
Grad School Event
Connect with graduate student and learn about graduate school options + experiences!
Nov 16 6–7:30PM @ Gates 114
P.S. Come for desserts!
RSVP

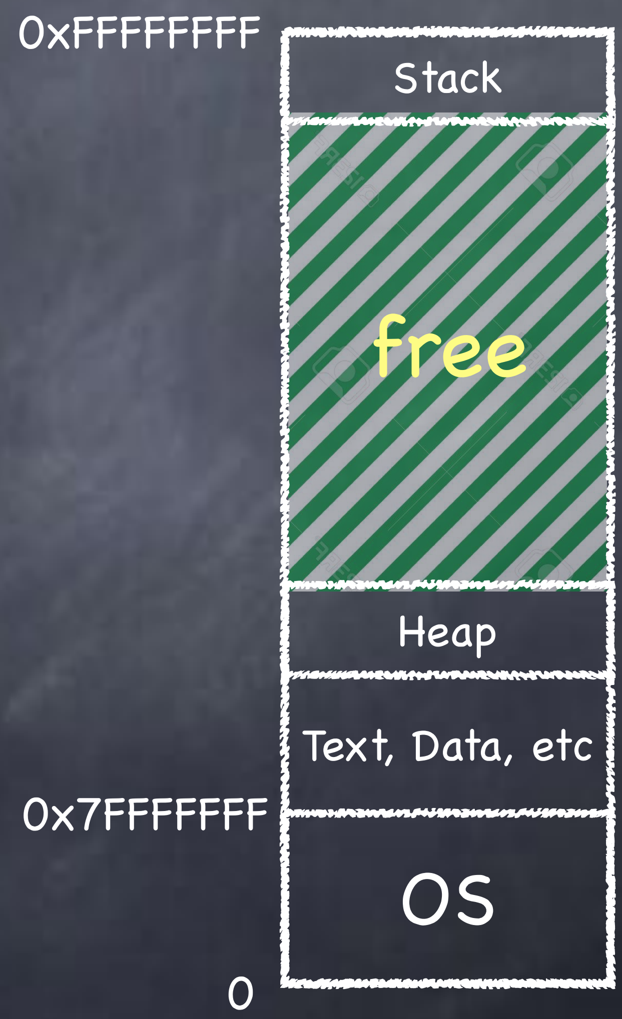
GO ACSU!

ANNOUNCEMENTS

- We have an alternate time for the prelim: 5:30pm. See Ed post #652 and your email for details.
- We will have a review session before the prelim on Tuesday, 11/15, from 6:00 pm to 8:00 pm, tentatively in Statler 196. Will confirm on Ed.
- Practice exams were made available. See Ed post #659
- The prelim will be comprehensive. See Ed post #409, #421, and #638
- Exam for students with SDS accommodations: November 17 at 6:45pm in Statler 198. Email to follow

The Identity Mapping

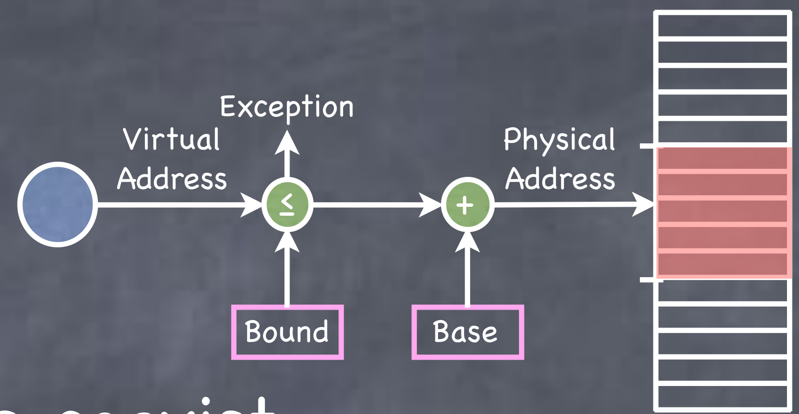
- Map each virtual address onto the identical physical address
  - Virtual and physical address spaces have the same size
  - Run a single program at a time
    - OS can be a simple library
    - very early computers
- Friendly amendment: leave some of the physical address space for the OS
  - Use loader to relocate process
    - early PCs



More sophisticated address translation

- How to perform the mapping efficiently?
  - So that it can be represented concisely?
  - So that it can be computed quickly?
  - So that it makes efficient use of the limited physical memory?
  - So that multiple processes coexist in physical memory while guaranteeing isolation?
  - So that it decouples the size of the virtual and physical addresses?
- Ask hardware for help!

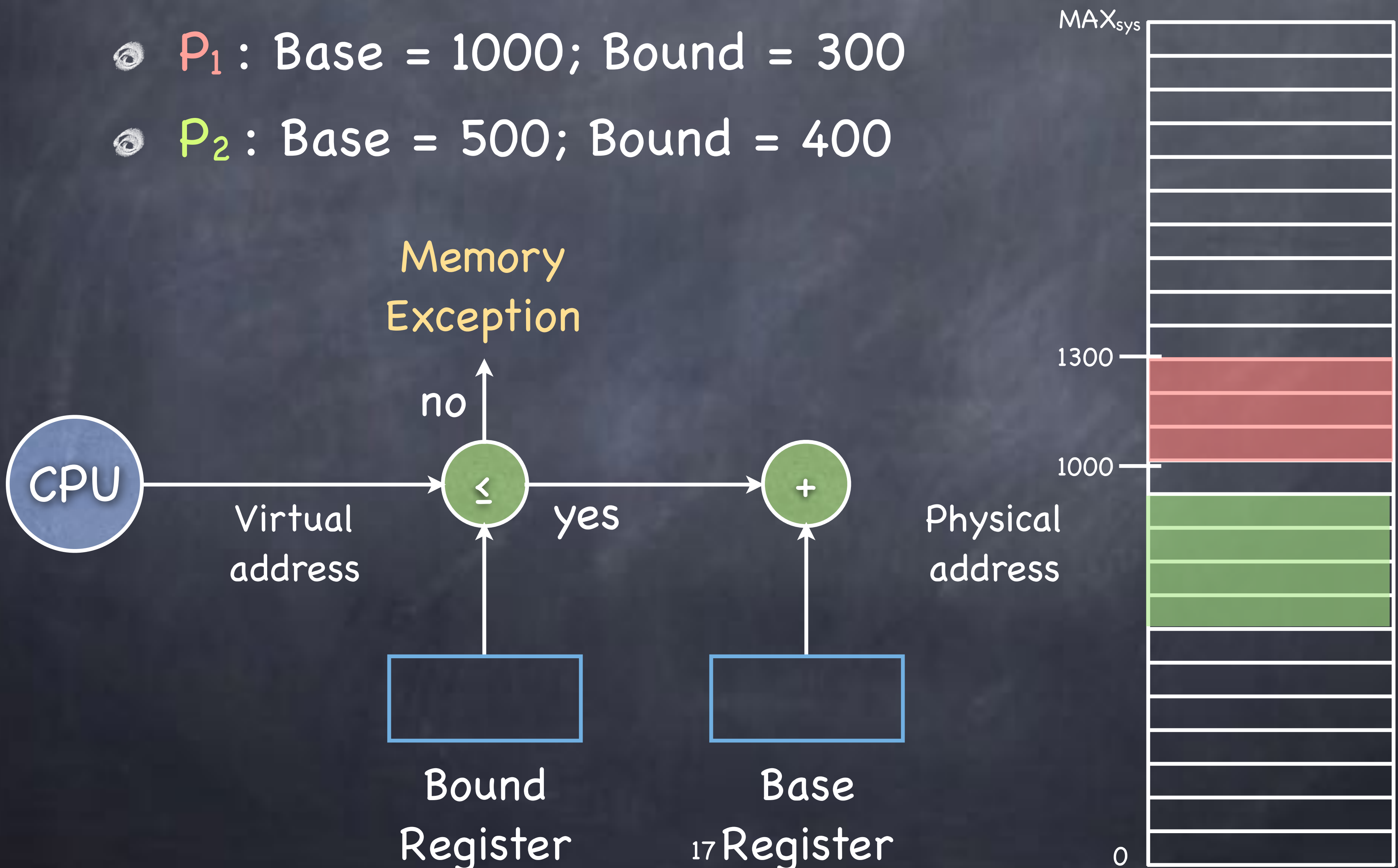
Base & Bound



- Goal: let multiple processes coexist in memory while guaranteeing isolation
- Needed hardware
  - two registers: Base and Bound (a.k.a. Limit)
  - Stored in the PCB
- Mapping
  - pa = va + Base
    - as long as $0 \leq va \leq$ Bound
  - On context switch, change B&B (privileged instruction)

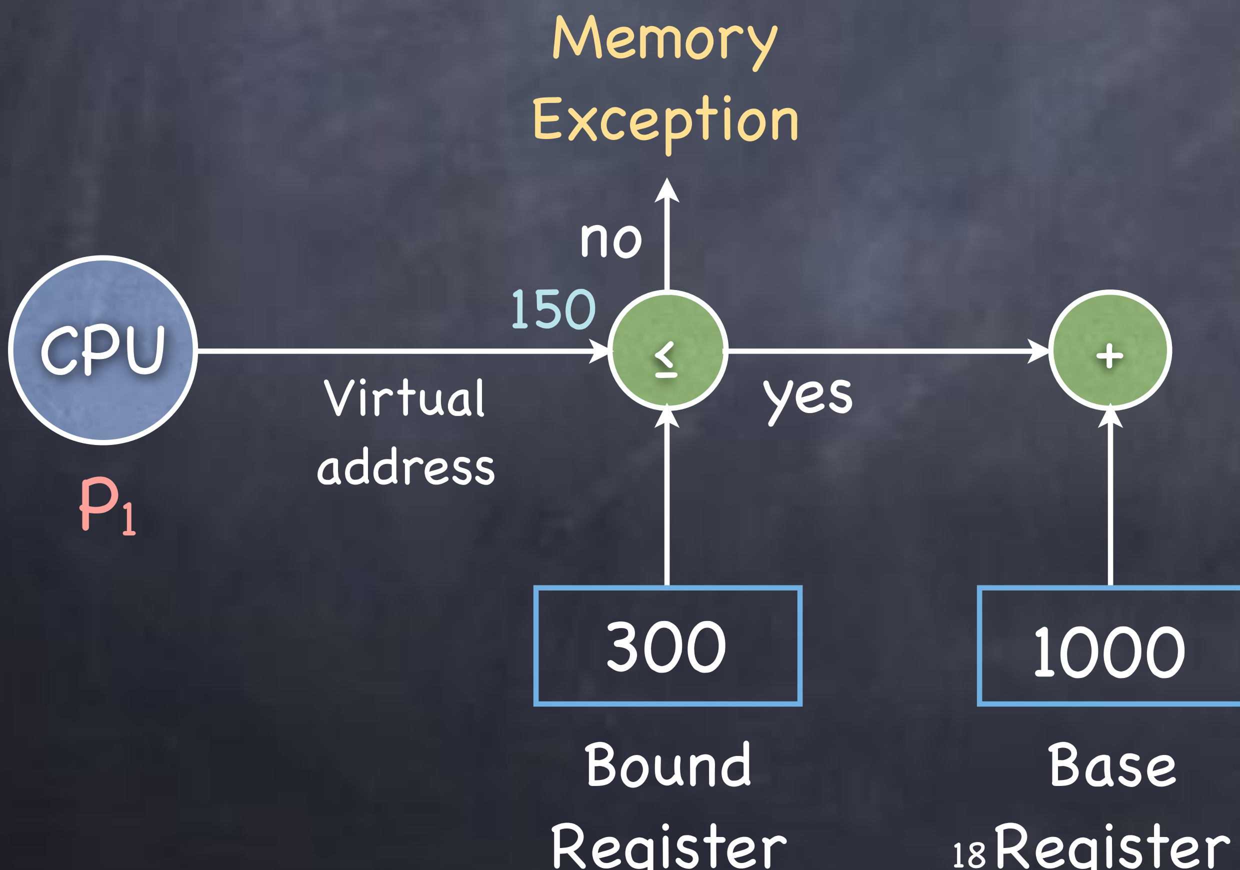
Base & Bound

- P_1 : Base = 1000; Bound = 300
- P_2 : Base = 500; Bound = 400



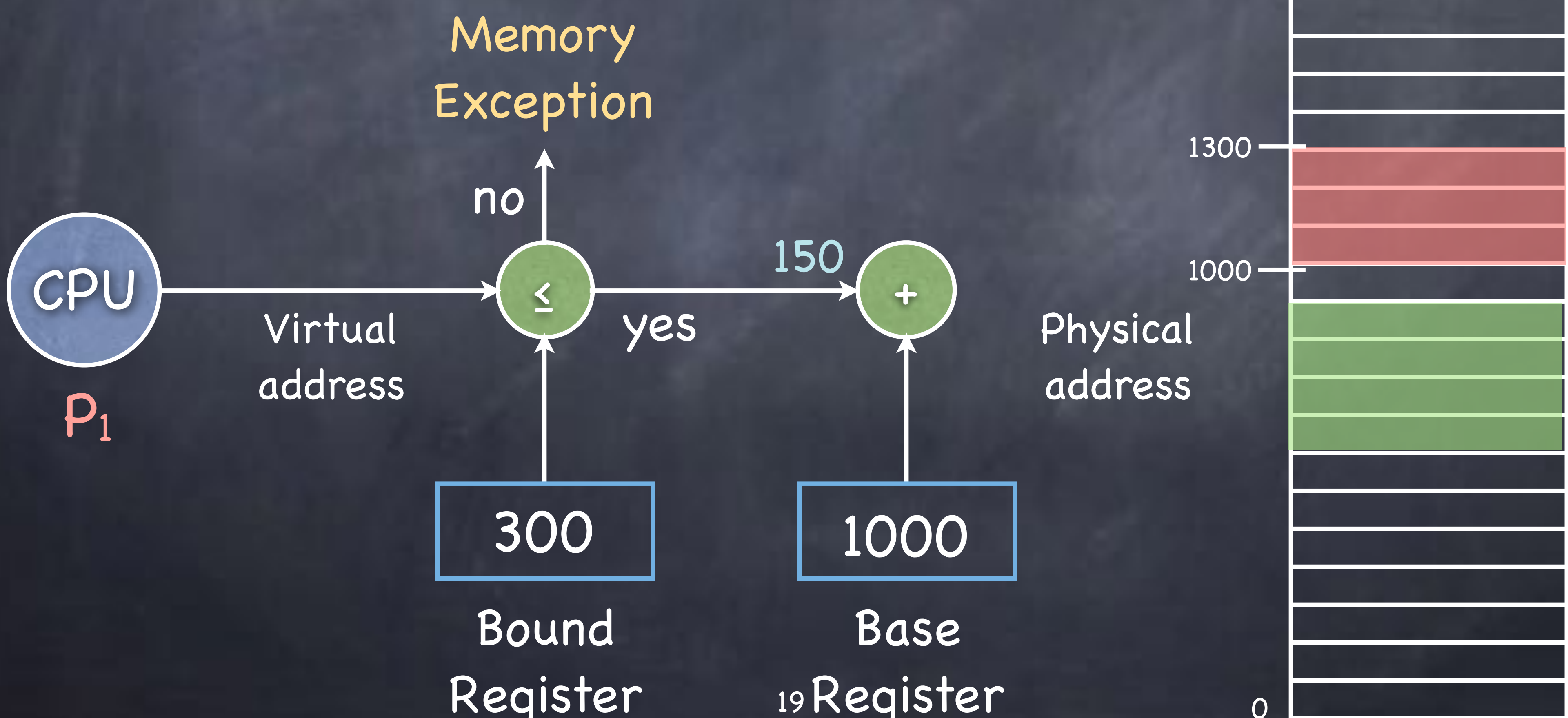
Base & Bound

- P_1 : Base = 1000; Bound = 300
- P_2 : Base = 500; Bound = 400



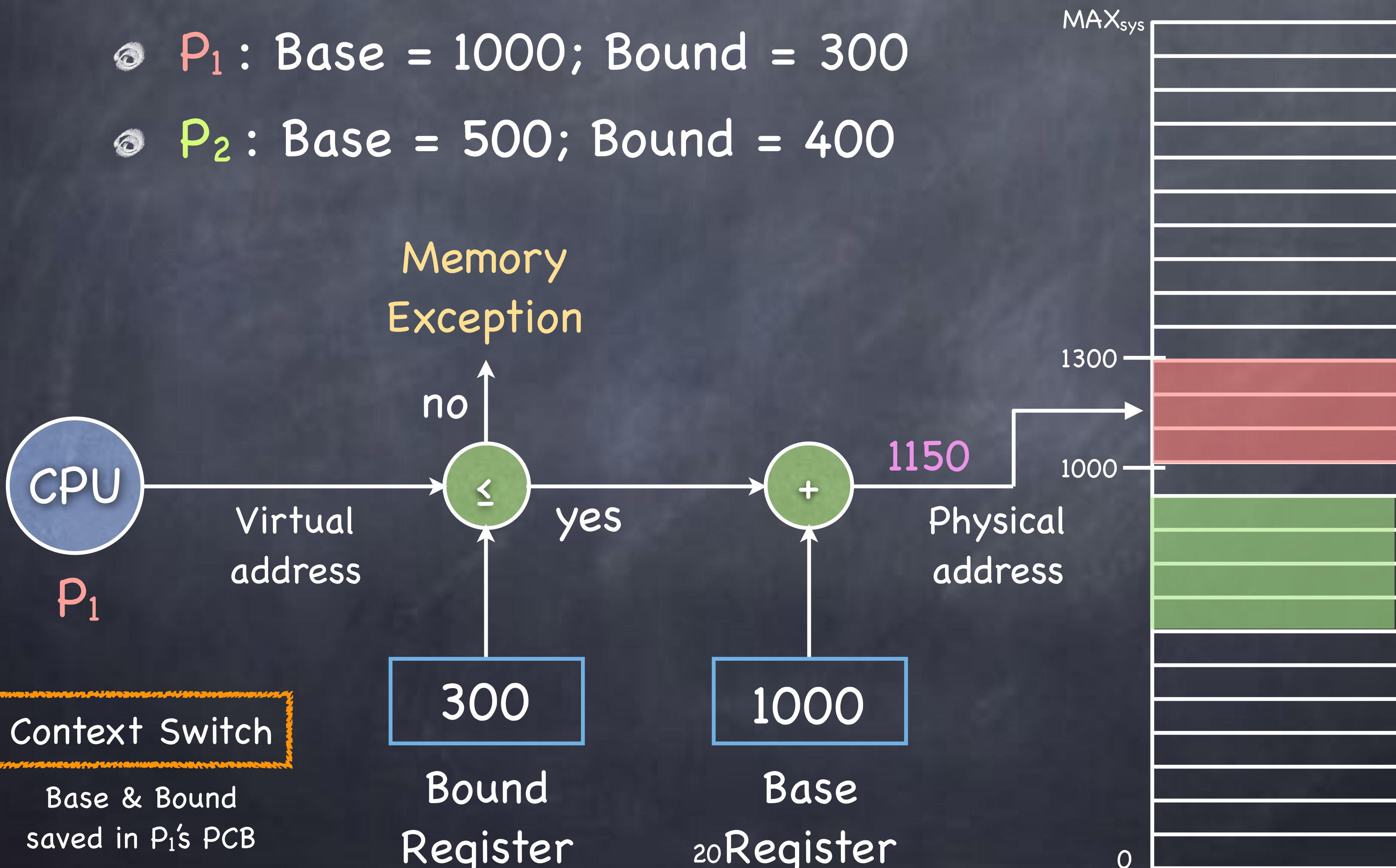
Base & Bound

- P_1 : Base = 1000; Bound = 300
- P_2 : Base = 500; Bound = 400



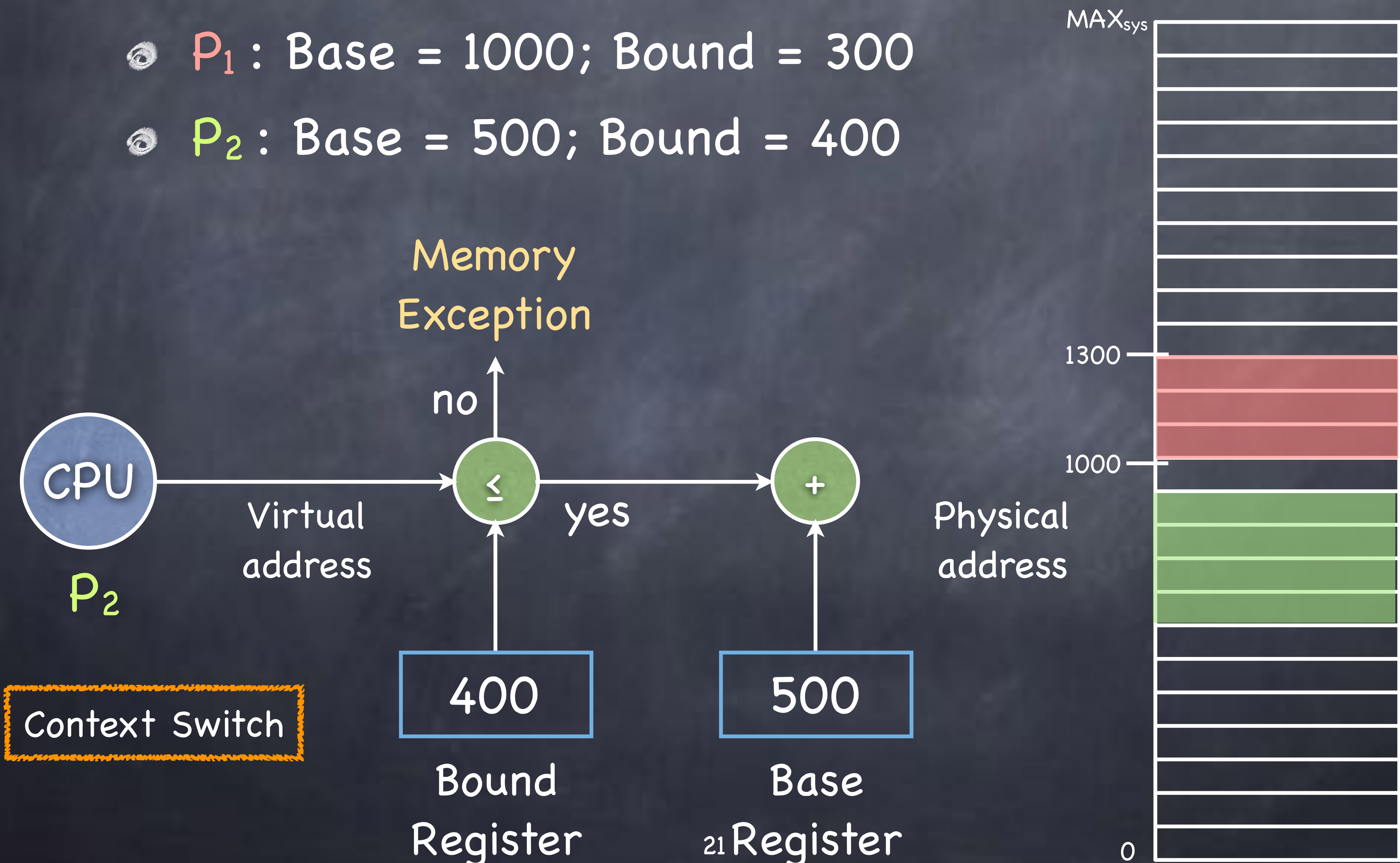
Base & Bound

- P_1 : Base = 1000; Bound = 300
- P_2 : Base = 500; Bound = 400



Base & Bound

- P_1 : Base = 1000; Bound = 300
- P_2 : Base = 500; Bound = 400



On Base & Bound

- Contiguous Allocation
  - contiguous virtual addresses are mapped to contiguous physical addresses
- But mapping entire address space to physical memory
  - is wasteful
    - lots of free space between heap and stack…
    - makes sharing hard
  - does not work if the address space is larger than physical memory
    - think 64-bit registers…

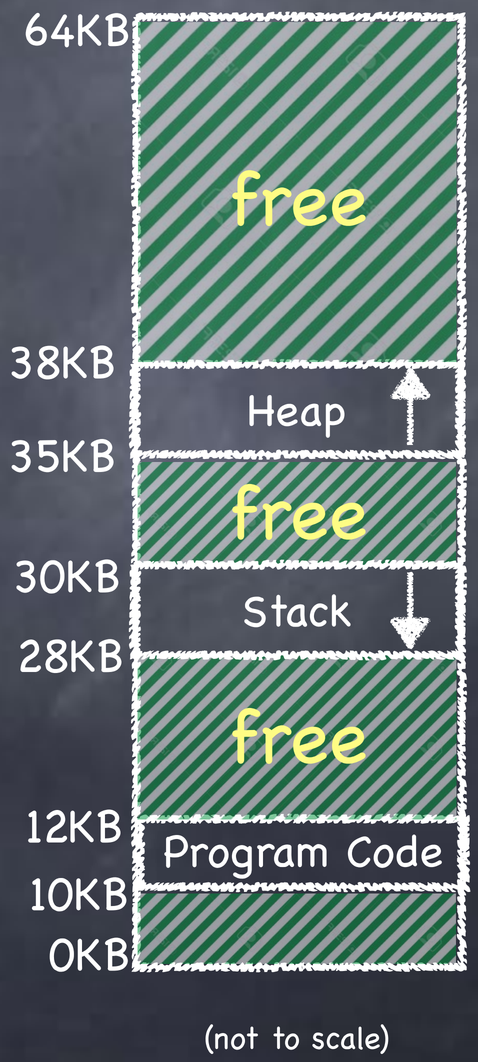
E Pluribus Unum

- An address space comprises multiple segments
  - contiguous sets of virtual addresses, logically connected
    - heap, code, stack, (and also globals, libraries...)
  - each segment can be of a different size

Segmentation: Generalizing Base & Bound

- Base & Bound registers to each segment
  - each segment independently mapped to a set of contiguous addresses in physical memory
    - no need to map unused virtual addresses

Segment	Base	Bound
Code	10K	2K
Stack	28	2K
Heap	35K	3K

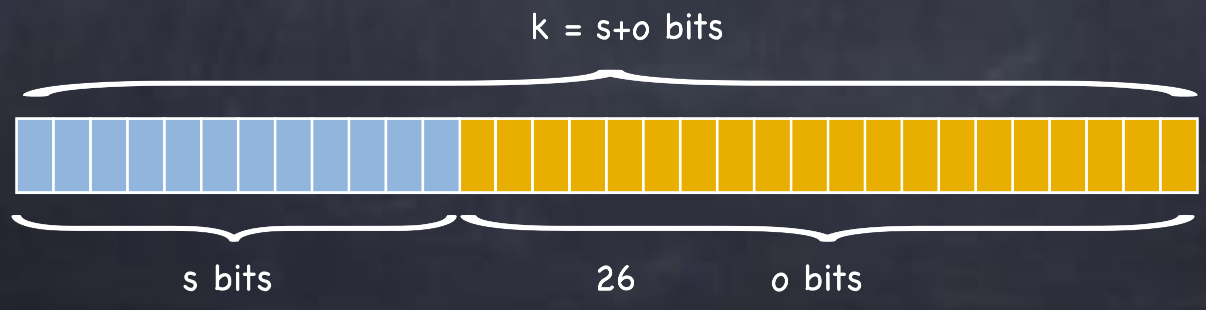


Segmentation

- Goal: Supporting large address spaces (while allowing multiple processes to coexist in memory)
- Needed hardware
  - two registers (Base and Bound) per segment
    - values stored in the PCB
  - if many segments, a segment table, stored in memory, at an address pointed to by a Segment Table Register (STBR)
    - process' STBR value stored in the PCB

Segmentation: Mapping

- How do we map a virtual address to the appropriate segment?
  - Read VA as having two components
    - s most significant bits identify the segment
      - at most 2^s segments
    - o remaining bits identify offset within segment
      - each segment's size can be at most 2^o bytes



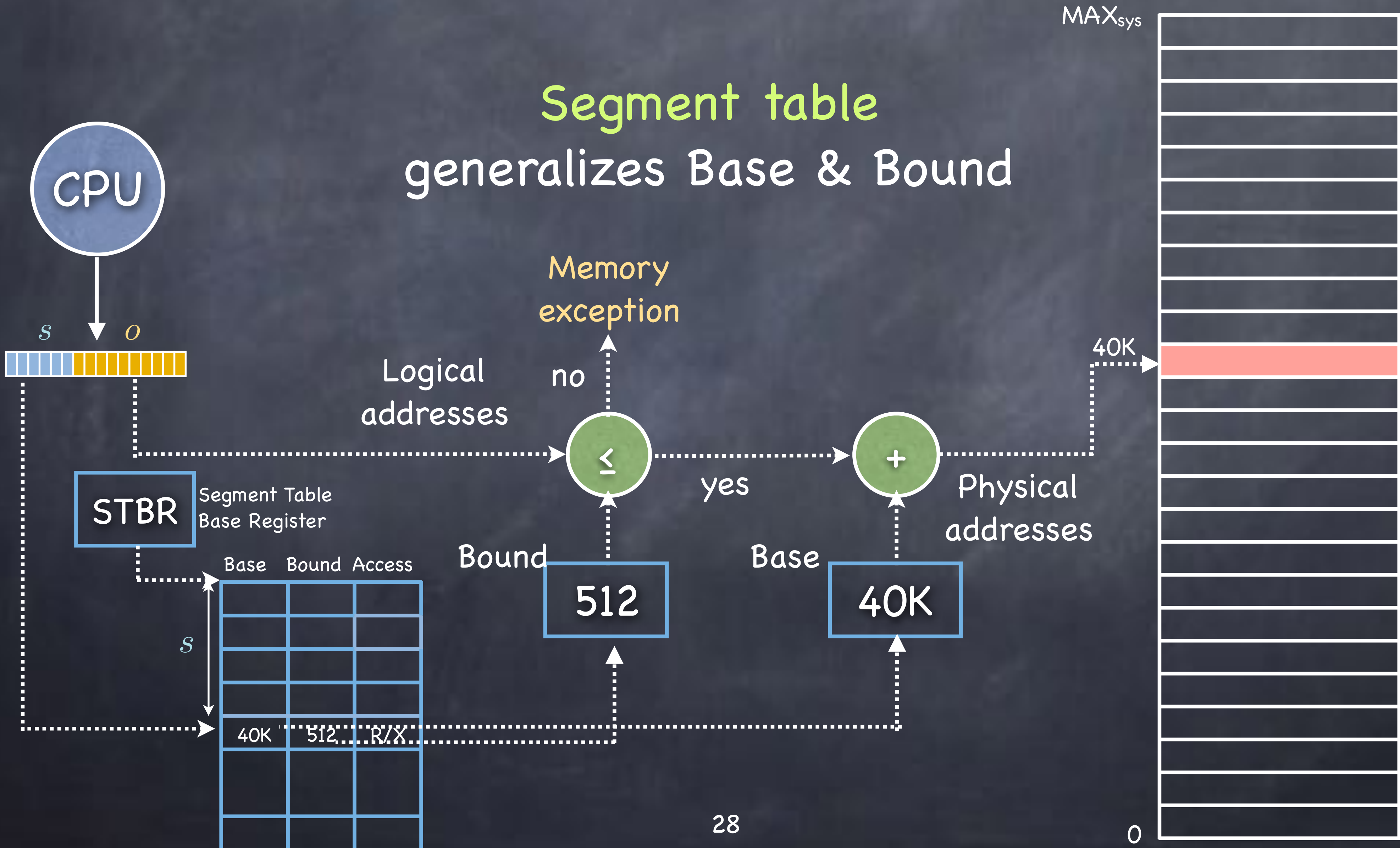
Segment Table

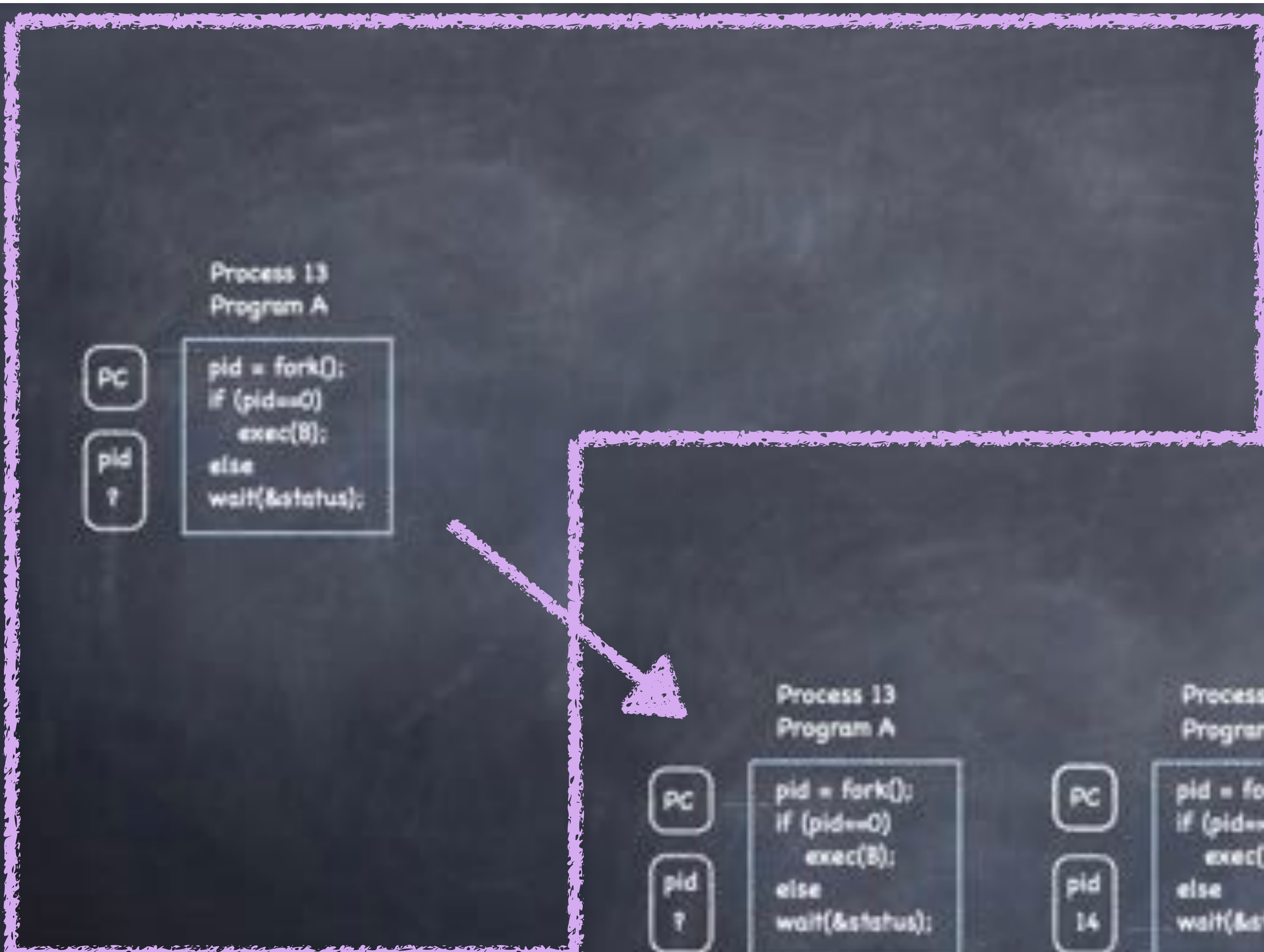
- Use s bits to index to the appropriate row of the segment table

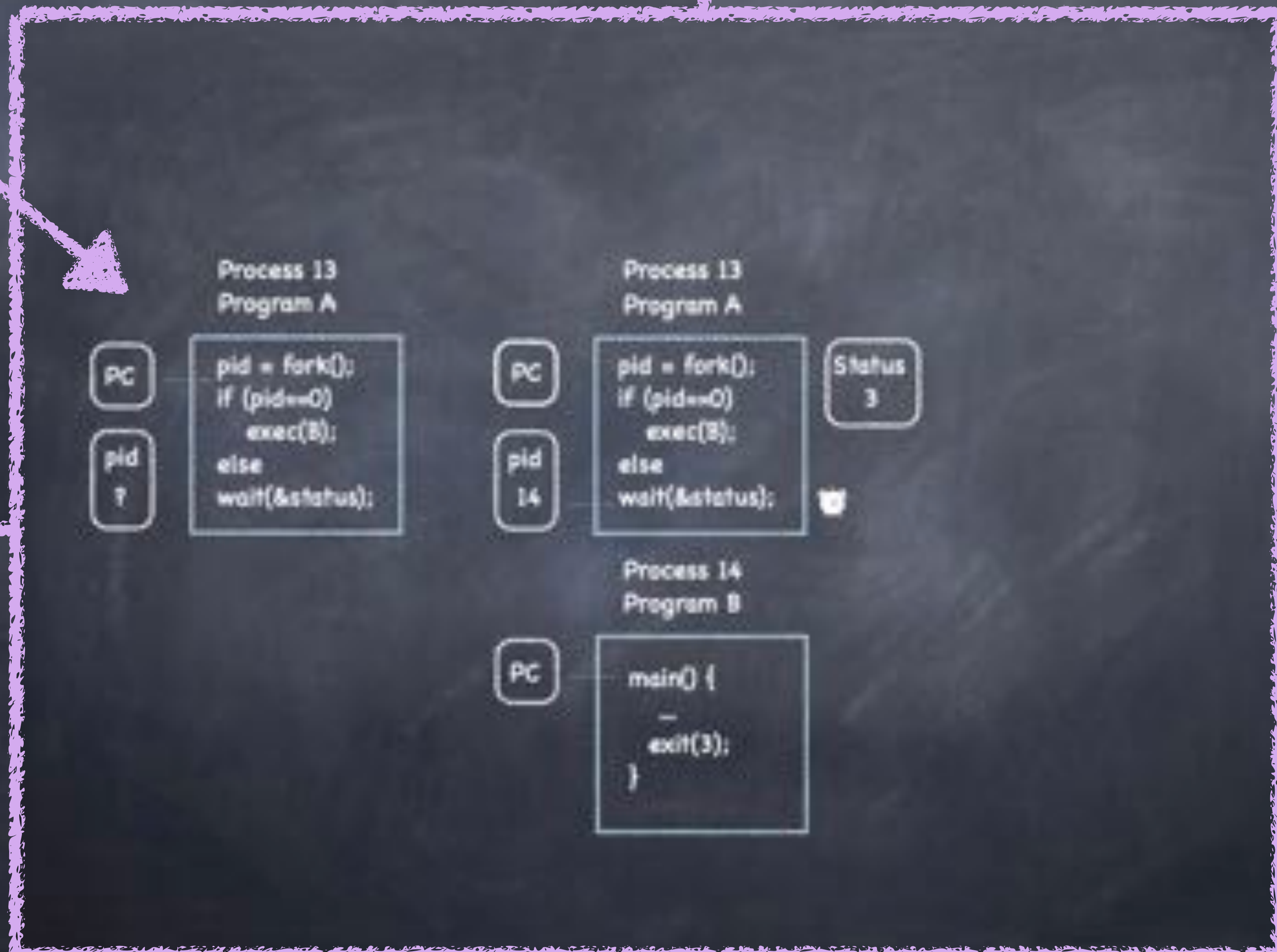
	Base	Bound (Max 4k)	Access
Code_{00}	32K	2K	Read/Execute
Heap_{01}	34K	3K	Read/Write
Stack_{10}	28K	3K	Read/Write

- Segments can be shared by different processes
  - use protection bits to determine if shared Read only (maintaining isolation) or Read/Write (if shared, no isolation)
    - processes can share code segment while keeping data private

Implementing Segmentation







Revisiting fork()

Revisiting fork()

- Copying an entire address space can be costly…
  - especially if you proceed to obliterate it right away with exec()!

Revisiting fork(): Segments to the Rescue

- Instead of copying entire address space, copy just segment table (the VA->PA mapping)

	Base	Bound	Access
Code	32K	2K	RX
Heap	34K	3K	RW
Stack	28K	3K	RW

Parent

	Base	Bound	Access
Code	32K	2K	RX
Heap	34K	3K	RW
Stack	28K	3K	RW

Child

- but change all writeable segments to Read only

Revisiting fork(): Segments to the Rescue

- Instead of copying entire address space, copy just segment table (the VA->PA mapping)

	Base	Bound	Access
Code	32K	2K	RX
Heap	34K	3K	R
Stack	28K	3K	R

Parent

	Base	Bound	Access
Code	32K	2K	RX
Heap	34K	3K	R
Stack	28K	3K	R

Child

- but change all writeable segments to Read only

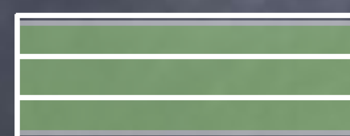
- Segments in VA spaces of parent and child point to same locations in physical memory

Copy on Write (COW)

- When trying to modify an address in a COW segment:
  - exception!
    - exception handler copies just the affected segment, and changes both the old and new segment back to writeable
- If exec() is immediately called, only stack segment is copied!
  - it stores the return value of the fork() call, which is different for parent and child

Managing Free space

- Many segments, different processes, different sizes
- OS tracks free memory blocks ("holes")
  - Initially, one big hole
- Many strategies to fit segment into free memory (think "assigning classrooms to courses")



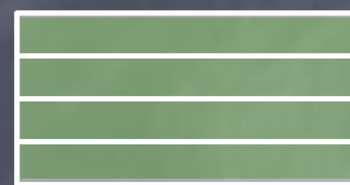
  - First Fit: first big-enough hole
  - Next Fit: Like First Fit, but starting from where you left off
  - Best Fit: smallest big-enough hole
  - Worst Fit: largest big-enough hole

External Fragmentation

- Over time, memory can become full of small holes
  - Hard to fit more segments
  - Hard to expand existing ones
- Compaction
  - Relocate segments to coalesce holes

External Fragmentation

- Over time, memory can become full of small holes
  - Hard to fit more segments
  - Hard to expand existing ones
- Compaction
  - Relocate segments to coalesce holes

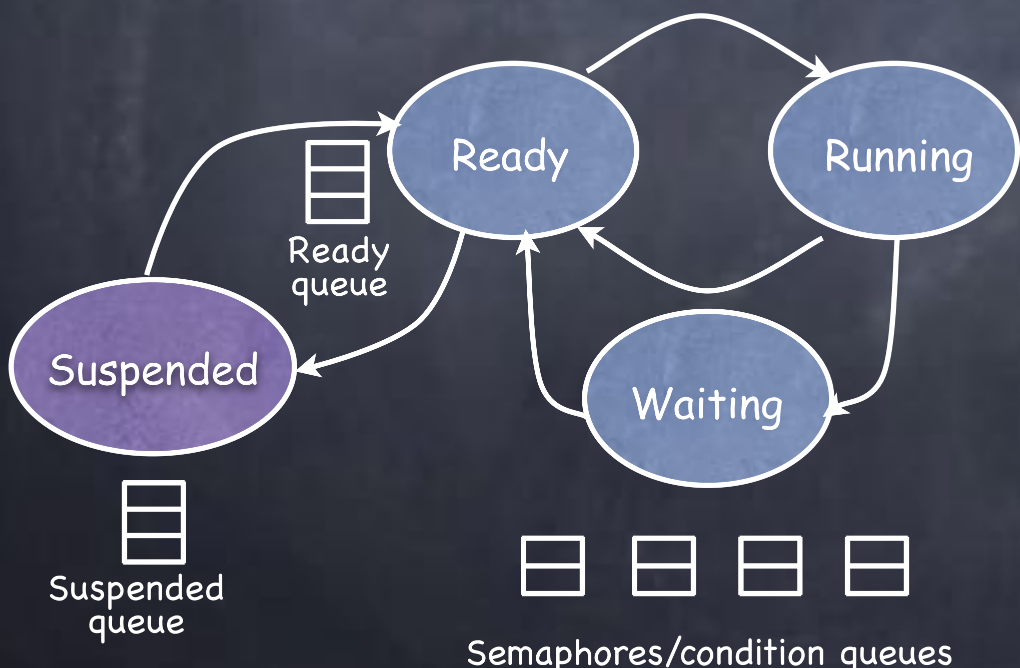


External Fragmentation

- Over time, memory can become full of small holes
  - Hard to fit more segments
  - Hard to expand existing ones
- Compaction
  - Relocate segments to coalesce holes
    - Copying eats up a lot of CPU time!
      - if 4 bytes in 10ns, 8 GB in 20s!
- But what if a segment wants to grow?

Eliminating External Fragmentation: Swapping

- Preempt processes and reclaim their memory



- Move images of suspended processes to backing store