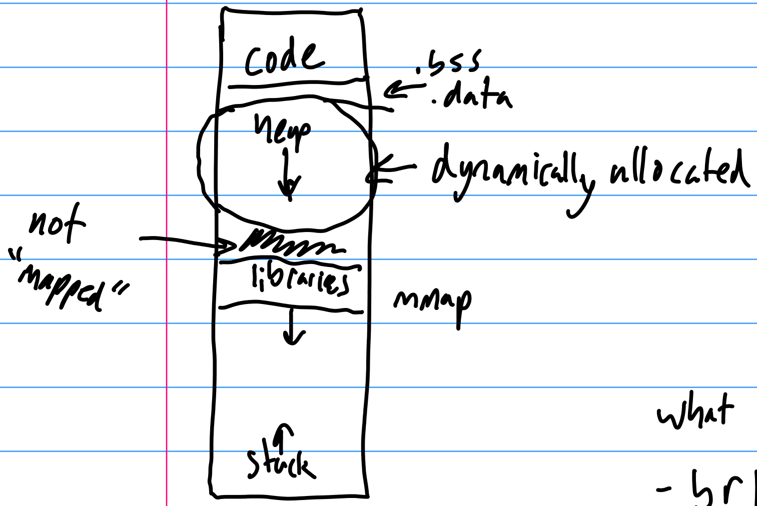
CS 3214 lecture # 18 : Malloc

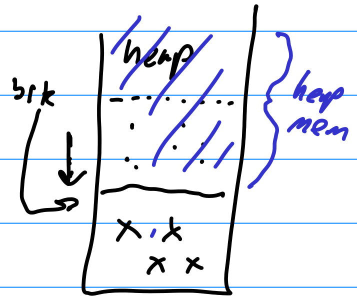
Memory Management

Process
Linking/Loading
Multithreading



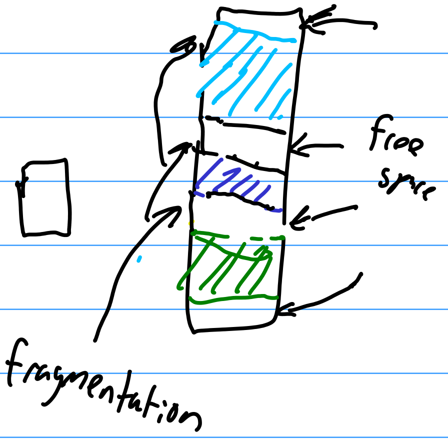
virtual memory

[malloc]



what does the kernel expose
- brk "system break"
sbrk

how to manage heap?



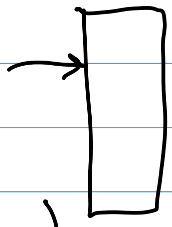
void * malloc (size_t size);
free(void * p);
void * realloc (void * p, size_t size);

why is malloc's job hard?
- we don't know how long memory will be used
order of alloc /free
size, distribution of sizes
- we can't use malloc inside malloc
data structures must be in our managed memory
- we can't move data we have given
touch data
- alignment constraints

Goals:

performance : alloc/free

(think not linear in size of mem)



→ space utilization

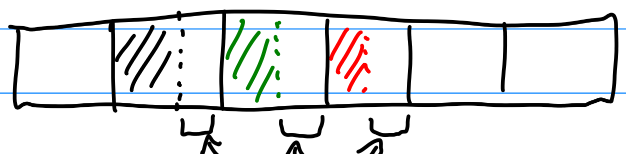
- fragmentation
- allocated data payload/mem

p ← malloc(size)

"payload"

Fragmentation

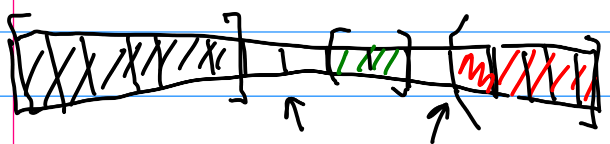
internal vs. external



split up mem into blocks

unused memory inside a block

internal fragmentation



unused memory outside allocation

"external" fragmentation

can measure based on what has been allocated

harder to measure

know what "too small" is

void* ← malloc (size)

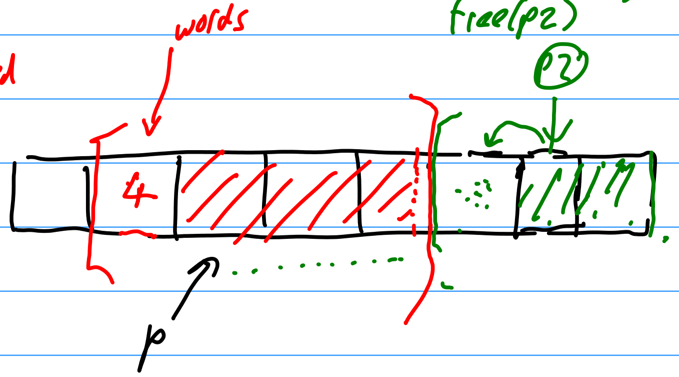
free(p)

8 byte aligned

words

p2 ← malloc(2)

free(p2)

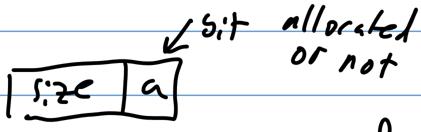


manual
automatic

p ← malloc(3)

How do we keep track of free blocks?

struct meta

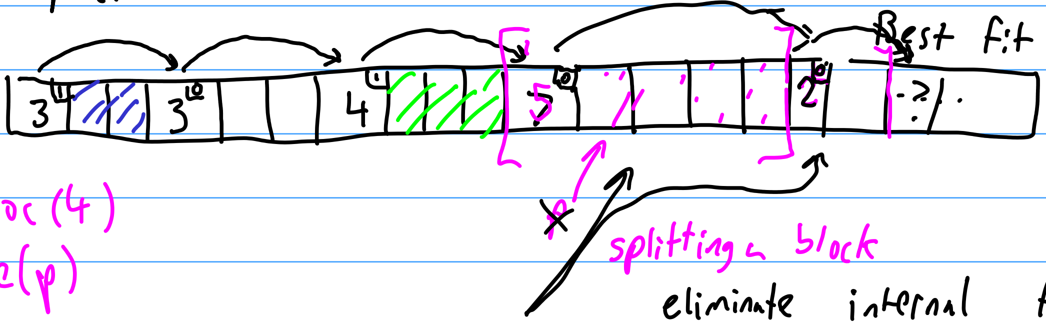


```
struct foo{
  unsigned a:1;
  unsigned s:31;
}
```

perf?
fragmentation?

First fit
Next fit
Best fit

1. implicit list

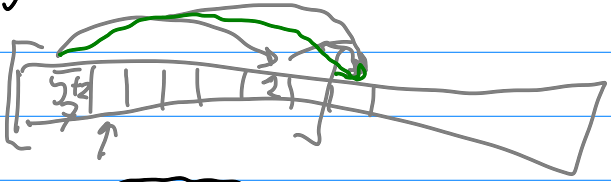


p = malloc(4)
free(p)

splitting a block

eliminate internal fragmentation

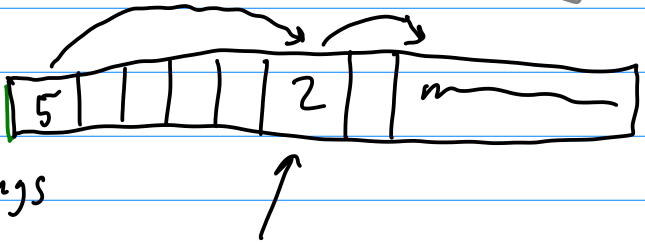
false fragmentation



Coalesce blocks

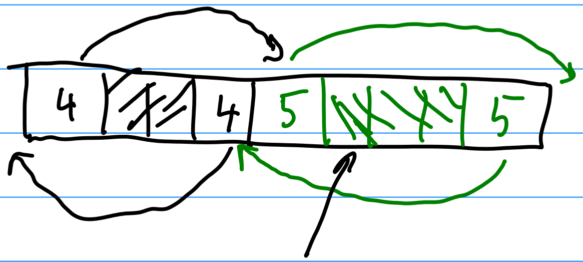
next: easy

prev?



Knuth: boundary tags

$\frac{\text{payload}}{\text{mem}}$ ratio ↓



Policies

- how we find free block (first, next, best)
  tput vs. frag
- splitting
- coalescing

Implicit list: impl simple
alloc: linear in size of mem
free: constant
mem usage: depends on fragmentation

Explicit free list