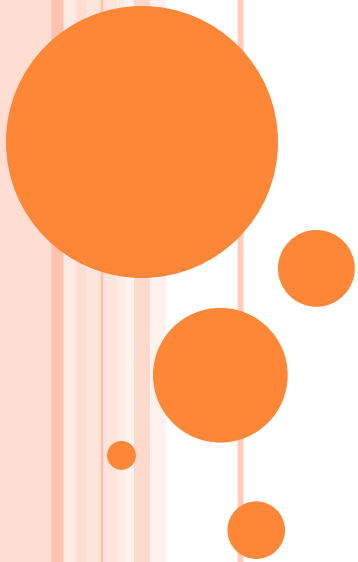


# INTRODUCTION TO SYSTEM SOFTWARE & OPERATING SYSTEM



# TOPICS

- Overview of all System S\W
  - Compiler
  - Assembler
  - Linker
  - Loader
- Operating system
- OS services and components
- Types of OS
- Virtual Machines
- System Calls
- Buffering and Spooling

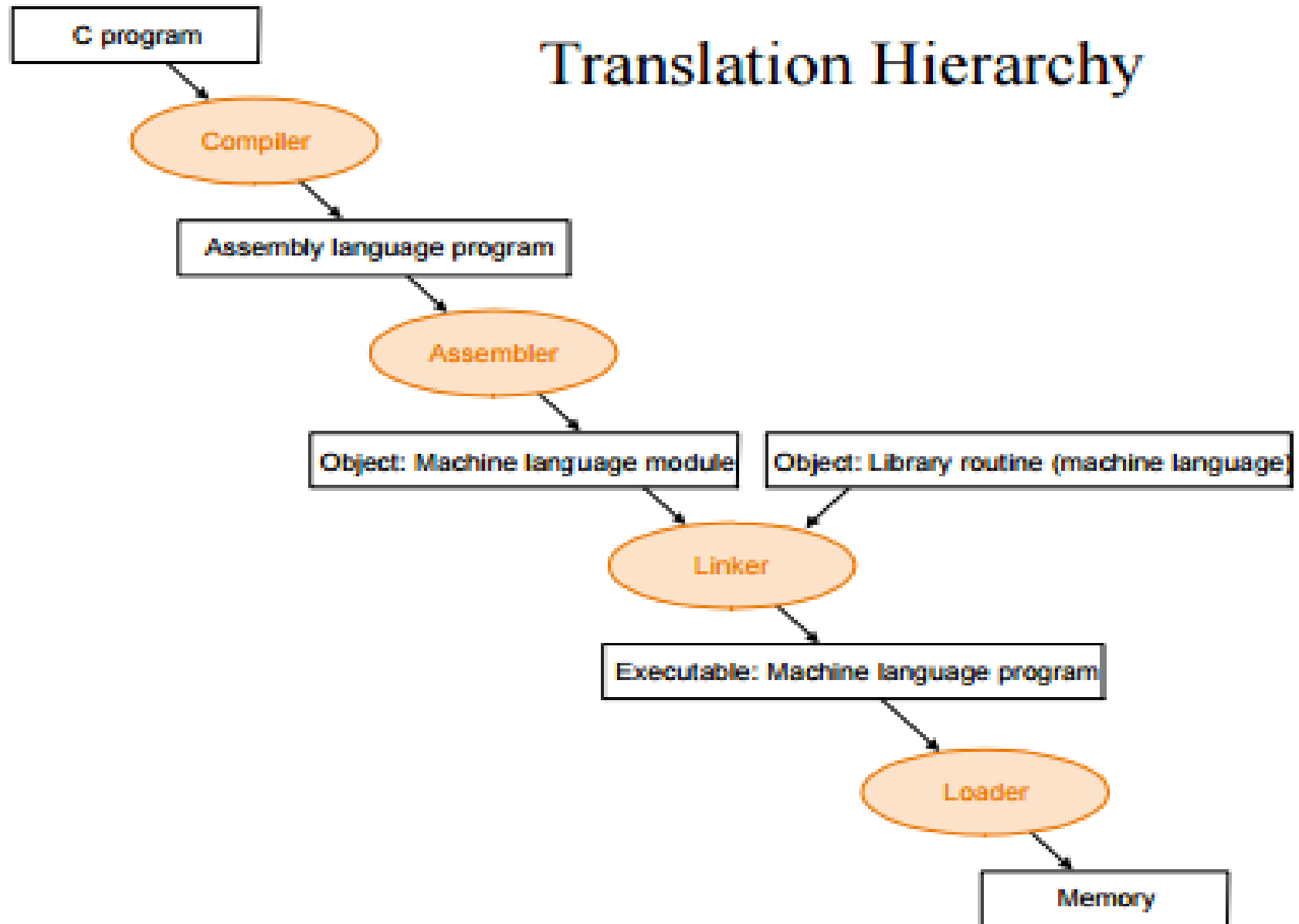


# SYSTEM SOFTWARE

- System software is a type of computer program that is designed to run a computer's hardware and application programs.
- If we think of the computer system as a layered model, the system software is the interface between the hardware and user applications
- An application program (often just called an application or app) performs a particular function for the user.
- Examples: Browsers, word processors and spreadsheets etc...



# Translation Hierarchy



# COMPILER

- A **compiler** is a computer program (or a set of programs) that transforms source code written in a programming language (the source language) into another computer language (the target language), with the latter often having a binary form known as **object code**.
- The most common reason for converting source code is to create an **executable program**.



# CLASSIFICATIONS

- **Cross-compiler:** the compiled program can run on a computer whose CPU or operating system is different from the one on which the compiler runs
- **Single pass compilers(One pass compiler):** is a compiler that passes through the parts of each compilation unit only once, immediately translating each part into its final machine code
- **Multi pass compilers:** compiler which converts the program into one or more intermediate representations in steps between source code and machine code, and which reprocesses the entire compilation unit in each sequential pass.
- **Optimizing compilers:** is a compiler that tries to minimize or maximize some attributes of an executable computer program.
- **Decompiler :** A program that translates from a low level language to a higher level one



# COMPILER STRUCTURE

**The front end:** Verifies syntax and semantics, and generates an intermediate representation or IR of the source code for processing by the middle-end.

- Generates errors and warning. Aspects of the front end include lexical analysis, syntax analysis, and semantic analysis.

**The middle end:** Performs optimizations, including removal of useless or unreachable code, discovery and propagation of constant values, relocation of computation to a less frequently executed place (e.g., out of a loop), or specialization of computation based on the context.

- Generates another IR for the backend.

**The back end:** Generates the assembly code, performing register allocation in process. (Assigns processor registers for the program variables where possible.)

- Optimizes target code utilization of the hardware by figuring out how to keep parallel execution units busy, filling delay slots.



# ASSEMBLER

- An assembler is a program that takes basic computer instructions and converts them into a pattern of bits that the computer's processor can use to perform its basic operations
- The output of the assembler program is called the object code or object program relative to the input source program.
- The sequence of 0's and 1's that constitute the object program is sometimes called machine code.
- The object program can then be run (or executed) whenever desired.





# TYPES

**One Pass:** During that single pass, the assembler handles both label definitions and assembly.

**Two Pass:**

- In the first pass it reads the entire source file, looking only for label definitions. All labels are collected, assigned values, and placed in the symbol table in this pass.
- No instructions are assembled and, at the end of the pass, the symbol table should contain all the labels defined in the program.
- In the second pass, the instructions are again read and are assembled, using the symbol table.



# LINKER

- Tool that merges the object files produced by separate compilation or assembly and creates an executable file
- Three tasks – Searches the program to find library routines used by program, e.g. printf(), math routines,...
- Determines the memory locations that code from each module will occupy and relocates its instructions by adjusting absolute references
- Resolves references among files



# TYPES

- **Static linking** is the result of the linker copying all library routines used in the program into the executable image.
- This may require more disk space and memory than dynamic linking, but is more portable, since it does not require the presence of the library on the system where it is run.
- **Dynamic linking**, that is the postponing of the resolving of some undefined symbols until a program is run.
- That means that the executable code still contains undefined symbols, plus a list of objects or libraries that will provide definitions for these.
- Loading the program will load these objects/libraries as well, and perform a final linking.



# LOADER

- Part of the OS that brings an executable file residing on disk into memory and starts it running

## Steps

- Read executable file's header to determine the size of text and data segments
- Create a new address space for the program
- Copies instructions and data into address space
- Copies arguments passed to the program on the stack
- Initializes the machine registers including the stack ptr
- Jumps to a startup routine that copies the program's arguments from the stack to registers and calls the program's main routine



# TYPES

## **ABSOLUTE LOADERS**

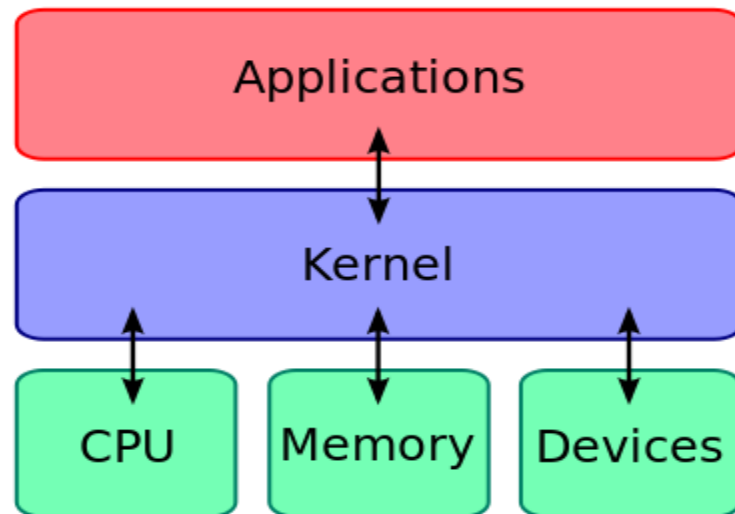
- The loader loads the file into memory at the location specified by the beginning portion (header) of the file, then passes control to the program.
- If the memory space specified by the header is currently in use, execution cannot proceed, and the user must wait until the requested memory becomes free by the assembler.

## **THE RELOCATING LOADER**


- The relocating loader will load the program anywhere in memory, altering the various addresses as required to ensure correct referencing.
- The decision as to where in memory the program is placed is done by the Operating System, not the programs header file

# KERNEL

- **Kernel** is a computer program that manages input/output requests from software, and translates them into data processing instructions for the central processing unit and other electronic components of a computer.
- The kernel is a fundamental part of a modern computer's operating system

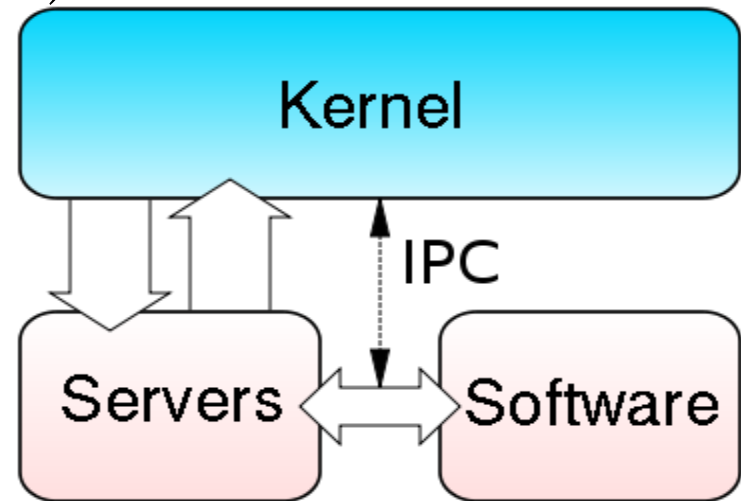
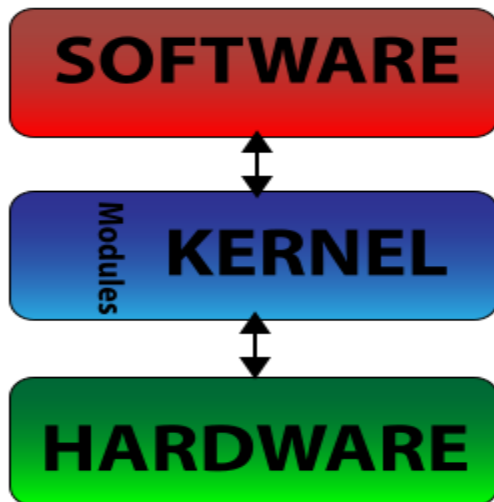


# MONOLITHIC KERNEL

- A **monolithic kernel** is an operating system architecture where the entire operating system is working in kernel space and is alone in supervisor mode.
  - The monolithic model differs from other operating system architectures (such as the microkernel architecture).
  - In that it alone defines a high-level virtual interface over computer hardware.
  - A set of primitives or system calls implement all operating system services such as process management, concurrency, and memory management.
  - Device drivers can be added to the kernel as modules.
- 

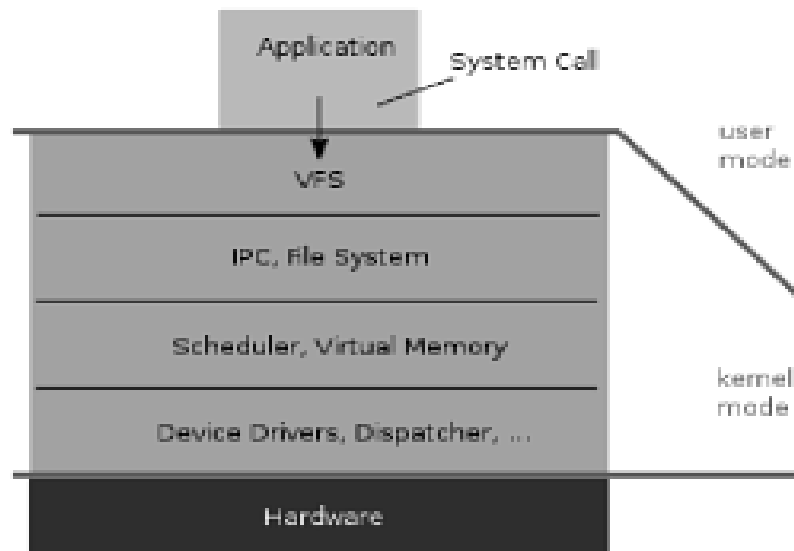
# MICROKERNEL

- A **microkernel** (also known as  *$\mu$ -kernel*) is the near-minimum amount of software that can provide the mechanisms needed to implement an operating system (OS).
- These mechanisms include low-level address space management, thread management, and inter-process communication (IPC)

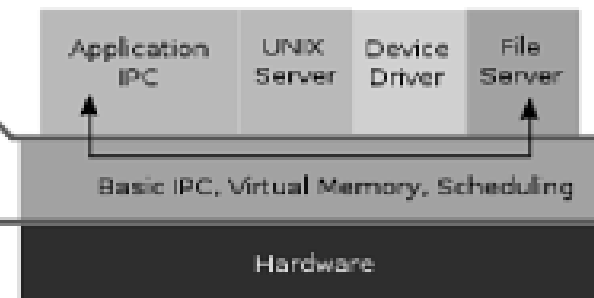




## Monolithic Kernel based Operating System



## Microkernel based Operating System



**Figure 1:** Structure of *monolithic* and *microkernel*-based operating systems, respectively



THANK YOU

