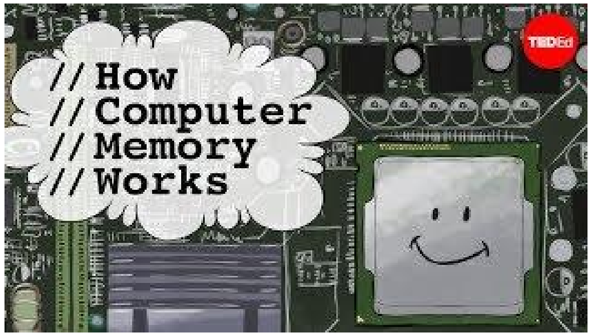


CSc 360 Operating Systems **Memory Management**

Wenjun Yang

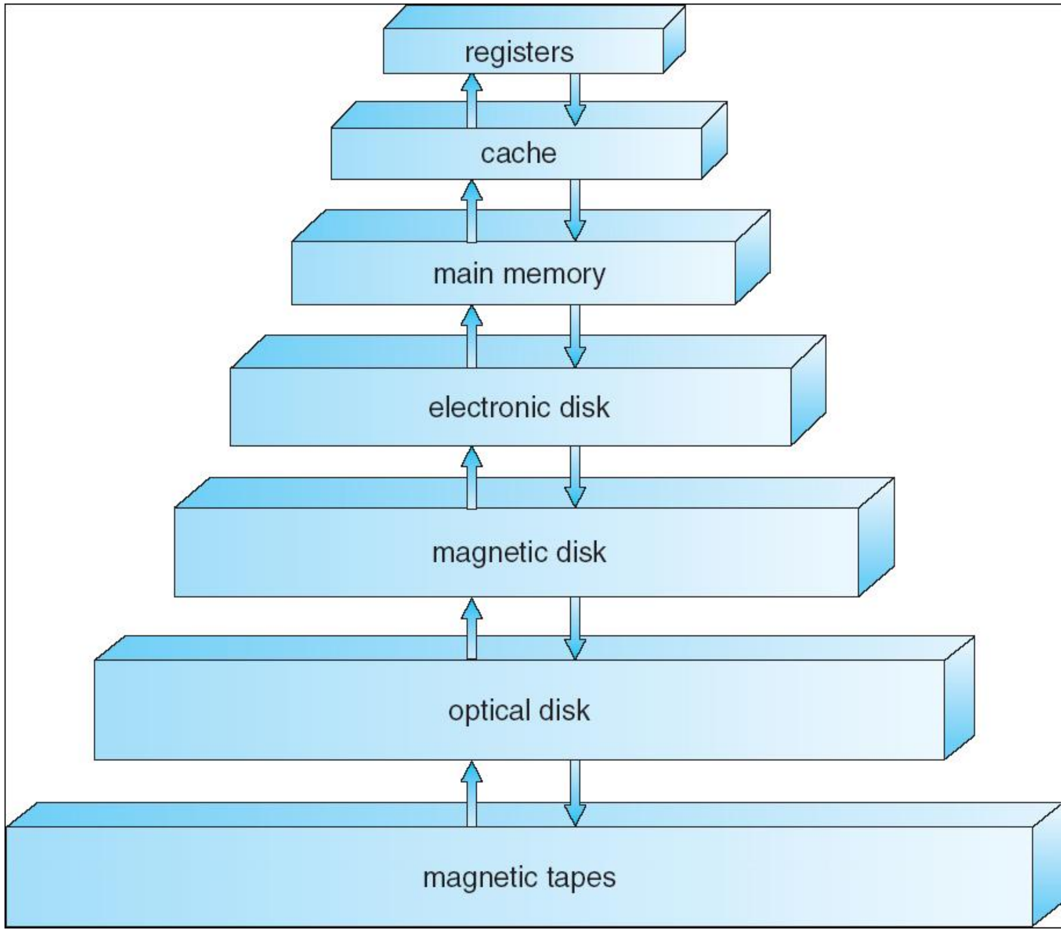
Fall 2025

Review

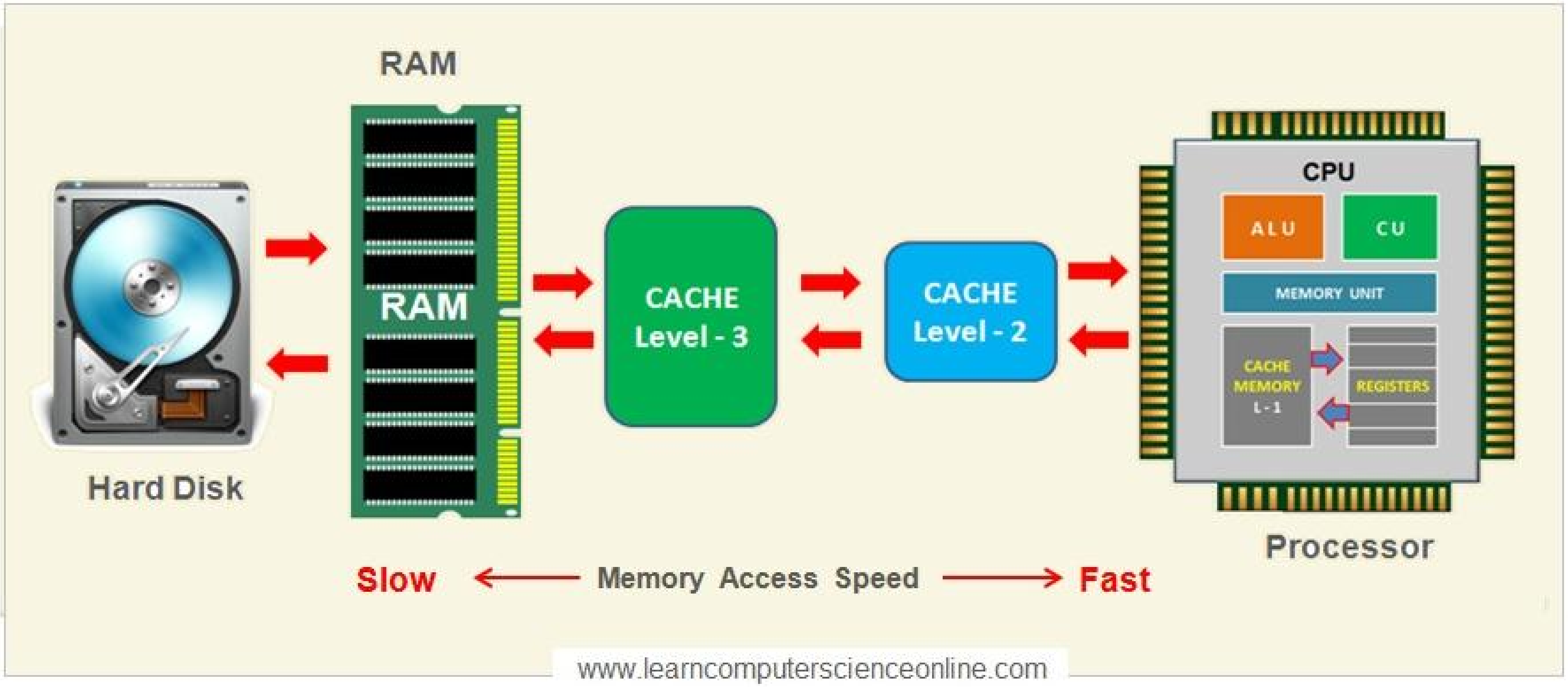
- CPU management: how to *share* CPU
  – scheduling algorithms
    - jobs, processes, threads
  – communication mechanisms
    - shared memory, message passing, etc
  – synchronization algorithms
    - mutual exclusion, deadlocks, livelocks
- Next: how to *share* memory

Storage hierarchy

- CPU direct access
  - registers
  - cache
  - (main) memory

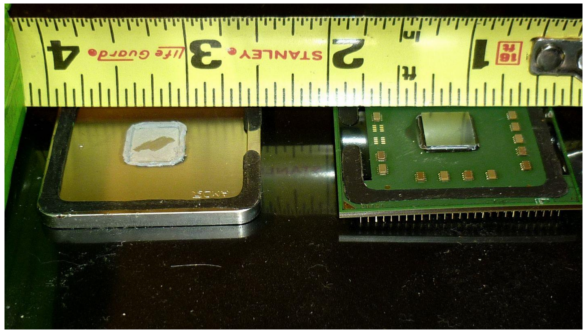


Storage hierarchy

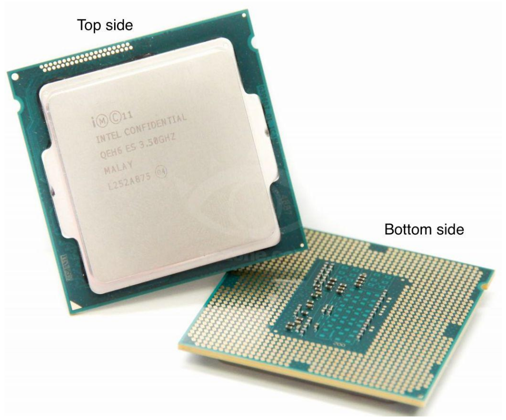


Caches

CPU Core



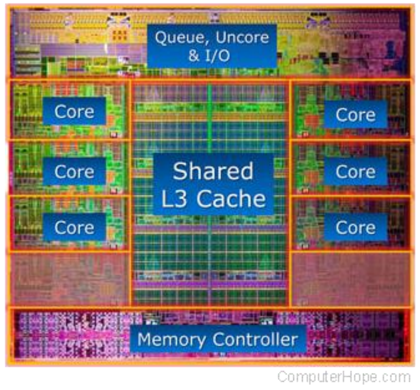
CPU Die: a single continuous piece of semiconductor material (usually silicon)



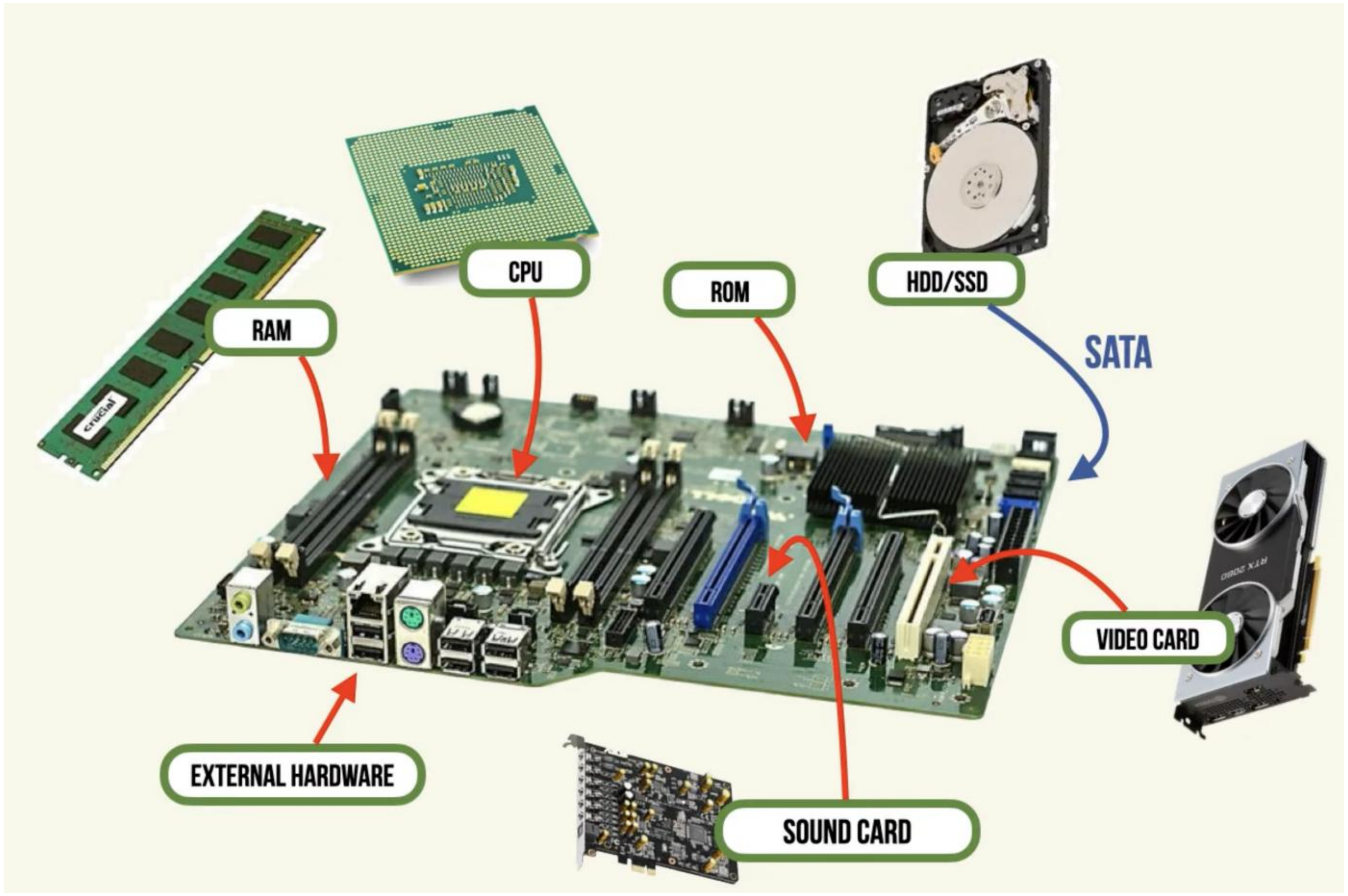
CPU package:

- A piece of board, containing pins or contacts on the bottom to make contact with a motherboard socket.
- A top shell, made of metal, sometimes ceramic, that protects the CPU die from physical damage.

Intel Core i7-3960X Processor Die Detail

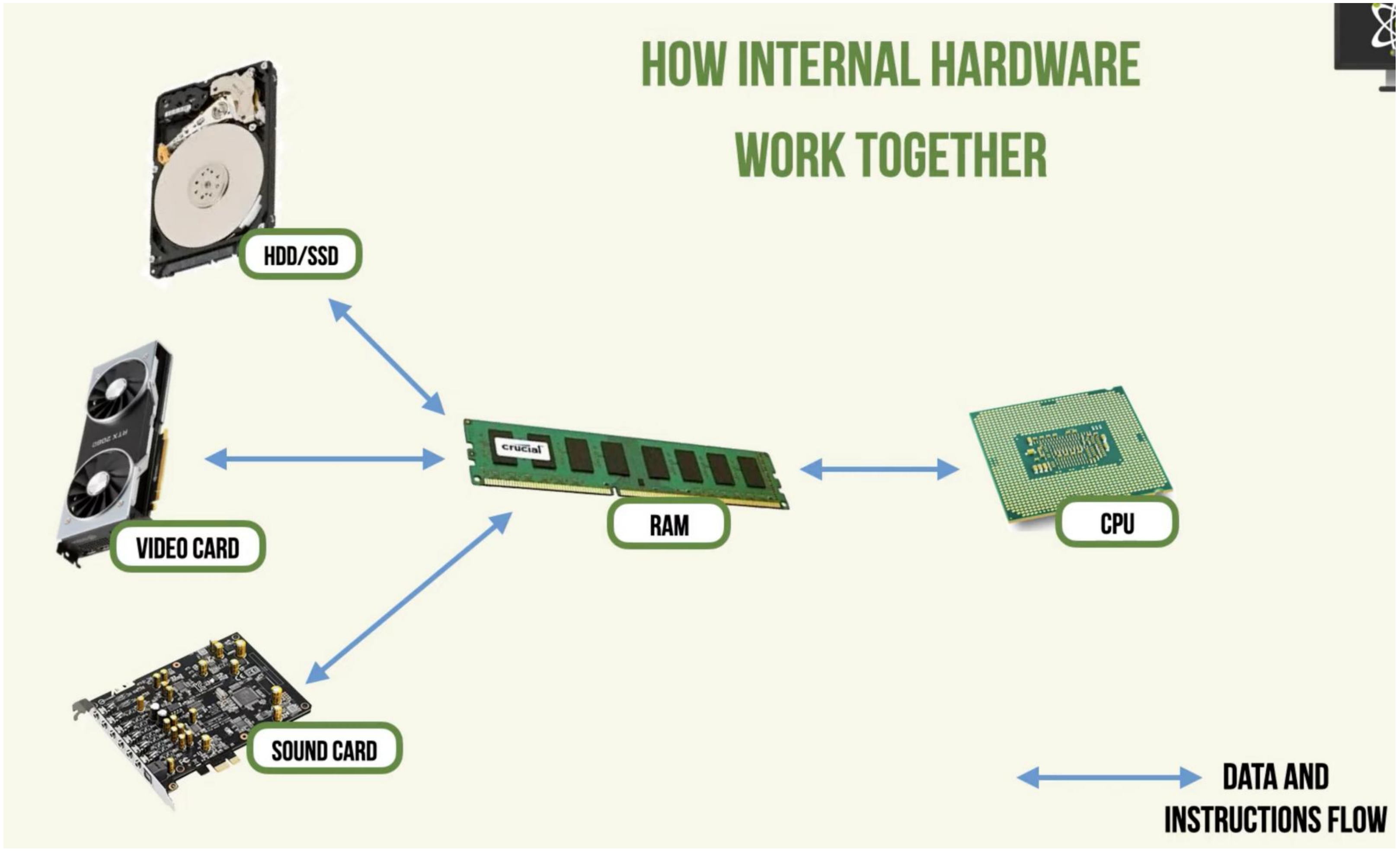


Motherboard



https://www.youtube.com/watch?v=E_OA08Naaas

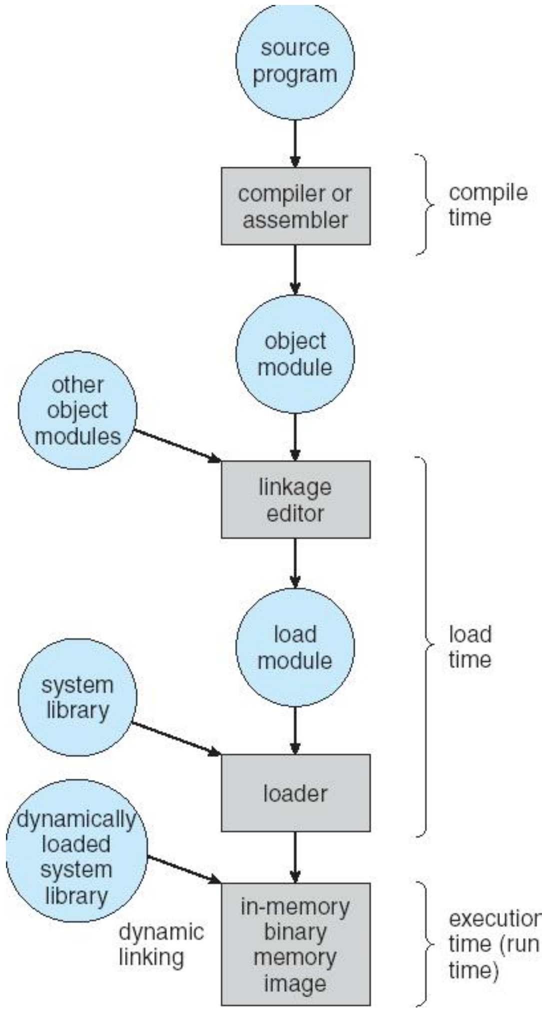
Motherboard


HOW INTERNAL HARDWARE WORK TOGETHER
HDD/SSD
VIDEO CARD
RAM
CPU
SOUND CARD
DATA AND INSTRUCTIONS FLOW

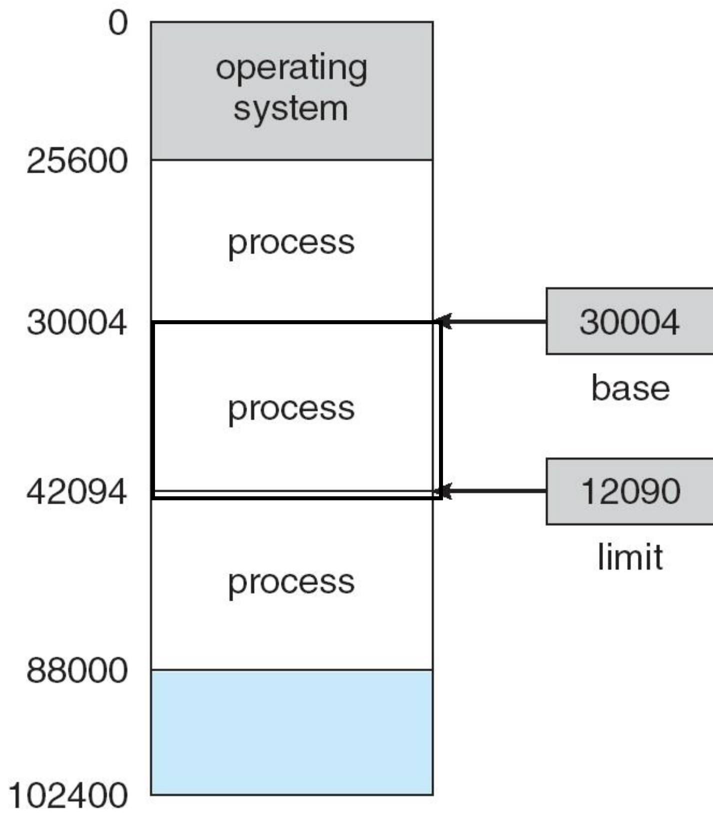
https://www.youtube.com/watch?v=E_OA08Naaas

Memory access

- Access by address
  - for both code and data
- Address binding
  - compiler time: absolute code
    - MS-DOS .COM format, 64KB limit
  - load time: relocatable code
    - MS-DOS .EXE format
  - execution time



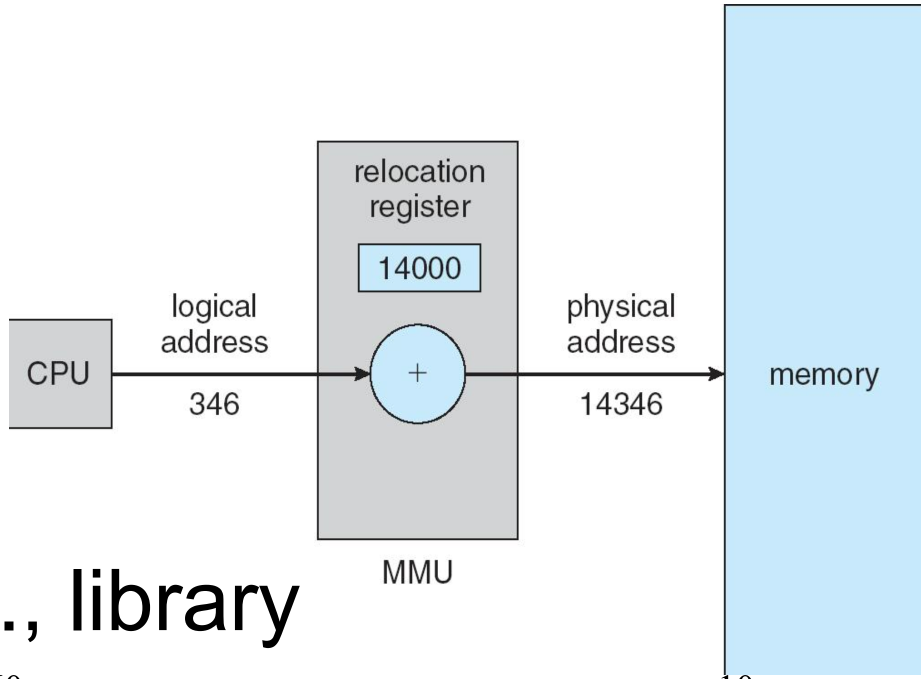
Memory space



- Logical memory
  - seen by CPU
  - virtual memory
- Physical memory
  - seen by memory unit
- Address binding at
  - compile/load time: logical/physical addr same
  - execution time: logical/physical address differ

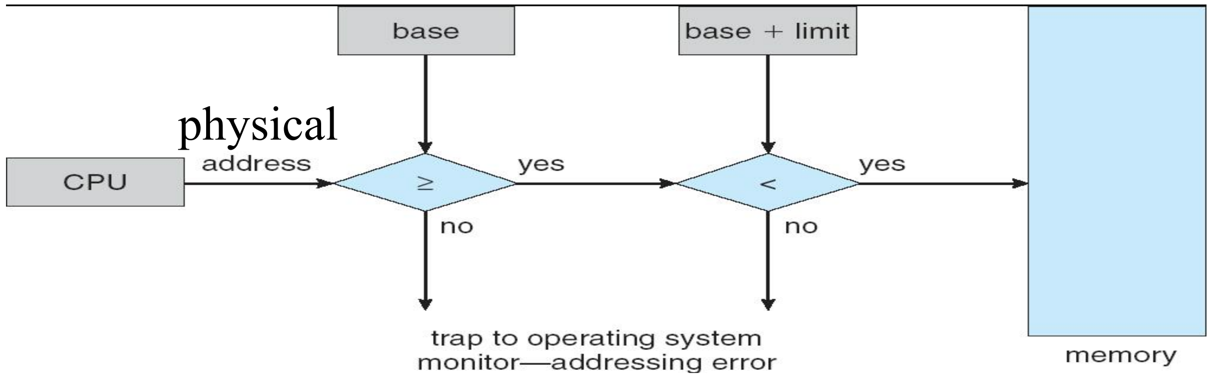
Memory management

- MMU: memory management unit
  - logical/physical memory mapping
- Relocation register
  - physical address = logical address + relocation base
- Dynamic loading
- Dynamic linking: e.g., library

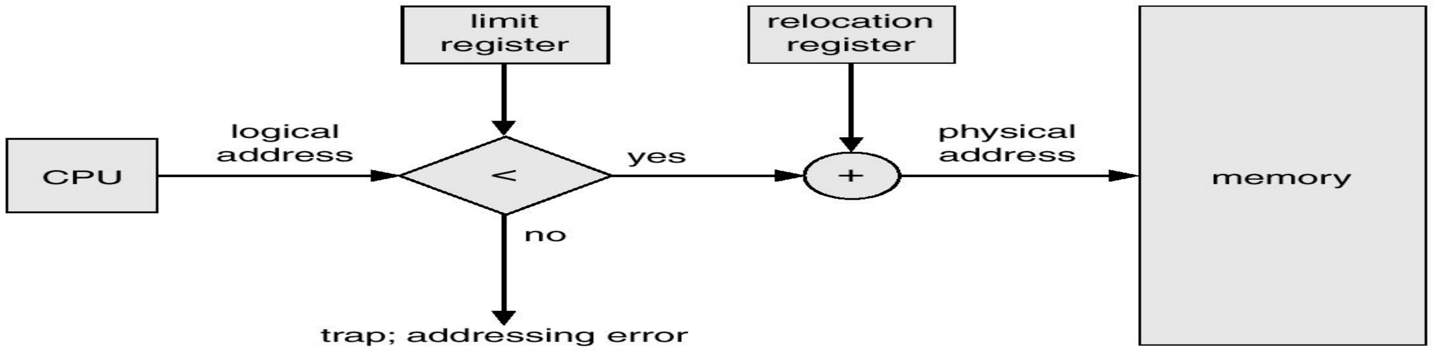


Memory protection

- With base and limit registers

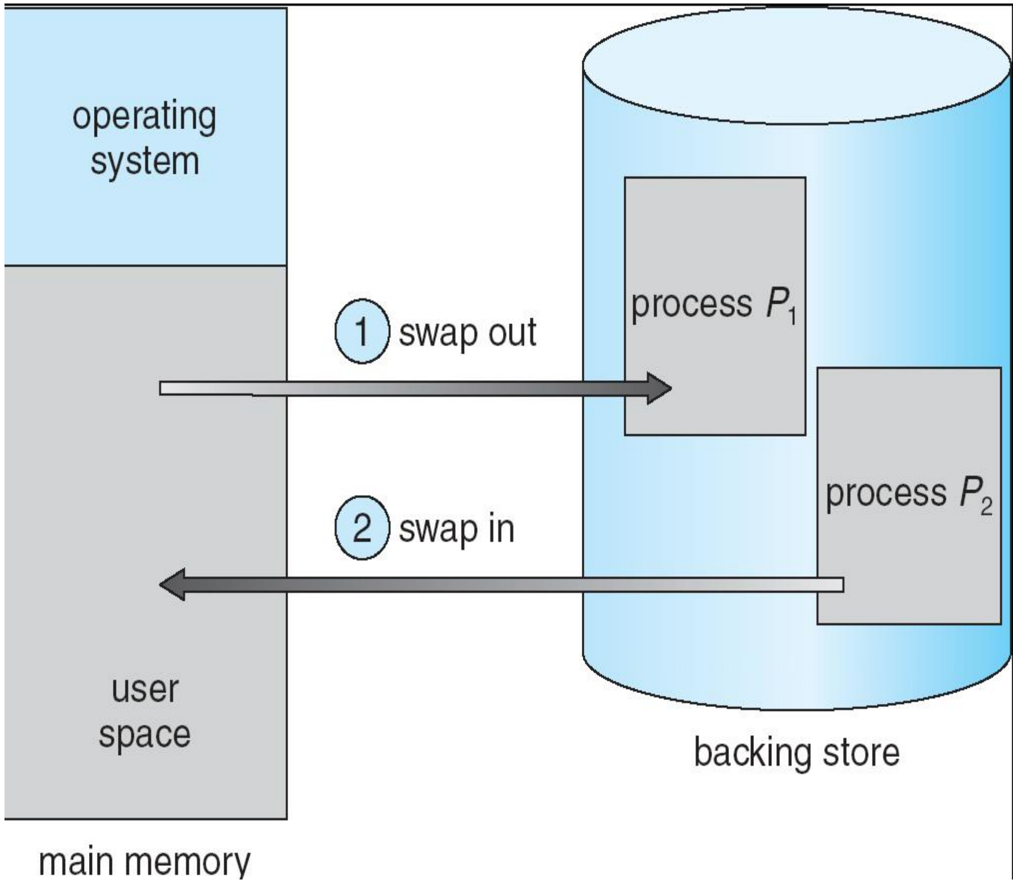


- With limit and relocation registers



Swapping

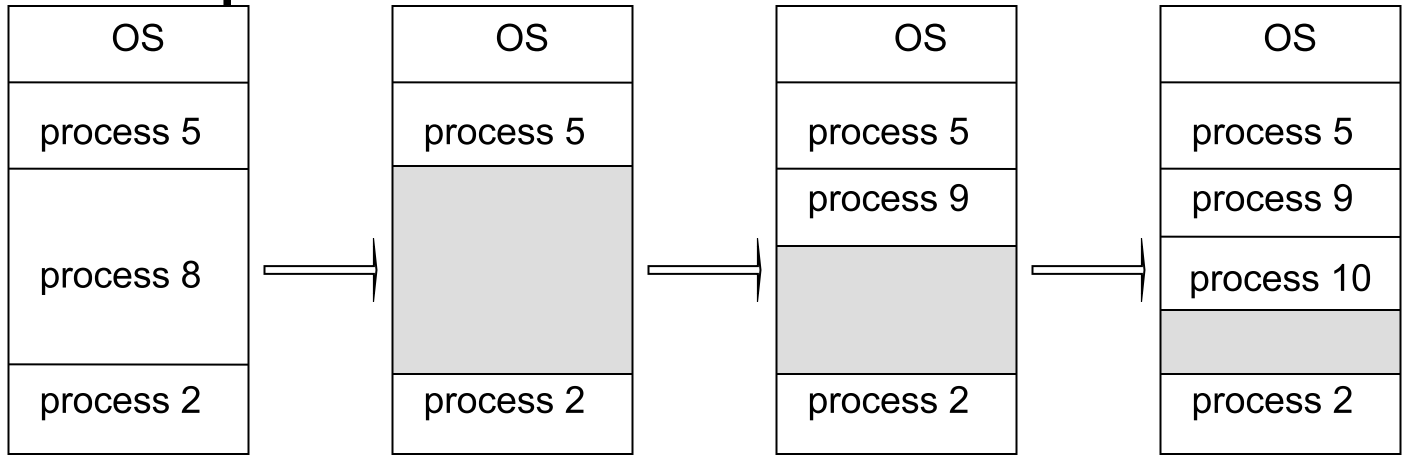
- Swap out
  - e.g., low priority
  - reduce the degree of multiprogramming
- Swap in
  - address binding
- Swapping overhead
  - on-demand



* medium-term scheduling

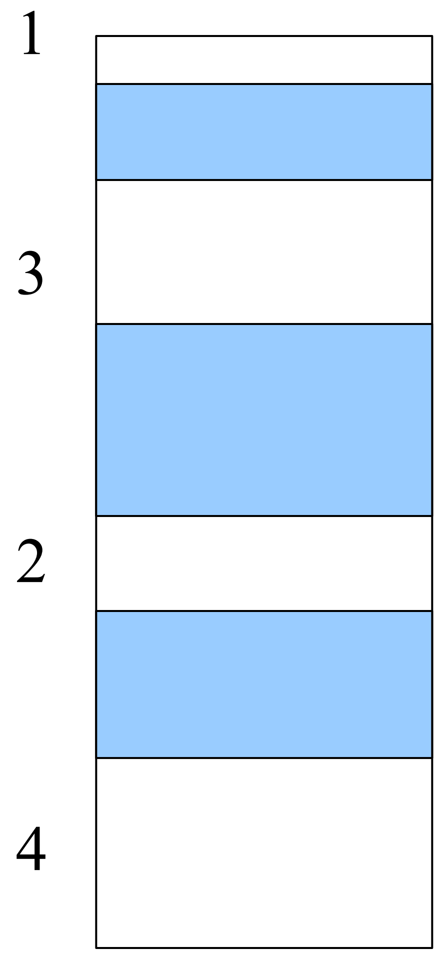
Contiguous allocation

- Single-partition allocation
  - one for OS
  - the other one for user process
- Multi-partition allocation



Partition allocation

- First-fit
  - first “hole” big enough to hold
  - *faster* search
- Best-fit
  - smallest “hole” big enough to hold
- Worst-fit
  - largest “hole” big enough to hold



Q: for a request of size 2? what if 5?

Fragmentation

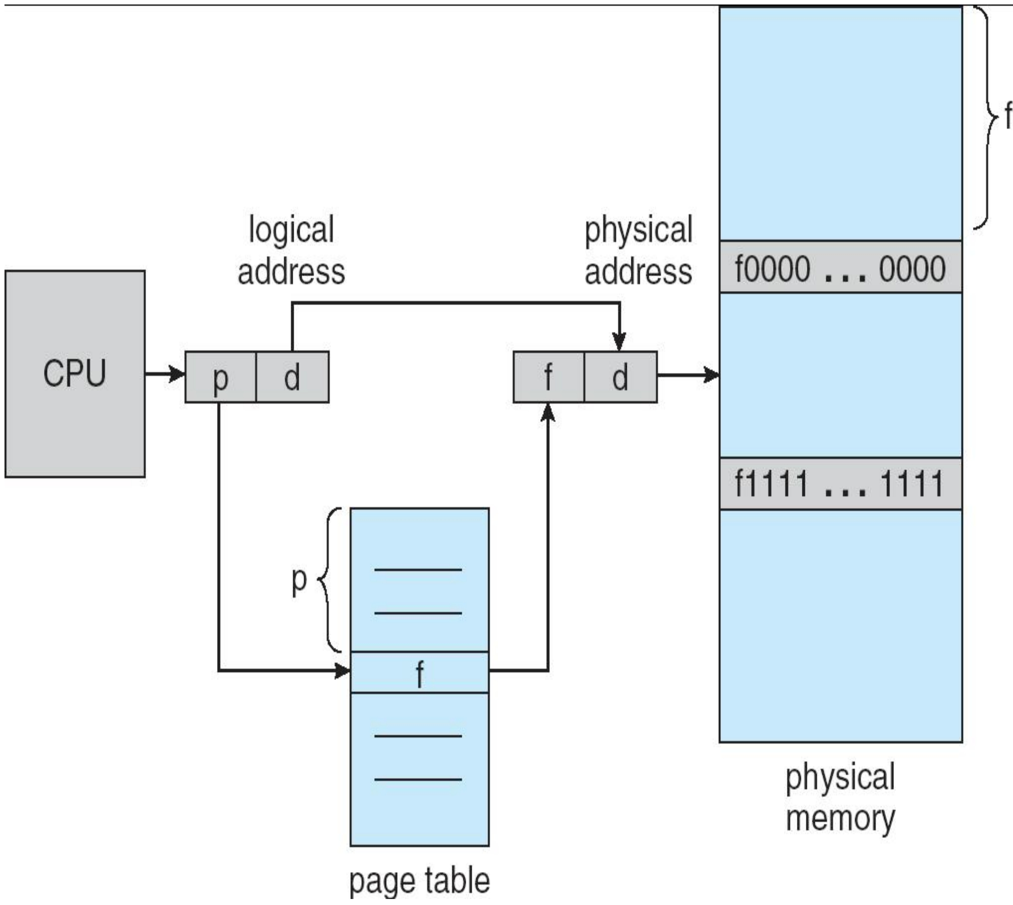
- External fragmentation
  - enough total available size, not individual ones
- Compaction
  - combine all free partitions together
  - possible if dynamic allocation at execution time
  - issues with I/O (e.g., DMA)
- Internal fragmentation
  - difference between allocated and request size

Paging

- Noncontiguous allocation
  - in fixed size pages
  - page size: normally 512B ~ 8KB
- Fragmentation
  - no external fragmentation
    - unless there is no free page
  - still have internal fragmentation
    - maximum: page_size - 1

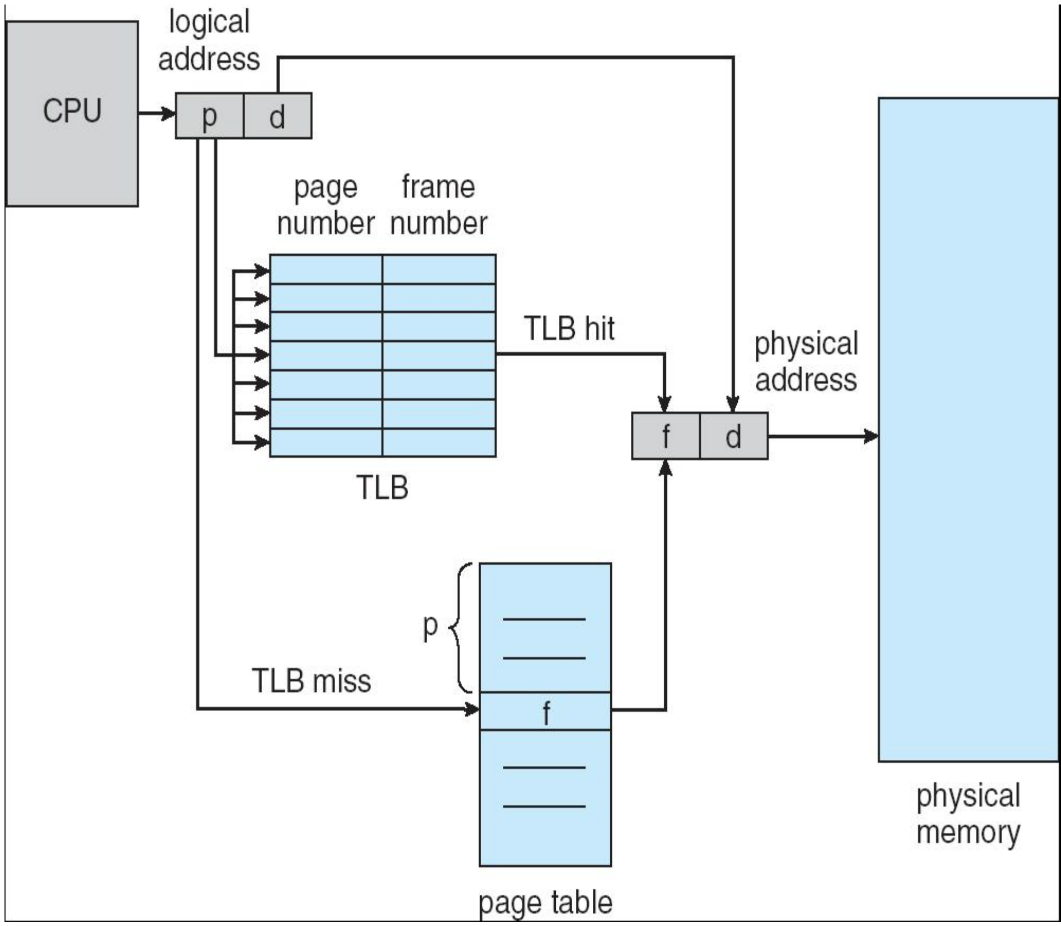
Supporting paging

- Access by address
  - seen by CPU
    - logical page number
    - page offset
    - “frame”
  - seen by memory
    - physical page number
    - page offset
- Page-table registers
  - one more memory access

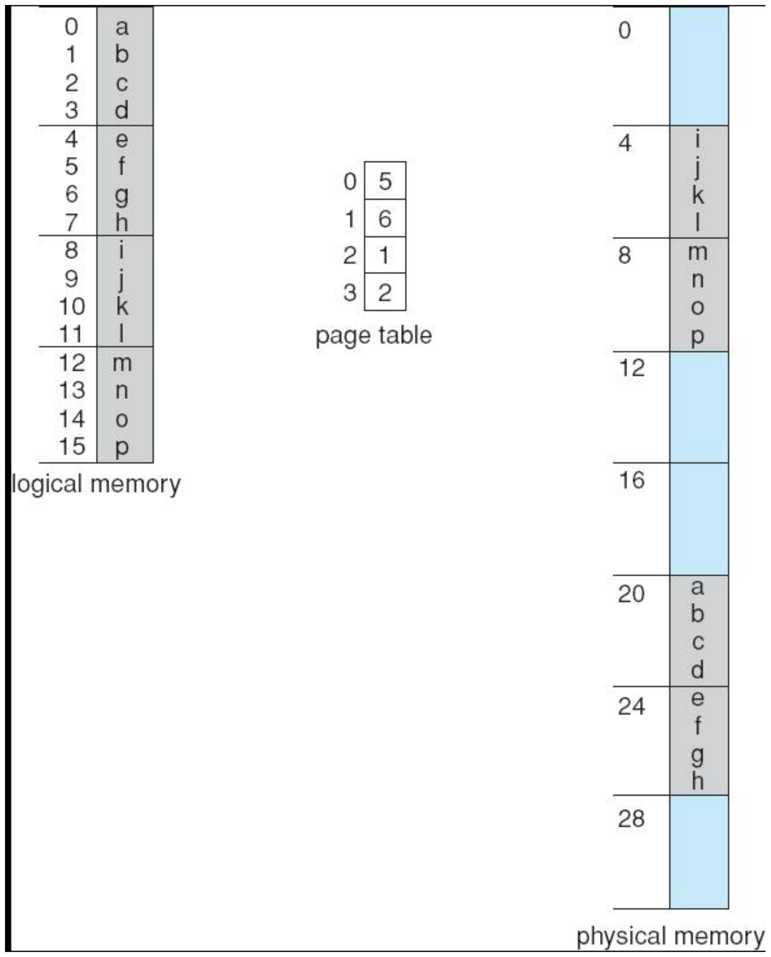


Supporting paging: more

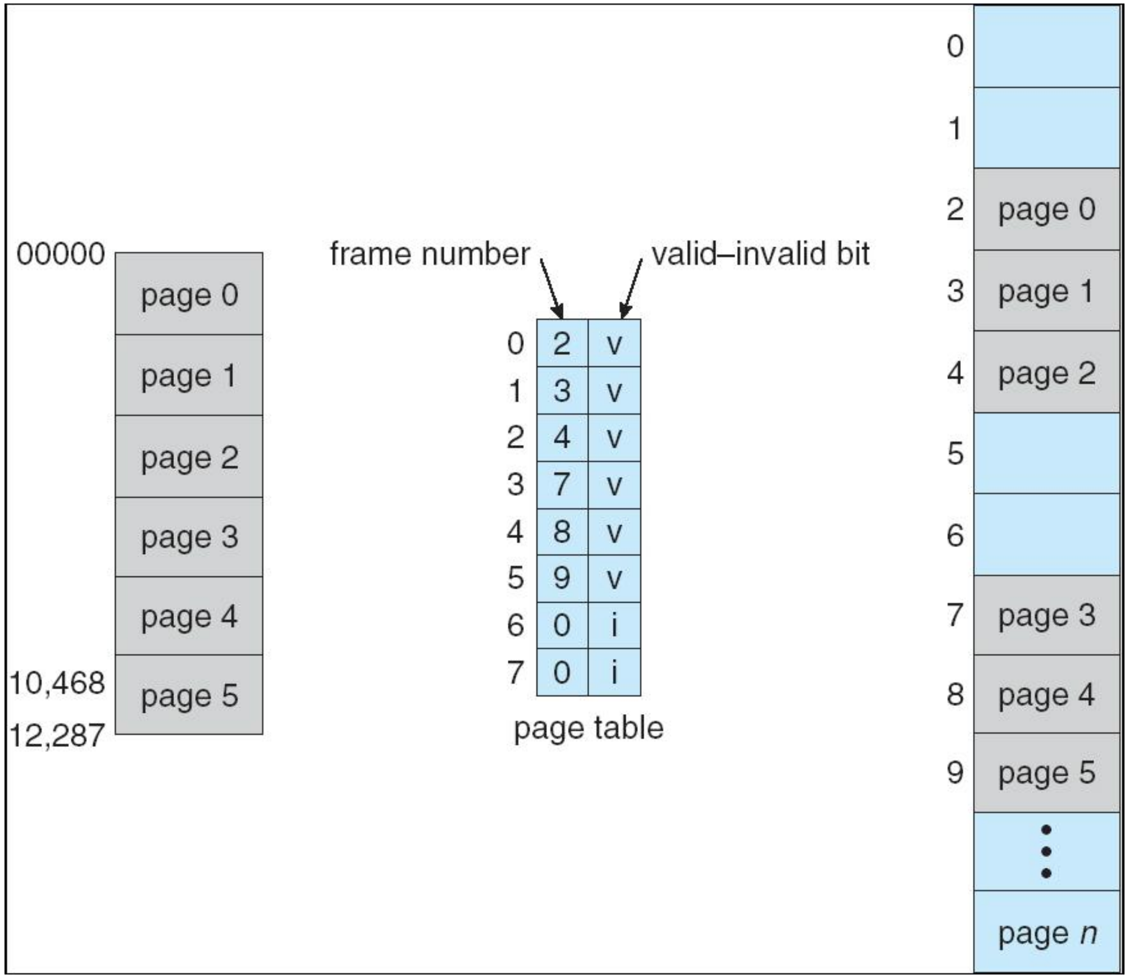
- TLB
  - translation look-aside buffer
  - associative
- Access by content
  - if hit, output frame #
  - otherwise, check page table



Paging example

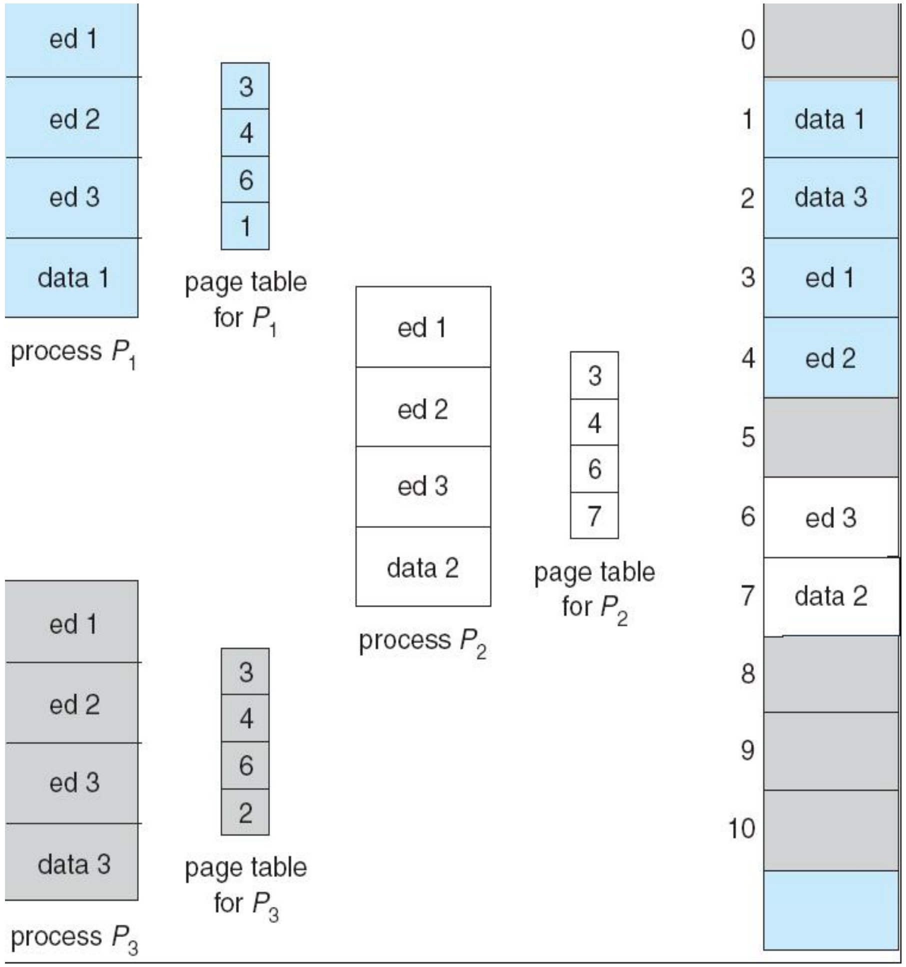


Page table: valid bit



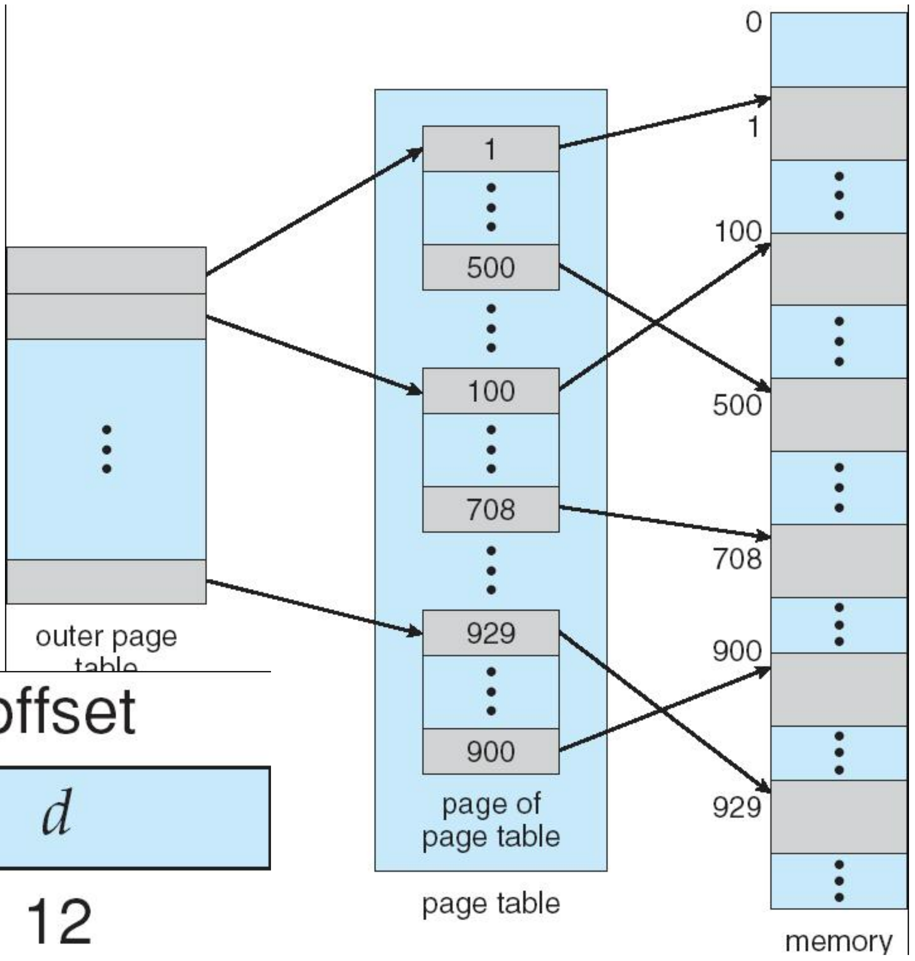
Shared pages

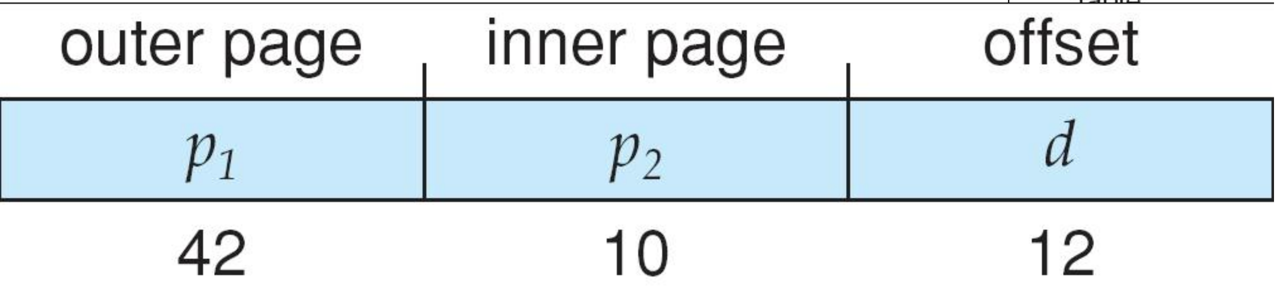
- Shared code
  - one read-only code
  - same address in logical space
- Private code + data
  - one copy per process



Hierarchical page table

- Difficulty with a table of too many entries
  - where to keep the table
  - how to lookup efficiently
- Solution
  - e.g., 2-level table

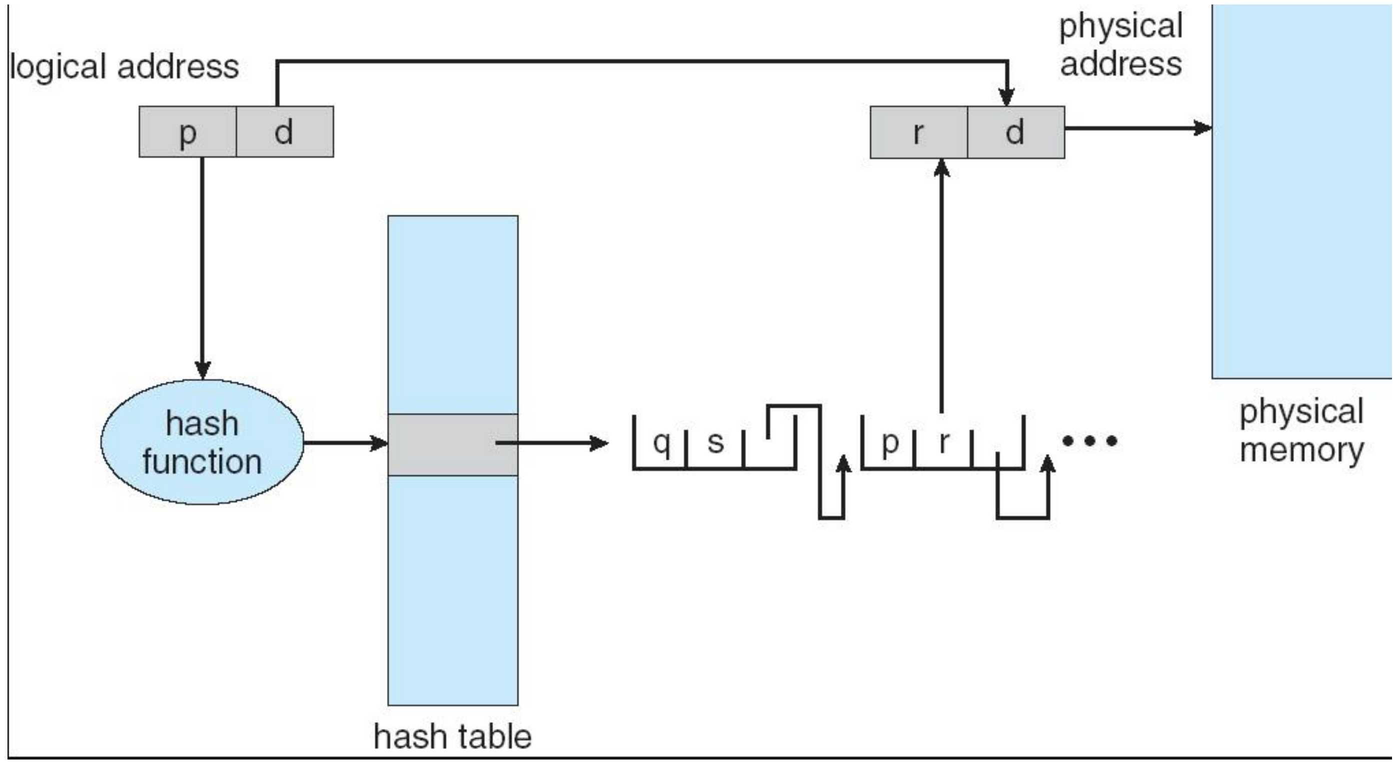




Q: any problems?

Hash page table

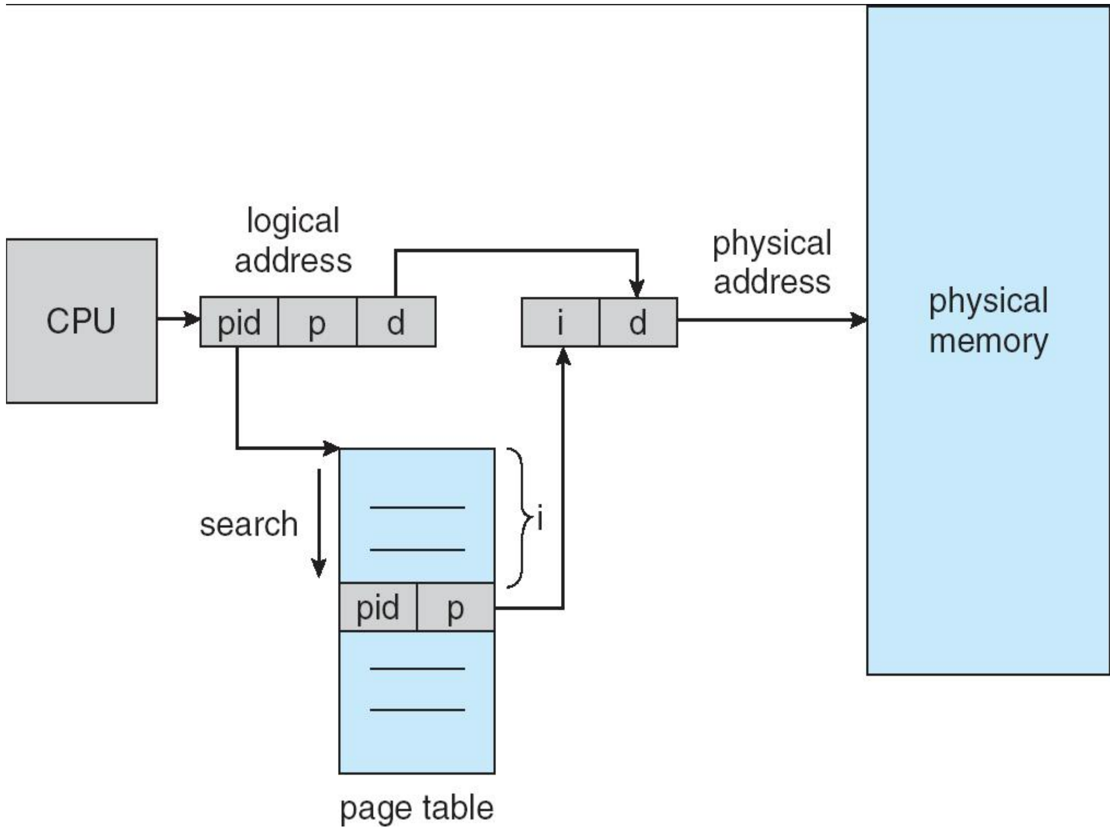
- Hash + linked list



Q: any problems?

Inverted page table

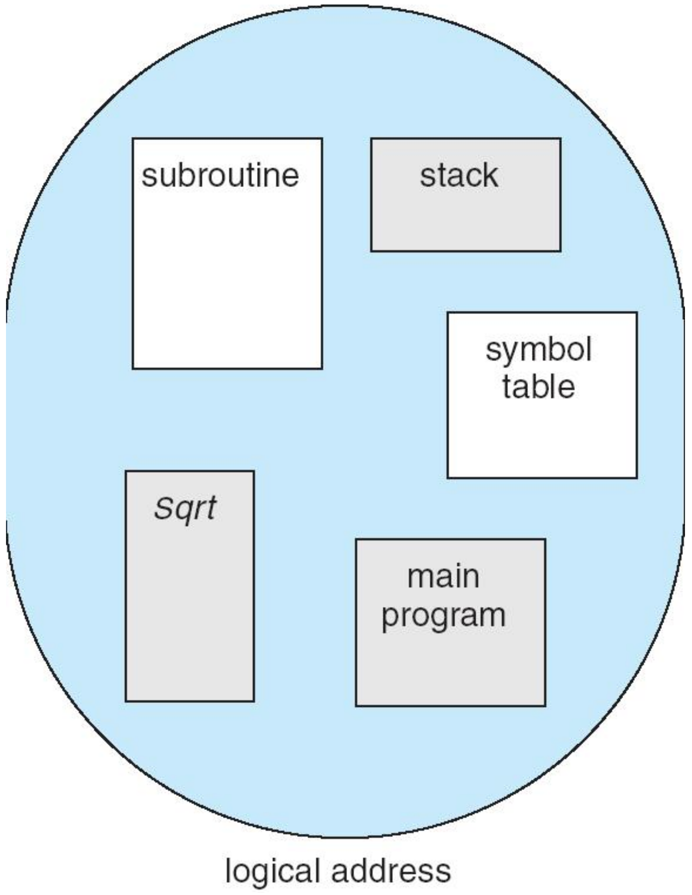
- When
  - physical space << logical space
- Tradeoff
  - time
  - space



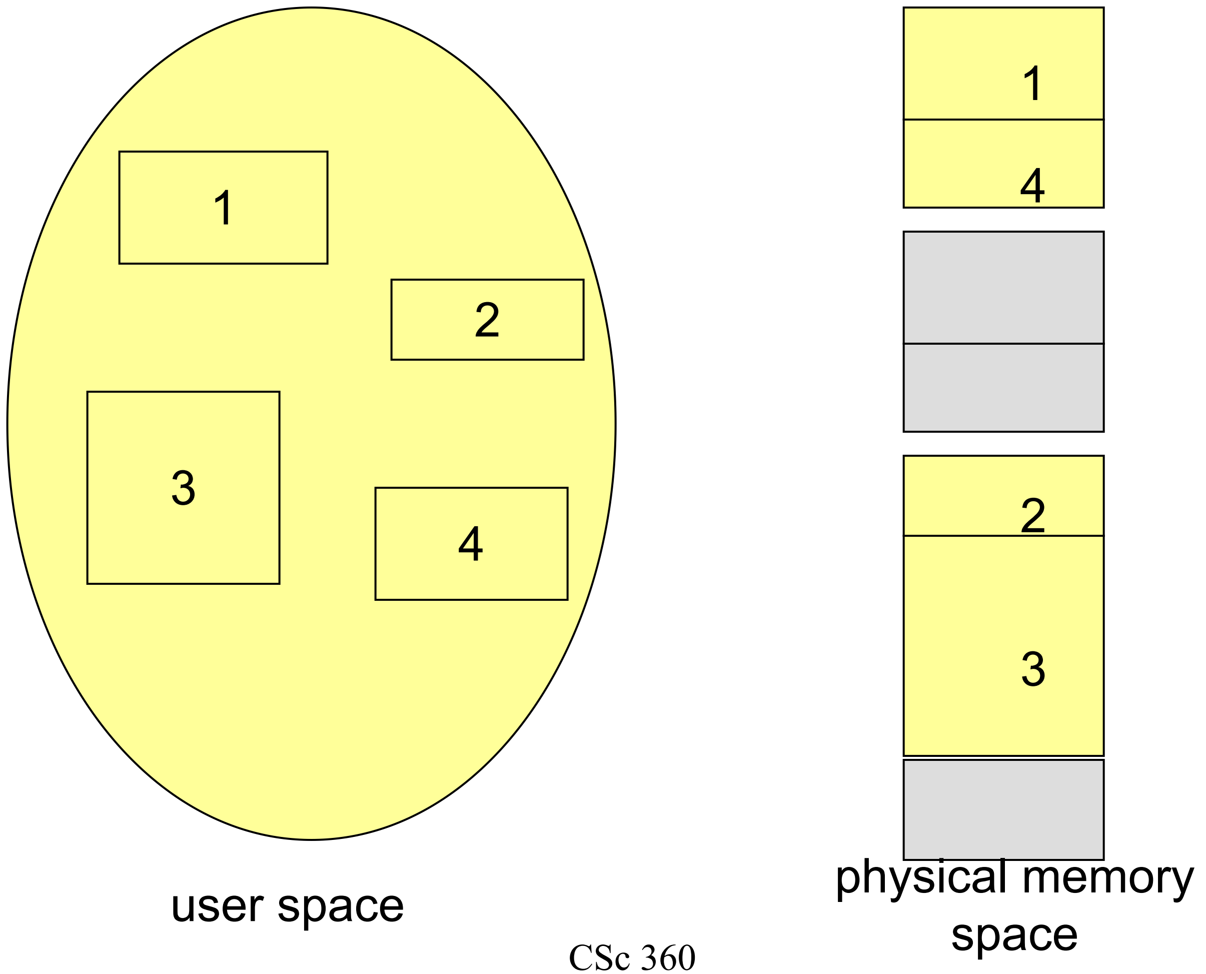
Q: any problems?

User's view of a program

- A collection of segments
  - main program
  - symbol table
  - procedures/functions
  - data
  - stacks
  - heaps



Logical view of segmentation



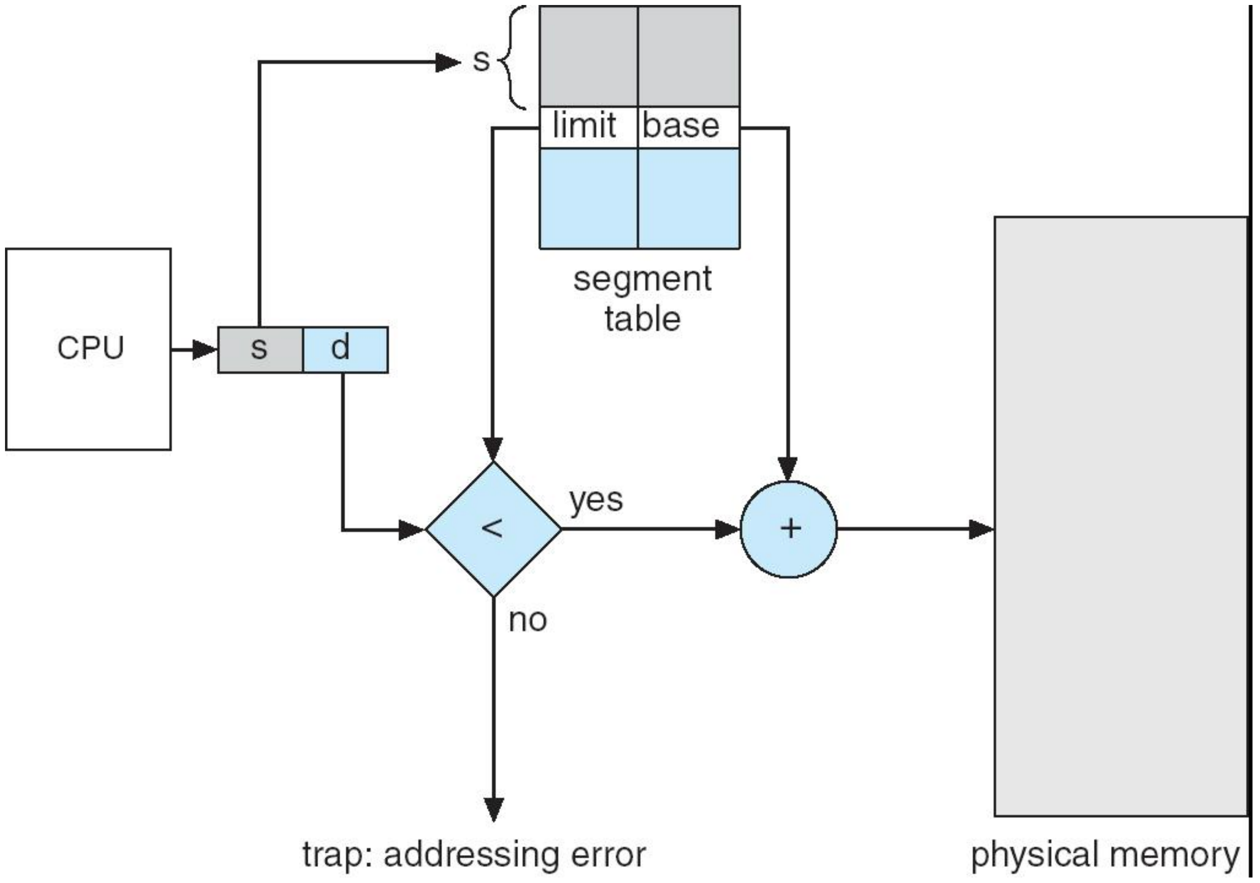
Segmentation Architecture

- Logical address consists of a two tuple:

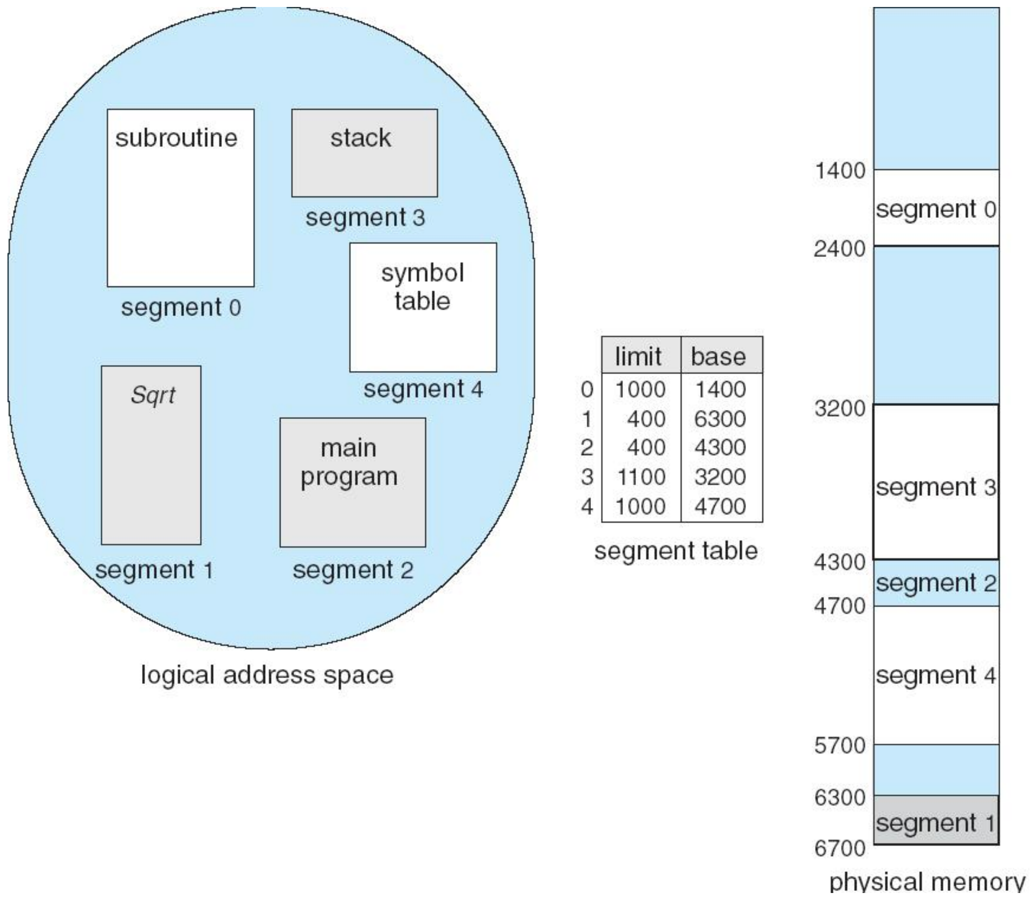
  <segment-number, offset>,
- **Segment table** – maps two-dimensional physical addresses; each table entry has:
  - **base** – contains the starting physical address where the segments reside in memory
  - **limit** – specifies the length of the segment
- **Segment-table base register (STBR)** points to the segment table’s location in memory
- **Segment-table length register (STLR)** indicates number of segments used by a program;

  segment number ***s*** is legal if ***s*** < **STLR**

Segment table



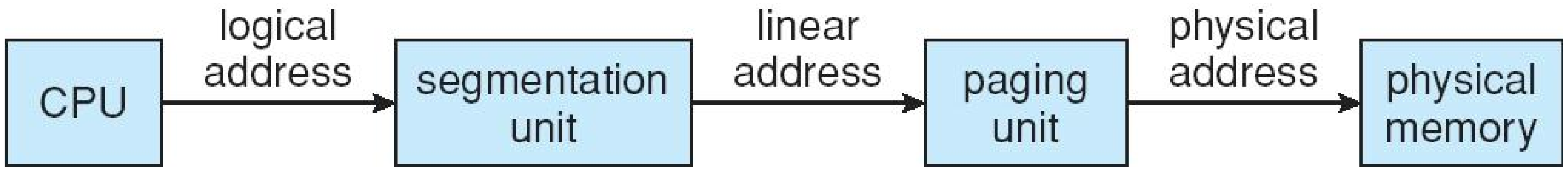
Example of segmenting



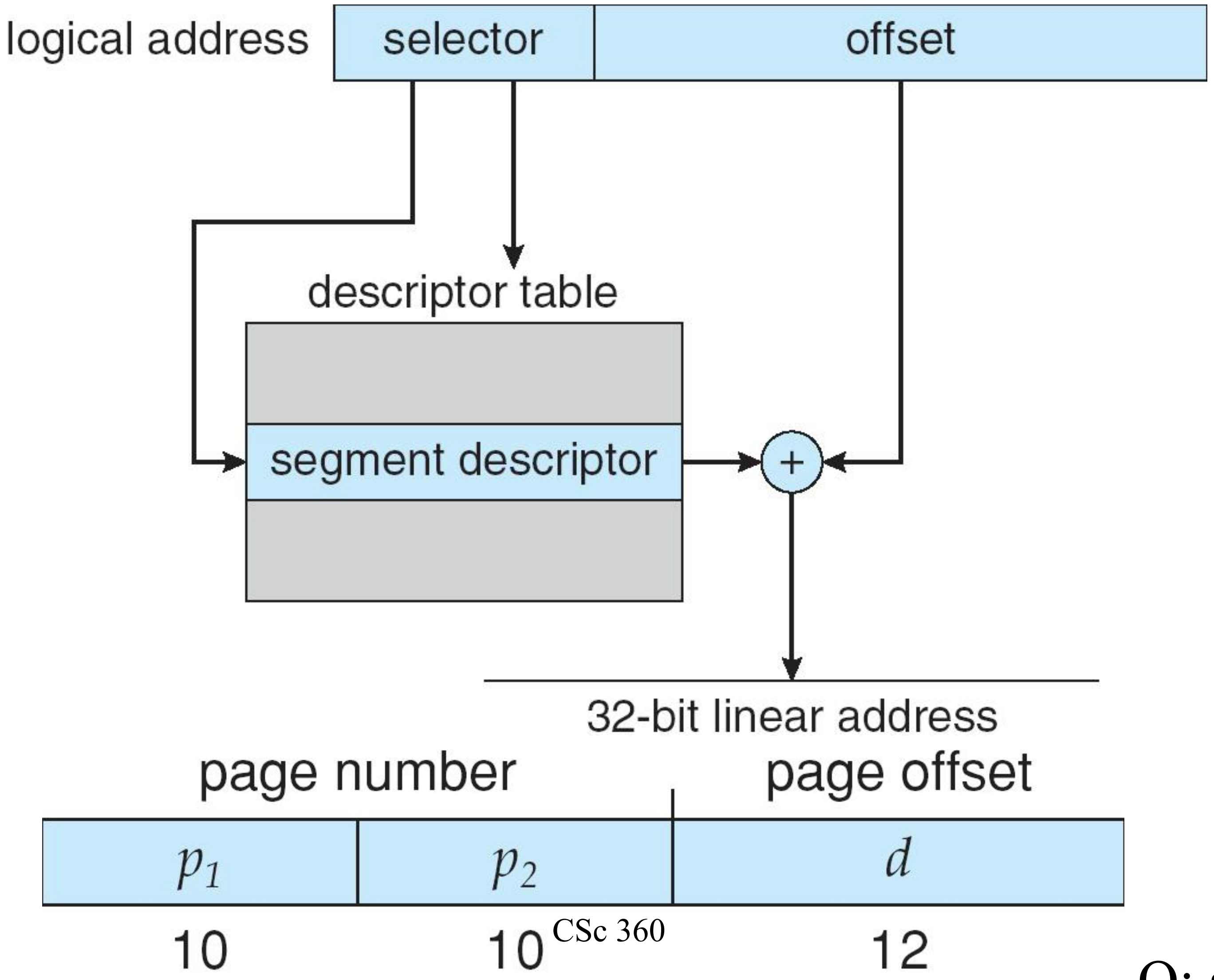
	limit	base
0	1000	1400
1	400	6300
2	400	4300
3	1100	3200
4	1000	4700

Examples: Intel Pentium

- Supports both segmentation and segmentation with paging
- CPU generates logical address
  - Given to segmentation unit
    - Which produces linear addresses
  - Linear address given to paging unit
    - Which generates physical address in main memory
    - Paging units form equivalent of MMU

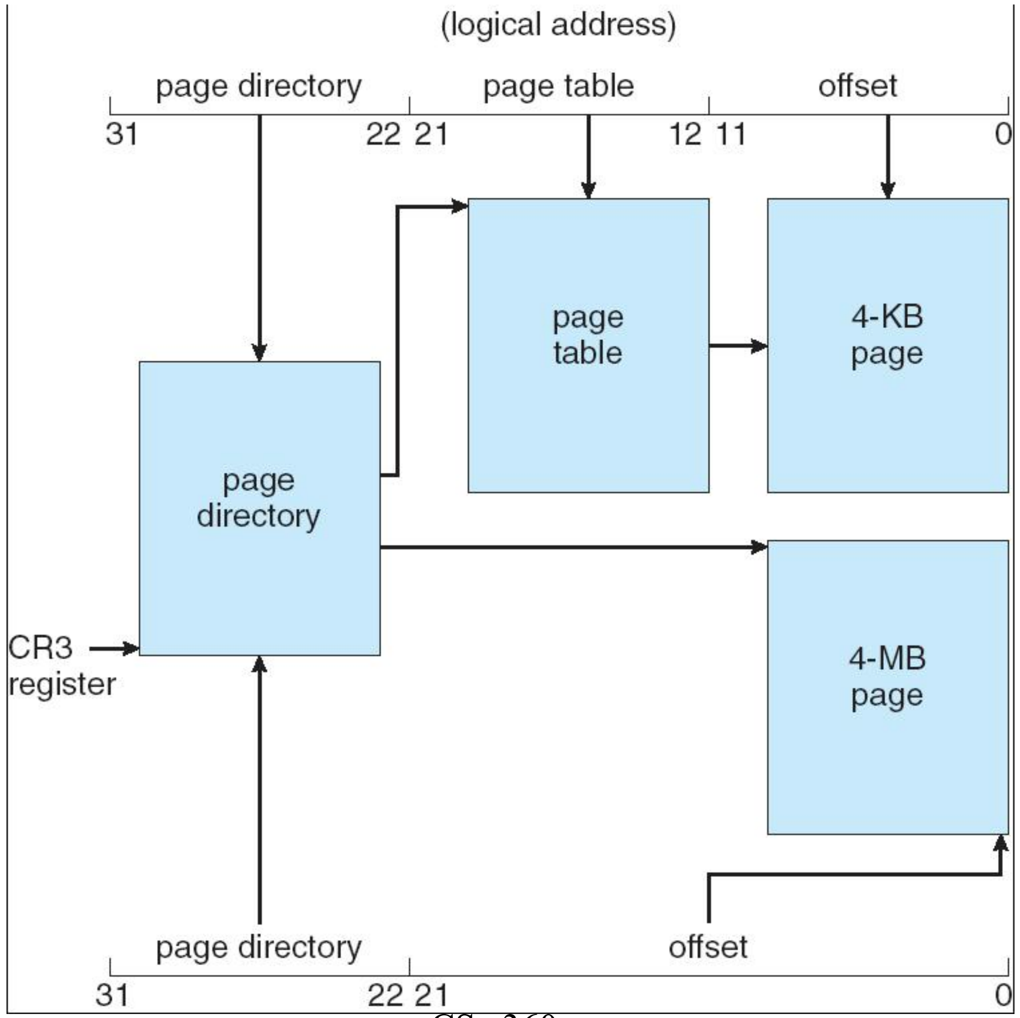


From logical to physical address



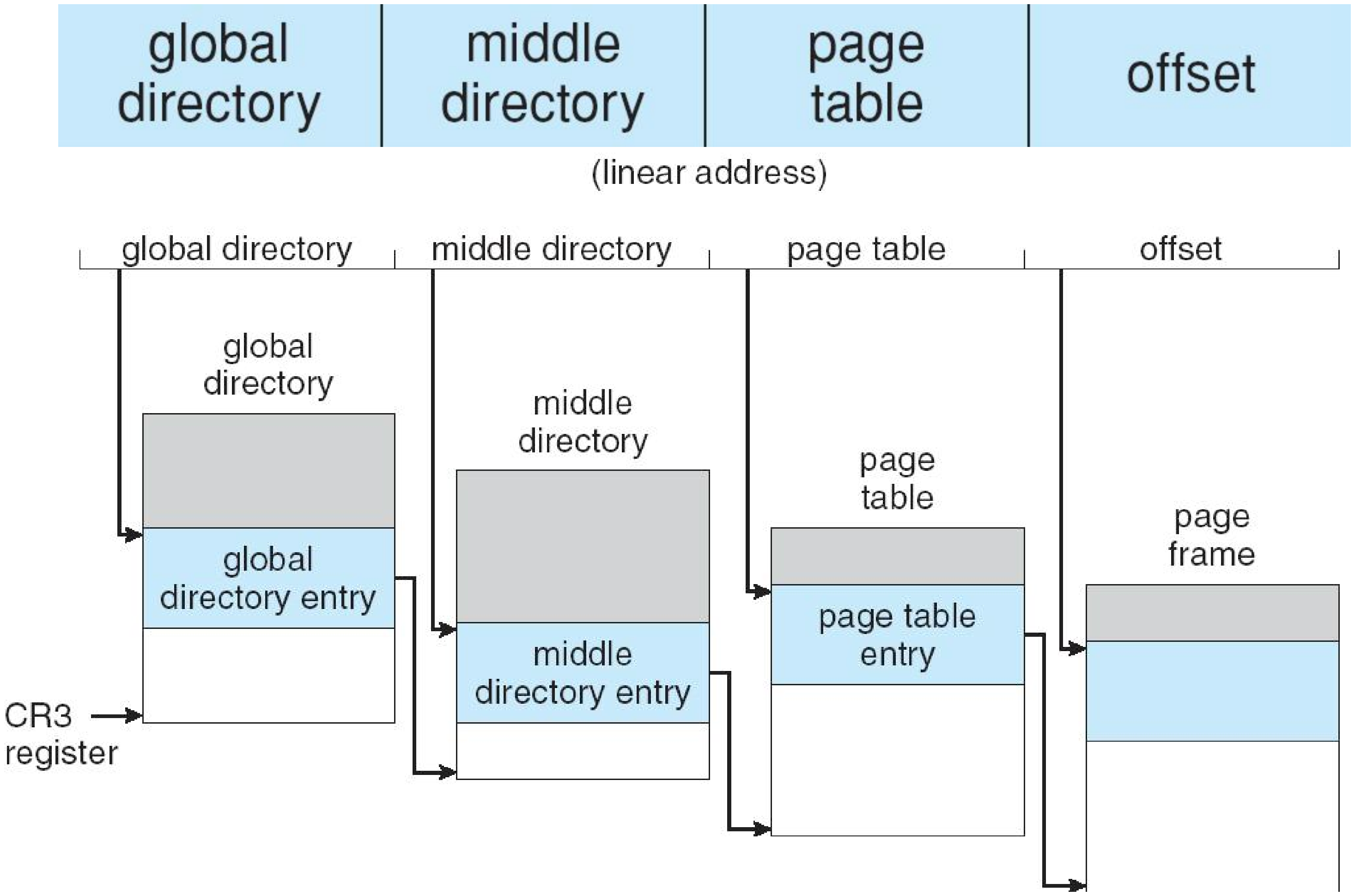
Q: segments?

From logical to physical (2)



Q: why different page size?

Examples: Linux paging



These lectures

- Memory management
  - structure and organization
  - memory allocation
  - paging
  - segmentation
- Explore further
  - /proc/meminfo
    - memory, swap

Next few lectures

- Virtual memory
  - what if main memory is not sufficient?