Operating Systems

0107451

Chapter 3 Page Replacement Policies

Dr. Naeem Odat

Tafial Technical University
Department of Computer and Communications Engineering

Page Replacement Policies

Problem Statement

A page is being brought into memory which has no free space. Which page should we replace to make space?

Page Replacement Policies

Mental Exercise

- What is the optimal policy, if we had the knowledge of the future page accesses?
- Policy: Choose the page which will be referenced farthest in the future.
- However, we don't know the future. Hope that the next few references will be for pages that were recently referenced. (Locality of references)
- What is the use of knowing about this policy? Will help us assess the performance of a real algorithm.

Page Replacement Policies

Replacement Policies

- Policies:
  - NRU (Not-Recently-Used)
  - FIFO (First-In-First-Out)
    - Second Chance.
    - Clock Algorithm(s).
  - LRU (Least-Recently-Used)
  - NFU (Not Frequently Used).
  - Working Set Algorithm
- Two issues:
  - How good is the decision?
  - Overhead?
    - Cost per memory access - should be very small
    - Cost per replacement - can be larger

Page Replacement Policies

Help From Hardware

For each page frame:

- Referenced Bit (R) - 1 if page frame has been referenced recently
- Modified Bit (M) - 1 if the page has been modified since it has been loaded. Also known as "dirty bit"

Page Replacement Policies

Not Recently Used Algorithm (NRU)

- Pages are classified into 4 groups:
  1. Class 0: not referenced, not modified (R=0, M=0)
  2. Class 1: not referenced, modified (R=0, M=1)
  3. Class 2: referenced, not modified (R=1, M=0)
  4. Class 3: referenced, modified (R=1, M=1)
- NRU removes page at random from the lowest numbered non empty class.
- The R bit is cleared periodically (at each clock tick, 20 ms).

Page Replacement Policies

FIFO

1	0	2	2	1	7	6	7	0	1	2	0	3	0	4	5	1	5	2	4	5	6	7	6	7	2	4	2	7	3	3	2	3
1	1	1	1	1	1	6	6	6	6	6	6	6	6	4	4	4	4	4	4	4	6	6	6	6	6	6	6	6	6	6	2	2
-	0	0	0	0	0	0	0	0	1	1	1	1	1	1	5	5	5	5	5	5	5	7	7	7	7	7	7	7	7	7	7	7
-	-	2	2	2	2	2	2	2	2	2	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	4	4	4	4	4	4	4
-	-	-	-	-	7	7	7	7	7	7	7	3	3	3	3	3	3	2	2	2	2	2	2	2	2	2	2	2	3	3	3	3
F	F	F			F	F			F		F	F		F	F	F		F			F	F				F			F		F	

Hit ratio

$Hit\ ratio = \frac{16}{33}$

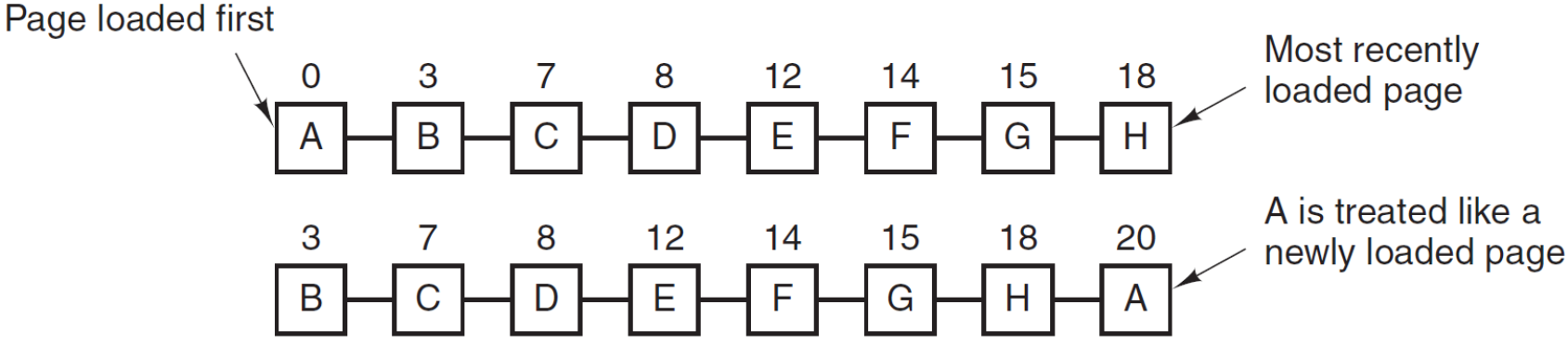
Page Replacement Policies

FIFO - Second Chance

- Based on FIFO
- Old pages are inspected for replacement. But given a "second chance" if they have been used recently (If R=1).

Page Replacement Policies

FIFO - Second Chance



- Pages sorted in FIFO order (time of arrival)
- If earliest page has R=1, then give it a second chance by moving it to the end of the list
- Keep moving pages around in its list (drawback)

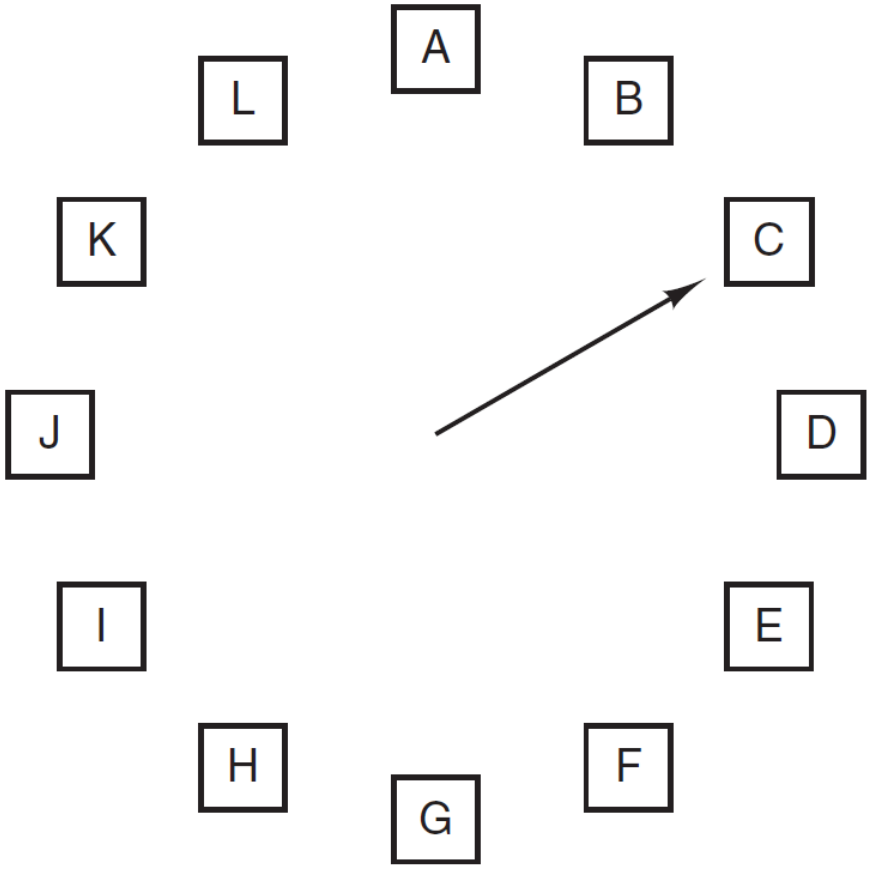
Page Replacement Policies

FIFO - Clock Algorithm

- Order pages in circular list.
- "Hand" of the clock points to the page to be replaced currently.
- When required to evict a page:
  - If page pointed to has R=0, then evict it and place new page there.
  - If R=1, then reset R and move hand forward.

Page Replacement Policies

FIFO - Clock Algorithm



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

Page Replacement Policies

Least Recently Used (LRU)

- Replace that page in memory which has been unused for the longest time.
- Locality of Reference: Pages used in the near past will be used in the near future. (True in typical cases)

Page Replacement Policies

Least Recently Used (LRU)

1	0	2	2	1	7	6	7	0	1	2	0	3	0	4	5	1	5	2	4	5	6	7	6	7	2	4	2	7	3	3	2	3
1	1	1	1	1	1	1	1	1	1	1	1	1	1	4	4	4	4	4	4	4	4	4	4	4	2	2	2	2	2	2	2	2
-	0	0	0	0	0	6	6	6	6	2	2	2	2	2	5	5	5	5	5	5	5	5	5	5	5	4	4	4	4	4	4	4
-	-	2	2	2	2	2	2	0	0	0	0	0	0	0	0	0	0	2	2	2	2	7	7	7	7	7	7	7	7	7	7	7
-	-	-	-	-	7	7	7	7	7	7	7	3	3	3	3	1	1	1	1	1	6	6	6	6	6	6	6	6	3	3	3	3
F	F	F			F	F		F		F		F		F	F	F		F			F	F			F	F			F			

Hit ratio

$Hit\ ratio = \frac{16}{33}$

Page Replacement Policies

LRU Implementation

- One possible implementation:
  - list of pages, most recently used at front, least at rear.
  - update this list every memory reference!!.
  - when required to evict a page, choose the one at the rear of the list.
- Way too expensive!.

Page Replacement Policies

LRU Implementation

Another possible implementation:

- keep a counter that is incremented for each instruction reference.
- each page table entry have a place this counter can be saved in.
- At each memory reference of a page the counter is stored in its entry in the page table.
- When page fault occurs OS examines all entries in page table to evict a page with lowest counter value.

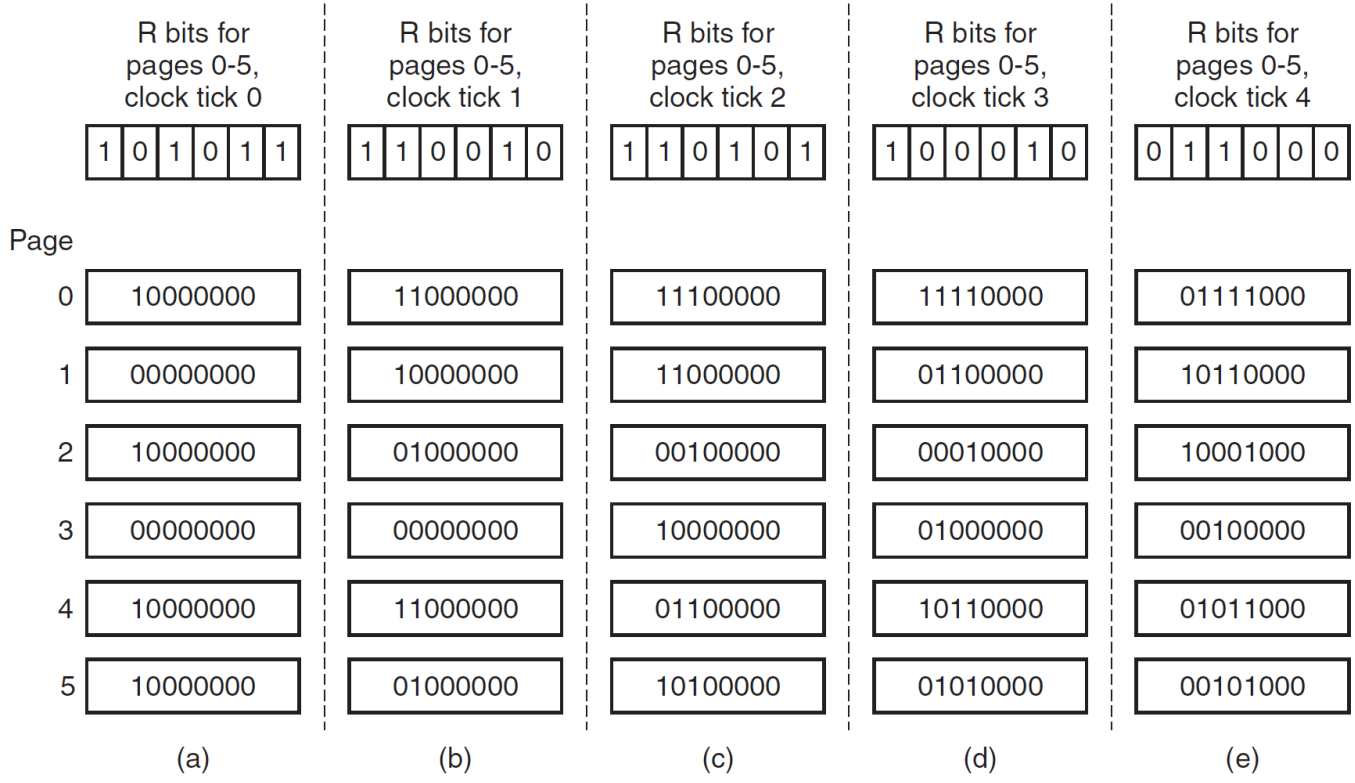
Page Replacement Policies

Not Frequently Used (NFU) - LRU Approximation

- Requires a software counter associated with each page, initially zero.
- At each clock interrupt, OS adds the R bit to the counter.
- The page with the lowest counter is chosen for evict.
- Drawback: OS will remove useful pages instead of pages no longer in use.

Page Replacement Policies

Aging - LRU Approximation



- The aging algorithm simulates LRU in software.
- Keeps an 8 bit structure for each page frame.
- Can provide an estimate of the usage history of the page.

Page Replacement Policies

Working Set

- Locality of Reference: During any phase of execution, a process references only a relatively small fraction of its pages.
- The **Working Set** of a process is the set of referenced pages in the last k memory references
- Each process should have the working set in the memory.
  - Keep track of working set
  - Make sure the process' working set is in memory before letting the process run. Loading the pages before letting it run (prepaging).
  - Reduce the page fault rate, avoid thrashing this way.

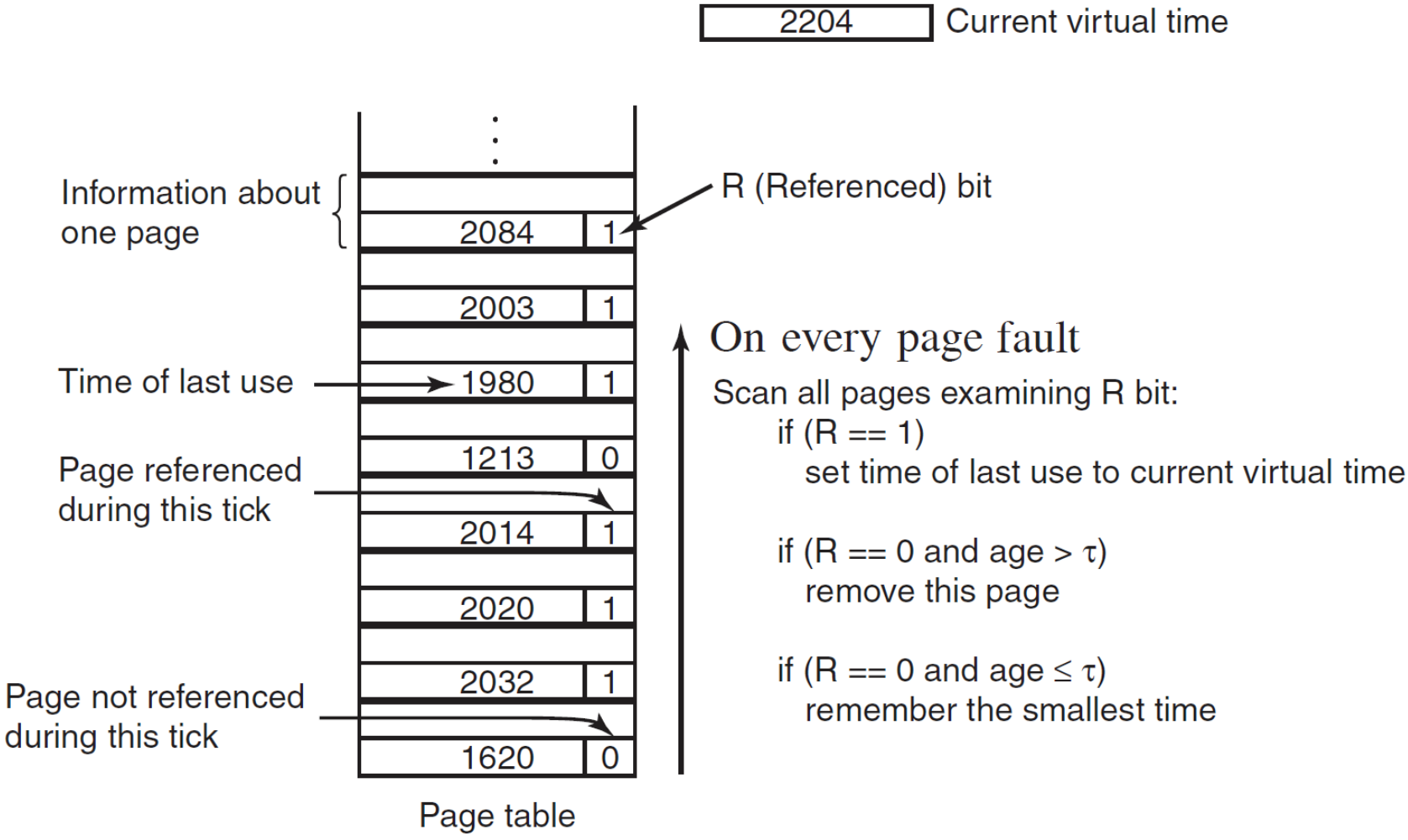
Page Replacement Policies

Working Set - Thrashing

- If the available memory is too small to hold the entire working set, the process will cause many page faults and run slowly.
- Consider a system that has exactly two page frames:
  - process A has a page in frame 1.
  - process B has a page in frame 2.
  - process B faults; the page in frame 1 is removed.
  - process A resumes execution and faults again; the page in frame 2 is removed.
  - ...
- The processes spend most of the time waiting for disk reads.
- A program causing page faults every few instructions is said to be **thrashing** [Denning, 1968]

Page Replacement Policies

Working Set Algorithm



Page Replacement Policies

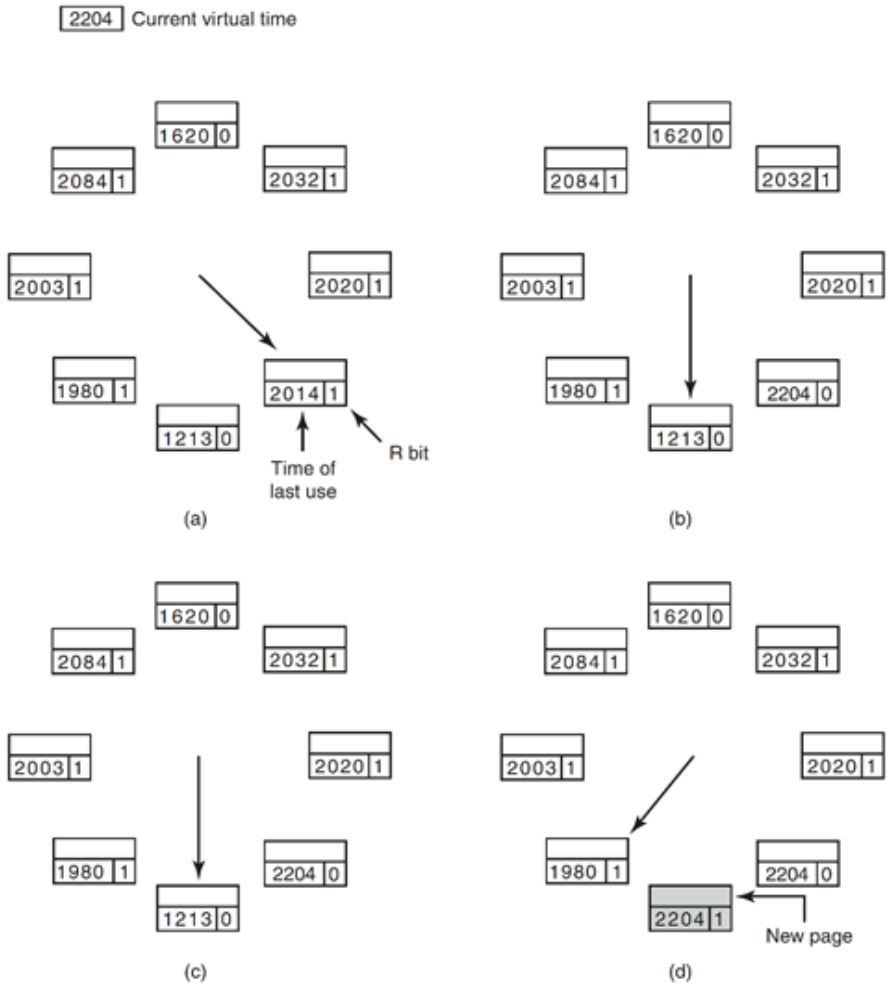
WSClock Algorithm

At each page fault the page pointed to by the hand is examined first.

- If R =1.
  - It is then set to 0.
  - Time of last use is set to current virtual time.
  - The hand advanced to the next page, and the algorithm repeated for that page.
- If R = 0 and M=0.
  - If the age is greater than τ, the page frame is simply claimed and the new page put there.
  - If the age is less than τ, the page frame is remembered.
- If R=0 , M=1 and age is greater than τ.
  - It cannot be claimed immediately.
  - A write to disk is scheduled.
  - The hand is advanced and the algorithm continues with the next page.

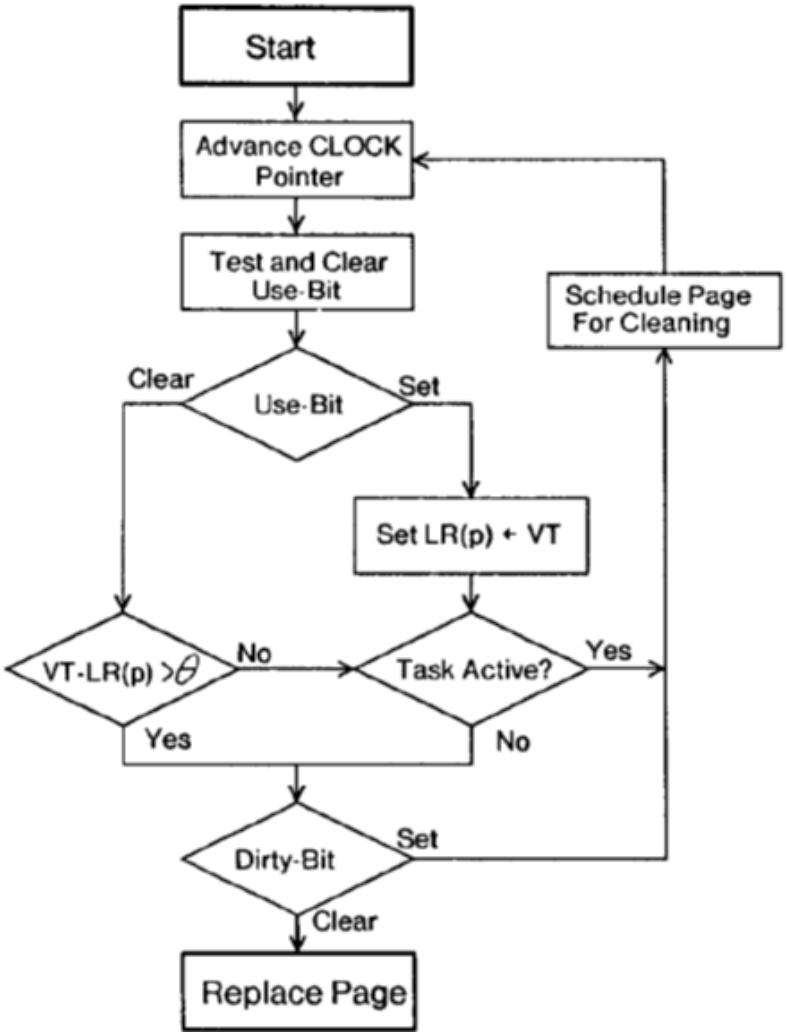
Page Replacement Policies

WSClock Algorithm



Page Replacement Policies

WSClock Algorithm Flowchart



Page Replacement Policies

Page Replacement Algorithms Summary

Algorithm	**Comment**
Optimal	Not implementable, useful as a benchmark
NRU (Not Recently Used)	Very crude approximation of LRU
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 of LRU
Aging	Efficient algorithm, approximates LRU well
Working set	Somewhat expensive to implement
WSClock	Good efficient algorithm

Page Replacement Policies

Example

Page frame	Time loaded	Time referenced	R	M
0	60	161	0	1
1	130	160	0	0
2	26	162	1	0
3	20	163	1	1

- Which page frame will be replaced, using?
  - FIFO
  - LRU
  - Clock

Page Replacement Policies

Answers

- PFN 3 since loaded longest ago at time 20
- PFN 1 since referenced longest ago at time 160
- Clear R in PFN 3 (oldest loaded), clear R in PFN 2 (next oldest loaded), victim PFN is 0 since R=0

Page Replacement Policies

Page Frame Allocation
Global vs. Local

- Global allocation:
  - All processes compete for page frames from a single pool.
  - Don't have to decide how many pages go to different processes.
  - High priority processes might get pages off lower priority processes
- Local allocation
  - Each process has its own private pool of page frames.
  - Equal allocation: processes get equal number of page frames.
  - Proportional allocation: number of page frames proportional to size of virtual memory used.

Page Replacement Policies

Belady's anomaly (FIFO - 1 17-page faults)

3	2	1	0	3	2	4	3	2	1	0	4	2	3	2	1	0	4
3	3	1	1	3	3	4	4	2	2	0	0	2	2	2	1	1	4
	2	2	0	0	2	2	3	3	1	1	4	4	3	3	3	0	0

Belady's anomaly (FIFO - 2 14-page faults)

3	2	1	0	3	2	4	3	2	1	0	4	2	3	2	1	0	4
3	3	3	0	0	0	4	4	4	4	4	4	2	2	2	2	0	0
	2	2	2	3	3	3	3	3	1	1	1	1	3	3	3	3	4
		1	1	1	2	2	2	2	2	0	0	0	0	0	1	1	1

Belady's anomaly (FIFO - 3 15-page faults)

3	2	1	0	3	2	4	3	2	1	0	4	2	3	2	1	0	4
3	3	3	3	3	3	4	4	4	4	0	0	0	0	0	1	1	1
	2	2	2	2	2	2	3	3	3	3	4	4	4	4	4	0	0
		1	1	1	1	1	1	2	2	2	2	2	3	3	3	3	4
			0	0	0	0	0	0	1	1	1	1	1	2	2	2	2

Page Replacement Policies

Performance

- Address translation is done on every memory reference
- Maybe twice per instruction:
  - Instruction fetch
  - Fetch Memory operand
- Translation better be fast!

Page Replacement Policies

Where do page tables go?

- Registers:
  - Fast translation.
  - Can be used for small page tables (a few 10s of entries).
  - Context Switch is quite expensive: reload new page translations into registers
- Memory:
  - Slow translation.
  - Large tables can be stored.
  - The page-table base register **(PTBR)** holds a pointer to the page table location.
  - Page-table length register **(PTLR)** indicates size of the page table.
  - Context Switch is quick, only need to change this register.
  - Currently used in most systems, because of the large page tables.
  - Problem: Cannot afford a memory access for each translation 4 memory in total for an instruction involving memory operands!

Page Replacement Policies

Faster Translations

- Translation Lookaside Buffer = Page Table Cache
- Frequent translations found in the high-speed cache - **TLB hit**.
- Rest will go to slower-speed memory - **(TLB miss)**

Page Replacement Policies

TLBs - Translation Lookaside Buffers

Valid	Virtual Page	Modified	Protection	Page Frame
1	140	1	RW	31
1	20	0	R X	38
1	130	1	RW	29
1	129	1	RW	62
1	19	0	R X	50
1	21	0	R X	45
1	860	1	RW	14
1	861	1	RW	75

- Speeds up paging by caching recent address translations.
- Typically small size - a few 10s of entries.
- TLB Hit rates are very important for performance

Page Replacement Policies

TLB = Associative Memeory

- Given a virtual address, check all the TLB locations simultaneously for a hit. (Requires expensive hardware)
- Usually between 64 - 1024 entries.
- Multiple address spaces: TLB contains translations for only one address space at a time.
- TLB flushed on every context switch.
- Contains translations for all address spaces simultaneously.
- Each entry should have an identifier for the address space

Page Replacement Policies

TLB impact on AAT (Average Access Time)

- Associative Lookup $= \epsilon$ time unit.
- Assume memory cycle time is m time unit.
- Hit ratio (α) - percentage of times that a page number is found in the associative registers.
- Hit ratio related to number of associative registers.
- Average Access Time **(AAT)**:

$$AAT = \alpha(m + \epsilon) + (1 - \alpha)(2m + \epsilon) = 2m + \epsilon - \alpha m$$

Page Replacement Policies

Impact of TLB on Performance

- TLB hit ratio = percent of time a translation can be found in the TLB.
- Example:
  - 80 percent hit ratio.
  - TLB search = 20 nsec,
  - Memory access = 100 nsec.
  - Effective memory access time = AAT =?
- Answer = 0.8 (120) + 0.2(220) = 140 nsec

Page Replacement Policies

Page-Table size

- Consider a full 2^{32}-byte address space
  - assume 4096-byte (2^{12}-byte) pages
  - 4 bytes per page table entry the page table would consist of $2^{32}/2^{12}$ ($=2^{20}$) entries.
  - its size would be 2^{22} bytes (or 4 megabytes)
- Imagine 2^{64}-byte address space.

Page Replacement Policies

One Solution

- Put the page tables themselves in virtual memory!.
- Only the currently active translations are in physical memory.