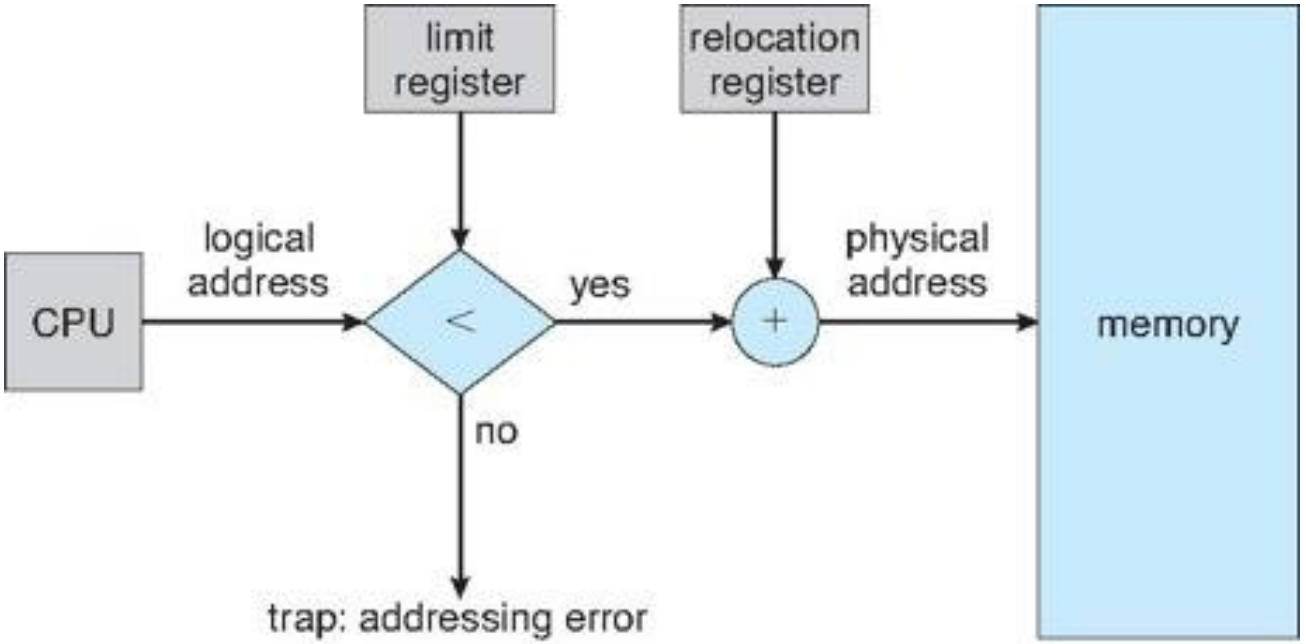
# Chapter 9: Memory Management – Paging overview

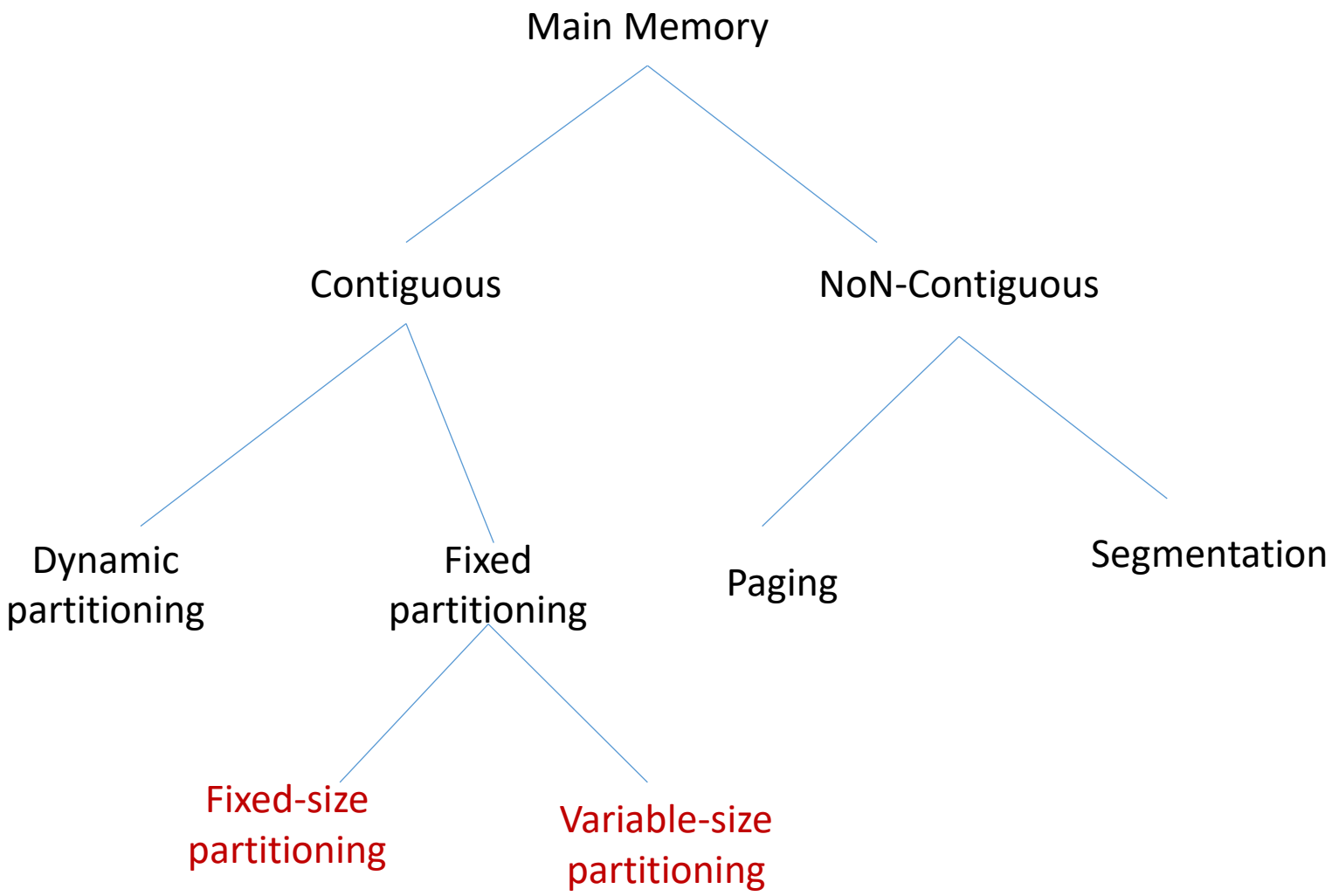
# Recall: Address Translation

- Logical addresses generated for each program
  - May have the same address space
  - MMU translates logical addresses into physical addresses

- Translation example



# Recall: Memory Allocation



# Paging

- Non-contiguous allocation approach
  - Allows parts of program to load in non-adjacent locations in memory
  - Resolves external fragmentation
    - Allows program parts to fill varying-sized memory holes (fragments)

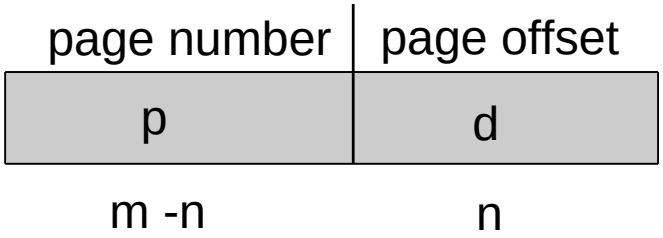
- Pages: Fixed and equal-sized blocks of logical address space
  - Size typically in powers of 2; 4 KB (2^10 bytes) to 4 MB (2^22 bytes) per page
  - In Linux: type getconf PAGESIZE

- Frames: Fixed-sized blocks of the physical memory

- Paging
  - Loads pages of programs into any available frames in the memory
  - Program of size N pages loaded in N free frames
  - Page size is equal to frame size

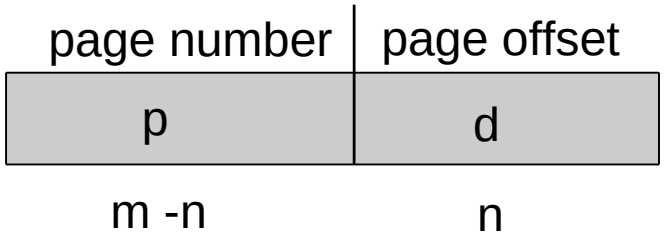
# Page Number and Offset

- Logical address division
  - $2^m$ logical addresses divided into pages of size $2^n$
  - m-n most significant bits of any address ⇒ Page number (p)
  - n least significant bits of any address ⇒ Offset (d) inside this page



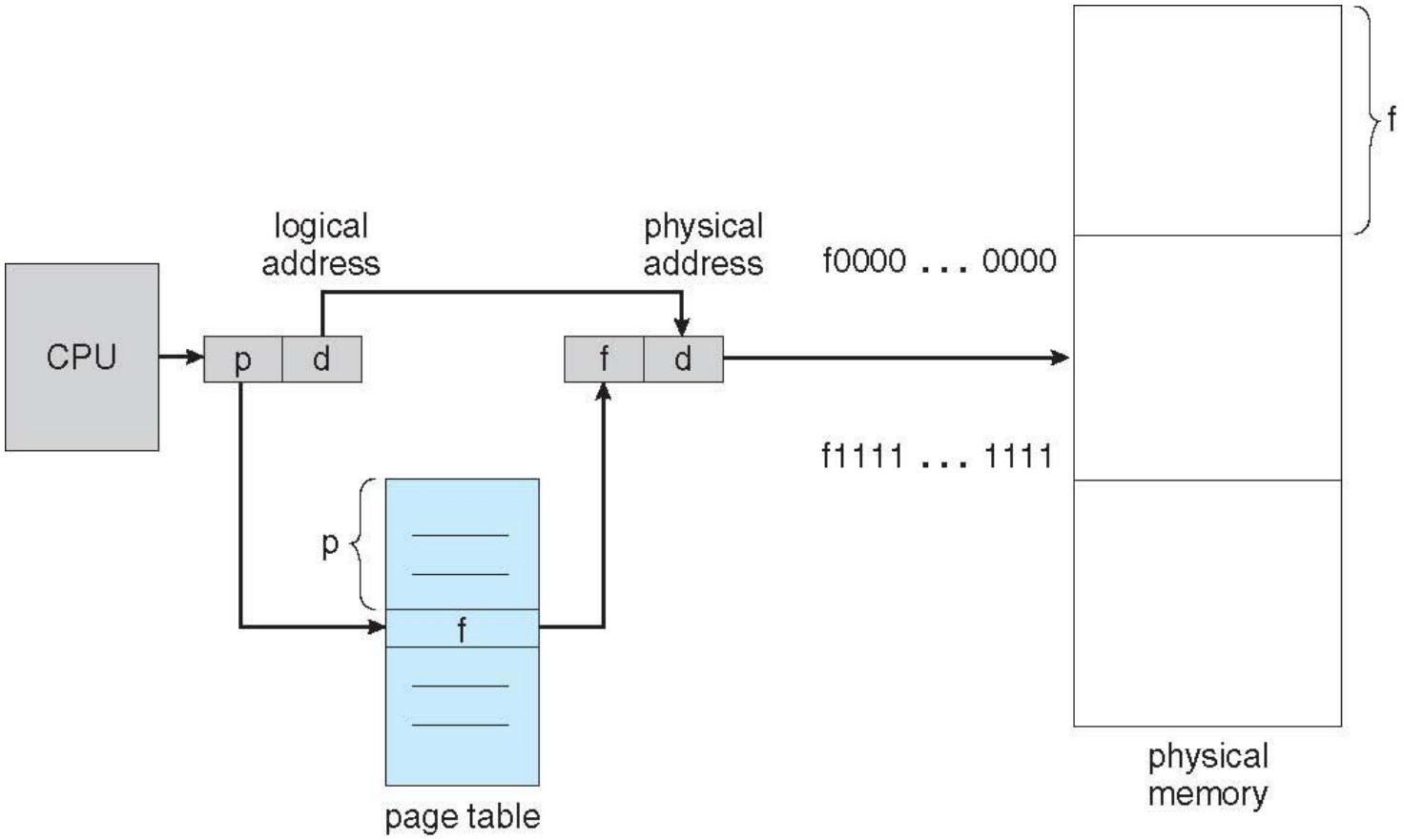

- Page number (p) – used as an index into a page table which contains base address of each page in physical memory
- Page offset (d) – combined with base address to define the physical memory address that is sent to the memory unit

# Page Number and Offset




- Example
  - 64 kB logical space divided into pages of 1 kB ⇒ m = 16, n = 10
  - No. of pages = $2^{(16-10)} = 2^6$ = 64 pages (pages 0 to 63)
  - A logical address 0xA934 = 1010 1001 0011 0100 (A=10 decimal)
    - p = 101010 ⇒ First address on page = 1010 1000 0000 0000 = 0xA800
    - d = 0000 0001 0011 0100 = 0x0134
    - Address = First address on page + d = 0xA800 + 0x0134 = 0A934

# Address Translation in Paging- paging hardware



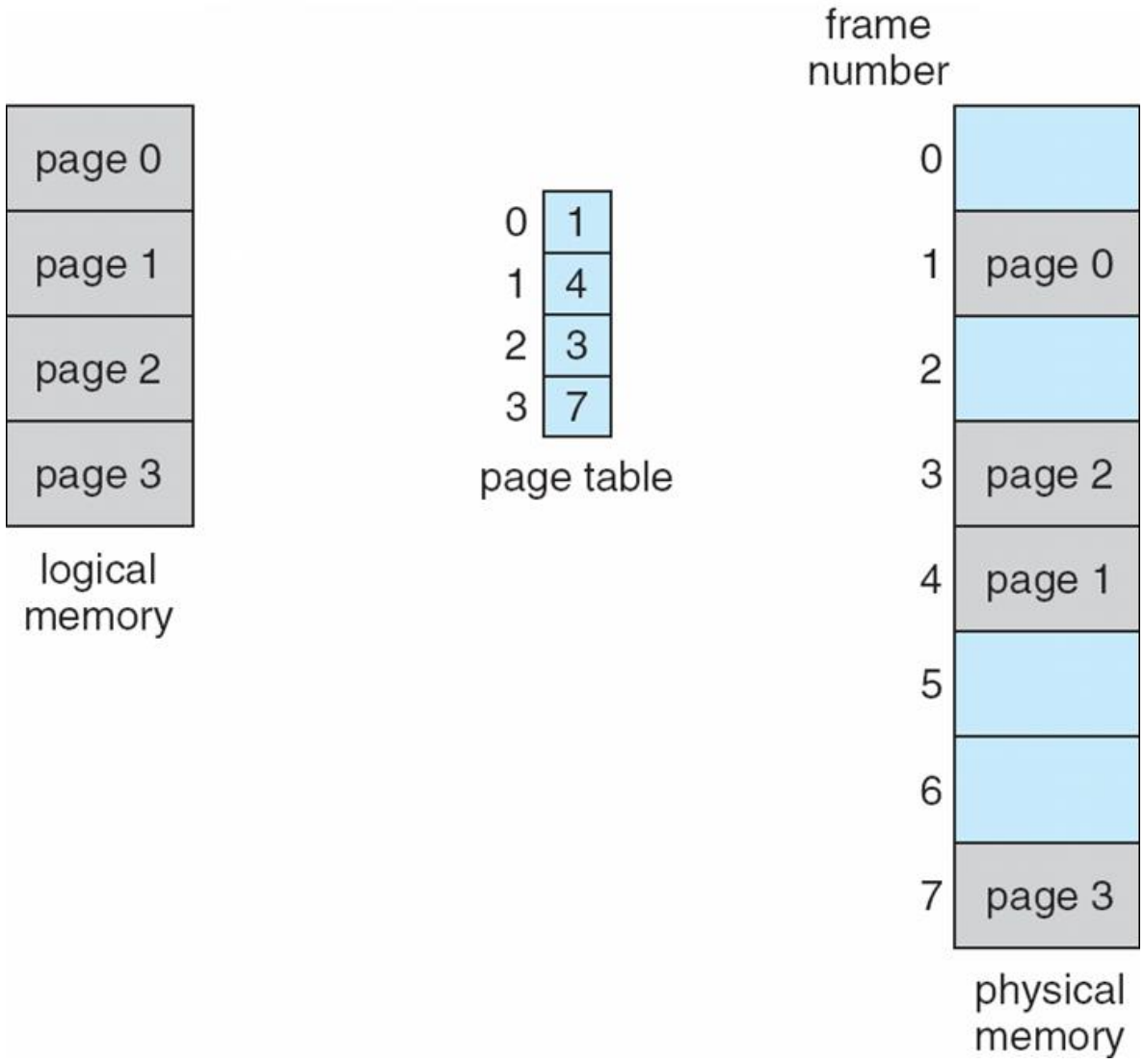

For each process, maintain 1 page table as each process will have page 0, page 1, and so on

# Address Translation in Paging

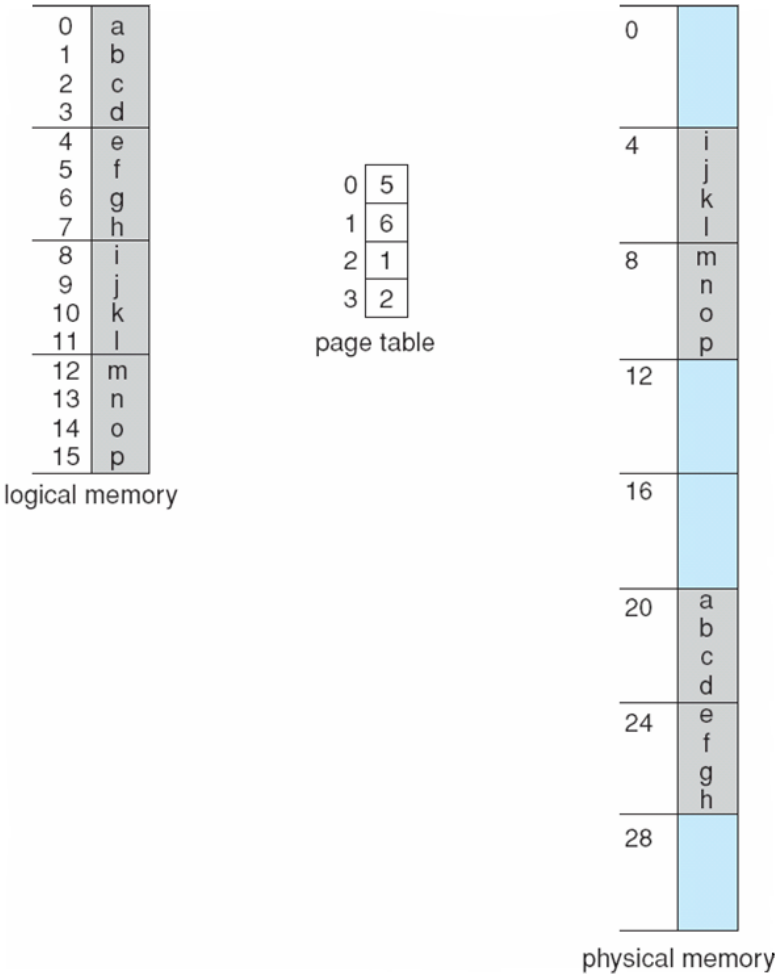
- Page Table
  - Data structure associated to each process
  - Page number ➜ Physical base address for frame
  - Filled at process creation
  - Offset of logical address points to physical address in frame starting from base address

- Translation Steps
  1. Extract p and d
  2. Get p's frame base address (f) from table
  3. Append/add d to f to obtain address in memory

# Paging Model of Logical and Physical Memory

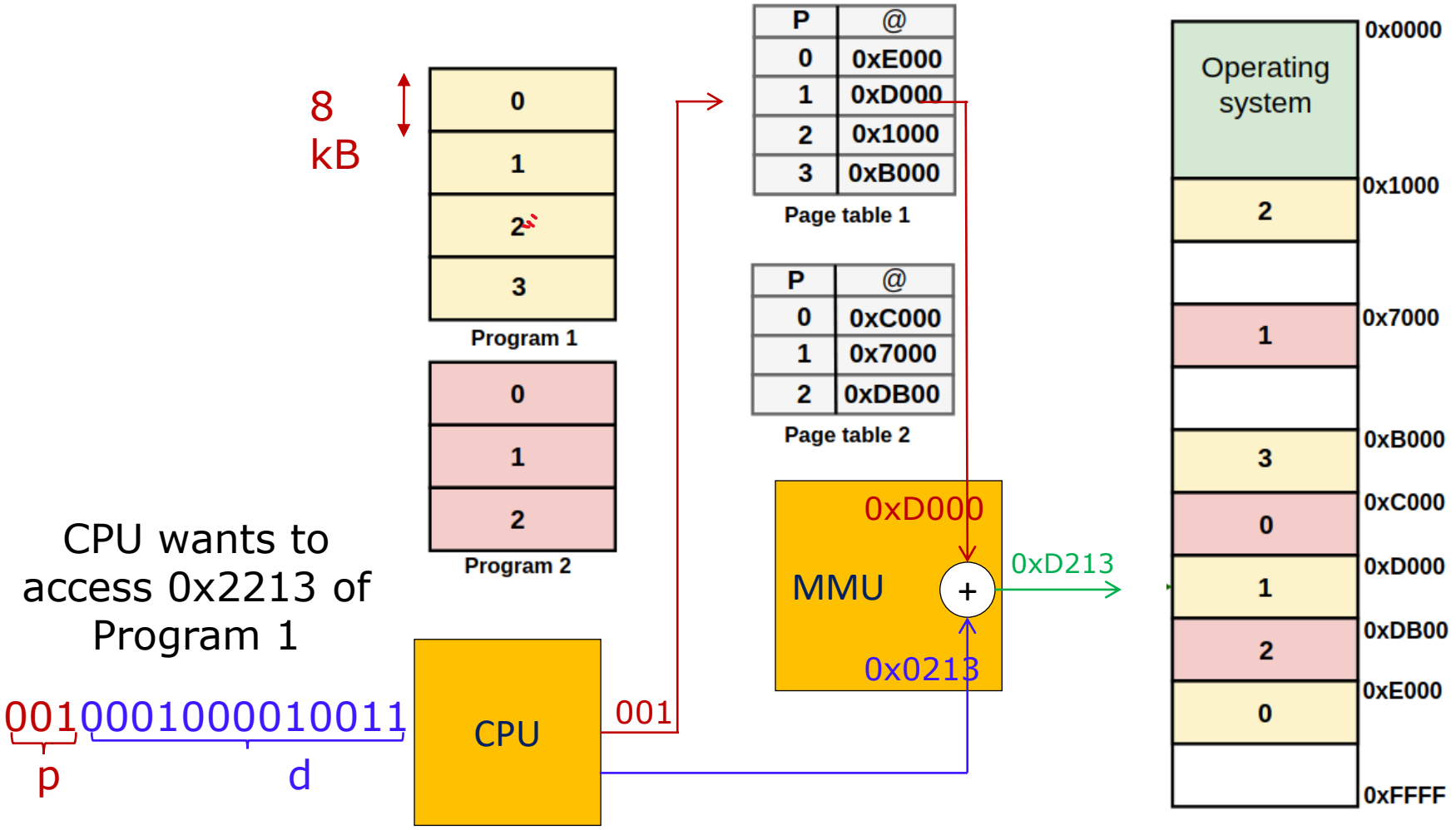


# Paging Example




*n*=2 and *m*=4  32-byte memory and 4-byte pages

# Example


8
kB
0
1
2
3
Program 1
0
1
2
Program 2
P
@
0
0xE000
1
0xD000
2
0x1000
3
0xB000
Page table 1
P
@
0
0xC000
1
0x7000
2
0xDB00
Page table 2
CPU wants to access 0x2213 of Program 1
0010001000010011
p
d
CPU
001
0xD000
MMU
+
0x0213
0xD213
Operating system
2
1
3
0
1
2
0
0x0000
0x1000
0x7000
0xB000
0xC000
0xD000
0xDB00
0xE000
0xFFFF

# Internal Fragmentation

- May still occur with paging
  - Last frame not filled with information in last page of program
- Example:
  - Page/frame size = 2,048 bytes
  - Process size = 72,766 bytes
  - 72766/2048 = 35 pages + 1,086 bytes
  - Internal fragmentation of 2,048 - 1,086 = 962 bytes
  - Allocation: 36 frames - one having 962 unused bytes
- Statistics
  - Worst case: process that is split into n pages plus 1 byte would be allocated n + 1 frames
  - Average case: ½ page/frame size
- Solution: Smaller page/frame sizes
  - Larger overhead in keeping and searching larger page tables

# Implementation of Page Table

- Via Dedicated High-speed Registers
  - Each register storing one entry in the page table
  - Pro: Very fast in address translation
  - Con: Very slow context switching (content of each register must be exchanged by the page table entry of the next process..)
  - Fact: Satisfactory for small page tables up to $2^8$ entries

- Via Memory (case of most contemporary computer )
  - Suitable for larger (most common) tables (e.g., $2^{20}$ entries)
  - Pointer to page table base address in memory stored in PCB
  - Page-table base register (PTBR)
    - Pointer loaded from PCB into PTBR at context switching
    - Pro: Much faster context switching time (changing processes require only change in this register)
    - Con: Slower memory access time – Two memory access to reach physical address: one to **locate the page table** and the other to access the identified **physical address**

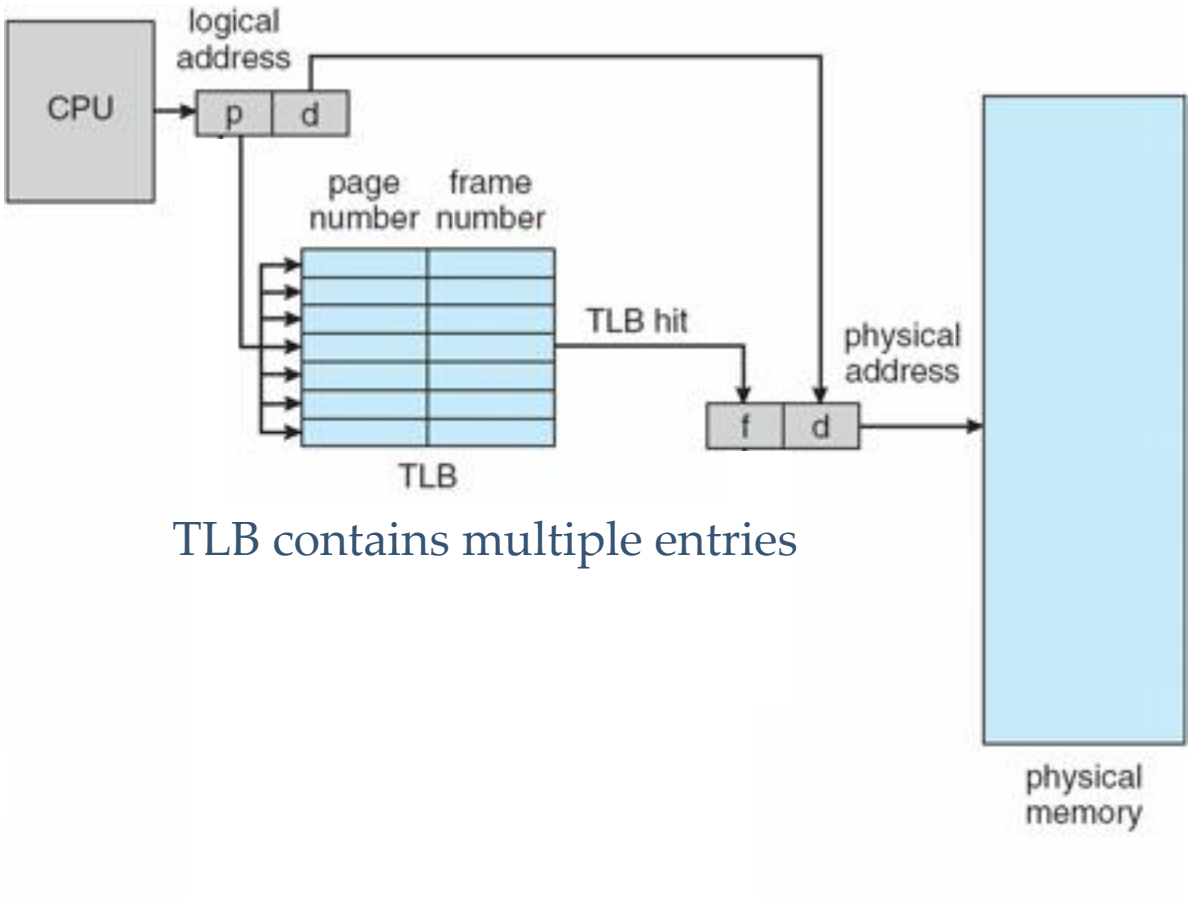
# Translation Look-Aside Buffer (TLB)

- Solution to 2-step memory access problem ➔ special fast-lookup hardware cache called the translation look-aside buffer (or TLB)
  - Stores the recently used virtual-to-physical address translations to speed up the memory access process

- TLB Used for Page Search:
  - TLB stores few page-table entries between 32 and 1,024
  - Sometimes two TLBs used, one for instructions and one for data
  - MMU checks if page number is in TLB:
    - If so: Hit - Base address extracted, used to access memory
    - If not: Miss - 2-step memory access: get the address from the page table

- Replacement applied on TLB:
  - Add recently called address in TLB instead of an existing one
  - Replacement policies: LRU (least recently used), RR, Random

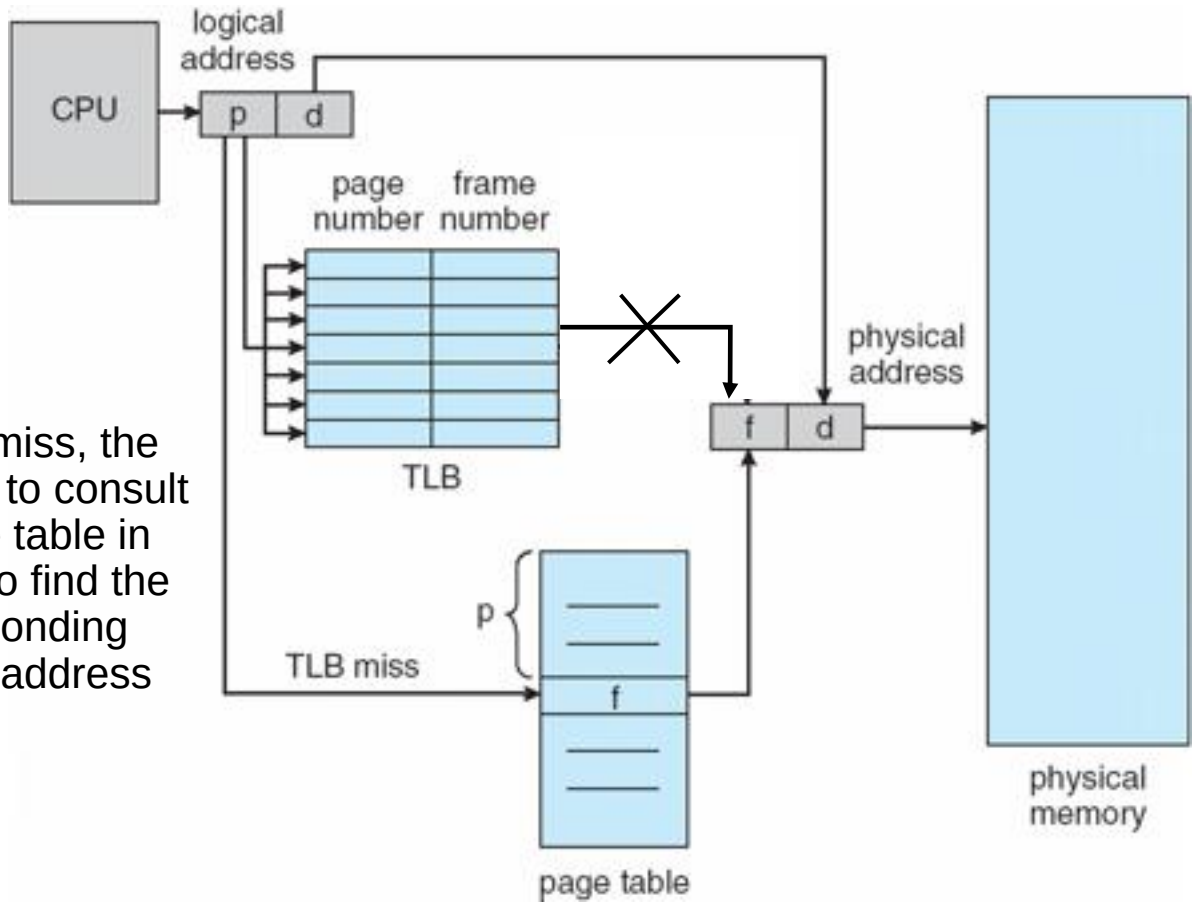
# TLB-Hit




TLB contains multiple entries

# TLB-Miss

On TBL-miss, the CPU need to consult the page table in memory to find the corresponding physical address



# Effective Access Time (EAT)

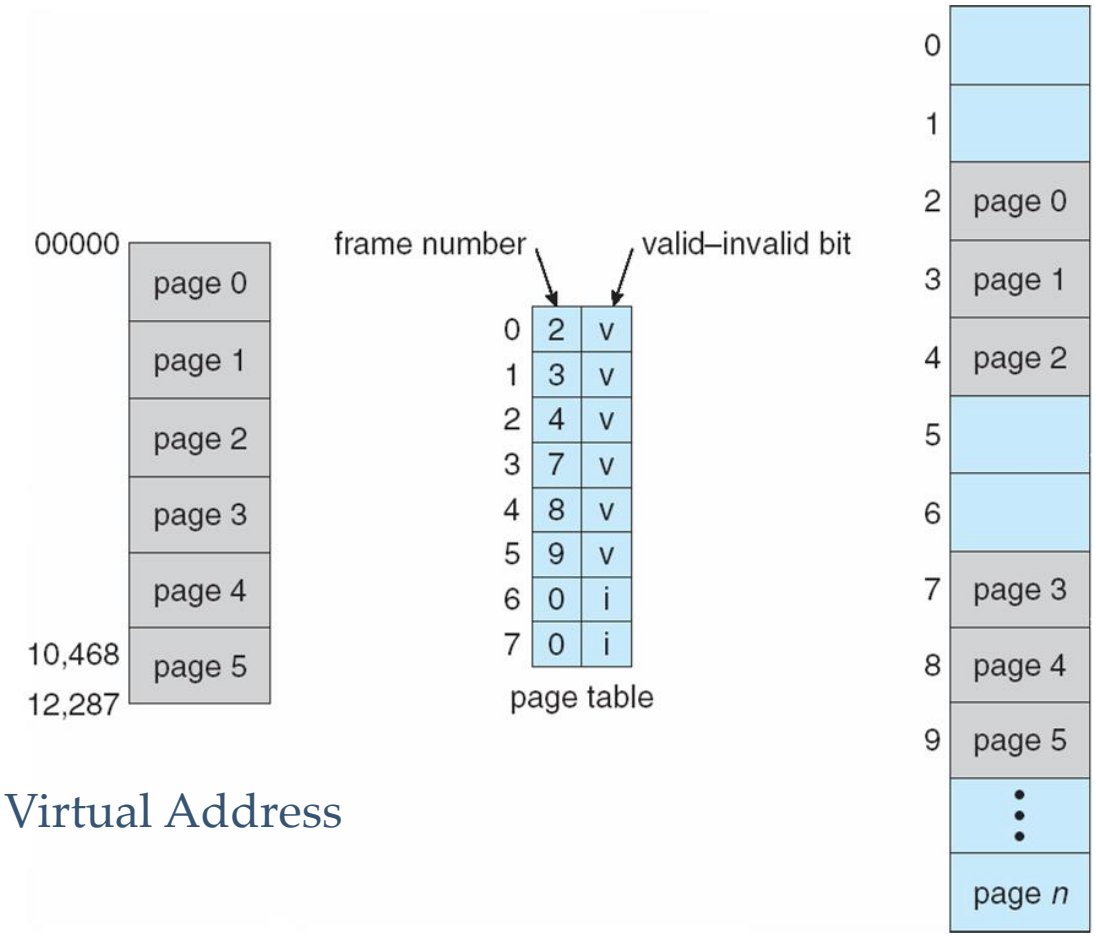
- Access time differs between TLB hit and miss cases
  - EAT: Defines the overall performance of the system
- Calculation
  - Hit Ratio: Proportion of time a page is found in TLB
  - Miss Ratio: Proportion of time a page is not found (1 – Hit Ratio)
  - Access Time: Time needed to access the memory once

  **EAT = Hit Ratio x Access Time + Miss Ratio x 2 x Access Time**
  **(2 is for 2-step memory access)**
- Example
  - Hit Ratio = 0.8
  - Access Time: 2 ns
  - EAT = 0.8 x 2 + 0.2 x 2 x 2 = 2.4 ns

# 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
  - Or use page-table length register (PTLR)
- Any violations result in a trap to the kernel

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



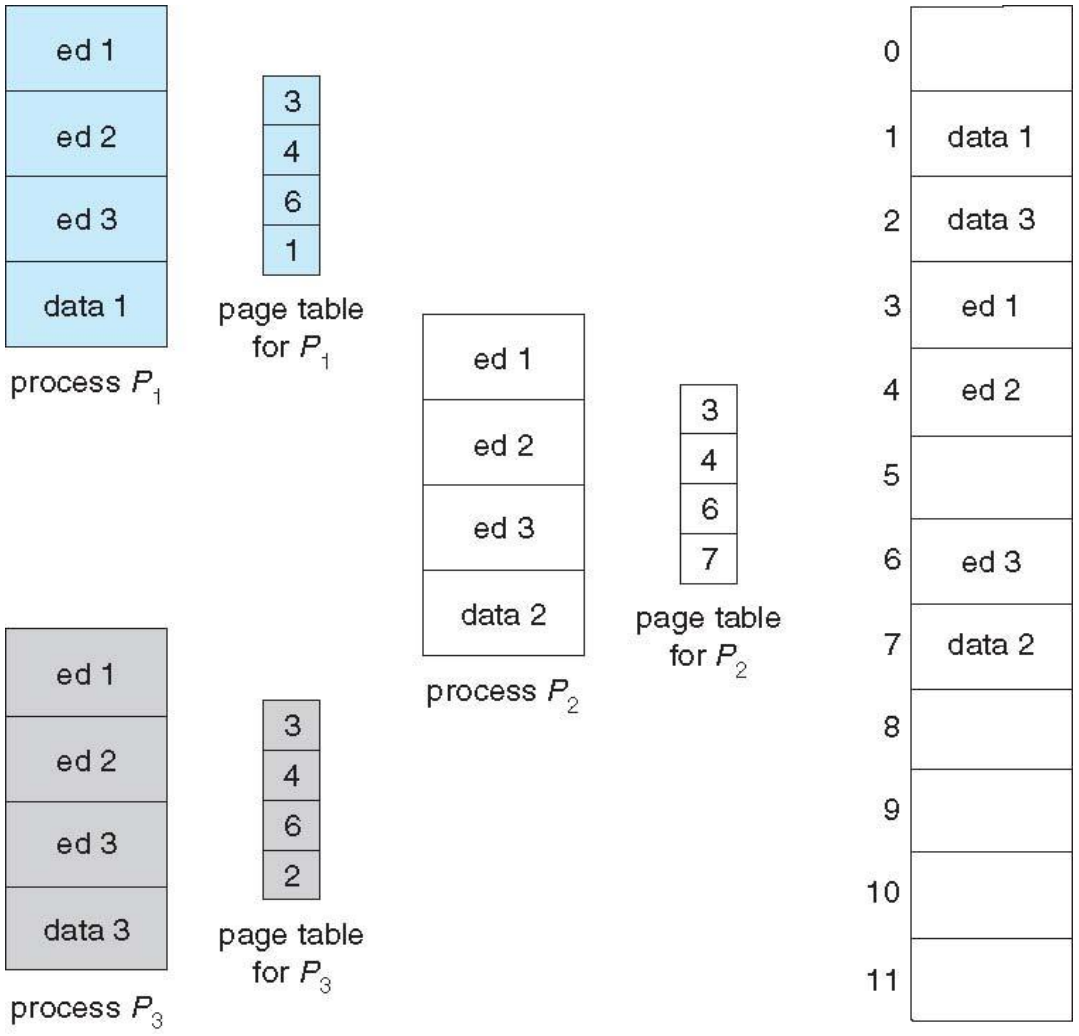

Virtual Address

Physical Address

# 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 Example



# Page Table Structures

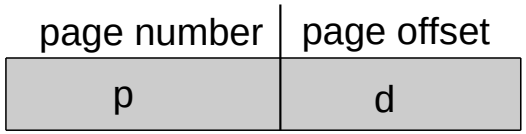
- Modern computers support large logical address spaces
  - 32-bit systems: $2^{32}$ addresses
  - 64 –bit systems: $2^{64}$ addresses.
- Excessively large page tables
  - Example: For 32-bit system
    - 4 kB page size ($2^{12}$) results in $2^{32}/2^{12} = 2^{20} \approx$ 1M table entries
    - If each entry is only 4 bytes, page table size = 4 MB
    - That amount of memory used to cost a lot
- Smarter Structures
  - Hierarchical Page Tables
  - Hashed Page Tables
  - Inverted Page Tables

# Hierarchical Page Table

- 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

# Hierarchical Paging

- Dividing table into smaller pages (paging in page table)
  - Remember: 1-level paging




m-bit logical address — m-n — n

32-bit logical address — 20 — 12

  - Hierarchical (2-level) paging

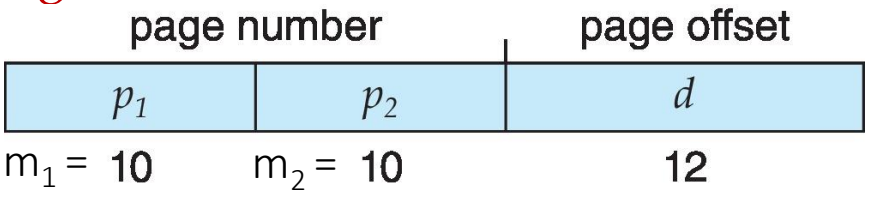
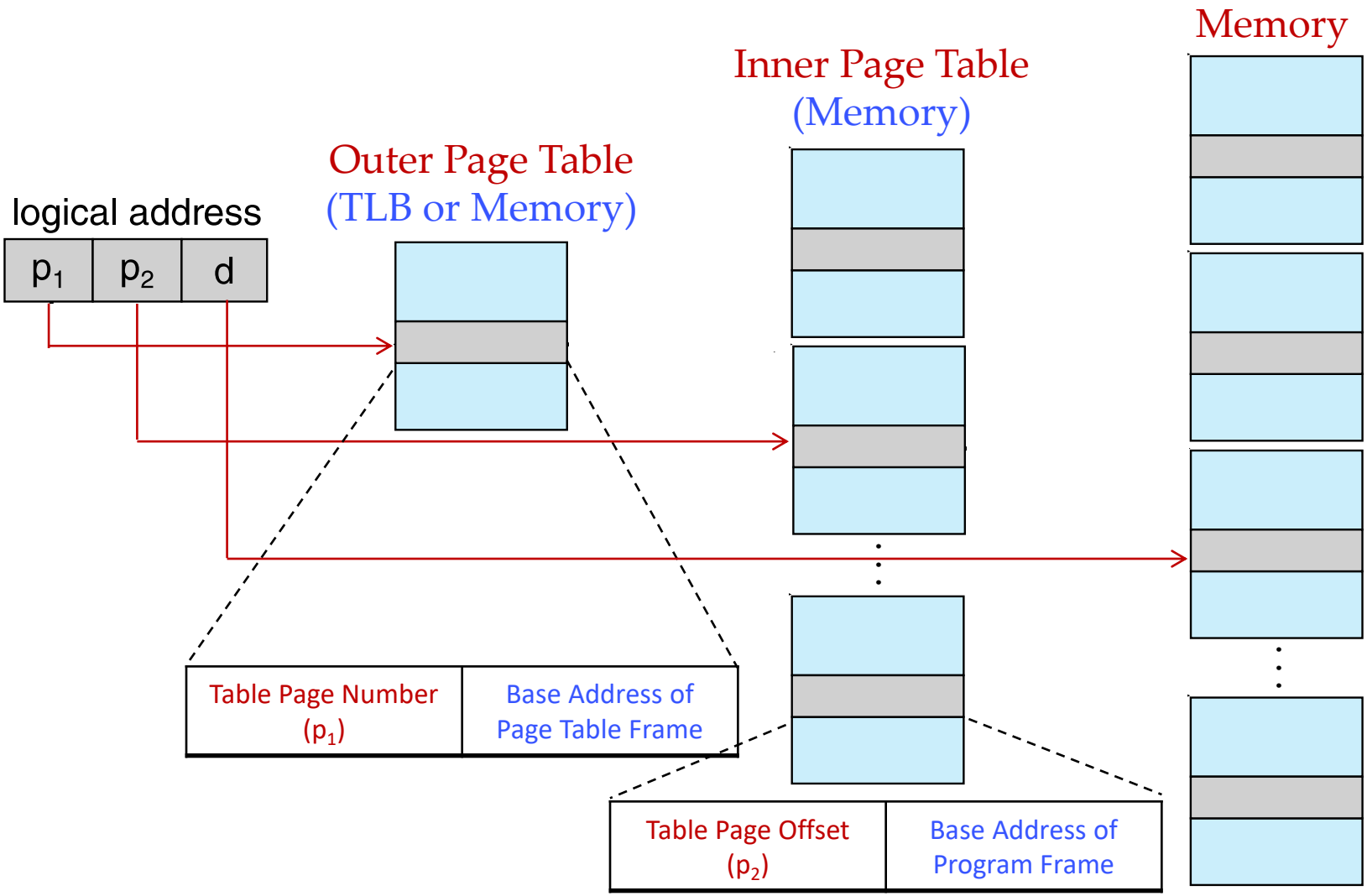


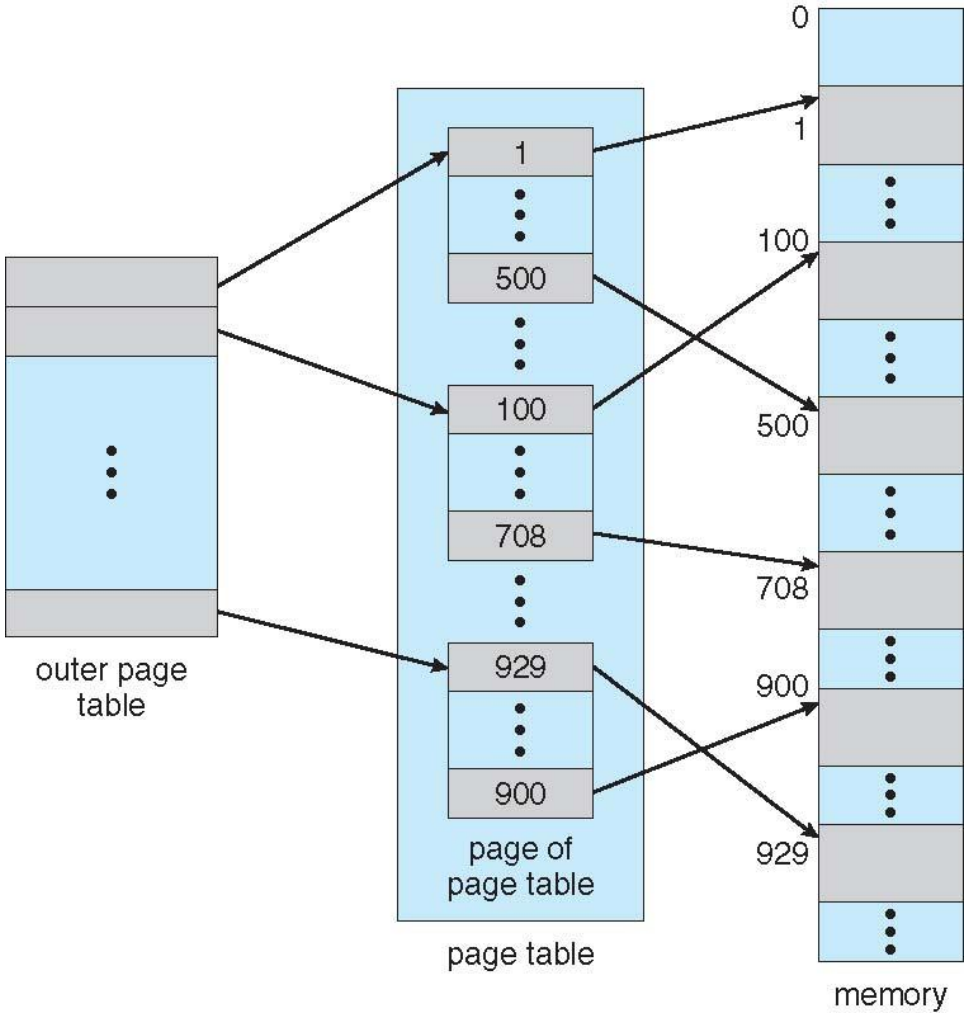

Table page number — Table page offset

  - Two types of frames
    - Page table frames: Frames in the memory storing pages of the page table
    - Program frames: Frames in the memory storing pages of program

# Address Translation



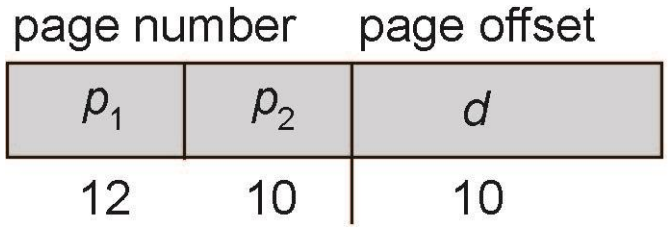
# Two-Level Page-Table Scheme



# Two-Level Paging Example

- 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

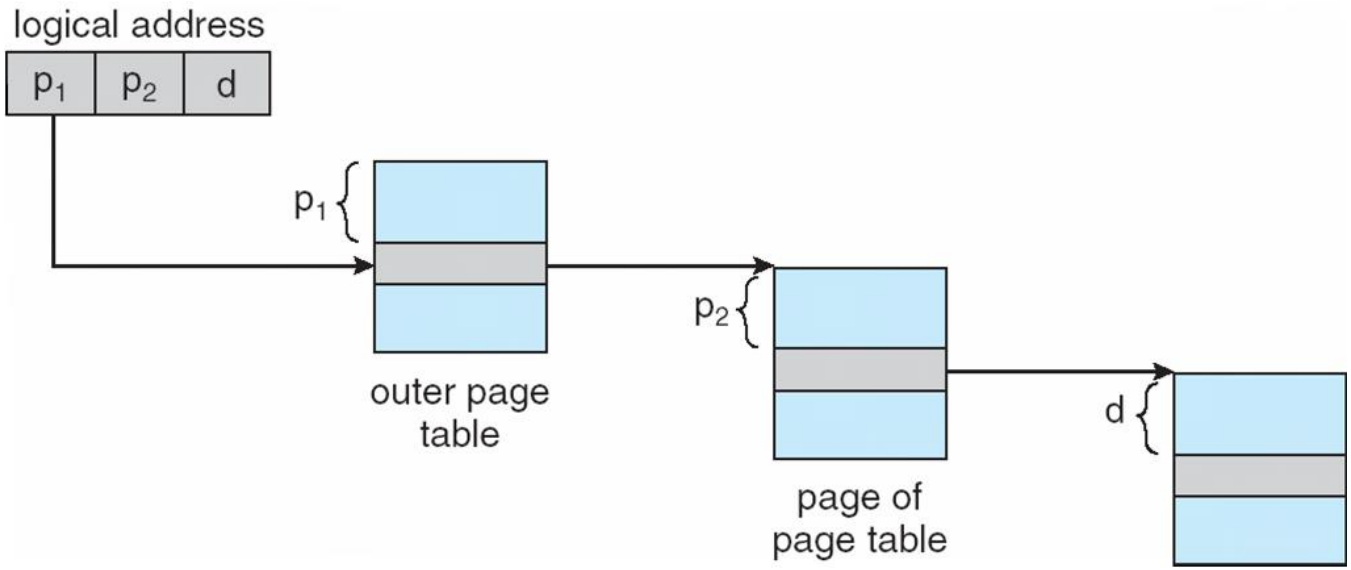
- Since the page table is paged, the page number is further divided into:
  - a 12-bit page number
  - a 10-bit page offset
- Thus, a logical address is as follows:




where p1 is an index into the outer page table, and p2 is the displacement within the page of the inner page table

- Known as forward-mapped page table

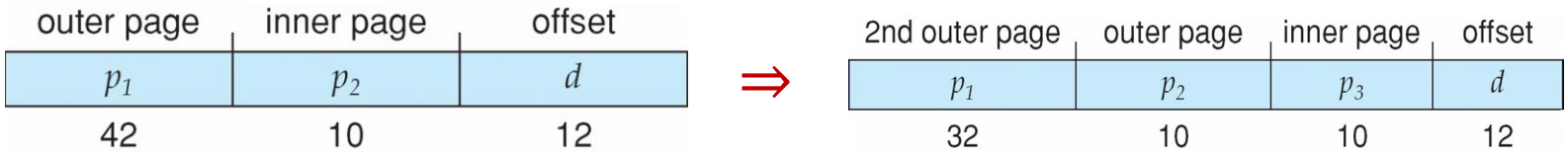
# Address-Translation Scheme



# Benefits

- Resolve huge page table problems
  - Storage: Table partitioned into $2^{m1}$ pages stored in non-contiguous frames
  - Search: Drops down from $O(2^{(m1 + m2)})$ in the conventional 1-level paging to $O(2^{m1} + 2^{m2})$ in the 2-level hierarchical paging
    - When $m_1 = m_2 = 10$, time reduced from $O(2^{20})$ to $O(2^{11})$ ⇒ 512 times less

- Extension
  - More levels in the hierarchy for larger addresses
  - Example: For 64-bit system:

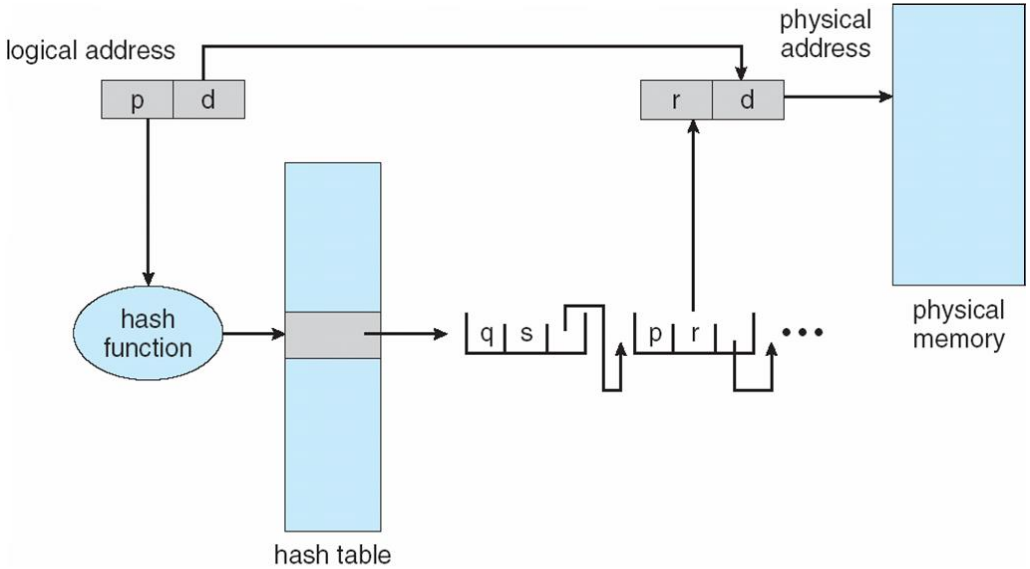



2-level paging

3-level paging

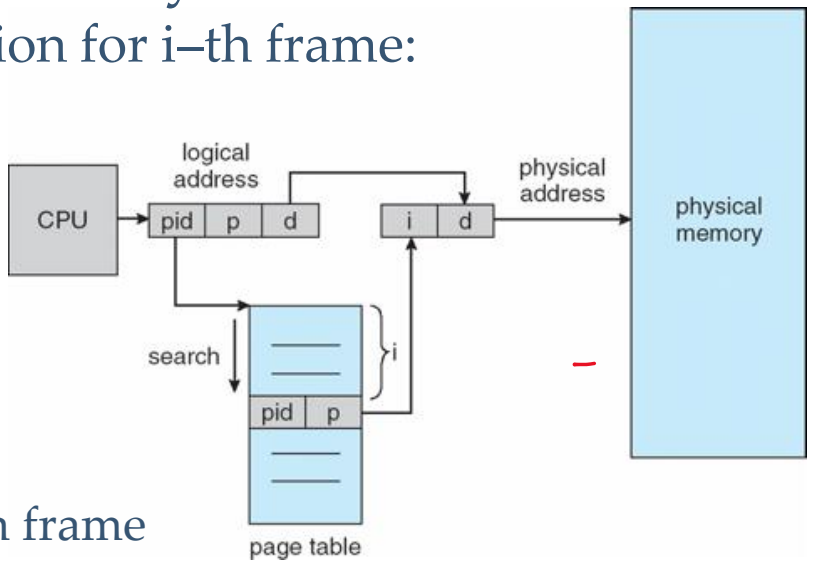
# Hashed Paging

- Hash function: Page number ➜ Entry in hash page table
  - Common in address spaces > 32 bits
  - The virtual page number is hashed into a page table
  - Entry: A linked list of elements
    - Include information for all page numbers hashable into this entry
  - Each element includes:
    - One Page number
    - Base address of corresponding frame in the memory
    - Pointer to next element in linked list



# Inverted Paging

- Inverted Page Table: One table sorted by order of frames
  - i-th entry in tables includes information for i–th frame:
    - Pid of process
    - Page number
  - Address translation
    - p extracted
    - Used with pid to search table
    - Match found in entry i
    - Physical address = d-th address in i-th frame




  - Pro: Decreases memory needed to store each page table
  - Con: Increases time needed to search table
  - Possible Solutions:
    - Hash table to limit search
    - TLB can accelerate access