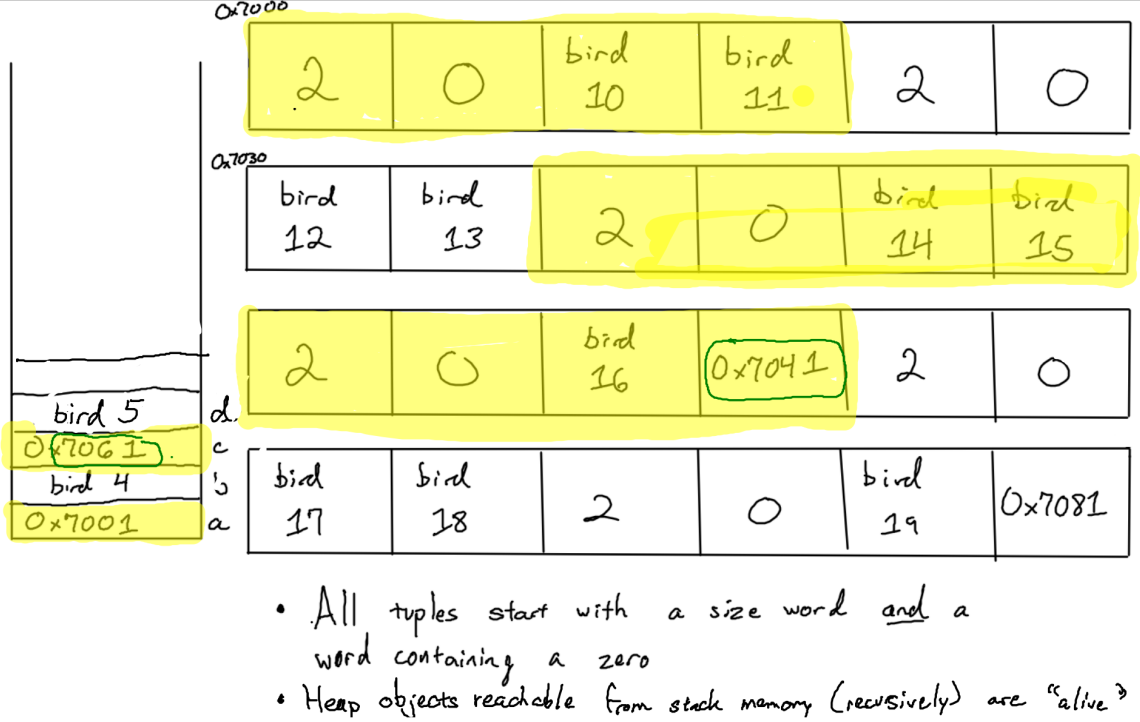
```
let a = (10, 11) in
let b =
    let b1 = (12,13) in
    4
in
let c =
    let c1 = (14, 15) in
    (16, c1)
in
let d =
    let d1 = (17,18) in
    let d2 = (19, d1) in
    5
in
(a, b, c, d)
```



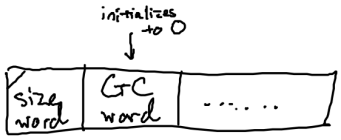
- All tuples start with a size word and a word containing a zero
- Heap objects reachable from stack memory (recursively) are "alive"

Gull (1) Move all of the live things to the left (heap cursor still works)

C (malloc) (2) Track allocation, find a big enough region of memory (leaving everything alone)

Gull garbage collection — Mark/compact (LISP2) "stop-the-world"

Tuples



Closures

GC Algorithm: take in # words I need
compact heap & set heap_cursor
if not enough memory, exit 7

1. Mark live objects: set GC word for all objects reachable from stack to 1
   (Note: all heap objects start w/ GC=0) DFS! !!
2. Forward each live object: walk the heap using an iterator and tracks next free memory in future heap
   for each live object — store future location in GC word
3. Update all pointers to live objects:
   walk stack and heap looking for heap pointers and setting them to GC word of target
4. Compact the heap:
   shift all live heap objects left and set heap_cursor
5. Unmark all live objects
   (set GC word to 0)