OPERATING SYSTEMS

Week 8

Much of the material on these slides comes from the recommended textbook by William Stallings

THE UNIVERSITY OF NEWCASTLE AUSTRALIA
FACULTY OF ENGINEERING AND BUILT ENVIRONMENT
www.newcastle.edu.au

Detailed content

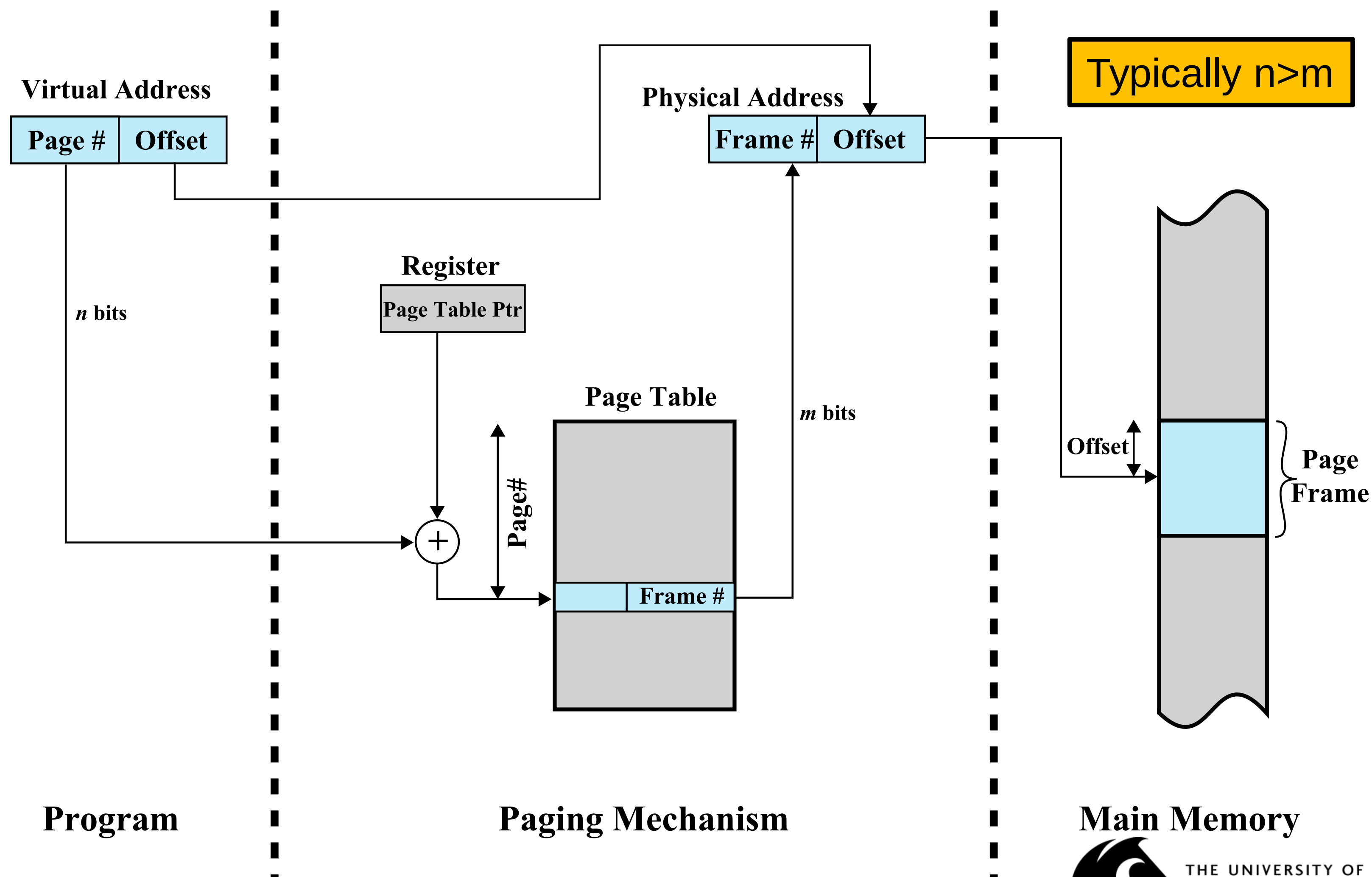
Weekly program

- ✓ Week 1 – Operating System Overview
- ✓ Week 2 – Processes and Threads
- ✓ Week 3 – Scheduling
- ✓ Week 4 – Real-time System Scheduling and Multiprocessor Scheduling
- ✓ Week 5 – Concurrency: Mutual Exclusion and Synchronization
- ✓ Week 6 – Concurrency: Deadlock and Starvation
- ✓ Week 7 – Memory Management
- ❑ **Week 8 – Memory Management II**
- ❑ Week 9 – Disk and I/O Scheduling
- ❑ Week 10 – File Management
- ❑ Week 11 – Real-world Operating Systems: Embedded and Security
- ❑ Week 12 – Revision of the course
- ❑ Week 13 – Extra revision (if needed)

Key Concepts From Last Lecture

- VM also allows a process to be broken up into pieces and
  - the pieces need not to be contiguously located in main memory
  - It is not even necessary for all of the pieces of the process to be in main memory during execution
- Two basic approaches to providing VM are
  - Paging
    - Process is divided into relatively small, fixed size blocks called pages
  - Segmentation
    - Segmentation provides for the use of pieces of varying size

Address translation



Week 08 Lecture Outline

Memory Management

- ❑ Fetch policy
- ❑ Placement policy
- ❑ Replacement policy
- ❑ Resident set management
- ❑ Cleaning policy
- ❑ Load control

H/W & S/W for virtual memory

- Virtual memory is supported by a **combination** of hardware and software.
- The hardware must provide a segmentation/paging mechanism.
- **The OS must provide algorithms for various aspects of memory managements**

OS Policies for virtual memory

- Key issue is performance
  - Minimize page faults

Fetch Policy Demand paging Prepaging **Placement Policy** **Replacement Policy** Basic Algorithms Optimal Least recently used (LRU) First-in-first-out (FIFO) Clock Page Buffering	**Resident Set Management** Resident set size Fixed Variable Replacement Scope Global Local **Cleaning Policy** Demand Precleaning **Load Control** Degree of multiprogramming

- A **good OS** should provide tools for **monitoring** and **tuning** such parameters.

- NOTE: we will mainly discuss paging rather than segmentation.

Fetch policy

- The **fetch policy** determines when a page should be brought into main memory.
  - **Demand paging** brings in a page only when a memory reference to that page is made.
  - This policy can cause a large number of page faults before a process "settles down" to a "steady state".

- **Pre-paging** brings in pages *likely to be referenced,* such as those adjacent to a referenced page.
  - When pages are stored contiguously on disk, the overhead in bringing in extra pages is small.
  - Pre-paging can be triggered by page faults.

- For most purposes, **demand paging** seems to be best.

Placement policy

- **Placement policy** determines **where** a new page/segment is to be placed.
  - For paging systems, the location of a page does not effect performance in any way.
  - For a segmented memory, the issues are the same as discussed under "simple segmentation“
    - **best-fit:** choose the block with minimum wasted space.
    - **first-fit:** choose the first block (from the top of memory) into which the process will fit.
    - **next-fit:** choose the first block (from last point) into which the process will fit

Replacement policy

- Deals with the selection of a page in main memory to be replaced when a new page must be brought in
  - Which one to replace?
    - Objective is that the page that is removed be the page least likely to be referenced in the near future

- Because of the principle of locality often there is a high correlation between recent referencing history and near-future referencing patterns.

- The more elaborate the replacement policy the greater the hardware and software overhead to implement it

Frame locking

- When a frame is locked the page currently stored in that frame may not be replaced
  - Kernel of the OS as well as key control structures are held in locked frames
  - I/O buffers and time-critical areas may be locked into main memory frames
  - Locking is achieved by associating a lock bit with each frame

Replacement policies

- **Optimal**
  - Choose the page for which the next reference is furthermost in the future.
    - it can be shown that this generates the least number of page faults.
    - it is <u>not practical</u> because of the overhead involved in the lookahead.
    - it is a standard by which other methods can be judged: how close is your method to optimal?

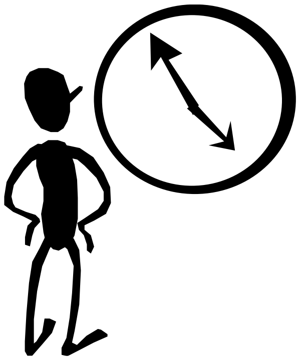
- **Least Recently Used (LRU)**
  - Choose the page which has not been referenced for the longest time.
    - The locality principle suggests that this is a good policy.
    - It can be proven to be <u>almost as good as optimal</u>.
    - There are many implementation difficulties: we have to tag each page with the time of the last reference.

Replacement policies

- **First In First Out (FIFO)**
  - simple to implement: one pointer per page, arranged in a FIFO queue.
  - removes the page which has been in memory the longest.
  - does not recognise that some pages are referenced more frequently than others.
  - performs badly.

Replacement Policies

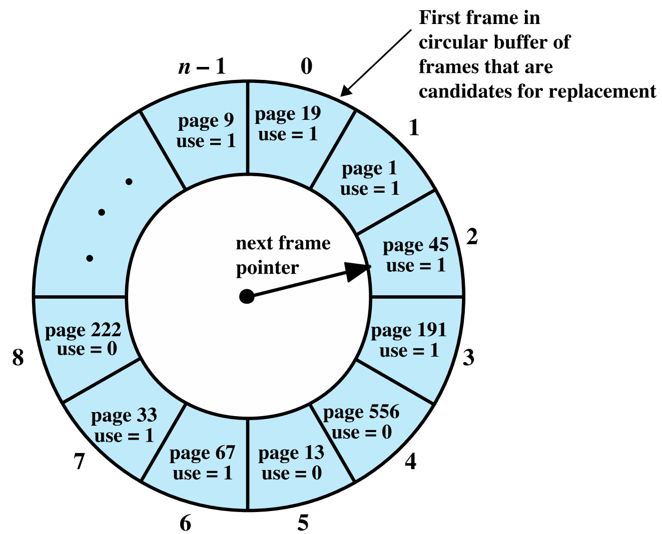
- **Clock policy**
  - Requires the association of an additional bit with each frame
    - referred to as the **use bit**
  - When a page is first loaded in memory or referenced, the use bit is set to 1
  - The set of frames is considered to be a circular buffer
  - Any frame with a use bit of 1 is passed over by the algorithm
  - Page frames visualized as laid out in a circle



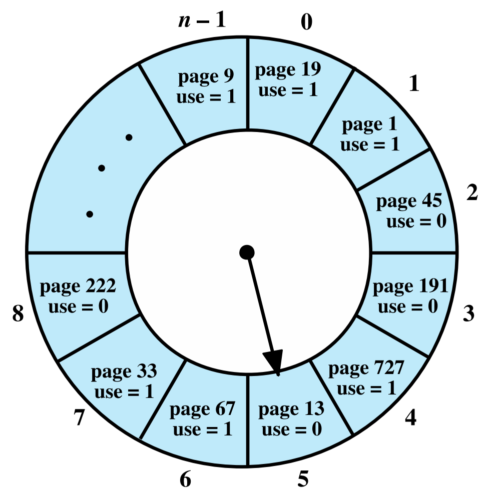
THE UNIVERSITY OF NEWCASTLE AUSTRALIA

Replacement Policies

- Clock policy



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



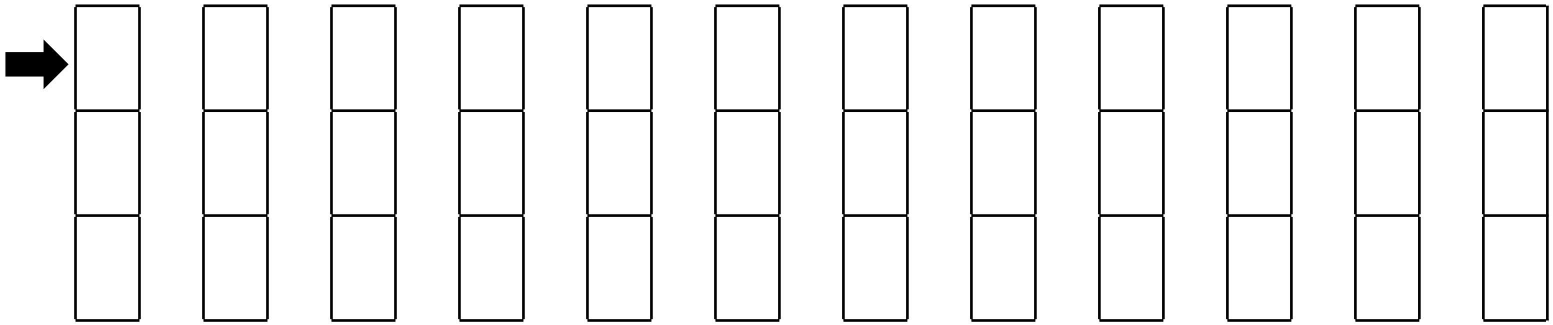
(b) State of buffer just after the next page replacement

Figure 8.15 Example of Clock Policy Operation

Clock policy example

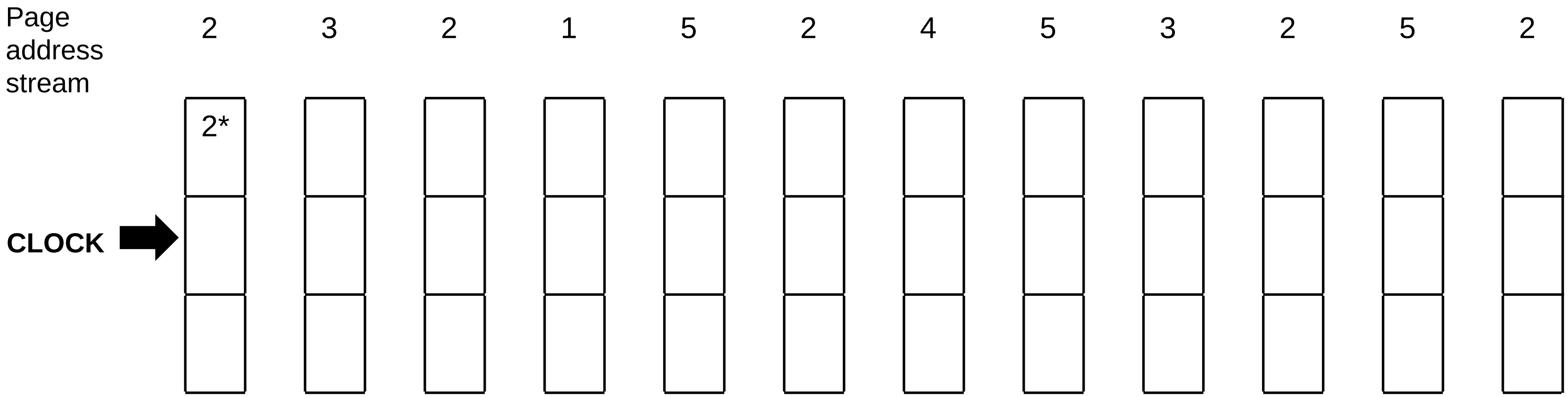
Page address stream: 2 3 2 1 5 2 4 5 3 2 5 2

CLOCK



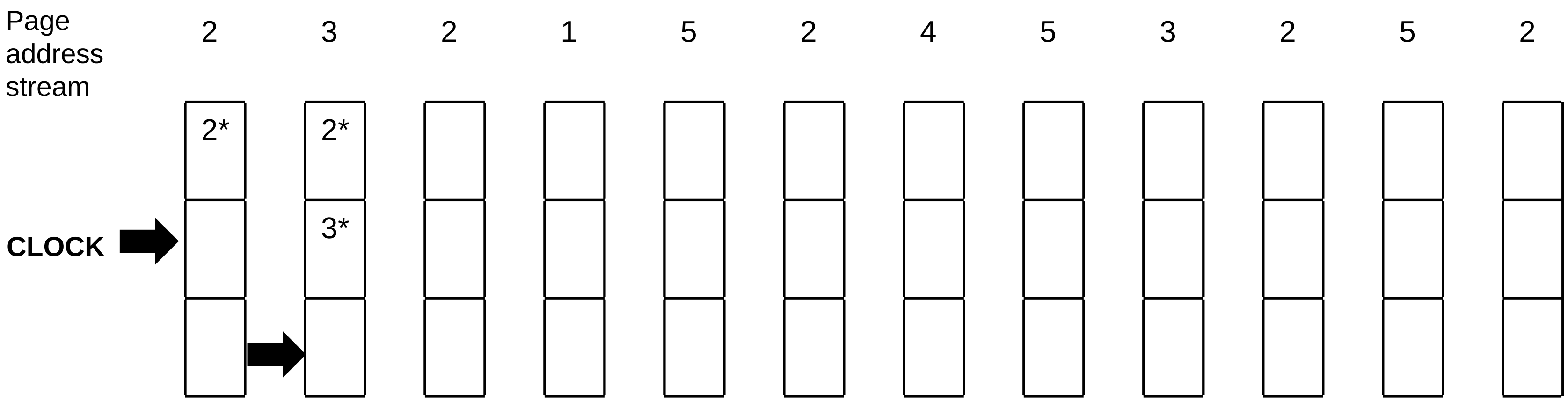
* use bit = 1

Clock policy example



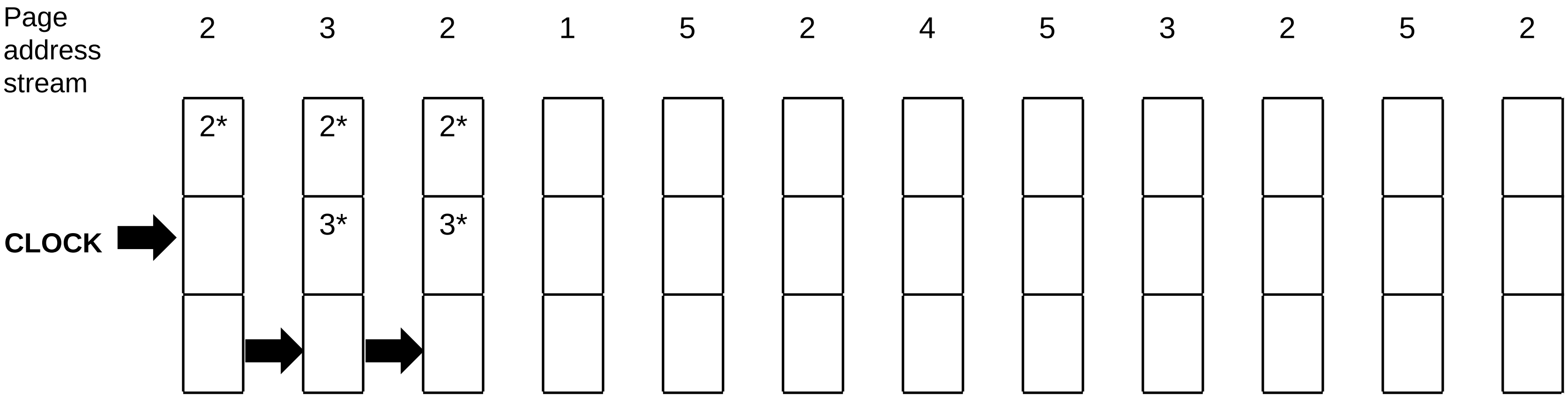
* use bit = 1

Clock policy example



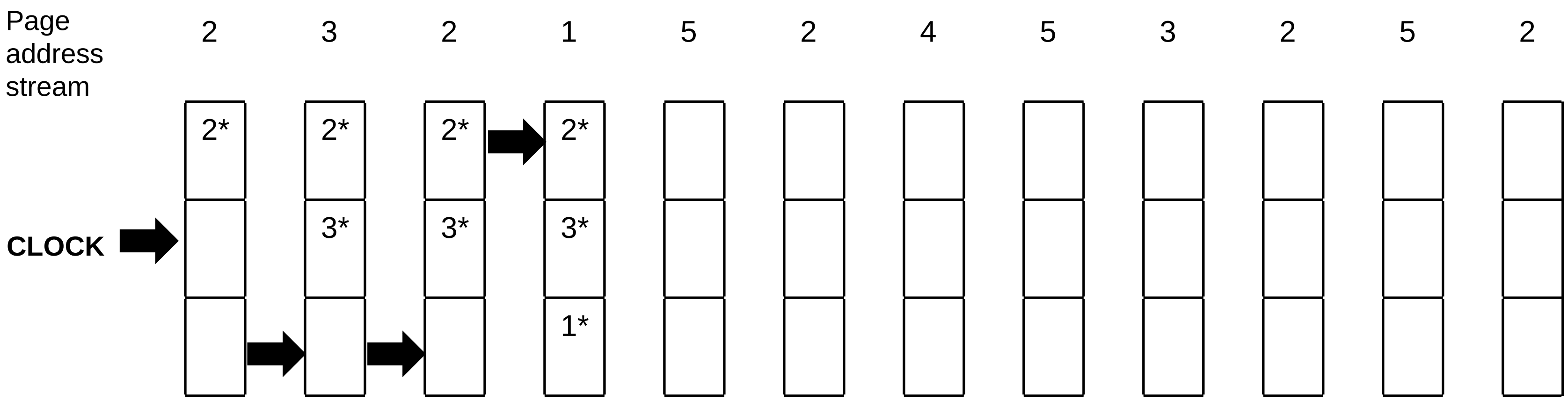
* use bit = 1

Clock policy example



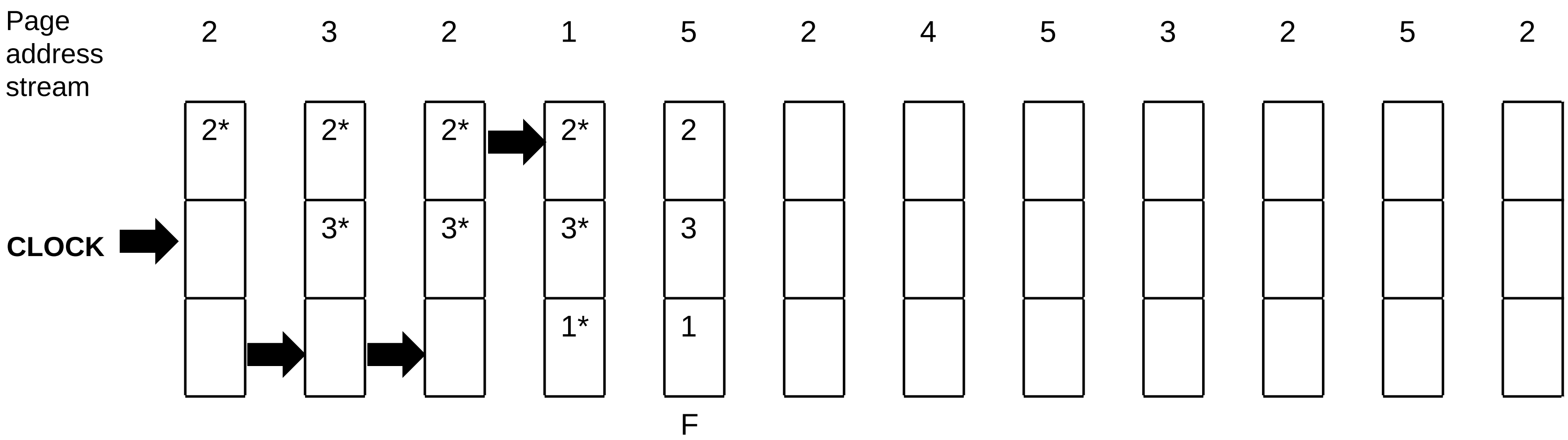
* use bit = 1

Clock policy example



* use bit = 1

Clock policy example

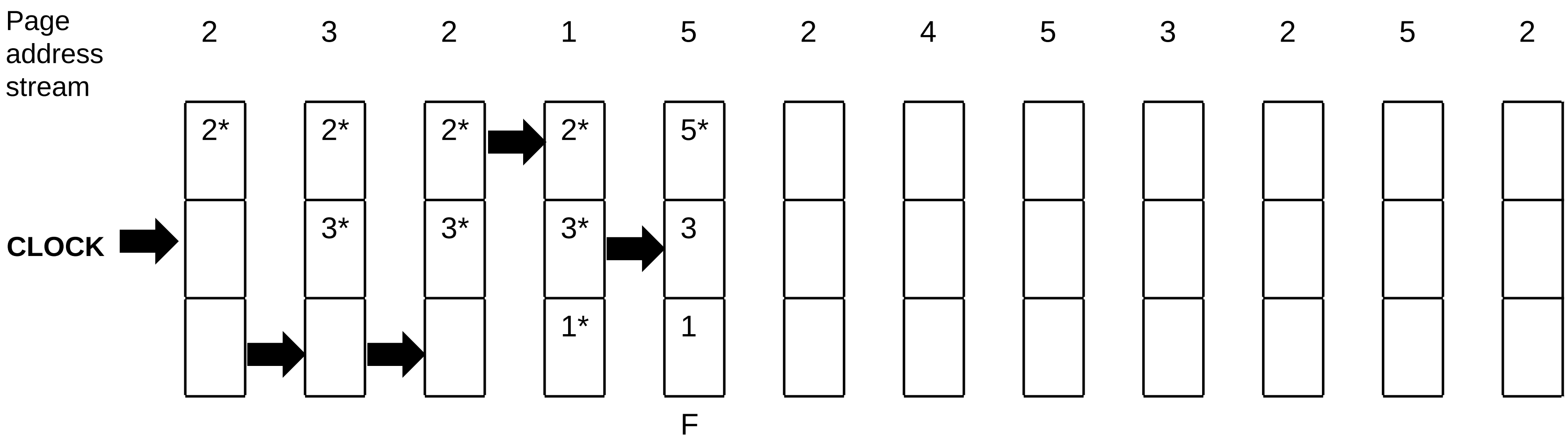


* use bit = 1

F = page fault

- When it comes time to replace a page, the operating system scans the buffer to find a frame with a use bit set to 0.
- Each time it encounters a frame with a use bit of 1, it resets that bit to 0 and continues on.

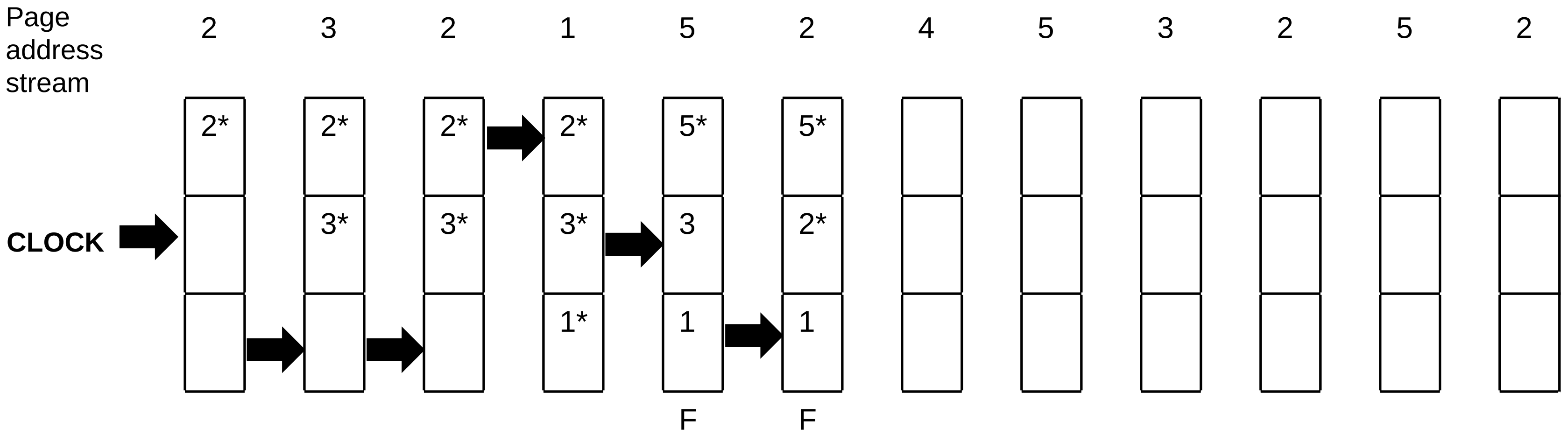
Clock policy example



* use bit = 1
F = page fault

- When it comes time to replace a page, the operating system scans the buffer to find a frame with a use bit set to 0.
- Each time it encounters a frame with a use bit of 1, it resets that bit to 0 and continues on.

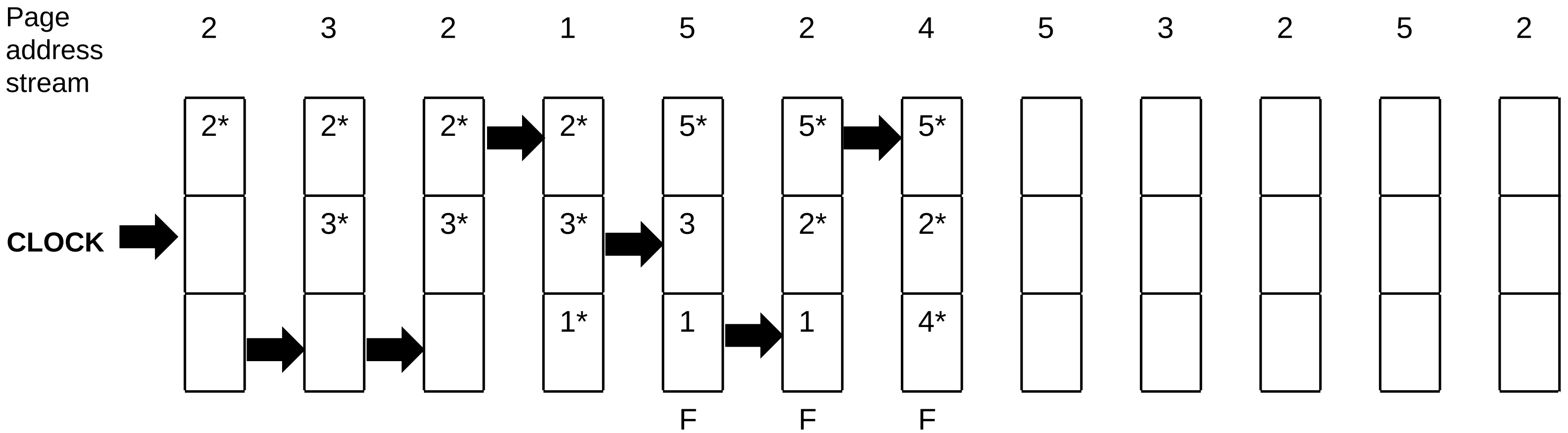
Clock policy example



* use bit = 1

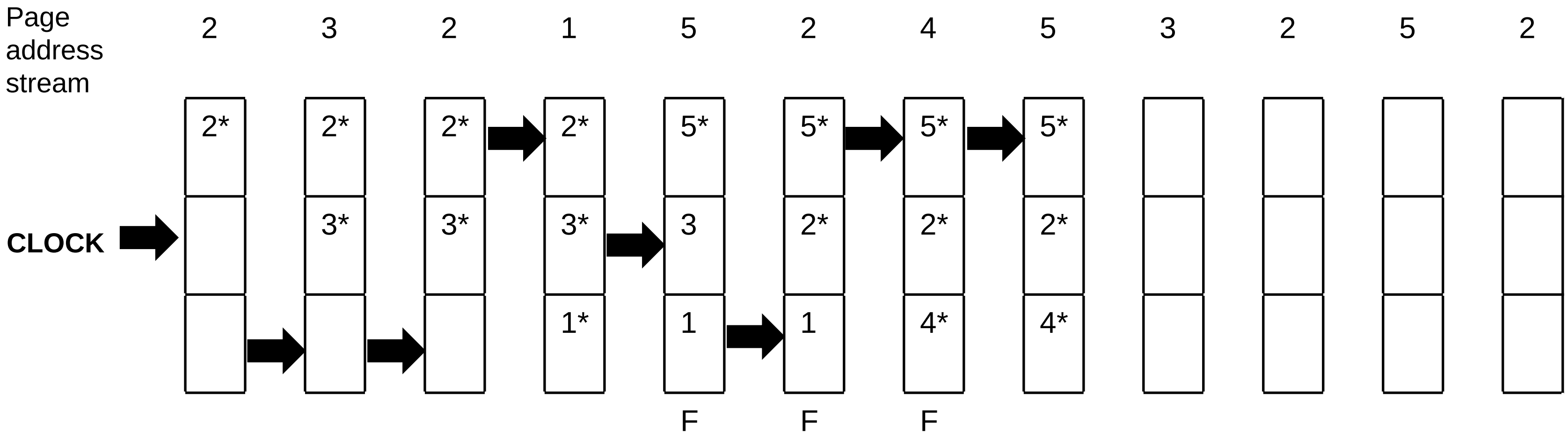
F = page fault

Clock policy example



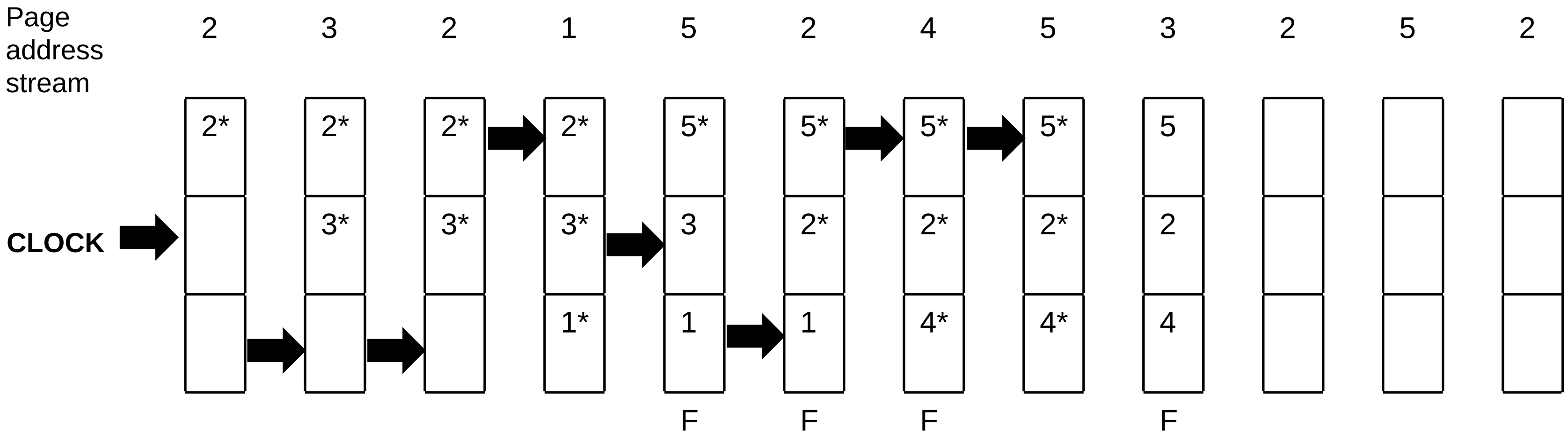
* use bit = 1
F = page fault

Clock policy example



* use bit = 1
F = page fault

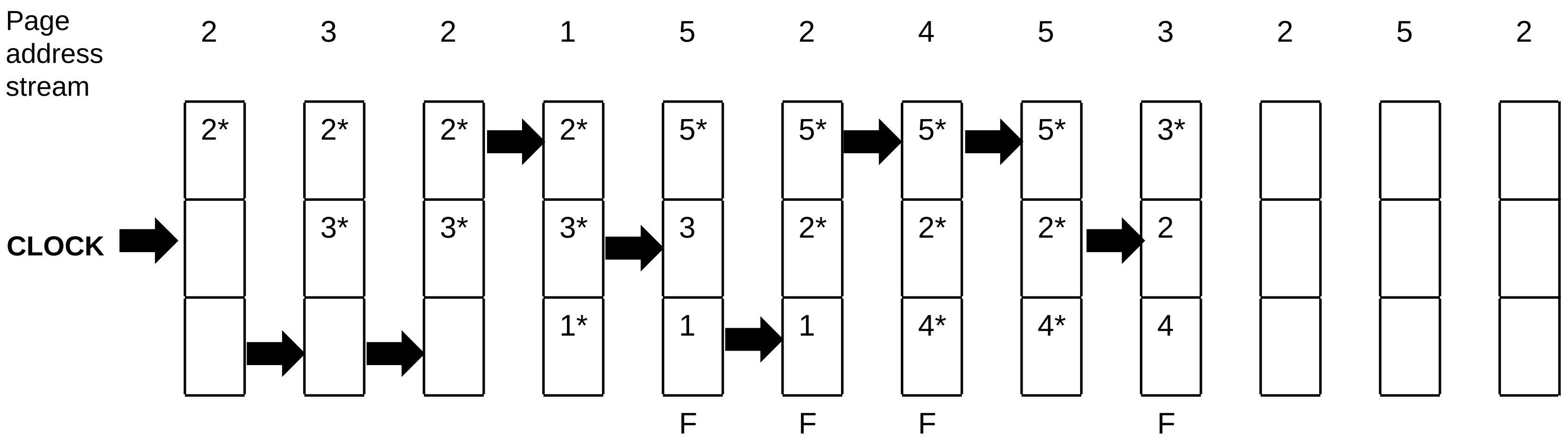
Clock policy example



* use bit = 1

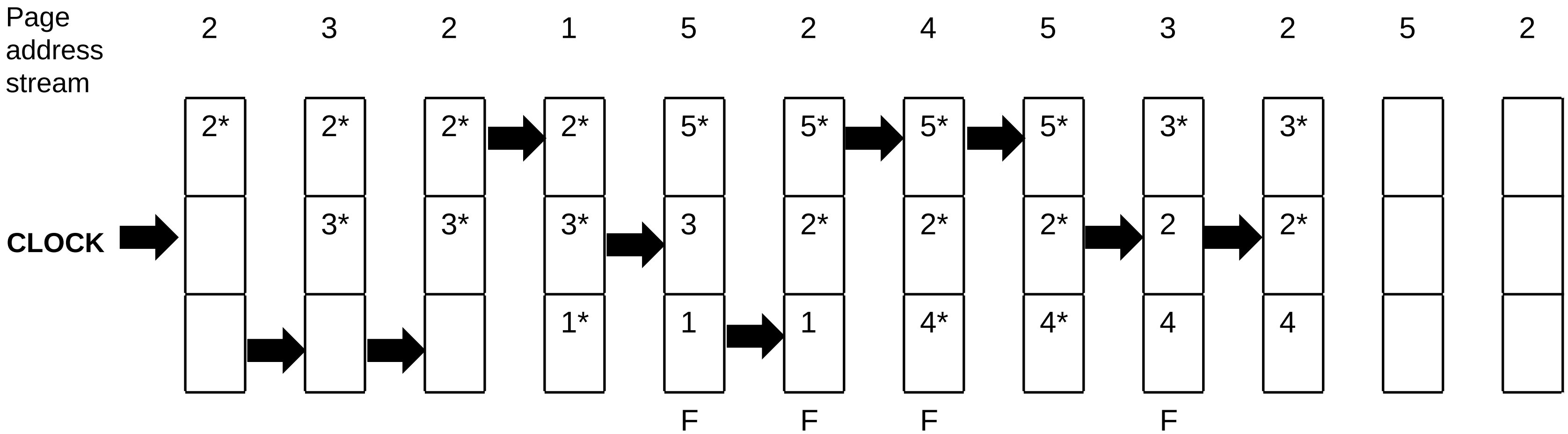
F = page fault

Clock policy example



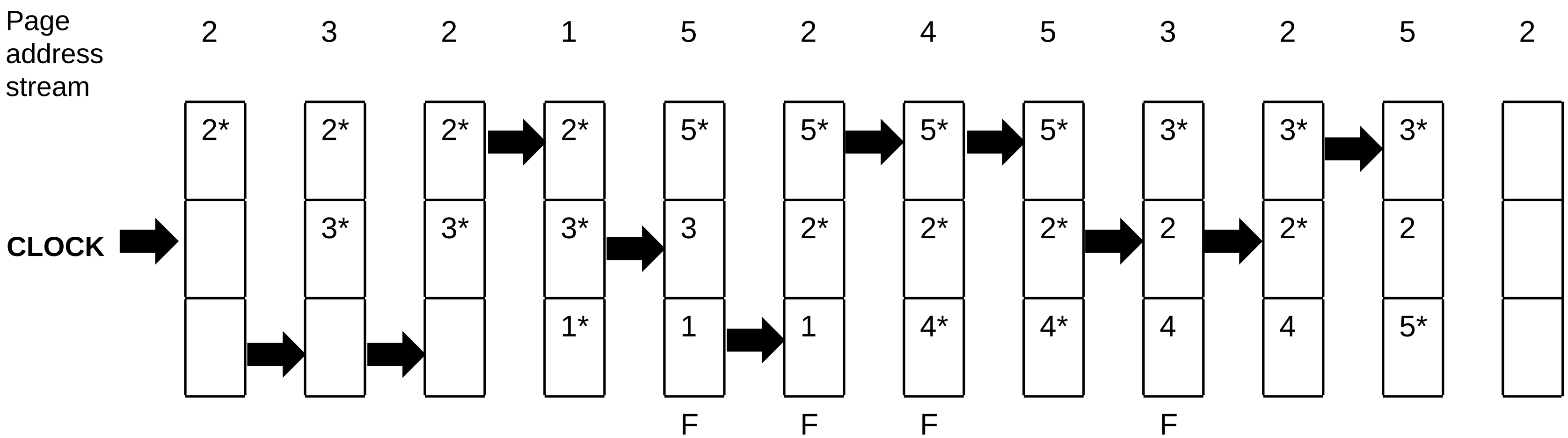
* use bit = 1
F = page fault

Clock policy example



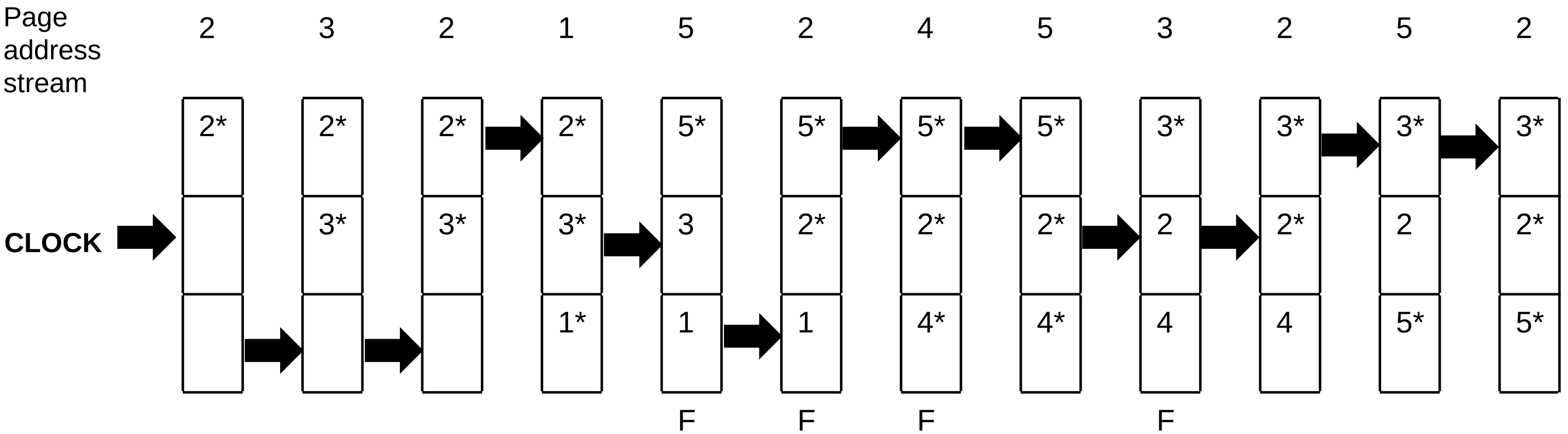
* use bit = 1
F = page fault

Clock policy example



* use bit = 1
F = page fault

Clock policy example



* use bit = 1
F = page fault

Combined examples

Page address stream	2	3	2	1	5	2	4	5	3	2	5	2
OPT	2	2 3	2 3	2 3 1	2 3 5 F	2 3 5	4 3 5 F	4 3 5	4 3 5	2 3 5 F	2 3 5	2 3 5
LRU	2	2 3	2 3	2 3 1	2 5 1 F	2 5 1	2 5 4 F	2 5 4	3 5 4 F	3 5 2 F	3 5 2	3 5 2
FIFO	2	2 3	2 3	2 3 1	5 3 1 F	5 2 1 F	5 2 4 F	5 2 4	3 2 4 F	3 2 4	3 5 4 F	3 5 2 F
CLOCK	2*	2* 3*	2* 3*	2* 3* 1*	5* 3 1 F	5* 2* 1 F	5* 2* 4* F	5* 2* 4*	3* 2 4 F	3* 2* 4	3* 2 5* F	3* 2* 5*

F = page fault occurring after the frame allocation is initially filled

Figure 8.14 Behavior of Four Page-Replacement Algorithms

Comparison of algorithms

Based on execution of 0.25×10^6 references in a FORTRAN program
Page size 256 words

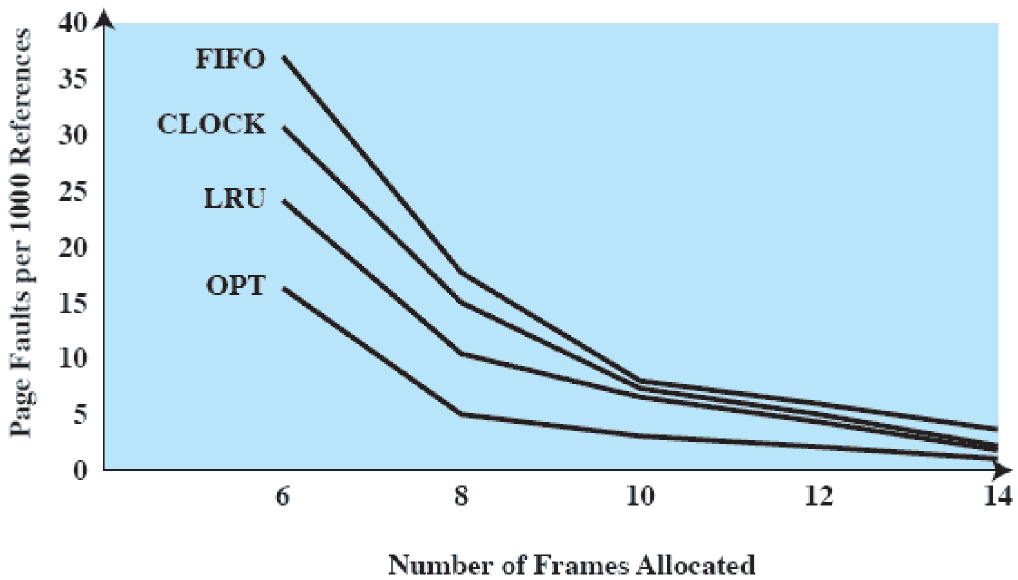


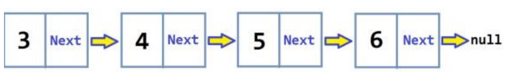
Figure 8.17 Comparison of Fixed-Allocation, Local Page Replacement Algorithms

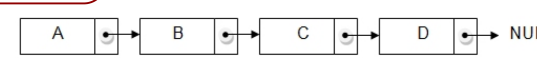
Page Buffering

- Improves paging performance and allows the use of a simpler page replacement policy

A replaced page is not lost, but rather assigned to one of two lists

- Free page list
  - list of page frames available for reading in pages
- Modified page list
  - pages are written out in clusters





Resident set management

- The OS must decide how many pages to bring into main memory
  - The smaller the amount of memory allocated to each process, the more processes can reside in memory
  - Small number of pages loaded increases page faults
  - Beyond a certain size, further allocations of frames will not effect the page fault rate
    - Principle of locality

Resident set size

- **Fixed-allocation**
  - gives a process a fixed number of frames in main memory within which to execute
    - Based on process type and guidance from programmer/system admin
  - when a page fault occurs, one of the pages of that process must be replaced

- **Variable-allocation**
  - allows the number of page frames allocated to a process to be varied over the lifetime of the process
    - Process with high page fault rate, more page is allocated
    - Process with exceptionally low page fault rate allocation is reduced
    - More powerful scheme but incurs more overhead

Replacement scope

- The scope of a replacement strategy can be categorized as ***global*** or ***local***
  - both types are activated by a page fault when there are no free page frames
- **Local**
  - chooses only among the resident pages of the process that generated the page fault
- **Global**
  - considers all unlocked pages in main memory
- No convincing evidence that local policies perform better than global policies

Resident set management

	Local Replacement	Global Replacement
Fixed Allocation	•Number of frames allocated to a process is fixed. •Page to be replaced is chosen from among the frames allocated to that process.	•Not possible.
Variable Allocation	•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.	•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.

Fixed allocation, local scope

- Necessary to decide ahead of time the amount of allocation to give a process

- If allocation is too small, there will be a high page fault rate
- If allocation is too large, there will be too few programs in main memory
  - increased processor idle time
  - increased time spent in swapping

Variable allocation, global scope

- Easiest to implement
  - Adopted in a number of operating systems

- OS maintains a list of free frames

- Free frame is added to resident set of process when a page fault occurs

- If no frames are available the OS must choose a page currently in memory – This is a challenge

- One way to counter potential problems is to use page buffering

Variable allocation, local scope

- When a new process is loaded into main memory, allocate to it a certain number of page frames as its resident set
- When a page fault occurs, select the page to replace from among the resident set of the process that suffers the fault
- Reevaluate the allocation provided to the process and increase or decrease it to improve overall performance
  - Assessment of likely future demands
- More complex but better performance

Cleaning policy

- Concerned with determining when a modified page should be written out to secondary memory

Demand Cleaning

a page is written out to secondary memory only when it has been selected for replacement

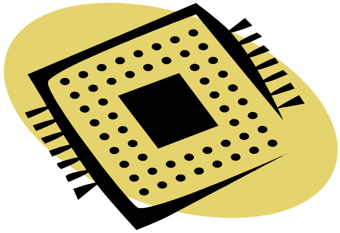
↓

Precleaning

allows the writing of pages in batches

Load control

- Determines the number of processes that will be resident in main memory
  - *multiprogramming* level
- Critical in effective memory management
- Too few processes, many occasions when all processes will be blocked and much time will be spent in swapping
- Too many processes will lead to thrashing



Multiprogramming

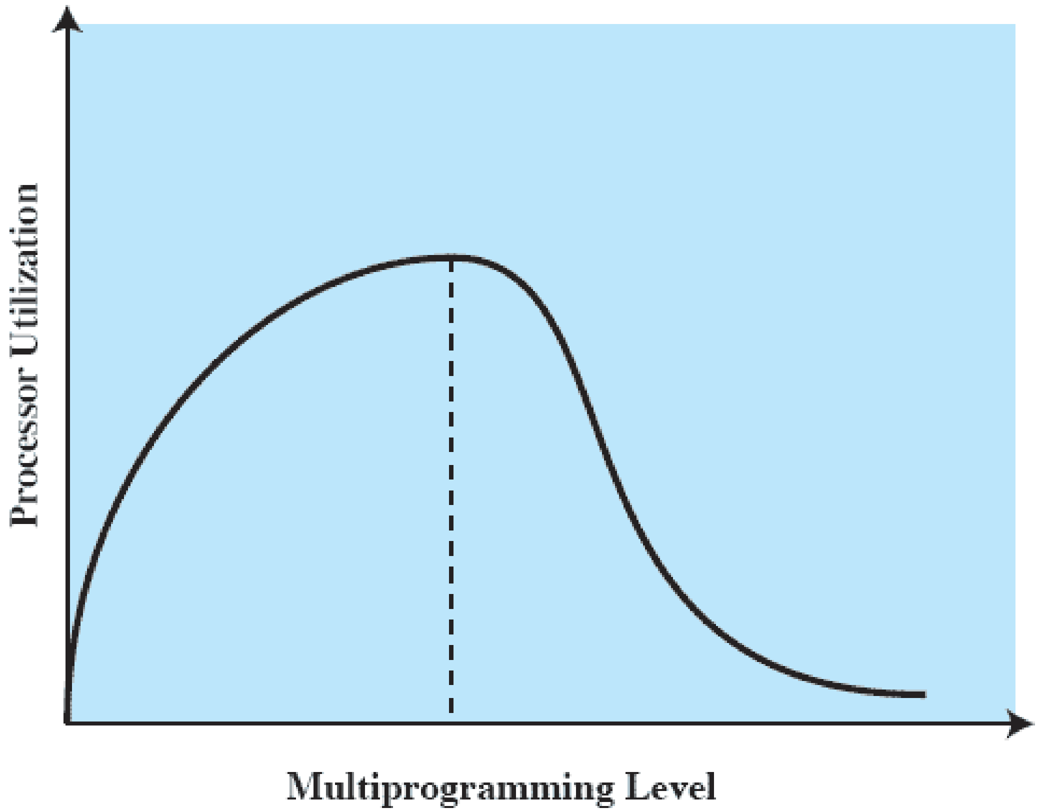


Figure 8.21 Multiprogramming Effects

Process suspension

- If the degree of multiprogramming is to be reduced, one or more of the currently resident processes must be swapped out

Six possibilities exist:

- Lowest-priority process
- Faulting process
- Last process activated
- Process with the smallest resident set
- Largest process
- Process with the largest remaining execution window

Summary of OS policies for VM

- Task of memory management in paging environment is very complex
- Changing a single policy may have noticeable effect on the overall performance
- Performance of a set of policies depends
  - Main memory size
  - Relative speed of main and secondary memory
  - Size and number of processes competing for resources
  - Execution behaviour of individual programs
- Thus there is no single set of policy that can be declared as the best

Summary

- A Number of design issues relate to OS support for memory management:
- **The OS must provide**
  - a policy for **fetching** a page into main memory: *demand*, or *pre-paging*
  - a policy for **placing** a page in memory
  - a policy for **replacing** a (or swapping out) a page in memory; there are several methods: optimal, LRU, FIFO, clock
  - a **resident set management policy**:
  - a **cleaning policy**: when should a modified page be written to disk?
    - only when replaced? or as soon as they are modified?
  - a **load control mechanism** for controlling the amount of page faults.

References

- **Operating Systems – Internal and Design Principles**
  - By William Stallings
- Chapter 8