Announcements

- Change of time for exercises session
- **Thursdays** from **16h00 to 17h00**
- Room **CM1 121**
- Feedback
  - Reminder: 3 ETCS course, 2nd-3rd year bachelor, only theory (this semester)

Recap – Week 4

Willy Zwaenepoel

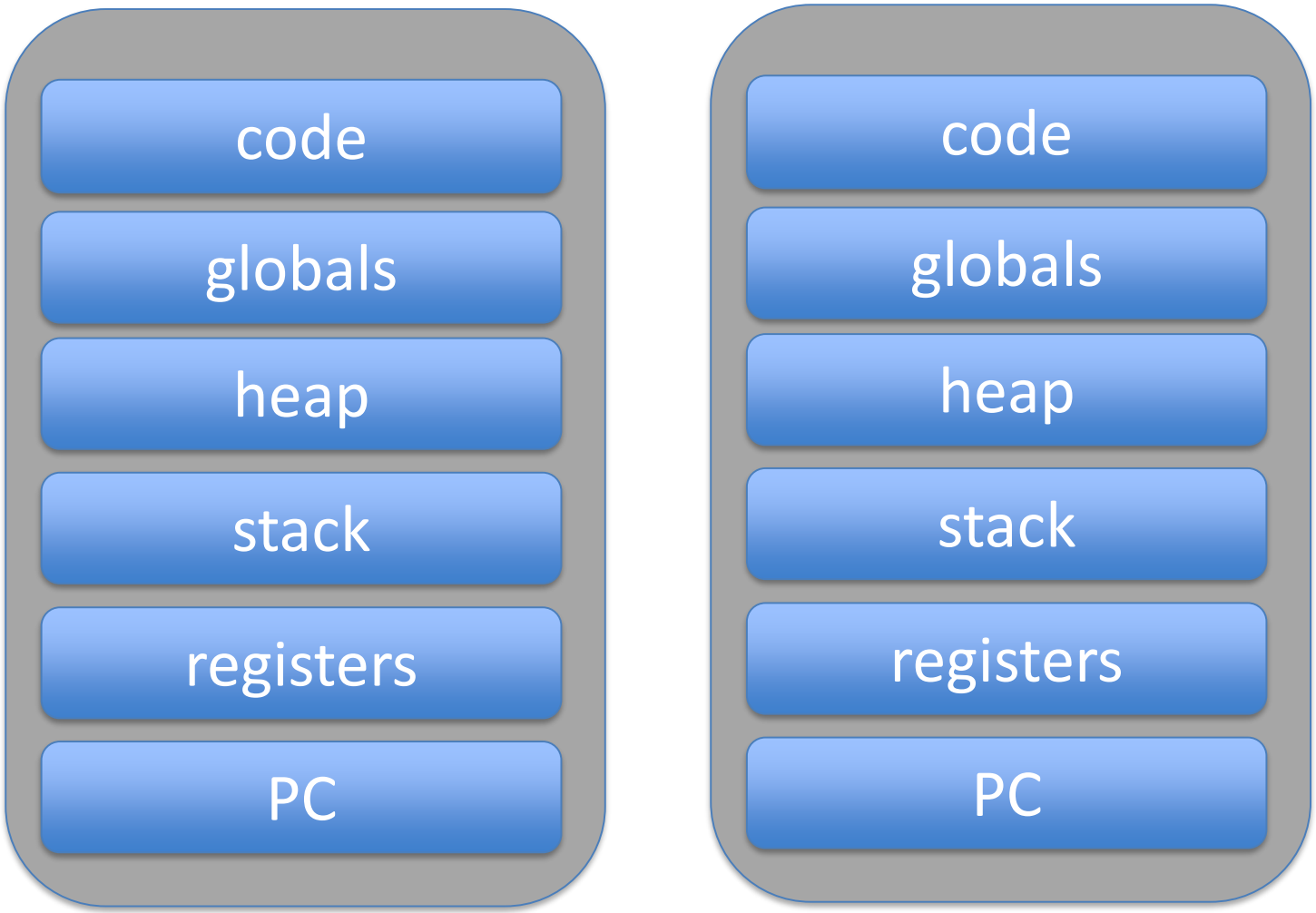
March 14, 2018

Multithreading vs Multiprocessing

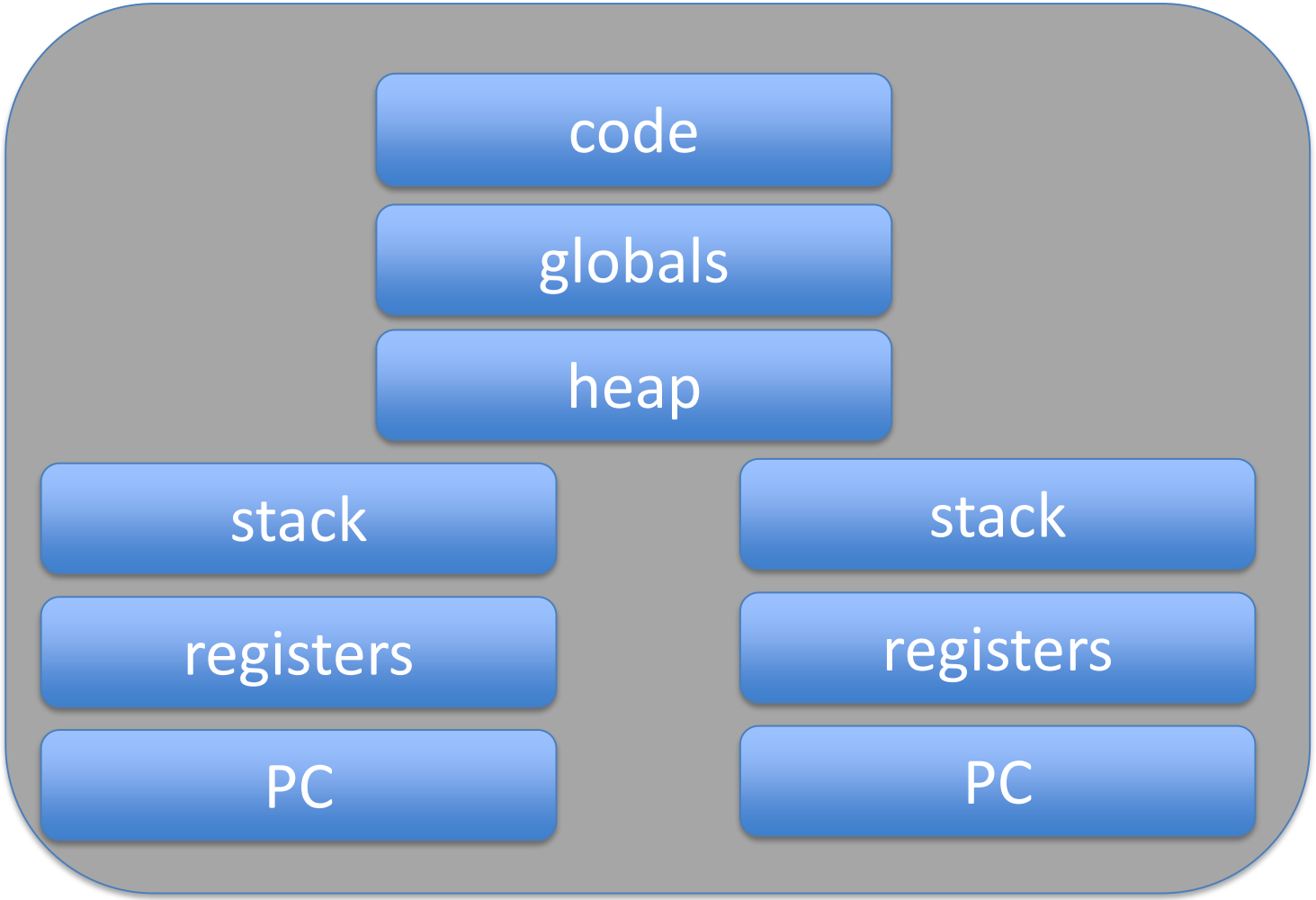
- Threads share code, heap and globals
- Threads have separate stack, PC, register

- Processes do not share anything

Two Processes



Two Threads in a Process



Multithreading

- Deriving multithreaded from single-threaded
  - Divide work
  - Locate shared data and accesses to it
  - Synchronize with one big lock
  - Optimize with fine-grain locks and privatization
  - Sometimes need to introduce shared data
- Pthreads

Week 5
Memory Management: Virtual Memory

Pamela Delgado

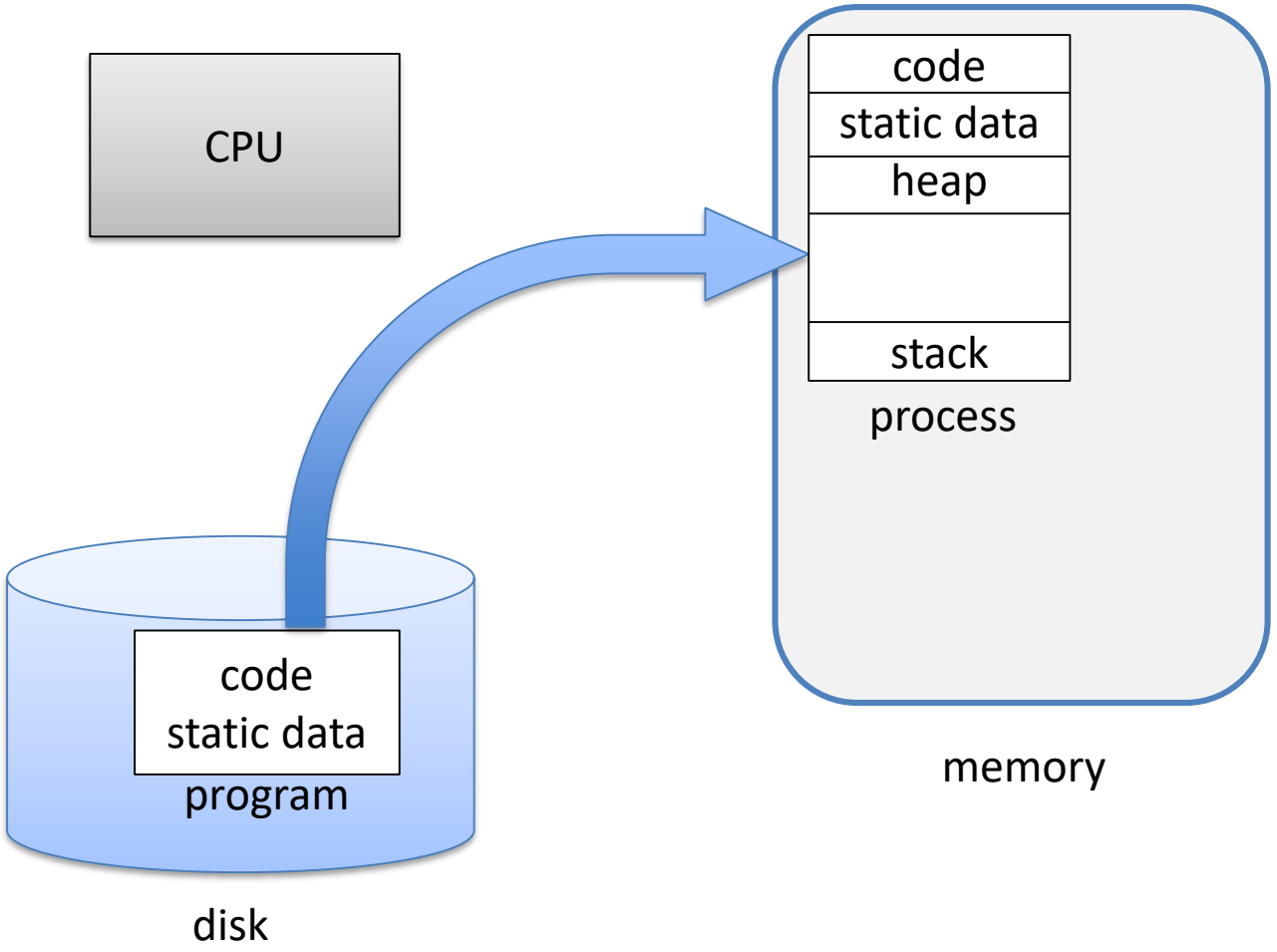
March 20, 2019

based on:
- W. Zwaenepoel slides
- Arpaci-Dusseau book
- Silbershatz book

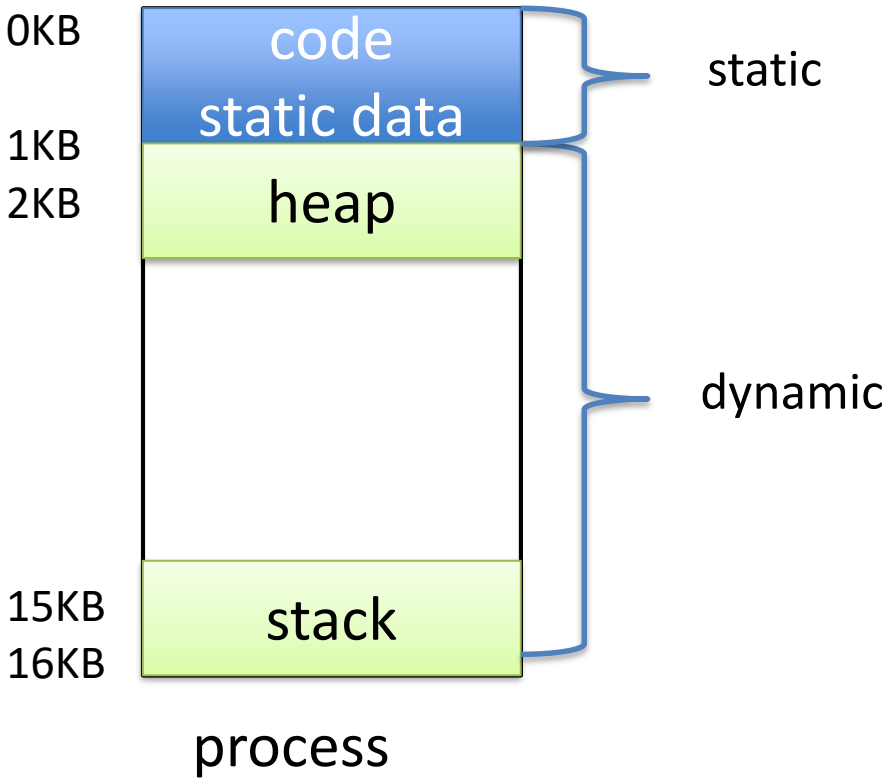
Key Concepts

- Virtual and physical address spaces
- Mapping between virtual and physical address
- Different mapping methods:
  - Base and bounds
  - Segmentation
  - Paging
  - Segmented paging
- Sharing, protection, memory allocation

Program to process



Process address space



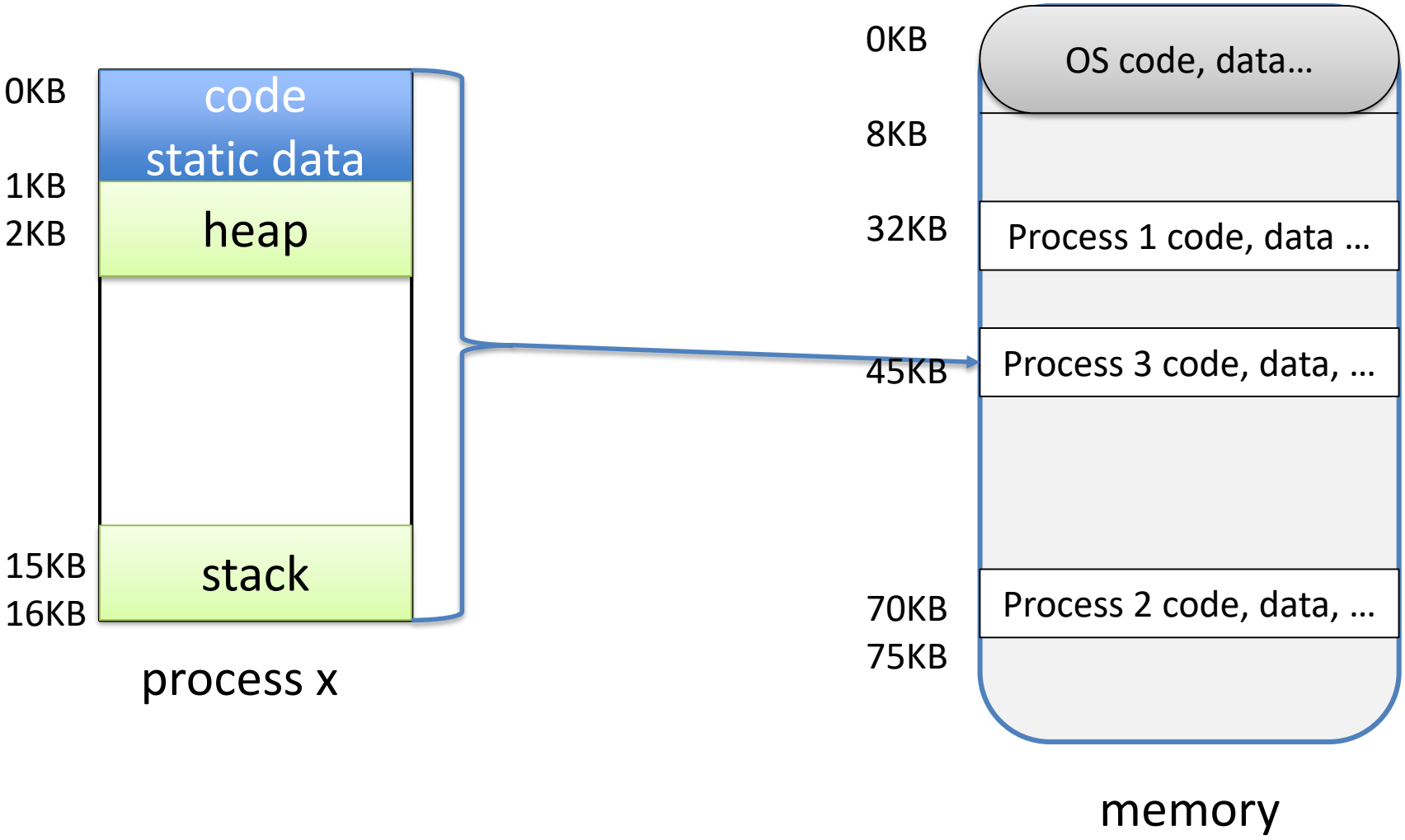
Ever seen a memory address?

- Pointer address?

```
int main (){
    printf((void *) main)
    int x = 1
    printf(&x)
    printf((void *) malloc(1))
}
```

- Some hexadecimal

Reality: virtual address



Questions

- What if process does not fit in memory?
- Deal with fragmentation?
- Process splitted in different parts in memory?
- How much memory to give to a program in the first place?

OS abstraction: memory

- Goals
  1. Main memory allocation
  2. Protection
     - Isolation from other programs
  3. Transparency*
     - User should not be aware of virtualization
  4. Efficiency
     - Space and time

Simplifying Assumption

- For this week's lecture only
- All of a program must be in memory
- Will revisit assumption next week

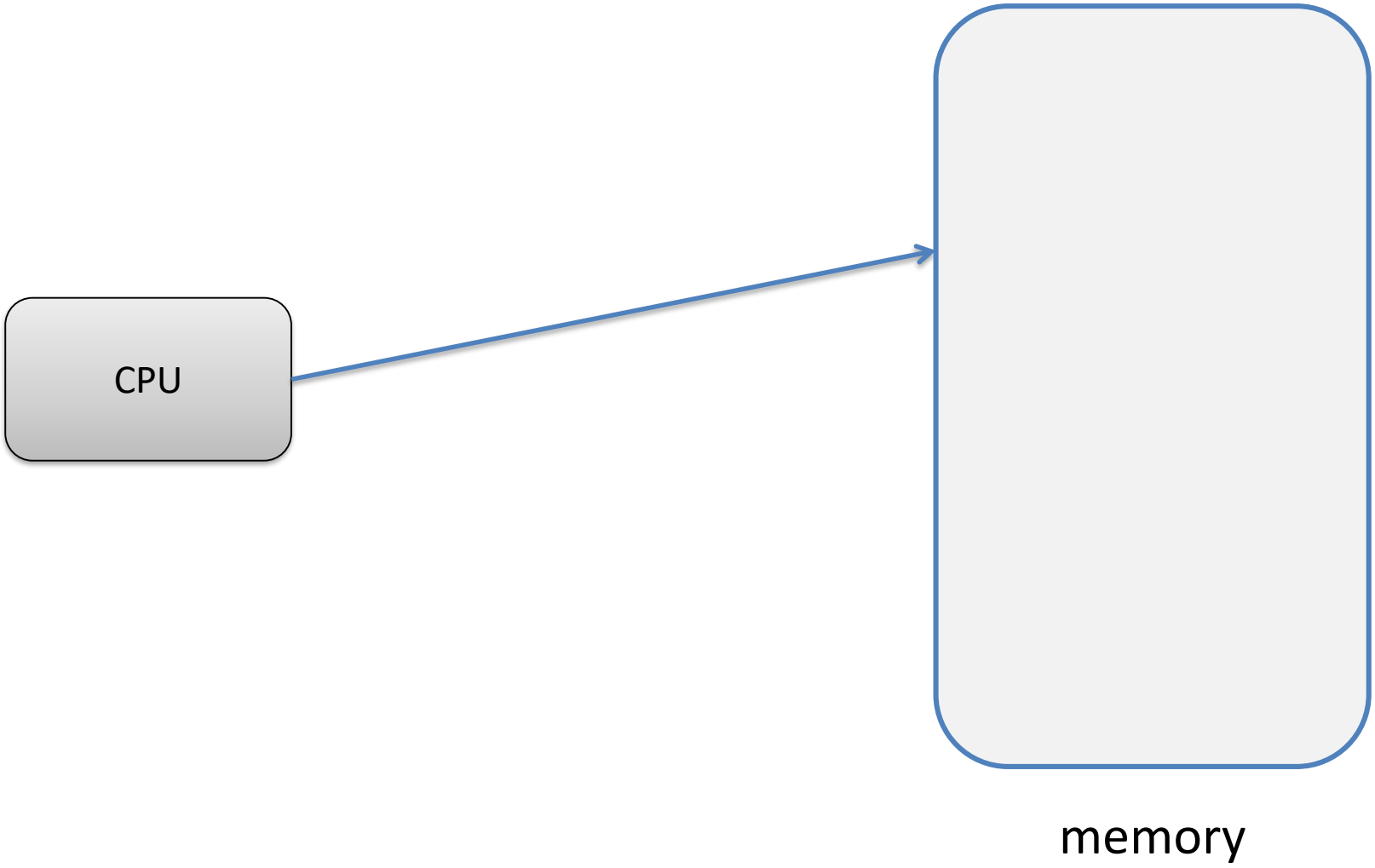
1. Main Memory Allocation

- Where to locate the kernel?
- How many processes to allow?
  - Called “degree of multiprocessing”
- What memory to allocate to processes?

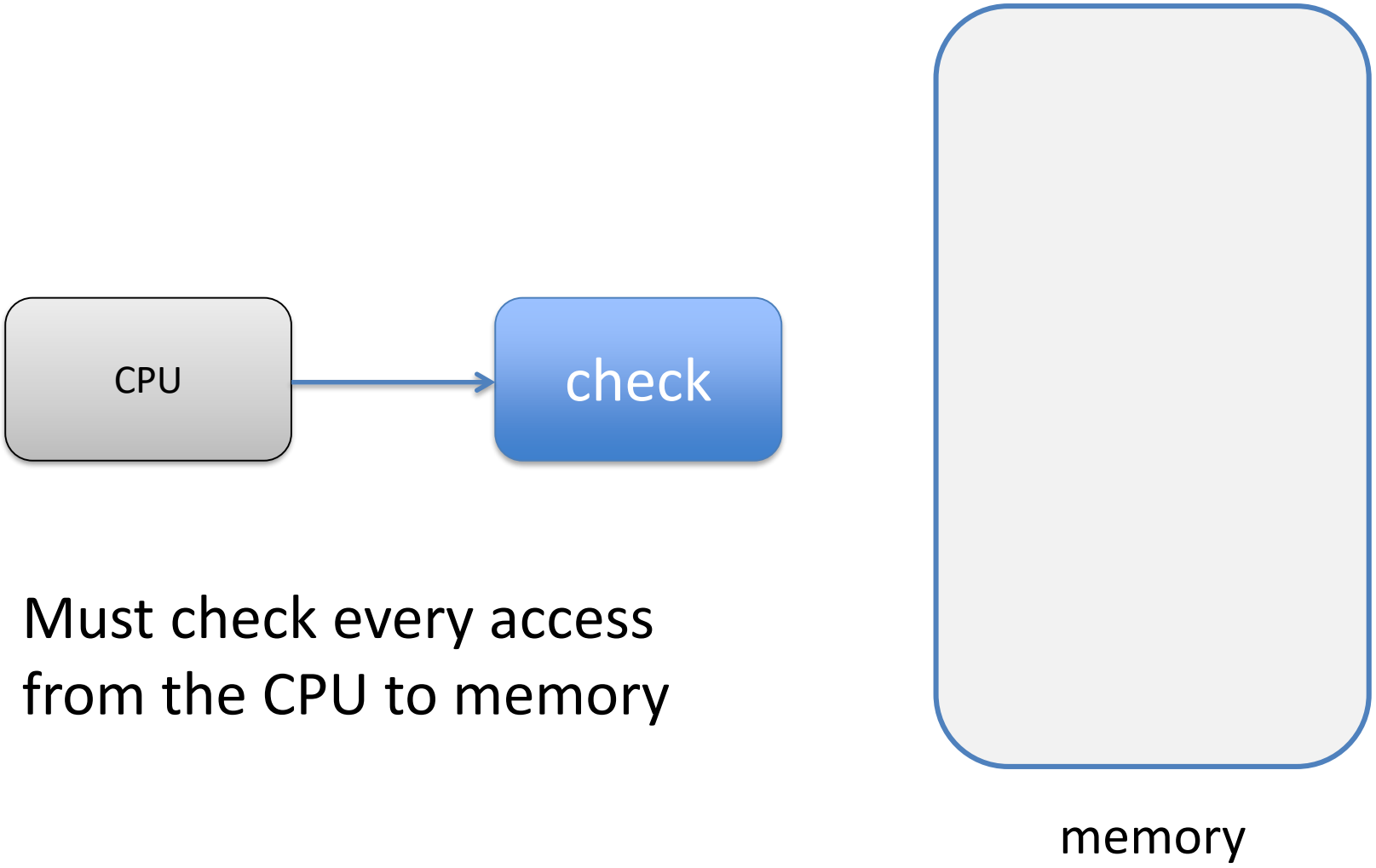
2. Protection

- One process must not be able to read or write the memory
  - of another process
  - of the kernel

Unprotected Access

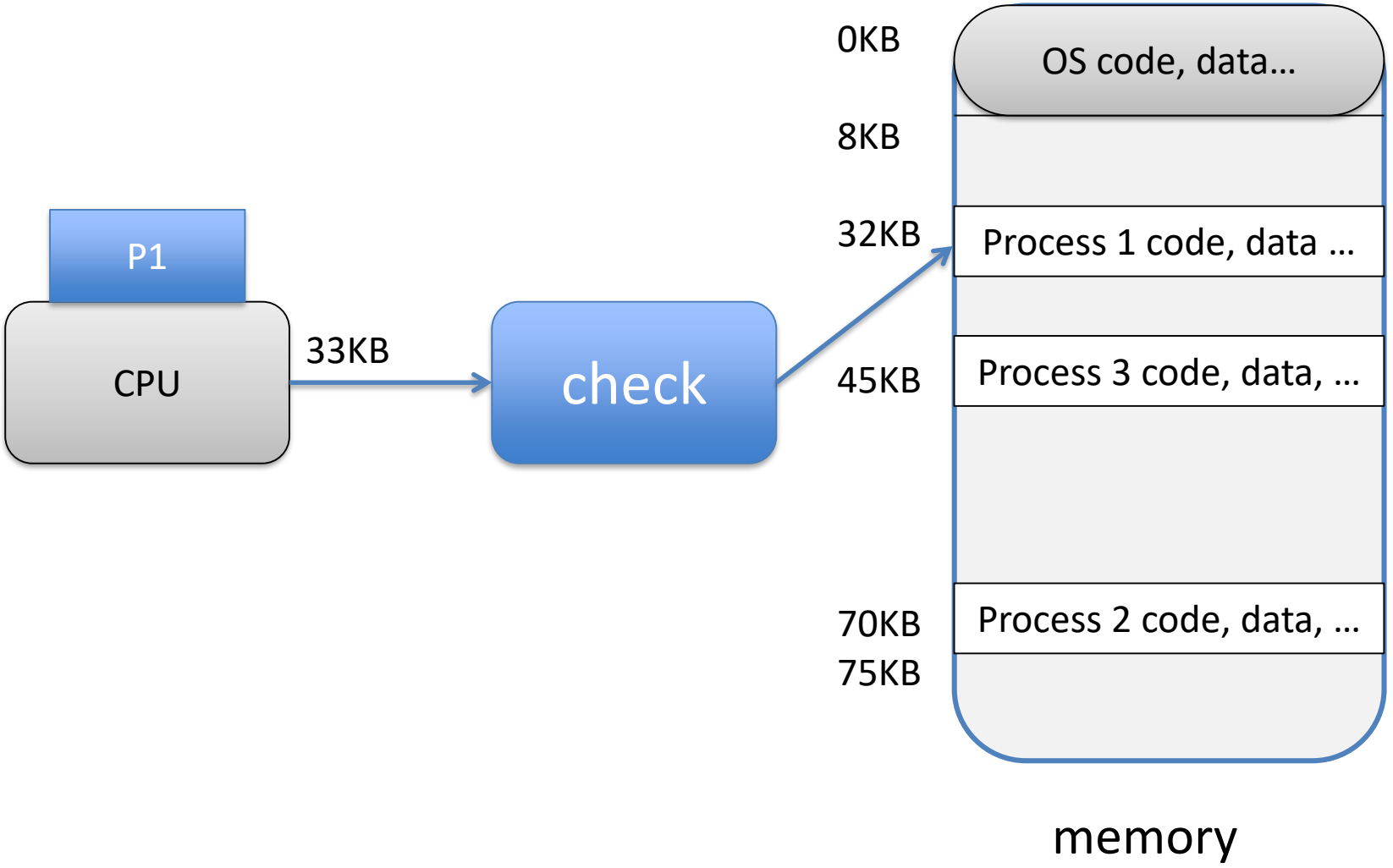


Protected Access

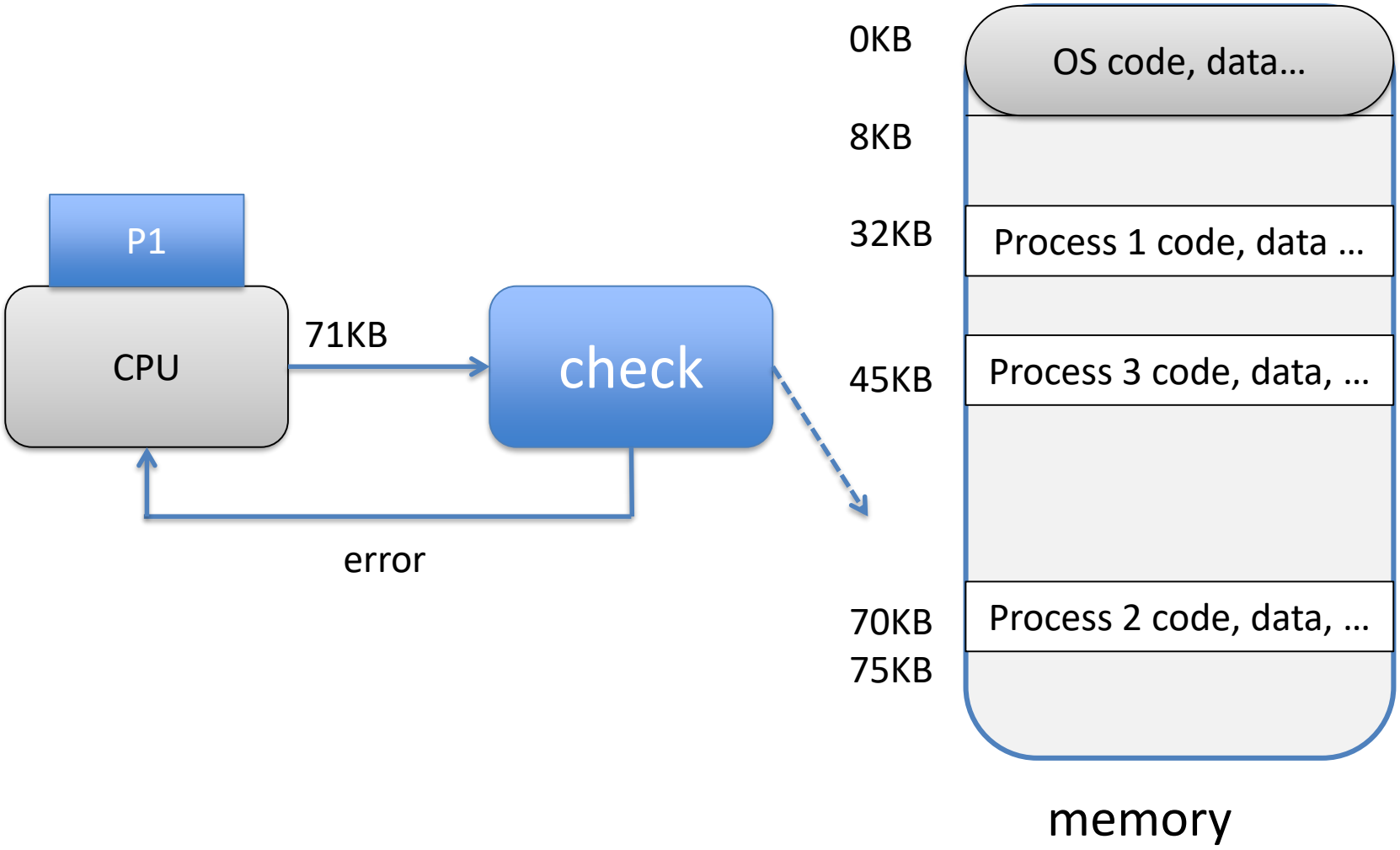


Must check every access
from the CPU to memory

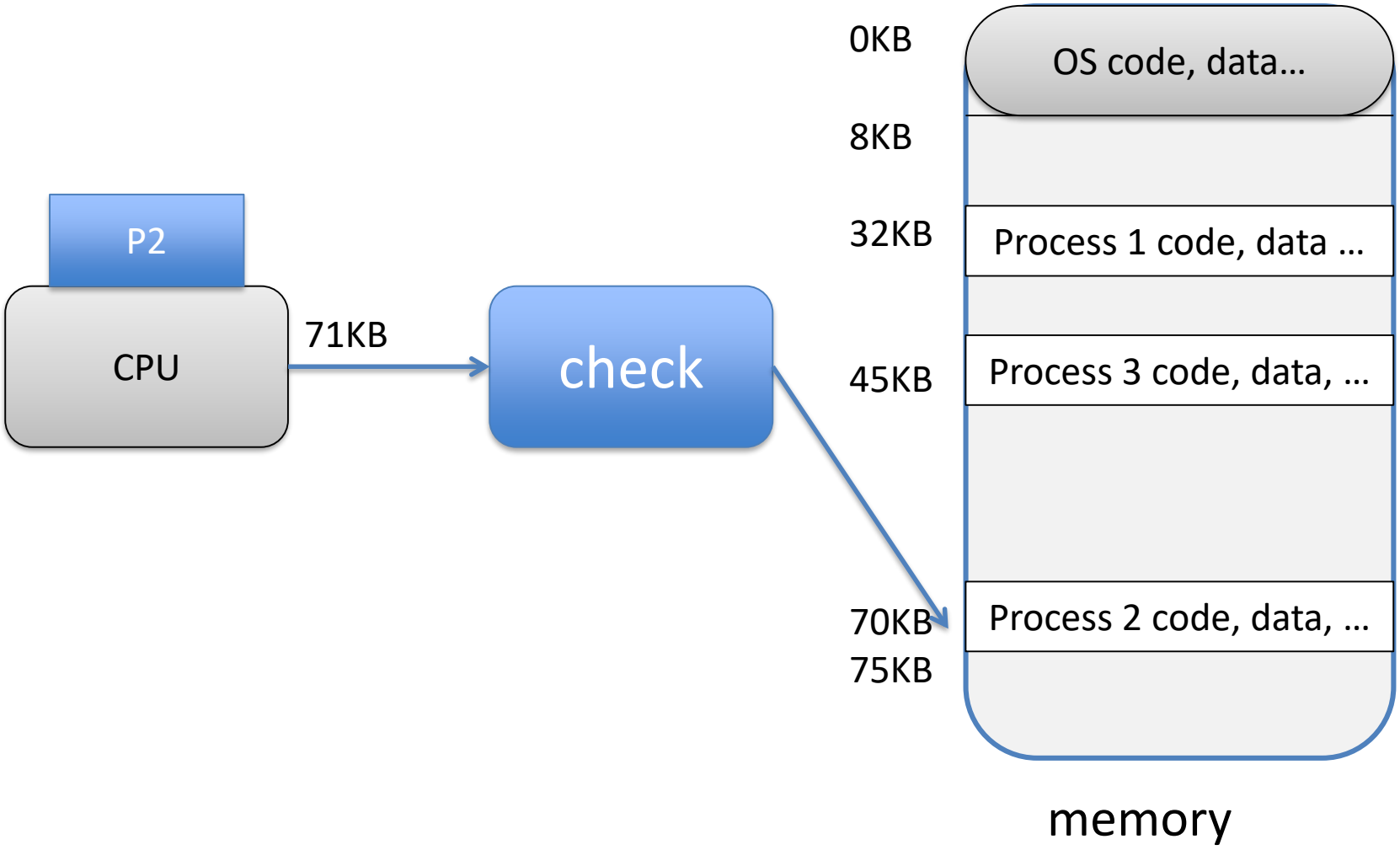
Protection: Examples



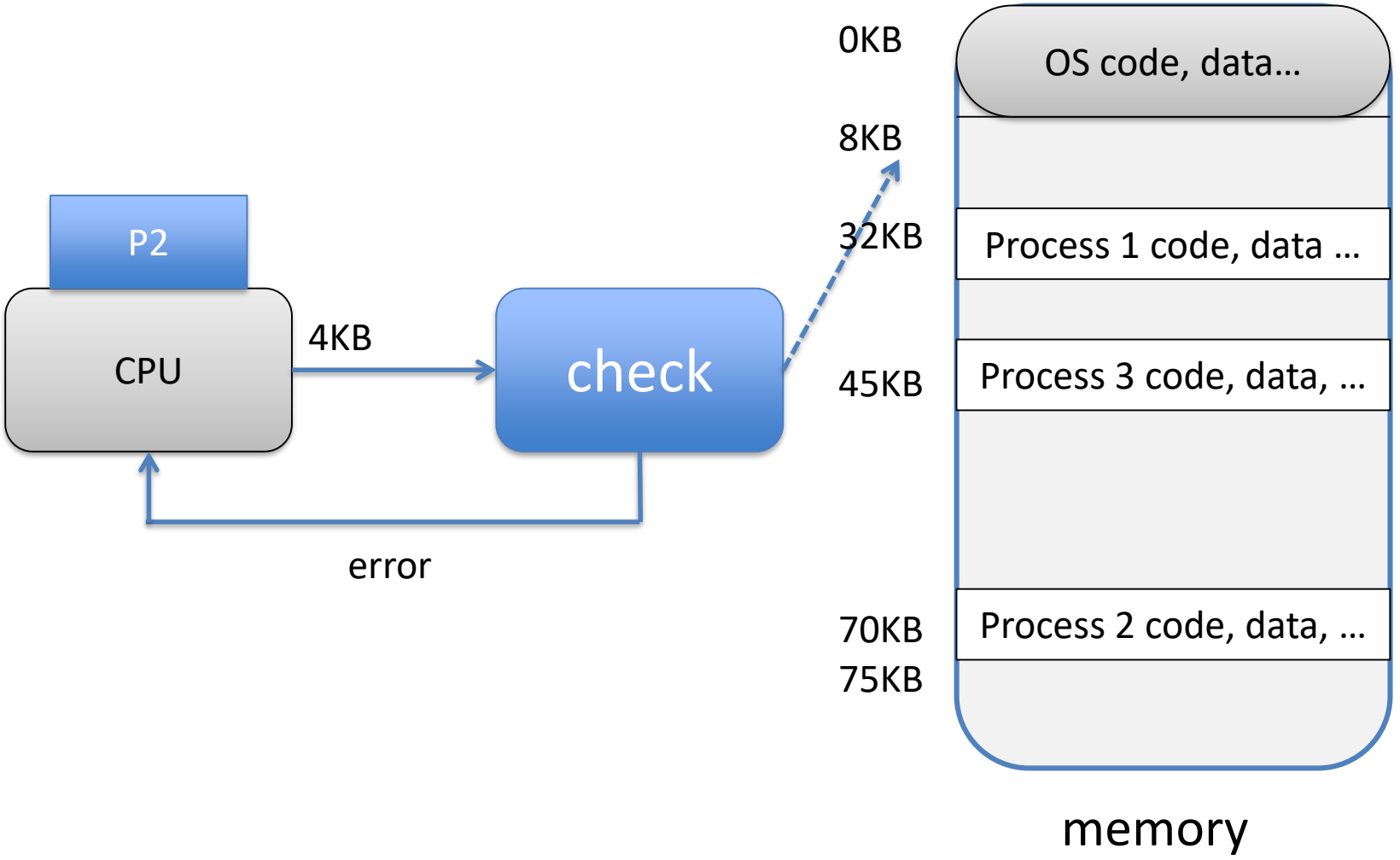
Protection: Examples



Protection: Examples



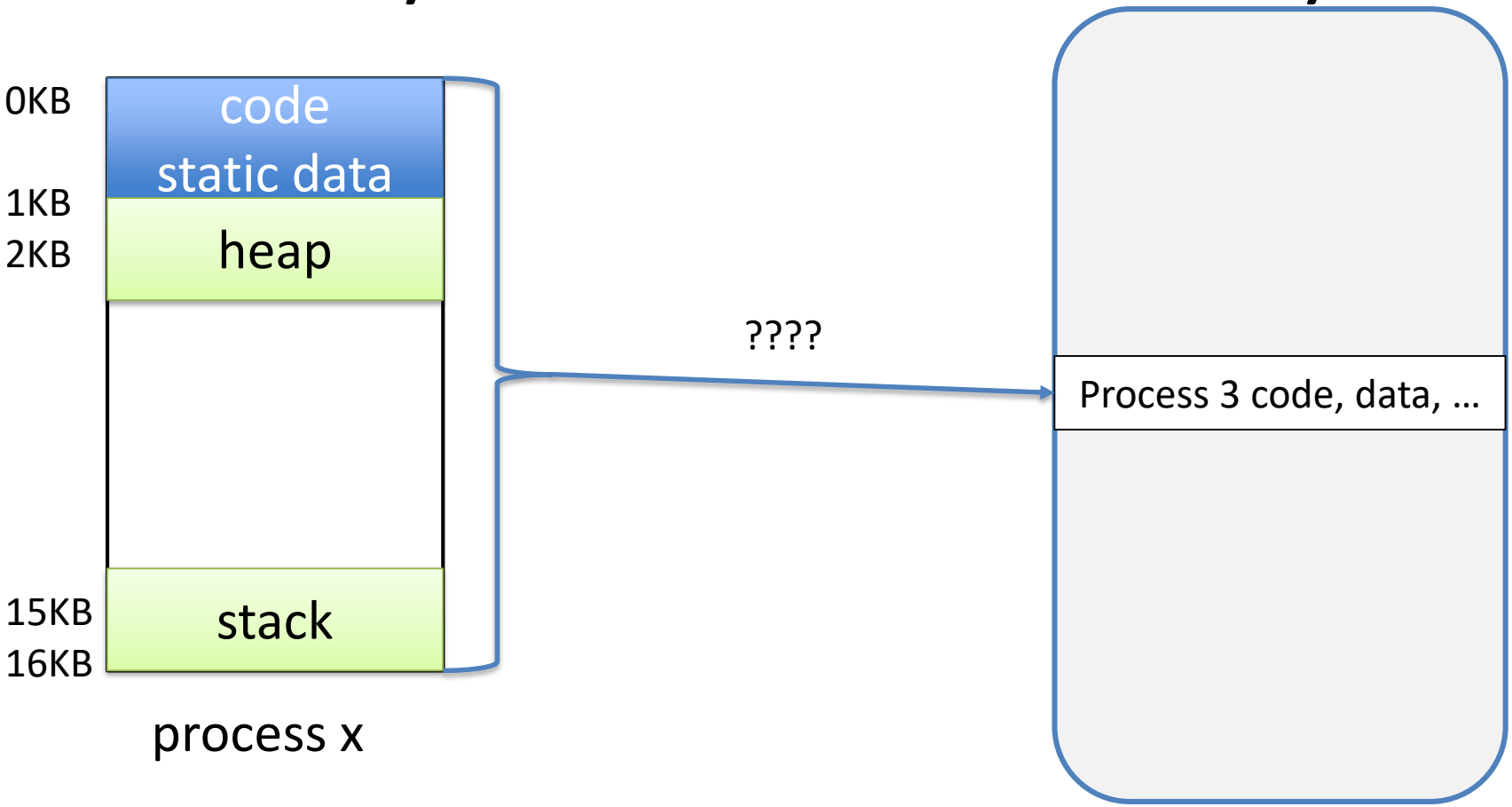
Protection: Examples



3. Transparency

- Programmer should not have to worry
  - where his program is in memory
  - where or what other programs are in memory

Transparency: Program can be Anywhere in Main memory

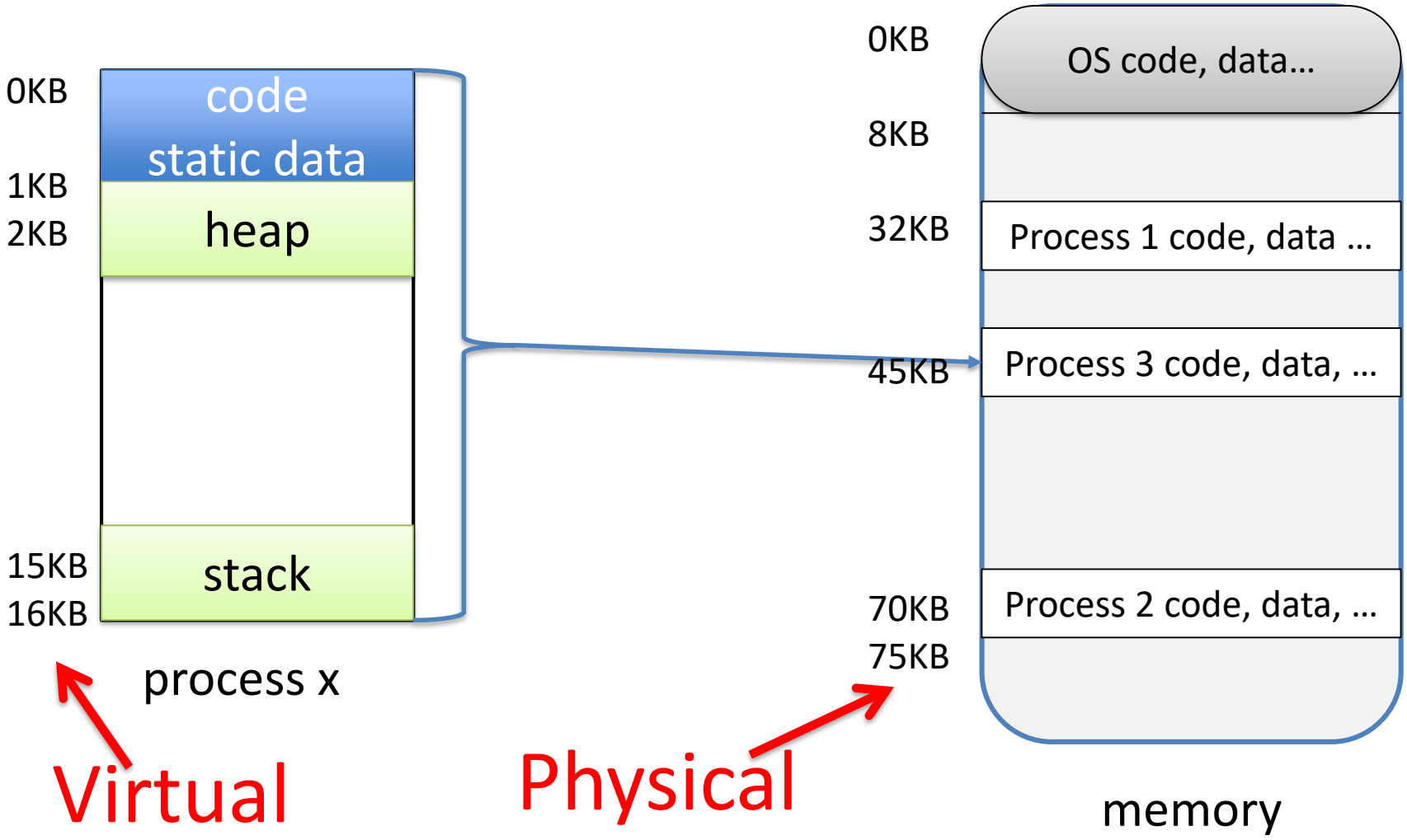


Program not aware of virtualization

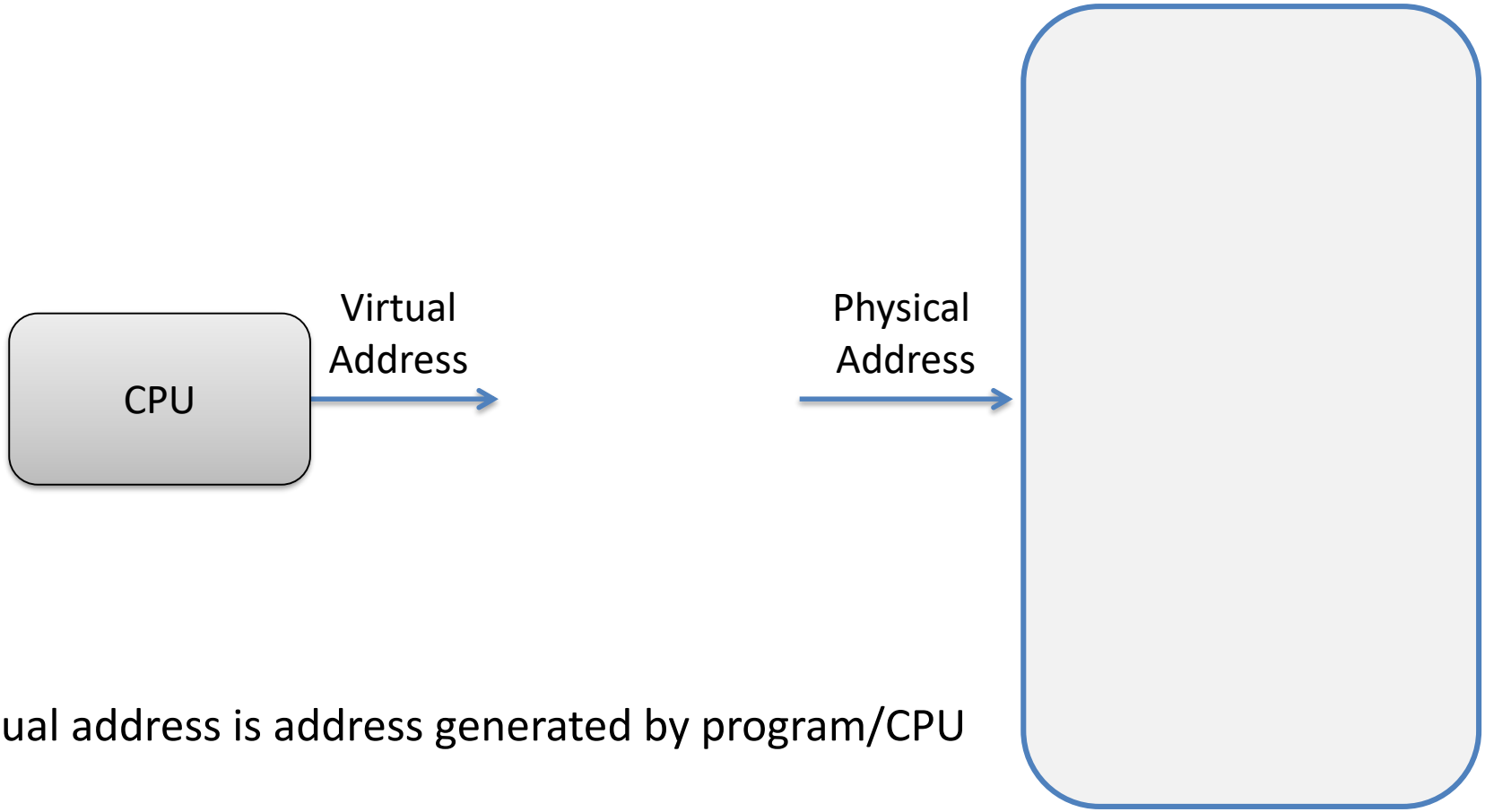
Virtual vs. Physical Address Space

- Virtual/logical address space =
  - What the program(mer) thinks is its memory
- Physical address space =
  - Where the program is in physical memory

Virtualization of memory



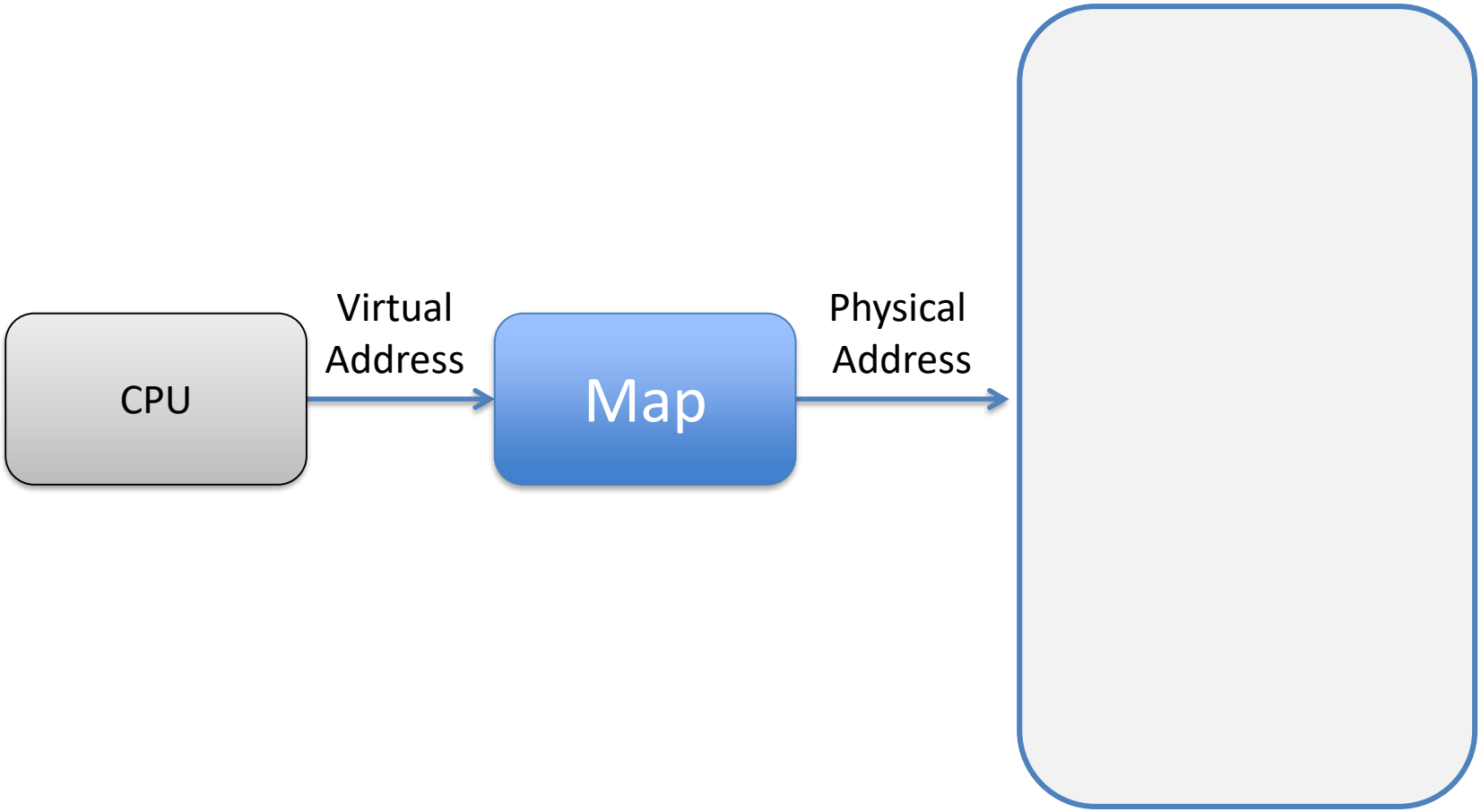
Another Way to Understand Virtual vs. Physical



Virtual address is address generated by program/CPU

Physical address is address that the memory sees

Translating Virtual to Physical

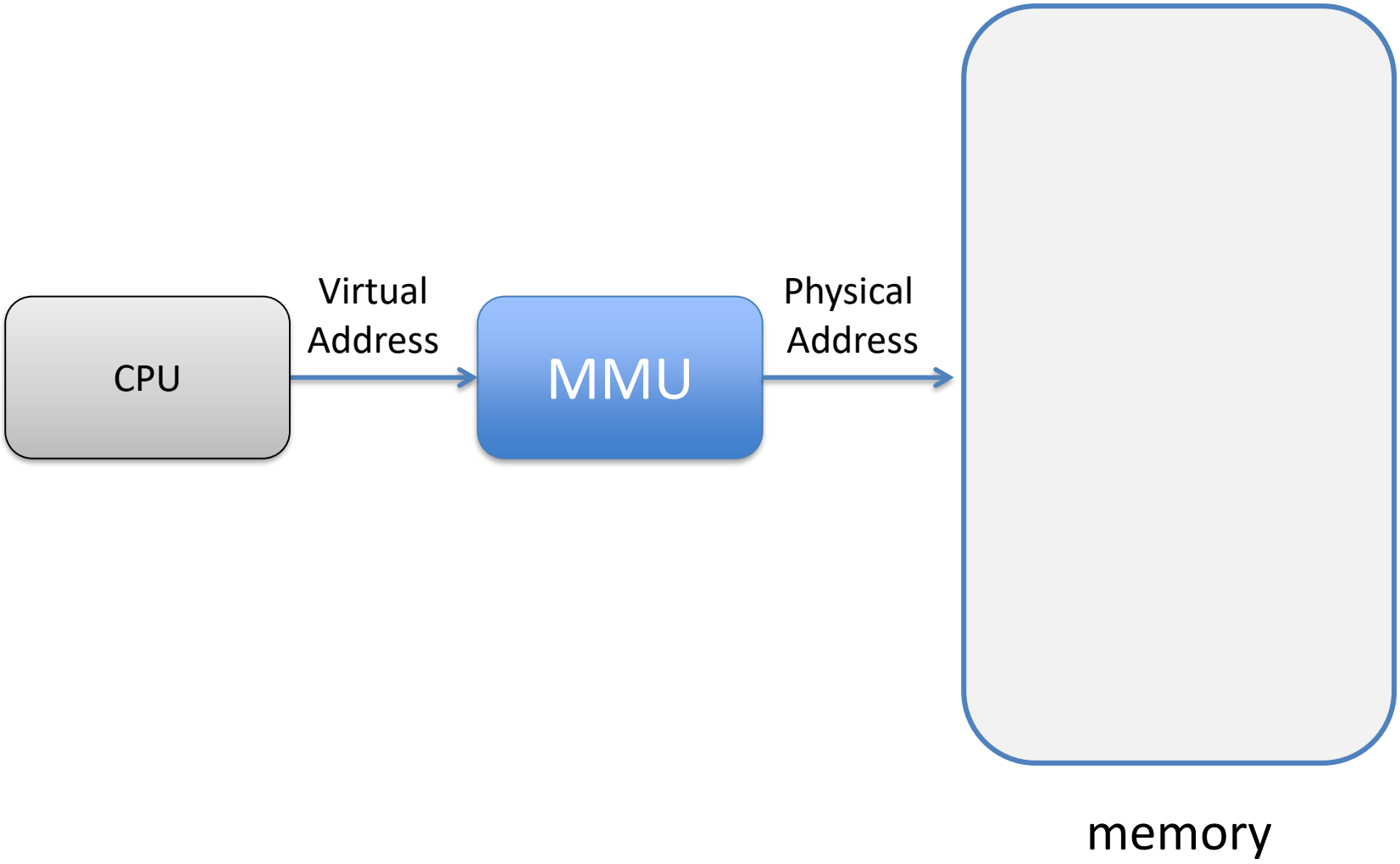


Must map/translate every access from the CPU to memory

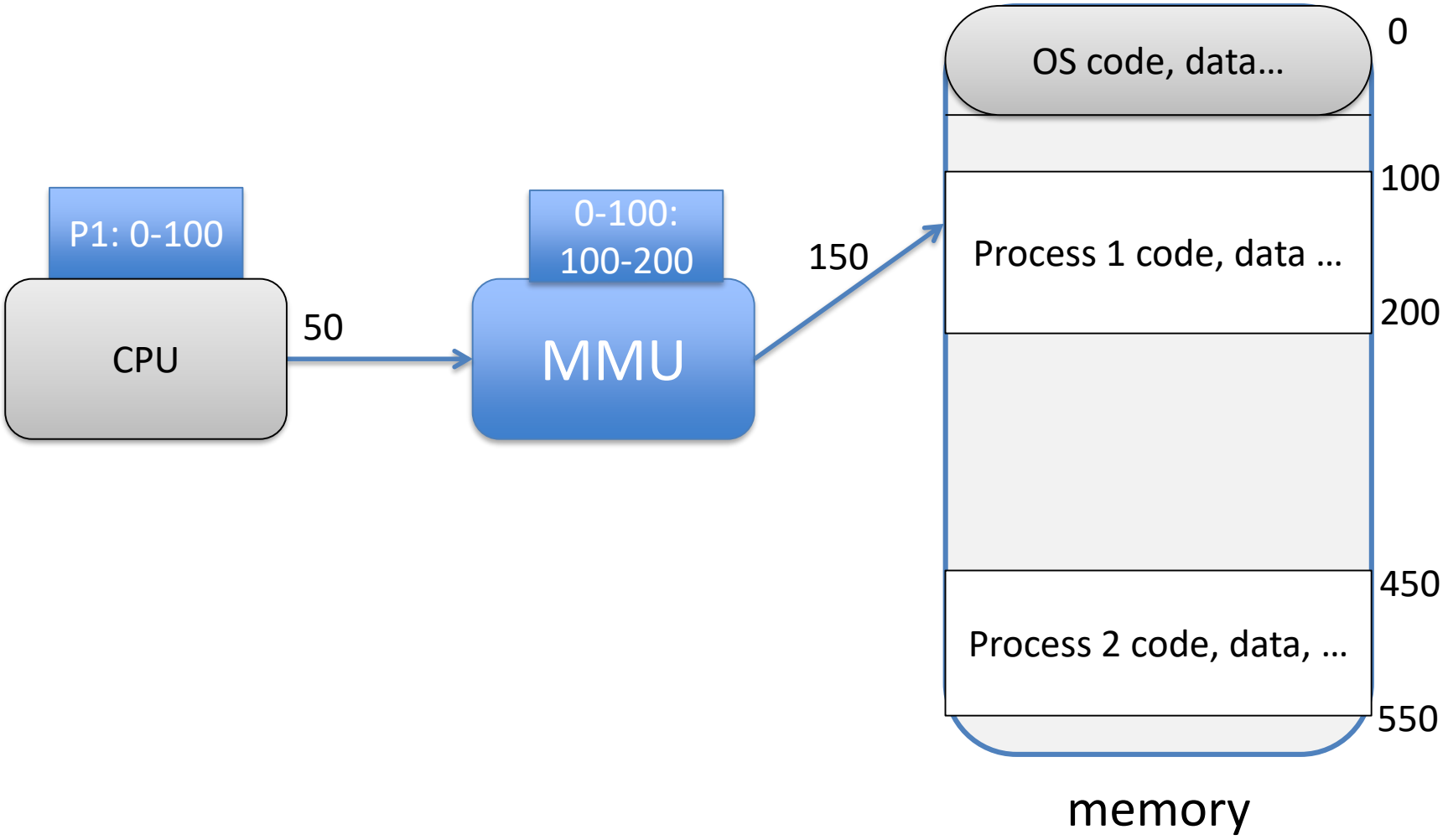
Memory Management Unit (MMU)

- Provides mapping virtual-to-physical
- Provides protection at the same time

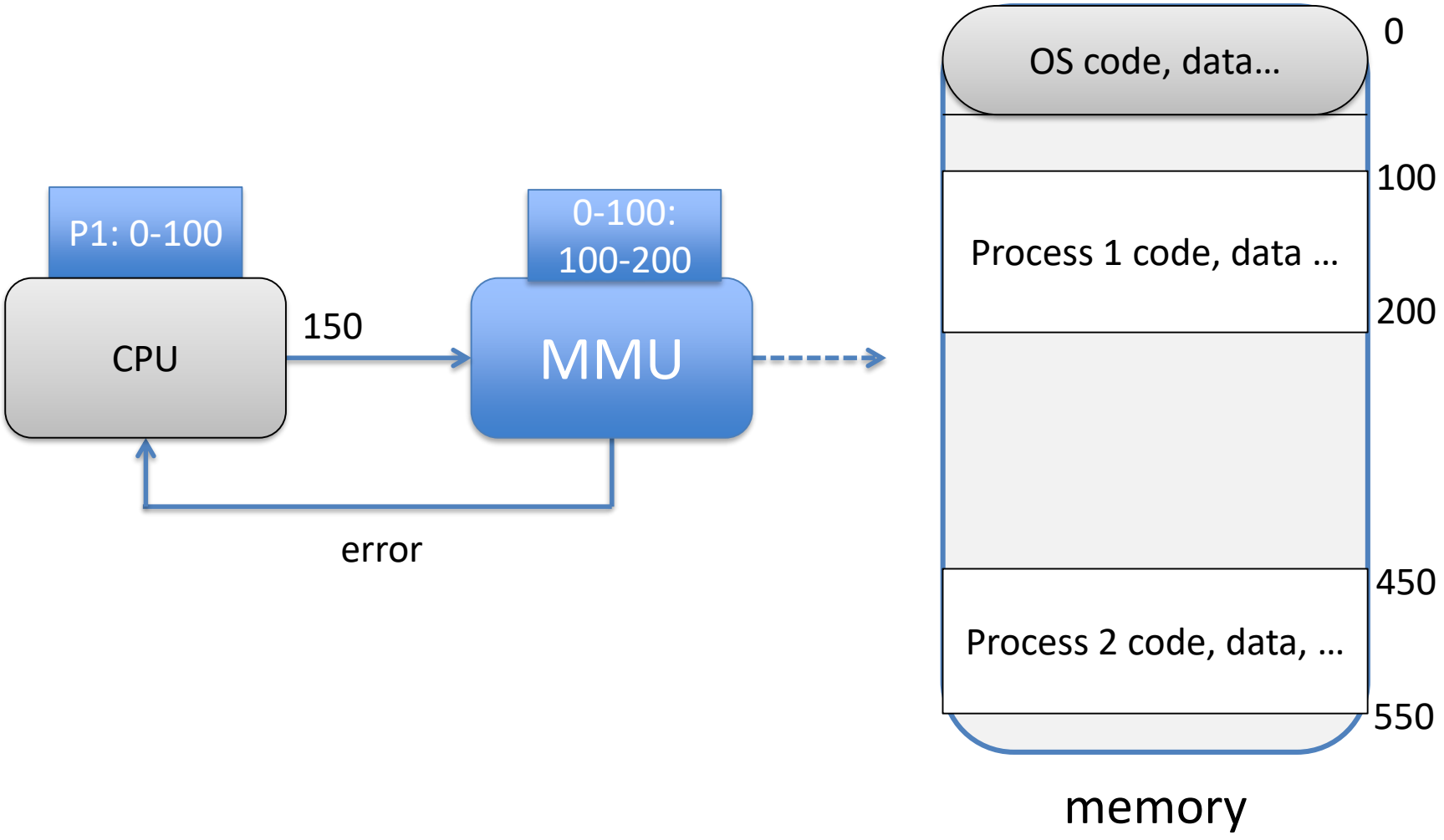
MMU: Virtual to Physical



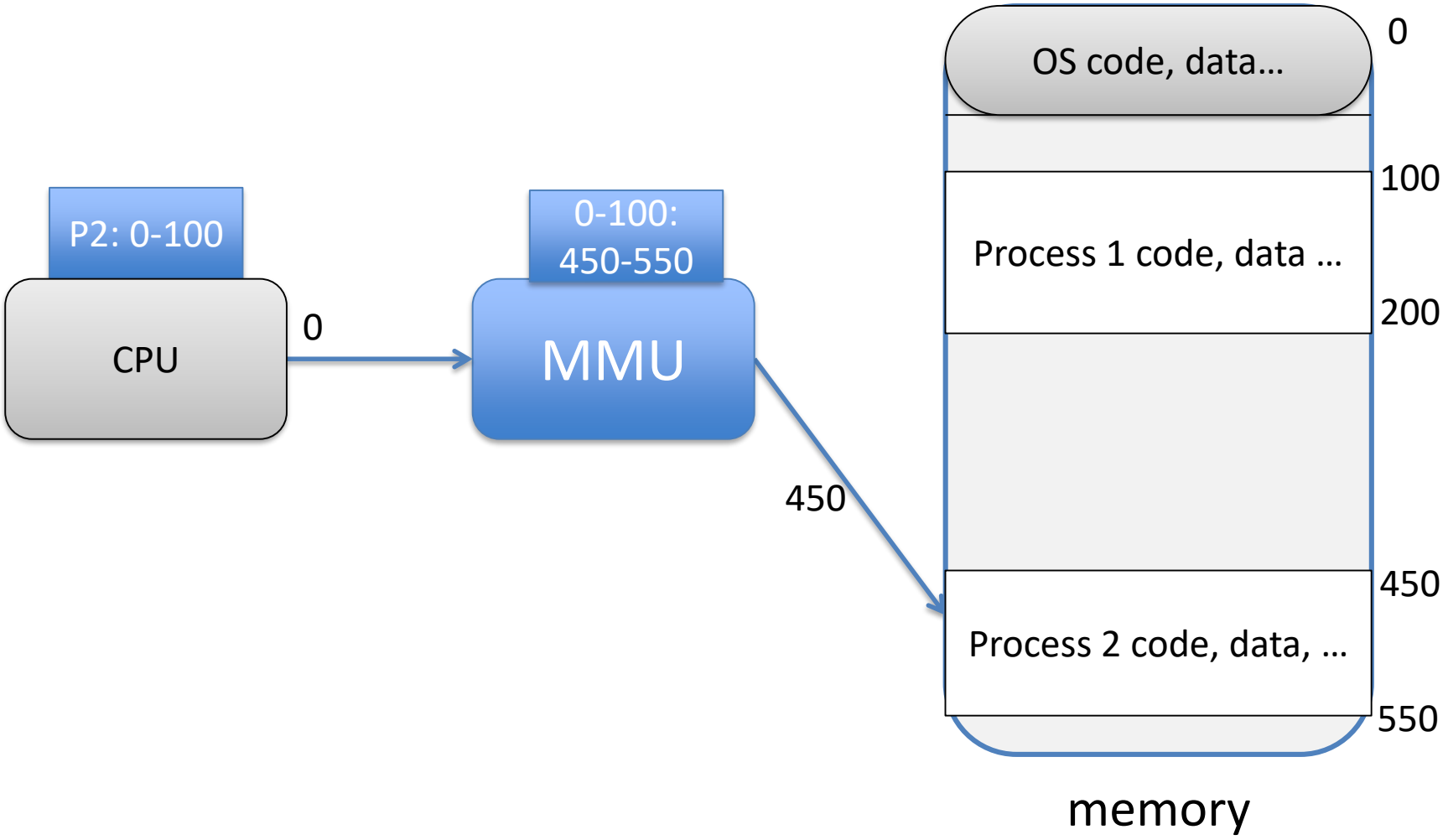
Mapping


P1: 0-100
CPU
50
0-100:
100-200
MMU
150
OS code, data…
0
100
Process 1 code, data …
200
450
Process 2 code, data, …
550
memory

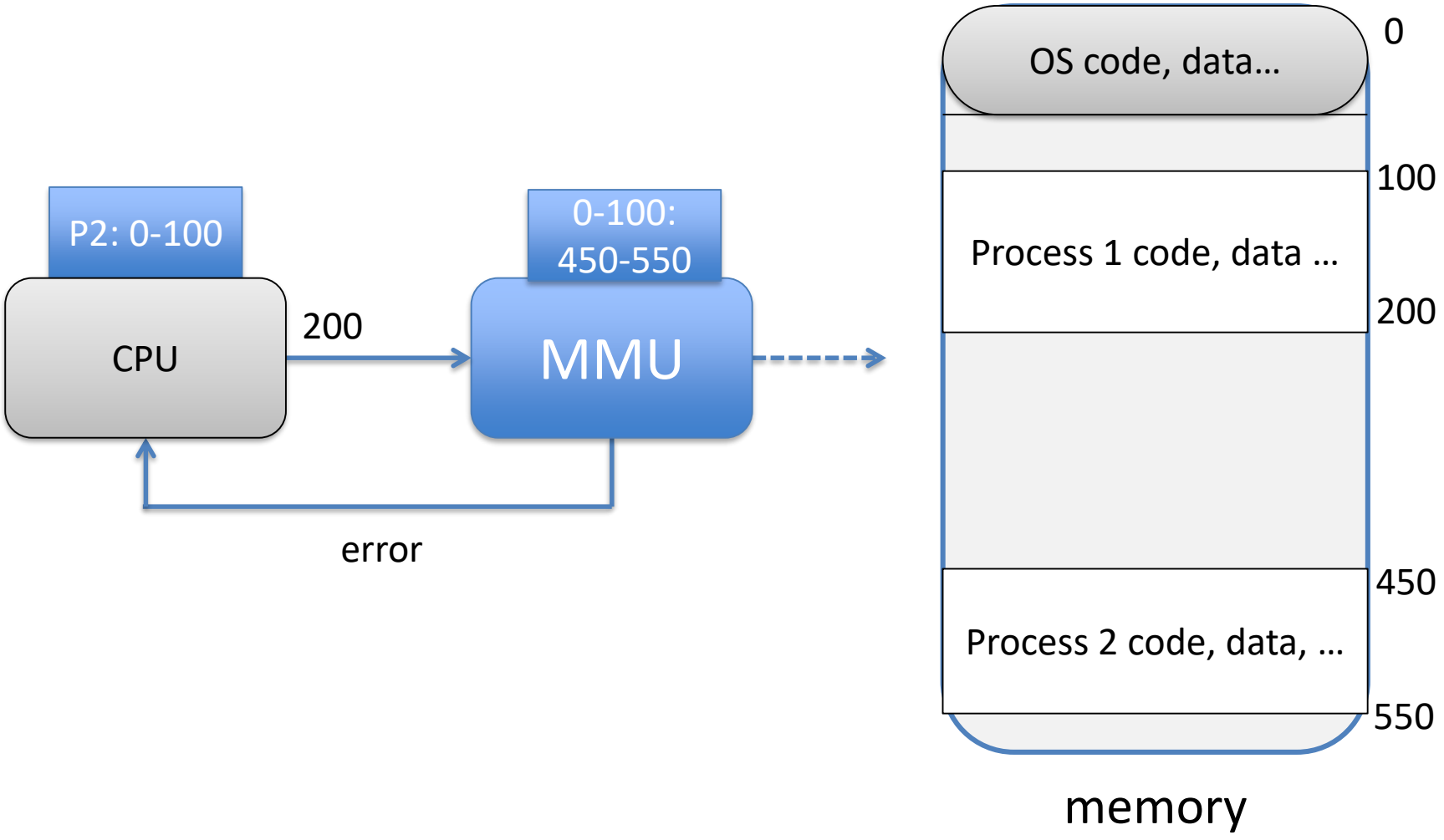
Mapping and Protection



Mapping


P2: 0-100
CPU
0
0-100:
450-550
MMU
450
OS code, data…
Process 1 code, data …
Process 2 code, data, …
0
100
200
450
550
memory

Mapping and Protection



Size of Address Spaces

- Maximum virtual address space size
  - Limited by address size of CPU
  - Typically 32 or 64 bit
  - Hence, 2^32 (4 Gbyte) or 2^64 (16 Exabyte – big!)
- Physical address space size
  - Limited by size of memory
  - Nowadays, order of Gigabytes

Different Virtual/Physical Schemes

1. Base and bounds
2. Segmentation
3. Paging
4. Segmentation with paging

For each Scheme

- Virtual address space
- Physical address space
- Virtual address
- MMU

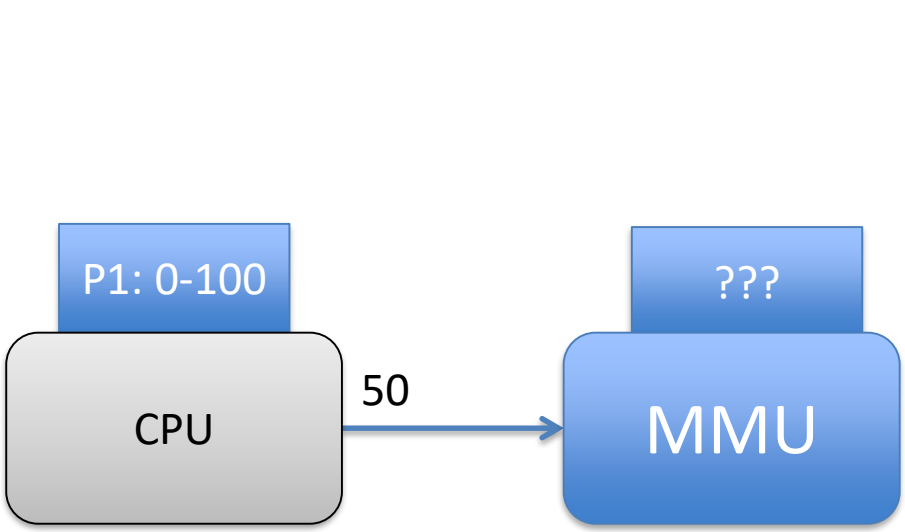
1. Base and Bounds: Virtual Address Space

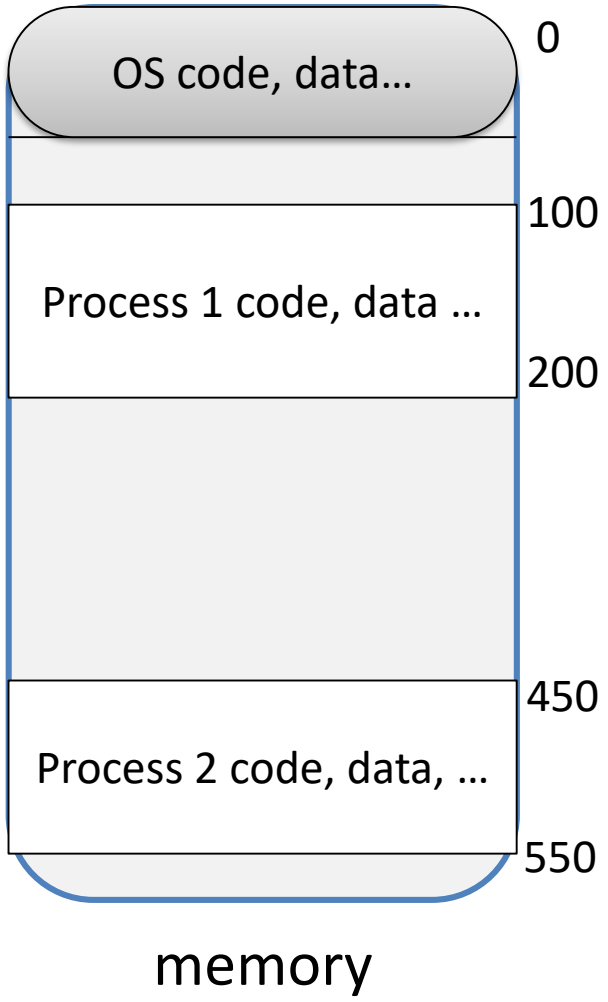
- Linear address space
- From 0 to MAX
- Also called dynamic relocation

1. Base and Bounds: Physical Address Space

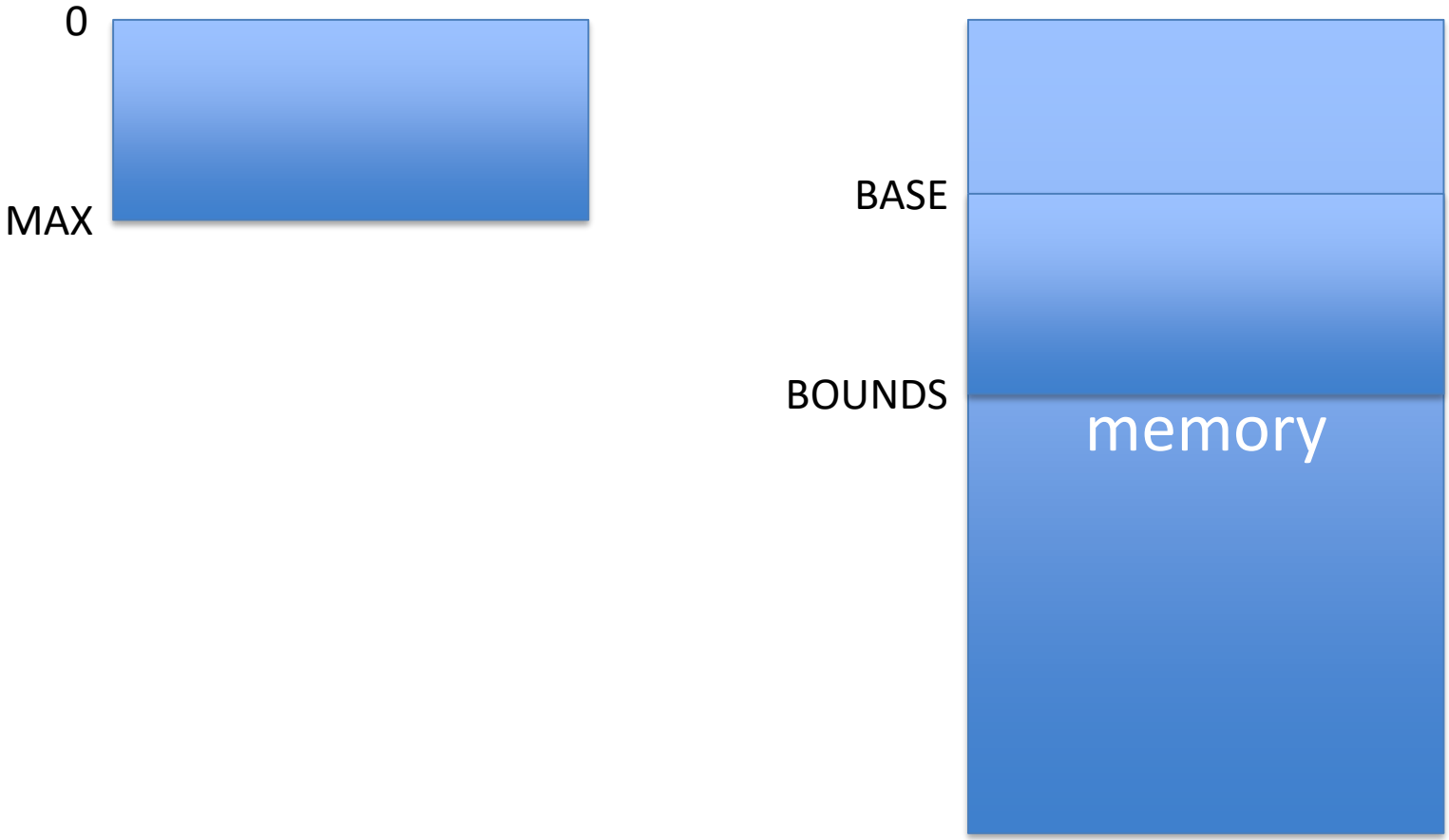
- Linear address space
- From BASE to BOUNDS=BASE+MAX

How would you do it?





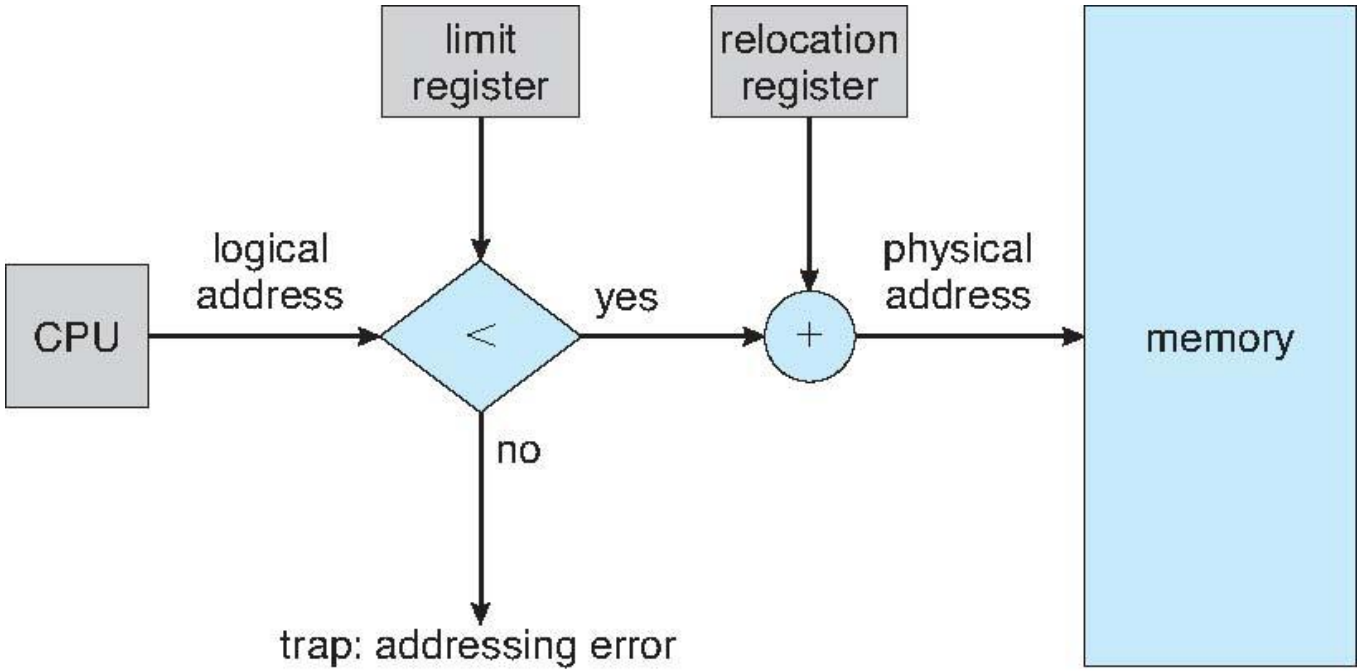
1. Base and Bounds: Virtual and Physical Address Space



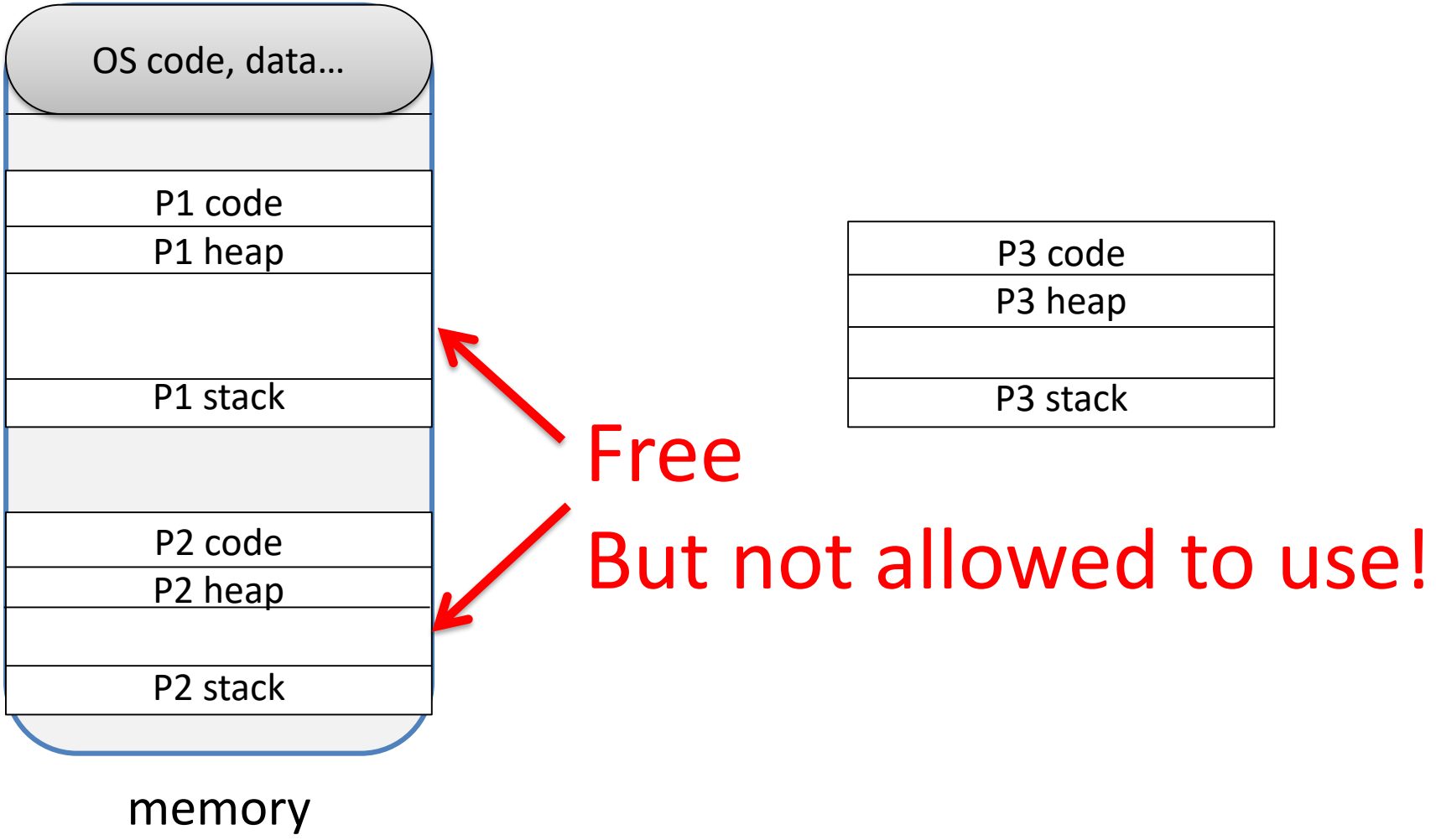
MMU for Base and Bounds

- Relocation register: holds the base value
- Limit register: holds the bounds value

MMU for Base and Bounds



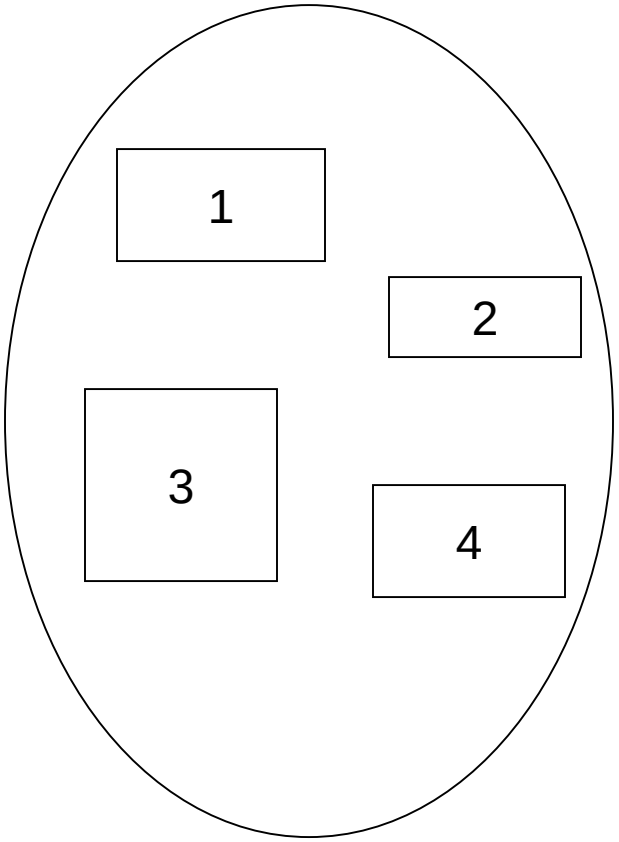
Problem: waste of space



2. Segmentation

- Virtual address space:
  - Two-dimensional
  - Set of segments 0 .. n
  - Each segment i is linear from 0 to MAX_i
- Physical address space
  - Set of segments, each linear

Segmentation: Virtual and Physical Address Space



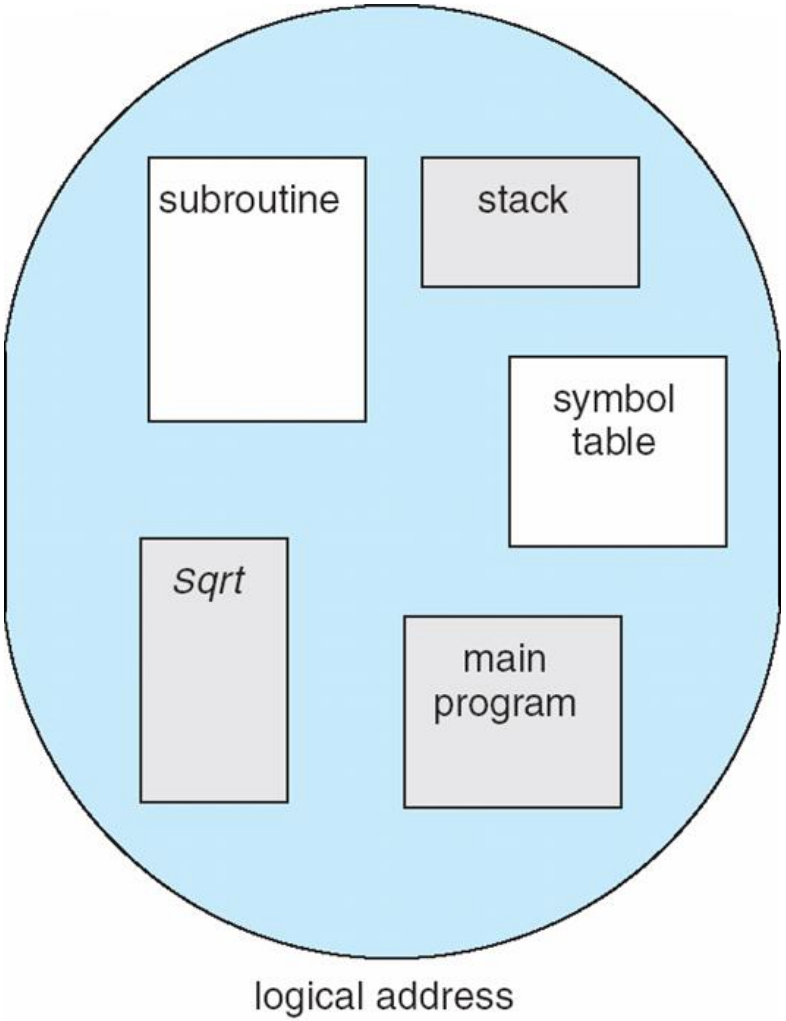
Virtual address space

Physical address space

What is a Segment?

- Anything you want it to be
- Typical examples:
  - Code
  - Heap
  - Stack

Segmentation: Virtual Address Space



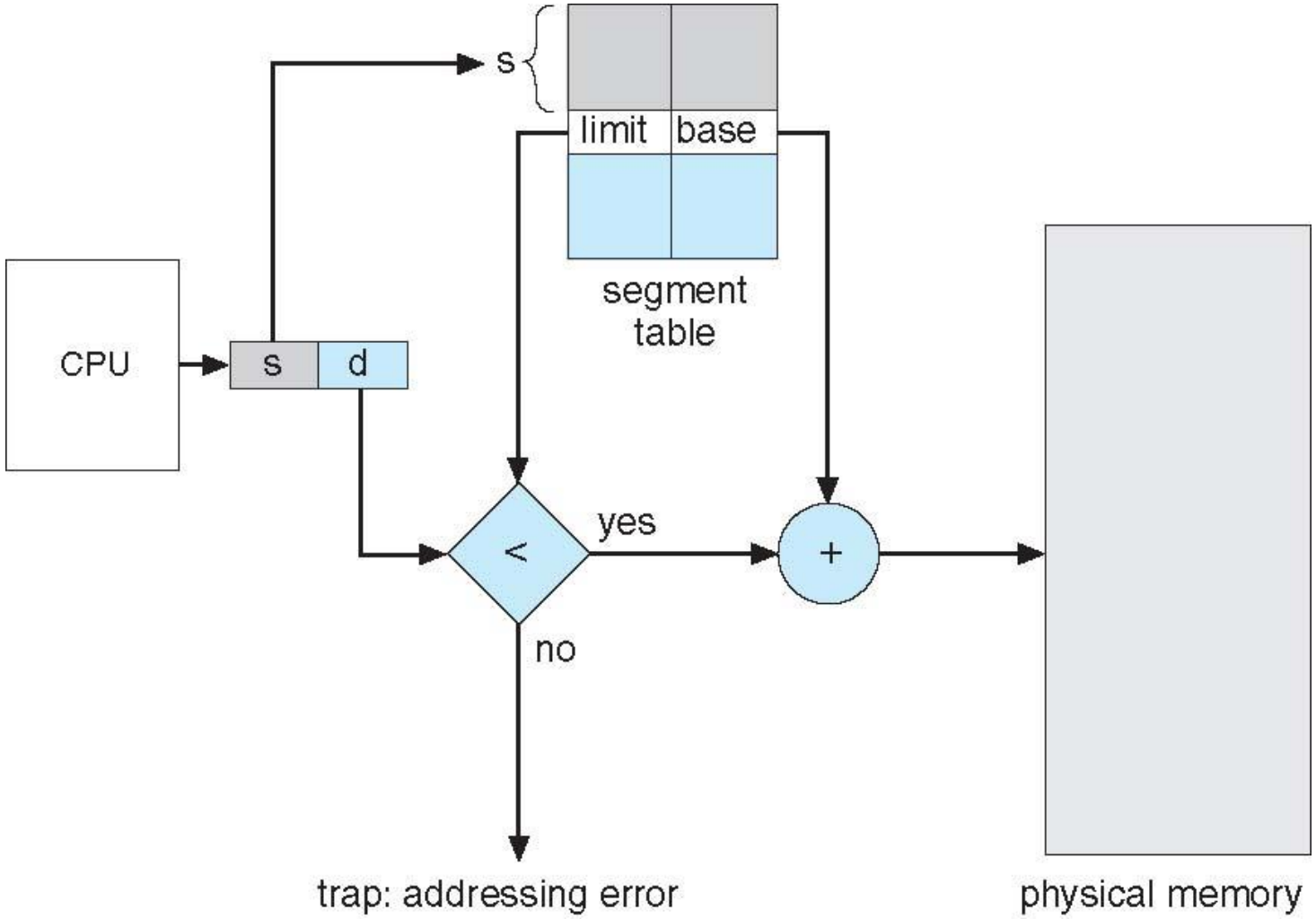
Segmentation: Virtual Address

- Two-dimensional address:
  - Segment number s
  - Offset d within segment (starting at 0)
- It is like multiple base-and-bounds

MMU for Segmentation

- STBR: points to segment table in memory
- STLR: length of segment table
- Segment table
  - Indexed by segment number
  - Contains (base, limit) pair
  - Base: physical address of segment in memory
  - Limit: length of segment

MMU for Segmentation



(check for s ≤ STLR not shown)

Problem: fragmentation

- Fragmentation in disk
  - this happens when swapping
- Variable sized pieces on memory
  - More challenging allocation

3. (Simplified version of) Paging

- Page: fixed-size portion of virtual memory
- Frame: fixed-size portion of physical memory
- Page size = frame size
- Typical size: 4k – 8k (always power of 2)
- Disk: fixed size blocks, same size (or clusters of) frames

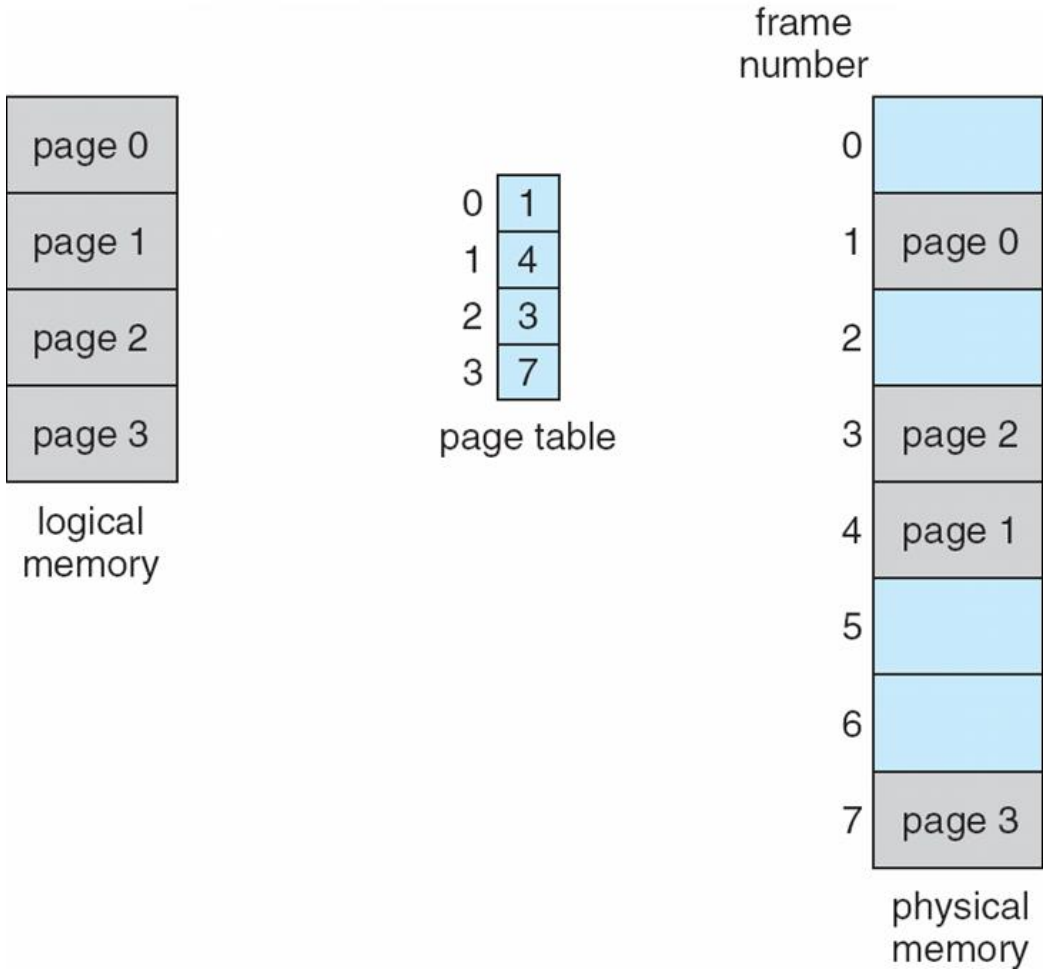
Paging: Virtual Address Space

- Linear from 0 up to a multiple of page size

Paging: Physical Address Space

- Noncontiguous set of frames, one per page

Paging:
Virtual and Physical Address Space



Paging Virtual Address

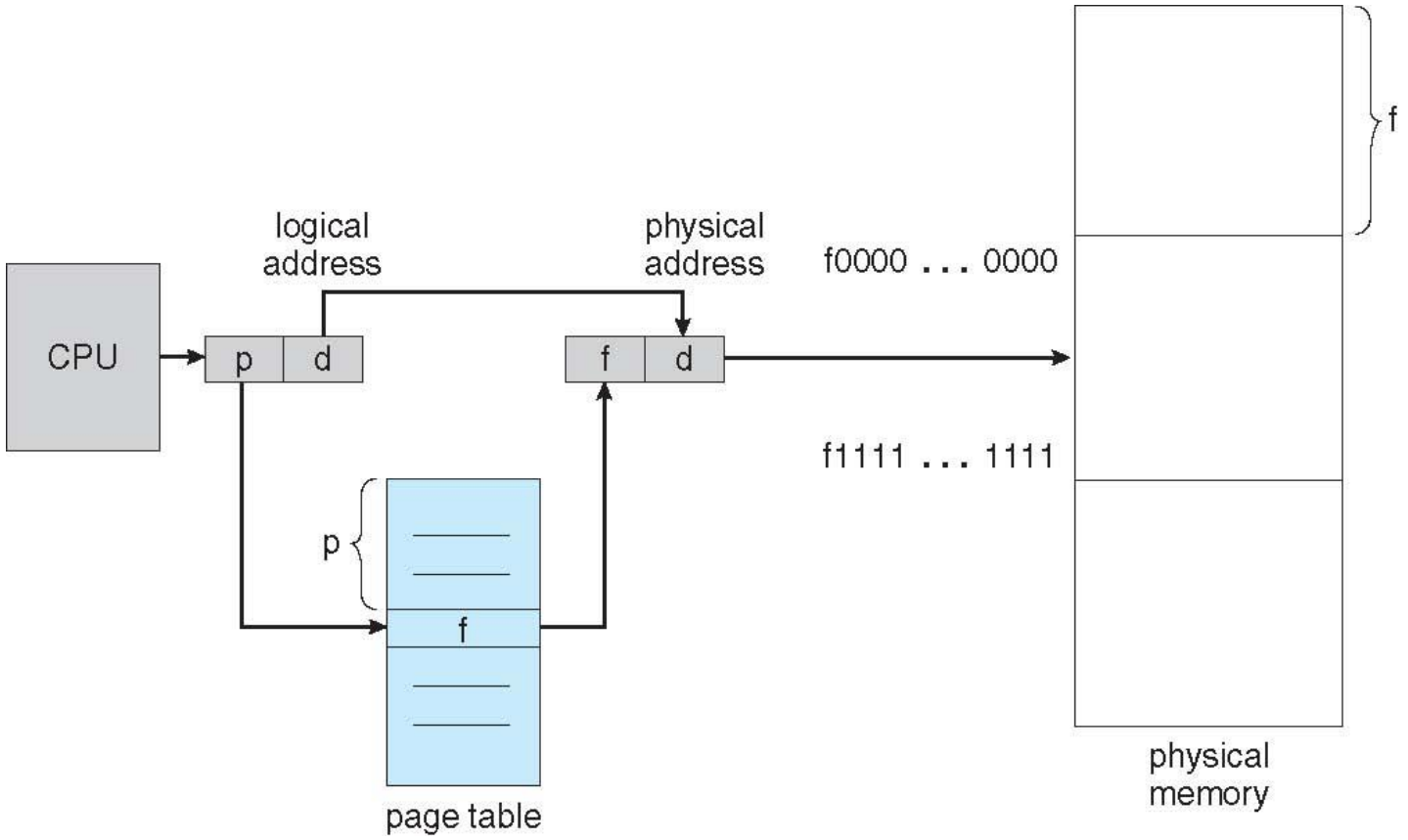
- Virtual address: 0 – MAX (page size multiple)

- Page size = 2^n
- Virtual address for mapping purposes:
  - Page number p: first sequence of bits
  - Offset within page d: n remaining bits

MMU for Paging

- PTBR: points to page table in memory
- PTLR: length of page table
- Page table
  - Indexed by page number
  - Contains frame number of page in memory

MMU for Paging



(check for p ≤ PTLR not shown)

4. Segmentation with Paging

- As segmentation
- But every segment is paged

Segmentation with Paging

- Virtual address space
  - Two-dimensional
  - Set of segments 0 .. n
  - Each segment linear from 0 to MAX_i
- Physical address space
  - Noncontiguous set of frames

Segmentation with Paging: Virtual Address

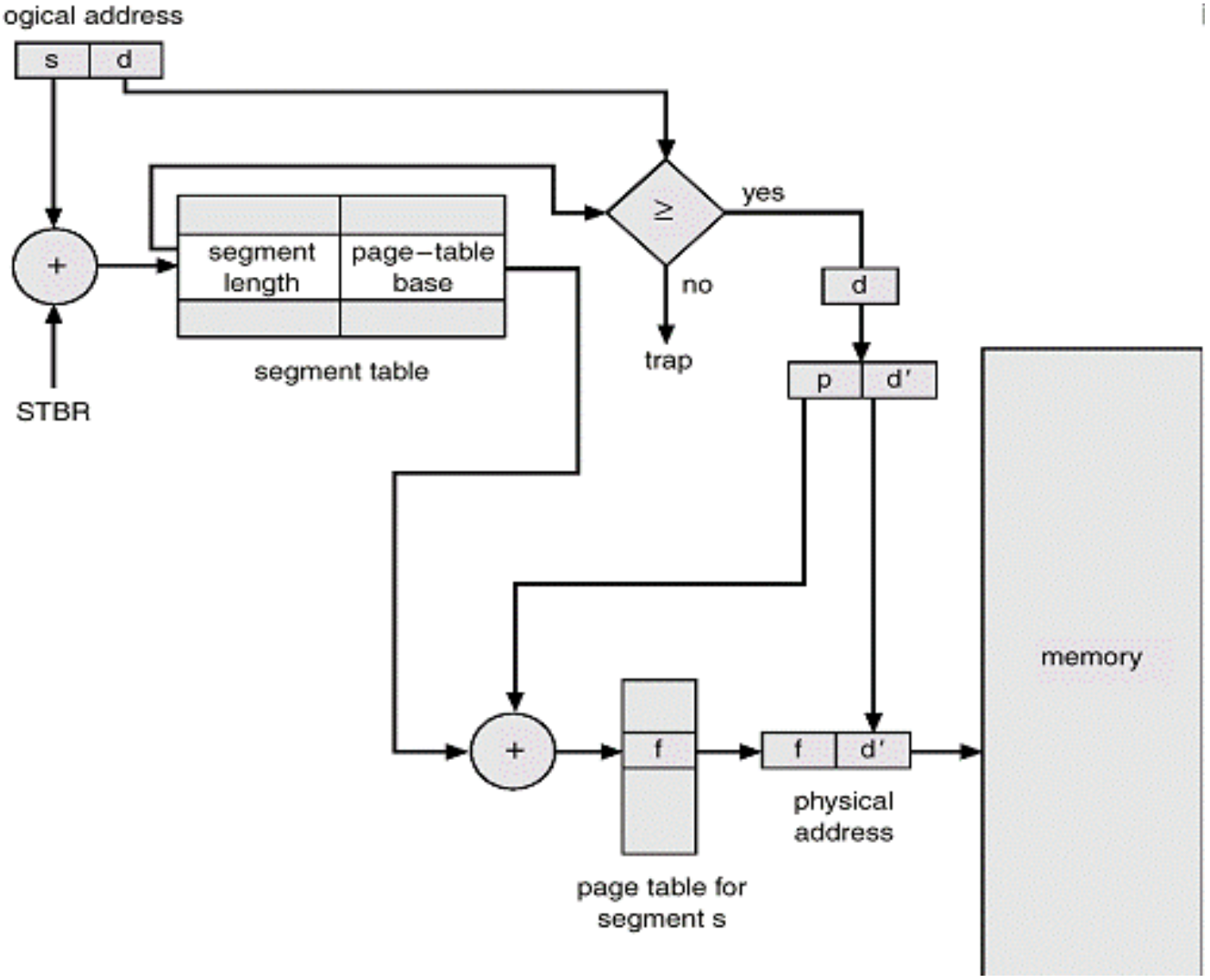
- Virtual address:
  - Segment number s
  - Offset with segment d

- Virtual address for mapping:
  - Segment number s
  - Page number within segment p
  - Offset within page d’

MMU for Segmentation with Paging

- STBR: points to segment table in memory
- STLR: length of segment table
- Segment table:
  - Indexed by segment number
  - Contains page table base, segment length
- Page table for each segment:
  - Index by page number
  - Contains frameno of frame that contains page

MMU for Segmentation with Paging

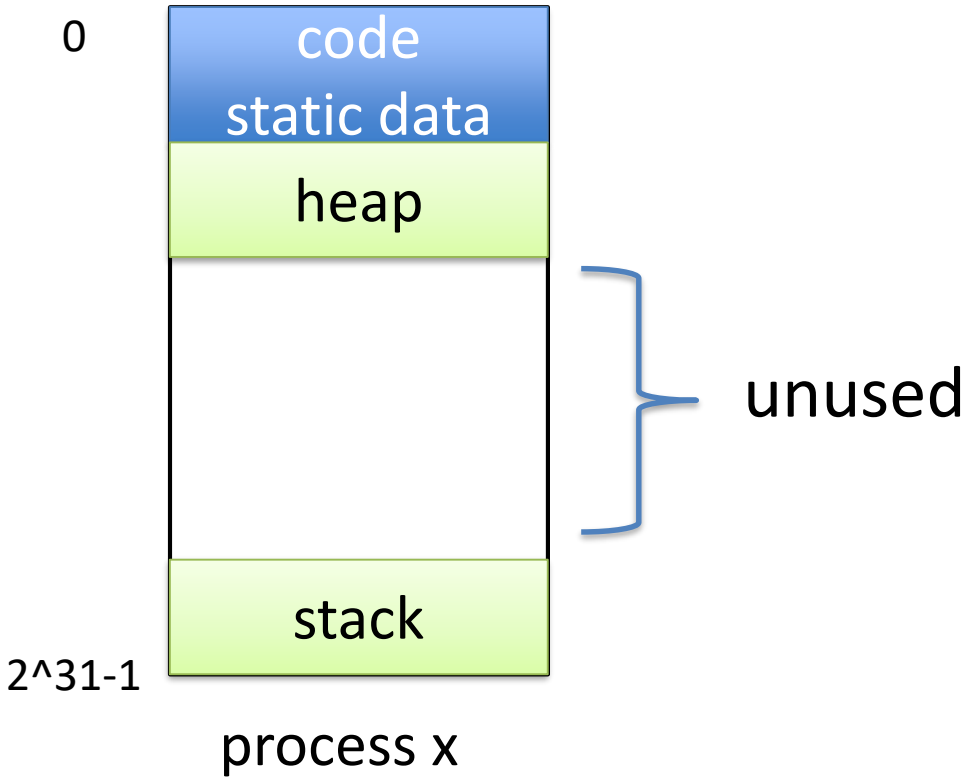


(check for s ≤ STLR not shown)

Revisit Paging: Virtual Address Space

- Linear from 0 up to a multiple of page size
- True, but address space is often sparsely used

Typical Virtual Address Space



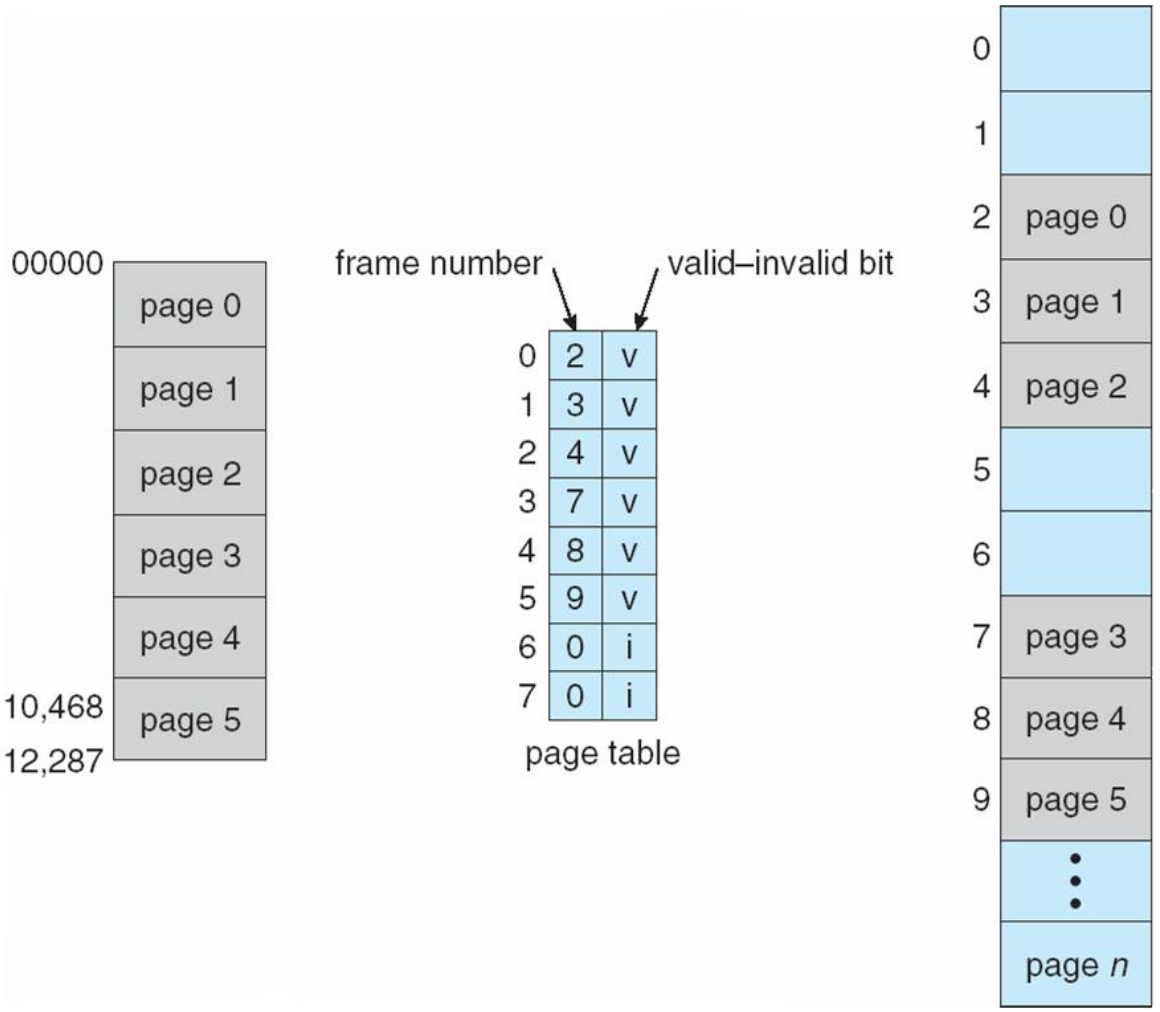
Problem?

- Access to unused portion will appear valid
- Would rather have an error

Solution

- Abandon PTLR
- Page table has length 2^p
- Instead, have valid bit in each PTE
  - Set to valid for used portions of address space
  - Invalid for unused portions
- This is the common approach

Valid (v) or Invalid (i) Bit In A Page Table



Back to Main Memory Allocation

- We already looked at where to locate kernel
- Answer: in low memory
- Now looking at processes

- As many processes in memory as possible
- Why? Quick switch on I/O of running process

Main Memory Allocation

- How find memory for a newly arrived process?

Main Memory Allocation Base-and-Bounds

- Main memory:
  - Regions in use
  - “Holes”
- New process needs to go in “hole”
- Which hole to pick?

Dynamic Memory Allocation Methods

- First-fit
  - Take first hole bigger than requested
  - Easy to find
- Best-fit
  - Take smallest hole bigger than requested
  - Leaves smallest hole behind
- Worst-fit?!
  - Takes largest hole
  - Leaves biggest hole behind

(External) Fragmentation

- Small holes become unusable
- Part of memory cannot be used
- Serious problem

Main Memory Allocation with Segmentation

- Remember
  - Segmentation ~= multiple base-and-bounds
- Problem is similar
  - Dynamic memory allocation
  - Pieces are typically smaller
  - But there are more (than 1) pieces
- Easier problem
  - External fragmentation smaller

Main Memory Allocation with Paging

- Logical address space: fixed size pages
- Physical address space: fixed size frames
- New process:
  - Find frames for all of process's pages
- Easier problem
  - Fixed size

(Internal) Fragmentation

- With paging
  - Address space = multiple of page size
- Part of last page may be unused
- With reasonable page size, not a big problem

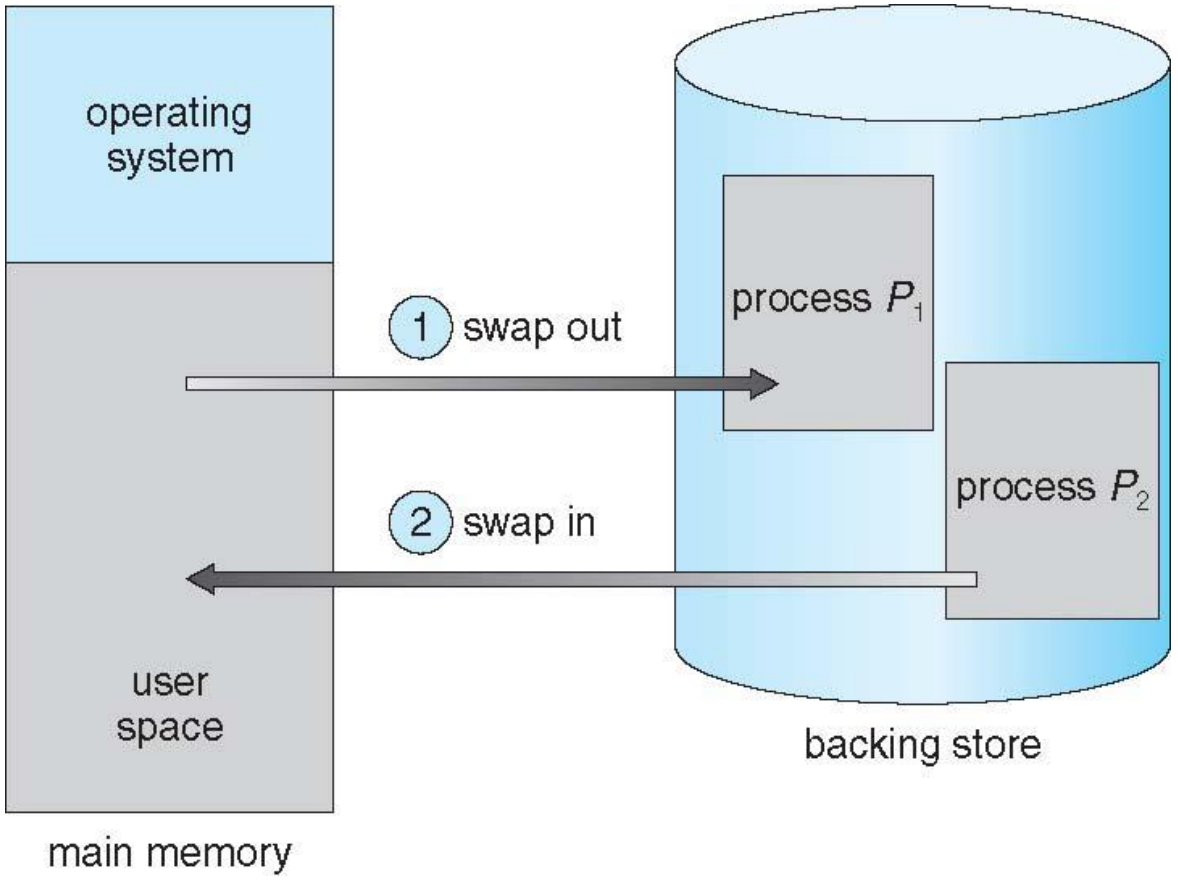
Main Memory Allocation

- Easier to do with paging

What if “out of memory”?

- Need to get rid of one or more processes
- Store them temporarily on disk
- This is called swapping

Schematic View of Swapping



Process Switch to a Swapped Process?

- Latency can be very high
- Need to read image from disk

- A better solution:
  - Demand paging
  - Not all of a process needs to be in memory
- Topic for next lecture

Finer-Grain Protection

- Means: different protections for different parts of address space

- Valid bit in page table
- May also have valid bit in segment table
- May also have other bits both in page/segment table
  - Read-only / read-write
  - Executable / not-executable

Finer-Grain Protection

- For instance, code should be
  - Valid, read-only and executable
- Base and bounds
  - Not really possible
- Segmentation
  - Set those bits in segment table
- Paging
  - Set those bits in every code page

Finer-Grain Protection Summary

- Easier to do with segmentation

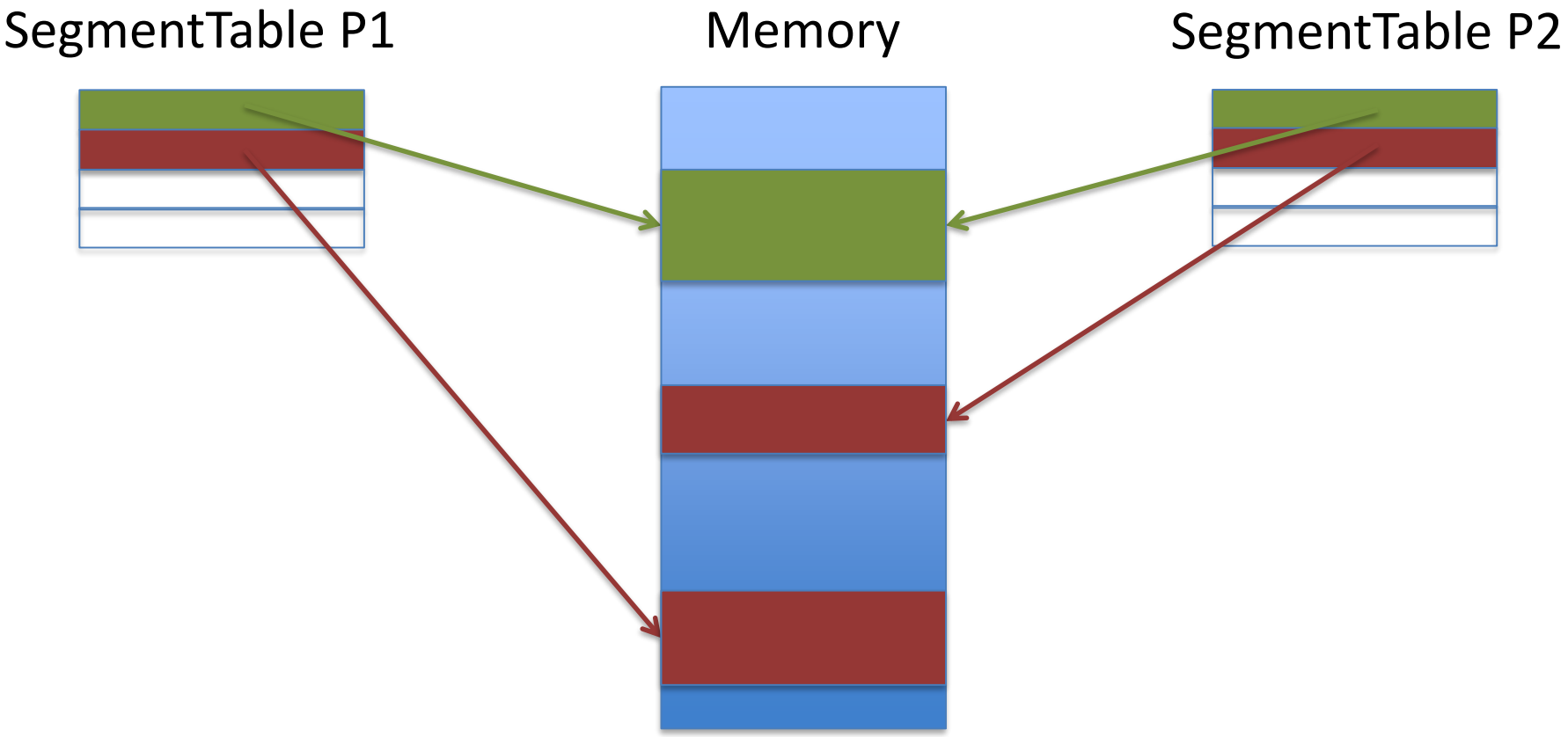
Sharing Memory between Processes

- Why would we want to do that?
- For instance,
  - Run twice the same program in different processes
  - May want to share code
  - Read twice the same file in different processes
  - May want to share memory corresponding to file

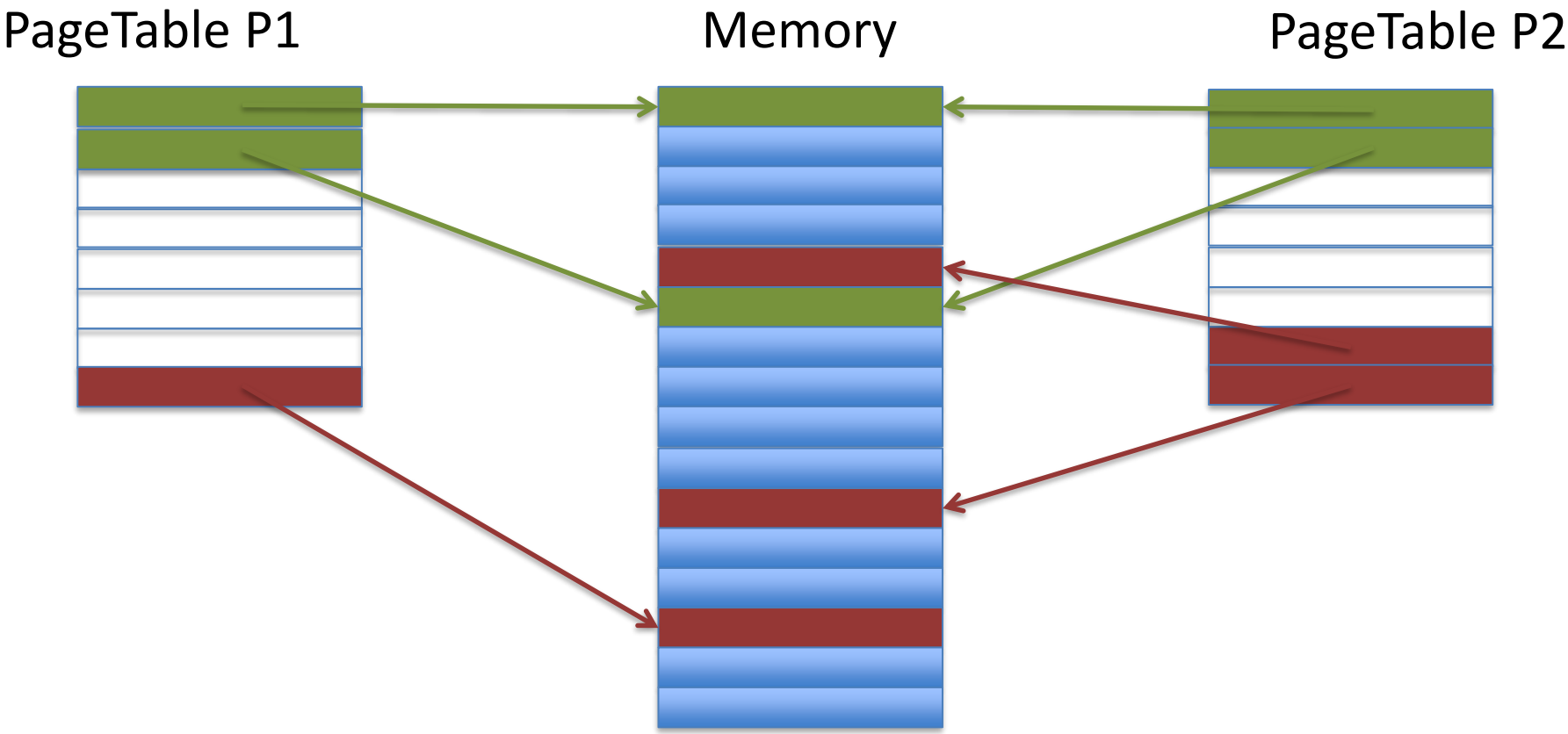
How to Share Memory?

- With base and bounds, not possible
- With segmentation
  - Create segment for shared data
  - Entry in segment table of both processes
  - Points to shared segment in memory
- With paging
  - Need to share pages
  - Entries in page table of both processes
  - Point to shared pages

P1 and P2 Share One Segment



P1 and P2 Share Two Pages



Sharing - Summary

- Sharing is easier to do with segmentation

Advantages / Disadvantages

	Segmentation	Paging	Segmentation with Paging
Sharing	☑		☑
Fine-grain protection	☑		☑
Memory allocation		☑	☑

If you wondered, why segmentation with paging?
Answer: it combines advantages of both.

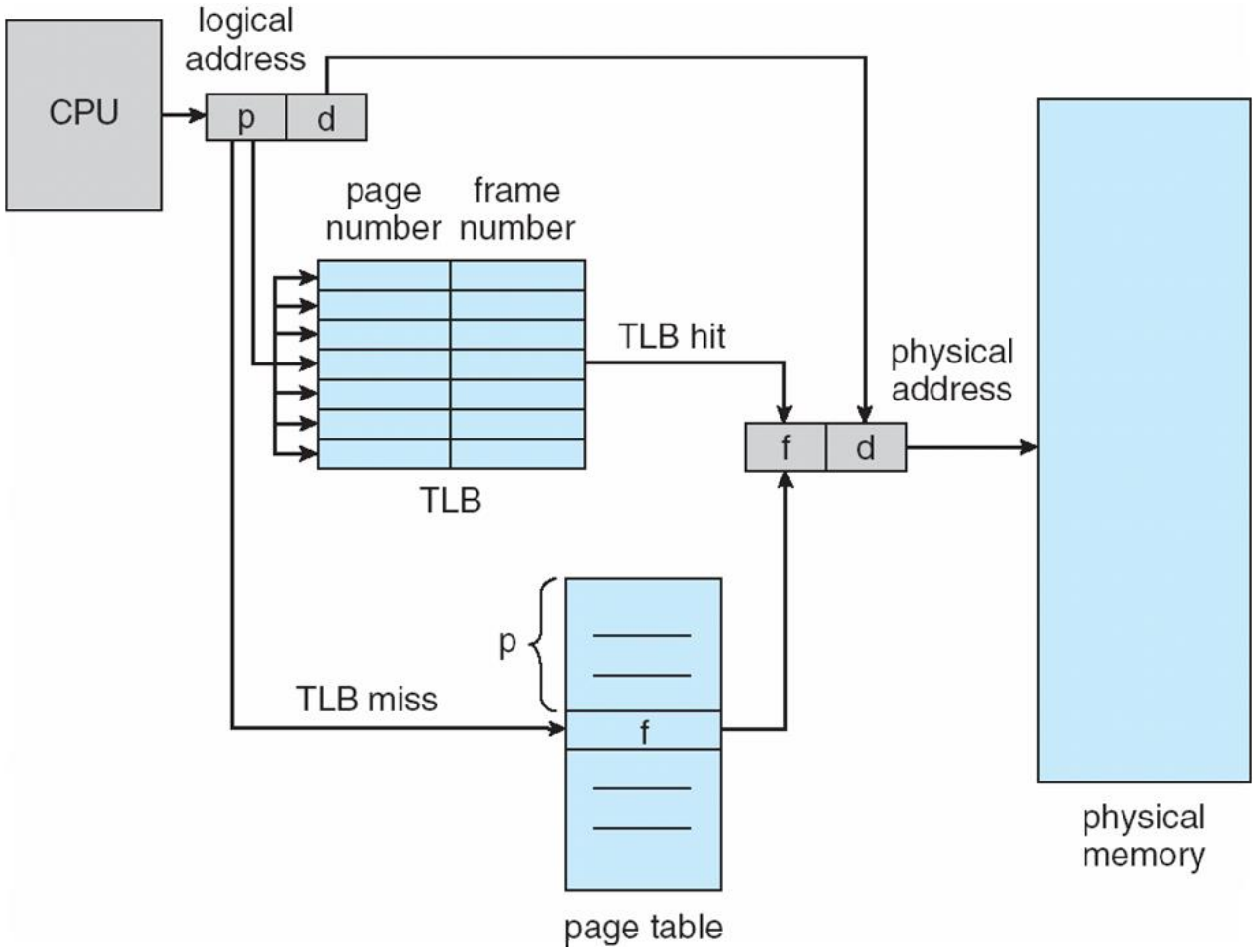
Address Translation Performance Issue

- Page table is in memory
- 1 virtual address → 2 physical memory accesses
- *Would reduce performance by factor of 2*
- Solution: Translation lookaside buffer (TLB)

TLB

- Small fast cache of (pageno, frameno) maps
- If mapping for pageno found in TLB
  - Use frameno from TLB
  - Abort mapping using page table
- If not
  - Perform mapping using page table
  - Insert (pageno, frameno) in TLB

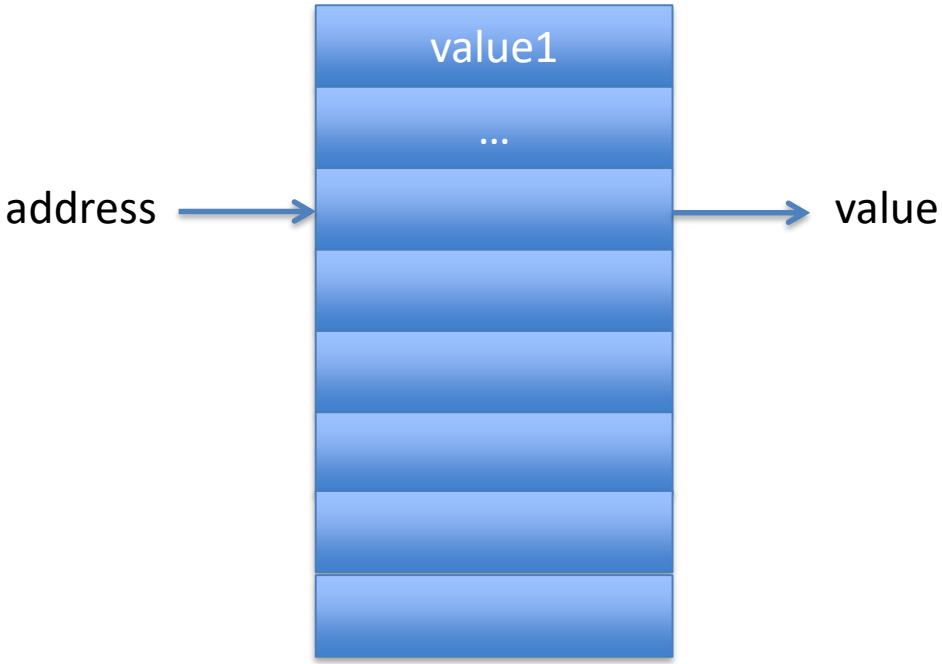
Paging Mapping With TLB



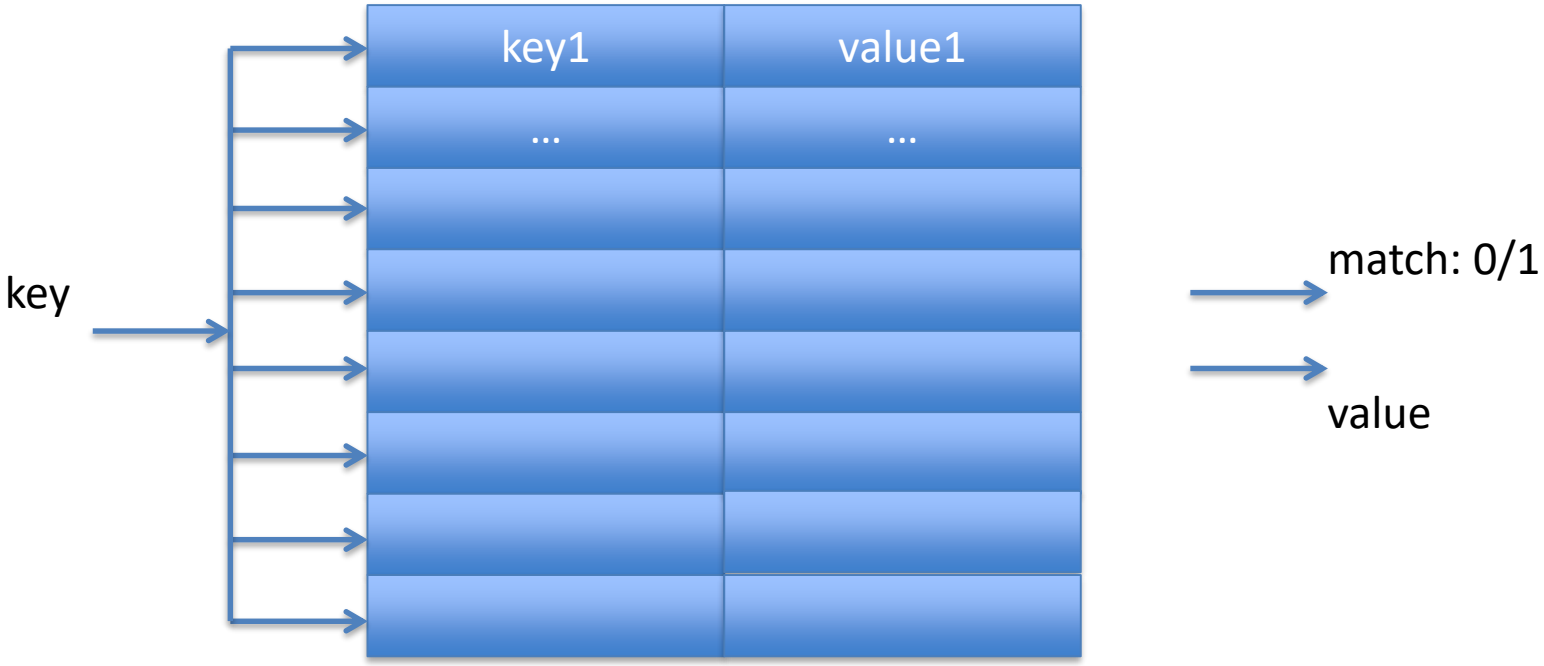
How to Make TLB fast?

- Use associative memory (special hardware)
- Regular memory
  - Look up by address
- Associative memory
  - Look up by contents
  - Search entire memory in parallel

Regular Memory



Associative Memory



TLB

TLB Size

- Associative memory is very expensive
- Therefore, TLB small (64 – 1,024 entries)
- If TLB full, need to replace existing entry

Summary

- Virtual and physical address spaces
- Mapping between virtual and physical address
- Different mapping methods
- Sharing, protection, memory allocation
- TLB

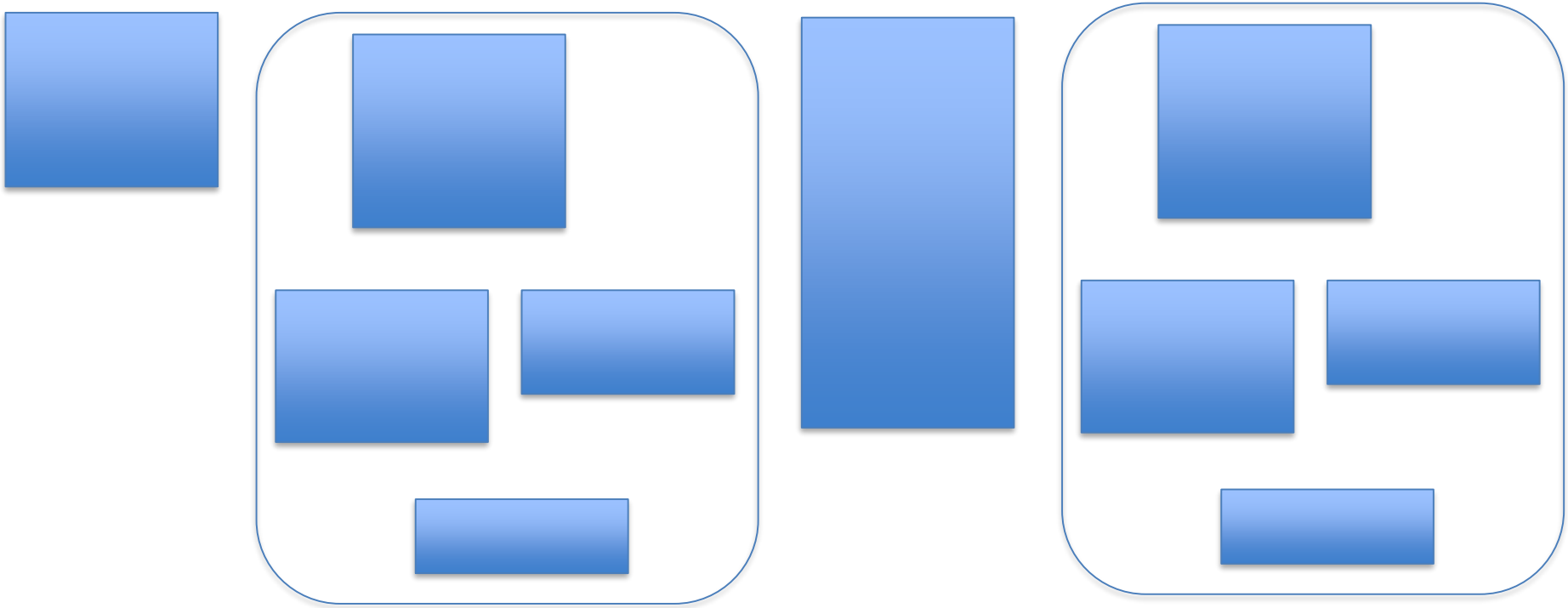
Virtual Address Space

Base and Bounds

Segmentation

Paging

Segmentation with Paging



Virtual Address Format

- Base and bounds

- Segmentation

- Paging

- Segmentation with Paging

Virtual Address Validity

- Base and bounds
  - Upto MAX
- Segmentation
  - Each segment upto MAX_i
- Paging
  - According to PTLR or valid bits
- Segmentation with paging
  - Each segment upto MAX_i (now multiple of page size)

Breakdown of Virtual Address for Mapping

- Base and bounds

- Segmentation

- Paging

- Segmentation with Paging

Physical Address Space