Operating system

Part VIII: Memory (Advanced)

By KONG LingBo (孔令波)

Goals

- Know the related concepts
  - Virtual Memory, Logical address, Physical addr ess, Address translation
- Virtual memory
  - **(on-demand) Paging** scheme
  - **(on-demand) Segmentation**
  - Segmentation + Paging ☾ **Hybrid**

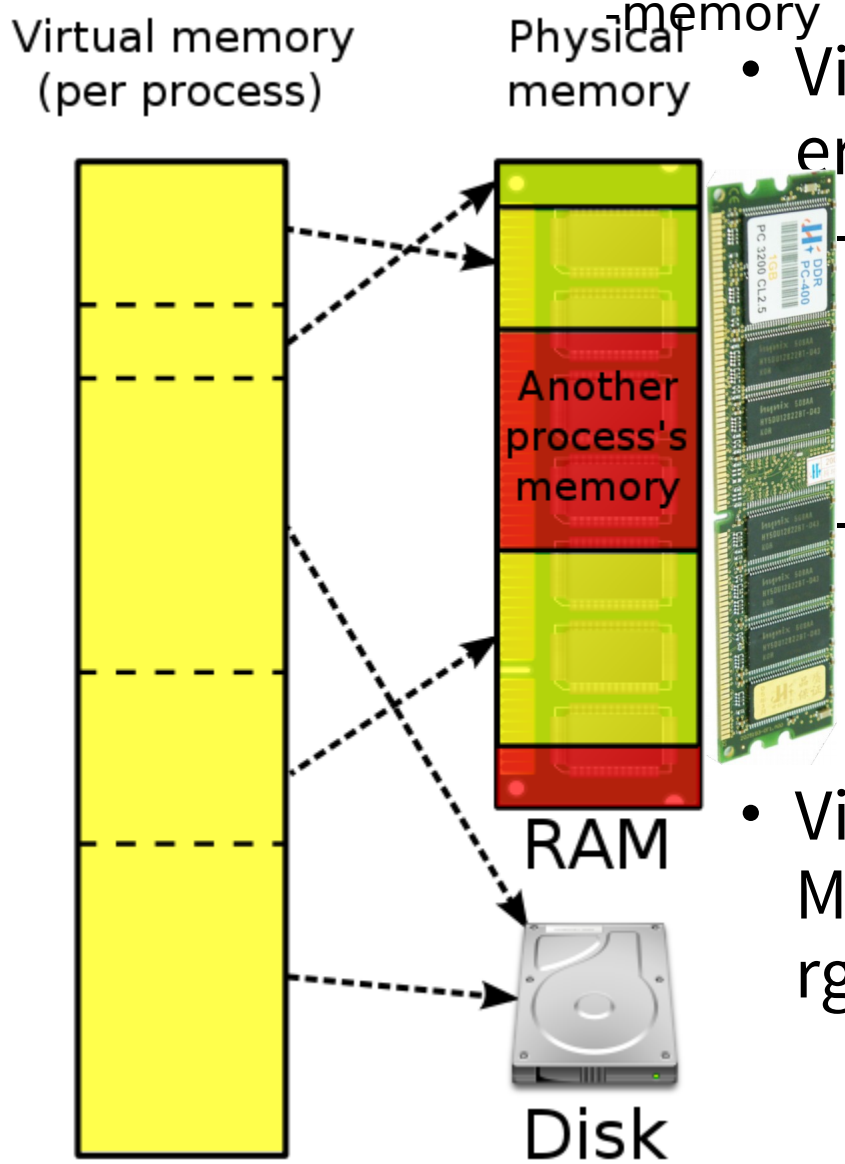
Now

- It’s time to learn the real techniques used by current OSs to provide **<u>VIRTUAL MEMORY</u>**

- We have known the fact
  - The instructions and data of a program should be stored in Main Memory first before its execution
  - With the experience on Windows®, we can infer that a program whose size is larger than the MM can still run

Ⓟ it seems that there is a “**<u>virtual</u>**” memory!

http://searchstorage.techtarget.com/definition/virtual-memory



- Virtual memory is a feature of an operating system that
  - enables a process to use a memory (RAM) address space that is independent of other processes running in the same system,
  - and use a space that is **larger than the actual amount of RAM** present, temporarily relegating some contents from RAM to a disk, with little or no overhead.
- Virtual memory combines active RAM and inactive memory to form a large range of contiguous addresses.

tp://en.wikipedia.org/wiki/Virtual_memory

Virtual Memory

- Paging
  - Basic paging
  - Paging-based VM
    - How to support the transparency of using space larger than the physical memory space
  - Page replacement algorithms
- Segmenting
  - Basic segmenting
  - Segmentation-based VM
    - How to support the transparency of using space larger than the physical memory space
- Segment-page scheme

Paging (分页)

PPTs from others\SCU_Zhaohui\OS\Chapter07.ppt

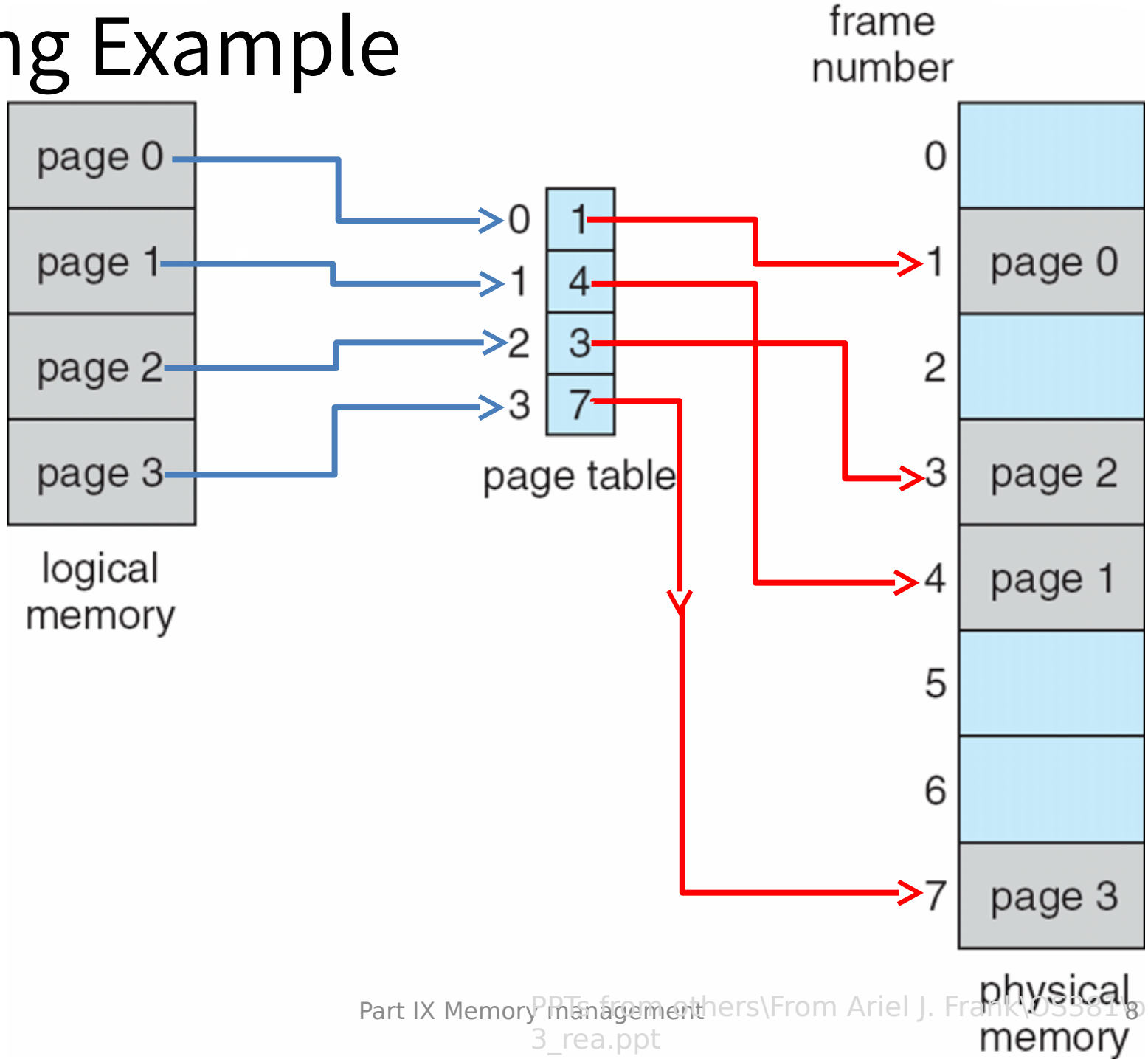
- The fundament of paging is **Fixed Partitioning** but with smaller size!
  - It “partition/cut” the logical space of a process into **page**s [页]
  - It “partition/cut” the physical space of MM into **frame**s [帧]
  - Be default, the sizes of page and frame are same
    - In 32 bit Windows, 4KB
- It seems that the mapping from the process space to MM is easy now
  - If there is available frame, we can assign a page to that frame
  - Since a page corresponds to a set of instructions, that process could run when executing those instructions
- Yes, that is in fact the basic functions of paging scheme!

Needed data structures

- To carry out the paging scheme
  - Besides the pages of a process, OS should kno w the mapping relationship between pages of t hat process and frames of the MM
- That is Page table [页表]

Page	Frame

Paging Example



Virtual Memory: Large as you wish!

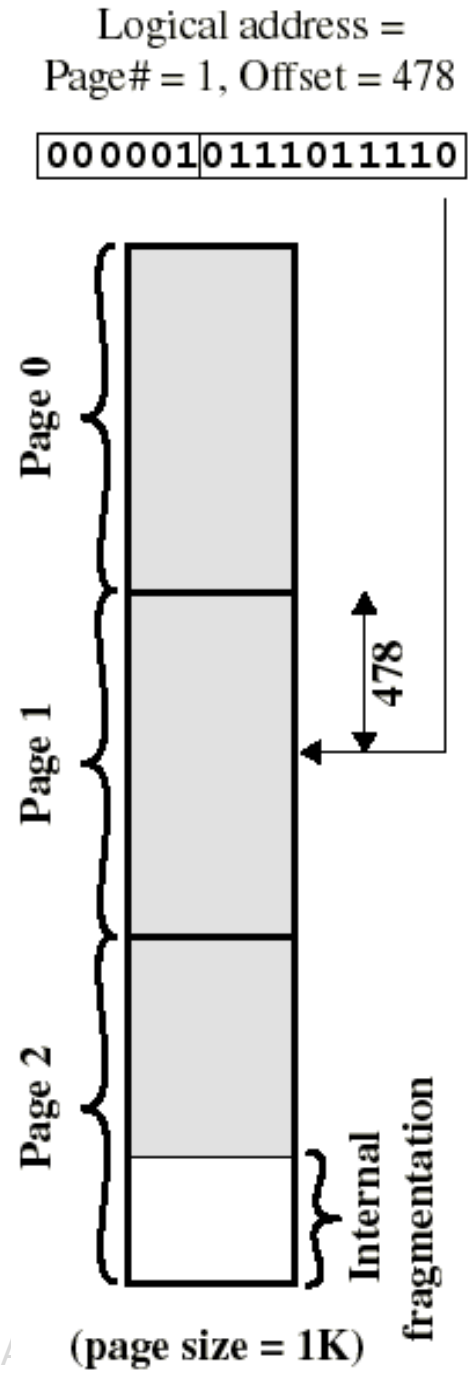
- Example:
  - Just 16 bits are needed to address a physical memory of 64KB.
  - Let's use a page size of 1KB so that 10 bits are needed for offsets within a page.
  - For the page number part of a logical address we may use a number of bits larger than 6, say 22 (a modest value!!), "**pretending**" a 32-bit address.
    - Now we have 2^{22} (=4M) pages, each of which is 1KB, so the VM of a process seems 4GB ($\approx 2^{22}*2^{10}=2^{32}$=4GB)
    - This explains why the max size of files in Win32 is 4GB

PPTs from other Prof. Ariel J. Frank 10238 1v08-1_vir.ppt

Logical address now

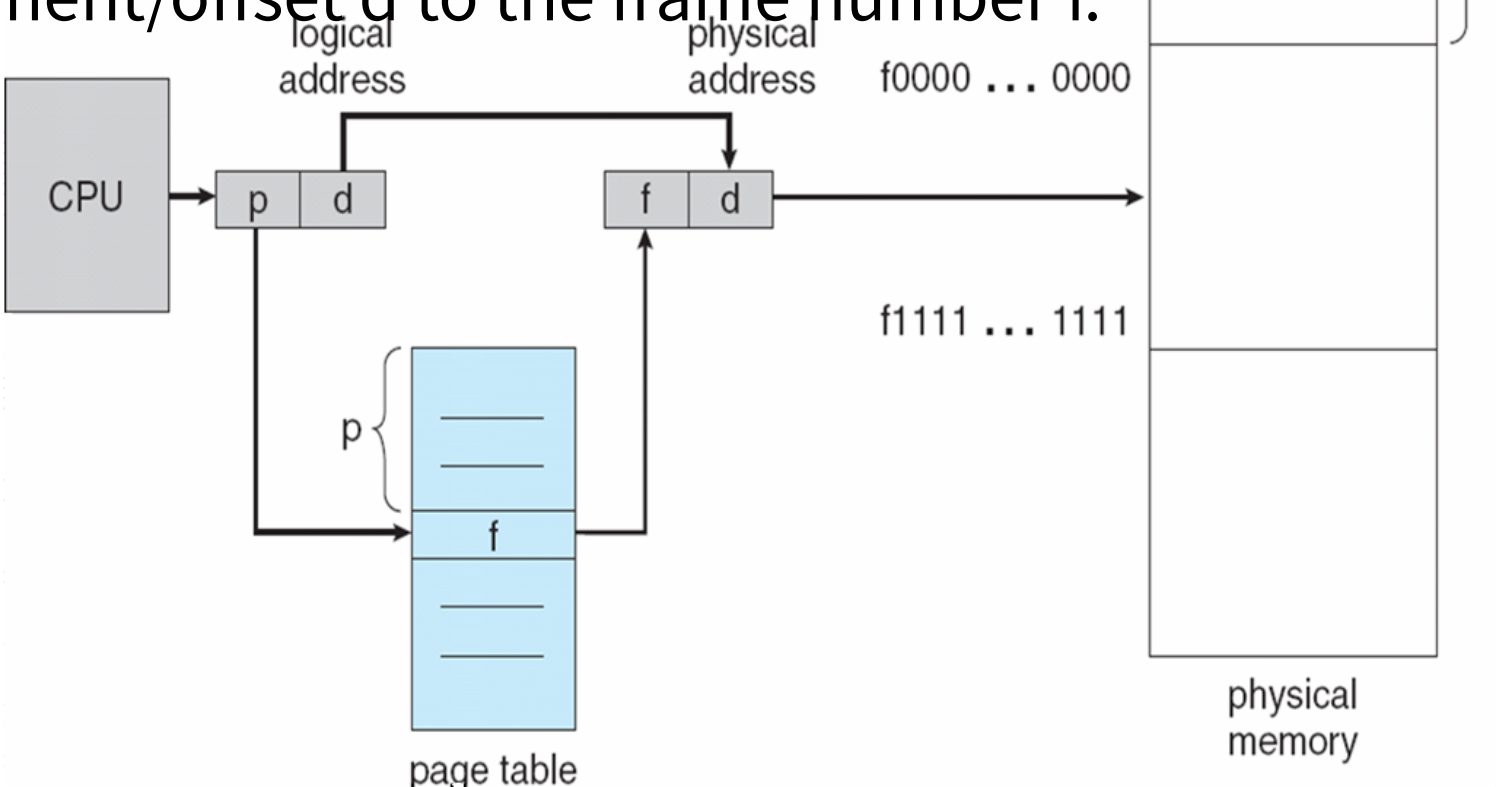
- Logical address now is divided int o two parts:
  - *Page number* (***p***) – used as an index i nto a *page table* which contains the b ase address of each page in physical memory.
  - *Page offset/displacement* (***d***) – combi ned with base address to define the physical memory address that is sent to the memory unit.

page number	page offset
p	***d***
m – n	***n***



Address Translation now

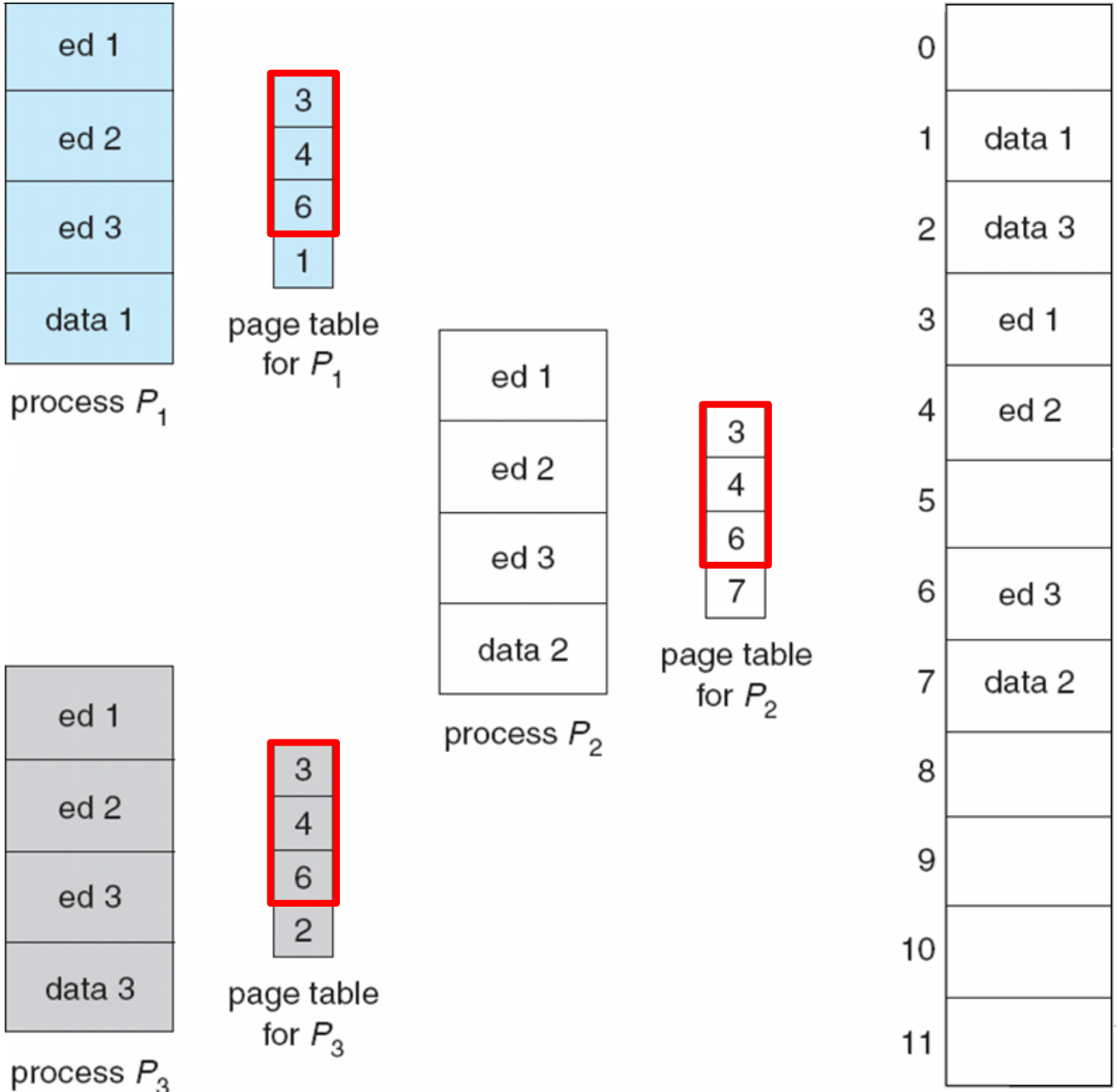
- Address translation at run-time is then easy to implement in hardware:
  - logical address **(p, d)** gets translated to p hysical address **(f, d)** by indexing the page table with p and appending the same disp lacement/offset d to the frame number f.



Sharing

- What can be shared
  - Reentrant code (pure code)
    - If the code is reentrant, then it never change s during execution. ⓟTwo or more processes can execute the same code at the same time .
  - Read-only data
- What can not be shared
  - Each process has its own copy of registers and data storage to hold the data for the process' s execution.
- The OS should provides facility to enforce some n ecessary property for sharing.

Shared Pages Example



Paging： The OS Concern

- What should the OS do?
  - –Which frames are al
  - –Which frames are
  - –How to allocate fra d rrived process?
    - Placement algorithm（放置算法）
    - Replacement algorithms（替換算法）

Similar as those discussed in Variable partition part!

PPTs from others\OS PPT in English\ch09.p

A computer's main memory is byte addressing, logical addresses and physical addresses are 32-bit, page table entry size is 4 bytes. Please answer the following questions. If use a paging management, and the logical address structure is :

Page Number （20 bits） offset （12 bits）

1. The size of the page is how many number of bytes?
2. And calculate the maximum number of bytes occupied by the page table? [2 pts]

In a paging memory management system, there is a page table as following:

Page No	0	1	2	3	4
Frame	2	1	6	3	7

If the page size is 4KB, then paging address hardware will convert logical address 0 into physical address （ ）.

A.8192 B.4096 C.2048 D.1024

Virtual Memory

- Paging
  - Basic paging
  - Paging-based VM
    - How to support the transparency of using space larger than the physical memory space
  - Page replacement algorithms
- Segmenting
  - Basic segmenting
  - Segmentation-based VM
    - How to support the transparency of using space larger than the physical memory space
- Segment-page scheme

Of course, we need some support for Virtual Memory

- Memory management hardware must support paging and/or segmentation [Discussed in former part].

- OS must be able to manage [Ⓟ **Concern of this part**]
  - the movement of pages and/or segments between external memory and main memory,
  - including placement and replacement of pages/segments.

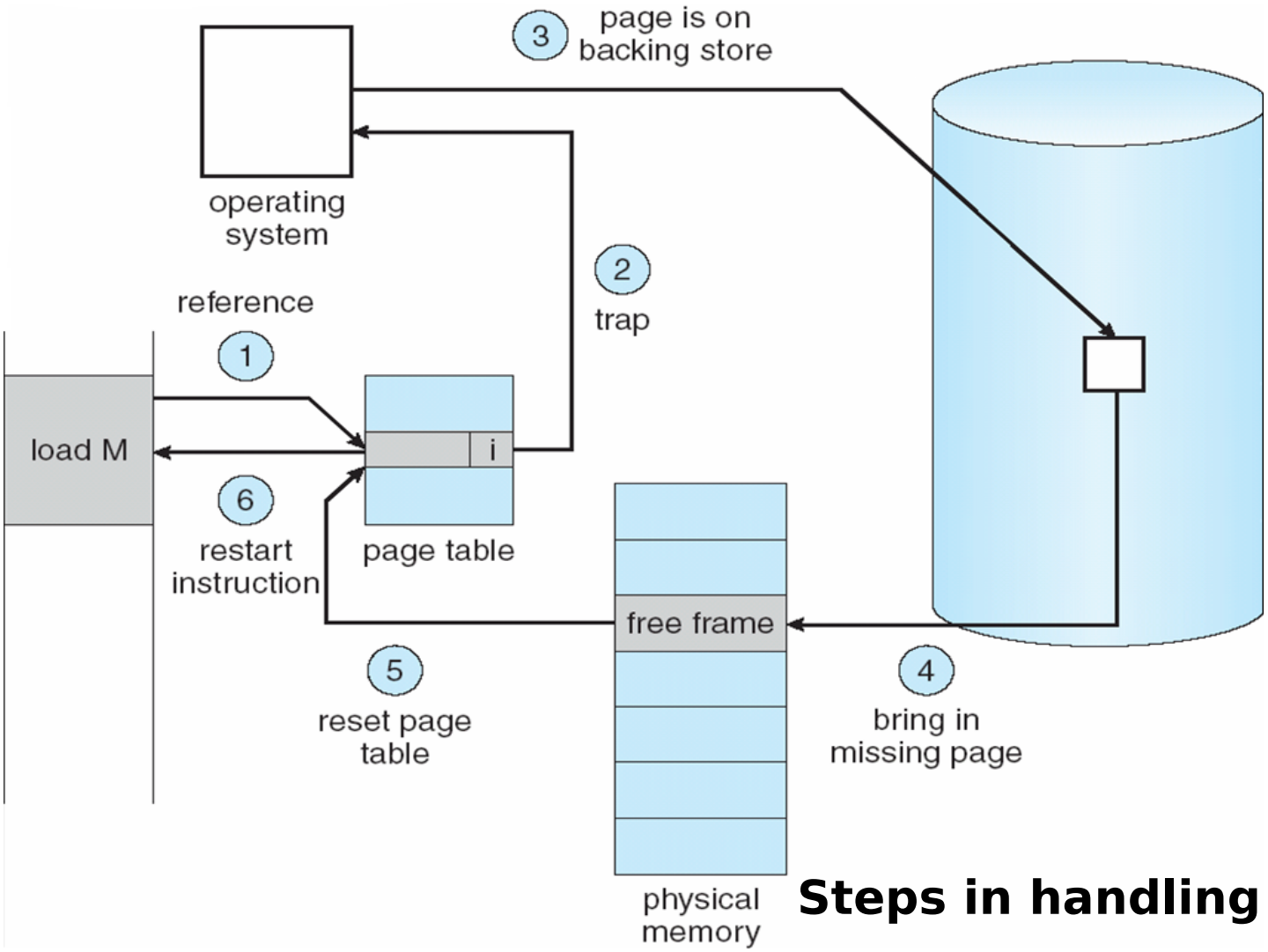
PPTs from others\From Ariel J. Frank\OS381\os8-1_vir.ppt

Now, the execution of a process looks like …

1. The OS brings into main memory only a few piece s of the program (including its starting point).
2. An interrupt (memory fault) is generated when the memory reference is on a piece that is not present in main memory – **<u>is needed</u>**.
   a. OS places the process in a
   b. OS issues a disk I/O Read r
      mory the piece referenc
   c. Another process is dispa
      takes place.
3. An interrupt is issued when disk I/O completes; thi s causes the OS to place the affected process back in the Ready state.

Called Demand Paging!

If one page is not in MM yet (**Page Fault**)



Steps in handling a Page Fa

Steps in handling a Page Fault

1. If there is ever a reference to a page not in memory, first reference will cause **page fau lt**.
2. Page fault is handled by the appropriate O S service routines.
3. Locate needed page on disk (in file or in back ing store).
4. Swap page into free frame (assume availab le).
5. Reset page tables – valid-invalid bit = 1.
6. Restart instruction.

PPts from others\from Ahery, Frank\CS381\os8-1_vir.ppt

What happens if there is no free frame?

- Page replacement – find some page in me mory, but not really in use, **swap** it out.
- Need **<u>page replacement algorithm</u>**.
- Performance – want an algorithm which wi ll result in minimum number of page faults.
- Same page may be brought into memory s everal times.

PPTs from others\From Ariel J. Frank\OS381\os8-1_vir.ppt

Effective Access Time (EAT)

- EAT = (1 – **p**) x memory access
  + **p** (page fault overhead
  + swap page out
  + swap page in
  + restart overhead)

- "p" means Page Fault Rate
  - $0 \leqslant p \leqslant 1.0$
  - if p = 0, no page faults
  - if p = 1, every reference is a fault

- Demand Paging Example
  - Memory access time = **200 nanoseconds [纳秒]**
  - Average page-fault service time = 8 milliseconds [毫秒]
  - EAT = (1 – p) x 200 + p (8 milliseconds)
    = (1 – p) x 200 + p x 8,000,000
    = 200 + p x 7,999,800
  - If one access out of 1,000 causes a page fault, then
    EAT = 200+0.001*7999800
    = 8199.8 nanoseconds
    = **8.2 microseconds [微秒]**.

1 millisec = 1,000 microsec = 1,000,000 nanosec

How to improve this kind of **slowdown**?

- The solution could be derived from the EAT equation - Improve the **access speed**, and decrease the **page fault**
- Two strategies
  - Keeping page table in a higher access speed m edia
    - Cache (high speed but quite expensive), and the **TL B** (**translation looka**[illegible] **buffe**[illegible])
  - Prefetching the possi[illegible]
    - When page 3 cause[illegible] ge 2,4,and 5 into MM[illegible]

Of course, decision is needed – namely the algorithms

a Translation Look-aside Buffer (TLB)

翻译后备缓冲器

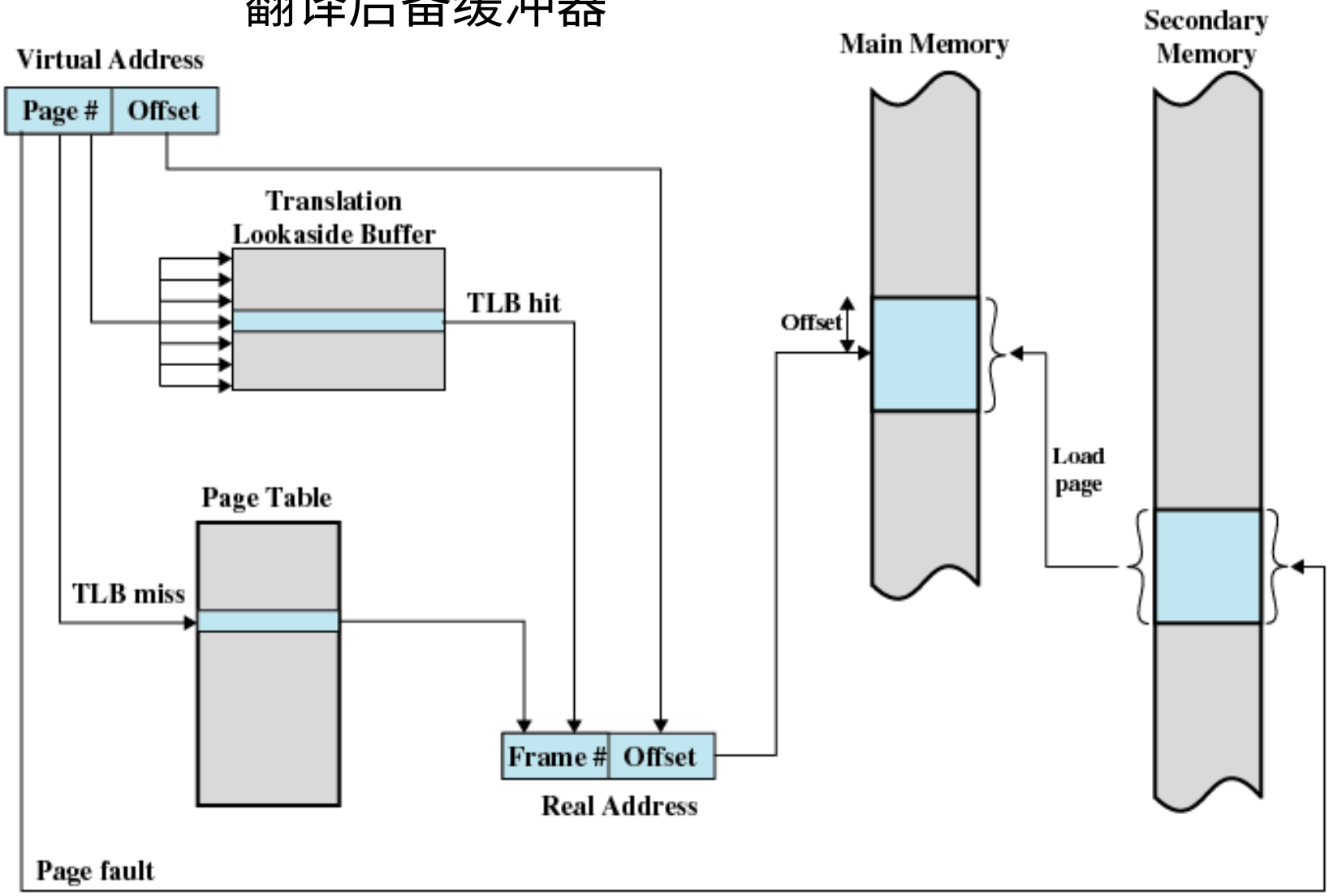


Figure 8.7 Use of a Translation Lookaside Buffer

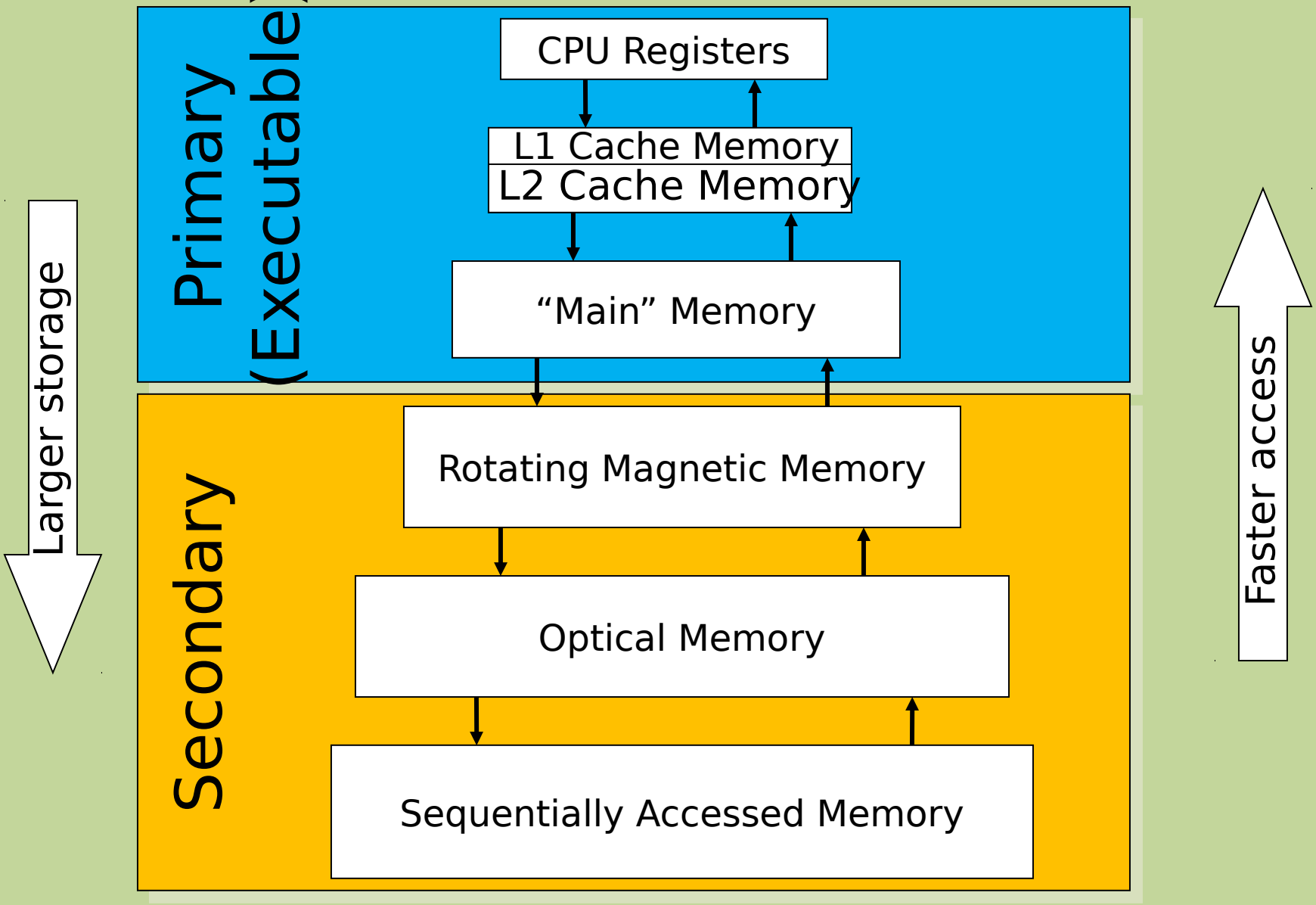
Now the EAT with TLB

- Parameters
  - TLB Lookup = 20 nanoseconds
  - Hit ratio (percentage of times that a particular page is found in the TLB) = 80%
  - Memory access time = **200 nanoseconds**
  - Average page-fault service time = 8 milliseconds
  - If one access for Page table out of 1,000 causes a page fault
- EAT = (20)*0.8 + [200+(8000000-200)*0.001]*0.2

  = 16+ 8199.8*0.2

  = 1655.96≈1.6 **microseconds**

Contemporary Memory Hierarchy & Dynamic Loading



Virtual Memory

- Paging
  - Basic paging
  - Paging-based VM
    - How to support the transparency of using space larger than the physical memory space
  - Page replacement algorithms
- Segmenting
  - Basic segmenting
  - Segmentation-based VM
    - How to support the transparency of using space larger than the physical memory space
- Segment-page scheme

Example

- A process makes references to 4 pages: A, B, E, and R
  - Reference stream: BEERBAREBEAR
- Physical memory size: 3 pages

The FIFO Policy

- Treats page frames allocated to a proce ss as a circular buffer:
  - When the buffer is full, the oldest page is r eplaced. Hence first-in, first-out:
    - A frequently used page is often the oldest, so i t will be repeatedly paged out by FIFO.
  - Simple to implement:
    - requires only a pointer that circles through th e page frames of the process.

PPTs from others\From Ariel J. Frank\OS381\os8-3_vir.ppt

FIFO

Memory page	B	E	E	R	B	A	R	E	B	E	A	R
1	B											
2												
3												

FIFO

Memory page	B	E	E	R	B	A	R	E	B	E	A	R
1	B											
2		E										
3												

FIFO

Memory page	B	E	E	R	B	A	R	E	B	E	A	R
1	B											
2		E	*									
3												

FIFO

Memory page	B	E	E	R	B	A	R	E	B	E	A	R
1	B											
2		E	*									
3				R								

FIFO

Memory page	B	E	E	R	B	A	R	E	B	E	A	R
1	B				*							
2		E	*									
3				R								

FIFO

Memory page	B	E	E	R	B	A	R	E	B	E	A	R
1	B				*							
2		E	*									
3				R								

FIFO

Memory page	B	E	E	R	B	A	R	E	B	E	A	R
1	B				*	A						
2		E	*									
3				R								

FIFO

Memory page	B	E	E	R	B	A	R	E	B	E	A	R
1	B				*	A						
2		E	*									
3				R			*					

FIFO

Memory page	B	E	E	R	B	A	R	E	B	E	A	R
1	B				*	A						
2		E	*					*				
3				R			*					

FIFO

Memory page	B	E	E	R	B	A	R	E	B	E	A	R
1	B				*	A						
2		E	*					*				
3				R			*					

FIFO

Memory page	B	E	E	R	B	A	R	E	B	E	A	R
1	B				*	A						
2		E	*					*	B			
3				R			*					

FIFO

Memory page	B	E	E	R	B	A	R	E	B	E	A	R
1	B				*	A						
2		E	*					*	B			
3				R			*					

FIFO

Memory page	B	E	E	R	B	A	R	E	B	E	A	R
1	B				*	A						
2		E	*					*	B			
3				R			*			E		

FIFO

Memory page	B	E	E	R	B	A	R	E	B	E	A	R
1	B				*	A					*	
2		E	*					*	B			
3				R			*			E		

FIFO

Memory page	B	E	E	R	B	A	R	E	B	E	A	R
1	B				*	A					*	
2		E	*					*	B			
3				R			*			E		

FIFO

Memory page	B	E	E	R	B	A	R	E	B	E	A	R
1	B				*	A					*	R
2		E	*					*	B			
3				R			*			E		

FIFO

- 7 page faults

Memory page	B	E	E	R	B	A	R	E	B	E	A	R
1	B				*	A					*	R
2		E	*					*	B			
3				R			*			E		

compulsory
D.J.[kəm'pʌlsəri]
adj. 必须做的，强制性

FIFO

- 4 compulsory cache misses

Memory page	B	E	E	R	B	A	R	E	B	E	A	R
1	***B***				*	***A***					*	R
2		***E***	*					*	B			
3				***R***			*			E		

Optimal Page Replacement

- The Optimal policy selects for replacement the page that will not be used for longest period of time.

- **<u>Impossible to implement</u>** (need to know the fu ture) but serves as a standard to compare with the other algorithms we shall study.

PPTs from others\From Ariel J. Frank\OS381\os8-3_vir.ppt

Optimal (MIN)

Memory page	B	E	E	R	B	A	R	E	B	E	A	R
1	B											
2		E	*									
3				R								

Optimal (MIN)

Memory page	B	E	E	R	B	A	R	E	B	E	A	R
1	B				*							
2		E	*									
3				R								

Optimal (MIN)

Memory page	B	E	E	R	B	A	R	E	B	E	A	R
1	B				*							
2		E	*									
3				R								

MIN

Memory page	B	E	E	R	B	A	R	E	B	E	A	R
1	B				*	A						
2		E	*									
3				R								

MIN

Memory page	B	E	E	R	B	A	R	E	B	E	A	R
1	B				*	A						
2		E	*									
3				R			*					

MIN

Memory page	B	E	E	R	B	A	R	E	B	E	A	R
1	B				*	A						
2		E	*					*				
3				R			*					

MIN

Memory page	B	E	E	R	B	A	R	E	B	E	A	R
1	B				*	A						
2		E	*					*				
3				R			*					

MIN

Memory page	B	E	E	R	B	A	R	E	B	E	A	R
1	B				*	A						
2		E	*					*				
3				R			*		B			

MIN

Memory page	B	E	E	R	B	A	R	E	B	E	A	R
1	B				*	A						
2		E	*					*		*		
3				R			*		B			

MIN

Memory page	B	E	E	R	B	A	R	E	B	E	A	R
1	B				*	A					*	
2		E	*					*		*		
3				R			*		B			

MIN

Memory page	B	E	E	R	B	A	R	E	B	E	A	R
1	B				*	A					*	R
2		E	*					*		*		
3				R			*		B			

MIN

- 6 page faults

Memory page	B	E	E	R	B	A	R	E	B	E	A	R
1	B				*	A					*	R
2		E	*					*		*		
3				R			*		B			

The LRU Policy
[least recently used: 最近最少使用算法]

- Replaces the page that has not been referenced for the longest time recently:
  - By the principle of locality, this should be the page least likely to be referenced in the near future.
  - performs nearly as well as the optimal policy.

LRU

Memory page	B	E	E	R	B	A	R	E	B	E	A	R
1	B											
2		E	*									
3				R								

LRU

Memory page	B	E	E	R	B	A	R	E	B	E	A	R
1	B				*							
2		E	*									
3				R								

LRU

Memory page	B	E	E	R	B	A	R	E	B	E	A	R
1	B				*							
2		E	*									
3				R								

LRU

Memory page	B	E	E	R	B	A	R	E	B	E	A	R
1	B				*							
2		E	*			A						
3				R								

LRU

Memory page	B	E	E	R	B	A	R	E	B	E	A	R
1	B				*							
2		E	*			A						
3				R			*					

LRU

Memory page	B	E	E	R	B	A	R	E	B	E	A	R
1	B				*							
2		E	*			A						
3				R			*					

LRU

Memory page	B	E	E	R	B	A	R	E	B	E	A	R
1	B				*			E				
2		E	*			A						
3				R			*					

LRU

Memory page	B	E	E	R	B	A	R	E	B	E	A	R
1	B				*			E				
2		E	*			A						
3				R			*					

LRU

Memory page	B	E	E	R	B	A	R	E	B	E	A	R
1	B				*			E				
2		E	*			A			B			
3				R			*					

LRU

Memory page	B	E	E	R	B	A	R	E	B	E	A	R
1	B				*			E		*		
2		E	*			A			B			
3				R			*					

LRU

Memory page	B	E	E	R	B	A	R	E	B	E	A	R
1	B				*			E		*		
2		E	*			A			B			
3				R			*					

LRU

Memory page	B	E	E	R	B	A	R	E	B	E	A	R
1	B				*			E		*		
2		E	*			A			B			
3				R			*				A	

LRU

Memory page	B	E	E	R	B	A	R	E	B	E	A	R
1	B				*			E		*		
2		E	*			A			B			
3				R			*				A	

LRU

Memory page	B	E	E	R	B	A	R	E	B	E	A	R
1	B				*			E		*		
2		E	*			A			B			R
3				R			*				A	

LRU

- 8 page faults

Memory page	B	E	E	R	B	A	R	E	B	E	A	R
1	B				*			E		*		
2		E	*			A			B			R
3				R			*				A	

The Clock (Second Chance) Policy

- Replaces an old page, but not the oldest page
- Arranges physical pages in a circle
  - With a clock hand
- Each page has a ***used bit***
  - Set to 1 on reference
  - On page fault, sweep the clock hand
    - If the used bit == 1, set it to 0 and **advance the hand**
    - If the used bit == 0, pick the page for replacement

PPTs from others\From Ariel J. Frank\OS381\os8-3_vir.ppt

The Clock (Second Chance) Policy

- The set of frames candidate for replacement is consi dered as a circular buffer.
- When a page is replaced, a pointer is set to point to t he next frame in buffer.
- A reference bit for each frame is set to 1 whenever:
  - a page is first loaded into the frame.
  - the corresponding page is referenced.
- When it is time to replace a page, the first frame enc ountered with the reference bit set to 0 is replaced:
  - During the search for replacement, each reference bit set to 1 is changed to 0.

PPTs from others\From Ariel J. Frank\OS381\os8-3_vir.ppt

CLOCK

Memory page	B	E	E	R	B	A	R	E	B	E	A	R
1	B*											
2												
3												

CLOCK

Memory page	B	E	E	R	B	A	R	E	B	E	A	R
1	B*											
2		E*										
3												

CLOCK

Memory page	B	E	E	R	B	A	R	E	B	E	A	R
1	B*											
2		E*	E*									
3												

CLOCK

Memory page	B	E	E	R	B	A	R	E	B	E	A	R
1	B*											
2		E*	E*									
3				R*								

CLOCK

Memory page	B	E	E	R	B	A	R	E	B	E	A	R
1	B*				B*							
2		E*	E*									
3				R*								

CLOCK

Since there is a "*" here, we need to advance the hand

Here we need to clear "*" and replacement

Memory page	B	E	E	R	B	A	R	E	B	E	A	R
1	B*				B*							
2		E*	E*									
3				R*								

CLOCK

Since there is a "*", clear "*" and advance th

Memory page	B	E	E	R	B	A	R	E	B	E	A	R
1	B*				B*	B						
2		E*	E*			E						
3				R*								

CLOCK

Since there is a “*”, clear “*” and advance th

Memory page	B	E	E	R	B	A	R	E	B	E	A	R
1	B*				B*	B						
2		E*	E*			E						
3				R*		R						

CLOCK

Now, we can assign A to this position

Memory page	B	E	E	R	B	A	R	E	B	E	A	R
1	B*				B*	A*						
2		E*	E*			E						
3				R*		R						

CLOCK

Memory page	B	E	E	R	B	A	R	E	B	E	A	R
1	B*				B*	A*						
2		E*	E*			E						
3				R*		R	R*					

CLOCK

Memory page	B	E	E	R	B	A	R	E	B	E	A	R
1	B*				B*	A*						
2		E*	E*			E		E*				
3				R*		R	R*					

CLOCK

A page fault again, clear "*" and advance the hand

Memory page	B	E	E	R	B	A	R	E	B	E	A	R
1	B*				B*	A*						
2		E*	E*			E		E*	E			
3				R*		R	R*					

CLOCK

Clear “*” and advance the hand

Memory page	B	E	E	R	B	A	R	E	B	E	A	R
1	B*				B*	A*						
2		E*	E*			E		E*	E			
3				R*		R	R*		R			

CLOCK

Clear “*” and advance the hand

Memory page	B	E	E	R	B	A	R	E	B	E	A	R
1	B*				B*	A*			A			
2		E*	E*			E		E*	E			
3				R*		R	R*		R			

CLOCK

Now put B here, because there is no “*” here

Memory page	B	E	E	R	B	A	R	E	B	E	A	R
1	B*				B*	A*			A			
2		E*	E*			E		E*	B*			
3				R*		R	R*		R			

CLOCK

Memory page	B	E	E	R	B	A	R	E	B	E	A	R
1	B*				B*	A*			A			
2		E*	E*			E		E*	B*			
3				R*		R	R*		R	E*		

CLOCK

Memory page	B	E	E	R	B	A	R	E	B	E	A	R
1	B*				B*	A*			A		A*	
2		E*	E*			E		E*	B*			
3				R*		R	R*		R	E*		

CLOCK

A page fault again, clear "*" and advance the hand

Memory page	B	E	E	R	B	A	R	E	B	E	A	R
1	B*				B*	A*			A		A*	A
2		E*	E*			E		E*	B*			
3				R*		R	R*		R	E*		

CLOCK

Clear "*" and advance the hand

Memory page	B	E	E	R	B	A	R	E	B	E	A	R
1	B*				B*	A*			A		A*	A
2		E*	E*			E		E*	B*			B
3				R*		R	R*		R	E*		

CLOCK

Clear "*" and advance the hand

Memory page	B	E	E	R	B	A	R	E	B	E	A	R
1	B*				B*	A*			A		A*	A
2		E*	E*			E		E*	B*			B
3				R*		R	R*		R	E*		E

CLOCK

Now put R here !

Memory page	B	E	E	R	B	A	R	E	B	E	A	R
1	B*				B*	A*			A		A*	R*
2		E*	E*			E		E*	B*			B
3				R*		R	R*		R	E*		E

Does adding RAM always reduce misses?

- Yes for LRU and MIN
  - Memory content of X pages ⊆ X + 1 pages

- **No for FIFO**
  - Due to modulo math
  - ***Belady's anomaly***: getting more page faults by increasing the memory size

Belady’s Anomaly

- 9 page faults

Memory page	A	B	C	D	A	B	E	A	B	C	D	E
1	A			D			E					*
2		B			A			*		C		
3			C			B			*		D	

Belady’s Anomaly

- 10 page faults

Memory page	A	B	C	D	A	B	E	A	B	C	D	E
1	A				*		E				D	
2		B				*		A				E
3			C						B			
4				D						C		

Possibility of **Thrashing**

- If a process does not have "enough" pages, the page-fault rate is very high. This leads to:
  - low CPU utilization.
  - operating system thinks that it needs to increase t he degree of multiprogramming.
  - another process added to the system.
  - This just increases the load on physical memory.

- **Thrashing** = a process is busy swapping page s in and out.

PPTs from others\From Ariel J. Frank\OS381\os8-2_vir.ppt

You can try those algorithms by yourself

- Assume:
  - 3 frames
  - Reference string: 7,0,1,2,0,3,0,4,2,3,0,3,2,1,2,0,1,7,0,1
  - Each of the numbers refers to a page number
- Your task now
  - FIFO
  - LRU
  - Clock

Virtual Memory

- Paging
  - Basic paging
  - Paging-based VM
    - How to support the transparency of using space larger than the physical memory space
  - Page replacement algorithms
- Segmenting
  - Basic segmenting
  - Segmentation-based VM
    - How to support the transparency of using space larger than the physical memory space
- Segment-page scheme

Motivation of Segmenting

- Paging
  - Mapping to allow differentiation between logical memory and physical memory.
  - Separation of the user’s view of memory and the actual physical memory.
  - Chopping a process into equally-sized pieces.

☾ Paging division is arbitrary; no natural/logical boundaries for protection/sharing.

Ⓟ Any scheme for dividing a process into a collection of semantic units?

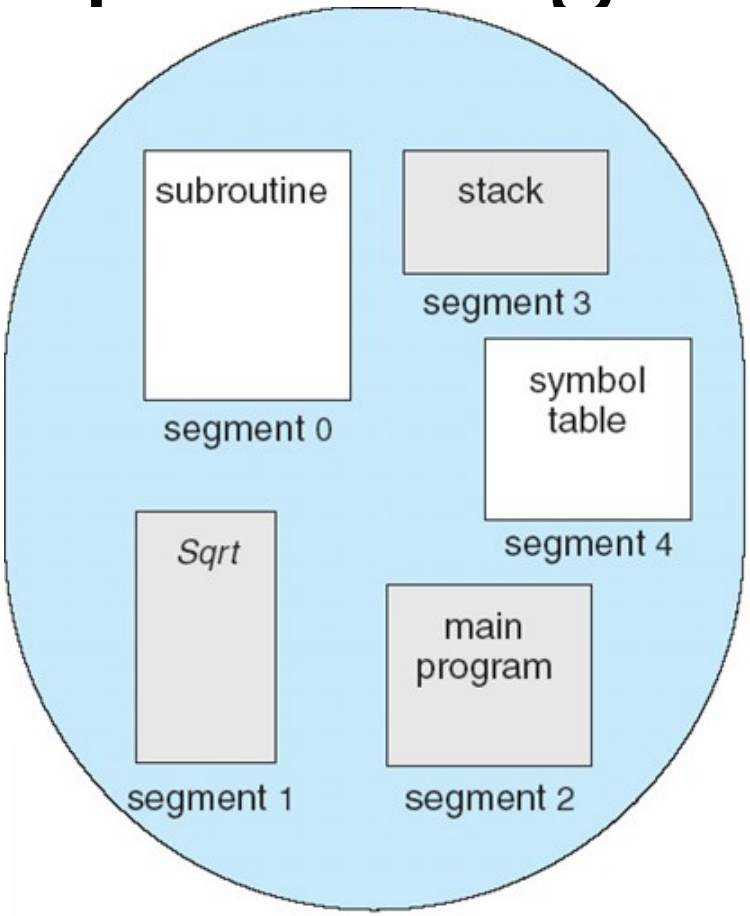
(syntactic [语法的], semantic [语义的])

Segmentation(分段)

- Segmentation could be seen as the extensi on of variable partitioning
  - Each program is subdivided into blocks of non-equal size called **segments**.
    - Cut your program according to semantic organizati on, such as following function, or class etc.
  - Allocate MM region whose size is just the size of the needed segment
    - When a process gets loaded into main memory, its d ifferent segments can be located anywhere.

PPTs from others\SCU_Zhaohui\OS\Chapter07.ppt

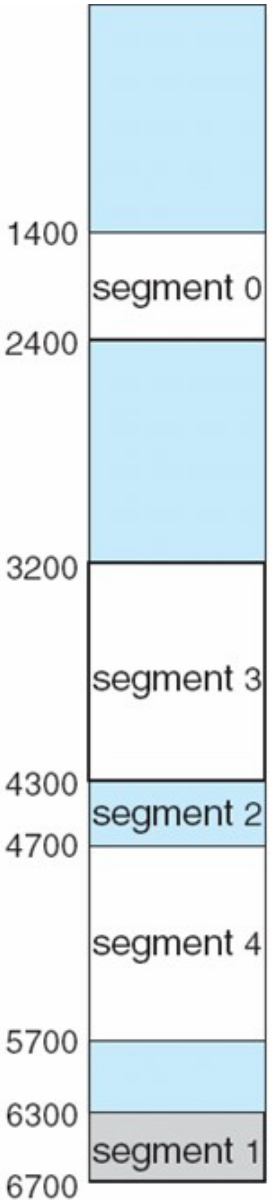
Example of Segmentation



logical address space

	limit	base
0	1000	1400
1	400	6300
2	400	4300
3	1100	3200
4	1000	4700

segment table



physical memory

Addressing consist of two parts - a segment number(段号) and an offset(位移量)

Dynamics of Simple Segmentation

- There is external fragmentation; it is reduced when using small segments.
  - Each segment is fully packed with instructions/data; **<u>no internal fragmentation</u>**.
- In contrast with paging, segmentation is visible to the programmer:
  - provided as a convenience to organize logically programs (**example: data in one segment, code in another segment**).
  - must be aware of segment size limit.
- The OS maintains a segment table for each process. Each entry contains:
  - the starting physical addresses of that segment.
  - the length of that segment (for protection).

Virtual Memory

- Paging
  - Basic paging
  - Paging-based VM
    - How to support the transparency of using space larger than the physical memory space
- Segmenting
  - Basic segmenting
  - Segmentation-based VM
    - How to support the transparency of using space larger than the physical memory space
- Segment-page scheme

Logical address used in segmentation

- Logical address now is divided into two parts:
  - (segment number, offset) = (s, d), the CPU indexes (with s) the **segment table** to obtain the starting physical address **b** and the length l of that segment.
- The physical address is obtained by adding d to b (in contrast with paging):
  - The hardware also compares the offset d with the length l of that segment to determine if the address is valid.

Address Translation in hybrid method

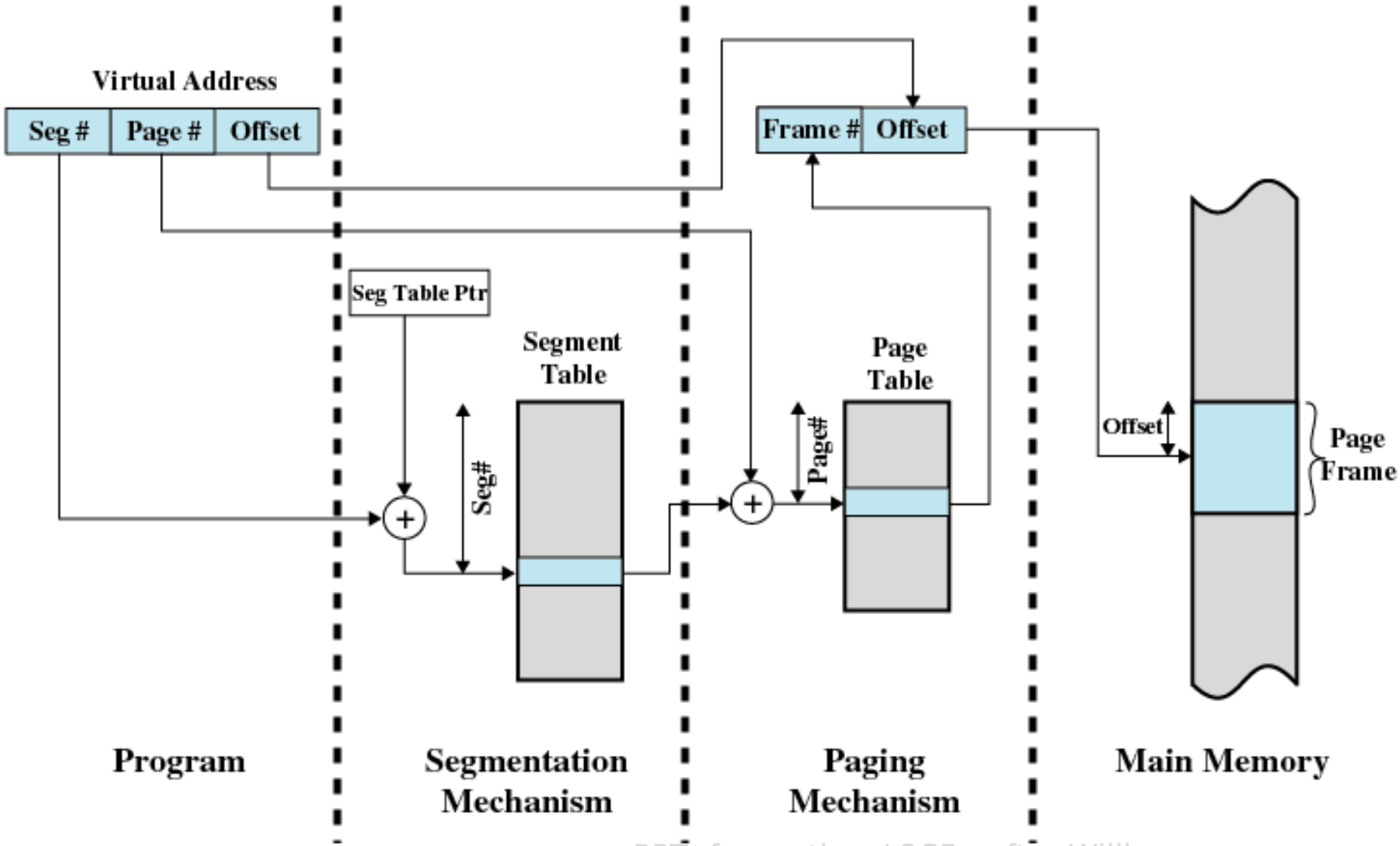
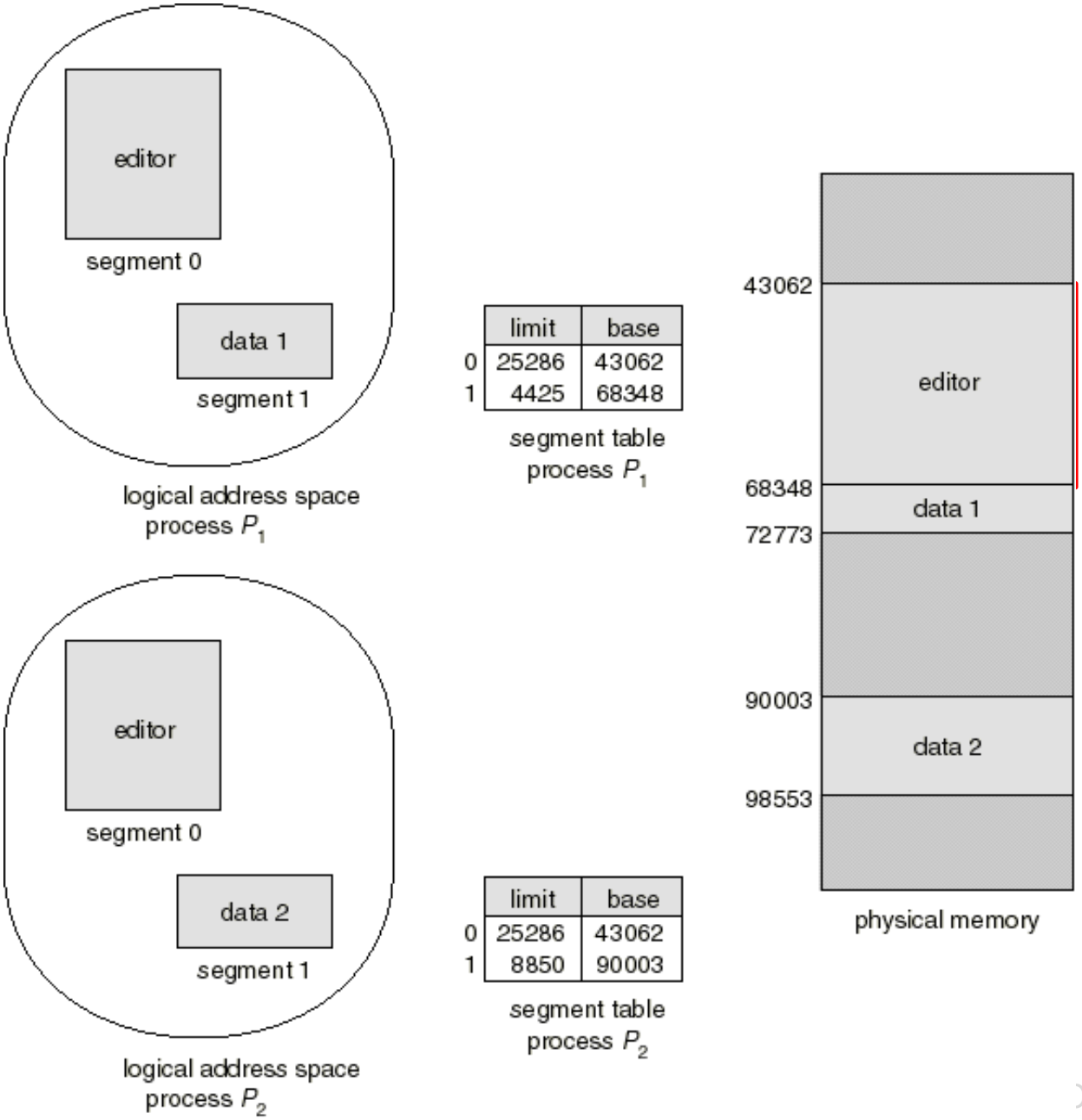


Figure 8.13 Address Translation in a Segmentation/Paging System

Sharing in Segmentation Systems

- Segments are shared when entries in the se gment tables of 2 different processes point to the same physical locations.
- Example: the same code of a text editor can be shared by many users:
  - Only one copy is kept in main memory.
- But each user would still need to have its o wn private data segment.

Shared Segments Example



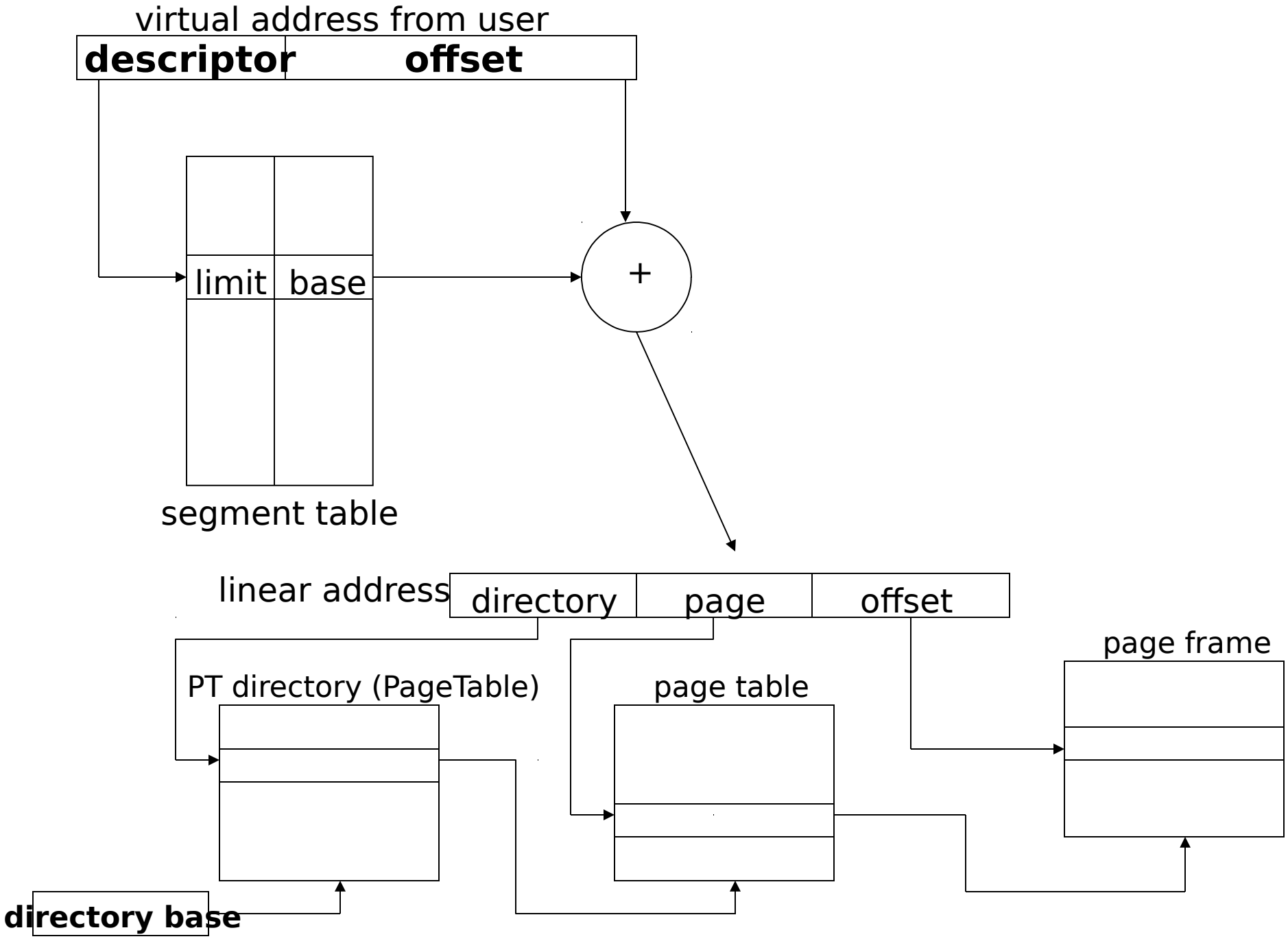
Virtual Memory

- Paging
  - Basic paging
  - Supporting VM
- Segmenting
  - Basic segmenting
  - Supporting VM
- Segment-page scheme

Segmentation + Paging

- Paging or segmentation?
- In the old days,
  - Motorola 68000 used paging.
  - Intel 80x86 used segmentation.
- Now
  - both combines paging and segmentation
- The OS for I386
  - OS/2 from IBM
  - NT from MS

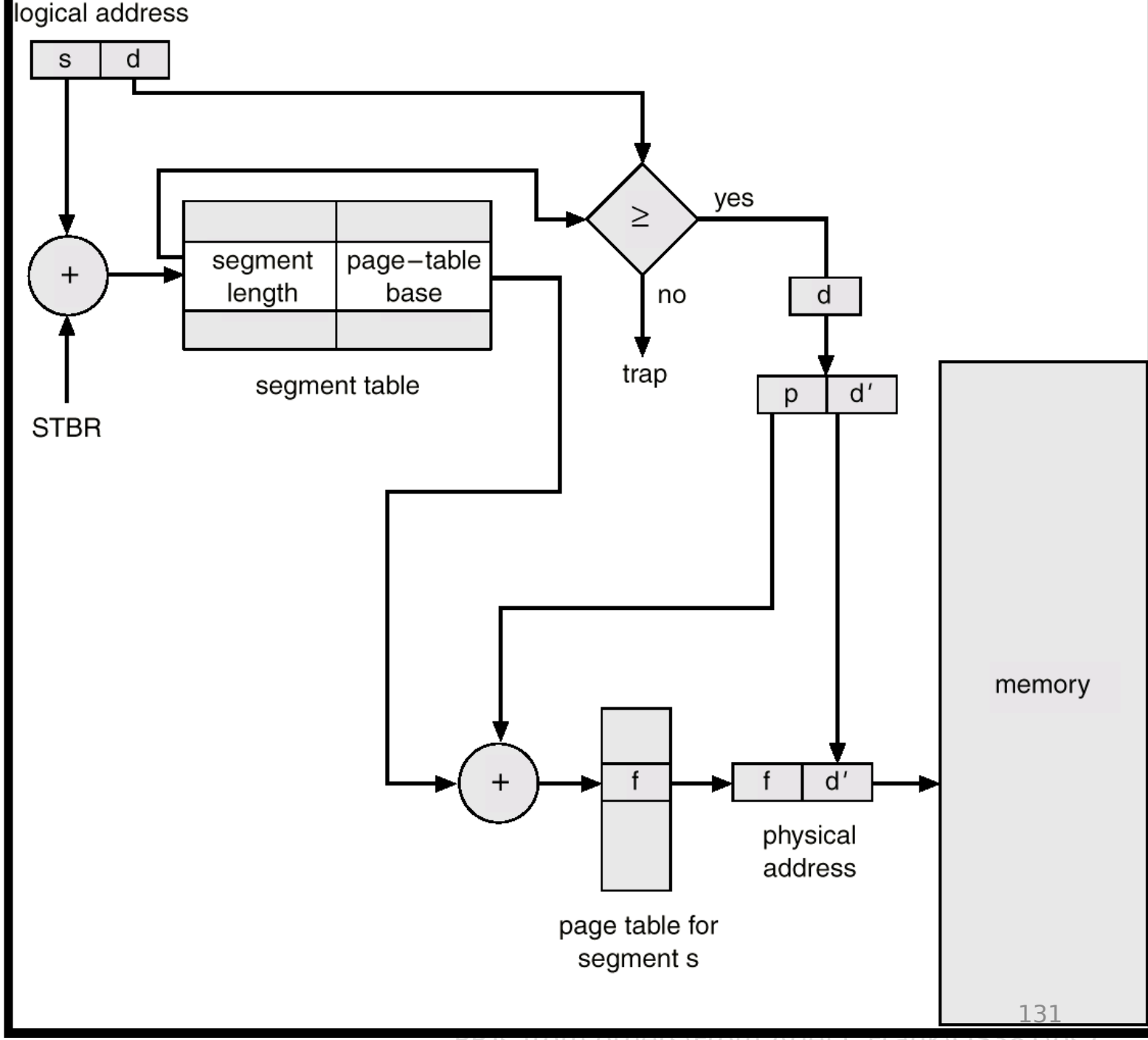
PPTs from others\OS PPT in English\ch09.p


virtual address from user
descriptor
offset
limit
base
+
segment table
linear address
directory
page
offset
page frame
PT directory (PageTable)
page table
directory base

Segmentation + Paging: **MULTICS**

- The MULTICS system solved problems of external fragmentation and lengthy search times by paging the segments.
- Solution differs from pure segmentation in that the segment-table entry contains not the base address of the segment, but rather the base address of a *page table* for this segment.
- Example in the next slide.

MULTICS Address Translation Scheme



Dealing with Large Page Tables

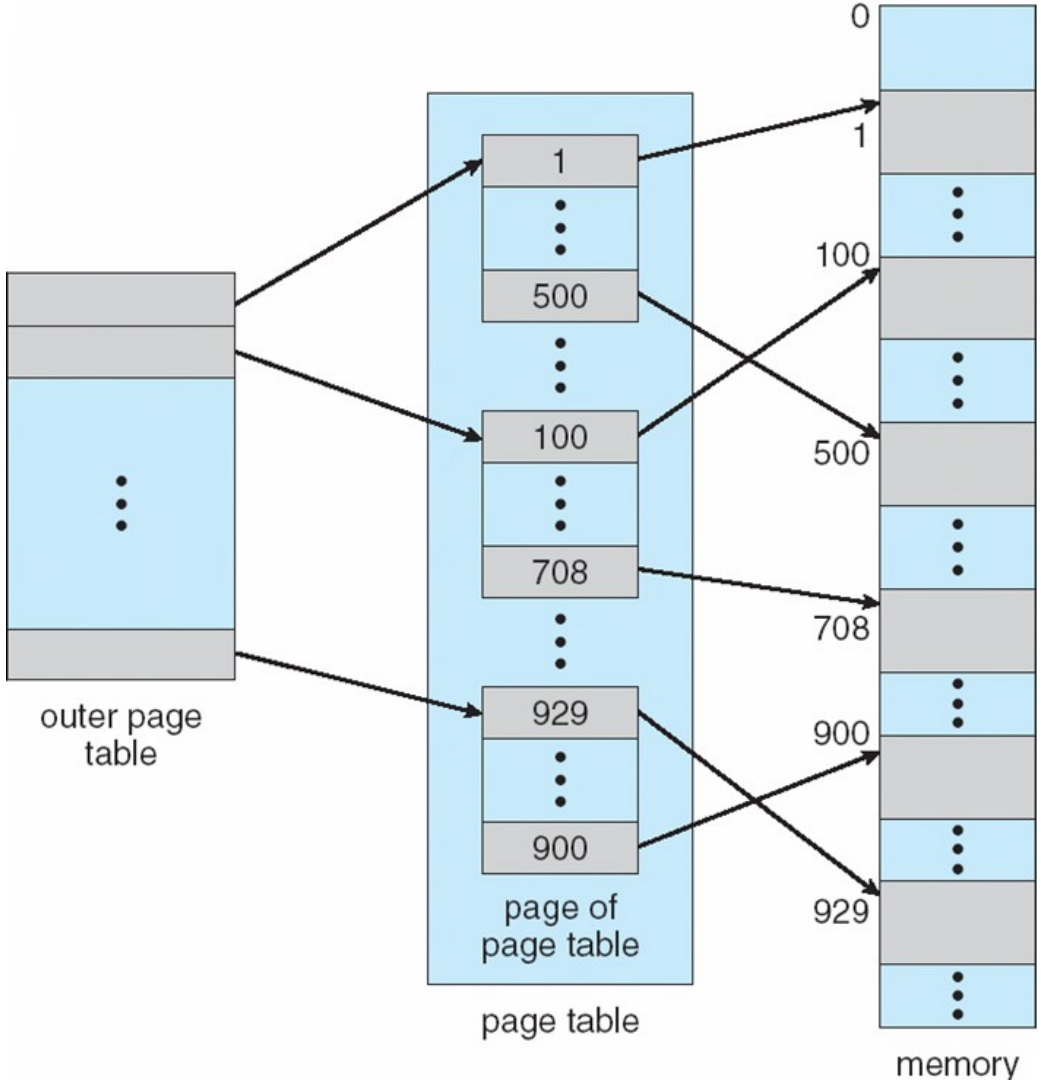
Structure of the Page Table

- Typically systems have large logical address space s
- Page table could be excessively large
- Example:
  - 32-bit logical address space
    - The number of possible addresses in the logical address spa ce is 2^{32}
  - Page size is 4 KB (4096 bytes or 2^{12})
  - Number of pages is 2^{20} (20 bits for page number)
  - Page table may consist of up to 1 million entries

Structure of Page Table

- Several approaches
  - Multi-Level Page tables
  - Hashed Page Tables
  - Inverted Page Tables

Two-Level Page-Table Scheme



- Page table is also paged
- Need to be able to index the outer page table

Two-Level Paging Example

- A logical address (on 32-bit machine with 1K page size) is divided into:
  - A page number consisting of 22 bits
  - A page offset consisting of 10 bits
- Since the page table is paged, the page number is further divided into:
  - A 12-bit page number
  - A 10-bit page offset

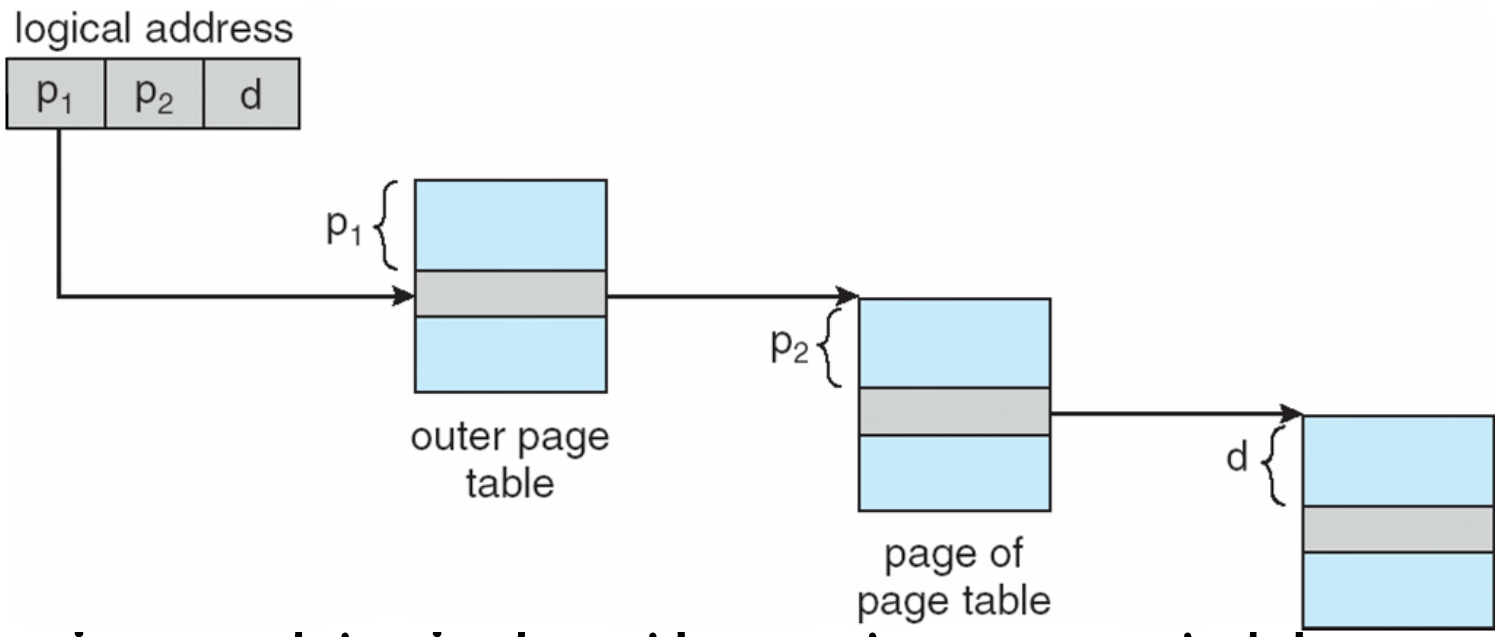
Two-Level Paging Example

- Thus, a logical address is as follows:

page number		page offset
p_1	p_2	d
12	10	10

where p_1 is an index into the outer page table, and p_2 is the displacement within the page of the outer page table

Address-Translation Scheme



- p_1 is used to index the outer page table
- p_2 is used to index the page table

Multi-level Paging

- What if you have a 64-bit architecture?
- Do you think hierarchical paging is a good id ea?
  - How many levels of paging are needed?
  - What is the relationship between paging and me mory accesses?
  - 64-bit means that even the outer page is large
    - The 64-bit UltraSPARC requires 7 levels of paging

Hashed Page Tables

- Common in address spaces larger than 32 b its
- The page number is hashed into a page tabl e
  - This page table contains a chain of elements ha shing to the same location
- Virtual page numbers are compared in this chain searching for a match
  - If a match is found, the corresponding physical frame is extracted

Case studies

Case Study – Windows

- Virtual memory with demand page
- Can support 32 or 64 bit
- Has a pool of free frames
- Uses prepaging (called clustering)
- What happens if the amount of free memory falls below some threshold?
  - Each process has a minimum number of processes
  - Windows will take away pages that exceed that min imum
- Applies LRU Locally

Case Study - Window

- Each process is guaranteed to have a minimum number of frames
- Each process has a maximum number of frames
- If a page fault occurs for a process that has the maximum number of frames a local replacement policy is used
- If a page fault occurs for a process that is below its working set maximum a free frame is used.

Case Study - Linux

- Virtual memory with demand paging
- Can support 32 or 64 bit
- Replacement
  - Least recently used (LRU) policy
  - Different implementations for different systems

Case Study-Android, IoS

- PCs and Servers: Support some form of sw apping
- Mobile devices -- Rely a lot on flash memor y for persistent storage
  - It's fast
  - Flash memory can tolerate a limited number of writes before it becomes unreliable
- Support
  - Typically no swapping
  - Paged systems

Case Study-Android

- No swap space for memory
- Does have
  - Paging
  - Map pages to physical pages
- Implications
  - Modified data (e.g., stack is not removed)
  - Read-only data (e.g., code) can be removed from the system and reloaded from flash memory

Case Study -- Android

zygote ['zaigəut]n. 合子，受精卵

- Sharing Memory
  - Each app is forked from an existing process called Zygote
    - Zygote loads framework code
    - RAM pages allocated for framework code is shared by application processes
  - Static data (e.g., code) is often mapped to specific pages
  - Some dynamic memory is explicitly shared by Android and applications
    - Example: Window surfaces use shared memory between the app and screen compositor

Case Study - Android

- Switching applications
  - Android keeps processes that are not hosting a foreground ("user visible") app component in a l east-recently used (**LRU**) cache.
  - The system keeps the process cached, so if th e user later returns to the app, the process is r eused for faster app switching.
  - As the system runs low on memory, it may kill p rocesses in the **LRU** cache beginning with the pr ocess least recently used

Case Study-Android

- Implications
  - Developers must carefully allocate and release memory to ensure that their applications do not use too much memory or suffer from memory le aks