# Memory Management

* Manual (today)
* Reference counting (problems in cycles)
* Garbage collection (Thursday)

Concerns:
- Invariants of system?
- Where is the responsibility?
- How is allocation managed?

## Albatross (C bird)

How do we manage memory in Albatross?

- creating tuple implicitly allocates memory
- expression: free(<expr>) : deallocates memory
- metadata (what is alloced vs. free) : either O(1) or very small constant on O(n)

```
let x = (1,2) in
let y = (3,4) in
let _ = free(x) in
(5,6)
```

```
let x = (1,2,3) in
let y = (4,5) in
let _ = free(x) in
(6,7,8,9,10)
```

1. Every "pointer" is actually a key in table where value are real pointers

| | |
|---|---|
| 5 | 0x8000000c |
| 3 | 0x80000000 |
| 4 | 0x80000004 |

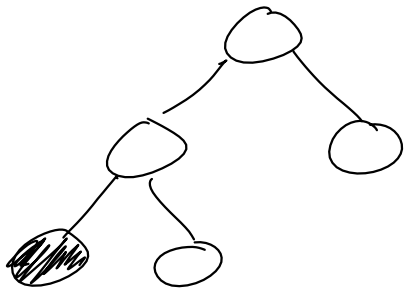
Keep heap_cursor.
When free memory: move ~~everything~~ to the left
How do I know how much memory I'm freeing?
* Invariant: I can look at anything on heap and tell how big it is
* Alt: table could remember
Free is O(n)

2. Divide heap into blocks of 32 bytes
For each block:
How much of block is used
Start of block
Linked-list-style "next" to allocate many blocks

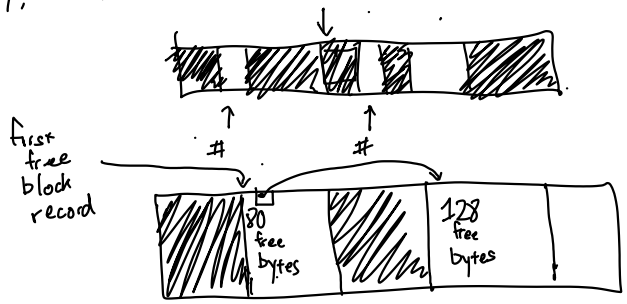
3. Bitmap: assume I have metadata = 1 bit per word in heap

Alloc/free: O(n)?

Suitable for disks.



4. Free block list




Alloc: walk the list looking for a big enough space; ask OS for more if need be