Memory

CPSC 457: Principles of Operating Systems
Winter 2024

Contains slides from Pavol Federl, Mea Wang, Andrew Tanenbaum and Herbert Bos, Silberschatz, Galvin and Gagne

Jonathan Hudson, Ph.D.
Instructor
Department of Computer Science
University of Calgary

Tuesday, 28 November 2024

Copyright © 2024

Topics

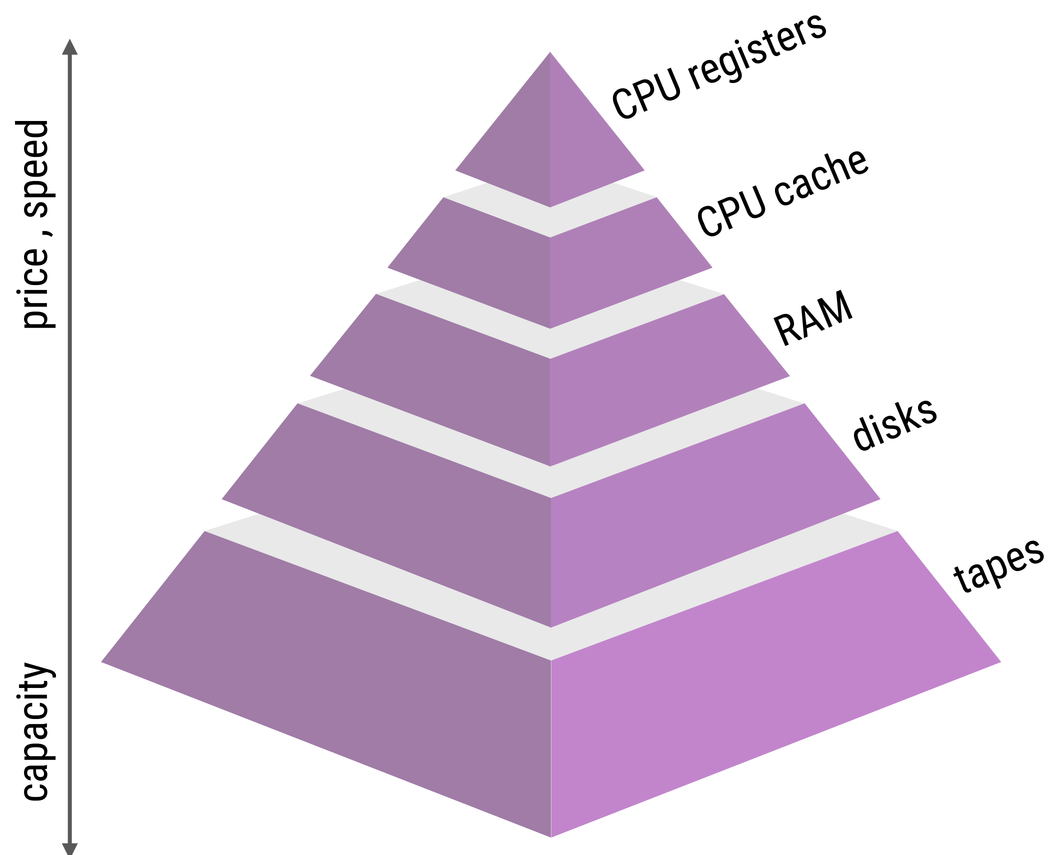
- "Programs expand to fill the memory available to them.”

- memory management
  - fixed partitioning
  - dynamic partitioning
  - placement algorithms

UNIVERSITY OF CALGARY

Memory

Memory



- programs need memory to run
- we expect computers to run multiple programs simultaneously
- we expect OS to manage memory for processes

UNIVERSITY OF CALGARY

Common memory management issues

- OS must give each process some portion of available memory (**address space**)
- if a program requests more memory later, e.g. by calling `malloc()` or `new`, how does that work?
- how does OS manage memory?
- how does `malloc()` and `new` work?

UNIVERSITY OF CALGARY

Memory allocation

UNIVERSITY OF CALGARY

Memory allocation

- at some point OS/malloc needs to
  - locate an unused area of memory
  - assign it to a process and mark it as used
  - eventually, when process quits or frees the memory, OS needs to mark it as unused again

- simple approaches can lead to memory fragmentation
  - lots of tiny unused chunks of memory,
    but none of them big enough to satisfy any requests

- OS needs to manage the memory in an efficient way
  - fast (searching, allocating, freeing, …)
  - minimize fragmentation

UNIVERSITY OF CALGARY

Memory allocation

- two general approaches: fixed partitioning and dynamic partitioning
- OS uses fixed partitioning to manage memory and to give it to processes
  - processes can request more memory from kernel using `brk()` or `mmap()` system calls
- each process then manages its own memory using dynamic partitioning, e.g. `malloc` & `new`
- `malloc` & `new` are not system calls, they are convenience functions that work in user mode
  - only if they run out of space, they ask kernel to allocate more space to the process using `brk()` or `mmap()`

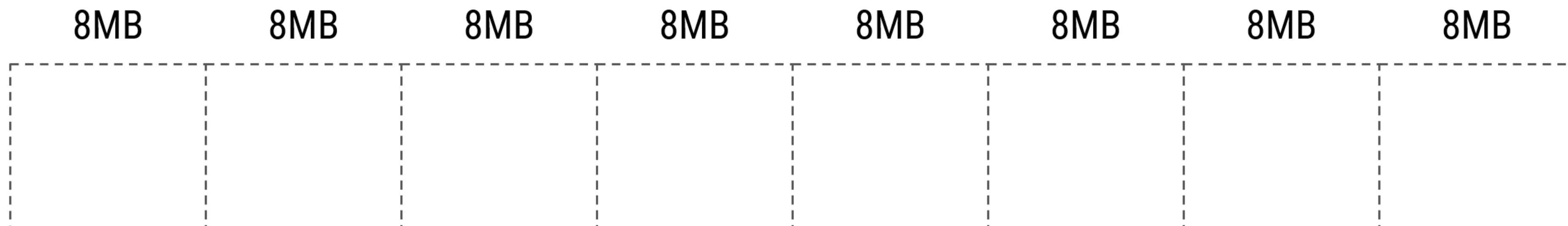
UNIVERSITY OF CALGARY

Fixed partitioning

UNIVERSITY OF CALGARY

Fixed partitioning

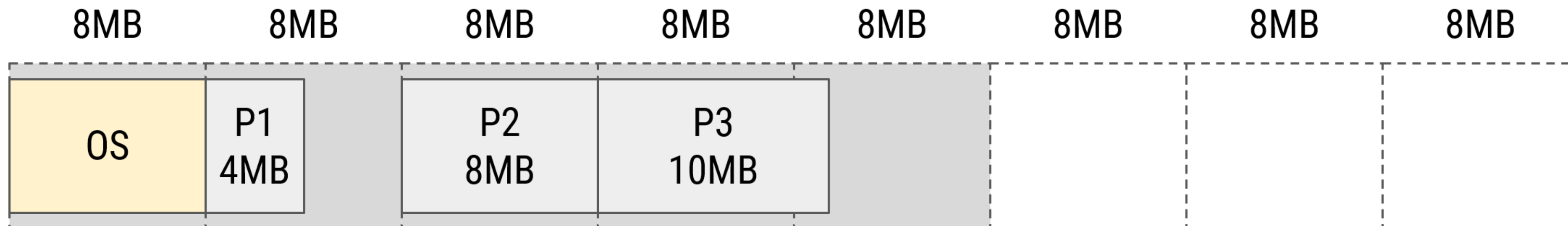
- memory is divided into partitions of equal size
- example:
  - total memory = 64MB, partition size = 8MB → 8 partitions
  - OS usually reserves some memory for itself, e.g. 1st partition
  - let's load 3 processes: P1 (4MB), P2 (8MB), P3 (10MB)



UNIVERSITY OF CALGARY

Fixed partitioning

- memory is divided into equal-sized partitions
- example:
  - total memory = 64MB, partition size = 8MB → 8 partitions
  - OS usually reserves some memory for itself (e.g. 1 partition, or 8MB)
  - let's load 3 processes: P1 (4MB), P2 (8MB), P3 (10MB)

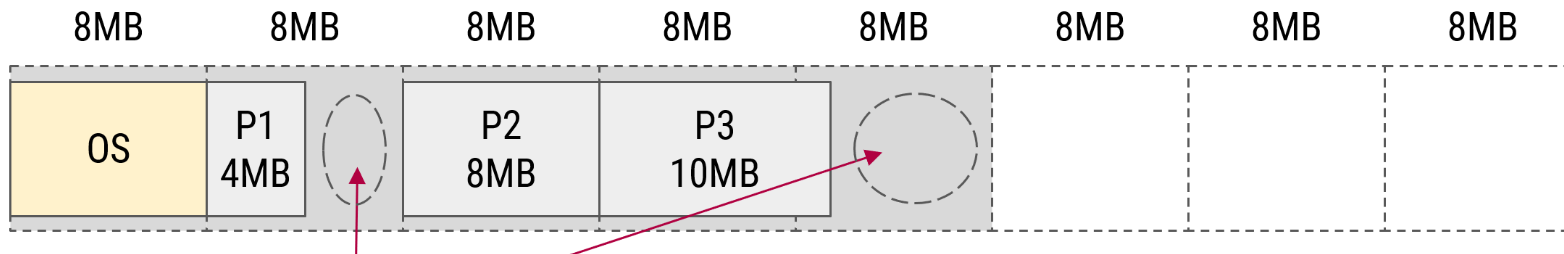


- Problems?

UNIVERSITY OF CALGARY

Fixed partitioning

- memory is divided into equal-sized partitions
- example:
  - total memory = 64MB, partition size = 8MB → 8 partitions
  - OS usually reserves some memory for itself (e.g. 1 partition, or 8MB)
  - let's load 3 processes: P1 (4MB), P2 (8MB), P3 (10MB)



- **internal fragmentation** – memory internal to a

  partition becomes fragmented
- leads to low memory utilization if partitions are big

Actual free memory: 34 MB

Usable free memory: 24 MB 😟

Dynamic partitioning

UNIVERSITY OF CALGARY

Dynamic partitioning

- create partitions that can fit requests perfectly
- example:
  - total memory = 64MB, minus 8MB taken by OS
  - load 3 processes: P1 (4MB), P2 (8MB), P3 (10MB)

UNIVERSITY OF CALGARY

Dynamic partitioning

- create partitions that can fit a request perfectly
- example:
  - total memory = 64MB, minus 8MB taken by OS
  - load 3 processes: P1 (4MB), P2 (8MB), P3 (10MB)

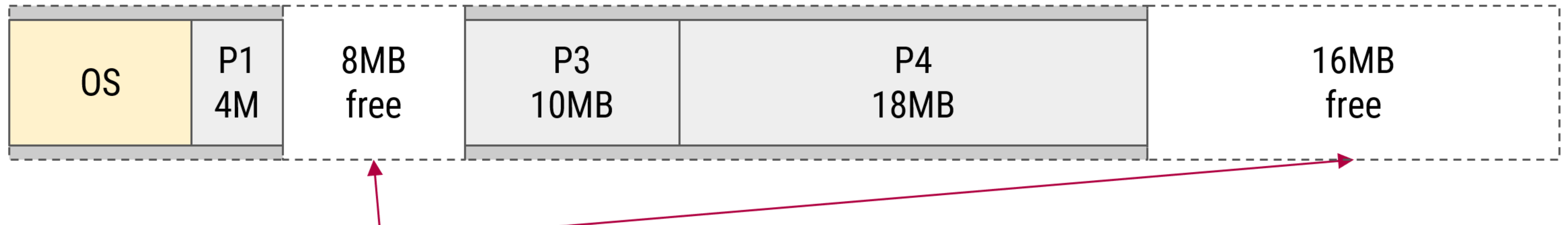
- no more internal fragmentation
- but what if P2 finishes, and P4 (18MB) gets added?

Usable free memory: 34 MB

Actual free memory: 34 MB 😀

Dynamic partitioning

- P2 finishes and P4 (18MB) gets added:



- **external fragmentation**: the memory that is external to all partitions becomes increasingly fragmented, leading to low memory utilization
- e.g. P5 (17MB) could not start, despite having enough free RAM

Usable free memory: 24 MB

Actual free memory: 24 MB

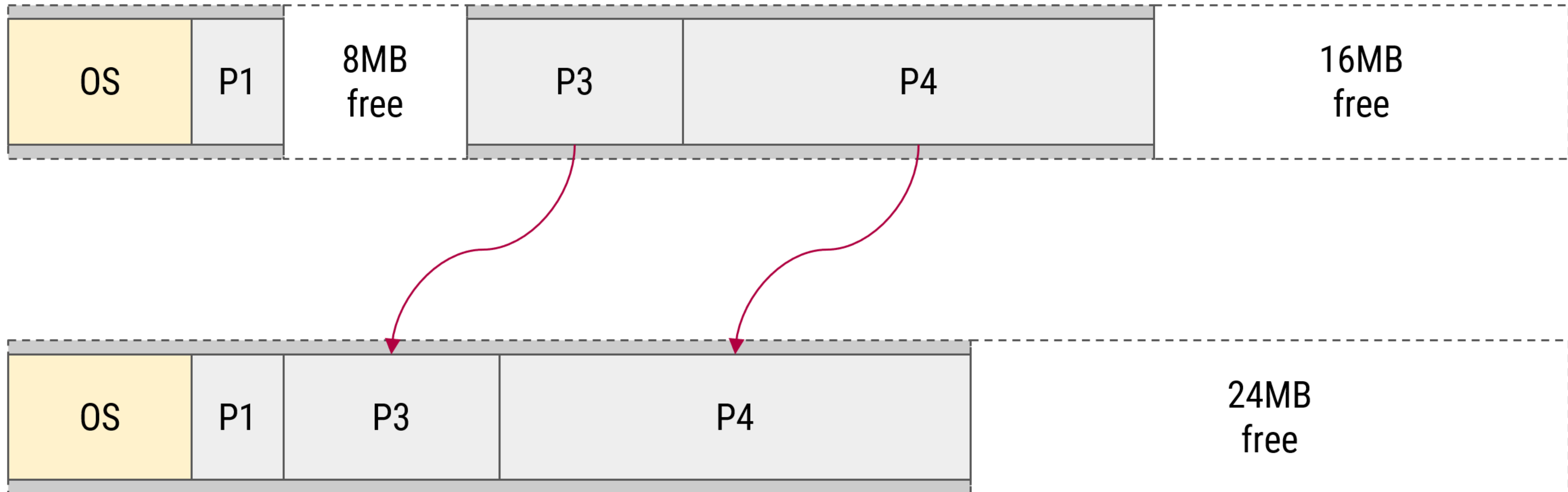
Largest free chunk: 16 MB 😳

Memory compaction

UNIVERSITY OF CALGARY

Memory compaction

- memory compaction is a mechanism that reduces external fragmentation
- from time to time, the OS re-arranges the used blocks of memory so that they are contiguous
- goal: free blocks are merged into a single large free block
- CPU intensive operation, not used*

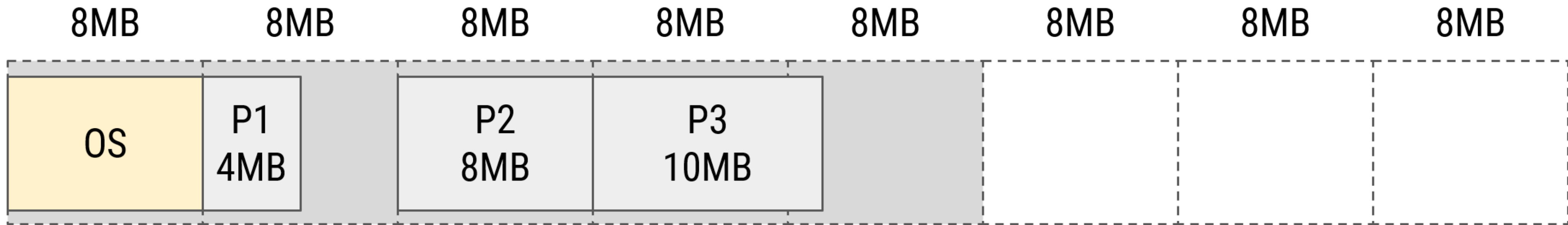


UNIVERSITY OF CALGARY

Implementing memory allocation algorithms

UNIVERSITY OF CALGARY

Implementing memory allocation algorithms

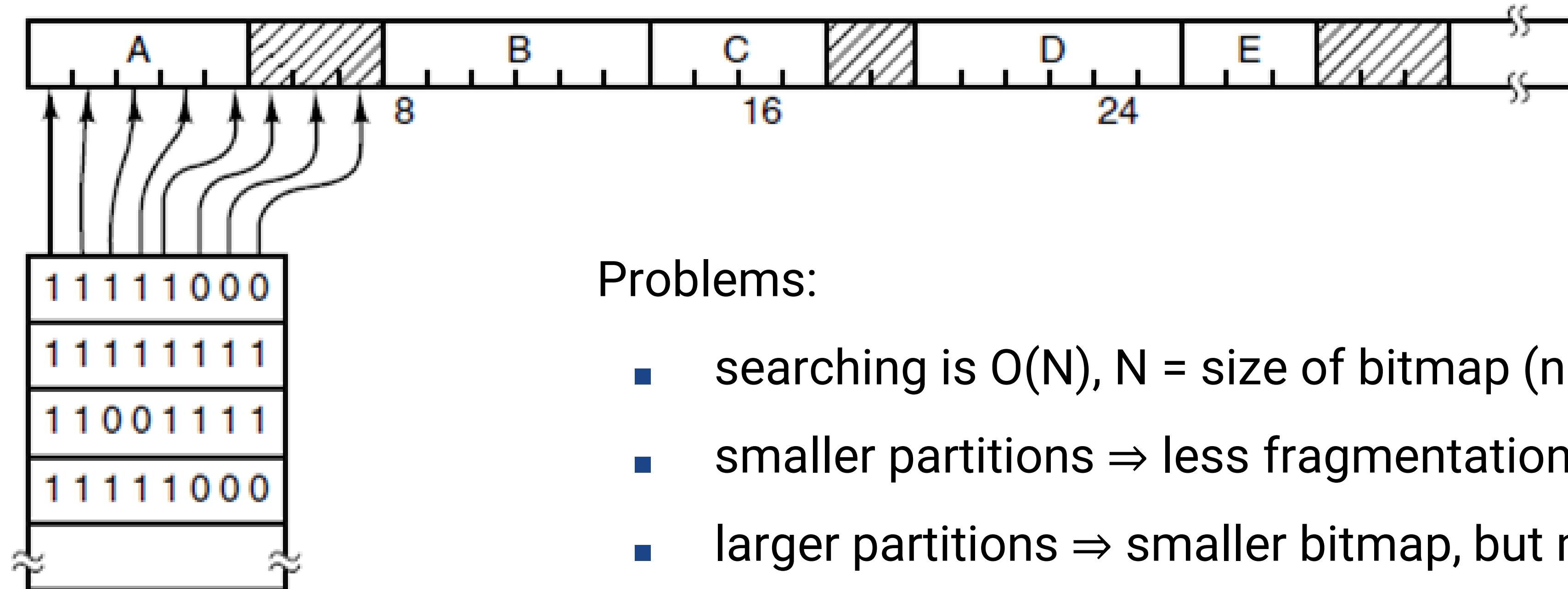


- how do we keep track of free memory and allocated memory?
- we need a data structure
  - that is efficient at searching free space
  - that is efficient at reclaiming free space
  - "deals" with fragmentation
- for fixed partitions we can use bitmaps
- for dynamic partitions we can use linked lists + trees

UNIVERSITY OF CALGARY

Bitmaps for fixed partitions

- memory is divided into equal partitions as small as few words and as large as several KB
- OS maintains a bitmap, 1 bit per partition, where 0=free, 1=occupied

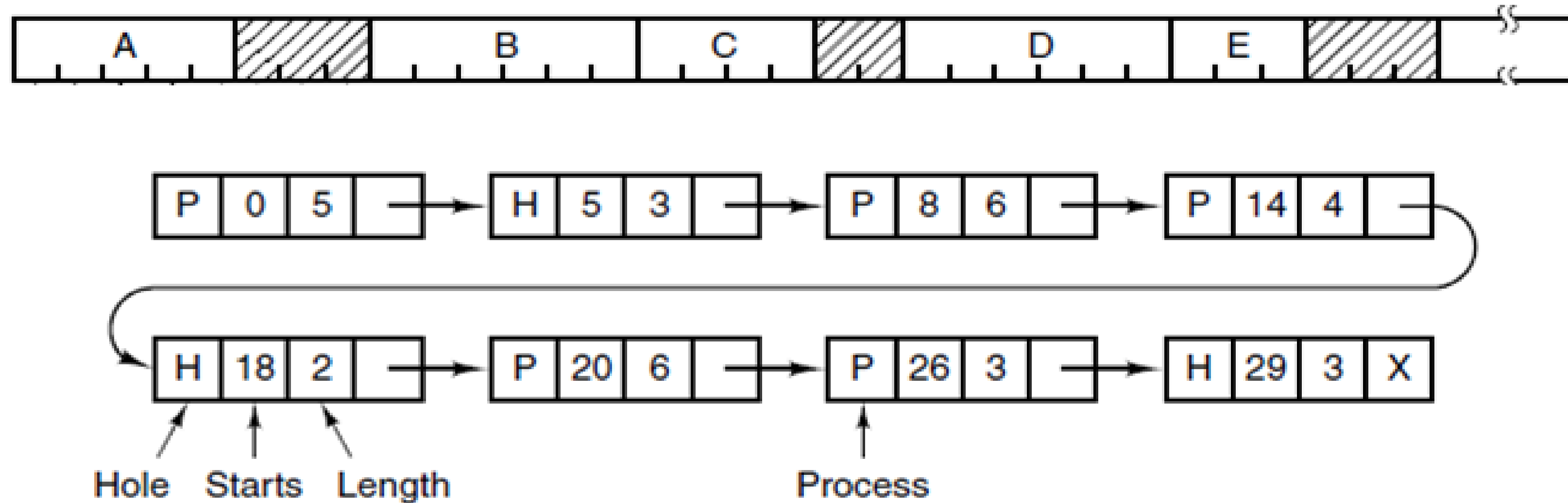


Problems:

- searching is O(N), N = size of bitmap (number of partitions)
- smaller partitions ⇒ less fragmentation, but larger bitmap
- larger partitions ⇒ smaller bitmap, but more fragmentation
- compromise between efficiency and fragmentation
- note: larger bitmap also implies more wasted memory

Linked lists for dynamic partitioning

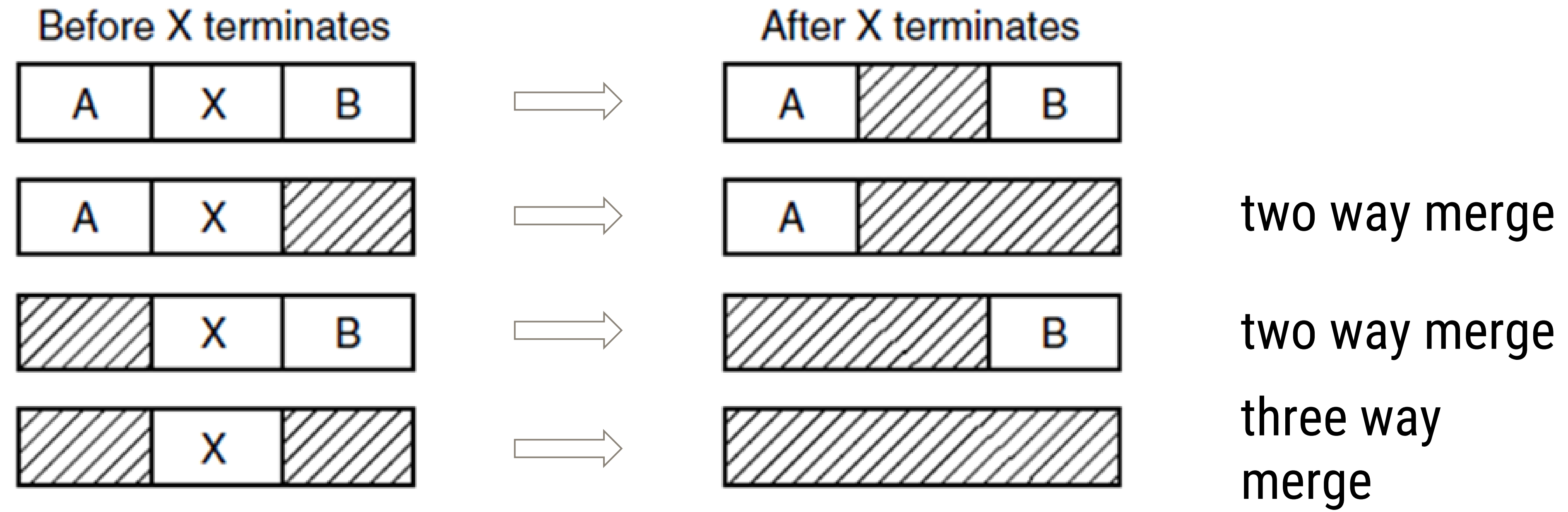
- memory is divided into partitions of dynamic size
- OS maintains a list of allocated and free memory partitions (**holes**), sorted by address



- searching for empty partitions in plain linked list is O(N), where N = number of partitions
  - we can use balanced trees to improve search for empty partitions ⟶ O(log n)
- reclaiming free space can be O(1)
  - using doubly-linked lists and keeping linked-list pointers within partitions

UNIVERSITY OF CALGARY

Memory management with linked lists - reclaiming space



UNIVERSITY OF CALGARY

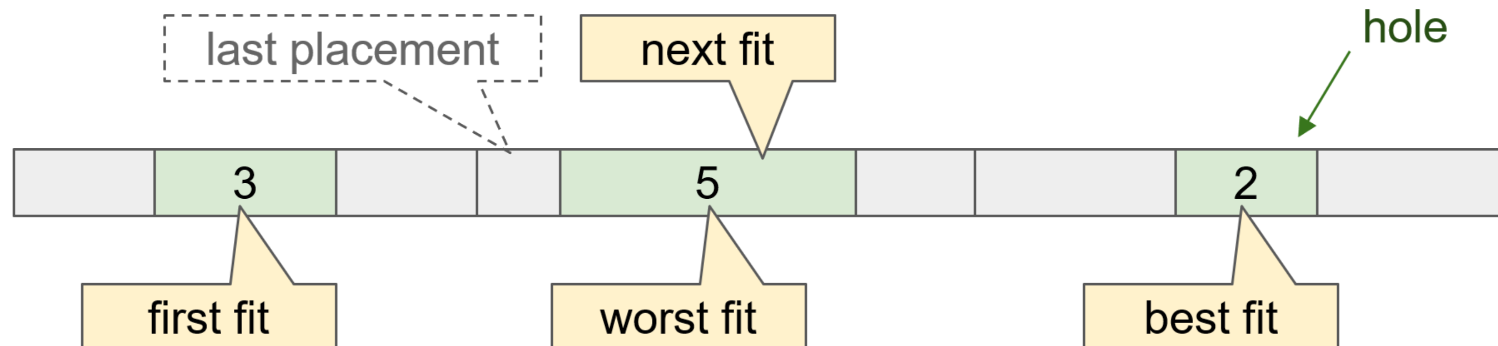
Finding Free Space

UNIVERSITY OF CALGARY

Memory allocation

- basic algorithms for finding a free space (hole) in a linked list:
  - first fit - find the first hole that is big enough, leftover space becomes new hole
  - next fit - same as first fit, but start searching at the location of last placement
  - best fit - find the smallest hole that is big enough, leftover (tiny) space becomes new hole
  - worst fit - find the largest hole, leftover (big) space is likely to be usable
- many other more sophisticated techniques

  e.g. quick fit - maintain separate lists for common request sizes, leads to faster search, but more complicated management
- example: request is to allocate memory for 2 units



UNIVERSITY OF CALGARY

Memory allocation simulation

UNIVERSITY OF CALGARY

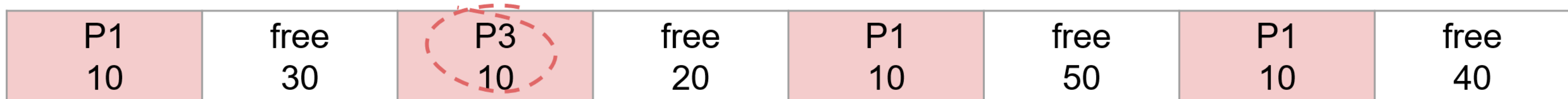
Memory allocation simulation (dynamic partitioning)

Operation								
`start with a free partition of size 180`	free 180							
`alloc(P1,10)`	P1 10	free 170						
`alloc(P2,30)`	P1 10	P2 30	free 140					
`alloc(P3,10)`	P1 10	P2 30	P3 10	free 130				
`alloc(P2,20)`	P1 10	P2 30	P3 10	P2 20	free 110			
`alloc(P1,10)`	P1 10	P2 30	P3 10	P2 20	P1 10	free 100		
`alloc(P2,50)`	P1 10	P2 30	P3 10	P2 20	P1 10	P2 50	free 50	
`alloc(P1,10)`	P1 10	P2 30	P3 10	P2 20	P1 10	P2 50	P1 10	free 40
`free(P2)`	P1 10	free 30	P3 10	free 20	P1 10	free 50	P1 10	free 40

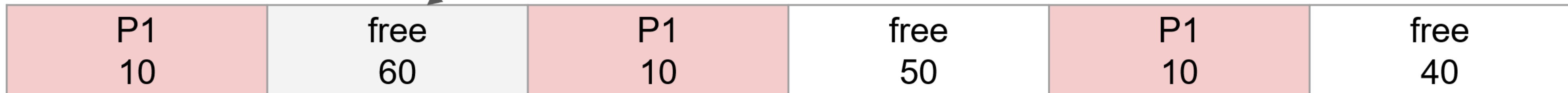
UNIVERSITY OF CALGARY

Memory allocation simulation (dynamic partitioning)

```
free(P3)
```

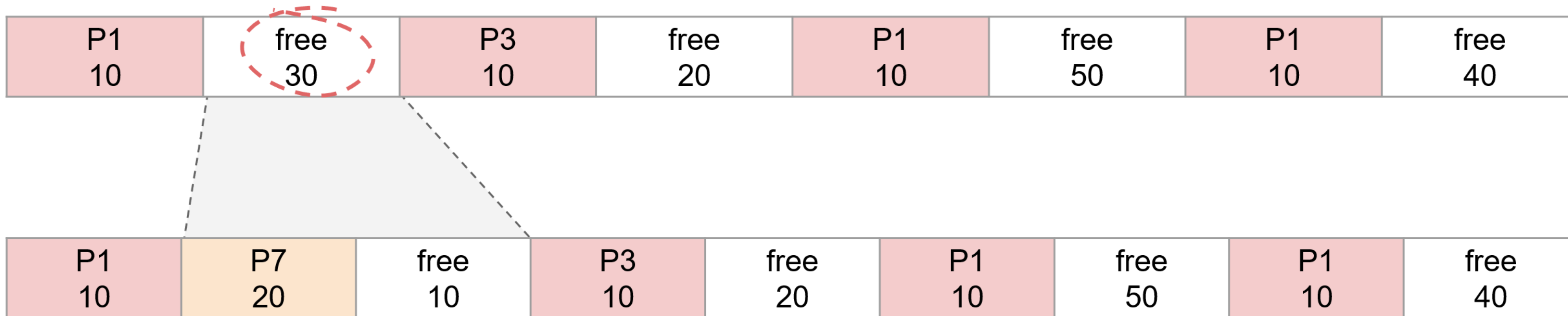


must merge adjacent free partitions into one

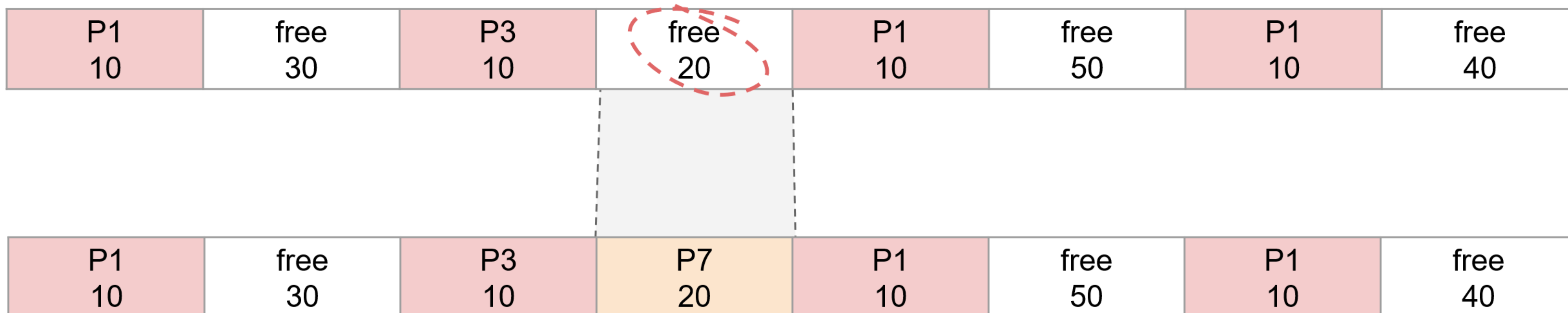


Memory allocation simulation (dynamic partitioning)

```
alloc(P7,20) # using first fit
```



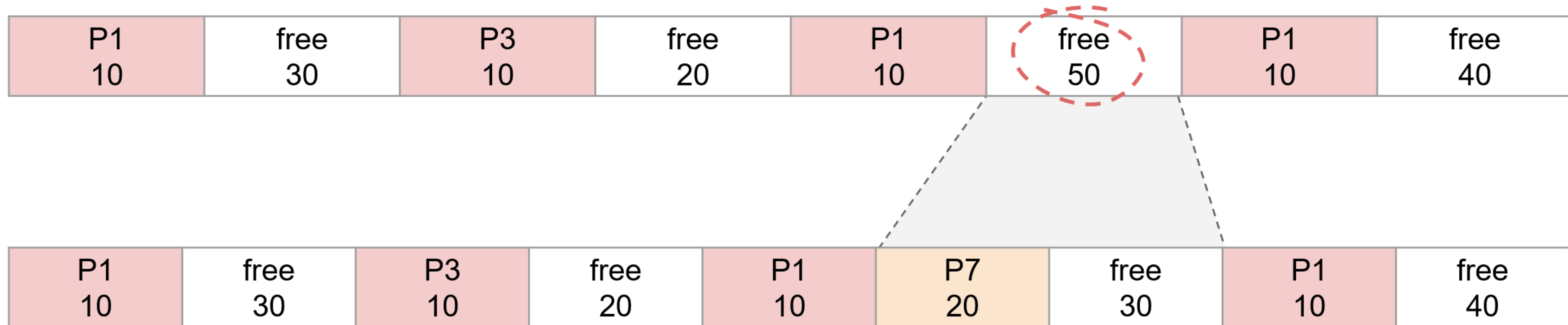

```
alloc(P7,20) # using best fit
```



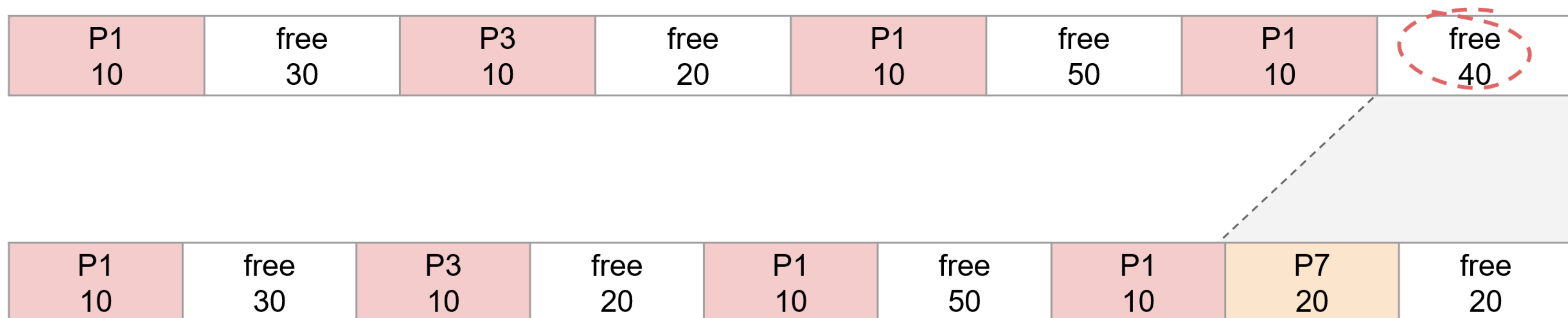
UNIVERSITY OF CALGARY

Memory allocation simulation (dynamic partitioning)

`alloc(P7,20) # using worst fit`



`alloc(P7,20) # using next fit`



UNIVERSITY OF CALGARY

Memory allocation simulation (dynamic partitioning)

- in order for memory allocation to be useful, we need to keep track of addresses of the memory that each partition represents, e.g. so that we know what to return by `malloc()`
- we can start with `address=0` for first partition
- each subsequent partition's address is previous partition's address + its size

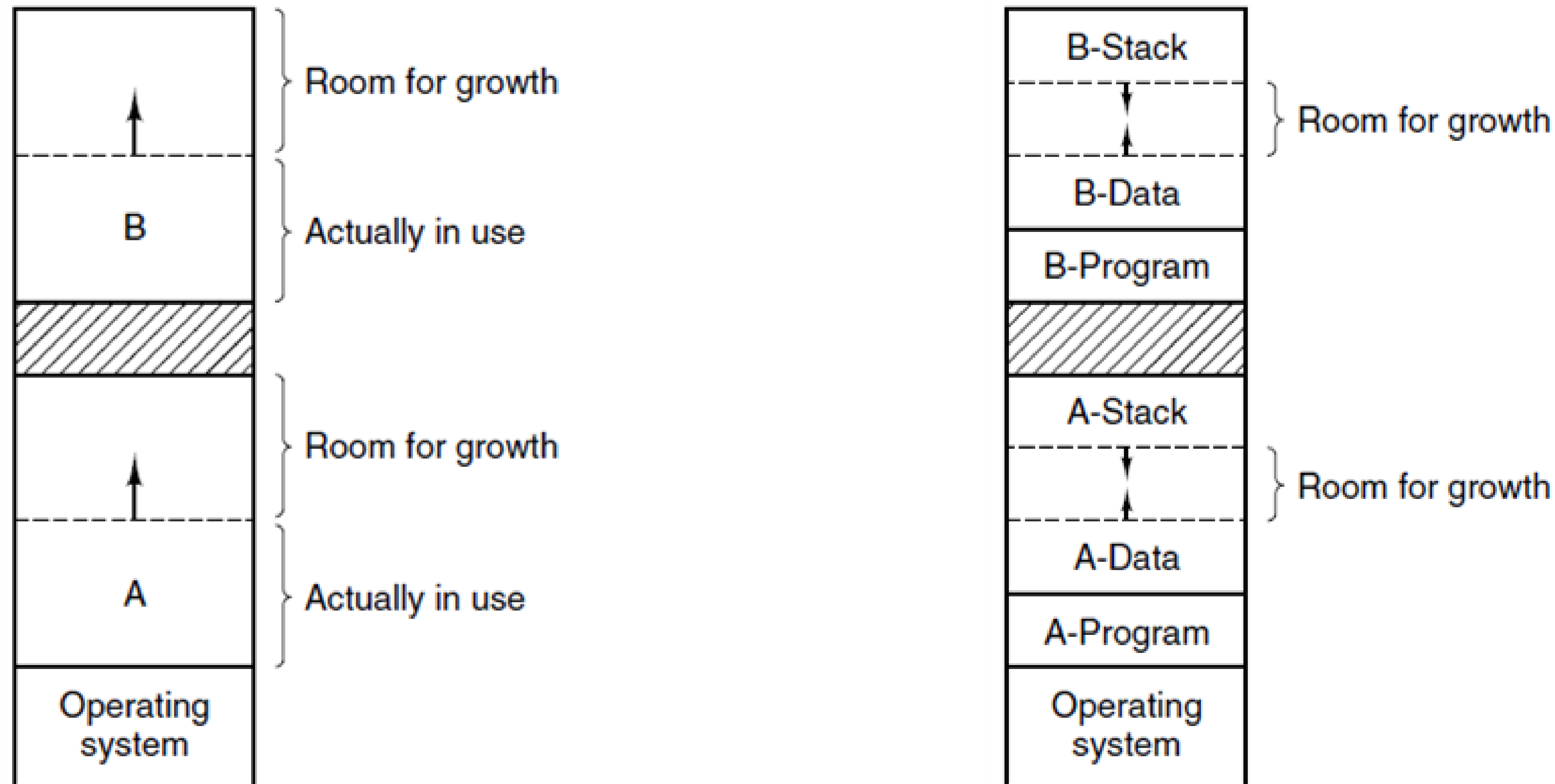
status: P1 size: 10 addr: 0	status: free size: 30 addr: 10	status: P3 size:10 addr: 40	status: free size:20 addr: 50	status: P1 size:10 addr: **70**	status: free size:50 addr: 80	status: P1 size:10 addr: 130	status: free size:40 addr: 140

Memory usage

UNIVERSITY OF CALGARY

Memory usage

- how much memory should OS allocate to each process?
- most programs *increase* their memory usage during execution
- possible solution is to find sufficiently large free memory chunk, and move process into it
- another improvement: OS can proactively allocates extra memory for each process



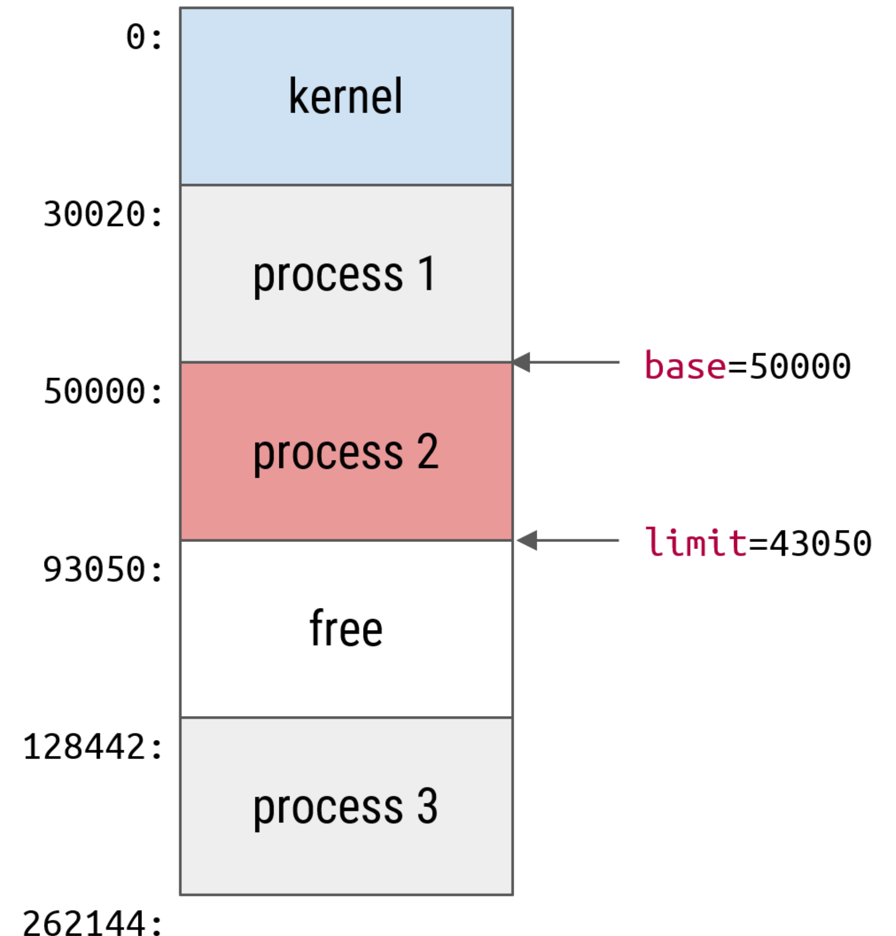
UNIVERSITY OF CALGARY

Address Protection

UNIVERSITY OF CALGARY

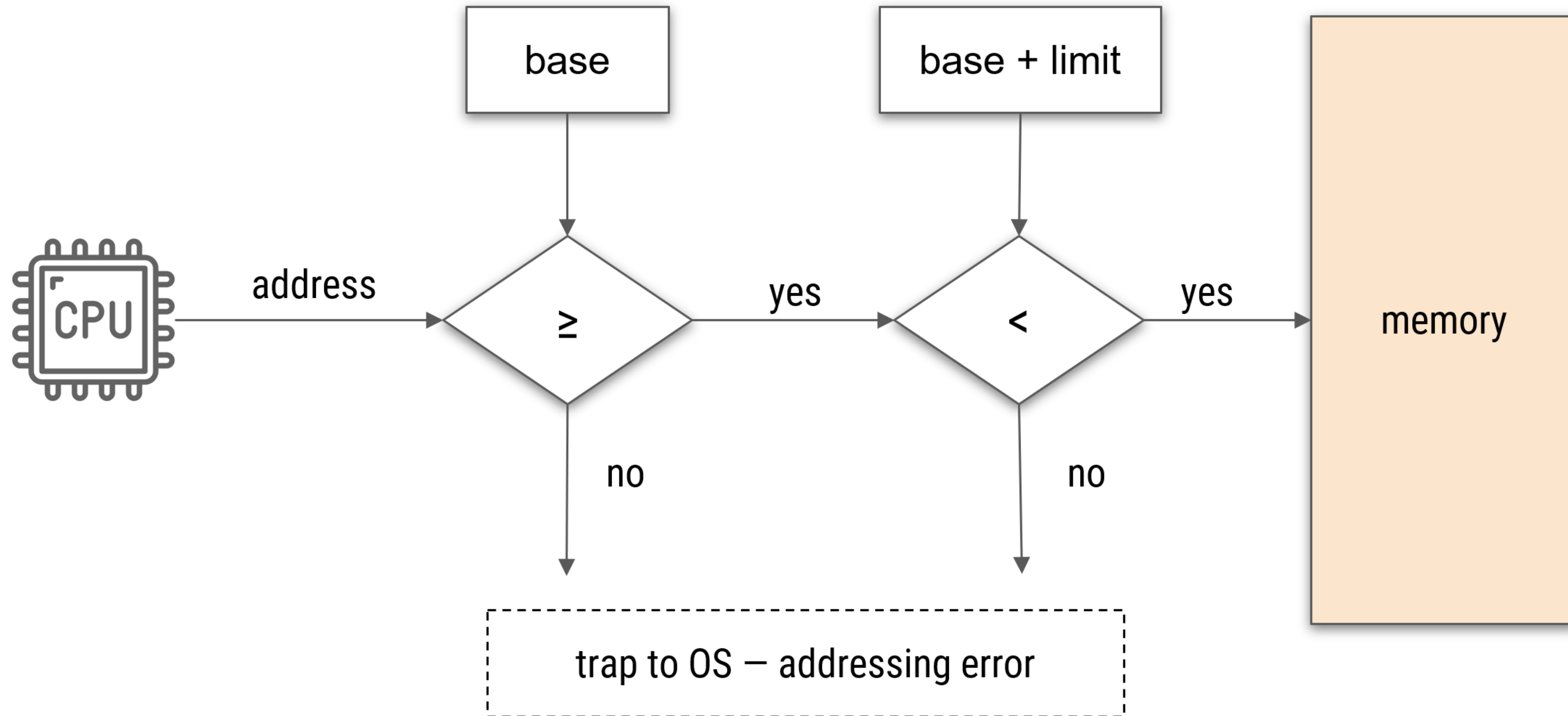
Base and Limit Registers — address protection in hardware

- we can add an extra pair of registers to CPU
- these can define the allowed range of addresses
  - base register = starting memory
  - limit register = size of memory
- the base and limit registers can be modified only in kernel mode
- CPU checks every memory access generated by a process
- when process tries to access invalid address CPU generates SW interrupt ⟶ trapped by kernel
- base & limit registers stored in PCB



UNIVERSITY OF CALGARY

Base and Limit Registers — address protection in hardware



UNIVERSITY OF CALGARY

Onward to … virtual memory

Jonathan Hudson
jwhudson@ucalgary.ca
https://pages.cpsc.ucalgary.ca/~jwhudson/

UNIVERSITY OF CALGARY