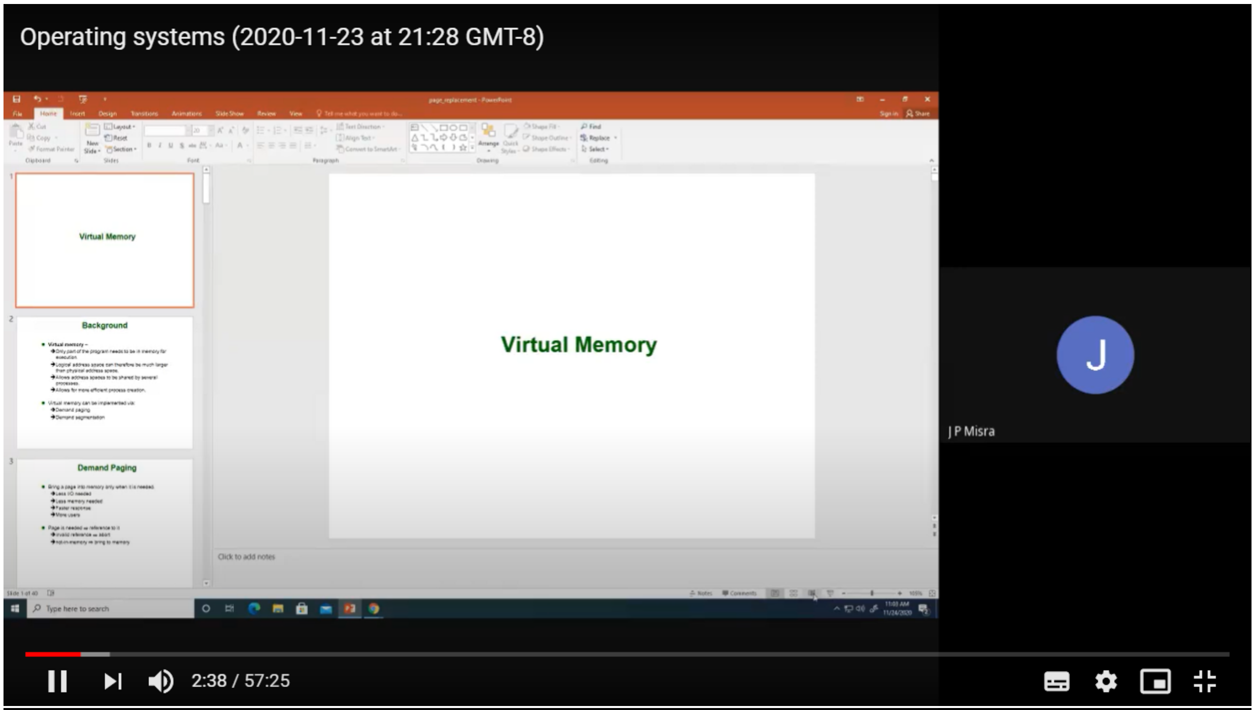
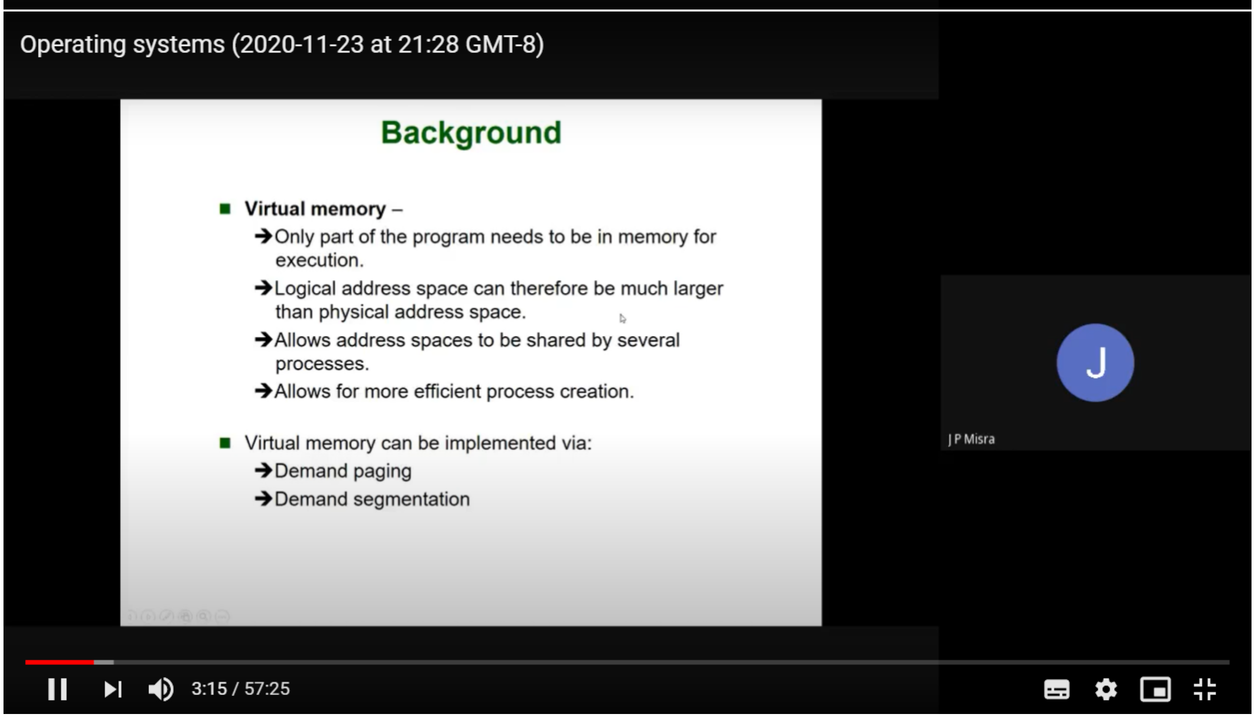
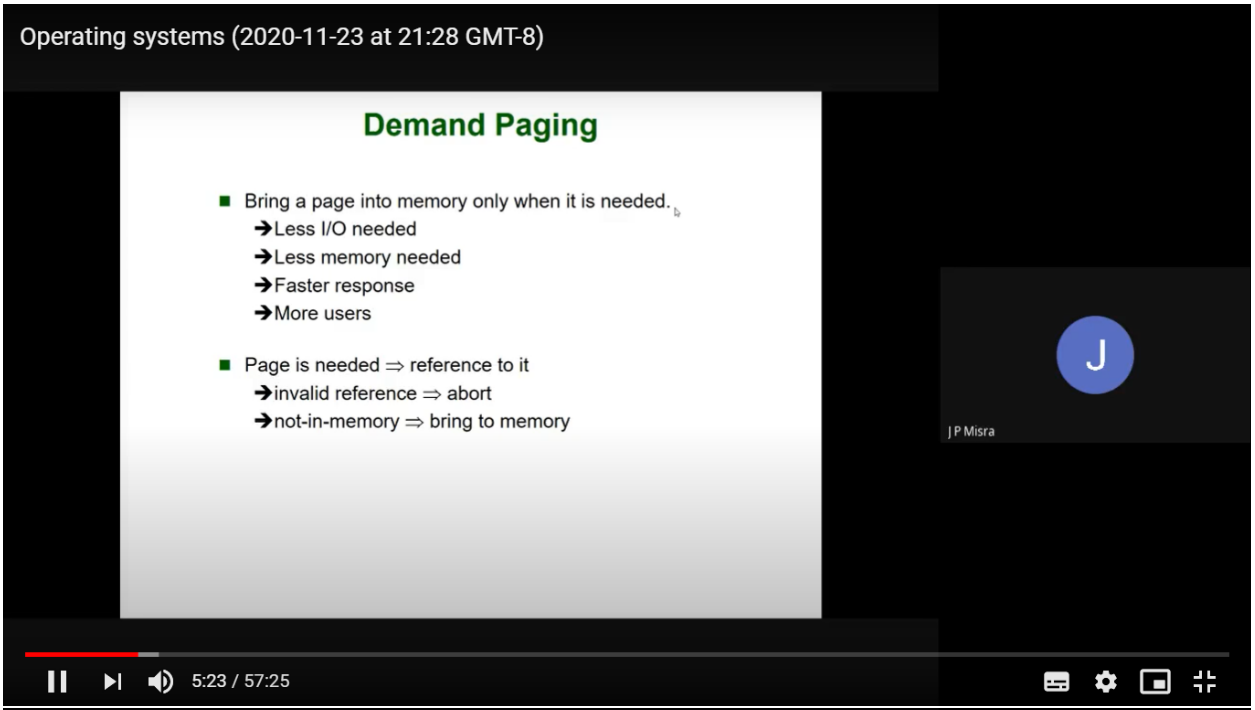
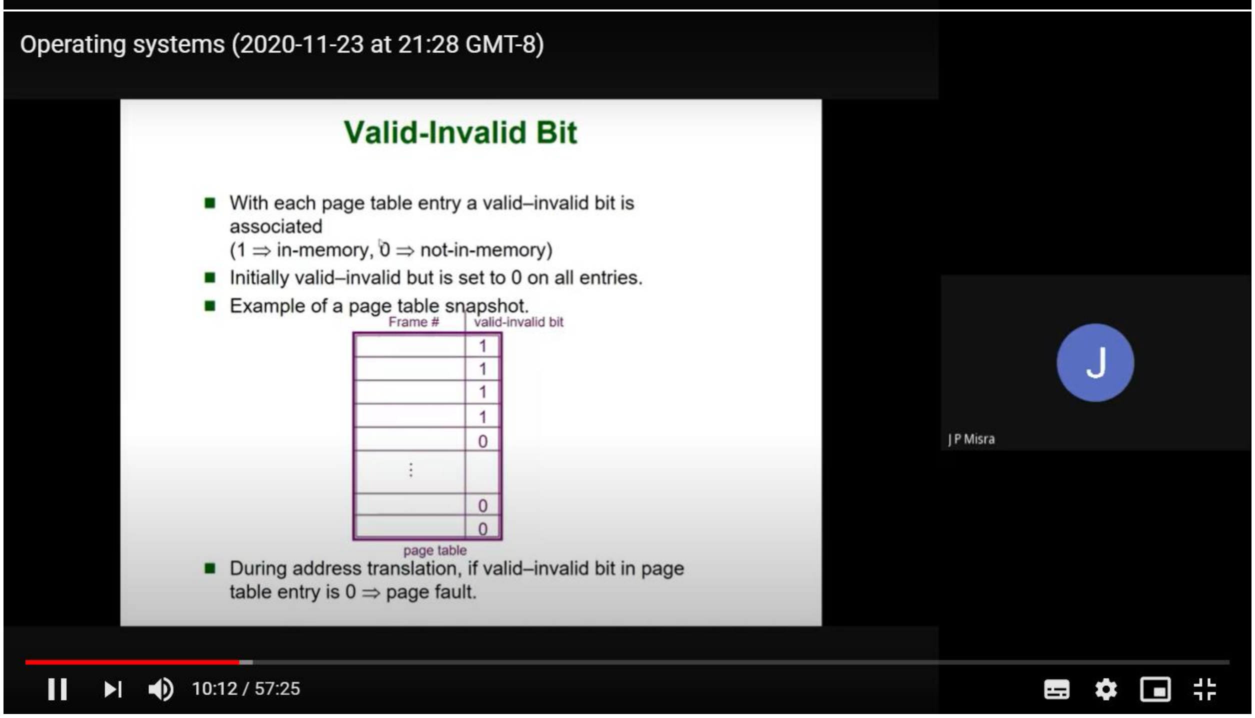
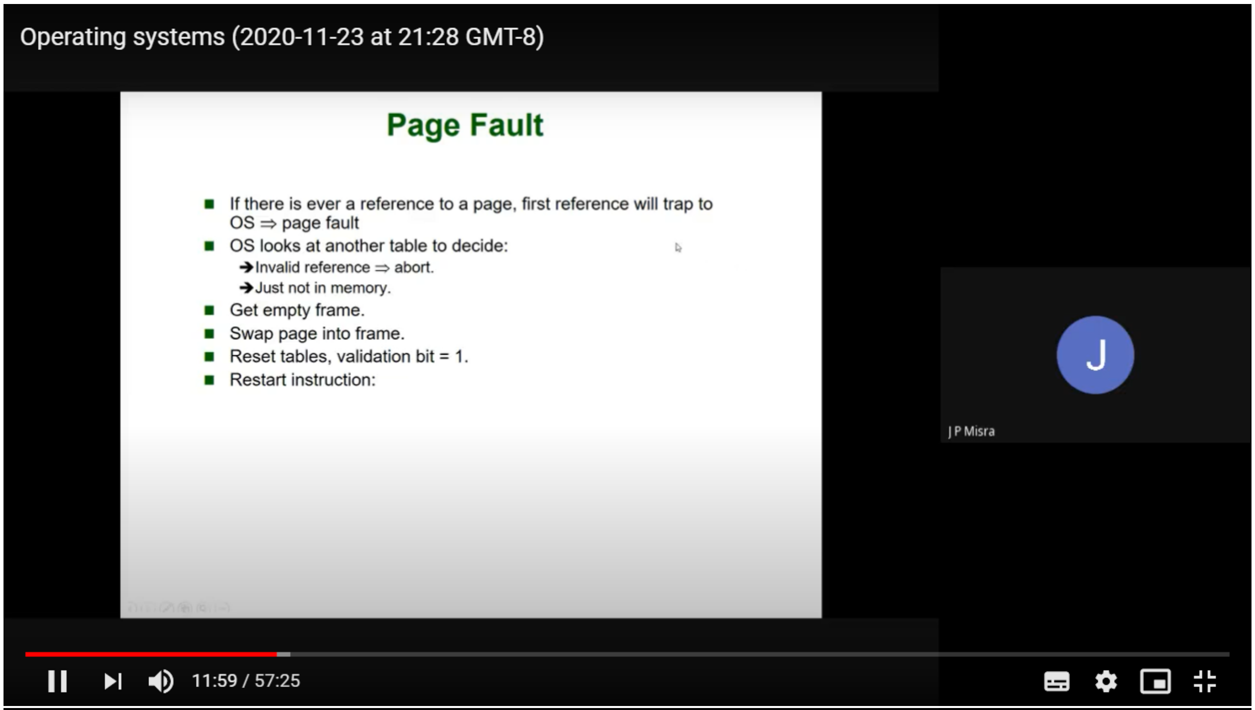
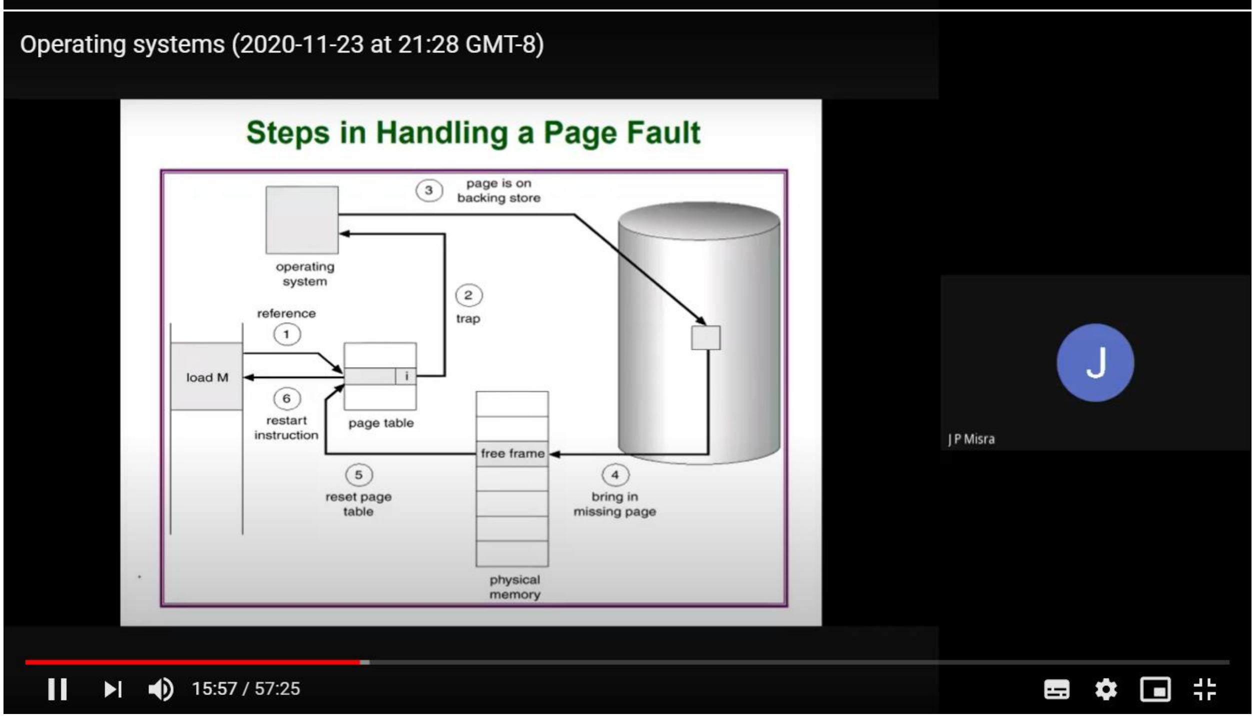

Operating systems (2020-11-23 at 21:28 GMT-8)
Virtual Memory
Background
Demand Paging
Virtual Memory
Click to add notes
J
J P Misra
2:38 / 57:25

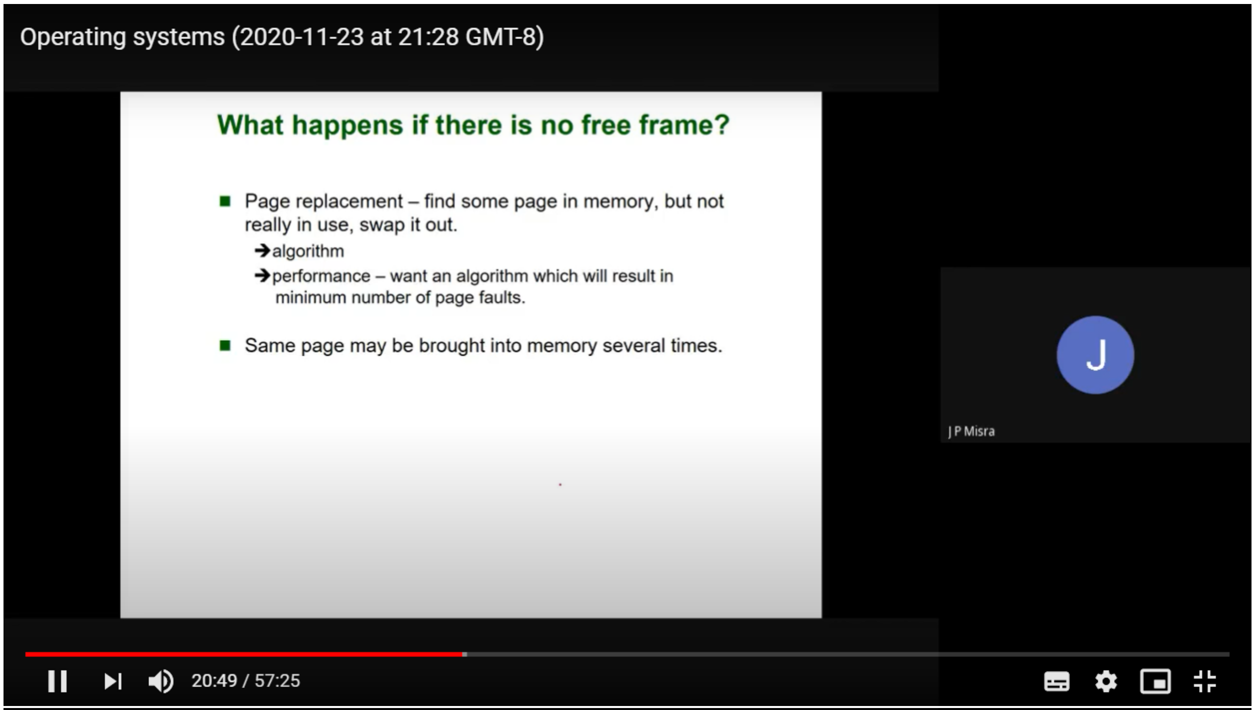

Operating systems (2020-11-23 at 21:28 GMT-8)
Background
Virtual memory –
Only part of the program needs to be in memory for execution.
Logical address space can therefore be much larger than physical address space.
Allows address spaces to be shared by several processes.
Allows for more efficient process creation.
Virtual memory can be implemented via:
Demand paging
Demand segmentation
J
J P Misra
3:15 / 57:25

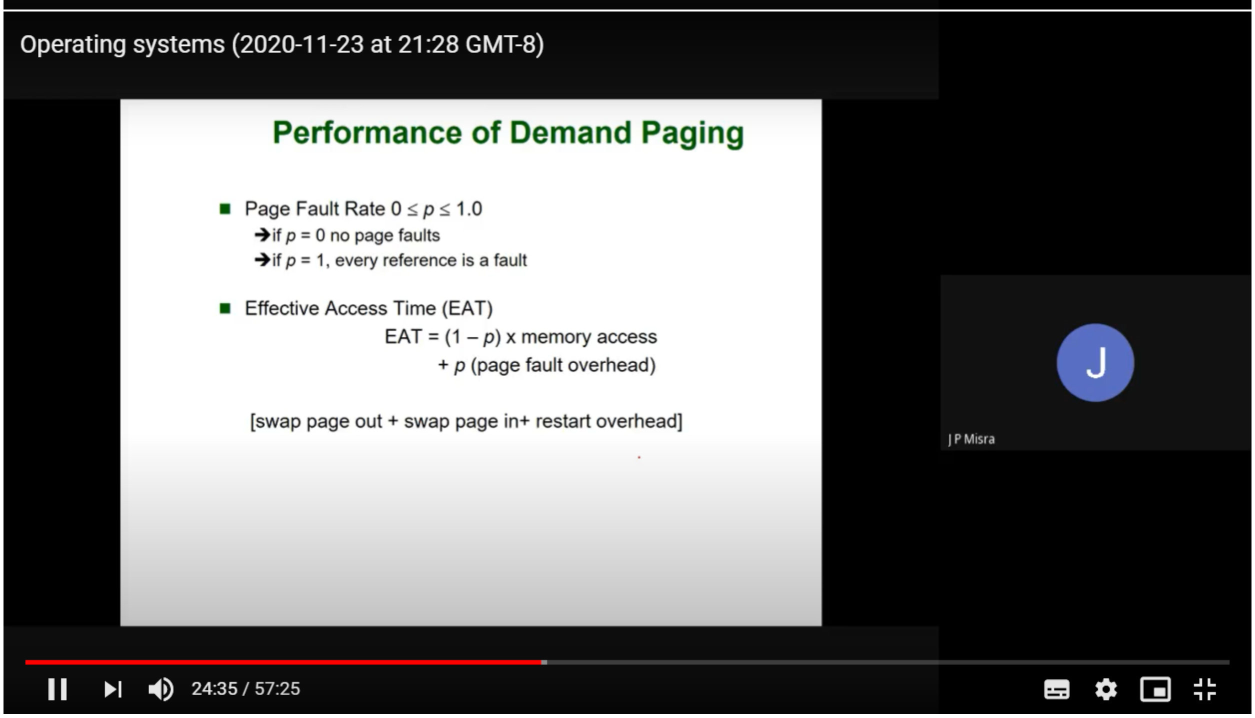

Operating systems (2020-11-23 at 21:28 GMT-8)
Demand Paging
Bring a page into memory only when it is needed.
➔Less I/O needed
➔Less memory needed
➔Faster response
➔More users
Page is needed ⇒ reference to it
➔invalid reference ⇒ abort
➔not-in-memory ⇒ bring to memory
J
J P Misra
5:23 / 57:25

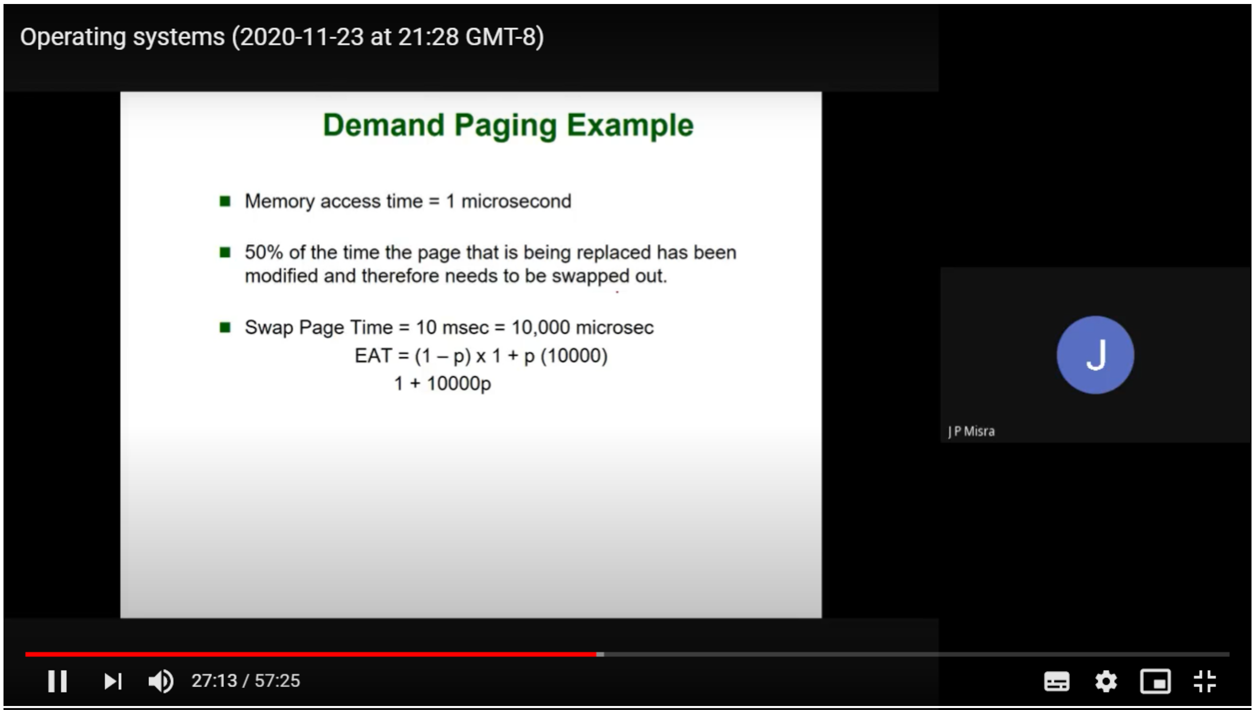

Operating systems (2020-11-23 at 21:28 GMT-8)
Valid-Invalid Bit
With each page table entry a valid–invalid bit is associated
(1 ⇒ in-memory, 0 ⇒ not-in-memory)
Initially valid–invalid but is set to 0 on all entries.
Example of a page table snapshot.
Frame #
valid-invalid bit
1
1
1
1
0
⋮
0
0
page table
During address translation, if valid–invalid bit in page table entry is 0 ⇒ page fault.
J
J P Misra
10:12 / 57:25

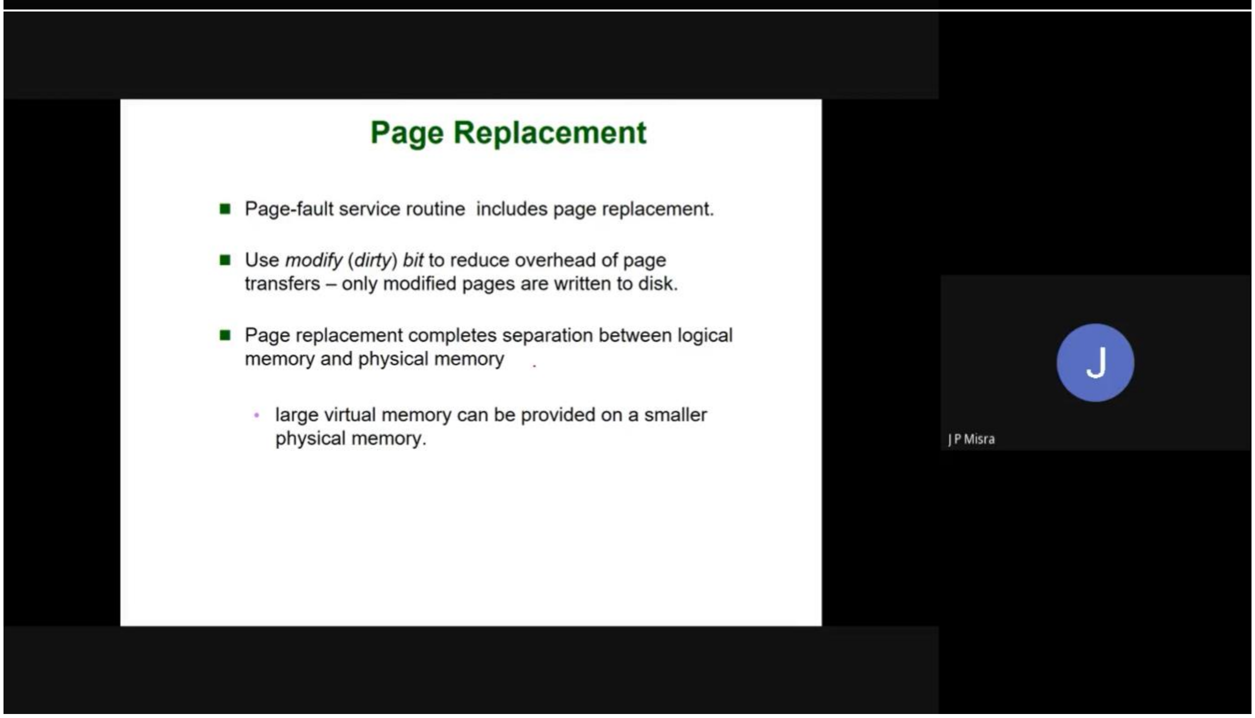

Operating systems (2020-11-23 at 21:28 GMT-8)
Page Fault
If there is ever a reference to a page, first reference will trap to OS ⇒ page fault
OS looks at another table to decide:
→Invalid reference ⇒ abort.
→Just not in memory.
Get empty frame.
Swap page into frame.
Reset tables, validation bit = 1.
Restart instruction:
J
J P Misra
11:59 / 57:25


Operating systems (2020-11-23 at 21:28 GMT-8)
Steps in Handling a Page Fault
3
page is on backing store
operating system
2
trap
reference
1
load M
i
6
restart instruction
page table
free frame
5
reset page table
4
bring in missing page
physical memory
J
J P Misra
15:57 / 57:25


Operating systems (2020-11-23 at 21:28 GMT-8)
What happens if there is no free frame?
Page replacement – find some page in memory, but not really in use, swap it out.
→algorithm
→performance – want an algorithm which will result in minimum number of page faults.
Same page may be brought into memory several times.
J
J P Misra
20:49 / 57:25


Operating systems (2020-11-23 at 21:28 GMT-8)
Performance of Demand Paging
Page Fault Rate $0 \le p \le 1.0$
→if $p = 0$ no page faults
→if $p = 1$, every reference is a fault
Effective Access Time (EAT)
EAT = $(1 – p)$ x memory access
+ p (page fault overhead)
[swap page out + swap page in+ restart overhead]
J
J P Misra
24:35 / 57:25


Operating systems (2020-11-23 at 21:28 GMT-8)
Demand Paging Example
Memory access time = 1 microsecond
50% of the time the page that is being replaced has been modified and therefore needs to be swapped out.
Swap Page Time = 10 msec = 10,000 microsec
EAT = (1 – p) x 1 + p (10000)
1 + 10000p
J
J P Misra
27:13 / 57:25


Page Replacement
Page-fault service routine includes page replacement.
Use *modify* (*dirty*) *bit* to reduce overhead of page transfers – only modified pages are written to disk.
Page replacement completes separation between logical memory and physical memory
large virtual memory can be provided on a smaller physical memory.
J
J P Misra

Replacement Policy

- Which page to replaced?
- Page removed should be the page least likely to be referenced in the near future
- Most policies predict the future behavior on the basis of past behavior

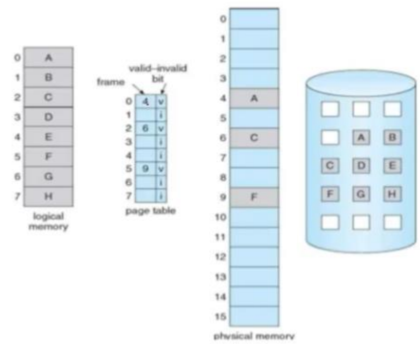
J P Misra

Replacement Policy

- Frame Locking
  - ➔If frame is locked, it may not be replaced
  - ➔Kernel of the operating system
  - ➔Key control structures
  - ➔I/O buffers
  - ➔Associate a lock bit with each frame

J P Misra

Page table when some pages are not in memory



J

J P Misra

What is Page fault ?

- When the data (page) requested by program is not present in main memory , it is known as page fault
- Why page replacement required ?
  - System may not have enough RAM to store all required data

J

J P Misra

Page Replacement Algorithms

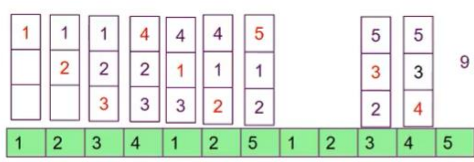
- Want lowest page-fault rate.
- Evaluate algorithm by running it on a particular string of memory references (reference string) and computing the number of page faults on that string.
- In all our examples, the reference string is

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

J P Misra

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)



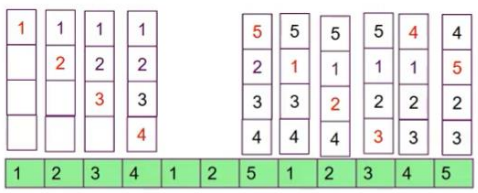
9 page faults

J P Misra

First-In-First-Out (FIFO) Algorithm

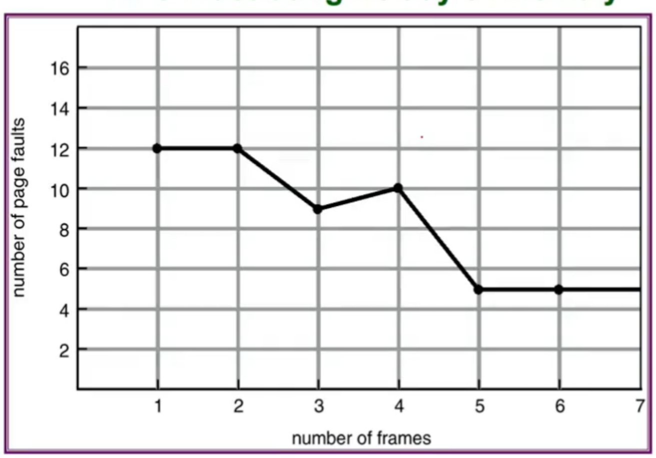
- Reference string: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5
- 4 frames (4 pages can be in memory at a time)
- In general more frames ⇒ less page faults
- FIFO Replacement – Belady's Anomaly

10 page faults



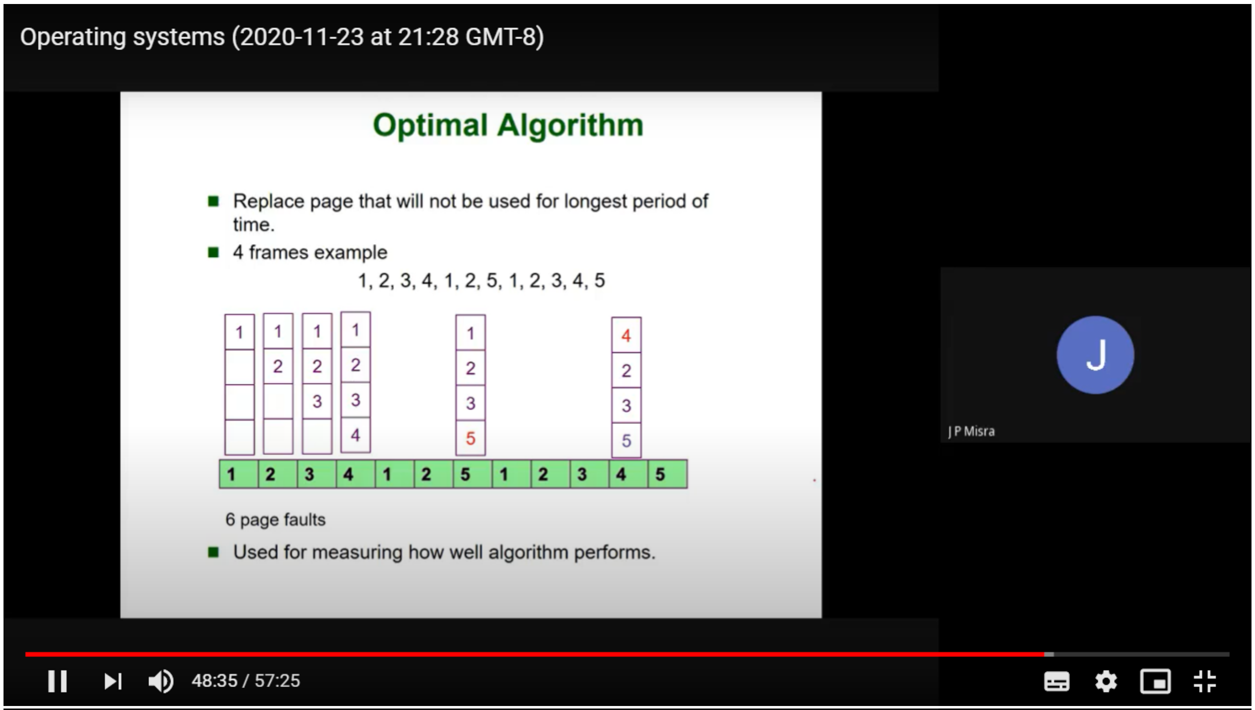
J P Misra

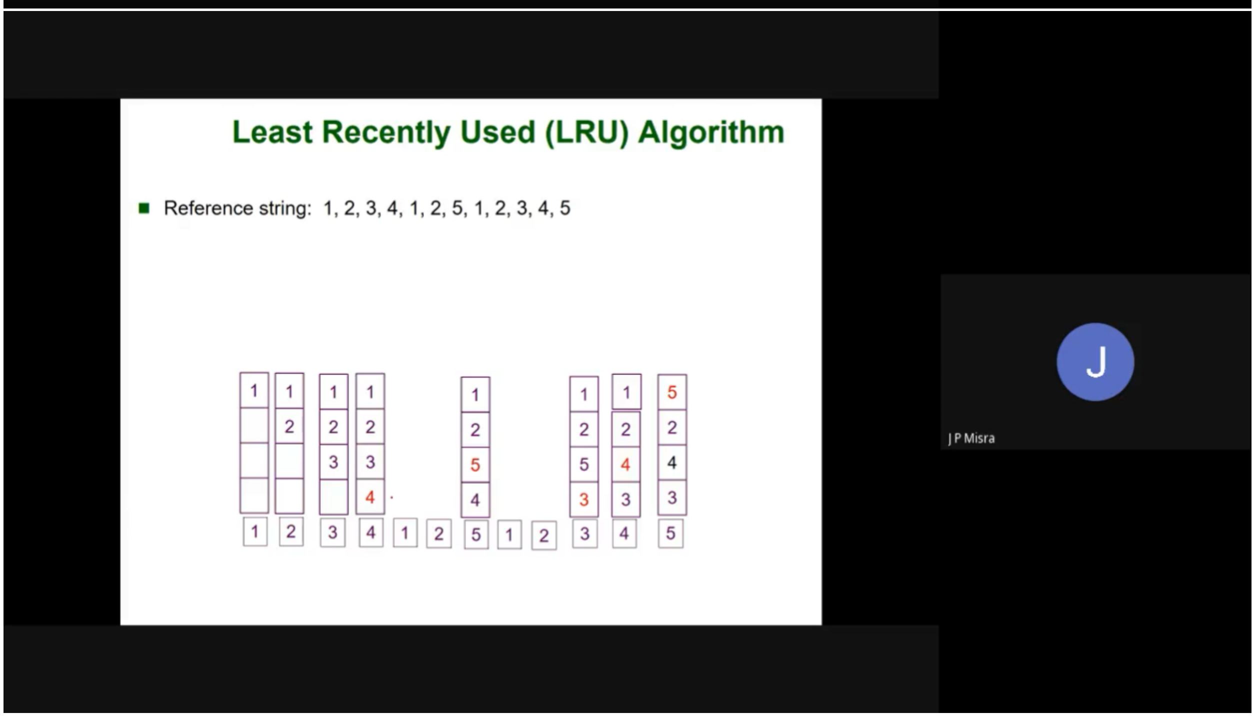
FIFO Illustrating Belady's Anomaly



J P Misra

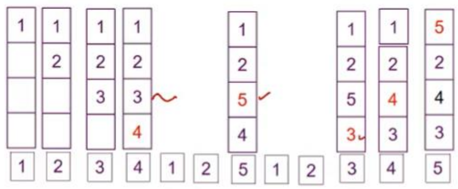
Minimize # of PFs when # frames >= # of unique pages





Least Recently Used (LRU) Algorithm

- Reference string: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5



J P Misra

LRU Implementation

- Counter implementation
  - → Every page entry has a counter; every time page is referenced through this entry, copy the clock into the counter.
  - → When a page needs to be replaced , look at the counters to determine which page to replace .

J P Misra

LRU Algorithm (Cont.)

- Stack implementation
  - keep a stack of page numbers in a double link form:

→whenever Page is referenced:

→move it to the top

→No search for replacement

J

J P Misra

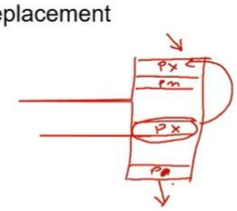
LRU Algorithm (Cont.)

- Stack implementation
  - keep a stack of page numbers in a double link form:

→whenever Page is referenced:

→move it to the top

→No search for replacement



J

J P Misra

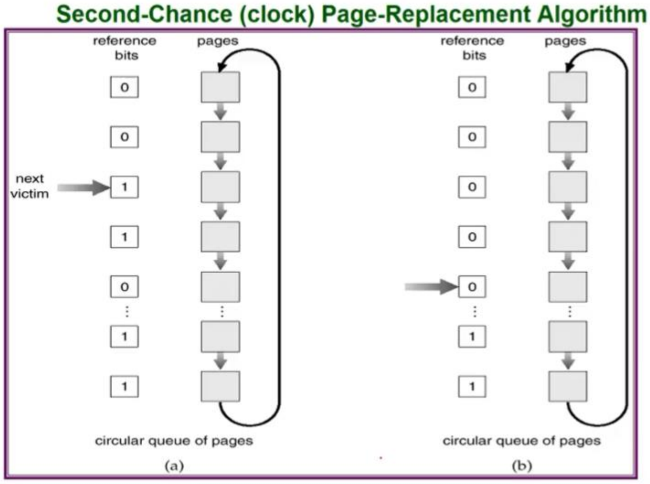
LRU Approximation Algorithms

- Reference bit
  - ➔With each page associate a bit, initially = 0
  - ➔When page is referenced bit set to 1.
  - ➔Replace the one which is 0 (if one exists). We do not know the order, however.
- Second chance
  - ➔Need reference bit.
  - ➔If page to be replaced (in clock wise order) has reference bit = 1. then:
    - ➔set reference bit 0.
    - ➔leave page in memory.
    - ➔replace next page (in clock wise order), subject to same rules.

J

J P Misra

Second-Chance (clock) Page-Replacement Algorithm



J

J P Misra

Enhanced Clock Policy

- In addition to reference bit use modify bit also
  - (0 ,0) not referenced not modified
  - (0, 1) Not recently used but modified
  - (1,0) recently used but not modified
  - (1,1) recently used and modified

J P Misra

Comparison

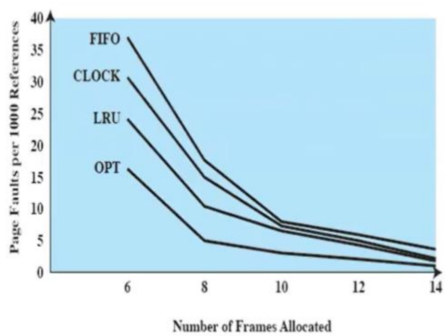


Figure 8.17 Comparison of Fixed-Allocation, Local Page Replacement Algorithms

J P Misra

Counting Algorithms

- Keep a counter of the number of references that have been made to each page.
- LFU Algorithm: replaces page with smallest count.
- MFU Algorithm: based on the argument that the page with the smallest count was probably just brought in and has yet to be used.

J P Misra

Allocation of Frames

- Each process needs **minimum** number of pages.
- Example:
- MOV source, destination
  - ➔instruction is 4 bytes, might span 2 pages.
  - ➔2 pages to handle **from**.
  - ➔2 pages to handle **to**.
- Two major allocation schemes.
  - ➔fixed allocation
  - ➔priority allocation

J P Misra

Fixed Allocation

- Equal allocation – e.g., if 100 frames and 5 processes, give each 20 pages.
- Proportional allocation – Allocate according to the size of process.

s_i = size of process p_i

$S = \sum s_i$

m = total number of frames

a_i = allocation for $p_i = \frac{s_i}{S} \times m$

$$m = 64$$
$$s_1 = 10$$
$$s_2 = 127$$
$$a_1 = \frac{10}{137} \times 64 \approx 5$$
$$a_2 = \frac{127}{137} \times 64 \approx 59$$

J

J P Misra

OS lect (2020-11-25 at 03:28 GMT-8)

Priority Allocation

- Use a proportional allocation scheme using priorities rather than size.
- If process P_i generates a page fault,
  - → select for replacement one of its frames.
  - → select for replacement a frame from a process with lower priority number.

J

J P Misra

44:34 / 55:42

Global vs. Local Allocation

- **Global** replacement – process selects a replacement frame from the set of all frames; one process can take a frame from another.
  - → Process cannot control its own Page fault rate
- **Local** replacement – each process selects from only its own set of allocated frames.
  - →Number of frames allocated to a process do not change
  - →Does not make use of less used pages belonging to other processes

J

J P Misra

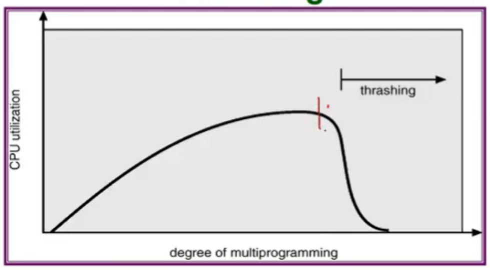
Thrashing

- If a process does not have “enough” pages, the page-fault rate is very high. This leads to:
  - →low CPU utilization.
  - →operating system thinks that it needs to increase the degree of multiprogramming.
  - →another process added to the system.
- **Thrashing** ≡ a process is busy swapping pages in and out.
  - → More pronounced for Global page replacement policy

J

J P Misra

Thrashing



- Why does paging work?
  Locality model
  - ➔Process migrates from one locality to another.
  - ➔Localities may overlap.
- Why does thrashing occur?
  Σ size of locality > total memory size

J

J P Misra

Working-Set Model

- $\Delta \equiv$ working-set window $\equiv$ a fixed number of page references
  Example: 10,000 instruction
- WSS_i (working set of Process P_i) =
  total number of pages referenced in the most recent Δ (varies in time)
  - ➔if Δ too small will not encompass entire locality.
  - ➔if Δ too large will encompass several localities.
  - ➔if $\Delta = \infty \Rightarrow$ will encompass entire program.
- $D = \Sigma\, WSS_i \equiv$ total demand frames
- if $D > m$*(Total number of available frames)* $\Rightarrow$ Thrashing
- Policy if $D >$ m, then suspend one of the processes.

J

J P Misra