DISTRIBUTED SYSTEMS
Principles and Paradigms

Second Edition

ANDREW S. TANENBAUM
MAARTEN VAN STEEN

Chapter 2
ARCHITECTURES

*** Modified by Prof. Rivalino Matias, Jr.**

Outline

- Architectural styles
- System architectures
- Architectures versus middleware

Architectural Styles (1)

- The effective implementation of DS implies to place software components in different physical locations (computers).
  - The basic idea is to organize the system into logically different components and distribute those components over various machines.
  - To do so, it is necessary to decide the architectural style adopted.

- An architectural style is formulated in terms of components, how they are connected to each other, their exchanged data, and how they are configured.

Architectural Styles (2)

- A component is a modular unit with well-defined interfaces.
- Components communicate to each other through connectors.
- A connector is a mechanism that mediates communication, coordination, or cooperation among components.
  - E.g., a connector can be formed by the facilities for remote procedure calls (RPC), message passing, or streaming data.
- Using both components and connectors, we can come to various configurations, aka, architectural styles for DS.

Architectural Styles (3)

Important styles of architecture for distributed systems:

- Layered architectures
- Object-based architectures
- Event-based architectures
- Data/Resource-based architectures

Architectural Styles (4)

Layered architectures:

- The basic idea of this style is simple, the components are organized in layers.
- A component in layer L_i has permission to call components in the underlying layer, L_{i-1}.
  - Only in exceptional cases will an upcall be made to a higher-level component.
- Layered style is used for client-server systems.
- This model has been widely adopted by the Network community.

Architectural Styles (5)

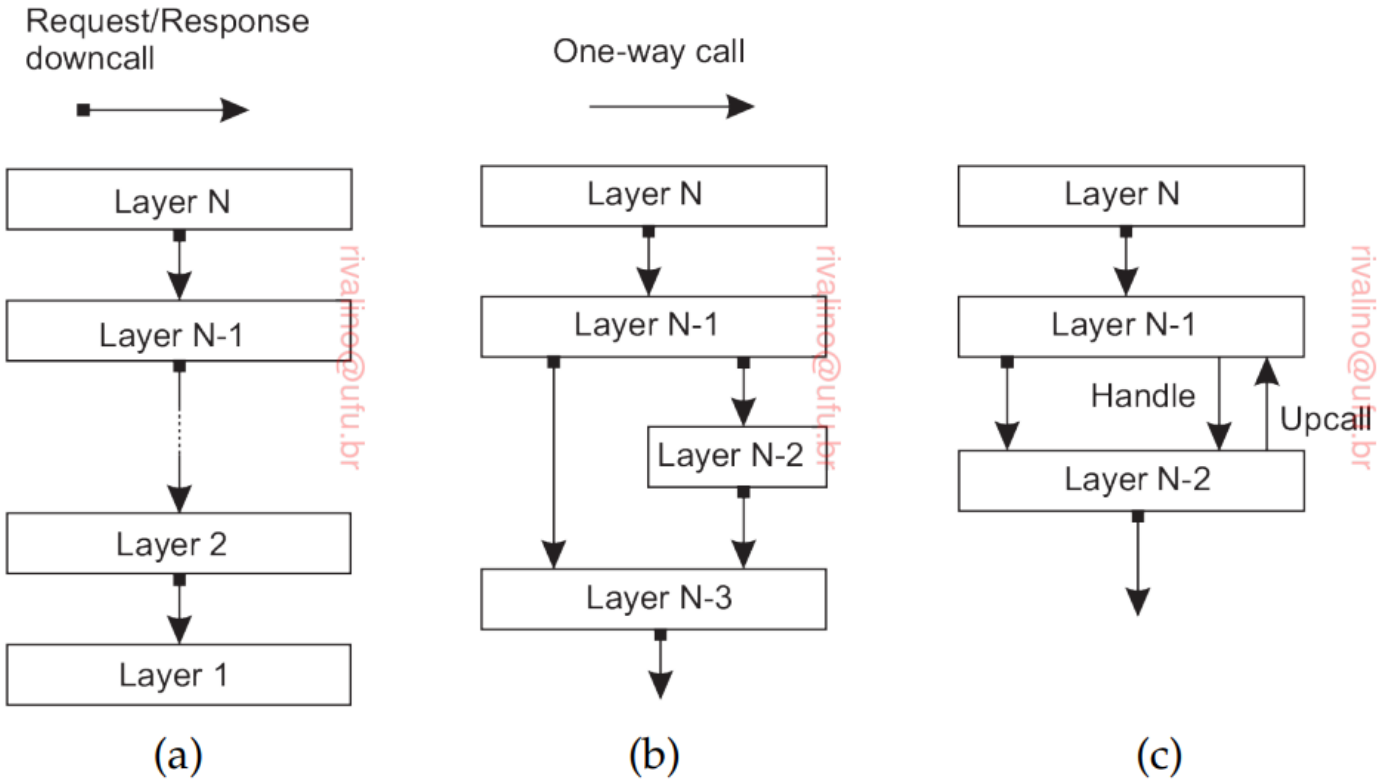


Figure 2.1: (a) Pure layered organization. (b) Mixed layered organization. (c) Layered organization with upcalls (adopted from [Krakowiak, 2009]).

Architectural Styles (6)

- Figure 2.1(a) shows a standard organization in which only *downcalls* are made.
  - This organization is commonly used in the case of network communication.
- Figure 2.1(b) shows the case in which the app. A uses the library L_{OS} to interface to an OS. A also uses library L_{math} that makes use of L_{OS} as well.
  - In Fig.2.1(b), A is implemented at layer N - 1, L_{math} at layer N – 2, and L_{OS} at layer N – 3.
- Figure 2.1(c) shows when it is convenient to have a lower layer do an upcall to its higher layer.
  - E.g., An OS signals the occurrence of an event, to which end it calls a user-defined routine (signal handler).

Architectural Styles (7)

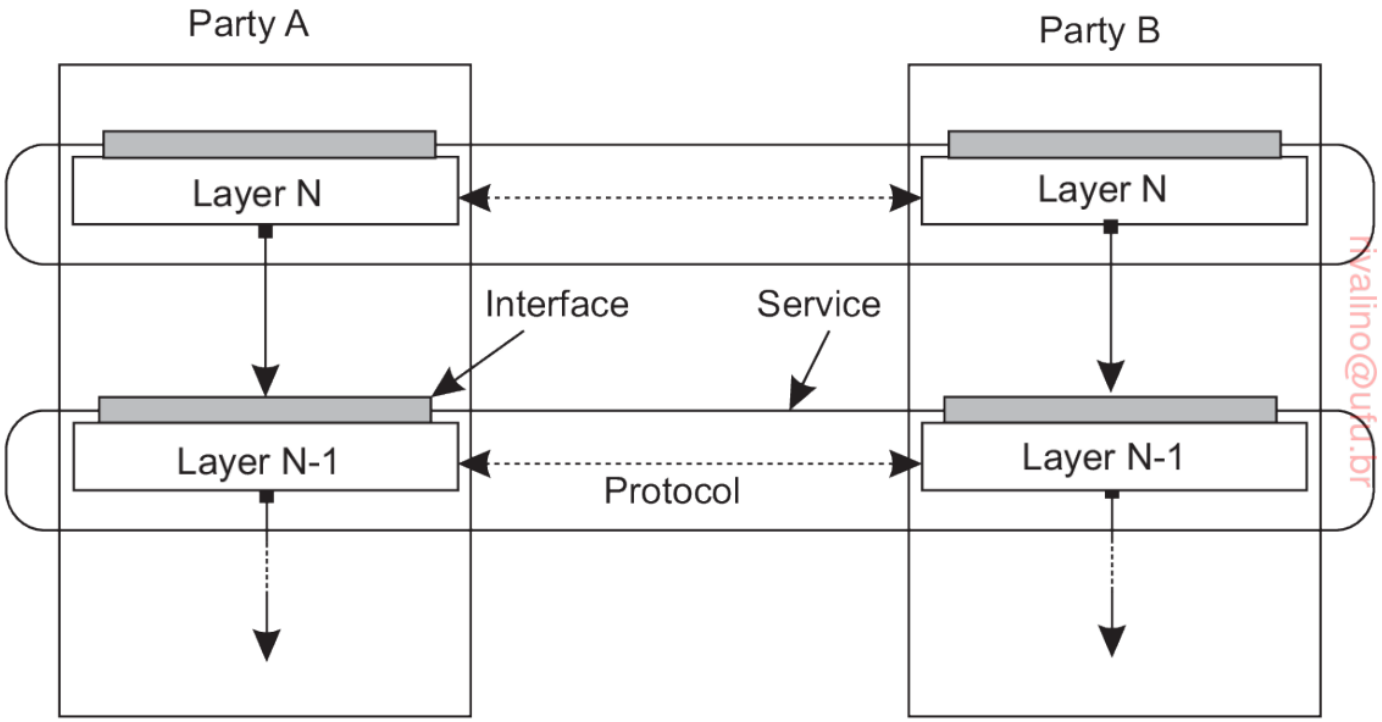


Figure 2.2: A layered communication-protocol stack, showing the difference between a service, its interface, and the protocol it deploys.

Architectural Styles (8)

- Another example of **layered architecture** can be seen in a large class of distributed applications supporting access to databases.
- In general, it is common to see three logical levels:
  - Application-interface.
  - Processing.
  - Data.
- Some practical examples:
  - Internet search engine.
  - System for stock brokerage.
  - Desktop packages (Office suites).

Architectural Styles (9)

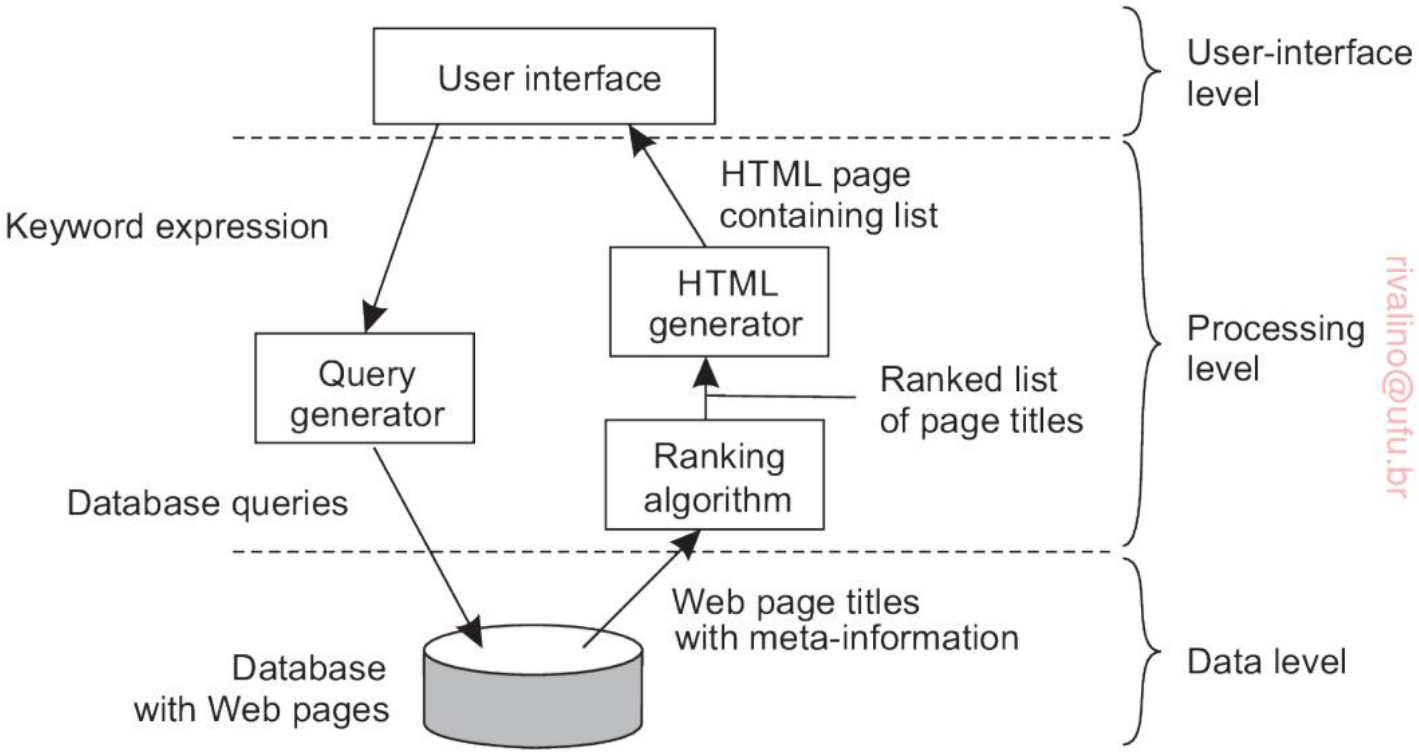


Figure 2.4: The simplified organization of an Internet search engine into three different layers.

Architectural Styles (10)

Object-based architectures:

- In this model, each object is a component, and these components are connected through a remote procedure (method) call mechanism.
- This model is attractive because it provides a natural way of encapsulating data (object's state) and their operations (object's methods).
- Distributed objects are placed at different locations (machines).
- Ultimately, this model is the foundation to implement service-oriented architectures (SOAs).

Architectural Styles (11)

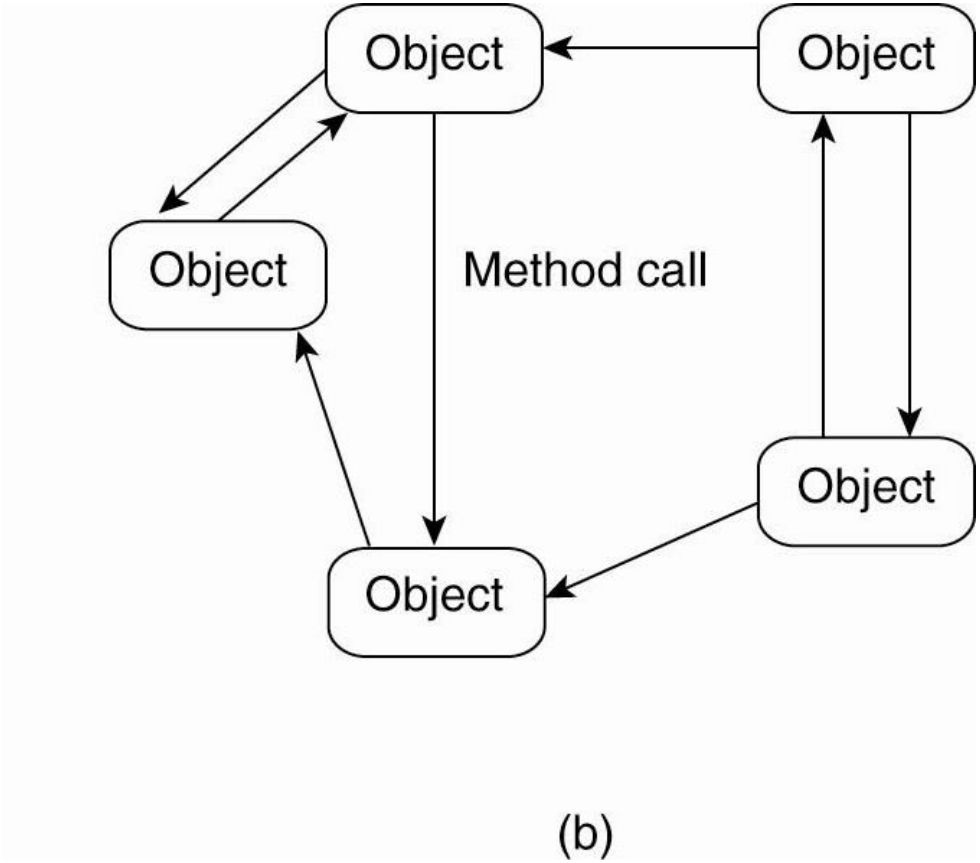


Figure 2-1. (b) The object-based architectural style.

Architectural Styles (12)

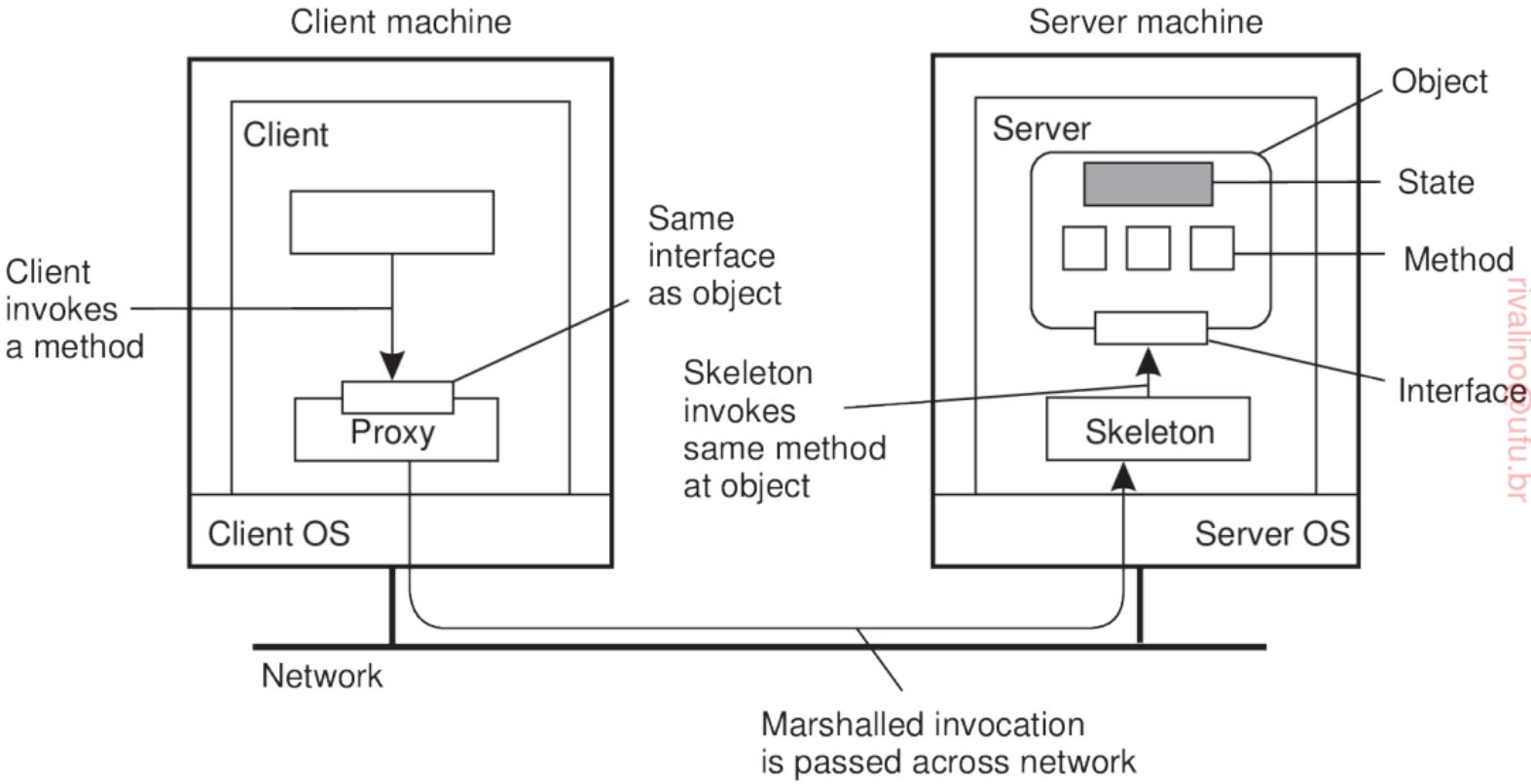


Figure 2.6: Common organization of a remote object with client-side proxy.

Architectural Styles (13)

Event-based architectures:

- In this model, processes communicate through the propagation of events, which optionally also carry data.
- For DS, event propagation has generally been associated with what are known as publish/subscribe systems.
- 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 this model is that processes are loosely coupled.

Architectural Styles (14)

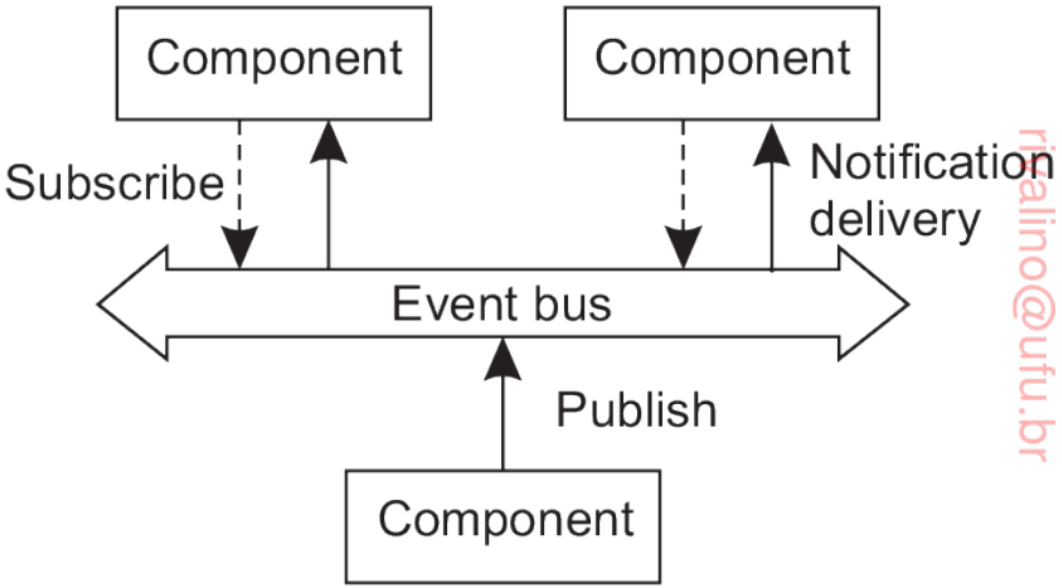


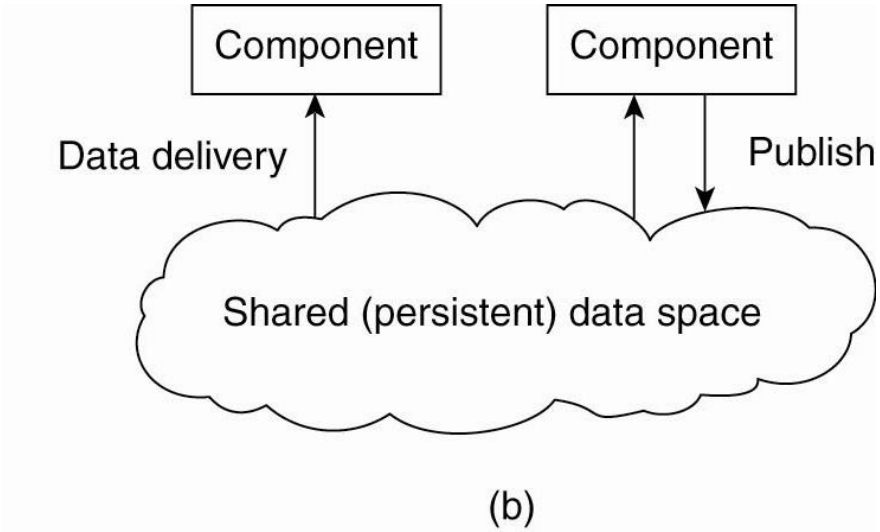
Figure 2-2. (a) The event-based architectural style and …

Architectural Styles (15)

Data/resource-based architectures:

- This model evolved around the idea that processes (or apps) communicate through a common (passive or active) repository.
  - Connecting various components can easily turn into an integration nightmare.
- The processes are decoupled in time; they need not both be active when communication takes place.
- One can view a DS as a huge collection of resources individually managed by components.
  - Resources can be added or removed by remote applications, and likewise can be retrieved or modified.

Architectural Styles (16)



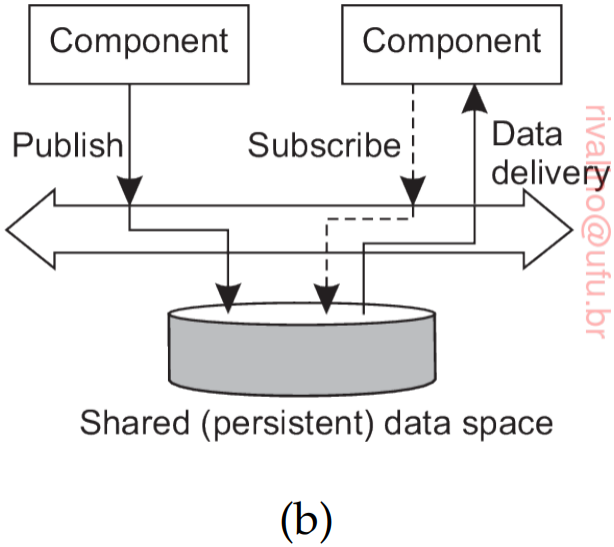


Figure 2-2. (b) The shared data-space architectural style.

System Architectures (1)

- Now let's to take a look at how many DSs are actually organized, by considering where software components are placed.
  - **Architectural styles** are related to the logical organization of the software components in a system.
  - **System architectures** are related to the physical location of software components (where they are placed).
- There are three main types of system architectures:
  - Centralized
  - Descentralized
  - Hybrid

System Architectures (2)

Centralized architectures:

- Despite the lack of consensus on many distributed systems issues, there is one issue that researchers and practitioners agree upon.
  - That thinking in terms of *clients* that request services from *servers* helps us understand and manage the complexity of DSs.

System Architectures (3)

Simple client-server architecture:

- In this model, processes are divided into two groups.
  - A **server** is a process implementing a service, e.g., file system service, database service, web service, etc.
  - 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 (C-S) interaction is also known as request-reply.

Centralized Architectures

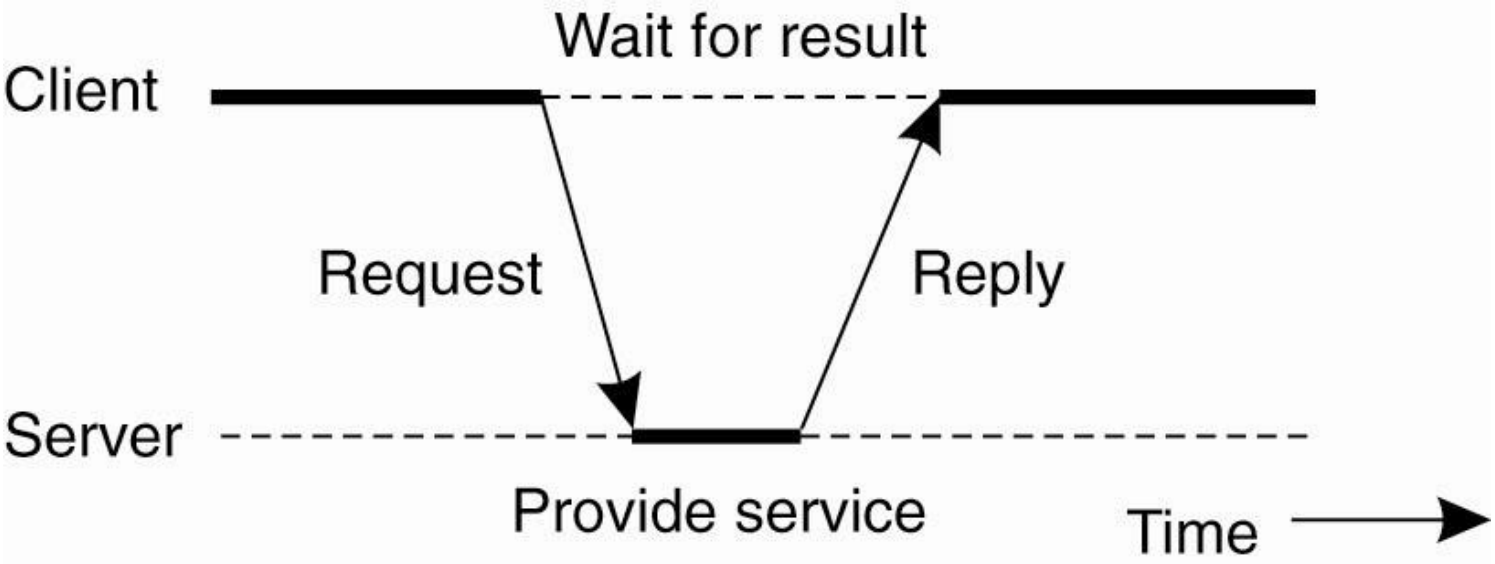


Figure 2-3. General interaction between a client and a server.

System Architectures (4)

Application layering:

- The simplest C-S model has two types of processes:
  - Client that implements (part of) the user interface.
  - Server that implements the rest, i. the processing and data level.
- However, many C-S applications can be organized not in two but three layers
  - Interface, processing, and data (see slide 10).

Application Layering

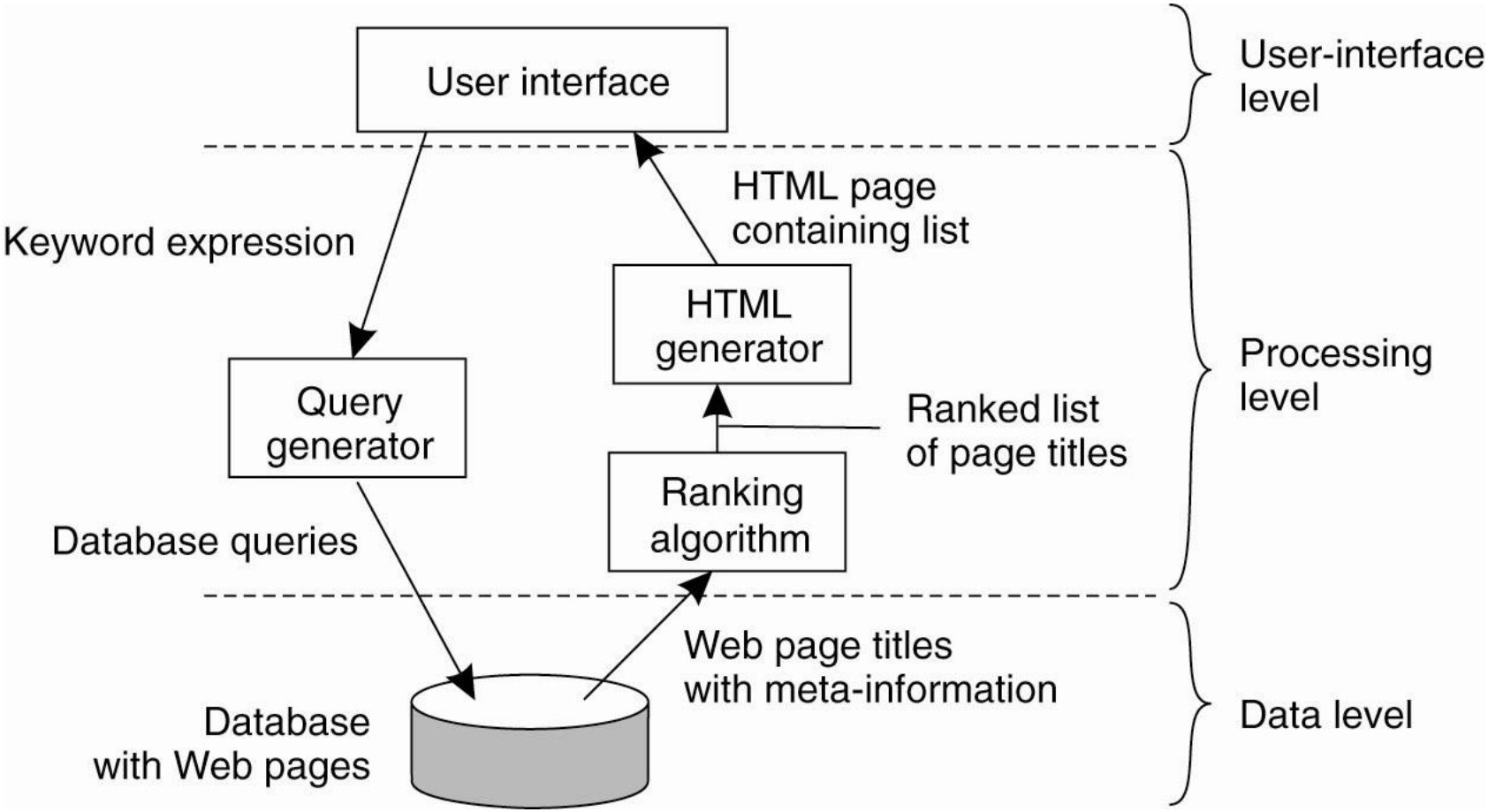


Figure 2-4. The simplified organization of an Internet search engine into three different layers.

Multitiered Architectures (1)

- The distinction into three levels as discussed so far, suggests several possibilities for physically distributing client-server applications across several machines.

Multitiered Architectures (2)

The simplest organization is to have only two types of machines:

- A client machine containing only the programs implementing (part of) the user-interface level.
- A server machine containing the rest,
  - the programs implementing the processing and data level.

Multitiered Architectures (3)

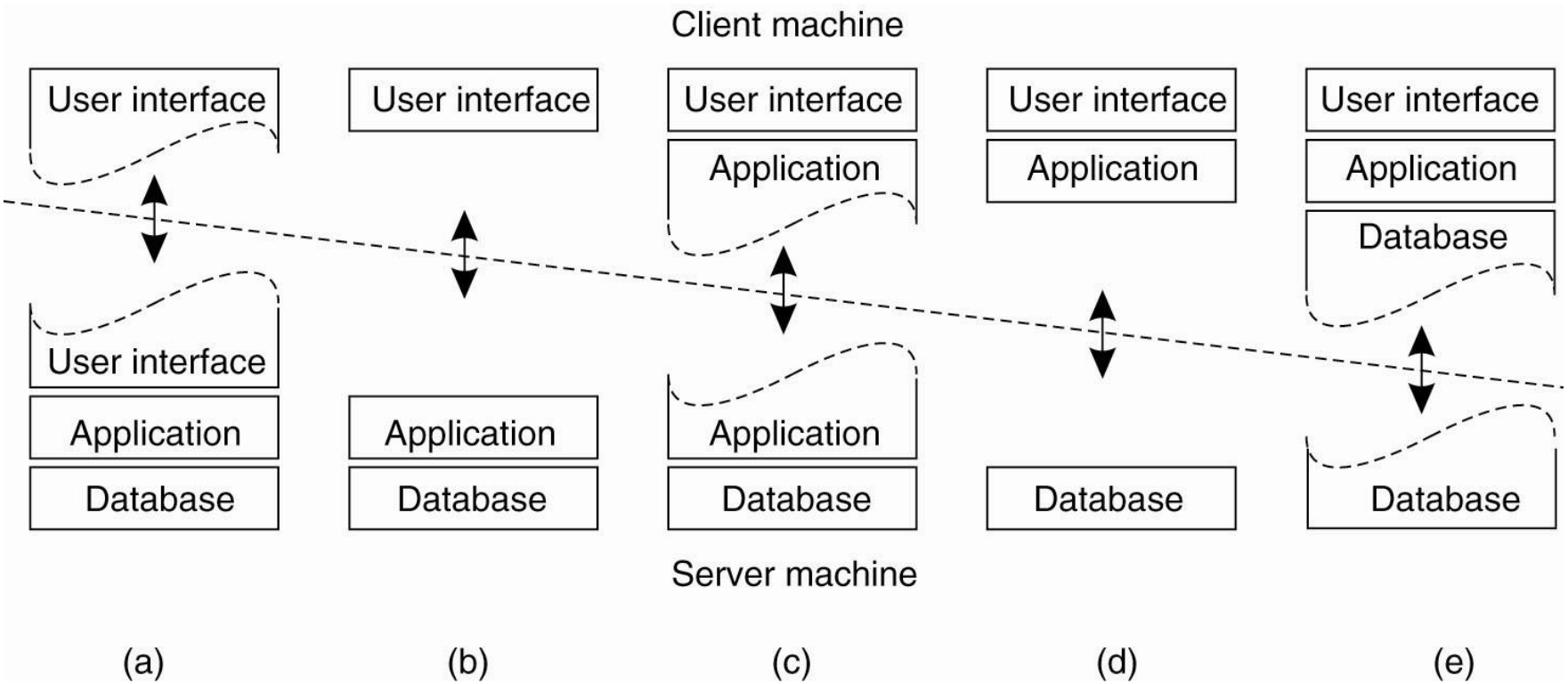


Figure 2-5. Alternative client-server organizations (a)–(e).

Multitiered Architectures (4)

- When thinking in terms of *client* and *server* machines we miss the point that the processing layer can be placed in a separated machine.
- It leads to a (physically) three-tiered C-S architecture.

Multitiered Architectures (5)

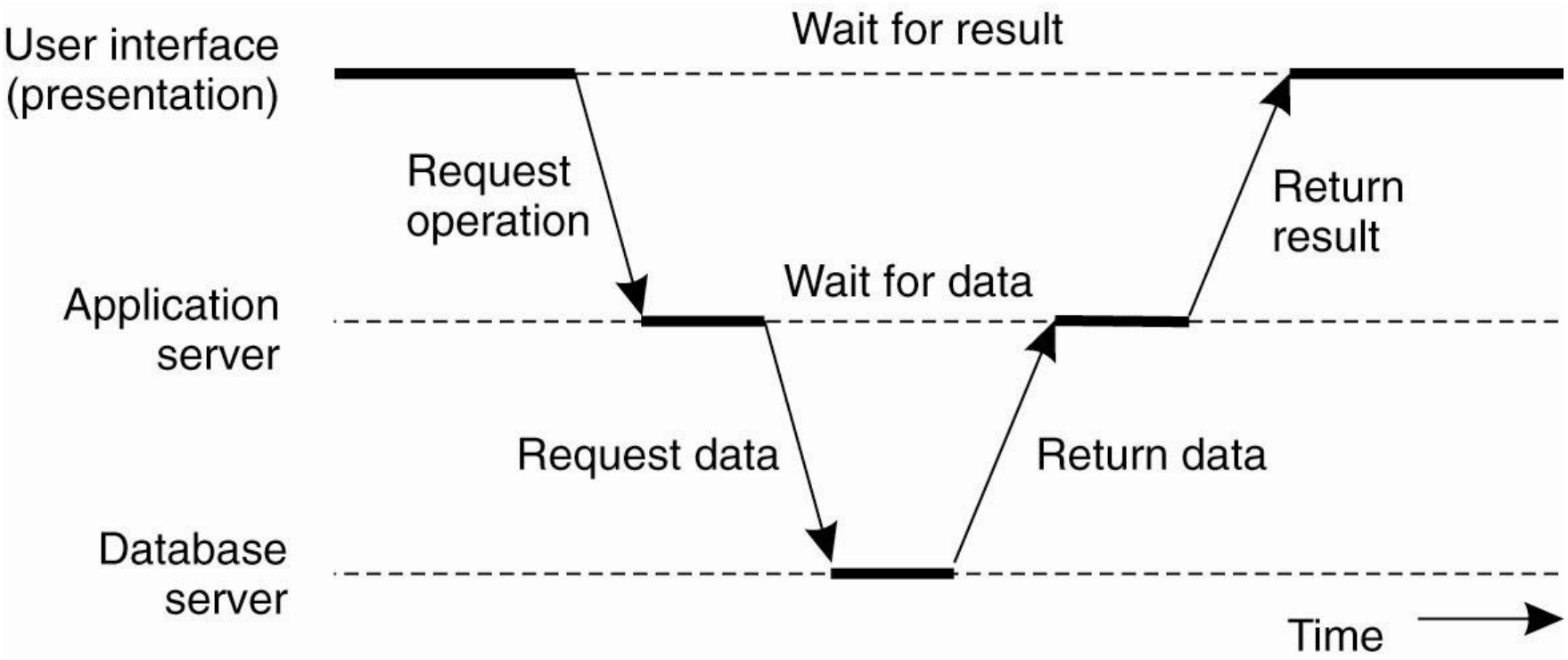


Figure 2-6. An example of a server acting as client.

System Architectures (6)

Decentralized architectures:

- In the last decades we have been seeing a tremendous growth in peer-to-peer systems.
  - **Structured P2P**: nodes are organized following a specific distributed data structure.
  - **Unstructured P2P**: nodes have randomly selected neighbors.
  - **Hybrid P2P**: some nodes are appointed special functions in a well-organized fashion.
- In virtually all cases, we are deadline with overlay networks: data is routed over connections setup between the nodes.

Structured Peer-to-Peer Architectures (1)

Basic idea:

- Organize the nodes in a structured overlay network such as a hypercube, or a logical ring, and make specific nodes responsible for services based only on their ID.
- The system provides an operation *LOOKUP*(key) that will efficiently route the lookup request to the associated node.

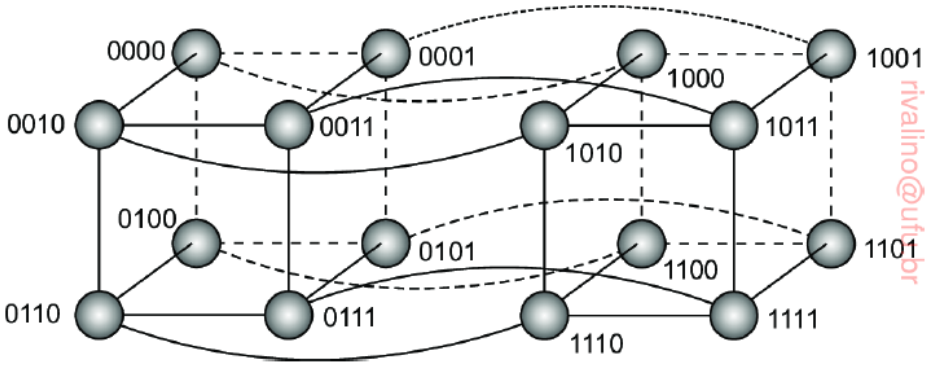


Figure 2.18: A simple peer-to-peer system organized as a four-dimensional hypercube.

Structured Peer-to-Peer Architectures (2)

Note 2.5 (Example: The Chord system)

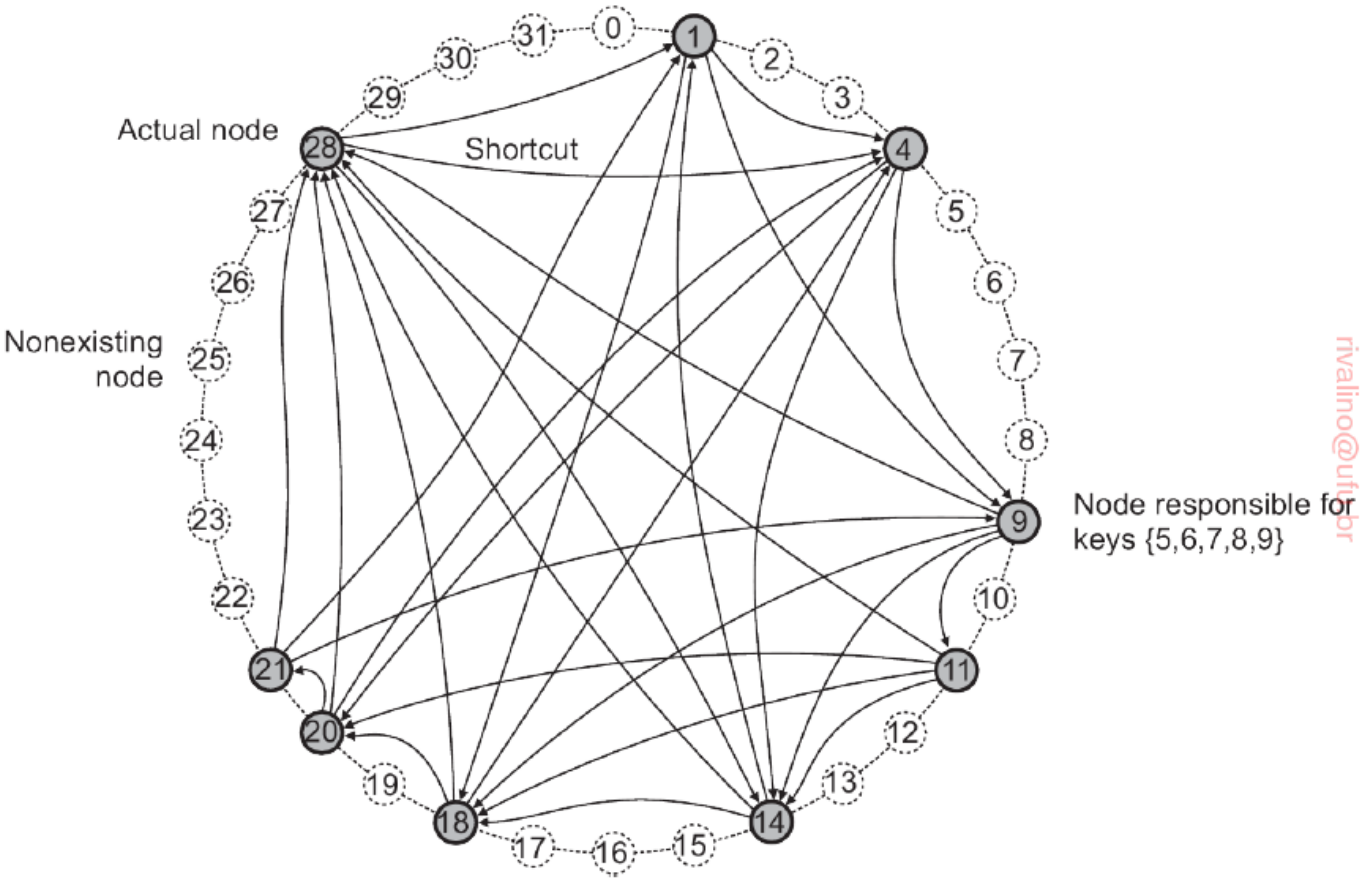


Figure 2.19: The organization of nodes and data items in Chord.

Unstructured Peer-to-Peer Architectures (1)

Basic idea:

- They are organized as a random overlay: two nodes are linked with probability *p*.
- We can no longer look up information deterministically but will have to resort to searching:
  - **Flooding**: node *u* sends a lookup query to all of its neighbors. A neighbor responds, or forwards (floods) the requests.
    - Limited flooding (max number of forwarding)
    - Probabilistic flooding (flood only with a certain probability).
  - **Random walk**: Randomly select a neighbor *v*. If *v* has the answer, it responds, otherwise *v* randomly selects one of its neighbors.

Topology Management of Overlay Networks (1)

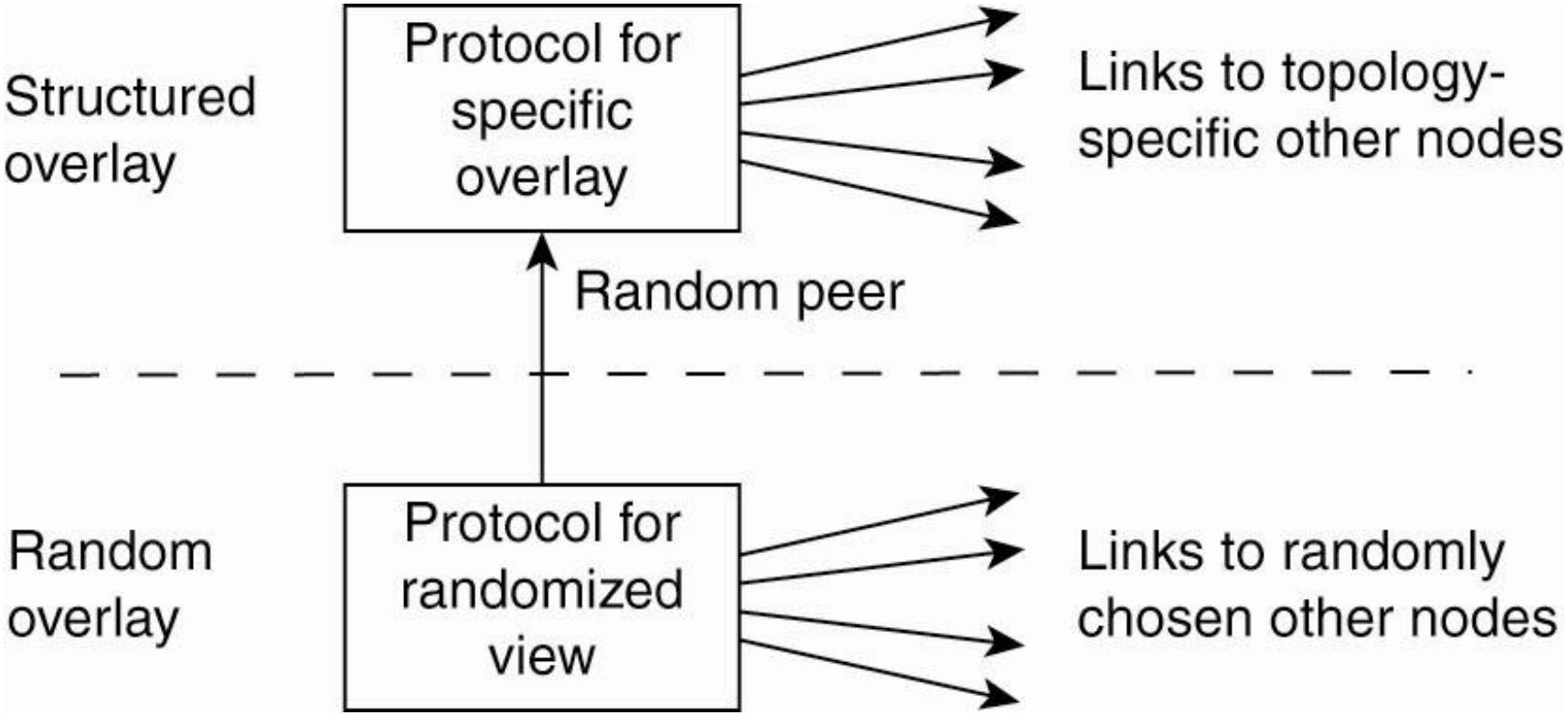


Figure 2-10. A two-layered approach for constructing and maintaining specific overlay topologies using techniques from unstructured peer-to-peer systems.

Topology Management of Overlay Networks (2)

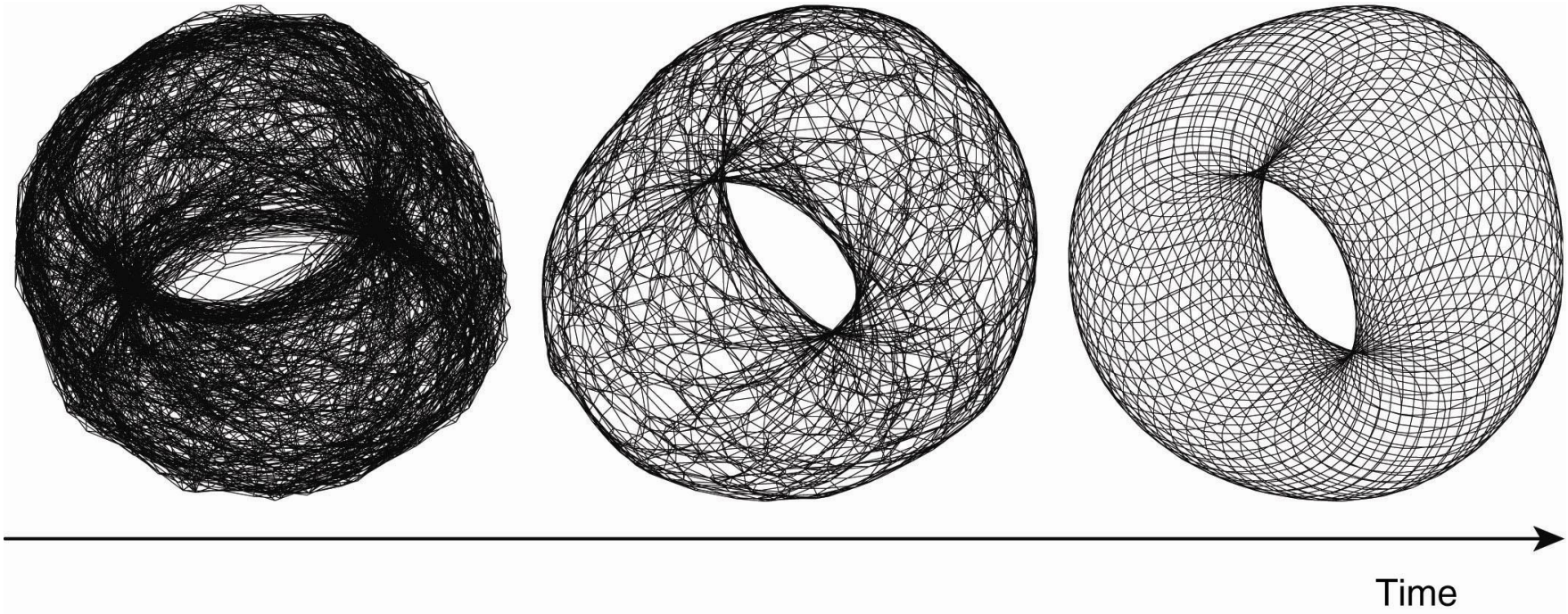


Figure 2-11. Generating a specific overlay network using a two-layered unstructured peer-to-peer system [adapted with permission from Jelasity and Babaoglu (2005)].

Unstructured Peer-to-Peer Architectures (2)

Sometimes it helps to select a few nodes to do specific work: Superpeers

- Peers maintaining an index (for search).
- Peers providing data cache (videos & audio files, web pages, databases, etc.)
- Peers monitoring the state of the network.
- Peers being able to setup connections.
- Peers providing naming services.
- …

Superpeers

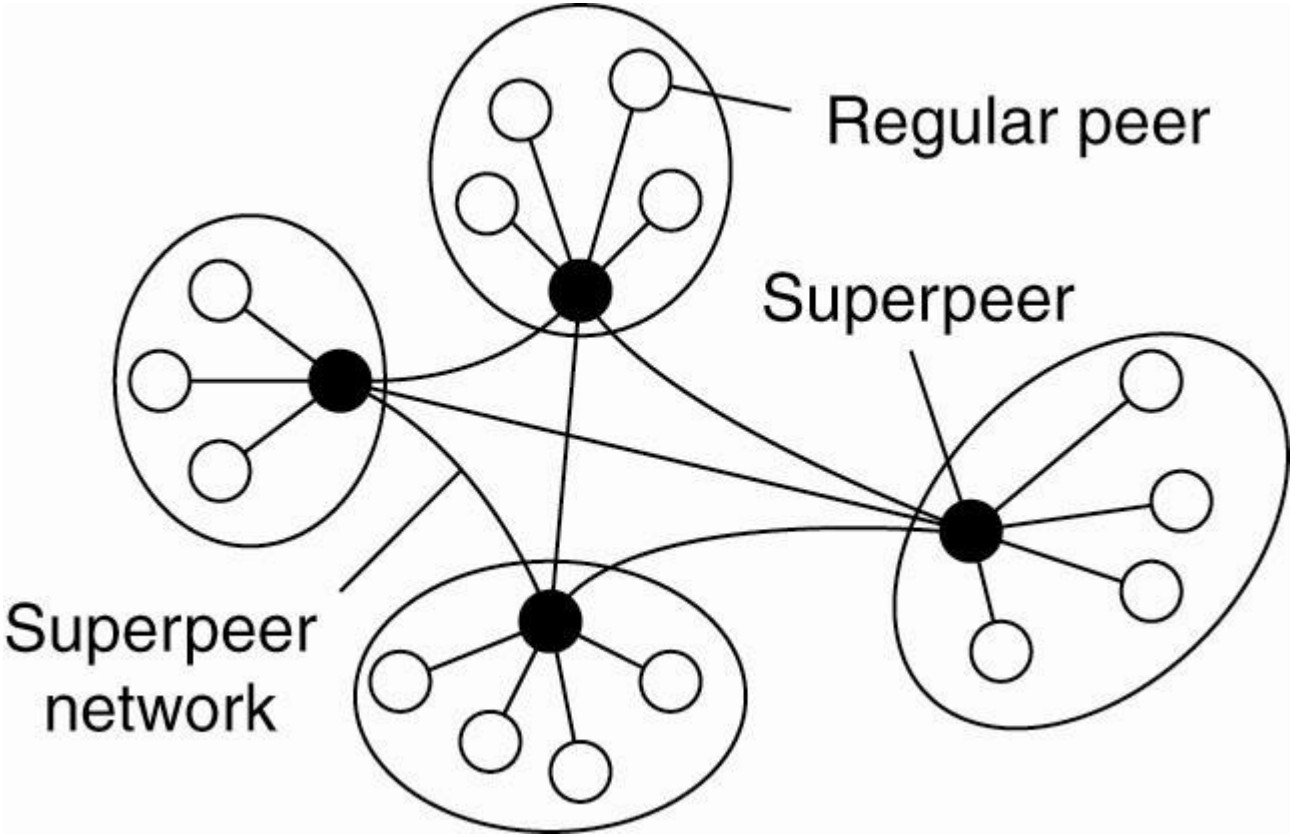


Figure 2-12. A hierarchical organization of nodes into a superpeer network.

Hybrid Peer-to-Peer Architectures (1)

Client-server combined with P2P

- Many distributed systems combine architectural features.
- Some special classes of distributed systems are based on a combination of client-server solutions with decentralized architectures.
- Two examples are:
  - Edge-server systems
  - Collaborative distributed systems

Hybrid Peer-to-Peer Architectures (2)

Edge-server systems

- These systems are deployed on Internet where servers are placed “at the edge” of the network.
- This edge is formed by the boundary between enterprise networks and the actual Internet, e.g., as provided by an Internet Service Provider (ISP).
- The ISP can be considered as residing at the edge of the Internet.
- The set of edge-server systems usually are organized as a peer-to-peer network.
- They are often used for Content Delivery Networks (CDN).

Edge-Server Systems

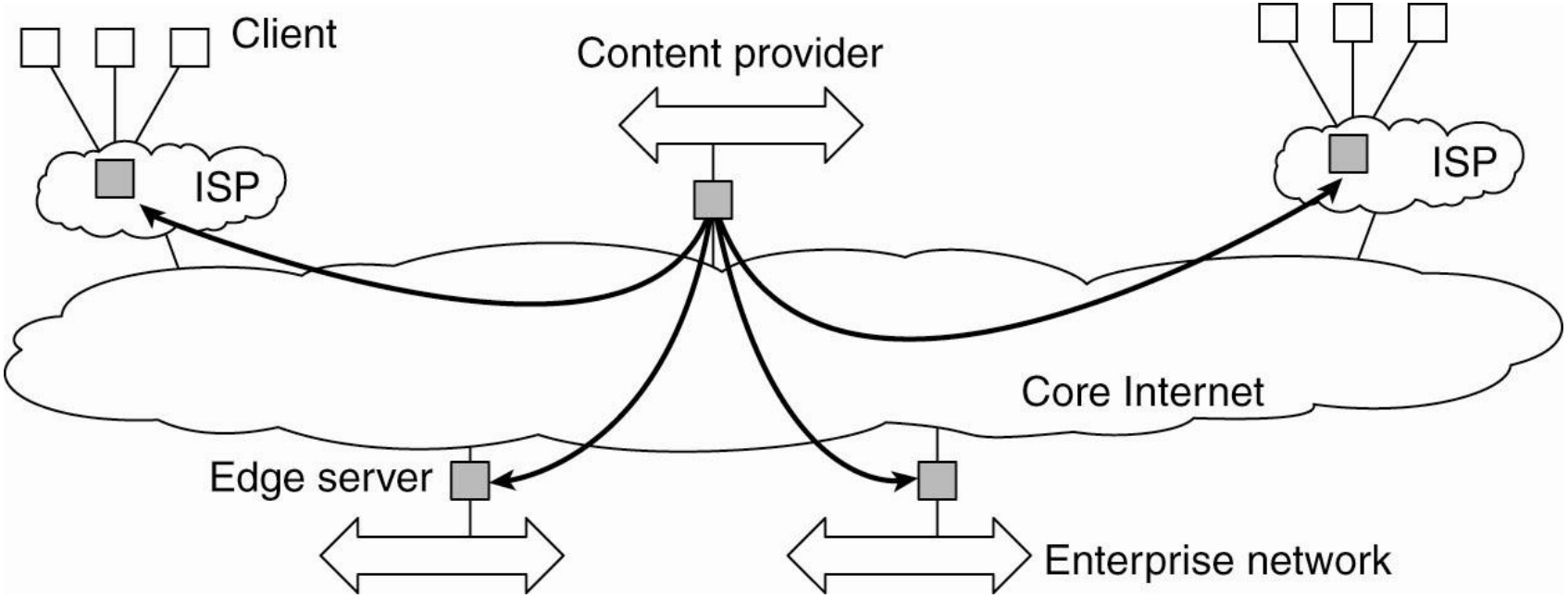


Figure 2-13. Viewing the Internet as consisting of a collection of edge servers.

Hybrid Peer-to-Peer Architectures (3)

Collaborative distributed systems

- The main issue in many collaborative systems is to first get started, for which often a traditional client-server scheme is deployed.
- Once a node has joined the system, it can use a fully decentralized scheme for collaboration.
- A good popular example is BitTorrent.
  - Once a node has identified where to download a file from, it joins a swarm of downloaders who in parallel get file chunks from the source, but also distribute these chunks amongst each other.

Collaborative Distributed Systems

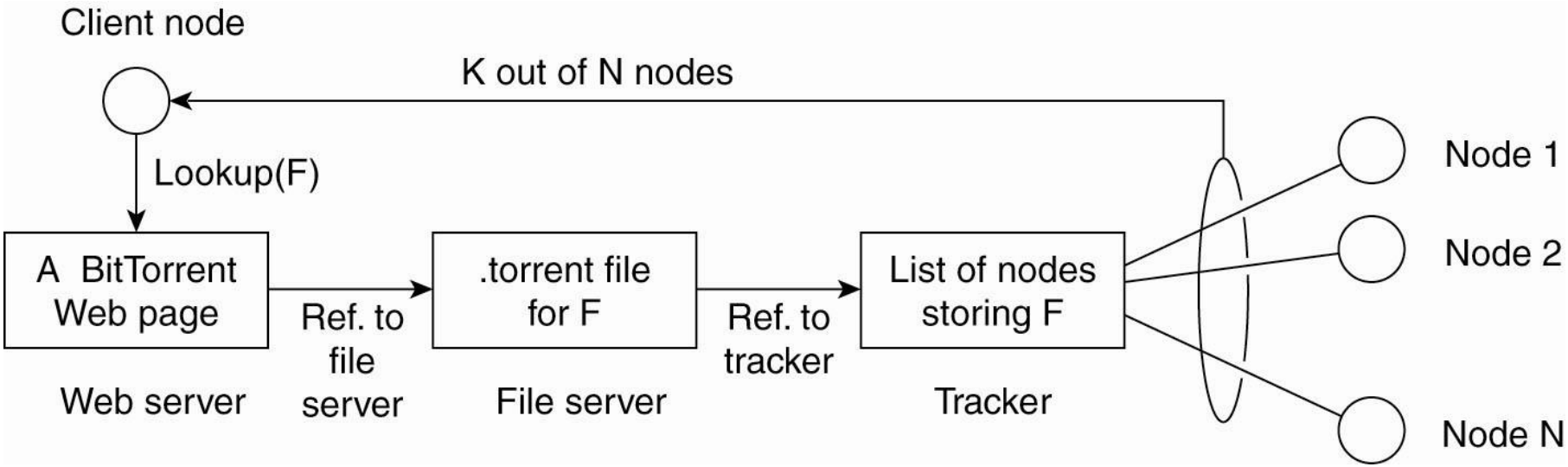


Figure 2-14. The principal working of BitTorrent [adapted with permission from Pouwelse et al. (2004)].

Architecture vs. Middleware

- When analyzing all architectural issues discussed so far, a question that comes is where middleware fits in.
  - It forms a layer between applications and distributed platforms.
  - An important goal is to provide a degree of distribution transparency, i.e., to a certain extent hiding the distribution of data, processing, and control from applications.
- **Problem**: In many cases, applications are developed according to a specific architectural style. The chosen style may not be optimal in all cases.
  - It requires to (dynamically) adapt the behavior of the middleware.
  - Interceptors help to make this integration.

Interceptors

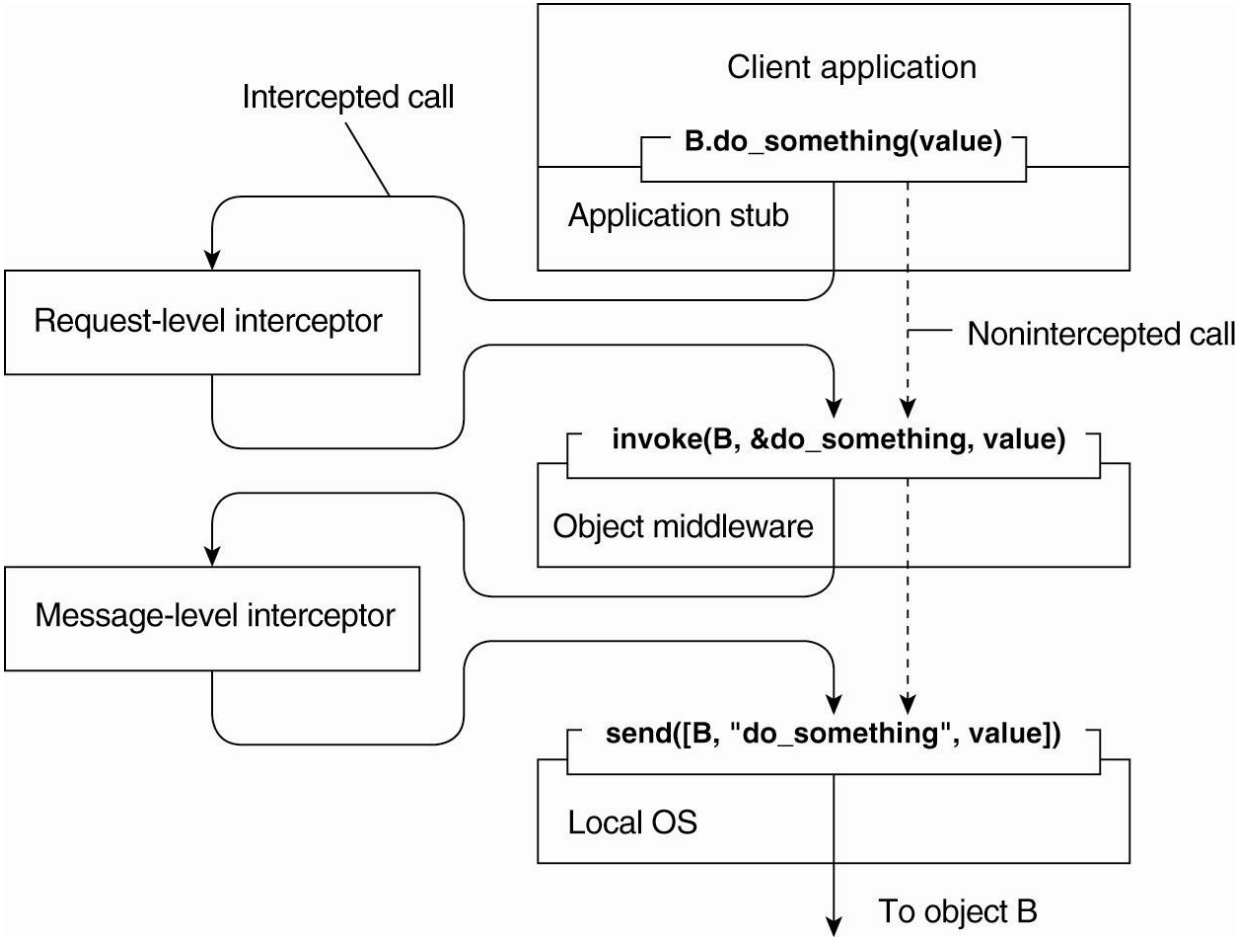


Figure 2-15. Using interceptors to handle remote-object invocations.

More Examples: Network Filesystem (NFS)

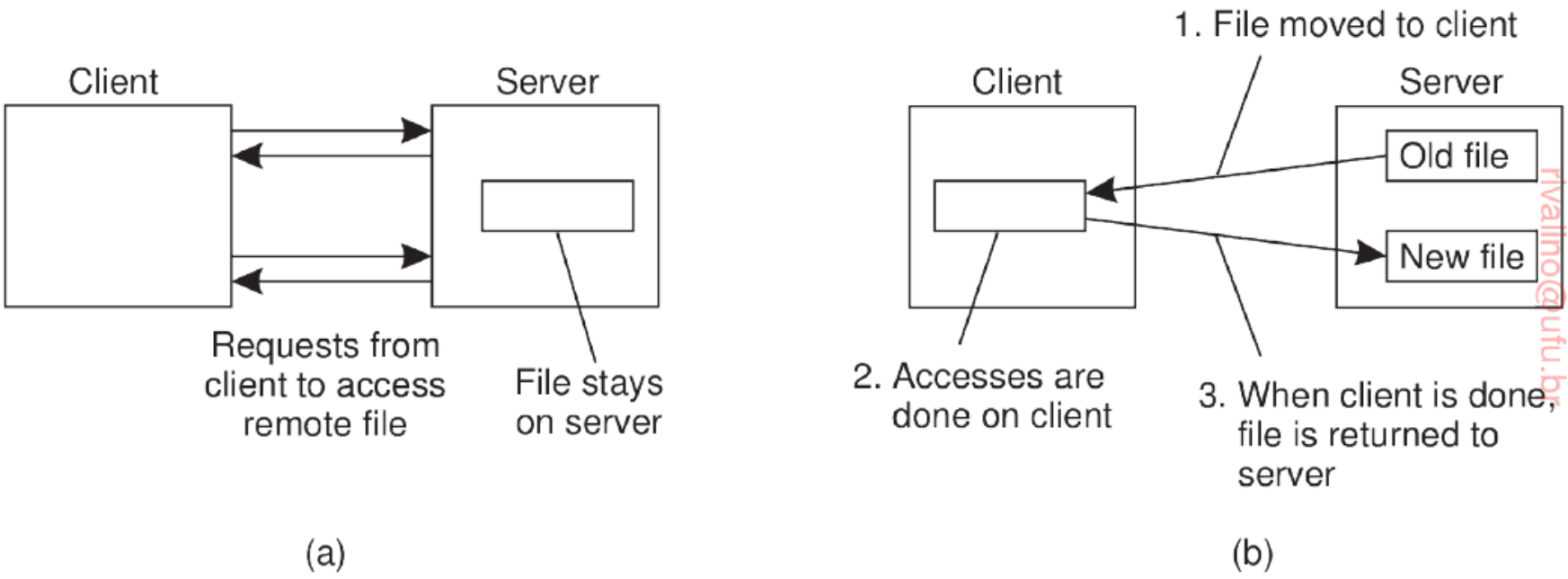


Figure 2.24: (a) The remote access model. (b) The upload/download model.

More Examples: Network Filesystem (NFS)

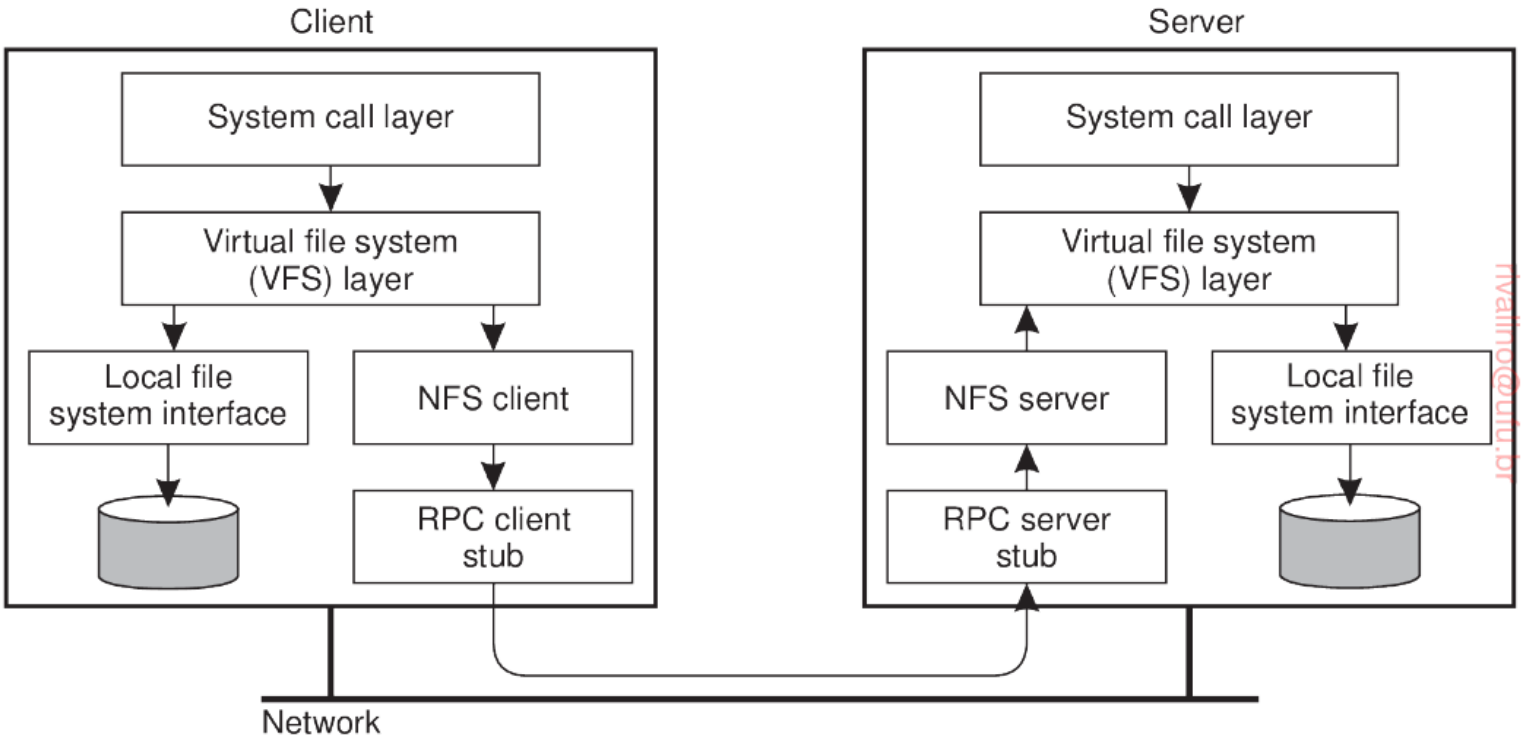


Figure 2.25: The basic NFS architecture for Unix systems.

More Examples: The Web (WWW)

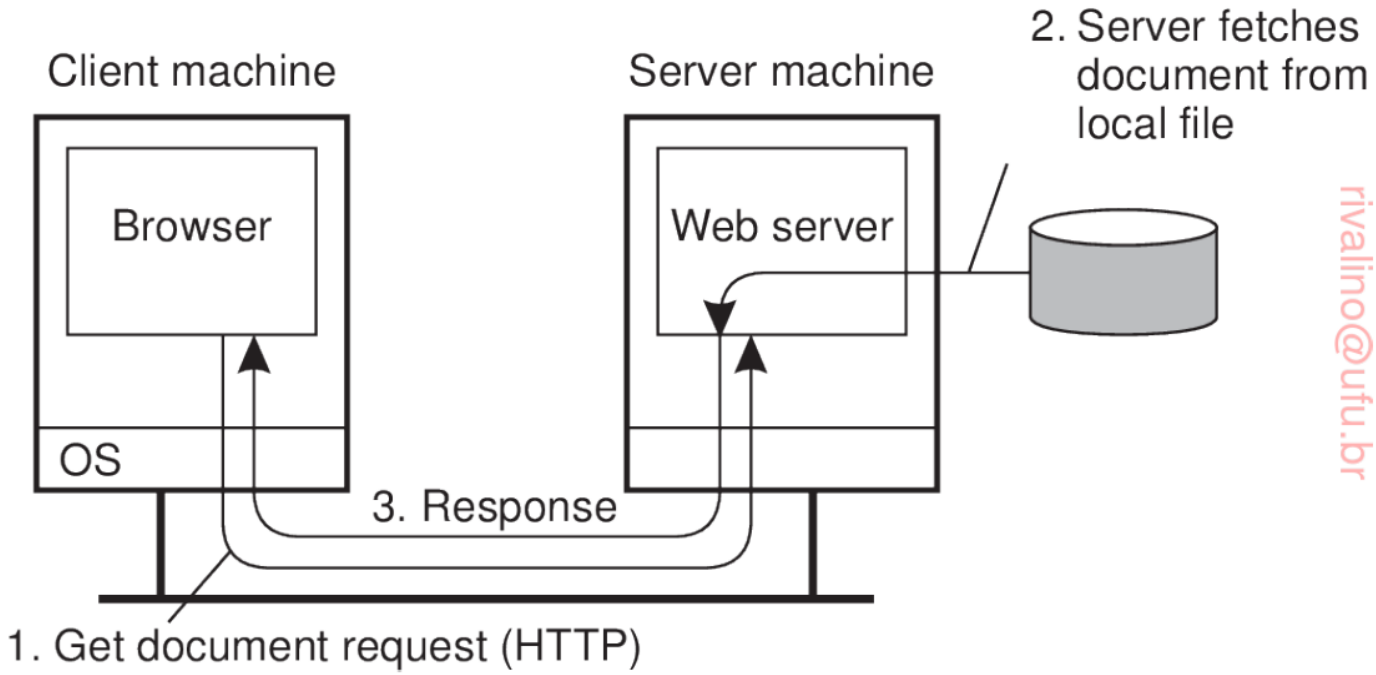


Figure 2.27: The overall organization of a traditional Web site.

More Examples: The Web (WWW)

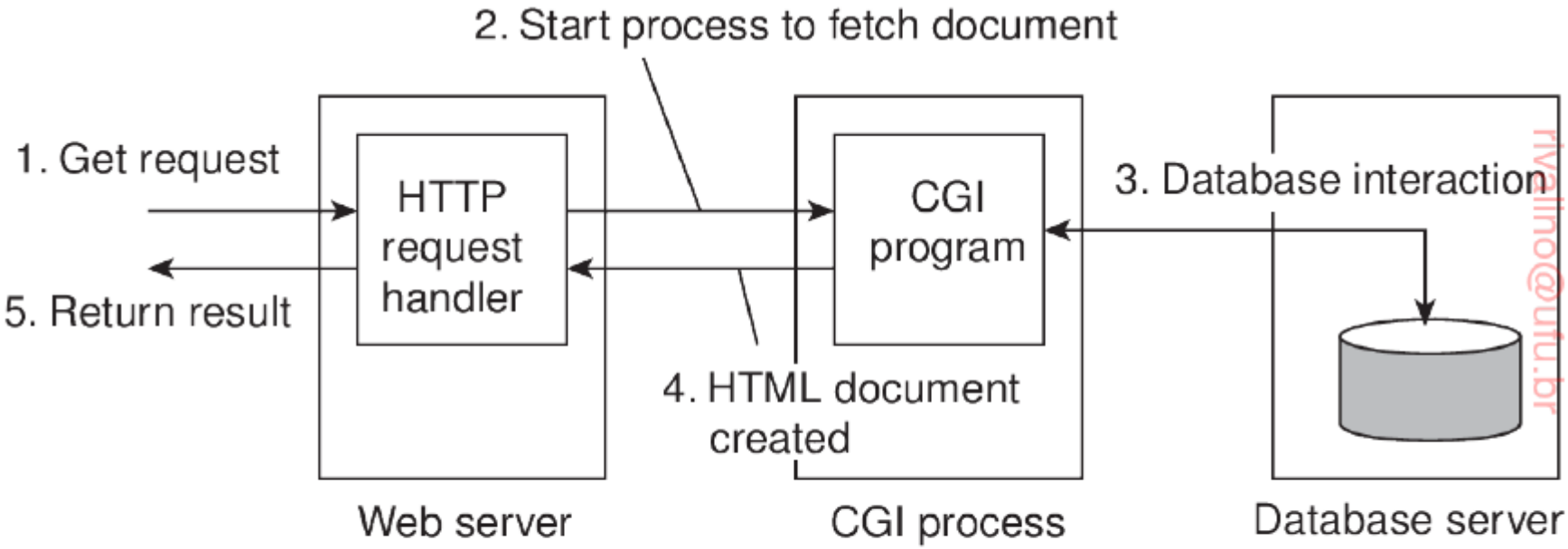


Figure 2.28: The principle of using server-side CGI programs.