OPERATING SYSTEMS

INTRODUCTION TO MEMORY MANAGEMENT

8 Memory Management

1

In a multiprogramming system, in order to share the processor, a number of processes must be kept in memory. Memory management is achieved through memory management algorithms. Each memory management algorithm requires its own hardware support. First we'll look requirements for memory management, and then techniques.

8 Memory Management

2

Introduction to memory management

- Requirements
  - Relocation
  - Protection
  - Sharing
  - Logical Organization
  - Physical Organization
- Techniques for mechanism of memory management
  - Partitioning
  - Paging
  - Segmentation

8.1 Requirements

3

- Requirements
  - Relocation
  - Protection
  - Sharing
  - Logical Organization
  - Physical Organization

8.1.1 Relocation

4

- Why
  - programmer does not know in advance where the program will be placed in memory when it is first loaded
  - while the program is executing, it may be swapped to disk and returned to main memory at a different location
- Consequences
  - memory references must be translated in the code to actual physical memory address. There are two type of relocation:
    - **Static** relocation is performed before or during the loading of program into memory
    - **Dynamic** relocation is performed during the execution of the program

8.1.2 Protection

5

- Why
  - Protect process from interference by other processes which requires **permission** to access its address space.
- Consequences
  - impossible to check addresses in advance since the program could be relocated
  - **must be checked at run time**

8.1.3 Sharing

6

- Why
  - Some processes can use the same programs or data. That leads to wasteful use of the memory.
- Consequences
  - Allow several processes to access the same data
  - Allow multiple programs to share the same program text

8.1.4 Logical Organization

7

- Programs organized into modules
  - stack, text, uninitialized data, libraries, etc.
- Modules may be compiled independently
- Different degrees of protection given to each modules
  - read-only, execute-only
- And thus modules may be shared

8.1.4 Physical Organization

8

- Memory organized into two levels:
  - *main and secondary memory*.
- Main memory relatively fast, expensive and volatile
- Secondary memory relatively slow, cheaper, larger capacity, and non-volatile
- Since sometimes main memory may be insufficient for a program with its data, secondary memory is used by OS without asking anything from users.

8.2 Partitioning

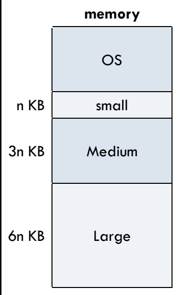
9

We will see two type of partitioning with their terms:

- Fixed Partitioning
  - Swapping
  - Fragmentation
- Variable Partitioning
  - First Fit
  - Best Fit
  - Worst Fit
  - Compaction

8.2.1 Fixed Partitioning

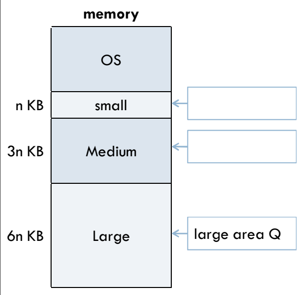
10



- In this method, memory is divided into partitions whose sizes are fixed.
- OS is placed into the lowest bytes of memory.
- Relocation of processes is not needed

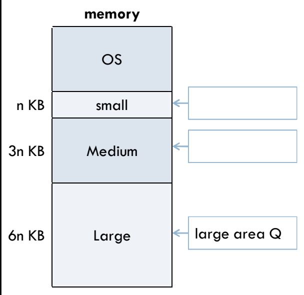
8.2.1 Fixed Partitioning

11



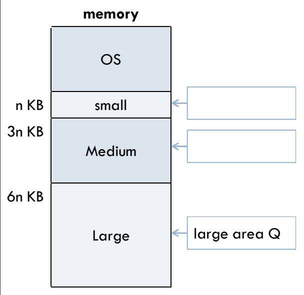
- Processes are classified on entry to the system according to their memory they requirements.
- We need one *Process Queue (PQ)* for each class of process.

8.2.1 Fixed Partitioning



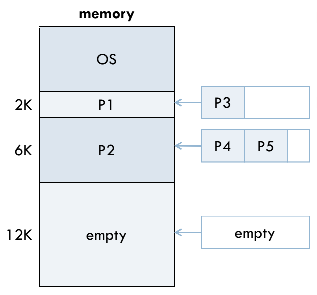
- If a process is selected to allocate memory, then it goes into memory and competes for the processor.
- The number of fixed partition gives the degree of multiprogramming.
- Since each queue has its own memory region, there is no competition between queues for the memory.

8.2.1 Fixed Partitioning



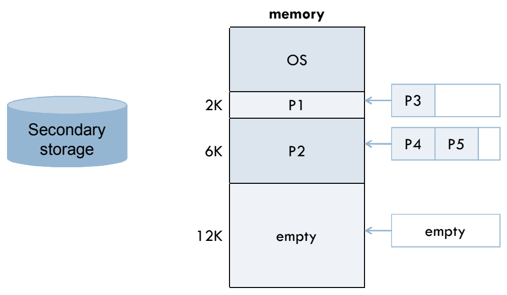
- The main problem with the fixed partitioning method is how to determine the number of partitions, and how to determine their sizes.

Fixed Partitioning with Swapping

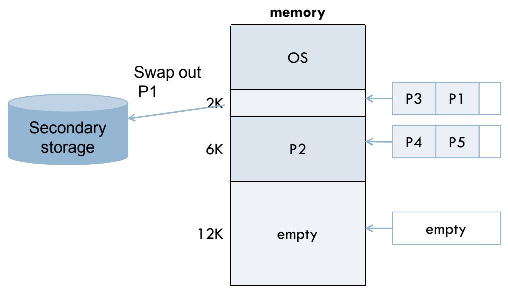


- This is a version of fixed partitioning that uses RRS with some time quantum.
- When time quantum for a process expires, it is swapped out of memory to disk and the next process in the corresponding process queue is swapped into the memory.

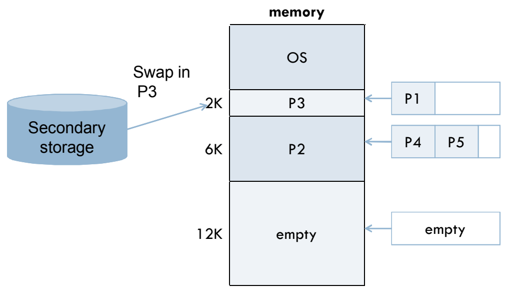
Fixed Partitioning with Swapping



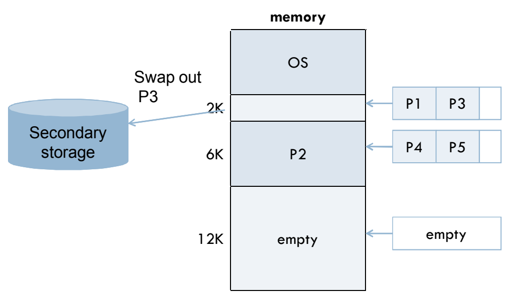
Fixed Partitioning with Swapping



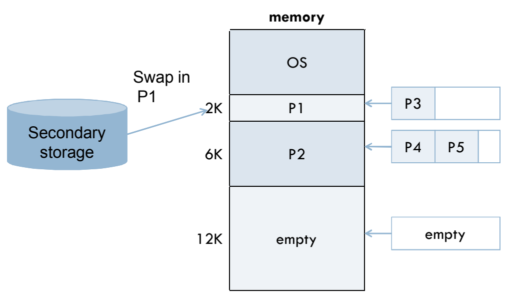
Fixed Partitioning with Swapping



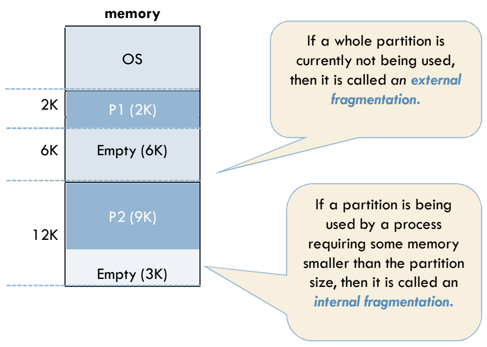
Fixed Partitioning with Swapping



Fixed Partitioning with Swapping



Fragmentation


memory
OS
2K
P1 (2K)
6K
Empty (6K)
12K
P2 (9K)
Empty (3K)
If a whole partition is currently not being used, then it is called an external fragmentation.
If a partition is being used by a process requiring some memory smaller than the partition size, then it is called an internal fragmentation.

8.2.2 Variable Partitioning

- With fixed partitions we have to deal with the problem of determining the number and sizes of partitions to minimize internal and external fragmentation.
- If we use variable partitioning instead, then partition sizes may vary dynamically.
- In the variable partitioning method, we keep a table (linked list) indicating used/free areas in memory.

8.2.2 Variable Partitioning

- Initially, the whole memory is free and it is considered as one large block.
- When a new process arrives, the OS searches for a block of free memory large enough for that process.
- We keep the rest available (free) for the future processes.
- If a block becomes free, then the OS tries to merge it with its neighbors if they are also free.

8.2.2 Variable Partitioning

There are three algorithms for searching the list of free blocks for a specific amount of memory.

- First Fit
- Best Fit
- Worst Fit

First fit

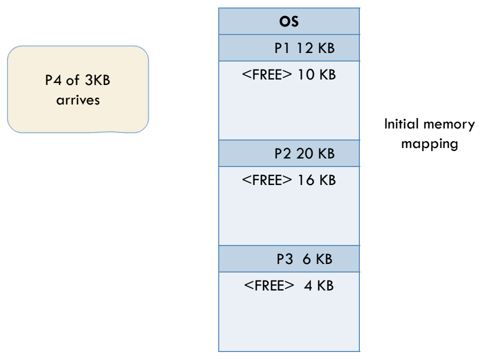
24

- Allocate the first free block that is large enough for the new process.

- This is a fast algorithm.

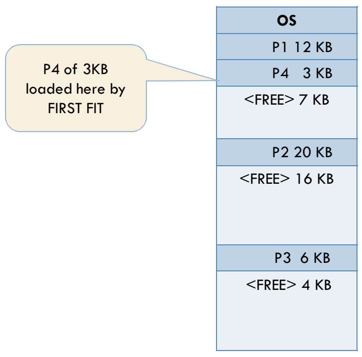
First fit

25



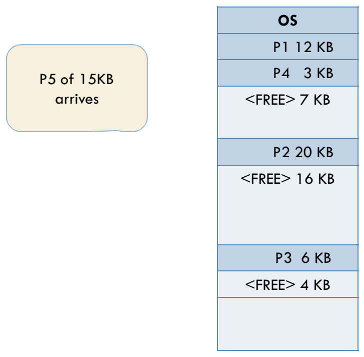
First fit

26



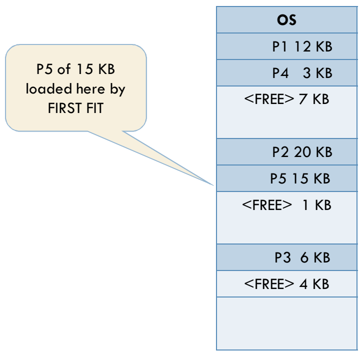
First fit

27



First fit

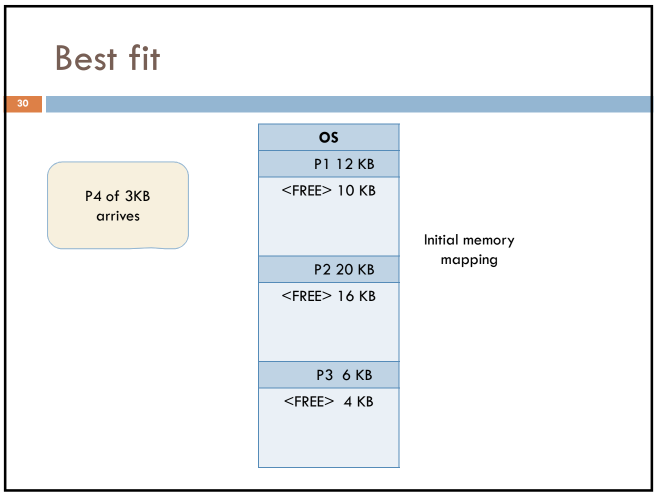
28



Best fit

29

- Allocate the smallest block among those that are large enough for the new process.
- In this method, the OS has to search the entire list, or it can keep it sorted and stop when it hits an entry which has a size larger than the size of new process.
- This algorithm produces the smallest left over block.
- However, it requires more time for searching all the list or sorting it
- If sorting is used, merging the area released when a process terminates to neighboring free blocks, becomes complicated.

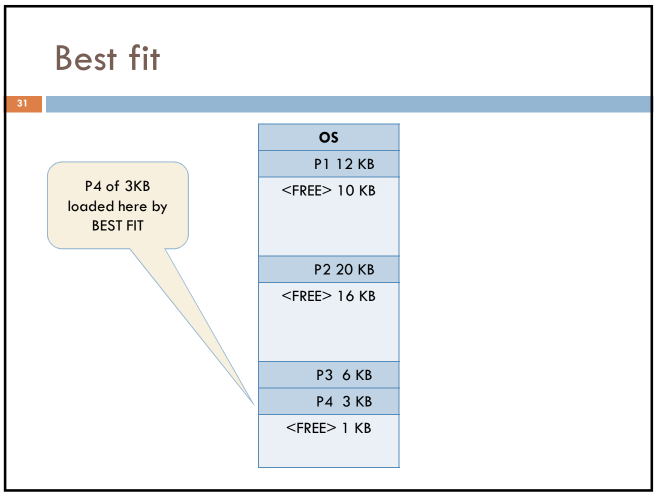


Best fit

P4 of 3KB arrives

OS
P1 12 KB
<FREE> 10 KB
P2 20 KB
<FREE> 16 KB
P3 6 KB
<FREE> 4 KB

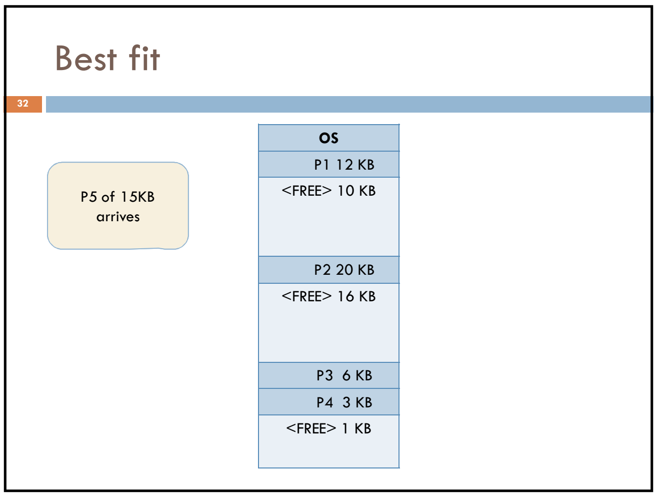
Initial memory mapping



Best fit

P4 of 3KB loaded here by BEST FIT

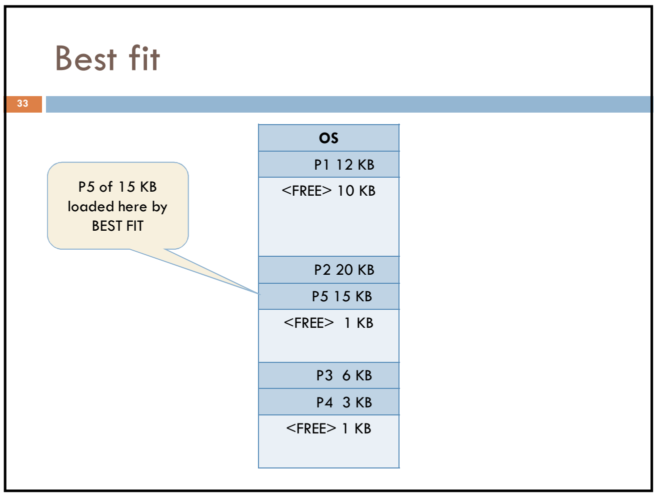
OS
P1 12 KB
<FREE> 10 KB
P2 20 KB
<FREE> 16 KB
P3 6 KB
P4 3 KB
<FREE> 1 KB



Best fit

P5 of 15KB arrives

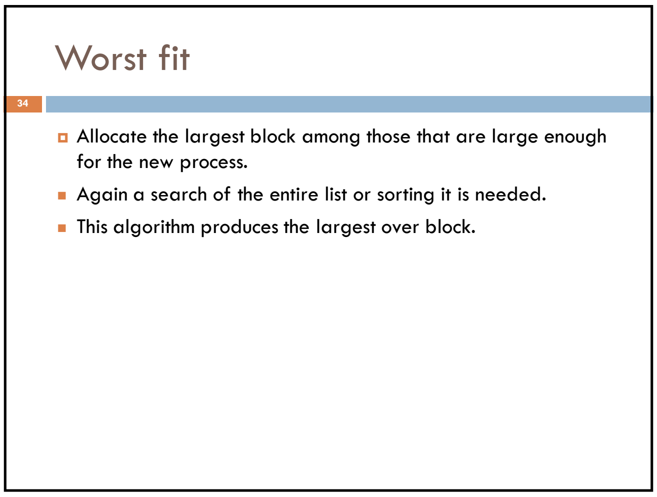
OS
P1 12 KB
<FREE> 10 KB
P2 20 KB
<FREE> 16 KB
P3 6 KB
P4 3 KB
<FREE> 1 KB



Best fit

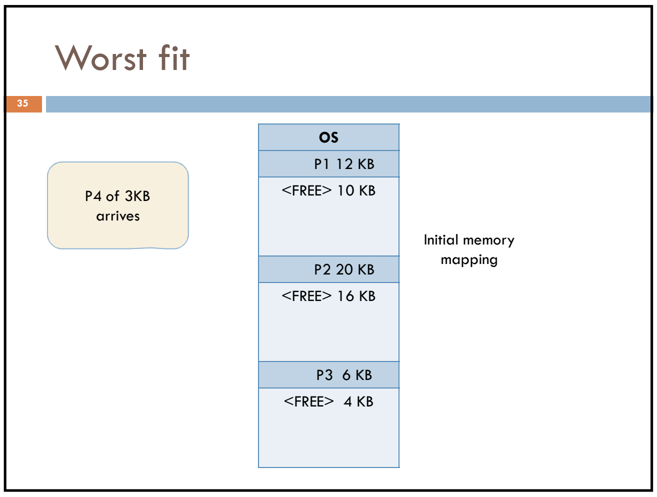
P5 of 15 KB loaded here by BEST FIT

OS
P1 12 KB
<FREE> 10 KB
P2 20 KB
P5 15 KB
<FREE> 1 KB
P3 6 KB
P4 3 KB
<FREE> 1 KB



Worst fit

- Allocate the largest block among those that are large enough for the new process.
  - Again a search of the entire list or sorting it is needed.
  - This algorithm produces the largest over block.



Worst fit

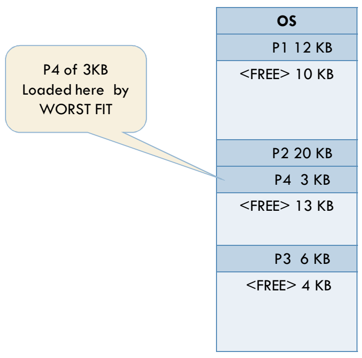
P4 of 3KB arrives

OS
P1 12 KB
<FREE> 10 KB
P2 20 KB
<FREE> 16 KB
P3 6 KB
<FREE> 4 KB

Initial memory mapping

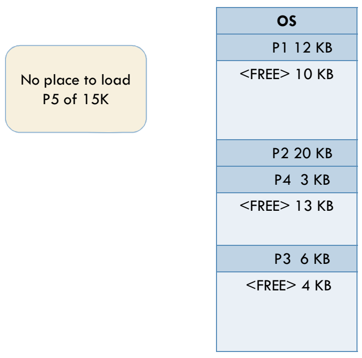
Worst fit

36



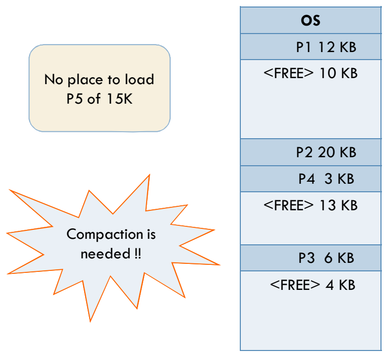
Worst fit

37



Worst fit

38



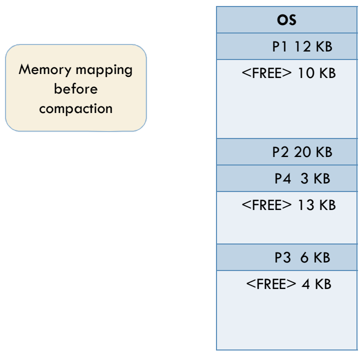
Compaction

39

- Compaction is a method to overcome the external fragmentation problem.
- All free blocks are brought together as one large block of free space.
- Compaction requires dynamic relocation.
- Certainly, compaction has a cost and selection of an optimal compaction strategy is difficult.
- One method for compaction is swapping out those processes that are to be moved within the memory, and swapping them into different memory locations

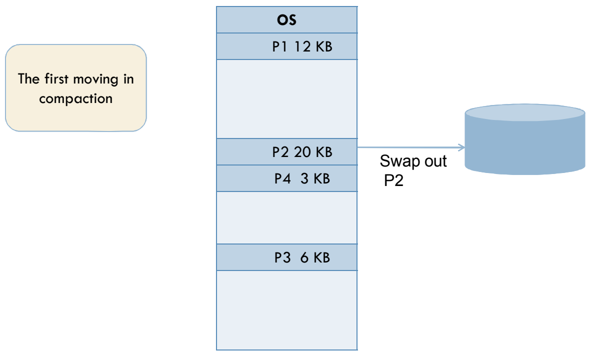
Compaction

40

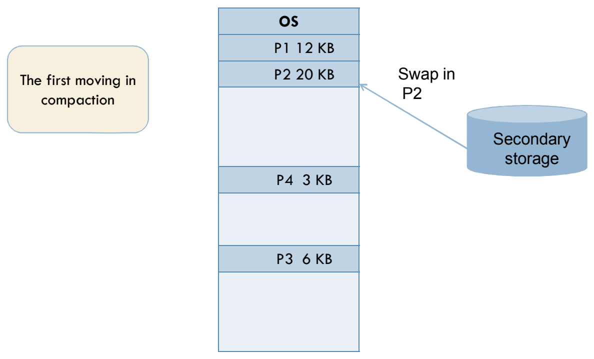


Compaction

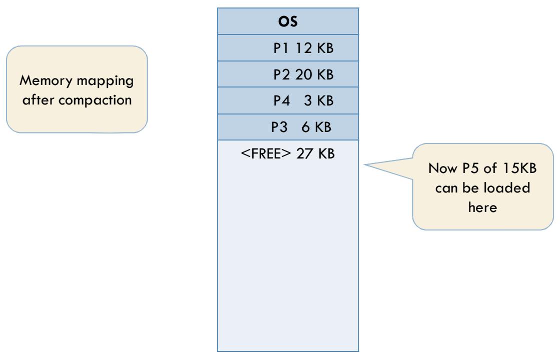
41



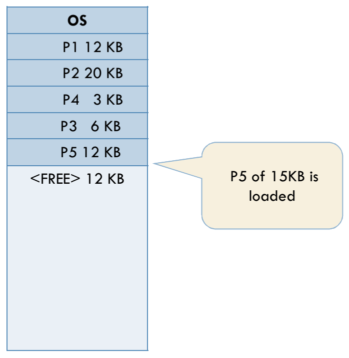
Compaction



Compaction


Memory mapping after compaction
OS
P1 12 KB
P2 20 KB
P4 3 KB
P3 6 KB
<FREE> 27 KB
Now P5 of 15KB can be loaded here

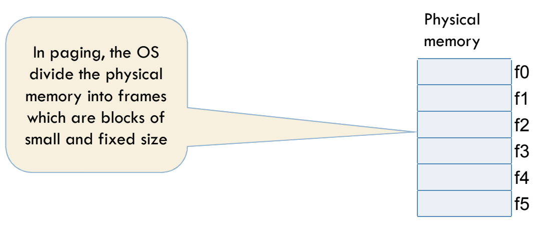
Compaction



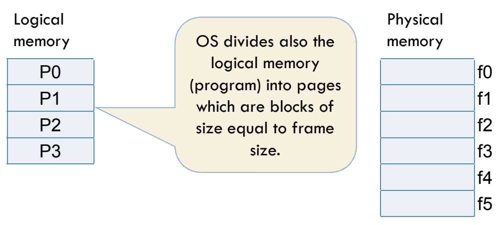
8.3 Paging

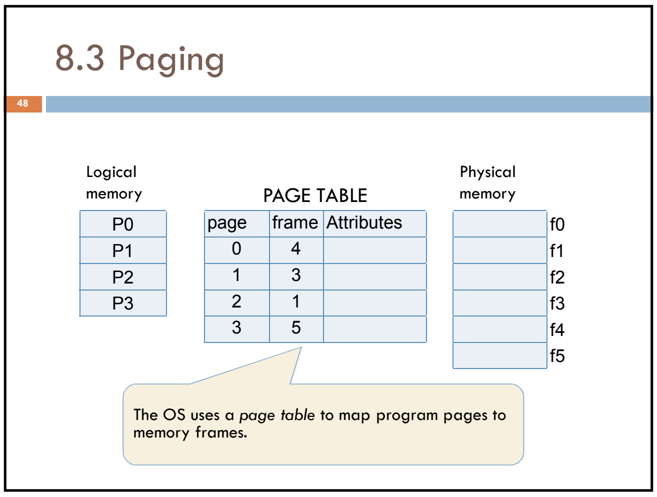
- Partition methods leads fragmentation problems.
- For dealing with them, logical memory (process) is divided into pieces of same size which are called pages,
- Physical memory is also divided into pieces called frames having the size of page.
- Then every page can be assigned into each frame.
- There will be only internal fragmentation especially in the last page of the process.

8.3 Paging


In paging, the OS divide the physical memory into frames which are blocks of small and fixed size
Physical memory
f0
f1
f2
f3
f4
f5

8.3 Paging


Logical memory
P0
P1
P2
P3
OS divides also the logical memory (program) into pages which are blocks of size equal to frame size.
Physical memory
f0
f1
f2
f3
f4
f5



8.3 Paging

48

Logical memory

P0
P1
P2
P3

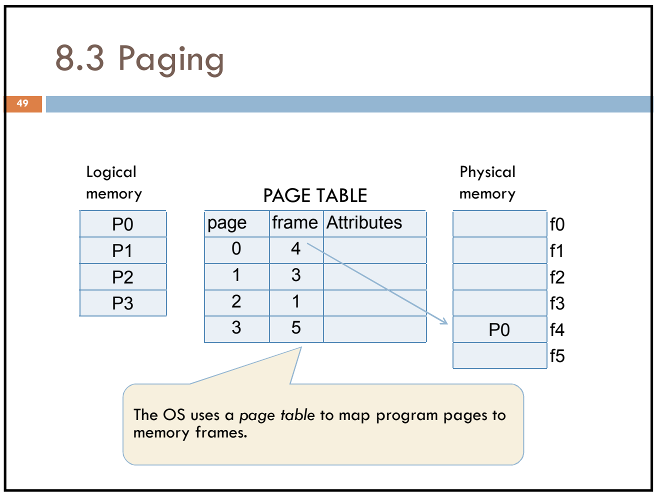
PAGE TABLE

page	frame	Attributes
0	4	
1	3	
2	1	
3	5	

Physical memory

	f0
	f1
	f2
	f3
	f4
	f5

The OS uses a *page table* to map program pages to memory frames.



8.3 Paging

49

Logical memory

P0
P1
P2
P3

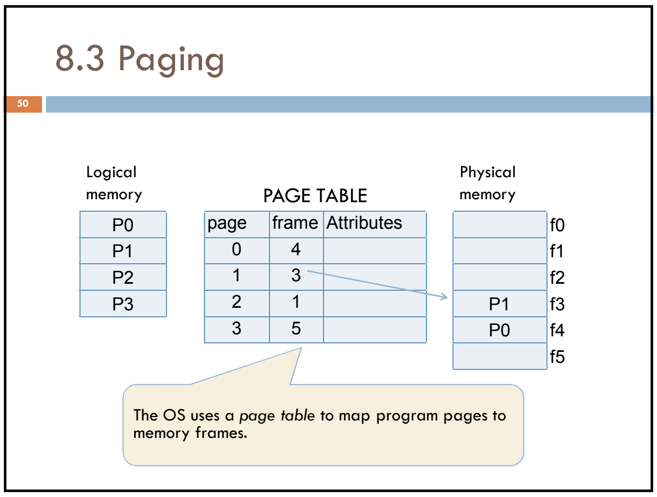
PAGE TABLE

page	frame	Attributes
0	4	
1	3	
2	1	
3	5	

Physical memory

	f0
	f1
	f2
	f3
P0	f4
	f5

The OS uses a *page table* to map program pages to memory frames.



8.3 Paging

50

Logical memory

P0
P1
P2
P3

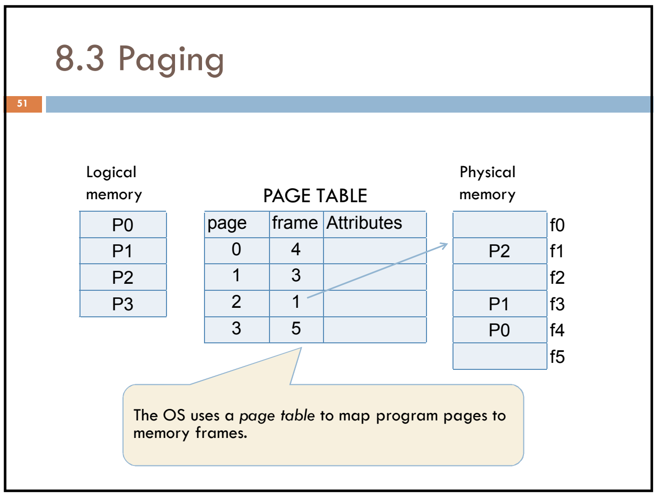
PAGE TABLE

page	frame	Attributes
0	4	
1	3	
2	1	
3	5	

Physical memory

	f0
	f1
	f2
P1	f3
P0	f4
	f5

The OS uses a *page table* to map program pages to memory frames.



8.3 Paging

51

Logical memory

P0
P1
P2
P3

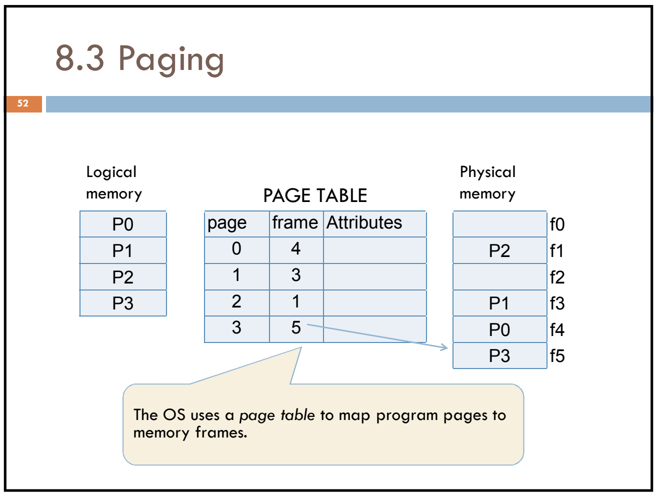
PAGE TABLE

page	frame	Attributes
0	4	
1	3	
2	1	
3	5	

Physical memory

	f0
P2	f1
	f2
P1	f3
P0	f4
	f5

The OS uses a *page table* to map program pages to memory frames.



8.3 Paging

52

Logical memory

P0
P1
P2
P3

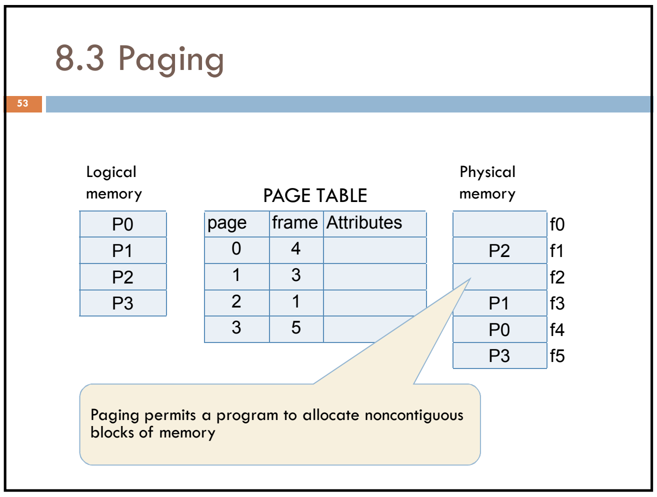
PAGE TABLE

page	frame	Attributes
0	4	
1	3	
2	1	
3	5	

Physical memory

	f0
P2	f1
	f2
P1	f3
P0	f4
P3	f5

The OS uses a *page table* to map program pages to memory frames.



8.3 Paging

53

Logical memory

P0
P1
P2
P3

PAGE TABLE

page	frame	Attributes
0	4	
1	3	
2	1	
3	5	

Physical memory

	f0
P2	f1
	f2
P1	f3
P0	f4
P3	f5

Paging permits a program to allocate noncontiguous blocks of memory

8.3 Paging

- Page size (S) is defined by the hardware.
  - Generally page size is chosen as a power of 2 such as 512 words/page or 4096 words/page etc.
- With this arrangement, the words in the program have an address called as *logical address*. Every logical address is formed of <p,d> pair

* To handle data most efficiently, all processors have a characteristic data size known as the word size (words). It is usually a power of 2 bytes

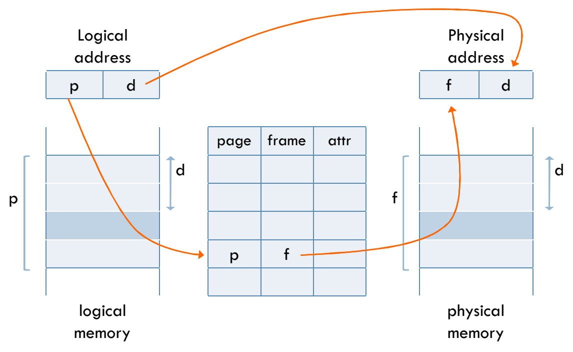
8.3 Paging

- Logical address: <p, d>
  - p *is page number*
    p = logical address div S
  - d is displacement (offset)
    d = logical address mod S

8.3 Paging

- When a logical address <p, d> is generated by the processor,
- At first, the frame number f corresponding to page p is determined by using the page table
- And then the physical address is calculated as (f*S+d) and the memory is accessed.

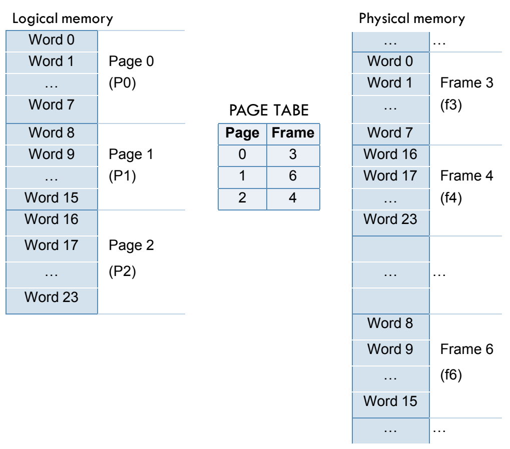
8.3 Paging



8.3 Paging

Example

- Consider the following information to form a physical memory map.
- Page Size = 8 words ➔ d : 3 bits
- Physical Memory Size = 128 words
  = 128/8=16 frames ➔ f : 4 bits
- Assume maximum program size is 4 pages ➔ p : 2 bits
- A program of 3 pages where P0 → f3; P1 → f6; P2 → f4



PAGE TABE

Page	Frame
0	3
1	6
2	4

8.3 Paging

60

Program Line	Logical Address	Offset
Word 0	00 000	000
Word 1	00 001	001
…	…	…
Word 7	00 111	111
Word 8	01 000	000
Word 9	01 001	001
…	…	…
Word 15	01 111	111
Word 16	10 000	000
Word 17	10 001	001
…	…	…
Word 23	10 111	111

8.3 Paging

61

Program Line	Logical Address	Offset	Page Number
Word 0	00 000	000	00
Word 1	00 001	001	00
…	…	…	…
Word 7	00 111	111	00
Word 8	01 000	000	01
Word 9	01 001	001	01
…	…	…	…
Word 15	01 111	111	01
Word 16	10 000	000	10
Word 17	10 001	001	10
…	…	…	…
Word 23	10 111	111	10

8.3 Paging

62

Program Line	Logical Address	Offset	Page Number	Frame Number
Word 0	00 000	000	00	0011
Word 1	00 001	001	00	0011
…	…	…	…	…
Word 7	00 111	111	00	0011
Word 8	01 000	000	01	0110
Word 9	01 001	001	01	0110
…	…	…	…	…
Word 15	01 111	111	01	0110
Word 16	10 000	000	10	0100
Word 17	10 001	001	10	0100
…	…	…	…	…
Word 23	10 111	111	10	0100

8.3 Paging

63

Program Line	Logical Address	Offset	Page Number	Frame Number	Physical Address
Word 0	00 000	000	00	0011	0011 000
Word 1	00 001	001	00	0011	0011 001
…	…	…	…	…	…
Word 7	00 111	111	00	0011	0011 111
Word 8	01 000	000	01	0110	0110 000
Word 9	01 001	001	01	0110	0110 001
…	…	…	…	…	…
Word 15	01 111	111	01	0110	0110 111
Word 16	10 000	000	10	0100	0100 000
Word 17	10 001	001	10	0100	0100 001
…	…	…	…	…	…
Word 23	10 111	111	10	0100	0100 111

8.3 Paging

64

- Every access to memory should go through the page table. Therefore, it must be implemented in an efficient way.
- The efficient ways to implement the page table
  - Using registers
  - Using main memory
  - Using associative registers

Using registers

65

- Keep page table in fast registers. Only the OS is able to modify these registers.
- However, if the page table is large, this method becomes very expensive since requires too many registers.

Using main memory

66

- In this second method, the OS keeps a page table in the memory, instead of registers.
- For every logical memory reference, two memory accesses are required:
    1. To access the page table in the memory, in order to find the corresponding frame number.
    2. To access the memory word in that frame
- This is cheap but a time consuming method.

Using associative registers

67

- Associative registers (cache) contains most recently used page table entries
- Since all registers run in parallel, searching is fast
- Associative registers are quite expensive. So, a small number of them should be used
- If page table entry is present (called hit), found frame number is formed into real address
- Otherwise, the page number is used to index the process page table in main memory

Using associative registers

68

Example: assume we have a paging system which uses associative registers. These associative registers have an access time (rat) of 30 ns, and the memory access time (mat) is 470 ns. On the other hand, the system has a hit ratio (h) of 90%.

- rat=30 ns
- mat=470ns
- h=0.9

Using associative registers

69

rat=30 ns, mat=470ns, h=0.9

- Now, if the page number is found in one of the associative registers, then the effective memory access time:
- $emat_{HIT} = 30 + 470 = 500$ ns.
- Because one access to associative registers and one access to the main memory is sufficient.

Using associative registers

70

rat=30 ns, mat=470ns, h=0.9

- On the other hand, if the page number is not found in associative registers, then the effective memory access time:
- $emat_{MISS} = 30 + (470+470) = 970$ ns.
- Since one access to associative registers and two accesses to the main memory are required.

Using associative registers

71

rat=30 ns, mat=470ns, h=0.9

$emat_{HIT} = 500$ ns, $emat_{MISS} = 970$ ns.

- Then, the weighted emat is calculated as follows:

$emat = h * emat_{HIT} + (1-h) * emat_{MISS}$

$= 0.9 * 500 + 0.1 * 970$

$= 450 + 97 = 547$ ns

Sharing Pages

- Sharing is an important advantage of paging.
- It is possible to share system procedures or programs, user procedures or programs, and possibly data area.
- Sharing pages is advantageous especially in time-sharing systems.

Sharing Pages

Example: Consider a system having page size=30 MB. There are 3 users executing an editor program which is 90 MB (3 pages) in size, with a 30 MB (1 page) data space.

- To support these 3 users, the OS must allocate
  3 * (90+30) = 360 MB space

- However, if the editor program is shared as read only, then all users can use it, and only one copy of the editor program is sufficient. Therefore, the OS must allocate only
  90 + 30 * 3 = 180 MB space

User-1

P0	e1
P1	e2
P2	e3
P3	data1

PT-1

Page#	Frame#
0	8
1	4
2	5
3	7

User-2

P0	e1
P1	e2
P2	e3
P3	data2

PT-2

Page#	Frame#
0	8
1	4
2	5
3	12

User-3

P0	e1
P1	e2
P2	e3
P3	data3

PT-3

Page#	Frame#
0	8
1	4
2	5
3	10

Physical Memory

f0	OS
f1	OS
f2	OS
f3	
f4	e2
f5	e3
f6	
f7	data1
f8	e1
f9	
f10	data3
f11	
f12	data 2
f13	
f14	
f15	

User-1

P0	e1
P1	e2
P2	e3
P3	data1

PT-1

Page#	Frame#
0	8
1	4
2	5
3	7

When User 2 terminates, Data2 page is removed from memory, but editor pages remain..

User-3

P0	e1
P1	e2
P2	e3
P3	data3

PT-3

Page#	Frame#
0	8
1	4
2	5
3	10

Physical Memory

f0	OS
f1	OS
f2	OS
f3	
f4	e2
f5	e3
f6	
f7	data1
f8	e1
f9	
f10	data3
f11	
f12	
f13	
f14	
f15	

When User 1 terminates, data1 is also removed from memory.

User-3

P0	e1
P1	e2
P2	e3
P3	data3

PT-3

Page#	Frame#
0	8
1	4
2	5
3	10

Physical Memory

f0	OS
f1	OS
f2	OS
f3	
f4	e2
f5	e3
f6	
f7	
f8	e1
f9	
f10	data3
f11	
f12	
f13	
f14	
f15	

When User 3 terminates, Data-3 and also editor segments are removed from memory.

Physical Memory

f0	OS
f1	OS
f2	OS
f3	
f4	
f5	
f6	
f7	
f8	
f9	
f10	
f11	
f12	
f13	
f14	
f15	

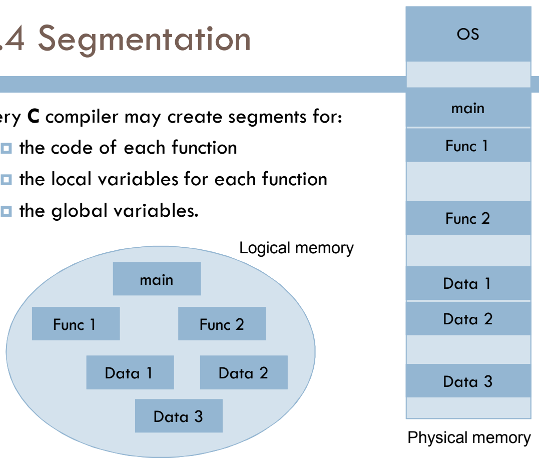
8.4 Segmentation

- By means of segmentation, programs can be divided into variable sized segments, as in variable partitioning.
- But programs are divided into small parts, as in paging.
- Every logical address is transformed into a segment value and an offset value.
- Programs are segmented automatically when they are compiled.

8.4 Segmentation

Every **C** compiler may create segments for:

- the code of each function
- the local variables for each function
- the global variables.

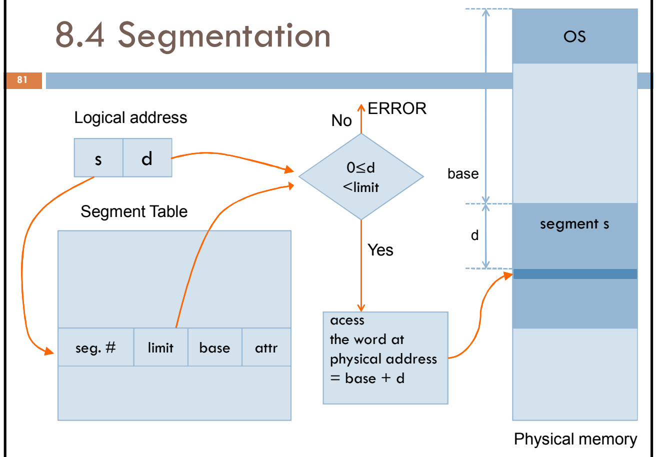


8.4 Segmentation

For transforming address, a table is used. When a logical address <s, d> is generated:

1. Base and limit values corresponding to segment s are determined using this segment table
2. The OS checks whether d is in the limit
   - $0 \leq d < \text{limit}$
3. If so, then the physical address is calculated as
   - base + d

8.4 Segmentation



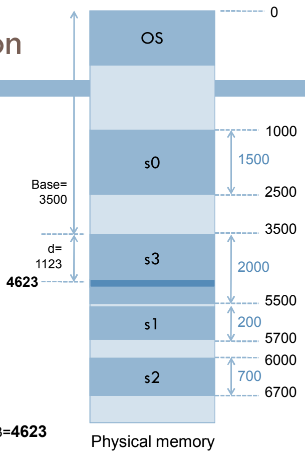
8.4 Segmentation

Example: By generating the memory map according to the given segment table, find the corresponding physical address for logical address of <3,1123>.

Segment	Limit	Base
0	1500	1000
1	200	5500
2	700	6000
3	2000	3500

8.4 Segmentation

Segment	Limit	Base
0	1500	1000
1	200	5500
2	700	6000
3	2000	3500



Logical address: <3,1123>
s=3, d=1123

Check if d<limit
1123<2000 ….. OK

Physical address (base+d) = 3500+1123=**4623**

8.4 Segmentation

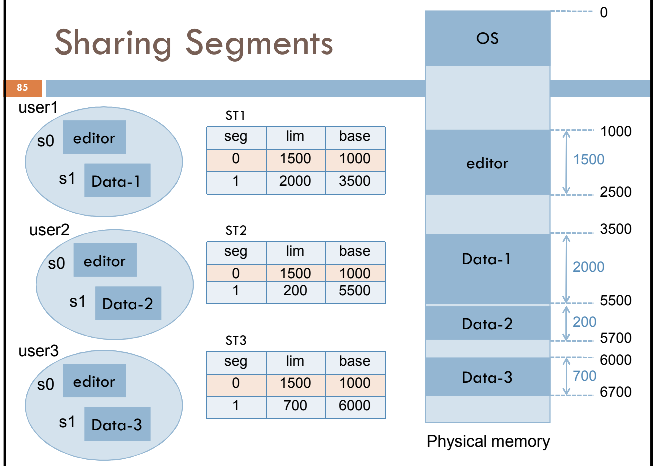
How can we implement the segment table efficiently?

- Segment tables may be implemented in the main memory or in associative registers, in the same way it is done for page tables.

What about the sharing?

- Also sharing of segments is applicable as in paging. Shared segments should be read only and should be assigned the same segment number.

Sharing Segments



ST1

seg	lim	base
0	1500	1000
1	2000	3500

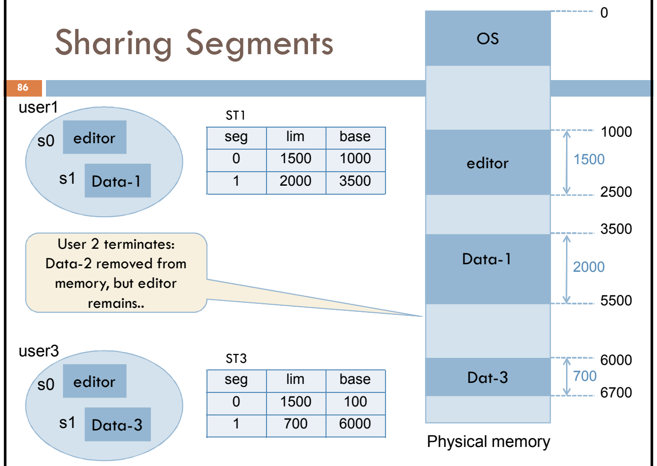
ST2

seg	lim	base
0	1500	1000
1	200	5500

ST3

seg	lim	base
0	1500	1000
1	700	6000

Sharing Segments



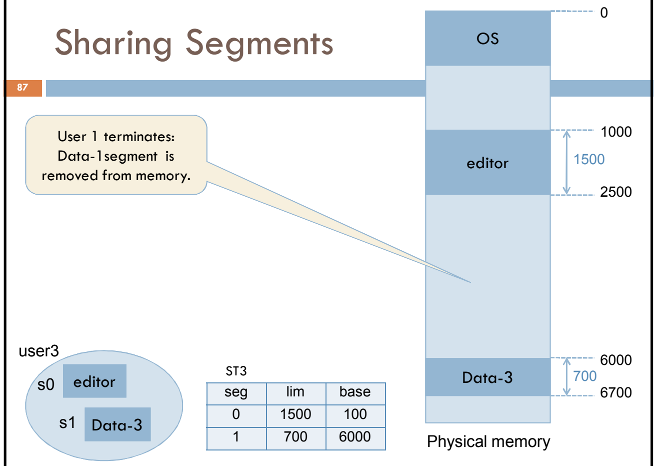
ST1

seg	lim	base
0	1500	1000
1	2000	3500

ST3

seg	lim	base
0	1500	100
1	700	6000

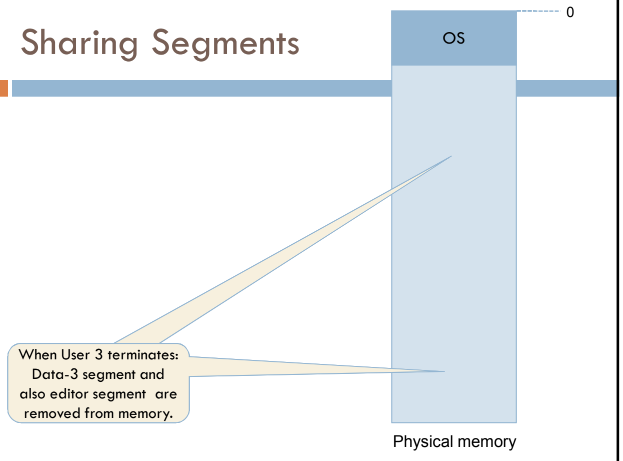
Sharing Segments



ST3

seg	lim	base
0	1500	100
1	700	6000

Sharing Segments


OS
0
When User 3 terminates: Data-3 segment and also editor segment are removed from memory.
Physical memory