Memory Management System

Virtual Memory

A technique which allows the execution of processes that may not be completely in memory

Memory Management System

Fact:

In many cases, for the execution of a program, one does not need to have the entire program in the main memory

Examples:

- *Codes for those options in an interface not chosen by user.*
- *Part of the process that handles unusual error conditions*
- *A good portion of an unused large declared arrays*

Memory Management System

Virtual Memory Adoption

- *Overlay*
- *Dynamic Loading*

Memory Management System

Overlay Concept

1. *Keep those pages of logical space that are essential to execution of a process all the time in memory and*
2. *Load other pages of the logical space when they are needed and load a needed page into space that was previously occupied by a page that is no longer needed.*

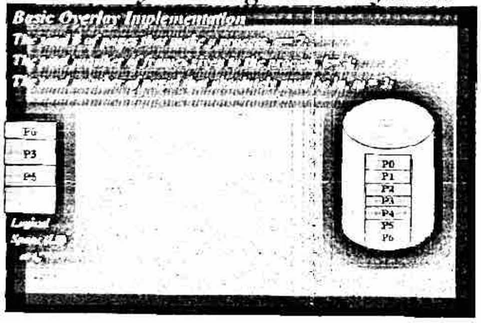
Memory Management System

Basic Overlay Implementation

The total # of pages that make a process [illegible]

The total number of frames given to the process [illegible]

[illegible]



Memory Management System

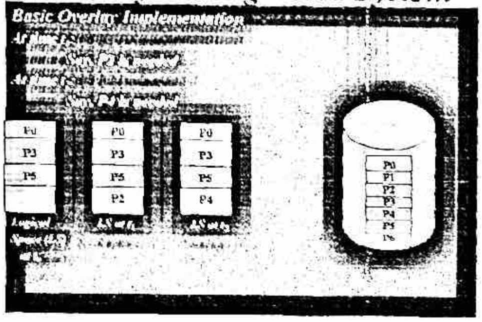
Basic Overlay Implementation

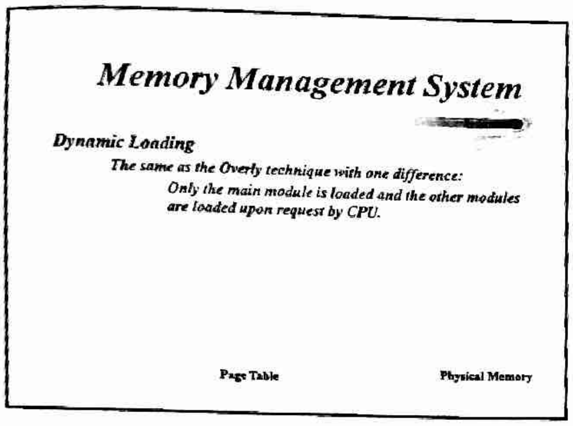
At time t [illegible]

[illegible]

At time t [illegible]

[illegible]





Memory Management System

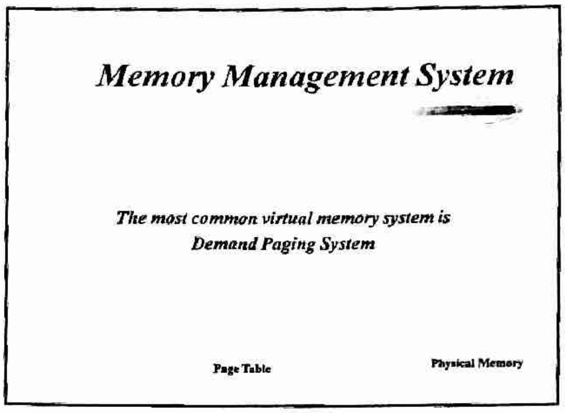
Dynamic Loading

The same as the Overly technique with one difference:

Only the main module is loaded and the other modules are loaded upon request by CPU.

Page Table

Physical Memory

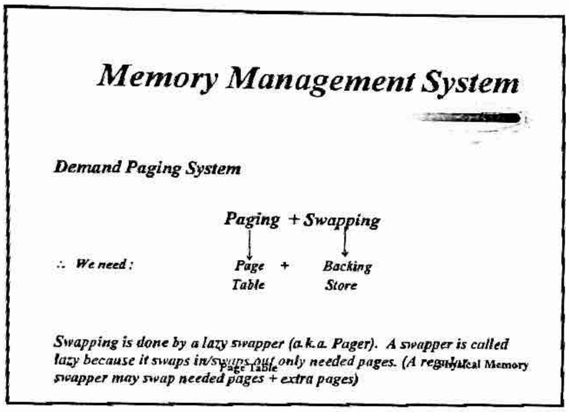


Memory Management System

The most common virtual memory system is Demand Paging System

Page Table

Physical Memory



Memory Management System

Demand Paging System

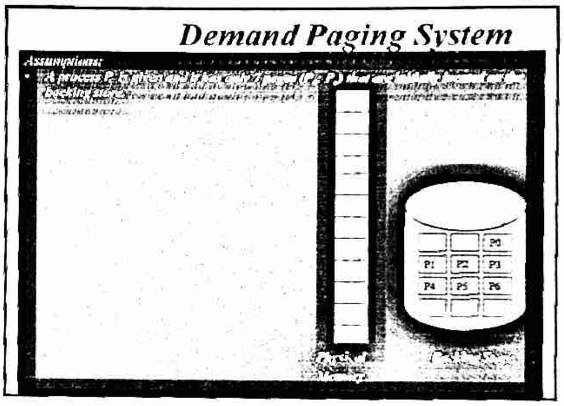
Paging + Swapping

∴ We need : Page Table + Backing Store

Swapping is done by a lazy swapper (a.k.a. Pager). A swapper is called lazy because it swaps in/swaps out only needed pages. (A regular swapper may swap needed pages + extra pages)

Page Table

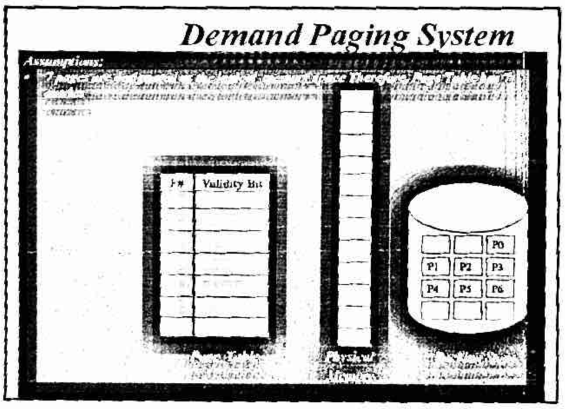
Physical Memory



Demand Paging System

Assumptions:

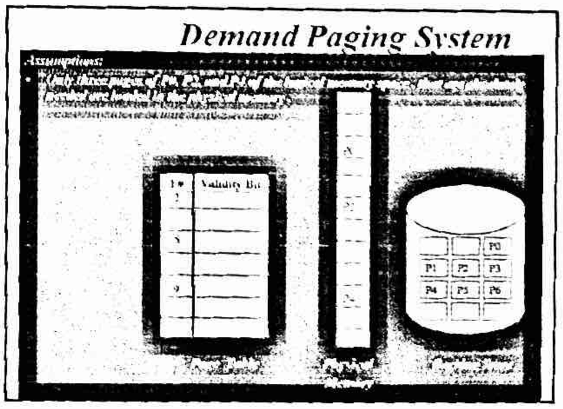
- *A process P is given and it has only 7 pages (P0-P) that are [illegible] backing store.*



Demand Paging System

Assumptions:

- *7 pages [illegible] Therefore Page Table [illegible]*



Demand Paging System

Assumptions:

- *Only three pages of P0, P2, and P4 of the logical memory space that are [illegible] needed are loaded in the physical memory.*

Memory Management System

	[illegible]	[illegible]	[illegible]
0	F2	00	1
1	F9	01	1
2	F8	00	1
3	F12	11	1
4			0
5			0
6			0
7			0

Validity bit says: whether the entry is empty (validity = 0) or not

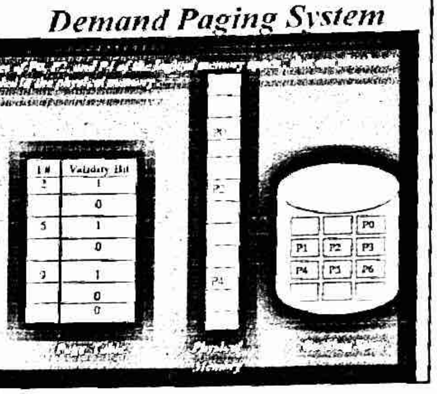
We add an extra meaning to the validity bit for use in DEMEND PAGING
Validity bit says.

1. whether the entry is valid or not and
2. the associated page is legal (i.e. page# verification that it belongs to the logical space is already completed.)

Demand Paging System

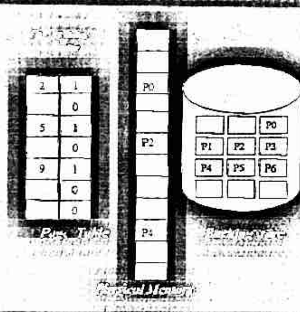
Assumptions:

- *Only three pages of P0, P2, and P4 of the logical memory [illegible] needed are loaded in the physical memory.*



I#	Validity Bit
2	1
	0
5	1
	0
9	1
	0
	0

Demand Paging System



2	1
	0
5	1
	0
9	1
	0
	0

Demand Paging System

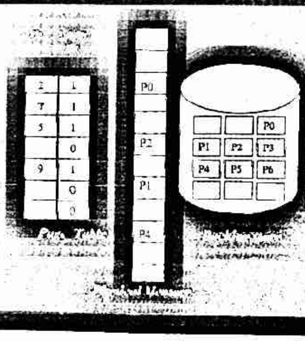
Error is trapped

1. The status of the process that caused the page fault is saved.
2. Since valid bit =0 an Internal Table. Kept with the PCB–Process Control Block, determines whether such page is legal or not.
3. If illegal

   Then. *terminate the process*

   Else:
   - *OS gets the demanded page from the backing store*
   - *A free frame is found by OS*
   - *Page is loaded in the physical memory*
   - *Page table and Internal Table is updated*
   - *Process is re-started and continues.*

Demand Paging System



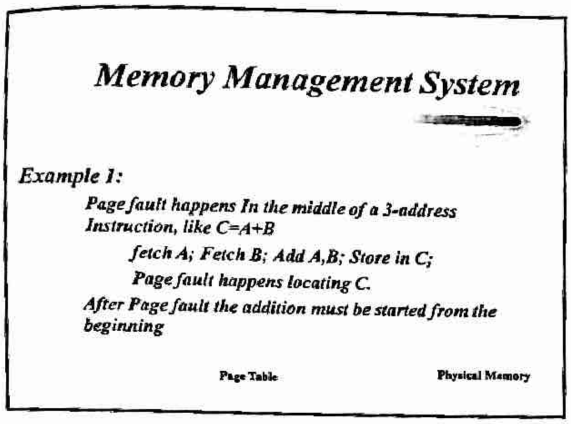
2	1
7	1
5	1
	0
9	1
	0
	0

Memory Management System

Problem:

(Sometimes) it is difficult to start execution of a process exactly from the point that caused the page fault.

Page Table Physical Memory



Memory Management System

Example 1:

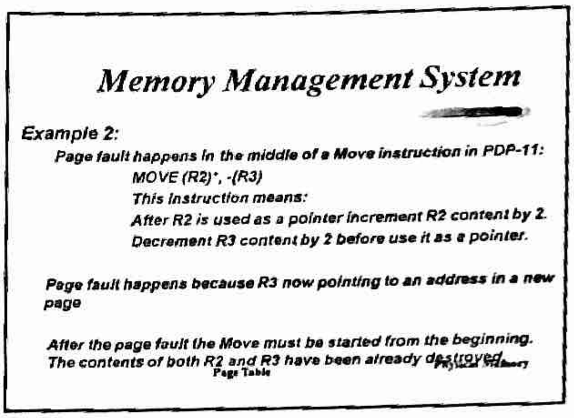
Page fault happens In the middle of a 3-address Instruction, like C=A+B

fetch A; Fetch B; Add A,B; Store in C;

Page fault happens locating C.

After Page fault the addition must be started from the beginning

Page Table Physical Memory



Memory Management System

Example 2:

Page fault happens in the middle of a Move instruction in PDP-11:

MOVE (R2)⁺, -(R3)

This instruction means:

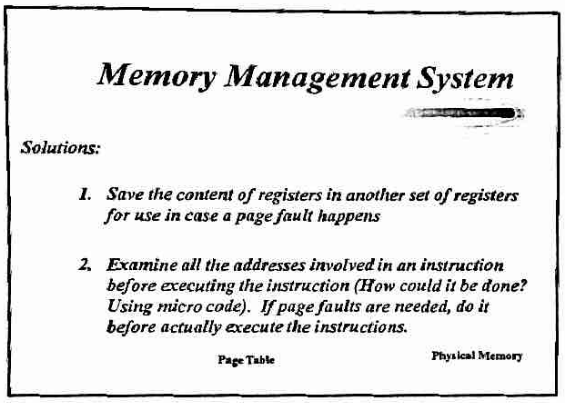
After R2 is used as a pointer increment R2 content by 2.

Decrement R3 content by 2 before use it as a pointer.

Page fault happens because R3 now pointing to an address in a new page

After the page fault the Move must be started from the beginning. The contents of both R2 and R3 have been already destroyed.

Page Table Physical Memory

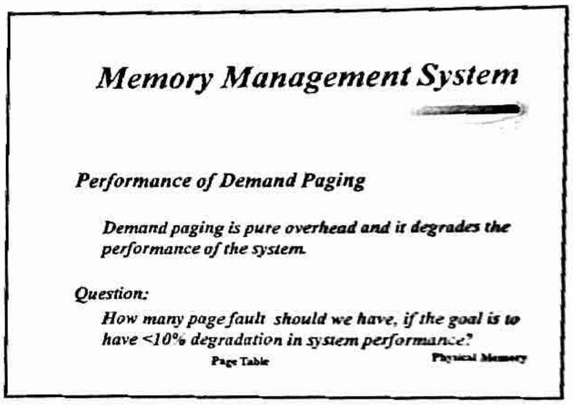


Memory Management System

Solutions:

1. *Save the content of registers in another set of registers for use in case a page fault happens*
2. *Examine all the addresses involved in an instruction before executing the instruction (How could it be done? Using micro code). If page faults are needed, do it before actually execute the instructions.*

Page Table Physical Memory



Memory Management System

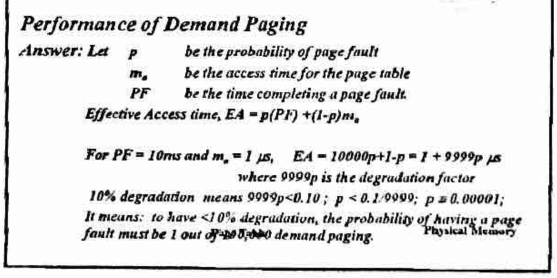
Performance of Demand Paging

Demand paging is pure overhead and it degrades the performance of the system.

Question:

How many page fault should we have, if the goal is to have <10% degradation in system performance?

Page Table Physical Memory



Memory Management System

Performance of Demand Paging

Answer: Let p *be the probability of page fault*

m_a *be the access time for the page table*

PF *be the time completing a page fault.*

Effective Access time, $EA = p(PF) + (1-p)m_a$

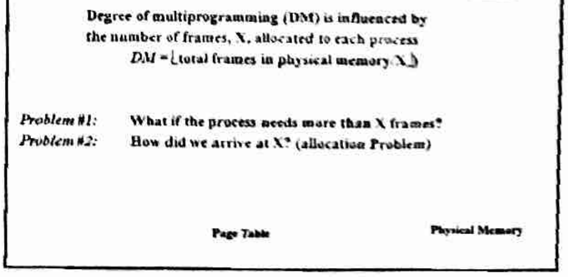
For $PF = 10ms$ *and* $m_a = 1\ \mu s$, $EA = 10000p+1-p = 1 + 9999p\ \mu s$

where 9999p is the degradation factor

10% degradation means $9999p<0.10$; $p < 0.1/9999$; $p \cong 0.00001$;

It means: to have <10% degradation, the probability of having a page fault must be 1 out of 100,000 demand paging.

Page Table Physical Memory



Memory Management System

Degree of multiprogramming (DM) is influenced by the number of frames, X, allocated to each process

$DM = \lfloor \text{total frames in physical memory}/X \rfloor$

Problem #1: What if the process needs more than X frames?

Problem #2: How did we arrive at X? (allocation Problem)

Page Table Physical Memory

Memory Management System

Answer to Problem #1: Use of Page Replacement Technique

Page Replacement Technique:

1- Find a frame (victim frame)
2- Write the content of the frame to the backing store
3- Update the page table and internal table
4- The freed frame now can be used as part of the page fault handling.

Note: Use of a *dirty bit* eliminates the second step.

Page Table **Physical Memory**

Memory Management System

Page Replacement Algorithms

Given:

Reference String: A string of memory references (Created artificially or by tracing a system)

Number of Frames given to the process (Higher the number of frames, lower the number of page faults)

Page Table **Physical Memory**

Memory Management System

Let say a process is made up of 50 pages. However, only 4 frames are dedicated to the process.

The page table has 50 entries. At any given time a maximum of four entries of the page table are full. These entries are related to the four pages that currently occupy the four frames (in RAM).

During the discussion of the algorithms we see something like the following picture: This picture shows the four frames of RAM that in reality may not be contiguous but we show them in contiguous form for ease of drawing. The picture says that the 4 frames have been occupied by page 7, page 2, page 3 and page 4.

[illegible]

Memory Management System

Page Replacement Algorithms

1. *FIFO (Replace the page that is at the head of queue)*
2. *Optimal Replacement (Replace the page that will not be used for the longest period of time)*
3. *LRU (Least Recently Used)*
4. *LRU Approximation*
5. *Second Chance Replacement*
6. *LFU (Least Frequently Used)*
7. *MFU (Most Frequently Used)*
8. *Page Classes*
9. *Ad Hoc Algorithms*

Page Table **Physical Memory**

Memory Management System

Page Replacement Algorithms

FIFO *(replace the page that is at the head of queue)*

- *Reference String: 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1*

```
7 7 7 0 1 2 3 0 4 2 3 0 1 2 7       15 page
  0 0 1 2 3 0 4 2 3 0 1 2 7 0       Replacement
    1 2 3 0 4 2 3 0 1 2 7 0 1
```

- *Belady's Anomaly: Reference String: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5*
  - *Using 4 frames*

```
1 1 1 1 2 3 4 5 1 2
  2 2 2 3 4 5 1 2 3 10
    3 3 4 5 1 2 3 4 P.R.
      4 5 1 2 3 4 5
```

Using 3 frames

```
1 1 1 2 3 4 1 2 5
  2 2 3 4 1 2 5 3 9
    3 4 1 2 5 3 4 P.R.
```

Memory Management System

Optimal Replacement ***(replace the page that will not be used for the***

Reference String: 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1

```
7 7 7 2 2 2 2 2 7       9 page
  0 0 0 0 4 0 0 0       Replacement
    1 1 3 3 3 1 1
```

LRU (Least Recently Used)

```
7 7 7 2 2 4 4 4 0 0 0 0       12 page
  0 0 0 0 0 0 3 3 3 3 3       Replacement
    1 1 3 3 2 2 2 2 2 2
```

Memory Management System

LRU Implementation

Using time-of-use (Clock)

Using a double-linked list

Memory Management System

LRU Implementation

Using time-of-use (Clock)

- A logical clock (a Counter) is used and set to zero Counter = 0
- Clock will be incremented by one when a page reference (in Reference string) is made.
- The new clock value is recorded for the referenced page.
- Always the page with smallest value of the counter is the least recent used page.

Memory Management System

LRU Implementation

- *Using time-of-use*

Example: Clock = 0 Reference String: 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1

Referencing 7: Clock =1, Referencing 0: Clock =2, Referencing 1: Clock =3,

Referencing 2: Clock =4 and page 7 with smallest clock is replaced.

Referencing 0: Clock =5 and no replacement. Referencing 3: Clock =6 and page 1 is replaced. Referencing 0: Clock =7 and no replacement (clock is updated for P=0.)

Referencing 4: Clock =8 and page 2 with smallest clock value is replaced.

P#	C	P#	C	P#	C	P#	C	P#	C	P#	C	P#	C	P#	C
7	1	7	1	7	1	2	4	2	4	2	4	2	4	4	8
		0	2	0	2	0	2	0	5	0	5	0	7	0	7
				1	3	1	3	1	3	3	6	3	6	3	6

Memory Management System

LRU Implementation

- *Use of double-linked list.*

There is a head and tail pointer. The most recent used page is the head of the list and time that a page is referenced it is moved to the head of list. Therefore, tail pointer always point to the LRU page.

Page Table

Physical Memory

Memory Management System

LRU Approximation

1- Use of reference bit .

Each page has its own reference bit.

The bit is initialized with zero when the page is coming to the physical Memory.

The bit is set to one if it is referenced again.

The page with reference bit = 0 is the least recently used page.

Reference string: 7, 1, 7, 2 and we have 3 frames

P#	P#	R	P#	R	P#	C	P#	C	P#	C
0	7	0	7	0	7	0	7	1	7	1
1			0	0	0	0	0	0	2	0
2					1	0*	1	0	1	0

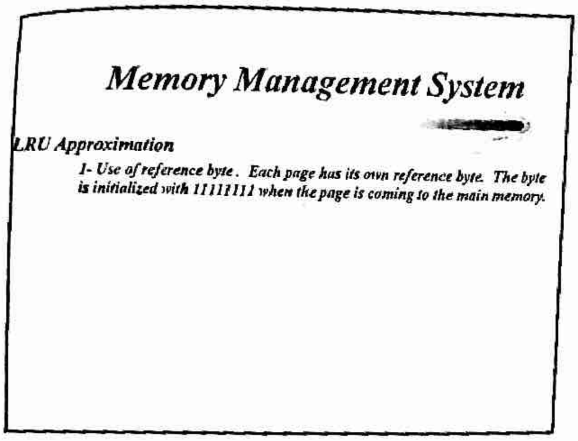
(Either of the two pages 1 or 2 may be replaced)

Memory Management System

Reference bit neither says anything about the history of the page nor the order of page use.

(That is the reason for calling it LRU Approximation)

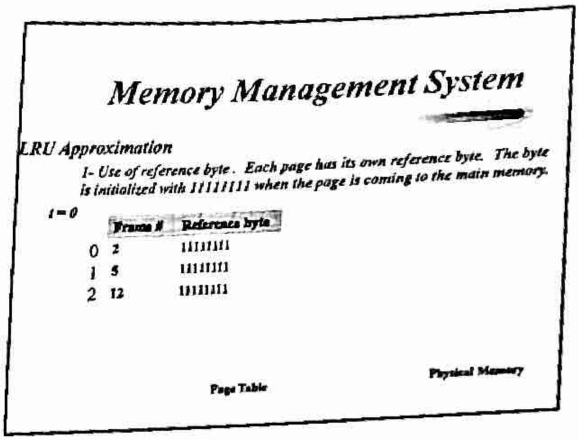
Page Table



Memory Management System

LRU Approximation

1- Use of reference byte . Each page has its own reference byte. The byte is initialized with 11111111 when the page is coming to the main memory.



Memory Management System

LRU Approximation

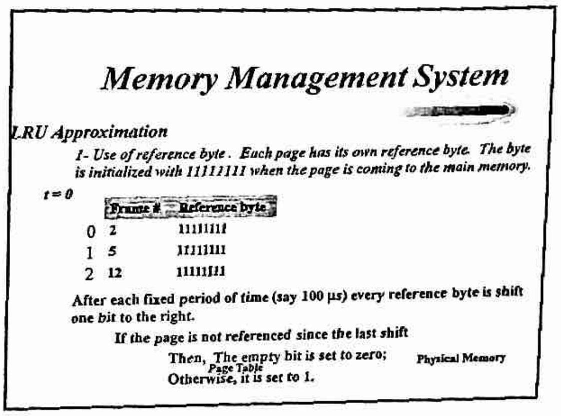
1- Use of reference byte . Each page has its own reference byte. The byte is initialized with 11111111 when the page is coming to the main memory.

t = 0

	Frame #	Reference byte
0	2	11111111
1	5	11111111
2	12	11111111

Page Table

Physical Memory



Memory Management System

LRU Approximation

1- Use of reference byte . Each page has its own reference byte. The byte is initialized with 11111111 when the page is coming to the main memory.

t = 0

	Frame #	Reference byte
0	2	11111111
1	5	11111111
2	12	11111111

After each fixed period of time (say 100 µs) every reference byte is shift one bit to the right.

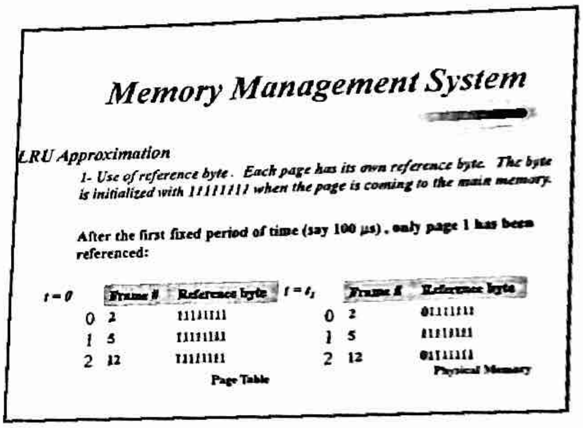
If the page is not referenced since the last shift

Then, The empty bit is set to zero;

Page Table

Otherwise, it is set to 1.

Physical Memory



Memory Management System

LRU Approximation

1- Use of reference byte . Each page has its own reference byte. The byte is initialized with 11111111 when the page is coming to the main memory.

After the first fixed period of time (say 100 µs) , only page 1 has been referenced:

t = 0

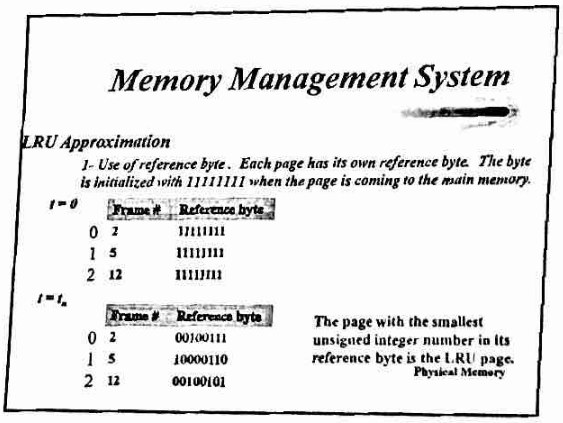
	Frame #	Reference byte
0	2	11111111
1	5	11111111
2	12	11111111

Page Table

t = t₁

	Frame #	Reference byte
0	2	01111111
1	5	11111111
2	12	01111111

Physical Memory



Memory Management System

LRU Approximation

1- Use of reference byte . Each page has its own reference byte. The byte is initialized with 11111111 when the page is coming to the main memory.

t = 0

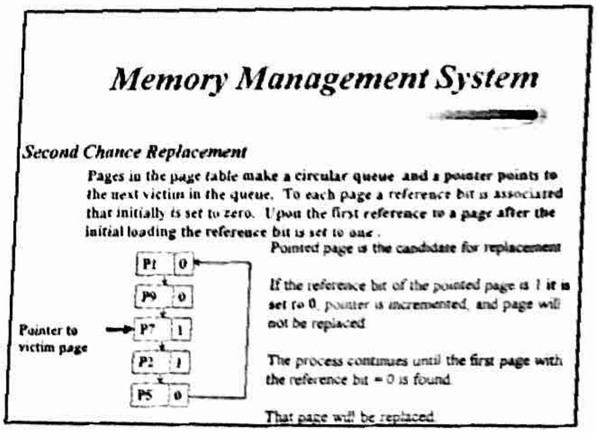
	Frame #	Reference byte
0	2	11111111
1	5	11111111
2	12	11111111

t = t_n

	Frame #	Reference byte
0	2	00100111
1	5	10000110
2	12	00100101

The page with the smallest unsigned integer number in its reference byte is the LRU page.

Physical Memory



Memory Management System

Second Chance Replacement

Pages in the page table make a circular queue and a pointer points to the next victim in the queue. To each page a reference bit is associated that initially is set to zero. Upon the first reference to a page after the initial loading the reference bit is set to one .

Pointed page is the candidate for replacement

If the reference bit of the pointed page is 1 it is set to 0, pointer is incremented, and page will not be replaced

The process continues until the first page with the reference bit = 0 is found

That page will be replaced