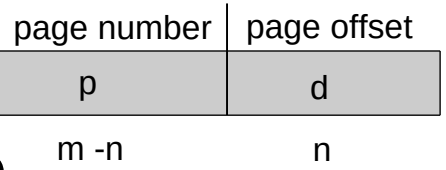
L25- IMPLEMENTATION OF PAGING

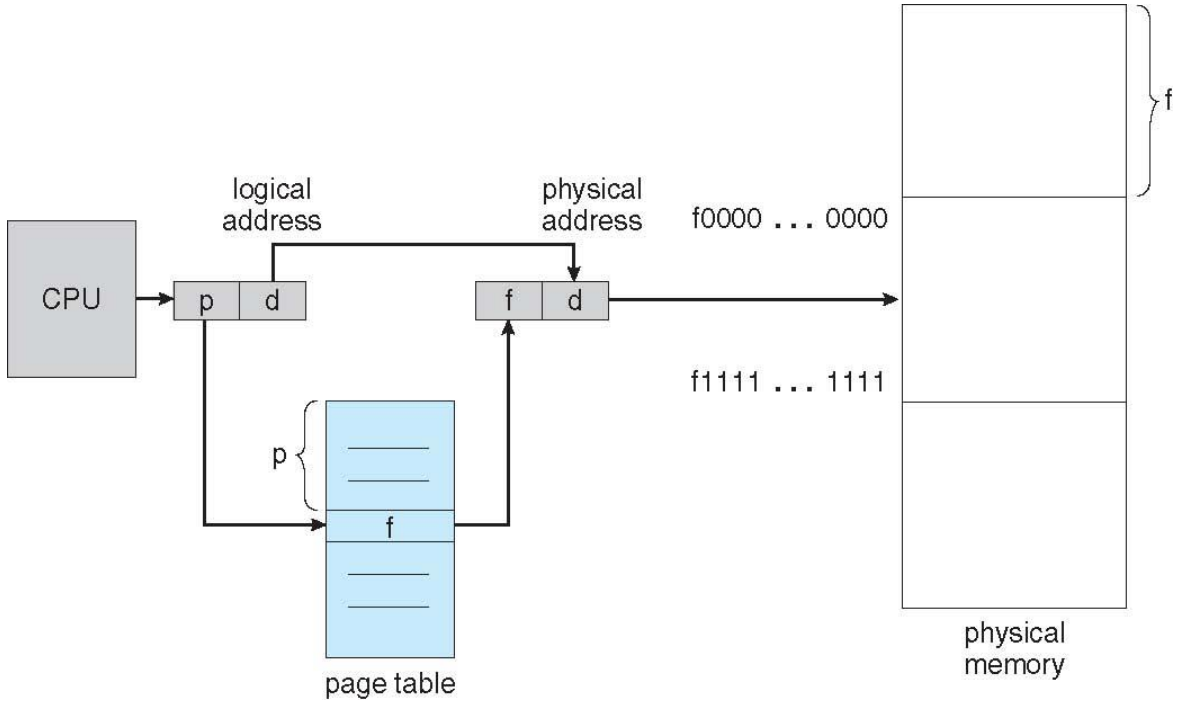
Overview of Memory Management

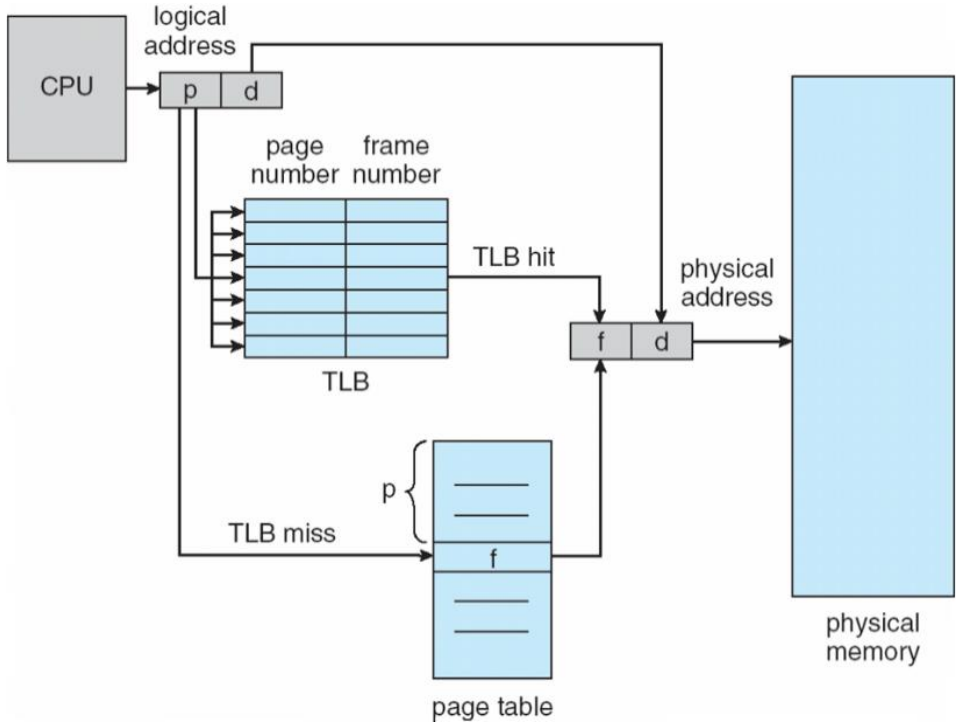
- Background
- Swapping
- Contiguous Memory Allocation
- Segmentation
- Paging
- Structure of the Page Table

Paging & Address Translation Scheme

- Address generated by CPU is divided into:
  - **Page number (*p*)** – used as an index into a page table
  - **Page offset (*d*)** – displacement within a page







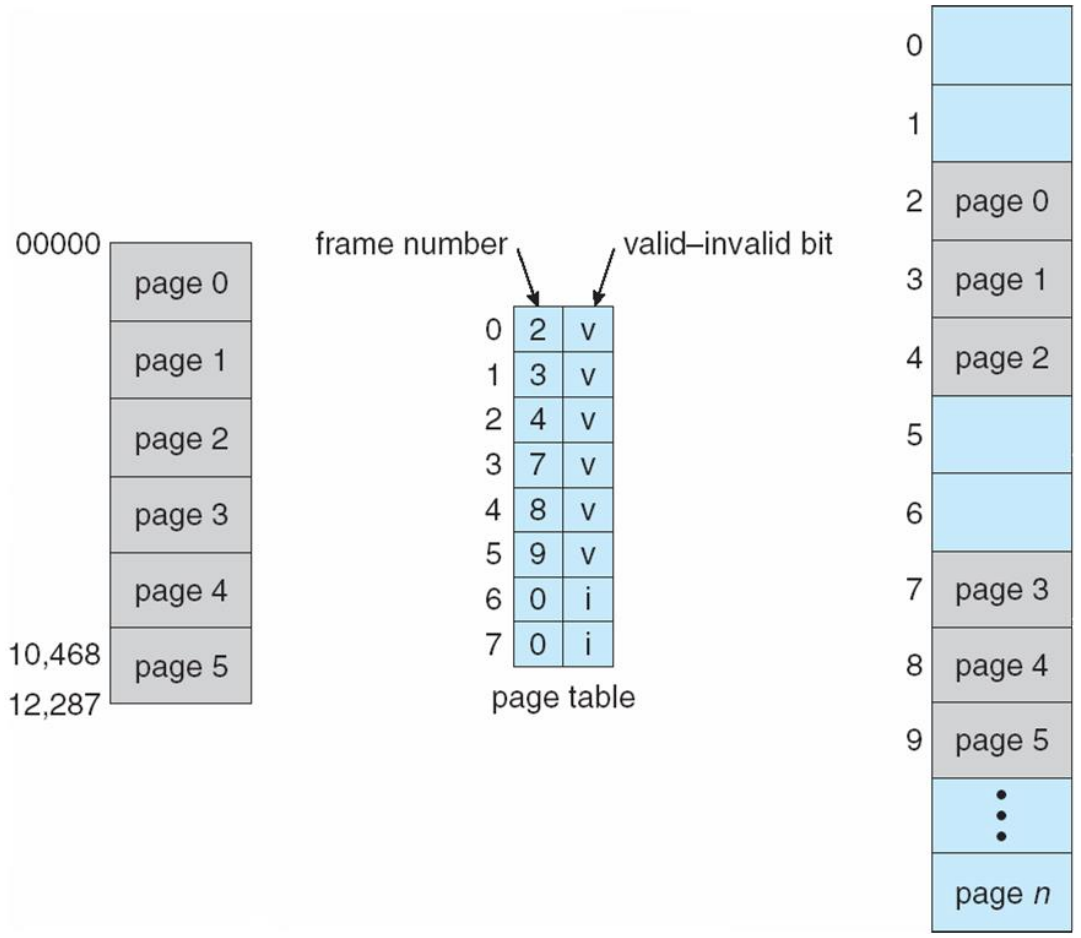
Implementation of Page Table

- 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
- Frequently used page table entries are kept in a fast-lookup hardware cache called **associative memory** or **translation look-aside buffers** (**TLBs**)

Memory Protection

- Memory protection implemented by associating protection bit with each frame to indicate if read-only or read-write access is allowed
  - Can also add more bits to indicate page execute-only, and so on
- **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
- Any violations result in a trap to the kernel

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



Structure of the Page Table

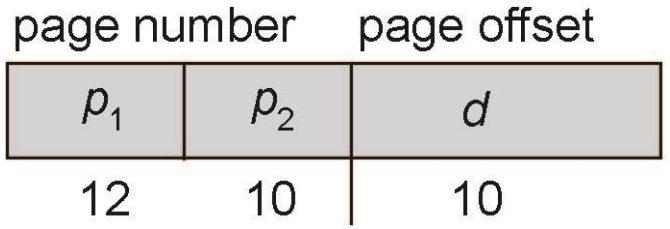
- Memory structures for paging can get huge using straight-forward methods
  - Consider a 32-bit logical address space as on modern computers
  - Page size of 4 KB (2^{12})
  - Page table would have 1 million entries (2^{32} / 2^{12})
  - If each entry is 4 bytes -> 4 MB of physical address space / memory for page table alone
- Hierarchical Paging
- Hashed Page Tables
- Inverted Page Tables

Hierarchical Page Tables

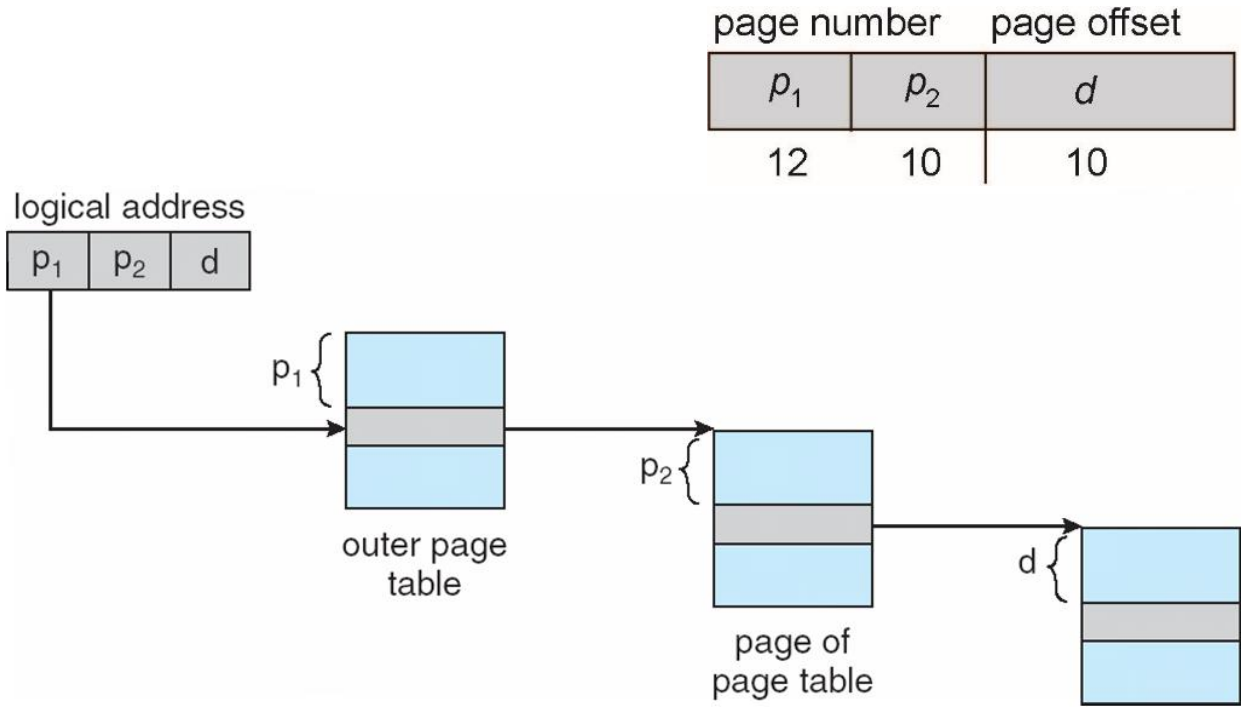
- Break up the logical address space into multiple page tables
- A simple technique is a two-level page table
- We then page the page table

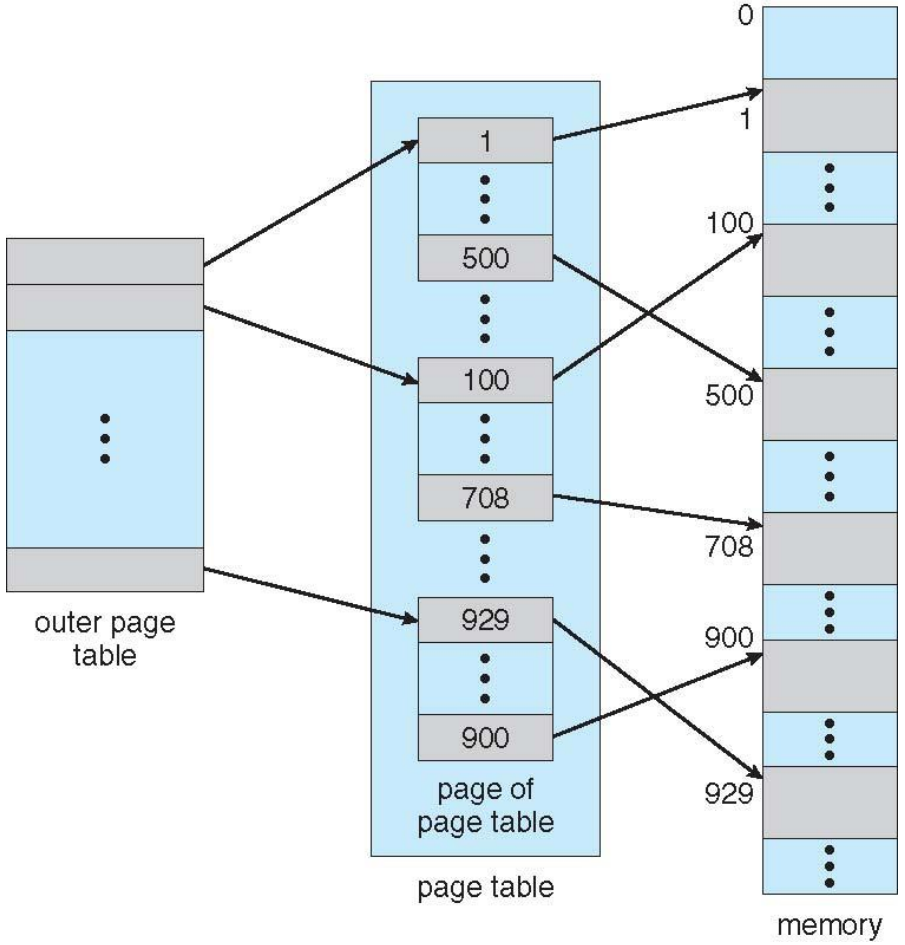
Simple Two-Level Paging

- A logical address (on 32-bit machine with 1K page size) is divided into:
  - a page number consisting of 22 bits
  - a page offset consisting of 10 bits
- Since the page table is paged, the page number is further divided into:
  - a 12-bit page number
  - a 10-bit page offset



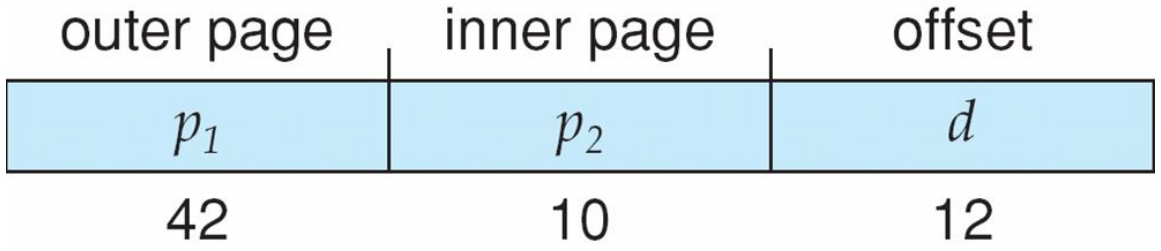
Address-Translation Scheme



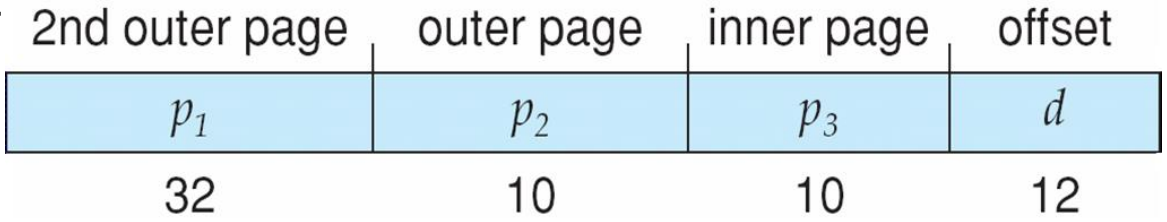


Larger Logical Address Space

- For 64 bits, even two-level paging scheme not sufficient
- If page size is 4 KB (2^{12})
  - Then page table has 2^{52} entries
  - Address would look like



  - Outer page table has 2^{42} entries or 2^{44} bytes



Hashed Page Tables

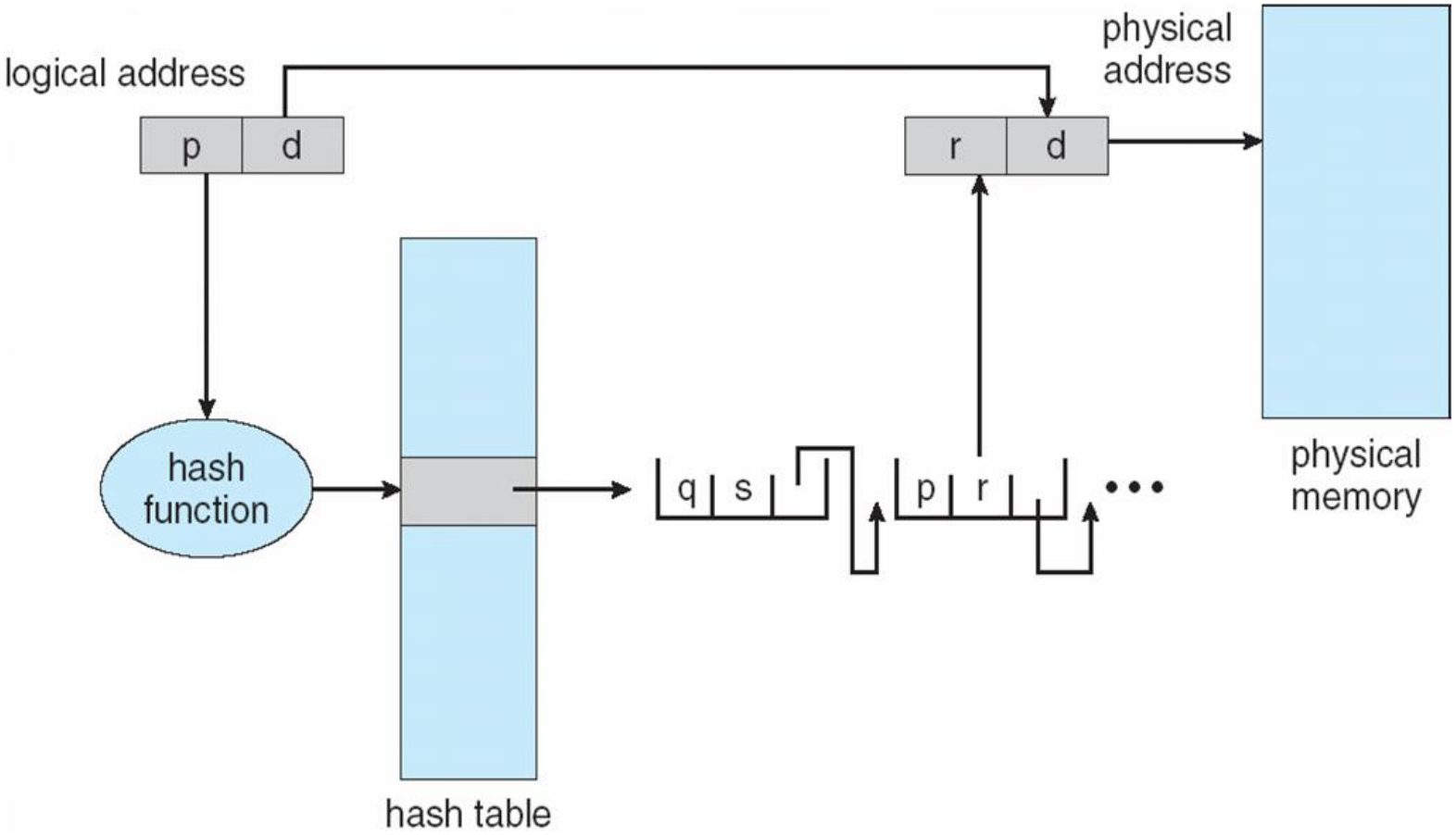
- The virtual page number is hashed into a page table
- This page table contains a chain of elements hashing to the same location
- Each element contains

  (1) the virtual page number

  (2) the value of the mapped page frame

  (3) a pointer to the next element
- Virtual page numbers are compared in this chain searching for a match
- If a match is found, the corresponding physical frame is extracted

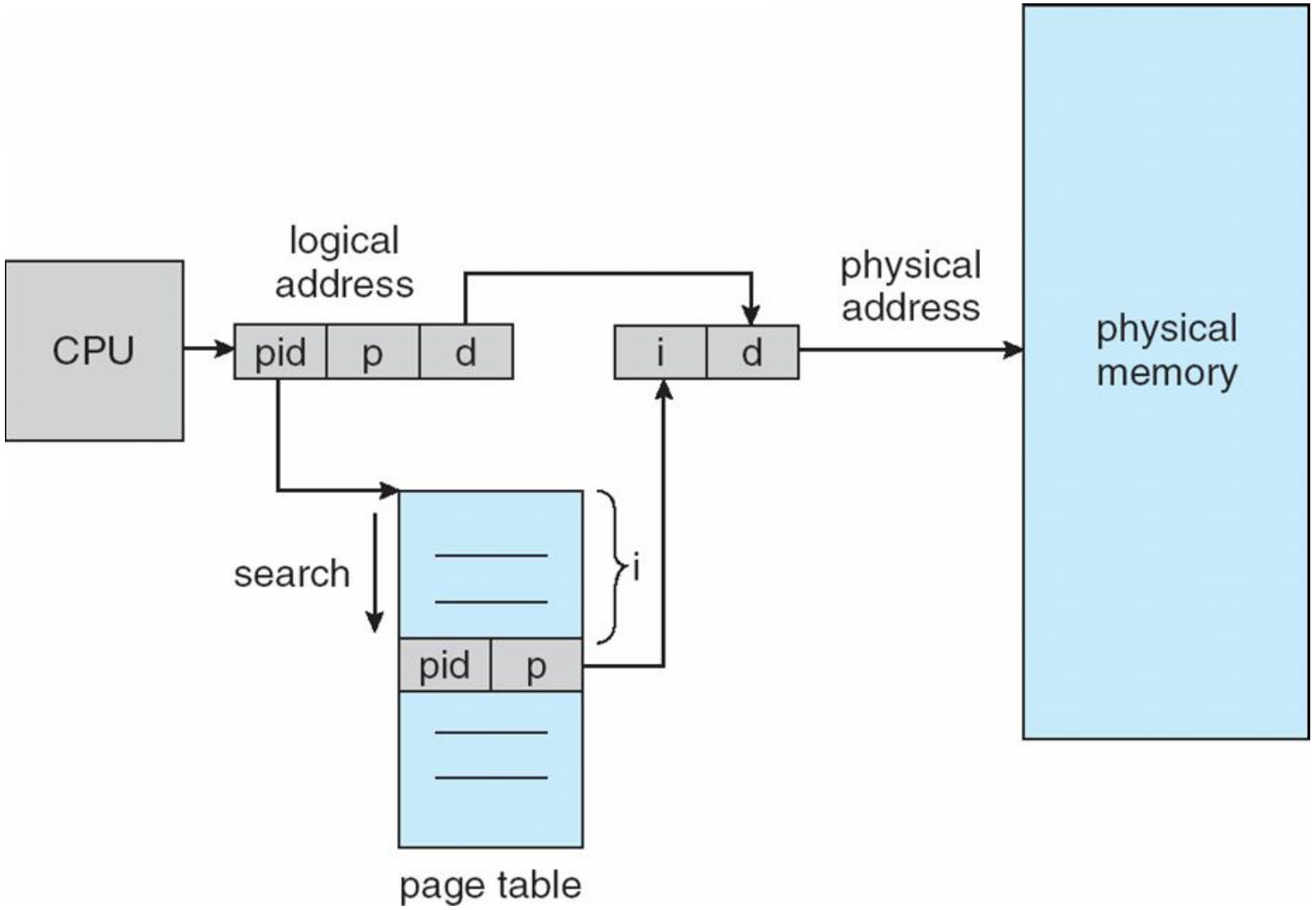
Hashed Page Tables



Inverted Page Table

- No page table per process
- 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,
- Increases time needed to search the table when a page reference occurs

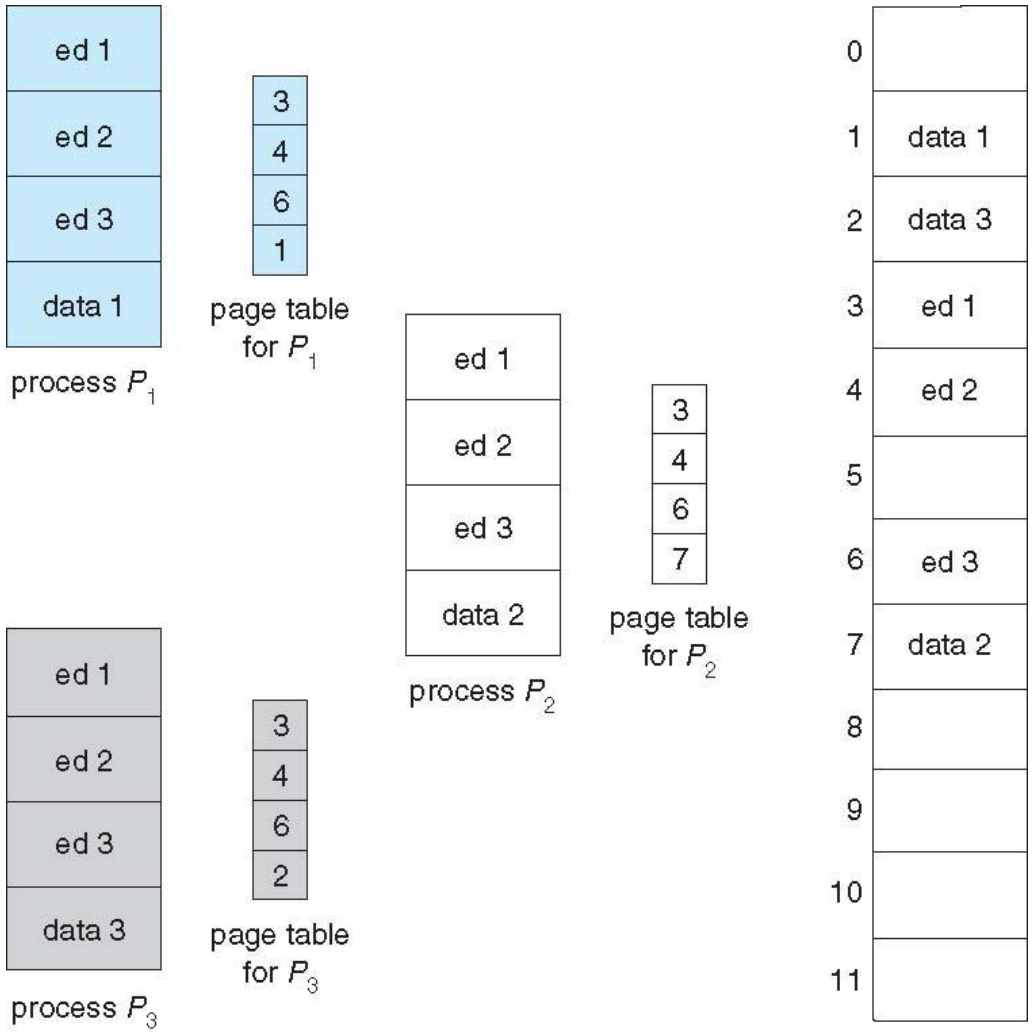
Inverted Page Table Architecture



Shared Pages

- **Shared code**
  - One copy of read-only (**reentrant**) code shared among processes (i.e., text editors, compilers, window systems)
  - Similar to multiple threads sharing the same process space
  - Also useful for interprocess communication if sharing of read-write pages is allowed
- **Private code and data**
  - Each process keeps a separate copy of the code and data
  - The pages for the private code and data can appear anywhere in the logical address space

Shared Pages



Thank You