DISTRIBUTED SYSTEMS
Principles and Paradigms
Second Edition

ANDREW S. TANENBAUM
MAARTEN VAN STEEN

Chapter 1
Architecture

Lecture Outline

- Architecture Styles
  - Layered Architecture
    - Client/Server
    - Multitier Architecture
  - Peer to Peer
    - Structured P2P
    - Unstructured P2P
    - Hybrid P2P
    - Collaborative

Architectural Styles (1)

Important styles of architecture for distributed systems

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

Architectural Styles (2)

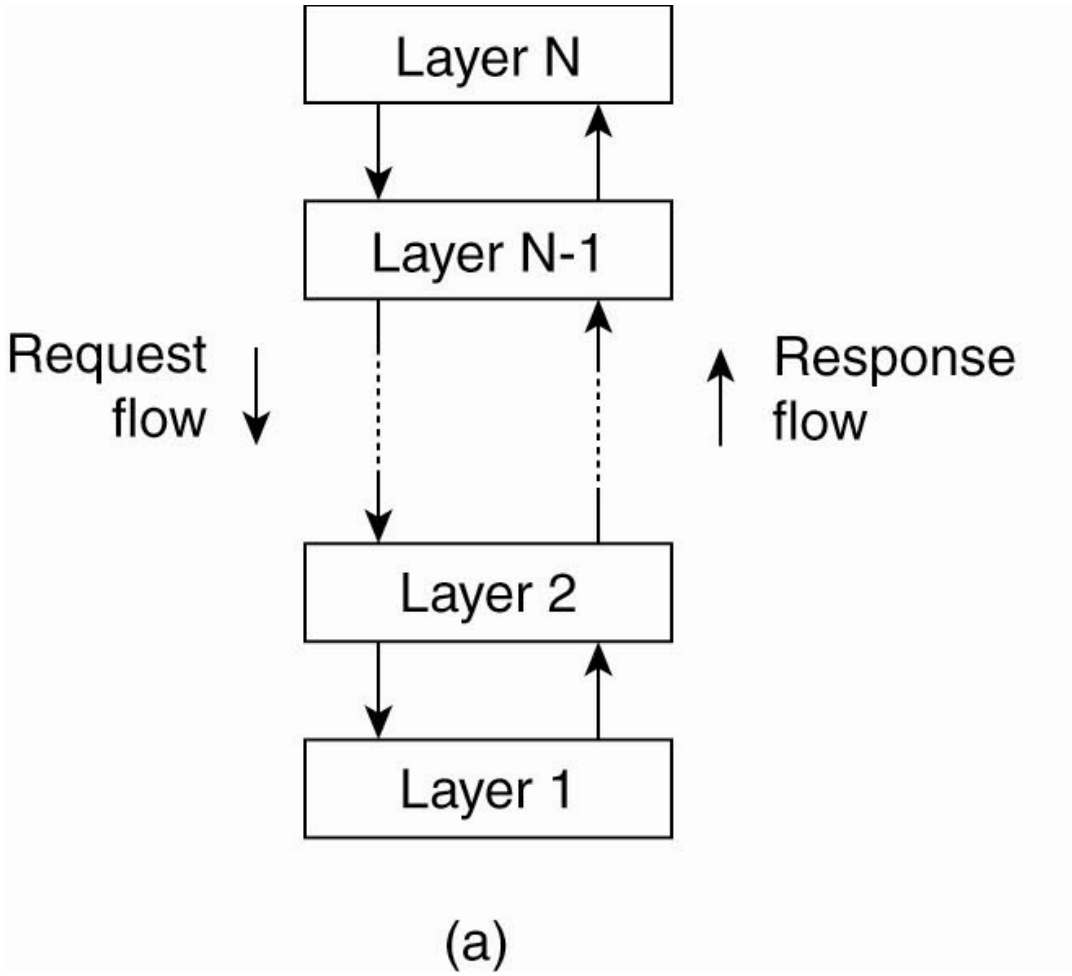


Figure 2-1. The (a) layered architectural style and …

Architectural Styles (3)

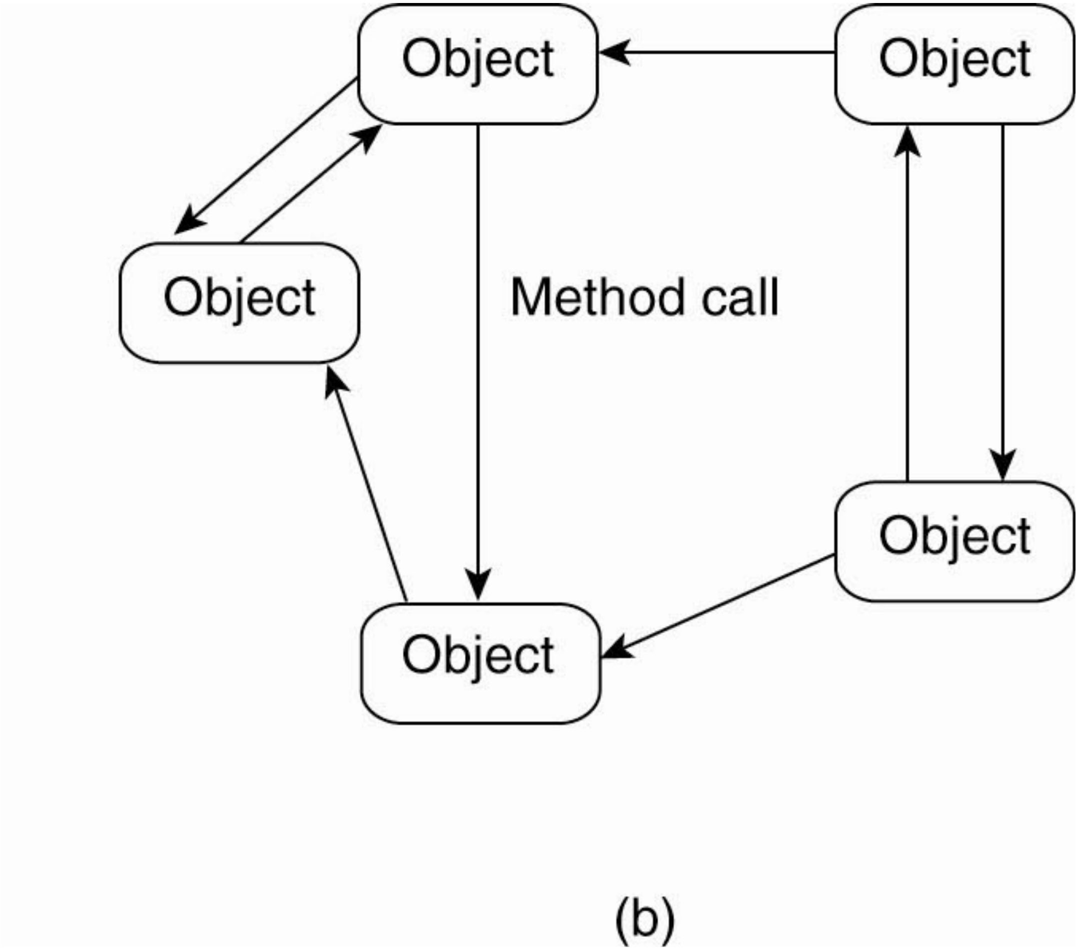


Figure 2-1. (b) The object-based architectural style.

Architectural Styles (4)

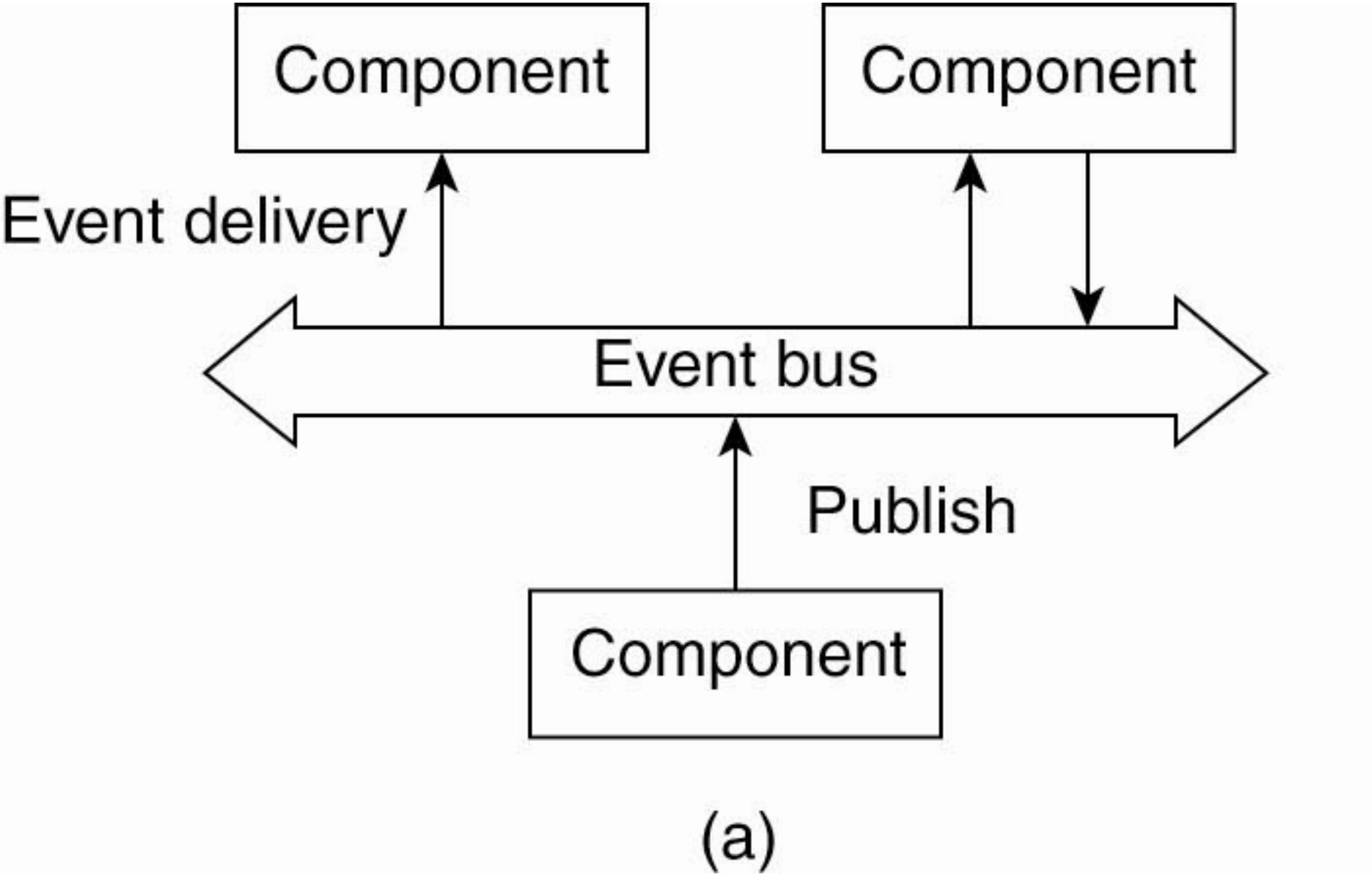


Figure 2-2. (a) The event-based architectural style and …

Architectural Styles (5)

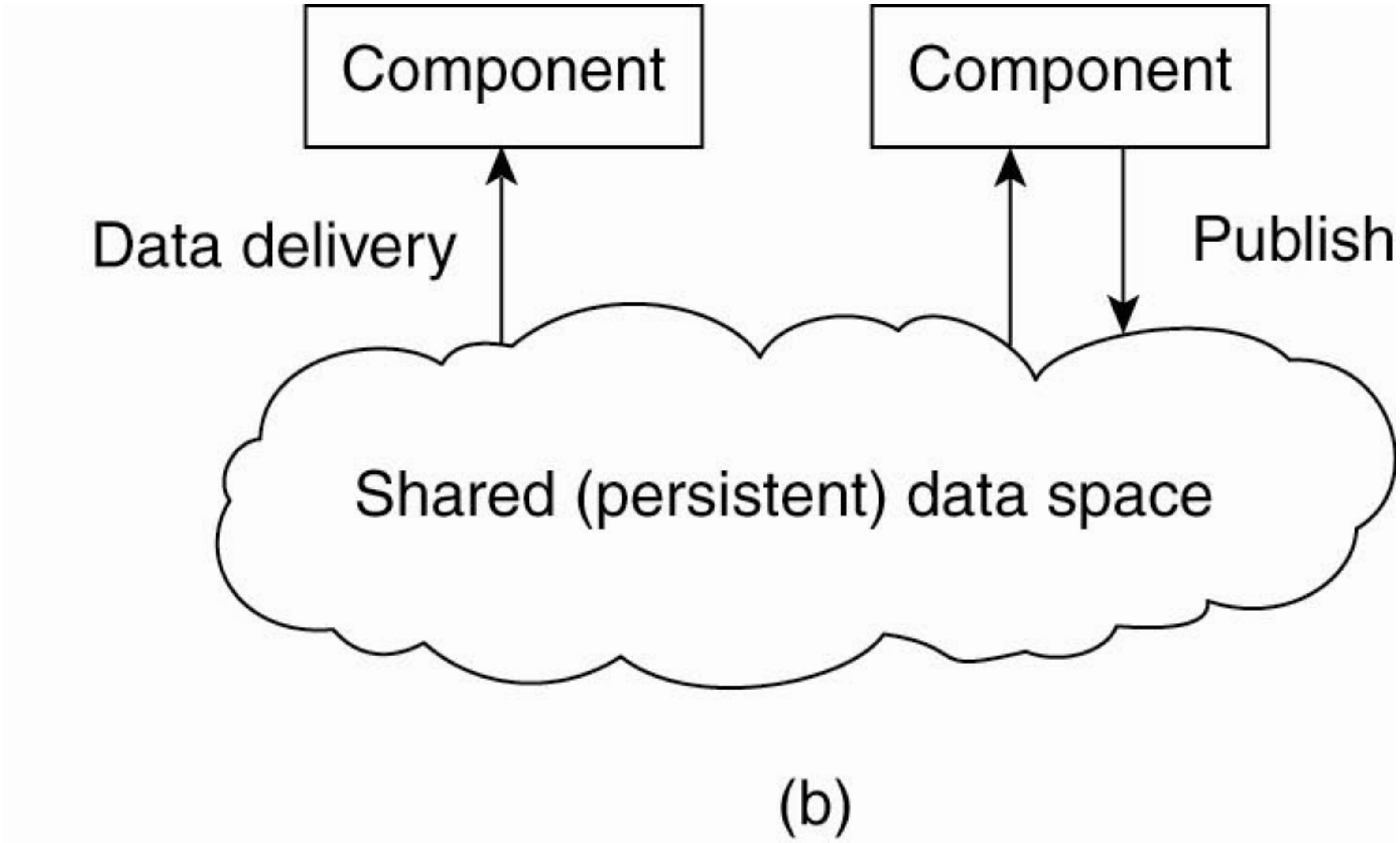


Figure 2-2. (b) The shared data-space architectural style.

Centralized Architectures

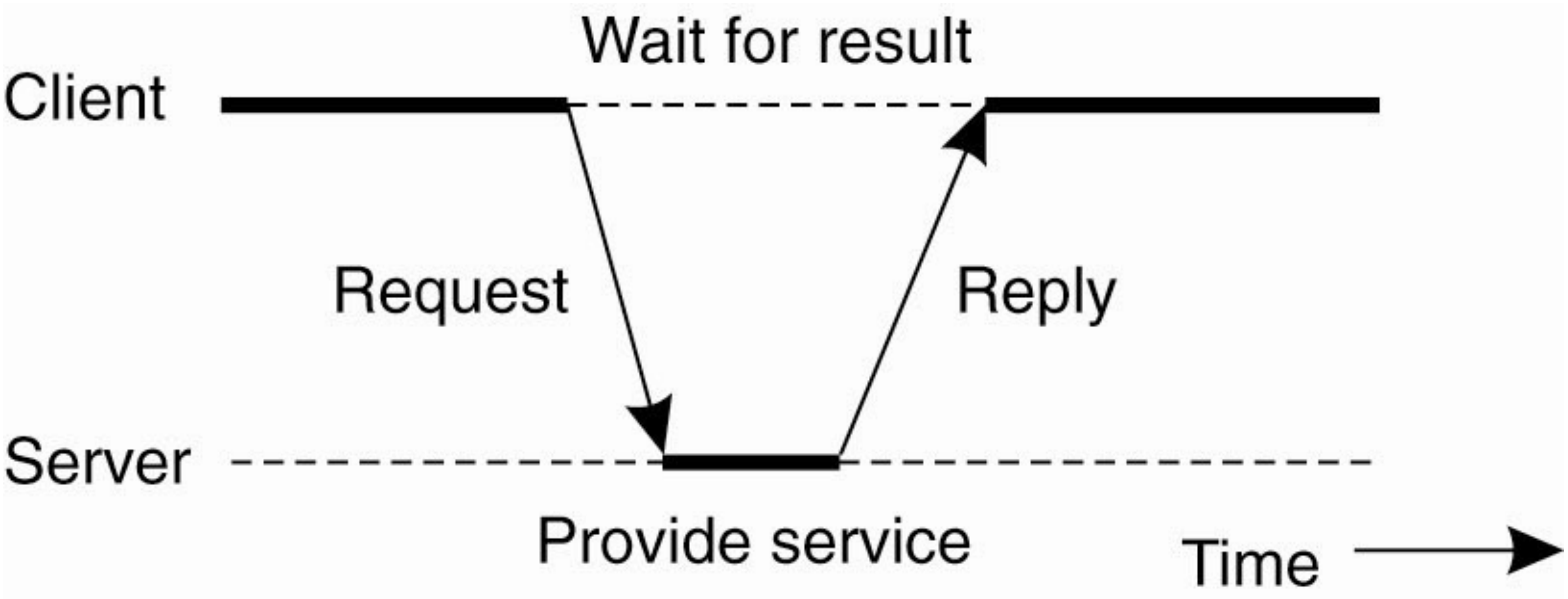


Figure 2-3. General interaction between a client and a server.

Application Layering (1)

Recall previously mentioned layers of architectural style

- The user-interface level
- The processing level
- The data level

Application Layering (2)

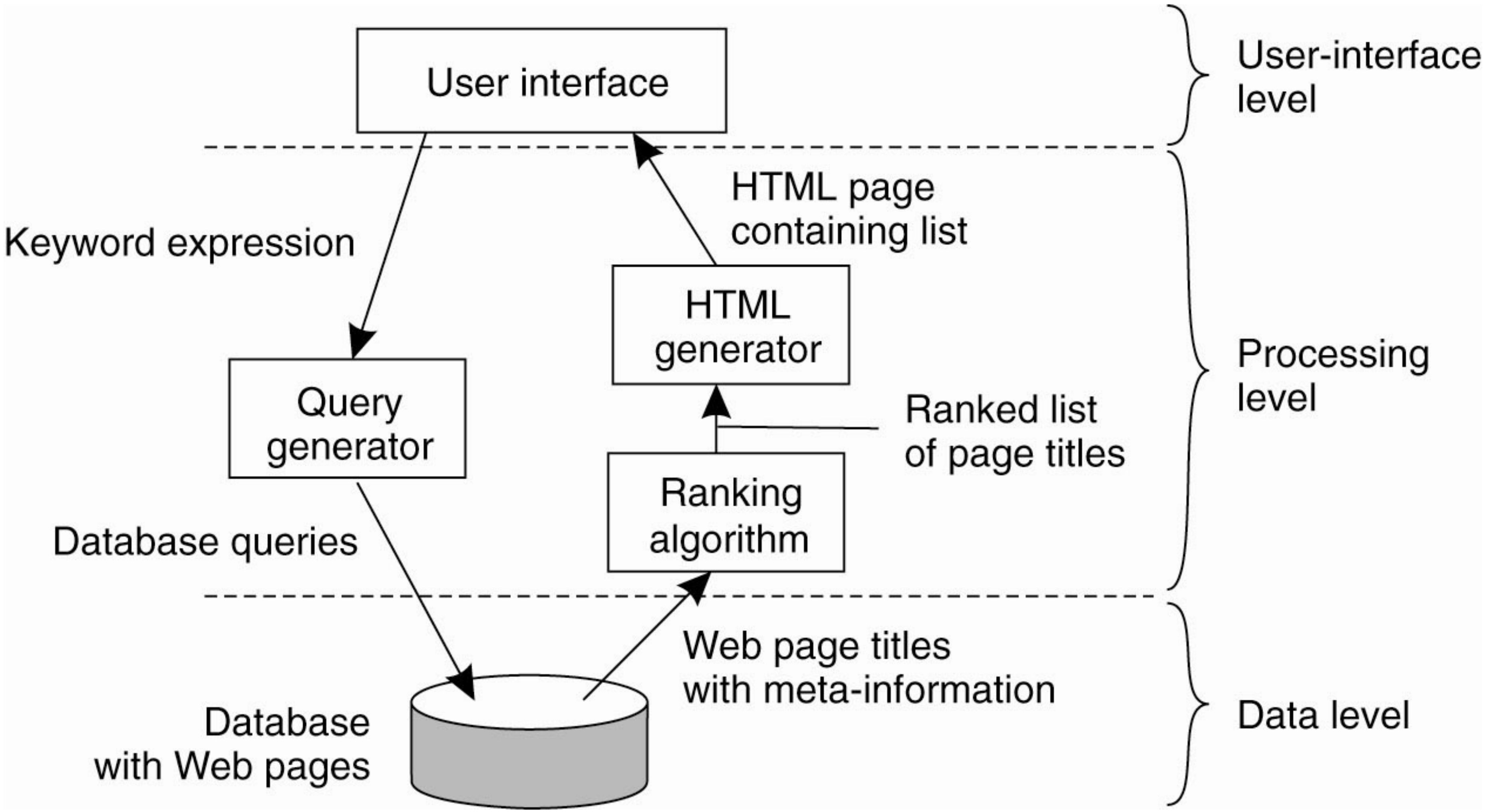


Figure 2-4. The simplified organisation of an Internet search engine into three different layers.

Multitiered Architectures (1)

The simplest organisation 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 (2)

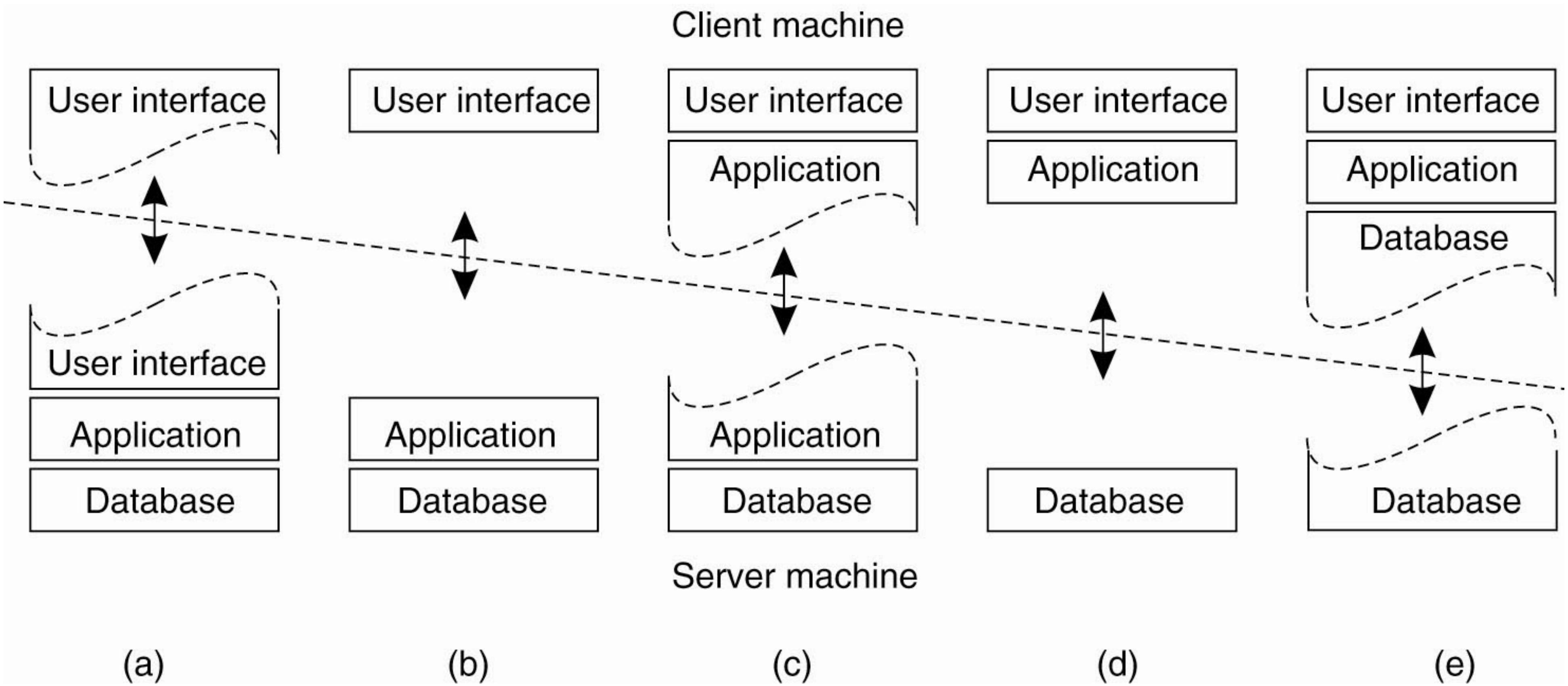


Figure 2-5. Alternative client-server organisations (a)–(e).

Multitiered Architectures (3)

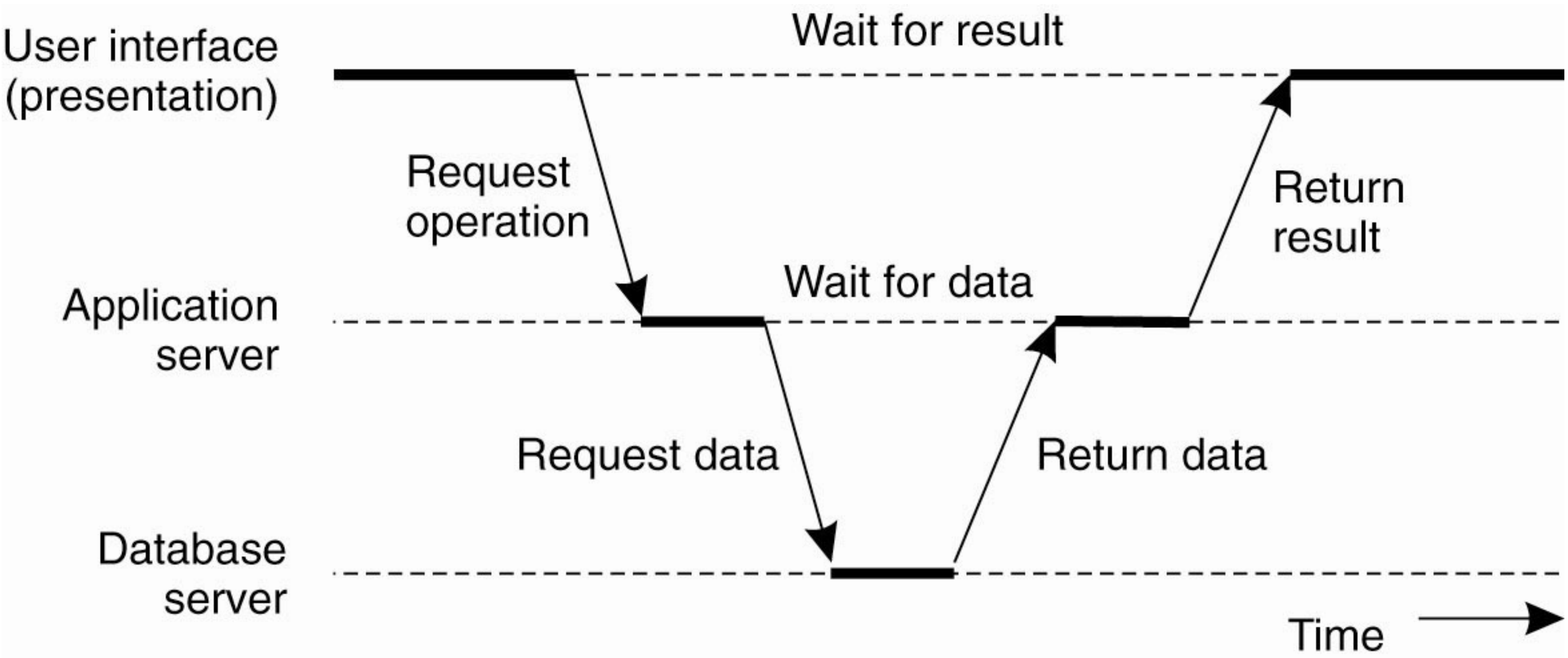
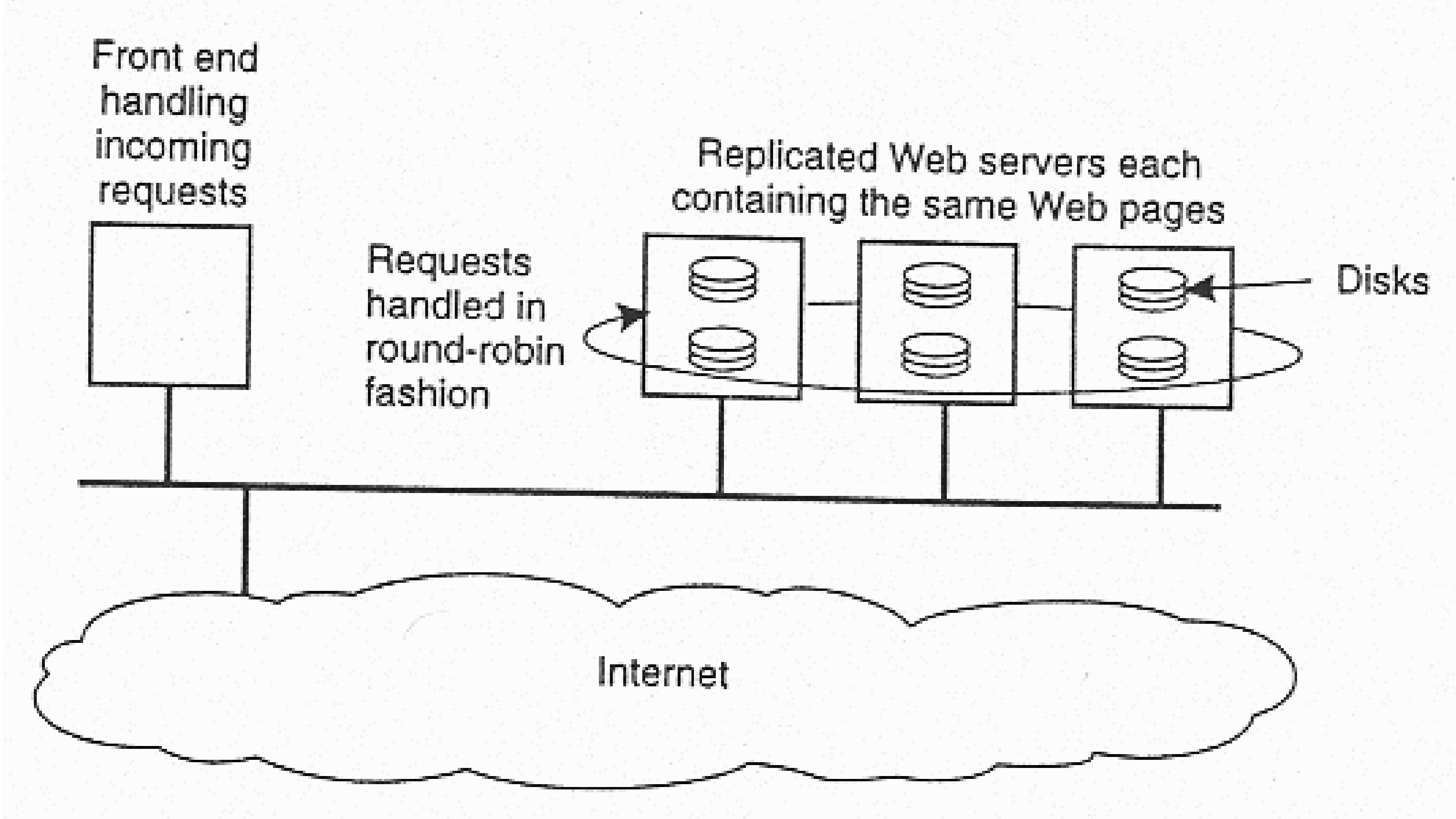


Figure 2-6. An example of a server acting as client.

Multi-tiered Vertical vs. Horizontal Architectures



- Vertical architecture is placing logically different components on different machines. It is related to vertical fragmentation concept used in distributed relational databases (tables are split column wise).
- Horizontal architecture (shown in figure), is placing shares of datasets on different machines (acting as clients or servers) to balance the load.

Decentralised Architectures - P2P

Observation

In the last couple of years 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

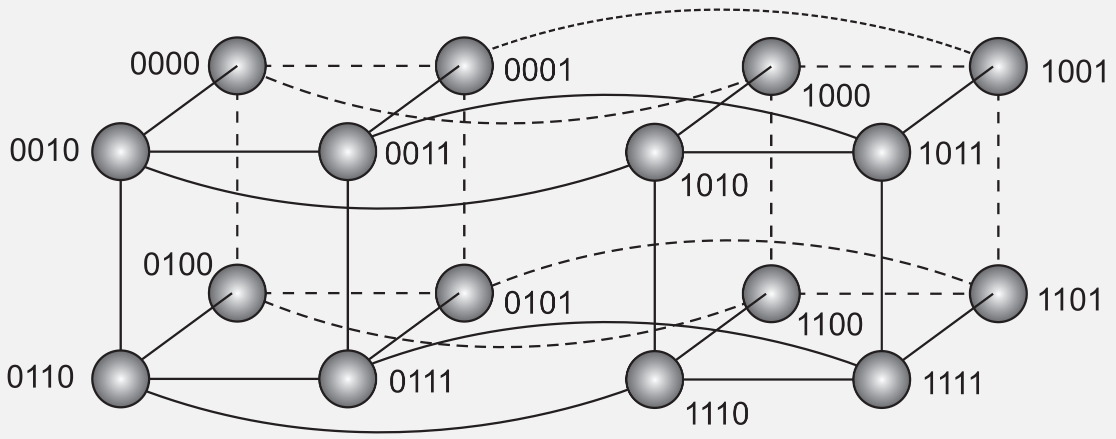
Note

In virtually all cases, we are dealing with overlay networks: data is routed over connections setup between the nodes (cf. application-level multicasting)

Structured P2P

Basic idea

Organize the nodes in a structured overlay network such as a logical ring, or a hypercube, and make specific nodes responsible for services based only on their ID.



Note

The system provides an operation *LOOKUP(key)* that will efficiently route the lookup request to the associated node.

Structured Peer-to-Peer Architectures (1)

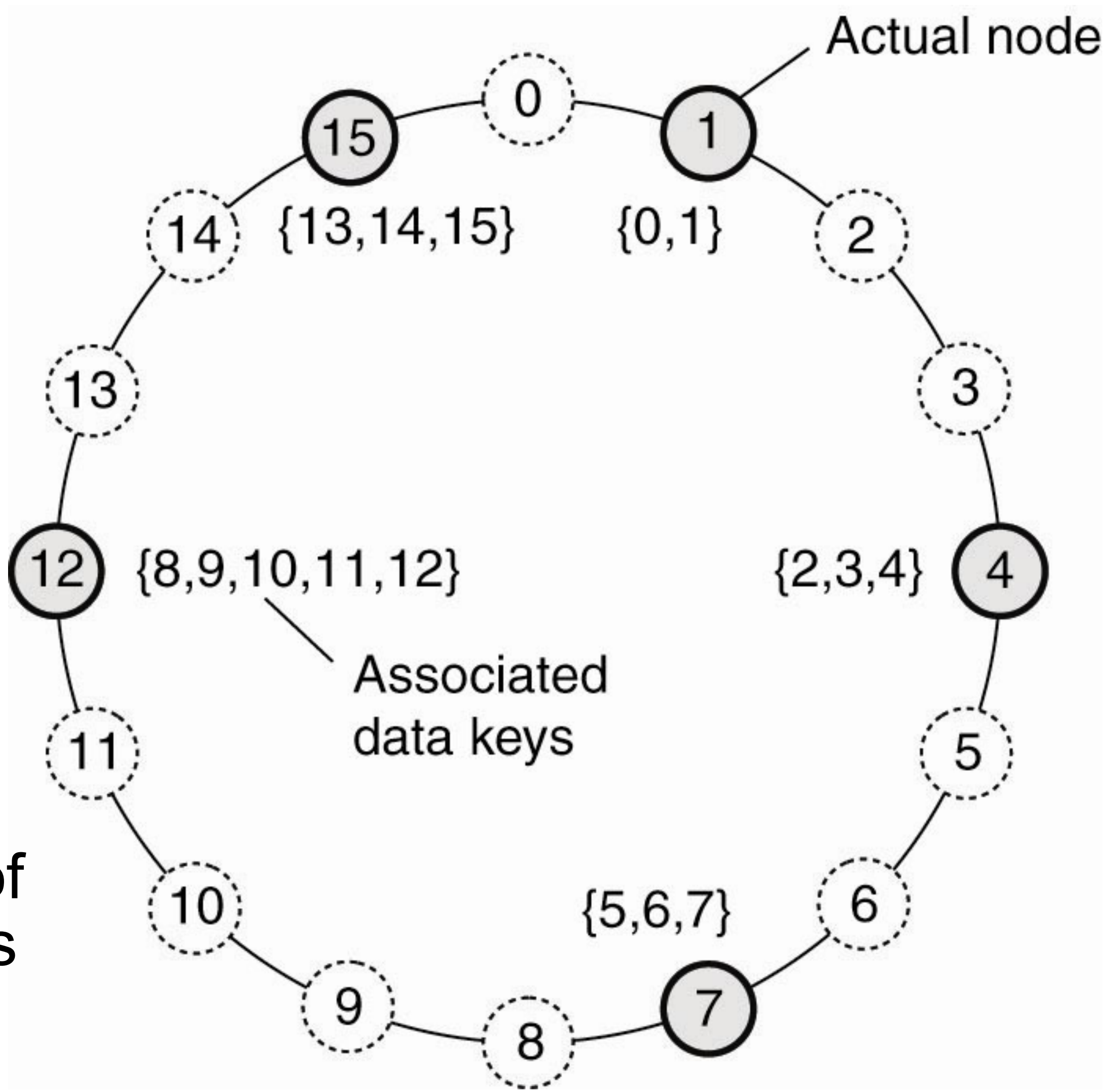


Figure 2-7. The mapping of data items onto nodes in Chord.

Structured Peer-to-Peer Architectures (2)

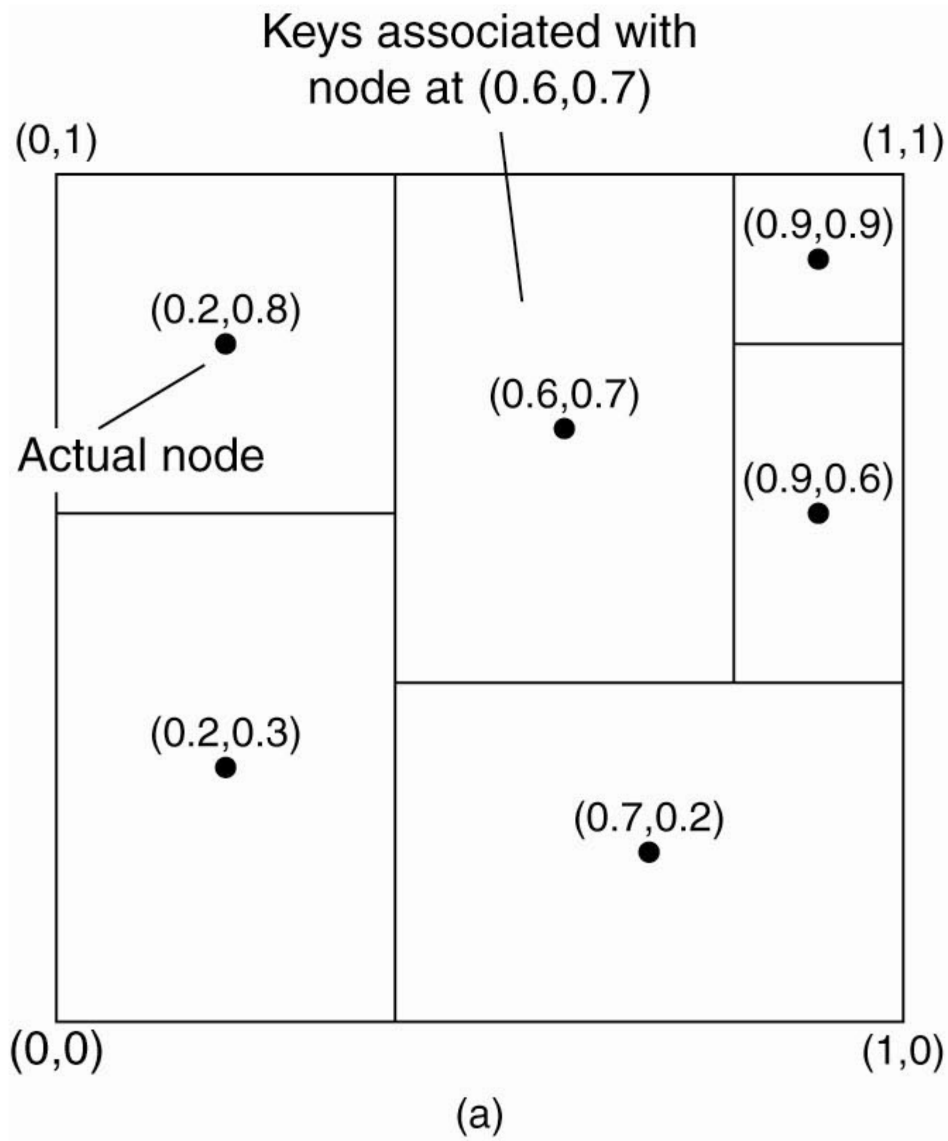
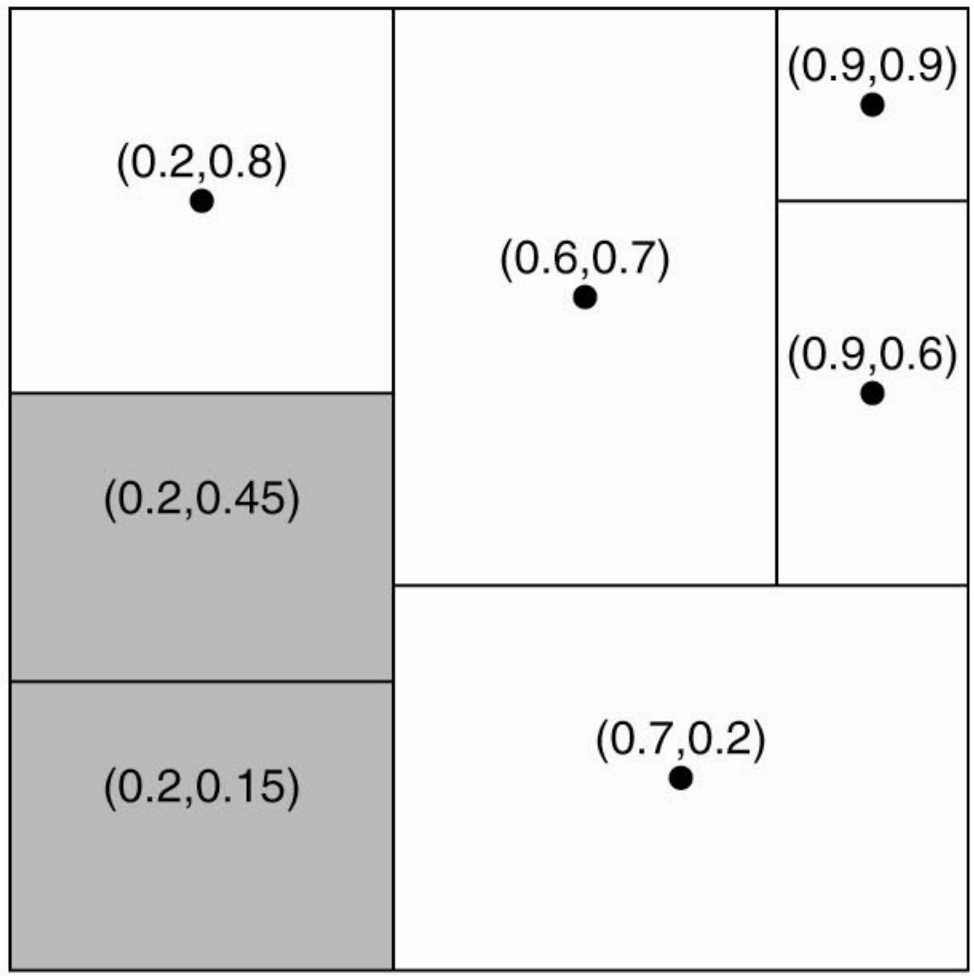


Figure 2-8. (a) The mapping of data items onto nodes in CAN.

Structured Peer-to-Peer Architectures (3)



(b)

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

Unstructured P2P

Essence

Many unstructured P2P systems are organized as a random overlay: two nodes are linked with probability p.

Observation

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 request. There are many variations:
  - Limited flooding (maximal number of forwarding)
  - Probabilistic flooding (flood only with a certain probability).
- Random walk: Randomly select a neighbor v. If v has the answer, it replies, otherwise v randomly selects one of *its* neighbors. Variation: parallel random walk. Works well with replicated data.

Unstructured Peer-to-Peer Architectures (1)

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 (2)

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)

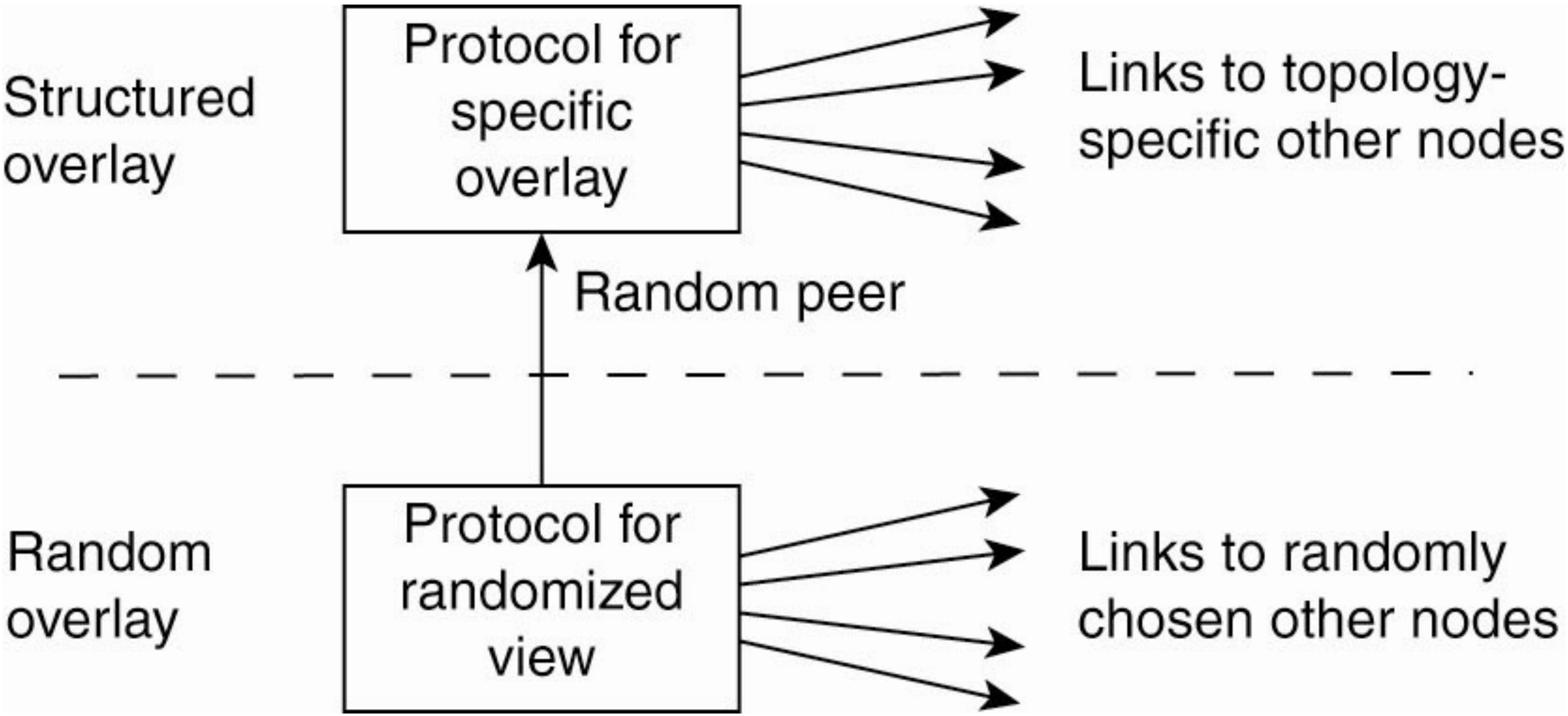


Figure 2-10. A two-layered approach for constructing and maintaining specific overlay topologies using techniques from unstructured peer-to-peer systems.

Topology Management of Overlay Networks (2)

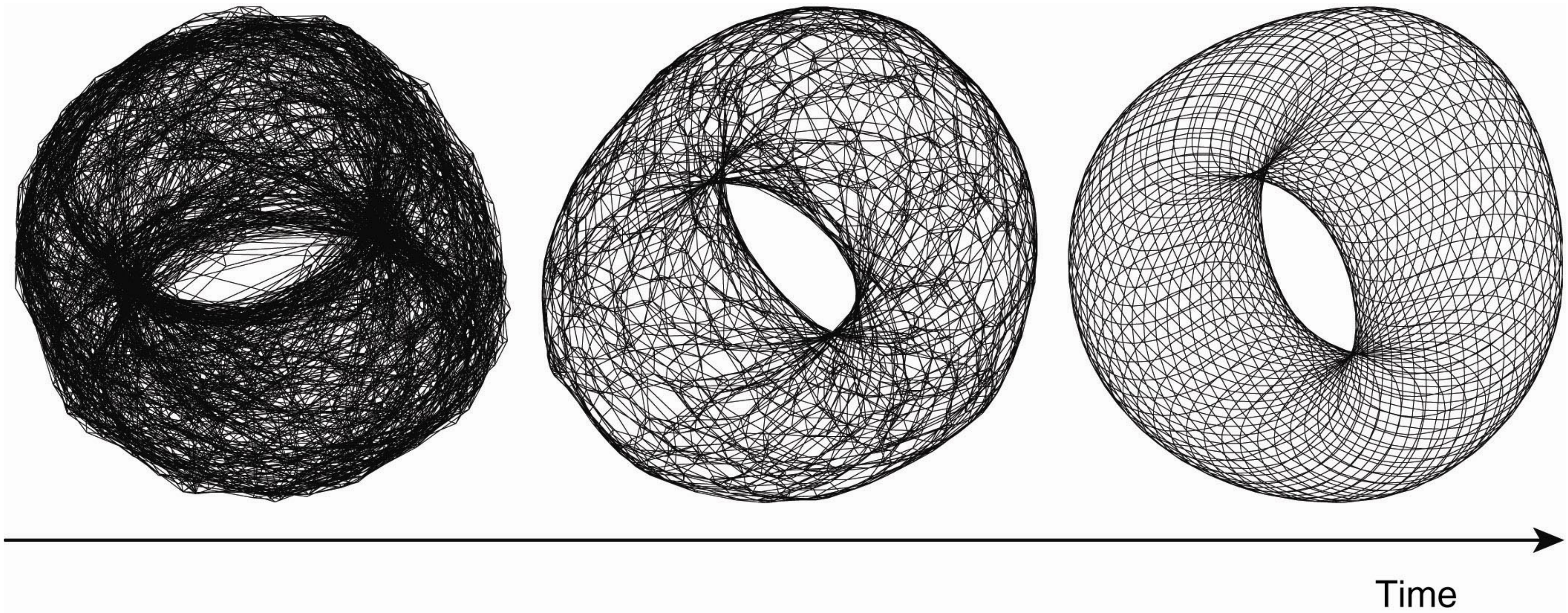
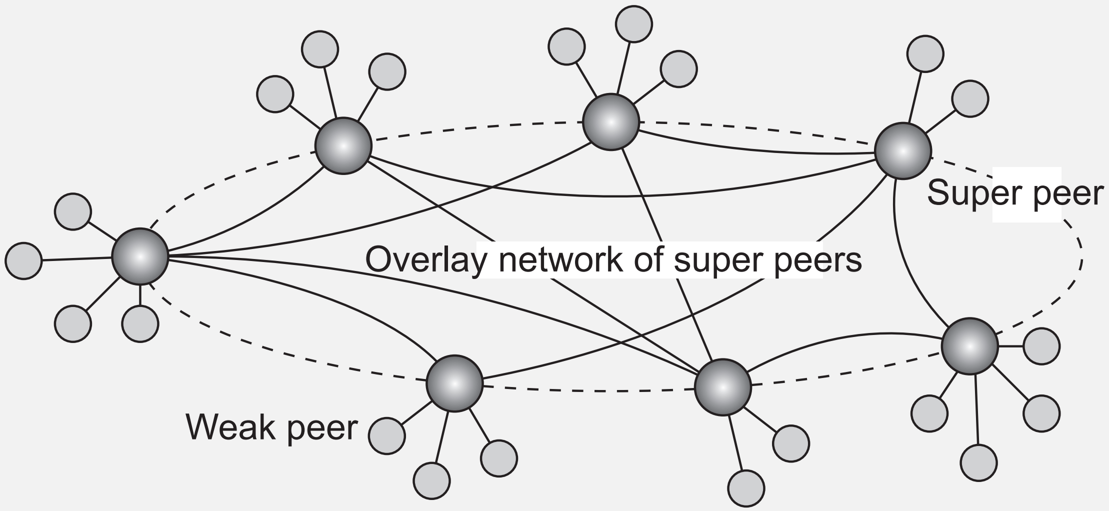


Figure 2-11. Generating a specific overlay network using a two-layered unstructured peer-to-peer system [adapted with permission from Jelasity and Babaoglu (2005)].

Superpeers

Observation

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



Examples

- Peers maintaining an index (for search)
- Peers monitoring the state of the network
- Peers being able to setup connections

Edge-Server Systems

Hybrid Architectures: Client-server combined with P2P: Edge-server architectures, which are often used for Content Delivery Networks

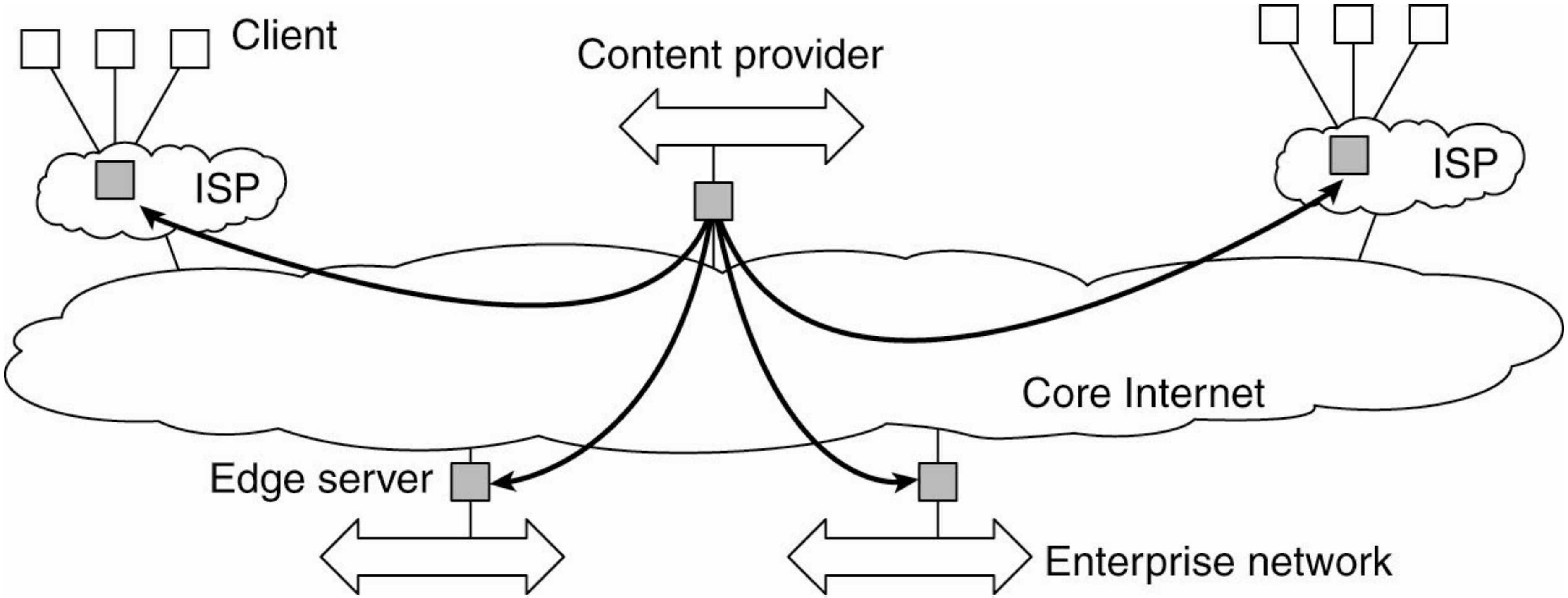
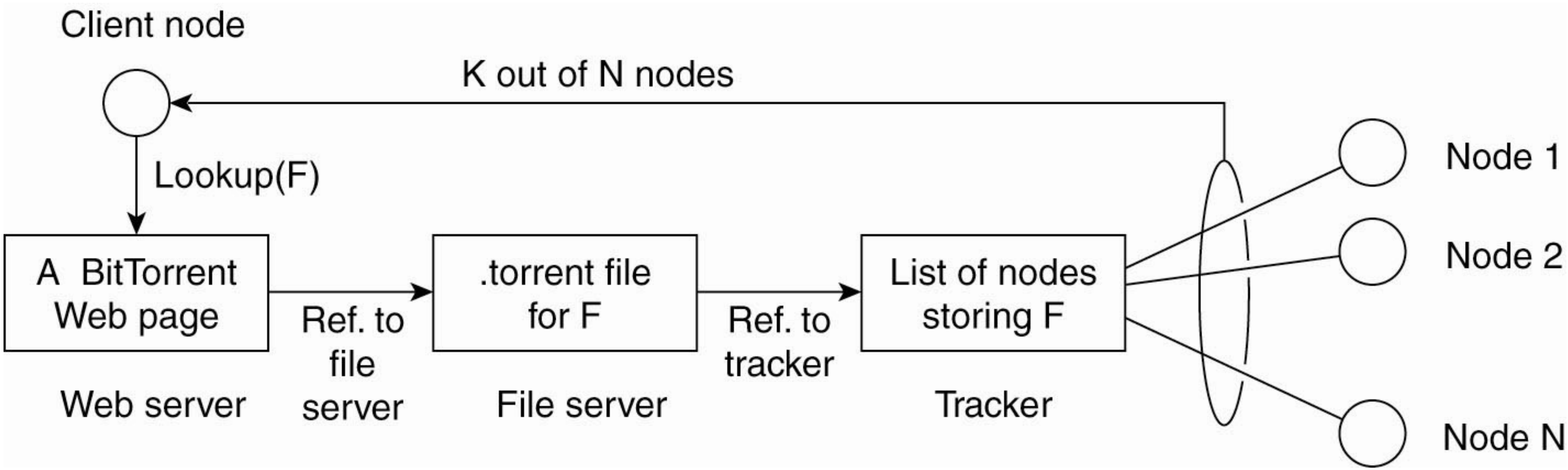


Figure 2-13. Viewing the Internet as consisting of a collection of edge servers.

Collaborative Distributed Systems (1)

Hybrid Architectures: C/S with P2P – BitTorrent



Basic idea

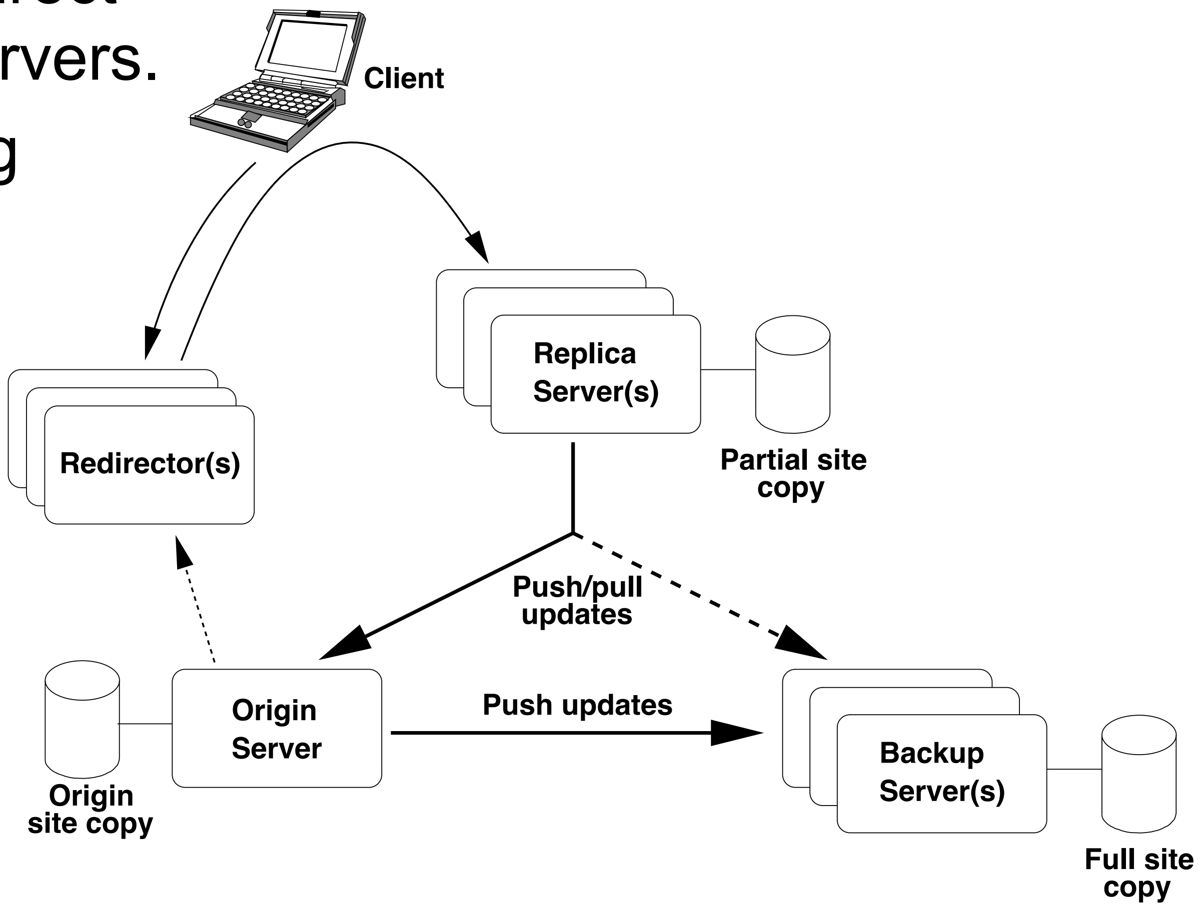
Once a node has identified where to download a file from, it joins a swarm of downloaders who in parallel get file chunks from the source, but also distribute these chunks amongst each other

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 analysing access patterns.
- A component for managing the replication of Web pages.

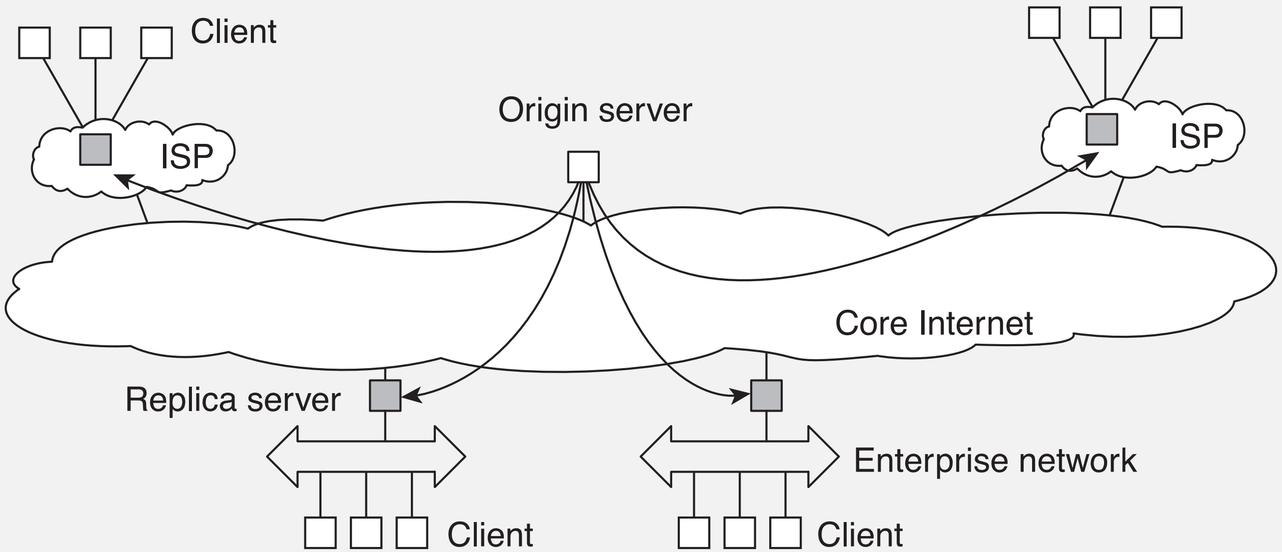


Example: Differentiating Replication Strategies in Globule (1)

Globule

Collaborative CDN that analyzes traces to decide where replicas of Web content should be placed. Decisions are driven by a general cost model:

$$cost = (w_1 \times m_1) + (w_2 \times m_2) + \cdots + (w_n \times m_n)$$



- Globule origin server collects traces and does what-if analysis by checking what would have happened if page *P* would have been placed at edge server *S*.
- Many strategies are evaluated, and the best one is chosen.

Example: Differentiating Replication Strategies in Globule (2)

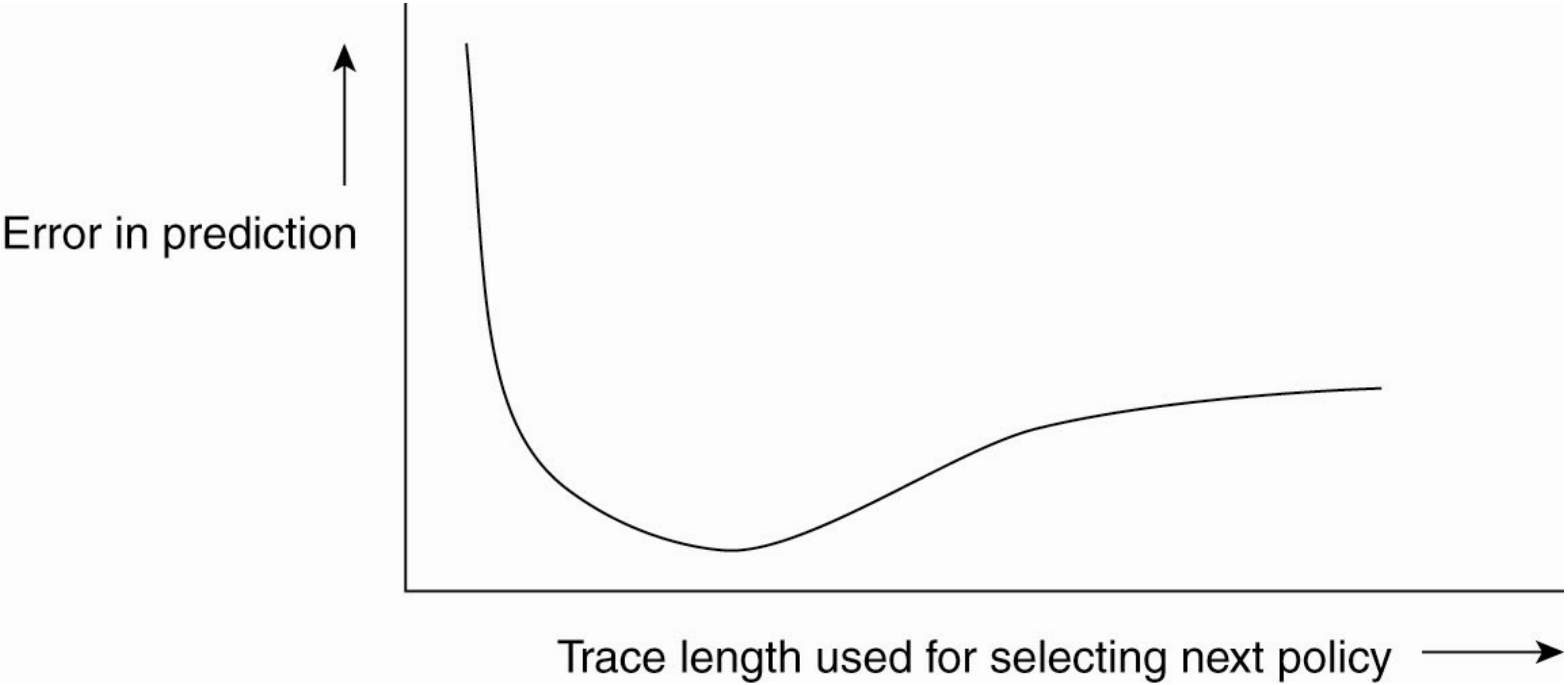


Figure 2-19. The dependency between prediction accuracy and trace length.

Architectures versus Middleware

Problem

In many cases, distributed systems/applications are developed according to a specific architectural style. The chosen style may not be optimal in all cases $\Rightarrow$ need to (dynamically) adapt the behavior of the middleware.

Interceptors

Intercept the usual flow of control when invoking a remote object.

Interceptors

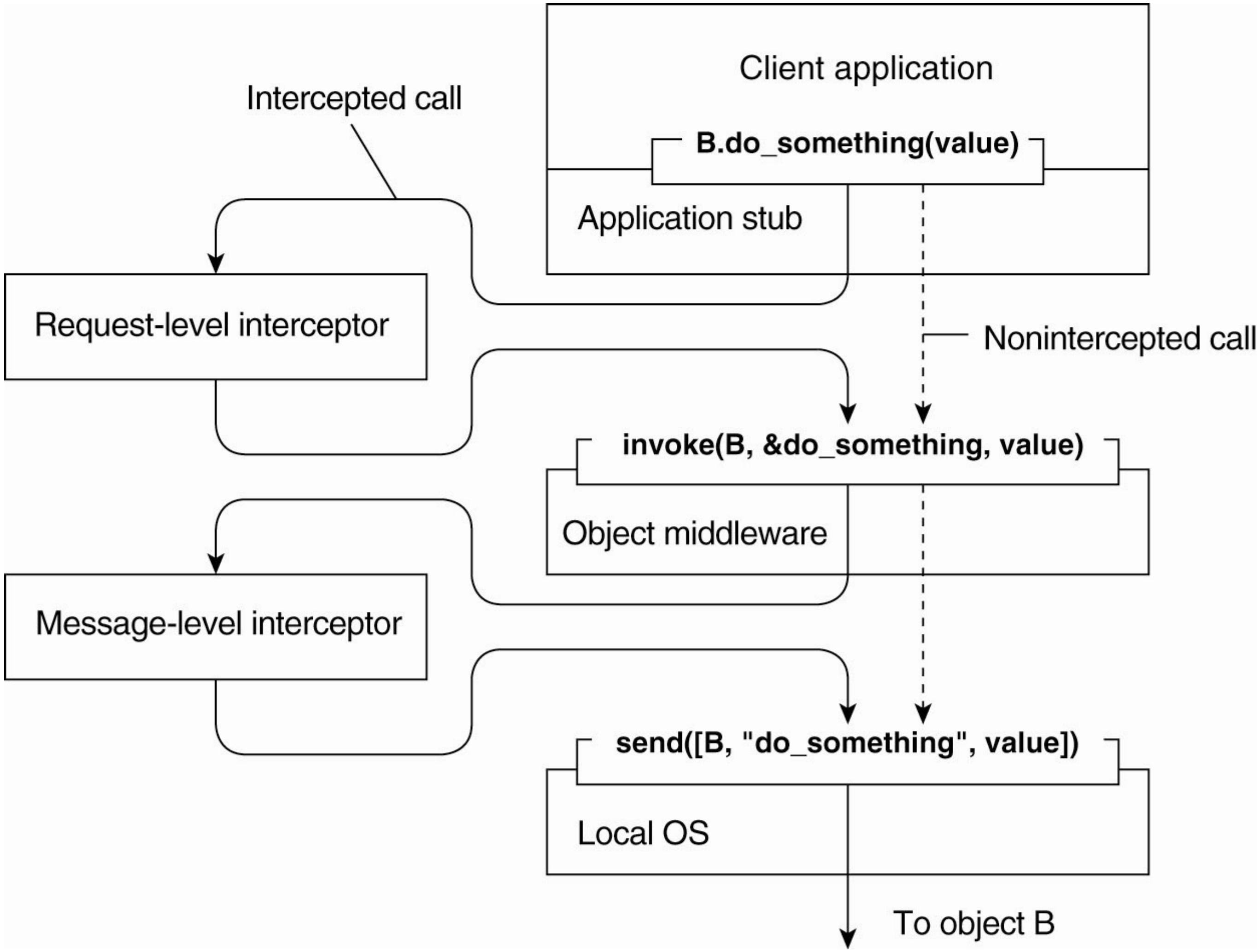


Figure 2-15. Using interceptors to handle remote-object invocations.

General Approaches to Adaptive Software

Separation of concerns: Try to separate extra functionalities and later weave them together into a single implementation $\Rightarrow$ only toy examples so far.

Computational reflection: Let a program inspect itself at runtime and adapt/change its settings dynamically if necessary $\Rightarrow$ mostly at language level and applicability unclear.

Component-based design: Organize a distributed application through components that can be dynamically replaced when needed $\Rightarrow$ highly complex, also many intercomponent dependencies.

Fundamental question

Do we need adaptive software at all, or is the issue adaptive systems?

Self-managing Distributed Systems

Observation

Distinction between system and software architectures blurs when automatic adaptivity needs to be taken into account:

- Self-configuration
- Self-managing
- Self-healing
- Self-optimizing
- Self-*

Warning

There is a lot of hype going on in this field of autonomic computing.

The Feedback Control Model

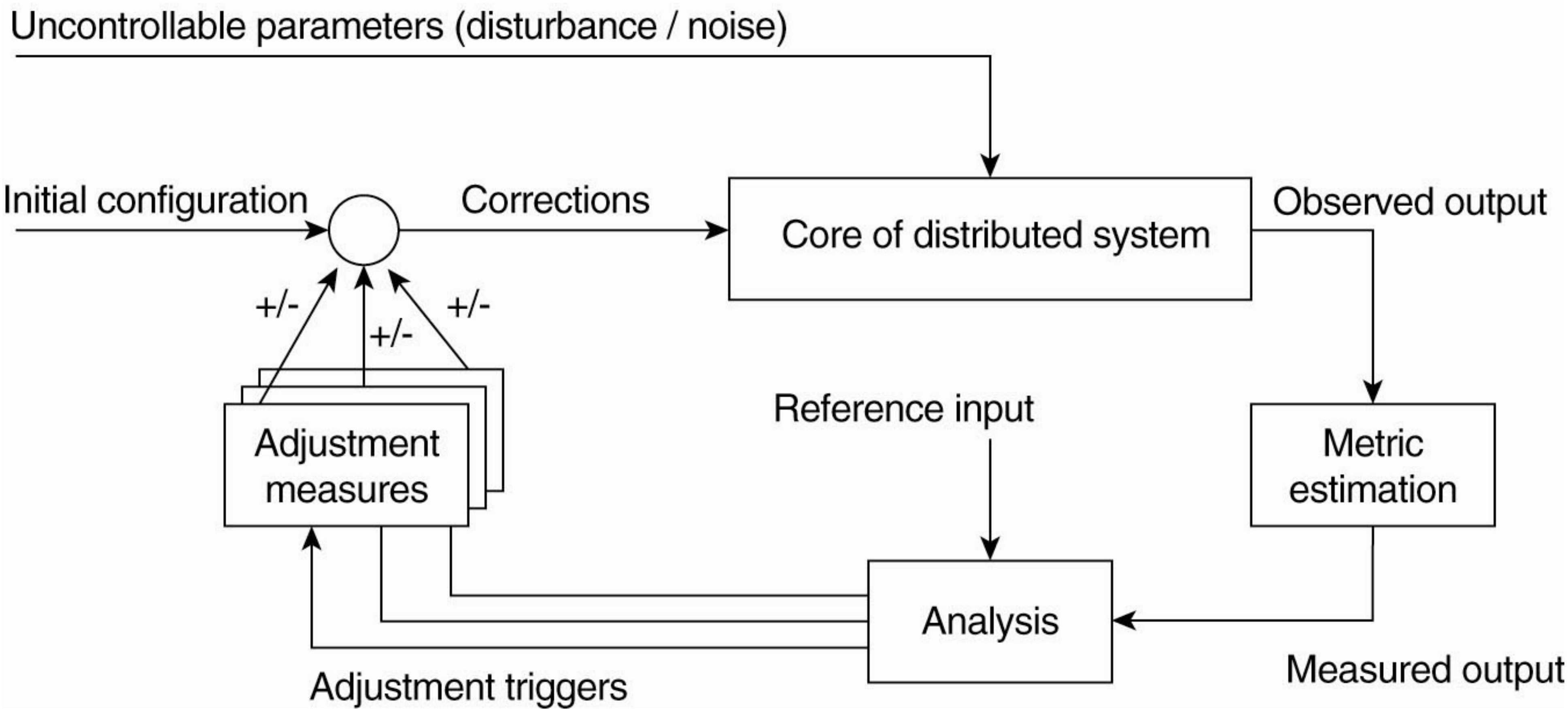


Figure 2-16. The logical organisation of a feedback control system.

Example: Automatic Component Repair Management in Jade

Steps required in a repair procedure:

- Terminate every binding between a component on a nonfaulty 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.

Example: Systems Monitoring with Astrolabe

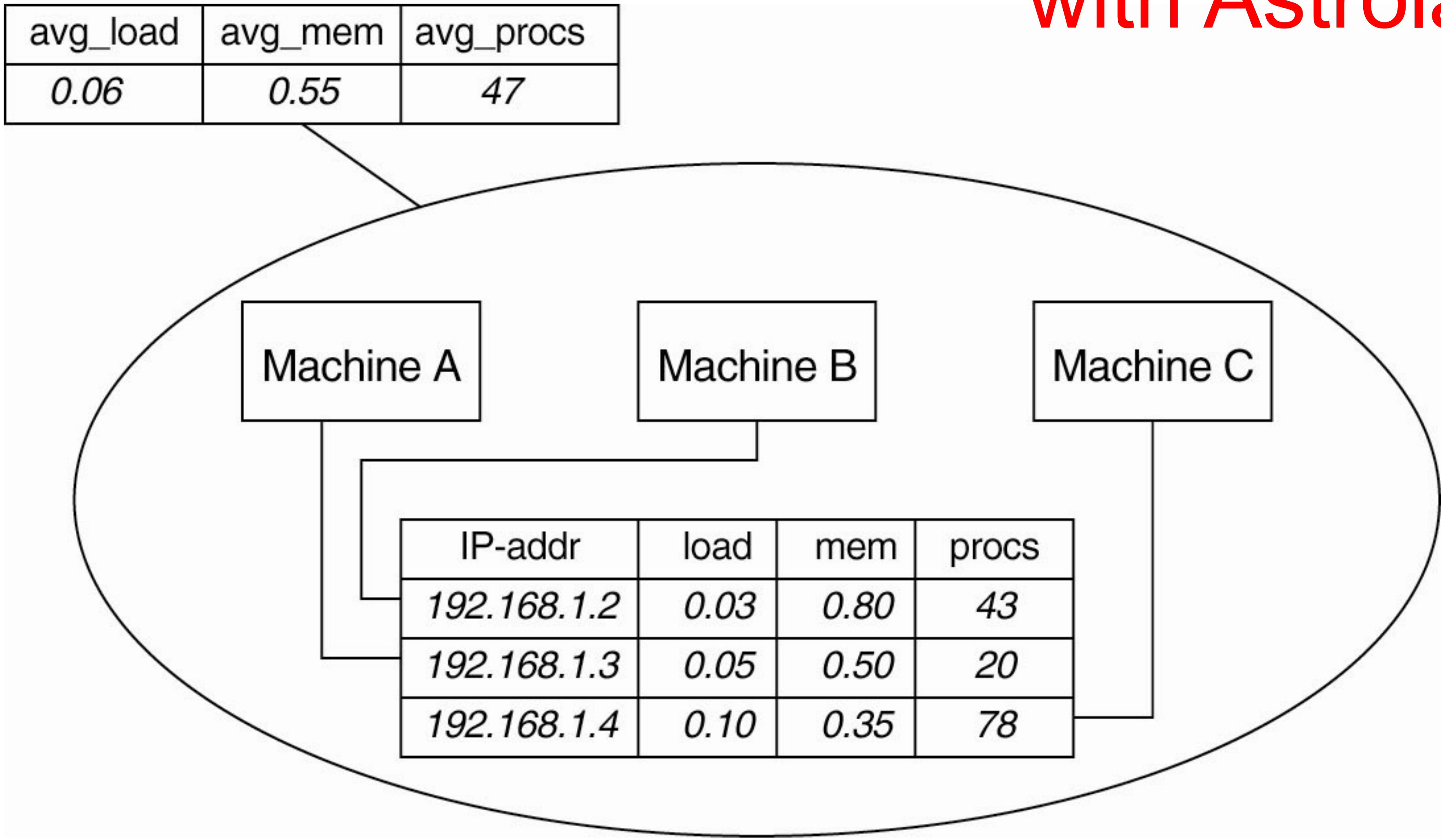


Figure 2-17. Data collection and information aggregation in Astrolabe.