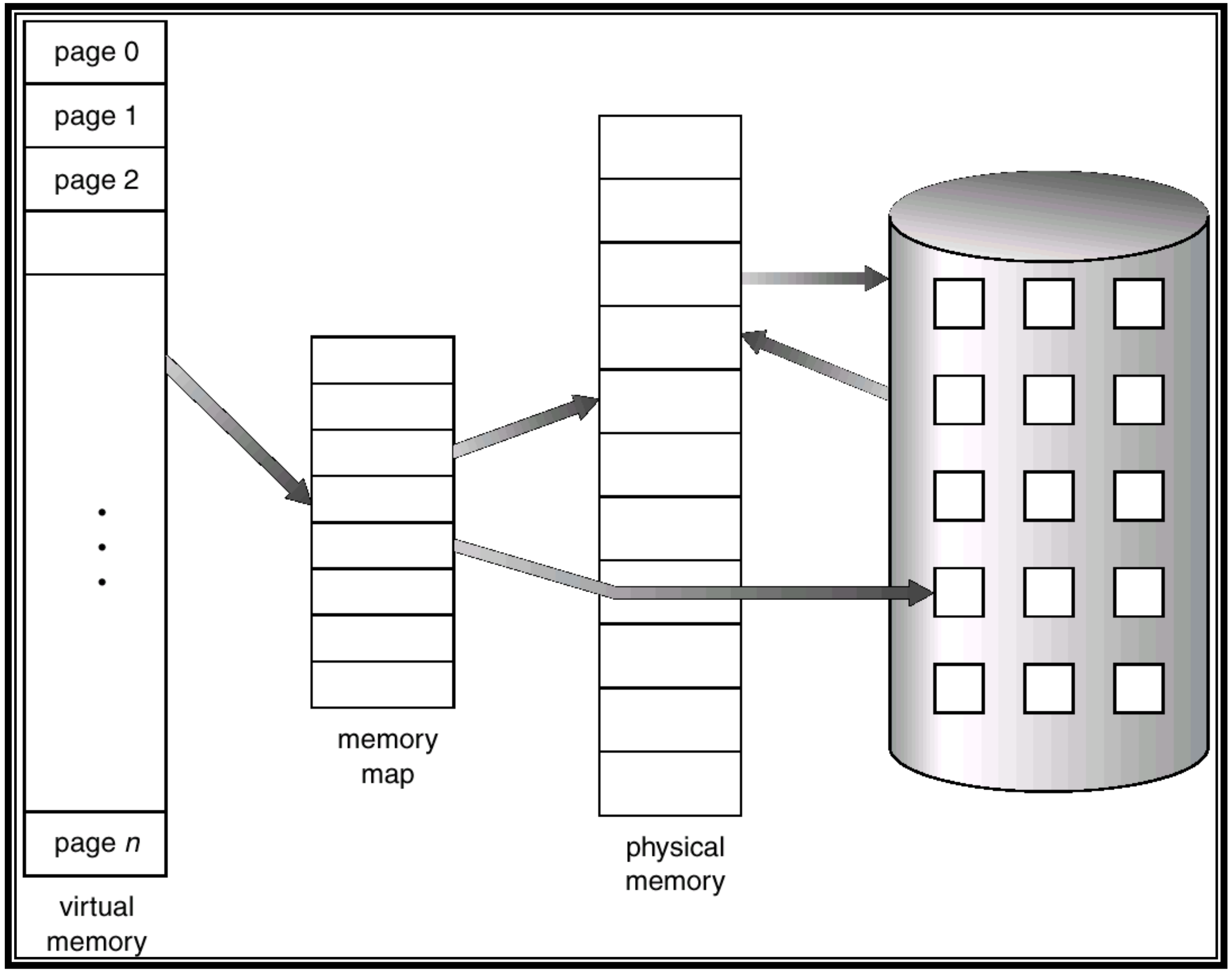
Virtual Memory That is Larger Than Physical Memory

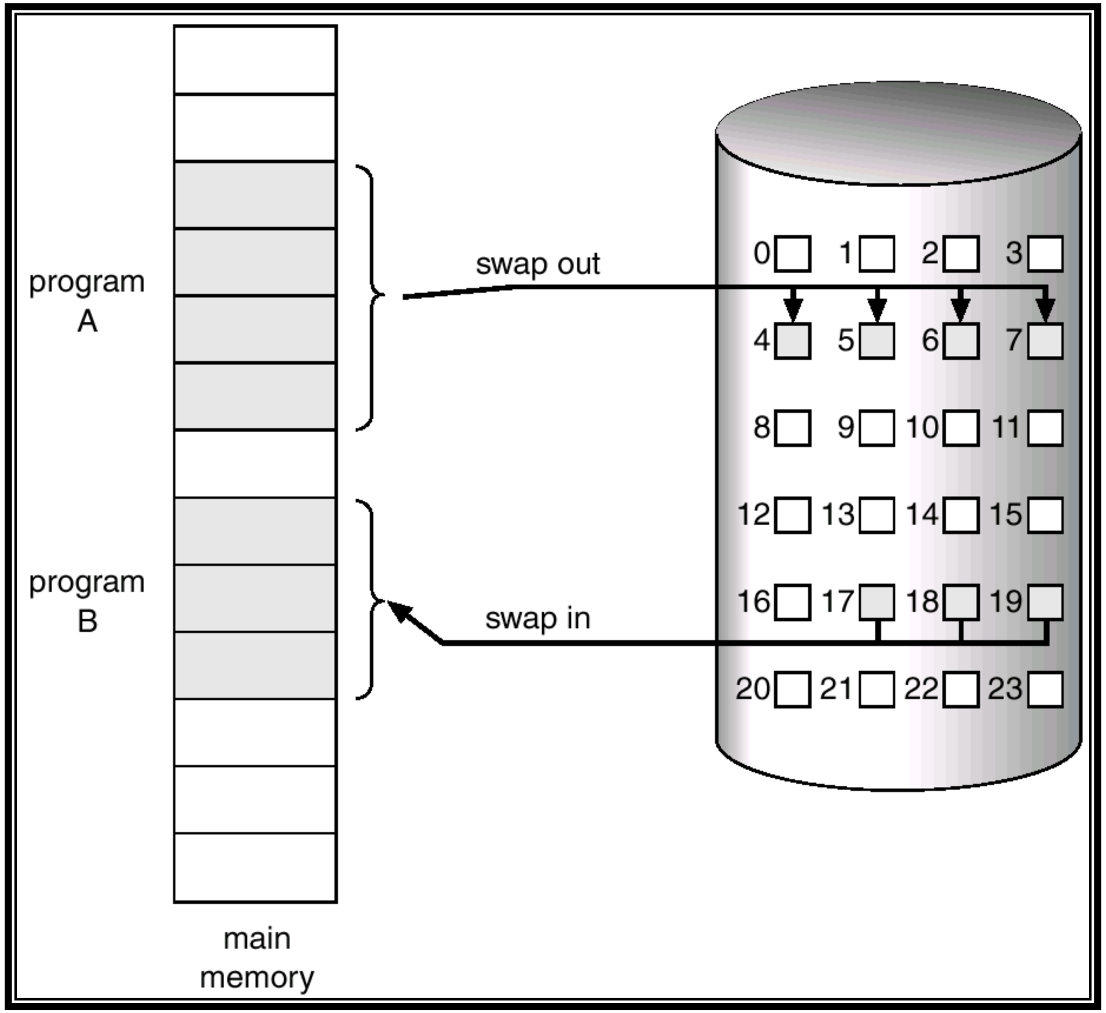


Demand Paging

- Bring a page into memory only when it is needed.
  - Less I/O needed
  - Less memory needed
  - Faster response
  - More users

- Page is needed ⇒ reference to it
  - invalid reference ⇒ abort
  - not-in-memory ⇒ bring to memory

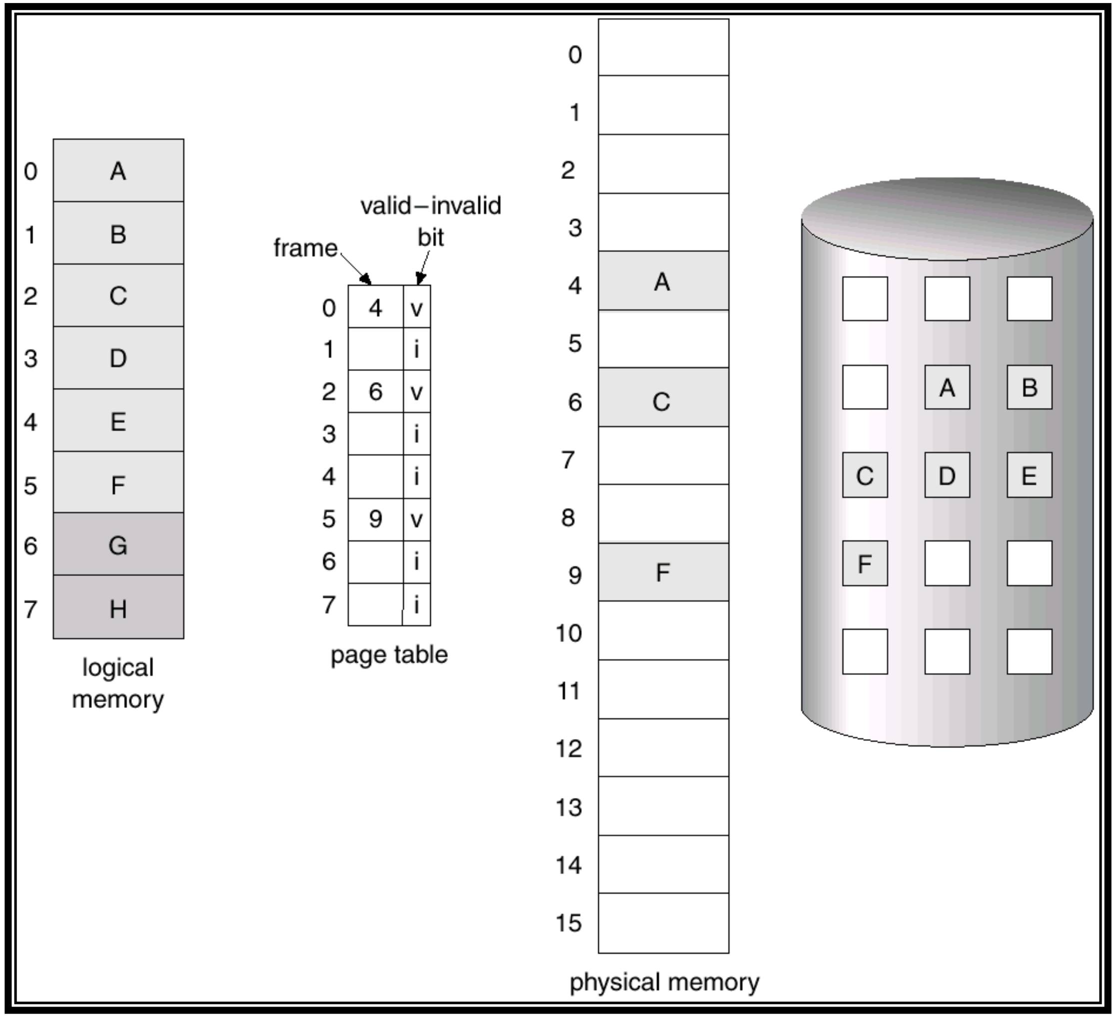
Transfer of a Paged Memory to Contiguous Disk Space



Valid-Invalid Bit

- With each page table entry a valid–invalid bit is associated
  (1 ⇒ in-memory, 0 ⇒ not-in-memory)
- Initially valid–invalid bit is set to 0 on all entries.
- During address translation, if valid–invalid bit in page table entry is 0 ⇒ page fault.

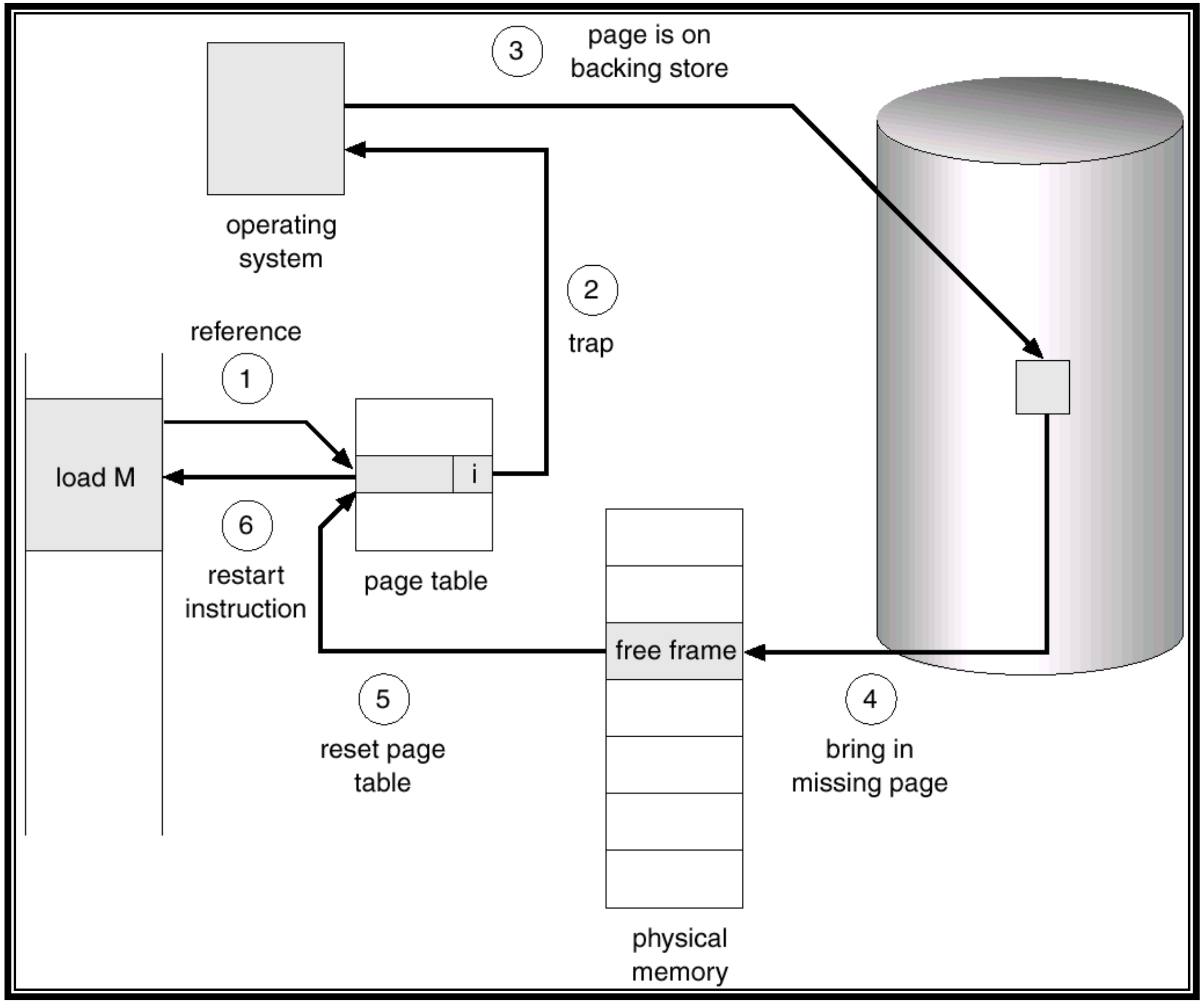
Page Table When Some Pages Are Not in Main Memory



Page Fault

- If there is ever a reference to a page, first reference will trap to
  OS ⇒ page fault
- OS looks at another table to decide:
  - Invalid reference ⇒ abort.
  - Just not in memory.
- Get empty frame.
- Swap page into frame.
- Reset tables, validation bit = 1.

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, swap it out.
  - algorithm
  - performance – want an algorithm which will result in minimum number of page faults.

- Same page may be brought into memory several times.

Performance of Demand Paging

- Page Fault Rate $0 \le p \le 1.0$
  - if $p = 0$ no page faults
  - if $p = 1$, every reference is a fault

- Effective Access Time (EAT)

$$\text{EAT} = (1 - p) \times \text{memory access} + p\,(\text{page fault overhead})$$

- Page fault overhead=service interrupt+read page in+restart proccess.

Demand Paging Example

- Memory access time = 200 nanosecond
- Average disk latency and seek time about 8ms=8x1000000 nanoseconds

EAT = (1 – p) x 200 + px8000000=

200+7999800xp nanoseconds

- Therefore page faults rate should be very low otherwise affects the overall performance of the system.

Copy-on-Write

- Copy-on-Write (COW) allows both parent and child processes to initially *share* the same pages in memory.

  If either process modifies a shared page, only then is the page copied.

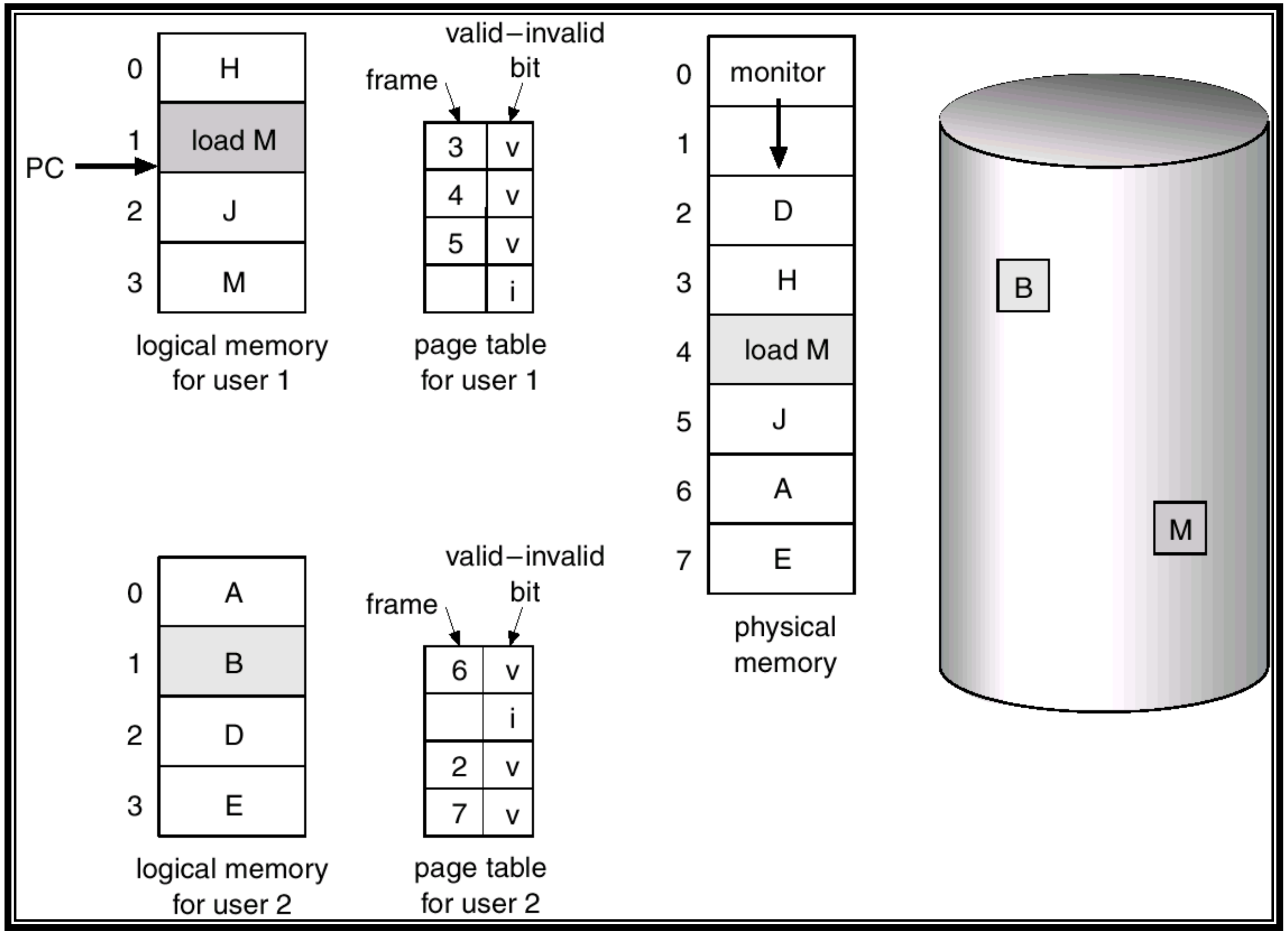
- COW allows more efficient process creation as only modified pages are copied.

- Free pages are allocated from a *pool* of zeroed-out pages.

Page Replacement

- Prevent over-allocation of memory by modifying page-fault service routine to include page replacement.
- Use *modify* (*dirty*) *bit* to reduce overhead of page transfers – only modified pages are written to disk.
- Page replacement completes separation between logical memory and physical memory – large virtual memory can be provided on a smaller physical memory.

Need For Page Replacement



Basic Page Replacement

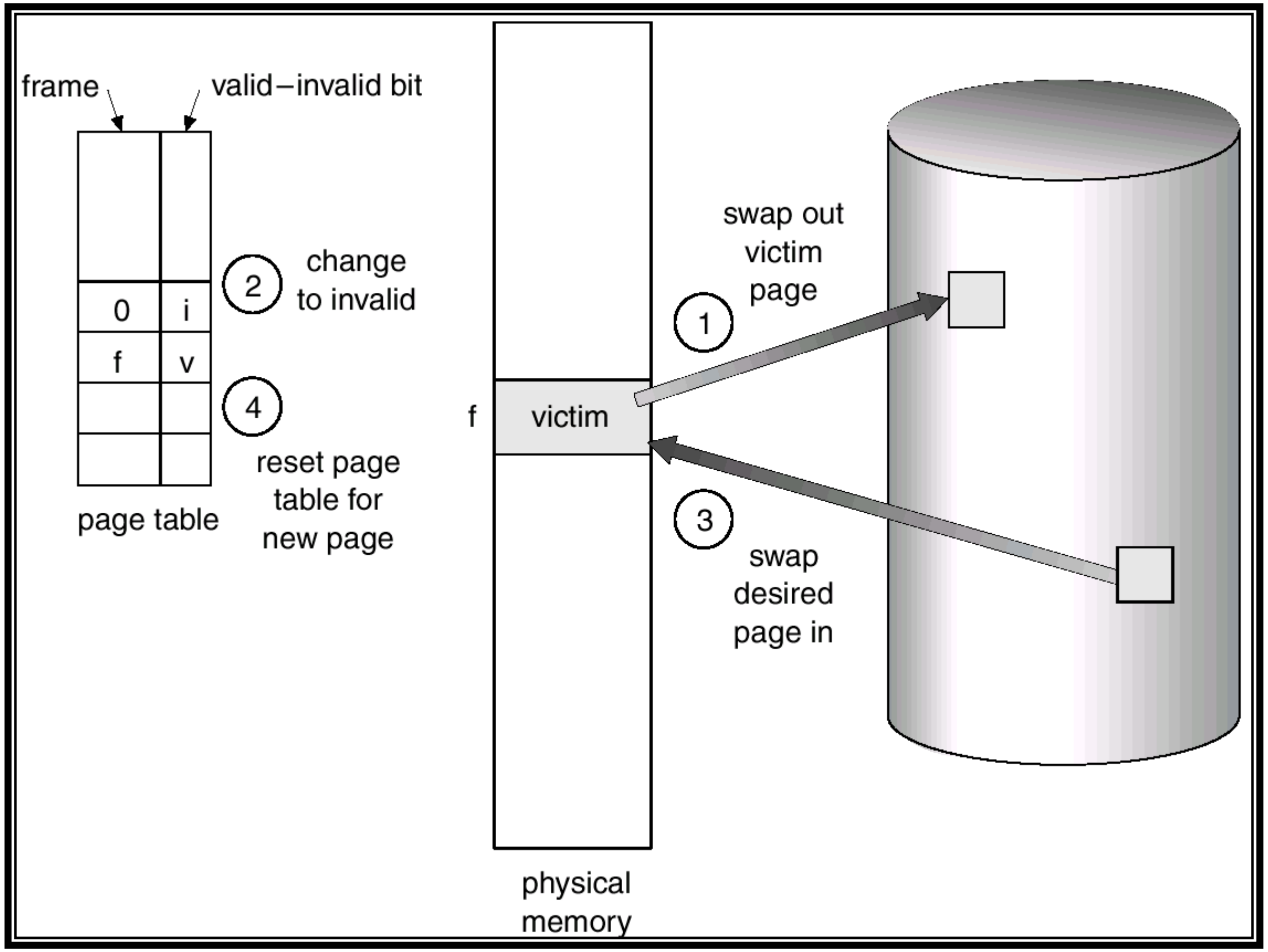
1. Find the location of the desired page on disk.

2. Find a free frame:
   - If there is a free frame, use it.
   - If there is no free frame, use a page replacement algorithm to select a *victim* frame.

3. Read the desired page into the (newly) free frame. Update the page and frame tables.

4. Restart the process.

Page Replacement

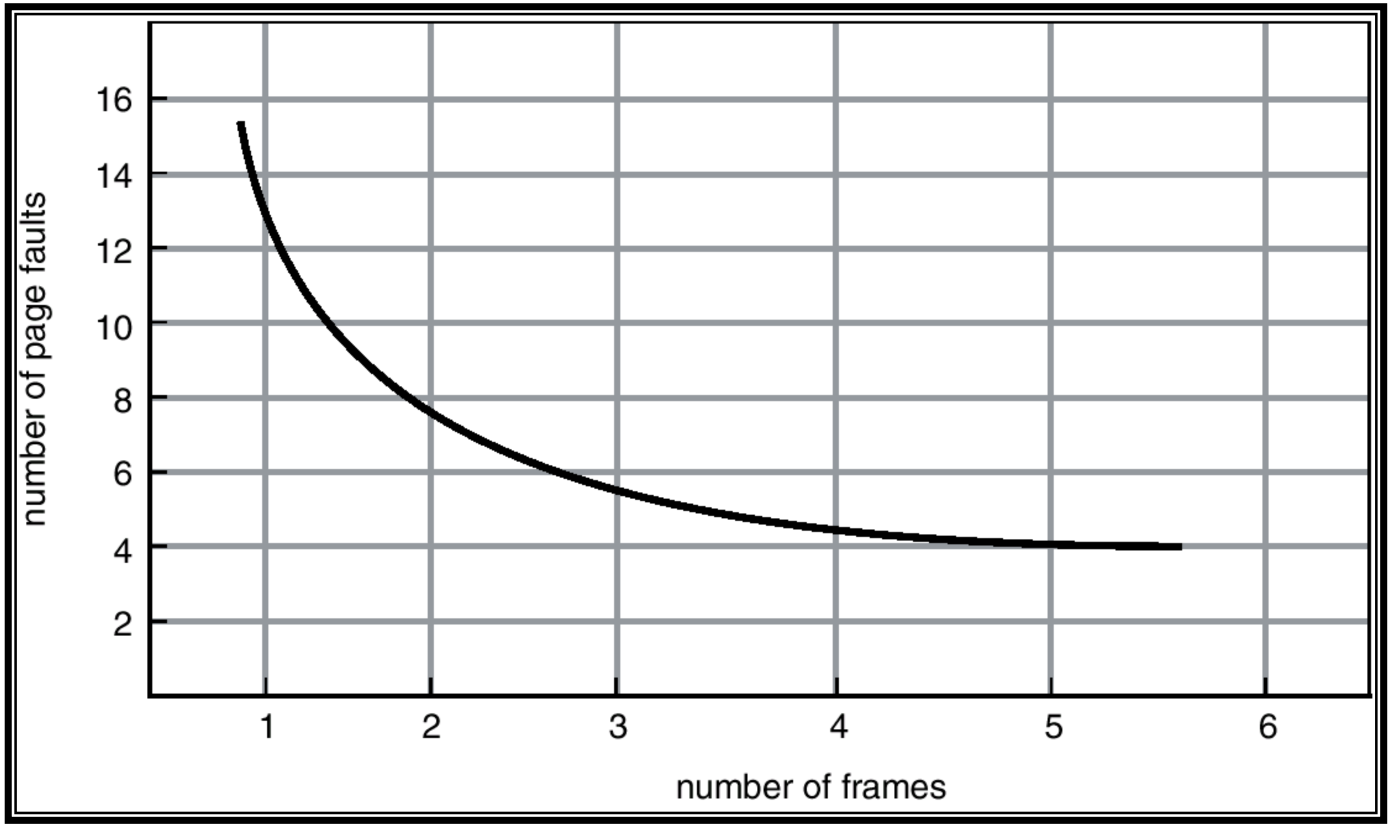


Page Replacement Algorithms

- Want lowest page-fault rate.
- Evaluate algorithm by running it on a particular string of memory references (reference string) and computing the number of page faults on that string.
- In all our examples, the reference string is

  1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5.

Graph of Page Faults Versus The Number of Frames



First-In-First-Out (FIFO) Algorithm

- Reference string: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5
- 3 frames

1	1	4	5
2	2	1	3
3	3	2	4

9 page faults

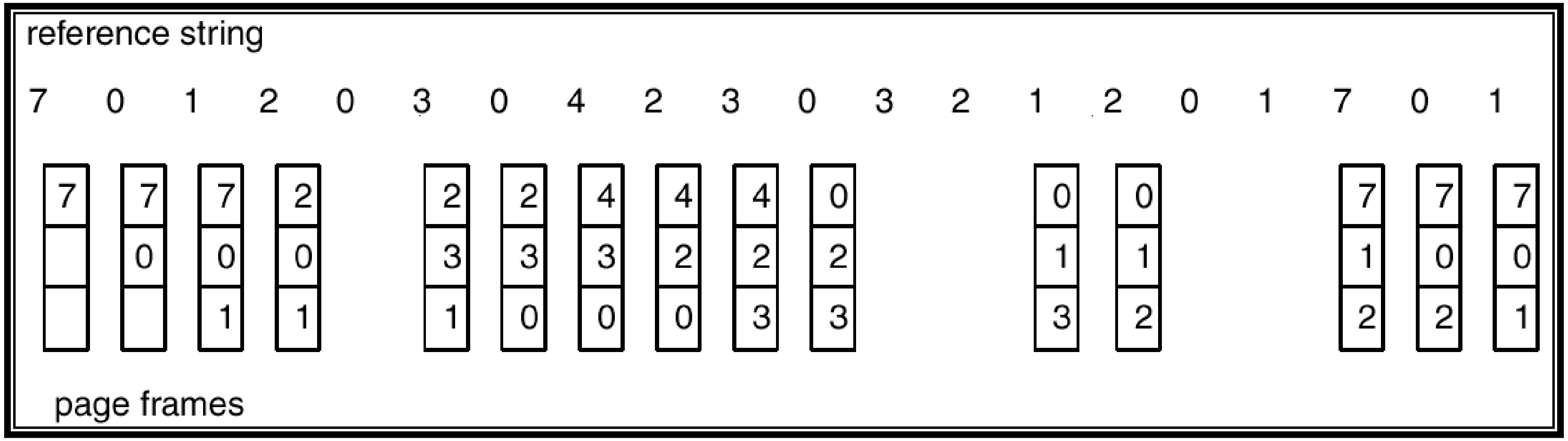
- 4 frames

1	1	5	4
2	2	1	5
3	3	2	
4	4	3	

10 page faults

- FIFO Replacement – Belady's Anomaly
  - more frames ⇒ more page faults

FIFO Page Replacement

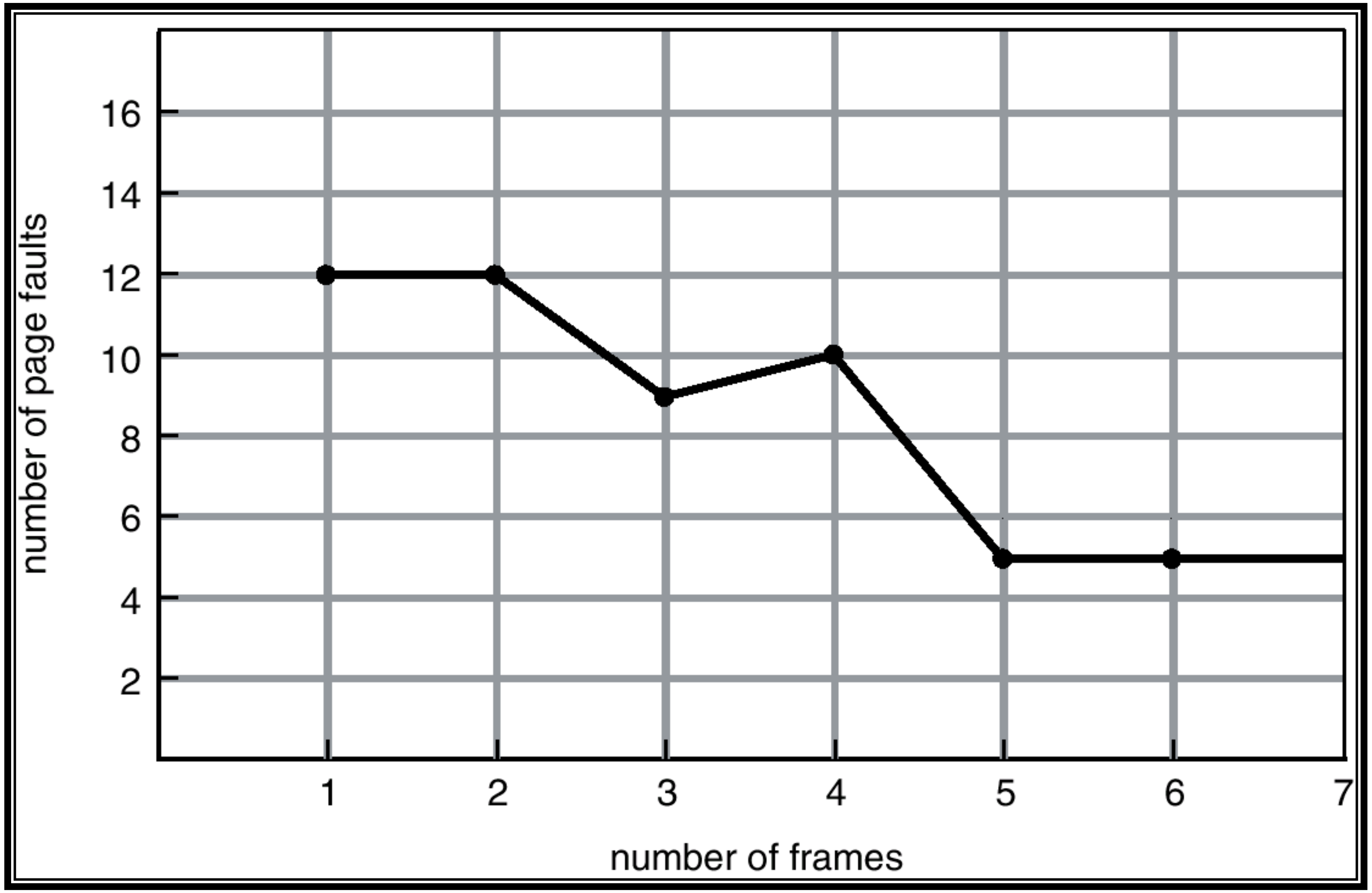


reference string

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

page frames

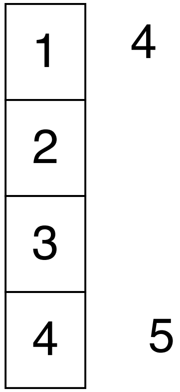
FIFO Illustrating Belady's Anamoly



Optimal Algorithm

- Replace page that will not be used for longest period of time.
- 4 frames example

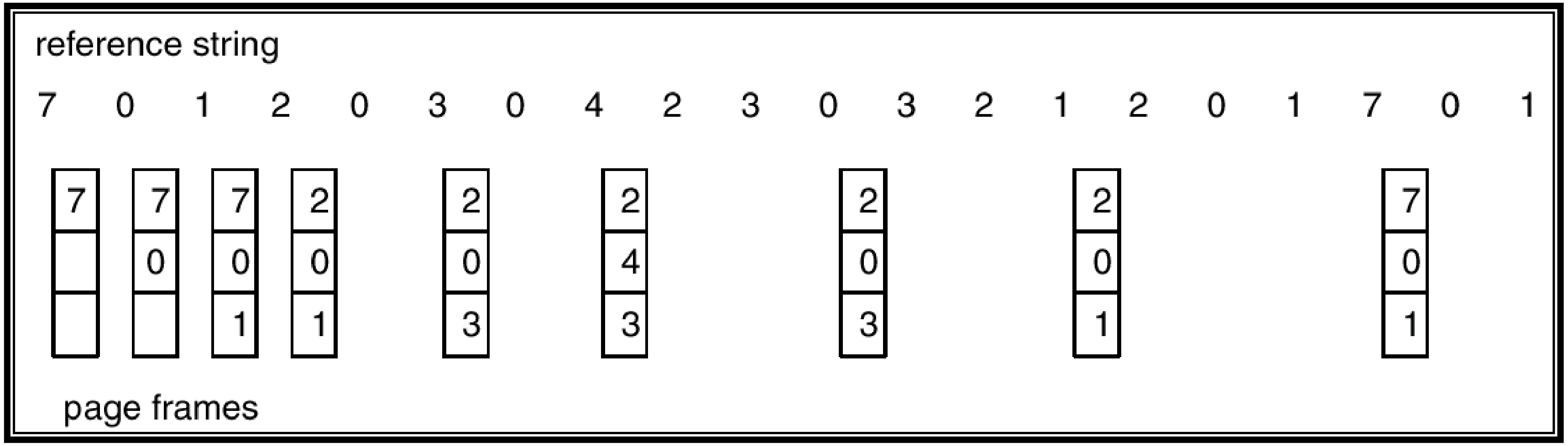
1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5



6 page faults

- How do you know this?
- Used for measuring how well your algorithm performs.

Optimal Page Replacement



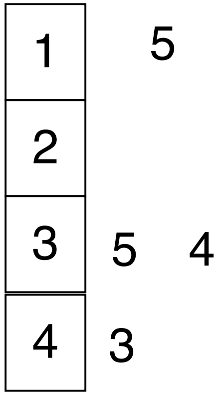
reference string

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

page frames

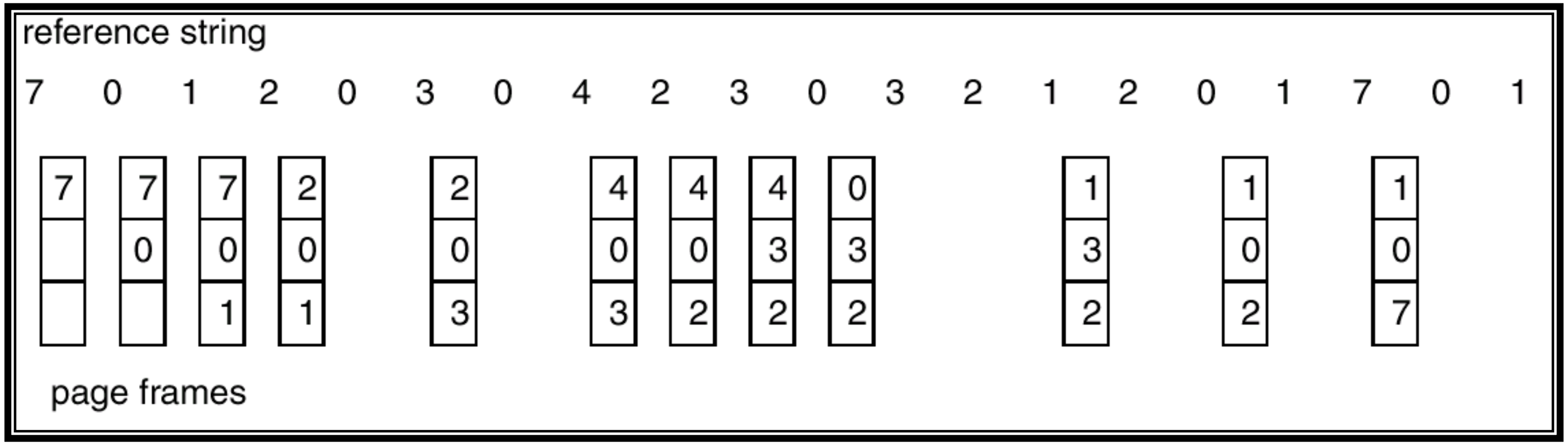
Least Recently Used (LRU) Algorithm

- Reference string: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5



- 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 determine which are to change.

LRU Page Replacement



reference string

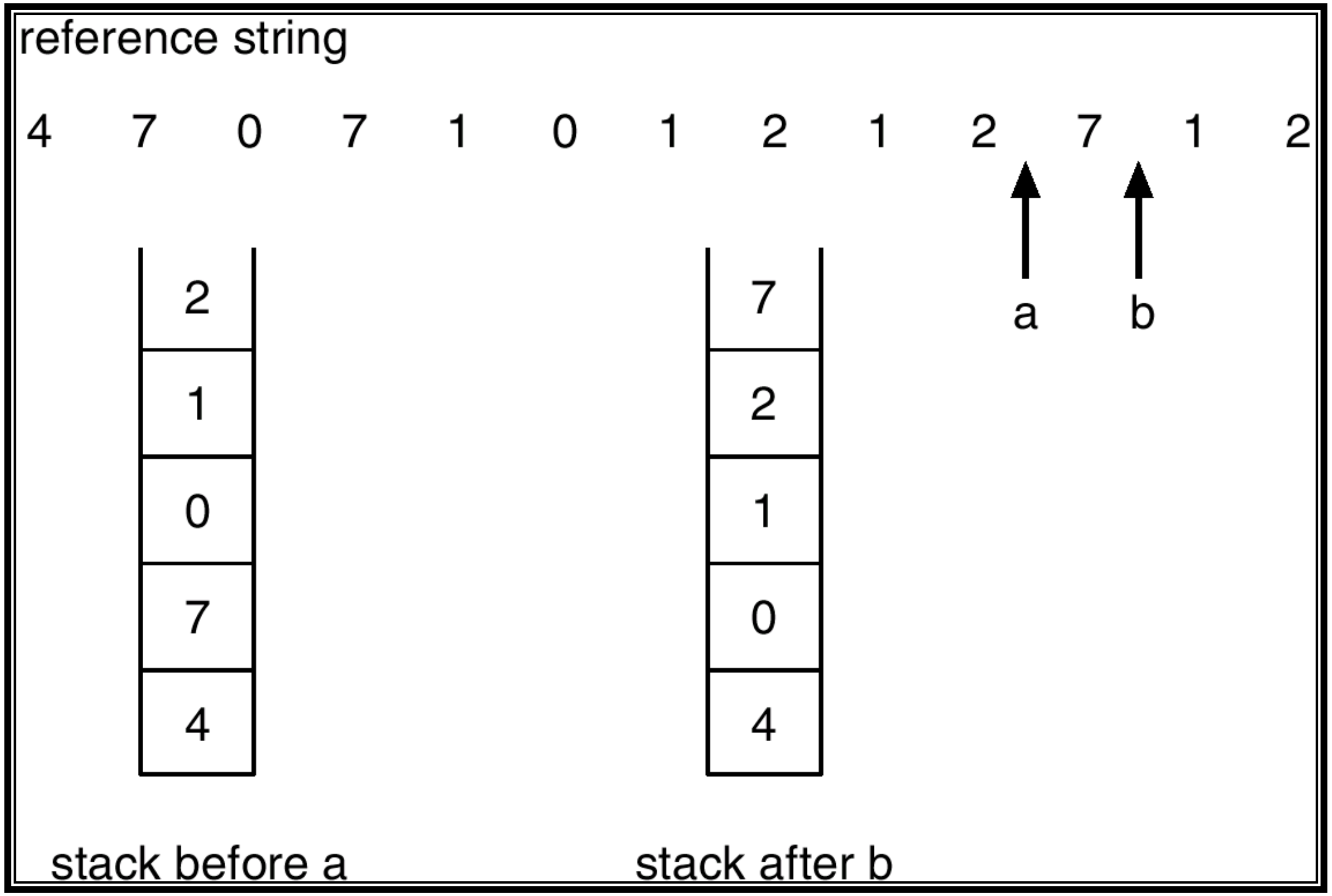
7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

page frames

LRU Algorithm (Cont.)

- Stack implementation – keep a stack of page numbers.
- Page referenced:
  - If on the stack
    - move it to the top
  - Otherwise pushed on the stack
- Replace page at the bottom of the stack.

Use Of A Stack to Record The Most Recent Page References



reference string

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

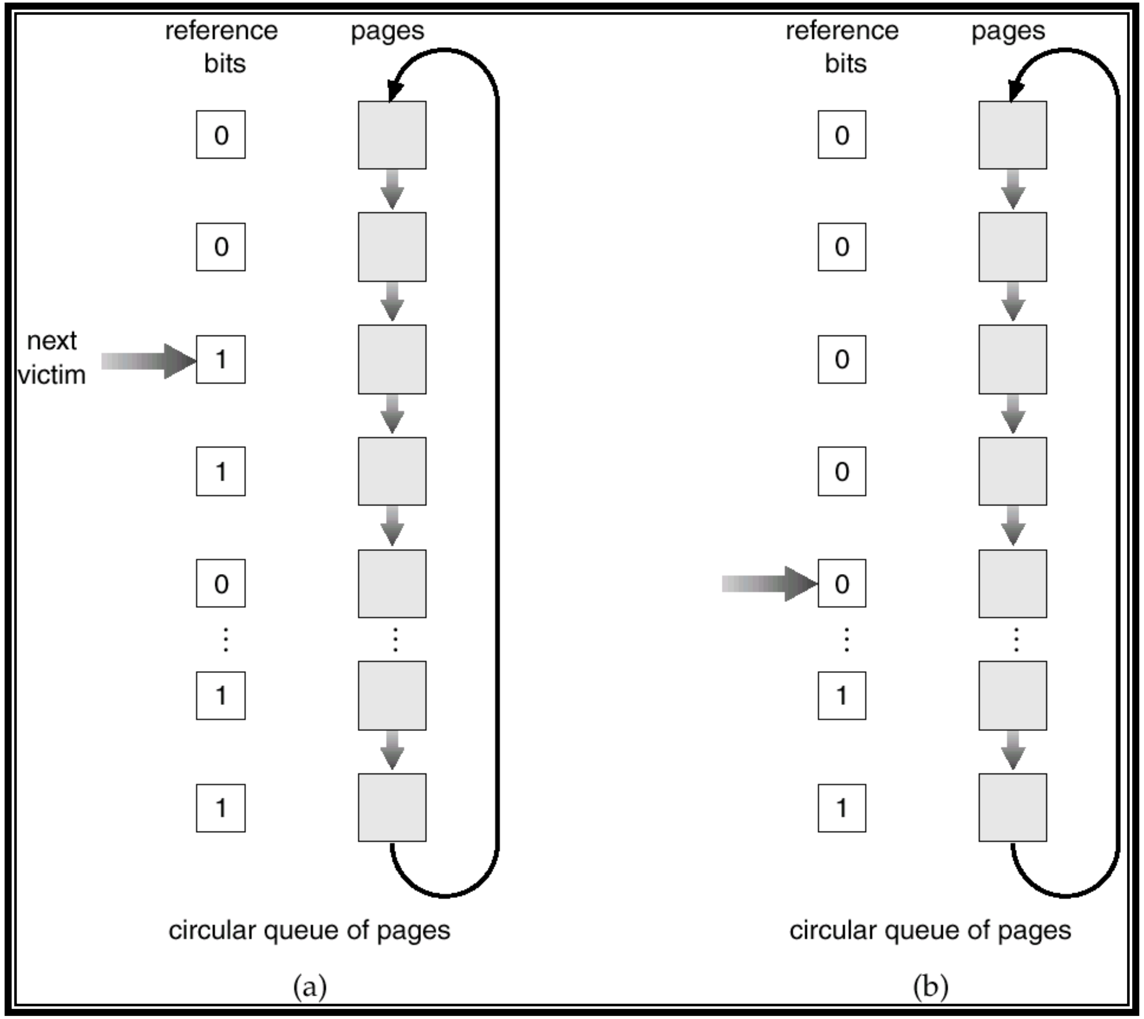
stack before a

stack after b

LRU Approximation Algorithms

- Reference bit
  - With each page associate a bit, initially = 0
  - When page is referenced, set bit to 1.
  - Replace the one which is 0 (if one exists). We do not know the order, however.
- Second chance
  - Need reference bit.
  - Clock replacement.
  - If page to be replaced (in clock order) has reference bit = 1, then:
    - set reference bit 0.
    - leave page in memory.
    - replace next page (in clock order), subject to same rules.

Second-Chance (clock) Page-Replacement Algorithm



Counting Algorithms

- Keep a counter of the number of references that have been made to each page.

- LFU Algorithm: replaces page with smallest count. Based on the argument that the page most used is still needed.
- MFU Algorithm: based on the argument that the page with the smallest count was probably just brought in and has yet to be used.

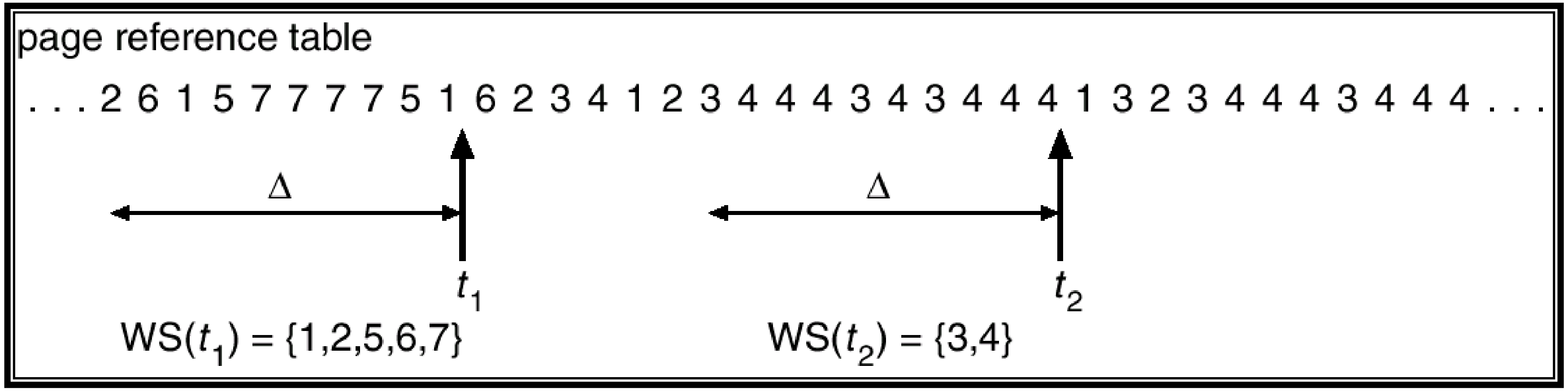
Thrashing

- In demand paging a process is started with none of its pages in memory.
- When the CPU tries to fetch the first instruction it gets a page fault.
- Using this strategy a process might get a large number of page faults when it starts.
- A program causing page faults every few instructions is said to be **thrashing**.

Working Set

- Most processes exhibit a **locality of reference.**
- During any phase of execution the process references only a small fraction of its pages.
- The set of pages that a process is currently using is called the **working set.**
- If the entire working set is in memory the process will run with no page faults.
- If the available memory is smaller than the working set the process will cause many page faults.

Working-set model



page reference table

. . . 2 6 1 5 7 7 7 7 5 1 6 2 3 4 1 2 3 4 4 4 3 4 3 4 4 4 1 3 2 3 4 4 4 3 4 4 4 . . .

$\mathrm{WS}(t_1) = \{1,2,5,6,7\}$

$\mathrm{WS}(t_2) = \{3,4\}$

- A good strategy
  - Keep track of the working set.
  - Make sure working set is in memory.
  - Prepaging.
- The working set changes slowly with time.
- We define the working set window *k* as the *k* most recent page references.
- Therefore the accuracy of our working set depends on the choice of *k*

- If k is too small it will not contain the entire locality
- If k is too large it will contain many localities.
- Let k_i be the working set size of process i.
- Define $D=\Sigma k_i$
  - If D is greater than the available frames thrashing will occur.
  - Usually k determined by adjusting the page fault frequency.

Memory Allocation

- When a process starts it is allocated a working set.
- If D becomes larger than the available frames
  - A process is suspended.
  - The process pages are swapped out.
  - The freed frames are reallocated to other processes.
- This strategy prevents thrashing while keeping the degree of multiprogramming as high as possible.

Page Replacement

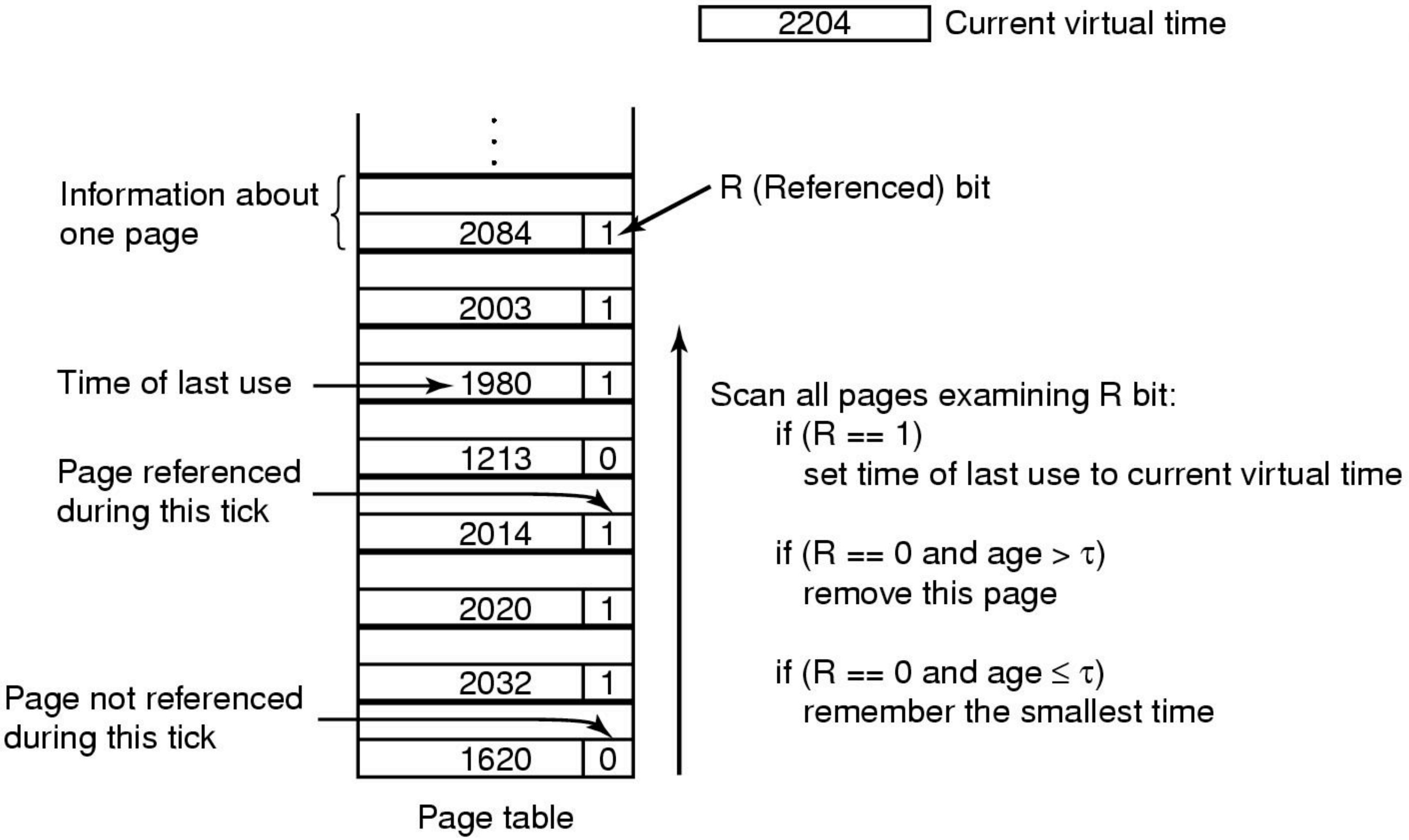
- Page replacement using the working set is simple.
- When a page fault occurs
  - Find a page not in the working set
  - Swap out the selected page.
- Do we scan for a page not in the working set of the process that caused the page fault (local replacement.
- Or scan for pages in all process (global replacement)?

Implementing the working Set

- Instead of defining the working set as those pages referenced during the previous, say, 10 million memory references,
- We can define it as those referenced during the last T msec of execution time.
- Each entry in the page table contains two fields
    - The time of last use.
    - Reference bit.

- The hardware sets the R bit whenever a page is referenced.
- We associate a clock with the algorithm.
  - We keep a counter of elapsed clock ticks called virtual time
  - At each clock tick the reference bit is cleared and the current virtual time is written into the time of last use field
  - We assume that the working set time spans multiple clock ticks.

- When a page fault occurs we inspected the page table entries
  - If R=1, the page is in the working set and is not removed.
  - If R=0, the page is a candidate for removal.
  - Compute the difference between the current time and the time of last use and compare it to T.
    - If the difference > T => the page is not in working set.
    - If we cannot find a page such that age>T select the oldest one.



```
Scan all pages examining R bit:
    if (R == 1)
        set time of last use to current virtual time

    if (R == 0 and age > τ)
        remove this page

    if (R == 0 and age ≤ τ)
        remember the smallest time
```

The working set algorithm

Page reference	123	41	52	362	62	47	
clock	0	1	2	3	4	5	6

C=1

1	R=0,LU=1
2	R=0,LU=1
3	R=0,LU=1

C=2

1	R=0,LU=2
2	R=0,LU=1
3	R=0,LU=1
4	R=0,LU=2

C=3

1	R=0,LU=2
2	R=0,LU=3
3	R=0,LU=1
4	R=0,LU=2
5	R=0,LU=3

6 referenced
Page replacement
Candidates 1&4

C=3

1	R=0,LU=2
2	R=0,LU=3
3	R=1,LU=1
4	R=0,LU=2
5	R=0,LU=3

	C=4
6	R=0,LU=4
2	R=0,LU=4
3	R=0,LU=4
4	R=0,LU=2
5	R=0,LU=3

	C=5
6	R=0,LU=5
2	R=0,LU=5
3	R=0,LU=4
4	R=0,LU=2
5	R=0,LU=3

7 referenced
Page replacement
Candidates 5

	C=5
6	R=0,LU=5
2	R=0,LU=5
3	R=0,LU=4
4	R=1,LU=2
5	R=0,LU=3