Western

UNIVERSITY · CANADA

Chapter 9A – Memory Management (Basics)

Spring 2023

Overview

- Background
- Basic Hardware
- Address Binding
- Logical versus Physical Address Space
- Dynamic Loading
- Dynamic Linking and Shared Libraries
- Contiguous Memory Allocation

Background

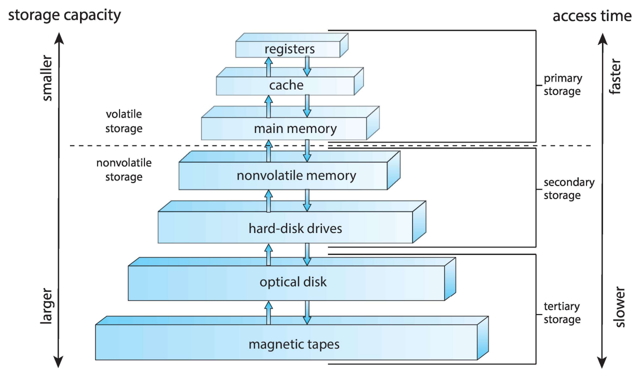
- Modern computer systems maintain several processes (at least partially) in memory during system execution
- Modern computer systems have a lot of memory
- Operating systems must manage the memory well
- Operating systems must closely integrate with the hardware to manage the memory
  - Hardware design and support is important

Background

- Memory management requires
  - Allocating memory to processes when needed from disk
  - Managing memory efficiently during runtime
  - Deallocate memory when processes are done

Basic Hardware

- The CPU can access **main memory** and **registers** directly. If the data is not in memory or in registers, the operating system must fetch the data from disk
    - Register access is very fast (one CPU clock cycle or less)
    - Memory access may generate a memory stall
        - CPUs have a **cache** to speed up memory access at the hardware level
- SSDs are ~4x faster than HDDs
- Memory is ~30x faster than SSDs
- Cache is ~100x faster than memory



Basic Hardware

- The CPU "sees" a stream of addresses and data
- The CPU cannot tell if a process has "permission" to read or write memory addresses

  wasting too much resource
  - Asking the operating system to police this activity comes with significant overhead
  - A hardware solution is needed

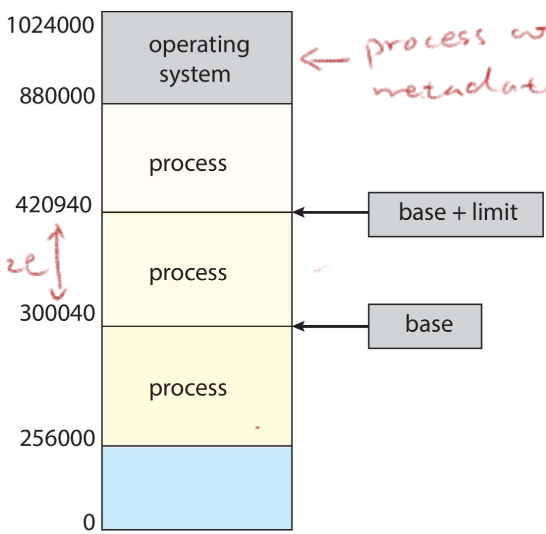
  context switch
- CPUs use a **base register** and a **limit register**

  similar with heap in the stack
  - Base register – contains the lowest address in memory a process can access
  - Limit register – contains the size of the memory space for a process

    outside the block => stop.
  - Base+Limit == the maximum address in memory a process can access

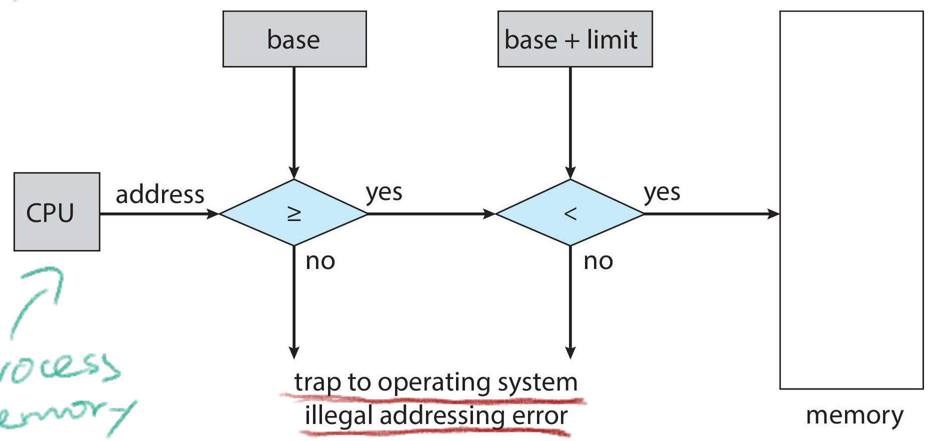
Basic Hardware

- One solution: When in user mode, the CPU must check every memory access to ensure it is between the base and the limit
- The instructions to load the base and limit registers are privileged



process control block, metadata.

(limit -> size ref)



process memory access

(illegal memory access)

Address Binding

it can be at any place in
physical memory randomly

- A user process could reside in any part of the physical memory
  - Data in the process will reside somewhere in its allocated block

process => anywhere
data => allocated block.

- It would be inconvenient for developers to be forced to keep track of exact physical addresses in memory. For example,
  - Programmers usually use a symbolic "address". E.g. `count`
  - The compiler binds the symbolic address to a relocatable address. E.g. "14 bytes from the beginning of this module" base
  - The linker binds the relocatable address to an absolute address. E.g. 74014

$\underbrace{74014 = 74000}_{\text{base}} + \underbrace{14}_{\text{module pos}}$

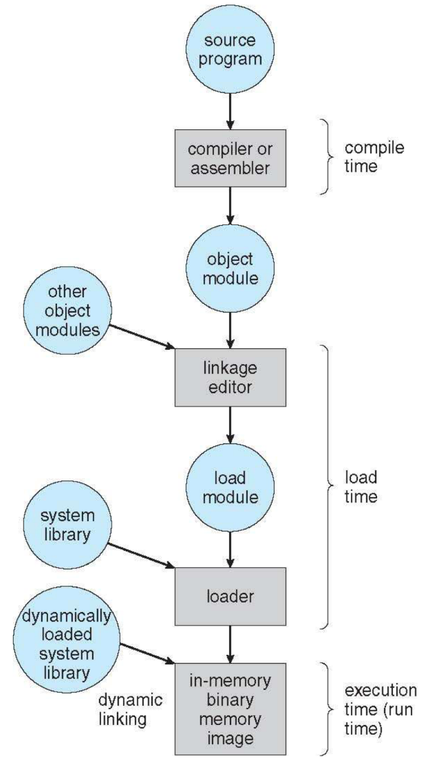
symbolic ("count") => position in module (14 byte from baseline)

compile time - abs
load time - relocatable
execution time - depends on the binding time; moveable mostly.

Address Binding

- The binding step can occur at:
  - **Compile time** – The location in memory may be known at compile time. The compiler generates **absolute** code. If the location changes, the program must be recompiled
    the abs mem location
  - **Load time** – The location in memory may not be known at compile time but it is decided when the program is loaded. The loader can use **relocatable** code. If the location changes, the program just needs to be reloaded
  - **Execution time** – If the process can be moved during its execution from one memory segment to another, binding must wait until run time. Special hardware is required to make this work. This is the method most modern operating systems use.

Address Binding



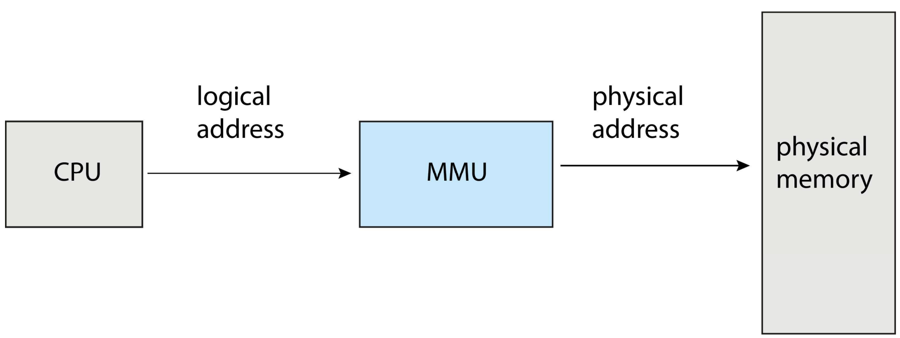
Logical vs Physical Addresses

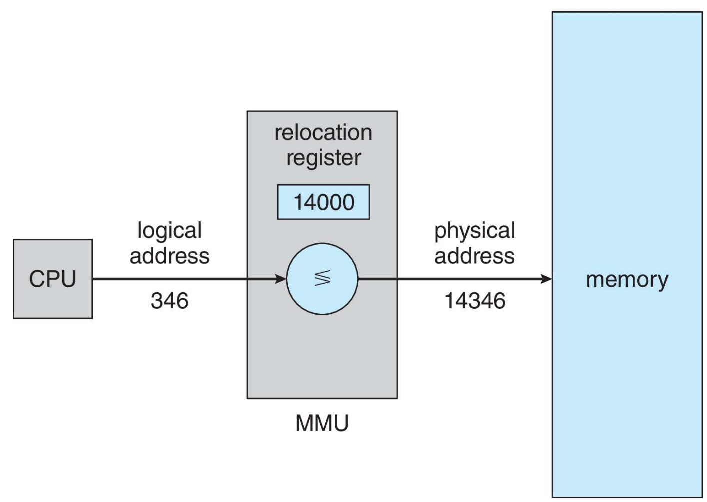
- **Logical address** – The "address" generated by the CPU
  - This is sometimes also known as a virtual address
- **Physical address** – The address of the memory-address register in memory
- If binding occurs at compile time, these are the same. If binding occurs at execution time, these are different

Logical vs Physical Addresses

- The translation from logical address to physical address is handled by a hardware device in the CPU called the **memory-management unit (MMU)**

- A simple implementation: Instead of a base register, use a **relocation register** in the MMU. The value will be added to every logical address to get the physical address





Logical vs Physical Addresses

- The logical addresses are in the range *0* to *max* => limit
  - All addresses generated by the user program *think* it is using address *0* to *max*
- The physical addresses are in the range *R+0* to *R+max* for a base value *R*

Dynamic Loading

- The entire program does not need to be in memory to execute
- To manage memory better, routines are kept on disk in relocatable load format
- Load routines in memory when it is called
  - Unused routines are never loaded (e.g. Very rare error handling functions)
- This is managed in user space without special support from the operating system
  - Operating systems may provide some tools to help implement dynamic loading

Dynamic Linking and Shared Libraries

- Some operating systems only support **static linking**. All system libraries are loaded into the binary program image. This is very inefficient
- **Dynamically linked libraries (DLLs)** (also known as shared libraries) are system libraries that loaded dynamically at execution time
  - This is a more efficient use of memory
  - It is also possible to share loaded libraries among multiple processes
  - Libraries can be updated without relinking the programs that use it. Programs could choose which version to use.

similar with <.h> files in C.
include when needed

Dynamic Linking and Shared Libraries

- Because DLLs are shared, for memory protection, operating system assistance is required

Contiguous Memory Allocation

- An early approach to memory management. The other approach is non-contiguous
  - Better known as **paging** and will be addressed later
- Main memory is divided into two partitions
  - **Operating system** can be in low or high memory. Most operating systems place it in high memory
  - **User processes** are held in the other partition
    - Each process is contained in a single **contiguous** section of memory

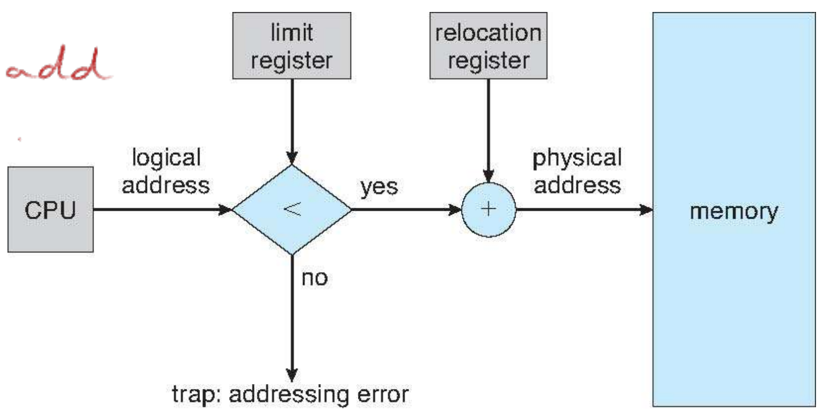
Contiguous Memory Allocation

- To protect user processes from each other and from the operating system itself, the MMU uses the relocation (base) register and the previously mentioned limit register

memory management unit.

- Each logical address must be less than the limit register
- These registers are updated when a context switch occurs

then store the base add of the next process.



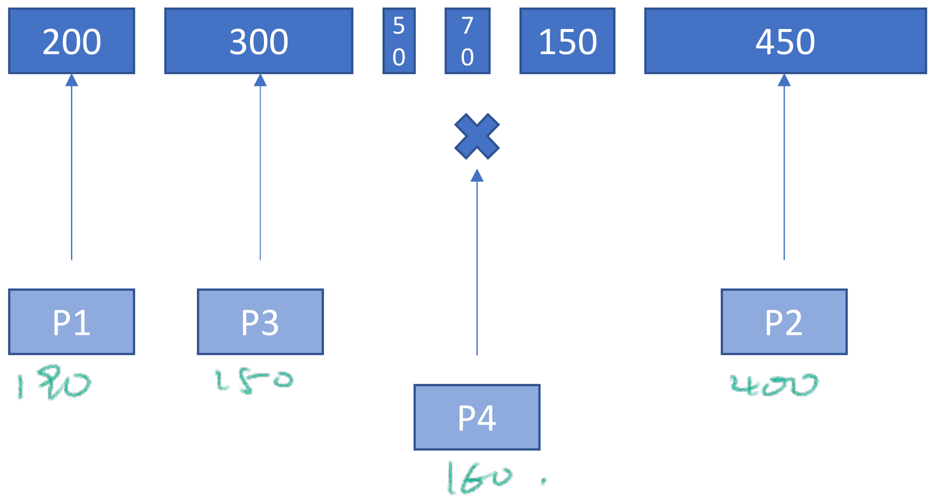
Contiguous Memory Allocation

- Each contiguous section of memory is a fixed or dynamic partition
  - Fixed partition
    - at boot time
    - Partitions are allocated with sizes that never change
    - Each partition may be the same or different sizes depending on implementation
      - If the sizes are different, select a partition for a process by first-fit or best-fit
    - partition is not fully used.
    - **Internal fragmentation** is a problem. There will always be portions of the partition that are wasted e.g. a 16 k partition that has 12 k process, rest of 4 k is wasted.
    - **External fragmentation** is a problem. There may be enough free space but spanning partitions is not allowed a 17 k is never allowed to run.

partition is inefficent for one block.

Contiguous Memory Allocation

- First-fit : find the first available partition and occupy.
  - P1 = 190
  - P2 = 400
  - P3 = 150
  - P4 = 160

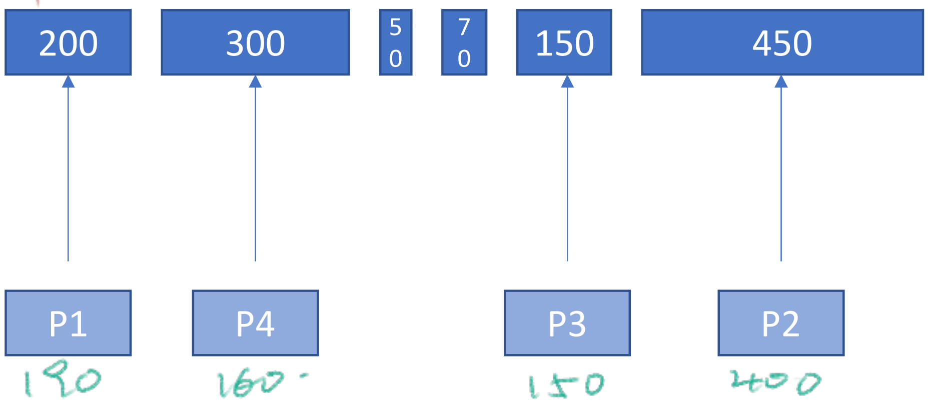


- Internal Fragmentation
  - P1 P2 P3.
    10 + 50 + 150 = 210
- External Fragmentation
  - 50 + 70 + 150 > 160

    P4 : though total size is enough, the process is unable to run.

Contiguous Memory Allocation

- Best-fit : find the best-fit one
  - P1 = 190
  - P2 = 400
  - P3 = 150
  - P4 = 160
- Internal Fragmentation
  - 10 + 50 + 0 + 140 = 200
- External Fragmentation
  - None



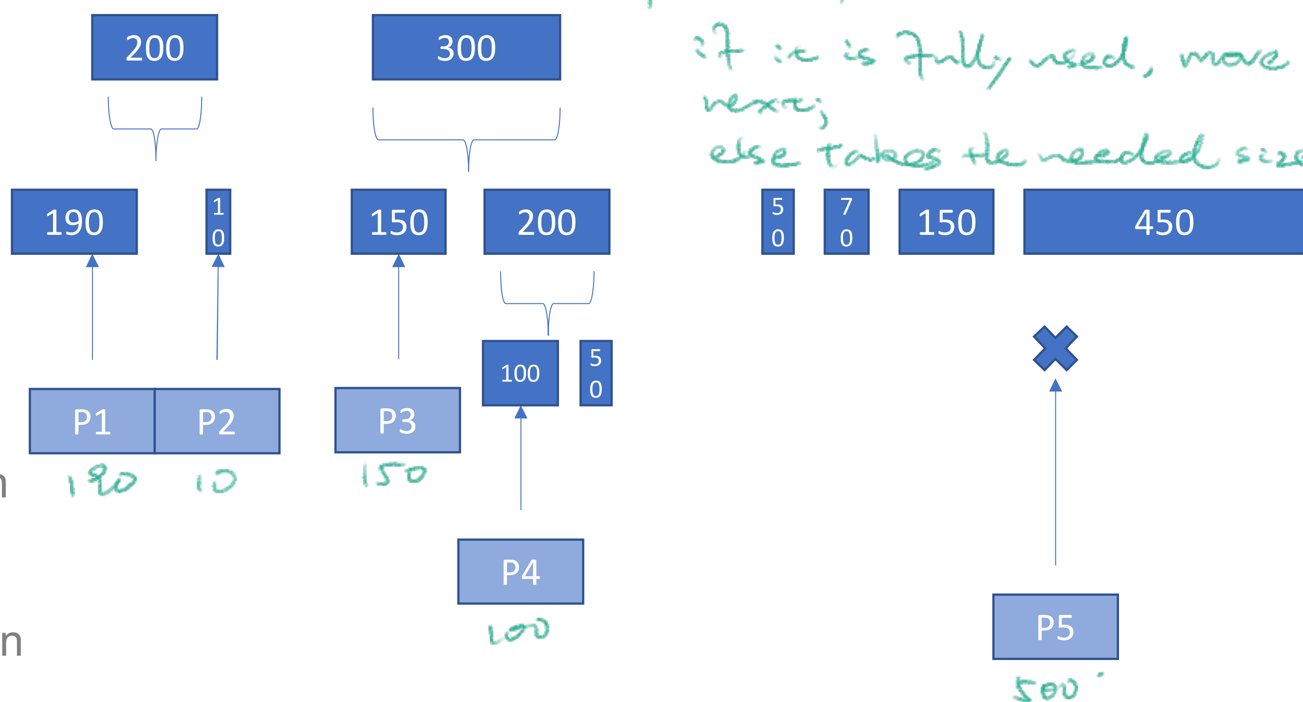
Contiguous Memory Allocation

- Each contiguous section of memory is a fixed or dynamic partition
  - Dynamic (variable) partition
    - Partitions are allocated with sizes that may change as needed
    - Empty partitions can be shifted and adjusted through **compaction**
    - Select a partition for a process by first-fit, best-fit, or worst-fit (This is okay because we use compaction) => just find some pattern, then we change partition later
    - **Internal fragmentation** is not a problem.
    - **External fragmentation** is a problem, but it is easily resolved with compaction

Contiguous Memory Allocation

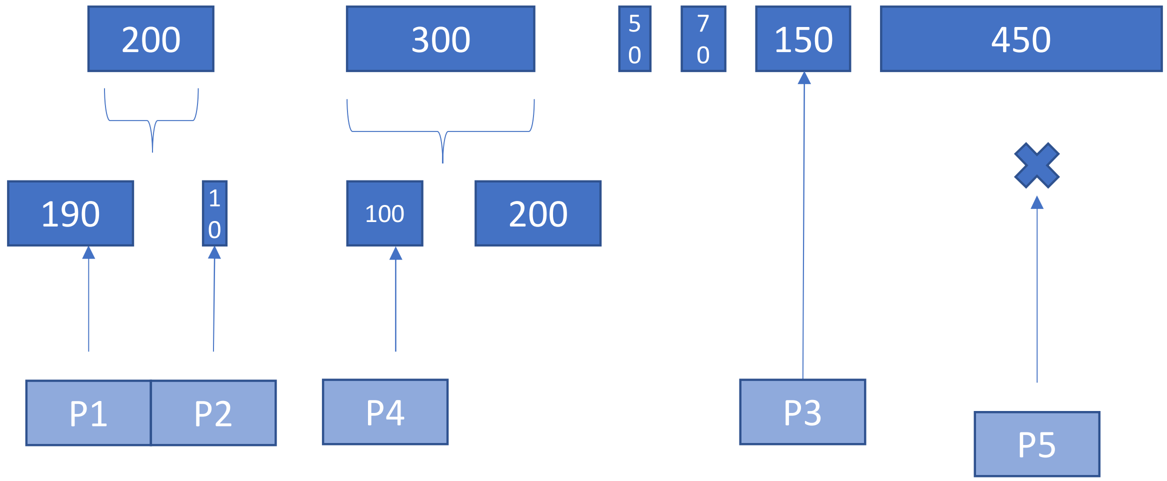
- First-fit :
    - P1 = 190
    - P2 = 10
    - P3 = 150
    - P4 = 100
    - P5 = 500
- Internal Fragmentation
    - None
- External Fragmentation
    - 50 + 70 + 150 + 450 > 500

Find a partition:
if it is fully used, move to the next;
else takes the needed size only, leave a new partition



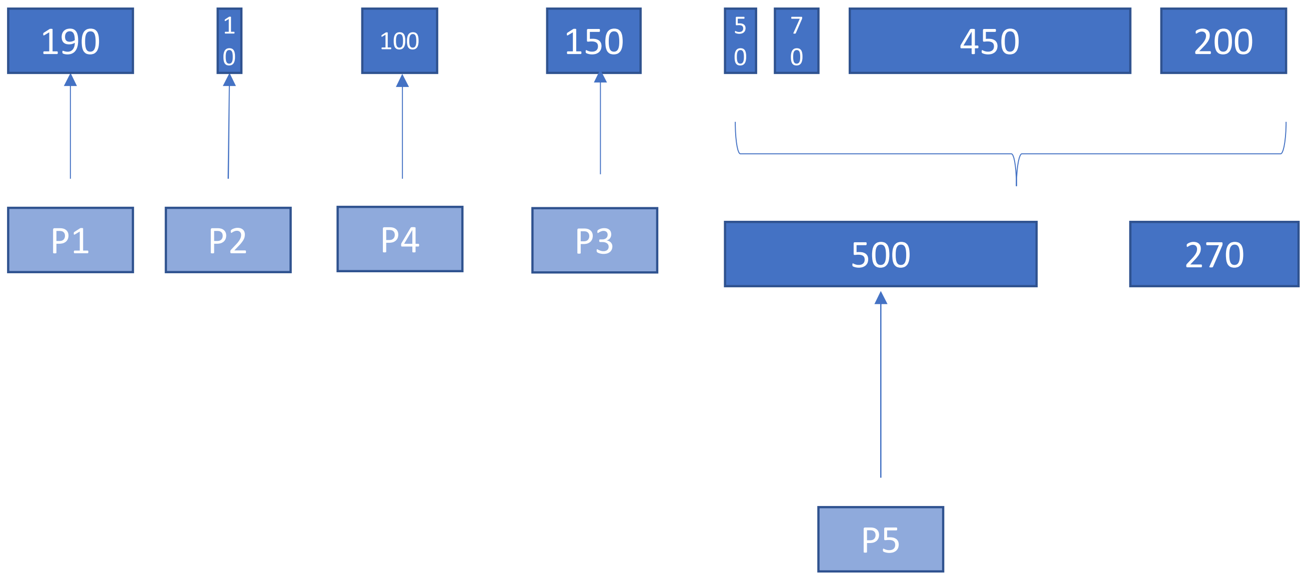
Contiguous Memory Allocation

- Best-fit
  - P1 = 190
  - P2 = 10
  - P3 = 150
  - P4 = 100
  - P5 = 500
- Internal Fragmentation
  - None
- External Fragmentation
  - 200 + 50 + 70 + 450 > 500



Contiguous Memory Allocation

- Compaction



Contiguous Memory Allocation

- Compaction
  - The compaction process is costly
  - A better solution is to permit non-contiguous memory allocation (paging)
  - Compaction is also used to manage storage so it will come up again

Western

UNIVERSITY · CANADA