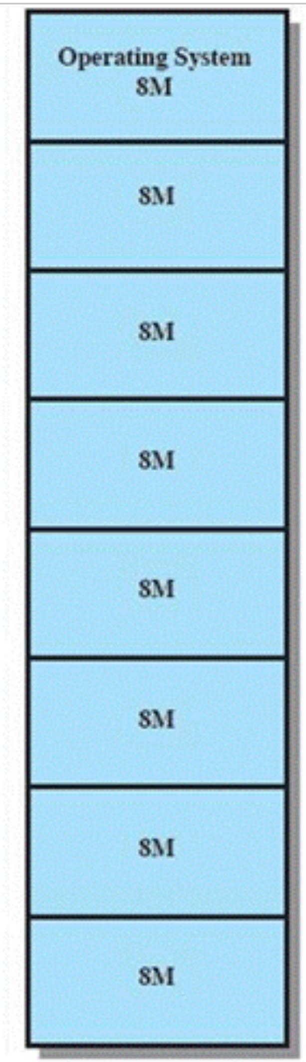
Memory Management

Types of Memory Management

- Fixed Partitioning
- Dynamic Partitioning
- Paging
- Segmentation
- Segmentation with Paging

Fixed Partitioning

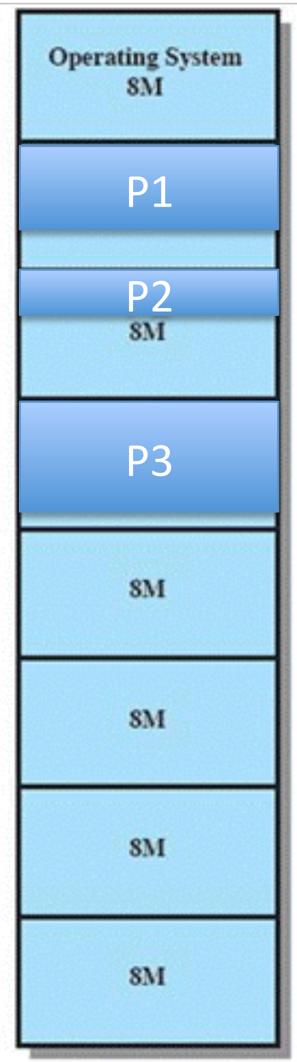
- Equal-size partitions
  - Any process whose size is less than or equal to the partition size can be loaded into an available partition



(a) Equal-size partitions

Fixed Partitioning

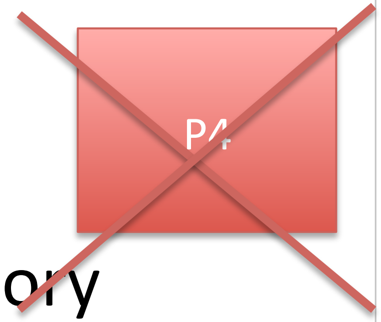
- Equal-size partitions
  - Any process whose size is less than or equal to the partition size can be loaded into an available partition

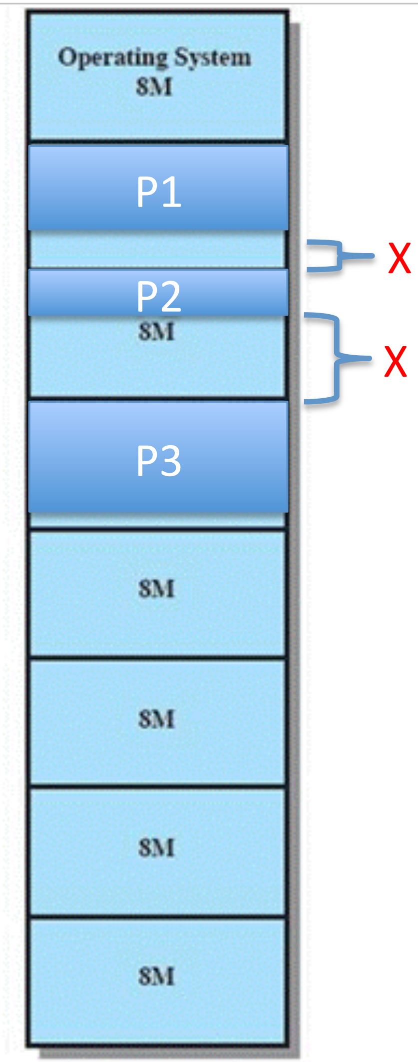


(a) Equal-size partitions

Fixed Partitioning

- Equal-size partitions
  - Any process whose size is less than or equal to the partition size can be loaded into an available partition
- Problems
  - Large process can't fit
  - Small process wastes memory
    - *Internal fragmentation*

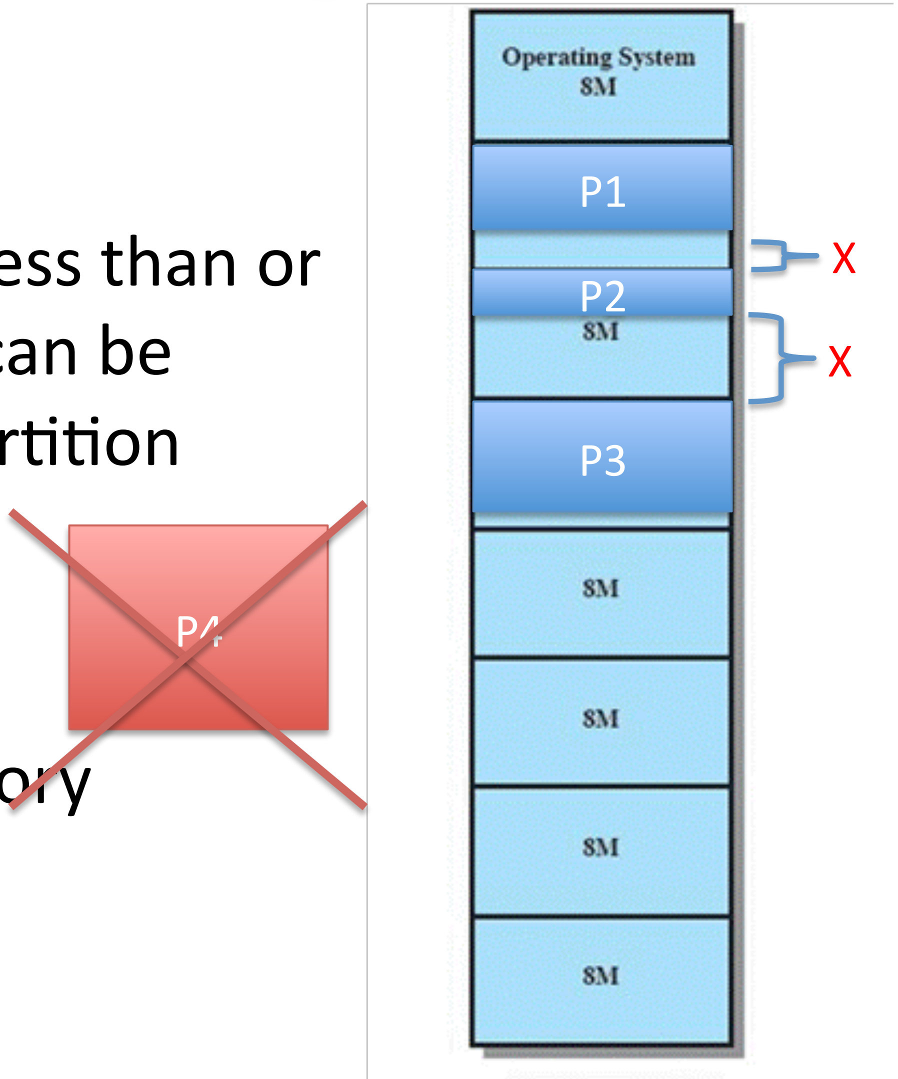




(a) Equal-size partitions

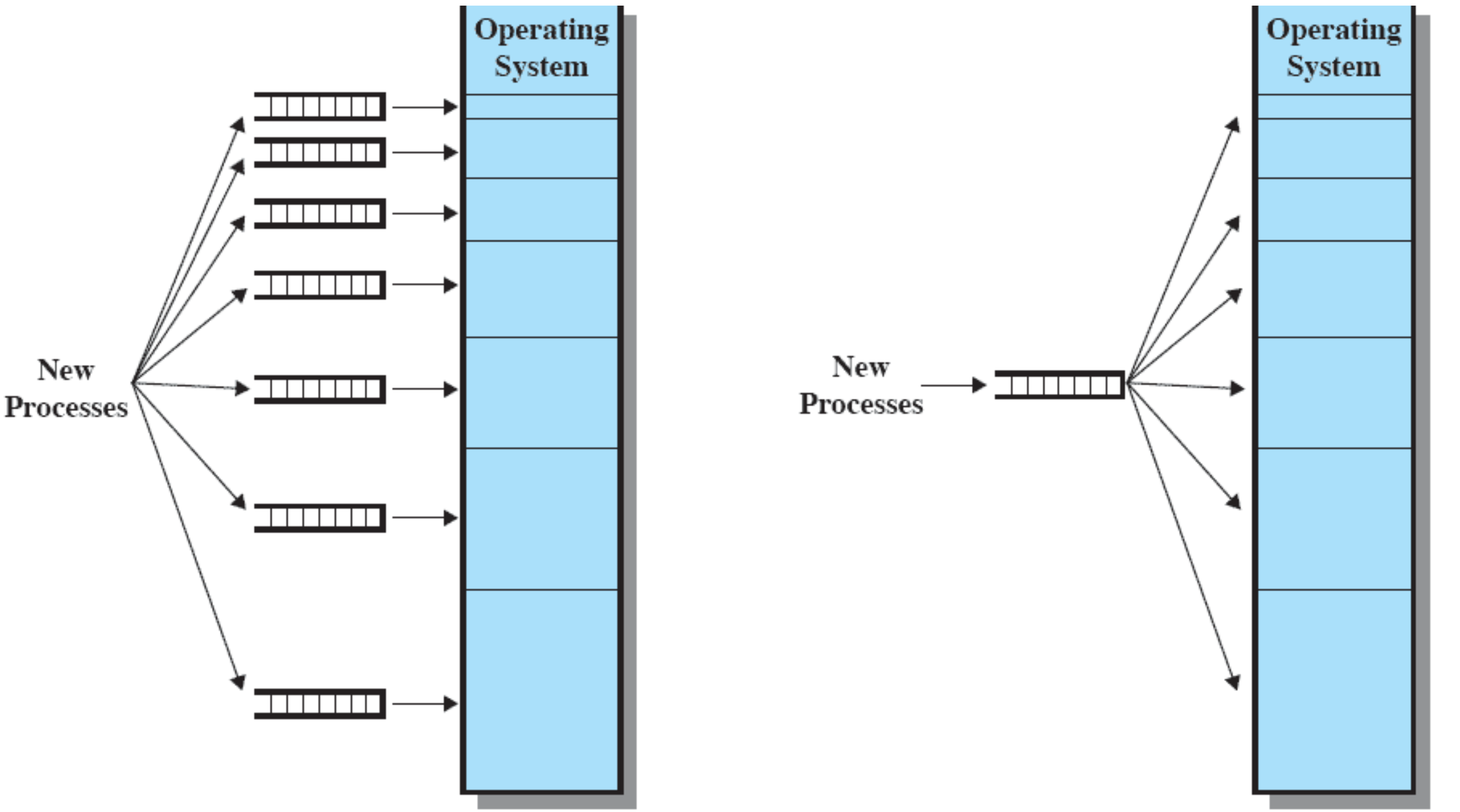
Fixed Partitioning

- Equal-size partitions
  - Any process whose size is less than or equal to the partition size can be loaded into an available partition
- Problems
  - Large process can't fit
  - Small process wastes memory
    - *Internal fragmentation*



(a) Equal-size partitions

Varied-Size Fixed Partitioning



(a) One process queue per partition

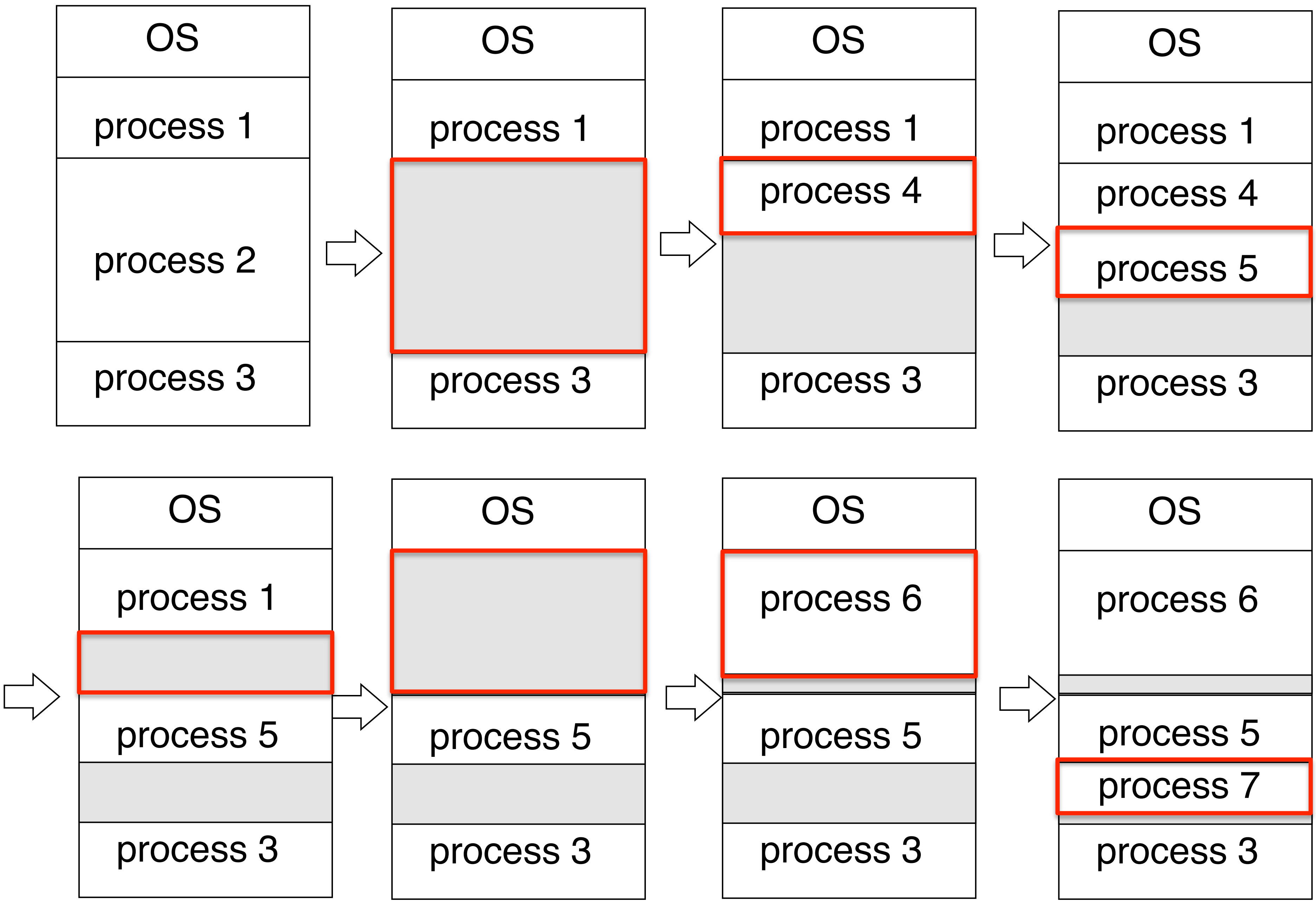
(b) Single queue

Problems with Fixed Partitions

- The number of active processes is limited by the system (to the pre-determined number of partitions)
- A large number of very small process will not use space efficiently
- Solutions?

Dynamic Partitioning

- Partitions are of variable length and number
- Process is allocated as much as required
- OS decides which free block to allocate


OS
process 1
process 2
process 3
OS
process 1
process 3
OS
process 1
process 4
process 3
OS
process 1
process 4
process 5
process 3
OS
process 1
process 5
process 3
OS
process 5
process 3
OS
process 6
process 5
process 3
OS
process 6
process 5
process 7
process 3

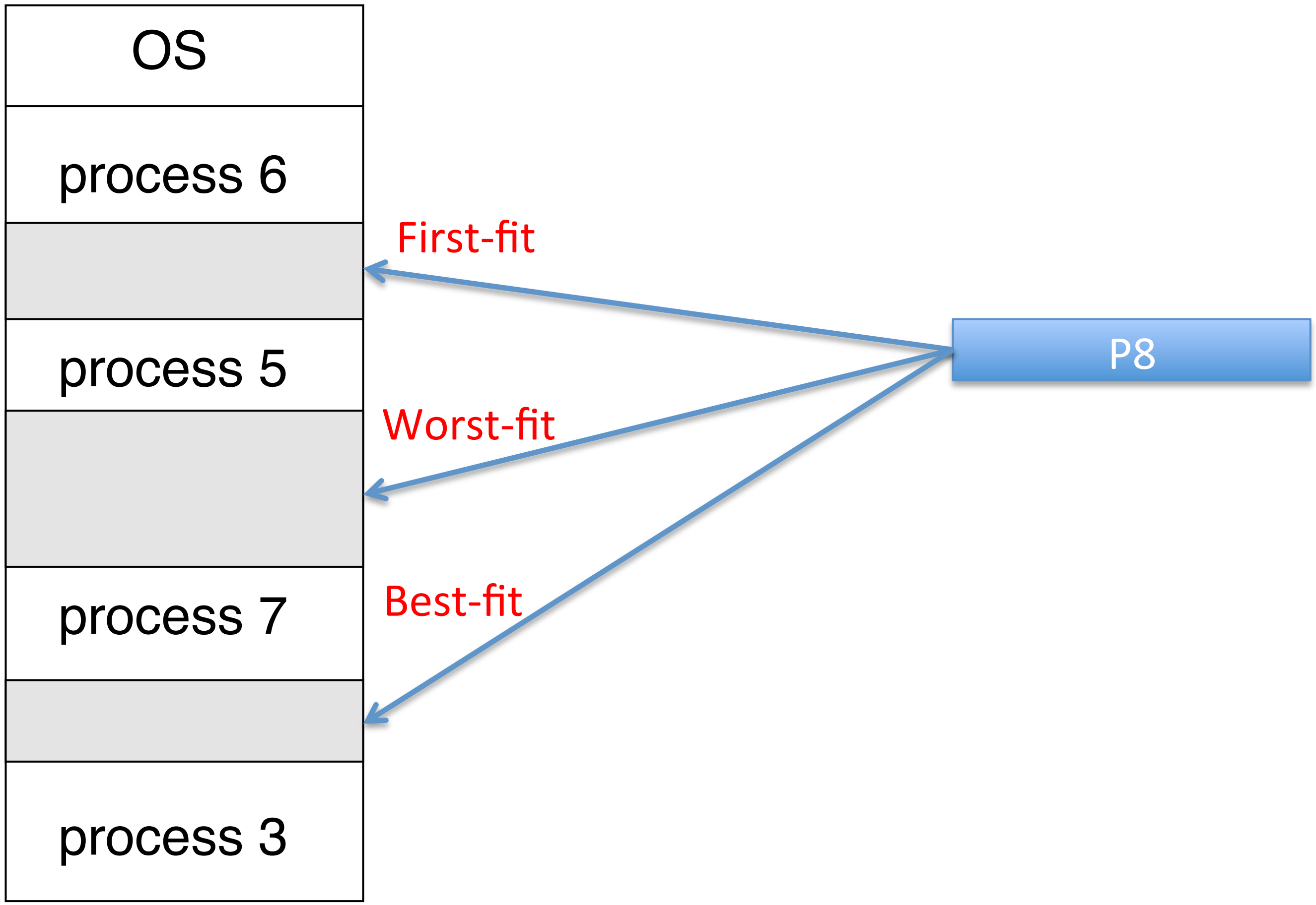
Allocation Strategy

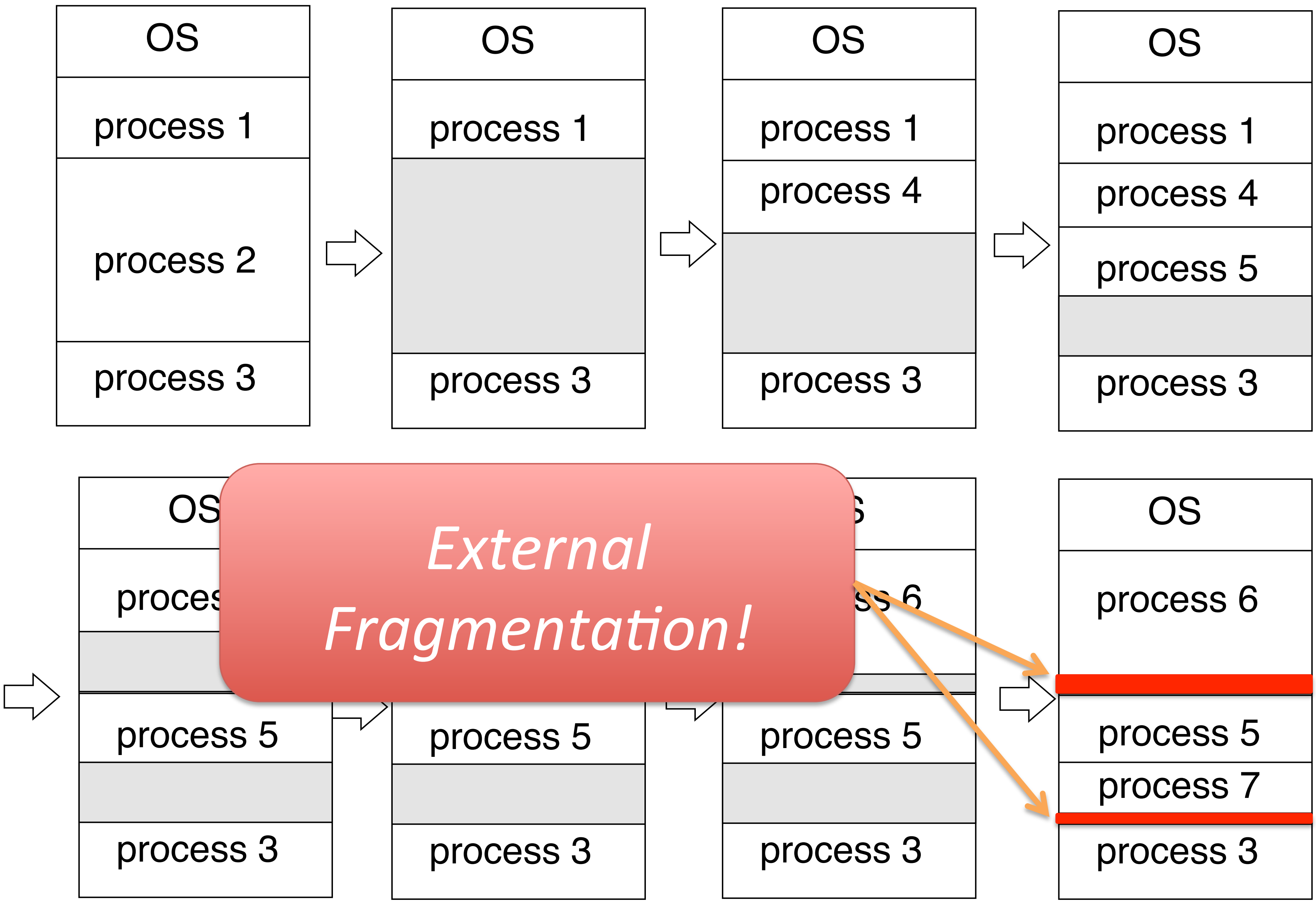
How to satisfy a request of size *n* from a list of free holes?

- **First-fit**: Allocate the *first* hole that is big enough
- **Best-fit**: Allocate the *smallest* hole that is big enough; must search entire list, unless ordered by size
  - Produces the smallest leftover hole
- **Worst-fit**: Allocate the *largest* hole; must also search entire list
  - Produces the largest leftover hole

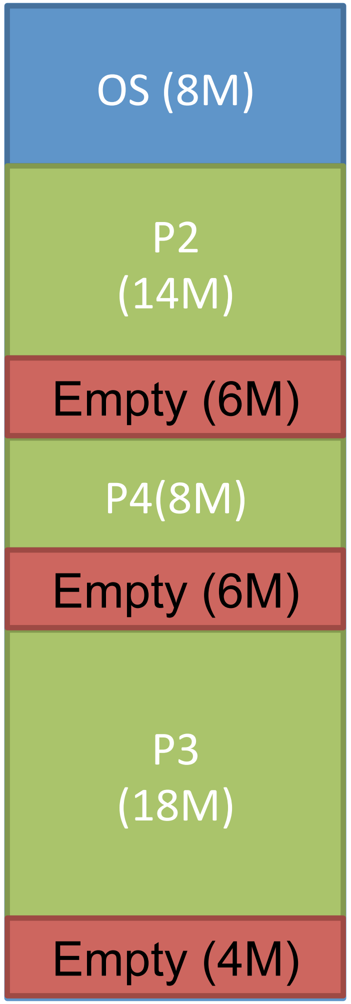
First-fit and best-fit better than worst-fit in terms of speed and storage utilization

Allocation Strategy




OS
process 1
process 2
process 3
OS
process 1
process 3
OS
process 1
process 4
process 3
OS
process 1
process 4
process 5
process 3
OS
process 5
process 3
process 5
process 3
process 5
process 3
OS
process 6
process 5
process 7
process 3
External Fragmentation!

Dynamic Partitioning Example



- ***External Fragmentation***
  - Memory external to all processes is fragmented
- ***Compaction***
  - OS moves processes so that they are contiguous
  - Time consuming and wastes CPU time

Fragmentation

- **External Fragmentation** – total free memory is enough for new process, but it is not contiguous
- **Internal Fragmentation** – allocated memory to a process but never used

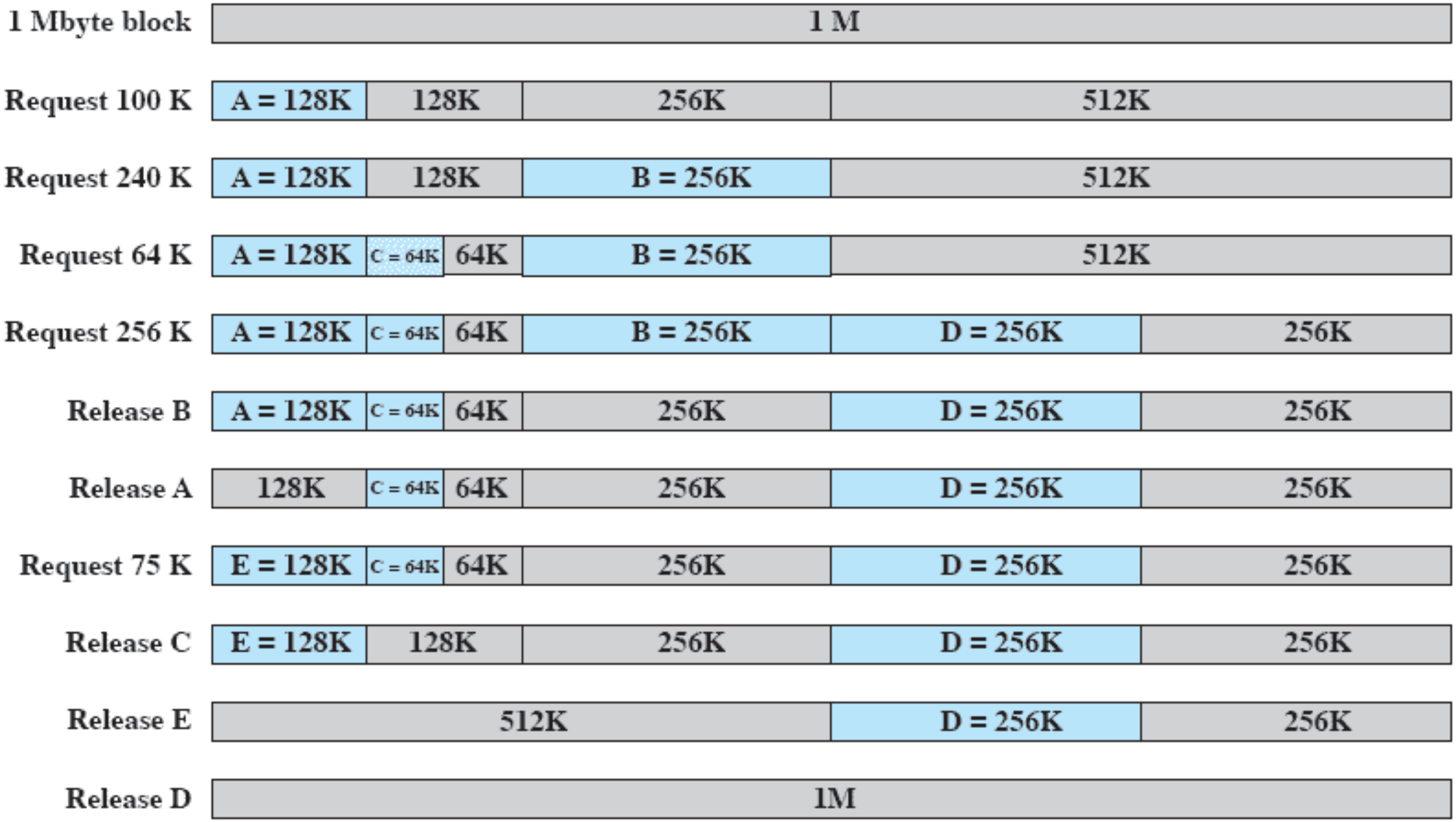
- *Fixed partitioning has only internal frag.*
- *Dynamic partitioning has only external frag.*

- First fit has 50-percent rule
  - given *N* blocks allocated, 0.5 *N* blocks lost to external fragmentation
  - Memory utilization = 2/3

Buddy System

- For allocation of a process
  - Divide the free memory block into two blocks
  - until it best fits to the block
- For deallocation of a process
  - Merge the freed block with buddy block
  - buddy block
    - The other block when it was divided into two
- Has both internal/external fragmentations

Example of Buddy System



Tree Representation of Buddy System

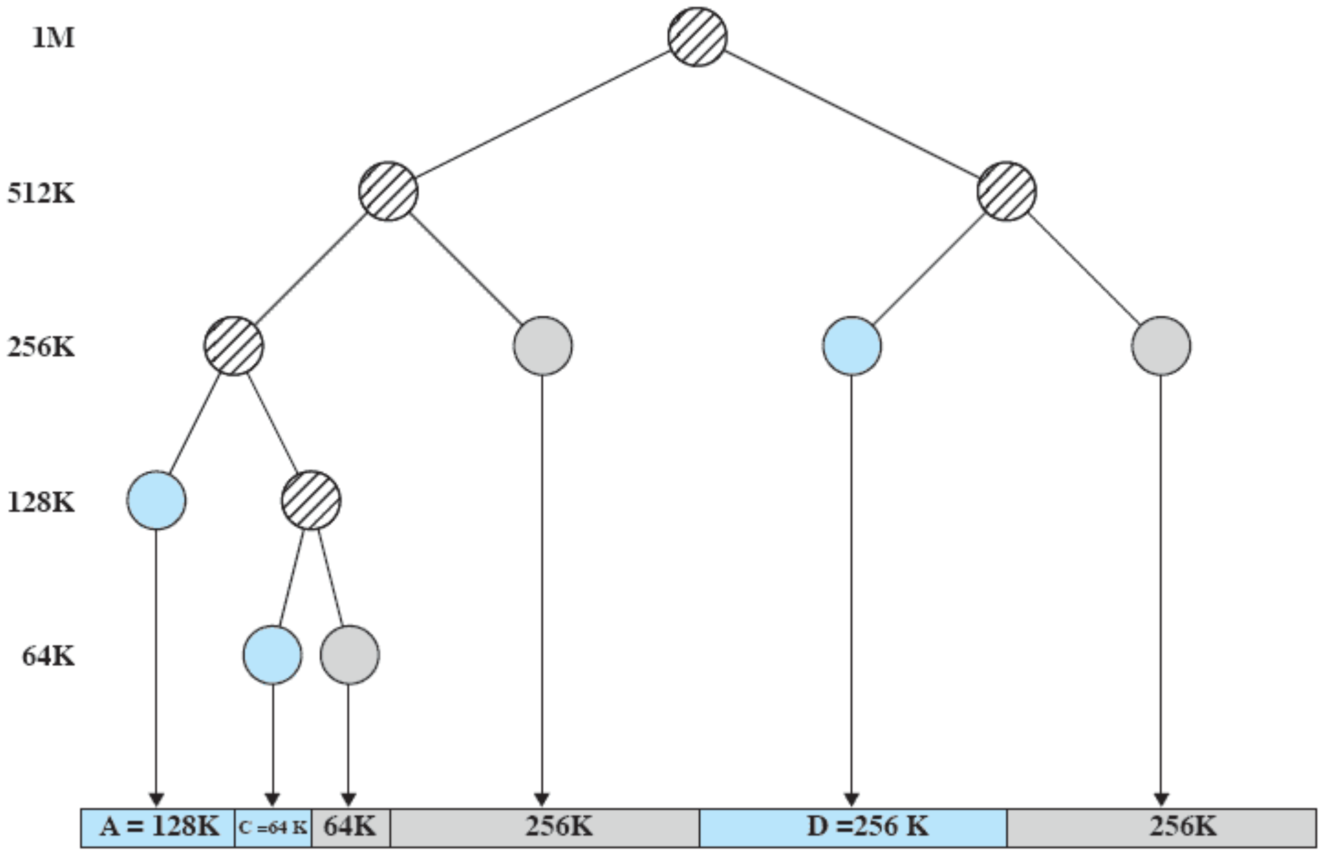


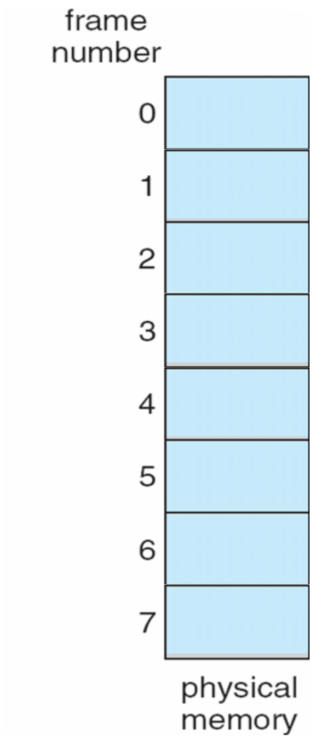
Figure 7.7 Tree Representation of Buddy System

Paging

- Goal
  - No external fragmentation problem
  - Efficient memory sharing
  - Flexible memory use
- Idea
  - Divide a process into multiple fragments
  - Allocation each fragment anywhere
  - Maintain where the fragments are

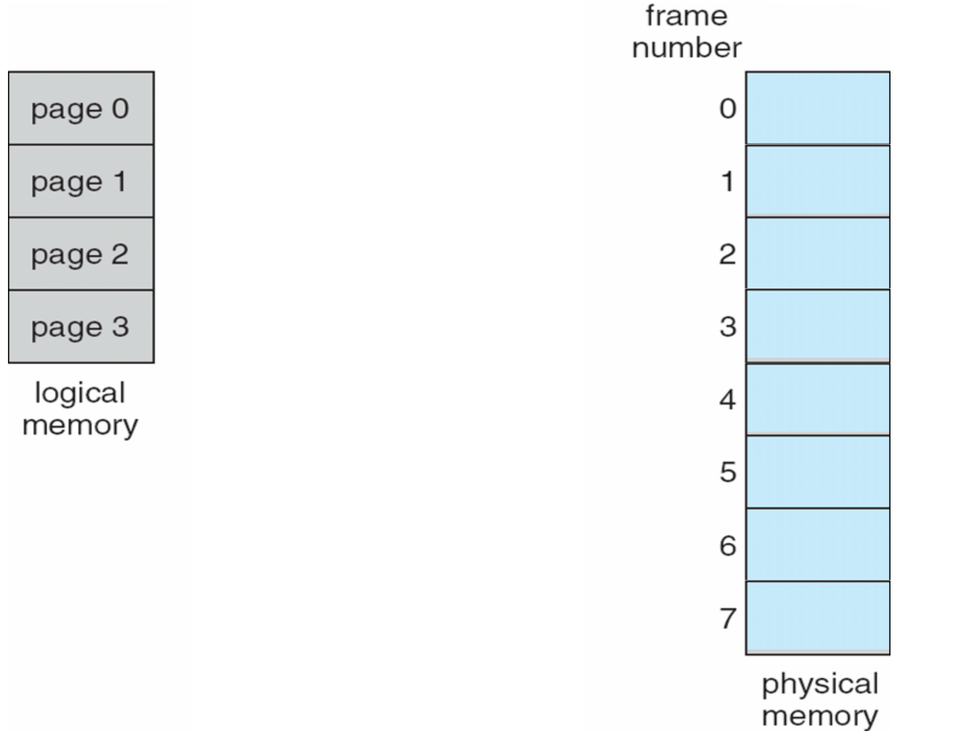
Paging

- Partition physical memory into equal size frames



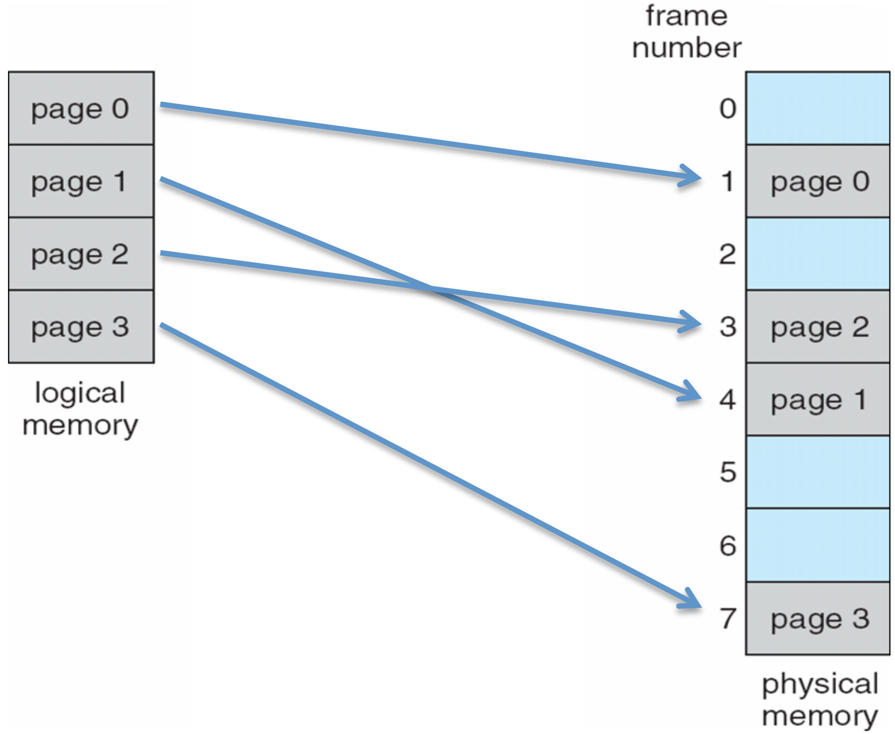
Paging

- Partition physical memory into equal size frames
- Divide logical memory into same-size pages



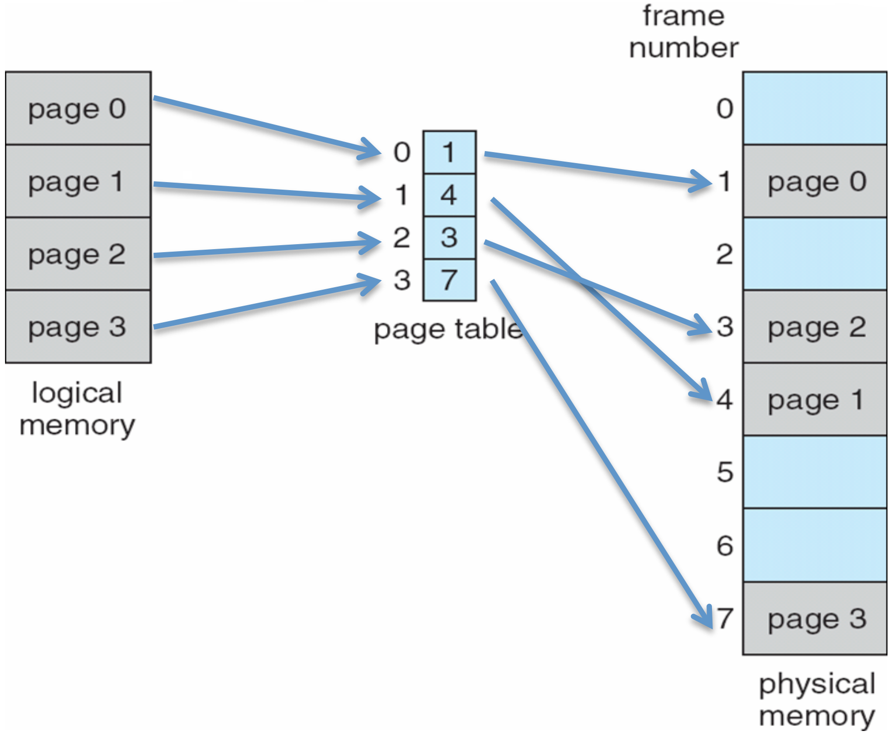
Paging

- Partition physical memory into equal size frames
- Divide logical memory into same-size pages
- Each page can go to any free frame



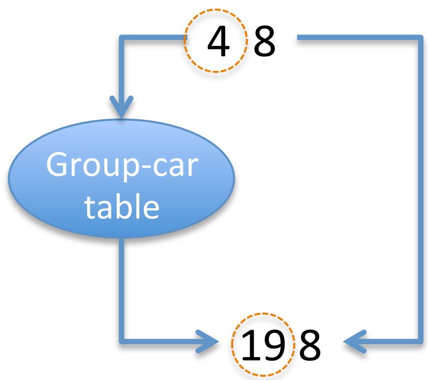
Paging

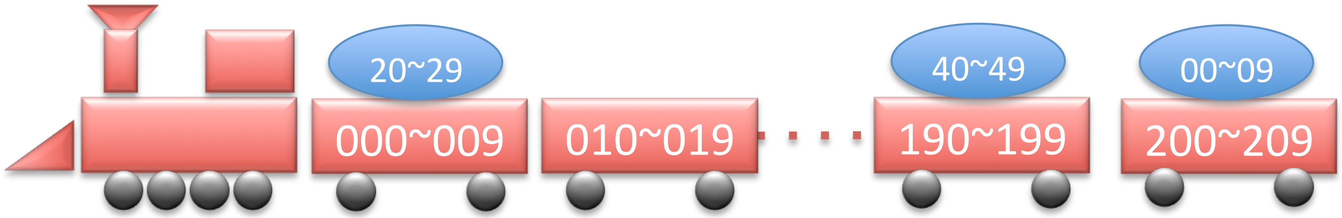
- Partition physical memory into equal size frames
- Divide logical memory into same-size pages
- Each page can go to any free frame
- OS knows the mapping
  - page table



Addressing with Paging

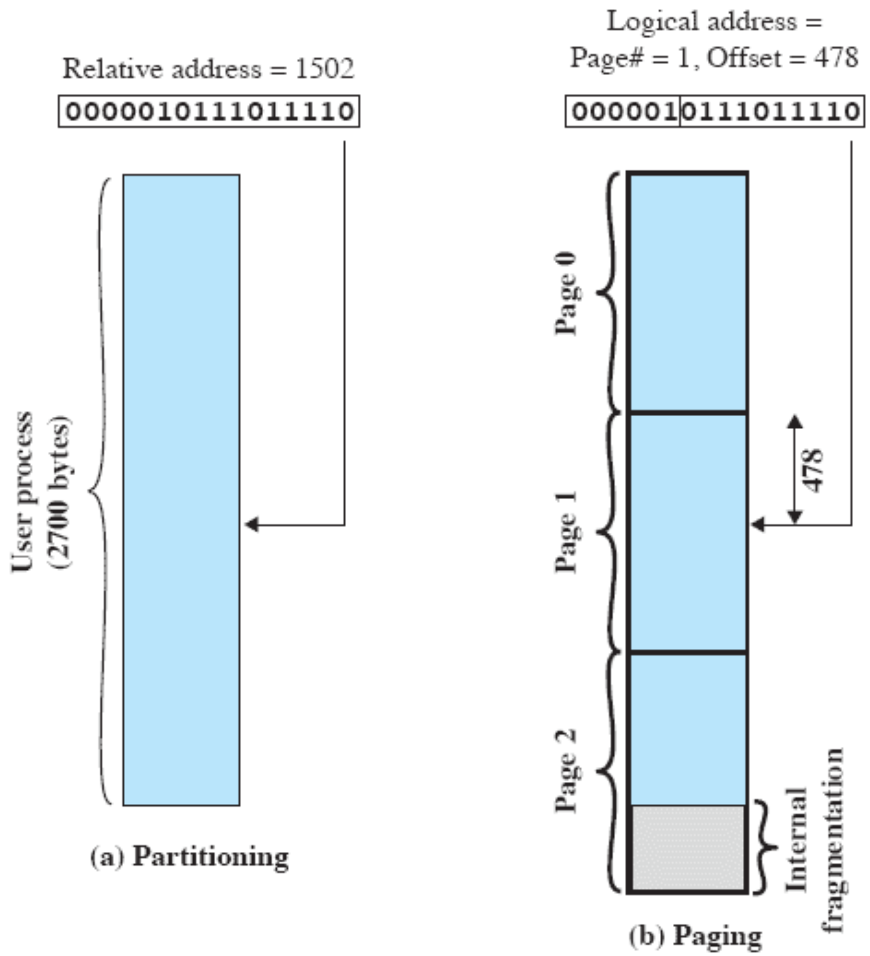
- Analogy
  - We have 100 students, from 00 to 99
  - 10 groups: 00~09 (group 0), 10~19 (group 1), 20~29 (g2), …
  - Ride on a train with 100 cars, 10 people on each
  - Each group on the same car
  - Mapping table: which group on which car
    - *Car*(group) ex: *Car*(4) = 19
  - Where is student 48?
    - 48 → (Car(4)=19, 8) = 198



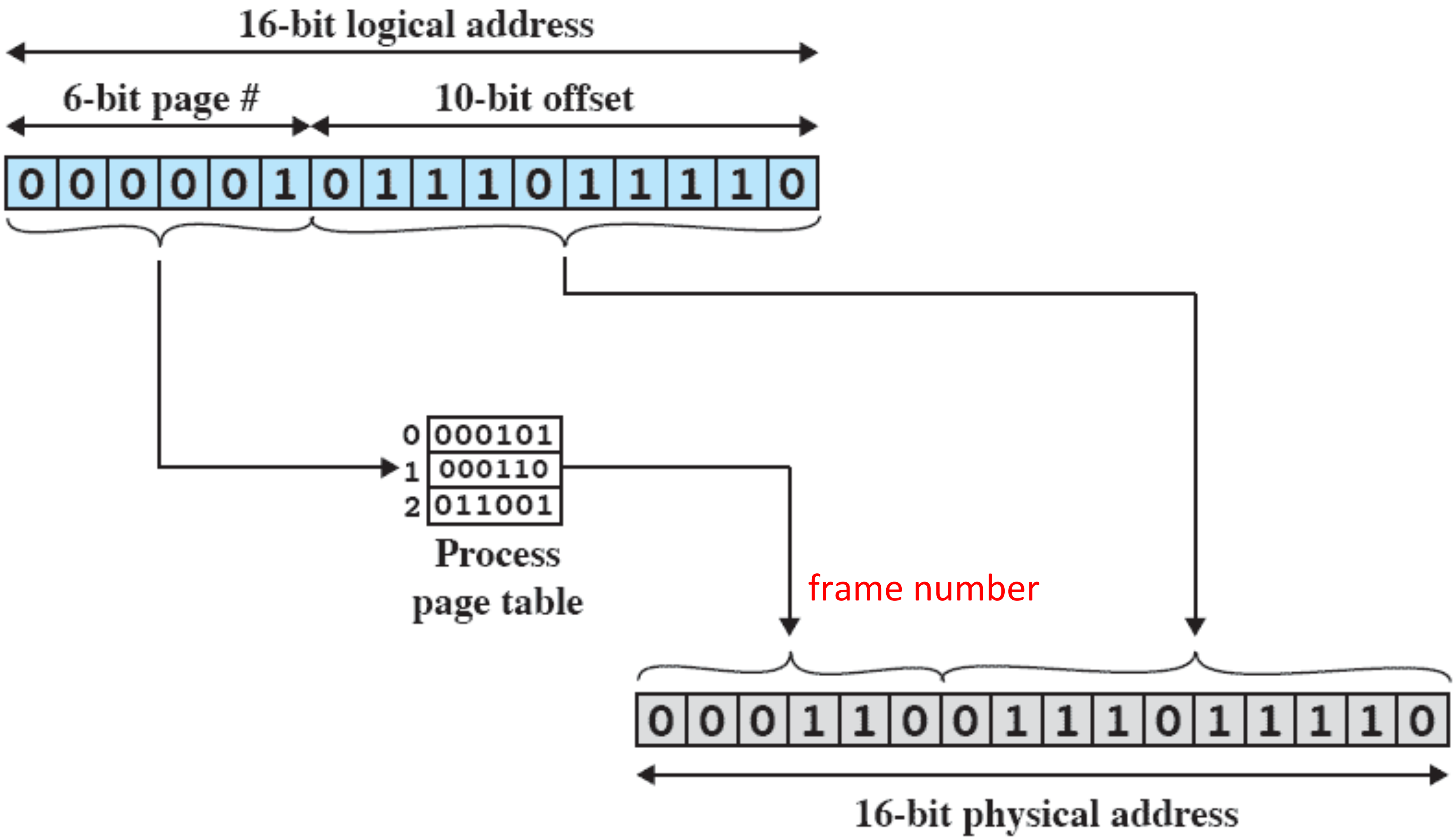


Paging: Logical Addresses

- 16-bit address, page size 1K=2^{10}, first 6 bit=page #, last 10bit = offset



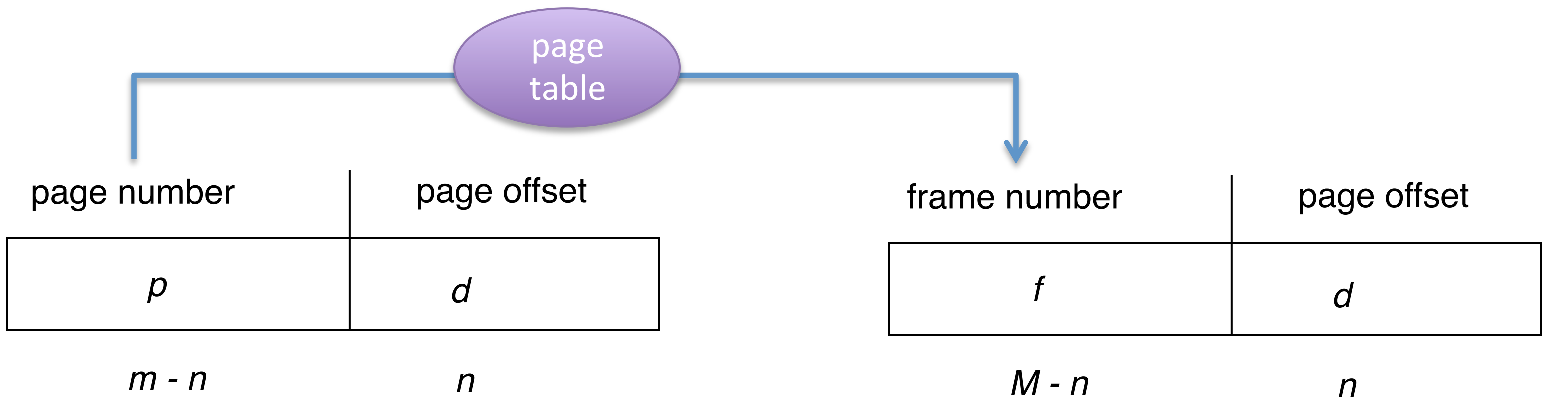
Paging: Logical to Physical Address



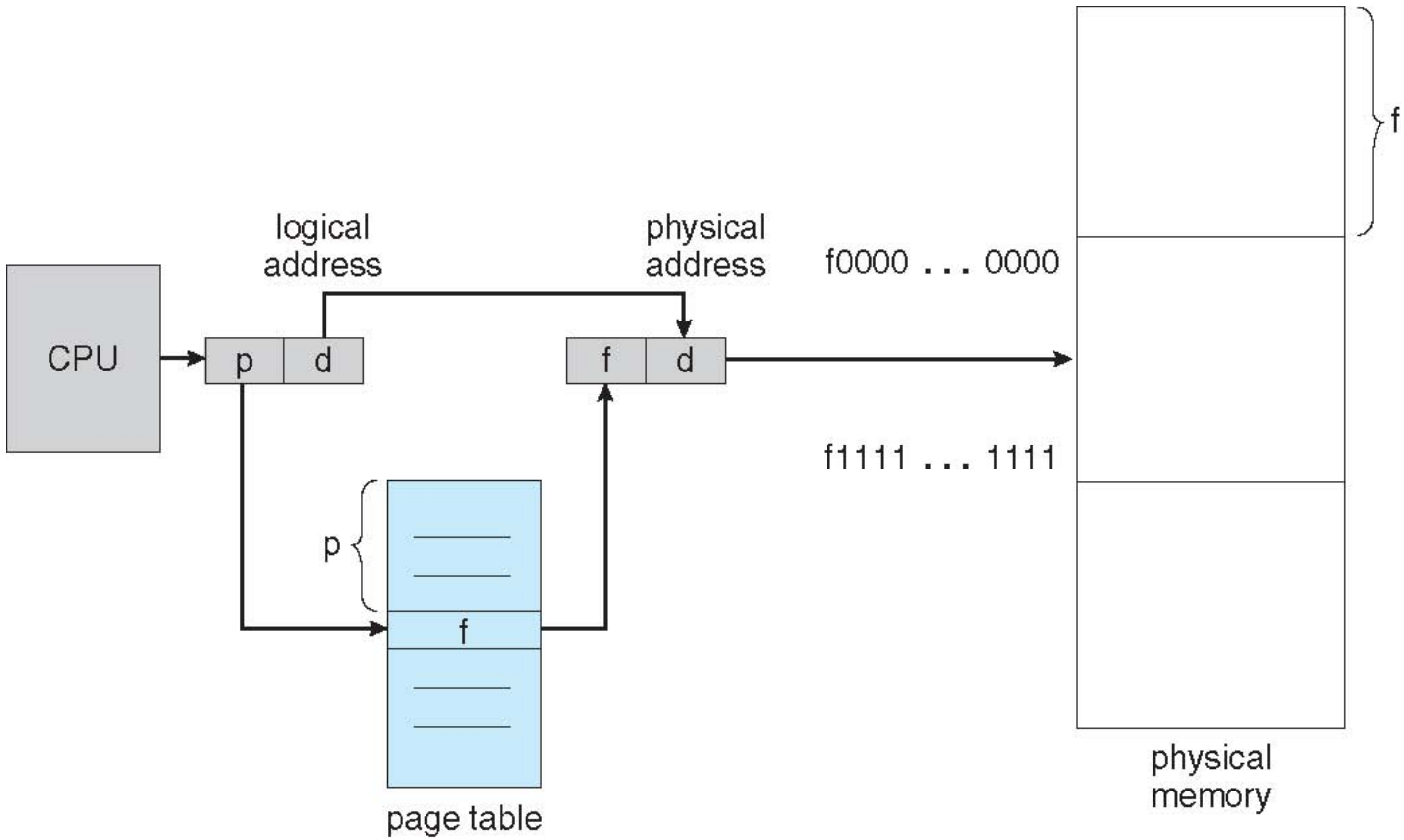
(a) Paging

Address Translation Scheme

- Address generated by CPU is divided into:
  - **Page number (*p*)**
    - index into a **page table** = (page #, frame #)
  - **Page offset (*d*)**
    - offset within the page (frame)
  - Given m bits logical address, page size 2^n
    - last n bit = offset = 0 ~ 2^n-1
    - first m-n bit = page number = 0 ~ 2^m-1
  - page table translates: page no → frame no (M-n bits)
    - M >= m



Paging Hardware



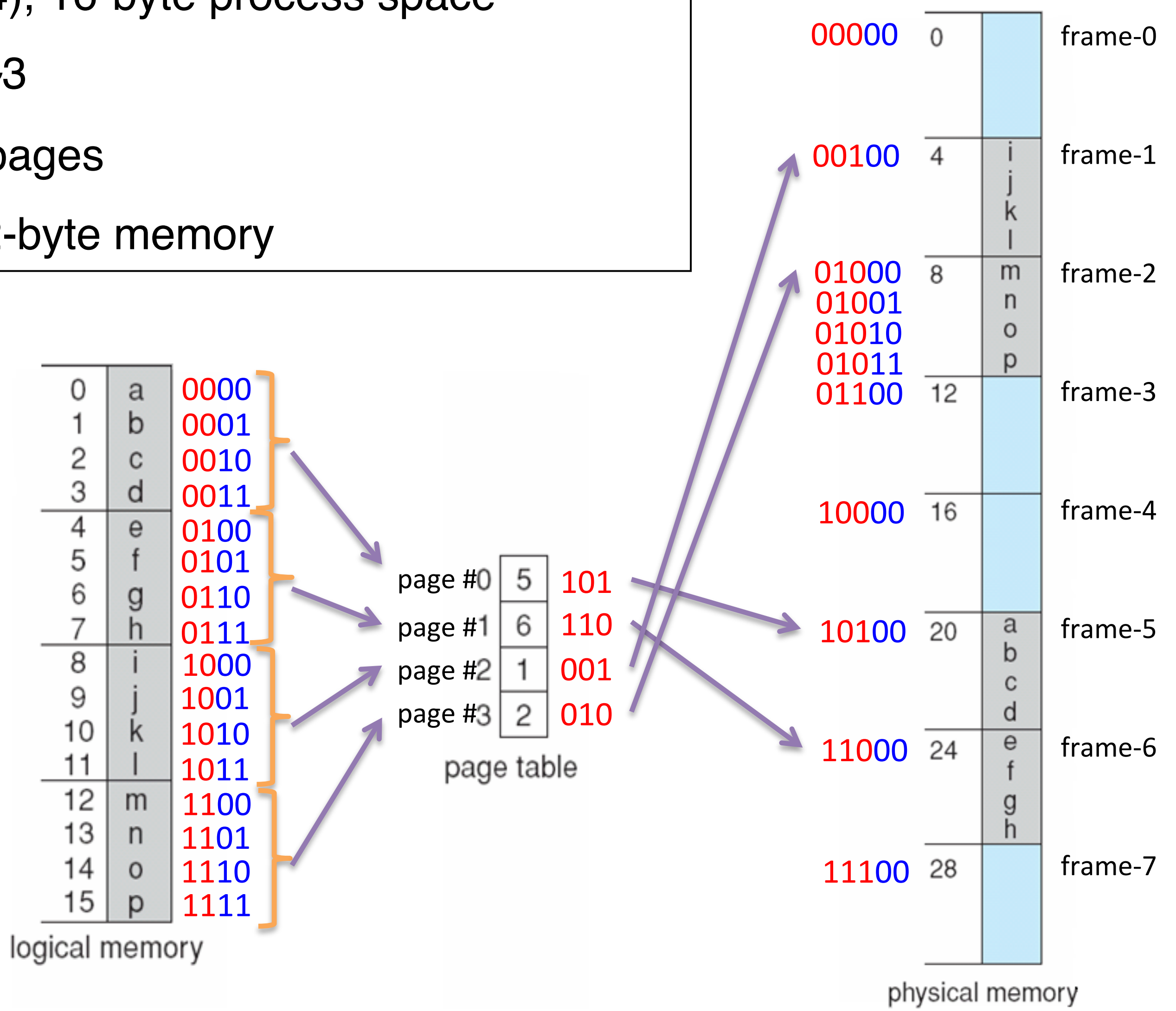
Paging Example

4-bit logical address (m=4), 16-byte process space

2-bit page no (m-n=2), 0~3

2-bit offset (n=2), 4-byte pages

5-bit physical address, 32-byte memory



Paging Example

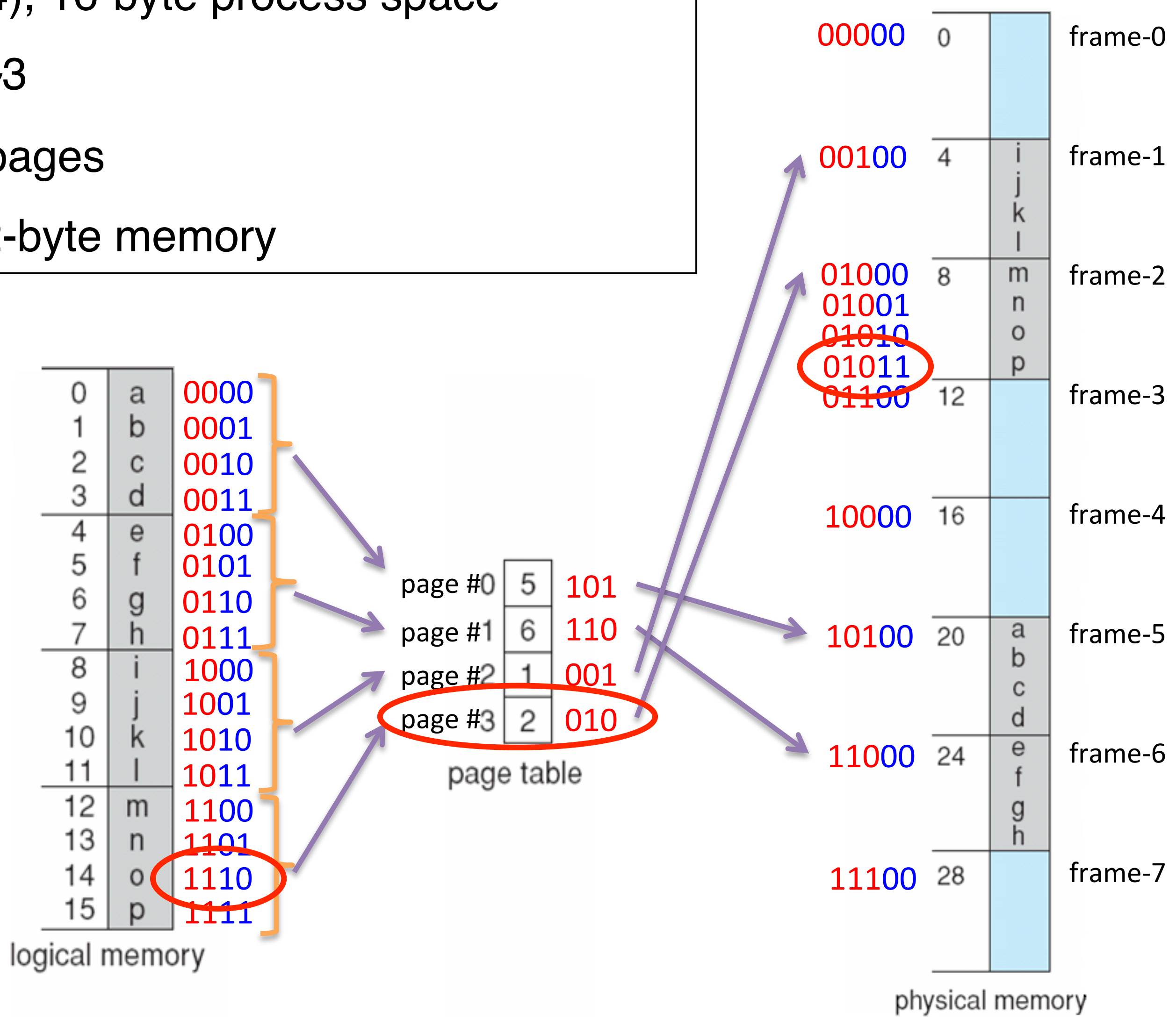
4-bit logical address (m=4), 16-byte process space

2-bit page no (m-n=2), 0~3

2-bit offset (n=2), 4-byte pages

5-bit physical address, 32-byte memory

Logical address 1110
→page 11, offset 10
→frame 010, offset 10
→Phys. addr. 01010



Fragmentation in Paging

- Internal fragmentation
  - Page size = 2,048 bytes
  - Process size = 72,766 bytes
  - 35 pages + 1,086 bytes
  - Internal fragmentation = 2,048 - 1,086 = 962 bytes
- Frame size & fragmentation
  - Internal fragmentation = 1byte ~ (frame size – 1)
  - Average fragmentation = 1 / 2 frame size
  - Small frame size better?

- Small frame size → Small internal fragmentation
  → Large page table
- Large frame size→ Small page table
  → More internal fragmentation

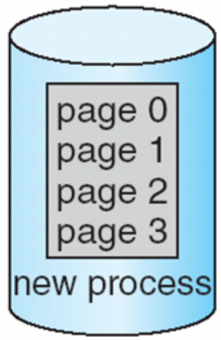
Quiz

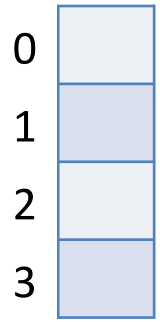
Consider a simple paging system with the following parameters: 2^{32} bytes of physical memory; page size of 2^{10} bytes; 2^{16} pages of logical address space.

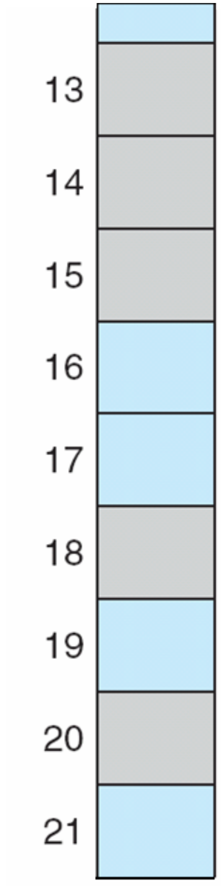
- How many bits are in a logical address?
- How many bytes in a frame?
- How many bits in the physical address specify the frame?
- How many entries in the page table?

Allocation of Free Frames

Free frame list: 14→13→18→20→15→•

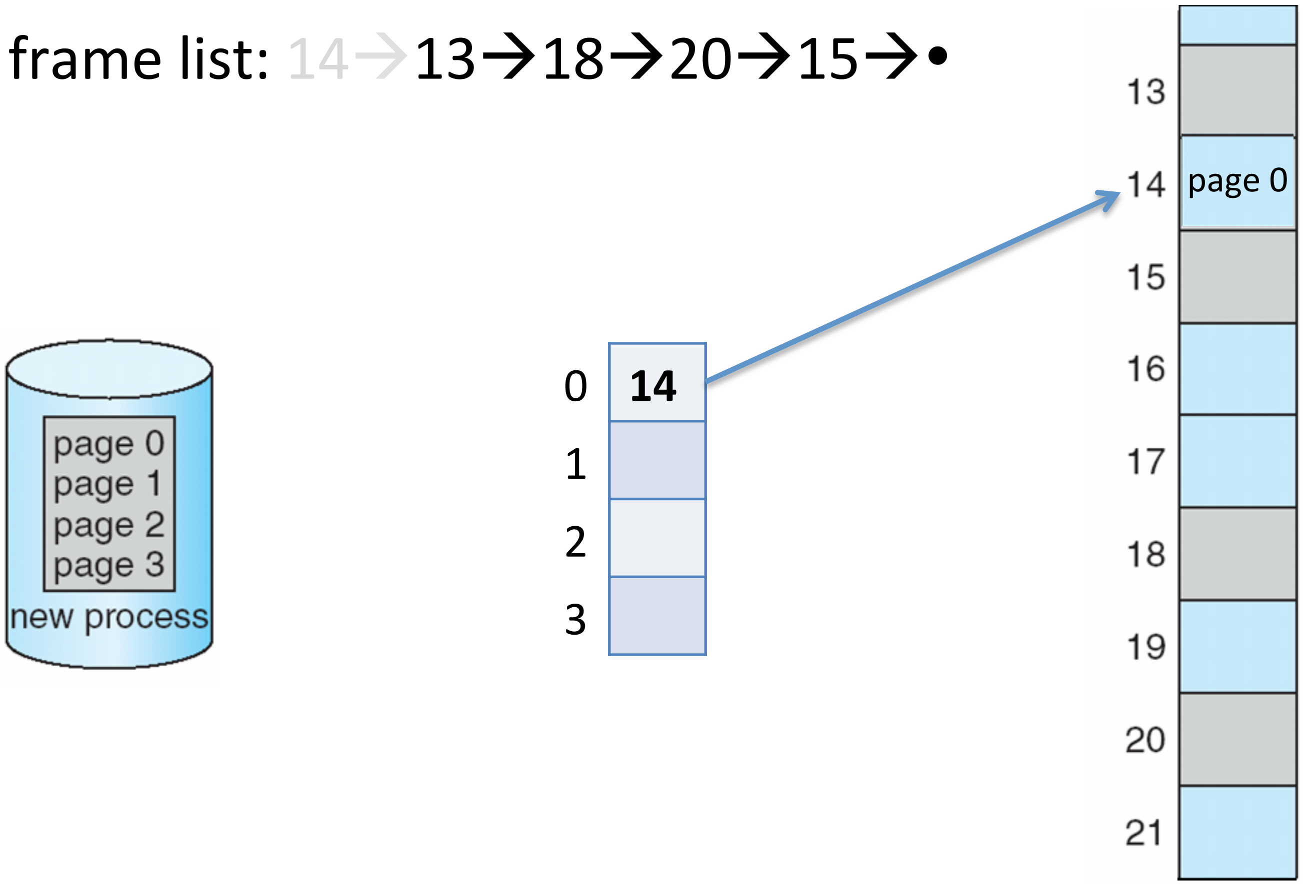






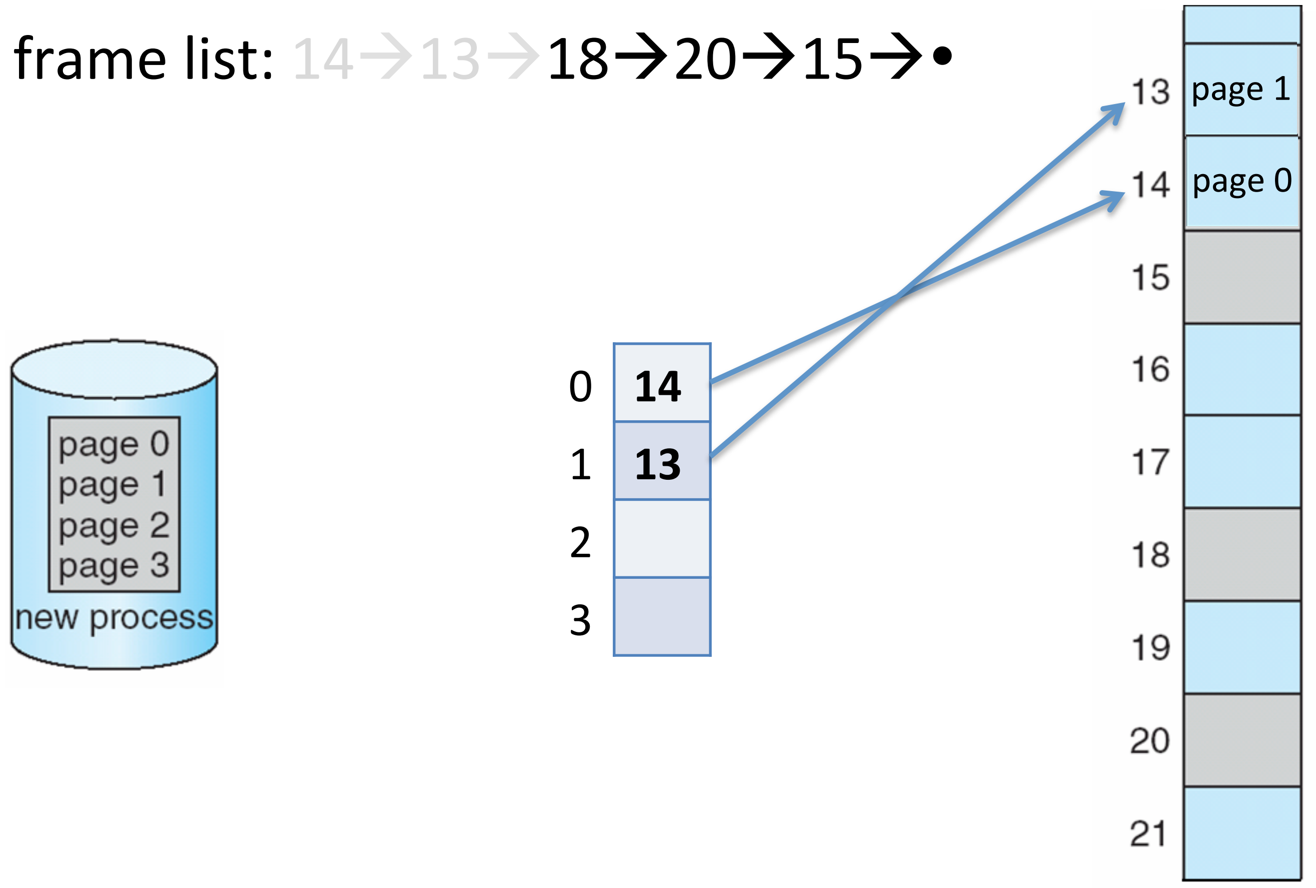
Allocation of Free Frames

Free frame list: 14→13→18→20→15→•



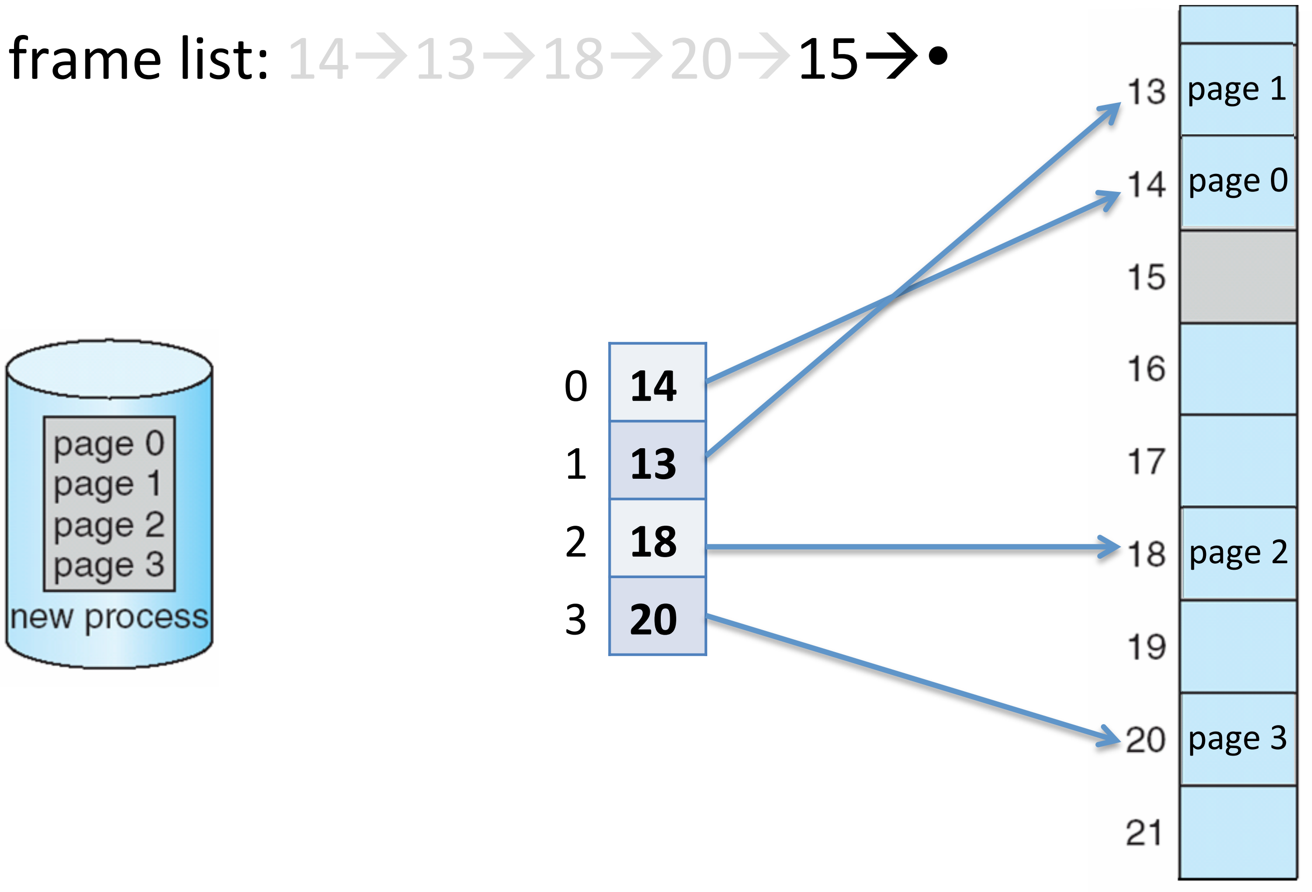
Allocation of Free Frames

Free frame list: 14→13→18→20→15→•



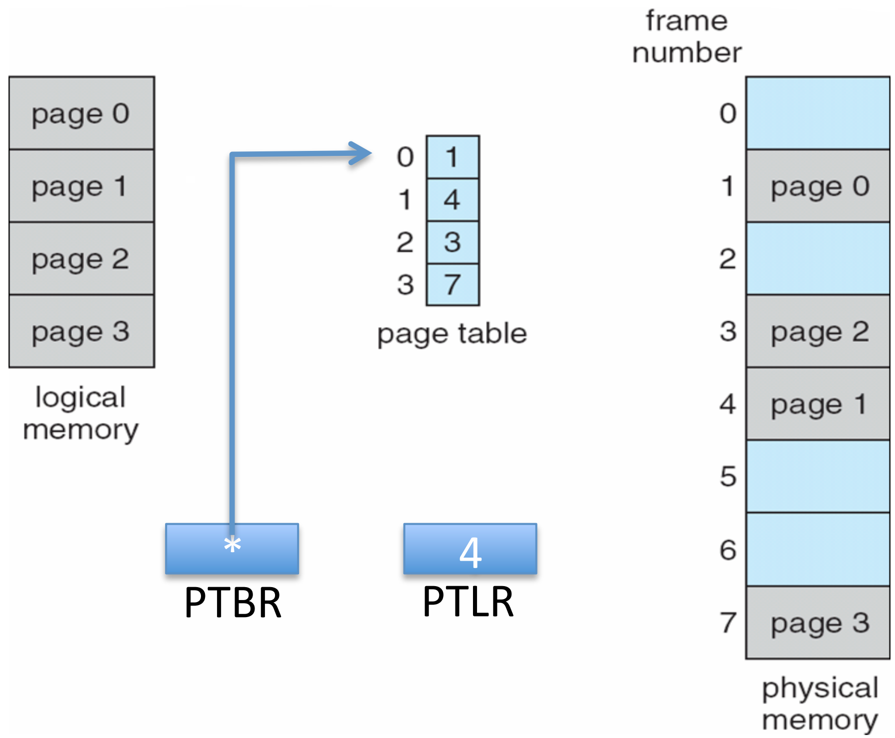
Allocation of Free Frames

Free frame list: 14→13→18→20→15→•



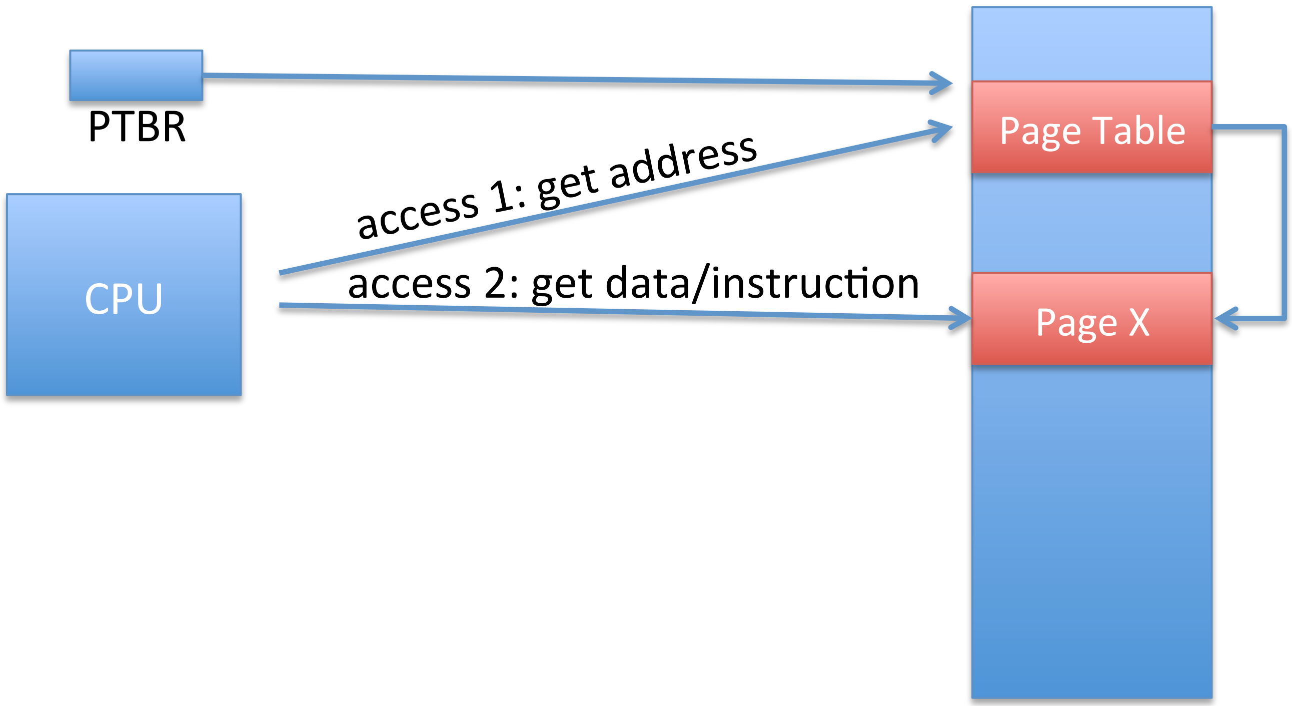
Implementation of Page Table

- 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



Memory Access with Paging

- With paging, every data/instruction access requires
  - 2 memory accesses
  - One for the page table and one for the data / instruction

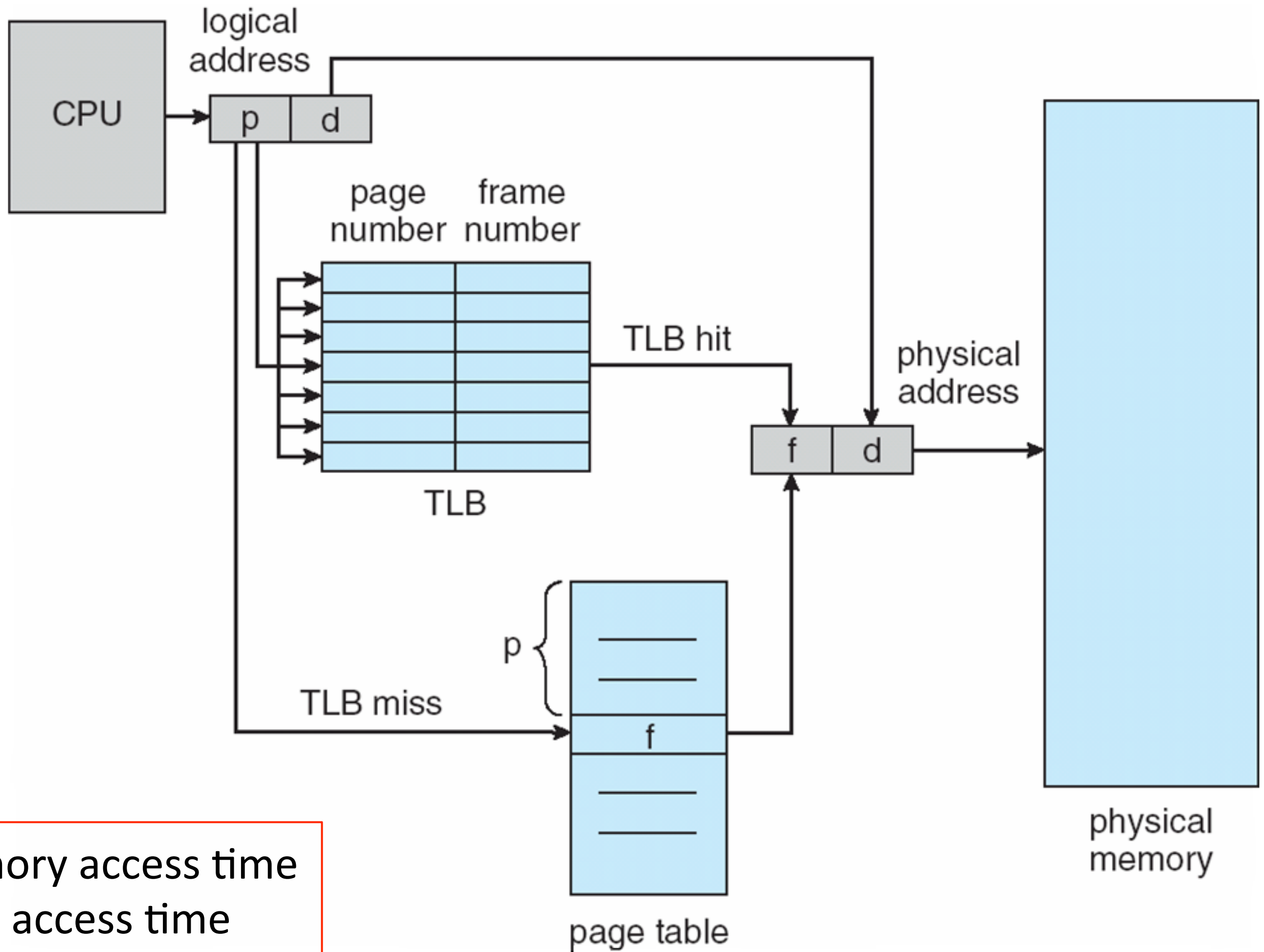


Memory Access with Paging

- Solution: **translation look-aside buffer**
  - a special fast-lookup hardware cache
  - **associative memory**
- **address-space identifiers (ASIDs)**
  - distinguish between entries of different processes
  - Otherwise need to flush at every context switch
- TLBs typically small (64 to 1,024 entries)

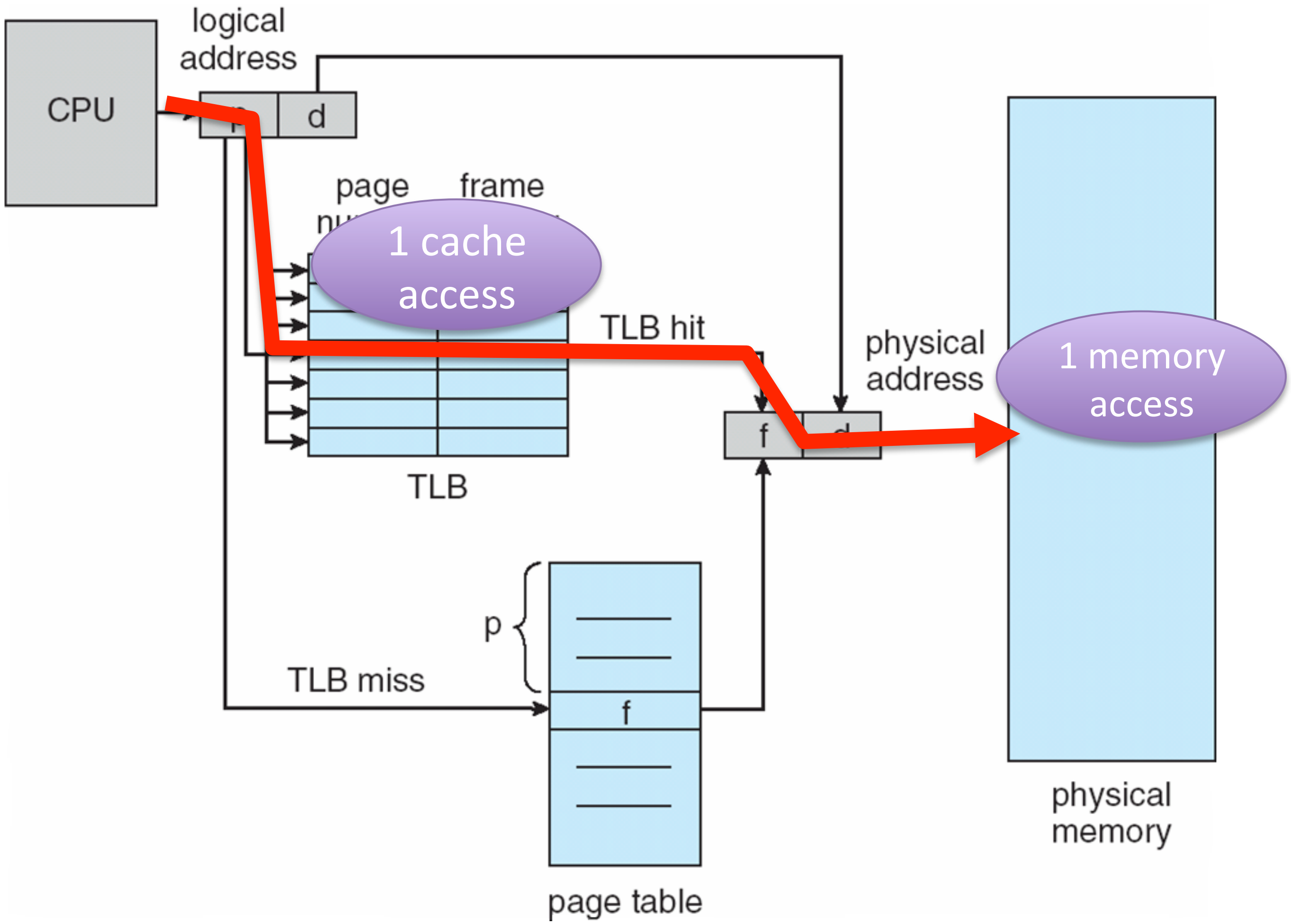
- Operation
  - Works like a cache
  - Replacement policies must be considered
  - Some entries can be **wired down** for permanent fast access

Paging With TLB



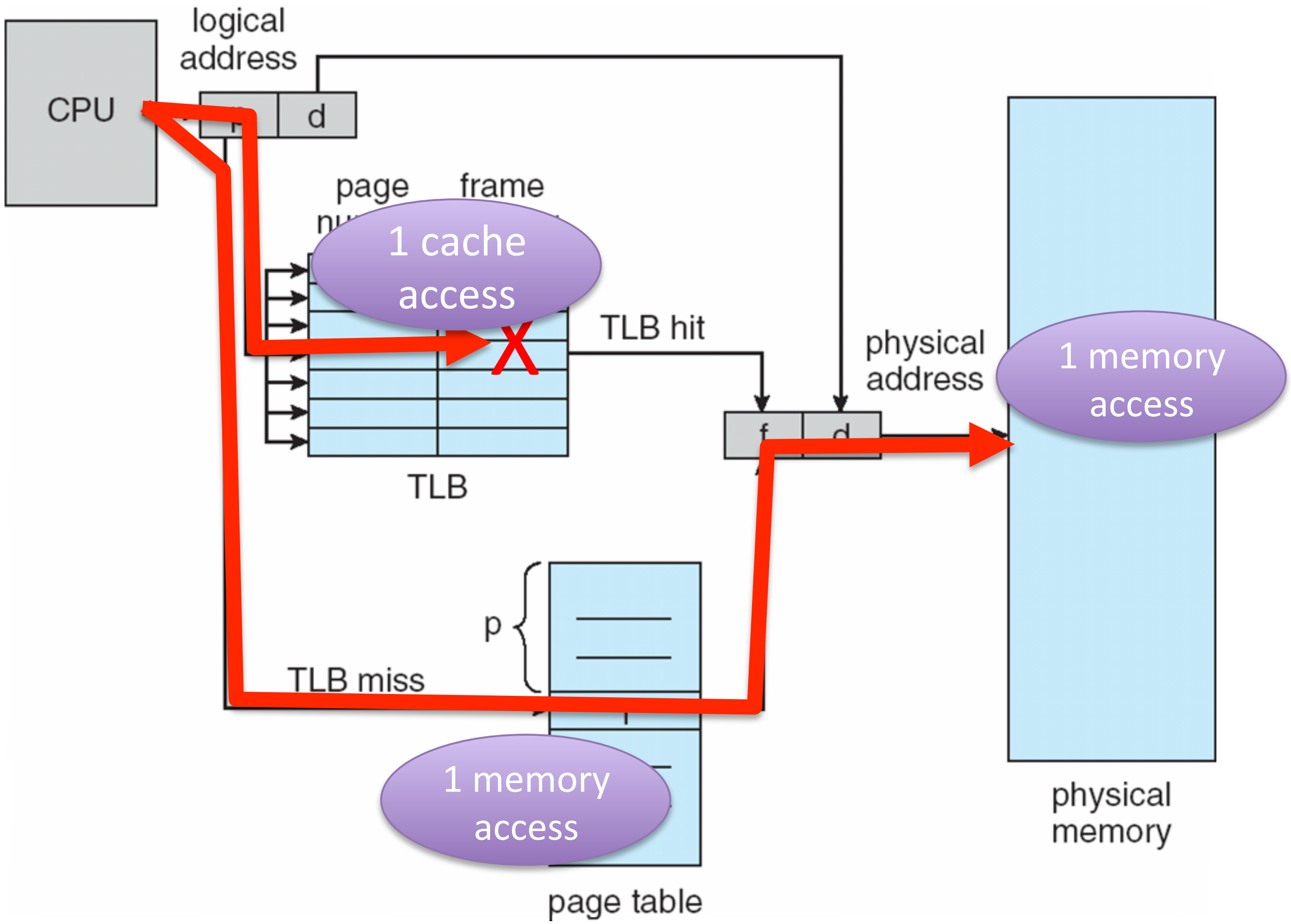
M: memory access time
e: cache access time
e << M

Paging With TLB-hit



Access time = e + M

Paging With TLB-miss



Access time = e + 2M

Effective Access Time

- Memory lookup = M time unit
- Associative Lookup = e time unit
  - Can be < 10% of memory access time
- Hit ratio = α
  - $0 < \alpha < 1$
- **Effective Access Time (EAT)**

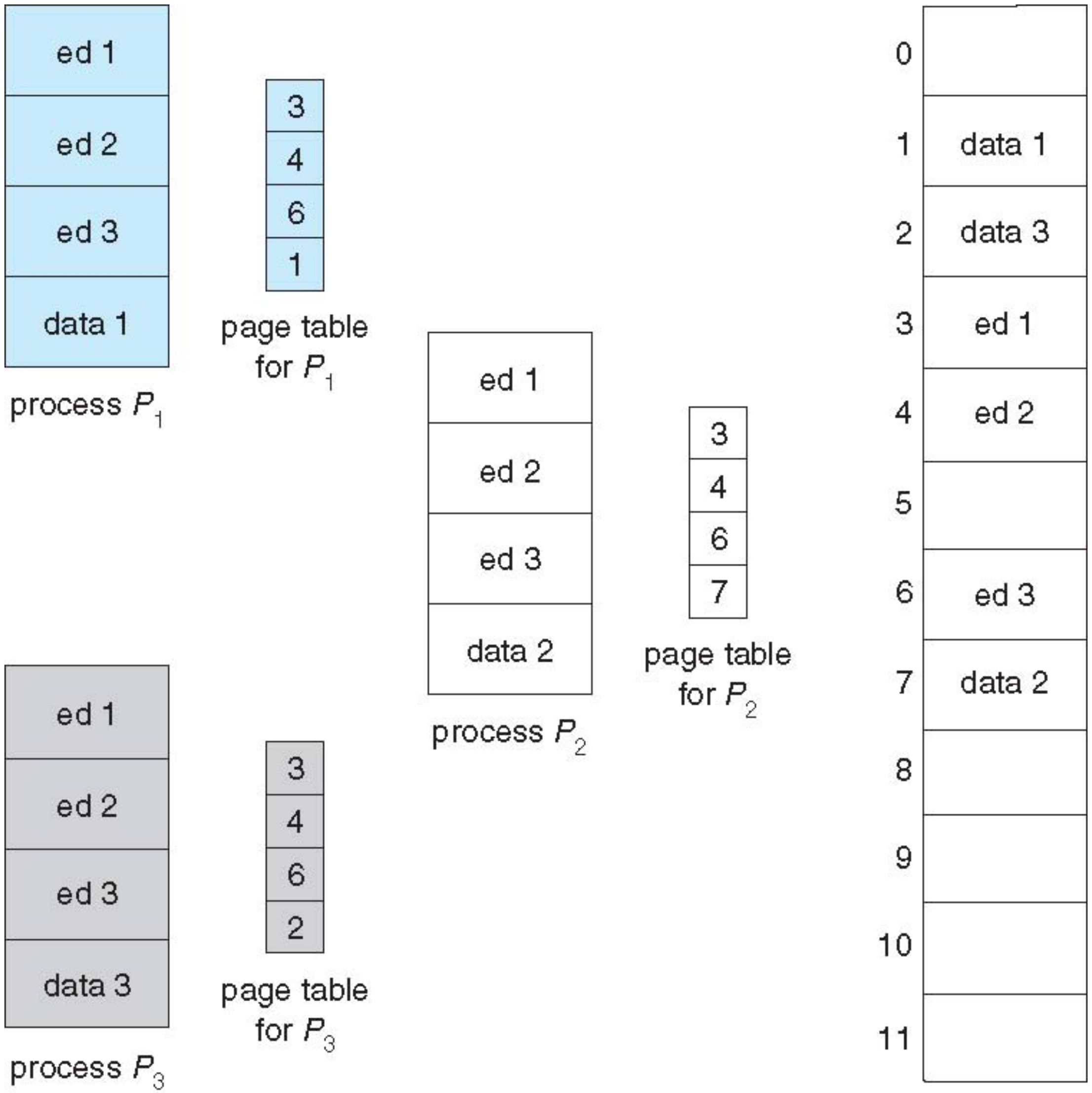
$$EAT = (M + e)\,\alpha + (2M + e)(1 - \alpha)$$
$$= 2M + e - \alpha$$

- Consider $\alpha = 80\%$, $e = 20$ ns, M=100 ns
  - EAT = 0.80 x 120 + 0.20 x 220 = 140ns

Shared Pages

- **Shared code**
  - One copy of read-only (**reentrant**) code shared among processes (i.e., text editors, compilers, window systems)
  - Also useful for interprocess communication if sharing of read-write pages is allowed

Shared Pages Example



Shared Pages Example