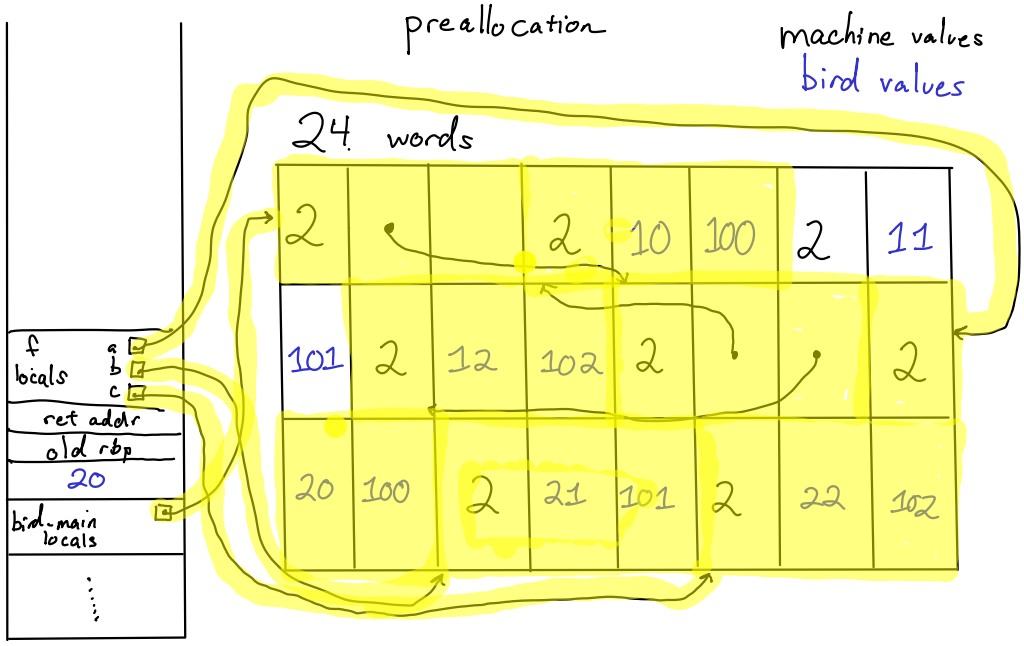
Memory Management

1. Make the programmer allocate/deallocate
2. We do it

Trace memory usage

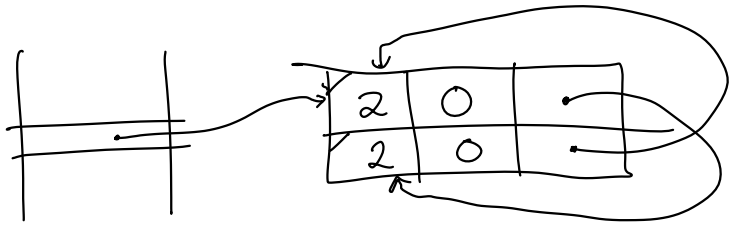
```
def f n =
    let a = (n, 100) in
    let b = (n+1, 101) in
    let c = (n+2, 102) in
    (a, c)
end

(f 10, f 20)
```



"Liveness": a heap object is "live" if a pointer from the stack eventually reaches it

Determining liveness: DFS (keeps a smaller frontier)



In Gull, heap objects have "GC word"

Tuples: 

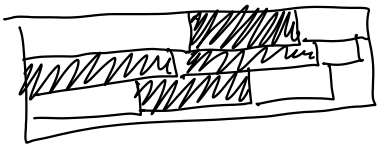
Closures: 

Mark / Compact

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

Compacting: steps 2–4

Invariant: step 5



(invariant: GC=0)

Step 1: Mark

Using DFS, follow every bird pointer and mark (set GC=1) every heap object reachable. Skip anything pointing outside of heap.

Step 2: Forward

Plan where each object will be. Store future location in GC word. (0 = unreachable) Don't move them!

Step 3: Update

Walk over stack & heap. For each bird ptr, get GC word of obj it points to. Not DFS. Replace bird ptr w/ GC word.

Step 4: Compact

Walk over heap and move each live object to where you planned it to be.

Step 5: Unmark

Set GC word to zero for all heap objects.