Virtual Memory Management

***Throughout the course we will use overheads that were adapted from those distributed from the textbook website.**
Slides are from the book authors, modified and selected by Jean Mayo, Shuai Wang and C-K Shene.

The danger of computers becoming like humans
is not as great as the danger of humans becoming like computers.

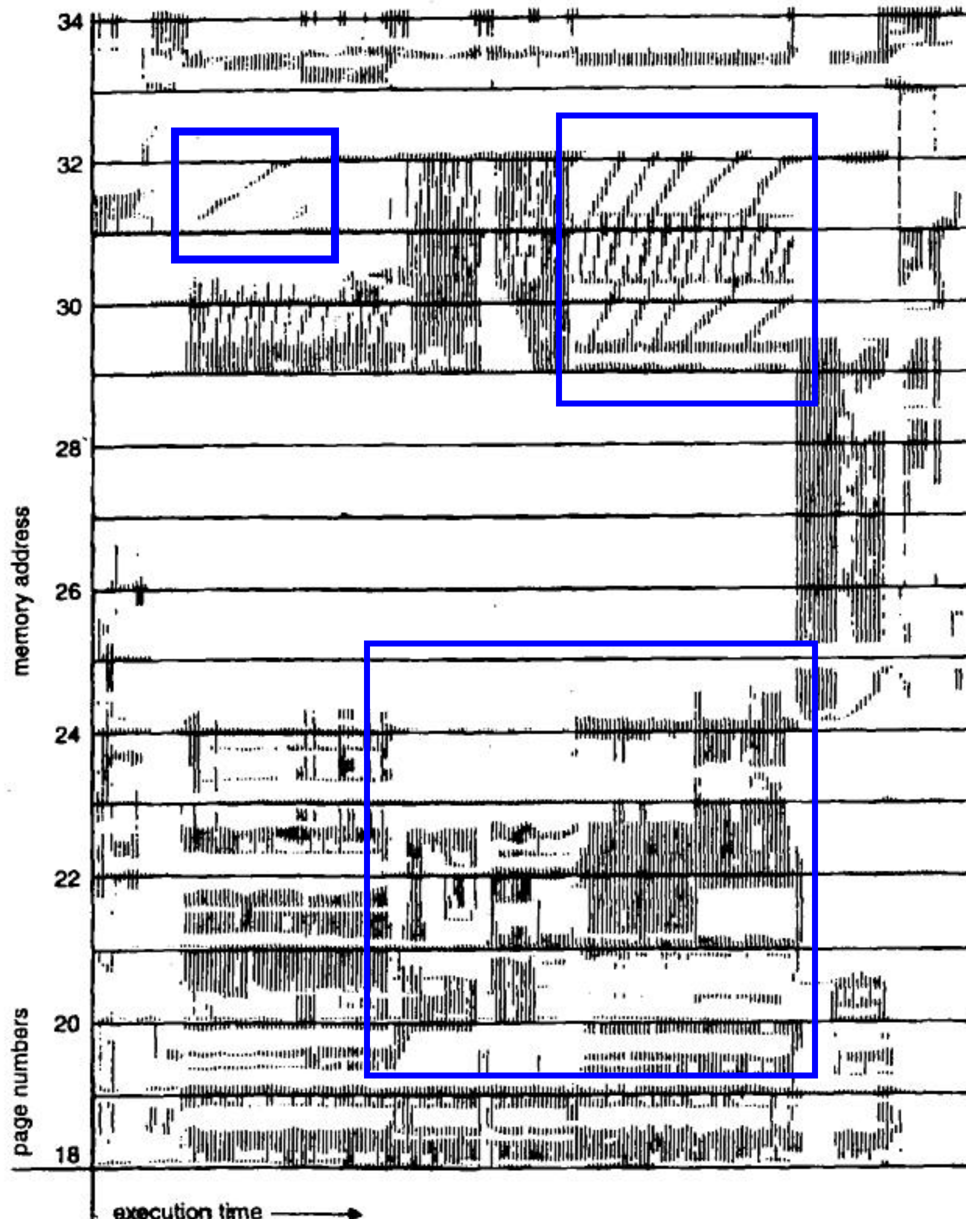
Konrad Zuse

Spring 2019

Definitions

- Cache
  - Copy of data that is faster to access than the original
  - **Hit**: if cache has copy
  - **Miss**: if cache does not have copy
- Cache block
  - Unit of cache storage (multiple memory locations)
- Temporal locality
  - Programs tend to reference the same memory locations multiple times
  - Example: instructions in a loop
- Spatial locality
  - Programs tend to reference nearby locations
  - Example: data in a loop

Locality of Reference



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

Main Points

- **Can we provide the illusion of near infinite memory in limited physical memory?**
  - **Demand-paged virtual memory**
  - **Memory-mapped files**
- **How do we choose which page to replace?**
  - **FIFO (First-In-First-Out), MIN (Optimal), LRU (Least Recently Used), LFU (Least Frequently Used), Second Chance, Clock**

Observations

- A complete program does not have to be in memory, because
  - error handling codes are not frequently used
  - arrays, tables, large data structures usually allocate 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?

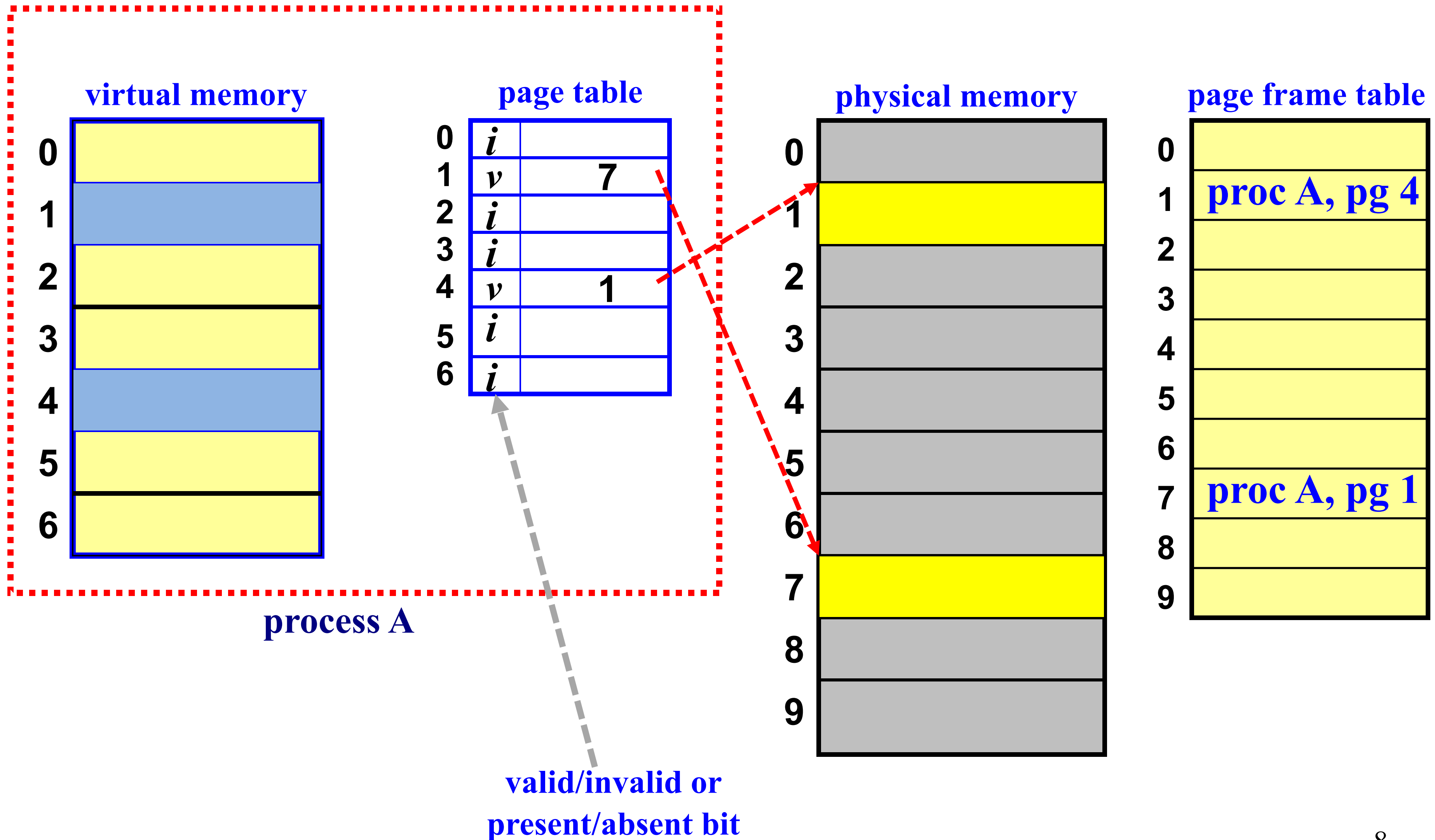
Benefits

- Program length is not restricted to real memory size. That is, virtual address size can be larger than physical memory size.
- Can run more programs because 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.

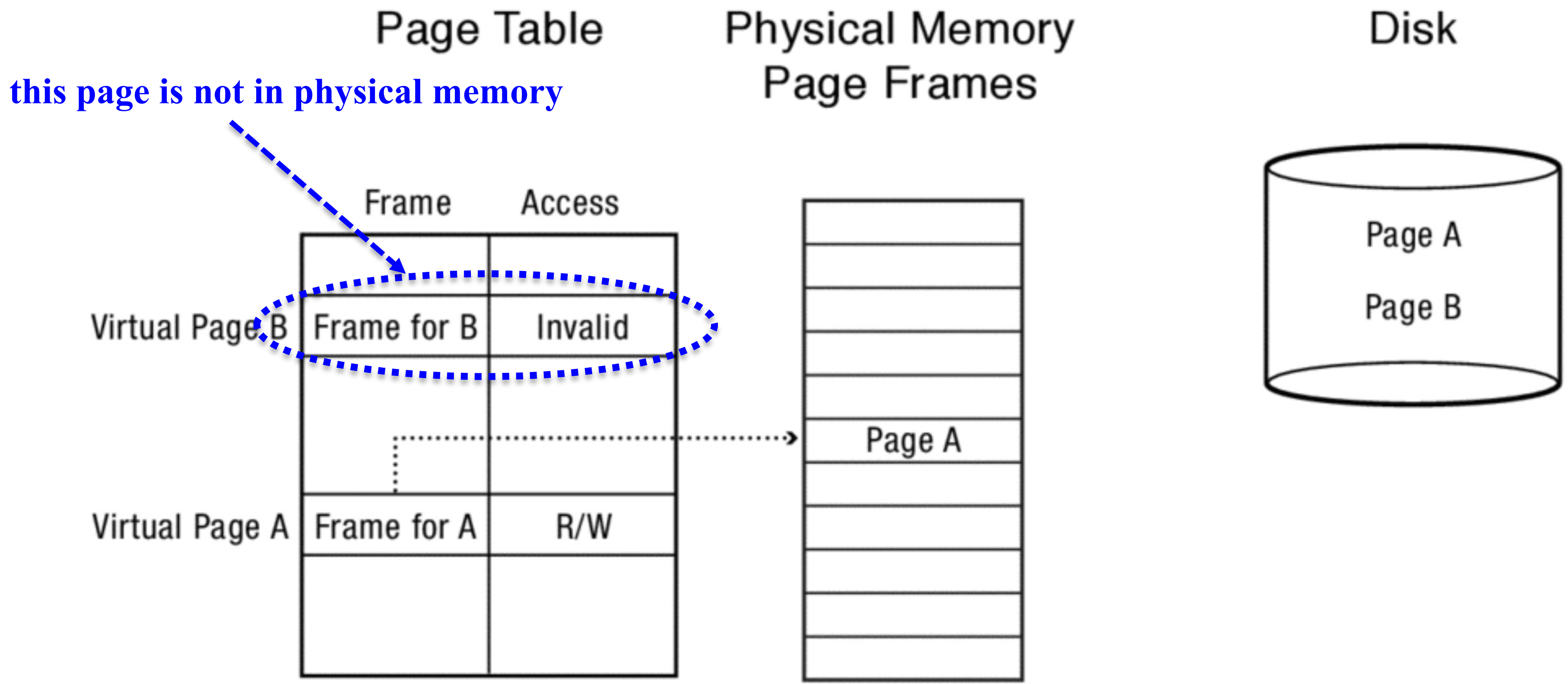
Virtual Memory

- **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 easier.**
- **Because memory segments 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

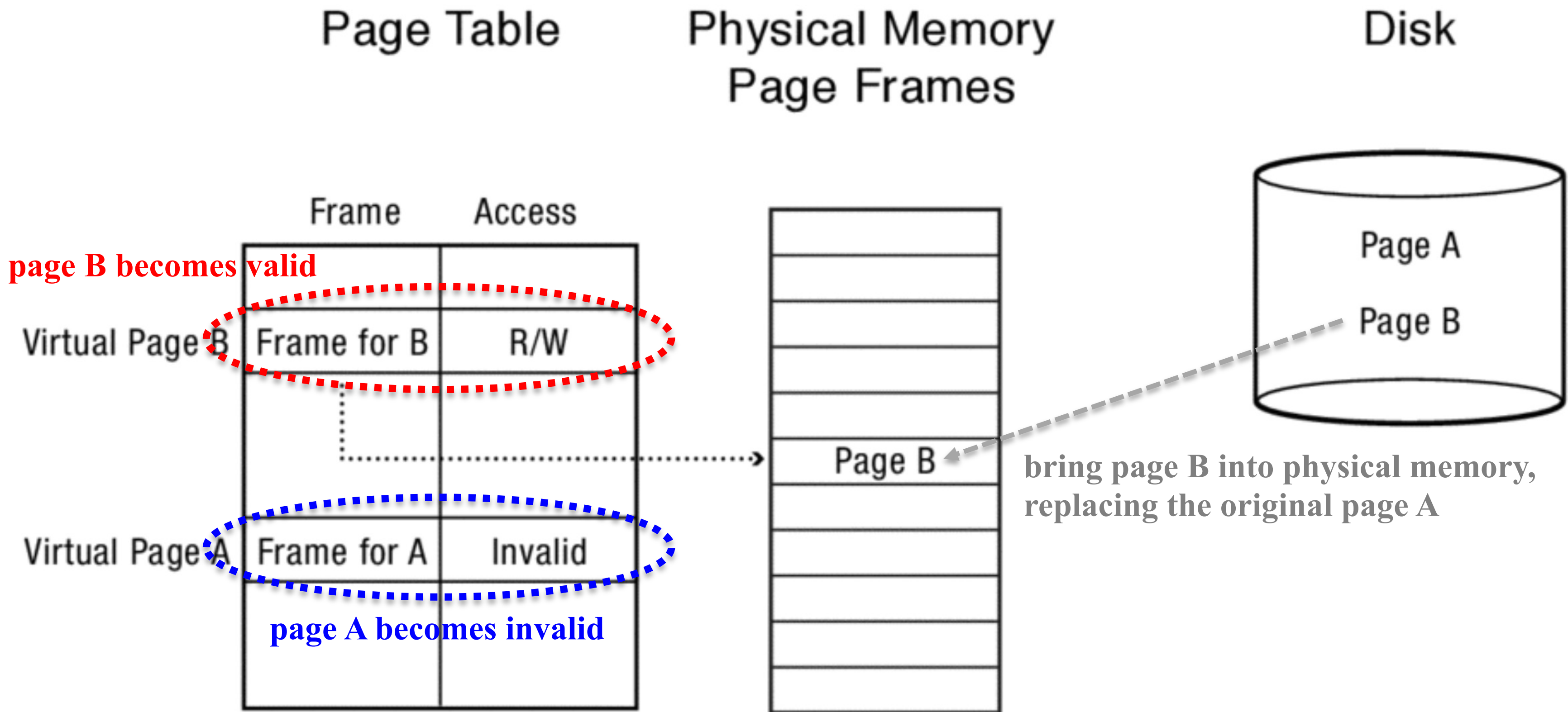


Demand Paging (Before)



- If a process accesses a page that is not in physical memory, a page fault (trap) is generated and trapped to the kernel.
- The kernel will find the needed page and load it into physical memory.
- The kernel also modifies the page table.

Demand Paging (After)



- The kernel finds the page in virtual memory, brings it into physical memory.
- If there is no available page frame available, the kernel find an “occupied” one.
- Suppose page A was chosen. The kernel brings page B in to replace page A.
- The kernel update page table.

Address Translation

- **Address translation from a *virtual address* to a *physical address* is the same as a paging system.**
- **However, there is an additional check. If the needed page is not in physical memory (*i.e.*, its 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

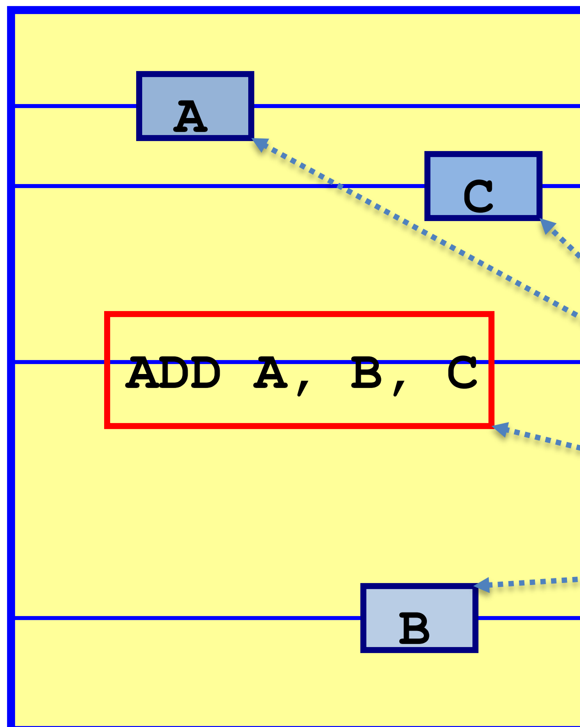
Trap to the OS **// a context switch occurs**
Make sure it is a page fault;
If the address is not a legal one then
address error, return
Find a page frame **// page replacement algorithm**
Write the victim page back to disk **// page out (if modified)**
Load the new page from disk // page in
Update both 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 a 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.**

Too Many Memory Accesses?!

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



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

Some CISC architecture machine instructions and operands can be rather long that could cross page boundary.

Performance Issue: 1/2

- Let p be the probability of a page fault, the **page fault rate**, $0 \le p \le 1$.
- 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
  - Page-in and page-out, *a bottleneck*
  - Resume the interrupted 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 milliseconds)**

  = (1-*p*)*100 + *p25,000,000 nanoseconds**

  = 100 + 24,999,900p* nanoseconds**

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

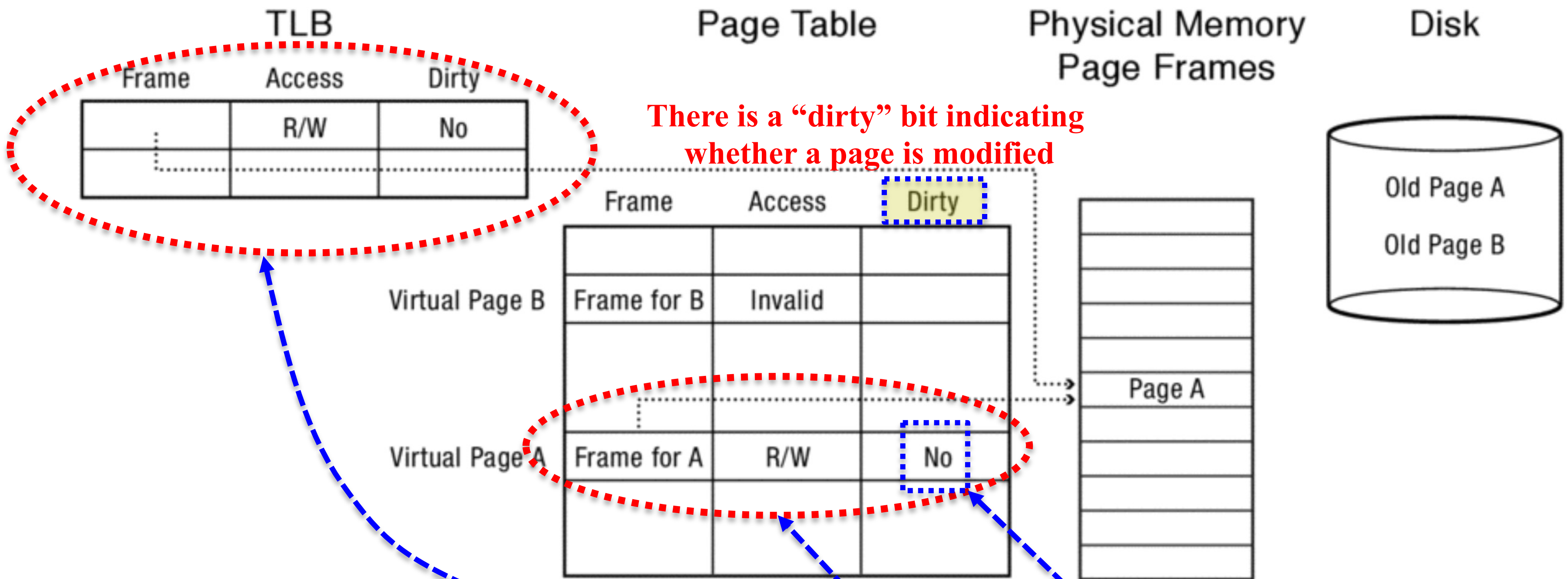
Three Important Issues in V.M.

- **Page tables 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. Remove any TLB entries (i.e., copies of now invalid page table entry).
- **Page replacement.** When a page fault occurs and there is no free page frame, a victim page must be found. If the victim is not selected properly, system degradation may be high.

How Do We Know If Page Has Been Modified?

- Every page table entry has some bookkeeping
  - Has page been modified?
    - Set by hardware on store instruction
    - In both TLB and page table entry
  - Has page been recently used?
    - Set by hardware on in page table entry on every TLB miss
- Bookkeeping bits can be reset by the OS kernel
  - When changes to page are flushed to disk
  - To track whether page is recently used

Keeping Track of Page Modifications (Before)

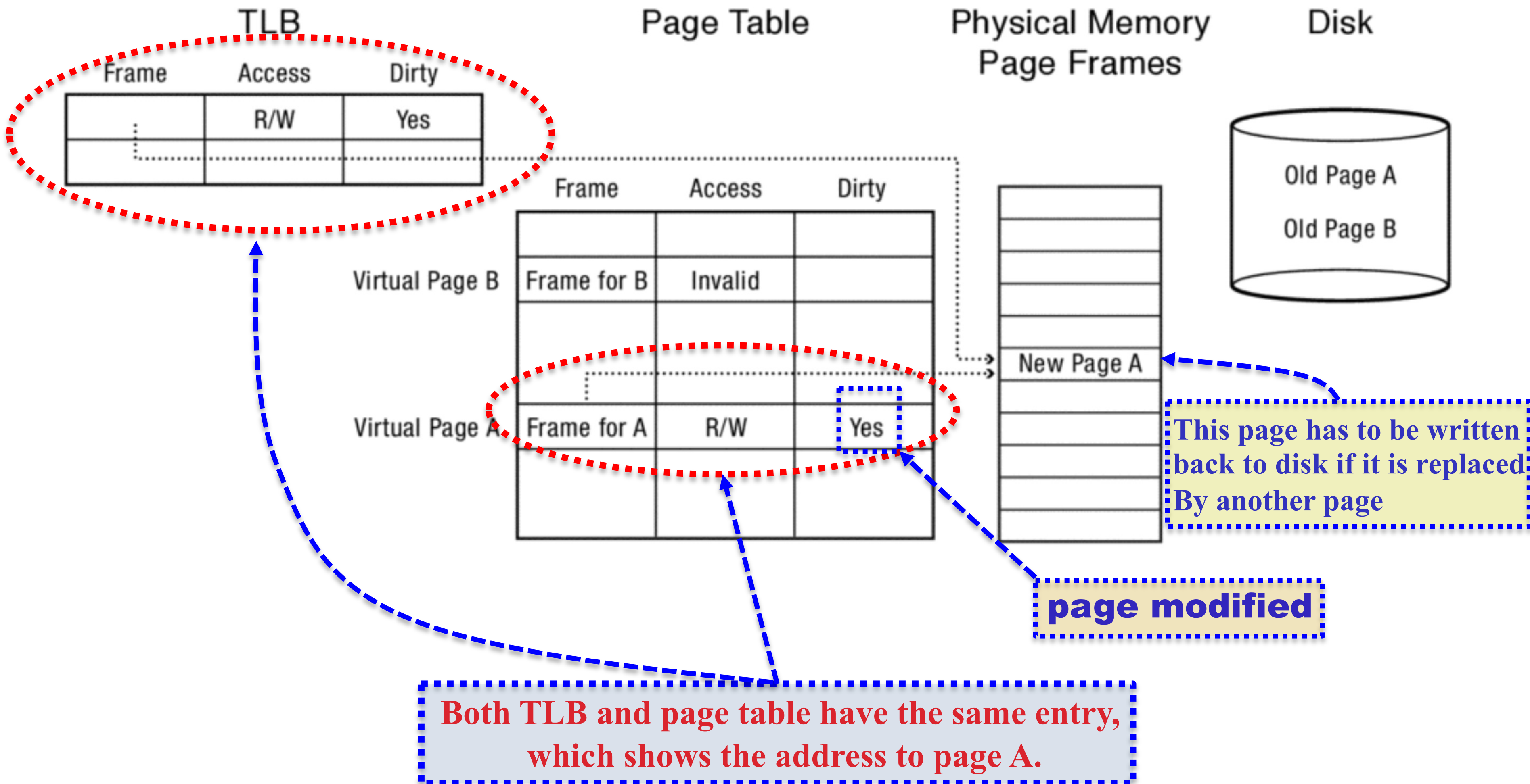


Some systems have a **reference** bit indicating whether a page has been used since it was loaded into memory. Whenever a location in a page is used (e.g., load, save, etc.), the reference bit is set. Of course, it is modified, the dirty and reference bits are all set.

Both TLB and page table have the same entry, which shows the address to page A.

page not modified

Keeping Track of Page Modifications (After)



Modified/Dirty & Referenced/Used Bits

- **Most machines keep dirty/use bits in the page table entry**
- **Physical page is**
  - **Modified if *any* page table entry that points to it is modified (Modified/Dirty bit)**
  - **Recently used if *any* page table entry that points to it is recently used (Referenced/Used bit)**
- **On MIPS, simpler to keep dirty/use bits in the core map**
  - **Core map: map of physical page frames**

Page Replacement: 1/2

- The following is a basic scheme
  - Find the desired page on 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 update the page table and page frame table
  - Read the desired page into the selected frame and update page tables and page frame table
  - Restart the interrupted instruction

Page Replacement: 2/2

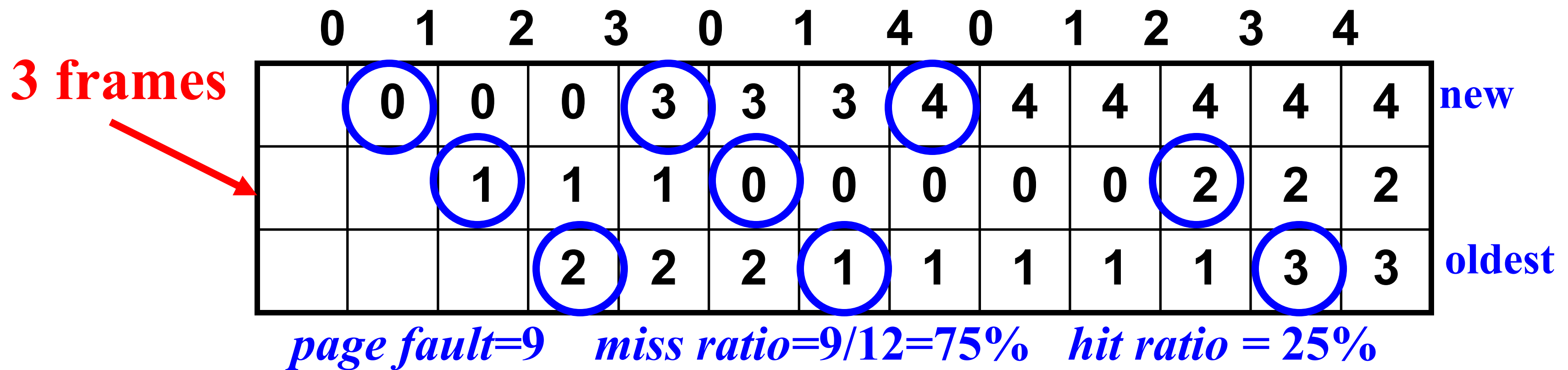
- If there is no free page frame, two page transfers (*i.e.*, page-in and page-out) may be required.
- A modified bit may be added to a page table entry. The modified bit is set if that page has been modified (*i.e.*, storing info into it). It is initialized to 0 when a page is loaded into memory.
- Thus, if a page is not modified (*i.e.*, modified bit = 0), it does not have to be written back to disk.
- Some systems may also have a referenced bit. When a page is referenced (*i.e.*, reading or writing), its referenced 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 performs.
- Page replacement algorithms work on page numbers. A string of page numbers is referred to as a page reference string.

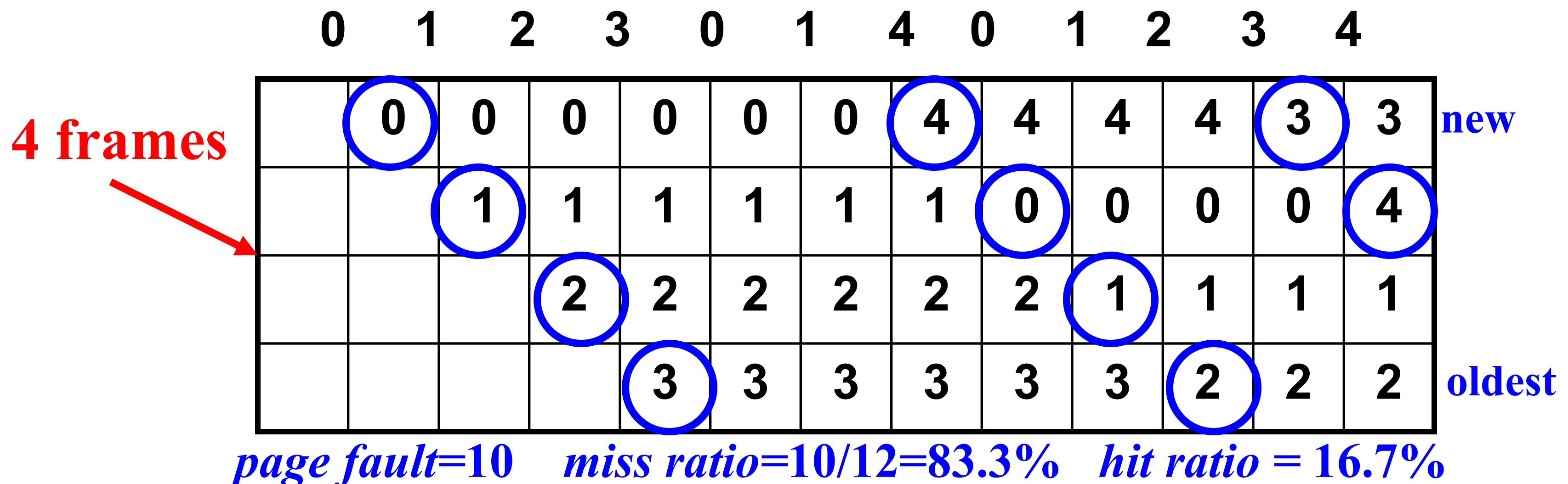
The FIFO Algorithm

- The FIFO algorithm always selects the "oldest" page to be the victim. Columns organized by "age".



	0	1	2	3	0	1	4	0	1	2	3	4
new	0	0	0	3	3	3	4	4	4	4	4	4
		1	1	1	0	0	0	0	0	2	2	2
oldest			2	2	2	1	1	1	1	1	3	3

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



	0	1	2	3	0	1	4	0	1	2	3	4
new	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
oldest				3	3	3	3	3	3	2	2	2

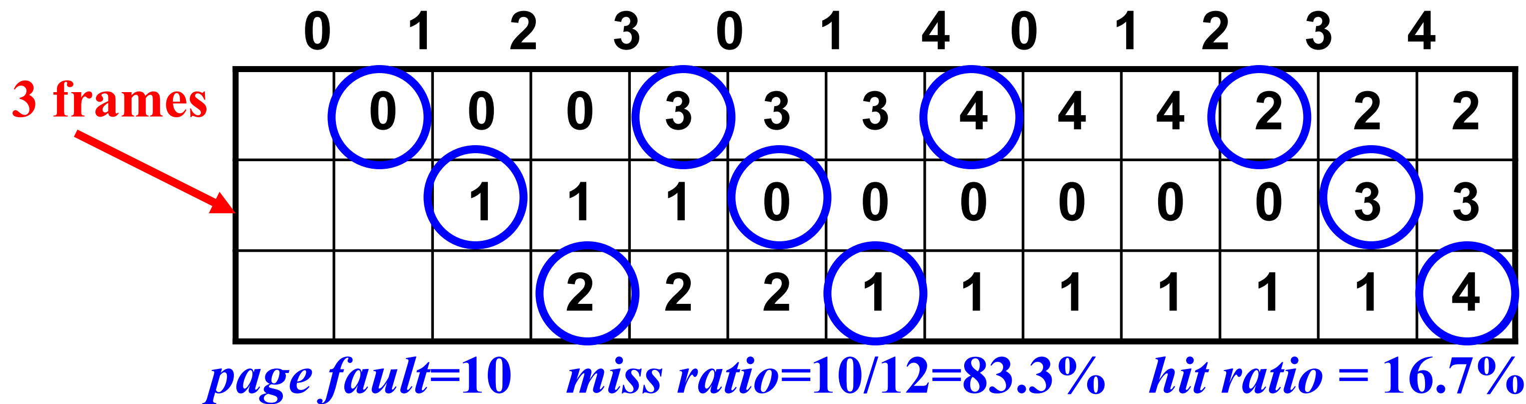
page fault=10 *miss ratio*=10/12=83.3% *hit ratio* = 16.7%

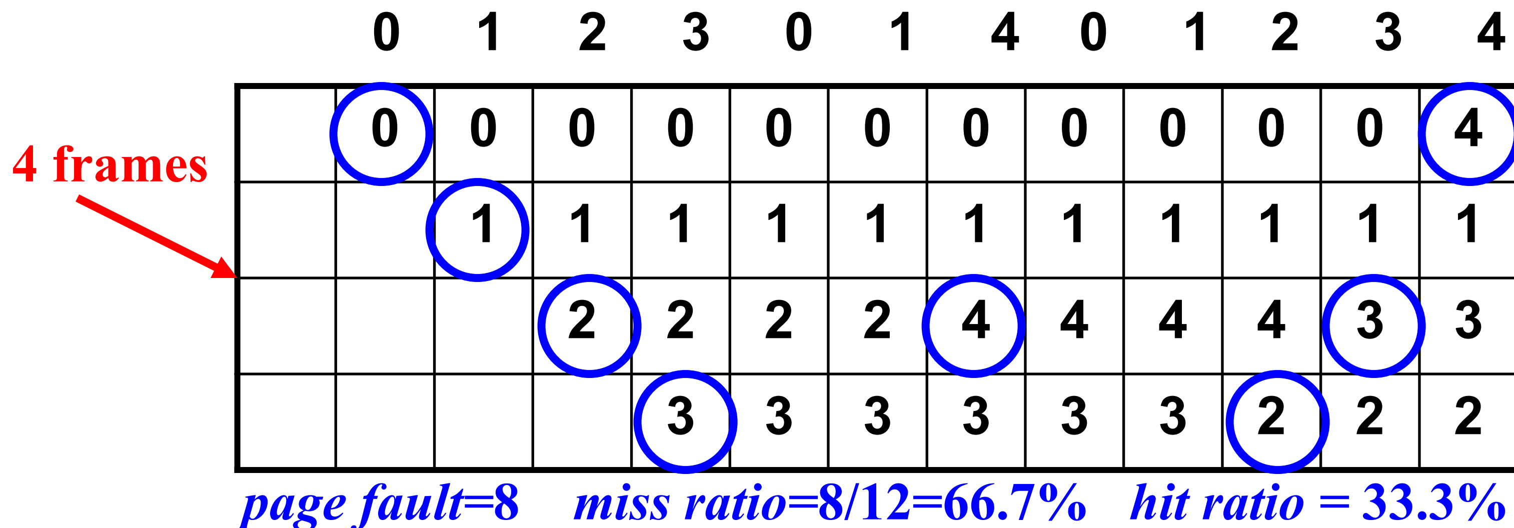
Belady Anomaly

- **Intuitively, increasing the number of page frames should reduce the number of page faults.**
- **However, some page replacement algorithms do not satisfy this "intuition." The FIFO algorithm is an example.**
- **Belady Anomaly: Page faults may increase as the number of page frames increases.**
- **FIFO was used in DEC VAX-78*xx* series and NT 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**.





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. **Why?**

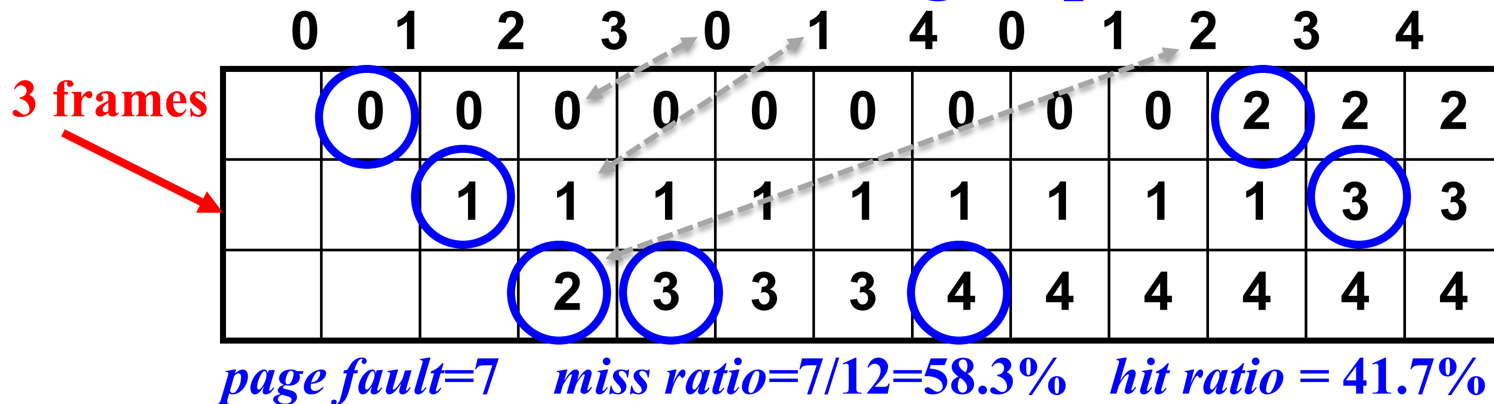
	0	1	2	3	0	1	4	0	1	2	3	4
	0	0	0	3	3	3	4	4	4	2	2	2
		1	1	1	0	0	0	0	0	0	3	3
			2	2	2	1	1	1	1	1	1	4

◯ *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

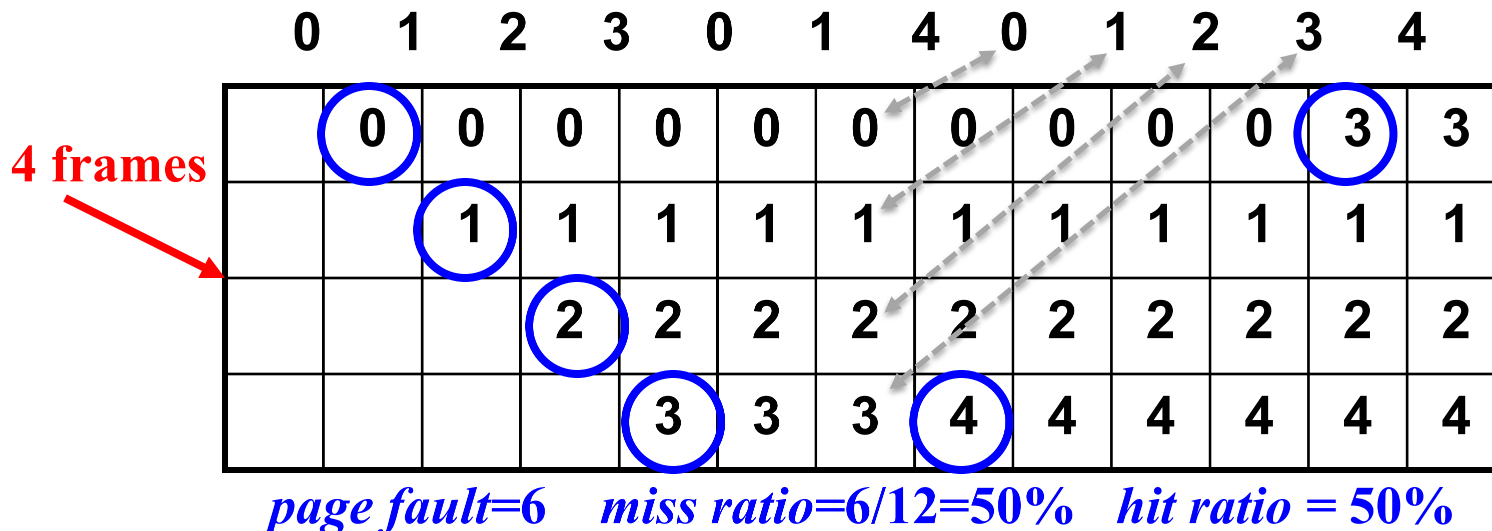
The Optimal Algorithm: 1/2

- 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

extra

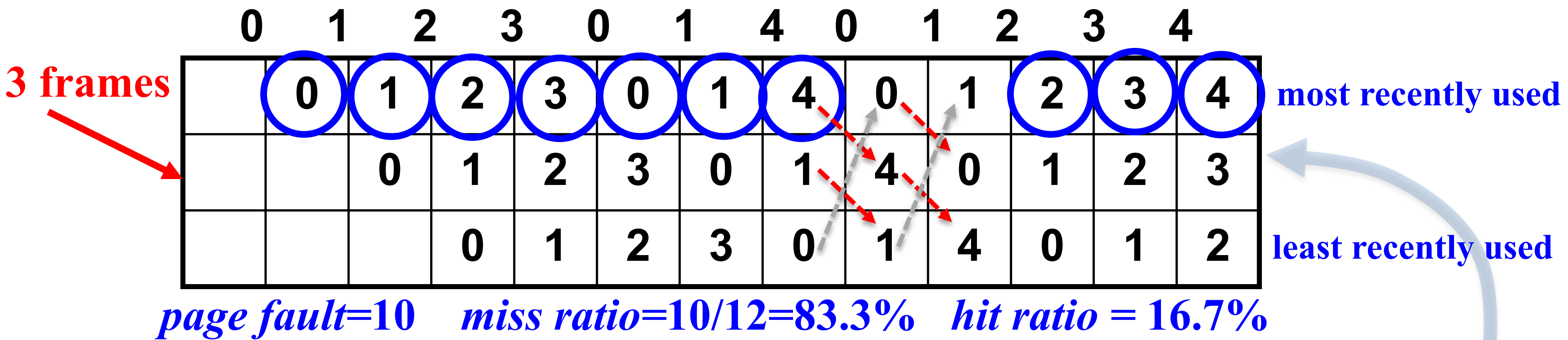
	0	1	2	3	0	1	4	0	1	2	3	4
	0	0	0	0	0	0	0	0	0	0	3	3
		1	1	1	1	1	1	1	1	1	1	1
			2	2	2	2	2	2	2	2	2	2
				3	3	3	4	4	4	4	4	4

The Inclusion Property

- Define the following notations:
  - $P = \langle p_1, p_2, \ldots, p_n \rangle$: a page trace
  - m : the number of page frames
  - $M_t(P, \alpha, m)$: the memory content after page p_t is referenced with respect to a page replacement algorithm α.
- A page replacement algorithm satisfies the **inclusion property** if $M_t(P,\alpha,m) \subseteq M_t(P,\alpha,m+1)$ holds for every t.
- **Homework:** Inclusion property means no Belady anomaly.

LRU Revisited

- Pages on each column are ordered from most recently used to least recently used.



4 frames

	0	1	2	3	0	1	4	0	1	2	3	4	
	0	1	2	3	0	1	4	0	1	2	3	4	most recently used
		0	1	2	3	0	1	4	0	1	2	3	
			0	1	2	3	0	1	4	0	1	2	
				0	1	2	3	3	3	4	0	1	least recently used

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

Do the same for the optimal algorithm MIN

LRU Approximation Algorithms

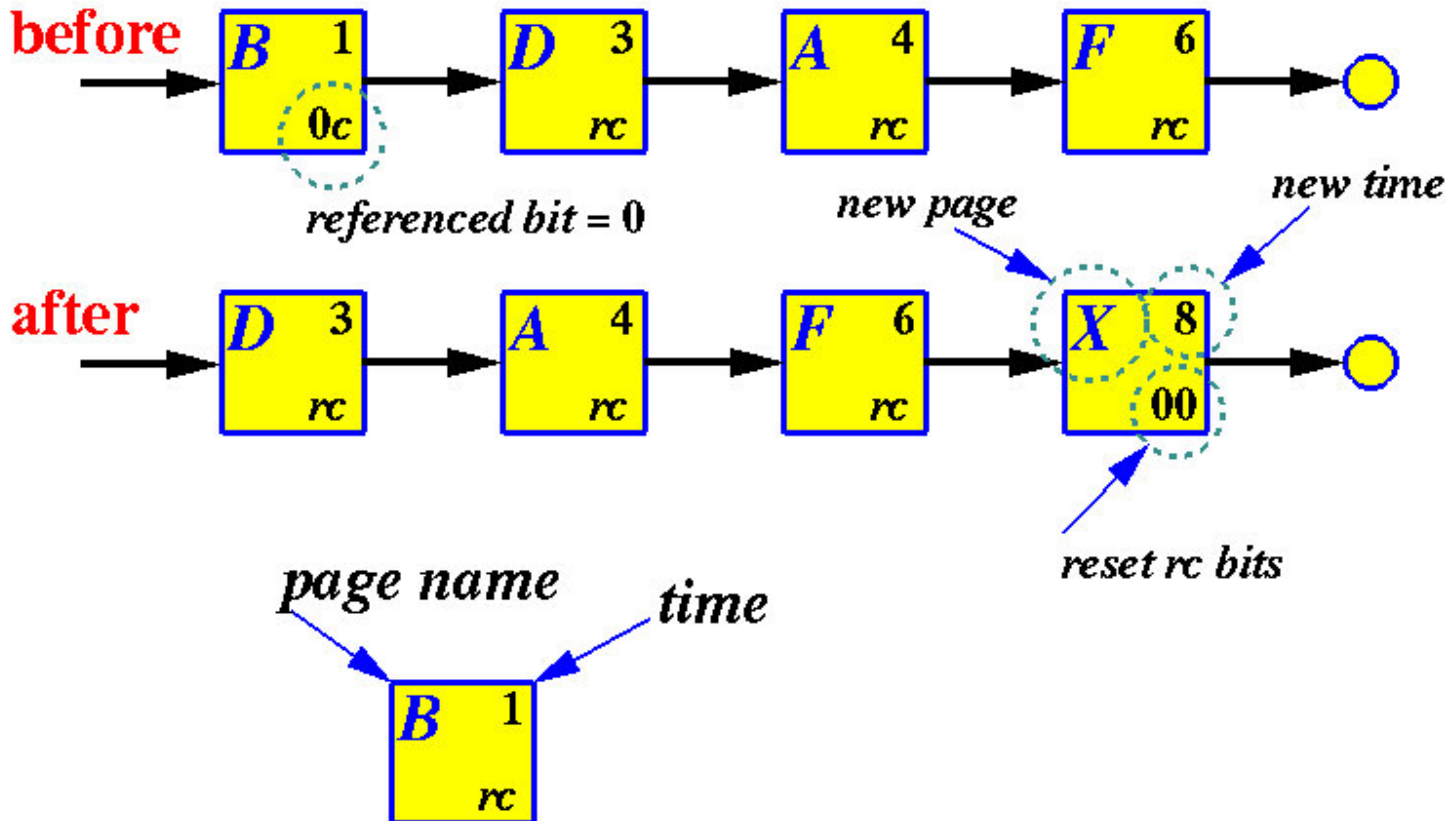
- FIFO has Belady anomaly, the Optimal algorithm requires the knowledge in the future, and the LRU algorithm requires accurate info of the past.
- The optimal and LRU algorithms are difficult to implement, especially the optimal algorithm. Thus, LRU approximation algorithms are needed. We will discuss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

Second-Chance Algorithm: 1/3

- The second chance algorithm is a FIFO algorithm. It uses the referenced bit of each page.
- The page frames are in page-in order (linked-list).
- If a page frame is needed, check the oldest (head):
  - If its referenced bit is 0, take this one
  - Otherwise, clear the referenced bit, move it to the tail, and (perhaps) set the current time. This gives this page frame a second chance.
- Repeat this procedure until a 0 referenced 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.

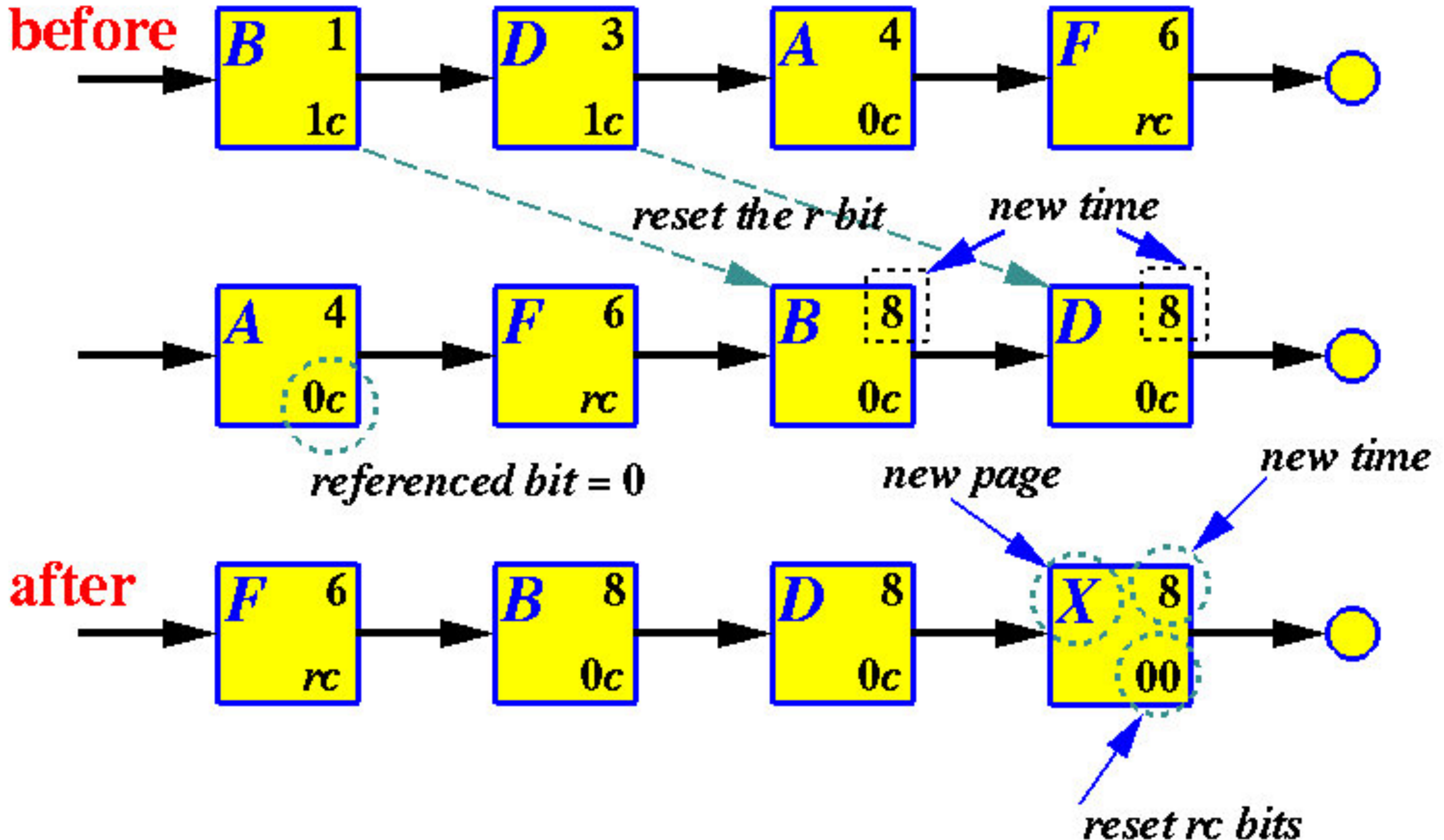
Second-Chance Algorithm: 2/3

new page = X *rc* = **referenced and changed/modified bit pair**



Second-Chance Algorithm: 3/3

new page = X

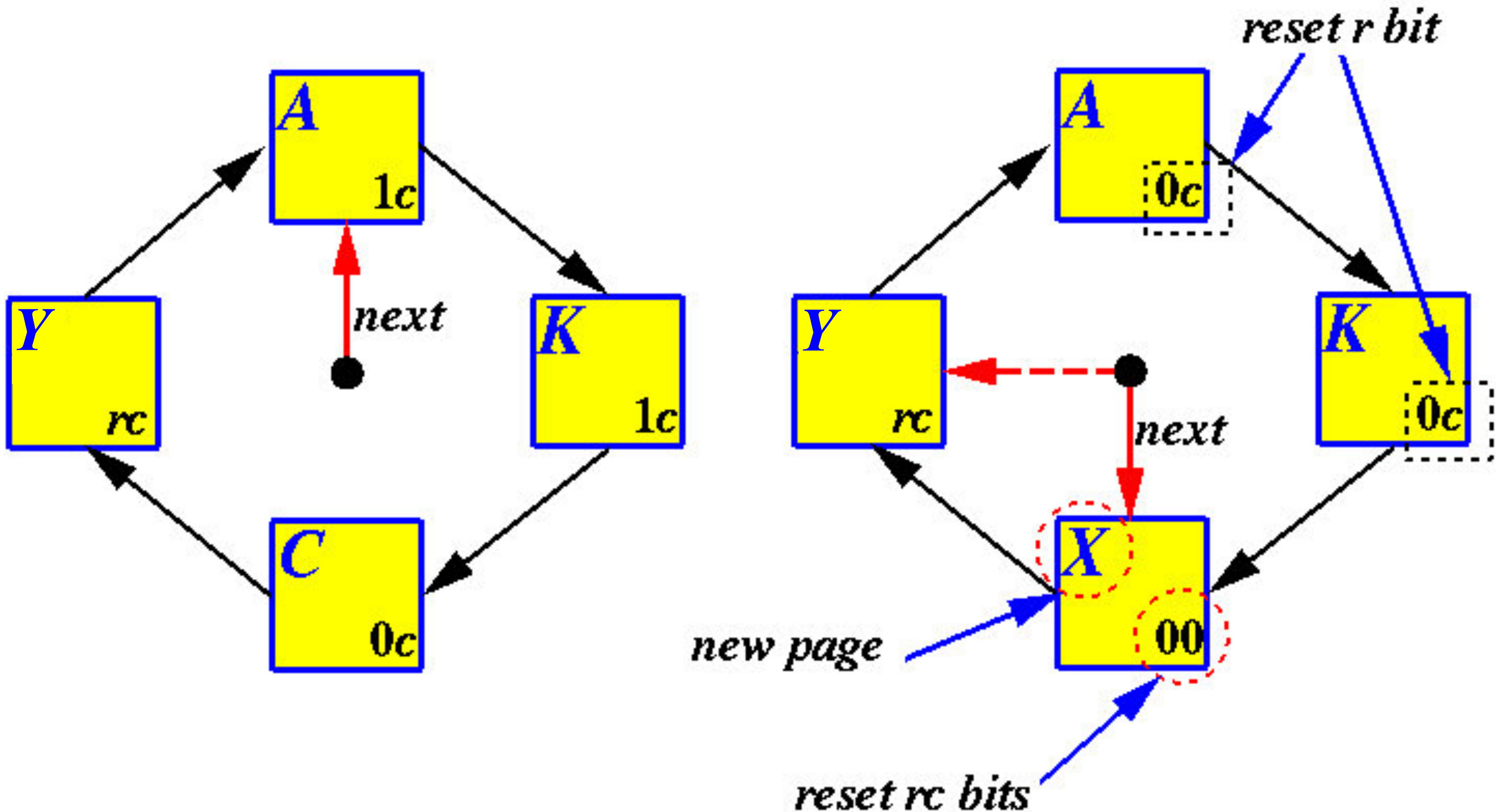


The Clock Algorithm: 1/2

- If the second chance algorithm is implemented with a *circular* list, we have the clock algorithm.
- A "next" pointer is needed.
- When a page frame is needed, we examine the page under the "next" pointer:
  - If its referenced 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



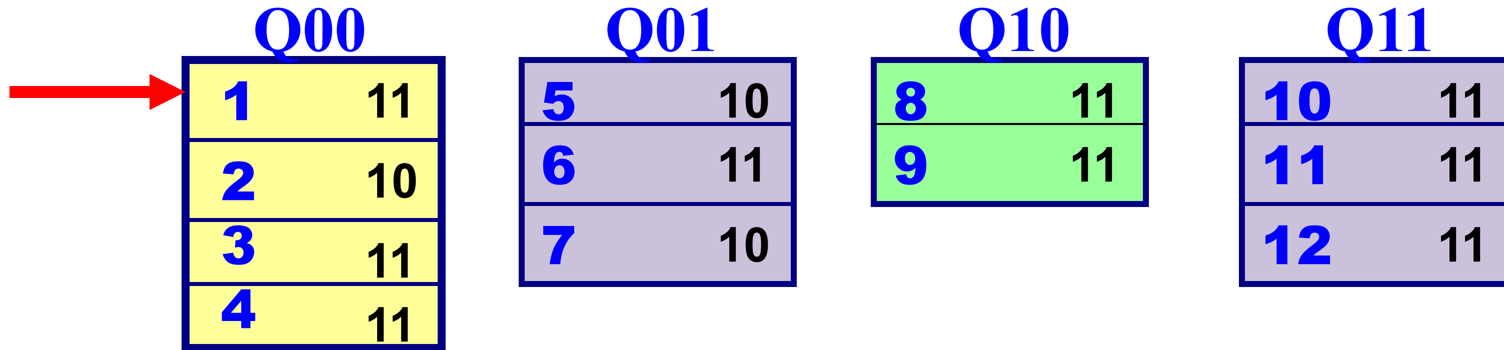
Enhanced Second-Chance Algorithm: 1/5

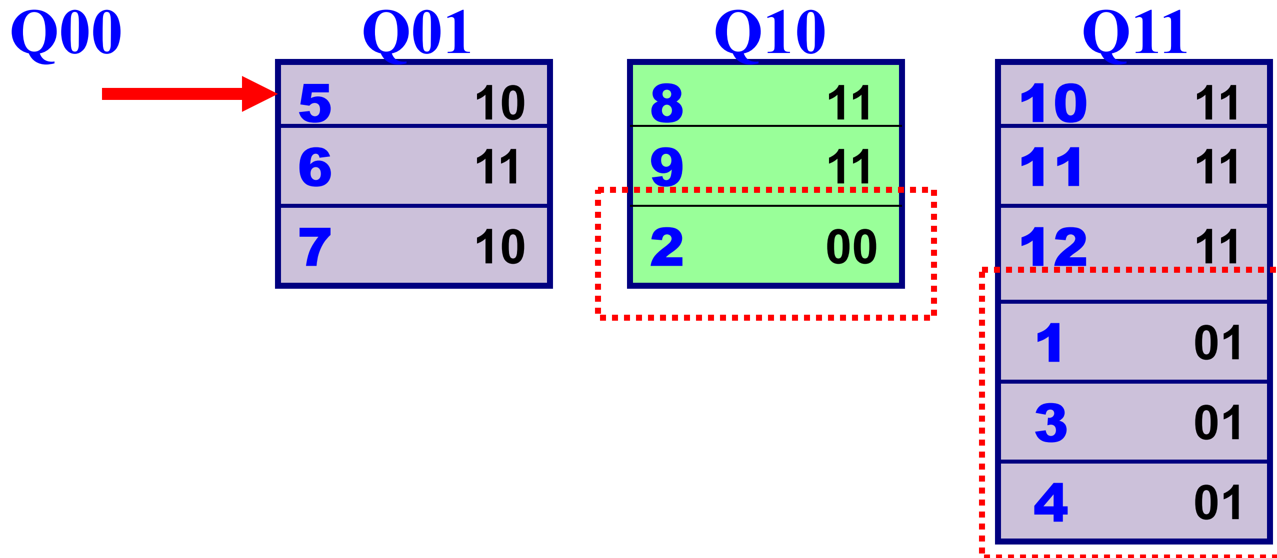
- Four page lists based on their reference-modify bits (*r*,*c*) are used:
  - **Q00** - pages were not recently referenced and not modified, the best candidates!
  - **Q01** - pages were changed but not recently referenced. Need a page-out.
  - **Q10** - pages were recently used but clean.
  - **Q11**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

Enhanced Second-Chance Algorithm: 2/5

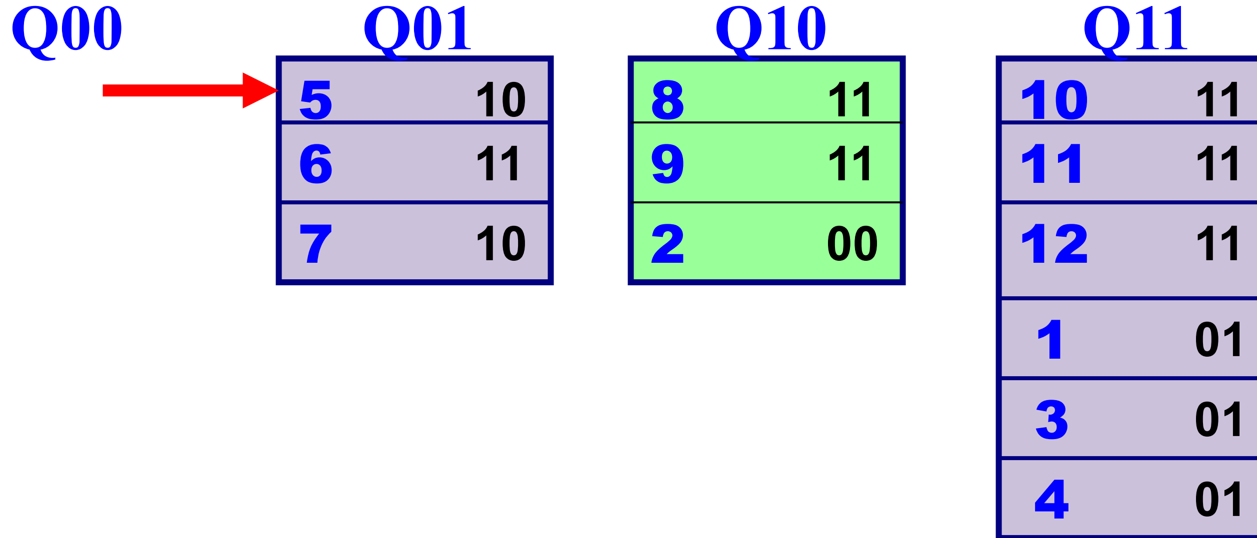
- We still need a "next" pointer.
- When a page frame is needed:
  - Does the "next" frame has 00 combination? If yes, victim is found. Otherwise, reset the referenced bit and move this page to the corresponding list (*i.e.*, Q10 or Q11).
  - If Q00 becomes empty, check Q01. If there is a frame with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

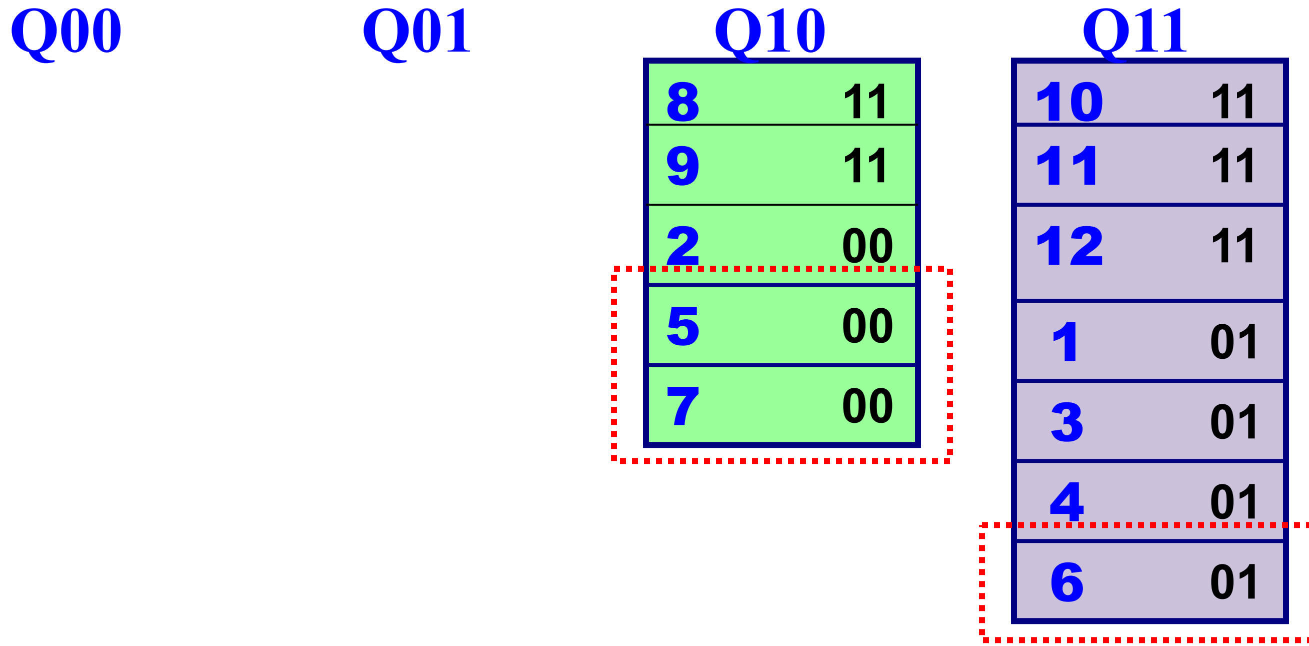
Enhanced Second-Chance Algorithm: 3/5





Enhanced Second-Chance Algorithm: 4/5





Q00	
8	11
9	11
2	00
5	00
7	00

Q01	
10	11
11	11
12	11
1	01
3	01
4	01
6	01

Q10

Q11

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

Q00	
2	00
5	00
7	00

Q01	
10	11
11	11
12	11
1	01
3	01
4	01
6	01

Q10

Q11	
8	01
9	01

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

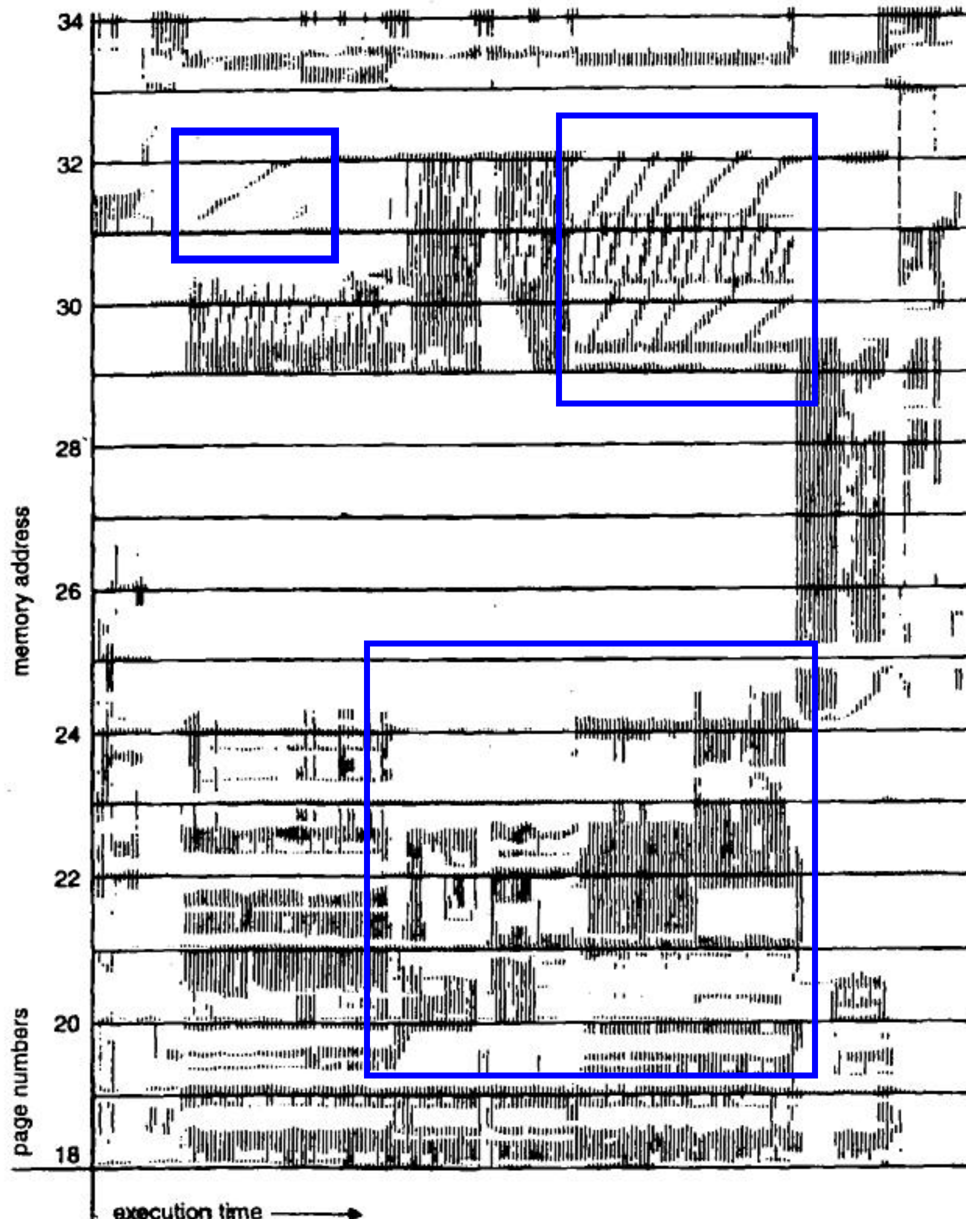
Global vs. Local Replacement

- **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, and, as a result, paging behavior of a process is affected by other processes and may be unpredictable.

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



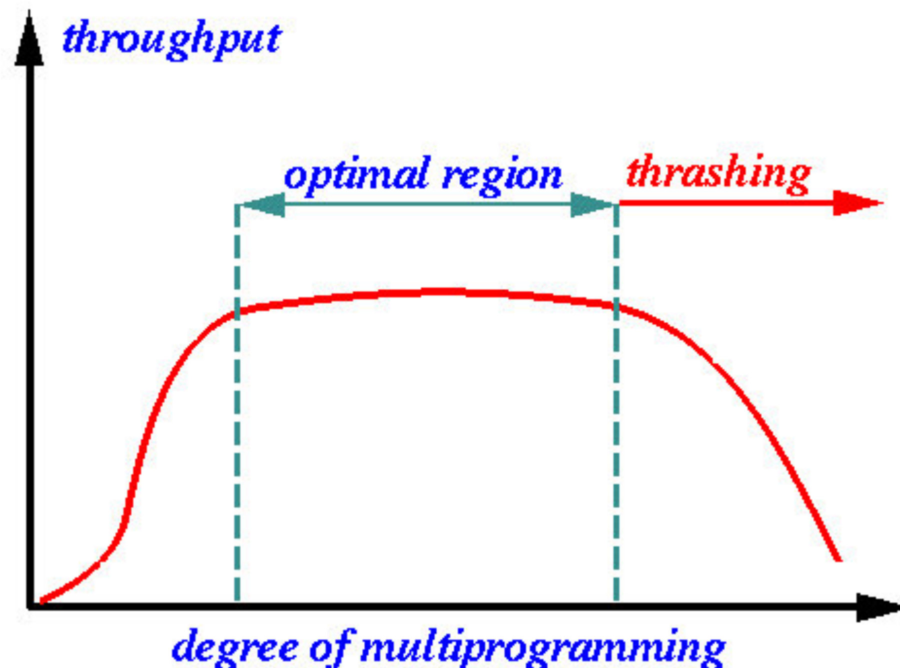
- During any phase of execution, the process references only a relatively small fraction of pages.

Thrashing

- **Thrashing** means a process spends more time paging than executing (*i.e.*, low CPU utilization and high paging rate).
- 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 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 $W(t,\theta)$, is the set of pages that were referenced in the interval $(t-\theta, t]$, where θ is the window size. These are "most recently used" pages, which can be ordered in the LRU way.
- $\theta = 3$. 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 $\theta = 4$ is different from that of LRU.

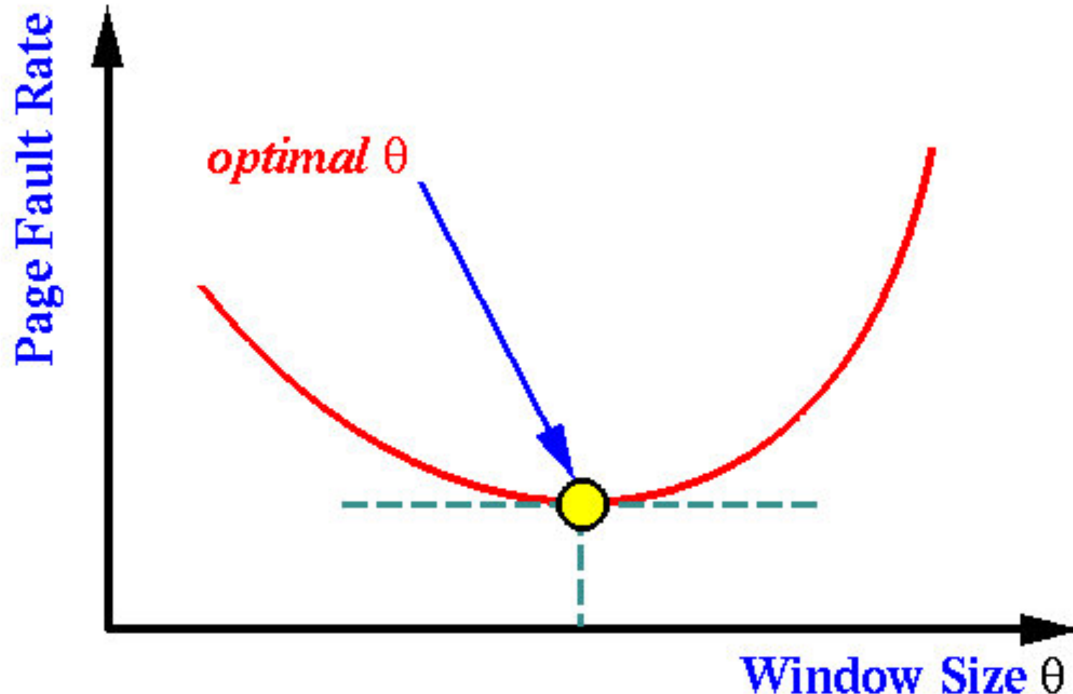
	0	1	2	3	0	1	4	0	1	2	3	4
	0	0	0	0	0	0	0	0	0	0	0	4
		1	1	1	1	1	1	1	1	1	1	1
			2	2	2	2	4	4	4	4	3	3
				3	3	3	3			2	2	2

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

only three pages here

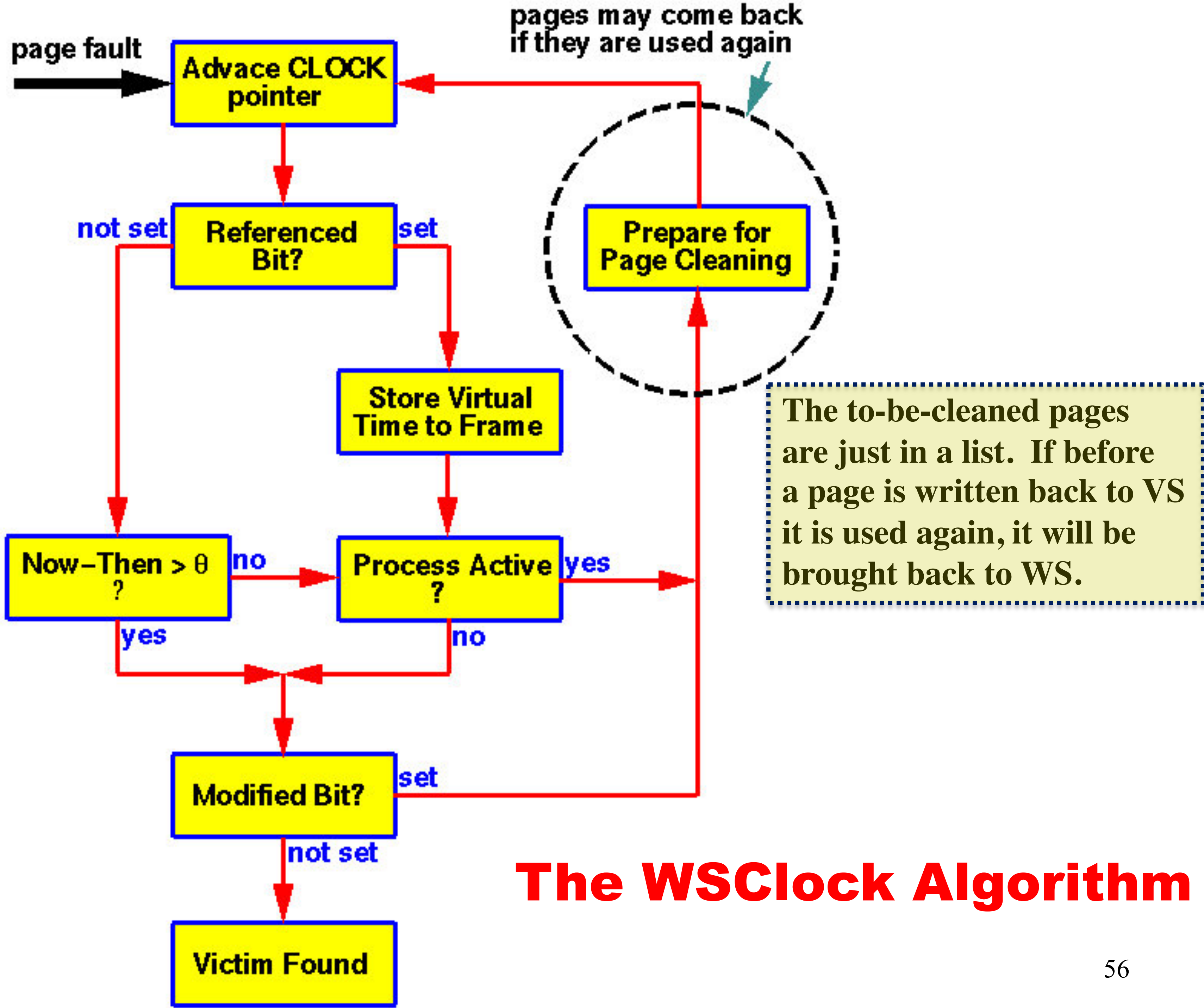
The Working Set Model: 3/4

- **The Working Set Policy**: Find a good θ, and keep $W(t,\theta)$ in memory for every t.
- What is the best value of θ? 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.**
- **But, the working set model does satisfy the inclusion property.**
- **A commonly used algorithm is the Working Set Clock algorithm, WSClock. This is a good and efficient approximation.**

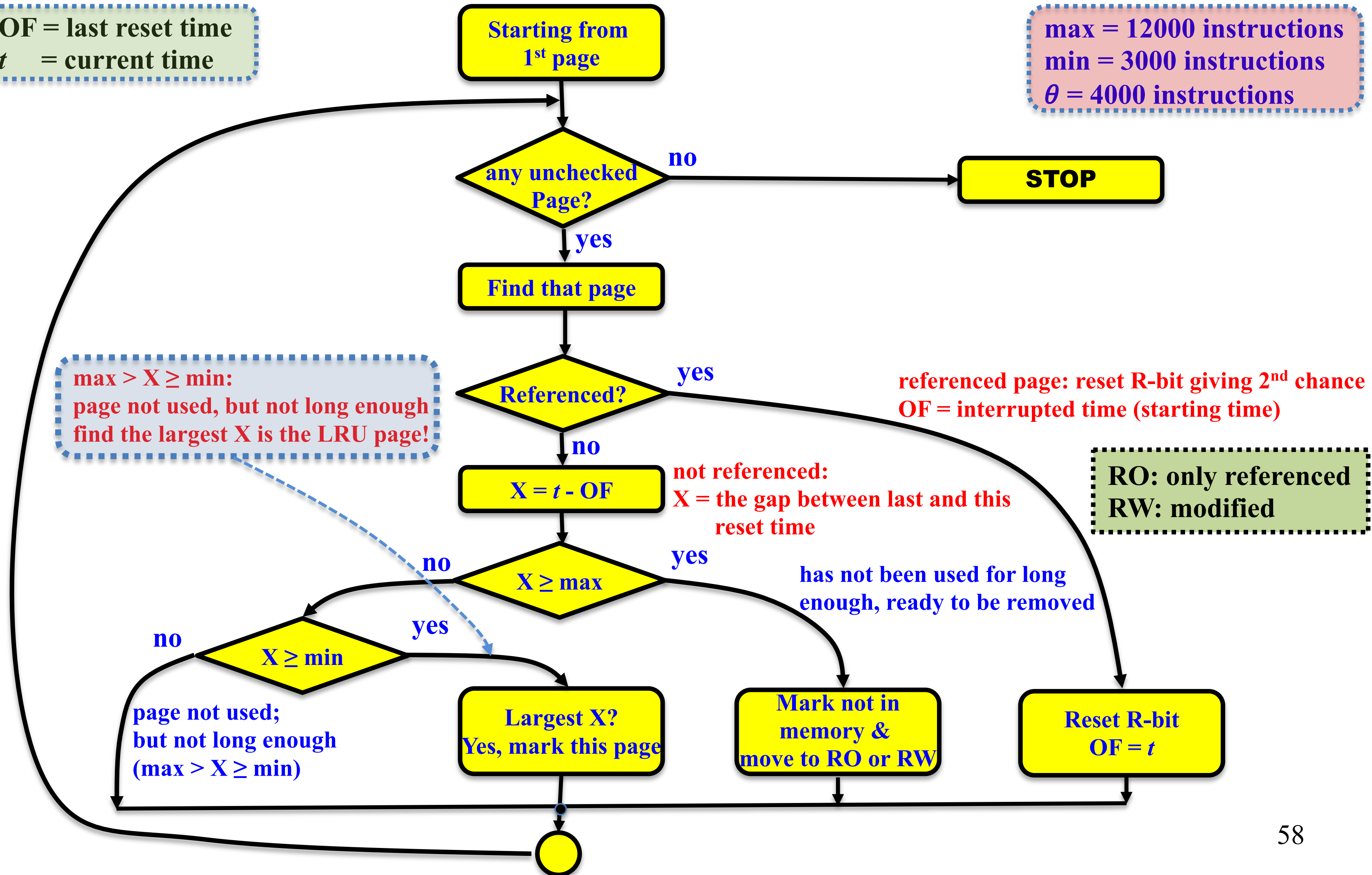

pages may come back
if they are used again
page fault
Advace CLOCK
pointer
not set
Referenced
Bit?
set
Prepare for
Page Cleaning
Store Virtual
Time to Frame
The to-be-cleaned pages are just in a list. If before a page is written back to VS it is used again, it will be brought back to WS.
Now−Then > θ
?
no
Process Active
?
yes
yes
no
Modified Bit?
set
not set
Victim Found
The WSClock Algorithm

Example VMOS: 1/2

- VMOS (**V**irtual **M**emory **O**perating **S**ystem) was an early OS (1970s) using working set.*
- This OS is designed for UNIVAC Spectra 70, similar to IBM System/370.
- Time for adjusting working set:
  1. Page fault
  2. A process finishes executing 4000 instruction. This time is the window size θ.
  3. For a process waiting for I/O, unless its working set has been adjusted within θ time, its working set has to be adjusted.

*M. H. Fogel, The VMOS Paging Algorithm: A Practical Implementation of the Working Set Model, SIGOPS, Vol. 8 (1974), pp. 8-17.

Example VMOS: 2/2



Virtual Relocation
Virtual Memory in a VM: 1/4

RM: real machine
VM: virtual machine
VS: virtual memory

VM1's VS

VM1

page-in

RM

a page-in in VM must have its page in RM to be used!

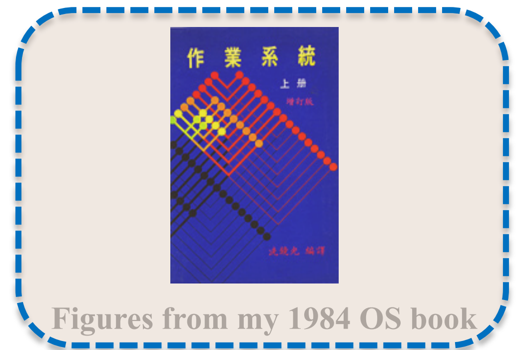
VM2's VS

VM 2

page-in

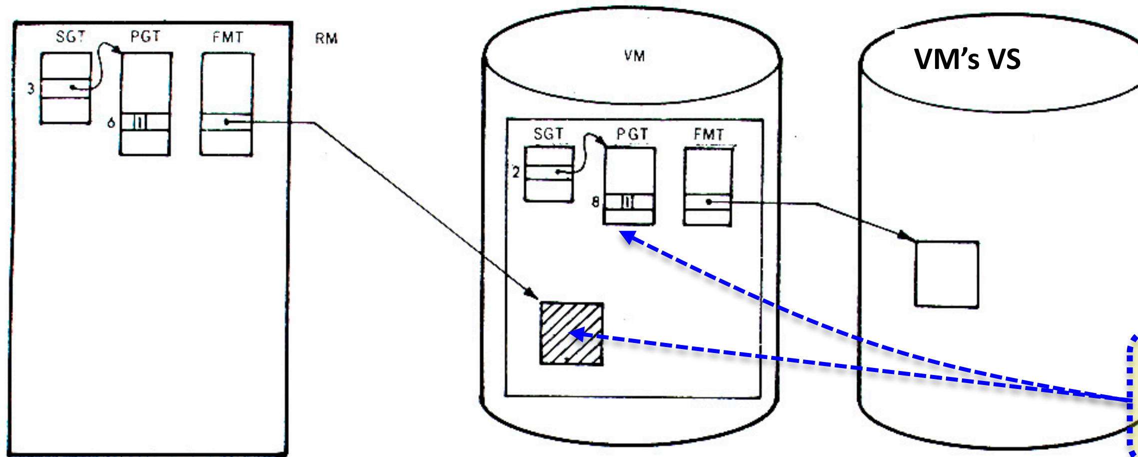
What is the VM supports virtual memory?

A page-in in a VM brings its page into its VS; but, Actually the page should be brought to RM



Figures from my 1984 OS book

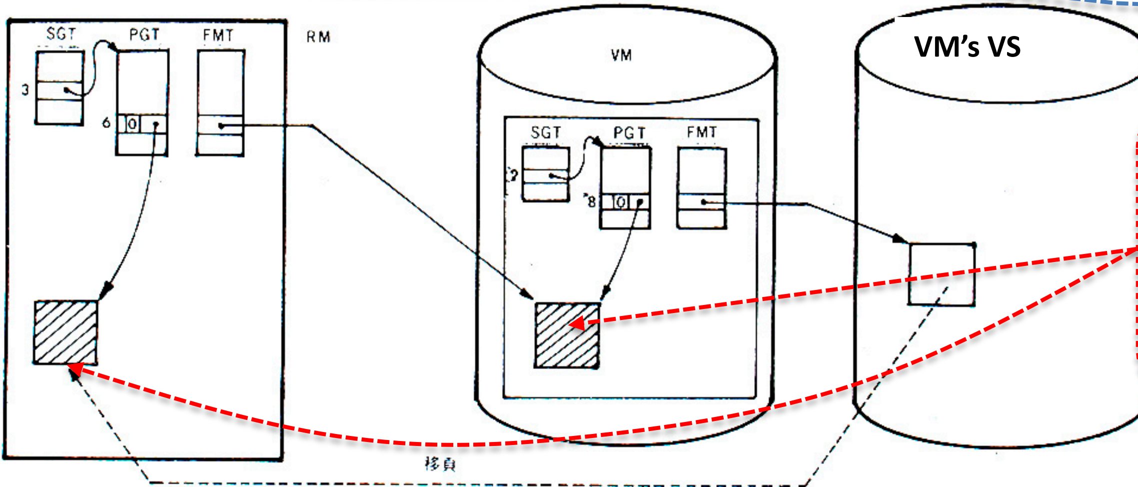
Virtual Relocation Virtual Memory in a VM: 2/4



From RM's point of view, VM is just something in RM's VS.

RM has its segment table SGT, page table (PGT) and page frame table FMT for its own virtual memory management.

Now, VM uses page 8 of segment 2.
Because this page is not in memory,
It has to be paged-in to the shaded frame.

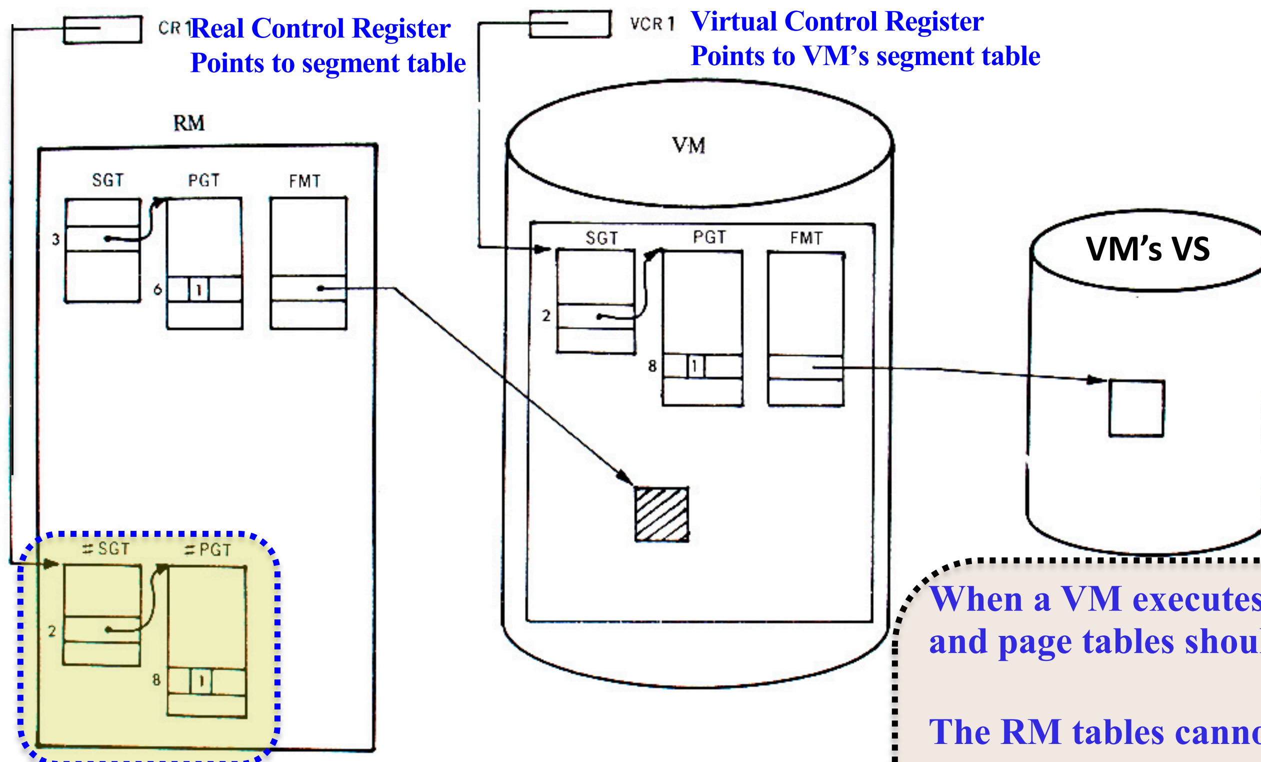


The VM starts page-in, moving the needed page from VM's VS into VM.

But, because this page has to be in RM to be used, the RM allocates a page Frame for this page and modifies RM's page table.

SGT: Segment Table, PGT: Page Table, FMT: Page Frame Table

Virtual Relocation
Virtual Memory in a VM: 3/4



SGT: Segment Table
PGT: Page Table
FMT: Page Frame Table

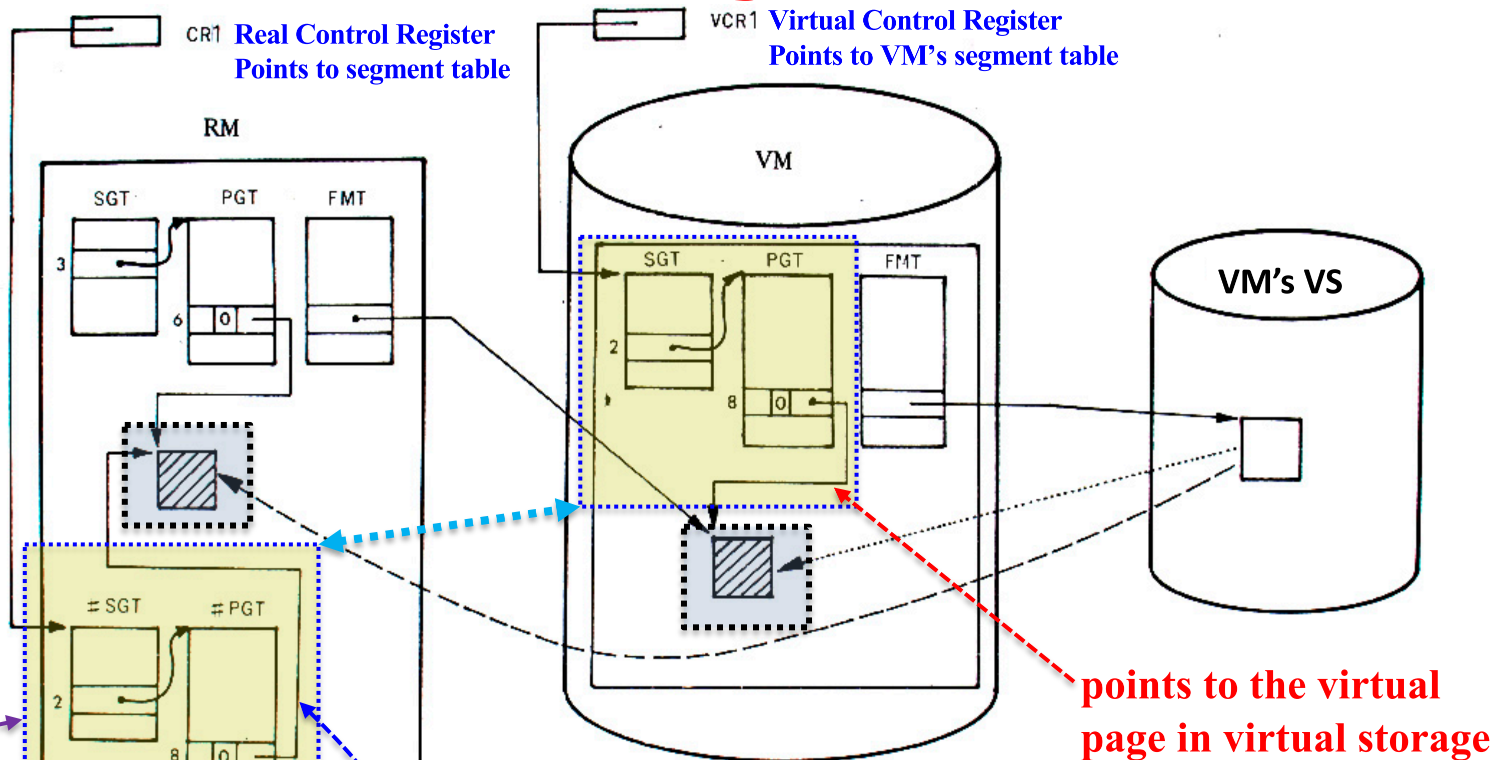
When a VM executes on a RM, which segment table and page tables should be used?

The RM tables cannot be the candidate because they are used to manage the VM.

Tables in VM cannot be used either because that VM runs on RM rather than a VM.

In fact, the RM builds a set of shadow tables for each VM.

Virtual Relocation
Virtual Memory in a VM: 4/4



This set of shadow tables have the "structure" as those in the VM; but, they point to REAL things.

points to the real page in memory

The control program running on RM builds a set of shadow pages Tables that are identical to those in the VM.

But, the page frame numbers are different. The page table in VM points to the page frame in VM, but the shadow page table in RM points to the real page frame.

Virtual Memory Management in Control Program (CP): 1/5

- The control program CP of VM/370 views each virtual machine as a process.
- CP uses a second chance page replacement and working set.
- All page frames are in two lists `FREELIST` and `FLUSHLIST`.
- `FREELIST` has all free page frames.
- If for some reason a VM cannot hold its page frames, all of its page frames are moved to `FLUSHLIST`. However, page tables are not modified, only showing these pages are not available to use.

Virtual Memory Management in Control Program (CP): 2/5

- **If the RM needs a page frame, then …**
  - **Take one from `FREELIST`**
  - **Or, if `FREELIST` has no page frame available, then take one from `FLUSHLIST`.**
  - **Or, if `FLUSHLIST` is also empty, then search the used page frame with the clock algorithm.**
  - **Note that page table entry has to be modified and page-out may be needed.**

Virtual Memory Management in Control Program (CP): 3/5

- A VM May or may not be allowed to get page frames.
- When a VM is allowed to get pages, the memory management MM component in RM monitors paging activity of this VM.
- Once this VM causes a page fault, MM monitors the number of in memory page of this VM, until this VM becomes not allowed to get page frames.
- At this moment, RM calculates the **Average Resident Pages** (**ARP**) of this VM.
- Note that among these page faults, some causes removing of its own page, while the others steal other VM's pages.

Virtual Memory Management in Control Program (CP): 4/5

- **The MM in CP determines the rate of page faults of this VM that requires stealing other VM's pages.**
- **If this rate is larger than 8%, this rate is recorded in *S*. Otherwise, *S* = 0.**
- **During this period (i.e., the time a VM allowed to get pages), let *P* be the average life span: time span in this period divided by the number of page faults.**
- **Let *I* be the global average life span: total CPU time so far divided by the number of page faults.**

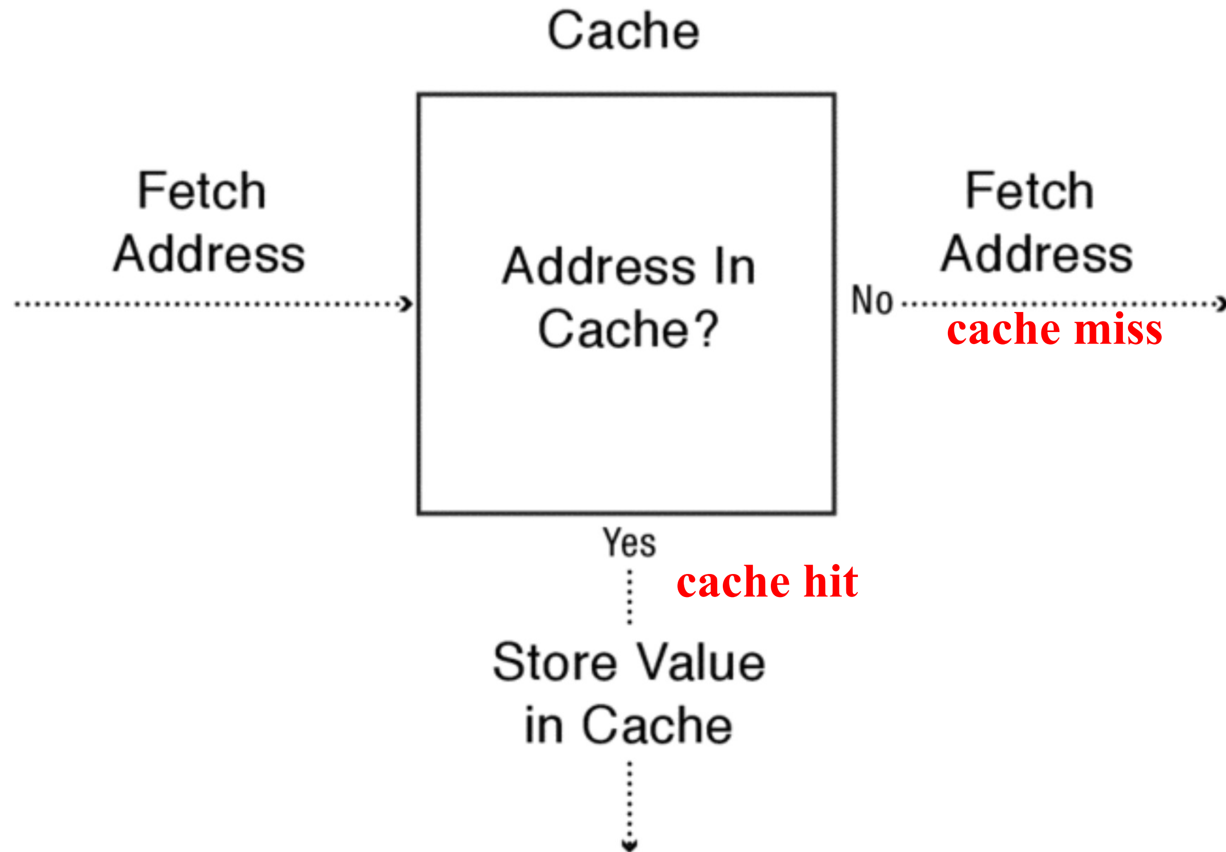
Virtual Memory Management in Control Program (CP): 5/5

- The MM component of CP uses the following to predict the average resident page in the next period:

$$\mathbf{newARP} = \max\left((\mathbf{ARP} + S)\sqrt{\frac{I}{P}}, 5 \right)$$

- This prediction is usually close to the actual working set size except for some odd situations. Note that the **newARP** has at least 5 pages.
- If system performance goes down because of this VM's high page faults ($I > P$ and/or $S > 0$), the new prediction is larger.
- Otherwise, I may be less than P, and hence the new prediction of ARP may be smaller.

Cache Concept (Read)



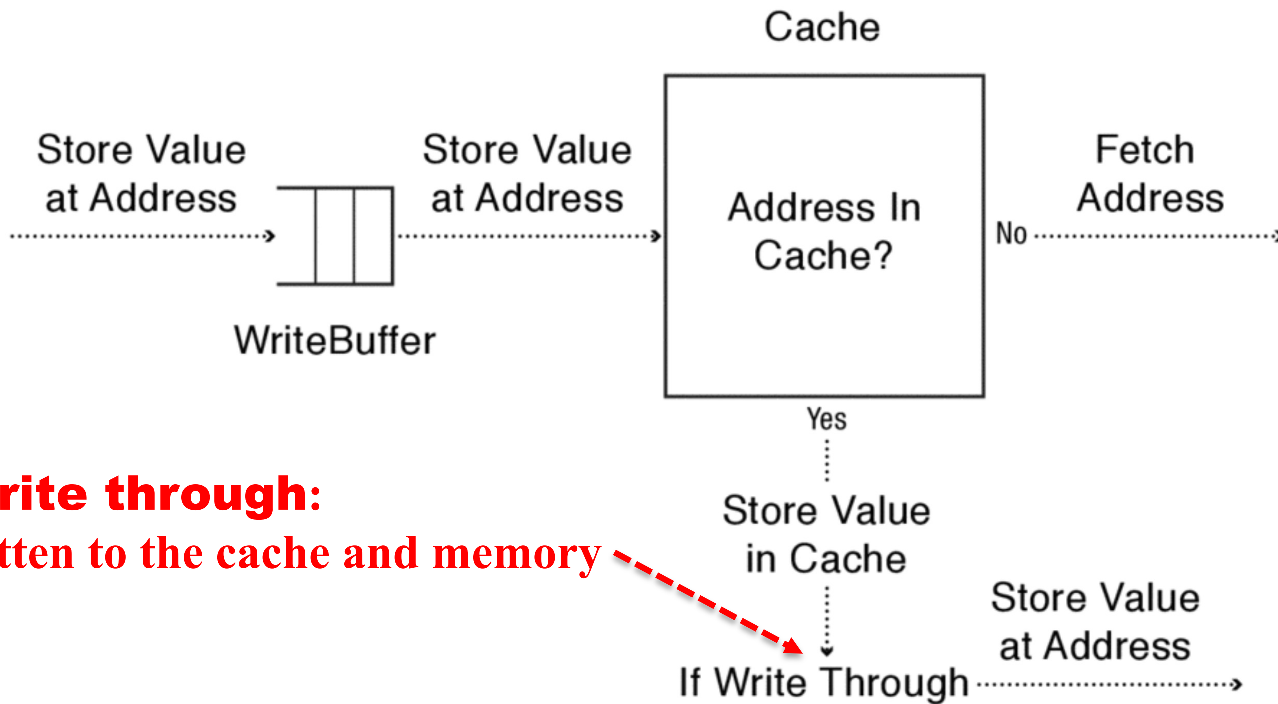
- Memory read requests are sent to the cache
- The cache either returns the value stored at that memory location, or it forwards the request onward to the next level cache

Memory Hierarchy

Cache	Hit Cost	Size
1st level cache/first level TLB	1 ns	64 KB
2nd level cache/second level TLB	4 ns	256 KB
3rd level cache	12 ns	2 MB
Memory (DRAM)	100 ns	10 GB
Data center memory (DRAM)	100 μs	100 TB
Local non-volatile memory	100 μs	100 GB
Local disk	10 ms	1 TB
Data center disk	10 ms	100 PB
Remote data center disk	200 ms	1 XB

i7 has 8MB as shared 3rd level cache; 2nd level cache is per-core

Cache Concept (Write)

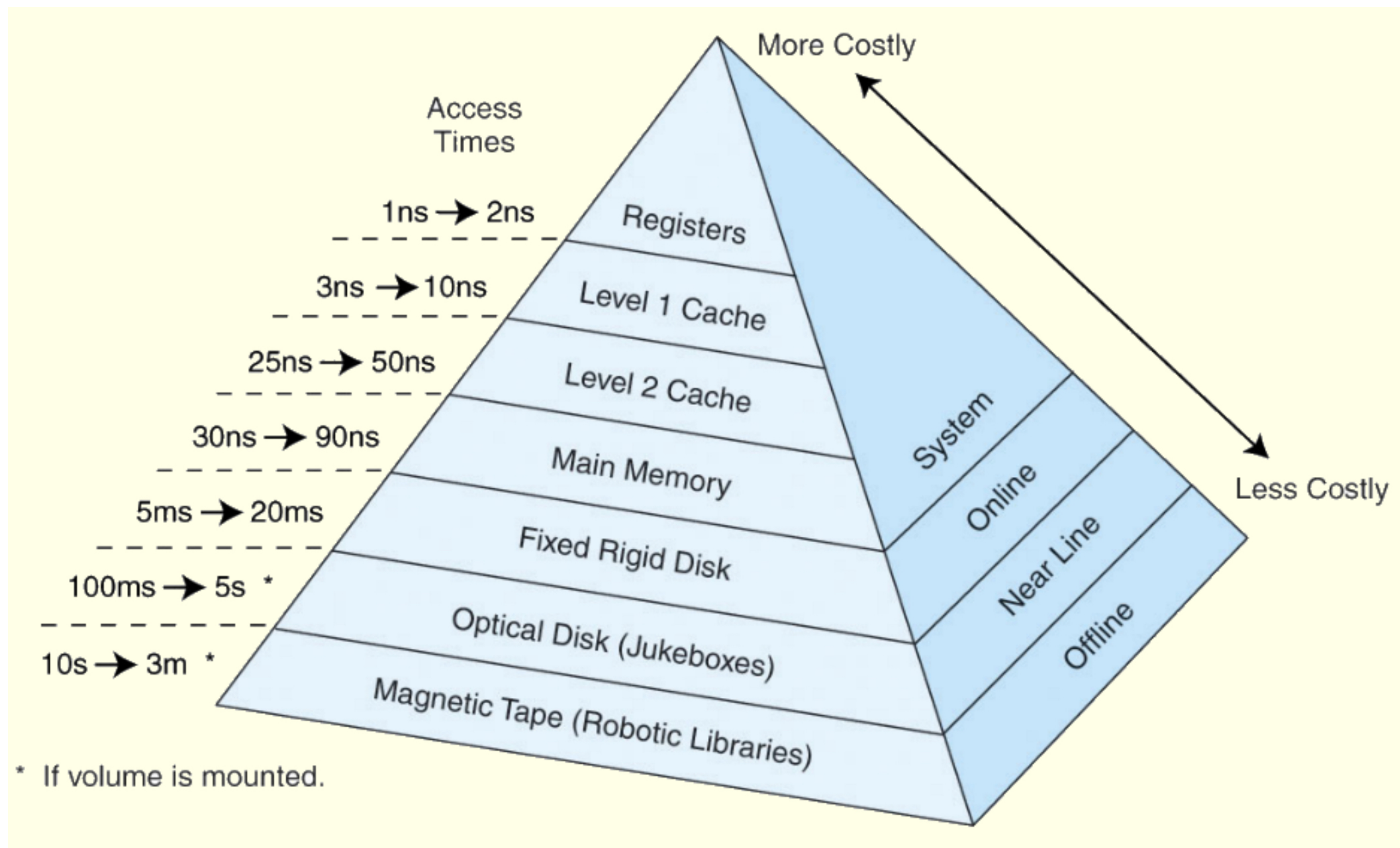


Cache write through:
Data is written to the cache and memory

- Memory requests are buffered and then sent to the cache in the background
- Typically, the cache stores a block of data, so each write ensures that the rest of the block is in the cache before updating the cache
- If the cache is write through, the data is then sent onward to the next level of cache or memory.

Memory Hierarchy

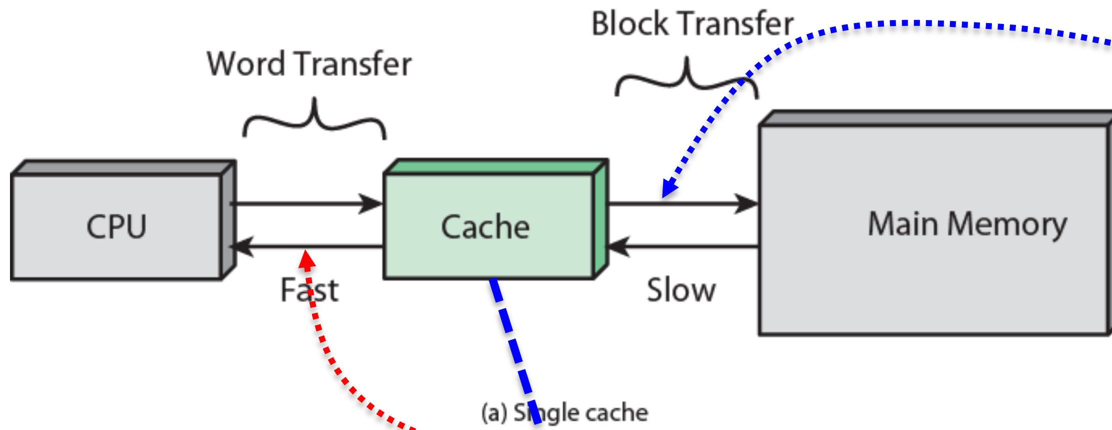
- Cache memory can be between CPU and memory, external device and memory, etc.

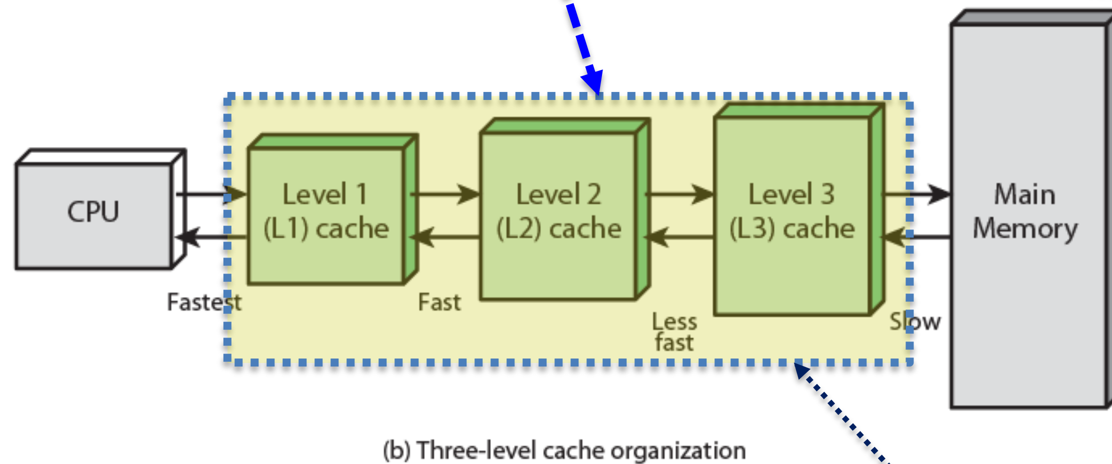


Cache Memory: 1/12

- It is possible to build a computer using only static RAM.
- This would be very fast, but the cost can be very high.
- During the course of the execution of a problem, memory references tend to cluster (e.g., loops).
- Thus, we only need a small amount of fast memory between physical memory and CPU, or even on CPU or module.

Cache Memory: 2/12



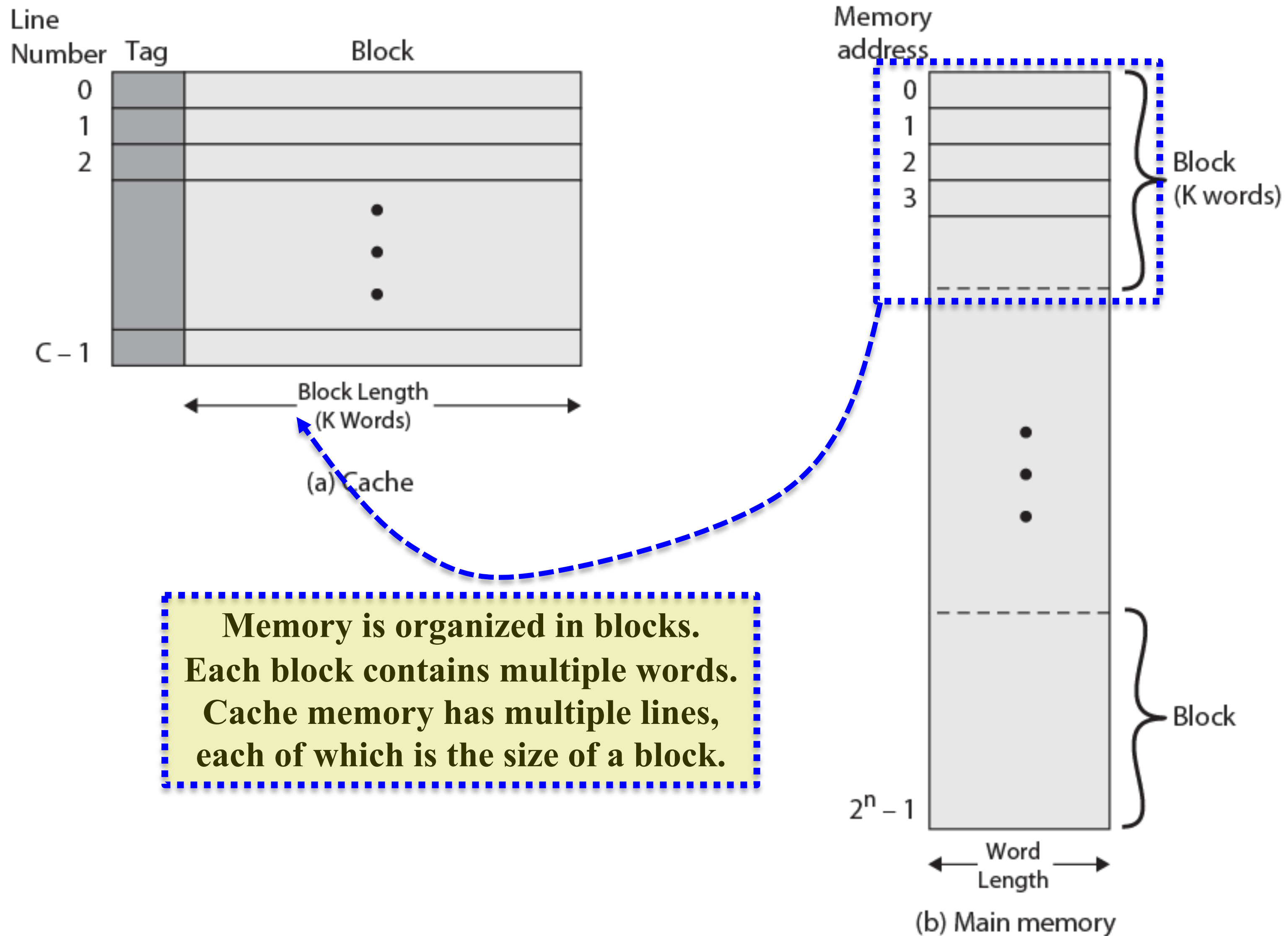


From memory to cache is block based, more than a byte/word.

From cache to CPU is perhaps word/byte based.

cache may have multiple levels

Cache Memory: 3/12

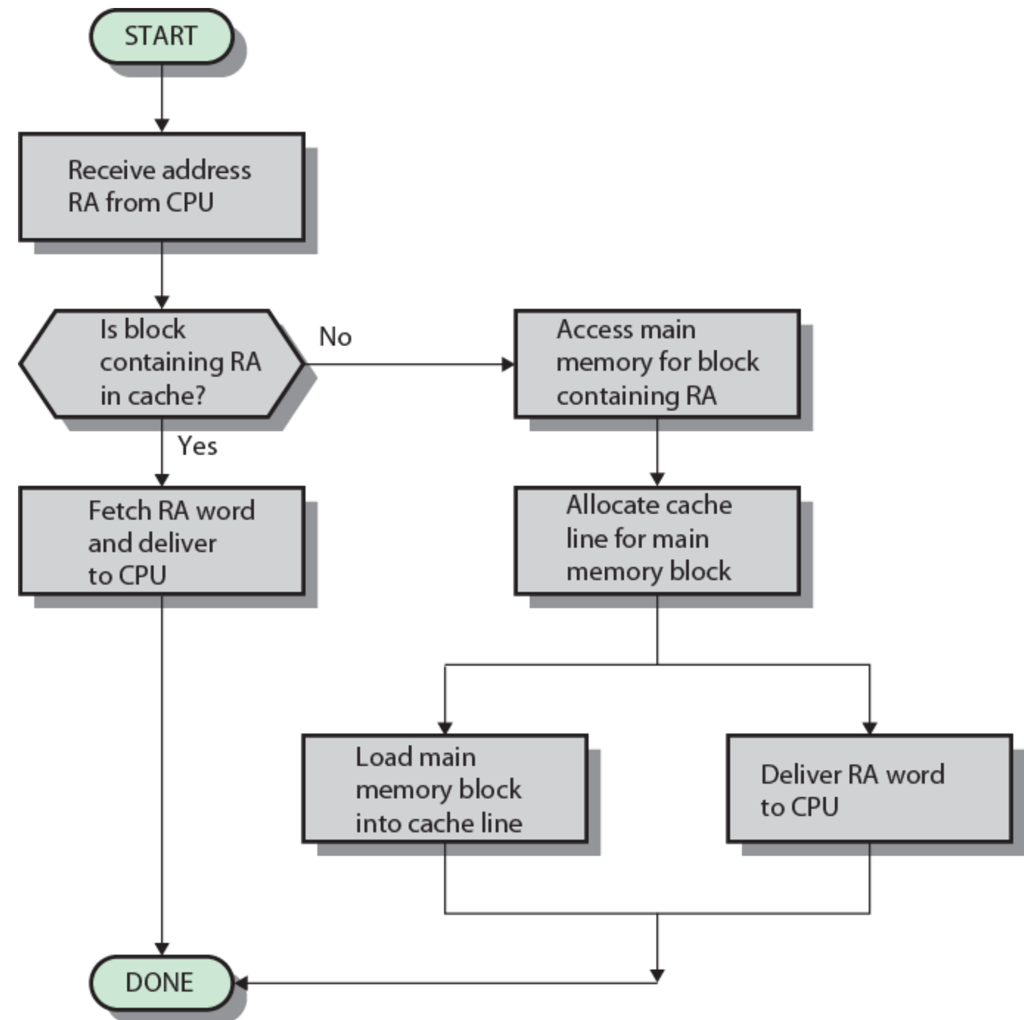


Cache Memory: 4/12

- Cache read Operation.

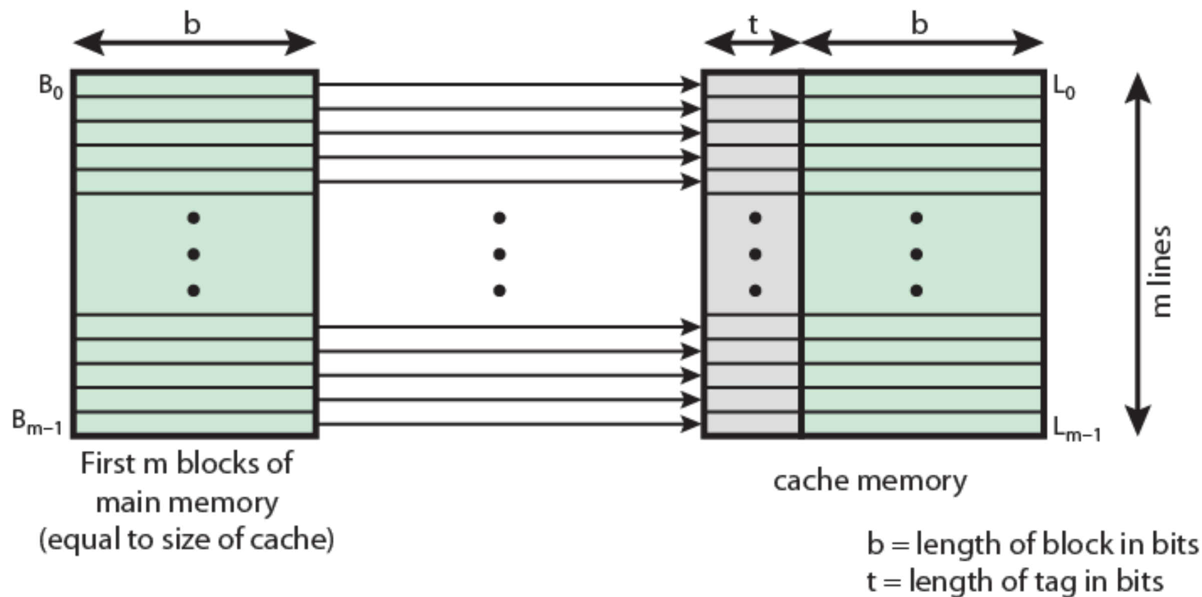
Issues in cache design:

1. Addressing
2. Size
3. Mapping
4. Replacement
5. Write policy
6. Block size
7. Number of caches



Cache Memory: 5/12

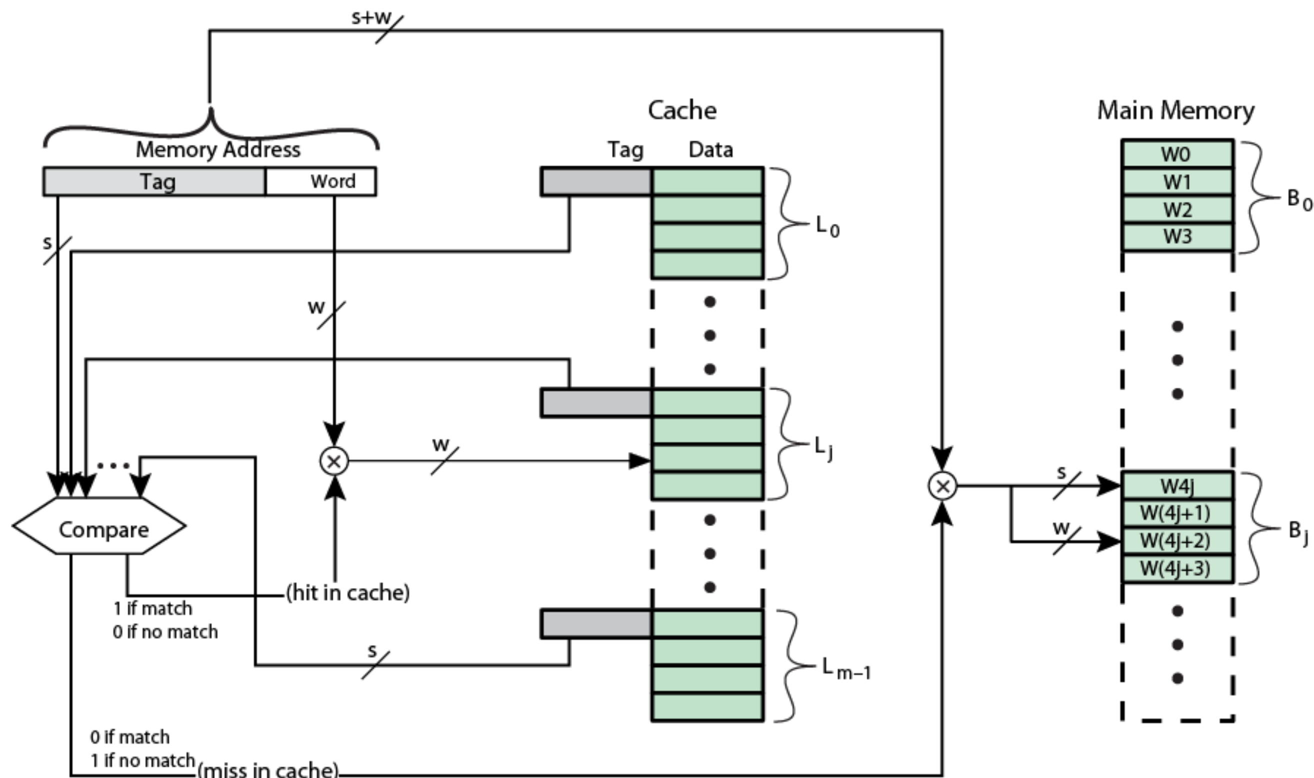
- Direct mapping, associative mapping and set associative mapping.
- Each block of physical memory maps to only one cache line.



(a) Direct mapping

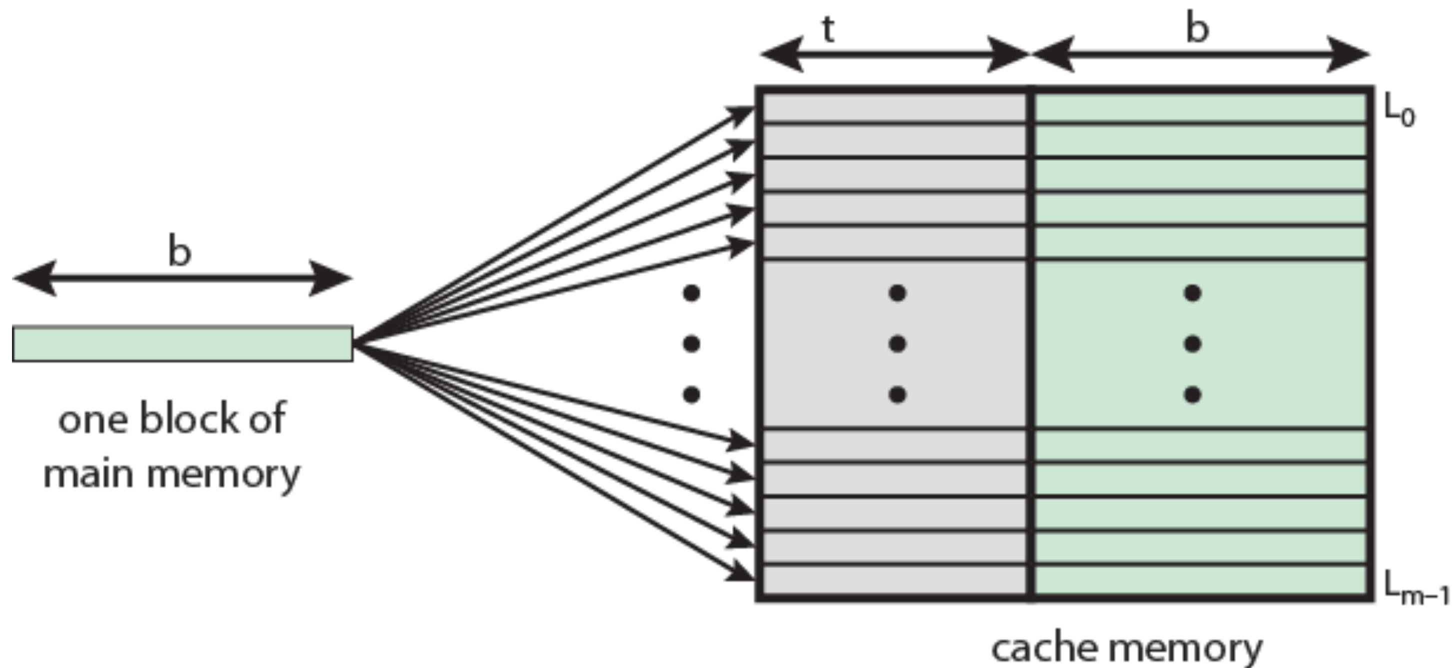
Cache Memory: 6/12

- Use the tag field to compare whether a block is in cache. If the block is in cache, we have a cache hit!
- If a miss, load the block to cache. Only one choice!



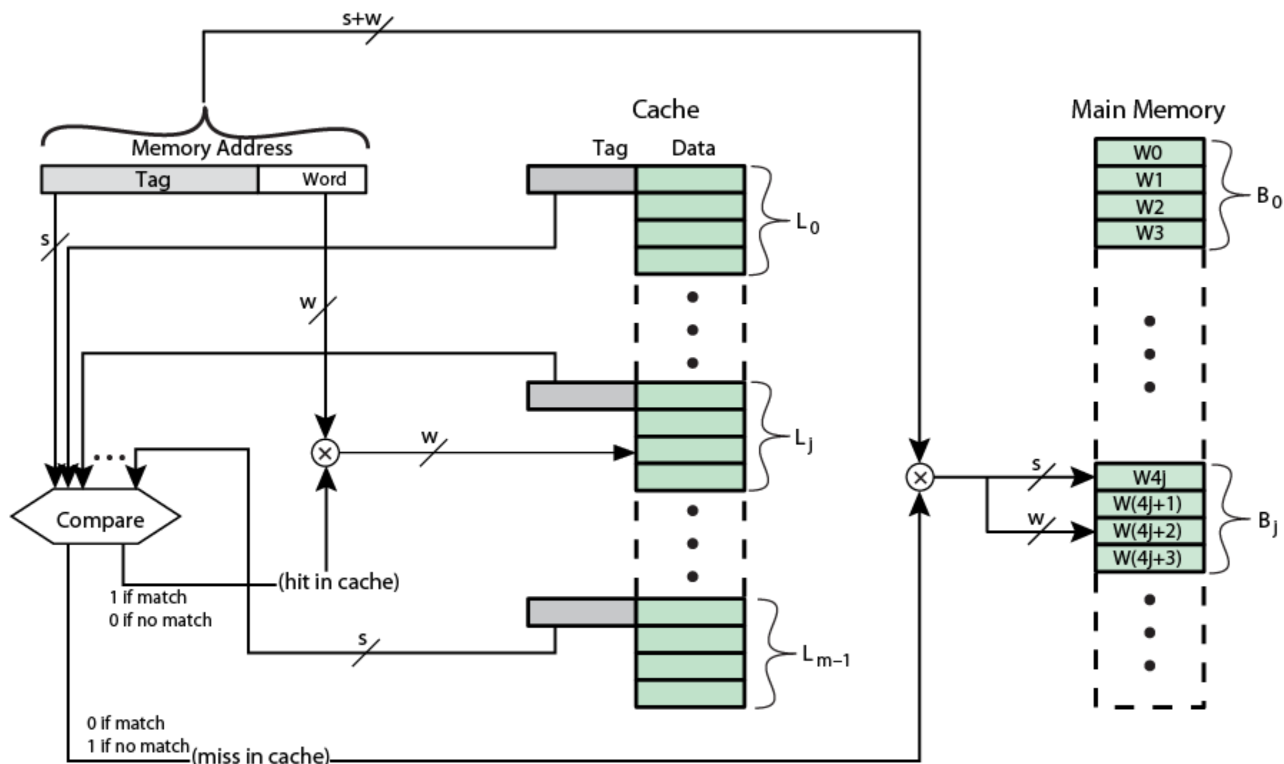
Cache Memory: 7/12

- Associative mapping: A physical memory block can be loaded into any line of cache.
- Memory address is interpreted as tab and word.
- Tag uniquely identifies block of memory.



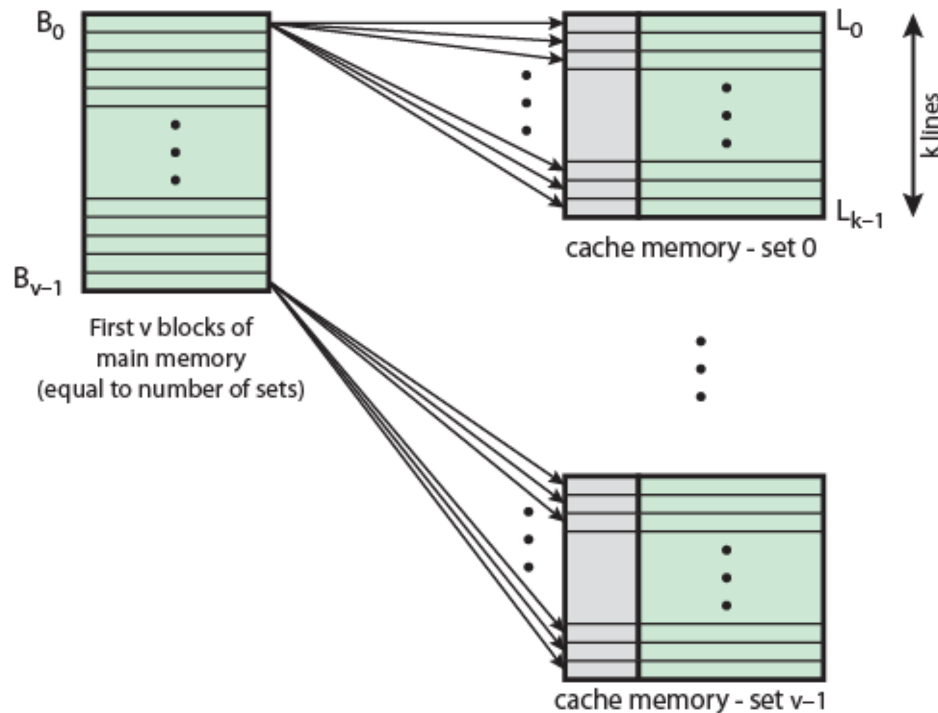
Cache Memory: 8/12

- Associative mapping: A physical memory block can be loaded into any line of cache.
- LRU is usually used to find a victim for the new block.



Cache Memory: 9/12

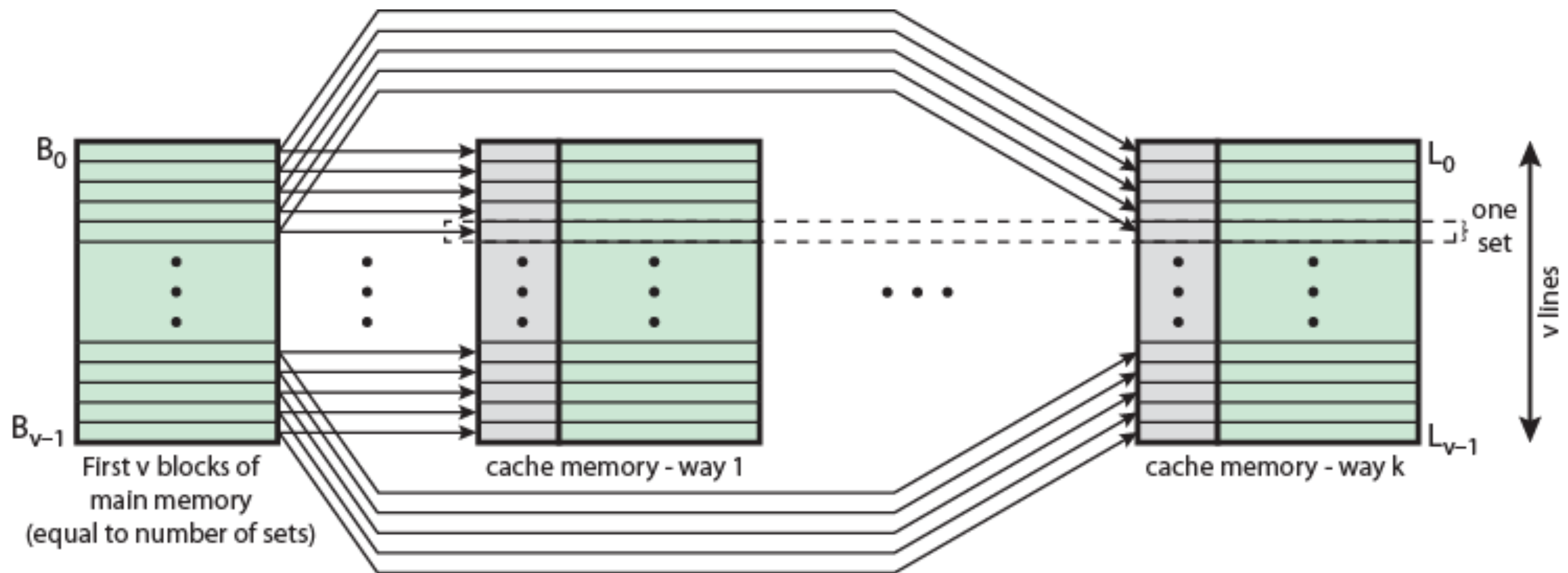
- Set Associative mapping: Cache is divided into a number of sets, each of which contains a number of lines.



(a) v associative-mapped caches

Cache Memory: 10/12

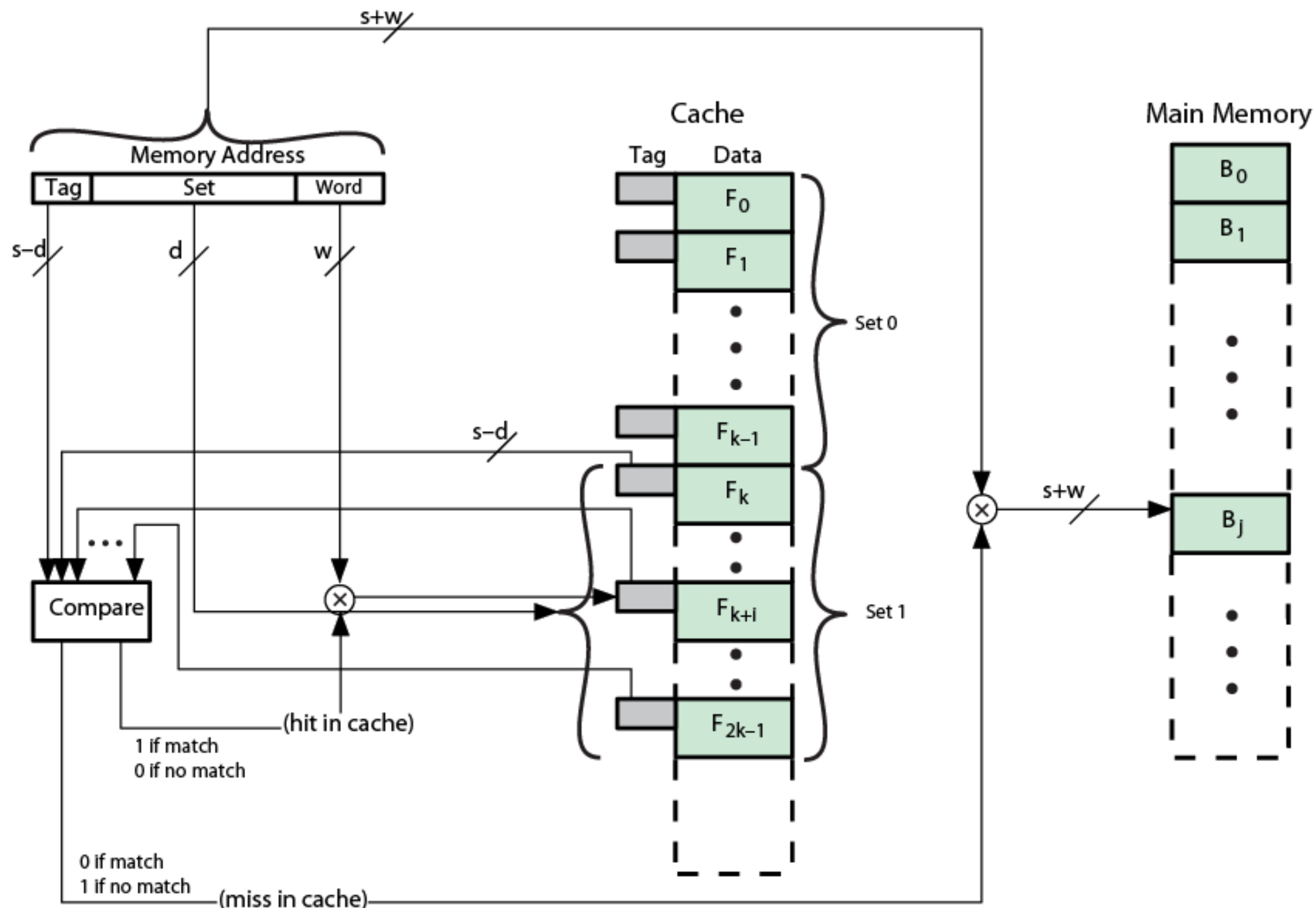
- ***k*-way Set Associative mapping:**



(b) k direct-mapped caches

Cache Memory: 11/12

- *k*-way Set Associative mapping:



Cache Memory: 12/12

- Replacement Algorithms:
  - **Direct Mapping**: There is no choice. Because each block maps to one line, a miss always replace that line.
  - **Associative and Set Associative Mapping**: Random, FIFO, LRU (e.g., in a 2-way set associative, it is easy to find the LRU), LFU (i.e., Least Frequently Used – replacing block which has had fewest hits).

The End