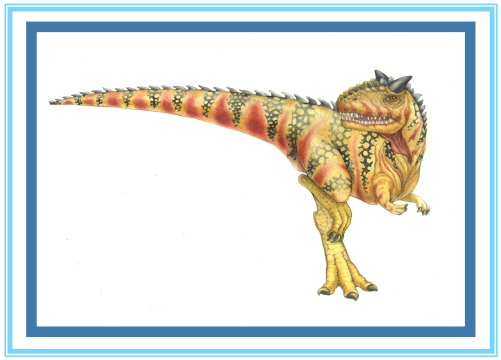
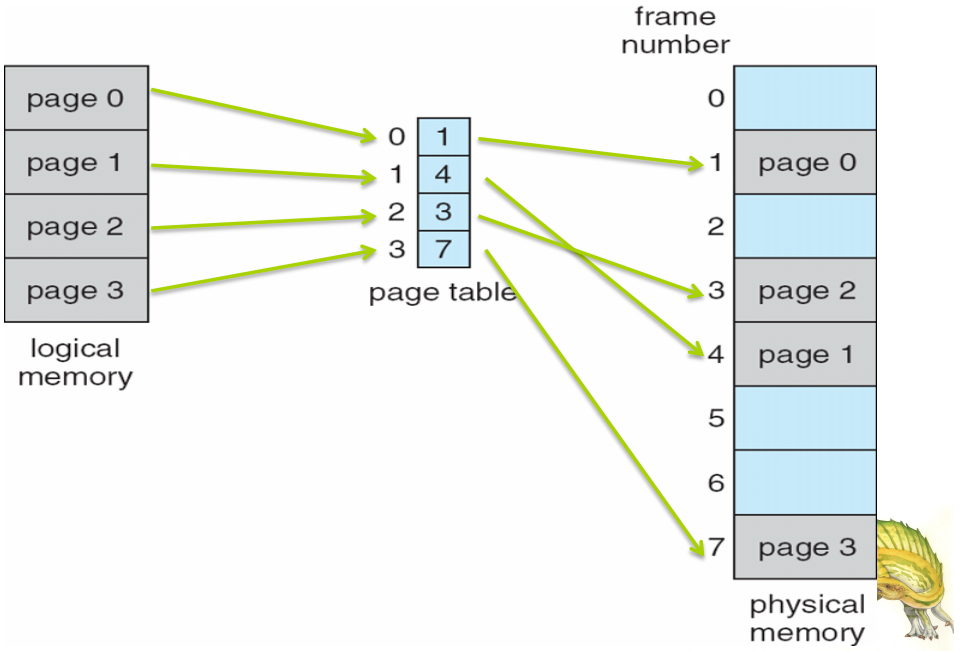
Chapter 8: Main Memory-part2

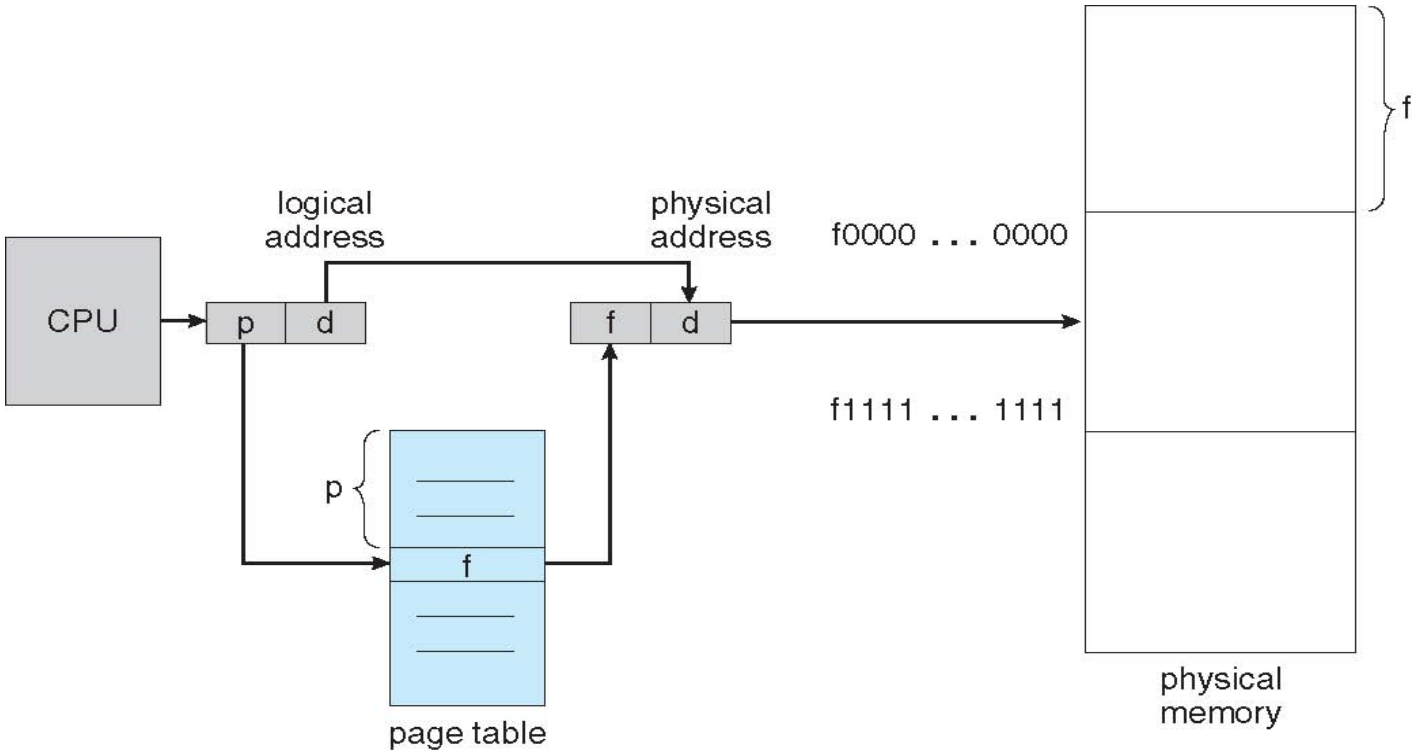


Paging

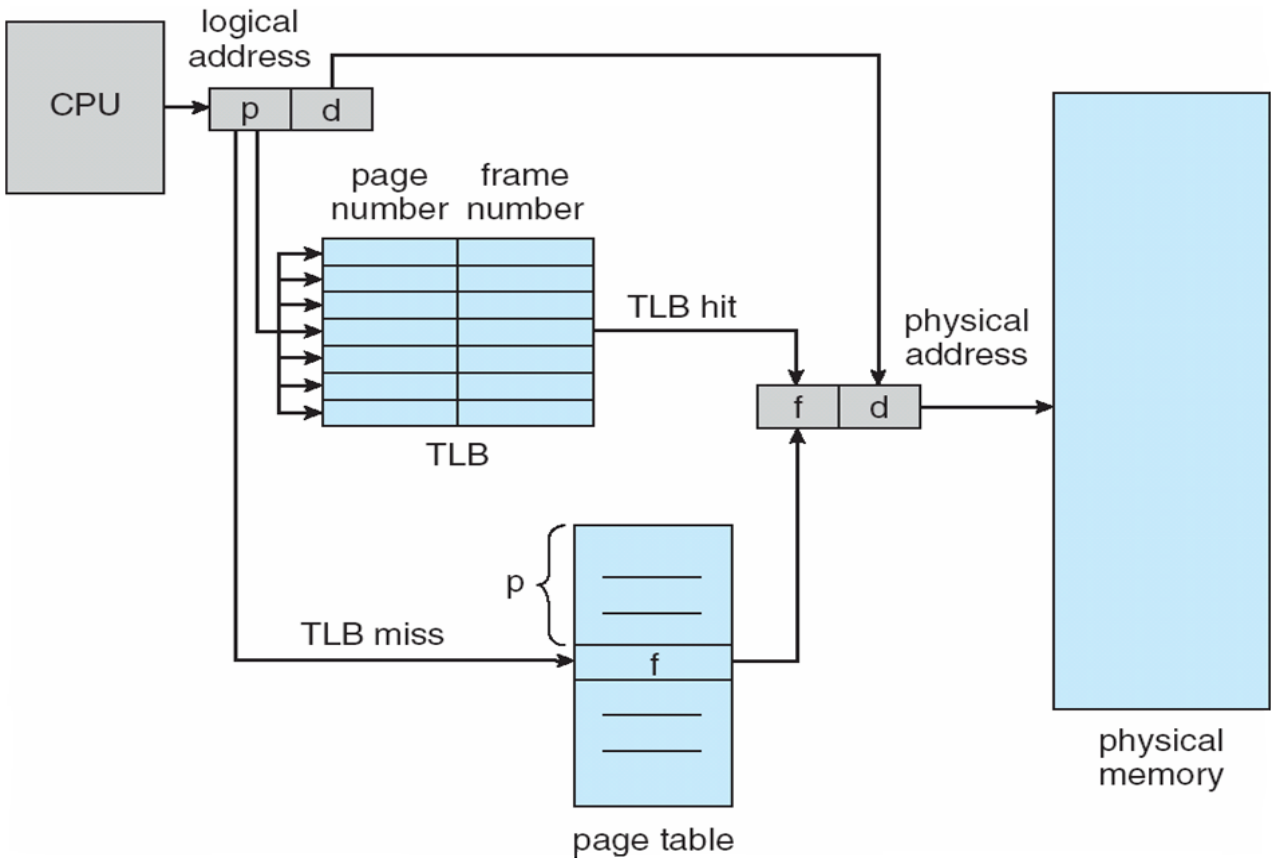
- Partition physical memory into equal size frames
- Divide logical memory into same-size pages
- Each page can go to any free frame
- OS knows the mapping
  - page table



Addressing with Paging



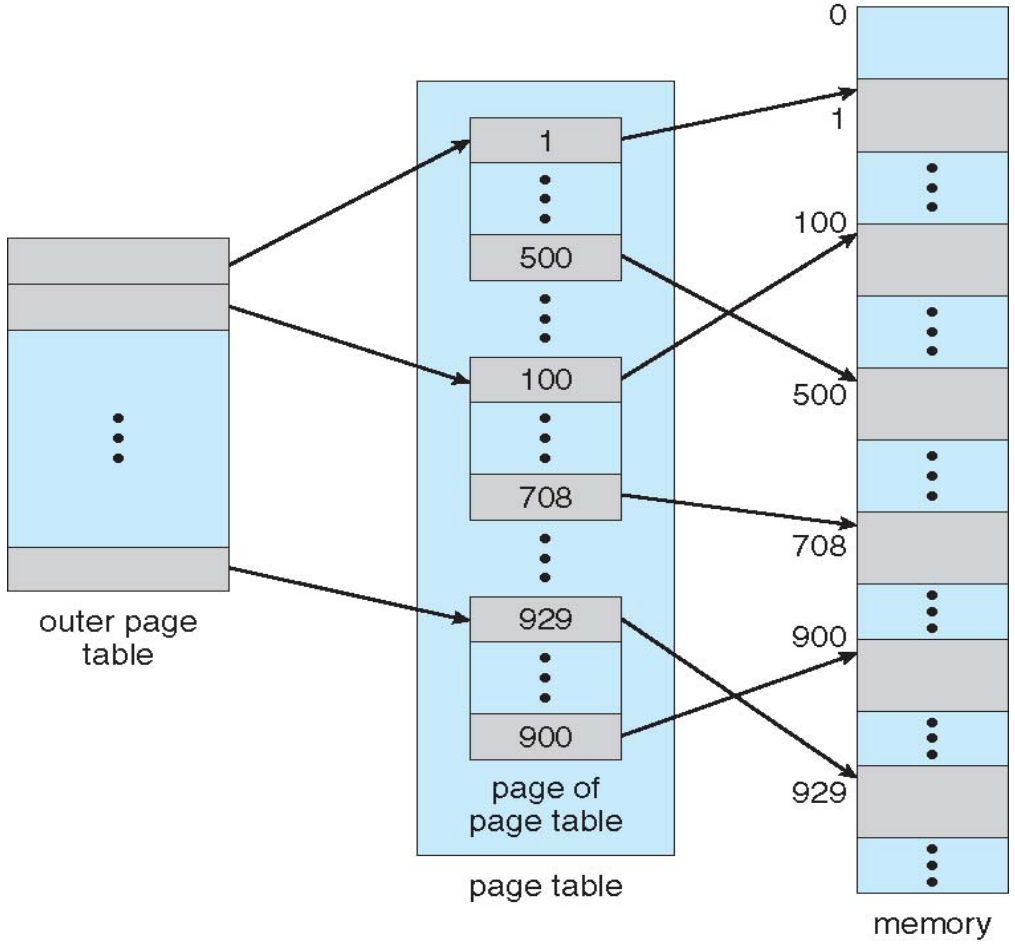
Paging With TLB



Structure of the Page Table

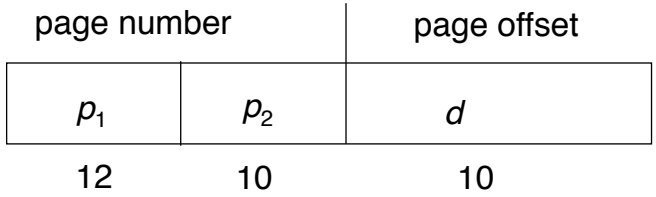
- Memory structures for paging can get huge using straight-forward methods
  - Consider a 32-bit logical address space as on modern computers
  - Page size of 4 KB (2^{12})
  - Page table would have 1 million entries ($2^{32} / 2^{12}$)
  - If each entry is 4 bytes -> 4 MB of physical address space / memory for page table alone
    - That amount of memory used to cost a lot
    - Don't want to allocate that contiguously in main memory
- Hierarchical Paging
- Hashed Page Tables
- Inverted Page Tables

Two-Level Page-Table Scheme



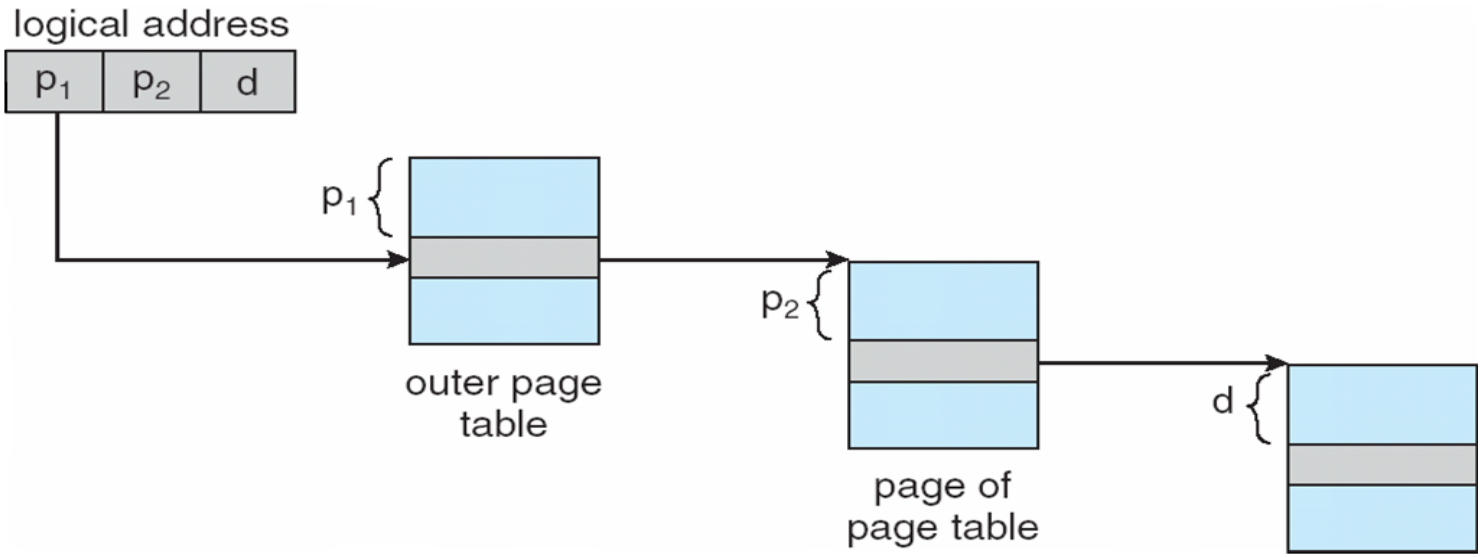
Two-Level Paging Example

- 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_1 is an index into the outer page table, and p_2 is the displacement within the page of the inner page table
- Known as **forward-mapped page table**

Address-Translation Scheme



64-bit Logical Address Space

- Even two-level paging scheme not sufficient
- If page size is 4 KB (2^{12})
  - Then page table has 2^{52} entries
  - If two level scheme, inner page tables could be 2^{10} 4-byte entries
  - Address would look like

outer page	inner page	page offset
p_1	p_2	d
42	10	12

  - Outer page table has 2^{42} entries or 2^{44} bytes
  - One solution is to add a 2^{nd} outer page table
  - But in the following example the 2^{nd} outer page table is still 2^{34} bytes in size
    - And possibly 4 memory access to get to one physical memory location

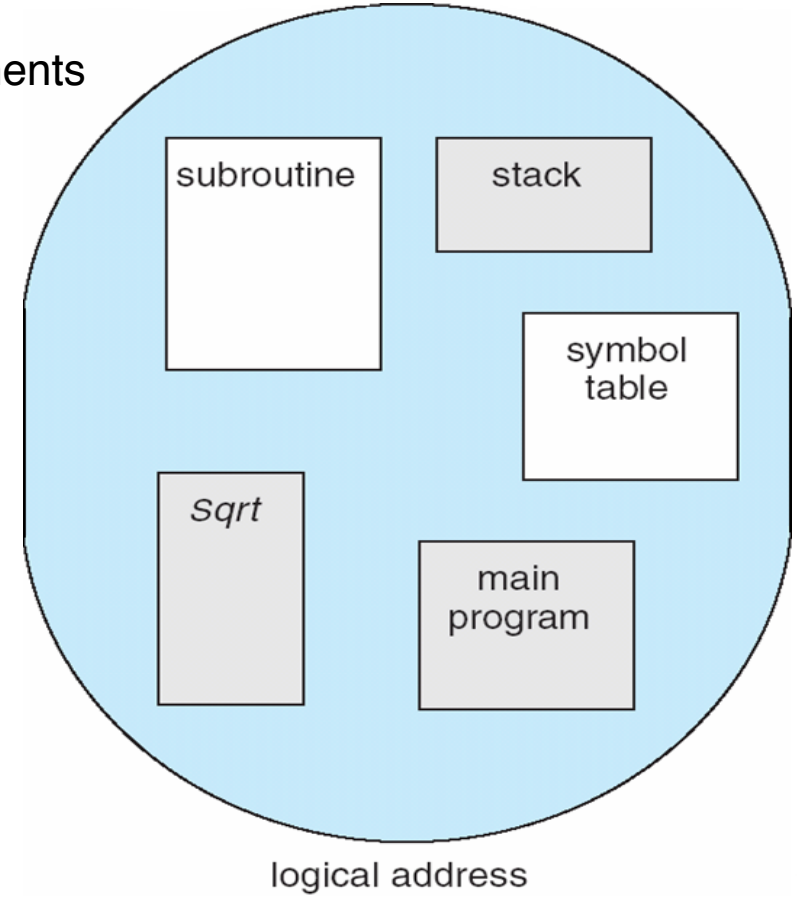
Three-level Paging Scheme

outer page	inner page	offset
p_1	p_2	d
42	10	12

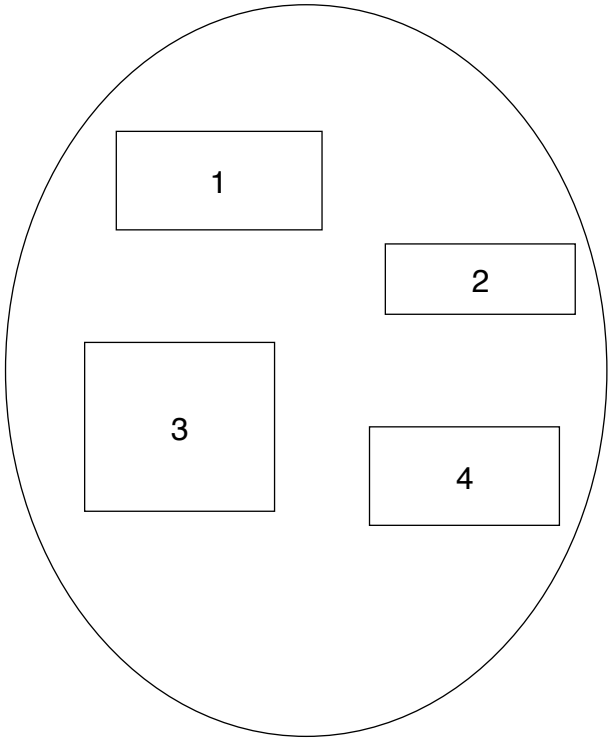
2nd outer page	outer page	inner page	offset
p_1	p_2	p_3	d
32	10	10	12

User's View of a Program

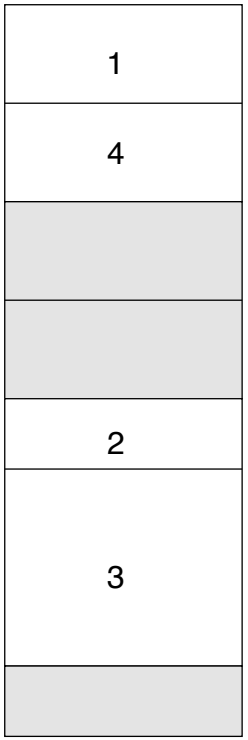
- A program is a collection of segments



Logical View of Segmentation

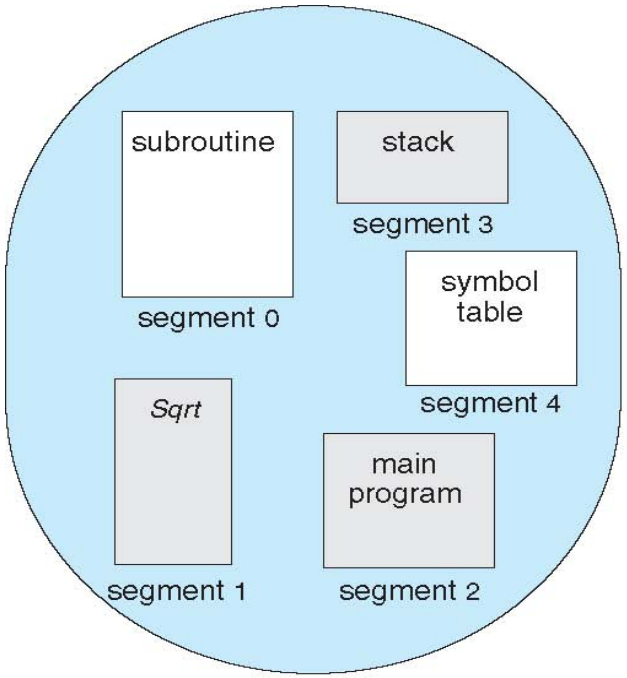


user space



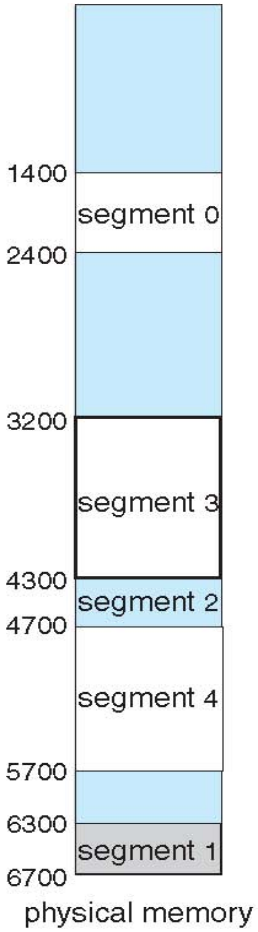
physical memory space

Example of Segmentation

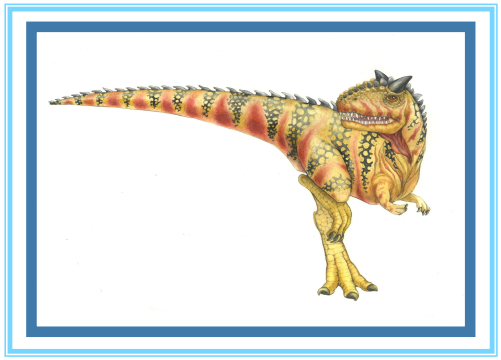


	limit	base
0	1000	1400
1	400	6300
2	400	4300
3	1100	3200
4	1000	4700

segment table



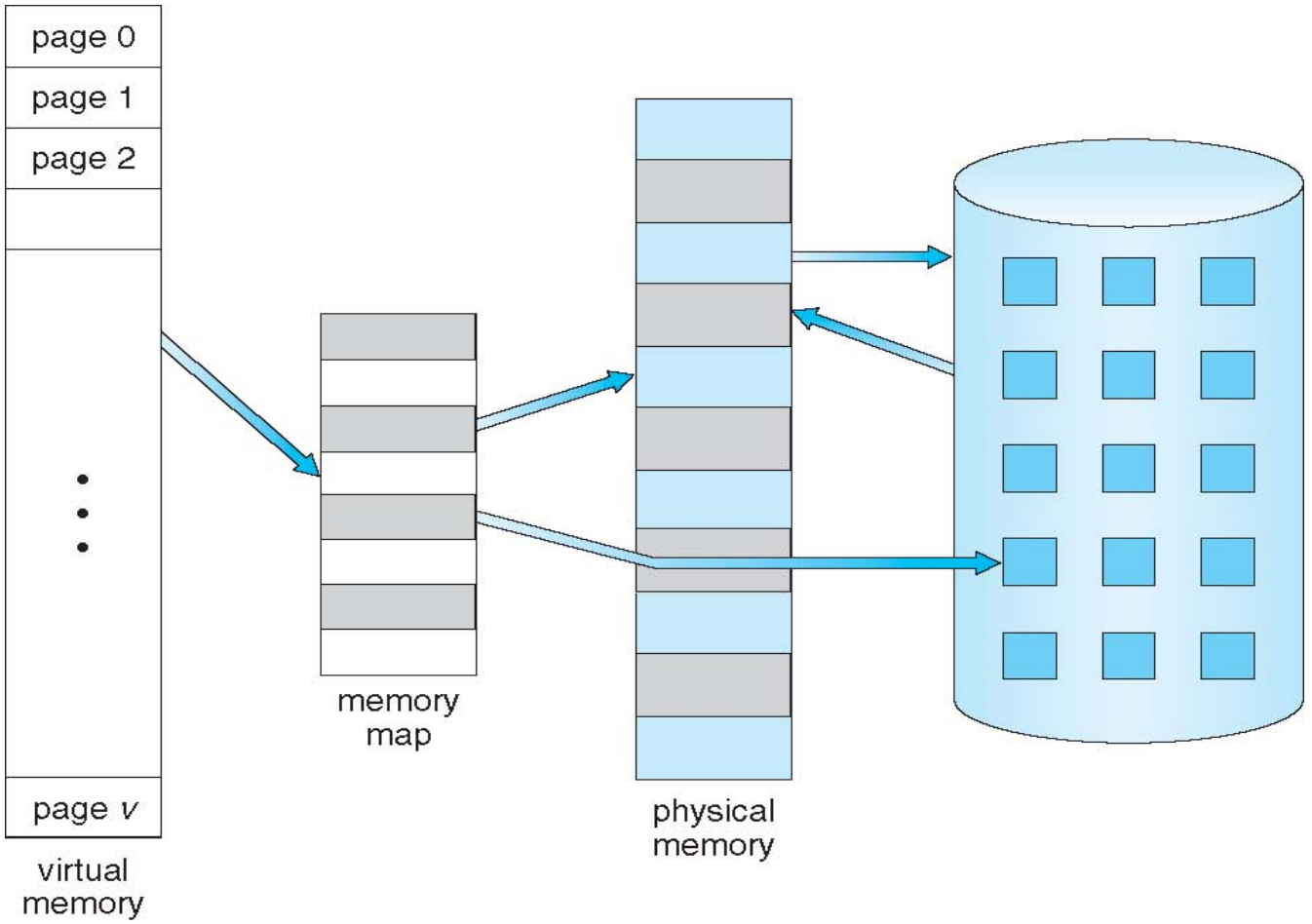
Chapter 9: Virtual Memory



Background

- **Virtual memory** – separation of user logical memory from physical memory
  - Only part of the program needs to be in memory for execution
  - Logical address space can therefore be much larger than physical address space
- Virtual memory can be implemented via:
  - Demand paging
  - Demand segmentation

Virtual Memory That is Larger Than Physical Memory



Demand Paging

- Bring a page into memory only when it is needed
  - Less I/O needed, no unnecessary I/O
  - Less memory needed
  - Faster response
  - More users

Valid-Invalid Bit

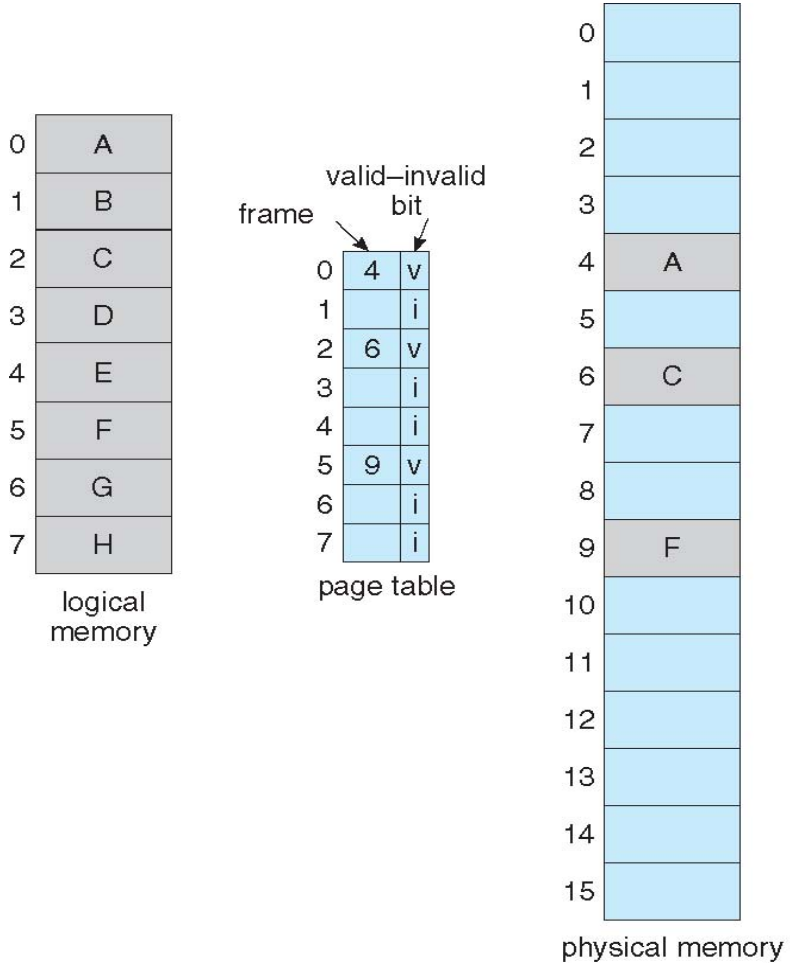
- With each page table entry a valid–invalid bit is associated (**v** ⇒ in-memory – **memory resident**, **i** ⇒ not-in-memory)
- Example of a page table snapshot:

Frame #	valid-invalid bit
	v
	v
	v
	v
	i
….	
	i
	i

page table

- During address translation, if valid–invalid bit in page table entry is **I** ⇒ page fault

Page Table When Some Pages Are Not in Main Memory



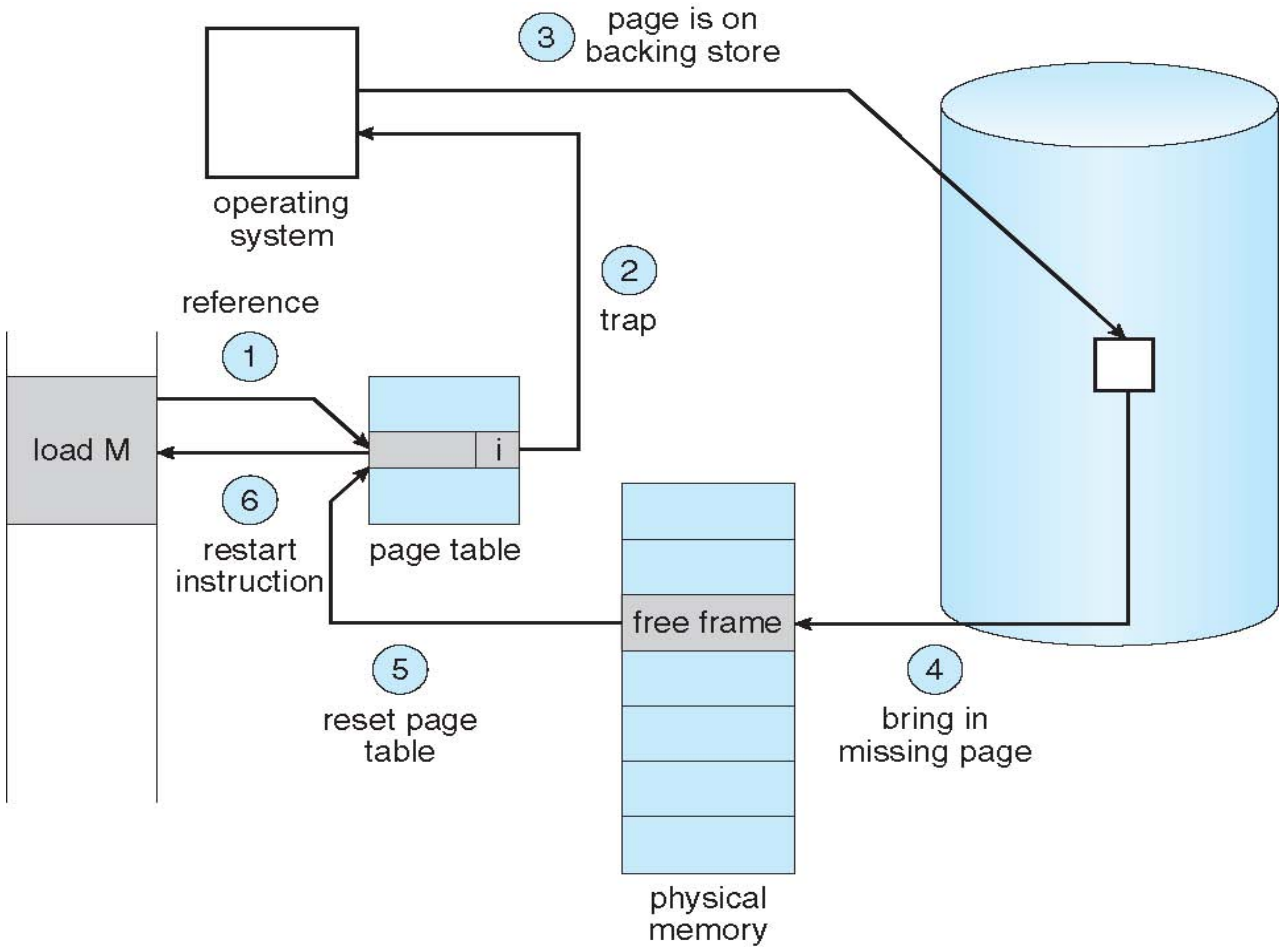
Page Fault

- If there is a reference to a page, first reference to that page will trap to operating system:

  page fault

1. Operating system looks at another table to decide:
   - Invalid reference ⇒ abort
   - Just not in memory
2. Get empty frame
3. Swap page into frame via scheduled disk operation
4. Reset tables to indicate page now in memory
   Set validation bit = **v**
5. Restart the instruction that caused the page fault

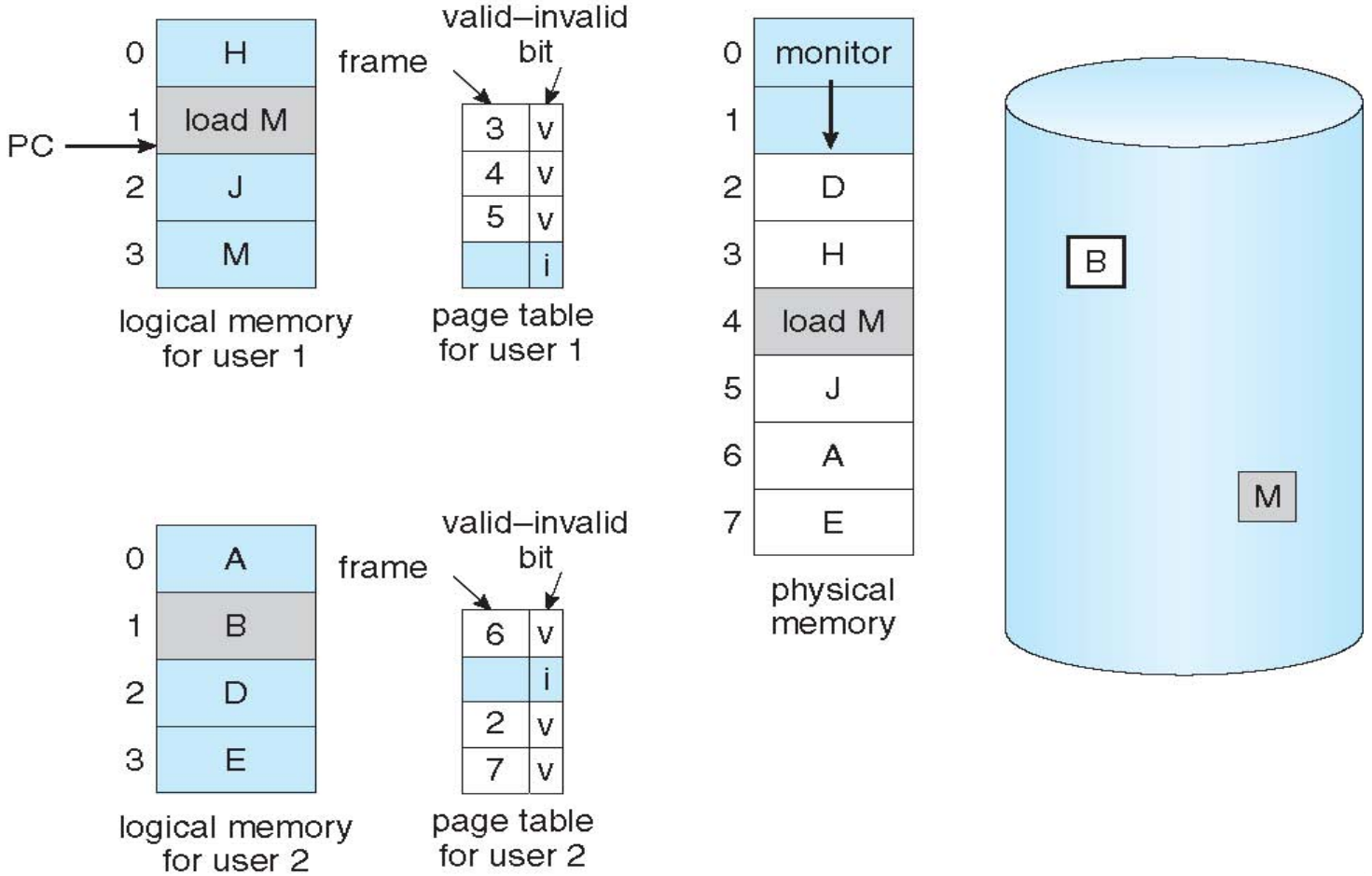
Steps in Handling a Page Fault



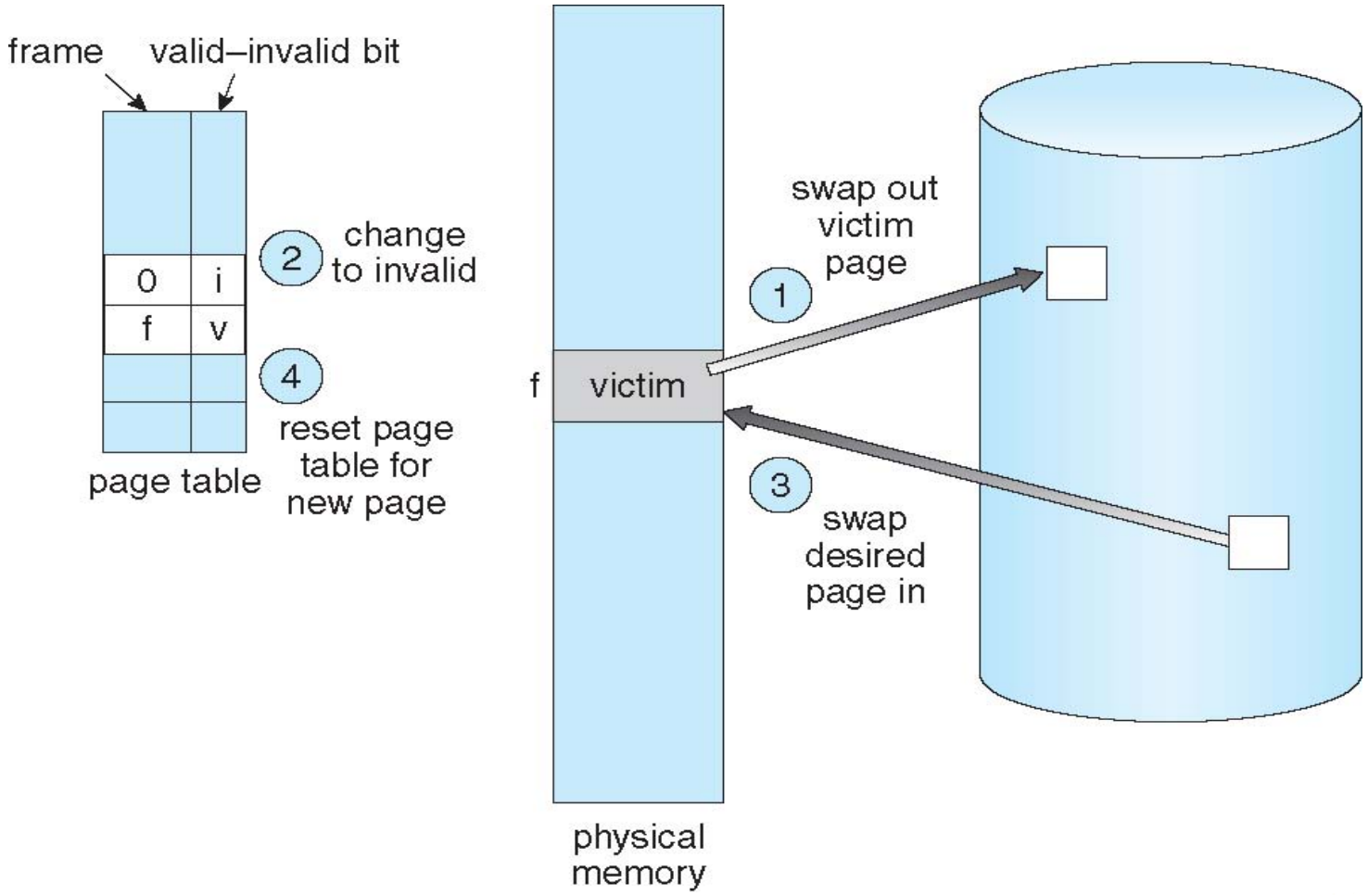
What Happens if There is no Free Frame?

- Page replacement
  - Find some page in memory, but not really in use, page it out
  - Performance – want an algorithm which will result in *minimum number of page faults*

Need For Page Replacement



Page Replacement



Page Replacement Algorithms

- **Page-replacement algorithm**
  - Want lowest page-fault rate on both first access and re-access
- **Optimal**
- **FIFO (First In First Out)**
- **Least Recently Used (LRU)**

Optimal Algorithm

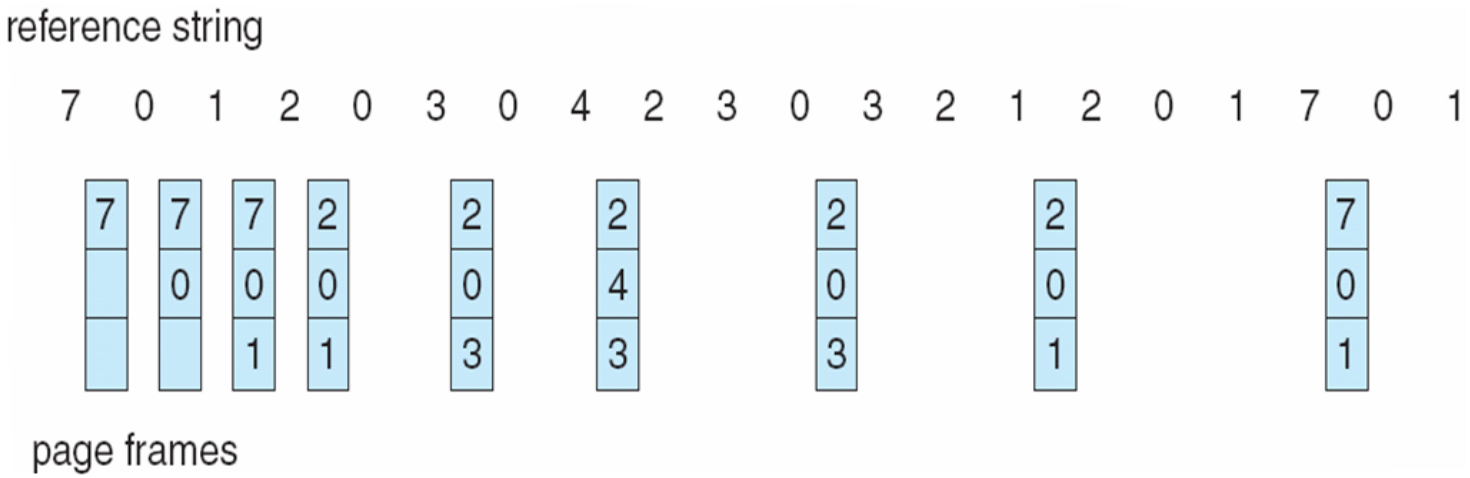
- Replace page that will not be used for longest period of time
- How do you know this?
  - Can't read the future
- Used for measuring how well your algorithm performs

Optimal Page Replacement

reference string

7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1

Optimal Page Replacement



FIFO Algorithm

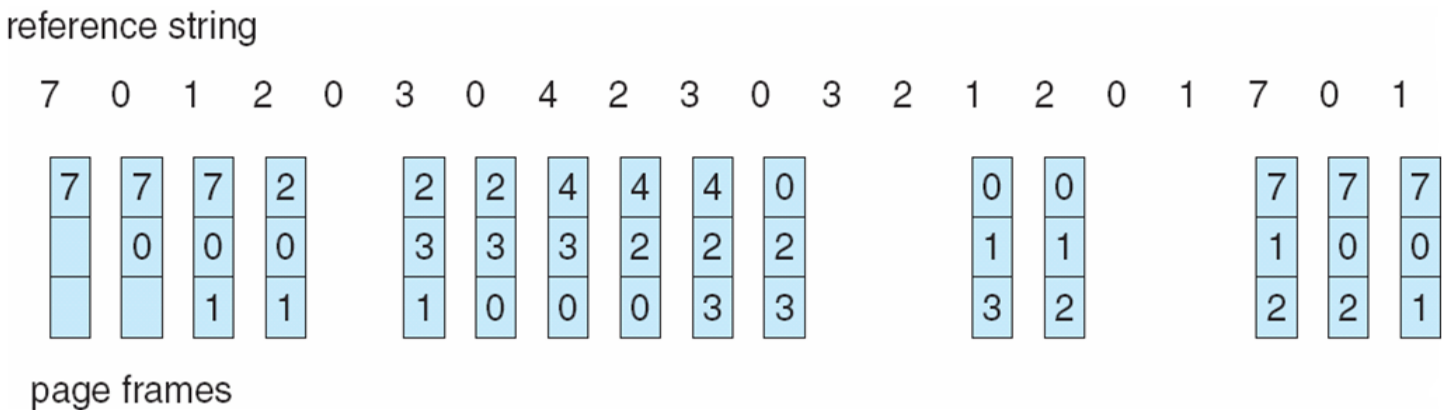
- Replace page that is oldest
- How do you know this?
  - FIFO queue

FIFO Page Replacement

reference string

7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1

FIFO Page Replacement



Least Recently Used (LRU) Algorithm

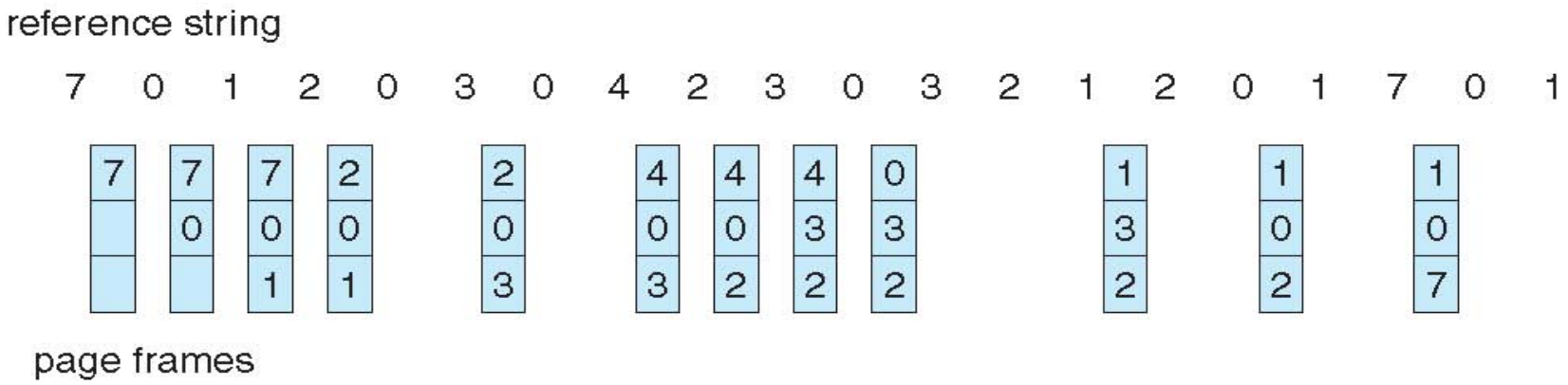
- Use past knowledge rather than future
- Replace page that has not been used in the most amount of time
- Associate time of last use with each page
- Generally good algorithm and frequently used
- But how to implement?

LRU Page Replacement

reference string

7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1

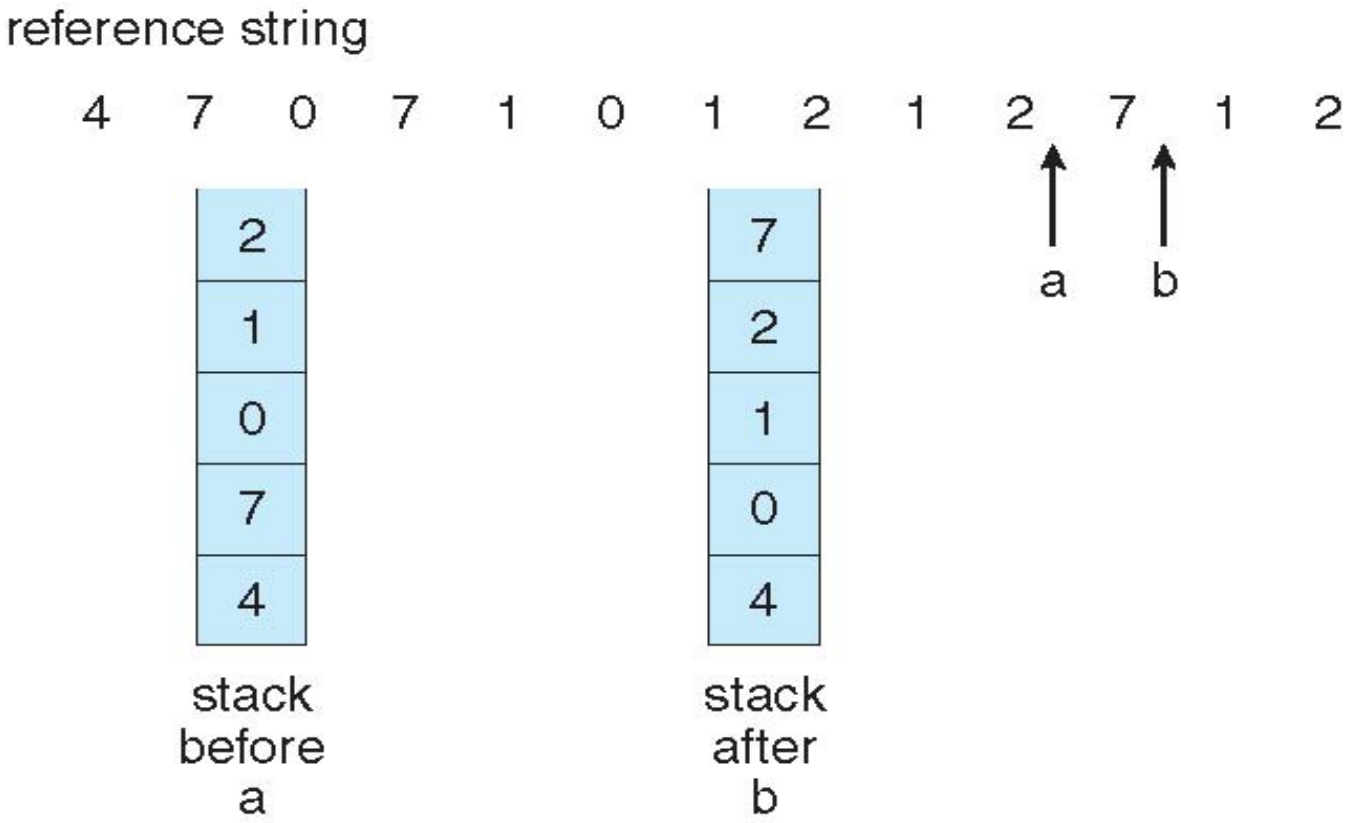
Least Recently Used (LRU) Algorithm



LRU Algorithm (Cont.)

- Counter implementation
  - Every page entry has a counter; every time page is referenced through this entry, copy the clock into the counter
  - When a page needs to be changed, look at the counters to find smallest value
    - Search through table needed
- Stack implementation
  - Keep a stack of page numbers in a double link form:
  - Page referenced:
    - move it to the top
    - requires 6 pointers to be changed
  - But each update more expensive
  - No search for replacement

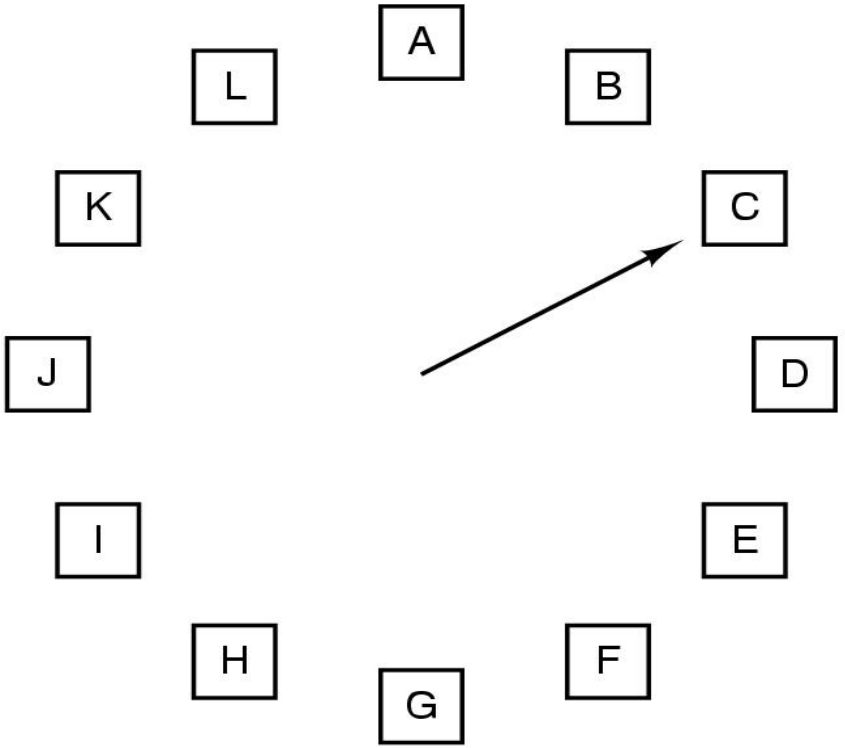
Use Of A Stack to Record The Most Recent Page References



Clock Policy

- Uses and additional bit called a “use bit”
- When a page is first loaded in memory or referenced, the use bit is set to 1
- When it is time to replace a page, the OS scans the set flipping all 1’s to 0
- The first frame encountered with the use bit already set to 0 is replaced.
- When a page is revisited, set to 1

Clock Policy



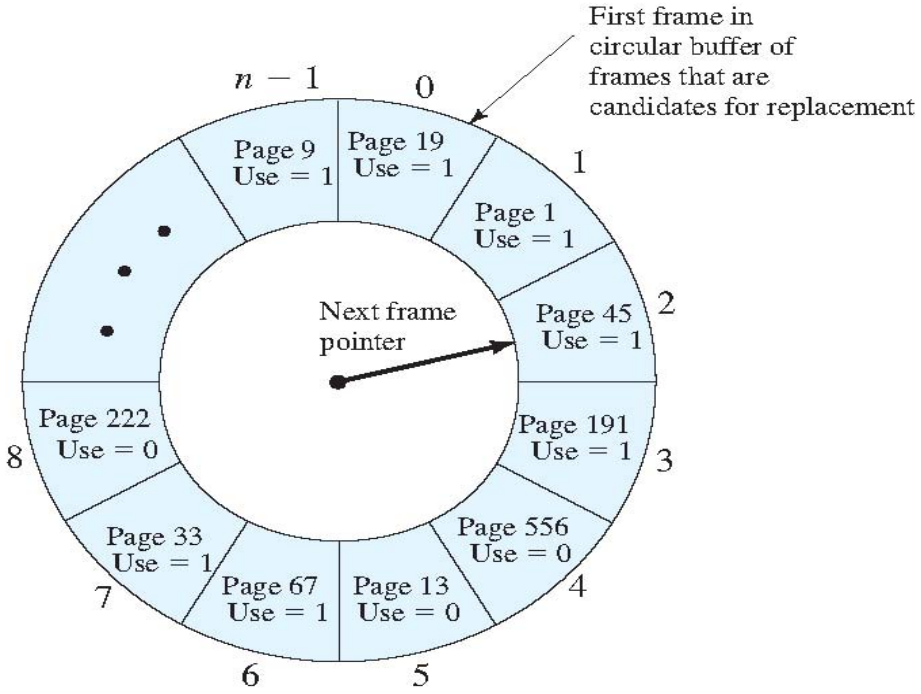
When a page fault occurs, the page the hand is pointing to is inspected. The action taken depends on the R bit:

R = 0: Evict the page

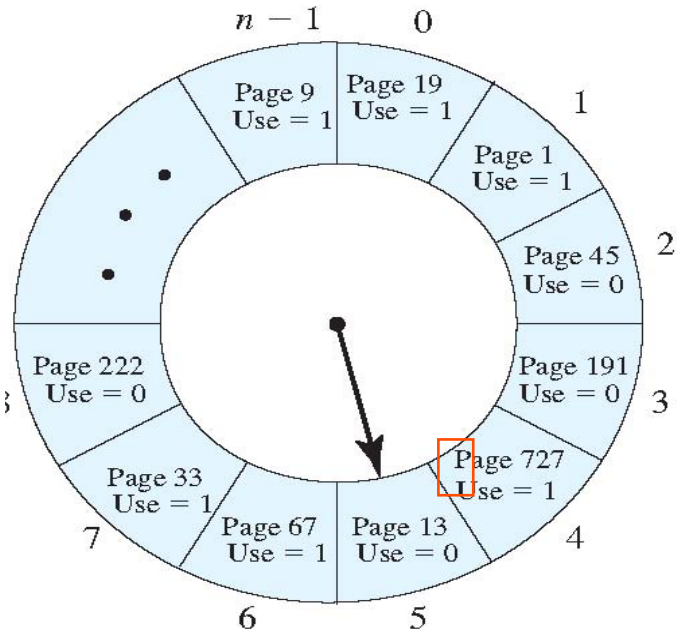
R = 1: Clear R and advance hand

Algorithmically identical with another algorithm called "2nd chance". Only the implementation is different.

Clock Algorithm: Example



(a) State of buffer just prior to a page replacement



) State of buffer just after the next page replacement

Clock Page Replacement

reference string

7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1