Operating system

Part VIII: Memory (basic)

By KONG LingBo (孔令波)

- Problems:
  - VM (Virtual Memory) is usually consid ered as the modern way to overcome the memory limitation for **running a program whose size is larger than t he physical memory – "Large progr am but small memory"**
  - However, overlay, dynamic linking etc . can also be used for that
  - What's exactly the difference?

Declaration

- The content of Silbersatz' s book leads to a confusion
  - It splits Paging, Segmenting techniques from the concept of virtual memory.
  - In fact, they are part of the techniques used to support VM concept!
    - http://en.wikipedia.org/wiki/Paging
      - "Paging is an important part of virtual memory implementation in most contemporary general-purpose operating systems, allowing them to use disk storage for data that does not fit into physical random-access memory (RAM)."

More **reasonable** organization

- MM with two parts
  - **Basic concepts and techniques**
    - Techniques supporting the mapping from lo gical address to physical address
    - Old ways to manage main memory for execu ting programs
      - Partitioning, Overlays/Swapping
  - **Virtual Memory (VM)**
    - (On-demand) Paging scheme, Segmenting s cheme, Segment-Page scheme

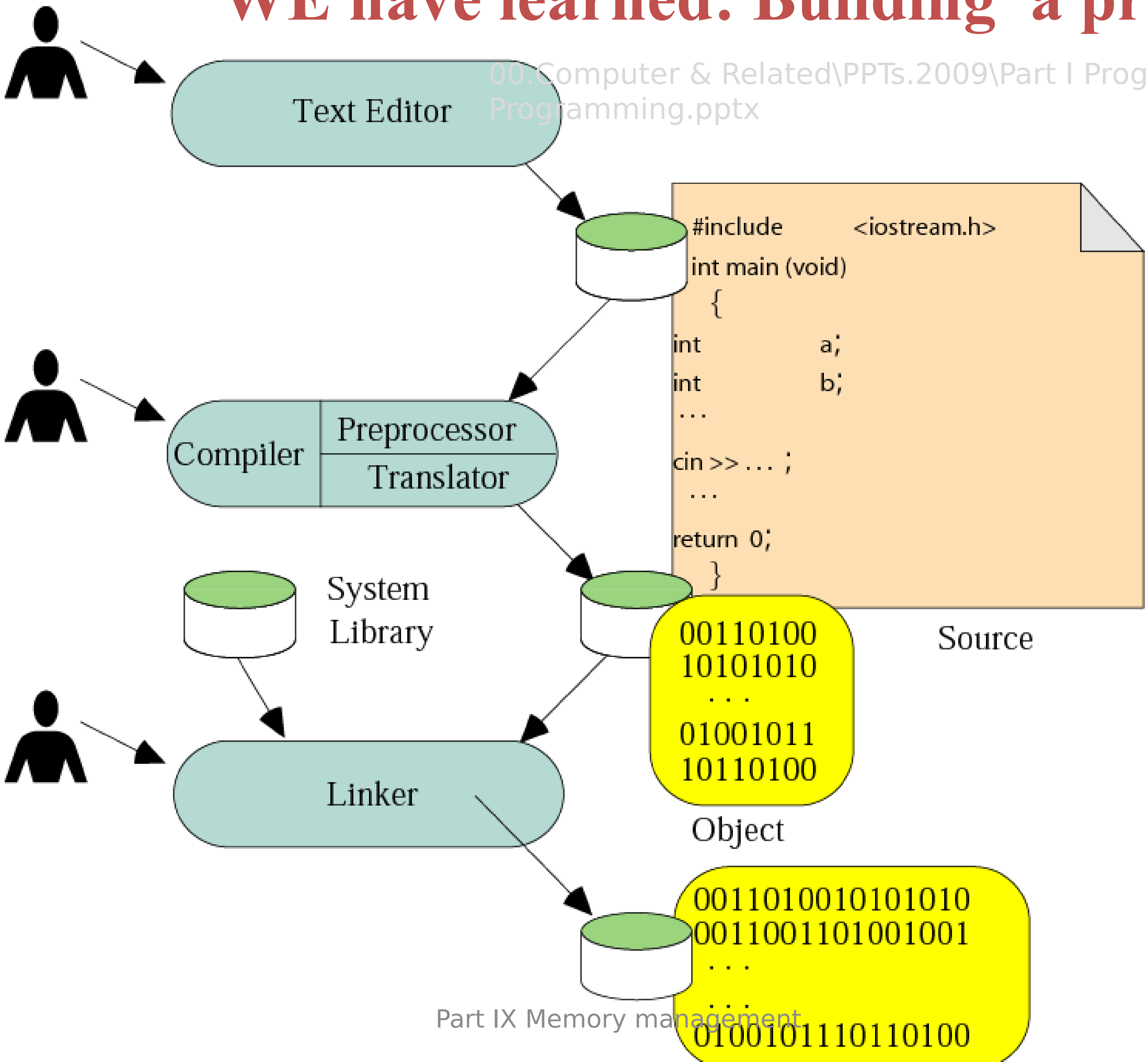
Goals

- Know the basic concepts of MM
  - Hierarchy of storage medias
  - Static/Dynamic Linking, Relocation, Protection
- Overlay
- Partitioning schemes
  - Fixed/Variable partition
  - Replacement algorithms

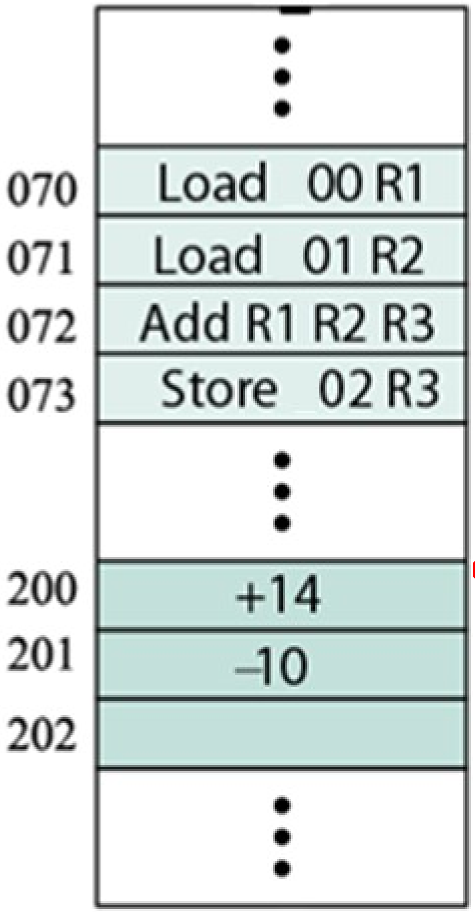
Me
mor
y

- Basic concepts
  - From Logic address to physical address
  - MMU for relocation – address translation
- Basic techniques of real memory management
  - Overlay
  - Dynamic linking
  - Partitioning (Static & Dynamic)
- For OS space
  - Knuth’s Buddy System

WE have learned: Building a program



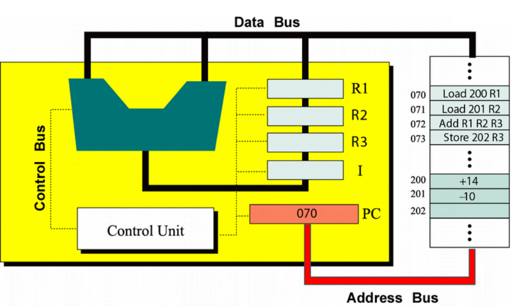
BUT, How are the instructions transferred into the main memory?



Program
X=14;
Y= -10;
Z= X+Y;

Assembly
Load **00** R1;
Load **01** R2;
ADD R1 R2 R3
Store **02** R3;

Machine code
[00] 14 (should be bi
[01] -10
[02] (used later)
[03] 0001 **0000000**
[04] 0010 **00000001**
[05] 0011
[06] 0100 **0000001**



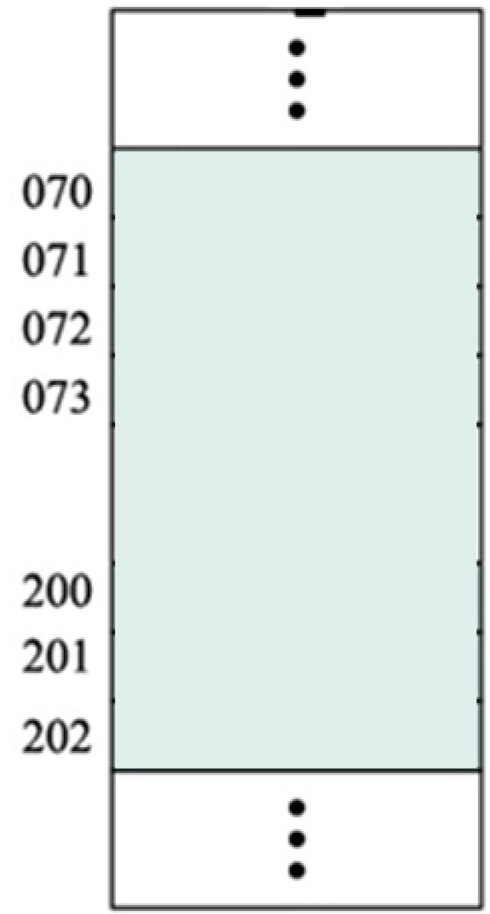
- Logical address [逻辑地址] ☾ **Program space** ☾ **File Space**
  - An address of one instruction or data item inside program space is commonly referred to as a logical address
- Physical address [物理地址] ☾ Hard Disk Space ☾ **MM Space**
  - whereas an address of one instruction or data item in the memory unit commonly referred to as a physical address, which could be really accessed by CPU.
    - To my thinking, addresses in HDD should also be understood as Physical Address

2 tasks for MM – also for HDD

- With the mapping slide, we can infer there are 2 tasks
  - **Memory allocation**
    - Given the segment of a progra m, assign a block/region in MM to keep that segment – namely copy the numbered instruction s together with the necessary d ata to that block/region
  - **Address translation**
    - Even the instructions are copie d to the memory block, some l ogic addresses are also kept wi th some instructions

Machine code

[00] 14 (should be bin)
[01] -10
[02] (used later)
[03] 0001 **00000000**
[04] 0010 **00000001**
[05] 0011
[06] 0100 **00000010**



- some logic addresses are still conveyed into the MM as the parameters of some instructions

Program

X=14;
Y= -10;
Z= X+Y;

Machine code

[00] 14 (should be bin)
[01] -10
[02] (used later)
[03] 0001 **00000000**
[04] 0010 **00000001**
[05] 0011
[06] 0100 **00000010**

Machine code in MM

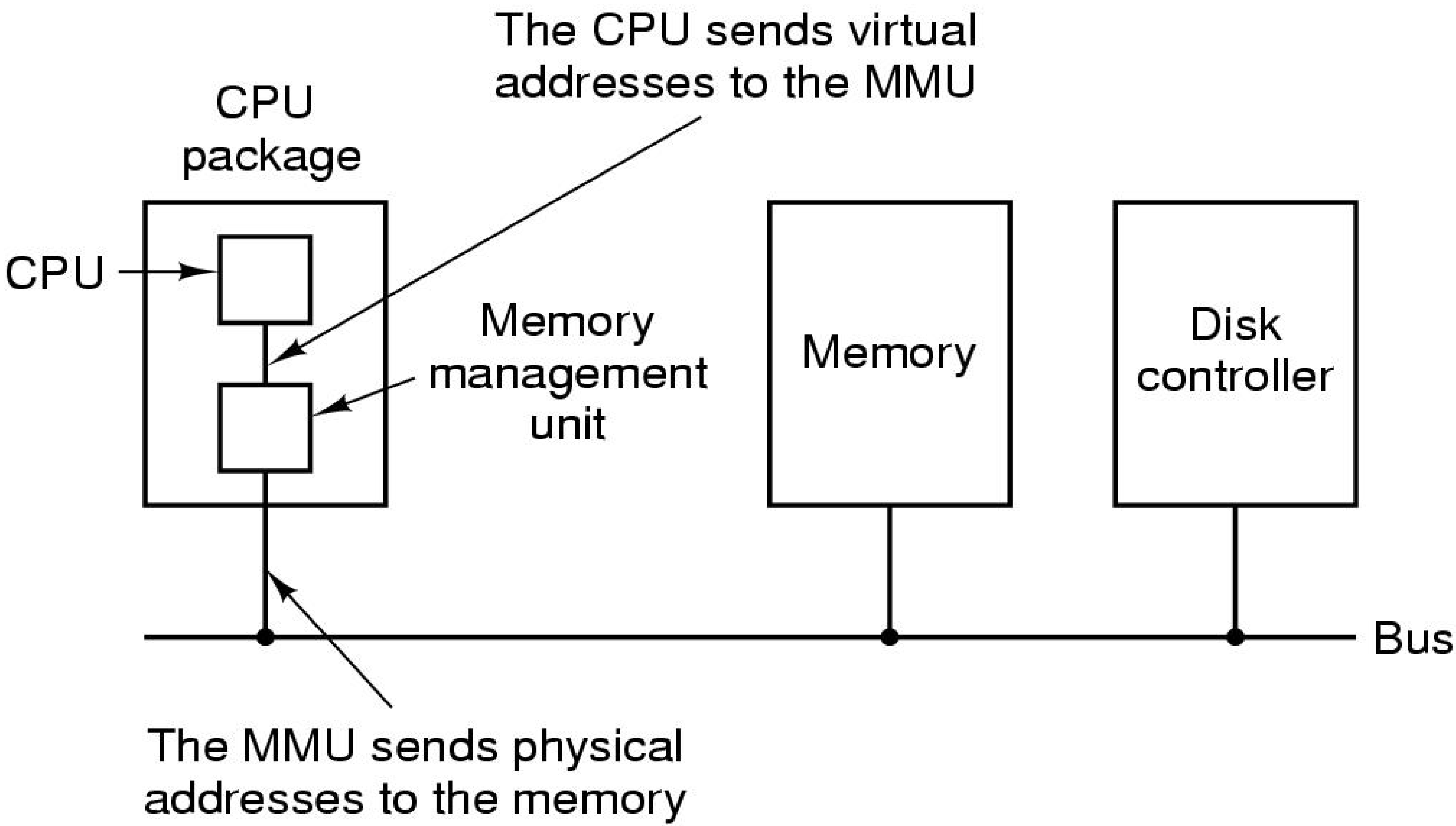
[200] 14
[201] -10
[202]

[070] 0001
00000000
[071] 0010
00000001

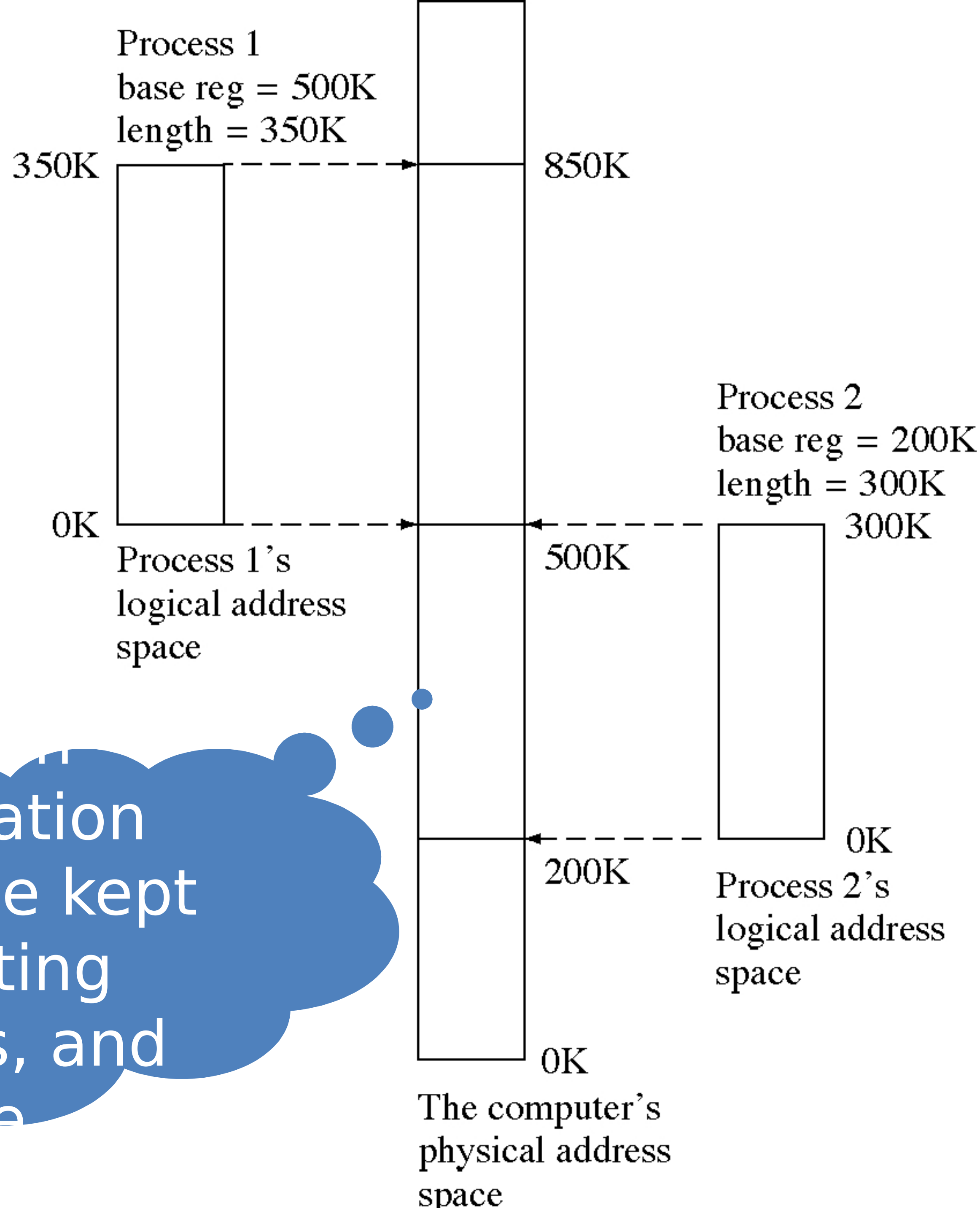
Relocation [重定位] – Address translation

- The concept to compute the physical addresses of the logic addresses like above
  - **Problem**: given the start position and the size of the block which is assigned to the segment, you are asked to compute the physical address of the memory unit to store the corresponding logic address
  - Called relocation!
- **MMU** – Memory Management Unit
  - The run-time mapping from logical to physical addresses is done by a hardware device called the memory-management unit.

CPU, **MMU** and Memory



Logical and Physica
ddress Spaces



information
should be kept
– starting
address, and

MMU – starting address + size as parameters

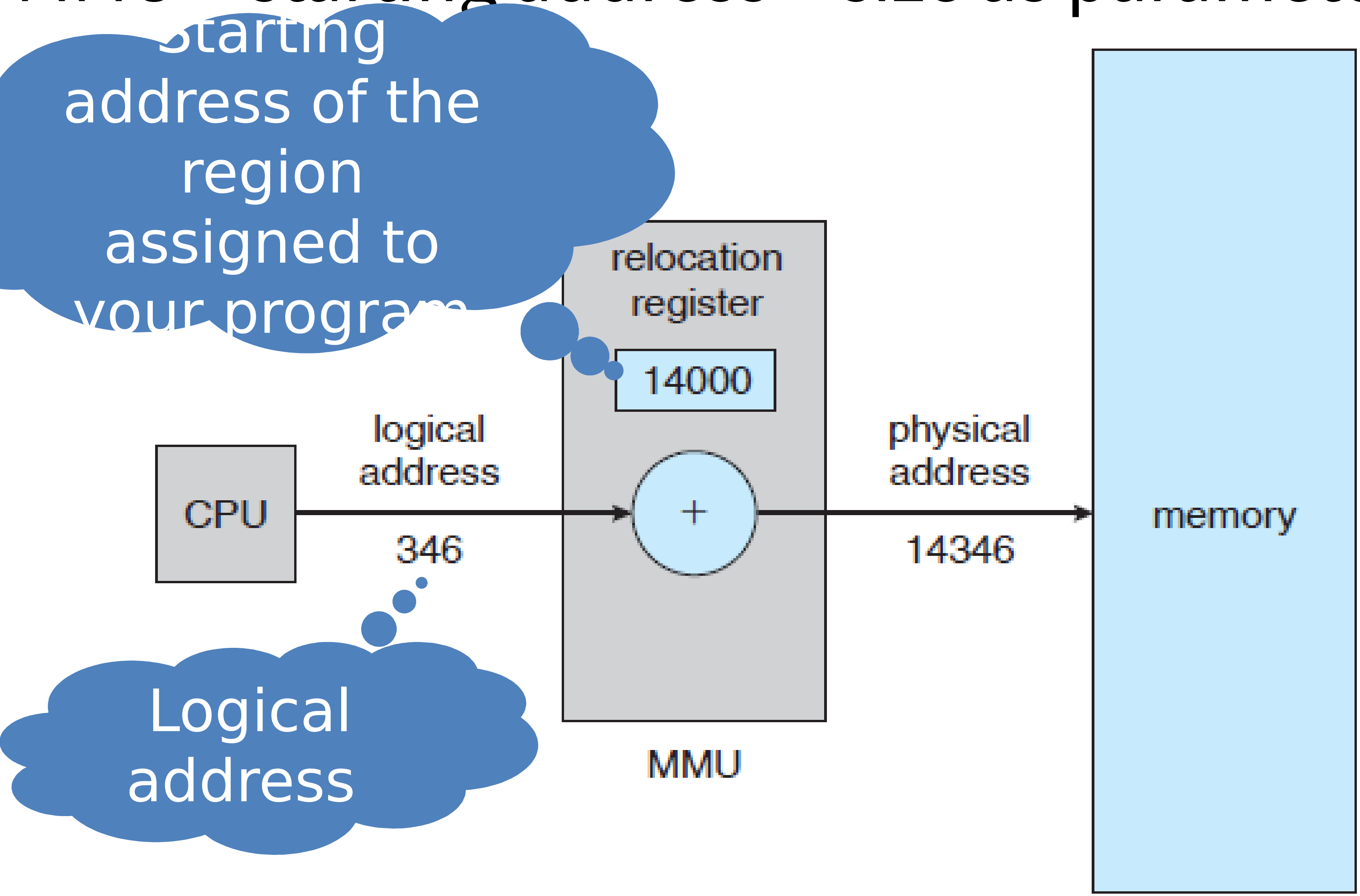


Figure 7.4 Dynamic relocation using a relocation register.

Memory Management Requirements (2)

- **Protection**:
  - Processes should not be able to reference memory locations in another process without permission.
  - Impossible to check addresses in programs at compile/load-time since the program could be relocated.
  - Address references must be checked at execution-time by hardware.

PPTs from others\From Ariel J. Frank\OS381\os7-1_rea.ppt

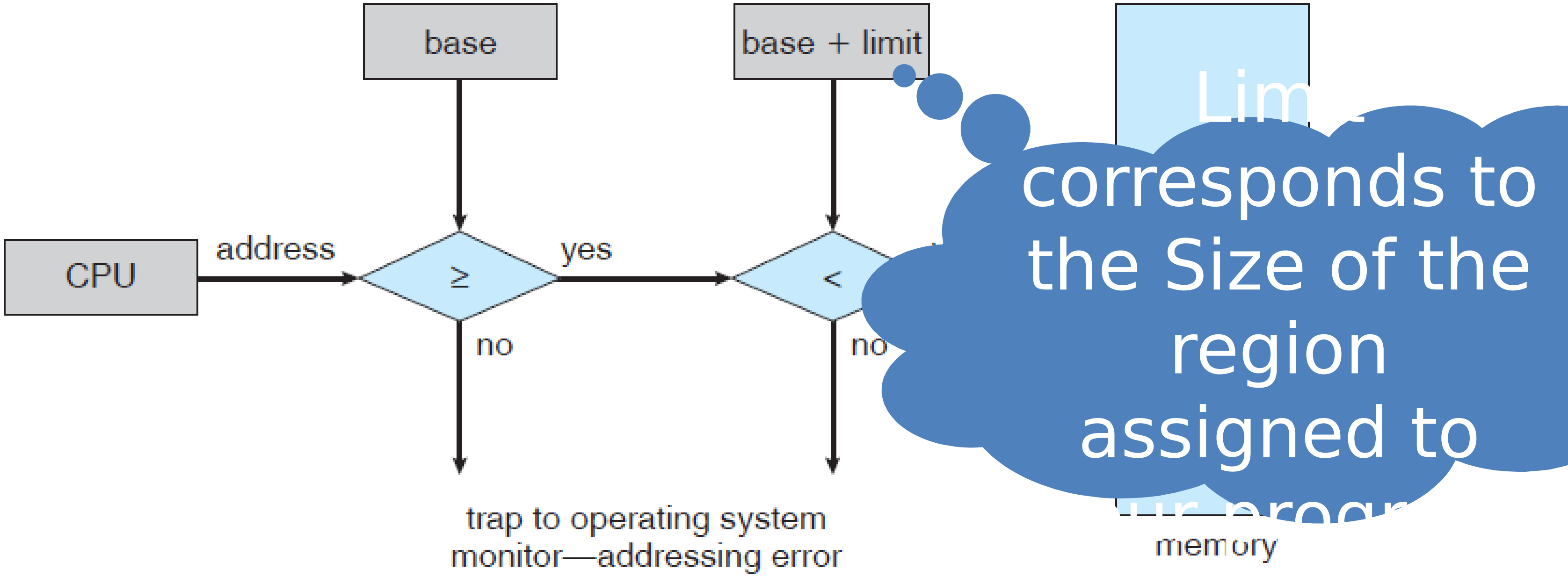


Figure 7.2 Hardware address protection with base and limit registers.

- Protection of memory space is accomplished by having the CPU hardware compare every address generated in user mode with the registers.
- Any attempt by a program executing in user mode to ac cess operating-system memory or other users’ memor y results in a trap to the operating system, which treats t he attempt as a fatal error

Me
mor
y

- Basic concepts
  - Memory in storage hierarchy
  - From program to process
    - Static & Dynamic linking
  - Requirements of MM
- Basic techniques of real memory mana gement
  - Partitioning (Static & Dynamic)
  - Other auxiliary skills – Overlay and DDL
- For OS space
  - Knuth' s Buddy System

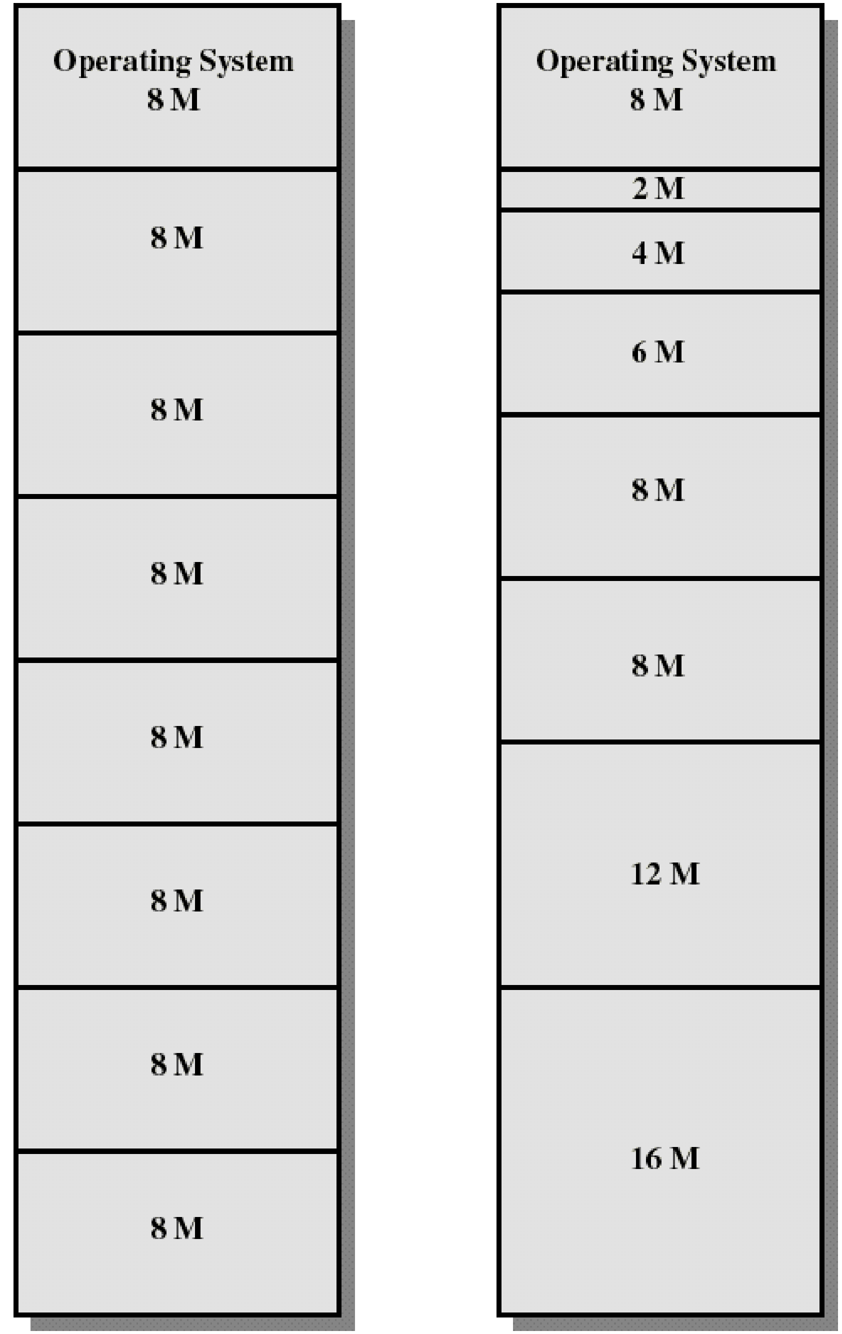
Real Memory Management Techniques

- Although the following simple/basic memory management techniques are not used in modern OSs, they lay the ground for a later proper discussion of **virtual memory**:
  - Fixed/Static Partitioning
  - Variable/Dynamic Partitioning

- In fact, I think that the basis of modern MM is partition and swapping

Fixed Partitioning

- Partition main memory into a set of non-overlapping memory regions called **partitions**.
- Fixed partitions can be of **equal** or **unequal** sizes.
- Leftover [剩余的；边角料的] space in partition, after program assignment, is called **internal fragmentation [内部碎片；内零头]**.



Equal-size partitions

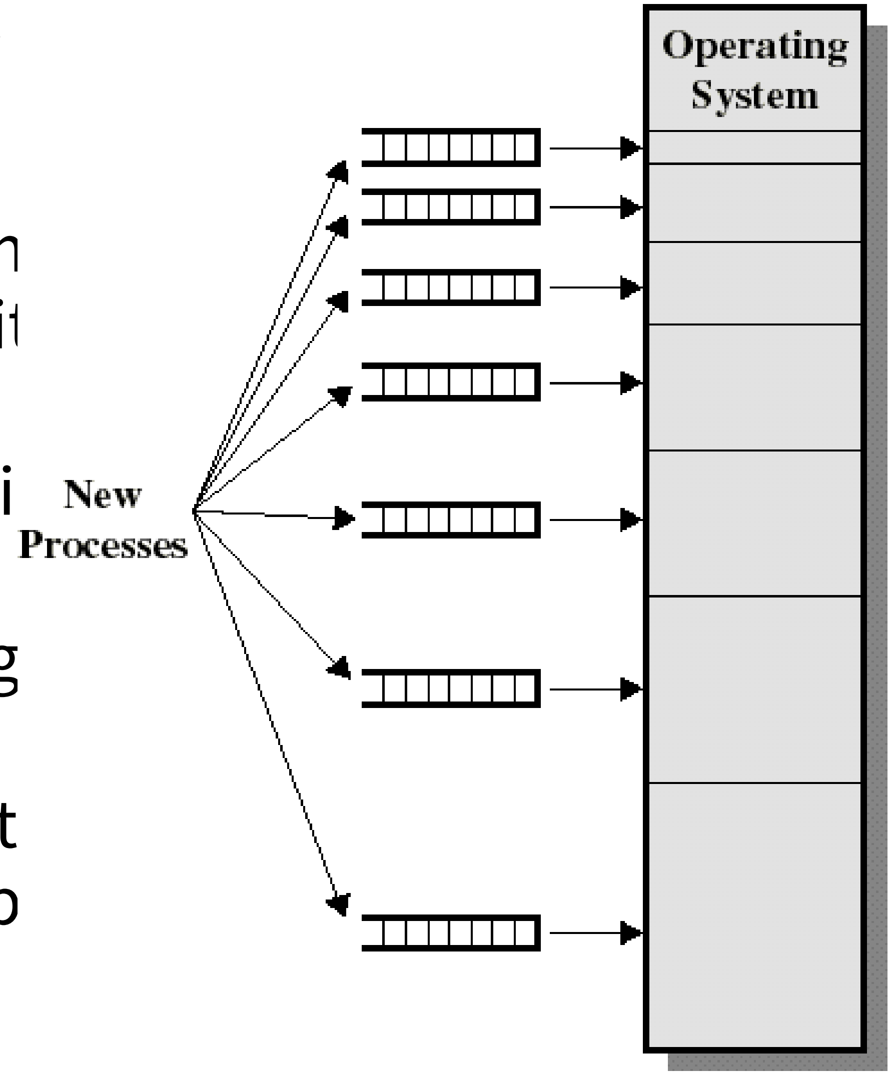
Unequal-size partitions

Placement Algorithm with Partitions

- Equal-size partitions:
  1. If there is an available partition, a process can be loaded into that partition –
     - because all partitions are of equal size, it does not matter which partition is used.
  2. If all partitions are occupied by blocked processes, choose one process to swap out to make room for the new process.

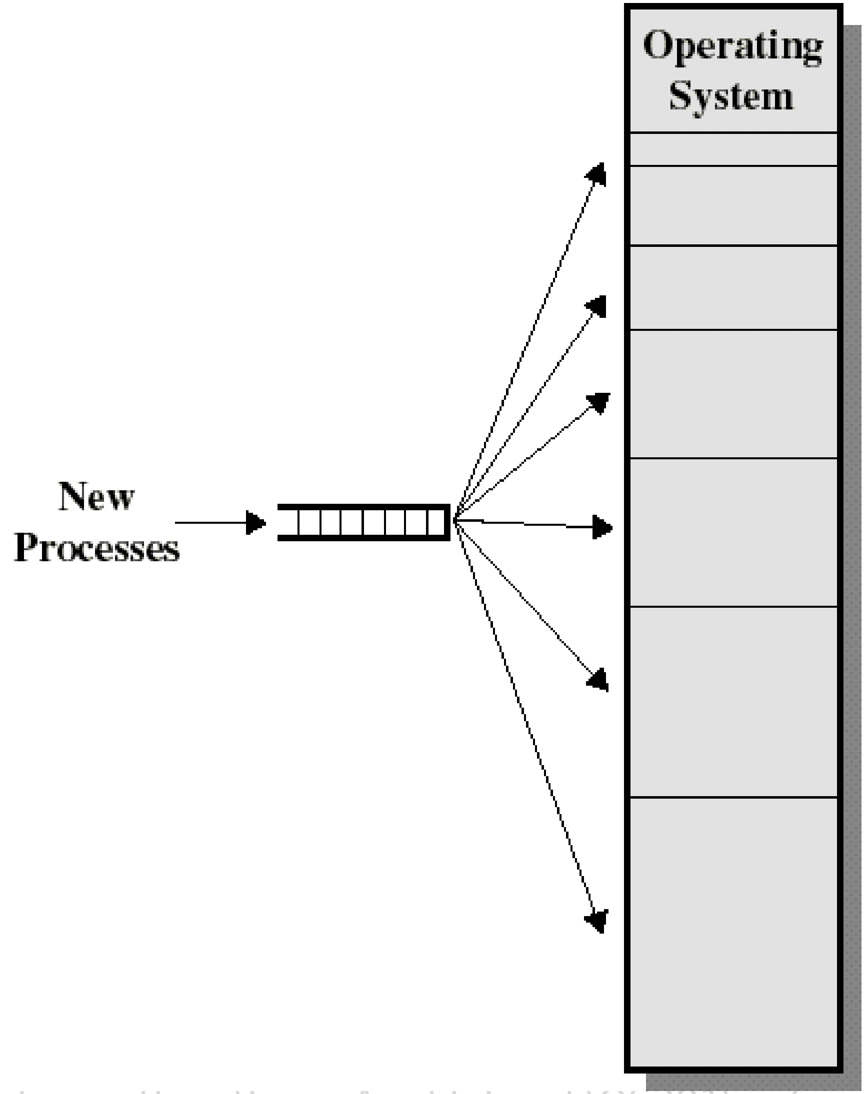
Placement Algorithm with Partitions

- Unequal-size partitions, using multiple queues:
  - assign each process to the smallest partition within which it will fit.
  - a queue exists for each partition size.
  - tries to minimize internal fragmentation.
  - problem: some queues might be empty while some might be loaded.

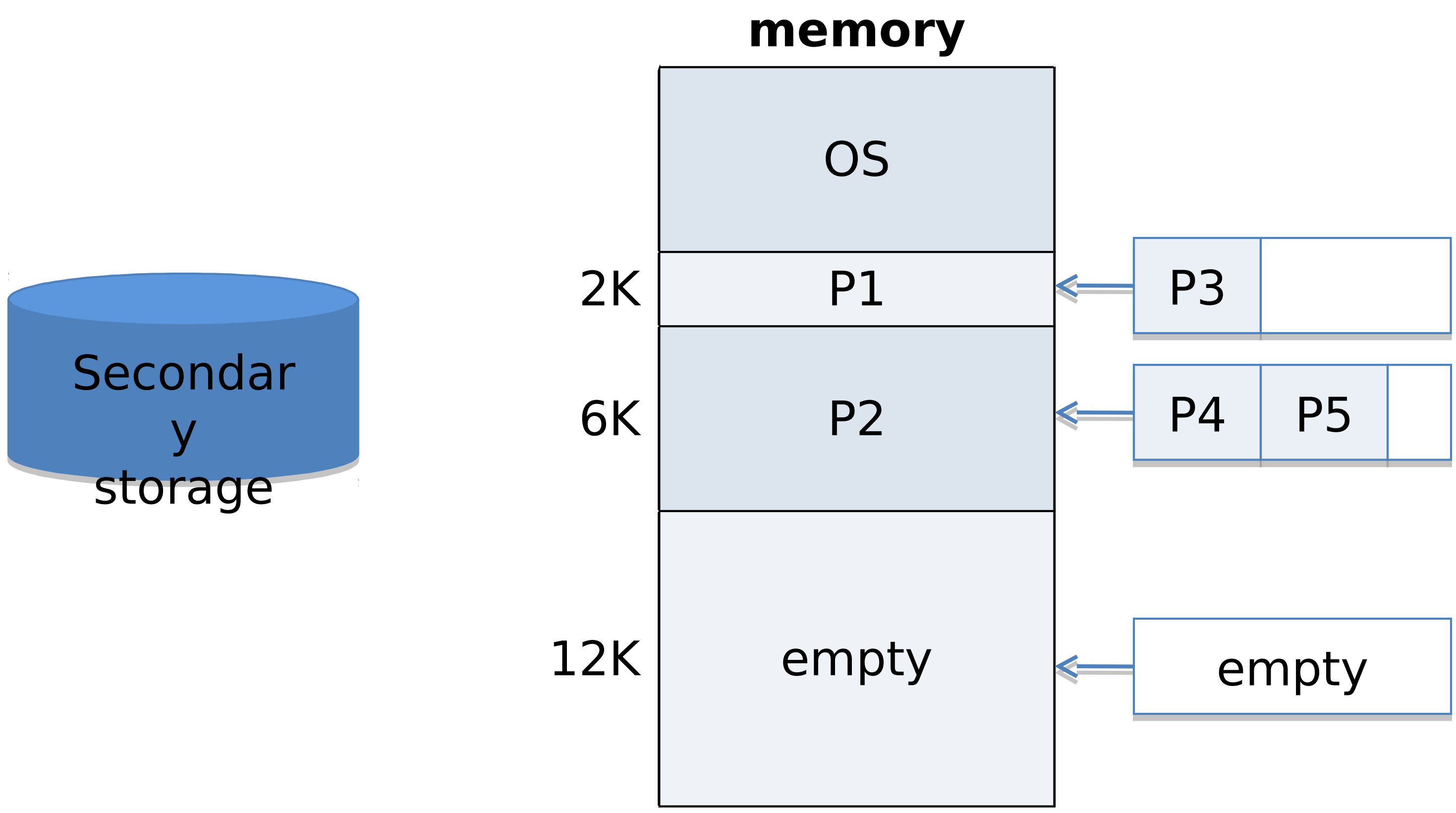


Placement Algorithm with Partitions

- Unequal-size partitions, using a single queue:
  - when its time to load a process into memory, the smallest available partition that will hold the process is selected.
  - increases the level of multiprogramming at the expense of internal fragmentation.
    - Because, maybe later process requires smaller space, but there is no available partition

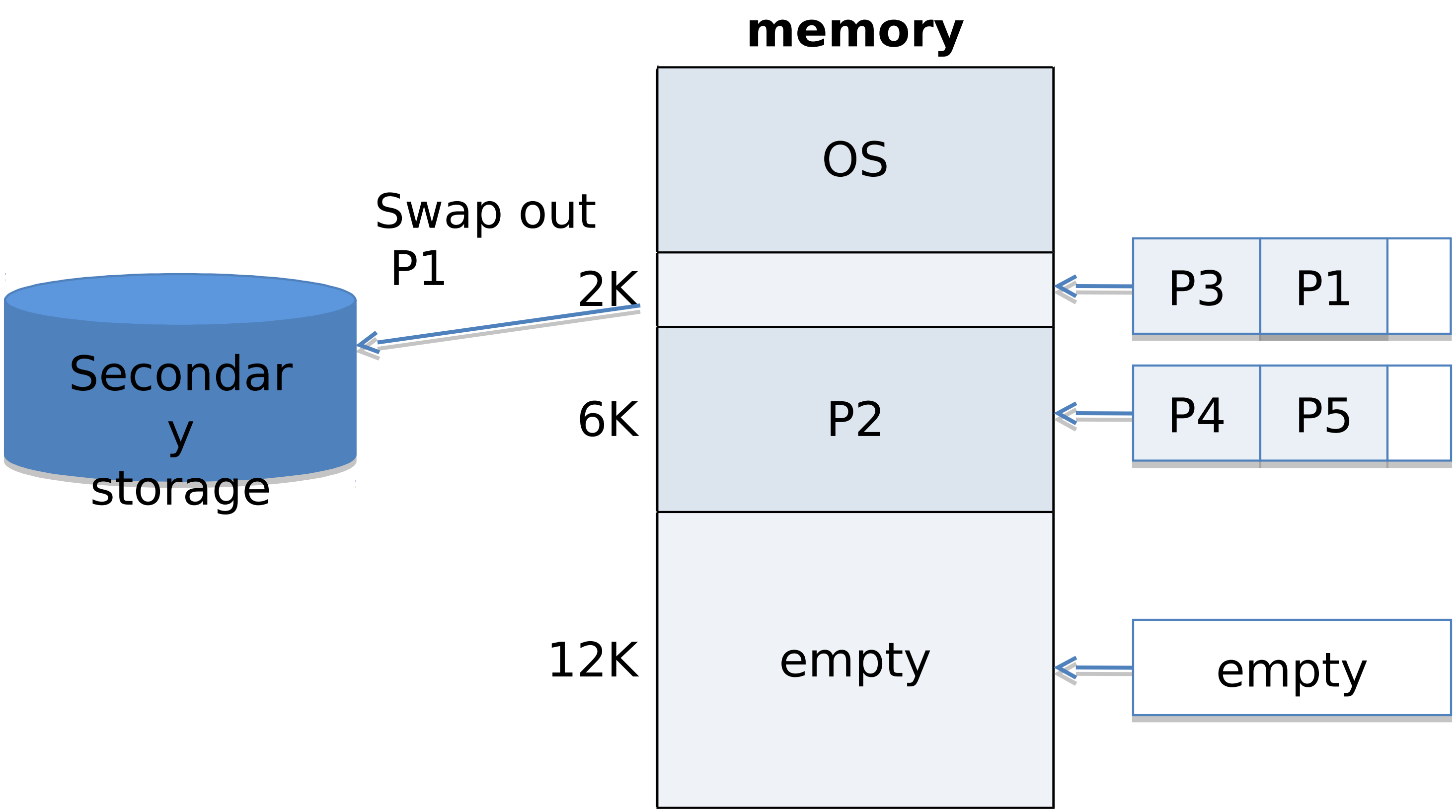


Fixed Partitioning with Swapping

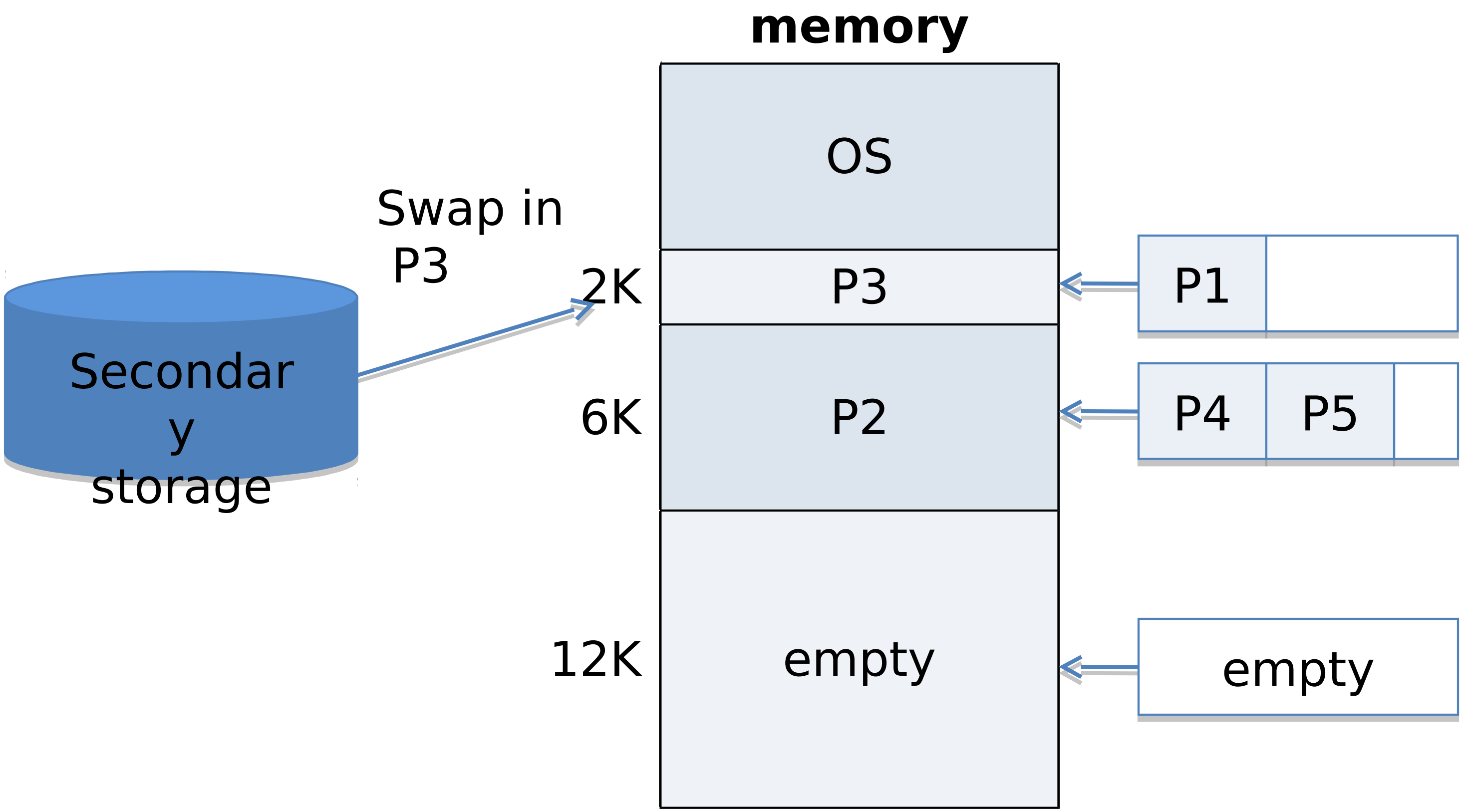


PPTs.2012\PPTs from others\www.eee.metu.edu.tr_~halici_courses_442\ch3part1.ppt

Fixed Partitioning with Swapping

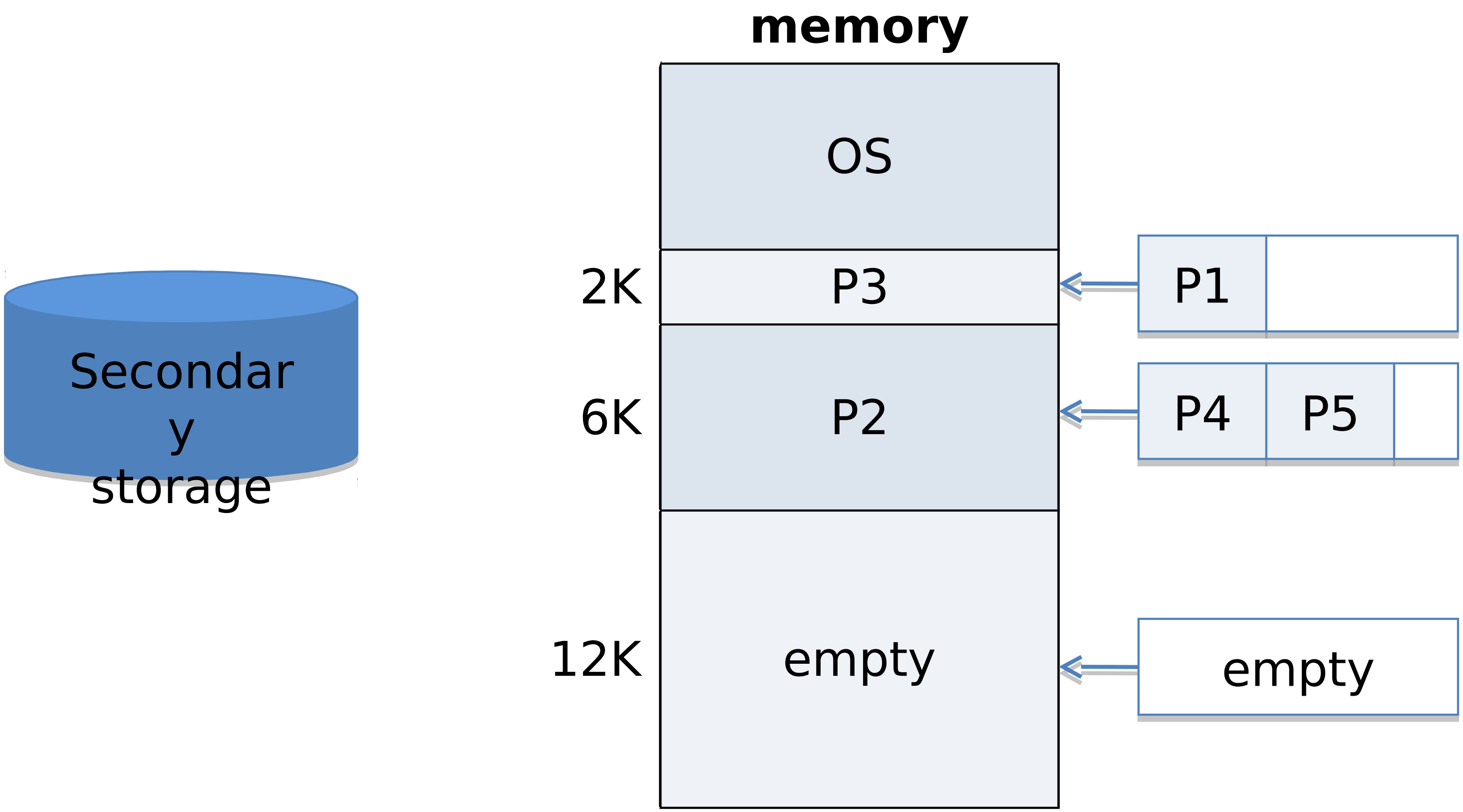


Fixed Partitioning with Swapping



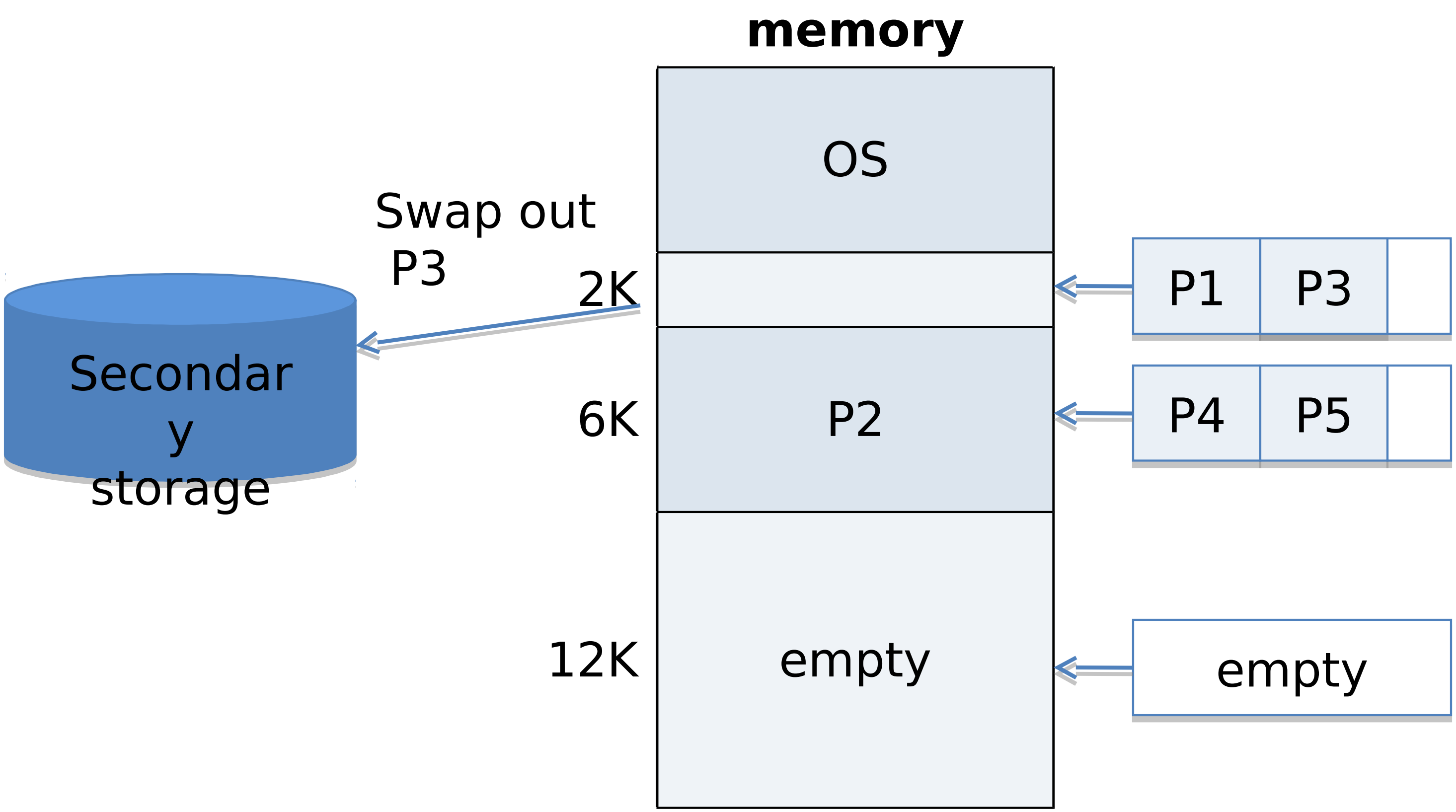
PPTs.2012\PPTs from others\www.eee.metu.edu.tr_~halici_courses_442\ch3part1.ppt

Fixed Partitioning with Swapping



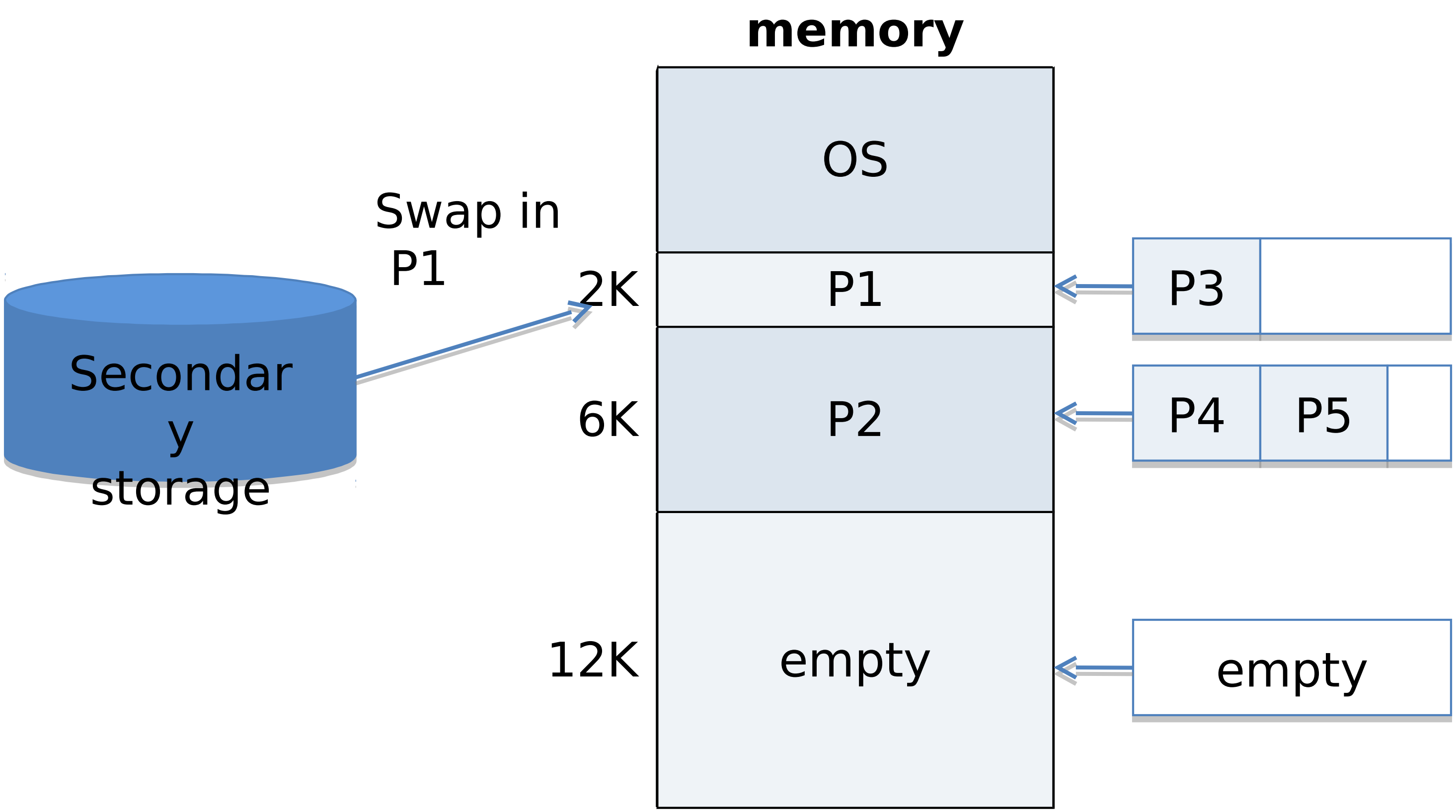
PPTs.2012\PPTs from others\www.eee.metu.edu.tr_~halici_courses_442\ch3part1.ppt

Fixed Partitioning with Swapping



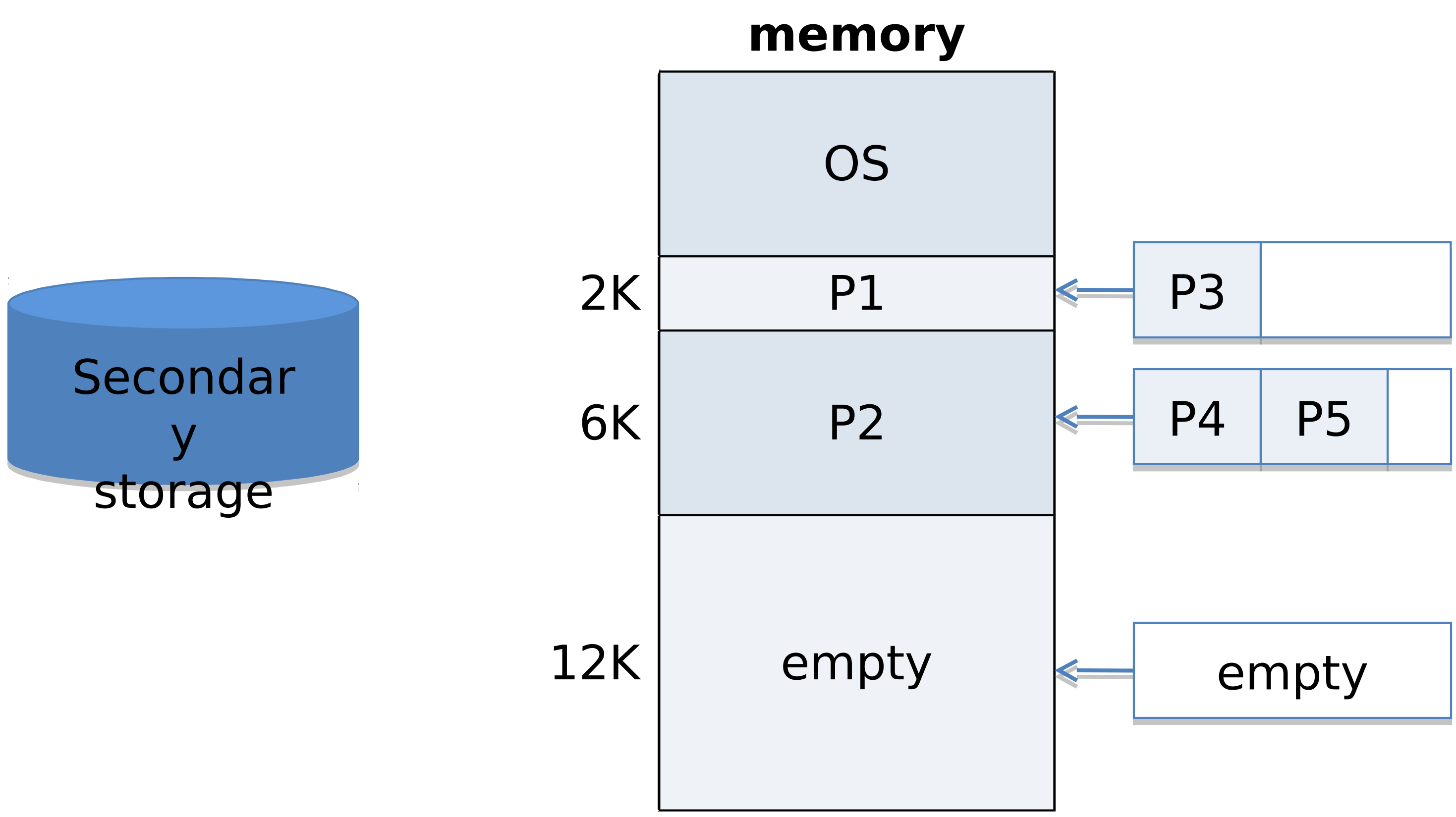
PPTs.2012\PPTs from others\www.eee.metu.edu.tr_~halici_courses_442\ch3part1.ppt

Fixed Partitioning with Swapping



PPTs.2012\PPTs from others\www.eee.metu.edu.tr_~halici_courses_442\ch3part1.ppt

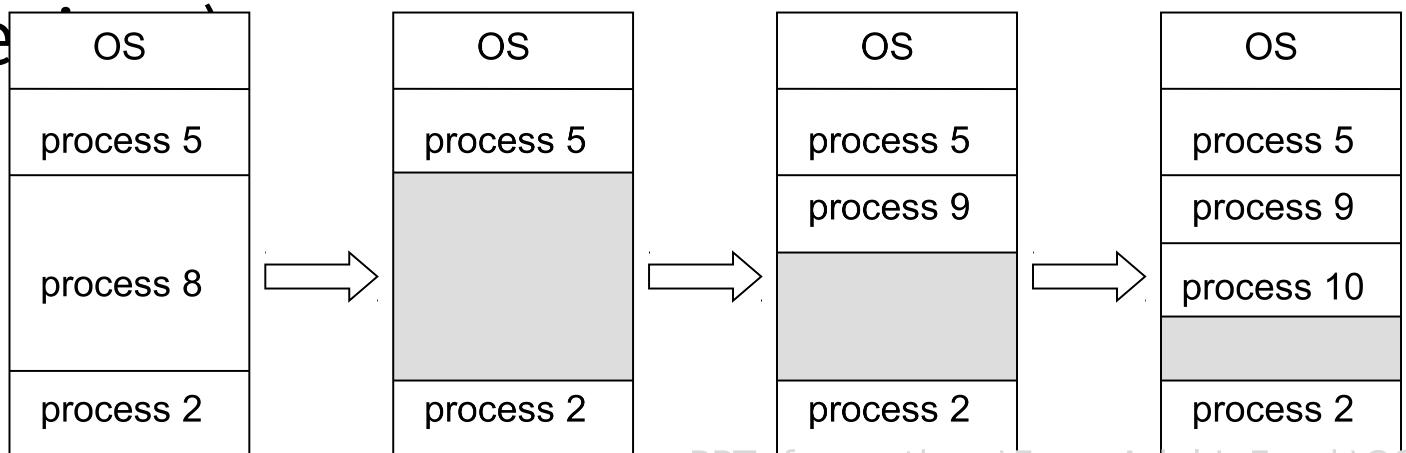
Fixed Partitioning with Swapping



PPTs.2012\PPTs from others\www.eee.metu.edu.tr_~halici_courses_442\ch3part1.ppt

Variable Partitioning

- When a process arrives, it is allocated with memory from a hole large enough to accommodate it.
  - ***Hole*** – block/region of available memory; holes of various sizes are scattered throughout memory.
- Operating system maintains information about: a) allocated partitions b) free partitions (holes/re

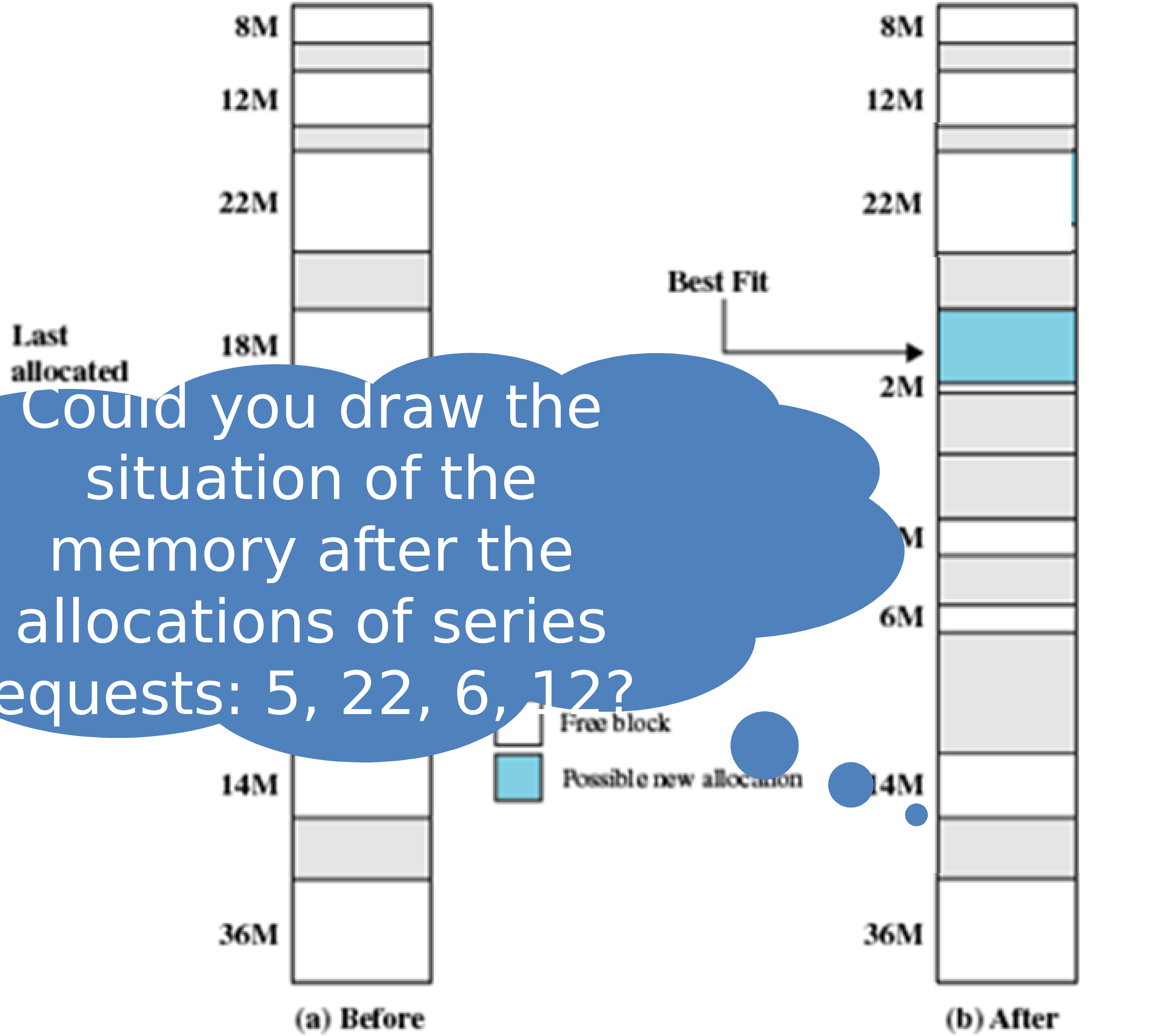


Dynamic Partitioning Placement Algorithm

- Operating system must decide which free b lock to allocate to a process
- **Best-fit algorithm**(最佳适配)
  - Chooses the block that is closest in size to the r equest
  - Worst performer overall
  - Since smallest block is found for process, the s mallest amount of fragmentation is left
  - Memory compaction must be done more often

PPTs from others\SCU_Zhaohui\OS\Chapter07.ppt

Best fit

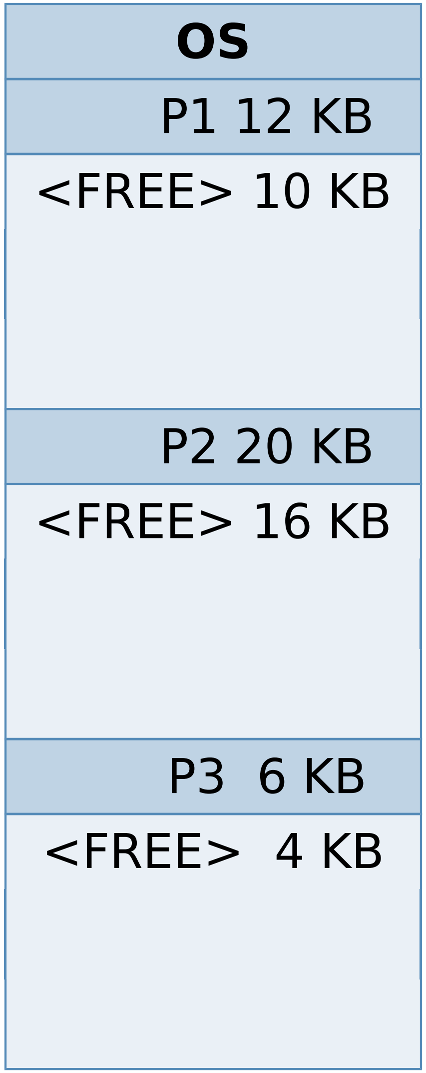


Could you draw the situation of the memory after the allocations of series requests: 5, 22, 6, 12?

Figure 7.5 Example Memory Configuration Before and After Allocation of 16 Mbyte Block

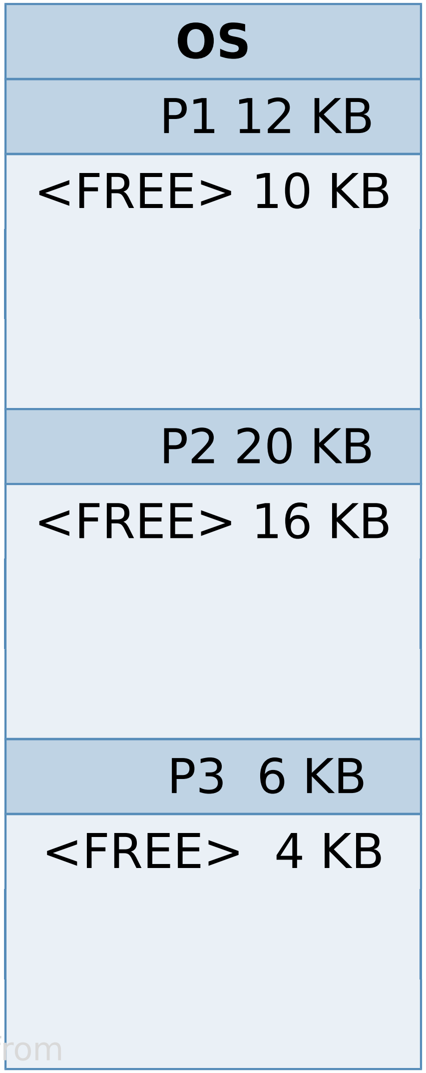
best fit

Initial memory mapping

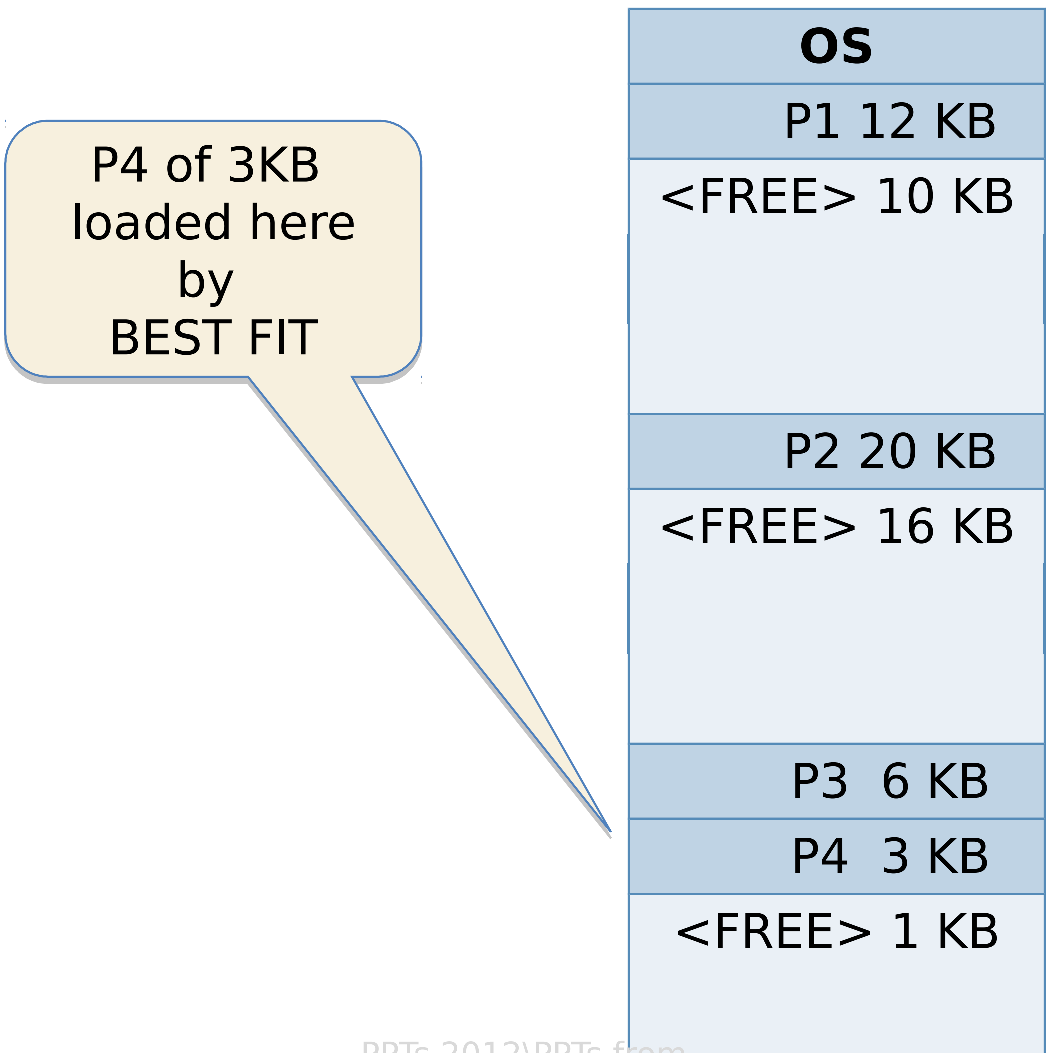


best fit

P4 of 3KB arrives

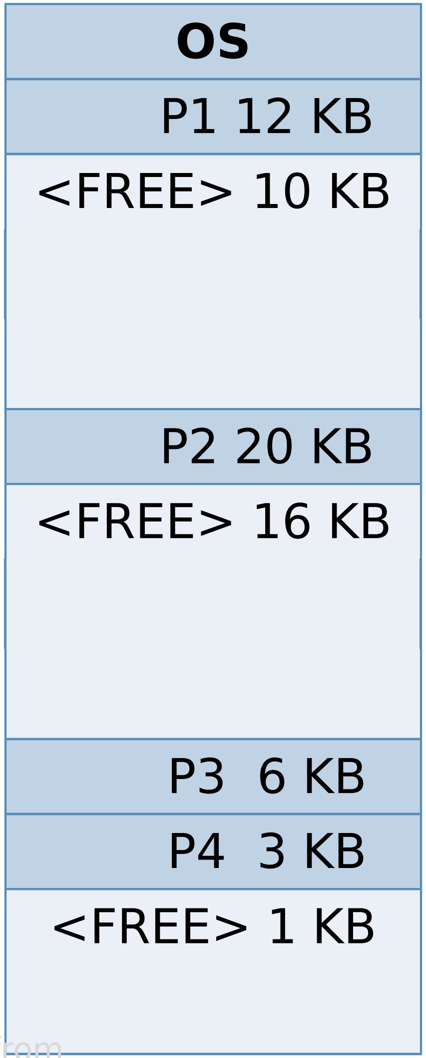


best fit

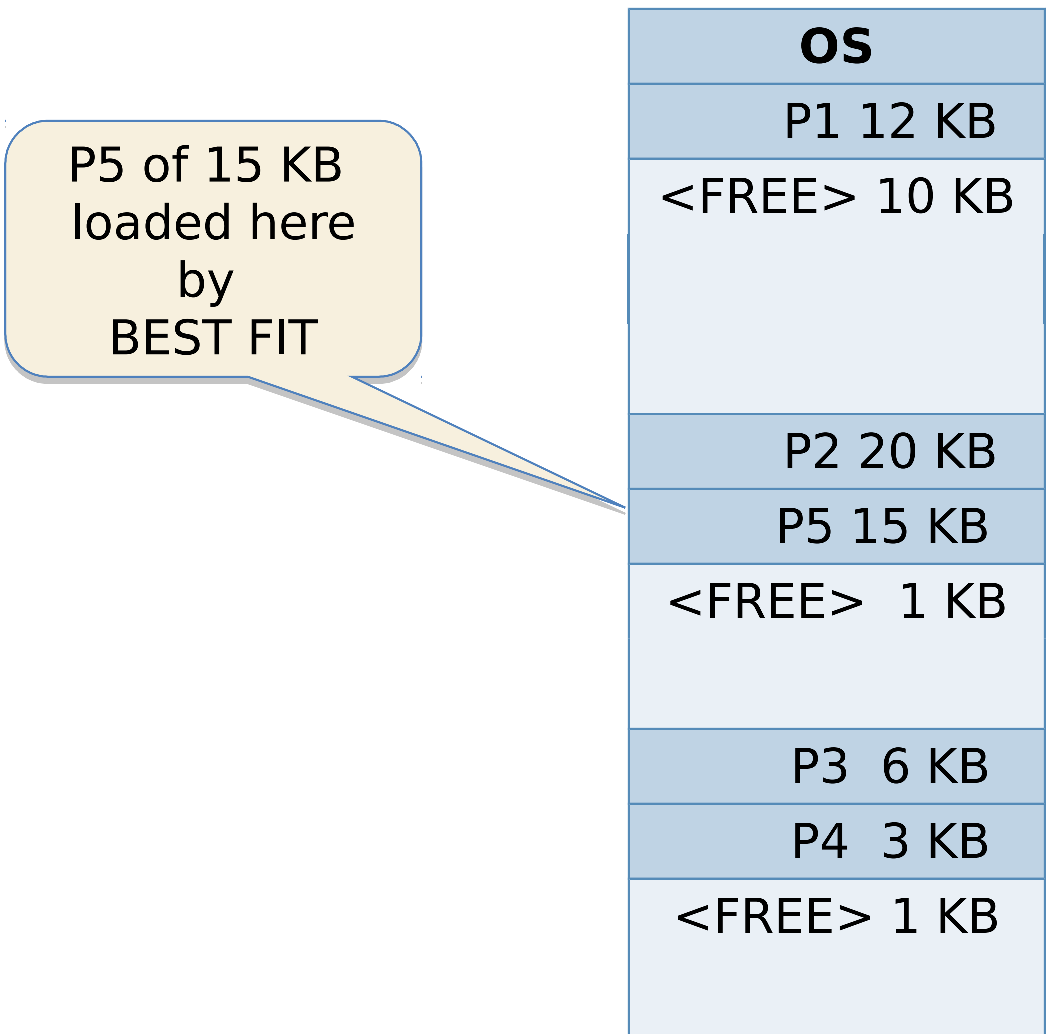

P4 of 3KB loaded here by BEST FIT
OS
P1 12 KB
<FREE> 10 KB
P2 20 KB
<FREE> 16 KB
P3 6 KB
P4 3 KB
<FREE> 1 KB

best fit

P5 of 15KB arrives



best fit



PPTs.2012\PPTs from others\www.eee.metu.edu.tr_~halici_courses_442\ch3part1.ppt

Dynamic Partitioning Placement Algorithm

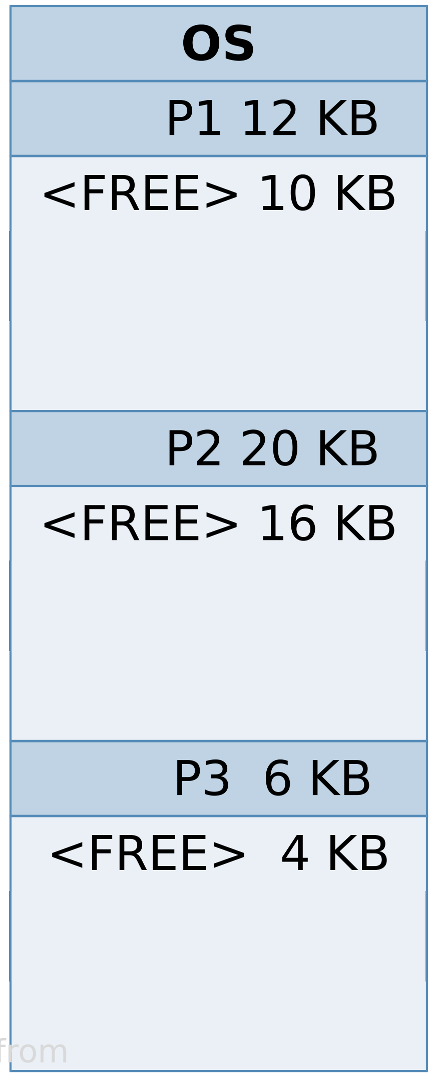
- Worst-fit (最差适配)
  - Scans memory from the location of the last placement
  - Allocate the largest block among those that are large enough for the new process.
  - Again a search of the entire list or sorting it i s needed.
  - This algorithm produces the largest over blo ck.

PPTs from others\SCU_Zhaohui\OS\Chapter07.ppt

PPTs.2012\PPTs from others\www.eee.metu.edu.tr_~halici_courses_442\ch3part1.ppt

worst fit

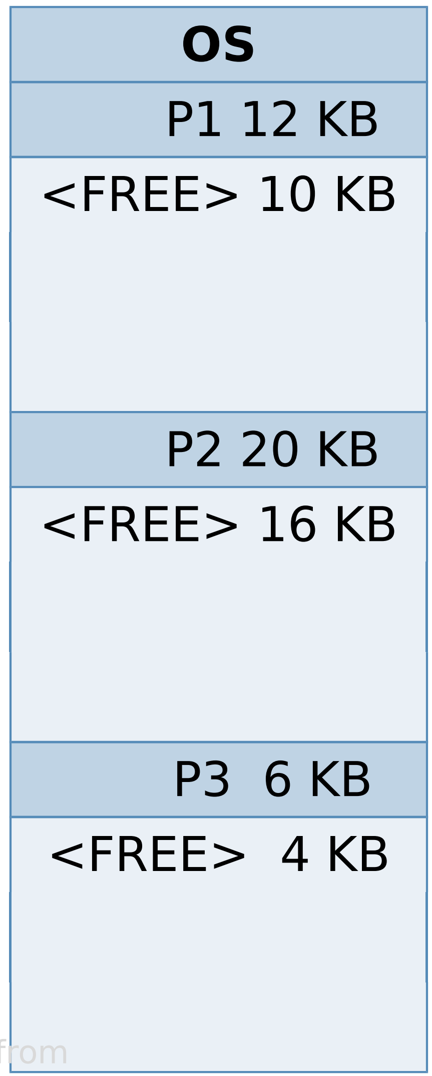
Initial memory mapping



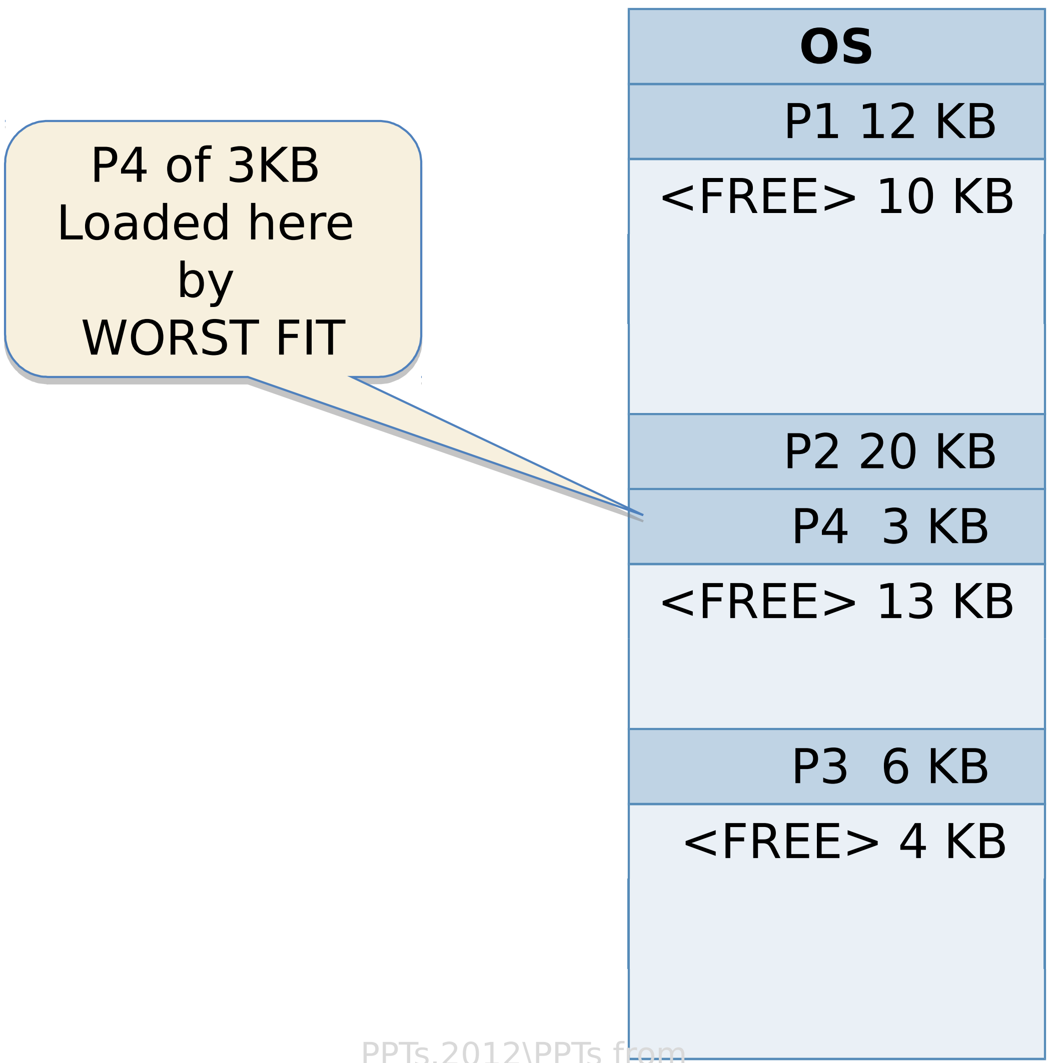
PPTs.2012\PPTs from others\www.eee.metu.edu.tr_~halici_courses_442\ch3part1.ppt

worst fit

P4 of 3KB arrives



worst fit

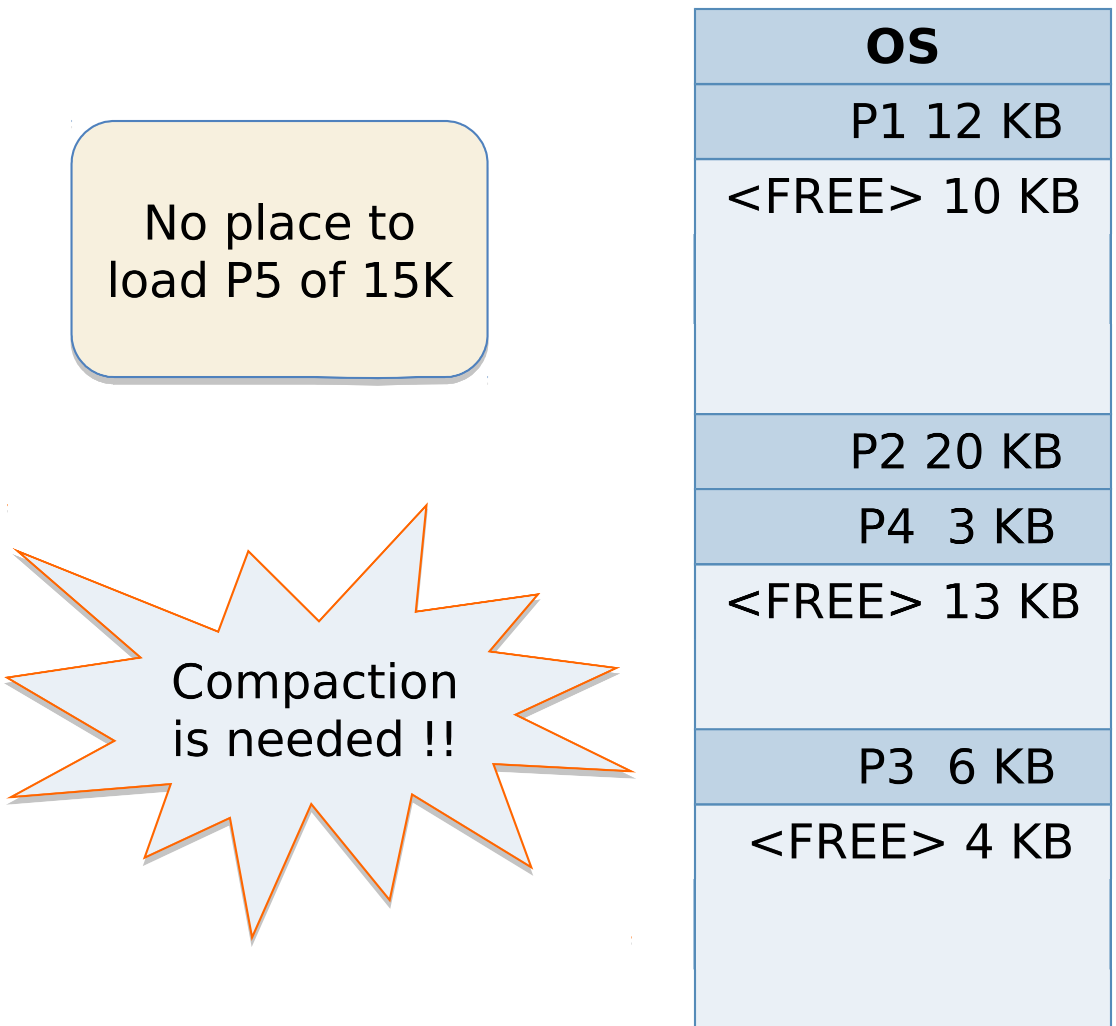

P4 of 3KB
Loaded here
by
WORST FIT
OS
P1 12 KB
<FREE> 10 KB
P2 20 KB
P4 3 KB
<FREE> 13 KB
P3 6 KB
<FREE> 4 KB

worst fit

No place to load P5 of 15K

OS
P1 12 KB
<FREE> 10 KB
P2 20 KB
P4 3 KB
<FREE> 13 KB
P3 6 KB
<FREE> 4 KB

worst fit



No place to load P5 of 15K
Compaction is needed !!
OS
P1 12 KB
<FREE> 10 KB
P2 20 KB
P4 3 KB
<FREE> 13 KB
P3 6 KB
<FREE> 4 KB

First-fit

Could you draw the situation of the memory after the allocations of series requests: 5, 22, 6, 12?

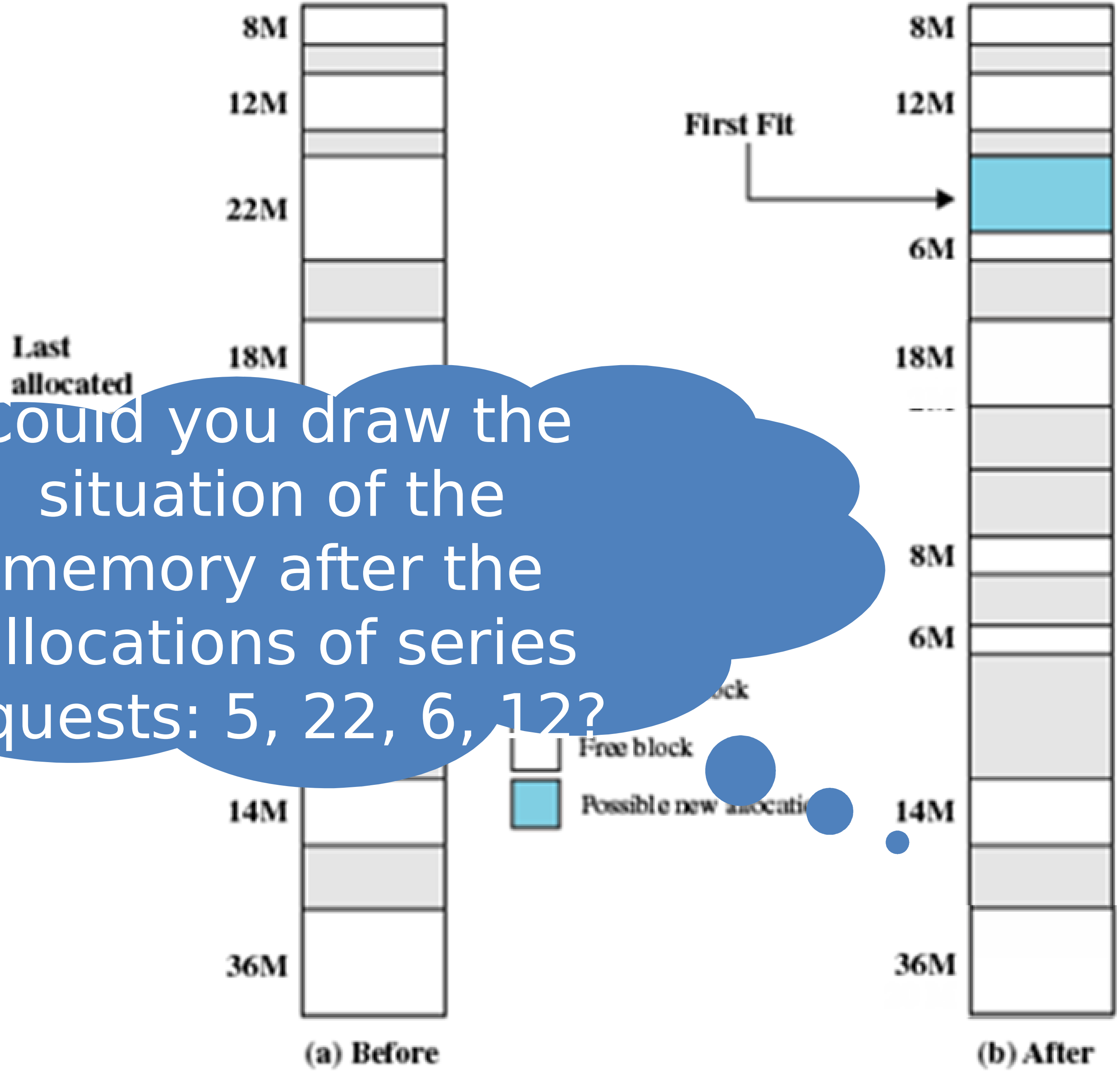


Figure 7.5 Example Memory Configuration Before and After Allocation of 16 Mbyte Block

Dynamic Partitioning Placement Algorithm

- First-fit algorithm(首次适配)
  - Scans memory **form the beginning** and chooses the first available block that is large enough
  - Fastest
  - May have many process loaded in the front end of memory that must be searched over when trying to find a free block

PPTs from others\SCU_Zhaohui\OS\Chapter07.ppt

Next-fit

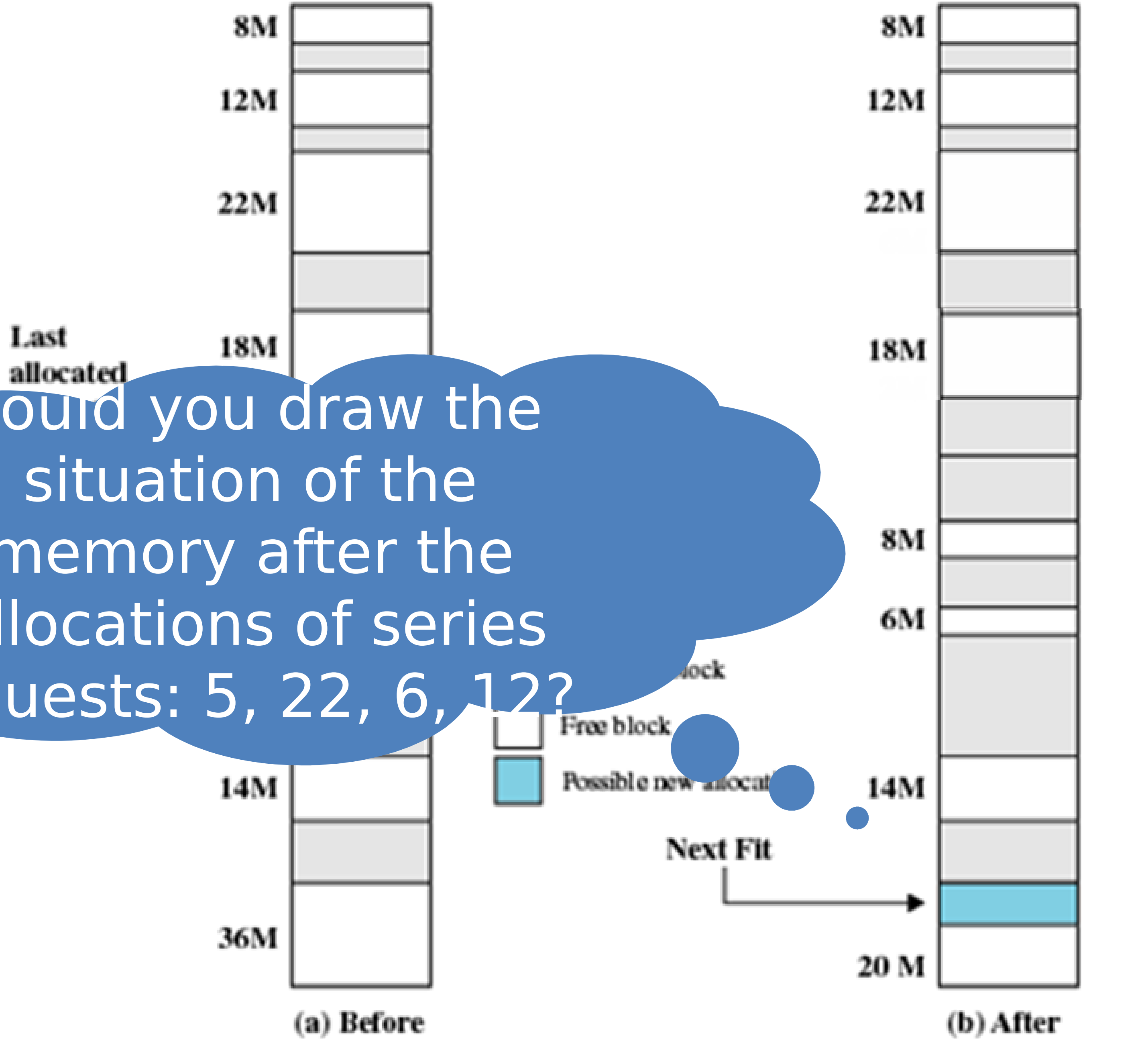


Figure 7.5 Example Memory Configuration Before and After Allocation of 16 Mbyte Block

Could you draw the situation of the memory after the allocations of series requests: 5, 22, 6, 12?

Dynamic Partitioning Placement Algorithm

- Next-fit(邻近适配)
  - Scans memory from the location of the last placement
  - More often allocate a block of memory at th e end of memory where the largest block is f ound
  - The largest block of memory is broken up in to smaller blocks
  - Compaction is required to obtain a large blo ck at the end of memory

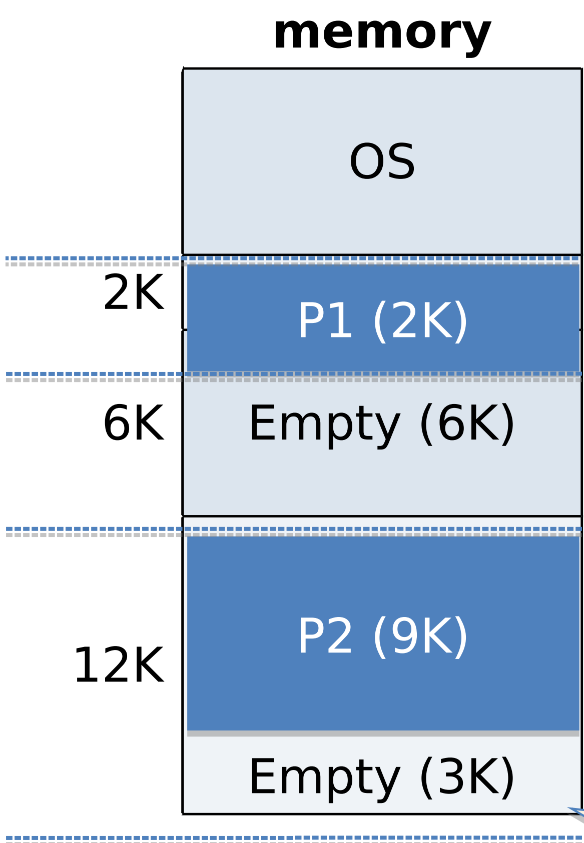
PPTs from others\SCU_Zhaohui\OS\Chapter07.ppt

Internal/External Fragmentation

- There are really two types of fragmentation:

1. **Internal Fragmentation** – allocated memory may be slightly larger than requested memory; this size difference is memory internal to a partition, but not being used.
2. **External Fragmentation** – total memory space exists to satisfy a size *n* request, but that memory is not contiguous.

Fragmentation [碎片整理]



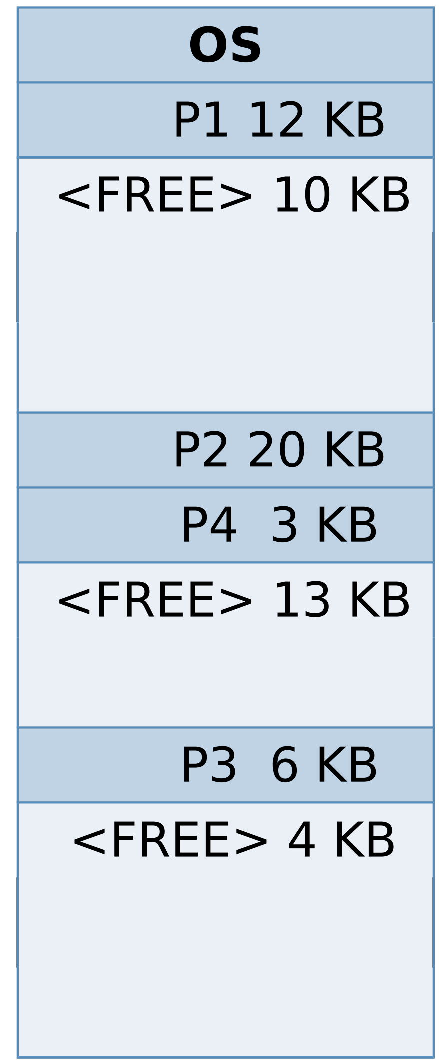
If a whole partition is currently not being used, then it is called *an external fragmentation*.

If a partition is being used by a process requiring some memory smaller than the partition size, then it is called an *internal fragmentation*.

PPTs.2012\PPTs from others\www.eee.metu.edu.tr_~halici_courses_442\ch3part1.ppt

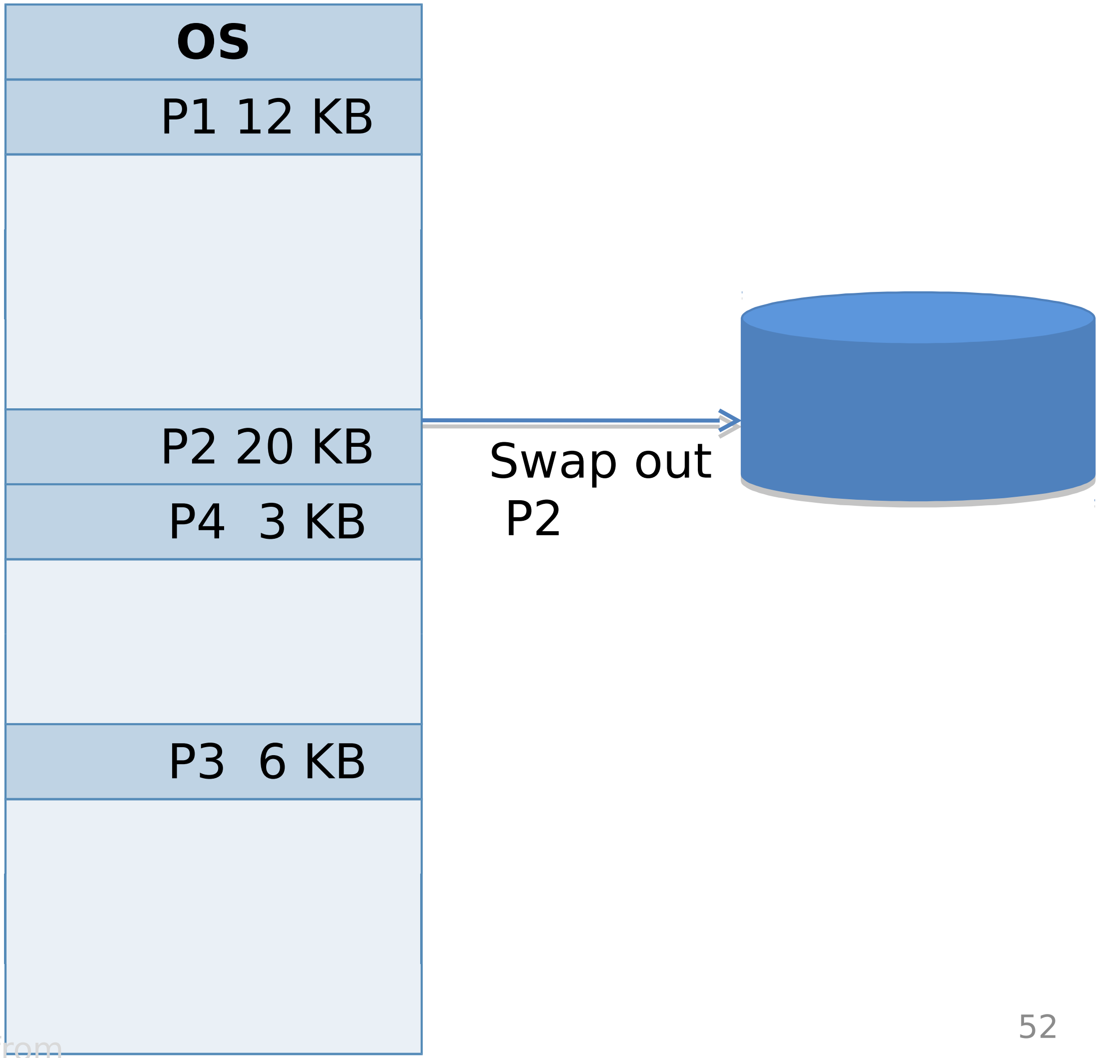
Compaction – Reducing External Fragmentation

- External fragmentation can be resolved through **compaction [压缩]**
  - Shuffles the memory contents to put all free memory together in one block



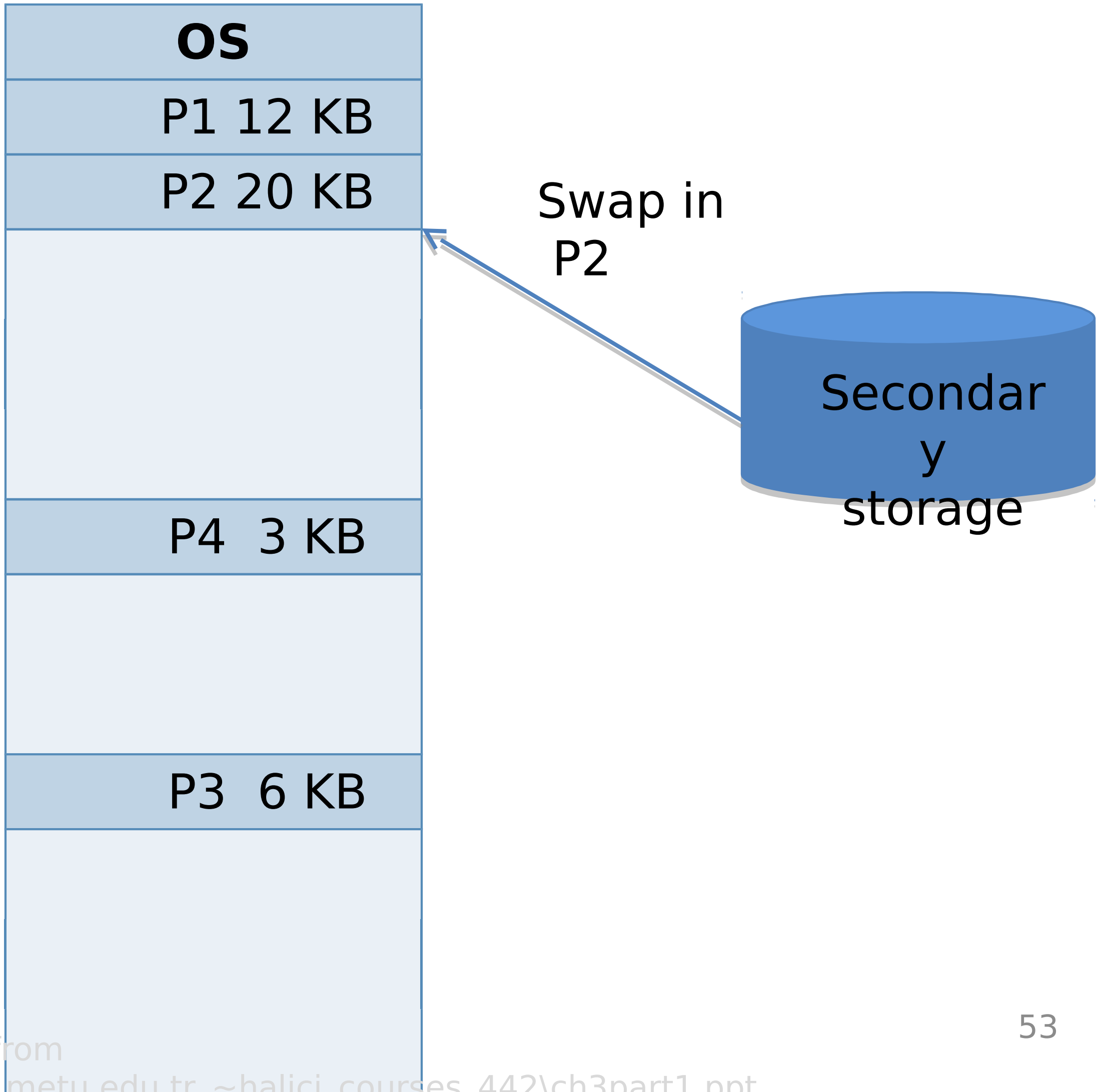
Memory mapping before compaction

compaction



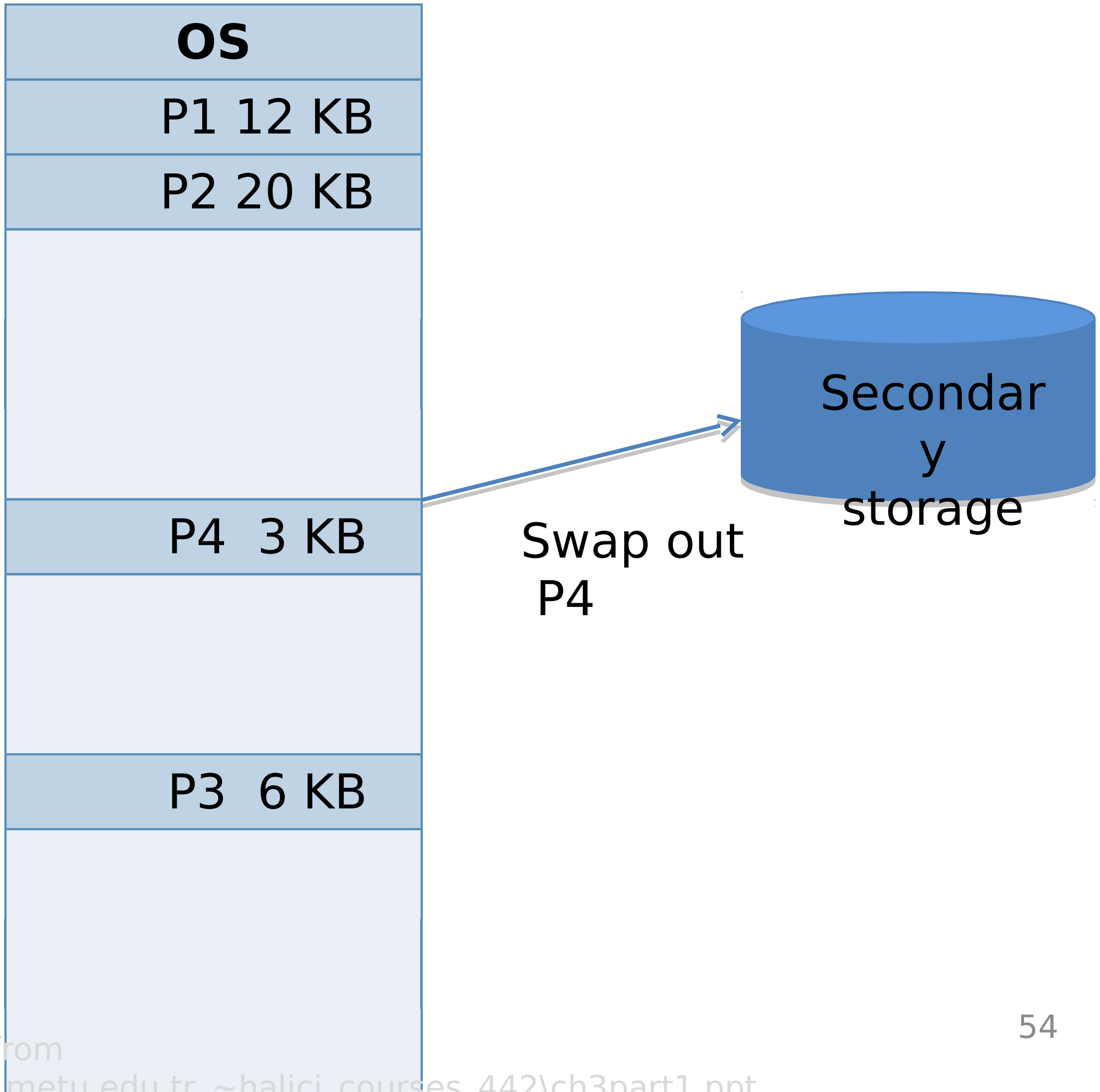
PPTs.2012\PPTs from others\www.eee.metu.edu.tr_~halici_courses_442\ch3part1.ppt

compaction



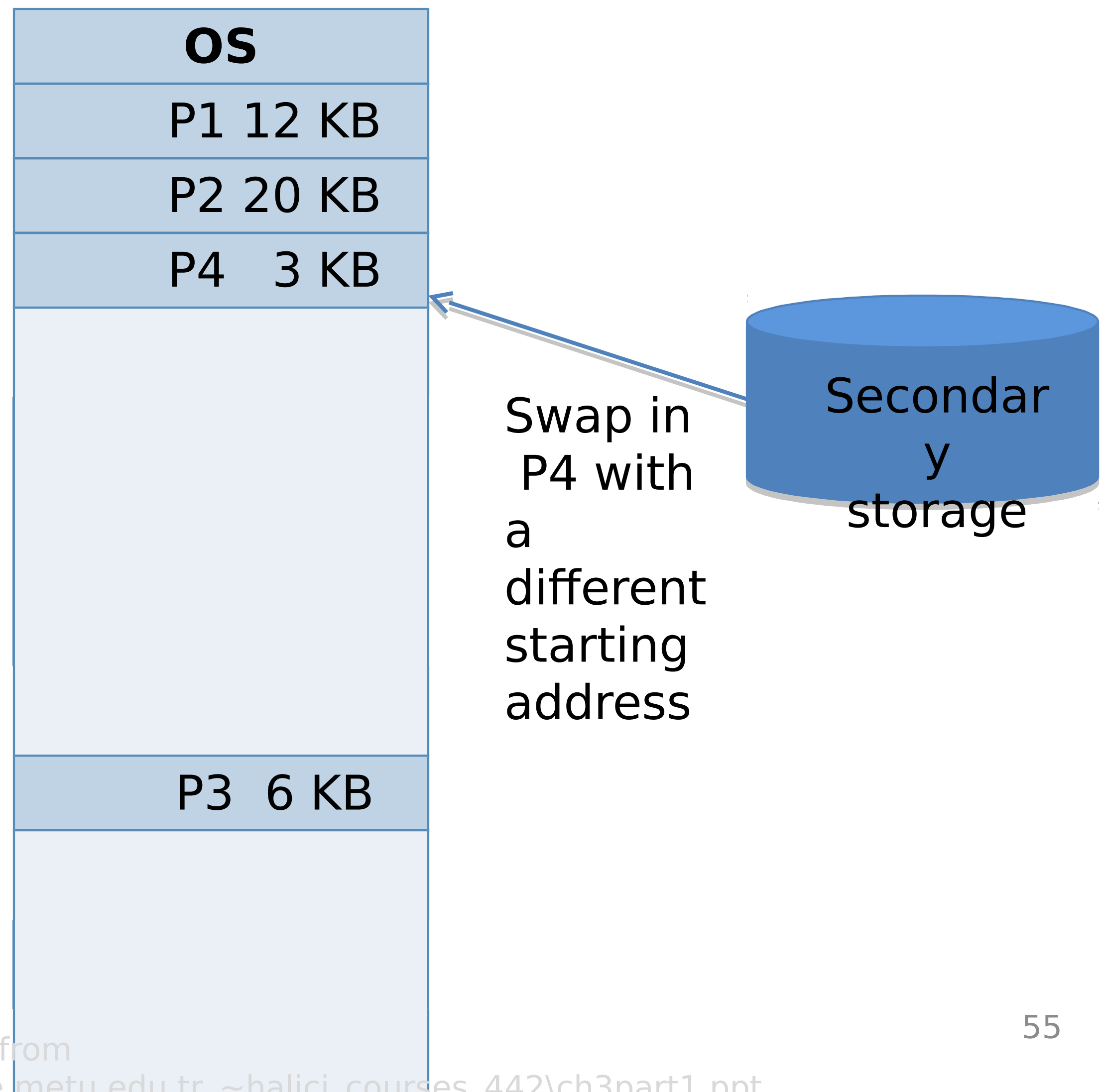
PPTs.2012\PPTs from others\www.eee.metu.edu.tr_~halici_courses_442\ch3part1.ppt

compaction



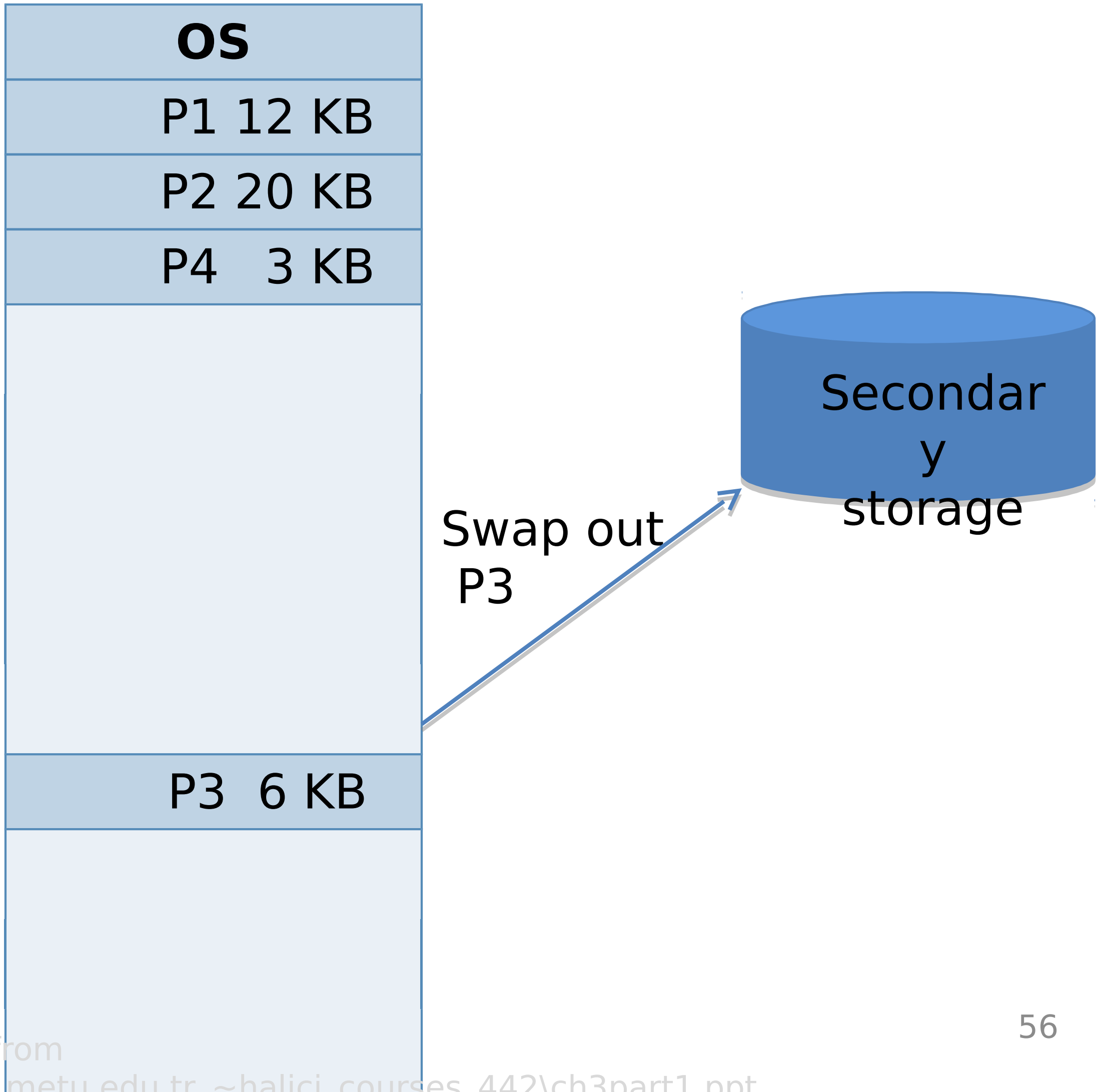
PPTs.2012\PPTs from others\www.eee.metu.edu.tr_~halici_courses_442\ch3part1.ppt

compaction



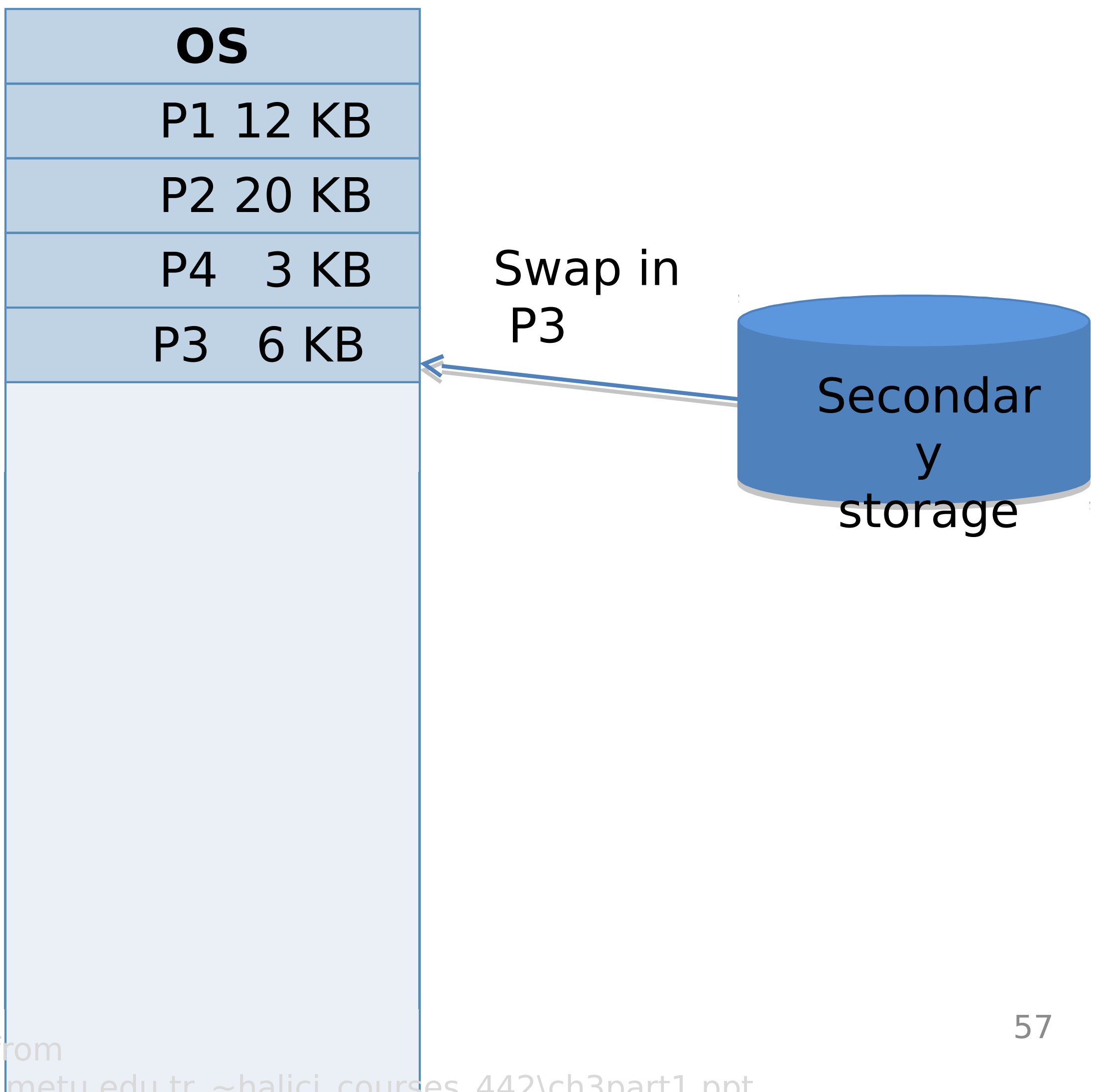
PPTs.2012\PPTs from others\www.eee.metu.edu.tr_~halici_courses_442\ch3part1.ppt

compaction



PPTs.2012\PPTs from others\www.eee.metu.edu.tr_~halici_courses_442\ch3part1.ppt

compaction



PPTs.2012\PPTs from others\www.eee.metu.edu.tr_~halici_courses_442\ch3part1.ppt

compaction

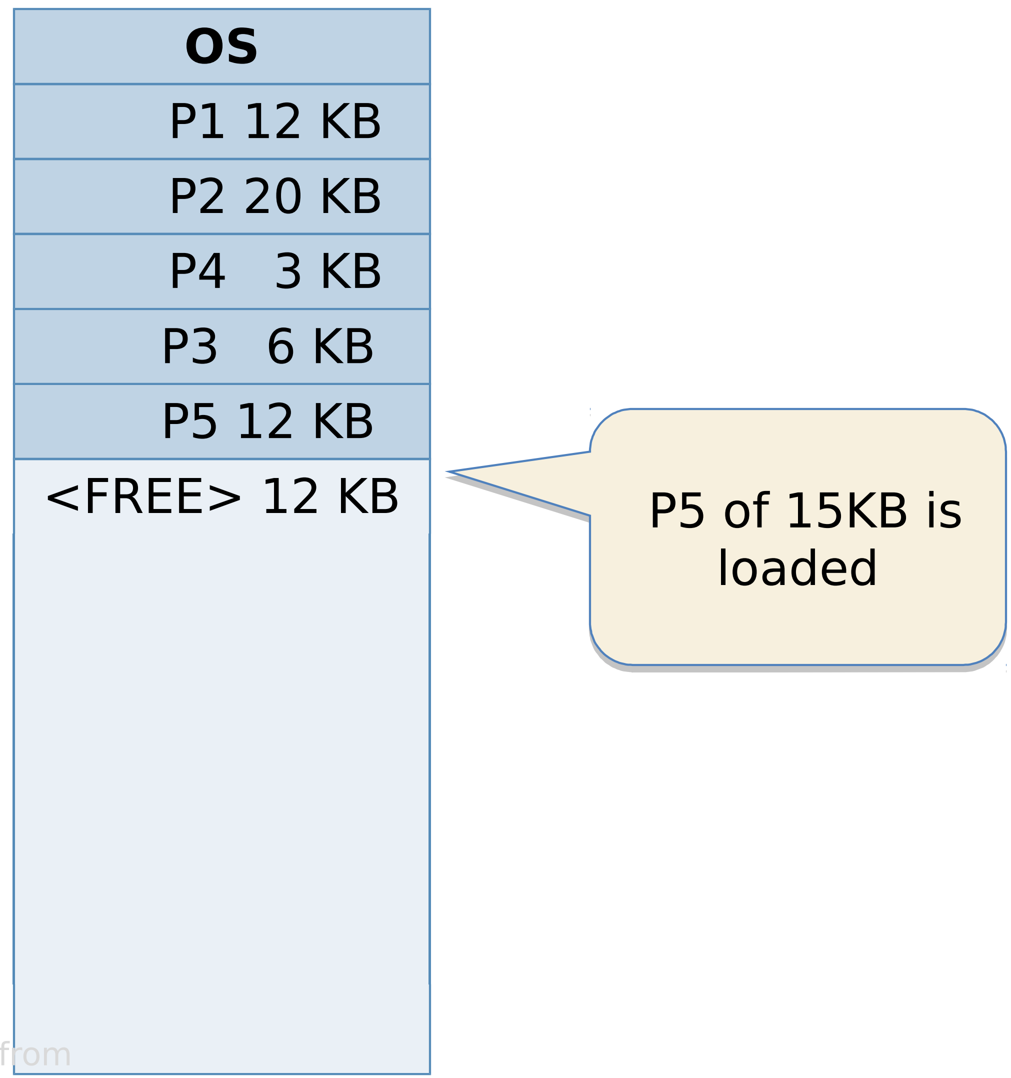
Memory mapping after compaction

OS
P1 12 KB
P2 20 KB
P4 3 KB
P3 6 KB
<FREE> 27 KB

Now P5 of 15KB can be loaded here

PPTs.2012\PPTs from others\www.eee.metu.edu.tr_~halici_courses_442\ch3part1.ppt

compaction



PPTs.2012\PPTs from others\www.eee.metu.edu.tr_~halici_courses_442\ch3part1.ppt

- Only possible if relocation is dynamic
  - Relocation requires moving program and data, changing the base register

- Compaction algorithm is expensive, but so is not making efficient use of memory, especially with a lot of concurrent processes
  - I/O problem:
    - Lock job in memory while it is involved in I/O.
    - Do I/O only into OS buffers.

Concluded rules

- MM and program could be cut into segment
  - why not using same size?
  - Later **paging** scheme inherits this with same siz ed cutting

- The segment could be swapped into or out of MM when needed
  - **Swapping** is the basis of later VM
  - Making the MM management to execute your pr ogram transparently

Replacement Algorithm [替换算法]

- When all processes in main memory are blocked, the OS must choose which process to replace:
  - A process must be swapped out (to a Blocked-Suspend state) and be replaced by a process from the Ready-Suspend queue or a new process.
  - Will be discussed in later **virtual memory**

Me
mor
y

- Basic concepts
  - From Logic address to physical address
  - MMU for relocation – address translation
- Basic techniques of memory manage ment
  - Partitioning (Static & Dynamic)
  - Other auxiliary skills – Overlay and DDL
- For OS space
  - Knuth’ s Buddy System

Program vs. Memory sizes

- What to do when program size is larger than the amount of memory/partition (that exists or can be) allocated to it?
- There **were** two basic attempts **within real memory management**:

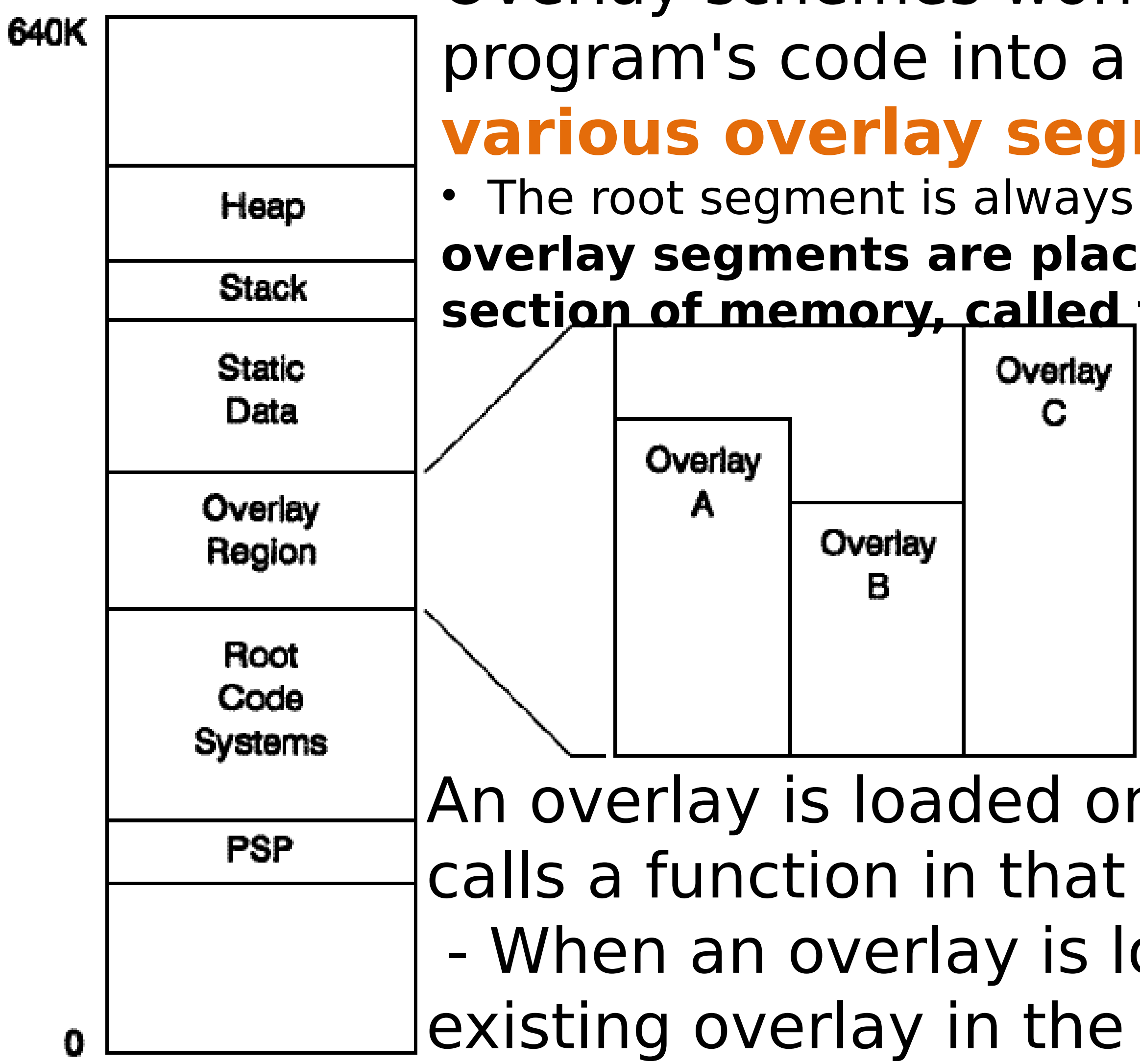
1. Overlays [覆盖，现在在嵌入式系统中找到了新的应用]
2. Dynamic Linking (Libraries – **DLLs**)

PPTs from others\From Ariel J. Frank\OS381\os7-1_rea.ppt

1. Overlay – once known as "Virtual Memory For 640K DOS"

http://www.digitalmars.com/ctg/vcm.html

Figure 1



Overlay schemes work by dividing up a program's code into a **root segment** and **various overlay segments**.

- The root segment is always resident in memory. **The overlay segments are placed into a reserved section of memory, called the overlay region**.

An overlay is loaded only when the program calls a function in that overlay.

- When an overlay is loaded, it replaces any existing overlay in the overlay region. The size of the overlay region is the size of the

2. Dynamic Linking

- Linking postponed until execution time.
- Small piece of code, stub, used to locate the a ppropriate memory-resident library routine.
- Stub replaces itself with the address of the ro utine, and executes the routine.
- OS needed to check if routine is in processe s’ memory address.

- Dynamic linking is particularly useful for shar ed/common libraries – here full OS support is needed.

PPTs from others\From Ariel J. Frank\OS381\os7-1_rea.ppt

Advantages of Dynamic Linking

- Executable files can use another version of the external module without the need of being modified.
- Each process is linked to the same external module.
  - Saves disk space.
- The same external modul… main memory only once.
  - Processes can share code and save memory.
- Examples:
  - Windows: external modules are **.DLL** files.
  - Unix: external modules are .SO files (shared library).

Your turn to understand this programming skill

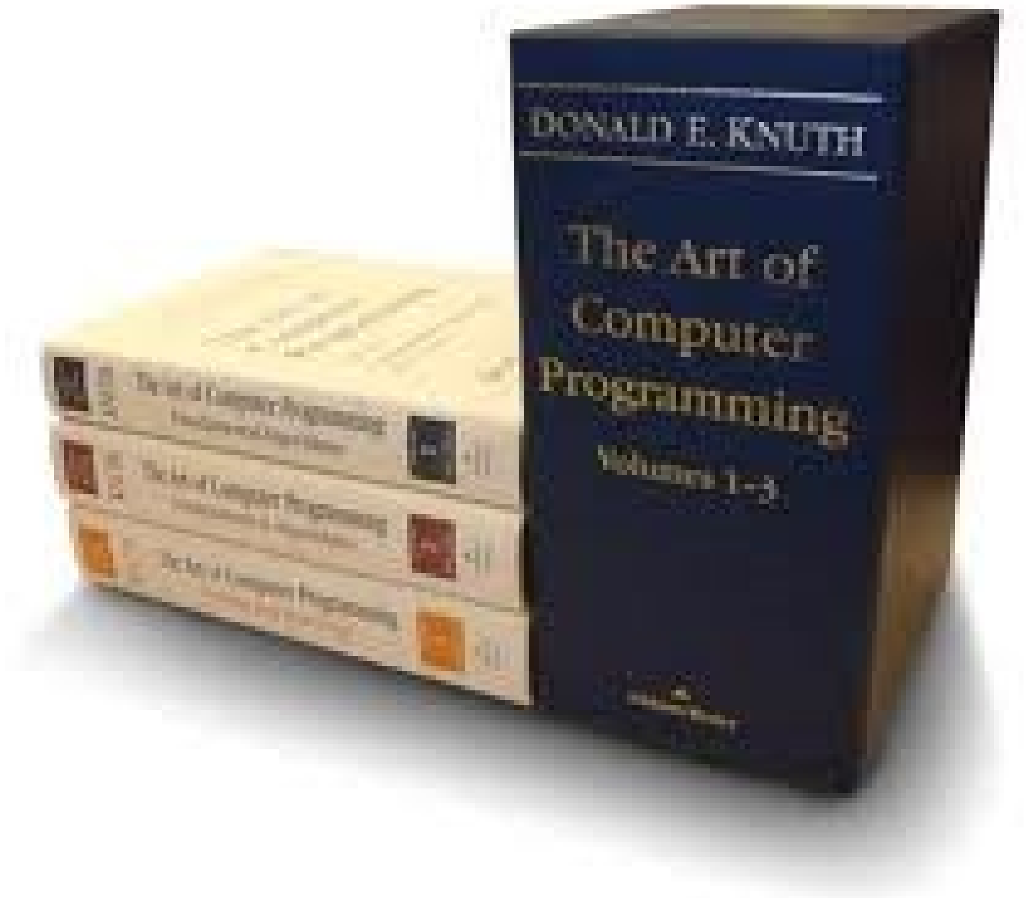
PPTs from others\From Ariel J. Frank\OS381\os7-1_rea.ppt

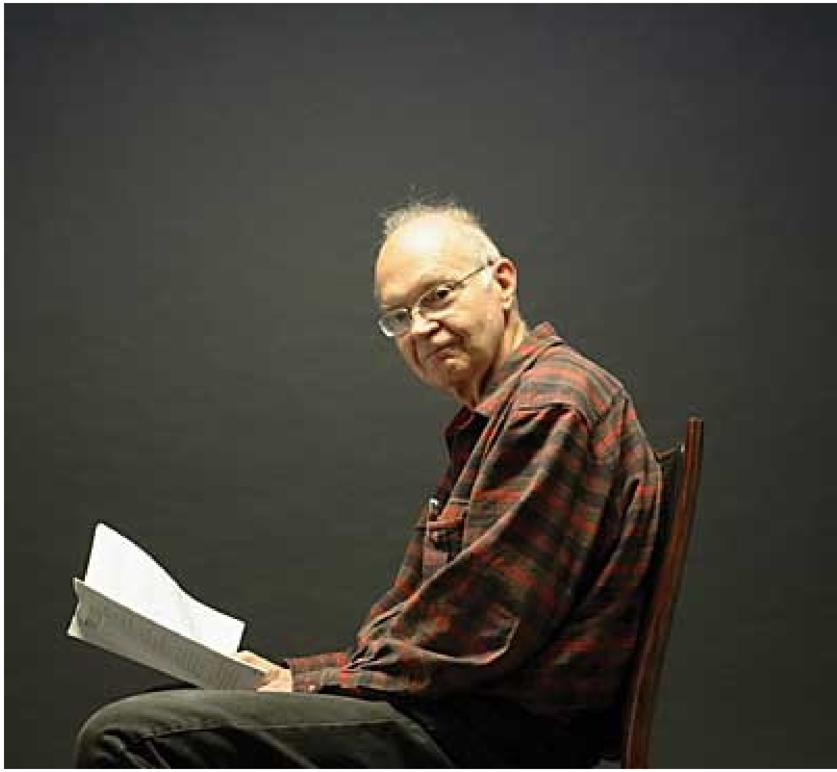
Memory

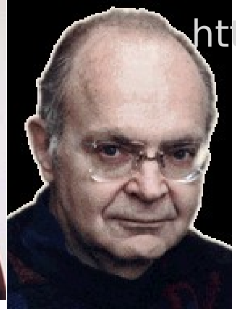
- Basic concepts
  - Memory in storage hierarchy
  - From program to process
    - Static & Dynamic linking
  - Requirements of MM
- Basic techniques of real memory management
  - Overlay
  - Partitioning (Static & Dynamic)
- For OS space
  - Knuth's Buddy System

THE **ART OF COMPUTER PROGRAMMING**

- There is even a book named after that







http://www-cs-faculty.stanford.edu/~uno/

Example: Knuth's Buddy System

- Entire space available is treated as a single block of 2^U

- If a request of size s such that $2^{U-1} < s <= 2^U$, entire block is allocated
  - Otherwise block is split into two equal buddies
  - Process continues **until smallest block greater than or equal to s is generated**

Example of Buddy System

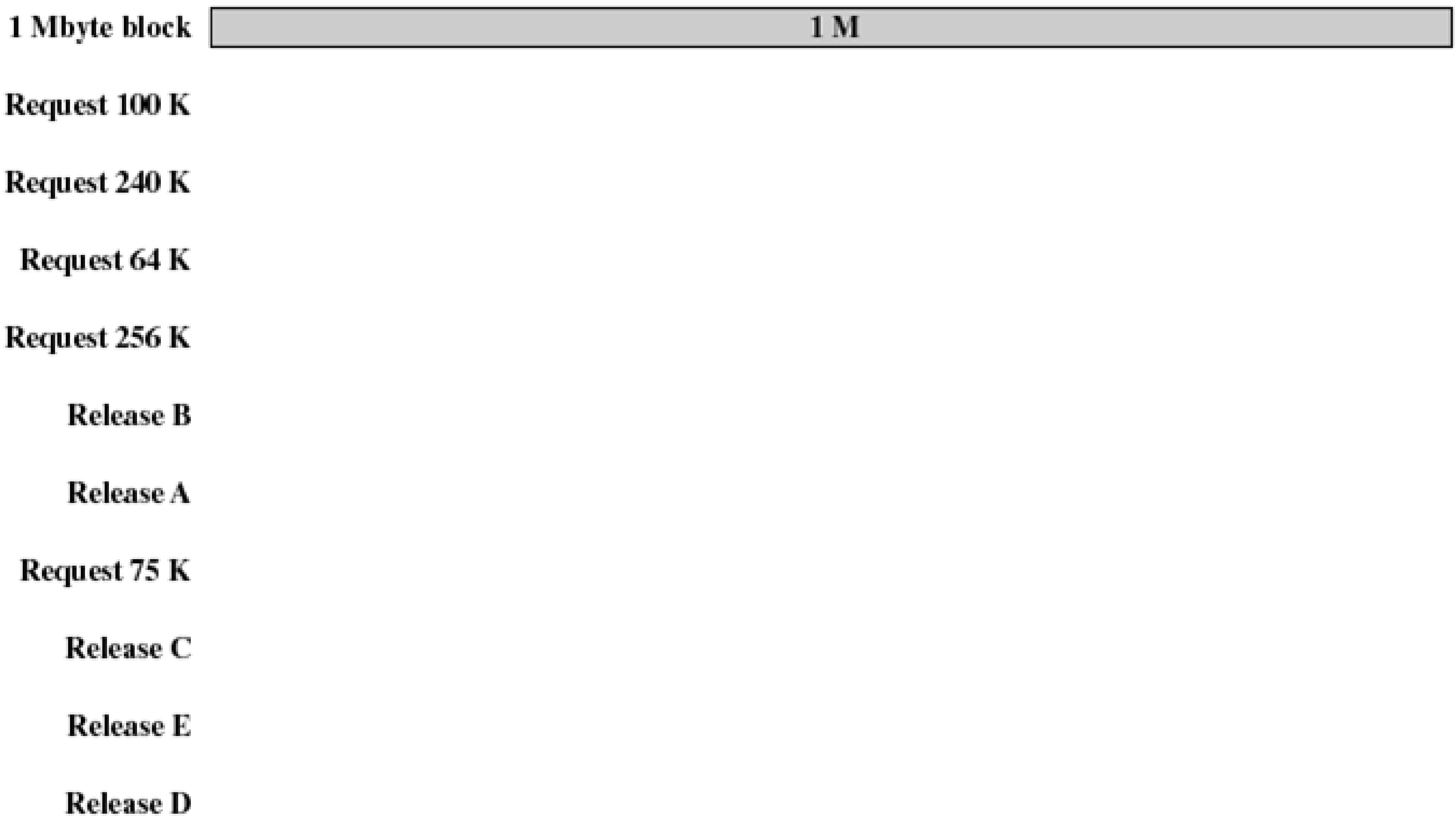


Figure 7.6 Example of Buddy System

Dynamics of Buddy System (1)

- We start with the entire block of size 2^U.
- When a request of size S is made:
  - If $2^{U-1} < S <= 2^U$ then allocate the entire block of size 2^U.
  - Else, split this block into two buddies, each of size 2^{U-1}.
  - If $2^{U-2} < S <= 2^{U-1}$ then allocate one of the 2 buddies.
  - Otherwise one of the 2 buddies is split again.
- This process is repeated until the smallest block gr eater or equal to S is generated.
- Two buddies are coalesced whenever both of them become unallocated.