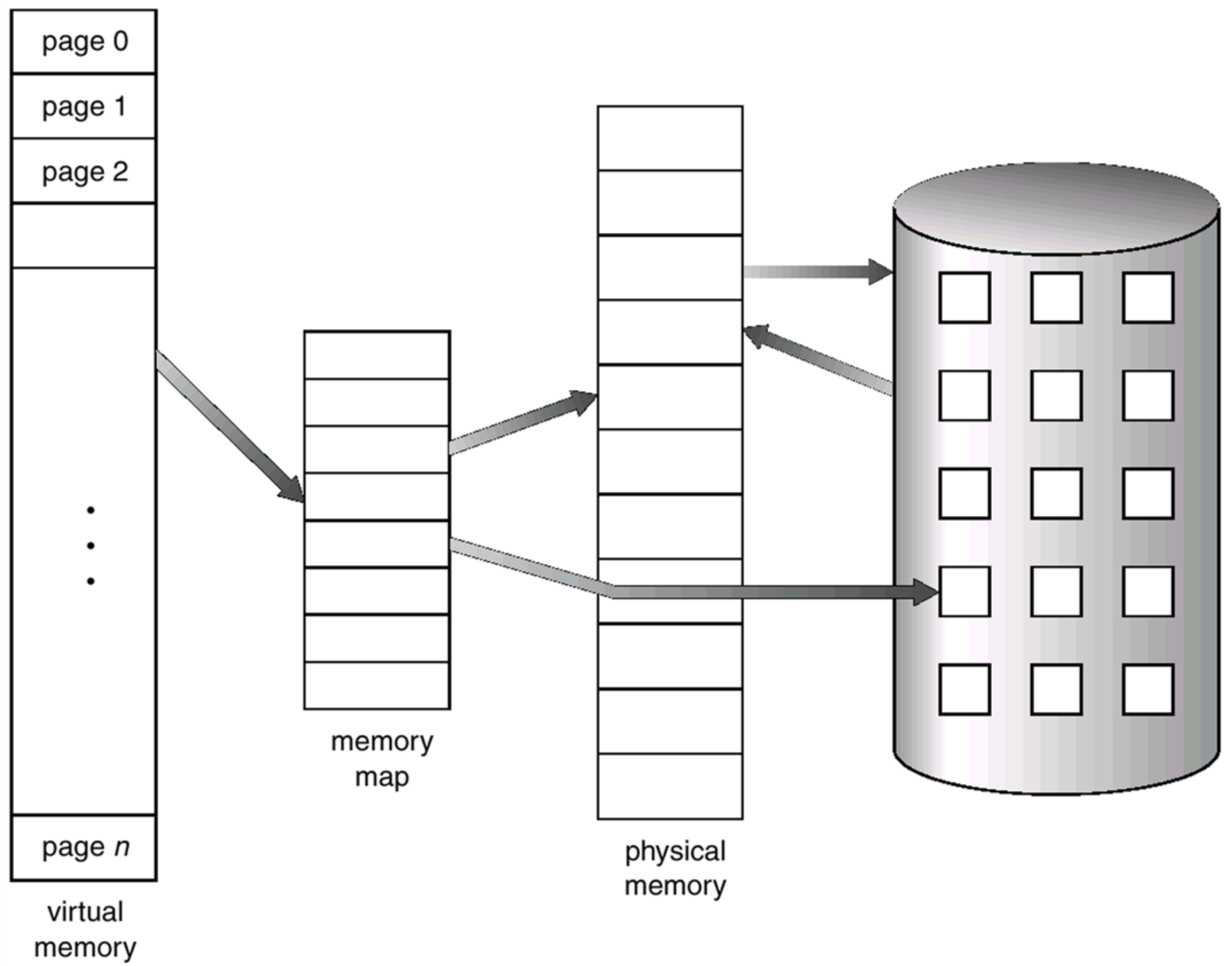
Virtual Memory

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
  - Allows address spaces to be shared by several processes
  - Allows for more efficient process creation
- Virtual memory can be implemented via:
  - Demand paging
  - Demand segmentation

Virtual memory:
result of a mechanism which combines main memory and secondary memories



Virtual Memory That is Larger Than Physical Memory

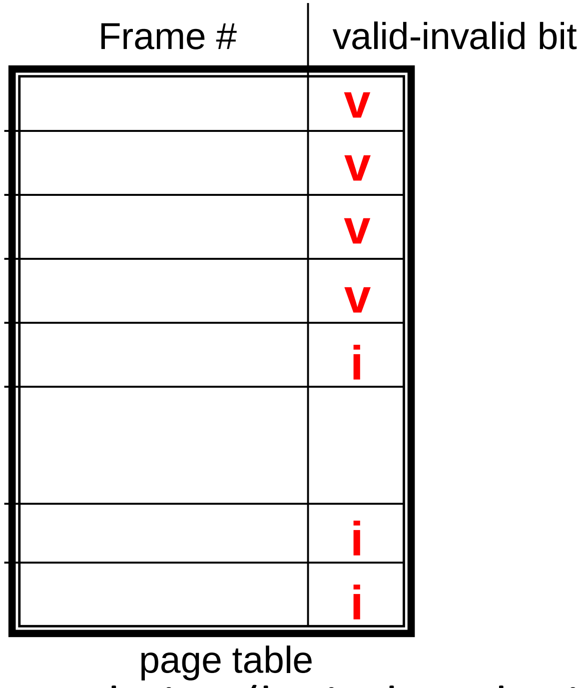
Demand Paging

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

- **Lazy swapper** – never swaps a page into memory unless page will be needed
  - Swapper that deals with pages is a **pager**

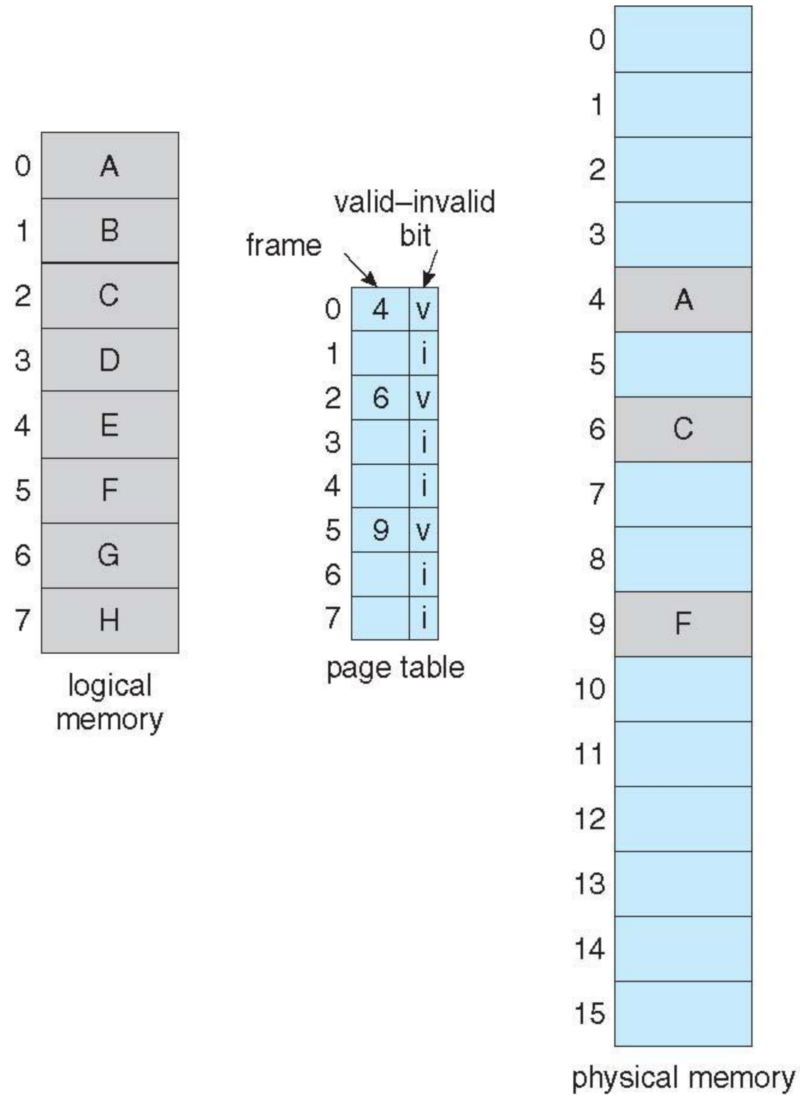
Valid-Invalid Bit

- With each page table entry a valid–invalid bit is associated (**v** ⇒ in-memory, **i** ⇒ not-in-memory)
- Initially valid–invalid bit is set to **i** on all entries
- Example of a page table snapshot:



- During address translation (logical to physical), if valid–invalid bit in page table entry
  is **I** ⇒ page fault

Page Table When Some Pages Are Not in Main Memory



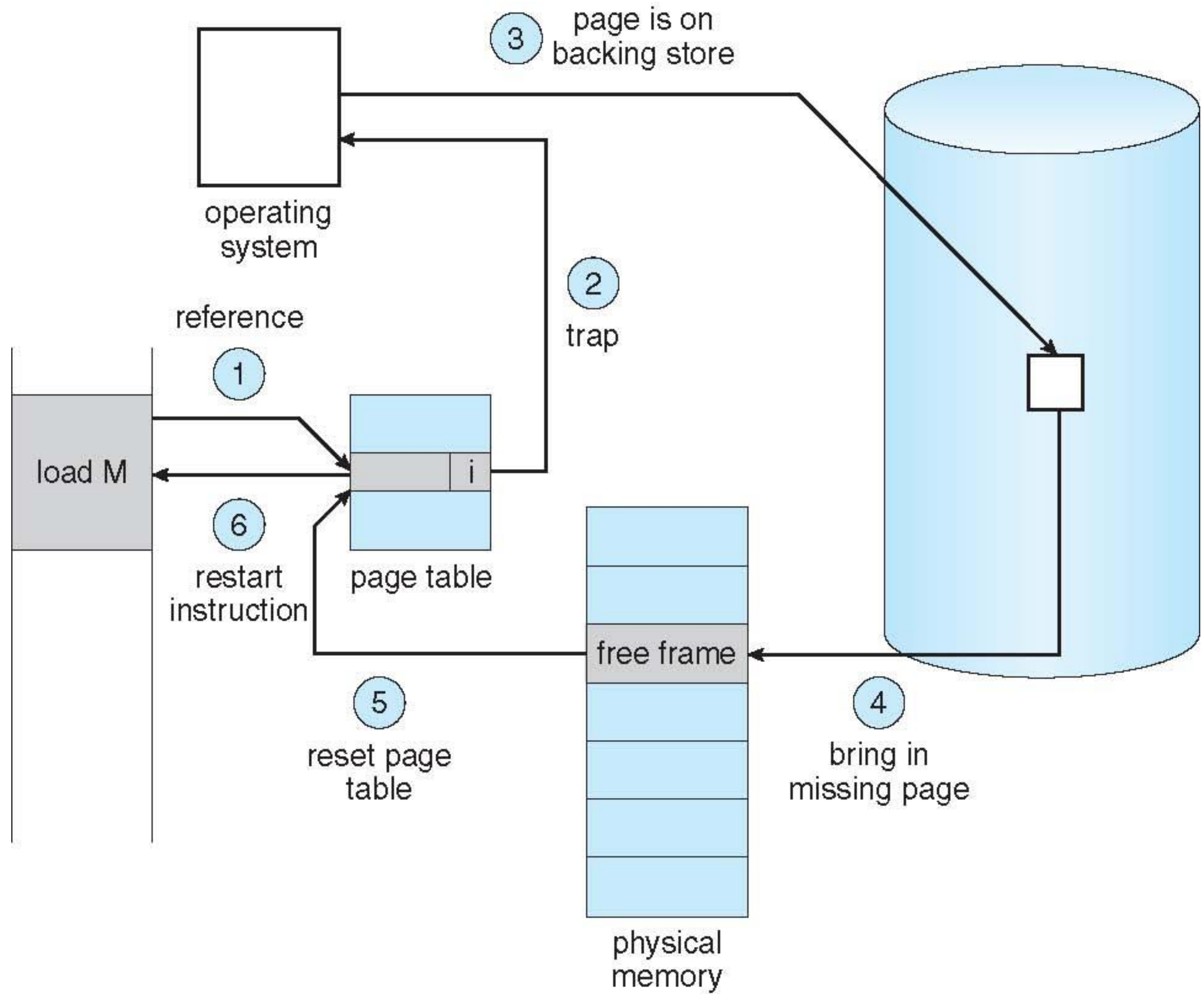
Page Fault

- An interruption is generated when the logical address refers to a part which is not in the resident set
  - defect of paging, fault page

1. Operating system looks at another table to decide:
   - - Invalid reference $\Rightarrow$ abort

     - Just not in memory
2. Get empty frame
3. Swap page into frame
4. Reset tables
5. Set validation bit = **v**
6. Restart the instruction that caused the page fault

Steps in Handling a Page Fault



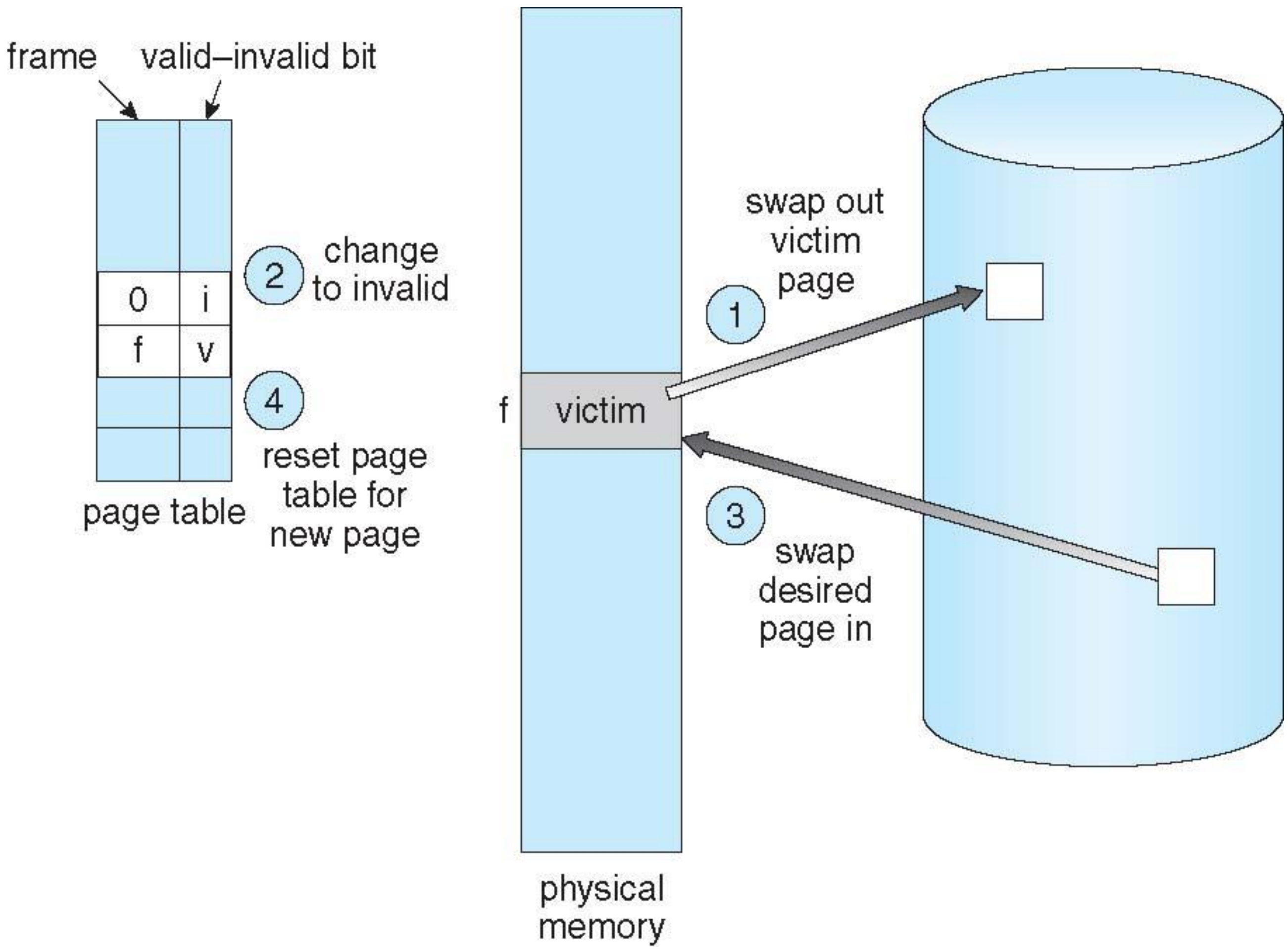
What happens if there is no free frame?

- Page replacement – find some page in memory, but not really in use, 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

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. Bring 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

Algorithms for the policy of replacement

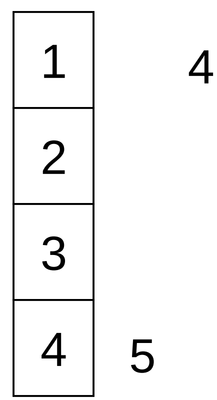
- The optimal algorithm (OPT) chooses for page to replace that will be referred most tardily (late)

  - produces the fewest number of page faults.
  - impossible to implement (need to know the future)
    - but serves as a standard to compare with the other algorithms we shall study.

Optimal Algorithm

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

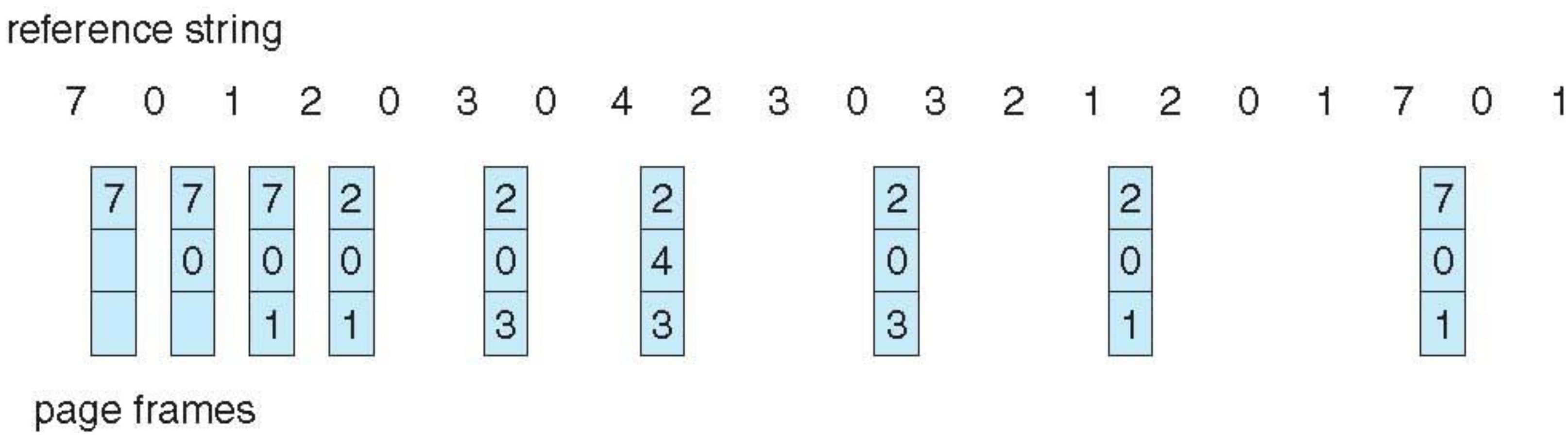
1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5



6 page faults

- How do you know this?
- Used for measuring how well your algorithm performs
- IMPOSSIBLE to implement
- We need to know the future
- Used to test the performance of other algorithms

Optimal Page Replacement



First In, First Out (FIFO)

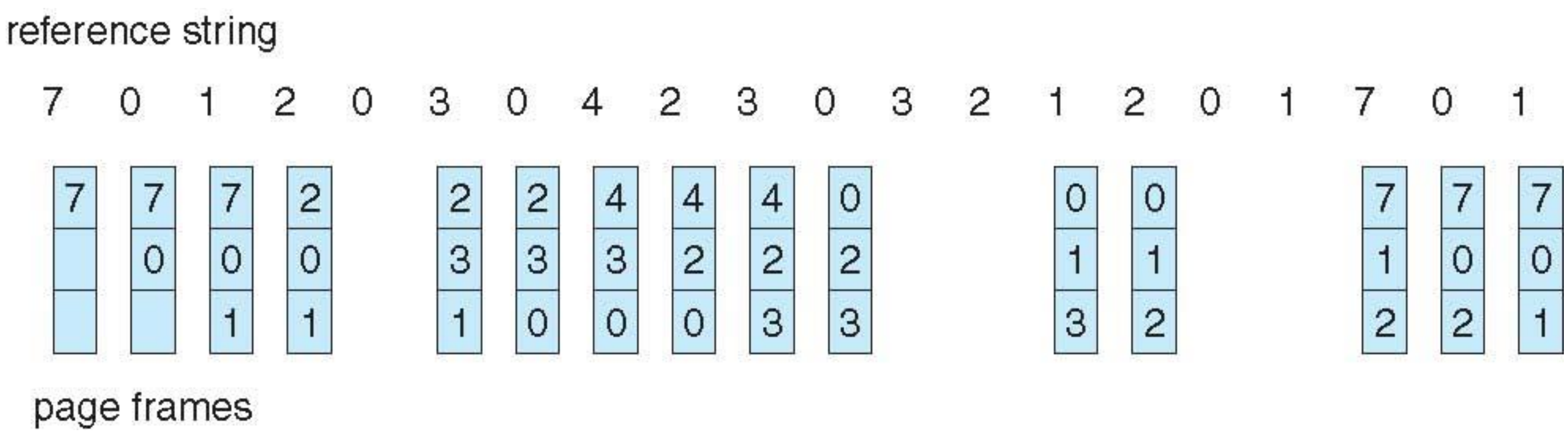
- Logic: a page which was a long time in memory had already its chance to be executed

- When the memory is full, the oldest page is replaced.
  - Thus: “first-in, first-out”

- Simple to apply

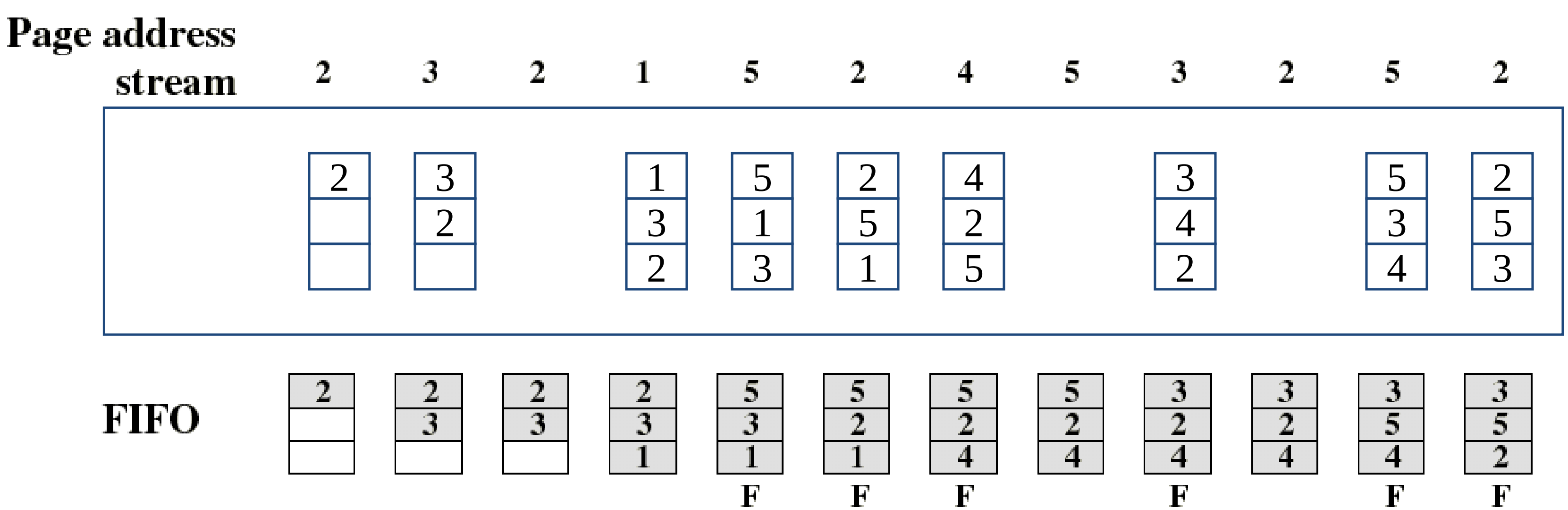
- But: A frequently used page is often the oldest,
  - it will be replaced by FIFO!

FIFO Page Replacement



Implementation of FIFO

- Easily implemented by using a queue of memory frameworks
  - Who should be updated at each defect of page
  - Exercise: Conceive this queue (previous example)



First-In-First-Out (FIFO) Algorithm

- Reference string: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5
- 3 frames (3 pages can be in memory at a time per process)

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	

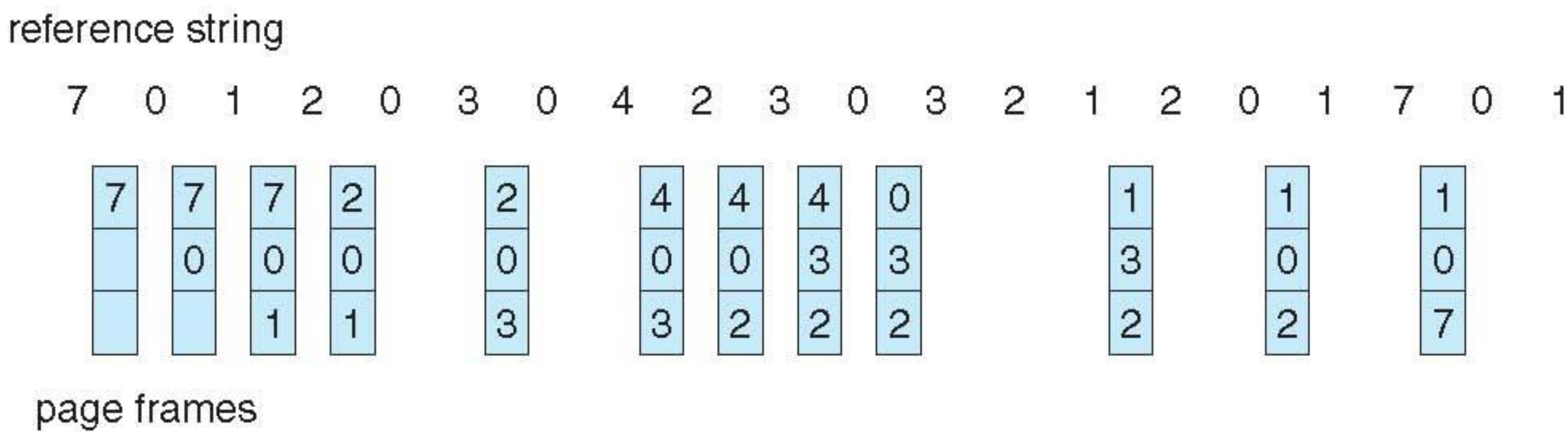
10 page faults

- Belady's Anomaly: more frames $\Rightarrow$ more page faults

Algorithms for the policy of replacement

- Chronological order of use (LRU)
  - Least Recently Used
  - Replace the page whose last reference goes back to the most remote time
    - past is used to predict the future

LRU Page Replacement



Comparison between FIFO and LRU

Page address stream	2	3	2	1	5	2	4	5	3	2	5	2
LRU	2	2	2	2	2	2	2	2	3	3	3	3
		3	3	3	5	5	5	5	5	5	5	5
				1	1	1	4	4	4	2	2	2
					F		F		F	F		
FIFO	2	2	2	2	5	5	5	5	3	3	3	3
		3	3	3	3	2	2	2	2	2	5	5
				1	1	1	4	4	4	4	4	2
					F	F	F		F		F	F

- Contrary to FIFO, LRU recognizes that the pages 2 and 5 are frequently used
- In this case, the performance of FIFO is worse:
  - LRU = 3+4, FIFO = 3+6

Comparison OPT-LRU

- Example: A process of 5 pages, and there are only 3 available physical pages.
- In this example, OPT causes 3+3 defects, LRU 3+4.

Page address stream	2	3	2	1	5	2	4	5	3	2	5	2
OPT	2	2	2	2	2	2	4	4	4	2	2	2
		3	3	3	3	3	3	3	3	3	3	3
				1	5	5	5	5	5	5	5	5
					F		F			F		
LRU	2	2	2	2	2	2	2	2	3	3	3	3
		3	3	3	5	5	5	5	5	5	5	5
				1	1	1	4	4	4	2	2	2
					F		F		F	F		