

Pengelolaan Memori: Virtual Memory



INFORMATIKA
UNPAR

Memory Management : Resume

- Monoprogramming
- Multiprogramming
 - Fixed Partition
 - Equal size
 - Unequal size
 - Multiple queue
 - Single queue
 - Swapping & Overlays
 - Internal Fragmentation
 - Dynamic Partition
 - Partitions are of variable length and number ; Process is allocated exactly as much memory as required
 - External Fragmentation – Compaction
 - Placement Algorithm: First-fit, best-fit, worst-fit, next-fit, quick fit

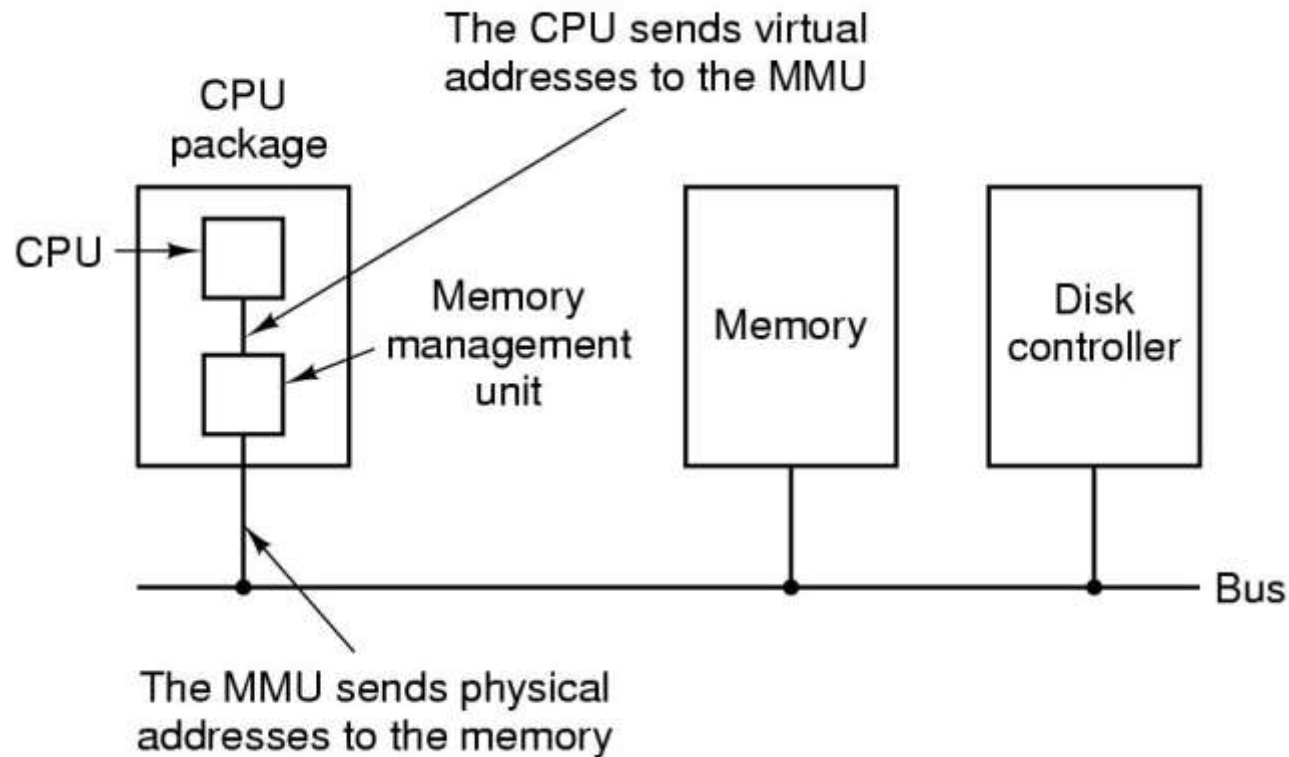
Virtual Memory

- There is a need to run programs that are too large to fit in memory
- Solution adopted in the 1960s, split programs into little pieces, called overlays
 - Kept on the disk, swapped in and out of memory
- **Virtual memory : *each program has its own address space, broken up into chunks called pages***

Virtual Memory

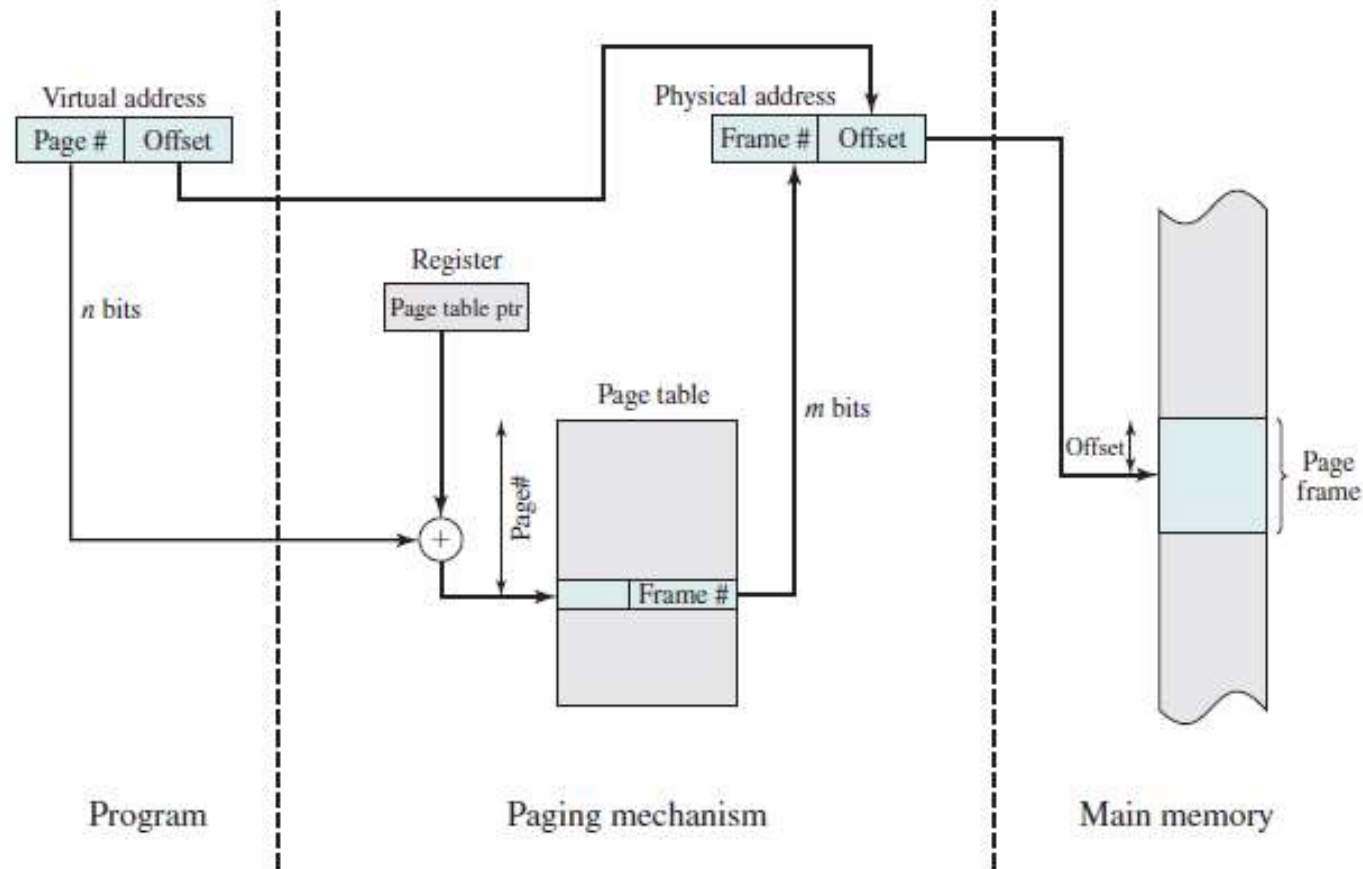
- Most virtual memory systems use a technique called **paging**
- Program-generated addresses are called **virtual addresses** and form the **virtual address space**
- When virtual memory is used, the virtual addresses do not go directly to the memory bus; **MMU (Memory Management Unit)** that maps the virtual addresses onto the physical memory addresses
- The virtual address space consists of fixed-size units called **pages**. The corresponding units in the physical memory are called **page frames**.
- The pages and page frames are generally the same size. In this example they are 4 KB, but page sizes from 512 bytes to a gigabyte have been used in real systems.

Virtual Memory Paging (1)



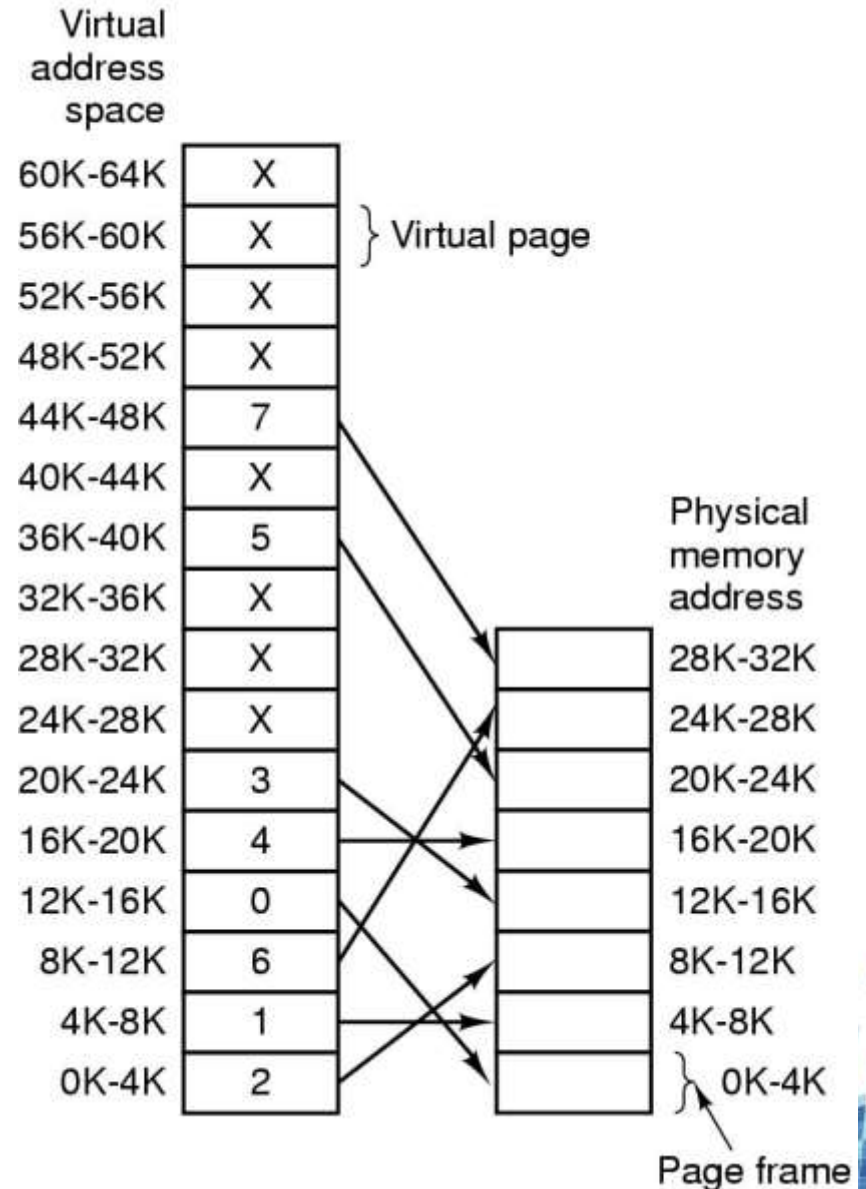
The position and function of the MMU

Translasi address pada Sistem Paging

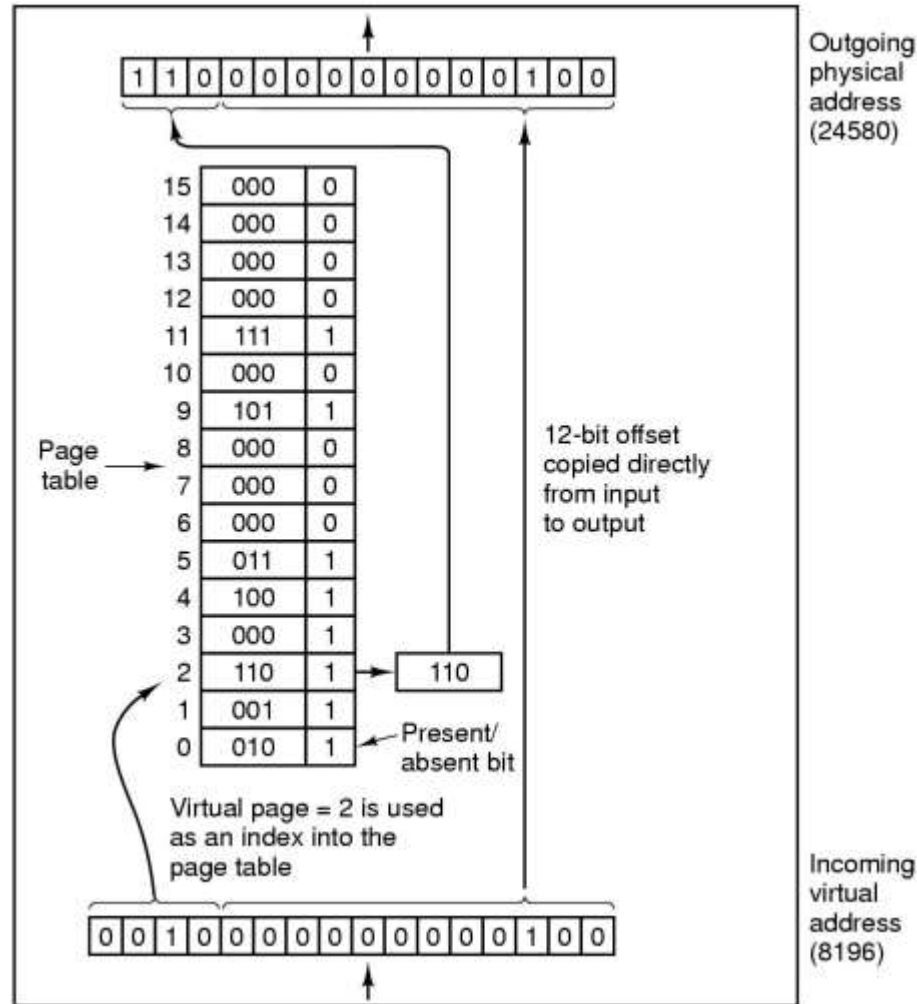


Paging (2)

The relation between virtual addresses and physical memory addresses given by page table



Page Tables (1)



Internal operation of MMU with 16 4 KB pages

Translasi dari Virtual Address (VA) ke Physical Address (VP)

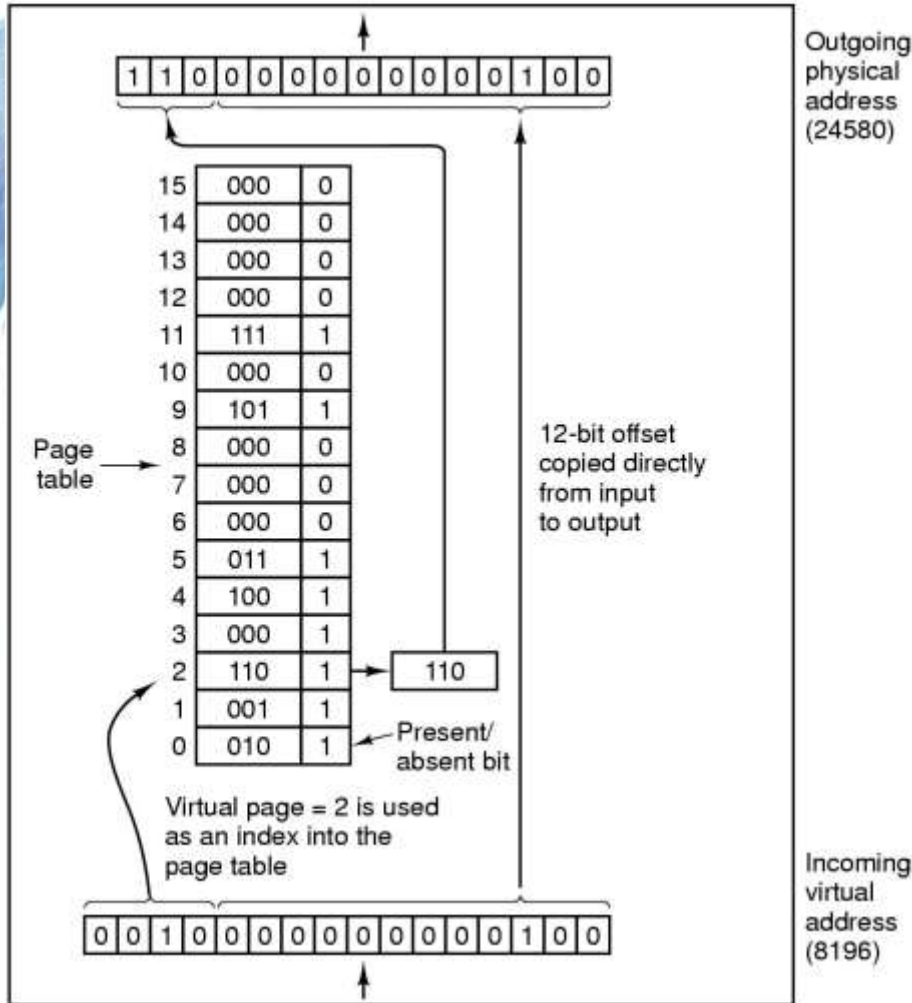
- **VA (decimal) → VP**
 - **VA: 2184 → 0000 1000 1000 1000**
 - Page # : 0000 (index = 0 di page table)
 - Offset : 1000 1000 1000
 - **: 010 1000 1000 1000 (decimal: 1037)**
 - Frame #: 010
 - Offset : 1000 1000 1000
 - **VA: 46407 → 1011 0101 0100 0111**
 - Page # : 1001 (index = 11 di page table)
 - Offset #: 0101 0100 0111
 - **: 111 0101 0100 0111 (decimal: 30023)**
 - Frame #: 111
 - Offset #: 0101 0100 0111

Translasi dari Virtual Address (VA) ke Physical Address (VP)

■ VA (decimal) → VP

- VA: 2184 → **0000** 1000 1000 1000
 - Page # : **0000** (index = 0 di page table)
 - Offset : 1000 1000 1000
- PA: **010** 1000 1000 1000 (decimal: 10376)
 - Frame #: **010**
 - Offset : 1000 1000 1000
- VA: 46407 → **1011** 0101 0100 0111
 - Page # : 1001 (index = 11 di page table)
 - Offset #: 0101 0100 0111
- PA: **111** 0101 0100 0111 (decimal: 30023)
 - Frame #: **111**
 - Offset #: 0101 0100 0111

Page Table



Internal operation of MMU with 16 4 KB pages

- Ukuran page: 4 KB (12 bit address)
- Jumlah virtual page: 16 (4 bit address)
- Ukuran virtual address space : 64 KB
- Memerlukan 4 + 12 bit address (virtual address) untuk table page
- Isi setiap table ada 3 bit utk page frame number dan 1 bit untuk present/absent bit
 - Berarti ada 8 page frame memory (8 x 4 KB = 32 KB)
- Jika proses memiliki virtual address space sebesar 64 MB dan ukuran virtual page 4KB
 - Berapa bit virtual address ?
- Jika virtual address space 1 GB dan ukuran virtual page 8KB
 - Berapa bit virtual address?

Structure of a Page Table Entry

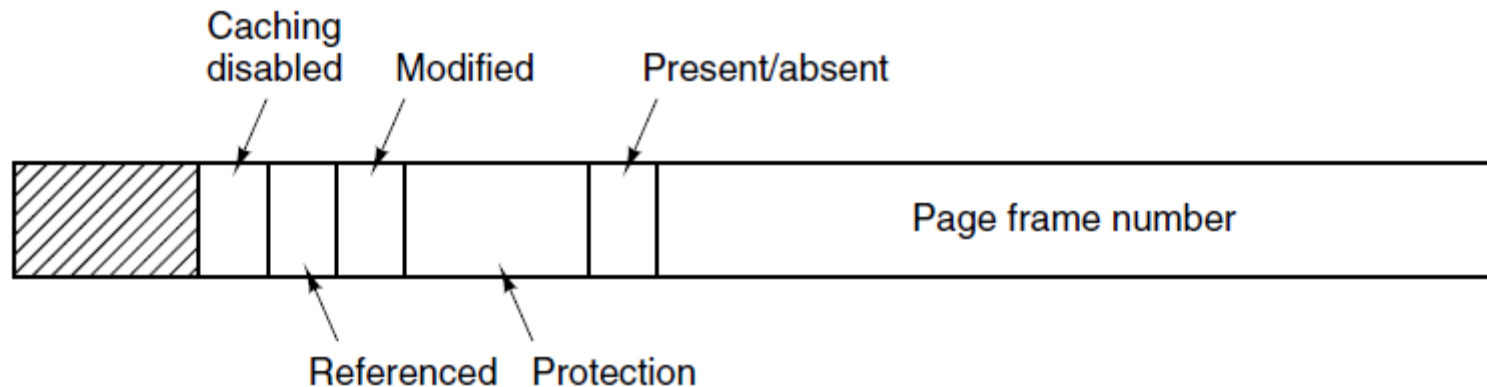
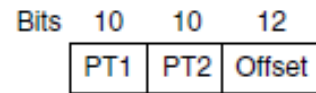


Figure 3-11. A typical page table entry.

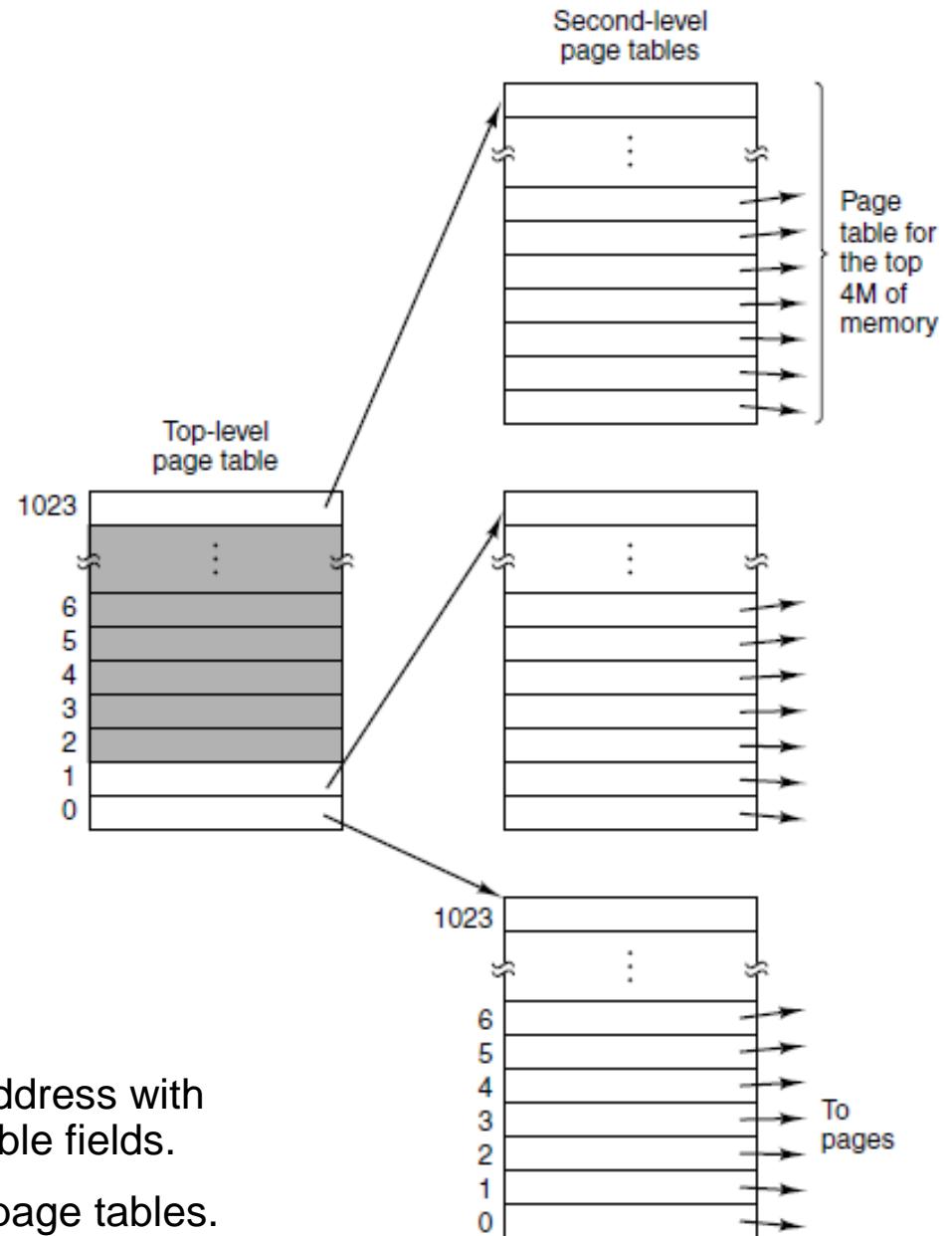
Multilevel Page Tables



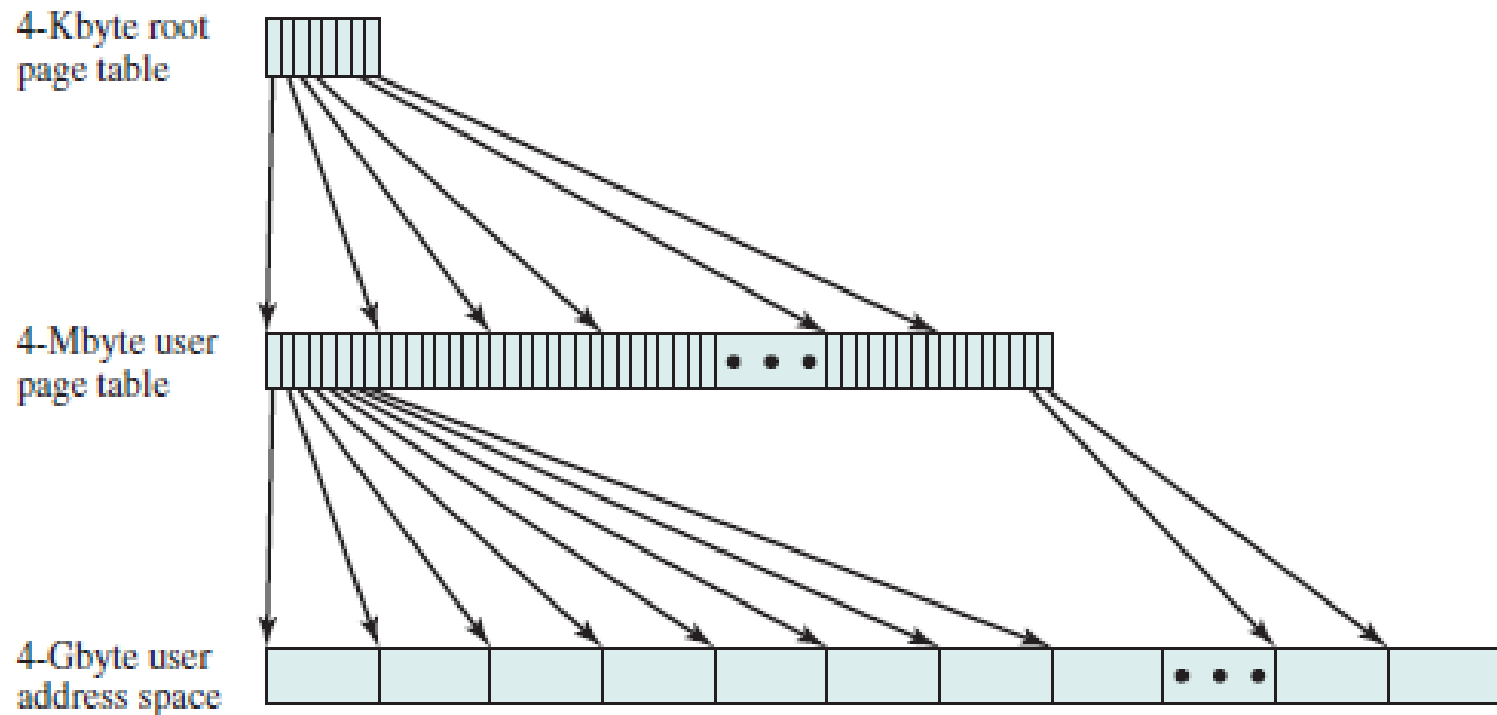
(a)

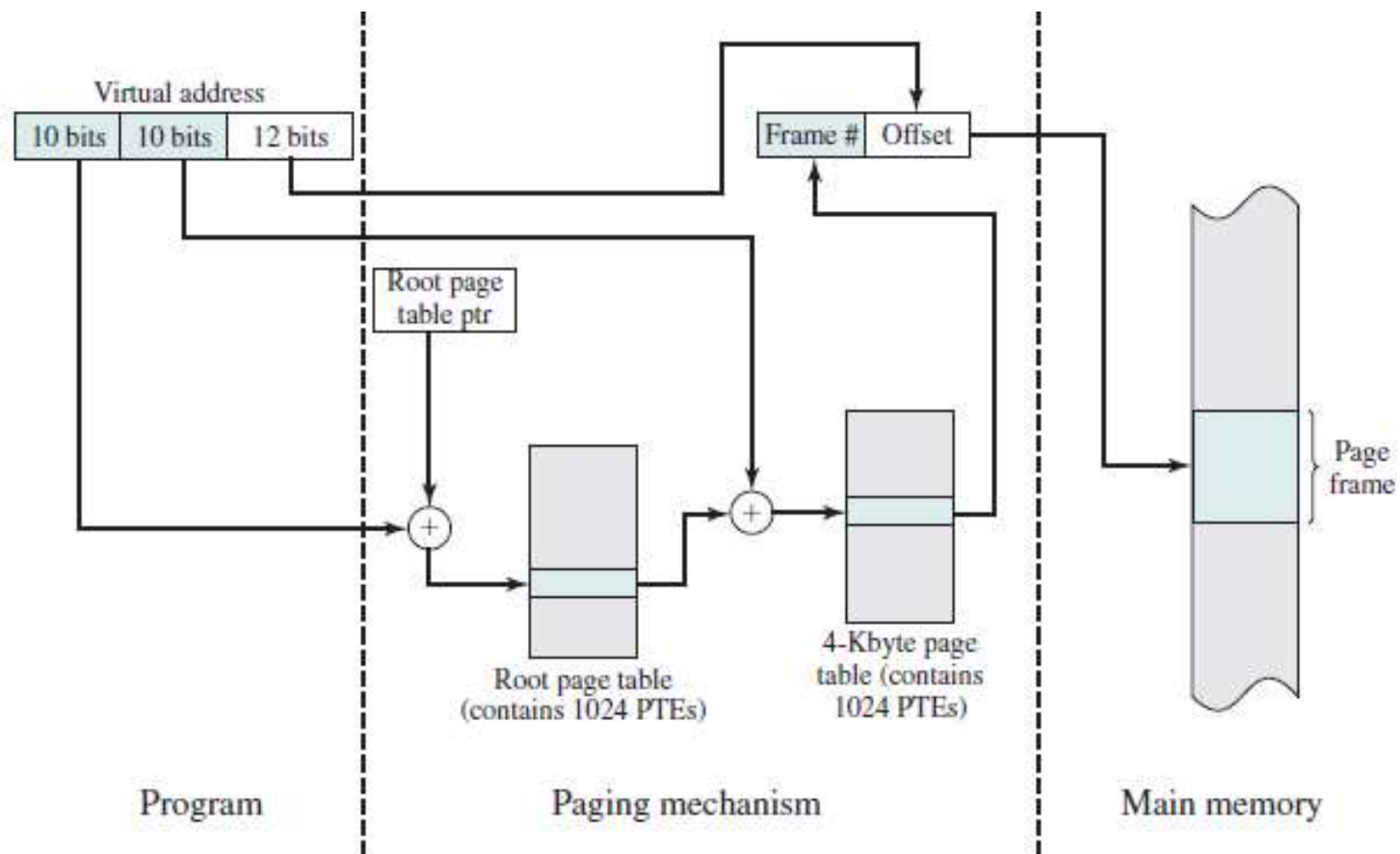
(a) A 32-bit address with two page table fields.

(b) Two-level page tables.



Page table hierarkhikal 2 level





■ 4206596 – 00 0000 0001 00 0000 0011 0000 0000
0100

- PT 1 = 1
- PT 2 = 3
- Offset = 4

Speeding Up Paging

Major issues faced:

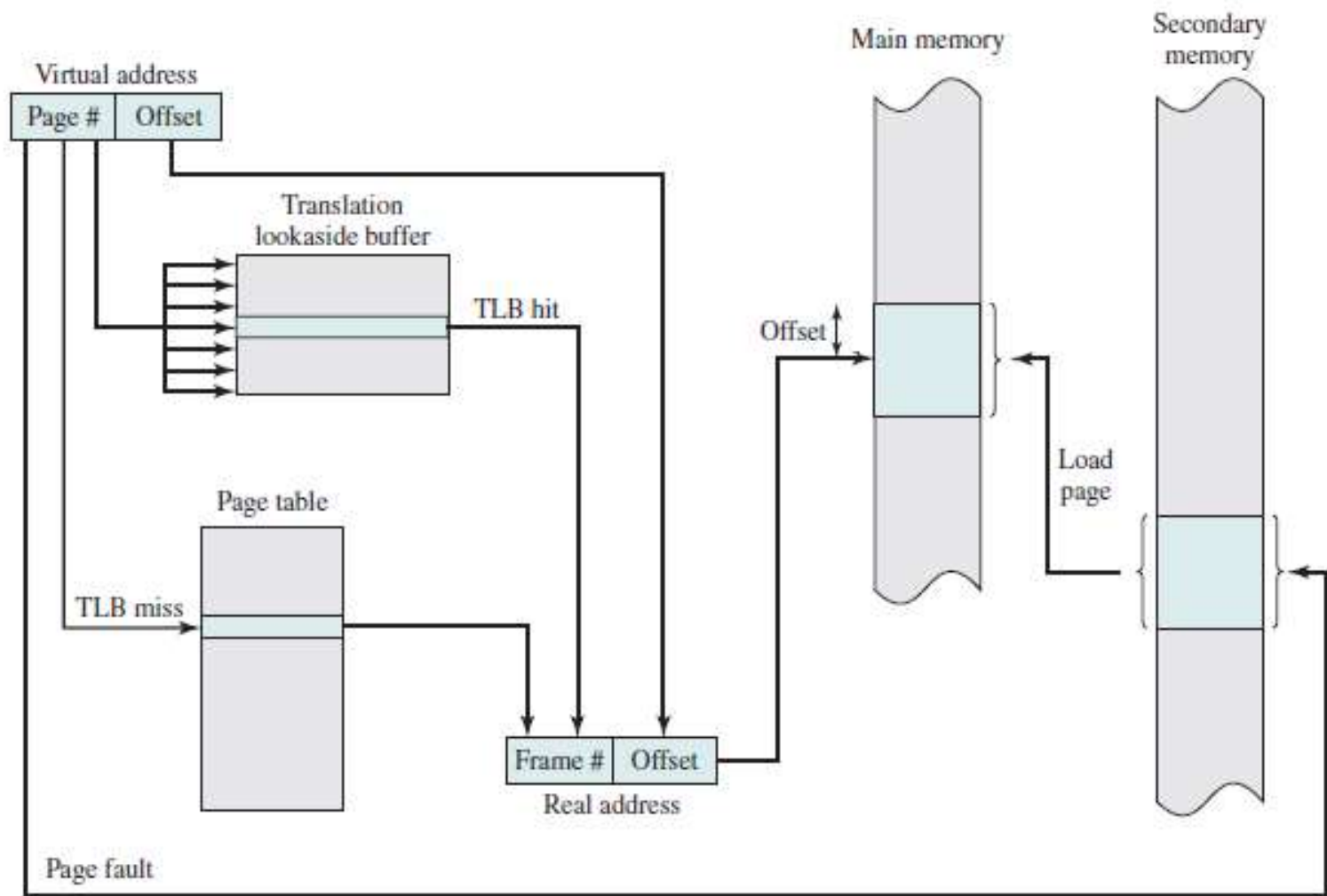
1. The mapping from virtual address to physical address must be fast.
2. If the virtual address space is large, the page table will be large.

Translation Lookaside Buffers

Valid	Virtual page	Modified	Protection	Page frame
1	140	1	RW	31
1	20	0	R X	38
1	130	1	RW	29
1	129	1	RW	62
1	19	0	R X	50
1	21	0	R X	45
1	860	1	RW	14
1	861	1	RW	75

Figure 3-12. A TLB to speed up paging.

Menggunakan TLB



Inverted Page Tables

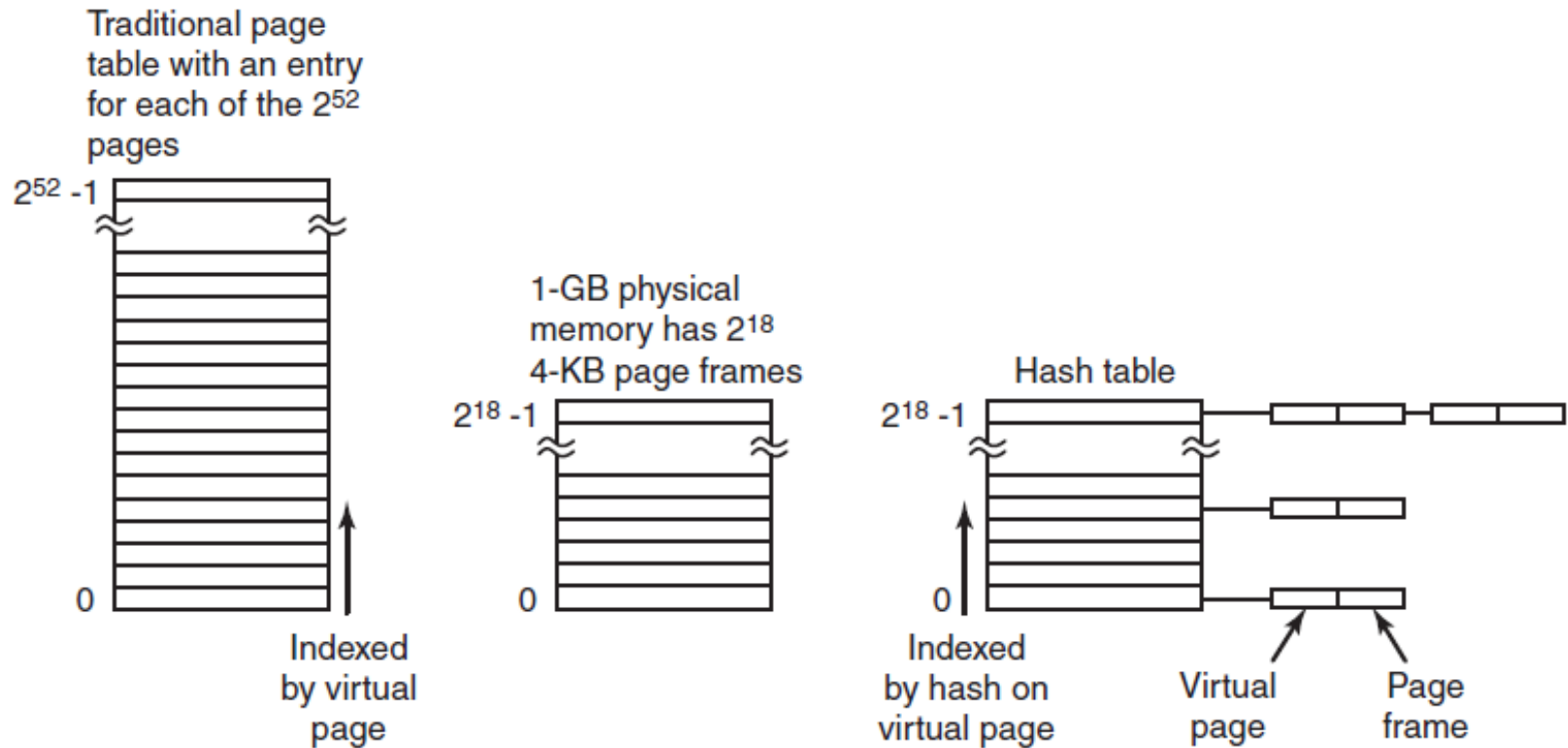


Figure 3-14. Comparison of a traditional page table with an inverted page table.

Page Replacement Algorithms

- Optimal algorithm
- Not recently used algorithm
- First-in, first-out (FIFO) algorithm
- Second-chance algorithm
- Clock algorithm
- Least recently used (LRU) algorithm
- Working set algorithm
- WSClock algorithm

Page Replacement Algorithms

- Page fault forces choice
 - which page must be removed
 - make room for incoming page
- Modified page must first be saved
 - unmodified just overwritten
- Better not to choose an often used page
 - will probably need to be brought back in soon

Optimal Page Replacement Algorithm

- Replace page needed at the farthest point in future
 - Optimal but unrealizable
- Estimate by ...
 - logging page use on previous runs of process
 - although this is impractical

Not Recently Used Page Replacement Algorithm (NRU)

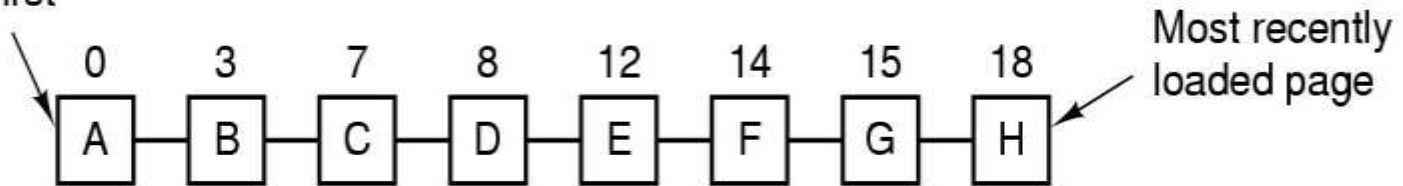
- Each page has Reference bit, Modified bit
 - bits are set when page is referenced, modified
- Pages are classified
 1. not referenced, not modified
 2. not referenced, modified
 3. referenced, not modified
 4. referenced, modified
- NRU removes page at random
 - from lowest numbered non empty class

FIFO Page Replacement Algorithm

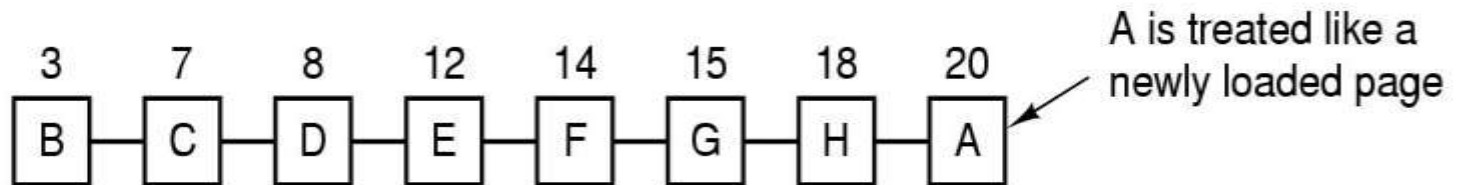
- Maintain a linked list of all pages
 - in order they came into memory
- Page at beginning of list replaced
- Disadvantage
 - page in memory the longest may be often used

Second Chance Page Replacement Algorithm

Page loaded first



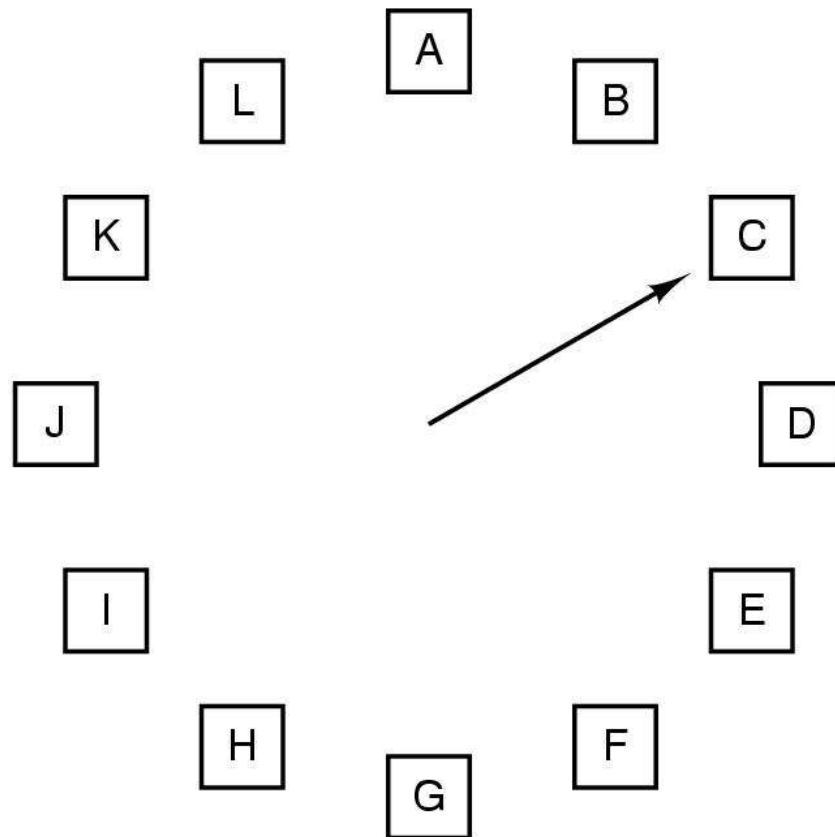
(a)



(b)

- 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)

The Clock Page Replacement Algorithm



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


Least Recently Used (LRU)

- Assume pages used recently will be used again soon
 - throw out page that has been unused for longest time
- Must keep a linked list of pages
 - most recently used at front, least at rear
 - update this list every memory reference !!
- Alternatively keep counter in each page table entry
 - choose page with lowest value counter
 - periodically zero the counter

Simulating LRU in Software (1)

	Page					Page					Page					Page					Page			
	0	1	2	3		0	1	2	3		0	1	2	3		0	1	2	3		0	1	2	3
0	0	1	1	1		0	0	1	1		0	0	0	1		0	0	0	0		0	0	0	0
1	0	0	0	0		1	0	1	1		1	0	0	1		1	0	0	0		1	0	0	0
2	0	0	0	0		0	0	0	0		1	1	0	1		1	1	0	0		1	1	0	1
3	0	0	0	0		0	0	0	0		0	0	0	0		1	1	1	0		1	1	0	0
(a)					(b)					(c)					(d)					(e)				
	0	0	0	0		0	1	1	1		0	1	1	0		0	1	0	0		0	1	0	0
	1	0	1	1		0	0	1	1		0	0	1	0		0	0	0	0		0	0	0	0
	1	0	0	1		0	0	0	1		0	0	0	0		1	1	0	1		1	1	0	0
	1	0	0	0		0	0	0	0		1	1	1	0		1	1	0	0		1	1	1	0
(f)					(g)					(h)					(i)					(j)				

LRU using a matrix – pages referenced in order
0,1,2,3,2,1,0,3,2,3



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

0,1,0,4 (page fault),1,5

Simulating LRU in Software (2)

	R bits for pages 0-5, clock tick 0	R bits for pages 0-5, clock tick 1	R bits for pages 0-5, clock tick 2	R bits for pages 0-5, clock tick 3	R bits for pages 0-5, clock tick 4
	1 0 1 0 1 1	1 1 0 0 1 0	1 1 0 1 0 1	1 0 0 0 1 0	0 1 1 0 0 0
Page					
0	10000000	11000000	11100000	11110000	01111000
1	00000000	10000000	11000000	01100000	10110000
2	10000000	01000000	00100000	00100000	10001000
3	00000000	00000000	10000000	01000000	00100000
4	10000000	11000000	01100000	10110000	01011000
5	10000000	01000000	10100000	01010000	00101000
	(a)	(b)	(c)	(d)	(e)

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

- Jika status R dari page 0,1,2,3,4,5 pada setiap clock tick
- Clock tick 5 : 101010
- Clock tick 6 : 110011
- Clock tick 7 : 001100
- Clock tick 8 : 101001
- Page berapa yang akan di-remove ssdh clock tick ke-8

Working Set Replacement Algorithm (1)

- Purest form of paging, process are started up with none of their pages in memory.
- Pages are loaded only on demand, not in advance → **demand paging** → many page fault
- Most process do not work this way. They exhibit a **locality of reference**, the process references only a relatively small fraction of its pages
- Set of page that a process is currently using is known as its **working set**

Working Set Replacement Algorithm (2)

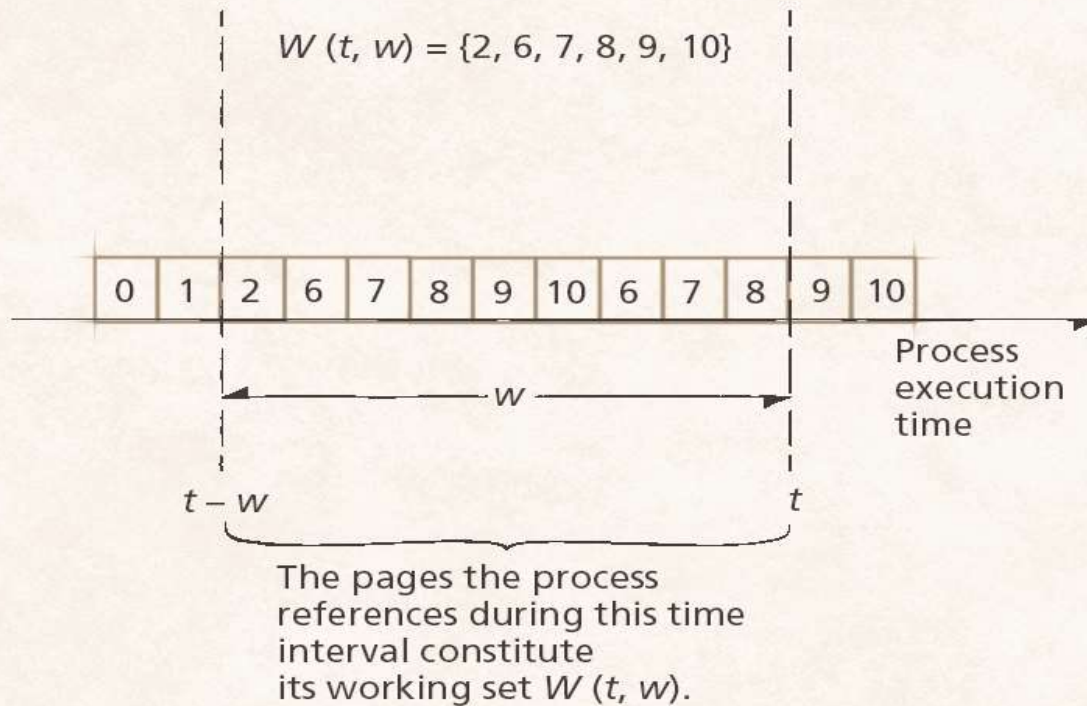
- A program causing page faults every few instructions is said to be **thrashing**
- Many paging system try to keep track of each process's working set and make sure that it is in memory before letting the process run → this approach is called the **working set model**

Working Set Replacement Algorithm (3)

- Loading the pages before letting processes run is also called **prepaging**
- At any instant of time, t , there exist a set of consisting of all the pages used by the k most recent memory references—this set, $w(k,t)$ is **the working set**
- To implement the working set model, it is necessary for the operating system to keep track of which pages are in the working set.
- When a page fault occurs, find a page not in working set and evict it.
- So..we need a precise way of determining which pages are in the working set
 - Working set is the set of pages used in the k most recent memory references

Working Set Model

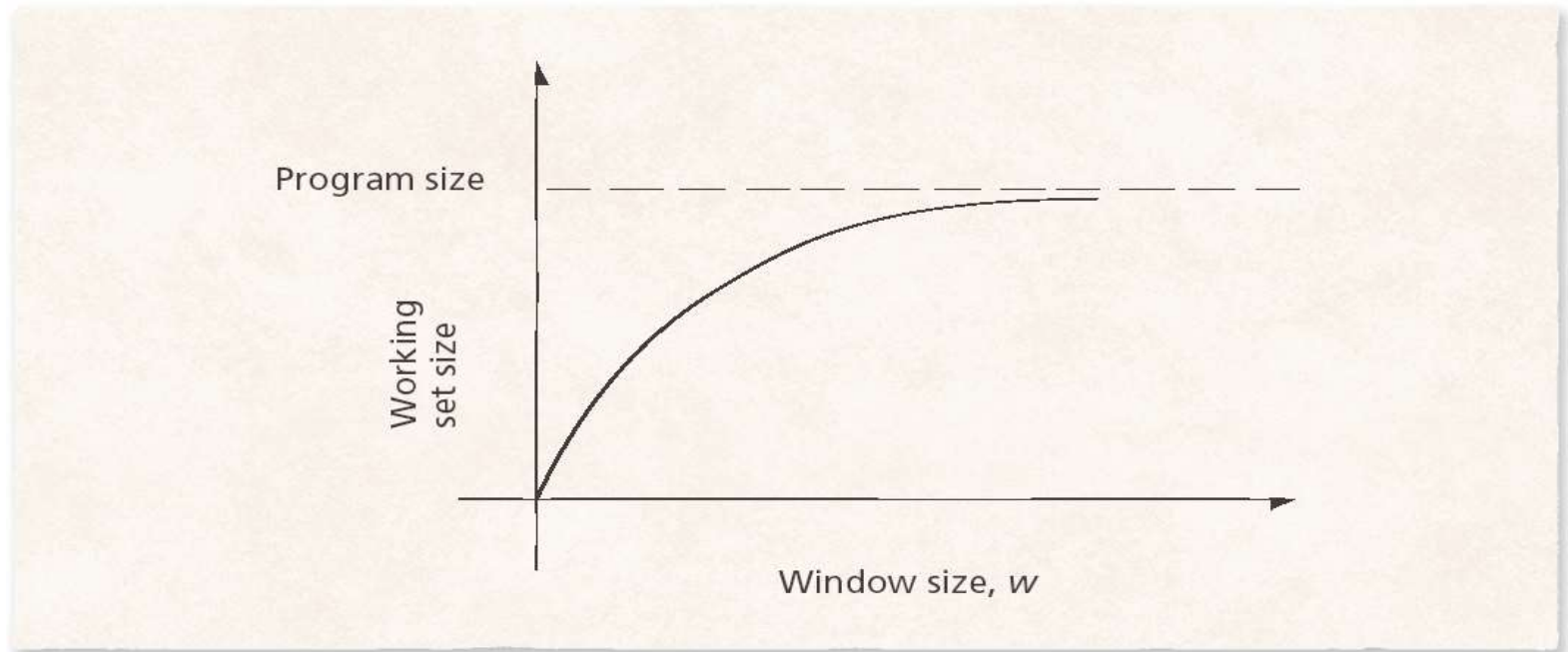
The process's working set of pages, $W(t, w)$, is the set of pages referenced by the process during the process-time interval $t - w$ to t .



Working Set Model

- The size of the process's working set increases asymptotically to the process's program size as its working set window increases

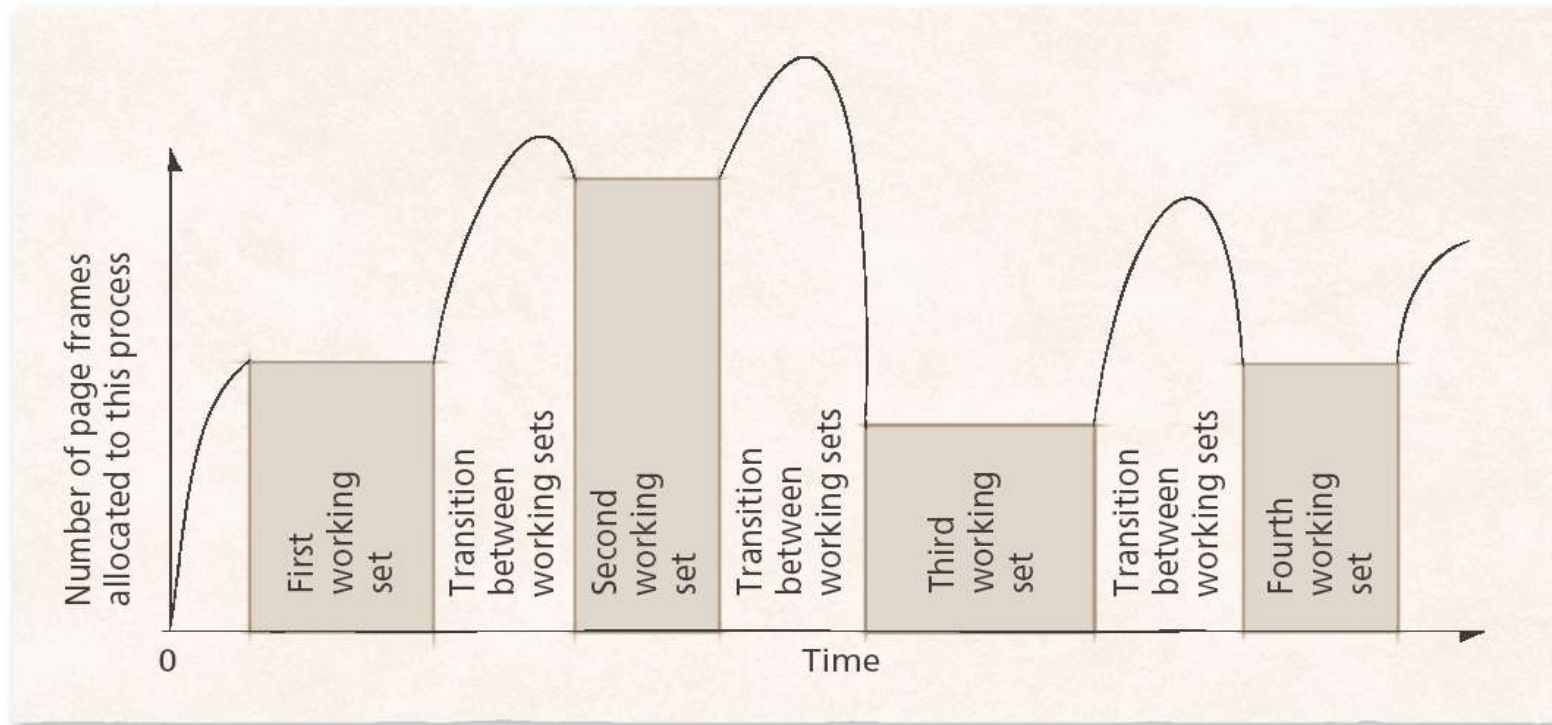
Working Set Model



Working Set Model

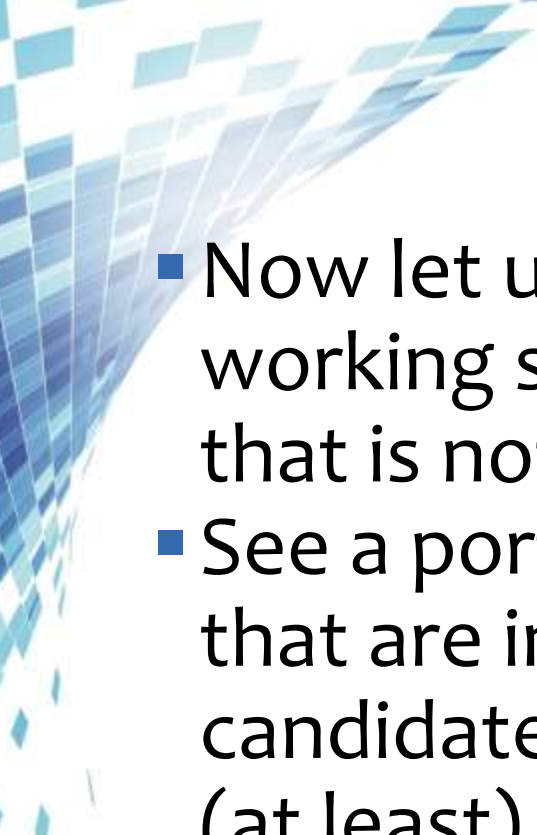
- As a process transitions between working sets, the system temporarily maintains in memory pages that are no longer in the process's current working set
 - Goal of working set memory management is to reduce this misallocation

Working Set Model

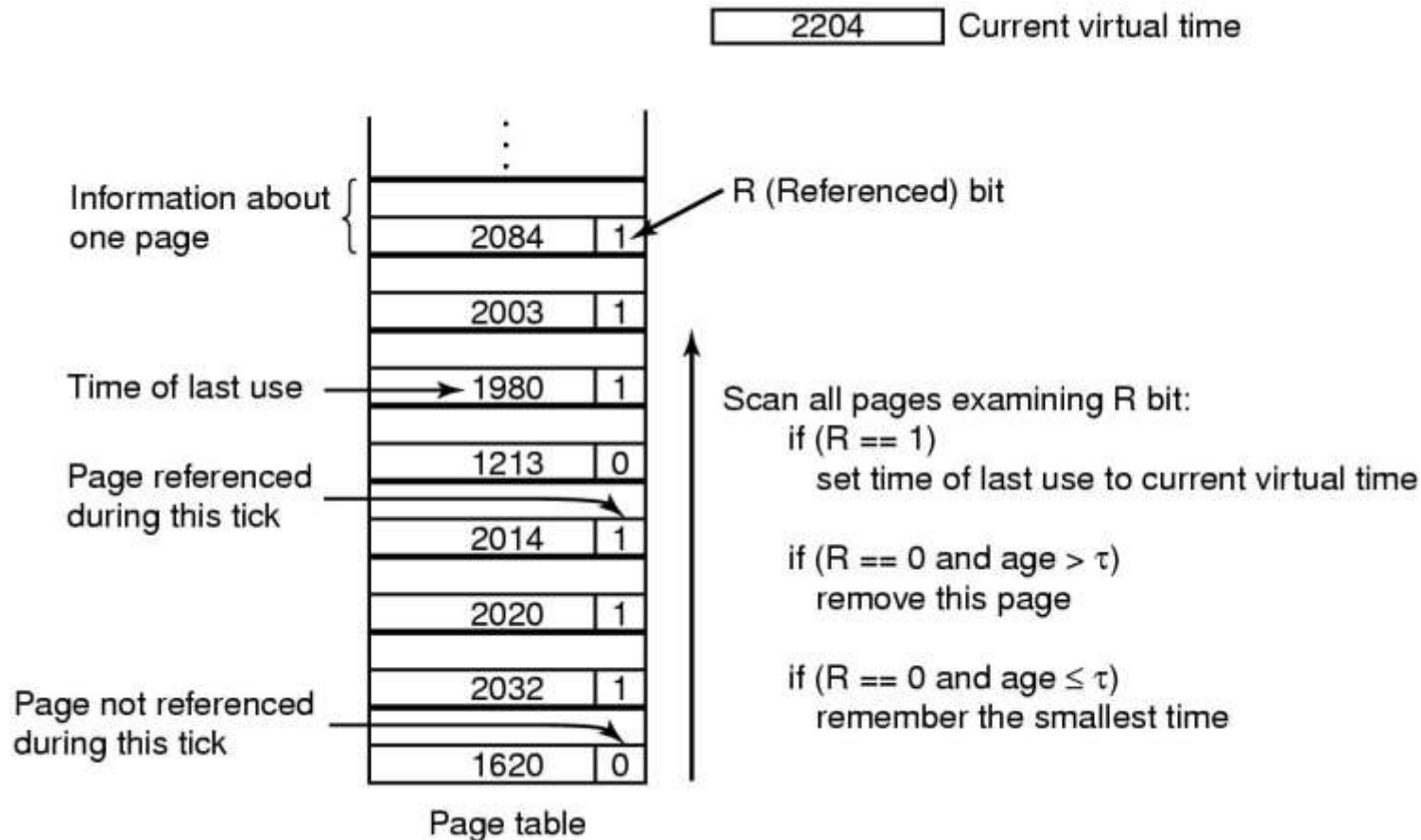


Working Set Replacement Algorithm

- In practice, if a process start running at time T and has had 40 msec of CPU time at real time $T + 100$ msec, for working set purposes its time is 40 msec
- The amount of CPU time a process has actually used since it started is often called its **current virtual time**
- With this appox, the working set of a process is a the set of pages it has referenced during the past τ seconds of virtual time

- 
- Now let us look at algorithm based on the working set. The basic idea is to find a page that is not in the working set and evict it.
 - See a portion of page table. Only pages that are in memory are considered as candidate for eviction. Each entry contain (at least) two key items of information: the (approx) time the page was last used and the R bit.

The Working Set Page Replacement Algorithm (2)



The working set algorithm

Working Set Replacement Algorithm

The algorithm works as follows

- **Assumption:**

- R and M bit is set by hardware
- A periodic clock interrupt, cleaning the R bit on every clock tick
- On every page fault, the page table is scanned to look for a suitable page to evict

Working Set Replacement Algorithm

The algorithm works as follows (2)

- As each entry is processed, the R bit is examined
 - If $R=1$, the current virtual time is written into the Time of Last Use field in the page table (indicating that the page was in use at the time the fault occurred)
 - The page has been referenced during current clock tick, it is in the working set and is not candidate for removal

Working Set Replacement Algorithm

The algorithm works as follows (3)

- If $R=0$ (the page has not been referenced during the current clock tick and may be a candidate for removal; the its age (the current virtual time – Time of Last use) is computed and compared to τ ;
 - If the age is greater than τ (the age $> \tau$), the page no longer the working set and the new page replace it; the scan continues updating the remaining entries
 - If the age $\leq \tau$, the page is still in the working set
 - The page with the greatest age is noted (is can

Working Set Replacement Algorithm

The algorithm works as follows (4)

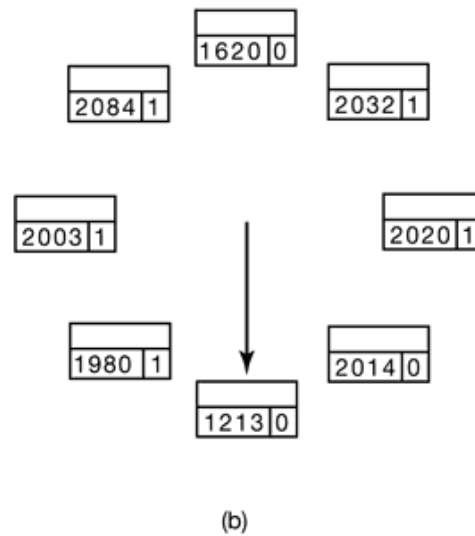
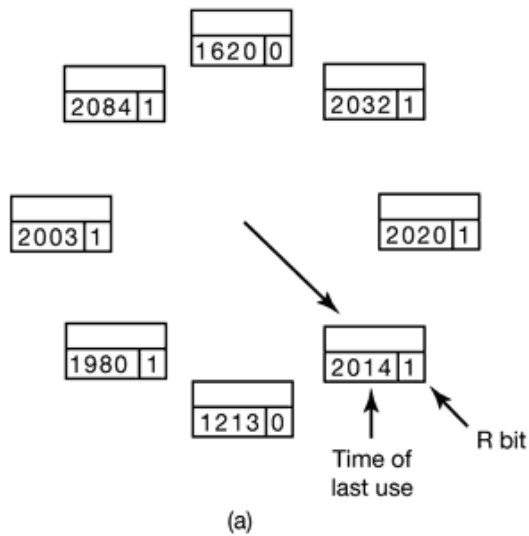
- If the entire table is scanned without finding a candidate to evict, that all pages are in the working set
 - If one or more pages with $R=0$ were found, the one with the greatest age is evicted
 - If all pages have $R=1$, so one is chose at random for removal, prefably a clean page if one exists

The WSClock Page Replacement Algorithm

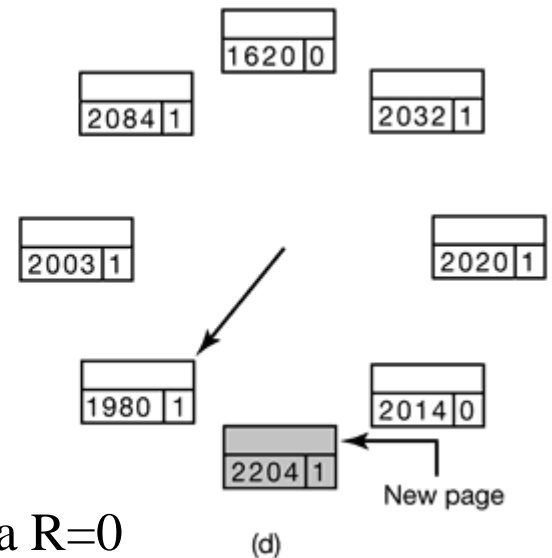
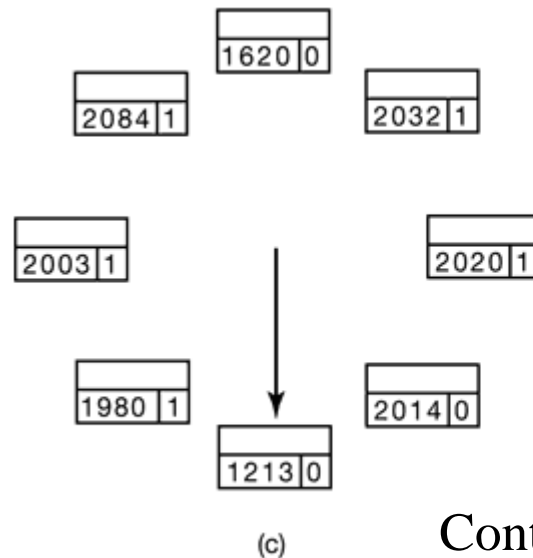
- The data structure needed is a circular list of page frames
- Initially, this list is empty. When the first page is loaded, it is added to the list. As more pages are added, they go into the list to form a ring.
- Each entry contains the *Time of last use* field from the basic working set algorithm, as well as the *R* bit (shown) and the *M* bit

The WSClock Page Replacement Algorithm

2204 Current virtual time



Contoh jika $R=1$



Contoh jika $R=0$

WSClock Replacement Algorithm

The algorithm works as follows

- As with the clock algorithm, at each page fault the page pointed to by the hand is examined first. If the R bit is set to 1, the page has been used during the current tick so it is not an ideal candidate to remove.
 - The R bit is then set to 0, the hand advanced to the next page, and the algorithm repeated for that page.
- if the page pointed to has $R = 0$; If the age is greater than τ and the page is clean, it is not in the working set and a valid copy exists on the disk. The page frame is simply claimed and the new page put there
- On the other hand, if the page is dirty, it cannot be claimed immediately since no valid copy is present on disk.
- To avoid a process switch, the write to disk is scheduled, but the hand is advanced and the algorithm continues with the next page.
- After all, there might be an old, clean page further down the line that can be used immediately.

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 Frequently Used)	Fairly crude approximation to LRU
Aging	Efficient algorithm that approximates LRU well
Working set	Somewhat expensive to implement
WSClock	Good efficient algorithm

LATIHAN

- Sebuah komputer memiliki 6 page frame. Tabel berikut menunjukkan masing-masing page dengan *time of loading* (waktu load ke memori), *time of last access* (waktu terakhir diakses/diacu), *R* bit dan *M* bit (waktu dalam *clock tick*). Page mana yang akan di-replace jika menggunakan algoritma NRU (Not Recently Used), FIFO, LRU (Least Recently Used) dan Second Chance?

Page	Loaded	Last ref.	R	M
1	126	280	1	0
2	230	265	0	1
3	140	270	0	0
4	110	285	1	1
5	120	260	1	0
6	130	278	0	0

Modeling Page Replacement Algorithms

Belady's Anomaly

All pages frames initially empty

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

Youngest page

	0	1	2	3	0	1	4	4	4	2	3	3
		0	1	2	3	0	1	1	1	4	2	2
Oldest page			0	1	2	3	0	0	0	1	4	4

P P P P P P P P P P 9 Page faults

(a)

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

10 Page faults

(b)

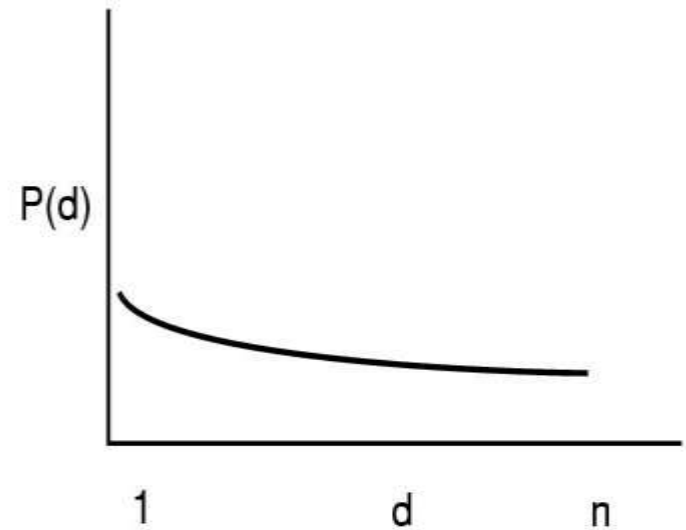
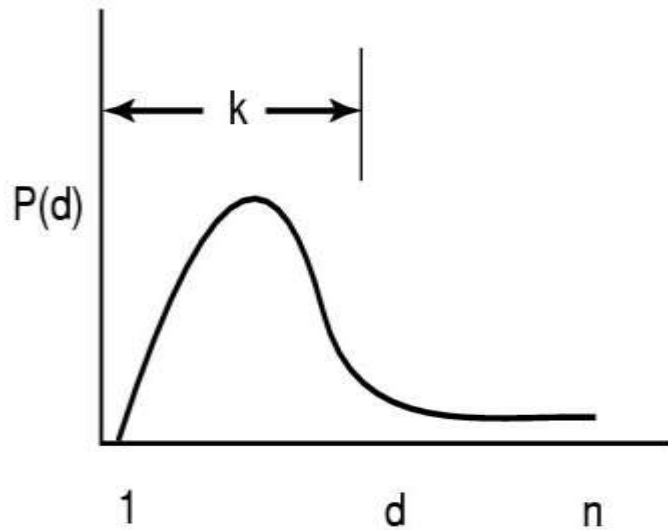
- FIFO with 3 page frames
- FIFO with 4 page frames
- P's show which page references show page faults

Stack Algorithms

Reference string	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	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	0
Page faults	P	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

The Distance String



Probability density functions for two hypothetical distance strings

The Distance String

$C_1 = 4$	← # times 1 occurs in distance string
$C_2 = 2$	
$C_3 = 1$	
$C_4 = 4$	
$C_5 = 2$	
$C_6 = 2$	← # times 6 occurs in distance string
$C_7 = 1$	
$C_\infty = 8$	

(a)

$F_1 = 19$	← $C_2 + C_3 + C_4 + \dots + C_\infty$
$F_2 = 17$	← $C_3 + C_4 + C_5 + \dots + C_\infty$
$F_3 = 16$	← $C_4 + C_5 + C_6 + \dots + C_\infty$
$F_4 = 12$	
$F_5 = 10$	← # of page faults with 5 frames
$F_6 = 10$	
$F_7 = 8$	
$F_\infty = 8$	

(b)

- Computation of page fault rate from distance string
 - the C vector
 - the F vector

Predicting Page Fault Rates

- The distance string also gives an easy way to predict page fault rates for a reference string for memories of different sizes
 - Essentially, obtain a count, C , of the number of times each number in the distance string occurs

- Then compute another vector, F , where each entry F_x is equal to the sum C_{x+1} to C_{∞}
- Mathematically, F is computed as:

$$F_m = \sum_{k=m+1}^n C_k + C_{\infty}$$

$C_1 = 4$
$C_2 = 2$
$C_3 = 1$
$C_4 = 4$
$C_5 = 2$
$C_6 = 2$
$C_7 = 1$
$C_{\infty} = 8$

Predicting Page Fault Rates

$C_1 = 4$
$C_2 = 2$
$C_3 = 1$
$C_4 = 4$
$C_5 = 2$
$C_6 = 2$
$C_7 = 1$
$C_\infty = 8$



$$F_m = \sum_{k=m+1}^n C_k + C_\infty$$



$F_1 = 20$
$F_2 = 18$
$F_3 = 17$
$F_4 = 13$
$F_5 = 11$
$F_6 = 9$
$F_7 = 8$
$F_\infty = 8$

- So, one page frame would result in 20 faults, two would result in 18, three in 13, etc...

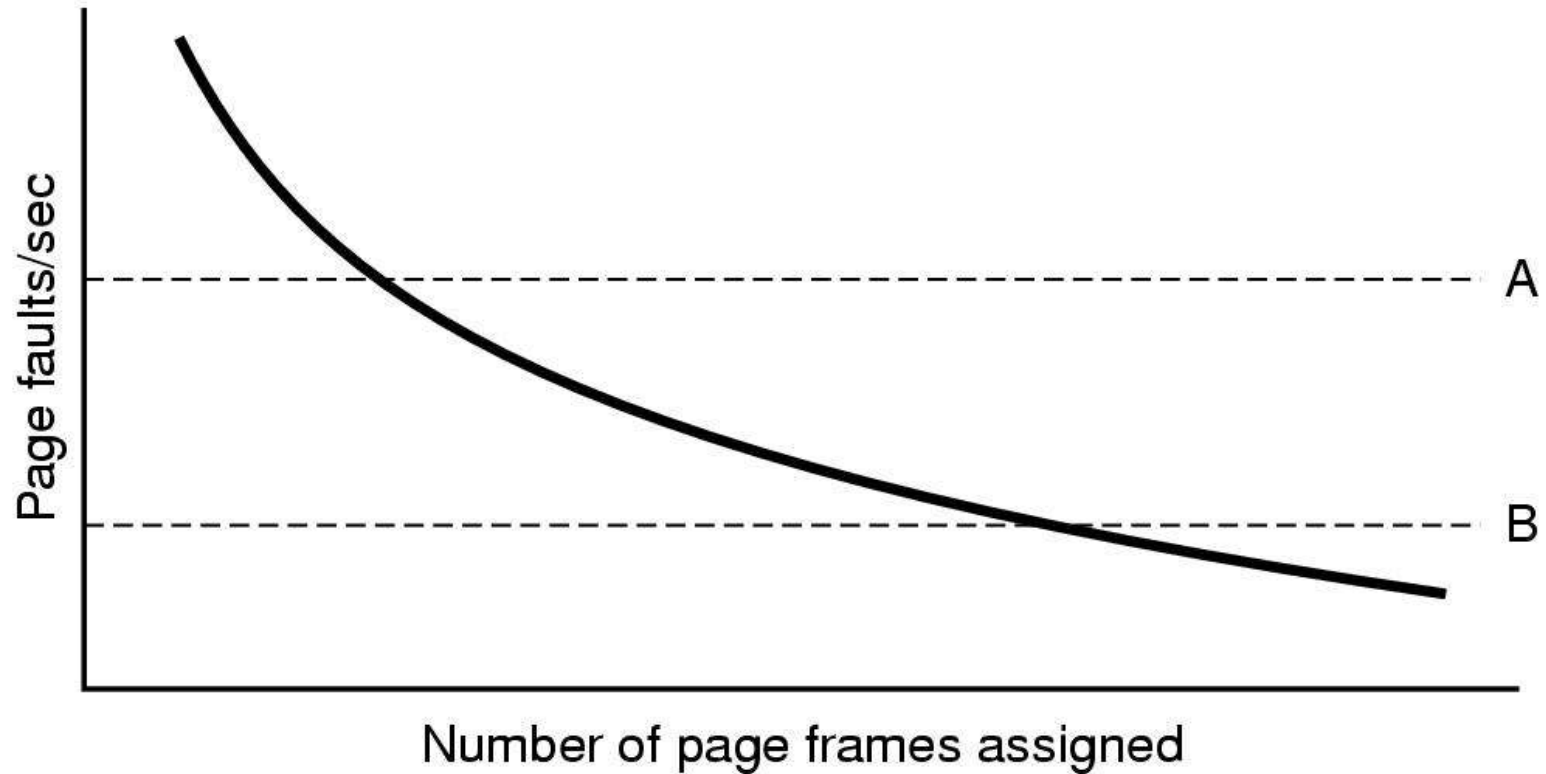
Design Issues for Paging Systems

Local versus Global Allocation Policies (1)

	Age		
A0	10	A0	A0
A1	7	A1	A1
A2	5	A2	A2
A3	4	A3	A3
A4	6	A4	A4
A5	3	A6	A5
B0	9	B0	B0
B1	4	B1	B1
B2	6	B2	B2
B3	2	B3	A6
B4	5	B4	B4
B5	6	B5	B5
B6	12	B6	B6
C1	3	C1	C1
C2	5	C2	C2
C3	6	C3	C3
(a)		(b)	(c)

- Original configuration
- Local page replacement
- Global page replacement

Local versus Global Allocation Policies (2)



Page fault rate as a function of the number of page frames assigned

Load Control

- Despite good designs, system may still thrash
- When PFF algorithm indicates
 - 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)

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)

- Overhead due to page table and internal fragmentation

- Where

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

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

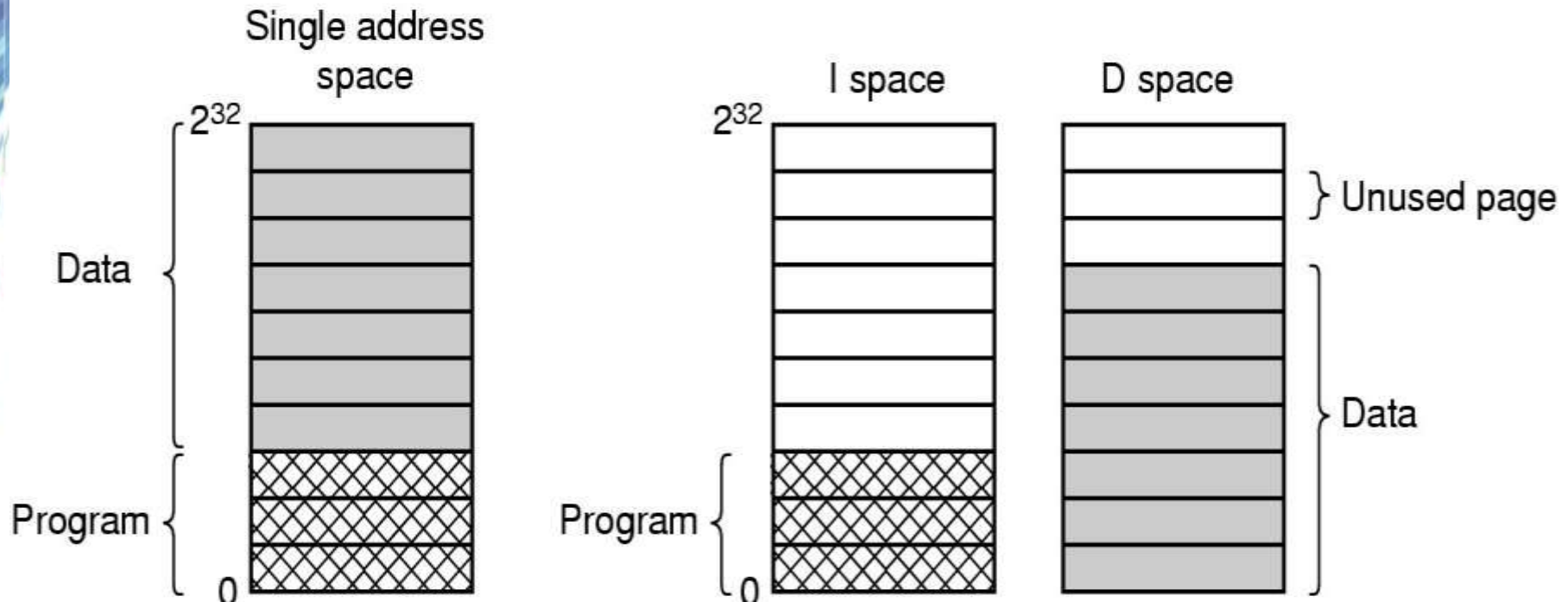
page table space

internal fragmentation

Optimized when

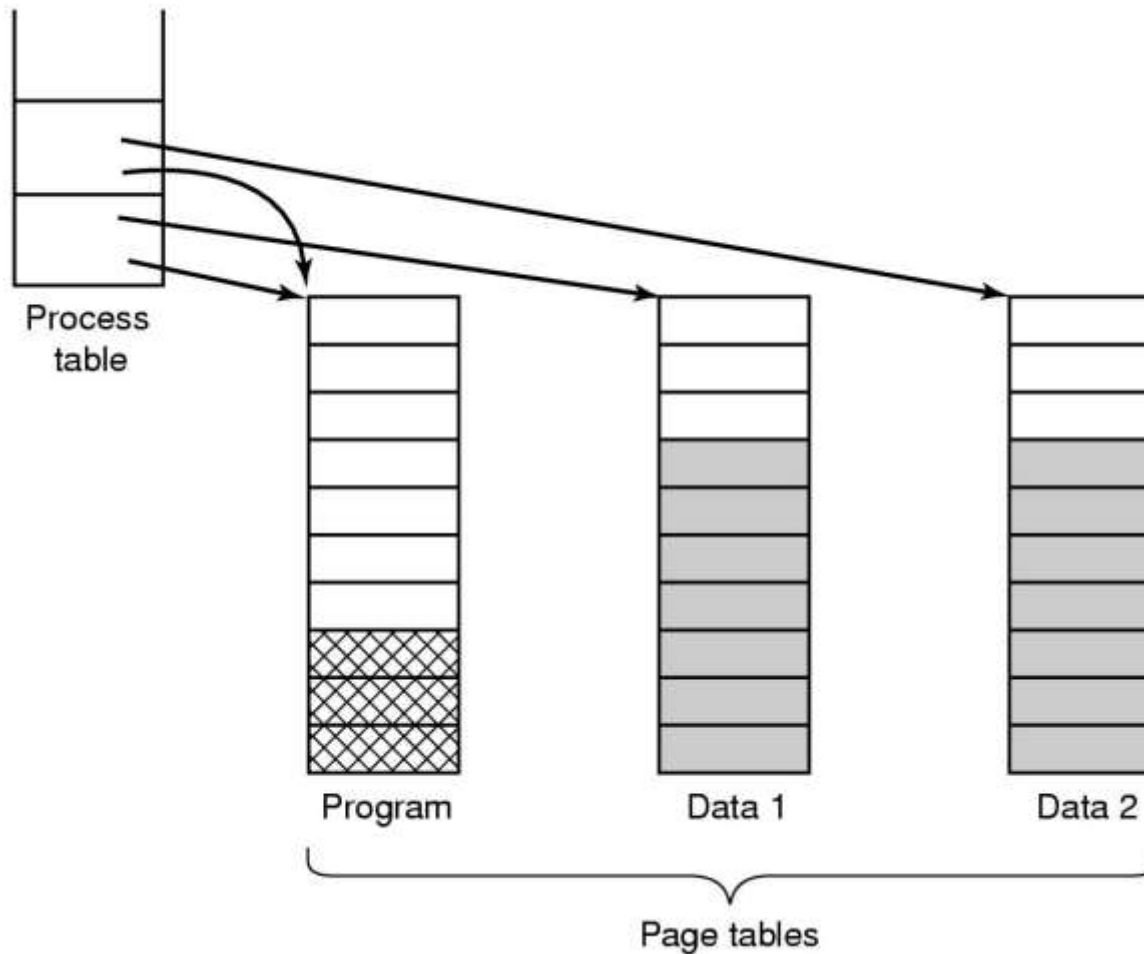
$$p = \sqrt{2se}$$

Separate Instruction and Data Spaces



- One address space
- Separate I and D spaces

Shared Pages



Two processes sharing same program sharing its page table

Cleaning Policy

- 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)
 - as regular page replacement algorithm but with diff ptr

Implementation Issues

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

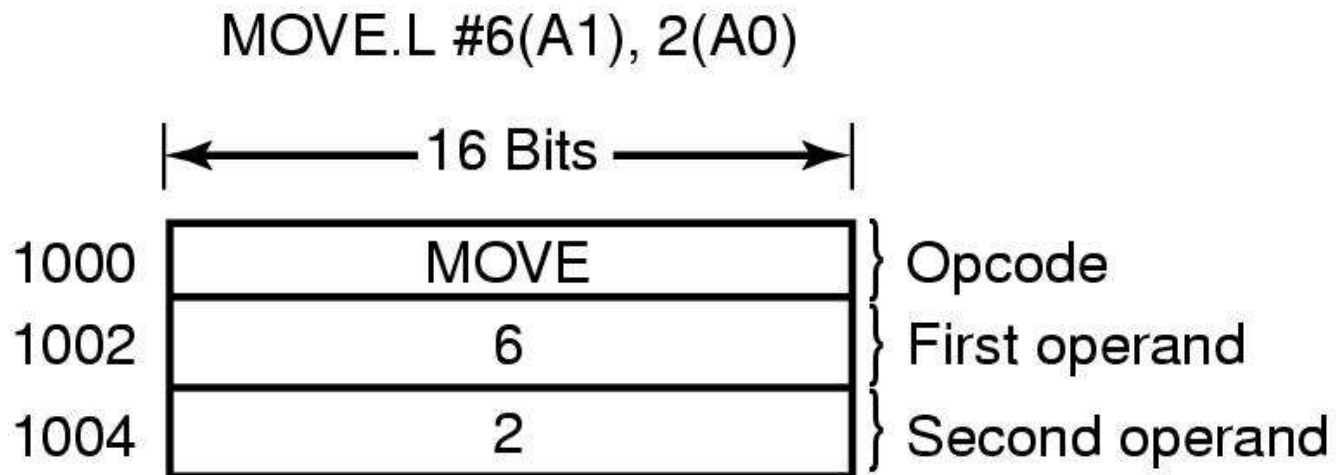
Page Fault Handling (1)

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

Page Fault Handling (2)

- 6. OS brings schedules new page in from disk
- 7. Page tables updated
 - Faulting instruction backed up to when it began
- 6. Faulting process scheduled
- 7. Registers restored
 - Program continues

Instruction Backup

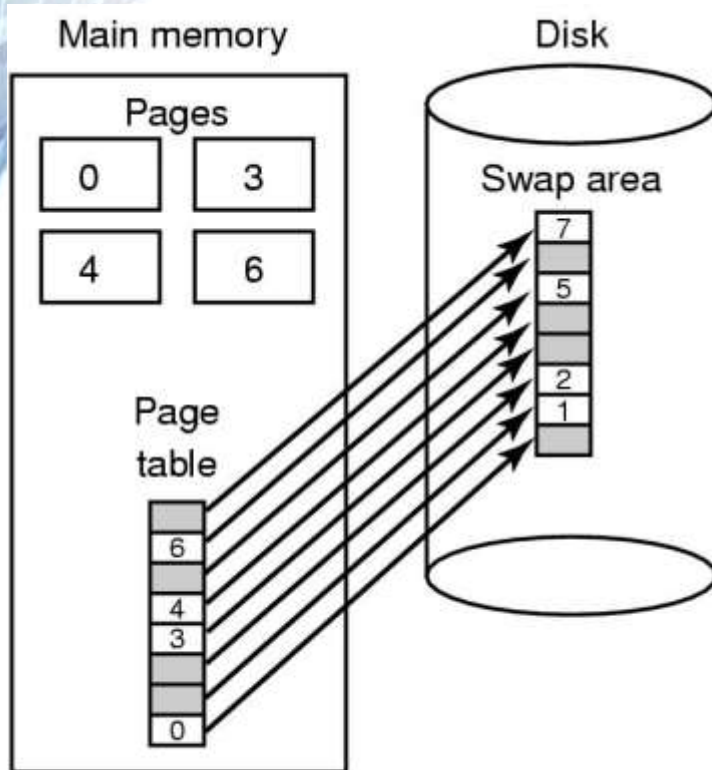


An instruction causing a page fault

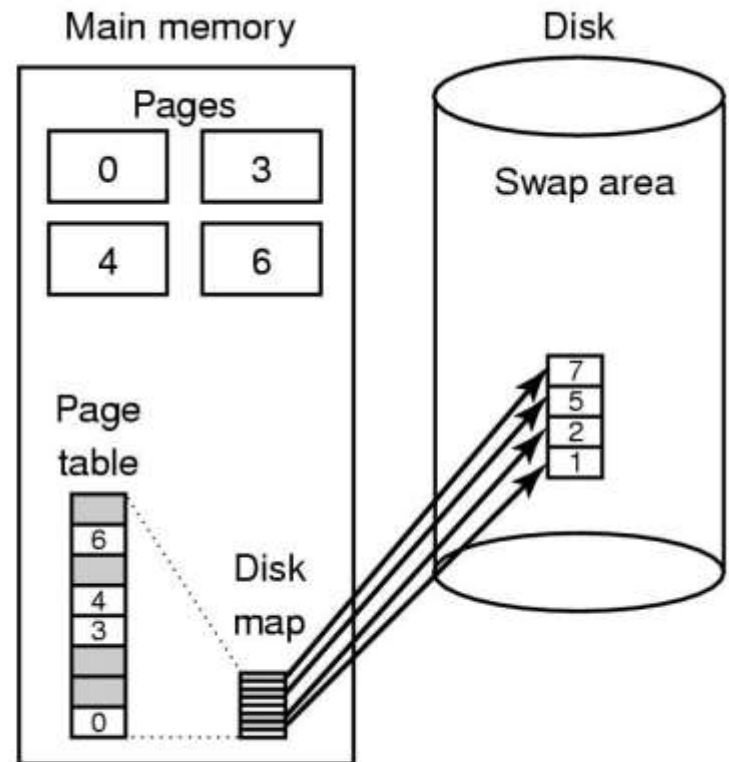
Locking Pages in Memory

- Virtual memory and I/O occasionally interact
- Proc issues call for read from device into buffer
 - while waiting for I/O, another processes starts up
 - has a page fault
 - buffer for the first proc may be chosen to be paged out
- Need to specify some pages locked
 - exempted from being target pages

Backing Store



(a)

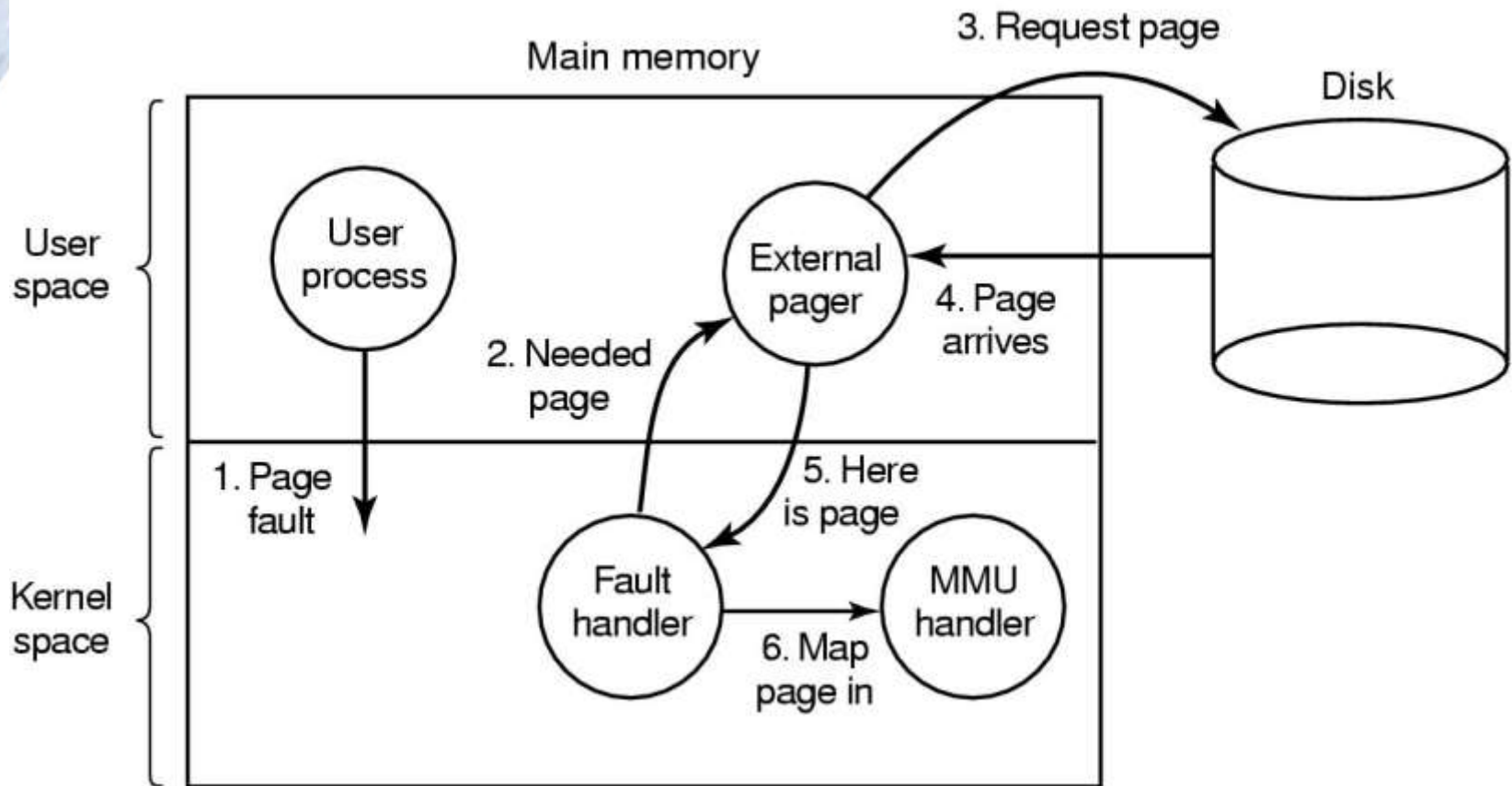


(b)

(a) Paging to static swap area

(b) Backing up pages dynamically

Separation of Policy and Mechanism



Page fault handling with an external pager

Latihan

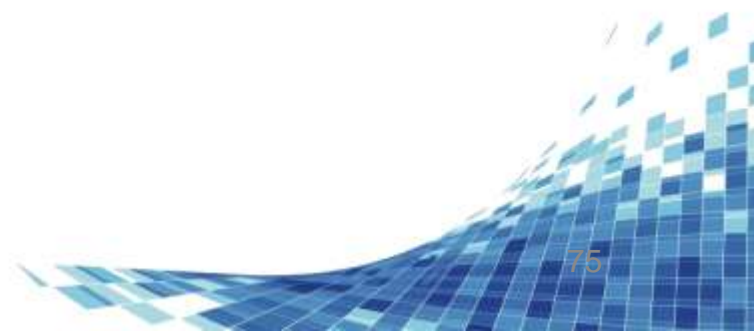
- Data pemakaian memori berukuran 1 MB adalah sbb. :

Alamat awal	ditempati oleh	Ukuran
0	Sistem operasi	312 KB
312 KB	Proses 1	8 KB
320 KB	Proses 2	32 KB
384 KB	Proses 3	120 KB

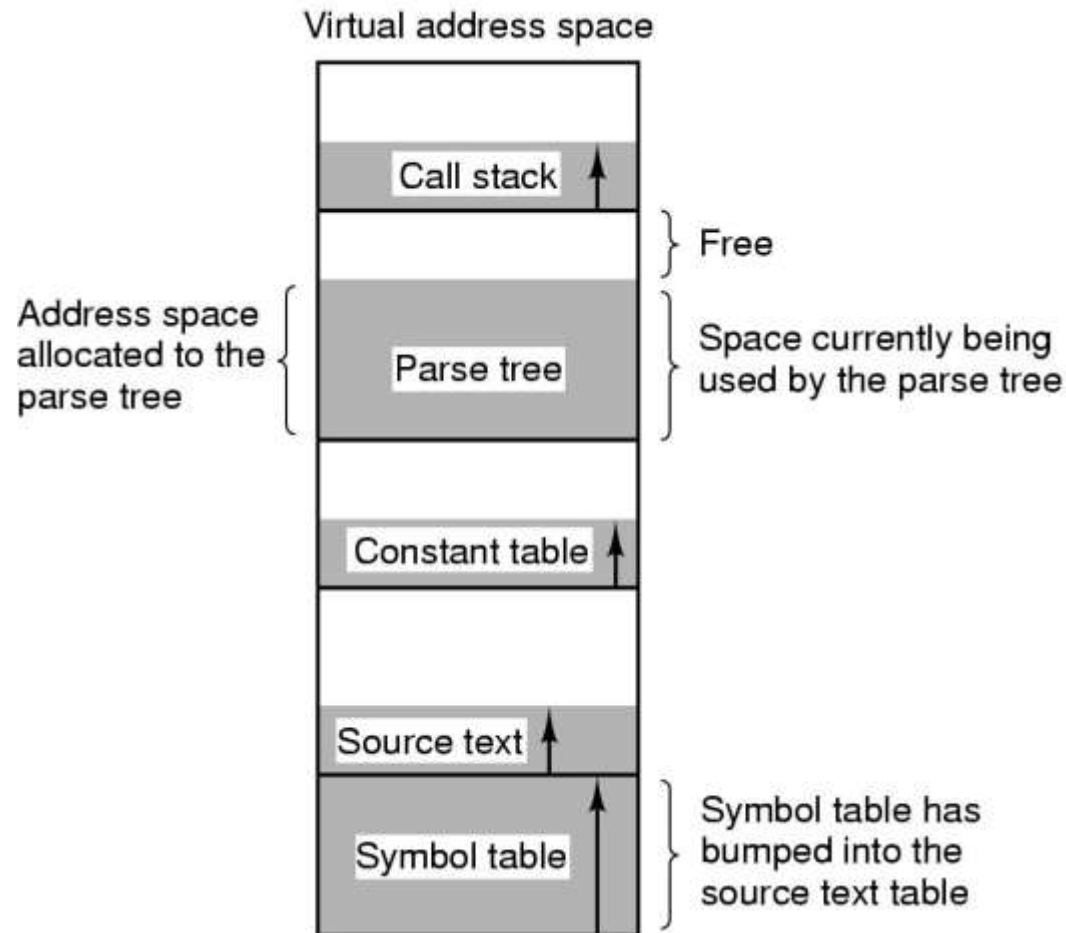
- Bila kemudian secara berturutan akan datang proses 4 dengan kebutuhan memori 24 KB, proses 5 dengan kebutuhan memori 128 KB, dan proses 6 dengan kebutuhan memori 256 KB, bagaimana alokasi memori setelah proses-proses tersebut berdatangan dengan algoritma penempatan :

- a. First Fit b. Best Fit c. Next Fit d. Worst Fit

- Petunjuk : Tunjukkan ada berapa partisi setelah ketiga proses tersebut ditempatkan ukuran setiap partisi dan alamatnya, dan status setiap partisi tersebut (terpakai atau kosong).

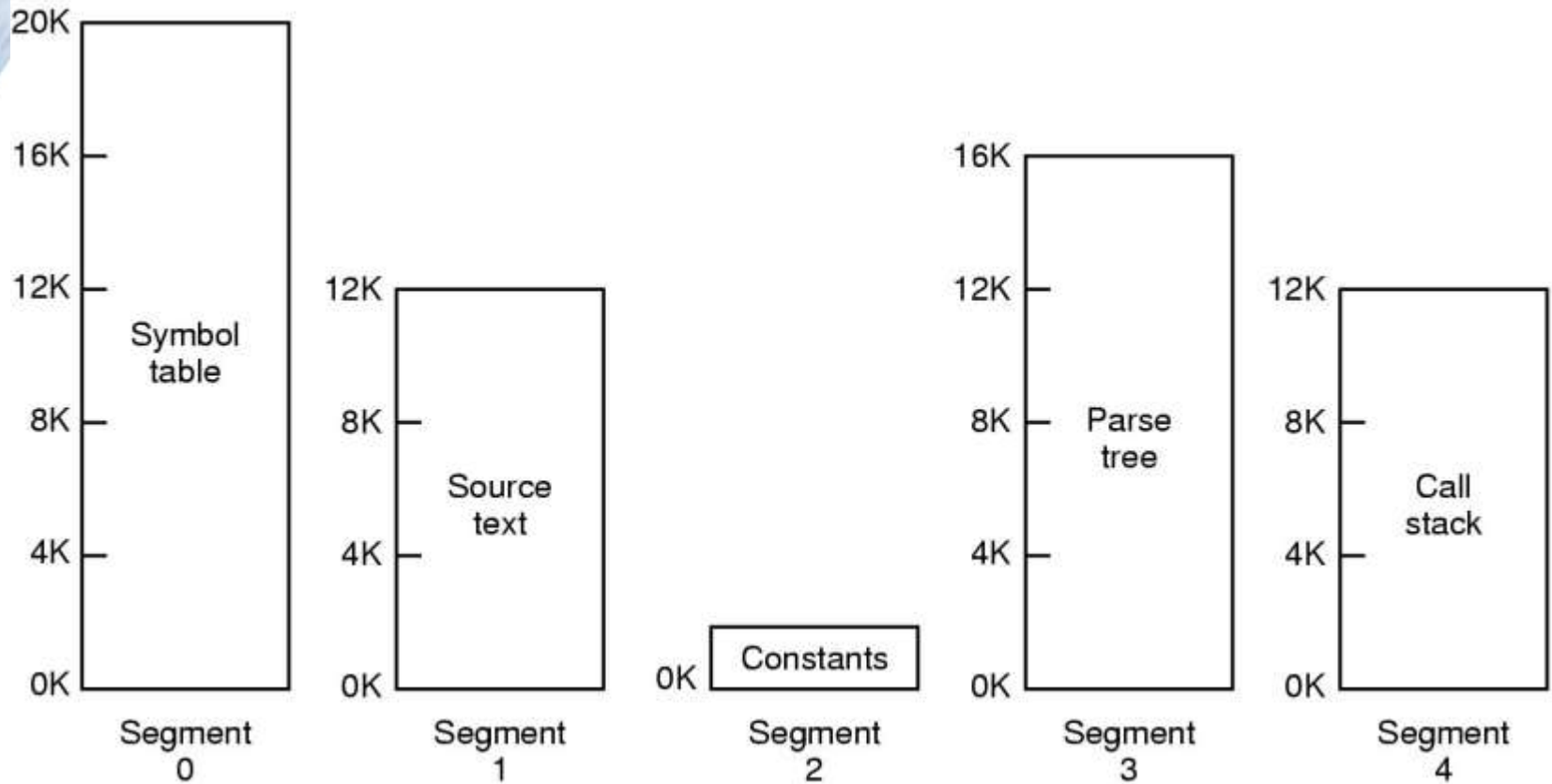


Segmentation (1)



- One-dimensional address space with growing tables
- One table may bump into another

Segmentation (2)



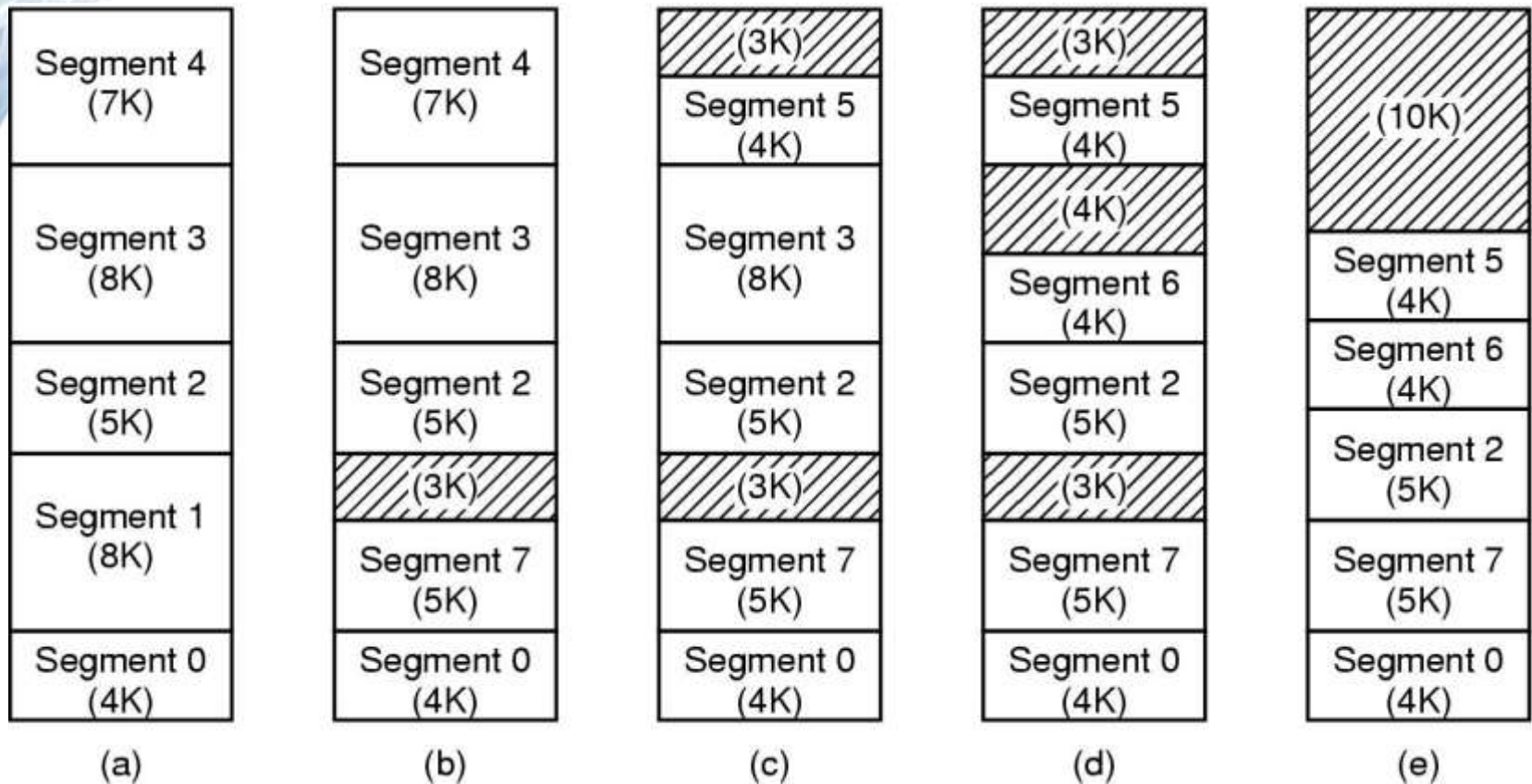
Allows each table to grow or shrink, independently

Segmentation (3)

Consideration	Paging	Segmentation
Need the programmer be aware that this technique is being used?	No	Yes
How many linear address spaces are there?	1	Many
Can the total address space exceed the size of physical memory?	Yes	Yes
Can procedures and data be distinguished and separately protected?	No	Yes
Can tables whose size fluctuates be accommodated easily?	No	Yes
Is sharing of procedures between users facilitated?	No	Yes
Why was this technique invented?	To get a large linear address space without having to buy more physical memory	To allow programs and data to be broken up into logically independent address spaces and to aid sharing and protection

Comparison of paging and segmentation

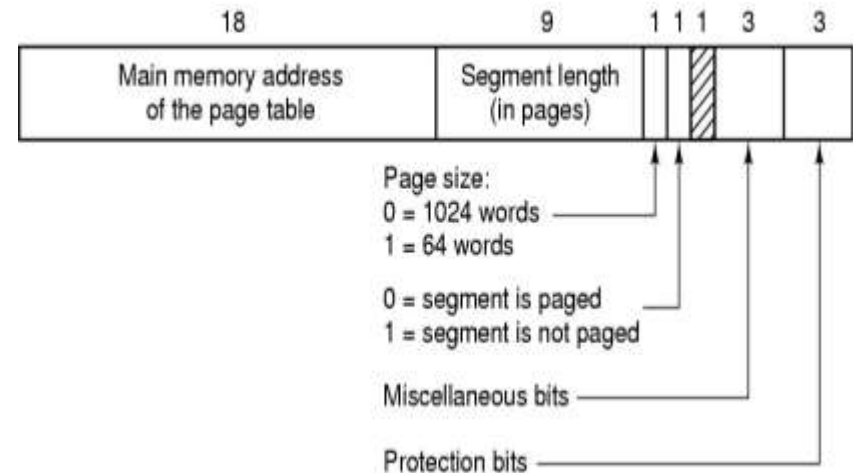
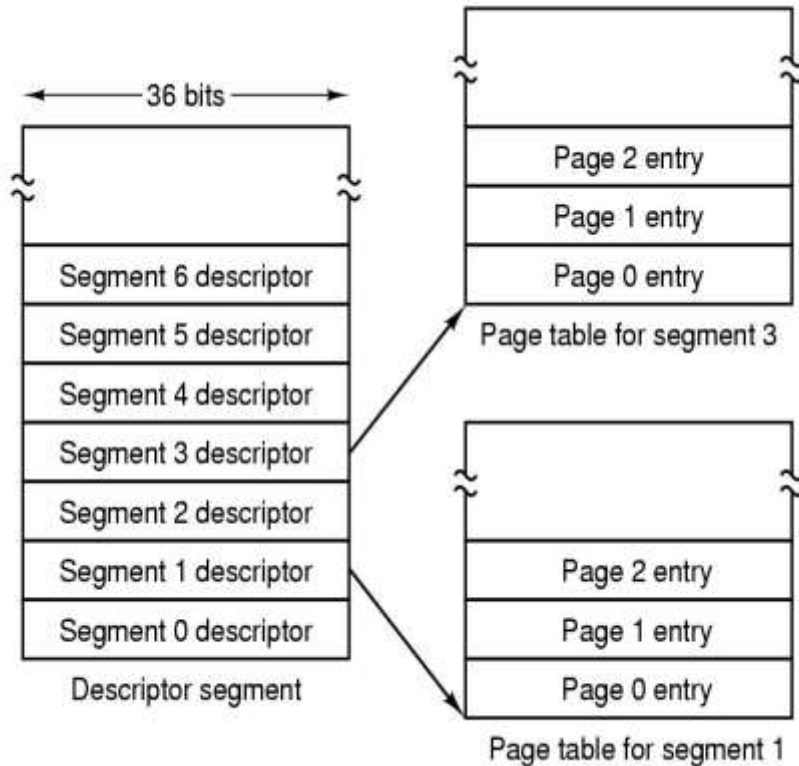
Implementation of Pure Segmentation



(a)-(d) Development of checkerboarding

(e) Removal of the checkerboarding by compaction

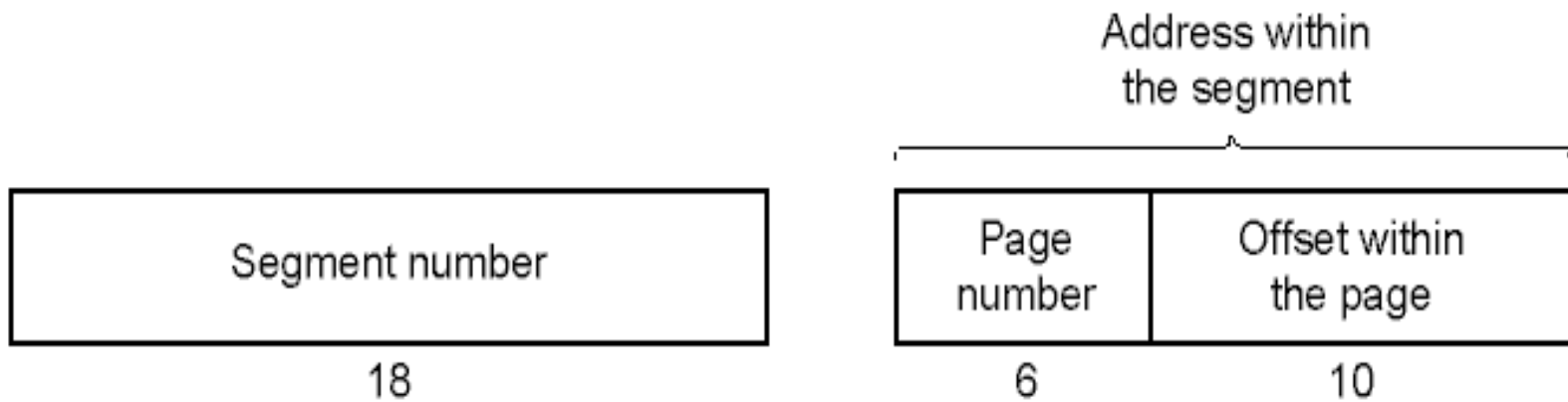
Segmentation with Paging: MULTICS (1)



- Descriptor segment points to page tables
- Segment descriptor – numbers are field lengths

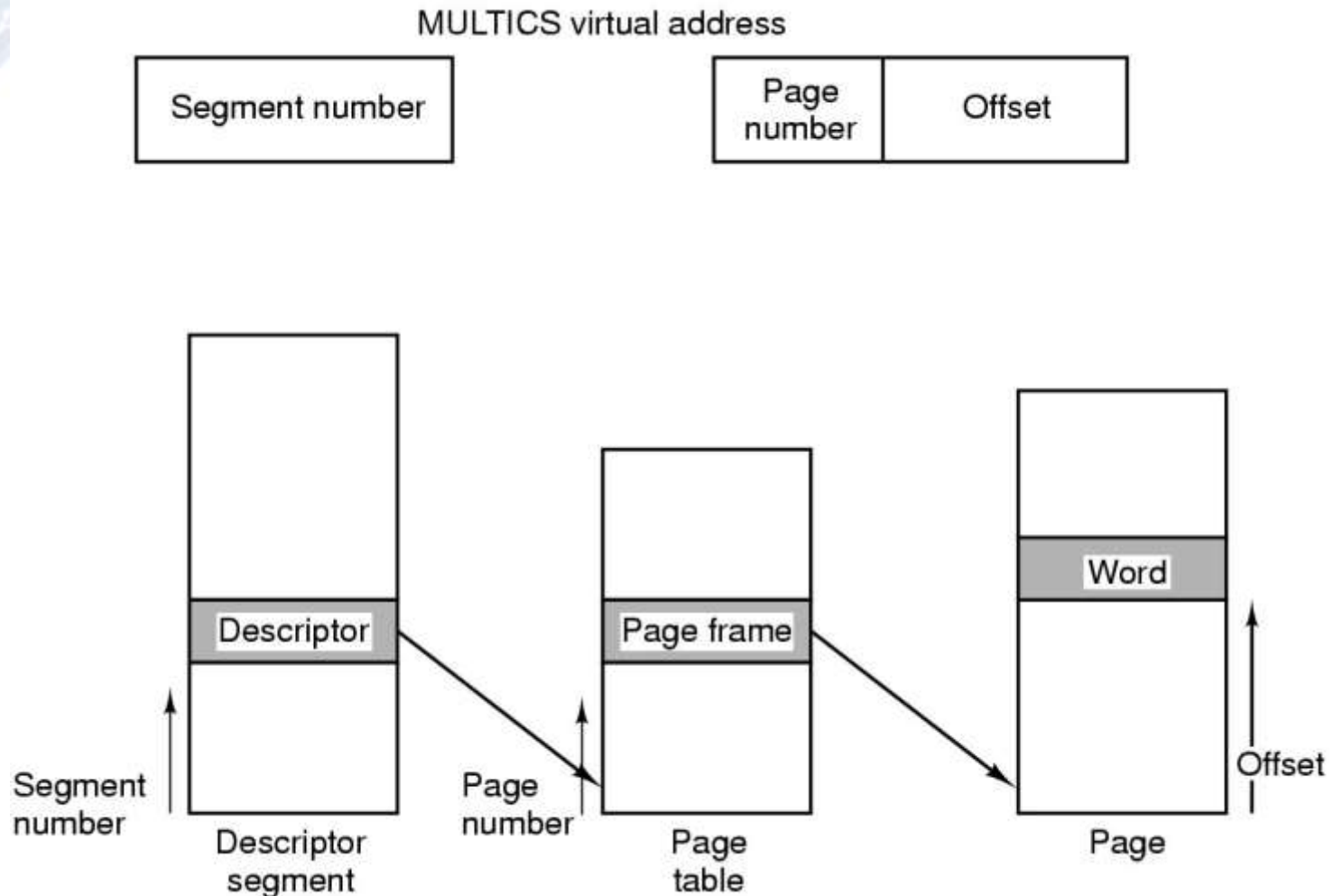
1. The segment number is used to find the segment descriptor.
2. A check is made to see if the segment's page table is in memory. If the page table is in memory, it is located. If it is not, a segment fault occurs. If there is a protection violation, a fault (trap) occurs.
3. The page table entry for the requested virtual page is examined. If the page is not in memory, a page fault occurs. If it is in memory, the main memory address of the start of the page is extracted from the page table entry.
4. The offset is added to the page origin to give the main memory address where the word is located.
5. The read or store finally takes place.

Segmentation with Paging: MULTICS (2)



A 34-bit MULTICS virtual address

Segmentation with Paging: MULTICS (3)



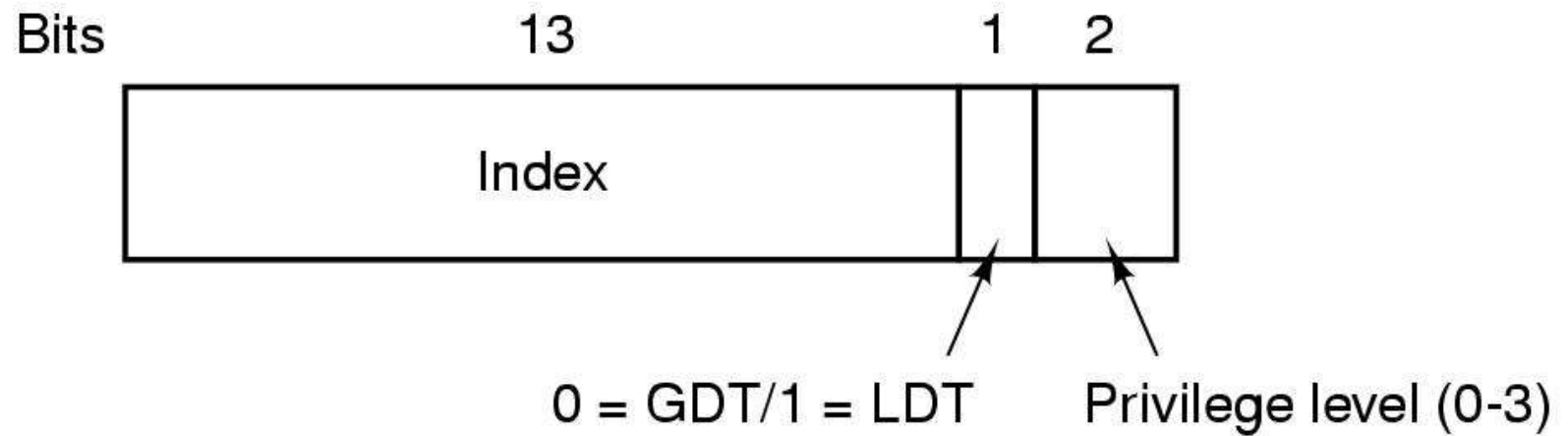
Conversion of a 2-part MULTICS address into a main memory address

Segmentation with Paging: MULTICS (4)

Comparison field		Page frame	Protection	Age	Is this entry used?
Segment number	Virtual page				↓
4	1	7	Read/write	13	1
6	0	2	Read only	10	1
12	3	1	Read/write	2	1
					0
2	1	0	Execute only	7	1
2	2	12	Execute only	9	1

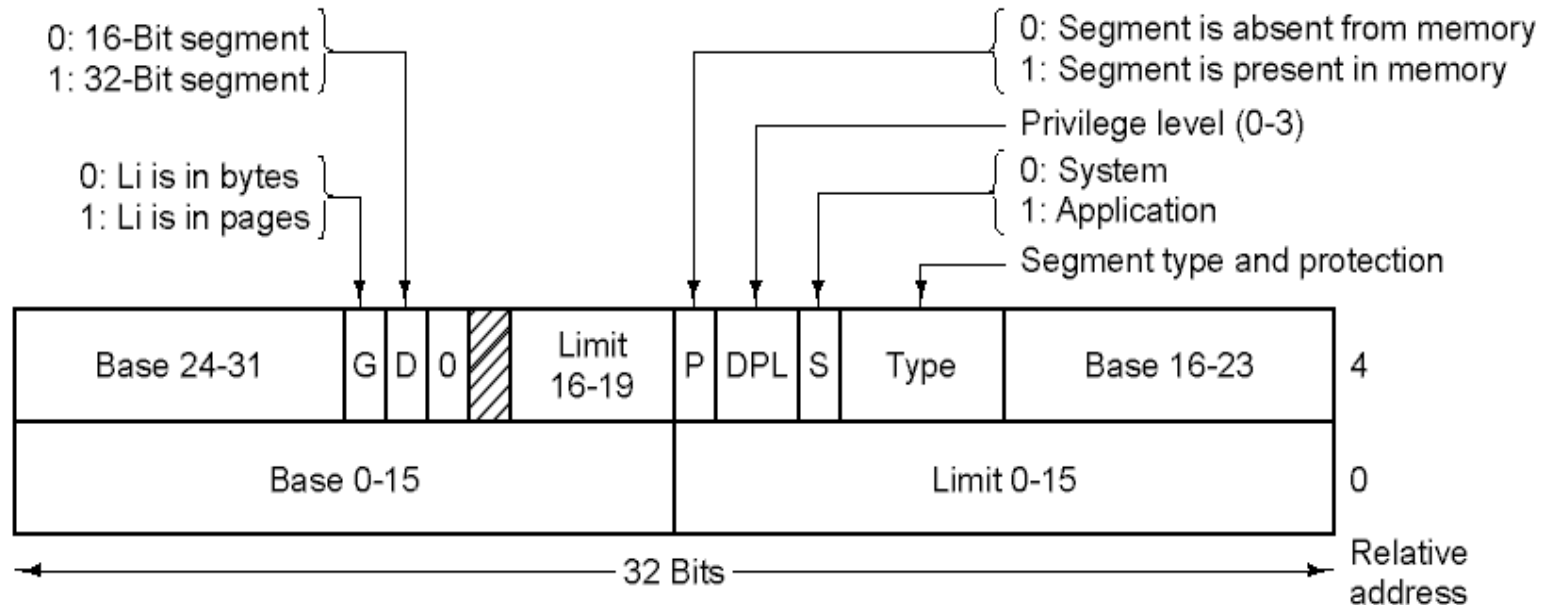
- Simplified version of the MULTICS TLB
- Existence of 2 page sizes makes actual TLB more complicated

Segmentation with Paging: Pentium (1)



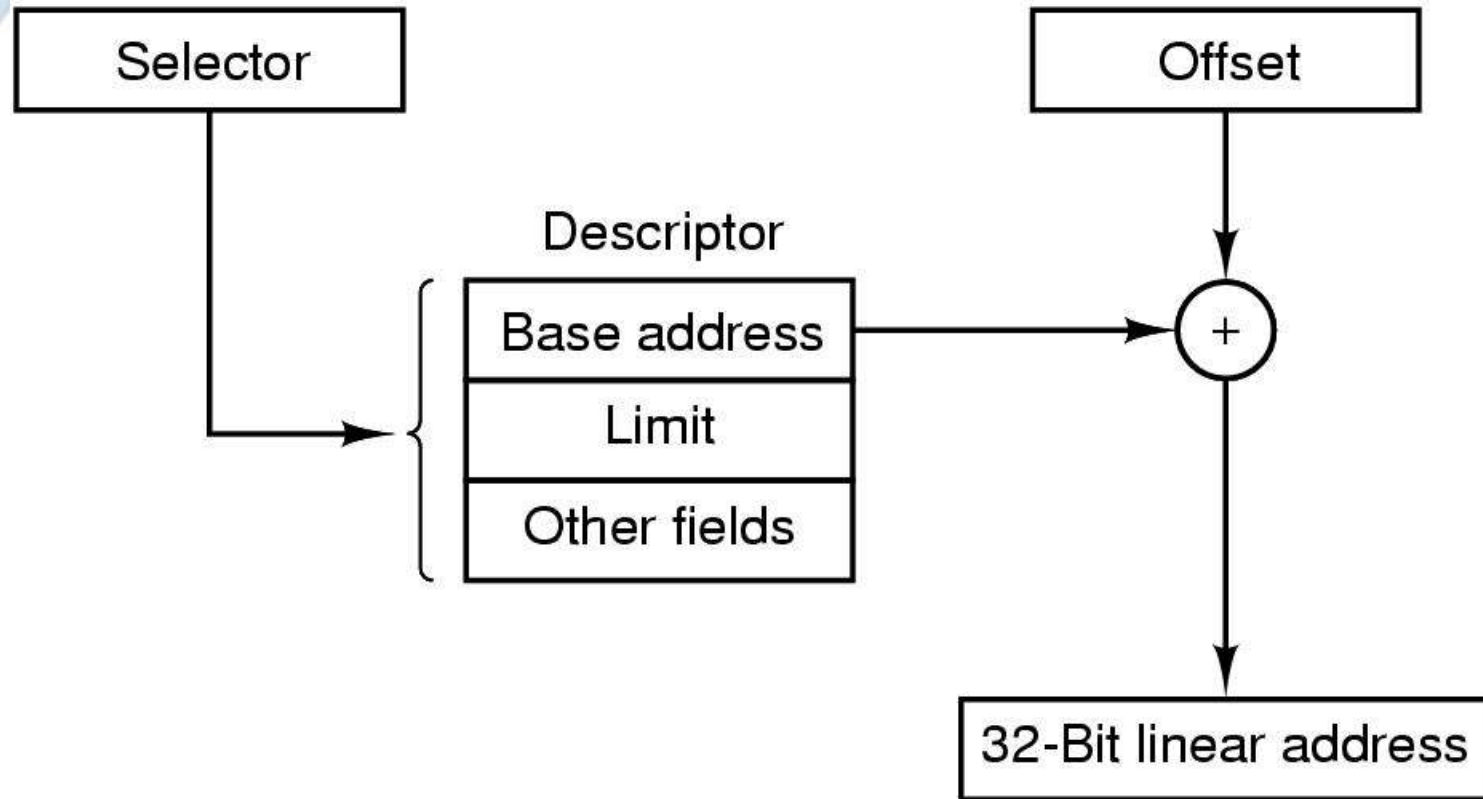
A Pentium selector

Segmentation with Paging: Pentium (2)



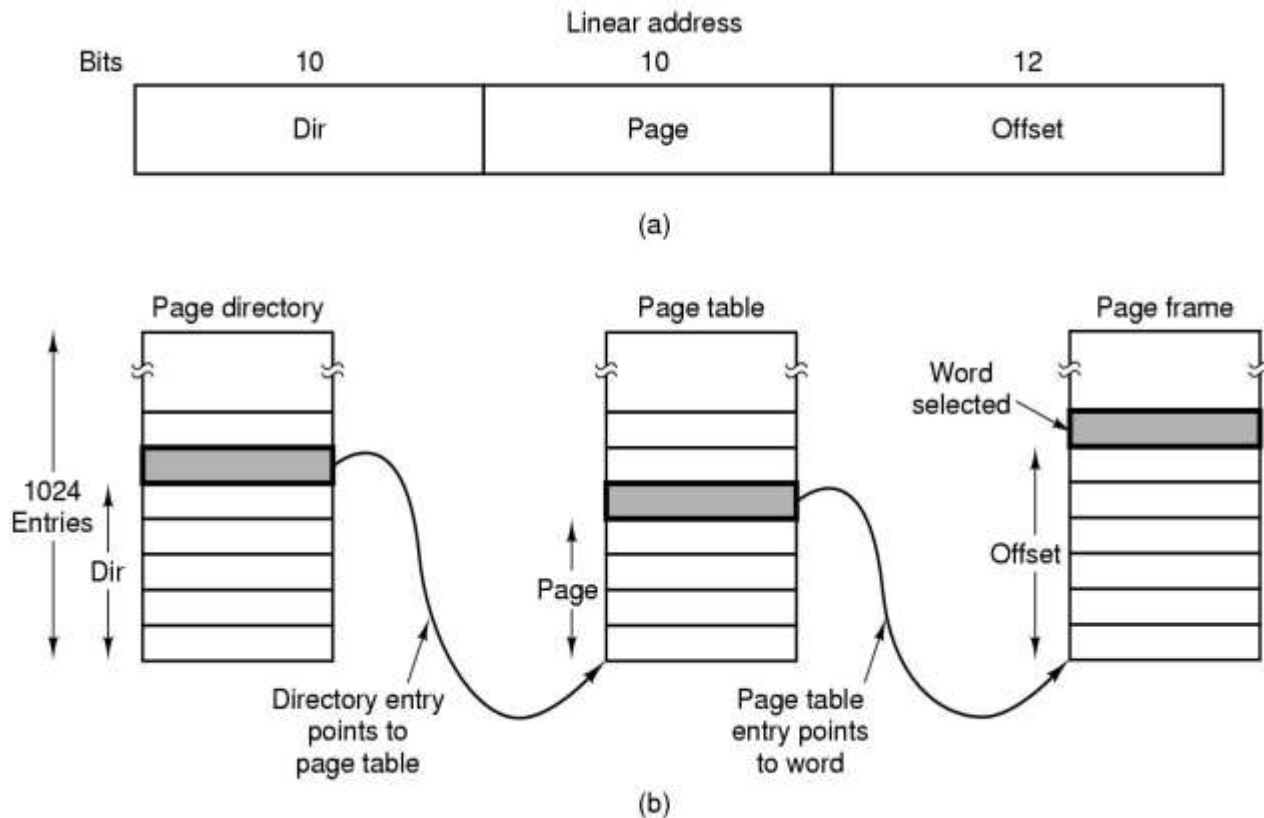
- Pentium code segment descriptor
- Data segments differ slightly

Segmentation with Paging: Pentium (3)



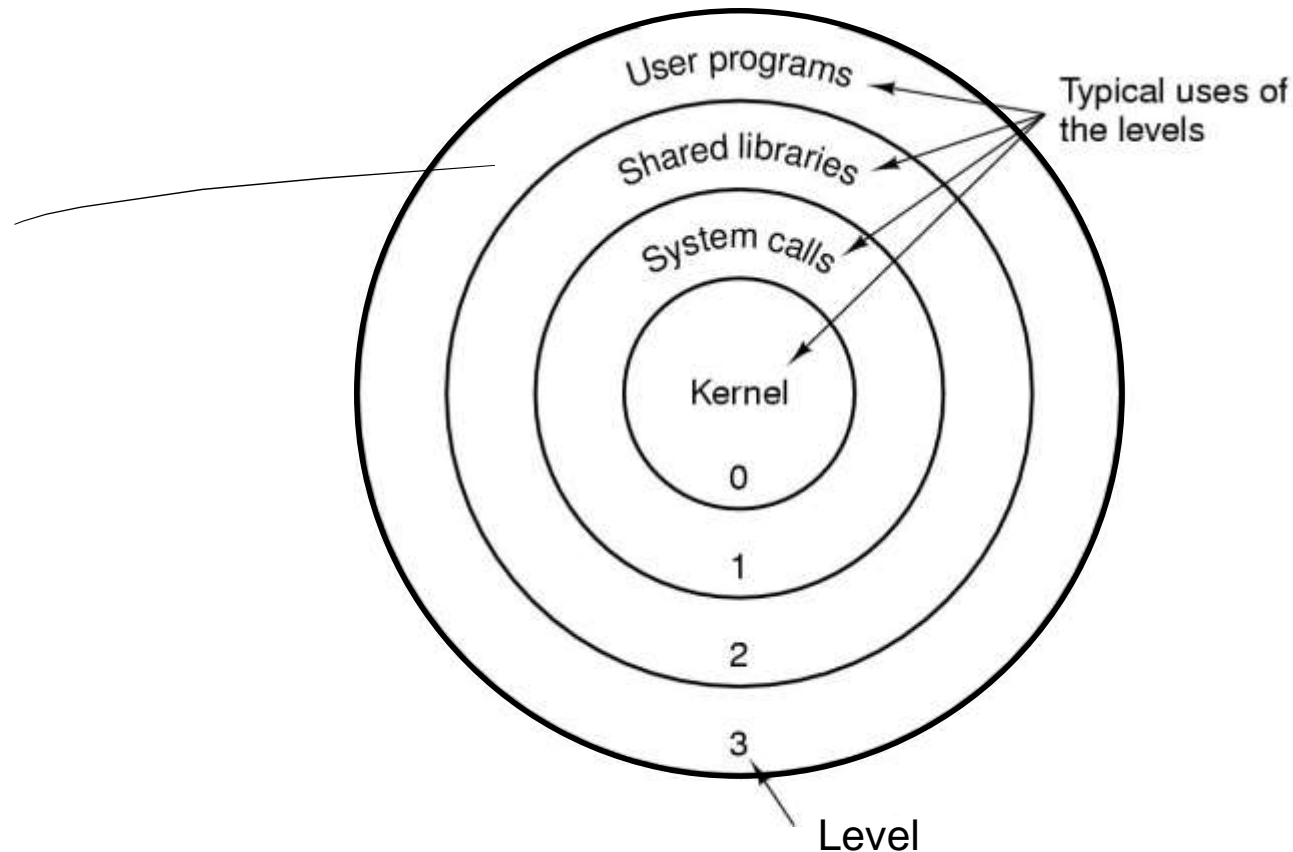
Conversion of a (selector, offset) pair to a linear address

Segmentation with Paging: Pentium (4)



Mapping of a linear address onto a physical address

Segmentation with Paging: Pentium (5)



Protection on the Pentium

AIF206

Sistem Operasi

Elisati Hulu
KBI Telematika

elisatih@unpar.ac.id
elisatih@gmail.com



INFORMATIKA
UNPAR