Free Space Management

Glenn Bruns
CSUMB

Lecture Objectives

After this lecture, you should be able to:

- Understand policies for free space management
- Be able to simulate these policies

Allocating memory

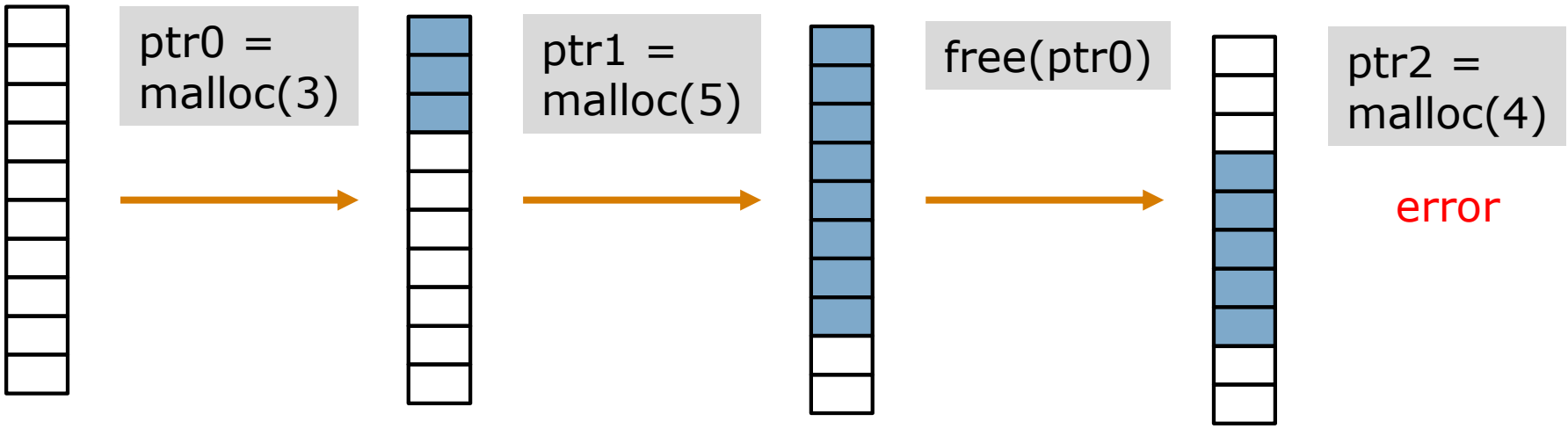
We've seen memory allocation a few times

- user programs use malloc for dynamically-created data structures
- the OS needs to allocate memory for processes

Memory allocator API:

- malloc(n) – get a pointer to (at least) n bytes of memory
- free(ptr) – return memory to allocator

Example:



What are the design goals?

- Correctness
- Speed
- Little memory waste
- As many requests as possible are satisfied

Avoiding fragmentation is a secondary goal

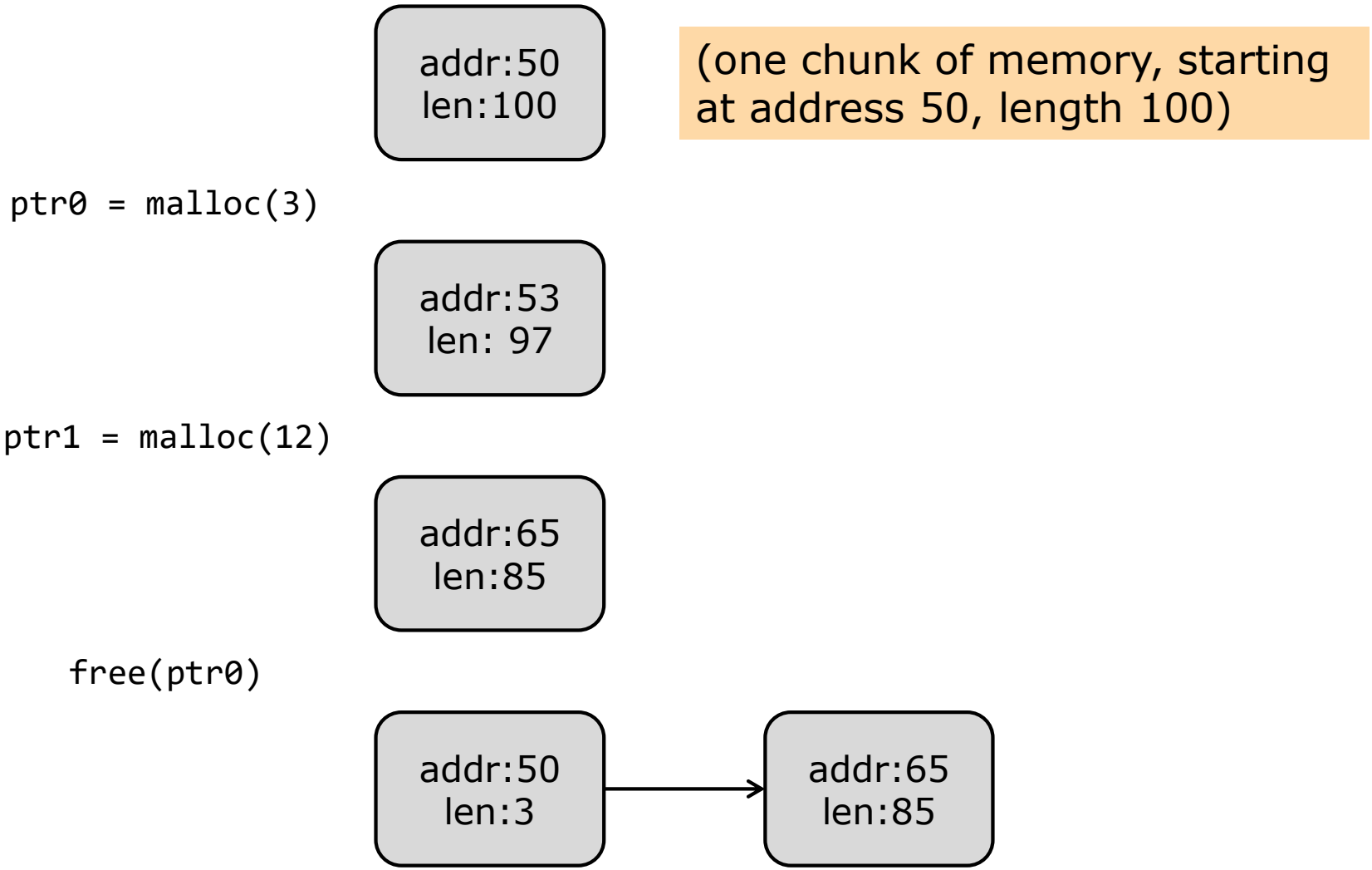
Mechanism: tracking free memory

Question: how would keep track of free memory?

Ideas:

- keep a list of the bytes that are busy
- keep a list of sections of memory that are busy
- keep a list of sections of memory that are free
- keep a history of all malloc and free calls that were performed successfully

Tracking free memory as a list



Free list in OSTEP simulator style

```
[ Size 1 ]: [ addr:50 sz: 100 ]
```

```
ptr0 = malloc(3)
```

```
[ Size 1 ]: [ addr:53 sz: 97 ]
```

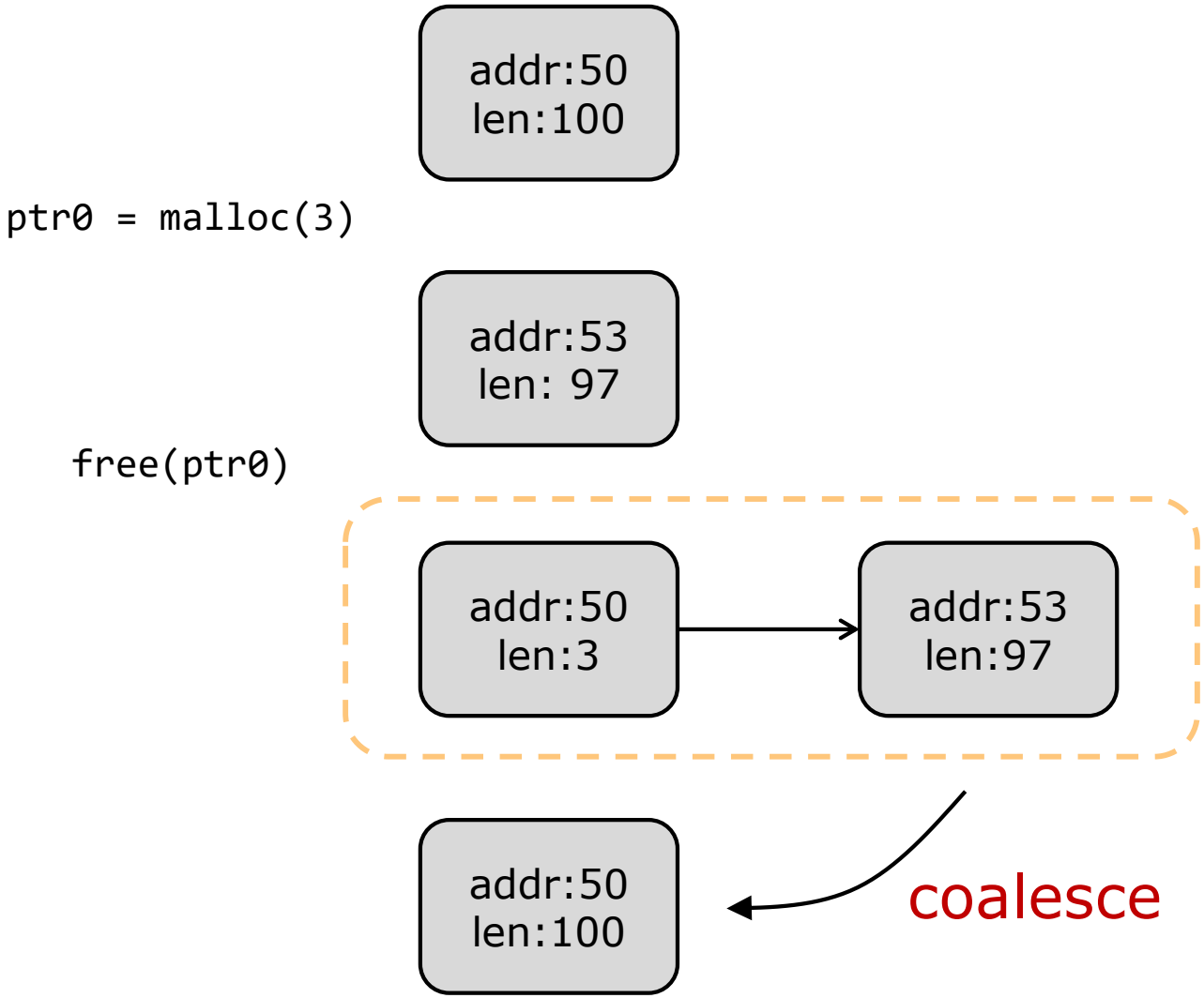
```
ptr1 = malloc(12)
```

```
[ Size 1 ]: [ addr:65 sz: 85 ]
```

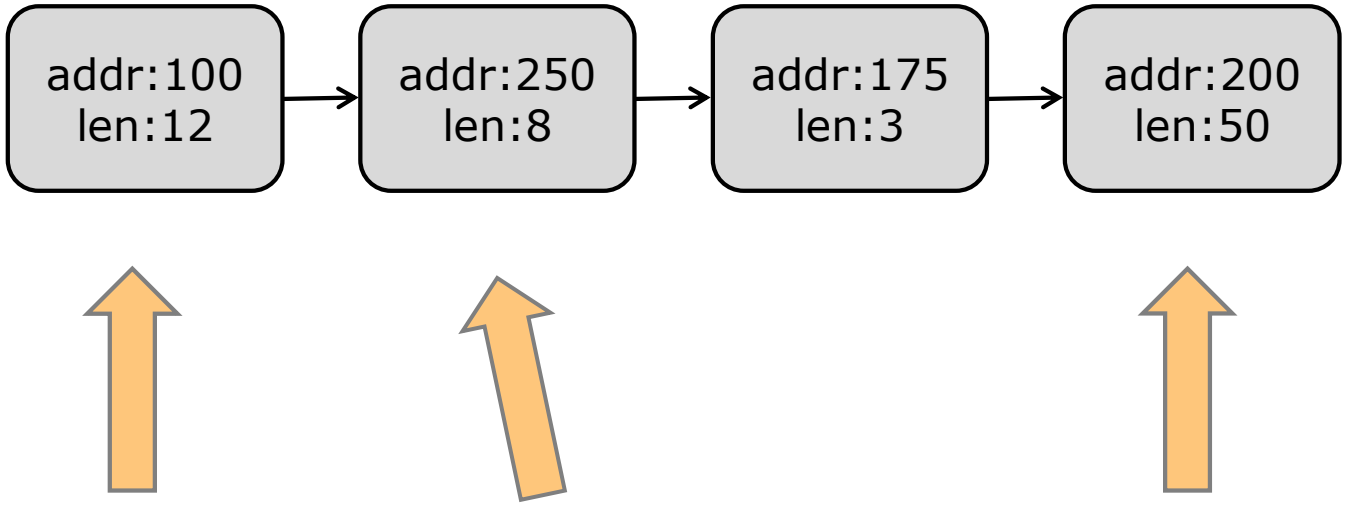
```
free(ptr0)
```

```
[ Size 2 ]: [ addr:50 sz: 3 ] [ addr:65 sz: 85 ]
```

Coalescing free list elements



Policy: which memory to allocate?



first fit: use first item that's large enough

best fit: use smallest item that's large enough

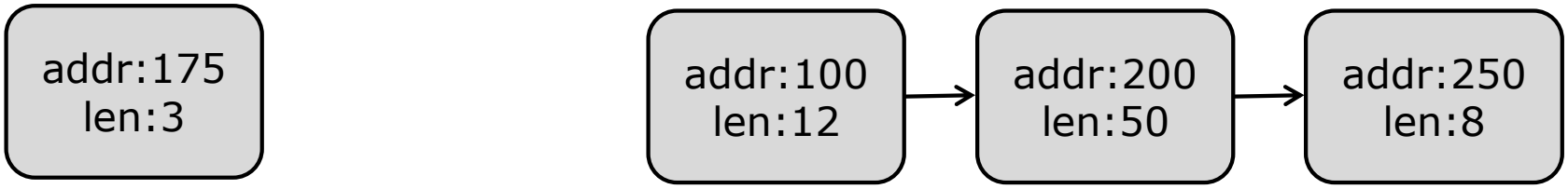
worst fit: use largest item (that's large enough)

fast

try to keep large chunks

avoid fragmentation

Policy: where to put freed items?



`free(175)`

front of list: fast

keep list ordered by addr: good for coalescing

keep list ordered by length: good when allocating

Summary of main policy options

which item to allocate?

- first fit, best fit, …?

where to put items in list when freed?

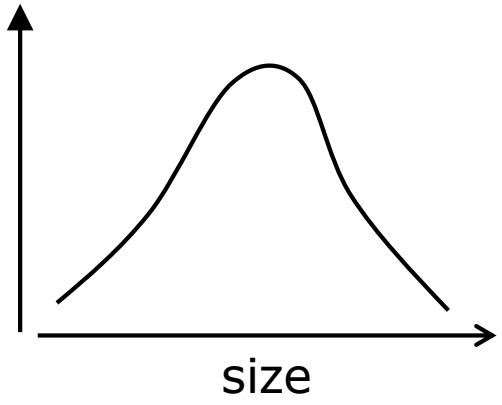
- front of list, back of list, in size order, in addr order?

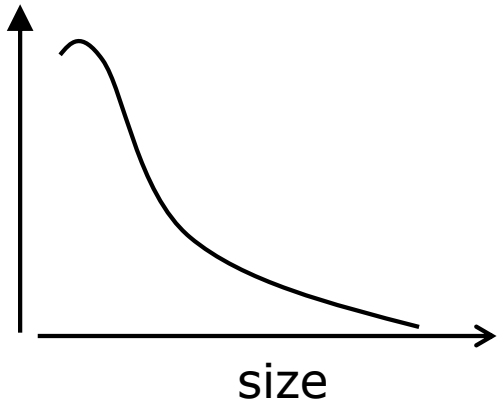
how often to coalesce?

- whenever memory is freed?
- every so often?
- never?

How to model workload?

What is the distribution of amount of requested memory?





Appendix A. Evaluation of Expected Allocation/Request Ratio

S_i	del	cdf (%)	pdf (%)	contr to avg req
2		0		
8	6	36	6	1.98
10	2	44	4	.76
15	5	54	2	1.30
25	10	84	3	6.15
30	5	94	2	2.8
35	5	96.5	.5	.825
40	5	97.5	.2	.38
50	10	98.5	.1	.455
70	20	99.3	.04	.484
100	30	99.6	.01	.2565
200	100	100	.004	.602

Average request = 15.9925
$\text{del} = S_i - S_{i-1}$.
pdf_i applies to range $[S_{i-1}, S_i]$.
$\text{cdf}_i = \text{cdf}_{i-1} + \text{del} \times \text{pdf}_i$.
contr to avg req $= \sum_{j=l_{i-1}+1}^{l_i} j \cdot \text{pdf}_i$.
The pdf corresponds to that in [4].

from Hirschberg, "A Class of Dynamic Memory Allocation Algorithms"

What metrics to use?

- Metric 1: how often are requests satisfied?
  - can test in simulation or with actual code
- Metric 2: how fast is the memory allocator?
  - best tested with actual code

Exercise

- Name three design goals
- Name three (policy) design choices

Summary

- Memory allocation happens at the OS and user levels
- Typically the allocator starts with a big chunk of free memory, then tracks what remains free after malloc and free calls
- It is a challenging design problem:
  - lots of design choices
  - lots of design goals