Computer Systems

CS107

Cynthia Lee

Today's Topics

- Managing the heap
- Important for your final assignment, assign7!

Heap Manager

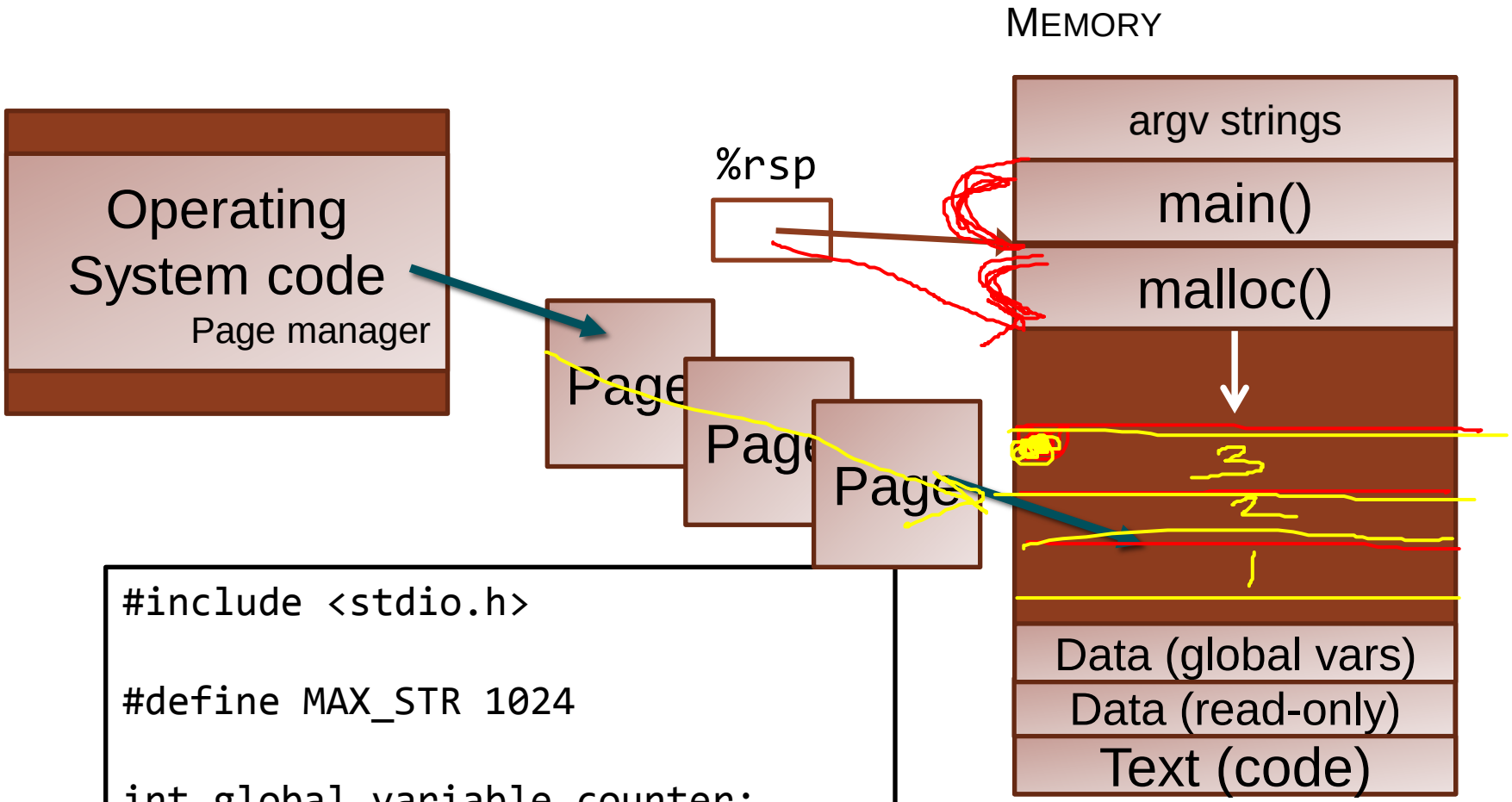
Remember what you know from being a *client* of malloc/free:

The heap manager functions like a hotel front desk—handing out room assignments based on size needs (malloc), and checking people out when they're ready to leave (free).

Heap Allocator: you (almost) don't need a spec for this assignment!

- You will implement three functions, whose arguments, return values, and behavior, you already know:
  - `void *malloc(size_t num_bytes_requested)`
  - `void *realloc(void *old_ptr, size_t new_bytes_requested)`
  - `void  free(void *ptr)`
- Other requirements/limits:
  - Maximum request size for a single request is INT_MAX bytes
  - Must be fast (throughput) and make efficient use of space (utilization)
  - No more than 500 bytes of global variables, other storage comes out of the heap memory you would otherwise give to user.

Big picture: what do you have to work with?



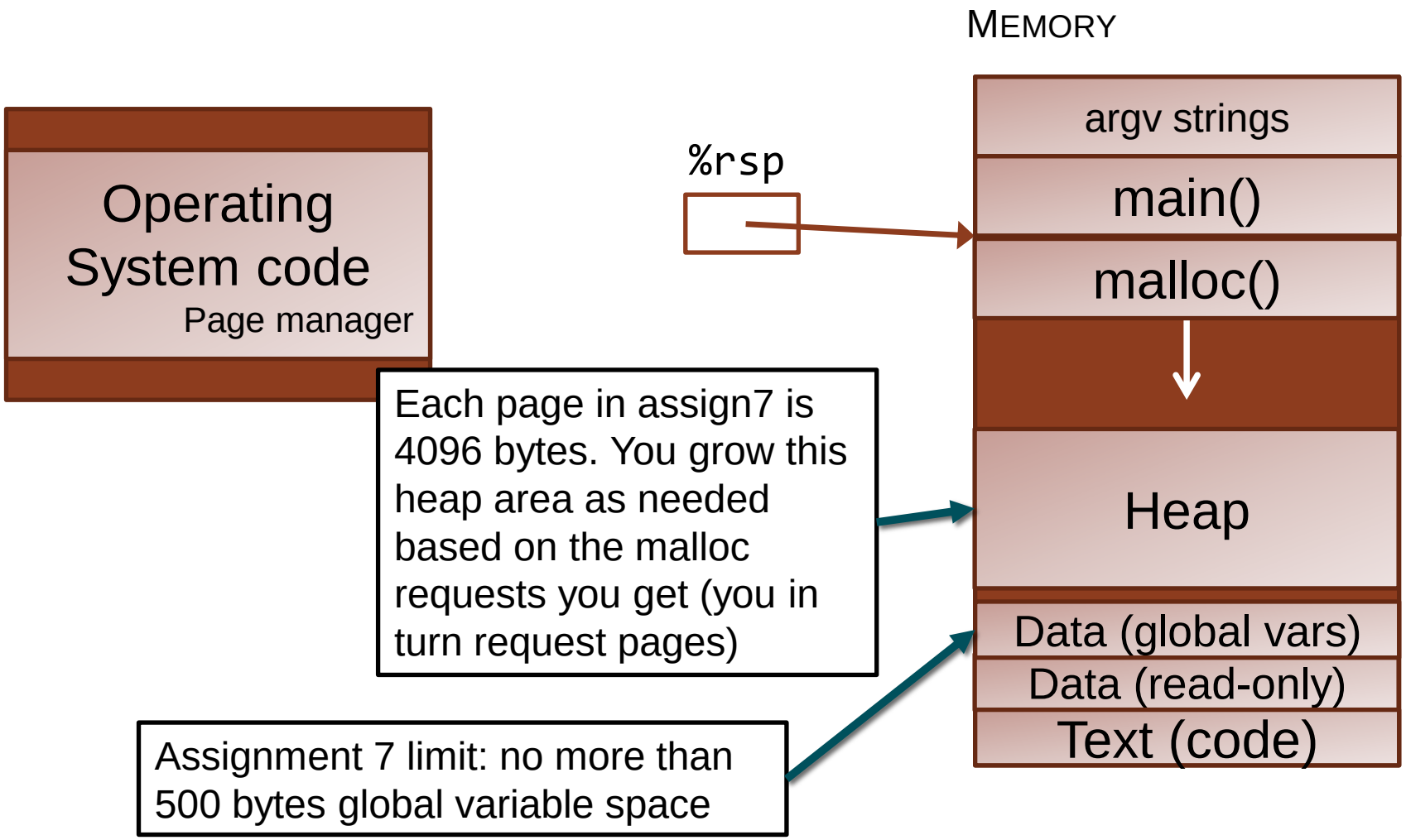

```
#include <stdio.h>

#define MAX_STR 1024

int global_variable_counter;

int main(int argc, char *argv[])
{
```

Big picture: what do you have to work with?



Heap Manager

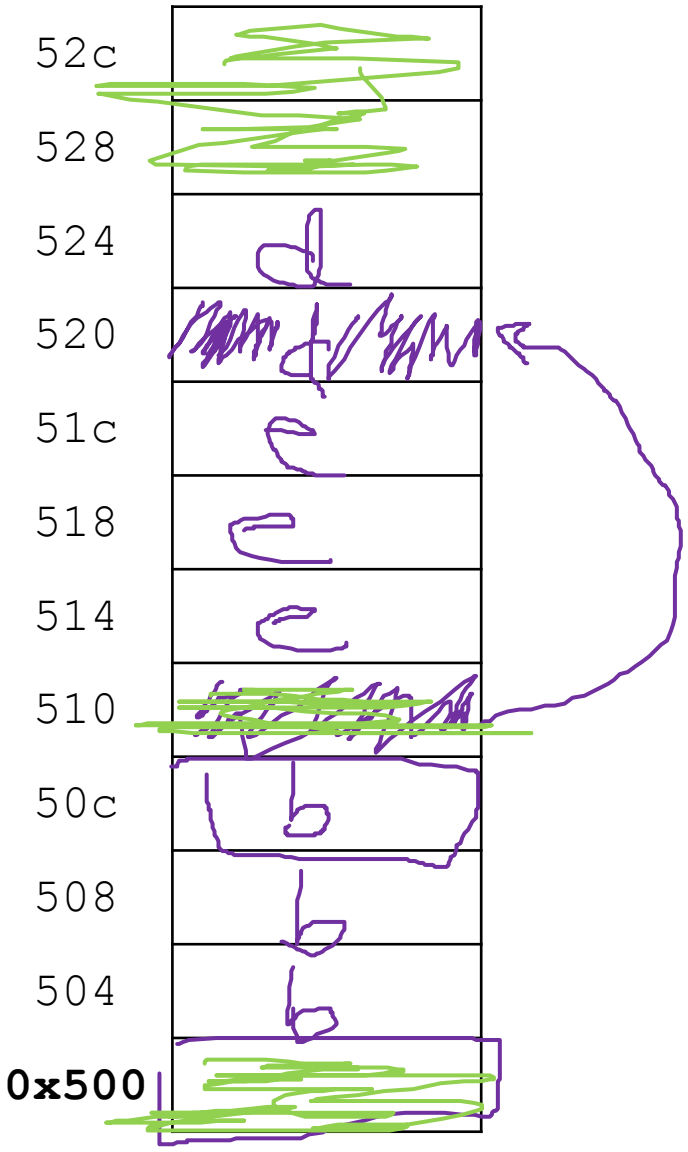
Some simple implementation framework examples

Heap Manager: Example client code

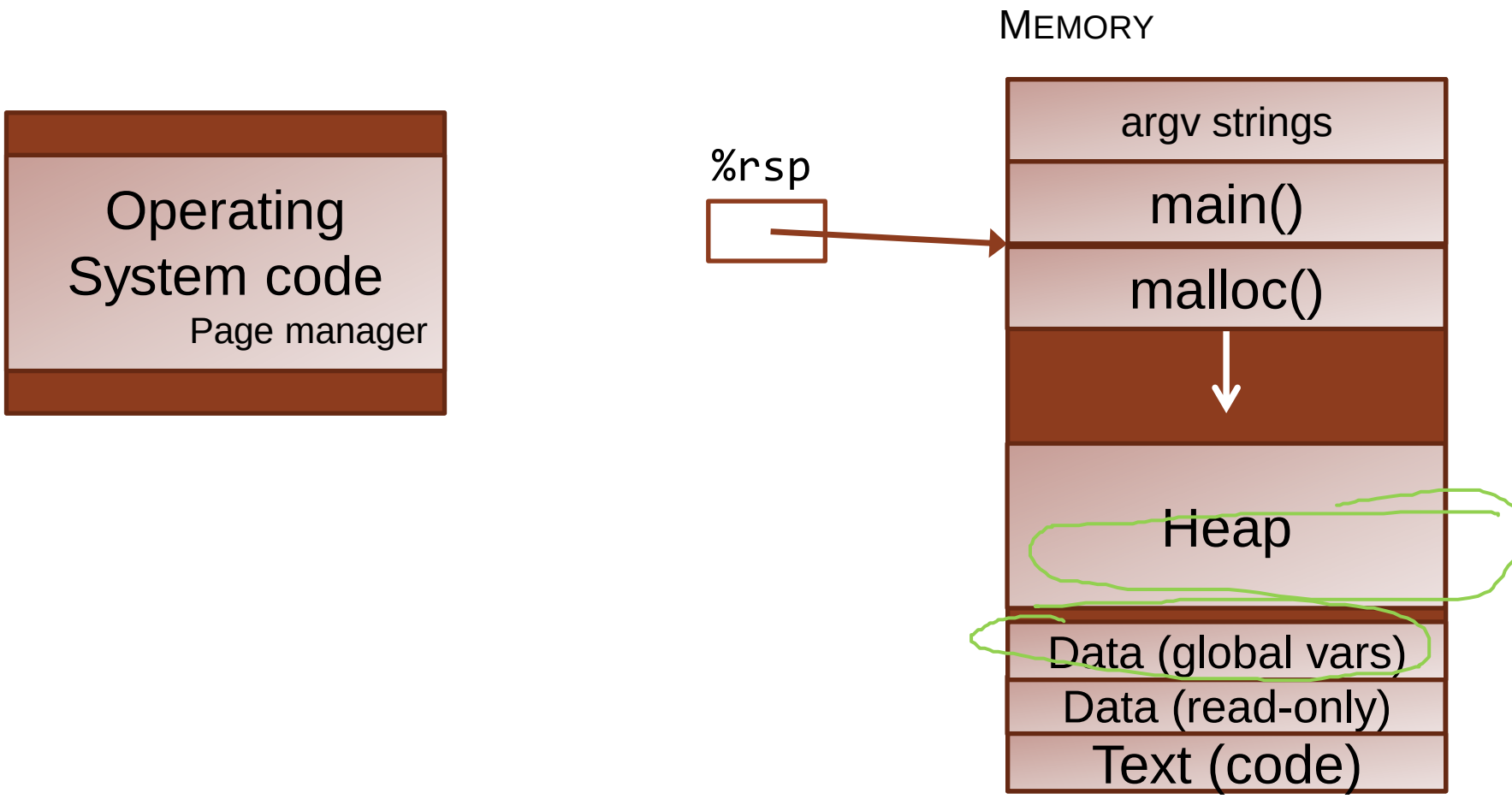
```
void *a, *b, *c, *d, *e, *f;
a = malloc(4);
b = malloc(8);
c = malloc(4);
d = malloc(4);
free(a);
free(c);
e = malloc(12);
b = realloc(b, 12);
d = realloc(d, 8);
f = malloc(12);
```

Implementation #1: global variable space lists

```
void *a, *b, *c, *d, *e, *f;
a = malloc(4);
b = malloc(8);
c = malloc(4);
d = malloc(4);
free(a);
free(c);
e = malloc(12);
b = realloc(b, 12);
d = realloc(d, 8);
f = malloc(12);
```



Big picture: what do you have to work with?



Discussion question

Could the heap manager pause and do a "defrag" operation on the heap at this point?

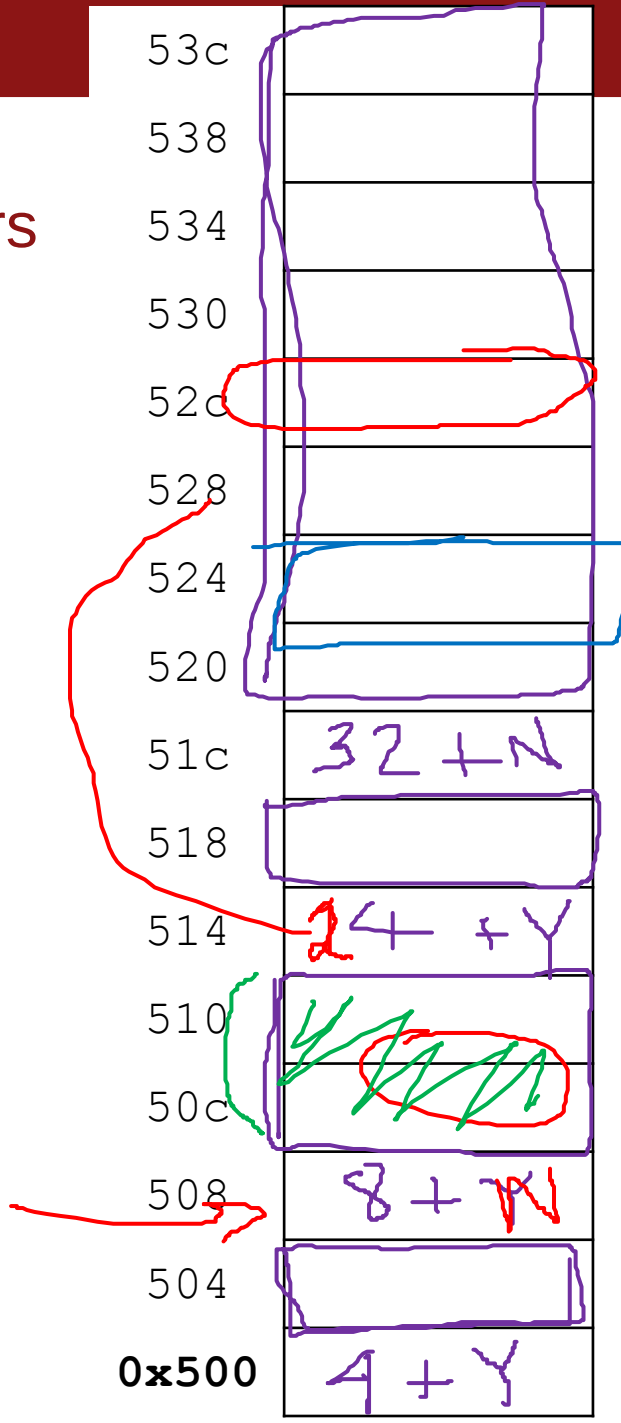
- Move data in allocated areas over to coalesce free space into a contiguous block
- Don't change size of any allocated block, and carefully copy data

A. YES, great idea!
B. YES it can be done, but not a good idea for some reason (e.g., not efficient use of time)
C. NO, it can't be done!

Why or why not?

Implementation #2: Basic pre-node headers

```
void *a, *b, *c, *d, *e, *f;
a = malloc(4);
b = malloc(8);
c = malloc(4);
d = malloc(4);
free(a);
free(c);
e = malloc(12);
b = realloc(b, 12);
d = realloc(d, 8);
f = malloc(12);
```



Discussion question

How do we handle realloc(b, 12)?

Discussion question

Is there anything about these approaches to the heap manager (generally, or specifically this pre-node header implementation) that strikes you as being a bit dangerous?

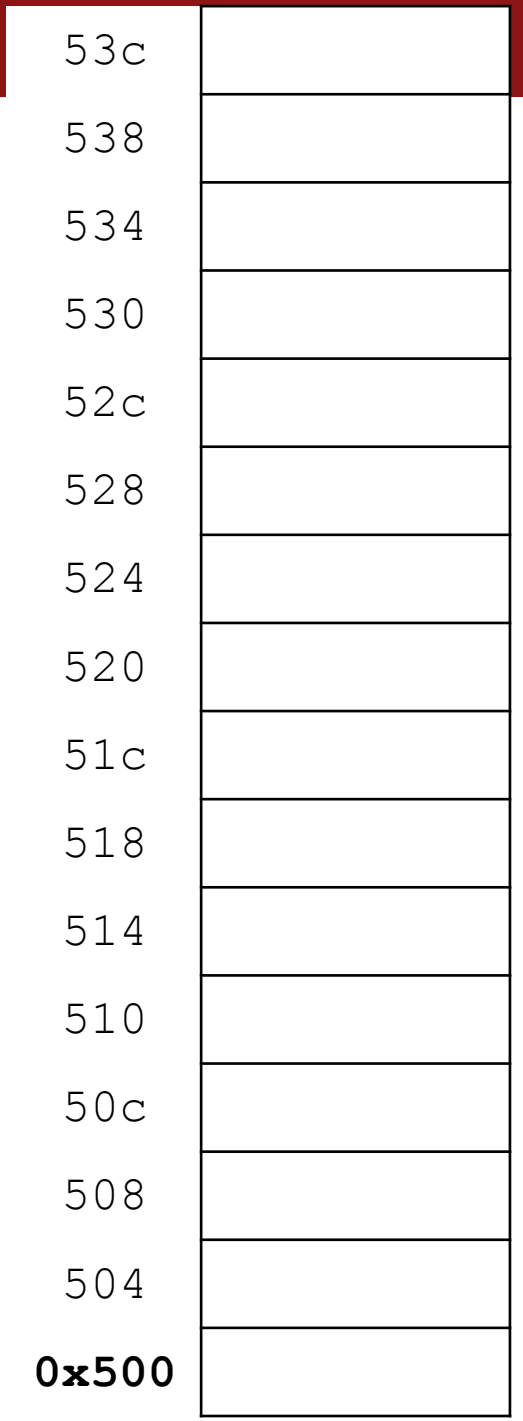
- i.e., from a security and software engineering perspective

What kind of accidental (or malicious?) bugs in the user's code could impact the heap and heap manager? How?

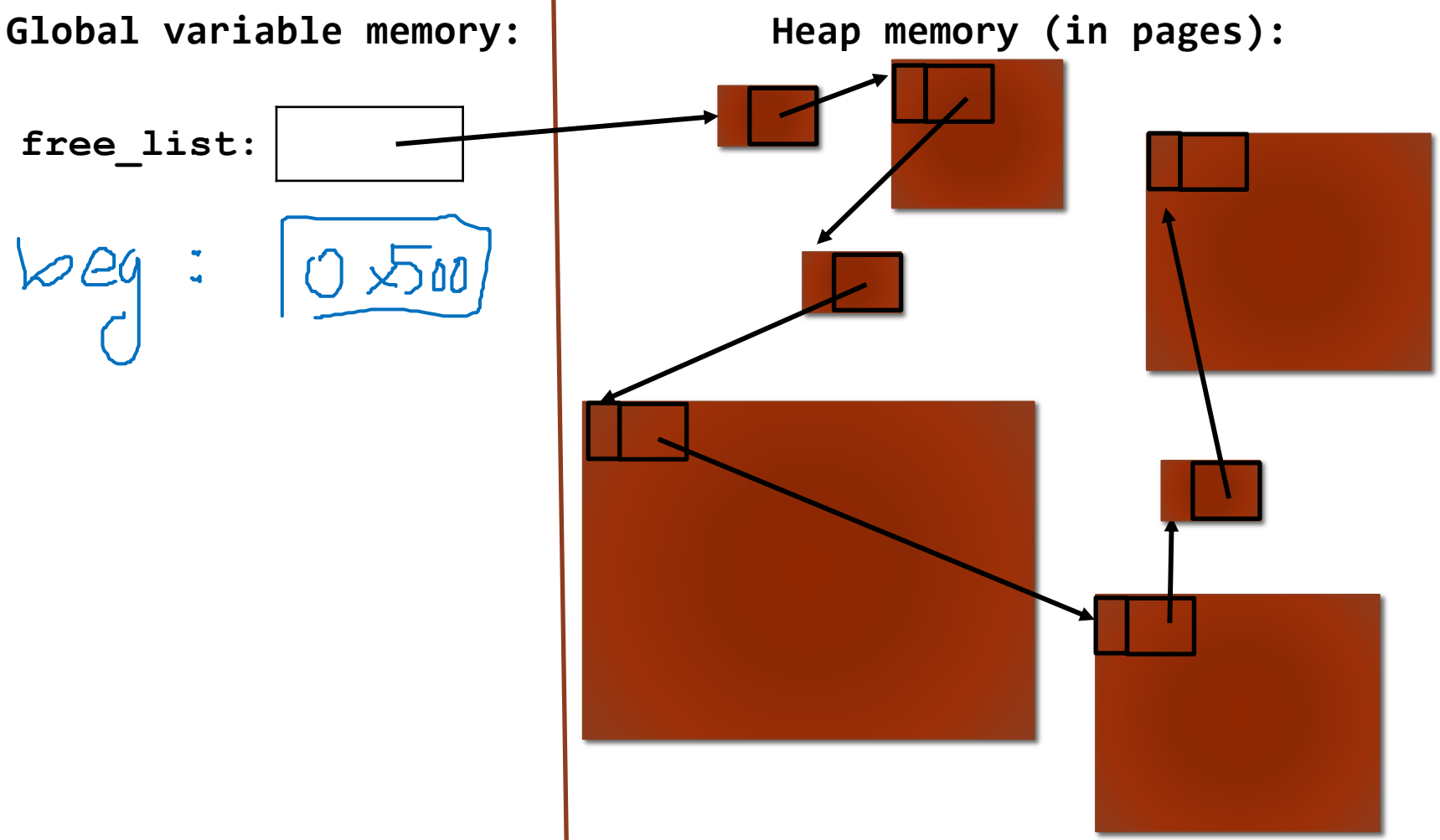
What defenses does the heap manager have available?

Implementation #3: Pre-node headers + free nodes organized into a linked list ("free list")

```
void *a, *b, *c, *d, *e, *f;
a = malloc(4);
b = malloc(8);
c = malloc(4);
d = malloc(4);
free(a);
free(c);
e = malloc(12);
b = realloc(b, 12);
d = realloc(d, 8);
f = malloc(12);
```



Free List



Heap Manager

Optimizations

Back to the "defrag" idea:

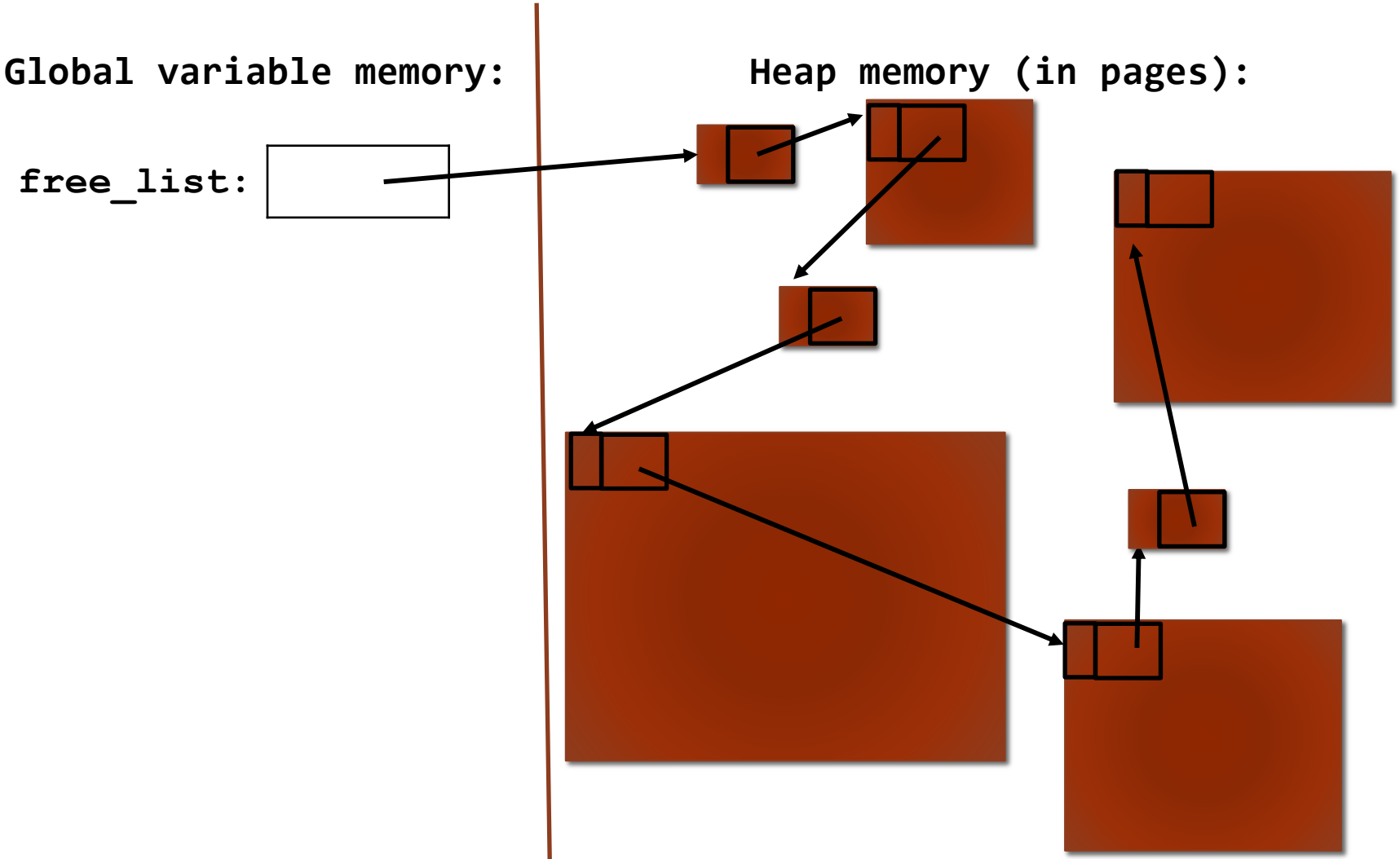
- We know that we can't rearrange pointers at will for currently-allocated memory
- However, there is nothing to stop us from taking adjacent already-freed blocks and coalescing them into a larger freed block

YES, great idea!

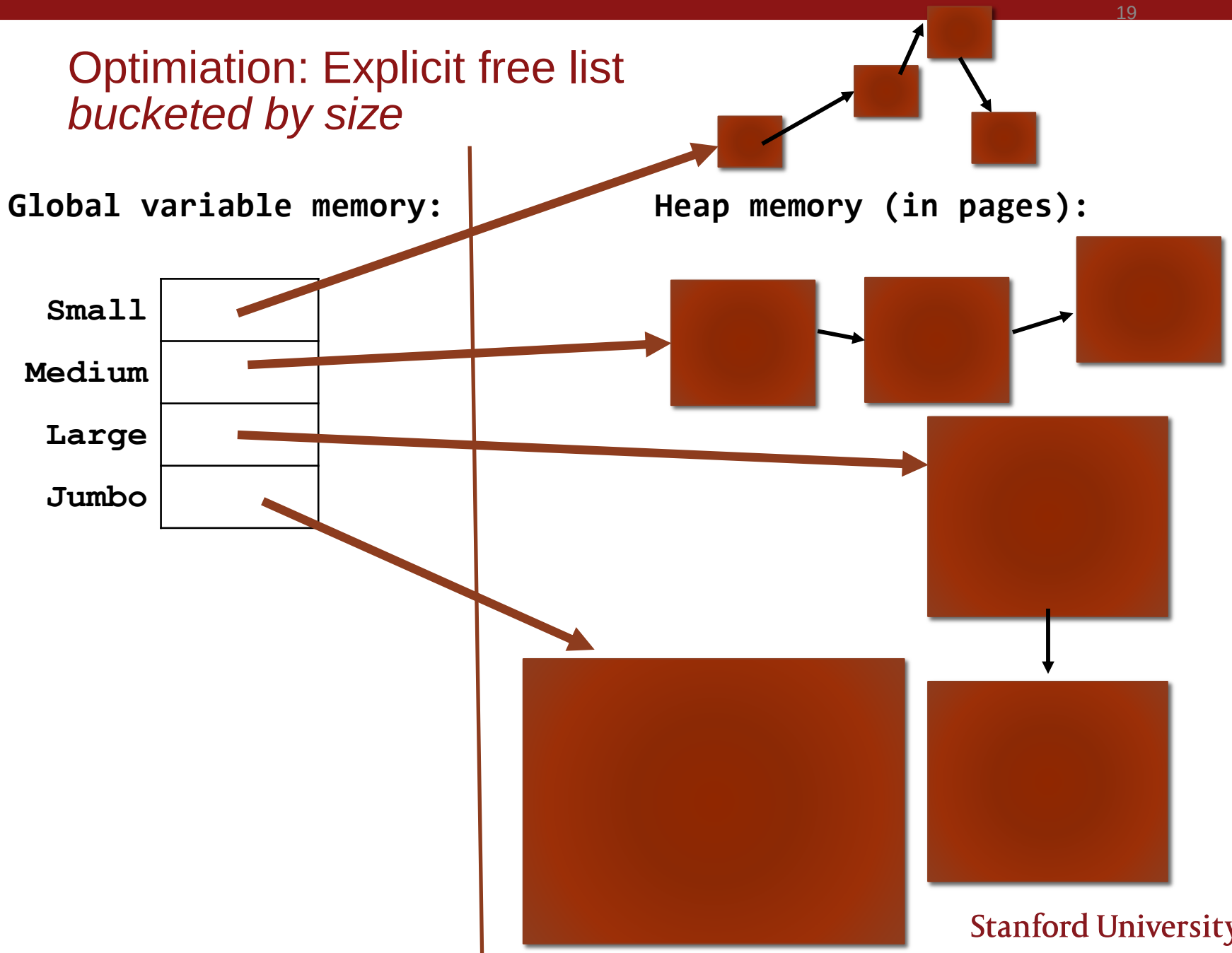
The question is, given our free list is a linked list, how do we "notify" incoming pointers to free blocks that the block has now been aggregated into one conglomerate node? Hm….

- See textbook's "footer" solution

Free List



Optimiation: Explicit free list *bucketed by size*



How can we balance the conflicting goals of the heap manager?

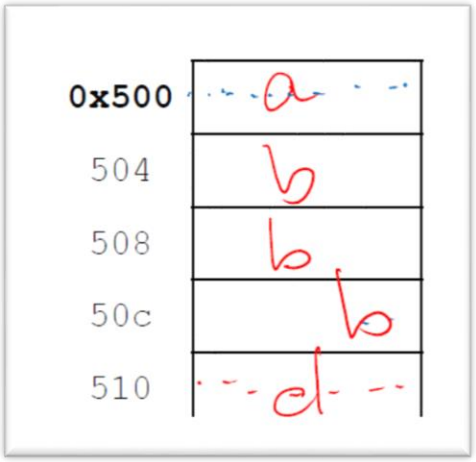
Correctness

(obviously)

Throughput
(speed of handling requests)

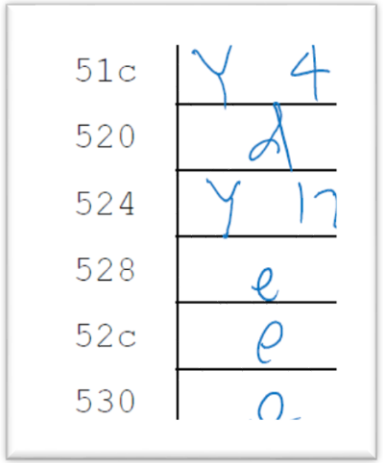
Utilization
(not too many holes in space)

Your Pinterest Board of Design Ideas for Heap Allocator



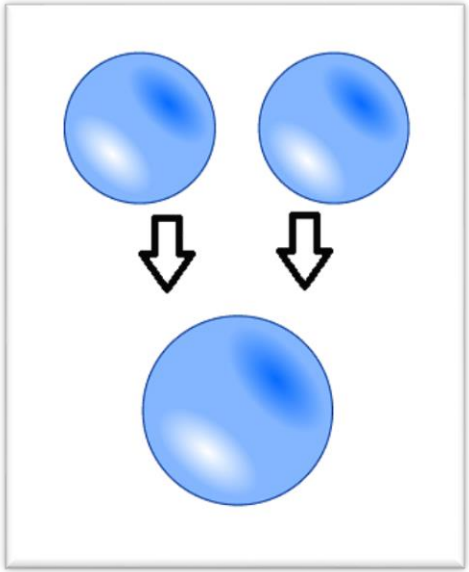
DIY Easy!

In-Use List + Free List



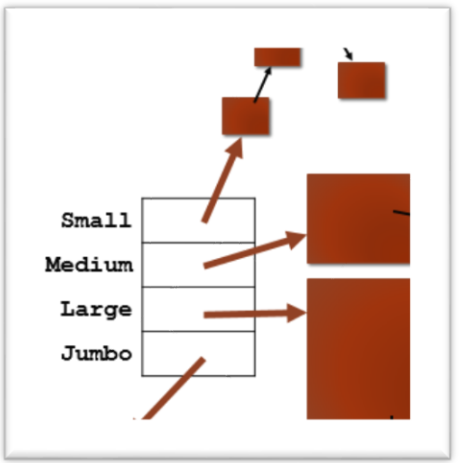
Cute Storage Idea!

Header holds block metadata adjacent to payload



Declutter Tip

Coalesce adjacent free blocks to reduce fragmentation



Organize!

Multiple free lists, bucketed by block size