CMSC421: Principles of Operating Systems

Nilanjan Banerjee

Assistant Professor, University of Maryland
Baltimore County
nilanb@umbc.edu
http://www.csee.umbc.edu/~nilanb/teaching/421/

Acknowledgments: Some of the slides are adapted from Prof. Mark Corner and Prof. Emery Berger's OS course at Umass Amherst

Announcements

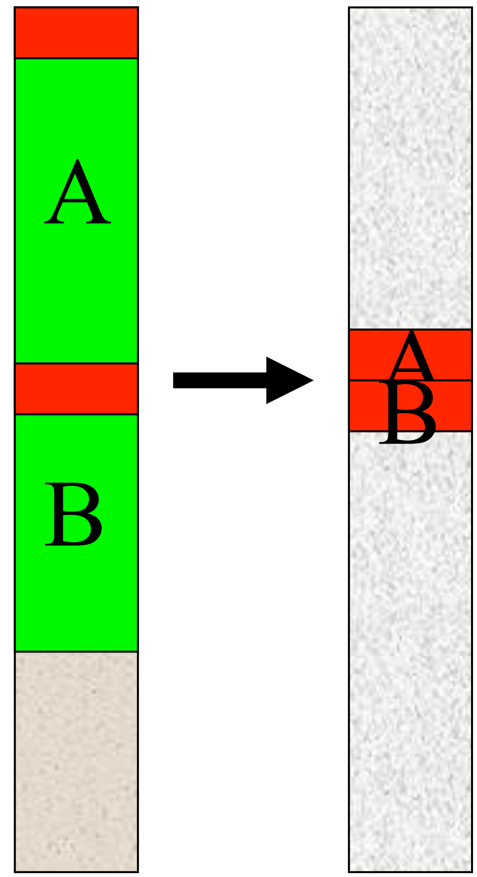
- Project 2 progress report due on Nov. 9th
- Homework 3 will be out soon (hopefully before the end of this week)

Quick Activity

- How much mem does a page table need?
  - 4kB pages, 32 bit address space
  - page table entry (PTE) uses 4 bytes
- 2^32/2^12*4=2^22 bytes=4MB
  - Is this a problem?
  - Isn’t this per process?
  - What about a 64 bit address space?
- Any ideas how to fix this?

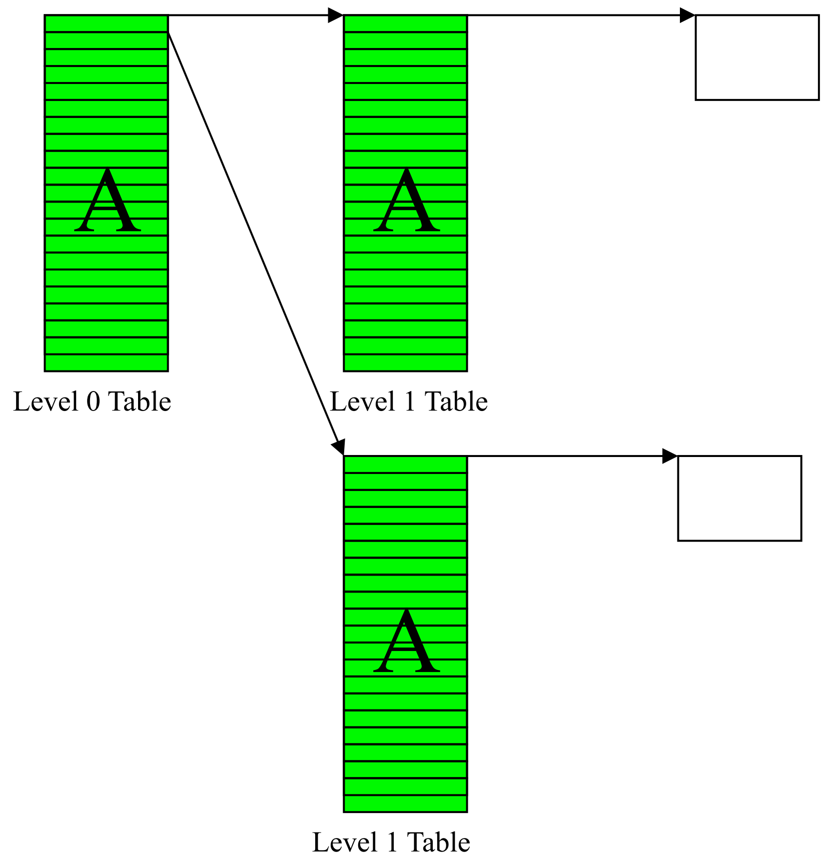
Locality

- Most programs obey 90/10 "rule"
  - 90% of time spent accessing 10% of memory
- Exploit this rule:
  - Only keep "live" parts of process in memory



Multi-Level Page Tables

- Use a multi-level page table



Quick Activity

- How much mem does a page table need?
  - 4kB pages, 32 bit address space
  - Two level page table
  - 20bits = 10 bits each level
  - page table entry (PTE) uses 4 bytes
  - Only first page of program is valid
- 2^10*4+2^10*4=2^13 bytes=8kB

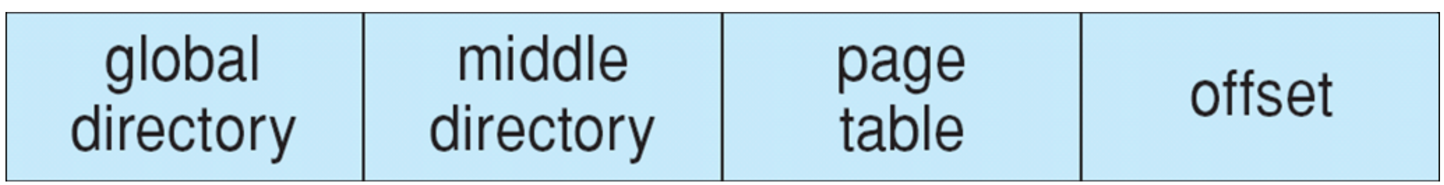
- Isn’t this slow?

Translation Lookaside Buffer (TLB)

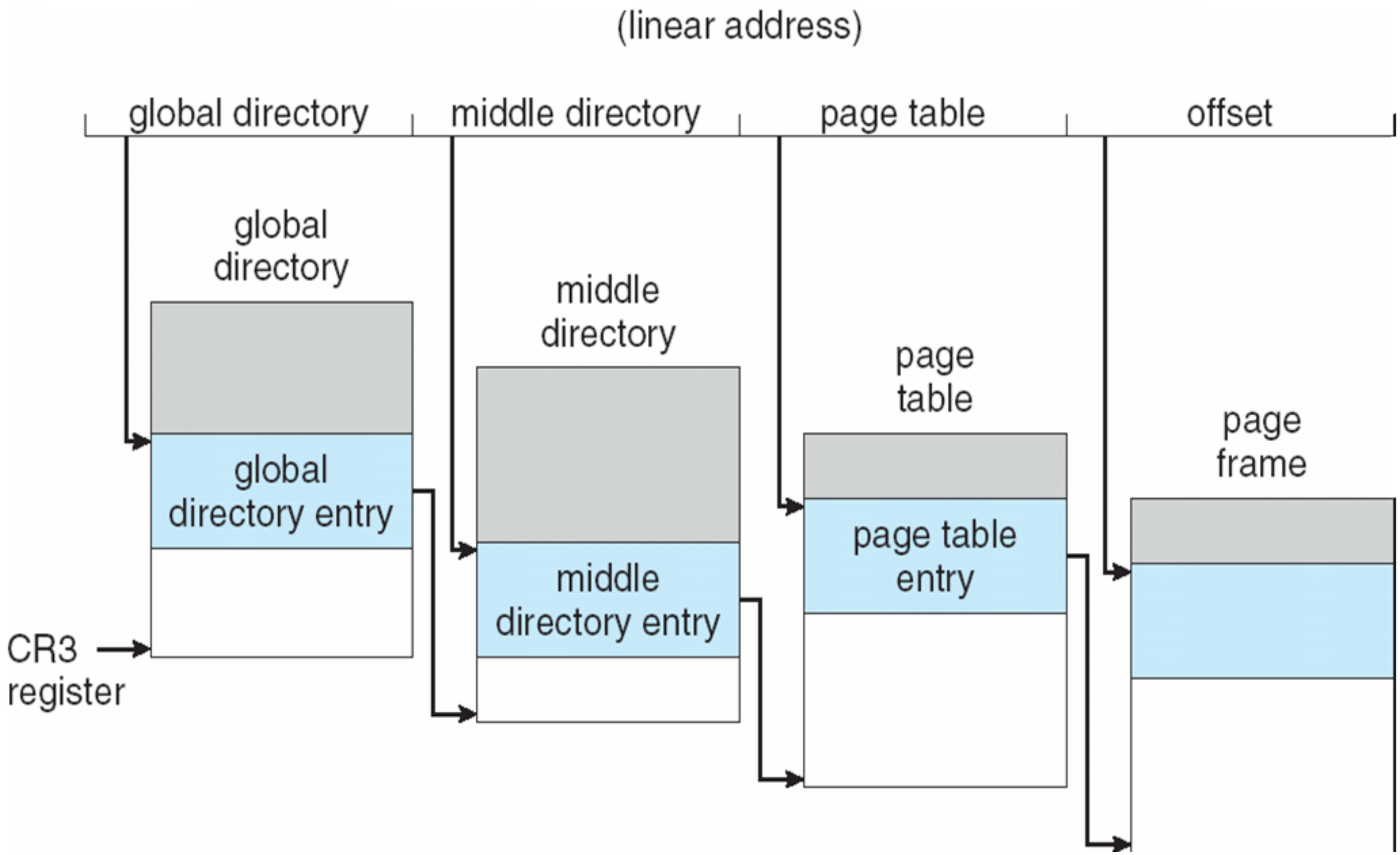
- TLB: fast, fully associative memory
  - Caches page table entries
  - Stores page numbers (key) and frame (value) in which they are stored
- Assumption: locality of reference
  - Locality in memory accesses = locality in address translation
- TLB sizes: 8 to 2048 entries
  - Powers of 2 simplifies translation of virtual to physical addresses

Linear Address in Linux

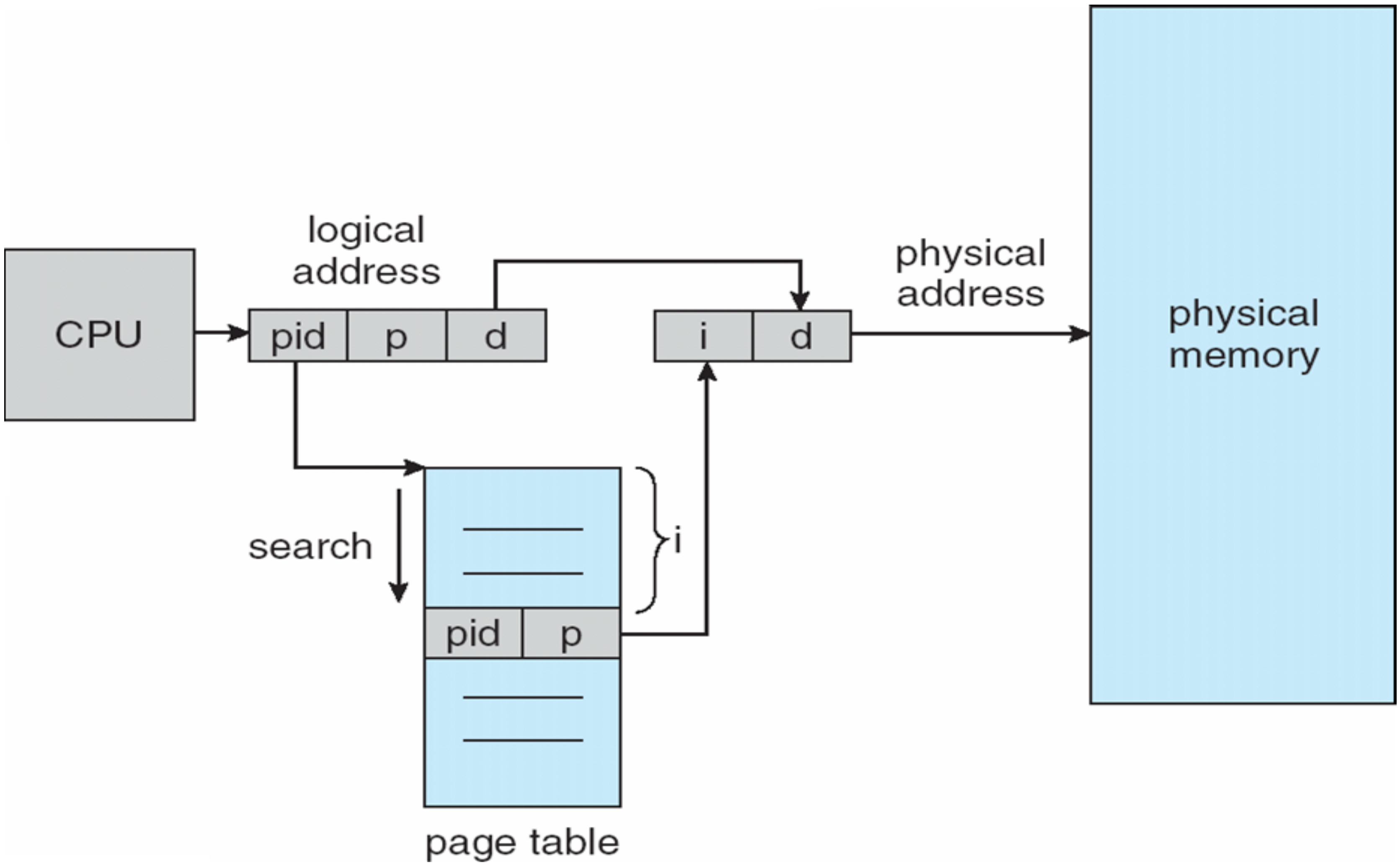
- Uses a three-level paging strategy that works well for 32-bit and 64-bit systems
- Linear address broken into four parts:



Three-level Paging in Linux

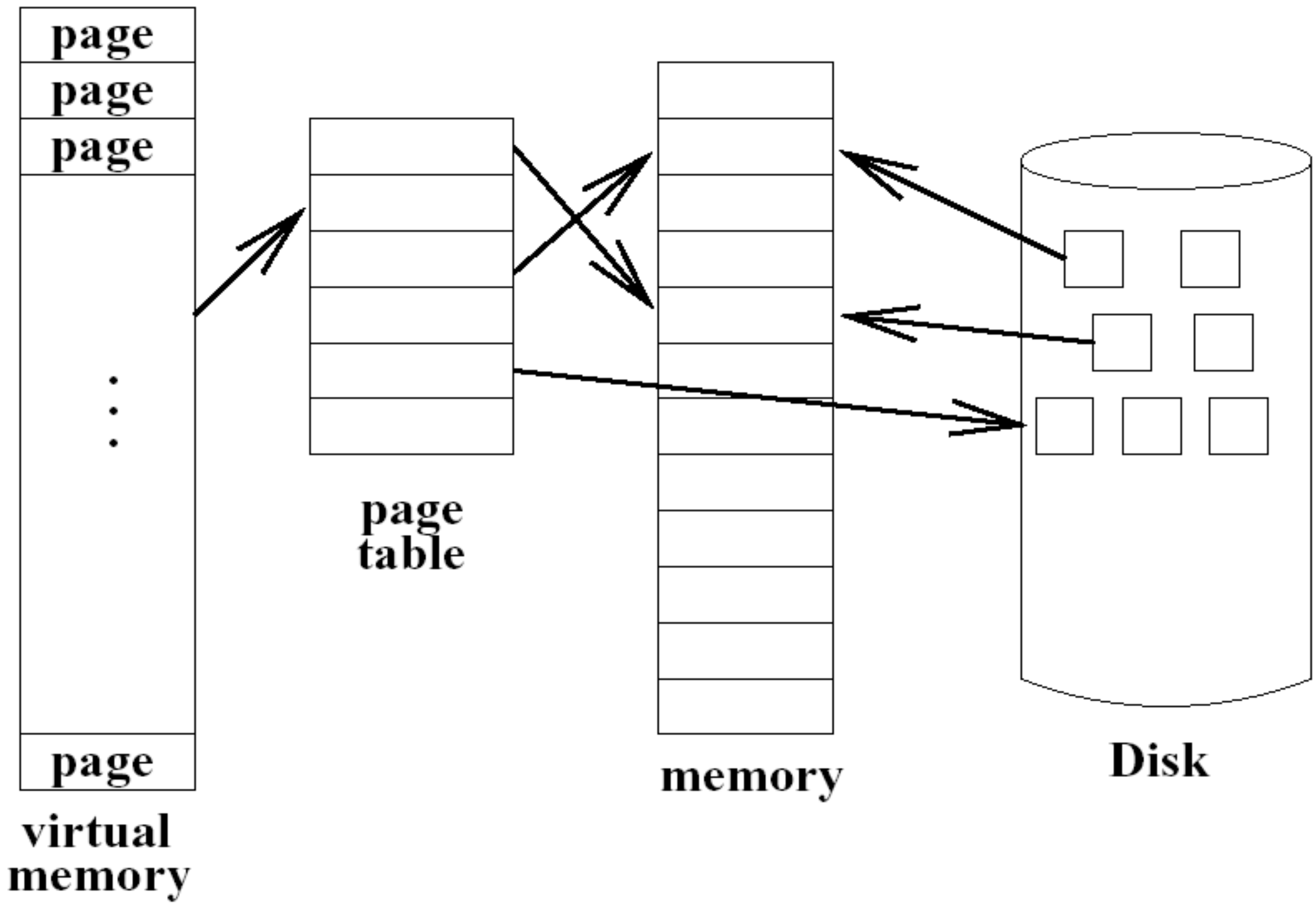


Inverted Page Tables

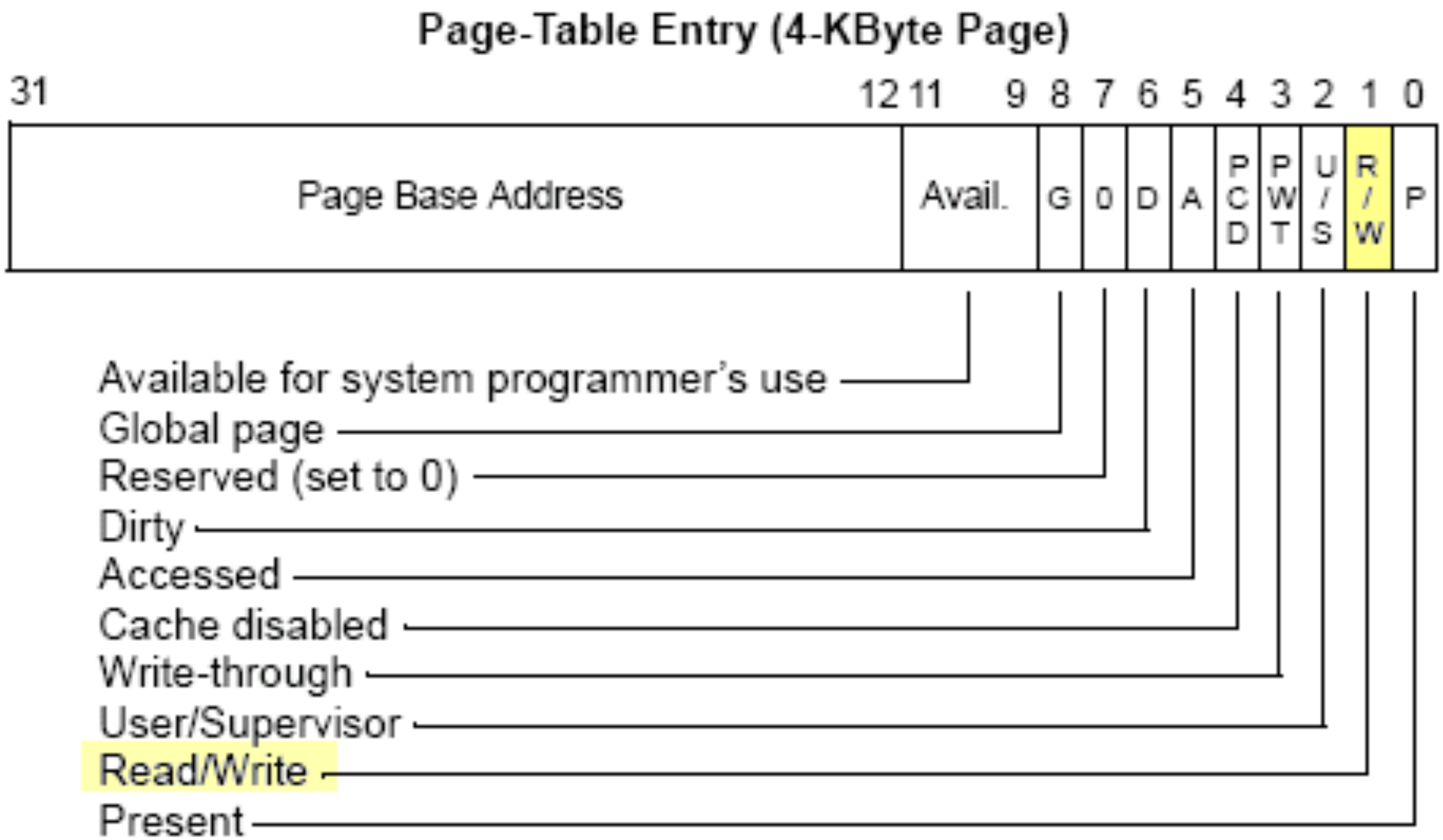


Maintain one global page table for all processes

Swap Space



Page table entry and page faults

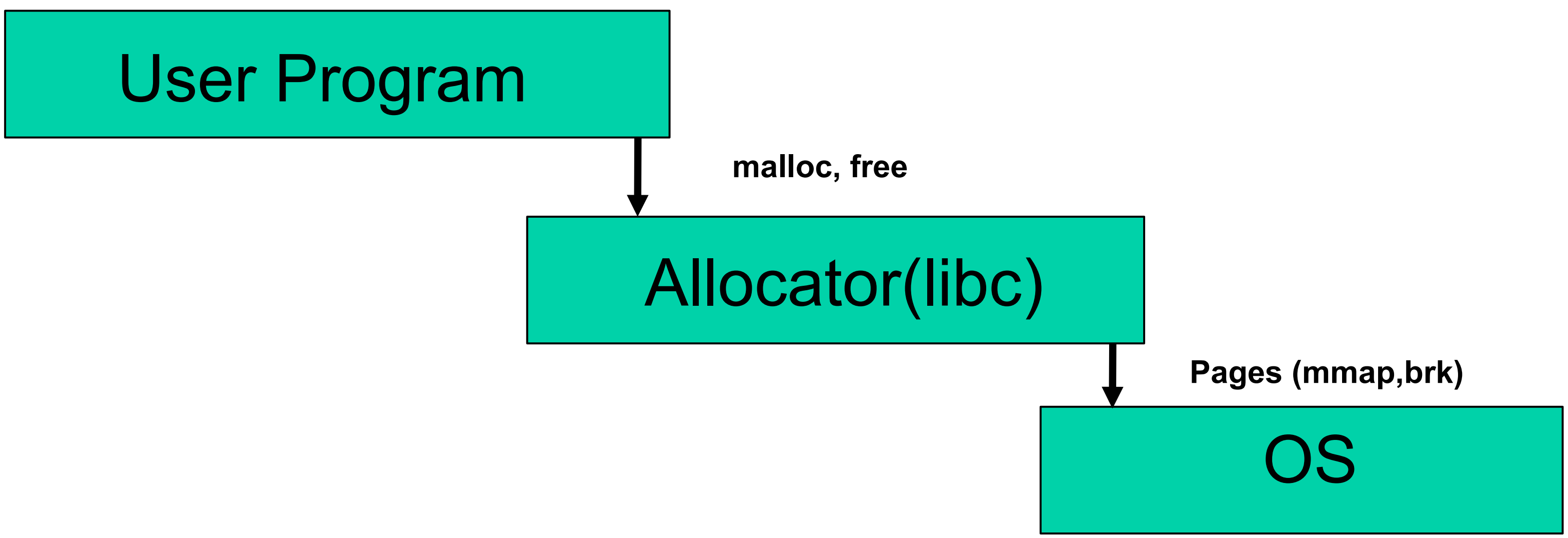


User-space memory allocation in the heap (malloc)

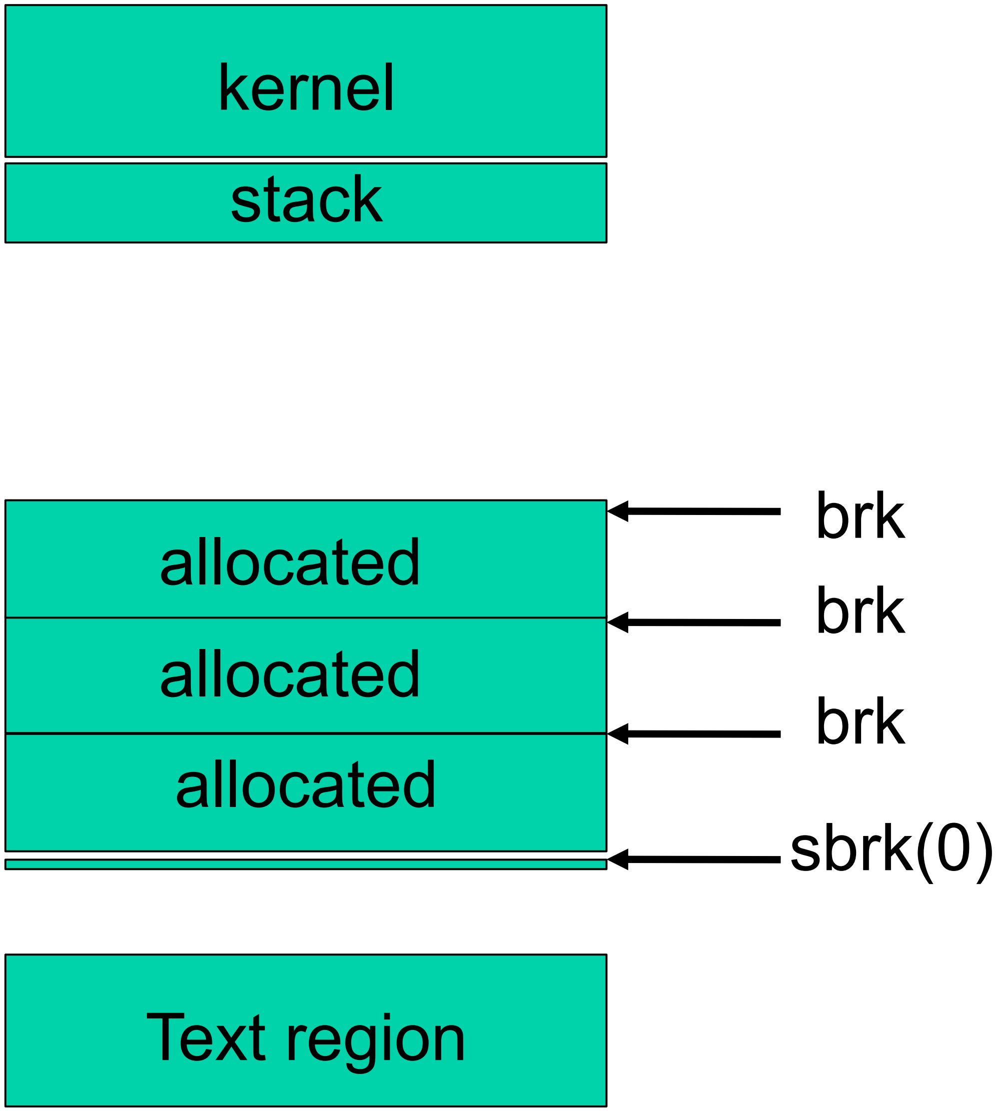
What happens

```
int *arg = (int *)malloc(sizeof(int))
```

- Programs ask memory manager
  - to allocate/free objects (or multiple pages)
- Memory manager asks OS
  - to allocate/free pages (or multiple pages)



User-space memory allocation in the heap (malloc)



sbrk(100)

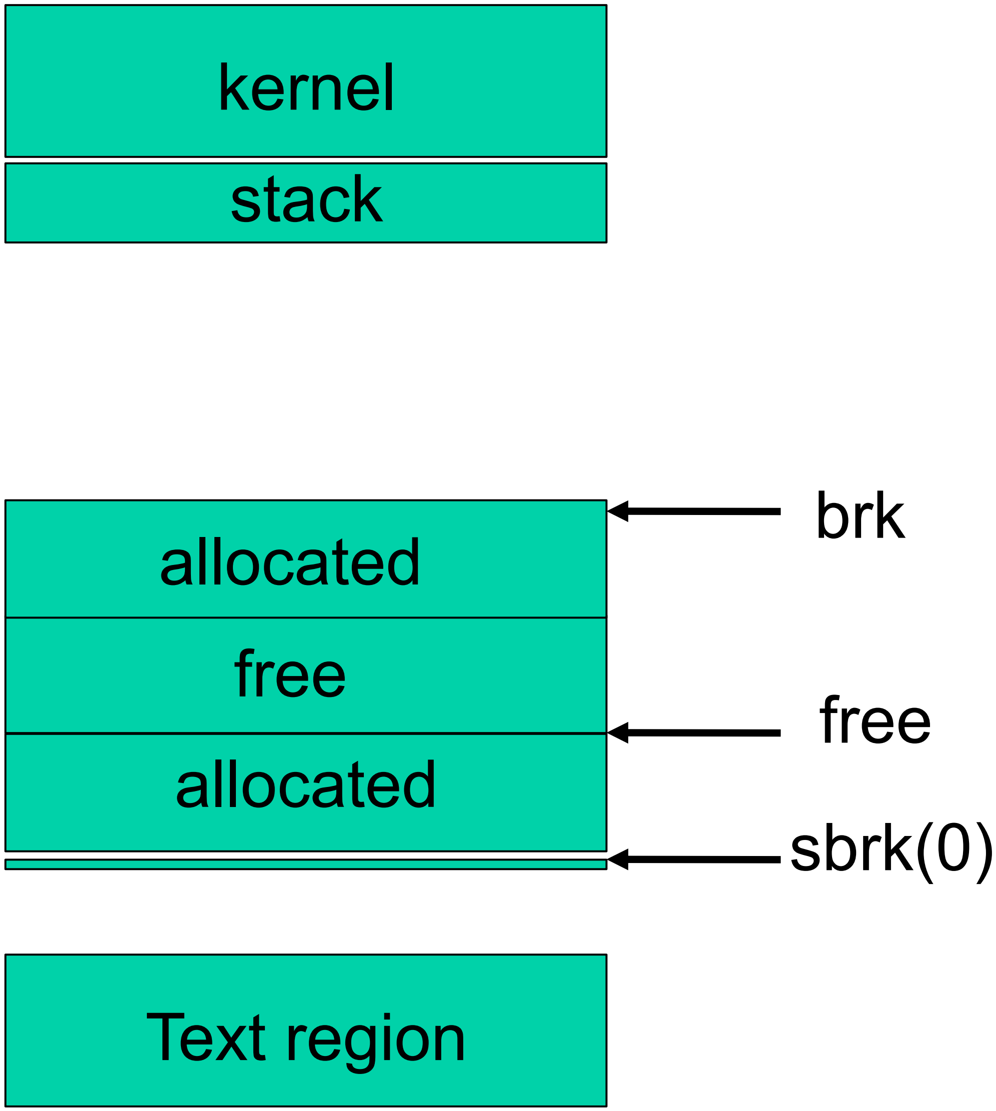
sbrk(100)

sbrk(100)

sbrk(0)

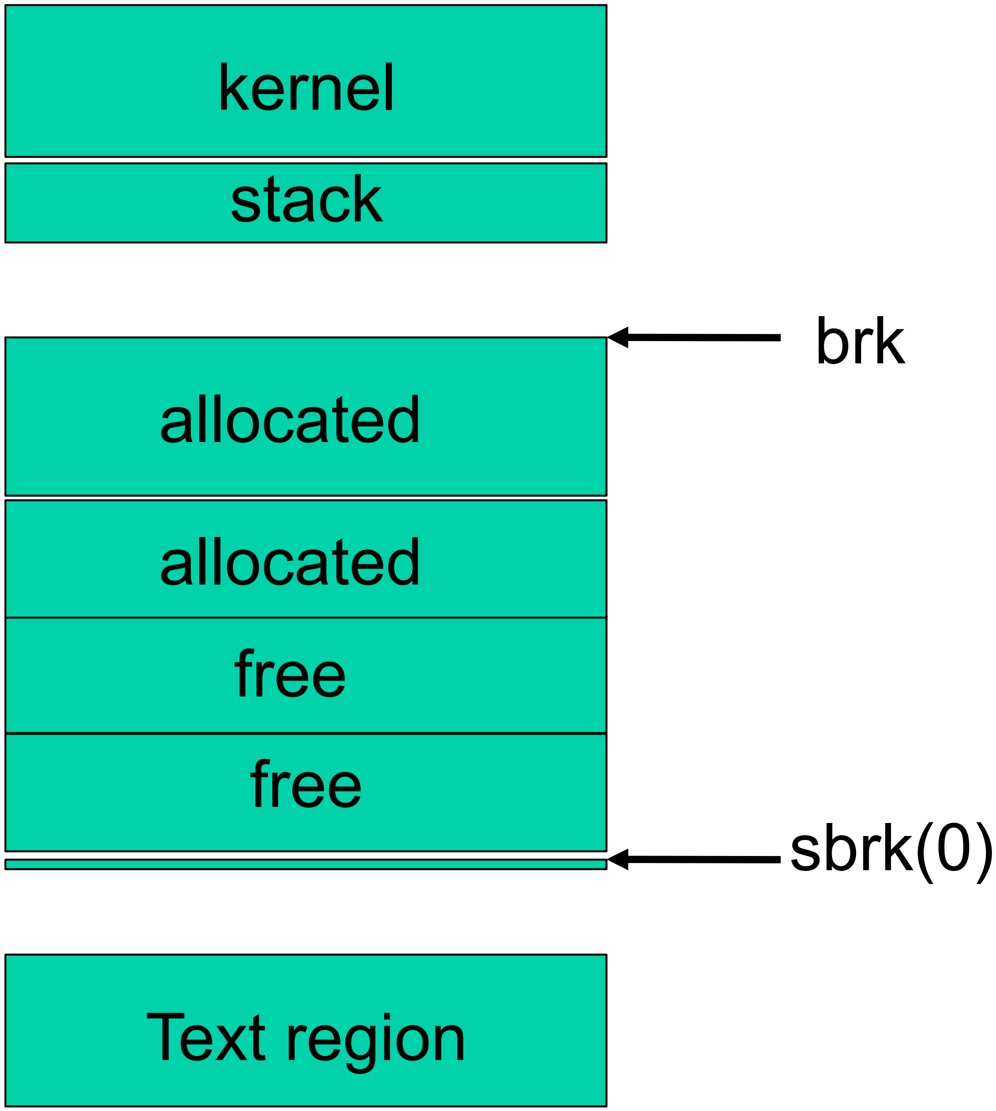
A demo?

User-space memory allocation in the heap (malloc)



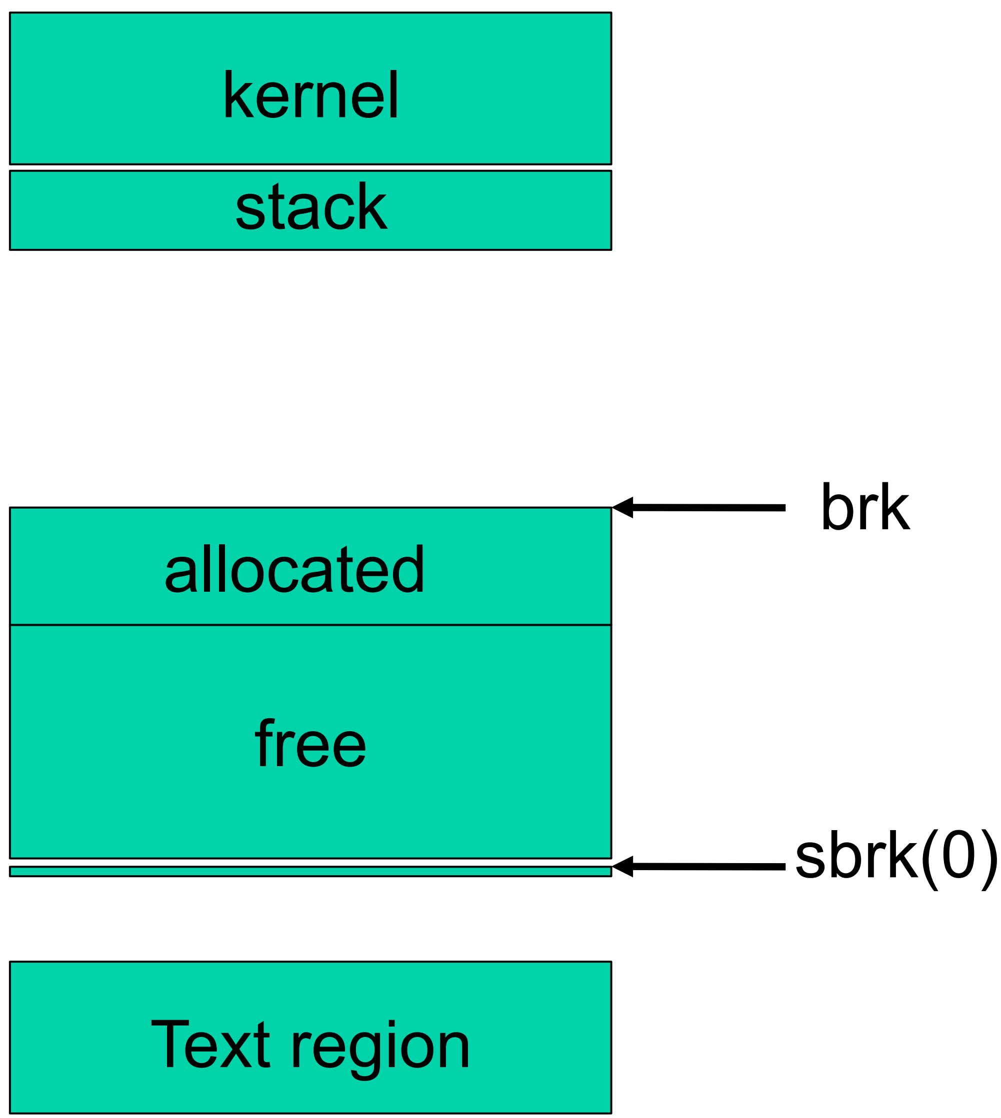
Memory allocator keeps track of the free blocks

Few Scenarios



malloc(200)
Waste of resources

Coalescing

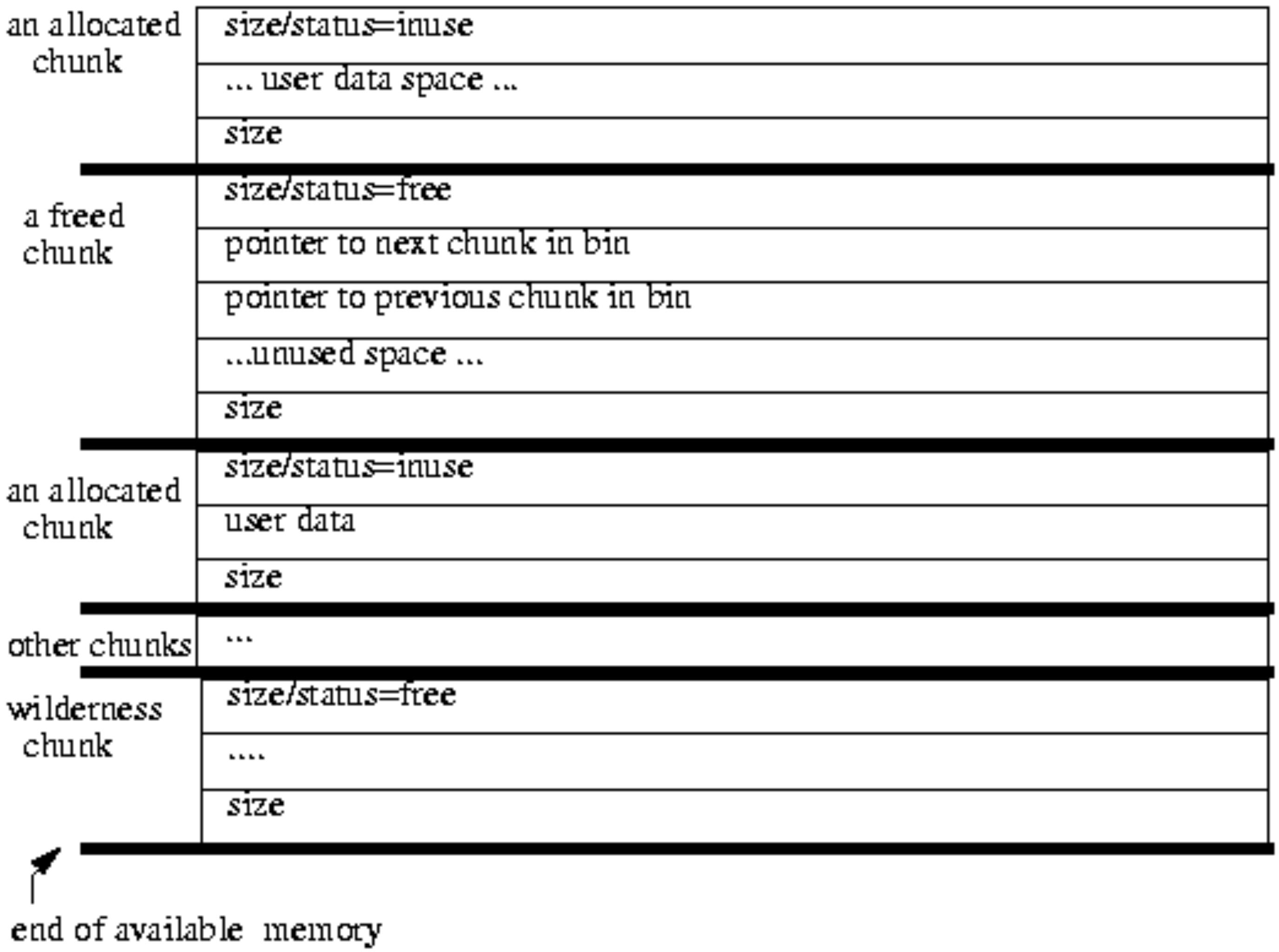


malloc(200)
Waste of resources

Jobs of a memory allocator like malloc

- Manage heap space in virtual memory
  - Use sbrk to ask for more memory from OS
- Coalescing
  - Keep track of free blocks
  - Merge them together when adjacent blocks are free
- Malloc needs to be really fast
  - Decide which free block to allocate
  - Lets take a look at the data structure that is used for implementing malloc and free

Memory layout of the heap



this linked list can be ordered in different ways

http://gee.cs.oswego.edu/dl/html/malloc.html

Selecting the free block to allocate: Fragmentation

- Intuitively, fragmentation stems from “breaking” up heap into unusable spaces
  - More fragmentation = worse utilization
- External fragmentation
  - Wasted space outside allocated objects
- Internal fragmentation
  - Wasted space inside an object

Classical Algorithms

- First-fit
  - find first chunk of desired size

Classical Algorithms

- Best-fit
  - find chunk that fits best
    - Minimizes wasted space

Classical Algorithms

- Worst-fit
  - find chunk that fits worst
  - name is a misnomer!
  - keeps large holes around

- Reclaim space: coalesce free adjacent objects into one big object

In-class Discussion