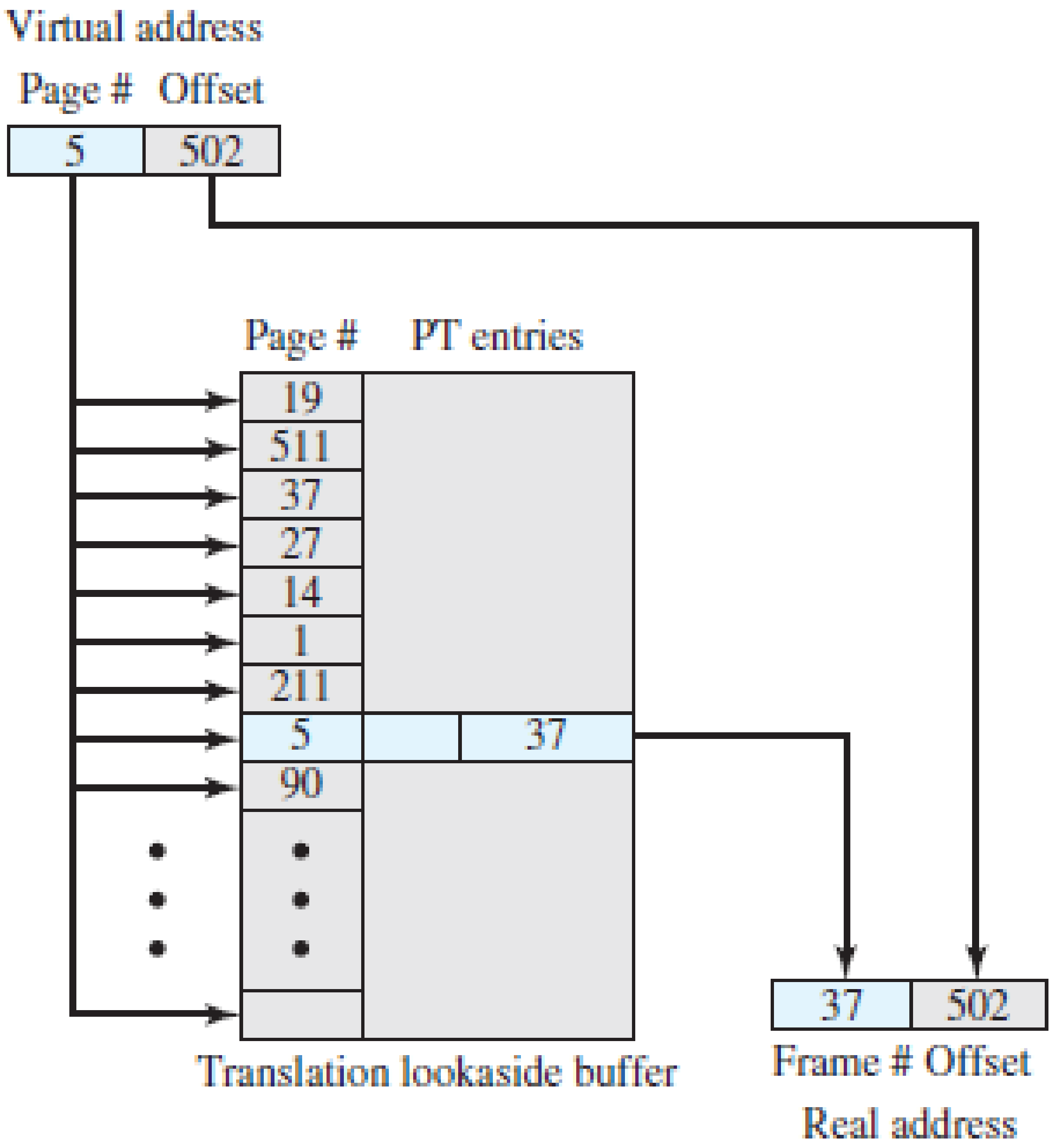
BITS Pilani
K K Birla Goa Campus

# Operating Systems

**Dr. Lucy J. Gudino**
**Dept. of CS and IS**

## Last Class

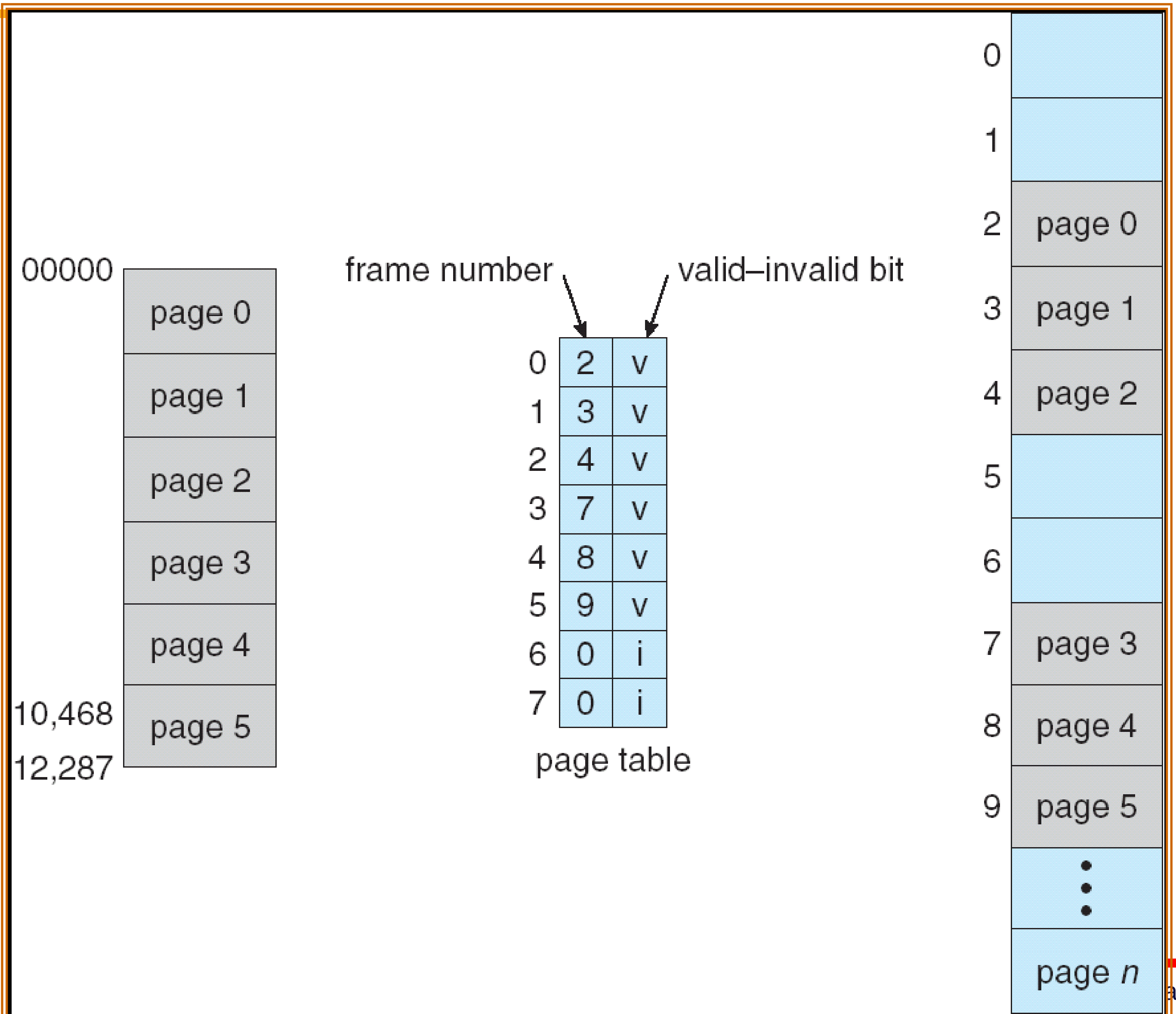
- Associative Cache
  - A main memory block can load into any line of cache
  - Memory address is interpreted as tag and word
  - Tag uniquely identifies block of memory
  - Every line’s tag is examined for a match
  - Cache searching gets expensive when number of lines increses
- Translation Look Ahead Buffer (TLB)
  - Each entry in the TLB consists of two parts: a key (or tag) and a value
  - Contains those page table entries that have been most recently used (Principle of locality of reference)
  - Entries can be wired down, meaning that they cannot be removed from the TLB
  - Address-Space IDentifiers (ASIDs) in each TLB entry
    - To uniquely identify each process
    - Provide address space protection


Virtual address
Page # Offset
5
502
Page # PT entries
19
511
37
27
14
1
211
5
37
90
Translation lookaside buffer
37
502
Frame # Offset
Real address

## Memory Protection

- Memory protection implemented by associating protection bits with each frame
  - Page to be read-write, read-only or execute only
    - Trap in case write attempt to read only page
  - **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

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



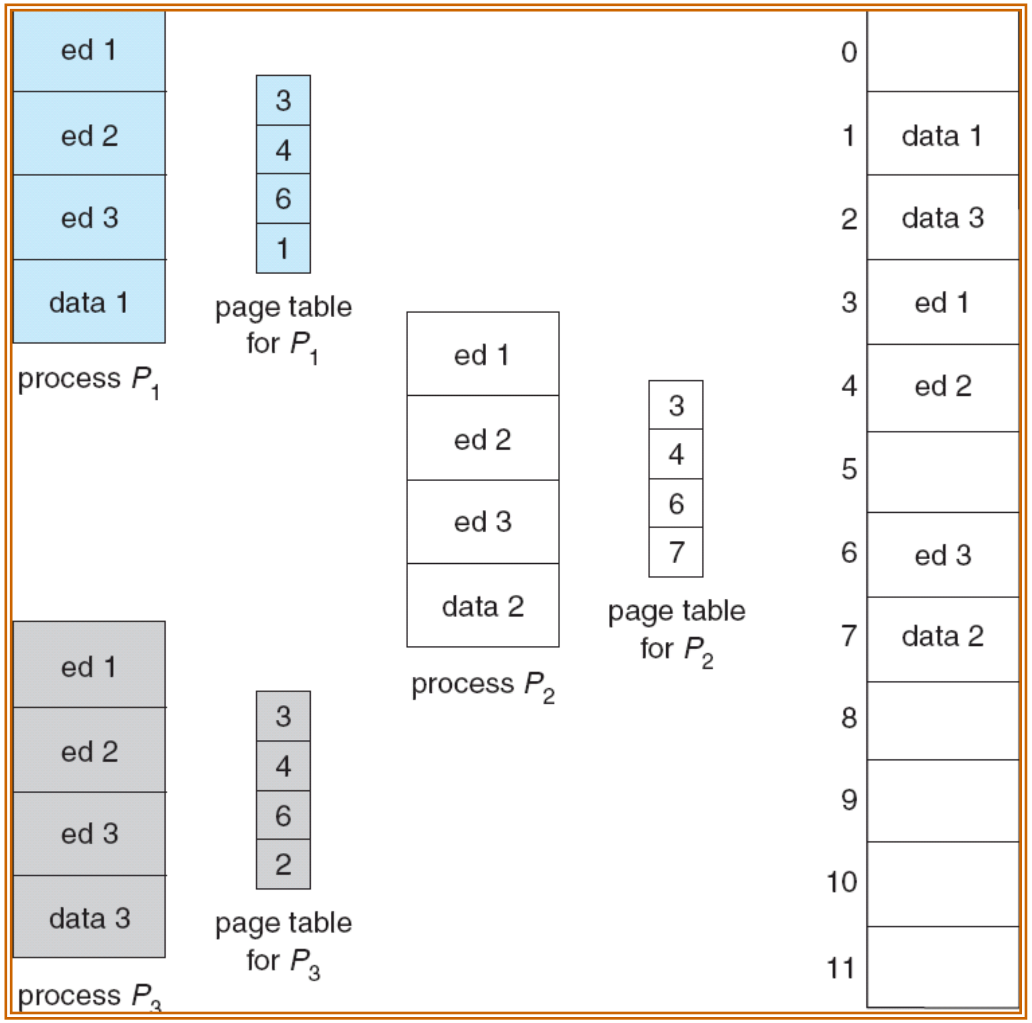
## Contd…

- Problem : Page with internal fragmentation
- Page – Table Length Register (PTLR) ➔ indicate the size of the page table
  - checked against every logical address to verify that the address is in the valid range for the process

## Shared Pages

- **Shared code**
  - One copy of read-only (reentrant) code shared among processes (i.e., text editors, compilers, window systems).
  - Shared code must appear in same location in the logical address space of all processes

# Shared Pages Example



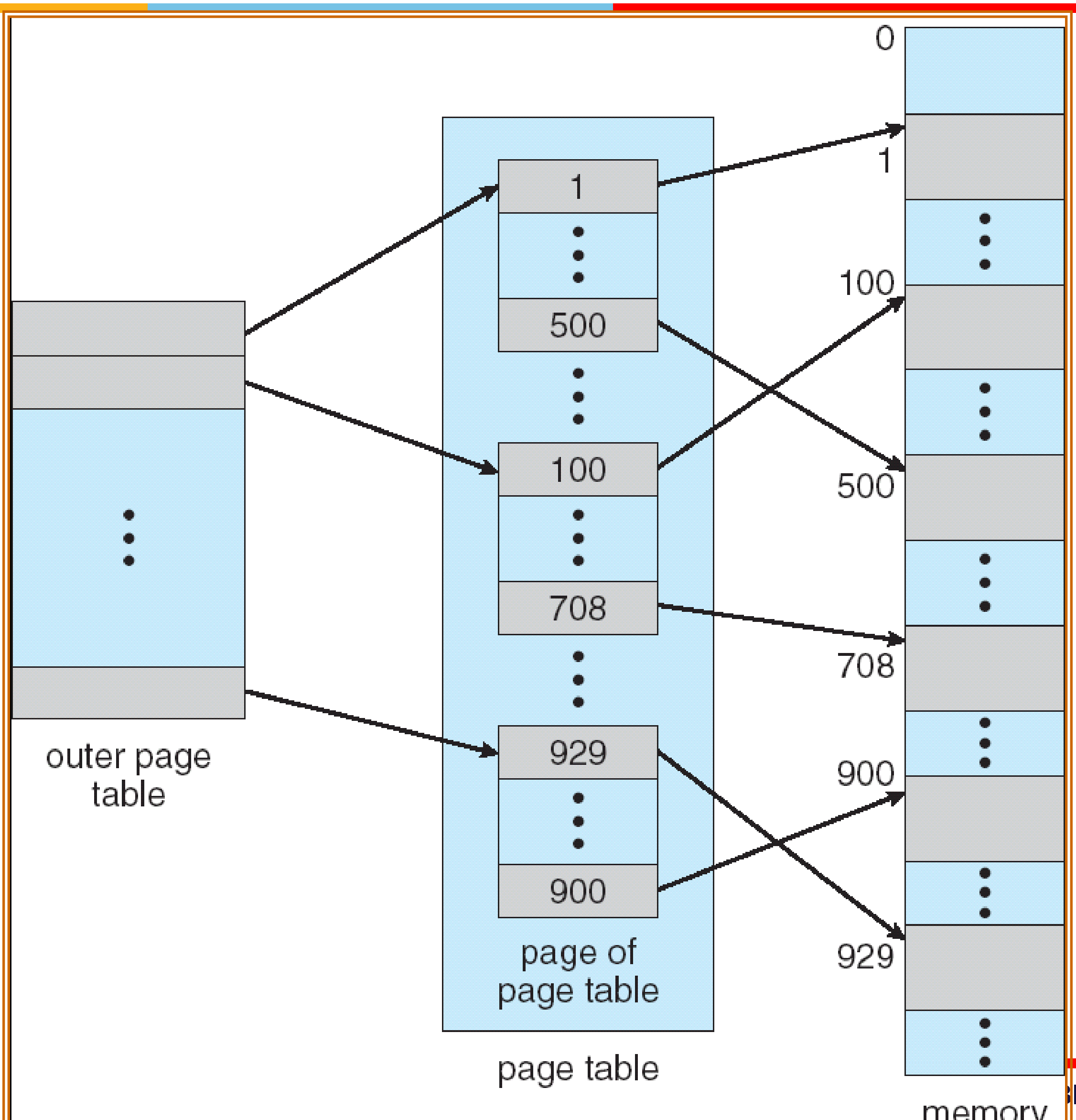
## Structure of the Page Table

- Hierarchical Paging
- Hashed Page Tables
- Inverted Page Tables

# Hierarchical Page Tables

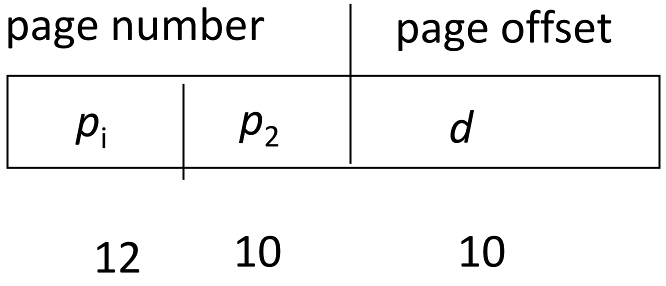
- Example: Logical address space ➔ $2^{32}$
  Page size ➔4 K ➔ $2^{12}$
  Number of entries in the page table ?
  Each entry contains ➔ 4 bytes
  Physical address space needed?
- Break up the logical address space into multiple page tables
- A simple technique is a two-level page table

# Two-Level Page-Table Scheme



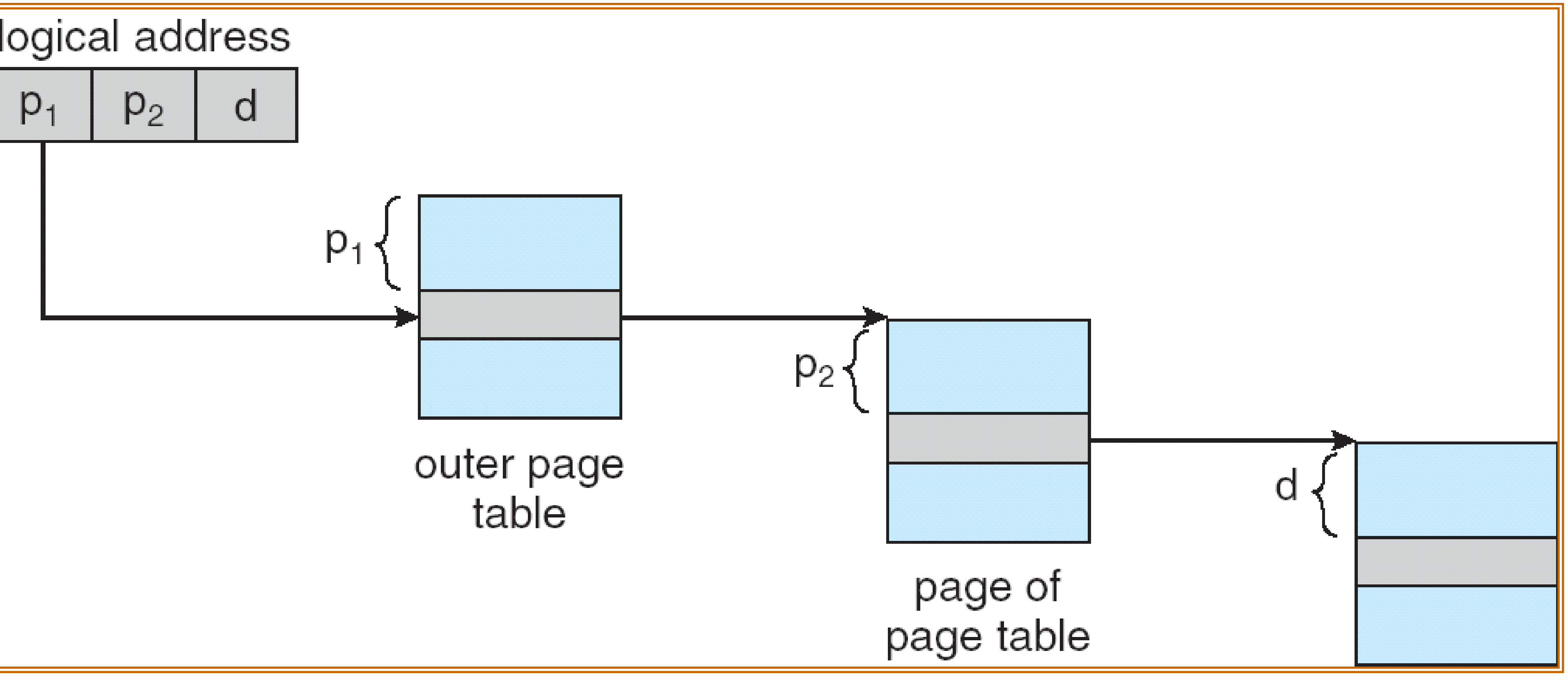
# Two-Level Paging Example

- 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 $p_i$ is an index into the outer page table, and $p_2$ is the displacement within the page of the outer page table

# Address-Translation Scheme



## Three-level Paging Scheme

| outer page | inner page | offset |
|---|---|---|
| $p_1$ | $p_2$ | $d$ |
| 42 | 10 | 12 |

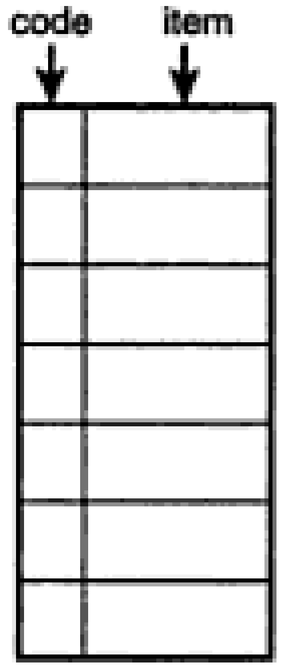
| 2nd outer page | outer page | inner page | offset |
|---|---|---|---|
| $p_1$ | $p_2$ | $p_3$ | $d$ |
| 32 | 10 | 10 | 12 |

## Contd…

- Although this process saves space, it comes at the expense of an additional memory lookup.
- A 2-level page table requires a total of three memory lookups for each logical address.

# Hashed Page Tables

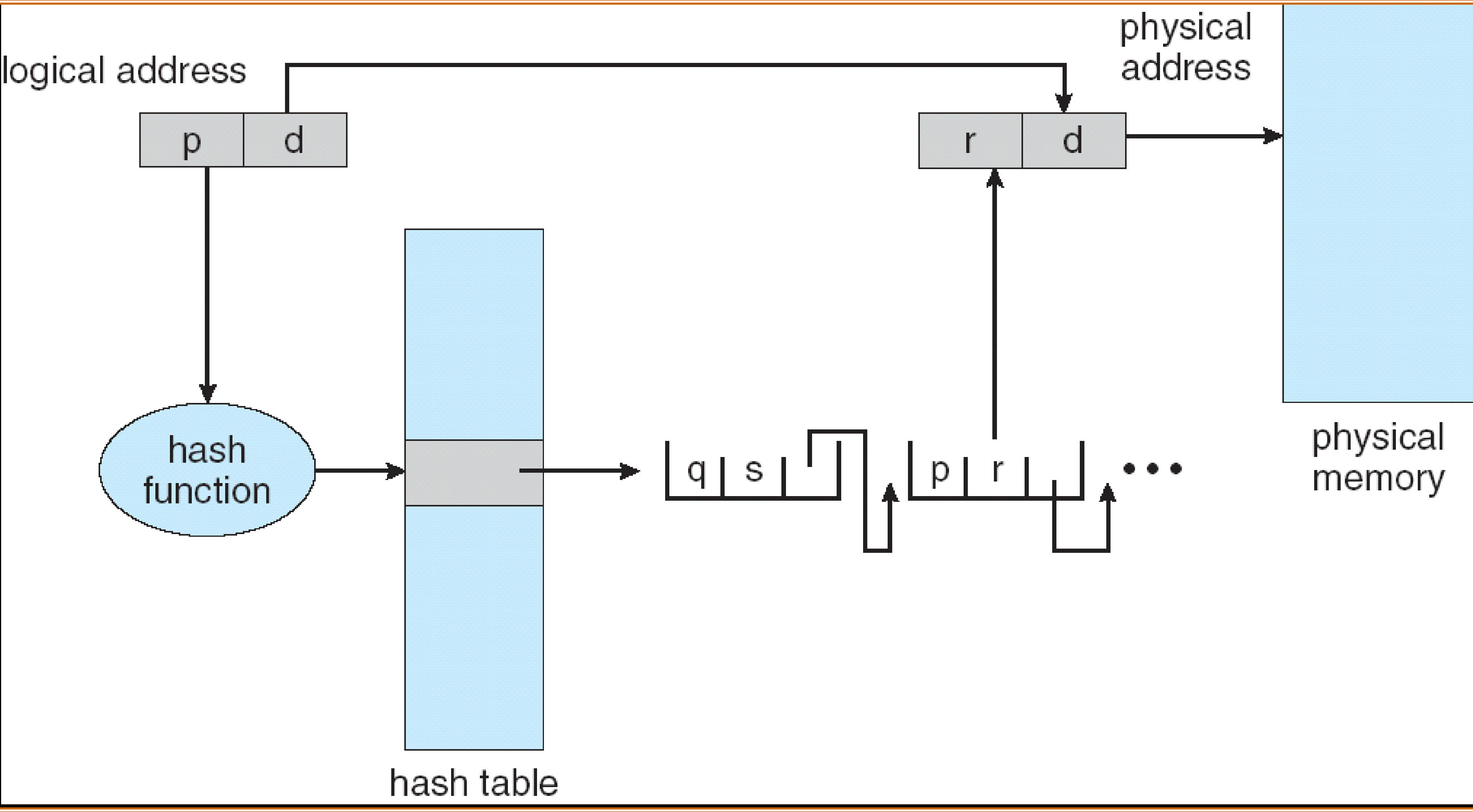
- A **hash table** is a data structure that computes a numerical value for an item, and then references that item by that numerical value in a table.
- Table has two entries: hash code and the item
- Operations: Insertion, Search and Deletion
- Insertion :
  - Compute the hash code for the object
  - insert the object at that index of the hash table
- Search :
  - Compute the hash code
  - use direct look up
- Two or more objects may collide in the hash table



# Hashed Page Tables

- Suitable for address spaces > 32 bits
- The virtual page number is used as hashed value
- Each entry contains the linked list of elements that hash to the same location
- Each element in the hash table contains following fields
  - Virtual page number
  - Mapped page frame value
  - Pointer to the next element in the linked list

# Hashed Page Table



## Procedure

1. Virtual page number is taken from virtual address
2. Virtual Page number is hashed in to the hash table
3. Virtual page number is compared with the first element of the linked list
4. If matched, frame value is used for calculating physical address
5. If not matched, next element is considered for matching and so on

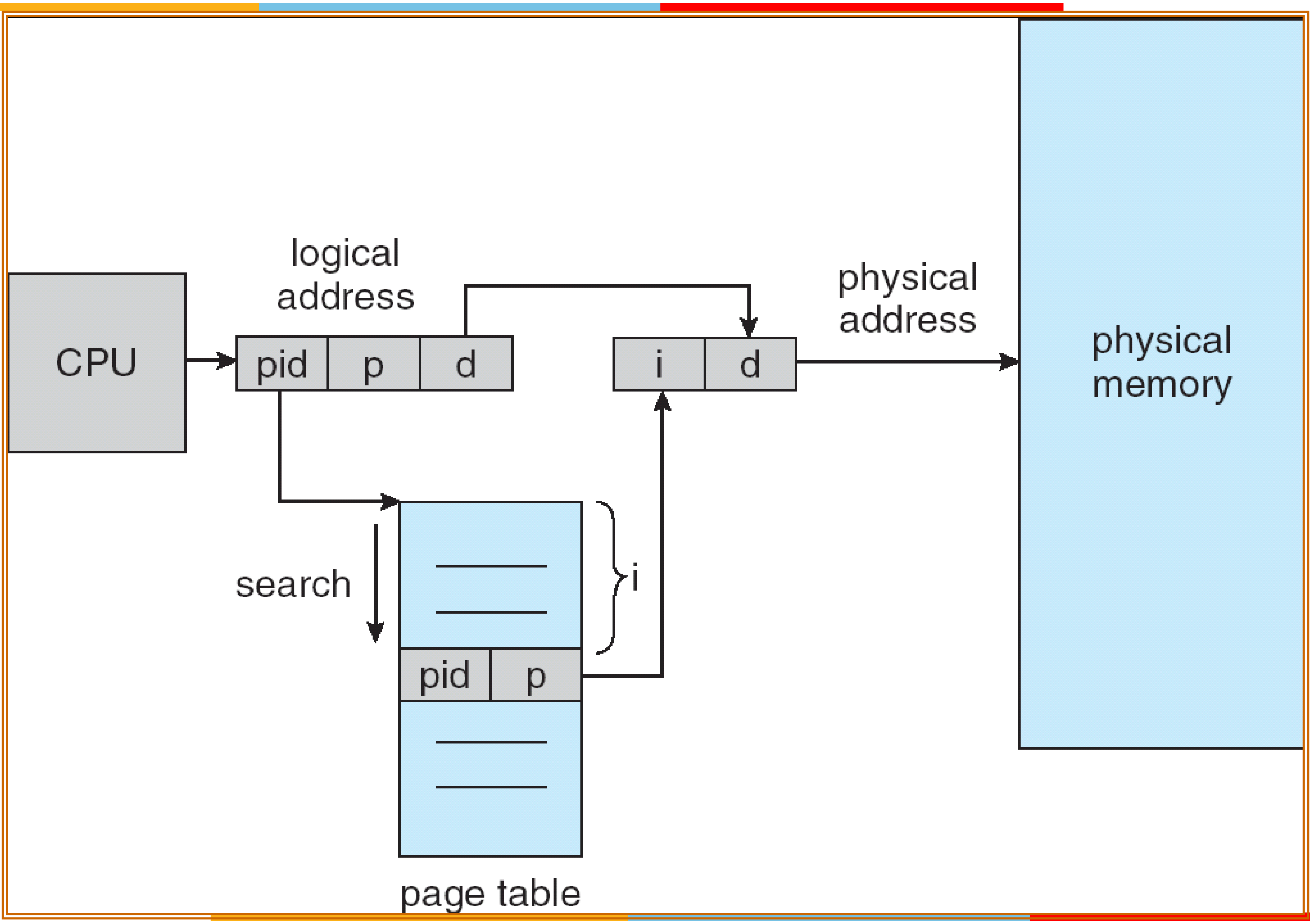
# Reading Assignment

1. Clustered Page Tables
2. Inverted Page Tables

# Inverted Page Table

- 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
- Use hash table to limit the search to one — or at most a few — page-table entries

# Inverted Page Table Architecture


logical address
CPU
pid
p
d
i
d
physical address
physical memory
search
pid
p
i
page table