]
  , p0 = x[0] + y
  , p1 = x[1] + y
in
  (p0, p1)
```



Result (rax) = 0x21



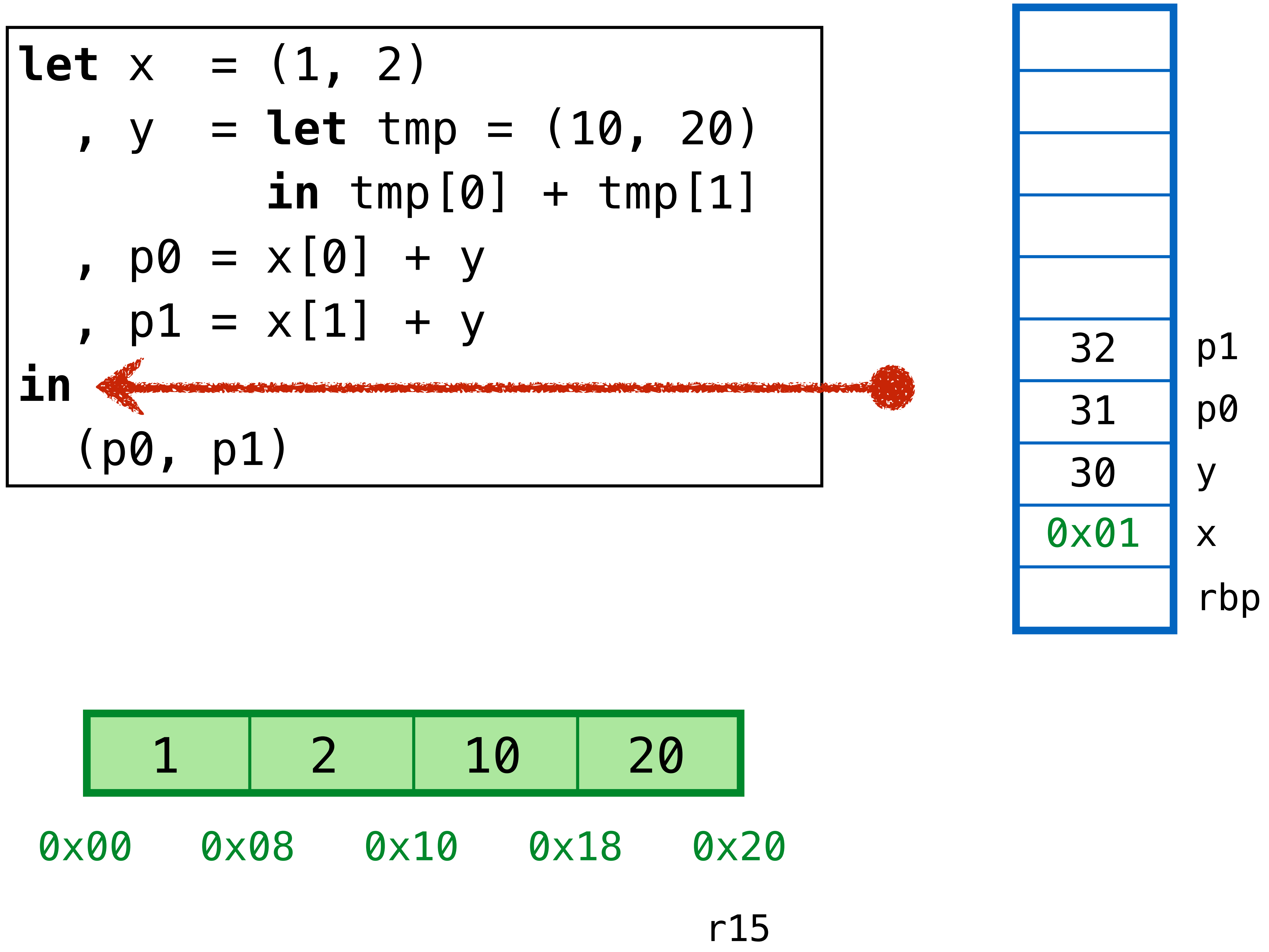
ex1: garbage at end



```
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

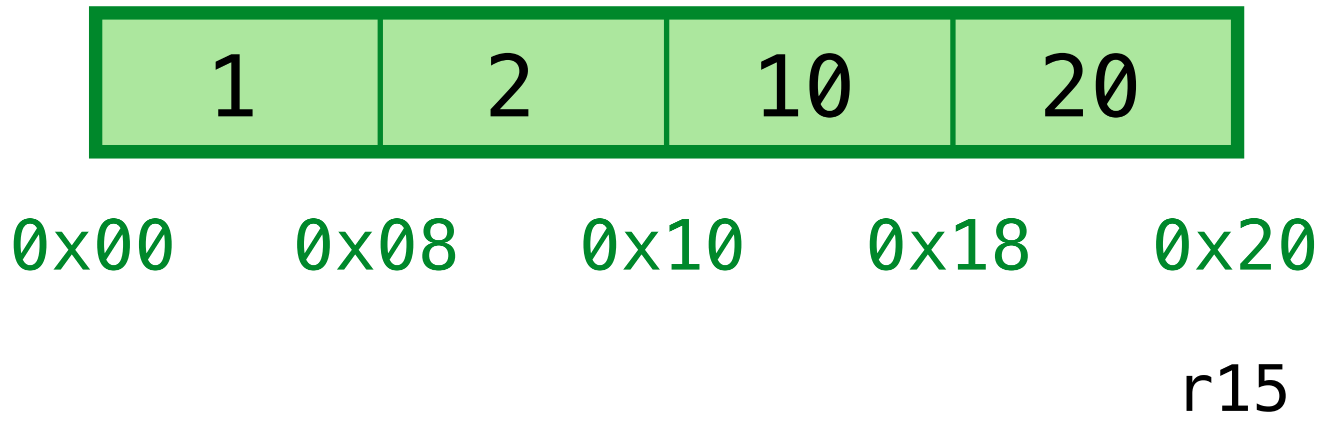


```
let x  = (1, 2)
  , y  = let tmp = (10, 20)
         in tmp[0] + tmp[1]
  , p0 = x[0] + y
  , p1 = x[1] + y
in
  (p0, p1)
```

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)
```

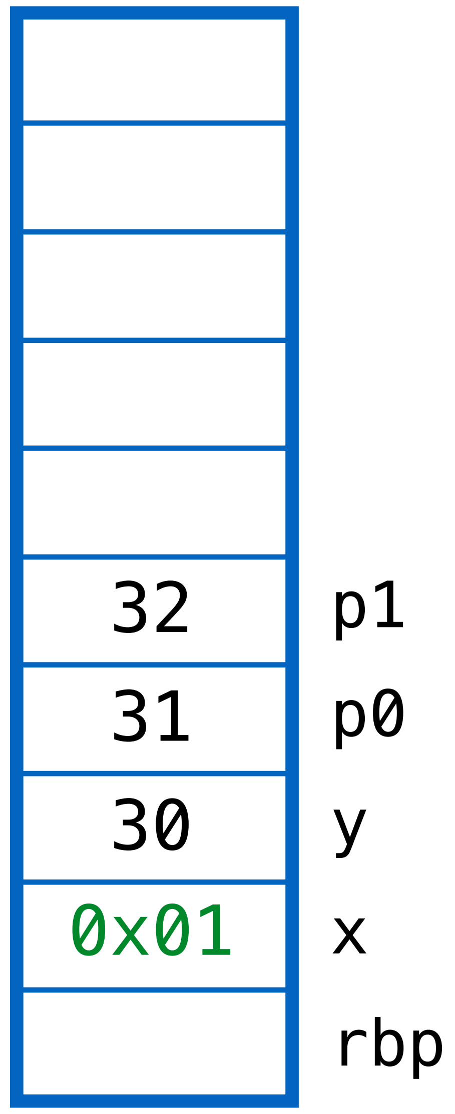
Out of memory!
Can't allocate **`(p0, p1)`**



1 | 2 | 10 | 20

0x00 0x08 0x10 0x18 0x20

r15



32 p1

31 p0

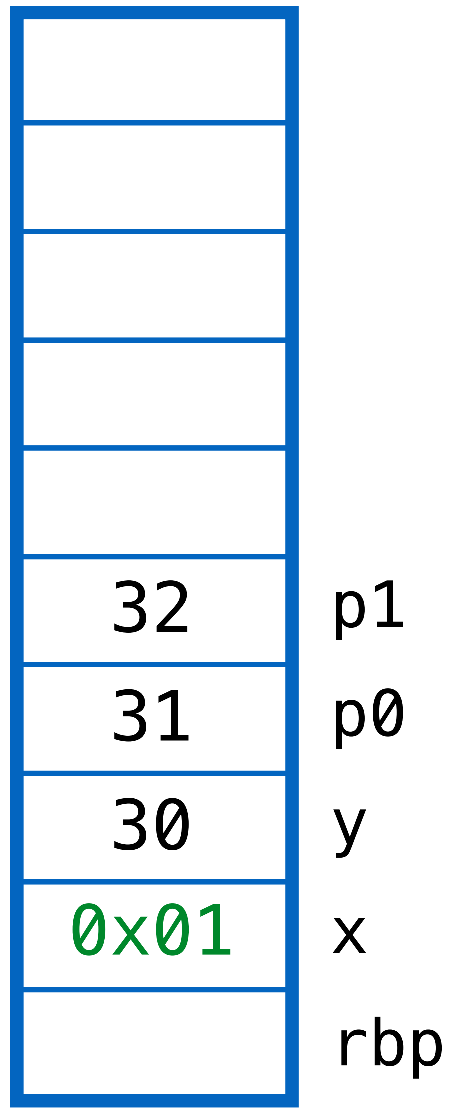
30 y

0x01 x

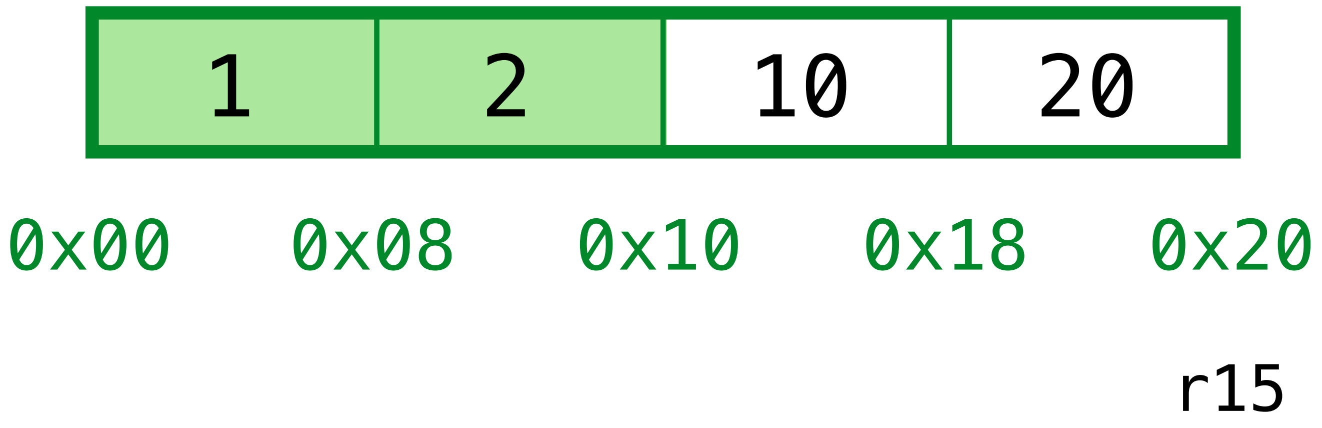
rbp

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)
```

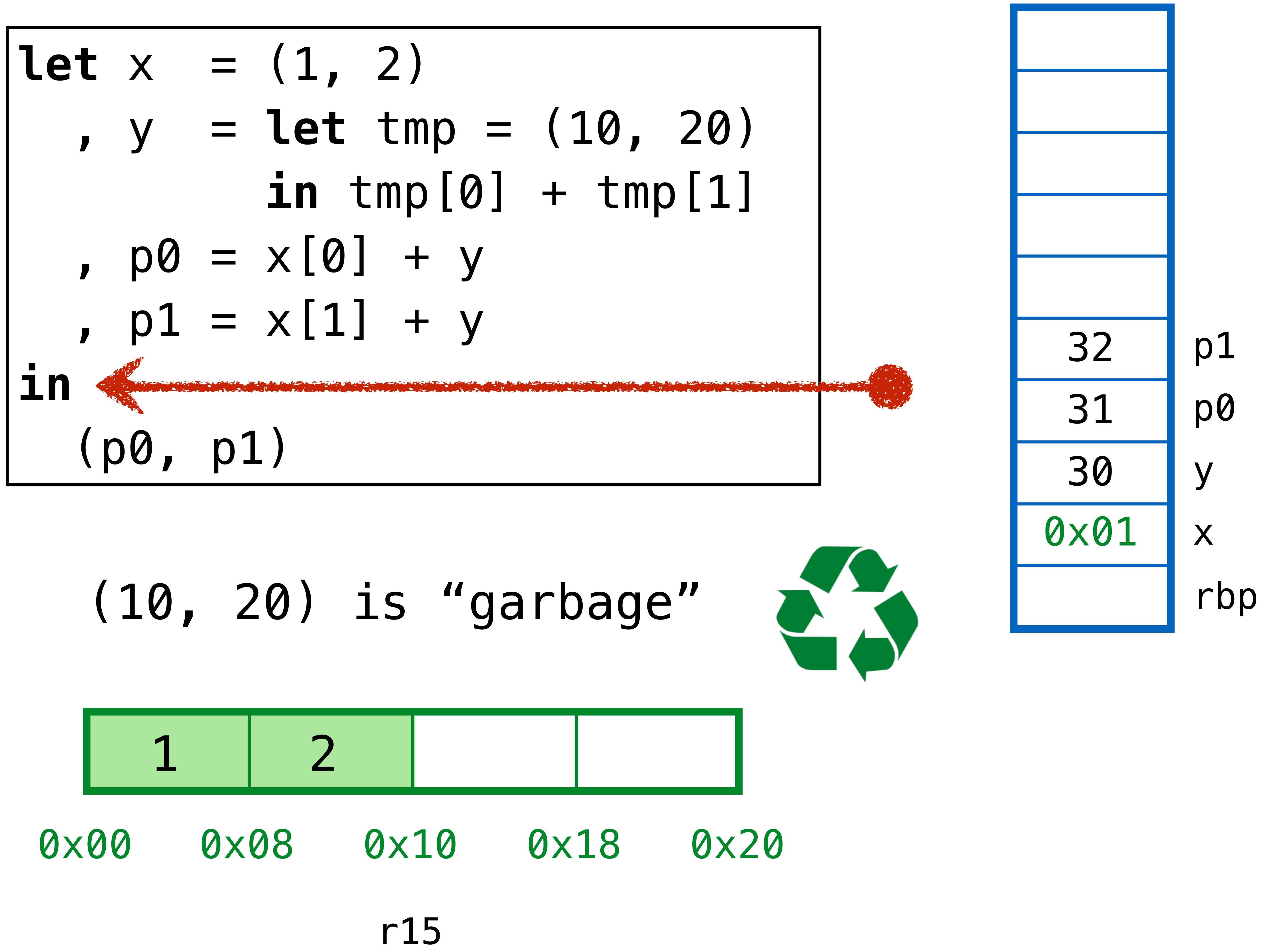


`(10, 20)` is “garbage”



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”

1	2		

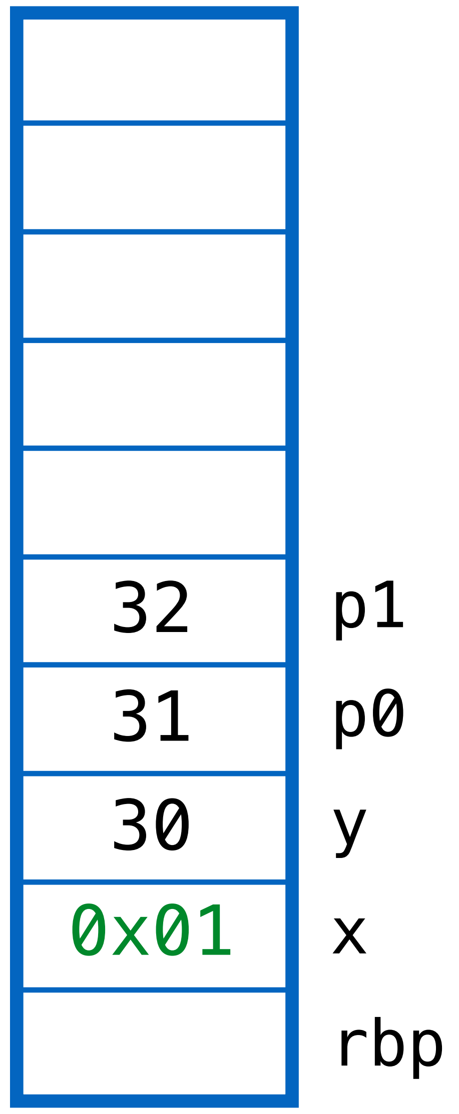
0x00 0x08 0x10 0x18 0x20

r15

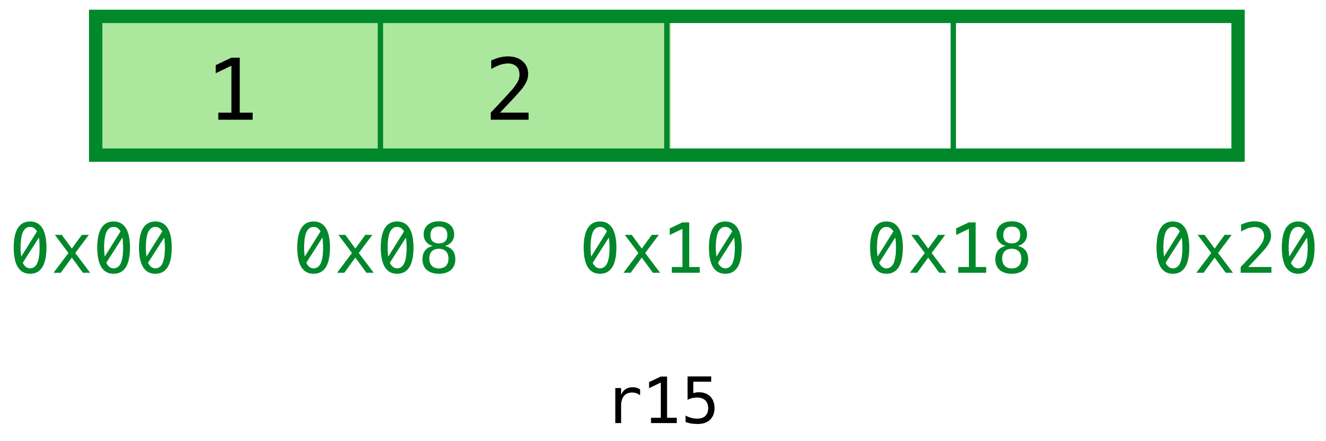
32	p1
31	p0
30	y
0x01	x
	rbp

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
rbp



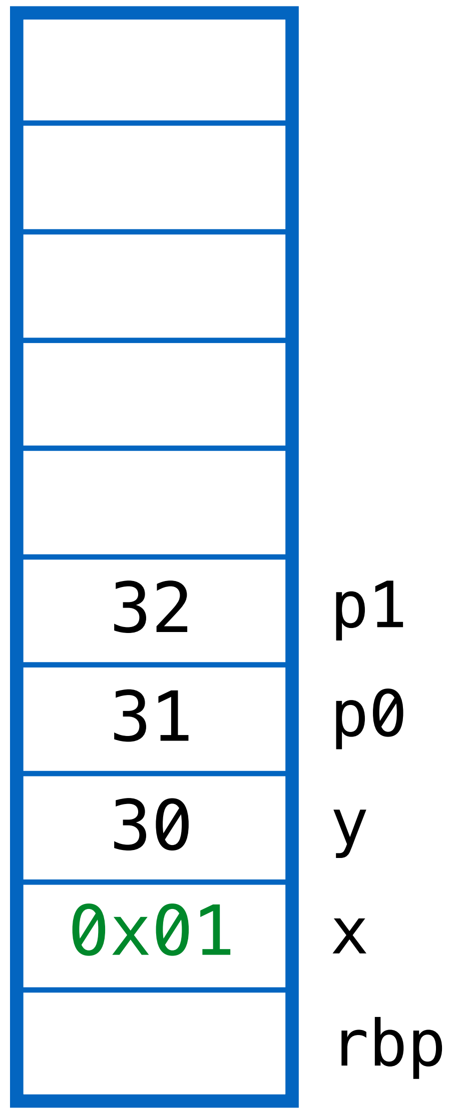
1 | 2 | |

0x00 0x08 0x10 0x18 0x20

r15

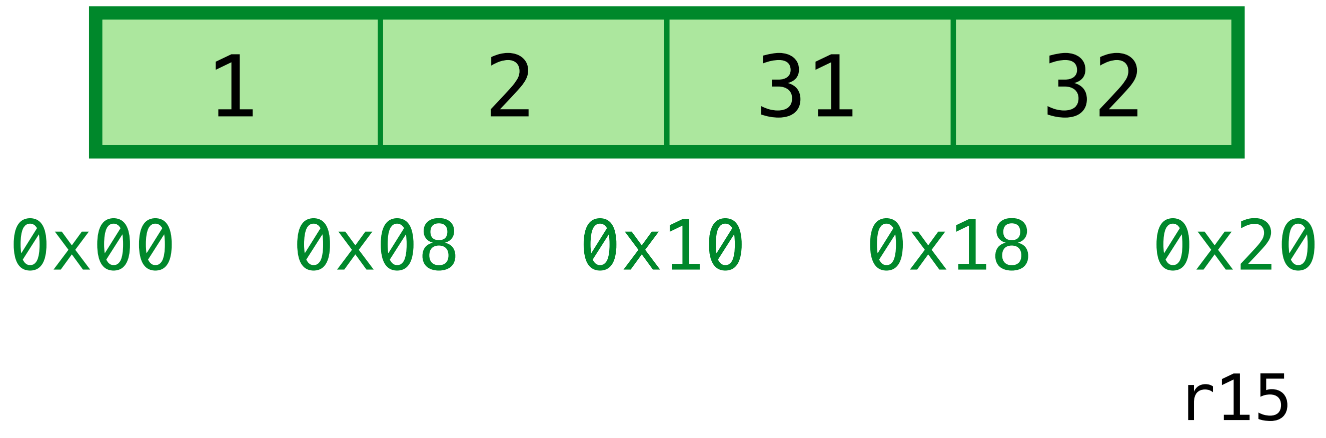
ex1: garbage at end

```
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
	rbp

Result (rax) = 0x11



1	2	31	32

0x00 0x08 0x10 0x18 0x20

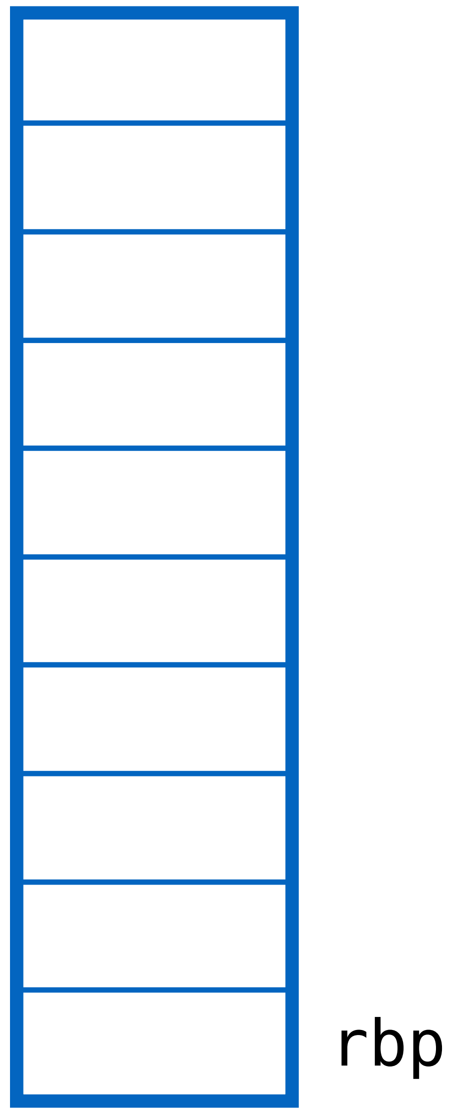
r15

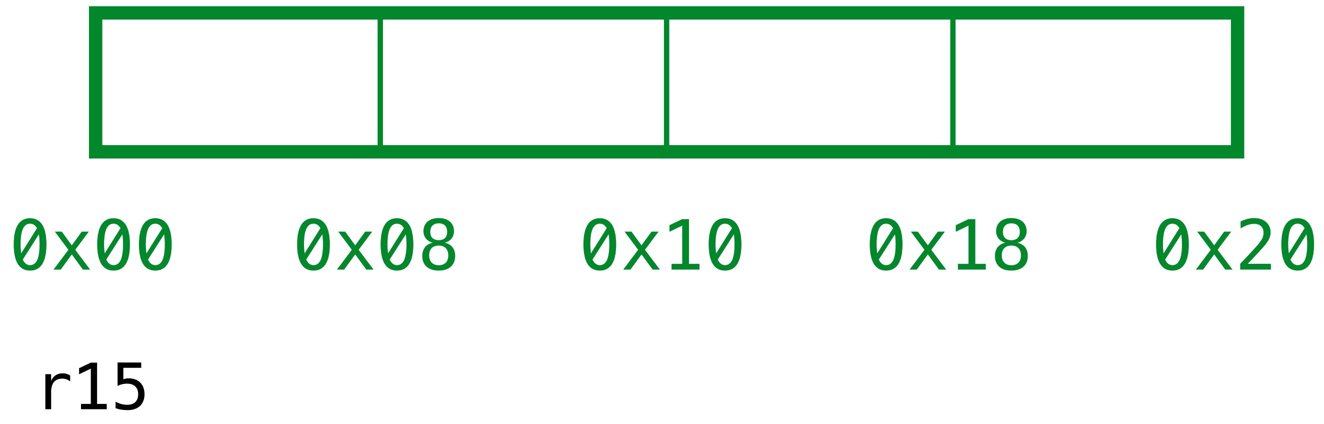
Garter / GC

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)
```

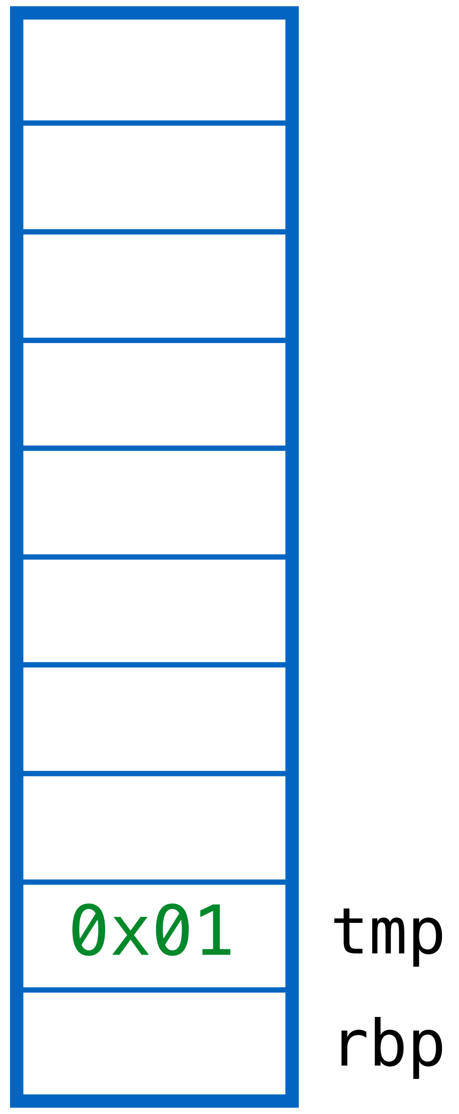


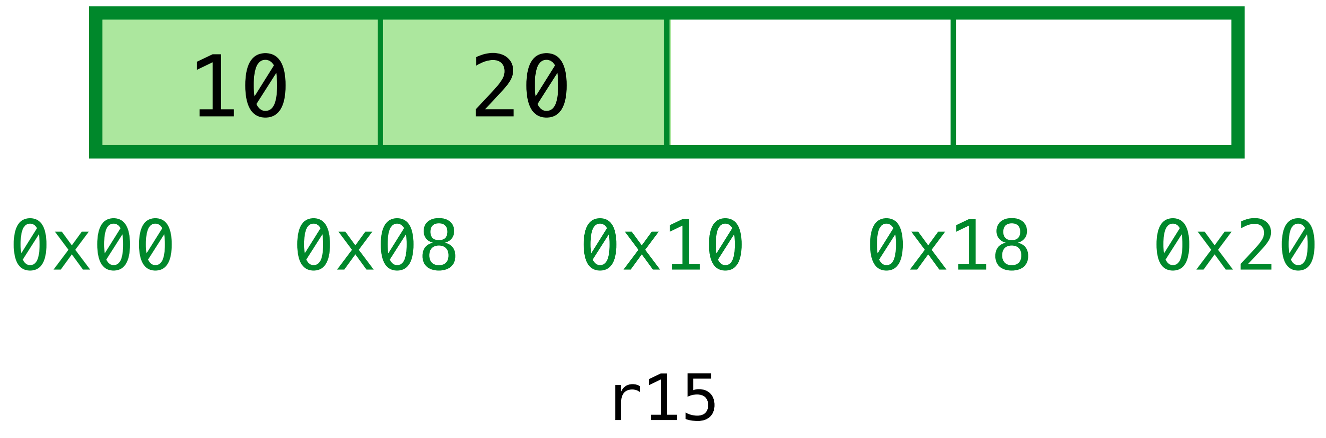


Start with a 4-word heap

ex2: garbage in the middle

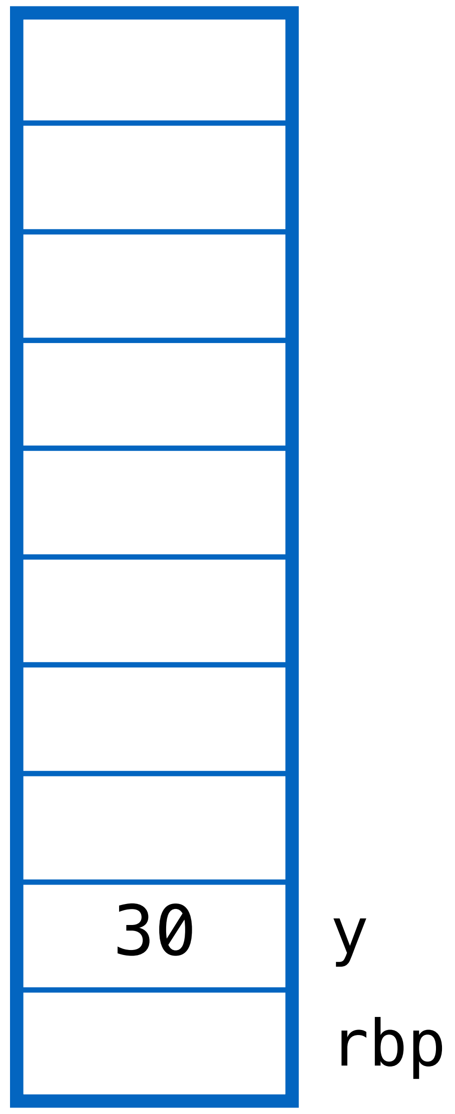
```
let y  = let tmp = (10, 20)
         in tmp[0] + tmp[1]
  , x  = (1, 2)
  , p0 = x[0] + y
  , p1 = x[1] + y
in
  (p0, p1)
```

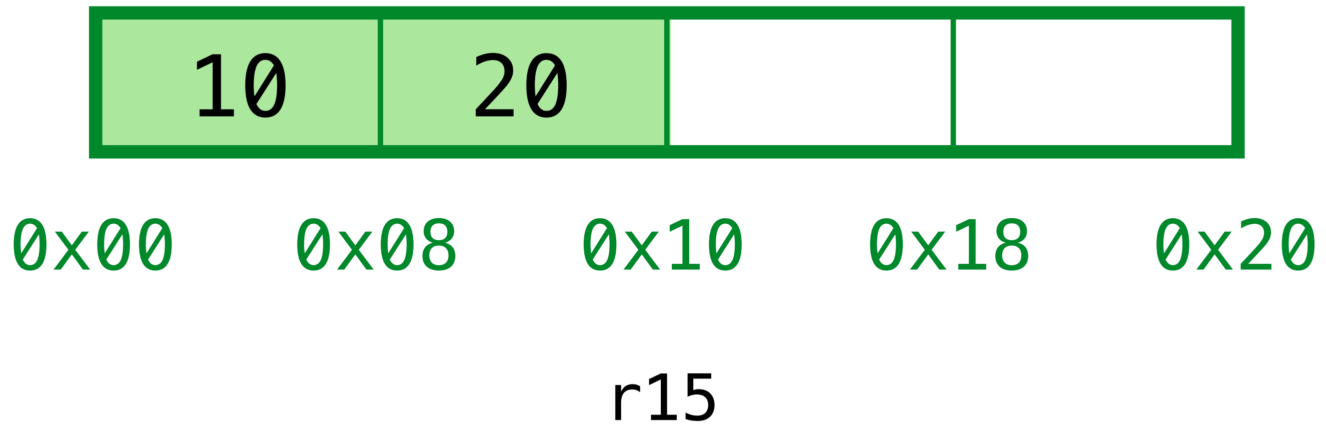




ex2: garbage in the middle

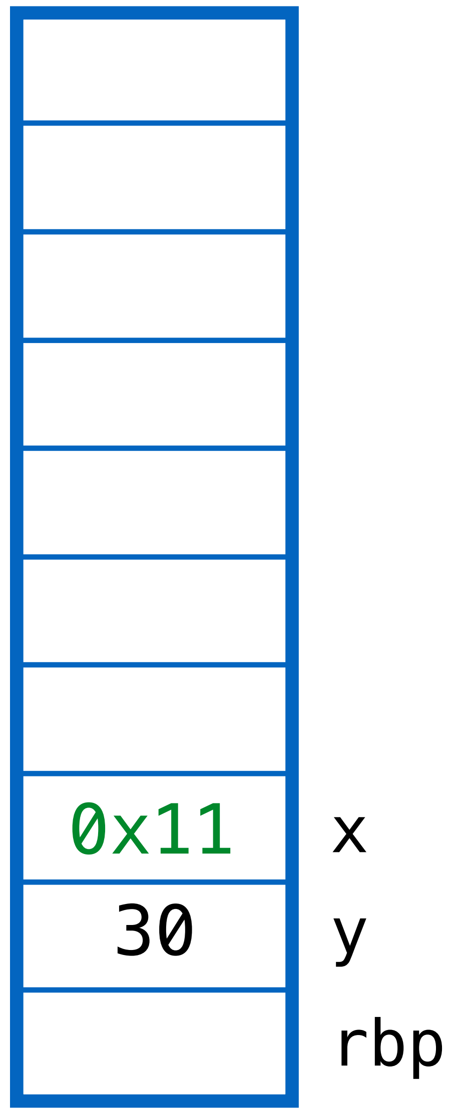
```
let y  = let tmp = (10, 20)
         in tmp[0] + tmp[1]
  , x  = (1, 2)
  , p0 = x[0] + y
  , p1 = x[1] + y
in
  (p0, p1)
```

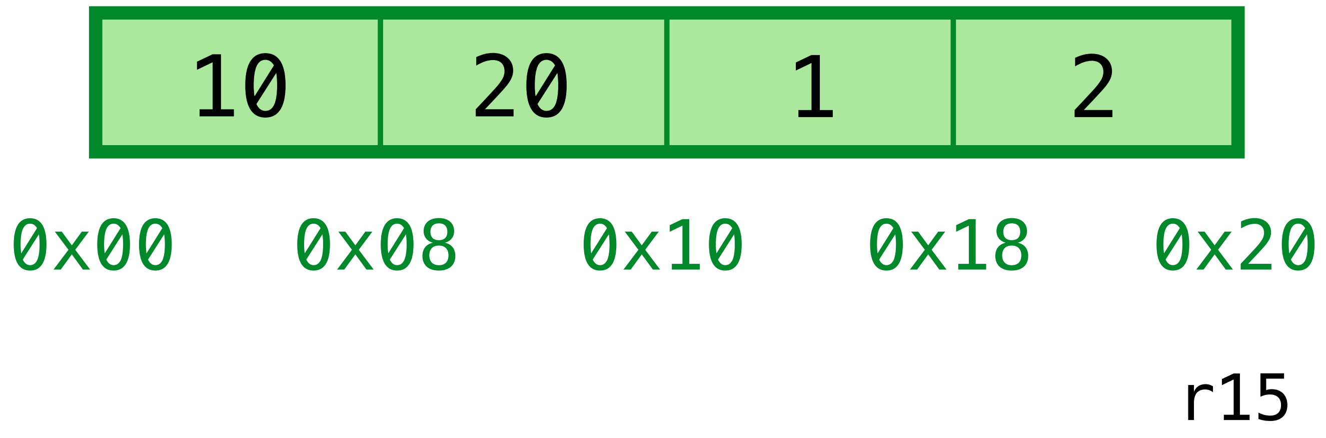




ex2: garbage in the middle

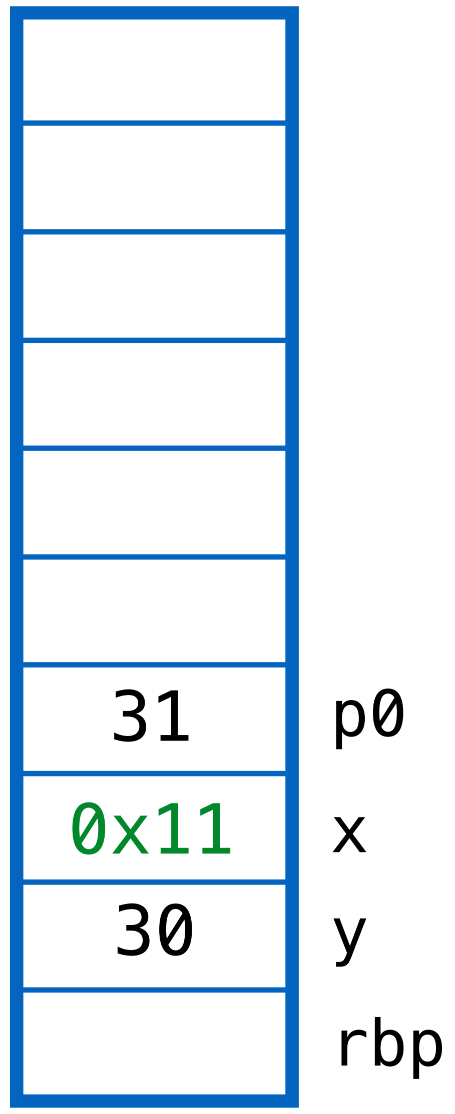
```
let y  = let tmp = (10, 20)
         in tmp[0] + tmp[1]
  , x  = (1, 2)
  , p0 = x[0] + y
  , p1 = x[1] + y
in
  (p0, p1)
```

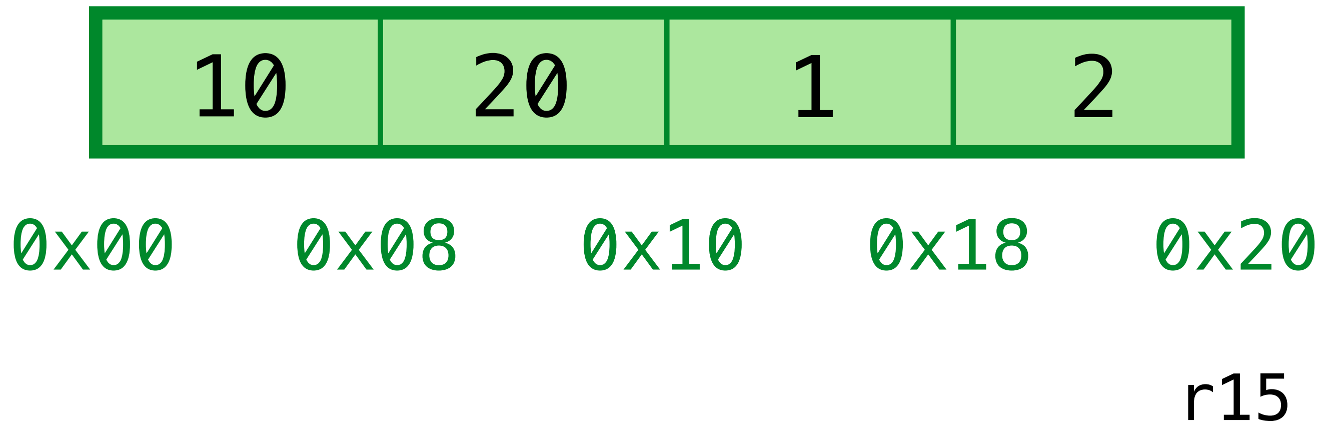




ex2: garbage in the middle

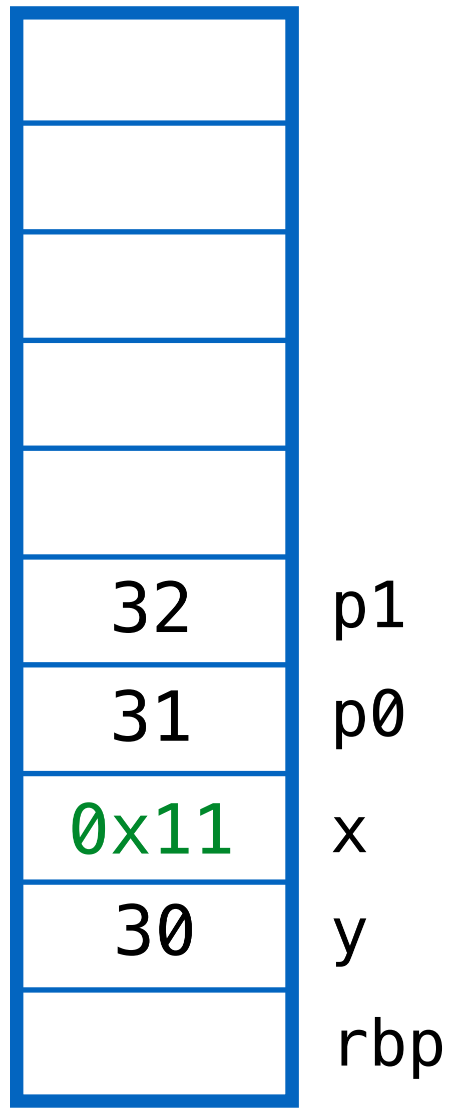
```
let y  = let tmp = (10, 20)
         in tmp[0] + tmp[1]
  , x  = (1, 2)
  , p0 = x[0] + y
  , p1 = x[1] + y
in
  (p0, p1)
```



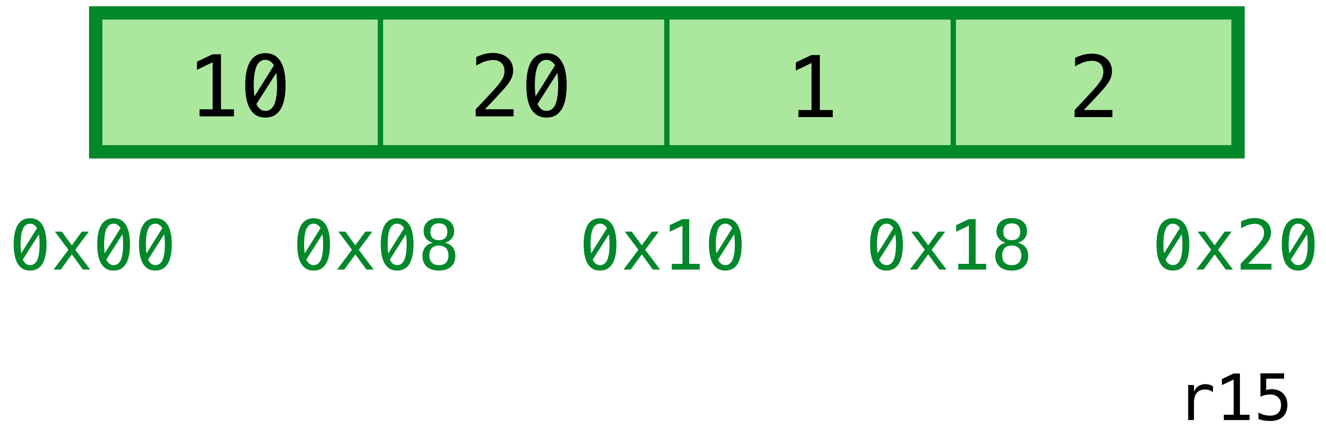


ex2: garbage in the middle

```
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)
```



32	p1
31	p0
0x11	x
30	y
	rbp



10	20	1	2

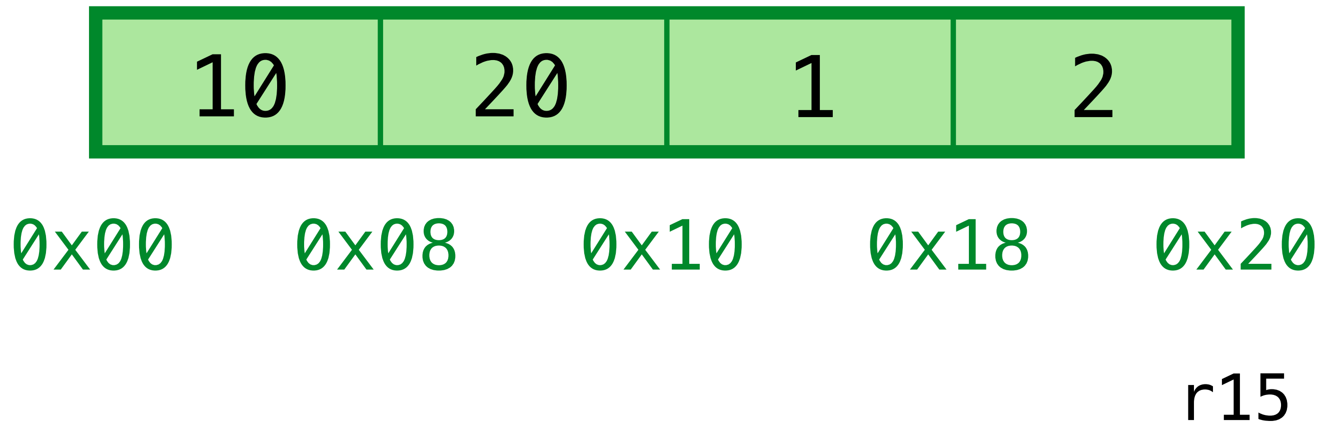
0x00 0x08 0x10 0x18 0x20

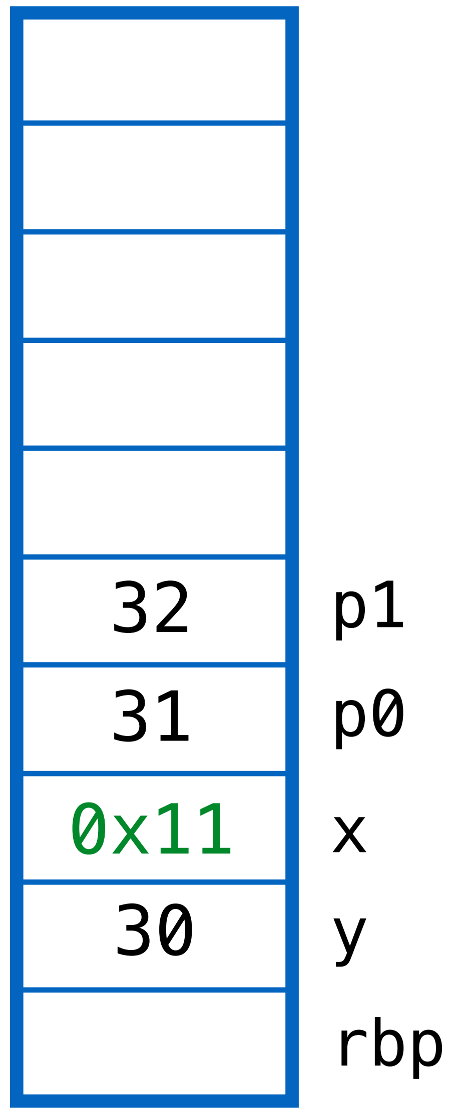
r15

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)
```

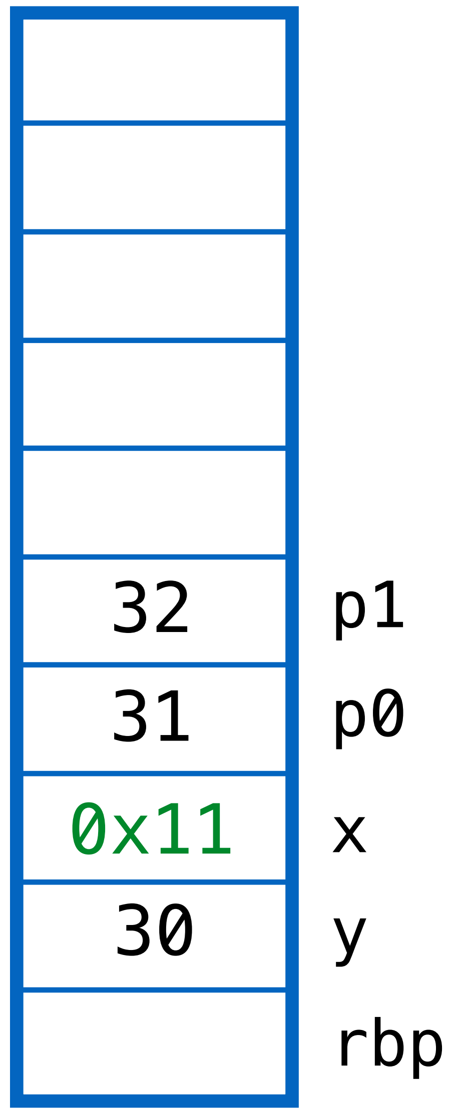
Out of memory!
Can’t allocate **`(p0, p1)`**



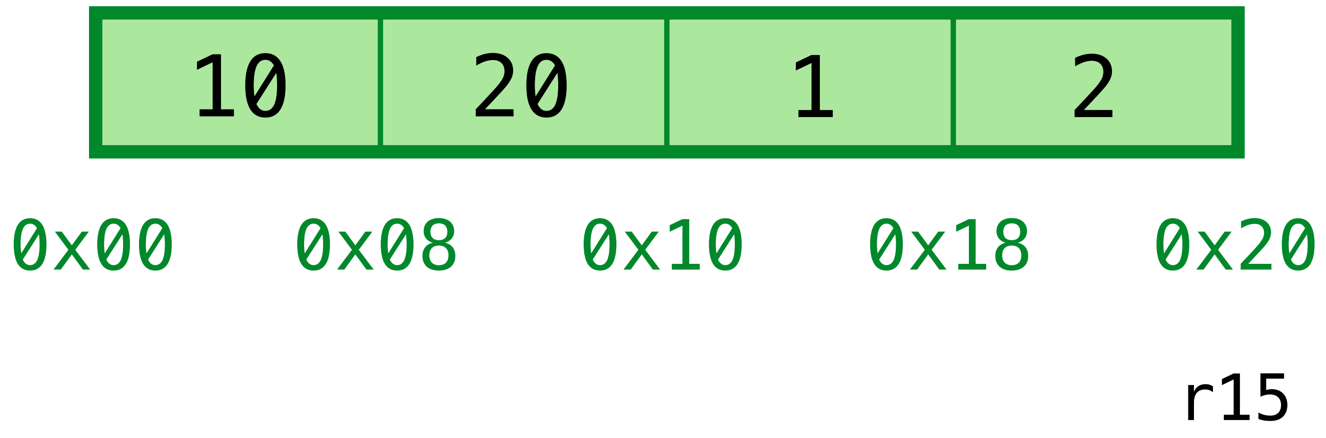


ex2: garbage in the middle

```
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)
```

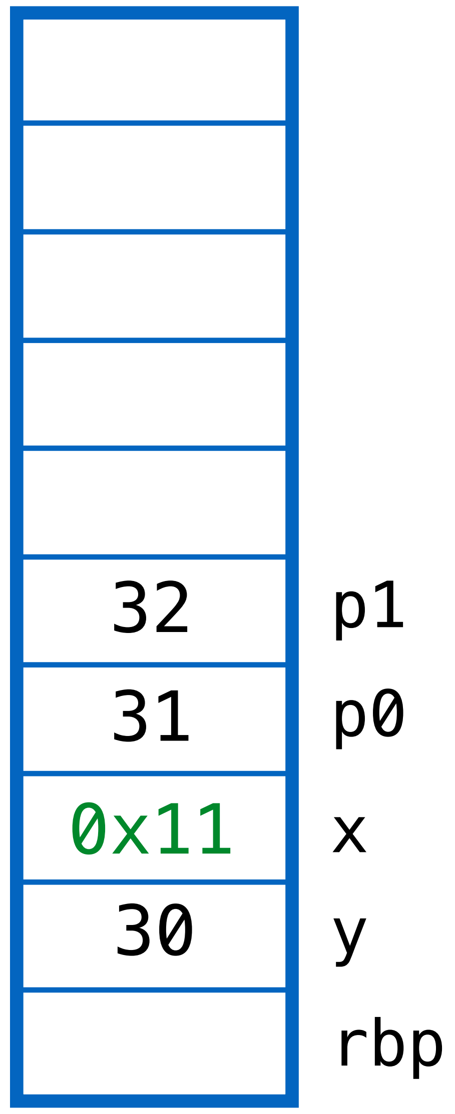


Lets reclaim & recycle garbage!



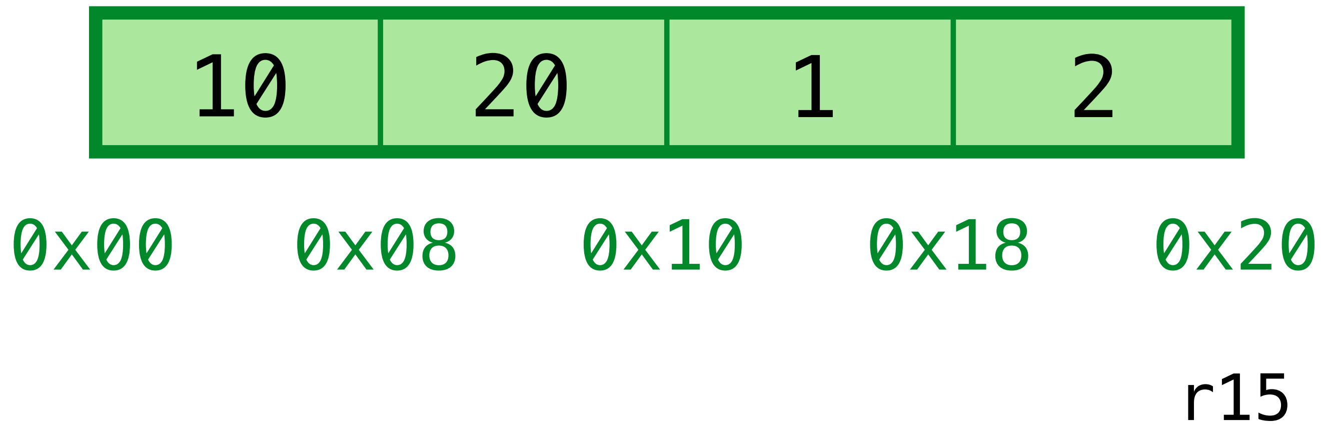
ex2: garbage in the middle

```
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
0x11	x
30	y
	rbp

Lets reclaim & recycle garbage!



10	20	1	2

0x00 0x08 0x10 0x18 0x20

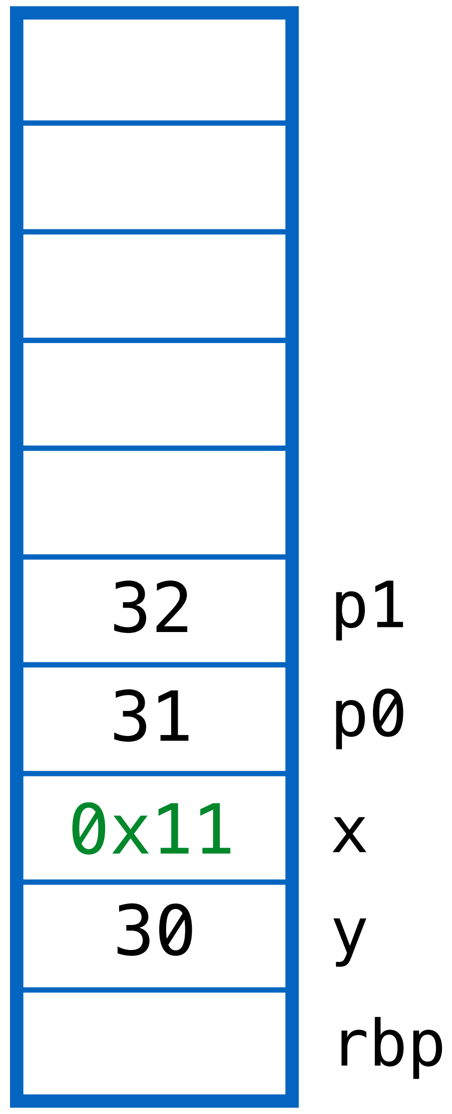
r15

QUIZ: Which cells are garbage?

(A) `0x00, 0x08` (B) `0x08, 0x10` (C) `0x18, 0x20` (D) None (E) All

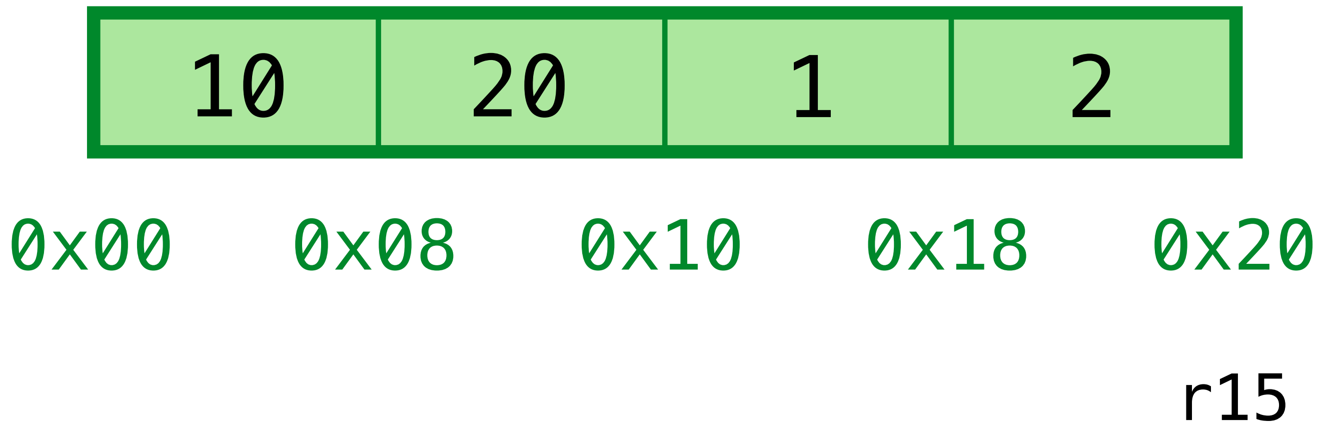
ex2: garbage in the middle

```
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
0x11	x
30	y
	rbp

Lets reclaim & recycle garbage!



10	20	1	2

0x00 0x08 0x10 0x18 0x20

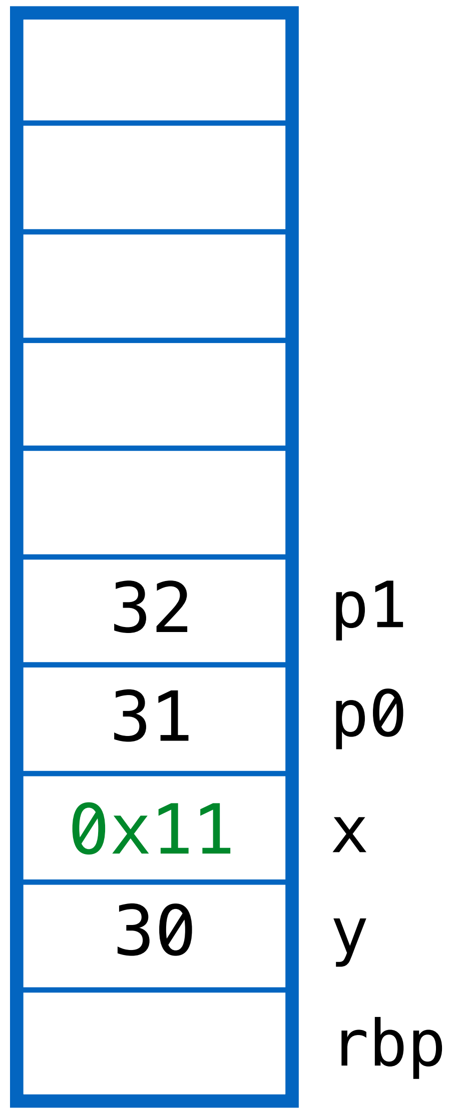
r15

QUIZ: Which cells are garbage?

Those that are *not reachable from stack*

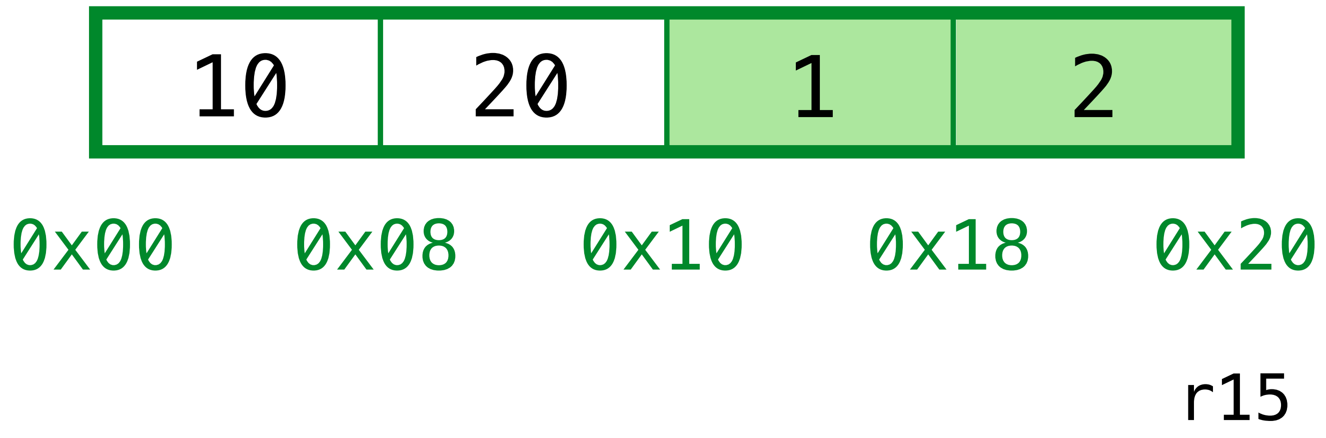
ex2: garbage in the middle

```
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
0x11	x
30	y
	rbp

Lets reclaim & recycle garbage!



10	20	1	2

0x00 0x08 0x10 0x18 0x20

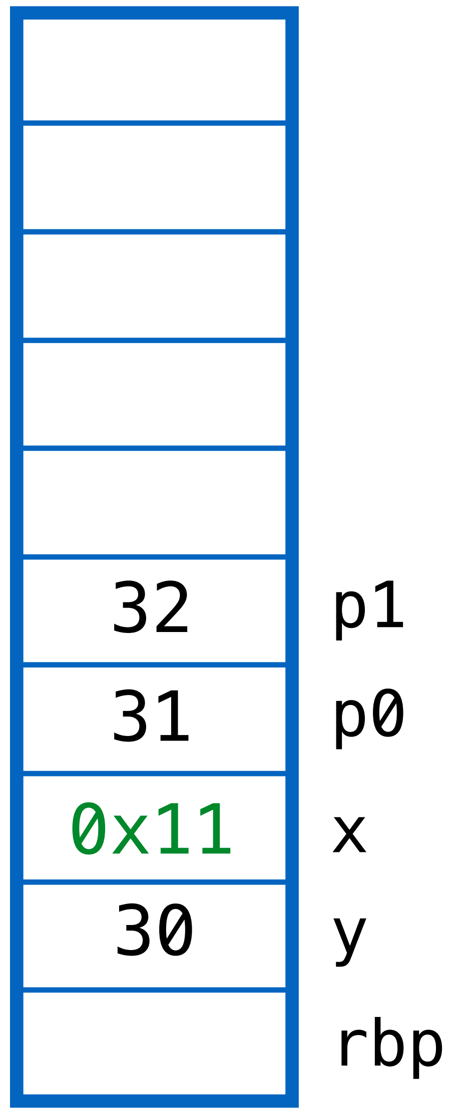
r15

QUIZ: Which cells are garbage?

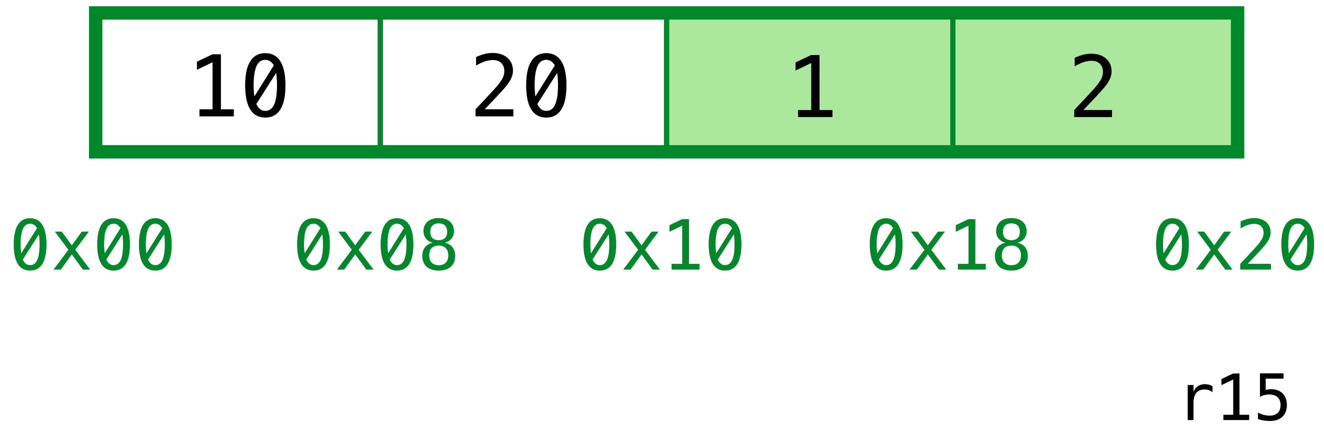
Those that are *not reachable from stack*

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!

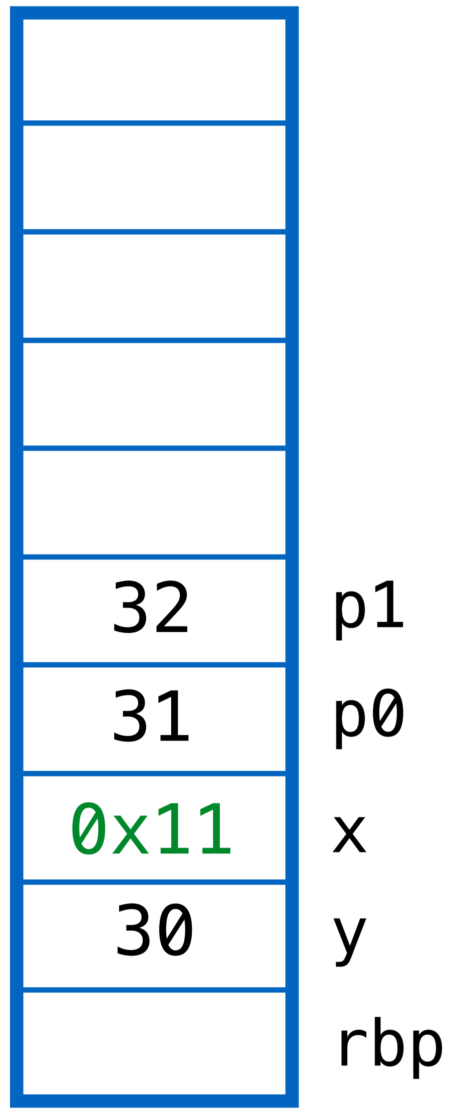


Q: How to reclaim space?

Why is it not enough to rewind `r15`?

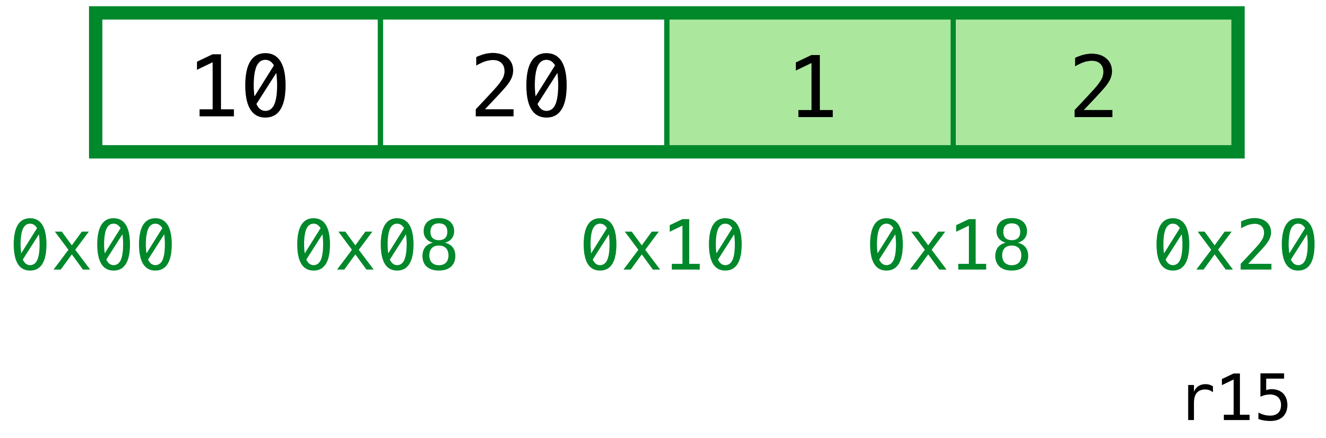
ex2: garbage in the middle

```
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
0x11	x
30	y
	rbp

Lets reclaim & recycle garbage!



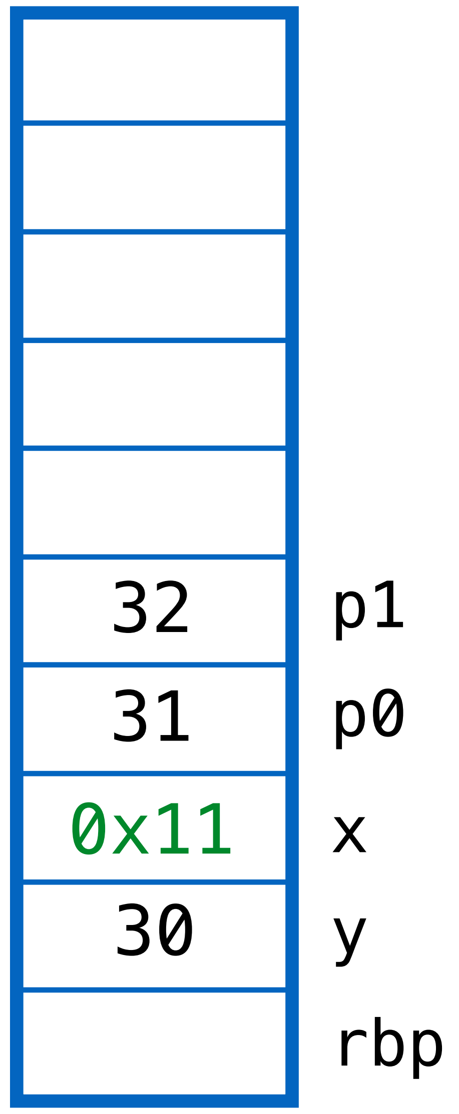
10	20	1	2	
0x00	0x08	0x10	0x18	0x20
				r15

Why is it not enough to rewind `r15`?

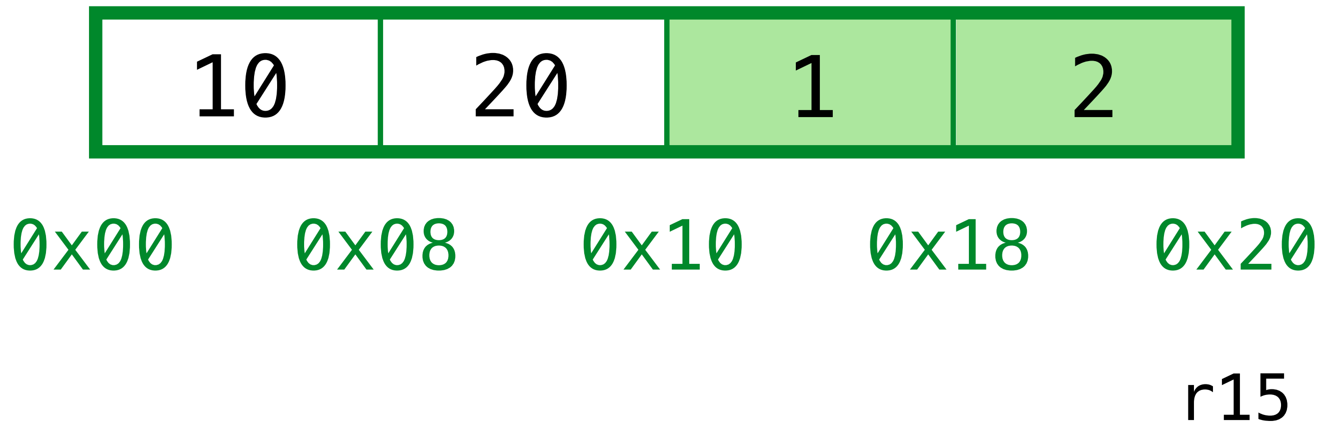
Want free space to be *contiguous* (i.e. go to end of heap)

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!

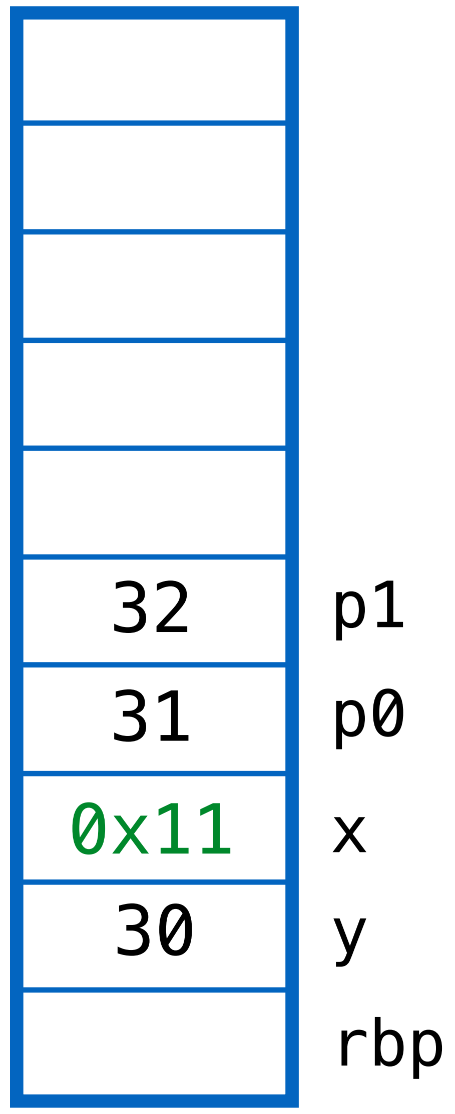


Solution: Compaction

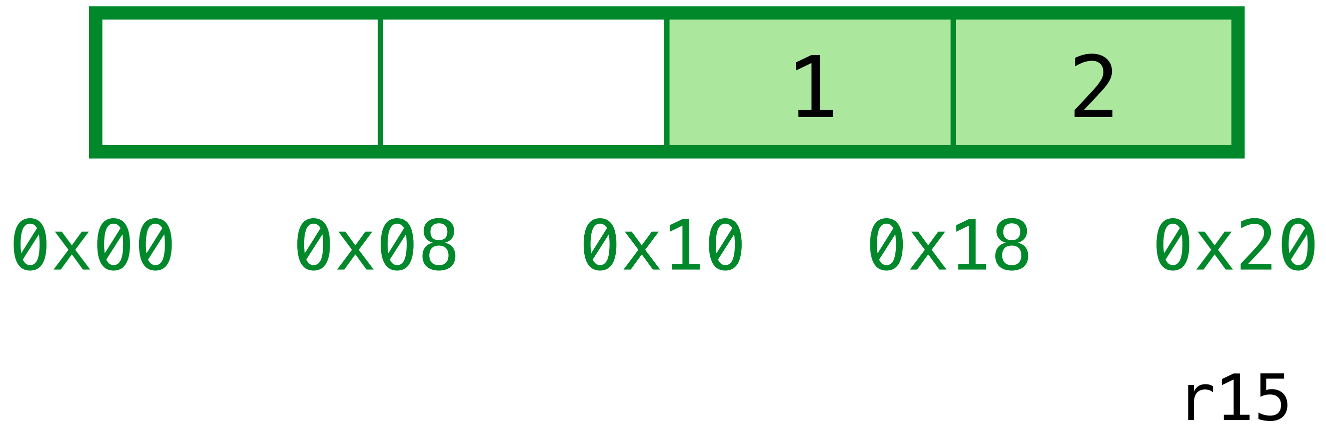
Copy “live” cells into “garbage” …

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!

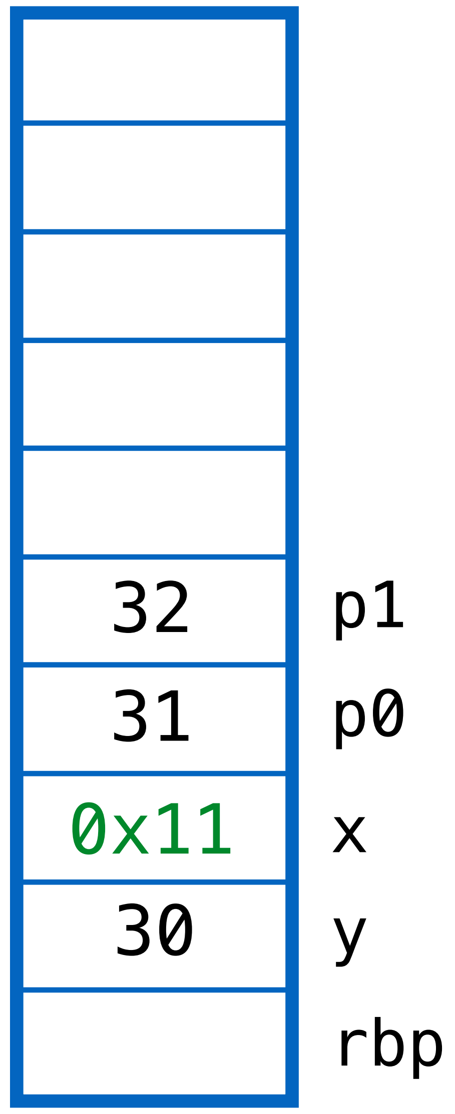


Solution: Compaction

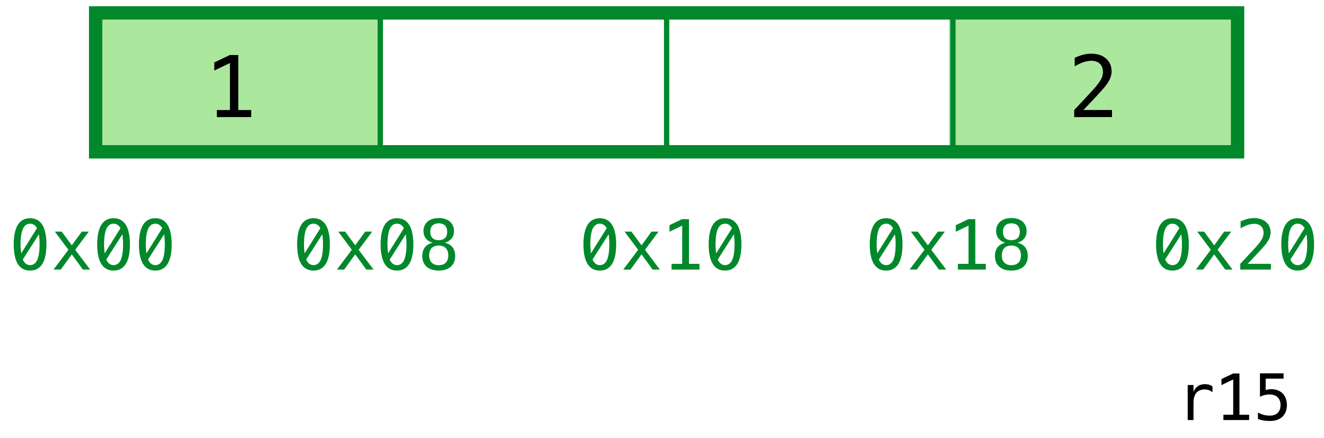
Copy “live” cells into “garbage” …

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)
```



Lets reclaim & recycle garbage!

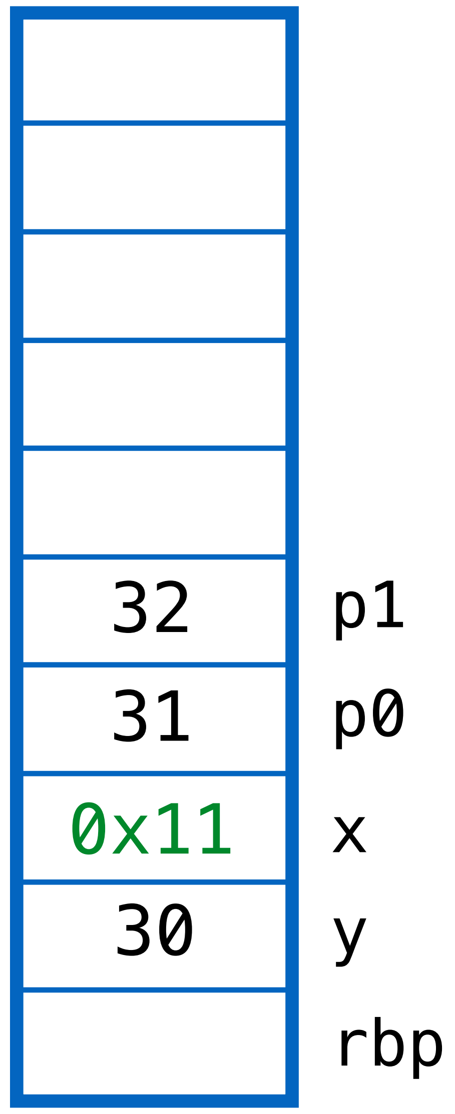


Solution: Compaction

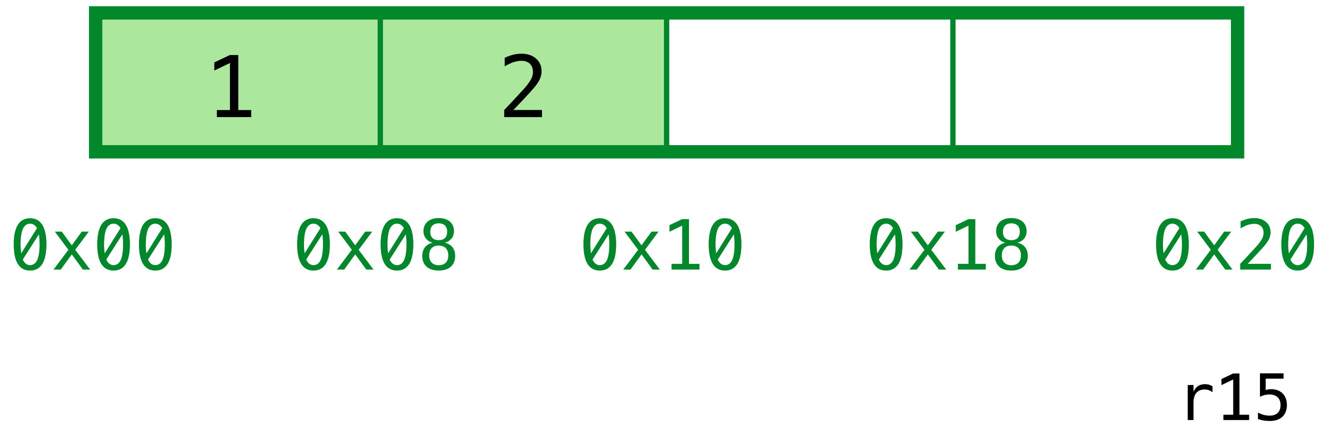
Copy “live” cells into “garbage” …

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!

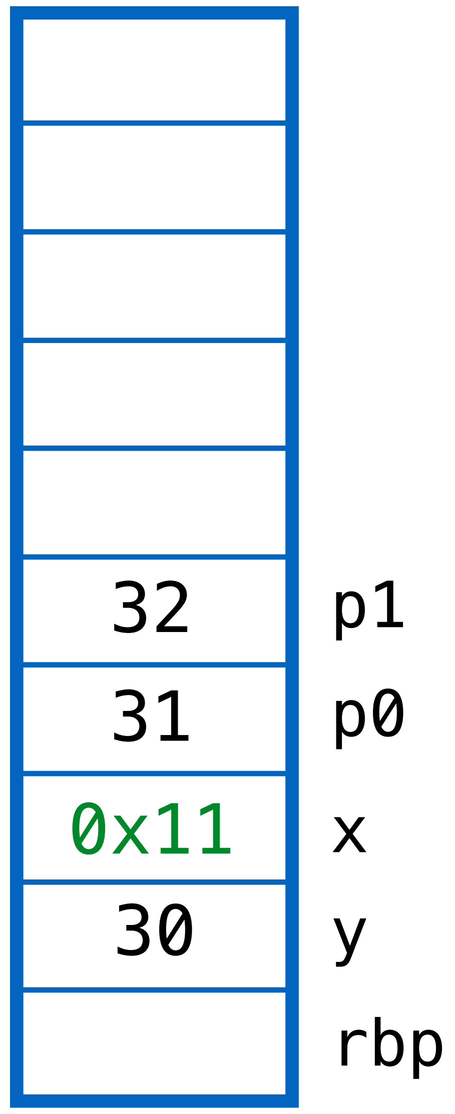


Solution: Compaction

Copy “live” cells into “garbage” …

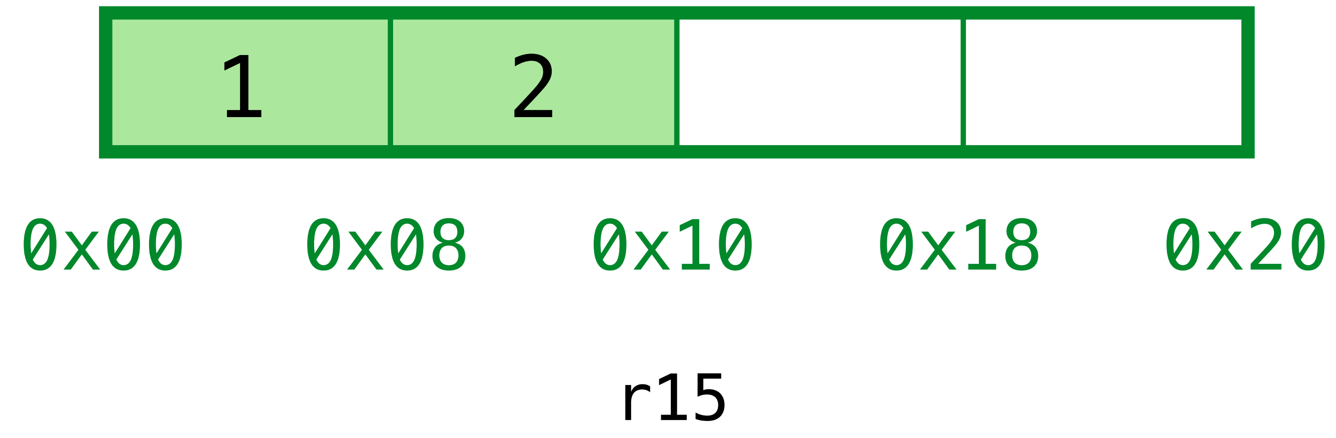
ex2: garbage in the middle

```
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
0x11	x
30	y
	rbp

Lets reclaim & recycle garbage!



1	2		

0x00 0x08 0x10 0x18 0x20

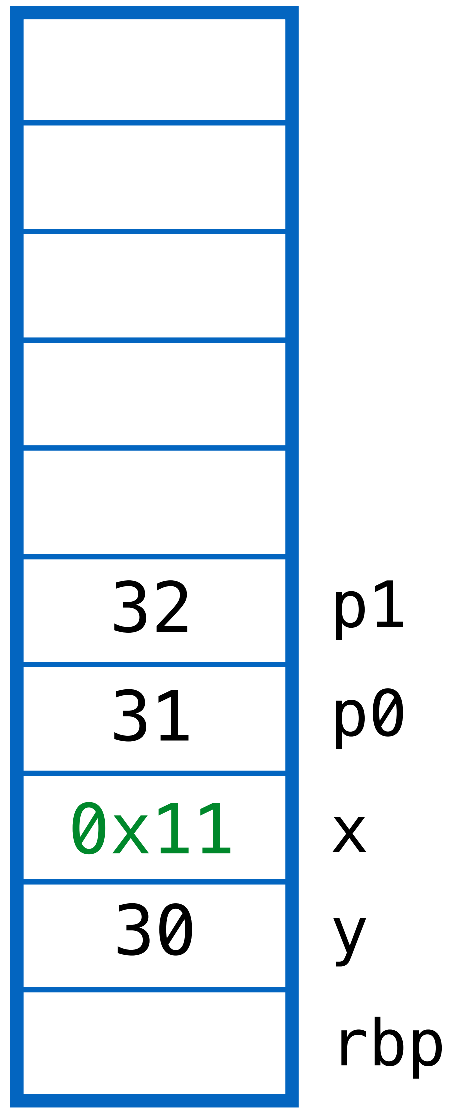
r15

Solution: Compaction

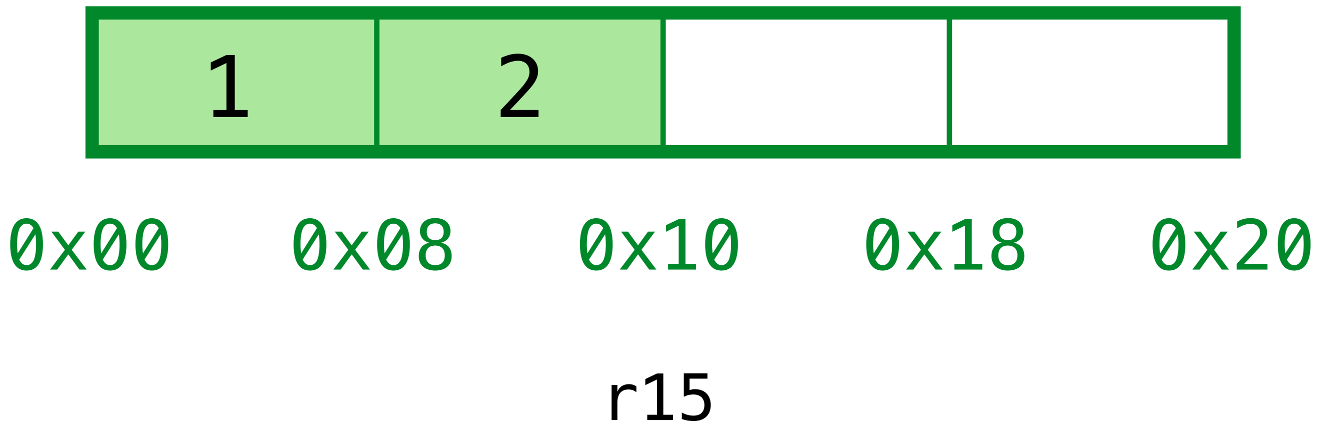
Copy “live” cells into “garbage” … ***and then*** … rewind `r15`!

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)
```

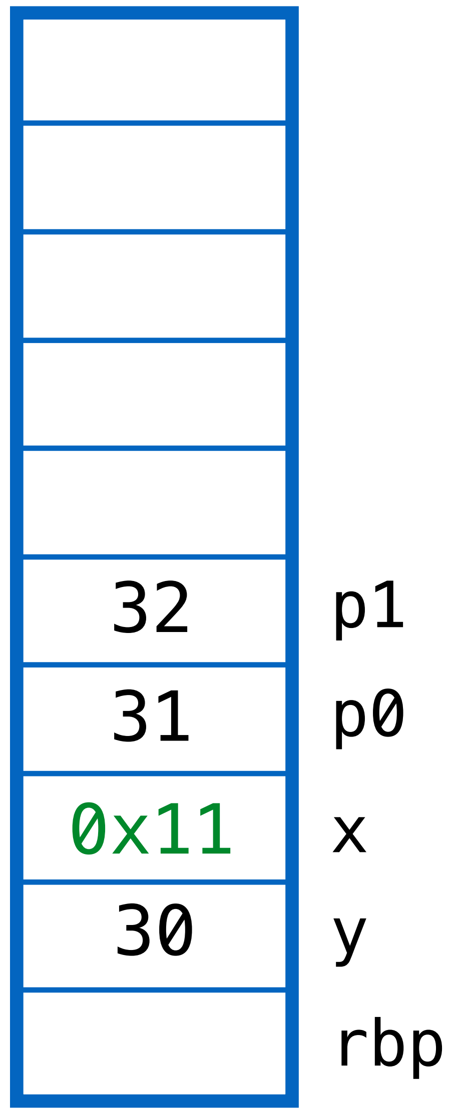


Yay! Have space for `(p0, p1)`



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

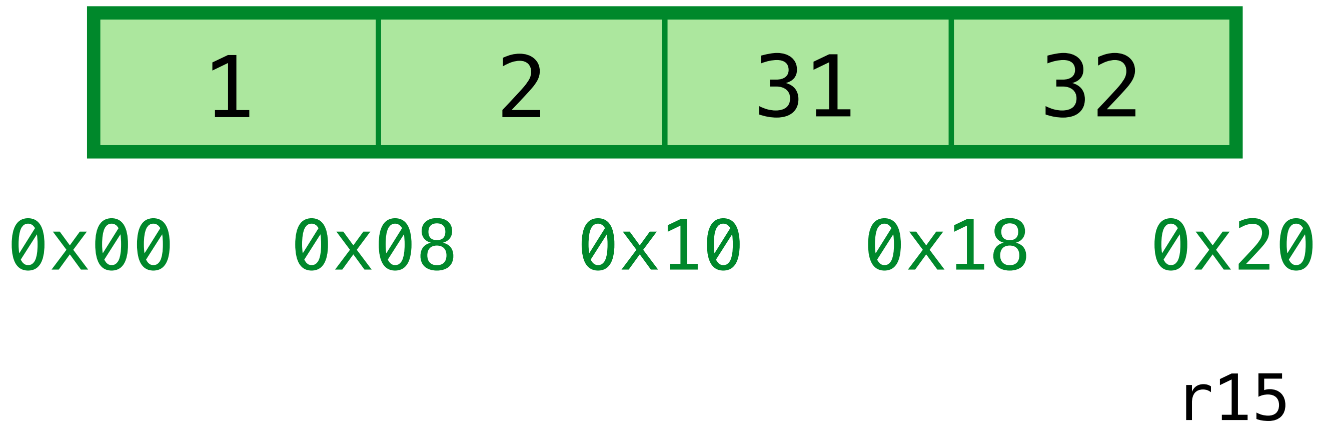
31 p0

0x11 x

30 y

rbp

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



1	2	31	32

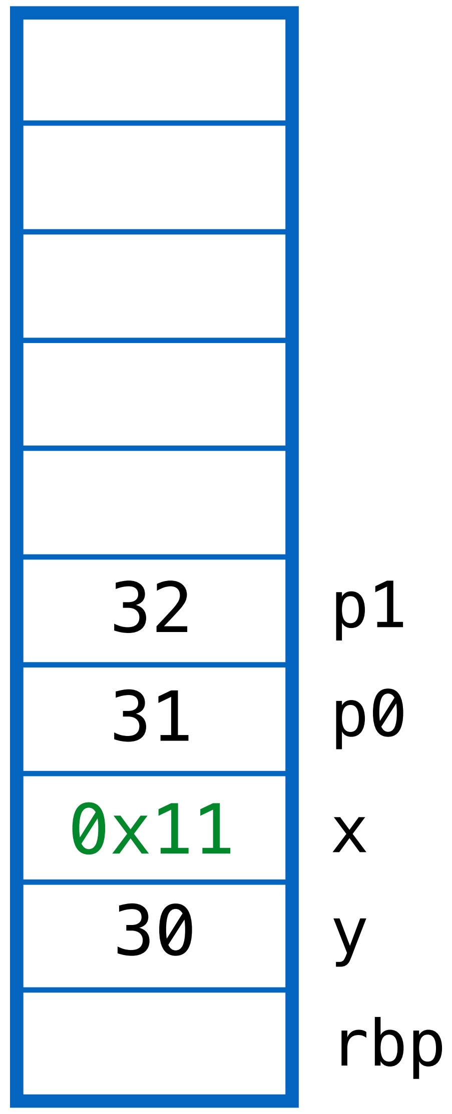
0x00 0x08 0x10 0x18 0x20

r15

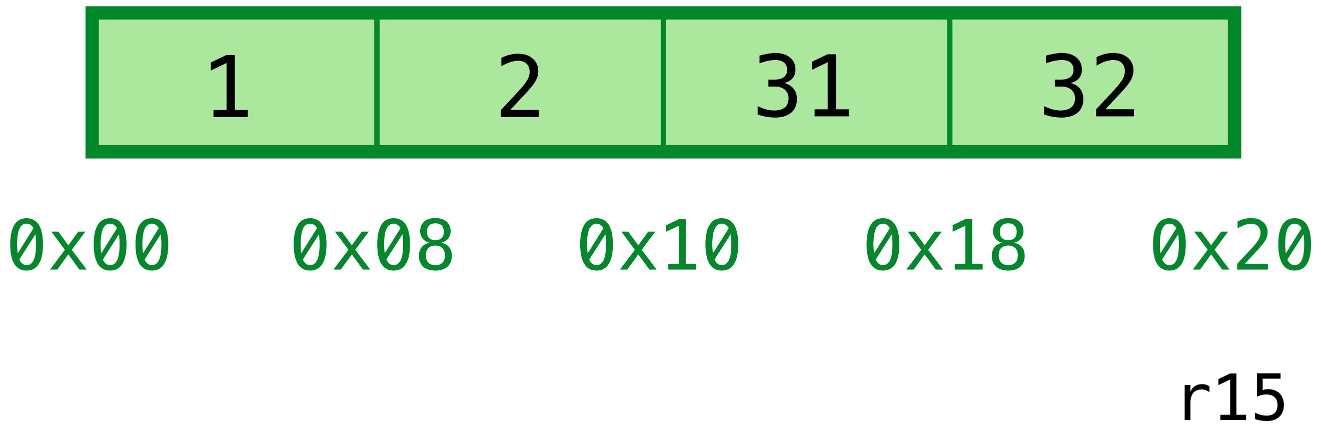
ex2: garbage in the middle

```
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 (rax) = 0x09



32	p1
31	p0
0x11	x
30	y
	rbp



1	2	31	32

0x00 0x08 0x10 0x18 0x20

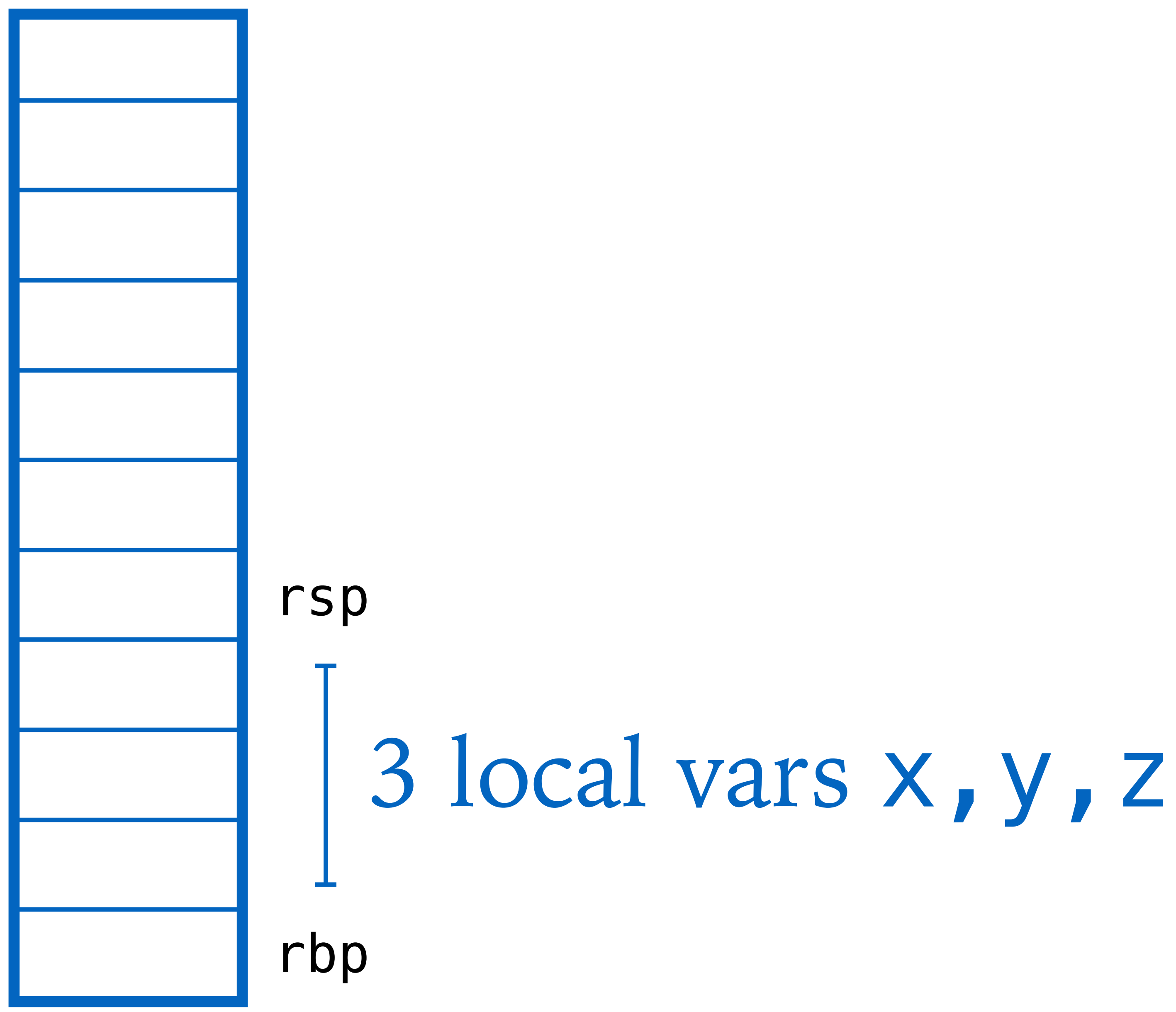
r15

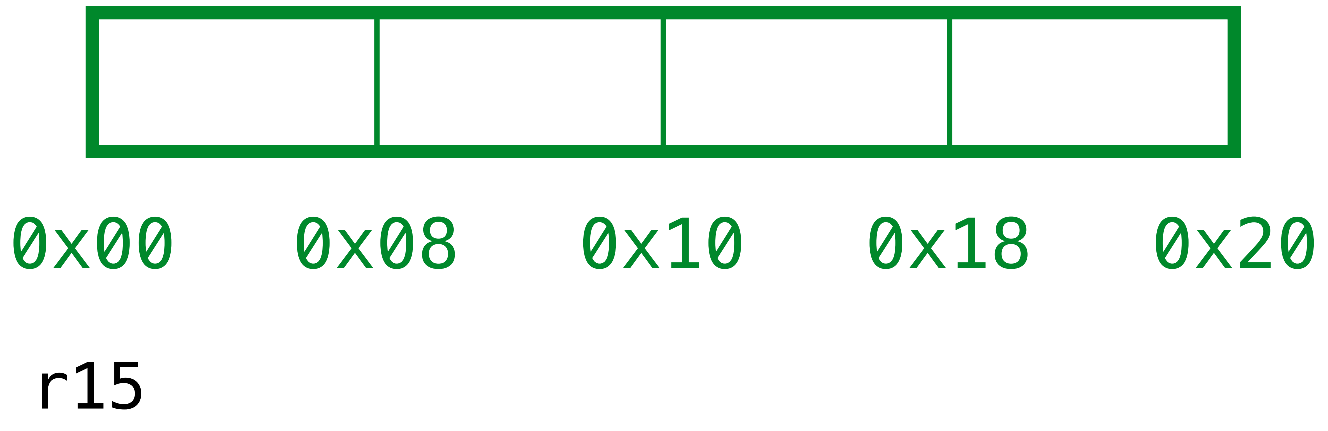
Garter / GC

Example 3

ex3: garbage in the middle (with stack)

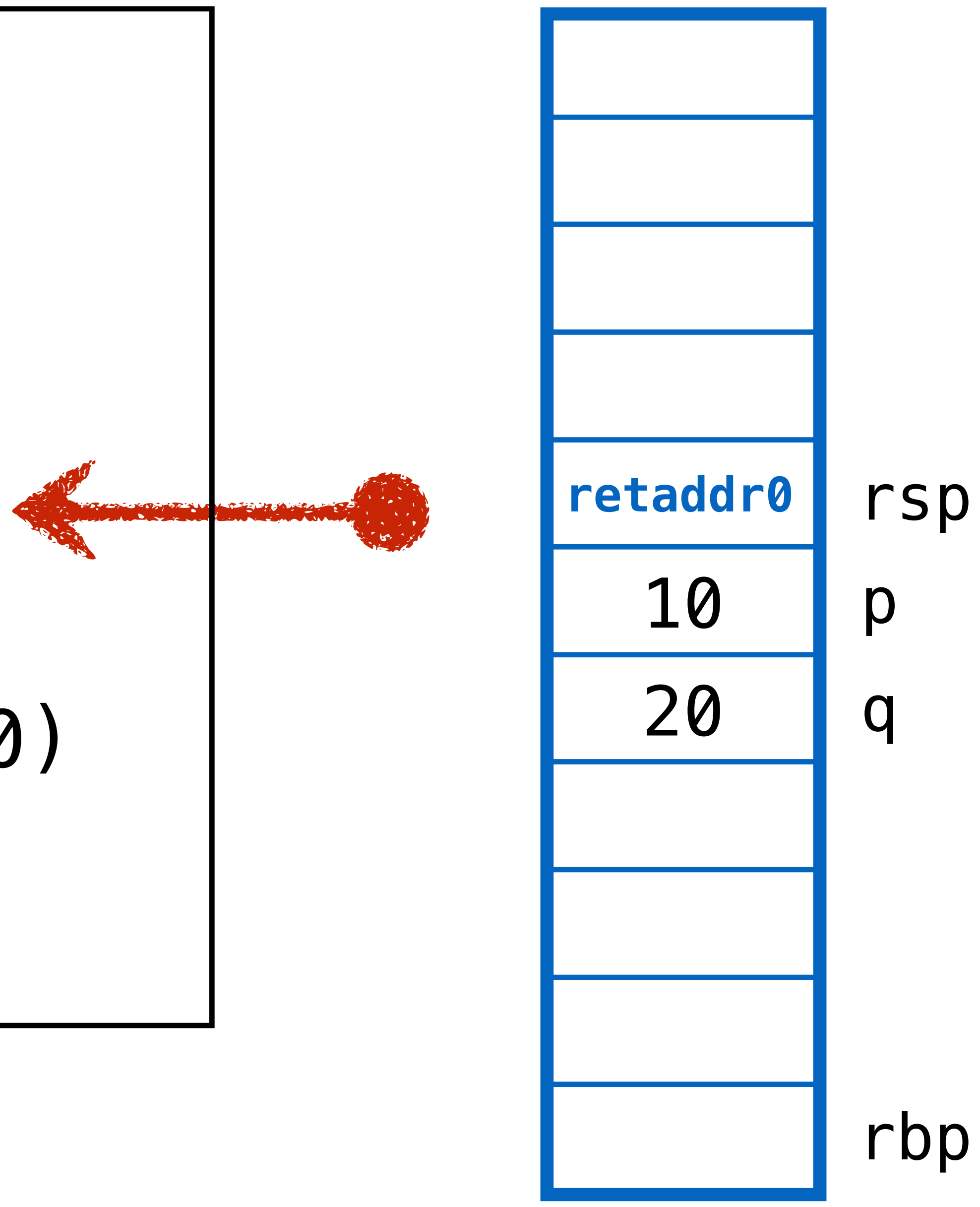
```
def foo(p, q):
  let tmp = (p, q)
  in tmp[0] + tmp[1]

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

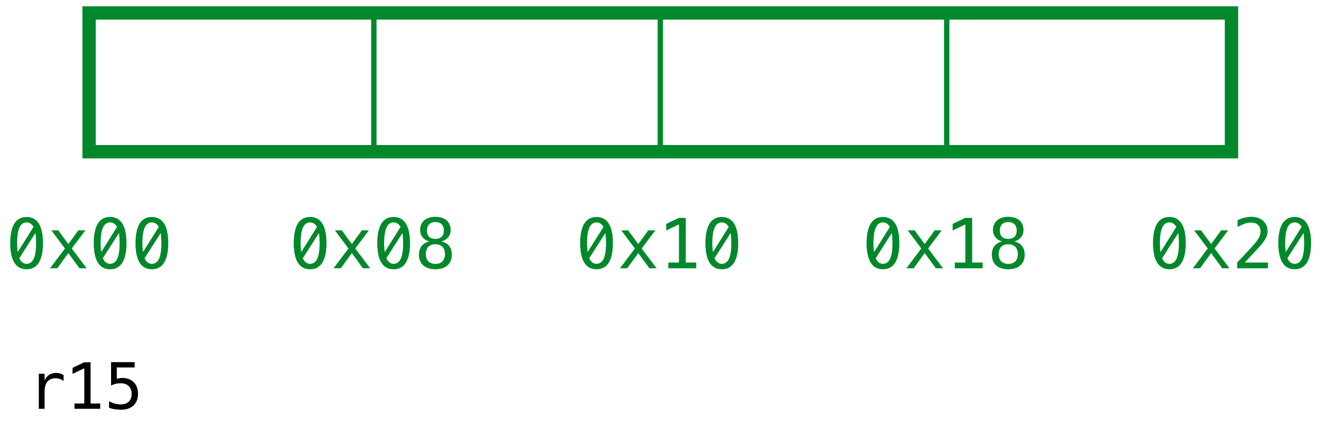




ex3: garbage in the middle (with stack)

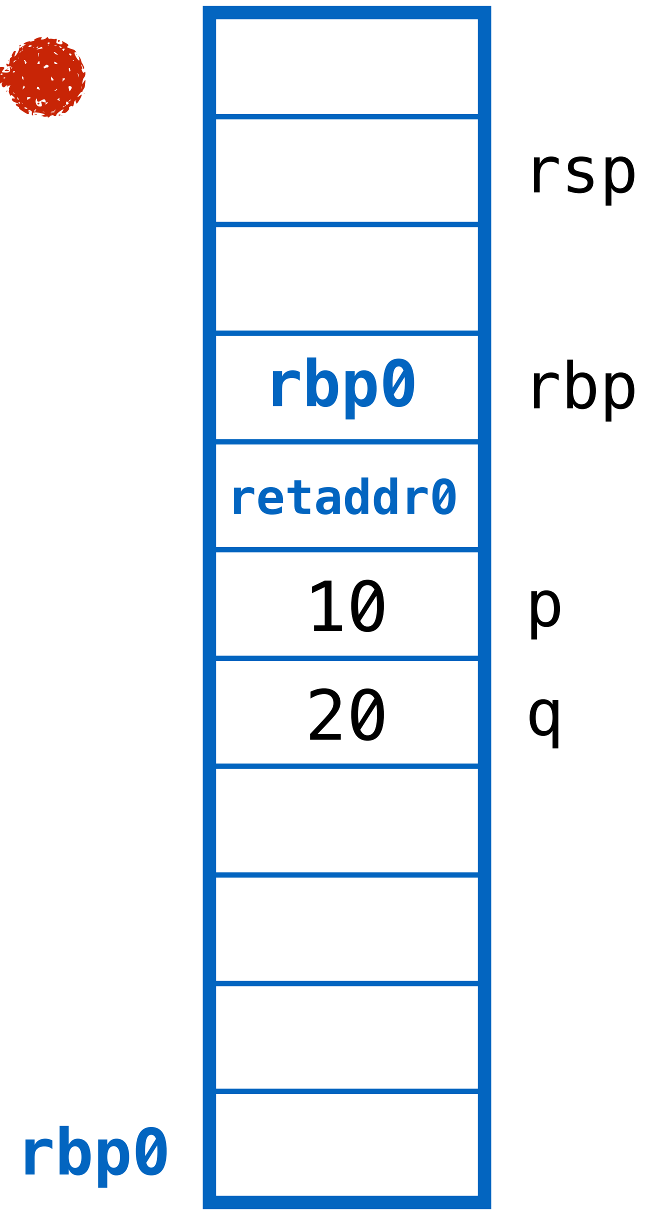
```
def foo(p, q):
  let tmp = (p, q)
  in tmp[0] + tmp[1]

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

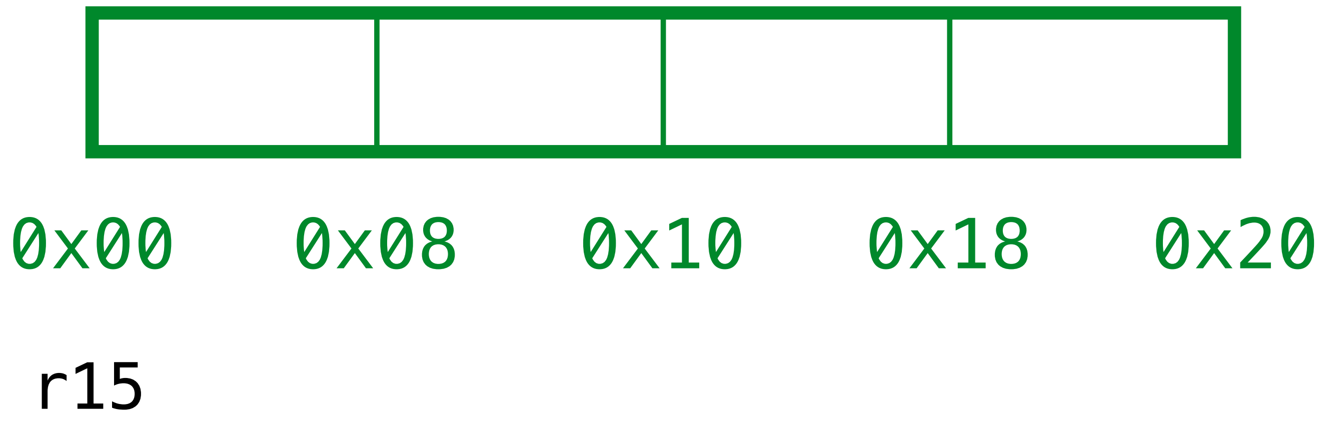




ex3: garbage in the middle (with stack)

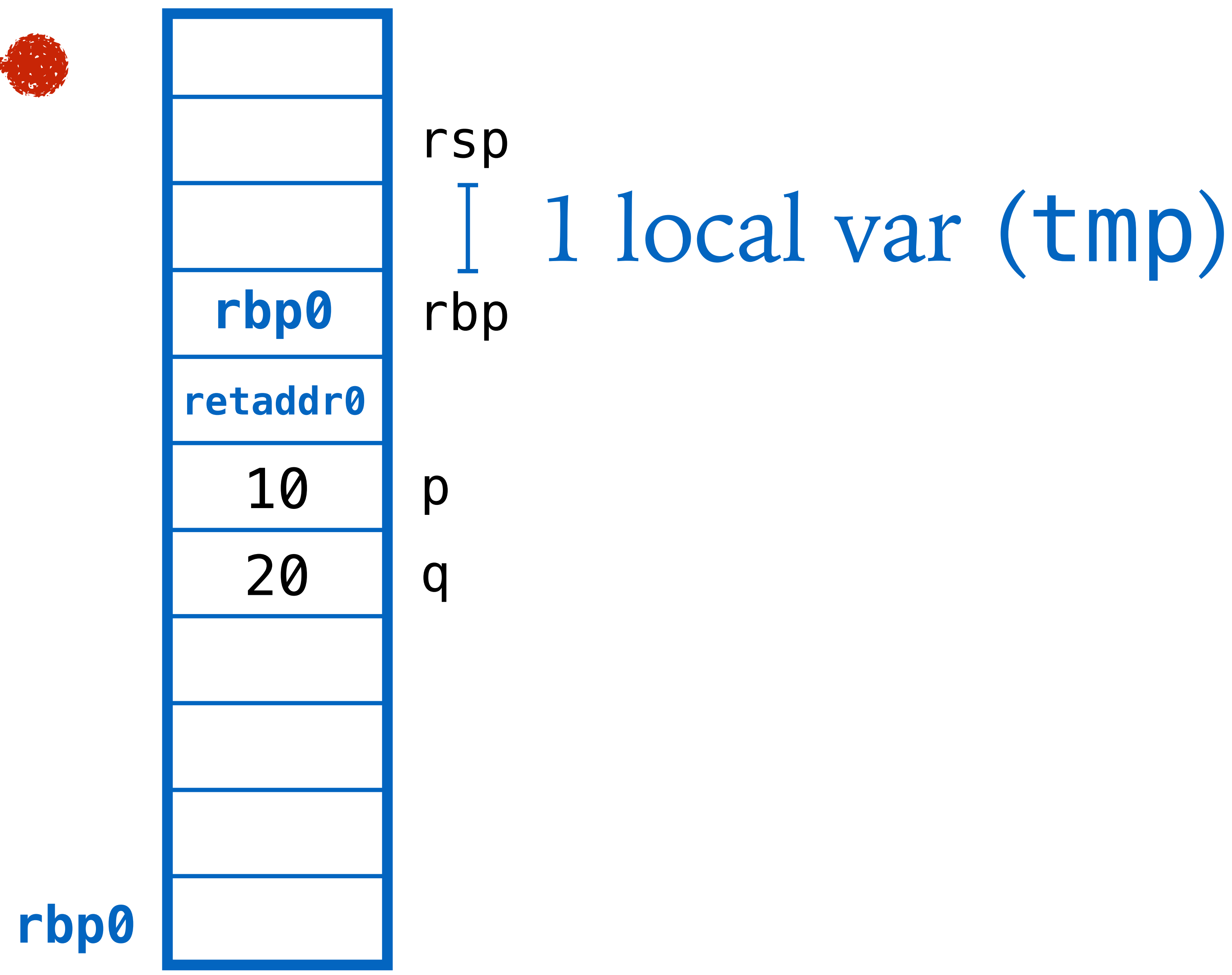
```
def foo(p, q):
  let tmp = (p, q)
  in tmp[0] + tmp[1]

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

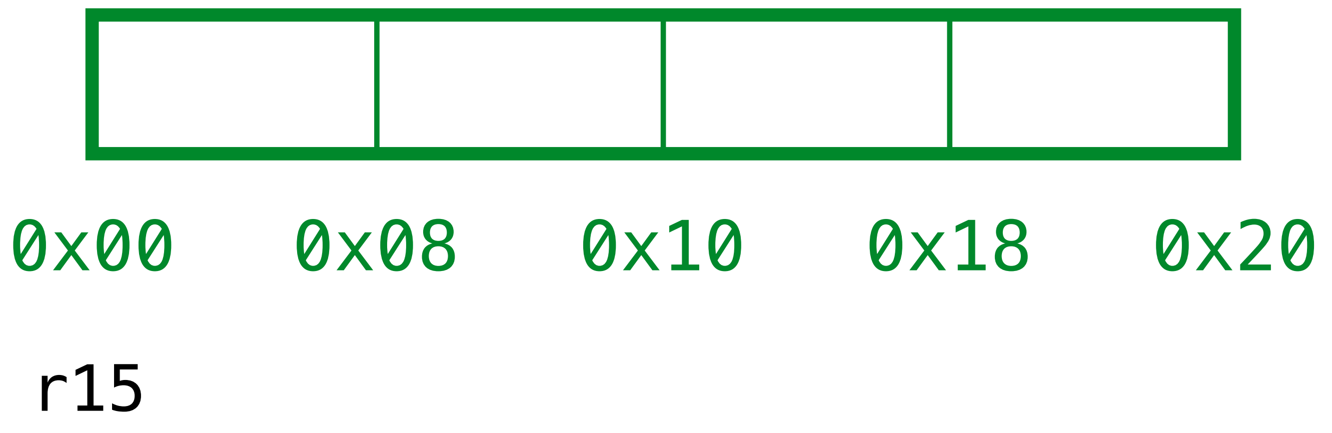




ex3: garbage in the middle (with stack)

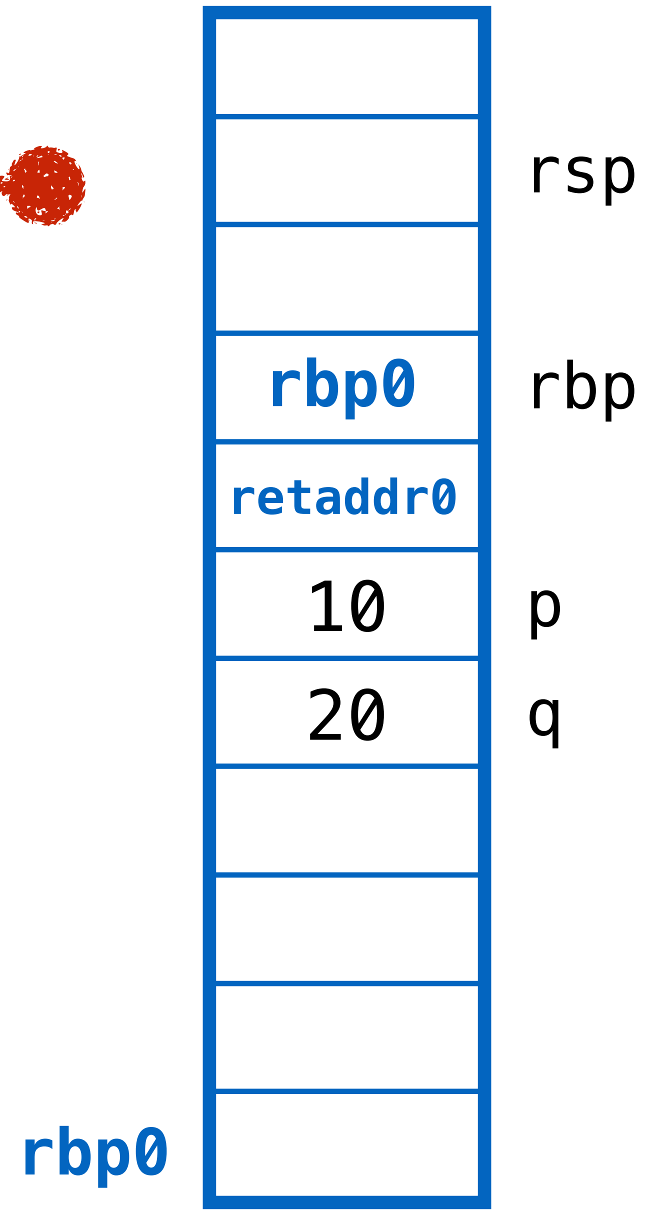
```
def foo(p, q):
  let tmp = (p, q)
  in tmp[0] + tmp[1]

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

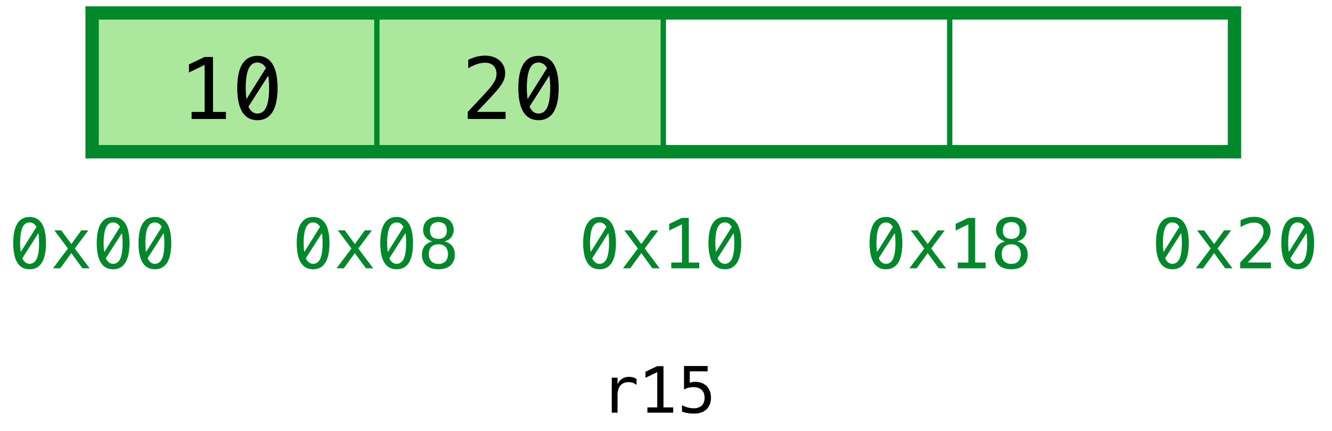




ex3: garbage in the middle (with stack)

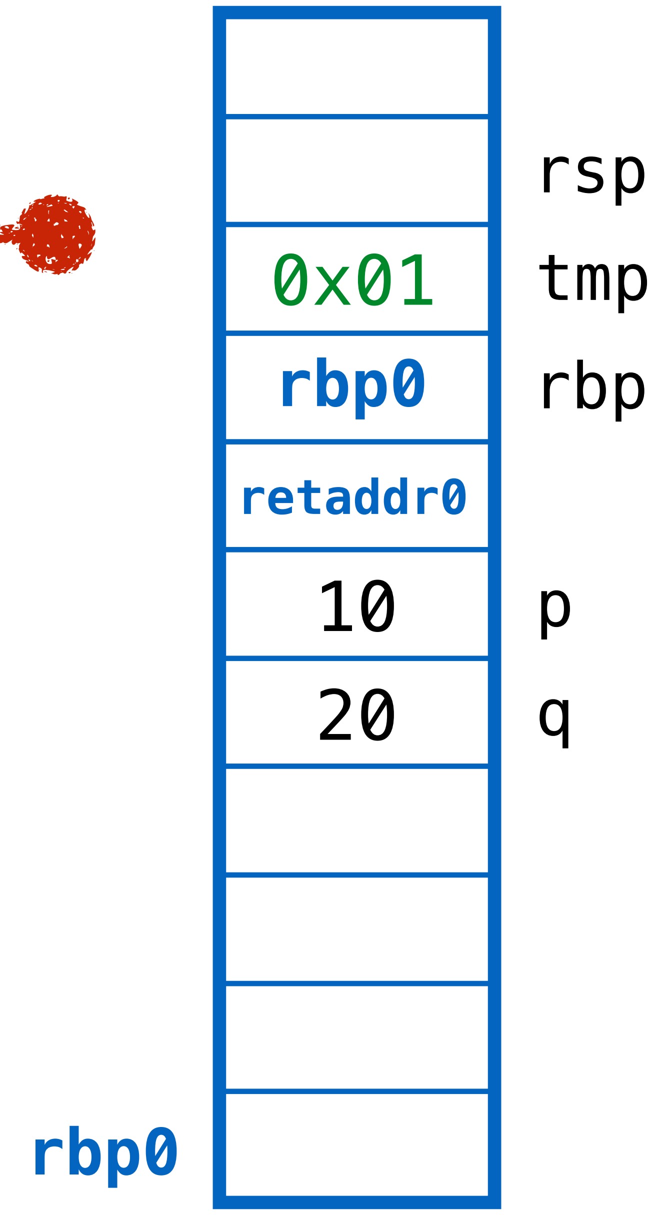
```
def foo(p, q):
  let tmp = (p, q)
  in tmp[0] + tmp[1]

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





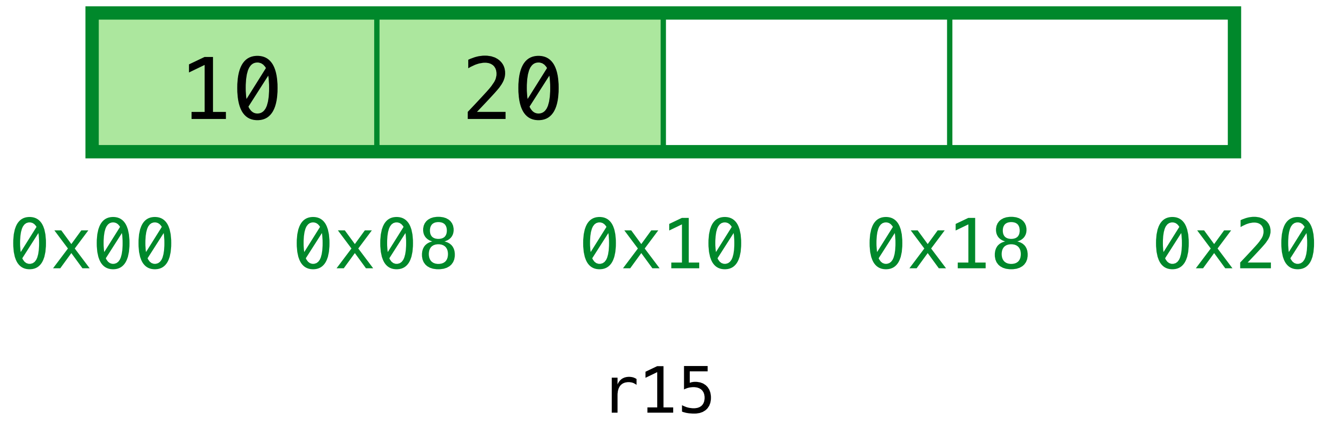
ex3: garbage in the middle (with stack)

```
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
```



	rsp
0x01	tmp
rbp0	rbp
retaddr0	
10	p
20	q

rbp0



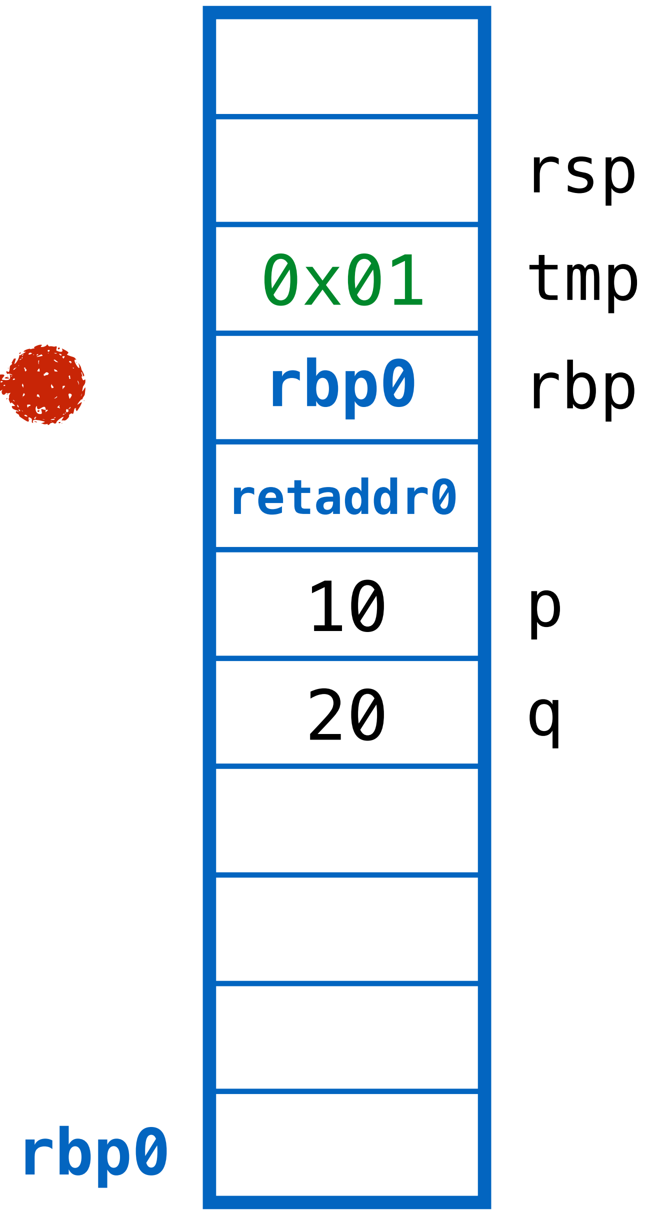
10	20		

0x00 0x08 0x10 0x18 0x20

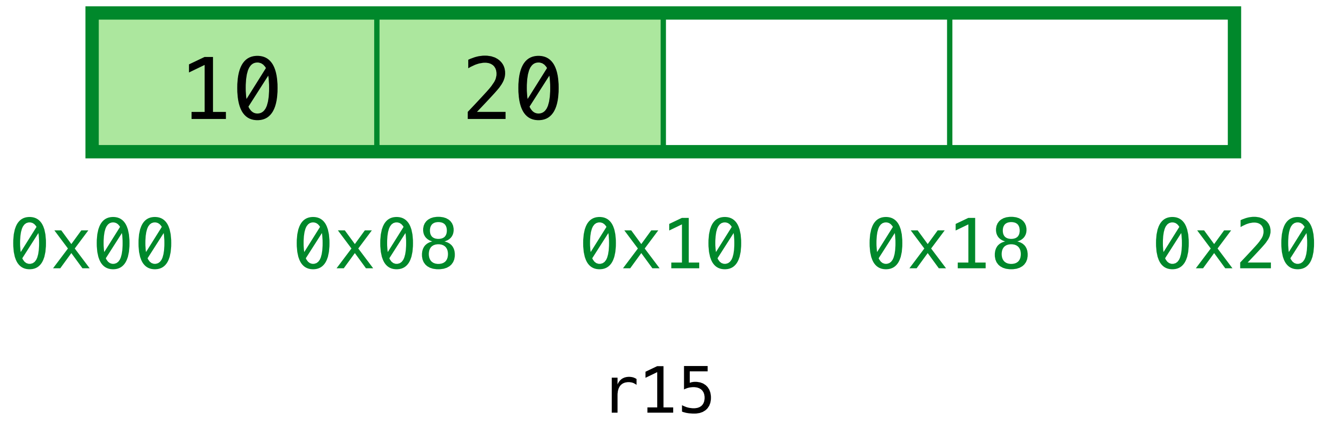
r15

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
```

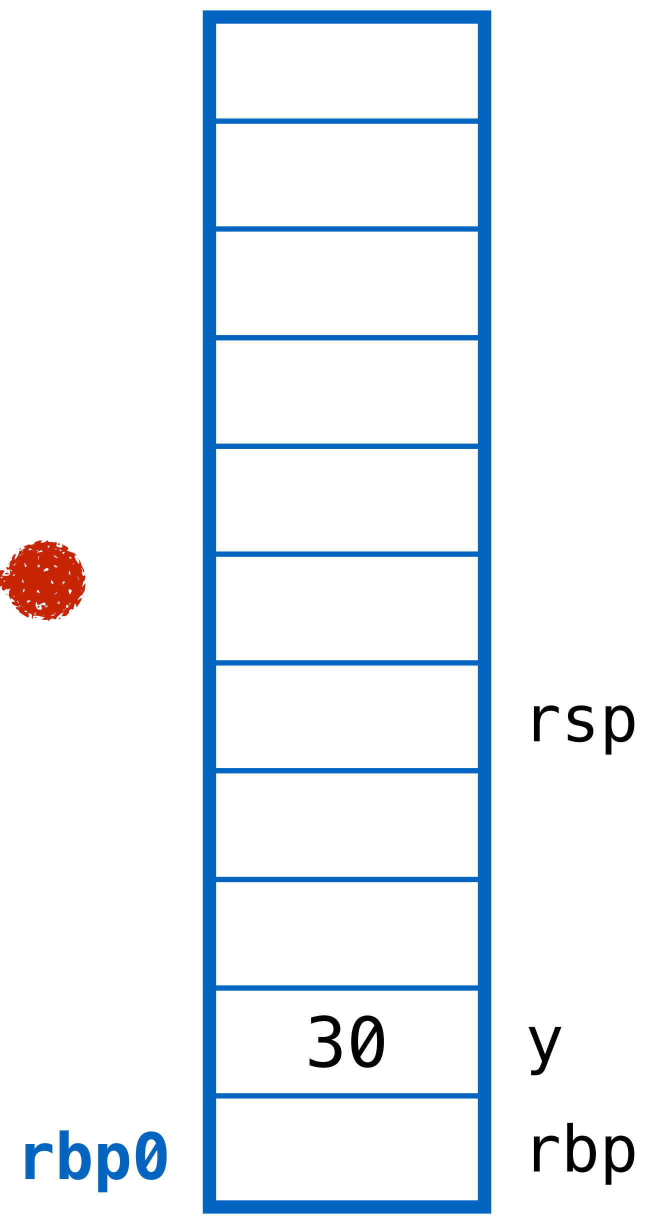


Return (rax) = 30

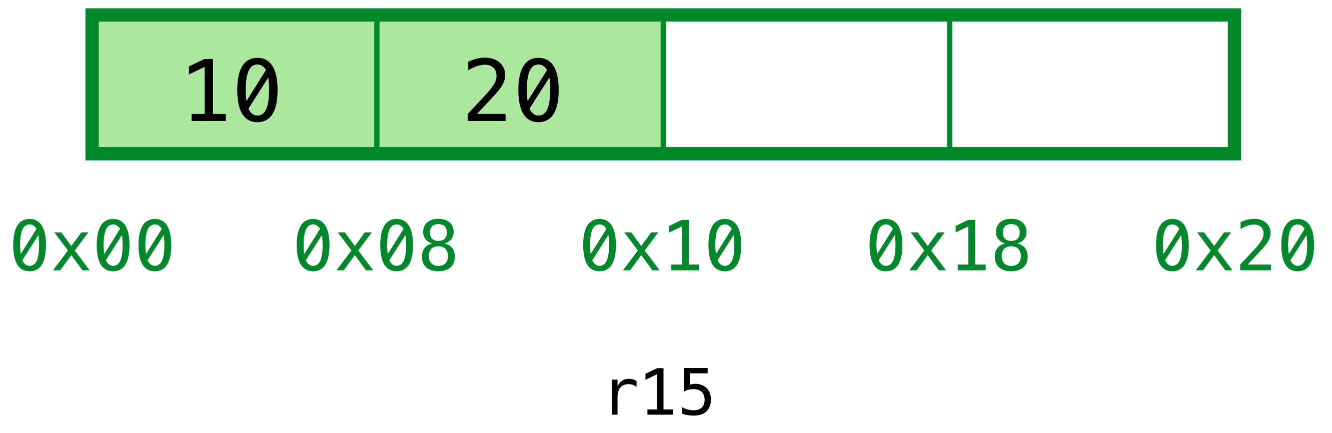


ex3: garbage in the middle (with stack)

```
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
```

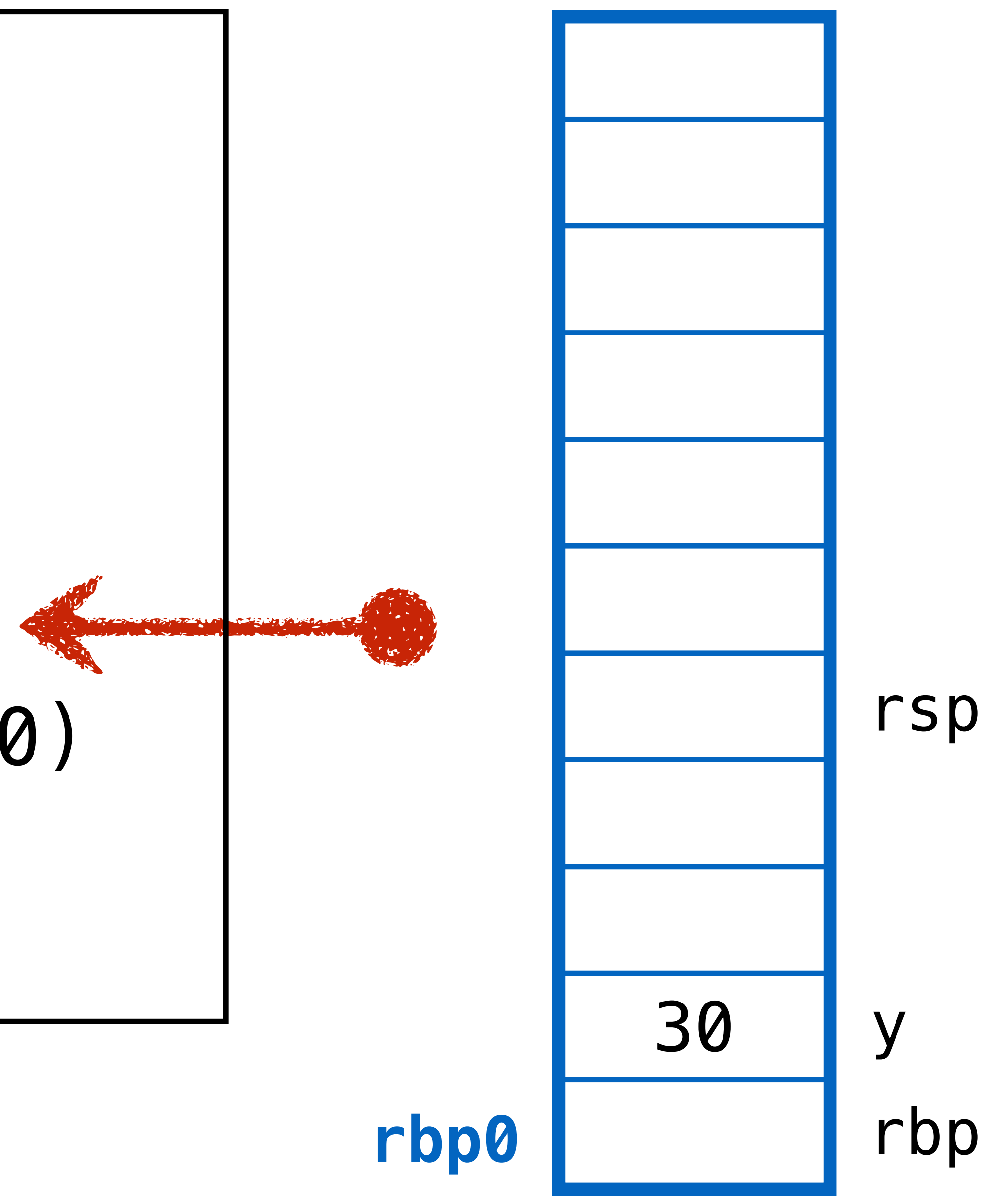


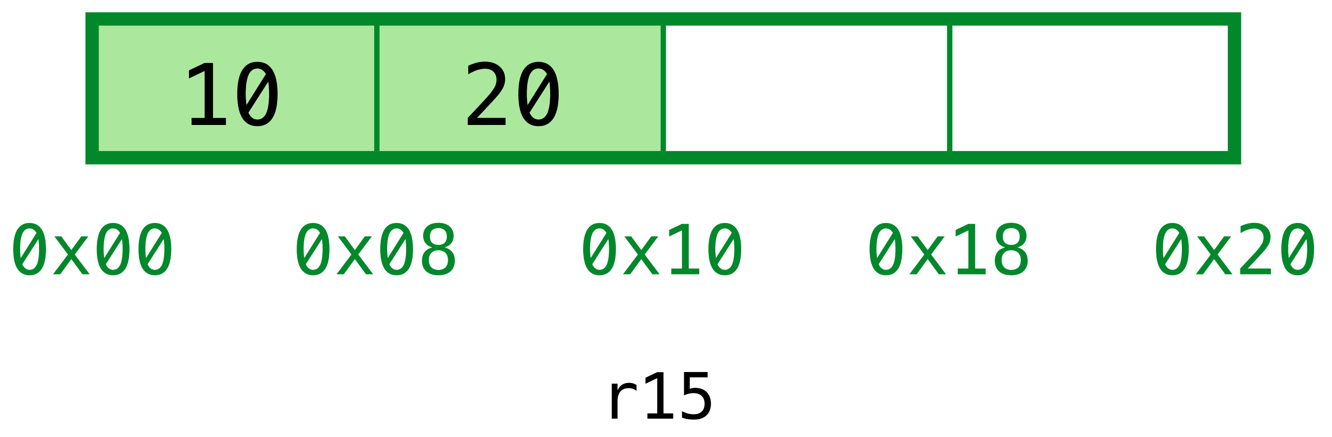
Return (rax) = 30



ex3: garbage in the middle (with stack)

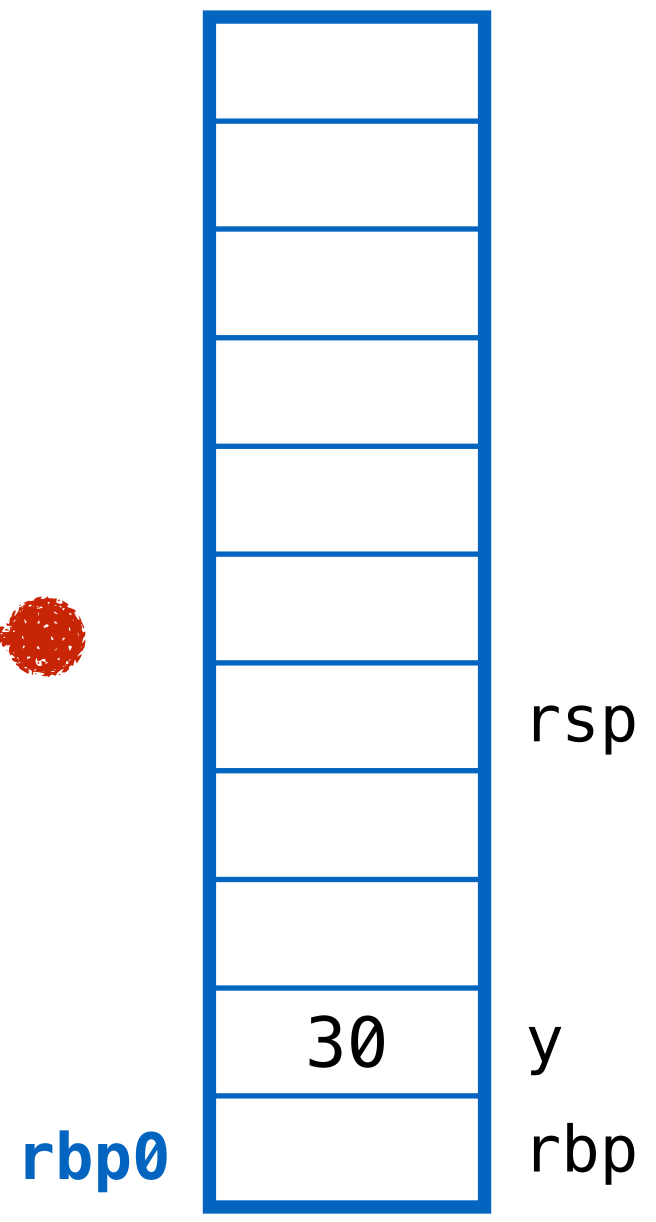
```
def foo(p, q):
  let tmp = (p, q)
  in tmp[0] + tmp[1]

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

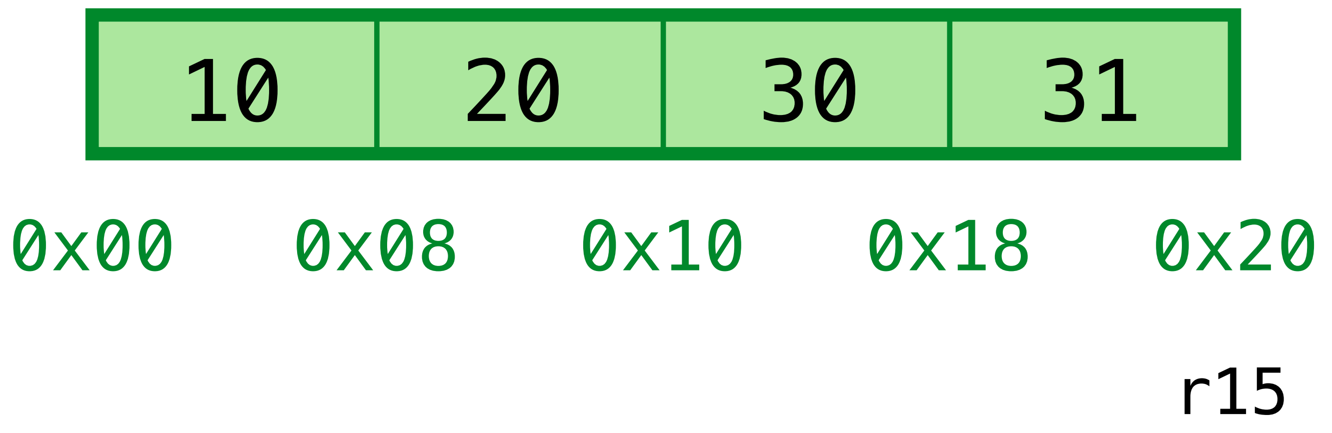




ex3: garbage in the middle (with stack)

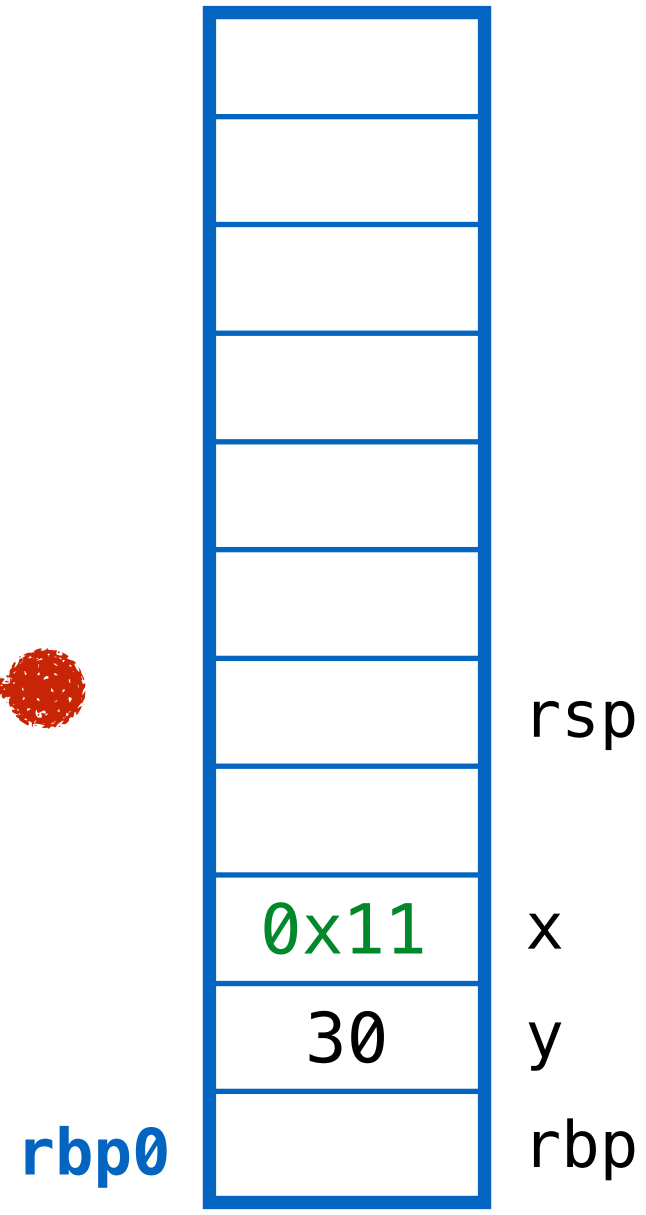
```
def foo(p, q):
  let tmp = (p, q)
  in tmp[0] + tmp[1]

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

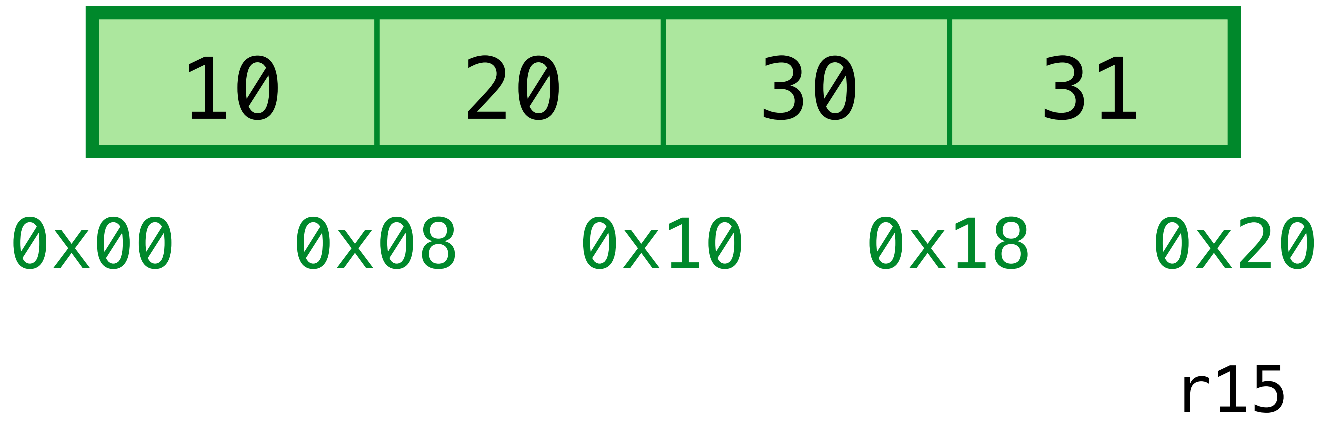




ex3: garbage in the middle (with stack)

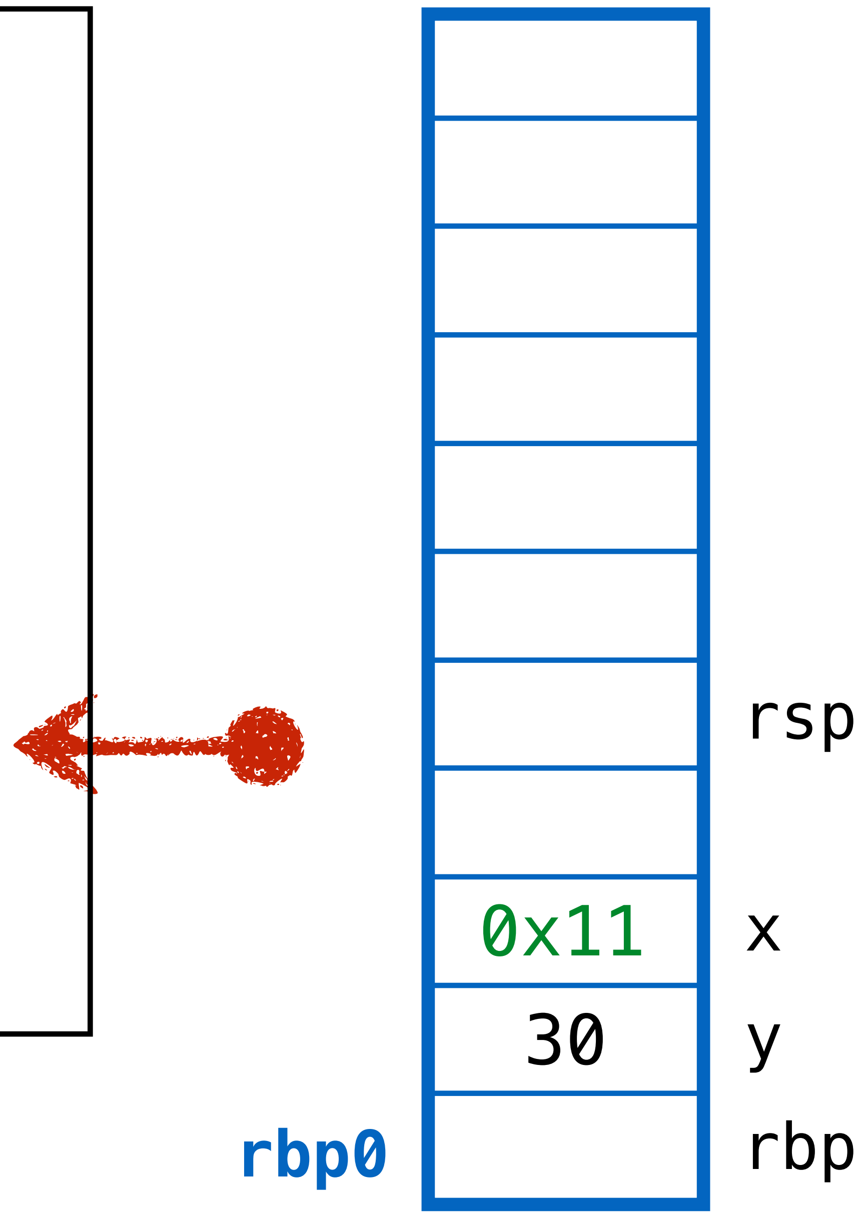
```
def foo(p, q):
  let tmp = (p, q)
  in tmp[0] + tmp[1]

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
```

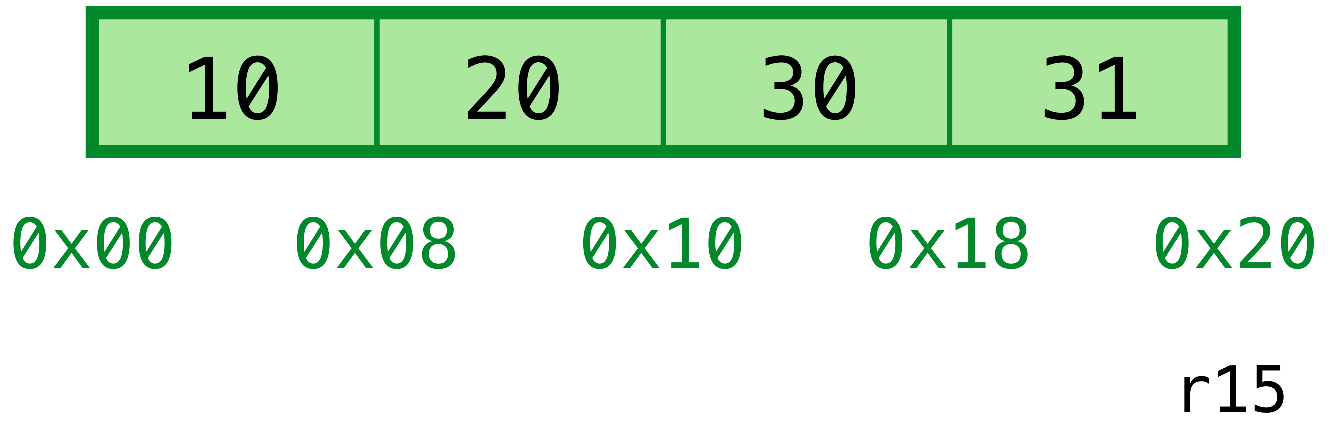


rsp

0x11 x

30 y

rbp0 rbp



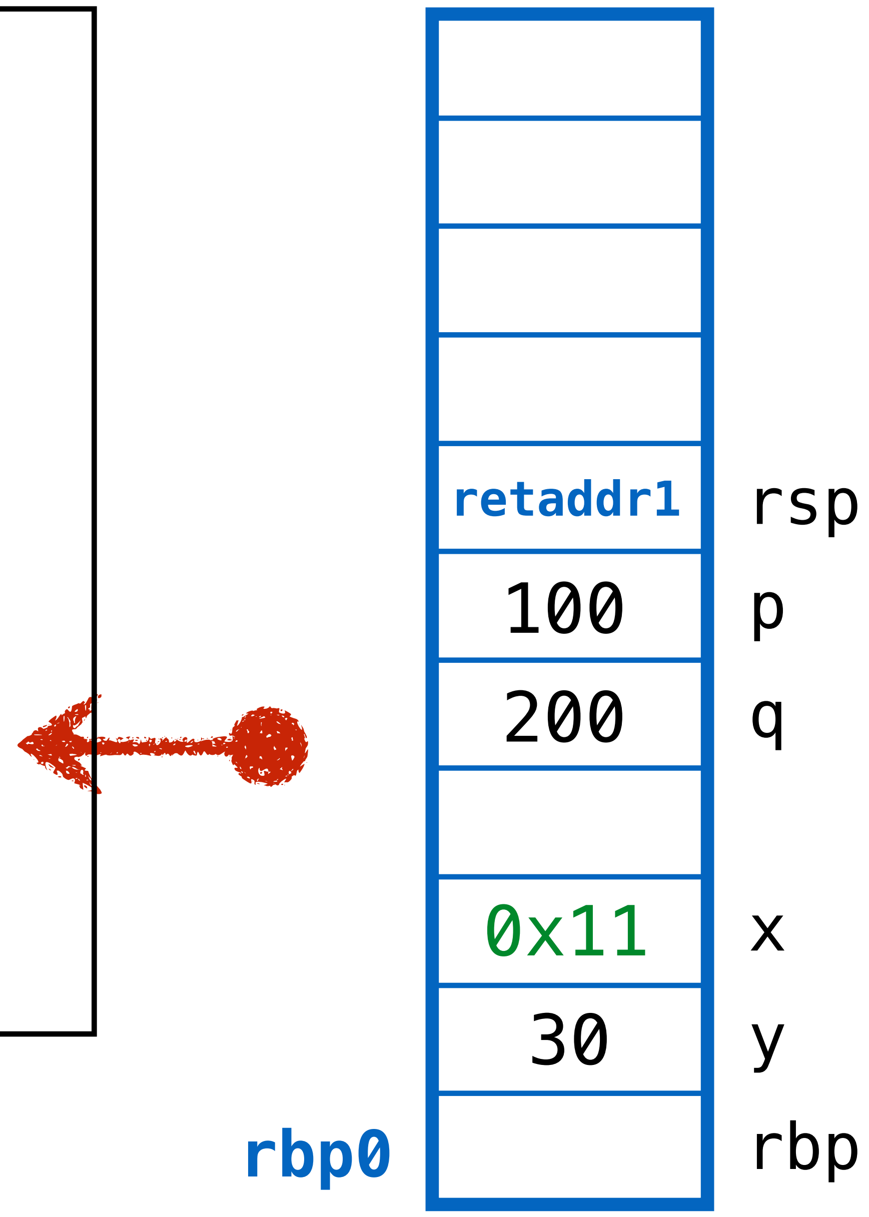
10 20 30 31

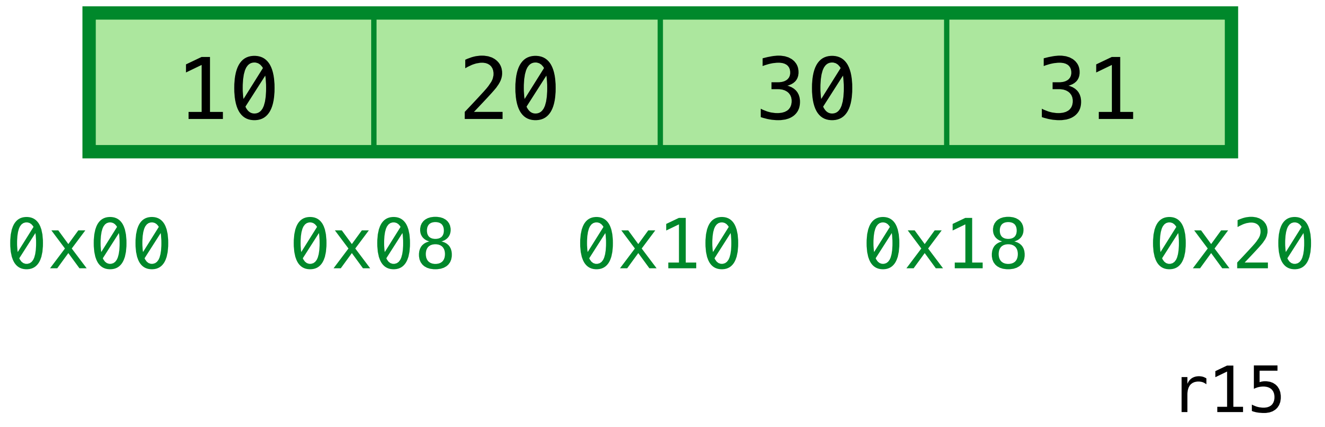
0x00 0x08 0x10 0x18 0x20

r15

ex3: garbage in the middle (with stack)

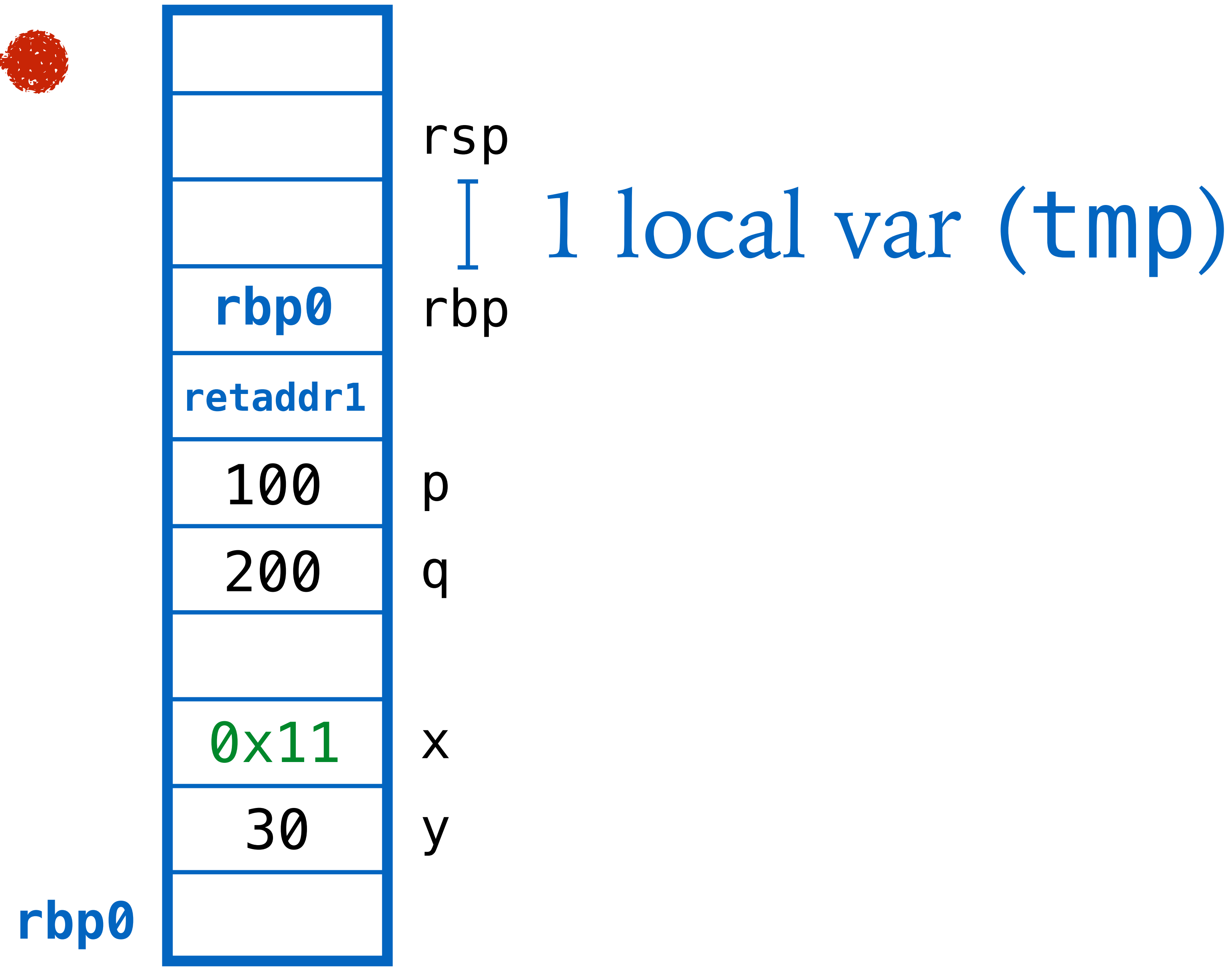
```
def foo(p, q):
  let tmp = (p, q)
  in tmp[0] + tmp[1]

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

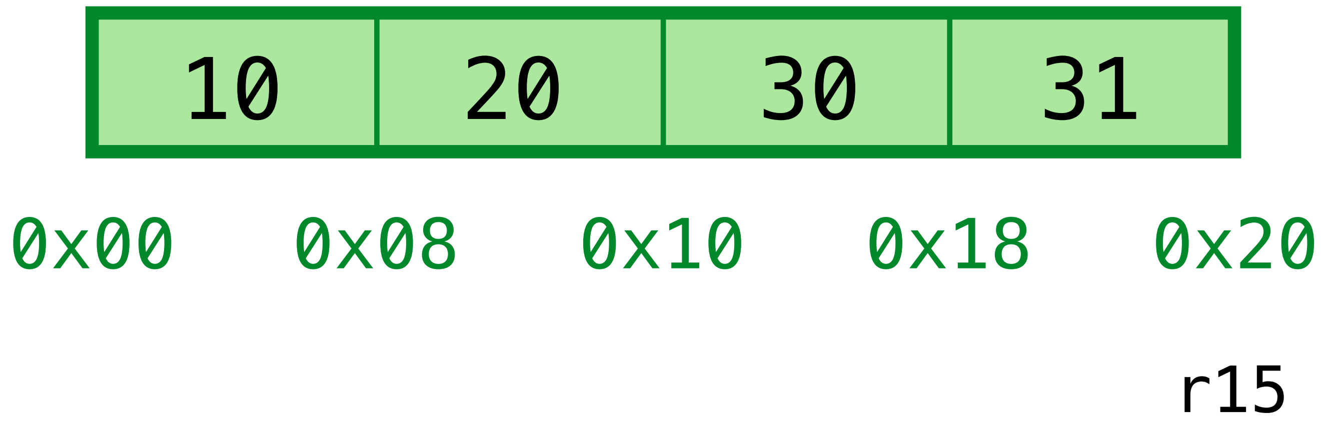




ex3: garbage in the middle (with stack)

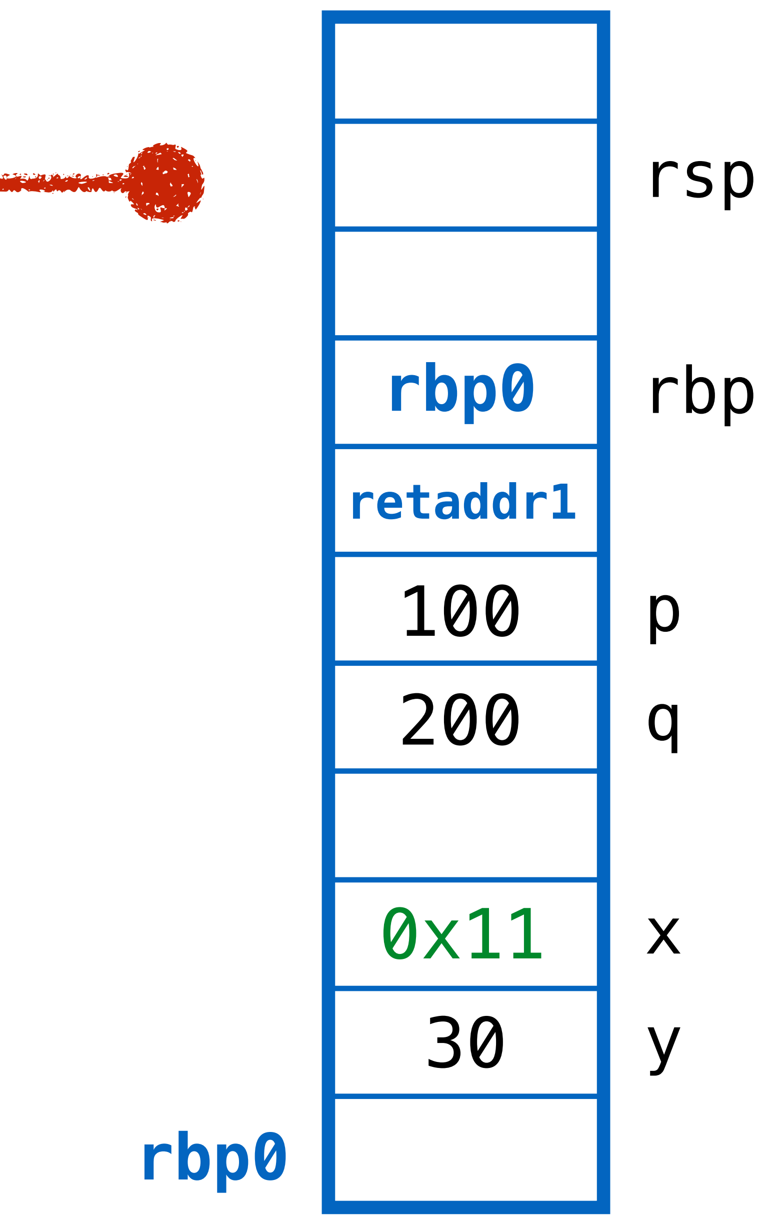
```
def foo(p, q):
  let tmp = (p, q)
  in tmp[0] + tmp[1]

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

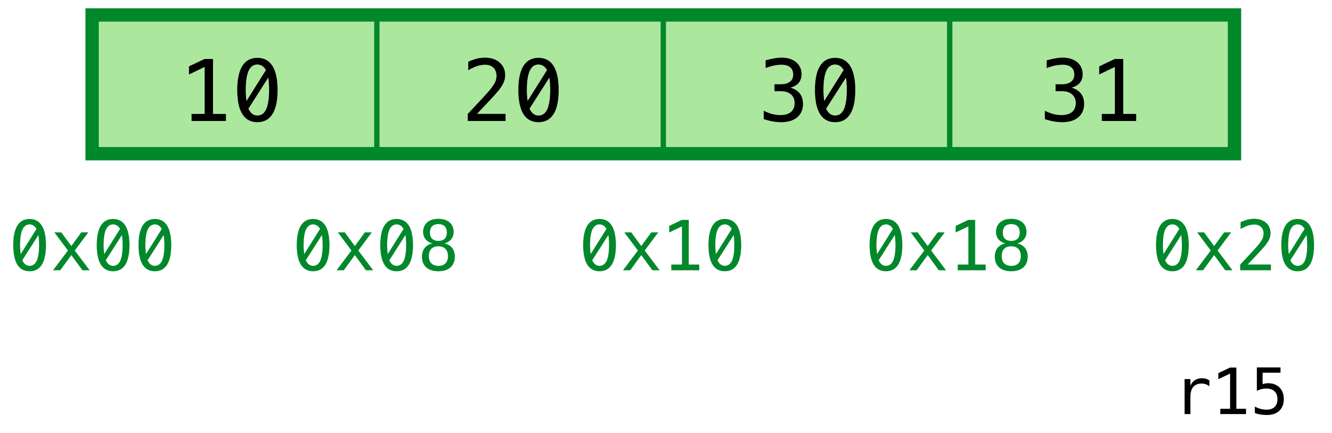




ex3: garbage in the middle (with stack)

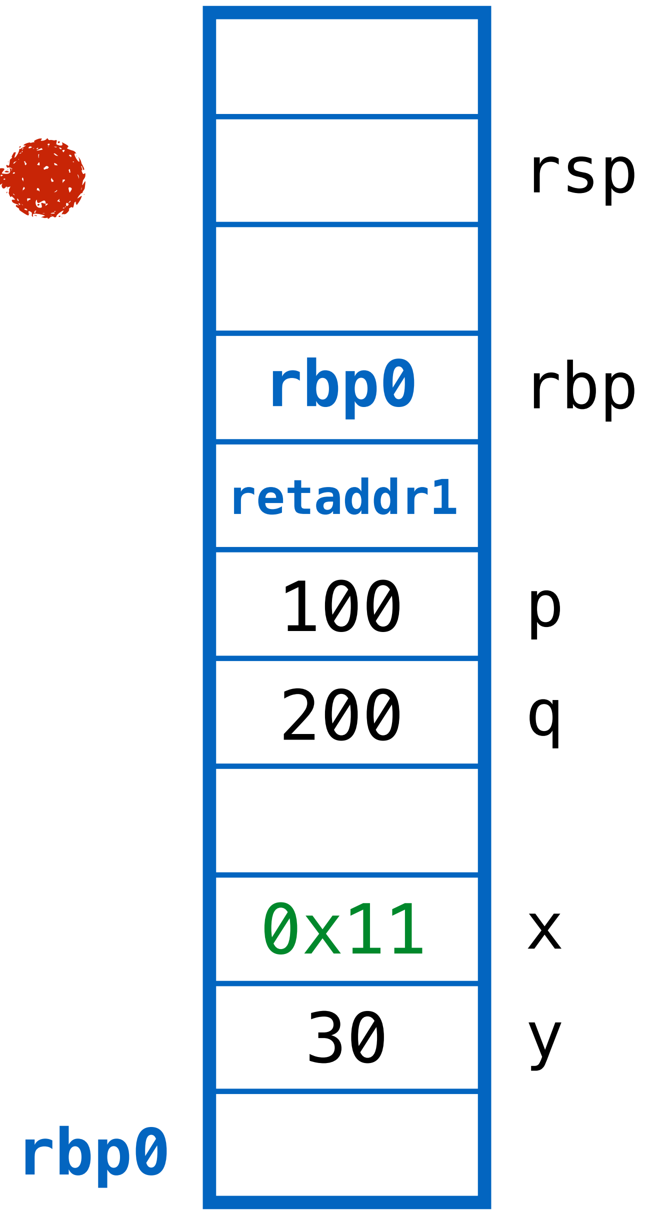
```
def foo(p, q):
  let tmp = (p, q)
  in tmp[0] + tmp[1]

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



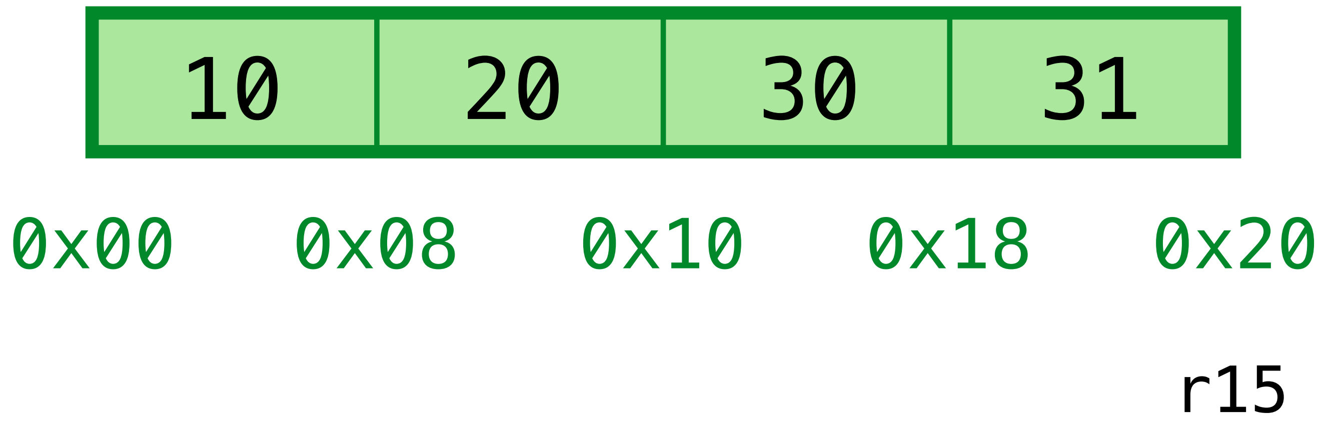


ex3: garbage in the middle (with stack)

```
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
```

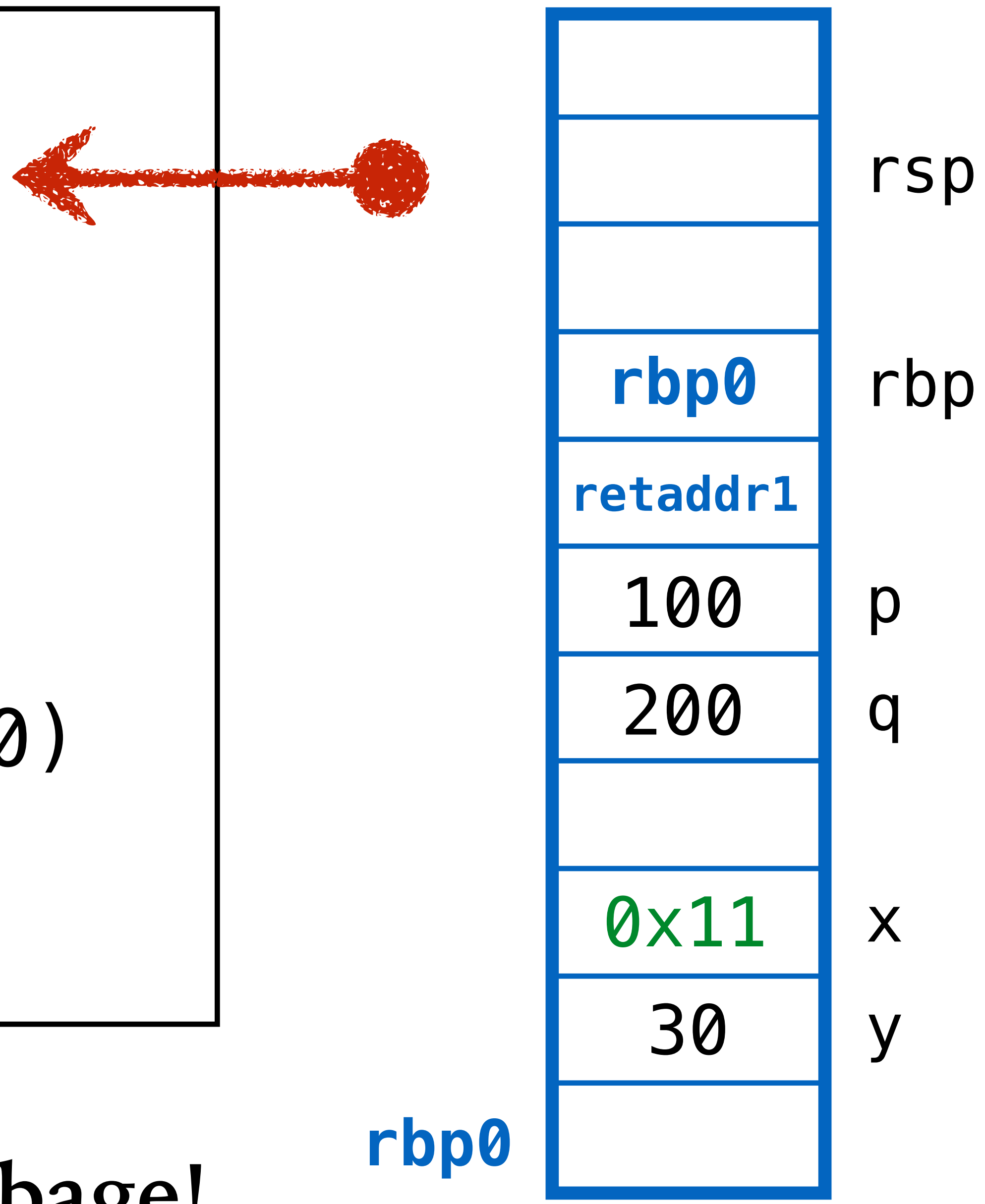


Lets reclaim & recycle garbage!

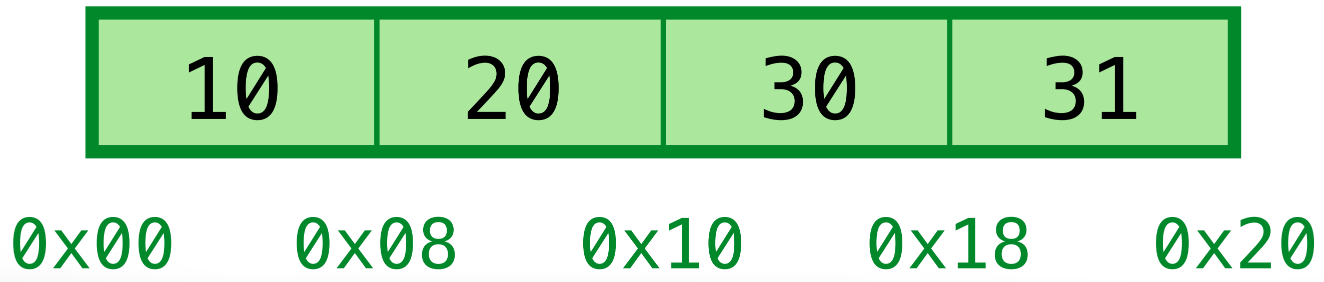


ex3: garbage in the middle (with stack)

```
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!

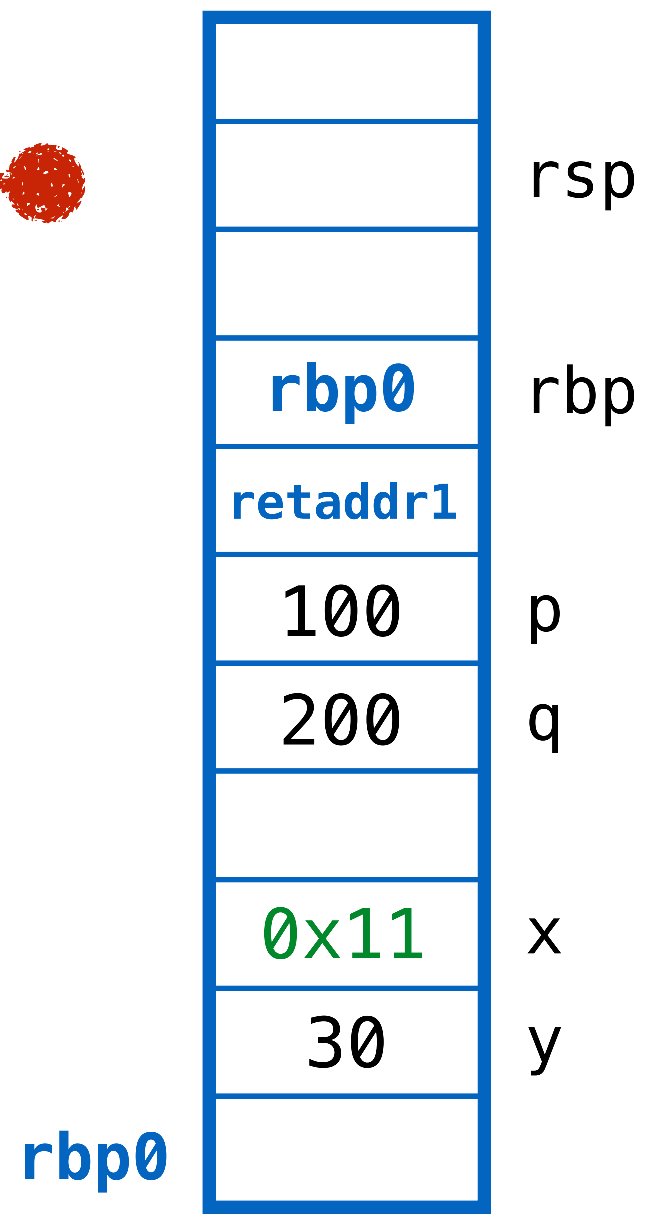


QUIZ: Which cells are garbage?

(A) 0x00, 0x08 (B) 0x08, 0x10 (C) 0x10, 0x18 (D) None (E) All

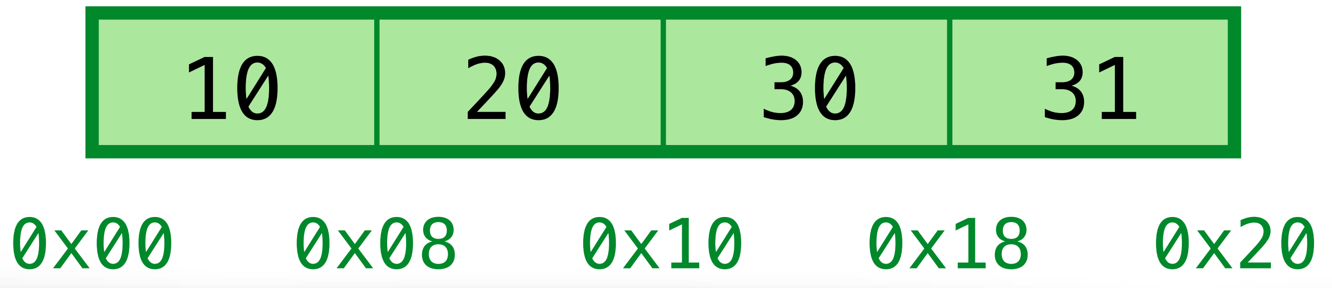
ex3: garbage in the middle (with stack)

```
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
```



	rsp
rbp0	rbp
retaddr1	
100	p
200	q
0x11	x
30	y

Lets reclaim & recycle garbage!



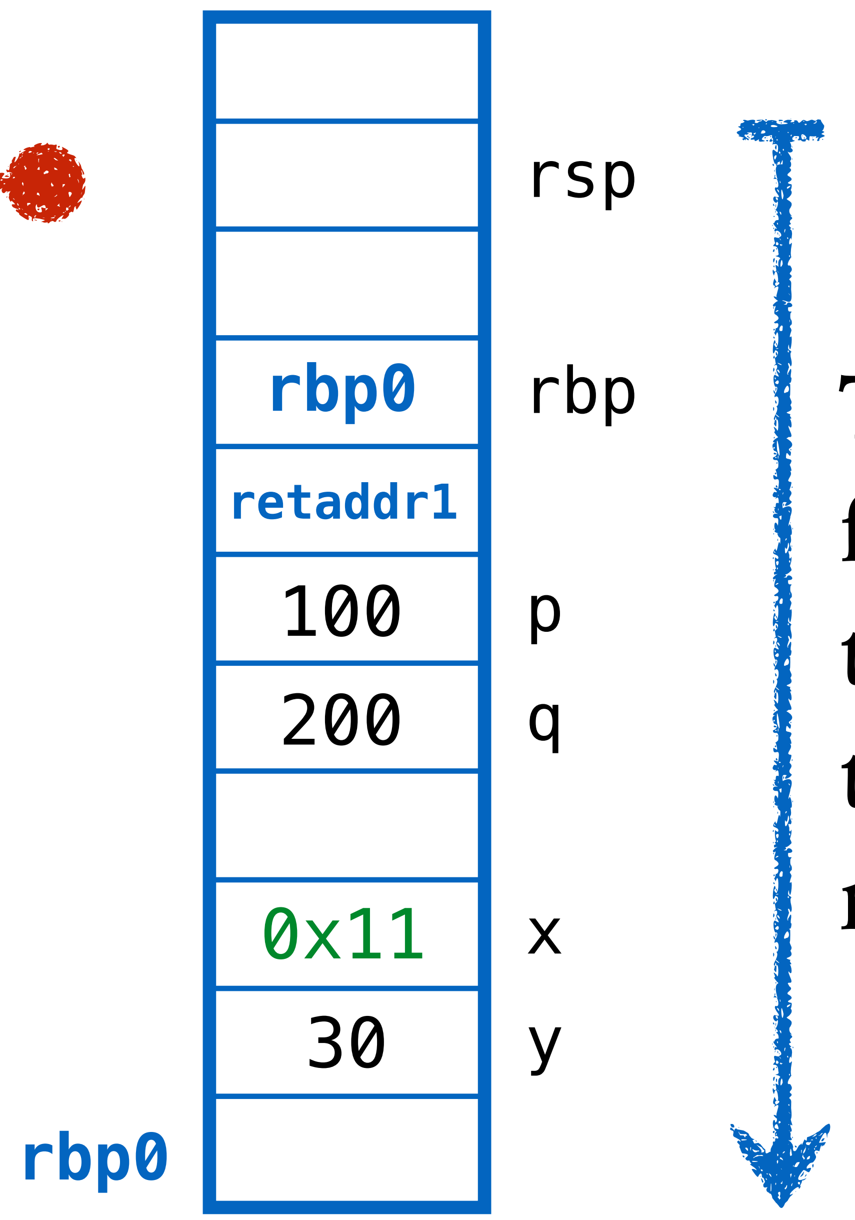
10	20	30	31

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 (`rsp`)
to bottom (`rbp0`)
to mark
reachable cells.

Lets reclaim & recycle garbage!

10	20	30	31

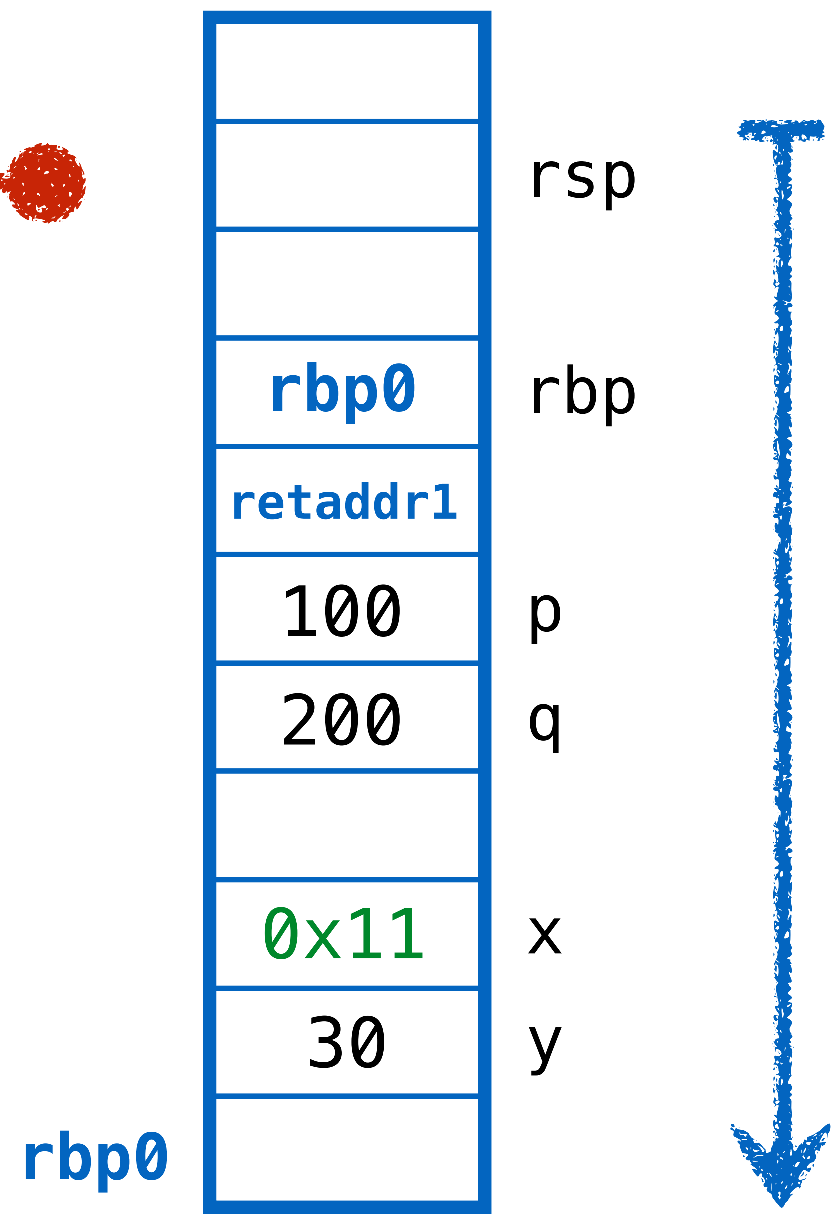
`0x00` `0x08` `0x10` `0x18` `0x20`

QUIZ: Which cells are garbage?

Those that are *not reachable from any stack frame*

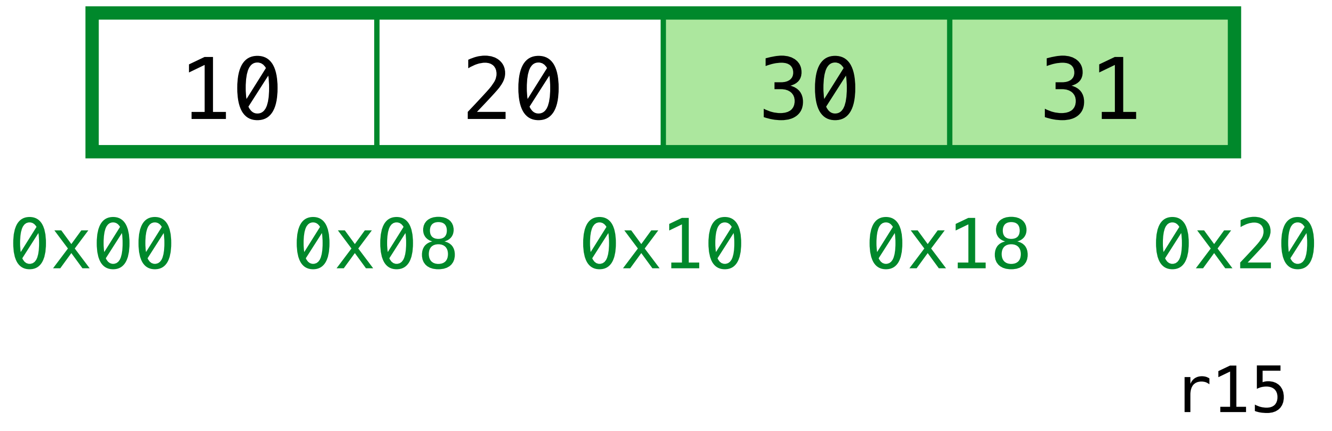
ex3: garbage in the middle (with stack)

```
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 (`rsp`)
to bottom (`rbp0`)
to mark
reachable cells.

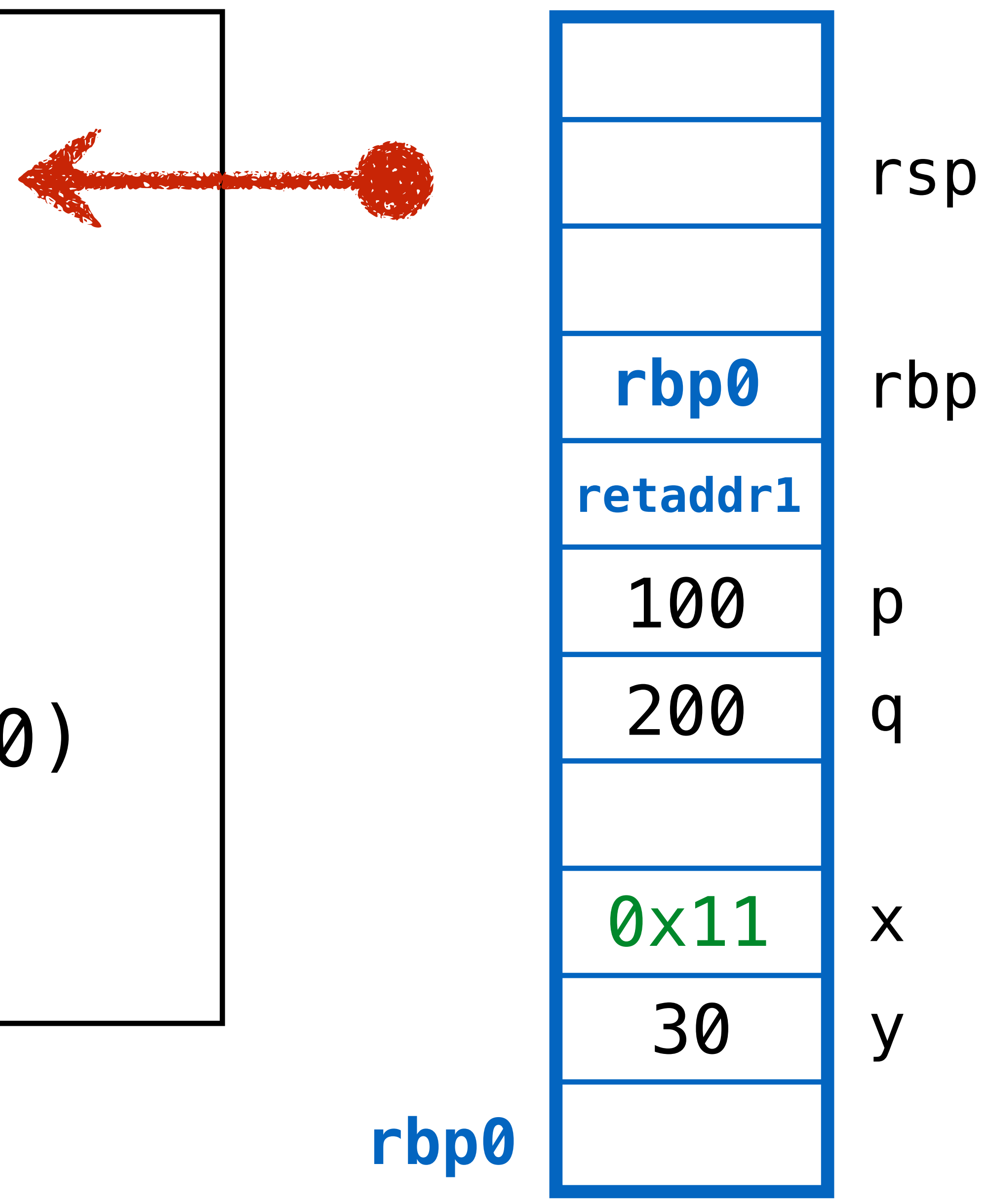
Lets reclaim & recycle garbage!



Which cells are garbage?

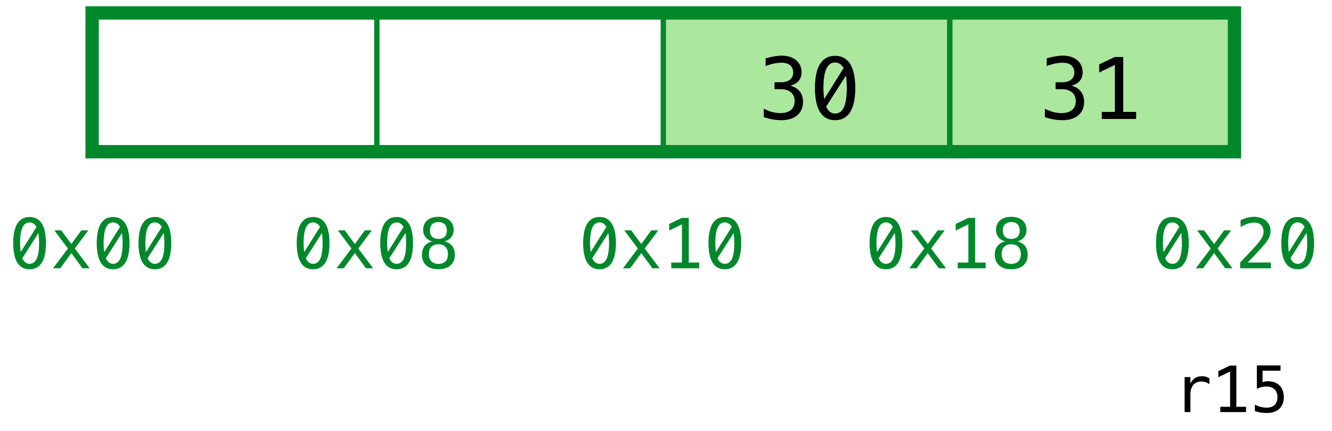
ex3: garbage in the middle (with stack)

```
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
```



	rsp
rbp0	rbp
retaddr1	
100	p
200	q
0x11	x
30	y

rbp0



		30	31

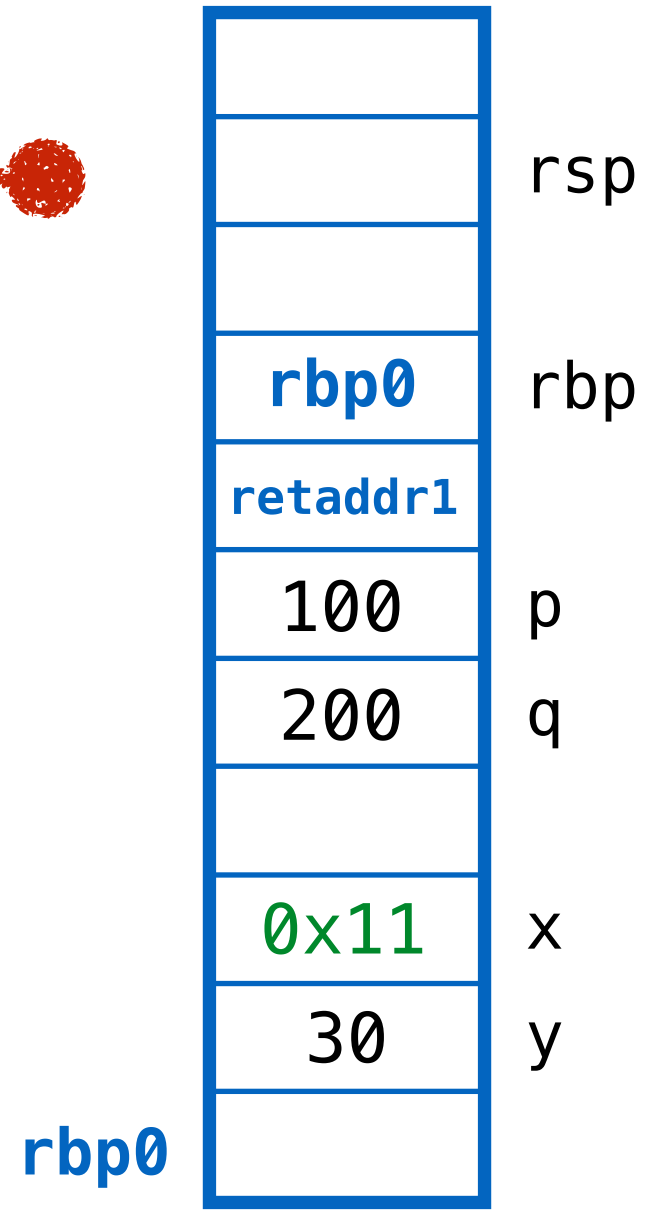
0x00 0x08 0x10 0x18 0x20

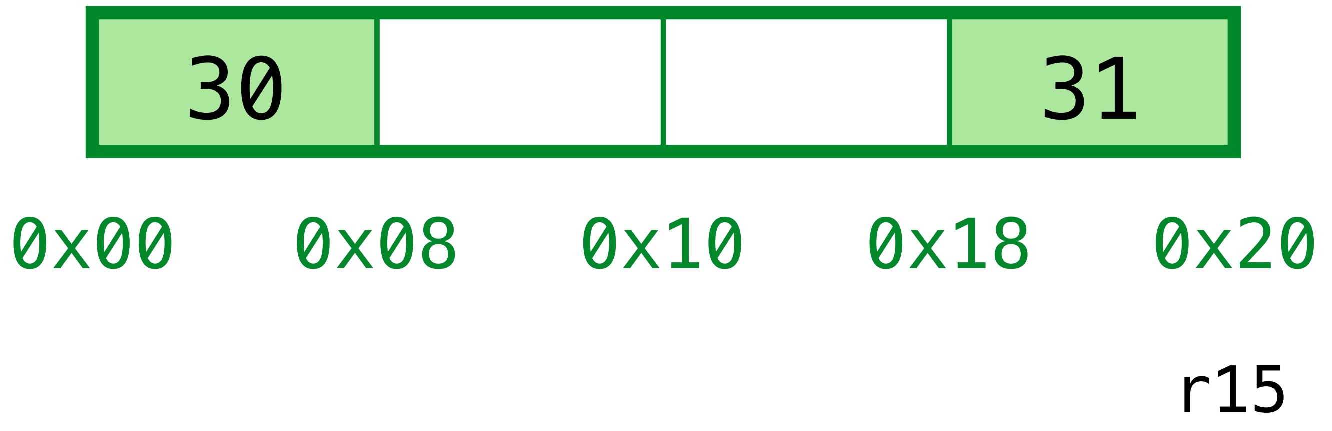
r15

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
```

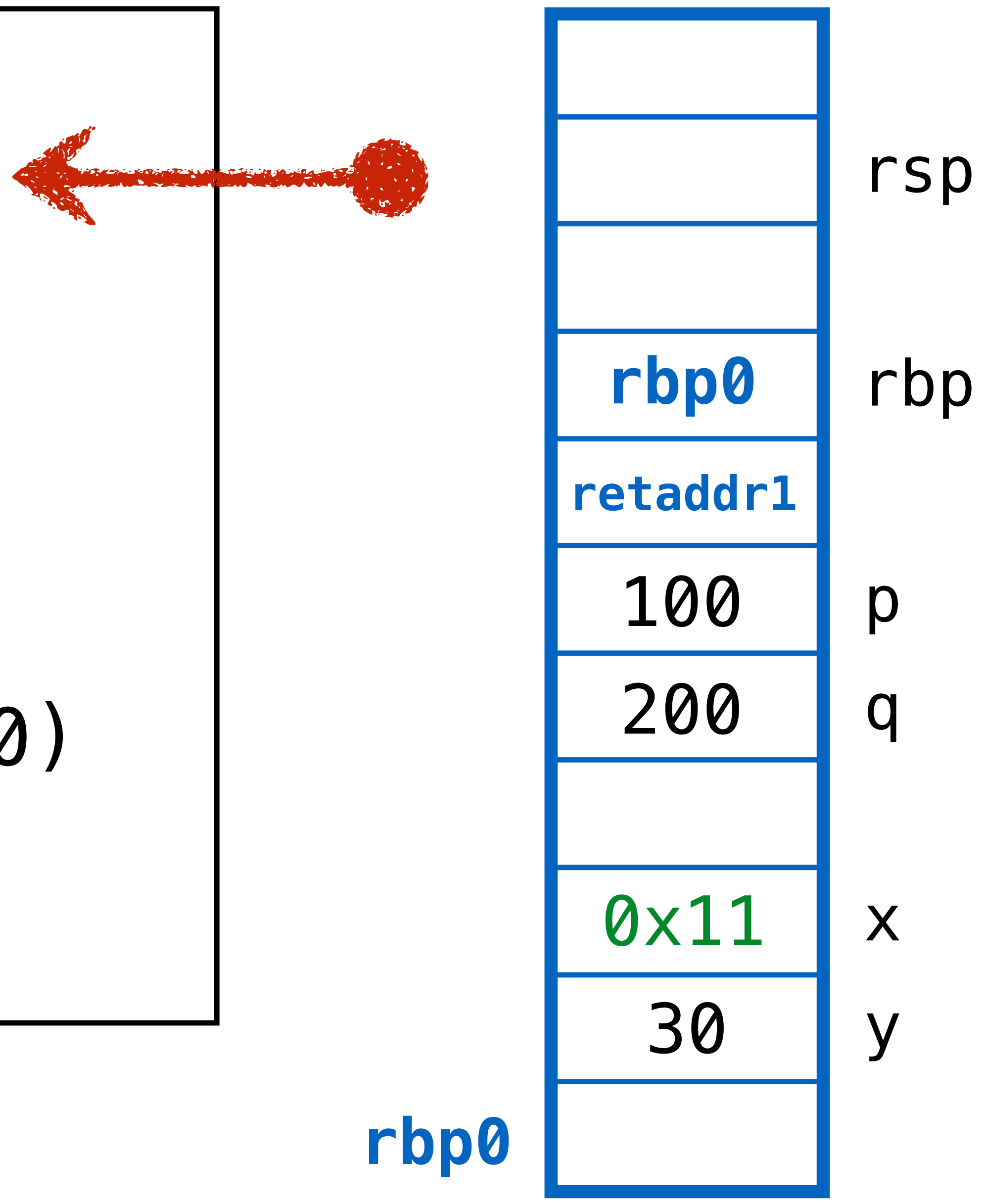


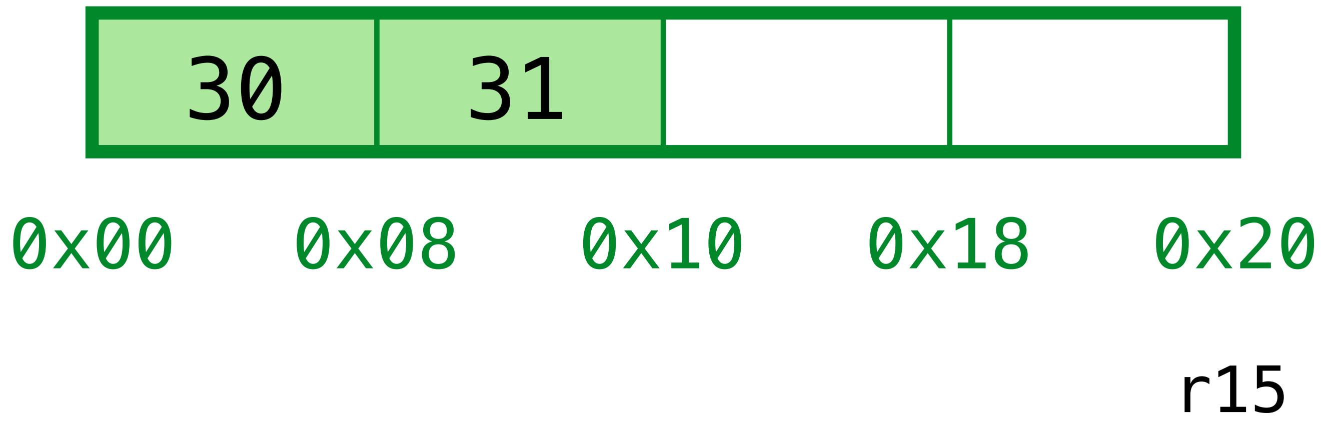


Compact the live cells

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
```

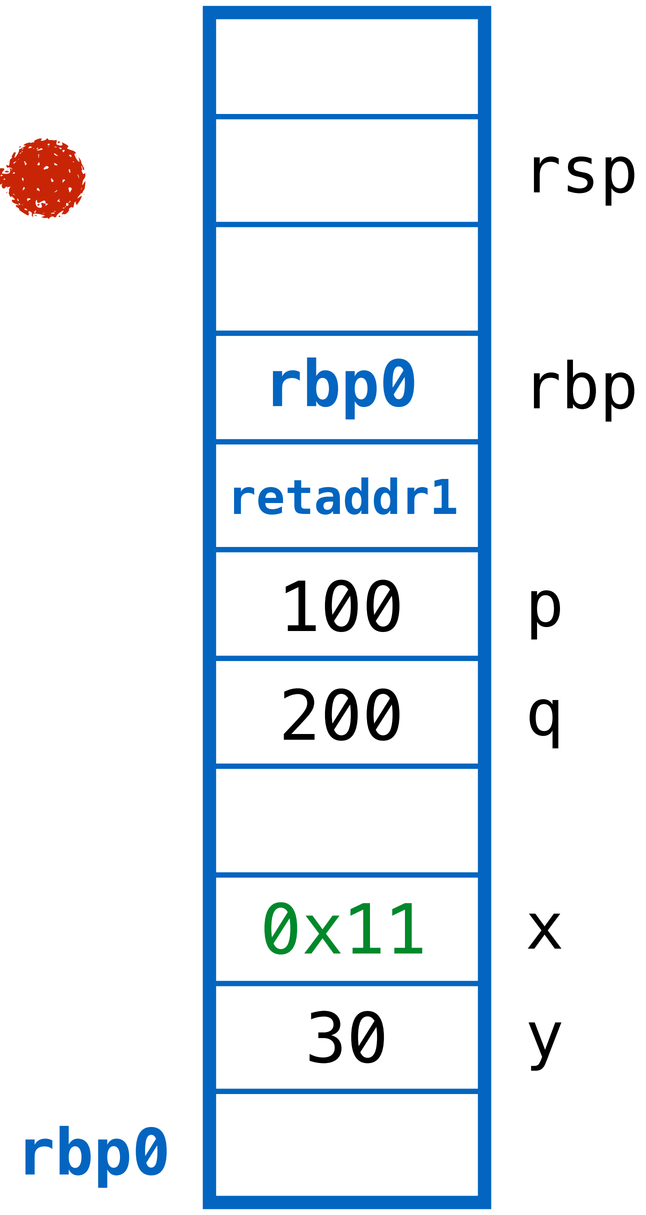


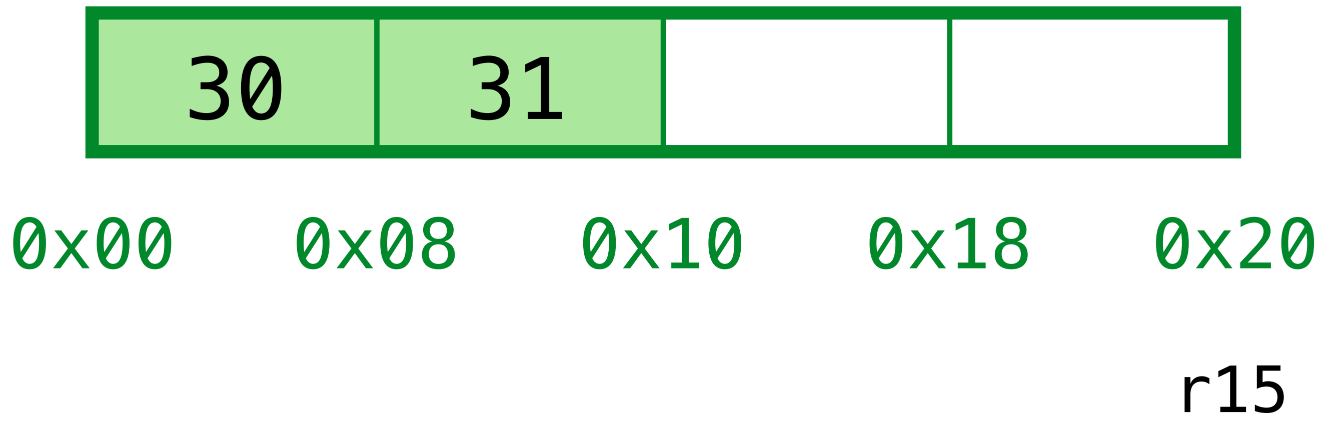


Compact the live cells

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
```

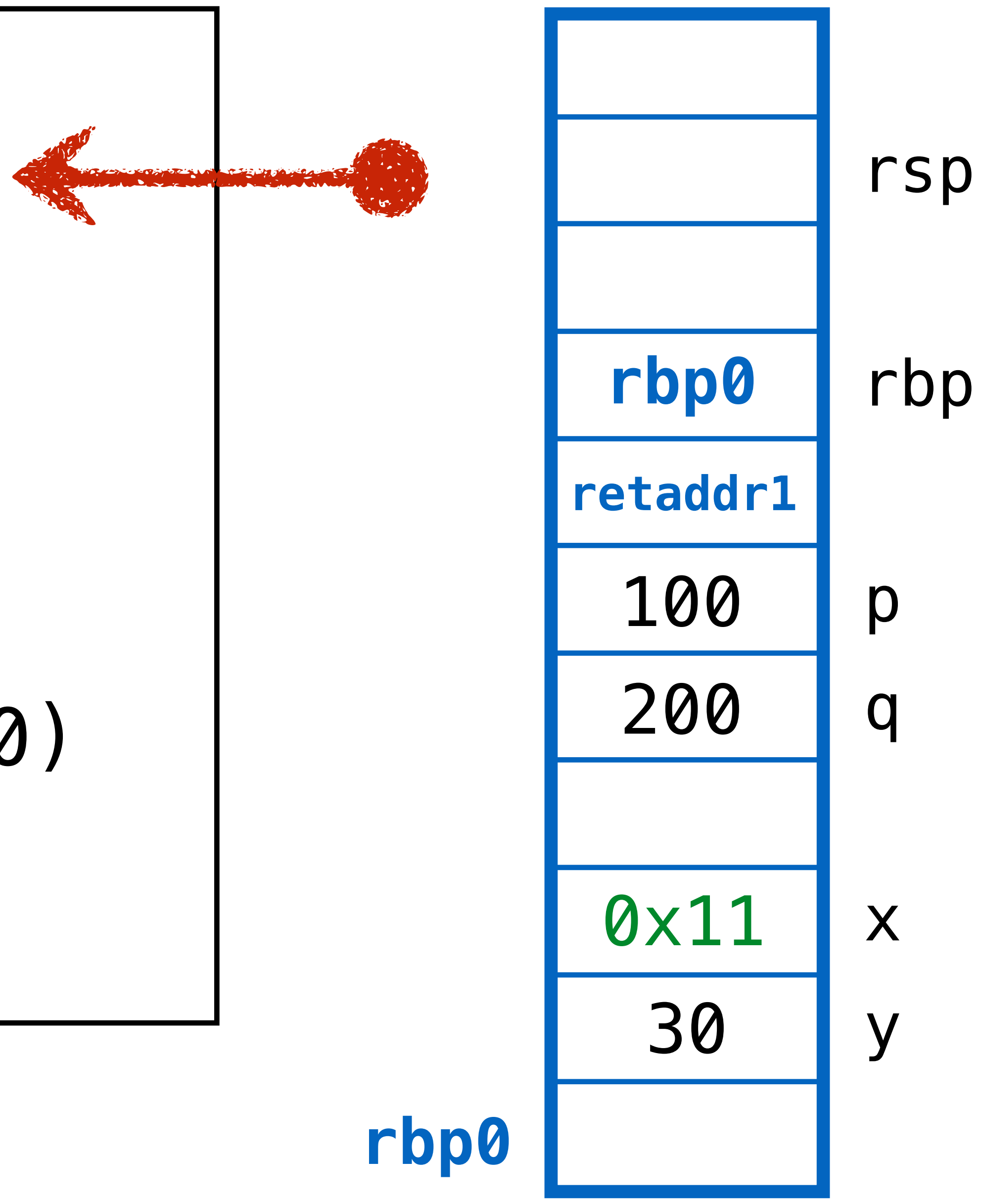


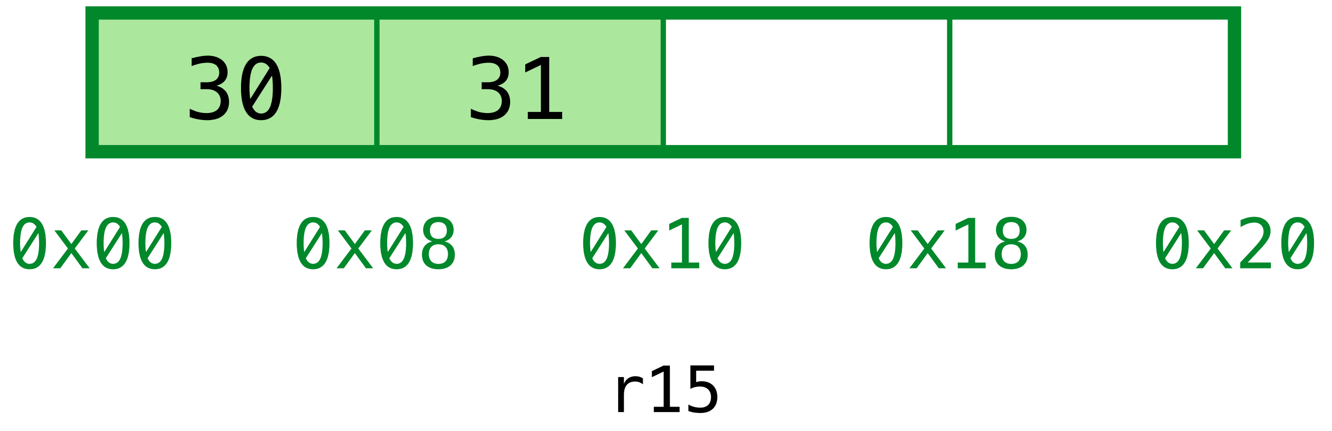


Compact the live cells … then rewind `r15`

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
```

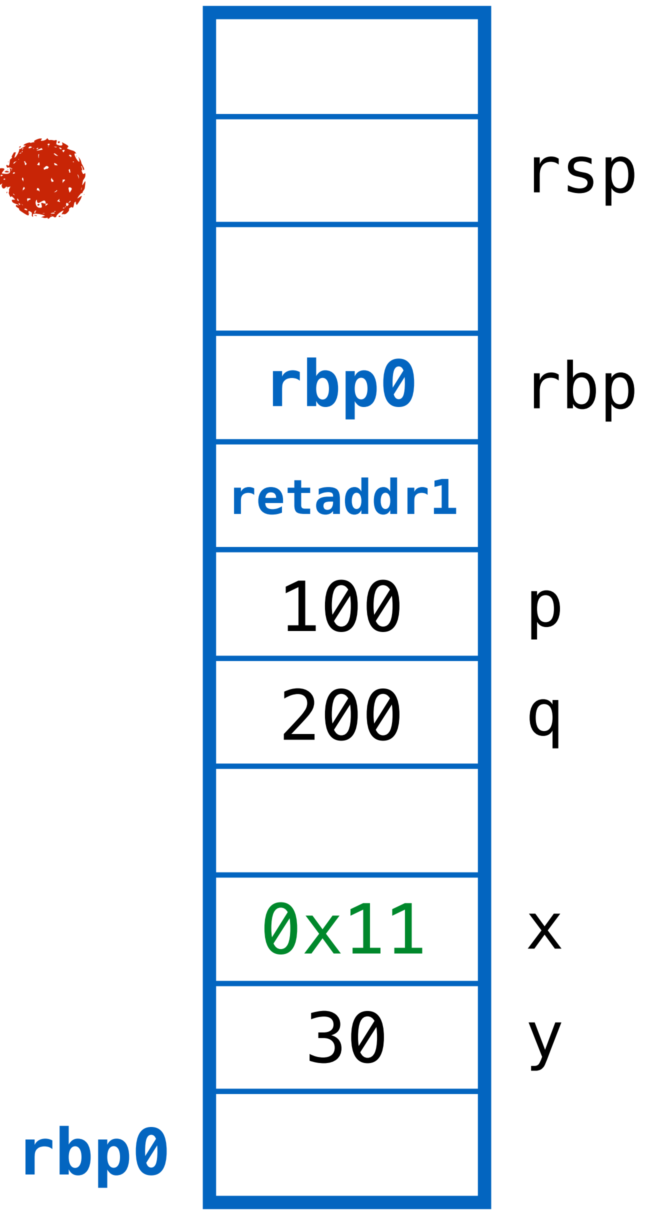


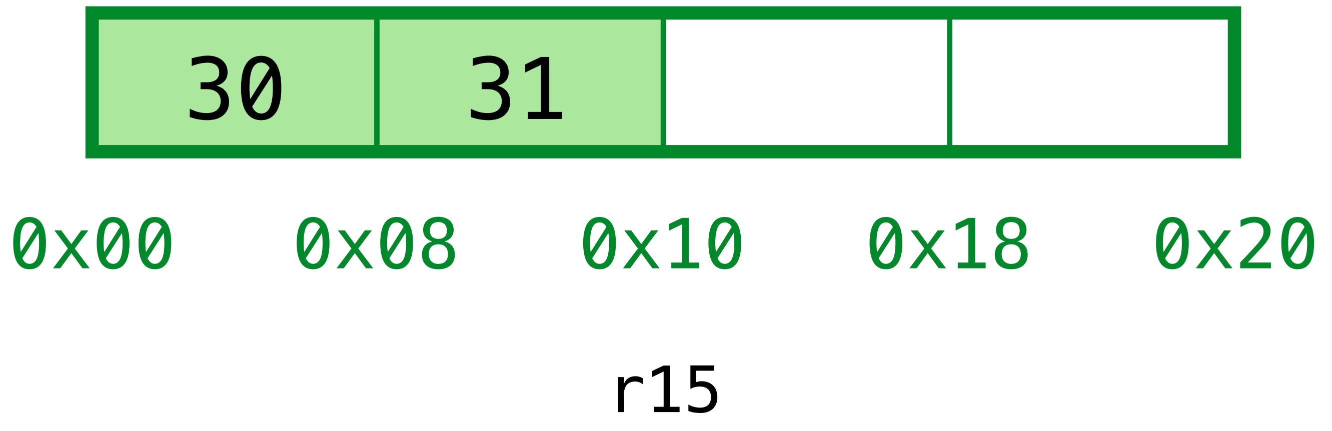


Compact the live cells … then rewind `r15`

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
```

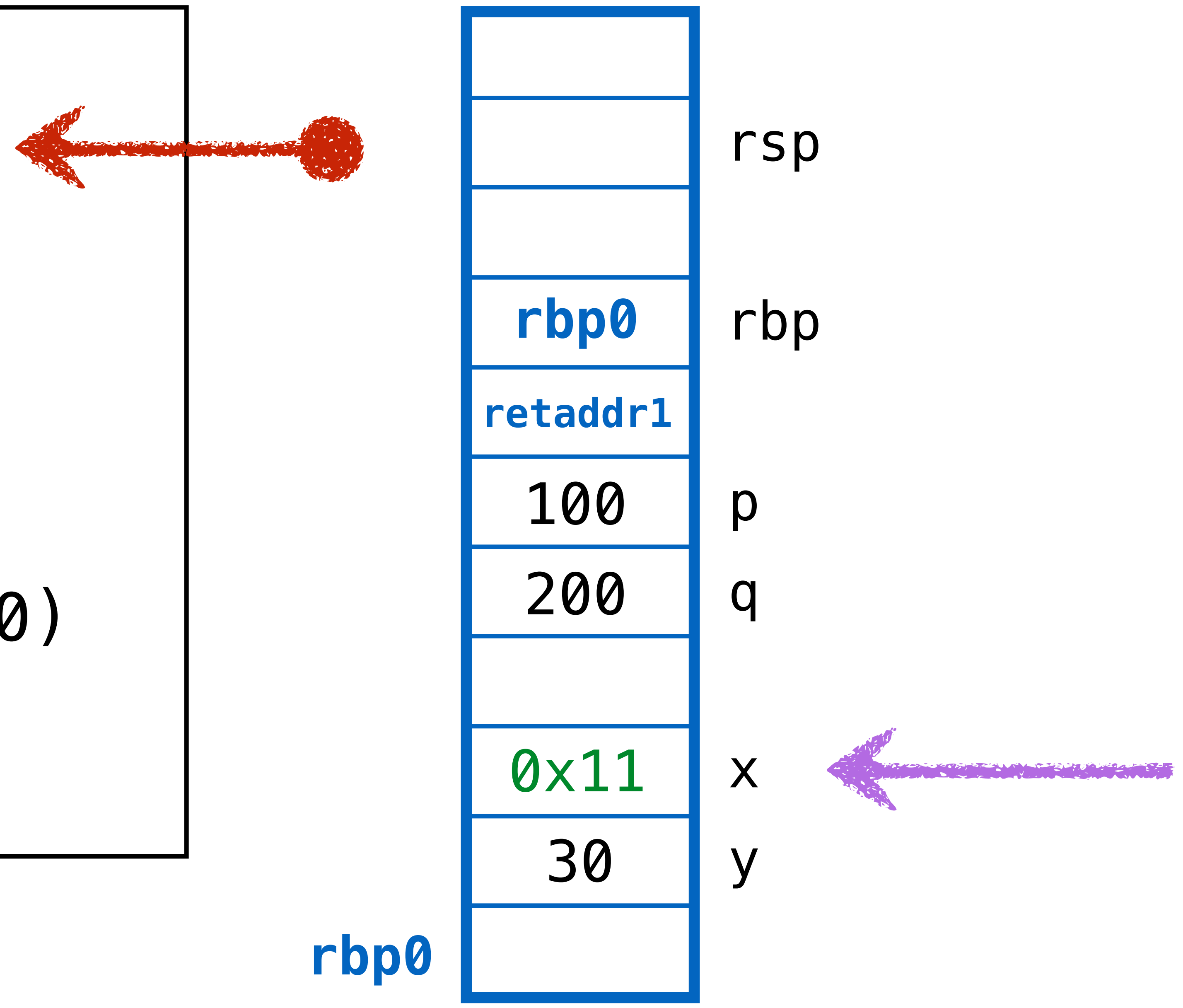


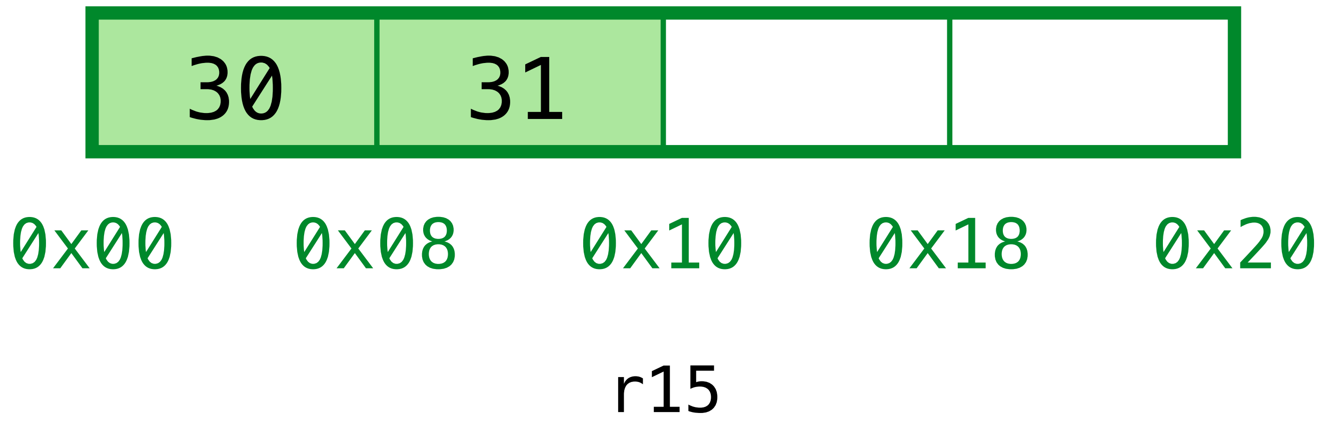


Problem???

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
```

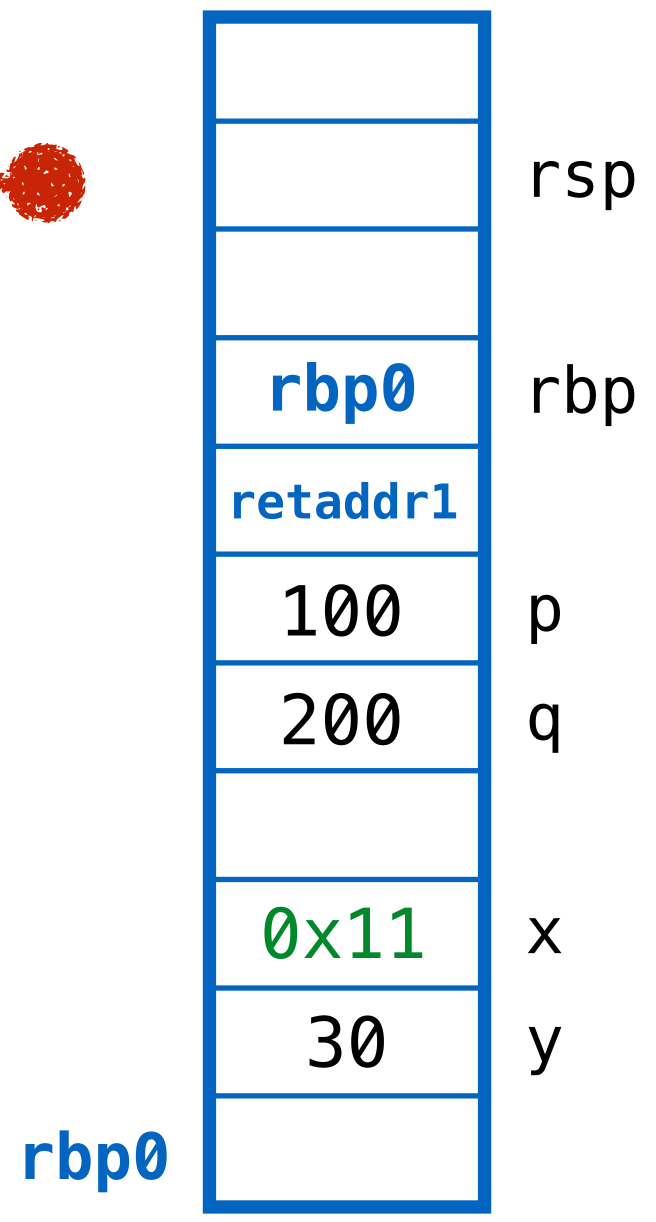


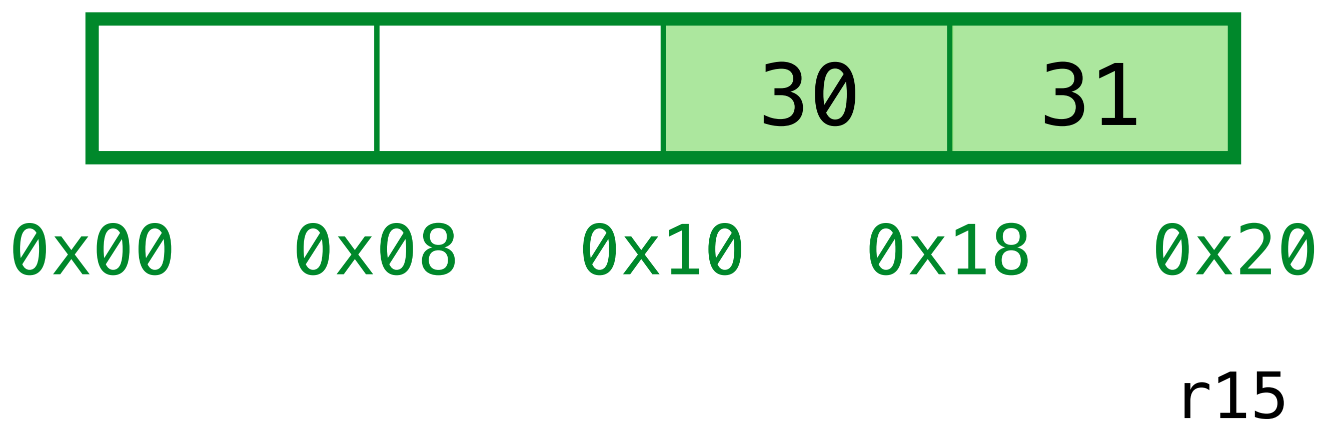


Problem! Have to REDIRECT existing pointers

ex3: garbage in the middle (with stack)

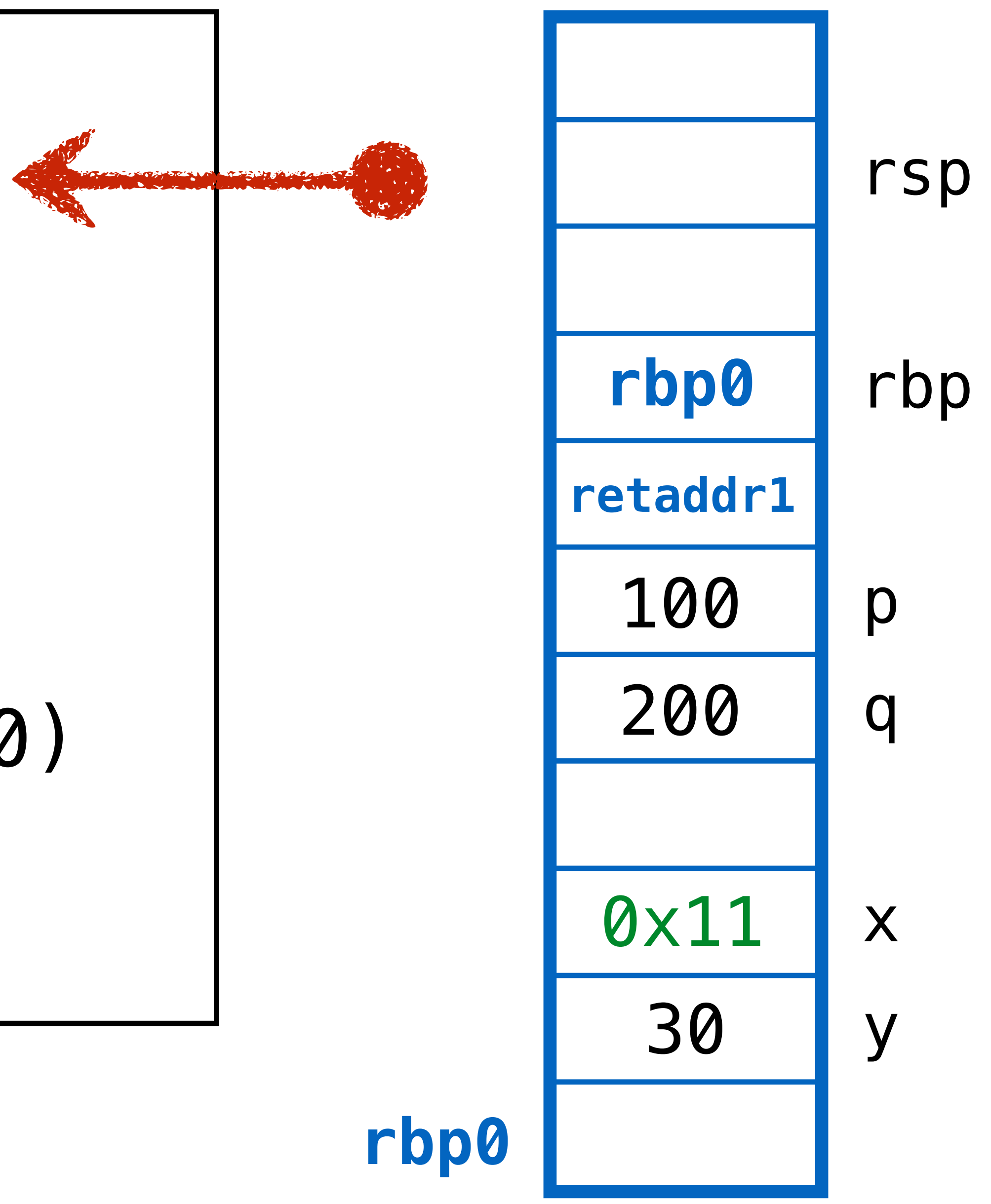
```
def foo(p, q):
  let tmp = (p, q)
  in tmp[0] + tmp[1]

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

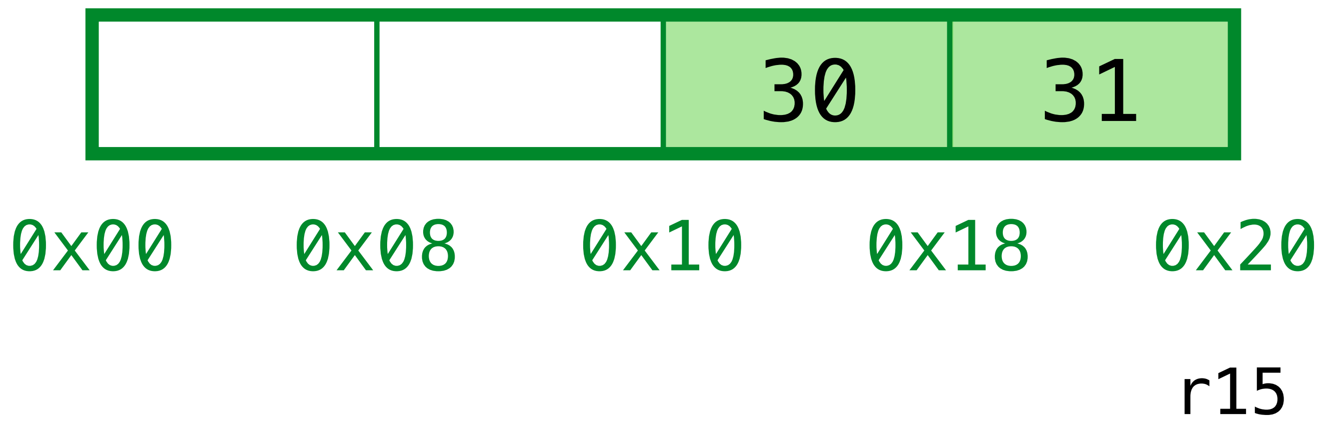




ex3: garbage in the middle (with stack)

```
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
```

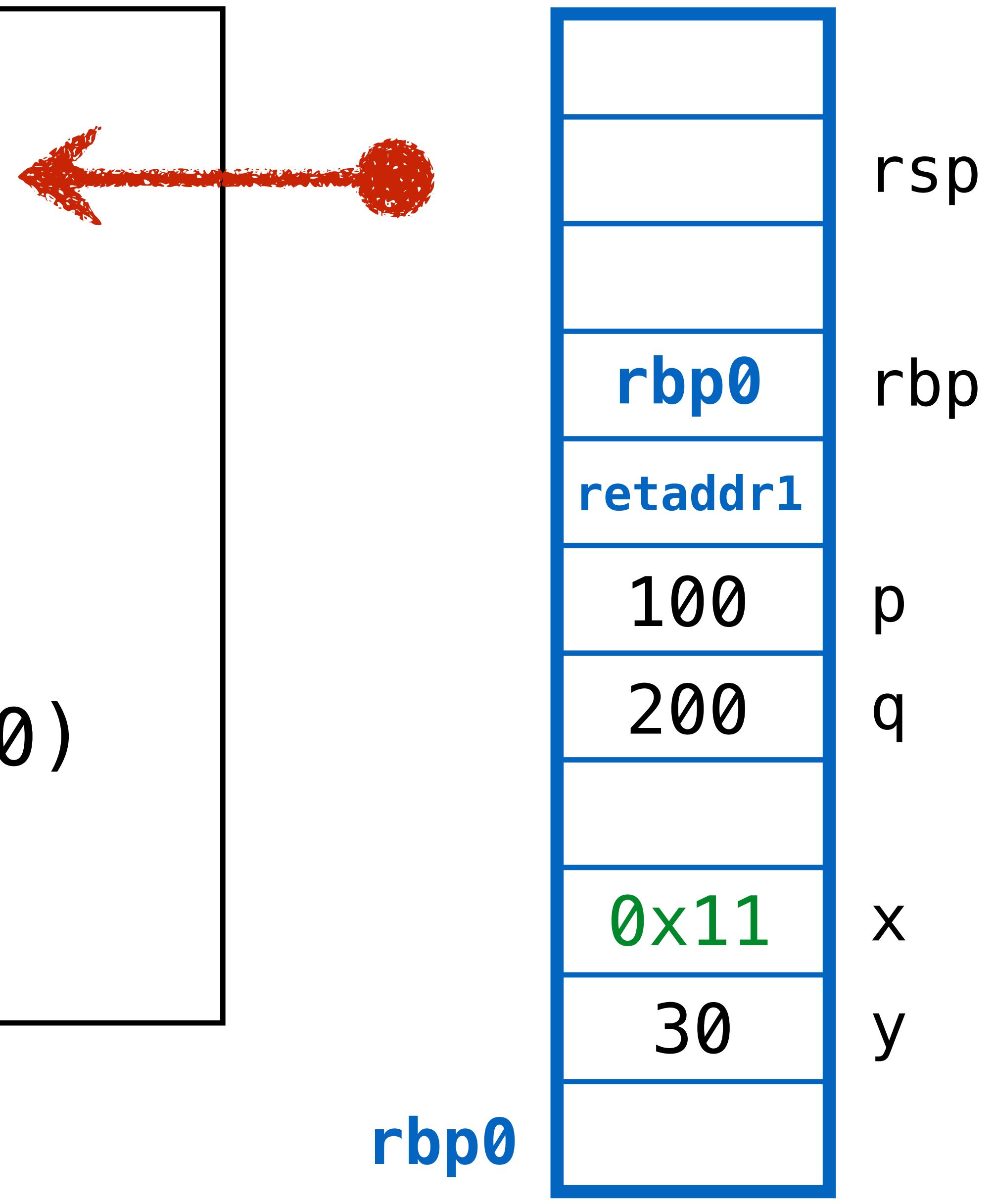




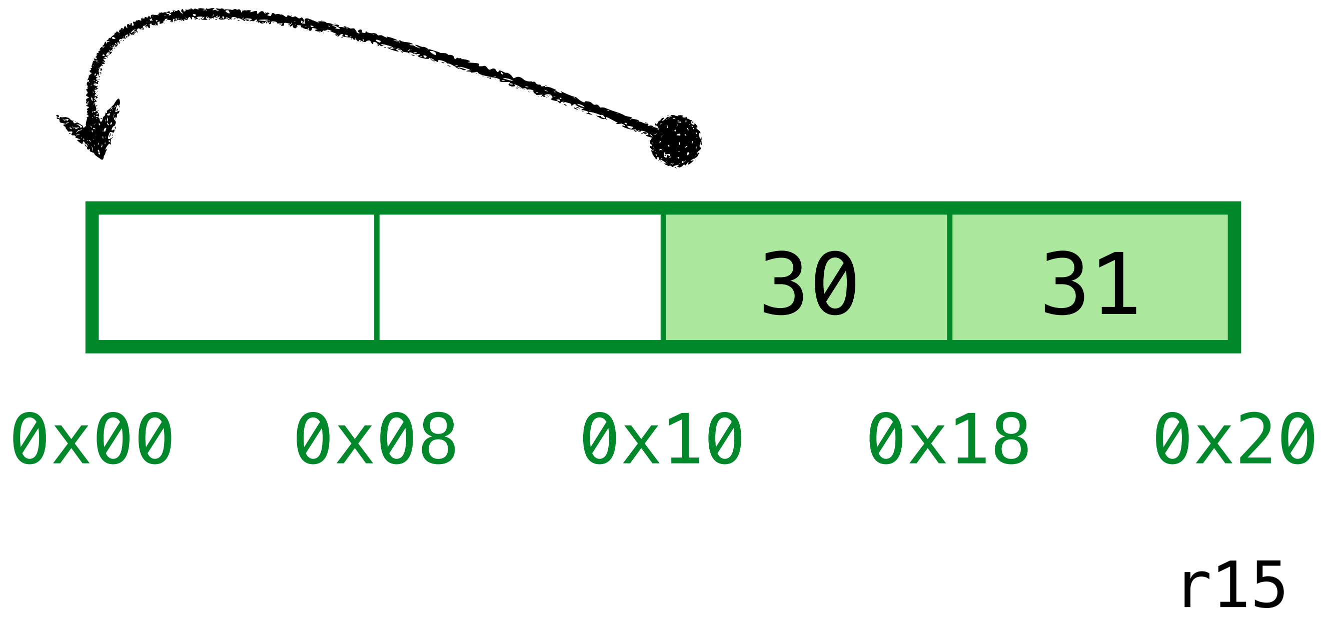
1. Compute FORWARD addrs
   (i.e. new compacted addrs)

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
```



		rsp
	rbp0	rbp
	retaddr1	
	100	p
	200	q
	0x11	x
	30	y
rbp0		

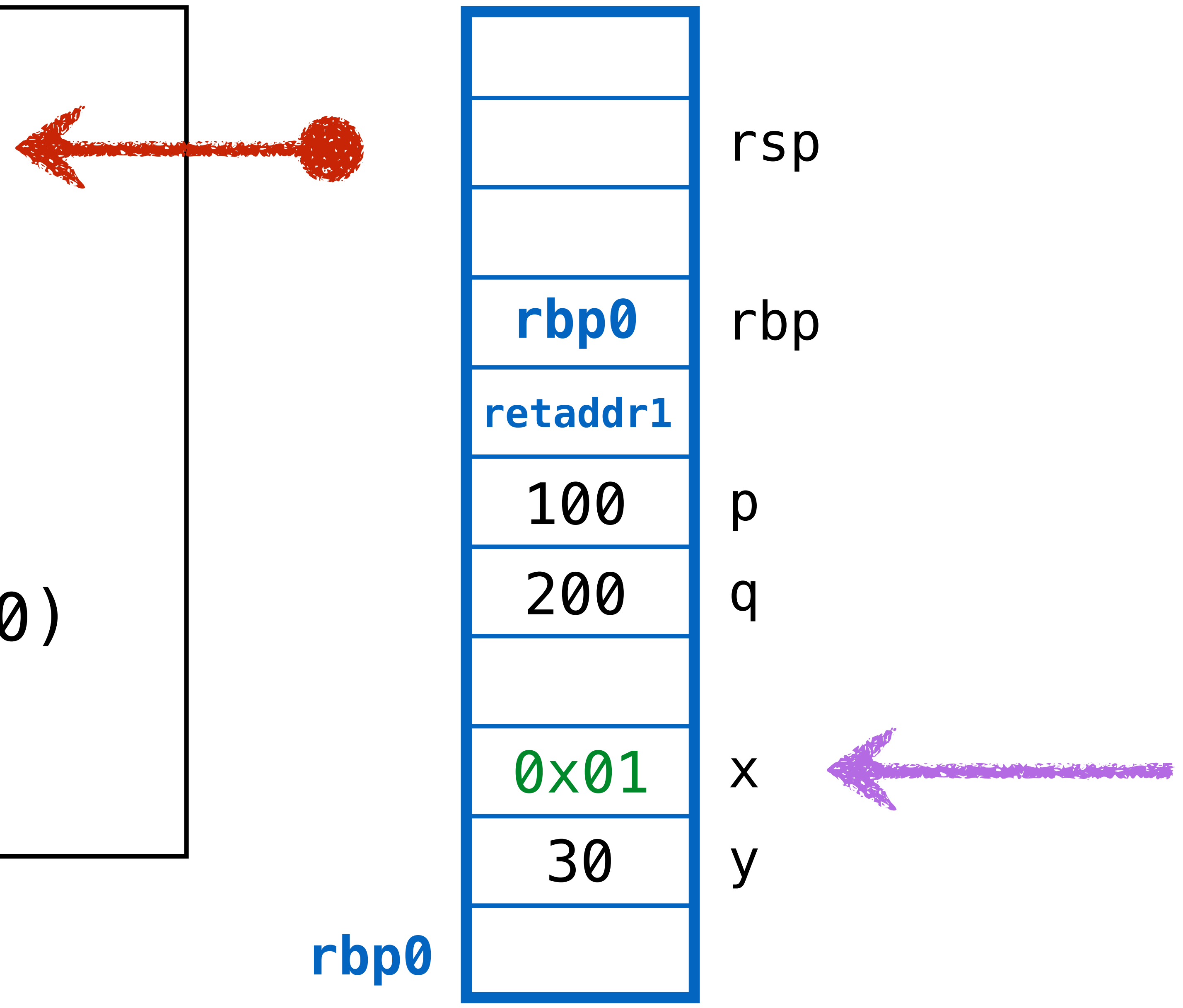


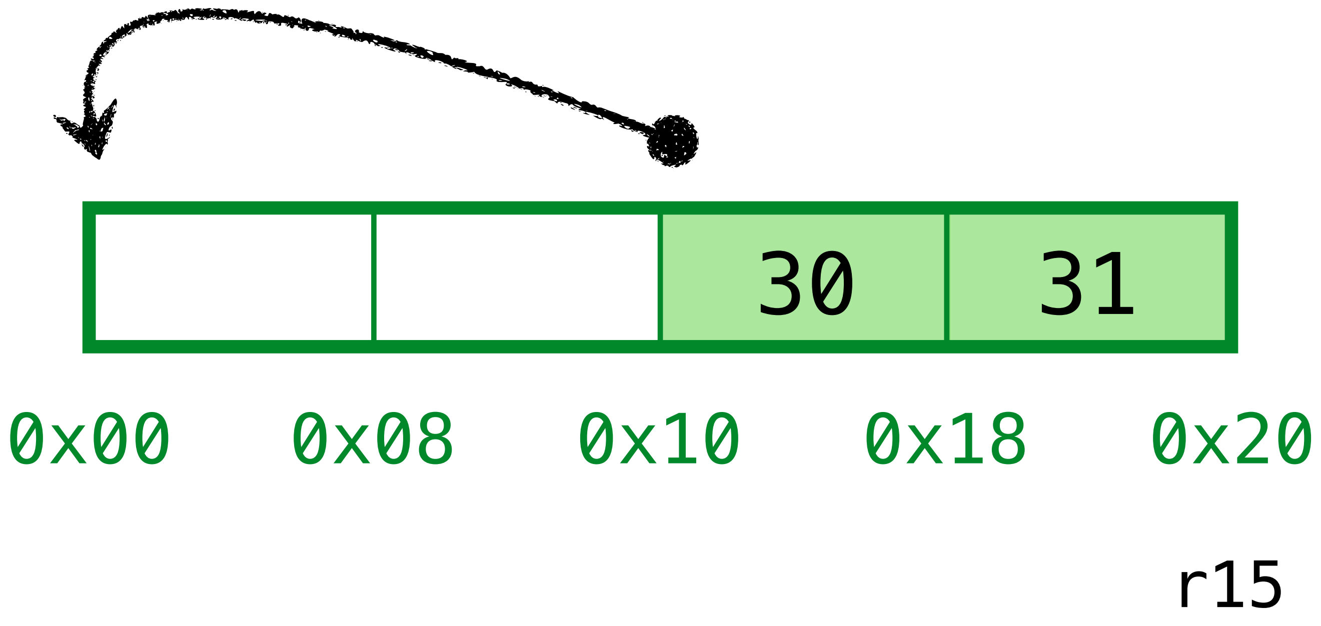
0x00	0x08	0x10	0x18	0x20
		30	31	
				r15

1. Compute FORWARD addrs
   e.g. 0x11 —> 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
```

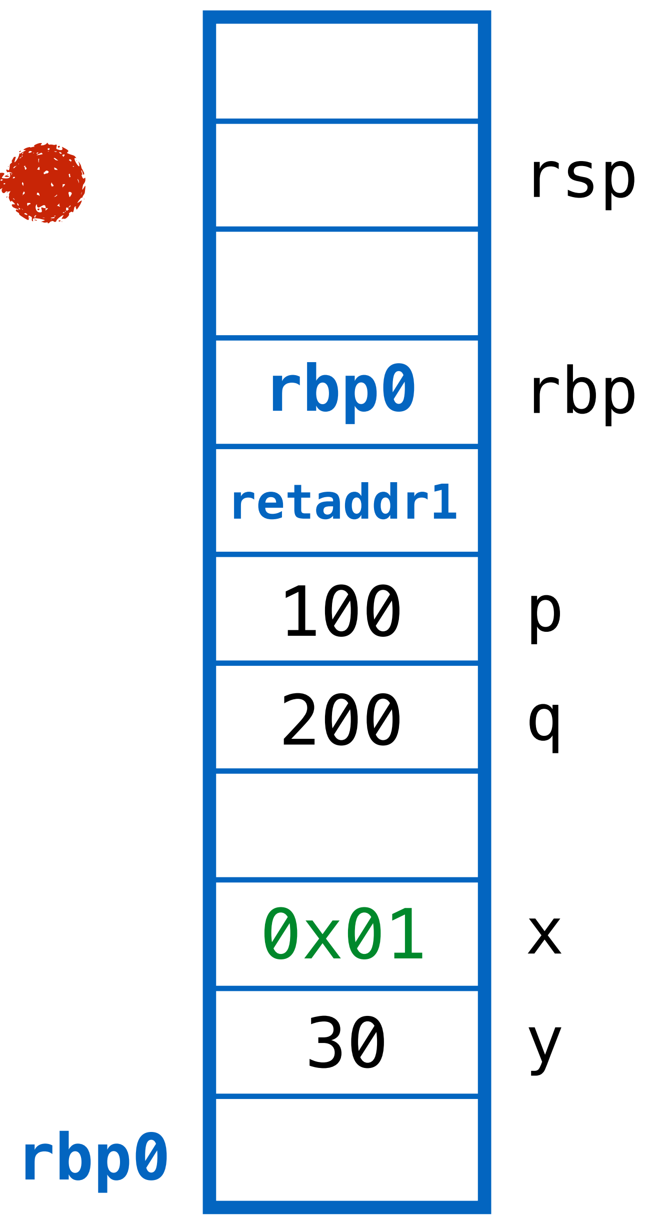


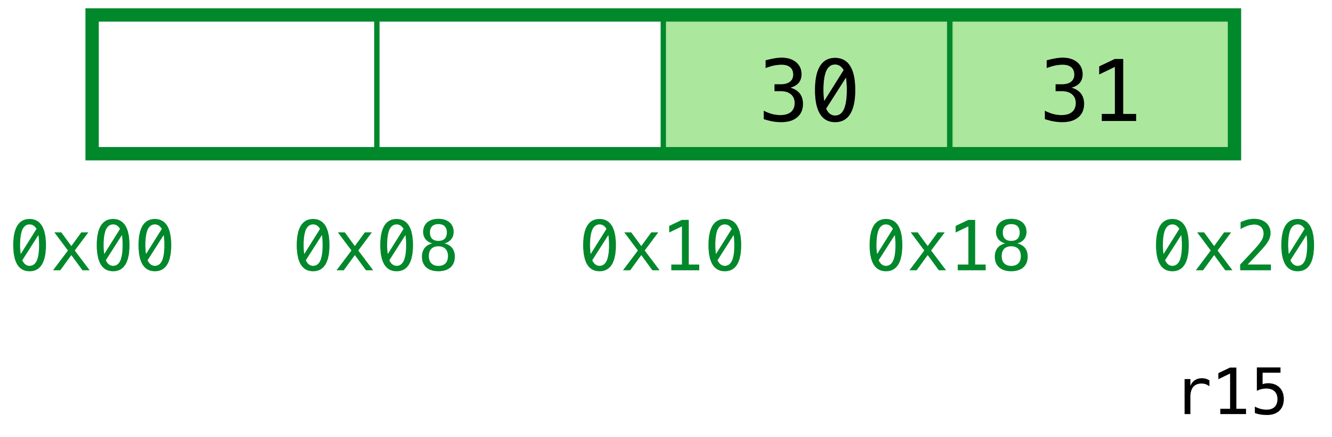


1. Compute FORWARD addrs
   e.g. 0x11 —> 0x01
2. REDIRECT addrs on stack

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
```





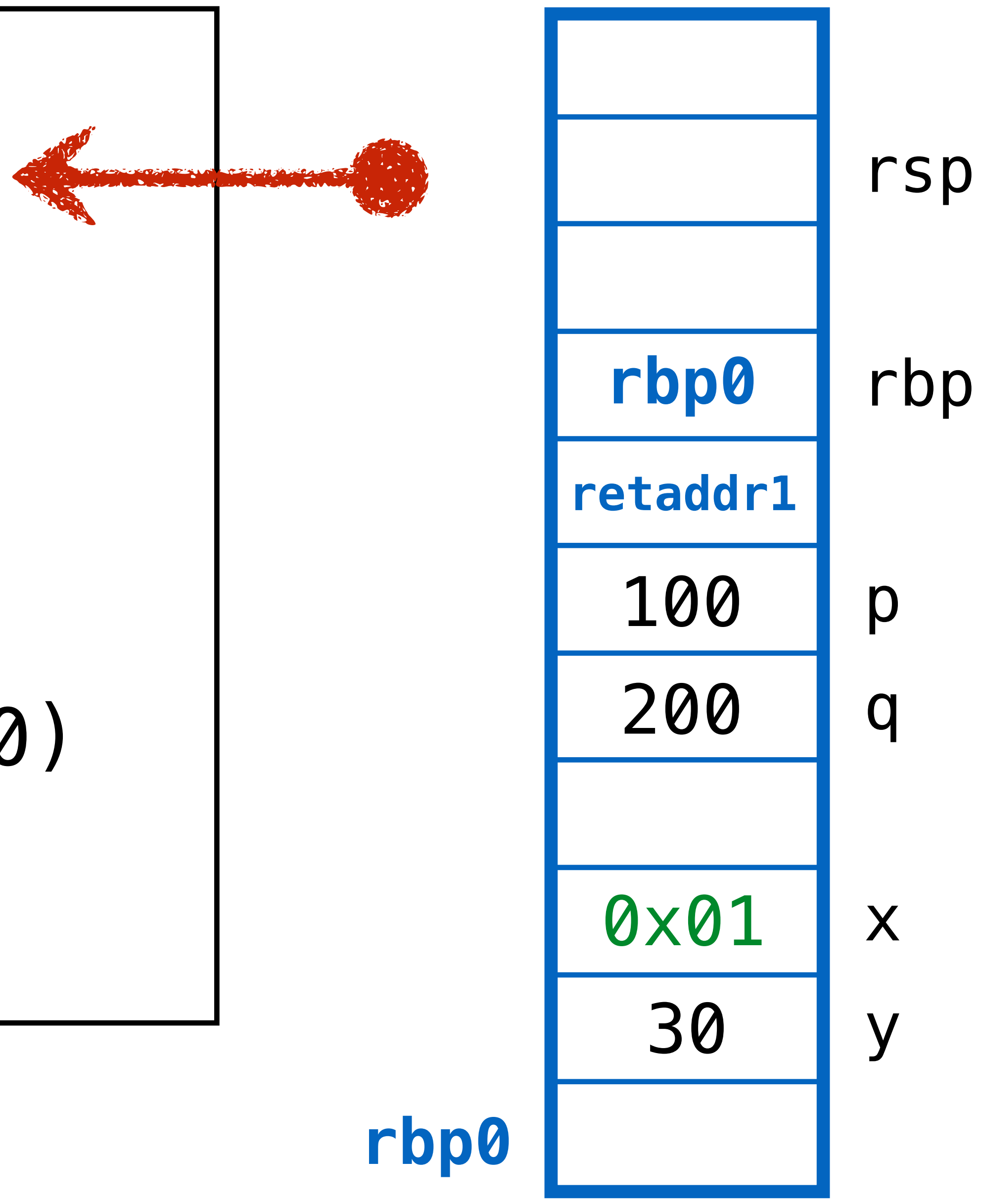
1. Compute FORWARD addrs
   e.g. 0x11 —> 0x01

2. REDIRECT addrs on stack

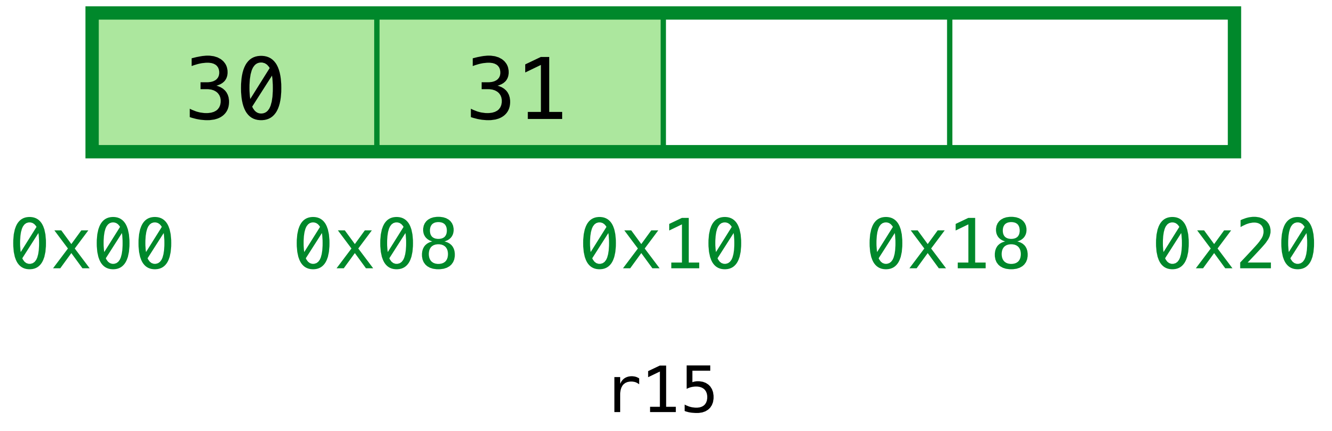
3. COMPACT cells on heap

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
```



```
|          |
|          | rsp
|          |
| rbp0     | rbp
| retaddr1 |
| 100      | p
| 200      | q
|          |
| 0x01     | x
| 30       | y
|          |
rbp0
```



```
| 30 | 31 |    |    |
0x00 0x08 0x10 0x18 0x20
          r15
```

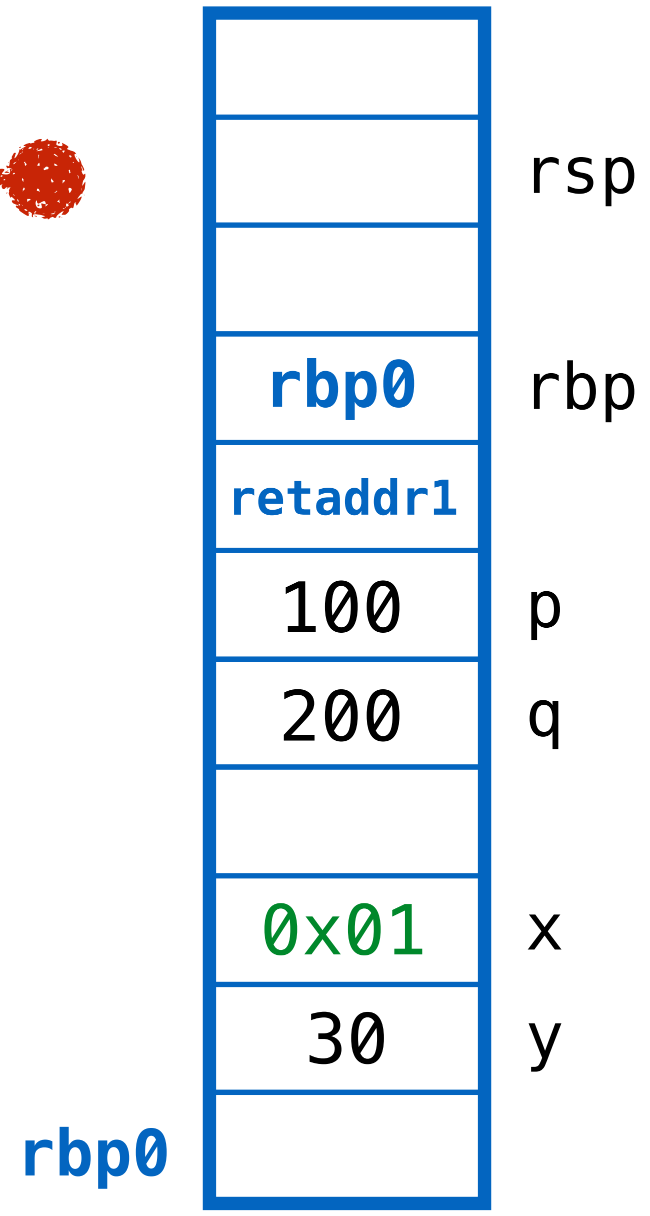
1. Compute FORWARD addrs
   e.g. 0x11 —> 0x01

2. REDIRECT addrs on stack

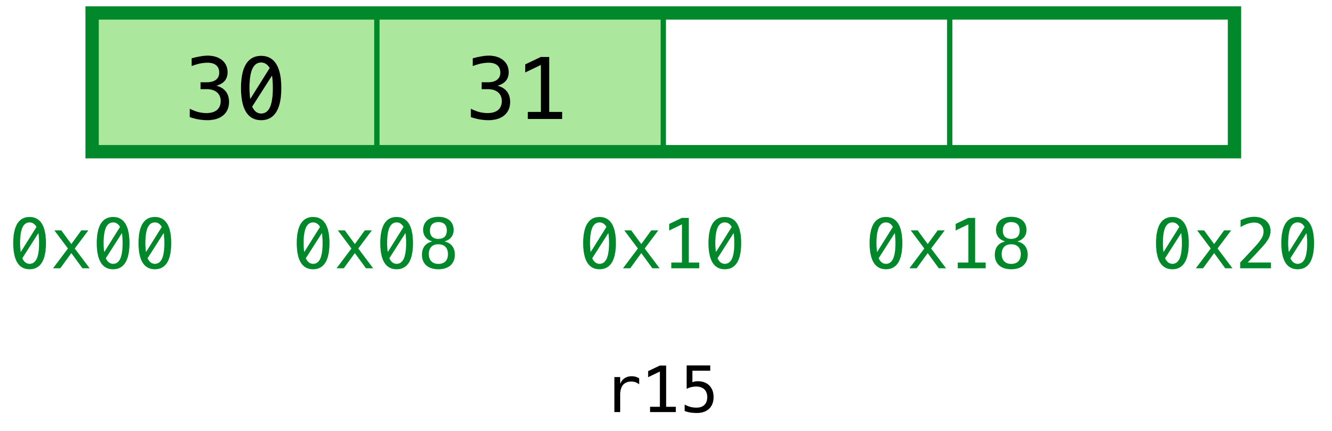
3. COMPACT cells on heap

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
```

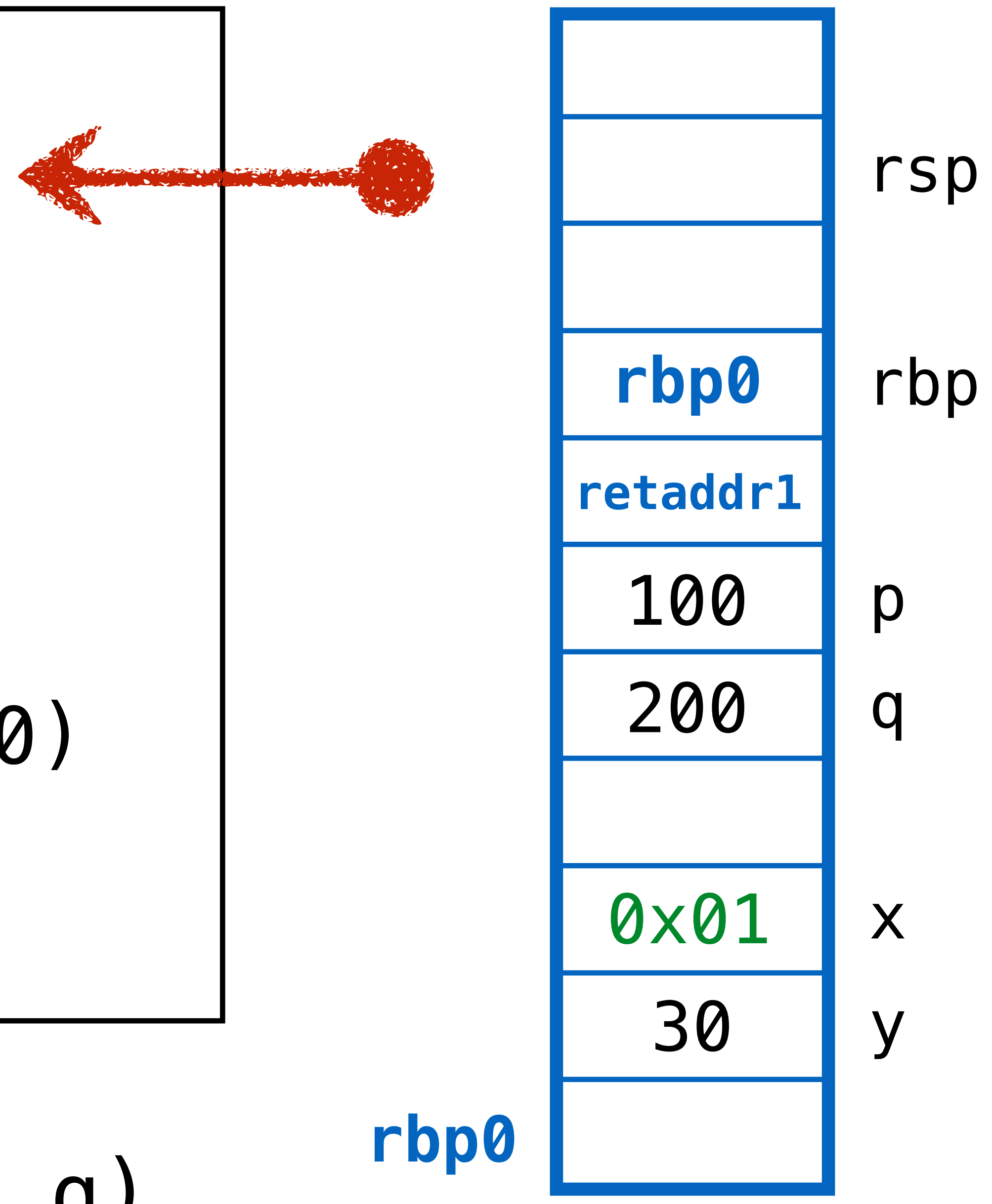


Yay! Have space for `(p, q)`

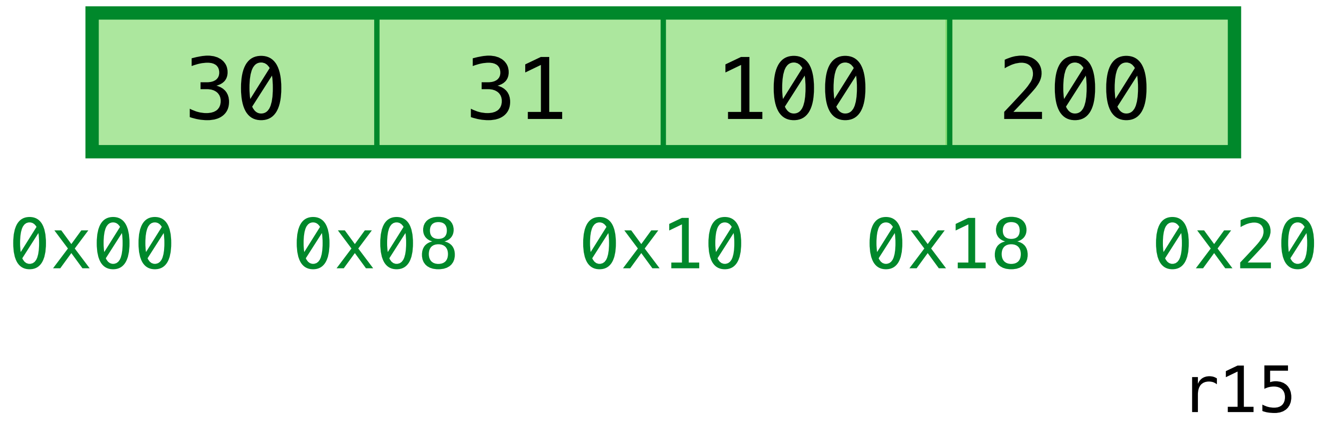


ex3: garbage in the middle (with stack)

```
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
```

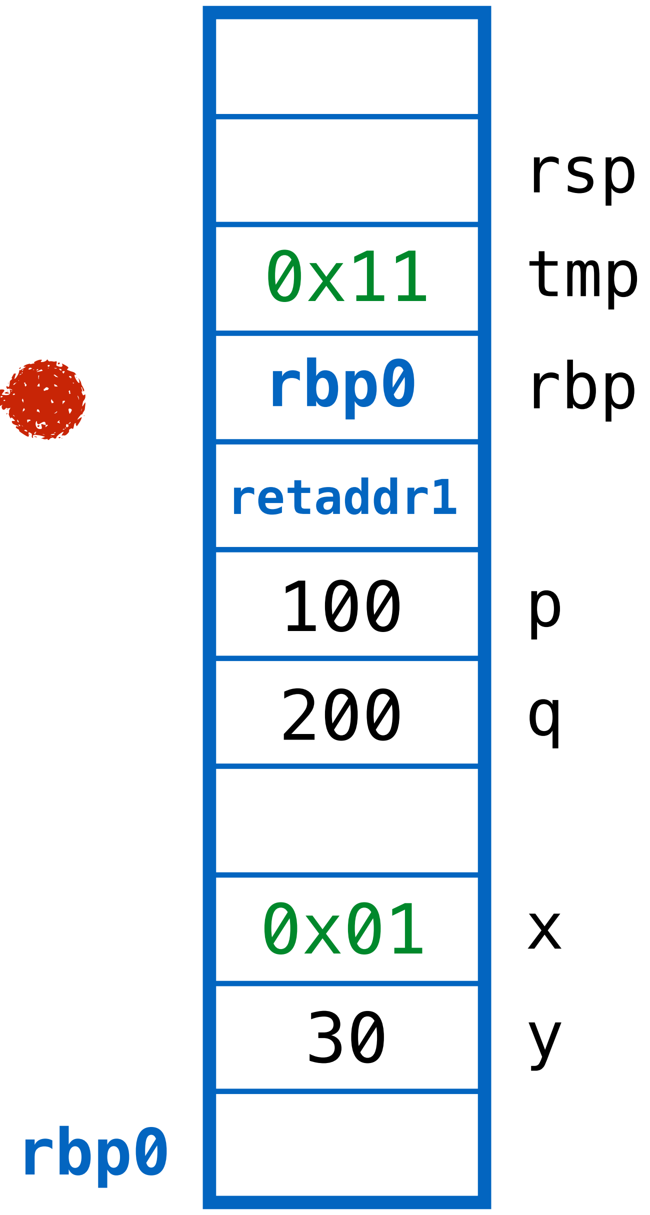


Yay! Have space for `(p, q)`



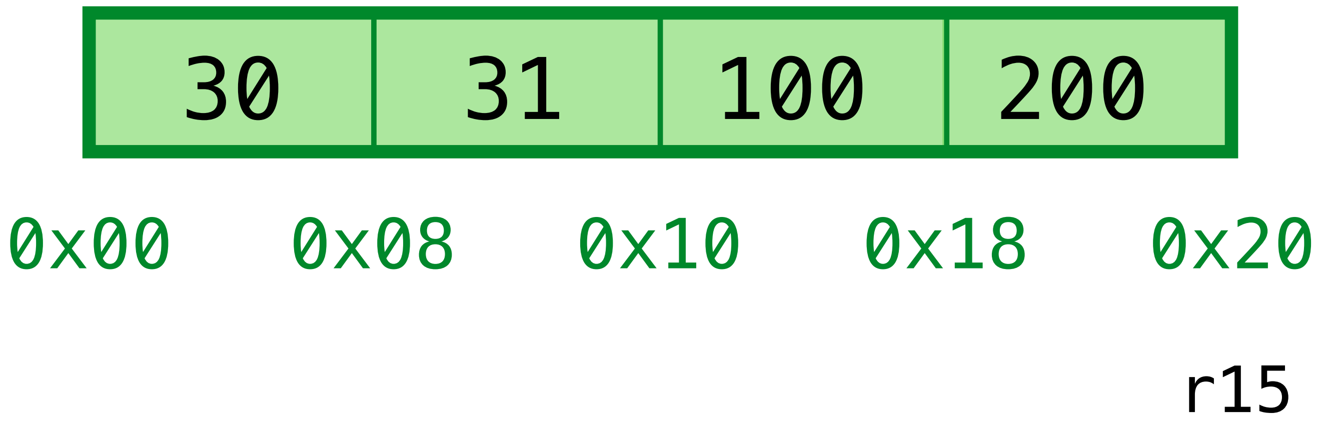
ex3: garbage in the middle (with stack)

```
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
```



	rsp
0x11	tmp
rbp0	rbp
retaddr1	
100	p
200	q
0x01	x
30	y

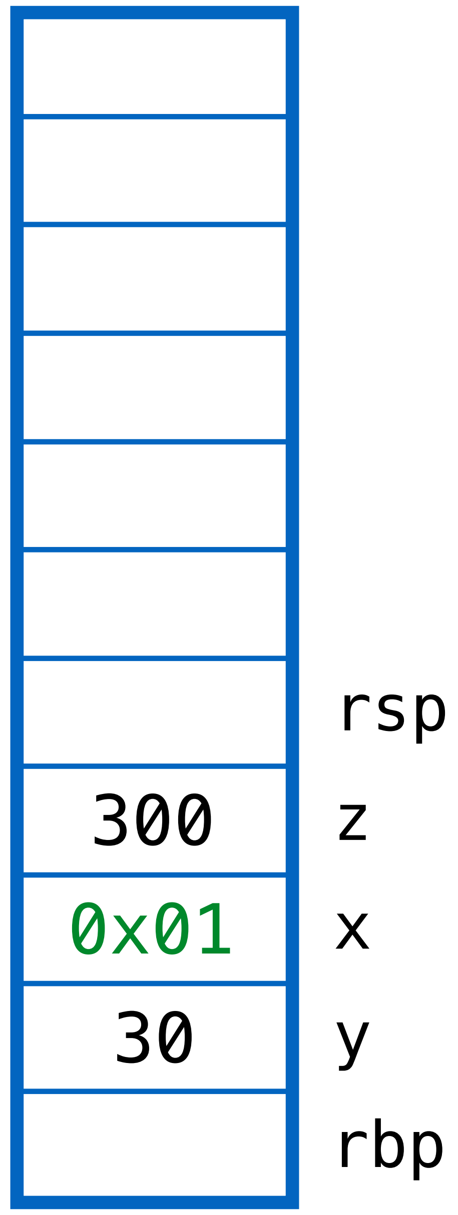
Return (rax) = 300



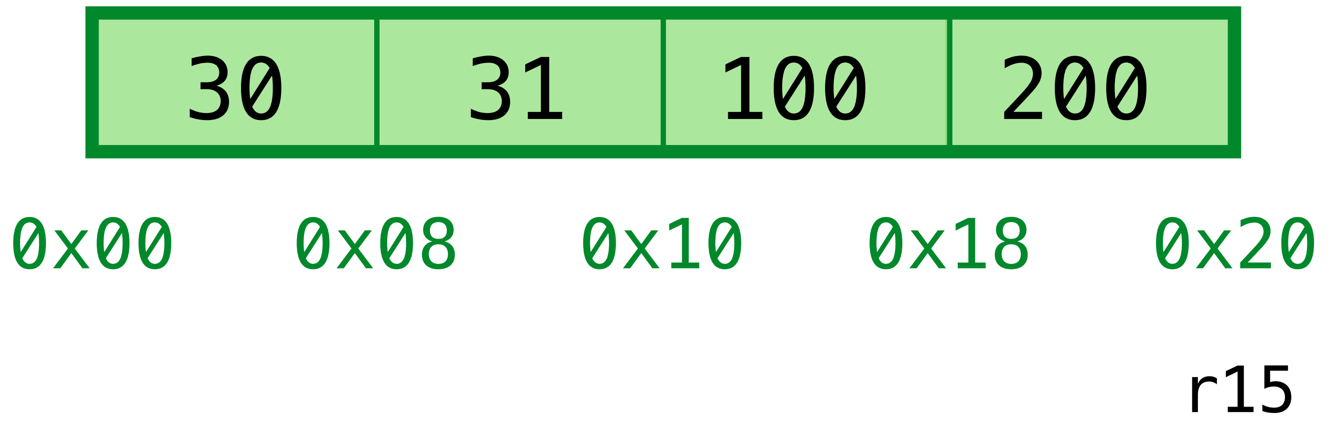
30	31	100	200	
0x00	0x08	0x10	0x18	0x20

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
```

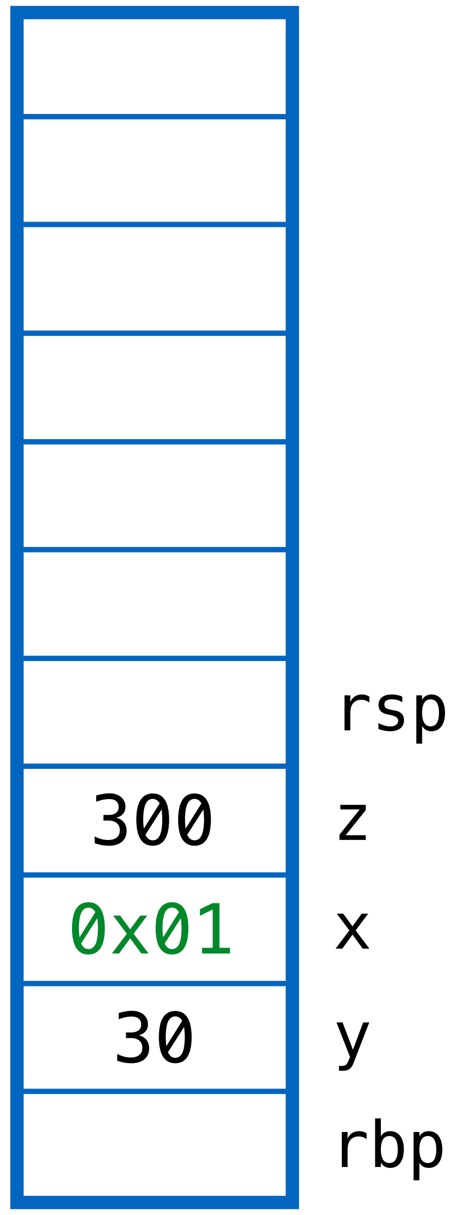


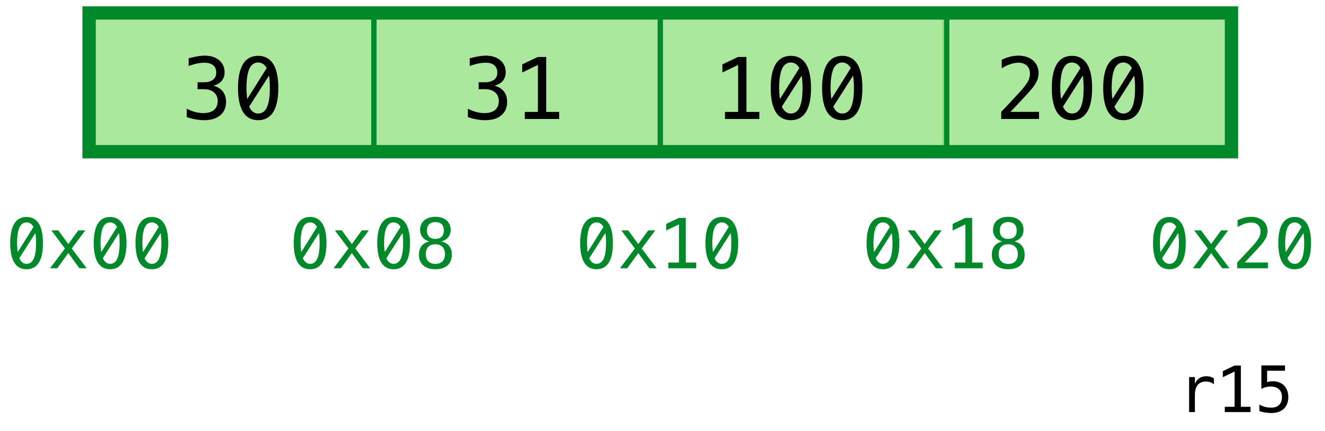
Return (rax) = 300



ex3: garbage in the middle (with stack)

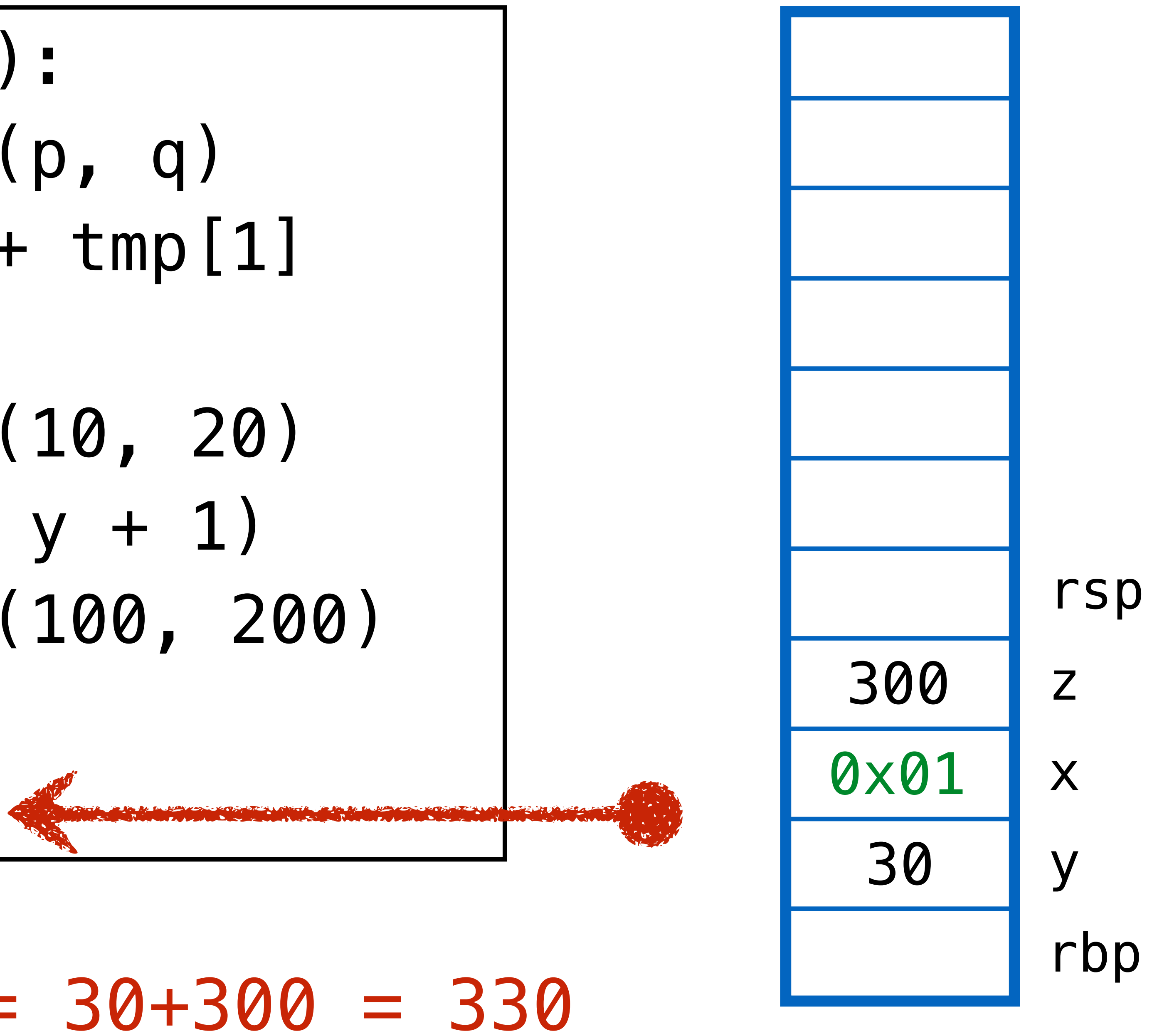
```
def foo(p, q):
  let tmp = (p, q)
  in tmp[0] + tmp[1]

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
```

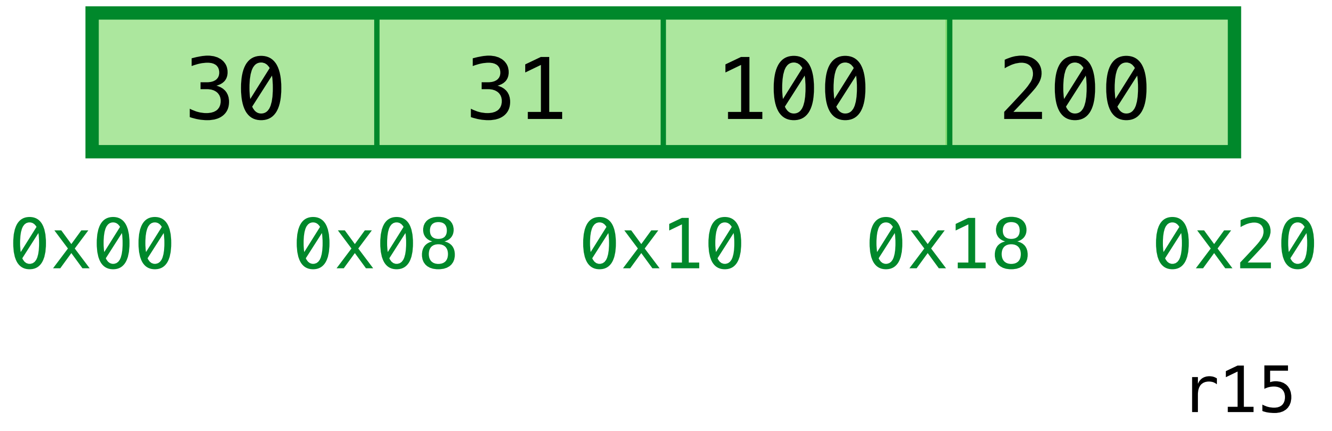


rsp

300	z
0x01	x
30	y

rbp

Return (rax) = 30+300 = 330



30	31	100	200

0x00 0x08 0x10 0x18 0x20

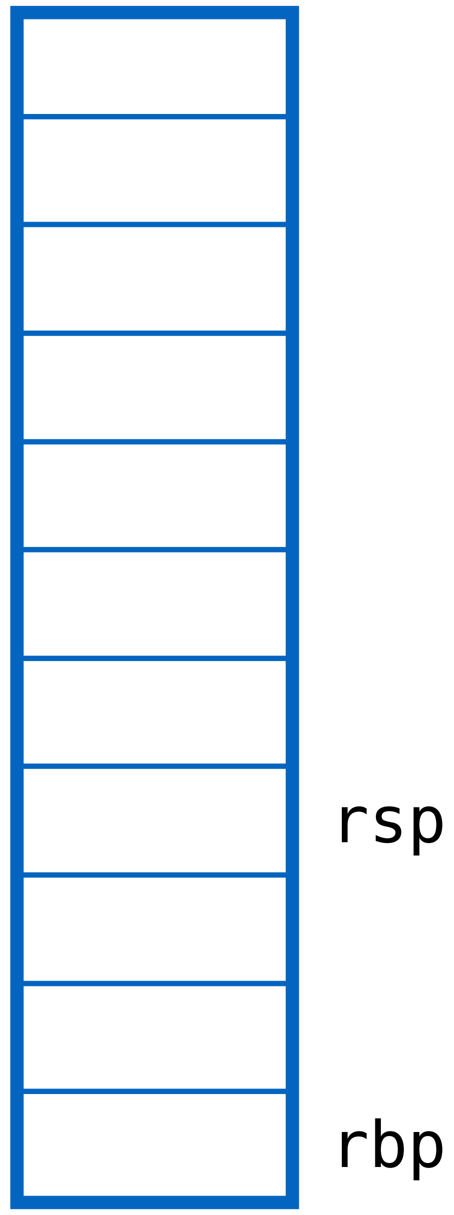
r15

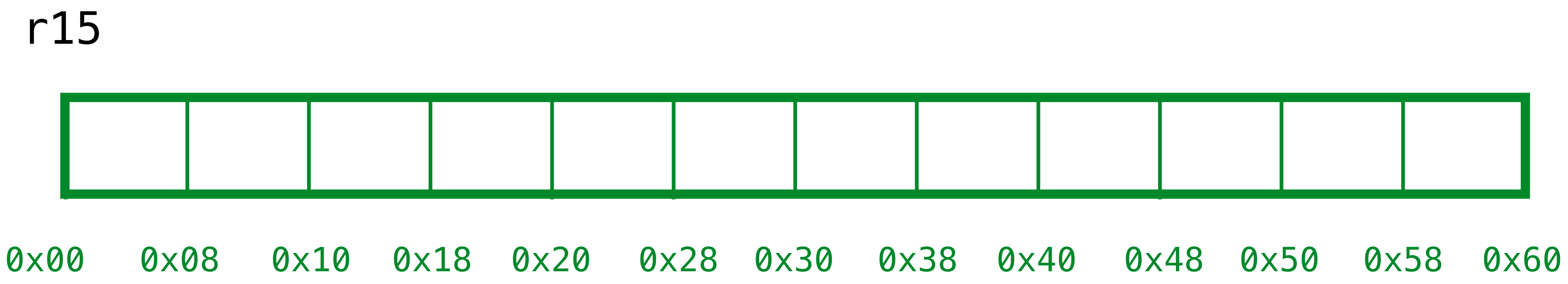
Garter / GC

Example 4

ex4: recursive data

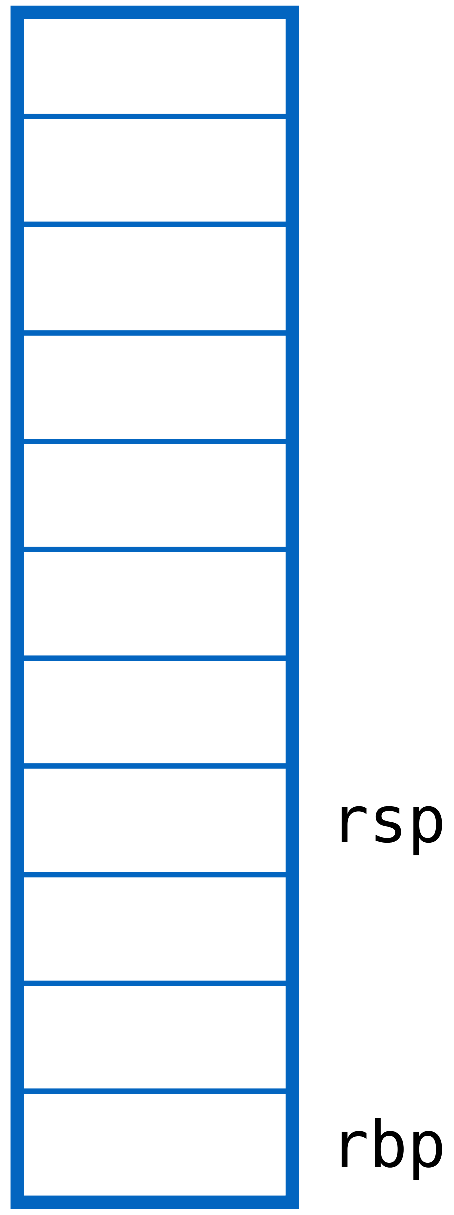
```
def range(i, j):
  if (j <= i): false else: (i,range(i+1, j))

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)
```



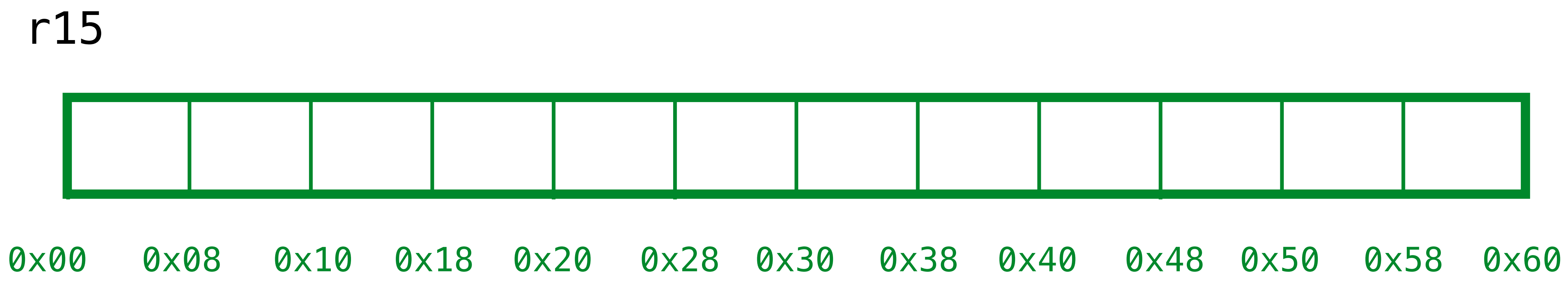


ex4: recursive data

```
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)
```

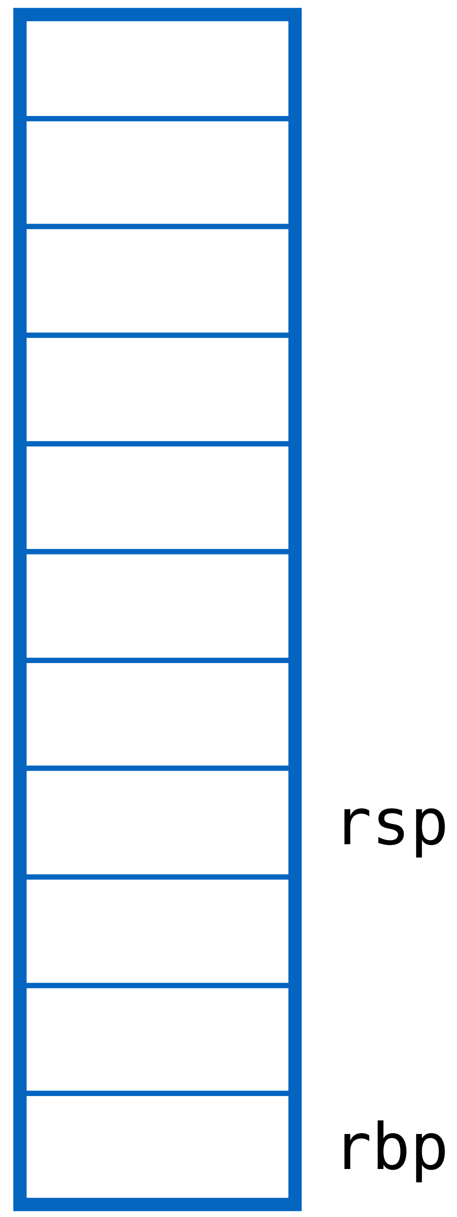


call range(0, 3)



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?

r15

(A)

0	0x11	1	0x21	2	false							
0x00	0x08	0x10	0x18	0x20	0x28	0x30	0x38	0x40	0x48	0x50	0x58	0x60

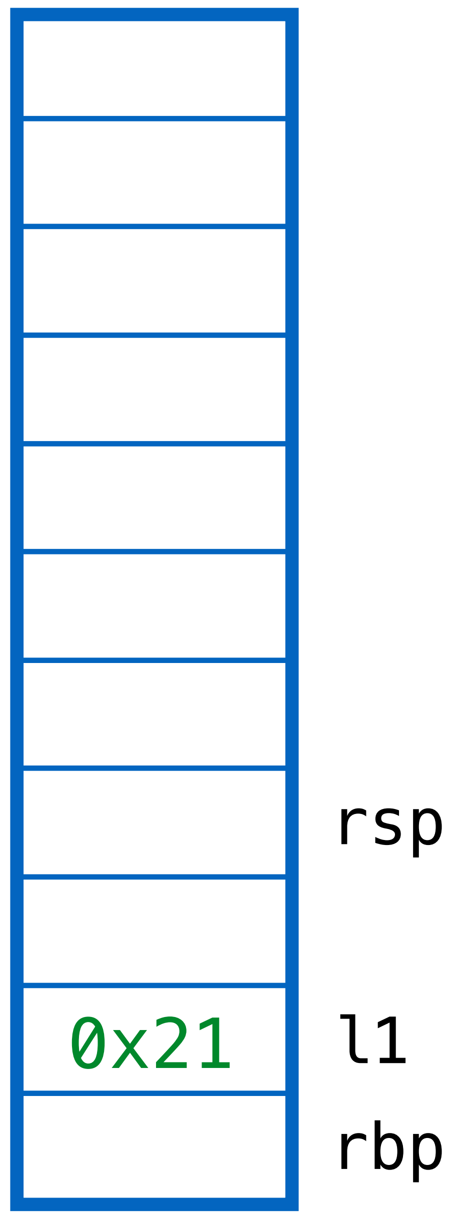
r15

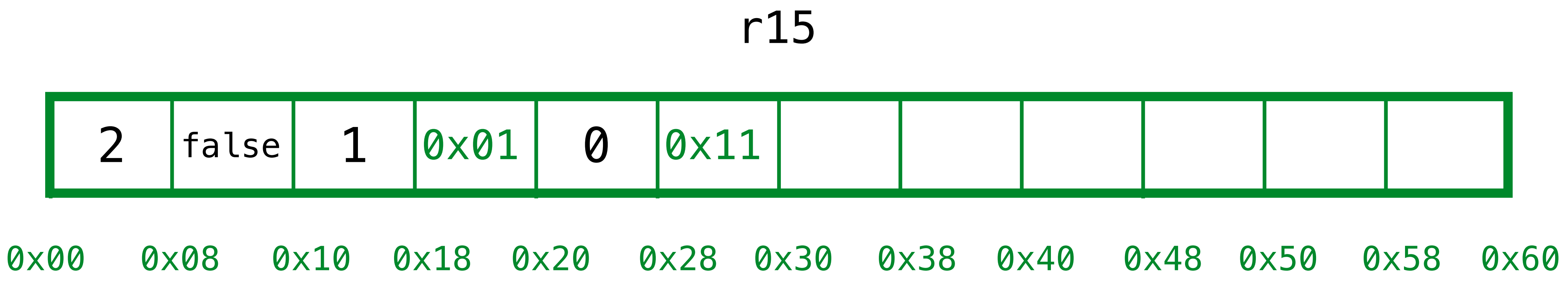
(B)

2	false	1	0x01	0	0x11							
0x00	0x08	0x10	0x18	0x20	0x28	0x30	0x38	0x40	0x48	0x50	0x58	0x60

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)
```



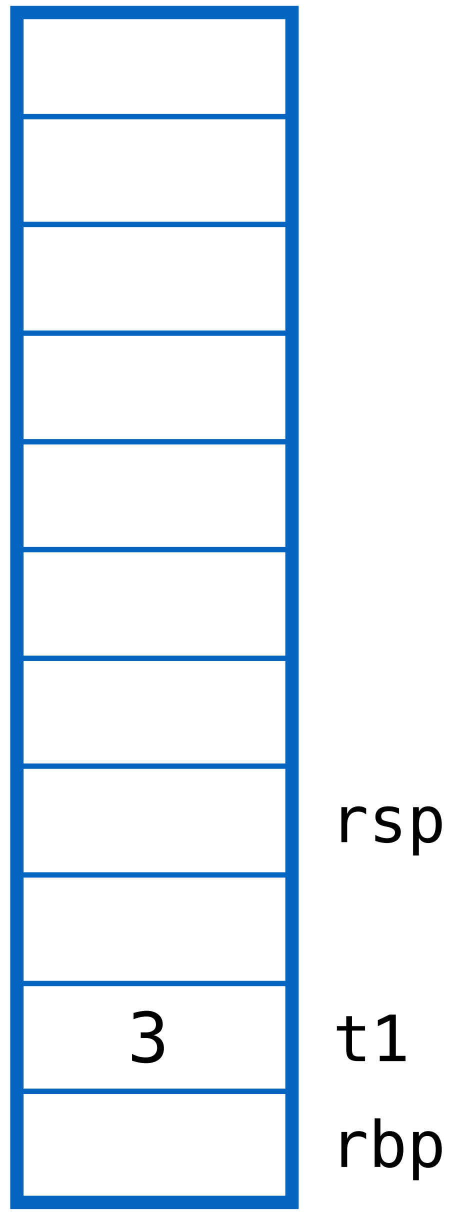


2	false	1	0x01	0	0x11						

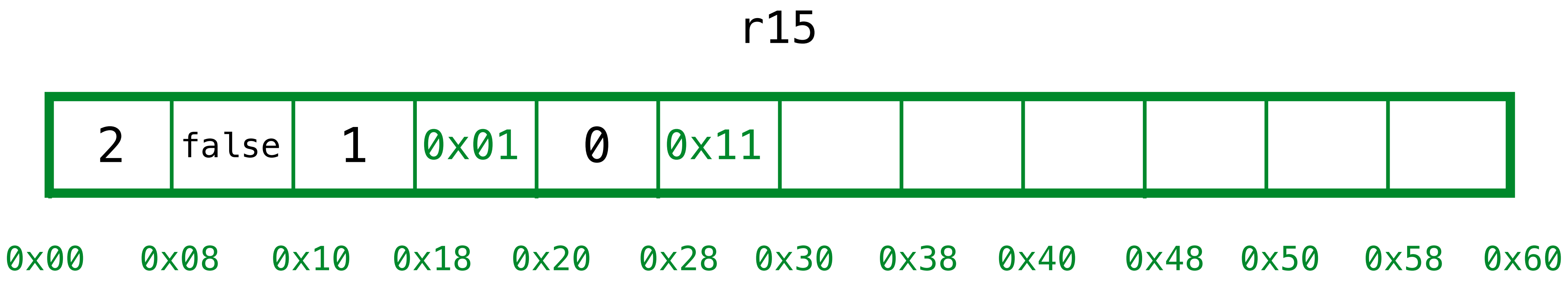
```
0x00  0x08  0x10  0x18  0x20  0x28  0x30  0x38  0x40  0x48  0x50  0x58  0x60
```

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)
```

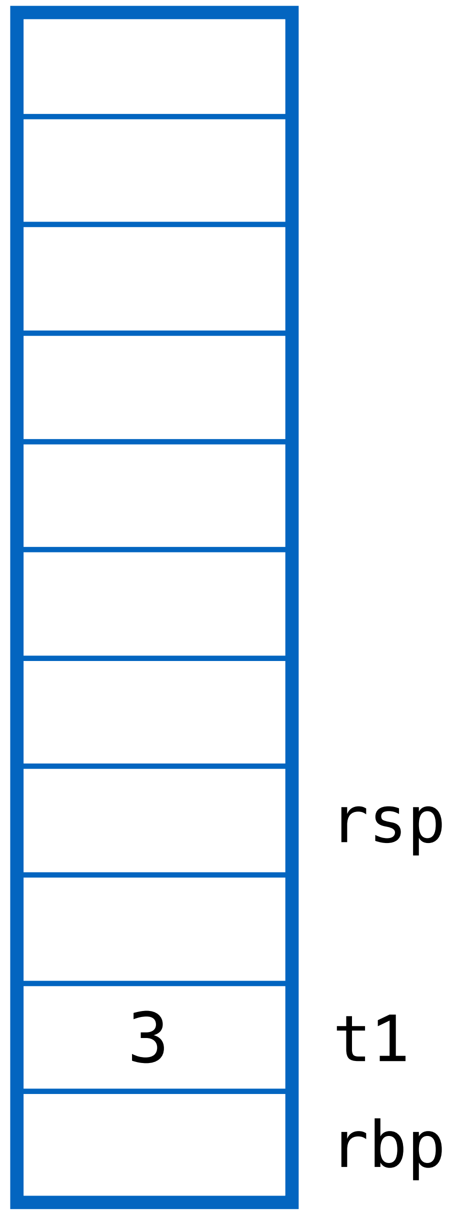


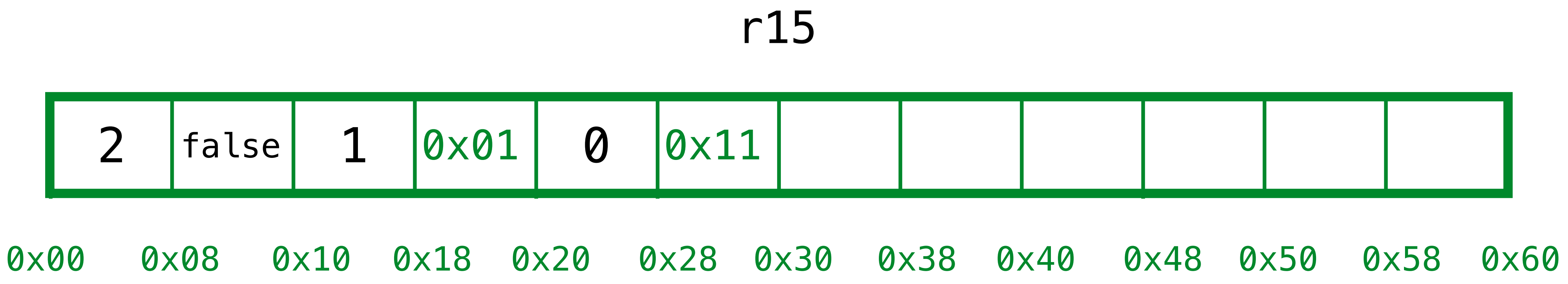
Result sum(0x11) = 3



ex4: recursive data

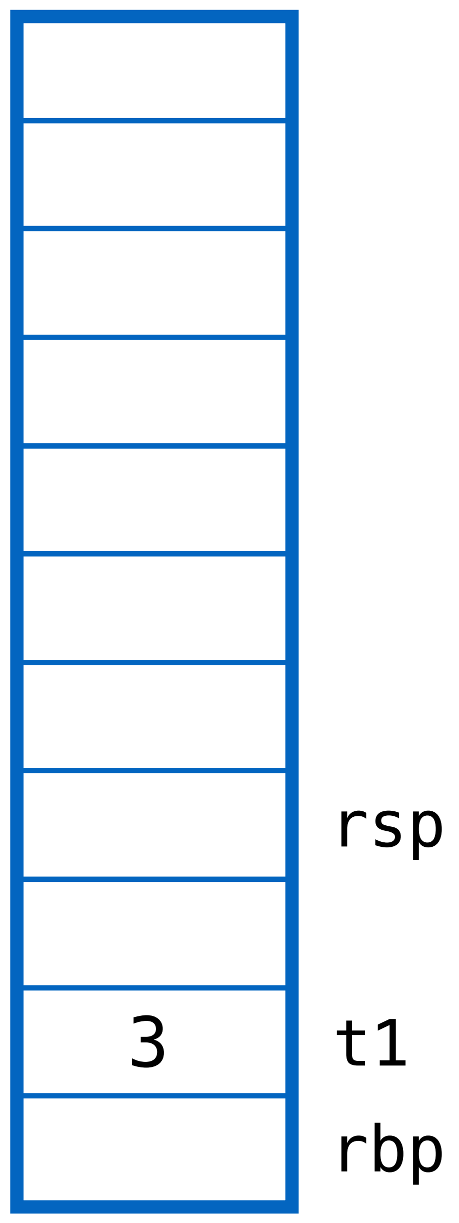
```
def range(i, j):
  if (j <= i): false else: (i,range(i+1, j))

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)
```



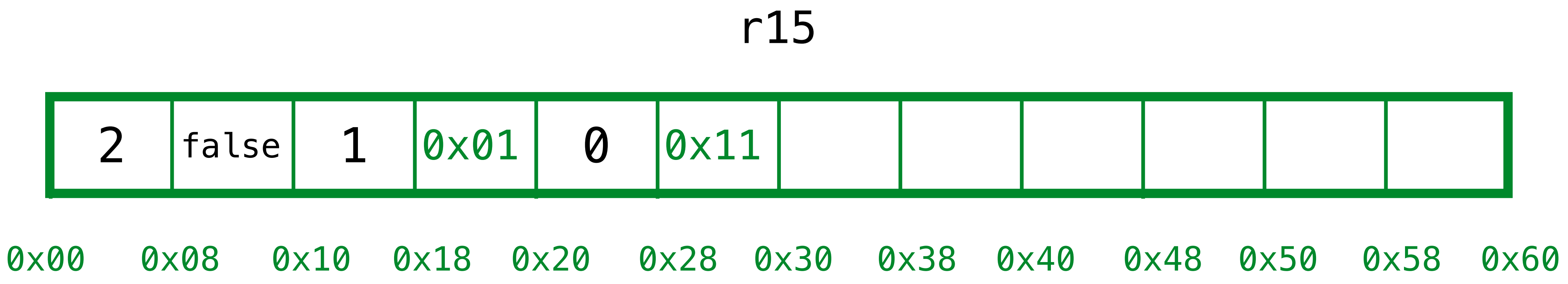


ex4: recursive data

```
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)
```

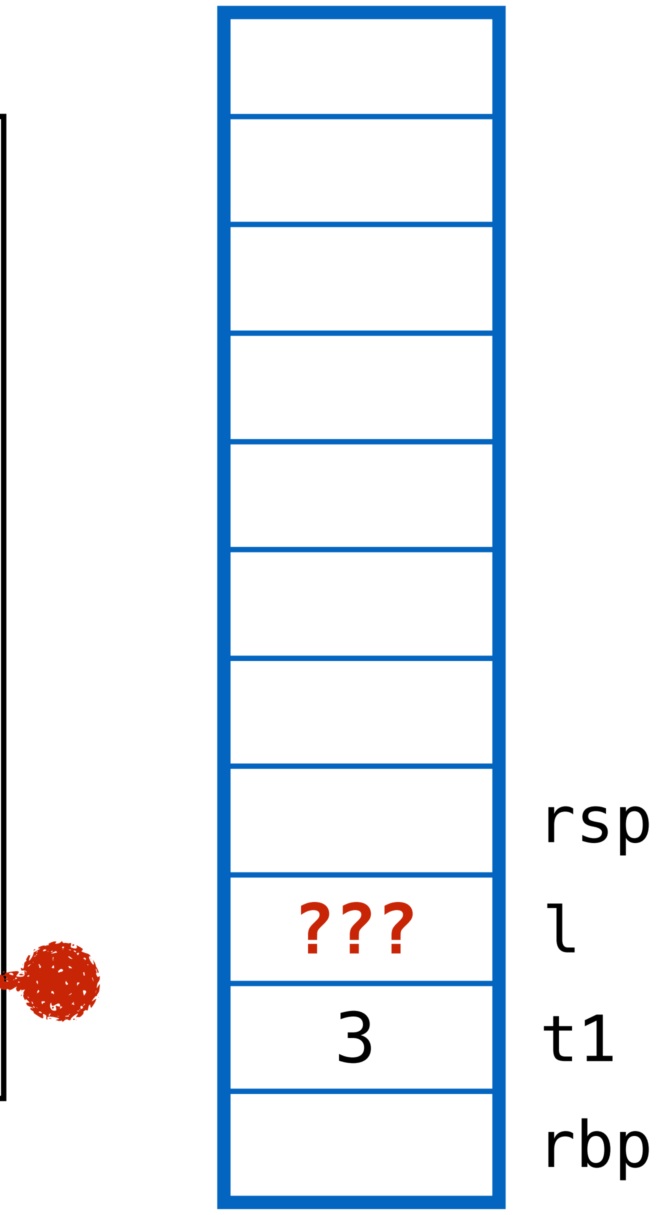


call range(3,6)

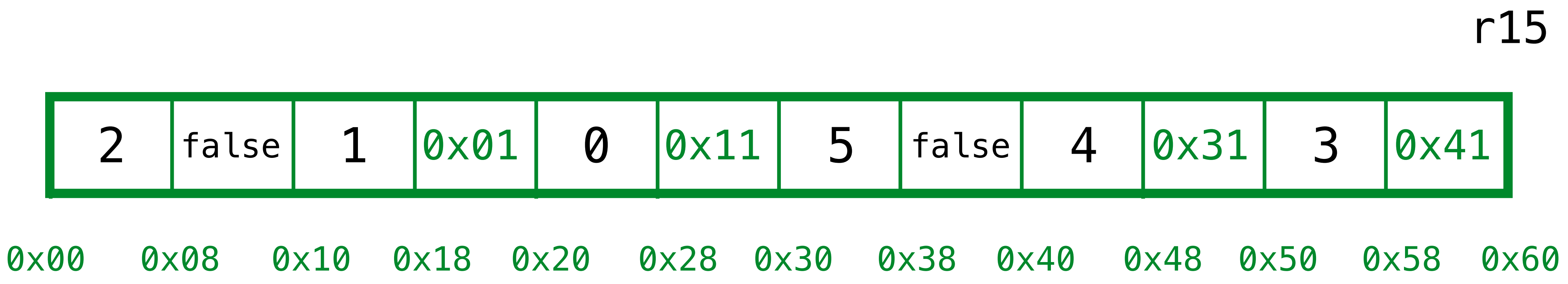


ex4: recursive data

```
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)
```



call range(3,6)



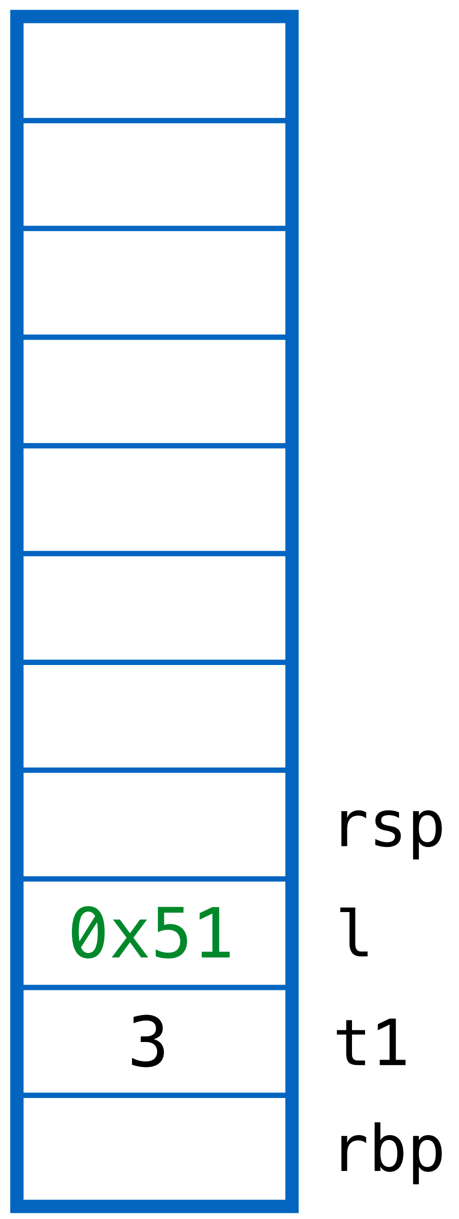
2	false	1	0x01	0	0x11	5	false	4	0x31	3	0x41

QUIZ: What is the value of l?

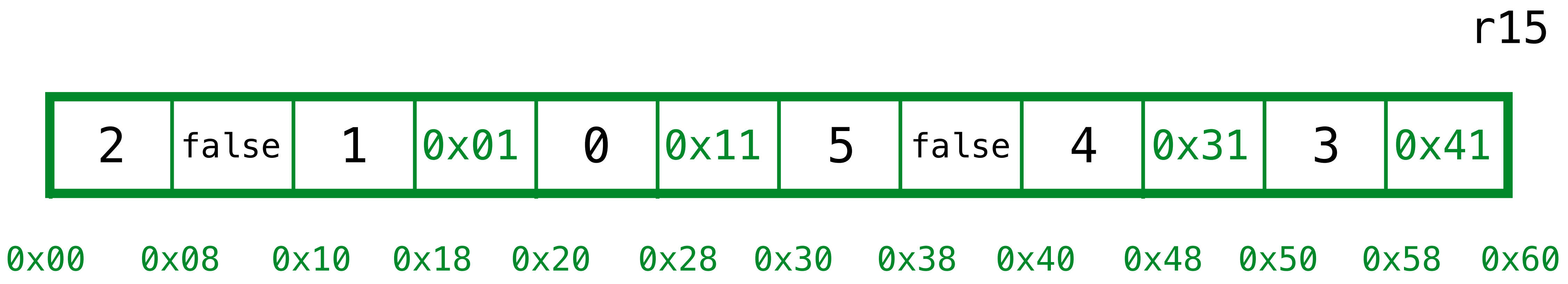
(A) 0x30 (B) 0x31 (C) 0x50 (D) 0x51 (E) 0x60

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

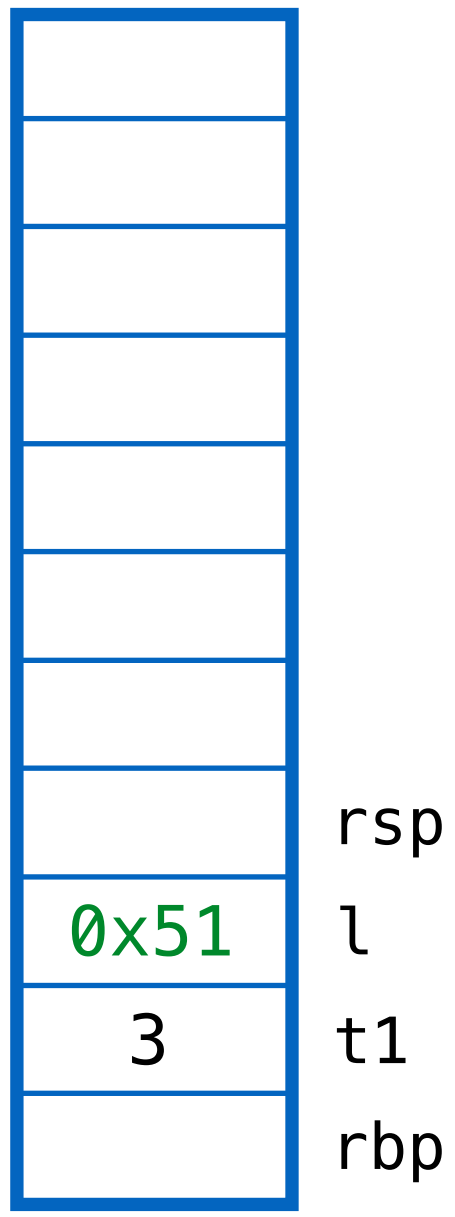
(B) 0x10

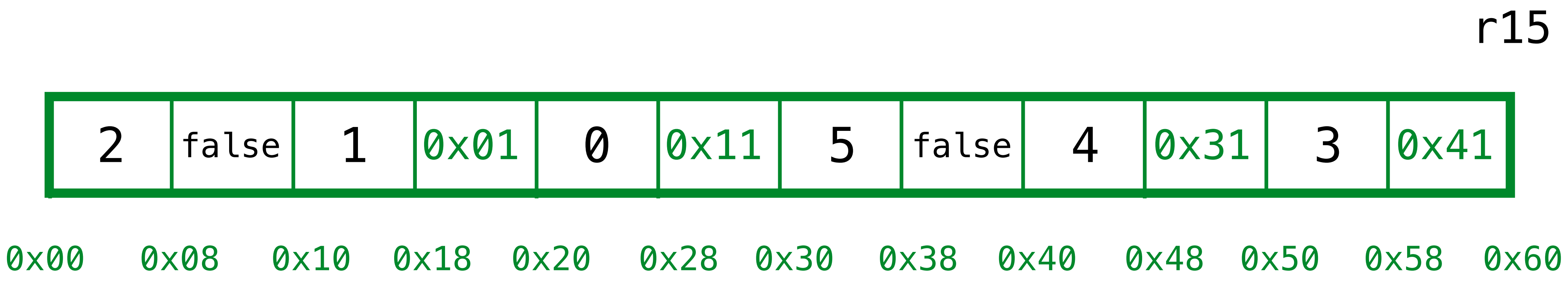
(C) 0x20

(D) 0x30

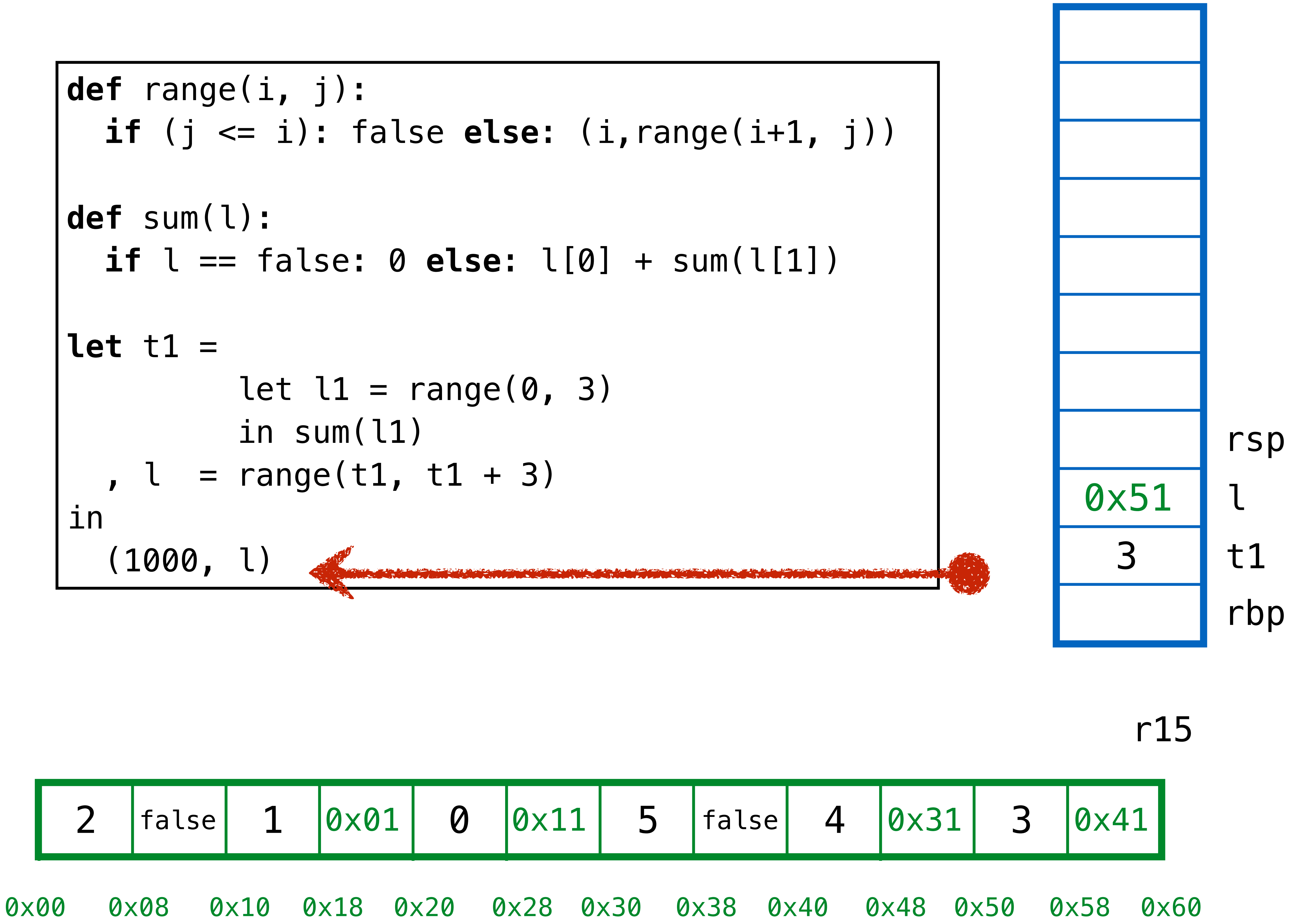
(E) 0x40

(F) 0x50





ex4: recursive data

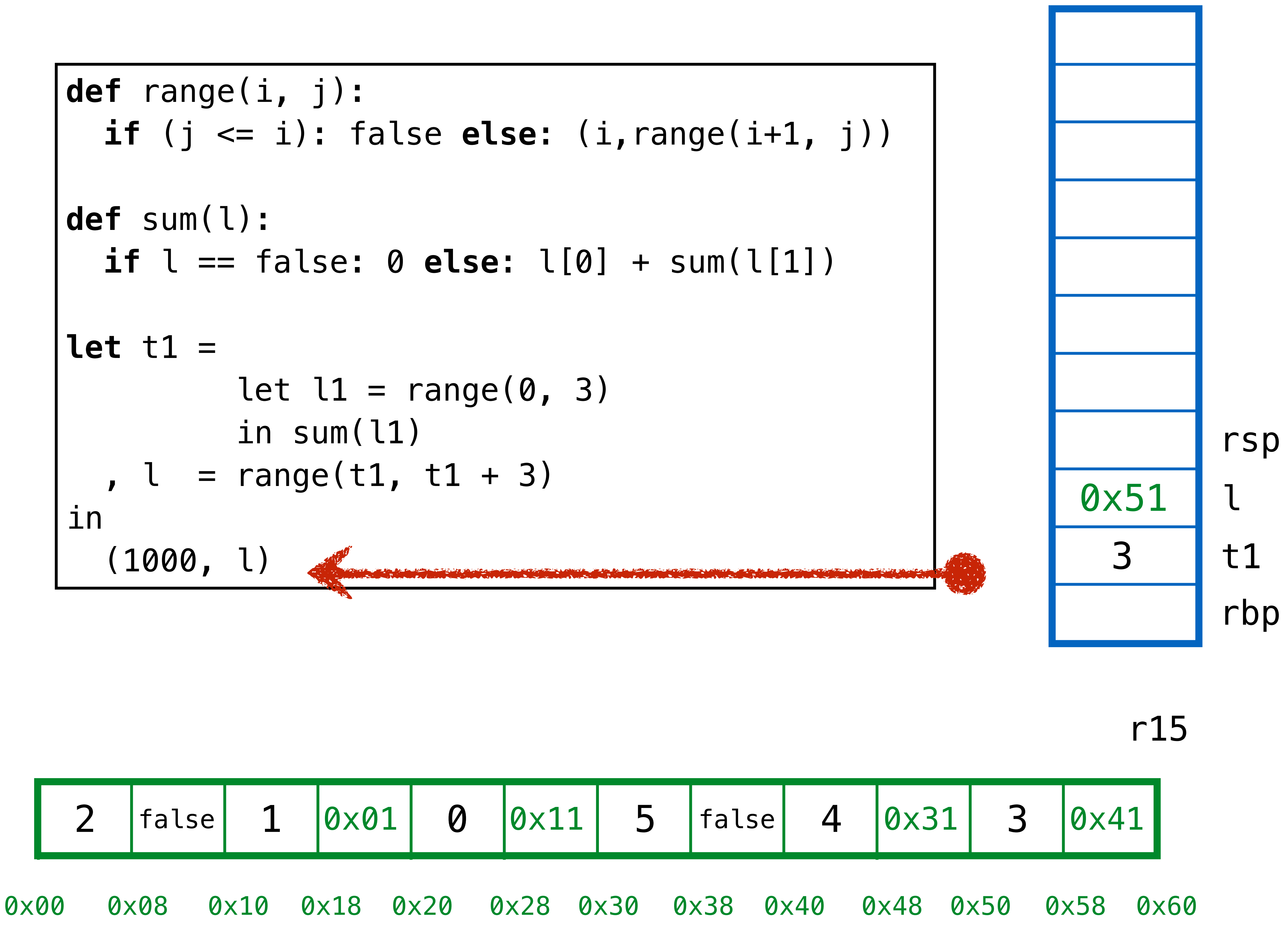


```
def range(i, j):
  if (j <= i): false else: (i,range(i+1, j))

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
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)
```

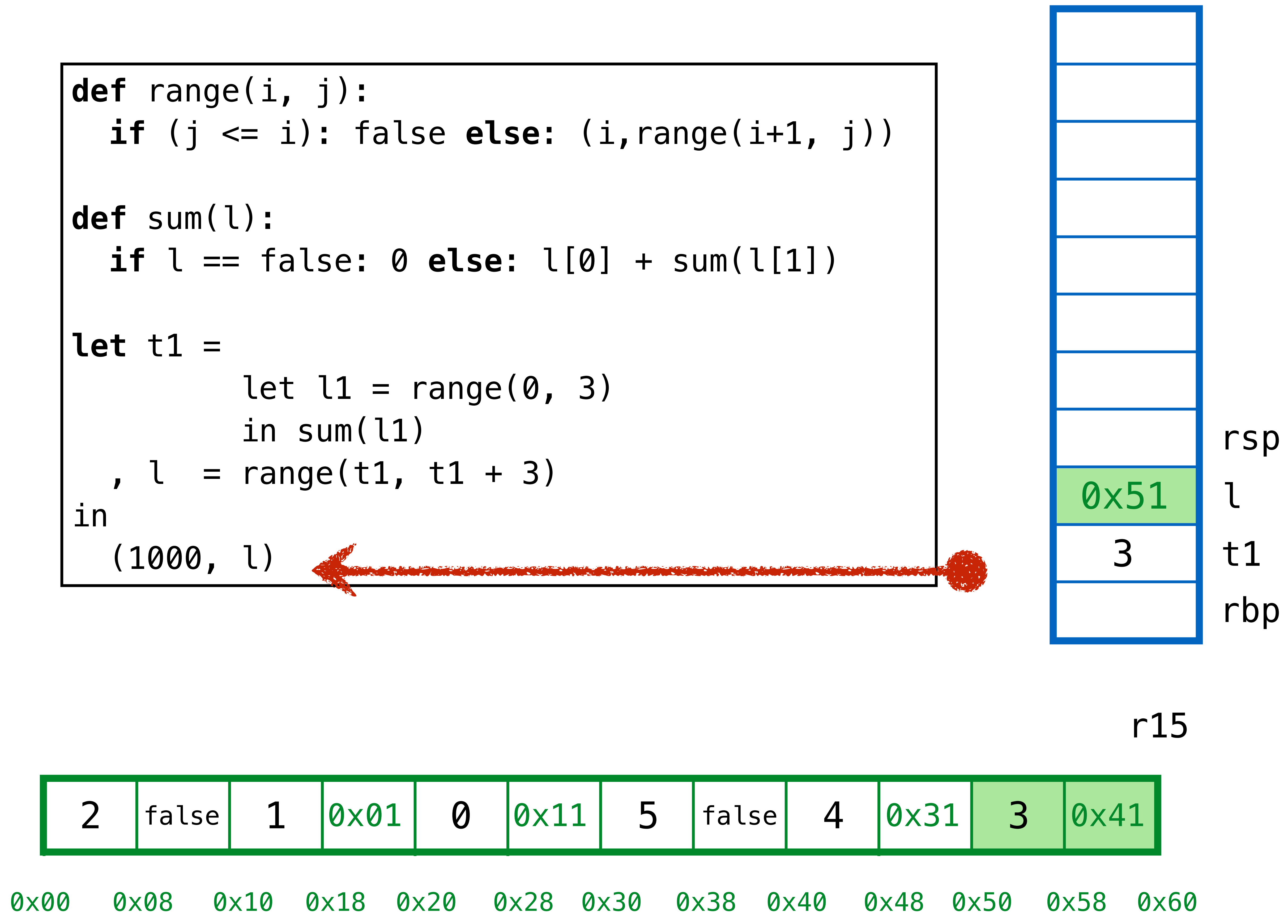


1. MARK live addrs

reachable from stack

ex4: recursive data

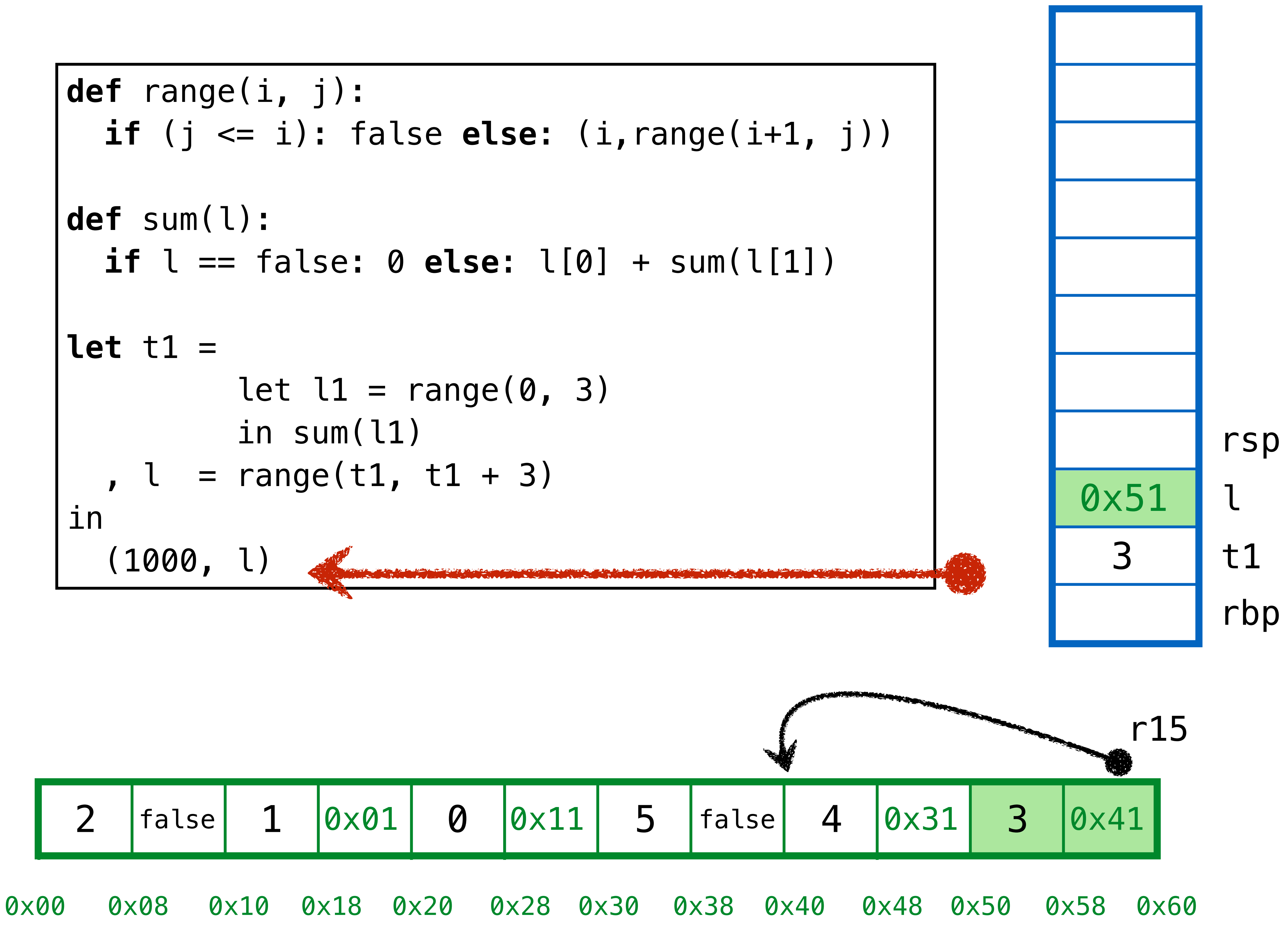
```
def range(i, j):
  if (j <= i): false else: (i,range(i+1, j))

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



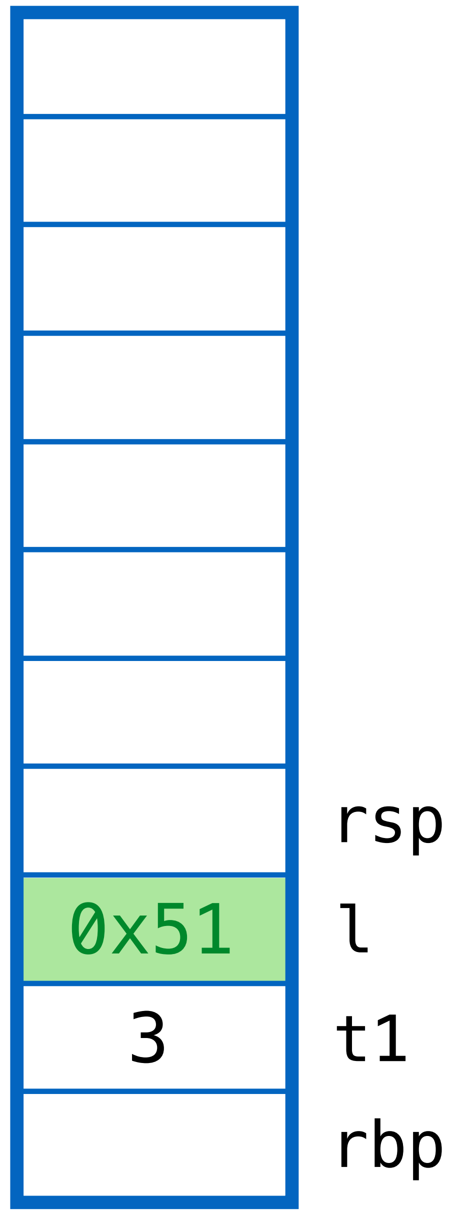
```
def range(i, j):
  if (j <= i): false else: (i,range(i+1, j))

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)
```

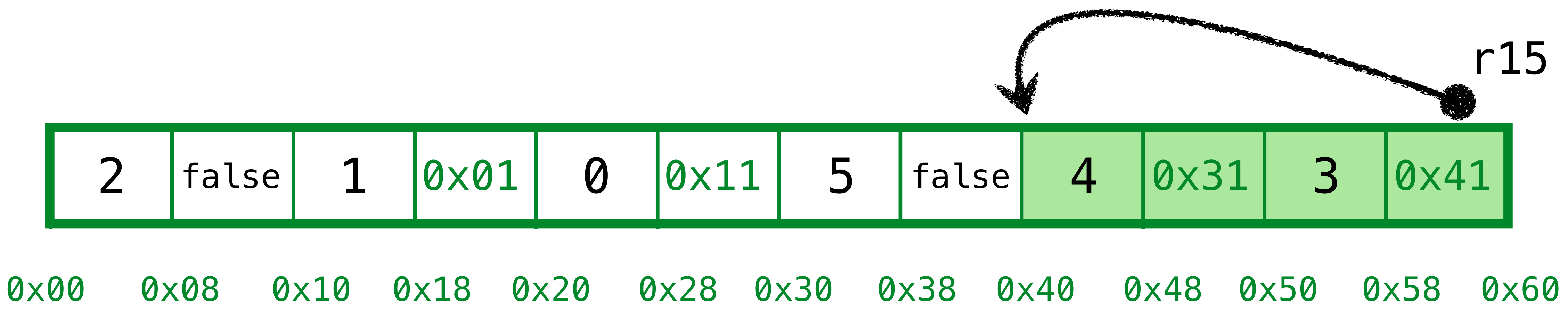
1. MARK live addrs

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)
```



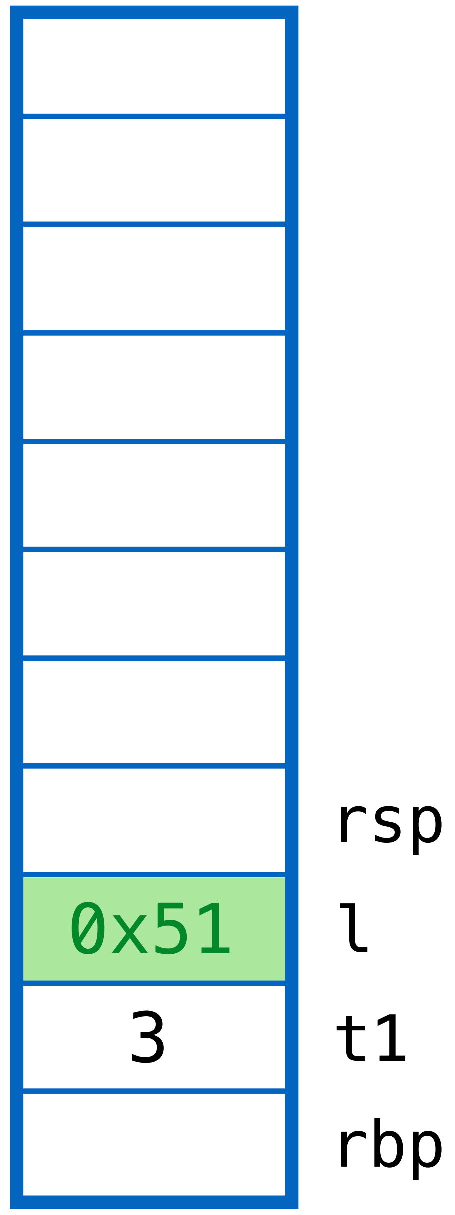


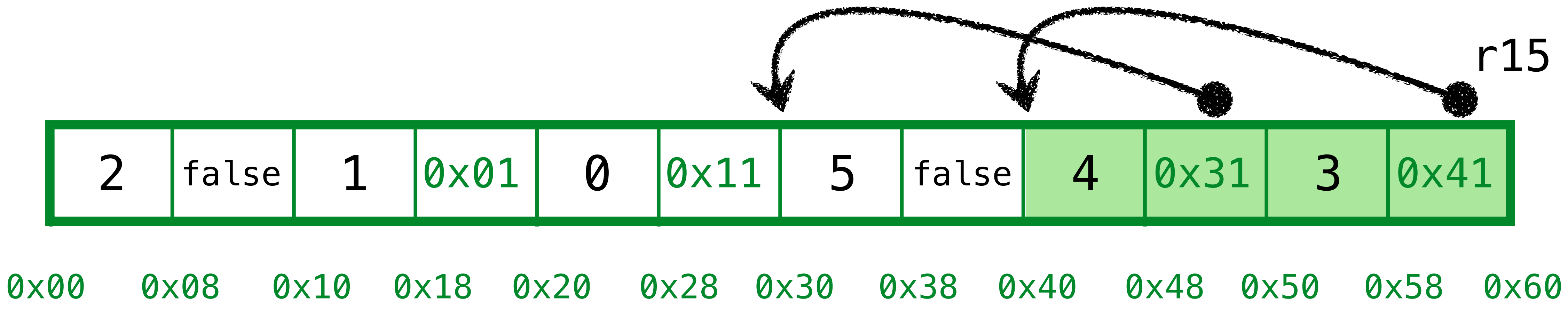
1. MARK live addrs

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)
```



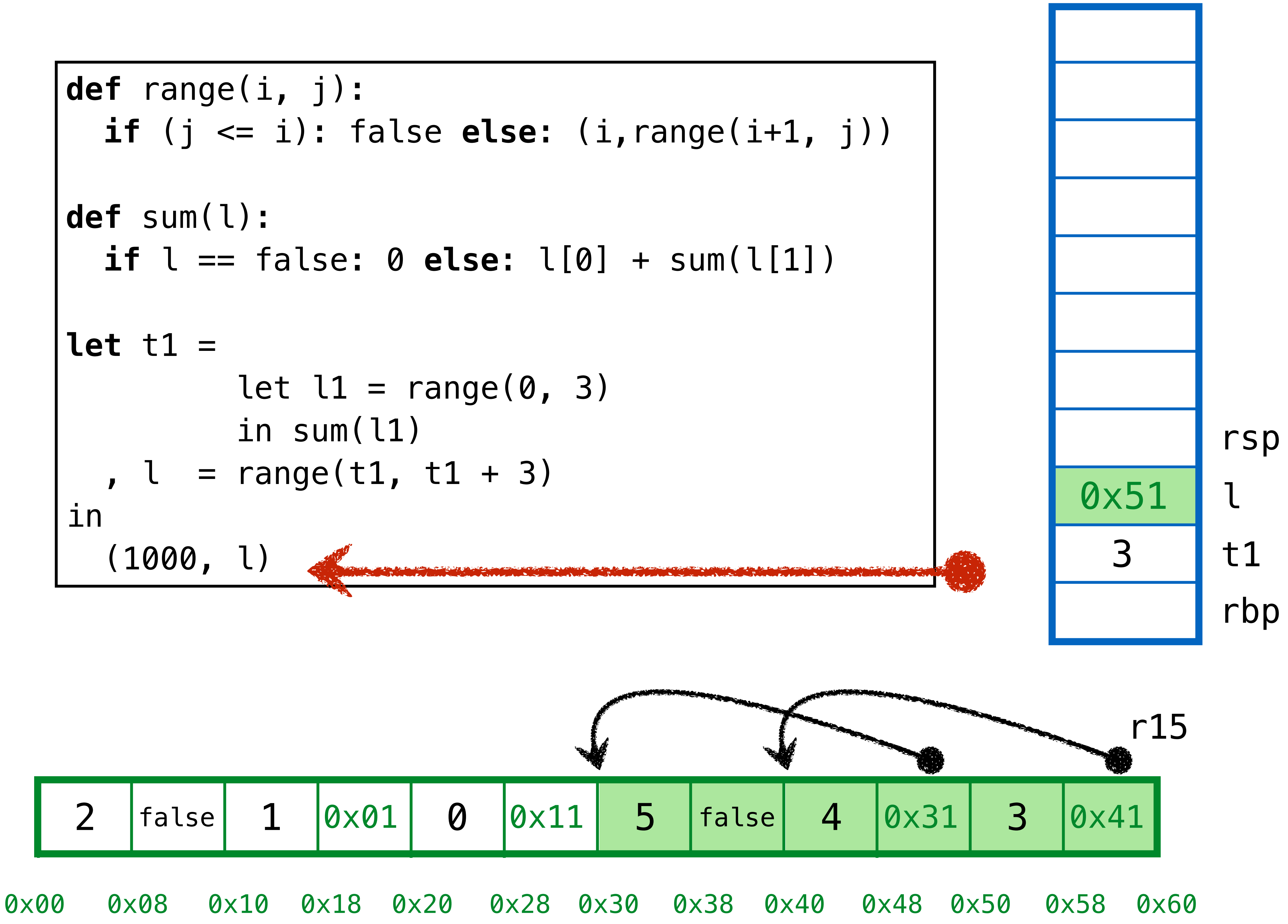


1. MARK live addrs

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)
```

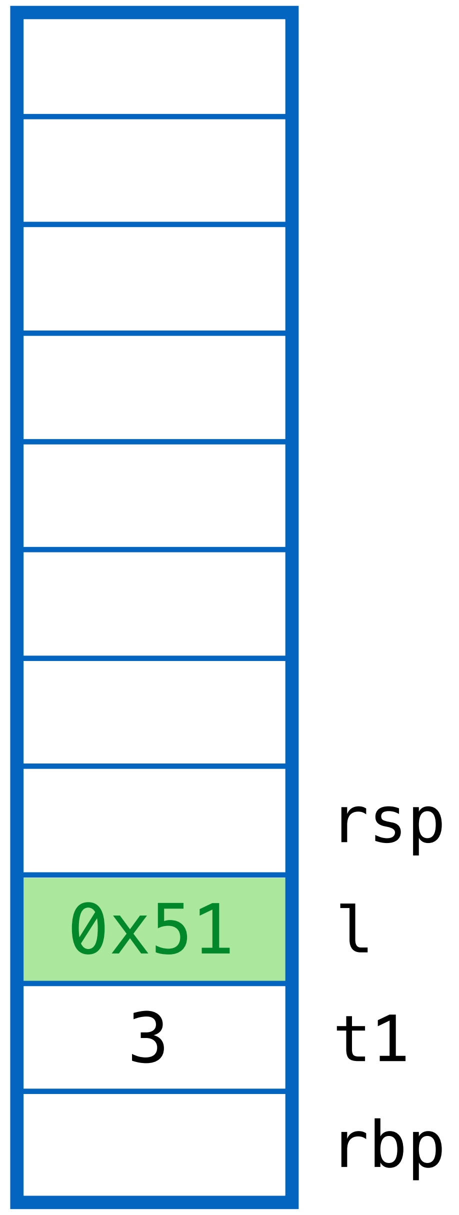


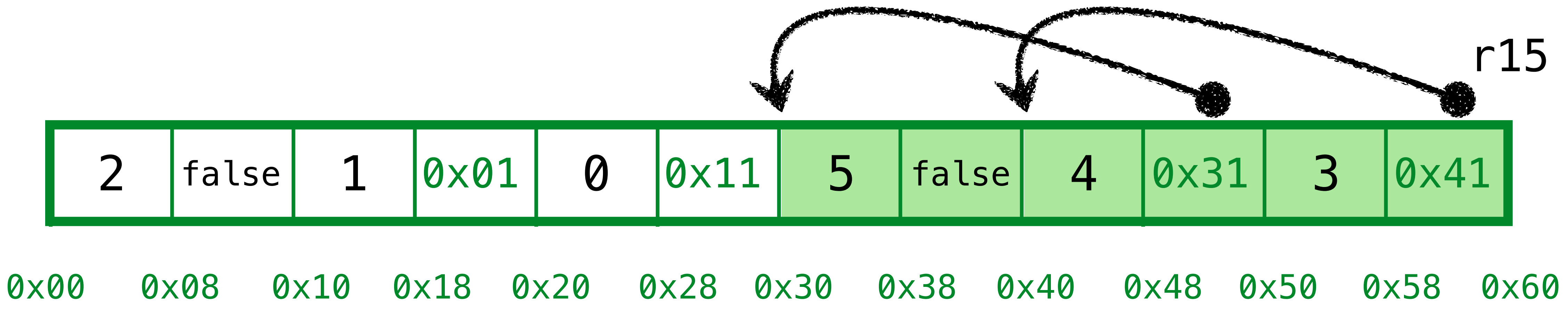
1. MARK live addrs

reachable from stack

ex4: recursive data

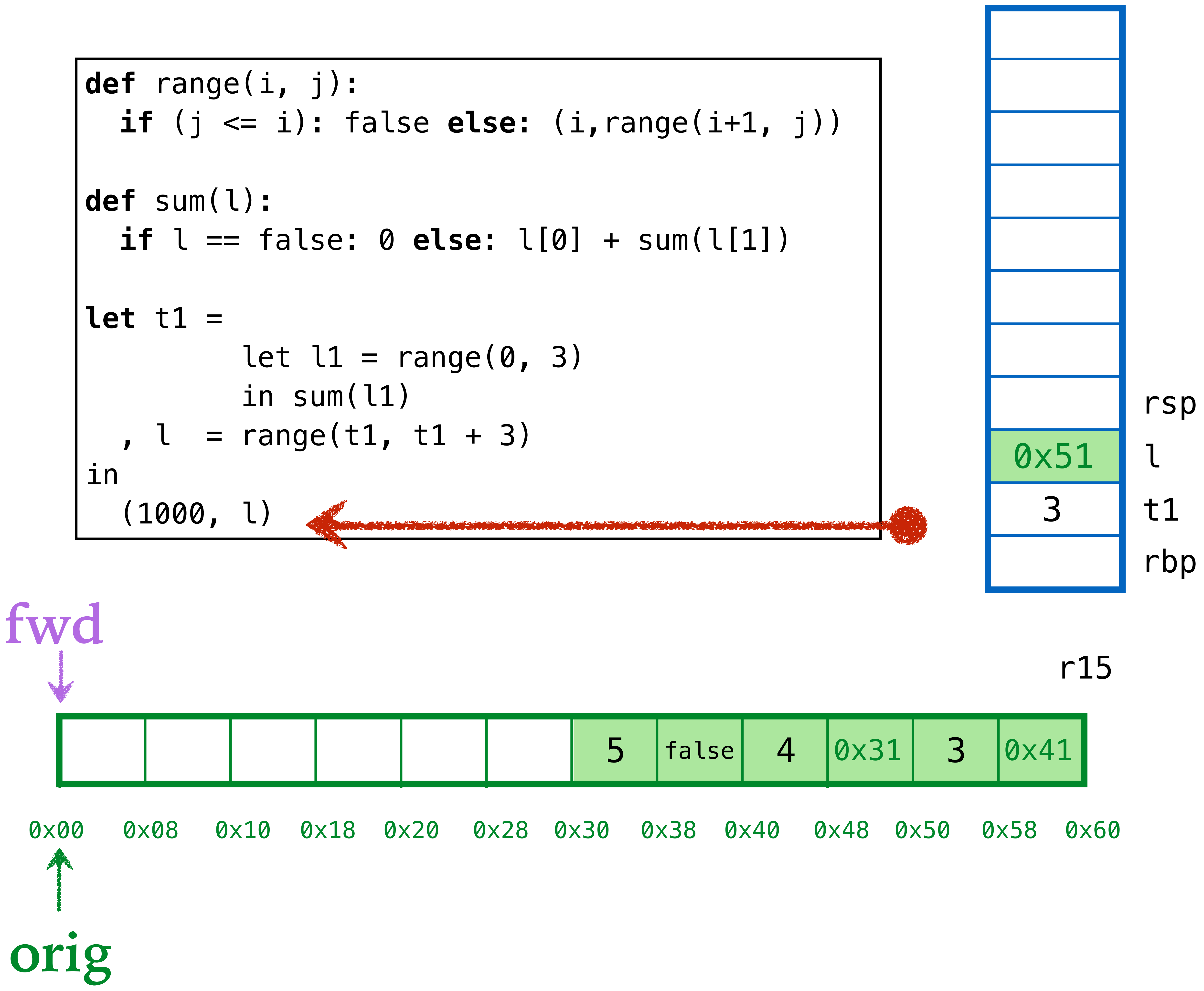
```
def range(i, j):
  if (j <= i): false else: (i,range(i+1, j))

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)
```





Done!

ex4: recursive data



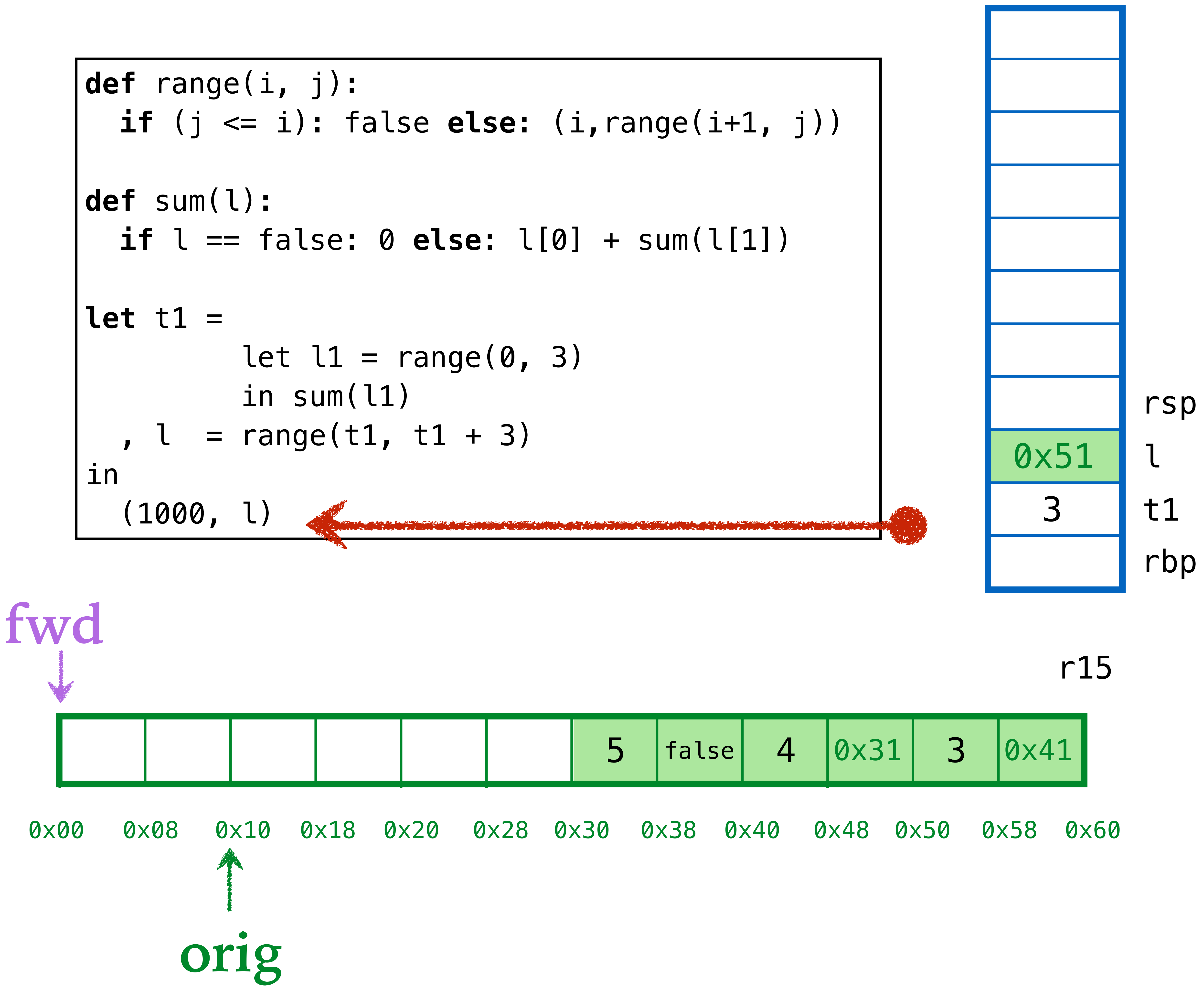
```
def range(i, j):
  if (j <= i): false else: (i,range(i+1, j))

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)
```

	rsp
0x51	l
3	t1
	rbp

0x00	0x08	0x10	0x18	0x20	0x28	0x30	0x38	0x40	0x48	0x50	0x58	0x60
						5	false	4	0x31	3	0x41	

2. Compute FORWARD addrs

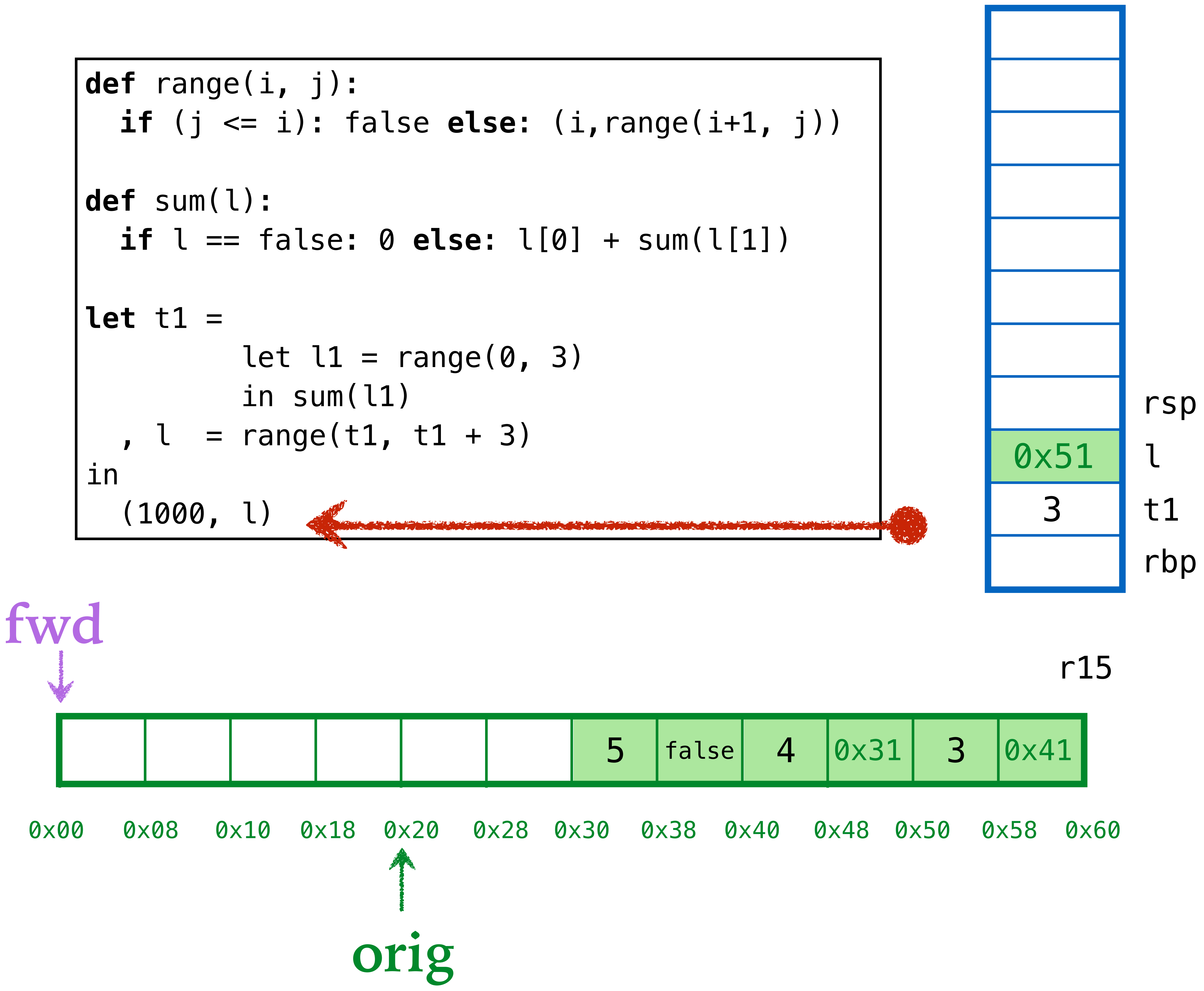
ex4: recursive data



```
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)
```

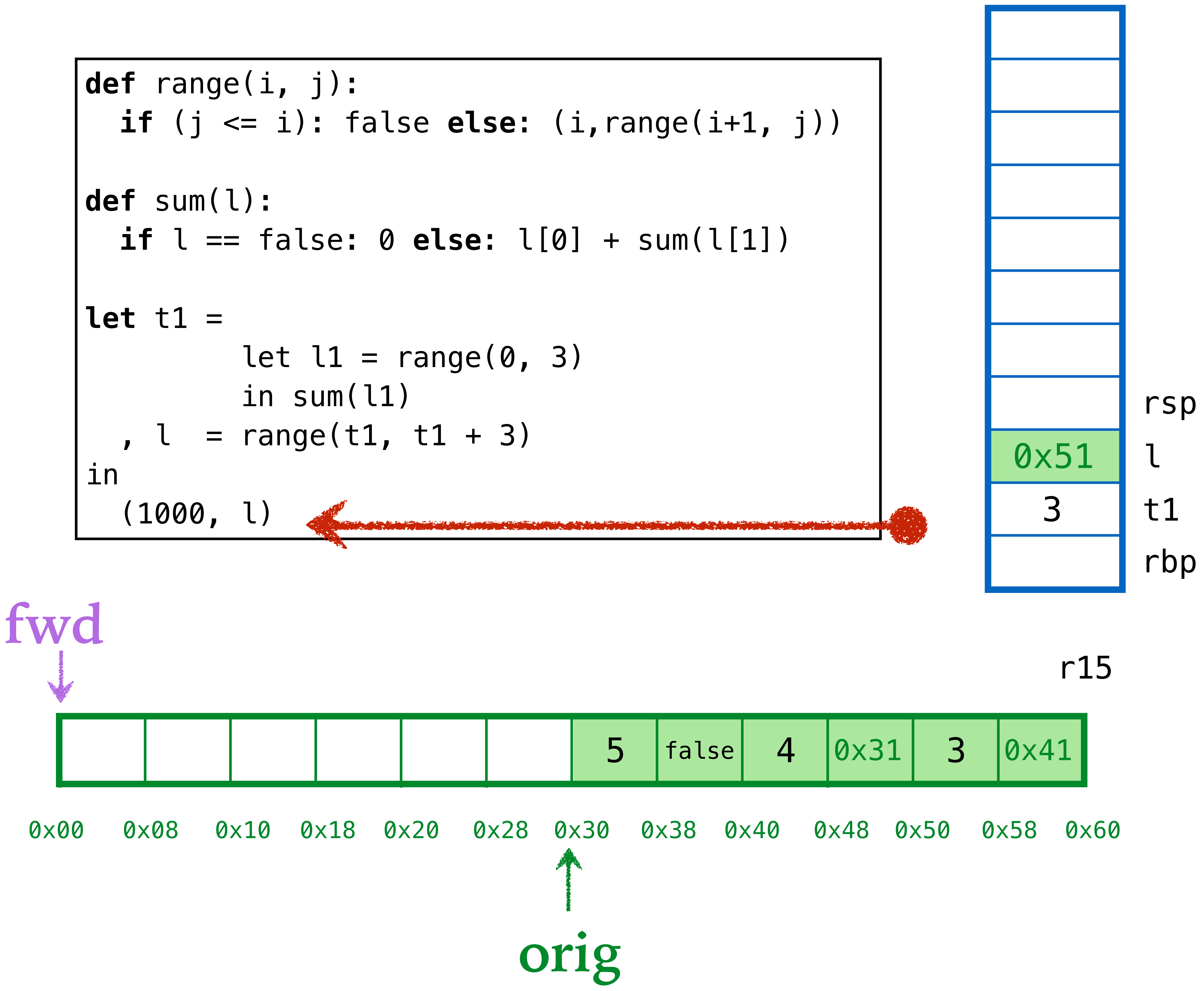
0x00	0x08	0x10	0x18	0x20	0x28	0x30	0x38	0x40	0x48	0x50	0x58
						5	false	4	0x31	3	0x41

0x60

2. Compute FORWARD addrs

ex4: recursive data

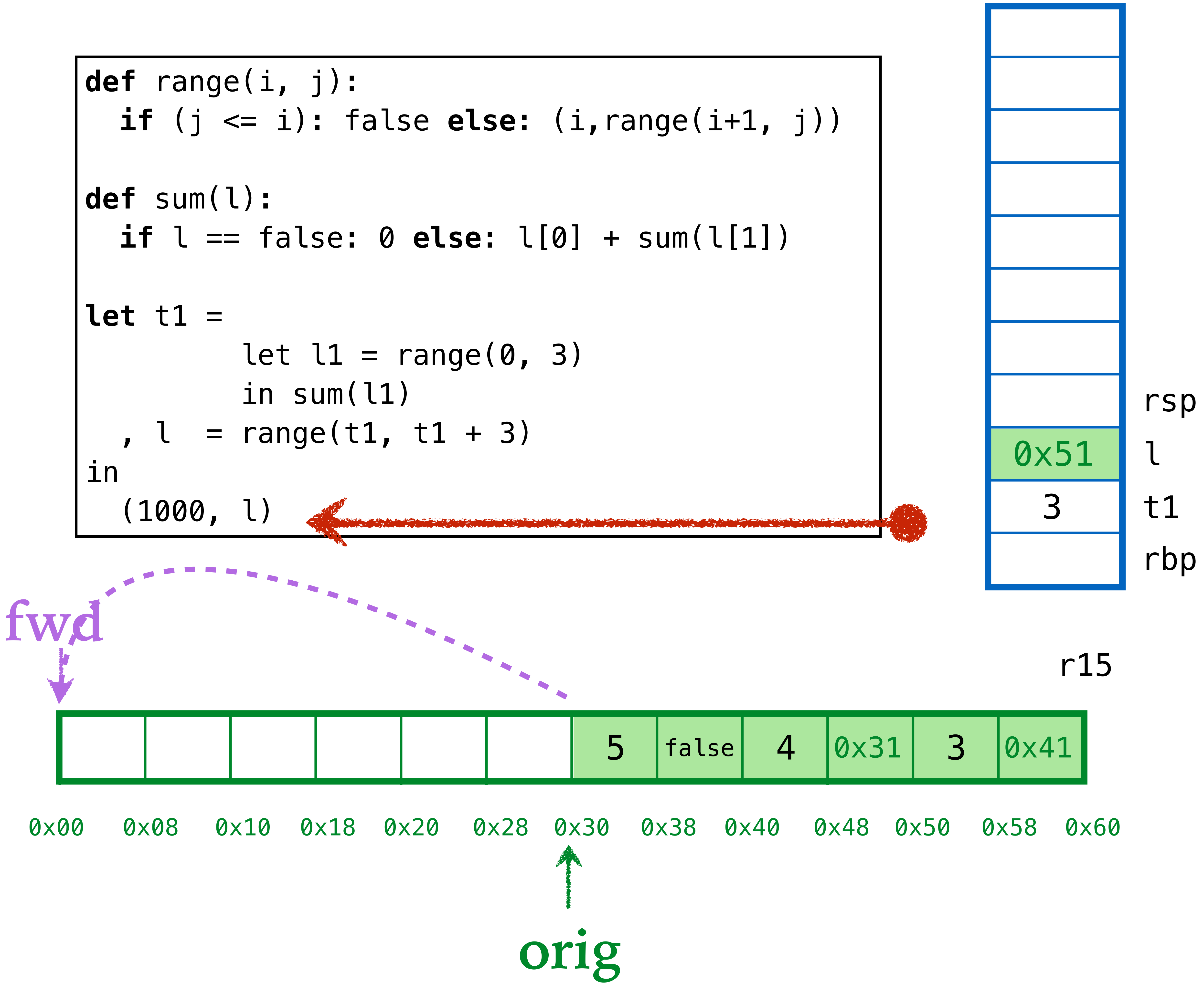
```
def range(i, j):
  if (j <= i): false else: (i,range(i+1, j))

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

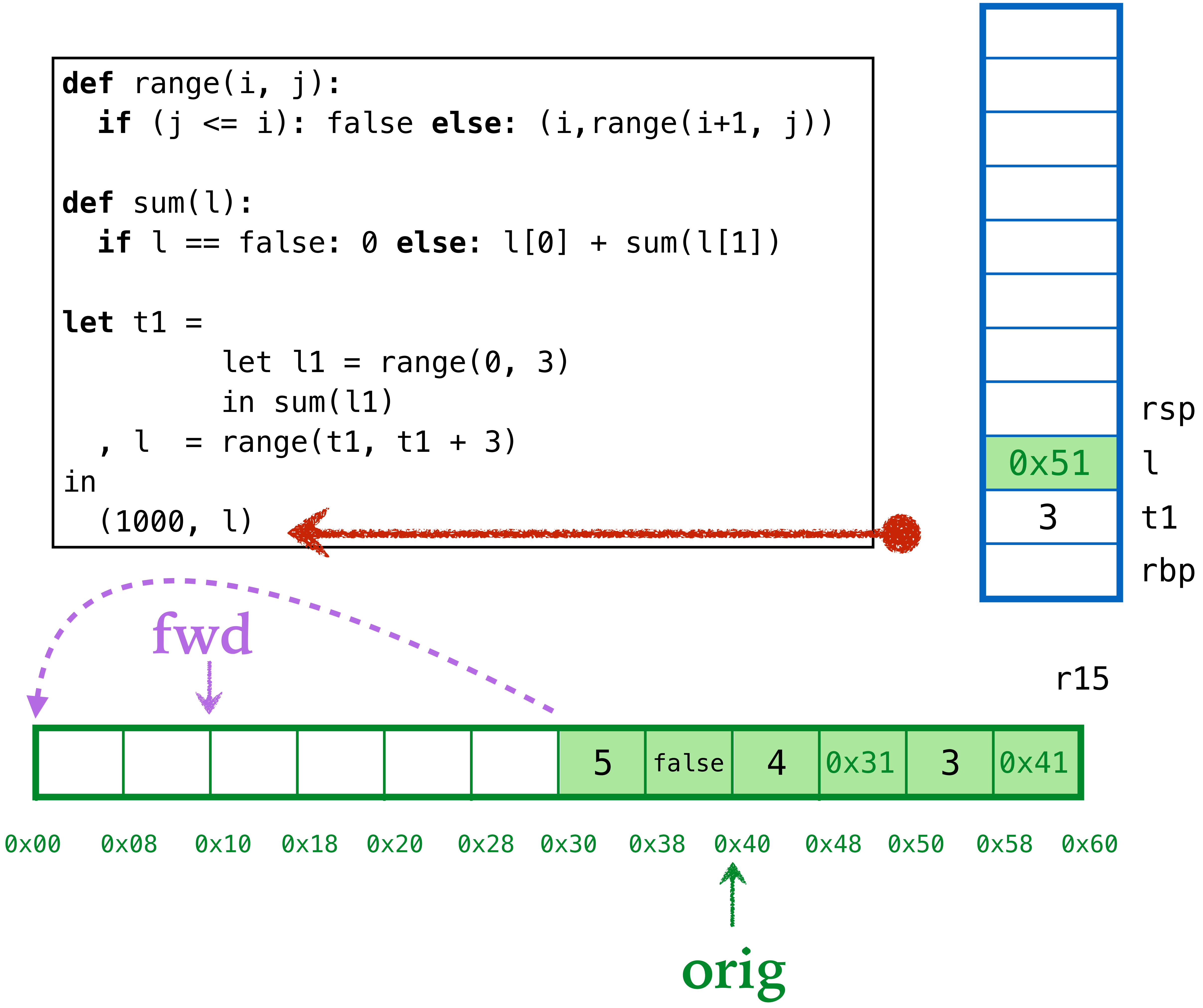
```
def range(i, j):
  if (j <= i): false else: (i,range(i+1, j))

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

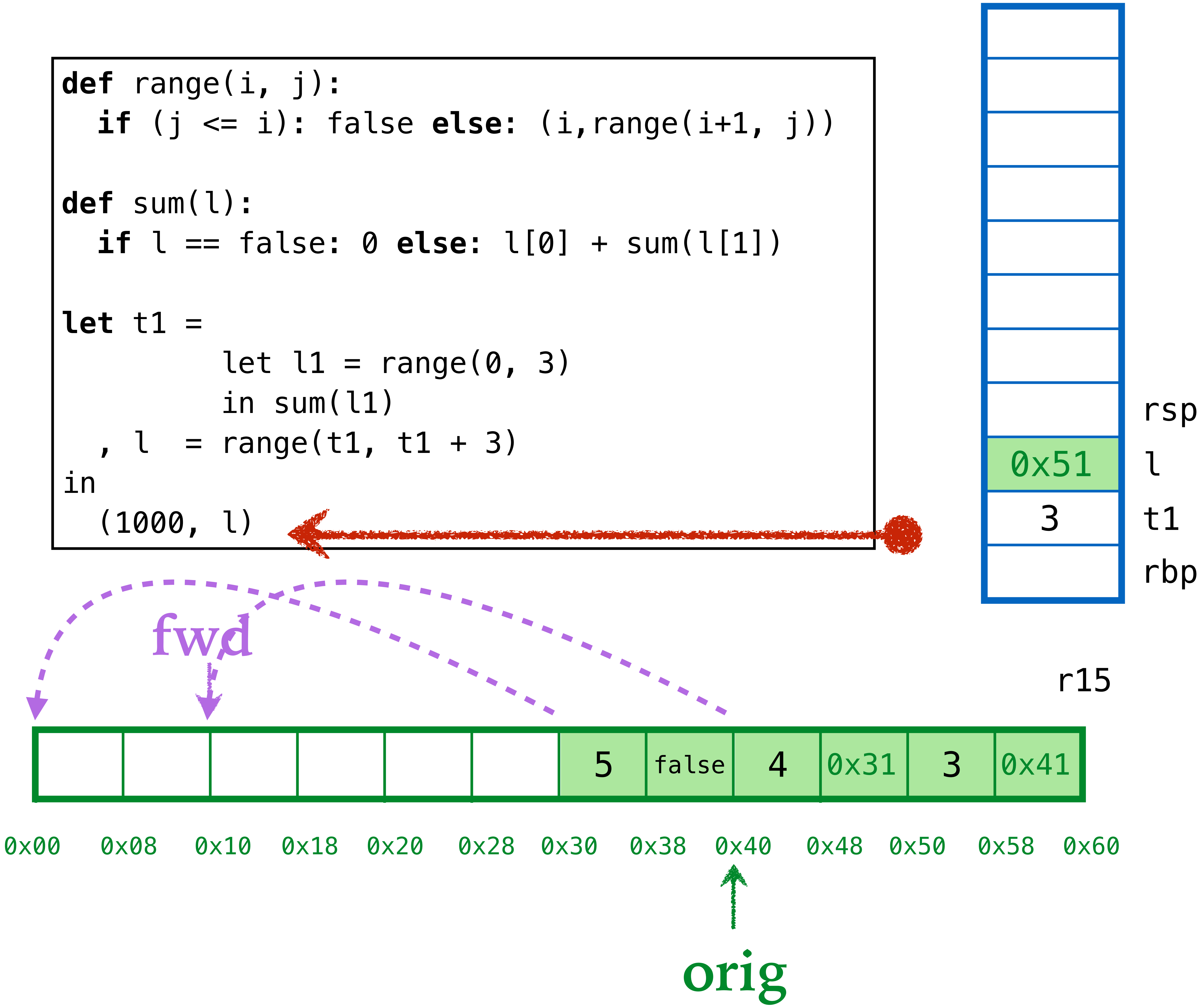
ex4: recursive data

```
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

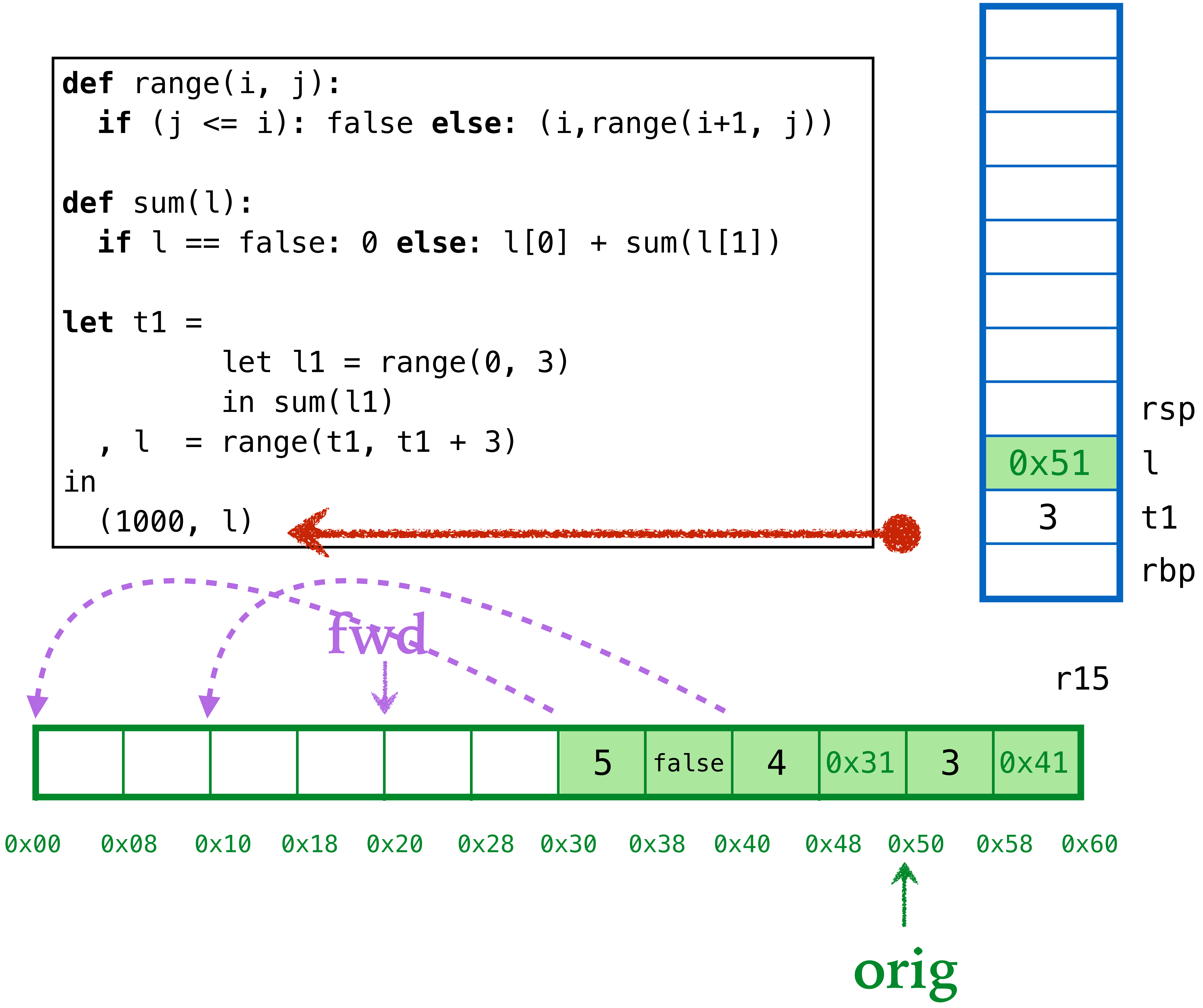


```
def range(i, j):
  if (j <= i): false else: (i,range(i+1, j))

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

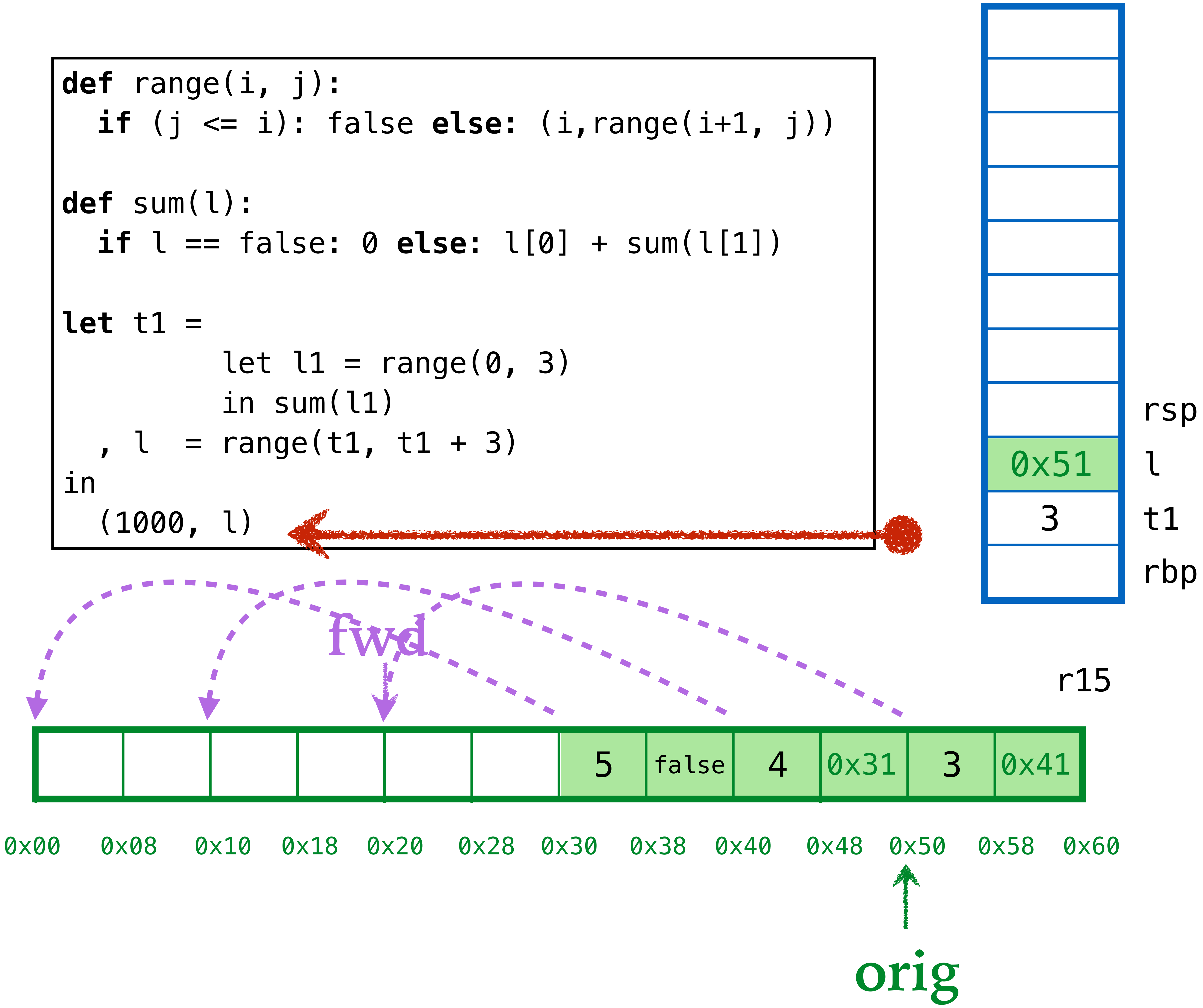
ex4: recursive data

```
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

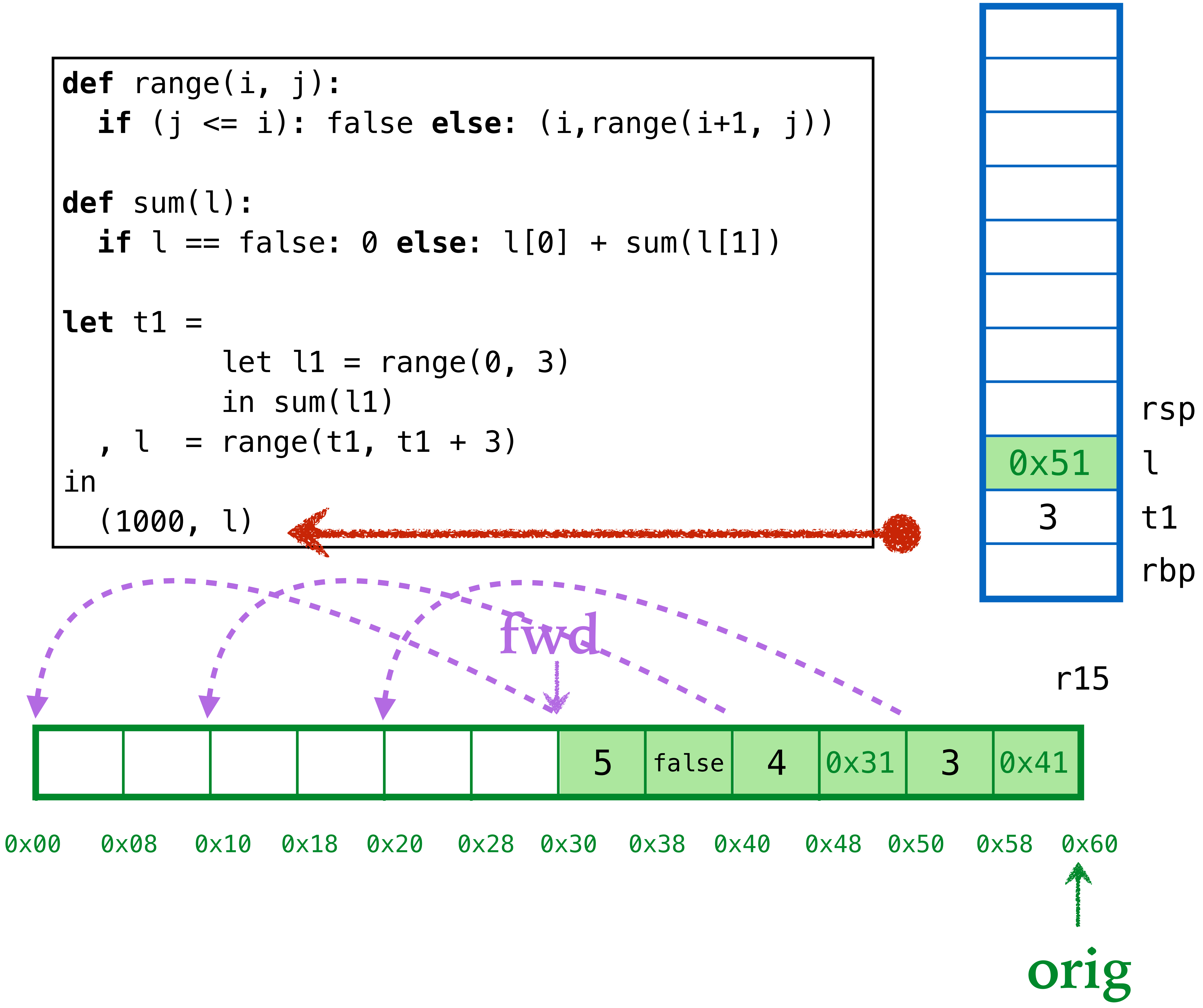
ex4: recursive data



```
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

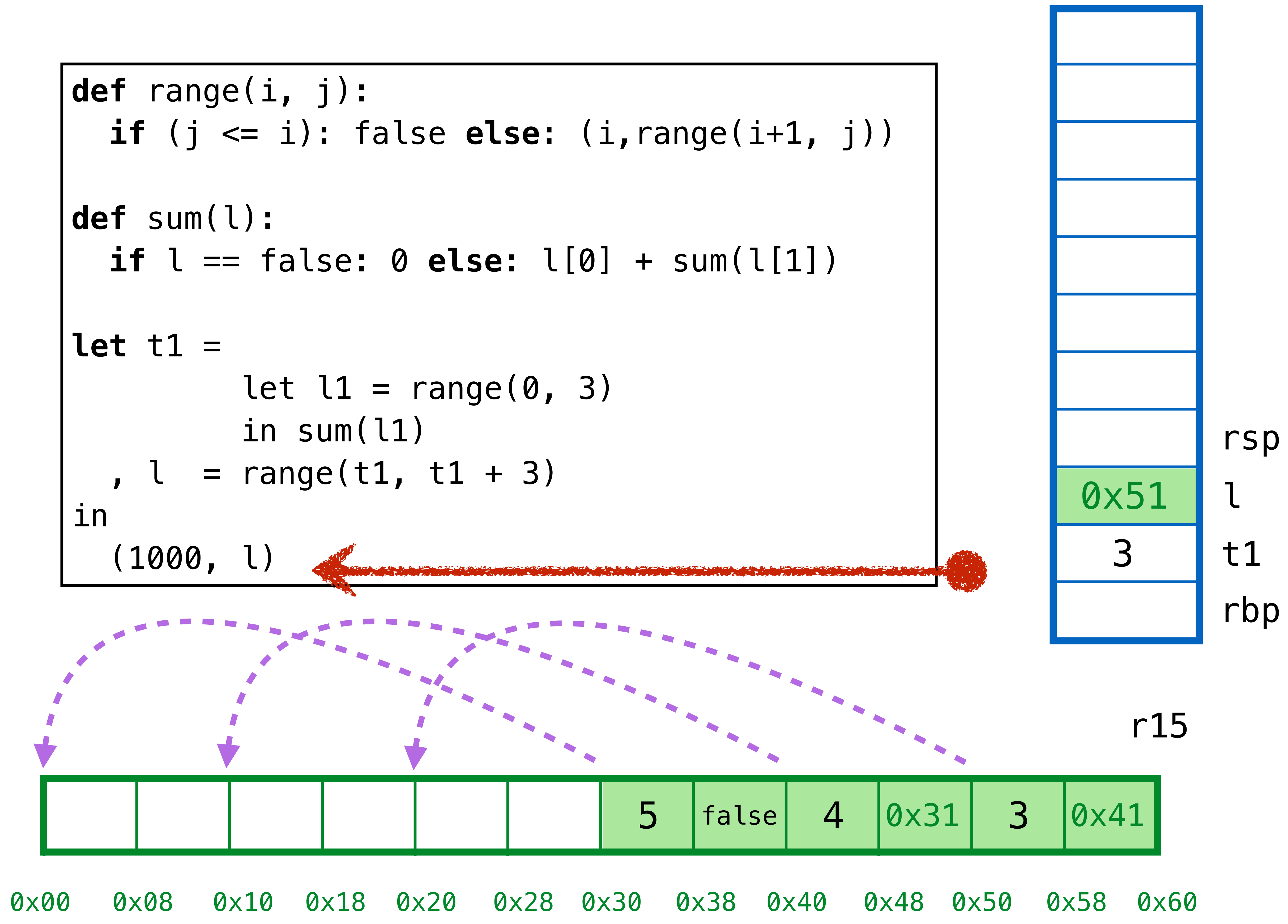


```
def range(i, j):
  if (j <= i): false else: (i,range(i+1, j))

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

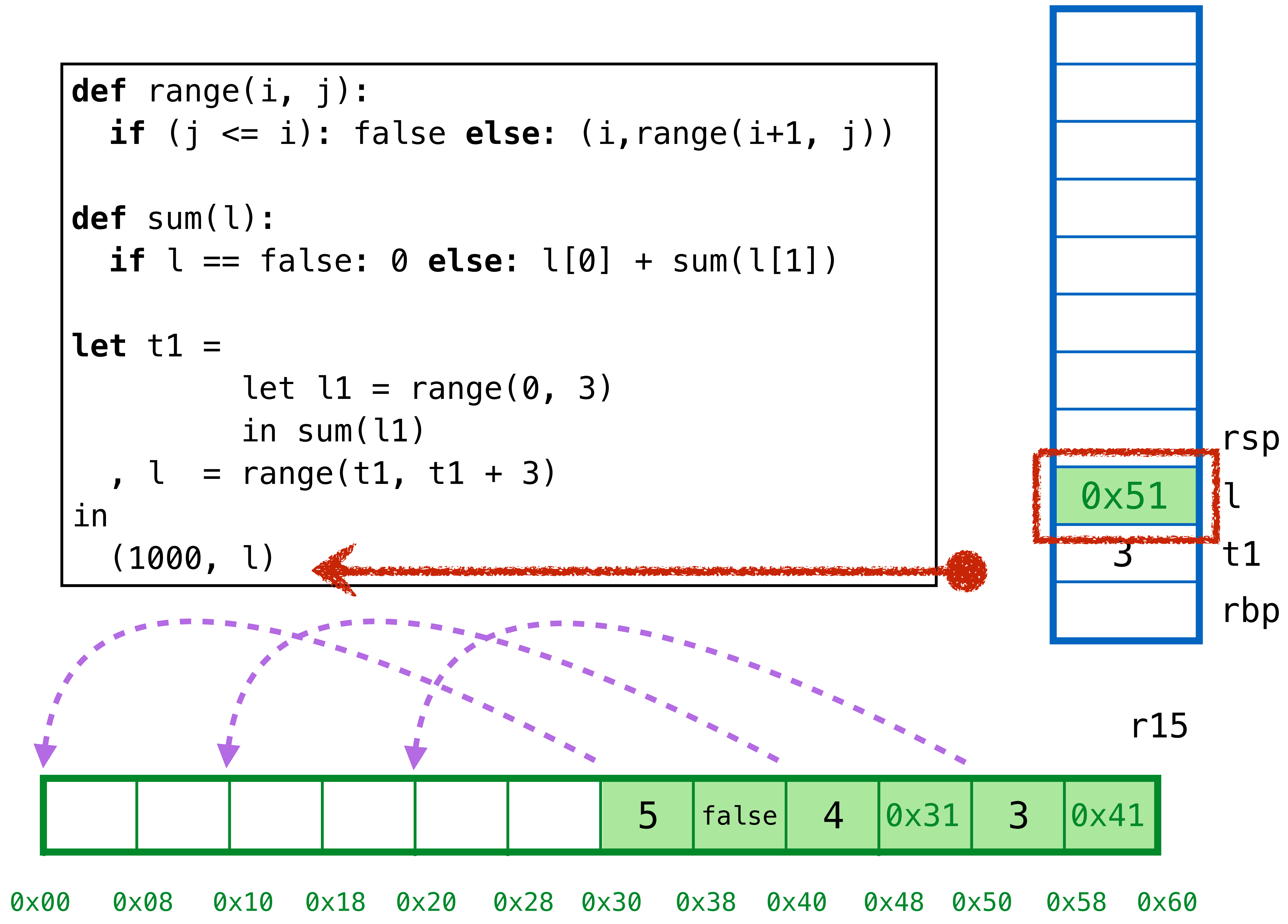
```
def range(i, j):
  if (j <= i): false else: (i,range(i+1, j))

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?

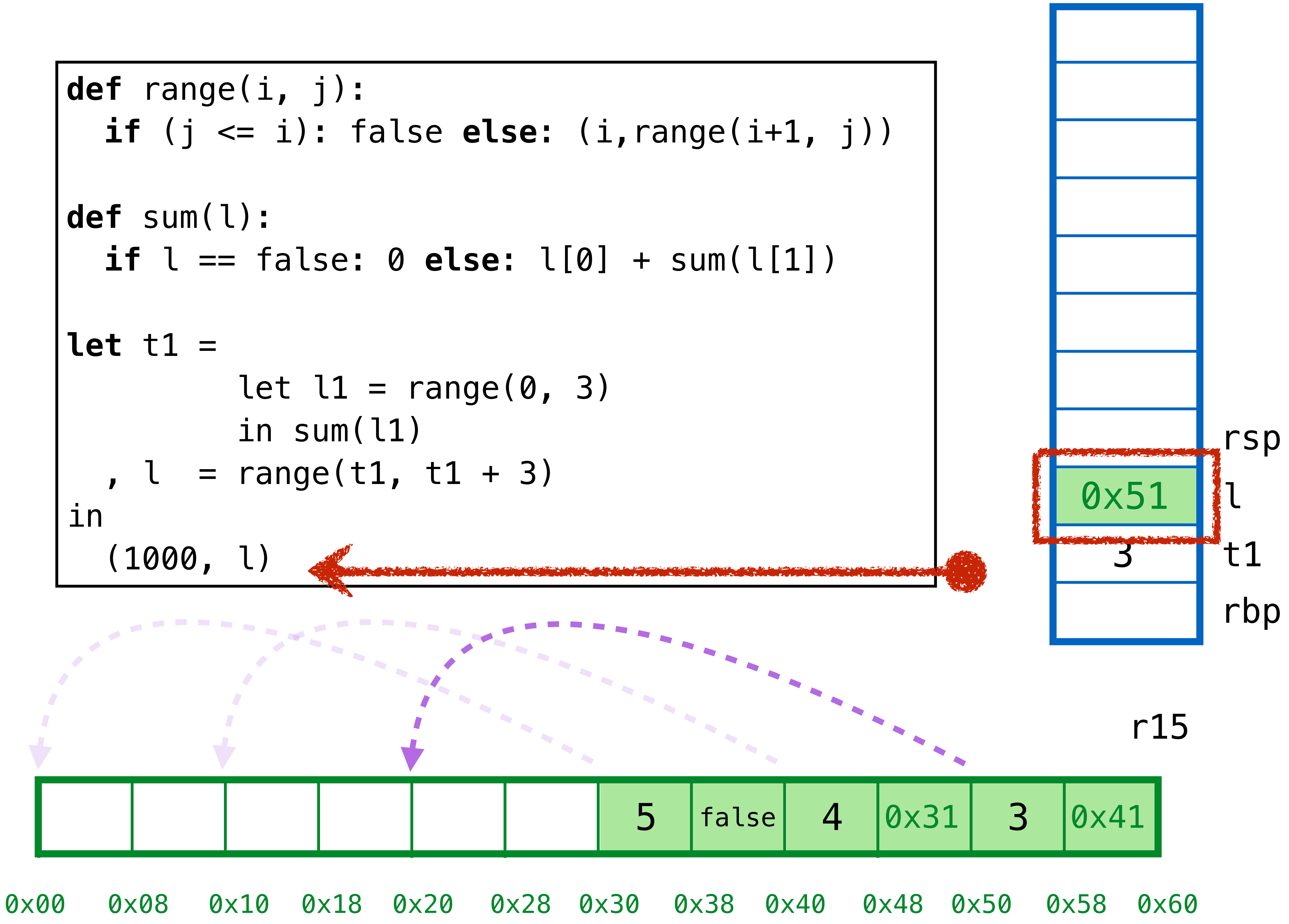
ex4: recursive data



```
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)
```

3. REDIRECT addrs on stack

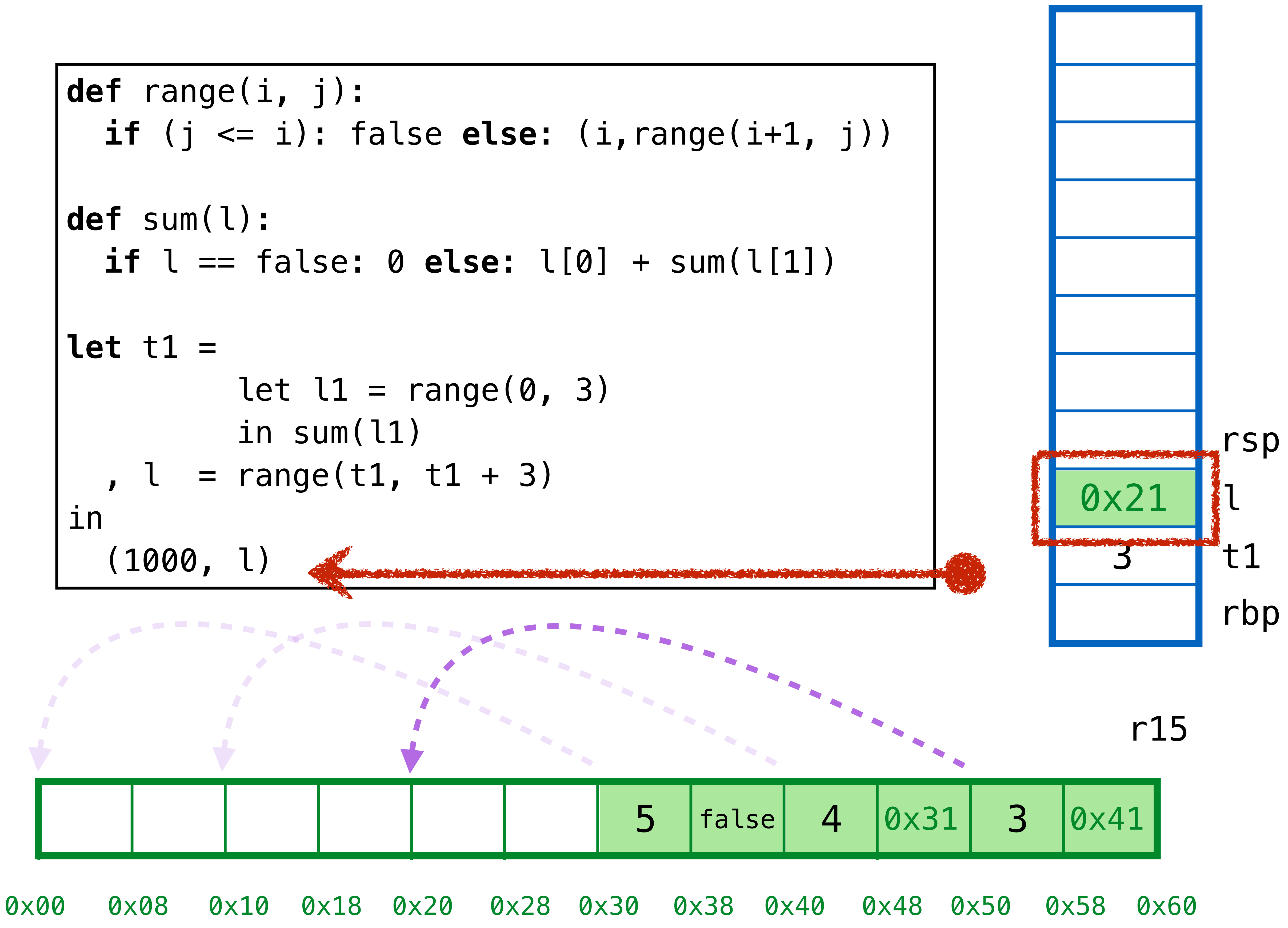
ex4: recursive data



```
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

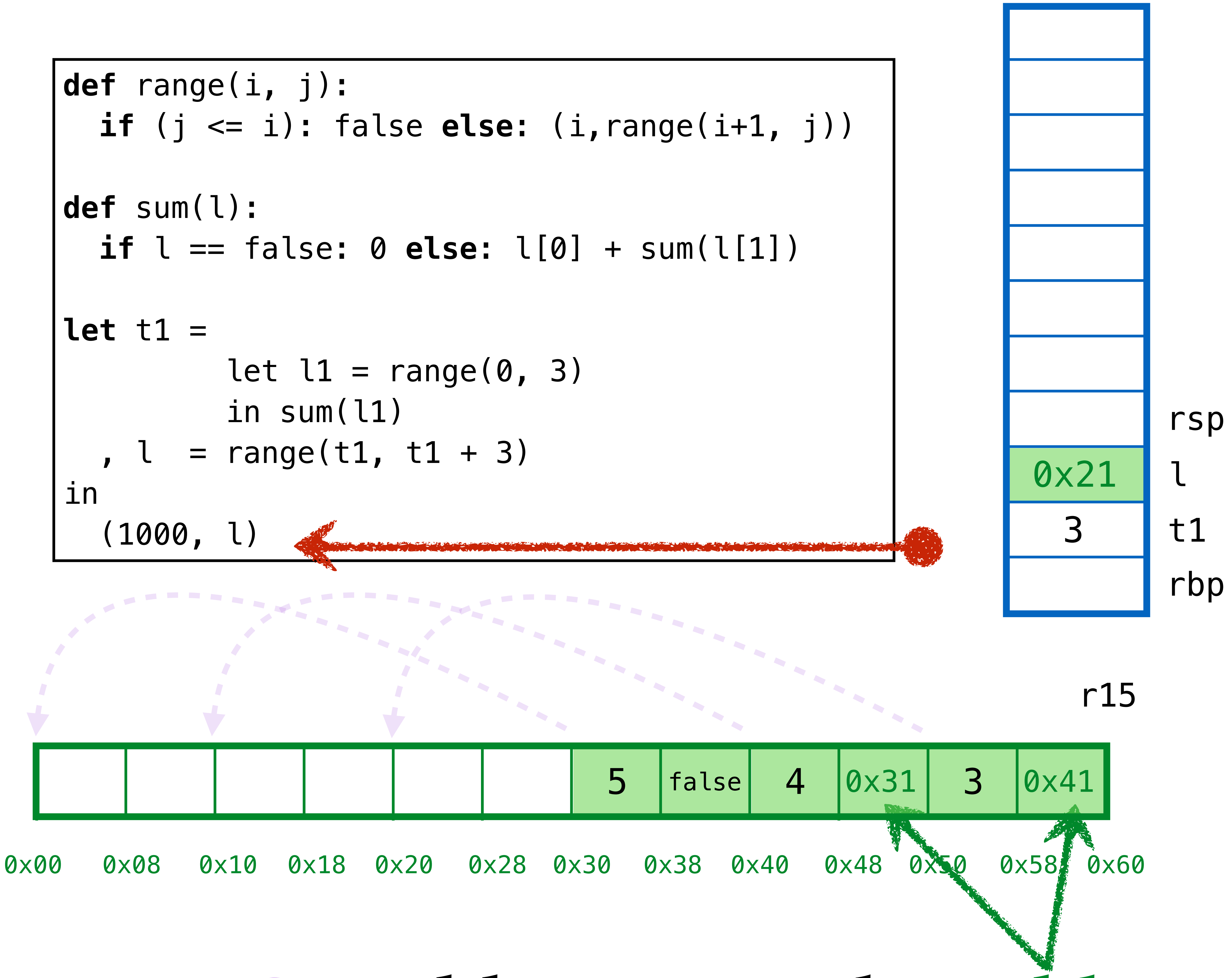
ex4: recursive data



```
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

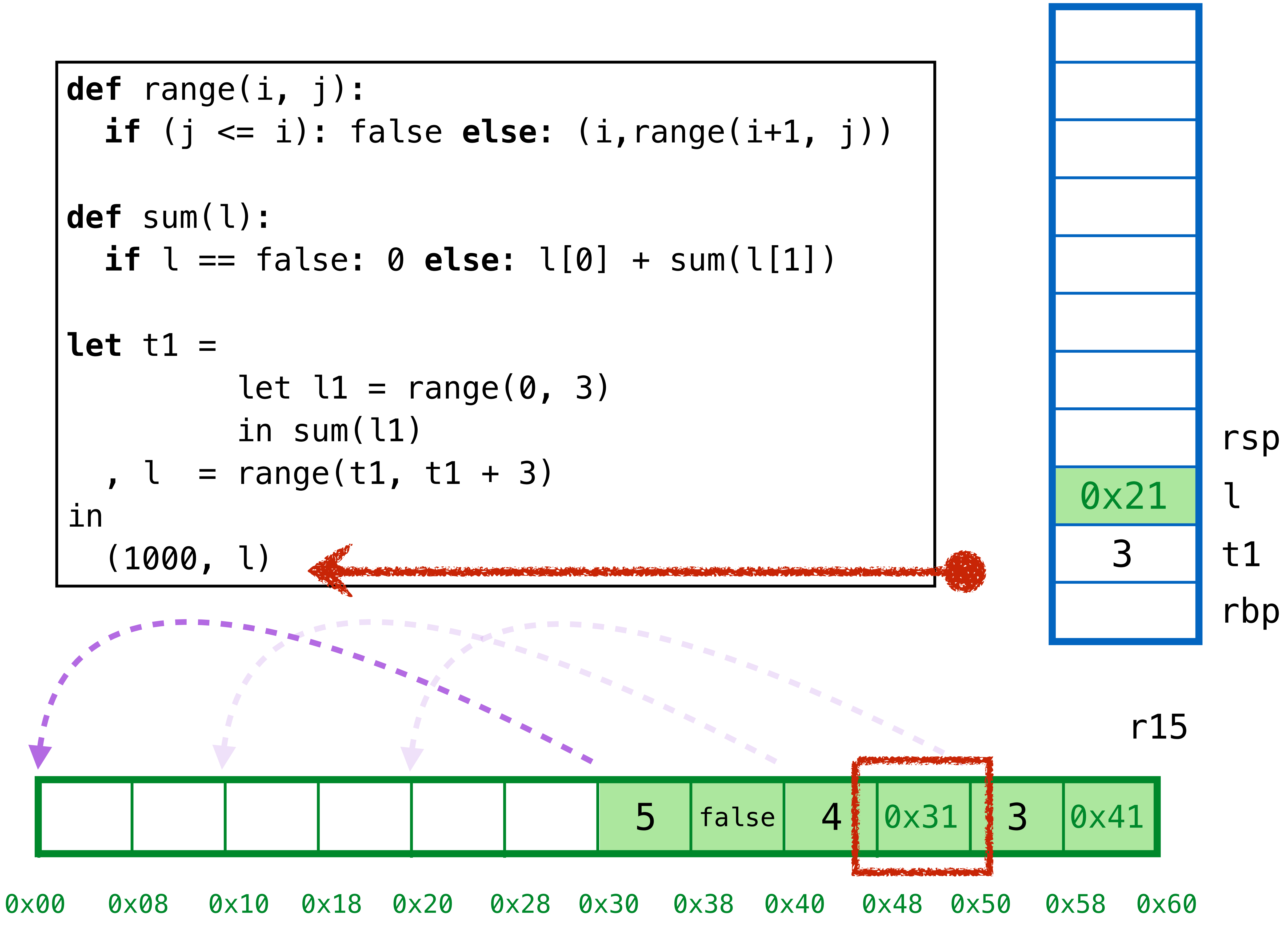
ex4: recursive data



```
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

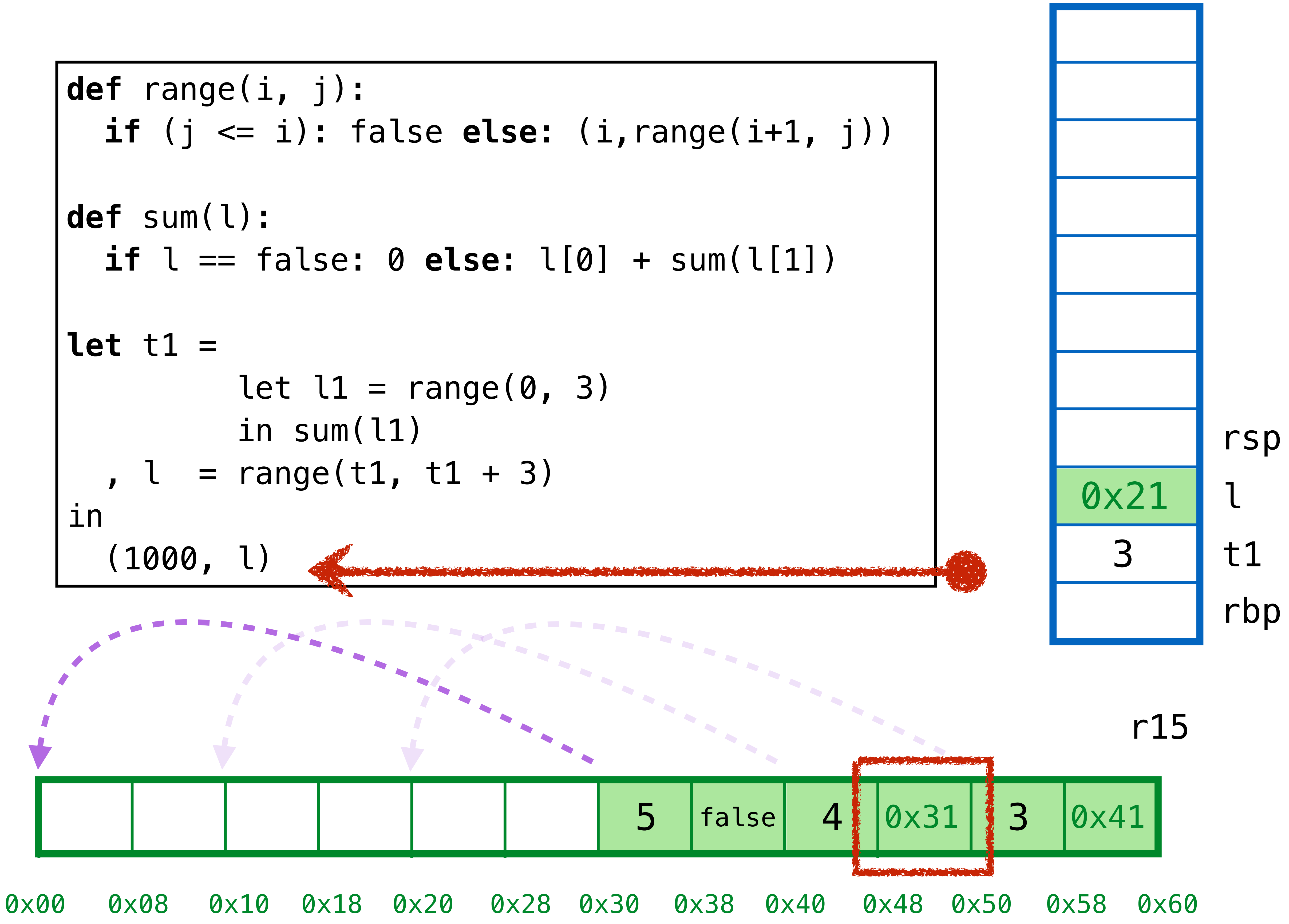


```
def range(i, j):
  if (j <= i): false else: (i,range(i+1, j))

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

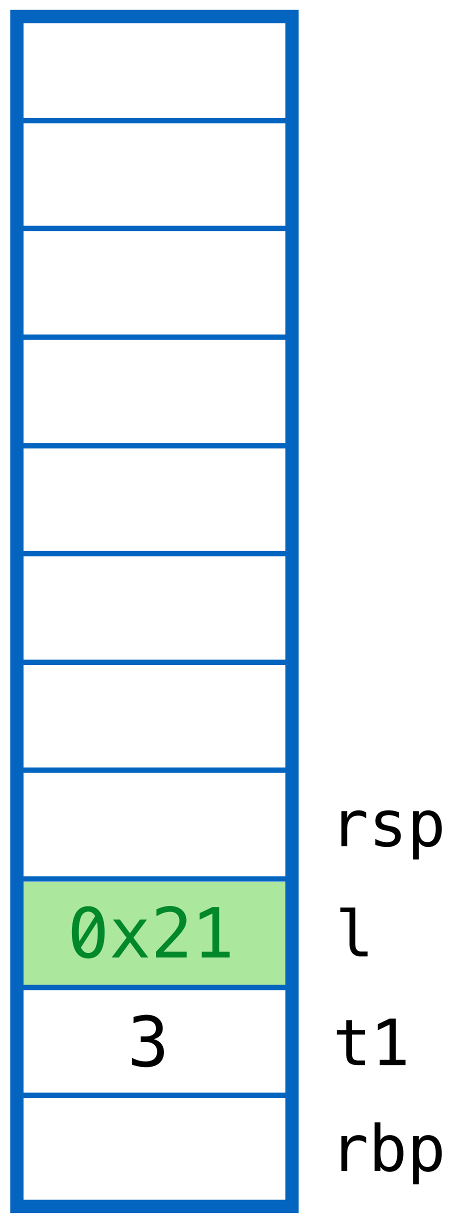
```
def range(i, j):
  if (j <= i): false else: (i,range(i+1, j))

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)
```

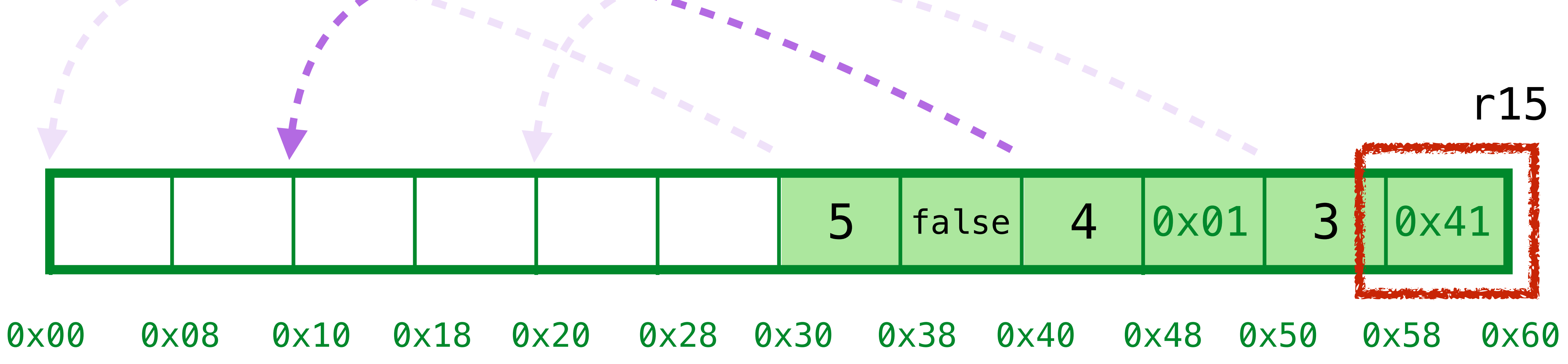


3. REDIRECT addrs on stack and heap!

ex4: recursive data

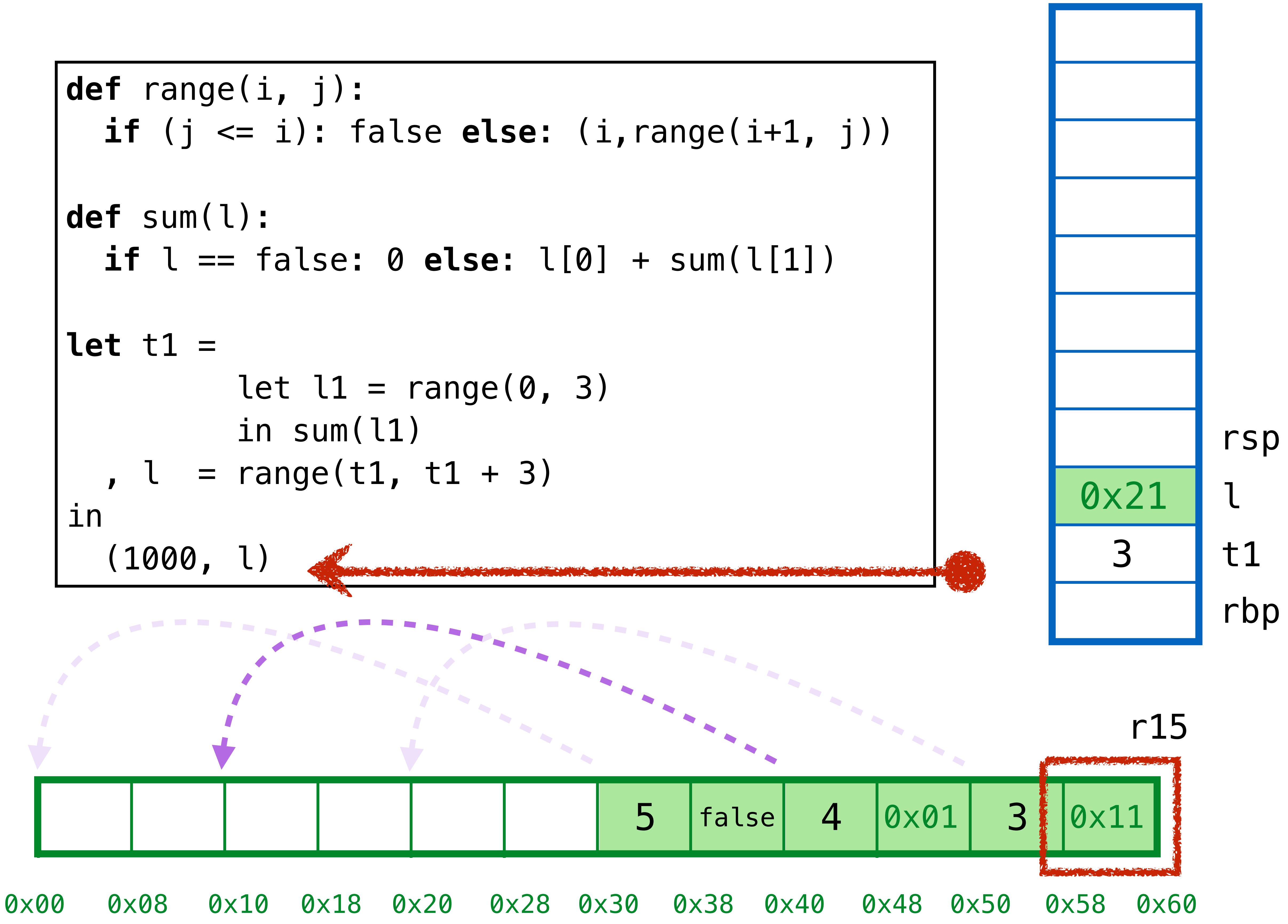
```
def range(i, j):
  if (j <= i): false else: (i,range(i+1, j))

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!

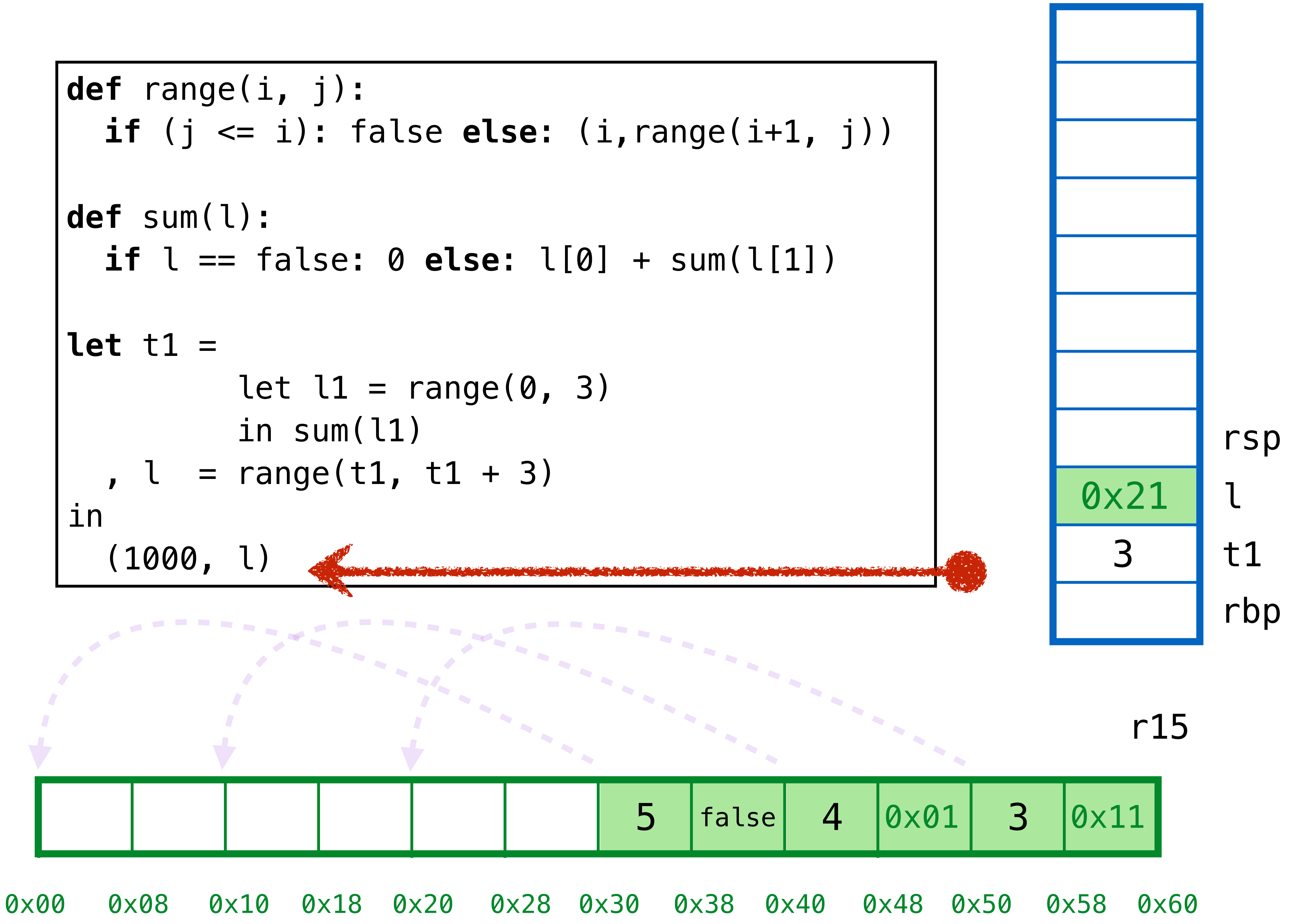
ex4: recursive data



```
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)
```

3. REDIRECT addrs on stack and heap!

ex4: recursive data



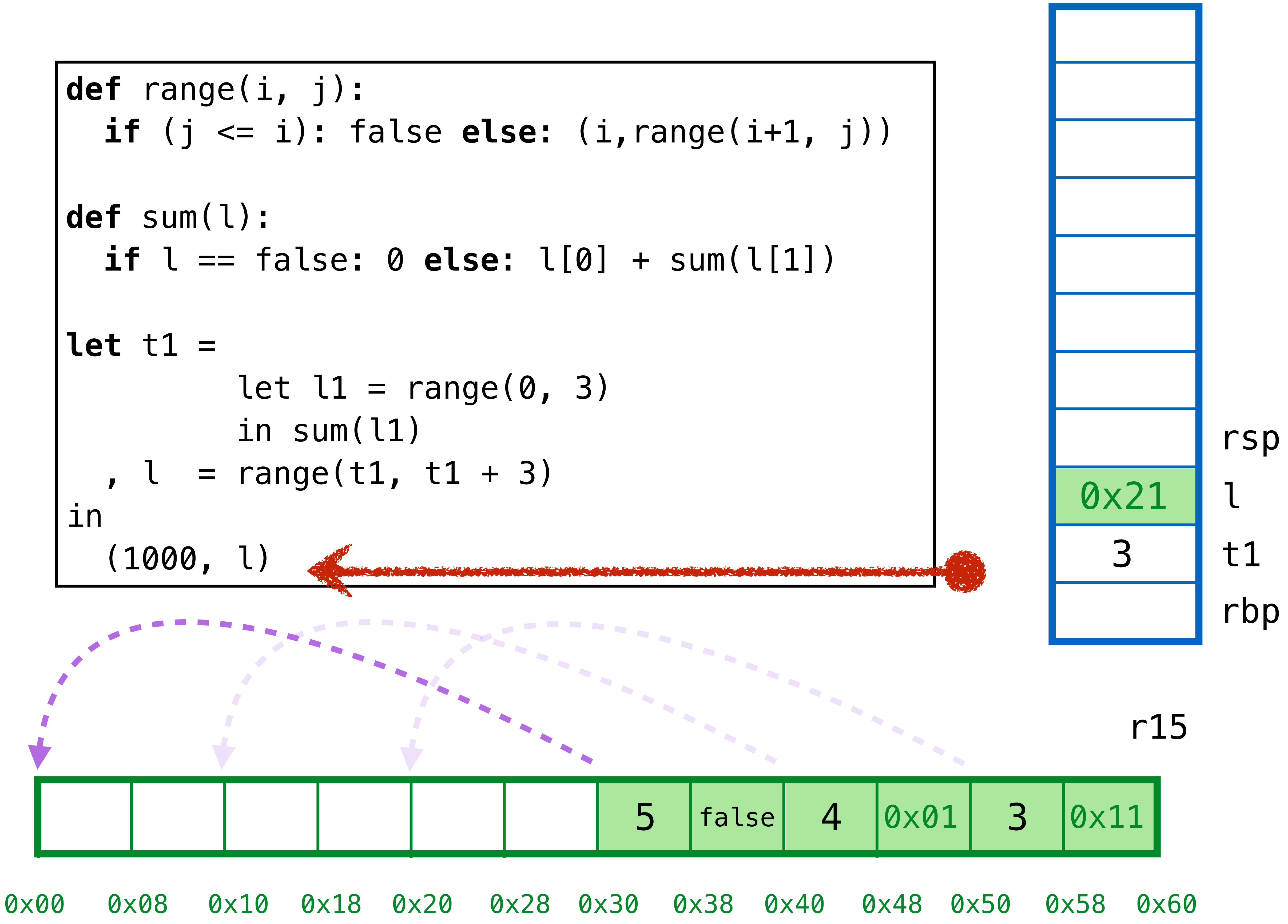
```
def range(i, j):
  if (j <= i): false else: (i,range(i+1, j))

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)
```

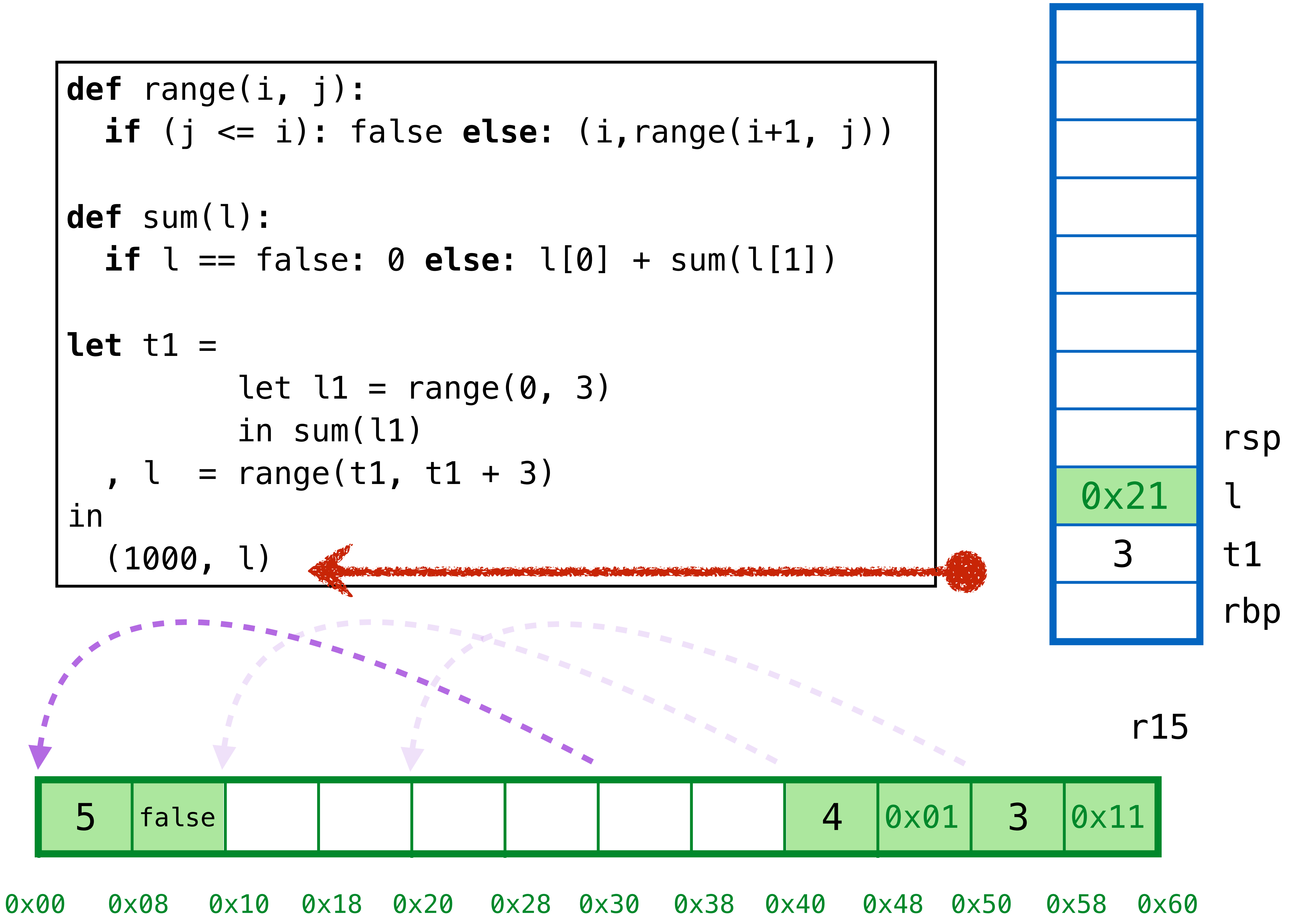


4. COMPACT cells on heap

Copy cell to forward addr!

ex4: recursive data

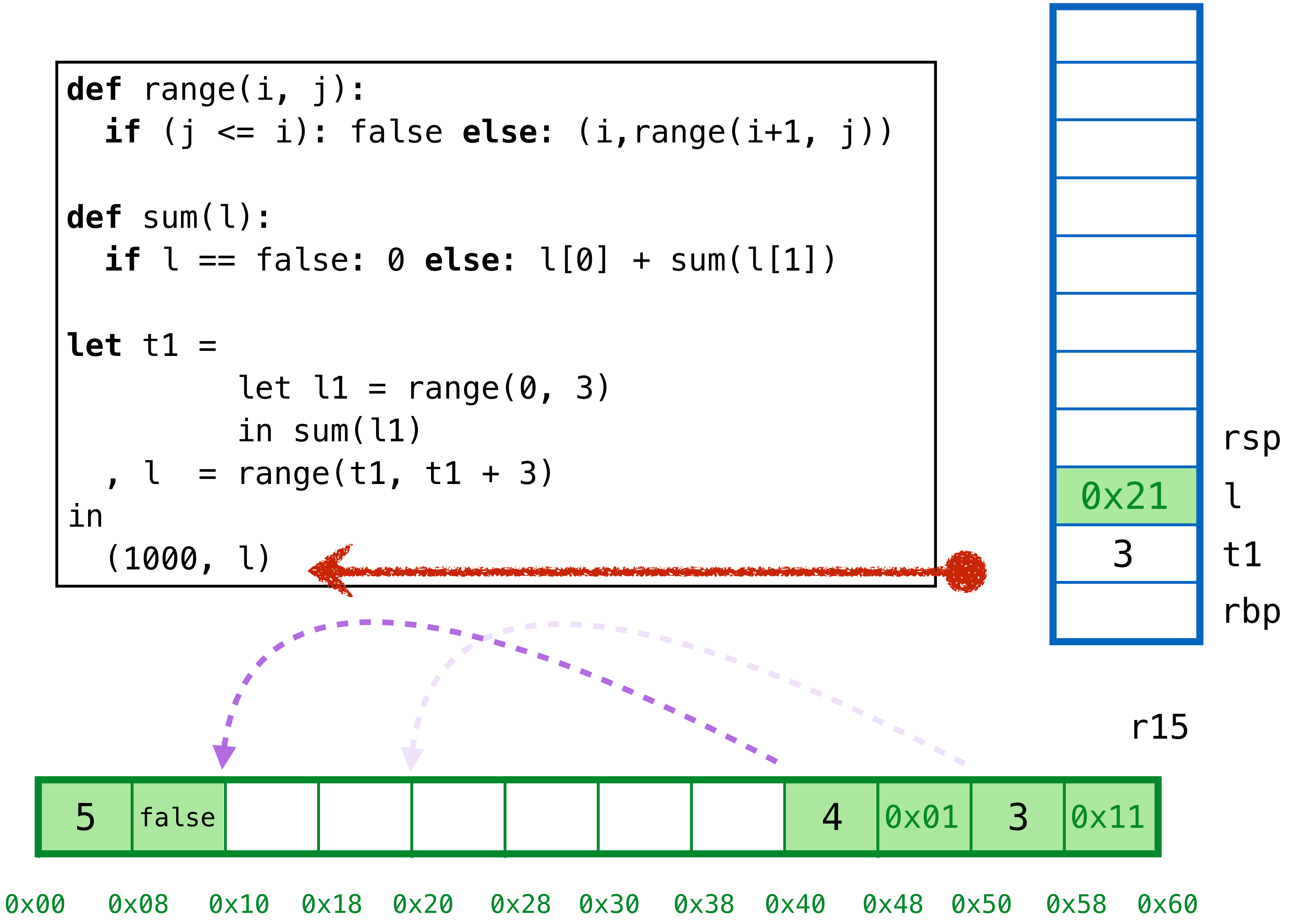
```
def range(i, j):
  if (j <= i): false else: (i,range(i+1, j))

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

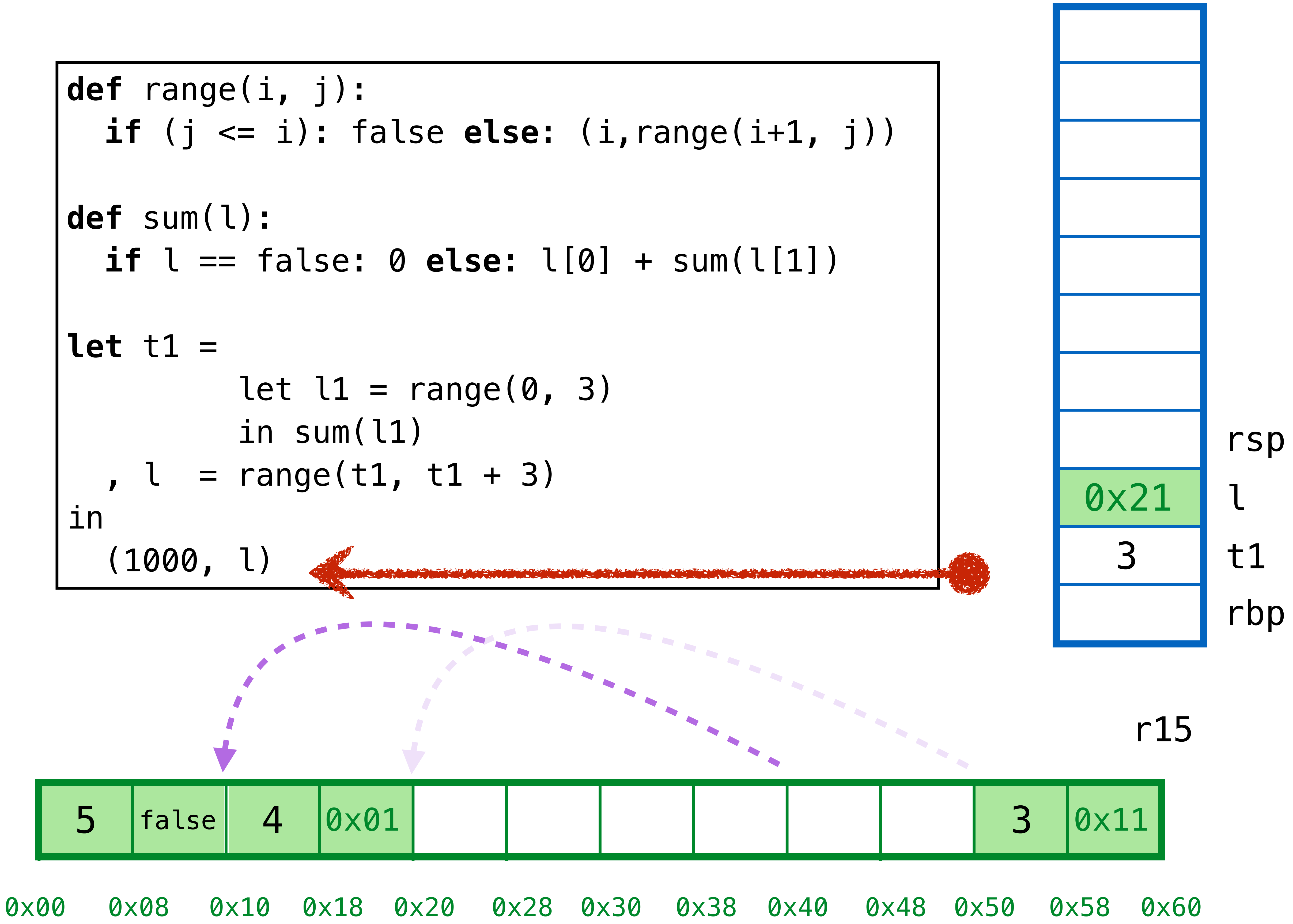


```
def range(i, j):
  if (j <= i): false else: (i,range(i+1, j))

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



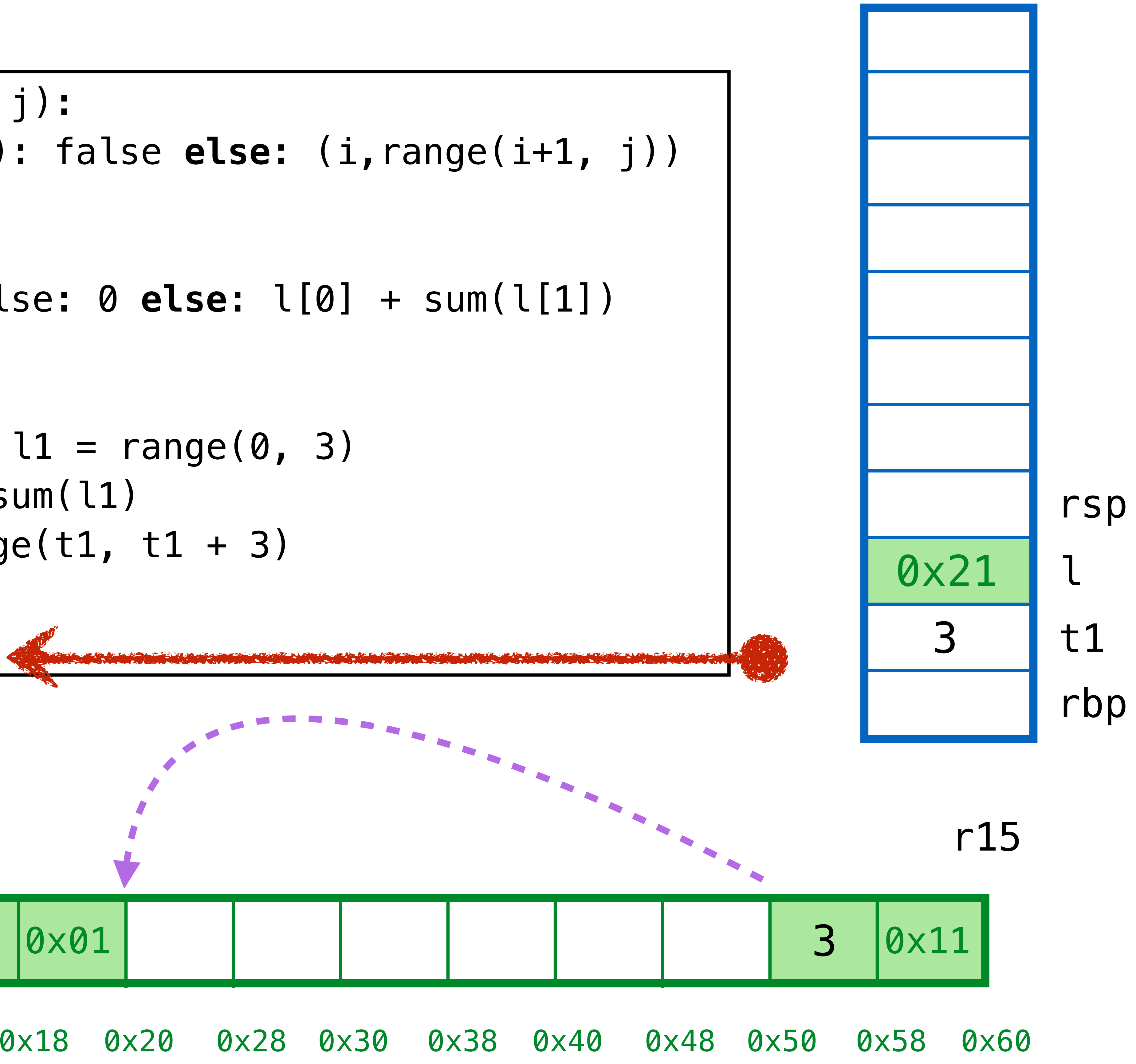
```
def range(i, j):
  if (j <= i): false else: (i,range(i+1, j))

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

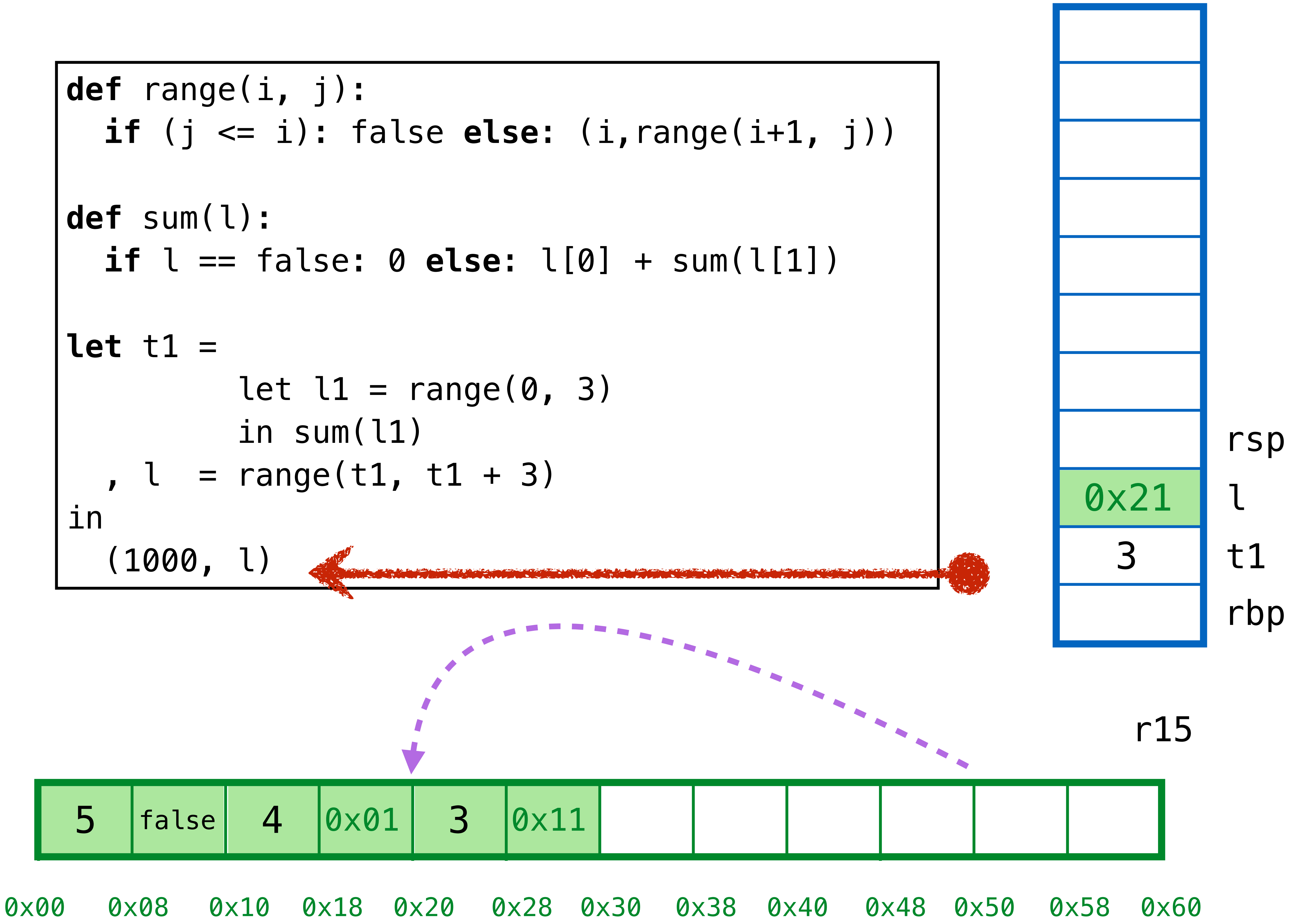
```
def range(i, j):
  if (j <= i): false else: (i,range(i+1, j))

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)
```

	rsp
0x21	l
3	t1
	rbp

r15

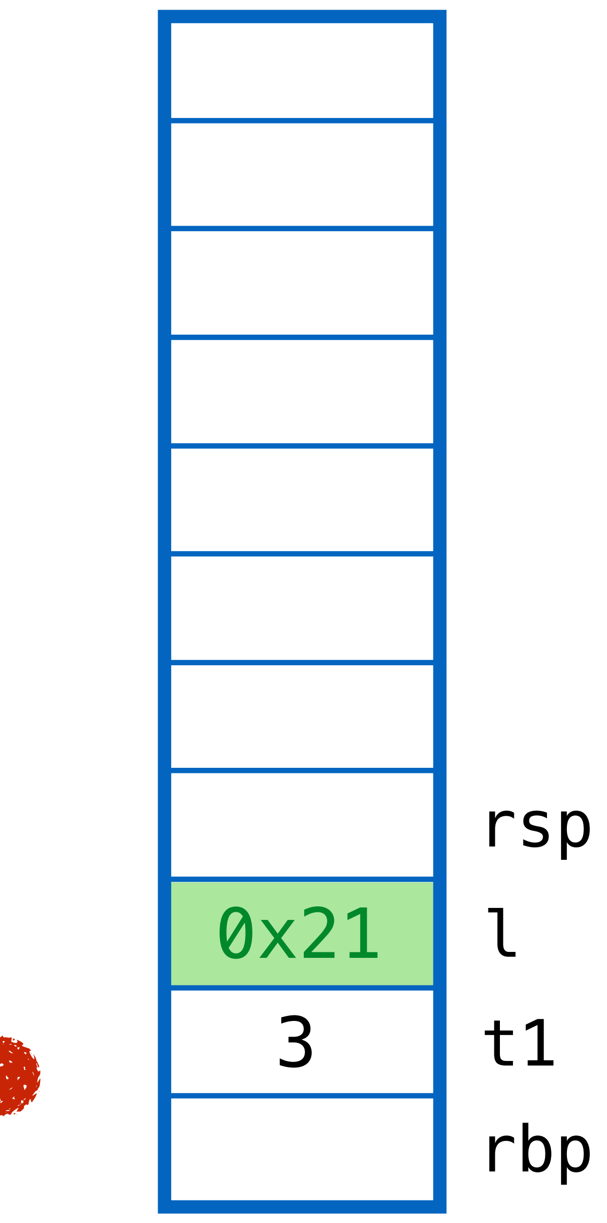
0x00	0x08	0x10	0x18	0x20	0x28	0x30	0x38	0x40	0x48	0x50	0x58
5	false	4	0x01	3	0x11						

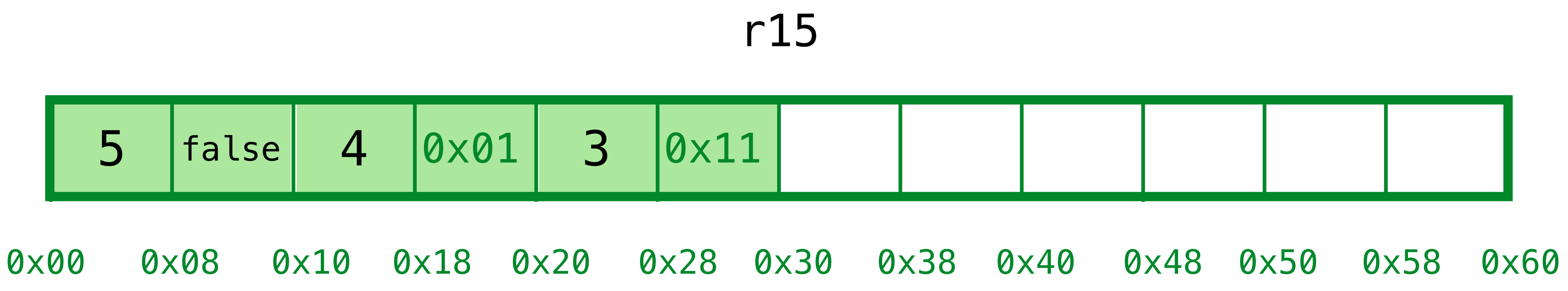
4. COMPACT cells on heap

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)
```



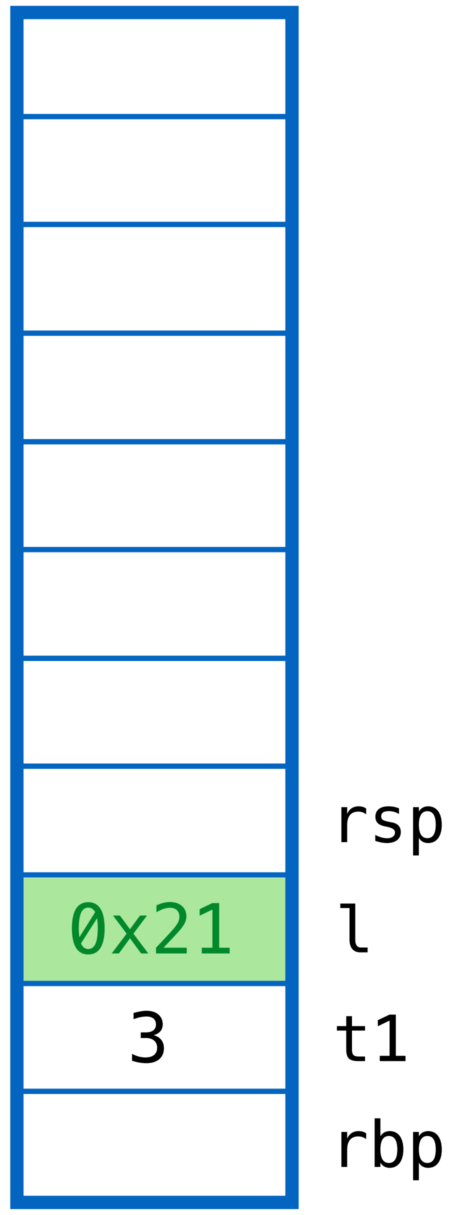


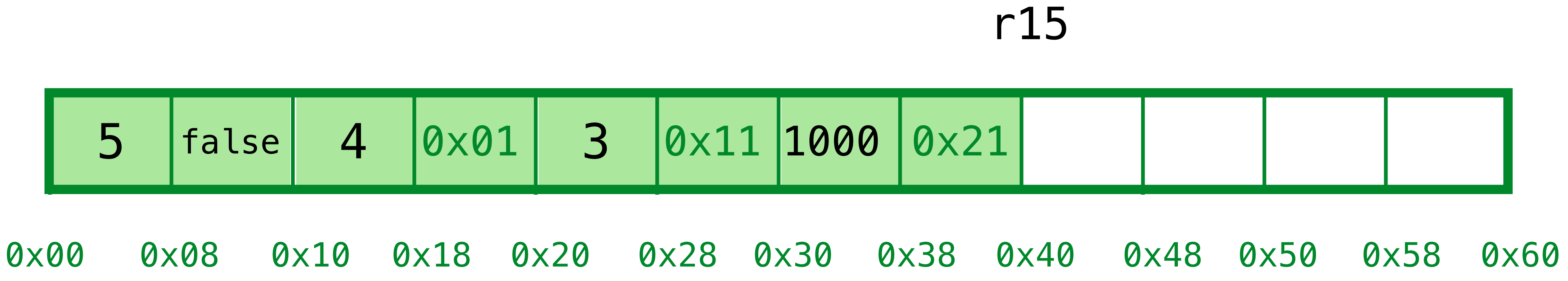
GC Complete!

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)
```





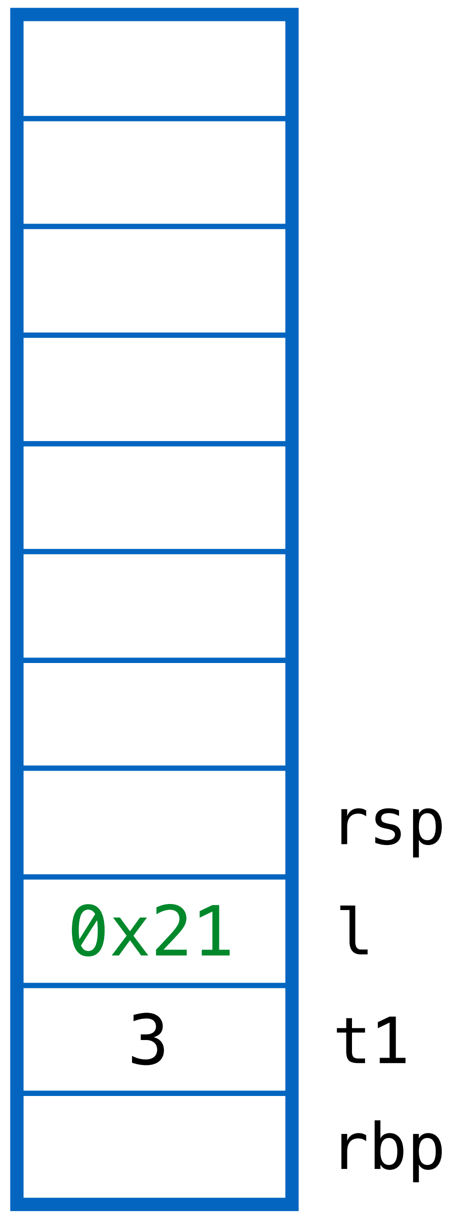
GC Complete!

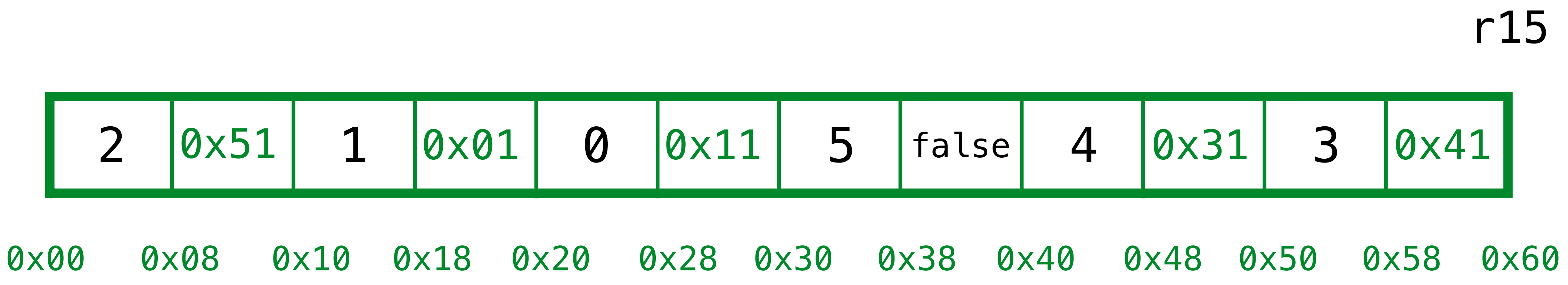
Have space for `(1000, l)`

ex4: recursive data

QUIZ: What should `print(0x21)` 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))))))
```





ex4: recursive data

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

(A) 0x00

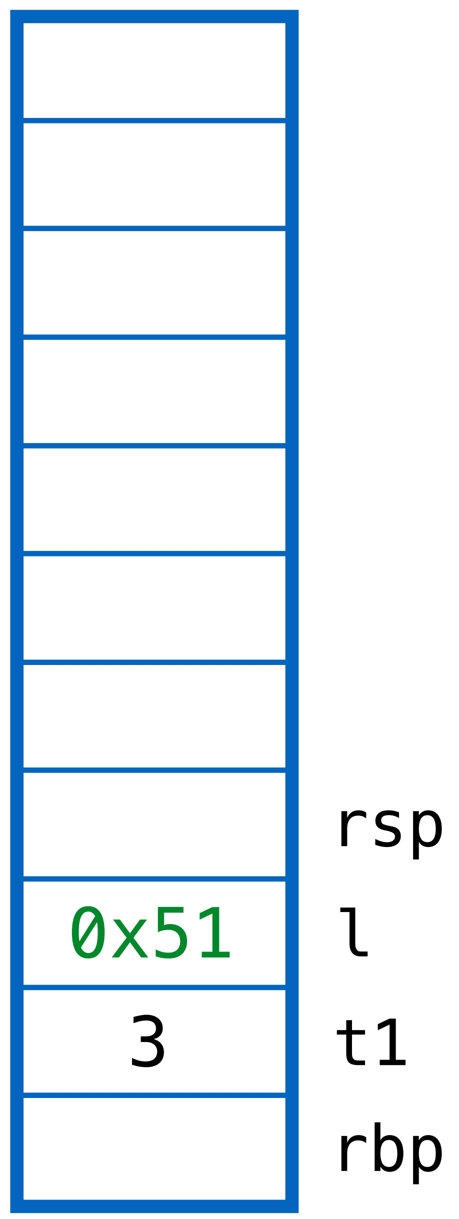
(B) 0x10

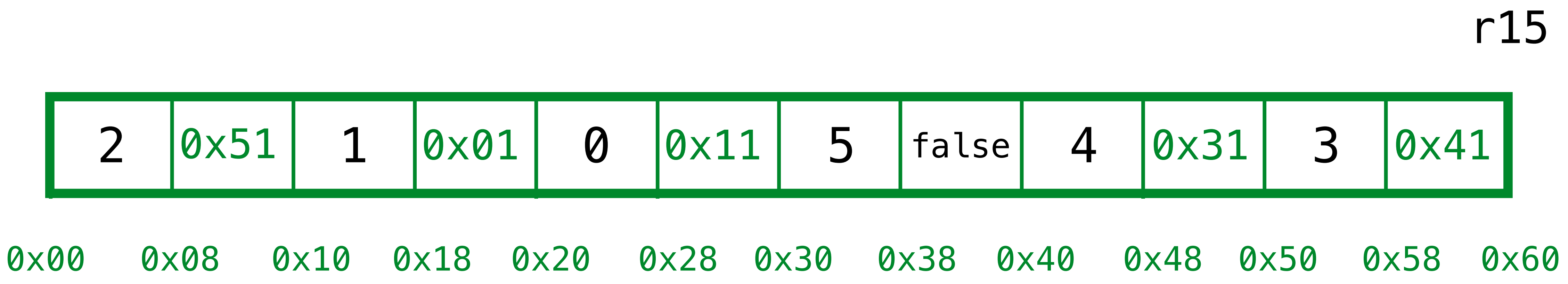
(C) 0x20

(D) 0x30

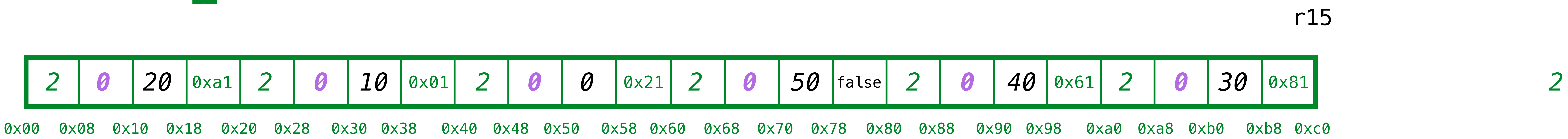
(E) 0x40

(F) 0x50





Heap

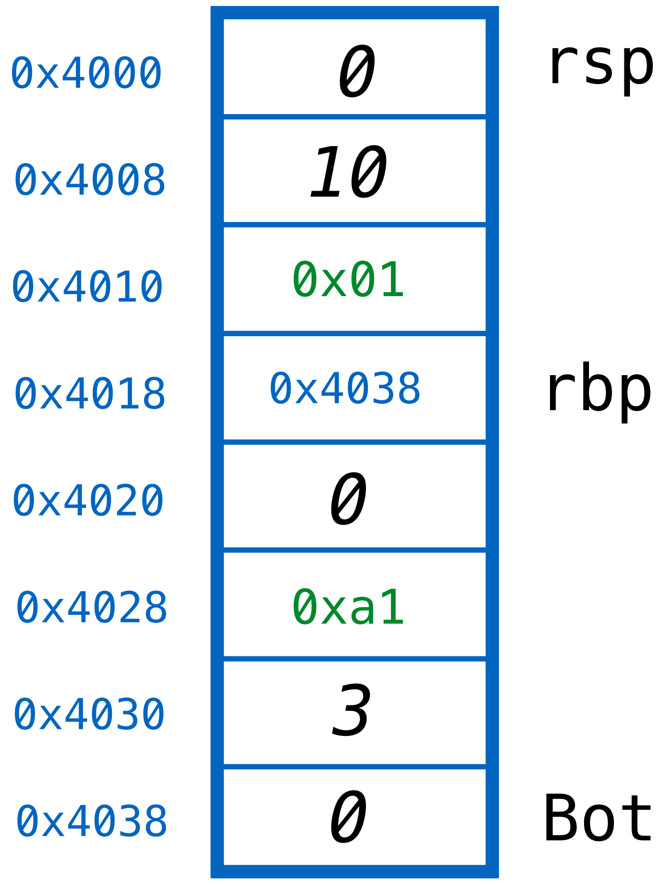


r15

0x00	0x08	0x10	0x18	0x20	0x28	0x30	0x38	0x40	0x48	0x50	0x58	0x60	0x68	0x70	0x78	0x80	0x88	0x90	0x98	0xa0	0xa8	0xb0	0xb8	0xc0
2	0	20	0xa1	2	0	10	0x01	2	0	0	0x21	2	0	50	false	2	0	40	0x61	2	0	30	0x81	

2

Stack



0x4000	0	rsp
0x4008	10	
0x4010	0x01	
0x4018	0x4038	rbp
0x4020	0	
0x4028	0xa1	
0x4030	3	
0x4038	0	Bot