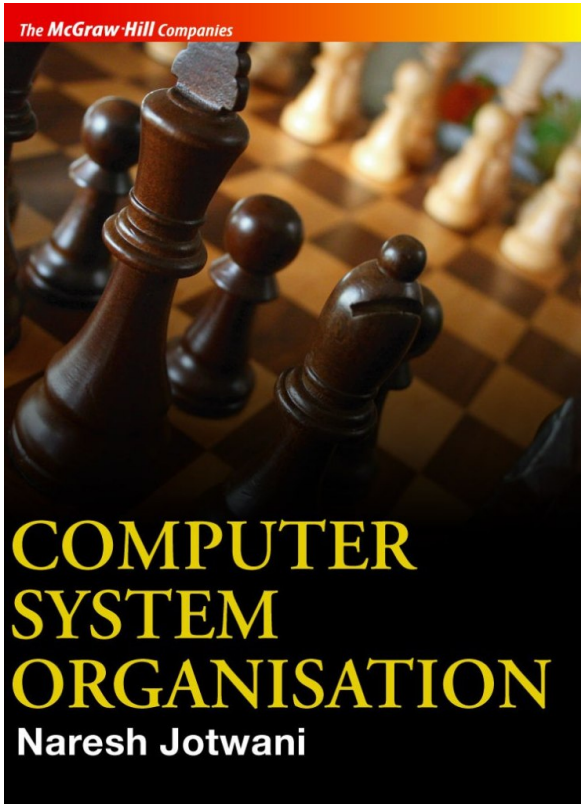
Mc Graw Hill Education

COMPUTER SYSTEM ORGANISATION

Naresh Jotwani

PowerPoint Slides



CHAPTER 15

SOFTWARE

Introduction

- Collection of programs available on a computer system.
- The word also refers collectively to any and all computer programs, as in ‘software industry’.

- Software consists of *application software* and *system software*.
  - Application software refers to programs which carry out some useful function on behalf of a user.
  - System software makes it possible for applications to be developed, stored, and executed on a single system or on a network of systems.

- This chapter provides a brief overview of system software, of the typical structure of application programs.

Operating System

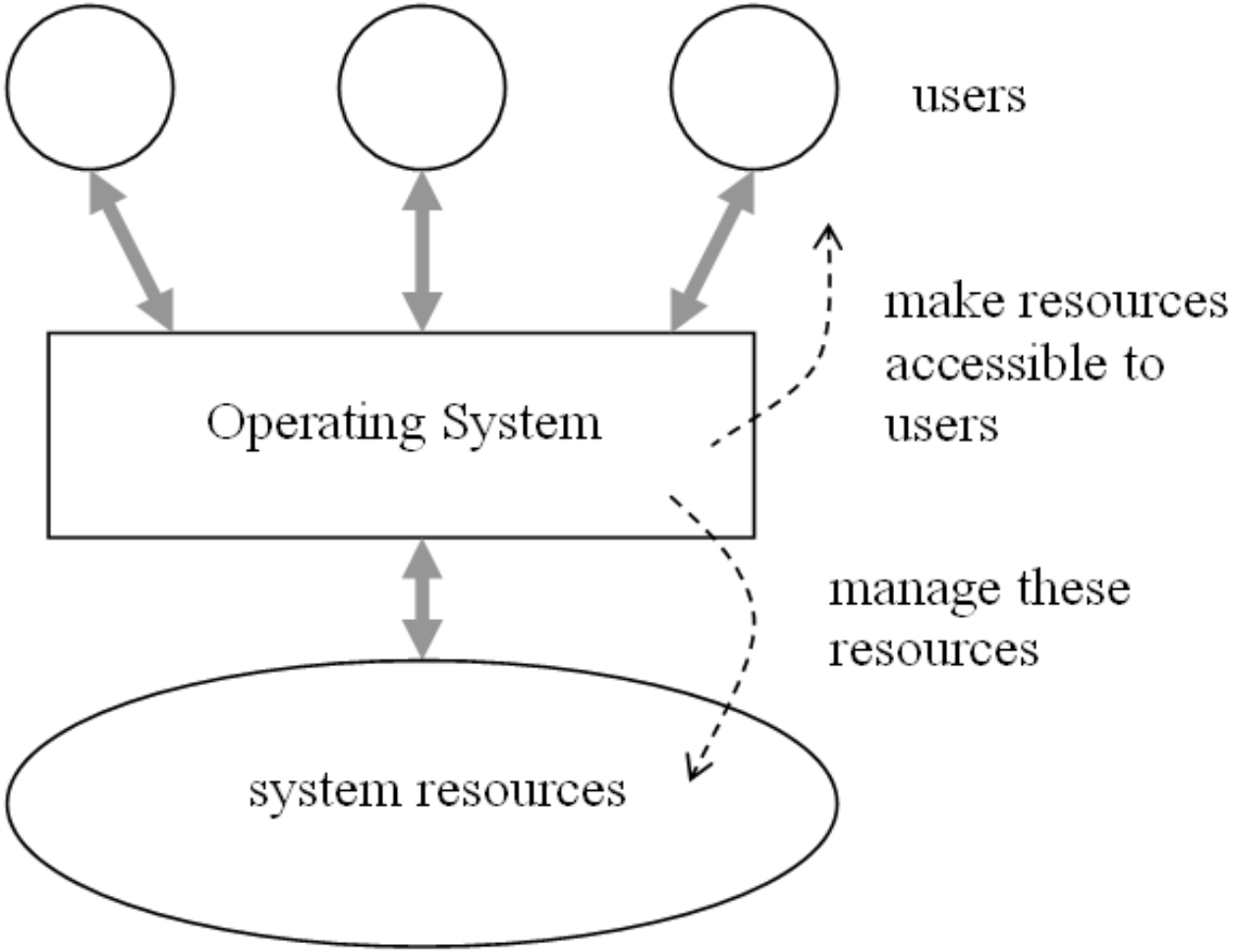
- Central and key component of system software, which intervenes between system resources and users, with the following objectives:

  (i) management of system resources for their sharing and effective utilization, and

  (ii) making system resources accessible to users, including software developers.

- System resources to be managed include processor(s), main memory, secondary storage, and communication and other input/output devices.

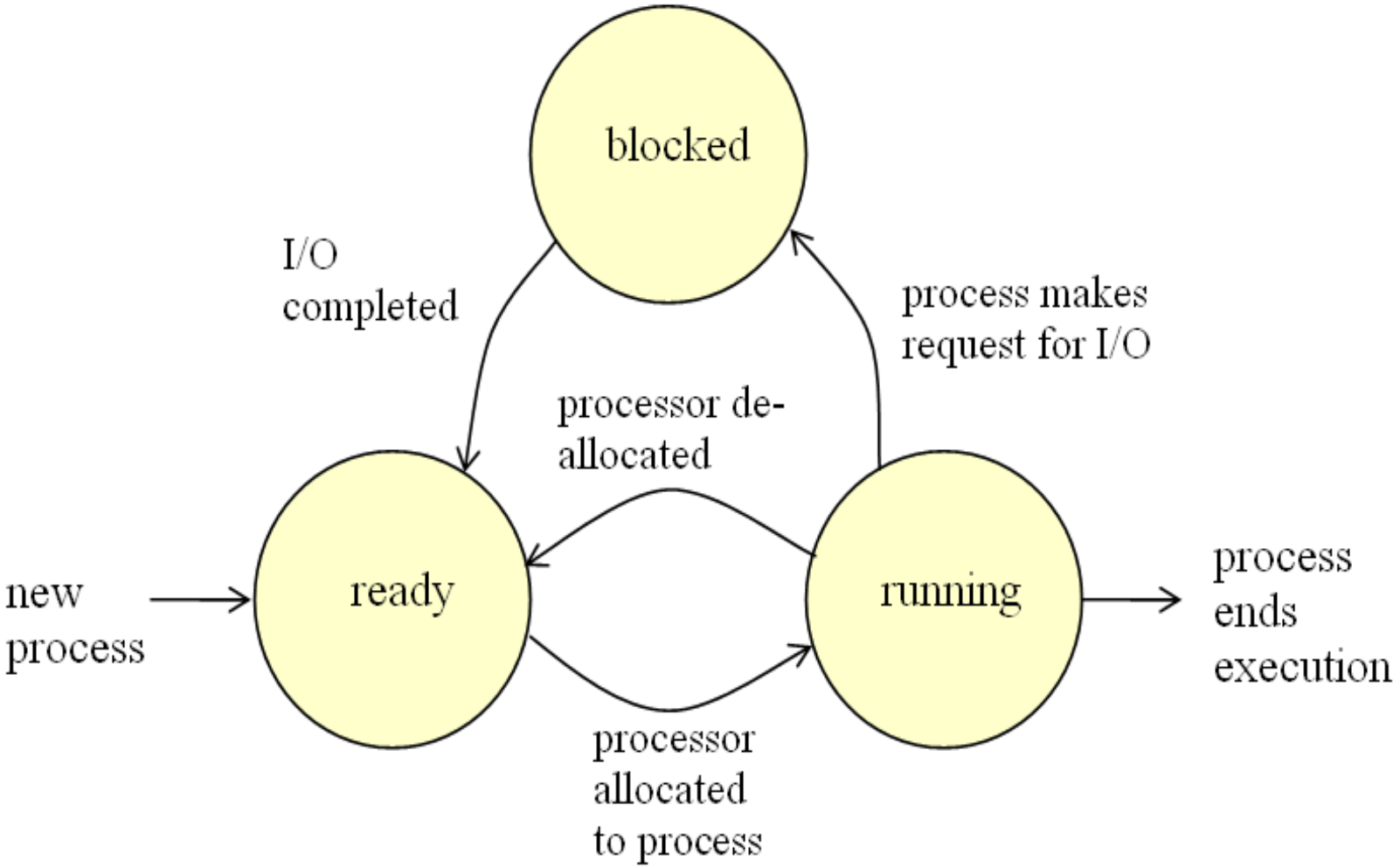
- The OS must manage system resources, satisfy user requirements, and provide a *user interface*.



- Figure illustrates the dual role of the operating system.

Processes

- A *process* is nothing but a running program. As its execution proceeds in time, a process makes explicit or implicit requests to the OS for system resources.
- As a process requests and makes use of resources, it goes through different *states*.
  - When a process is running on a processor – i.e. its machine instructions are being executed – it is in *running* state.
  - When it requests for a resource or an I/O operation, it is changed to *blocked* state, since it does not use processor time in this state.
  - On completion of I/O, the process changes to *ready* state – i.e. ready to run if a processor is available on which it can be *scheduled*.



- Figure shows process states, and the events causing a process to change state.

- The number of processes in *running* state cannot exceed the number of processors available on the system.
- Processes in *blocked* state are waiting for a resource to be allocated, or an I/O operation to complete.
- Processes in *ready* state are waiting to be *scheduled* to run on an available processor.
- OS maintains a queue of *ready* processes, and schedules them on available processor(s), so as to balance the twin requirements of (1) high resource utilization, and (2) acceptable response time to users.
- When a new process is created, it is in *ready* state – because it is given at the time of creation all the resources it needs to execute its first instruction.
-
- When process in *running* state executes the operating function *exit*, it is terminated.

Management of main memory

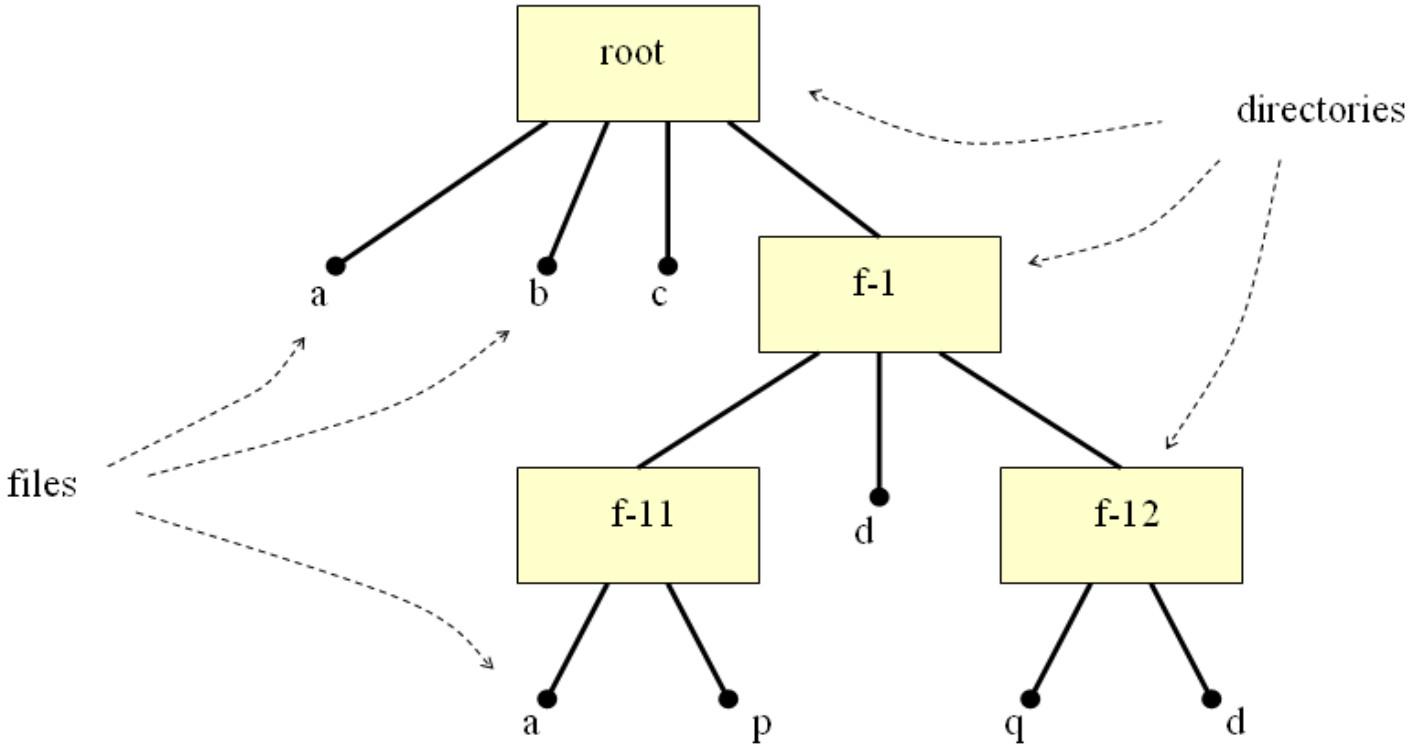
- Higher level functions, which cannot be performed by the MMU (memory management unit), are performed by the *memory manager* component of the OS, e.g.
  - Maintaining the logical-to-physical address map of each process,
  - Allocating/de-allocating memory to/from processes,
  - Taking appropriate action in response when MMU flags a memory access violation.
- MMU provides for efficient logical-to-physical address translation, while the *memory manager* of the OS ensures efficient overall utilization of main memory.

Files and directories

- Data and programs are stored on secondary storage in the form of *files*. A file is identified by its *filename*.
  - *File system* records several *attributes* of every file, such as file type, size, date of creation, owner's name, and so on.

- A computer system today may store thousands or tens of thousands of files on secondary storage.
  - Files are also stored by system and application software.
  - For efficient organization of all this information, related files are grouped together into a structure of *directories / sub-directories (folders)*.

- A *directory* is a special type of file which records information about other files and sub-directories.



- Figure shows a typical directory structure.

- Since directories contain sub-directories, the resulting structure becomes a *hierarchy* of directories.
- In practice the *directory hierarchy* grows both in depth and in breadth.

- File name must be unique within its directory – but two different directories may have files with the same name.
- Directory stores relevant attributes of each file – such as size, type, date of creation, owner name, etc.
- Also stored is the physical address of the file on the secondary storage device. On magnetic disk, physical address takes the form of track and sector numbers.
- *File system* maps the *logical* directory and file structure to the *physical* structure of secondary storage device.
- For any secondary storage device, this mapping is done in such a way as to satisfy the twin requirements of (1) effective utilization of storage space, and (2) efficient access to required files.

Input and output

- Input/output and storage devices for computer systems span a very wide variety.
- In general, every program must be able to use the input/output and storage devices on a system.
- Devices are made accessible to programs by providing abstract, higher level functions.
- Programs must be provided with standard interface for functions which is *device independent*.
- This ensures that different I/O or storage devices can be accessed by a program in a uniform manner.

- An important part of the operating system – the *input/output subsystem* – makes this possible.
- To correctly operate the various input/output and storage devices, this part of OS has the *device drivers*, including *interrupt handlers*, for each device.
- The job of a *device driver* is to initiate a device operation in response to a request by a running program.
- An *interrupt handler* takes care of any hardware interrupt received from the device, which may happen when the initiated operation completes successfully, or when an error is encountered during the operation.

Software Development

- In the very early days of computers, programs were written in machine language by specialists, who put in 1s and 0s into main memory using switches provided on the system.

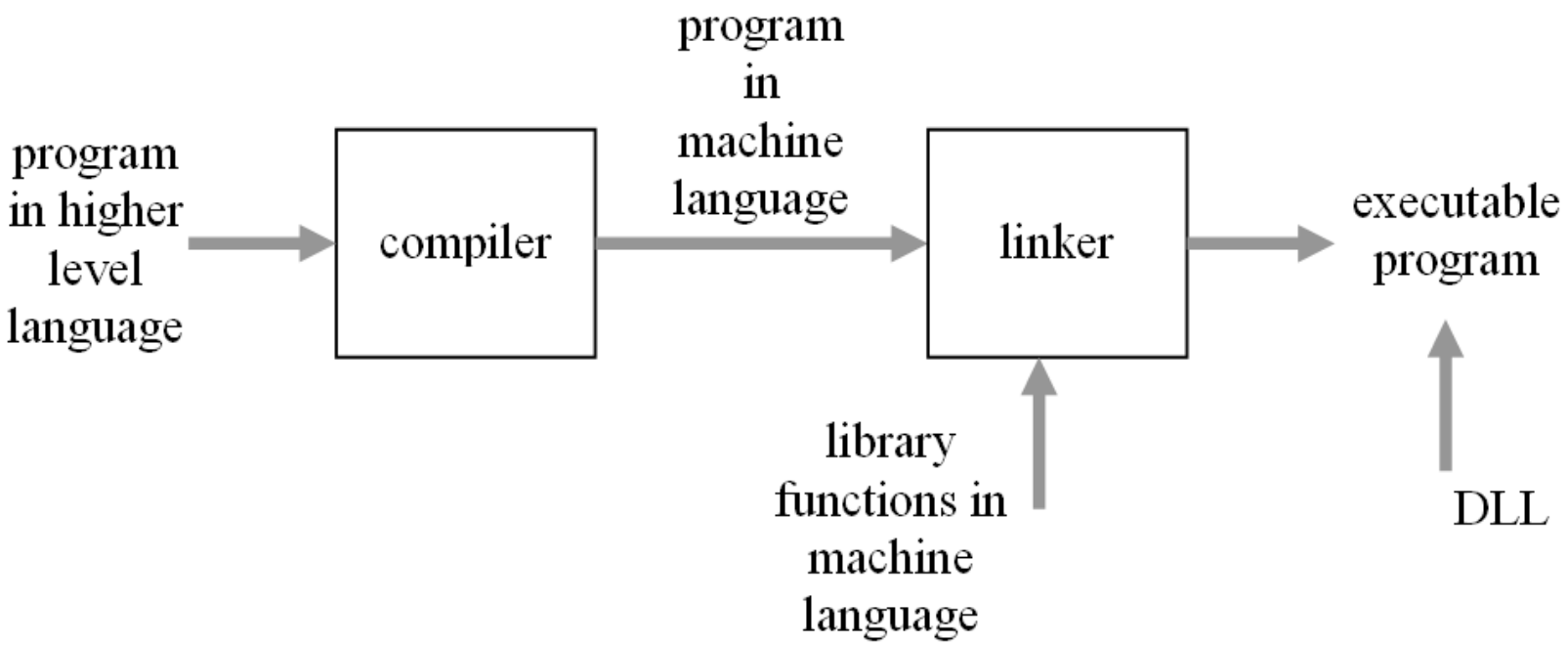
- Soon it became clear that binary machine language is not the right way to program computers. Since the early days, much progress has been made in the design of higher level programming languages.

- Modern concepts such as *object-oriented programming* arose from decades of thought given by computer scientists on how computer programs should be written.

- Consider the program below, which is written in programming language C:

```
/* Sample program in C */
#include <stdio.h>
main()
{
   int i;
   for (i=1;i<=10;i++)
     printf("%d\n", i);
}
```

- This program, when executed, will write out integers from 1 to 10 on the display screen.
  - The first line of the program is a *comment line*, which documents and communicates information about the program.
  - The remaining lines of this program determine the actions which will be taken when the program is run.
- But since this program is not in machine language, in its present form it cannot be run on a computer system.



- Before it is executed, the program must go through a couple of processing steps, as illustrated in the figure.
- The program seen earlier is known as *source program*.
- Another program, known as a *compiler*, converts the source program into an equivalent machine language program, known as the *object program*.

- But an object program in machine language cannot be executed directly on a computer system, because many functions useful to a running program are provided by the system software and the operating system.

- In the program above, one such function is being used. Note the program statement:

```
printf("%d\n", i);
```

- This statement invokes a function named *printf*, for an integer value to be written out to the display screen.

  - The user is not expected to learn how *printf* performs this action, but the user must know what actions this function performs.

- Within parentheses after function name, arguments to the function are specified.

- Since function *printf* is not written by the user, we do not see in the program above the instructions making up this function.
- But clearly the function must exist somewhere within the final executable program, so that it can be called.

- Many such functions useful in program development are provided with the OS, or with the compiler.

- These functions are made available to the software developer in the form of *function libraries*.

- A process known as *linking* combines the user's object program with the library functions used by the program, to produce the final executable version of the program.

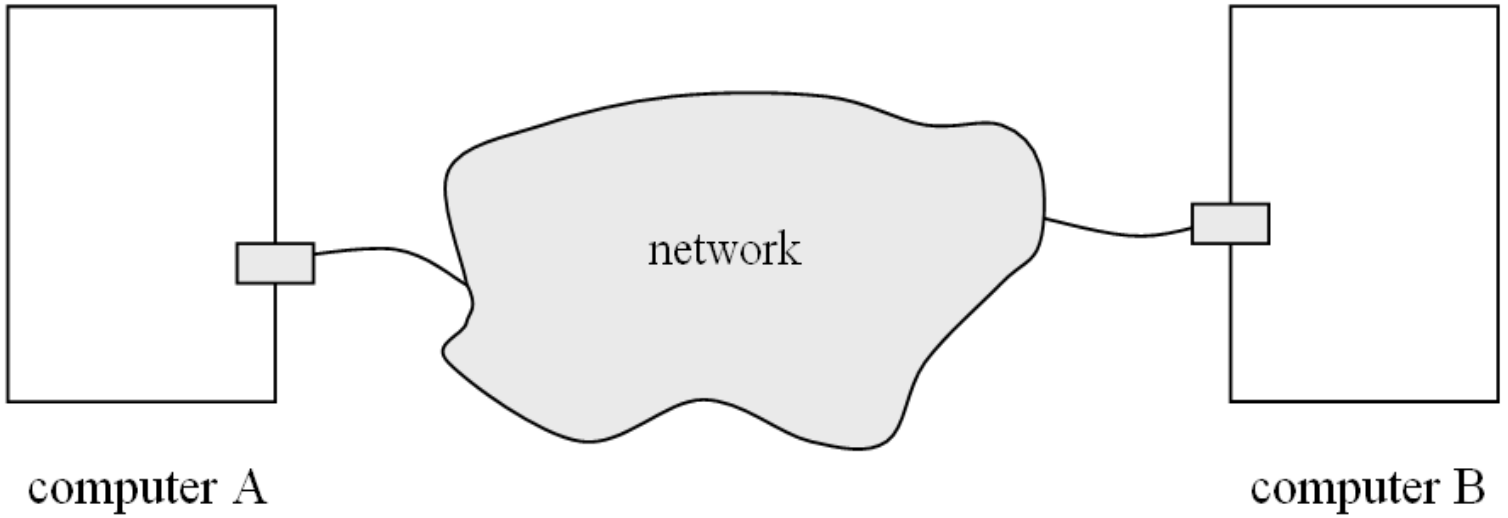
- Use of dynamically linked libraries of functions (DLLs) allows library functions to be linked with an executable program at the time of its execution.

- It is the final executable version of a program which runs when the user either double-clicks on an icon, or types in the program name from a command line.

- A new process is created by the operating system at that time, and necessary resources allocated to the process for it to begin execution, as explained earlier.

Computer Networks

- When multiple computers are connected together using communication adaptors and links – the resulting inter-connected structure is known as a *computer network*.
  - Two types of computer networks commonly in use today are *local area networks* – within an office building or a campus – and *wide area networks* which span the globe.
- Through the use of standard *protocols* *TCP* and *IP*, the Internet reaches across all such networks. Data is exchanged between computers in the form of *packets*, which may vary in size from a few tens of bytes to several kilobytes.
- On its way from source to destination, a packet needs to be *routed* over the intervening computers and links. For this purpose, the packet contains the *network address* of the destination computer which is to receive it.



- We see a 'virtual link' between A and B, which may be made of several intervening physical links in the network.
- This 'virtual link' – or *connection* – is set up on demand, and it lasts only for the duration of data exchange between the two computers.

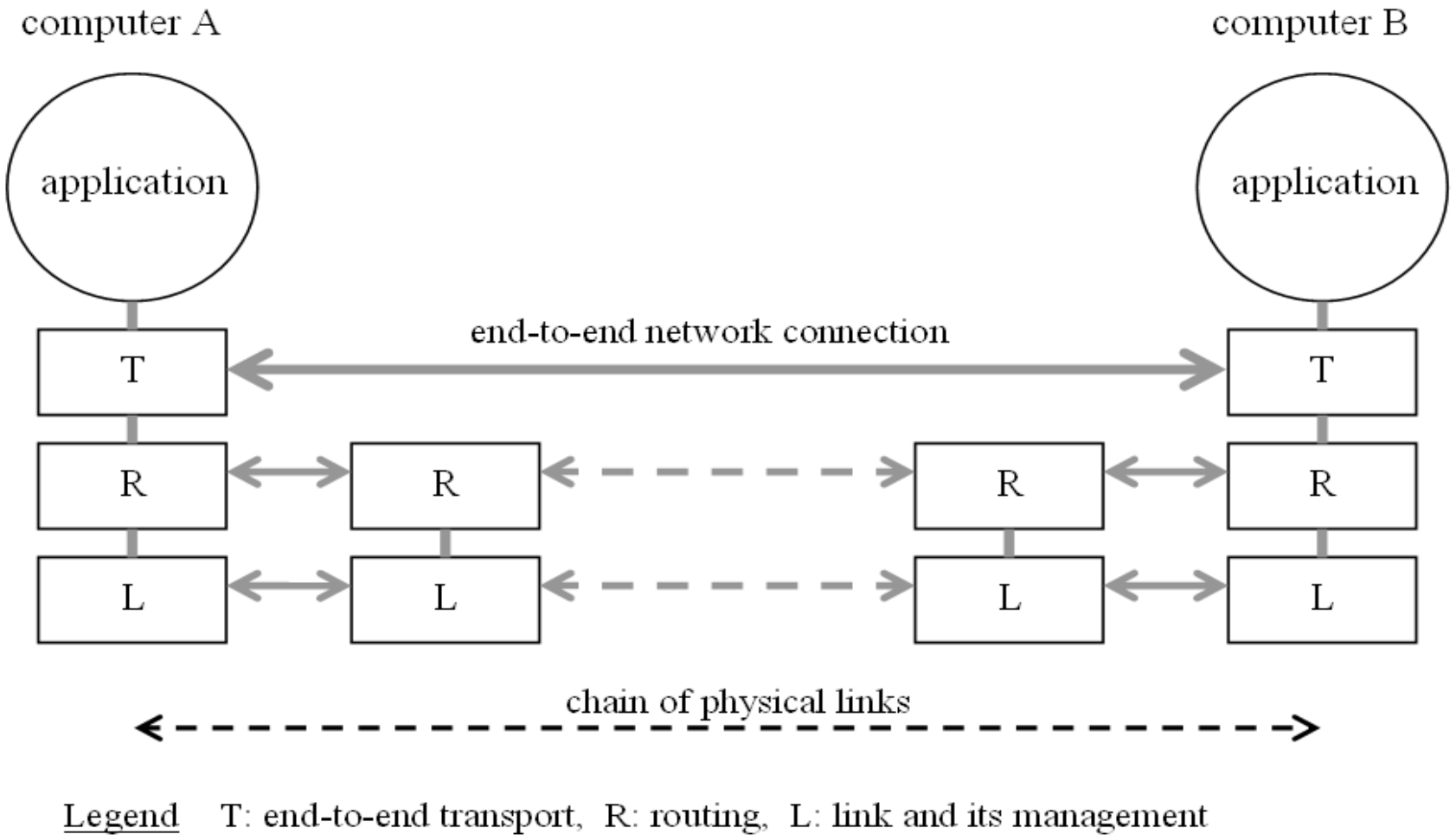
- Applications (such as a railway reservation system or e-mail) make use of such connections to perform their required functions.

- Such connections are known as *end-to-end* connections, because they stretch between two ends of an application running on the network.

- The component of the network which provides end-to-end communication is known as the *transport layer*.

- Applications make use of the functions of transport layer, and are therefore usually depicted as being 'above' the transport layer. The next figure depicts this relationship between an application and the transport layer.

- But, for the data transport function to operate faithfully and reliably, it must rely on 'lower level' *routing* functions of the network. These have been shown in the figure using rectangular boxes with letter 'R' inside



- Data packets originating at one end are *routed* through intervening nodes to the destination computer, and acknowledgement packets are routed in the reverse direction.

- Routing of a packet from one node to the next assumes:

  (a) there is a communication link between the two nodes, and

  (b) correct software runs at both ends of the link to send and receive data packets reliably

- In the figure, this function is shown as 'link and its management', in rectangular boxes with letter 'L' inside.

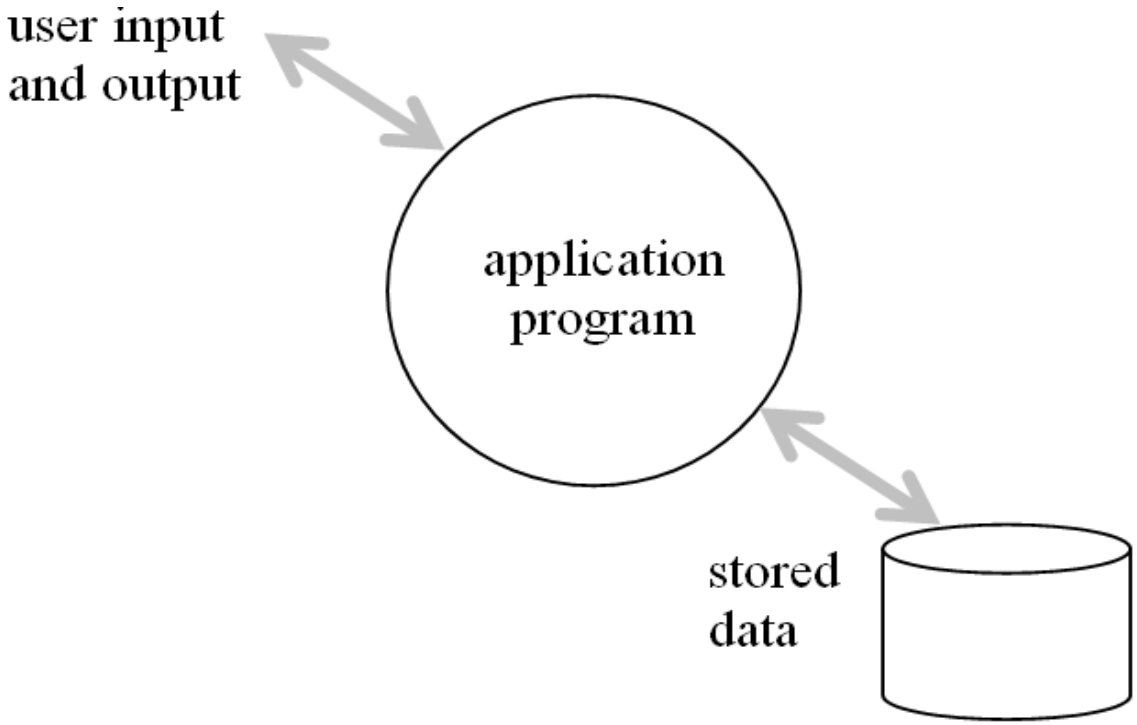
- Software plays a crucial role in *link management*, *routing*, and *end-to-end transport* functions. At each layer of functionality, the two communicating entities follow a common *protocol*.

- A network node is defined by the functions it performs, and not by its internal details. The Internet operates correctly because its millions of nodes follow their defined network protocols.

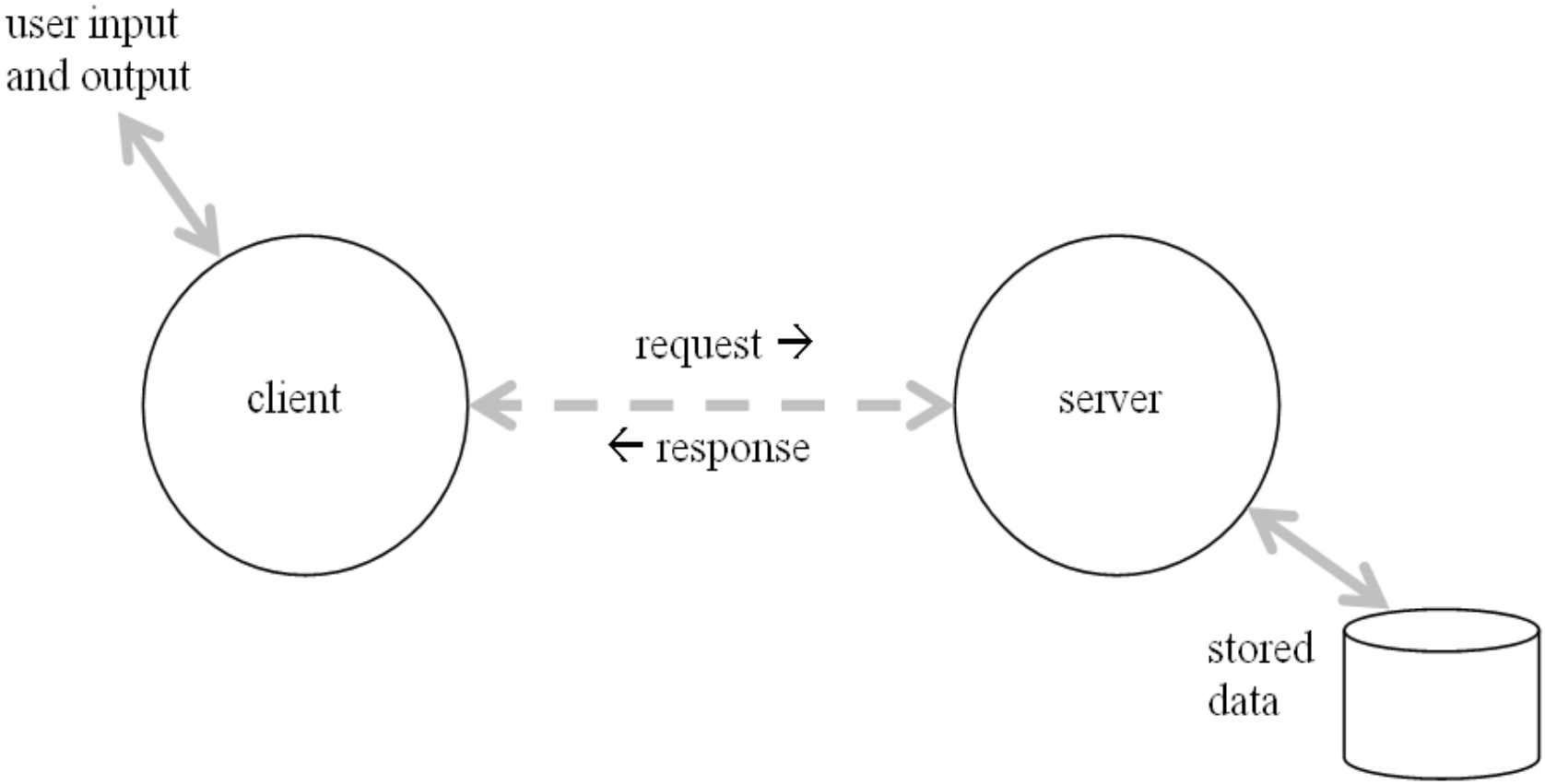
- At the application level, the most logical and convenient communication model is the asynchronous *message-passing model* which we have seen in Chapter 13.
- With the world-wide reach and sophistication of modern computer networks, applications have been developed with similar reach and sophistication.
- The *world-wide web* and its associated protocols enable the deployment of innovative *web applications*.
- With reliable and efficient data transport available, network applications today span a wide range – email, e-banking, online commerce, search engines, discussion forums, online storage, and many others.
- Today the network may appear to users as one giant multi-user and multi-functional system.

Applications

- The structure of a typical application program on a single computer system is shown in the next figure.
- One part of the program interacts with the user, as shown on the left, while another part accesses and stores data on secondary storage, as shown on the right.
- If an application is designed for multiple users operating on common data, then appropriate synchronization must be provided between user processes – to ensure that changes made by user processes to shared data do not result in inconsistent results.



- The figure above shows an application running on a single computer system. But, when an application is designed to work across a network, it has the typical structure shown in the next figure.



- Two processes are shown in the figure, with a network connection established between them.

- Process on the left is the *client* process. It accepts input from a user and provides output to the user.

- Process on the right is the *server* process, responsible for managing data on secondary storage and for making data available to client processes over the network.

- This is the *client-server model*, today the commonly used model for network applications.

Example

- A large application such as airline reservation may have millions of potential users across the world. At any time many users may be using the application, connecting as *clients* to the application *server*.

- On the *server* side, clusters of processors and disks may be employed to keep up with the demands of such large applications.

- *Message-passing* model is used – with *request messages* going from client process to server process, and *response messages* returning from server process to client process.

- Libraries of useful functions are available for developing such network-based applications.

- The most widely used protocol for web applications today is the *hypertext transfer protocol* (HTTP).

- *Hypertext markup language* (HTML) is a technique to enrich text with special markers – so as to define the displayed formats of documents stored and accessed using the world-wide web.

- Using HTTP protocol, a *web browser* program is able to access an HTML document from the *web server* on which the document is hosted.

- Within this basic model, an application running on a web server can define its user interactions using HTML, or one of its variants or extensions.

- Thus a simple web browser program on the client system becomes the common window through which web applications can be accessed.

- This explains the enormous power of this simple application level protocol.

SUMMARY

- Operating System
- Software development
- Computer networks
- Structure of applications