Operating Systems

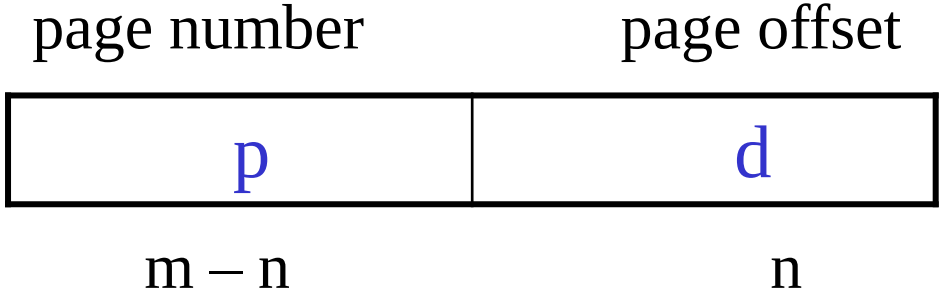
Jinghui Zhong （钟竞辉）
Office： B3-515
Email： jinghuizhong@scut.edu.cn

Contents

- Some issues related to paged memory management
  - ✓Local /Global allocation
  - ✓Page Size
  - ✓Share pages
  - ✓Paging Damon
  - ✓Page Fault
  - ✓Locking Pages
  - ✓Backing Store
- Segmentation

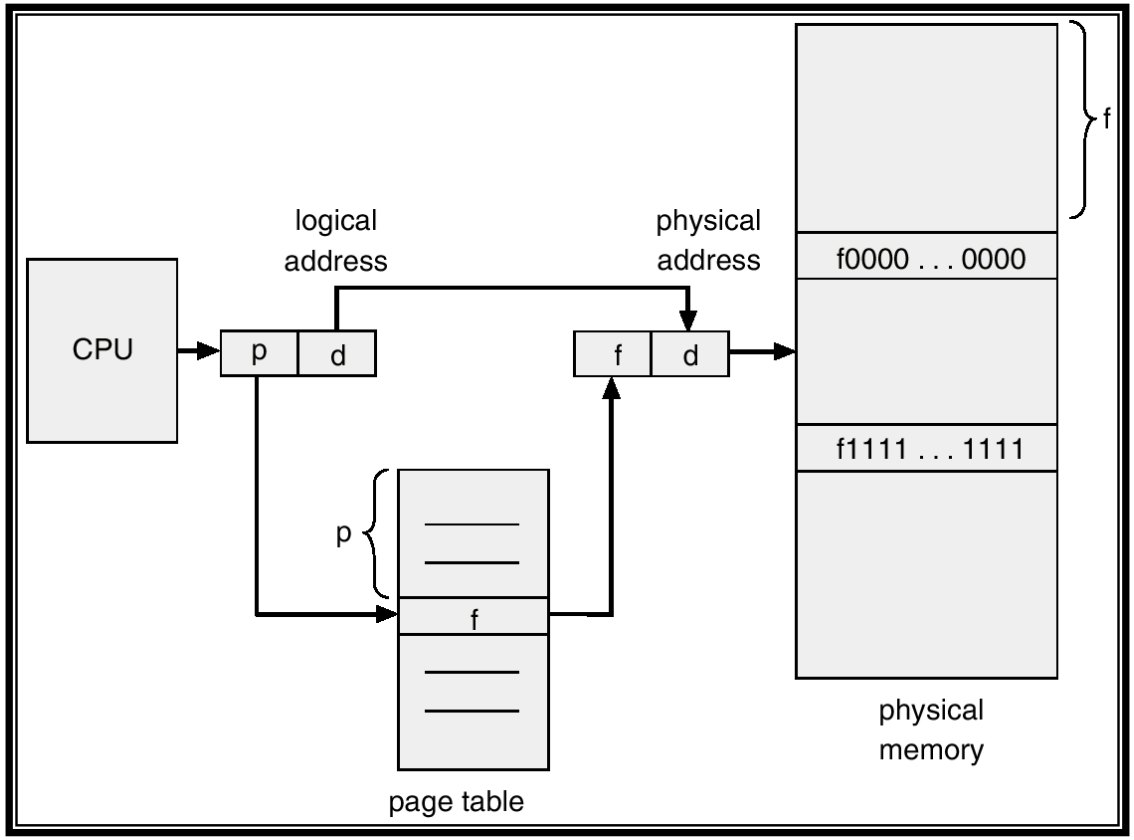
Paged Memory Management

- Address generated by CPU is divided into:
  - *Page number* (*p*) – used as an index into a *page table* which contains base address of each page in physical memory.
  - *Page offset* (*d*) – combined with base address to define the physical memory address that is sent to the memory unit.



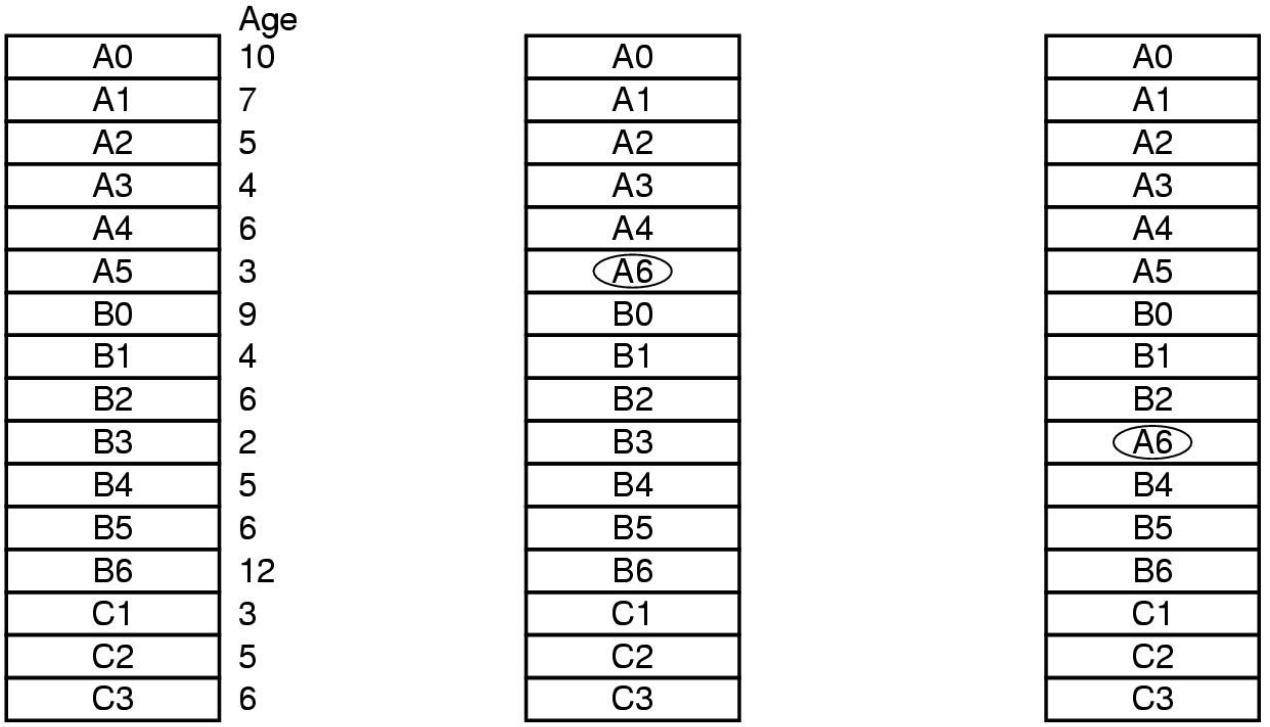
Where **p** is an index to the page table and **d** is the displacement within the page.

Paged Memory Management



Local versus Global Allocation Policies

- Global algorithms dynamically allocate page frames among all runnable processes. Local algorithms allocate pages for a single process.

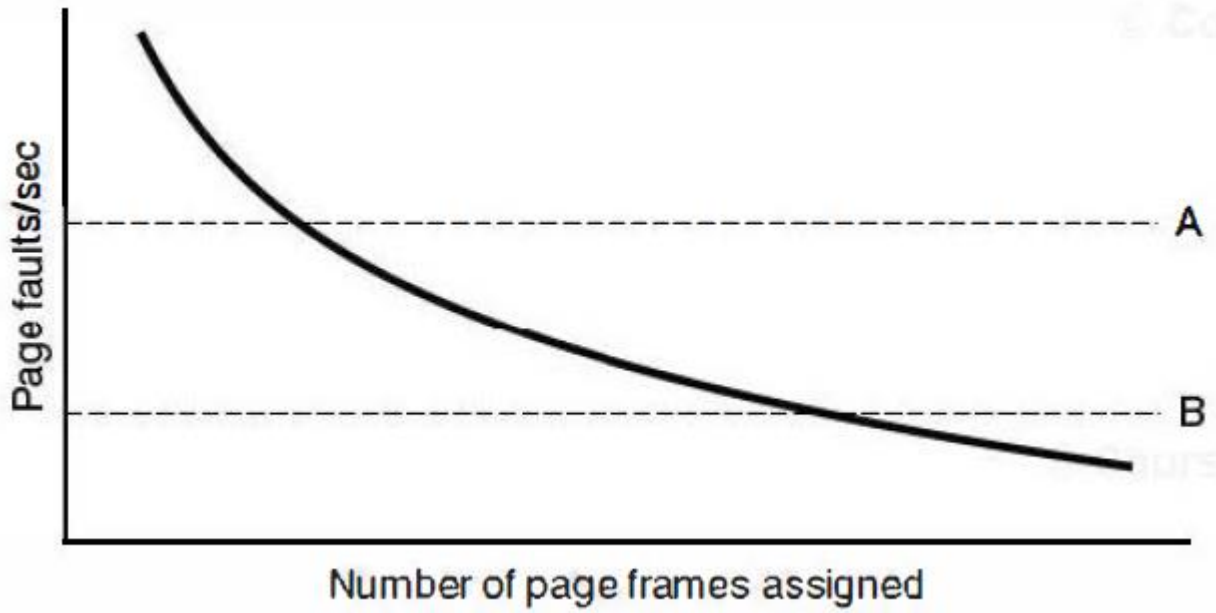


Local versus Global Allocation Policies

- A global algorithm is used to prevent thrashing and keep the paging rate within acceptable bounds:

✓A：too high ➔ assign more page frames to the process.

✓B：too low ➔ assign process fewer page frames.



Page Size

- Small page size

✓**Advantages:**

less internal fragmentation

✓**Disadvantages:**

programs need many pages → larger page tables

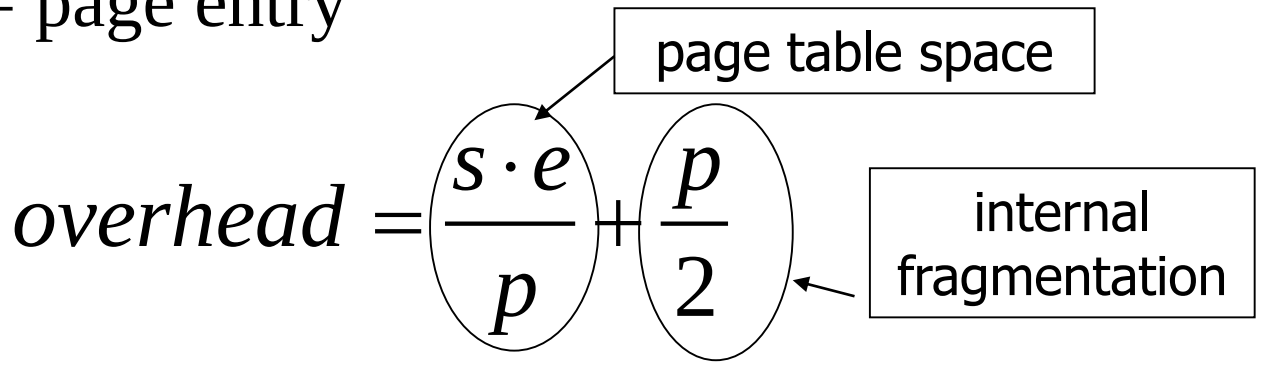
Page Size

- Overhead due to page table and internal fragmentation

  s = average process size in bytes,

  p = page size in bytes,

  e = page entry



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

Page Size

- Example:

  s = 1M

  e = 8

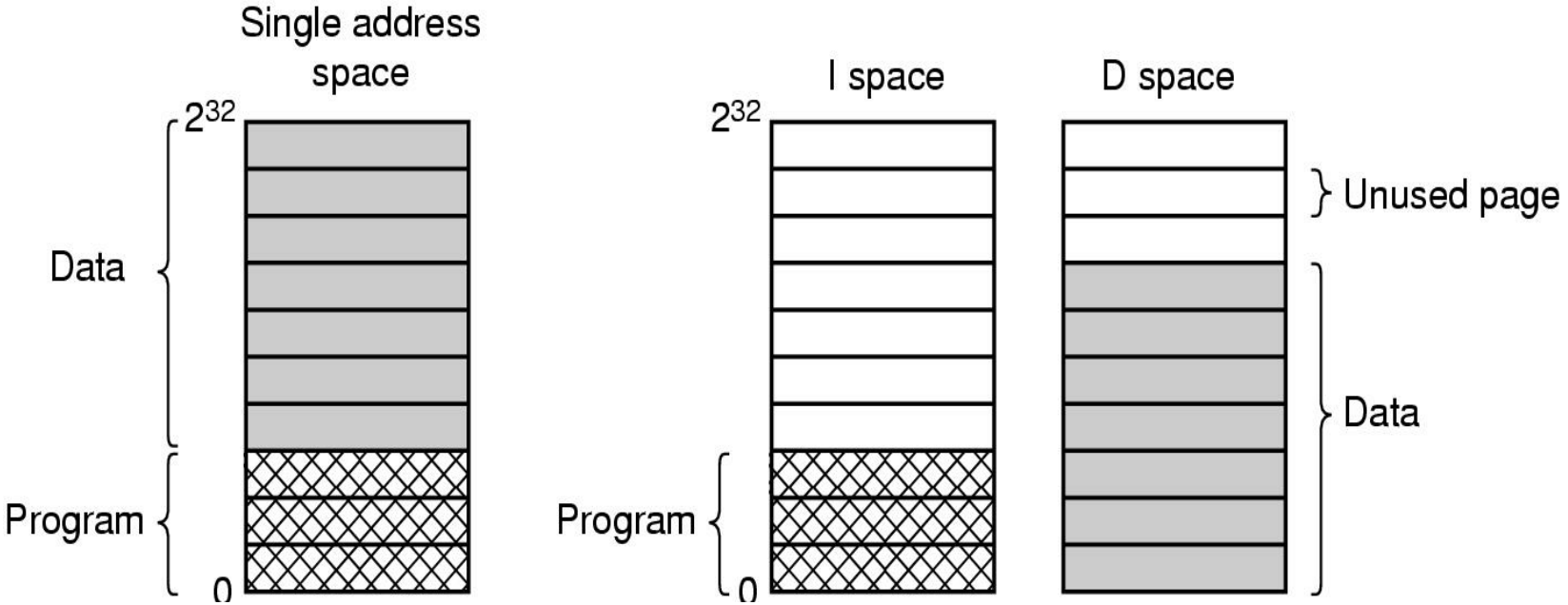
  ==》 p = square root of (2(1M)(8)) = 4K

- Common range:

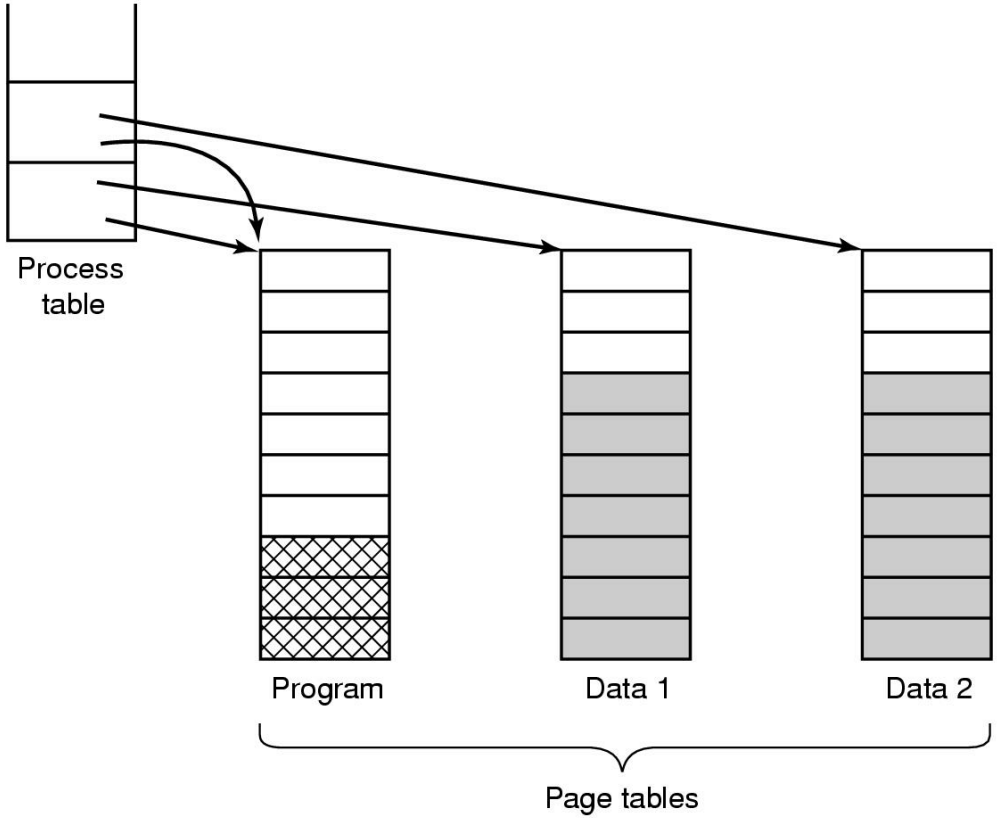
  512 <= p <= 64k

Separate Instruction and Data Spaces

- Most systems separate address spaces for instructions (program text) and data. A process can have two pointers in its process table: one to the instruction page and one to the data page. A shared code can be pointed by two processes.



Shared Pages



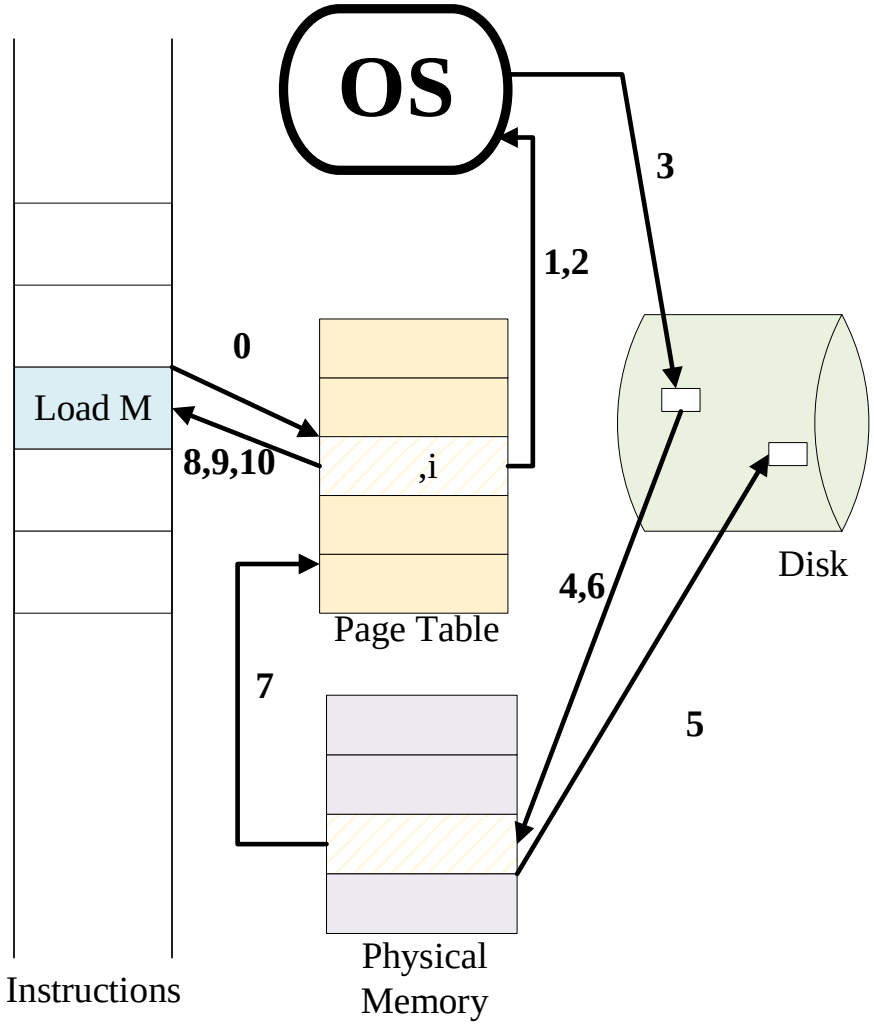
Two processes sharing same program sharing its page table

Cleaning Policy

- **Paging Daemon:**
  - ✓A background process, in sleep state in most of the time;
  - ✓Periodically woken up to inspect state of memory
  - ✓When too few frames are free, selects pages to evict using a replacement algorithm.

Page Fault Handling

1. Hardware traps to kernel
2. Save general registers
3. Determines which virtual page needed
4. Seeks page frame
5. If the selected frame is dirty, write it to disk
6. Brings new page in from disk
7. Page tables updated
8. Instruction backed up to when it began
9. Faulting process scheduled
10. Registers restored & Program continues

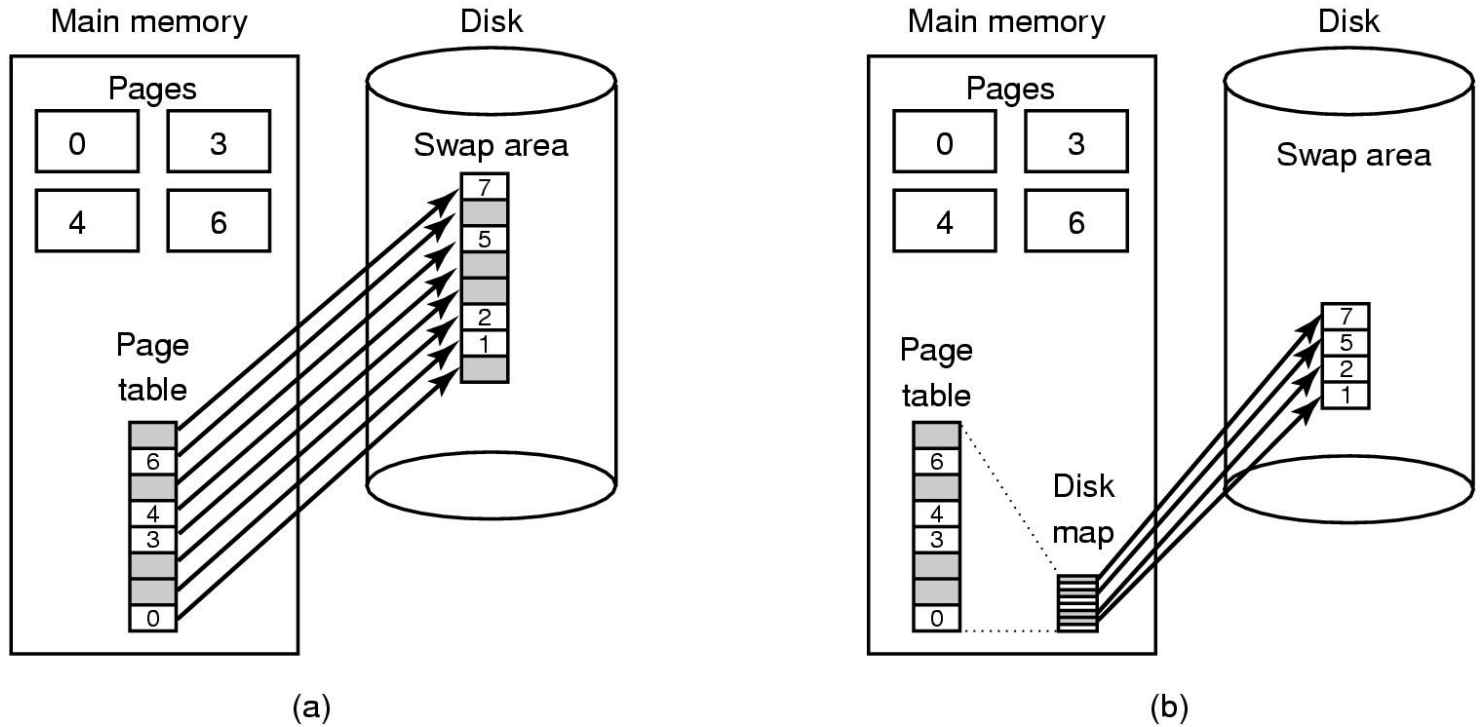


Locking Pages in Memory

- Process issues call for read from device into buffer
  - ✓while waiting for I/O, another process starts up
  - ✓has a page fault
  - ✓buffer for the first process may be chosen to be paged out
- If a page transferring data through the I/O is paged out, it will cause part of the data in buffer and part in the newly loaded page. In this case, the page need to be locked (**pinning**).

Backing Store

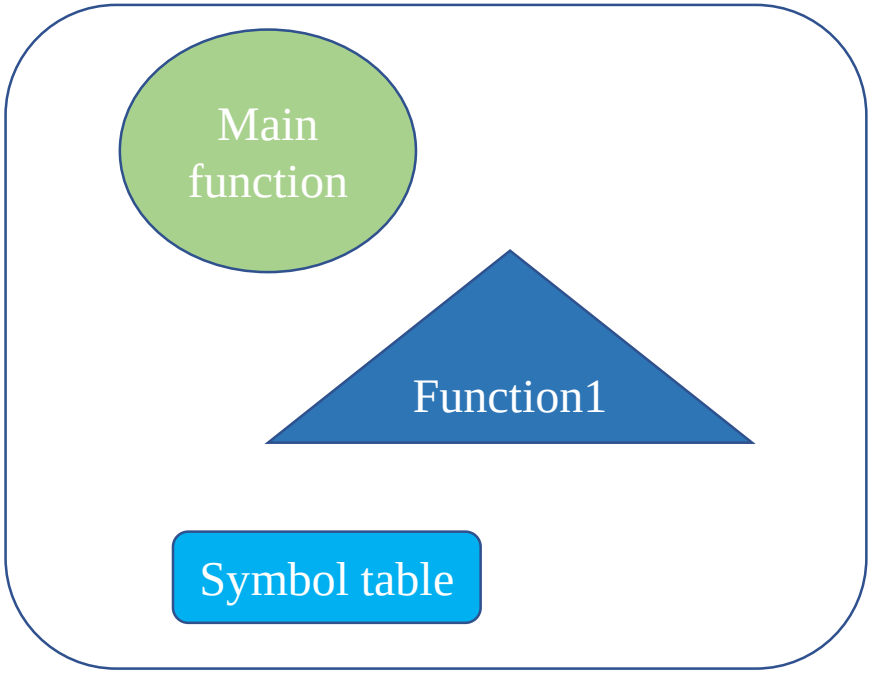
- Two approaches to allocate page space on the disk:
  1）Paging to static swap area
  2）Backing up pages dynamically with a disk map



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

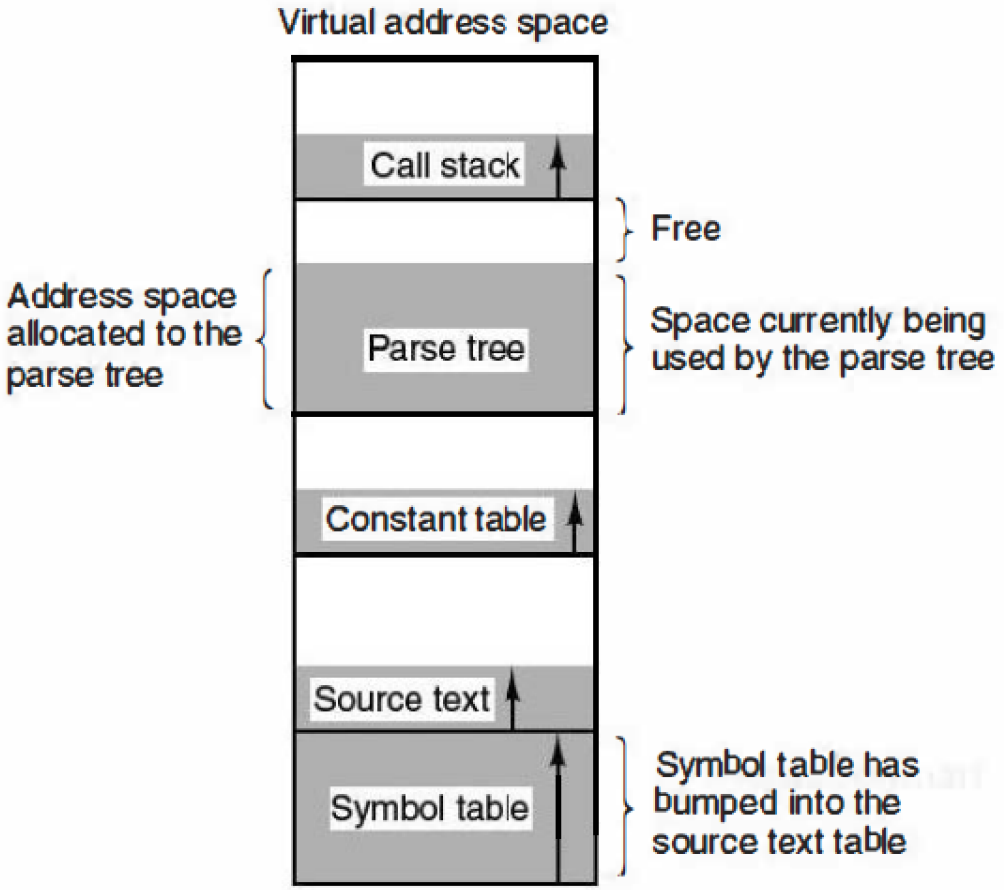
Segmentation

- **Programmer's view memory is not usually as a single linear address space:**

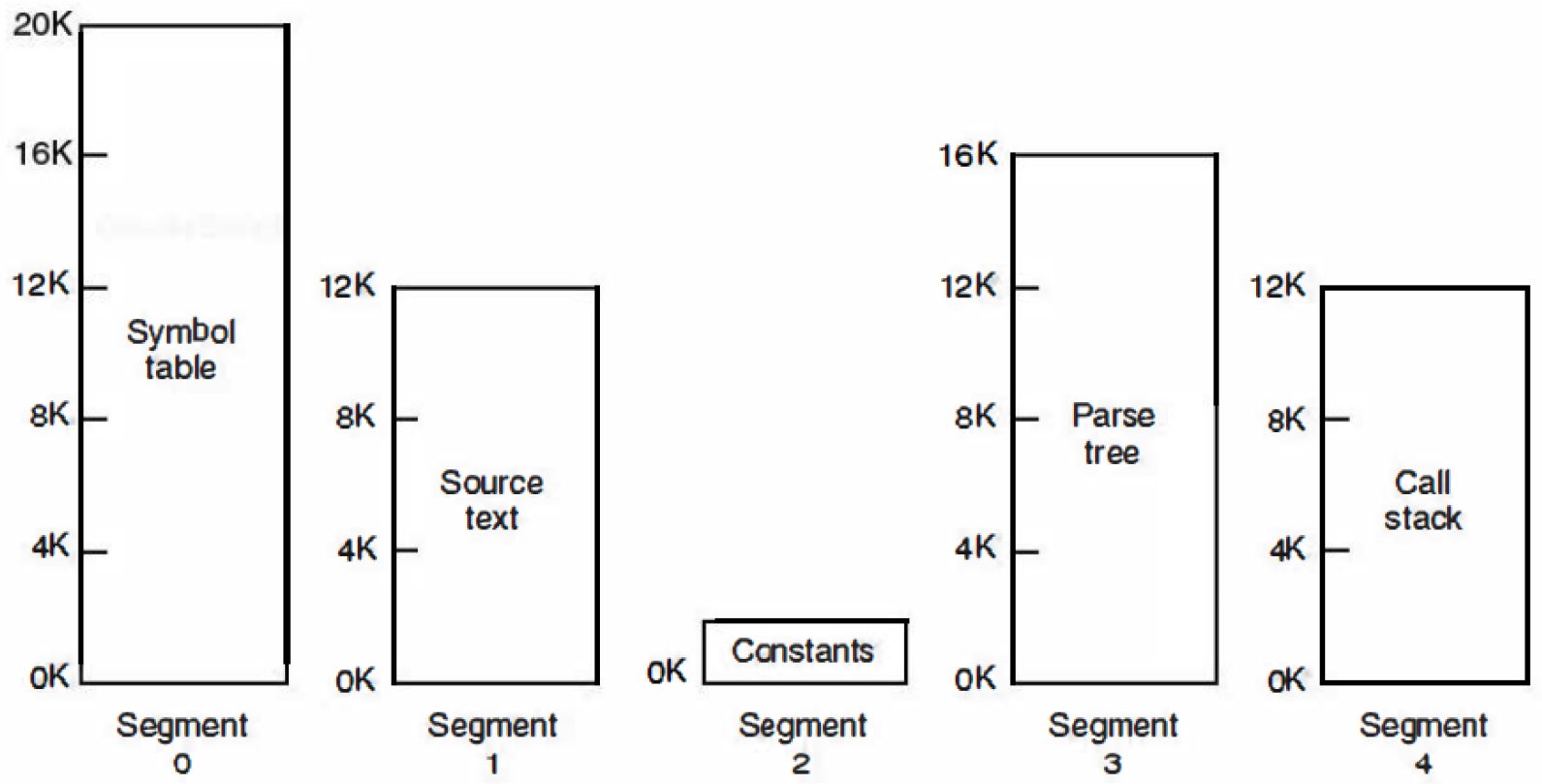


- **Programmer doesn't know how large these will be, or how they will grow, and doesn't want to manage where they go in virtual memory.**

Compiler with One-dimensional Address Space

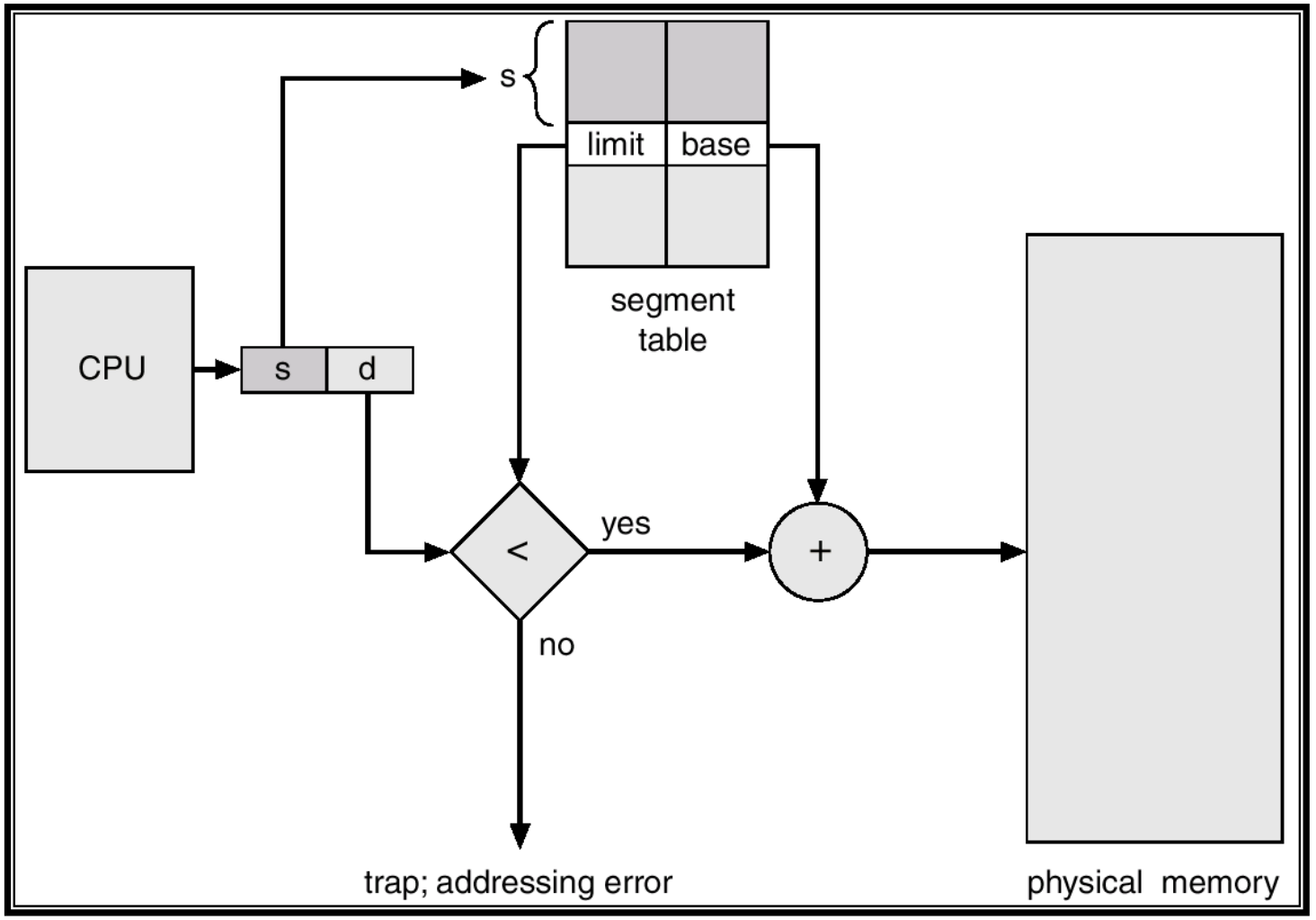


Compiler with Segmentation



- Segmentation maintains multiple separate virtual address spaces per process.
- Allows each table to grow or shrink, independently.

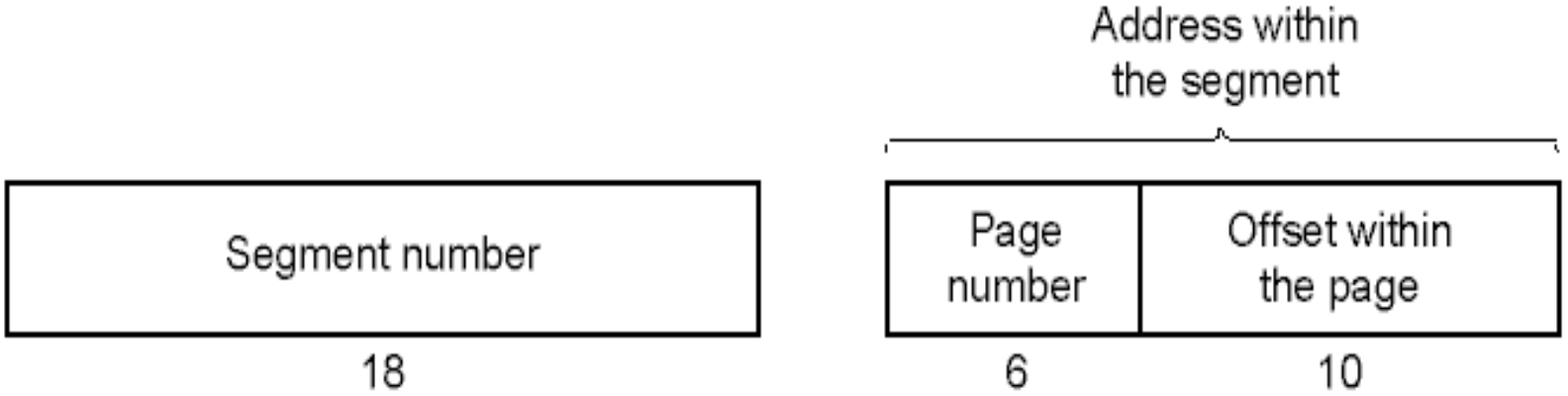
Pure Segmentation



Segmentation

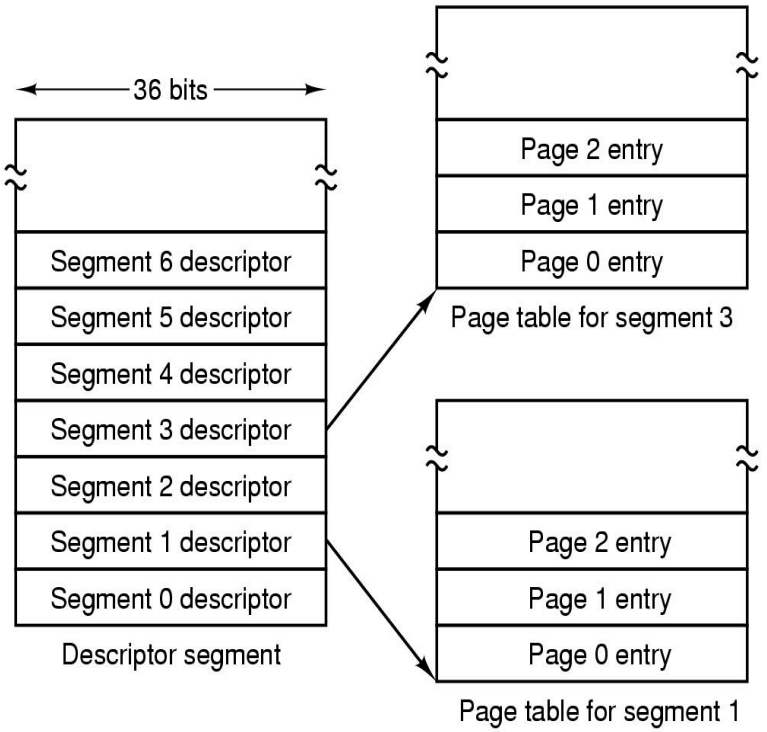
- **A segment is a logically independent address space.**
  - ✓segments may have different sizes
  - ✓their sizes may change dynamically
  - ✓the address space uses 2-dimensional memory addresses and has 2 parts:
  - ✓(segment #, offset within segment)
  - ✓segments may have different protections
  - ✓allows for the sharing of procedures and data between processes.

Segmentation with Paging (MULTICS)

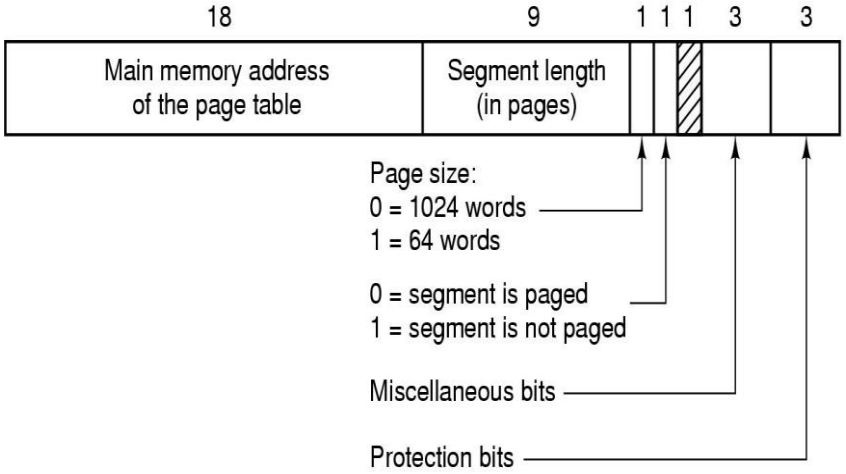


A 34-bit virtual address

Segmentation with Paging (MULTICS)

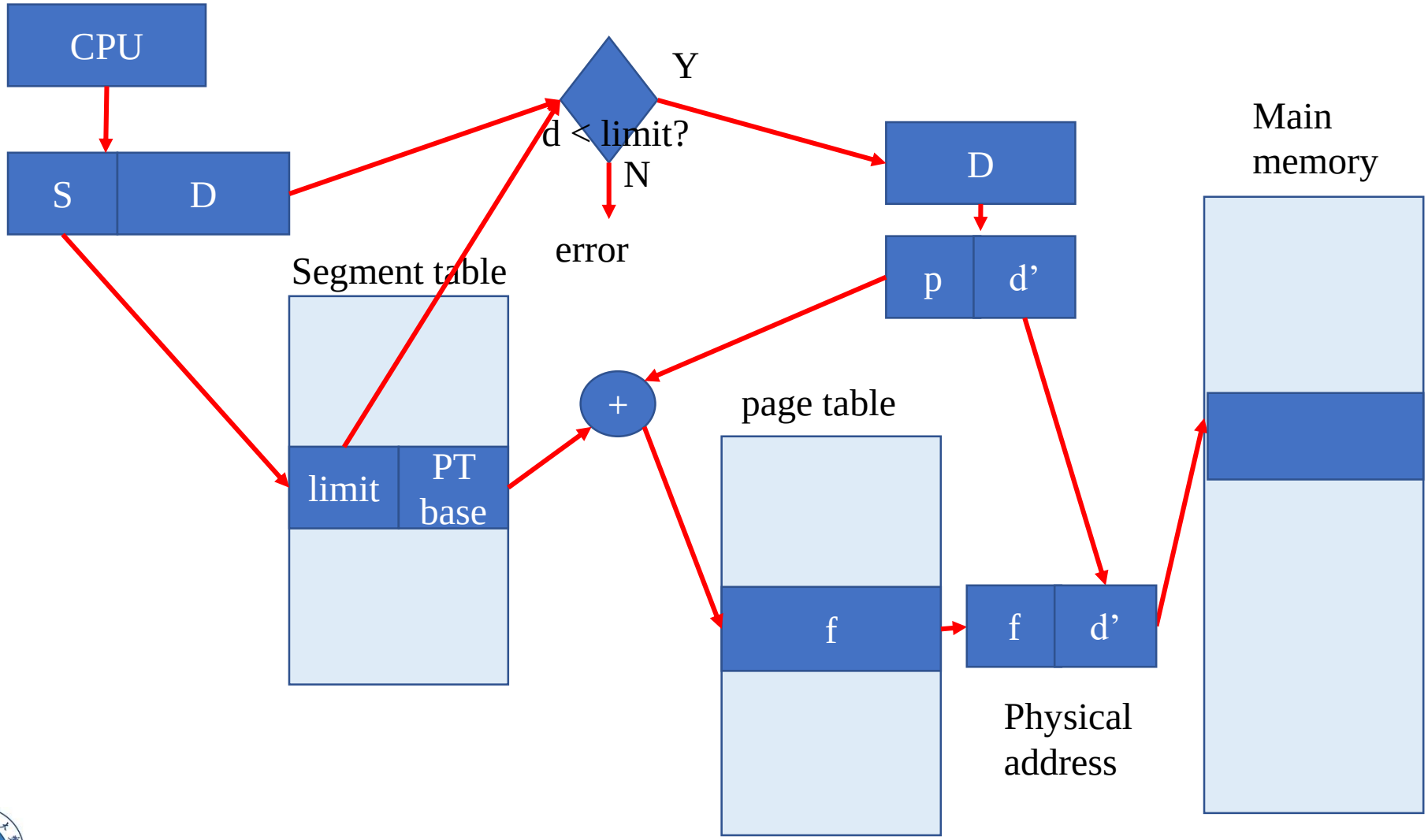


(a) Descriptor segment points to page tables.



(b) Segment descriptor – numbers are field lengths

Segmentation with Paging (MULTICS)



Segmentation with Paging (MULTICS)

Comparison field: Segment number	Comparison field: Virtual page	Page frame	Protection	Age	Is this entry used?
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 (**Translation Look-aside Buffers**)

Check Points

A computer has four page frames. The time of loading, time of last access, and the *R* and *M* bits for each page are as shown below (the times are in clock ticks):

Page	Loaded	Last ref.	R	M
0	126	280	1	0
1	230	265	0	1
2	140	270	0	0
3	110	285	1	1

(a) Which page will NRU replace?
(b) Which page will FIFO replace?
(c) Which page will LRU replace?
(d) Which page will second chance replace?

Check Points

A small computer on a smart card has four page frames. At the first clock tick, the *R* bits are 0111 (page 0 is 0, the rest are 1). At subsequent clock ticks, the values are 1011, 1010, 1101, 0010, 1010, 1100, and 0001. If the aging algorithm is used with an 8-bit counter, give the values of the four counters after the last tick.

Suppose that a machine has 48-bit virtual addresses and 32-bit physical addresses.

(a) If pages are 4 KB, how many entries are in the page table if it has only a single level? Explain.

(b) Suppose this same system has a TLB (Translation Lookaside Buffer) with 32 entries. Furthermore, suppose that a program contains instructions that fit into one page and it sequentially reads long integer elements from an array that spans thousands of pages. How effective will the TLB be for this case?

Check Points

Consider the following C program:

```
int     X[N];
int step = M;       /* M is some predefined constant */
for (int i = 0; i < N; i += step) X[i] = X[i] + 1;
```

(a) If this program is run on a machine with a 4-KB page size and 64-entry TLB, what values of M and N will cause a TLB miss for every execution of the inner loop?

(b) Would your answer in part (a) be different if the loop were repeated many times? Explain.

Presentation & Poster

① Group 1,10: Process Scheduling
② Group 2,11: The Banker's Algorithm
③ Group 3,12: Virtual Memory Paging
④ Group 4,13: Page Replacement Algorithm
⑤ Group 5,14: Memory Management (Link List)
⑥ Group 6,15: Disk space Management
⑦ Group 7,16: Interrupt/DMA
⑧ Group 8,17: Disk Arm Scheduling Algorithm
⑨ Group 9,18: Files: Link List Allocation

华南理工大学2019～2020学年度第一学期校历

年月、周次 / 日期 / 星期	2019年																		2020年							
	8月	9月				10月				11月					12月				1月		2月					
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	寒 假					
星期一	26	2	9	16	23	30	7	14	21	28	4	11	18	25	2	9	16	23	30	6	13	20	27	3	10	17
星期二	27	3	10	17	24	1	8	15	22	29	5	12	19	26	3	10	17	24	31	7	14	21	28	4	11	18
星期三	28	4	11	18	25	2	9	16	23	30	6	13	20	27	4	11	18	25	1	8	15	22	29	5	12	19
星期四	29	5	12	19	26	3	10	17	24	31	7	14	21	28	5	12	19	26	2	9	16	23	30	6	13	20
星期五	30	6	13	20	27	4	11	18	25	1	8	15	22	29	6	13	20	27	3	10	17	24	31	7	14	21
星期六	31	7	14	21	28	5	12	19	26	2	9	16	23	30	7	14	21	28	4	11	18	25	1	8	15	22
星期日	1	8	15	22	29	6	13	20	27	3	10	17	24	1	8	15	22	29	5	12	19	26	2	9	16	23
	教 学 18 周																		考试2周		寒 假 6 周					

第6周	星期一	5-8节	B3	138
第12周	星期二	5-8节	B3	231
第13周	星期一	1-4节	B3	138
第13周	星期三	1-4节	B3	138
第13周	星期五	1-4节	B3	234

Session 1

【Objective and Requirement】

Objective： Be familiar with the creation of process and thread.

Requirement：

Task 1: Create a console application, “child”, which keeps printing out “The child is talking at [system time]” (in a loop, one per 1s).

Task 2: Create another console application, “parent”. It create a child **process** to execute “child”. At the same time, the “parent” process keeps printing out “The parent is talking at [system time]”. (one per 1s). Execute “parent” and explain the output you see.

Task 3: Create a child thread in the “mainThread” program. Both the main thread and the child thread keep printing out “[ThreadID] + [System time]”.

Task 4: Create a console application, which contains a shared integer shared_var. The initial value of shared_var is 0. The application will create a child **thread** after it starts. The main thread keeps increasing the value of shared_var by 1, while the child thread keeps decreasing the value of shared_var by 1. Explain the observed results.

【Environment】

Operating System：Linux;