Operating System
Session 6

Hakim Sabzevari University
Dr.Malekzadeh

Memory Management

Resource Management

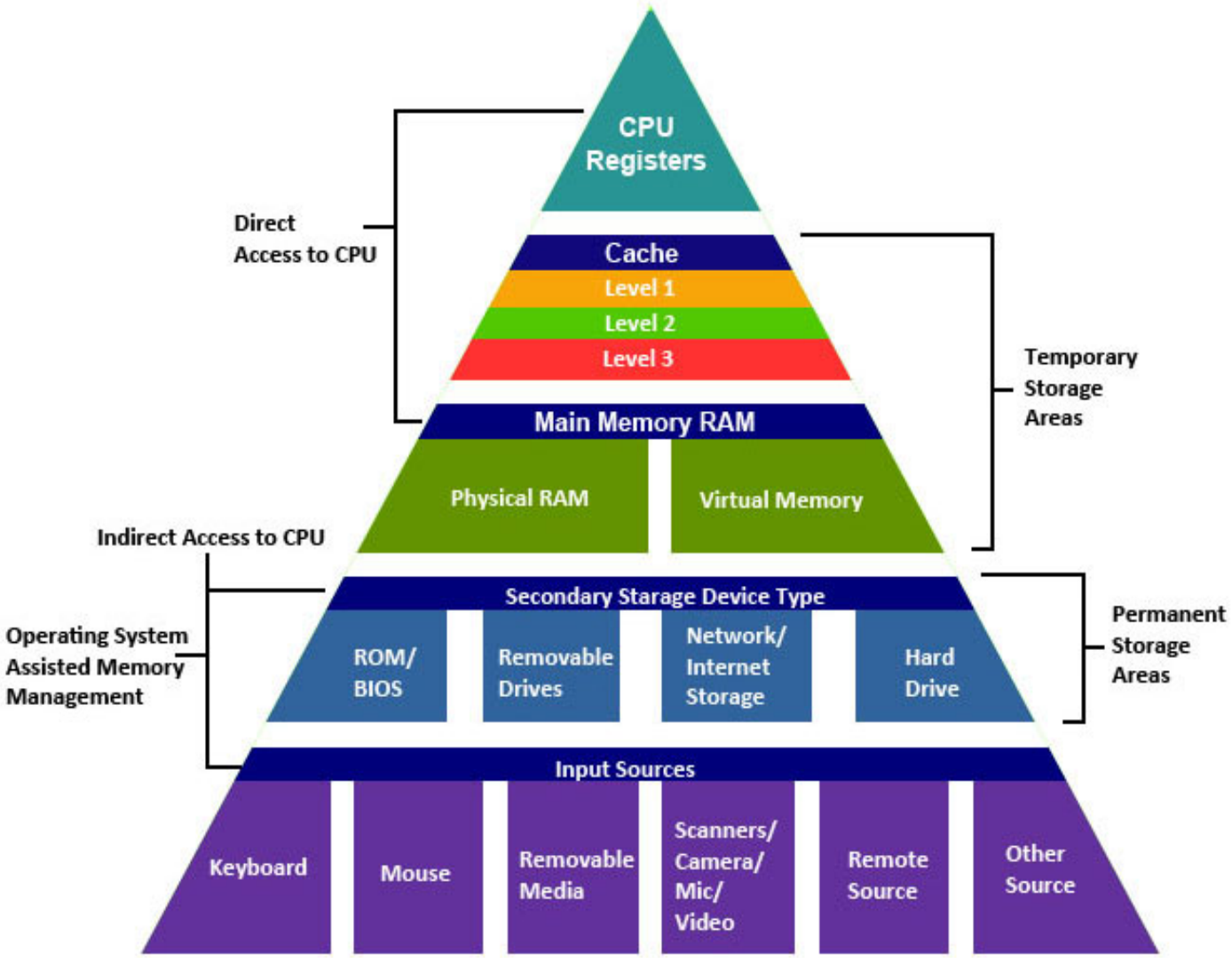
- Depending on the type of resources, there are different types of resource management:
  - Process Management: sharing the CPU
  - IO management: sharing IO devices
  - **Memory Management**: just as processes share the CPU, they also share physical memory. Memory management is about mechanisms for doing that sharing.
  - Disk Management: sharing secondary memory (HDD)

Memory Management

- Ideal programmers want memory that is:
  - Infinitely Large
  - Infinitely Fast
  - Non-volatile
  - Inexpensive

- Because there is no such a single memory, most computers have a memory hierarchy. Memory management is a part of the OS which manages the memory hierarchy.

Memory hierarchy



Memory Management Requirements

- Sharing: allows several processes coexist in main memory to access a common portion of main memory without compromising protection. Cooperating processes may need to share access to the same data. Better to allow each process to access the same copy of the data rather than have their own separate copy.

- Efficiency: both of CPU and memory should not be degraded badly by sharing. After all, the purpose of sharing is to increase overall efficiency.

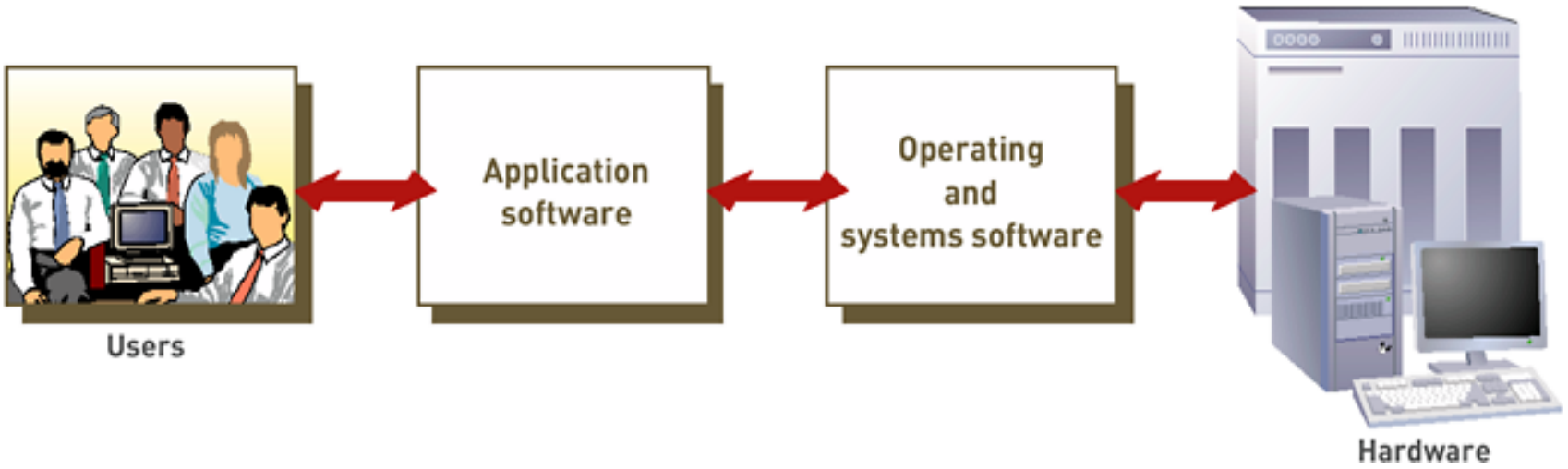
- Transparency: Multiple processes that coexist in memory, should not be aware that the memory is shared. Each process should execute regardless of where it is located in memory which results in efficient resource usage.

Memory Management Requirements cont…

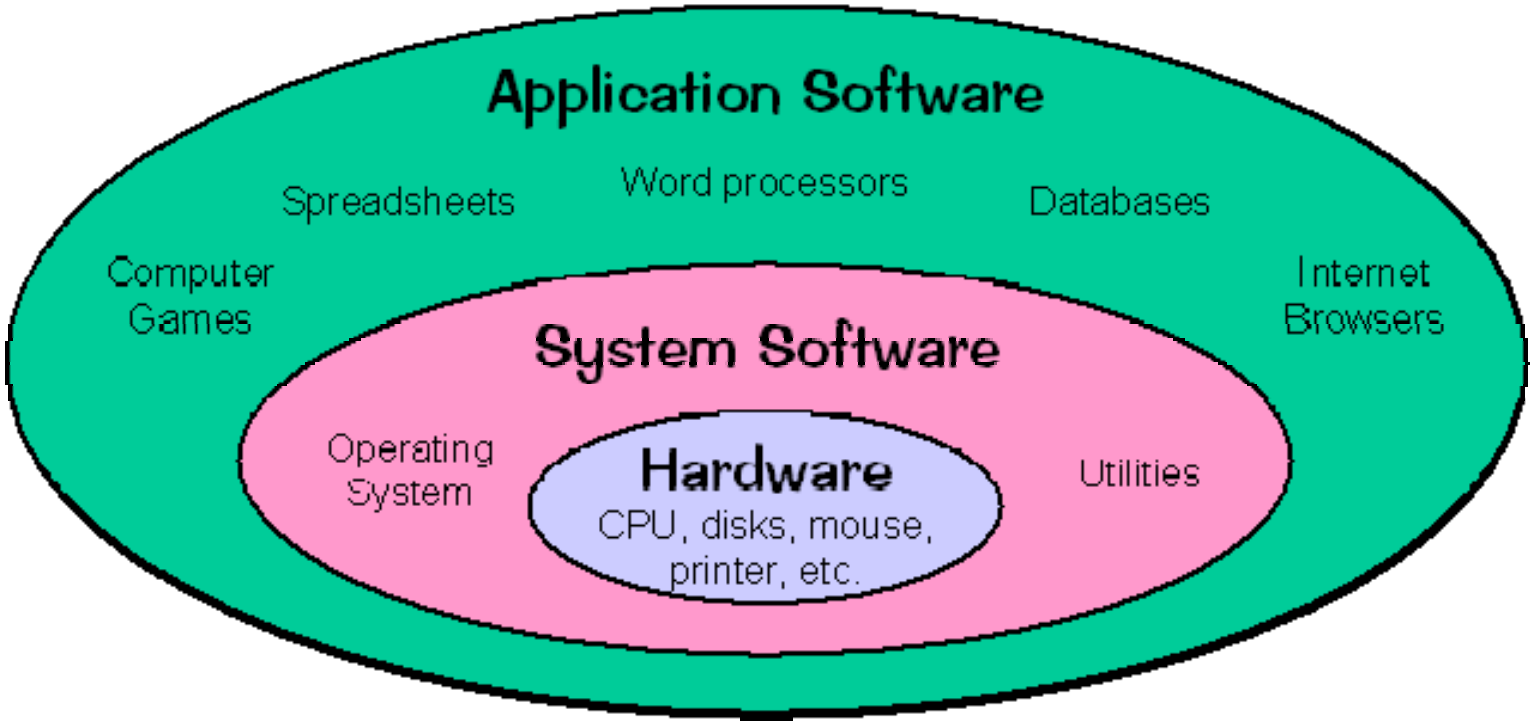
- Protection: Cannot access data of OS or other processes without permission. Protect the operating system from access by user programs, and, in addition, to protect user programs from one another (when 2 or more processes share the same physical memory, the OS must protect them from writing into each others' memory). Protection in **MMU** scheme is provided by adding two registers to the CPU which are called Base/Relocation and Bounds registers.

System software vs. Application software

- System Software: programs that control a computer system and allow you to use your computer. Types of systems software:
  - Operating systems
  - **Utility programs**
- Application Software: an application is a job or task a user wants to accomplish through a computer. These programs that help a user perform a specific job.



System software vs. Application software cont…



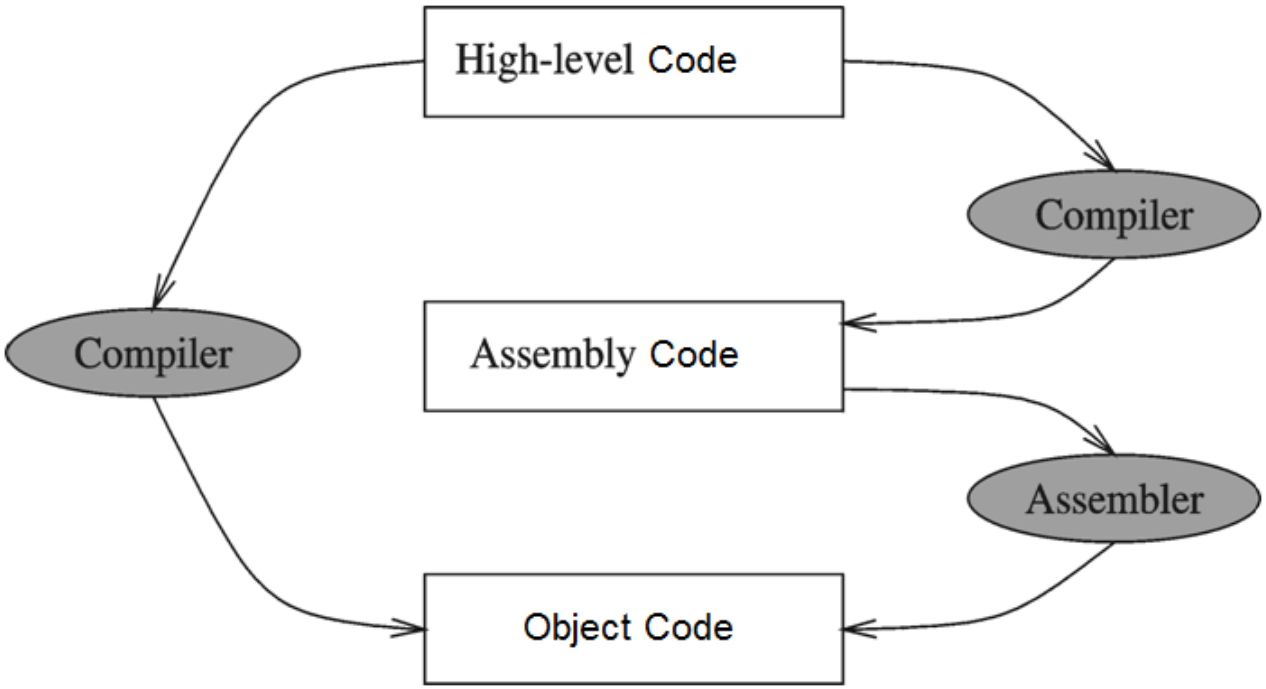
Programming tools

- OS control the computer hardware and act as an interface with application programs.

- Utilities/programming tools help OS and provide services that are not included in OS:
  - Assemblers
  - Compiler
  - Linker
  - Loader
  - Libraries

Compiler & Assembler

- Assemblers translate assembly programs to object code
- Compilers translate high-level programs to assembly or object code:
  - Either directly, or
  - Indirectly via an assembler



Object files contain a combination of machine instructions, data, and information needed to place instructions properly in memory.

Linker

- Linking: is the task of mixing the program modules together.
- Linker/Link Editor/ Linkage Editor: Links many object files into one single executable file.
- **Linker may be a part of compiler.**

Loader

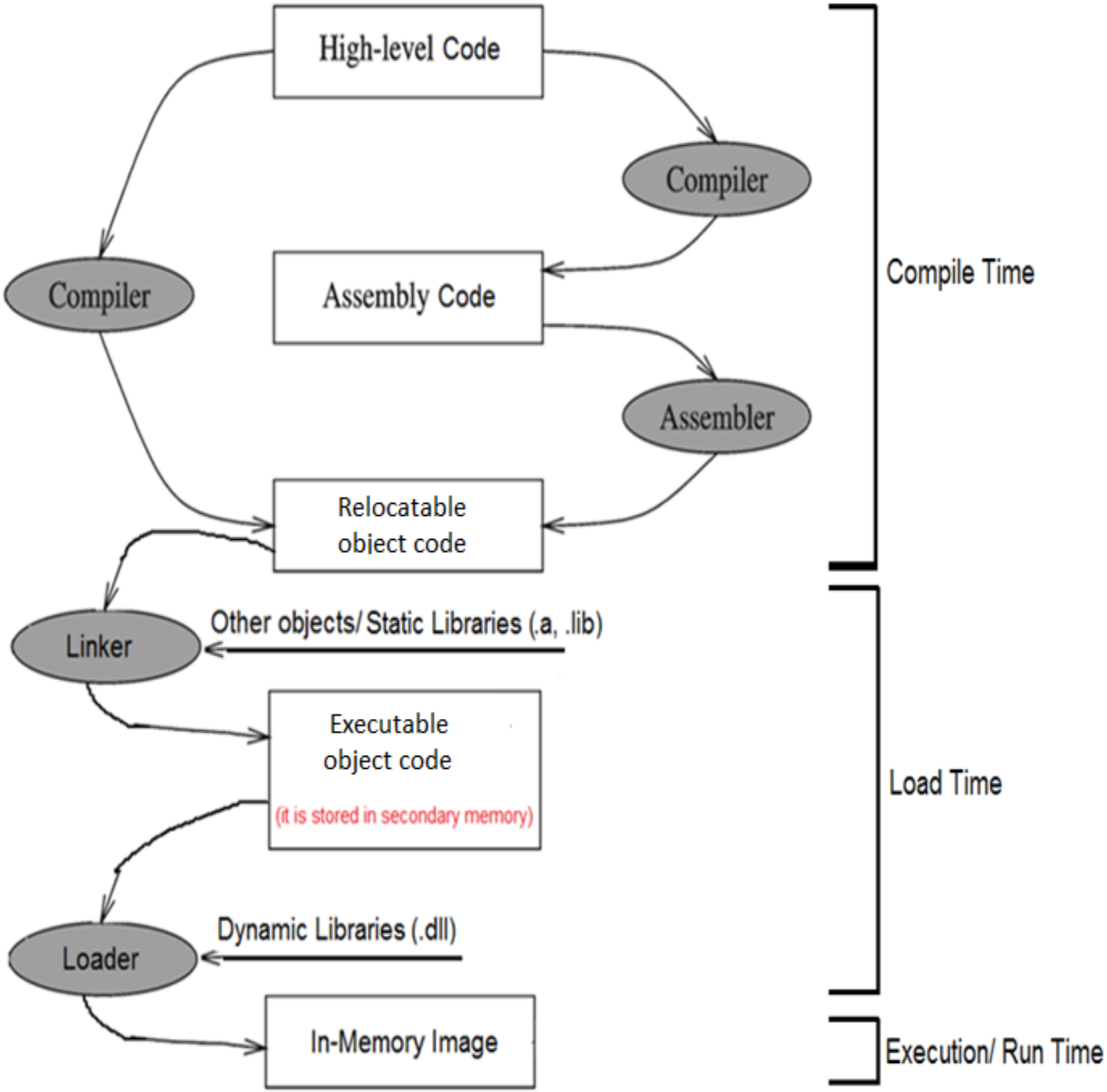
- Loading: placing the program in main memory is called loading.

- Loader: Loads an executable file into memory (at specified location), prepares them for execution, and directs the CPU to begin the execution with the first instruction. It also performs relocation of the loaded programs. Loader brings the machine language program from disk to appropriate location in memory by consulting by the MMU.

- The loader **is usually a part of the operating systems** and usually is loaded at system boot time and stays in memory until the system is rebooted, shut down, or powered off.

Translation hierarchy

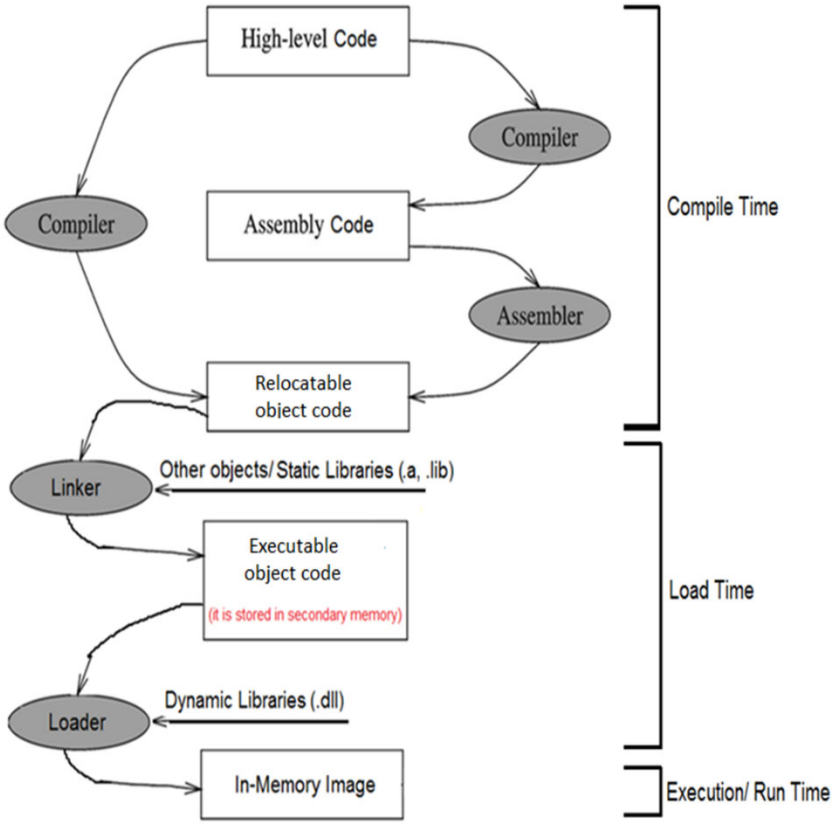
Relocation: no guarantee that the process will load in the same place in memory



Difference between Object file and Executable file

- Relocatable object code is acceptable as input to a linker; multiple files in this format can be combined to create an executable program.

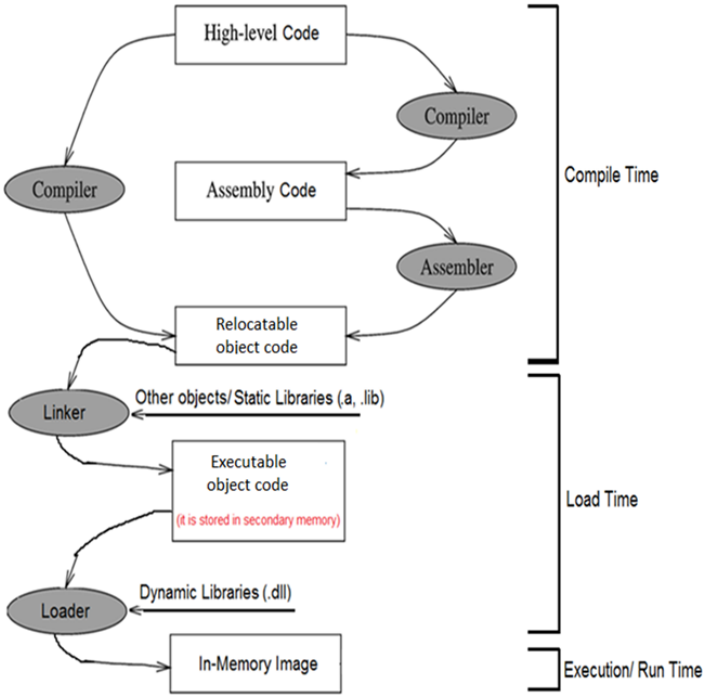
- Executable object code is acceptable as input to a loader: it can be brought into memory and run.



Difference between Object file and Executable file cont…

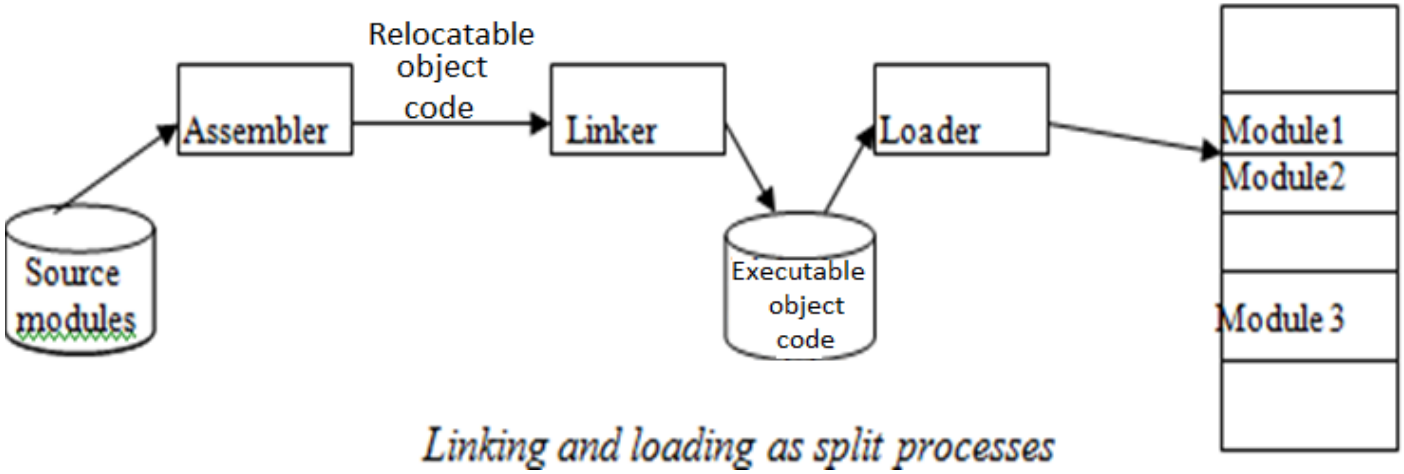
- Both are binary files but they are different as: We can execute an executable file while we cannot execute an object file.

- An object file is a file where compiler has not yet linked to the libraries, so you get an object file just before linking to the libraries, so still some of the symbols or function definitions are not yet resolved which are actually present in the libraries, and that's why we cannot execute it.

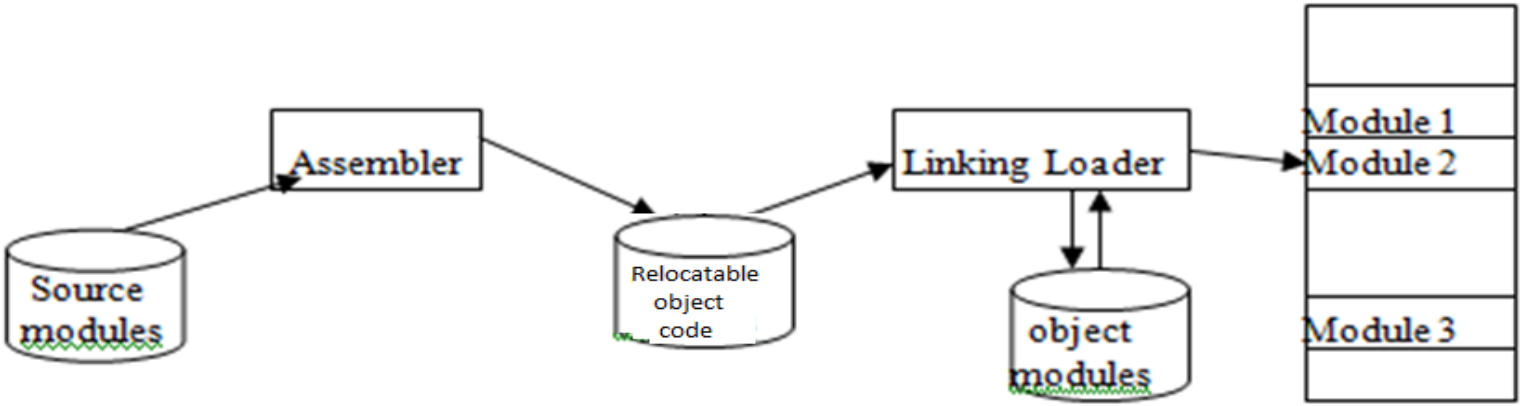


Linking loader

- Sometimes Linking and loading are frequently done at one time by a linking loader in which the code is directly loaded in memory..



Linking and loading as split processes



Linking loader

Types of addresses

- Passing through these stages, generates different type of addresses:
  - Logical/Virtual/Relative/Program Memory Address: the address are in the form of **segment:offset.** The user program deals with *logical* addresses; it never sees the *real* physical addresses.
  - Physical/Absolute/Real Address: the address seen by the memory unit or the address loaded into the MAR is a physical address. RAM deals with absolute addresses.

Logical address

- Addresses in program are generally symbolic (e.g. variable names). The variable names, constants, and instruction labels are the basic elements of the symbolic address space.
- Thus to bring the program to memory, these symbolic addresses must be translated to actual address.
- The logical address is generated by CPU to fetch instruction during instruction execution cycle.
- A program never works with RAM, instead it works with its own address space. These are the internal program addresses that internally are fixed at the time the source file is compiled and linked.

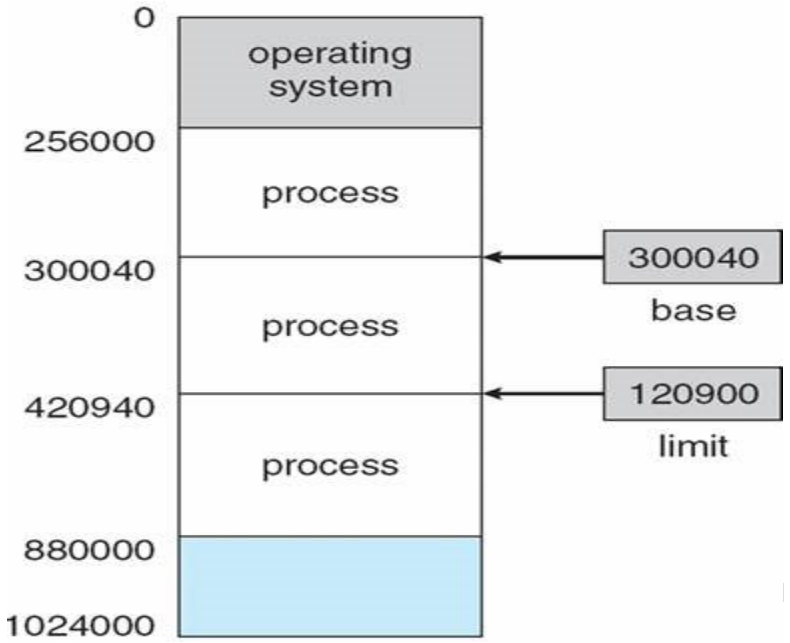
Logical address cont…

- Thus it is said that the Logical addresses are generated by the CPU, since when code is compiled for a program, the program has no idea where the code will be loaded in memory. All the compiler does is generating virtual addresses which later can be converted to real physical memory addresses.

Logical address cont…

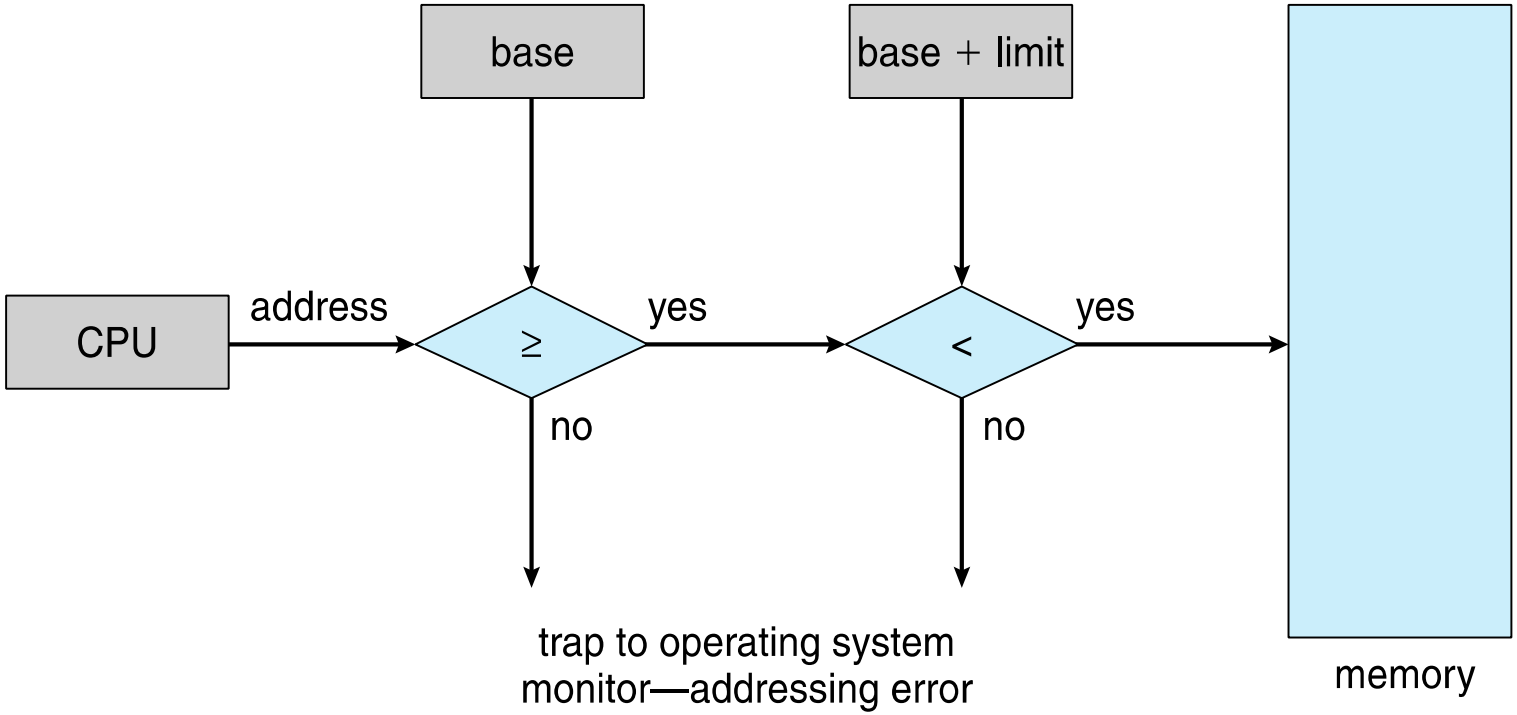
- A pair of base and limit registers define the logical address space:
  - The base register stores the start address of the block. We need base register for memory relocation.
  - The limit register holds the length of the block. We need limit register for memory protection.
- When a process runs, the dispatcher loads the base register (in the CPU) and the bound register to determine the starting physical address and ending physical address as part of the context switch.

If the base register holds 300040 and the limit register is 120900, then the program can legally access all addresses from 300040 through 420939



Logical address cont…

- Since multiple processes resides in memory, we need memory protection with the following conditions:
  - VA≥Base
  - VA<Base+Limit



Physical Address

- logical address is address related to program. It shows how much memory a particular process will take, not tell what will the exact location of the process.
- This exact location is called physical address and is generated by using address mapping.
- While the address generated by the CPU is refereed as Logical address, the address which is loaded into MAR is refereed as the Physical Address.

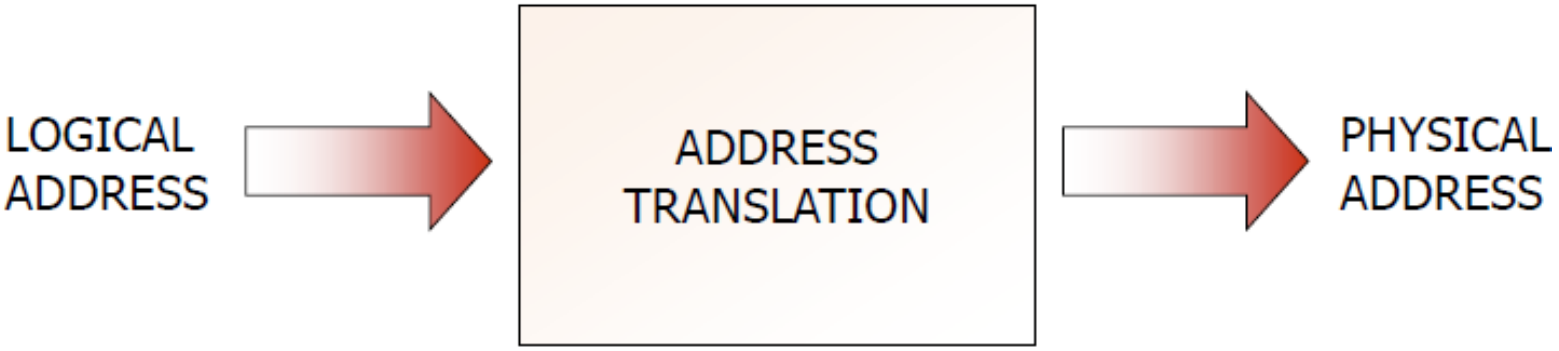
Relocatable vs. Non-Relocatable code

- Having two types of addresses, there can be two types of codes:
  - Absolute/non-Relocatable code: all addresses are physical. It must always be loaded at a particular known fixed location in memory. This is useful when particular software needs to be found in clearly pre-defined locations.
  - Relocatable/offset-based code: does not use physical addresses which means all addresses are virtual. It can be loaded anywhere in the memory.

The ability to load and execute a given program to arbitrary place in memory is called relocation.

Address translation

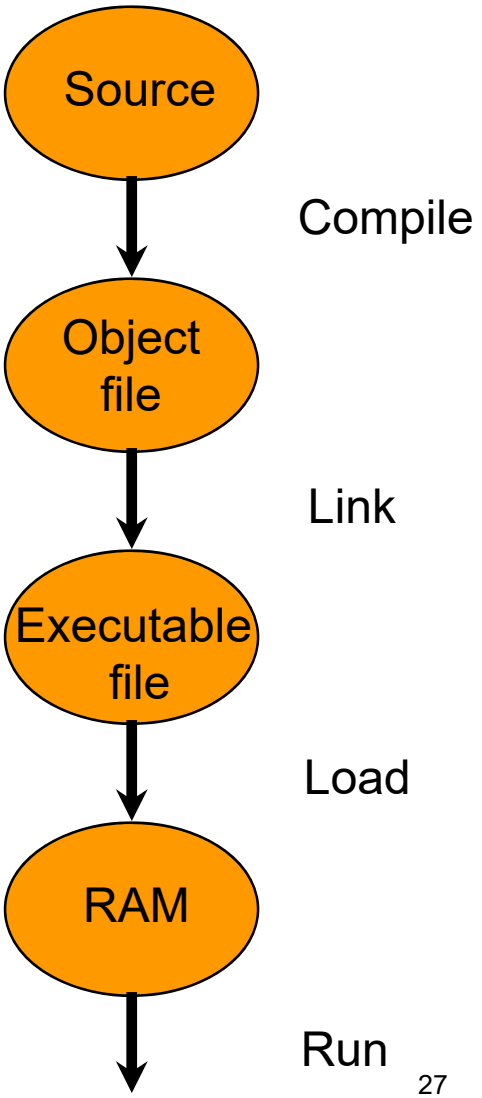
- The program's instructions and data must be map/bound to actual physical addresses.
- Thus, Address translation/ address binding/address mapping is done to map virtual addresses to physical addresses.



Address translation cont…

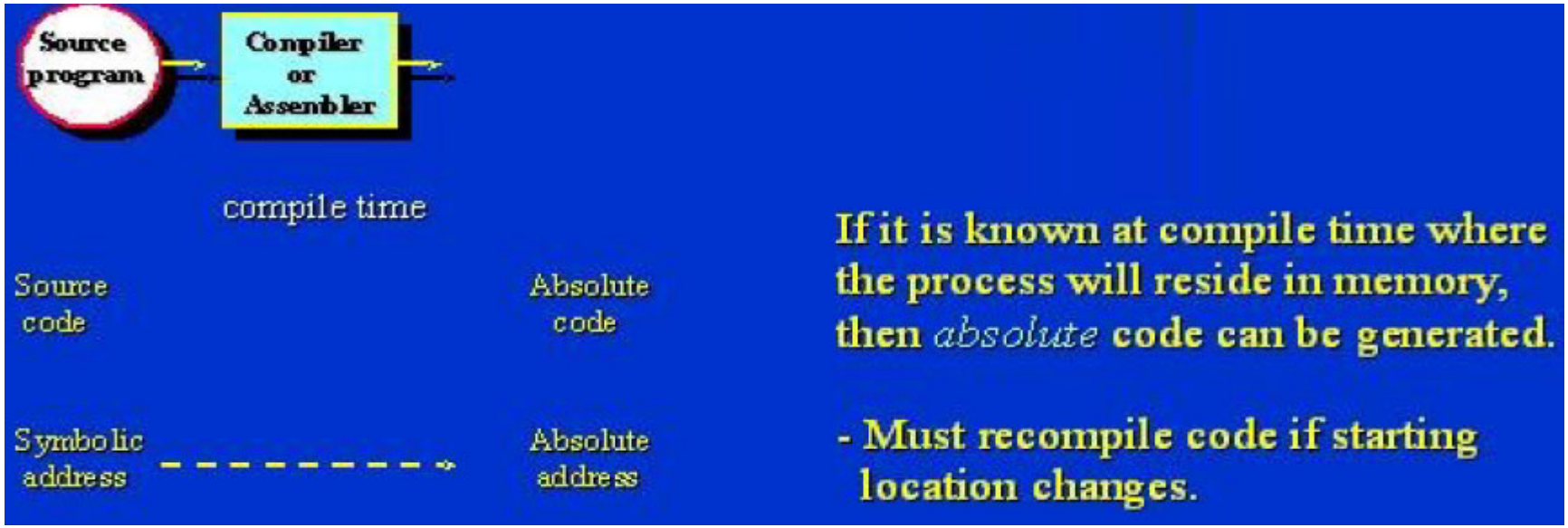
- The address binding can happen either:
  - At Compile time
  - At Link time
  - At Load time
  - At Execution/Run time

- Types of address binding:
  - Static binding (old scheme): bind before execution time (either compile, linking, or load time).
  - Dynamic Binding: delay address binding until execution time.



Address binding at compile time

- If the compiler knows that during compile time where process will reside in memory then it generates the absolute address during compilation.
- The compiler generates the physical location in memory starting from some fixed position (OS does nothing), so the code must be **re-compiled** if starting location of the process changes.
- This type of address translation only needs limit register (since starting address is fixed) and also it does not need linker.



Address binding at compile time cont…

- Address binding at compile time is only possible in systems where:
  - we know the contents of the main memory in advance
  - we know what address in the main memory we have to start the allocation from. For example, MS-DOS .COM files always load at address 0x100.

- Knowing both of these things is not possible in modern multi-programming systems. So it can be safely said the compile time binding would be possible in systems not having support for multi-programming.

Address binding at link time

- If compiler does not known at the compile time where process will reside, a relocatable address is generated (by CPU) at compile time. Now, if the linker knows in advance where the linked program will be loaded in memory, it takes this relocatable address from the compiler and generates physical address.

- In this case, the linker can convert the relocatable address (taking from compiler) to absolute address, thus, the address binding is delayed until link time.

- Programmer who are placing their applications in ROM memory (such as BIOS ROM) often employ this scheme.

- This is easy since it is assumed the linked program will be loaded at 0. This translation scheme needs small hardware requirements (since starting address is fixed, it only needs limit register).

Address binding at load time

- Load time binding does **not** fix the starting address thus needs mode both base and limit registers. When the OS loads the program into memory, it decides where to place it.

- If compiler does not known at the compile time where process will reside, a relocatable address is generated (by CPU) at compile time. Now, if the loader knows in advance where the program will be loaded in memory, it takes this relocatable address from the compiler and translates it to physical address.

- The loader contains the base address of the process in the main memory. So when the time for loading a process into the main memory comes, the based address is added to all the logical addresses by the loader to generate the physical addresses.

- This way, the process cannot be moved (relocated) during execution. If the base address changes you need to **re-load** the whole process to fix up all the relocatable addresses.

Relocatable means that the program image can reside anywhere in physical memory

Address binding at execution time

- Address binding at execution time is used if process can be moved from one memory segment to another **during execution.**

load time binding is done by loader and run time binding is done by CPU.

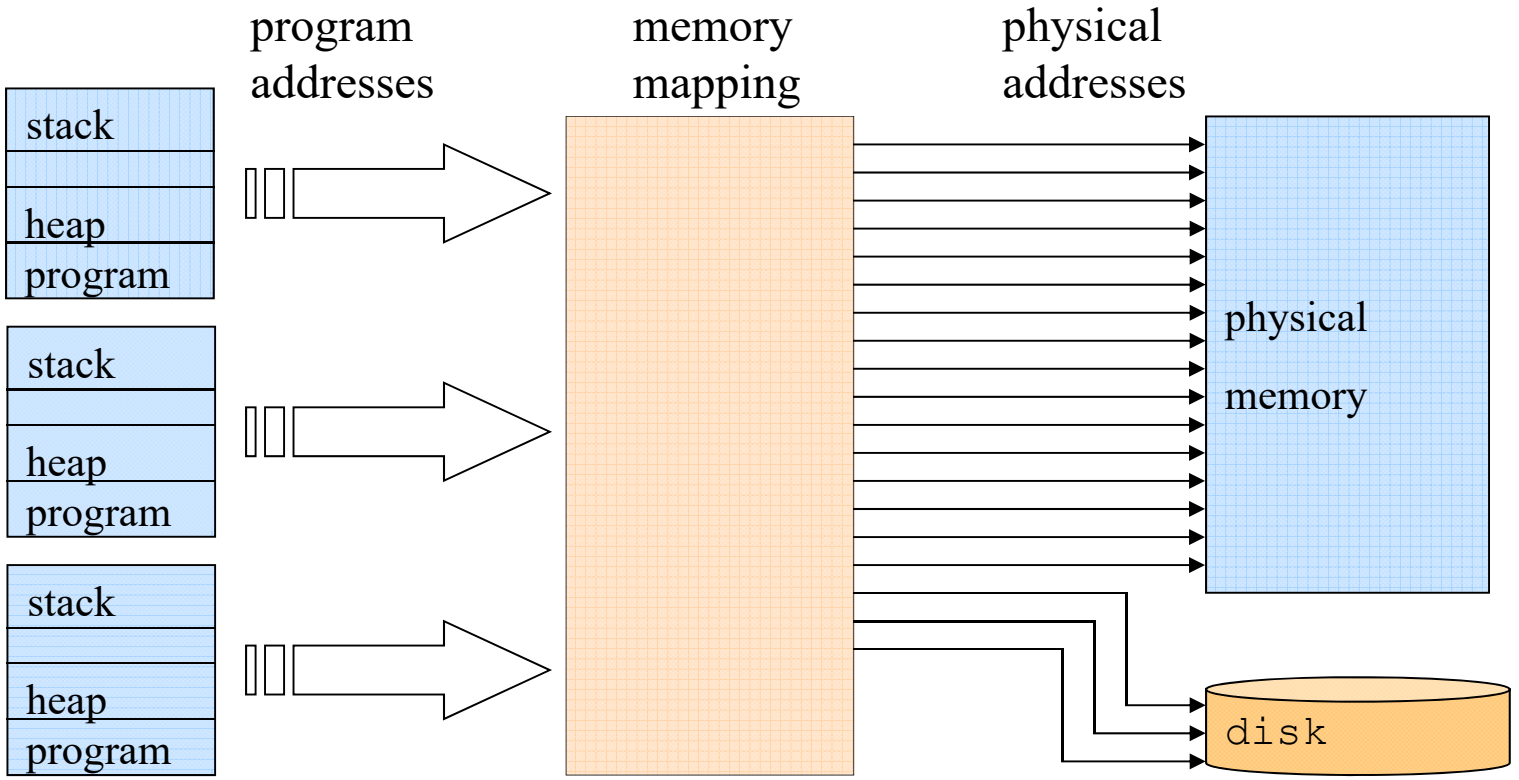
- **Compile time:** Code is fixed to an absolute address. Recompilation is necessary if the starting location changes. (MS-DOS .COM format)
- **Load time:** Code can be loaded to any portion of memory. (Relocatable code)
- **Run time:** Code can be move to any portion of memory during its execution.

Address binding at execution time cont…

- The run time mapping of logical to physical addresses is handled by the memory management unit (MMU).
- The base register is now called relocation register.

Memory management unit (MMU)

- The input to MMU (its memory mapping hardware) is the program's logical address and output is the corresponding physical address.



MMU: is a hardware device that maps **virtual** to **physical** address