CS343: Operating System

Memory Management: Segmentation and Paging

Lect31 : 27th Oct 2023

Dr. A. Sahu

Dept of Comp. Sc. & Engg.

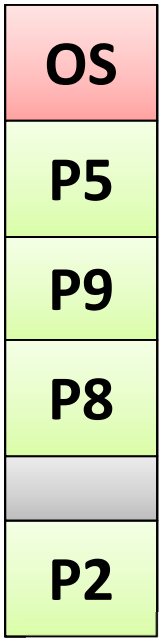
Indian Institute of Technology Guwahati

Outline

- Memory Management
  - Continuous Memory allocation
  - Buddy System
  - Segmentation
  - Paging

Memory Allocation: Top Down

Assume 4GB of RAM,
Avg Process Size 200MB
4 Segments/Process



Finding Hole
First Fit
Worst fit
Best Fit

Very low complexity

Very High Mem Wastage

Process Continuous Allocation
10 process and 20 holes

Segment continuous Allocation
40 segment and 80 holes

low complexity

High Memory Wastage

Paging: 200MB process, 4KB page
Two Issue: pages and frames

Very Low Mem Wastage

High Complexity: 51200 pages/Process

Loading 51K pages in 10^6 frames

High Complexity: 4GB/4KB=2^{20}=10^6 frames

SOL

Demand Paging Load the required 50-100 pages and load as an when necessary

4GB paged

SOL

Allocate 100-200 frames per process: reduce search Space

Fragmentation

- **External Fragmentation** – Total memory space exists to satisfy a request, but it is not contiguous
- **Internal Fragmentation** – allocated memory may be slightly larger than requested memory
  - This size difference is memory internal to a partition, but not being used

Fragmentation: First Fit

- Analysis reveals that given *N* blocks allocated, 0.5 *N* blocks lost to fragmentation
  - 1/3 may be unusable
  - **50-percent rule**

Fragmentation (Cont.)

- Reduce external fragmentation by **compaction**
  - Shuffle memory contents to place all free memory together in one large block
  - Compaction is possible *only* if relocation is dynamic, and is done at execution time
  - I/O problem
    - Latch job in memory while it is involved in I/O
    - Do I/O only into OS buffers
- Now consider that backing store has same fragmentation problems

Segmentation

Segmentation

- Memory-management scheme that supports user view of memory
- A program is a collection of segments
  - A segment is a logical unit such as:

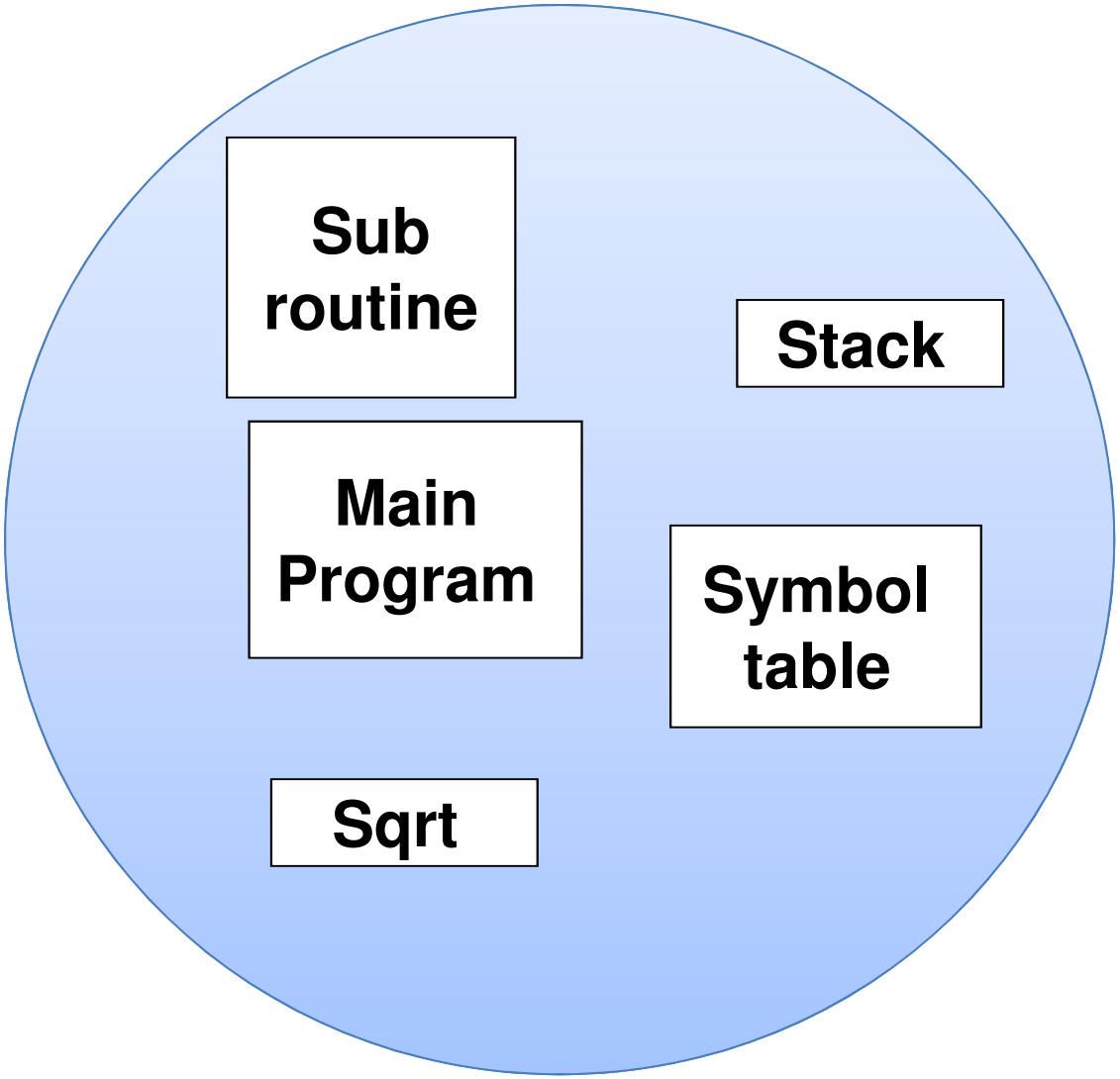
    main program,
    procedure/function/method,

    object, local variables, global variables

    common block, stack, symbol table

    arrays

User's View of a Program

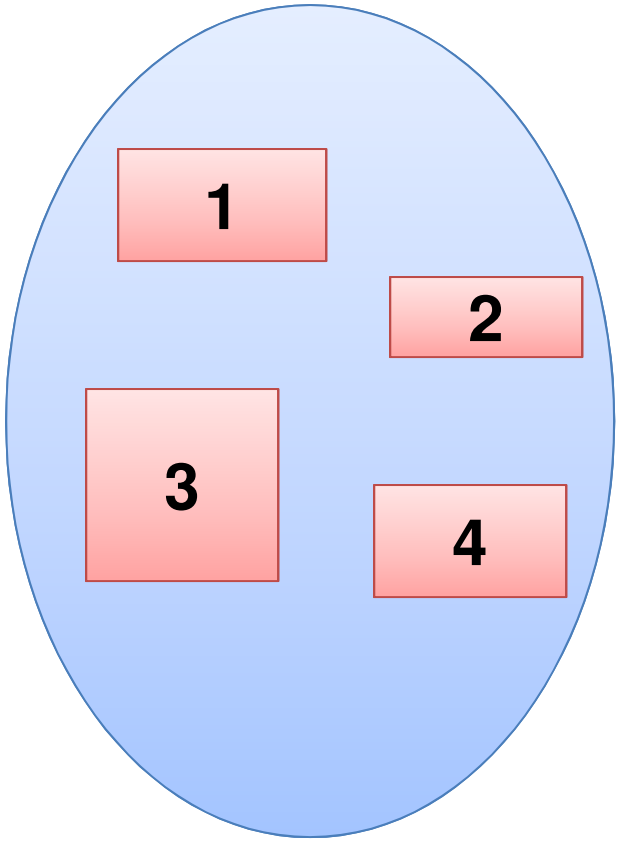


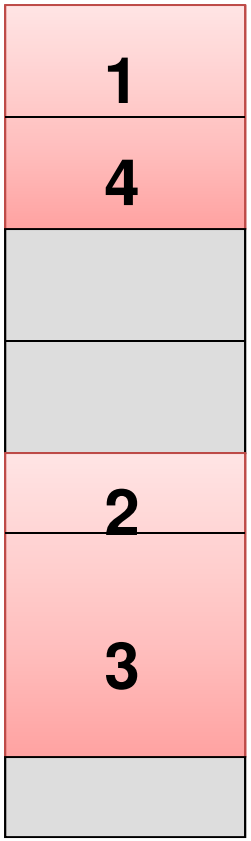
Logical address

Segmentation : Key Rule

- Instead of allocation memory for whole process
- Divide the program into smaller block call segments
  - User view : Already segmented
  - Finer granularity, less fragmentation
    - **Mustard in Bag Vs Brinjal in Bag**
- Each of which is allocated to memory independently
- Segments are variable size

Logical View of Segmentation





user space

physical memory space

Segmentation Architecture

- Logical address consists of a two tuple:

  <segment-number, offset>,

- **Segment table** – maps two-dimensional physical addresses; each table entry has:
  - **base** – contains the starting physical address where the segments reside in memory
  - **limit** – specifies the length of the segment

Segmentation Architecture

- **Segment-table base register (STBR)** points to the segment table's location in memory
- **Segment-table length register (STLR)** indicates number of segments used by a program;

  segment number ***s*** is legal if ***s*** < **STLR**

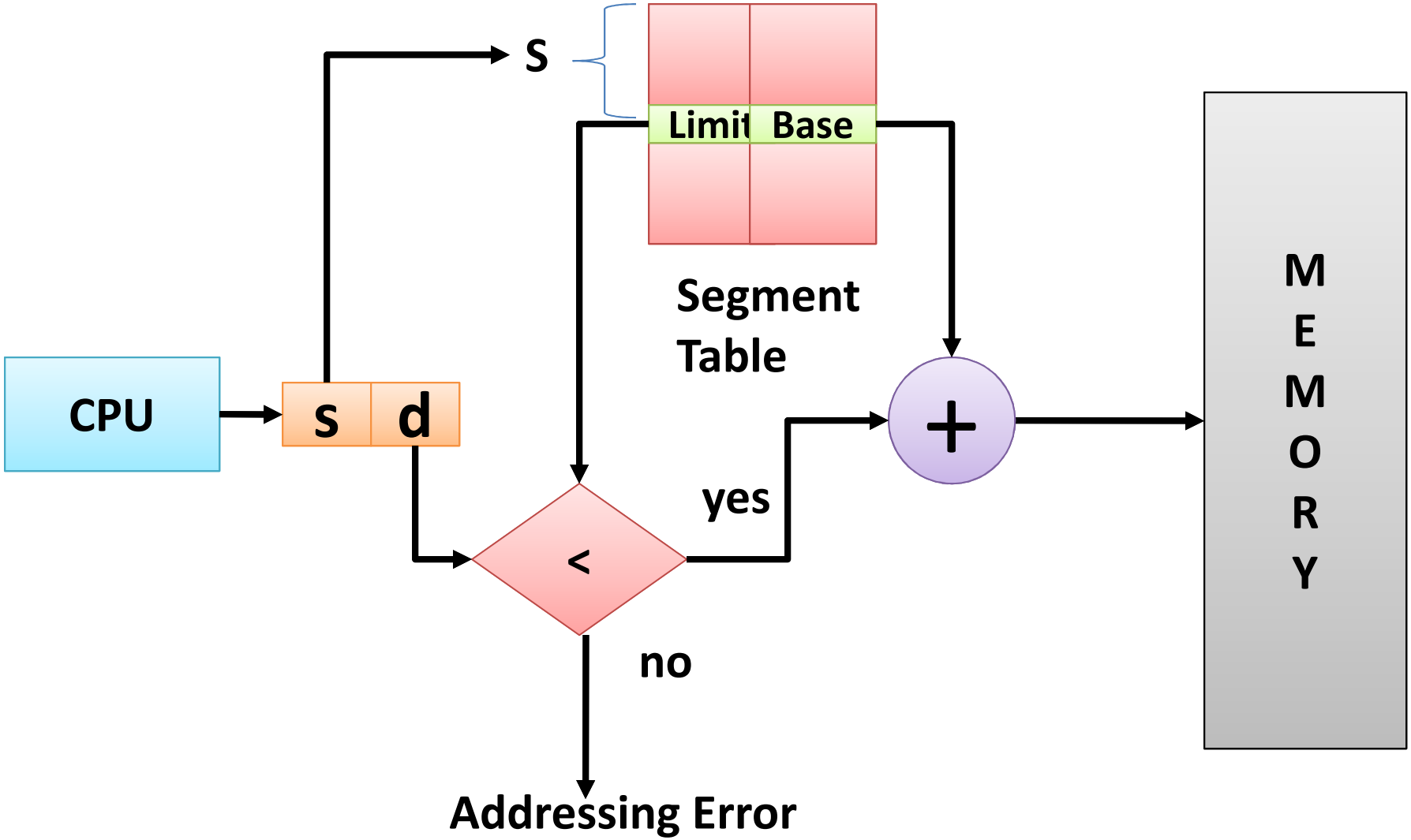
Every memory access get translated to Two accesses

☹ ☹ ☹

Segmentation Architecture (Cont.)

- Protection
  - With each entry in segment table associate:
    - validation bit = 0 $\Rightarrow$ illegal segment
    - read/write/execute privileges
- Protection bits associated with segments; code sharing occurs at segment level
- Since segments vary in length, memory allocation is a dynamic storage-allocation problem
- A segmentation example is shown in the following diagram

Segmentation Hardware



Segmentation and Paging

- Divide the program into smaller block call segments : User view, already segmented
  - Finer granularity, less fragmentation
    - **Mustard in Bag Vs Brinjal in Bag**
- Divide the program in to smaller but uniform size unit called page
  - A program may contain many pages
  - Last page of program may be partially filled
- Divide the memory in to smaller size units called Frame
- Page get mapped to Frame

Paging

Paging

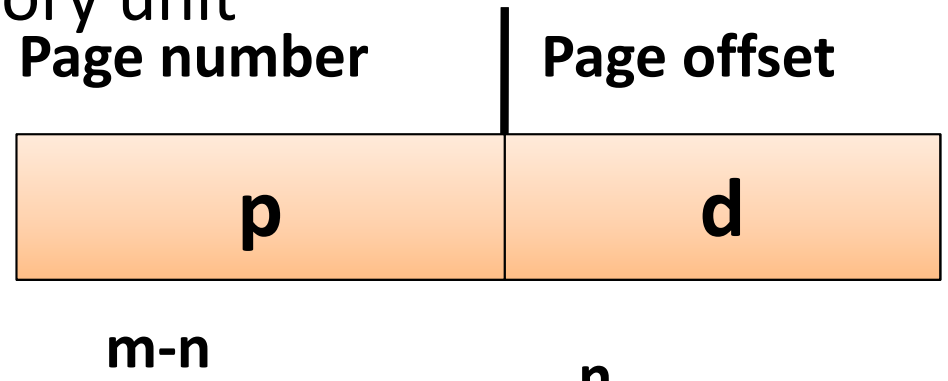
- Physical address space of a process can be noncontiguous; process is allocated physical memory whenever the latter is available
  - Avoids external fragmentation
  - Avoids problem of varying sized memory chunks
- Divide physical memory into fixed-sized blocks called **frames**
  - Size is power of 2, between 512 bytes and 16 Mbytes , **getpagesize() in C → Demo**

Paging

- Divide logical/program memory into blocks of same size called **pages**
- Keep track of all free frames
- To run a program of size ***N*** pages, need to find ***N*** free frames and load program
- Set up a **page table** to translate logical to physical addresses
- Backing store likewise split into pages
- Still have Internal fragmentation

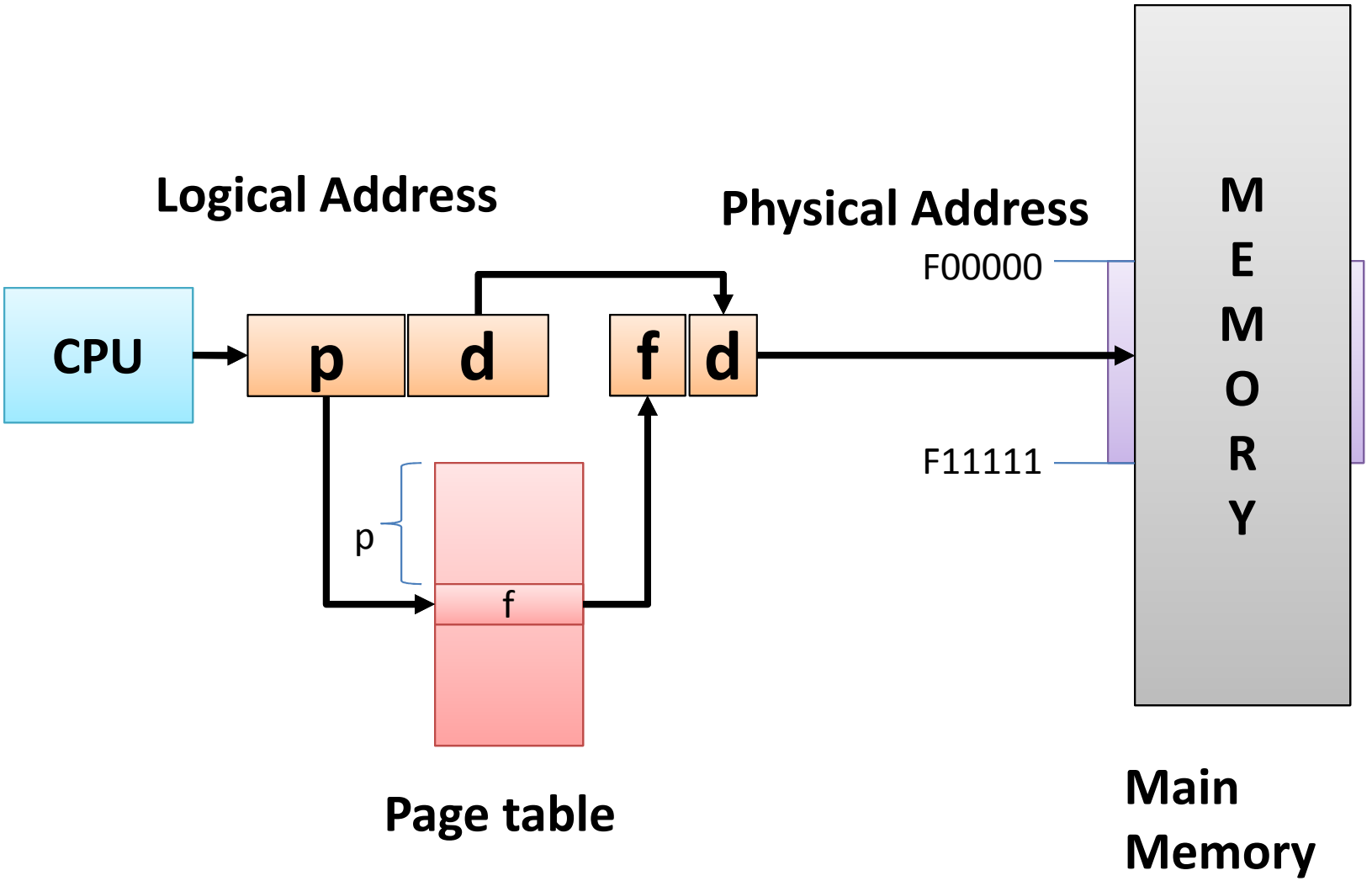
Address Translation Scheme

- 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

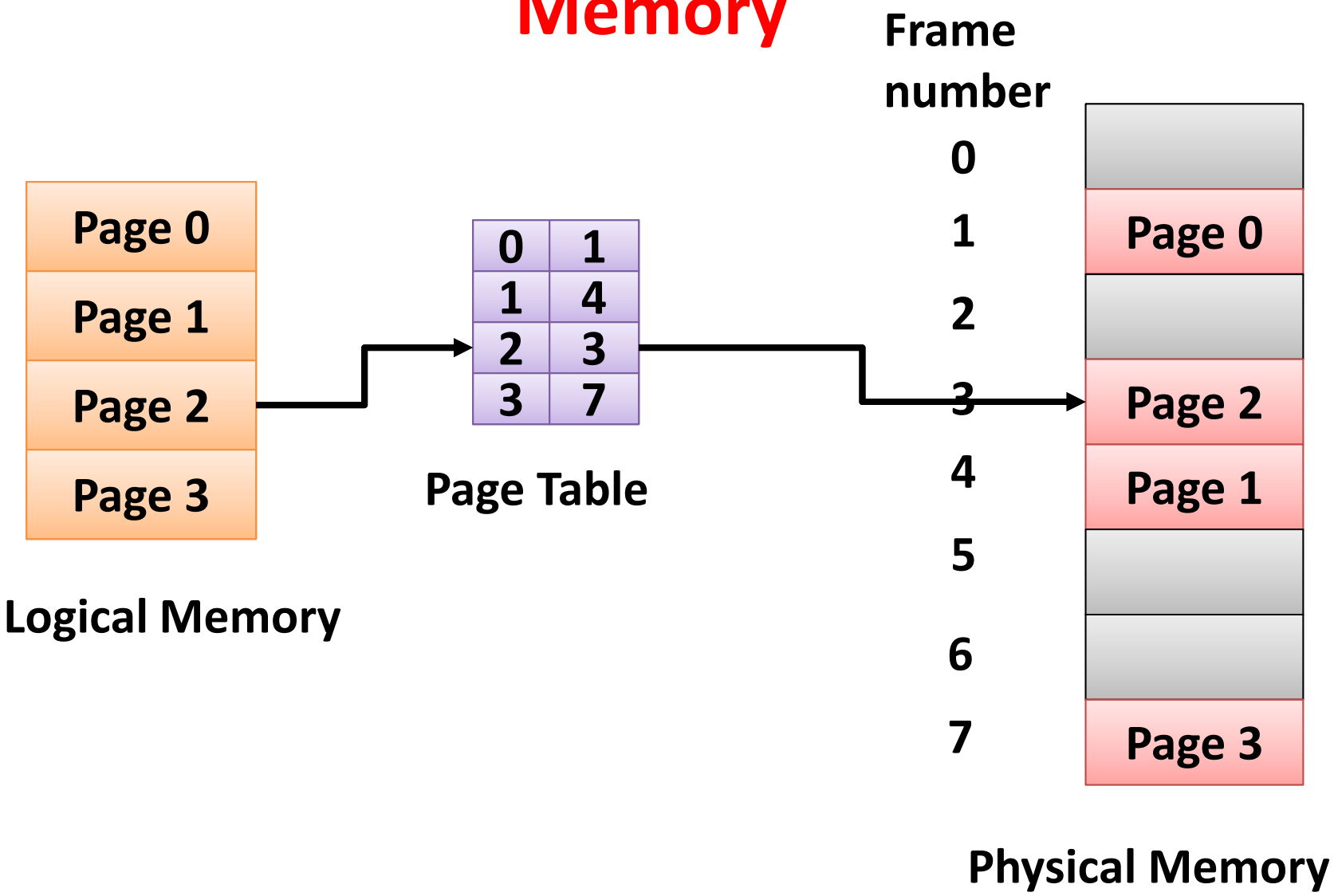


  - For given logical address space 2^m and page size 2^n

Paging Hardware



Paging Model of Logical and Physical Memory



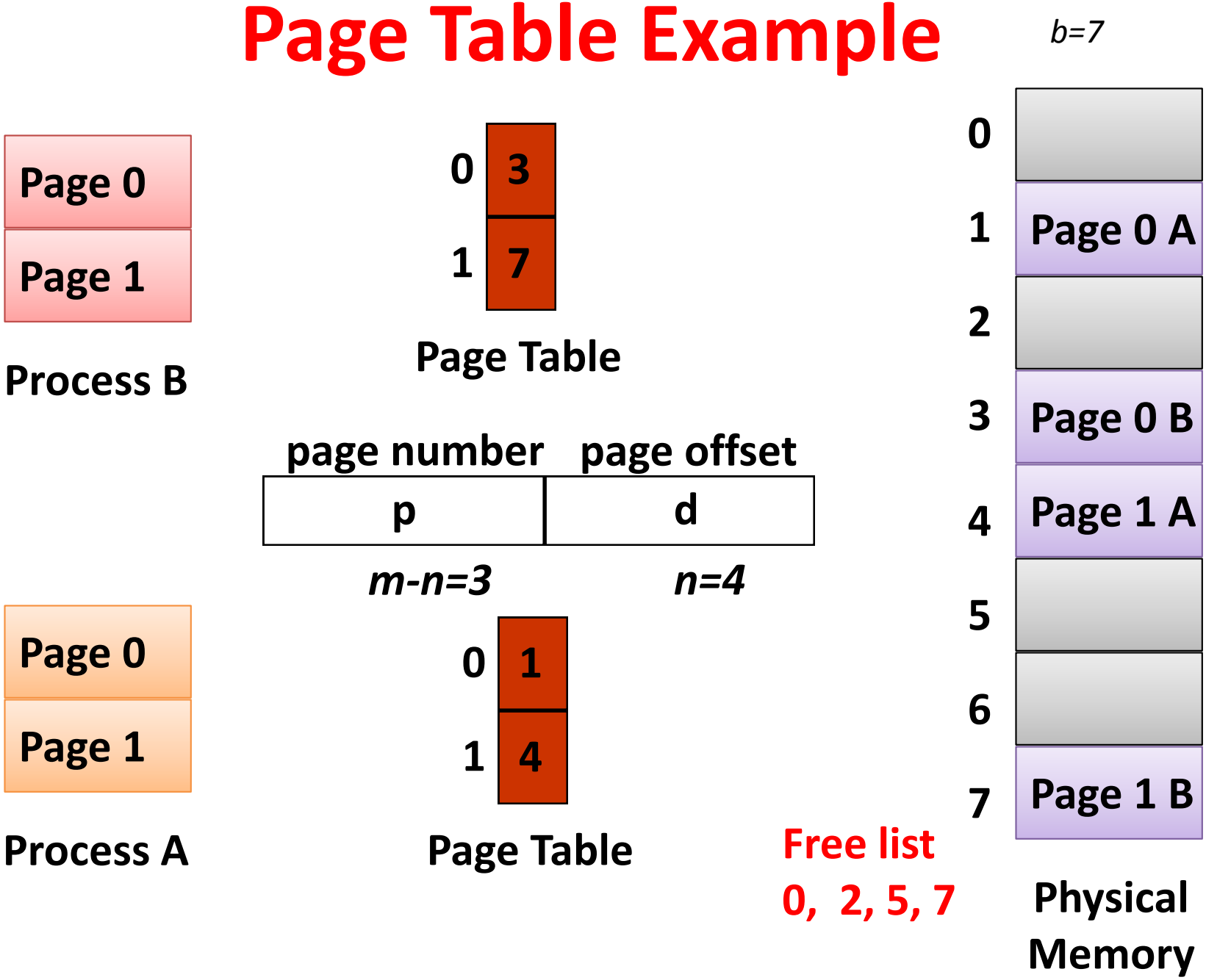
Paging (Cont.)

- Calculating internal fragmentation
  - Page size = 2,048 bytes
  - Process size = 72,766 bytes
  - 35 pages + 1,086 bytes
  - Internal fragmentation of 2,048 - 1,086 = 962 bytes
  - Worst case fragmentation = 1 frame – 1 byte
- Another Example: Suppose you want to withdraw Rs 533 from ATM
  - ATM allow only multiple of 100
  - Either 500 or 600, Fragmentation 100-33=67
  - **Worst case, need to withdraw Rs 501, Frag.= 100-1= 99**

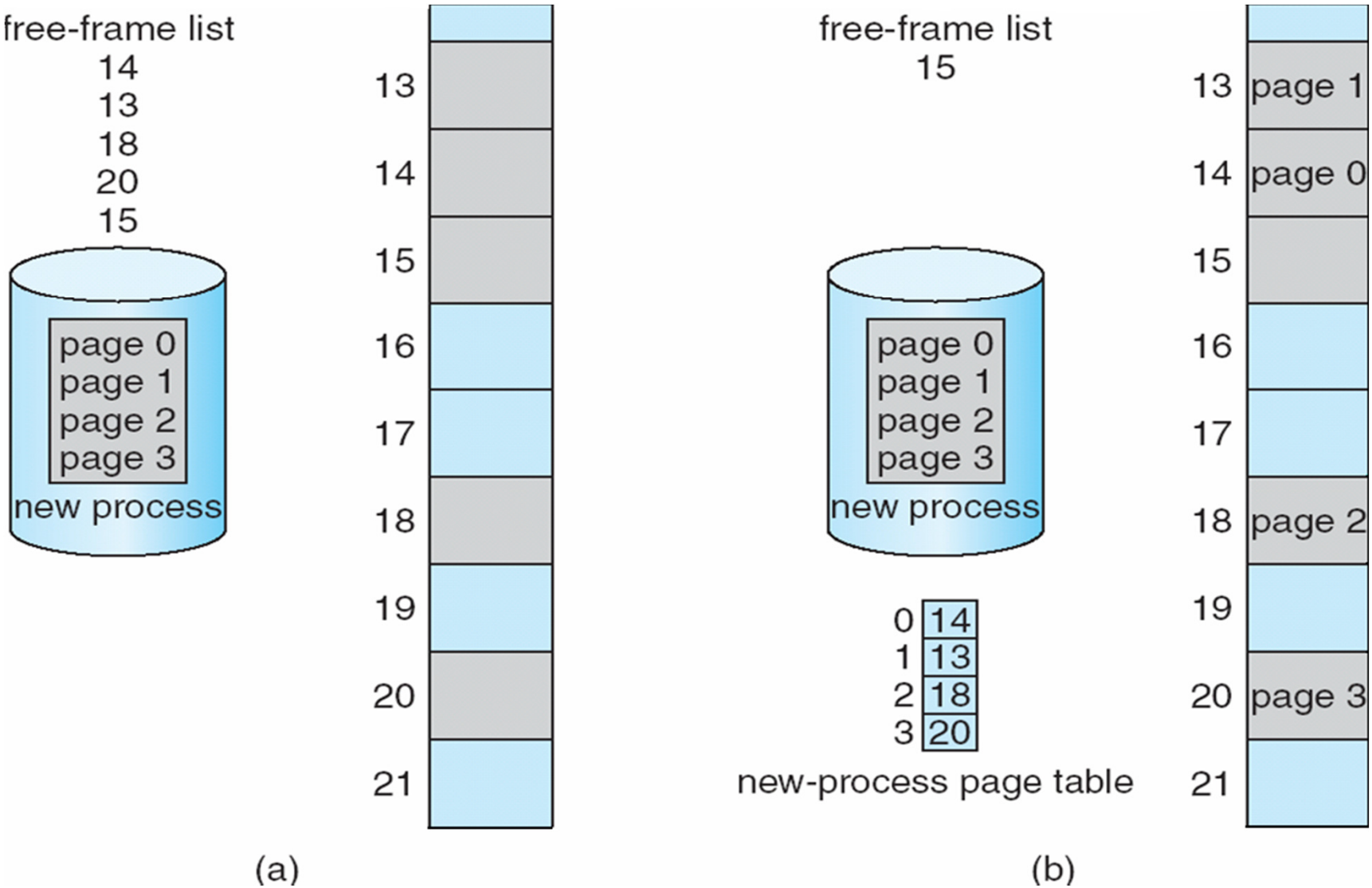
Paging (Cont.)

- Calculating internal fragmentation Ctd..
  - On average fragmentation = 1 / 2 frame size
  - So small frame sizes desirable?
  - But each page table entry takes memory to track
  - Page sizes growing over time
    - Solaris supports two page sizes – 8 KB and 4 MB
- Process view and physical memory now very different
- By implementation process can only access its own memory

Page Table Example



Free Frames



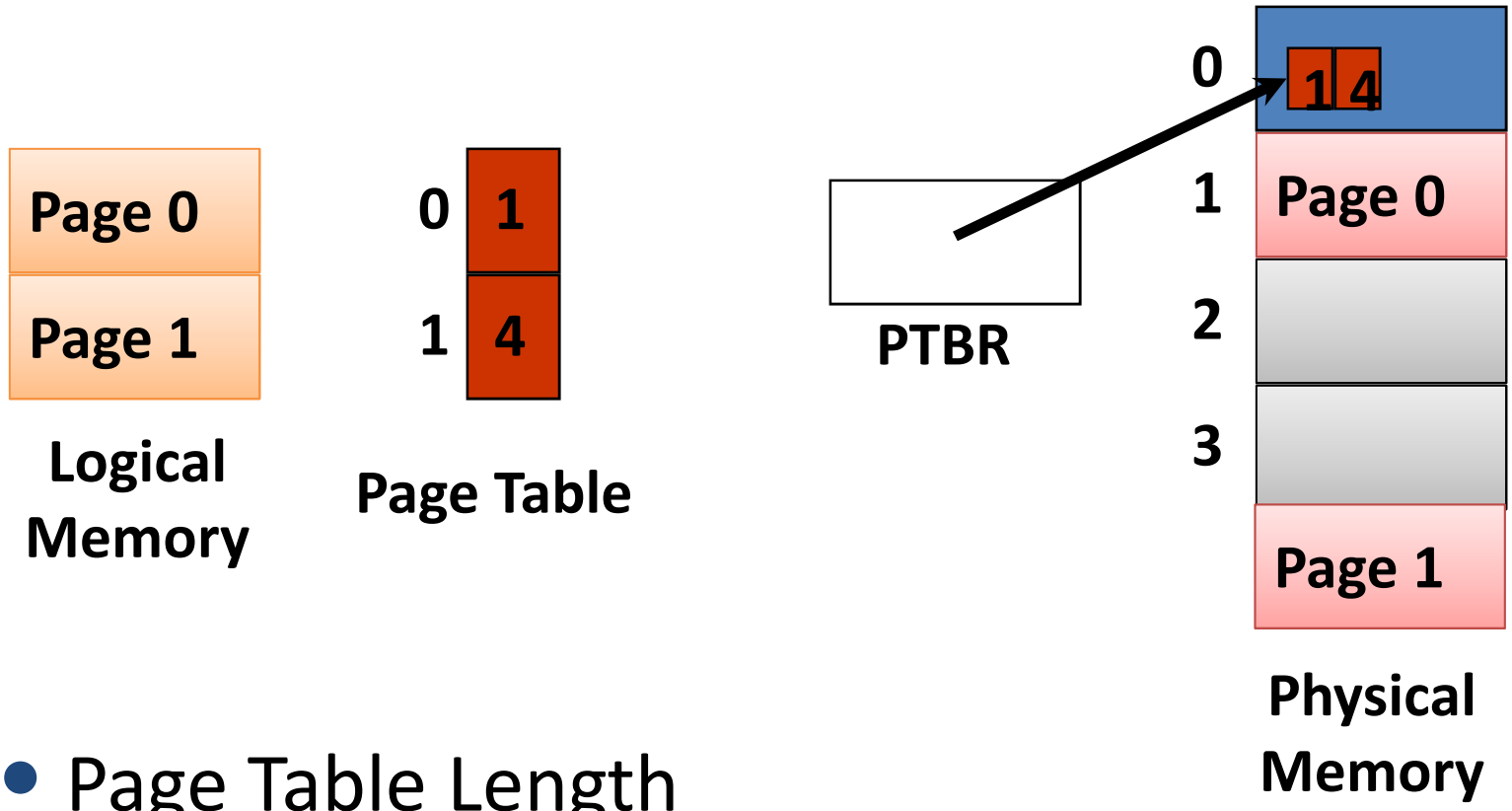
Before allocation

After allocation

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

Implementation of Page Table



- Page Table Length
- **Two memory accesses per data/inst access.**
  - **Solution? *Associative Registers***

Implementation of Page Table (Cont.)

- The two memory access problem can be solved by the use of a special fast-lookup hardware cache called **associative memory** or **translation look-aside buffers** (**TLBs**)
  - **Get implemented in Processor as CAM**
- 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

Implementation of Page Table (Cont.)

- 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

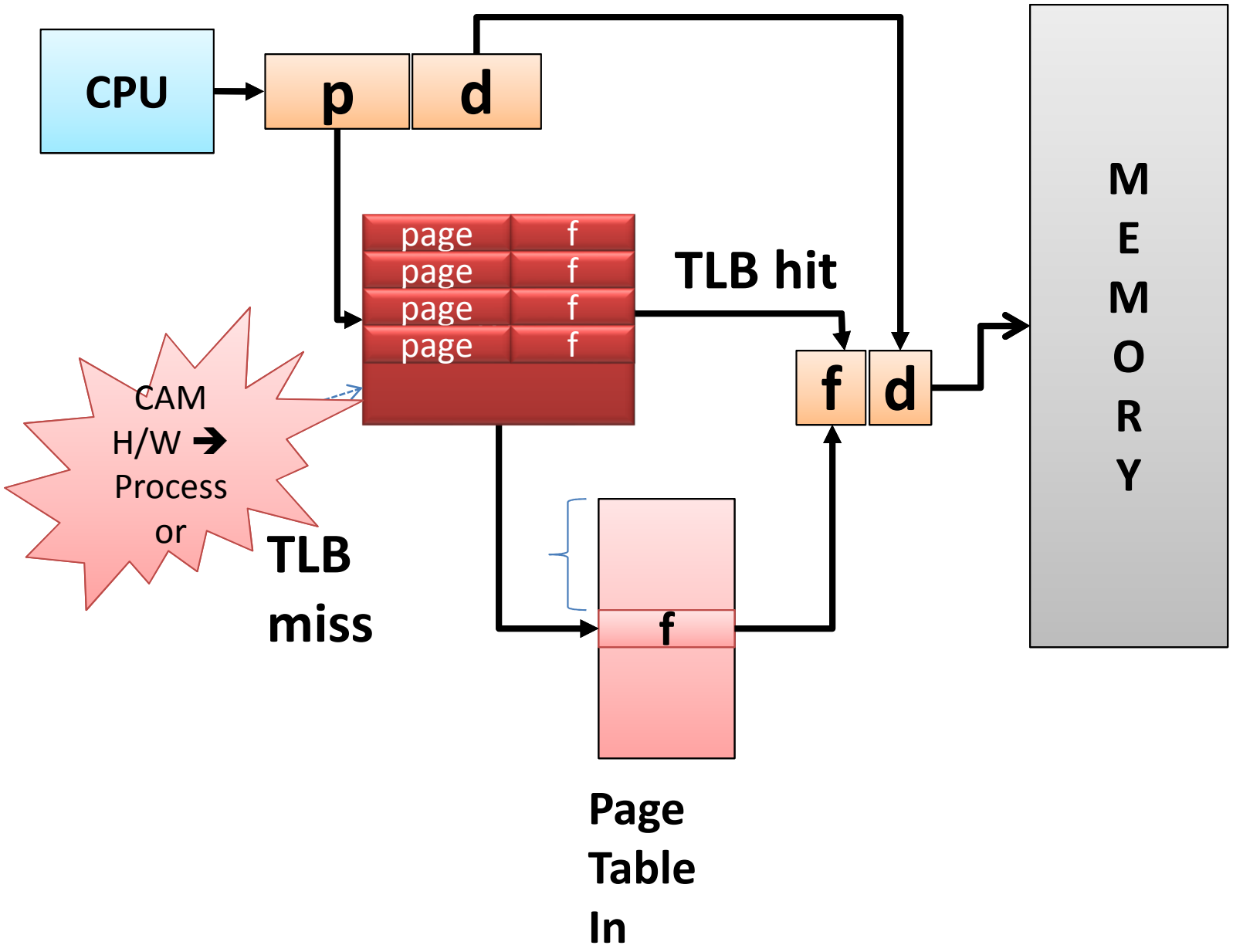
Associative Memory: CAM

- Associative memory – parallel search

Page #	Frame #
1	2
3	15
2	12

- **Content Addressable memory : CAM**
- Address translation (p, d)
  - If p is in associative register, get frame # out
  - Otherwise get frame # from page table in memory

Paging Hardware With TLB



Thanks