

Virtual Memory

Mohammad S. Hasan

Staffordshire University, UK

Lecture Outline

- ◆ Reason for virtual memory & what it is.
- ◆ Demand Paging
- ◆ Performance of Demand Paging
- ◆ Page Replacement
- ◆ Page-Replacement Algorithms
- ◆ Allocation of Frames
- ◆ Thrashing

The need for Virtual Memory

- ◆ **Memory is a limited resource e.g. 2GB, 4GB etc.**
- ◆ **Problem situation**
 - **Total memory requirement of all processes > total physical memory**
- ◆ **Long Term Scheduler**
 - **Decides whether to allow processes into the system**
 - **It might refuse to run new processes until some of the processes are finished and release memory back to the OS**

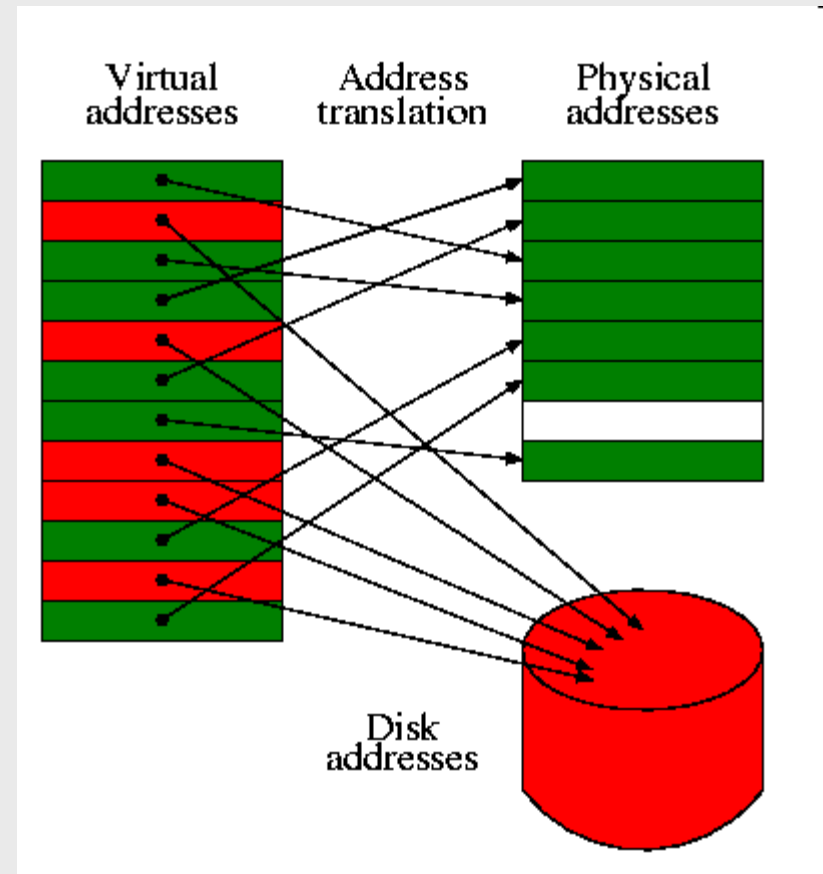
The need for Virtual Memory (Cont.)

- ◆ **Virtual memory is used to get round this problem**
 - it allows a computer with X amount of physical memory to run a number of processes where the total memory requirement is $> X$
- ◆ **Typical situation**
 - Physical memory: 2GB RAM
 - Hard disk: 500 GB
 - Requirement of OS + other processes: 2.5 GB

The need for Virtual Memory (Cont.)

Virtual Memory Technique

- A proportion of a process's memory is held in physical memory
- Some of it is left on the hard disk



The need for Virtual Memory (Cont.)

◆ Allocation Scheme

- The scheme that determines which parts of the process' memory will be in physical memory

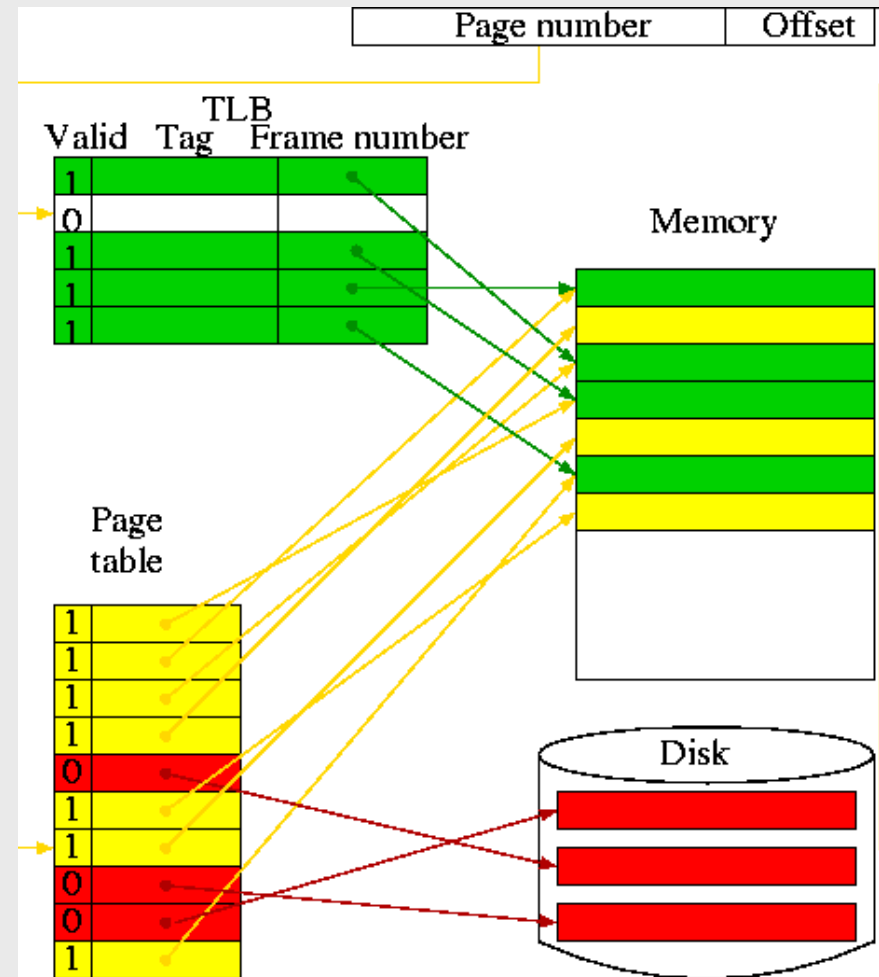
◆ The general idea

- keep the items of data and instructions in physical memory that
 - are currently being used or
 - are likely to be used in the near future - implemented using demand paging or demand segmentation

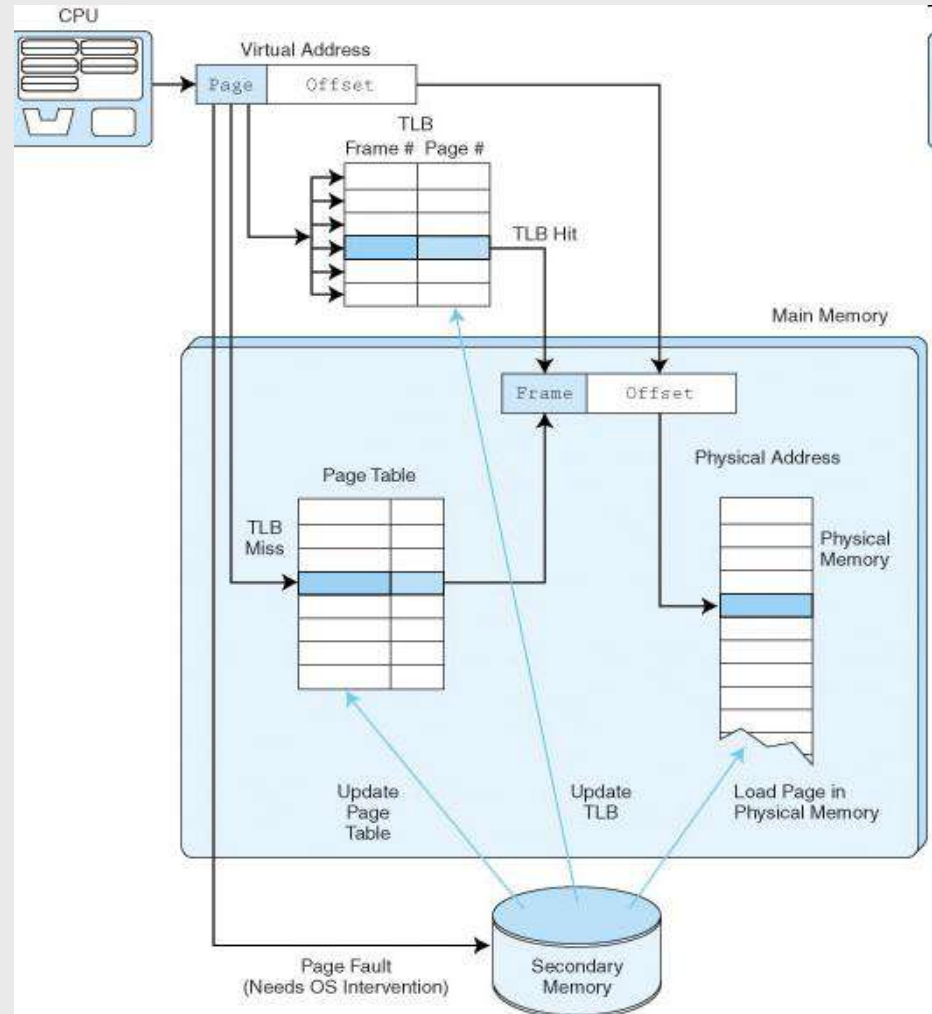
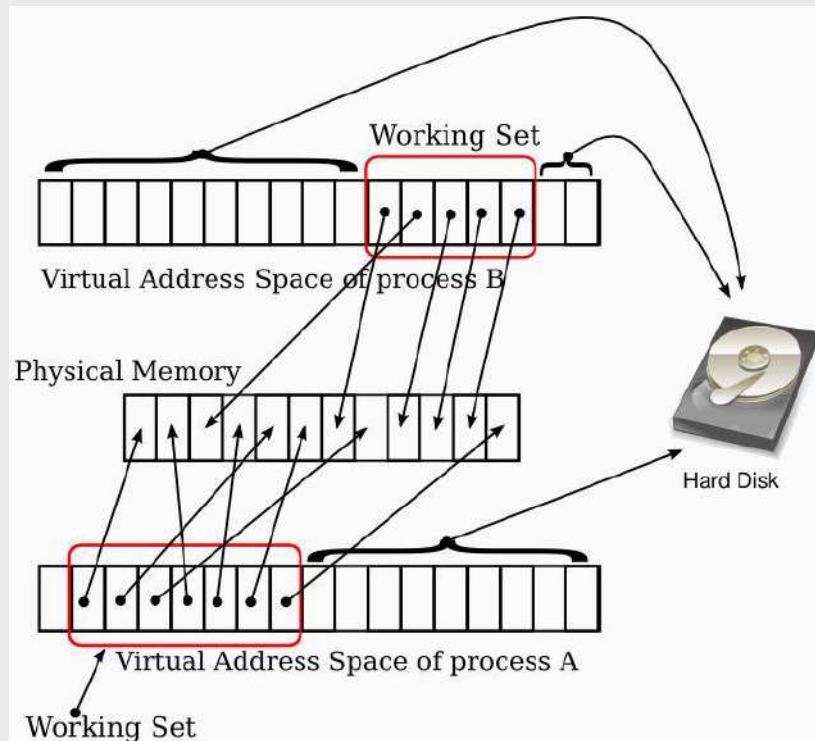
The need for Virtual Memory (Cont.)

Replacement policy

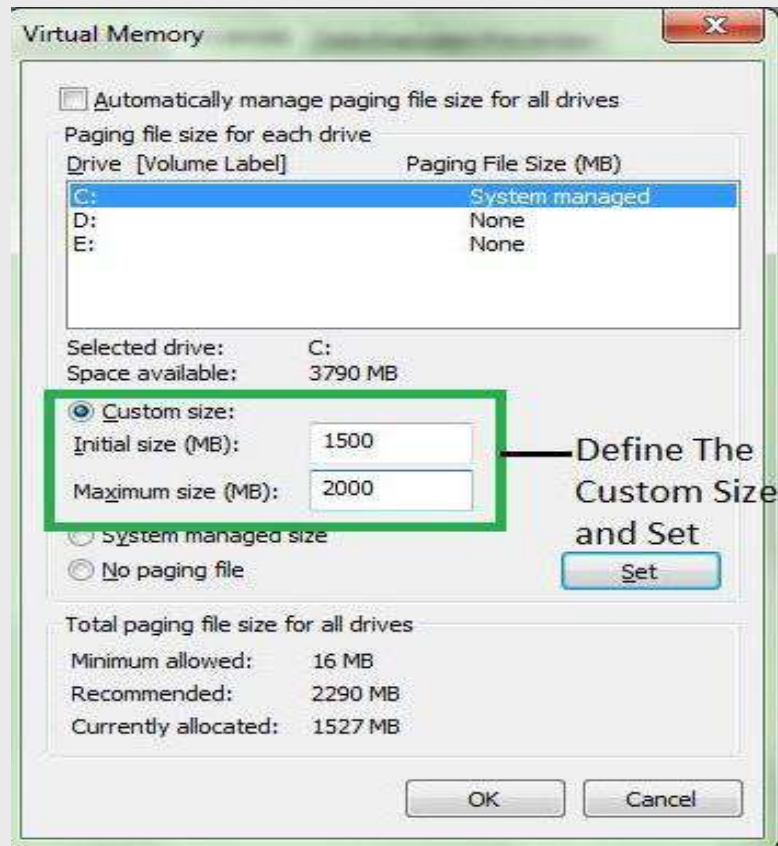
- when the allocated physical memory of a process becomes full and new pages need to be loaded, this scheme of OS determines which memory needs to be replaced with the new memory



The need for Virtual Memory (Cont.)

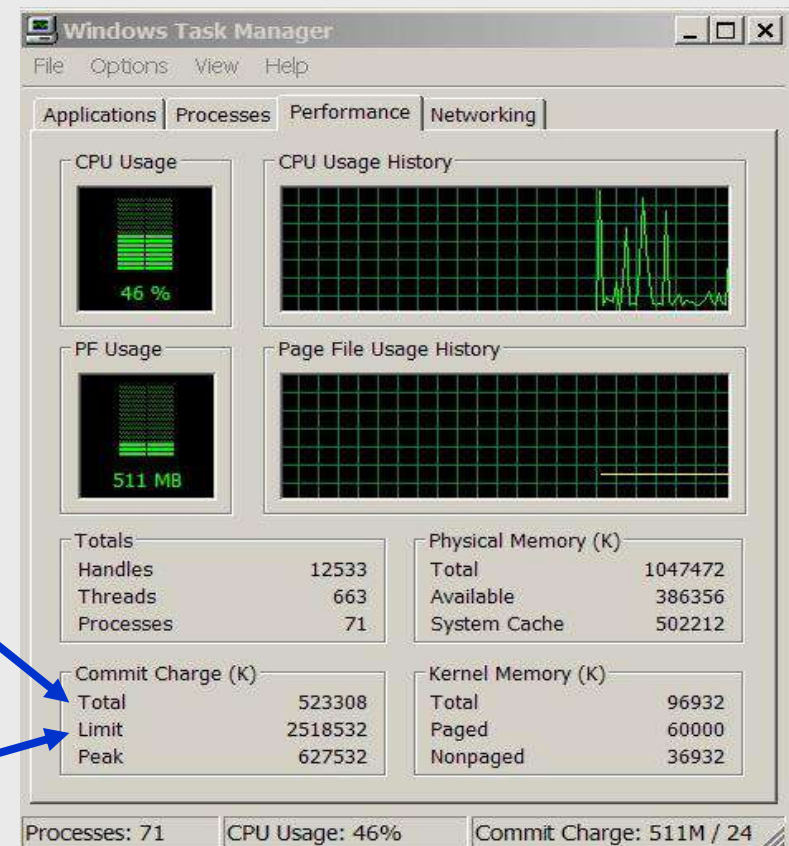


Virtual Memory – Windows



Total VM that has committed storage

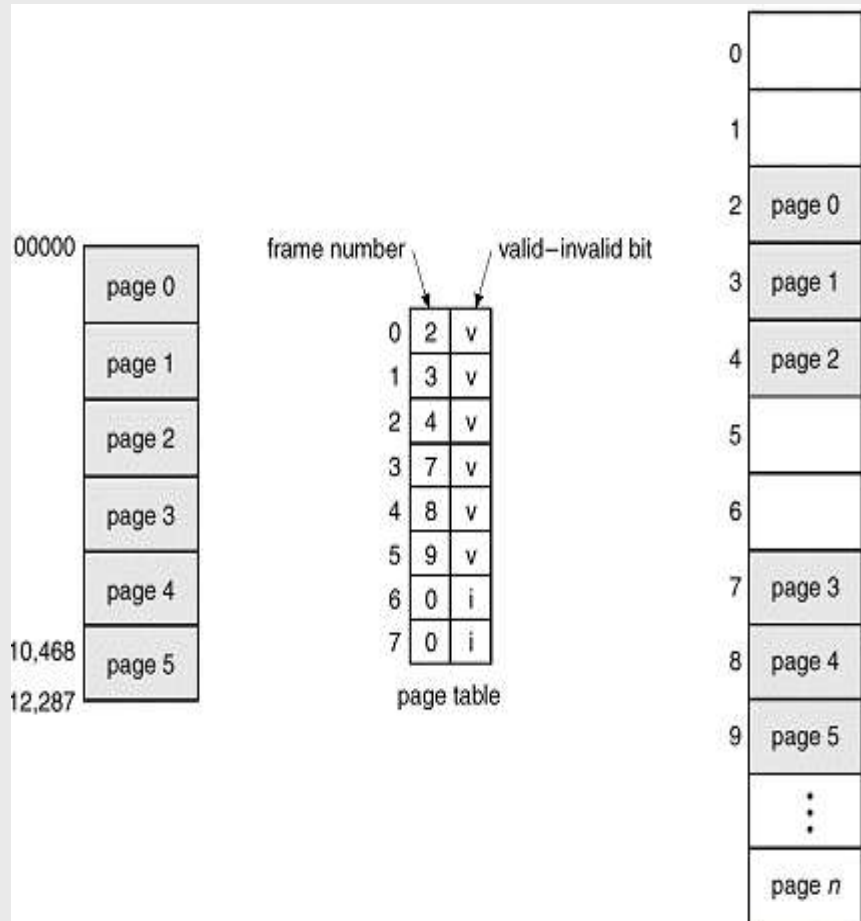
Max VM without expanding page file



Demand Paging

- ◆ Bring a page into memory only when it is needed.
- ◆ Uses a “lazy approach” to bringing into memory
- ◆ A process makes a reference to a page of its memory while running on the CPU
 - if reference is invalid (e.g. no such program address) then abort process
 - if page not in memory then fetch into memory

Valid-Invalid Bit



◆ A valid-invalid bit is associated with each page table entry

- 1 means the page is in memory
- 0 means the page is not in memory
- Initially set to 0 on all entries.

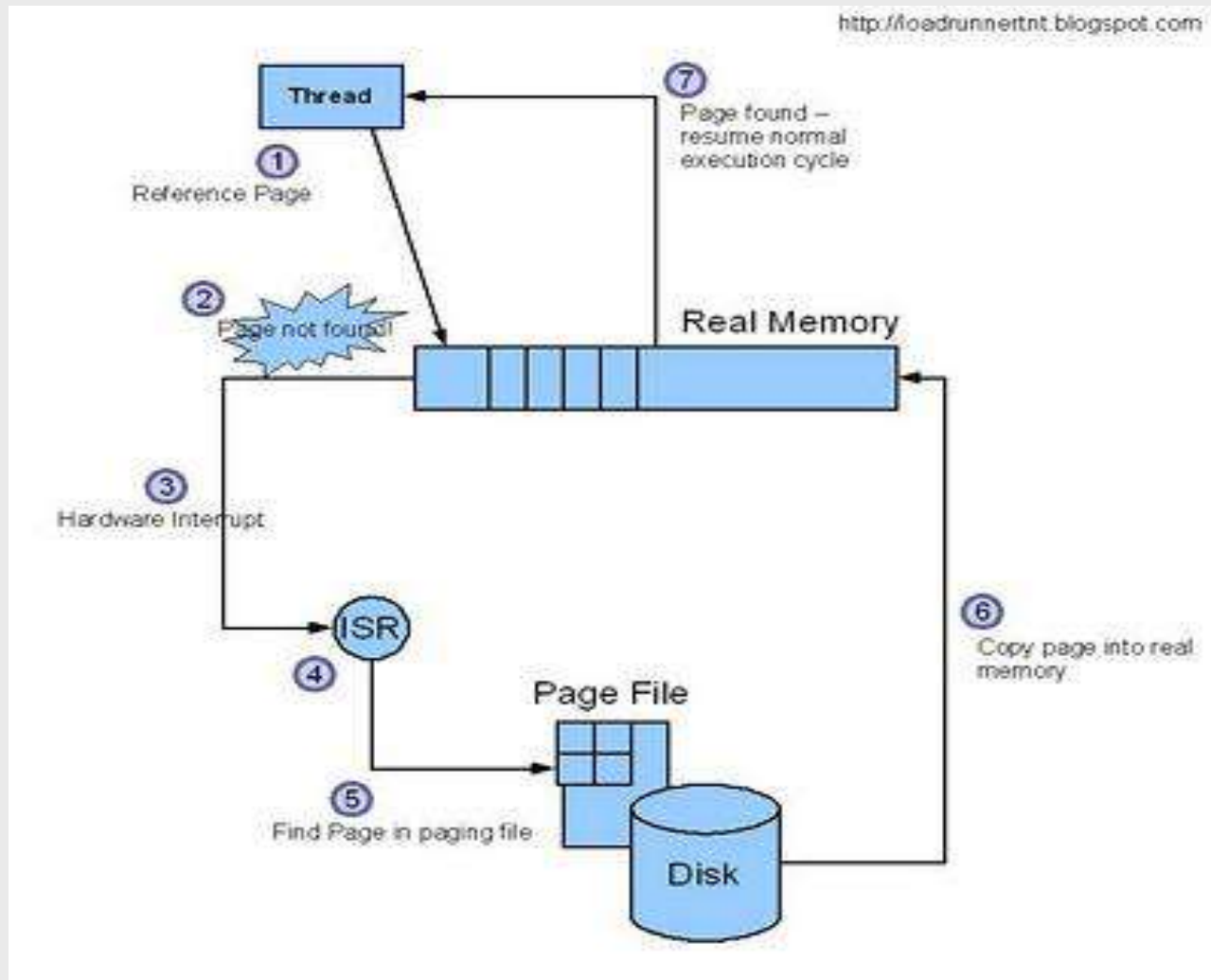
◆ if valid–invalid bit in page table entry is 0 then a page fault is generated

Page Fault

◆ Page fault

- An interrupt that signals the OS that a page that has been referenced by a process is not in memory
- ◆ OS looks at table in **Process Control Block** to decide whether:
 - the reference is invalid (address not in processes address space) then abort OR
 - Address not in memory in which case it needs to be brought into memory

Page Fault (Cont.)



Page Fault (Cont.)

- ◆ The very first reference to a page always generates a page fault
- ◆ To bring in new page requires:
 - Finding empty frame or victim for replacement
 - Swapping page into frame.
 - Updating page table and PCB table
 - Setting valid bit = 1.
- ◆ Restart instruction that made memory reference, but now page with address in it, is in memory

What happens if there is no free frame?

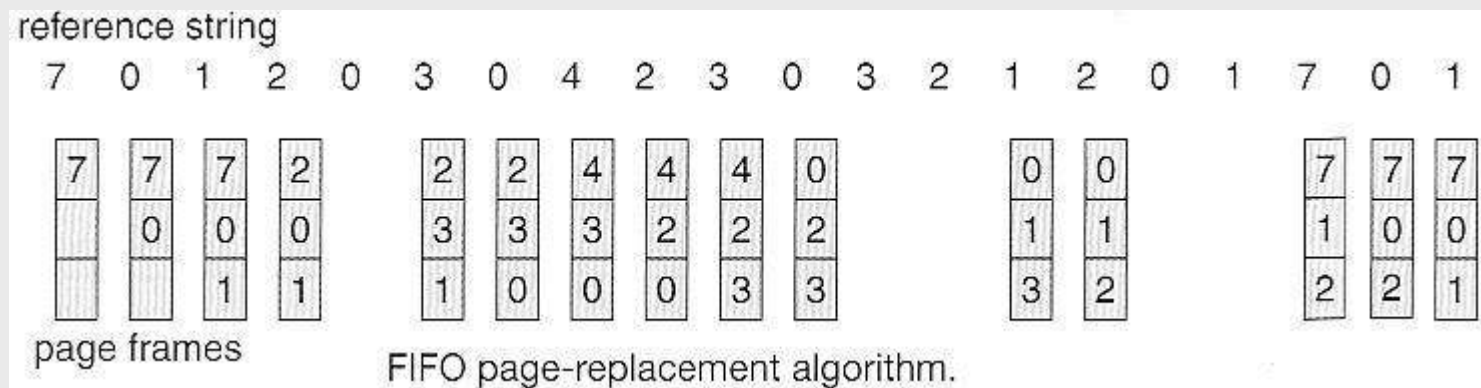
Page replacement

- find some page in memory that is not being used much and replace it with new page.

Performance

- Algorithm which results in minimum number of page faults.

- Same page may be brought in/out of memory several times during lifetime of process



Performance of Demand Paging

◆ Page fault / restart overheads include

- switching process contexts,
- handling interrupt,
- finding free frame,
- initiating disk transfer,
- updating tables, etc.

Performance of Demand Paging (Cont.)

◆ Page Fault Rate $0 \leq p \leq 1.0$

- if $p = 0$ no page faults
- if $p = 1$, every reference generates a fault

◆ Effective Access Time (EAT) =

- $(1 - p) * \text{physical memory access time} + p * (\text{page fault overhead} + \text{time to copy page out (if modified)} + \text{time to copy page in} + \text{restart overhead} + \text{physical memory access time})$

Demand Paging Example

◆ For example

- Physical memory access time = 100 ns
- 50% of the time the required page is found in the main memory, i.e. $p = 0.5$
- Swap Page Time
 - Time to copy an old page out to HDD + time to copy the desired page into main memory
 - 10 ms = 10,000,000 ns
- We will ignore page fault and restart overheads.

Demand Paging Example (Cont.)

◆ EAT

$$= (1 - p) * 100 + p * (0 + \text{time to copy page out} + \text{time to copy page in} + 0 + 100)$$

$$= (1-0.5) * 100 + 0.5 * (\text{100000000} + 100)$$

$$= 0.5 * 100 + 0.5 * (10000100)$$

$$= 5000100 \text{ ns}$$

◆ The page fault rate has huge effect on EAT and overall Virtual Memory

Page-Replacement Algorithms

◆ Criteria

- Want lowest page-fault rate.

◆ Reference string

- A particular string or sequence of memory references

- ◆ The performance of an algorithm can be judged by running it on a reference string and then examining the number of page faults it generates

- ◆ In all examples that follow, the reference string is
1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5.

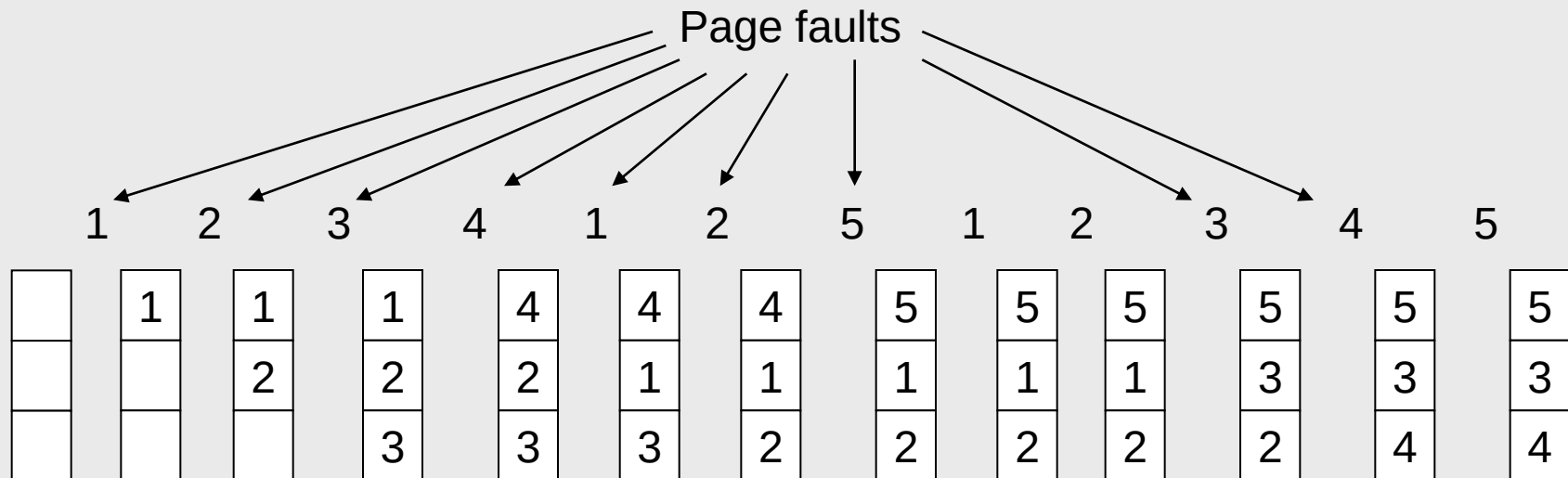
First-In-First-Out (FIFO) Algorithm

Reference string

1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

3 frames

3 pages can be in memory at a time per process

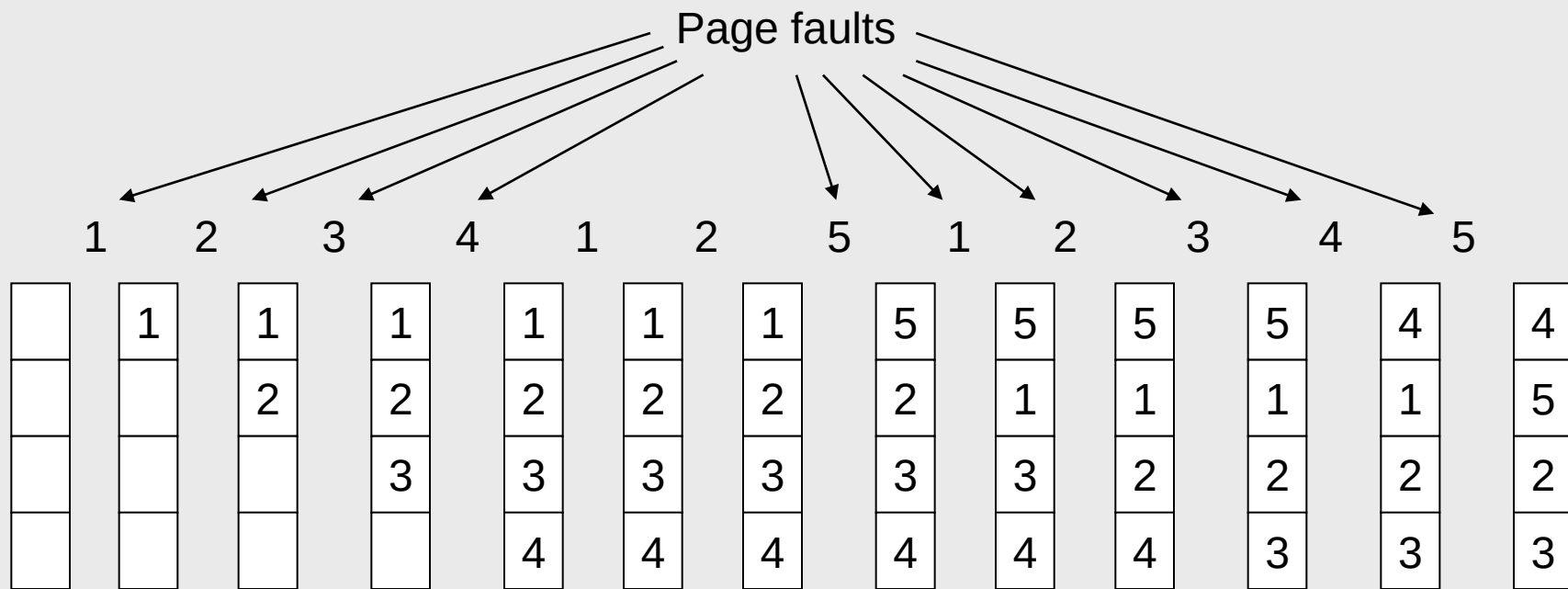


First-In-First-Out (FIFO) Algorithm (Cont.)

◆ **4 frames**

● **Sometimes more frames can produce more page faults**

■ **Belady's Anomaly**

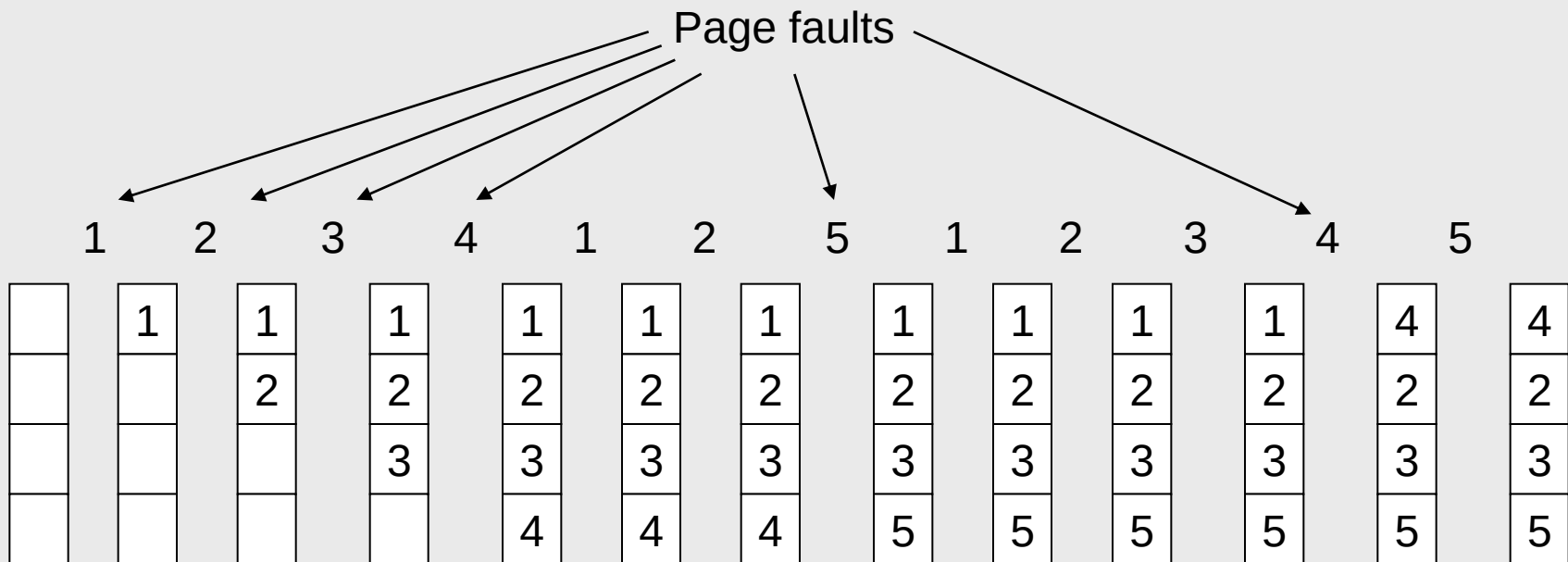


An Optimal Algorithm

➤ Replace page that will not be used for longest period of time in the future.

➤ Reference string

● 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5 with 4 frame



An Optimal Algorithm (Cont.)

◆ Problem

- Which page will be the one that is not referenced for the longest time in the future?

- Do NOT know the future

◆ It is theoretical algorithm and used as benchmark in comparison of algorithms.

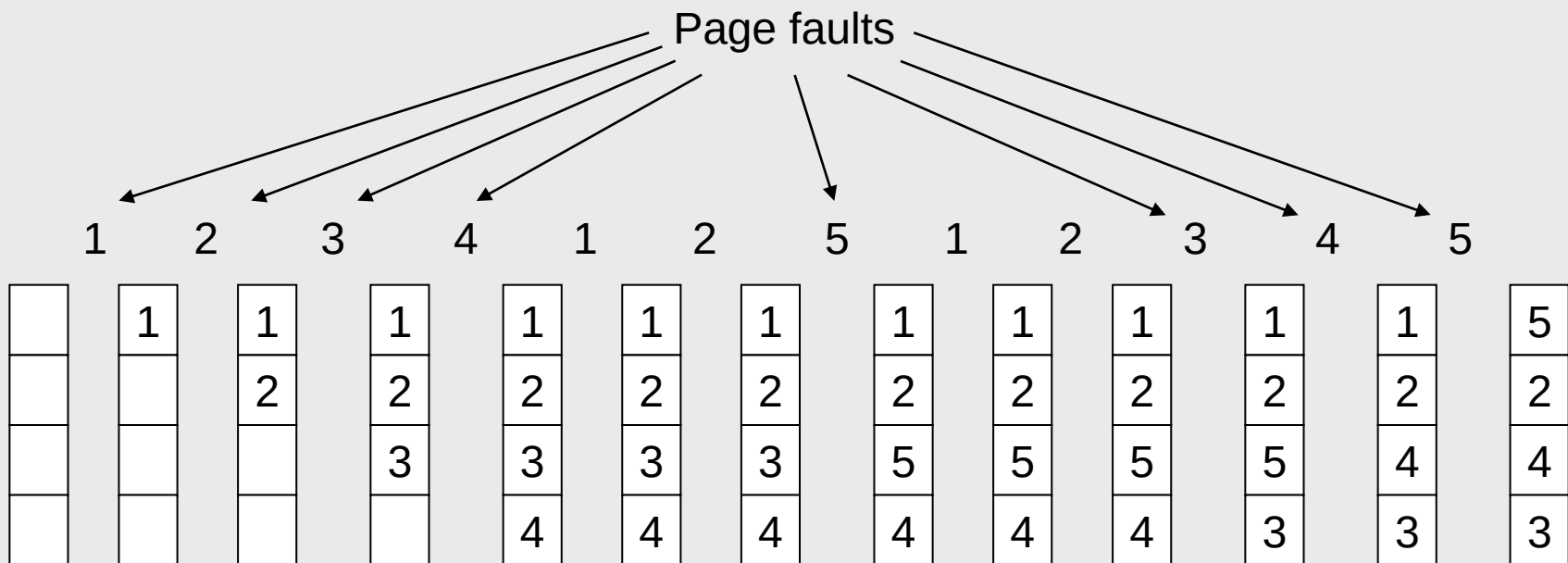
◆ Practical solution - Least Recently Used (LRU)

- Taking recent past as guide to future.
- replacing page that has not been used for longest period of time

Least Recently Used (LRU) Algorithm

Reference string

1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5



LRU Algorithm (Cont.)

- ◆ **Implementation of LRU using counters**
 - **Every page entry has a counter;**
 - **Every time page is referenced, copy the clock into the counter.**
 - **When a page needs to be changed, look at the counters to determine which is oldest.**

LRU Algorithm (Cont.)

- ◆ **Implementation of LRU using a stack**
 - **Keep a stack of page numbers in a double linked list**
 - **When a page is referenced, move it to the top of the stack**
 - **Bottom of stack always LRU page**
 - **Updating stack requires changing set of pointer values**

Allocation of Frames

- ◆ Each process needs minimum number of pages to operate effectively.
- ◆ Two major allocation schemes.
 - **Equal allocation**
 - All processes get the equal share
 - If 100 frames and 5 processes, give each 20 pages
 - **proportional allocation**
 - Allocate according to the size of process.
 - If 100 frames and $P1=400\text{KB}$, $P2=200\text{KB}$, $P3=200\text{KB}$ then $P1=50$ frames, $P2=25$ frames and $P3=25$ frames

Global vs. Local Replacement

◆ Global replacement

- selects a page for replacement in a frame from the set of all frames;
- can use frames for replacement that currently belong to a different process from the one generating the page fault.

◆ Local replacement

- only selects a page for replacement from a set of frames that have been allocated to the process generating the page fault.

Locality Model

Locality model

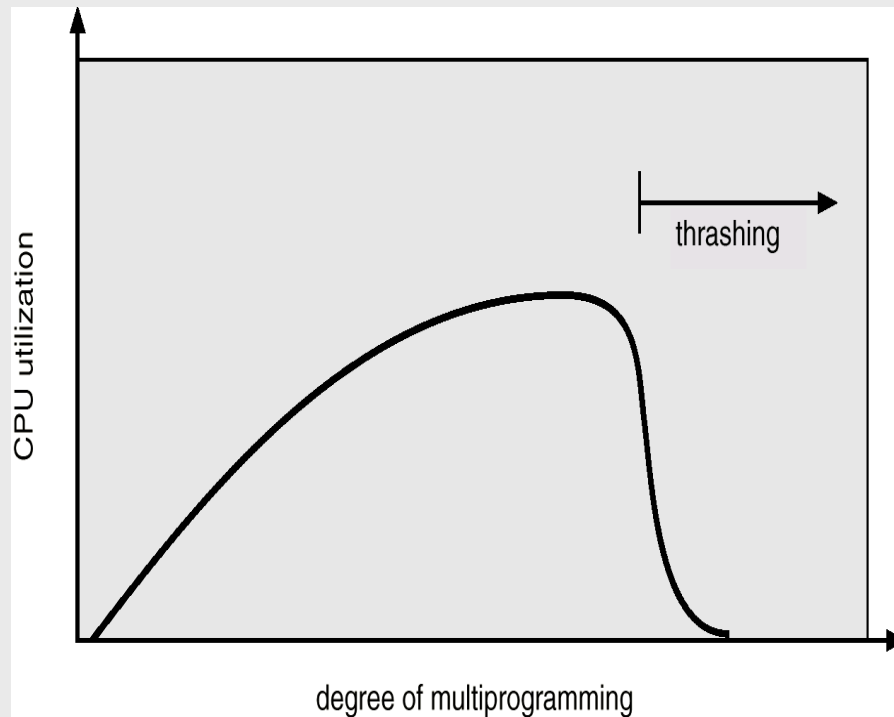
- Set of addresses/pages a process accesses over a period of time during execution tend to form a stable set - known as a locality
- After a period of time the process will start accessing a new set of addresses/pages which in turn will form a new stable set.
- Localities may overlap i.e. some pages used throughout lifetime of program



Thrashing

- ◆ If a process does not have “enough” pages, the page-fault rate is very high.
 - This can lead to low CPU utilisation.
- ◆ Thrashing
 - A process is busy swapping pages in and out from disk
 - Very time consuming.
 - The actual execution of the process is low

Thrashing and Locality Model



◆ **Why does thrashing occur?**

- **Sum of sizes of current localities of all processes on system > total hardware memory size**

Demand Segmentation

- ◆ Used when insufficient hardware to implement demand paging.
- ◆ OS/2 allocates memory in segments
 - it keeps track through segment descriptors
- ◆ Segment descriptor contains a valid bit to indicate whether the segment is currently in memory.
 - If segment is in main memory, access continues,
 - If not in memory, segment fault.



References

◆ **Operating System Concepts. - Chapter 9.**