Chapter VII
Memory Management
(short version)

Jehan-François Pâris
jfparis@uh.edu

Chapter Overview

- A very brief survey on how older systems managed their main memory
  - Explains why modern systems use virtual memory
- A ***shorter version*** of what is typically covered
  - Compensates for a lost week of classes

The very early computers

- ***No OS and no memory management***
- Programmers
  - Had access to whole main memory of the computer
  - Had to enter the bootstrapping routine loading their programs into main memory
    - Time-consuming and error-prone.

Uniprogramming systems

- Had a memory-resident monitor
- Invoked every time a user program would terminate
- Would immediately fetch the next program in the queue
  - ***Batch processing***

The good and the bad

- ***Advantage:***
  - No time was lost re-entering manually the bootstrapping routine
- ***Disadvantage:***
  - CPU remained idle every time the user program does an I/O.

Multiprogramming with fixed partitions

- OS dedicated multiple partitions for user processes
  - Partition boundaries were ***fixed***

The good and the bad

- ***Advantage:***
  - No CPU time is lost while system does I/O

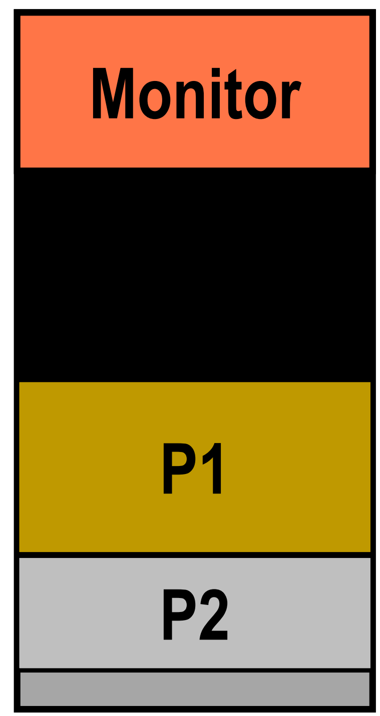
- ***Disadvantages:***
  - Partitions were ***fixed*** while processes have different memory requirements
  - Many systems required processes to occupy a ***specific partition***

Multiprogramming with variable partitions

- ***No fixed partitions***
  - Much more flexible memory allocation
- OS allocates contiguous extents of memory to processes
  - Wherever it can find available space
- Address translation mechanism lets swapped out processes return to ***any*** main memory location

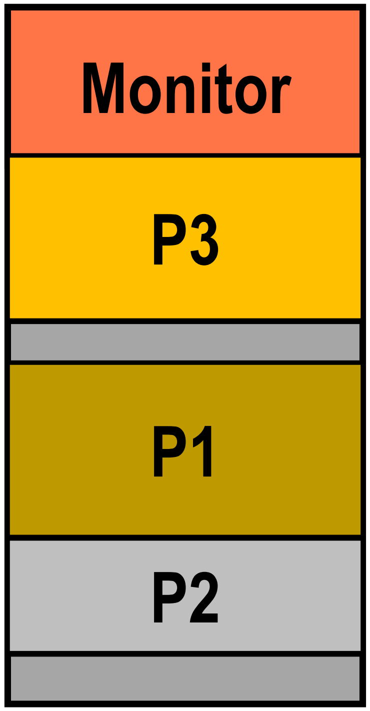
Multiprogramming with variable partitions

- Initially everything works fine
  - Three processes occupy most of memory
  - Unused part of memory is very small



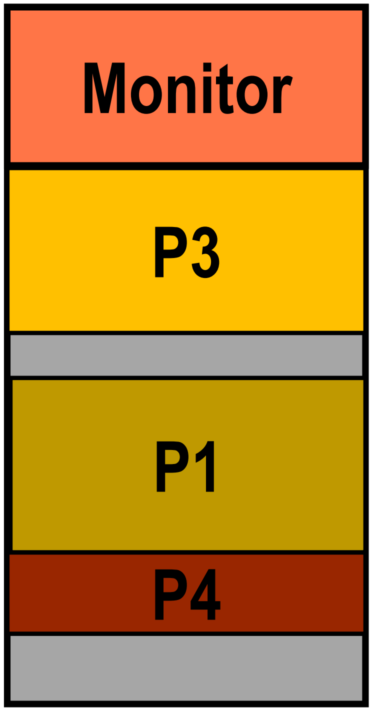
Multiprogramming with variable partitions

- When P0 terminates
  - Replaced by P3
  - P3 must be smaller than P0
- Start wasting memory space



Multiprogramming with variable partitions

- When P2 terminates
  - Replaced by P4
  - P4 must be smaller than process it replaces plus the free space
- We waste more memory space



The bad news: External fragmentation

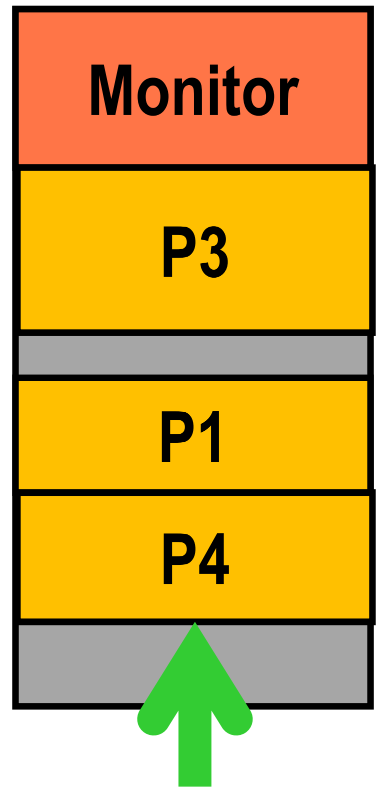
- Happens in all systems using multiprogramming with variable partitions
- Occurs because new process must fit in the hole left by terminating process
  - Typically the new process will be a bit smaller than the terminating process
  - Creates many small unusable fragments

An Analogy

- Replacing an old book by a new book on a bookshelf
- New book must fit in the hole left by old book
  - Very low probability that both books have exactly the same width
  - We will end with empty shelf space between books
- Solution it to push books left and right

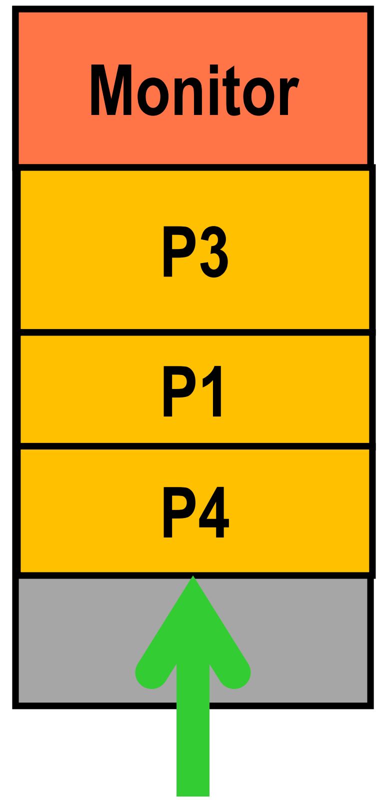
Memory compaction

- When external fragmentation becomes a problem
  - ***Push*** processes around in order to consolidate free spaces
- Worked well with ***small memory sizes***



Memory compaction

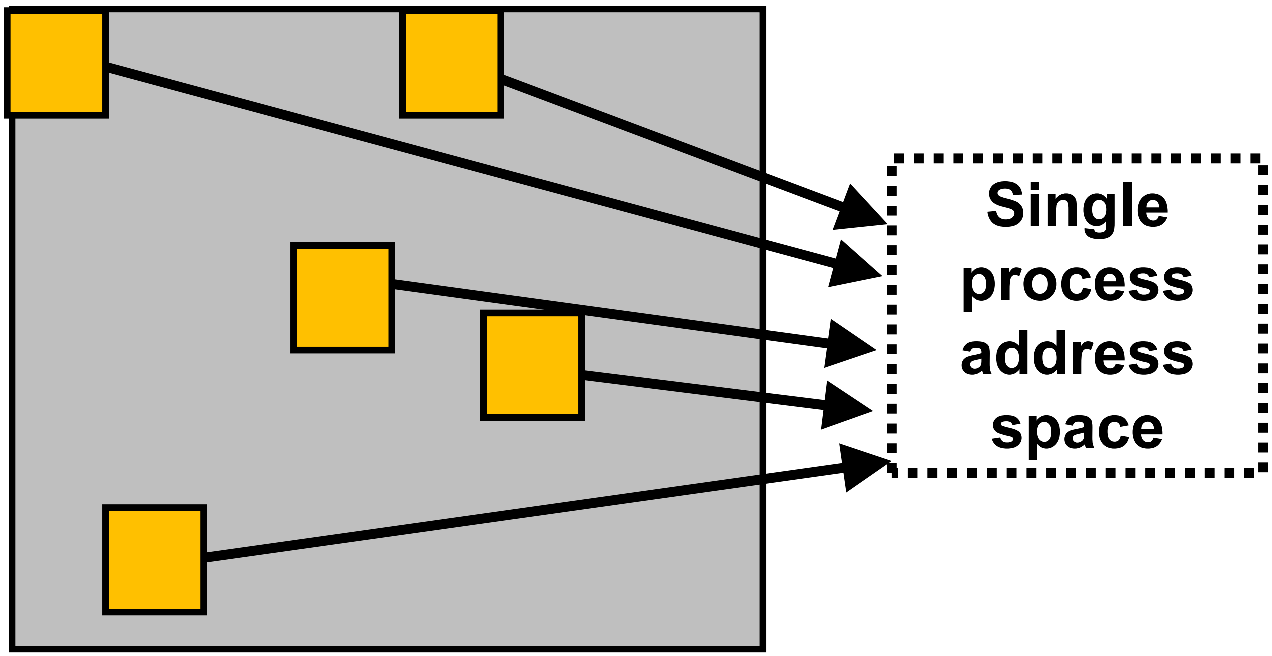
- When external fragmentation becomes a problem
  - ***Push*** processes around in order to consolidate free spaces
- Worked well with ***small memory sizes***



Non-contiguous memory allocation

- ***Non-contiguous allocation***
  - Partition physical memory into fixed-size entities
    - ***Page frames***
  - ***A***llocate non-contiguous page frames to processes
  - Let MMU handle the address translation

Non-contiguous allocation



Virtual v. real

- Processes are provided with the illusion of a vast linear address space
  - Virtual addresses starting at address zero
- In reality, this address space is made up of disjoint page frames
  - Non-contiguous real addresses