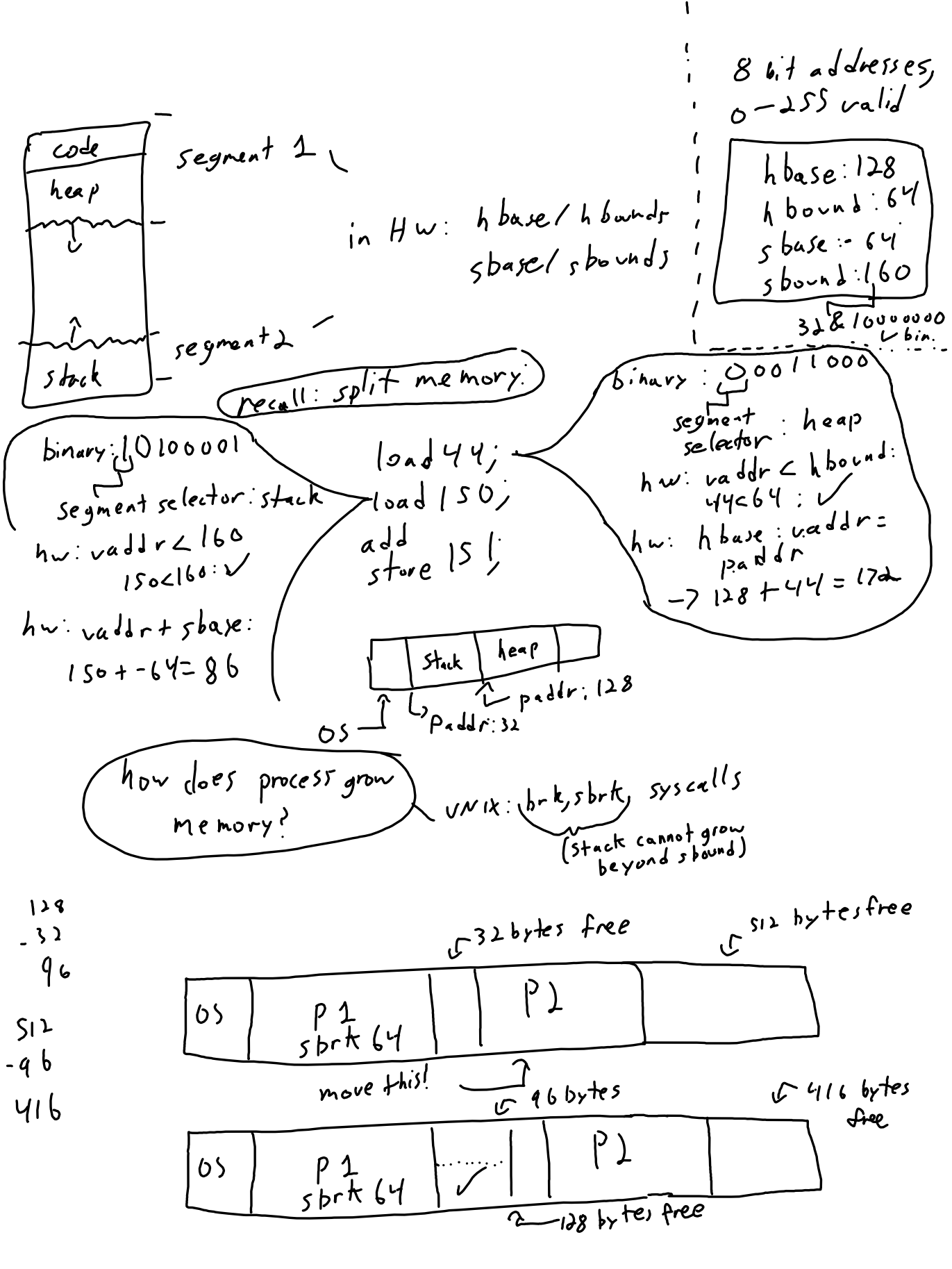
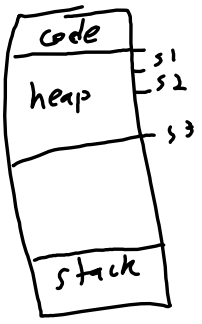

8 bit addresses, 0-255 valid
code
heap
stack
segment 1
segment 2
in Hw: hbase/hbounds
sbase/sbounds
hbase: 128
hbound: 64
sbase: 64
sbound: 160
32 & 10000000 bin.
recall: split memory
binary: 00011000
segment selector: heap
hw: vaddr < hbound:
44 < 64 : ✓
hw: hbase: vaddr = paddr
-> 128 + 44 = 172
load 44;
load 150;
add
store 151;
binary: 10100001
segment selector: stack
hw: vaddr < 160
150 < 160: ✓
hw: vaddr + sbase:
150 + -64 = 86
Stack
heap
paddr: 128
Paddr: 32
OS
how does process grow memory?
UNIX: brk, sbrk, syscalls
(stack cannot grow beyond sbound)
128
-32
96
512
-96
416
32 bytes free
512 bytes free
OS
P1
sbrk 64
P2
move this!
96 bytes
416 bytes free
OS
P1
sbrk 64
P2
128 bytes free

really expensive! (can be worse: swapping out)

idea: many segments — non-contigous



still need to swap sometimes - be judicious!

interlude: malloc/free in C

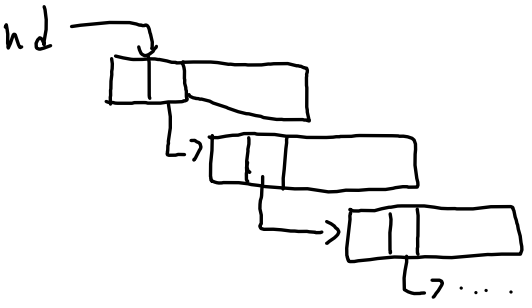
goal: don't call brk/sbrk until you have to

idea: re-use free memory; don't call OS

store memory in a free list

```
struct flistNode {
  uint size;
  struct flistNode* next;
  char space[];
}
```

← empty: dynamic size




```
malloc(n){
  flistNode* curr = hd;
  while(curr->size > n) {
    curr = curr->next;
    if(curr = nullptr) {....}
  }
  return (char*) curr) + sizeof(uint) + sizeof(void*);
}
```

```
free(void*p) { flistNode* metadata = ((flistNode*)p) - 1
               metadata->next = hd;
               hd = metadata; }
```

issues?

step 1: split chunks

step 2: best-fit, first-fit algorithms

problem: tiny chunks!!

solution: ??? (can't change virtual addresses)

idea: lots of tiny segments??

↳ fragmentation in OS...

lots of lookup hw...

just accept fragmentation (for now)

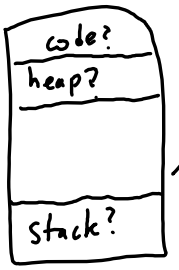
problem: application wants big chunks!

Aside: the tyranny of C:

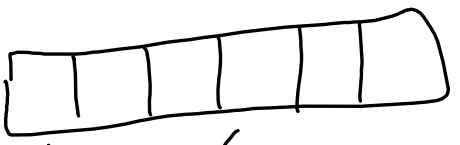
threads? activation records?

why are we special-casing heap and stack...

rethink: no bias for applications.



X don't overfit!!!



lots of base + bound?
(same issues of fragmentation)

need something better than the free list...

- back to C -

slabs / buddy allocators

no more variable-sized!

size 2^8: 

size 2^a: 

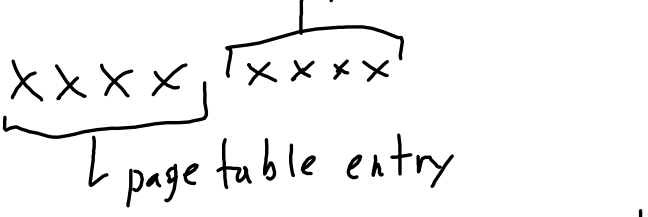
use best-fit list, O(k) where k = # slots

- back to OS -

pages: 4k segments.

how to store / access? lists are dynamic... non-starter

page table



just use a big array! (aside: 1-bit page table = traditional segments)