DISTRIBUTED SYSTEMS
Principles and Paradigms
Second Edition
ANDREW S. TANENBAUM
MAARTEN VAN STEEN

Chapter 2
ARCHITECTURES

Distributed System Organization

- *Software architectures*: how various software components are organized and how they interact
- *System architecture*: An instance of a software architecture after deciding on the software components, their interaction and their placement.

Architectural Styles (1)

Important styles of architecture for distributed systems

- Layered architectures
- Object-based architectures
- Data-centered architectures
- Event-based architectures

Architectural Styles (2)

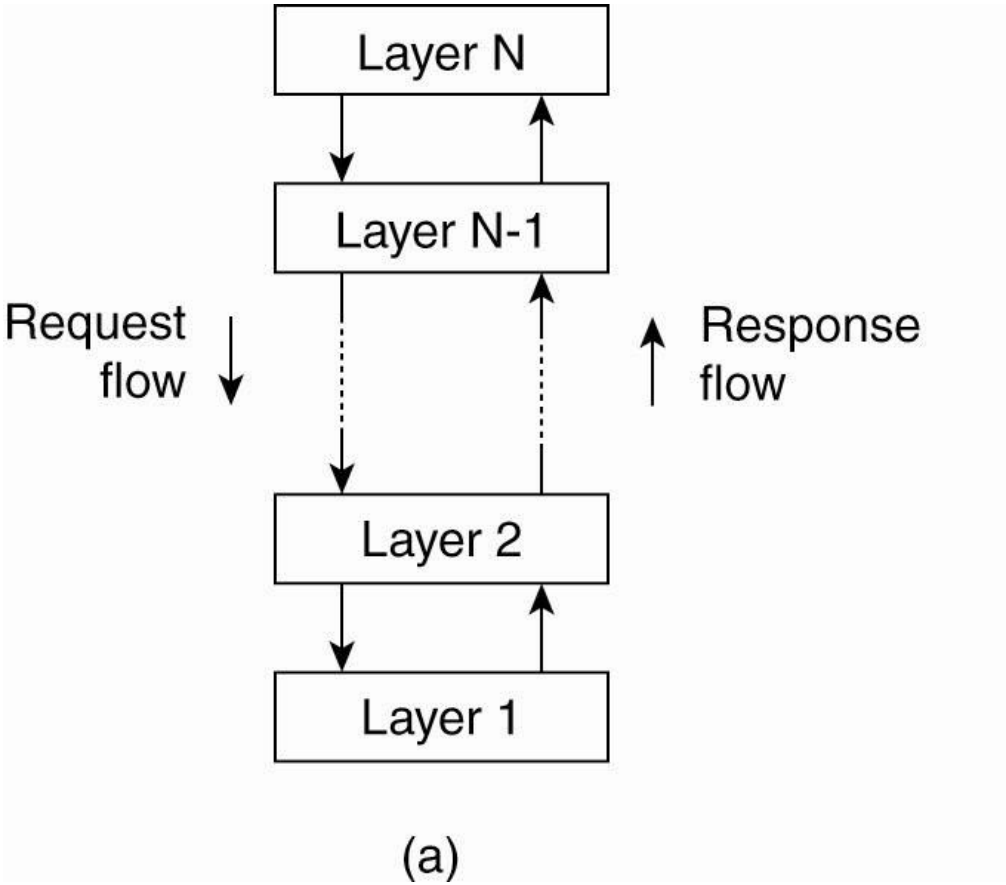

Figure 2-1. The (a) layered architectural style and …

Architectural Styles (3)

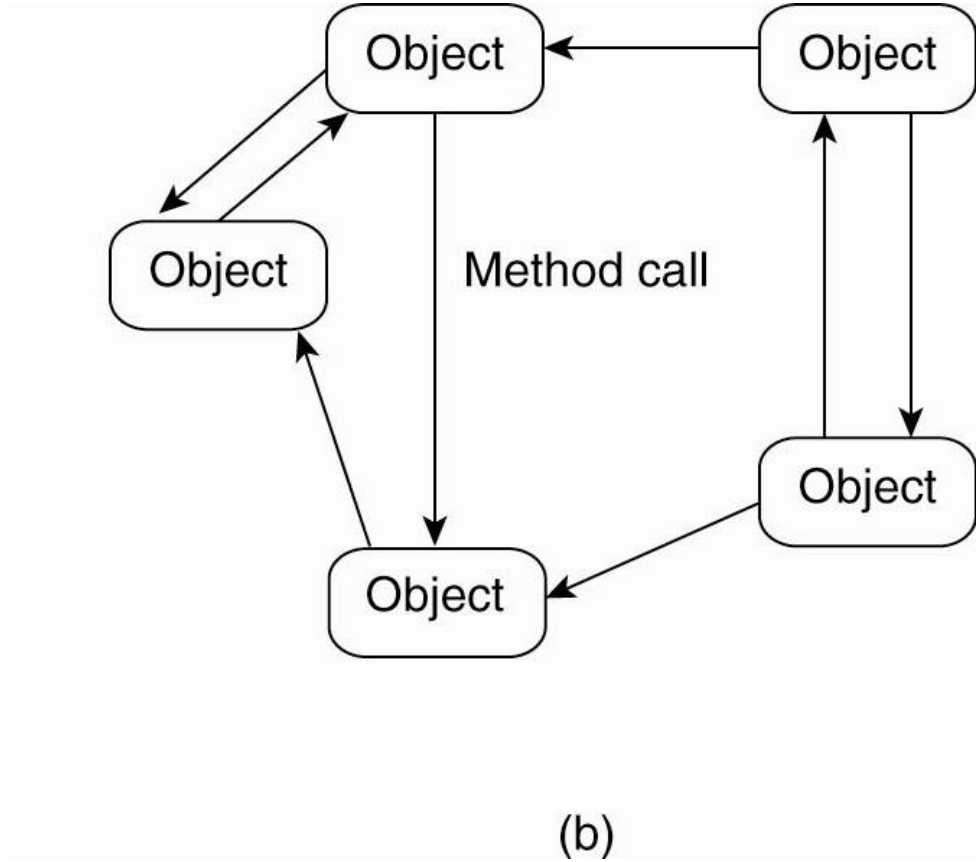

Figure 2-1. (b) The object-based architectural style.

Architectural Styles (4)

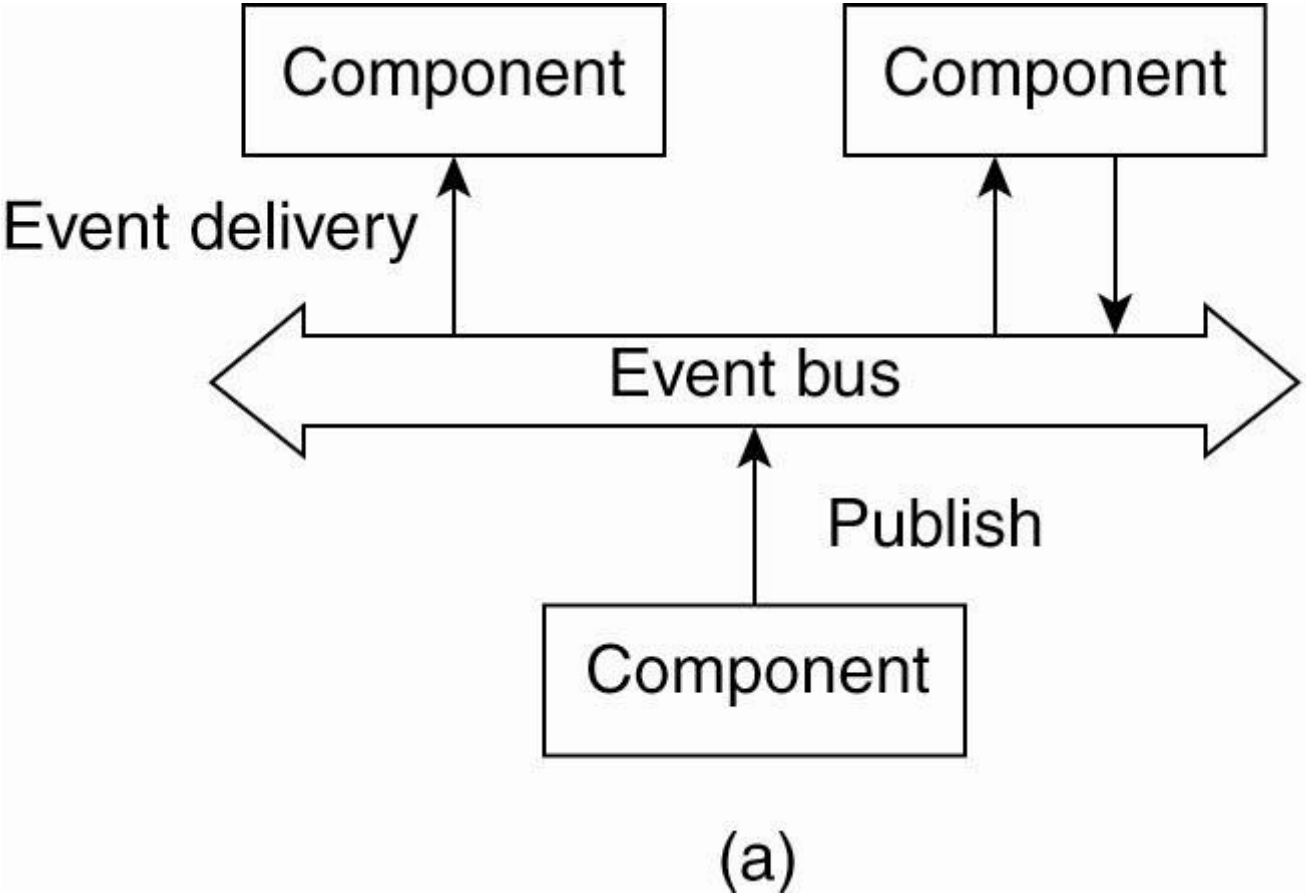

Figure 2-2. (a) The event-based architectural style and …

Architectural Styles (5)

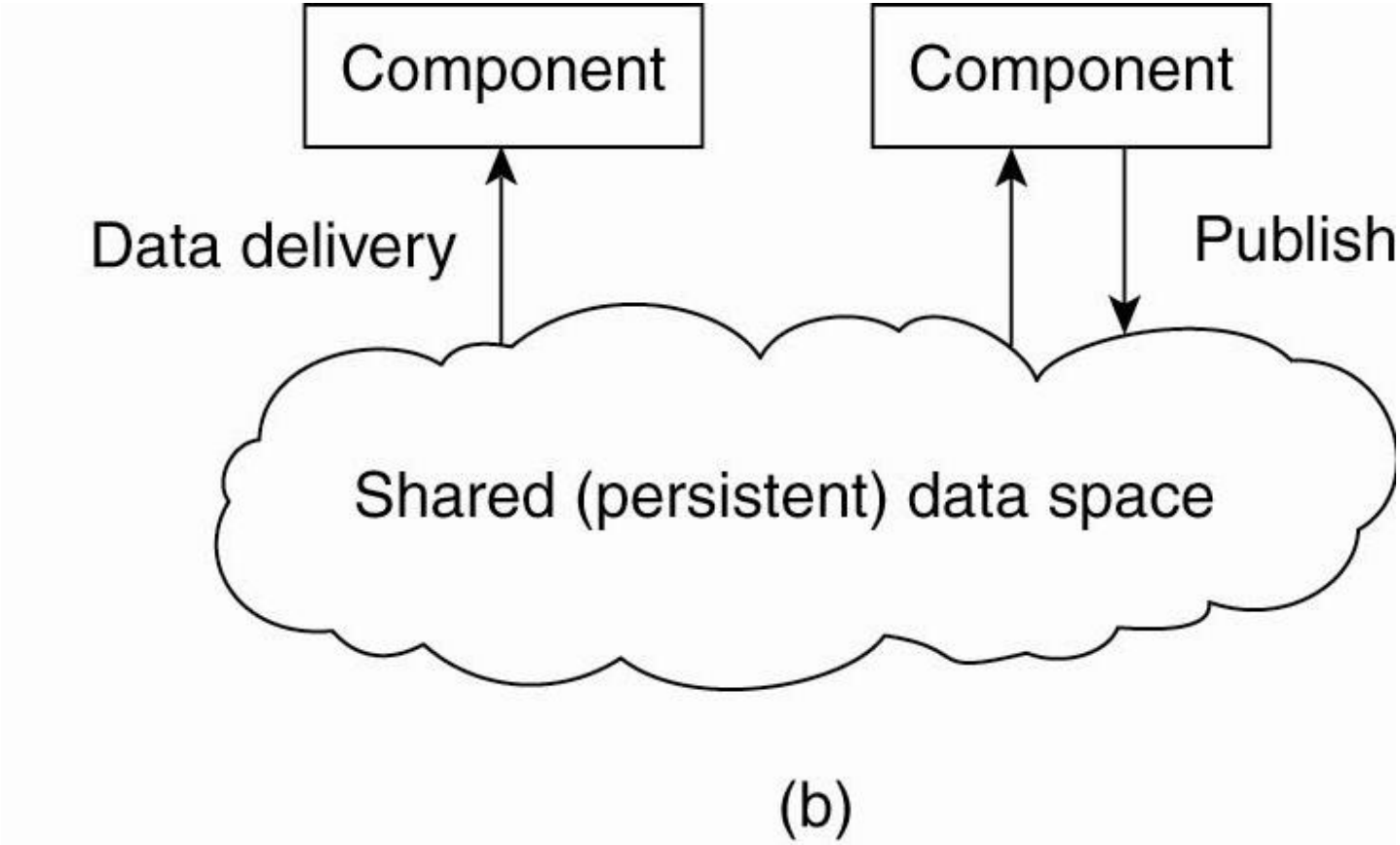

Figure 2-2. (b) The shared data-space architectural style.

System Architecture

- Centralized architectures
  - Application layering
  - Multitiered architectures
- Decentralized architectures
  - Structured peer-to-peer
  - Unstructured peer-to-peer
- Hybrid architectures
  - Edge-server systems
  - Collaborative distributed systems

Centralized Architectures

A centralized architecture consists of a server and multiple clients.

- A *server* is a process implementing a sepcific service such as 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 a reply.
- Communication between server/clients:
  - *Connectionless protocol*: Works only for idempotent operations. UDP is an example. When an operation can be repeated multiple times without harm, it is said to be *idempotent*.
  - *Reliable connection-oriented protocol*: TCP is the most common example.

Centralized Architectures

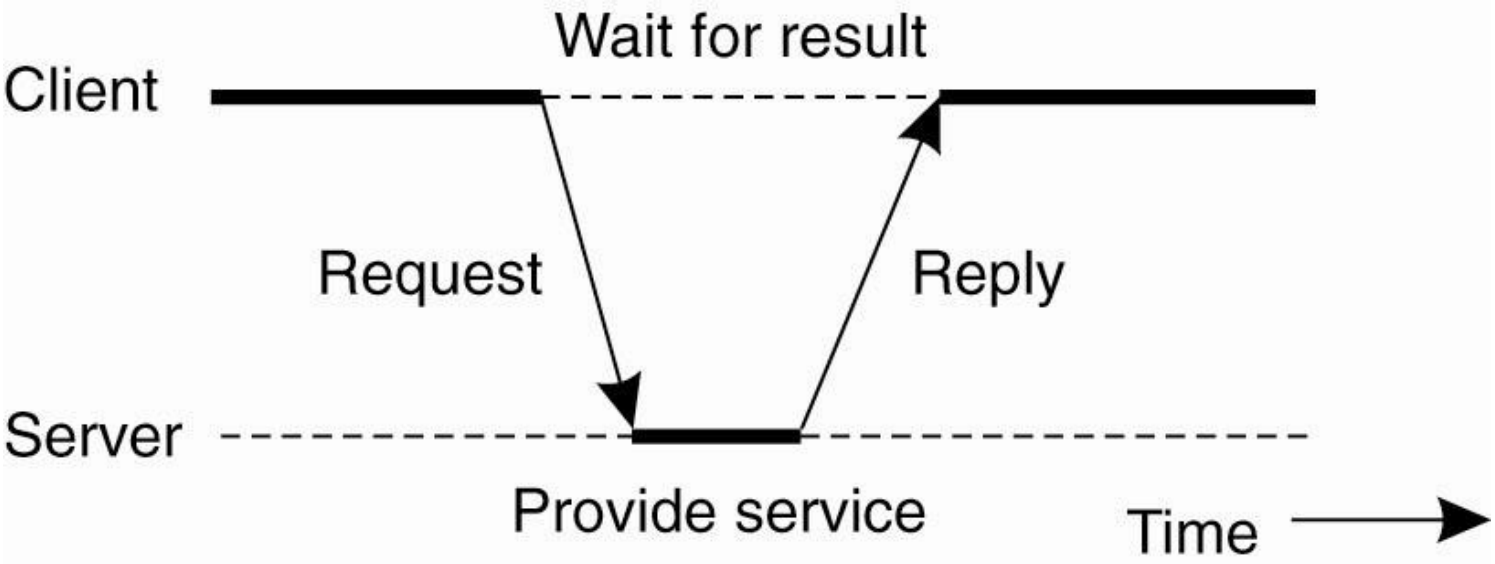

Figure 2-3. General interaction between a client and a server.

Application Layering (1)

Layers of architectural style

- The user-interface level: *Client*
- The processing level: *Client* or *Server*
  - *Internet search engine*: User-interface, Query-ranking-results generator, database of web pages and associated indices
  - *Decision support system for a stock broker*: user-interface, analysis program, backend for accessing database for financial data
- The data level: *Server.* This level is persistent, which implies data is stored outside of an application. Typically a file system or database but could also be a NoSQL store (like Hadoop) or an object store.

Application Layering (2)

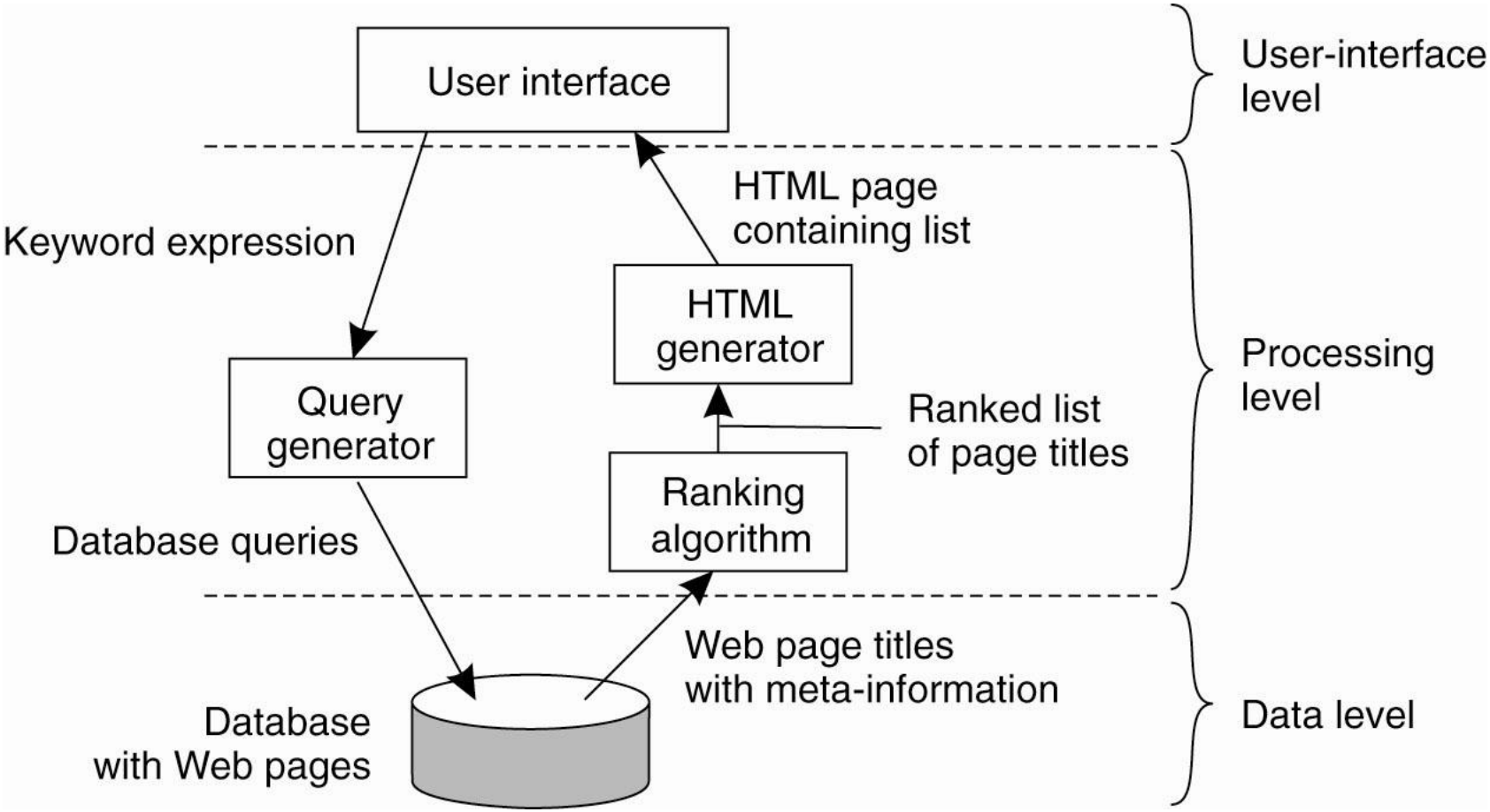

Figure 2-4. The simplified organization of an Internet search engine into three different layers.

Multitiered Architectures (1)

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
- Clients can be *fat* or *thin* clients. Servers can act as clients as well, leading to a three-tier design.

Multitiered Architectures (2)

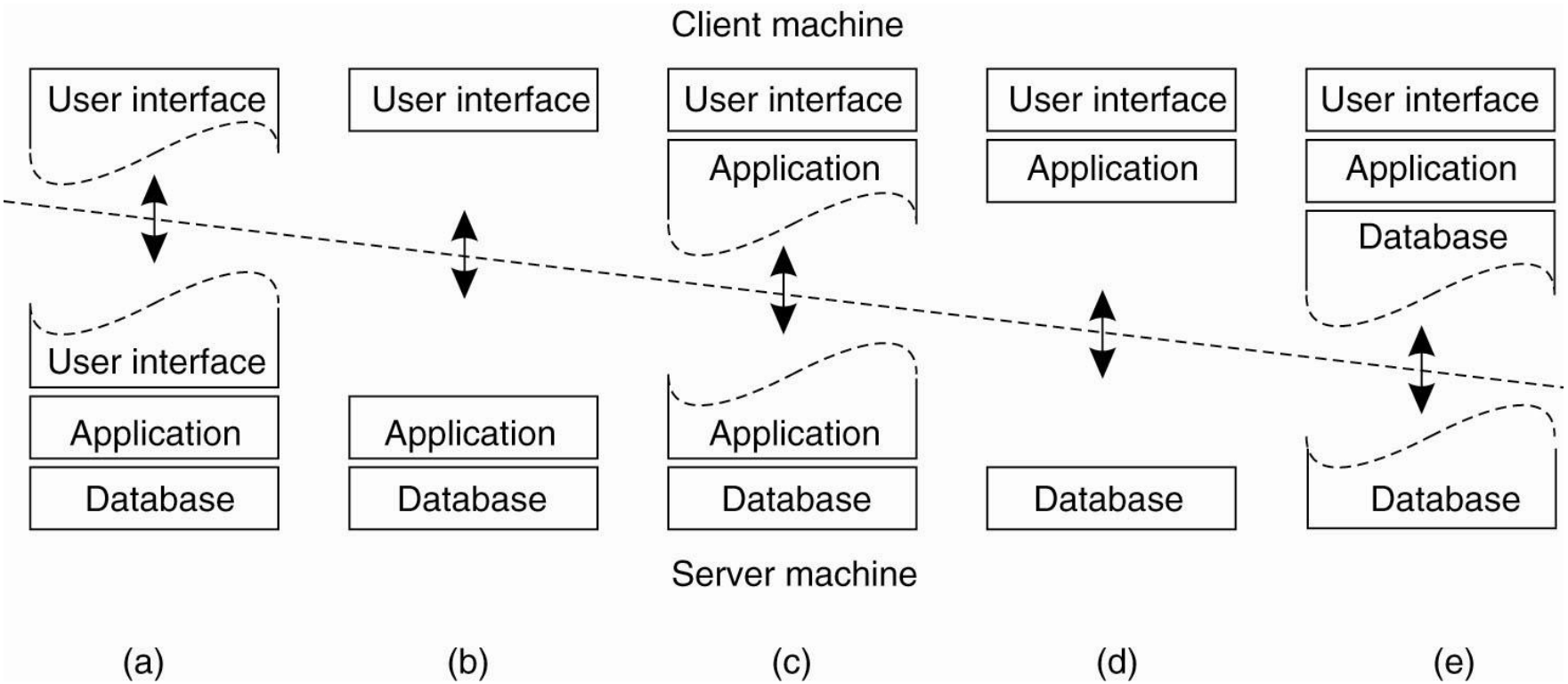

Figure 2-5. Alternative client-server organizations (a)–(e).

Multitiered Architectures (3)

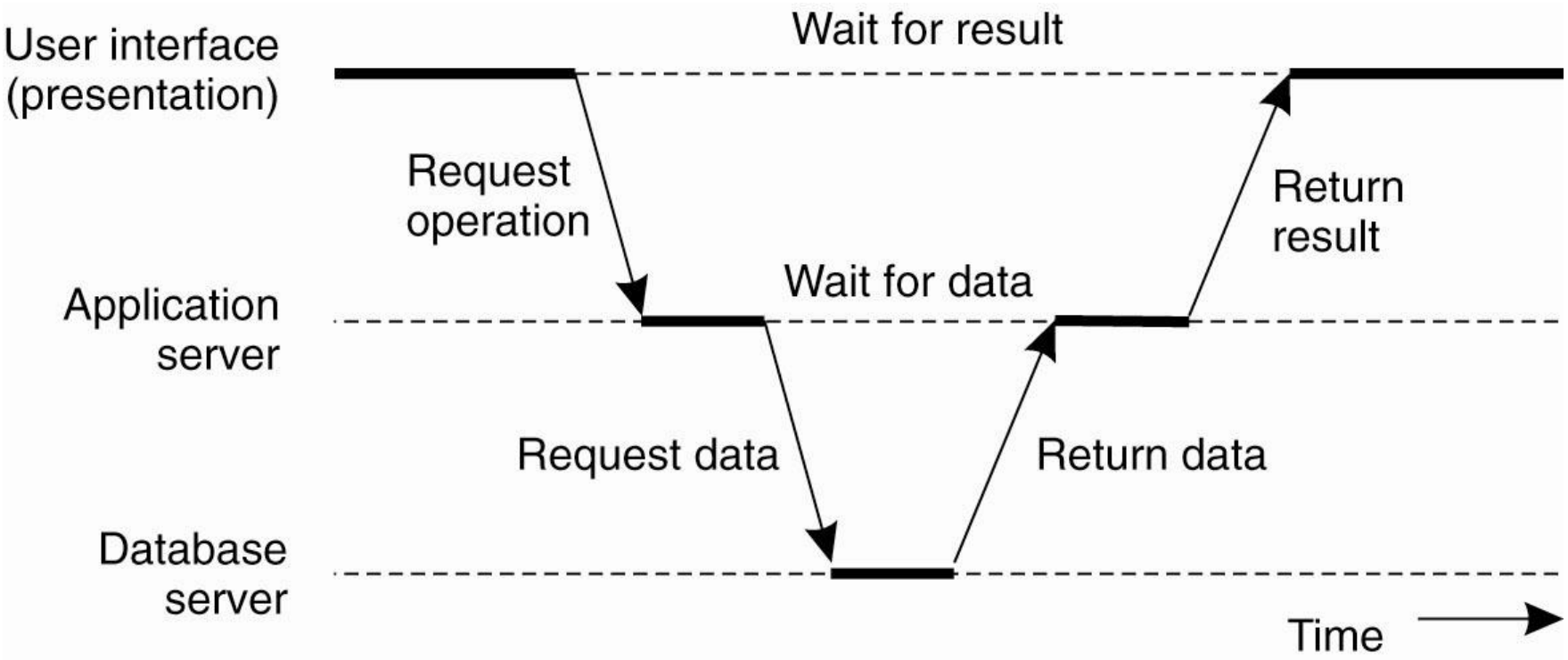

Figure 2-6. An example of a server acting as client.

Decentralized Architecture

- Vertical versus horizontal distribution. Horizontal distribution leads to peer-to-peer architectures.
- Each process acts as a client and server at the same time. These are referred to as *servents*. Then the nodes are arranged in an *overlay network* based on the connections between them. These are of two types:
  - Structured peer-to-peer
    - Distributed Hash Table
    - Content-Addressable Network
  - Unstructured peer-to-peer

Structured Peer-to-Peer Architectures (1)

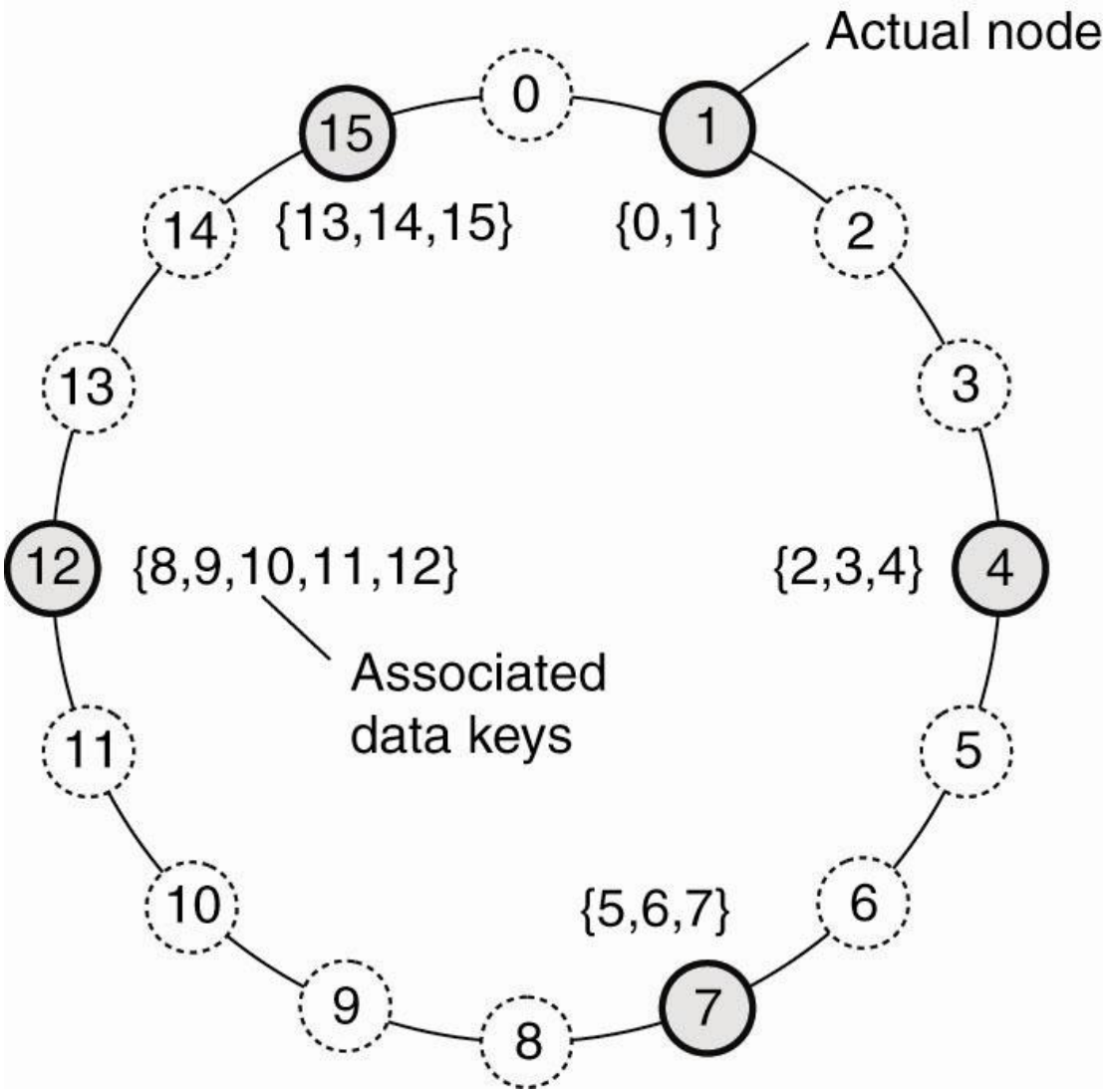

Figure 2-7. The mapping of data items onto nodes in Chord.

Structured Peer-to-Peer Architectures (2)

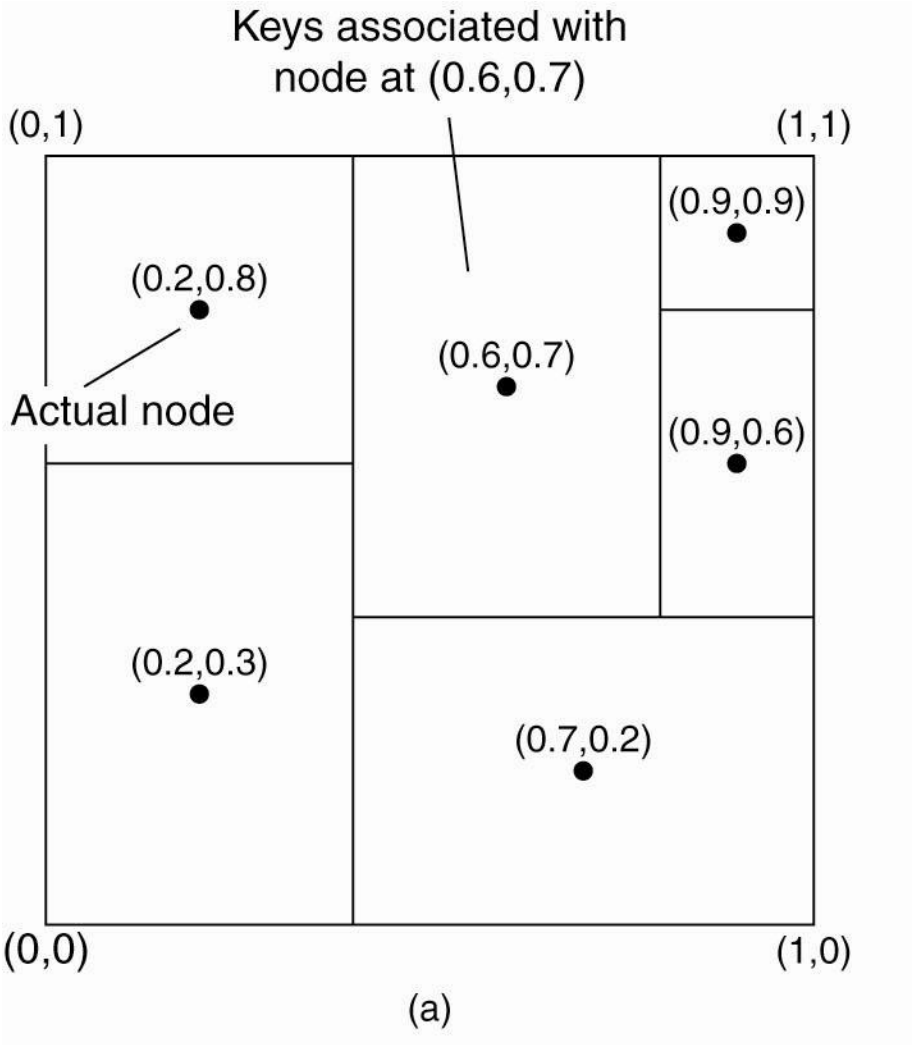

Figure 2-8. (a) The mapping of data items onto nodes in CAN.

Structured Peer-to-Peer Architectures (3)

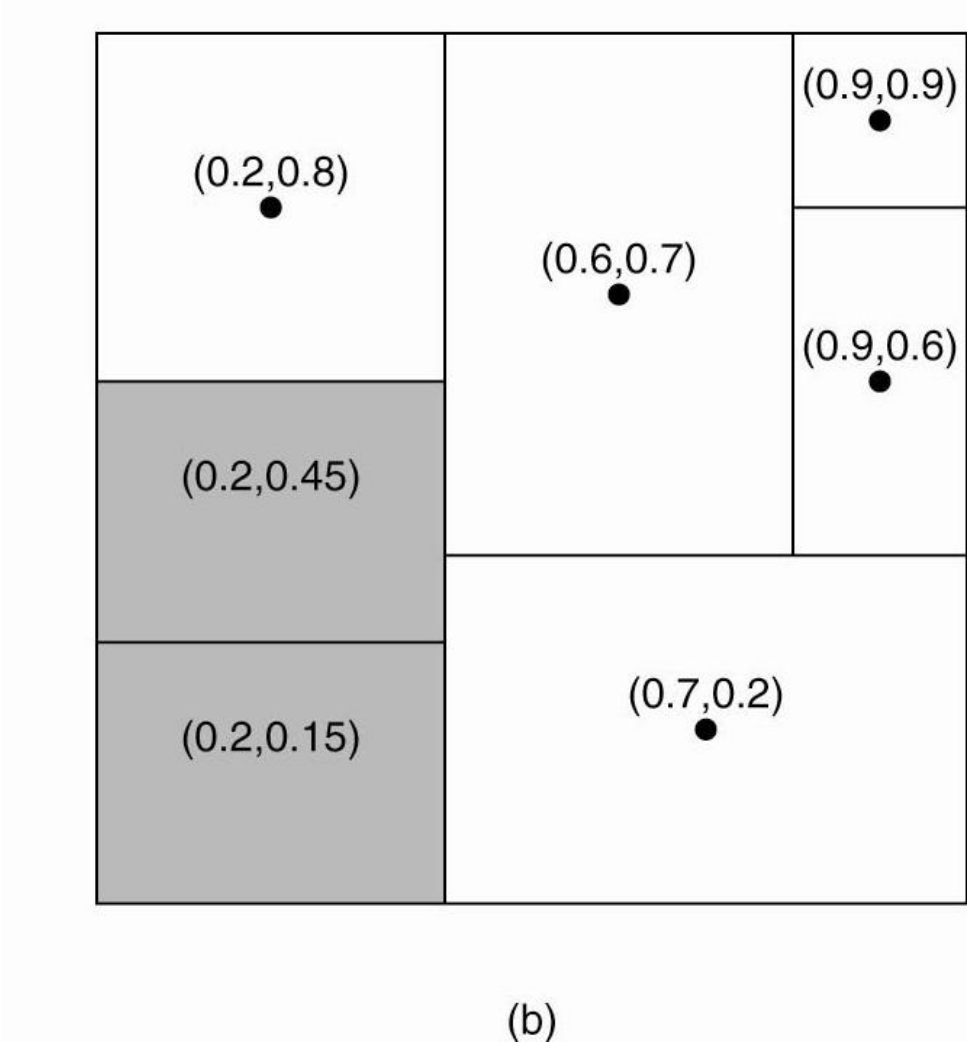

Figure 2-8. (b) Splitting a region when a node joins.

Unstructured Peer-to-Peer Architectures (1)

The overlay network resembles a *random graph*. Each node maintains a list of *c* neighbors, where, ideally, each of theses neighbors represents a randomly chosen *live* node from the current set of nodes. The list of neighbors is referred to as a *partial view*.

How to maintain a partial view?

- Nodes are in *push* or *pull* mode. Using only one mode leads to isolated sub-networks so most nodes will do both (*exchange* mode)
- To add to the group, simply contact any node. To leave, simply leave.

Unstructured Peer-to-Peer Architectures (2)

Actions by active thread (periodically repeated):

```
select a peer P from the current partial view;
if PUSH_MODE {
    mybuffer = [(MyAddress, 0)];
    permute partial view;
    move H oldest entries to the end;
    append first c/2 entries to mybuffer;
    send mybuffer to P;
} else {
    send trigger to P;
}
if PULL_MODE {
    receive P's buffer;
}
construct a new partial view from the current one and P's buffer;
increment the age of every entry in the new partial view;
```

(a)

Figure 2-9. (a) The steps taken by the active thread.

Unstructured Peer-to-Peer Architectures (3)

Actions by passive thread:

```
receive buffer from any process Q;
if PULL_MODE {
    mybuffer = [(MyAddress, 0)];
    permute partial view;
    move H oldest entries to the end;
    append first c/2 entries to mybuffer;
    send mybuffer to P;
}
construct a new partial view from the current one and P's buffer;
increment the age of every entry in the new partial view;
```

(b)

Figure 2-9. (b) The steps take by the passive thread

Topology Management of Overlay Networks (1)

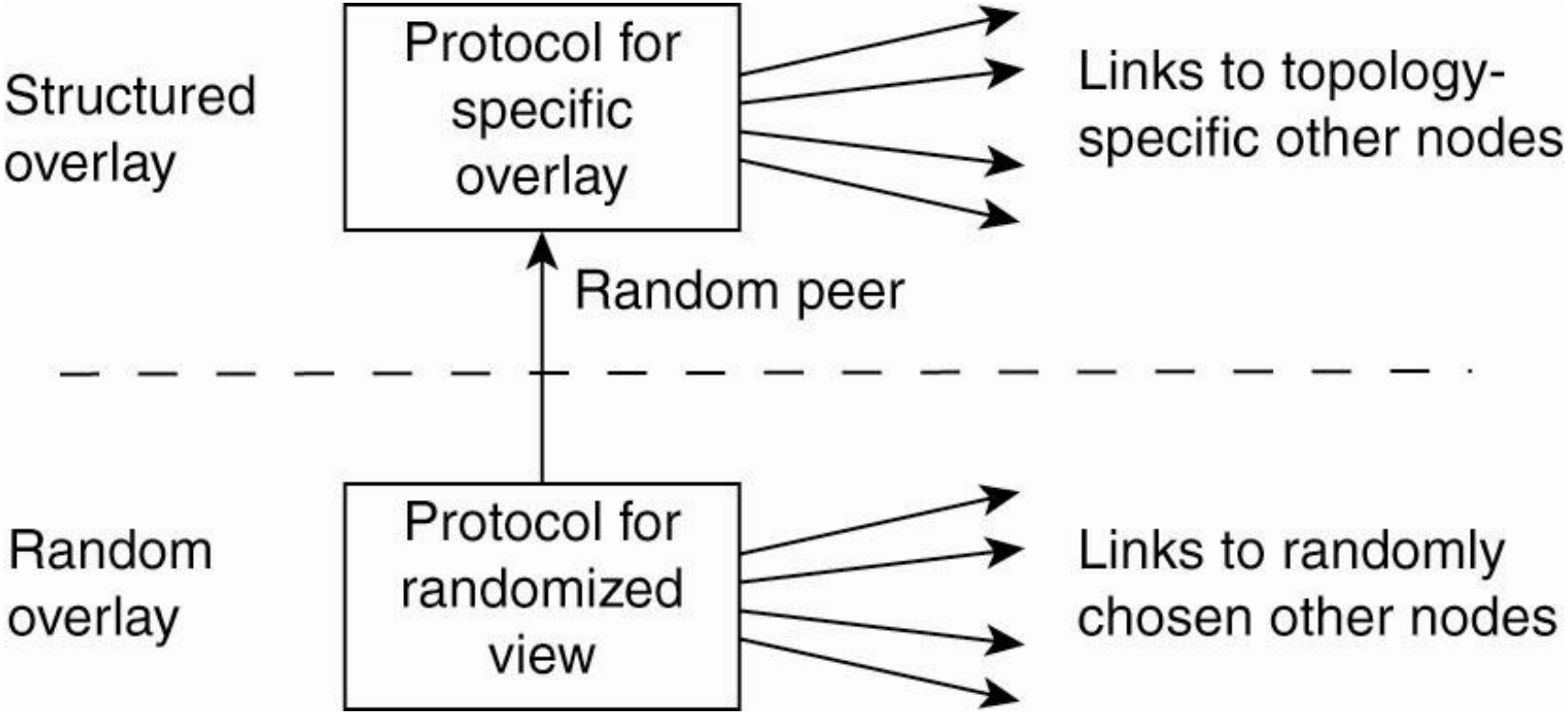

Figure 2-10. A two-layered approach for constructing and maintaining specific overlay topologies using techniques from unstructured peer-to-peer systems.

Topology Management of Overlay Networks (2)

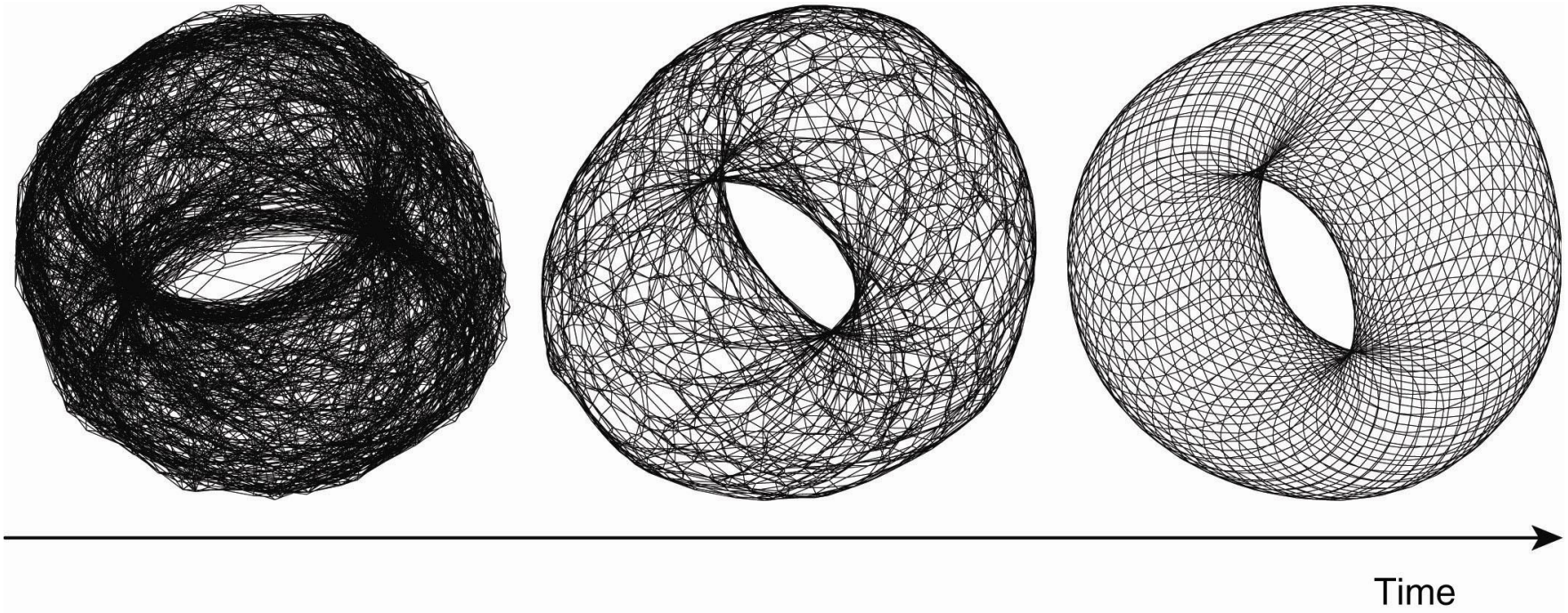

Figure 2-11. Generating a specific overlay network using a two-layered unstructured peer-to-peer system [adapted with permission from Jelasity and Babaoglu (2005)].

Superpeers

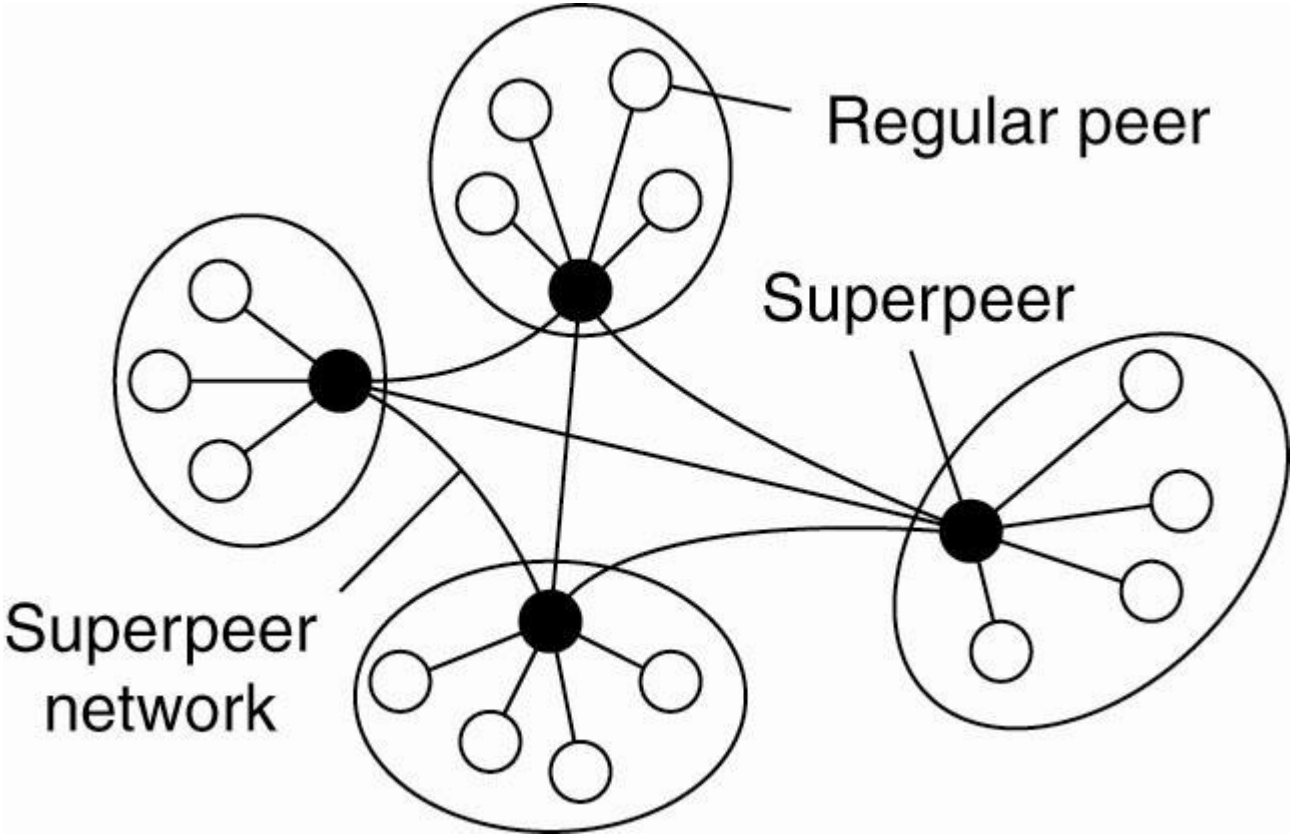

Figure 2-12. A hierarchical organization of nodes into a superpeer network.

Hybrid Architectures

- Edge-server systems
- Collaborative distributed systems

Edge-Server Systems

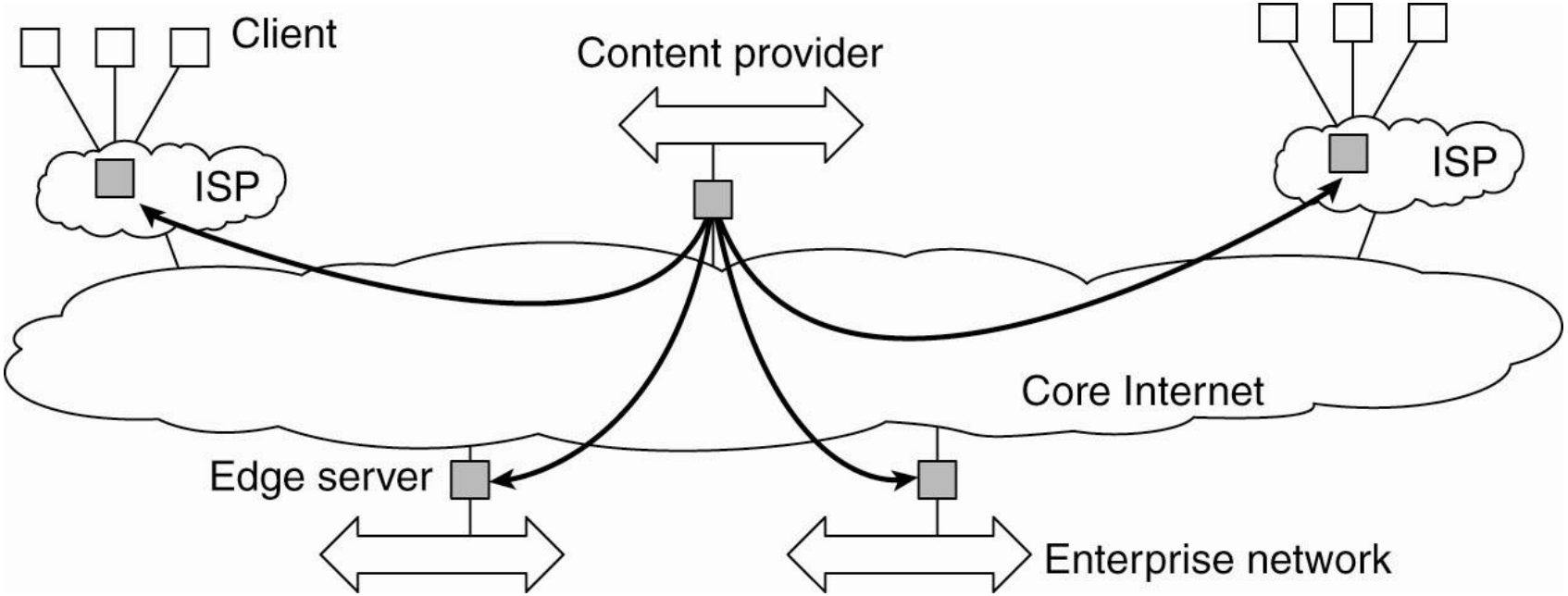

Figure 2-13. Viewing the Internet as consisting of a collection of edge servers.

Collaborative Distributed Systems (1)

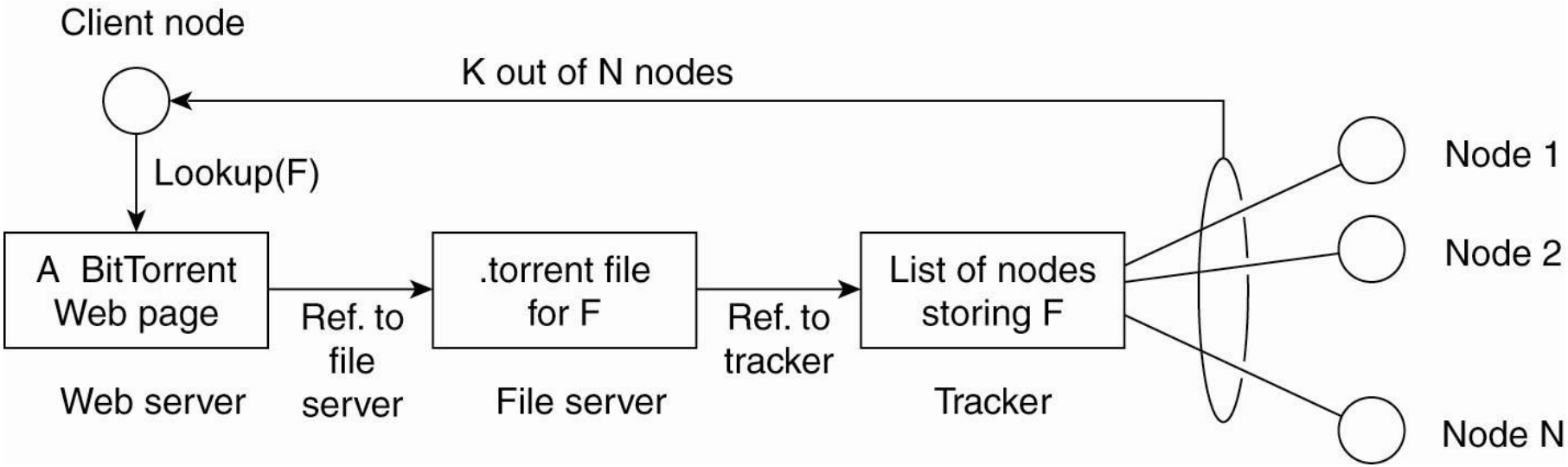

Figure 2-14. The principal working of BitTorrent [adapted with permission from Pouwelse et al. (2004)].

Collaborative Distributed Systems (2)

Components of Globule collaborative content distribution network:

- A component that can redirect client requests to other servers.
- A component for analyzing access patterns.
- A component for managing the replication of Web pages.

Interceptors

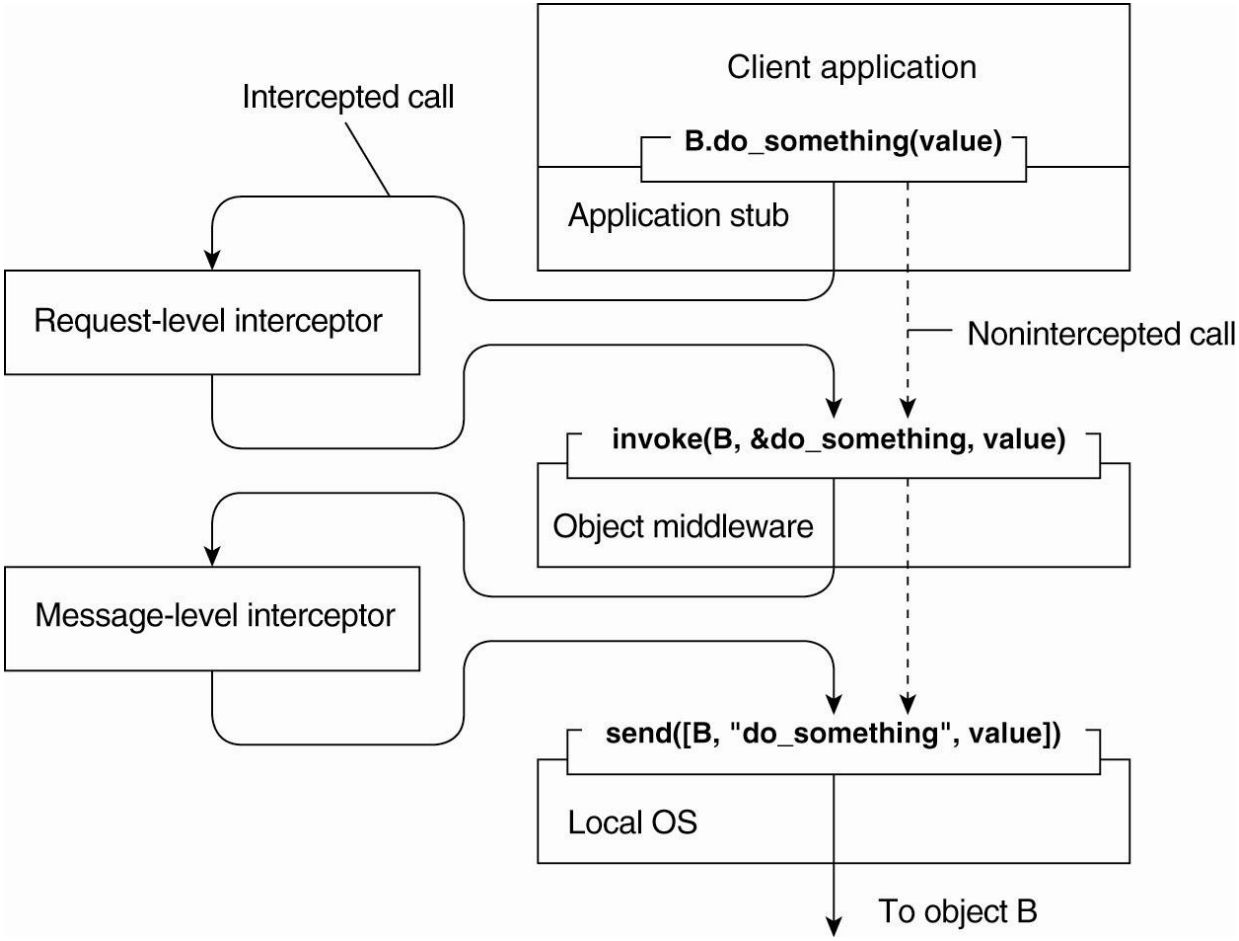

Figure 2-15. Using interceptors to handle remote-object invocations.

General Approaches to Adaptive Software

Three basic approaches to adaptive software:

- Separation of concerns
- Computational reflection
- Component-based design

The Feedback Control Model

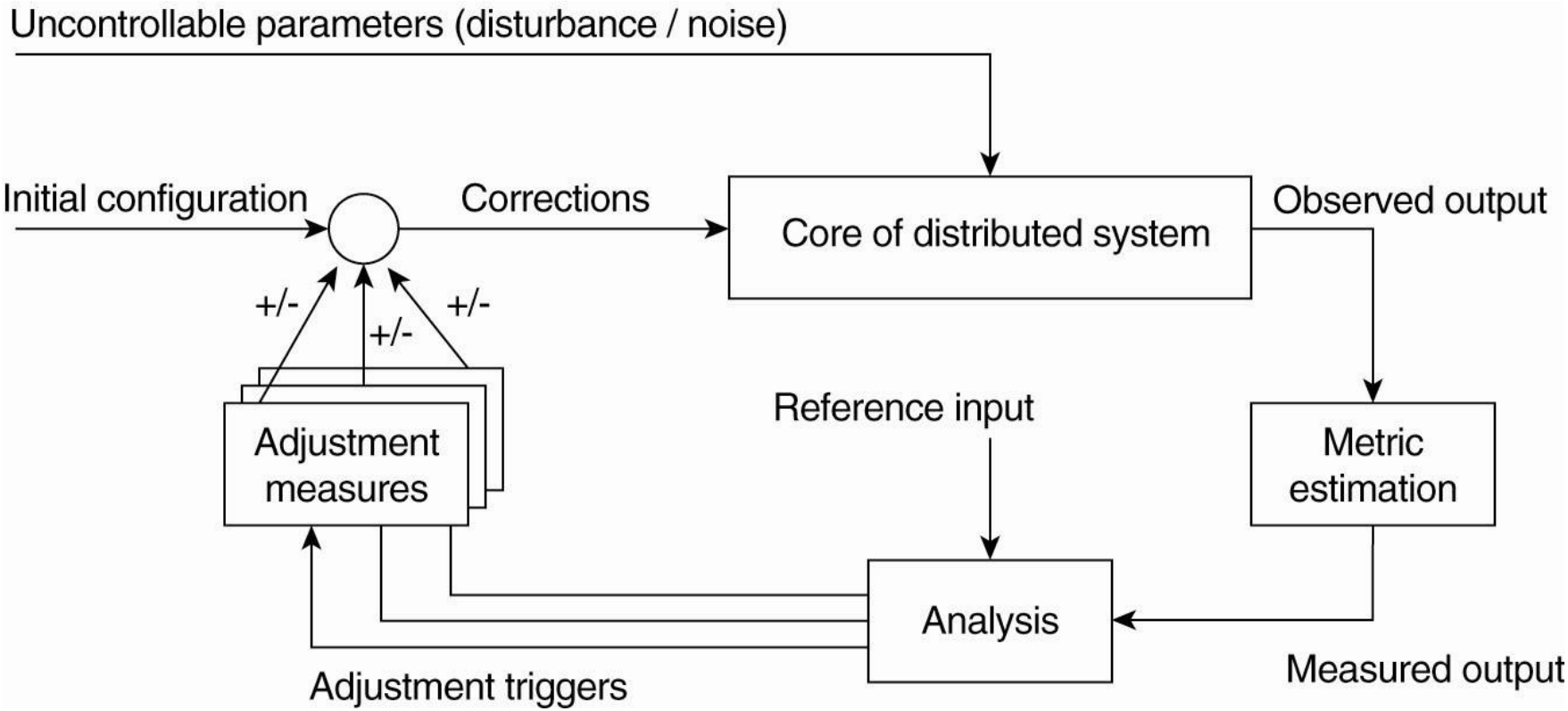

Figure 2-16. The logical organization of a feedback control system.

Example: Systems Monitoring with Astrolabe

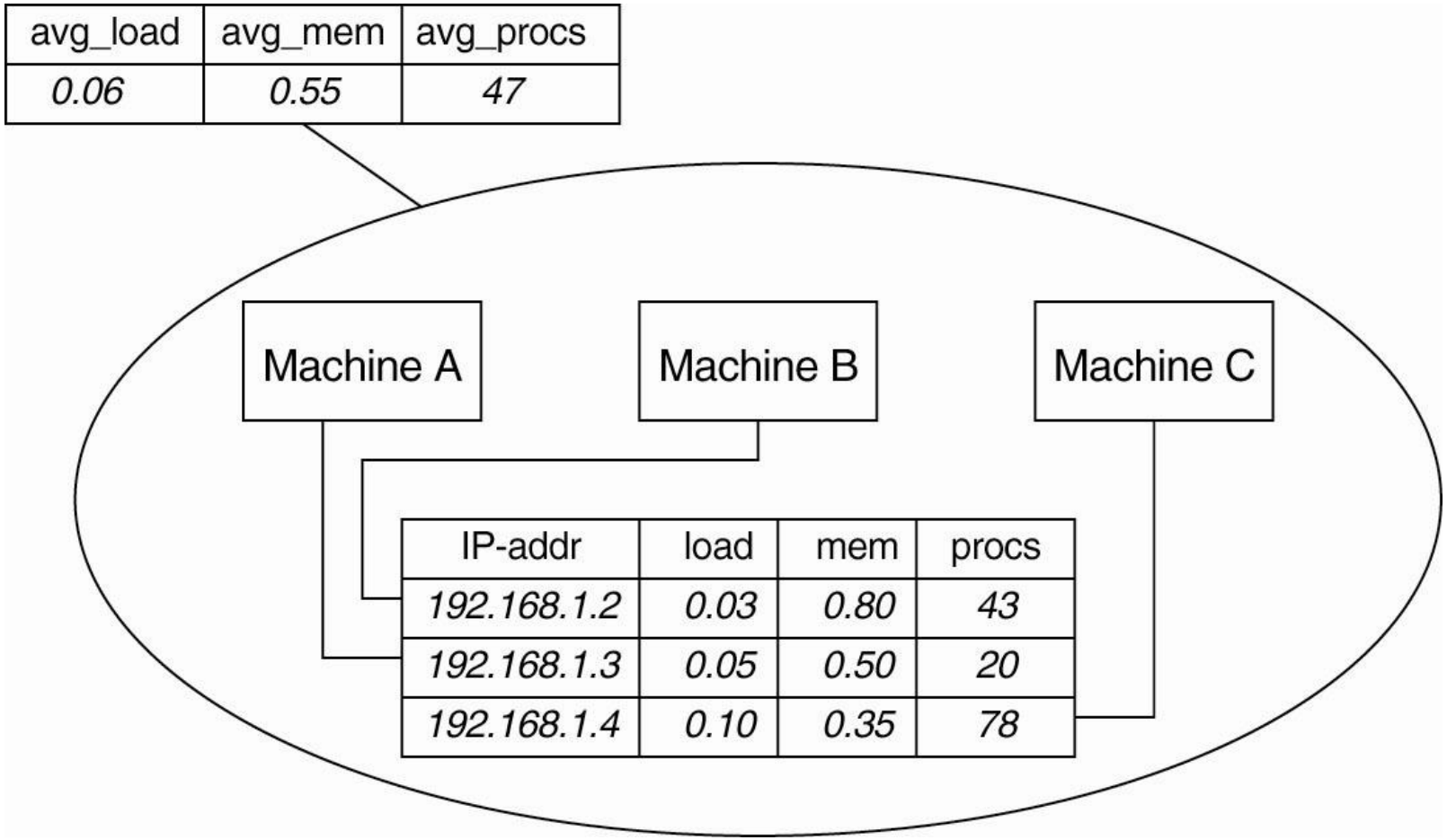

Figure 2-17. Data collection and information aggregation in Astrolabe.

Example: Differentiating Replication Strategies in Globule (1)

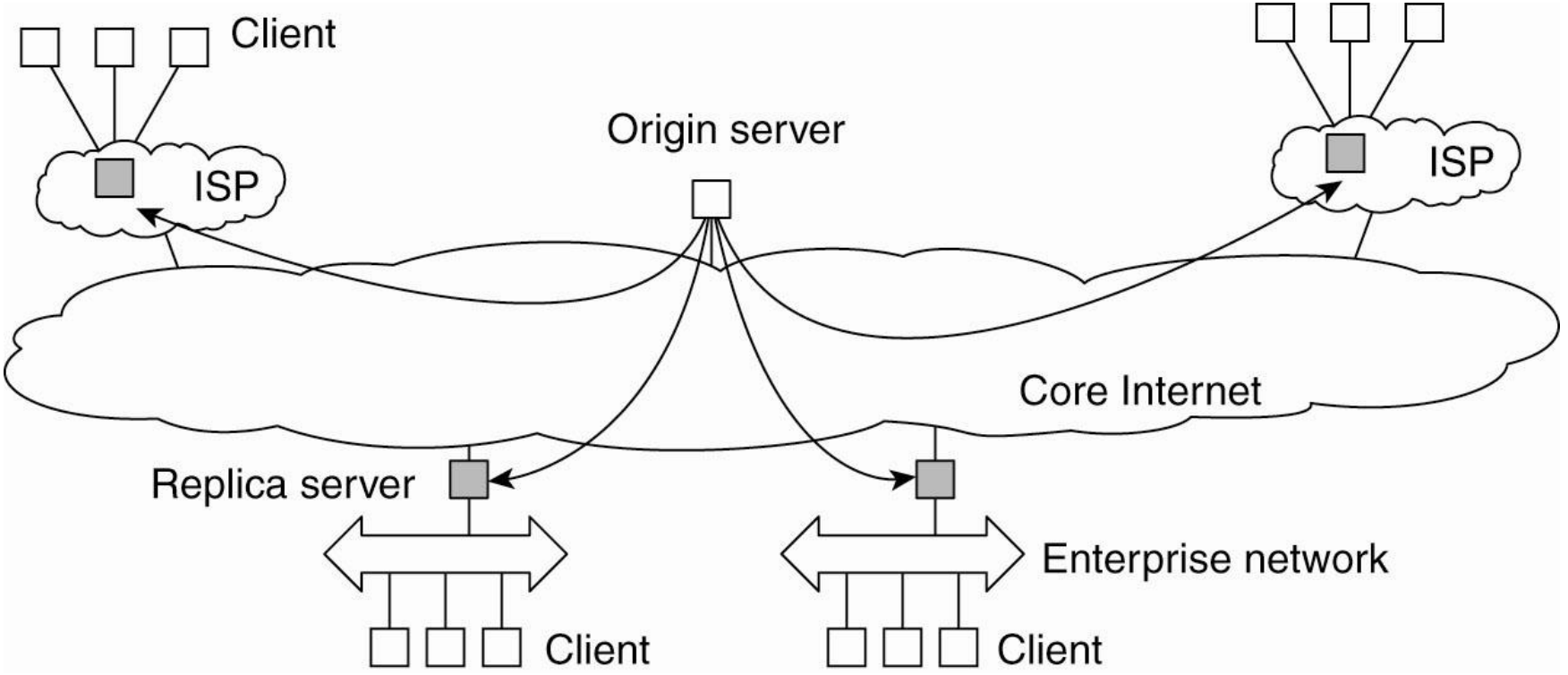

Figure 2-18. The edge-server model assumed by Globule.

Example: Differentiating Replication Strategies in Globule (2)

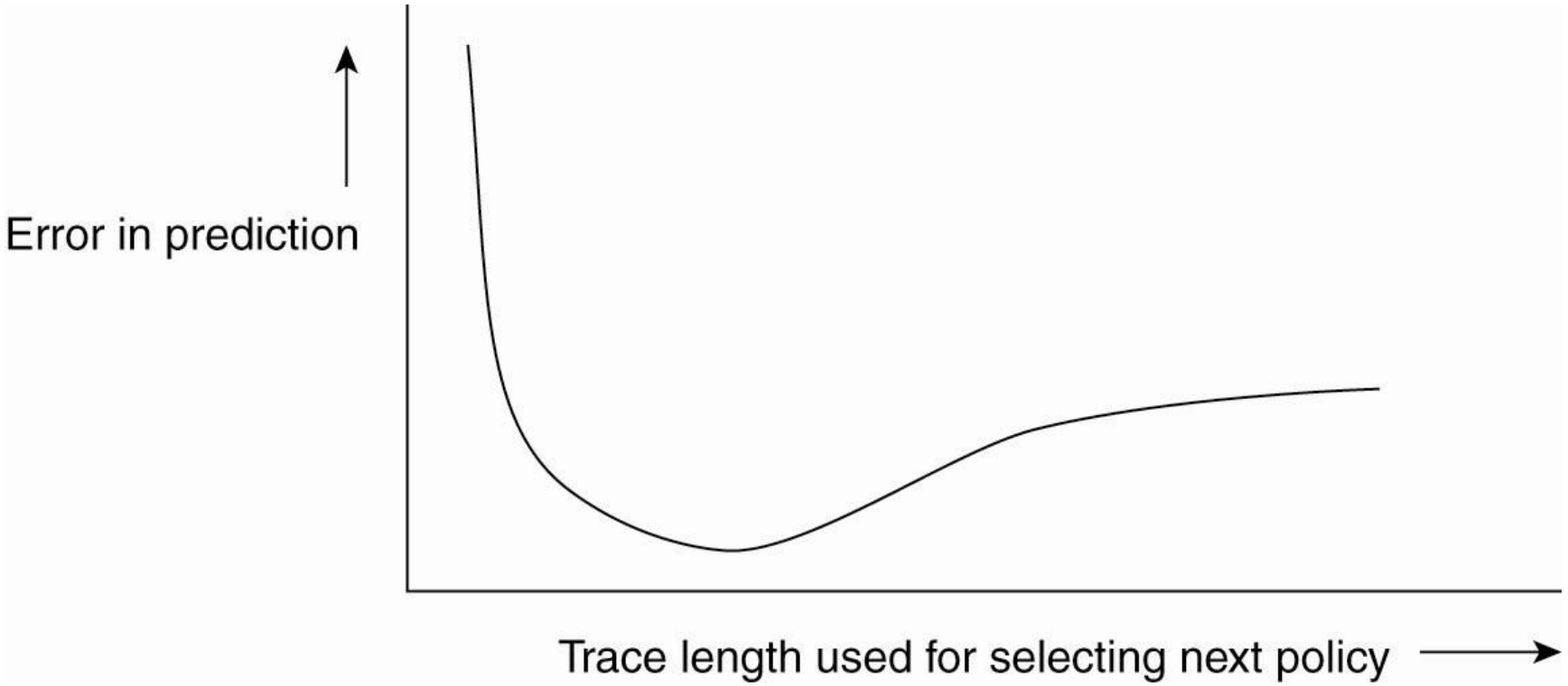

Figure 2-19. The dependency between prediction accuracy and trace length.

Example: Automatic Component Repair Management in Jade

Steps required in a repair procedure:

- Terminate every binding between a component on a non-faulty node, and a component on the node that just failed.
- Request the node manager to start and add a new node to the domain.
- Configure the new node with exactly the same components as those on the crashed node.
- Re-establish all the bindings that were previously terminated.