Distributed Systems

Mahmoud Abou El-Magd Soliman
Department of Computer Science
Faculty of Computers and Artificial Intelligence
Sohag University

Chapter Two

Architectures

Introduction

Distributed systems are often complex pieces of software of which the components are by definition dispersed across multiple machines. To master their complexity, it is crucial that these systems are properly organized. There are different ways on how to view the organization of a distributed system, but an obvious one is to make a distinction between the logical organization of the collection of software components and on the other hand the actual physical realization.

Introduction

The organization of distributed systems is mostly about the software components that constitute the system. These software architectures tell us how the various software components are to be organized and how they should interact. In this chapter we will first pay attention to some commonly applied approaches toward organizing distributed systems.

Architectural Styles

For our discussion, the notion of an architectural style is important. Such a style is formulated in terms of components, the way that components are connected to each other, the data exchanged between components. and finally, how these elements are jointly configured into a system. A component is a modular unit with well-defined required and provided interfaces that is replaceable within its environment.

Architectural Styles

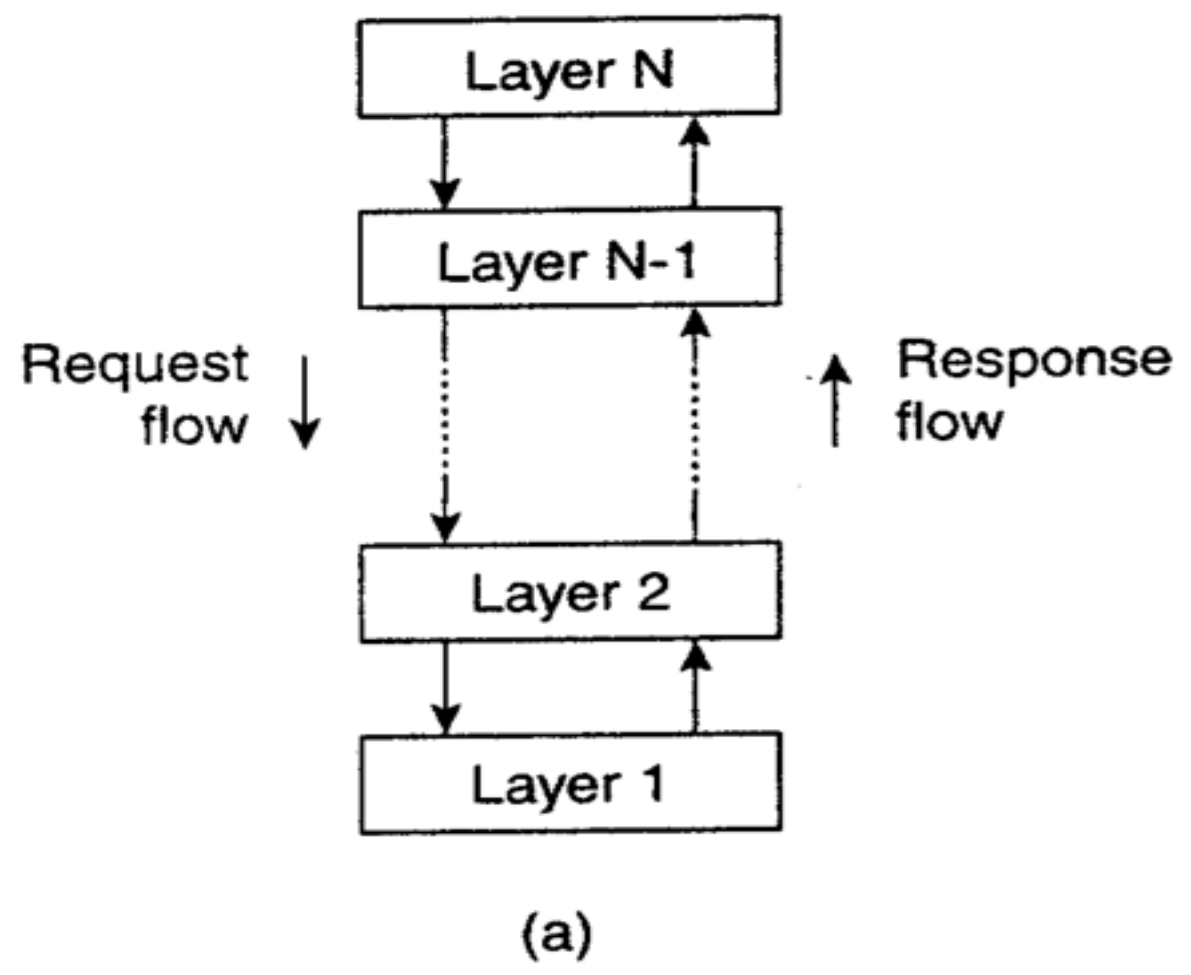
Several styles have by now been identified, of which the most important ones for distributed systems are:

1) Layered architectures
2) Object-based architectures
3) Data-centered architectures
4) Event-based architectures

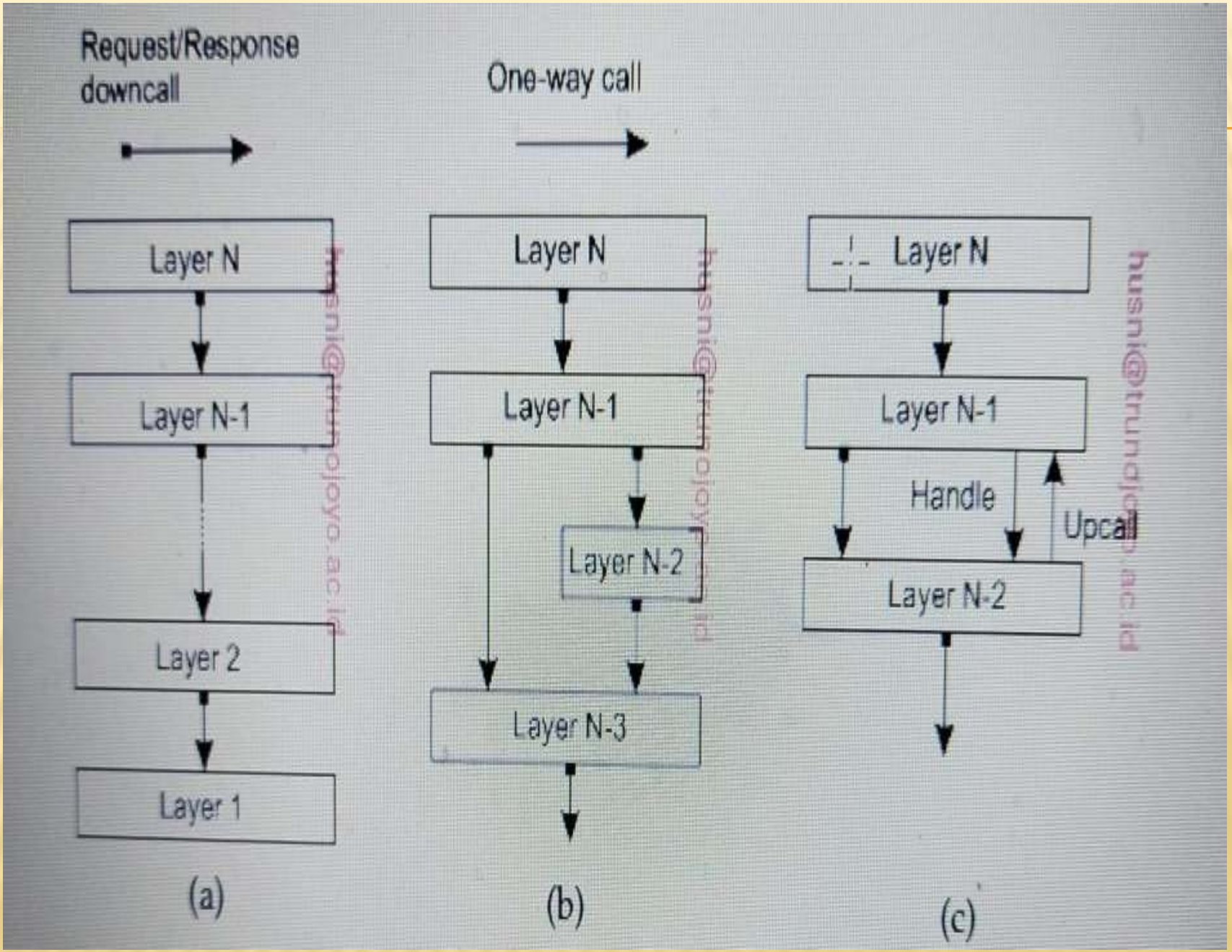
Architectural Styles

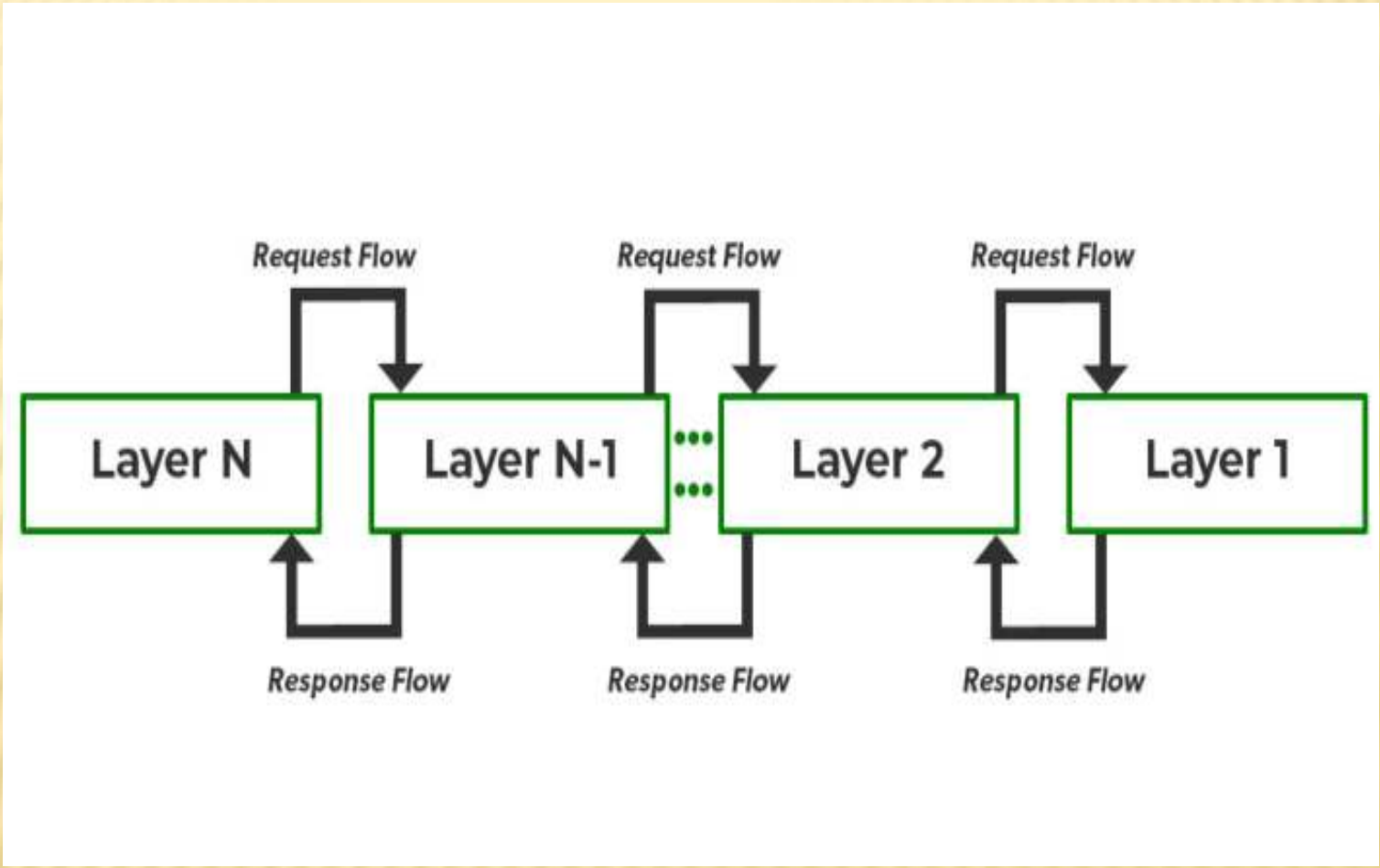
The basic idea for the layered style is simple: components are organized in a layered fashion where a component at layer N; is allowed to call components at the underlying layer Ni, as shown in Figure. This model has been widely adopted by the networking community. A key observation is that control generally flows from layer to layer: requests go down the hierarchy whereas the results flow upward.

Layered Style



(a): The layered architectural style.

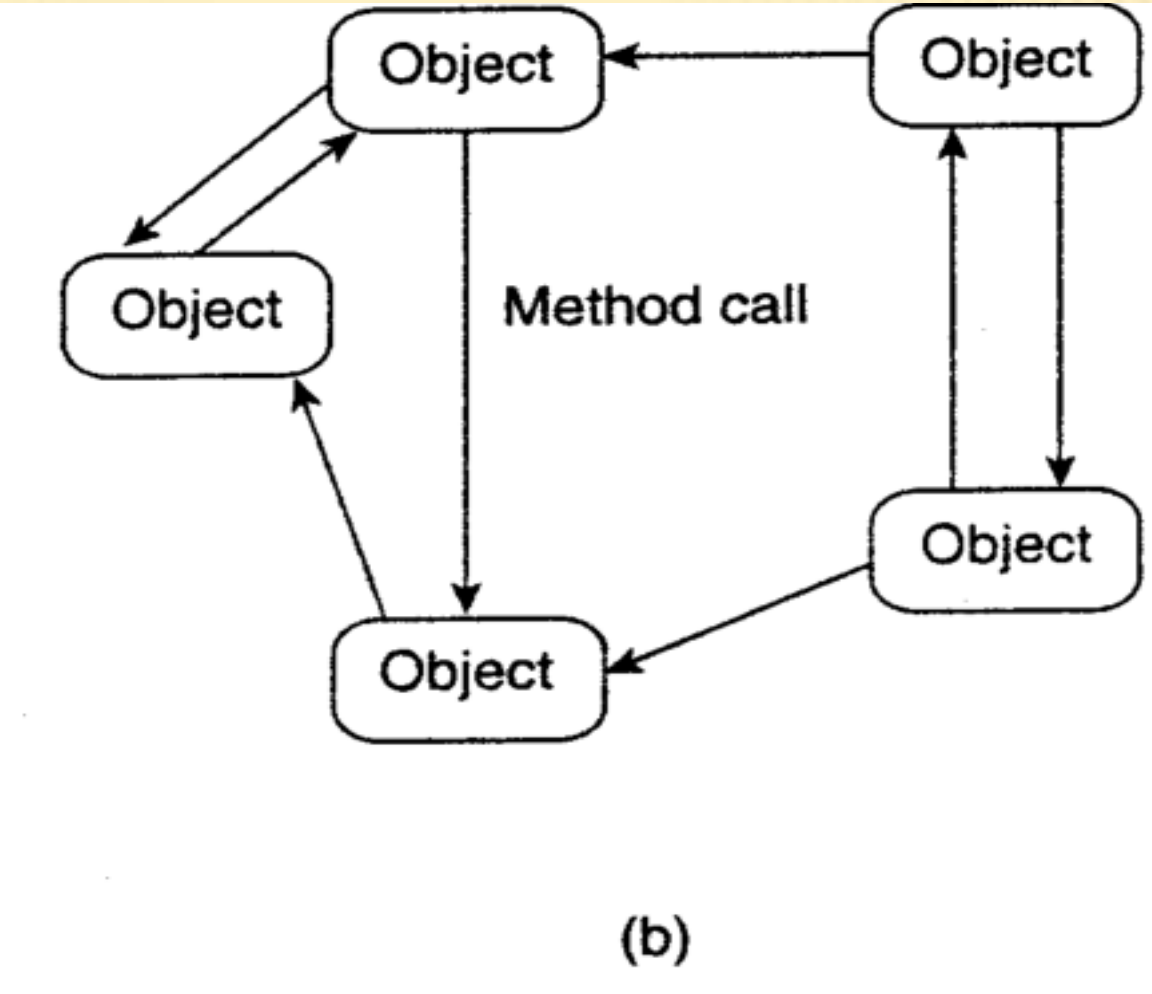

Request/Response
downcall
Layer N
Layer N-1
Layer 2
Layer 1
(a)
One-way call
Layer N
Layer N-1
Layer N-2
Layer N-3
(b)
Layer N
Layer N-1
Handle
Upcall
Layer N-2
(c)


Request Flow
Request Flow
Request Flow
Layer N
Layer N-1
Layer 2
Layer 1
Response Flow
Response Flow
Response Flow

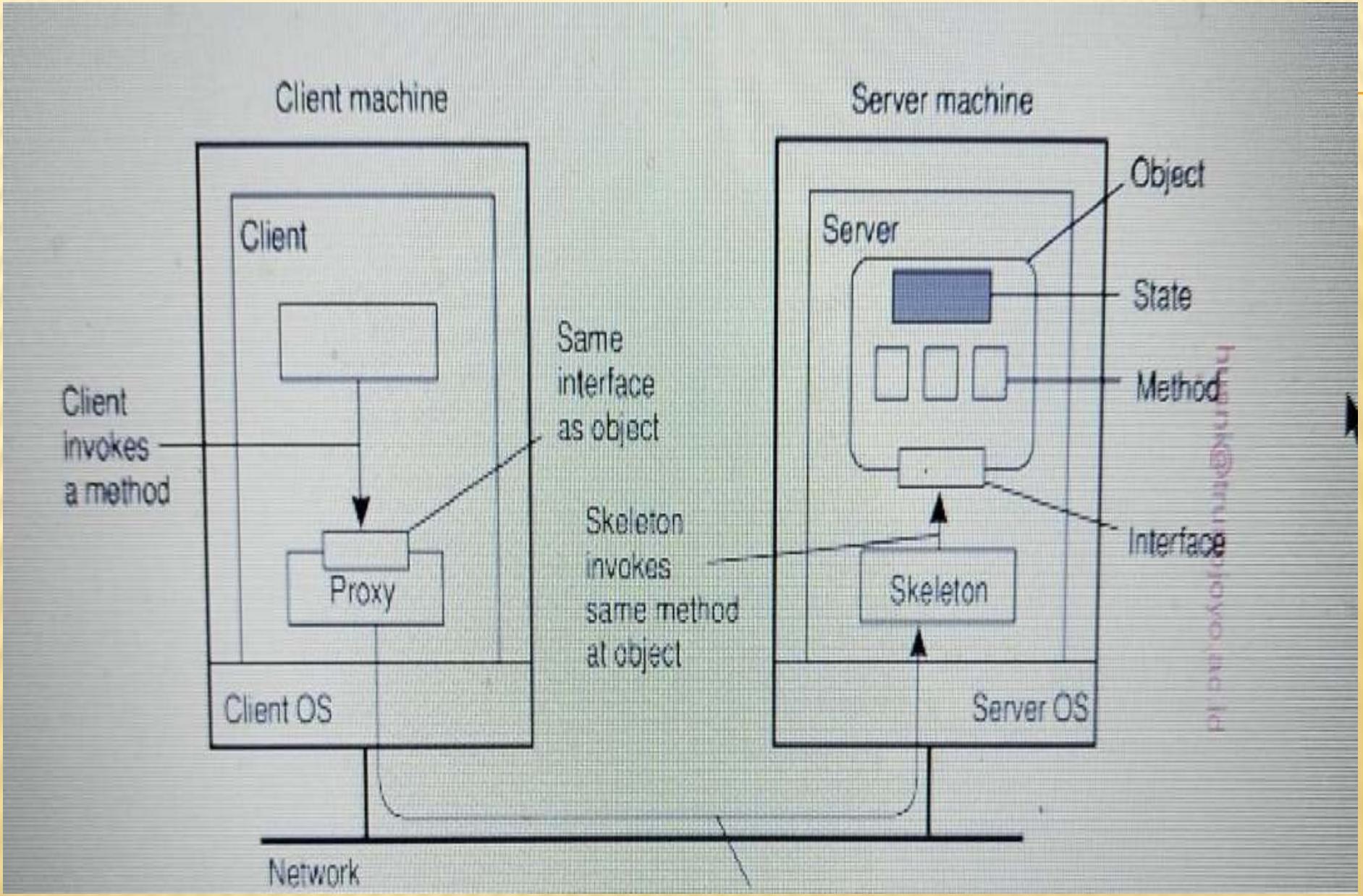
Object-Based Architectural Style

A far looser organization is followed in object-based architectures, which are illustrated in the Figure. In essence, each object corresponds to what we have defined as a component, and these components are connected through a (remote) procedure call mechanism. Not surprisingly, this software architecture matches the client-server system architecture. The layered and object-based architectures still form the most important styles for large software systems.

Object-Based Architectural Style



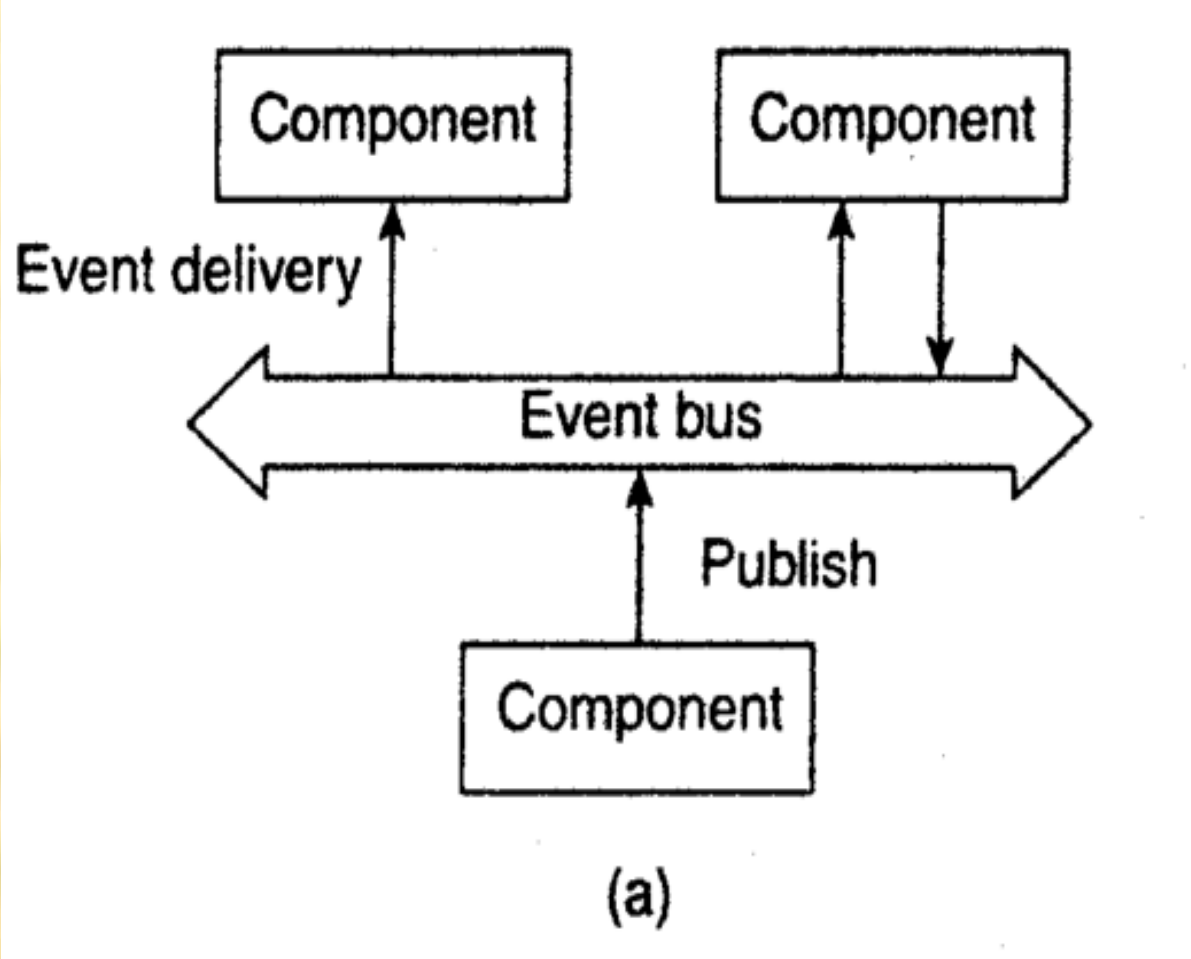
(b): The object-based architectural style


Client machine
Server machine
Client
Server
Object
State
Method
Interface
Same interface as object
Client invokes a method
Proxy
Skeleton invokes same method at object
Skeleton
Client OS
Server OS
Network

Event-Based Architectures

In event-based architectures, processes essentially communicate through the propagation of events, which optionally also carry data, as shown in the Figure. The basic idea is that processes publish events after which the middleware ensures that only those processes that subscribed to those events will receive them. The main advantage of event-based systems is that processes are loosely coupled. In principle, they need not explicitly refer to each other.

Event-Based Architectures

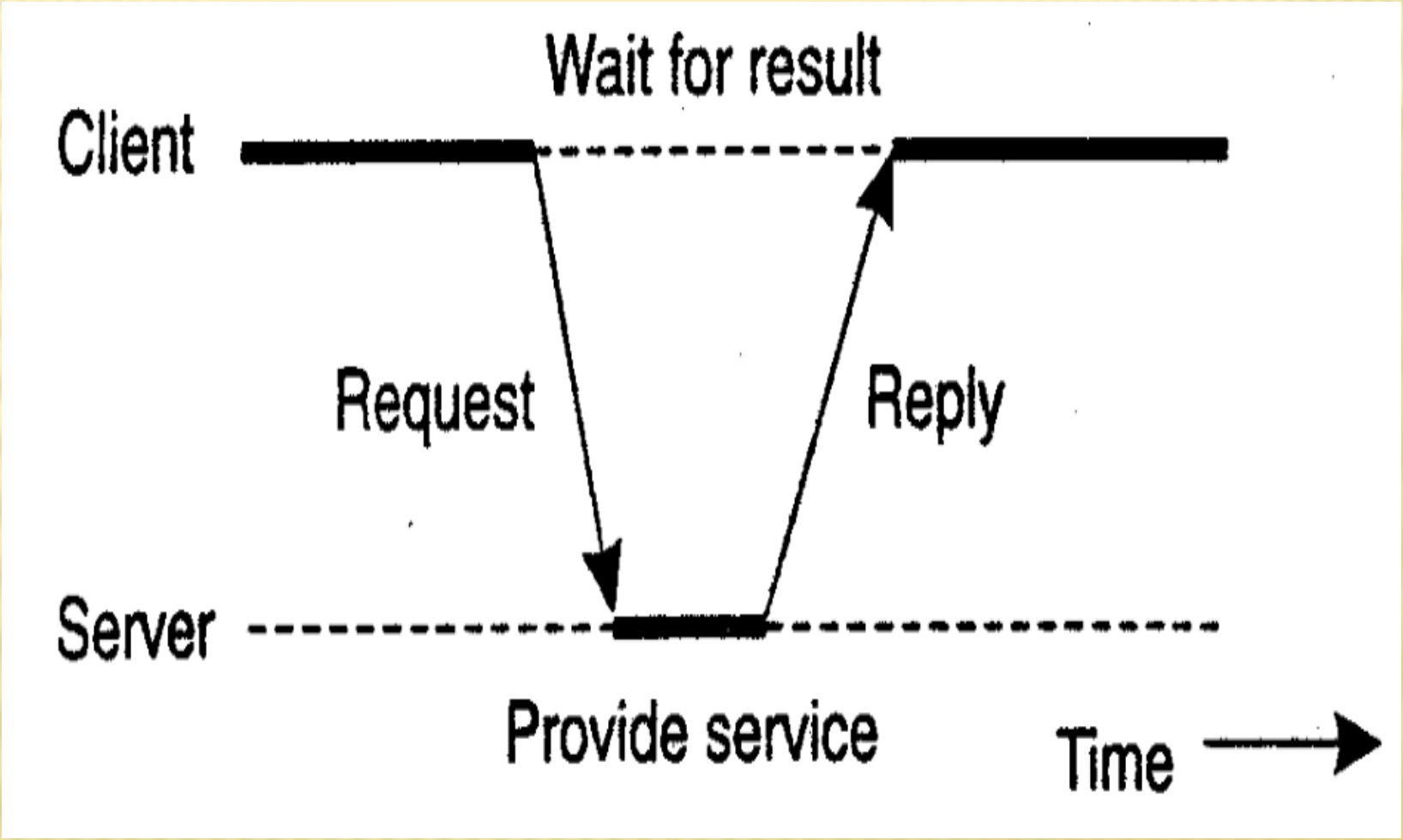


(a): The event-based architectural style.

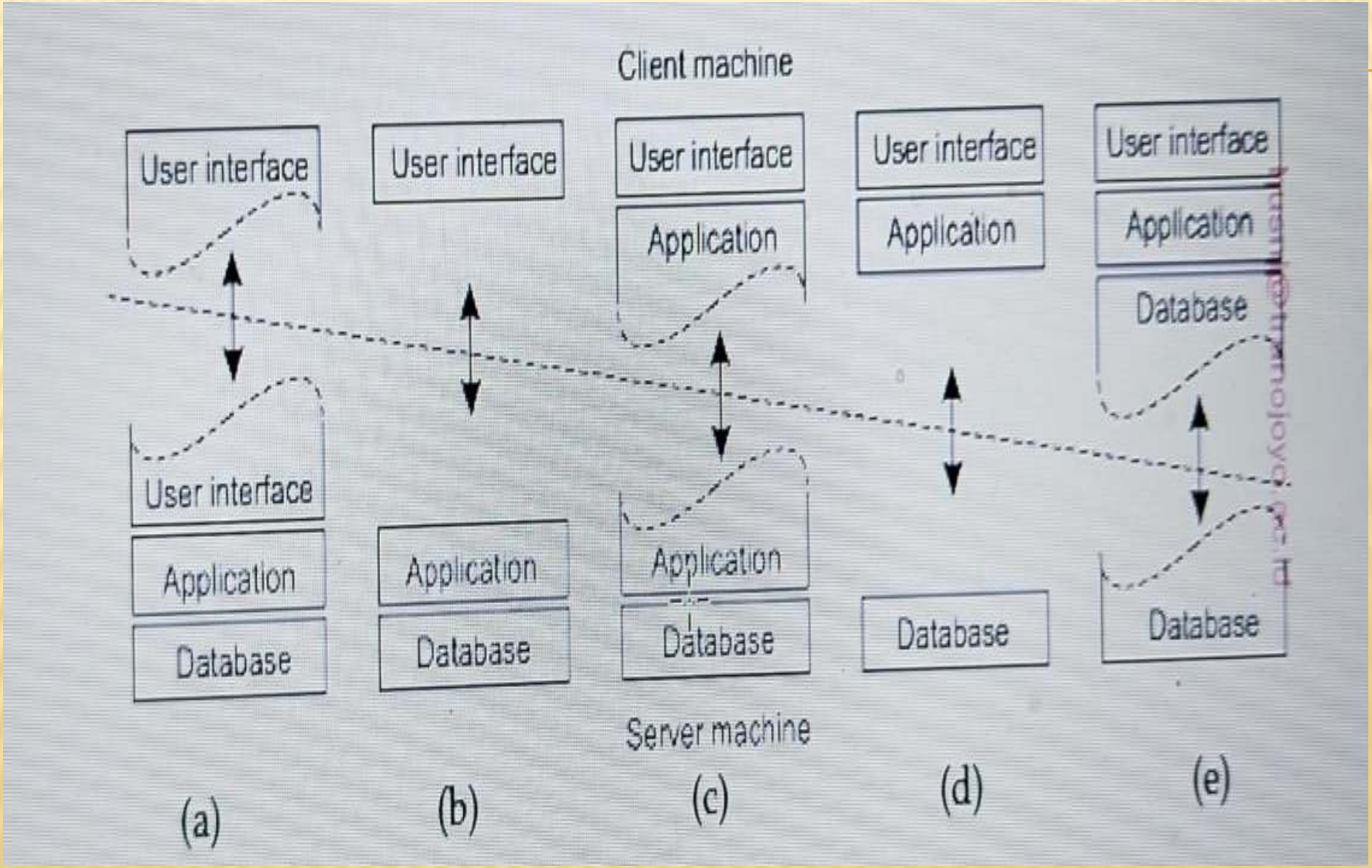
Centralized Architectures

In the basic client-server model, processes in a distributed system are divided into two (possibly overlapping) groups. A server is a process implementing a specific service, for example, a file system service or a database service. A client is a process that requests a service from a server by sending it a request and subsequently waiting for the server's reply. This client-server interaction, also known as request-reply behavior is shown in Figure.

Client-Server Model



General interaction between a client and a server.


Client machine
User interface
User interface
User interface
User interface
User interface
Application
Application
Application
Database
User interface
Application
Application
Application
Database
Database
Database
Database
Database
Server machine
(a)
(b)
(c)
(d)
(e)

Client-Server Model

Communication between a client and a server can be implemented by means of a simple connectionless protocol when the underlying network is fairly reliable as in many local-area networks. In these cases, when a client requests a service, it simply packages a message for the server, identifying the service it wants, along with the necessary input data. The message is then sent to the server. The latter, in turn, will always wait for an incoming request, subsequently process it, and package the results in a reply message that is then sent to the client.

Client-Server Model

Using a connectionless protocol has the obvious advantage of being efficient. As long as messages do not get lost or corrupted, the request/reply protocol just sketched works fine. Unfortunately, making the protocol resistant to occasional transmission failures is not trivial. The only thing we can do is possibly let the client resend the request when no reply message comes in. The problem, however, is that the client cannot detect whether the original request message was lost, or that transmission of the reply failed. If the reply was lost, then resending a request may result in performing the operation twice.

Client-Server Model

As an alternative, many client-server systems use a reliable connection-oriented protocol. Although this solution is not entirely appropriate in a local-area network due to relatively low performance, it works perfectly tine in wide-area systems in which communication is inherently unreliable. For example, virtually all Internet application protocols are based on reliable TCP/IP connections. In this case, whenever a client requests a service, it first sets up a connection to the server before sending the request. The server generally uses that same connection to send the reply message, after which the connection is torn down.

Client-Server Model

However, considering that many client-server applications are targeted toward supporting user access to databases, many people have advocated a distinction between the following three levels:

1) The user-interface level

2) The processing level

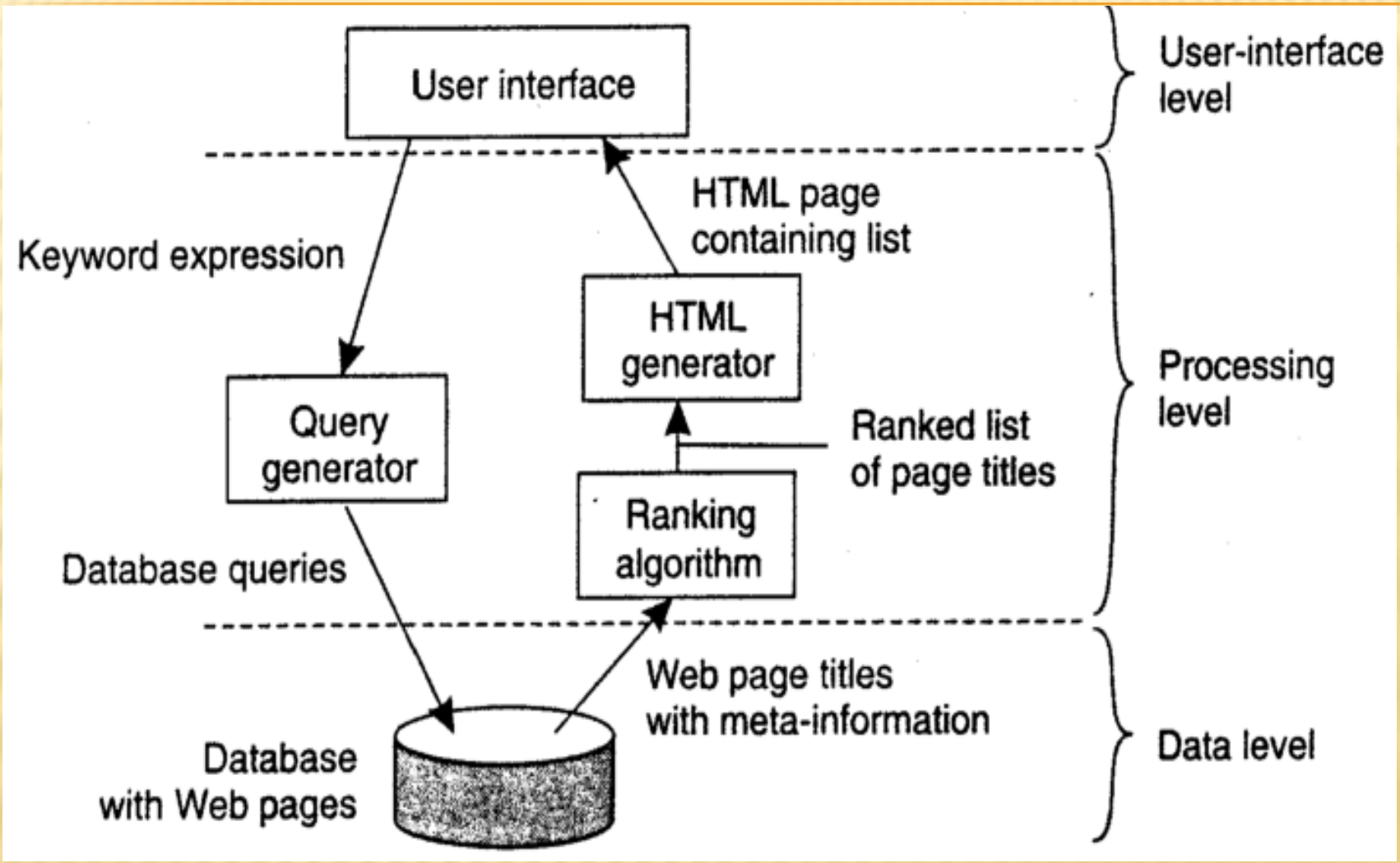
3) The data level

Client-Server Model

The user-interface level contains all that is necessary to directly interface with the user, such as display management. The processing level typically contains the applications. The data level manages the actual data that is being acted on.

Clients typically implement the user-interface level. This level consists of the programs that allow end users to interact with applications.

Client-Server Model



The simplified organization of an Internet search engine into three different layers.

Client-Server Model

Multitiered Architectures:

The distinction into three logical levels as discussed so far, suggests a number of possibilities for physically distributing a client-server application across several machines. The simplest organization is to have only two types of machines:

1) A client machine containing only the programs implementing (part of) the user-interface level

2) A server machine containing the rest, that is the programs implementing the processing and data level

Decentralized Architectures

Multitiered client-server architectures are a direct consequence of dividing applications into a user-interface, processing components, and a data level. **In many business environments, distributed processing is equivalent to organizing a client-server application as a multitiered architecture.** We refer to this type of distribution as vertical distribution. The characteristic feature of vertical distribution is that it is achieved by placing logically different components on different machines.

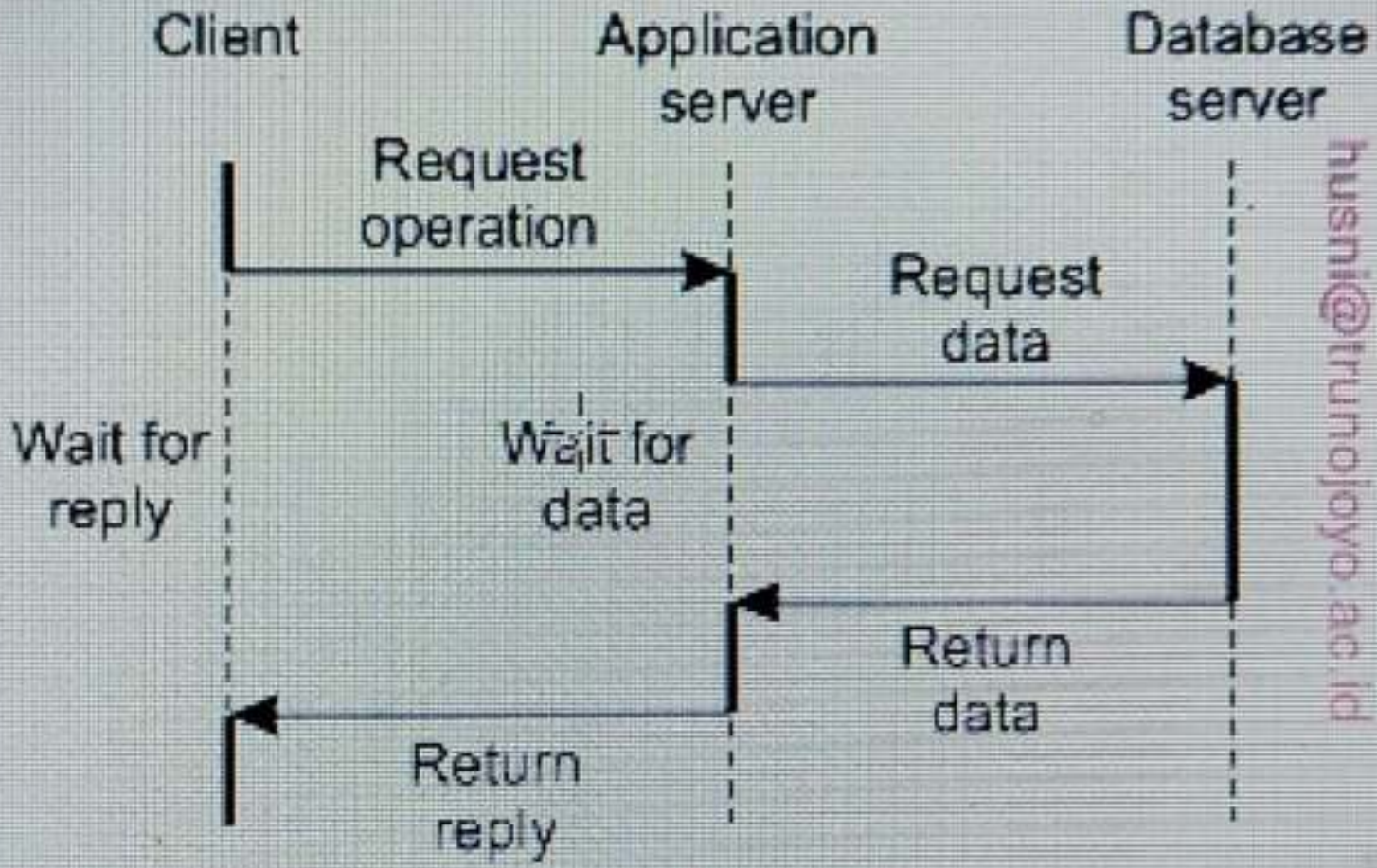


Figure 2.17: An example of a server acting as client.

Decentralized Architectures

Again, from a system management perspective, having a vertical distribution can help: functions are logically and physically split across multiple machines, where each machine is tailored to a specific group of functions. However, vertical distribution is only one way of organizing client-server applications.

Structured Peer-to-Peer Architectures

In a structured peer-to-peer architecture, the overlay network is constructed using a deterministic procedure (Hash Function). Likewise, nodes in the system are also assigned a random number from the same identifier space. Most importantly, when looking up a data item, the network address of the node responsible for that data item is returned. Effectively, this is accomplished by routing a request for a data item to the responsible node.

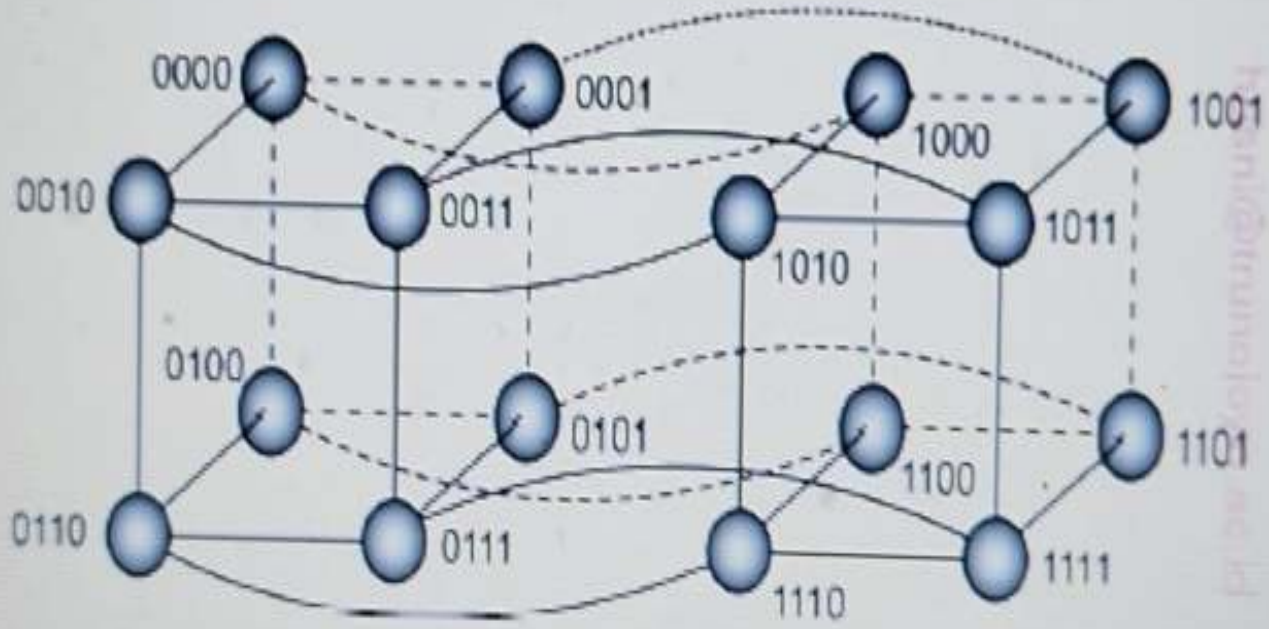


Figure 2.18: A simple peer-to-peer system organized as a four-dimensional hypercube.

Unstructured Peer-to- Peer Architectures

Unstructured peer-to-peer systems largely rely on randomized algorithms for constructing an overlay network. The main idea is that each node maintains a list of neighbors, but that this list is constructed in a more or less random way. Likewise, data items are assumed to be randomly placed on nodes. As a consequence, when a node needs to locate a specific data item, the only thing it can effectively do is flood the network with a search query.

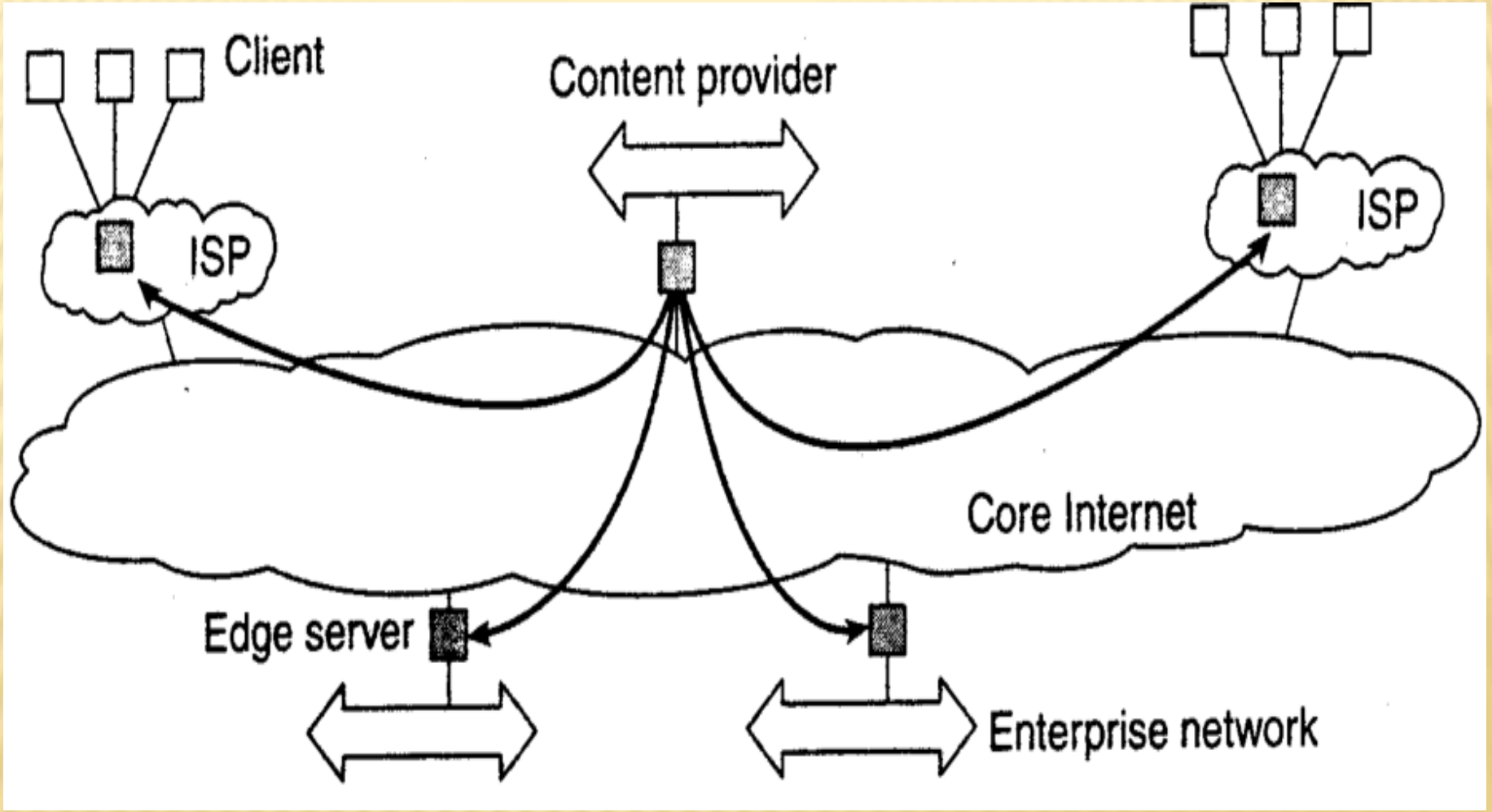
Hybrid Architectures

In this section we take a look at some specific classes of distributed systems in which client-server solutions are combined with decentralized architectures.

Edge-Server Systems:

An important class of distributed systems that is organized according to a hybrid architecture is formed by edge-server systems. These systems are deployed on the Internet where servers are placed "at the edge" of the network.

Edge-Server Systems



Viewing the Internet as consisting of a collection of edge servers.

Architectures Versus Middleware

When considering the architectural issues, we have discussed so far, a question that comes to mind is where middleware fits in. As we discussed in Chap. 1, middleware forms a layer between applications and distributed platforms. As shown in Fig. 1-1. An important purpose is to provide a degree of distribution transparency, that is, to a certain extent hiding the distribution of-data, processing, and control from applications. For example, many middleware solutions have adopted an object-based architectural style, such as CORBA.