

CS202 – Advanced Operating Systems

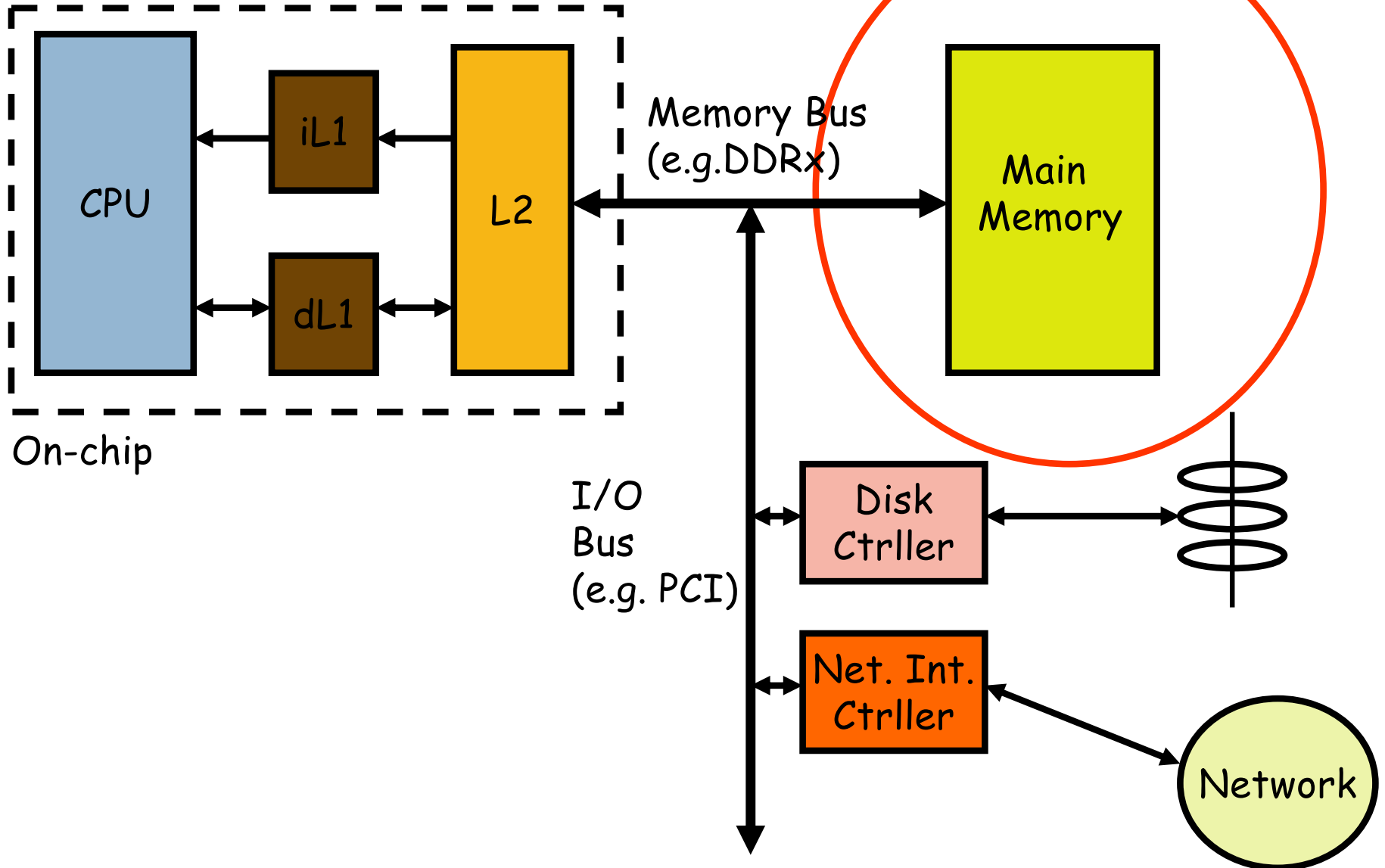
Memory Management

January 15, 2025

Check your understanding

2

- True or False: a process can move from the running state to the waiting state
 - ▣ Yes, when the process asks for a blocking system call
- True or False: There is a separate kernel stack and user stack for each process
 - ▣ Yes, its dangerous to allow a process to access an OS page
- Where is process related information stored?
 - ▣ In the Process Control Block



Need for Memory Management



- Physical memory (DRAM) is limited
 - ▣ A single process may not all fit in memory
 - ▣ Several processes (their address spaces) also need to fit at the same time
- Disallow any process from accessing another processes memory.
 - ▣ Each process may have different rights to memory

Solution strategies



- **Contiguous allocation**

- ▣ The requested size is granted as one big contiguous chunk.

- From the process's (programmer's) perspective

- **Non-contiguous**

- ▣ The requested size is granted as several pieces (and typically each of these pieces is of the same – fixed - size).

Best of Both!



□ Virtual Memory

- Contiguous memory region for the process
 - Virtual memory
- Use memory device to “back” process flexibly
 - Physical memory

□ Process

- Get a contiguous region for the stack – extends downward
- And the heap - Extend the heap as needed – malloc/free

□ Memory Use

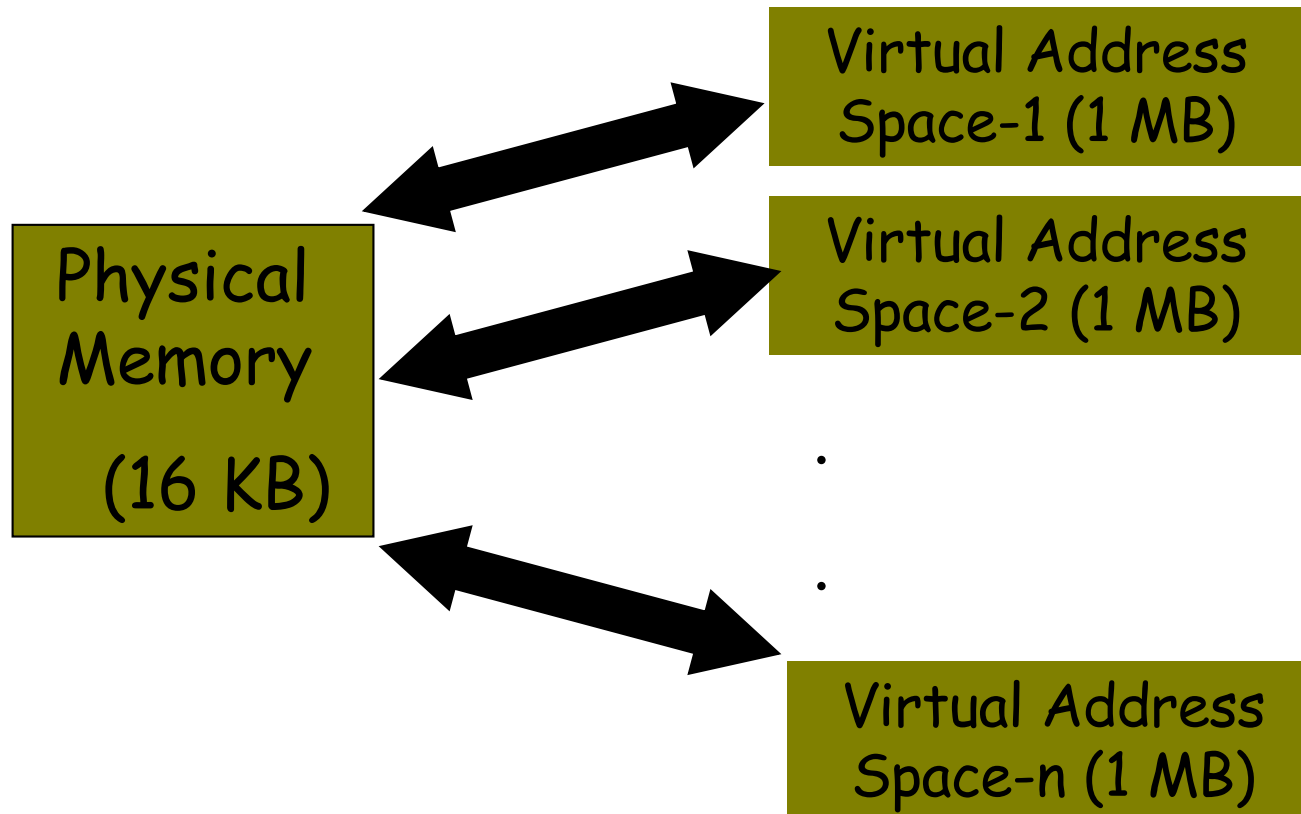
- Manage by the OS
- OS provides memory to back extension requests on demand
- OS manages use of memory among multiple processes


Non-contiguous Allocation



- You may get several chunks of physically separate DRAM space.
- However, these get stitched together as 1 contiguous “virtual” space that the program sees!

Example




- 
- Programs are provided with a virtual address space (say 1 MB).
 - Role of the OS to fetch data from either physical memory or disk.
 - Done by a mechanism called **paging**.
 - Divide the virtual address space into units called “virtual pages” each of which is of a fixed size (usually 4K or 8K).
 - Previous example, we have 256 4K-sized pages.
 - Divide the physical address space into “physical frames”
 - Previous example, we would have 4 4K-sized frames.

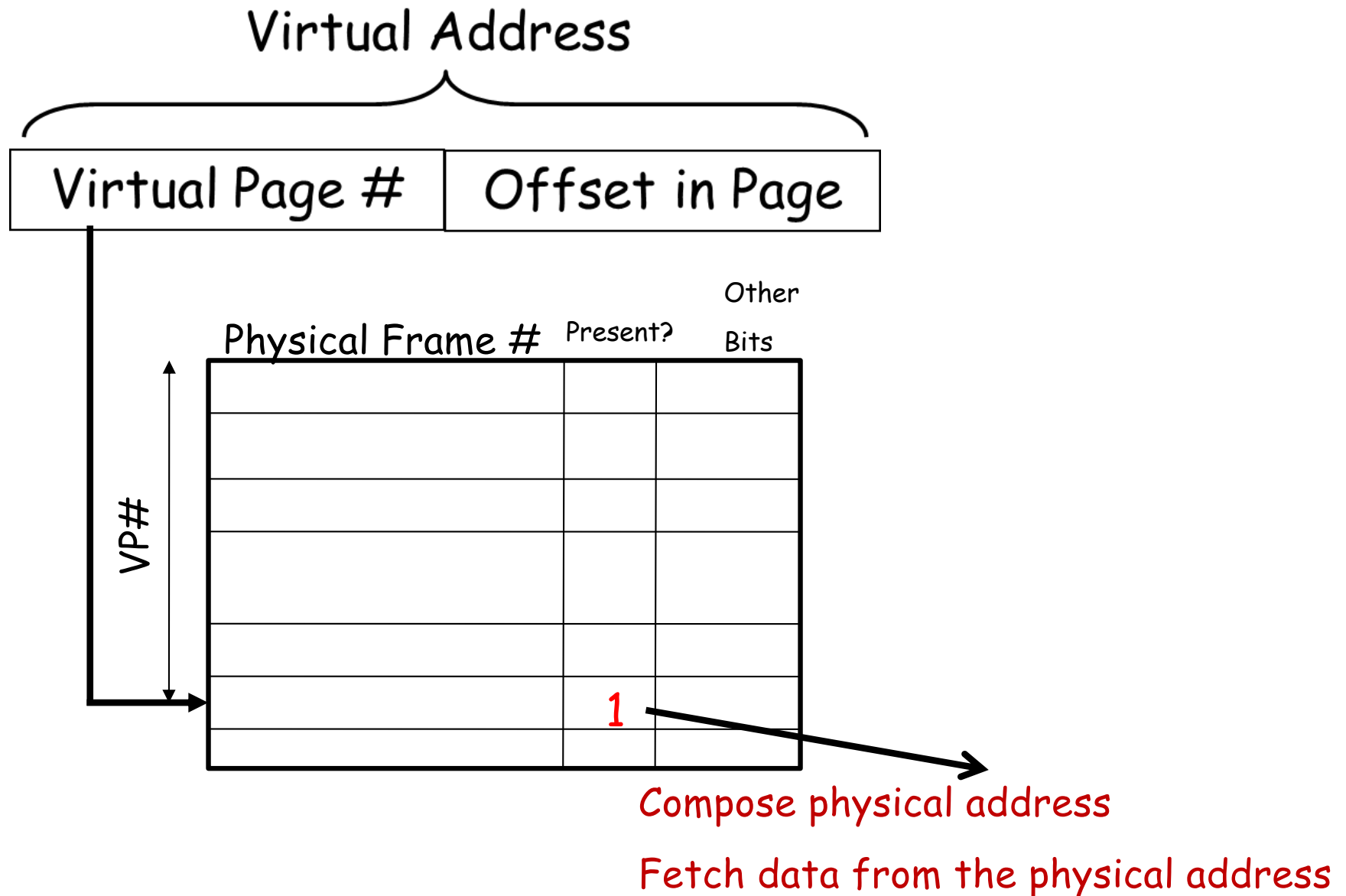
Basic Idea



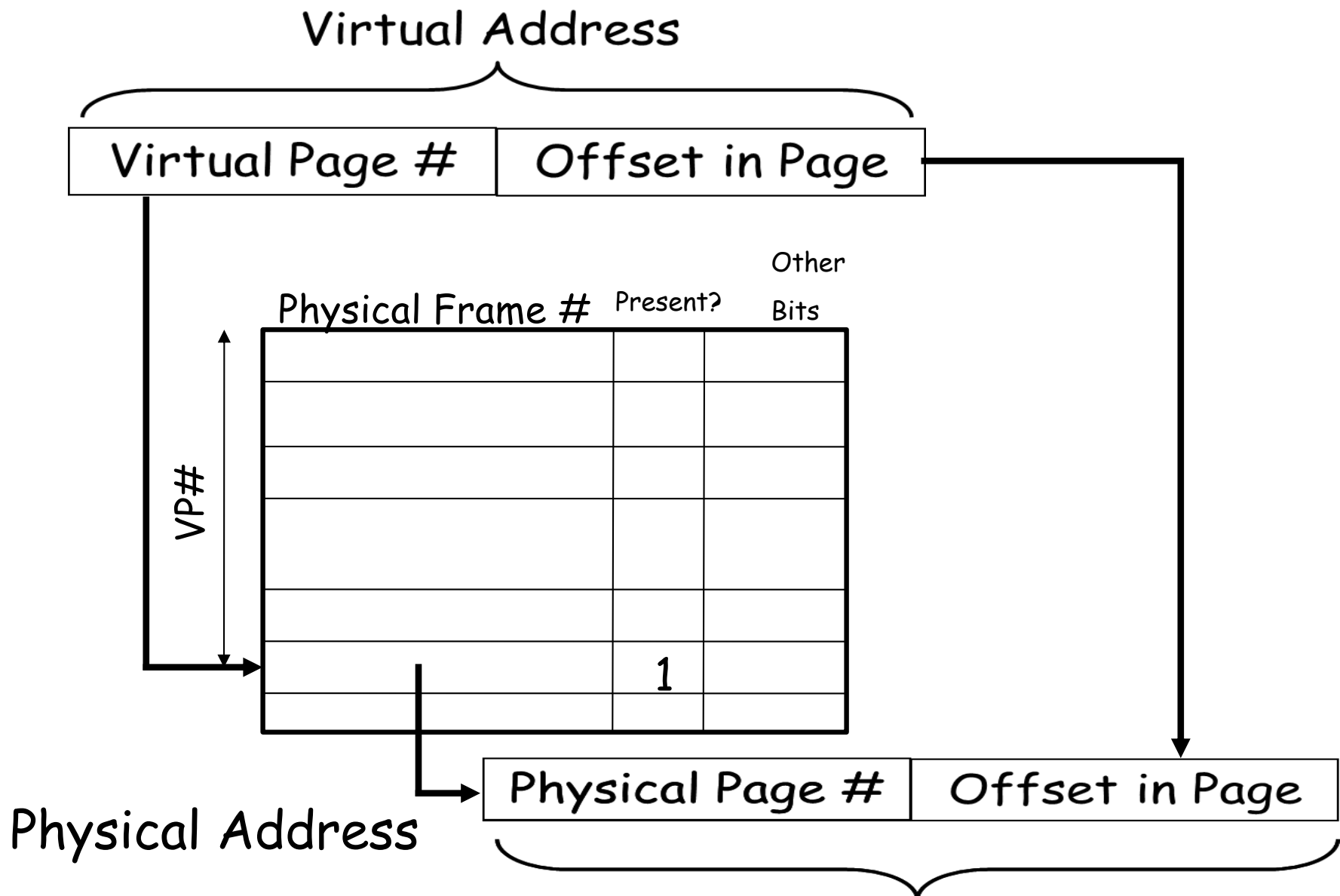
- At any time, the number of virtual pages in memory is limited by the number of physical frames.
- Rest would have to be stored on disk in a region called the “**swap space**”.
 - Previous example, we could only have 4 virtual pages residing in physical memory.
- A “**page**” is thus the unit of transfer between disk and physical memory.

- 
- Role of the OS to keep track of which virtual page is in physical memory and if so where?
 - Maintained in a data structure called the “page table” that the OS builds.
 - “Page tables” map Virtual-to-Physical addresses.

Page Tables



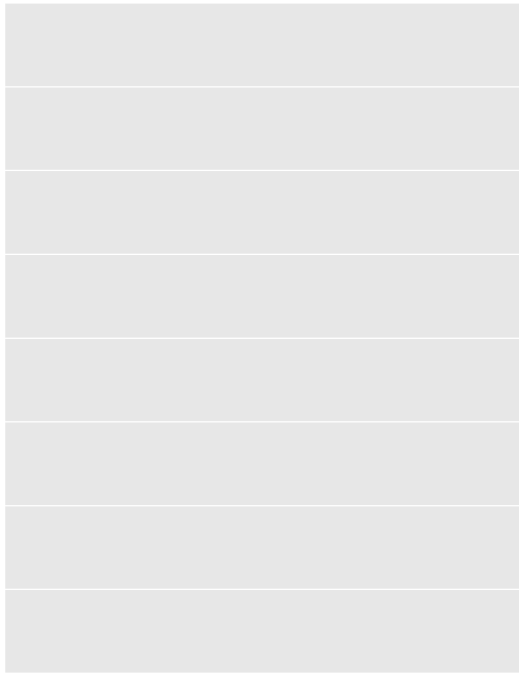
Page Tables



An example

Virtual address space
 $4\text{GB} = 2^{32}$ bytes

Page size
 $= 4096$ bytes
 $= 2^{12}$ bytes



No. of virtual pages ?

No. of bits in virtual address ?

An example

Virtual address space
 $4\text{GB} = 2^{32}$ bytes

Page size
 $= 4096$ bytes
 $= 2^{12}$ bytes



No. of virtual pages ?
 $= 2^{32} / 2^{12}$
 $= 2^{20}$

No. of bits in virtual address ?
 $= 32$ bits
 $= \{20 \text{ bits page id}\} \{12 \text{ bits page offset}\}$

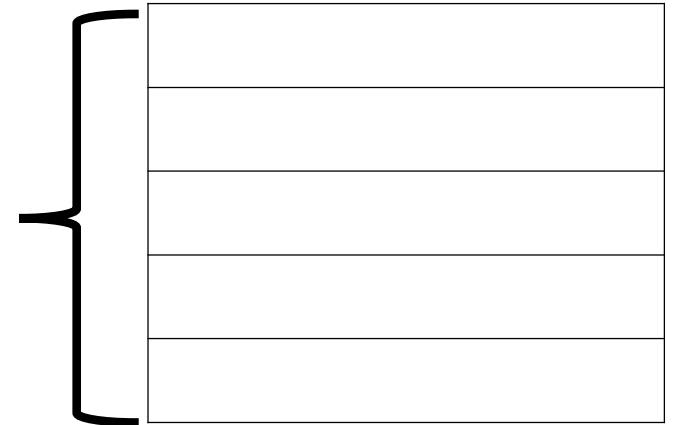
An example

Virtual address space
 $4\text{GB} = 2^{32}$ bytes



Page size
 $= 4096$ bytes
 $= 2^{12}$ bytes

Physical memory size
 $256\text{KB} = 2^{18}$ bytes



No. of physical frames ?

No. of bits in physical address ?

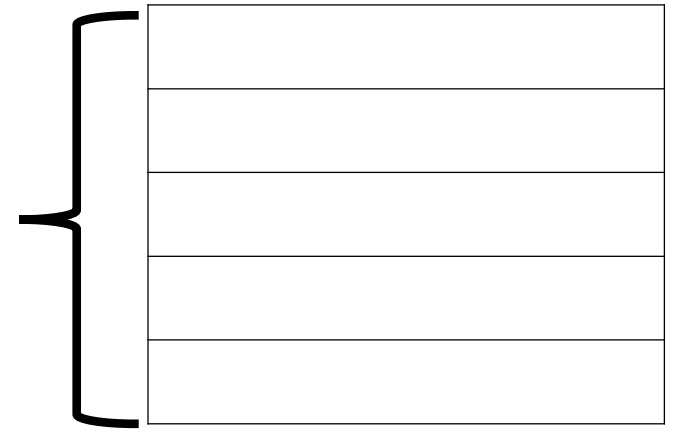
An example

Virtual address space
 $4\text{GB} = 2^{32}$ bytes



Page size
 $= 4096$ bytes
 $= 2^{12}$ bytes

Physical memory size
 $256\text{KB} = 2^{18}$ bytes

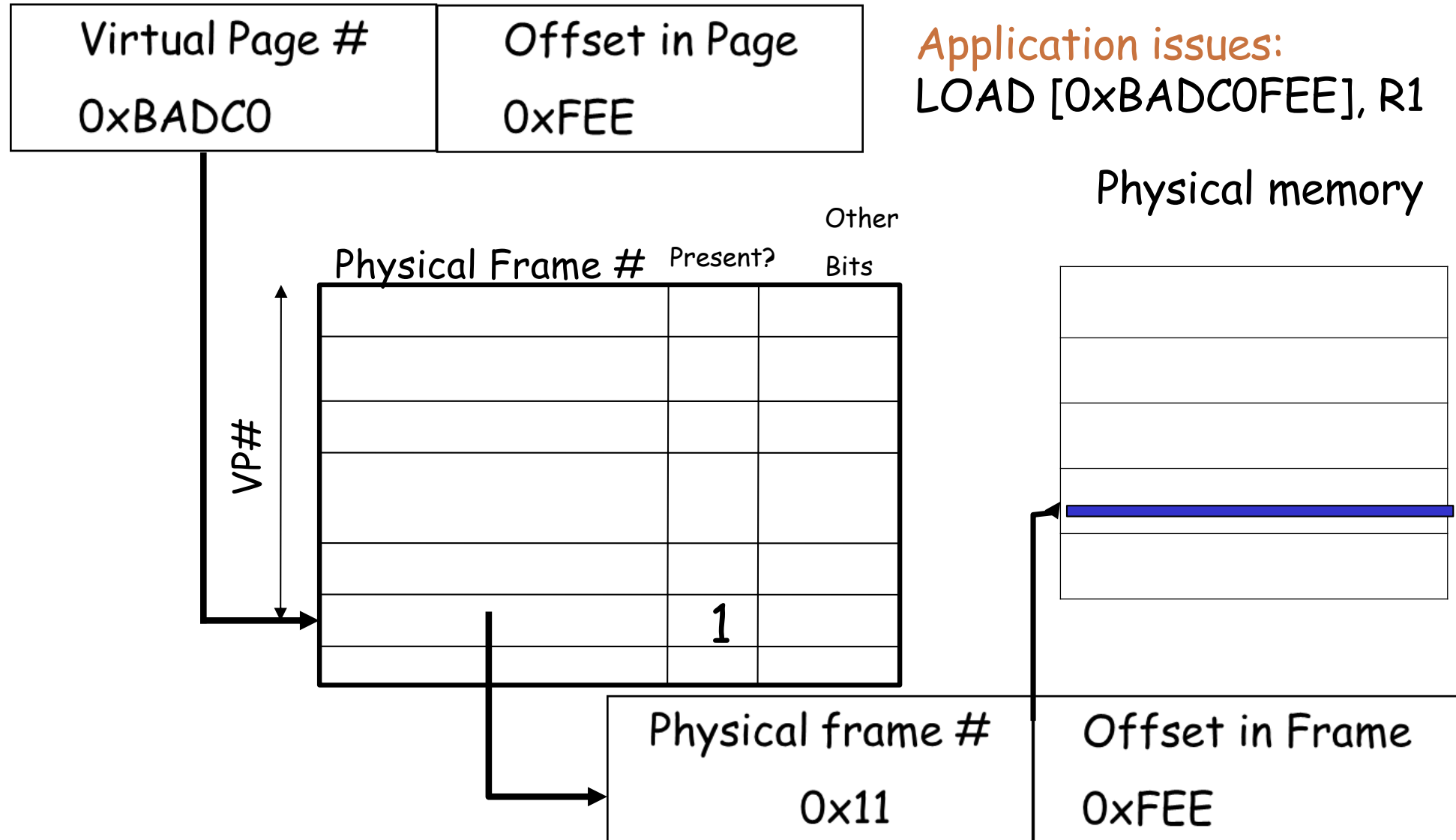



No. of physical frames
 $= 2^{18} / 2^{12}$
 $= 2^6$

Physical address
 $= 18$ bits

$= \{6 \text{ bits frame id}\} \{12 \text{ bits frame offset}\}$

Page table lookup



- 
- Page-table lookup needs to be done on every memory reference for both code & data!
 - Can be very expensive if this is done by software.
 - Usually done by a hardware unit called the **MMU** (Memory-Management Unit).
 - Located between CPUs and caches.

Role of the MMU



- Given a Virtual Address, index in the page-table to get the mapping.
- Check if the *present* bit in the mapping is set, and if so put out the physical address on the bus and let hardware do the rest.
- If it is not set, you need to fetch the data from the disk (swap-space).
 - We do not wish to do that in hardware!

More about Virtual Memory

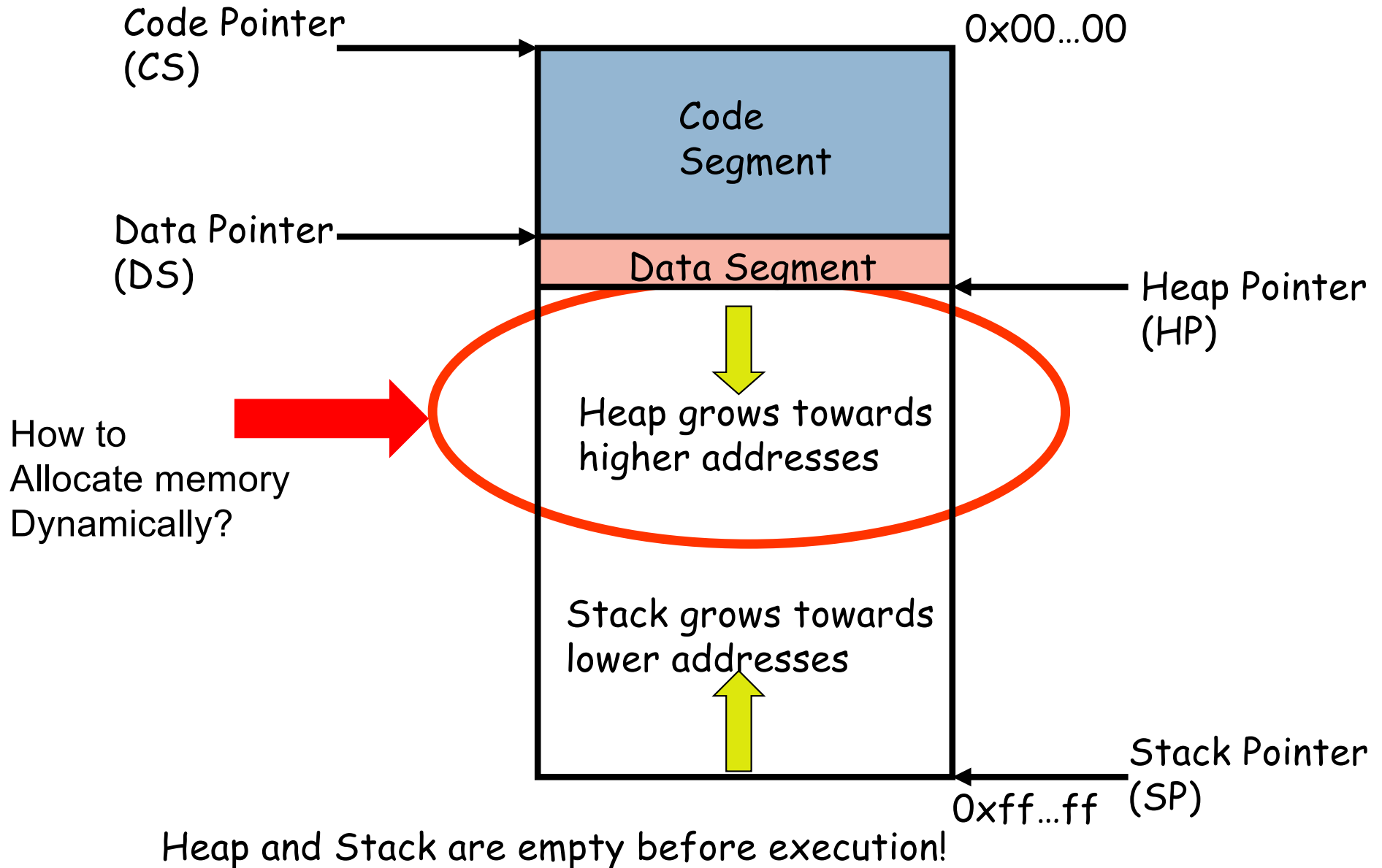
- Reduce Space Usage of Page Tables
 - ▣ Have **hierarchical page tables** (multi-level)
 - E.g., first level holds reference to second-level table, if needed, like in xv6 (for Project 1)
- Improve Performance of Address Translation
 - ▣ **Translation Lookaside Buffer** (TLB) is a cache of virtual page to physical frame mappings
 - Avoid page table walks due to multiple levels
- Page Eviction – Separate Policy from Mechanism
 - ▣ Which page to evict when memory is full?
 - ▣ **Page replacement policies** (e.g., least recently used, clock)

Contiguous Allocation

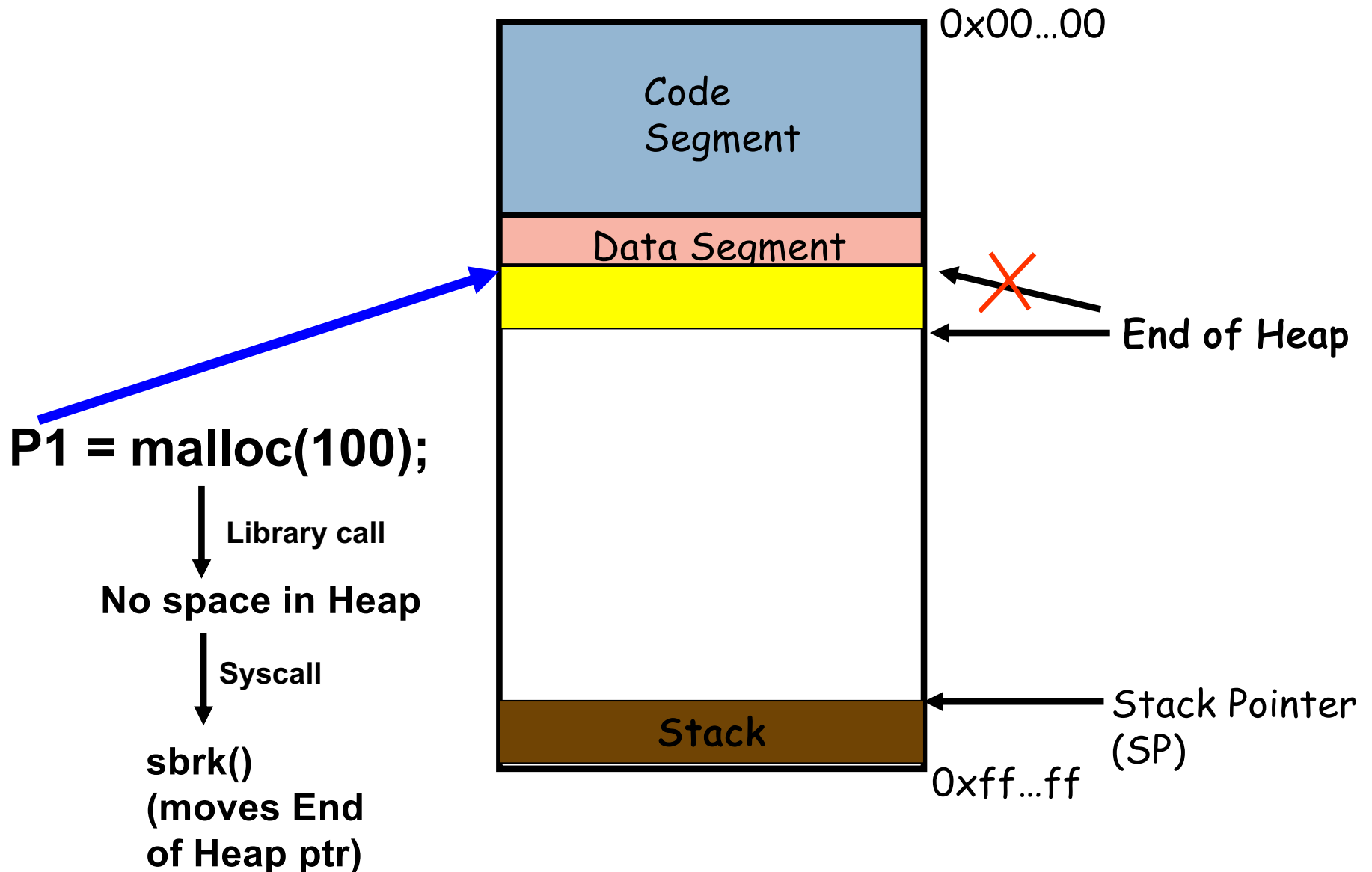


- Programmers want to work with contiguous memory
- How does the OS help programmers manage use of a contiguous memory space efficiently?
 - ▣ In terms of space
 - ▣ And in performance
- Discuss how **malloc** and **free** work

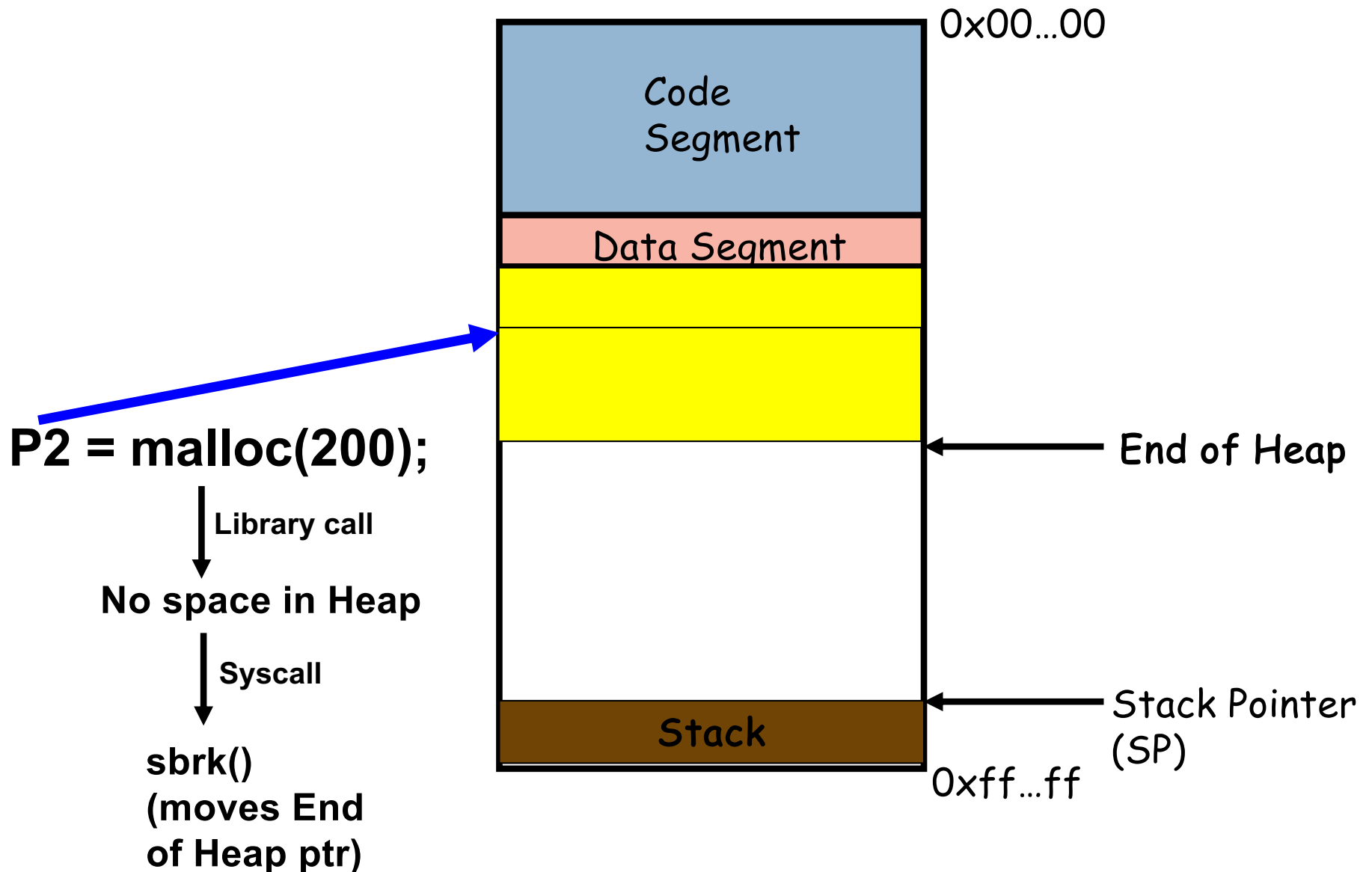
Address Space of a Process



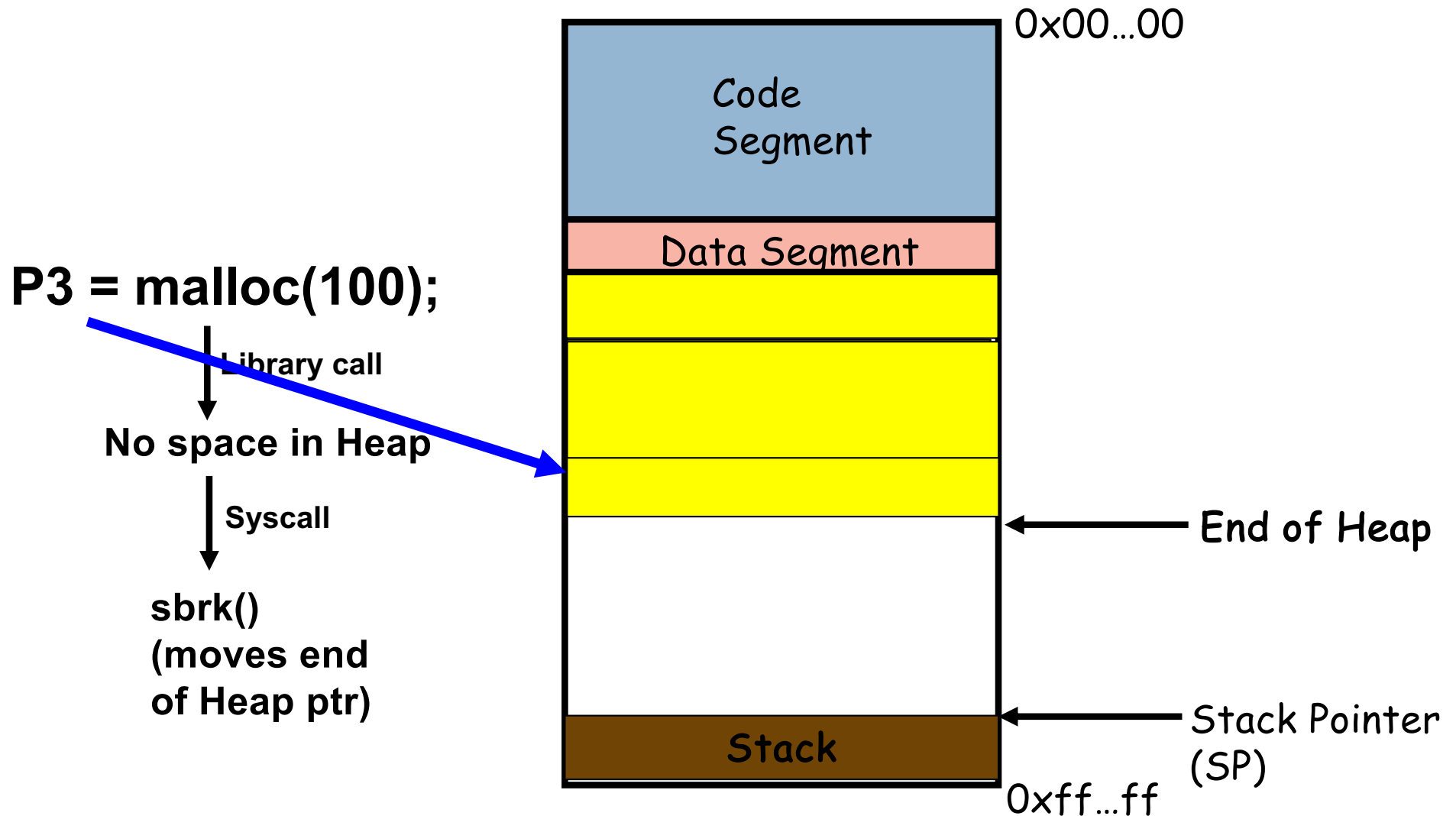
Malloc()



Malloc()

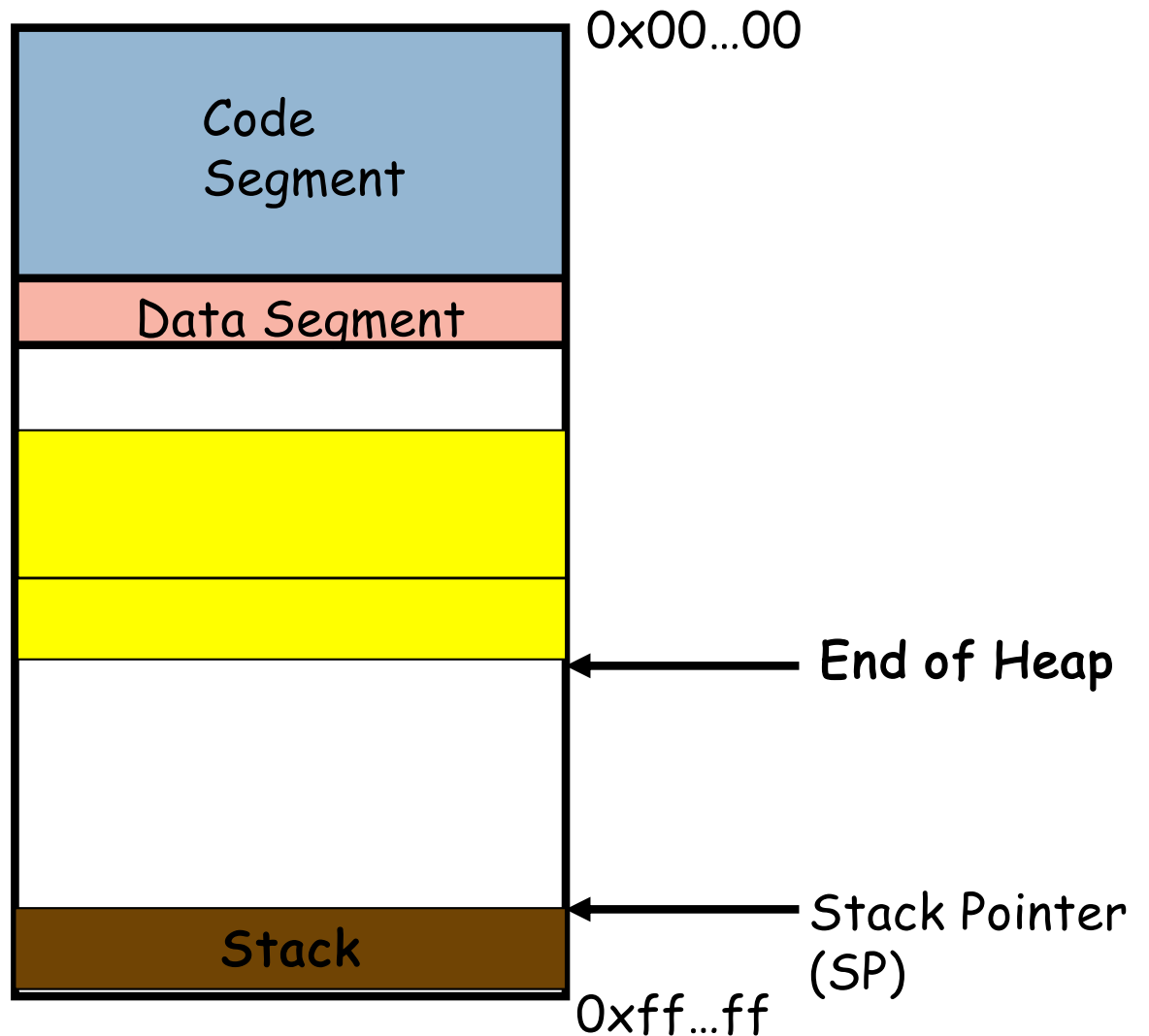


Malloc()



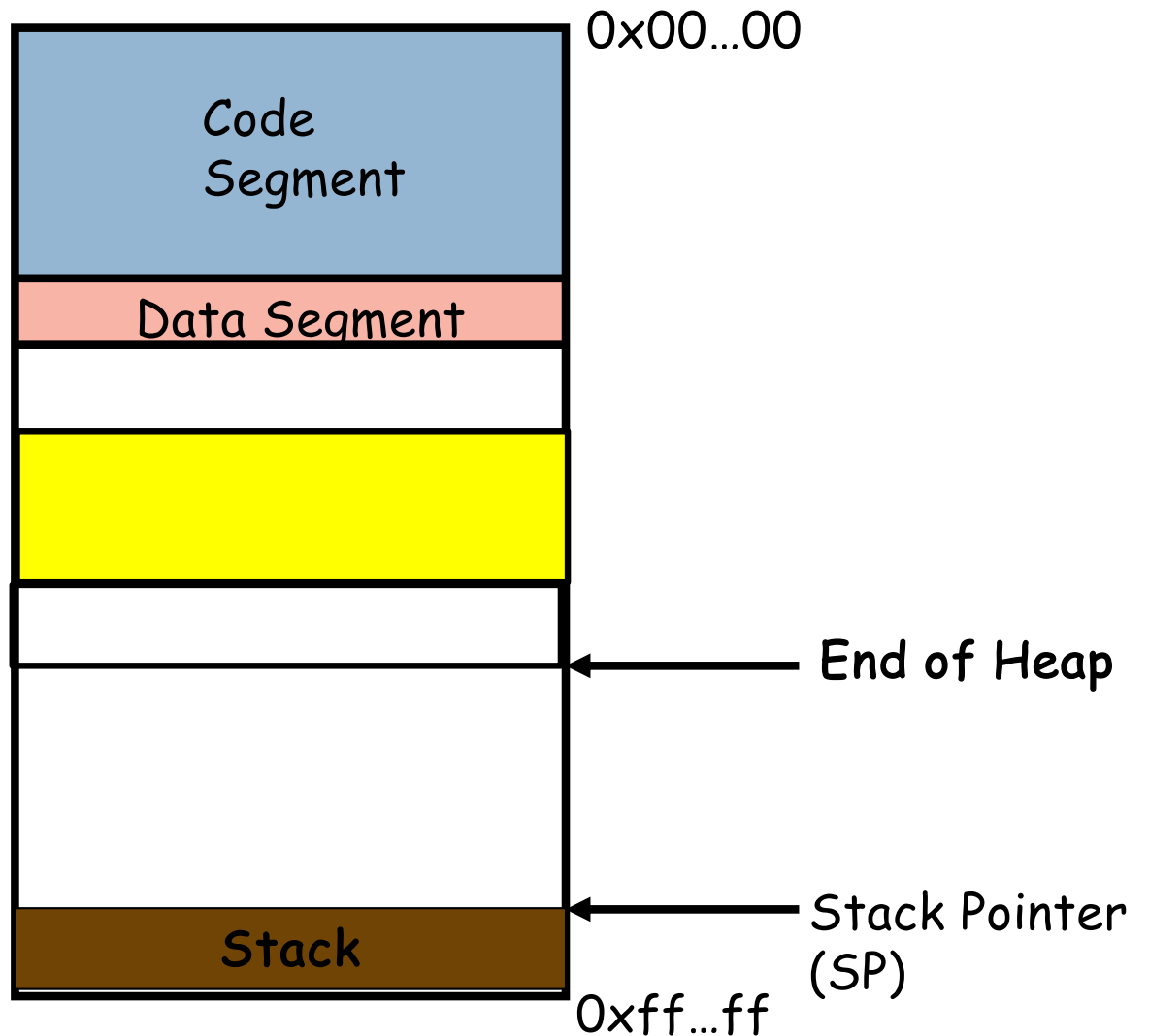
Malloc()

free(P1);



Malloc()

free(P3);

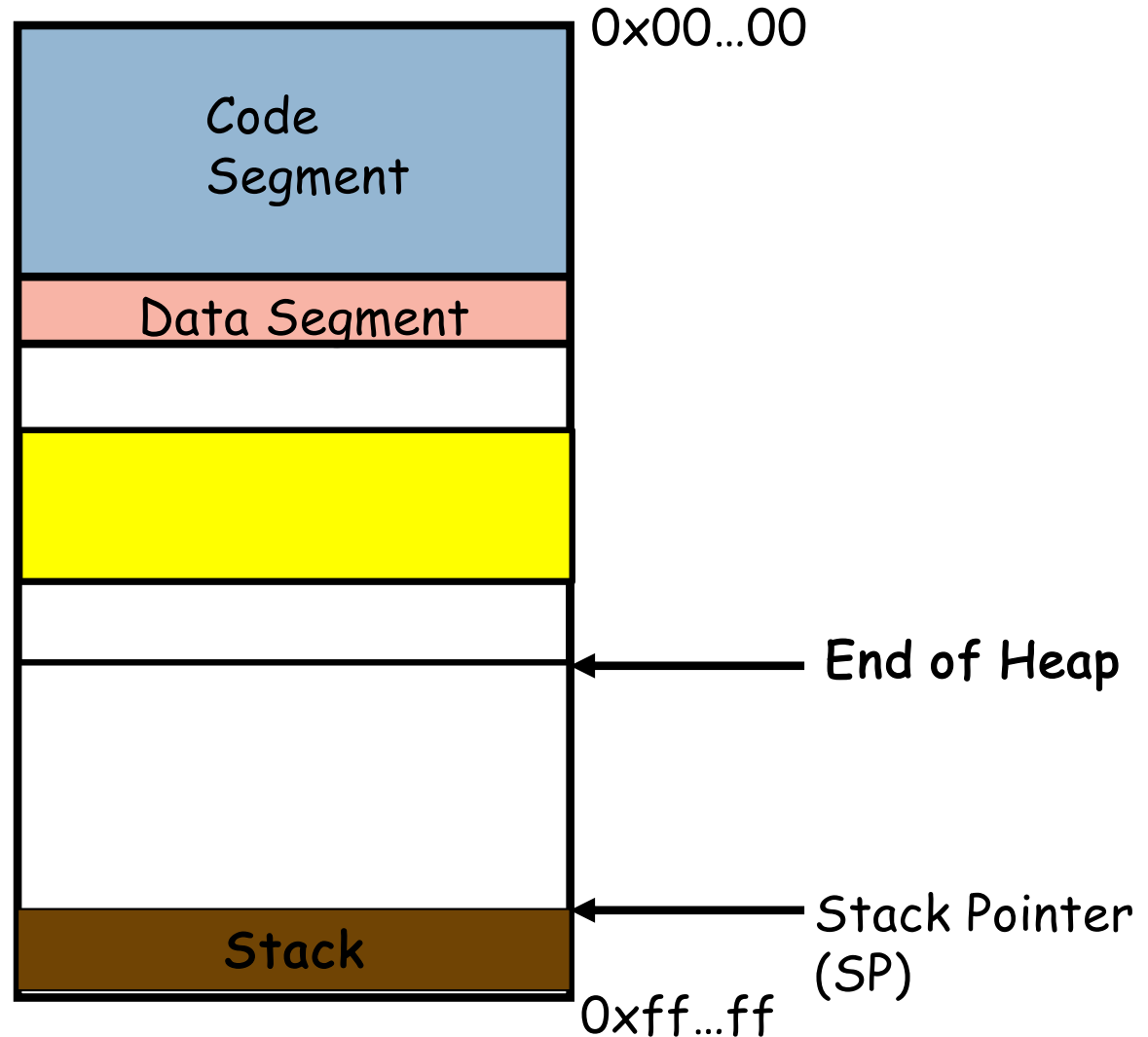


Malloc()

Malloc(150)

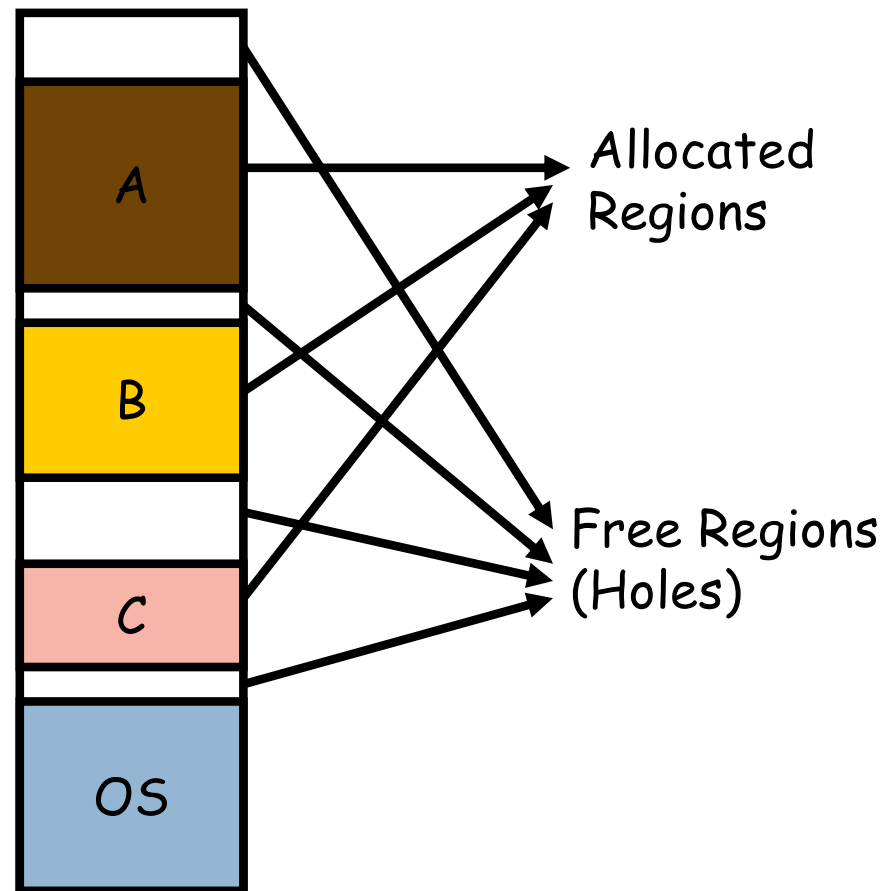
What should we do?

- Extend heap again (syscall overheads)
- Compact (costly)



Makes allocation of free space to malloc an important problem

The Memory Allocation Problem

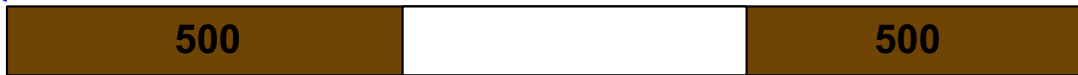
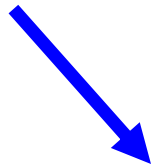


Question: How do we perform these allocations?

Goals

- Program wants allocation to be contiguous. Note
“p = malloc(1000)” requires 1000 bytes to be contiguous for

p[842] to work



- Allocation() and Free() should be fairly efficient
 - ▣ In time and space

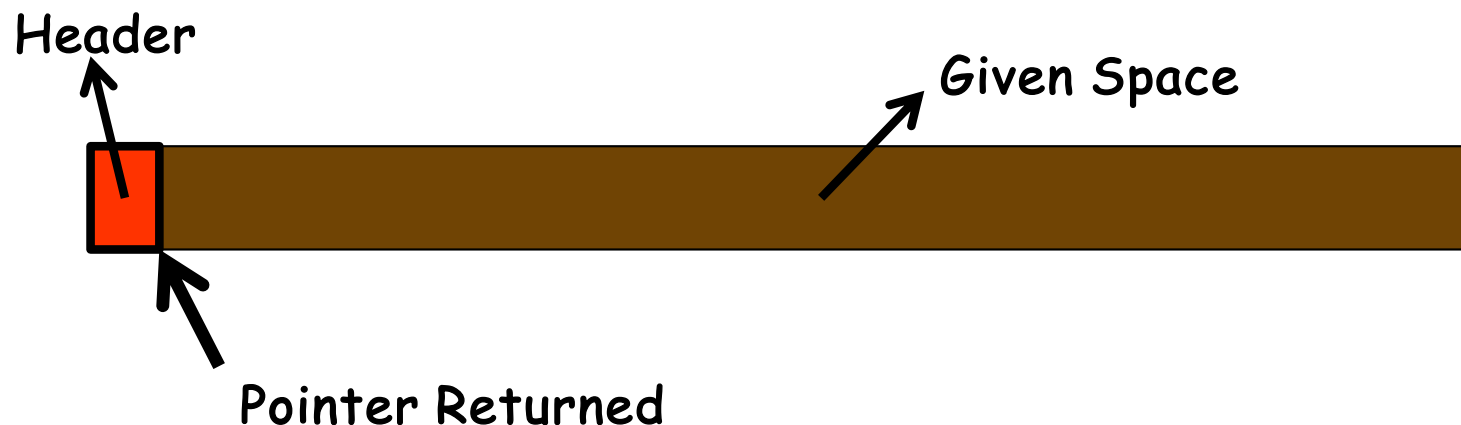
What do you do for a free(p)?


- You need to determine whether nearby regions (on either side) is free, and if so you need to make a larger hole.
- To do this:
 - ▣ We need to know what size you are free-ing
 - ▣ We need to know whether either side of this is free



How do you know what size you are freeing?

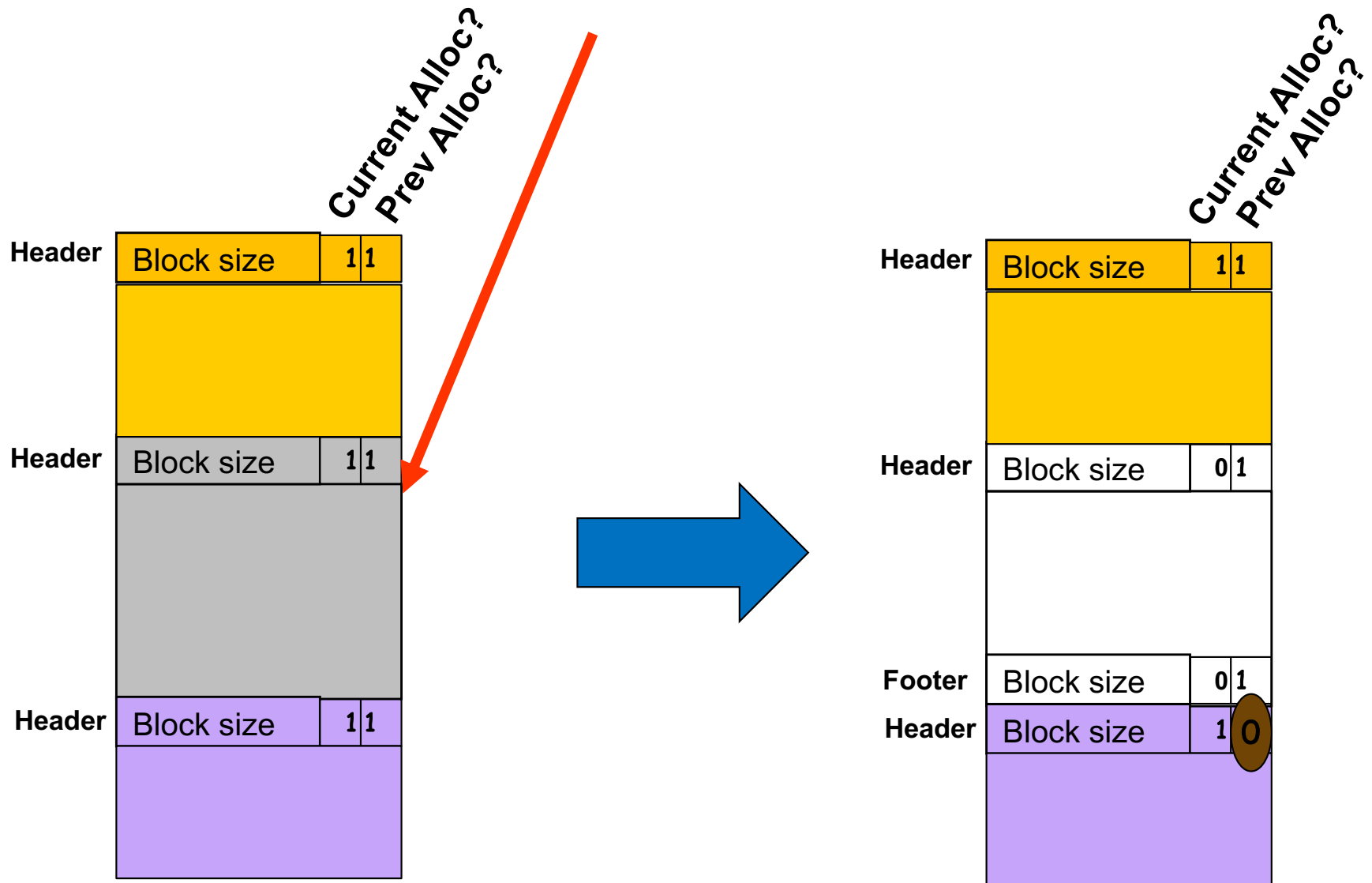
- Allocate extra space for a small header and return pointer after this header for an allocation.
- Do a negative offset of the pointer passed by free to find size!



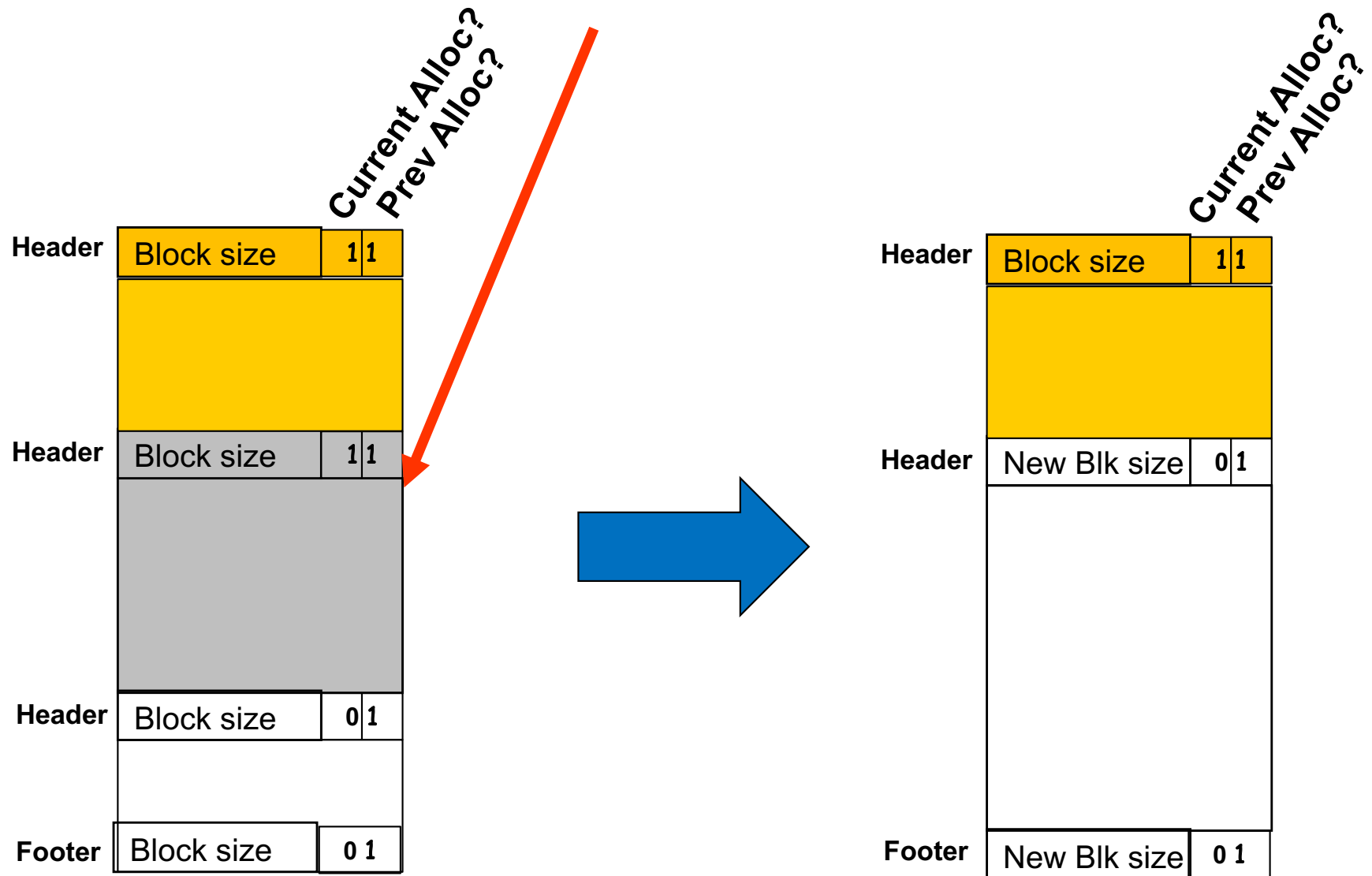


You know whether either side of what you are freeing is free, and if so, merge with them to create a larger hole

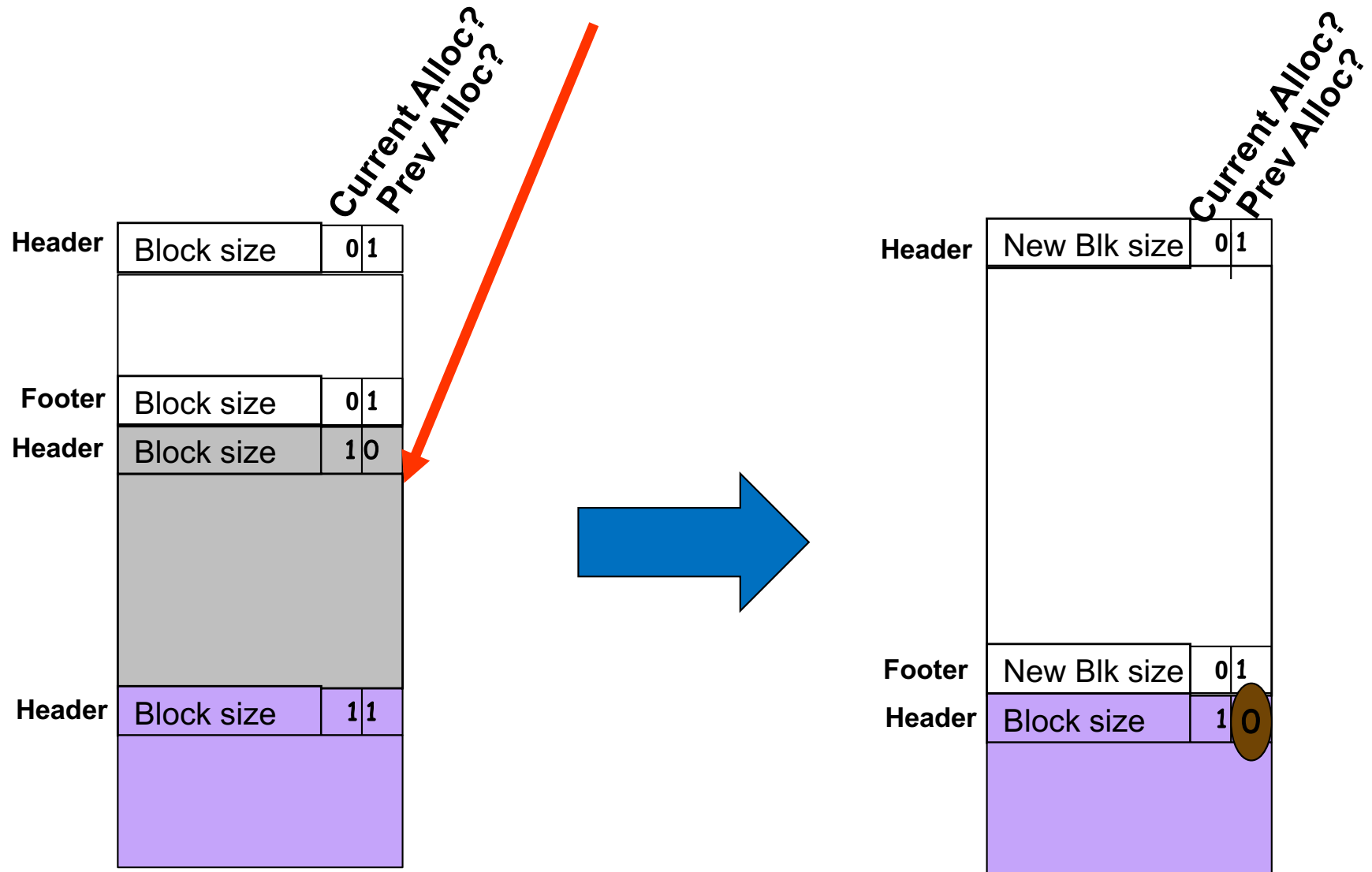
free(p) – Case 1



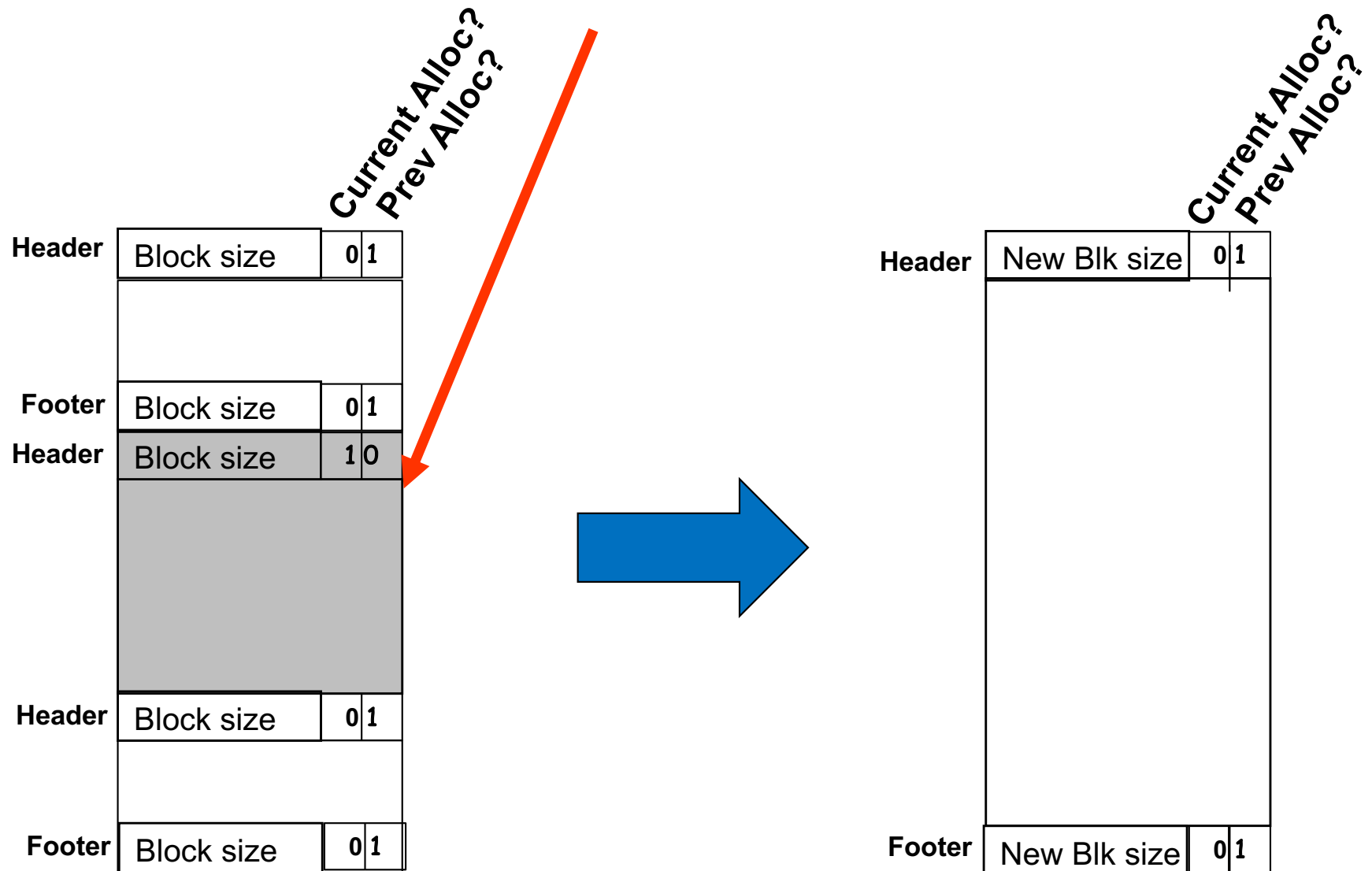
free(p) – Case 2



free(p) – Case 3



free(p) – Case 4



Costs



- Free: Look only at neighbors ($O(1)$)
- Allocation: Search through list ($O(N)$).

More to Memory Allocation

- Where to Allocate? – Separate Policy from Mechanism
 - ▣ Which free space to use?
 - ▣ **Allocation policies** (e.g., small-hole first, largest-hole first)
- Simplify Allocation
 - ▣ **Slab allocation**
 - Allocate space to allocate many commonly used objects
 - Same size holes
- Prevent Attacks
 - ▣ **Type-based memory reuse**
 - Only allocate objects of one type in each memory space
 - Avoid exploitation of **dangling pointers** – i.e., active pointer to freed memory

Conclusions

43

- Today was a review of memory management
- Memory is a limited hardware resource
 - ▣ Virtual memory enables the OS to share this memory among multiple processes
 - Physical memory is divided into fixed-sized frames
 - Virtual memory decided which pages are stored in each frame
- But, programmers want to use memory contiguously
 - ▣ Memory allocators manage memory in a contiguous, but expandable space, efficiently
- Next time
 - ▣ Research on kernel structures

Questions

44

