Fundamentos de los Sistemas Operativos (FSO)

Departamento de Informática de Sistemas y Computadoras (DISCA)
Universitat Politècnica de València

Part 3: Memory management

Unit 10

Virtual memory (II)

fSO

DISCA

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Goals and bibliography

- **Goals**
  - To understand **2nd chance** page replacement algorithm as an LRU approximation
  - To know the **thrashing** problem and its solutions
  - To analyse memory **frame management** techniques

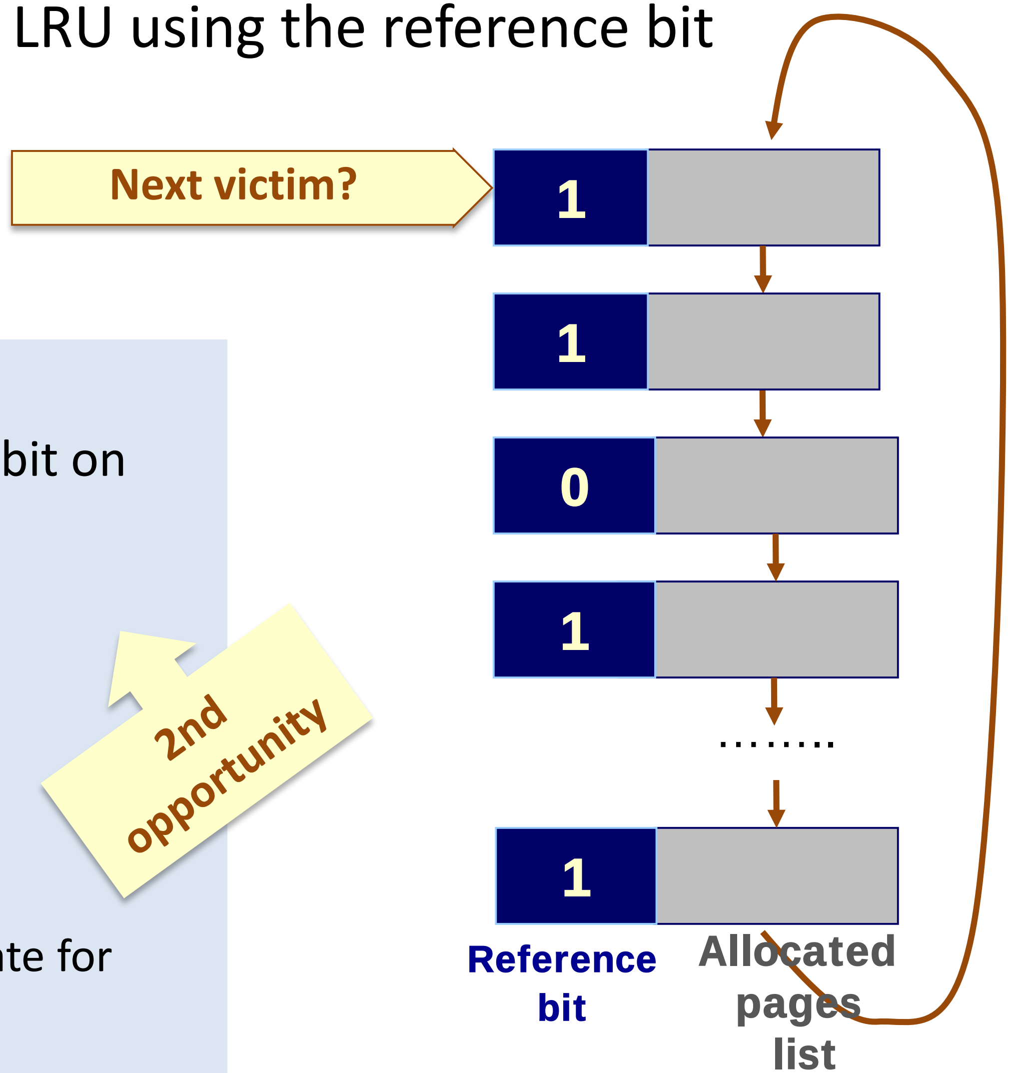
- Bibliography
  - Silberschatz, chapter 9

Content

- **2nd chance replacement algorithm**
- Frame allocation
- Thrashing
- Frame reservation

2nd chance replacement algorithm

- Supporting LRU as replacement algorithm is too expensive
- **Solution**: To approximate LRU using the reference bit

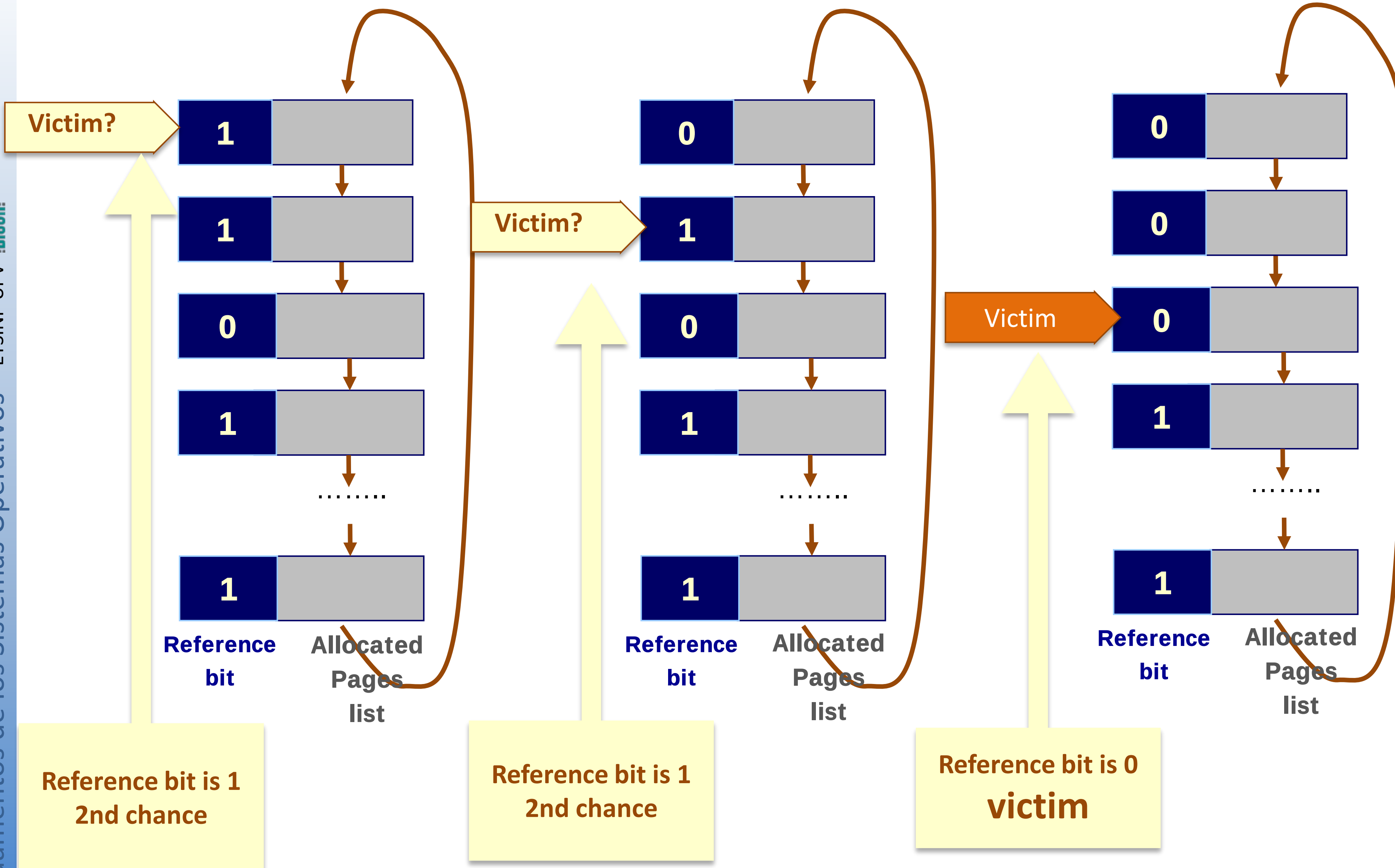


2nd chance algorithm

- It reviews the page reference bit on FIFO ordering
- **If the page reference bit is 1**
  - Put reference bit to 0
  - Let page in memory
- **If the page reference bit is 0**
  - The page is chosen as victim
  - Next page is the first candidate for next victim

2nd opportunity

2nd chance replacement algorithm

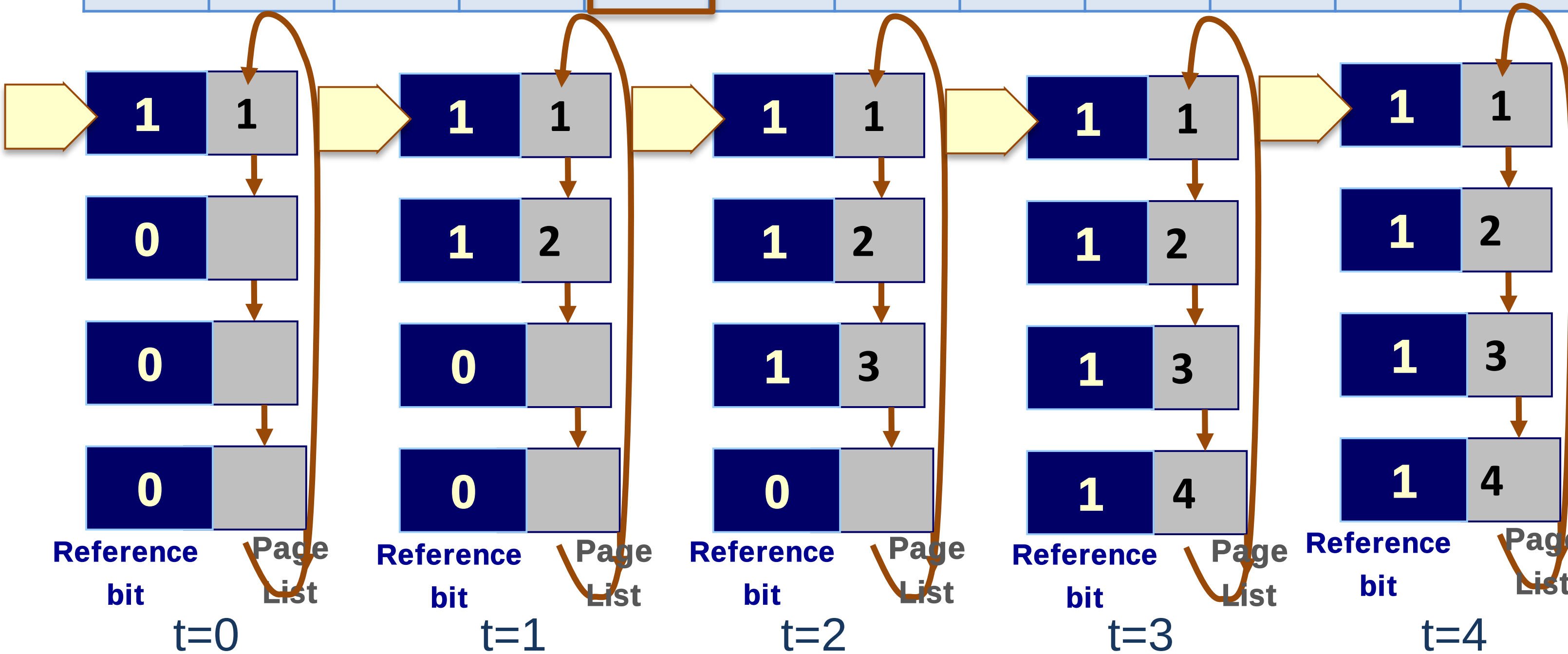


2nd chance replacement algorithm

fSO

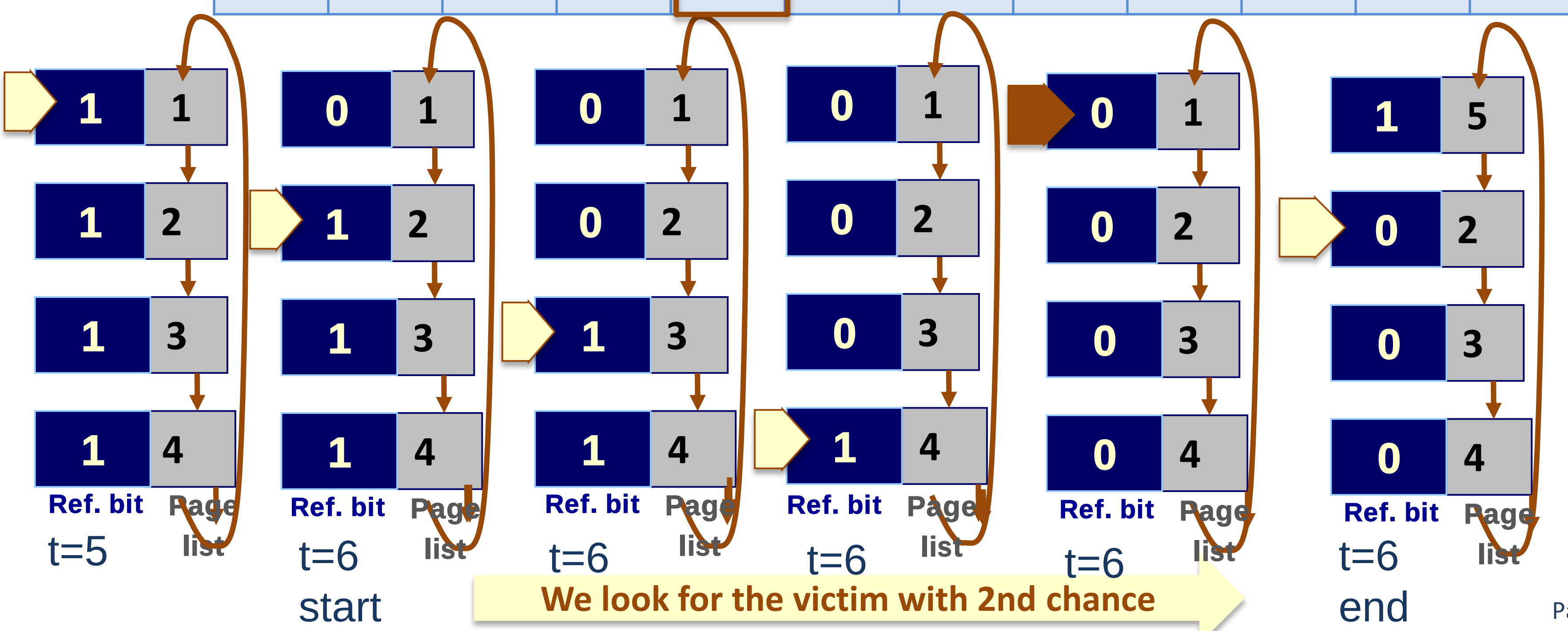
Example: Main memory with 4 frames

T	0	1	2	3	4	5	6	7	8	9	10
Reference string	1	2	3	4	1	2	5	2	3	1	2
Frame 0	1	1	1	1	1						
Frame 1		2	2	2	2						
Frame 2			3	3	3						
Frame 3				4	4						



Example: Main memory with 4 frames

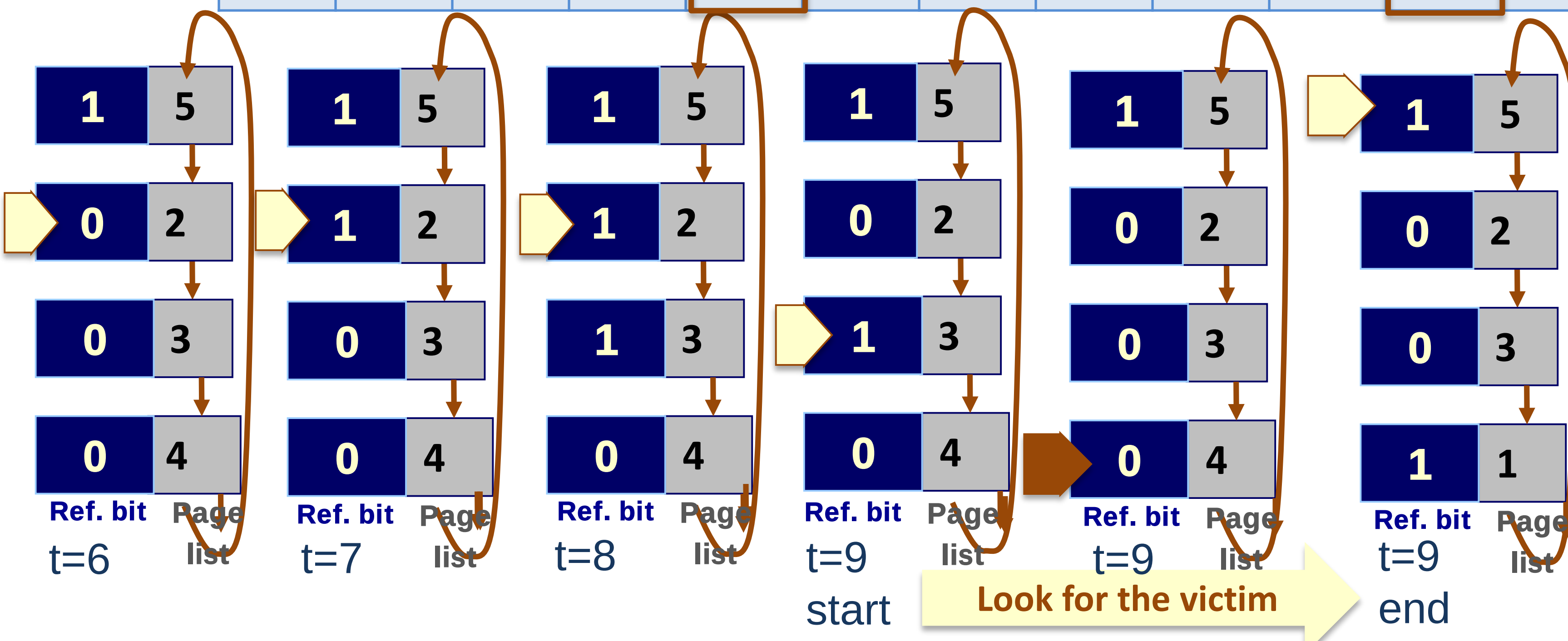
T	0	1	2	3	4	5	6	7	8	9	10
Reference string	1	2	3	4	1	2	5	2	3	1	2
Frame 0	1	1	1	1	1	1	5				
Frame 1		2	2	2	2	2	2				
Frame 2			3	3	3	3	3				
Frame 3				4	4	4	4				



2nd chance replacement algorithm

Example: Main memory with 4 frames

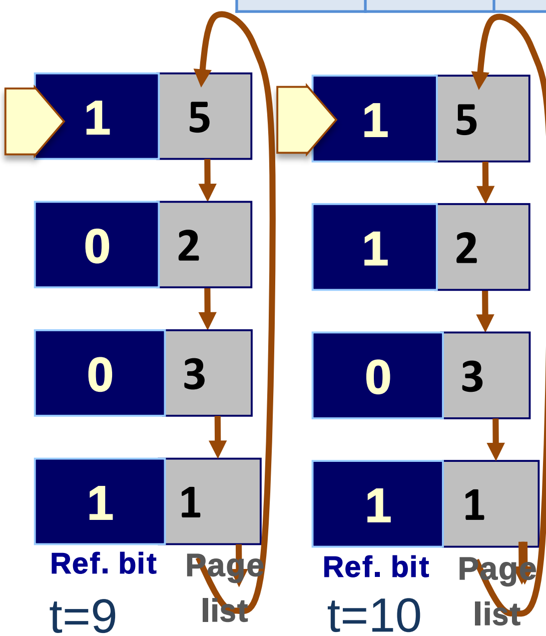
T	0	1	2	3	4	5	6	7	8	9	10
Reference string	1	2	3	4	1	2	5	2	3	1	2
Frame 0	1	1	1	1	1	1	5	5	5	5	
Frame 1		2	2	2	2	2	2	2	2	2	
Frame 2			3	3	3	3	3	3	3	3	
Frame 3				4	4	4	4	4	4	1	



2nd chance replacement algorithm

Example: Main memory with 4 frames

T	0	1	2	3	4	5	6	7	8	9	10
Reference string	1	2	3	4	1	2	5	2	3	1	2
Frame 0	1	1	1	1	1	1	5	5	5	5	5
Frame 1		2	2	2	2	2	2	2	2	2	2
Frame 2			3	3	3	3	3	3	3	3	3
Frame 3				4	4	4	4	4	4	1	1



Total page faults 6 (with replacement 2)

Content

- 2nd chance replacement algorithm
- **Frame allocation**
- Thrashing
- Frame reservation

Frame allocation

- Frame allocation problem
  - Free frame list:
    - Frame management requires a **data structure** where **free frames** are kept
  - Frame to process delivery policy and the OS
    - The OS gets the required number of frames to execute itself
    - Processes receive the minimum initial number of frames and the remaining ones on demand
    - The minimum number of initally assigned frames depends on the indirection level in the CPU instruction set → To execute an instruction all its operands must be allocated in main memory

Frame allocation

- Frame delivery policies given **m frames** and **n processes**
  - **Fair allocation**: All processes allocate **A_i** frames equally

    $$A_i = m/n$$

  - **Proportional allocation**: A process P_i with size S_i allocate **A_i** frames computed as:

    $$A_i = \frac{S_i}{\sum S_i} * m$$

  - **Priority allocation**: higher priority processes allocate more frames

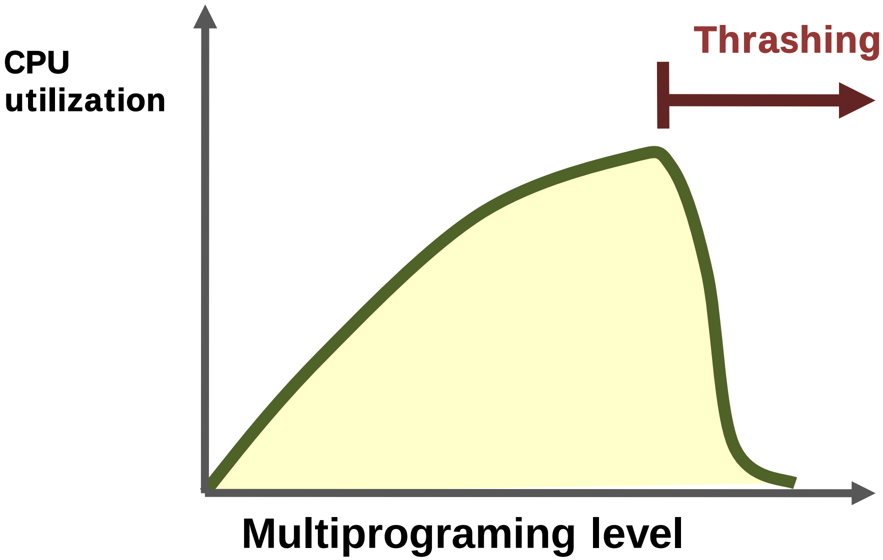
Frame allocation

- **Replacement policy scope**
  - **Local replacement**: only pages allocated to the process that generates the page fault can be replaced
    - It cannot choose a victim from another process
    - The number of process allocated frames does not change
    - A process execution is not affected by the remaining processes
      - Advantage: Sensible response time
      - Disadvantage: Worse global memory management
  - **Global replacement**: the victim is chosen between all allocated pages in main memory
    - The victim can belong to another process
    - The number of process allocated frames can change
      - Disadvantage: Response time sensitive to system load
      - Advantage: Better global memory management

- 2nd chance replacement algorithm
- Frame allocation
- **Thrashing**
- Frame reservation

Thrashing

- The thrashing problem:
  - Memory becomes scarce and processes generate **a lot of page faults** → I/O time becomes dominant
  - The OS allows more processes entering for execution regarding the **low level of CPU use**
  - This makes things worse because the same amount of memory has to be shared with more and more processes → page fault rate keeps increasing



How to solve thrashing?

–To anticipate an to prevent the problem

–Using a working set model

–Controlling page fault rate

–Once thrashing is detected

–Swap out processes with a medium term scheduler

Thrashing

- Reference locality principle
  - Instructions and data processed recently (and the ones close to them) have a high probability of being processed in the near future
  - Locality:
    - Set of pages that a process uses as a whole
    - It is hard to identify
  - Thrashing happens when

    locality sizes > total main memory size

Thrashing

- **Working set model**: preventive technique
  - It assumes **reference locality principle**
  - Obtain the **number of pages that a process needs to have simultaneously in memory** to avoid thrashing
  - **Working set (WS):**
    - Set of pages accessed in a process last referenced logical addresses
    - Working set window:
      - Fixed consecutive number of references Δ used to compute WS
      - WS is made of the set of pages accessed in the last Δ references
    - Working set size (WSS): Number of different pages that belong to the WS
    - In a system with **m frames and n processes** $P_1 \ldots P_n$ there is thrashing when $\boldsymbol{\Sigma\, WSS_i > m}$

- **Example of WS model**
  - WS window Δ = 9
  - Compute WS and WSS in t1 and t2

Sequence of referenced pages

... 5 3 5 2 3 5 2 3 5 4 5 4 5 9 8 9 4 6 9 2 3 9 2

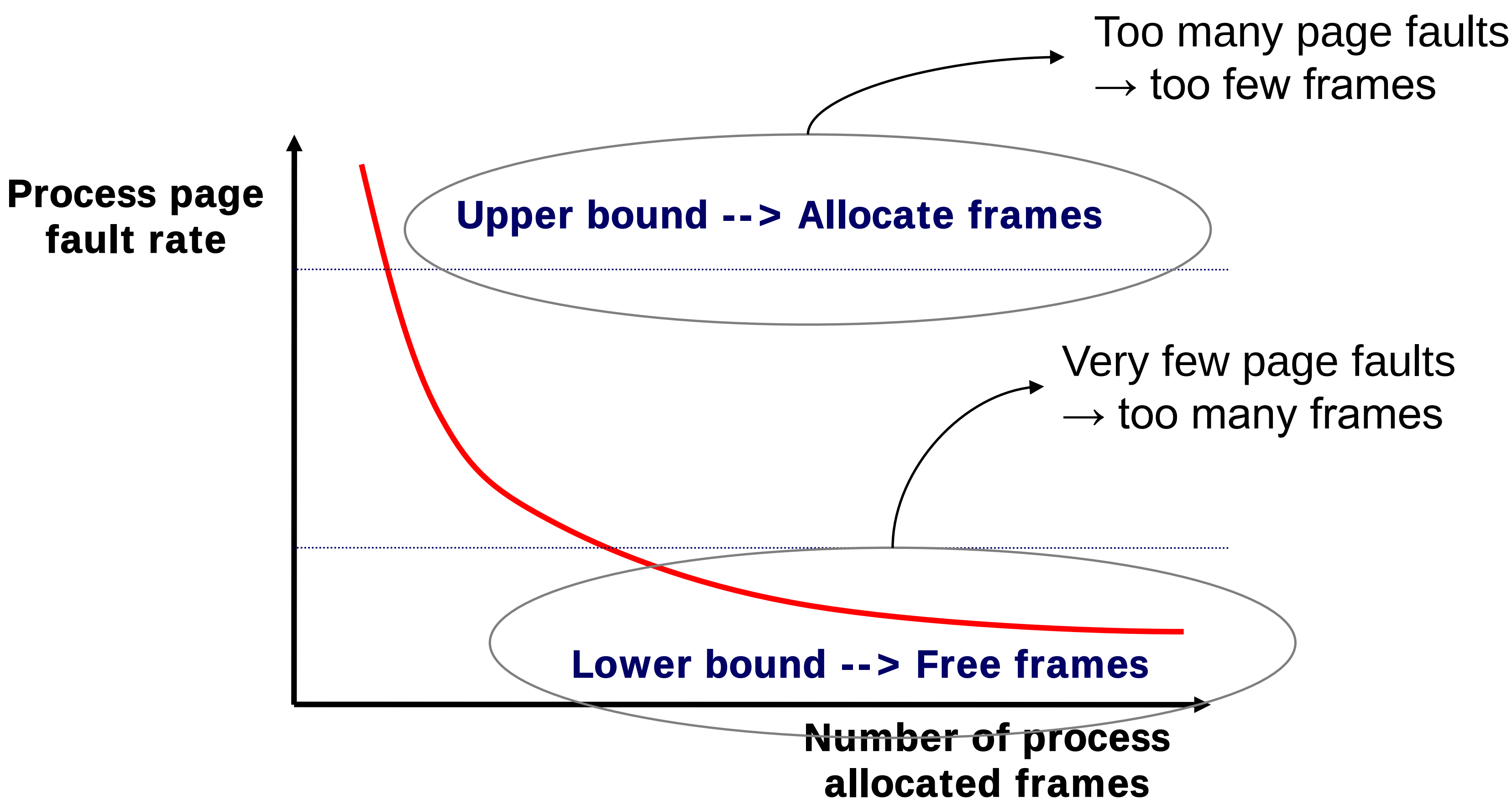
Δ=9 Δ=9

t1 **t2**

WS(t1) = {2,3,5}
WSS(t1)=3

WS(t2) = {4,5,6,8,9}
WSS(t2)=5

- **Page fault rate control**
  - Preventive technique that analyses directly the page fault rate to guess if thrashing is near to happen



Content

- 2nd chance replacement algorithm
- Frame allocation
- Thrashing
- **Frame reservation**

Frame reservation

- **Concept**
  - Modern OS keep a percentage of main memory as a store of free frames (**reservation frames**)
- **Goals**
  - **To reduce the time taken to serve a page fault**
    - Attempt to have free frames available
    - The replacement algorithm is used:
      - Only when the free frame level gets too low
      - To look for victims to amortize its use
    - Page out
      - Several pages are written at once to disk
  - **To avoid thrashing**

Frame reservation

- **Reservation frame management**
  - The OS guarantees that there will be always a set of free frames
  - Some thresholds are set:
    - Minimum number of free frames (M_{MIN})
    - Recommended number of free frames (M_{REC})

      $\mathbf{M_{REC} >> M_{MIN}}$
  - Very efficient replacement algorithms are NOT required
    - The first VMS systems used FIFO because their MMU did not have reference bit
    - It is common to use a 2nd chance algorithm (Windows, UNIX SVR4, UNIX 4.4BSD, Linux, HP OpenVMS, ...).

Frame reservation

- **Monitor process**
  - There is an internal process that periodically **accounts for the number of free frames** (frame_free):
    - If **frame_free > Mrec** then do nothing
    - If **frame_free < Mmin** then:
      - Swap out some processes until reaching REC free frames
      - Victims are process that spent more time suspended
    - If **Mmin <= frame_free <= Mrec** then:
      - Seek for processes with too many frames (very low frame rate) to “steal” them some frames applying a replacement algorithm
      - Several victims are selected in every process, the actual number differs on every OS

Frame reservation

- **Frame reservation management in thrashing**
  - When thrashing happens the number of free frames decreases quickly
  - The process monitor detects it when:
    - Between two monitor activation **frame_free** decreases too much
  - Solutions:
    - Swap out whole processes until reaching **frame_free** = Mrec
      - OpenVMS and Windows NT
    - Free a constant number of frames in every monitor activation if **frame_free** < Mrec
      - The monitor activation frequency increases
      - UNIX SVR4

Frame reservation

- **Reservation frames content**
  - The frame of a victim selected by the OS **is not allocated immediately to another page**, instead the frame goes to the **reservation stock**
    - If victim pages are referenced again soon by the process:
      - There is a high probability that they be in the reservation stock and then they can be relocated **without having to read them on disk**
      - The OS remembers which is the content of every frame in the reserve stock
    - If frames included in the reservation stock correspond to **modified pages**:
      - **They are not considered to solve page faults immediately** because its content has to be written on disk (into a file or paging area)
        - » When a **threshold is reached** all these pages are written at once
        - » Page out **overhead is amortized** -> the number of global page writing is minimized
      - After writing on disk, frames become free and can serve page faults