SCS 1214: Operating Systems

Dr. Dinuni Fernando, PhD

Based on Operating System Concepts by A.Silberschatz, P.Galvin, and G.Gagne

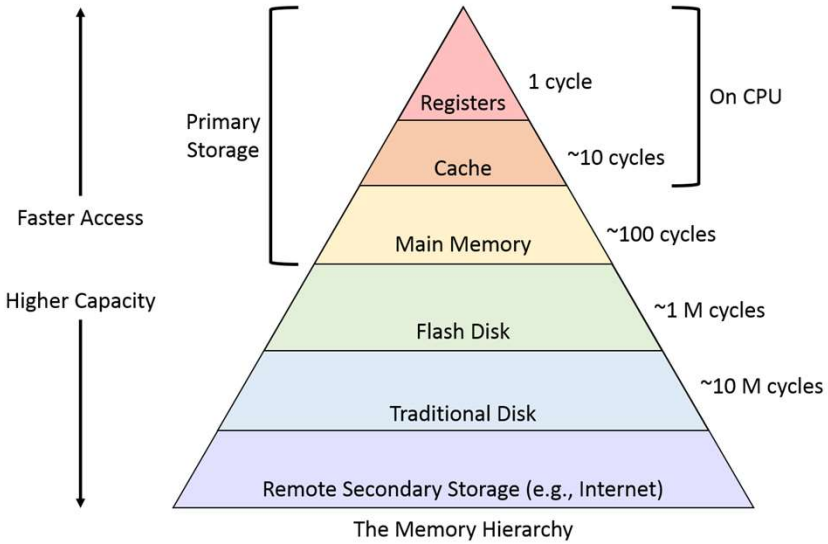
Page Replacement Algorithms

Learning Objectives

- To describe the benefits of a virtual memory system
- To explain the concepts of page tables, demand paging, page-replacement algorithms, and allocation of page frames

Background

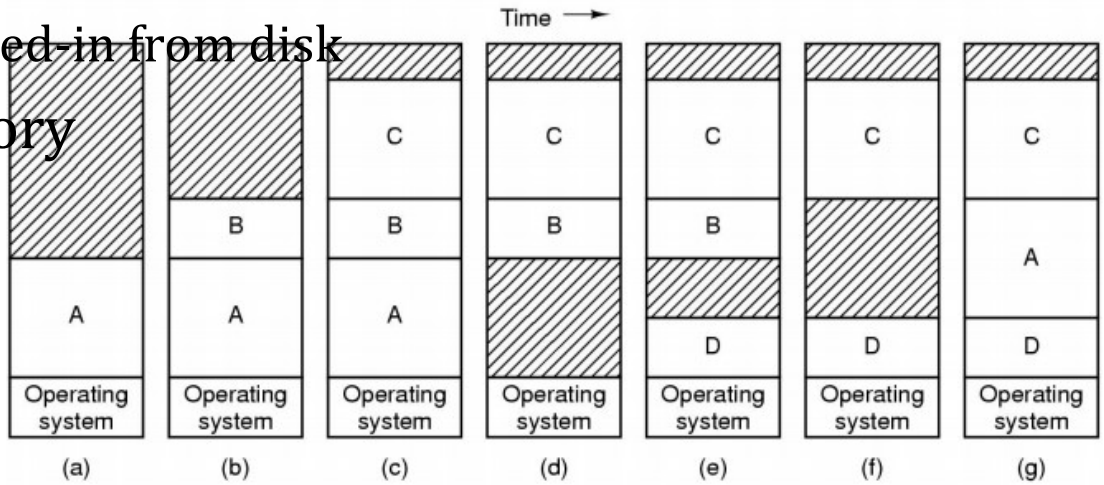
- Ideally programmers want memory to be
  - Large
  - Fast
  - Persistent (Non-volatile)



The Memory Hierarchy

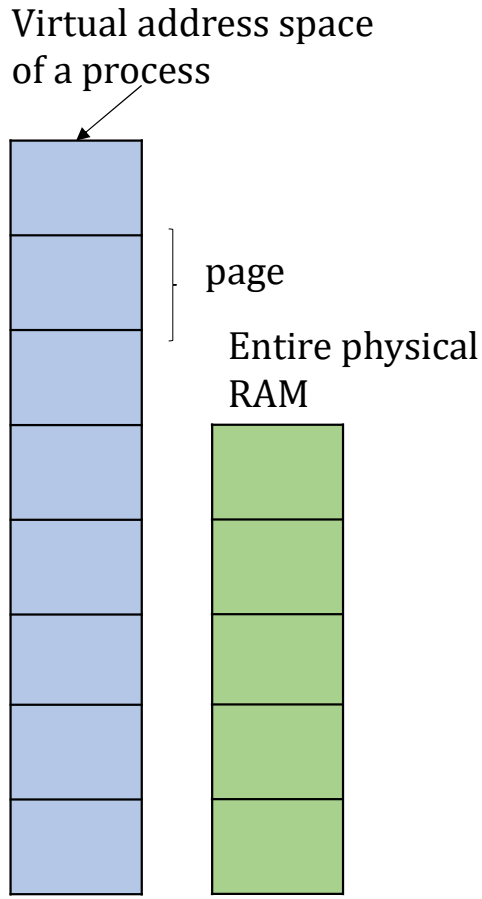
What if physical memory is not enough to hold all processes? Swapping

- Physical memory may not be enough to accommodate the needs of all processes
- Memory allocation changes as
  - processes come into memory
  - leave memory and are swapped out to disk
  - Re-enter memory by getting swapped-in from disk
- Shaded regions are unused memory



Virtual Memory

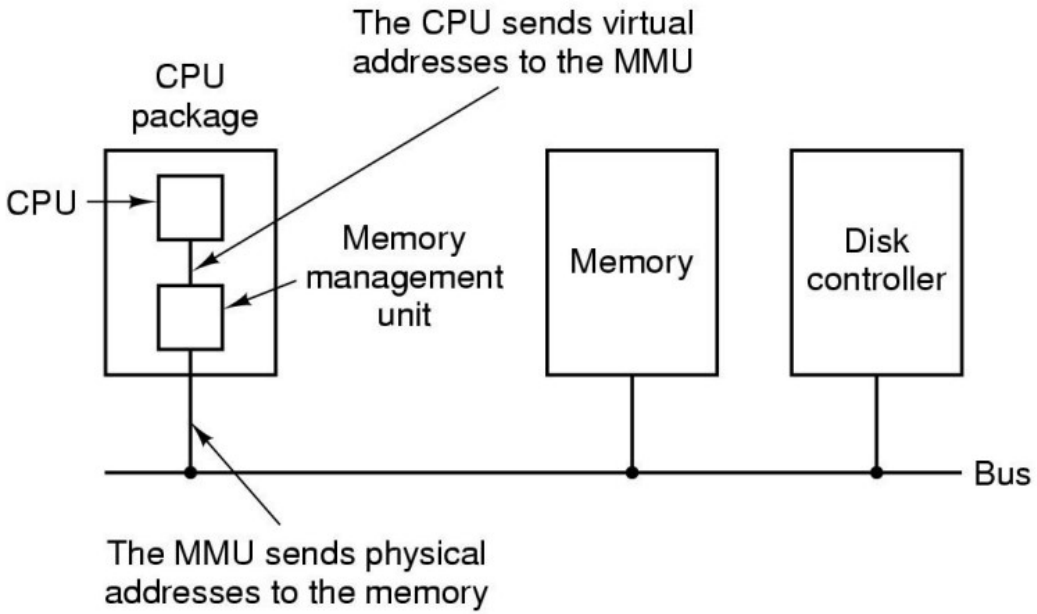
- Virtual Memory – each process gets an illusion that it has more memory than the physical RAM in the system.
- Swapping the memory of an entire process is useful when the sum of memory needed by all processes is greater than the total RAM available in the system.
  - But, sometimes, a single process might require more memory than the total RAM in the system.
  - In this situation, swapping an entire process is not enough.
  - Rather, we break up memory space of a process into smaller equal-sized pieces called pages.
  - OS decide which page need to say in memory and which should move to disk.



Virtual Memory (cont.)

- **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 address space.
- Virtual memory can be implemented via:
  - Demand paging
  - Demand segmentation

Memory Management Unit

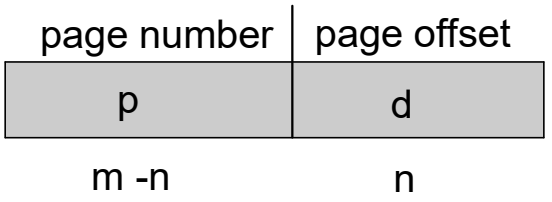


- MMU is a hardware module that accompanies the CPU
- It translates the Virtual Address used by executing instructions to Physical Addresses in the main memory.

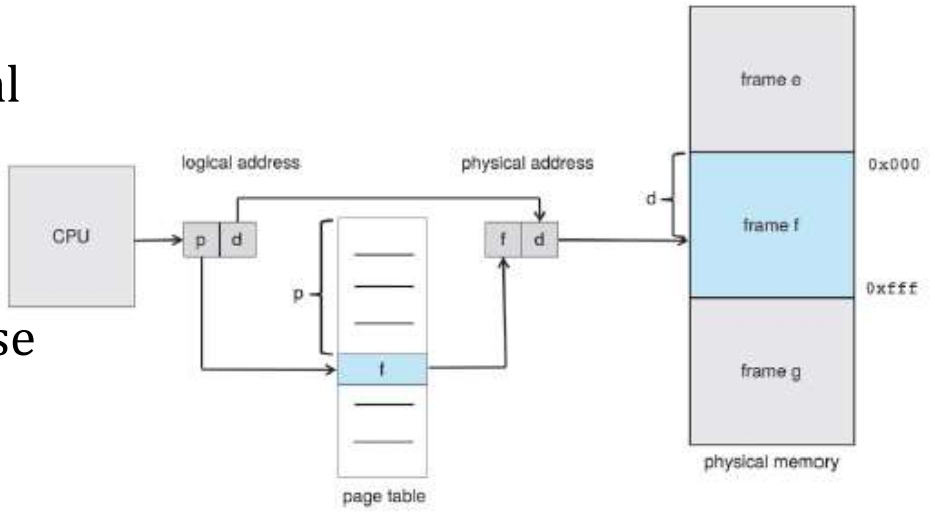
Paging

- Paging – a memory management scheme that permits a process's physical address space to be non-contiguous.
- Paging avoids
  - External fragmentation
  - Need for compaction
- Implemented through cooperation between OS and computer hardware.
- Basic method
  - Break physical memory into fixed-size blocks called frames.
  - Break logical memory into blocks of the same size called pages.
  - When a process is to be executed, its pages are loaded into available memory frames from their source (FS / BS).
  - BS is divided into fixed-sized blocks that are the same size as the memory frames.
- Set up a page table to translate logical to physical addresses.
- Still have internal fragmentation

Paging (cont'd) – paging model



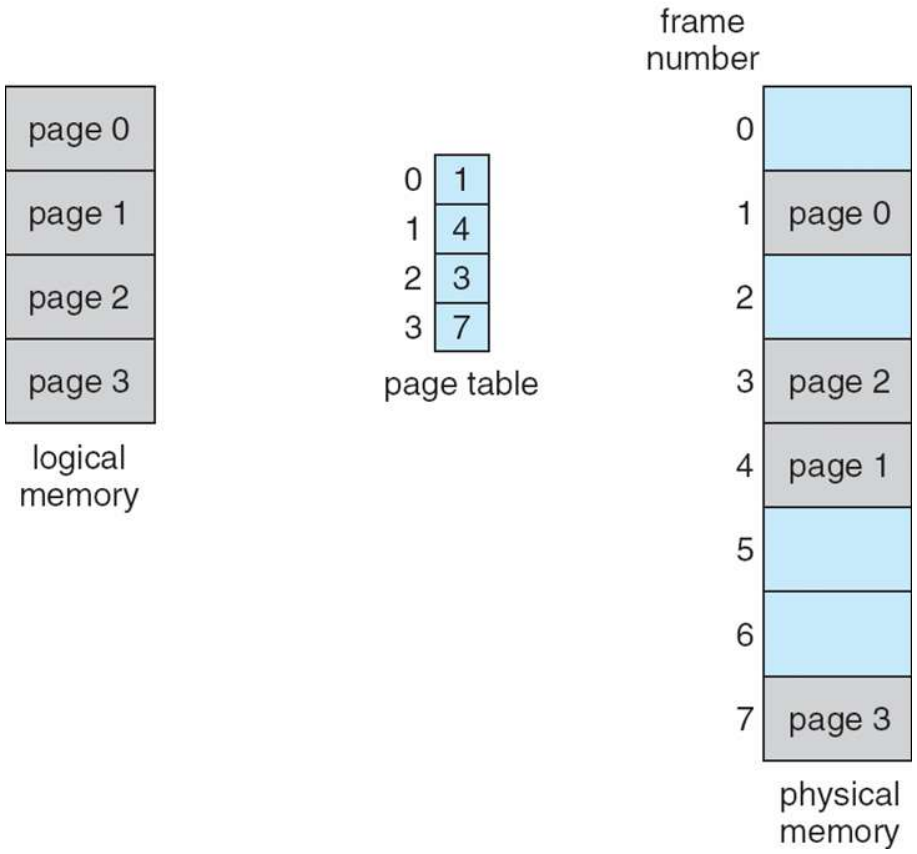
- Every address generated by CPU divided into 2 parts.
  - **Page number** (***p***) – used as an index into a **page table** which contains base address of each page in physical memory
  - **Page offset** (***d***) – combined with base address to define the physical memory address that is sent to the memory unit
- Page table contains,
  - Base address of each frame in physical memory,
  - Offset – location in the frame being referenced.
  - Physical address – combination of base address of the frame and page offset



Translation of logical address to physical address

1. Extract the page number p and use it as an index into the page table.
2. Extract the corresponding frame number f from the page table.
3. Replace the page number p in the logical address with the frame number f.

Paging model of physical and logical memory



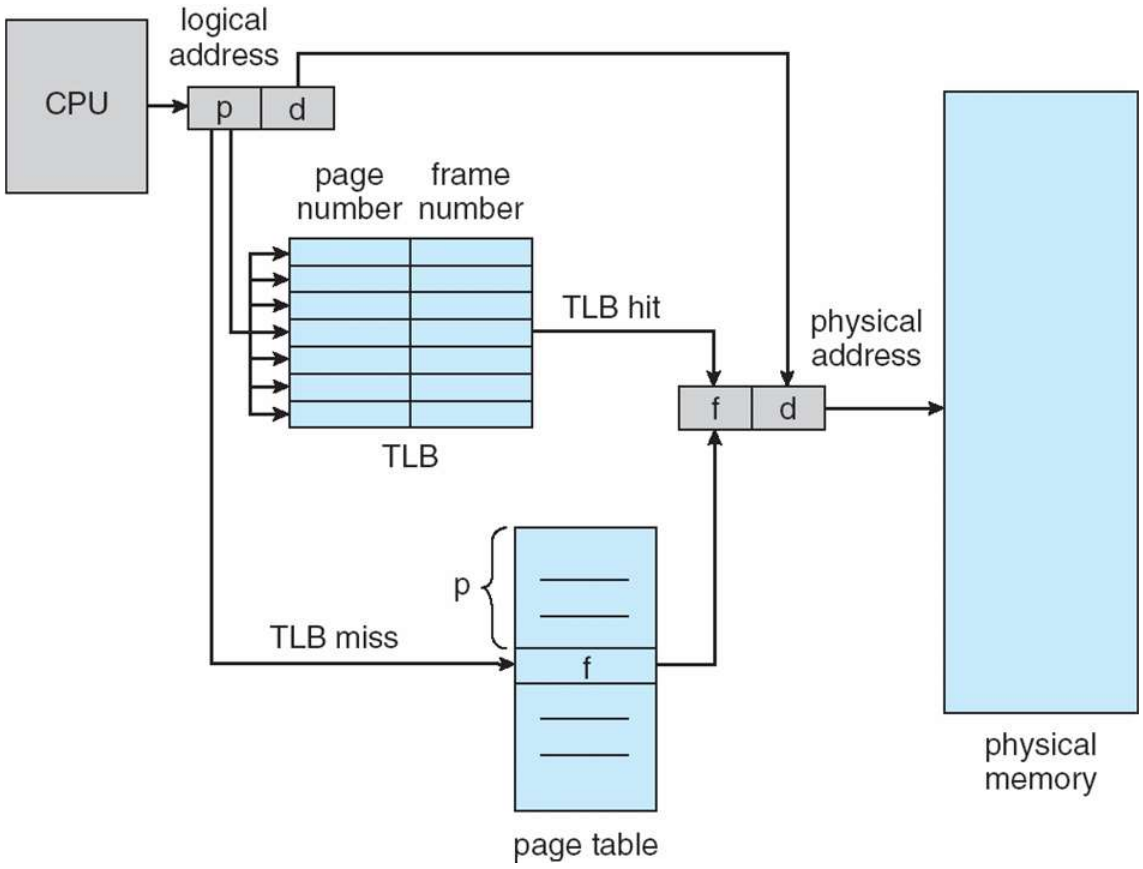
Implementation of page table

- Page table is kept in main memory
  - **Page-table base register** (**PTBR**) points to the page table
  - **Page-table length register** (**PTLR**) indicates size of the page table
- In this scheme every data/instruction access requires two memory accesses
  - One for the page table and one for the data / instruction
- The two-memory access problem can be solved by the use of a special fast-lookup hardware cache called **translation look-aside buffers** (**TLBs**) (also called **associative memory**).

Translation Look-Aside Buffer

- Some TLBs store **address-space identifiers** (**ASIDs**) in each TLB entry – uniquely identifies each process to provide address-space protection for that process
  - Otherwise need to flush at every context switch
- TLBs typically small (64 to 1,024 entries)
- On a TLB miss, value is loaded into the TLB for faster access next time
  - Replacement policies must be considered
  - Some entries can be **wired down** for permanent fast access

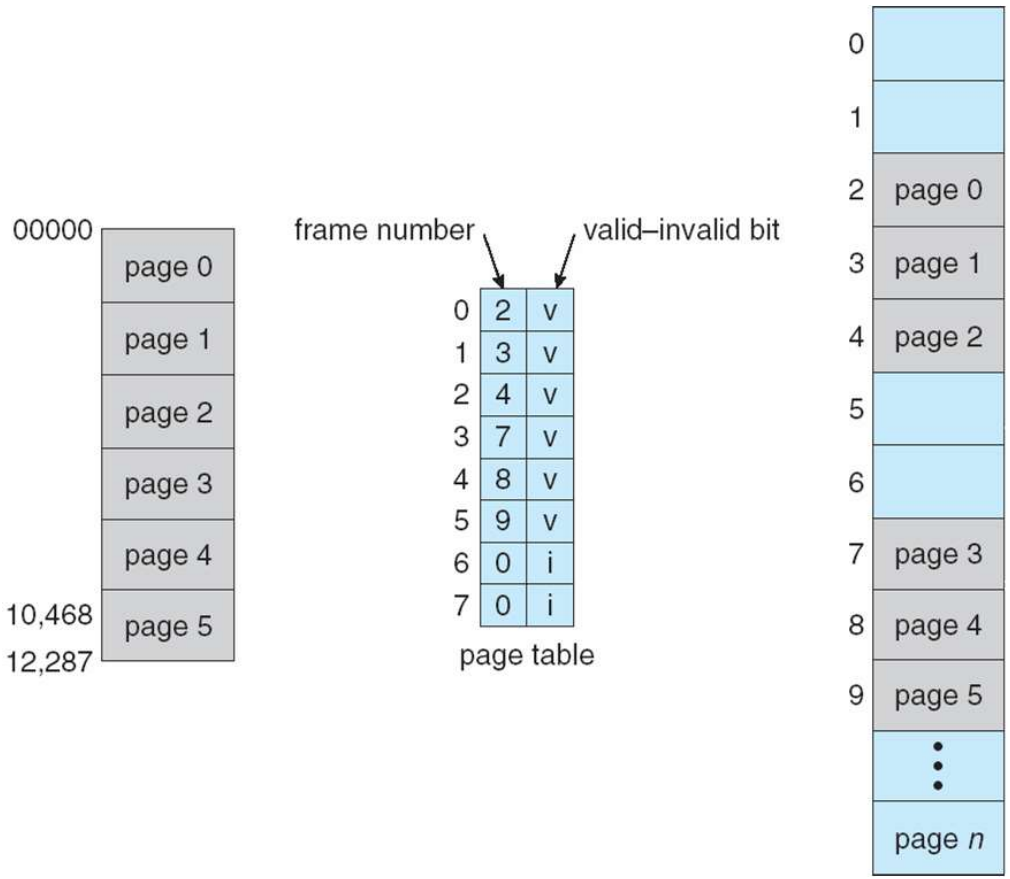
Paging hardware with TLB



Cold Start Penalty

- Immediately after a context switch, the incoming process may find that all (or majority) of TLB entries are invalid.
  - On some x86 processors, TLB has to be "flushed" upon every context switch because there is no field in TLB to identify the process context.
- Cold Start Penalty = Cost of repopulating the TLB (and other caches) upon a context switch.
  - Every memory access results in a TLB miss.
  - MMU walks the page-table to repopulate the missing TLB entry.
  - Handling a TLB miss has a high latency, affecting the performance of incoming process, hence the term "penalty".

Valid(v) and invalid(i) bit in a page table

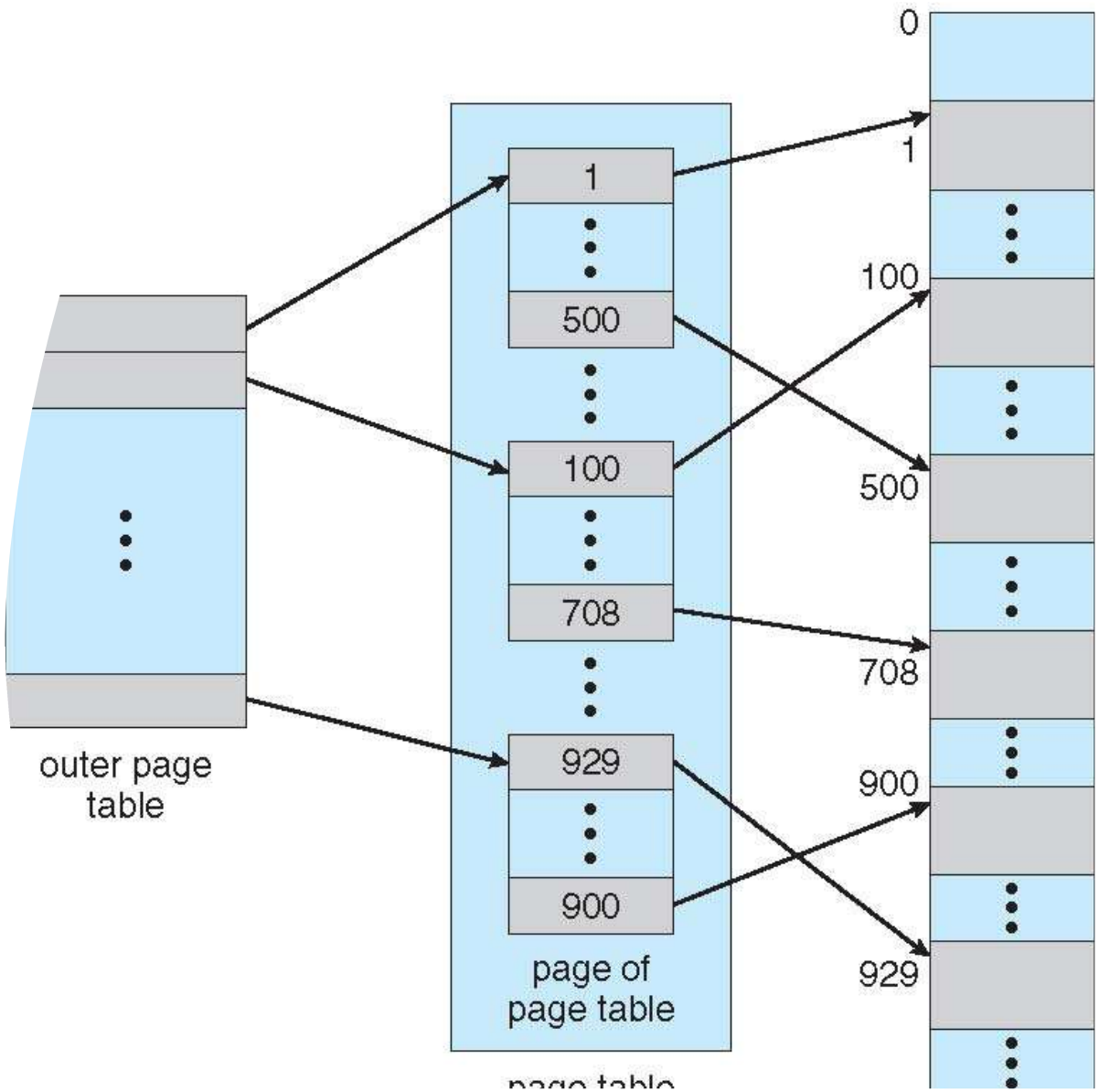


Structure of the Page Table

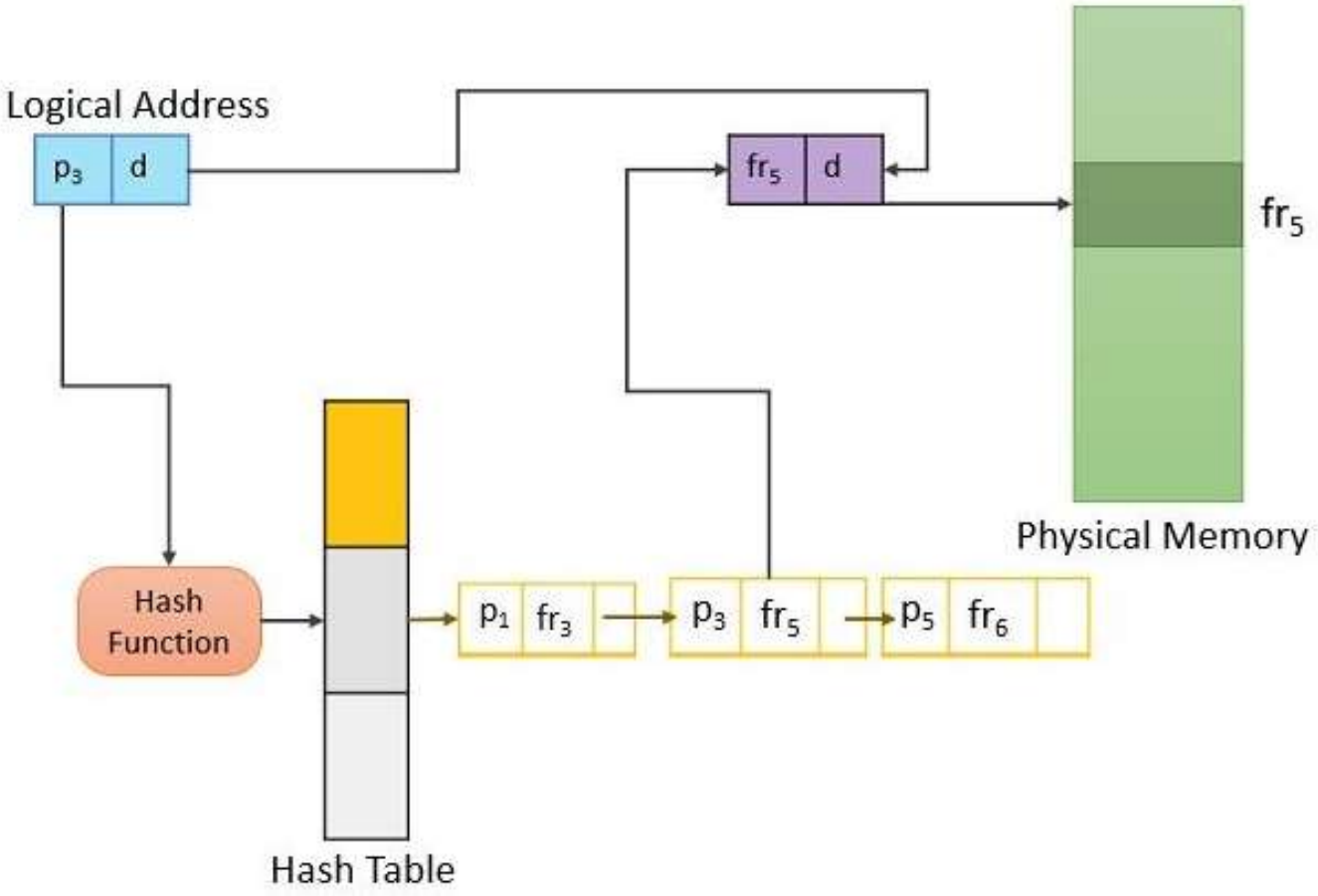
- Memory structures for paging can get huge using straight-forward methods
  - Consider a 32-bit logical address space as on modern computers
  - Page size of 4 KB (2^{12})
  - Page table would have 1 million entries (2^{32} / 2^{12})
  - If each entry is 4 bytes ➔ each process 4 MB of physical address space for the page table alone
    - Don't want to allocate that contiguously in main memory
  - One simple solution is to divide the page table into smaller units
    - Hierarchical Paging
    - Hashed Page Tables
    - Inverted Page Tables

Hierarchical Page Tables

- 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



Hash Page table



Inverted Page Table

- Rather than each process having a page table and keeping track of all possible logical pages, track all physical pages
- One entry for each real page of memory
- Entry consists of the virtual address of the page stored in that real memory location, with information about the process that owns that page
- Decreases memory needed to store each page table, but increases time needed to search the table when a page reference occurs

Inverted Page Table Architecture