

Virtual memory

Operations and policies

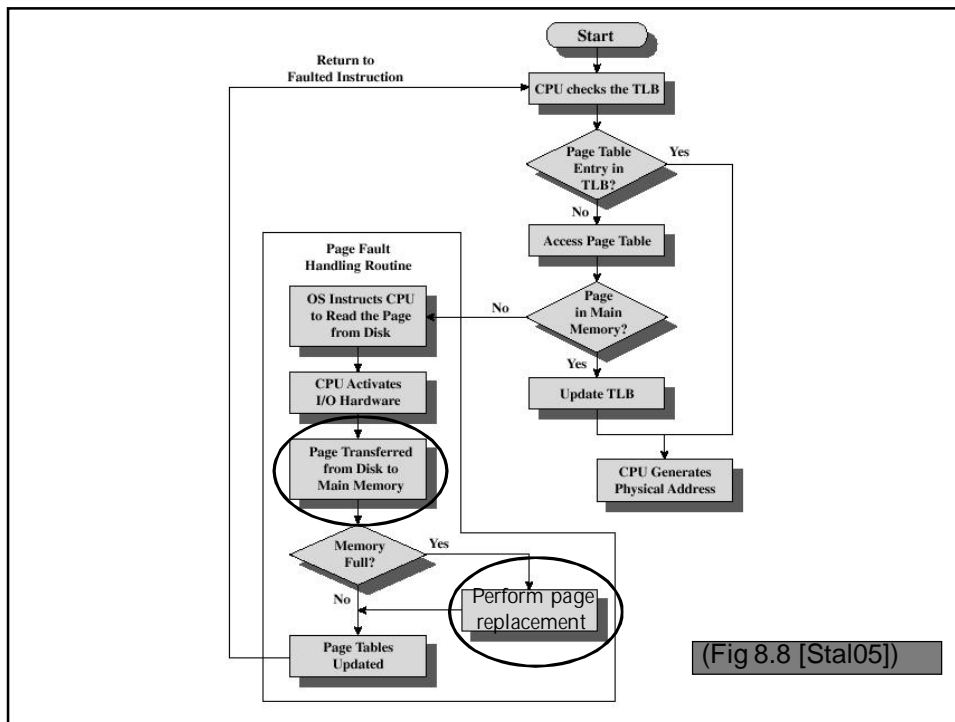
Chapters 3.4. – 3.7

1

Policies and methods

- Fetch policy (*Noutopolitiikka*)
 - When to load page to memory?
- Placement policy (*Sijoituspolitiikka*)
 - Where to place the new page?
- Replacement policy (*Korvaus/poistopolitiikka*)
 - Which page to be evicted and replaced
- cleaning policy (*Levyille kirjoitus –politiikka*)
 - When to evict a modified page?
- Multiprogramming, load control (*Moniajoaste*)
 - How many processes in the system at the same time?
 - How much memory, how many pages per process?
- Working set (*Käyttöjoukko*)
 - How many page frames allocated for one process? (resident set)
 - Which areas have been referenced recently?

2



Page fault (*Sivun puutos*)

- If the requested page is not in memory, MMU causes interrupt
- OS process the interrupt and notice a page fault
 - Move the process to blocked state
 - Allocate page frame for the new page
 - Start page load from disk using controller
 - Switch another process for execution
- After the driver interrupt
 - OS notices the termination of page transfer
 - Move process to ready state
 - Continue the process now or later (scheduling decision)

4

Locking pages

- To avoid page fault, page can be locked to memory
 - Kernel pages
 - Key data structures of OS
 - I/O buffers (at least during the transfer)
 - Process pages (at least in real time systems)
- How?
 - Page frame table? has a lock bit on the frame
 - Per frame
 - Or page table entry has the lock bit
 - Per process

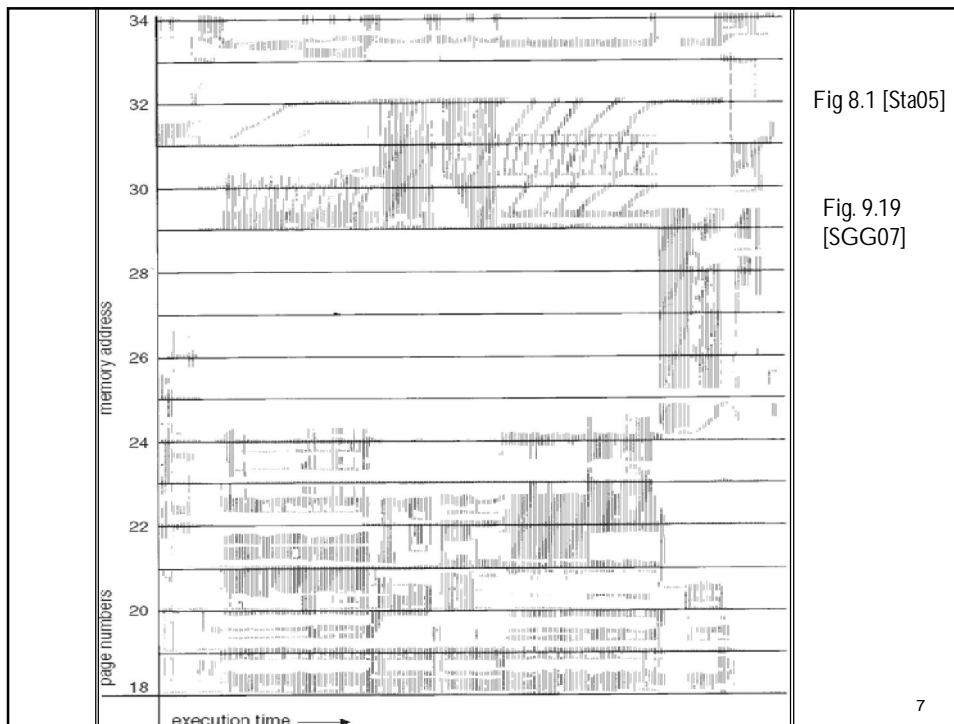
5

Page replacement

- When?
 - Page fault
 - Memory too full, no free frames (usually some kept free!)
 - Not enough free unallocated frames (limit value)
- Which page to replace?
 - One, that is not needed in the (near) future
 - Predict based on the past behaviour
 - Locality principle helps: no more references to the page, process passed that phase
 - Some time mistakes happen

See Fig 8.1 [Stal05]

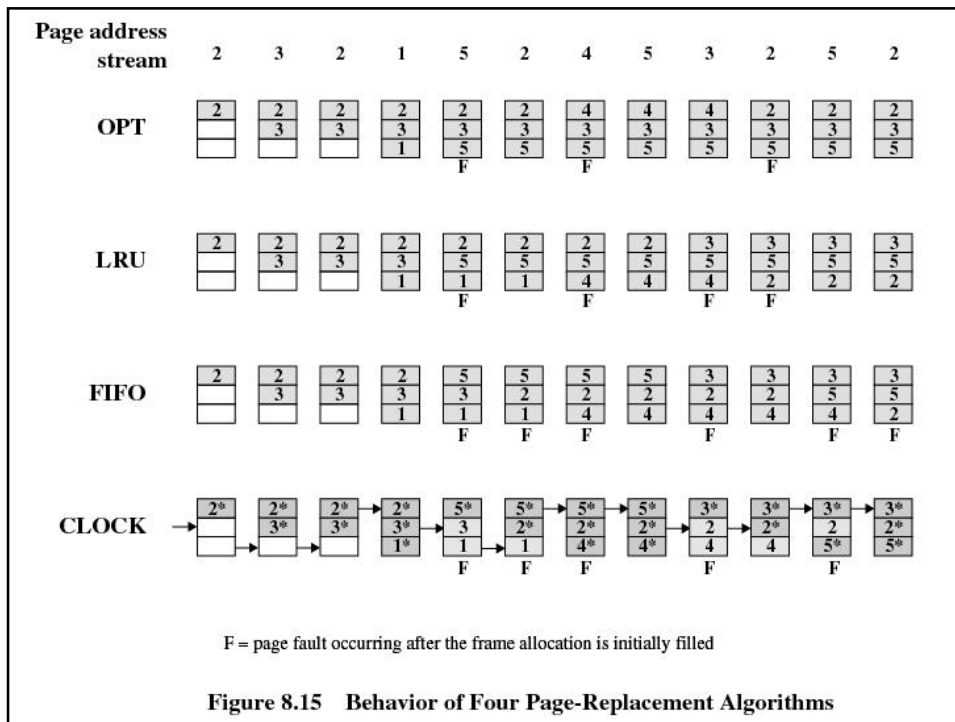
6



Page Replacement Algorithms

- Page fault forces choice
 - which page must be removed
 - make room for incoming page
- Modified page must first be saved
 - unmodified just overwritten
- Better not to choose an often used page
 - will probably need to be brought back in soon

8



Optimal Page Replacement Algorithm

- Replace page needed at the farthest point in future
 - Optimal but unrealizable
 - Clairvoyant (not possible!)
 - can be found afterwards from trace
- Used as a reference point for other algorithms
 - Nothing is better than optimal!
- Estimate by ...
 - logging page use on previous runs of process
 - although this is impractical

Least Recently Used (LRU)

- Recently used pages assumed to be used again soon
- Evict page that has been unused for longest time
 - How to find?
- Heavy bookkeeping
 - Update each reference time to the pages
 - Maintain pages in the referenced order (linked list)
- Infeasible if followed strict

11

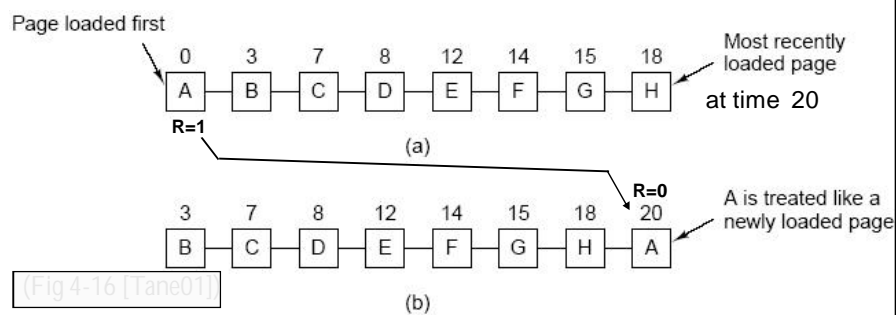
FIFO (First-In First-Out) Page Replacement Algorithm

- Replace the page that has been in the memory longest time
- Page at beginning of list replaced
- Maintain a linked list of all pages
 - In order they came into memory
 - Alternatively: store the load time on each page
- Disadvantage
 - Page in memory the longest may be often used and should not be evicted

12

Second Chance

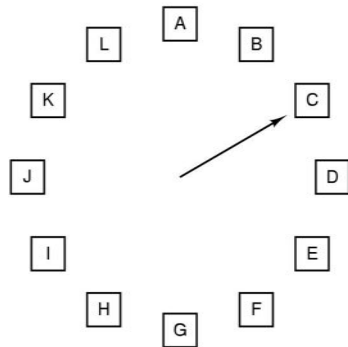
- Pages sorted in FIFO order
- Use reference bit R and modified bit M
- If the page to be evicted has been recently used (R=1) give it second chance; move to end of queue and set R=0
 - Will be evicted, if all other pages are also given second chance



Not Recently Used Page Replacement Algorithm

- Each page has Reference bit, Modified bit
 - bits are set when page is referenced, modified
 - R cleared periodically, M cleared at disk writes
- Pages are classified
 1. not referenced, not modified
 2. not referenced, modified
 3. referenced, not modified
 4. referenced, modified
- NRU removes page at random
 - from lowest numbered non empty class

The Clock Page Replacement Alg.



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

- Go through all pages in circular fashion
- Try to locate unused page with the NRU classification, used page gets second chance

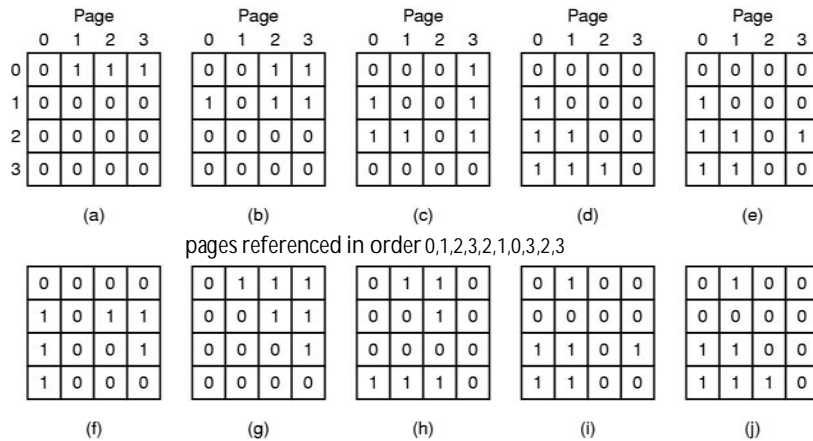
15

Least Recently Used (LRU)

- Strict alternatives:
 - Keep a linked list of pages in reference order, update this list on every memory reference !!
 - Keep counter in each page table entry
 - Use special hardware to increment the counter
 - choose page with lowest value counter
 - periodically zero the counter
- Simulating algorithms: NFU, Aging
 - NFU – Not Frequently used
 - Aging
 - In the adding: shift counter right, add R bit to leftmost
 - Forgets the history

16

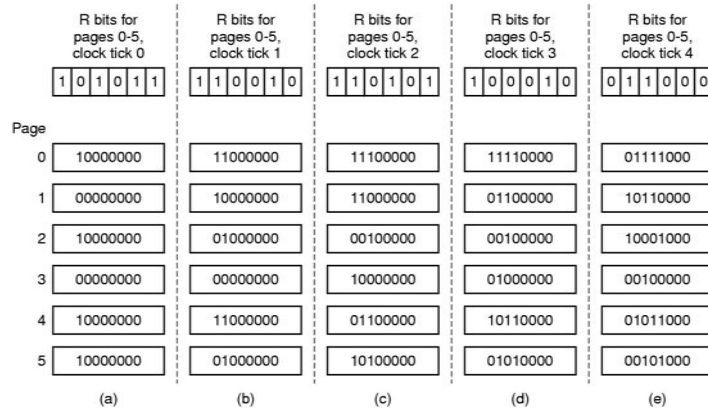
LRU hardware using matrix



- On reference: set row all 1s and column all 0s
- At any time the row with lowest binary value is LFU page

17

Aging algorithm simulates LRU



- On each clock time process the counters:
- Shift right one bit
- Set left-most bit to one on pages referenced after previous tick

18

Working set model

19

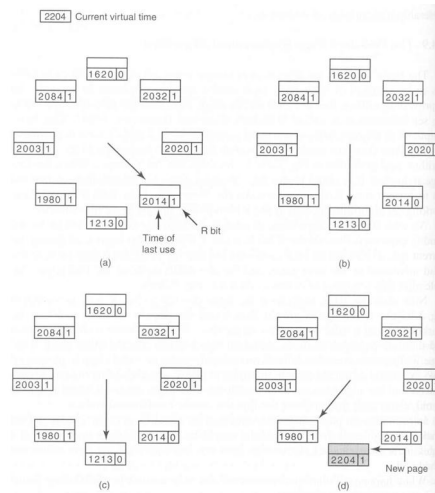
Resident Set vs. Working Set

- Working set = Set of pages the process is currently using
 - $w(k,t)$ is the size of the working set at time, t
- Resident set = Page frames currently allocated for process
- Number of allocated page frames (*sivukehysten lkm*)
 - Too small
 - more processes in the memory, more page faults
 - Too many? Trashing? (*ruuhkautuminen*)
 - Too large
 - occupy space that beneficial for other processes
 - CPU utilisation (*käyttöaste*) too low
 - "wasted space"

20

The WSClock Page Replacement Algorithm

- As clock, but only residence set
- Evict unmodified page not in working set and
- Schedule for disk write modified page not in WS
- If no candidate page found during one round, wait for writes or evict any clean (=unmodified) WS page



23

Review of Page Replacement Algorithms

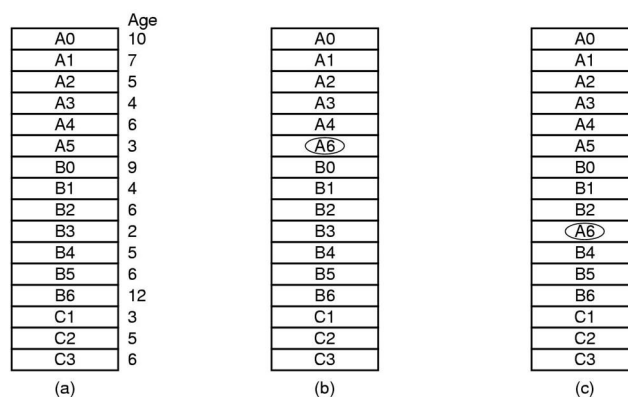
Algorithm	Comment
Optimal	Not implementable, but useful as a benchmark
NRU (Not Recently Used)	Very crude
FIFO (First-In, First-Out)	Might throw out important pages
Second chance	Big improvement over FIFO
Clock	Realistic
LRU (Least Recently Used)	Excellent, but difficult to implement exactly
NFU (Not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

24

Design Issues

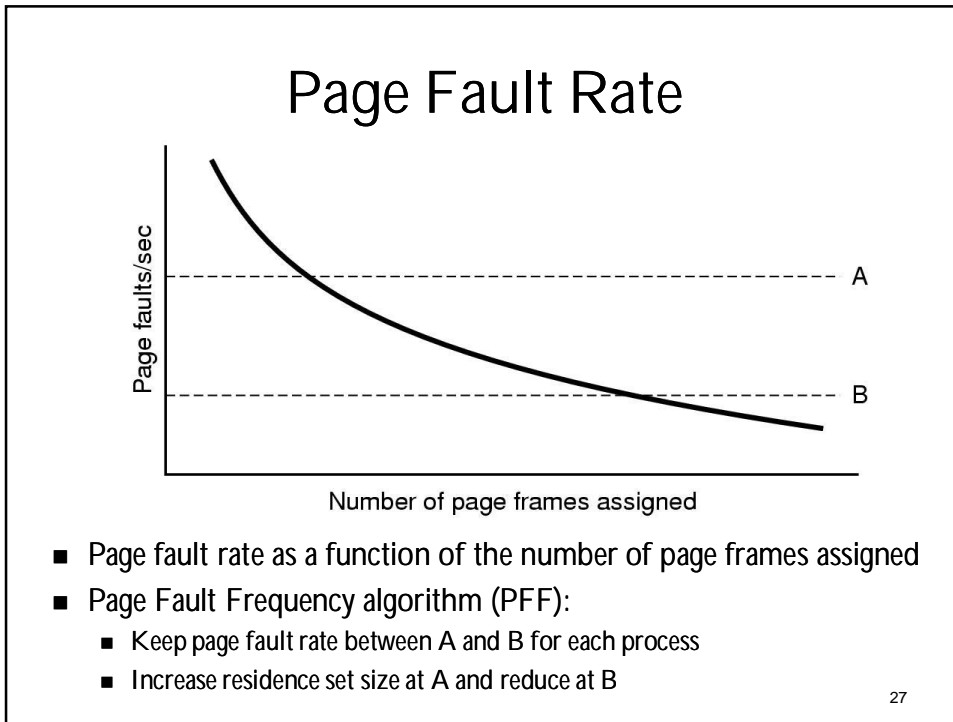
25

Design Issues for Paging Systems: Local versus Global Allocation Policies



- (a) Original configuration
- (b) Local page replacement
- (c) Global page replacement

26



Allocation and replacement

	Local Replacement	Global Replacement
Fixed Allocation	<ul style="list-style-type: none"> •Number of frames allocated to process is fixed. •Page to be replaced is chosen from among the frames allocated to that process. 	<ul style="list-style-type: none"> •Not possible.
Variable Allocation	<ul style="list-style-type: none"> •The number of frames allocated to a process may be changed from time to time, to maintain the working set of the process. •Page to be replaced is chosen from among the frames allocated to that process. 	<ul style="list-style-type: none"> •Page to be replaced is chosen from all available frames in main memory; this causes the size of the resident set of processes to vary.

Tbl 8.4 [Stal05]

28

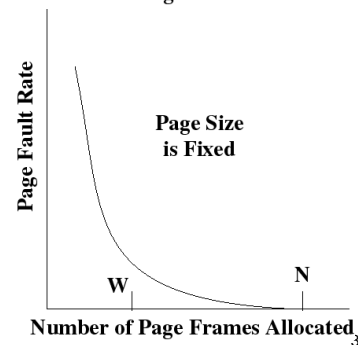
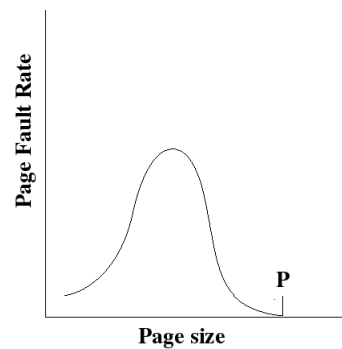
Load Control

- Despite good designs, system may still thrash
- When Page Fault Frequency (PFF) algorithm indicates
 - some processes need more memory
 - but no processes need less
- Solution :
 - Reduce number of processes competing for memory
 - swap one or more to disk, divide up pages they held
 - reconsider degree of multiprogramming

29

Page fault rate

- More small pages to the same memory space
- References from large pages more probable to go to a page not yet in memory
- References from small pages often to other pages -> often used pages selected to the memory
- Too small working set size -> larger page fault rate
- Large working set size -> nearly all pages in the memory



30

Page Size

- Reduce internal fragmentation → small
- Reduce page table size → large
- Proportional (1x, 2x, ...) to disk block size
- Optimal value is different for different programs
- Increase TLB hit ratio → large
- Different page size for different applications?
- How does MMU know the page size?

Tbl 8.2 [Stal05]

31

Table 8.2 Example Page Sizes

Computer	Page Size
Atlas	512 48-bit words
Honeywell-Multics	1024 36-bit word
IBM 370/XA and 370/ESA	4 Kbytes
VAX family	512 bytes
IBM AS/400	512 bytes
DEC Alpha	8 Kbytes
MIPS	4 kbytes to 16 Mbytes
UltraSPARC	8 Kbytes to 4 Mbytes
Pentium	4 Kbytes or 4 Mbytes
PowerPc	4 Kbytes
Itanium	4 Kbytes to 256 Mbytes

32

Page Size

- Overhead due to page table and internal fragmentation

- Where

- s = average process size in bytes
- p = page size in bytes
- e = page entry

$$overhead = \frac{s \cdot e}{p} + \frac{p}{2}$$

Annotations: $\frac{s \cdot e}{p}$ is labeled "page table space" and $\frac{p}{2}$ is labeled "internal fragmentation".

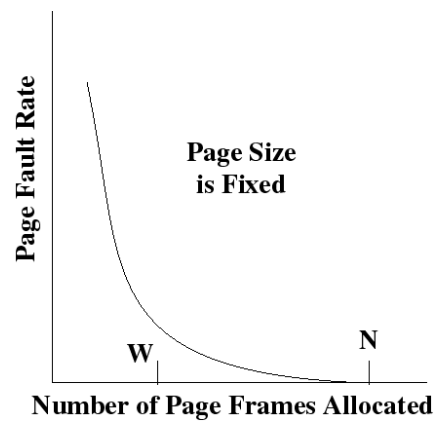
Optimized when

$$p = \sqrt{2se}$$

$s = 1 \text{ MB} \quad e = 8\text{B} \rightarrow p_{opt} = 4 \text{ KB}$
 $s = 100 \text{ MB} \quad e = 8\text{B} \rightarrow p_{opt} = 40 \text{ KB}$

Working set

- Set of pages used during last k references (window size)
- Pages not in the working set are candidates to be released or replaced
- Suitable size for the working set? Estimate based on page fault rate



$$1 \leq |W(t, \Delta)| \leq \min(\Delta, n)$$

Cleaning Policy

- Need for a background process, paging daemon
 - periodically inspects state of memory
- When too few frames are free
 - selects pages to evict using a replacement algorithm
- It can use same circular list (clock)
 - as regular page replacement algorithm but with different pointer

35

When to clean the page? (write to disk)

- Only when changed
- Demand cleaning (*tarvetalletus*)
 - Write to disk only when the page frame is needed
 - Long delay in freeing, crash?
- Precleaning (*ennaltalletus*)
 - Write to disk periodically in bigger groups
- Cleaning the freed page frame
 - Zeroing might be wasted work
 - What if the page is needed soon again?
 - unmodified free page might be reused
- What if the disk utilization is low (no work on disk)?

36

Page buffering

- Maintain certain number of free page frames
- ~ page frame cache (of freed pages frames)
 - Fast recovery from page frame buffer, if the page on freed frame is referenced
- Page marked to be cleaned
 - Add to free page frame list, if not modified
 - Add to write-to-disk list, if modified
- Page still on the same allocated page frame
 - Remove the reference from page table
- Allocate page frames from the free frames list using FIFO order

37