COMP30023 – Computer Systems

Operating systems: Memory Management

Olya Ohrimenko

Project 1 is out

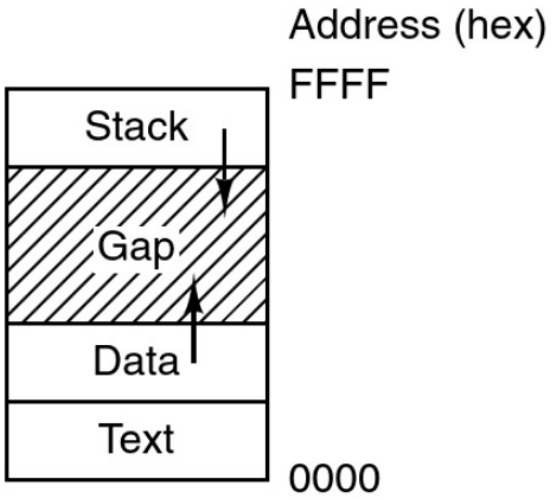
- Announcement on LMS
- Spec is available via LMS
- Extra consultation hours

Recap: Process

Process: program in execution

Process memory has three segments:

- text (program code, usually “read only”)
- data (constant data strings, global vars)
- stack (local vars)



Today

- Memory hierarchy
- Basic memory management
- Memory allocation

Memory specifications

Requirements:

- Fast
- Cheap
- Large
- Non-volatile

Reality:

- Different types of memory with different properties
- Higher speed, smaller capacity, greater cost

Memory hierarchy

Hardware includes CPU AND OTHER THINGS

cache is slower than registers because for registers, it actually on the CPU (MANAGED INSIDE THE CPU), while cache is still in hardware outside CPU and shared between CPU ???

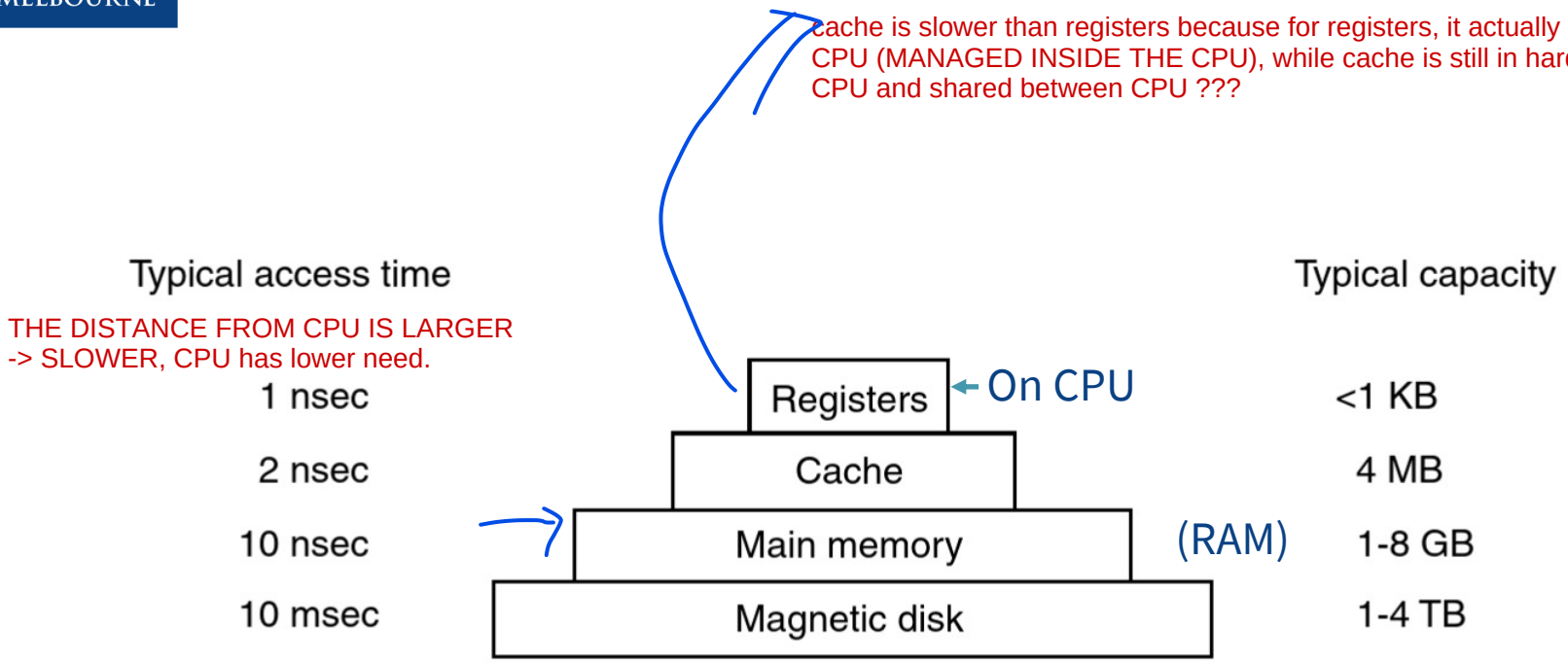


Figure 1-9. A typical memory hierarchy. The numbers are very rough approximations.

Caches

WE HAVE 3 TYPES OF CACHE: L1, L2, L3 (SIZE >>). L2 is shared between CPUS, WHILE L1 IS TOO SMALL TO BE SHARED.

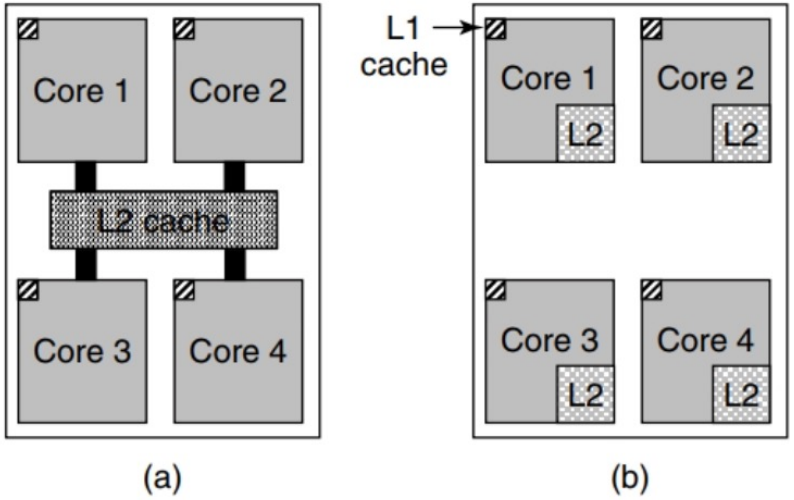


Figure 1-8. (a) A quad-core chip with a shared L2 cache. (b) A quad-core chip with separate L2 caches.

so which one controls registers ????

- Controlled by hardware
- Split in cache lines (typically, 64 byte each)
- Cache Hit : Desired data is in current level of cache
- Cache Miss : Desired data is not present in current level

Cache Decisions

1. When to put a new item into the cache.
2. Which cache line to put the new item in.
3. Which item to remove from the cache when a slot is needed.
4. Where to put a newly evicted item in the larger memory.

Memory or RAM

outside of hardware

- RAM (Random Access Memory)
- Volatile
- Megabytes to gigabytes in size

Disks

+ Cheaper than RAM per bit

+ Often 2x magnitude larger

- Slow random memory access

Memory hierarchy

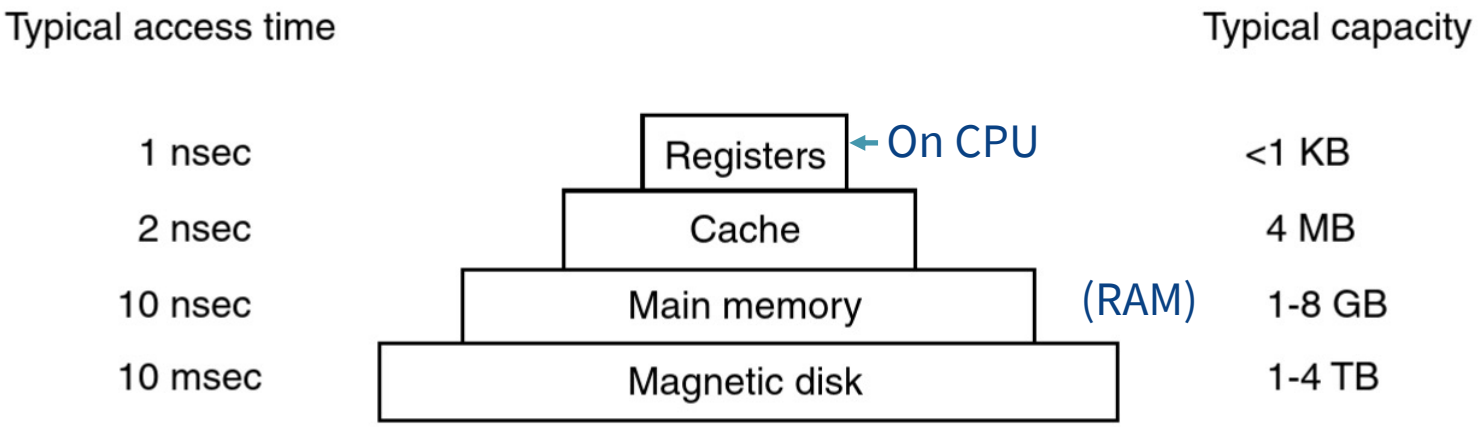
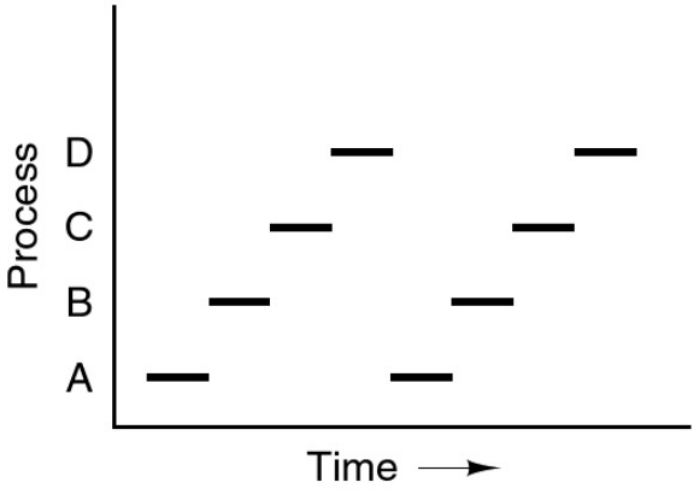
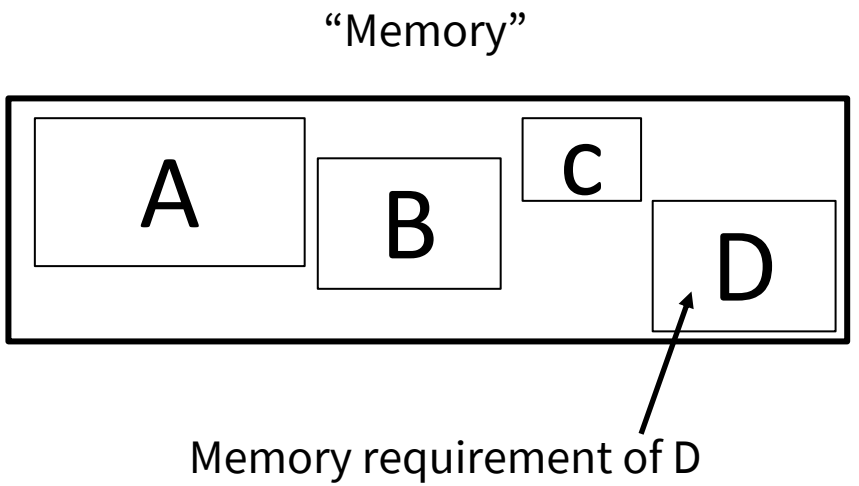


Figure 1-9. A typical memory hierarchy. The numbers are very rough approximations.

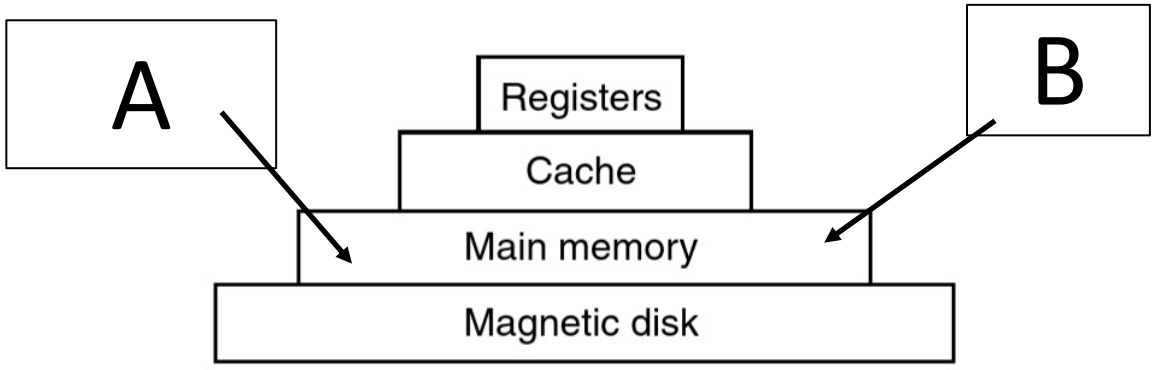
Why memory management (1)



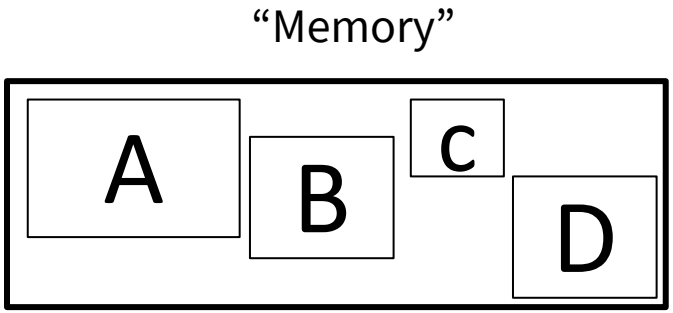


Why memory management (2)

- The sizes of main memories have been increasing dramatically, roughly quadrupling every three years.
- However, the demands for main memory have been increasing almost as fast.



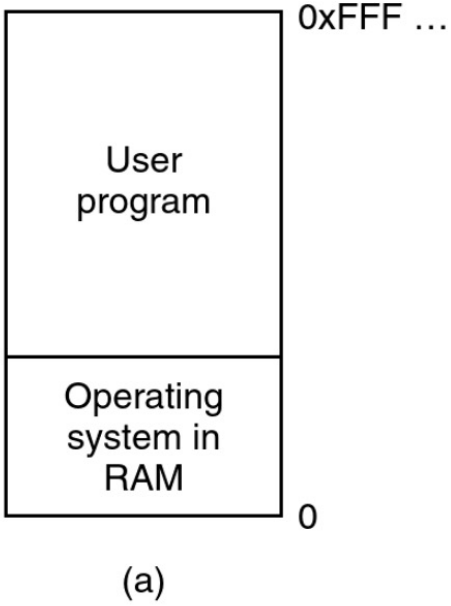
Memory manager functionality



- to allocate memory to processes when they require it
- to deallocate when finished
- to protect memory against unauthorized accesses, and
- to simulate the appearance of a bigger main memory by moving data automatically between main memory and disk
- to keep track of which parts of memory are free and which parts are allocated (and to which process)

What if no memory abstraction

- Absolute memory
- Relocate
- Expose physical memory to processes
- Consequences: security and multiple processes
- Q: How to ensure mutually exclusive access?



(a)

TB 3-1

No memory abstraction: multiple processes

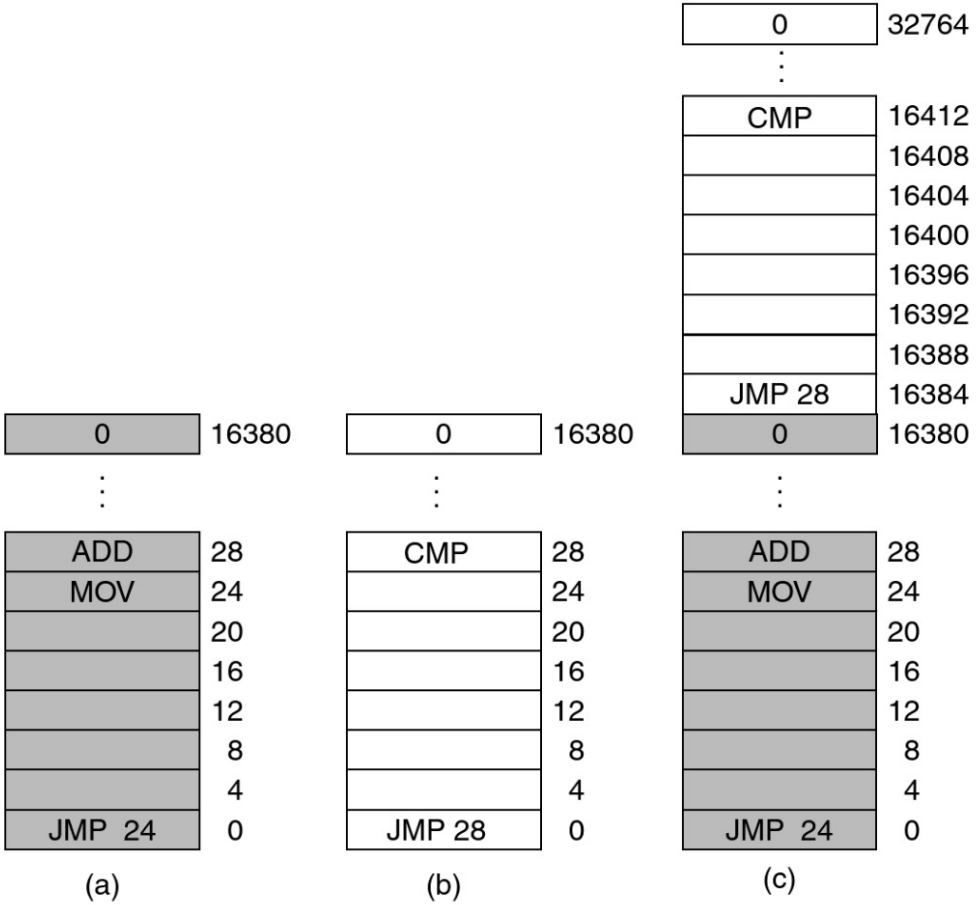


Figure 3-2. Illustration of the relocation problem. (a) A 16-KB program. (b) Another 16-KB program. (c) The two programs loaded consecutively into memory.

No memory abstraction: multiple processes

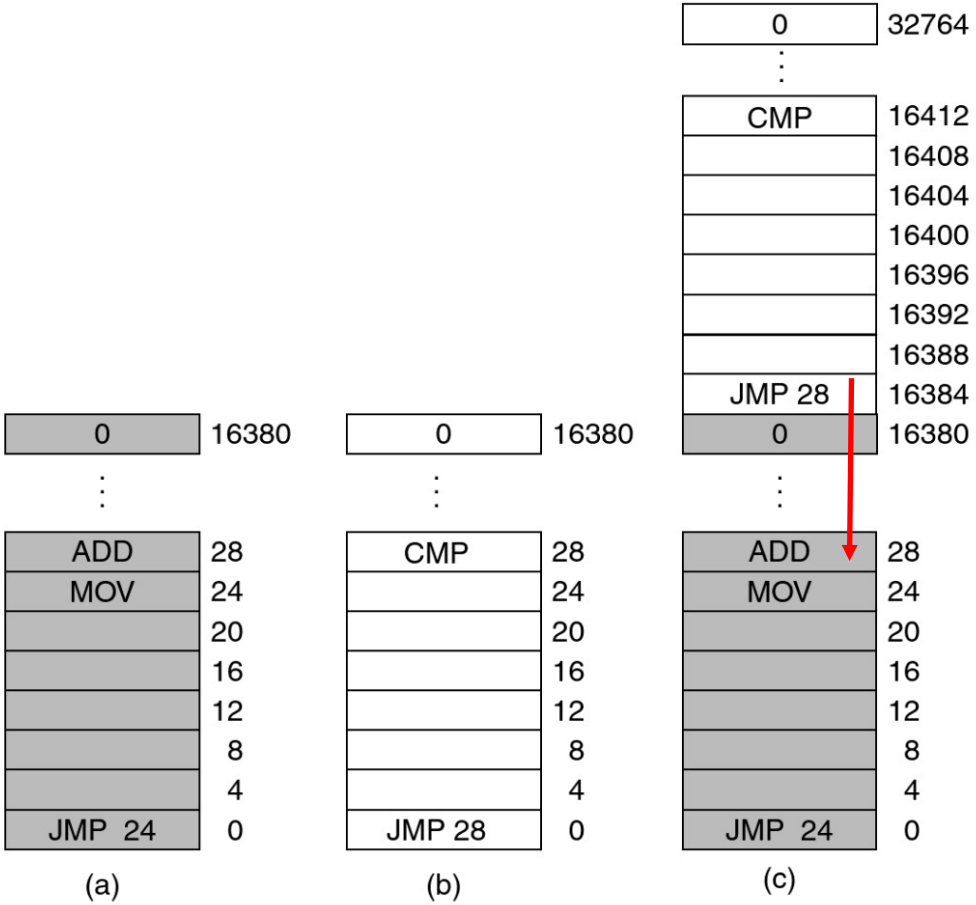


Figure 3-2. Illustration of the relocation problem. (a) A 16-KB program. (b) Another 16-KB program. (c) The two programs loaded consecutively into memory.

No memory abstraction: multiple processes

Base register

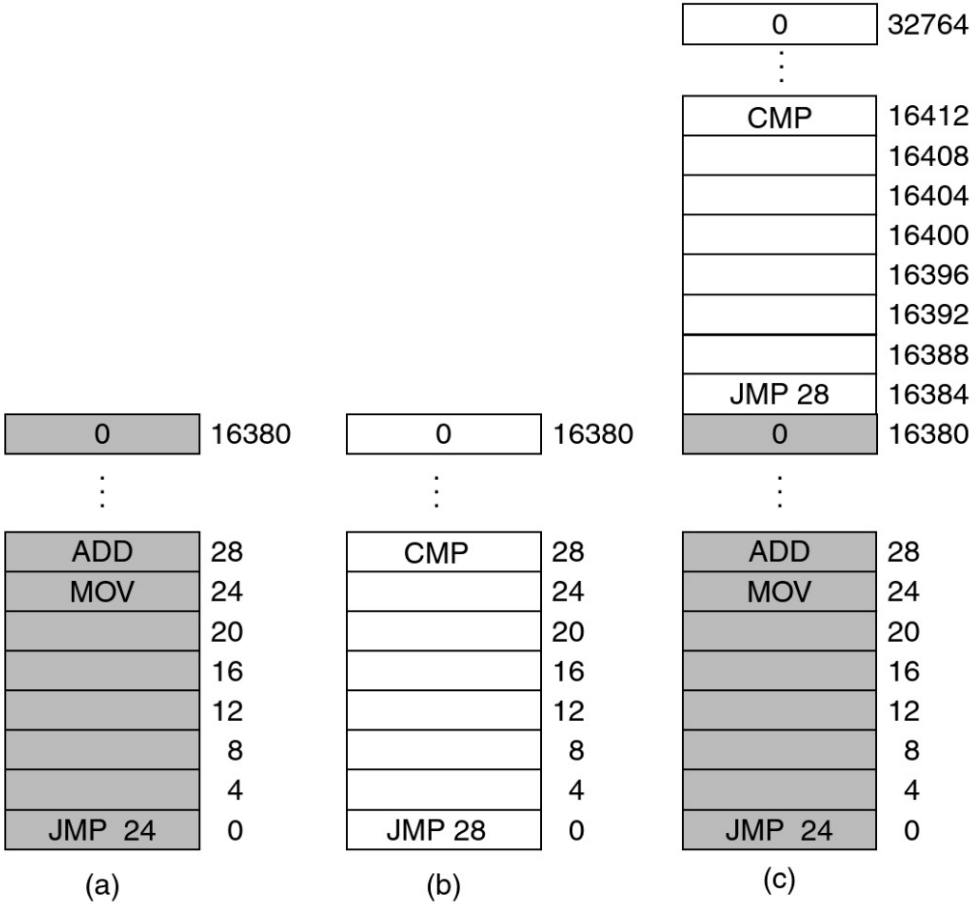


Figure 3-2. Illustration of the relocation problem. (a) A 16-KB program. (b) Another 16-KB program. (c) The two programs loaded consecutively into memory.

No memory abstraction: multiple processes

Base register

Limit register

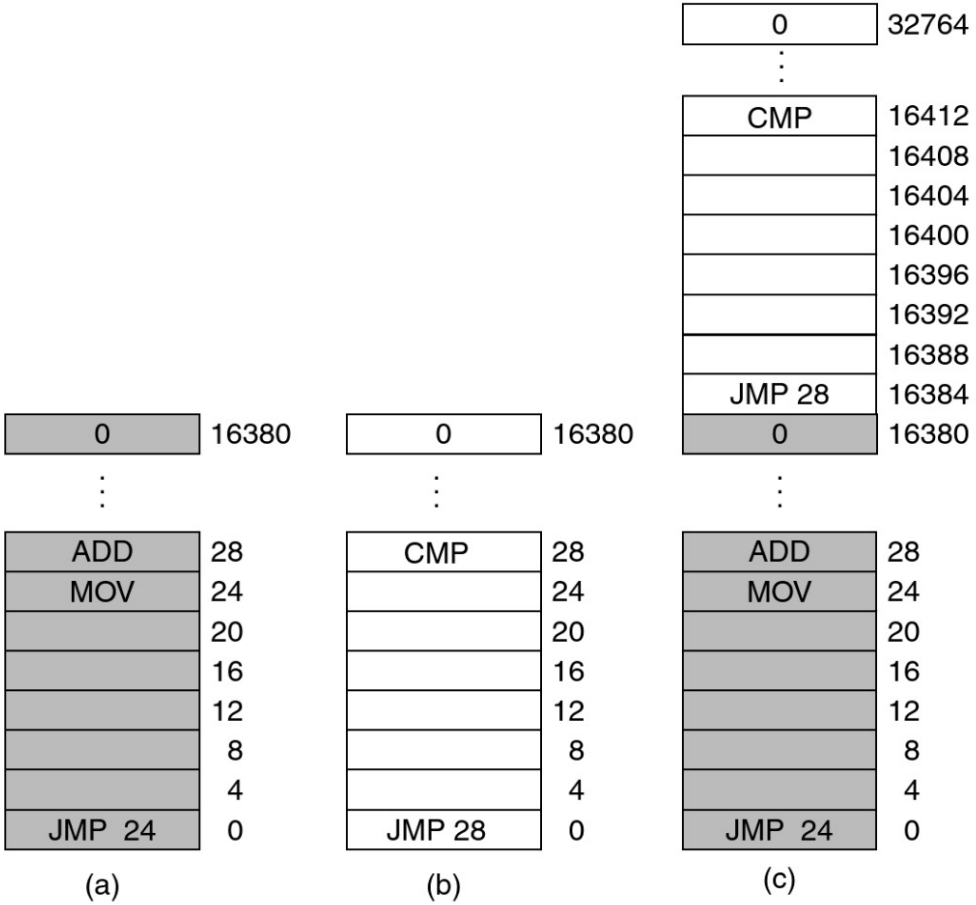
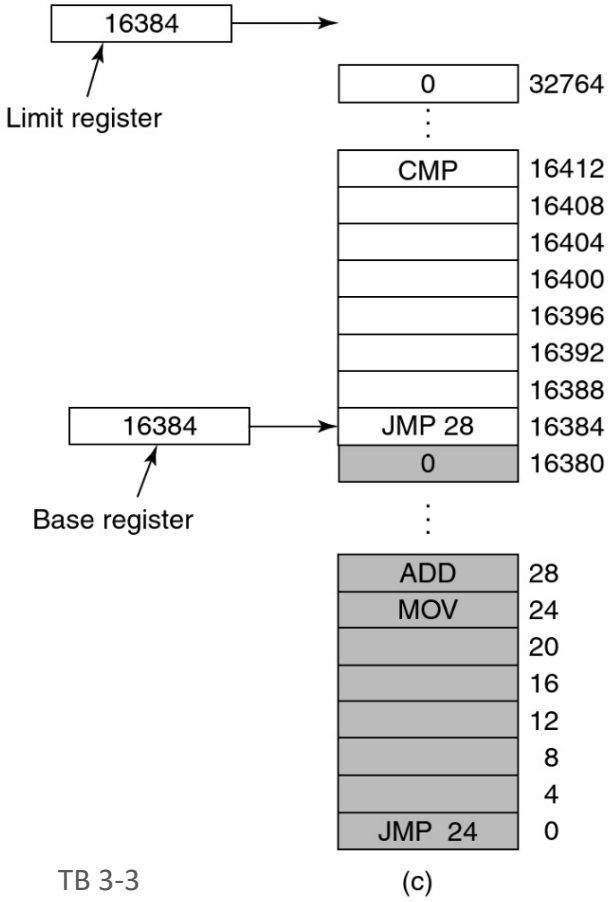


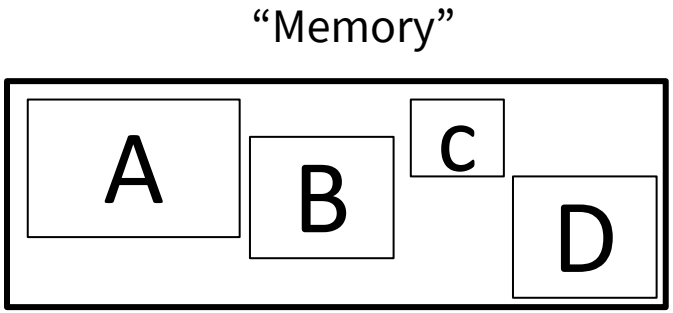
Figure 3-2. Illustration of the relocation problem. (a) A 16-KB program. (b) Another 16-KB program. (c) The two programs loaded consecutively into memory.

Base and Limit Registers

Base and limit registers can be used to give each process a separate address space.

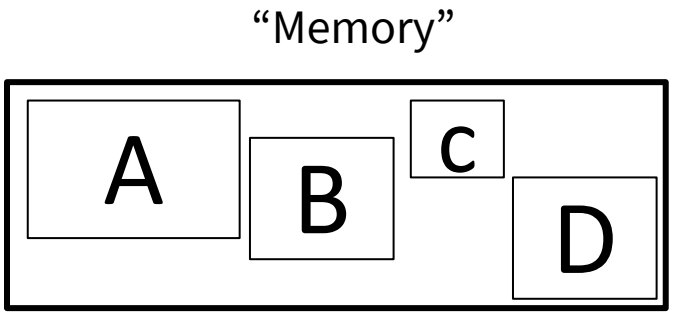


Memory manager functionality



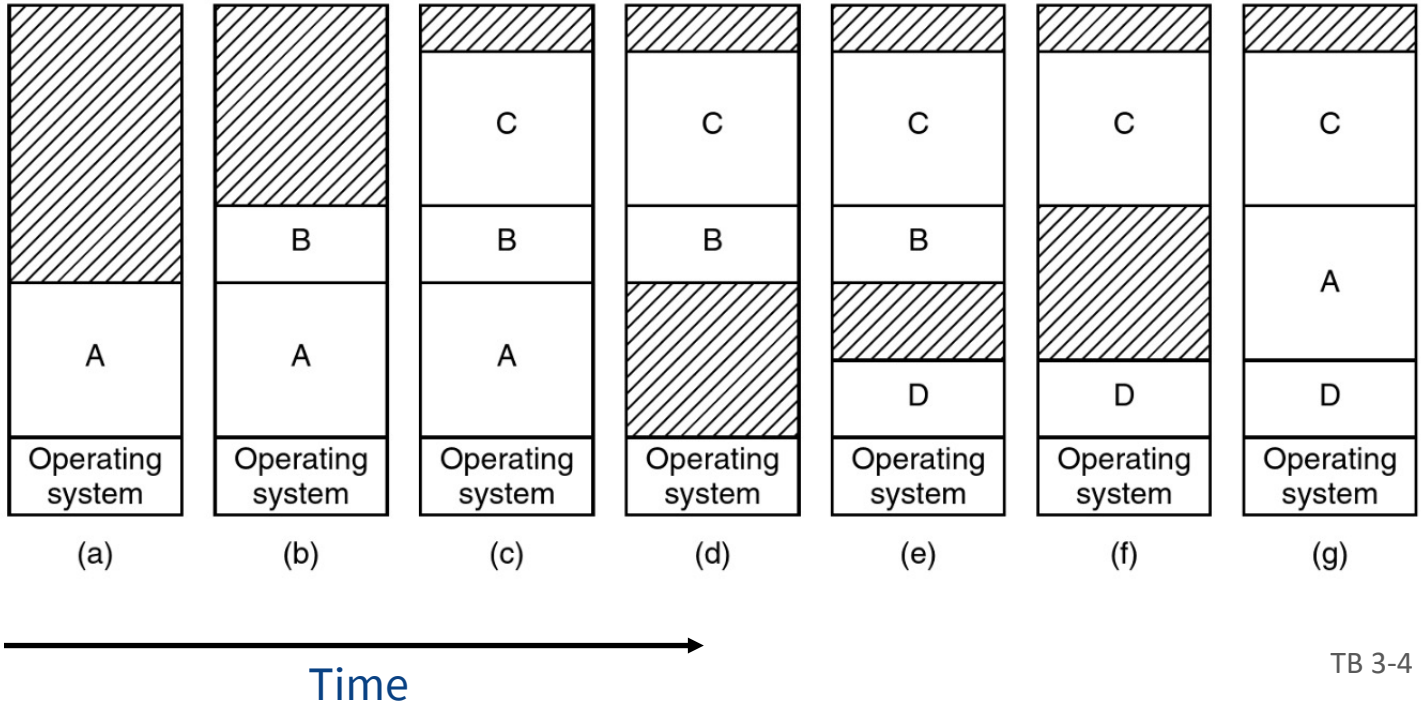
- to allocate memory to processes when they require it
- to deallocate when finished
- to protect memory against unauthorized accesses, and
- to simulate the appearance of a bigger main memory by moving data automatically between main memory and disk
- to keep track of which parts of memory are free and which parts are allocated (and to which process)

Memory manager functionality



- to allocate memory to processes when they require it
- to deallocate when finished
- to protect memory against unauthorized accesses, and
- to simulate the appearance of a bigger main memory by moving data automatically between main memory and disk
- to keep track of which parts of memory are free and which parts are allocated (and to which process)

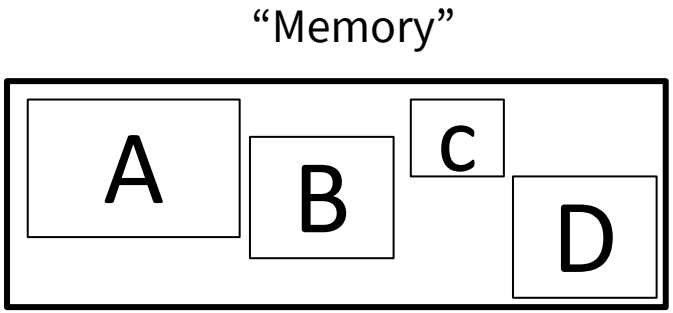
Swapping



TB 3-4

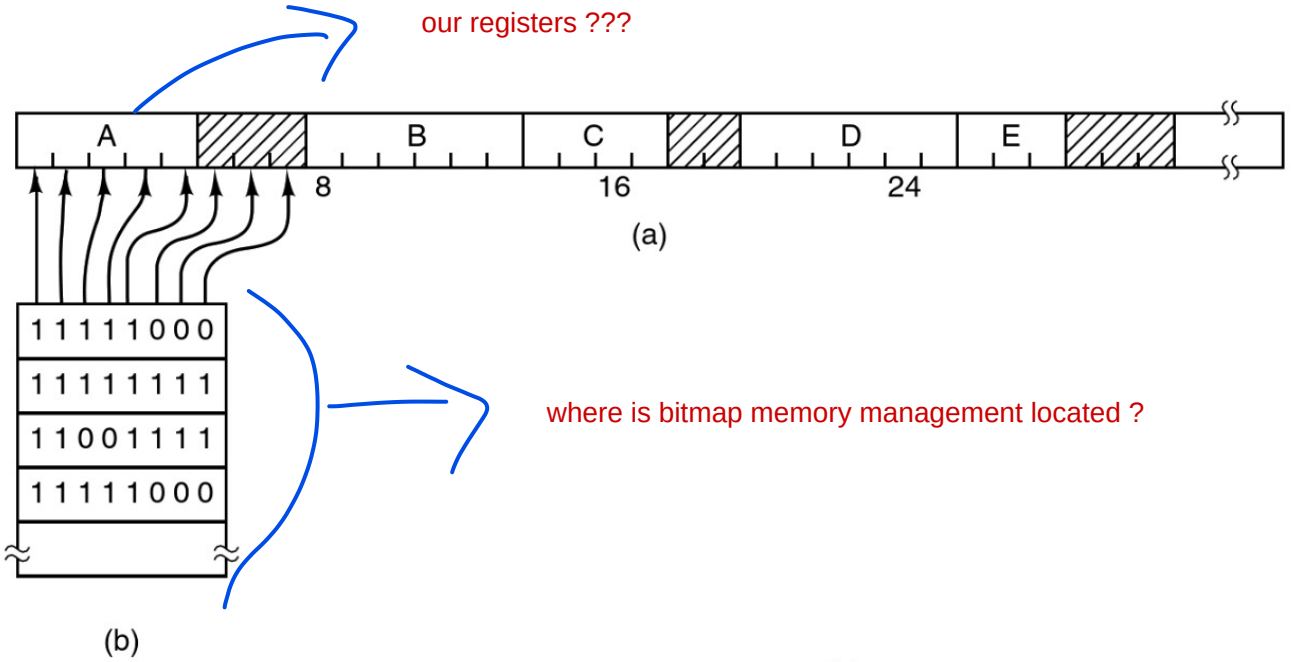
Memory allocation changes as processes come into memory and leave it. The shaded regions are unused memory

Memory manager functionality



- to allocate memory to processes when they require it
- to deallocate when finished
- to protect memory against unauthorized accesses, and
- to simulate the appearance of a bigger main memory by moving data automatically between main memory and disk
- to keep track of which parts of memory are free and which parts are allocated (and to which process)

Memory Management with Bitmaps

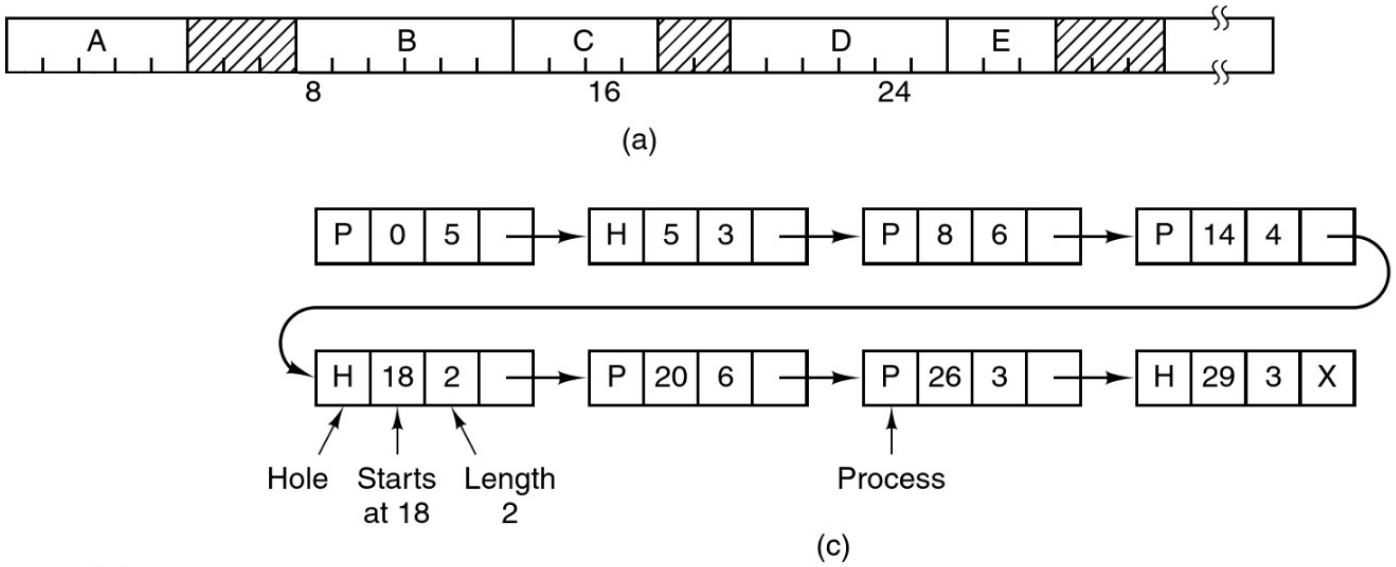


TB 3-6

(a) A part of memory with five processes and three holes. The tickmarks show the memory allocation units. The shaded regions (0 in the bitmap) are free

(b) The corresponding bitmap

Memory Management with Bitmaps



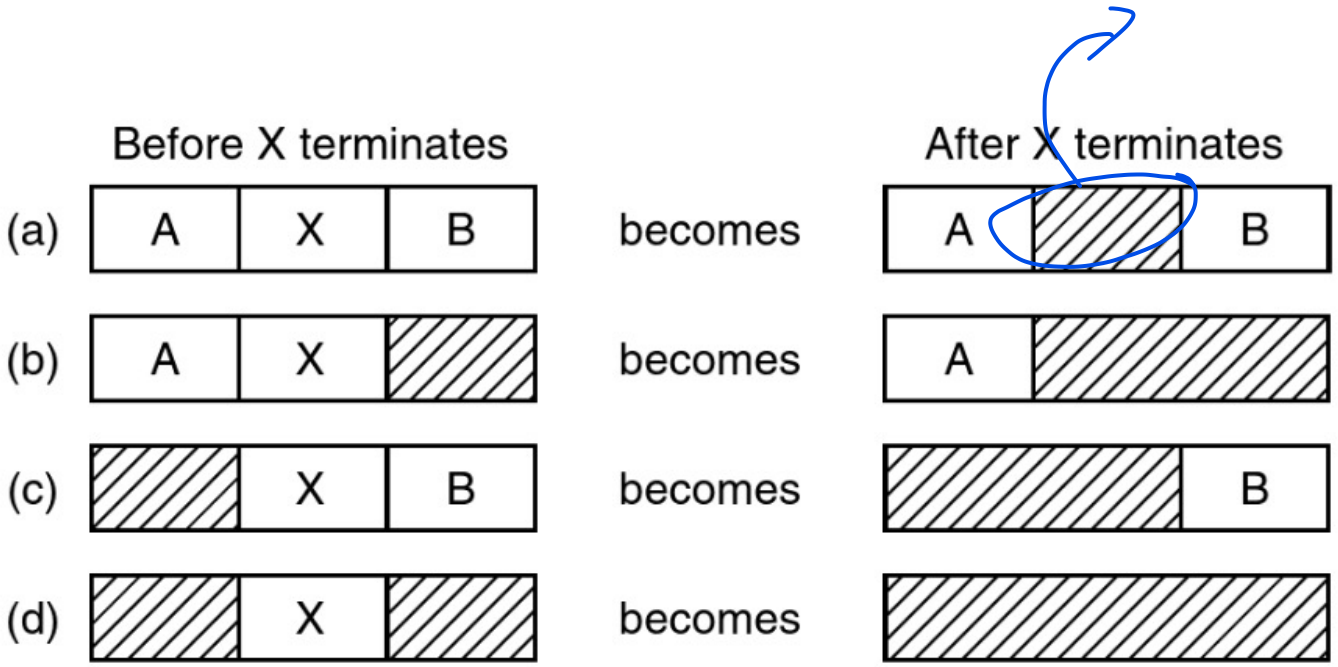
TB 3-6

(a) A part of memory with five processes and three holes. The tickmarks show the memory allocation units. The shaded regions (0 in the bitmap) are free

(c) The same information as a list.

Memory Management with Linked Lists

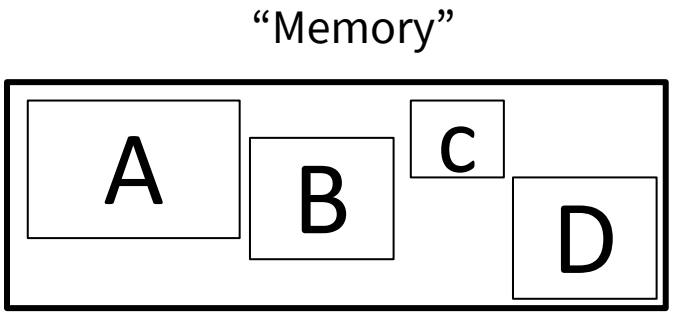
make X empty.



TB 3-7

Four neighbor combinations for the terminating process, *X*.

Memory manager functionality

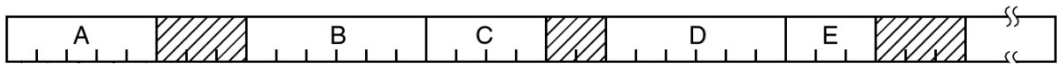


- to allocate memory to processes when they require it
- to deallocate when finished
- to protect memory against unauthorized accesses, and
- to simulate the appearance of a bigger main memory by moving data automatically between main memory and disk
- to keep track of which parts of memory are free and which parts are allocated (and to which process)

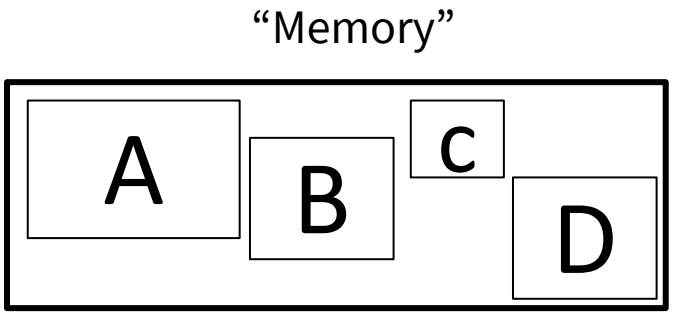
Memory Management Algorithms

- First fit
- Next fit
- Best fit
- Worst fit
- Quick fit

Managing free memory:
How to choose the next free block?

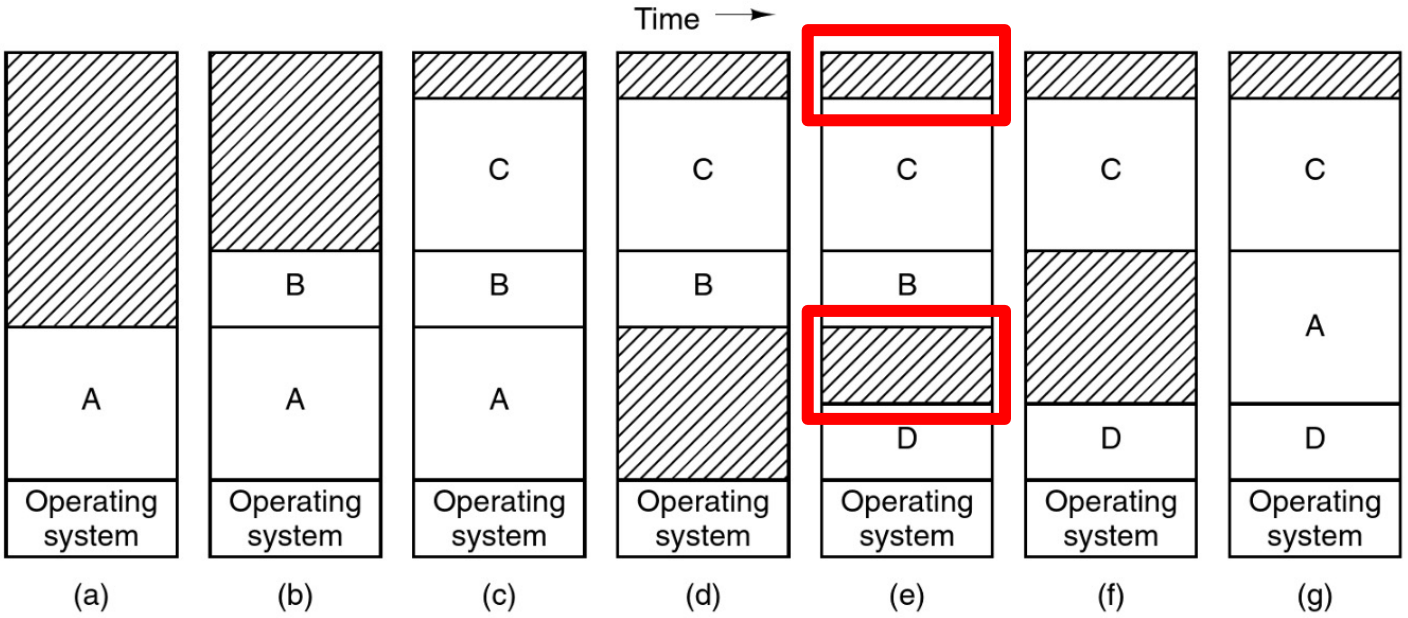


Memory manager functionality



- to allocate memory to processes when they require it
- to deallocate when finished
- to protect memory against unauthorized accesses, and
- to simulate the appearance of a bigger main memory by moving data automatically between main memory and disk
- to keep track of which parts of memory are free and which parts are allocated (and to which process)

Swapping & Fragmentation



External fragmentation

Summary

- Memory hierarchy
- Memory allocation

Project 1 is out

- Announcement on LMS
- Spec is available via LMS
- Extra consultation hours

Acknowledgement

- The slides were prepared by Olya Ohrimenko.
- Some material is borrowed from slides developed by Lachlan Andrew, Chris Culnane, Michael Kirley, Zoltan Somogyi, Rao Kotagiri, James Bailey and Chris Leckie.
- Some of the images included in the notes were supplied as part of the teaching resources accompanying the text books listed in lecture 1.
- Reference: TB 1.3.2, 3, 3.1, 3.2