Modern Operating Systems
Chapter 3 – Page Replacement

Zhang Yang

Spring 2022

Content of the Lecture

- 3.4 Page Replacement Algorithms
  - Paging Policies
  - Page Replacement Algorithms
    - 1. OPT
    - 2. FIFO
    - 3. NRU
    - 4. Second Chance
    - 5. Clock
    - 6. LRU
    - 7. NFU
    - 8. Aging
    - 9. Working Set
    - 10.WSClock

Paging Policies

- Fetch Policy (discussed later)
  - When should a page be brought into primary (main) memory from secondary (disk) storage?
- Placement Policy
  - When a page is brought into primary storage, where is it to be put?
- Replacement Policy
  - Which page now in primary storage is to be removed from primary storage when some other page is to be brought in and there is not enough room?

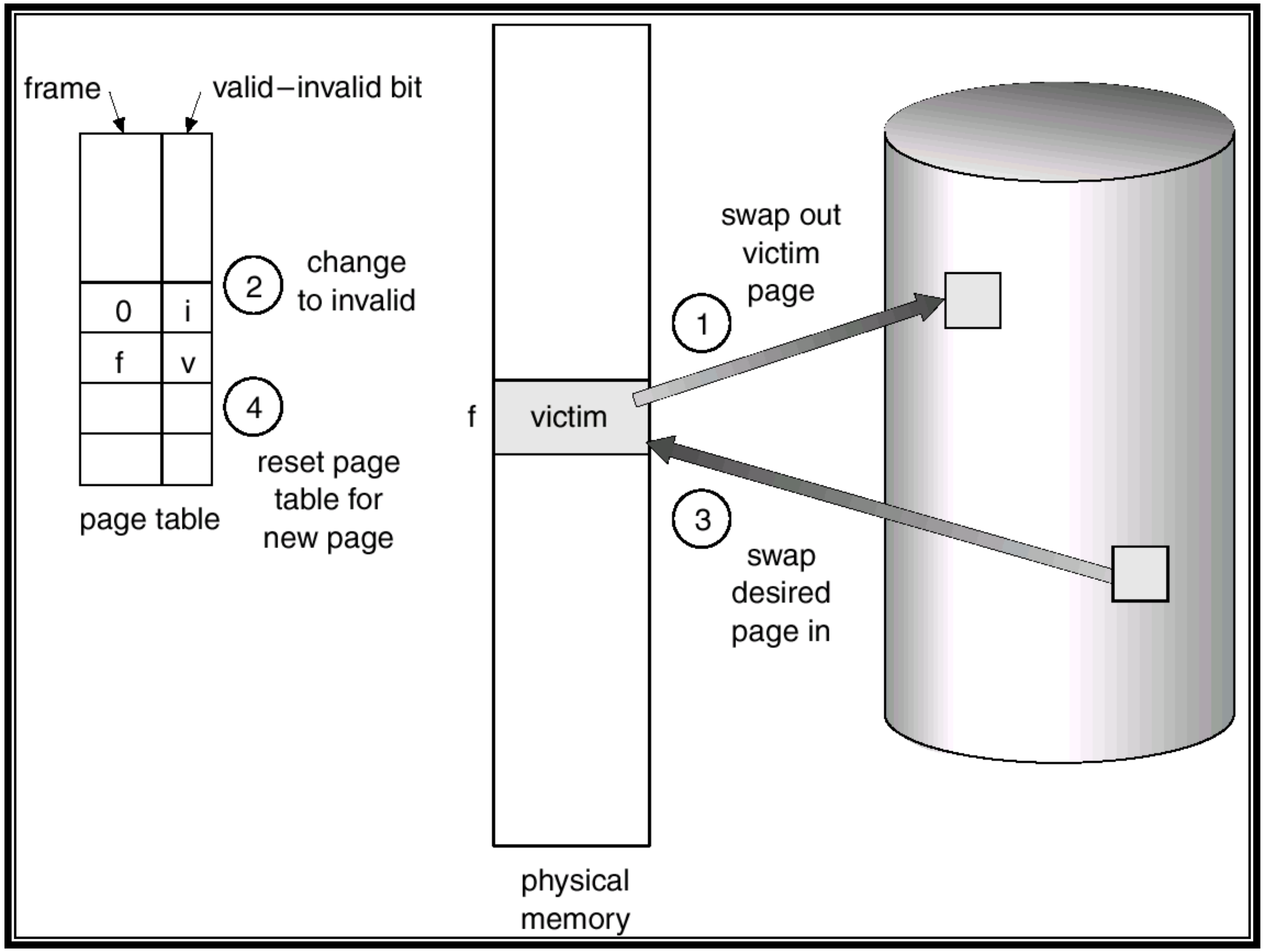
Placement Policy

- Determines where in real memory a process piece resides.
- For pure segmentation systems:
  - First-fit, next fit... are possible choices (a real issue)
- For paging (and segmentation with paging):
  - The hardware decides where to place the page: the chosen frame location is irrelevant since all memory frames are equivalent (not an issue)

Basic Page Replacement (1)

1. Find location of page on disk.
2. Find a free page frame.
   1) If there is a free page frame then use it.
   2) Otherwise, select a victim frame using the page replacement algorithm.
   3) Write the selected page to the disk and update any necessary tables.
3. Read the requested page from the disk.
4. Restart the user process.

Basic Page Replacement (2)



How to Select a Victim Frame?

- Hopefully, kick out a less-useful page.
  - Dirty pages require writing, clean pages don’t.
    - Hardware has a dirty bit for each page frame indicating this page has been updated or not.
  - Where do you write? To “swap space”.
- Problem: how do you determine utility?
  - Heuristic: temporal locality exists
  - Kick out pages that aren’t likely to be used again.

Page Replacement Algorithm (1)

- Objectives
  - Achieve a low page fault rate. (Main Objective)
    - Insure that heavily used pages stay in memory.
    - The replaced page should not be needed for some time.
  - Reduce latency of a page fault.
    - Efficient code.
    - Replace pages that do not need to be written out.

Page Replacement Algorithm (2)

- Reference String
  - Reference string is the sequence of pages being referenced.
  - Example: If user has the following sequence of addresses:
    - 123, 215, 600, 1234, 76, 96
    - If the page size is 100, then the reference string is: 1, 2, 6, 12, 0, 0

Page Replacement Algorithm (3)

- Usage of Reference String
  - Evaluate a page replacement algorithm by running it on a particular string of memory references (reference string).
  - Compute the number of page faults on that string.
- Most page replacement algorithms operate on some data structure that represents physical memory.
  - Bitmap or
  - Link list

Page Replacement Algorithms (4)

- The Optimal Algorithm (OPT or MIN)
  - Selects for replacement that page for which the time to the next reference is the longest.
- First-in First-Out (FIFO) Algorithm
  - Replace the page that has been in primary memory the longest.
- Second Chance Algorithm
  - Variant of FIFO.
- Clock Algorithm
  - Better implementation of second chance.
- NRU Algorithm
  - Enhanced second chance.

Page Replacement Algorithms (5)

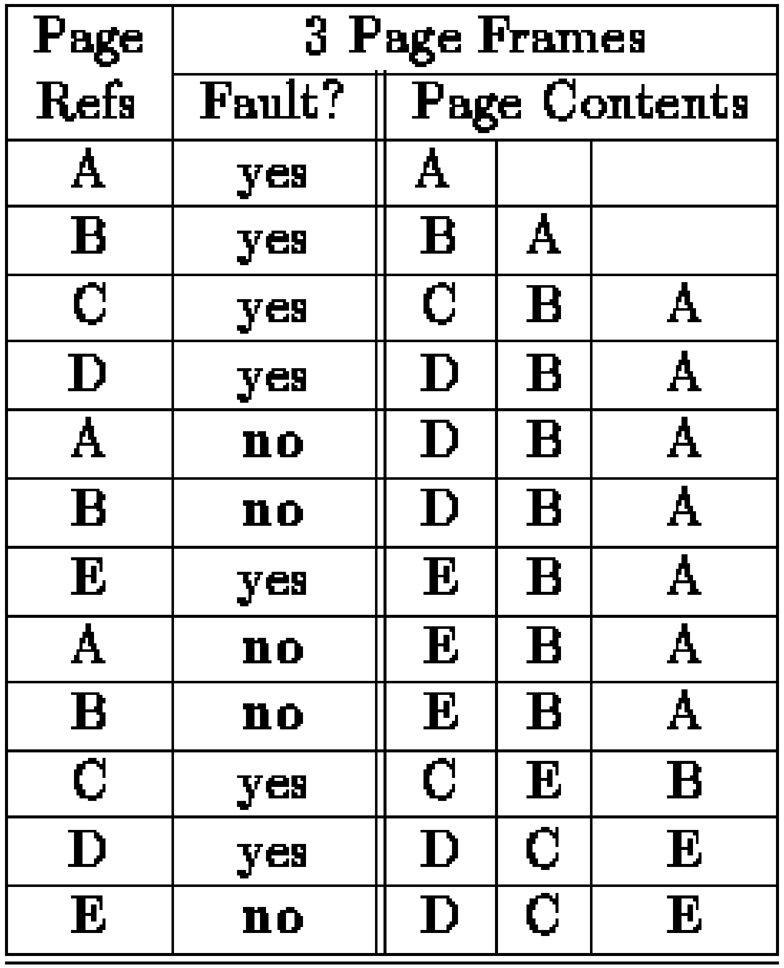
- LRU Algorithm
  - Replace the page that has not been used for the longest time.
- NFU Algorithm
  - Simulated LRU.
- Aging Algorithm
  - Modified NFU.
- Working Set Algorithm
  - Keep in memory those pages that the process is actively using.
- WS Clock Algorithm
  - The modified working set algorithm based on clock algorithm.

The Optimal Algorithm (1)

- Selects for replacement that page for which the time to the next reference is the longest.
- It can be shown that this algorithm results in the fewest number of page faults.
- This algorithm provides a basis for comparison with other schemes.
  - Often used for comparative studies, i.e., algorithm X is within Y of optimal.
- Impractical because it needs future references!

The Optimal Algorithm (2)

- Example
  - 12 references, 7 faults



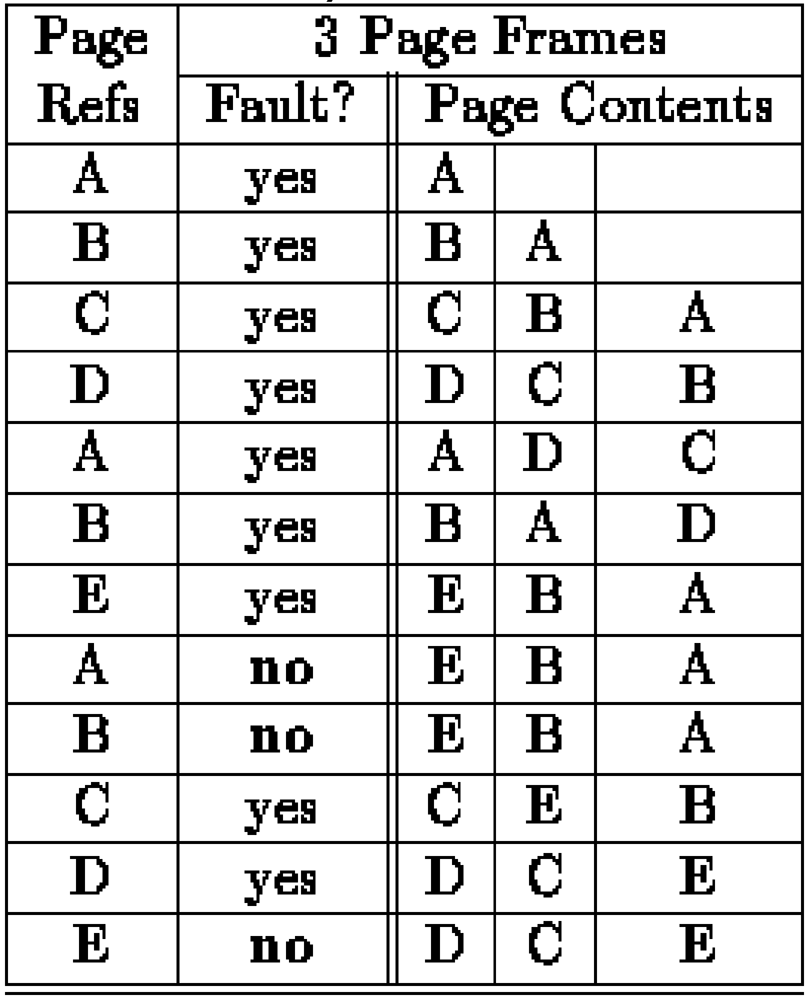
Page Refs	3 Page Frames			
	Fault?	Page Contents		
A	yes	A		
B	yes	B	A	
C	yes	C	B	A
D	yes	D	B	A
A	no	D	B	A
B	no	D	B	A
E	yes	E	B	A
A	no	E	B	A
B	no	E	B	A
C	yes	C	E	B
D	yes	D	C	E
E	no	D	C	E

First-in First-out (FIFO) (1)

- Replace the page which has been in memory for the longest time.
- Implementation (FIFO queue)
  - Maintain a list of allocated pages.
  - When the length of the list grows to cover all of physical memory, pop first page off list and allocate it.
- Why might FIFO be good?
  - Maybe the page allocated very long ago isn't being used anymore.
- Why might FIFO not be so good?
  - Doesn’t consider locality of reference!
  - Suffers from *Belady's Anomaly*: Performance of an application might get *worse* as the size of physical memory *increases!!!*

First-in First-out (2)

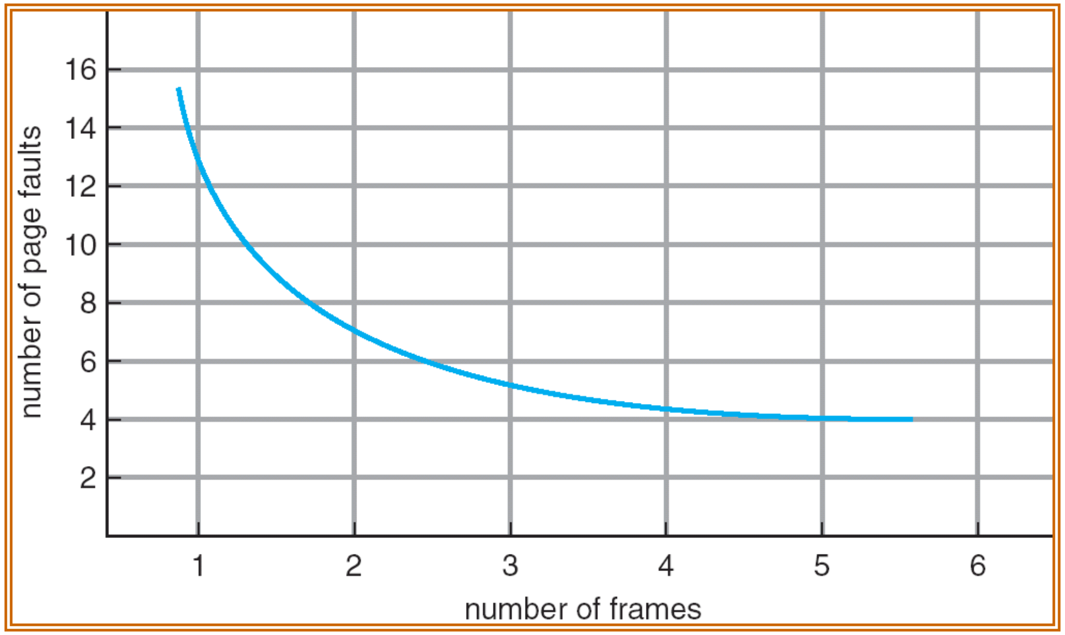
- Example
  - 12 references, 9 faults



Page	3 Page Frames			
Refs	Fault?	Page Contents		
A	yes	A		
B	yes	B	A	
C	yes	C	B	A
D	yes	D	C	B
A	yes	A	D	C
B	yes	B	A	D
E	yes	E	B	A
A	no	E	B	A
B	no	E	B	A
C	yes	C	E	B
D	yes	D	C	E
E	no	D	C	E

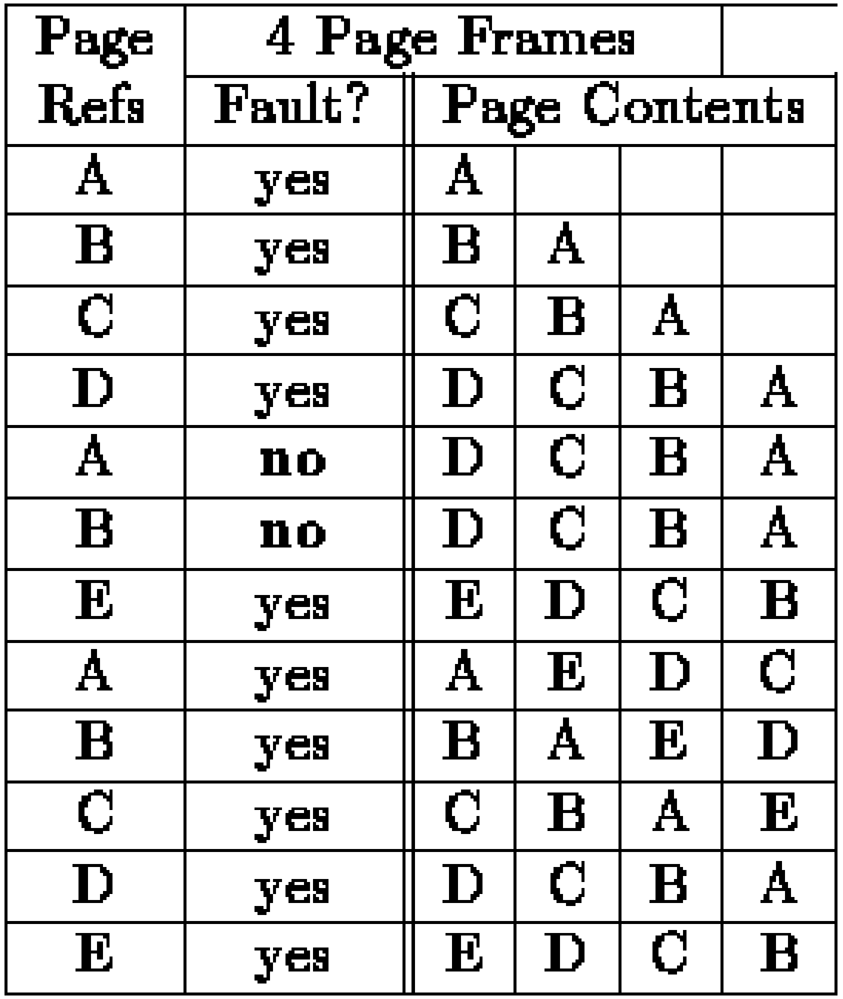
Belady's Anomaly (1)

- Belady discovered more page frames might not always have fewer page faults. This is called Belady's anomaly.
- While the normal case is as the following figure:



Belady's Anomaly (2)

- Example
  - 12 references, 10 faults.
  - As the number of page frames increase, so does the fault rate.



Page Refs	4 Page Frames				
	Fault?	Page Contents			
A	yes	A			
B	yes	B	A		
C	yes	C	B	A	
D	yes	D	C	B	A
A	no	D	C	B	A
B	no	D	C	B	A
E	yes	E	D	C	B
A	yes	A	E	D	C
B	yes	B	A	E	D
C	yes	C	B	A	E
D	yes	D	C	B	A
E	yes	E	D	C	B

The Second Chance Algorithm(1)

- Variant of FIFO.
- Only use one referenced bit in the page table entry.
  - R=0, Initially
  - R=1, When a page is referenced.
- Choose "victim" to evict.
  - Select head of FIFO.
  - If the page has reference bit set, then
    - It's reference bit is cleared (R=0), and its arrival time is reset to the current time.
    - It is put onto the end of the list of pages.
  - Select next page in FIFO list.
  - Keep processing until reach page with zero referenced bit and page that one out.

What if all pages have R=1?

The Second Chance Algorithm(2)

- Example

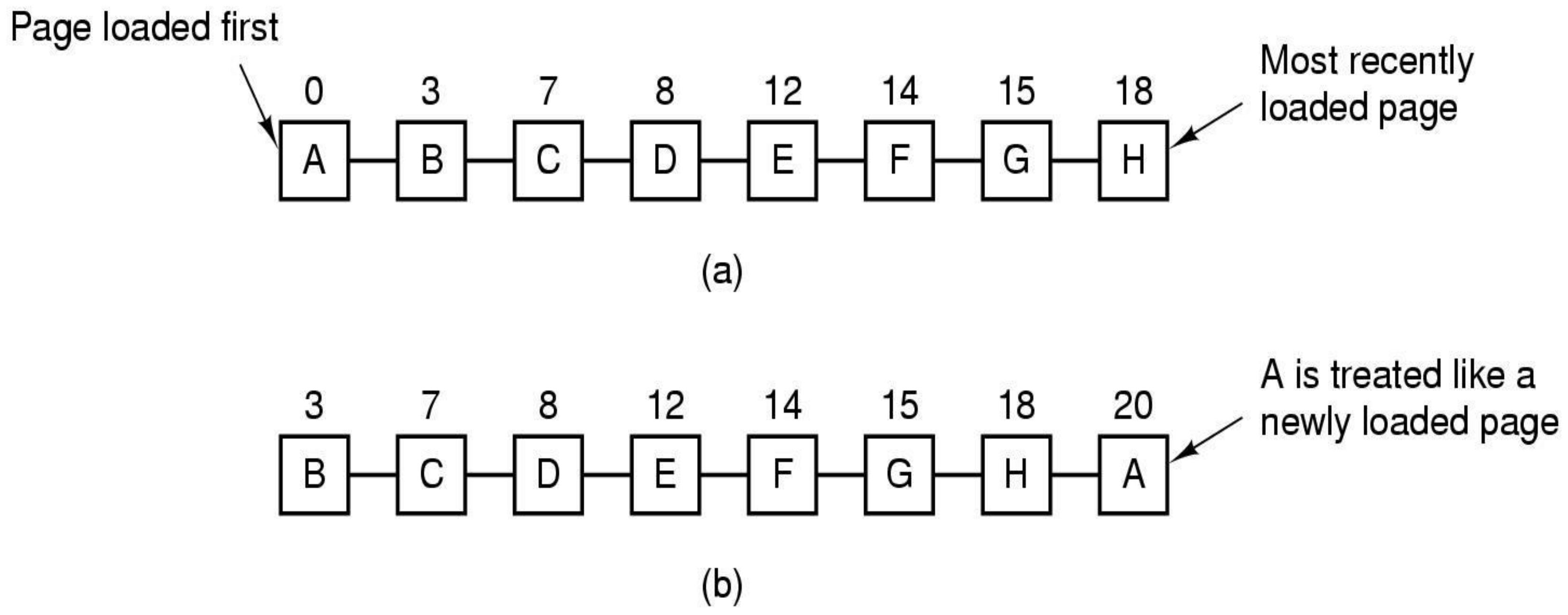
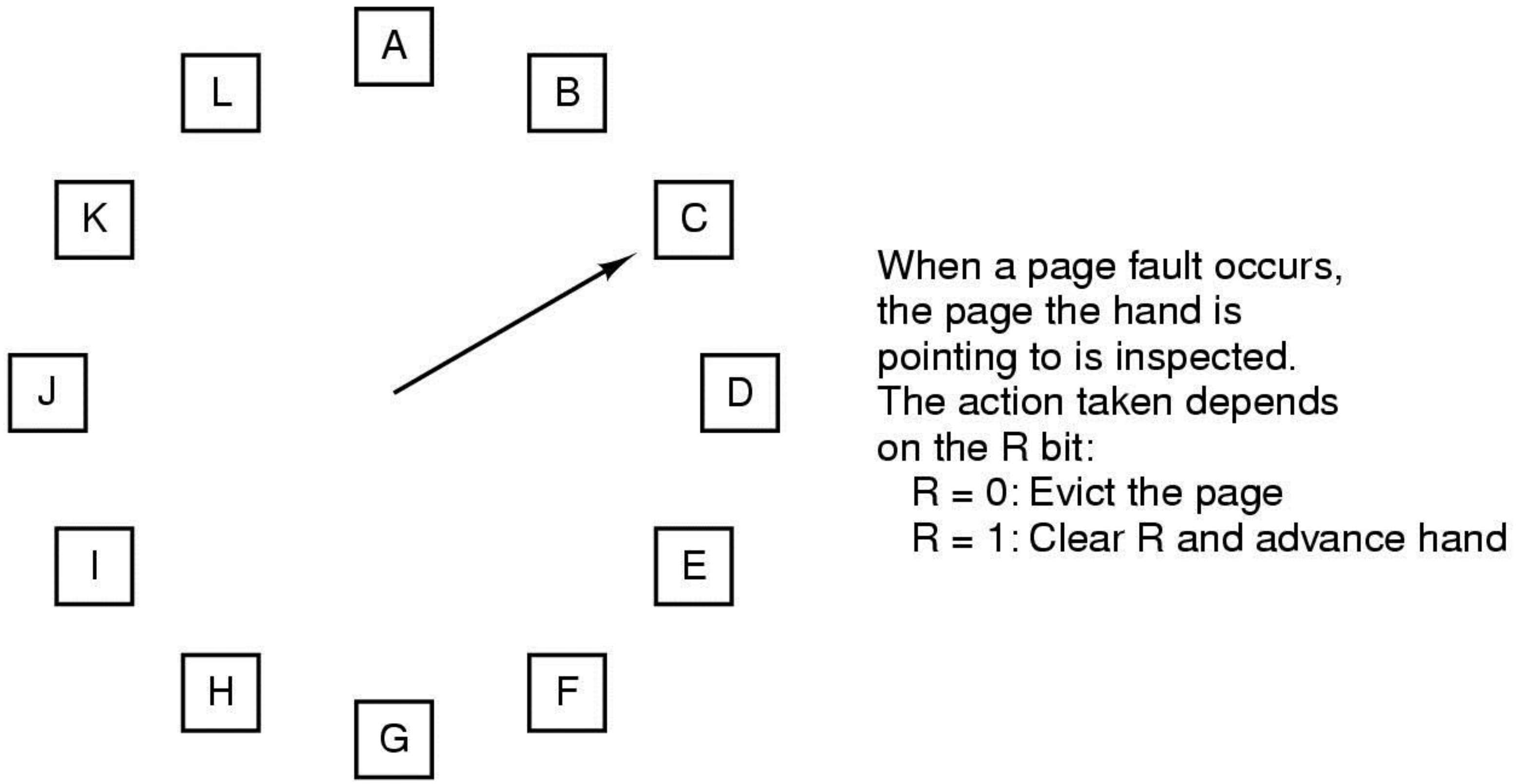


Figure 3-15. Operation of second chance.
(a) Pages sorted in FIFO order.
(b) Page list if a page fault occurs at time 20 and A has its R bit set. The numbers above the pages are their load times.

The Clock Algorithm (1)

- Better implementation of second chance.
- Basic Idea

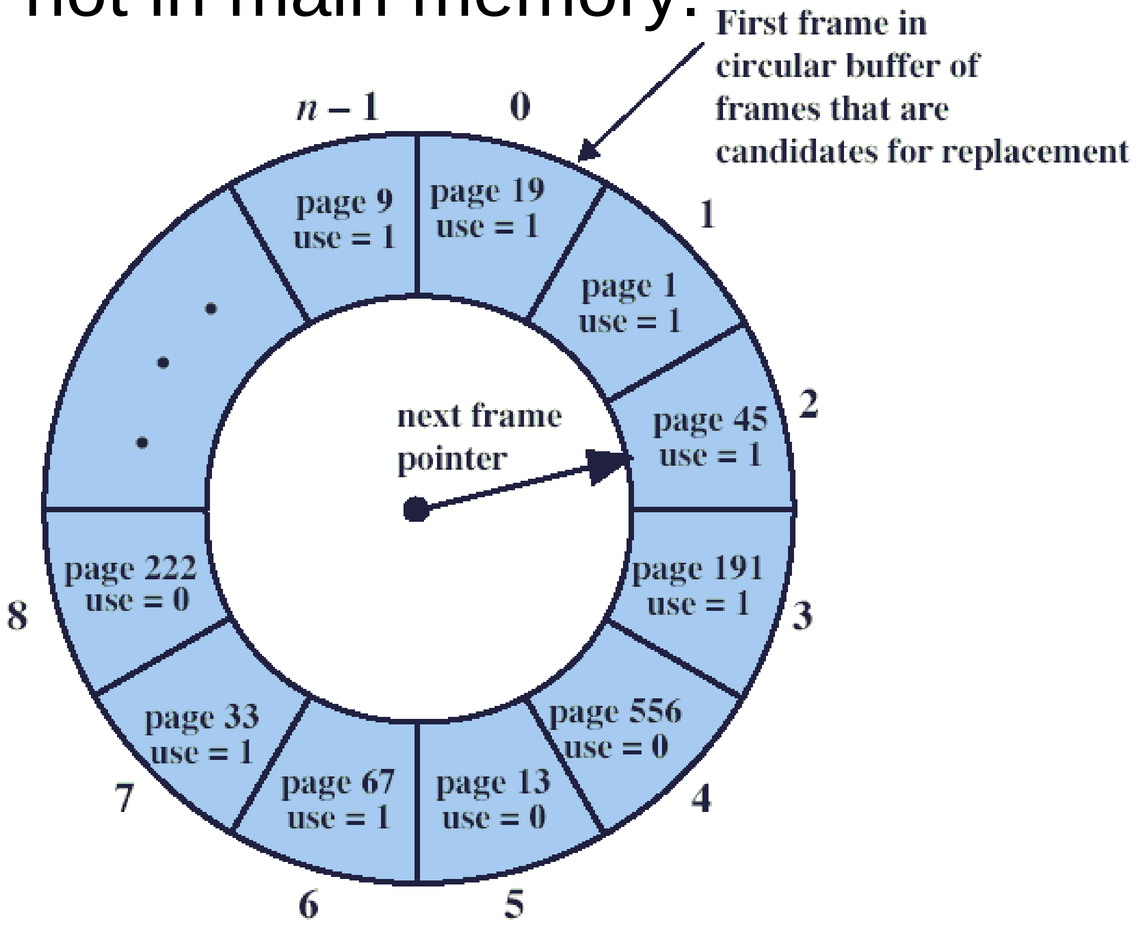


When a page fault occurs, the page the hand is pointing to is inspected. The action taken depends on the R bit:
R = 0: Evict the page
R = 1: Clear R and advance hand

Figure 3-16. The clock replacement algorithm

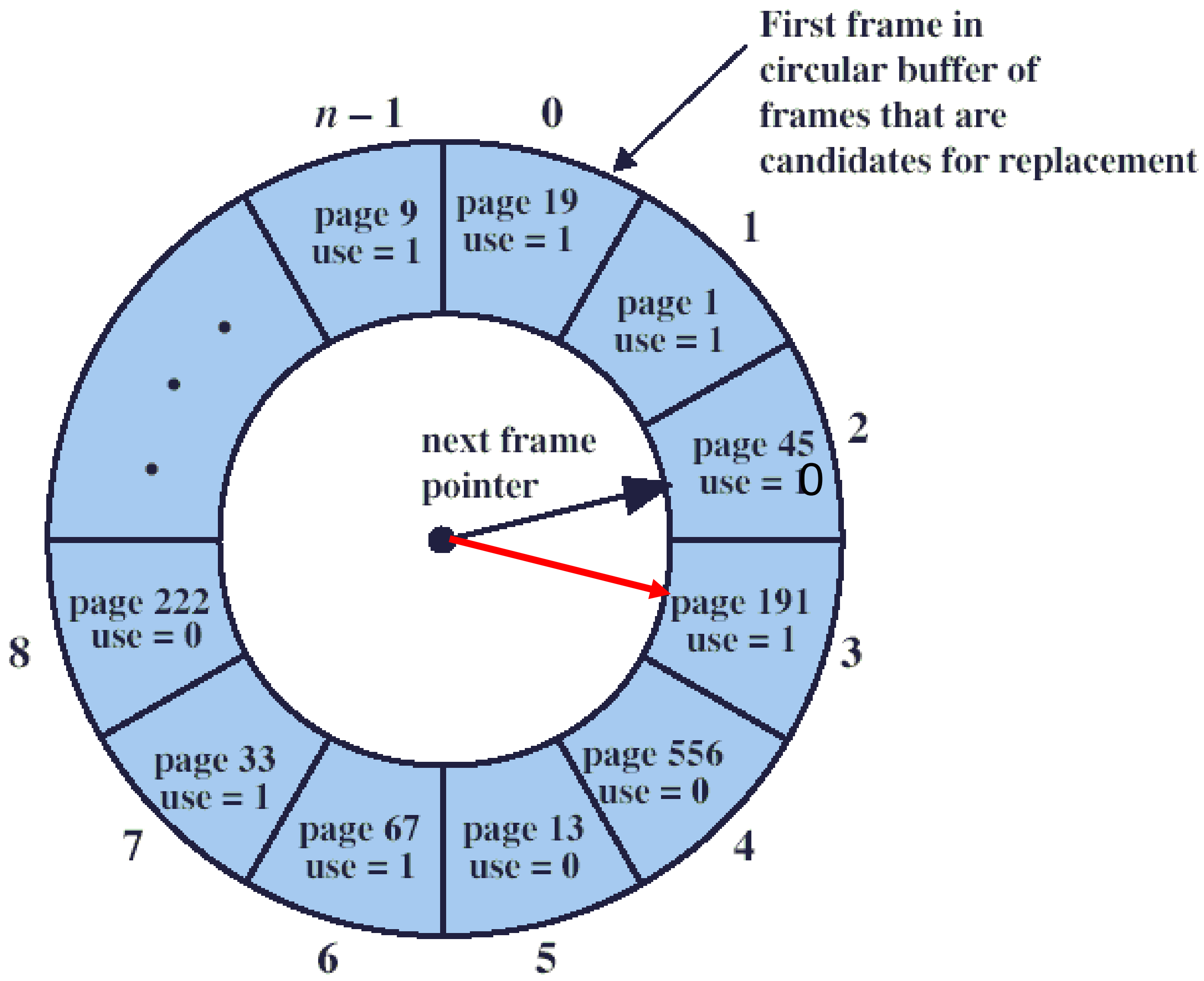
The Clock Algorithm (2)

- Example
  - A page fault occurs, because page 727 is currently not in main memory.



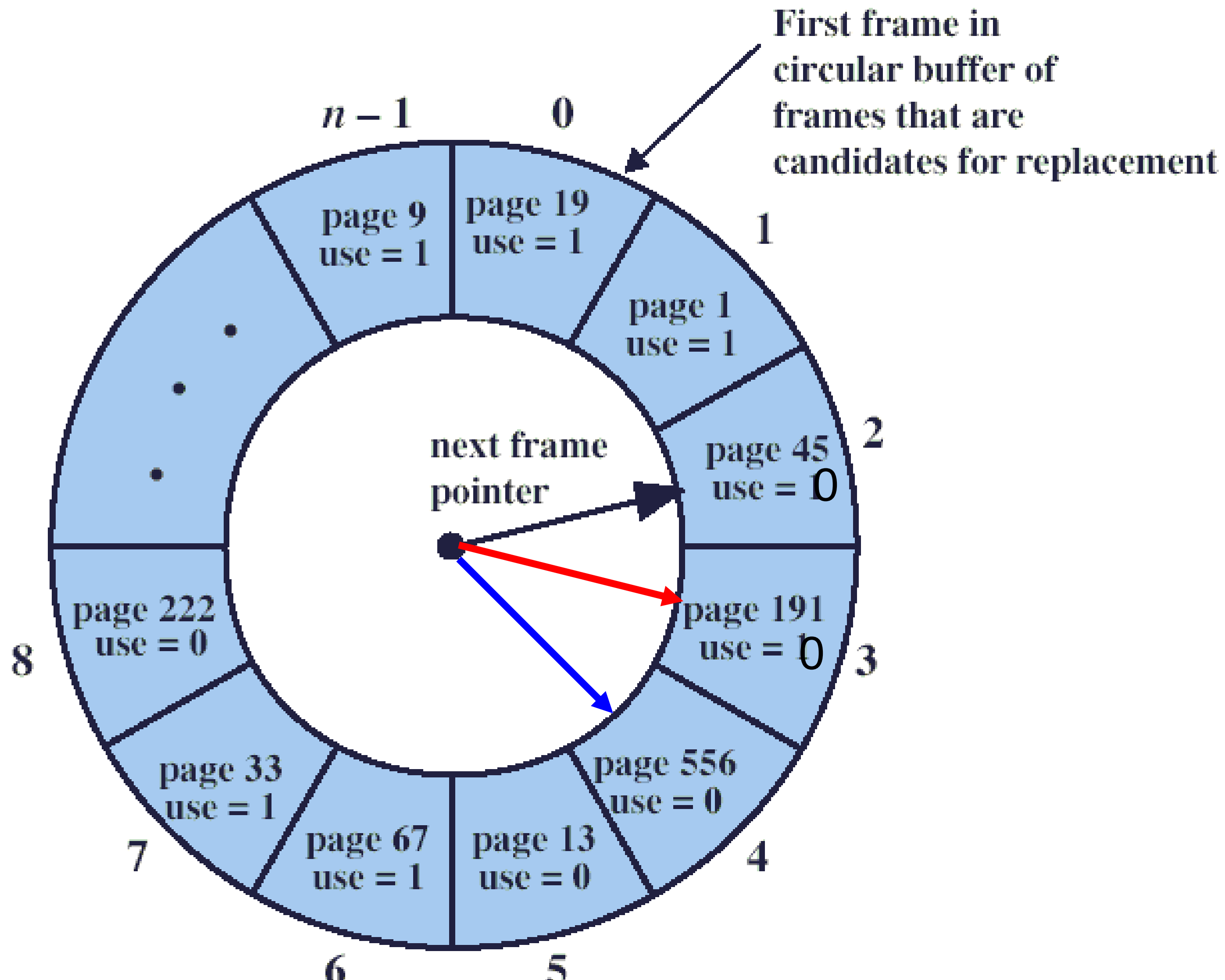
The Clock Algorithm (3)

- Example (ctd.)



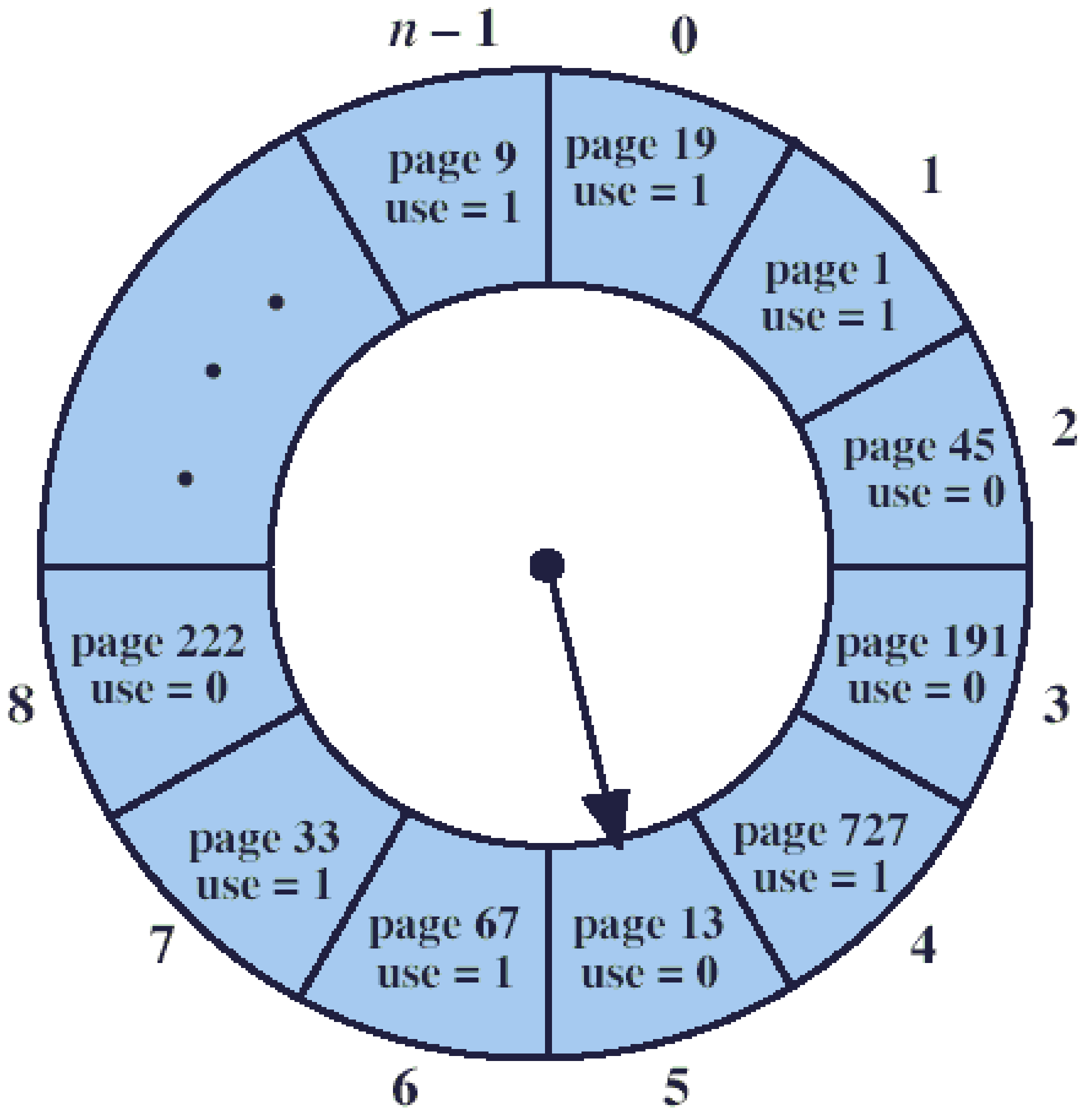
The Clock Algorithm (4)

- Example (ctd.)



The Clock Algorithm (5)

- Example (ctd.)



Not-Recently-Used (NRU) (1)

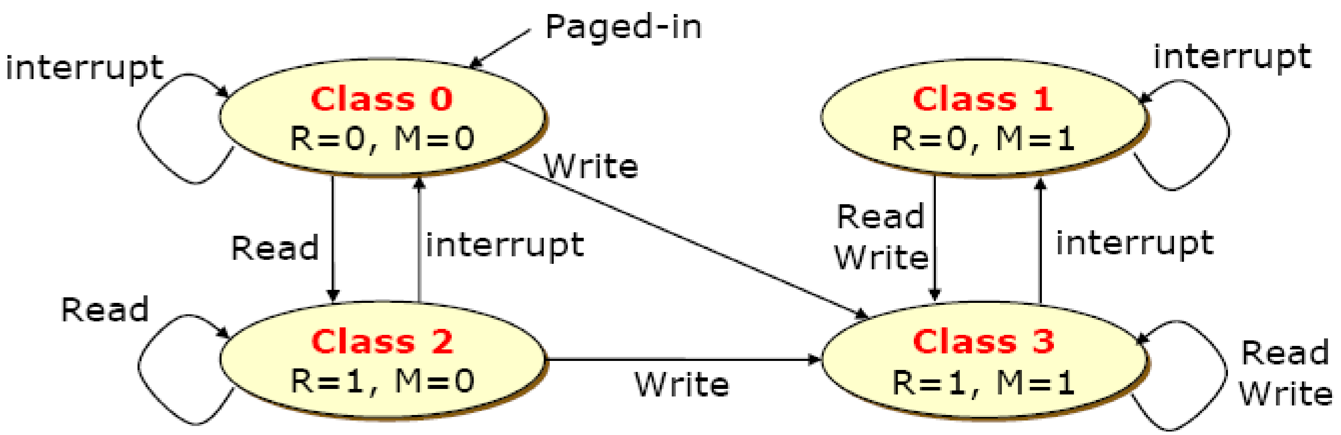
- Enhanced second chance.
- Replace the page which is not used recently.
- Use the referenced (R) & modified (M) bits in the page table entry.
  - R bit is set when the page is referenced (read or written).
  - M bit is set when the page is modified (contents changed - written).

Not-Recently-Used (NRU) (2)

- Initially, all pages have
  - Referenced Bit = 0
  - Modified Bit = 0
- Periodically (e.g. whenever a timer interrupt occurs) clear the referenced bit.
  - To distinguish pages that have been referenced recently.
- Note
  - Can't clear modified bit!!!
    - Needed to indicate which pages need to be flushed to disk.

Not-Recently-Used (NRU) (3)

- Categorize each page into a class:
  - Class 0: Referenced = 0 Modified = 0
  - Class 1: Referenced = 0 Modified = 1
  - Class 2: Referenced = 1 Modified = 0
  - Class 3: Referenced = 1 Modified = 1



Not-Recently-Used (NRU) (4)

- Algorithm: remove a page from the lowest non-empty class.
  - Select a page at random from that class.
  - Randomly choose a victim page from class 0 … why?
  - If none, randomly choose a page from class 1 … why?
  - If none, randomly choose a page from class 2 … why?
  - If none, randomly choose a page from class 3 … why?

Not-Recently-Used (NRU) (5)

- Implementation
  - Hardware
    - R & M bits are set by hardware on every memory reference.
  - Software
    - R & M bits in the page table entry is modified at page faults.
- Advantages
  - Easy to understand and implement.
  - Performance adequate (though not optimal).

Least Recently Used (LRU) (1)

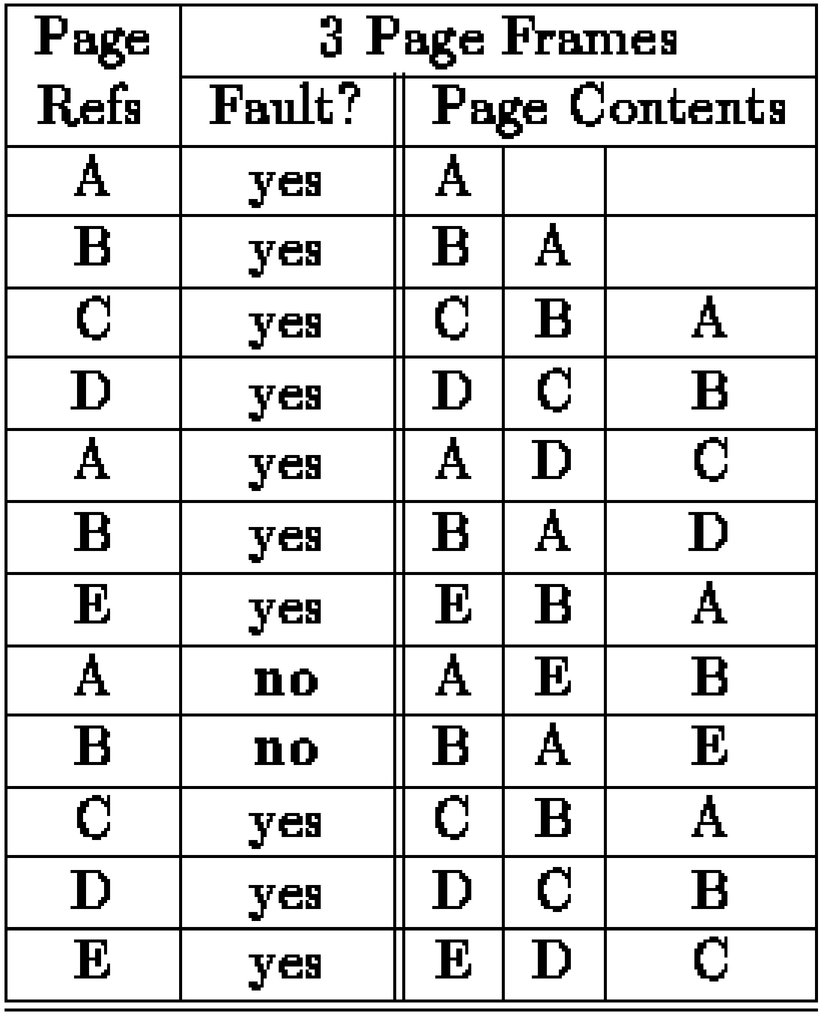
- Approximate OPT under certain assumptions.
- LRU associates with each page the time of that page’s last use.
- Based on principal of “Locality of Reference”.
  - A page that has been used in the near past is likely to be used in the near future.
- Replace the page which has been unused for the longest time.
  - Keep track of when pages are referenced to make a better decision.
  - Use past behaviour to predict future behaviour.
    - LRU uses past information, while OPT uses future information.

Least Recently Used (LRU) (2)

- Example
  - 12 references, 10 faults
- Question: When does

LRU work well,

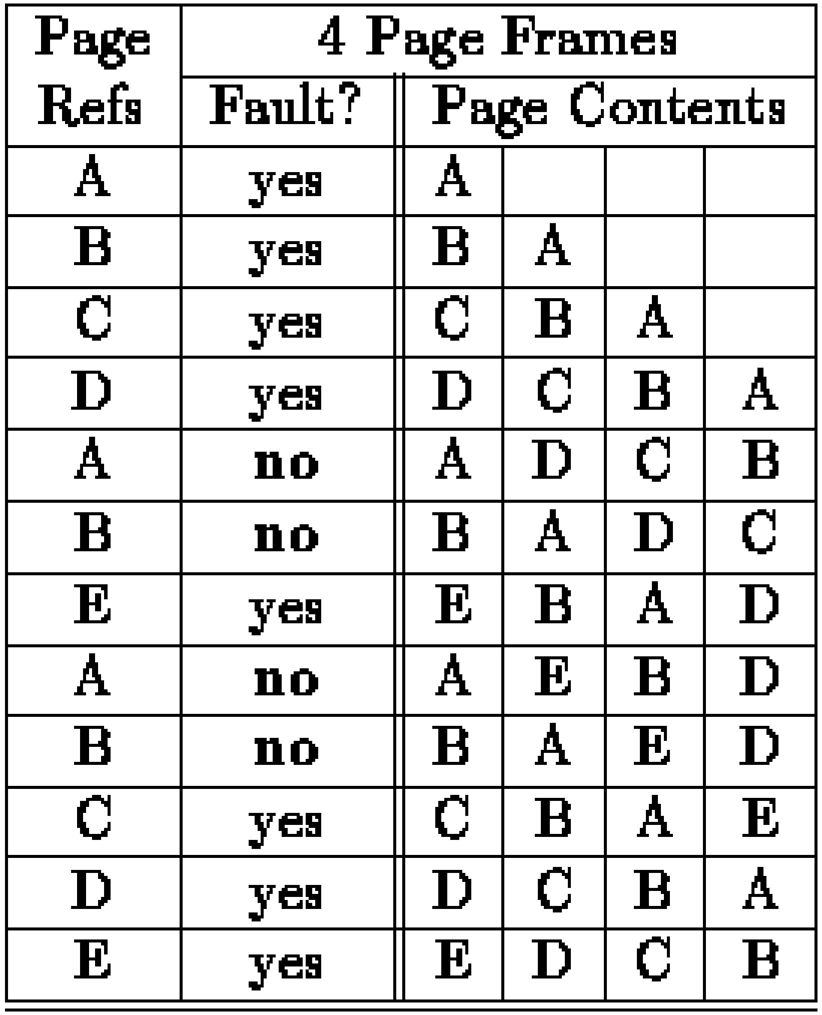
and when does it not?



Page Refs	3 Page Frames		
	Fault?	Page Contents		
A	yes	A		
B	yes	B	A	
C	yes	C	B	A
D	yes	D	C	B
A	yes	A	D	C
B	yes	B	A	D
E	yes	E	B	A
A	no	A	E	B
B	no	B	A	E
C	yes	C	B	A
D	yes	D	C	B
E	yes	E	D	C

LRU Discussion (1)

- Advantages
  - Quite efficient, good to approximate OPT.
  - Does not suffer from Belady's anomaly (also OPT).
    - Example: 12 references, 8 faults



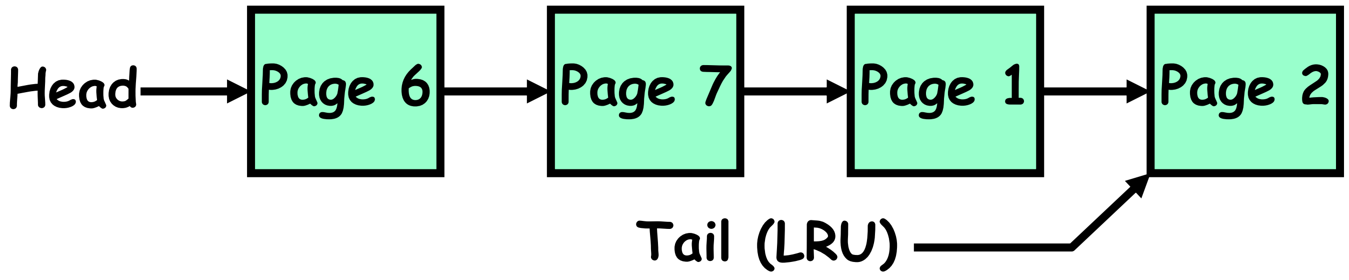
Page Refs	4 Page Frames				
	Fault?	Page Contents			
A	yes	A			
B	yes	B	A		
C	yes	C	B	A	
D	yes	D	C	B	A
A	no	A	D	C	B
B	no	B	A	D	C
E	yes	E	B	A	D
A	no	A	E	B	D
B	no	B	A	E	D
C	yes	C	B	A	E
D	yes	D	C	B	A
E	yes	E	D	C	B

LRU Discussion (2)

- Disadvantages
  - Its implementation is not very easy.
  - Its implementation may require substantial hardware assistance.
- Several ways to implement this algorithm, but very expensive.

Implementation of LRU (1)

- Implementation #1 linked list (Hardware)
  - Maintain the LRU order of accesses in frame list by hardware.
  - On every memory reference, move that page to the front of the list.
  - The page at the tail of the list is replaced.
  - Problem
    - Expensive!! "*on every memory reference...*"



Implementation of LRU (2)

- Implementation #2 counter (Hardware)
  - MMU maintains a 64-bit counter C.
  - Increment C after every clock cycle.
  - Every time a page table entry is used, MMU writes the counter value to the entry “timestamp” / “time-of-last-use” field.
  - When a page fault occurs, software looks through the page table.
  - Identifies the entry with the oldest timestamp.

Implementation of LRU (3)

- Implementation #2 counter (Hardware) (ctd.)
  - Problems
    - The number of counters is equal to the number of virtual pages. So the space cost much.
    - Each page table entry has a field large enough to include the value of the counter. So the space cost much.
    - When we choose page to replace, we need compare all counters. So the time cost much.

Implementation of LRU (4)

- Few computers have the necessary hardware to implement full LRU.
  - Linked-list method impractical in hardware.
  - Counter-based method could be done, but it’s slow to find the desired page.
  - Both approaches require large processing overhead, more space, or hardware support.
- What if we don’t have hardware support?
  - Software simulated.

Not Frequently Used (NFU) (1)

- Simulated LRU
- There is a software counter for each page.
- The counter is initially zero. At each clock interrupt , the OS looks at each page
  - If the Reference Bit is set (R=1)
    - Add the R bit to a software counter for each page & clear the bit.
  - If the Reference Bit is not set (R=0)
    - Add the R bit to a software counter for each page.
- Remove page with lowest counter value.

Not Frequently Used (NFU) (2)

- Problem
  - Some page may be heavily used. (Its counter is large.)
  - The program's behavior changes.(Now, this page is not used ever again (or only rarely).)
  - But this algorithm never forgets!
  - This page will never be chosen for replacement!

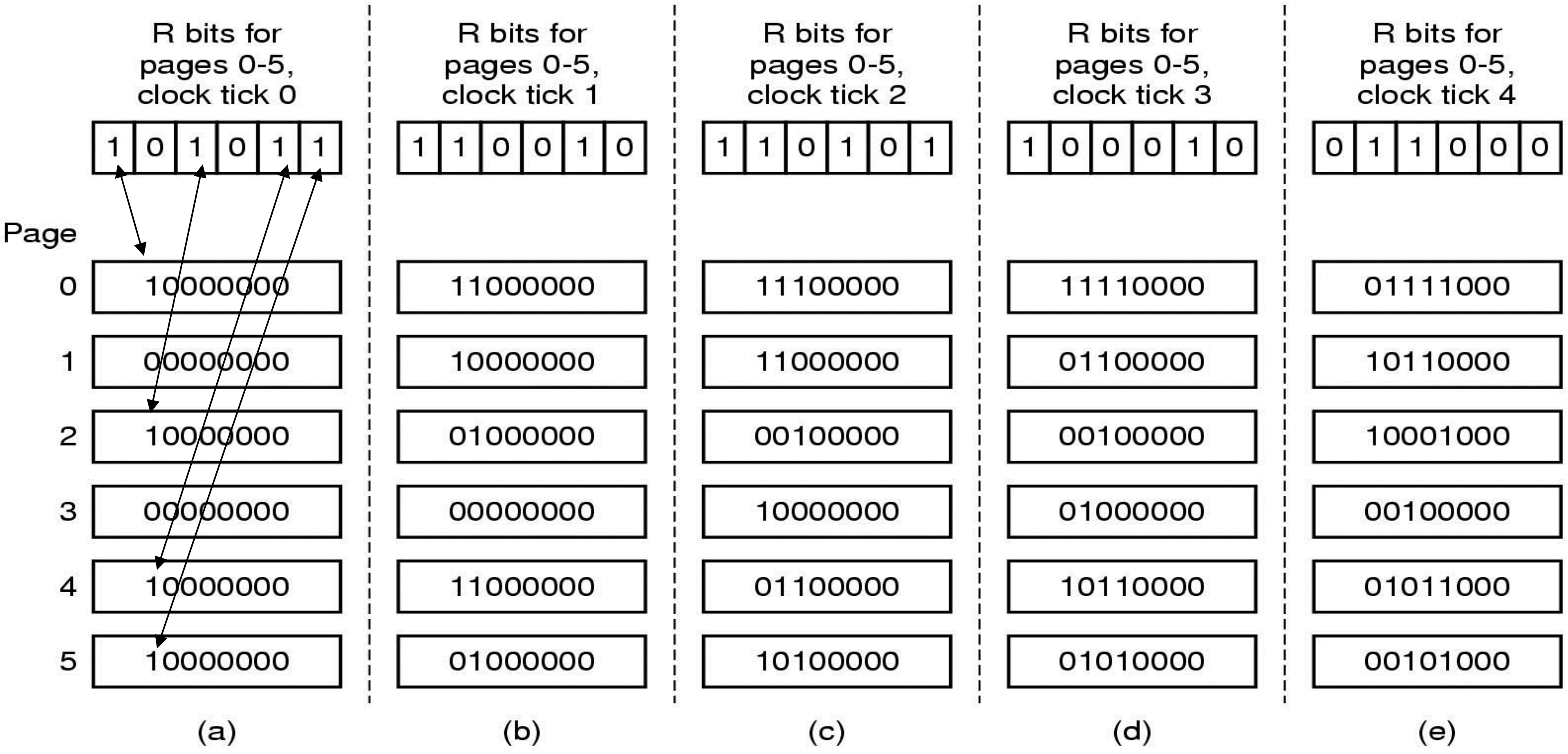
Not Frequently Used (NFU) (3)

- Differences between LRU and NFU
  - LRU updated after every instruction so it’s resolution is very fine.
  - NFU is coarse (updated after n instructions execute between clock interrupts).
    - A given page referenced by n-1 instruction is given equal weight to a page referenced by only 1 instruction (between clock interrupts).
    - n/2 references to a given page at the beginning of the interval are given equal weight with n/2 references to another page at the end of the interval.

The Aging Algorithm (1)

- Modified NFU: NFU + forgetting
- Take NFU but…
- At each clock interrupt
  - Right shift the counters (divide by 2)
  - Add the R bit to the left (MSB)
    - More weight given to more recent references!
- As for NFU, remove pages with the lowest counter.
- Note: this is different from LRU since the time granularity is a clock tick and not every memory reference!

The Aging Algorithm (2)



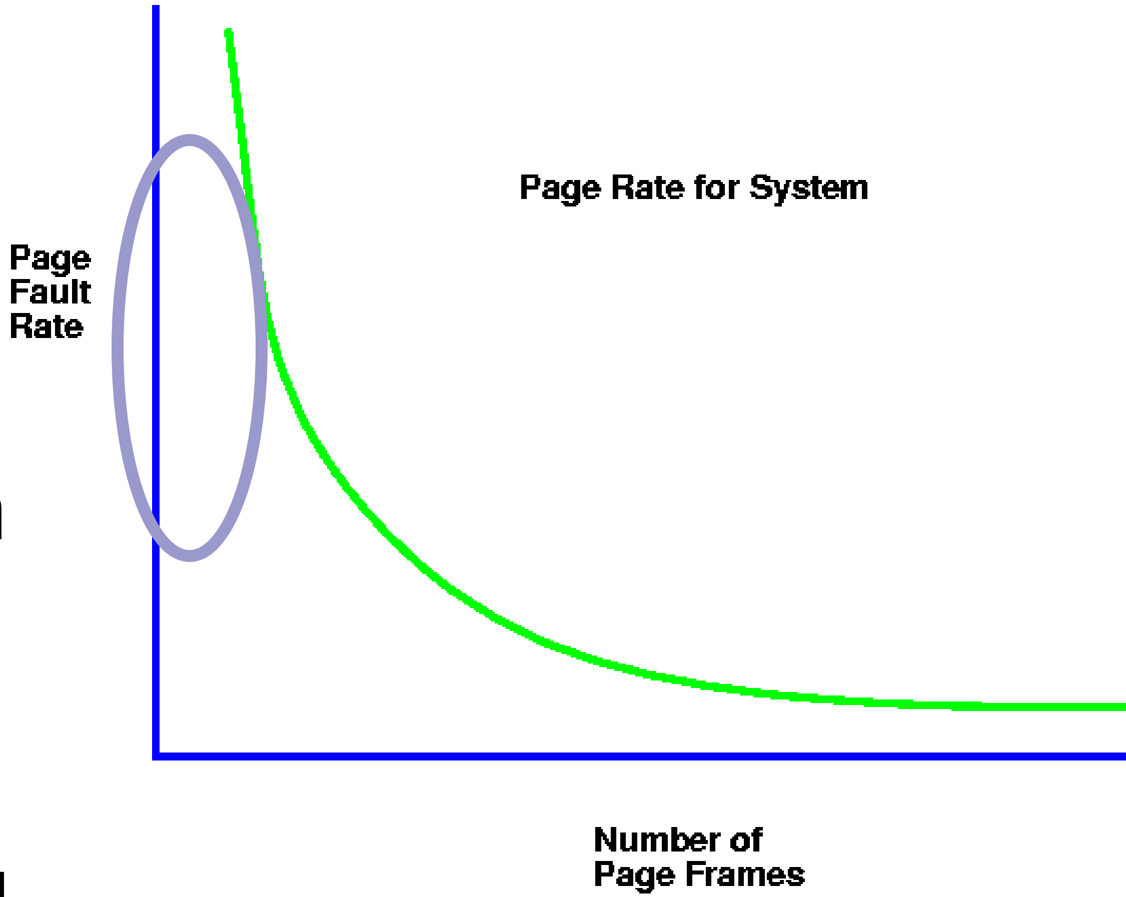
- The aging algorithm simulates LRU in software
- Note 6 pages for 5 clock ticks, (a) – (e)

Fetch Policy

- When should a page be loaded into main memory?
- Demand Paging Policy
  - Pages are loaded on demand, not in advance.
  - Process starts up with none of its pages loaded in memory; page faults load pages into memory.
- Pre-paging Policy
  - Load pages before process runs.
  - Need a *working set* of pages to load on context switches.
- Few systems use pure demand paging, but instead, does pre-paging.

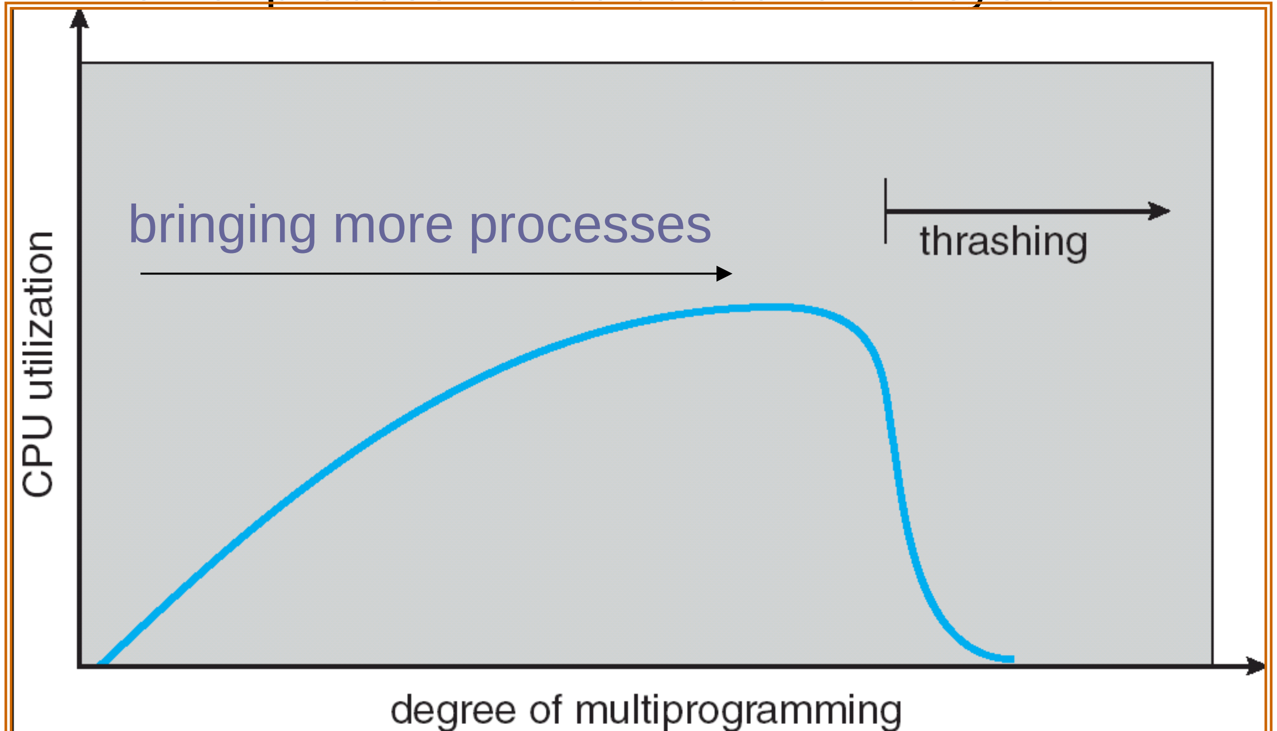
Thrashing (1)

- **Thrashing** ≡ a process is busy swapping pages in and out. (spends more time paging than executing)
- Why Thrashing?
  - Too few page frames allocated to a process!



Thrashing (2)

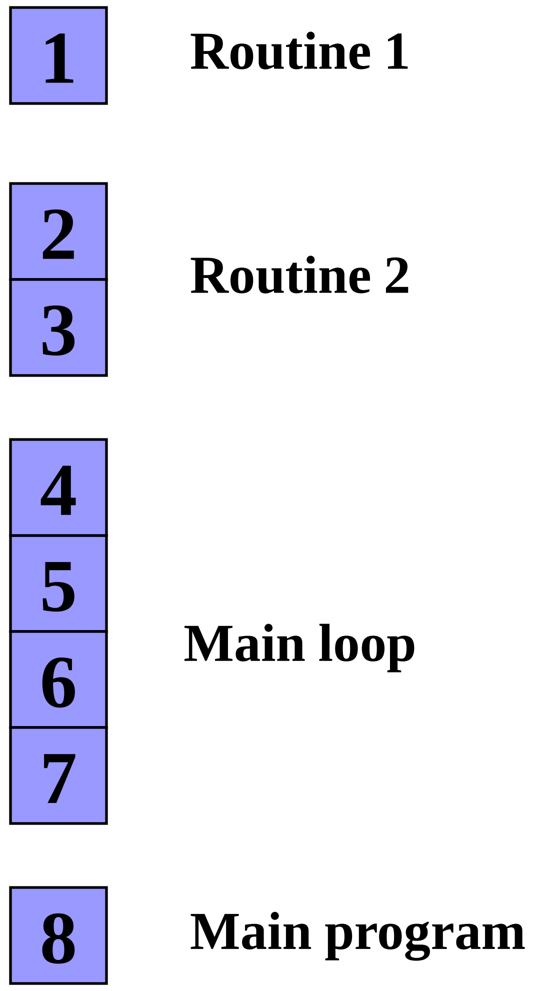
- Results of Thrashing
  - If a process does not have “enough” pages, the page-fault rate is very high. This leads to
    - Low CPU utilization.
    - OS thinks that it needs to increase the degree of multiprogramming.
    - Another process will be added to the system.



Solutions to Thrashing

- Approach 1
  - Working Set
    - Provide a process as many frames as it needs
- Approach 2
  - Page Fault Frequency (discussed in Chapter 3.5)
- Working Set
  - A process's *working set* is the set of pages that it currently “needs”. (Denning’s)
  - Example (next page)

Working Set Example (1)

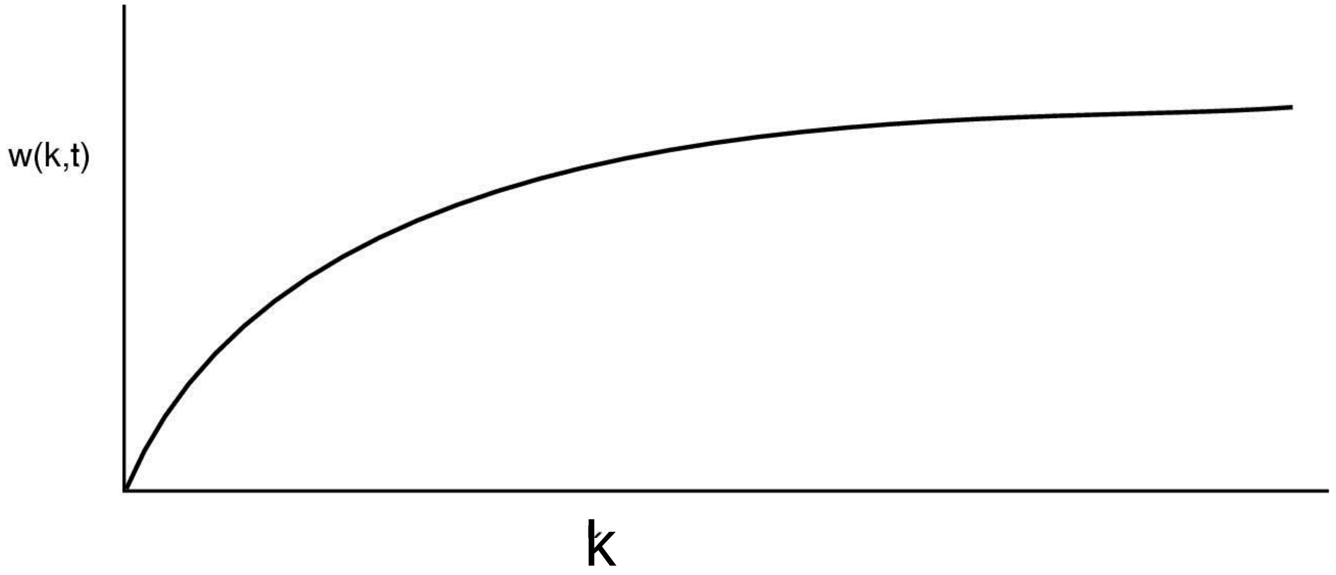


- Program Execution
  - Start with main program (page 8)
  - Main program loads the main loop (pages 4,5,6,7)
  - Program executes in the main loop for ***20*** seconds
  - Then routine 1 is called (page 1) which executes ***2*** seconds
  - Finally routine 2 (pages 2, 3) is called to execute for ***3*** seconds

Working Set Example (2)

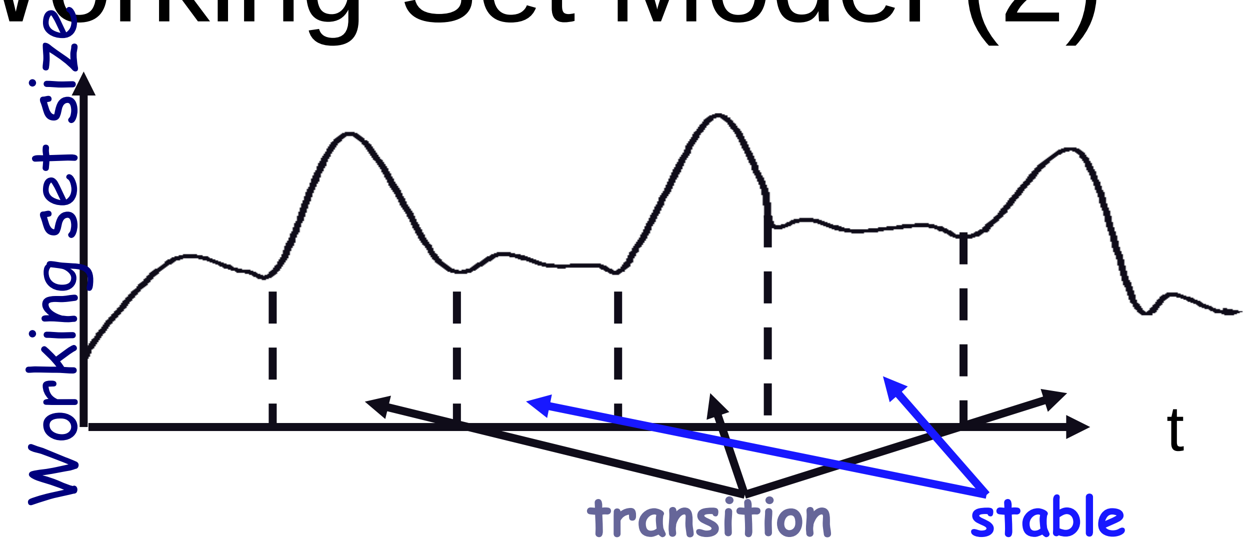
- In the previous example, the process needs pages 4, 5, 6, 7 for most of the time (20 seconds in a total of 25 seconds execution time).
- If these pages are kept in memory the number of page faults will decrease. Otherwise, thrashing may occur.
- ***Rule***
    - Do not run a process if its working set can not be kept in memory.

Working Set Model (1)



- The working set is the set of pages used by the *k* most recent memory references.
- The function w(k,t) is the size of the working set at time *t.*
  - w(k, t) is a monotonically nondecreasing function of k.
  - The limit of w(k, t) is finite.

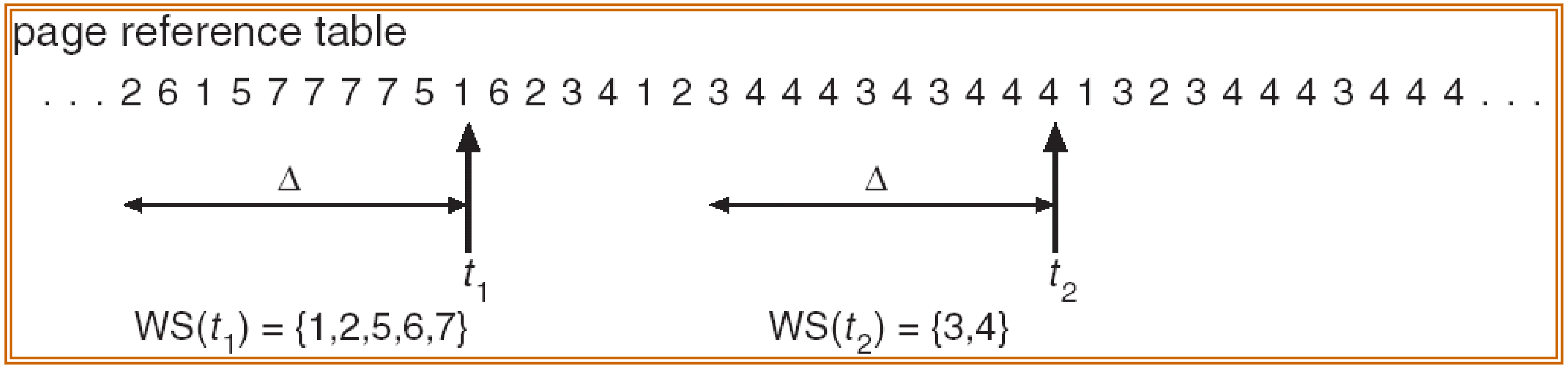
Working Set Model (2)



- When a process first starts up, its working set grows.
- Then stabilizes by the principle of locality.
- Grows again when the process shifts to a new locality. (transition period)
  - Up to a point where the working set contains pages from two localities.
- Then decreases (stabilizes) after it settles in the new locality.

Working Set Model (3)

- k or Δ
  - working set window= a fixed number of page references
  - Example Δ=10



- Approximate Working Set
  - The set of pages of a process used in the last τ seconds.

The Working Set Algorithm (1)

- Basic Idea
  - When a page fault occurs, find a page that is not in the working set and evict it.
  - Working set: the set of pages used in the k most recent memory references.
- Approximation
  - Drop the idea of counting back k memory references and use execution time.
  - Current Virtual Time: The amount of CPU time a process has actually used since it started is called its current virtual time.
  - The working set of a process is the set of pages it has referenced during the past τ seconds of virtual time.

The Working Set Algorithm (2)

- WS Algorithm in Practice
  - On each clock tick, clear all R bits and record process virtual time *t.*
  - When looking for eviction candidates, scan **all** pages of process in physical memory:
    - If R == 1

      Store *t* in *LTU* (last time used) of PTE and clear R.
    - If R == 0

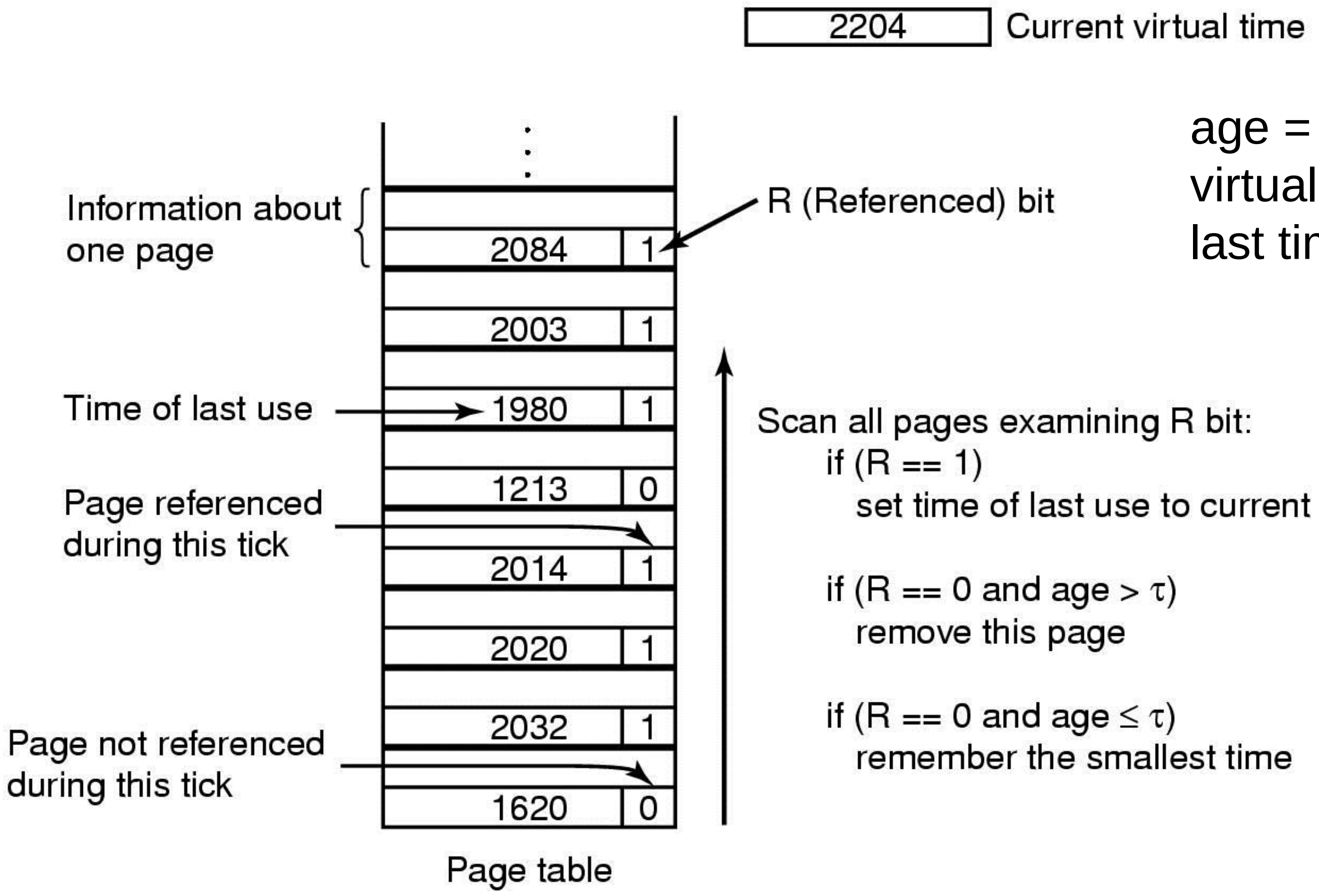
      If (*t* – *LTU*) > τ, evict the page (because it is not in working set).

      If (t – LTU) <= τ, record the page of greatest age.

      age = current virtual time – last time used

The Working Set Algorithm (3)

- WS Algorithm Example



```
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
```

age = current virtual time – last time used

← (greatest age/least virtual time) and choose that one if no better candidate exists.

The Working Set Algorithm (4)

- What happens if the entire table is scanned without finding a candidate to evict?
  - If one or more pages with R=0 were found, the one with the greatest age is evicted.
  - The worst case: no page with R=0, choose one page at random (preferably a clean page).

The WSClock Algorithm (1)

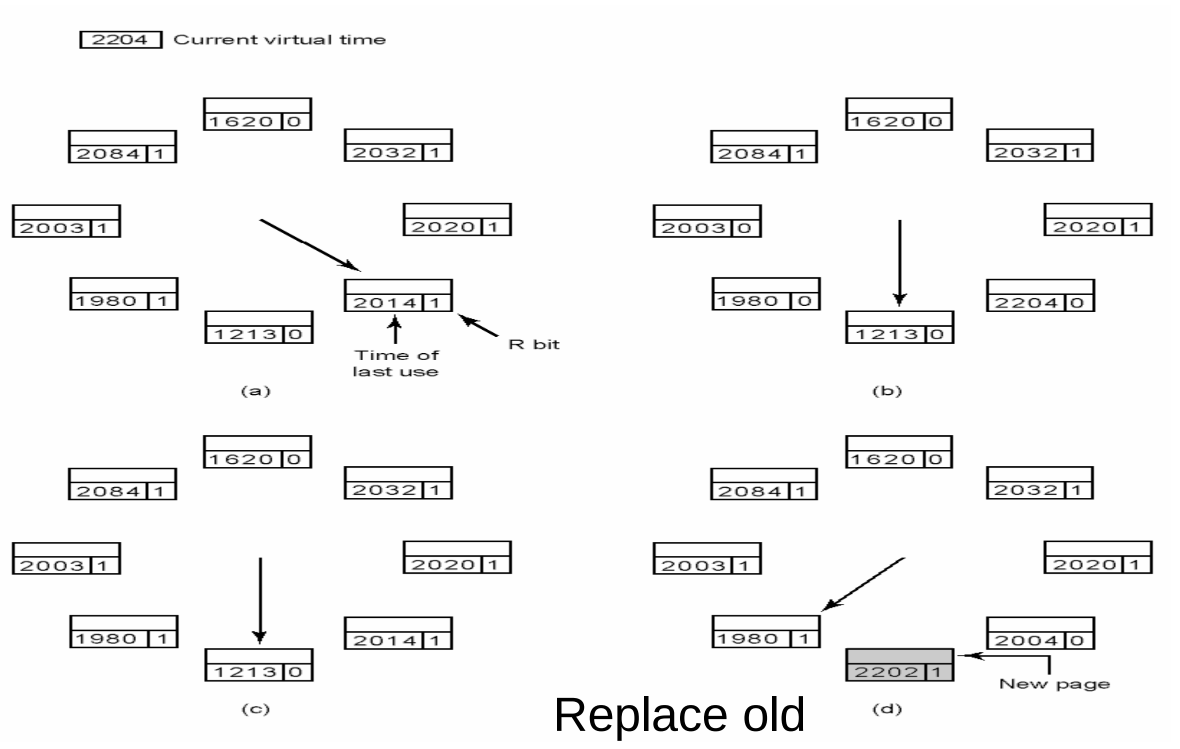
- Previous WS algorithm requires entire page table be scanned at each page fault.
- An improved algorithm
  - Simple, efficient, widely used.
  - Based on the clock algorithm.
  - Also uses the working set information.
  - Uses circular list of page frames.
- WSClock Algorithm
  - All pages are kept in a circular list.
  - As pages are added, they go into the ring.
  - The “clock hand” advances around the ring.
  - Each entry contains R, M, *time of last use.*

The WSClock Algorithm (2)

- WSClock (ctd.)
  - Upon a page fault
    - If R = 1, set R=0 and update “*time of last use” field.* Advance the hand and keep looking for.
    - If R = 0
      - If *age* <= τ, advance the hand and keep looking for.
      - If *age* > τ*,* and if clean, then claim it.
      - if *age* > τ*,* and if dirty, then schedule a disk write. Advance the hand and keep looking for.
- Very common in practice.

The WSClock Algorithm (3)

- WSClock Example



Replace old and advance.

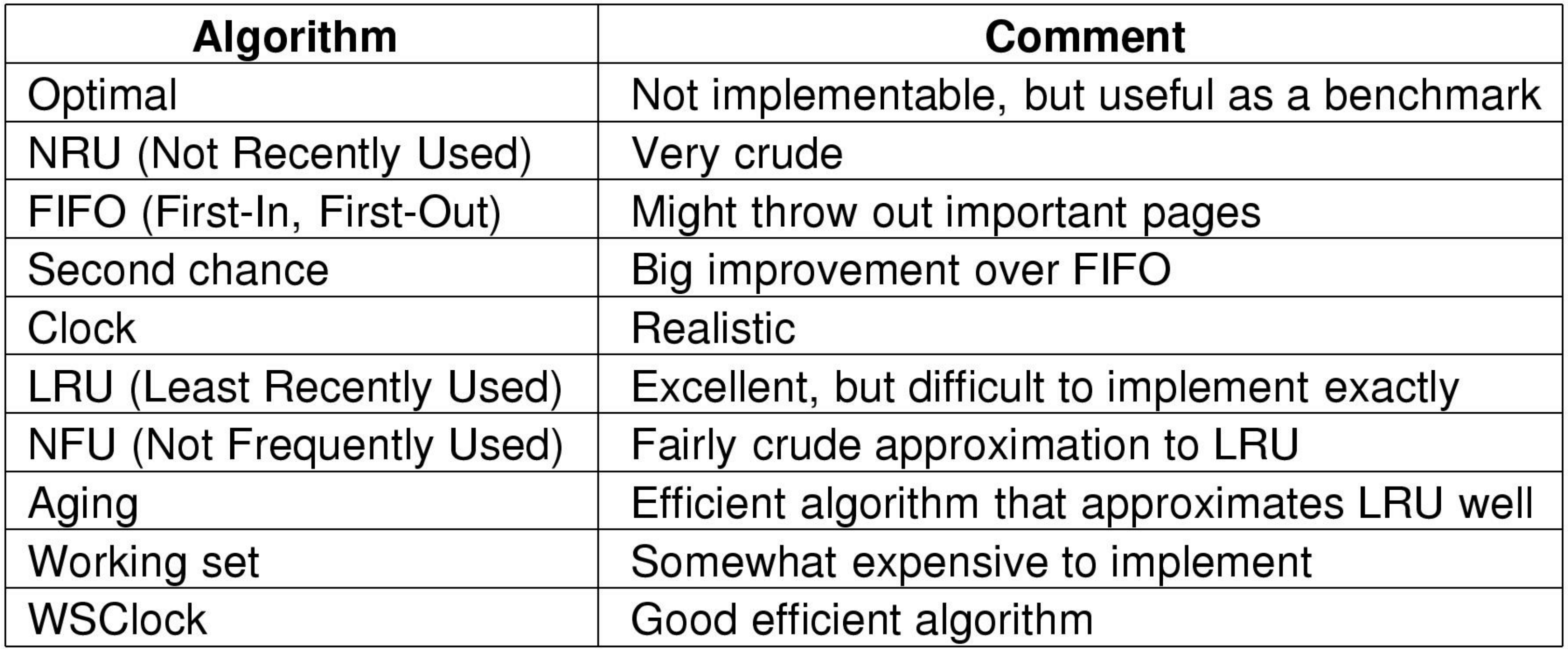
Figure 3-20 Operation of the WSClock algorithm

The WSClock Algorithm (4)

- What happens if the hand comes all the way around to its starting point?
  - Case1: At least one write has been scheduled.
    - The first clean page encountered is evicted.
  - Case 2: No writes have been scheduled (all pages are in working set)
    - Claim any clean page and use it.
    - If no clean pages exist, the current page is chosen and written back to disk.

Summary of Page Replacement Algorithms

- Two Best Algorithms
  - Aging
  - WSClock



Algorithm	Comment
Optimal	Not implementable, but useful as a benchmark
NRU (Not Recently Used)	Very crude
FIFO (First-In, First-Out)	Might throw out important pages
Second chance	Big improvement over FIFO
Clock	Realistic
LRU (Least Recently Used)	Excellent, but difficult to implement exactly
NFU (Not Frequently Used)	Fairly crude approximation to LRU
Aging	Efficient algorithm that approximates LRU well
Working set	Somewhat expensive to implement
WSClock	Good efficient algorithm

Homework

- P256: 28, 30, 33, 36