Fundamentos de los Sistemas Operativos (FSO)

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

Part 4: Memory management

Unit 12

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

- **LRU review: problem**
- 2nd chance replacement algorithm
- Frame allocation
- Thrashing
- Frame reservation

LRU review

- **Optimal replacement algorithm**
  - Victim page: the one which take longer to be referenced
  - Minimum number of faults -> impossible to implement (future is unknown)

- **LRU (Least Recently Used).** [Optimal approximation]
  - Victim page: the one that lasted more without being referenced
  - It is a stack algorithm
  - Implementation
    - Using counters: Every page has associated a counter or timestamp
    - Using an ordered list of referenced pages: When a page is referenced it is moved to the end. The victim page is the first one
    - Using a stack: when a page is referenced, it is added on top. The victim page is the last one. As a referenced page could be in the middle of the stack, it is better to use a double linked list with a head and tail pointer.

Exercise 4. Replacement scope

On a virtual memory system, with 1024 byte page size, the OS has allocated 6 frames (from 0 to 5) to two processes A y B. At time t = 10, A and B page tables have the following content:

Process A page table

	Frame	Valid bit	Counter
0		i	
1		i	
2	2	v	10
3	5	v	3
4		i	
5	4	v	5
6		i	
7		i	

Process B page table

	Frame	Valid bit	Counter
0		i	
1		i	
2		i	
3	1	v	2
4		i	
5		i	
6		i	
7		i	

Then the processes emit the following logical address sequence. Consider that all the addresses are legal:

A,100; A,4000; B,100; A,7000; B,2100; B,1028; A,5800; A,100

Obtain what pages are allocated on every frame and the physical address translation of the first and the last access in the following situations:

a) The replacement algorithm is **LRU** with **global scope**
b) The replacement algorithm is **LRU** with **local scope**. Process A has 4 frames and process B has 2 frames

Exercise 4. Replacement scope

On a virtual memory system, with 1024 byte page size, the OS has allocated 6 frames (from 0 to 5) to two processes A y B. At time t = 10, A and B page tables have the following content:

A and B have 8 pages
1 page 1024 bytes = 2^10
Size 8 * 1024 = 8192
Number of bits: 2^13 = 8192
Logical address: 3 + 10 bits

Process A page table

	Frame	Valid bit	Counter
0		i	
1		i	
2	2	v	10
3	5	v	3
4		i	
5	4	v	5
6		i	
7		i	

Process B page table

	Frame	Valid bit	Counter
0		i	
1		i	
2		i	
3	1	v	2
4		i	
5		i	
6		i	
7		i	

a) The replacement algorithm is **LRU** with **global scope**

	A	A	B	A	B	B	A	A
L.Add	100	4000	100	7000	2100	1028	5800	100
Page	0	3	0	6	2	1	5	0
offset	100	928	100	856	52	4	680	100

frame	t = 10	11	12	13	14	15	16	17	18
0	**free**	**A0, 11**	**A0, 11**	**A0, 11**	**A0, 11**	**A0, 11**	**A0, 11**	**A5, 17**	**A5, 17**
1	**B3, 2**	**B3, 2**	**B3, 2**	**B3, 2**	**A6, 14**	**A6, 14**	**A6, 14**	**A6, 14**	**A6, 14**
2	**A2, 10**	**A2, 10**	**A2, 10**	**A2, 10**	**A2, 10**	**A2, 10**	**B1, 16**	**B1, 16**	**B1, 16**
3	**free**	**free**	**free**	**B0, 13**	**B0, 13**	**B0, 13**	**B0, 13**	**B0, 13**	**B0, 13**
4	**A5, 5**	**A5, 5**	**A5, 5**	**A5, 5**	**A5, 5**	**B2, 15**	**B2, 15**	**B2, 15**	**B2, 15**
5	**A3, 3**	**A3, 3**	**A3, 12**	**A3, 12**	**A3, 12**	**A3, 12**	**A3, 12**	**A3, 12**	**A0, 18**

Exercise 4. Replacement scope

On a virtual memory system, with 1024 byte page size, the OS has allocated 6 frames (from 0 to 5) to two processes A y B. At time t = 10, A and B page tables have the following content:

A and B have 8 pages
1 page 1024 bytes = 2^10
Size 8 * 1024 = 8192
Number of bits: 2^13 = 8192
Logical address: 3 + 10 bits

Process A page table

	Frame	Valid bit	Counter
0		i	
1		i	
2	2	v	10
3	5	v	3
4		i	
5	4	v	5
6		i	
7		i	

Process B page table

	Frame	Valid bit	Counter
0		i	
1		i	
2		i	
3	1	v	2
4		i	
5		i	
6		i	
7		i	

a) The replacement algorithm is **LRU** with **global local**

	A	A	B	A	B	B	A	A
L.Add	100	4000	100	7000	2100	1028	5800	100
Page	0	3	0	6	2	1	5	0
offset	100	928	100	856	52	4	680	100

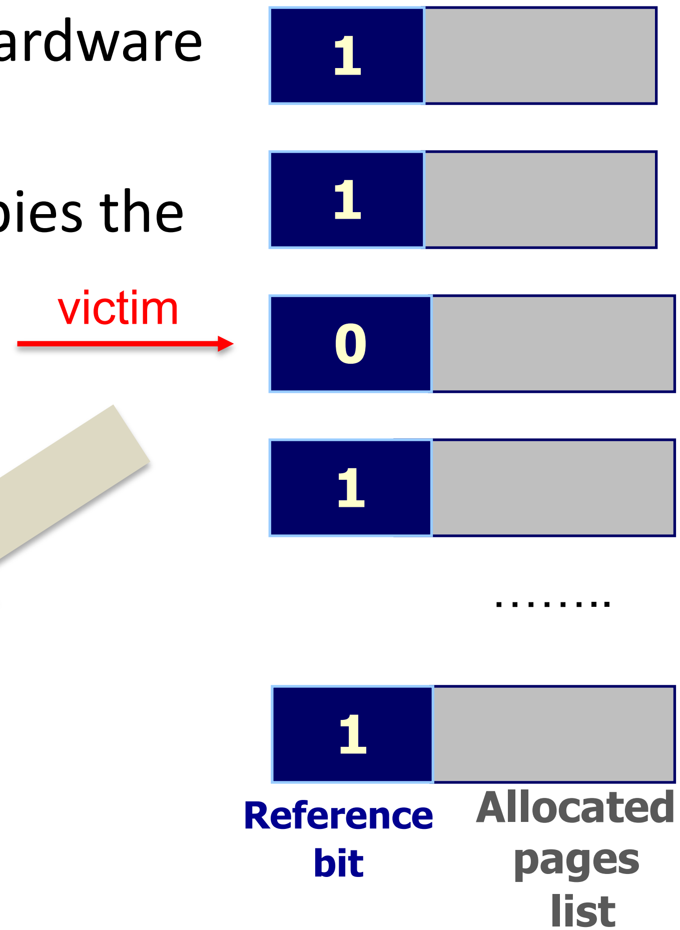
frame	t = 10	11	12	13	14	15	16	17	18
0	**free**	**A0, 11**	**A0, 11**	**A0, 11**	**A0, 11**	**A0, 11**	**A0, 11**	**A0, 11**	**A0, 18**
1	**B3, 2**	**B3, 2**	**B3, 2**	**B3, 2**	**B3, 2**	**B2, 15**	**B2, 15**	**B2, 15**	**B2, 15**
2	**A2, 10**	**A2, 10**	**A2, 10**	**A2, 10**	**A2, 10**	**A2, 10**	**A2, 10**	**A5, 17**	**A5, 17**
3	**free**	**free**	**free**	**B0, 13**	**B0, 13**	**B0, 13**	**B1, 16**	**B1, 16**	**B1, 16**
4	**A5, 5**	**A5, 5**	**A5, 5**	**A5, 5**	**A6, 14**	**A6, 14**	**A6, 14**	**A6, 14**	**A6, 14**
5	**A3, 3**	**A3, 3**	**A3, 12**	**A3, 12**	**A3, 12**	**A3, 12**	**A3, 12**	**A3, 12**	**A3, 12**

Content

- LRU review: problem
- **2nd chance replacement algorithm**
- Frame allocation
- Thrashing
- Frame reservation

LRU approximation: reference bit

- Supporting LRU as replacement algorithm is too expensive
- **Solution**: To approximate LRU using the reference bit
- **Reference bit** can be set by the hardware every time a page is referenced.
- **Scheme**: Every 100ms, the OS copies the reference bits and set to 0.



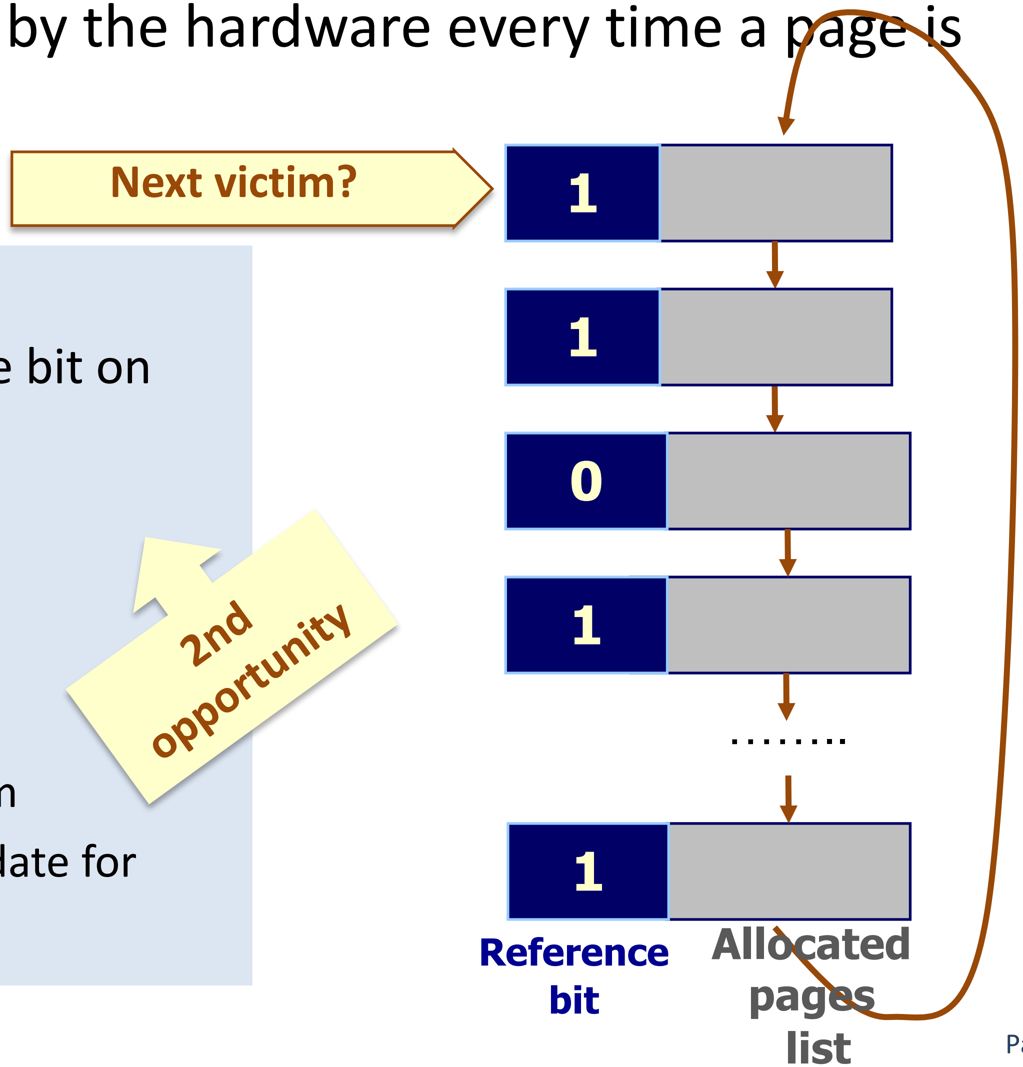
1	0	0	0	0
1	0	1	0	1
1	0	0	1	0
0	1	1	0	0
1	0	1	1	0
1	0	0	1	0
1	1	0	1	1
1	0	1	0	1
1	0	1	0	1
0	1	1	0	0
1	0	0	1	1

2nd chance replacement algorithm

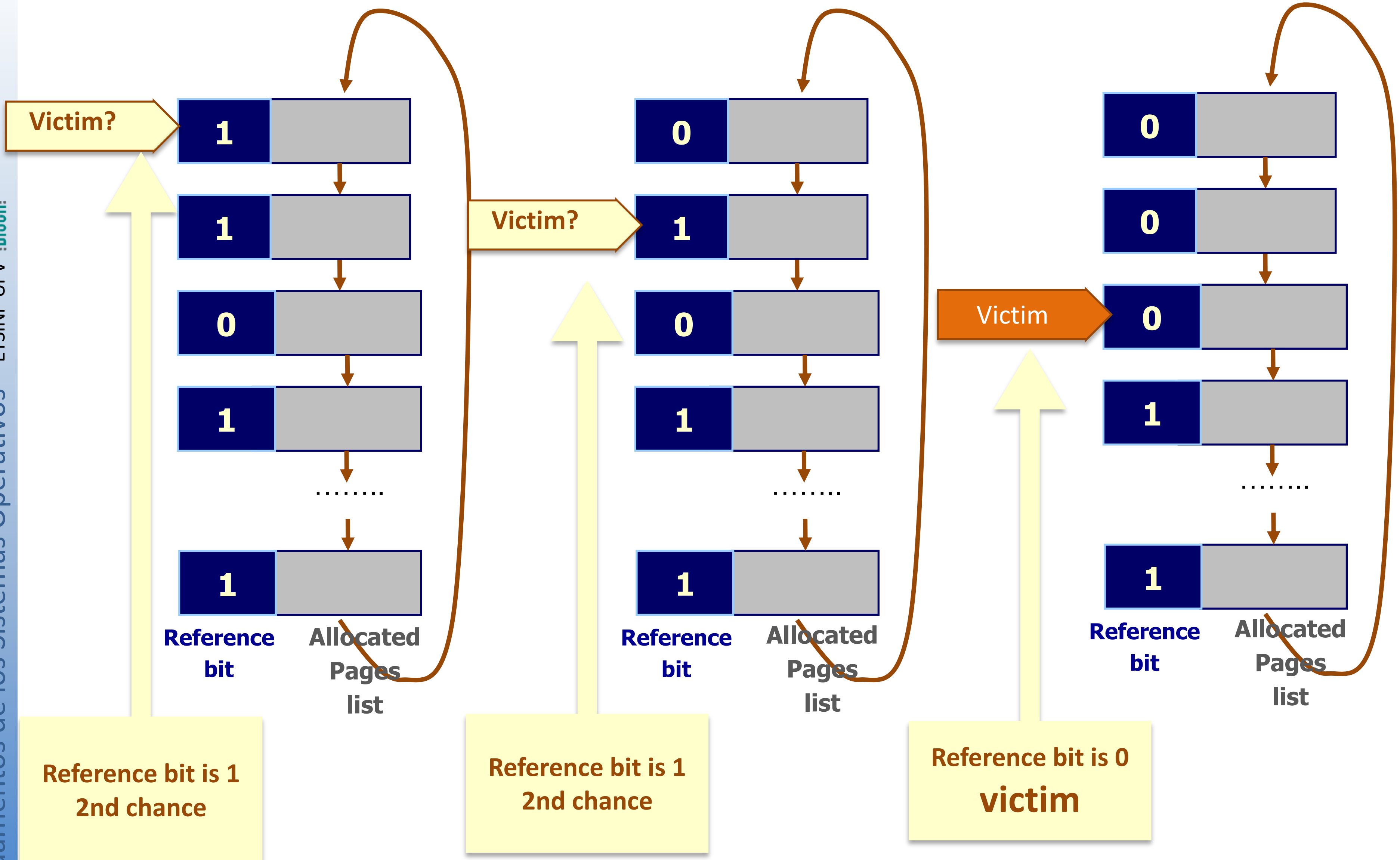
- Supporting LRU as replacement algorithm is too expensive
- **Solution**: To approximate LRU using the reference bit
- Reference bit can be set by the hardware every time a page is referenced.

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 chance replacement algorithm

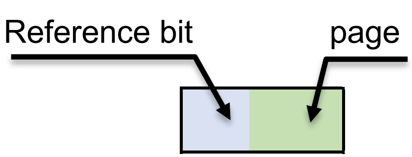


2nd chance replacement algorithm

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						
Frame 1		2	2	2	2						
Frame 2			3	3	3						
Frame 3				4	4						

1	1	1	1	1	1	1	1	1	1
0		1	2	1	2	1	2	1	2
0		0		1	3	1	3	1	3
0		0		0		1	4	1	4
Time 0		1		2		3		4	

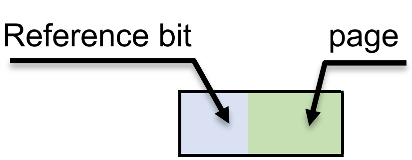


2nd chance replacement algorithm

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	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

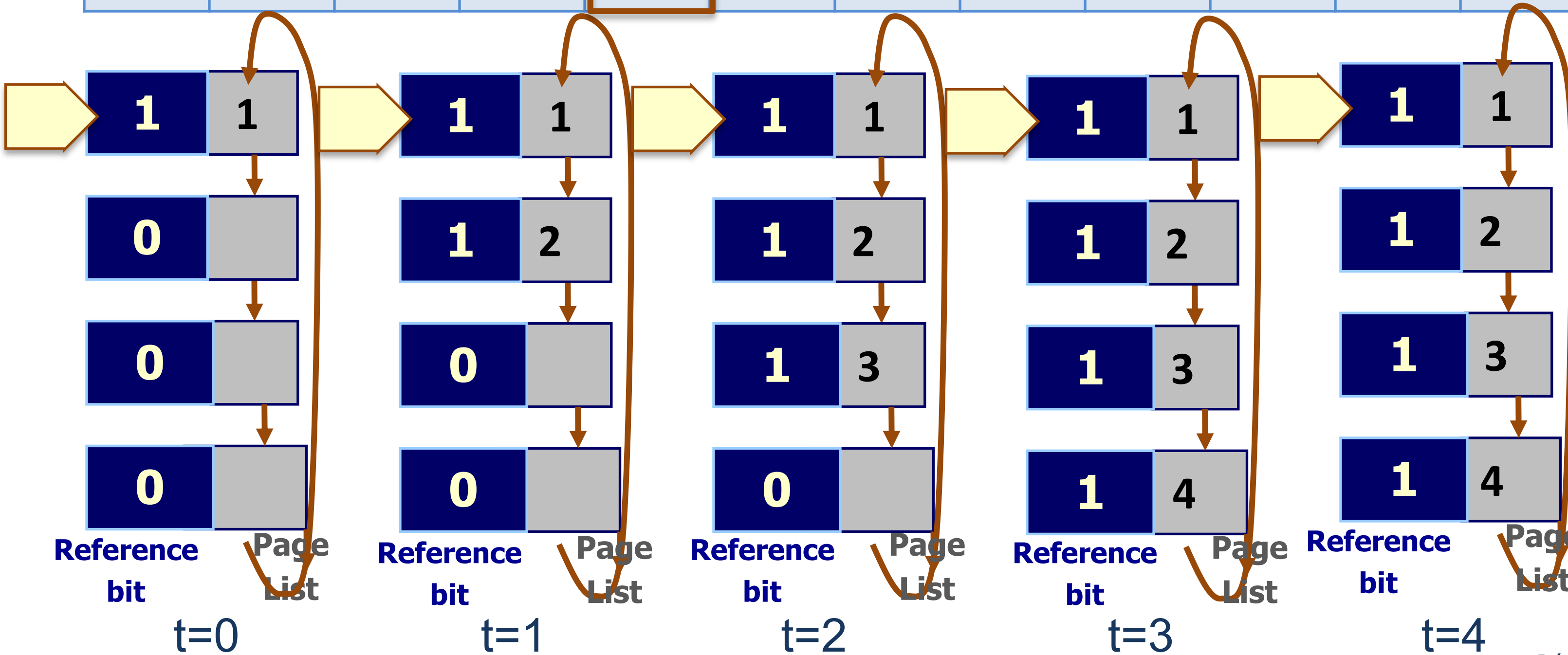
	1	1	1	1	1	1	1	1	1	1
	0		1	2	1	2	1	2	1	2
	0		0		1	3	1	3	1	3
	0		0		0		1	4	1	4
Time	0		1		2		3		4	



2nd chance replacement algorithm

Example: Main memory with 4 frames

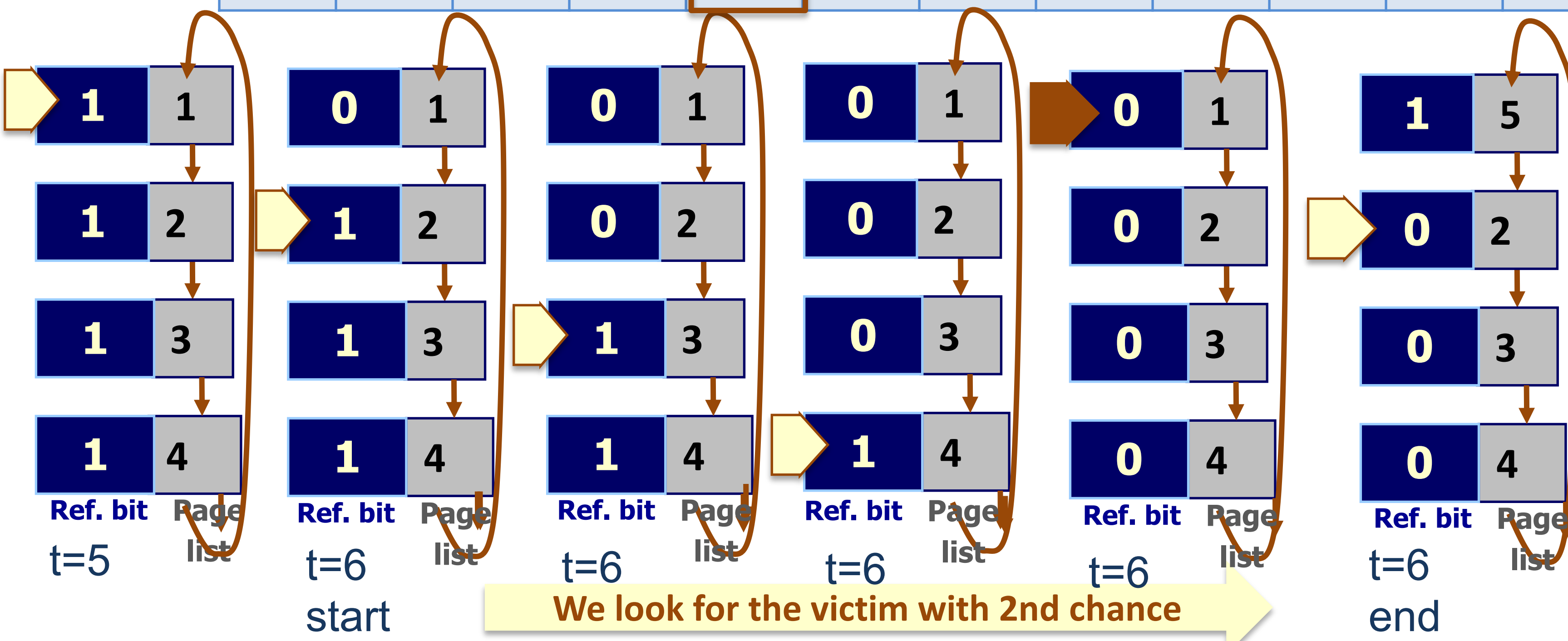
T	0	1	2	3	4	5	6	7	8	9	10
Reference string	1	2	3	4	1	2	5	2	3	1	2
Frame 0	1	1	1	1	1						
Frame 1		2	2	2	2						
Frame 2			3	3	3						
Frame 3				4	4						



2nd chance replacement algorithm

Example: Main memory with 4 frames

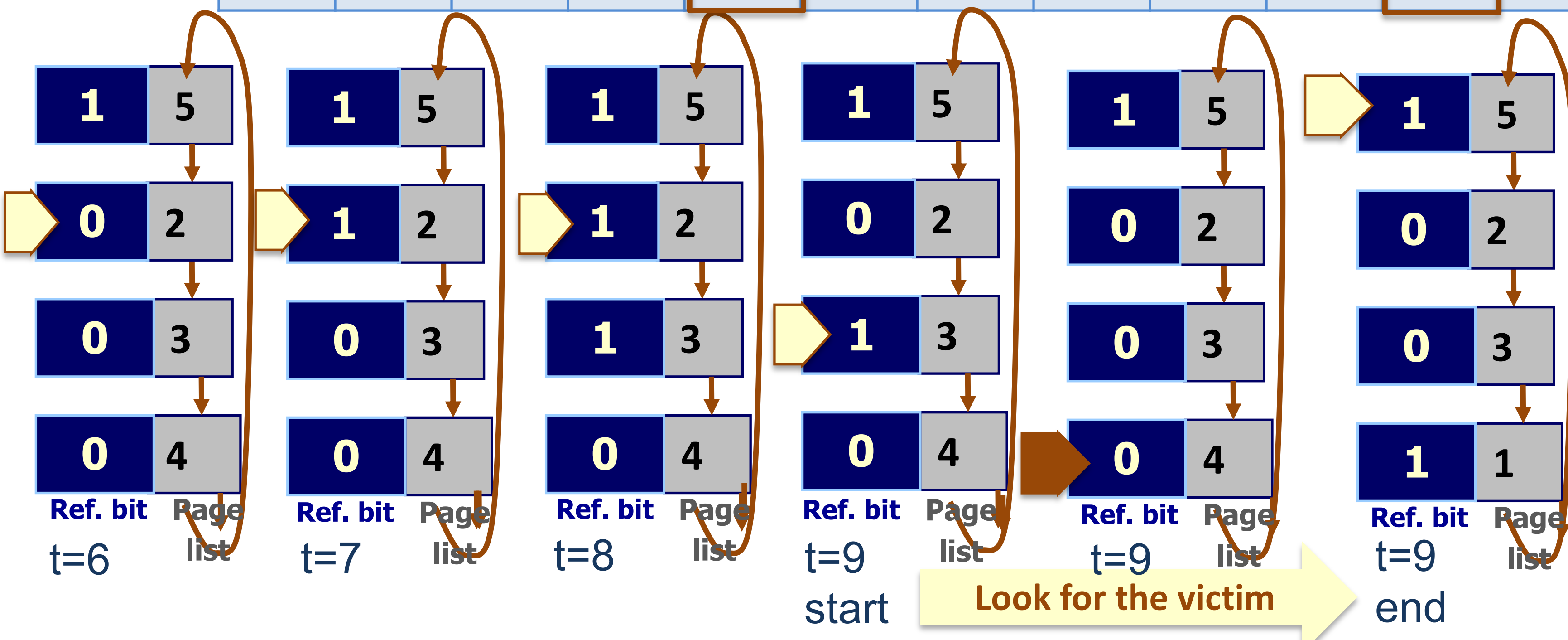
T	0	1	2	3	4	5	6	7	8	9	10
Reference string	1	2	3	4	1	2	5	2	3	1	2
Frame 0	1	1	1	1	1	1	5				
Frame 1		2	2	2	2	2	2				
Frame 2			3	3	3	3	3				
Frame 3				4	4	4	4				



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	



Counting based replacement

There exist several algorithms

- **The least frequently used** (LFU) page-replacement algorithm requires that the page with **the smallest count be replaced**. The reason for this selection is that an actively used page should have a large reference count. A problem arises, however, when a page is used heavily during the initial phase of a process but then is never used again.

- **The most frequently used** (MFU) page-replacement algorithm is based on the argument that the page with the smallest count was probably just brought in and has yet to be used.

Content

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

Frame allocation

Question:

- How do we allocate the fixed amount of free memory among the various processes?
- If we have 38 free frames and 2 processes, how many frames does each process get?

- Simple case: 128 frames
  - OS: 35
  - User processes: 93

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 initially 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

What the hell is this?

Frame allocation

- Consider a machine in which all memory-reference instructions may reference two memory addresses.
- In this case, we need at least **one frame for the instruction** and **two frames for the memory references.**
  - **ADD R1, A, B R1 <- M[A] + M[B]**
- Additionally, if **one-level indirect addressing is allowed** (for example, a load instruction on frame 16 can refer to an address on other frame, which is an indirect reference to another), then **paging requires at least five frames** per process.
- Depending on the instruction size, an instruction could be allocated in the limit of the page and the next one
- **6 pages could be required** (minimum number of pages).

Frame allocation: frame distribution

- Frame distribution policies:

  (Considering **m frames** and **n processes**)

  - **Fair allocation**: All processes allocate $\mathbf{A_i}$ frames equally

    $$Ai = \left\lfloor \frac{m}{n} \right\rfloor$$ **Remainder to a frame free pool**

  - **Proportional allocation**: A process P_i with size S_i allocate $\mathbf{A_i}$ frames computed as:

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

  - **Priority allocation**: higher priority processes allocate more frames. If a high priority process generates a page fault, the victim can be one of its frames or a frame of a process with lower priority.

Frame allocation

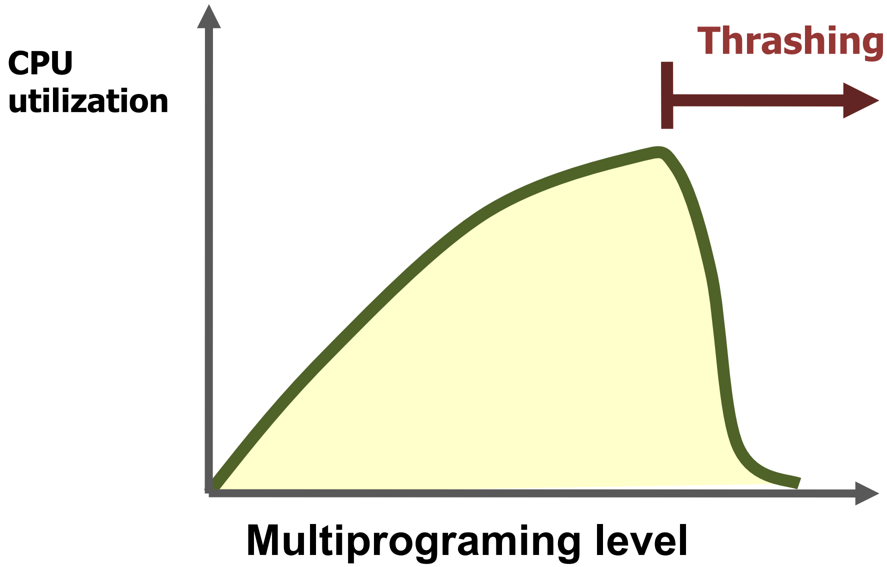
- **Replacement policy scope**
  - **Local replacement**: only pages allocated to the process that generates the page fault can be replaced
    - It cannot choose a victim from another process
    - The number of process allocated frames does not change
    - A process execution is not affected by the 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

Content

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

Thrashing (Hiperpaginación)

- 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 and to prevent the problem
- – Using a working set model
- – Controlling page fault rate

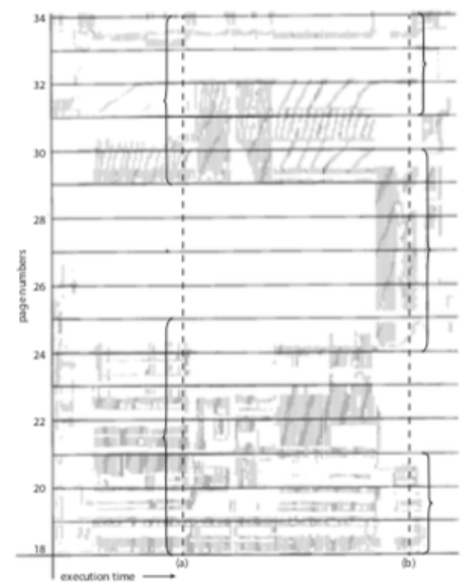
–Once thrashing is detected
- – Swap out processes with a medium term scheduler

A process is thrashing if it is spending more time paging than executing => performance problems

Thrashing (Hiperpaginación)

- 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: logical addresses last referenced
    - **Working set window (WS)**:
      - Fixed consecutive number of references Δ used to compute WS
      - WS is built with the set of pages accessed in the last Δ references
    - **Working set size (WSS)**: Number of different pages that belong to the WS. WSSi => number of pages of the WS of the process Pi
    - 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 $\Delta = 9$
  - 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

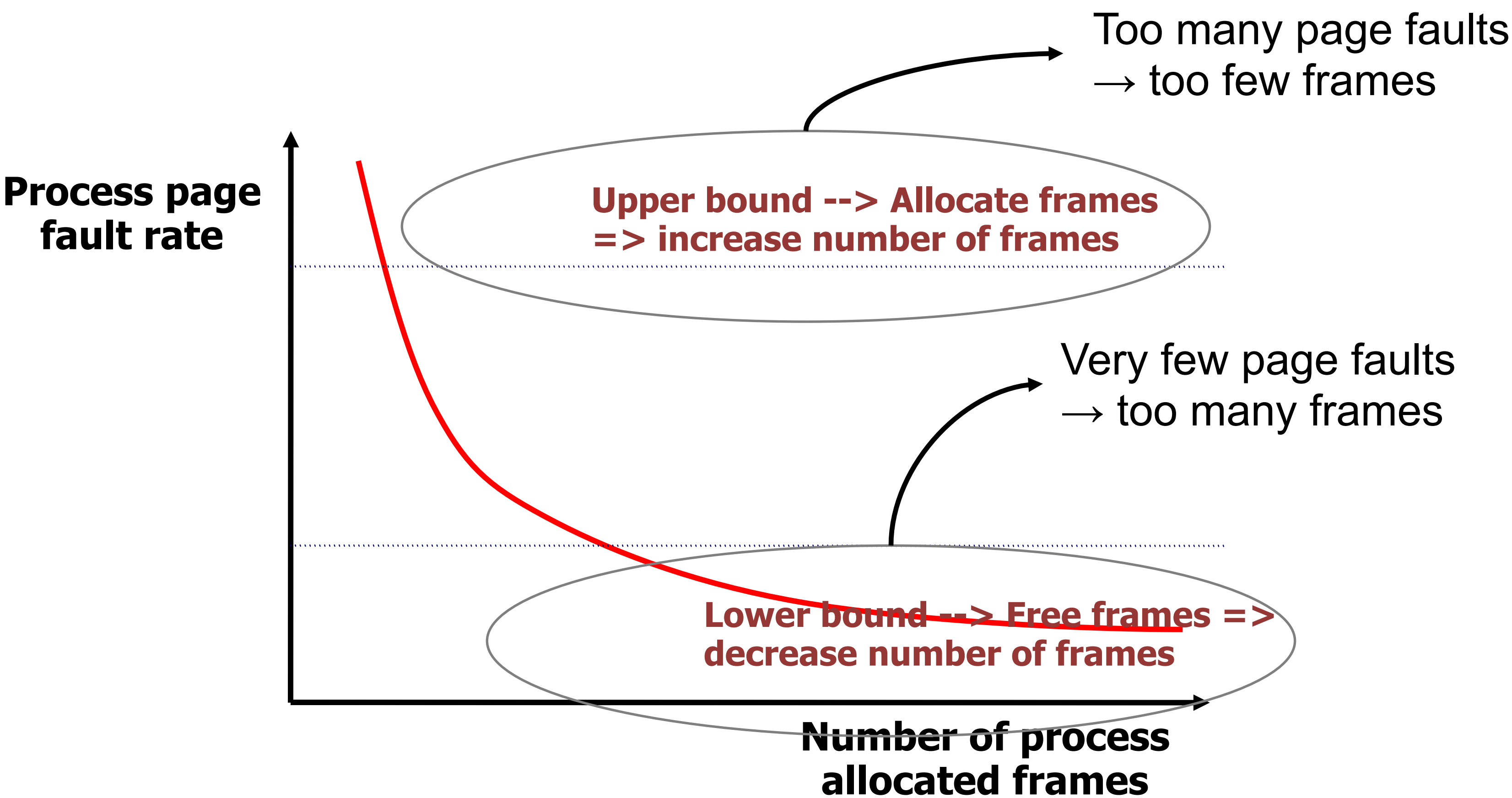
Δ=9 (up to t1) Δ=9 (from t1 to t2)

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 OSs 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

Conclusions

- Virtual memory abstracts physical memory into an extremely large uniform array of storage.
- The benefits of virtual memory include the following:
  - a program can be larger than physical memory,
  - a program does not need to be entirely in memory,
  - processes can share memory,
  - processes can be created more efficiently.
- Demand paging is a technique where by pages are loaded only when they are demanded during program execution.
- A page fault occurs when a page that is currently not in memory is accessed.
- Several algorithms have been analysed: Optima, LRU, aproximmations to LRU, etc.
- Thrashing occurs when a system spends more time paging than executing.