Part III
Storage Management
Chapter 9: Virtual Memory

Observations

- The entire program does not have to be in memory, because
  - error handling codes are not frequently used
  - arrays, tables, large data structures are allocated memory more than necessary and many parts are not used at the same time
  - some options and cases may be used rarely
- If they are not needed, why must they be in memory?

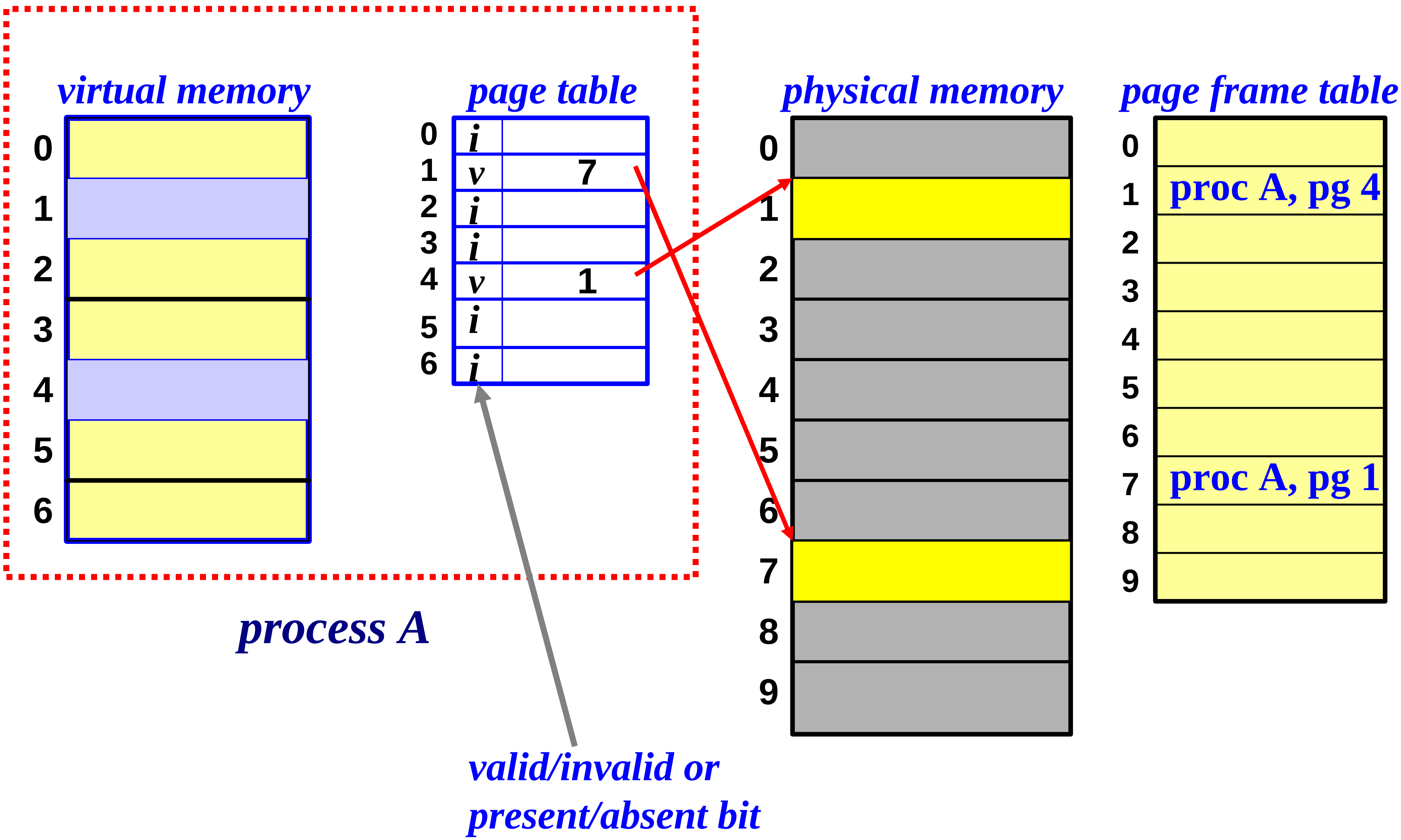
Benefits

- Program length is not restricted to real memory size. That is, virtual address size can be larger than physical address size.
- Can run more programs because those space originally allocated for the un-loaded parts can be used by other programs.
- Save load/swap I/O time because we do not have to load/swap a complete program.

Virtual Memory

- Virtual memory is the separation of user logical memory from physical memory.
- This permits to have extremely large virtual memory, which makes programming large systems much easier to use.
- Because pages can be shared, this further improves performance and save time.
- Virtual memory is commonly implemented with demand paging, demand segmentation or demand paging+segmentation.

Demand Paging



Address Translation

- The translation from a virtual address to a physical address is the same as what a paging system does.
- However, there is an additional check. If the page is not in physical memory (*i.e.*, the valid bit is not set), a page fault (*i.e.*, a trap) occurs.
- If a page fault occurs, we need to do the following:
  - Find an unused page frame. If no such page frame exists, a victim must be found and evicted.
  - Write the old page out and load the new page in.
  - Update both page tables.
  - Resume the interrupted instruction.

Details of Handling a Page Fault

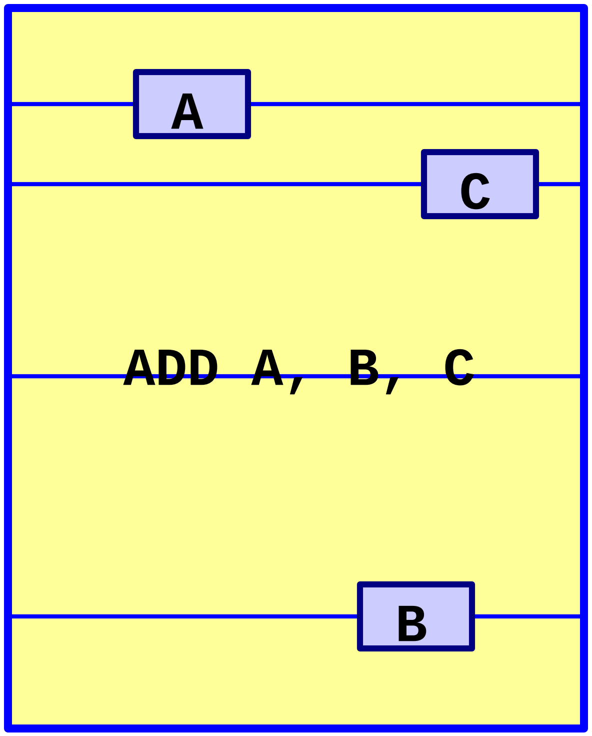
Trap to the OS // a context switch occurs
Make sure it is a page fault;
If the address is not a legal one then
    address error, return
Find an unused page frame // page replacement algorithm
Write the page back to disk // page out
Load the new page from disk // page in
Update both page tables // two pages are involved!
Resume the execution of the interrupted instruction

Hardware Support

- **Page Table Base Register**, **Page Table Length Register**, **and a** **Page Table.**
- **Each entry of the page table must have a valid/invalid bit.** ***Valid*** **means that that page is in physical memory. The address translation hardware must recognize this bit and generate a page fault if the valid bit is not set.**
- **Secondary Memory: use a disk.**
- **Other hardware components may be needed and will be discussed later in this chapter.**

Too Many Memory Accesses?!

- Each address reference may cause **at least two** memory accesses, one for page table look up and the other for accessing the item. It may be worse! **See below:**



How many memory accesses are there?
May be more than eight!

Performance Issue: 1/2

- Let *p* be the probability of a page fault, the page fault rate, 0 £ *p* £ 1.
- The effective access time is

  (1-*p*)*memory access time + *p**page fault time
- The page fault rate *p* should be small, and memory access time is usually between 10 and 200 nanoseconds.
- To complete a page fault, three components are important:
  - Serve the page-fault trap
  - Read in the page, *a bottleneck*
  - Restart the process

Performance Issue: 2/2

- Suppose memory access time is 100 nanoseconds, paging requires 25 milliseconds (software and hardware). Then, effective access time is

  $(1-p)*100 + p*(25 \text{ milliseconds})$

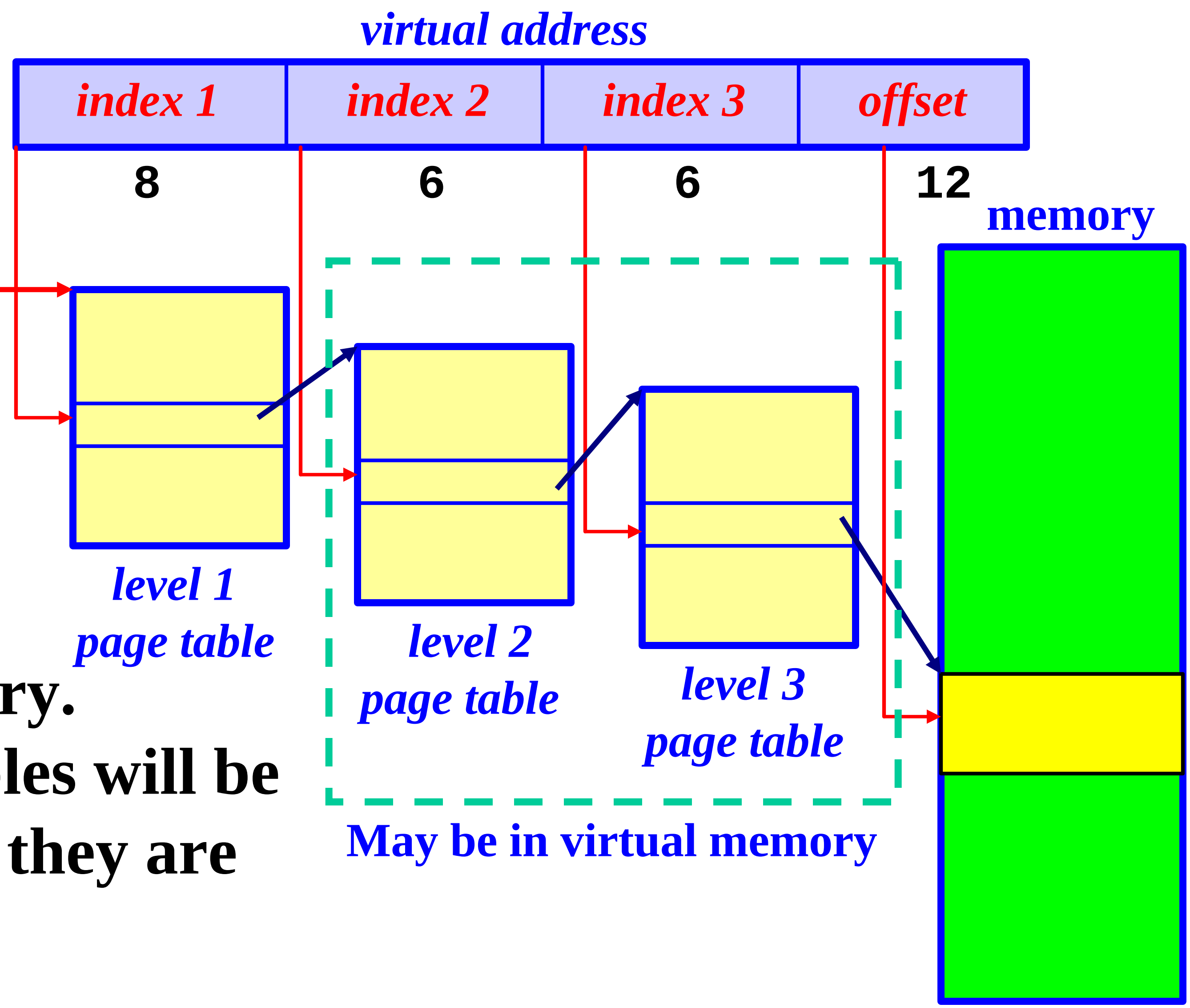
  $= (1-p)*100 + p*25{,}000{,}000$ nanoseconds

  $= 100 + 24{,}999{,}900*p$ nanoseconds
- If the page fault rate is 1/1000, the effective access time is 25,099 nanoseconds = 25 microseconds. It is 250 times slower!
- If we wish it is only 10% slower, effective access time is no more than 110 and p=0.0000004.

Three Important Issues in V.M.

- **A page table can be very large.** If an address has 32 bits and page size is 4K, then there are $2^{32}/2^{12}=2^{20}=(2^{10})^2=$ 1M entries in a page table per process!
- **Virtual to physical address translation must be fast.** This is done with TLB.
- **Page replacement.** When a page fault occurs and there is no free page frame, a victim page must be selected. If the victim is not selected properly, system degradation may be high.

Page Table Size



Only level 1 page tables will reside in physical memory. Other page tables will be paged-in when they are referred to.

Page Replacement: 1/2

- The following is a basic scheme
  - Find the desired page on the disk
  - Find a free page frame in physical memory
    - If there is a free page frame, use it
    - If there is no free page frame, use a page-replacement algorithm to find a victim page
    - Write this victim page back to disk and change the page table and page frame table
  - Read the desired page into the frame and update page table and page frame table
  - Restart the interrupted instruction

Page Replacement: 2/2

- If there is no free page frame, two page transfers (*i.e.*, page-in and page-out) are required.
- A **modify bit** may be added to a page table entry. The modify bit is set if that page has been modified (*i.e.*, storing info into it). It is initialized to 0 when a page is brought into memory.
- Thus, if a page is not modified (*i.e.*, modify bit = 0), it does not have to be written back to disk.
- Some systems may also have a **reference bit**. When a page is referenced (*i.e.*, reading or writing), its reference bit is set. It is initialized to 0 when a page is brought in.
- **Both bits are set by hardware automatically.**

Page Replacement Algorithms

- We shall discuss the following page replacement algorithms:
  - First-In-First-Out - FIFO
  - The Least Recently Used – LRU
  - The Optimal Algorithm
  - The Second Chance Algorithm
  - The Clock Algorithm
- The fewer number of page faults an algorithm generates, the better the algorithm it is.
- Page replacement algorithms work on page numbers. A string of such page numbers is referred to as a page reference string.

The FIFO Algorithm

- The FIFO algorithm always selects the "oldest" page to be the victim.

3 frames

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

page fault=9 *miss ratio*=9/12=75% *hit ratio* = 25%

4 frames

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

page fault=10 *miss ratio*=10/12=83.3% *hit ratio* = 16.7%

Belady Anomaly

- **Intuitively, increasing the number of page frames should reduce page faults.**
- **However, some page replacement algorithms do not satisfy this "intuition." The FIFO algorithm is an example.**
- **Belady Anomaly: Page faults increase as the number of page frames increases.**
- **FIFO was used in DEC VAX-78*xx* series because it is easy to implement: append the new page to the tail and select the head to be a victim!**

The LRU Algorithm: 1/2

- The LRU algorithm always selects the page that has **not** been used for the **longest period of time**.

3 frames

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

page fault=10 *miss ratio*=10/12=83.3% *hit ratio* = 16.7%

4 frames

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

page fault=8 *miss ratio*=8/12=66.7% *hit ratio* = 33.3%

The LRU Algorithm: 2/2

- The memory content of 3-frames is a subset of the memory content of 4-frames. This is the inclusion property. With this property, Belady anomaly never occurs.

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

extra

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

The Optimal Algorithm: 1/2

- The optimal algorithm always selects the page that **will not** be used for the **longest period of time**.

3 frames

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

page fault=7 miss ratio=7/12=58.3% hit ratio = 41.7%

4 frames

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

page fault=6 miss ratio=6/12=50% hit ratio = 50%

The Optimal Algorithm: 2/2

- The optimal algorithm always delivers the fewest page faults, if it can be implemented. It also satisfies the inclusion property (*i.e.*, no Belady anomaly).

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

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

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

extra

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

LRU Approximation Algorithms

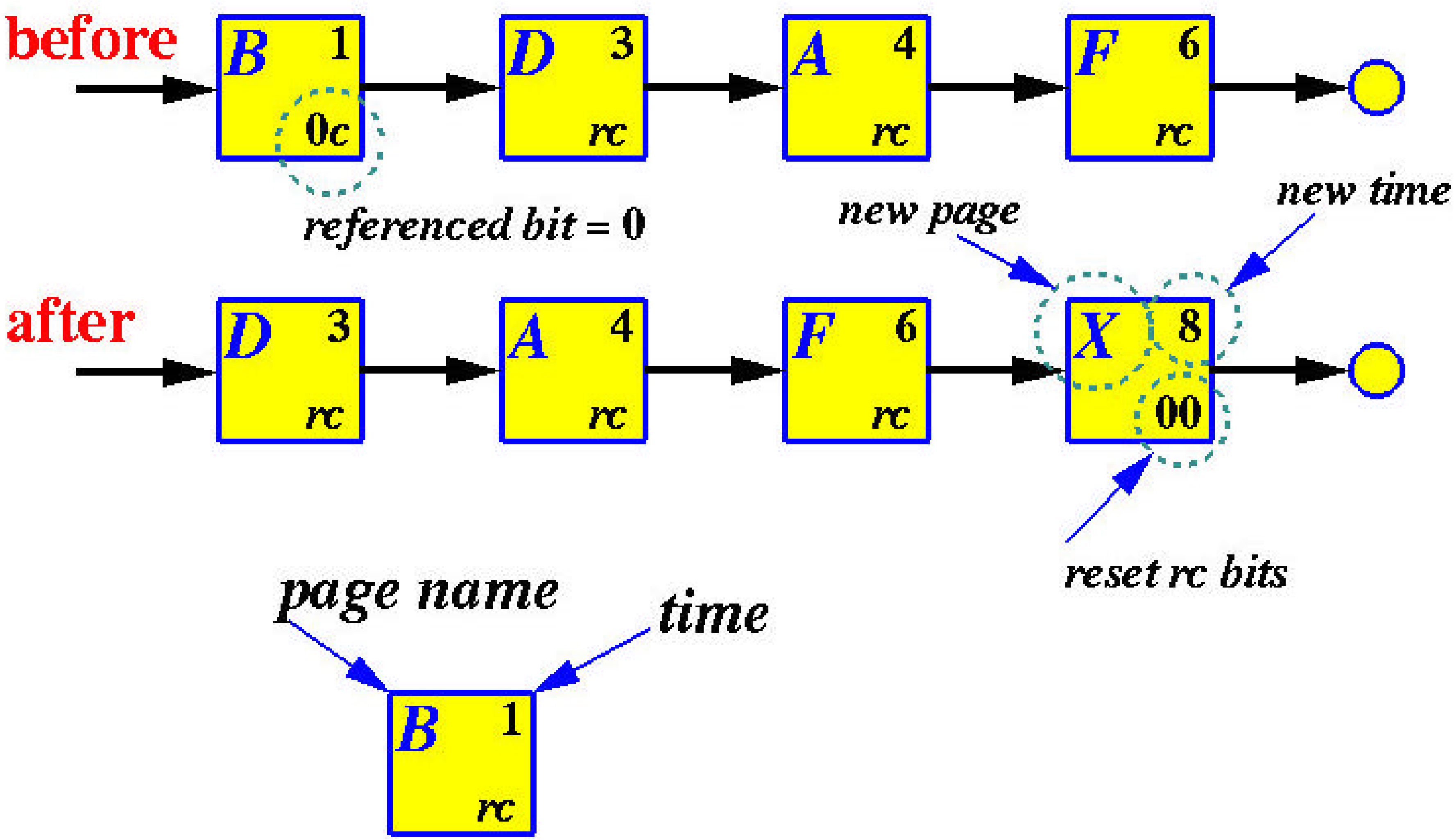
- FIFO has Belady anomaly, the Optimal algorithm requires the knowledge in the future, and the LRU algorithm requires accurate info of the past.
- The optimal and LRU algorithms are difficult to implement, especially the optimal algorithm. Thus, LRU approximation algorithms are needed. We will discuss three:
  - The Second-Chance Algorithm
  - The Clock Algorithm
  - The Enhanced Second-Chance Algorithm

The Second-Chance Algorithm: 1/3

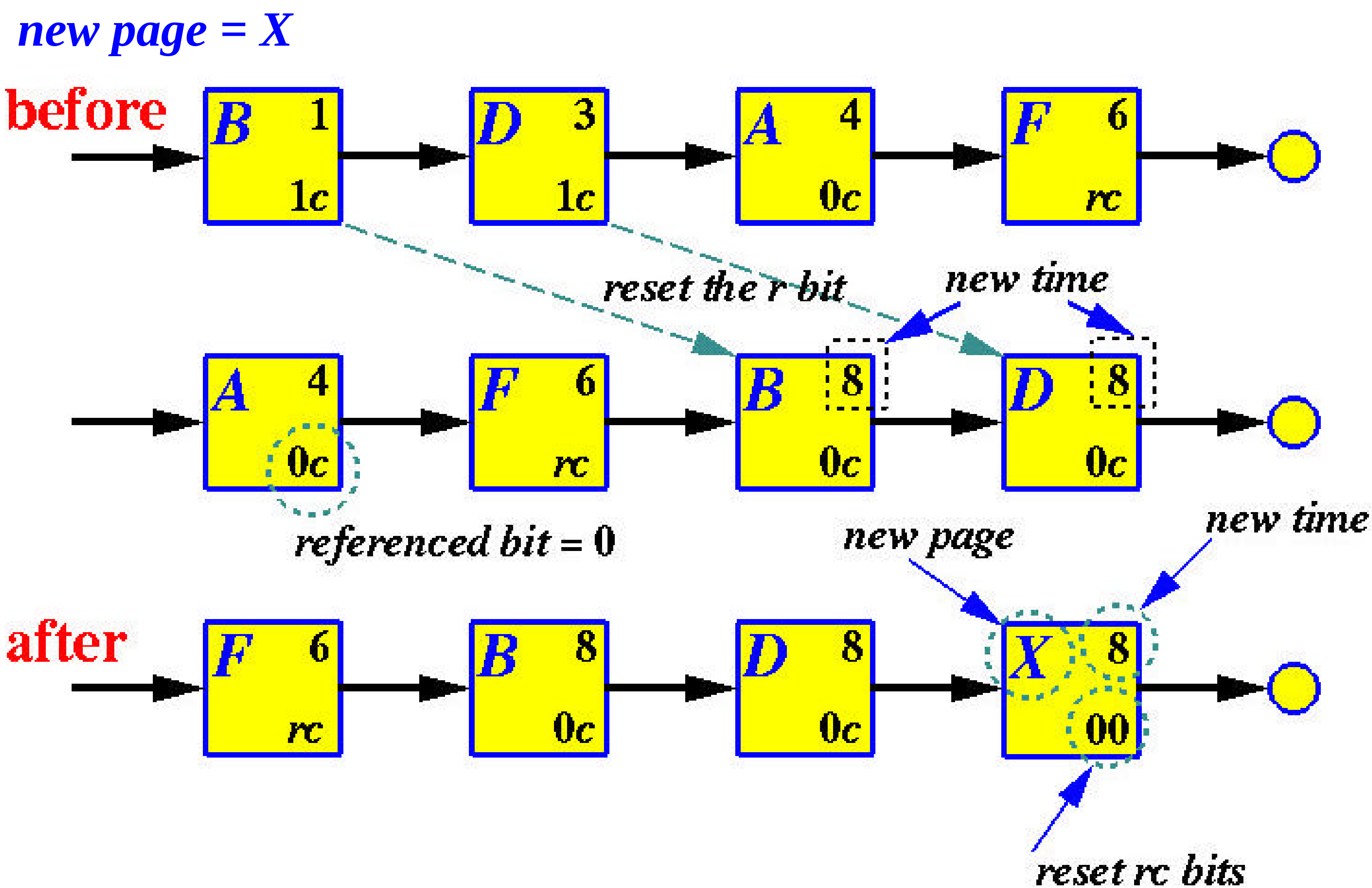
- The second chance algorithm is basically a FIFO algorithm. It uses the reference bit of each page.
- When a page frame is needed, check the oldest one:
  - If its reference bit is 0, take this one
  - Otherwise, clear the reference bit, move it to the tail, and (perhaps) set the current time. This gives it a second chance.
- Repeat this procedure until a 0 reference bit page is found. Do page-out and page-in if necessary, and move it to the tail.
- Problem: Page frames are moved too frequently.

The Second-Chance Algorithm: 2/3

new page = X



The Second-Chance Algorithm: 3/3

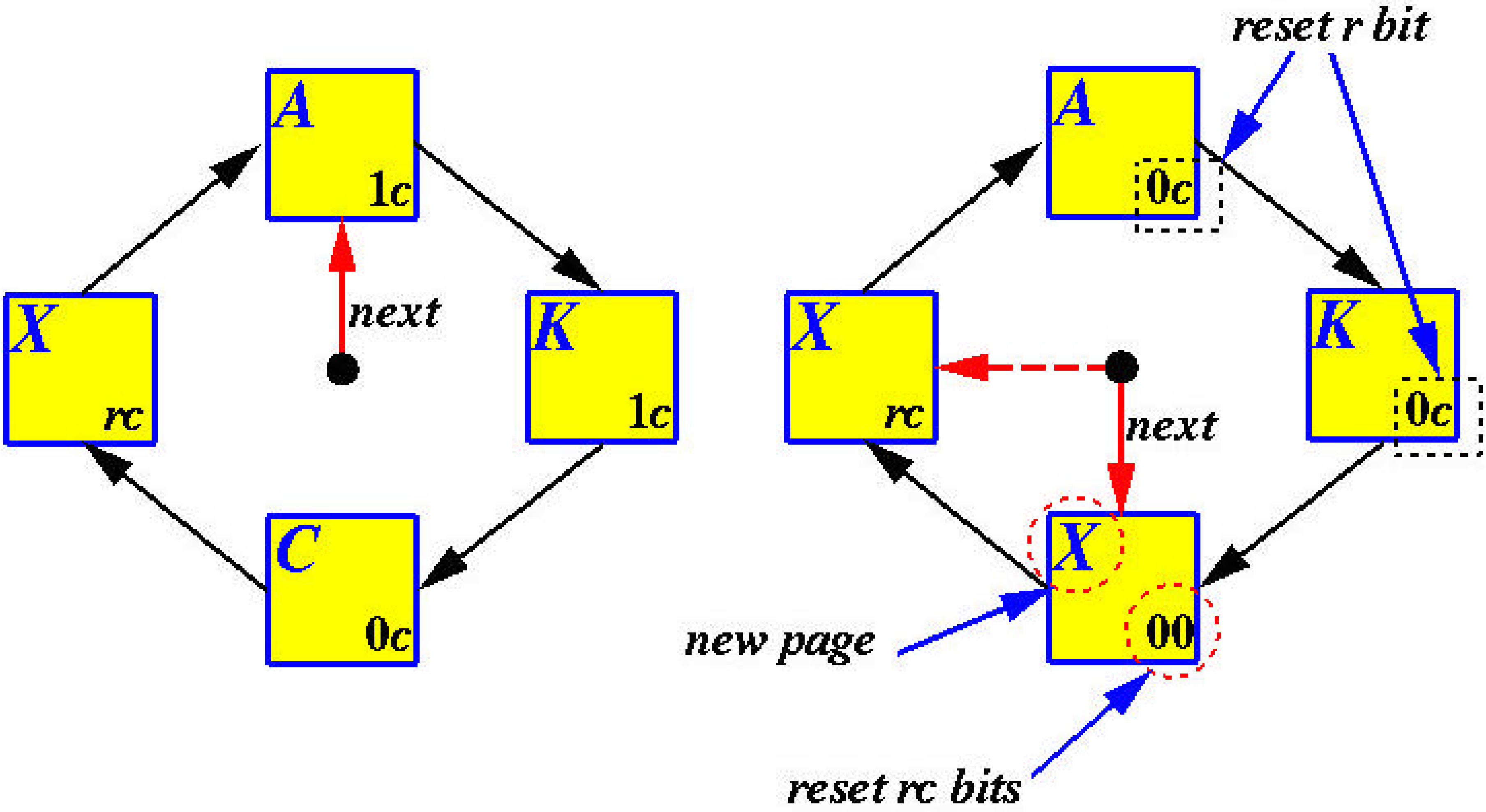


The Clock Algorithm: 1/2

- If the second chance algorithm is implemented with a circular list, we have the clock algorithm.
- We need a "next" pointer.
- When a page frame is needed, we examine the page under the "next" pointer:
  - If its reference bit is 0, take it
  - Otherwise, clear the reference bit and advance the "next" pointer.
- Repeat this until a 0 reference bit frame is found.
- Do page-in and page-out, if necessary

The Clock Algorithm: 2/2

new page = X



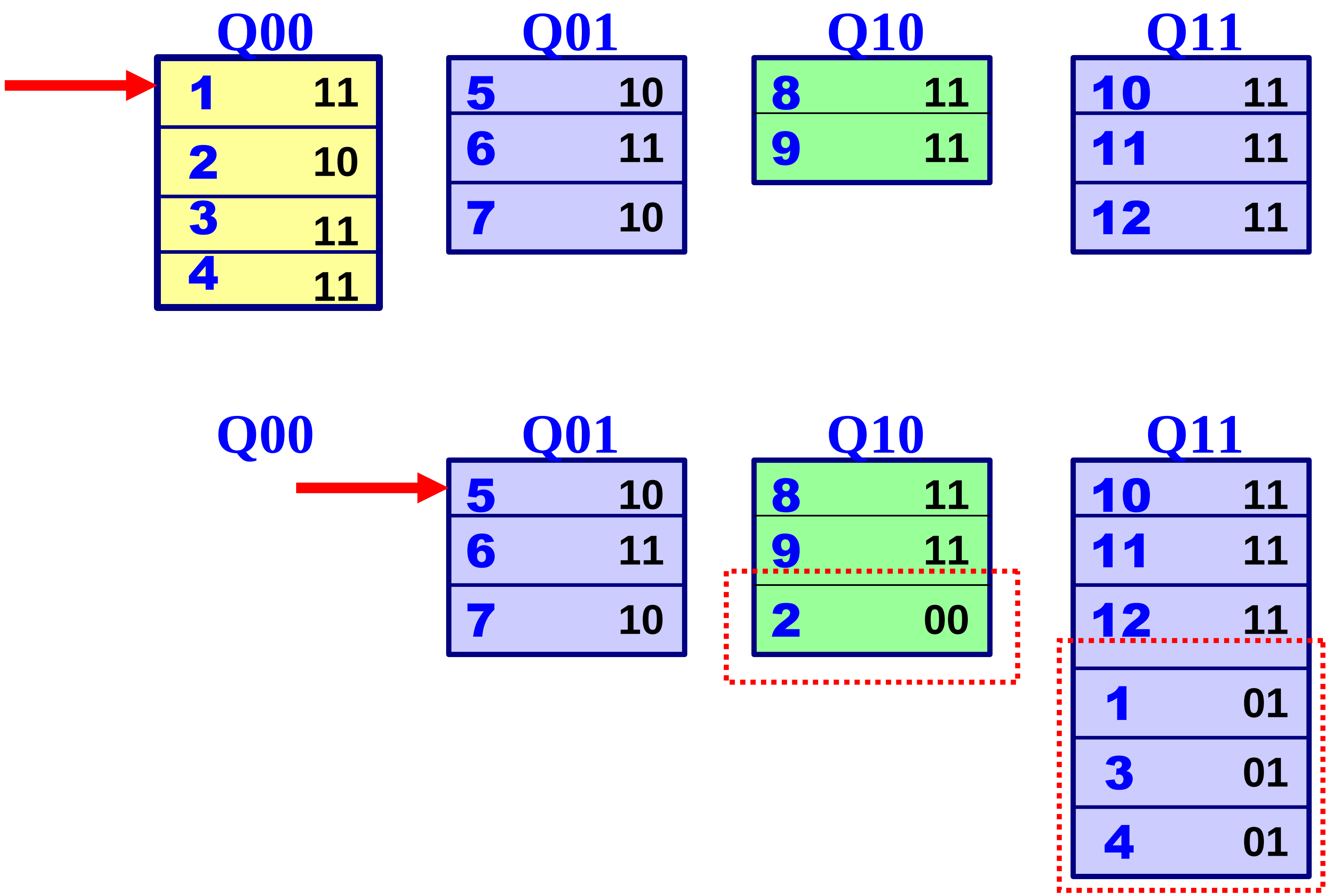
Enhanced Second-Chance Algorithm: 1/5

- We have four page lists based on their reference-modify bits (*r*,*c*):
  - **Q00** - pages were not referenced and not modified. The best candidates!
  - **Q01** - pages were changed but not recently referenced. Need a page-out.
  - **Q10** - pages were recently used but clean.
  - **Q11** - pages were recently used and modified. Need a page-out.

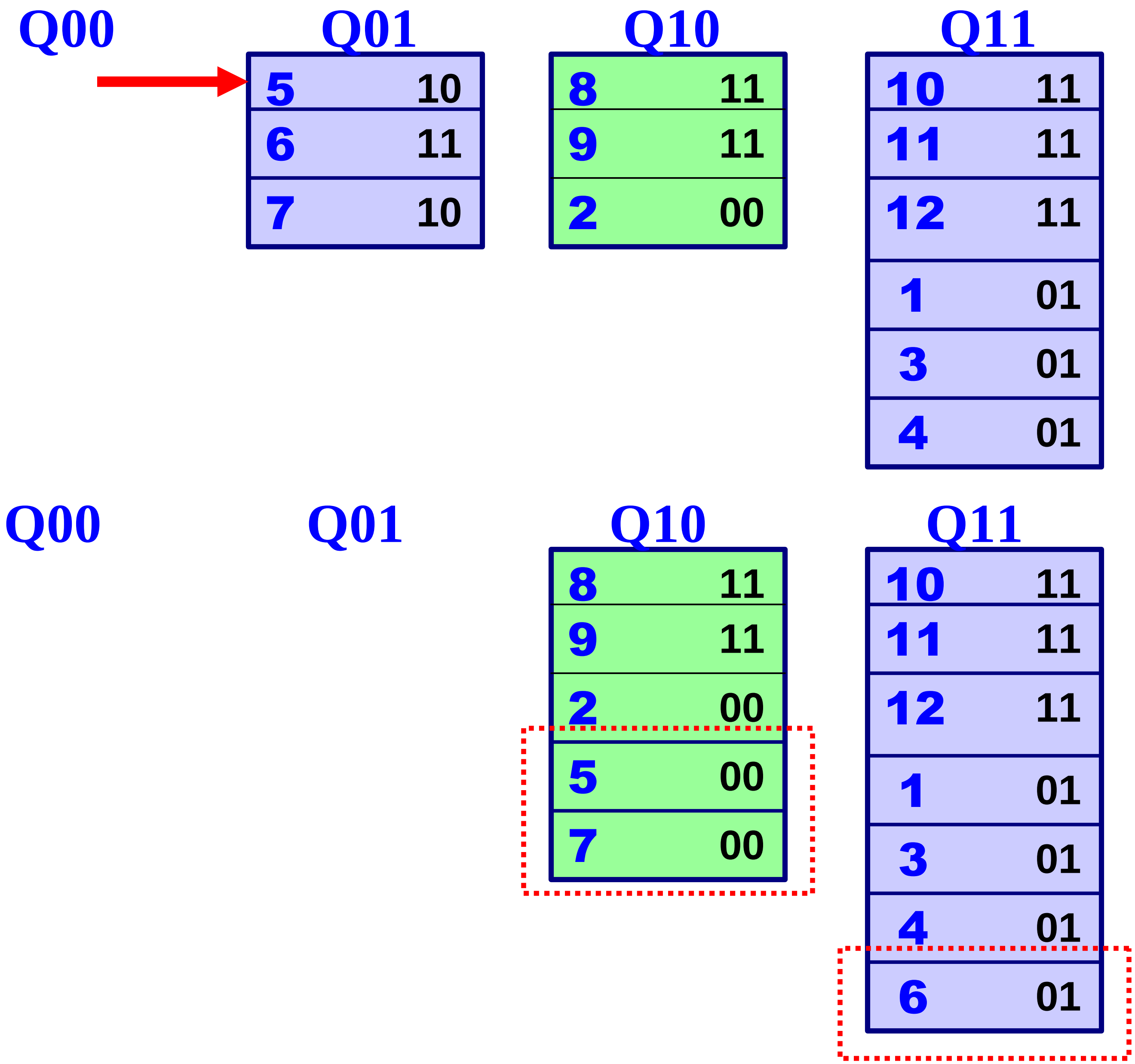
Enhanced Second-Chance Algorithm: 2/5

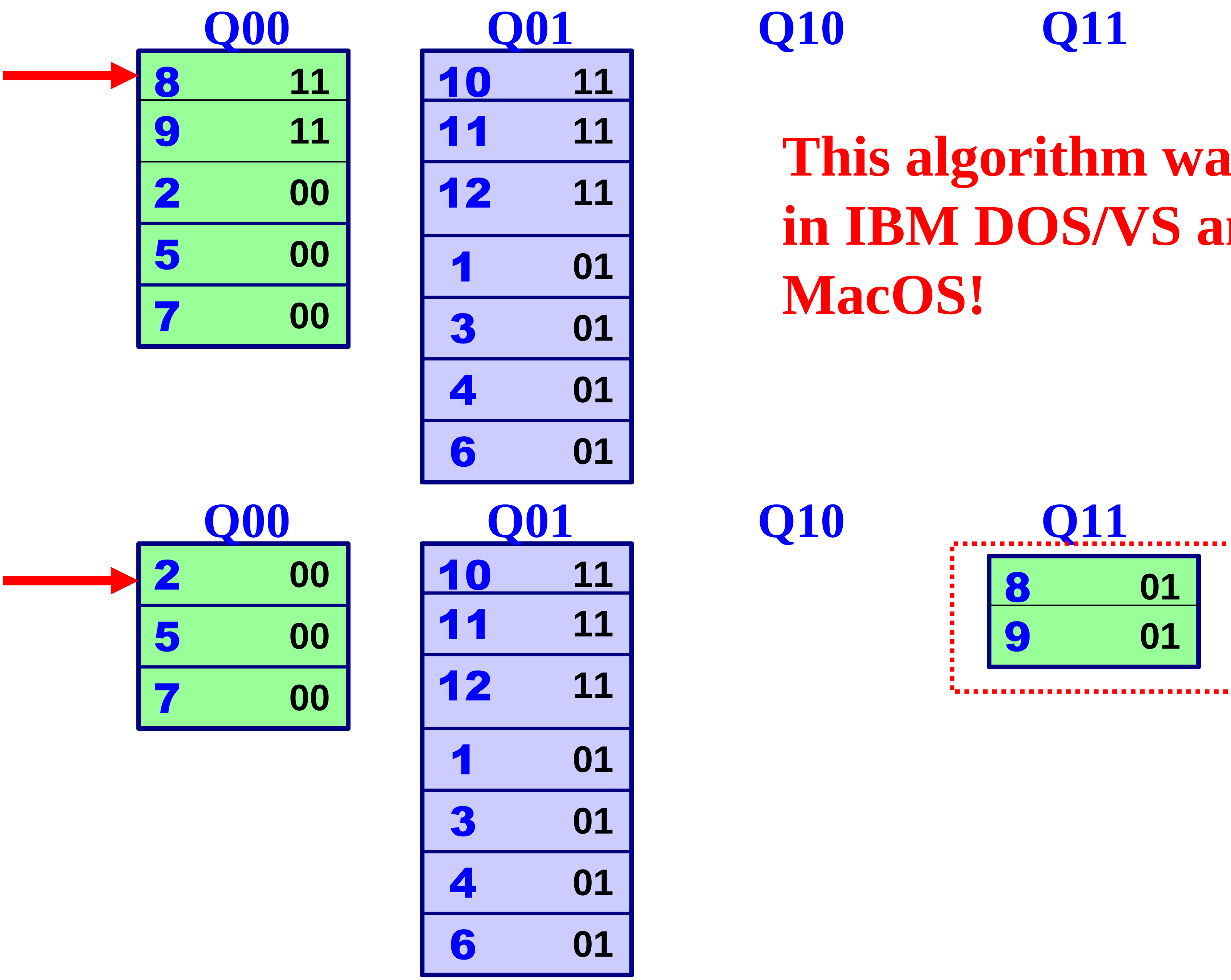
- We still need a "next" pointer.
- When a page frame is needed:
  - Does the "next" frame has 00 combination? If yes, victim is found. Otherwise, reset the reference bit and move this page to the corresponding list.
  - If Q00 becomes empty, check Q01. If there is a frame with 01 combination, it is the victim. Otherwise, reset the reference bit and move the frame to the corresponding list.
  - If Q01 becomes empty, move Q10 to Q00 and Q11 to Q01. Restart the scanning process.

Enhanced Second-Chance Algorithm: 3/5



Enhanced Second-Chance Algorithm: 4/5





This algorithm was used in IBM DOS/VS and MacOS!

Other Important Issues

- Global vs. Local Allocation
- Locality of Reference
- Thrashing
- The Working Set Model
- The Working Set Clock Algorithm
- Page-Fault Frequency Replacement Algorithm

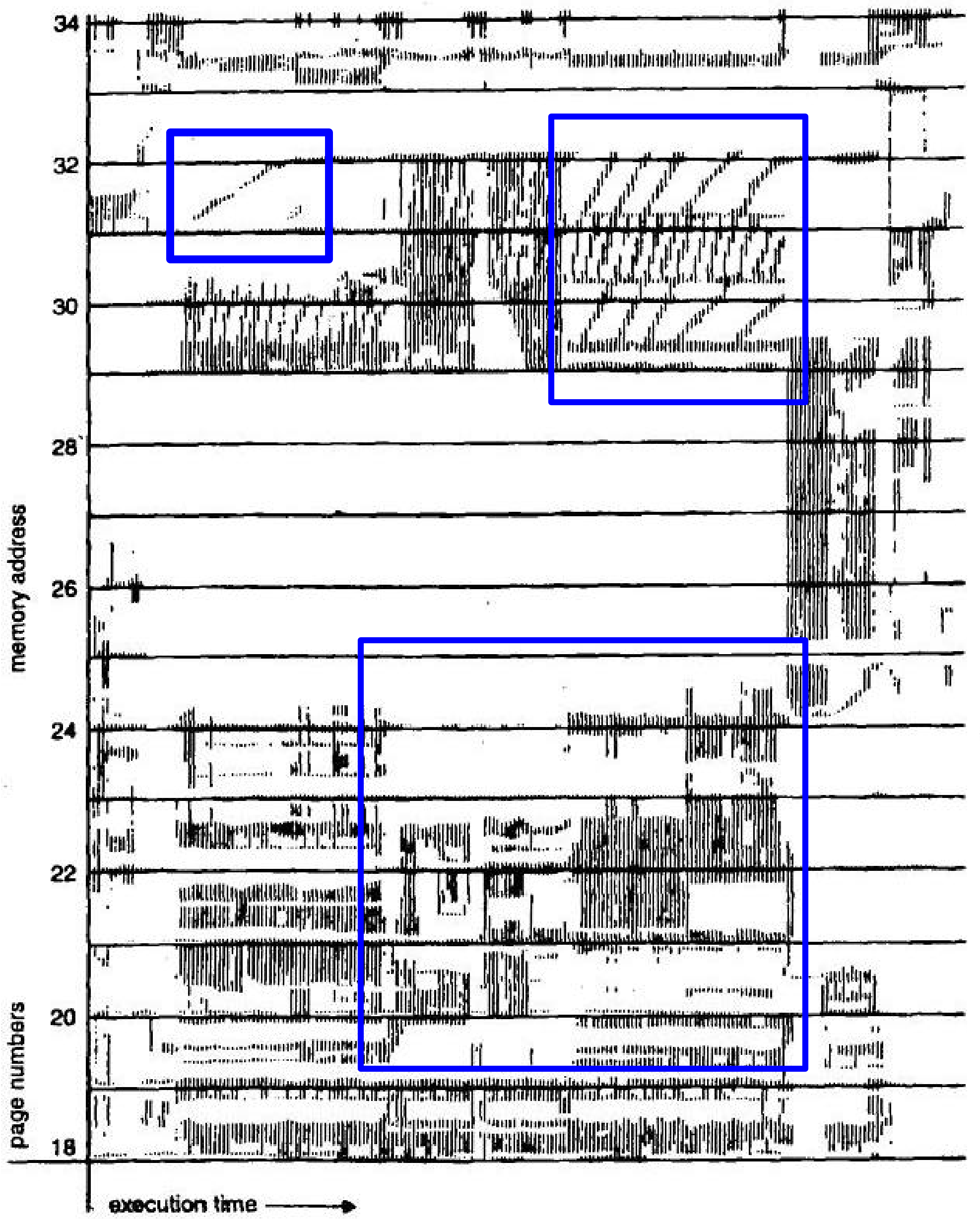
Global vs. Local Replacement

- **Global replacement allows a process to select a victim from the set of all page frames, even if the page frame is currently allocated to another process.**
- **Local replacement requires that each process selects a victim from its own set of allocated frames.**
- **With a global replacement, the number of frames allocated to a process may change over time. With a local replacement algorithm, it is likely a constant.**

Global vs. Local: A Comparison

- With a global replacement algorithm, a process cannot control its own page fault rate, because the behavior of a process depends on the behavior of other processes. The same process running on a different system may have a totally different behavior.
- With a local replacement algorithm, the set of pages of a process in memory is affected by the paging behavior of that process only. A process does not have the opportunity of using other less used frames. Performance may be lower.
- With a global strategy, throughput is usually higher, and is commonly used.

Locality of Reference



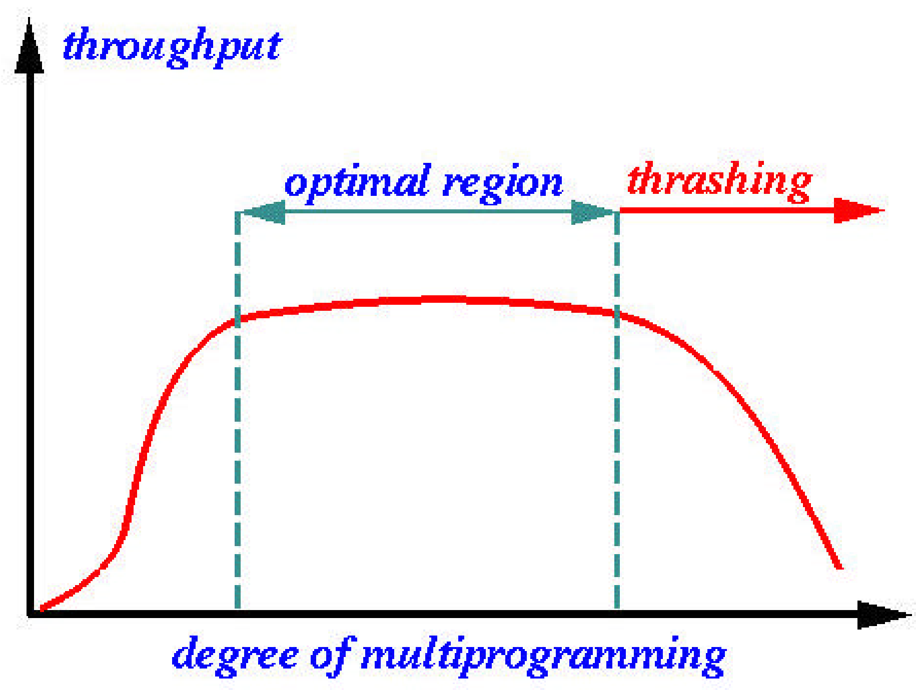
- During any phase of execution, the process references only a relatively small fraction of pages.

Thrashing

- **A high paging activity is called thrashing. This means a process is spending more time paging than executing (*i.e.*, low CPU utilzation).**
- **If CPU utilization is too low, the medium-term scheduler is invoked to swap in one or more swapped-out processes or bring in one or more new jobs. The number of processes in memory is referred to as the degree of multiprogramming.**

Degree of Multiprogramming: 1/3

- **We cannot increase the degree of multiprogramming arbitrarily as throughput will drop at certain point and thrashing occurs.**
- **Therefore, the medium-term scheduler must maintain the optimal degree of multiprogramming.**



Degree of Multiprogramming: 2/3

1. Suppose we use a global strategy and the CPU utilization is low. The medium-term scheduler will add a new process.
2. Suppose this new process requires more pages. It starts to have more page faults and page frames of other processes will be taken by this process.
3. Other processes also need these page frames. Thus, they start to have more page faults.
4. Because pages must be paged- in and out, these processes must wait, and the number of processes in the ready queue drops. CPU utilization is lower.

Degree of Multiprogramming: 3/3

5. Consequently, the medium-term scheduler brings in more processes into memory. These new processes also need page frames to run, causing more page faults.
6. Thus, CPU utilization drops further, causing the medium-term scheduler to bring in even more processes.
7. If this cycle continues, the page fault rate increases dramatically, and the effective memory access time increases. Eventually, the system is paralyzed because the processes are spending almost all time to do paging!

The Working Set Model: 1/4

- The **working set** of a process at virtual time t, written as $W(t,q)$, is the set of pages that were referenced in the interval $(t- q, t]$, where q is the window size.
- q = 3. The result is identical to that of LRU:

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

page fault=10 *miss ratio*=10/12=83.3% *hit ratio* = 16.7%

The Working Set Model: 2/4

- However, the result of q = 4 is different from that of LRU.

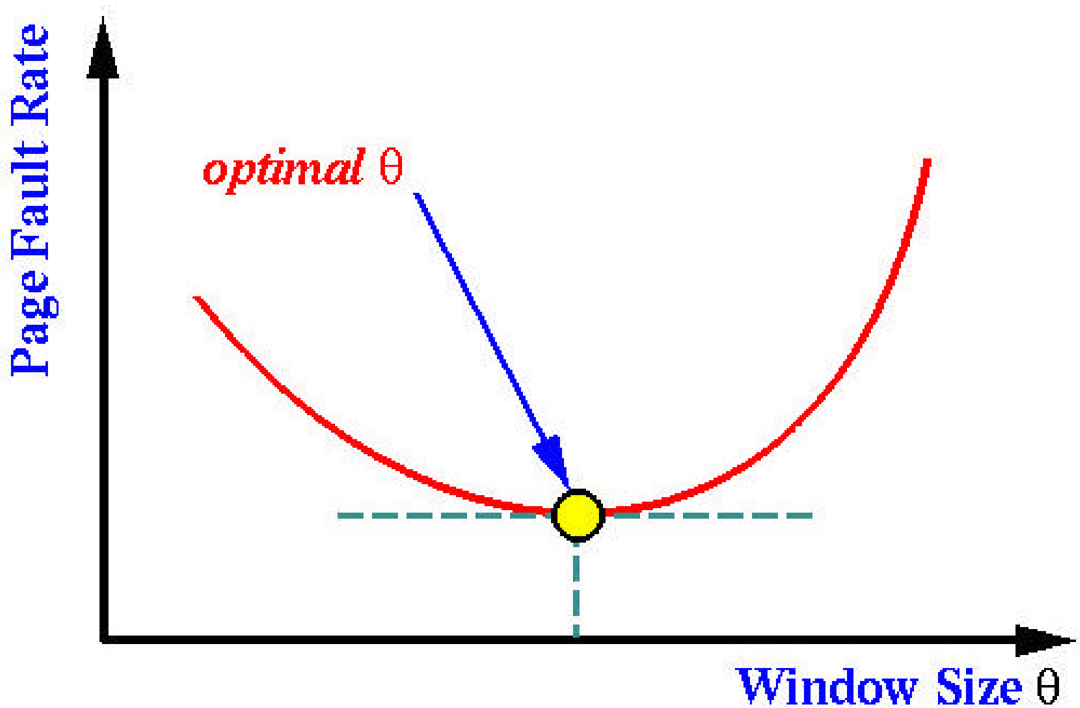
	0	1	2	3	0	1	4	0	1	2	3	4
	0	0	0	0	0	0	0	0	0	0	0	4
		1	1	1	1	1	1	1	1	1	1	1
			2	2	2	2	4	4	4	4	3	3
				3	3	3	3			2	2	2

page fault=8 miss ratio=8/12=66.7% hit ratio = 33.3%

only three pages here

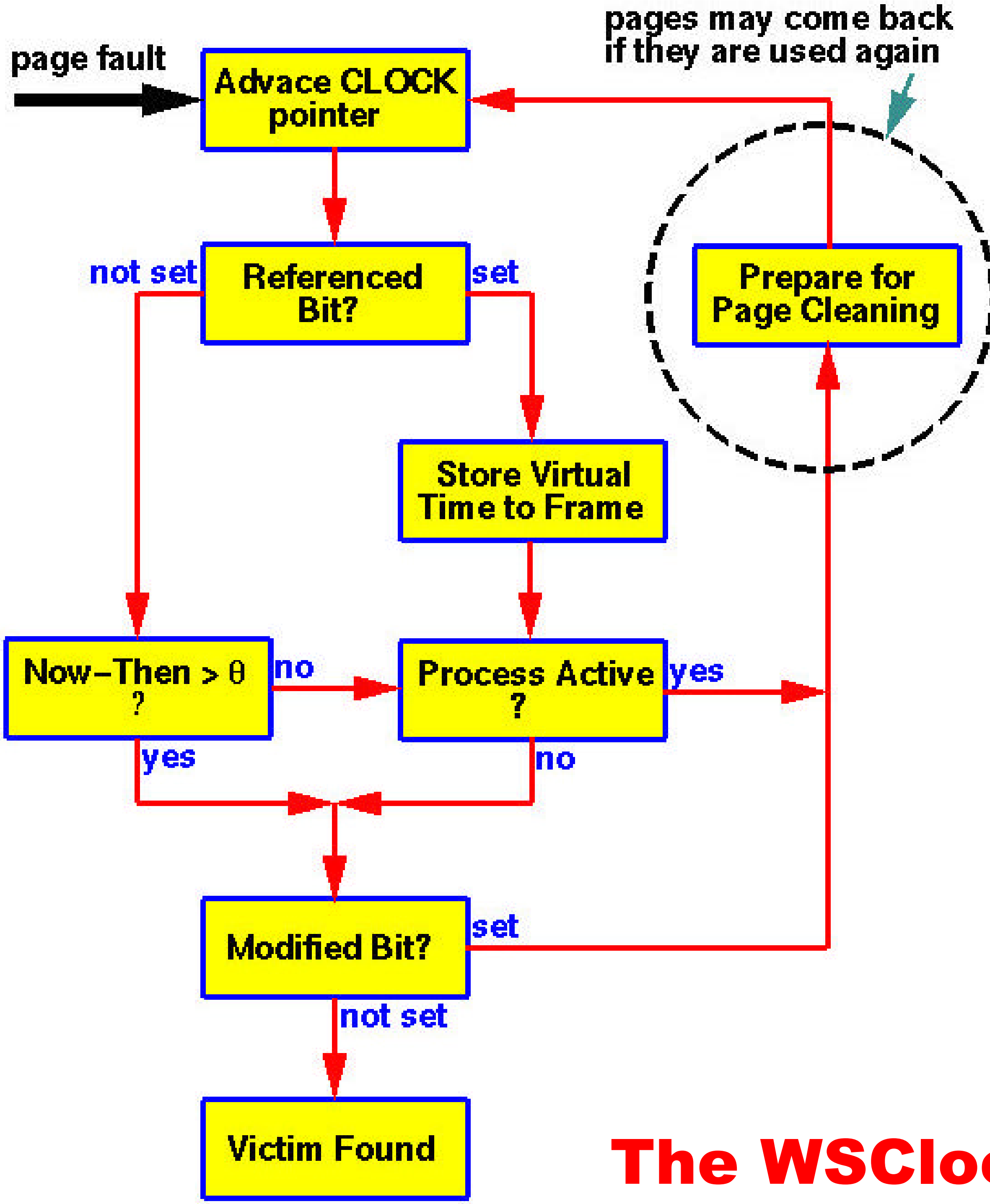
The Working Set Model: 3/4

- **The Working Set Policy: Find a good q, and keep $W(t,q)$ in memory for every t.**
- **What is the best value of q? This is a system tuning issue. This value can change as needed from time to time.**



The Working Set Model: 4/4

- Unfortunately, like LRU, the working set policy cannot be implemented directly, and an approximation is necessary.
- A commonly used algorithm is the **Working Set Clock algorithm**, **WSClock**. This is a good and efficient approximation.



The WSClock Algorithm

The Page-Fault Frequency Algorithm: 1/2

- Since thrashing is due to high page-fault rate, we can control thrashing by controlling page-fault rate.
- If the page-fault rate of a process is too high, this process needs more page frames. On the other hand, if the page-fault rate is too low, this process may have too many page frames.
- Therefore, if we can always maintain the page-fault rate of a process to certain level, we control the number of page frames that process can have.

The Page-Fault Frequency Algorithm: 2/2

- We establish an upper bound and a lower bound, and monitor the page-fault rate periodically.
- If the rate is higher (*resp*., lower) than the upper (*resp*., lower) bound, a new (*resp*., existing) page is allocated to (*resp*., removed from) this process.