Gull

GC:

- ☆ User doesn't allocate/deallocate
- ☆ GC finds unused memory and repurpose
- ☆ User has less direct control

Mark/Compact

1. Mark
2. Forward
3. Update
4. Compact
5. Unmark

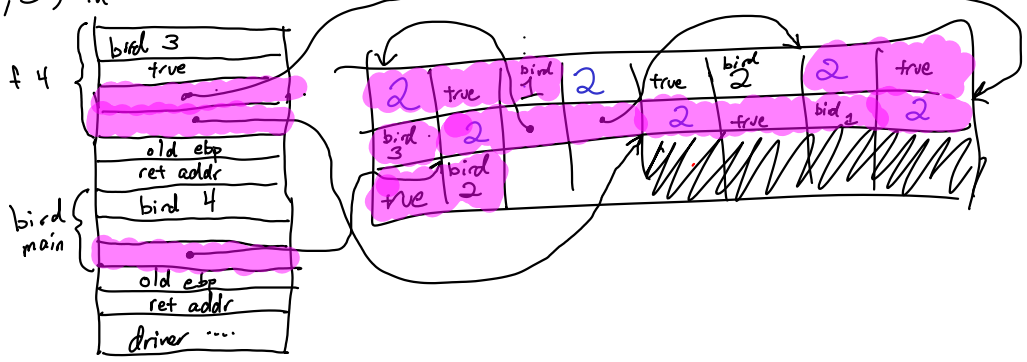
☆ Keep heap_cursor

```
def f x =
  let a = (true, 1) in
  let b = (true, 2) in
  let c = (true, 3) in
  (a, c)
end

(f 0, f 4)
```

Assume heap is 20 words in size.

Falcon heap digram



Falcon

Tuple

Closure

Gull

Invariant: while GC not running, GC word = 0x00000000

0x80000000

2	0x8000 0000	true	bird 1	2	0	true	bird 2
2	0x8000 0010	true	bird 3	2	0x8000 0020	0x8000 0001	0x8000 0021
2	0x8000 0030	true	bird 1	2	0x8000 0040	true	bird 2

1. Mark : "paint" heap objects

   DFS starting from stack

   Set GC word to 0x00000001

2. Forward : decide where each heap object should go, store in GC word

   NOT DFS

   iterate over heap

   At end : each GC word is either 0 (if obj is garbage) or the machine addr to which it should be moved

3. Update ptrs

   Either DFS or iteration

   Update : follow ptr to heap obj, get GC word, convert to bird ptr, then change the ptr var

4. Compact :

   Iterate over heap ; for each obj :

   If GC word ≠ 0, move obj to its new home

5. Unmark

When you allocate heap memory :

first : calc how much

then : if end_of_heap - heap_cursor < need : run GC(need)

Running

(...(..))

* After releasing stack memory, set to zero
* Only operate on bird ptrs between start & end of heap