DISTRIBUTED SYSTEMS
Principles and Paradigms

Chapter 2
ARCHITECTURES

Contents

- ✓ Architectural Styles
- ✓ System Architectures
- ✓ Architectures versus Middleware
- ✓ Self-management in Distributed Systems

Architectural Styles (1)

Important styles of architecture for distributed systems

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

Architectural Styles (2)

Layered architectures

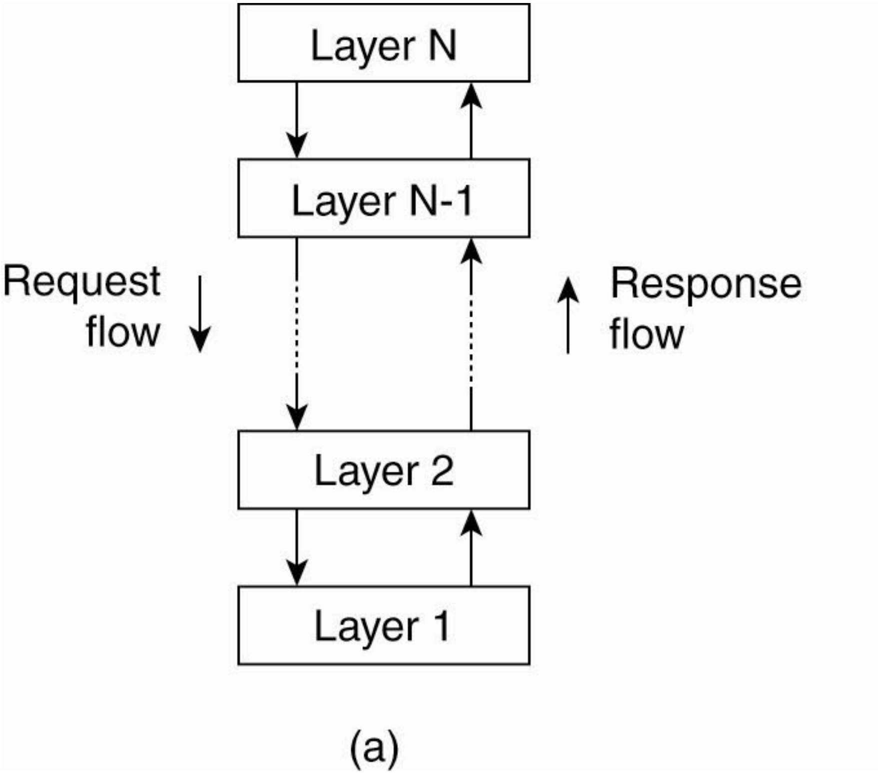

Figure 2-1. The (a) layered architectural style and …

Basic idea:

- Simple-Components are organized in a layered fashion where a component at layer L; is allowed to call components at he underlying layer Li.
- But not the other way around.
- This model has been widely adopted by the networking community.
- An key observation is that control generally flows from layer to layer: requests go down the hierarchy whereas the results flow upward.

Architectural Styles (3)

Object-based architectures

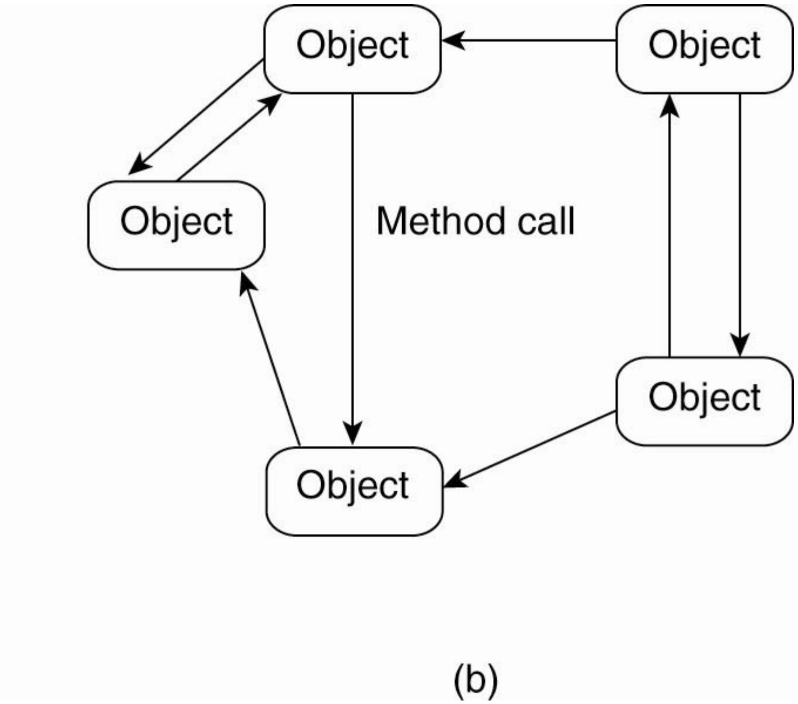

Figure 2-1. (b) The object-based architectural style.

- In essence, each object corresponds to what we have defined as a component, and these components are connected through a (remote) procedure call mechanism.

Architectural Styles (4)

Event-based architectures

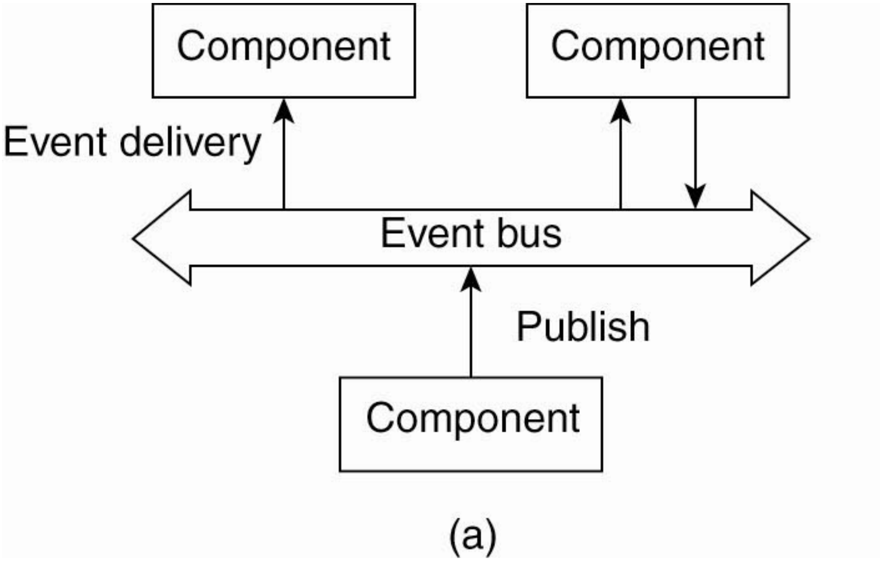

Figure 2-2. (a) The event-based architectural style and …

- Processes essentially communicate through the propagation of events, which optionally also carry data.
- For distributed systems, event propagation has generally been associated with what are known as publish/subscribe systems.
- **Basic idea**: Processes publish events after which the middleware ensures that only those processes that subscribed to those events will receive them.
- **Advantage**: Event-based systems is that processes are loosely coupled.

Architectural Styles (5)

Shared data-space architecture

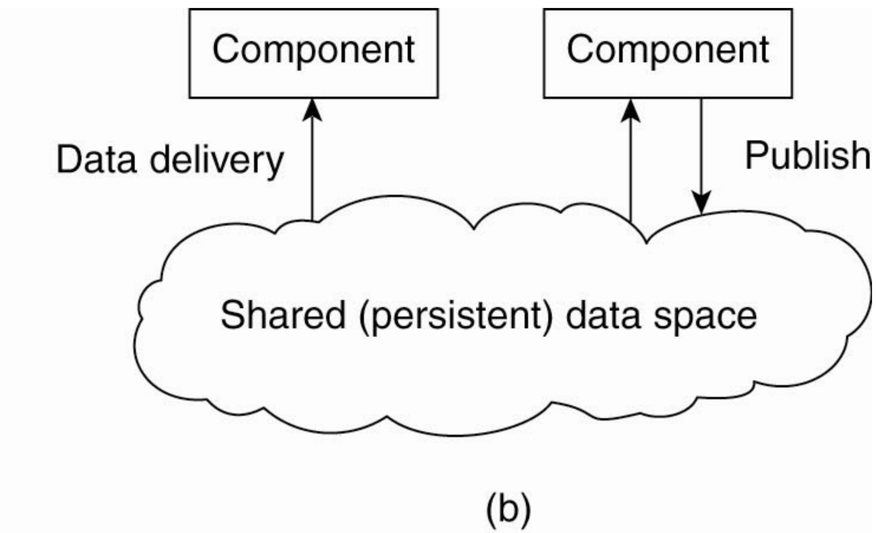

Figure 2-2. (b) The shared data-space architectural style.

➔**Data-centered architectures** evolve around the idea that processes communicate through a common (passive or active) repository.

➔**Event-based architectures** can be combined with **data-centered architectures**,yielding what is also known as shared data spaces.

➔The essence of **shared data spaces** is that **processes are now also decoupled in time**: they need not both be active when communication takes place.

➔Furthermore, many shared data spaces use a **SQL-like interface** to the shared repository in that sense that data can be accessed using a description rather than an explicit reference, as is the case with files.

System Architectures

- Centralized Architectures
- Decentralized Architectures
- Hybrid Architectures

Centralized Architectures

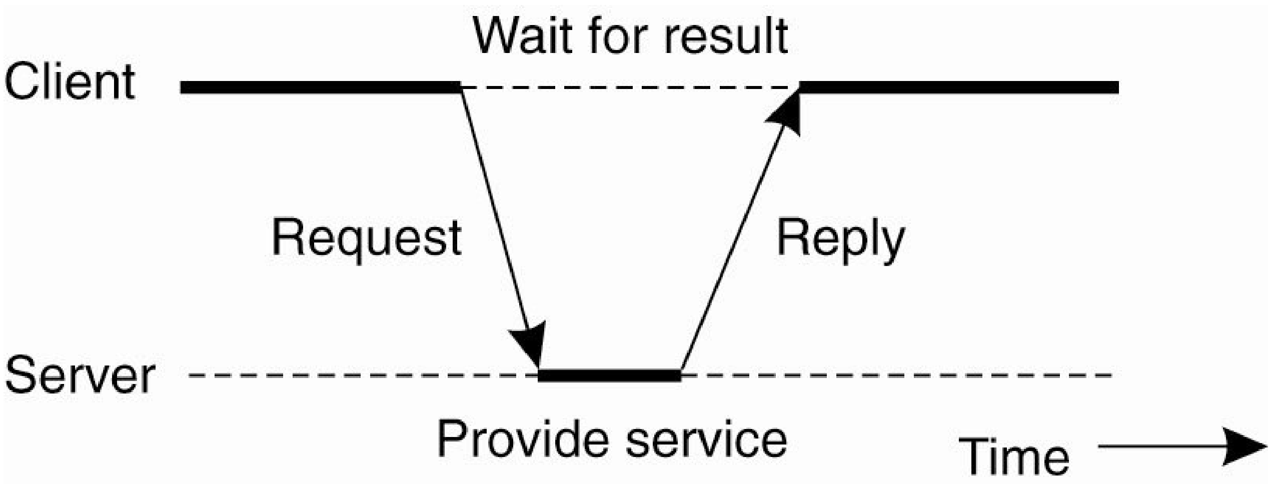

Figure 2-3. General interaction between a client and a server.

- 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.

Centralized Architectures(contd..)

Application Layering (1)

Recall previously mentioned layers of architectural style

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

Centralized Architectures(contd..)

Application Layering (2)

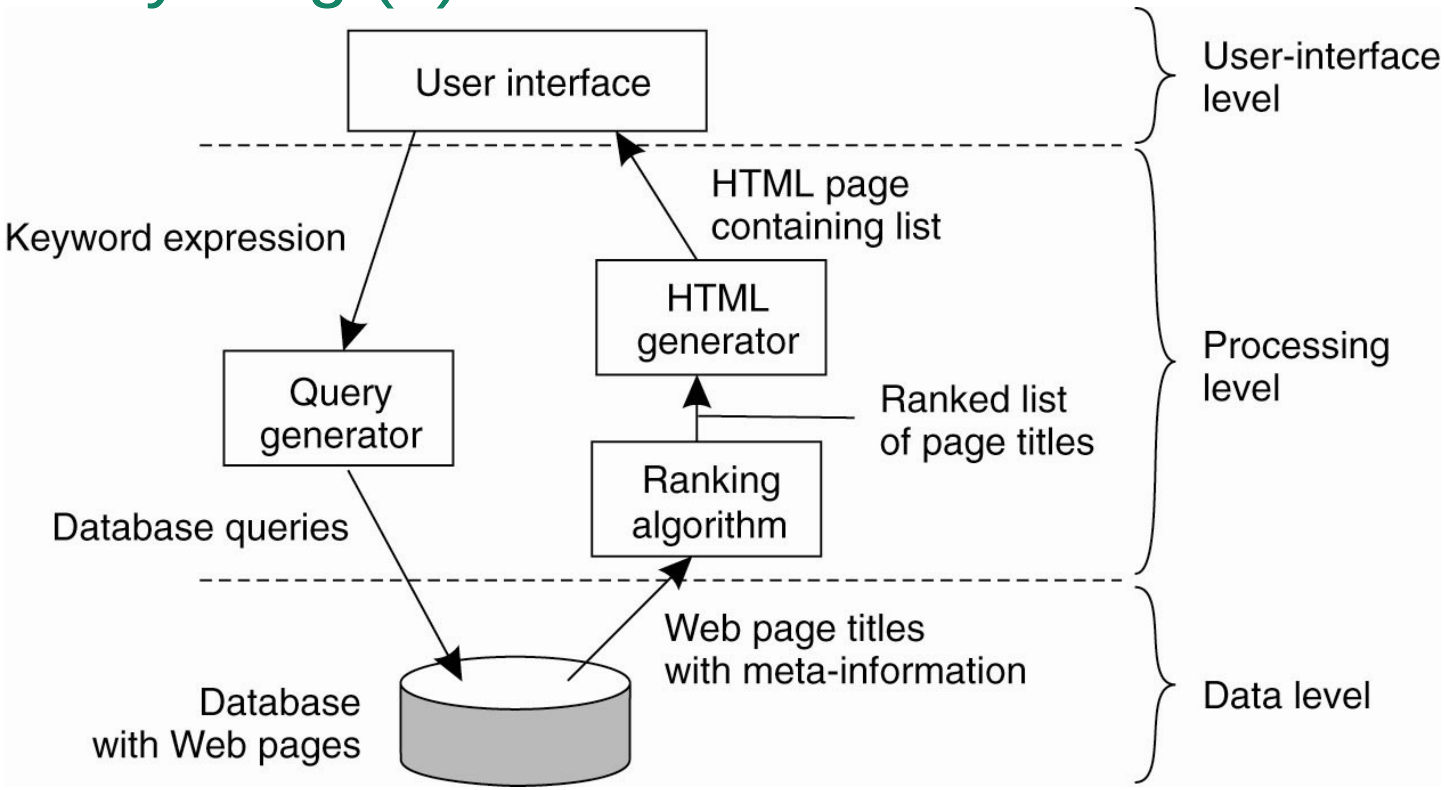

Figure 2-4. The simplified organization of an Internet search engine into three different layers.

- 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.

Centralized Architectures(contd..)

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

Centralized Architectures(contd..)

Multitiered Architectures (2)

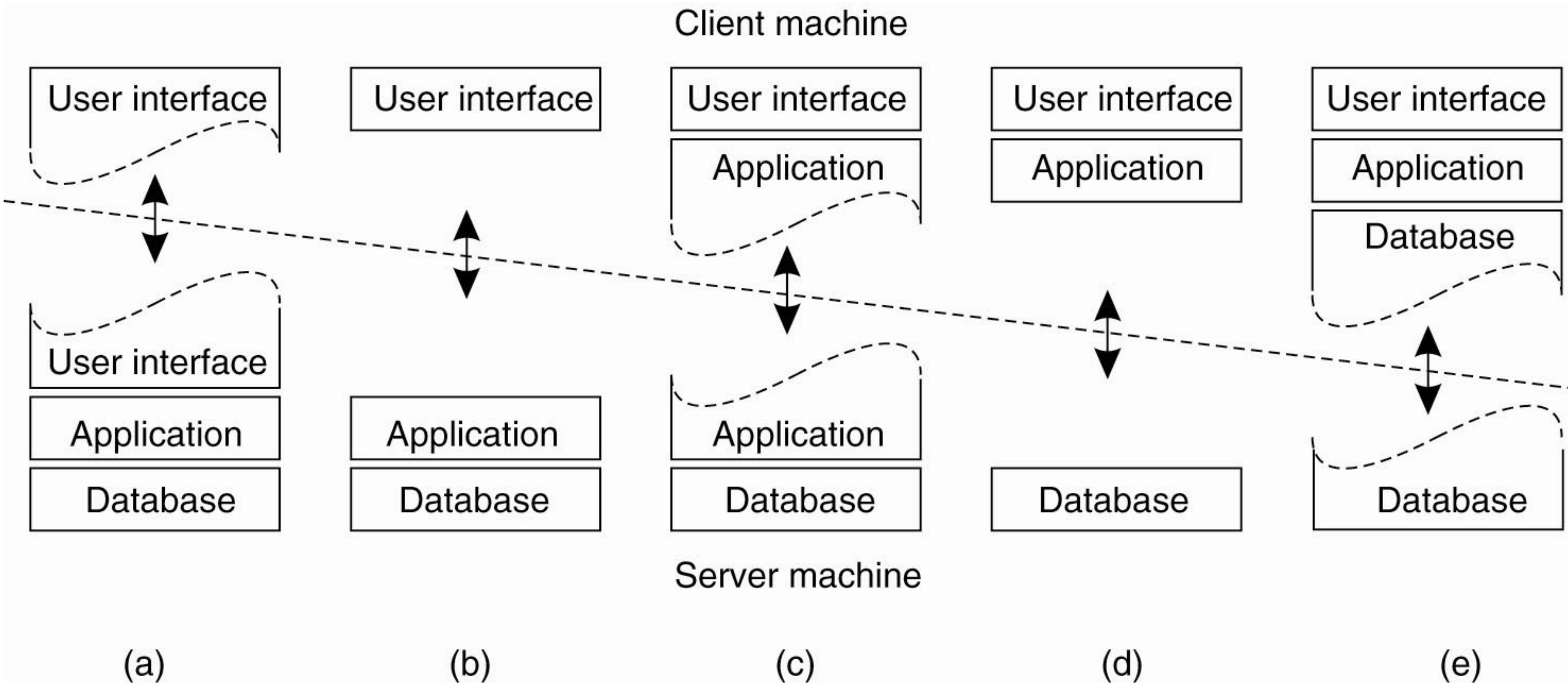

Figure 2-5. Alternative client-server organizations (a)–(e).

Centralized Architectures(contd..)

Multitiered Architectures (3)

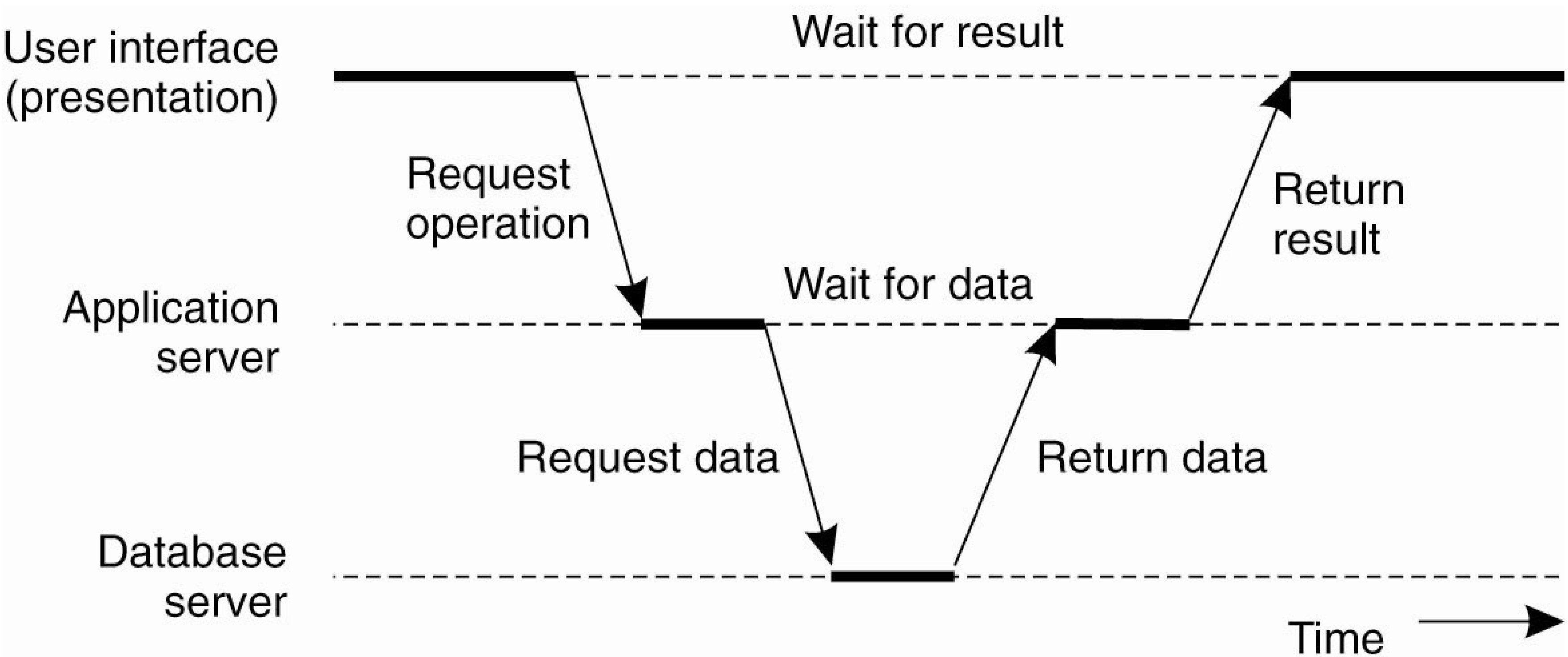

Figure 2-6. An example of a server acting as client.

Decentralized Architectures

Structured Peer-to-Peer Architectures (1)

- In a structured peer-to-peer architecture, the overlay network is constructed using a deterministic procedure.
- By far the most-used procedure is to organize the processes through a distributed hash table (DHT).
- In a DHT -based system, data items are assigned a random key from a large identifier space, such as a 128-bit or 160-bit identifier.
- Likewise, nodes in the system are also assigned a random number from the same identifier space.
- The crux of every DHT-based system is then to implement an efficient and deterministic scheme that uniquely maps the key of a data item to the identifier of a node based on some distance metric.
- 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.

Decentralized Architectures

Structured Peer-to-Peer Architectures (2)

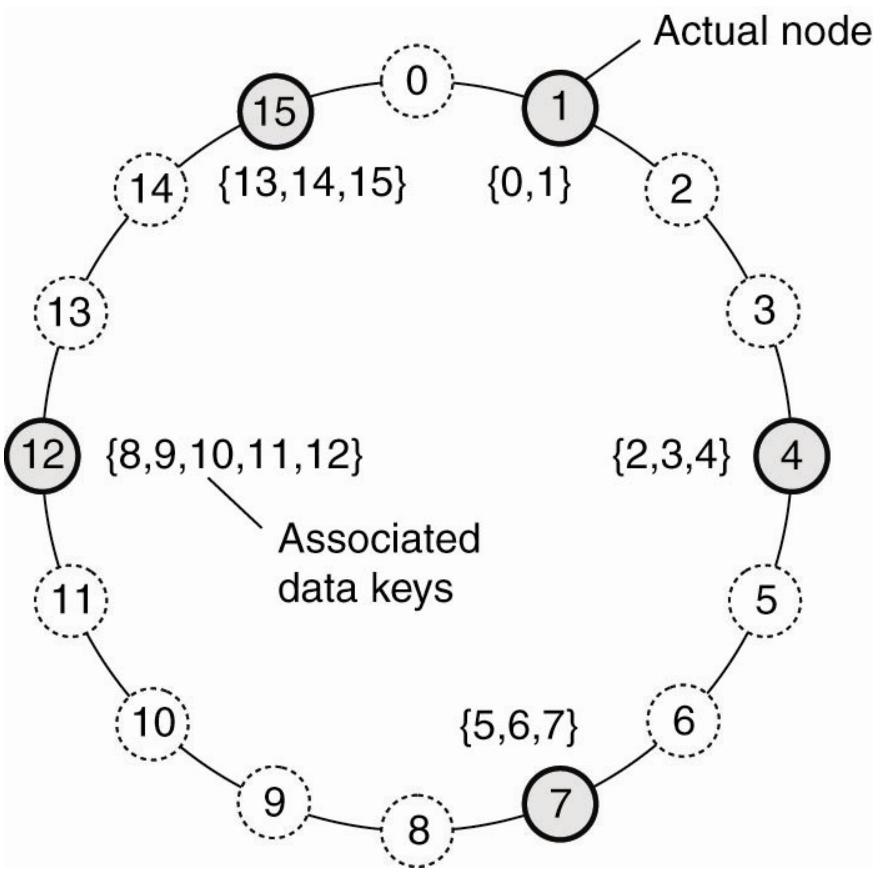

Figure 2-7. The mapping of data items onto nodes in Chord.

For example, in the Chord system the nodes are logically organized in a ring such that a data item with key k is mapped to the node with the smallest identifier id ~ k.

This node is referred to as the successor of key k and denoted as succ(k), as shown in Fig. 2-7.

To actually look up the data item, an application running on an arbitrary node would then call the function LOOKUP(k)which would subsequently return the network address of succ(k).

At that point, the application can contact the node to obtain a copy of the data item.

Decentralized Architectures

Structured Peer-to-Peer Architectures (3)

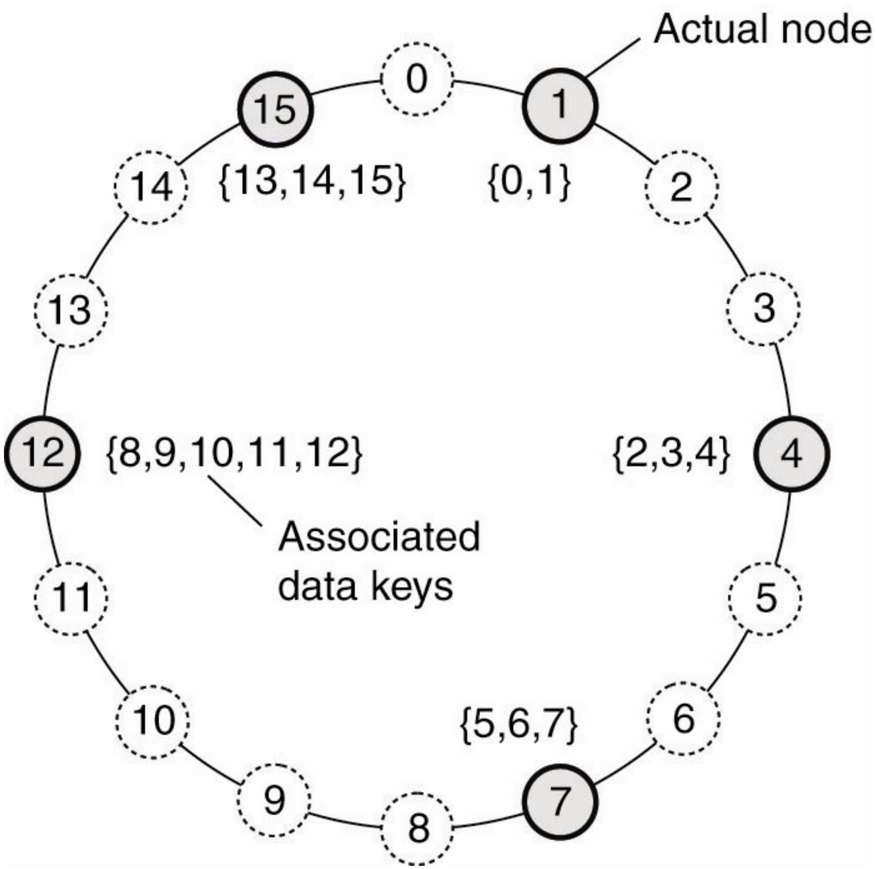

Figure 2-7. The mapping of data items onto nodes in Chord.

- When a node wants to join the system, it starts with generating a random identifier id.
- Then, the node can simply do a lookup on id, which will return the network address of succiid).
- At that point, the joining node can simply contact succ(id) and its predecessor and insert itself in the ring.
- Of course, this scheme requires that each node also stores information on its predecessor.
- Insertion also yields that each data item whose key is now associated with node id, is transferred from succ(id).
- Leaving is just as simple: node id informs its departure to its predecessor and successor, and transfers its data items to succ(id).

Decentralized Architectures(contd..)

Structured Peer-to-Peer Architectures (4)

- **Content Addressable Network (CAN)**, deploys a d-dimensional cartesian coordinate space, which is completely partitioned among all the nodes that participate in the system.

- Fig.2-8(a) shows how the two-dimensional space [0, l]x[O, 1] is divided among six nodes.
- Each node has an associated region.
- Every data item in CAN will be assigned a unique point in this space, after which it is also clear which node is responsible for that data.
- When a node P wants to join a CAN system, it picks an arbitrary point from the coordinate space and subsequently looks up the node Q in whose region that point falls.
- This lookup is accomplished through positioned-based routing.

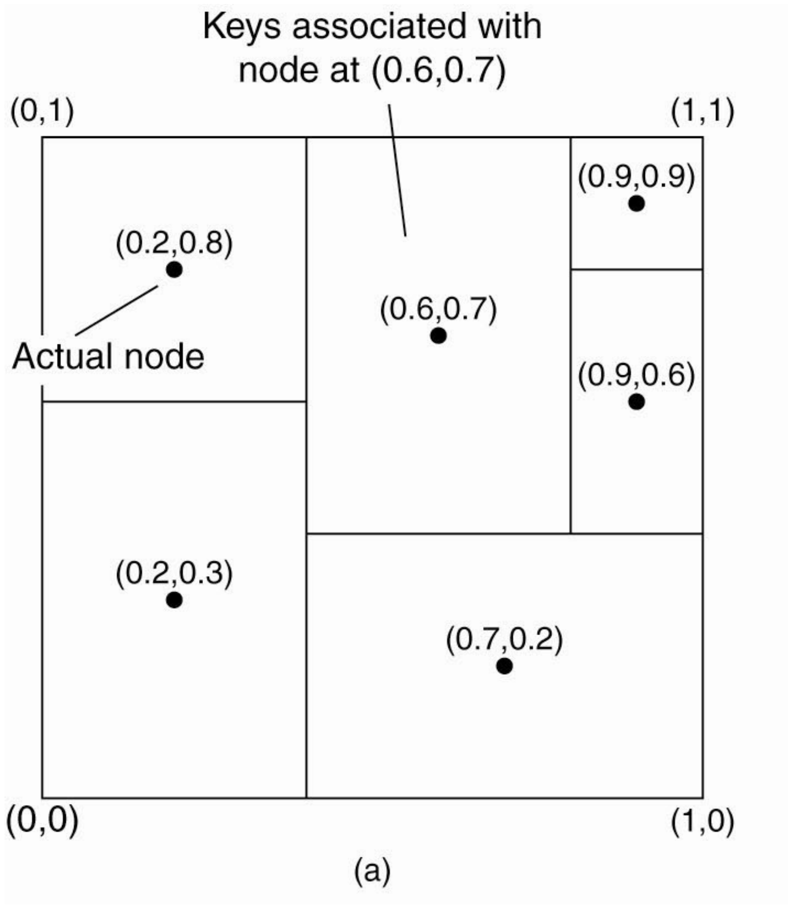

Figure 2-8. (a) The mapping of data items onto nodes in CAN.

Decentralized Architectures(contd..)

Structured Peer-to-Peer Architectures (5)

- Node Q then splits its region into two halves, as shown in Fig. 2-8(b). and one half is assigned to the node P.
- Nodes keep track of their neighbors, that is, nodes responsible for adjacent region.
- When splitting a region, the joining node P can easily come to know who its new neighbors are by asking node P.
- Assume that in Fig. 2-8. the node with coordinate (0.6, 0.7) leaves.
- Its region will be assigned to one of its neighbors, say the node at (0.9,0.9), but it is clear that simply merging it and obtaining a rectangle cannot be done.
- In this case, the node at (0.9,0.9) will simply take care of that region and inform the old neighbors of this fact.

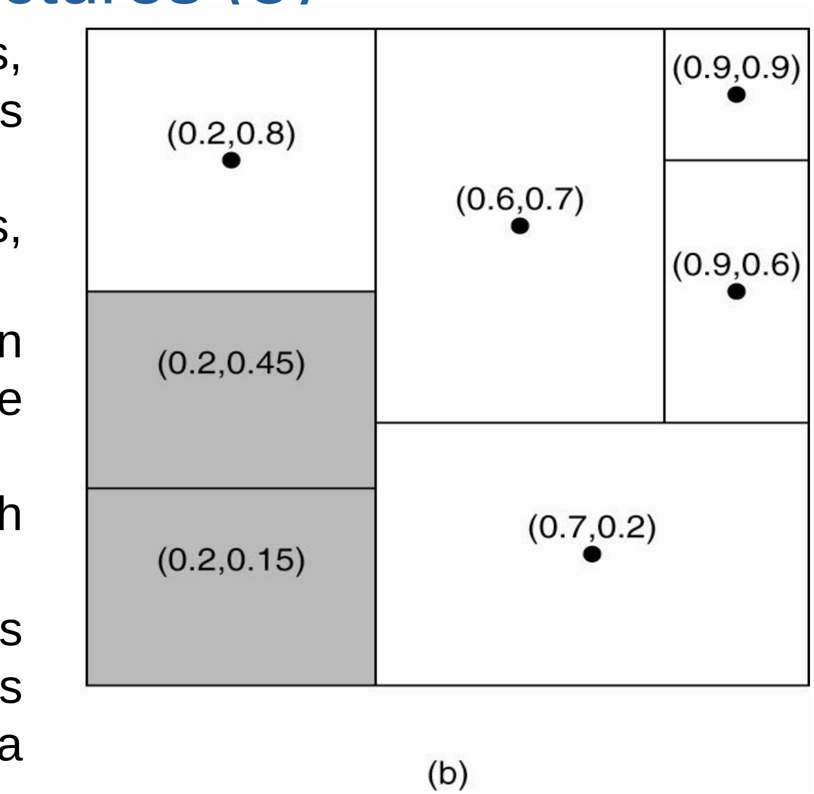

Figure 2-8. (b) Splitting a region when a node joins.

- Obviously. this may lead to less symmetric partitioning of the coordinate space, for which reason a background process is periodically started to repartition the entire space.

Decentralized Architectures(contd..)

Unstructured Peer-to-Peer Architectures (1)

- 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.
- Goal: Systems is to construct an overlay network that resembles a random graph.
- Basic model: Each node maintains a list of c neighbors, where, ideally, each of these neighbors represents a randomly chosen live node from the current set of nodes.
- The list of neighbors is also referred to as a partial view.
- There are many ways to construct such a partial view.
- In this framework, it is assumed that nodes regularly exchange entries from their partial view.
- Each entry identifies another node in the network, and has an associated age that indicates how old the reference to that node is.
- Two threads are used, as shown in Fig. 2-9.

Decentralized Architectures(contd..)

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.

Decentralized Architectures(contd..)

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

Decentralized Architectures(contd..)

Topology Management of Overlay Networks (1)

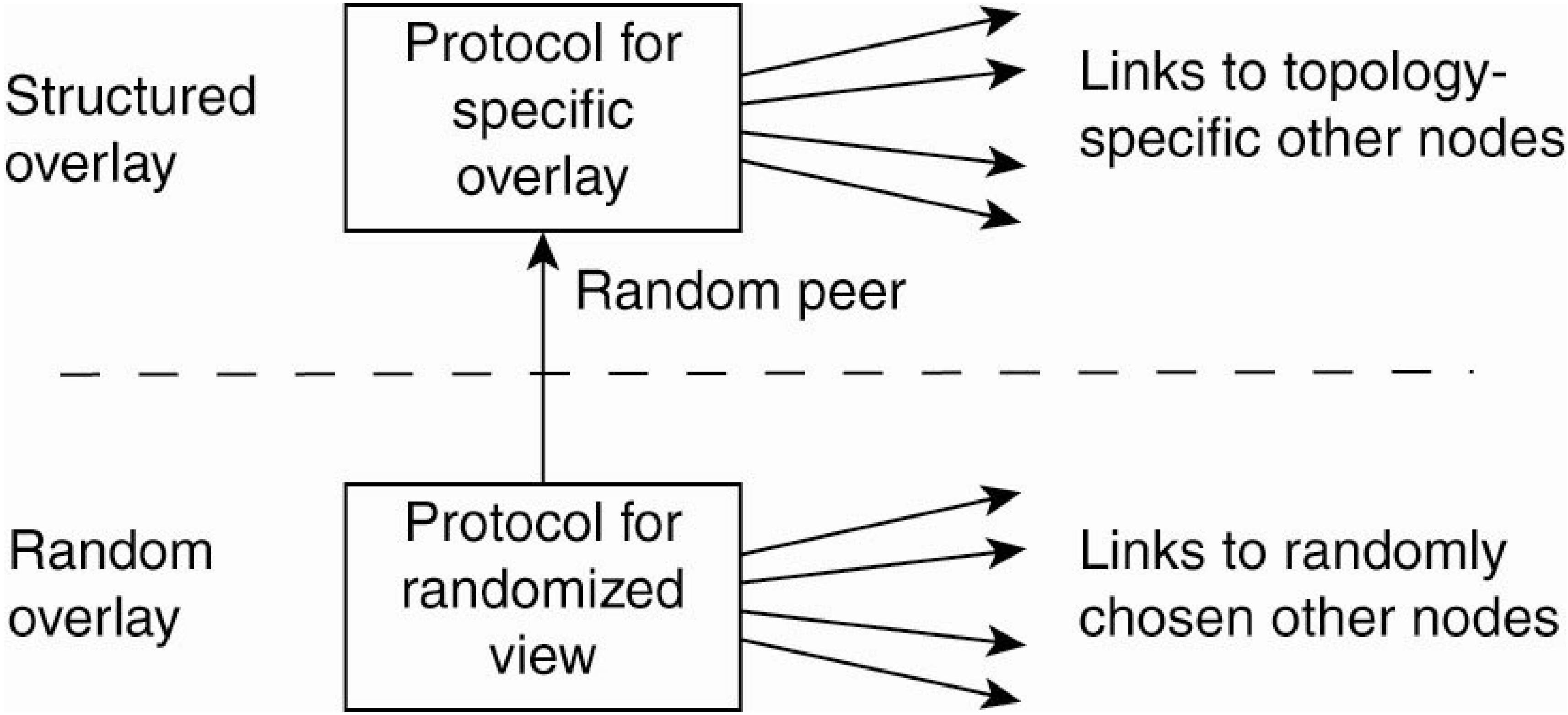

Figure 2-10. A two-layered approach for constructing and maintaining specific overlay topologies using techniques from unstructured peer-to-peer systems.

Decentralized Architectures(contd..)

Topology Management of Overlay Networks (2)

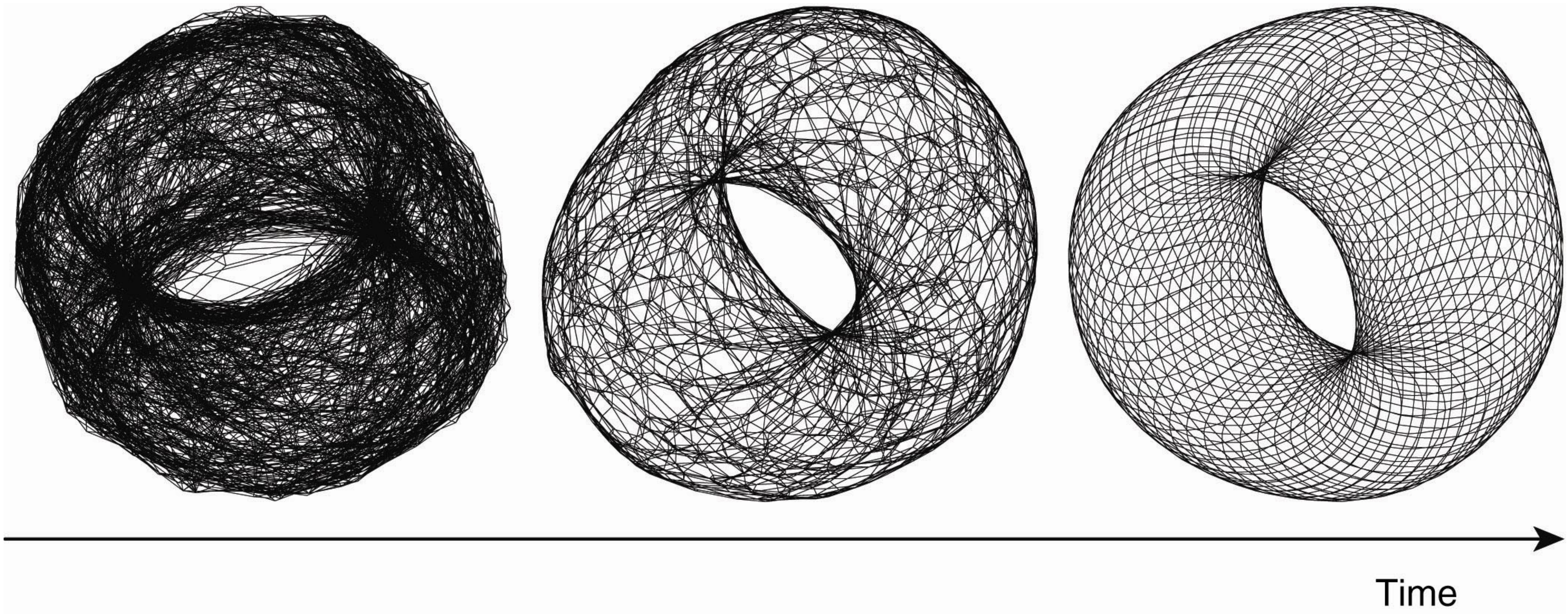

Figure 2-11. Generating a specific overlay network using a two-layered unstructured peer-to-peer system [adapted with permission from Jelasity and Babaoglu (2005)].

Decentralized Architectures(contd..)

Superpeers

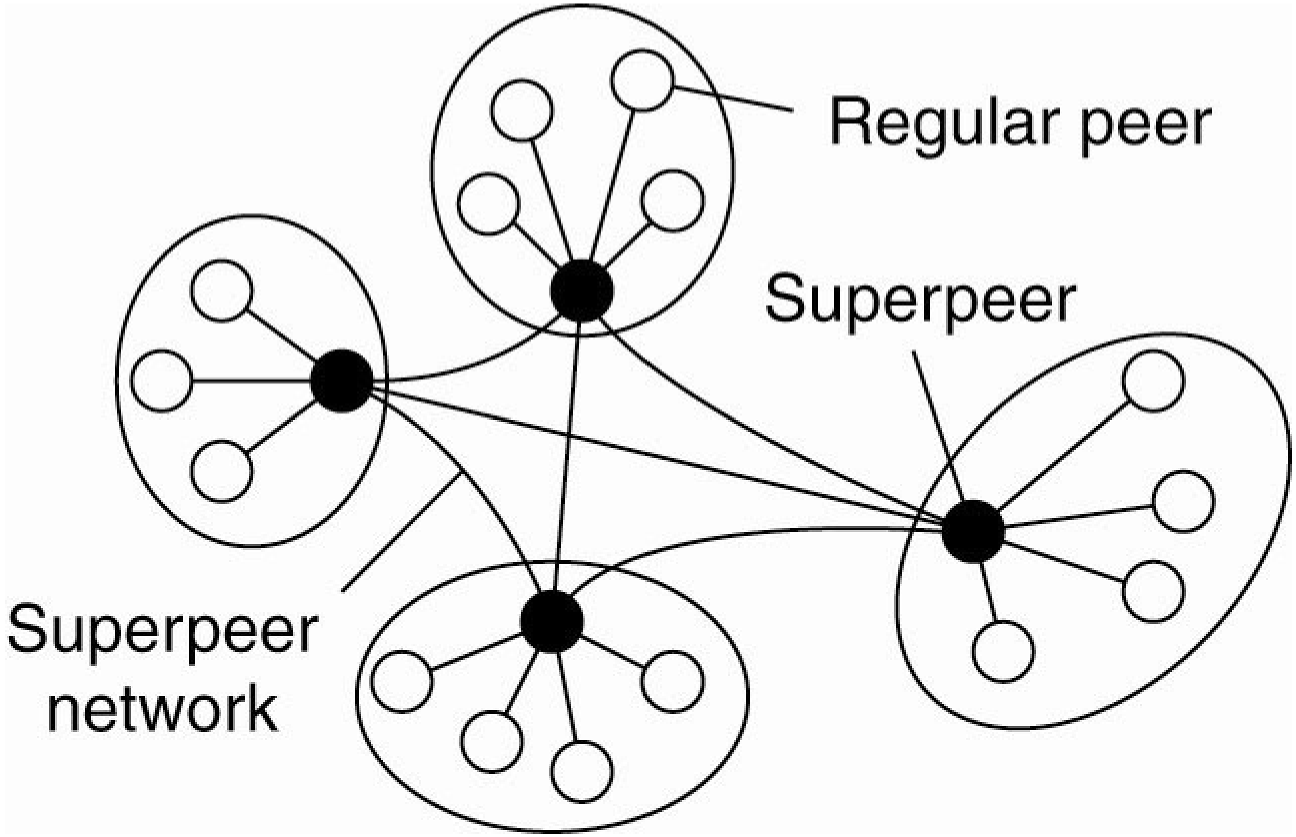

Figure 2-12. A hierarchical organization of nodes into a superpeer network.

Hybrid Architectures

Edge-Server Systems

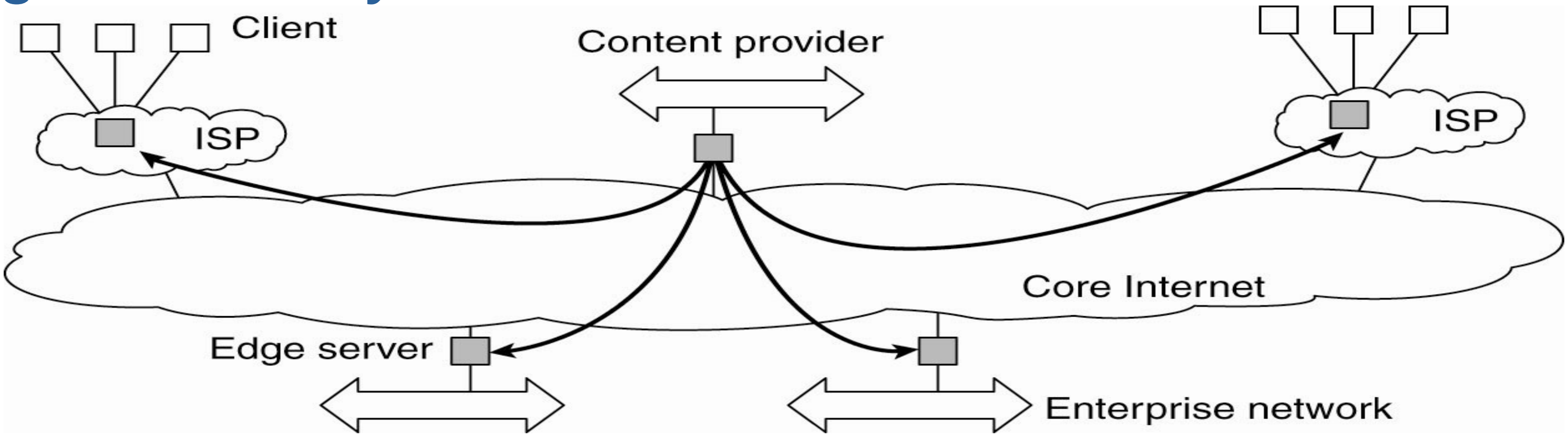

Figure 2-13. Viewing the Internet as consisting of a collection of edge servers.

- 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.
- This edge is formed by the boundary between enterprise networks and the actual Internet, for example, as provided by an Internet Service Provider (ISP).
- Likewise, where end users at home connect to the Internet through their ISP, the ISP can be considered as residing at the edge of the Internet.
- The edge server's main purpose is to serve content, possibly after applying filtering and transcoding functions.

Hybrid Architectures(contd..)

Collaborative Distributed Systems (1)

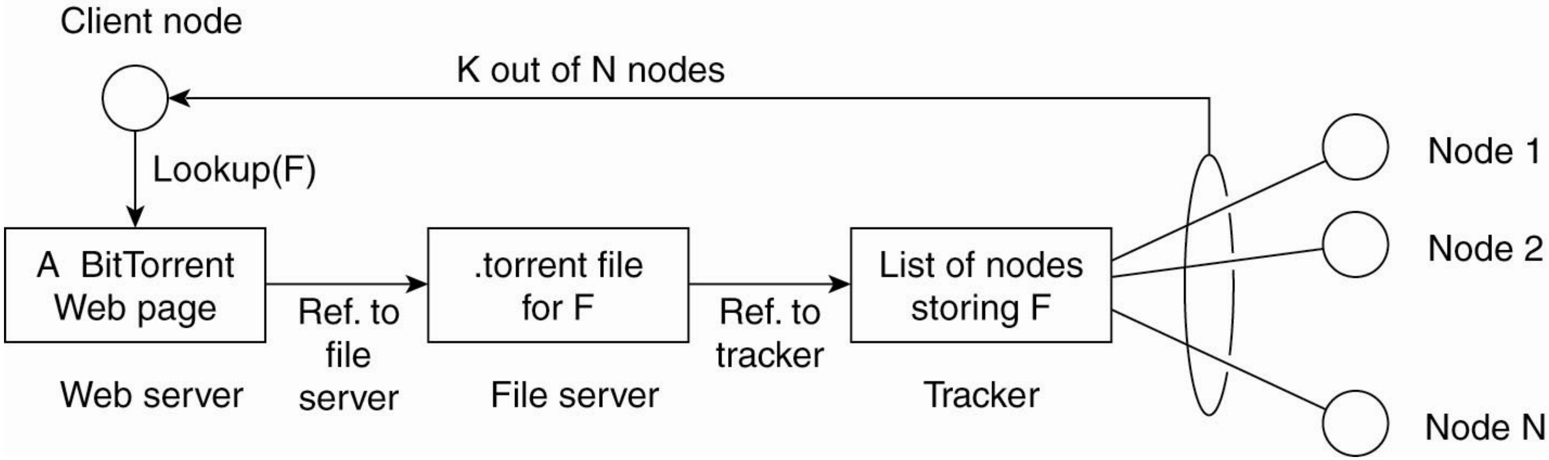

Figure 2-14. The principal working of BitTorrent [adapted with permission from Pouwelse et al. (2004)].

- Let us first consider the BitTorrent file-sharing system (Cohen, 2003).
- BitTorrent is a peer-to-peer file downloading system.Its principal working is shown in Fig. 2-14.
- The basic idea is that when an end user is looking for a file, he downloads chunks of the file from other users until the downloaded chunks can be assembled together yielding the complete file.
- An important design goal was to ensure collaboration. In most file-sharing systems, a significant fraction of participants merely download files but otherwise contribute close to nothing.
- To this end, a file can be downloaded only when the downloading client is providing content to someone else. We will return to this "tit-for-tat" behavior shortly.

Hybrid Architectures(contd..)

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.

ARCHITECTURES VERSUS MIDDLEWARE

Interceptors

- An **interceptor** is nothing but a software construct that will break the usual flow of control and allow other (application specific) code to be executed.
- Consider interception as supported in many object-based distributed systems. The basic idea is simple: an object A can call a method that belongs to an object B, while the latter resides on a different machine than A.
- A remote-object invocation is carried as a three-step approach:

1. Object A is offered a local interface that is exactly the same as the interface offered by object B. A simply calls the method available in that interface.

2. The call by A is transformed into a generic object invocation, made possible through a general object-invocation interface offered by the middleware at the machine where A resides.

3. Finally, the generic object invocation is transformed into a message that is sent through the transport-level network interface as offered by A's local operating system.

ARCHITECTURES VERSUS MIDDLEWARE

Interceptors

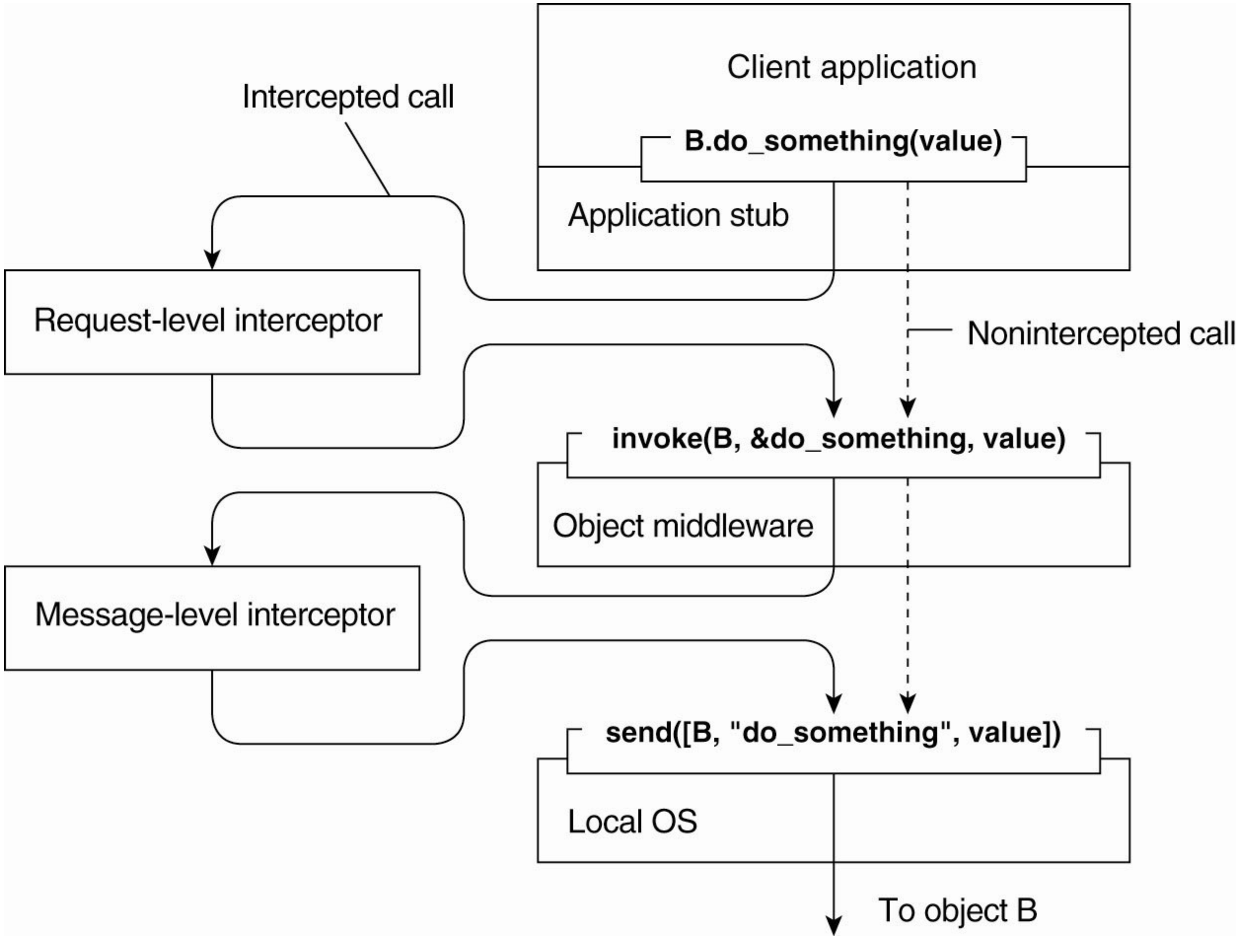

Figure 2-15. Using interceptors to handle remote-object invocations.

ARCHITECTURES VERSUS MIDDLEWARE

General Approaches to Adaptive Software

Three basic approaches to adaptive software:

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

SELF -MANAGEMENT IN DISTRIBUTED SYSTEMS

The Feedback Control Model

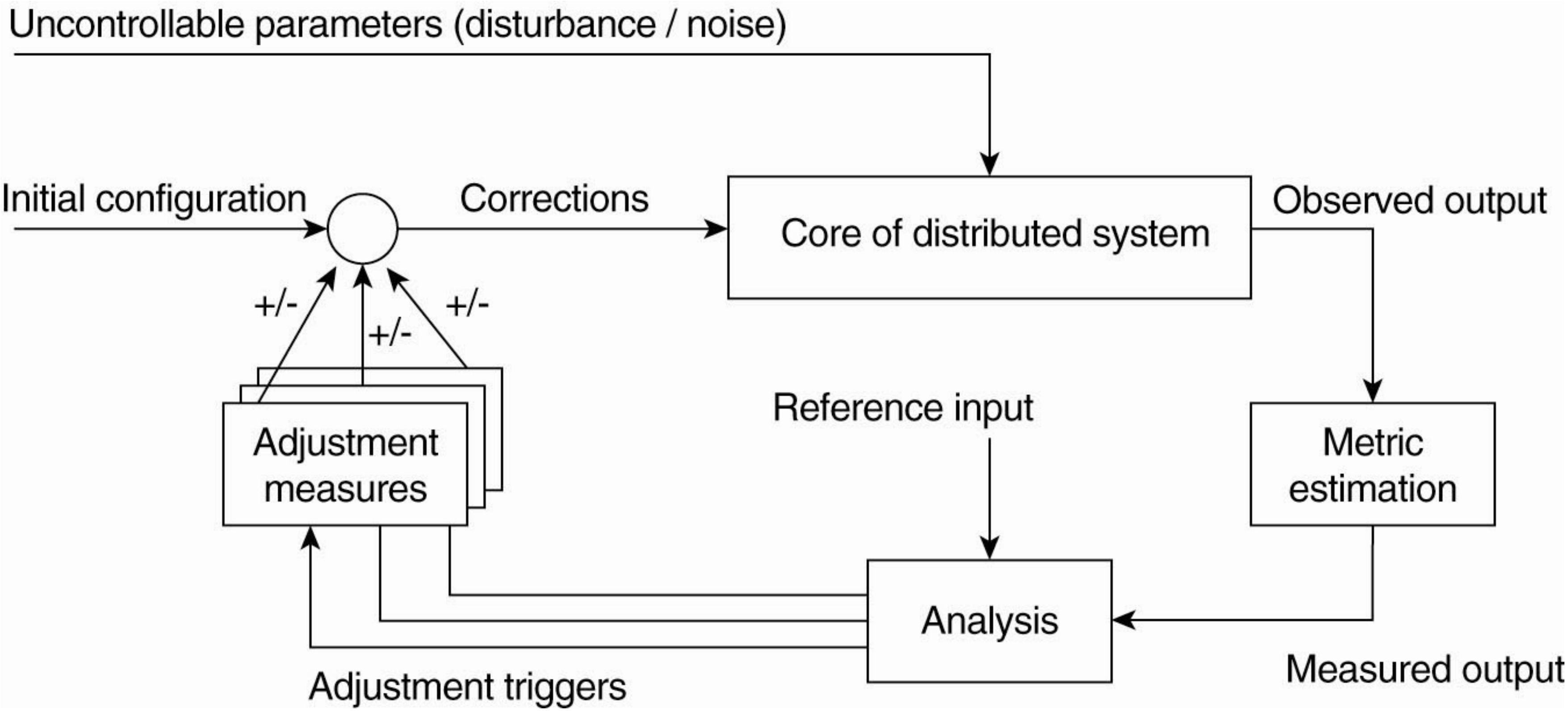

Figure 2-16. The logical organization of a feedback control system.

Example: Systems Monitoring with Astrolabe

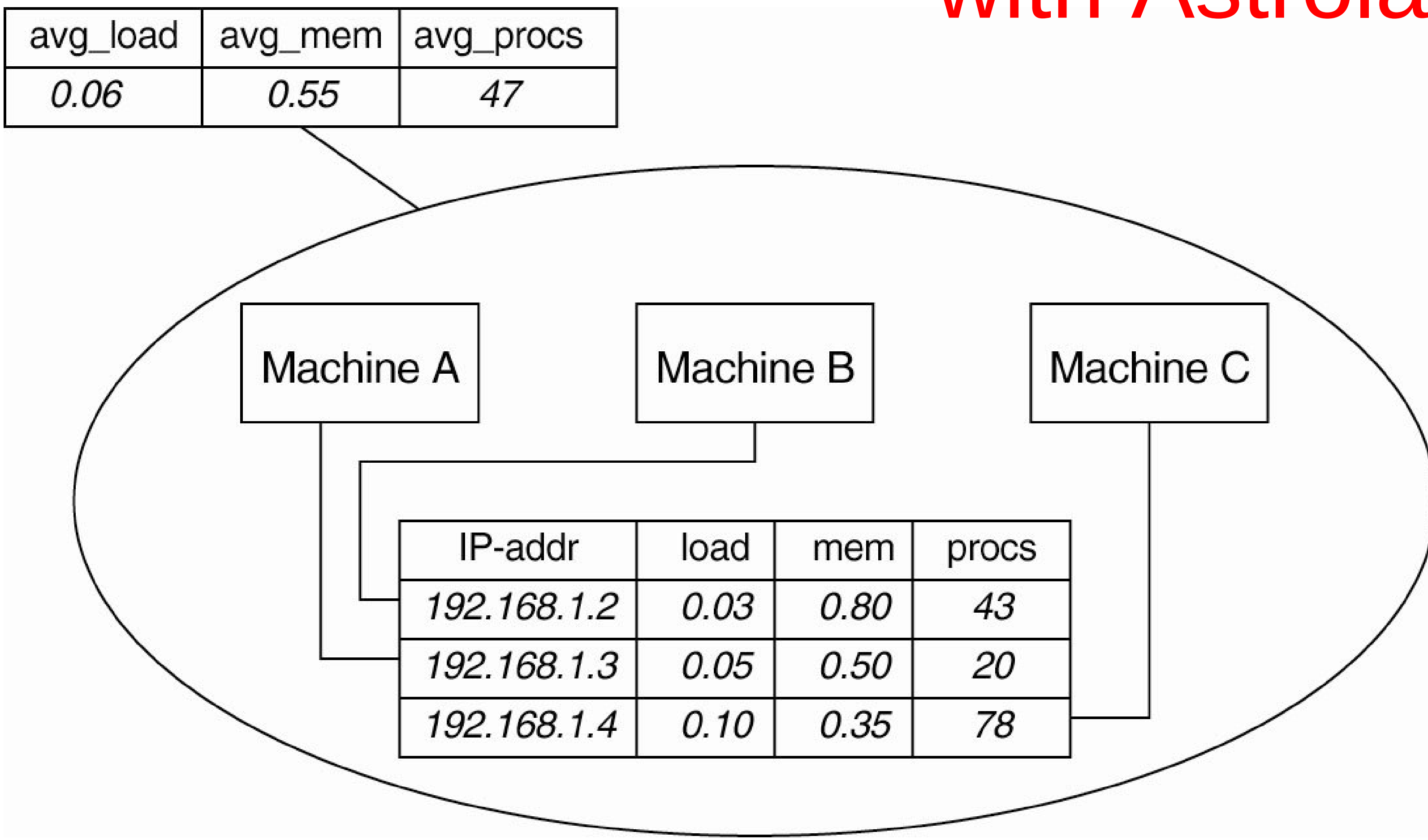

Figure 2-17. Data collection and information aggregation in Astrolabe.

Example: Differentiating Replication Strategies in Globule (1)

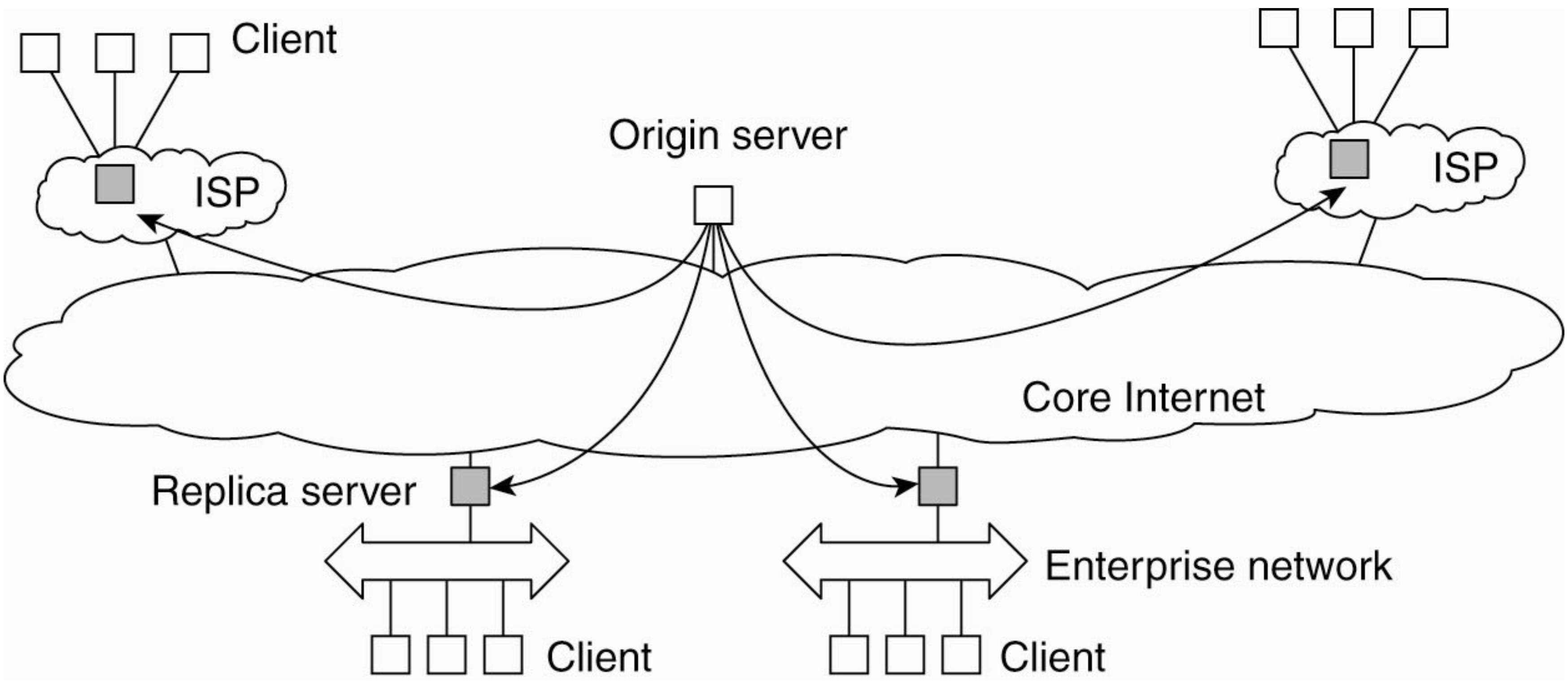

Figure 2-18. The edge-server model assumed by Globule.

Example: Differentiating Replication Strategies in Globule (2)

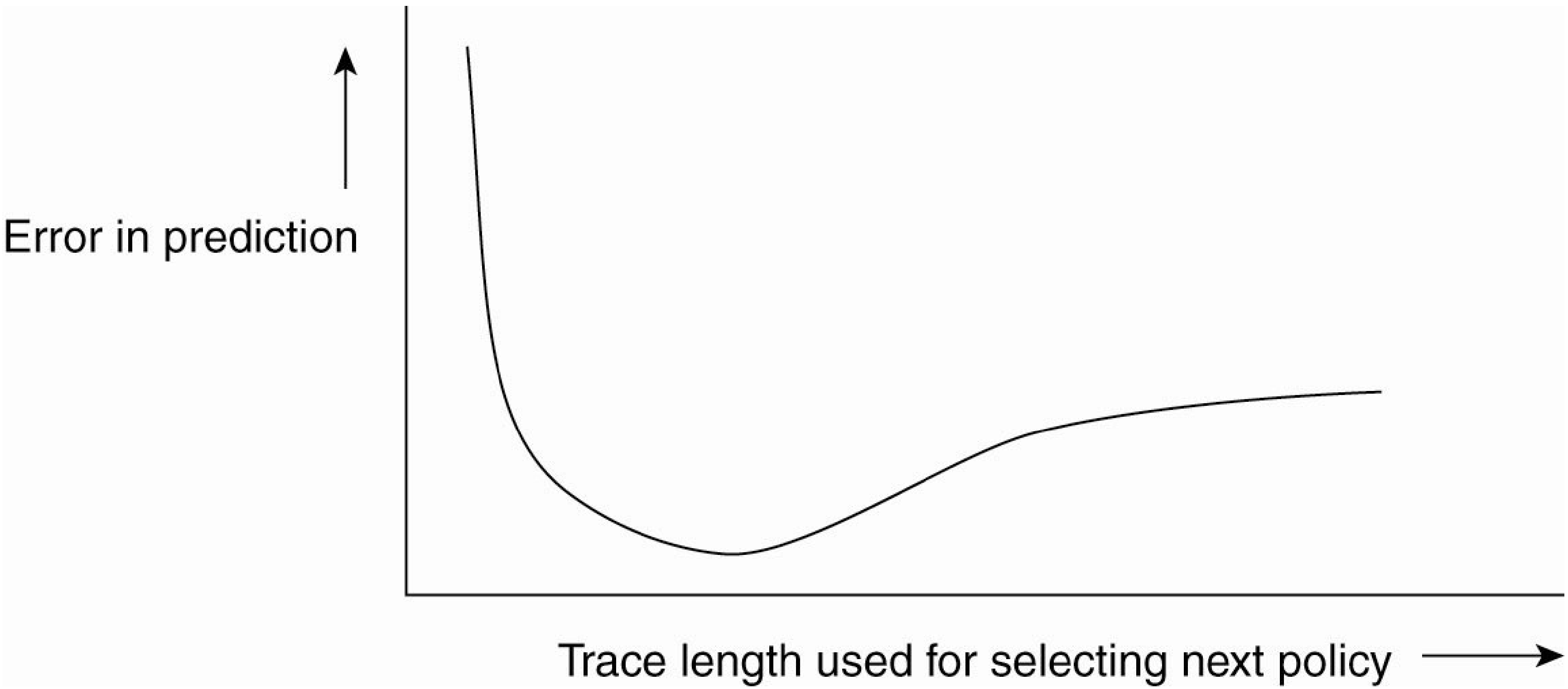

Figure 2-19. The dependency between prediction accuracy and trace length.

Example: Automatic Component Repair Management in Jade

Steps required in a repair procedure:

- Terminate every binding between a component on a 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.