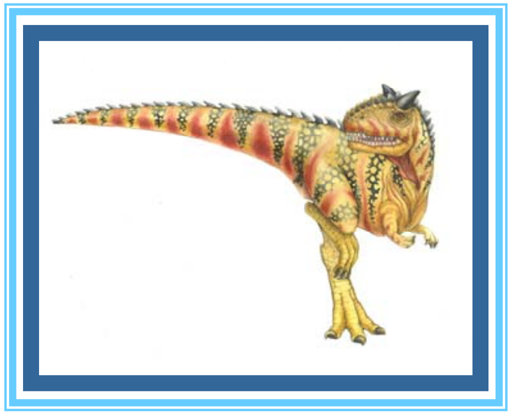
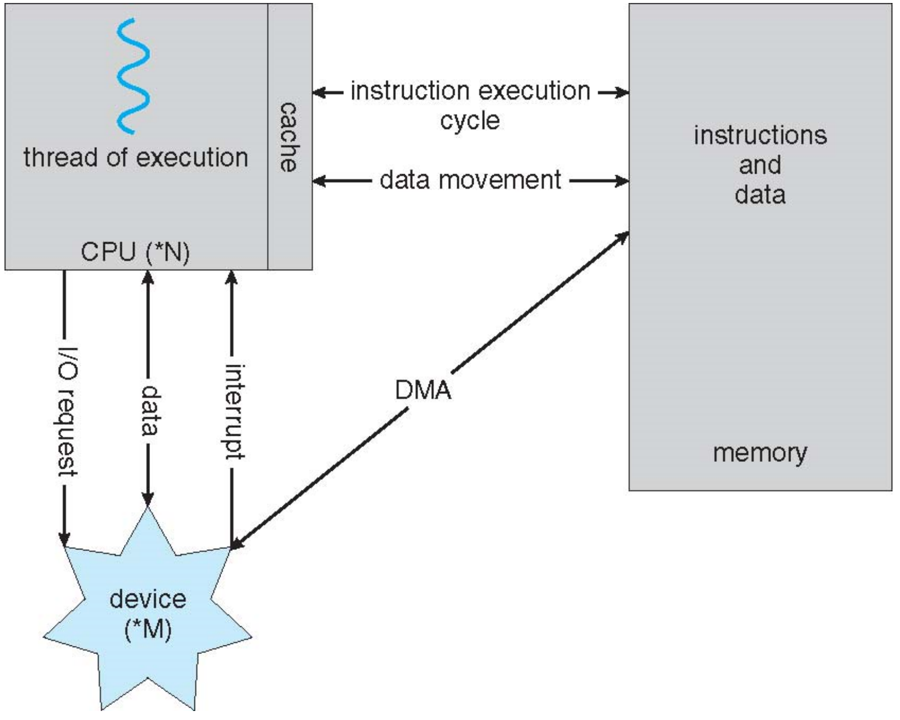
Chapter 9: Main Memory



Ch.1: How a Modern Computer Works

A von Neumann architecture and a depiction of the interplay of all components of a computer system:

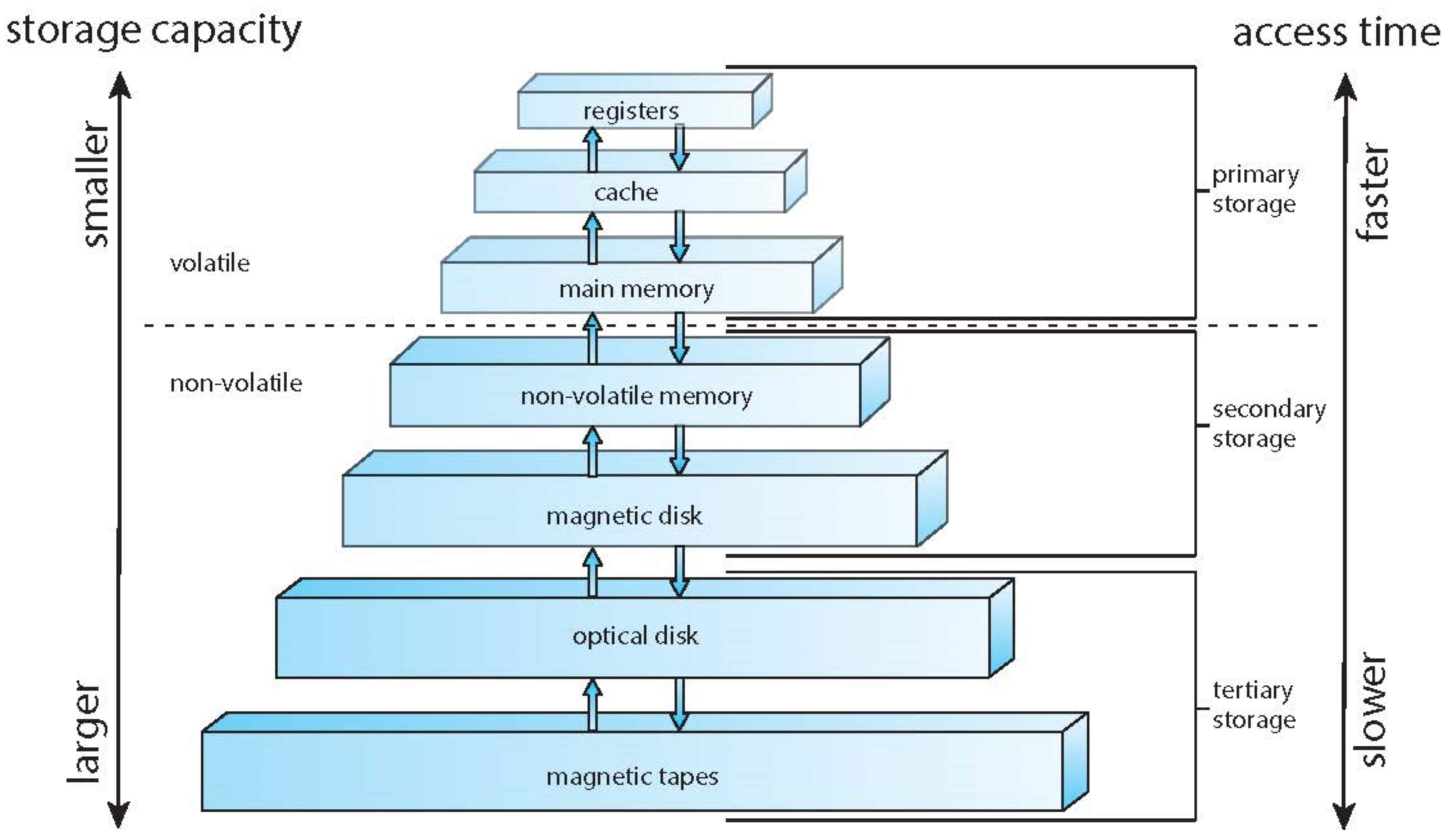


Q: Where can be the system bottleneck?

- CPU?
- Memory access?
- Disk access?
- External input (User input, network connection, etc.)?

- Depends on the application
  - *Q: For each system component above, describe an application/program that would create a bottleneck*

Ch.1: Storage-device hierarchy



Ch.1: Performance of Various Levels of Storage

Level	1	2	3	4	5
Name	registers	cache	main memory	solid state disk	magnetic disk
Typical size	< 1 KB	< 16MB	< 64GB	< 1 TB	< 10 TB
Implementation technology	custom memory with multiple ports CMOS	on-chip or off-chip CMOS SRAM	CMOS SRAM	flash memory	magnetic disk
Access time (ns)	0.25 - 0.5	0.5 - 25	80 - 250	25,000 - 50,000	5,000,000
Bandwidth (MB/sec)	20,000 - 100,000	5,000 - 10,000	1,000 - 5,000	500	20 - 150
Managed by	compiler	hardware	operating system	operating system	operating system
Backed by	cache	main memory	disk	disk	disk or tape

Movement between levels of storage hierarchy can be explicit or implicit

Duties of a Memory Management System

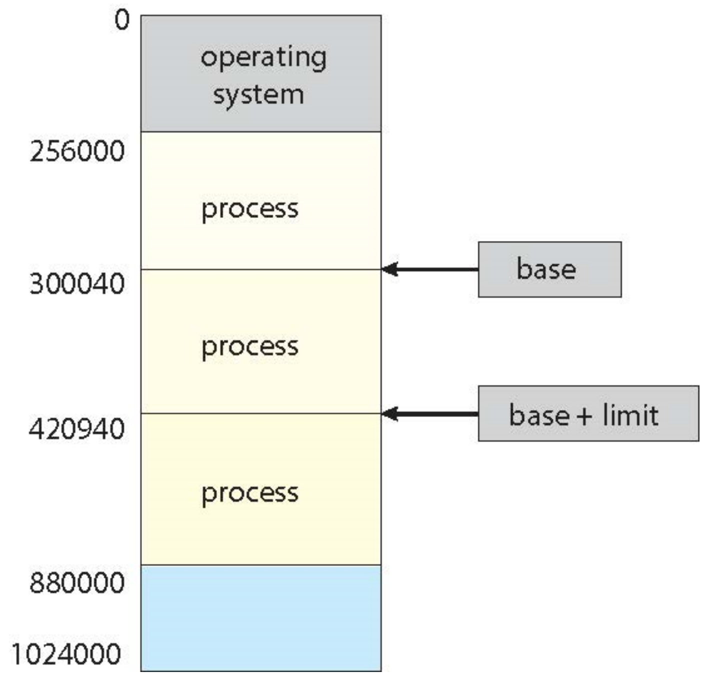
- Define an address representation that can be translated between CPU, memory, and program (the code)
- Memory access protection
- Efficient handling of memory
  - Efficient algorithms to choose the best memory portion(s) for a given demand
  - Reduce unused part of memory
- Decide which process to add or remove from memory
- …

Background

- Program must be brought (from disk) into memory and placed within a process for it to be run
- Memory unit only sees a stream of:
  - addresses + read requests, or
  - address + data and write requests
- Memory unit does not know how these addresses were generated
  - Address representation (used by CPU, memory, and processes) should be defined clearly
- Access to the main memory can take many cycles, causing a CPU **stall**
  - Register access is done in one CPU clock
  - **Cache** sits between main memory and CPU registers
- Protection of memory required to ensure correct operation

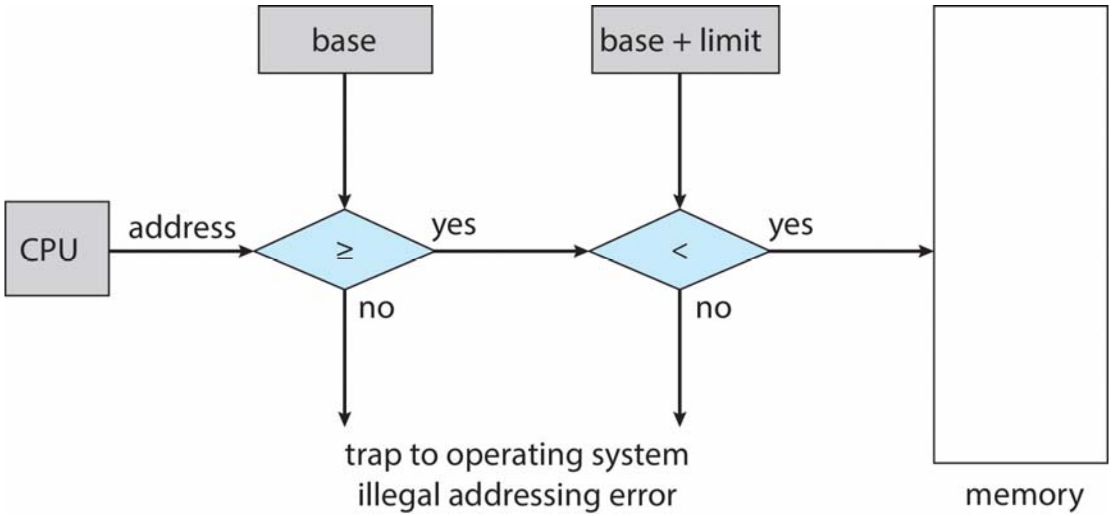
Memory Protection

- Q: Protect which part of memory from whom?
  - Need to ensure that a **user process** can only access to its address (memory) space
- We can provide this protection by using a pair of **base** and **limit registers** that define the logical address space of a process
- OS has access to:
  - these two registers
  - user process memory
    - Q: Why?
  - and, of course, OS memory



Hardware Address Protection

- CPU must check every memory access generated in user mode to be sure it is between base and limit for that user



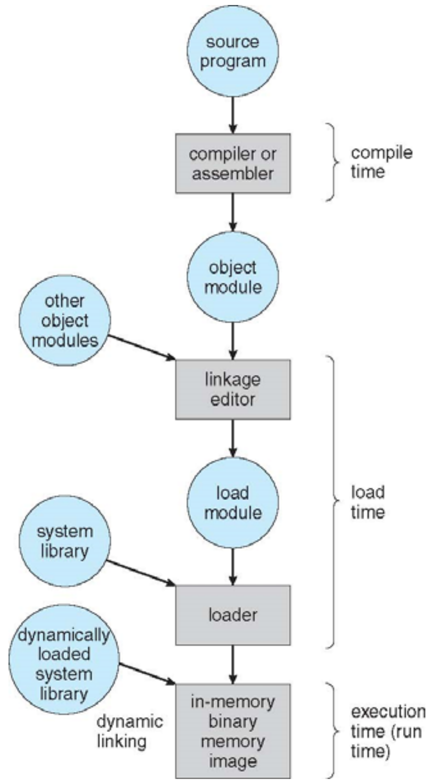
- the instructions to loading the base and limit registers are privileged

Address Binding

- Programs on disk, ready to be brought into memory to execute form an **input queue**
- Addresses represented in different ways at different stages of a program's life
- An example:
  - Source code addresses are usually ***symbolic***
    - e.g., variables: "int i", "int *j"
  - Compiler **binds** them to ***relocatable addresses***
    - e.g. "14 bytes from beginning of this module"
  - Linker or loader will bind relocatable addresses to ***absolute addresses***
    - e.g. 74014
  - Each binding maps one address space to another

Address Binding Options

- Binding of instructions and data to absolute memory addresses can happen at different stages:
  - **Compile time**: If memory location known a priori, **absolute code** can be generated
    - must recompile the code if the starting location changes
    - MS-DOS was using this method for .COM executables
      - loaded at a pre-set address: at offset 0100h
  - **Load time**: if memory location is not known at compile time, compiler must generate **relocatable code,** the absolute address can be generated at load time
    - If initial address changes, load again
  - **Execution time**: Binding delayed until run time if the process can be moved during its execution from one memory segment to another
    - Need hardware support for address maps (e.g., base and limit registers)
  - Q: Why would one need Execution Time binding?

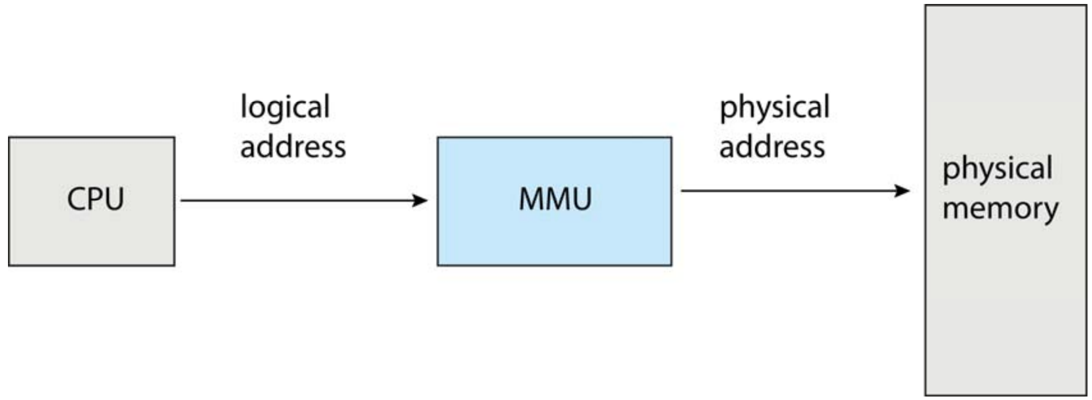


Logical vs. Physical Address Space

- The concept of a logical address space that is bound to a separate **physical address space** is central to proper memory management
  - **Logical address** – used by the CPU; also referred to as **virtual address**
  - **Physical address** –used by the memory
- Logical and physical addresses are the same in compile-time and load-time address-binding schemes. Q: Why?
- Logical (virtual) and physical addresses differ in execution-time address-binding scheme
- **Logical address space** is the set of all logical addresses used by a program
- **Physical address space** is the set of all physical addresses used by a program

Memory-Management Unit (MMU)

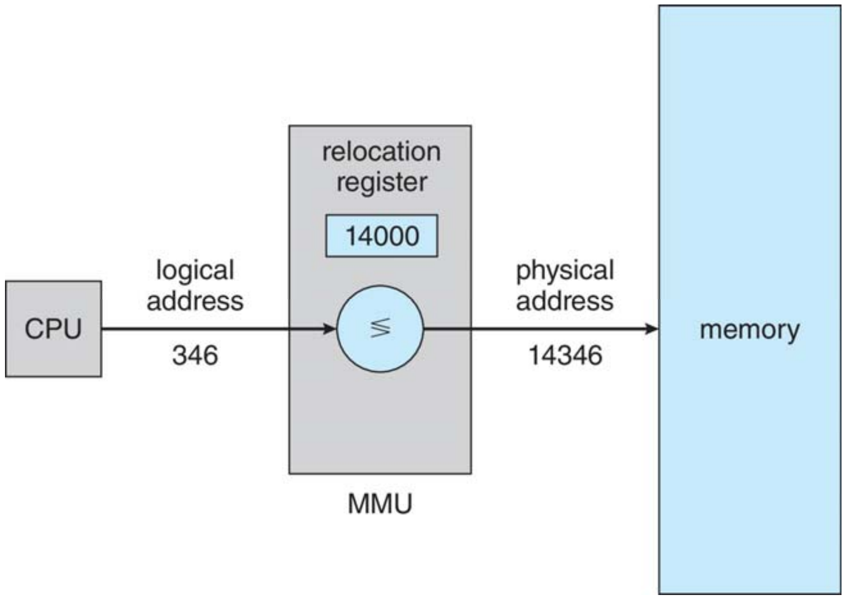
- Hardware device that at run time maps virtual to physical address



- The user program deals with *logical* addresses; it never sees the *real* physical addresses
  - Execution-time binding occurs when reference is made to location in memory
- Many methods possible, covered in the rest of this chapter

Memory-Management Unit (Simple Sol'n)

- Consider simple scheme, which is a generalization of the *base-register* scheme
- The base register now called **relocation register**
- The value in the relocation register is added to every address generated by a user process at the time it is sent to memory



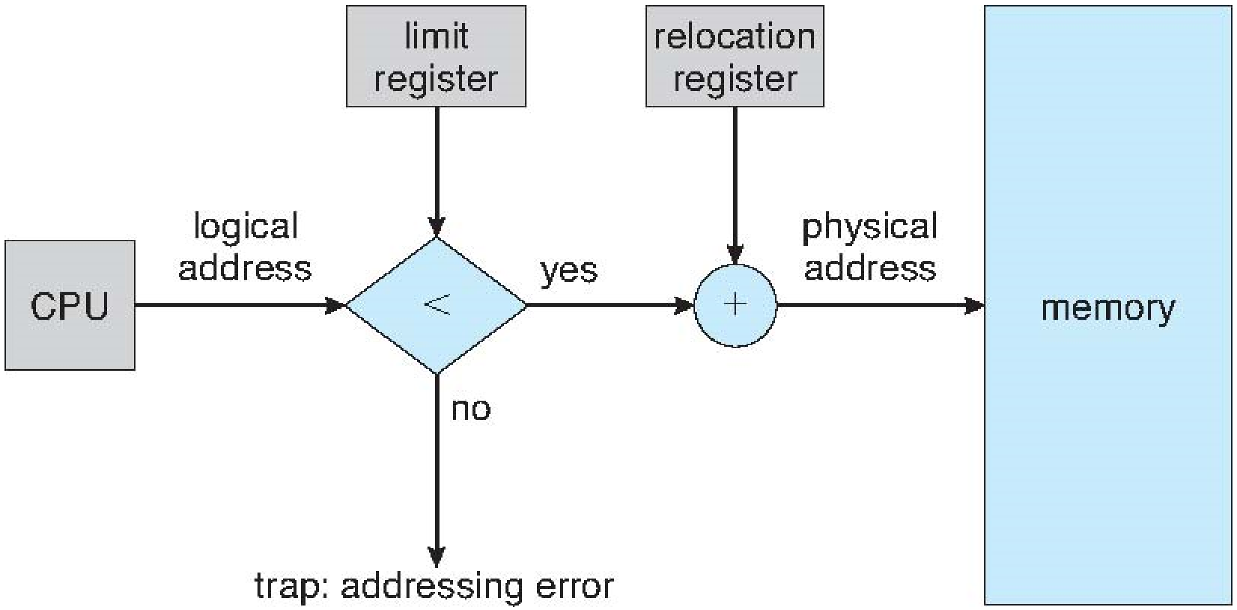
Contiguous Allocation

- Main memory must support both OS and user processes
- Limited resource, must allocate efficiently
- Contiguous allocation is one early method
- Main memory usually into two **partitions**:
  - Resident operating system, usually held in low memory
  - User processes then held in high memory
  - Each process contained in single contiguous section of memory

Contiguous Allocation (Cont.)

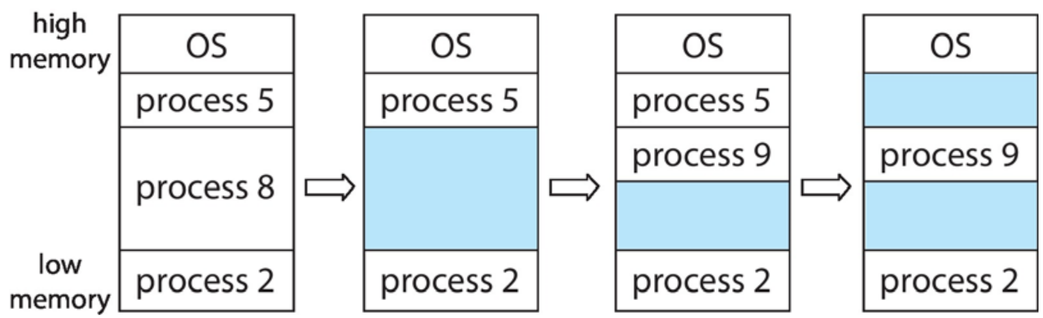
- Relocation registers enable protecting the user processes from each other, and from changing operating-system code and data
  - Relocation register contains the smallest ________ address
  - Limit register contains the upper limit of ________ addresses
  - At context switch, the dispatcher loads the relocation and limit registers with the correct values
  - Allows actions such as kernel code being **transient** and kernel changing size
    - e.g., by removing from the memory the code and data related to a device or service that is not being used

Hardware Support for Relocation and Limit Registers



Memory Partitioning

- Simple approach: Fixed size partitions
  - Q: Disadvantages?
    - Degree of multiprogramming limited by number of partitions
    - Inefficient use of space
- Modern alternative:
  - **Variable-partition** sizes for efficiency (sized to a given process' needs)
    - When a process arrives, it is allocated memory from a **hole** large enough to accommodate it
    - Process exiting frees its partition, adjacent free partitions combined
    - Operating system maintains information about:
      a) allocated partitions   b) free partitions (hole)



Dynamic Storage-Allocation Problem

Q. How to satisfy a request of size ***n*** from a list of free holes?

- **First-fit**: Allocate the ***first*** hole that is big enough
- **Best-fit**: Allocate the ***smallest*** hole that is big enough; must search entire list, unless ordered by size
  - Produces the smallest leftover hole
- **Worst-fit**: Allocate the ***largest*** hole; must also search entire list
  - Produces the largest leftover hole
- **Quick-fit**: Keep a list of holes grouped by sizes, e.g., 4K, 8K, etc.
  - Fast allocation
  - Size changes (e.g. merging) are costly to manage

Statistically, first-fit and best-fit is better than worst-fit in terms of storage utilization

Fragmentation

- **External Fragmentation**
  - total memory space exists to satisfy a request, but it is not contiguous
- **Internal Fragmentation**
  - allocated memory may be slightly larger than requested memory;
    - e.g., process requires 18462 bytes, but we have a hole of 18464 bytes
  - management of the small (e.g. 2 bytes) holes is more costly than using them
  - so, give the whole space to the process
- First fit analysis reveals that given *N* blocks allocated, 0.5 *N* blocks lost to fragmentation
  - 1/3 may be unusable -> **50-percent rule**