CSCI 4730/6730 OS
(Ch #9 Main Memory)

In Kee Kim

Department of Computer Science

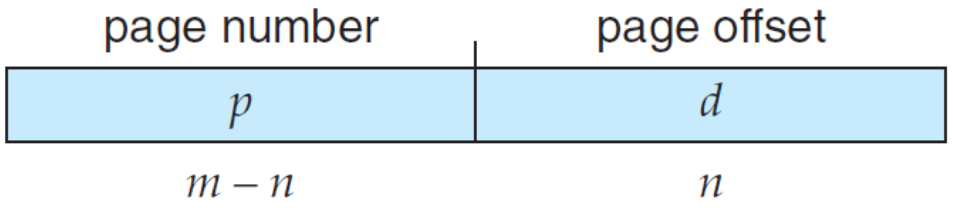
University of Georgia

Recap: Paging

- **90/10 rule**: Processes spend 90% of their time accessing 10% of their space in memory!
- Address external fragmentation
- Divide logical memory into blocks of same size called ***pages***
- OS keeps track of all free ***frames***
- To run a program of size ***N*** pages, need to find ***N*** free frames and load the program
- Set up a **page table** to translate logical address to physical address
- However, paging does not eliminate ***internal*** fragmentation (1/2 pages per process)

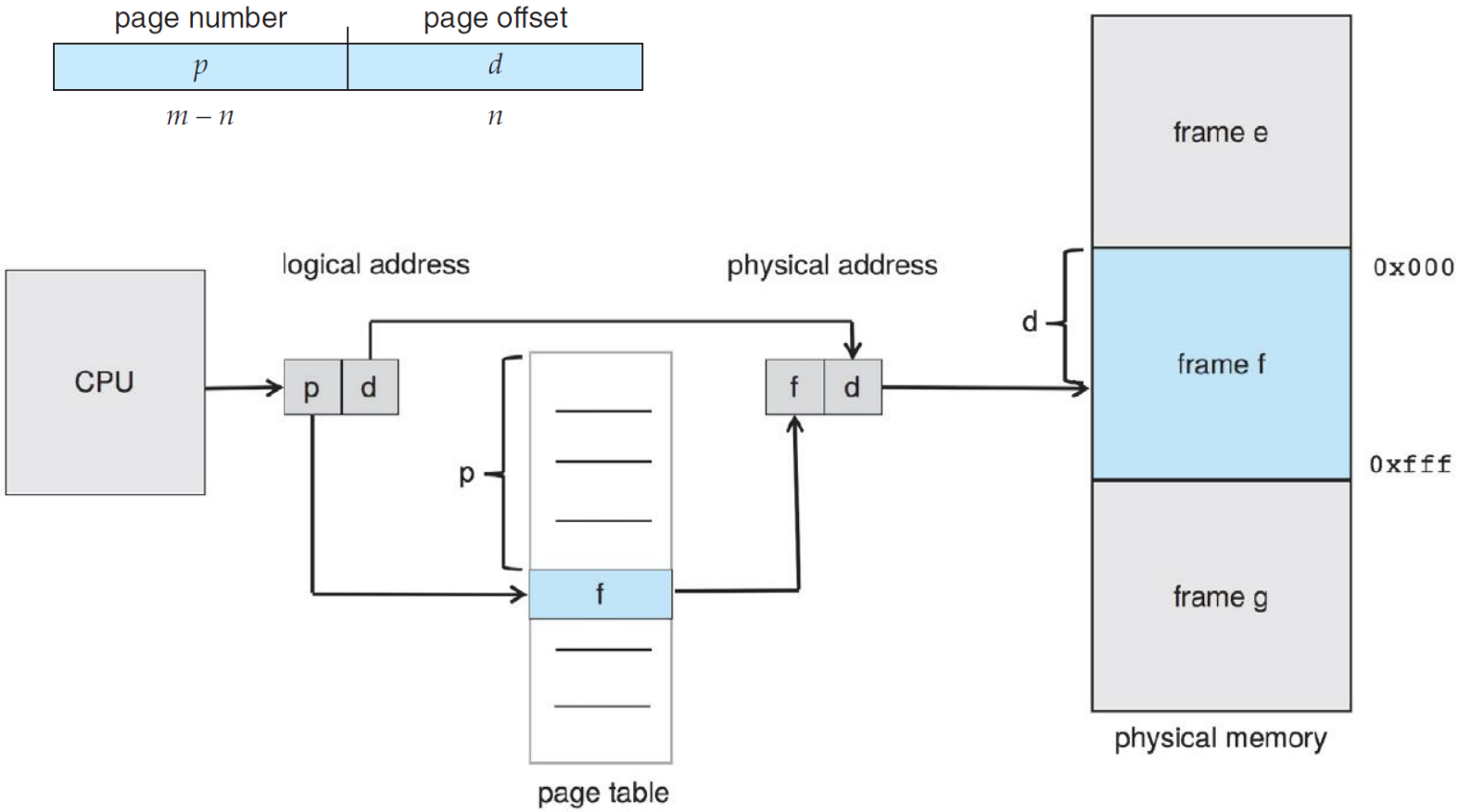
Recap: Address Translation Scheme

- Address generated by CPU is divided into:
  - **Page number (*p*)**
    - Used as an index into a **page table** which contains base address of each page in physical memory
  - **Page offset (*d*)**
    - Combined with base address to define the physical memory address that is sent to the memory unit

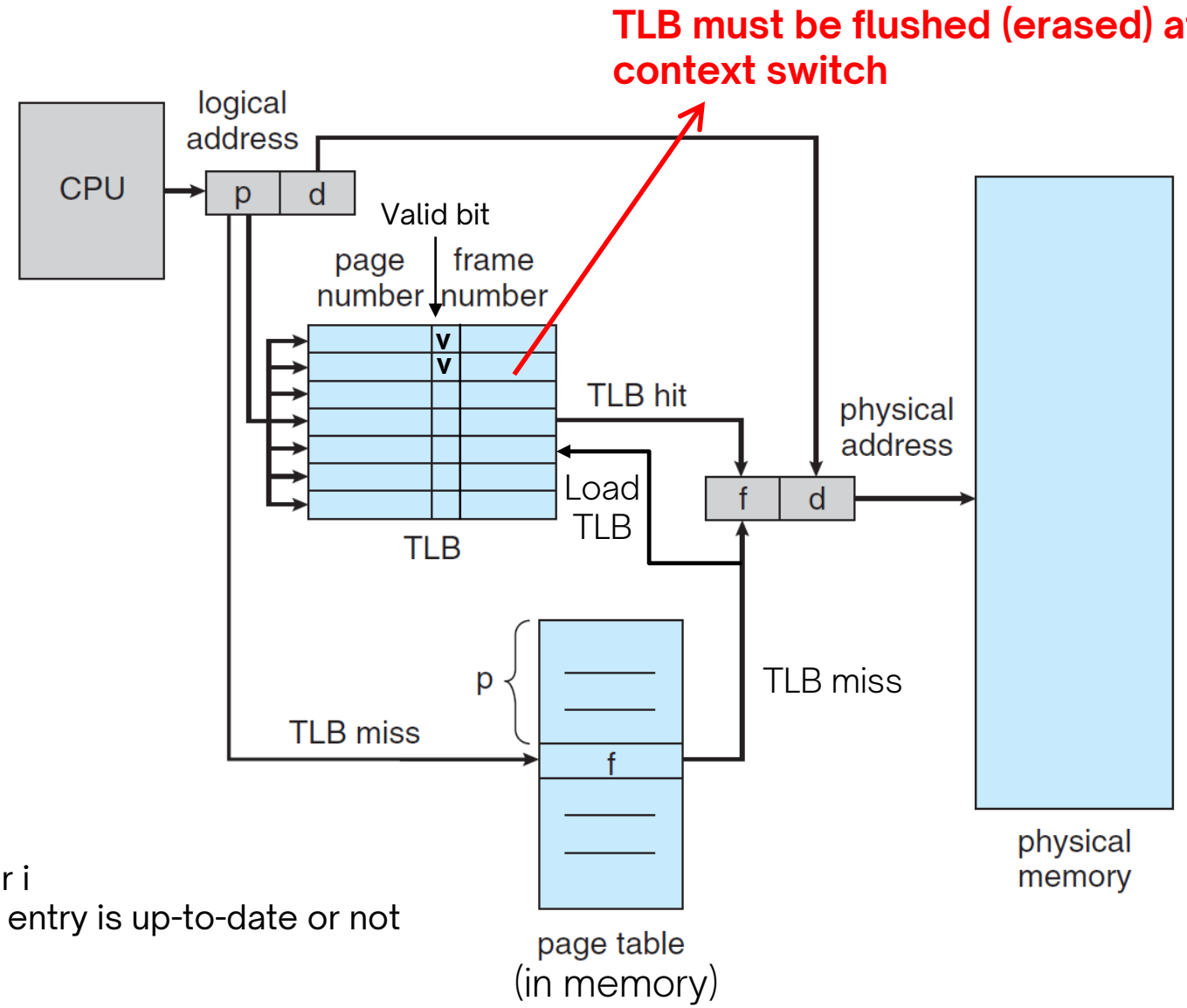


  - For given logical address space 2^m and page size 2^n

Recap: Paging Hardware



Recap: Paging Hardware with TLB



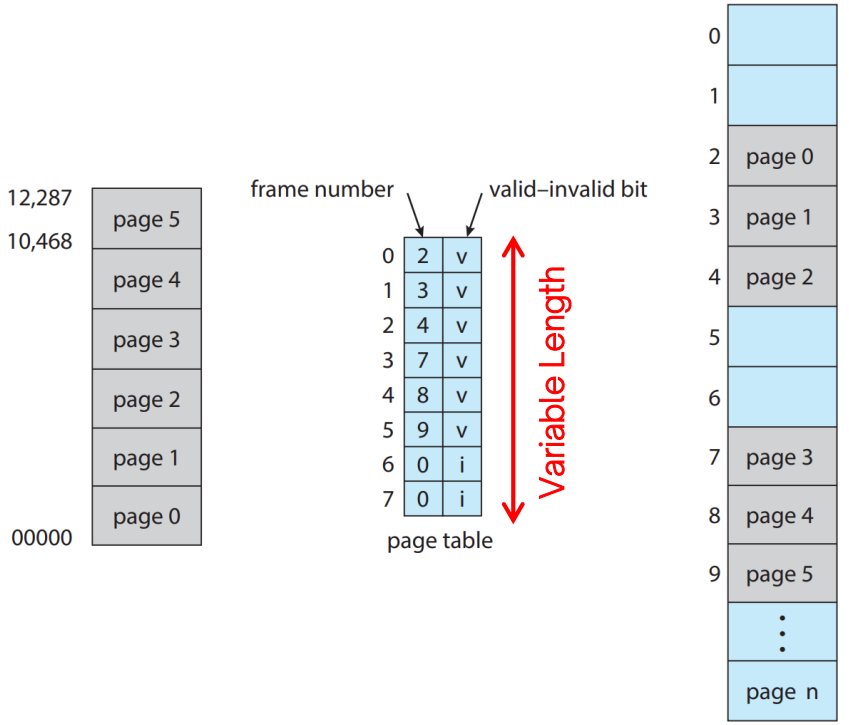
Valid Bit: v or i
– saying the entry is up-to-date or not

Recap: Effective Access Time

- What is the effective access time (EAT) if the page table is in memory?
  - Two memory access time
  - EAT = 2 * MAT (memory access time)
- What is EAT with a TLB?
  - TLB hit ratio: *p*, TLB: TLB access time
  - EAT = (TLB + MAT) * *p* + (TLB + 2 * MAT) * (1 - *p*)
- As *p* increases, EAT decreases.

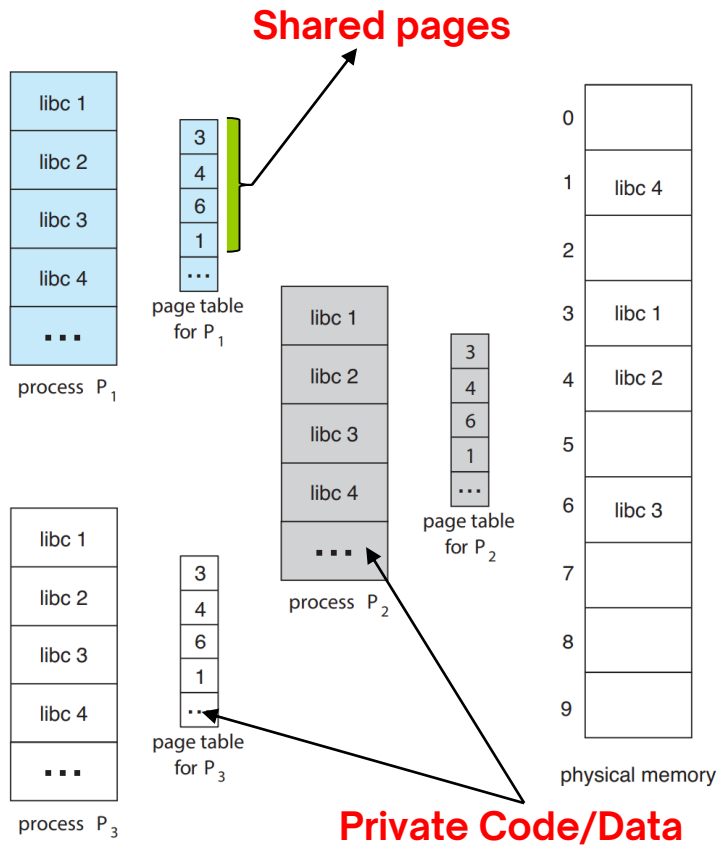
Memory Protection

- **Valid-invalid** bit attached to each entry in the page table:
  - "**valid**" indicates that the associated page is in the process' logical address space, and is thus a legal page
  - "**invalid**" indicates that the page is not in the process' logical address space
  - Some systems use **page-table length register** (**PTLR**)



Shared Pages

- Shared page: Processes can share common code.
  - e.g., **`libc`** (system calls)
  - Only one copy in the physical memory and page tables of different processes can point the same frames.
  - What is benefit?



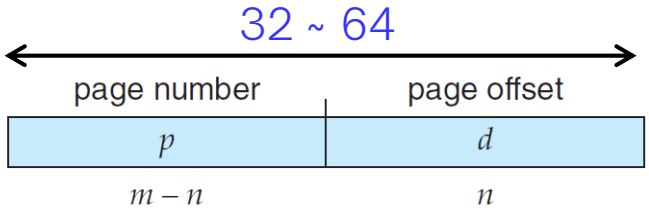
Updated New Process Arrival with TLB

- A process needing ***n*** pages arrives
- If ***n*** frames are available, allocate these frames to pages
- OS puts each page in a frame and then puts the frame number in the corresponding entry in the page table
- OS flushes the TLB (mark all "**invalid**")
- OS start the process
- As process executes, OS loads TLB entries as each page is accessed, replacing an existing entry if TLB is full

PCB Modification for TLB Support

- PCB must be extended to contain
  - Page Table (**PTBR**)
  - A copy of TLB
- On a Context Switch
  - Copy **PTBR** to PCB
  - Copy TLB to PCB (optional)
  - Flush TLB (mark all "**invalid**")
  - Restore **PTBR**
  - Restore the TLB if it was saved

Structure of Page Table



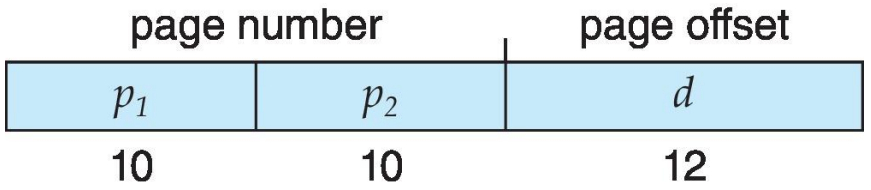
- Memory structures for paging can get huge using straight-forward methods for modern OS
  - 32bit or 64bit; logical address space 2^{32} to 2^{64}
  - Consider a 32bit logical address space
  - Page size of 4 KB (2^{12})
    - Q. Then what is the number of page table entries?
  - Page table would have 1,048,576 entries (2^{32} / 2^{12})
  - If each entry is 4 bytes (Don't be confused with page size.)
    - Q. What is the page table size for each process?
    - 4MB (4B * 1M) of physical address space for the page table alone
    - Don’t want to allocate that contiguously in main memory

Structure of Page Table

- Memory structures for paging can get huge using straight-forward methods for modern OS
- One simple solution is to divide the page table into smaller units
  - Hierarchical Paging
  - Hashed Page Tables
  - Inverted Page Tables

Hierarchical Paging

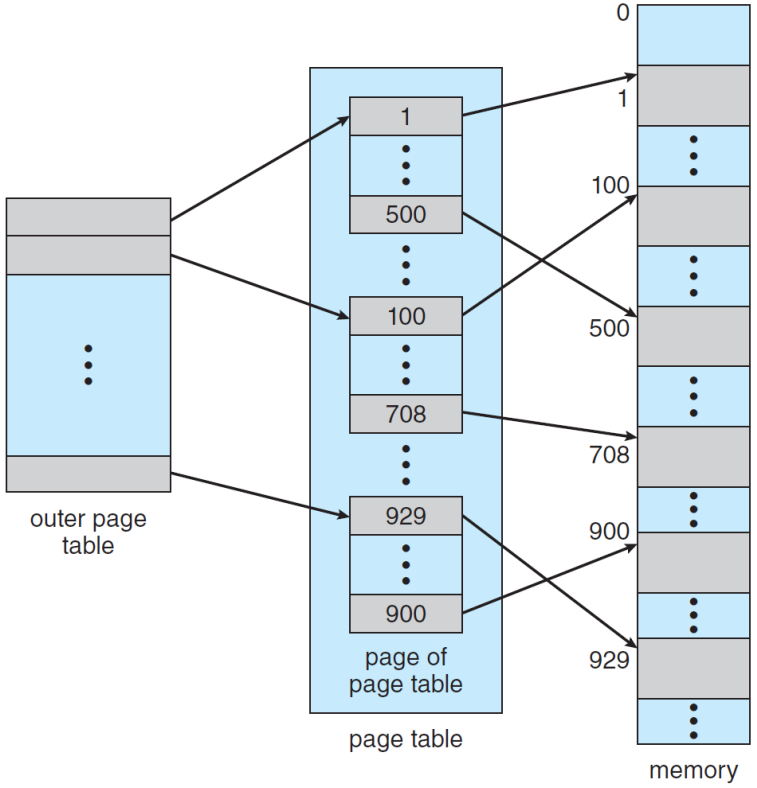
- A logical address (on 32bit machine with 4K page size) is divided into:
  - a page number consisting of 20bits (2^{20} is big!)
  - a page offset consisting of 12bits
- **20bits for page number is paged!**
  - 20bits is divided into
    - a 10-bit page number
    - a 10-bit page offset

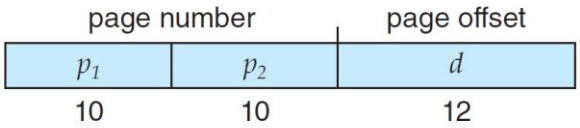


  - p_1 is an index into the outer page table, and p_2 is the displacement within the page of the inner page table
  - Known as **forward-mapped page table**

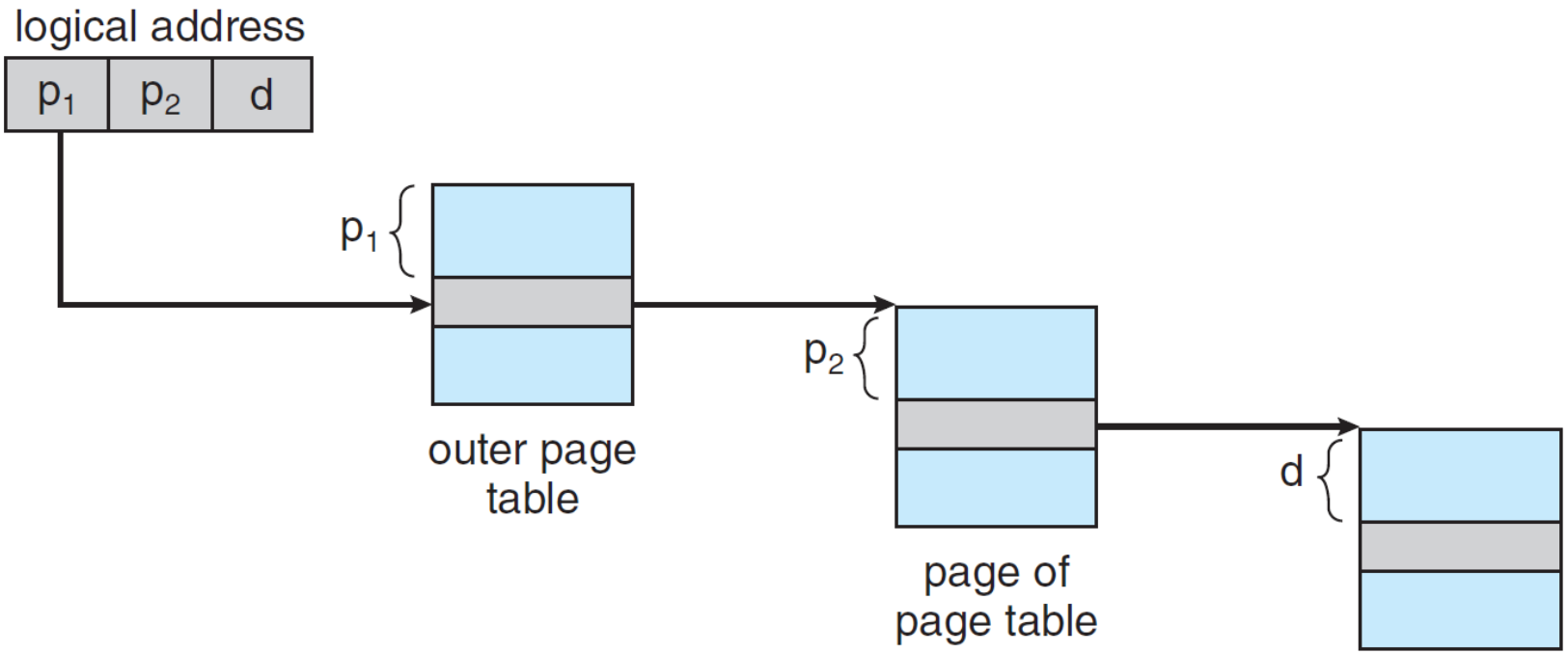
Hierarchical Paging

- Break up the logical address space into multiple page tables
- A simple technique is a two-level page table
- We then page the page table

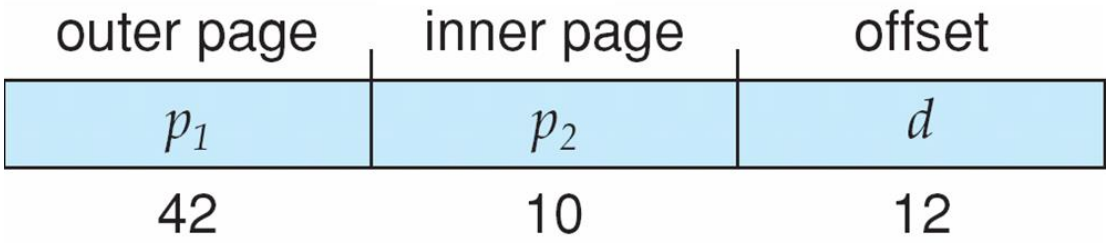


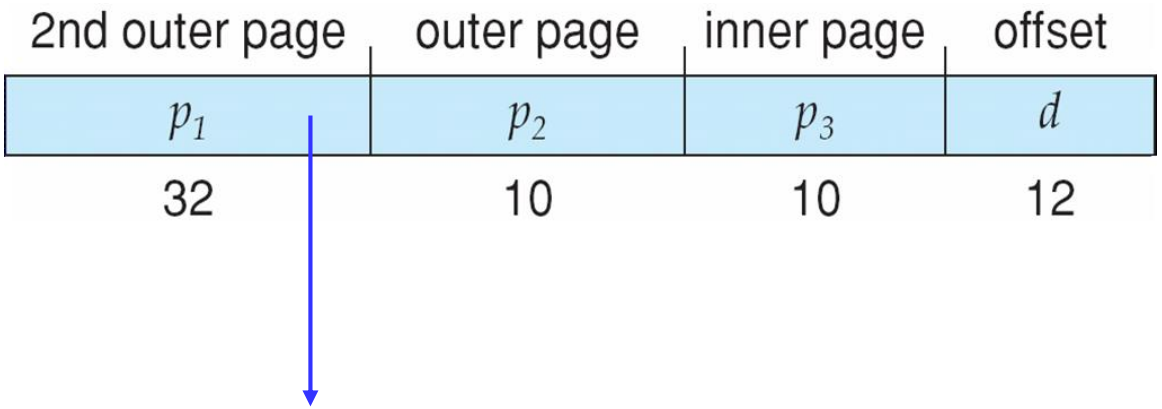


Address-Translation Scheme



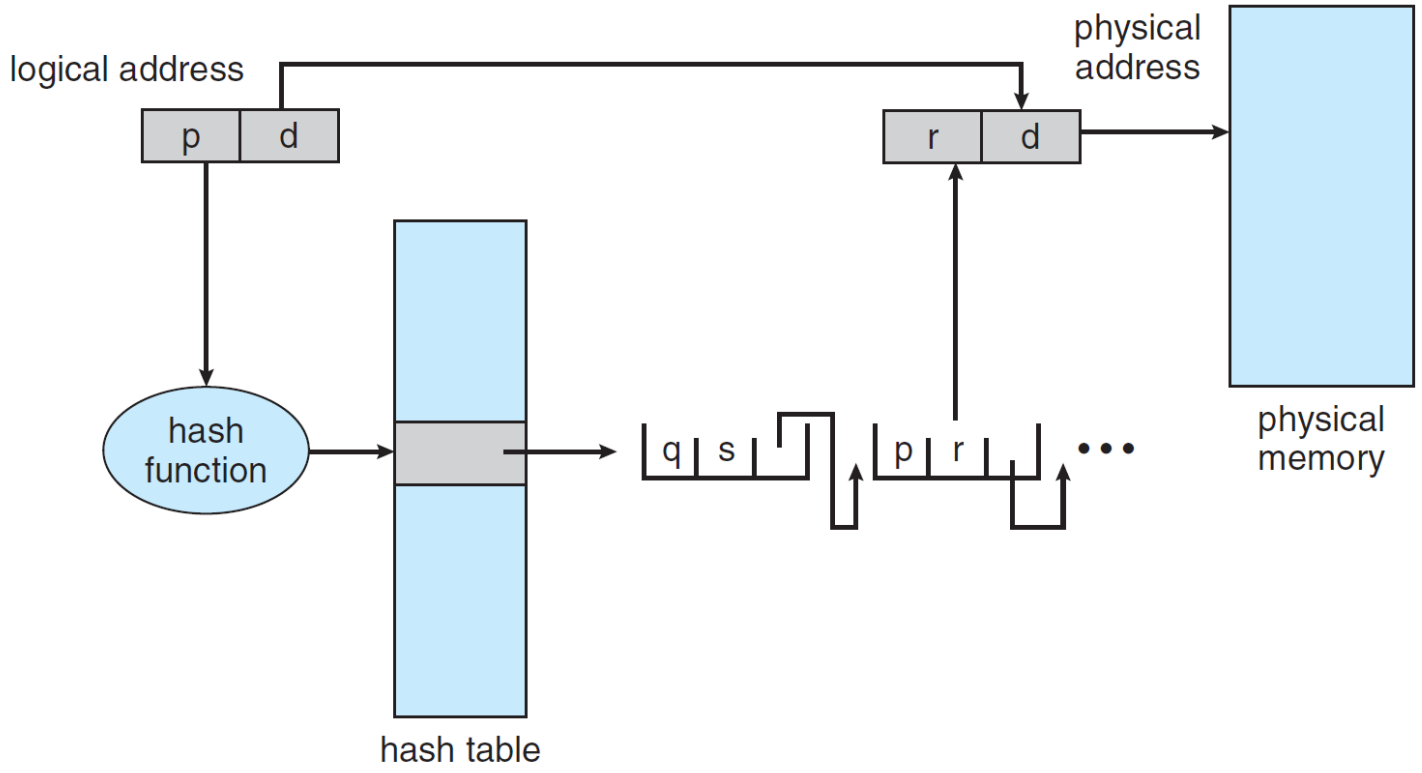
How about 64bit machine?





Hashed Page Tables

- Common in address spaces > 32 bit



Paging Summary

- Paging is a significant improvement over relocation (and compaction)
  - Eliminates external fragmentation and no need for compaction
- However, paging has its costs
  - Translating from a logical address to a physical address is more time consuming (compared to contiguous allocation)
  - Paging requires HW support (TLB) to be efficient enough
  - Paging becomes more complex in modern OS and architecture
    - Increased management cost

End of Chapter 9

CSCI 4730/6730 OS
(Chap #10 Virtual Memory – Part I)

In Kee Kim

Department of Computer Science

University of Georgia

Announcement

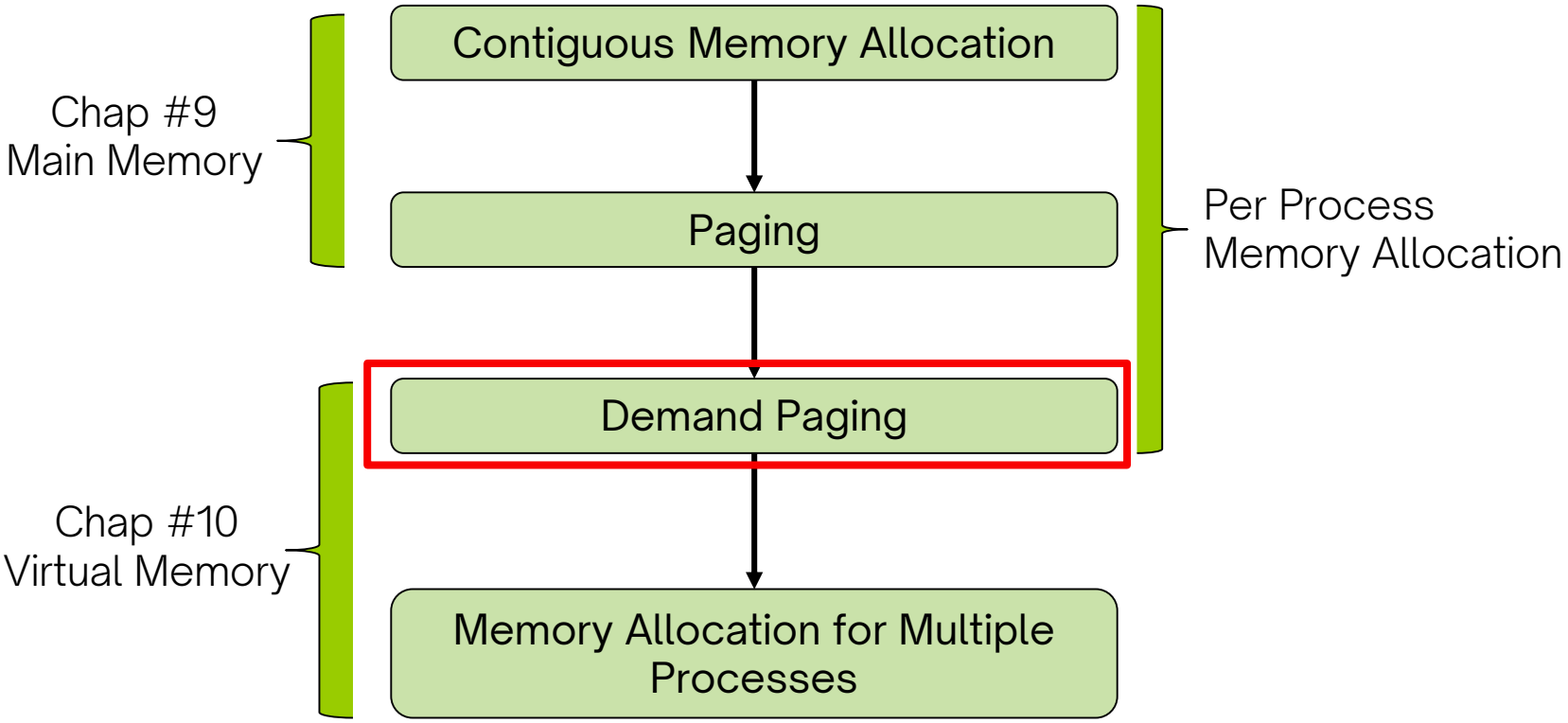
- Grad Presentation (two assignments on eLC)
  - (for Grad) [Grad Student] Presentation Team Building
    - Due: Nov/6 – 11:30 p.m.
    - One or two members per team.
    - If you don't submit this, you will not do presentation.
      - 10% of total grade is gone
      - But submission by your teammate is fine.
  - (for Under) [For Undergrad] Interesting Topic for Grad Presentations
    - Two interesting topics for grad presentation, Due: Nov/7
    - For extra credit

Chap #10 Virtual Memory

- Demand Paging
- Page Replacement
- Thrashing
- Memory allocation for multi-programming
- Allocating Kernel Memory

Where are we?

- Memory Management



Why we study Virtual Memory

- In Chap #9 – Main Memory
  - Logical (virtual) address space of a process fits in memory
  - In other words, we assumed that the logical address space was all in memory

- In practice, virtual memory > physical memory
  - `ps -eo pid,rss,vsz,comm`
  - Linux uses much less memory to run processes

Why we study Virtual Memory

- A process can be larger than physical memory
- A process can execute even if all process info is not in memory
  - e.g., heap, stack, data…
- OS should allow more processes than memory size to increase the degree of multi-programming

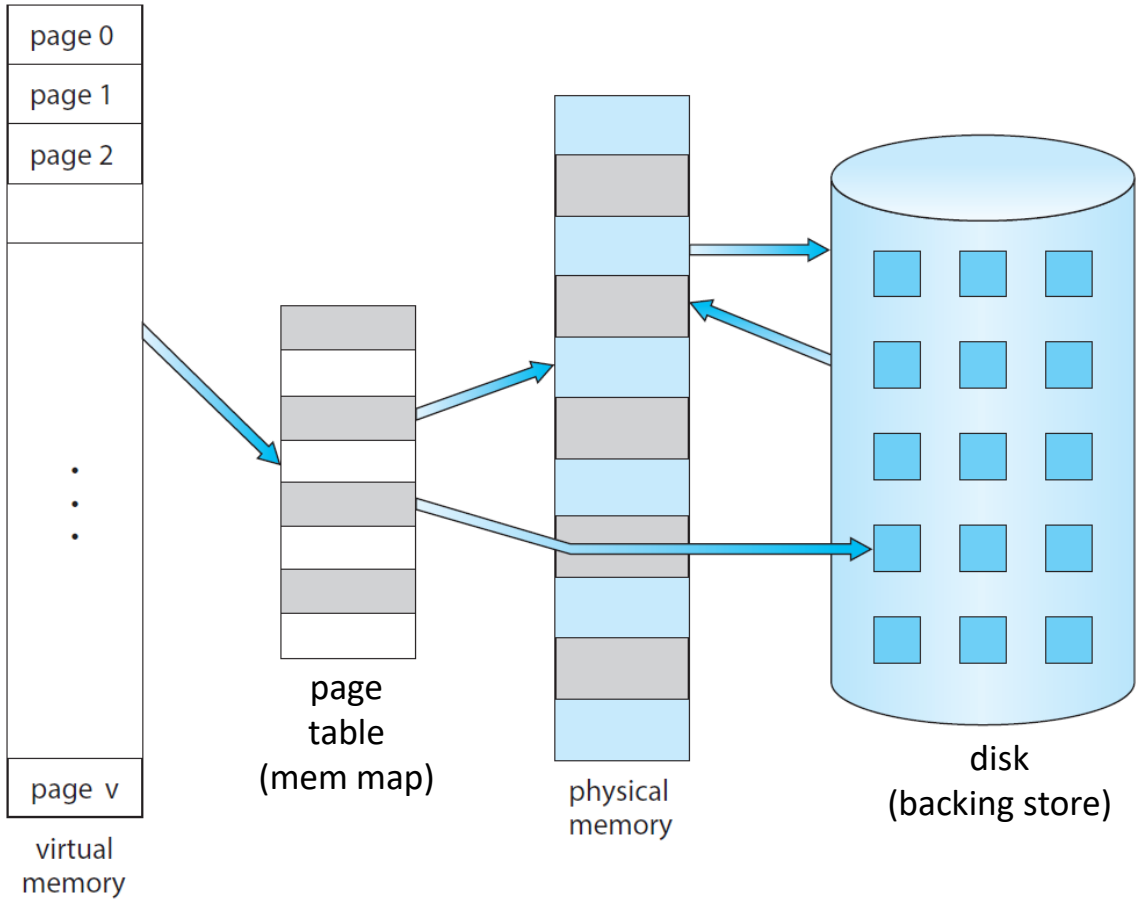
Virtual Memory

- **Virtual memory** – separation of user logical memory from physical memory
  - Only part of the program needs to be in memory for its execution – **90/10 rule!**
  - Logical address space can be much larger than physical address space
  - Allows address spaces to be shared by several processes
  - Allows for more efficient process creation
  - More programs running concurrently

Virtual Memory

- **Virtual address space** – logical view of how process is stored in memory
  - Usually start at address 0, contiguous addresses until end of space
  - Meanwhile, physical memory organized in page frames
  - MMU must map logical to physical
- **Demand Paging**

Virtual Memory > Physical Memory



Demand Paging

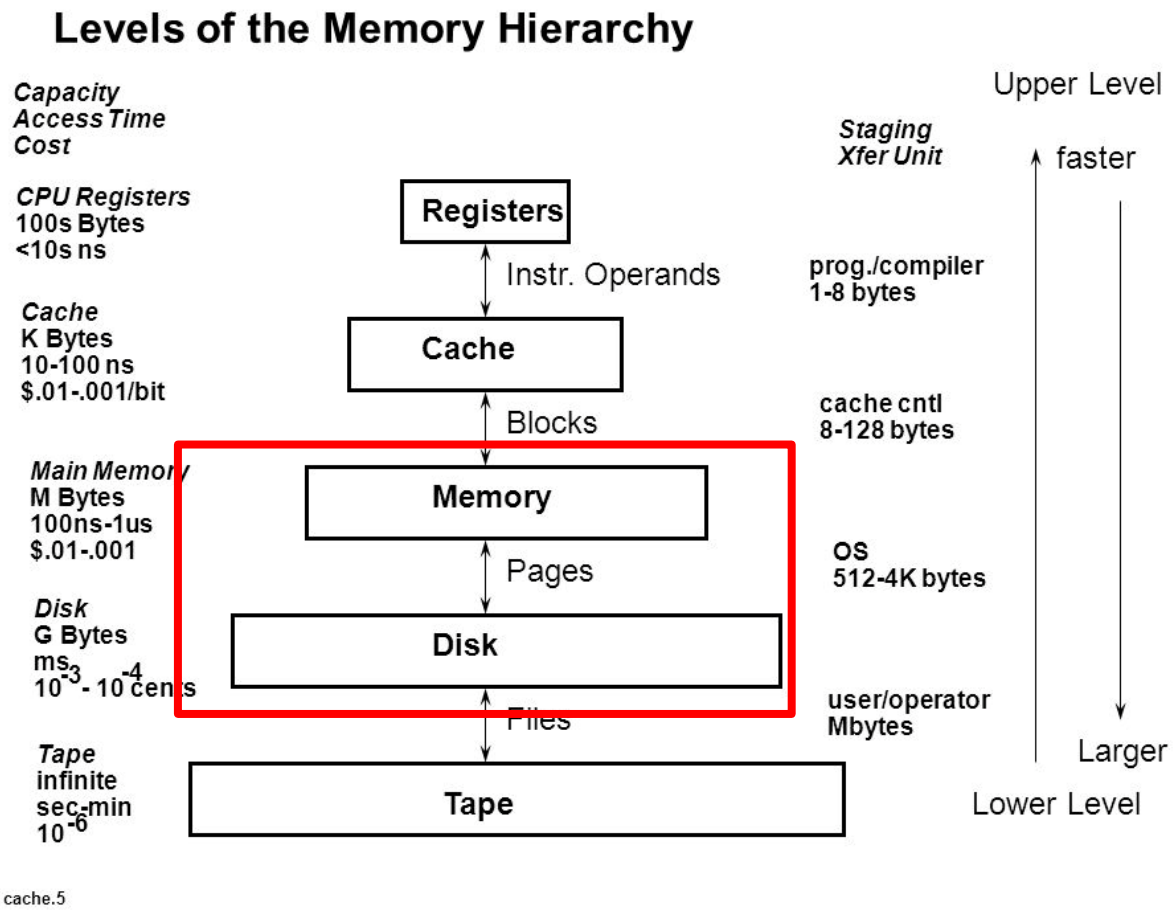
Virtual Memory > Physical Memory

- Code needs to be in memory to execute, but **entire program rarely used**
  - Error code, unusual routines, large data structures
- **Entire program code not needed** at the same time

Virtual Memory > Physical Memory

- Consider ability to execute partially-loaded program
  - Program no longer constrained by limits of physical mem
- Lots of potential benefits!
  - Each program takes less memory while running
    - Then, more programs run at the same time
    - Increase the degree of multiprogramming
    - Increased CPU utilization and throughput
  - Less I/O needed to load or swap programs into memory
    - Each user program runs faster

Demand Paging

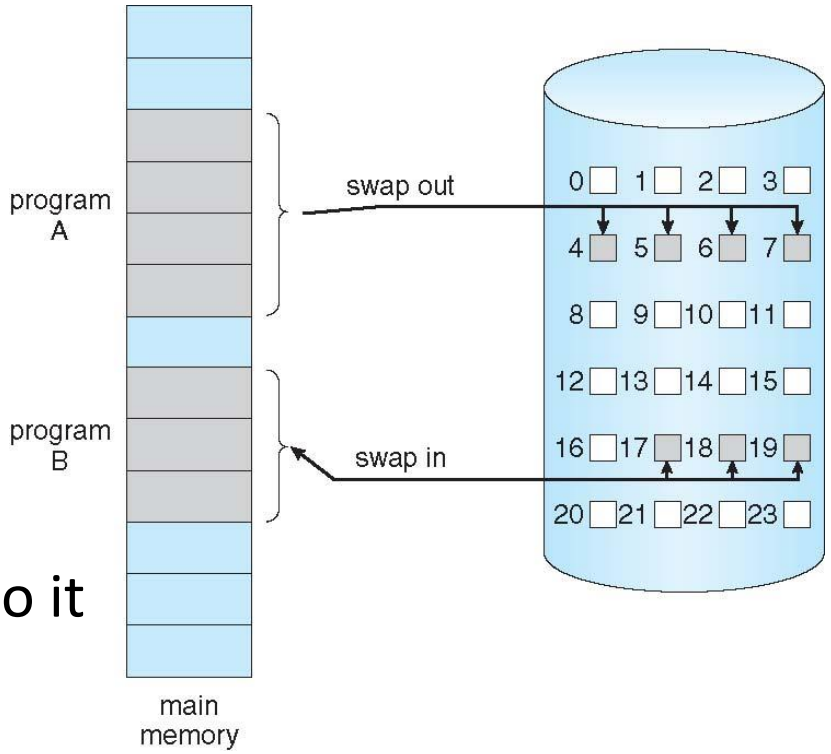


Demand Paging

- Demand Paging uses a memory as a cache of the disk
- Key idea
  - Code needs to be in memory to execute, but **entire program rarely used**
  - **90/10 rule** – Processes spend 90% of their time accessing 10% of their space in memory!
    - Locality and working set
    - Working set – small portion of process actually being used.

Demand Paging

- Bring a page into memory only when it is needed
  - Less I/O needed, no unnecessary I/O
  - Less memory needed
  - Faster response with higher degree of multi-programming
- Similar to paging system with swapping
- When a page is needed; reference to it
  - Invalid reference → abort & swap-in
  - Not-in-Memory → swap-in (bring it to memory)



Demand Paging – When to load?

- **At process start time**
  - Could bring entire process into memory at load time
  - Possible if virtual address space <= physical mem size
- **Pre-defined Paging**
  - Program (or app developer) tells when to load and remove pages
  - Good –
  - Bad –

Demand Paging – When to load?

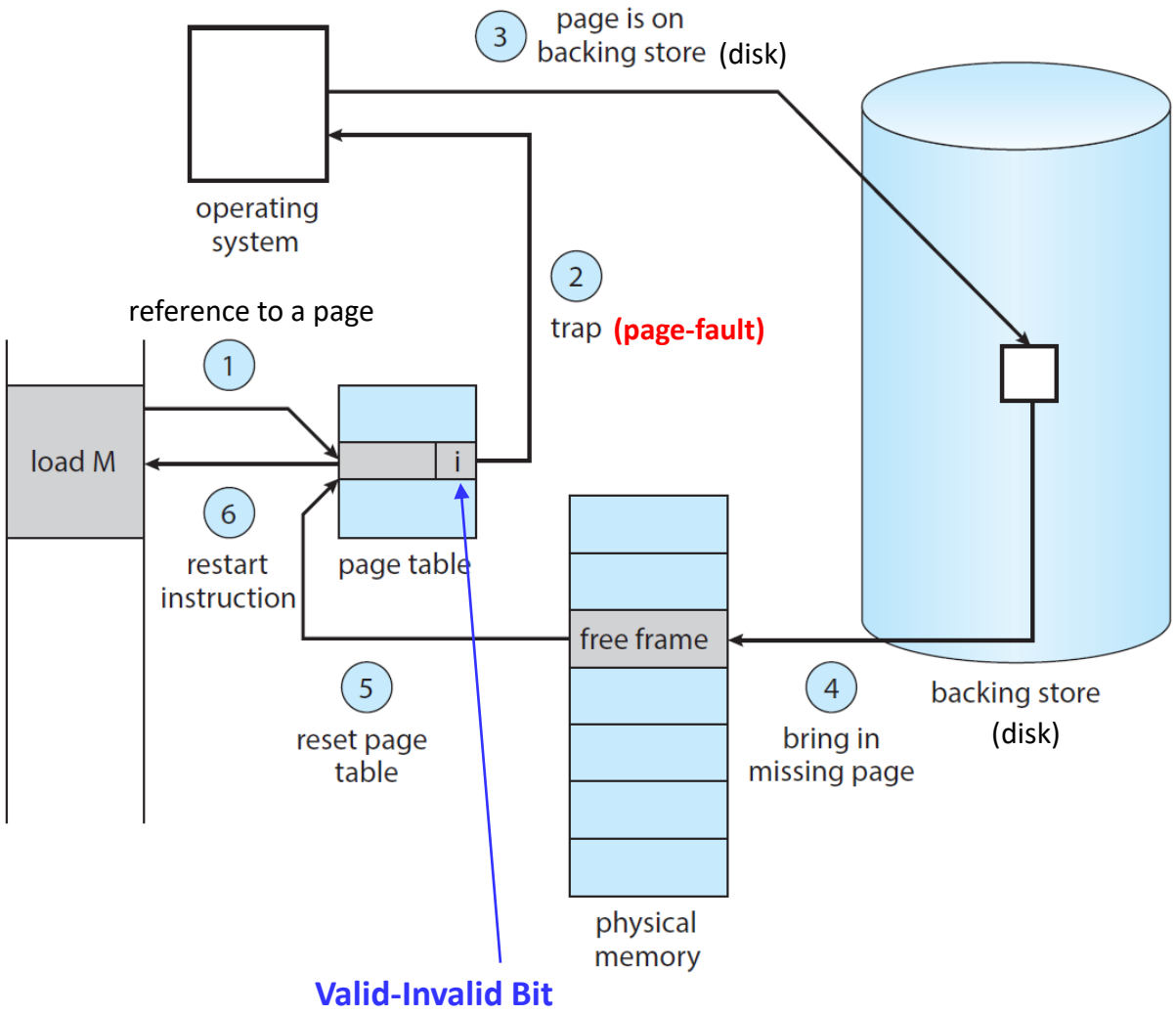
- **Pre-paging**
  - OS predicts which pages the process will need and pre-loads them into memory
  - Two different prediction cases
    - Which pages (in disk) will be needed
    - Which pages (in mem) will not be referenced in future
  - Potential Issues?
    - What if OS is wrong?
    - OS can load useless pages (not referenced in future)
    - OS can remove useful pages
  - Difficult to determine the right pages due to branches in code

Demand Paging – When to load?

What Demand Paging does is…

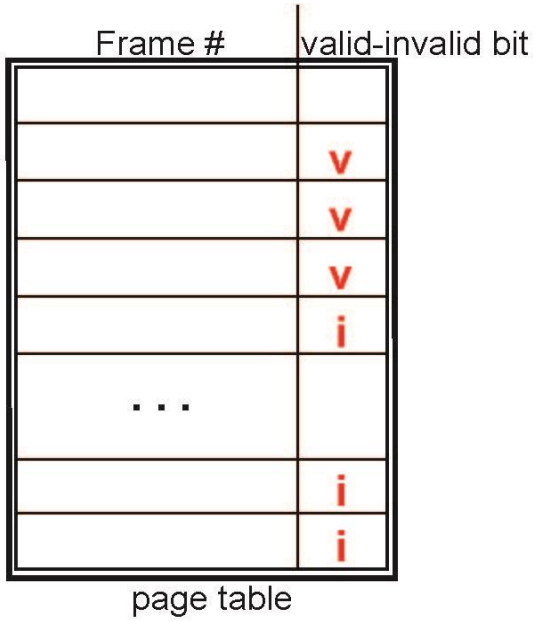
- OS loads a page the first time it is referred.
- Process must give up CPU while the page is being loaded
  - How long dose it take to load page from disk to mem?
  - Disk access time
- If page table is full?
  - **Page replacement!**
- ***Page-fault:*** interrupt that occurs when an instruction references a page that is not in memory
  - Page request → not in mem → go to disk and load it to mem
  - What is the cost of page fault?
  - *Disk access is expensive;* need to consider pre-paging again?

Demand Paging

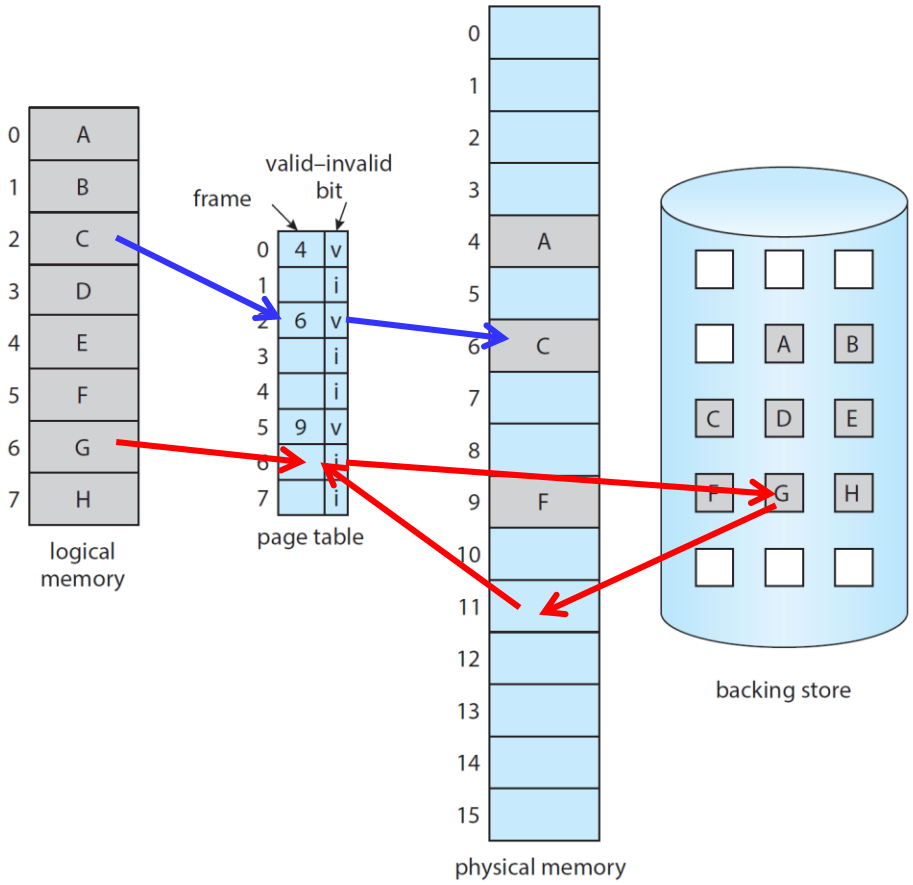


Valid-Invalid Bit

- With each page table entry a valid–invalid bit is associated
  - **v**: in-memory, **i**:not-in-memory (in disk)
- Initially valid–invalid bit is set to **i** on all entries
- During MMU address translation, if valid–invalid bit in page table entry is **i** ⇒ ***page fault***



Valid-Invalid Bit



Page Fault Handling

1. If there is a reference to a page, first reference to that page will trap to operating system
   - **Page fault**
2. Operating system looks at another table to decide:
   - **Invalid reference → abort (stop the instruction)**
   - **Just not in memory**
3. Find free frame in memory
4. Swap page into frame via scheduled disk operation
5. Reset tables to indicate page now in memory
   Set validation bit = ***v***
6. Restart the instruction that caused the page fault

HW Support for Demand Paging

- The hardware to support demand paging is the same as the hardware for paging and swapping
- Hardware support needed for demand paging
  - Page table with valid / invalid bit
  - Secondary memory (Disk) – swap device with **swap space**
  - Instruction restart

Swap Space

- What happens when a page is removed from memory?
  - If the page contained **code**, we could simply remove it since it can be reloaded from the **disk**
  - If the page contained **data**, we need to save the data so that it can be reloaded if the process it belongs to refers to it again **(swap)**
  - ***Swap Space***: A portion of disk is reserved for storing pages that are evicted from memory
- At any given time, a page of virtual mem *might* exist in one or more of:
  - The file system
  - Physical memory
  - Swap space (not part of file system)
- Page table must be more sophisticated so that it knows where to find a page.