

Operating System Principles

操作系统原理

Memory Management

李旭东

leexudong@nankai.edu.cn

Nankai University



Objectives

- ~~*No Memory Abstraction*~~
- ~~*Basic Memory Management*~~
- Virtual Memory Management
- Page Replacement Algorithms
- Design Issues for Paging Systems
- Implementation Issues
- More topics



Memory Management

- Memory is an important resource that must be carefully managed

While the average home computer nowadays has a thousand times as much memory as the IBM 7094, the largest computer in the world in the early 1960's

- Memory hierarchy

- *Volatile* cache memory

a small amount, very fast, expensive

- Volatile main memory(RAM)

tens of megabytes, medium-speed, medium-price

- Nonvolatile disk storage

tens or hundreds of gigabytes, slow, cheap



Limitation of Basic Memory Management

- Execute a program that is only fully in memory
 - If the size of a program is larger than physical memory, the program cannot be executed
 - If the total size of many programs ...
- How to extend memory?
 - Physical way
 - Logical way

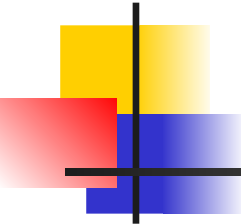


?Whether it is necessary that
execute a program that is only
fully in memory



Principle of locality

- Principle of locality
 - locality of reference
 - 局部性原理 , 1972, Denning
 - P.J. Denning, The Locality Principle, Communications of the ACM, Volume 48, Issue 7, (2005), Pages 19–24
 - a phenomenon describing the same value, or related storage locations, being frequently accessed
 - cache
 - instruction prefetch
- catalog
 - Temporal Locality 时间局部性
 - Spatial Locality 空间局部性



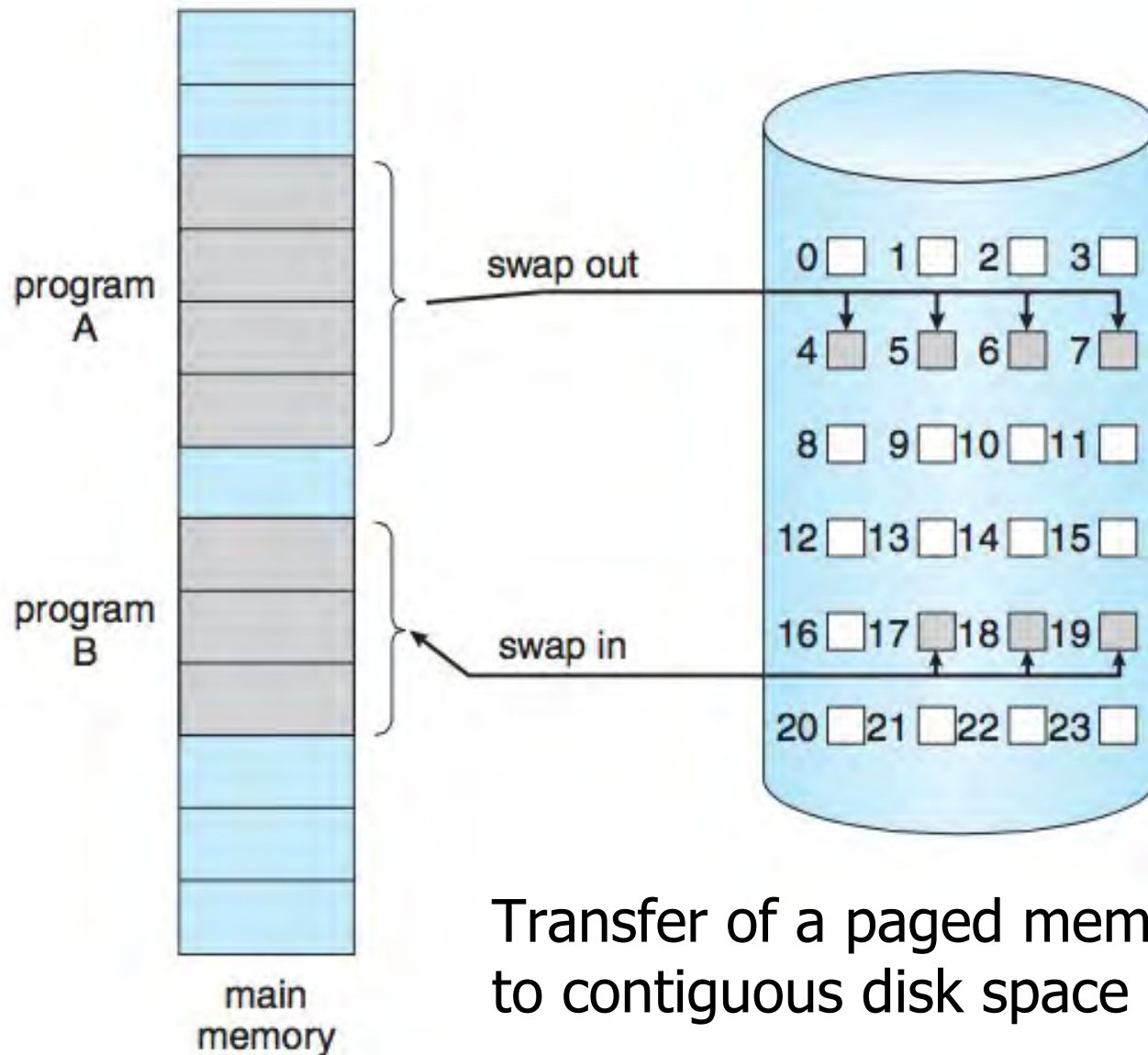
?Whether it has ability to execute
a program that is only partially in
memory



Demand Paging

- Demand Paging 请求分页
 - With demand-paged virtual memory, pages are loaded only when they are demanded during program execution
 - Pages that are never accessed are thus never loaded into physical memory
 - A virtual memory system
- Swapper
 - A lazy swapper never swaps a page into memory unless that page will be needed

Demand Paging





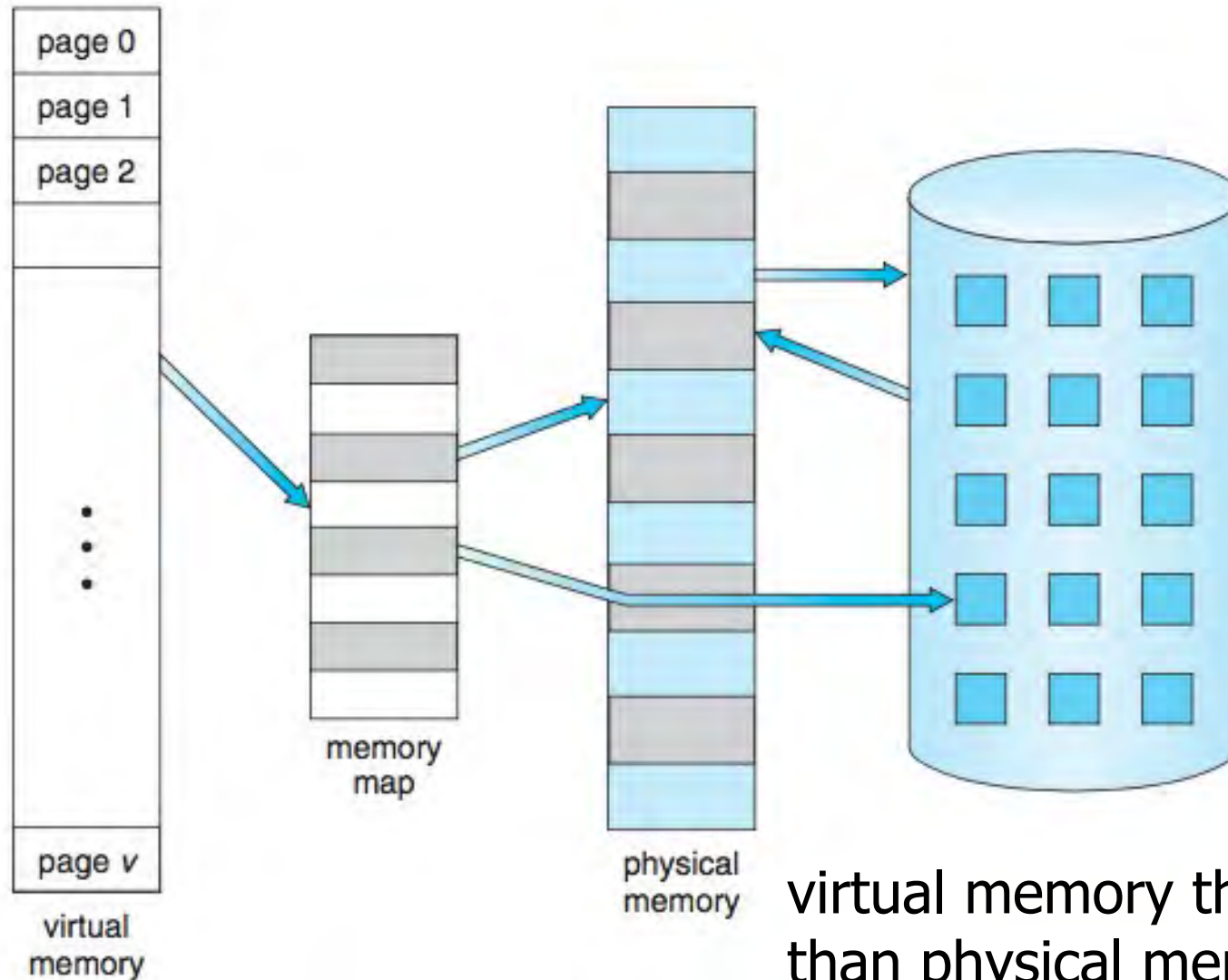
Smarter Paging

- Prepaging
 - Fetching pages in advance



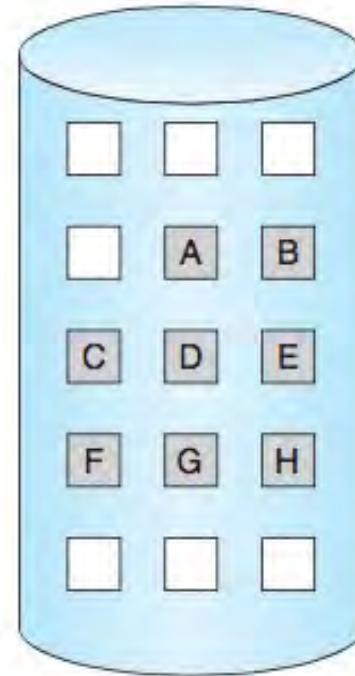
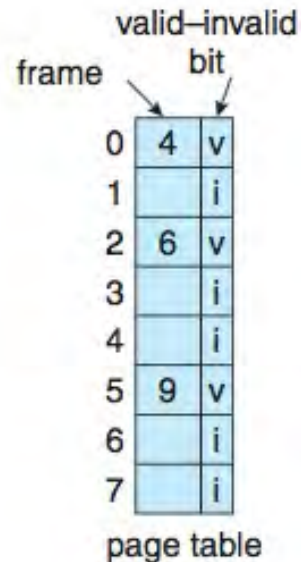
Virtual Memory Management

Virtual Address Space



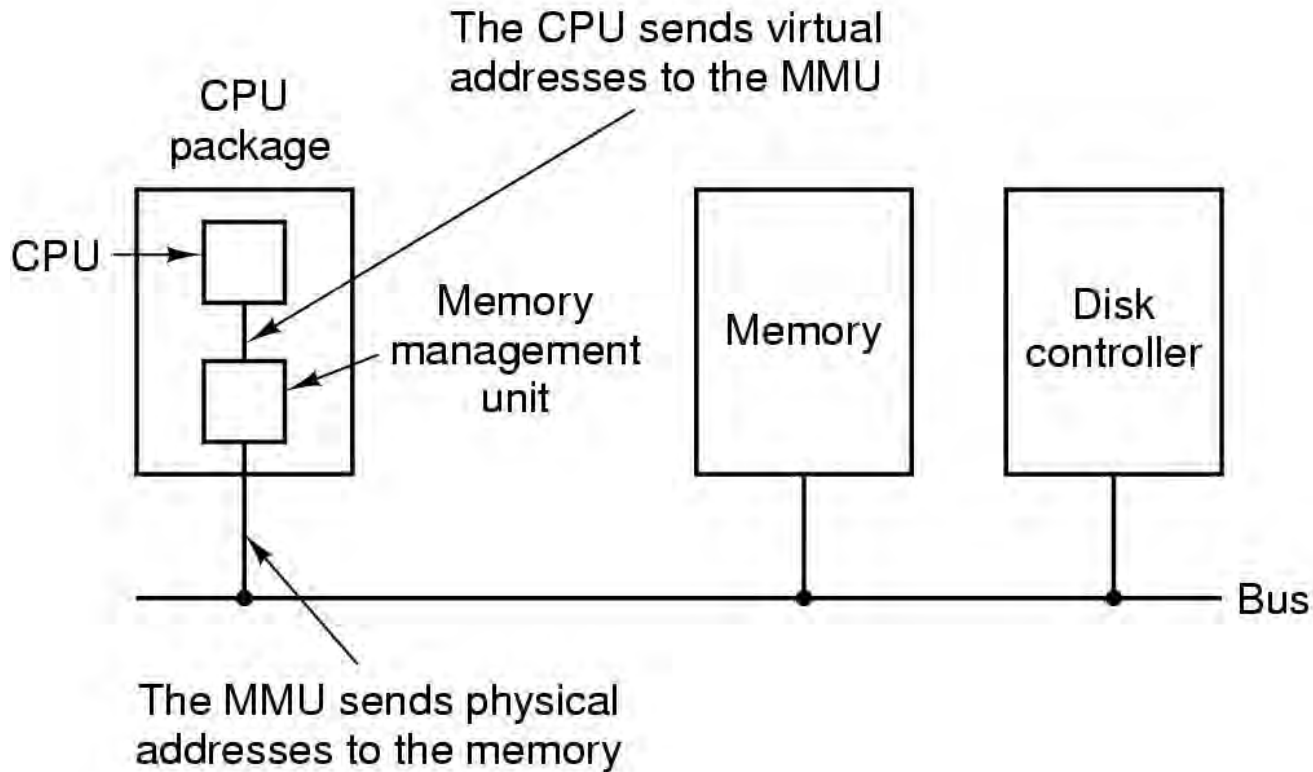
virtual memory that is larger
than physical memory

Memory Resident

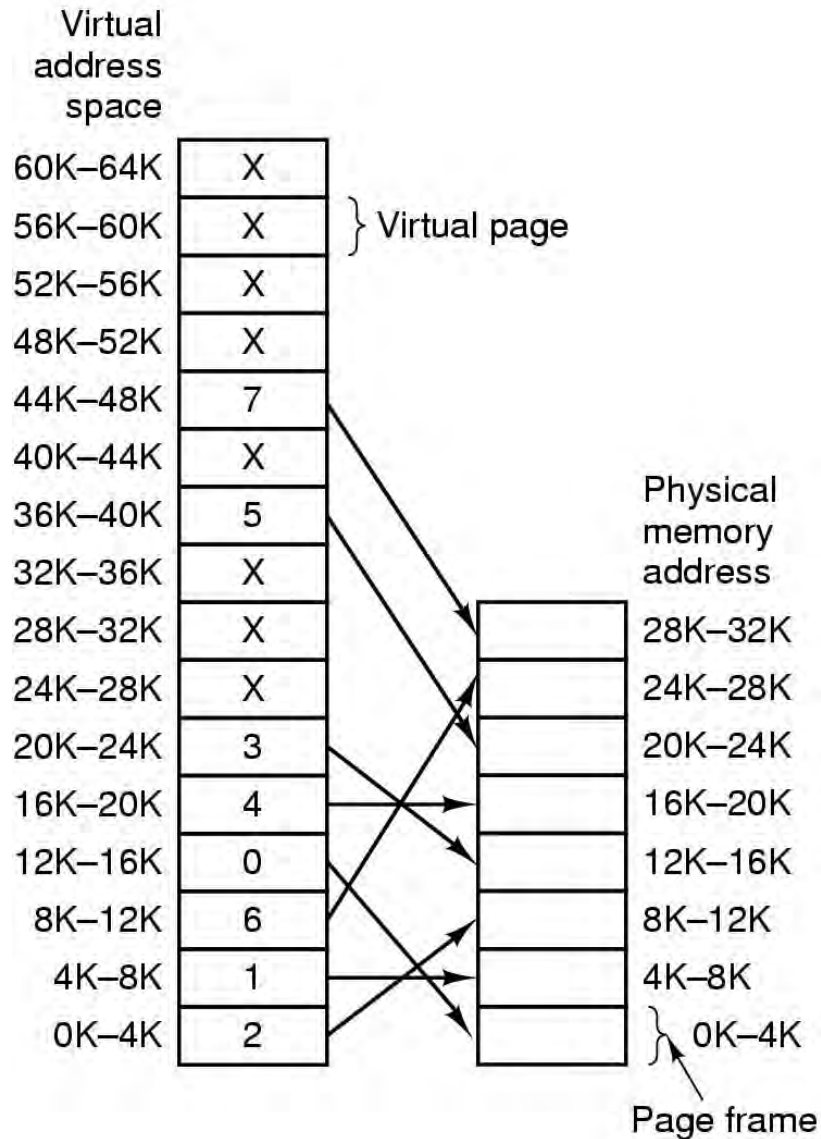


Page table when some pages are not in main memory

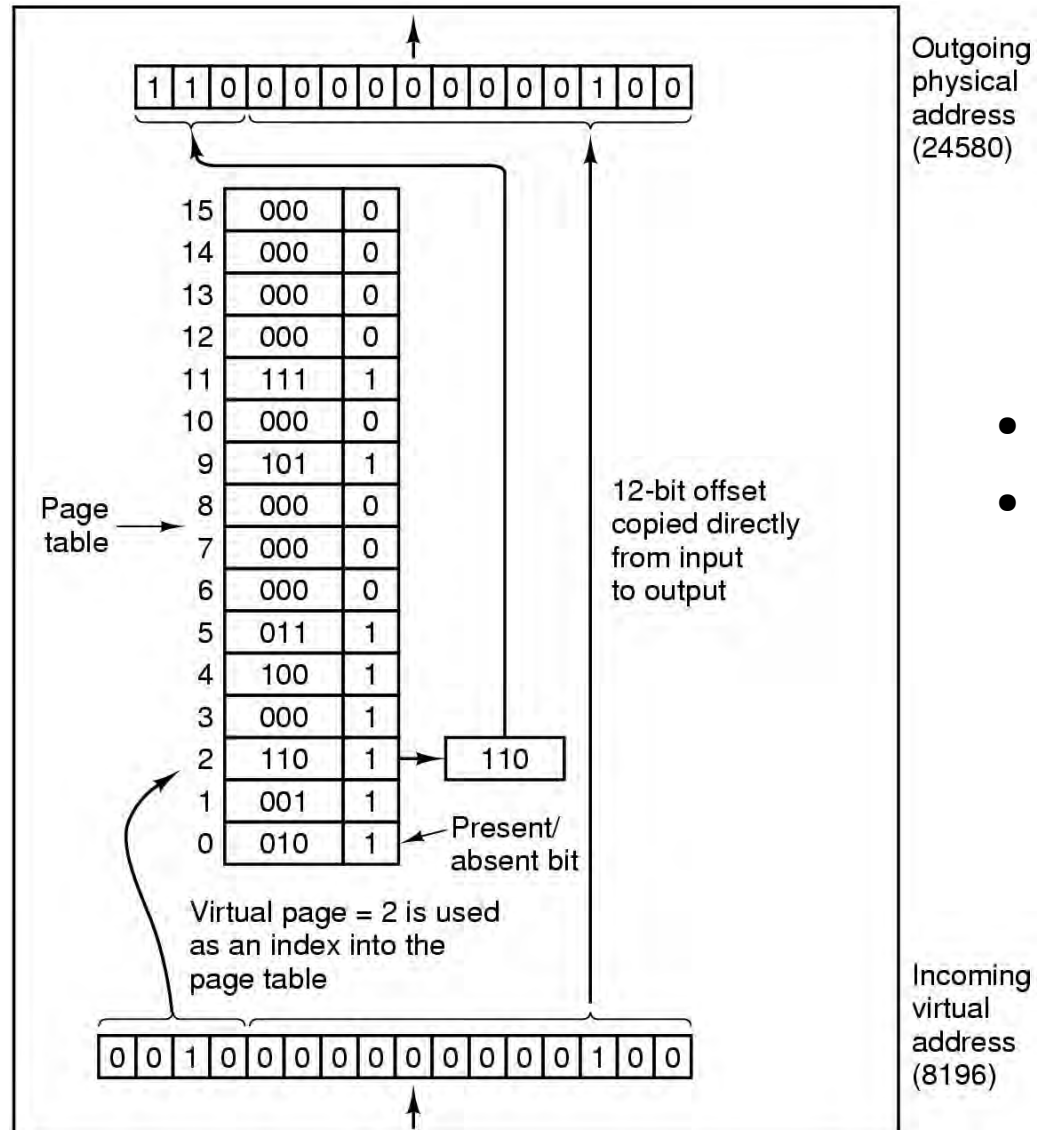
MMU: Memory Management Unit



MMU: Page Table

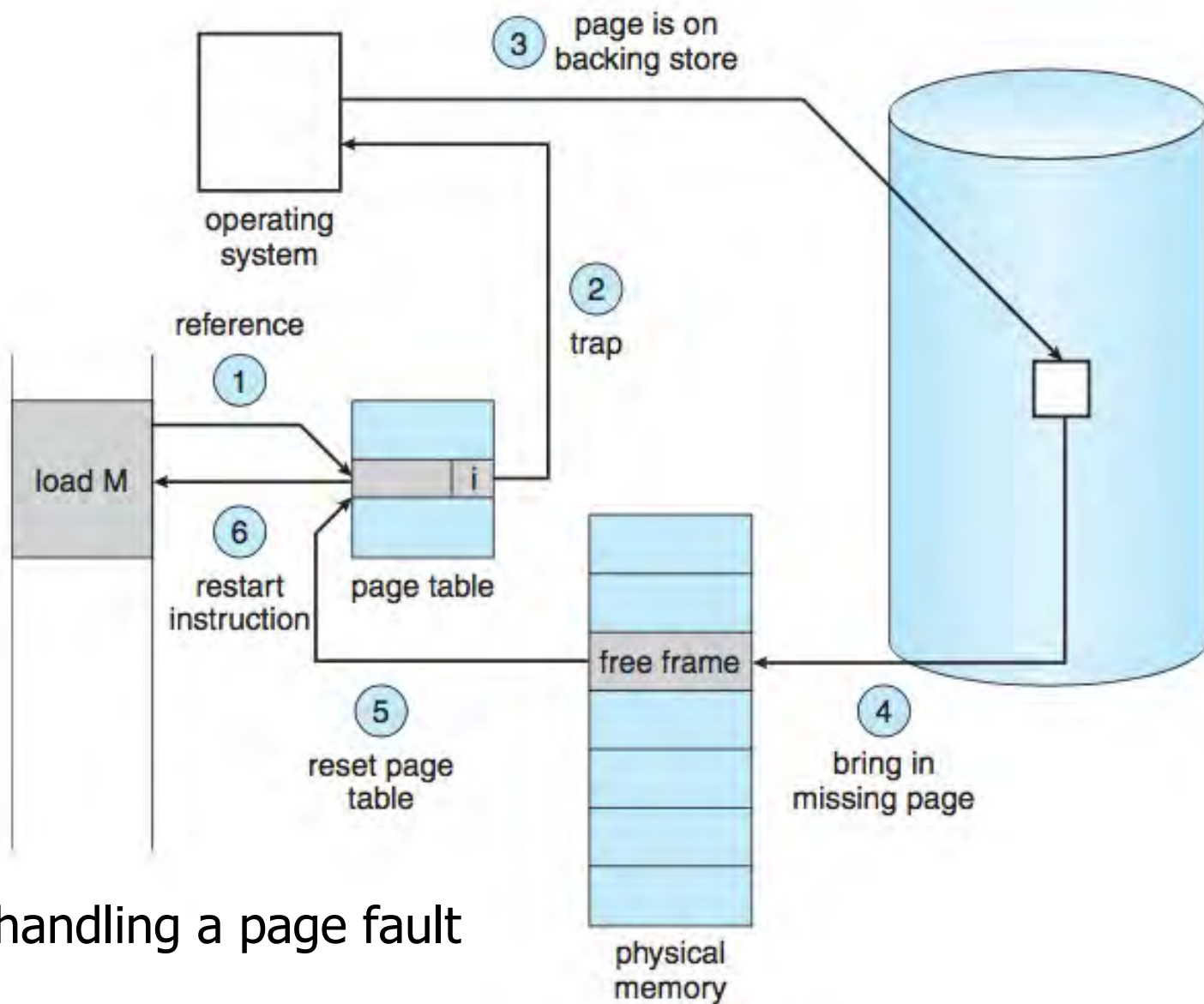


Page Table: State



- Present
- Absent

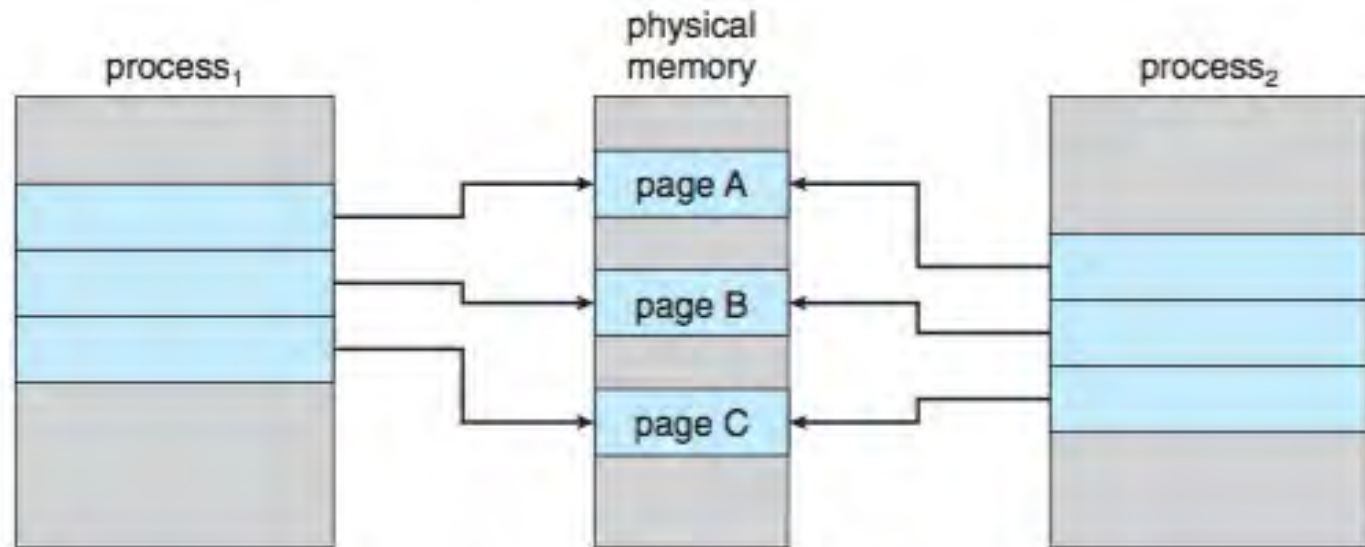
Page Fault



Steps in handling a page fault

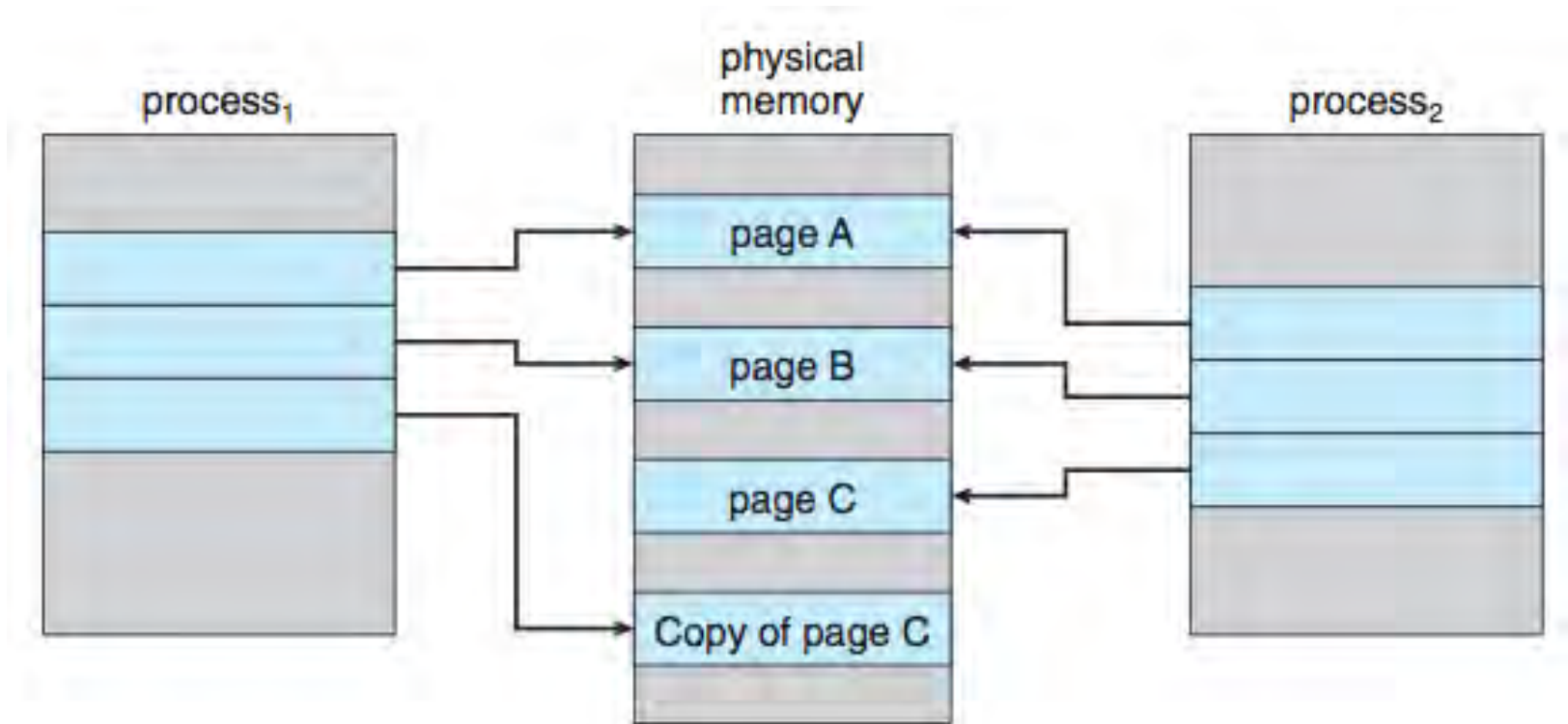
Copy-on-Write

- This technique provides rapid process creation and minimizes the number of new pages that must be allocated to the newly created process.
- These shared pages are marked as copy-on-write pages, meaning that if either process writes to a shared page, a copy of the shared page is created



Before process 1 modifies page C

Copy-on-Write



After process 1 modifies page C



Page Allocation 1/2

- Minimum Number of Frames
 - ensure the normal execution of the process
- Page Allocation Policy
 - Equal, Proportion, Priority
- Page Replacement Policy
 - Fixed Allocation, Local Replacement
 - Variable Allocation, Global Replacement
 - Variable Allocation, Local Replacement

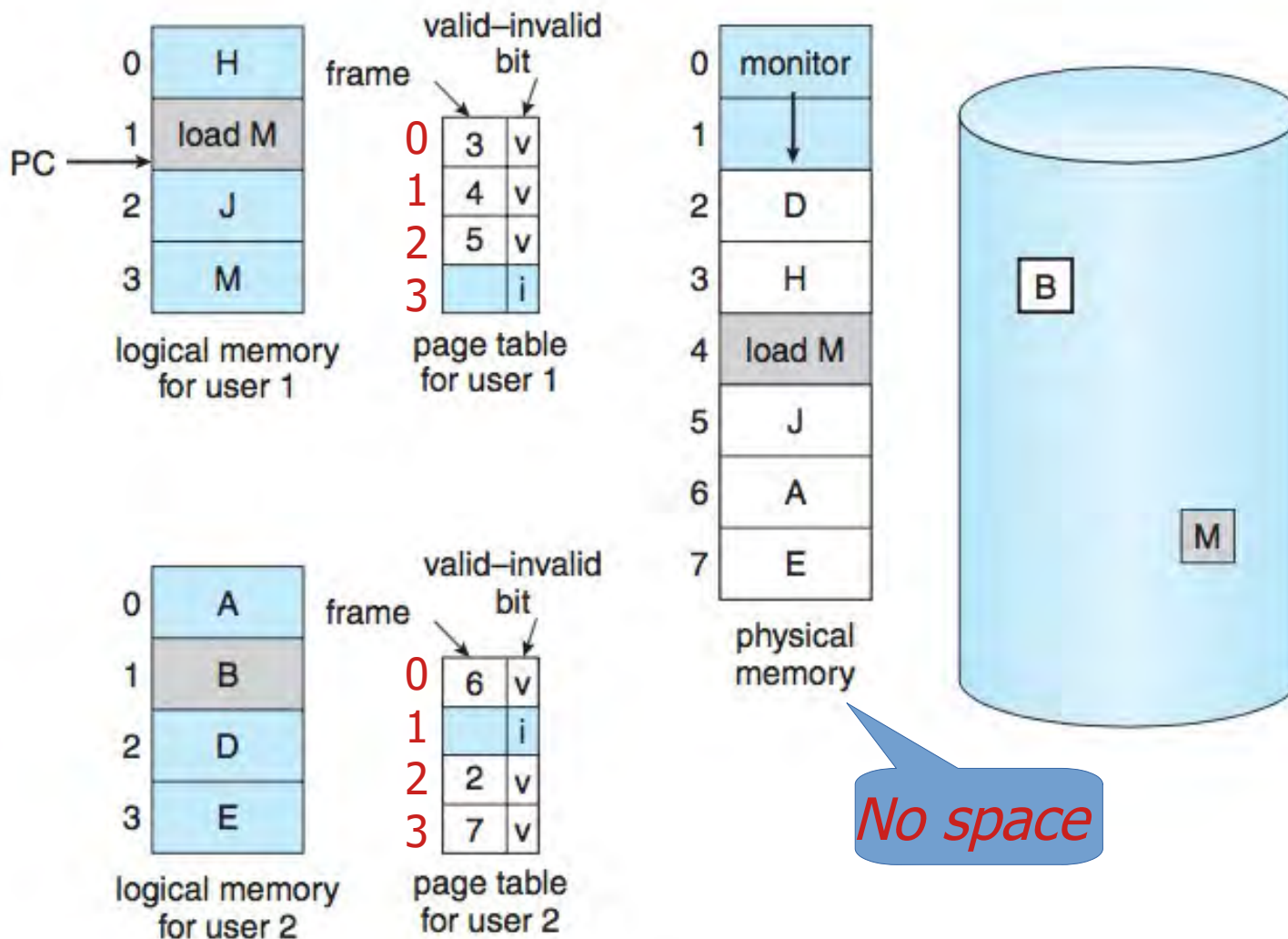


Page Allocation 2/2

- When to Load Page
 - Prefetch 预取
 - On-Demand 按需
- Page Replacement Algorithms*

Page Replacement Algorithms

- Over-allocating memory : 过度分配



Optimal Replacement Algorithm

■ Optimal Replacement Algorithm

- 1966, Belady
- Replace the page that will not be used for *the longest period* of time

the optimal page-replacement algorithm is difficult to implement, because it requires future knowledge of the reference string

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Page ID	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
Reference string																				


Frame	7	7	7	2	2	2	2	2	2	2	2	2	2	2	2	2	2	7	0	1
		0	0	0	0	4		0		0		0		0						
			1	1	3	3		3		1		1		1						

times?

FIFO Replacement Algorithm

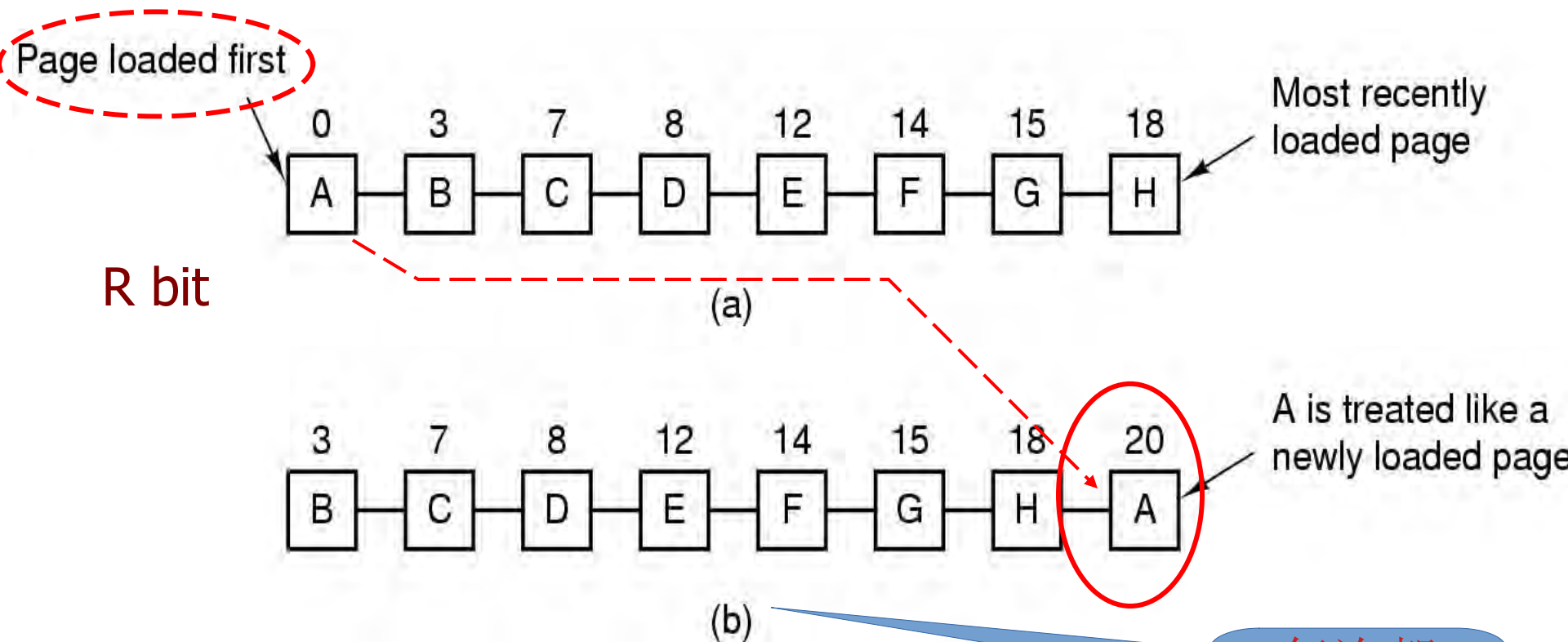
■ FIFO

- When a page must be replaced, the oldest page is chosen
- the simplest page-replacement algorithm
- its performance is not always good



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Page ID	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
Reference string																				
Frame	7	7	7	2		2	2	4	4	4	0			0	0			7	7	7
		0	0	0		3	3	3	2	2	2			1	1			1	0	0
			1	1		1	0	0	0	3	3			3	2			2	2	1

Second Chance Page Replacement Algorithm

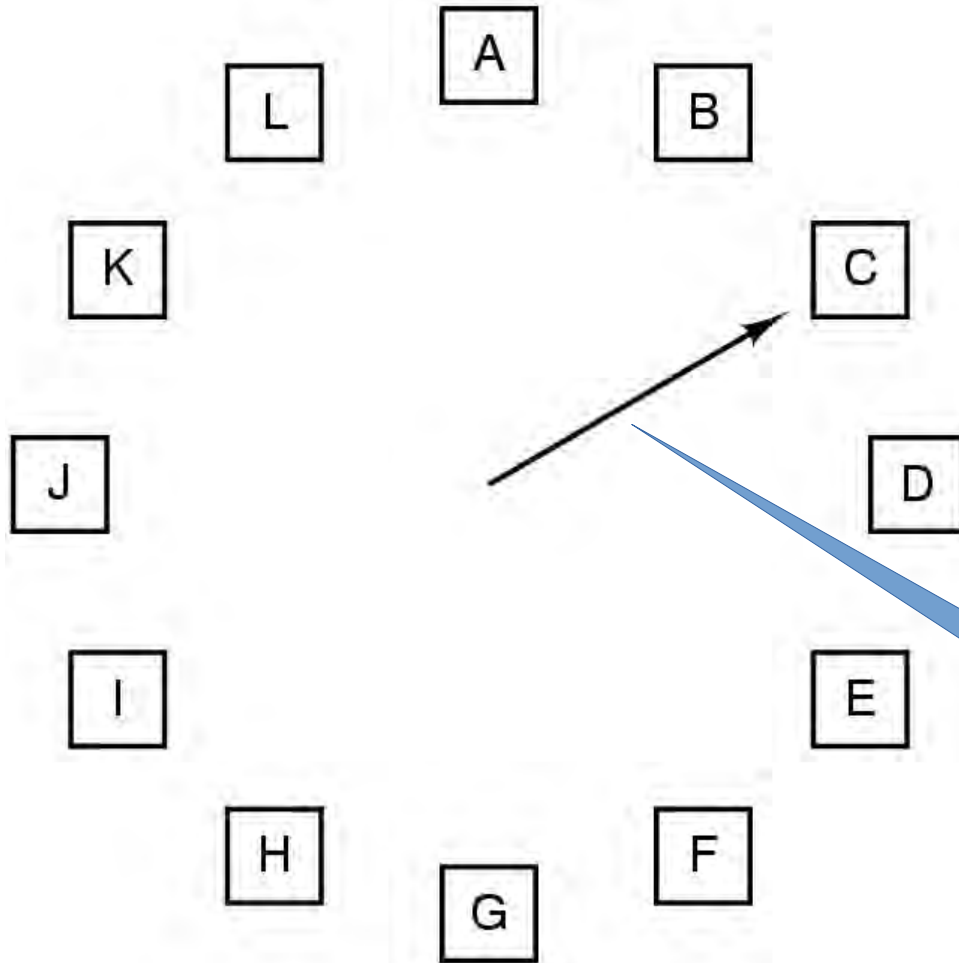


■ Operation of a second chance

- pages sorted in **FIFO** order
- Page list if fault occurs at time 20, A has R bit set (numbers above pages are loading times)
- Page list header: Fixed

每次都
从头开始

(Simple) Clock Page Replacement Algorithm



R bit

NRU(Not Recently Used)
: 最近未使用算法

When a page fault occurs,
the page the hand is
pointing to is inspected.
The action taken depends
on the R bit:

R = 0: Evict the page

R = 1: Clear R and advance hand

区别于
二次机会算法

LRU Replacement Algorithm

- **Least Recently Used:** 最近久未使用
 - we use the recent past as an approximation of the near future, then we can replace the page that has not been used for the longest period of time
 - Counter: we associate with each page-table entry a time-of-use field and add to the CPU a logical clock or counter.
- **Do not have to consider before the last visit**

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Page ID	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
Reference string																				
Frame	7	7	7	2		2		4	4	4	0			1		1		1		
		0	0	0		0		0	0	3	3			3		0		0		
			1	1		3		3	2	2	2			2		2		7		

Times?

LRU Replacement Algorithm

- How to Implement LRU
 - Theoretically realizable, But it is *not cheap*
 - Maintain a linked list of all pages in memory
 - 知道所有页面访问的先后顺序
 - With the most recently used page at the front
 - And the least recently used page at the rear
 - The difficulty is that the list must be updated on every memory reference
 - Implement LRU with special hardware
 - Simulating LRU in software

LRU 1 -

LRU with Hardware : Counter

- Counter based on hardware
 - 64-bit counter
 - Automatically incremented after each instruction
 - Each page table entry contains the counter variable
 - After each memory reference, the current value of Counter is stored in the page table entry
 - A queue which is sorted by the counter variable in page table entry

LRU 2 -

LRU with Hardware: Matrix

- Matrix (*the minimum will be evicted*)
 - pages referenced in order: 0,1,2,3,2,1,0,3,2,3

Page				
	0	1	2	3
0	0	1	1	1
1	0	0	0	0
2	0	0	0	0
3	0	0	0	0

(a)

Page				
	0	1	2	3
0	0	0	1	1
1	1	0	1	1
2	0	0	0	0
3	0	0	0	0

(b)

Page				
	0	1	2	3
0	0	0	0	1
1	1	0	0	1
2	1	1	0	1
3	0	0	0	0

(c)

Page				
	0	1	2	3
0	0	0	0	0
1	1	0	0	0
2	1	1	0	0
3	1	1	1	0

(d)

Page				
	0	1	2	3
0	0	0	0	0
1	1	0	0	0
2	1	1	0	1
3	1	1	0	0

(e)

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

(f)

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

(g)

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

(h)

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

(i)

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

(j)

LRU 3 -

LRU: Stack Replacement

Stack:

Whenever a page is referenced, it is removed from the stack and put on the top.

reference string

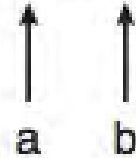
4 7 0 7 1 0 1 2 1 2 7 1 2



stack
before
a



stack
after
b



In this way, the most recently used page is always at the top of the stack and *the least recently used page is always at the bottom*

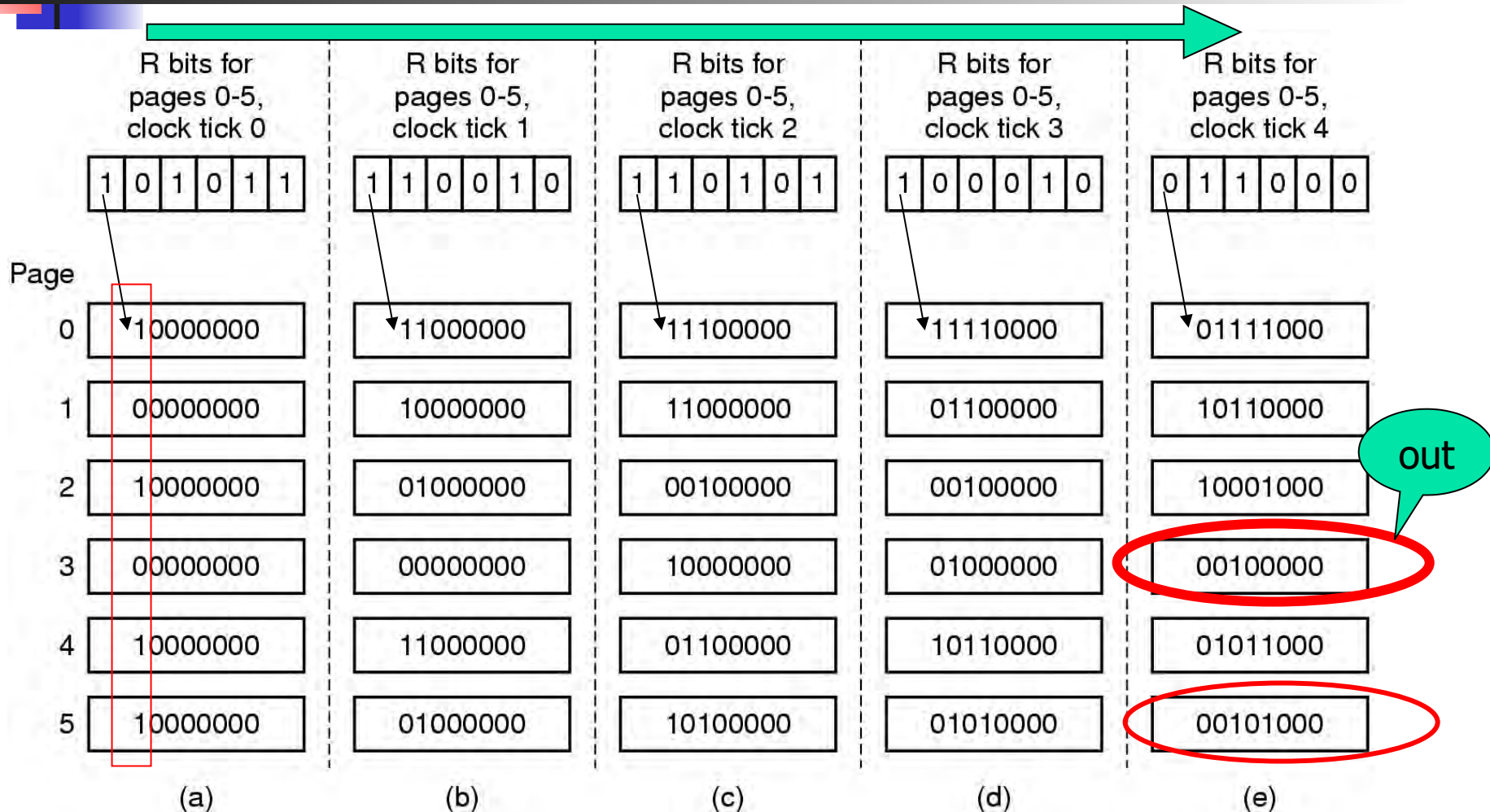


NFU (Not Frequently Used)

- NFU(Not Frequently Used)
 - A software counter associated with each page
 - R bit
 - At each clock interrupt, the os scans all the pages in memory
 - Each page, the R bit, which is 0 or 1, is added to the counter
 - The page with the lowest counter will be evicted
 - The main problem
 - NFU never forgets anything
 - **LRU - Approximation Replacement*

同一个时钟中断内
无数次访问都只算 1 次，
无法知其精确先后


Aging Algorithm 老化算法



- The aging algorithm simulates LRU in software
- Note 6 pages for 5 clock ticks, (a) – (e)

Aging Algorithm (cont.,)

- A reference bit
- and an 8-bit byte for each page in a table in memory: $R = R_{n-1}R_{n-2}R_{n-3} \dots R_2R_1R_0$
- At regular intervals (say, every 100 milliseconds), a timer interrupt transfers control to the operating system



The operating system shifts the reference bit for each page into the high-order bit of its 8-bit byte, shifting the other bits right by 1 bit and discarding the low-order bit

- These 8-bit shift registers contain the history of page use for the last eight time periods
- If we interpret these 8-bit bytes as unsigned integers, the page with *the lowest number* is the LRU page, and it can be replaced
- *LRU - Approximation Replacement
- 不同于 NFU，本算法带有序优先级

Least Frequently Used (LFU)

■ Least Frequently Used (LFU): 最少使用置换算法

- A reference bit and an 8-bit byte for each page in a table in memory: $R = R_{n-1}R_{n-2}R_{n-3}\dots R_2R_1R_0$
- At regular intervals (say, every 100 milliseconds), a timer interrupt transfers control to the operating system
- The operating system shifts the reference bit for each page into the high-order bit of its 8-bit byte, shifting the other bits right by 1 bit and discarding the low-order bit
- These 8-bit shift registers contain the history of page use for the last eight time periods
- $\text{Min}(R_0 + R_1 + \dots + R_{n-1})$ page, and it can be replaced
- *LRU-Approximation Replacement
- 不同于 **NFU** 和 **Aging**，本算法只记录最近 **n** 个阶段



- Not Recently Used Page Replacement Algorithm (NRU)
 - i.e. Enhanced Second-Chance Algorithm
 - **Two bits: (Read, Modify)**
 1. (0, 0) neither recently used nor modified—best page to replace
 2. (0, 1) not recently used but modified—not quite as good, because the page will need to be written out before replacement
 3. (1, 0) recently used but clean—probably will be used again soon
 4. (1, 1) recently used and modified — probably will be used again soon, and the page will be need to be written out to disk before it can be replaced
 - When page replacement is called for, we use the same scheme as in the **clock algorithm**
 - but instead of examining whether the page to which we are pointing has the reference bit (read) set to 1, we examine the class to which that page belongs



Page-Buffering Algorithms

- Systems commonly keep **a pool of free frames**
 - When a page fault occurs, a victim frame is chosen as before
 - However, the desired page is read into a free frame from the pool before the victim is written out
 - This procedure allows the process to restart as soon as possible, without waiting for the victim page to be written out.
- An expansion of this idea is to maintain a list of modified pages
 - Whenever the paging device is idle, a modified page is selected and is written to the disk, its modify bit is then reset
 - This scheme increases the probability that a page will be clean when it is selected for replacement and will not need to be written out.

页面缓冲算法

work set

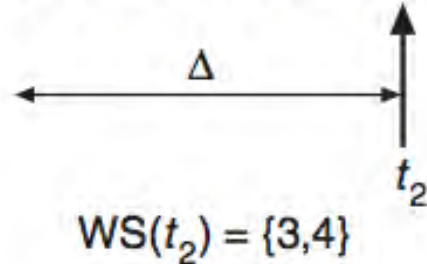
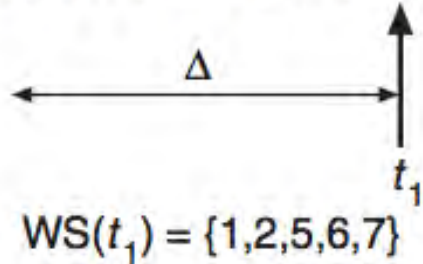
- work set

- 工作集 , 1968, Denning
- If the entire working set is in memory, the process will run without causing many faults until it moves into another execution phase
- If the available memory is too small to hold the entire working set, the process will cause many page faults and run slowly, since executing an instruction takes a few nanoseconds and reading in a page from the disk typically takes 10 milliseconds
- The set of pages

work set

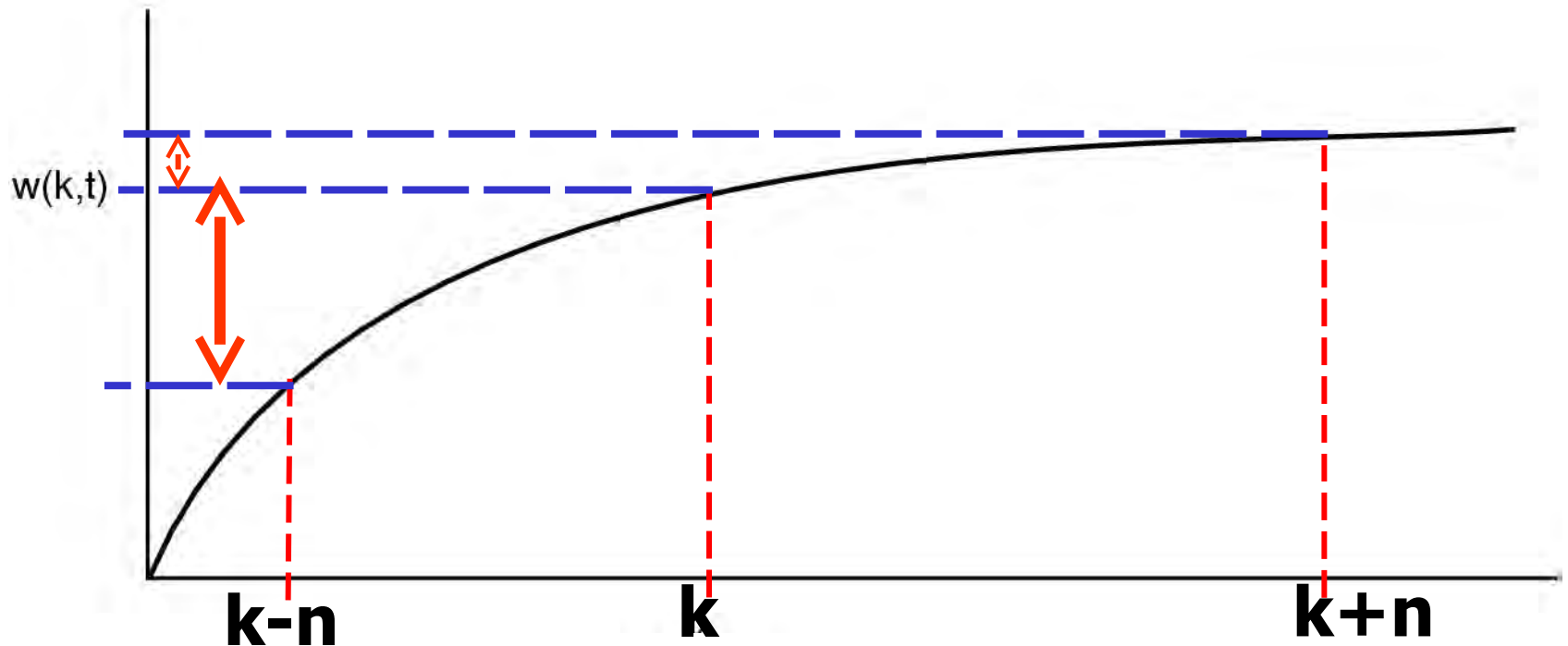
page reference table

... 2 6 1 5 7 7 7 7 5 1 6 2 3 4 1 2 3 4 4 4 3 4 3 4 4 4 4 1 3 2 3 4 4 4 3 4 4 4 ...



- working-set window
- working-set model

work set



- k : the k -th access memory
- t : the time
- The working set is the set of pages used by the k most recent memory references
- $w(k, t)$ is the size of the working set at time, t

The WorkingSet Replacement Algorithm

To find a page that is not in the working set and evict it

R:Referenced

M:Modified

2204

Current virtual time

the referenced bit is cleared on every clock tick

Information about
one page

Time of last use

Page referenced
during this tick

Page not referenced
during this tick

R (Referenced) bit

每次都需要遍历
所有页表项，找最优

Scan all pages examining R bit:

if ($R == 1$)

set time of last use to current virtual time

if ($R == 0$ and $\text{age} > \tau$)

remove this page

if ($R == 0$ and $\text{age} \leq \tau$)

remember the smallest time

⋮		
	2084	1
	2003	1
	1980	1
	1213	0
	2014	1
	2020	1
	2032	1
	1620	0

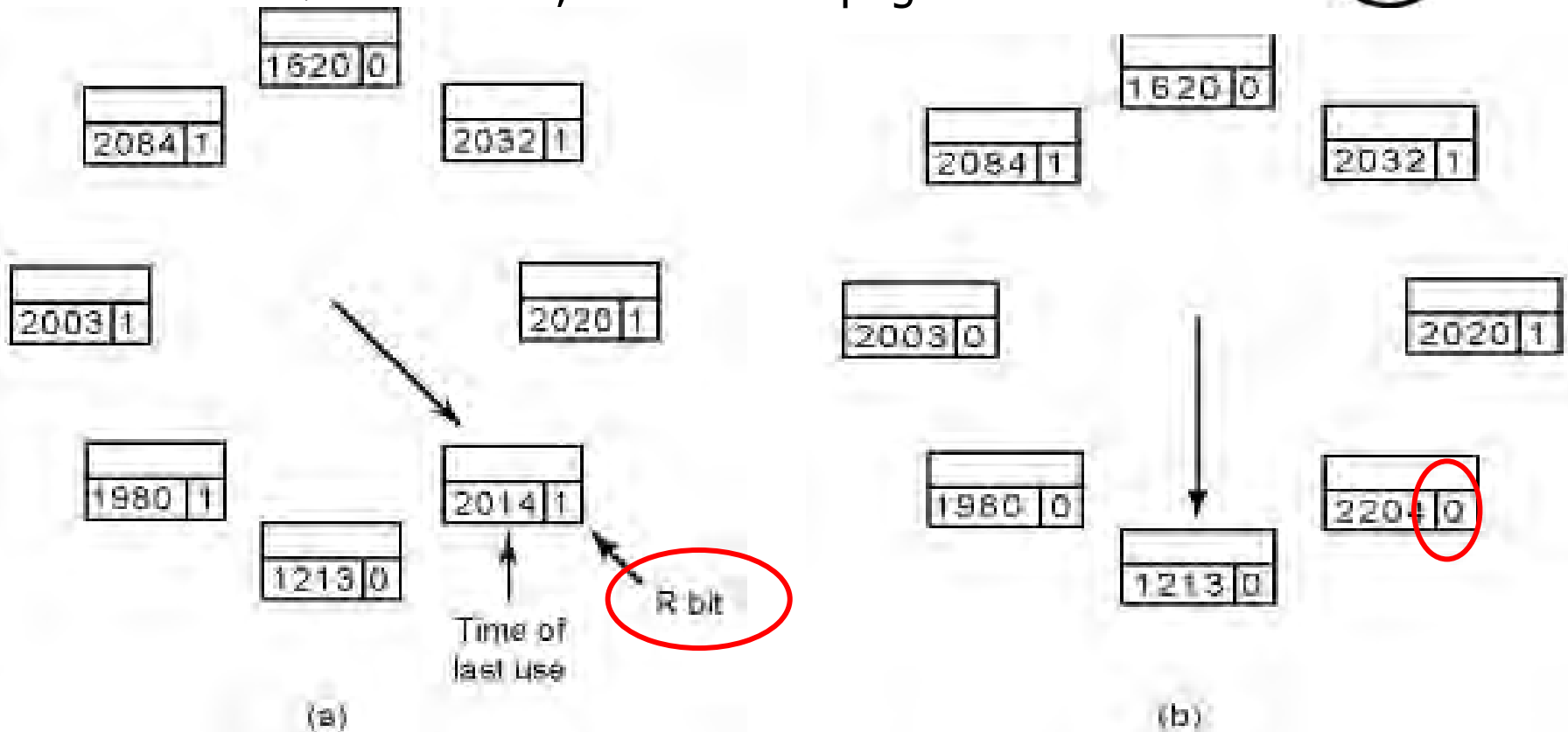
Page table

The WSClock Page Replacement Algorithm

■ 工作集时钟算法

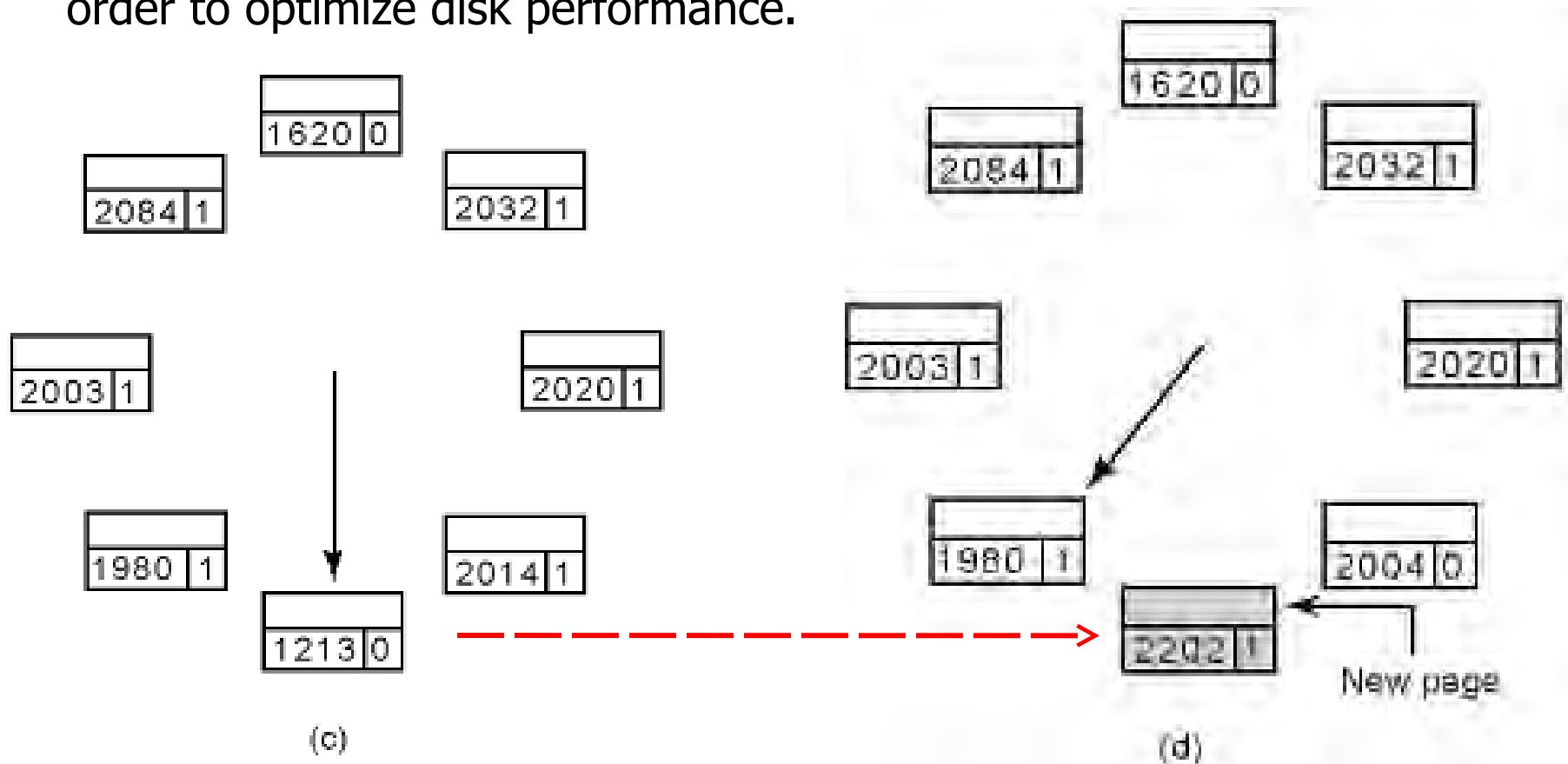
2204 Current virtual time

- An Improved algorithm
- If $r=1$, then $r=0$, handle next page



The WSClock Page Replacement Algorithm 2/2

- If ($r == 0$ and $\text{age} > \tau$ and $m == 0$) then evicted
- If ($m == 1$) then look for the next page
- The first clean page encountered is evicted. This page is not necessarily the first write scheduled because the disk driver may reorder writes in order to optimize disk performance.



Review of Page Replacement Algorithms

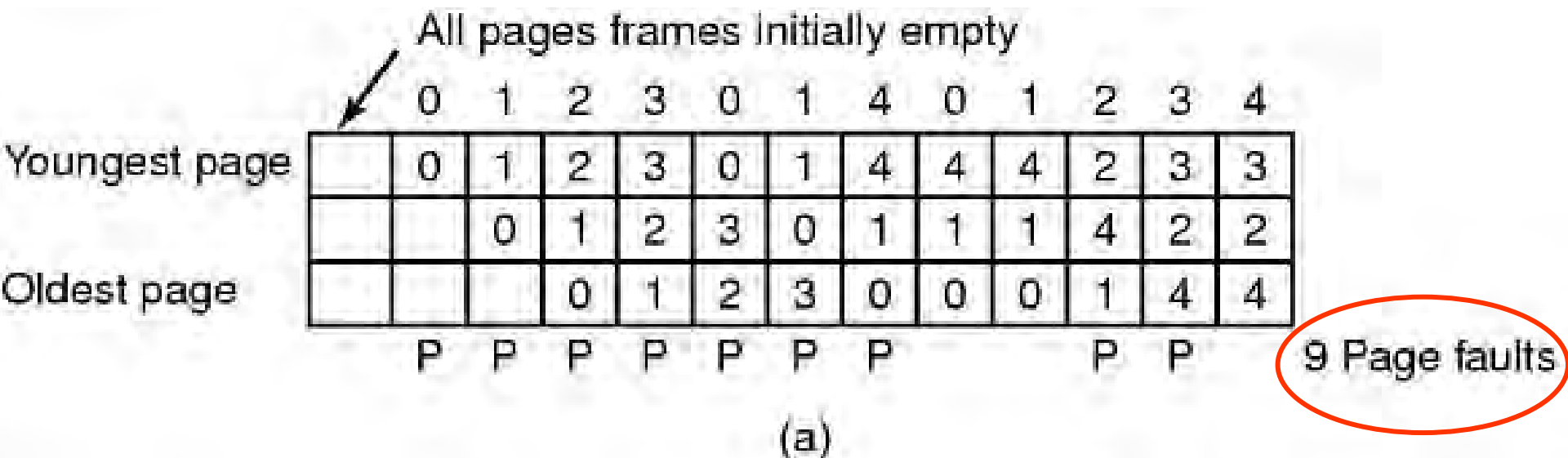
Algorithm	Comment
Optimal	Not implementable, but useful as a benchmark
NRU (Not Recently Used)	Very crude
FIFO (First-In, First-Out)	Might throw out important pages
Second chance	Big improvement over FIFO
Clock	Realistic
LRU (Least Recently Used)	Excellent, but difficult to implement exactly
NFU (Not Frequent)	Fairly crude approximation to LRU
Aging	Efficient algorithm that approximates LRU well
Working set	Somewhat expensive to implement
WSClock	Good efficient algorithm



Performance of Demand Paging

- access time
 - ma: memory access time: 10ns~200ns
 - p: the probability of a page fault ($0 \leq p \leq 1$)
 - 8ms = 8000us
- effective access time
 - $(1 - p) \times \text{ma} + p \times \text{page fault time}$
 - $(1 - p) \times (200) + p (8 \text{ milliseconds})$
 - $(1 - p) \times 200 + p \times 8000000$
 - $200 + 7999800 \times p$
- How to improve the performance
 - Paging system
 - Disk I/O

Belady's Anomaly 1/2



- FIFO with 3 page frames
- *P*'s show which page references show page faults
- ?????When FIFO with 4 page frames

Belady's Anomaly 2/2

		0	1	2	3	0	1	4	0	1	2	3	4
Youngest page		0	1	2	3	3	3	4	0	1	2	3	4
			0	1	2	2	2	3	4	0	1	2	3
Oldest page				0	1	1	1	2	3	4	0	1	2
					0	0	0	1	2	3	4	0	1
		P	P	P	P			P	P	P	P	P	P

(b)

10 Page faults

When FIFO with 4 page frames

Laszlo Belady



- Laszlo Belady
 - (*29th April 1928) is a (Hungarian born) computer scientist notable for devising the Belady's Min theoretical memory caching algorithm in 1966 while working at IBM Research.
- In his later years at IBM he was responsible for Software engineering worldwide.
- In 1984, he co-founded Microelectronics Computer Corporation in Austin. From 1990 to 1998 he served and later served as President and CEO of Mitsubishi Electric Laboratories, Inc.
- He has been in various University advisory roles including a member of the computer science advisory board at the University of Colorado at Boulder.
- Belady studied aeronautical Engineering in Budapest, Hungary. He left Hungary after the uprising 1956. He first lived in Paris, then in London and came 1961 to the US.

Stack algorithm

- Belay's anomaly
 - for some page-replacement algorithms, the page-fault rate may increase as the number of allocated frames increases
- Stack algorithm 栈式算法
 - reference string
 - 0 2 1 3 5 4 6 3 7 4 7 3 3 5 5 3 1 1 1 7 2 3 5 3
 - a paging system
 - 1. the reference string of the executing process
 - 2. the page replacement algorithm
 - 3. the number of page frames available in memory, m

Stack Algorithms

LRU Replacement Algorithm

Reference string	0	2	1	3	5	4	6	3	7	4	7	3	3	5	5	3	1	1	1	7	2	3	4	3		
M			0	2	1	3	5	4	6	3	7	4	7	3	3	5	5	3	1	1	1	7	1	3	4	1
				0	2	1	3	5	4	6	3	7	4	7	7	3	3	5	3	3	3	1	7	1	3	4
					0	2	1	3	5	4	6	3	3	4	4	7	7	7	5	5	5	3	3	7	1	3
						0	2	1	3	5	4	6	6	6	6	4	4	4	7	7	7	5	5	5	7	7
							0	2	1	1	5	5	5	5	5	6	6	6	4	4	4	4	4	4	5	5
								0	2	2	1	1	1	1	1	1	1	1	6	6	6	6	6	6	6	6
									0	0	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
											0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Page faults		P	P	P	P	P	P		P					P			P								P	
Distance string	∞	∞	∞	∞	∞	∞	∞	4	∞	4	2	3	1	5	1	2	6	1	1	4	7	4	6	5		

State of memory array, M , after each item in reference string is processed

Stack Algorithms

$$M(m, r) \subseteq M(m + 1, r)$$

- $M(m, r)$
 - m varies over the page frames
 - r is an index into the reference string
 - M keeps track of the state of memory
- Stack Algorithms have the above property
- ***FIFO** dose not have it
 - **! write your answer why not**

Distance Strings 1/2

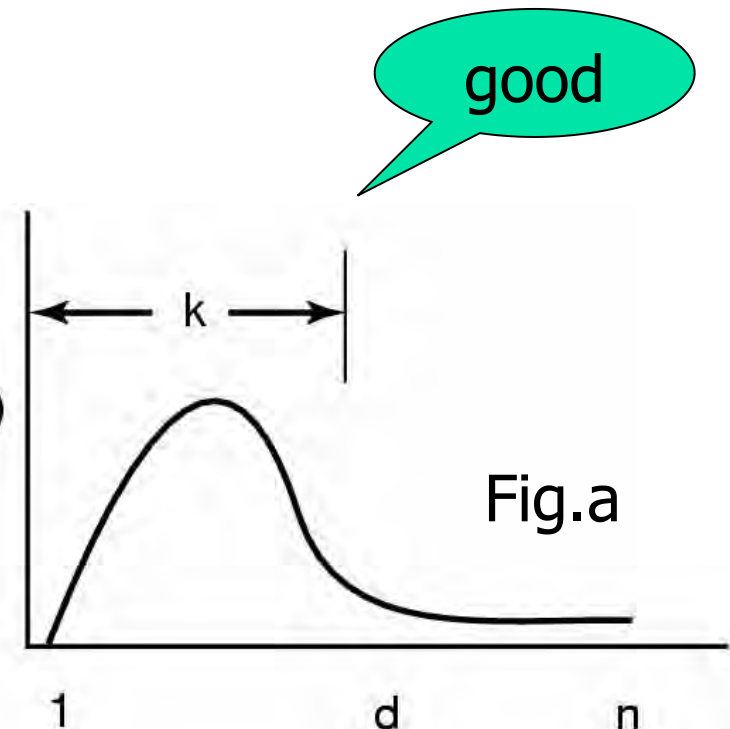
■ Distance Strings 距离字符串

∞ ∞ ∞ ∞ ∞ ∞ ∞ ∞ 4 ∞ 4 2 3 1 5 1 2 6 1 1 4 7 4 6 5

- Depend on
 - reference string
 - Page replacement algorithms
 - To forecast **page fault rate** in different memory sizes

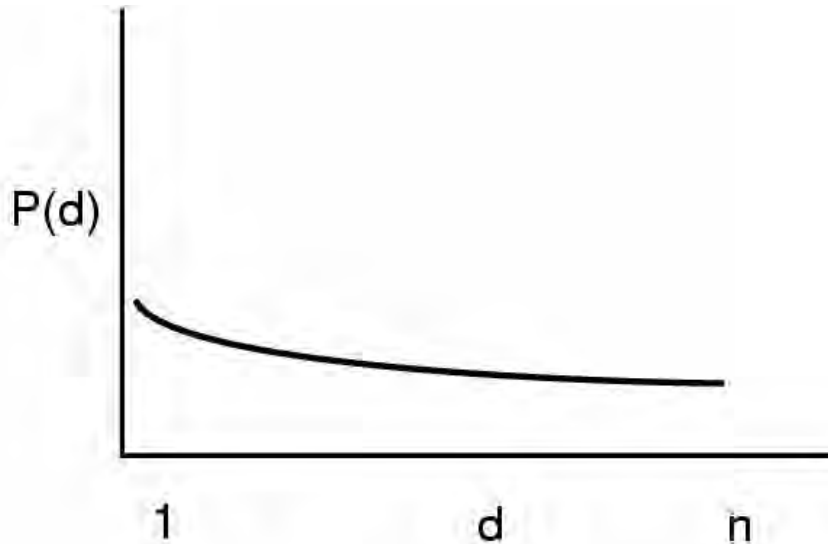
■ Probability density functions

- $P(d)$: the probability density function $P(d)$ for the entries in a distance string d
 - $d: 1, 2, 3, 4, 5, \dots$
- Fig.a: with a memory of k page frames, few page faults occur



Distance Strings 2/2

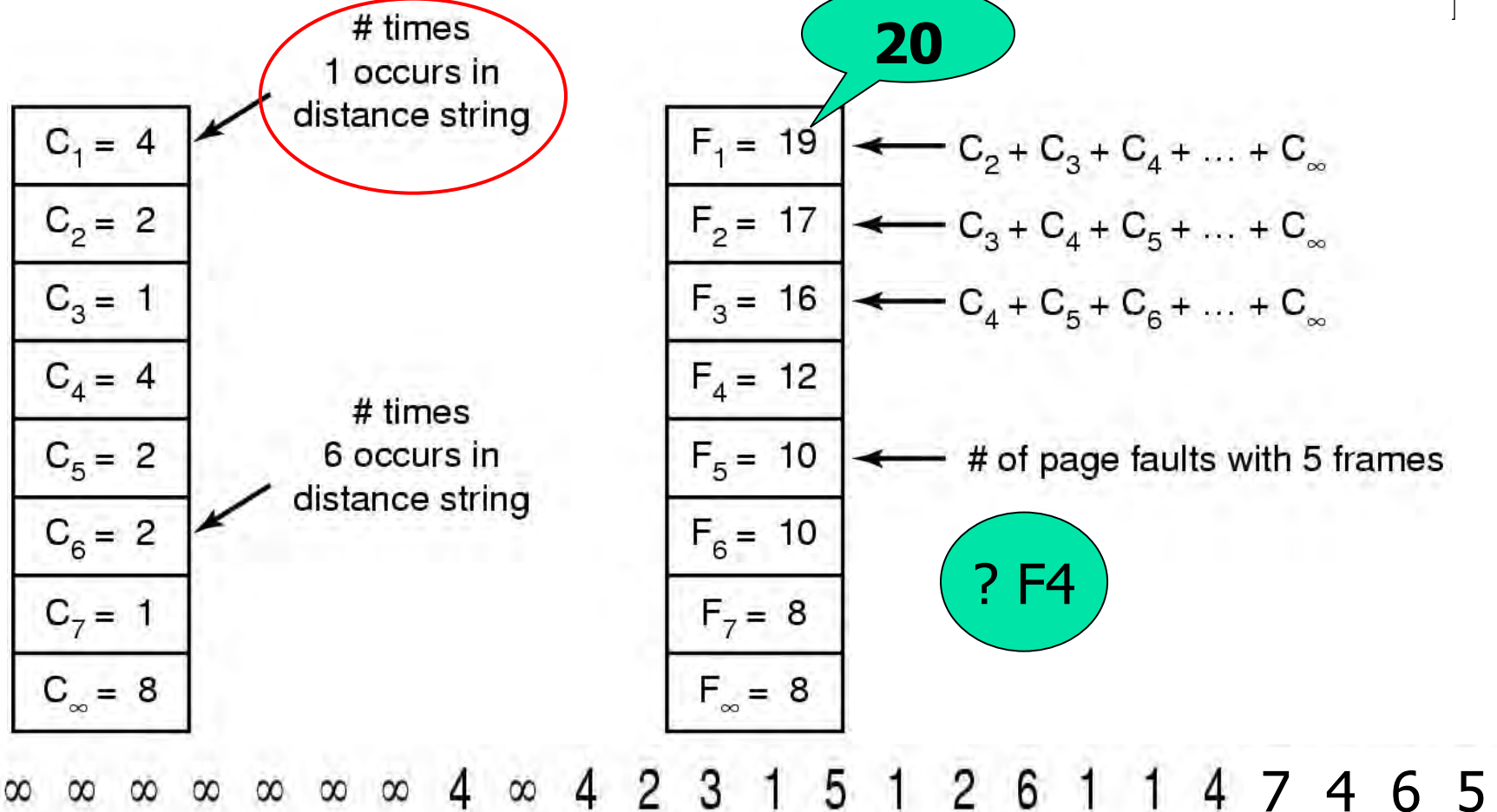
- Probability density functions (cont.,)
 - $P(d)$: so spread out
 - The only way to avoid a large number of page faults is to give the program as many page frames as it has virtual pages



Bad luck

Predicting Page Fault Rates

- $F_m = \sum c_k + C_\infty, \quad m+1 \leq k \leq n$
- F_m : #page faults occurring with a given distance string and m page frames



DESIGN ISSUES FOR PAGING SYSTEMS

- Local versus Global Allocation Policies
- Load Control
- Page Size
- Separate Instruction and Data Spaces
- Shared Pages
- Cleaning Policy
- Virtual Memory Interface

Local versus Global Allocation Policies 1/2

- Original configuration: request page A6
- Local page replacement 、 Global page replacement

	Age
A0	10
A1	7
A2	5
A3	4
A4	6
A5	3
B0	9
B1	4
B2	6
B3	2
B4	5
B5	6
B6	12
C1	3
C2	5
C3	6

(a)

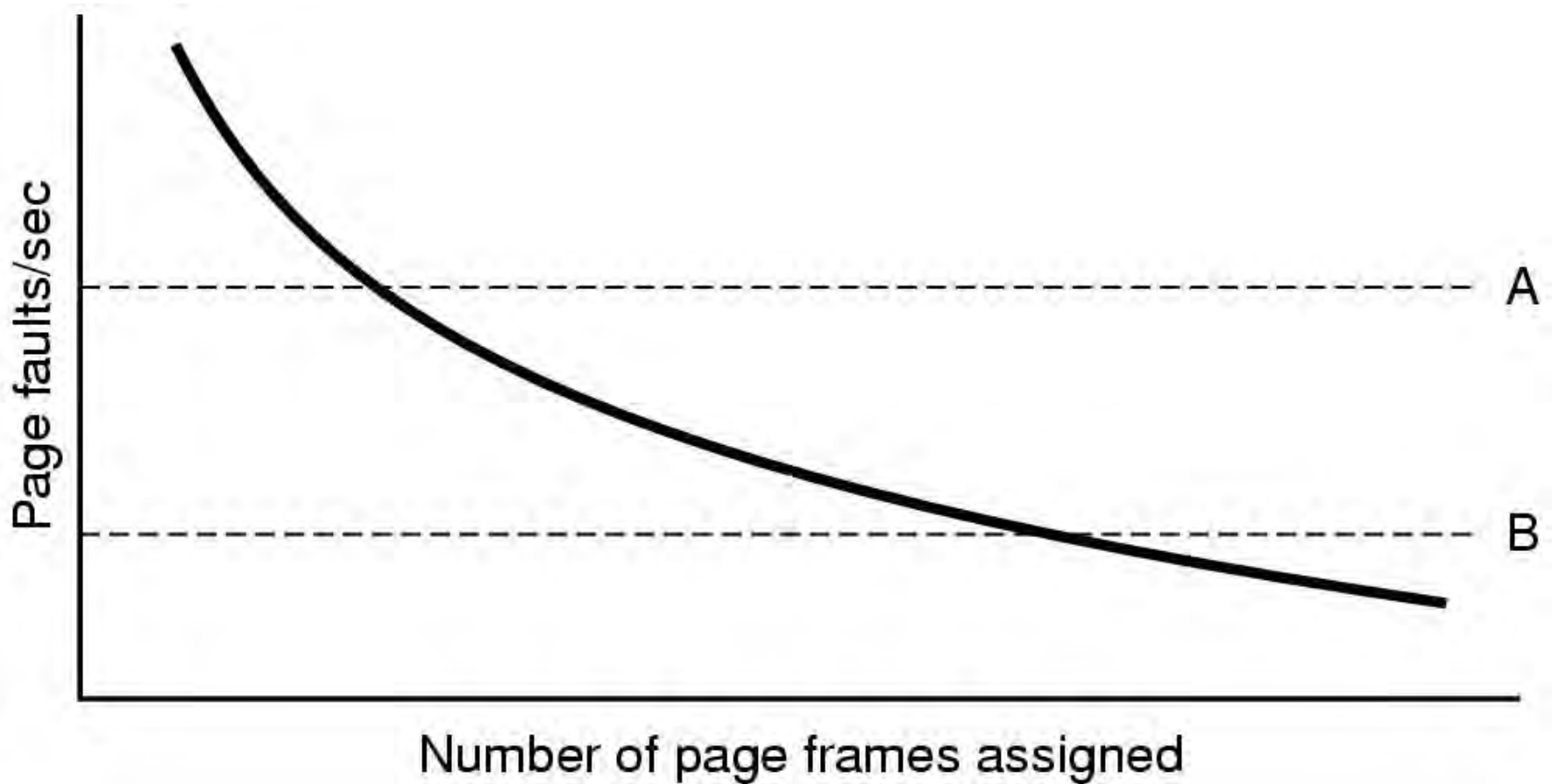
A0
A1
A2
A3
A4
A6
B0
B1
B2
B3
B4
B5
B6
C1
C2
C3

(b)

A0
A1
A2
A3
A4
A5
B0
B1
B2
A6
B4
B5
B6
C1
C2
C3

(c)

Local versus Global Allocation Policies 2/2



- **Page fault rate** as a function of the number of page frames assigned
- Page Fault Frequency, PFF 页面失效频率

Thrashing

- Thrashing

- 抖动, 颠簸
- A program causing page faults every few instruction is said to be thrashing
- Swap In, Swap Out
- The high paging activity is called thrashing
- A process is thrashing if it is spending more time paging than executing

Cause of Thrashing

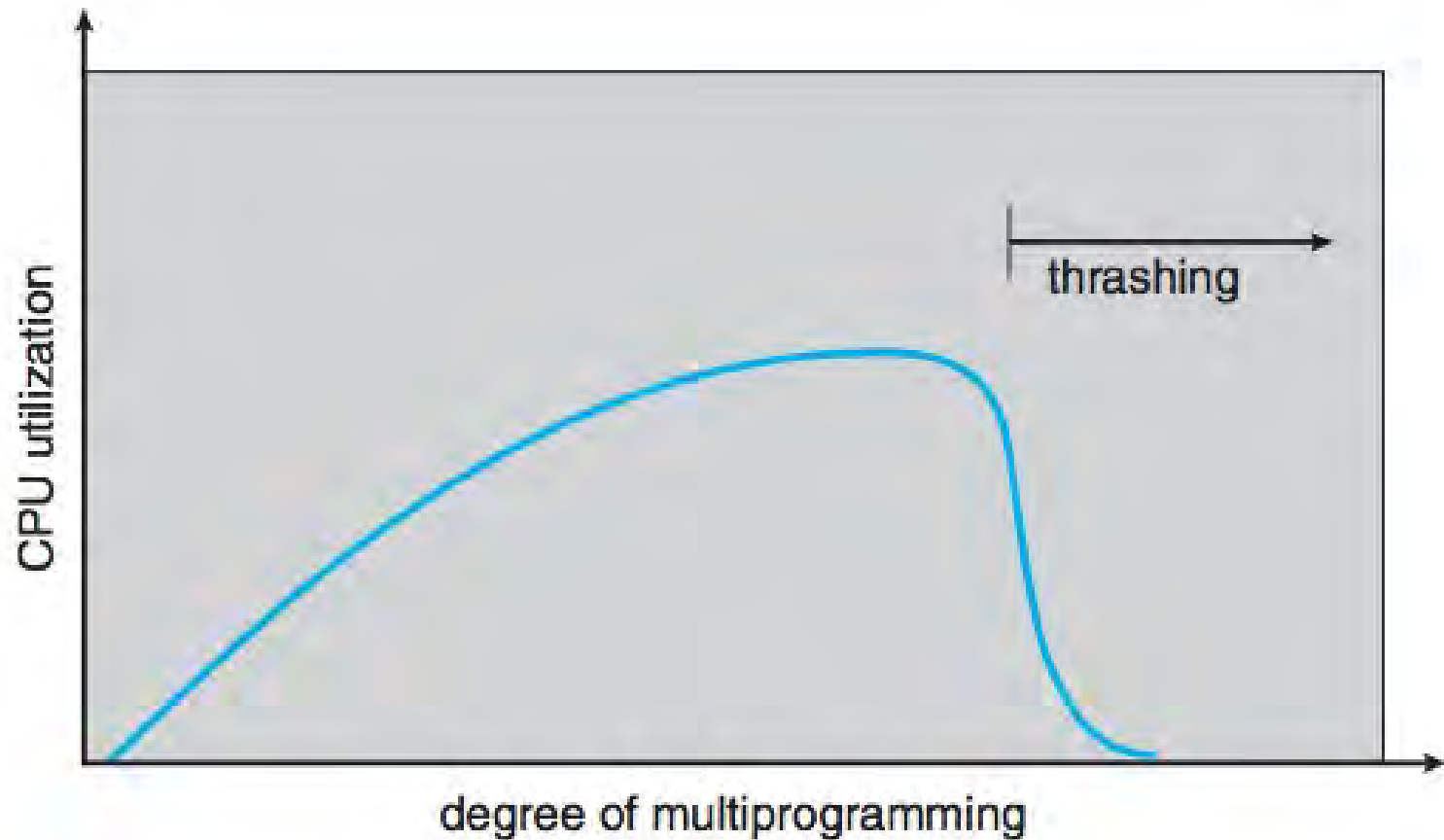
- Cause 1:
 - Multi-processes
 - A global page-replacement algorithm
 - Circlely take from frames of another processes
 - So, these processes are thrashing because they are spending more time paging than executing

Cause of Thrashing

■ Cause 2:

- The operating system monitors CPU utilization.
- If CPU utilization is too low, we increase the degree of multiprogramming by introducing a new process to the system.
- A global page-replacement algorithm is used; it replaces pages without regard to the process to which they belong.
- Now suppose that a process enters a new phase in its execution and needs more frames.
 - It starts faulting and taking frames away from other processes. These processes need those pages, however, and so they also fault, taking frames from other processes.
 - These faulting processes must use the paging device to swap pages in and out. As they queue up for the paging device, the ready queue empties.
- As processes wait for the paging device, CPU utilization decreases.

Cause of Thrashing



Thrash Prevention

- a local replacement algorithm
 - or priority replacement algorithm
- Working-Set Model
 - Working-set window
- $L=S$ (Denning,1980)
 - L:average page-fault frequency
 - S: average page-fault dispose time
 - CPU: best utilization rate
- Suspend some processes

Load Control 负载控制

- Despite good designs, system may still ***thrash***
 - Whenever the combined working sets of all processes exceed the capacity of memory, thrash can be expected
- When PFF algorithm indicates
 - (Page Fault Frequency, PFF)
 - some processes need more memory
 - but no processes need less
- Solution:
Reduce number of processes competing for memory
 - swap one or more to disk, divide up pages they held
 - reconsider degree of multiprogramming

Page Size 1/2

Small page size

■ Advantages

- less internal fragmentation
- better fit for various data structures, code sections
- less unused program in memory

■ Disadvantages

- programs need many pages, larger page tables

Page Size 2/2

- Overhead(开销) due to page table and internal fragmentation

$$overhead = \frac{s \cdot e}{p} + \frac{p}{2}$$

Diagram illustrating the overhead components:

- The first term, $\frac{s \cdot e}{p}$, is labeled "page table space".
- The second term, $\frac{p}{2}$, is labeled "internal fragmentation".

- Where

- s = average process size in bytes
- p = page size in bytes
- e = page entry

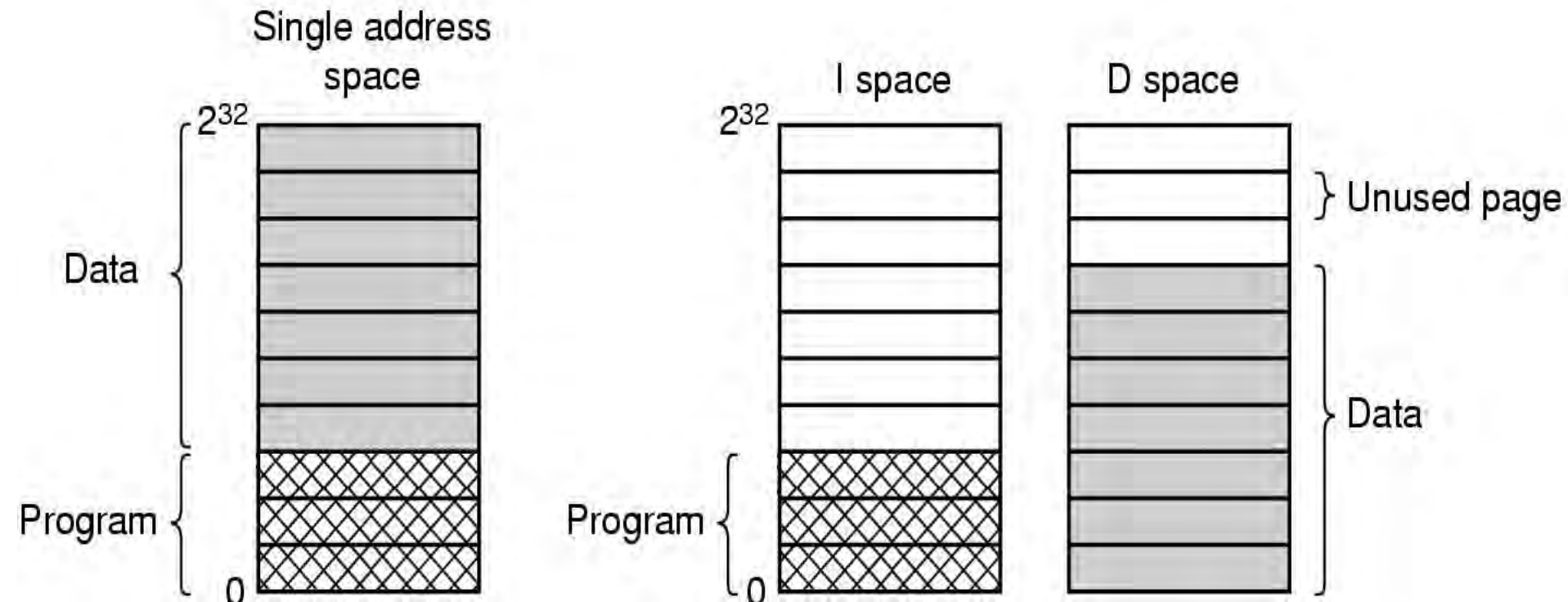
Optimized when

求导 : $-se/(p \cdot p) + 1/2 = 0$

$$p = \sqrt{2se}$$

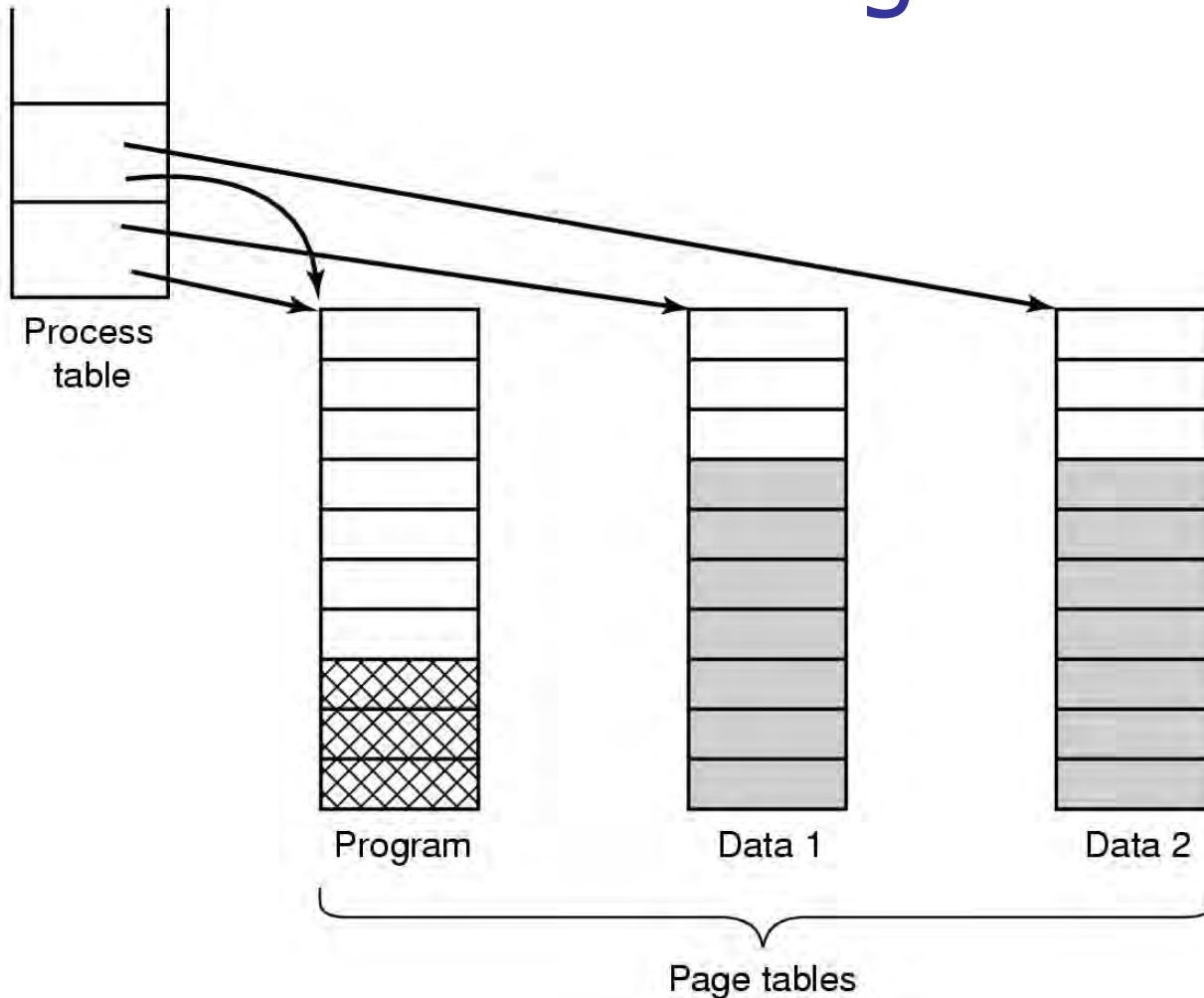
$$\bullet s=1\text{MB}; e=8\text{Bytes}; p_{\text{opt}}=4\text{KB}$$

Separate Instruction and Data Spaces



- One address space
- Separate I and D spaces
 - Instruction Address Space
 - Data Address Space

Shared Pages



Two processes sharing same program sharing its page table

Cleaning Policy

- Cleaning Policy
 - 清除策略
 - Paging works best when there are plenty of free frames that can be claimed as page faults occur
- Need for a background process
 - **paging daemon**, periodically inspects state of memory
- When too few frames are free
 - selects pages to evict using a replacement algorithm
- It can use same circular list (clock)
 - Two-handed clock
 - The front hand is controlled by the paging daemon
 - The back hand is used for page replacement

Virtual Memory Interface

- Virtual Memory Interface

- 虚拟存储器

- In some advanced systems, programmers have some control over the memory map and can use it in nontraditional ways to enhance program behavior

- Cases

- Two or more processes to share the same memory
 - A high-performance message-passing system
 - Only the page names have to be copied, instead of all the data
 - Distributed shared Memory
 - Allow multiple processes over a network to share a set of pages, possibly, but not necessarily, as a single shared linear address space

Implementation Issues

- Operating system Involvement with Paging
- Page Fault Handling
- Instruction Backup
- Locking Pages in Memory
- Backing Store
- Separation of Policy and Mechanism

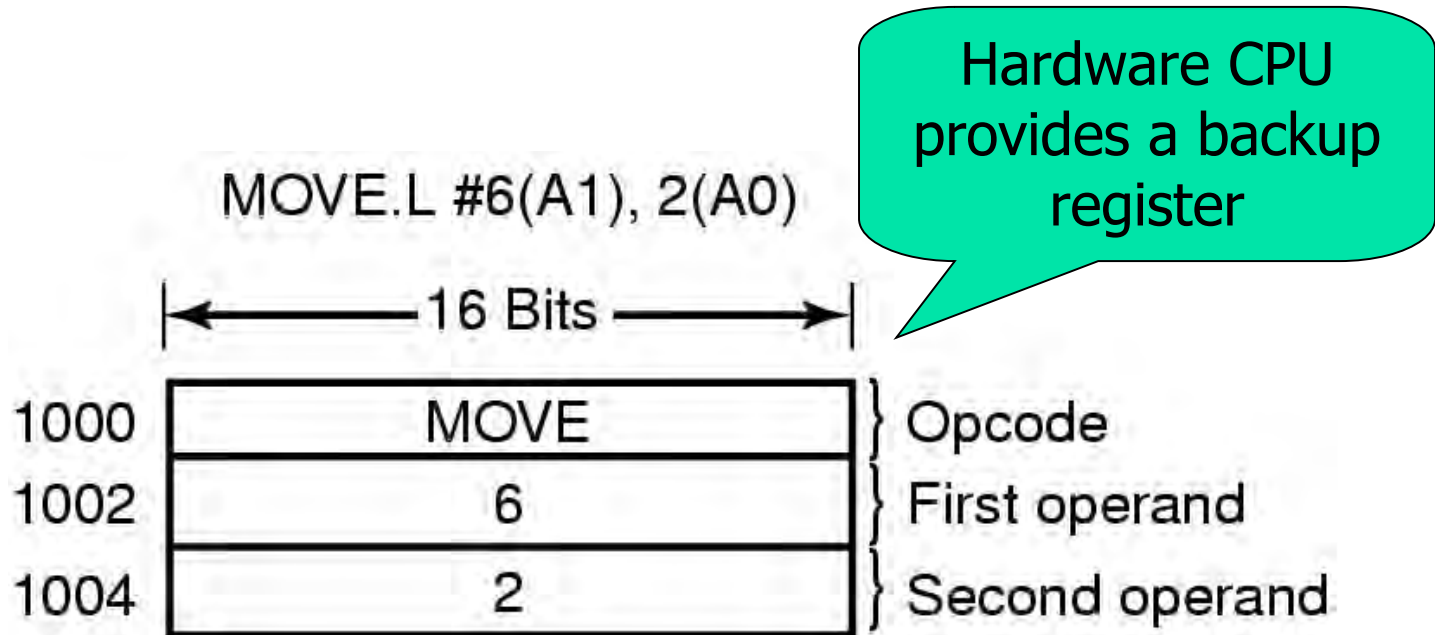
Operating System Involvement with Paging

- Four times when OS involved with paging
- 1.Process creation
 - determine program size
 - create page table
- 2.Process execution
 - MMU reset for new process
 - TLB flushed
- 3.Page fault time
 - determine virtual address causing fault
 - swap target page out, needed page in
- 4.Process termination time
 - release page table, pages

Page Fault Handling

- 1) Hardware traps to kernel
- 2) General registers saved
- 3) OS determines which virtual page needed
- 4) OS checks validity of address, seeks page frame
- 5) If selected frame is dirty, write it to disk
- 6) OS brings schedules new page in from disk
- 7) Page tables updated
- 8) Faulting instruction backed up to when it began
- 9) Faulting process scheduled
- 10) Registers restored
- 11) Program continues

Instruction Backup 指令备份

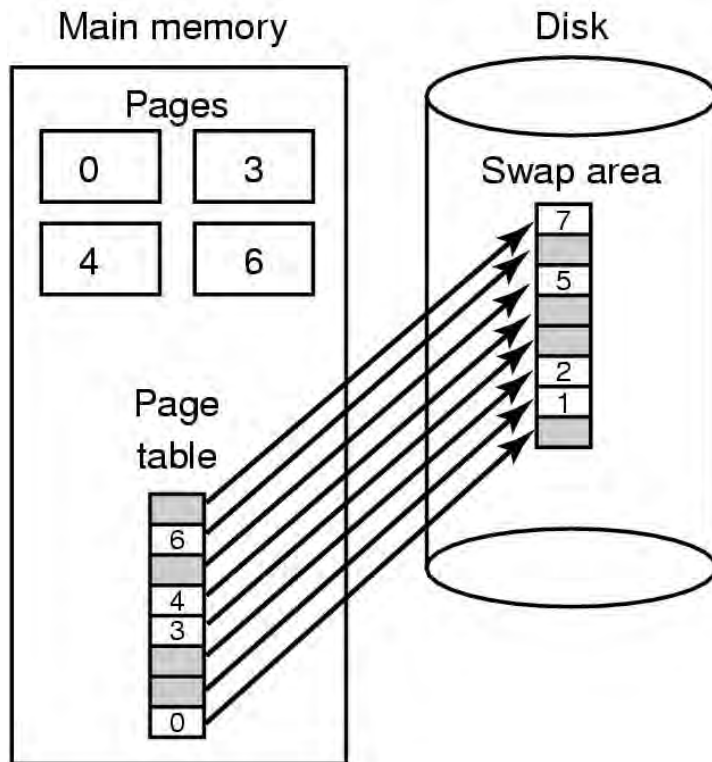


An instruction causing a page fault

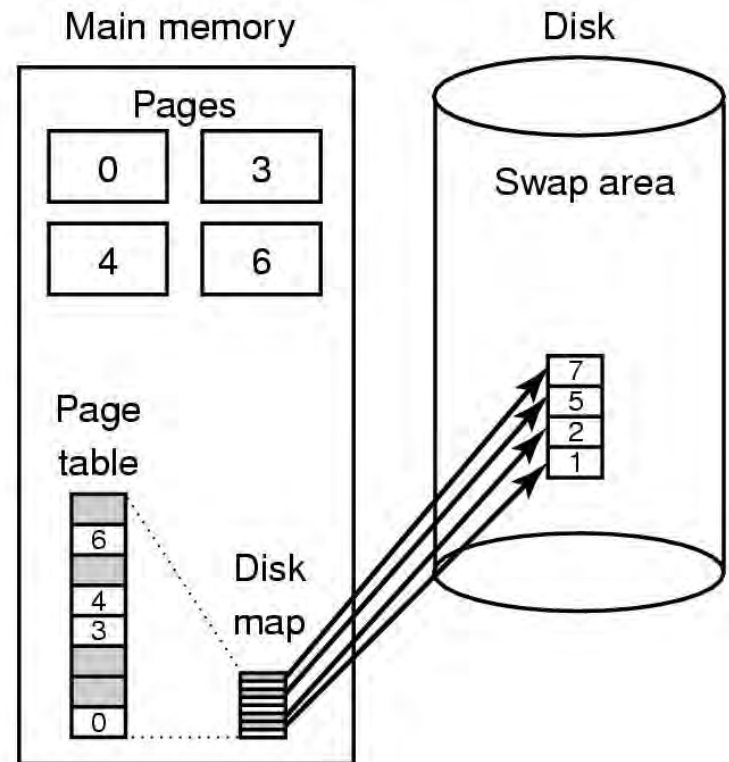
Locking Pages in Memory

- Virtual memory and I/O occasionally interact
 - Consider a process that has just issued a system call to read from some file or device into a buffer within its address space.
 - While waiting for the I/O to complete the process is suspended and another process is allowed to run. This other process gets a page fault.
 - If the paging algorithm is global, there is a small, but nonzero, chance that the page containing the I/O buffer will be chosen to be removed from memory
 - Scence: DMA
- Need to specify some pages locked
 - exempted from being target pages
- way 1:
 - pinning(钉住页面)
- way 2:
 - Do all I/O to kernel buffers and then copy the data to user pages later

Backing Store 后备存储



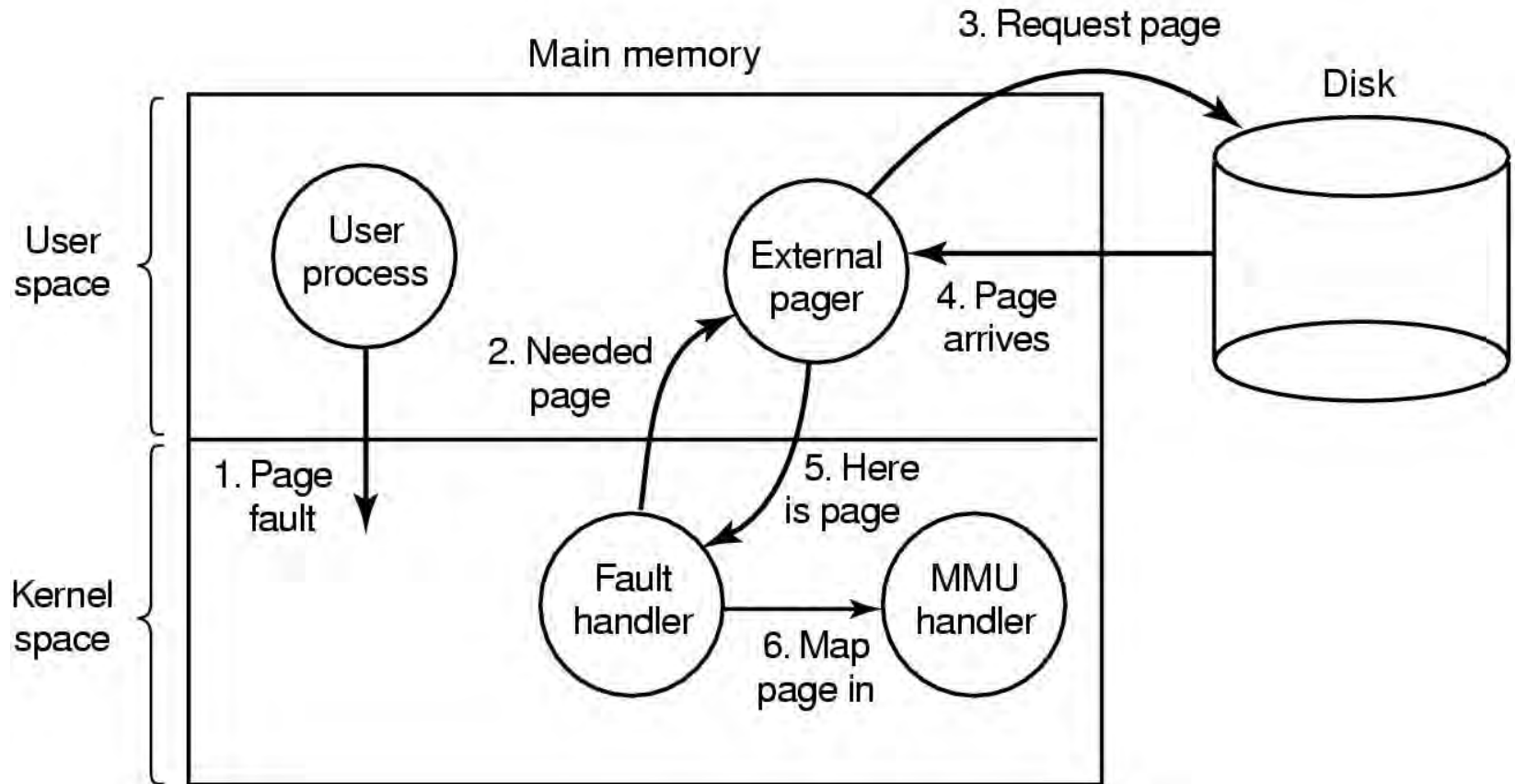
(a)



(b)

- (a) Paging to static swap area
- (b) Backing up pages dynamically

Separation of Policy and Mechanism



Page fault handling with an external pager



Summary

- Virtual Memory Management
- Page Replacement Algorithms
- Design Issues for Paging Systems
- Implementation Issues
- More topics



Q&A?

[illegible]