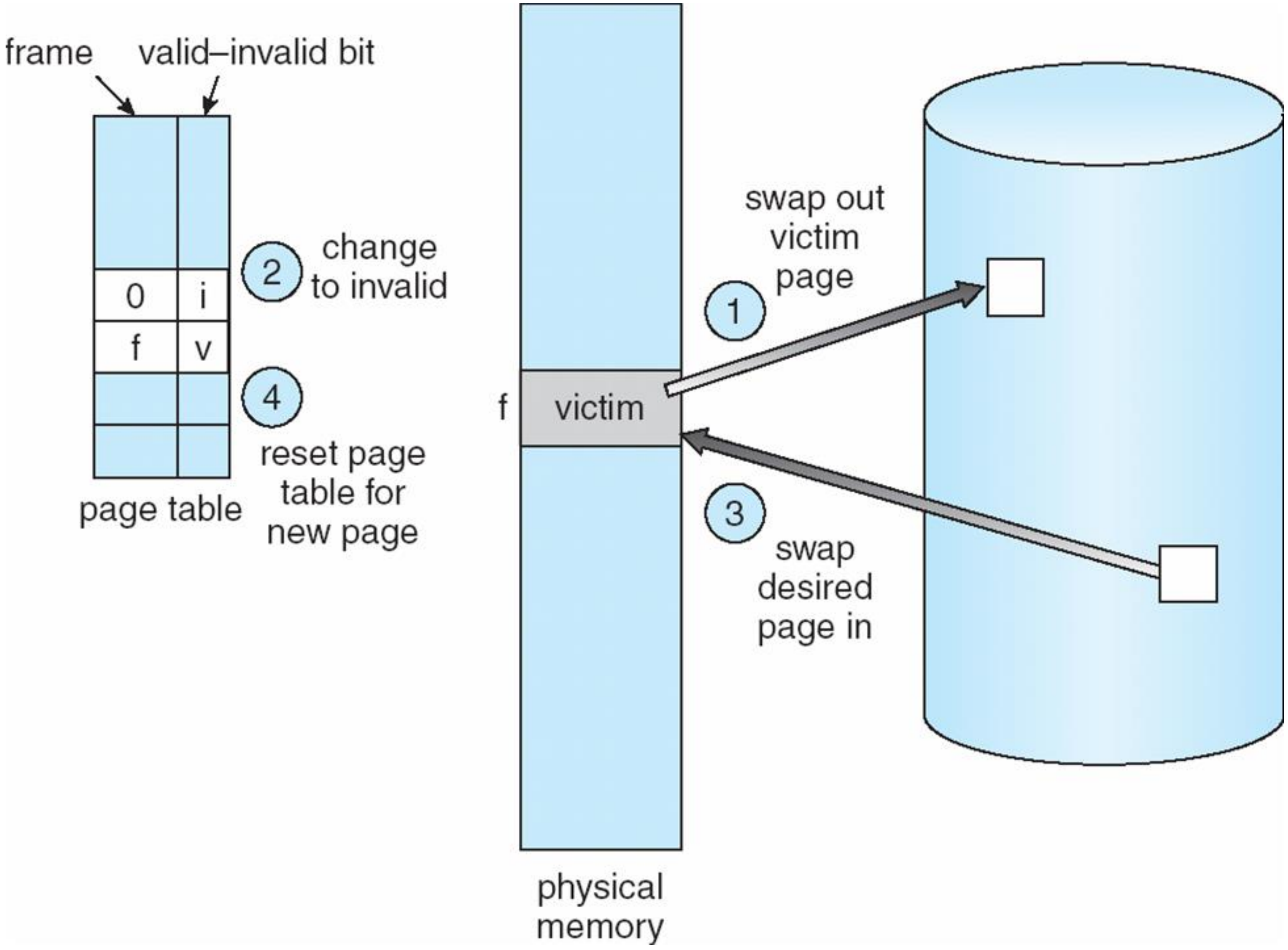
Memory Management

Disclaimer: some slides are adopted from book authors' slides with permission

Page Replacement Procedure

- On a page fault
  - Step 1: allocate a free page frame
    - If there's a free frame, use it
    - If there's no free frame, choose a **victim frame** and evict it to disk (if necessary) → **swap-out**
  - Step 2: bring the stored page on disk (if necessary)
  - Step 3: update the PTE (mapping and valid bit)
  - Step 4: restart the instruction

Page Replacement Procedure



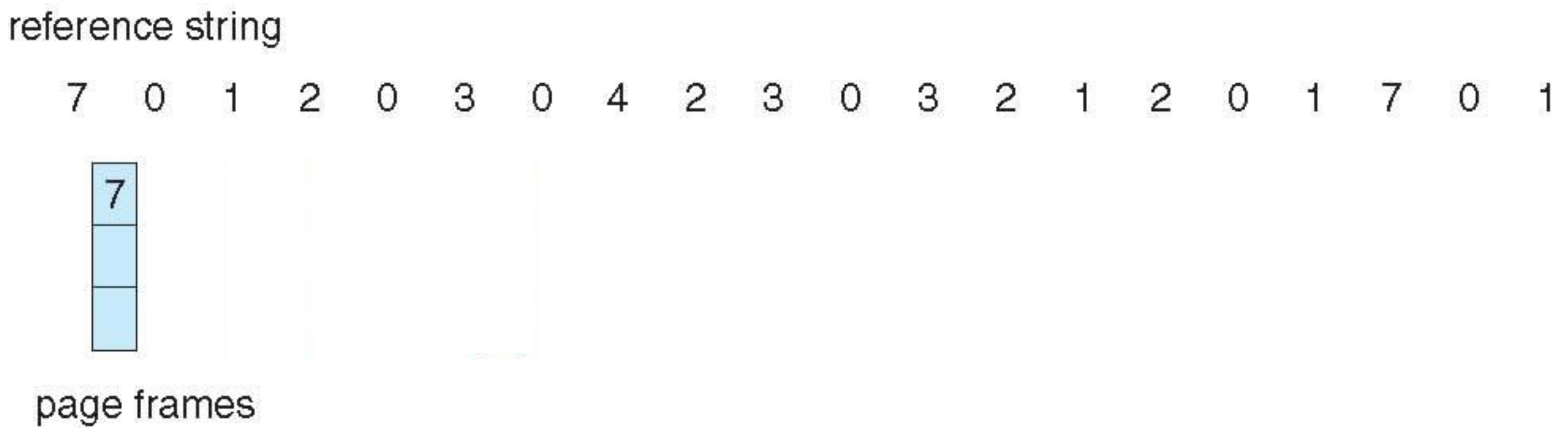
Page Replacement Policies

- FIFO (First In, First Out)
  - Evict the oldest page first.
  - Pros: simple, fair
  - Cons: can throw out frequently used pages
- Optimal
  - Evict the page that will not be used for the longest period
  - Pros: best performance
  - Cons: you need to know the future

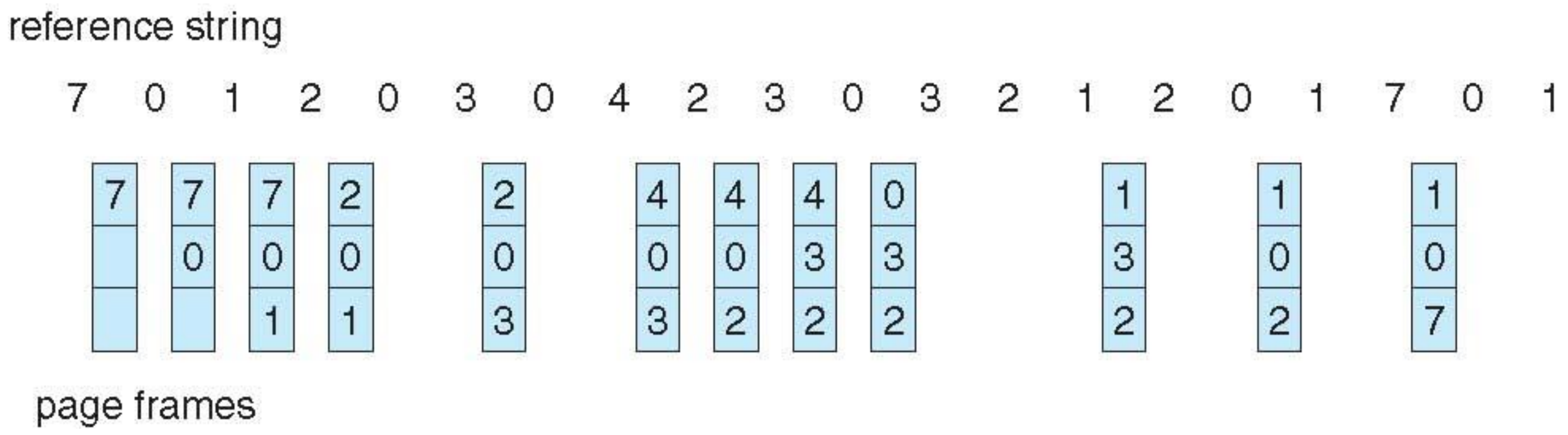
Page Replacement Policies

- Random
  - Randomly choose a page
  - Pros: simple. TLB commonly uses this method
  - Cons: unpredictable
- LRU (Least Recently Used)
  - Look at the past history, choose the one that has not been used for the longest period
  - Pros: good performance
  - Cons: complex, requires h/w support

LRU Example



LRU Example

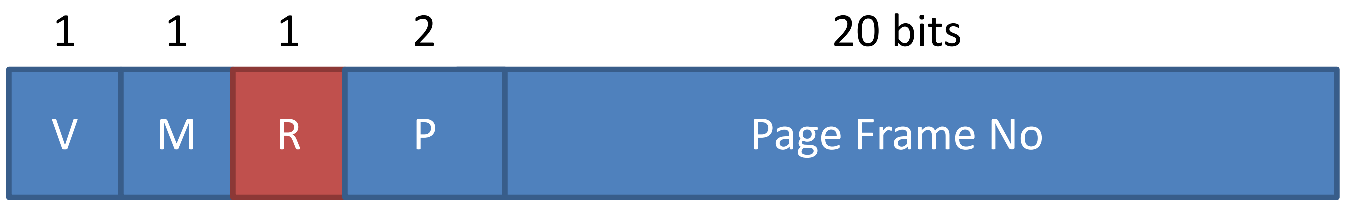

reference string
7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1
7 | 7 0 | 7 0 1 | 2 0 1 | 2 0 3 | 4 0 3 | 4 0 2 | 4 3 2 | 0 3 2 | 1 3 2 | 1 0 2 | 1 0 7
page frames

Implementing LRU

- Ideal solutions
  - Timestamp
    - Record access time of each page, and pick the page with the oldest timestamp
  - List
    - Keep a list of pages ordered by the time of reference
    - Head: recently used page, tail: least recently used page
  - Problems: very expensive (time & space & cost) to implement

Page Table Entry (PTE)

- PTE format (architecture specific)

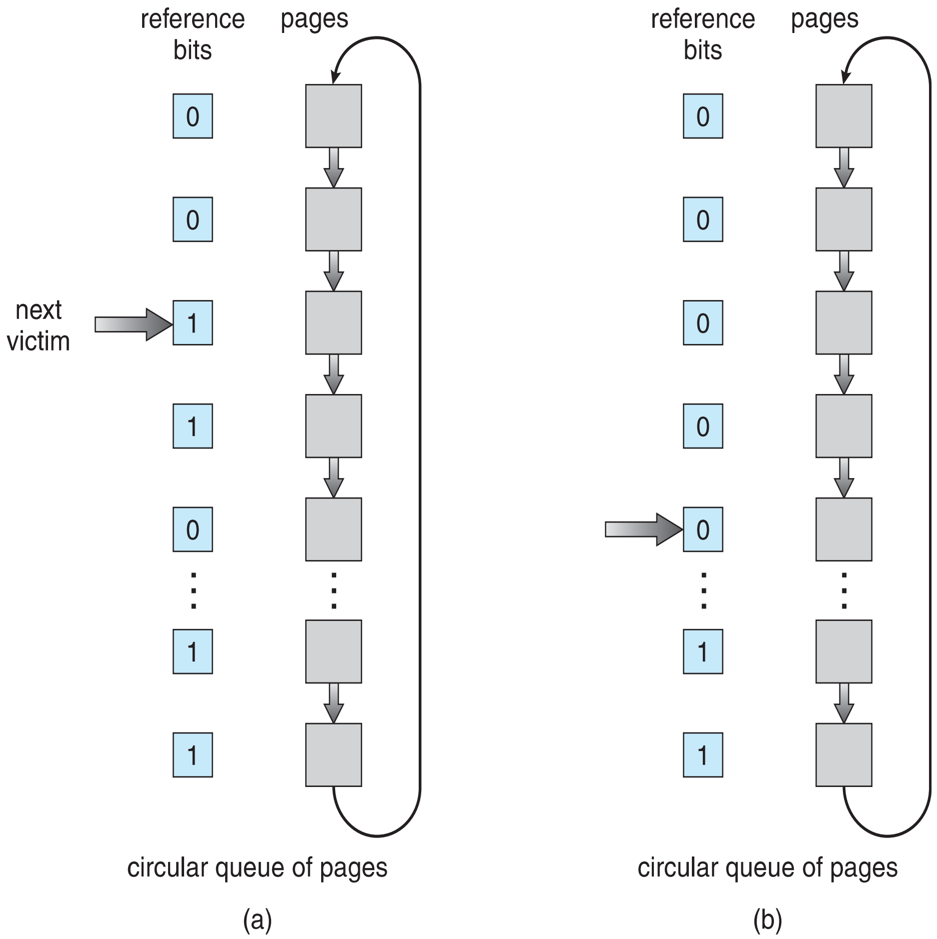


  - Valid bit (V): whether the page is in memory
  - Modify bit (M): whether the page is modified
  - **Reference bit (R): whether the page is accessed**
  - Protection bits(P): readable, writable, executable

Implementing LRU: Approximation

- Second chance algorithm (or clock algorithm)
  - Replace **an old** page, not **the oldest** page
  - Use ‘reference bit’ set by the MMU
- Algorithm details
  - Arrange physical page frames in circle with a pointer
  - On each page fault
    - Step 1: advance the pointer by one
    - Step 2: check the reference bit of the page:
      1 → Used recently. Clear the bit and go to Step 1
      0 → Not used recently. Selected victim. End.

Second Chance Algorithm



Implementing LRU: Approximation

- N chance algorithm
  - OS keeps a counter per page
  - On a page fault
    - Step 1: advance the pointer by one
    - Step 2: check the reference bit of the page: check the reference bit
      1 → reference=0; counter=0
      0 → counter++; if counter =N then found victim, otherwise repeat Step 1.
  - Large N → better approximation to LRU, but costly
  - Small N → more efficient but poor LRU approximation

Performance of Demand Paging

- Three major activities
  - Service the interrupt – hundreds of cpu cycles
  - **Read/write the page from/to disk – lots of time**
  - Restart the process – again just a small amount of time
- Page Fault Rate $0 \le p \le 1$
  - if $p = 0$ no page faults
  - if $p = 1$, every reference is a fault
- Effective Access Time (EAT)

$$
\begin{aligned}
\text{EAT} = (1 - p) \times \text{memory access} \\
+ p\,(\text{page fault overhead} \\
+ \text{swap page out} \\
+ \text{swap page in})
\end{aligned}
$$

Performance of Demand Paging

- Memory access time = 200 nanoseconds
- Average page-fault service time = 8 milliseconds
- EAT = (1 – p) x 200 + p (8 milliseconds)

  = (1 – p) x 200 + p x 8,000,000

  = 200 + p x 7,999,800
- If one access out of 1,000 causes a page fault, then

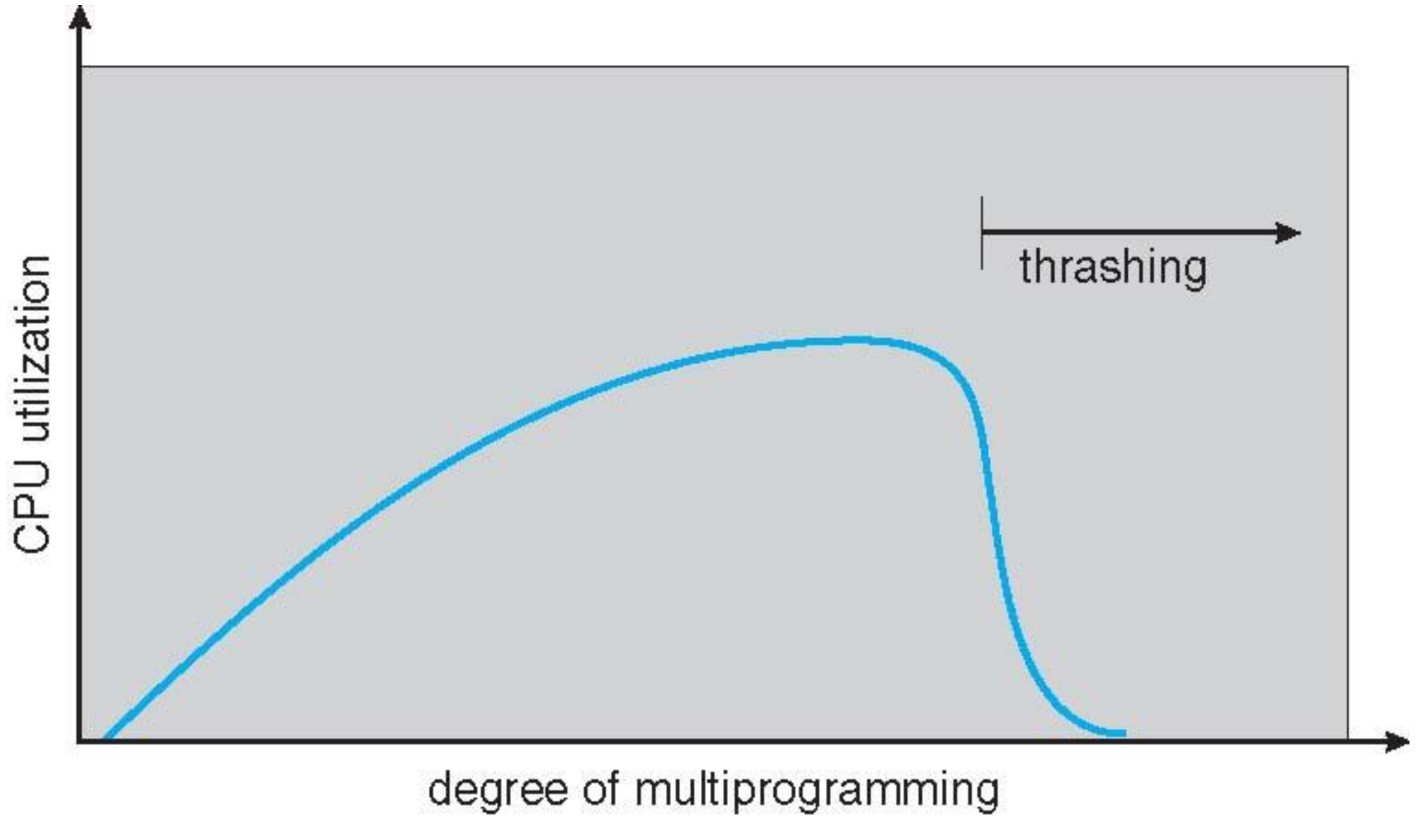
  EAT = 8.2 microseconds. → This is a slowdown by a factor of 40!!
- If want performance degradation < 10 percent
  - 220 > 200 + 7,999,800 x p

    20 > 7,999,800 x p
  - p < .0000025
  - < one page fault in every 400,000 memory accesses

Thrashing

- A process is busy swapping pages in and out
  - Don’t make much progress
  - Happens when a process do not have “enough” pages in memory
  - Very high page fault rate
  - Low CPU utilization (why?)
  - CPU utilization based admission control may bring more programs to increase the utilization → more page faults

Thrashing


thrashing
CPU utilization
degree of multiprogramming