Operating Systems and Concurrency

Lecture 17: Memory Management VI

G52OSC

Geert De Maere and Isaac Triguero
{Geert.DeMaere,Isaac.Triguero}@Nottingham.ac.uk

University Of Nottingham
United Kingdom

2018

Goals for Today

Overview

- Several **key decisions** have to be made when **using virtual memory**
  - **What pages** are **removed** from memory $\Rightarrow$ page replacement algorithms
  - **How many pages** are allocated to a process and are they **local or global**
  - **When** are pages **removed** from memory $\Rightarrow$ paging daemons
- What **problems** may occur in virtual memory $\Rightarrow$ thrashing

Page Replacement

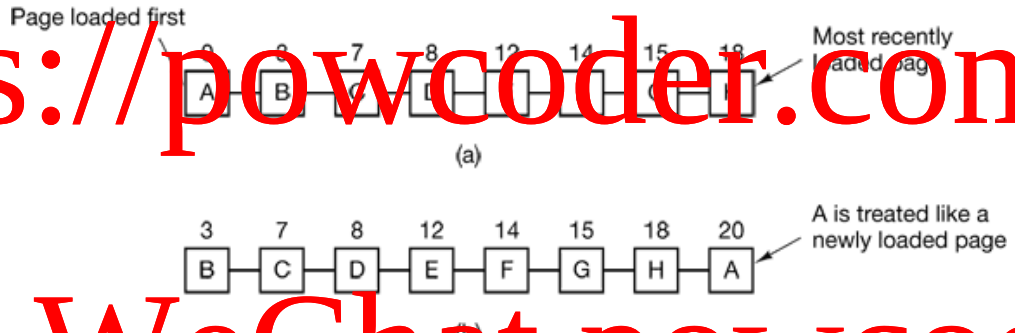
First-In, First-Out (FIFO)

- FIFO maintains a **linked list** and **new pages** are added at the end of the list
- The **oldest page** at the **head of the list** is evicted when a page fault occurs
- The **(dis-)advantages** of FIFO include:
  - It is **easy** to understand/implement
  - It **performs poorly** $\Rightarrow$ heavily used pages are just as likely to be evicted as a lightly used pages

Page Replacement

Second Chance FIFO

- Second chance is a **modification of FIFO**:
  - If a page at the front of the list has **not been referenced** it is **evicted**
  - If the reference bit is set, the page is **placed at the end** of list and its **reference bit reset**



- The **(dis-)advantages** of second chance FIFO include:
  - It **works better** than standard FIFO
  - **Relatively simple**, but it is **costly to implement** because the list is constantly changing (pages have to be added to the end of the list again)
  - It **can degrade to FIFO** if all pages were initially referenced

Page Replacement

The Clock Replacement Algorithm

- The **second-chance implementation** can be improved by **maintaining the page list as a circle** (this is the only difference)
  - A **pointer** points to the last "visited" page
  - In this form the algorithm is called (one handed) clock
  - It is faster, but can still be **slow if the list is long**
- The **time spent** on maintaining the list **is reduced**

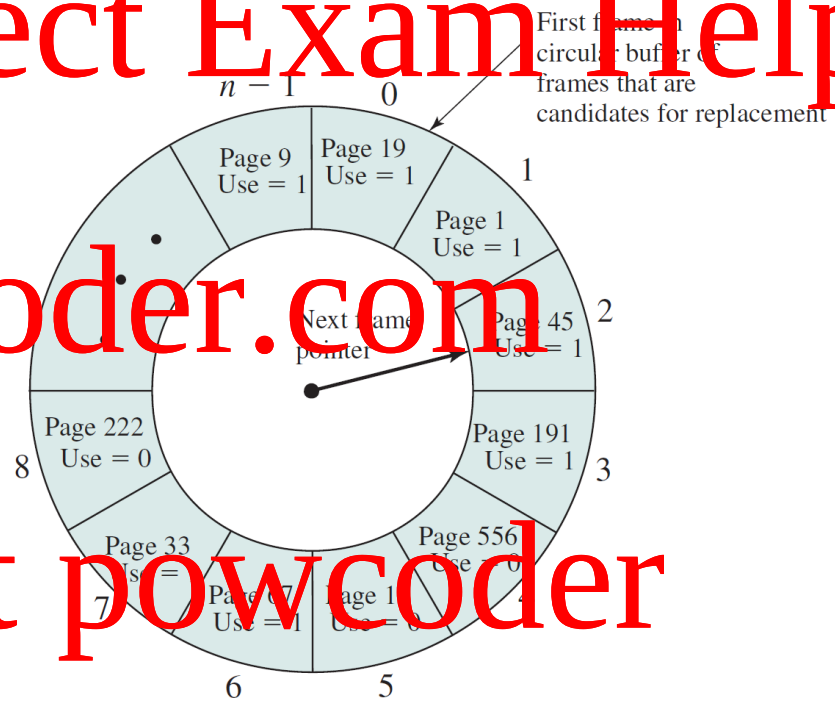


Figure: Clock Replacement Algorithm (Stallings)

Page Replacement

Not Recently Used (NRU)

- **Referenced** and **modified** bits are kept in the page table
  - Referenced bits are set to 0 at the start, and **reset periodically** (e.g. system clock interrupt or when searching the list)
- Four different **page "types"** exist
  - class 0: not referenced recently, not modified
  - class 1: not referenced recently, modified
  - class 2: referenced recently, not modified
  - class 3: referenced recently, modified

Page Replacement

Not Recently Used (NRU, Cont'ed)

- **Page table entries** are inspected upon every **page fault**.
- We could implement this as a clock in the following way:
  1. Find a page from **class 0** to be removed.
  2. If step 1 fails, scan again looking for **class 1**. During this scan, set the reference bit to 0 on each page that is bypassed.
  3. If step 2 fails, start again from step 1 (Now we should find elements from class 2 and 3 that have been moved to class 0 or 1).
- The NRU algorithm provides a **reasonable performance** and is easy to understand and implement

Page Replacement

Least-Recently-Used

- Least recently used **evicts the page** that has **not been used the longest**
  - The OS must **keep track** of when a page was **last used**
  - Every **page table entry** contains a **field for the counter**
  - This is **not cheap** to implement as we need to maintain a **list of pages** which are **sorted** in the order in which they have been used (or search for the page)
- The algorithm can be **implemented in hardware** using a **counter** that is incremented after each instruction

Page Replacement

Least-Recently-Used

- Assume we have a system with eight logical address pages & four physical page frames
- Consider the following **page references** in order:

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

- The number of **page faults** that are generated is **12**

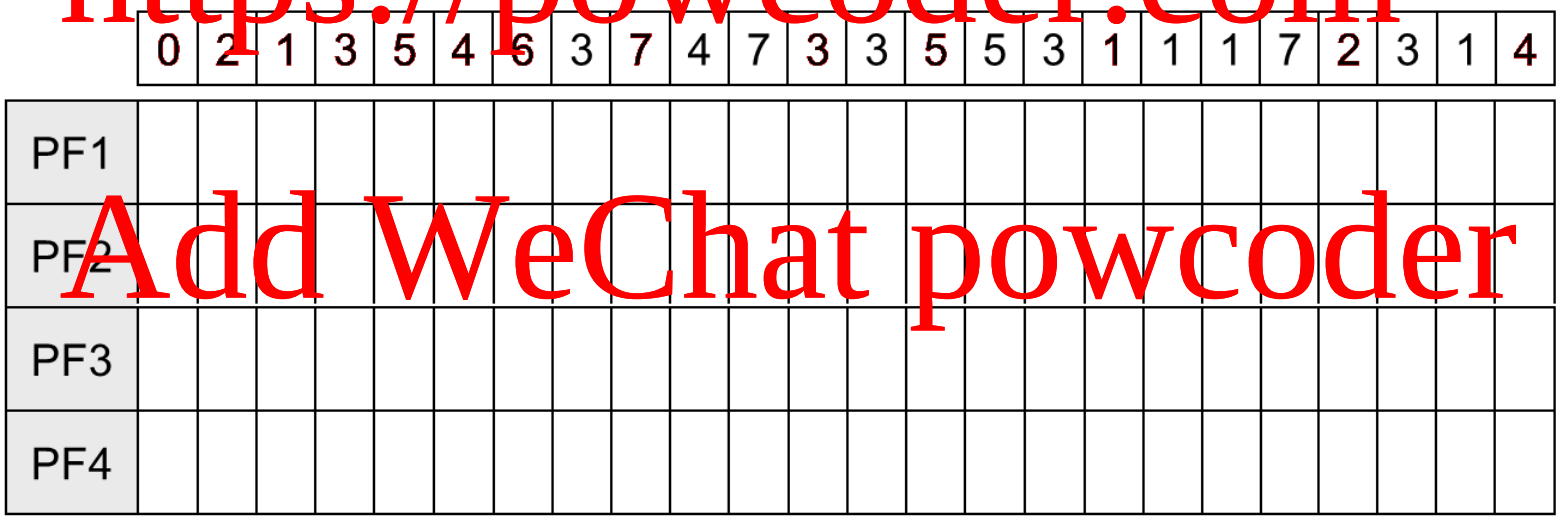


Figure: Least Recently Used

Page Replacement

Least-Recently-Used

- Assume we have a system with eight logical address pages & four physical page frames
- Consider the following **page references** in order:

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

- The number of **page faults** that are generated is **12**

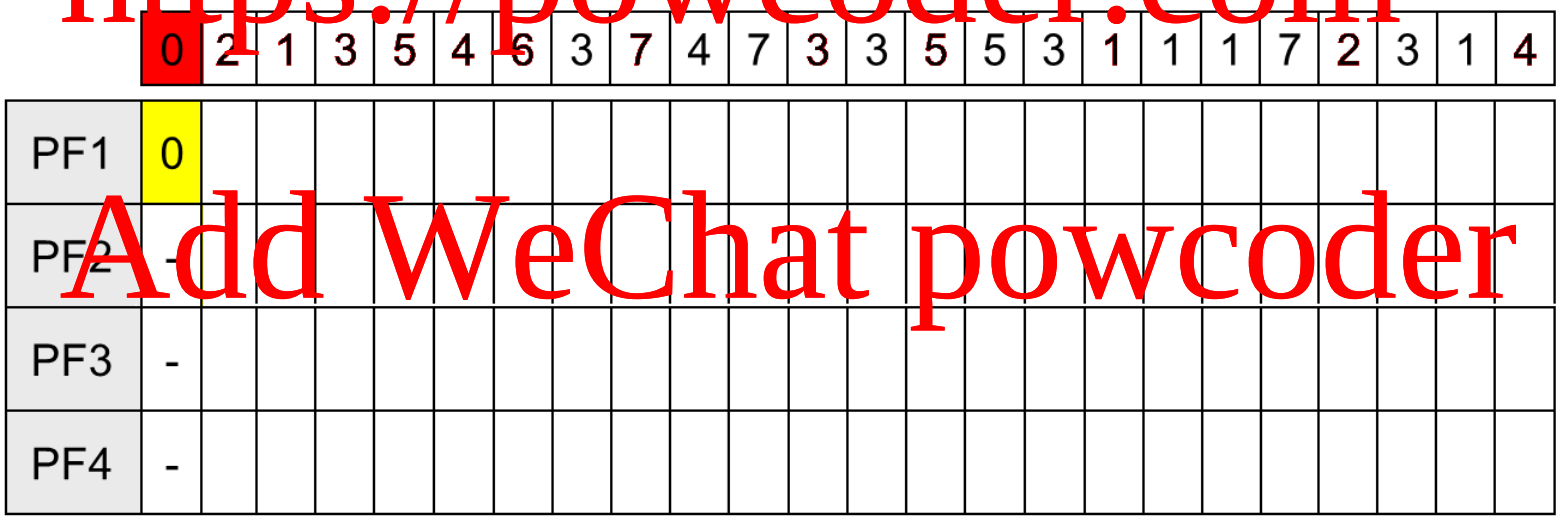


Figure: Least Recently Used

Page Replacement

Least-Recently-Used

- Assume we have a system with eight logical address pages & four physical page frames
- Consider the following **page references** in order:

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

- The number of **page faults** that are generated is **12**

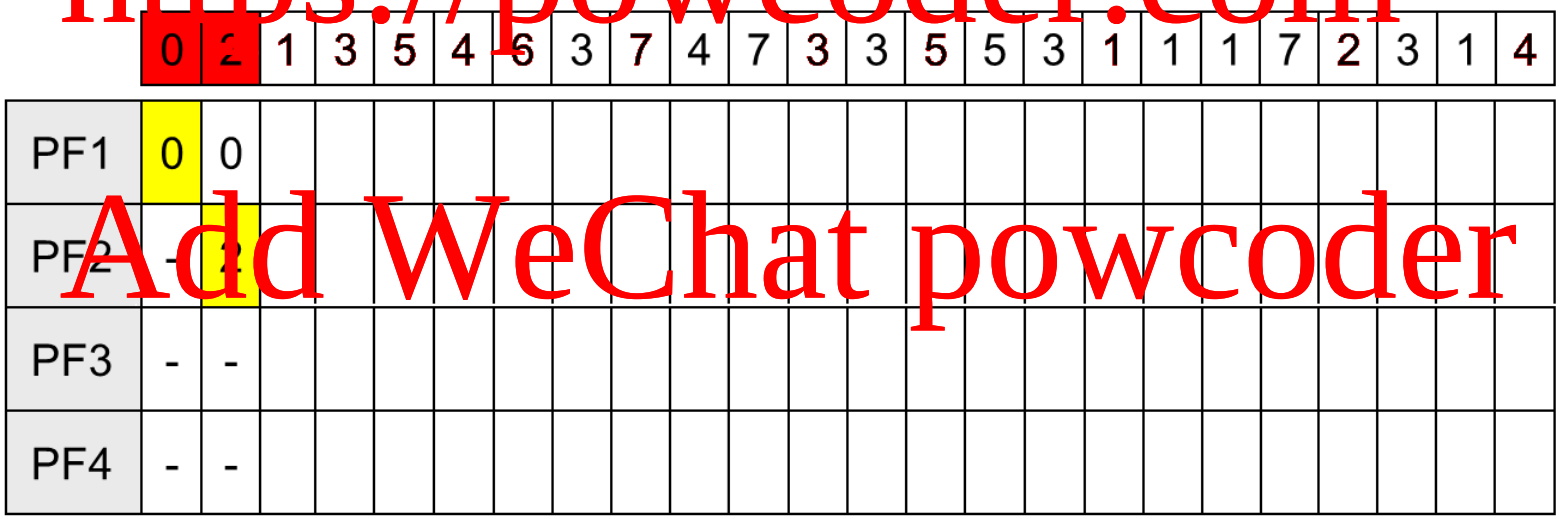


Figure: Least Recently Used

Page Replacement

Least-Recently-Used

- Assume we have a system with eight logical address pages & four physical page frames
- Consider the following **page references** in order:

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

- The number of **page faults** that are generated is **12**

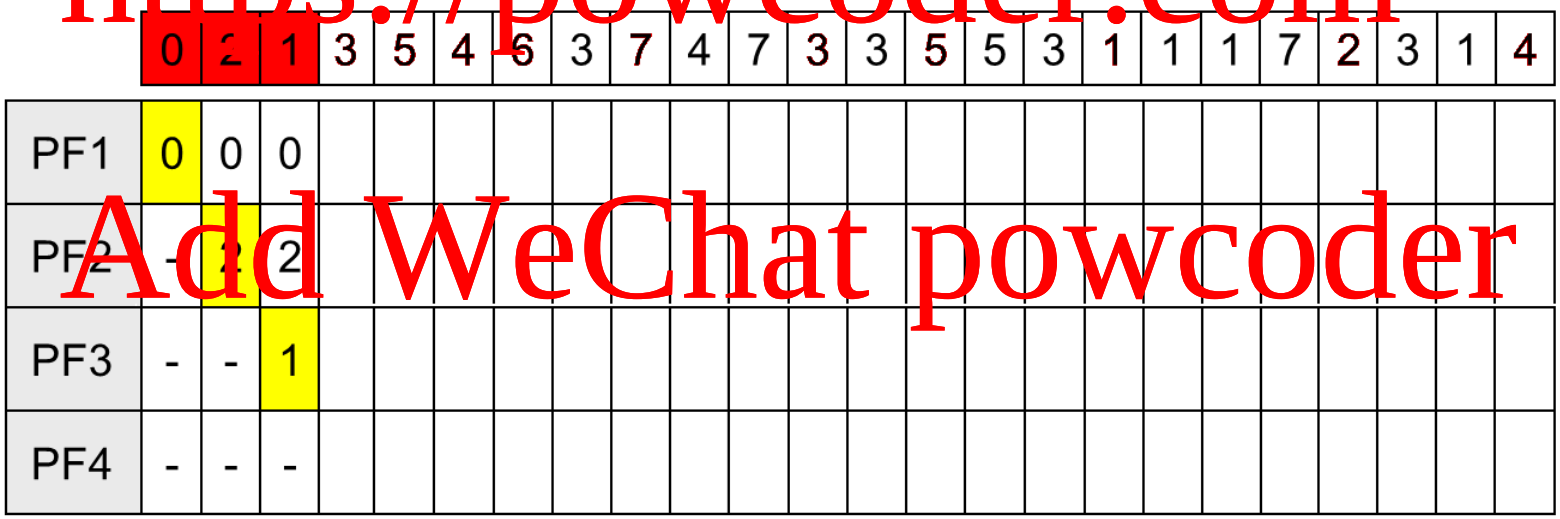


Figure: Least Recently Used

Page Replacement

Least-Recently-Used

- Assume we have a system with eight logical address pages & four physical page frames
- Consider the following **page references** in order:

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

- The number of **page faults** that are generated is **12**

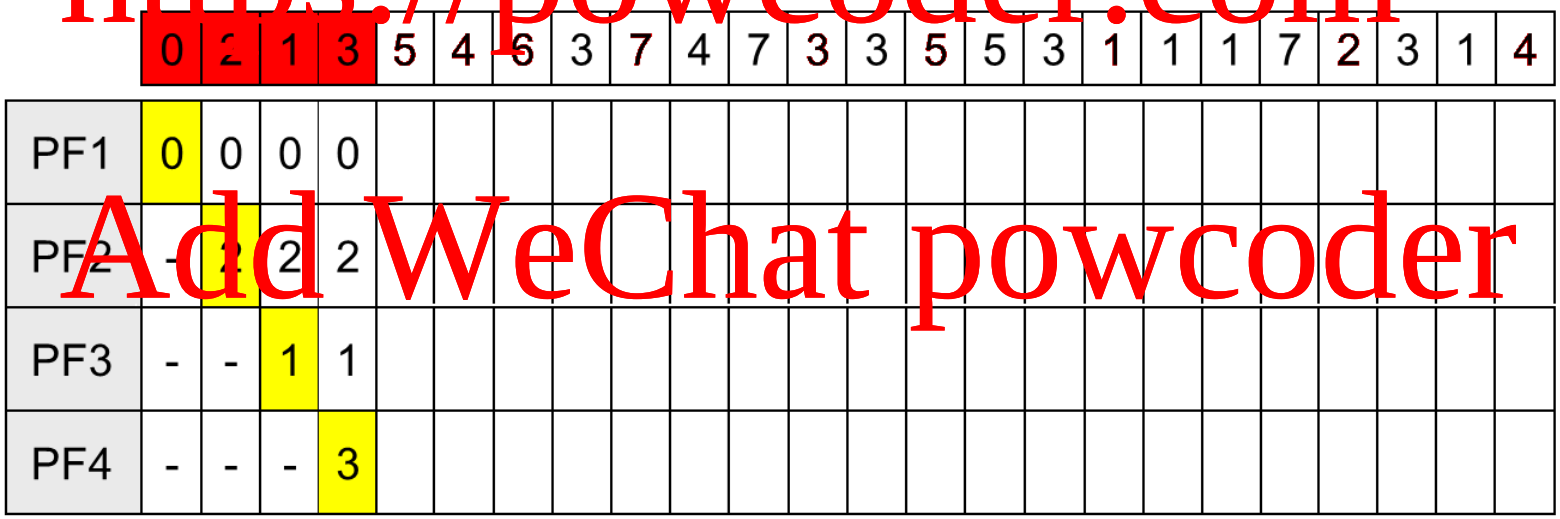


Figure: Least Recently Used

Page Replacement

Least-Recently-Used

- Assume we have a system with eight logical address pages & four physical page frames
- Consider the following **page references** in order:

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

- The number of **page faults** that are generated is **12**

	0	2	1	3	5	4	6	3	7	4	7	3	3	5	5	3	1	1	1	7	2	3	1	4
PF1	0	0	0	0	5																			
PF2	-	2	2	2	2																			
PF3	-	-	1	1	1																			
PF4	-	-	-	3	3																			

Figure: Least Recently Used

Page Replacement

Least-Recently-Used

- Assume we have a system with eight logical address pages & four physical page frames
- Consider the following **page references** in order:

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

- The number of **page faults** that are generated is **12**

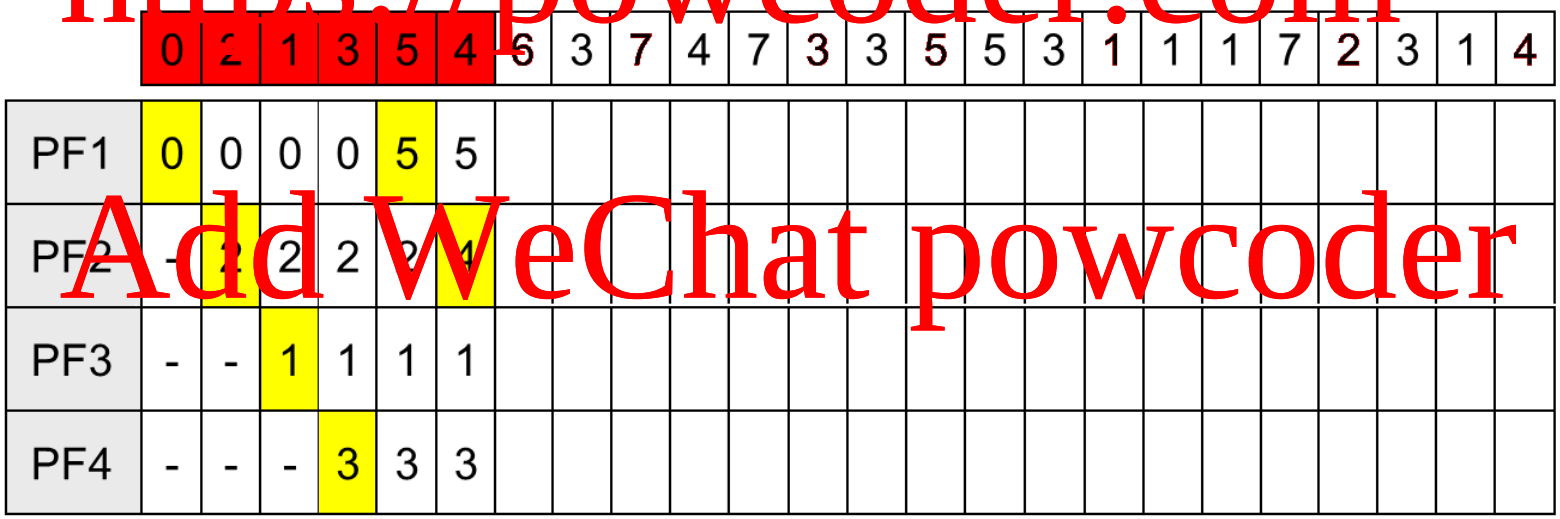


Figure: Least Recently Used

Page Replacement

Least-Recently-Used

- Assume we have a system with eight logical address pages & four physical page frames
- Consider the following **page references** in order:

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

- The number of **page faults** that are generated is **12**

	0	2	1	3	5	4	6	3	7	4	7	3	3	5	5	3	1	1	1	7	2	3	1	4
PF1	0	0	0	0	5	5	5																	
PF2	-	2	2	2	2	4	4																	
PF3	-	-	1	1	1	1	6																	
PF4	-	-	-	3	3	3	3																	

Figure: Least Recently Used

Page Replacement

Least-Recently-Used

- Assume we have a system with eight logical address pages & four physical page frames
- Consider the following **page references** in order:

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

- The number of **page faults** that are generated is **12**

	0	2	1	3	5	4	6	3	7	4	7	3	3	5	5	3	1	1	1	7	2	3	1	4
PF1	0	0	0	0	5	5	5	5																
PF2	-	2	2	2	2	4	4	4																
PF3	-	-	1	1	1	1	6	6																
PF4	-	-	-	3	3	3	3	3																

Figure: Least Recently Used

Page Replacement

Least-Recently-Used

- Assume we have a system with eight logical address pages & four physical page frames
- Consider the following **page references** in order:

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

- The number of **page faults** that are generated is **12**

	0	2	1	3	5	4	6	3	7	4	7	3	3	5	5	3	1	1	1	7	2	3	1	4
PF1	0	0	0	0	5	5	5	5	7															
PF2	-	2	2	2	2	4	4	4	4															
PF3	-	-	1	1	1	1	6	6	6															
PF4	-	-	-	3	3	3	3	3	3															

Figure: Least Recently Used

Page Replacement

Least-Recently-Used

- Assume we have a system with eight logical address pages & four physical page frames
- Consider the following **page references** in order:

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

- The number of **page faults** that are generated is **12**

	0	2	1	3	5	4	6	3	7	4	7	3	3	5	5	3	1	1	1	7	2	3	1	4
PF1	0	0	0	0	5	5	5	5	7	7	7	7	7											
PF2	-	2	2	2	2	4	4	4	4	4	4	4	4											
PF3	-	-	1	1	1	1	6	6	6	6	6	6	6											
PF4	-	-	-	3	3	3	3	3	3	3	3	3	3											

Figure: Least Recently Used

Page Replacement

Least-Recently-Used

- Assume we have a system with eight logical address pages & four physical page frames
- Consider the following **page references** in order:

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

- The number of **page faults** that are generated is **12**

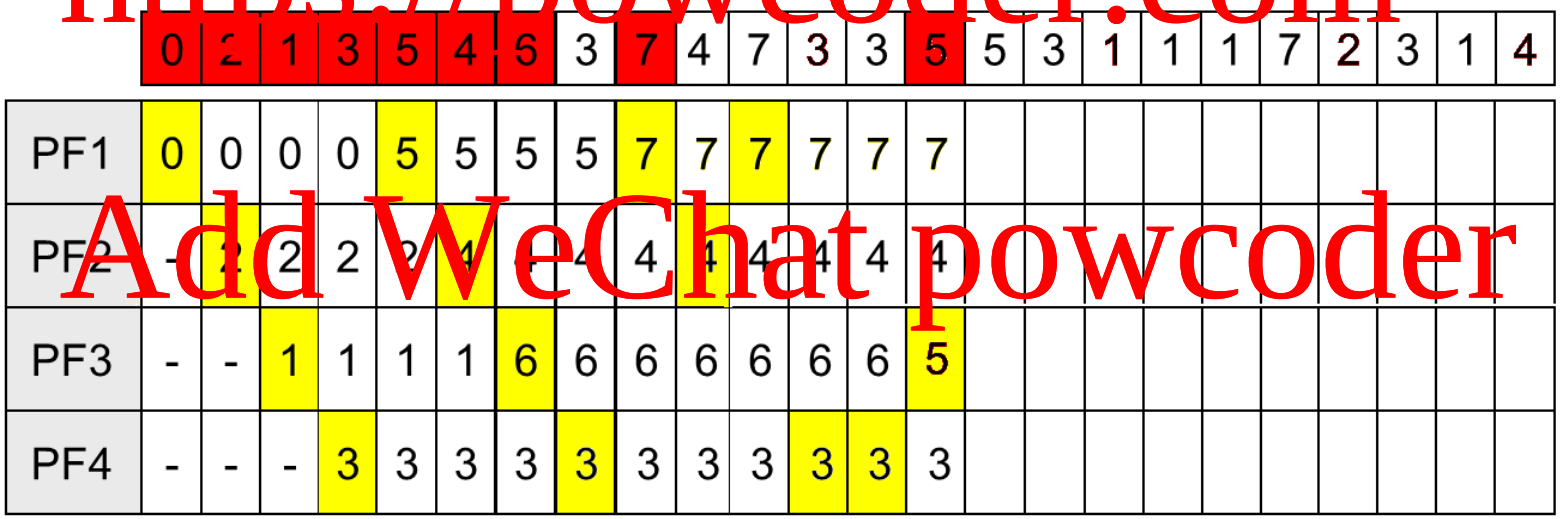


Figure: Least Recently Used

Page Replacement

Least-Recently-Used

- Assume we have a system with eight logical address pages & four physical page frames
- Consider the following **page references** in order:

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

- The number of **page faults** that are generated is **12**

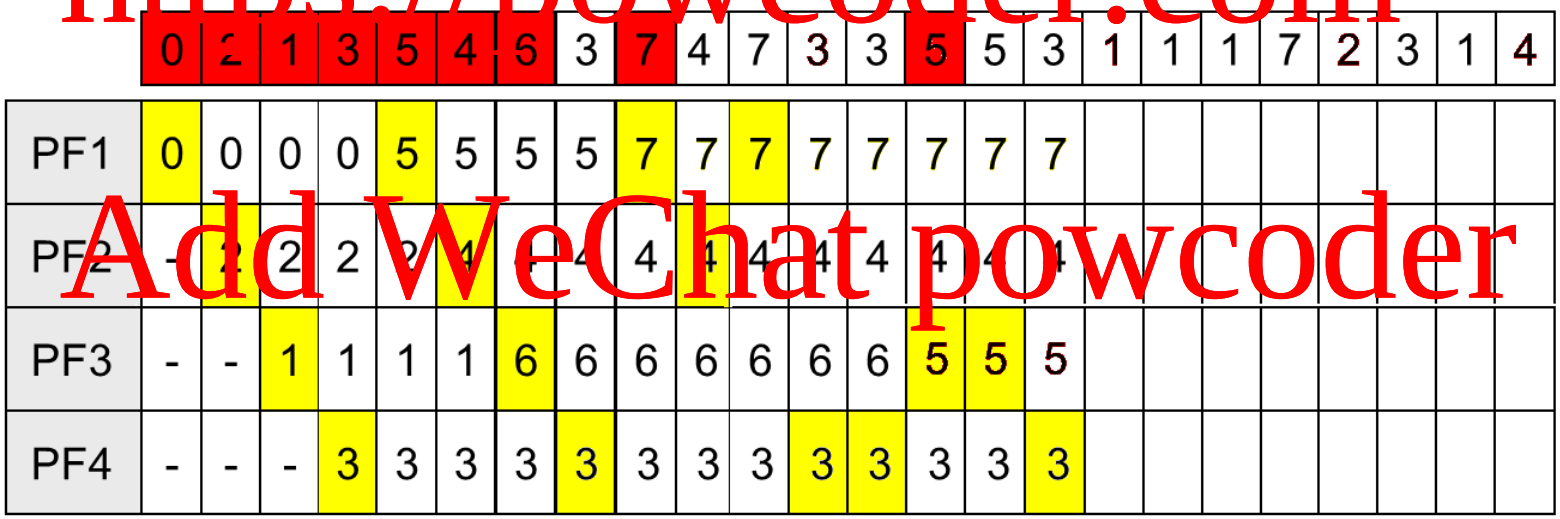


Figure: Least Recently Used

Page Replacement

Least-Recently-Used

- Assume we have a system with eight logical address pages & four physical page frames
- Consider the following **page references** in order:

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

- The number of **page faults** that are generated is **12**

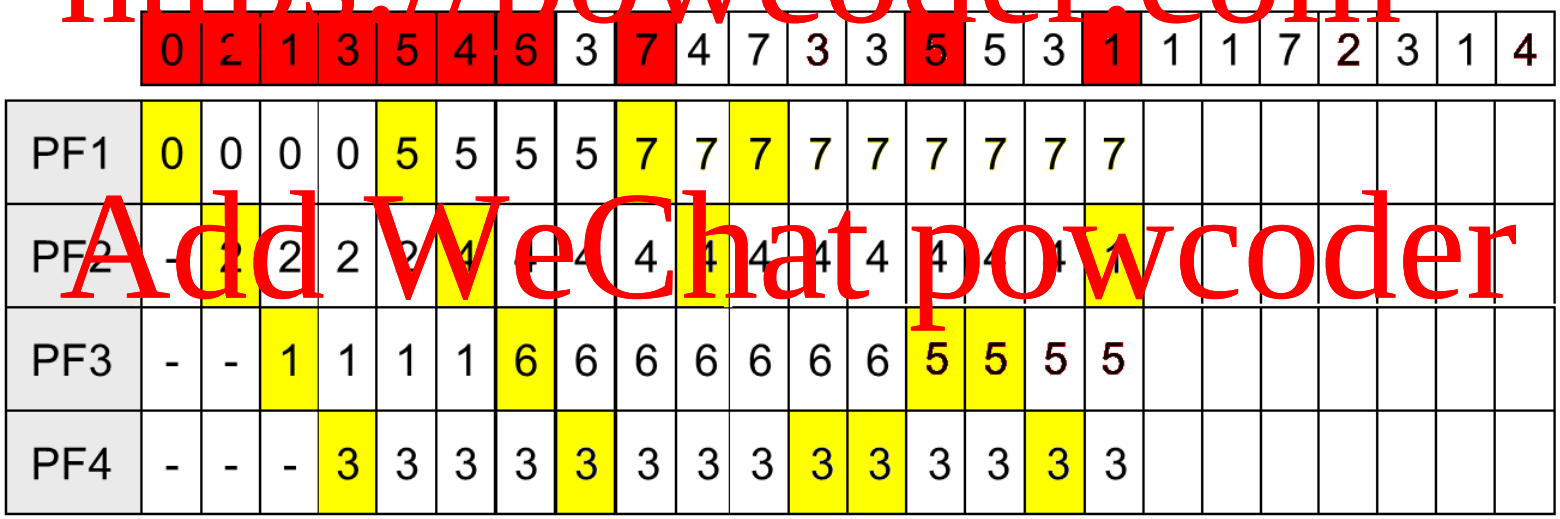


Figure: Least Recently Used

Page Replacement

Least-Recently-Used

- Assume we have a system with eight logical address pages & four physical page frames
- Consider the following **page references** in order:

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

- The number of **page faults** that are generated is **12**

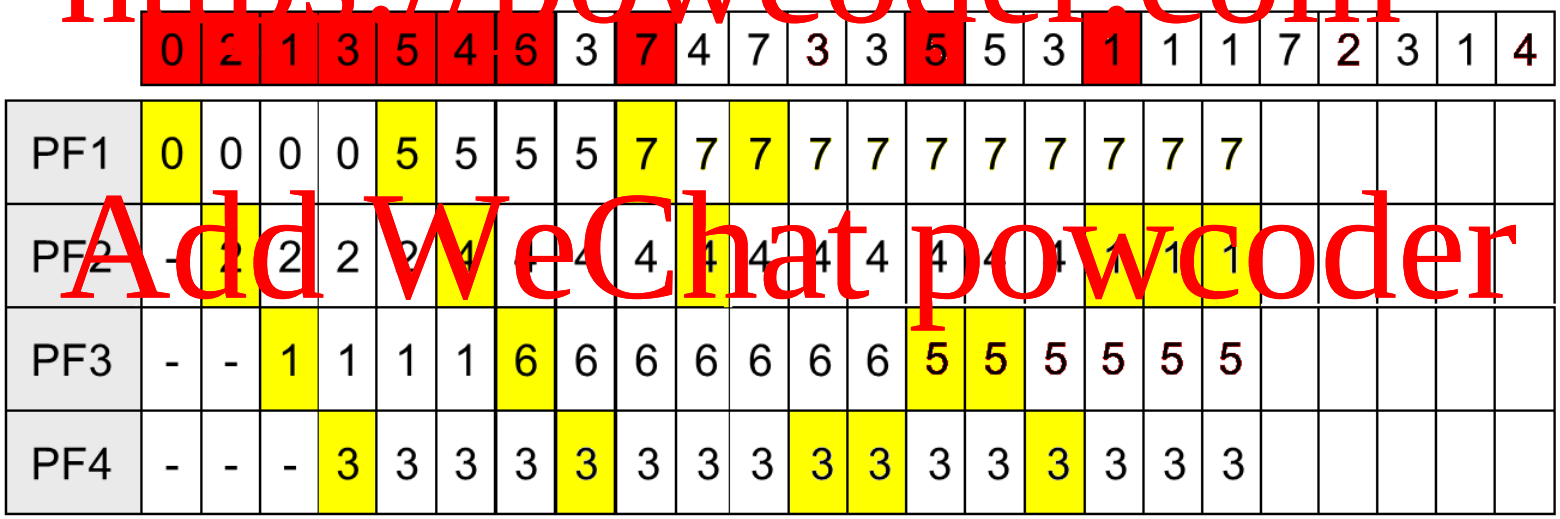


Figure: Least Recently Used

Page Replacement

Least-Recently-Used

- Assume we have a system with eight logical address pages & four physical page frames
- Consider the following **page references** in order:

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

- The number of **page faults** that are generated is **12**

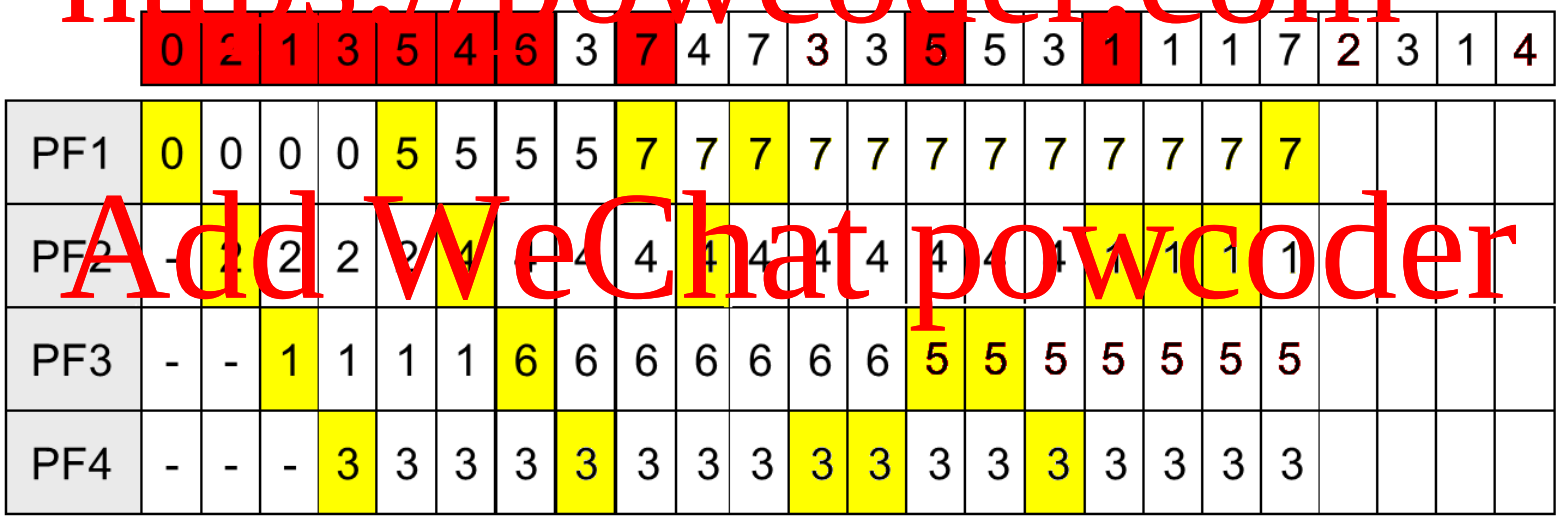


Figure: Least Recently Used

Page Replacement

Least-Recently-Used

- Assume we have a system with eight logical address pages & four physical page frames
- Consider the following **page references** in order:

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

- The number of **page faults** that are generated is **12**

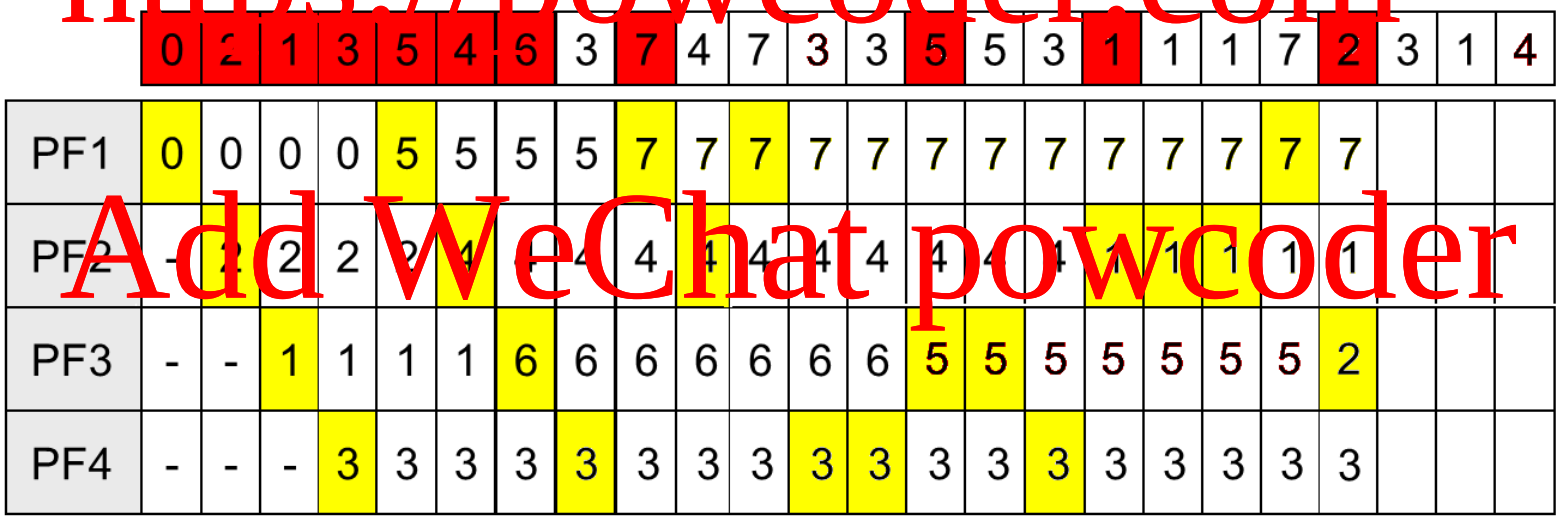


Figure: Least Recently Used

Page Replacement

Least-Recently-Used

- Assume we have a system with eight logical address pages & four physical page frames
- Consider the following **page references** in order:

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

- The number of **page faults** that are generated is **12**

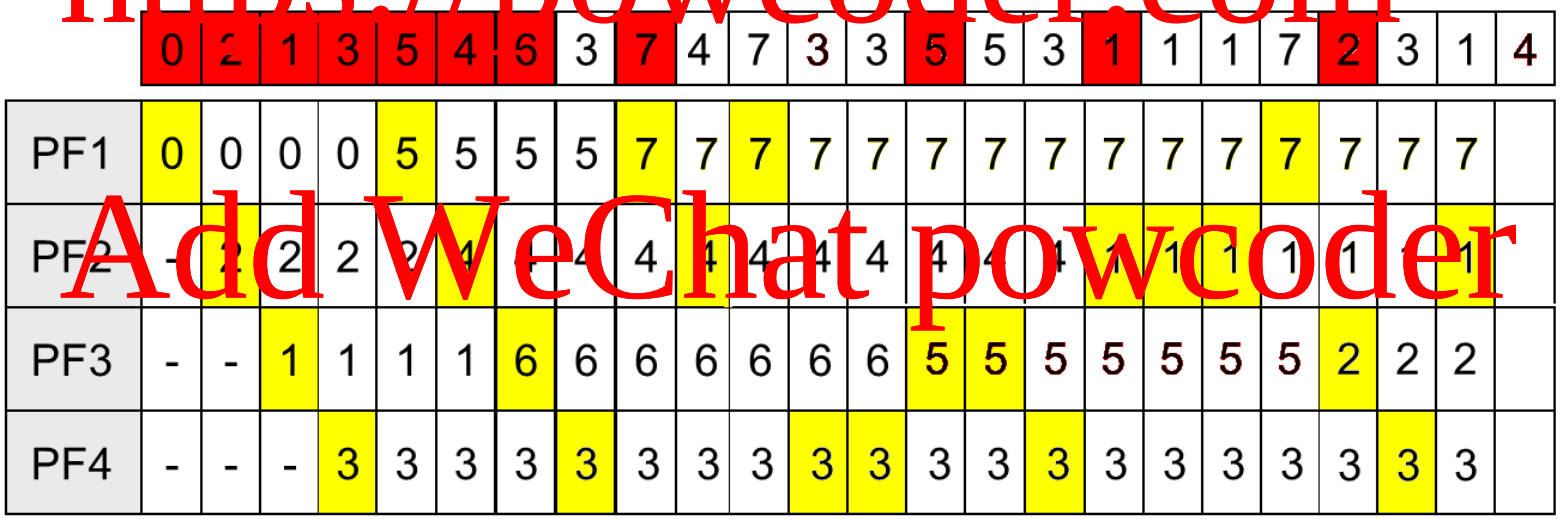


Figure: Least Recently Used

Page Replacement

Least-Recently-Used

- Assume we have a system with eight logical address pages & four physical page frames
- Consider the following **page references** in order:

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

- The number of **page faults** that are generated is **12**

	0	2	1	3	5	4	6	3	7	4	7	3	3	5	5	3	1	1	1	7	2	3	1	4
PF1	0	0	0	0	5	5	5	5	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	4
PF2	-	2	2	2	2	4	4	4	4	4	4	4	4	4	4	4	1	1	1	1	1	1	1	1
PF3	-	-	1	1	1	1	6	6	6	6	6	6	6	5	5	5	5	5	5	5	2	2	2	2
PF4	-	-	-	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3

Figure: Least Recently Used

Page Replacement Algorithms

Summary

1. **Optimal** page replacement: Optimal but not realisable.
2. **FIFO** page replacement: Poor performance, but easy to implement.
   - Second chance replacement: Better than FIFO, not a great implementation.
   - Clock replacement: Easy maintenance of the list, but can still be slow.
3. **Not recently used** (NRU): Easy to understand, moderately efficient (Kind of an approx. of LRU).
4. **Least recently used** (LRU): Good approx. to optimal. More difficult to implement (hardware may help).

Note that there are other alternatives such as aging or WSClock (basically variations of LRU).

Resident Set

Size of the Resident Set

- How many pages should be allocated to individual processes:
  - **Small resident sets** enable to store **more processes** in memory ⇒ improved CPU utilisation
  - **Small resident sets** may result in **more page faults**
  - **Large resident sets** may **no longer reduce** the **page fault rate** (**diminishing returns**)
- A trade-off exists between the **sizes of the resident sets** and **system utilisation**

Resident Set

Size of the Resident Set

- Resident set sizes may be **fixed** or **variable** (i.e. adjusted at runtime)
- For **variable sized** resident sets, **replacement policies** can be:
  - **Local**: a page of the same process is replaced
  - **Global**: a page can be taken away from a different process
- Variable sized sets require **careful evaluation of their size** when a **local scope** is used (often based on the **working set** or the **page fault frequency**)

Resident Set

Size of the Resident Set: Local vs. Global approaches

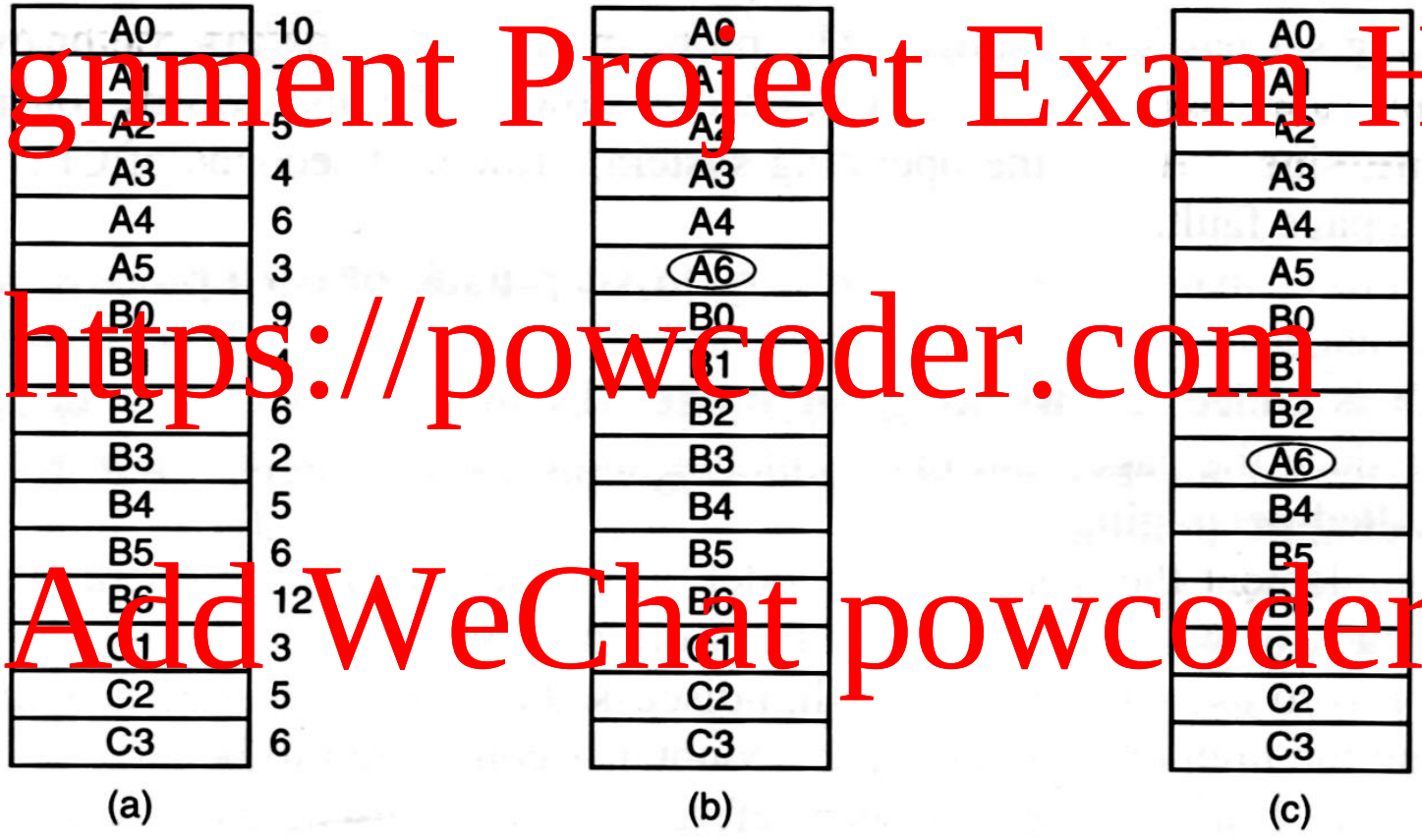


Figure: Local vs. global page replacement. (a) Original config, number at the right represents loading time (b) Local (c) Global (Tanenbaum)

Working Sets

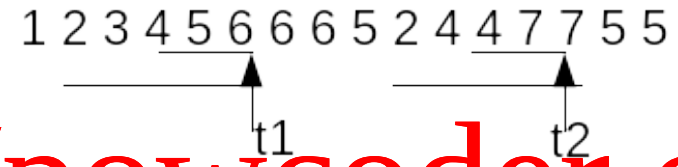
Defining and Monitoring Working Sets

- The **resident set** comprises the set of pages of the process that are in memory
- The **working set** $W(t, k)$ comprises the set referenced pages in the last k (= working set window) virtual time units for the process
- k can be defined as "**memory references**" or as "**actual process time**"
  - The the set of most recently used pages
  - The set of pages used within a pre-specified time interval
- The **working set size** can be used as a guide for the number frames that should be allocated to a process

Working Sets

Monitoring Working Sets: Example

- Consider the following page **references in order**:



- If $k = 3$:
  - At t_1, $W(t_1, 3) = \{4, 5, 6\}$
  - At t_2, $W(t_1, 3) = \{4, 7\}$

- If $k = 5$:
  - At t_1, $W(t_1, 5) = \{2, 3, 4, 5, 6\}$
  - At t_2, $W(t_1, 5) = \{2, 4, 7\}$

Working Sets

Defining and Monitoring Working Sets

- The working set is a **function of time** t:
  - Processes **move between localities**, hence, the pages that are included in the working set **change over time**
  - **Stable** intervals alternate with intervals of **rapid change**
- $|W(t,k)|$ is then variable in time. Specifically:

$$1 \leq |W(t,k)| \leq min(k,N) \tag{1}$$

where N is the total number of pages of the process.

Working Sets

Monitoring Working Sets

- Choosing the right value for k is paramount:
  - Too **small**: inaccurate, pages are missing
  - Too **large**: too many unused pages present
  - **Infinity**: all pages of the process are in the working set
- Working sets can be used to guide the **size of the resident sets**
  - Monitor the working set
  - Remove pages from the resident set that are not in the working set
- The working set is costly to maintain $\Rightarrow$ **page fault frequency (PFF)** can be used as an approximation
  - If the PFF is increased -> we need to increase k
  - If PFF is very reduced -> we may try to decrease k

Resident Sets

Local vs. Global Replacement

- **Global replacement policies** can select frames from the entire set, i.e. they can be "taken" from other processes
  - Frames are **allocated dynamically** to processes
  - Processes cannot control their own page fault frequency, i.e., the **PFF** of one process is **influenced by other processes**
- **Local replacement policies** can only select frames that are allocated to the current process
  - Every process has a **fixed fraction of memory**
  - The locally **"oldest page"** is not necessarily the globally "oldest page"
- Windows uses a **variable approach** with **local replacement**
- Page replacements algorithms explained before can use both policies.

Paging Daemon

Pre-cleaning ($\Leftrightarrow$ demand-cleaning)

- It is more efficient to **proactively** keep a number of **free pages** for **future page faults**
  - If not, we may have to **find a page** to evict and we **write it to the drive** (if modified) first when a page fault occurs
- Many systems have a background process called a **paging daemon**
  - This process **runs at periodic intervals**
  - It inspect the state of the frames and, if too few frames are free, it **selects pages to evict** (using page replacement algorithms)

Paging Daemon

Pre-cleaning ($\Leftrightarrow$ demand-cleaning)

- It is more efficient to **proactively** keep a number of **free pages** for **future page faults**
  - If not, we may have to **find a page** to evict and we **write it to the drive** (if modified) first when a page fault occurs
- Many systems have a background process called a **paging daemon**
  - This process **runs at periodic intervals**
  - It inspect the state of the frames and, if too few frames are free, it **selects pages to evict** (using page replacement algorithms)
- Paging daemons can be combined with **buffering** (free and modified lists) $\Rightarrow$ write the modified pages **but keep them in main memory** when possible

Thrashing

Defining Thrashing

- Assume **all available pages are in active use** and a new page needs to be loaded:
  - The page that will be **evicted** will have to be **reloaded soon afterwards**, i.e., it is still active
- **Thrashing** occurs when pieces are swapped out and loaded again immediately

Thrashing

A Vicious Circle?

- CPU utilisation is too low $\Rightarrow$ scheduler **increases degree of multi-programming**
  - $\Rightarrow$ Frames are allocated to new processes and **taken away from existing processes**
    - $\Rightarrow$ I/O **requests are queued** up as a consequence of page faults
- CPU **utilisation drops further** $\Rightarrow$ scheduler increases degree of multi-programming

Thrashing
Causes/Solutions

- **Causes** of thrashing include:
  - The degree of multi-programming is too high, i.e., the total **demand** (i.e., the sum of all **working set** sizes) **exceeds supply** (i.e. the available frames)
  - An individual process is allocated **too few pages**
- This can be **prevented** by, e.g., using good **page replacement policies**, reducing the **degree of multi-programming** (medium term scheduler), or adding more memory
- The **page fault frequency** can be used to detect that a system is thrashing

Summary

Take-Home Message [1]

- Second Chance FIFO, Clock Replacement, NRU, LRU page replacement
- Page allocations to processes (variable, fixed, local, global)
- Page Daemons
- Thrashing

[1] Tanenbaum Section 3.4, 3.5.1, 3.5.8