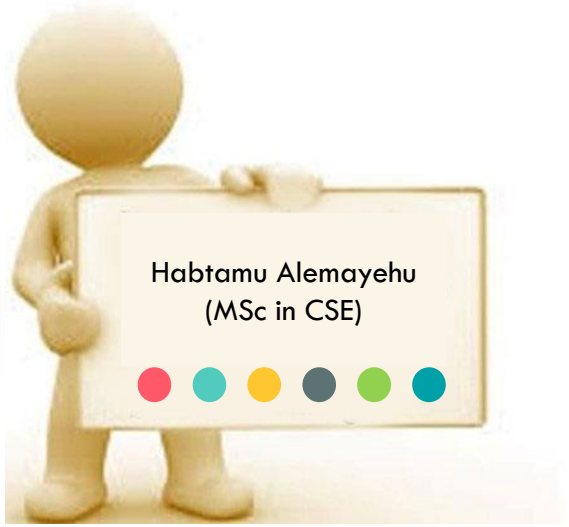
CHAPTER TWO
ARCHITECTURES

(CS, CCI, WKU, Ethiopia, 2023)

Lecturer Name:



Outline

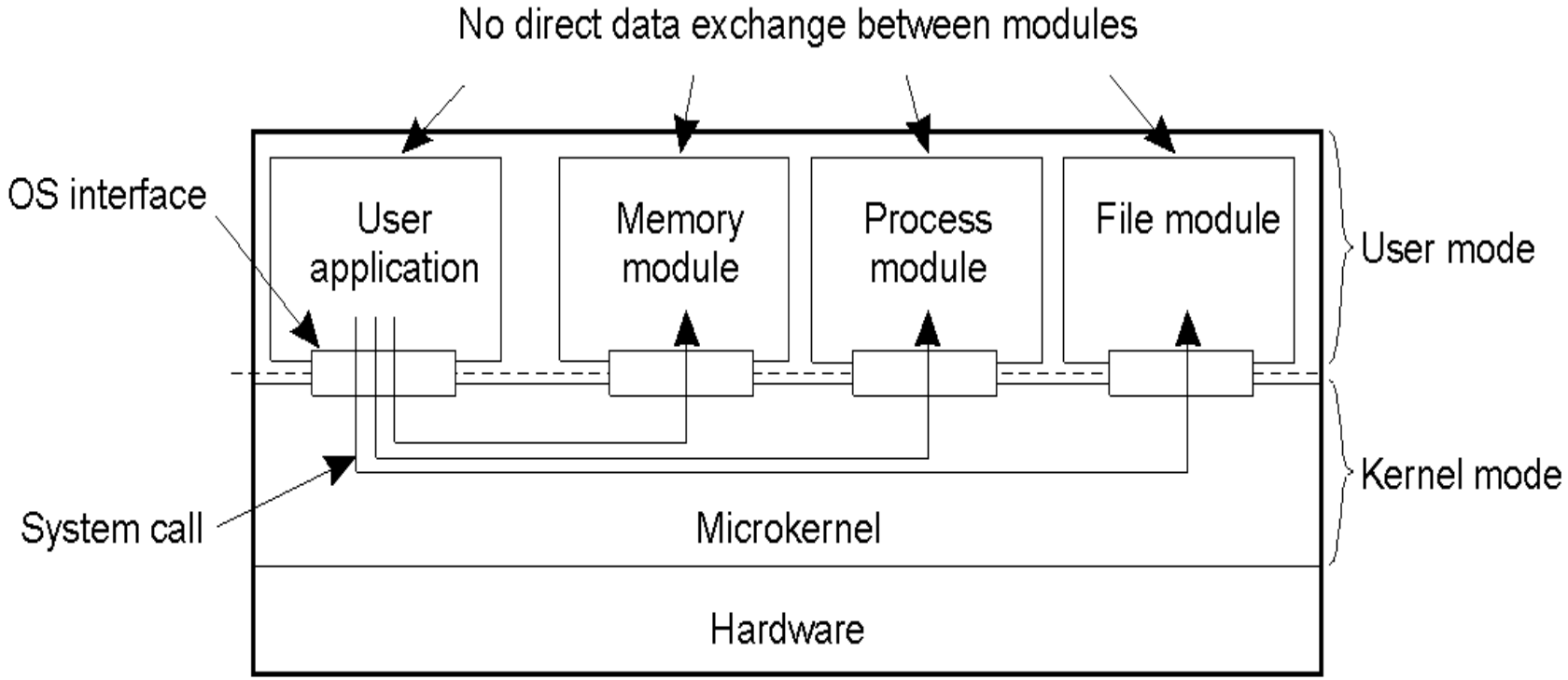
📖 In this chapter:

- Introduction
- Architectural Styles
- System Architectures

Modeling Distributed Systems

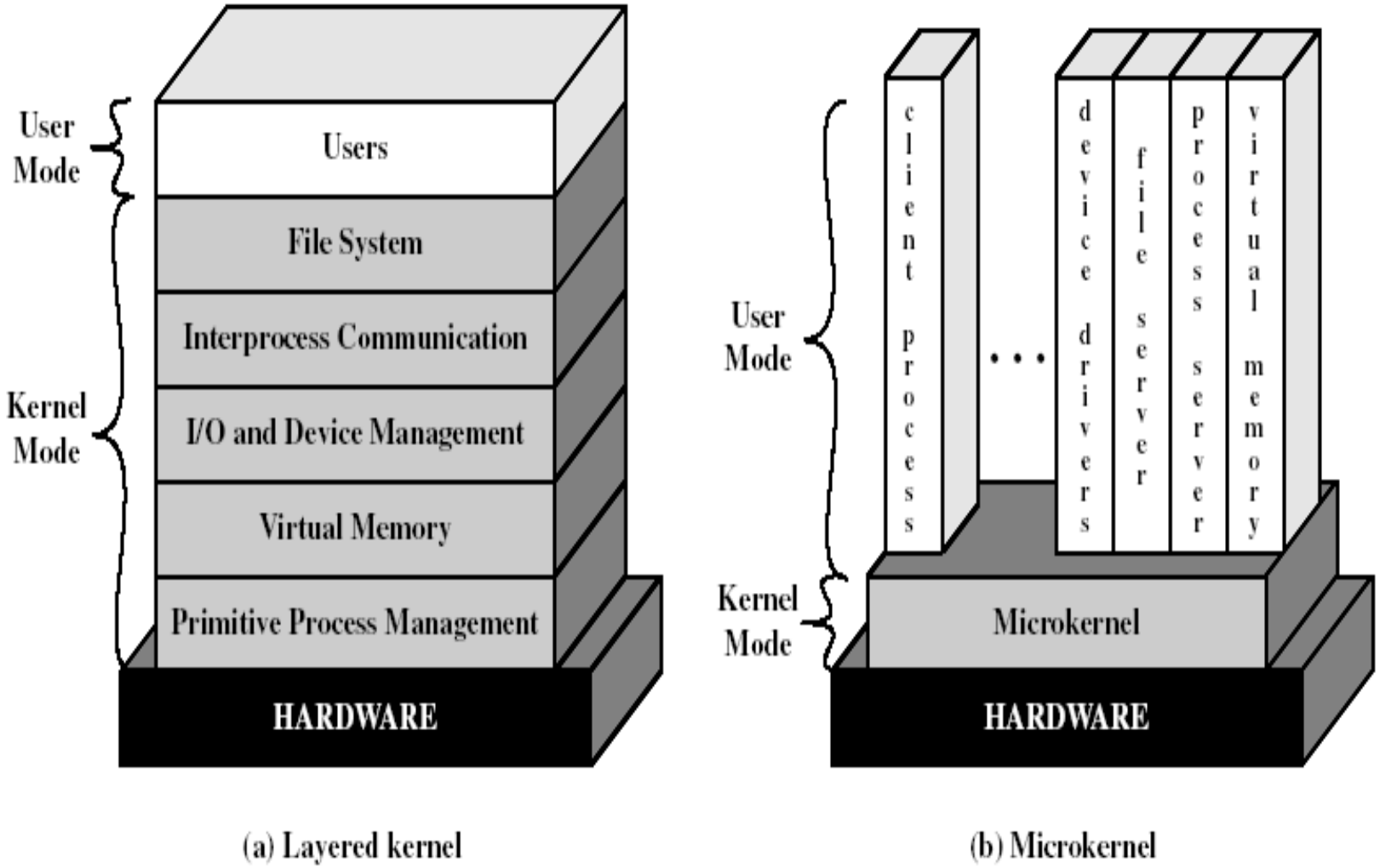
- When building distributed applications, system builders have often looked to the non-distributed systems world for models to follow (… inspiration?).
- Consequently, distributed systems tend to exhibit certain characteristics that are already familiar to us.
- This applies equally to hardware concepts as it does to software concepts.

Uniprocessor Operating Systems



Separating applications from operating system code through a “microkernel” – can provide a good base upon which to build a distributed OS (DOS)

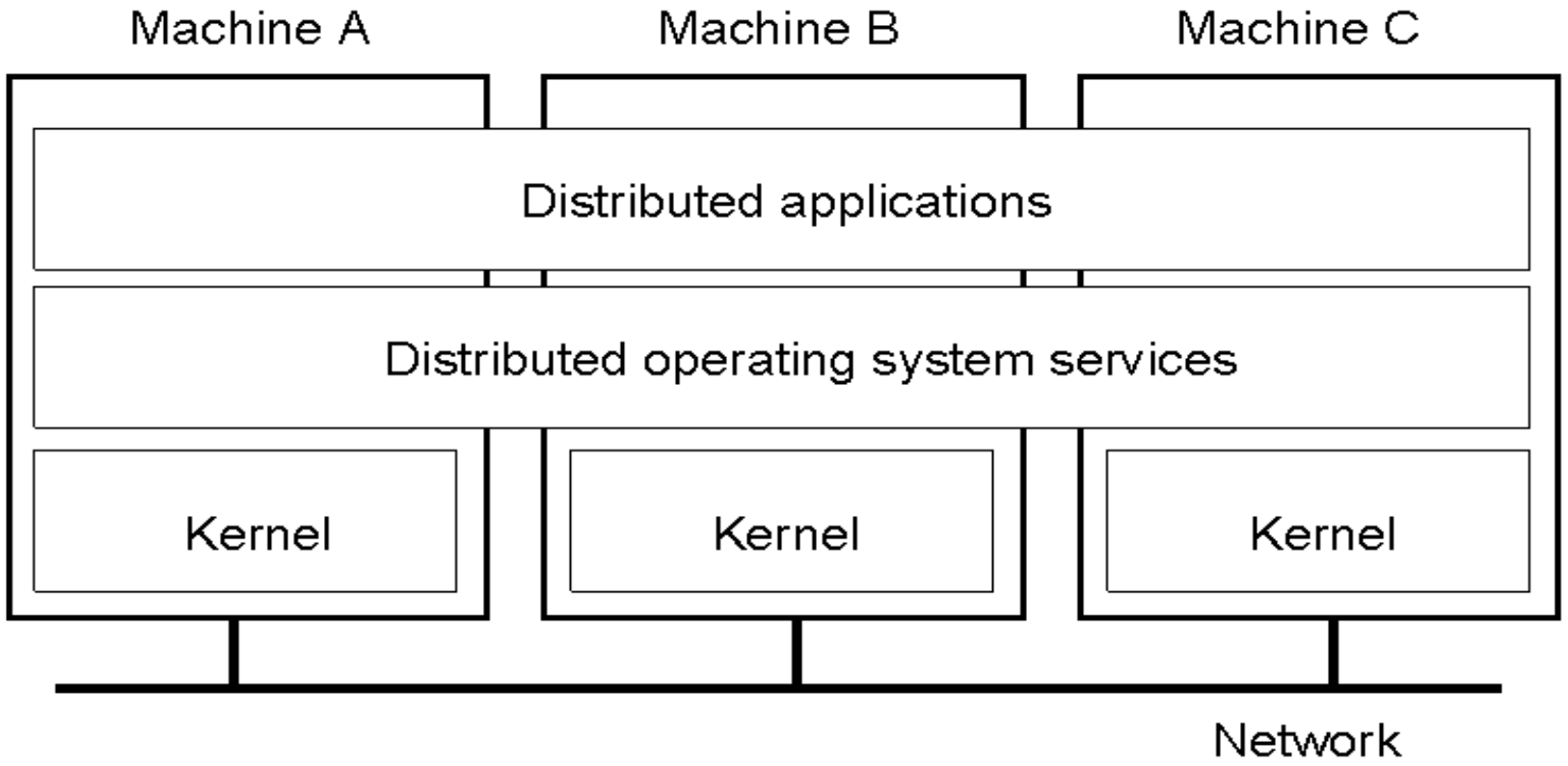
Layered vs. Microkernel Architecture



Types of Operating Systems

System	Description	Main Goal
DOS	Tightly-coupled operating system for multi-processors and homogeneous multicomputers	Hide and manage hardware resources
NOS	Loosely-coupled operating system for heterogeneous multicomputers (LAN and WAN)	Offer local services to remote clients
Middleware	Additional layer atop of NOS implementing general-purpose services	Provide distribution transparency

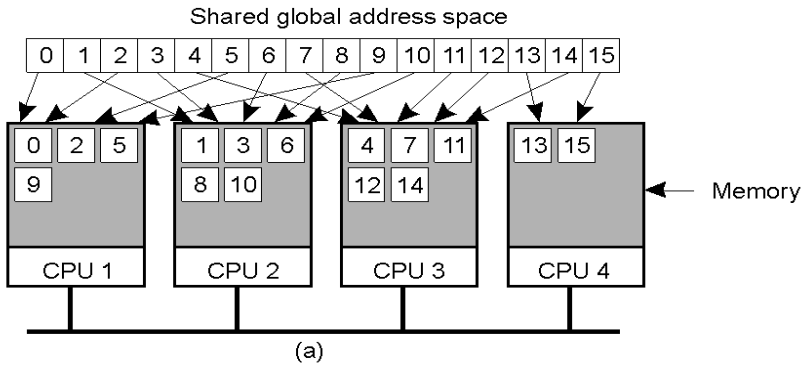
Multicomputer Operating Systems



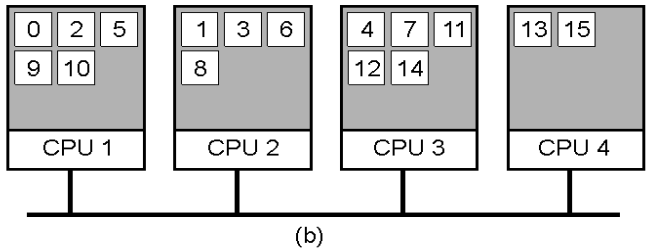
General structure of a multicomputer operating system – all the systems are of the same type: homogeneous

Distributed Shared Memory Systems (1)

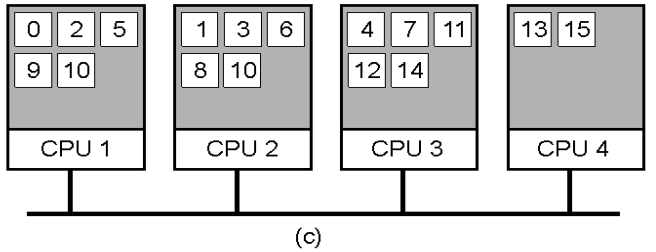
a) Pages of address space distributed among four machines



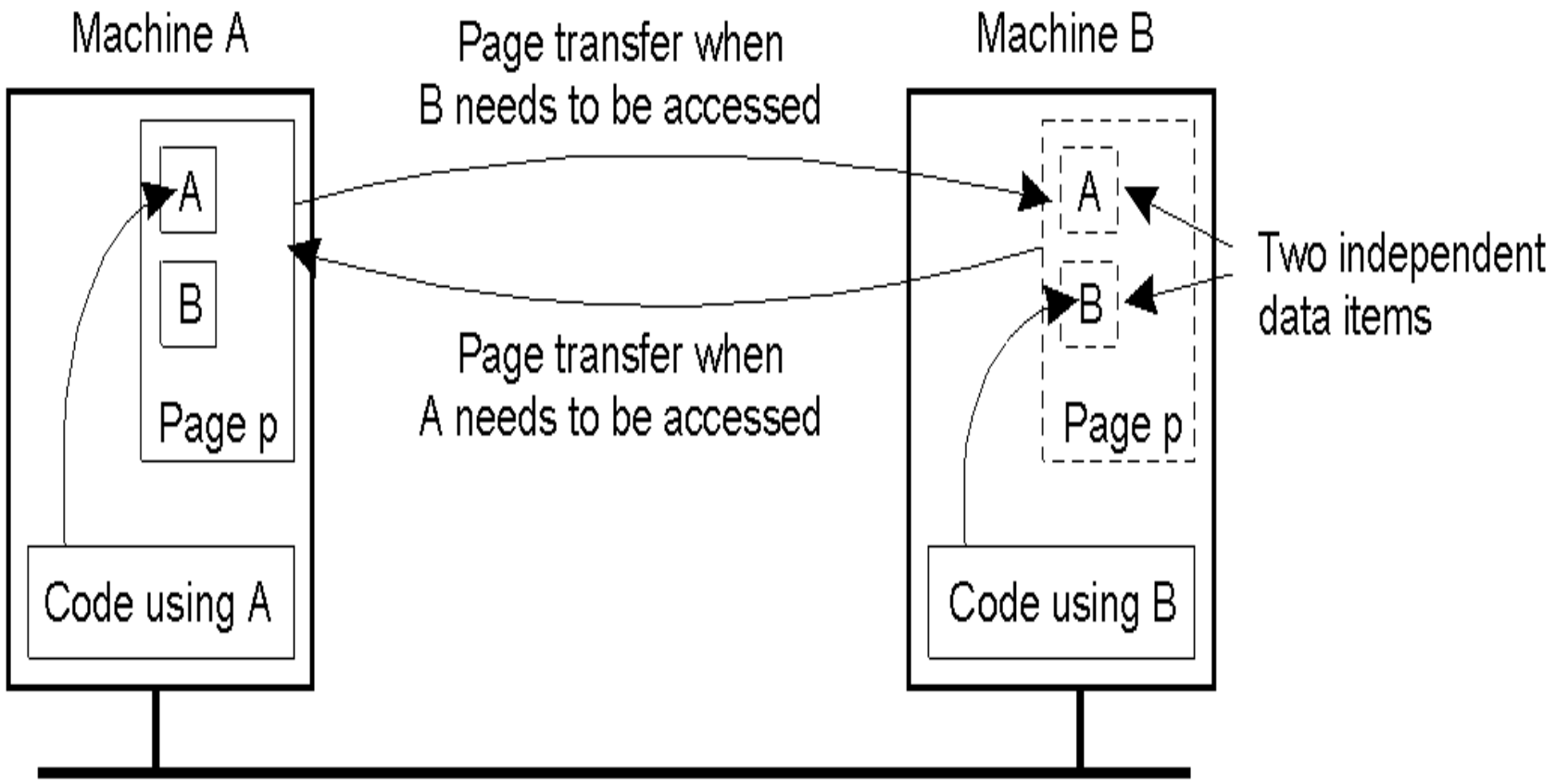
b) Situation after CPU 1 references page 10



c) Situation if page 10 is read only and replication is used

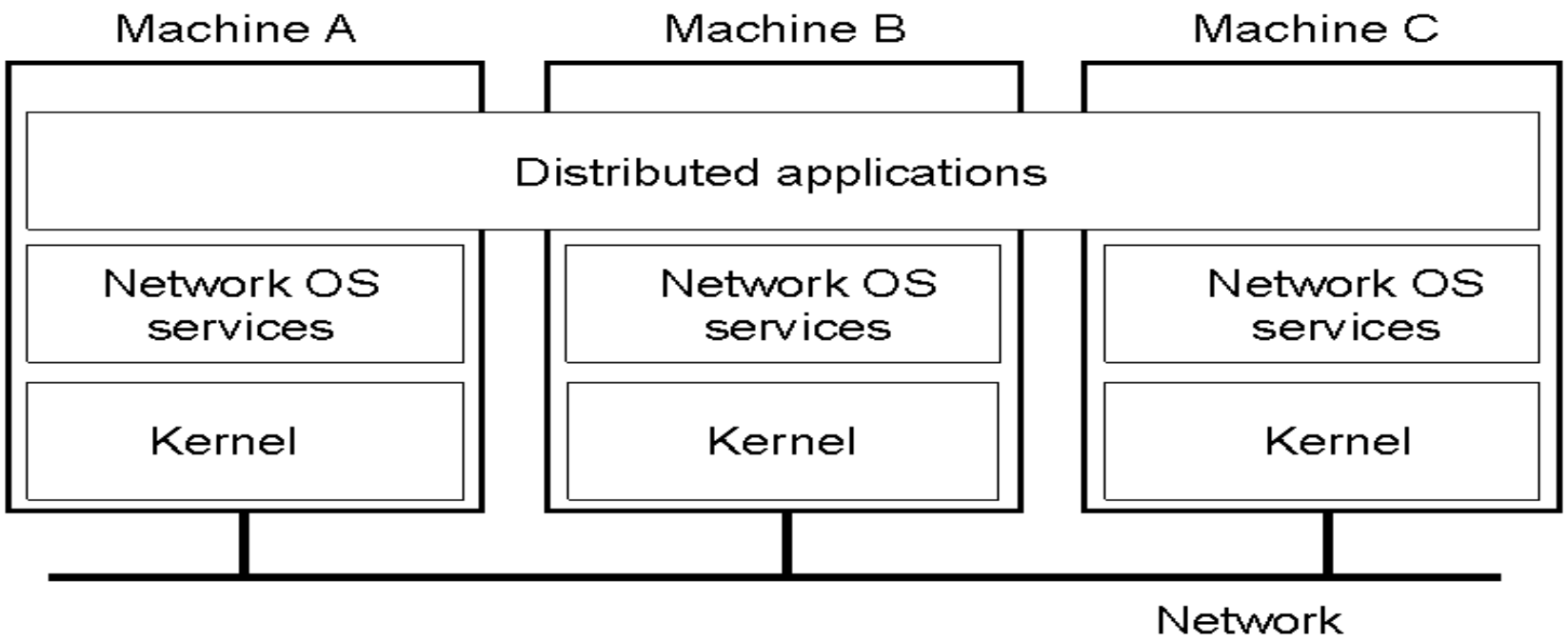


Distributed Shared Memory Systems (2)



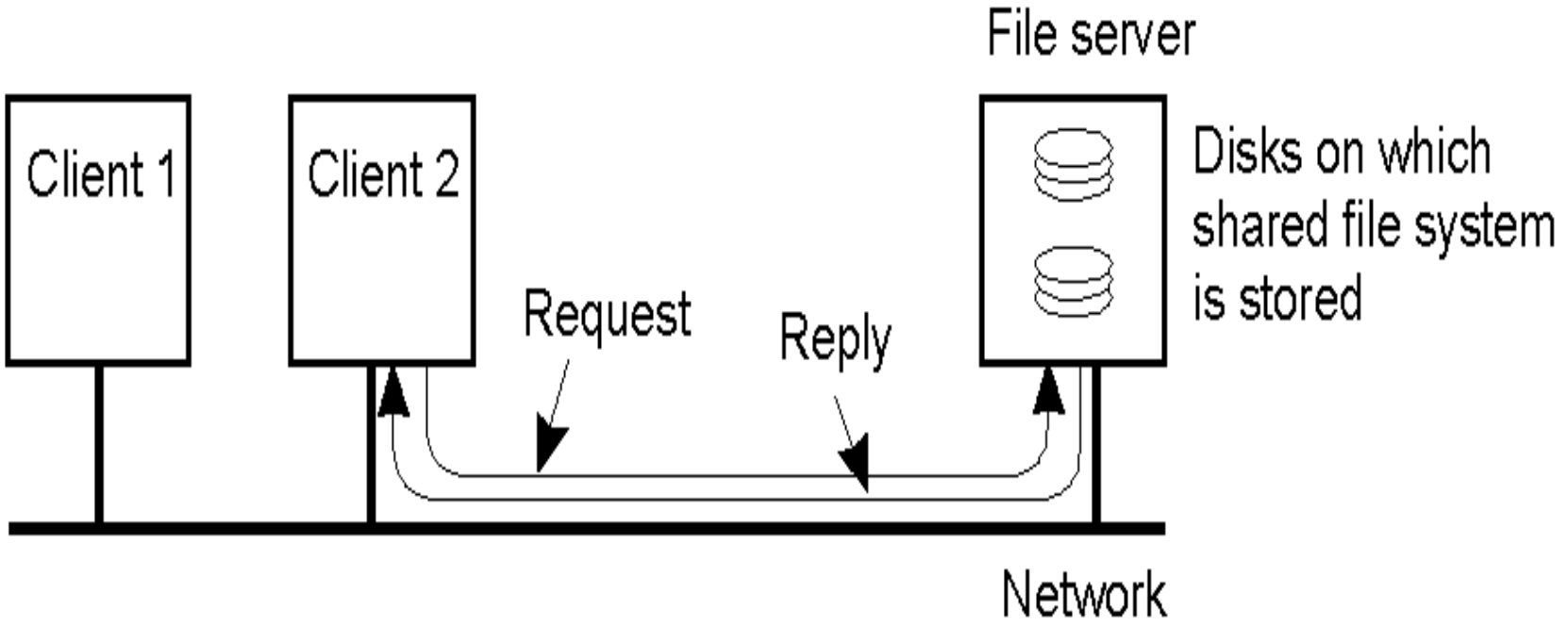
False sharing of a page between 2 independent processes

Network Operating System (1)



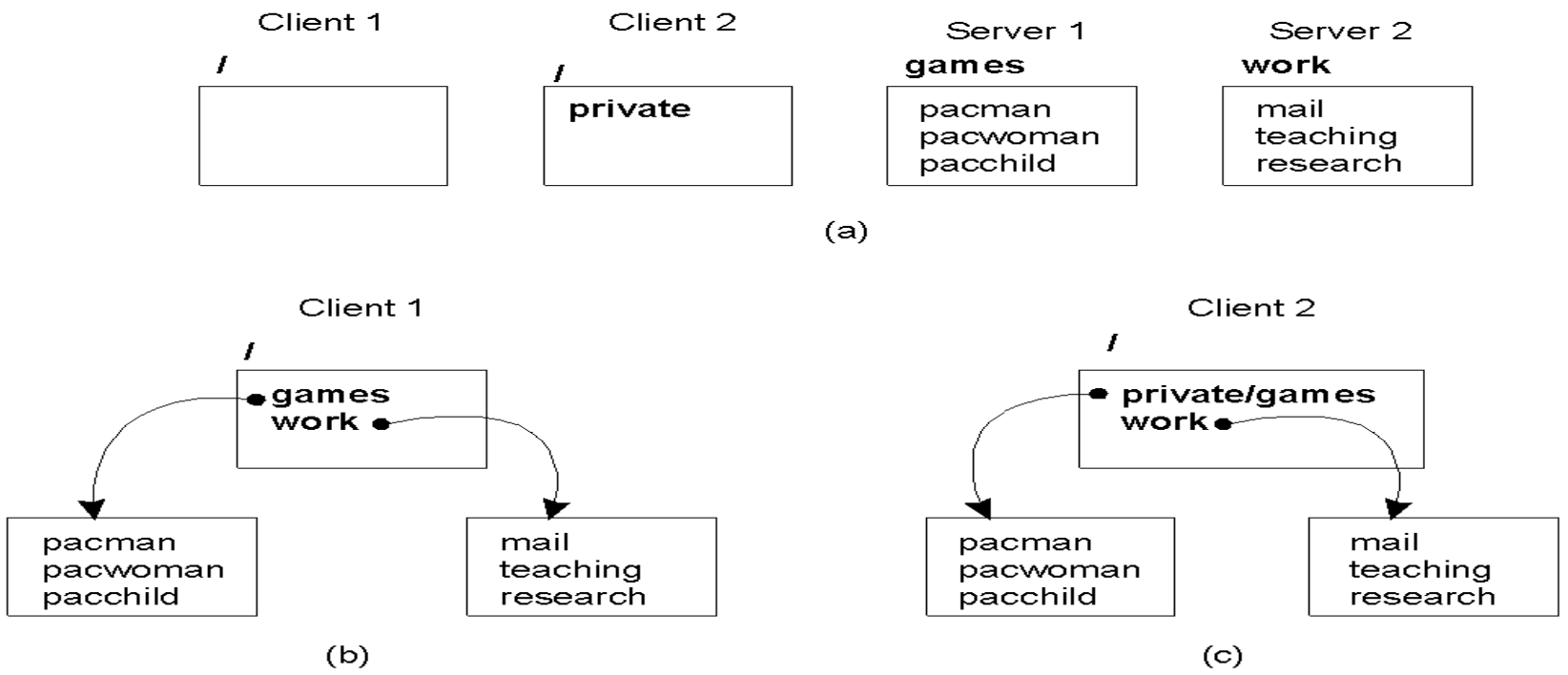
General structure of a network operating system – all the systems are of different types: heterogeneous

Network Operating System (2)



Two clients and a server in a network operating system – relatively primitive set of services provided

Network Operating System (3)

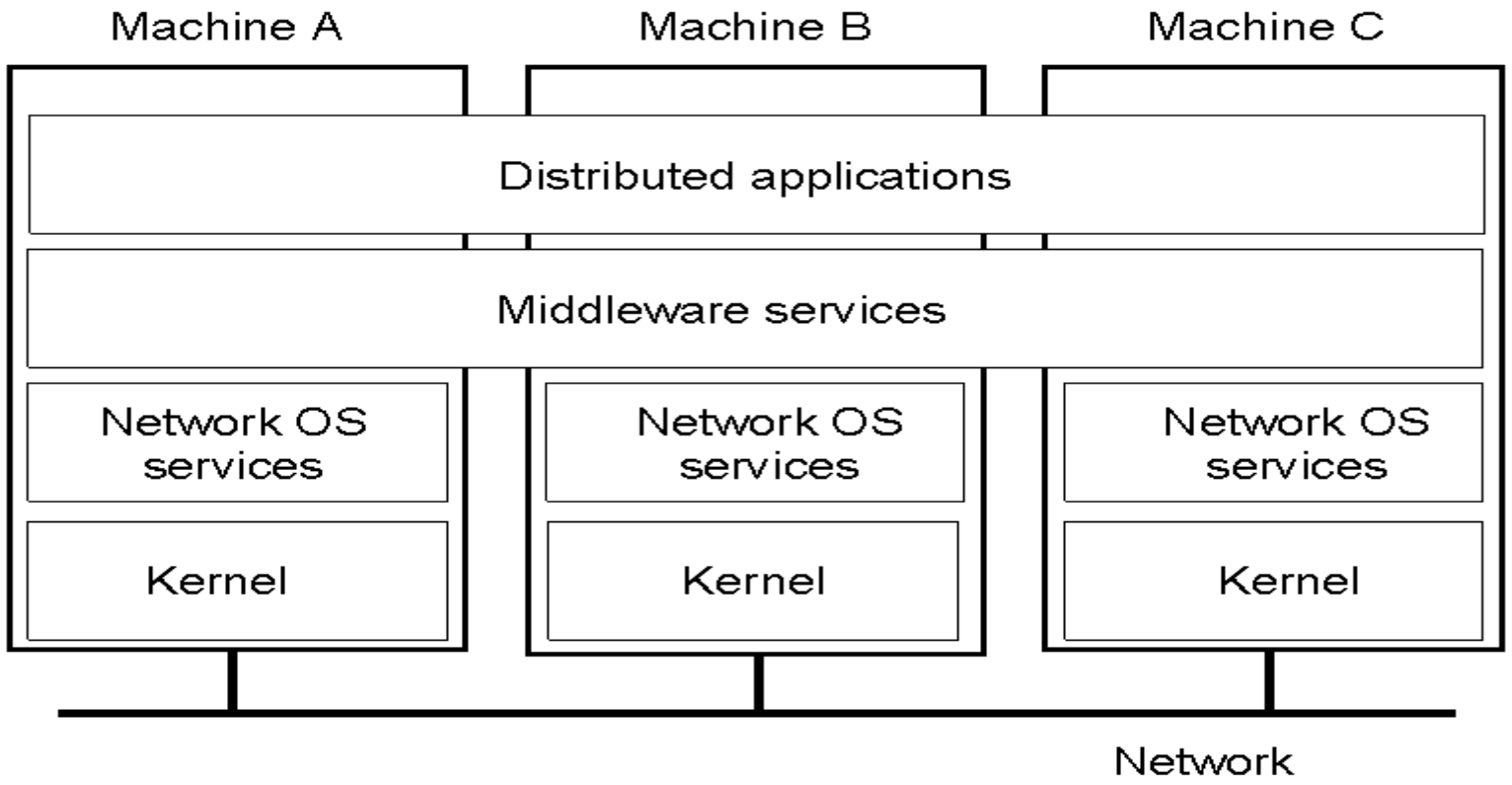


Different clients may mount the servers in different places – difficult to maintain a consistent “view” of the system

The Best of Both Worlds?

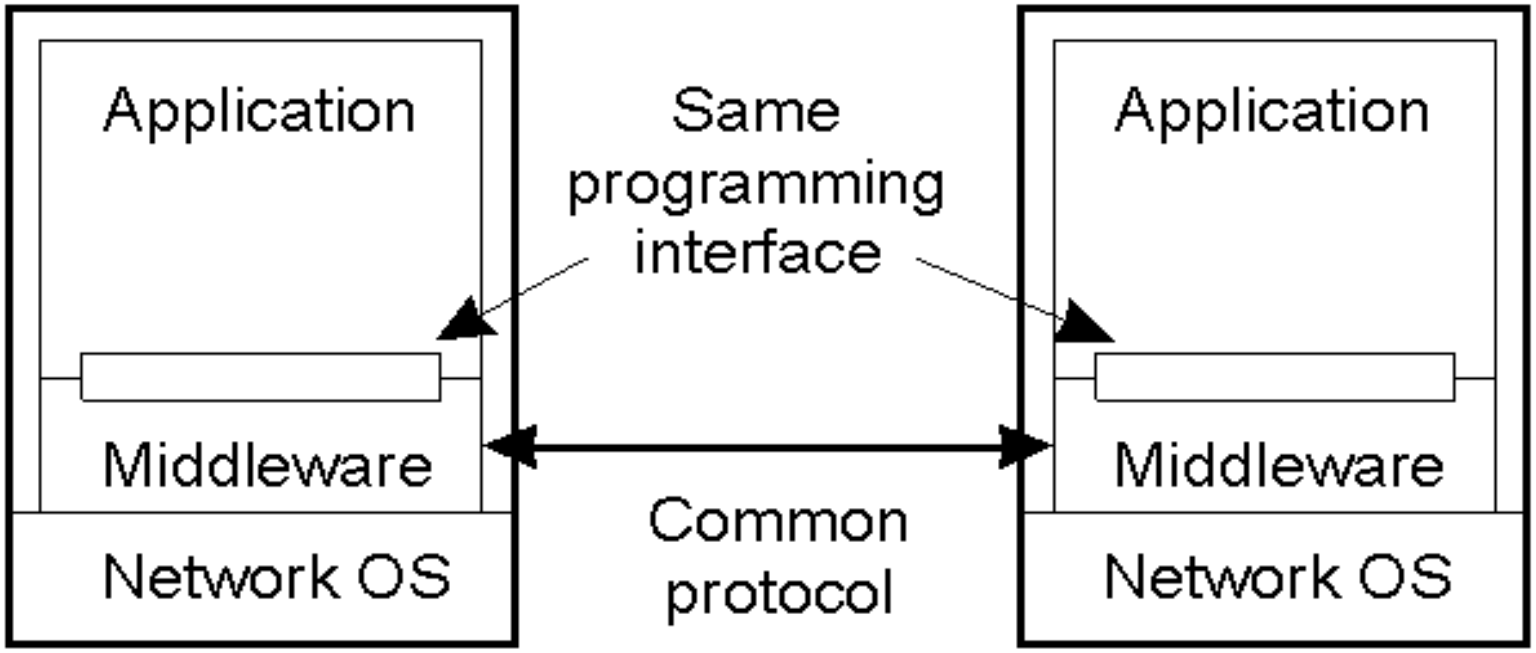
- DOS: too inflexible (all systems of the same type).
- NOS: too primitive (lowest common denominator – too much diversity).
- “Middleware” – best possible compromise?
- Middleware = NOS + additional software layer.

Positioning Middleware



General structure of a distributed system as middleware

Middleware and Openness



In an open middleware-based distributed system, the protocols used by each middleware layer should be the same, as well as the interfaces they offer to applications. This is a much higher level of abstraction than (for instance) the NOS Socket API.

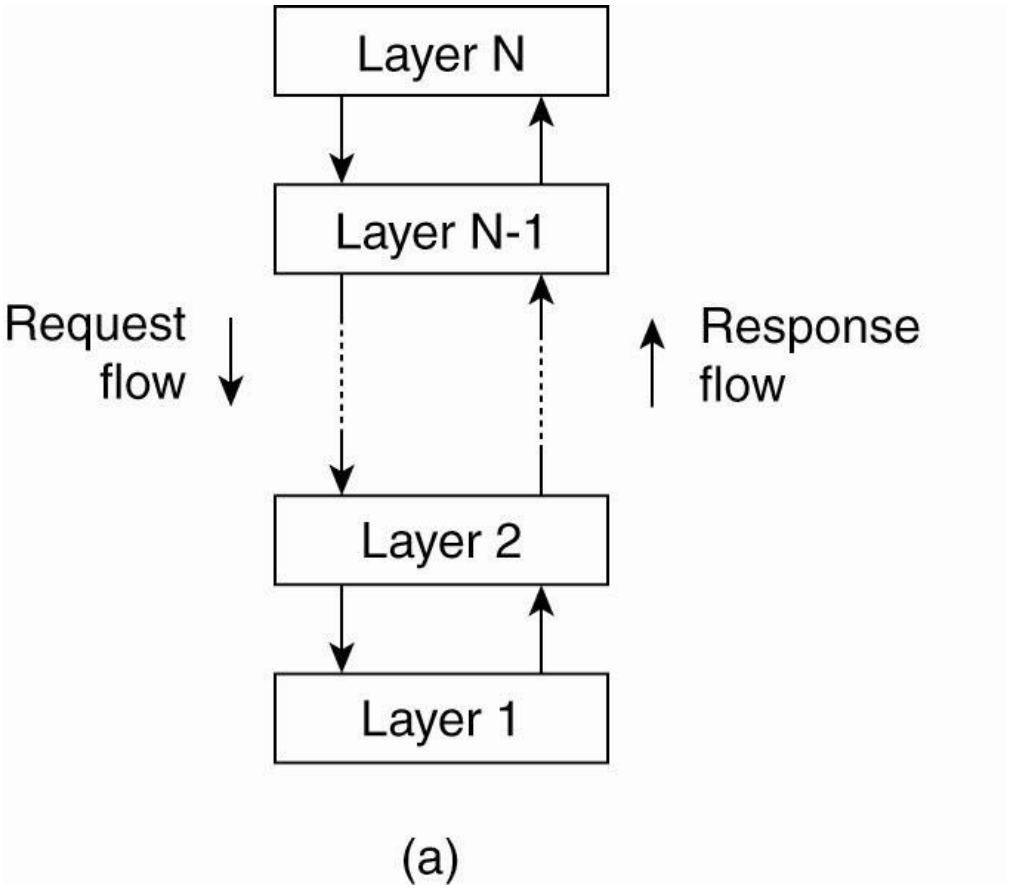
Comparison between the systems

Item	Distributed OS		Network OS	Middleware-based OS
	Multiproc.	Multicomp.		
Degree of transparency	Very High	High	Low	High
Same OS on all nodes	Yes	Yes	No	No
Number of copies of OS	1	N	N	N
Basis for communication	Shared memory	Messages	Files	Model specific
Resource management	Global, central	Global, distributed	Per node	Per node
Scalability	No	Moderately	Yes	Varies
Openness	Closed	Closed	Open	Open

Architectural Styles

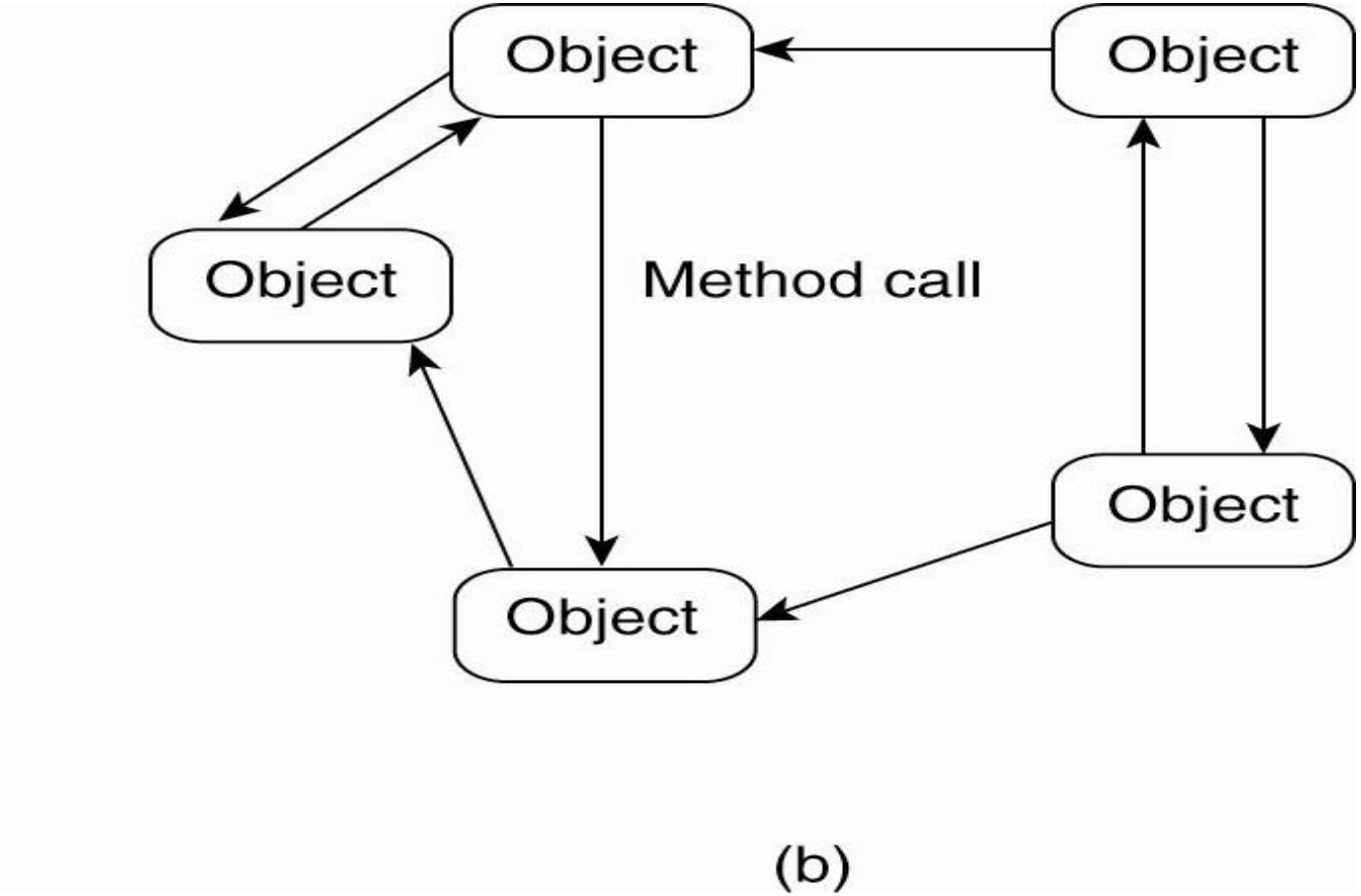
- Important styles of architecture for distributed systems:
  - Layered architectures
  - Object-based architectures
  - Data-centered architectures
  - Event-based architectures

Architectural Styles (2)



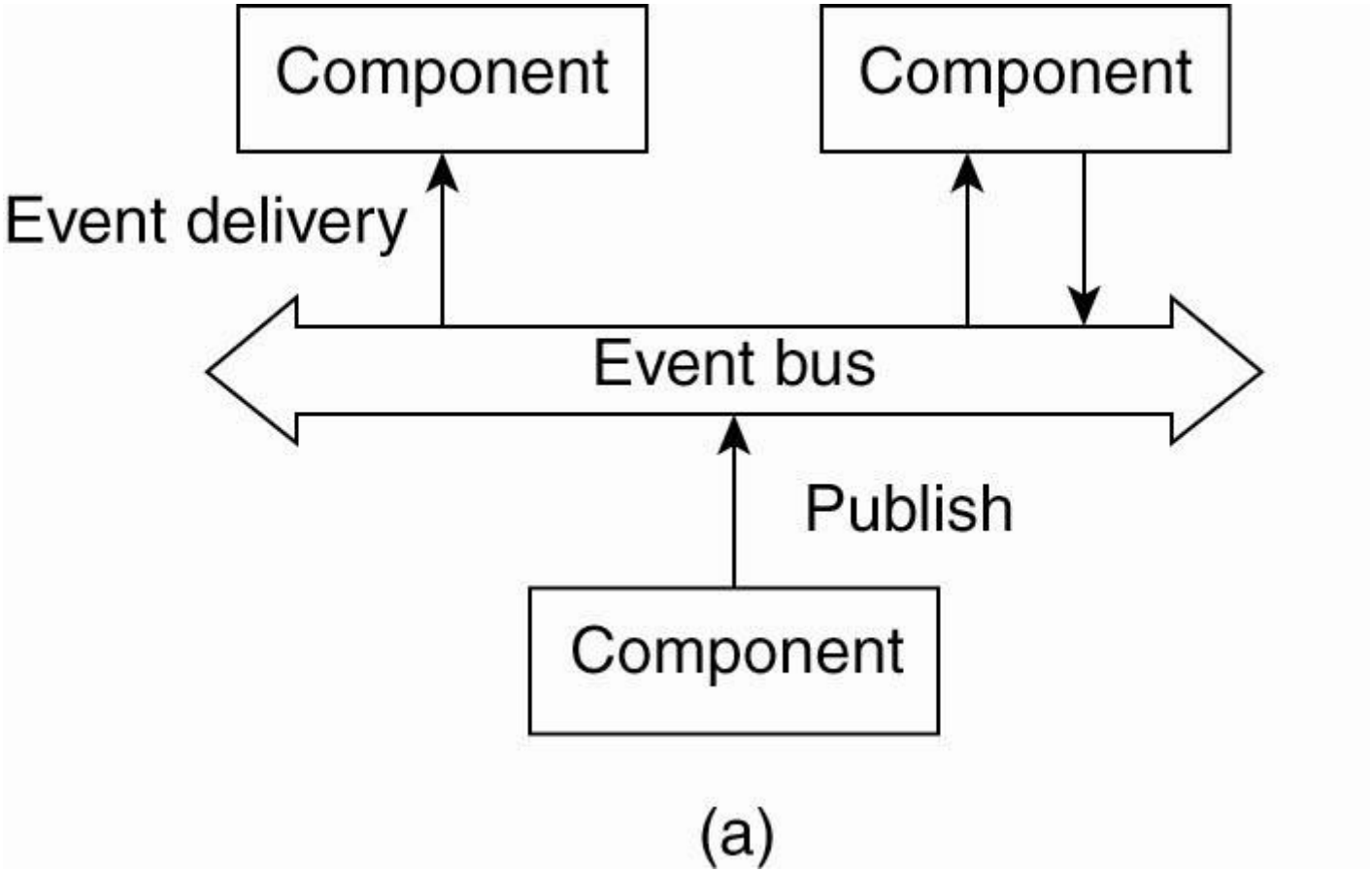
The (a) layered architectural style and …

Architectural Styles (3)



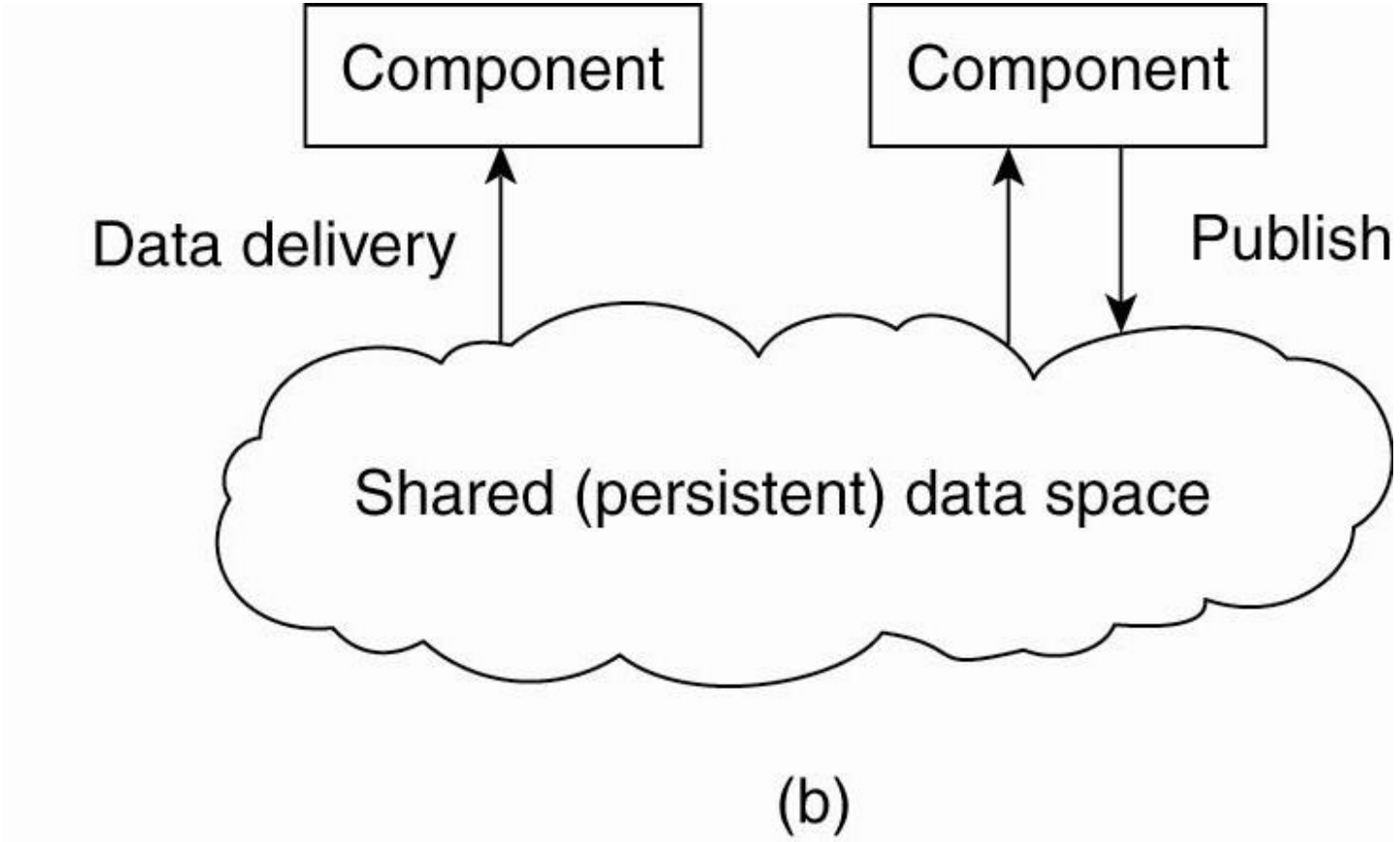
(b) The object-based architectural style

Architectural Styles (4)



(a) The event-based architectural style and …

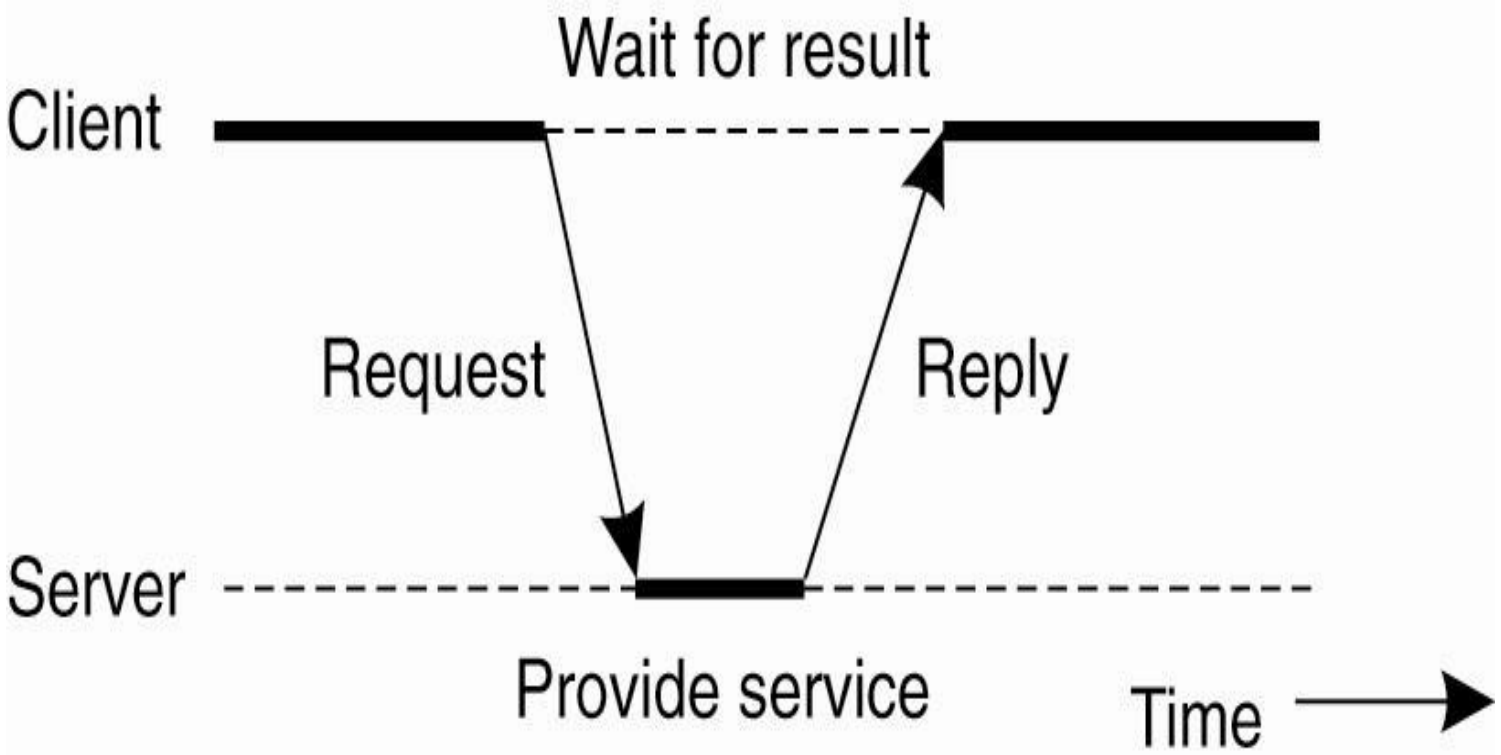
Architectural Styles (5)



(b) The shared data-space architectural style

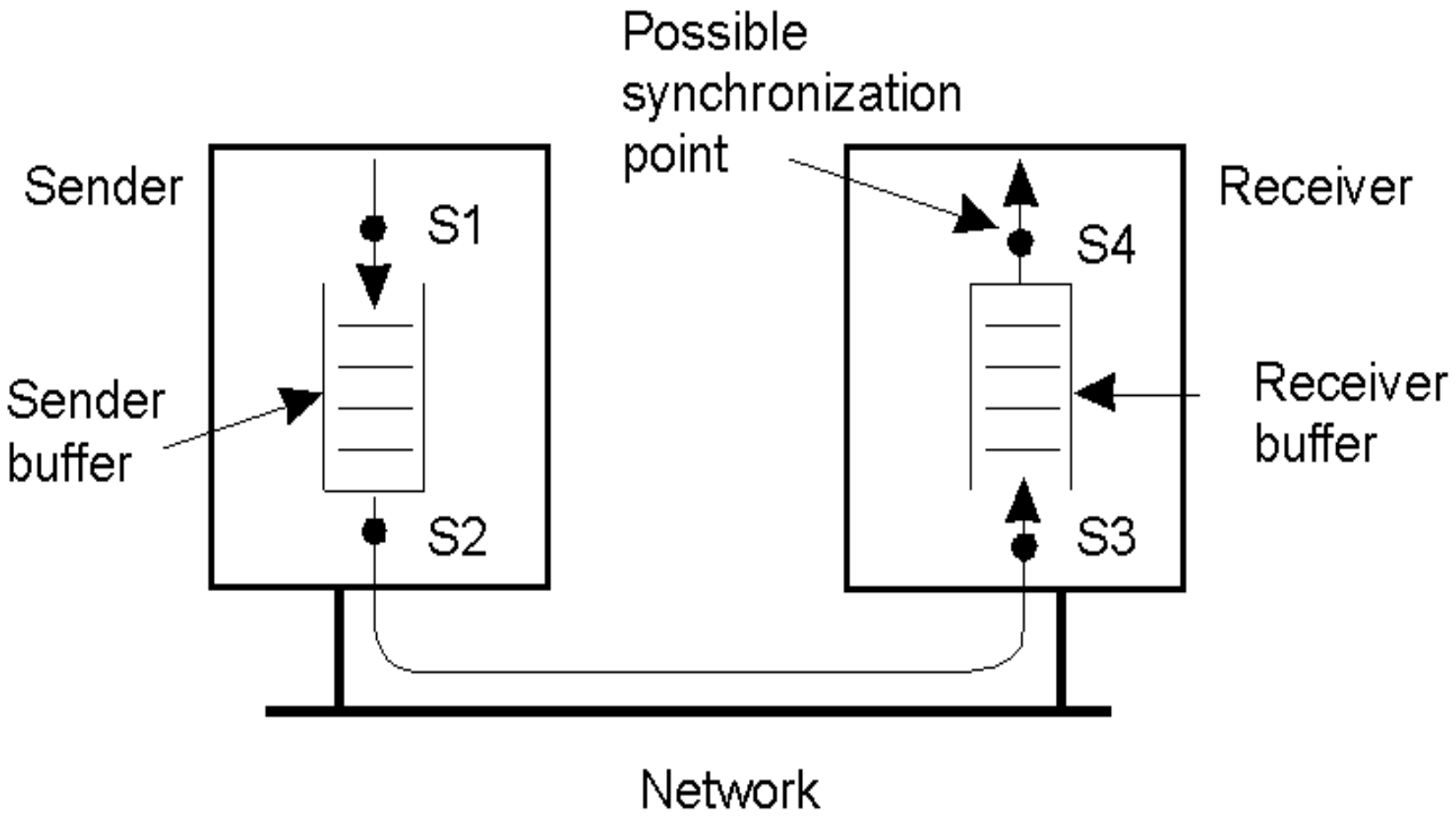
Centralized Architectures

Client/Server Model



General interaction between a client and a server

Alternatives for blocking and buffering in message passing



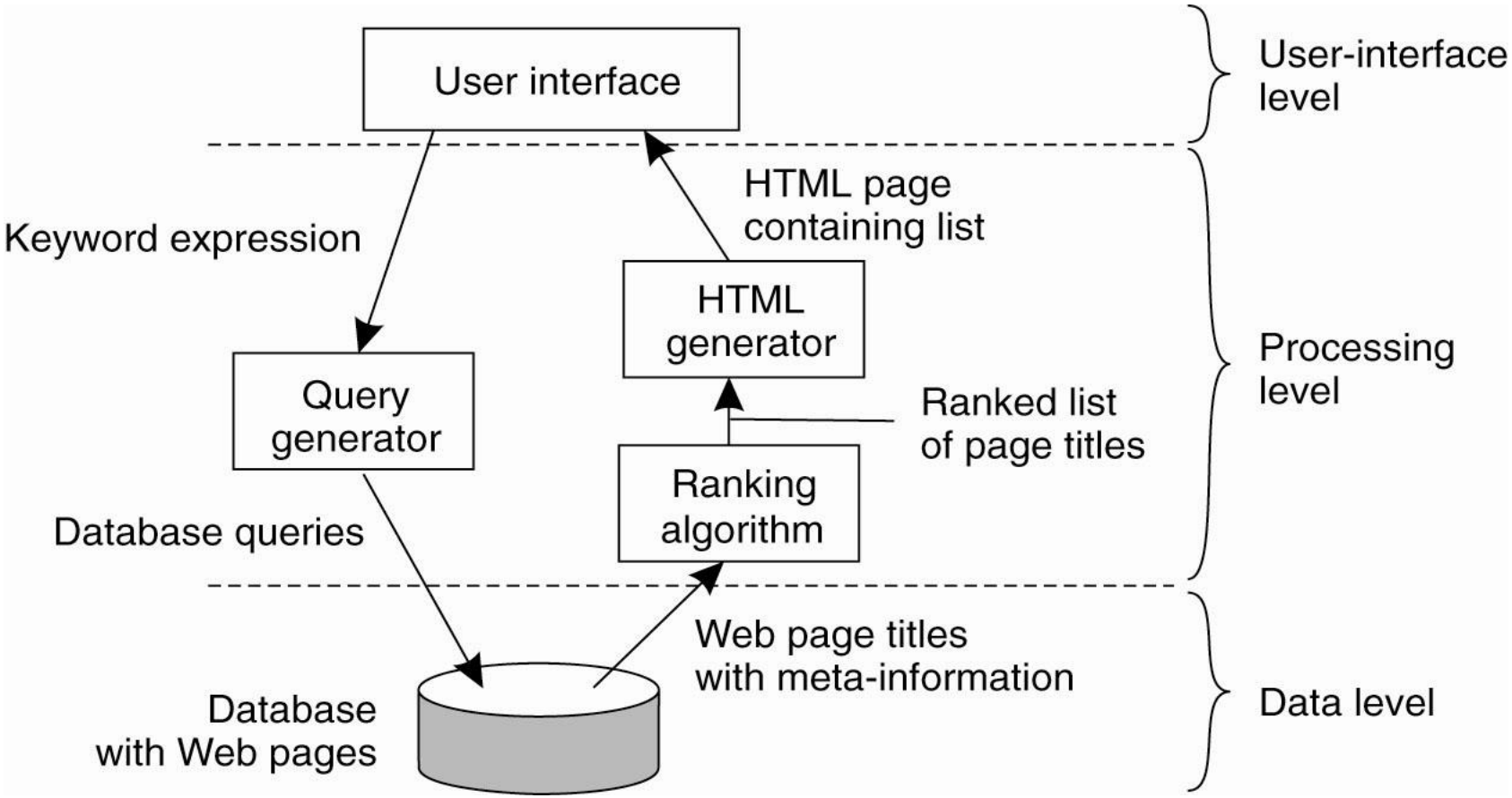
Relation between blocking, buffering, and reliable communications

Synchronization point	**Sender buffer**	**Reliable communication guaranteed?**
Block sender until buffer not full	Yes	Not necessarily
Block sender until message sent	No	Not necessarily
Block sender until message received	No	Necessary
Block sender until message delivered	No	Necessary

Application Leveling (1)

- Traditional three levels of architectural style:
  1. User-interface level
  2. Processing level
  3. Data level

Application Leveling (2)

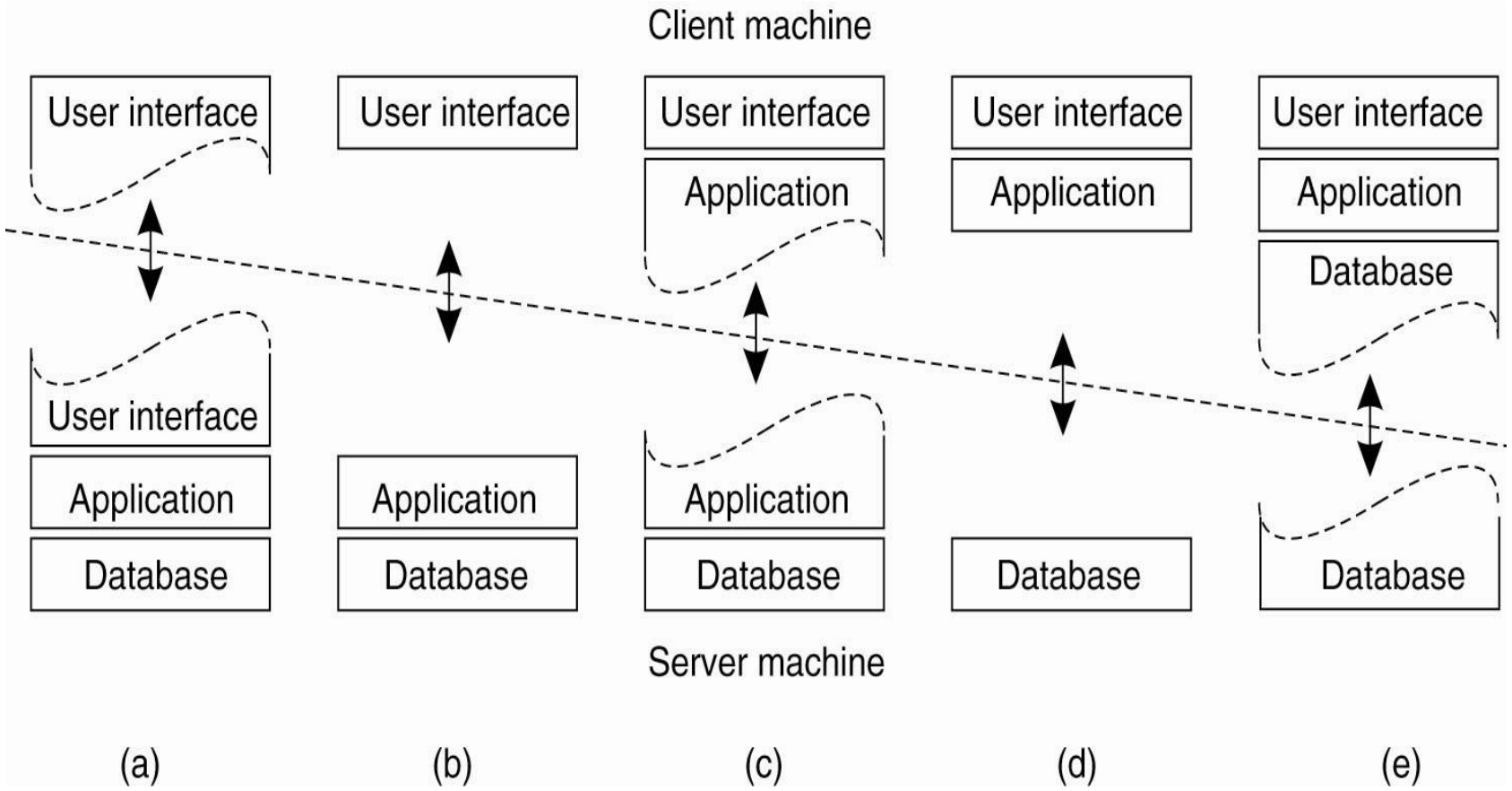


The simplified organization of an Internet search engine into three different levels

Multi-tiered Architectures (1)

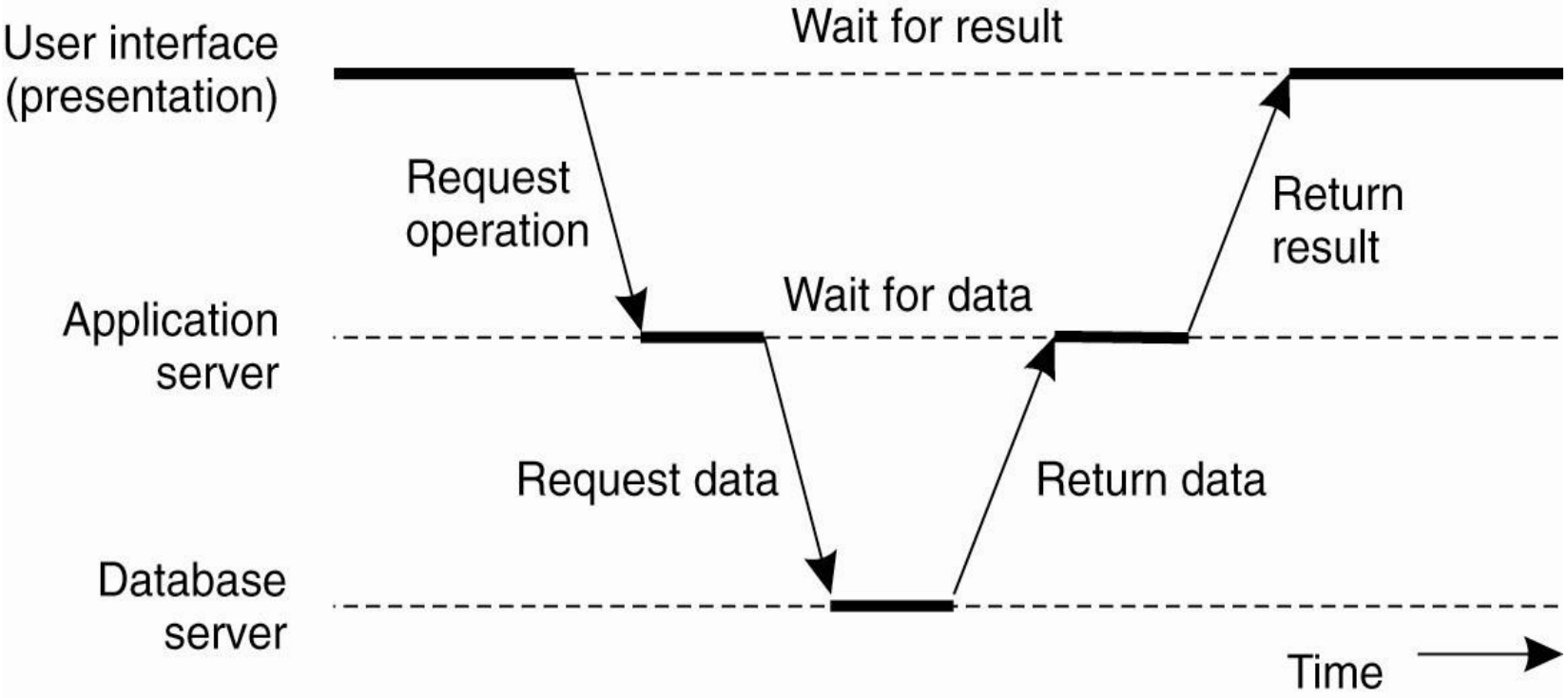
- The simplest organization is to have only two types of machines:
  1. A client machine containing only the programs implementing (part of) the user-interface level.
  2. A server machine containing the rest, the programs implementing the processing and data levels.

Multi-tiered Architectures (2)



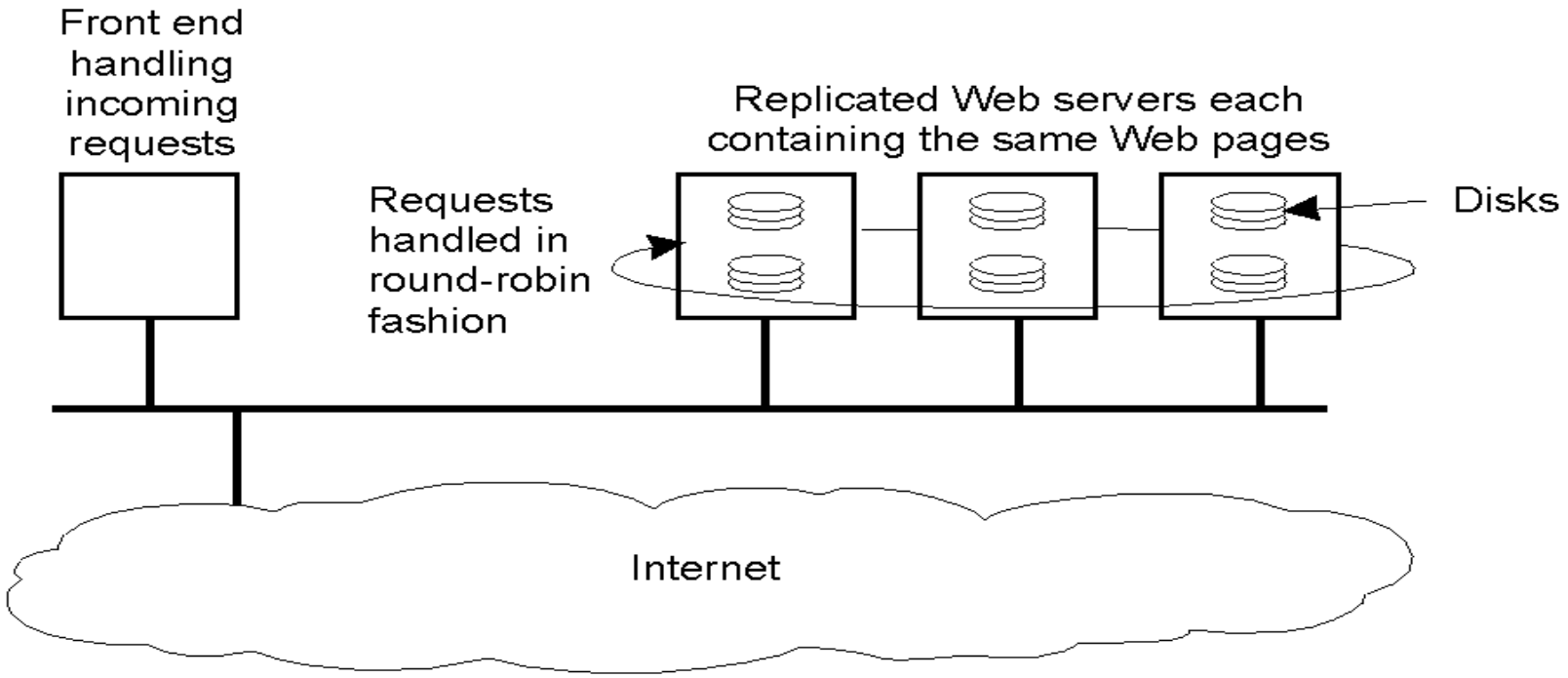
Alternative client-server organizations (a)–(e)

Multi-tiered Architectures (3)



An example of a server acting as a client – this is a very common **vertical distribution** model for distributed systems

Replicated Web Servers

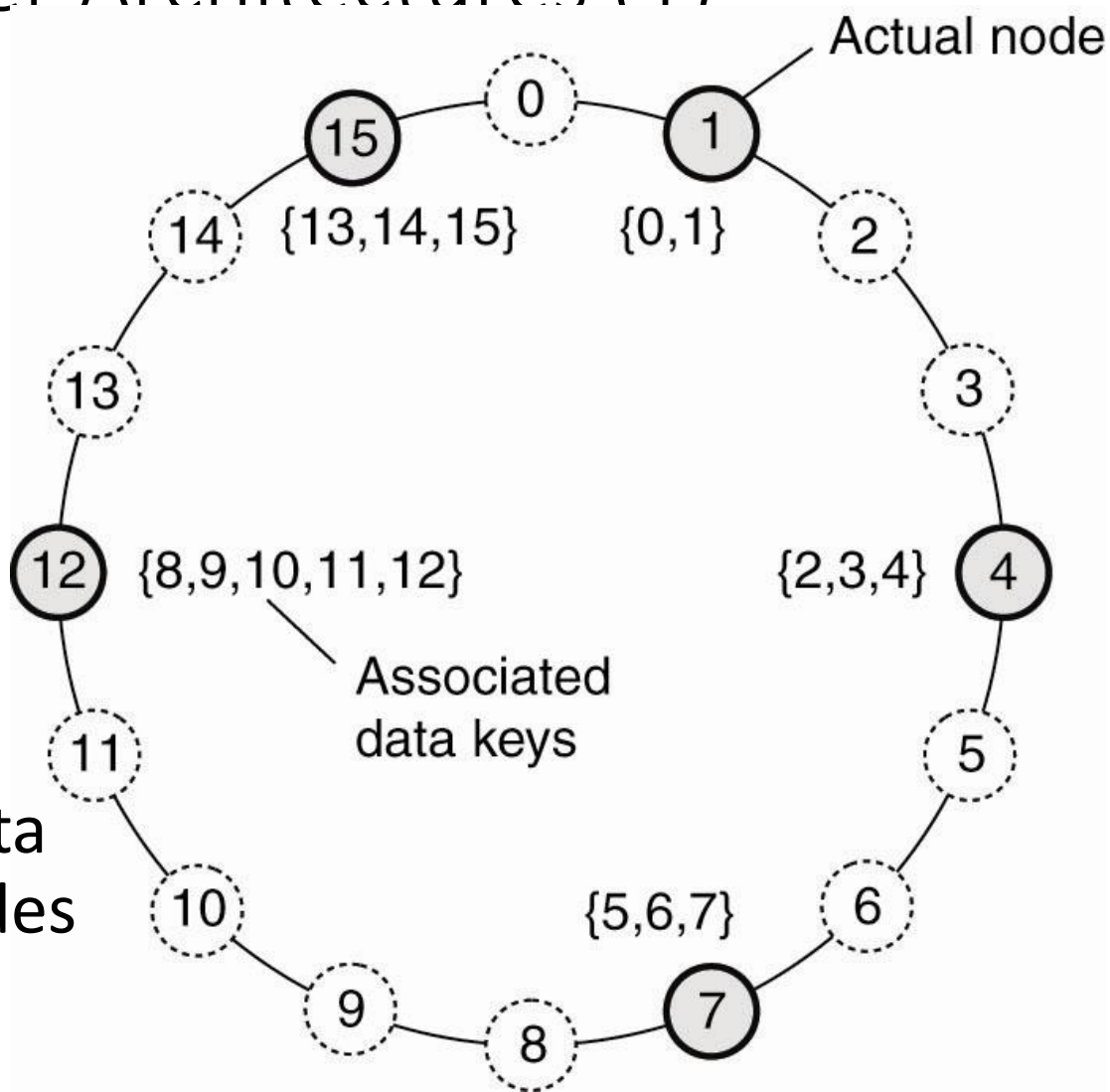


An example of **horizontal distribution** of a Web service (often also referred to as “clustering”)

Decentralized Architectures

- Three types of Peer-to-Peer (P2P) systems:
  1. Structured P2P: nodes are organized following a specific distributed data structure.
  2. Unstructured P2P: nodes have randomly selected neighbors.
  3. Hybrid P2P: some nodes are appointed special functions in a well-organized fashion.

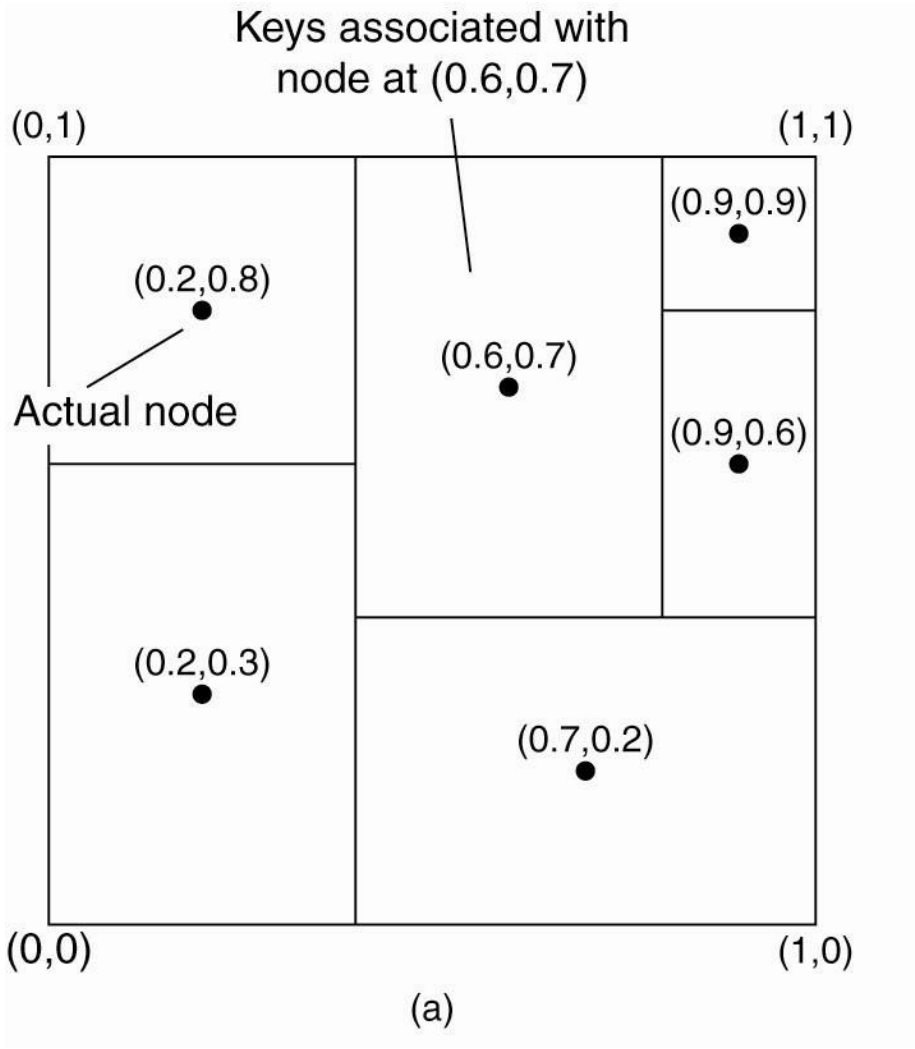
Structured Peer-to-Peer Architectures (1)



The mapping of data items onto nodes in Chord

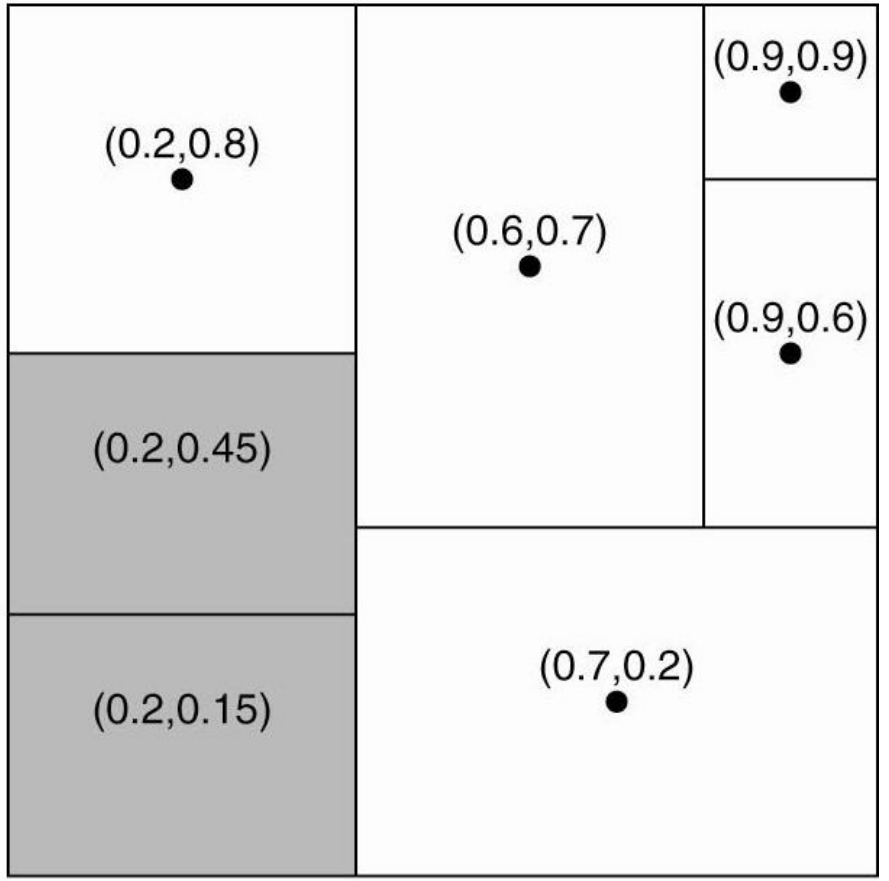
Structured Peer-to-Peer Architectures (2)

(a) The mapping of data items onto nodes in CAN



Structured Peer-to-Peer Architectures (3)

(b) Splitting a region when a node joins



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)

(a) The steps taken by the active thread

Unstructured Peer-to-Peer Architectures (2)

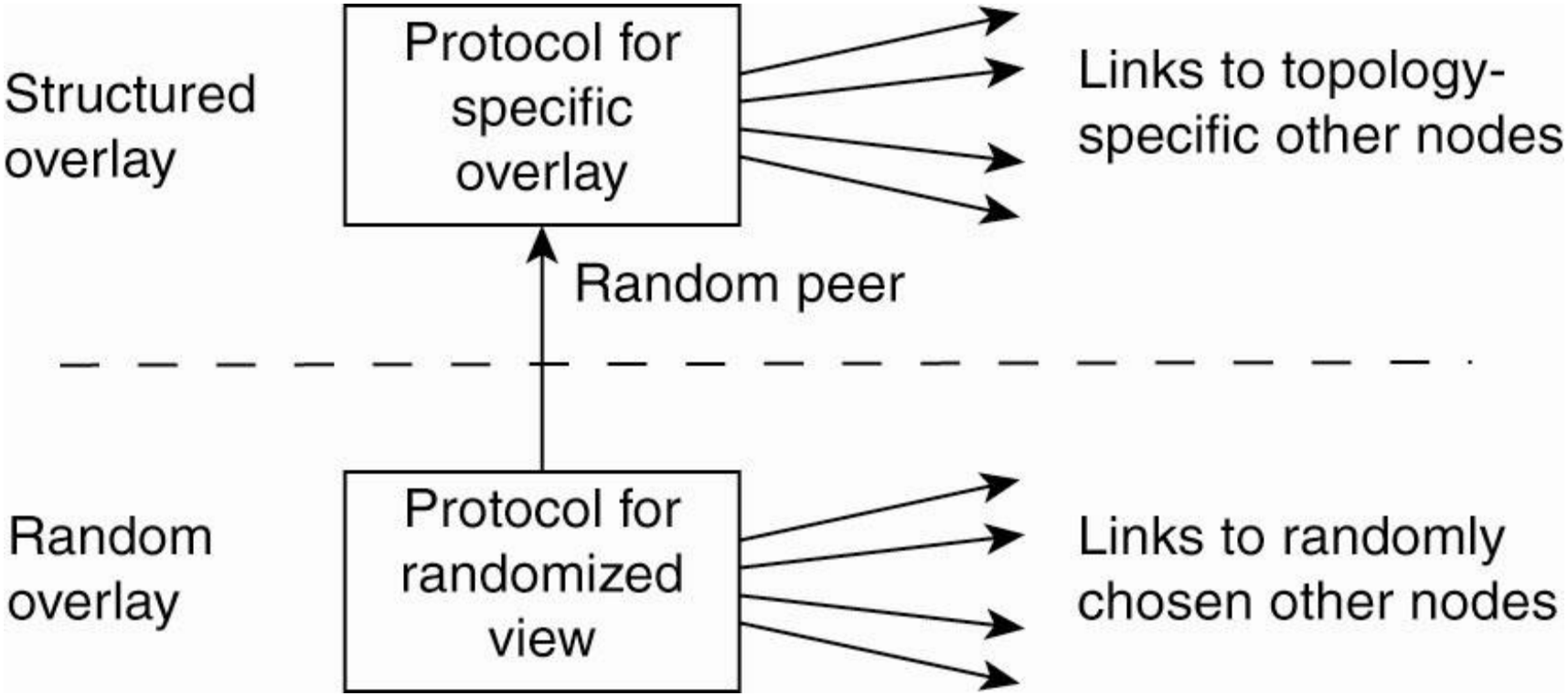
Actions by passive thread:

```
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)

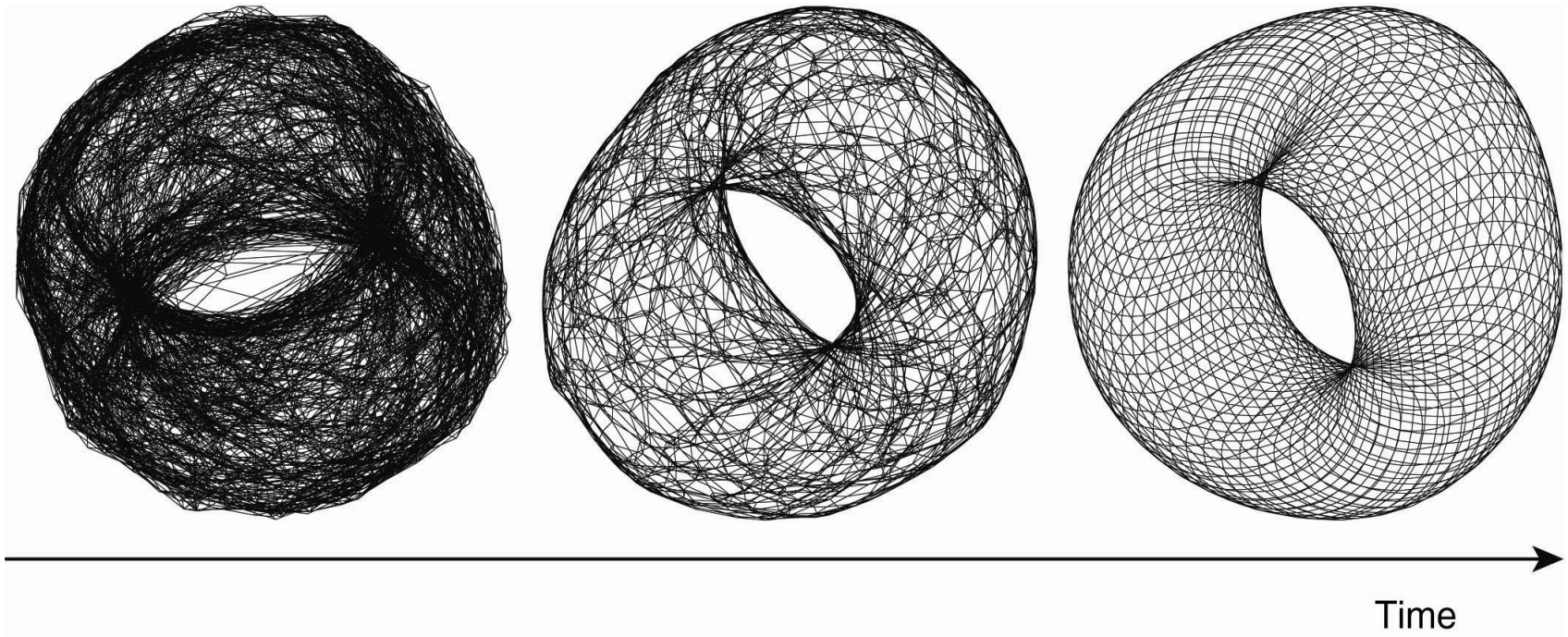
(b) The steps taken by the passive thread

Topology Management of Overlay Networks (1)



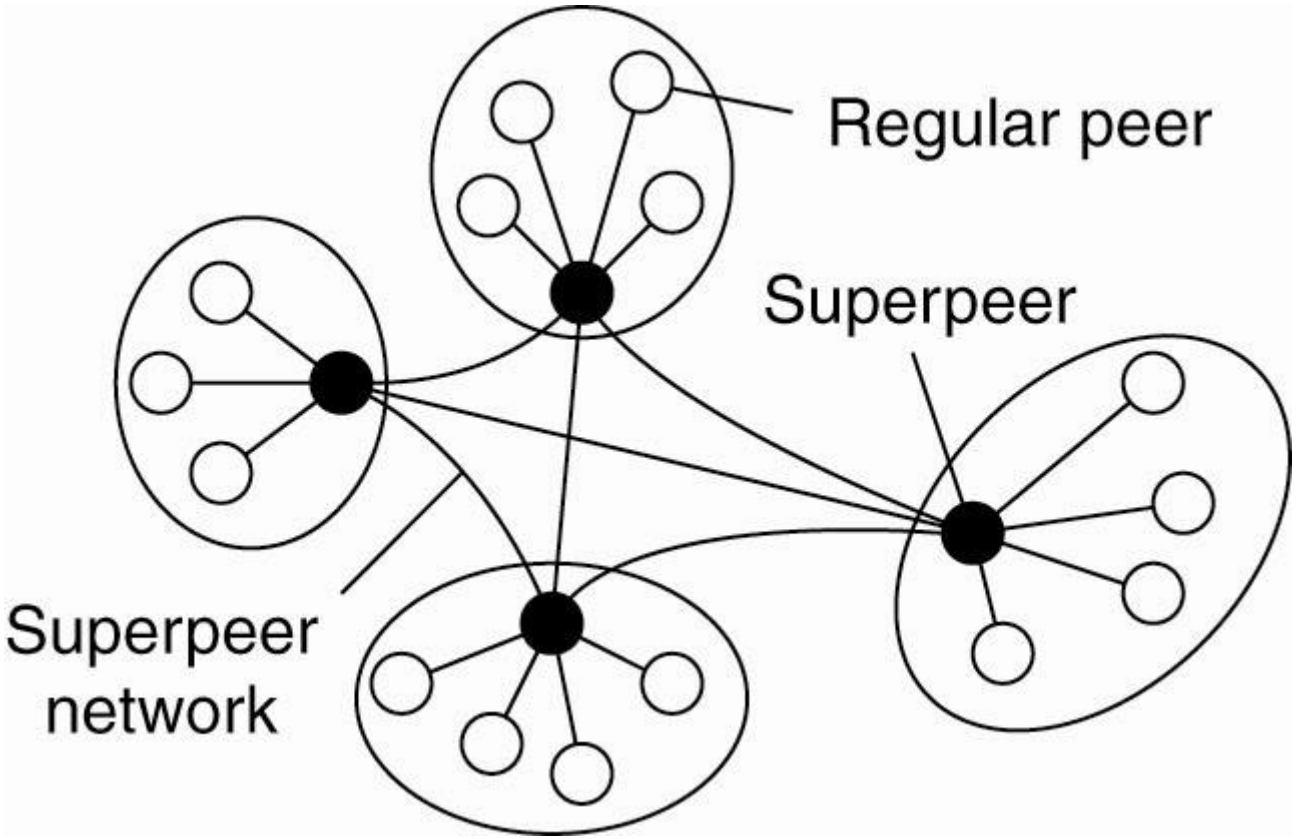
A two-layered approach for constructing and maintaining specific overlay topologies using techniques from unstructured peer-to-peer systems

Topology Management of Overlay Networks (2)



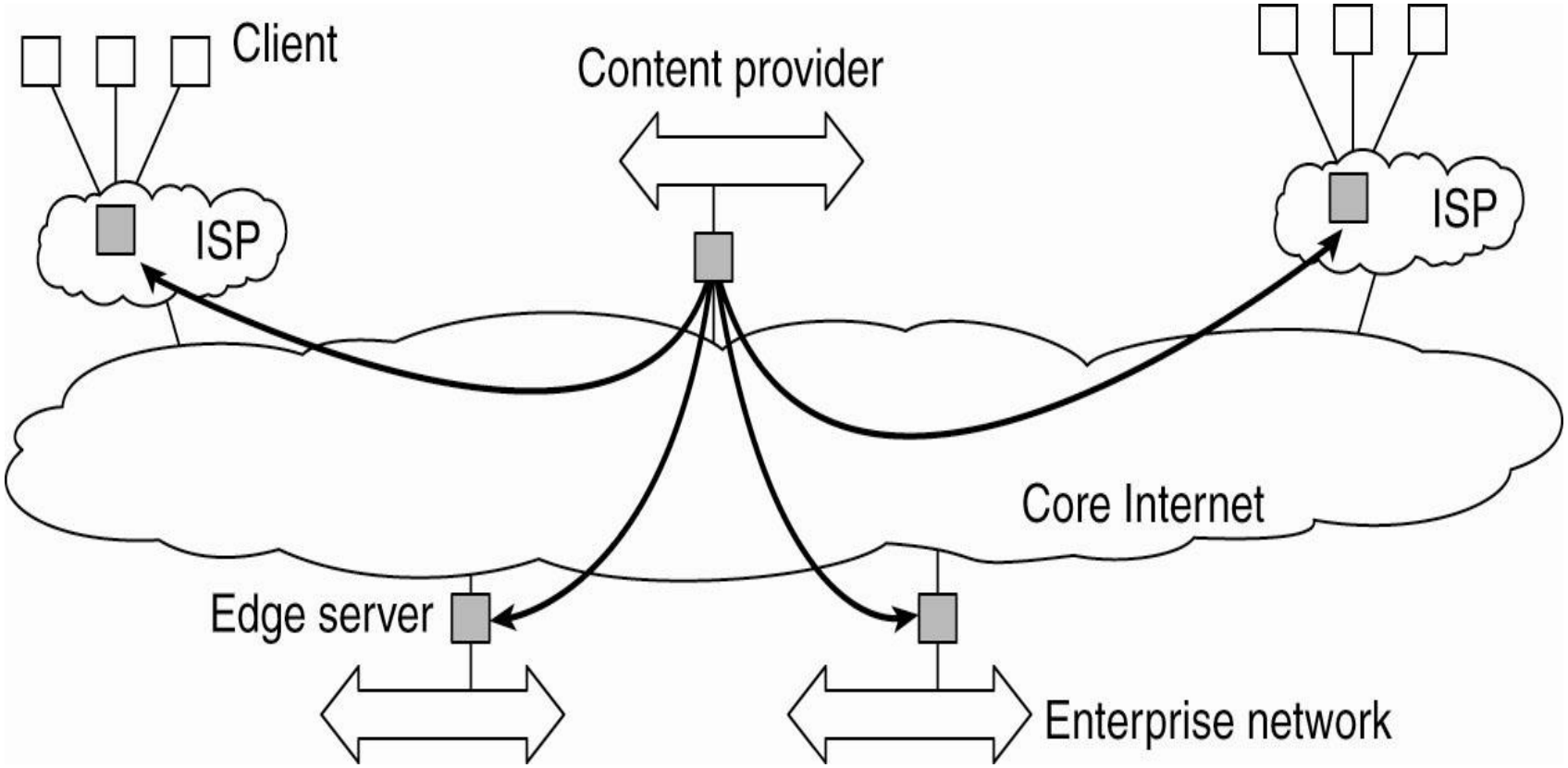
Generating a specific overlay network using a two-layered unstructured peer-to-peer system [adapted with permission from Jelasity and Babaoglu (2005)]

Superpeers



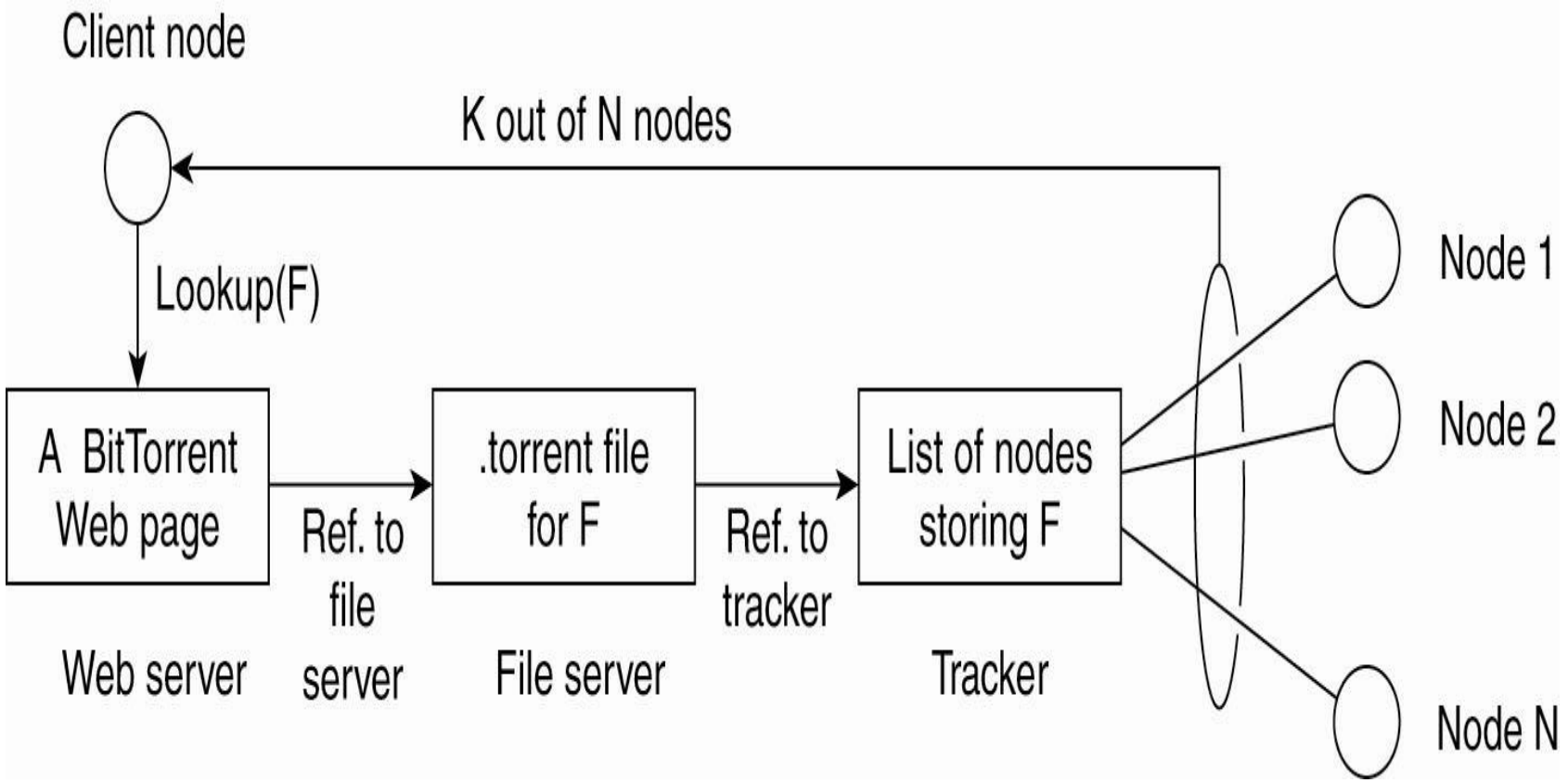
A hierarchical organization of nodes into a superpeer network

Hybrid Architectures
Edge-Server Systems



Viewing the Internet as consisting of a collection of edge servers

Collaborative Distributed Systems (1)

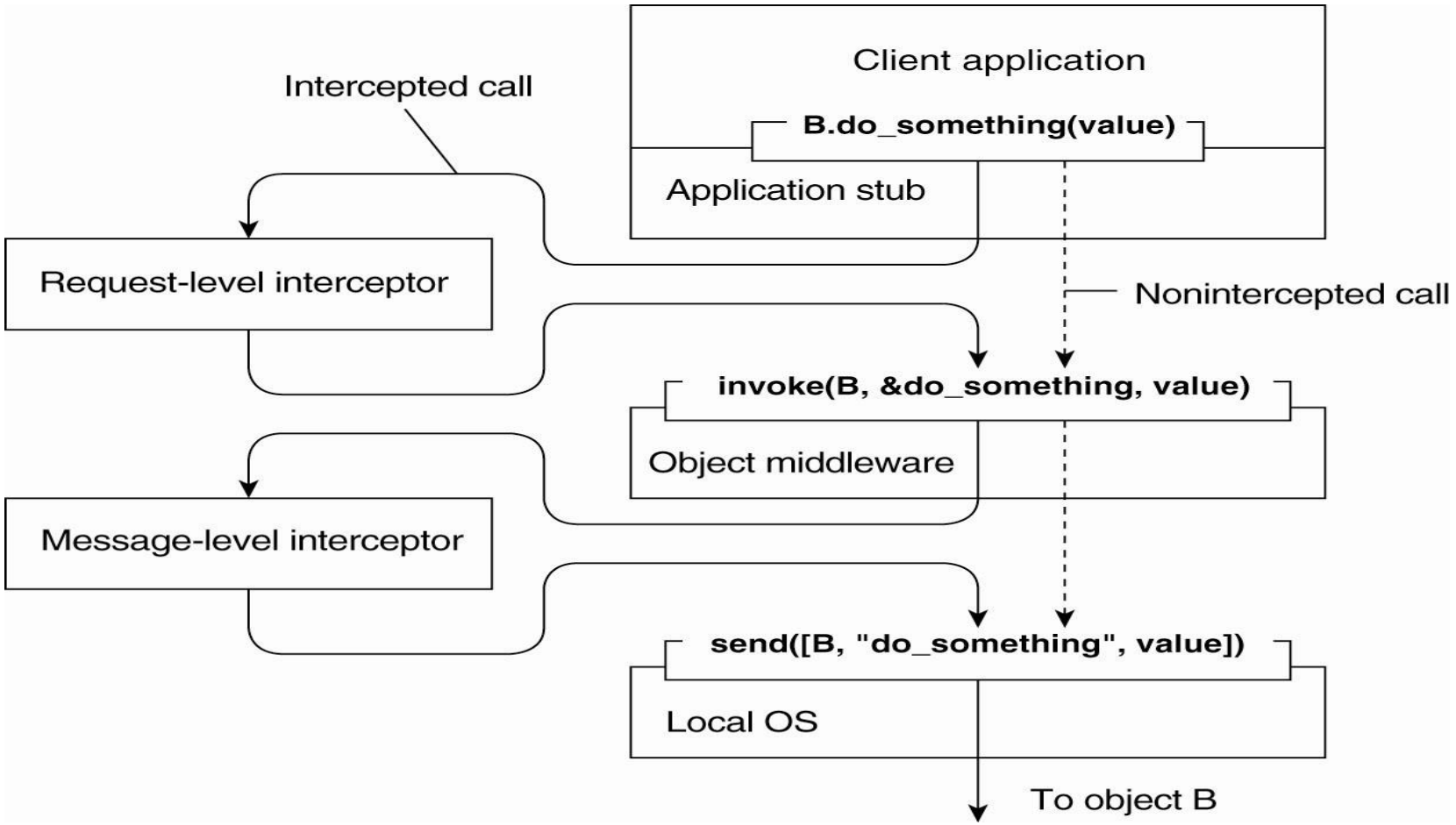


The principal working of BitTorrent [adapted with permission from Pouwelse et al. (2004)]

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

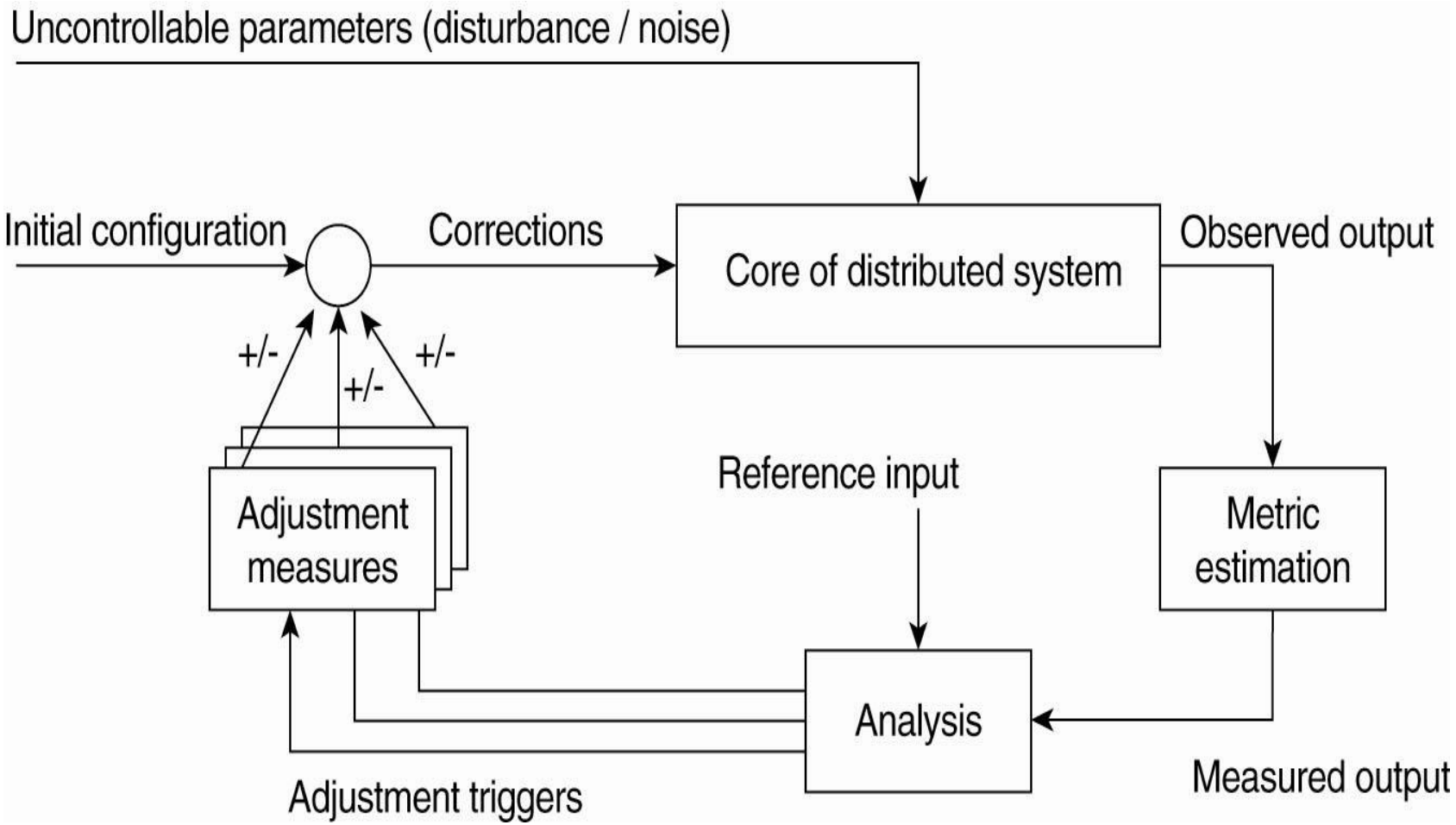


Using interceptors to handle remote-object invocations

General Approaches to Adaptive Software

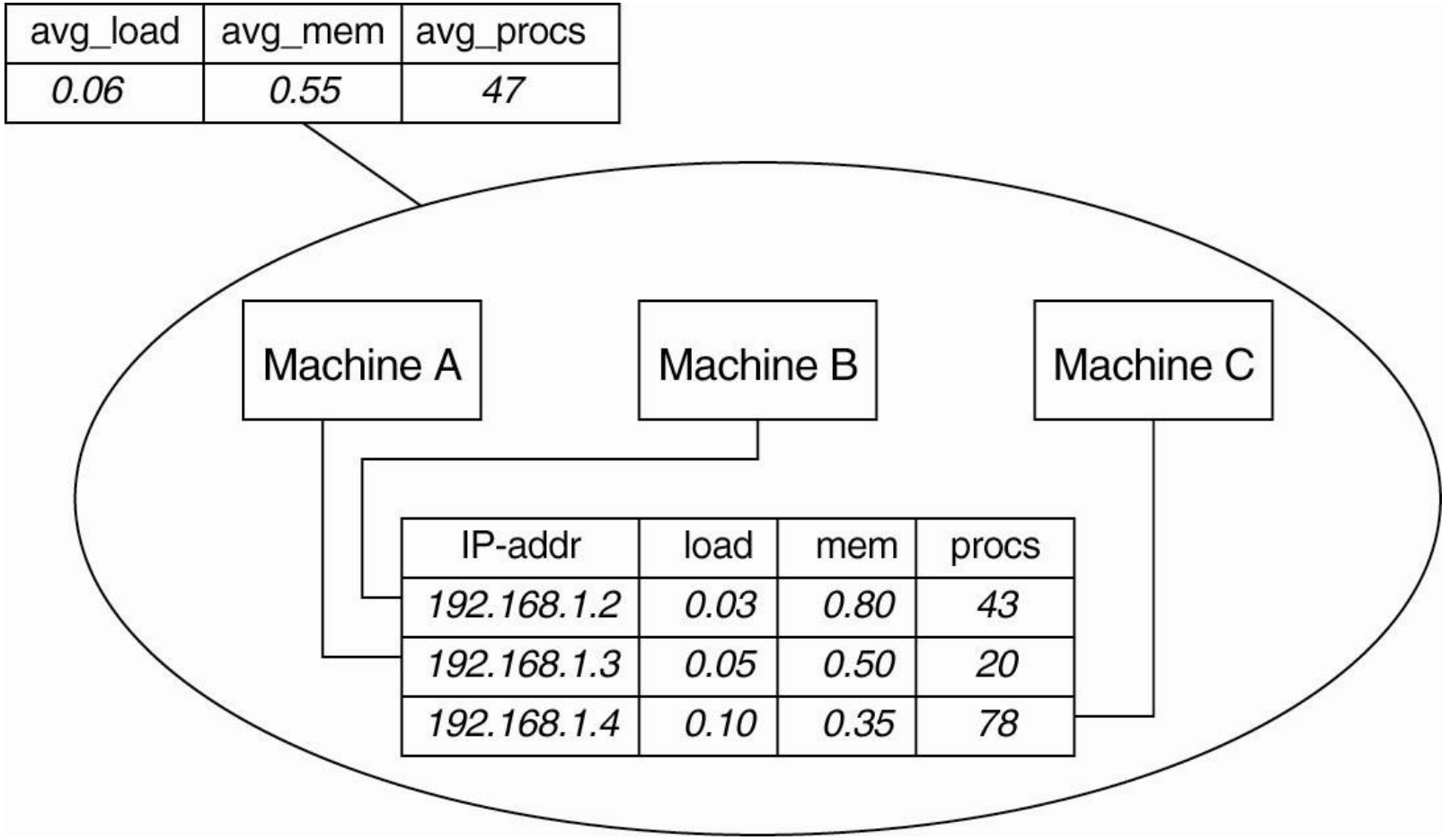
- Three basic approaches to adaptive software:
  1. Separation of concerns
  2. Computational reflection
  3. Component-based design

Self-Managing DSs
The Feedback Control Model



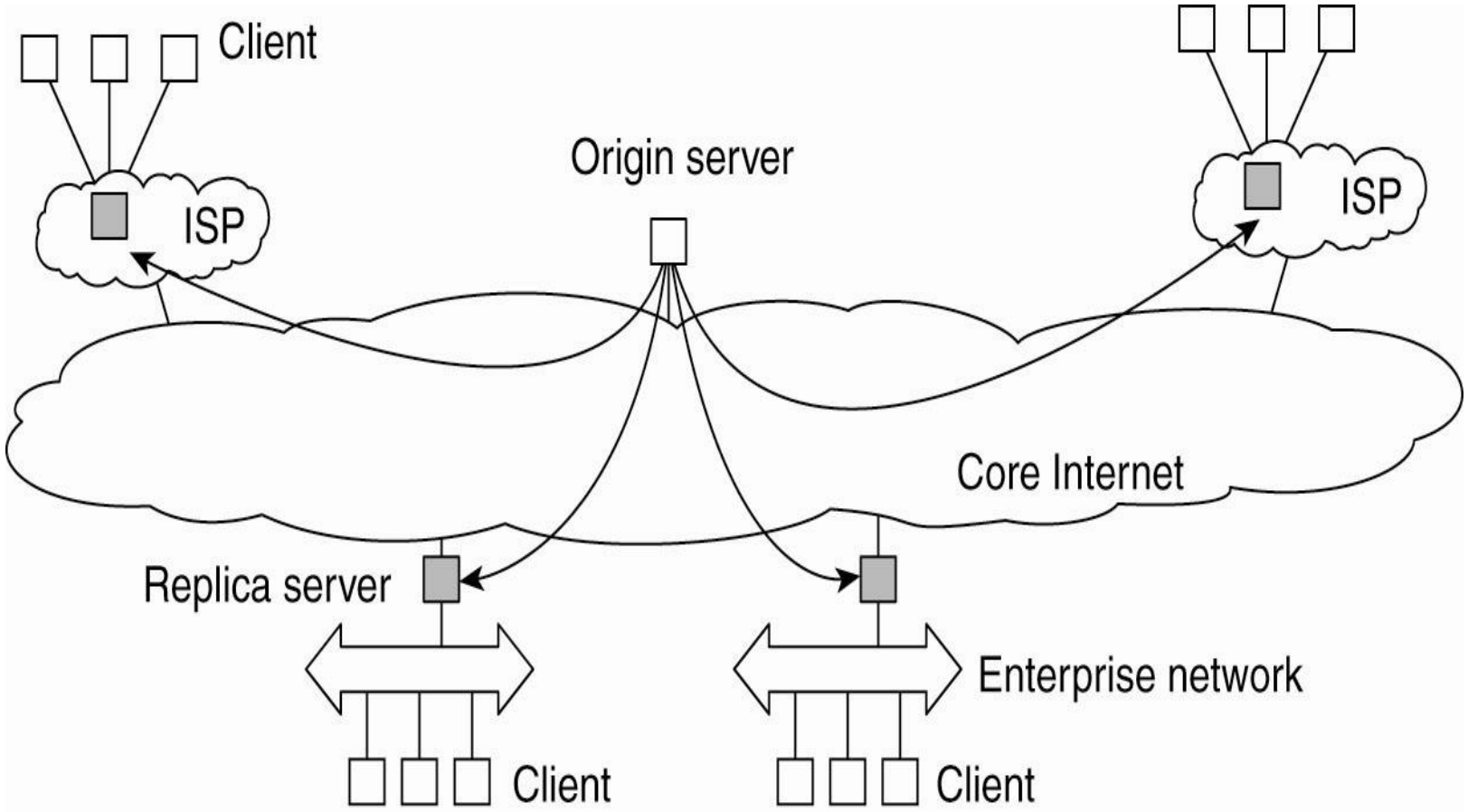
The logical organization of a feedback control system

Example: Systems Monitoring with Astrolabe



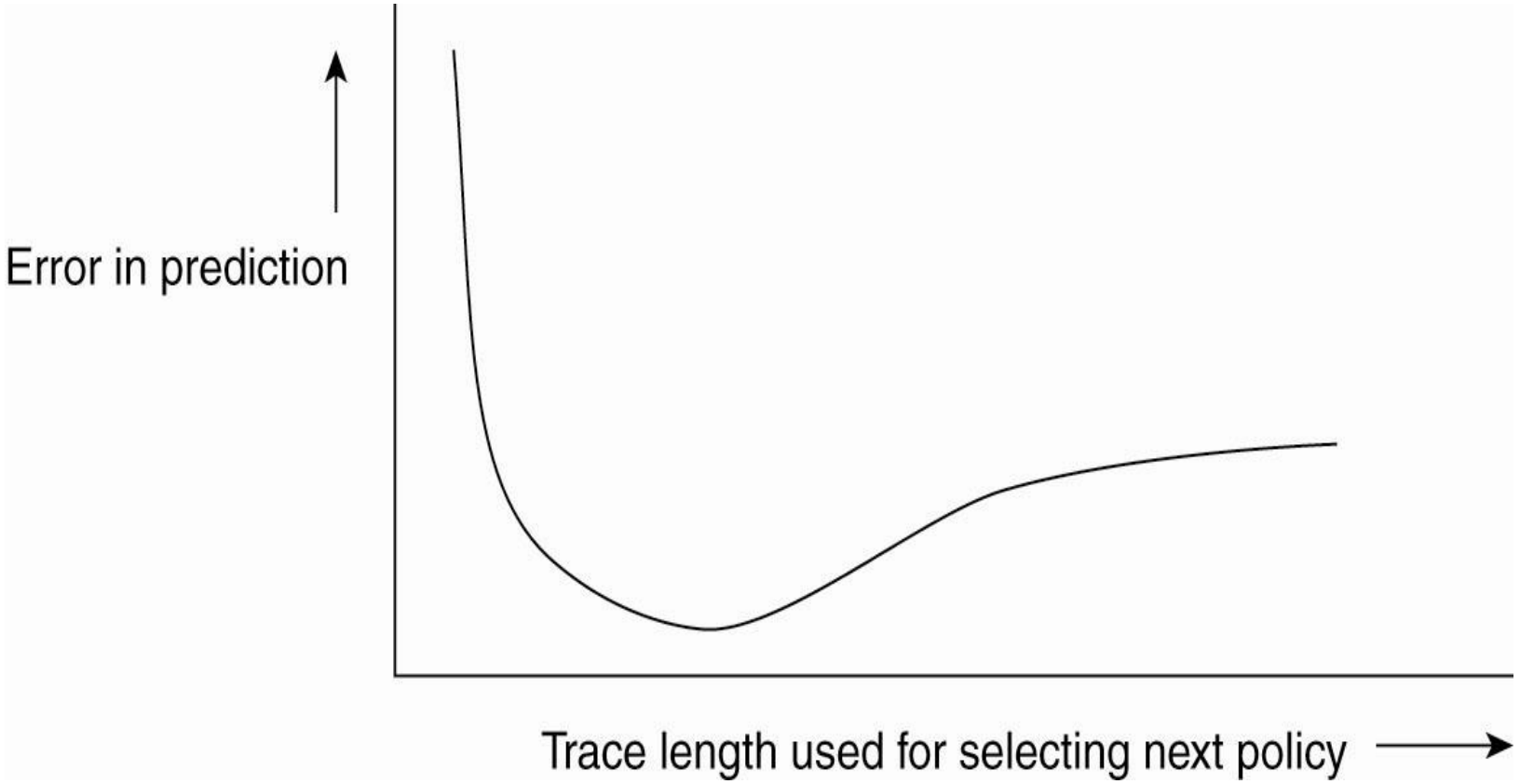
Data collection and information aggregation in Astrolabe

Example: Differentiating Replication Strategies in Globule (1)



The edge-server model assumed by Globule

Example: Differentiating Replication Strategies in Globule (2)



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 crashed node.
  - Re-establish all the bindings that were previously terminated.

Thank You !!!