# CS 492: Operating Systems

## *Page Replacement (2)*

*Instructor: Iraklis Tsekourakis*

Email: itsekour@stevens.edu

# Question?

- A machine has 48-bit virtual addresses and 32-bit physical addresses. Pages are 8 KB. How many entries are needed for a single-level linear page table?

# Algorithm 2: First In First Out (FIFO)

- FIFO: First in First Out
- Maintain a linked list of all pages
  - In the order that they came into memory
- Page at the beginning of list is replaced

# FIFO Example

- Assume 3 frames
- Reference string: 0, 1, 2, 3, 0, 1, 4, 0, 1, 2, 3, 4

**3 frames**

| 0 | 1 | 2 | 3 | 0 | 1 | 4 | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 0 | 1 | 4 | 4 | 4 | 2 | 3 | 3 |
| | 0 | 1 | 2 | 3 | 0 | 1 | 1 | 1 | 4 | 2 | 2 |
| | | 0 | 1 | 2 | 3 | 0 | 0 | 0 | 1 | 4 | 4 |
| P | P | P | P | P | P | P | | | P | P | |

**Tail of List**

**Beginning of List**

**9 Page faults**

# Discussions: FIFO

- Advantages
  - Simple to implement

- Disadvantages
  - Page in memory the longest may be often replaced
  - FIFO suffers from “Belady’s Anomaly”
    - The rate fault might increase when the algorithm is given more memory.

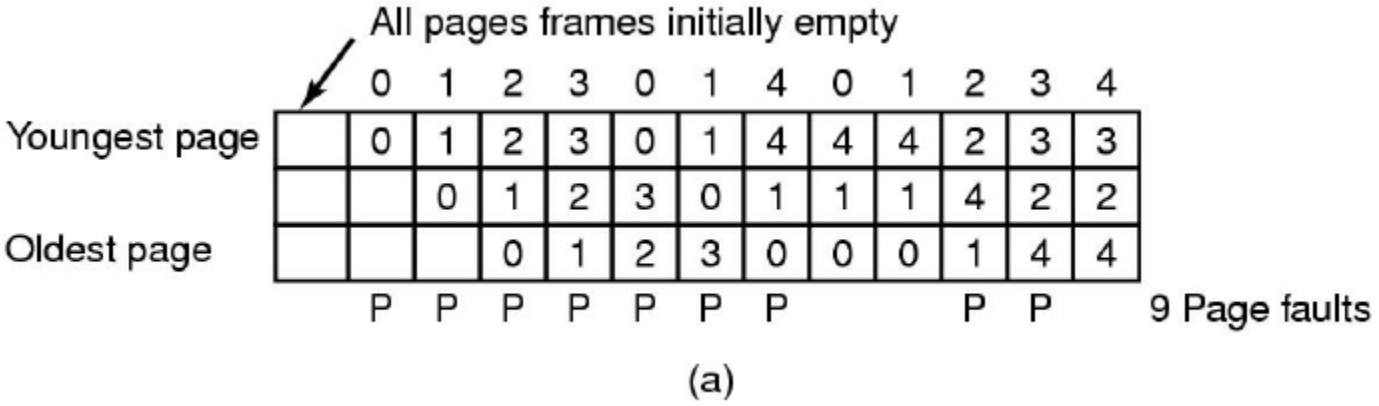
# FIFO & Belady's Anomaly

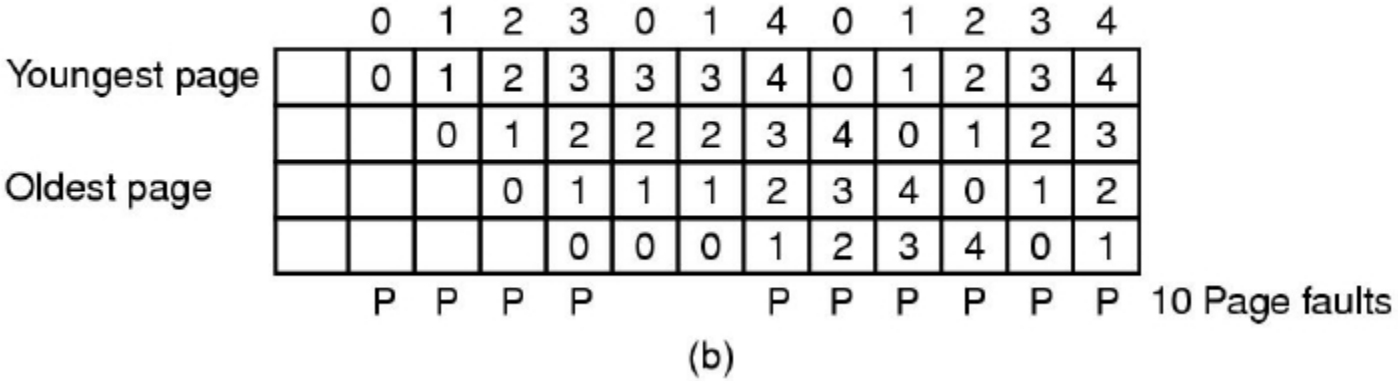
- Given a reference string: 0, 1, 2, 3, 0, 1, 4, 0, 1, 2, 3, 4
- We have shown that there are 9 page faults for the given reference string when there are 3 page frames in memory?

| 0 | 1 | 2 | 3 | 0 | 1 | 4 | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 0 | 1 | 4 | 4 | 4 | 2 | 3 | 3 |
| | 0 | 1 | 2 | 3 | 0 | 1 | 1 | 1 | 4 | 2 | 2 |
| | | 0 | 1 | 2 | 3 | 0 | 0 | 0 | 1 | 4 | 4 |
| P | P | P | P | P | P | P | | | P | P | |

- How many page faults are there if now it has **4 page frames** in memory?

# FIFO & Belady's Anomaly



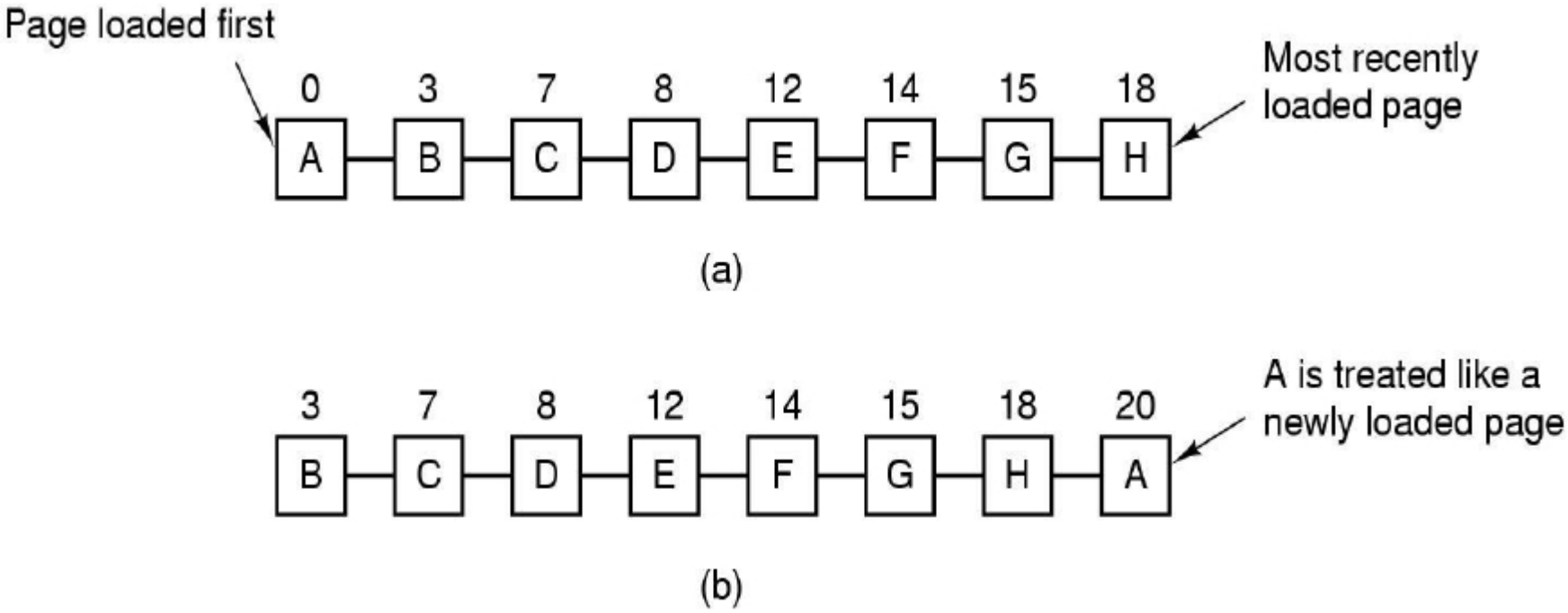




- Belady's Anomaly: more frames ⇒ more page faults

# FIFO Variant (1): Second Chance

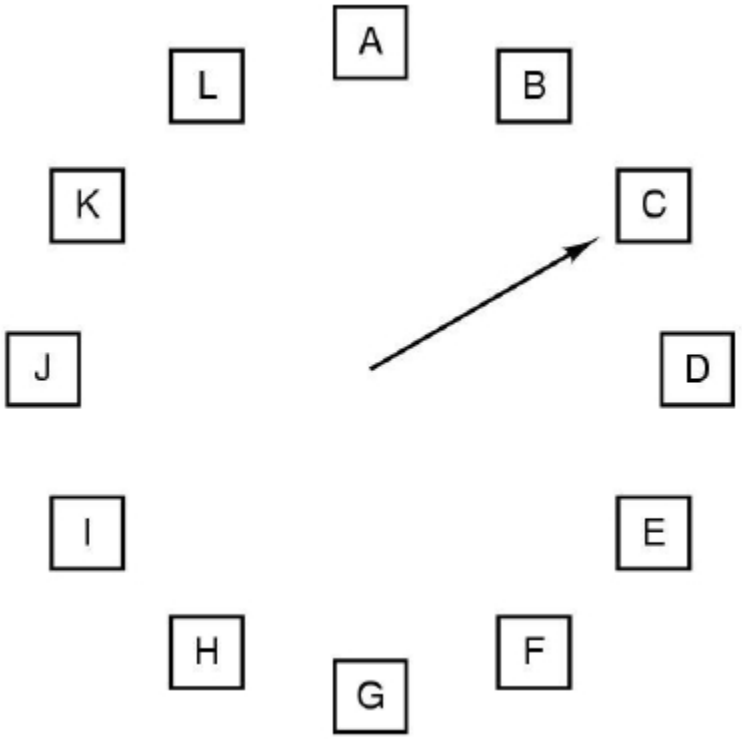
- When free page frames are needed, examine pages in FIFO order starting from beginning
  - Consider "reference bit" R
    - Reference: the page is either read or written
  - If R=0, replace page
  - If R=1, set R=0 and place at the end of FIFO list (hence, the second chance)
  - If not enough replaces on first pass, revert to pure FIFO on second pass

# Second Chance Example



# FIFO Variant (2): Clock

- Problem of the *second chance* mechanism: moving pages around on list is not efficient
- Clock: variant of FIFO, better implementation of Second Chance
- Details

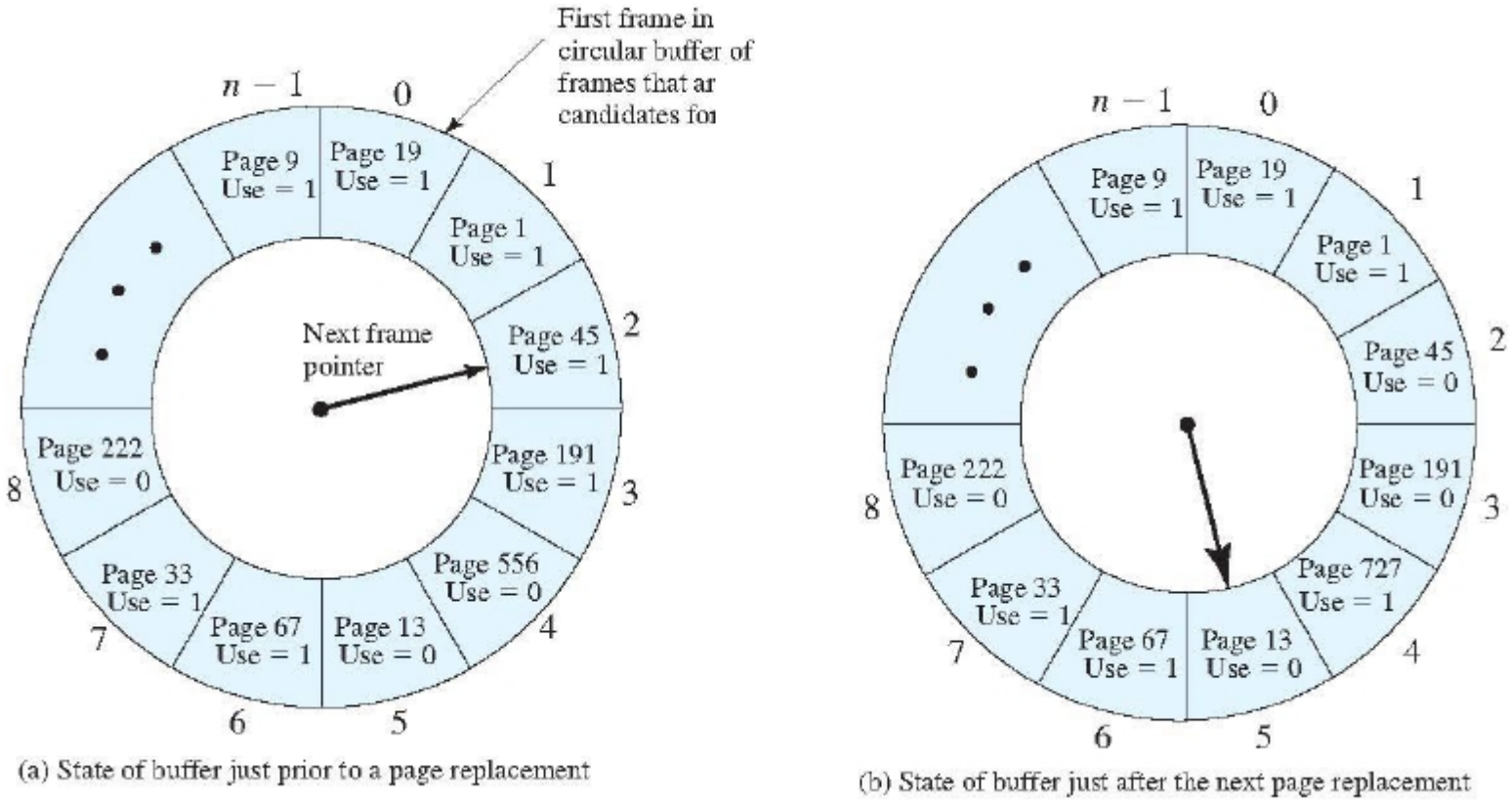



When a page fault occurs, the page the hand is pointing to is inspected. The action taken depends on the R bit:

R = 0: Evict the page

R = 1: Clear R and advance hand

# Clock Algorithm: Example




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

(b) State of buffer just after the next page replacement

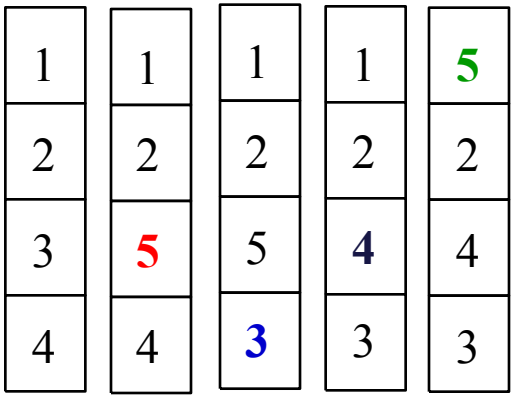
"Use" = "Reference"

# Some Observations

- Locality of reference – next reference is more likely to be near the last one
- This motivates the algorithm named Least Recently Used (LRU).

# Algorithm 3: Least Recently Used (LRU)

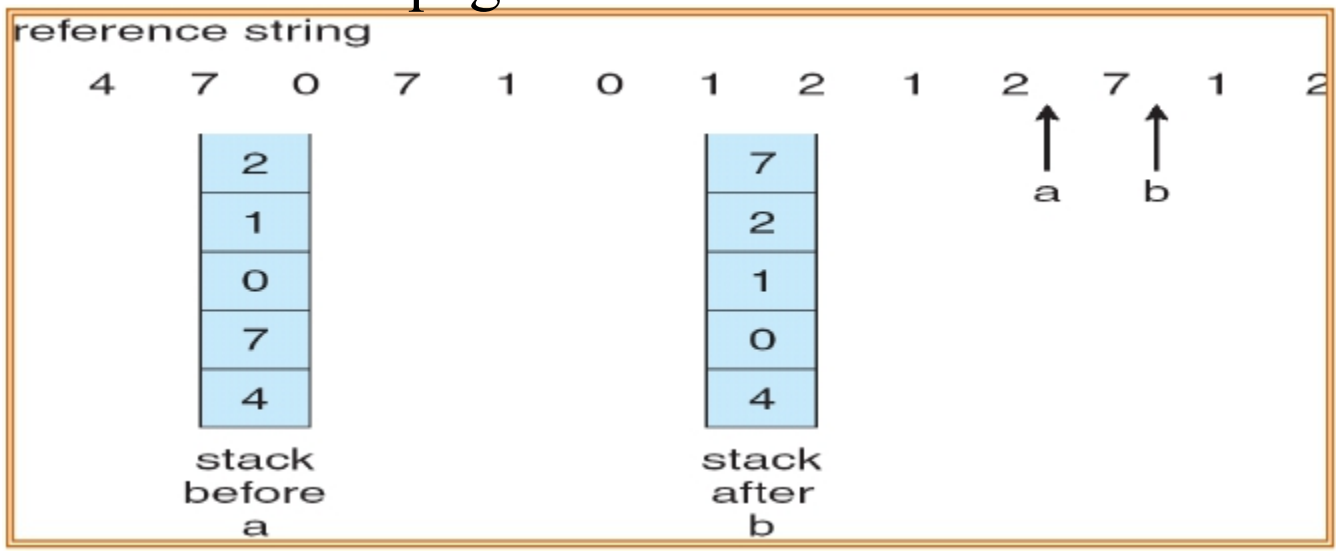
- Make use of locality property: pages used recently will be used again soon
  - Throw out page that has been unused for longest time
- Example: Reference string: 1, 2, 3, 4, 1, 2, **5**, 1, 2, **3**, **4**, **5**




- Optimal VS LRU
  - Optimal: Throw out pages the furthest in the *future*
  - LRU: Throw out pages the furthest in the *past*

# How to Implement LRU?

- Stack solution:
  - keep a stack of page numbers in a double link form:
    - Most recently used at front, least at rear
    - Always remove the page at rear
    - Referenced page is moved to the front



# Implementing LRU using Stack

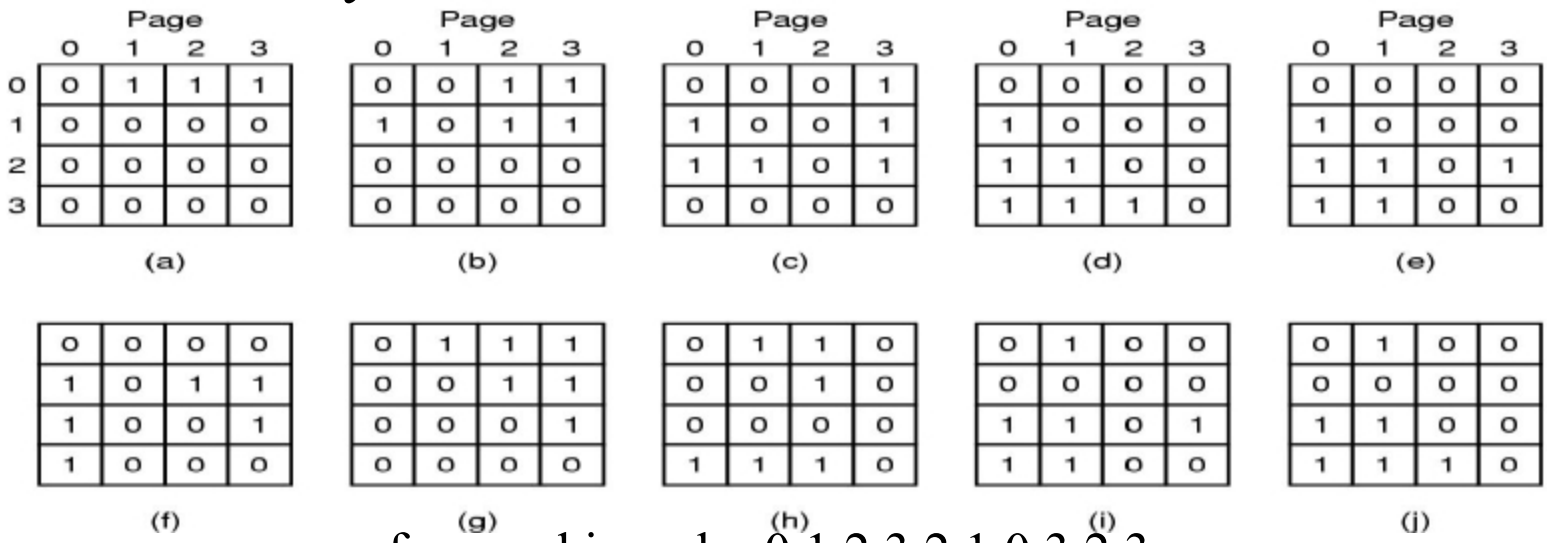
- Update this list at every memory reference!
  - Update includes: find a page in the list, move it at front
  - Time consuming!!!
  - So we need other ways to implement LRU

# Implementing LRU using Counter

- Counter implementation
  - Equip the hardware with a 64-bit counter
  - At every instruction increment the counter
  - Every page table entry must also have a field to save the counter. When a page is reference copy the counter to that field
    - The more recent the page is used, the larger the counter value is
  - For replacement, evict page with the **lowest** counter value (i.e., the oldest page)

# Implementing LRU using Matrix

- Matrix implementation – n pages, a matrix of n × n bits
- Initially all bits are 0
- When *page k* is referenced,
  - All the bits of row *k* are set to 1 (increase the binary value of page k)
  - All the bits of column *k* are set to 0 (decrease the binary value of other pages)
- Page replacement: replace page k, if row k corresponds to lowest binary value




pages referenced in order 0,1,2,3,2,1,0,3,2,3

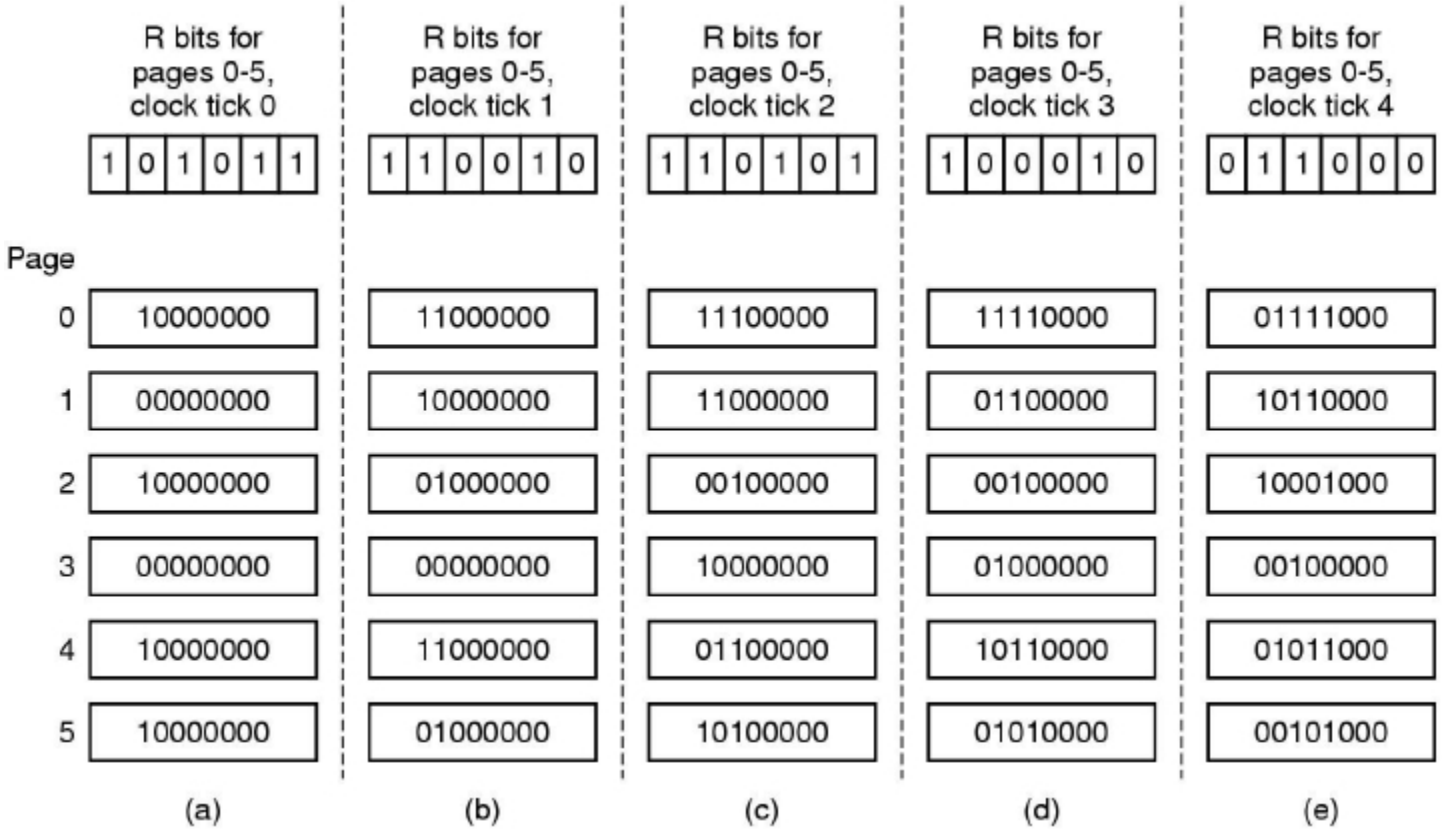
# Implementing LRU – Not Frequently Used (NFU)

- Most machines do not have the hardware to perform true LRU, but it may be simulated.
- At each clock interrupt, for each page, the R bit (valued 0 or 1) is added to its software counter.
- Page with the lowest R-bit is evicted.
  - What is the problem with this type of history keeping?

# Solution: Aging

- 1st step: shift counter to the right
  - Reduce counter values over time by dividing it by two
- 2nd step: add R to the leftmost bit
  - Recent R bit is added as most significant bit, therefore more weight given to more recent references!
- Page replacement: replace page with the lowest counter

# Aging Example




- Disadvantage
  - With 8-bit counter, have only a memory of 8 clock ticks back

# Question

- A small computer on a smart card has four page frames. At the first clock tick, the R bits are 0111 (page 0 is 0, the rest are 1). At subsequent clock ticks, the values are 1011, 1010, 1101, 0010, 1010, 1100, and 0001. If the aging algorithm is used with an 8-bit counter, give the values of the four counters after the last tick.

# Not Recently Used (NRU) Algorithm

- Each page has a Reference bit and a Modified bit
  - Reference bit: the page is accessed (read or written).
  - Modified bit: the page is written.
  - bits are set when page is referenced/modified
- Pages are classified into 4 types
  - Class 0: not referenced, not modified
  - Class 1: not referenced, modified (Not possible)
  - Class 2: referenced, not modified
  - Class 3: referenced, modified
- At a page fault, NRU removes a page from the lowest numbered non-empty class

# Discussion: NRU Algorithm

- Easy to understand and efficient to implement
- However, pages that are frequently referenced might be removed