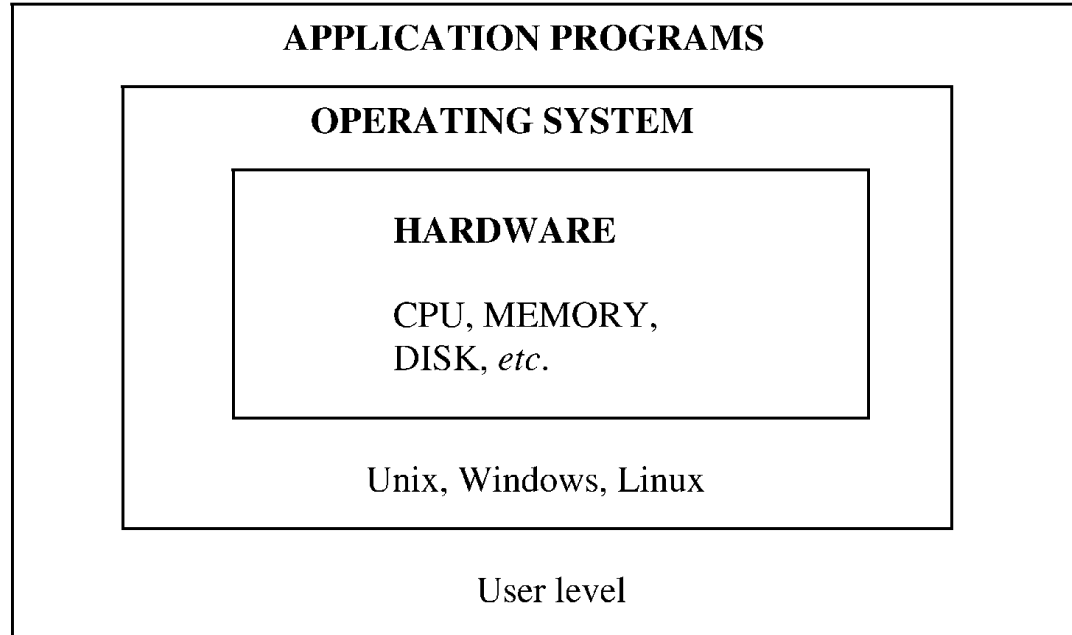


# Computer System Organization

Modern computer systems have a layered organization:



# Application Level

- **Who?** Application programmers
- **What?** Application programs
- **Language?** High-level language  
(Java, Python, Prolog, C++, C#, C, *etc.*)
- **How executed?** Typically *translated* by a *compiler* into a lower-level language  
(JVM byte code, assembly language, machine code, *etc.*)

Programs at this level typically run on a high-level language *virtual machine* or *runtime system*.

# Systems Level

- **Who?** Systems programmers
- **What?** Window systems, system utilities, system libraries, language implementations, *etc.*
- **Language?** Lower-level language (often C, assembly language)
- **How executed?** Typically *translated* by a *compiler* or *assembler* into *machine instructions* (directly executed) and *system calls* (interpreted by the OS).

# OS Level

- **Who?** OS programmers
- **What?** Device drivers, filesystems, memory management, process/thread schedulers, *etc.*
- **Language?** Lower-level language (usually C, assembly language)
- **How executed?** *Translated* by a *compiler* or *assembler* into *machine instructions* (directly executed).

# Hardware Levels

- **Who?** Hardware designers
- **What?** CPU, RAM, cache, bus, peripheral devices
- **Language?** Hardware description languages
- **How executed?** *Realized* in hardware  
(e.g. with “silicon compilers” and the like).

# Translation versus Interpretation

Two schemes for executing programs:

- **Translation:** Program written in a “higher-level language” H is *translated* or *compiled* into an equivalent program in “machine language” M, and then executed.  
(e.g. Java translated to “byte code” and executed by JVM)
- **Interpretation:** Program written in a “higher-level language” is directly executed by an *interpreter* which inspects the program and performs an equivalent series of machine language instructions “on-the-fly”.  
(e.g. the JVM is actually a byte-code interpreter)

# Systems-Level Programming

This course: works in the layer between the application and the operating system.

- Modern high-level programming languages (e.g. Java) insulate you from underlying hardware and software details.
- You can build application software today without knowing about digital logic design (e.g. NAND gates, flip-flops) or electronics (e.g. transistors, VLSI).

- But to be an effective software engineer, you still need to know some “under-the-hood” details:
  - *Organization of memory*
  - *Relationship of processor and memory*
  - *How data is represented in memory and processed*
  - *How applications interact with the OS*

Learning to program in lower-level languages (e.g. C and assembler) exposes you to these details.

# High-level versus Low-level Languages

Programming languages have become more and more *high-level* over the history of computing.

- **High-level languages:** Problem-oriented, human-friendly, very distant from how the hardware actually carries out the computation.
- **Low-level languages:** Machine-oriented, not as human-friendly, closer to the way the hardware actually carries out the computation.

Understanding how the hardware actually carries out a computation originally expressed in a high-level language will make you a better programmer.