VM: Beyond Physical

Sridhar Alagar

Not enough Physical Memory

- Single process' address space can be larger than the physical memory
  - Need this for flexibility in programming
- Combined address spaces of all the processes larger than physical memory
- OS needs to provide illusion of large physical memory

Not enough Physical Memory

A process request memory for a page and the memory is full. How to handle this request?

Demand Paging

- Load a page into memory on demand
- If page needed is not in memory
  - Need to detect and load the page into memory from storage
    - without changing program behavior
    - without programmer needing to change code
- Need hardware support

Demand Paging

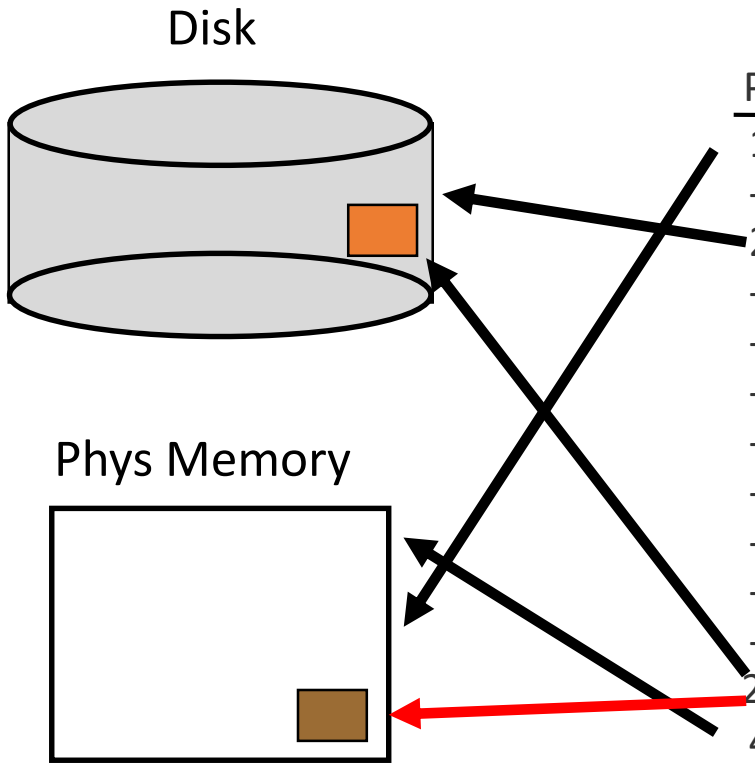
- Mechanism
  - How to identify whether a page is in memory or on disk?
  - How to fetch page from disk transparently?

- Policy
  - Which page(s) should be replaced?

Mechanism

- Each page in virtual address space maps to one of three locations
  - Physical main memory: Small, fast, expensive
  - Disk (backing store): Large, slow, cheap
  - Nothing (error): Free
- Extend page tables with an extra bit: present
  - permissions (r/w), valid, present
  - Page in memory: present bit **set** in PTE
  - Page on disk: present bit cleared
    - PTE points to block on disk
    - Causes trap into OS when page is referenced **(page fault)**

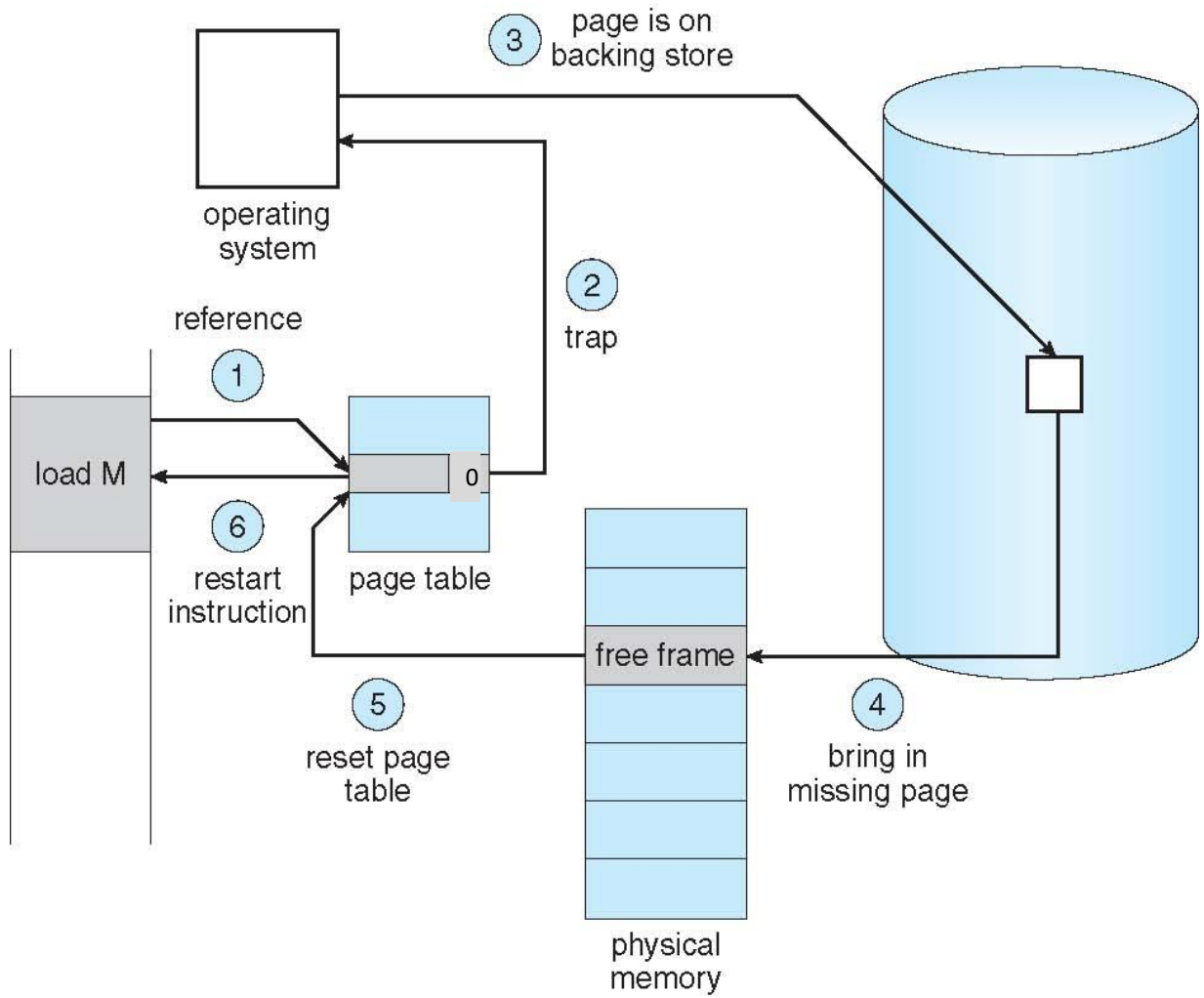
Present Bit



PFN	valid	prot	present
10	1	r-x	1
-	0	-	-
23	1	rw-	0
-	0	-	-
-	0	-	-
-	0	-	-
-	0	-	-
-	0	-	-
-	0	-	-
-	0	-	-
-	0	-	-
28 6	1 1	rw-	0 1
4	1	rw-	1

What if vpn 0xb referenced?

Handling Page Fault

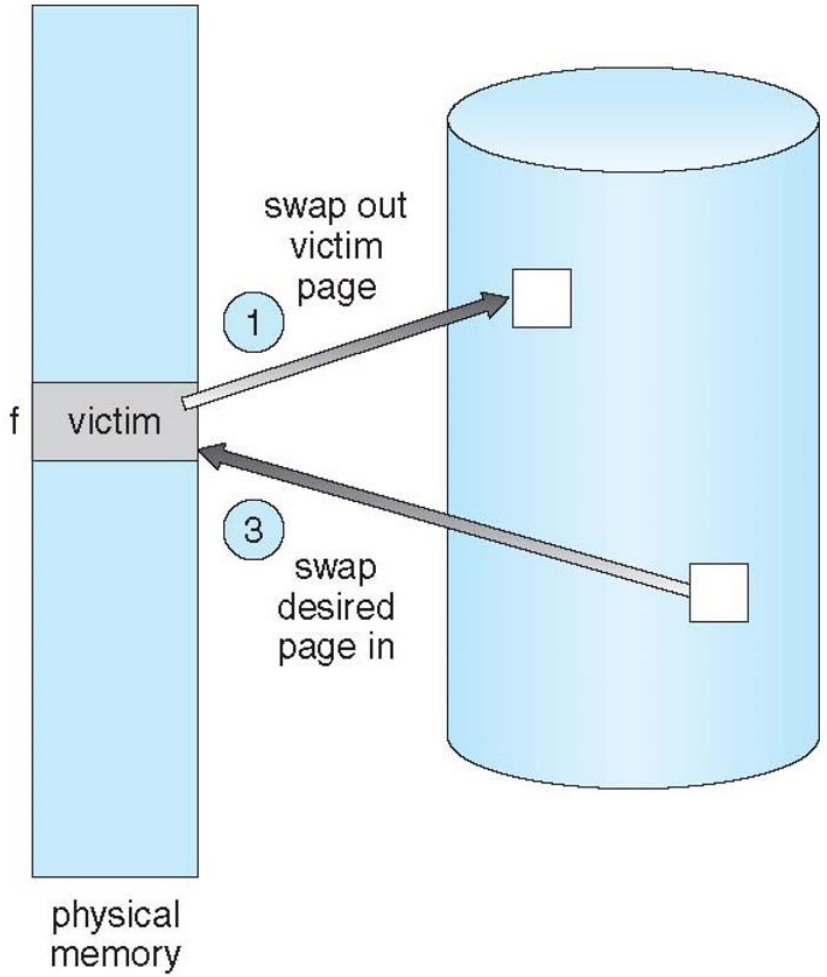


Aspects of demand paging

- Extreme case – start process with *no* pages in memory
  - Page fault on first access of every process pages
  - **Pure demand paging**
- An instruction could access multiple pages -> multiple page faults
- Hardware support needed for demand paging
  - Page table with protection bit
  - Secondary memory (swap device with **swap space**)
  - Instruction restart

Frame Replacement

- No free frame to bring in a page
- Find a victim to evict
- Policy decision
- Plenty of time to make



Replacement algorithms

- Goal
  - Reduce the number of page faults

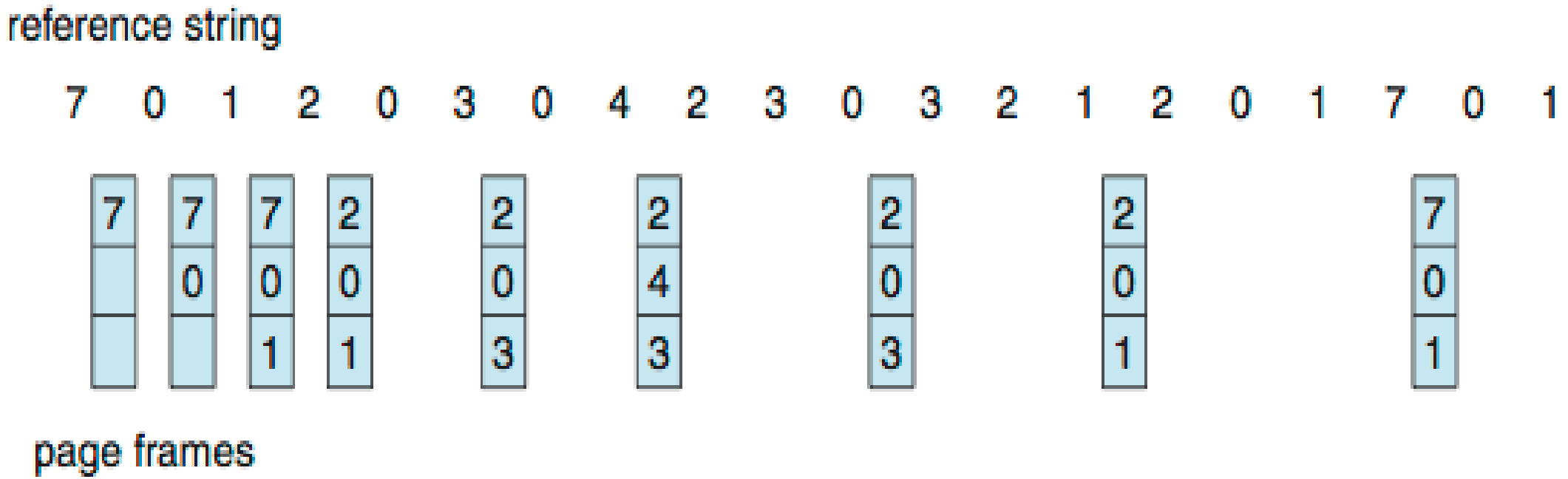
- Several replacement algorithms

The Optimal replacement Policy

- Leads to the fewest number of faults overall
- Replaces the page that will be accessed furthest in the future
- Serve only as a comparison point, to know how close we are to perfect

Optimal Policy

Assume 3 frames

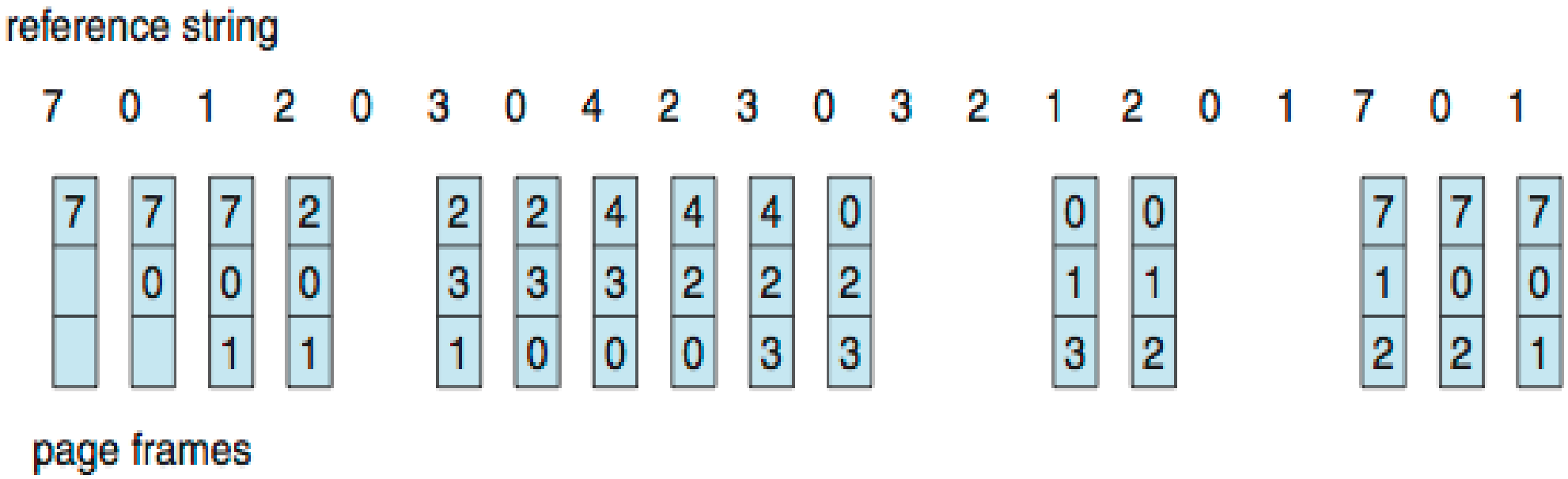


9 faults

FIFO

- Pages are placed in a queue when they enter the system.
- When a replacement occurs, the page on the tail of the queue(the "First-in" pages) is evicted.
- It is simple to implement.

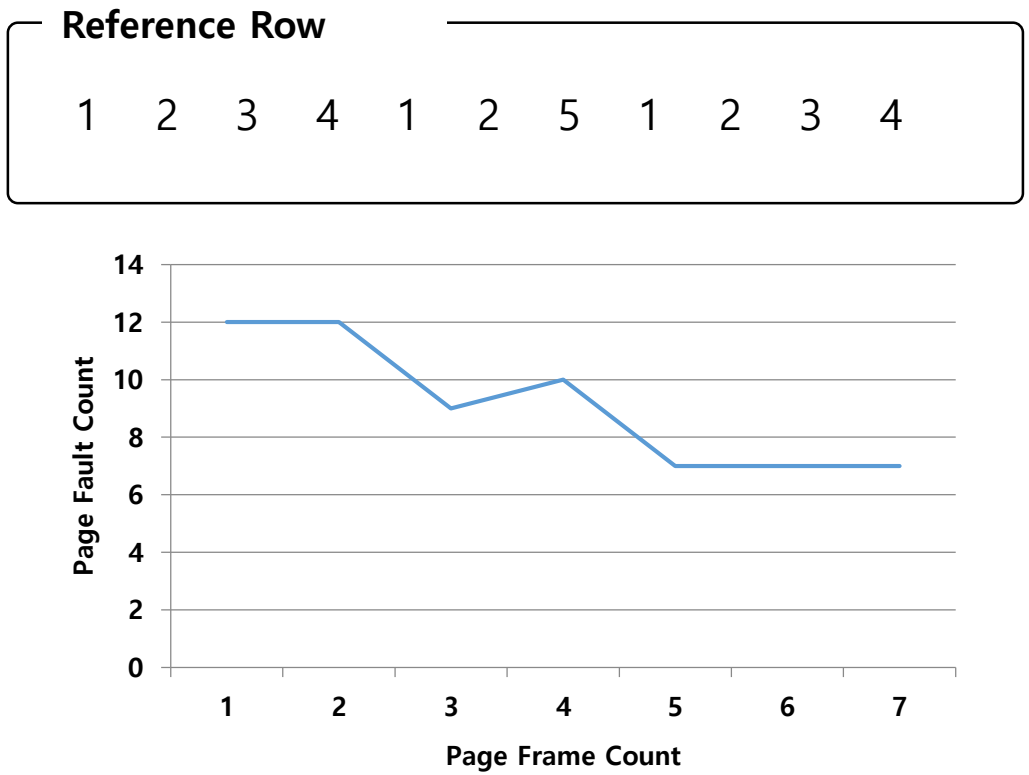
FIFO



15 faults

Belady's anomaly in FIFO

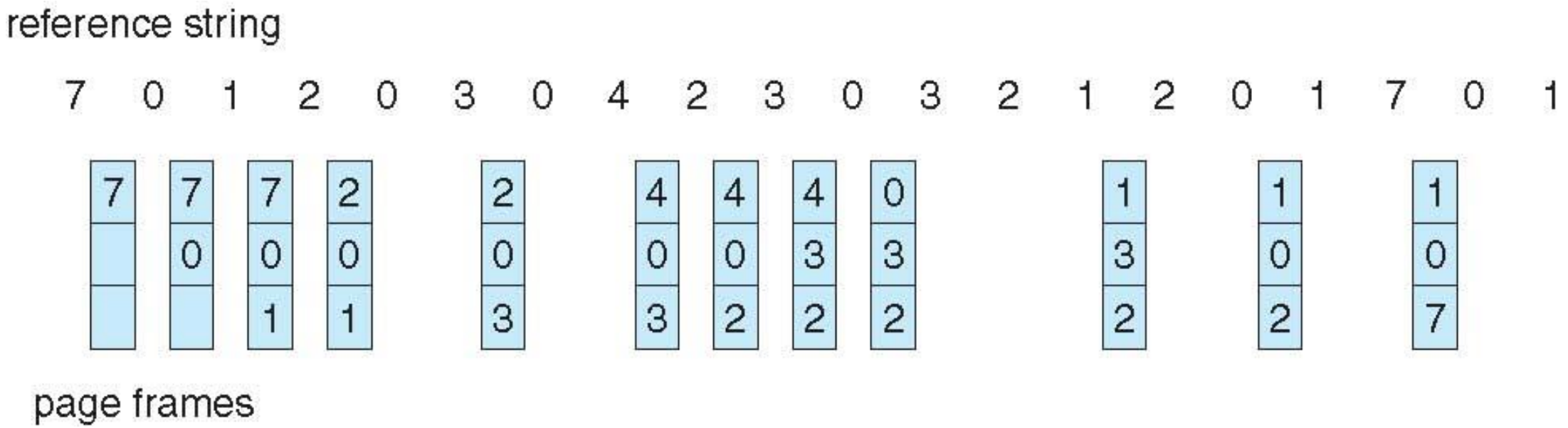
- Increasing the number frames to 4 increases the page fault



Least Recently Used (LRU)

- Since future is not known, use past to predict
- Replace the least recently used frame

Least Recently Used (LRU)



12 faults

Implementing LRU

- Software Perfect LRU

- OS maintains ordered list of physical pages by reference time
  - When page is referenced: Move page to front of list
  - Victim: last page of the list
  - **Trade-off**: Slow on memory reference, fast on replacement

Implementing LRU

- Hardware Perfect LRU
- Associate timestamp register with each frame
- When frame is referenced: Store system clock in register
- Victim: Scan through registers to find the oldest timestamp
- **Trade-off**: Fast on memory reference, slow on replacement (especially as size of memory grows)

Approximate LRU

- In practice, do not implement Perfect LRU
- LRU is an approximation anyway, so approximate more

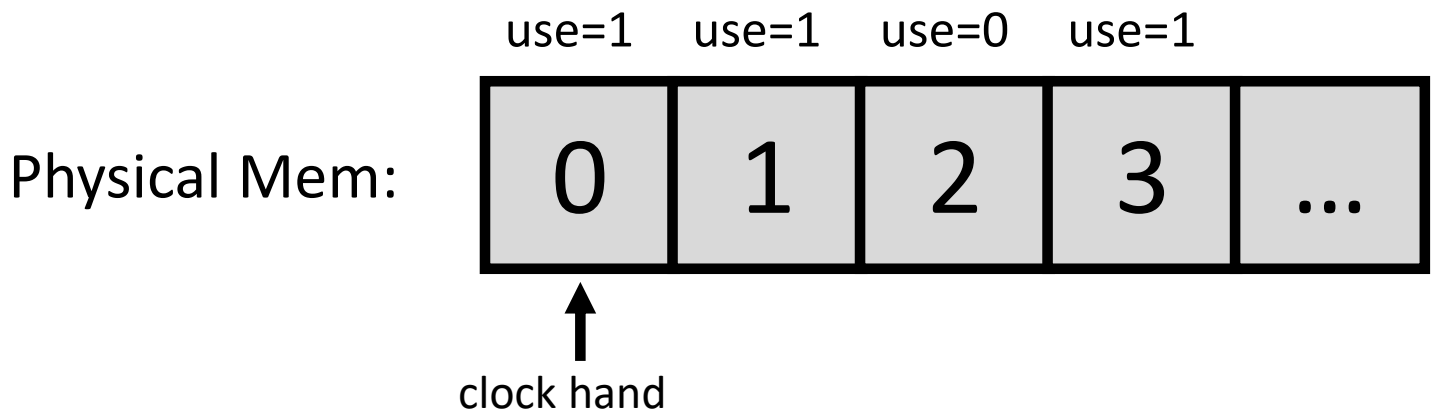
- Goal: Find an old page, but not necessarily the very oldest

- Hardware provides a single bit – reference/use bit
- The bit is set whenever the page is referenced

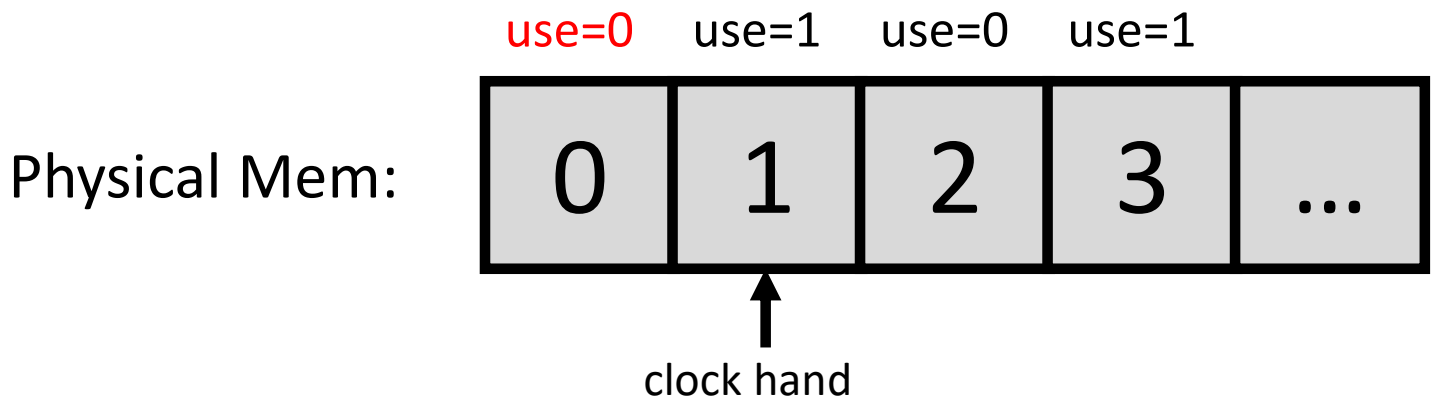
Clock Algorithm

- OS executes page replacement algorithm:
  - Look for page with reference bit cleared (has not been referenced for awhile)
- Implementation:
- Keep a pointer to last replaced frame
- Traverse frames in circular fashion
- Clear reference bit if it is set
- Stop when found a page with already cleared bit, replace this frame

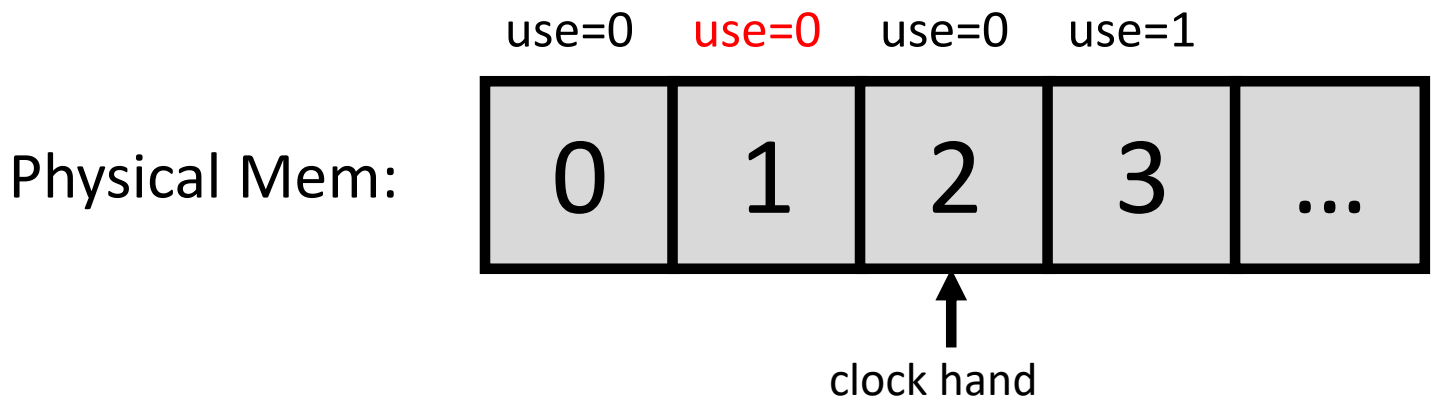
Clock: Look for a frame



Clock: Look For a Page

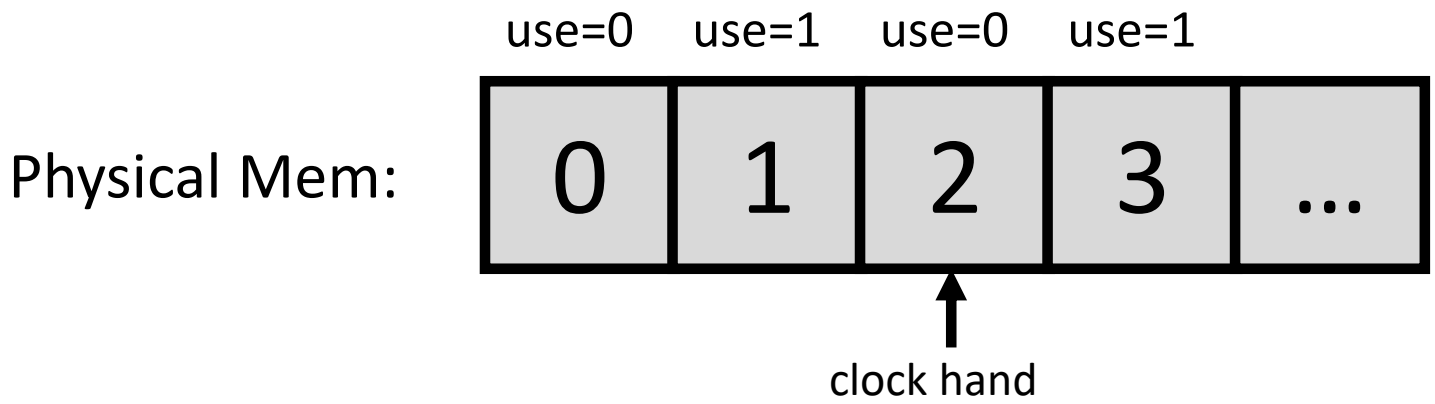


Clock: Look For a Page



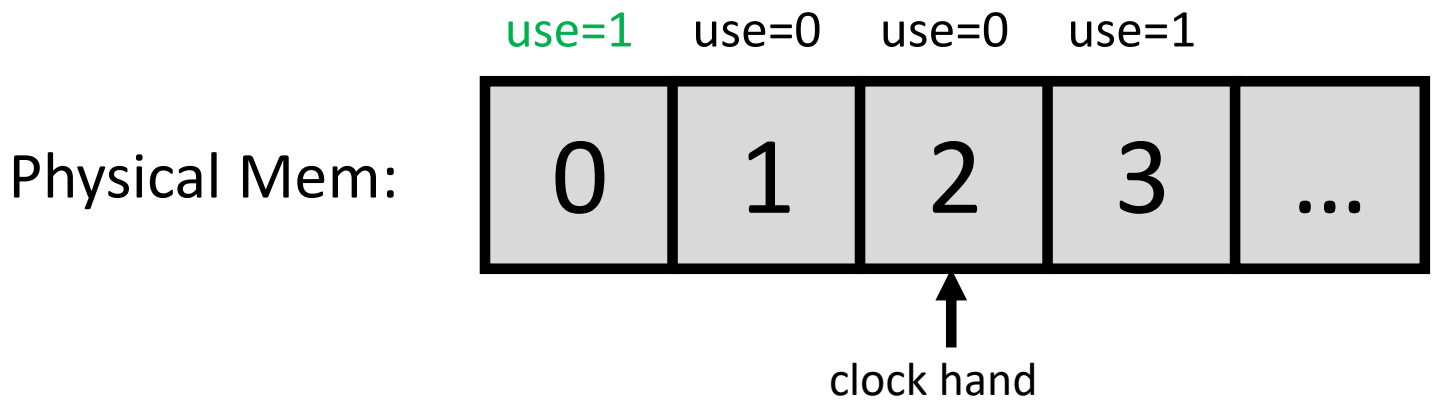
evict **frame 2** because it has not been recently used

Clock: Look For a Page



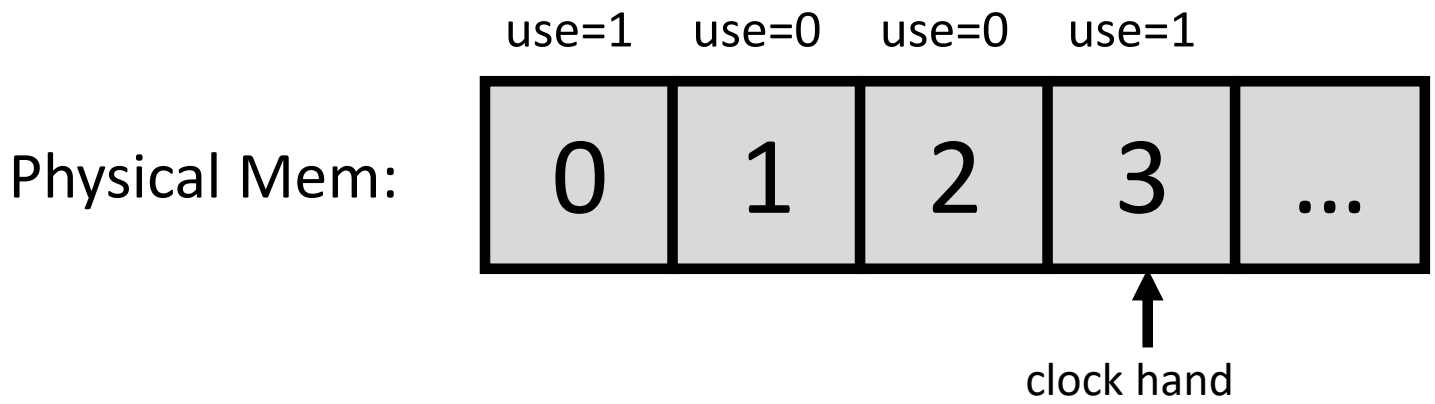
frame 0 is accessed

Clock: Look For a Page



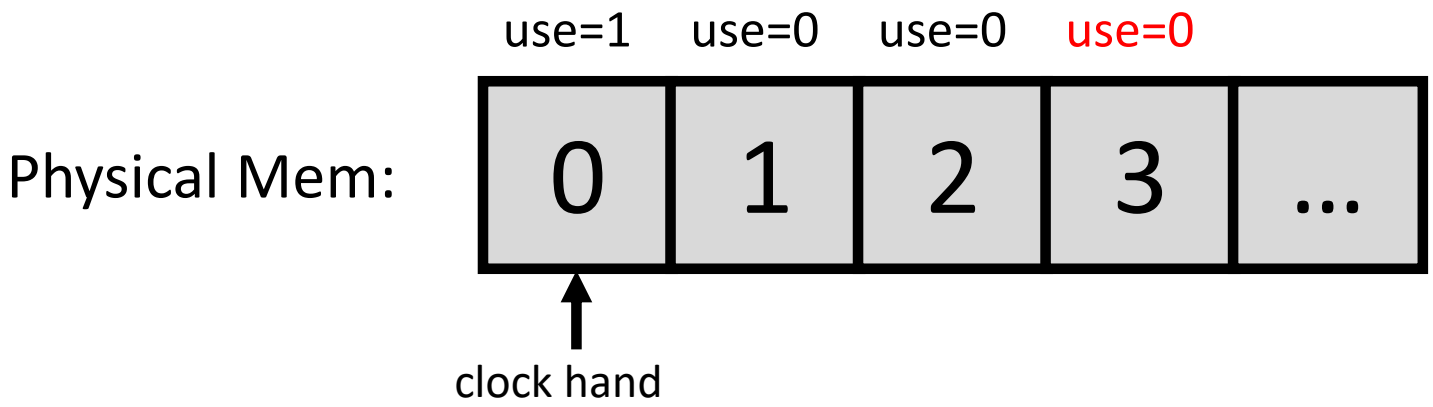
frame 0 is accessed

Clock: Look for a frame

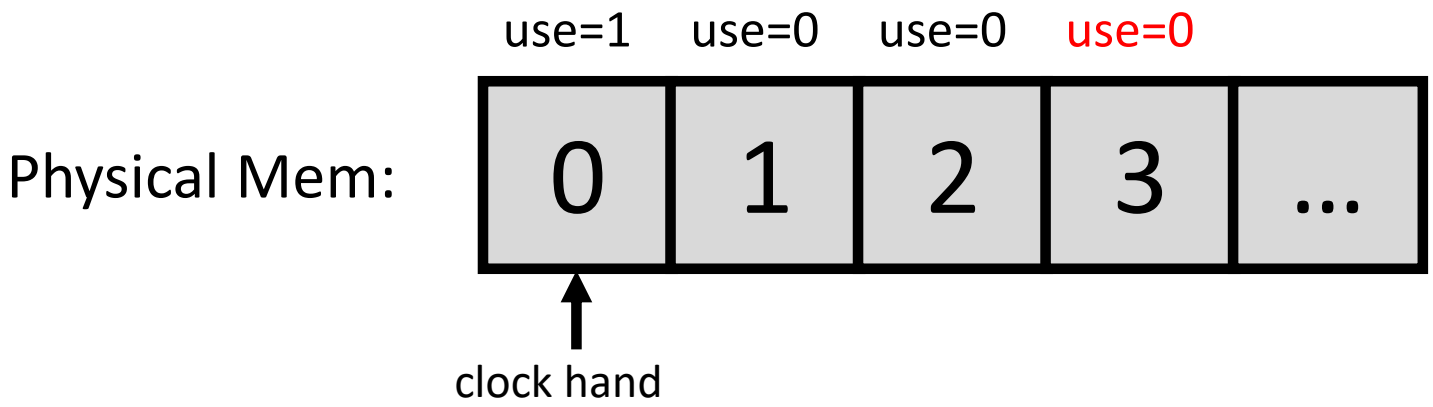


New request for a frame

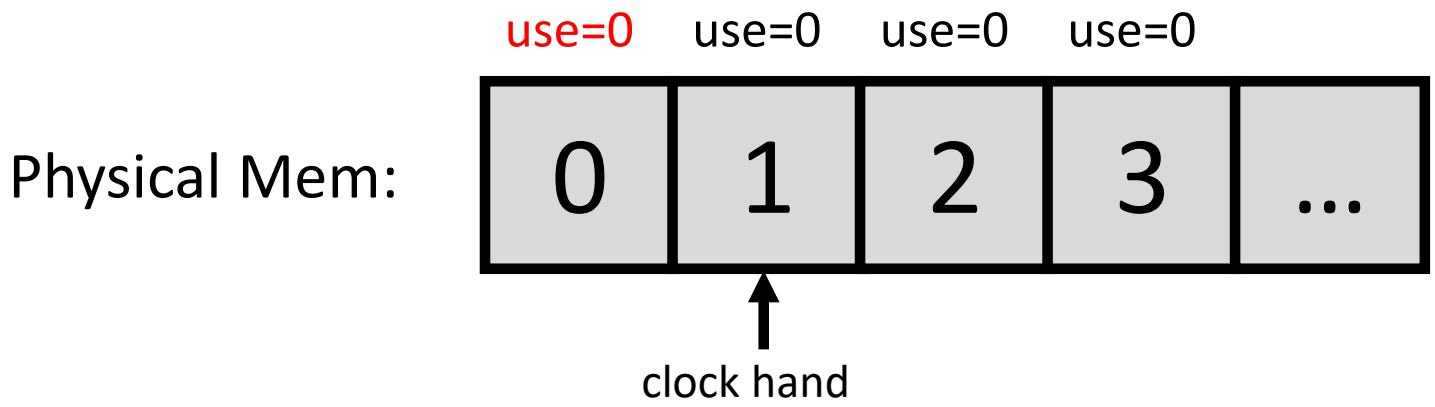
Clock: Look for a frame



Clock: Look for a frame



Clock: Look for a frame



evict **frame 1** because it has not been recently used

When to run replacement algorithm?

- Expensive to run for every page fault
- Maintain two thresholds for free frames:
  - High water mark
  - Low water mark
- When # free frames < low water mark run replacement algorithm
- Free frames till high water mark crossed

Other extensions

- Avoid replacing dirty pages if possible
  - dirty bit part of PTE.
  - set whenever a page is modified

- Pre-fetch pages

Disclaimer

- Some of the materials in this lecture slides are from the lecture slides by Prof. Arpaci, Prof. Youjip, and other educators. Thanks to all of them.