Unit 2

Impact of Machine Architectures

Introduction

- Early languages were designed to run programs efficiently on expensive hardware. Therefore, early languages had a design that translated programs into efficient machine code, even if the programs were hard to write.
- Today, machines are inexpensive, machine cycles are usually plentiful, but programmers are expensive. There is a greater emphasis in designing programs that are easy to write correctly even if they execute somewhat more slowly.
- The machine architecture influences the design of a language in two ways:
    1. The underlying computer on which programs written in the language will execute.
    2. The execution model, or virtual computer, that supports that language on the actual hardware.

The Operation of a Computer

- A **computer** is an integrated set of algorithms and data structures capable of storing and executing programs.
- A computer may be an actual physical device (**actual computer** or **hardware computer**) or a **software-simulated computer** running on another computer.
- A programming language is implemented by constructing a **translator**, which translates programs in the language into machine language programs that can be directly executed by some computer (hardware or virtual computer).
- A computer consists of six major components that correspond closely to the major aspects of a programming language design:
  1. **Data.** A computer must provide various kinds of elementary data items and data structures to be implemented.

The Operation of a Computer

2. **Primitive operations.** A computer must provide a set of primitive operations for manipulating the data.
3. **Sequence control.** A computer must provide mechanisms for controlling the sequence in which the primitive operation are to be executed.
4. **Data access.** A computer must provide mechanisms for controlling the data supplied to each execution of an operation.
5. **Storage management.** A computer must provide mechanisms to control the allocation of storage for programs and data.
6. **Operating environment.** A computer must provide mechanisms for communication with an external environment containing programs and data to be processed.

The Operation of a Computer

- **Computer Hardware:**
  - Hardware computer organizations vary widely. Here we discuss *von Neumann architecture*.
  - A ***main memory*** contains programs and data to be processed. Processing is performed by an ***interpreter***, which takes each machine language instruction in turn, decodes it, and calls the designated primitive operation with the designated operands as input. The ***primitive operations*** manipulate the data in main memory and in high-speed registers and also may transmit programs or data between memory and the external operating environment.
  - **Data.** Data storage component are *main memory*, *cache memory*, *high-speed registers*, and *external files*. A computer has certain built-in data types that can be manipulated directly by hardware primitive operations. Programs are also a form of data and represented as machine language instructions composed of an operation code and a set of operands.

The Operation of a Computer

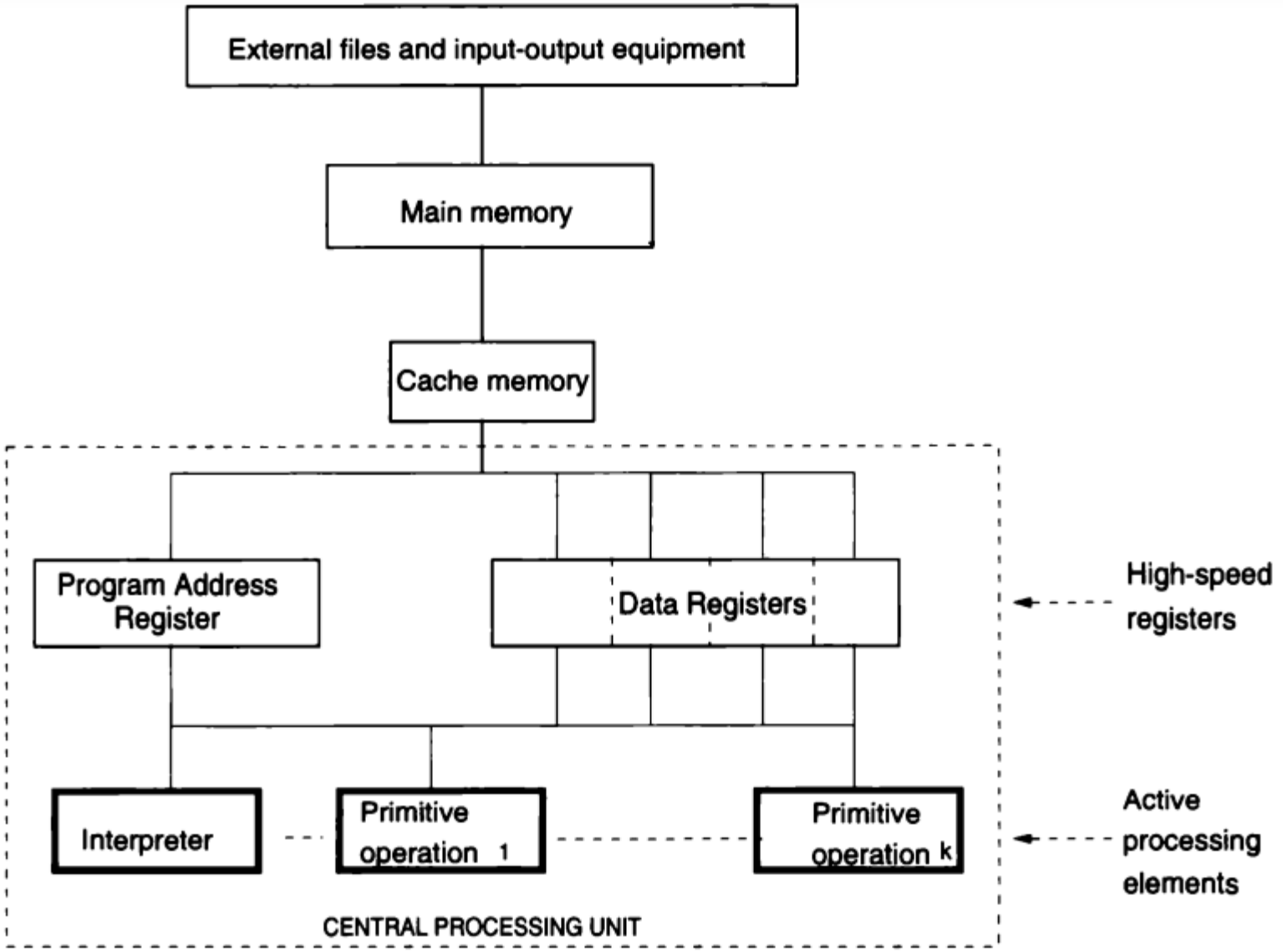

Fig: Organization of a conventional computer

The Operation of a Computer

- **Operations.** A computer must contain a set of built-in primitive operations, usually paired one to one with operation codes that may appear in machine language instructions.
- **Sequence control.** The next instruction to be executed at any point during execution of a machine language program is usually determined by the contents of a special ***program address register*** (also called ***location counter***), which always contains the memory address of the next instruction. The ***interpreter*** actually uses the program address. The Interpreter is central to the operation of a computer. Typically, the interpreter executes the simple cycle algorithm as shown on the next slide.
- **Data access.** Besides an operation code, each machine instruction must specify the operands that the designated operation is to use. A computer must incorporate a means of designating operands and a mechanism for retrieving operands. Likewise, the result of a primitive operation must be stored in some location.

The Operation of a Computer

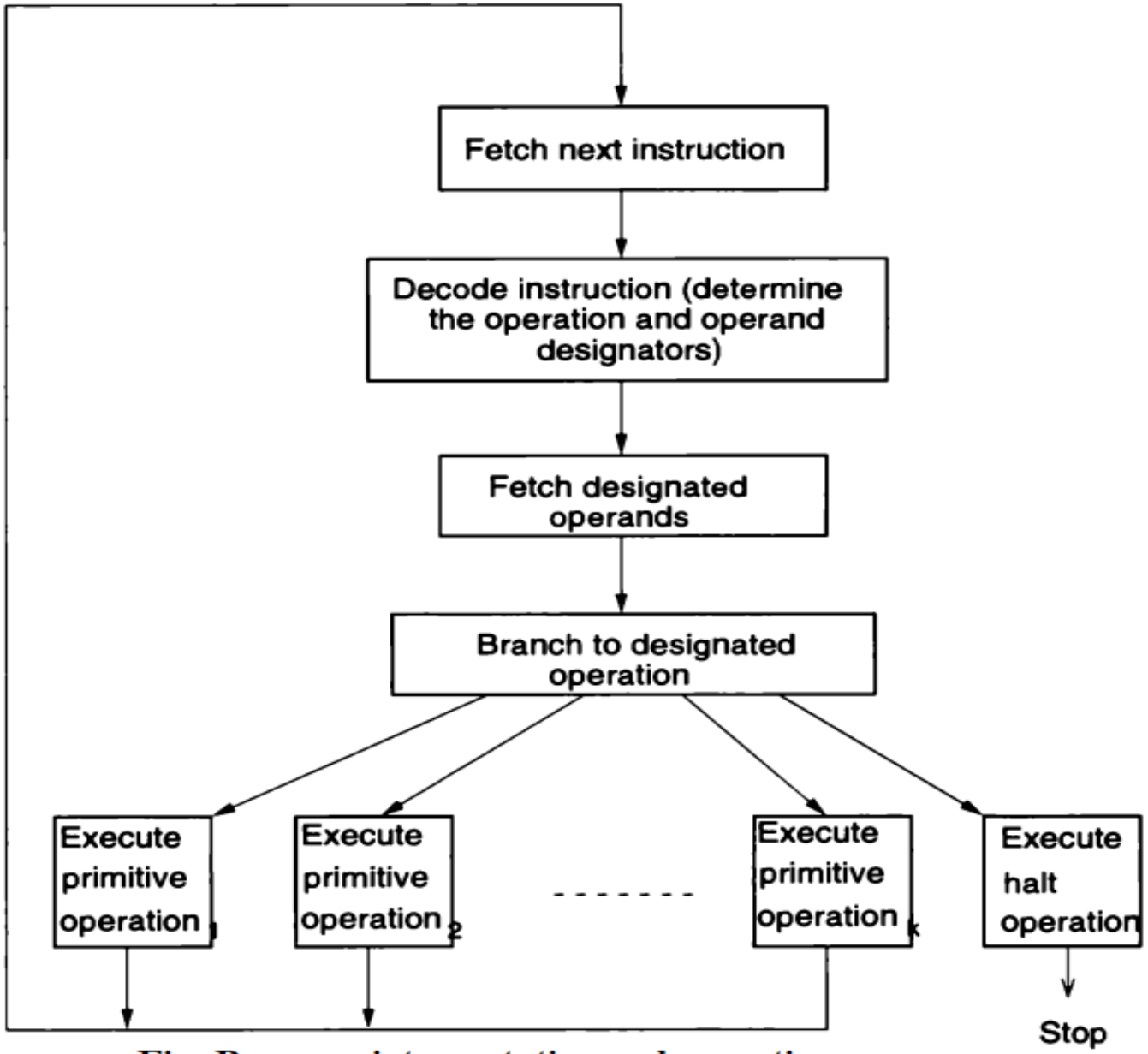

Fig: Program interpretation and execution

The Operation of a Computer

- **Storage management.** To balance speed of CPU, memory access, and disk access, various storage management facilities are employed. For example, programs and data reside in memory throughout program execution, ***multiprogramming*** is used where the computer will execute another program if a program waits for data to be read from disk, ***paging*** try to anticipate which program and data addresses will be most likely used in the near future so that hardware can make them available to the CPU, and ***cache memory*** is used for imbalance between main memory and CPU.
- **Operating environment.** Operating environment ordinarily consists of peripheral storage and I/O devices that represent outside world to the computer and any communication with the computer must be by way of the operating environment.

- **Alternative computer architecture (**Multiprocessors). Multiprocessor uses multiple CPUs with single sets of memory, disks etc. to solve imbalance problem between external data devices and CPU registers. The OS runs different programs on different CPUs.

The Operation of a Computer

- **Computer state.** A program executes through a series of states, each defined by the contents of memory, registers, and external storage at some point during execution. The initial contents of these storage areas define the *initial state* of the computer. Each step in program execution transforms the existing state into a new state through modification of the contents of one or more of these storage areas. This transformation of state is termed a *state transition*. When program execution is complete, the *final state* is defined by the final contents of these storage areas. Program execution may be seen as the sequence of state transitions made by the computer.

The Operation of a Computer

- **Firmware Computer:**
  - Actual hardware computers usually have a rather low-level machine language.
  - A common alternative to the strict hardware realization of a computer is the ***firmware computer*** simulated by a microprogram running on a special microprogrammable hardware computer.
  - The machine language of firmware computer consists of an extremely low-level set of microinstructions, which usually specify simple transfers of data between main memory and high-speed registers, between the registers themselves, and from registers through processors such as adders and multipliers to other registers.
  - A special microprogram is coded, using this simple instruction set, that defines the interpretation cycle and the various primitive operations of the desired computer.
  - The microprogram simulates the operation of the desired computer on the microprogrammable host computer.

The Operation of a Computer

- Ordinarily the microprogram resides in a special read-only memory in the host computer and is executed at high speed by the host computer hardware.
- This microprogram simulation of a computer is essentially the same, in concept, as the software simulation technique, except that the host computer in this case is especially designed for microprogramming and provides execution speeds for the simulated computer comparable to those obtained by the direct hardware realization.
- Microprogram simulation of a computer is sometimes termed ***emulation***. We also refer to the resulting computer as a ***virtual computer*** because it is simulated by the microprogram.

The Operation of a Computer

- **Translators and Virtual Architectures:**
  - Programming, of course, is most often done in a high-level language far removed from the hardware machine language. The two solutions to execute the high-level language on the actual computer are **translators** and **software simulation**.
  - **Translators:**
    - A translator could be designed to translate programs in the high-level language into equivalent programs in the machine language of the actual computer.
    - In general, Translators accept programs in some source language as input and produce functionally equivalent programs in another object language as output.
    - Several specialized types of translators are **assembler**, **compiler**, **loader** or **link editor**, and **preprocessor** or **macroprocessor**.
    - An **assembler** is a translator whose object language is machine language for an actual computer but whose source language is assembly language. Most instructions in the source language are translated one for one into object language instructions.

The Operation of a Computer

- A **compiler** is a translator whose source language is a high-level language and whose object language is close to the machine language of an actual computer, either being an assembly language or some variety of machine language.
- A **loader** or **link editor** is a translator whose object language is actual machine code and whose source language is almost identical. It makes a single file from several files of relocatable machine code.
- A **preprocessor** or a **macroprocessor** is a translator whose source language is an extended form of some high-level language and whose object language is the standard form of the same language. The object program produced by a preprocessor is then ready to be translated and executed by the usual processor for the standard language.
- Translation of a high-level source language into executable machine language programs often involves more than one translation step. Moreover, the compilation step may involve a number of passes that progressively translate the program onto various intermediate forms before producing the final object program.

The Operation of a Computer

- **Software Simulation (Software Interpretation) :**
  - Rather than translating the high-level language programs into equivalent program in object language, software simulator executes the input program directly.
  - Software simulator is a computer whose machine language is the high level language. In such a case, we say that the host computer creates a *virtual machine* simulating the high-level language.
  - When the host computer is executing the high-level program, it is not possible to tell whether the program is being executed directly by the hardware or is converted to the low-level machine language of the hardware computer.
  - More commonly, a language is implemented on a computer by a combination of translation and simulation as shown in the figure on the next slide.

The Operation of a Computer

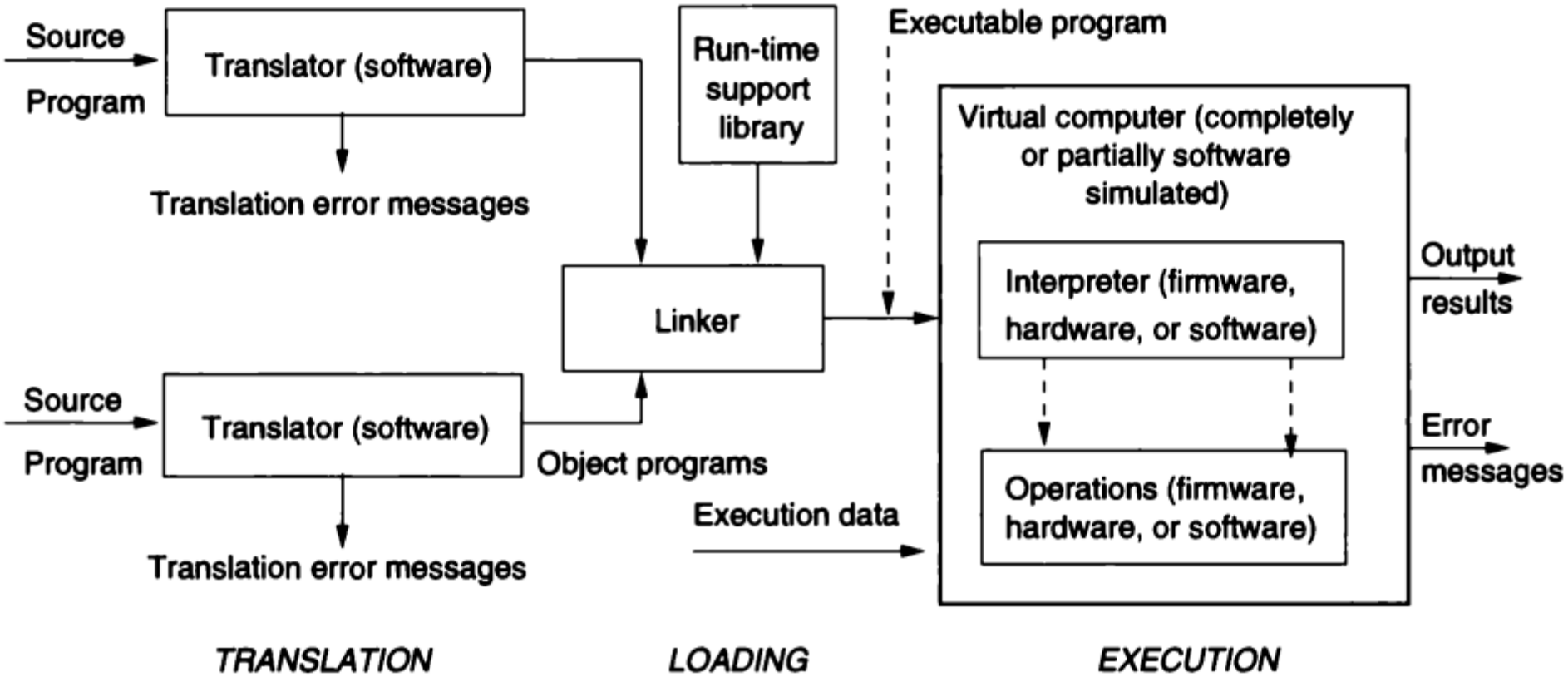

Figure Structure of a typical language implementation

The Operation of a Computer

- The basis for common division of languages (more precisely, language implementations) are those that are **compiled** and those that are **interpreted**. This division is based on whether the base representation of the program during execution is that of the machine language of the actual computer being used or not.
  - **Compiled Languages:** Programs in compiled languages are usually translated into the machine language of the actual computer being used before execution begins. For example, C, C++, FORTRAN, Pascal and Ada are compiled languages.
  - **Interpreted Languages:** The translator in such languages does not produce machine code for the computer being used. Instead, the translator produces some intermediate form of the program that is more easily executable than the original program from yet that is different from machine code. The interpretation procedure for execution this translated program form must be represented by software because the hardware interpreter cannot be used directly. Interpreted languages are slower than compiled languages.
  - Java changed some of these rules with Java compiler producing intermediate set of bytecode for Java virtual machine.

Virtual Computers

- A given computer might actually be constructed:
  1. Through a **hardware realization**, representing data structures and algorithms directly with physical devices.
  2. Trough a **firmware realization**, representing data structures and algorithms by microprogramming a suitable hardware computer.
  3. Through a **virtual machine**, representing data structures and algorithms by programs and data structures in some other programming language.
  4. Through some **combination** of these techniques, representing various parts of the computer directly in hardware, in microprograms, of by software simulation as appropriate.
- **Virtual Computers and Language Implementations:**
  - Three factors lead to differences among implementation of the same language.
  - First, each implementor has wide latitude in determining the virtual computer structures.

Virtual Computers

- Second, various hardware and software facilities are available in the underlying computer and the costs of their use.
- Third, the choices made by each implementor to simulate the virtual computer elements using the facilities provided by the underlying computer and to construct the translator so as to support these choices of virtual computer representations.

- **Hierarchies of Virtual Machines:**
  - The virtual machine that a programmer uses to create an application is in fact formed from a *hierarchy of virtual computers*.
  - At the bottom, there must, of course, lie an actual hardware computer. However, the ordinary programmer seldom has any direct dealing with this computer.
  - This hardware computer is successively transformed by layers of software (or microprograms) into a virtual machine.

Virtual Computers

- The second level of virtual computer (or the third if a microprogram forms the second level) is usually defined by the complex collection of routines known as operating system. This virtual computer is usually available to the implementor of a high-level language.
- The language implementor provides a new layer of software that runs on the operating-system-defined computer and simulates the operation of the virtual computer for the high-level language. The implementor also provides a translator for translating user programs into the machine language of the language-defined virtual computer.
- The hierarchy in fact does not end with the high-level language implementation. More levels of virtual machines might appear to the user on the WWW.
- Web virtual machines created by browsers (created using C language or equivalent) process web pages.
- At the highest level in this particular hierarchy is the Web application computer.

Virtual Computers

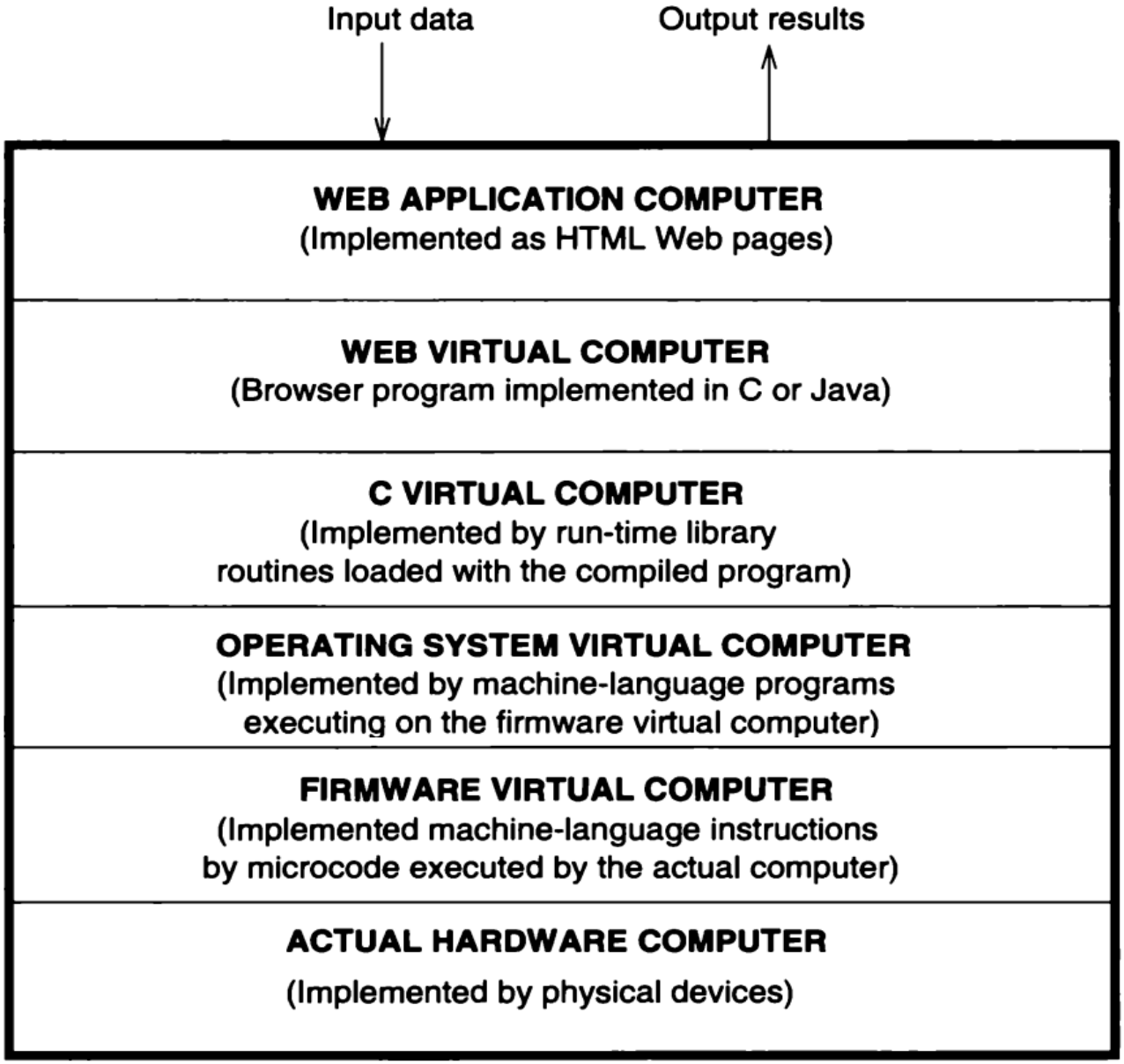

Figure . Layers of virtual computers for a Web application

Binding and Binding Time

- ***Binding*** of a program element to a particular characteristic or property is simply the choice of the property from a set of possible properties.
- The time during program formulation or processing when this choice is made is termed the ***binding time*** of that property for that element.
- There are many different varieties of bindings and binding times in programming languages.
- The properties of program elements are fixed either by the definition of the language or its implementation.
- **Classes of Binding Times:**
  - **Execution time (run time):**
    - Many bindings are performed during program execution. For example, bindings of variables to their values and binding of variables to particular storage locations.

Binding and Binding Time

- The two important subcategories may be distinguished.

  (a) On entry to a subprogram or block. In most languages, bindings are restricted to occur only at the time of entry to a subprogram or block during execution. For example, the binding of formal to actual parameters and the binding of formal parameters to particular storage locations may occur only on entry to a subprogram in C and C++.

  (b) At arbitrary points during execution. Some bindings may occur at any point during execution of a program. For example, in binding of variables to values through assignment, languages like LISP, Smalltalk, and ML permit such binding to occur at arbitrary points in the program.

- **Translation time (compile time):**
  - Three different subcategories may be distinguished.

    (a) Bindings have been chosen by the programmer. In writing a program, the programmer consciously produces some decisions concerning choices of variable names, types for variables, program statement structures, etc. that describe bindings during translation. The language translator creates the use of these bindings to decide the final structure of the object code.

Binding and Binding Time

(b) Bindings chosen by the translator. Some bindings are selected by the language translator without a direct programmer requirement. For example, the relative area of a data object in the storage designated for a phase is usually managed without knowledge or intervention by the programmer.

(c) Bindings chosen by the loader. A program generally includes multiple subprograms that should be combined into a single executable program. The translator generally binds variables to addresses within the storage designated for each subprogram. This storage should be assigned actual addresses within the physical computer that will execute the program.

- **Language implementation time:**
  - Some aspects of a language definition may be the same for all programs that are run using a particular implementation of a language, but they may vary between implementations. For example, the choice of a particular sequence of bits to represent 10 at execution time is usually made at language implementation time.
- **Language definition time:**
  - Most of the structure of a programming language is fixed at the time the language is defined. For example, set of allowable types for a variable X is often fixed at language definition time.