# Architectural Models of distributed systems

## Agenda

Physical models

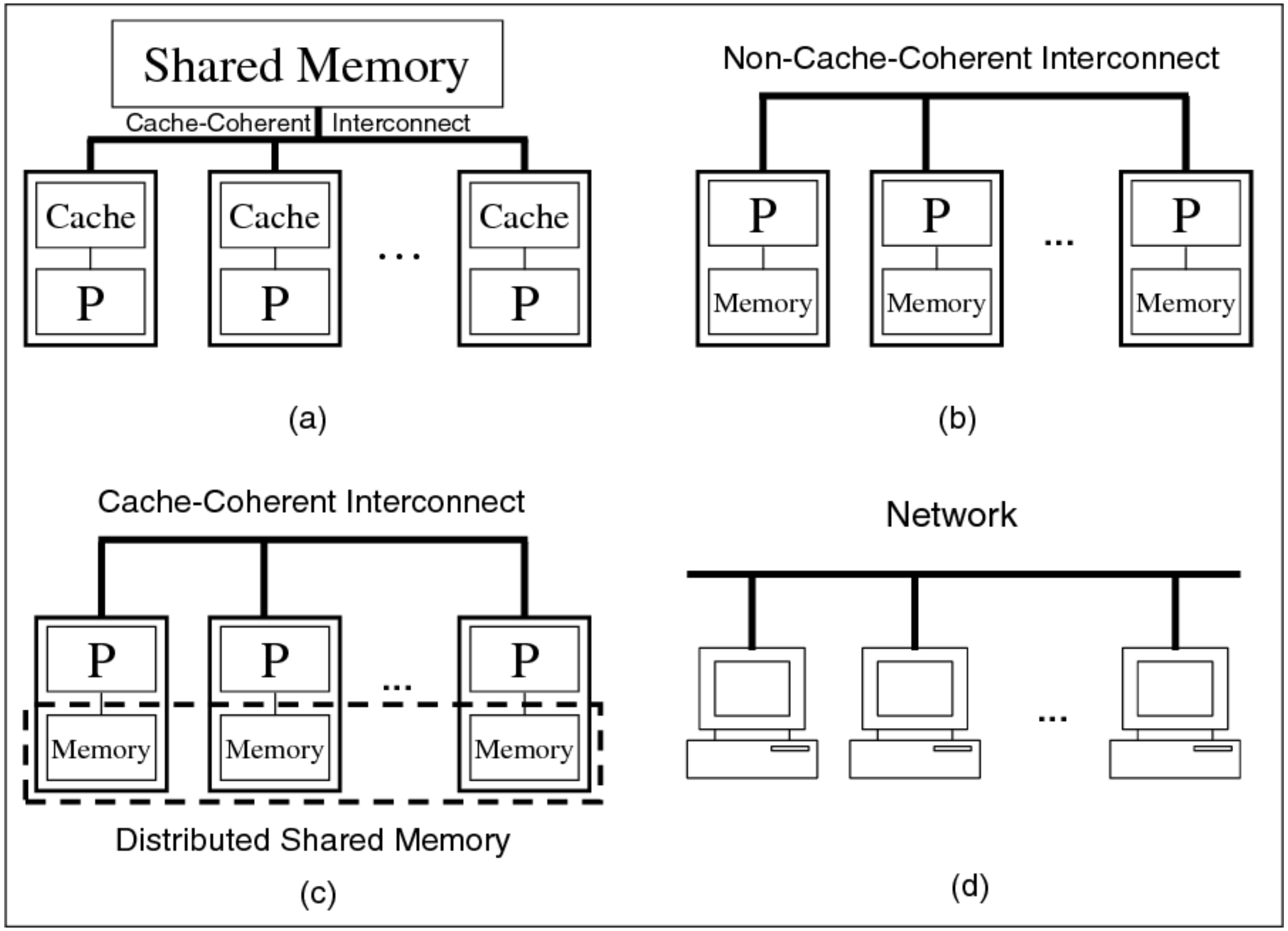
Architectural models (eg. for middleware)

Fundamental models (for distributed algorithms)

## Models of distributed systems

1. **Physical models** consider the types of computers and devices that constitute a system and their interconnectivity, without details of specific technologies. Eg. Shared memory or no shared memory
2. **Architectural models** describe a system in terms of the computational and communication tasks performed by its computational elements; the computational elements being individual computers or aggregates of them supported by appropriate network interconnections. Client-server and peer-to-peer are two of the most commonly used forms of architectural model for distributed systems.
3. **Fundamental models** take an abstract perspective in order to describe solutions to issues faced by most distributed systems: eg. Models for clock synchronization

## Shared memory vs distributed memory


Shared Memory
Cache-Coherent Interconnect
Cache
P
Cache
P
Cache
P
(a)
Non-Cache-Coherent Interconnect
P
Memory
P
Memory
P
Memory
(b)
Cache-Coherent Interconnect
P
Memory
P
Memory
P
Memory
Distributed Shared Memory
(c)
Network
(d)

The machine in (a) has physically shared memory, whereas the others have distributed memory. However, the memory in (c) is accessible to all processors

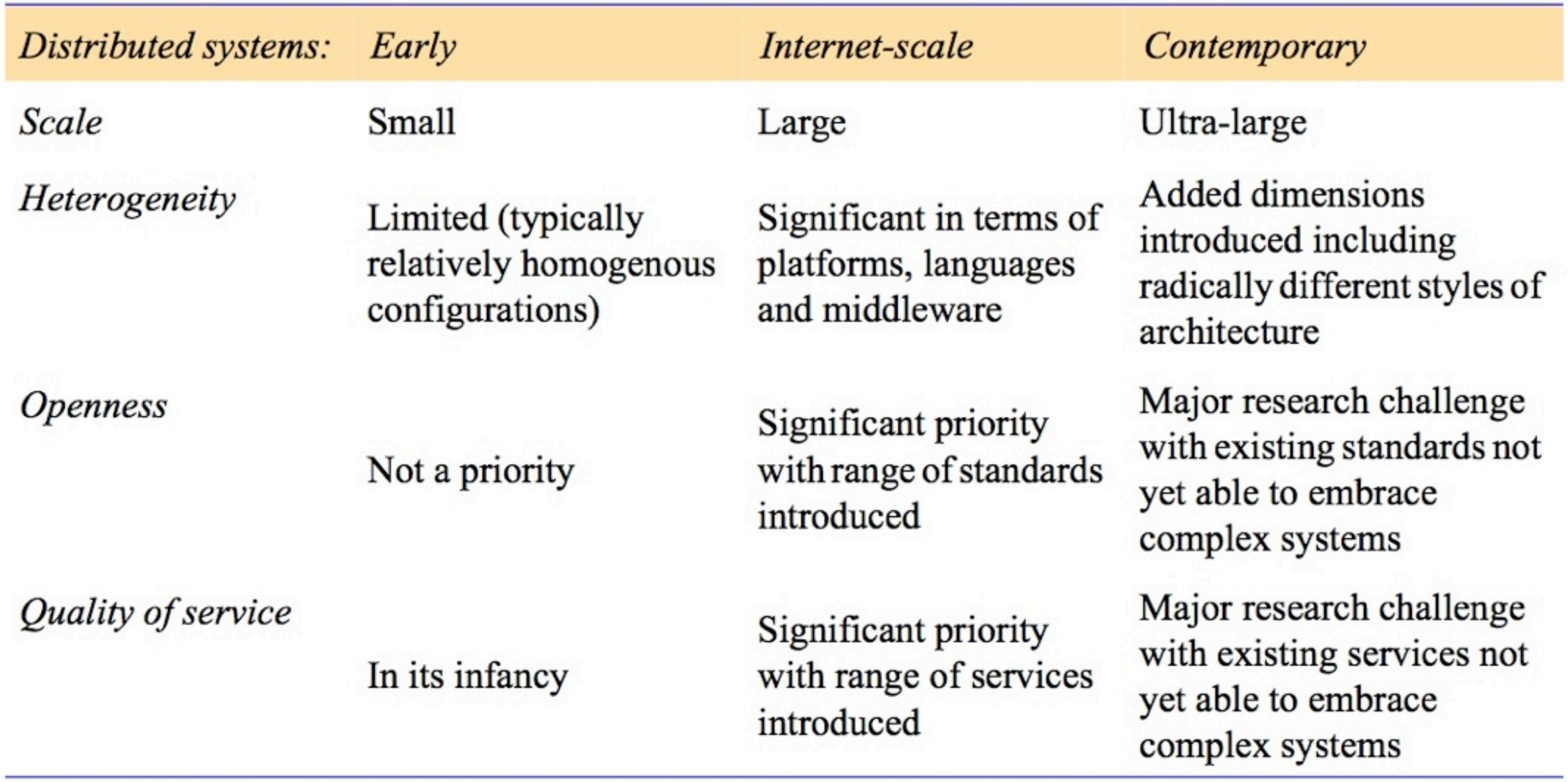
## Problematic issues in distributed systems

- There is no global time in a distributed system: the clocks on different computers do not give the same time as one another.
- All communication between processes is by means of messages. Message communication over a computer network is affected by delays, can suffer from a variety of failures and is vulnerable to security attacks.

These issues are addressed by three models:

1. The **interaction** model deals with performance and with the difficulty of setting time limits to some operations in a distributed system, for example for message delivery.
2. The **failure** model attempts to give a precise specification of the faults that can be exhibited by processes and communication channels. It defines reliable communication and correct processes.
3. The **security** model discusses the possible threats to processes and communication channels. It introduces the concept of a secure channel, which is secure against those threats.

## Generations of distributed systems



| *Distributed systems:* | *Early* | *Internet-scale* | *Contemporary* |
|---|---|---|---|
| *Scale* | Small | Large | Ultra-large |
| *Heterogeneity* | Limited (typically relatively homogenous configurations) | Significant in terms of platforms, languages and middleware | Added dimensions introduced including radically different styles of architecture |
| *Openness* | Not a priority | Significant priority with range of standards introduced | Major research challenge with existing standards not yet able to embrace complex systems |
| *Quality of service* | In its infancy | Significant priority with range of services introduced | Major research challenge with existing services not yet able to embrace complex systems |

## Coordinable entities in distributed systems

- What entities do they communicate in a distributed system?
- How do they communicate, or, more specifically, what communication paradigm is used?
- What (potentially changing) roles and responsibilities do they have in the overall architecture?
- How are they mapped on to the physical distributed infrastructure (what is their deployment)?

## Architectural styles

A **style** is formulated in terms of (replaceable) components with well-defined interfaces

- the way that components are connected to each other
- the data exchanged between components
- how these components and connectors are jointly configured into a system.

**Connector**

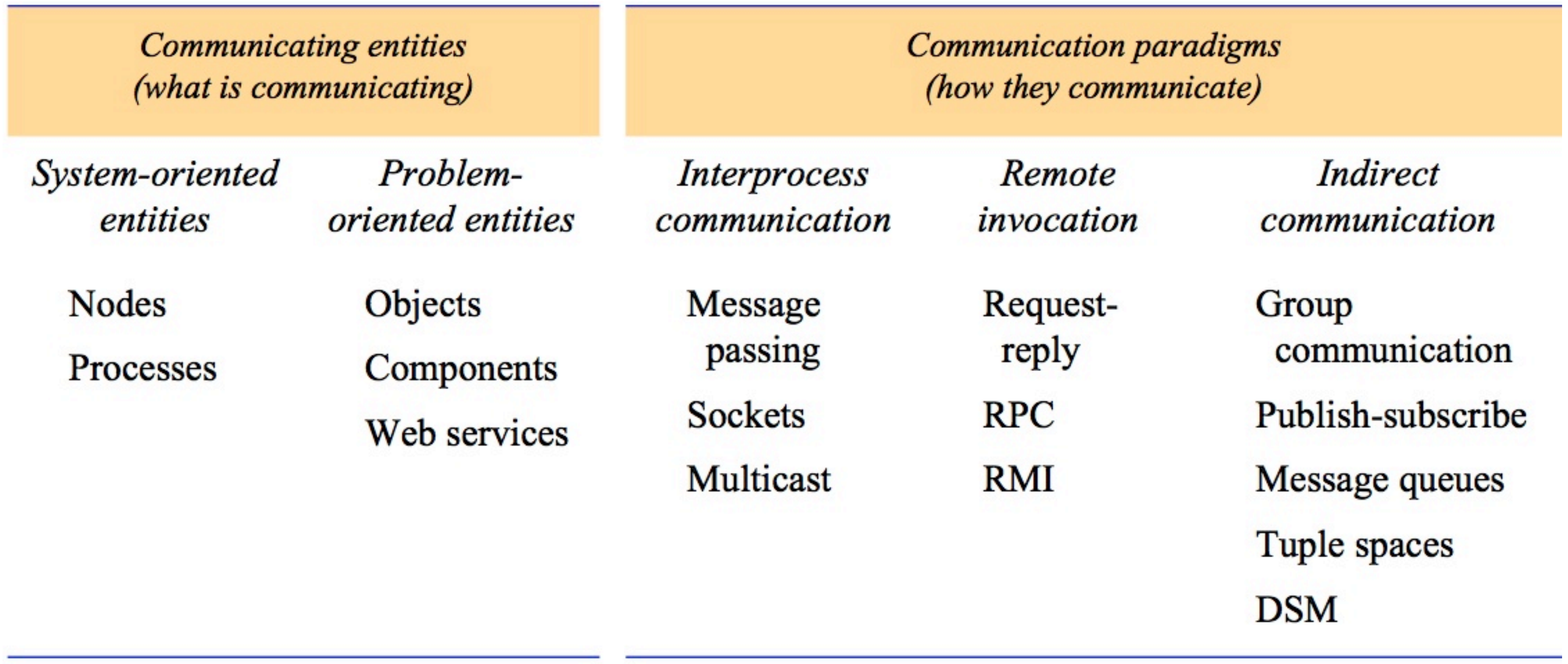
A (software) mechanism that mediates communication, coordination, or cooperation among components.

Example : facilities for (remote) procedure call, messaging, or streaming.

## Communication paradigms

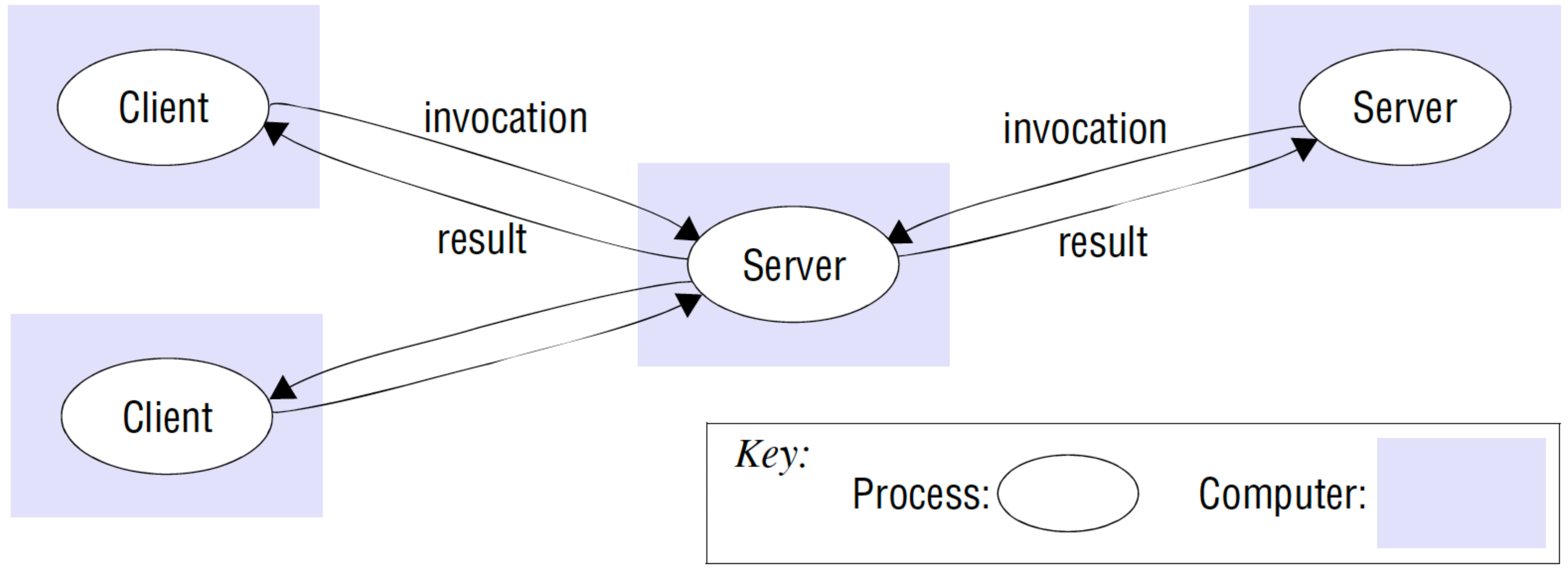
- interprocess communication
- remote invocation
- indirect communication

## Communicating entities and communication paradigms



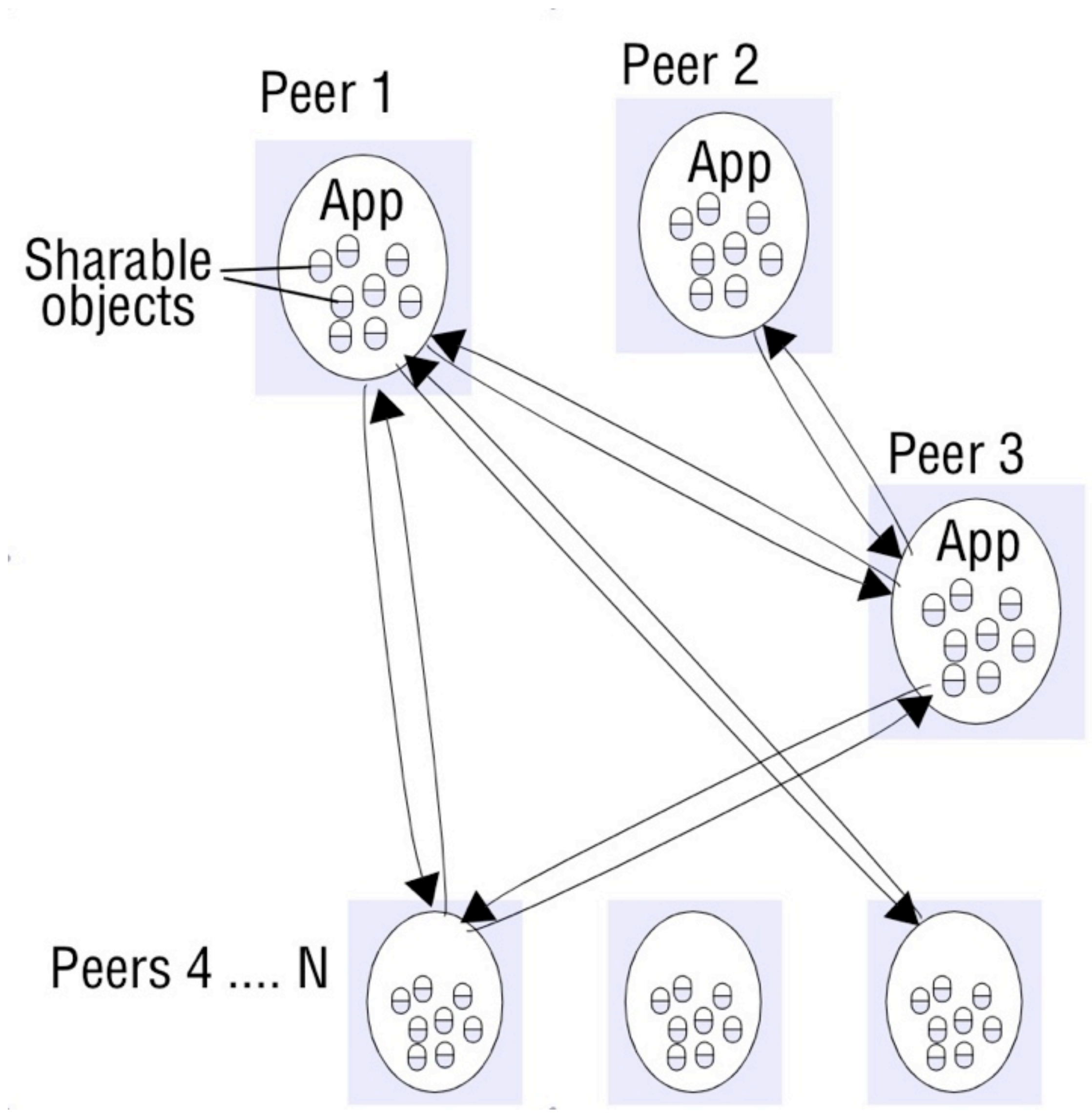
| *Communicating entities (what is communicating)* | | *Communication paradigms (how they communicate)* | | |
|---|---|---|---|---|
| *System-oriented entities* | *Problem-oriented entities* | *Interprocess communication* | *Remote invocation* | *Indirect communication* |
| Nodes | Objects | Message passing | Request-reply | Group communication |
| Processes | Components | Sockets | RPC | Publish-subscribe |
| | Web services | Multicast | RMI | Message queues |
| | | | | Tuple spaces |
| | | | | DSM |

## Clients invoke individual servers

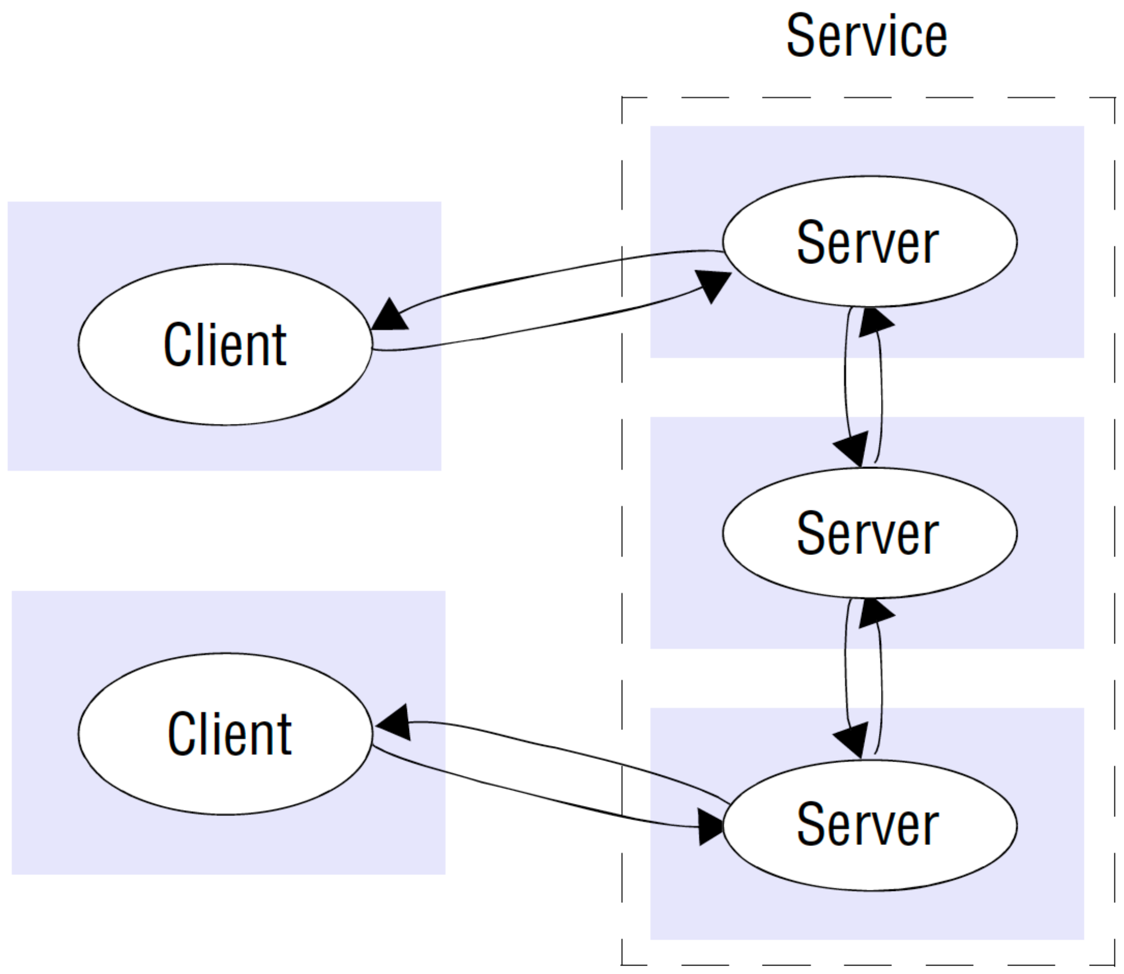


a web server is often a client of a local file server that manages the files in which the web pages are stored. Web servers and most other Internet services are clients of the DNS service, which translates Internet domain names to network addresses

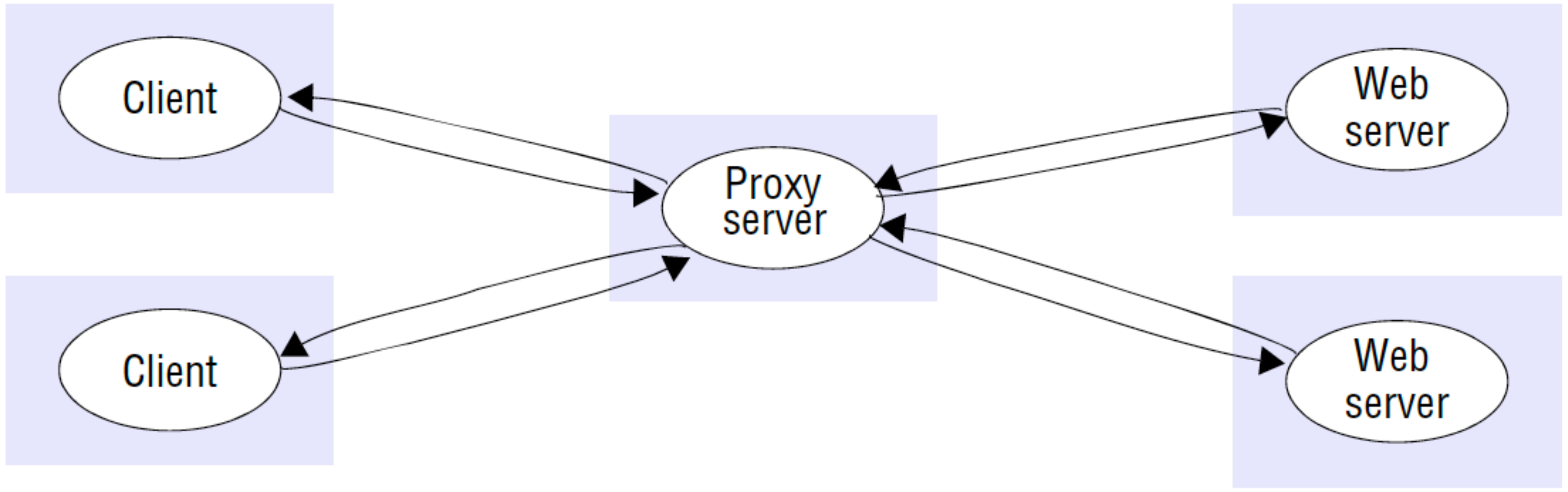
## Peer-to-peer architecture



## A service provided by multiple servers



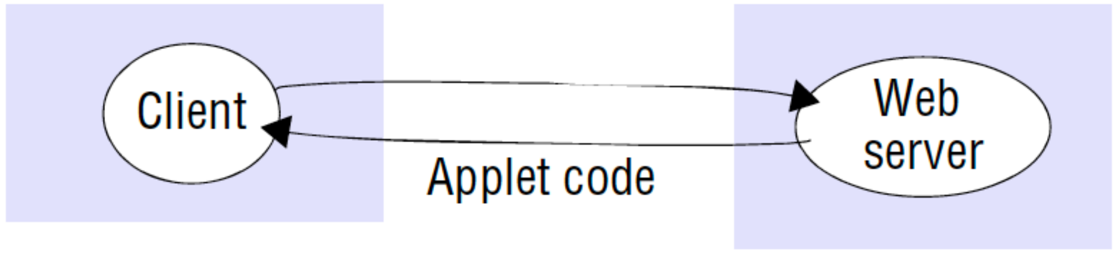
## Web proxy server



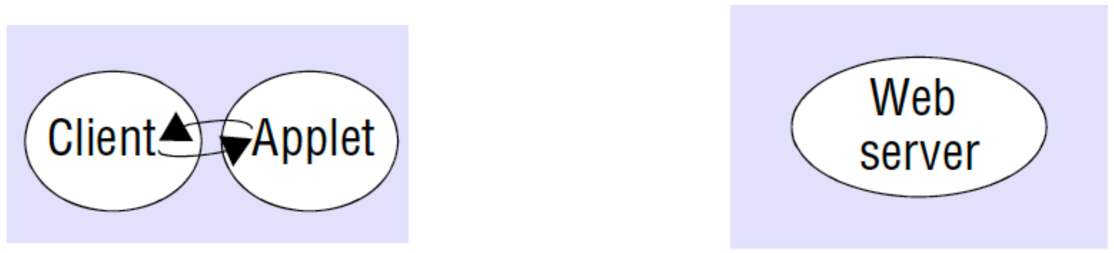
Web proxy servers provide a shared cache of web resources for the client machines at a site or across several sites.
The purpose of proxy servers is to increase the availability and performance of the service by reducing the load on both the network and the servers.

## Web applets

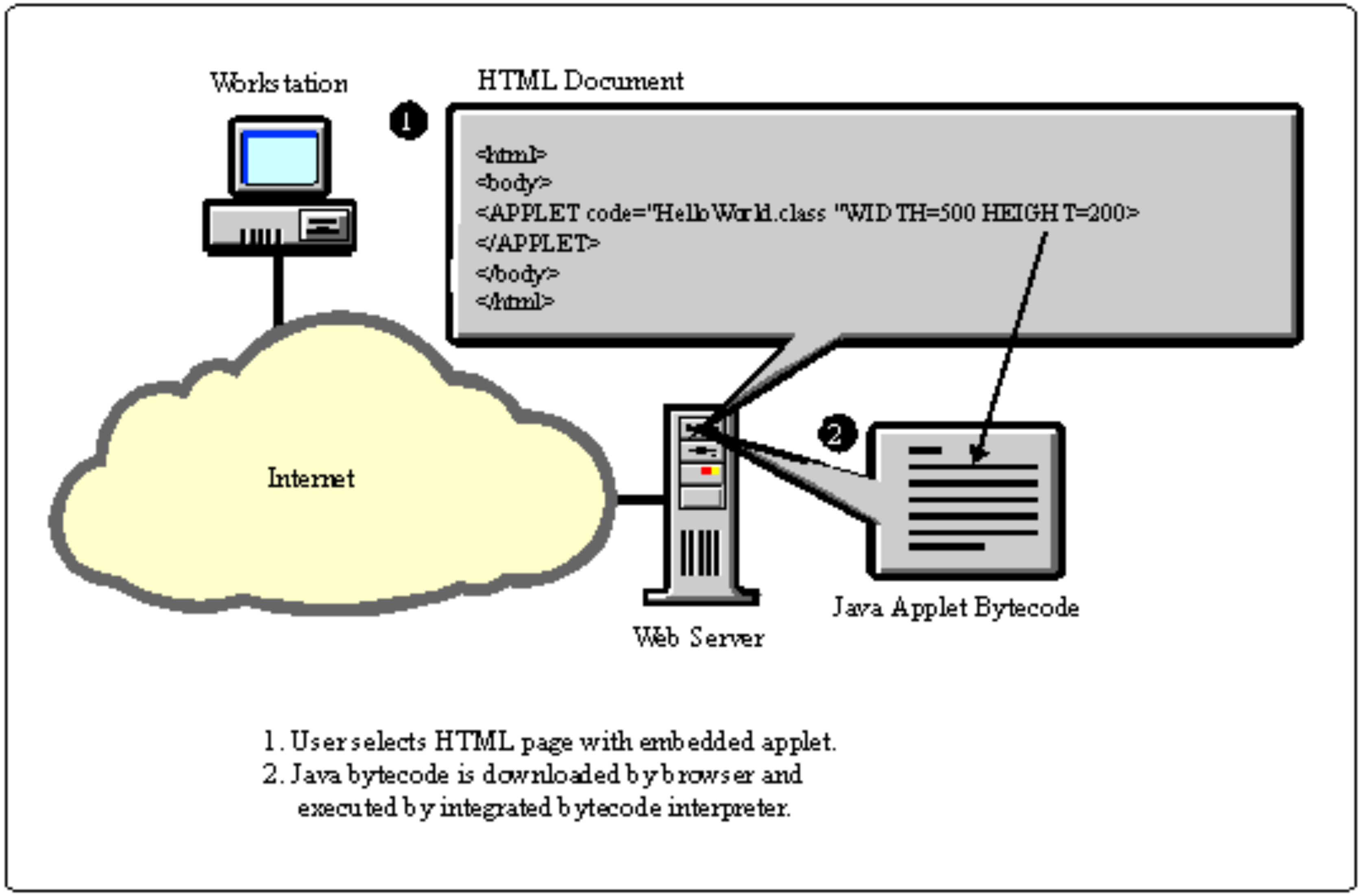
a) client request results in the downloading of applet code



b) client interacts with the applet



## Anatomy of a Java applet

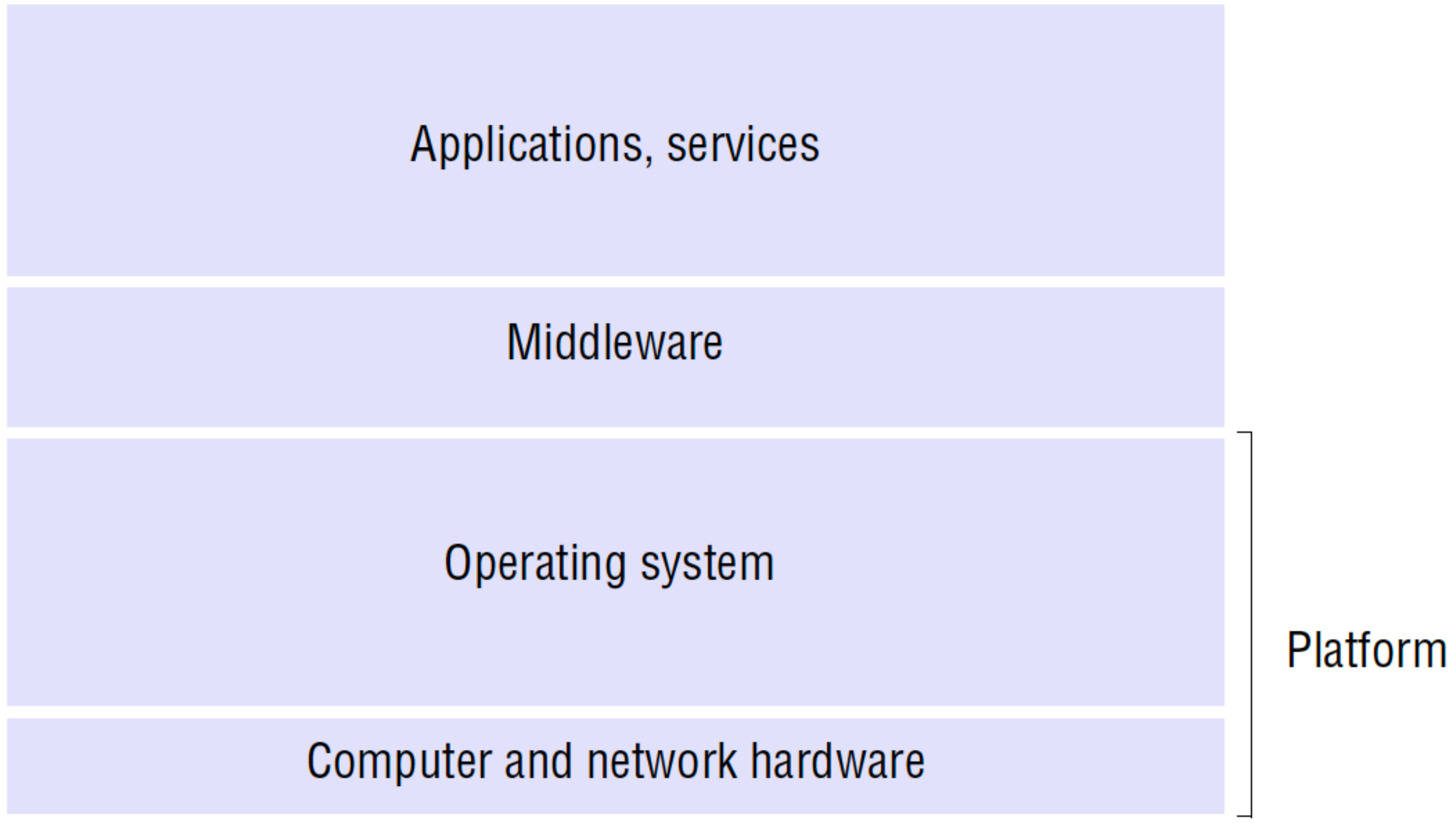


```
<html>
<body>
<APPLET code="HelloWorld.class "WIDTH=500 HEIGHT=200>
</APPLET>
</body>
</html>
```

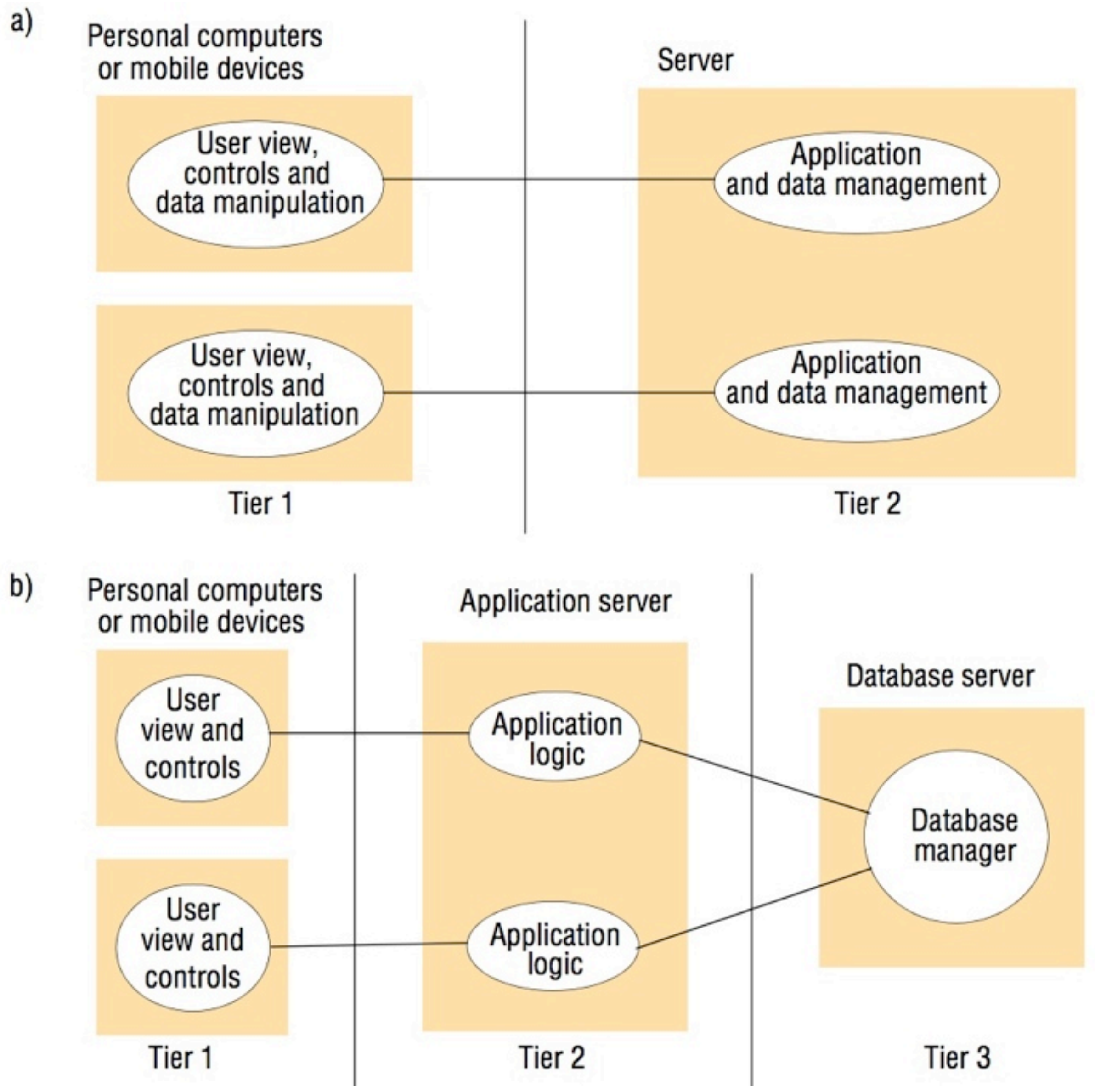
1. User selects HTML page with embedded applet.
2. Java bytecode is downloaded by browser and executed by integrated bytecode interpreter.

https://support.novell.com/techcenter/articles/dnd19960801.html#l12jadm

## Software and hardware service layers in distributed systems



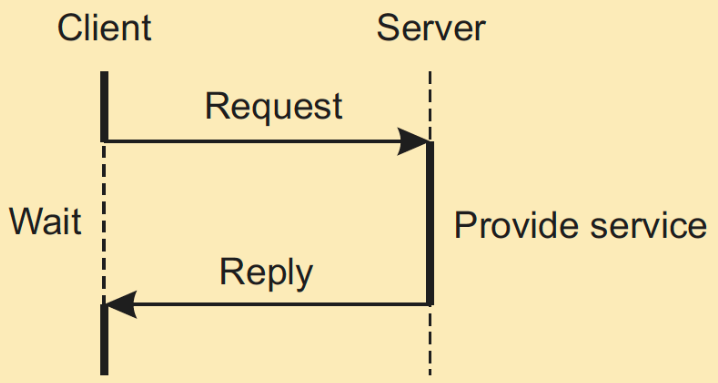
## Two-tier and three-tier architectures


a)
Personal computers or mobile devices
Server
User view, controls and data manipulation
Application and data management
User view, controls and data manipulation
Application and data management
Tier 1
Tier 2
b)
Personal computers or mobile devices
Application server
Database server
User view and controls
Application logic
Database manager
User view and controls
Application logic
Tier 1
Tier 2
Tier 3

## Basic client-server style

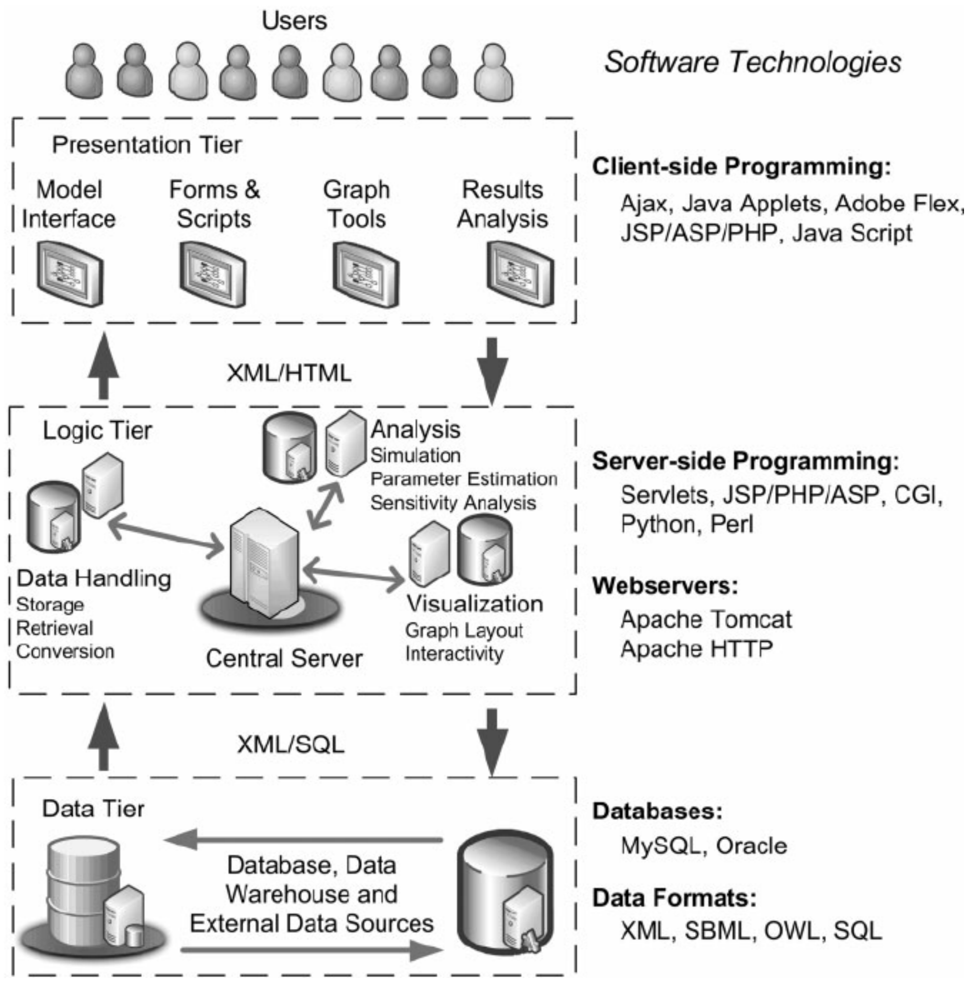
Characteristics:

- There are processes offering services (servers)
- There are processes that use services (clients)
- Clients and servers can be on different machines
- Clients follow request/reply model with respect to using services



## 3- tier architecture with AJAX

The 3-tier architecture is used in most web applications; AJAX is used client-side
It is the ‘glue’ that supports the construction of such applications; it provides a communication mechanism enabling front-end components running in a browser to issue requests and receive results from back-end components running on a server



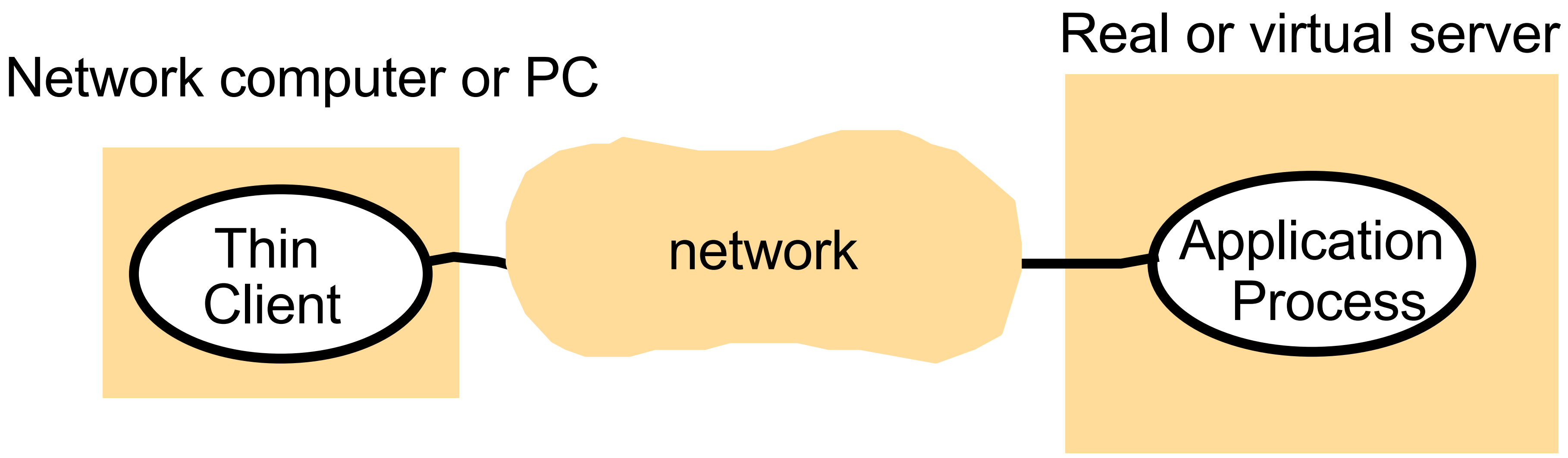
## AJAX example: soccer score updates (using a JavaScript library)

```
new Ajax.Request('scores.php?
                 game=Arsenal:Liverpool’,
                {onSuccess: updateScore});
function updateScore(request) {
.....
```

(request contains the state of the Ajax request including the returned result. The result is parsed to obtain some text giving the score, which is used to update the relevant portion of the current page.)

```
.....
}
```

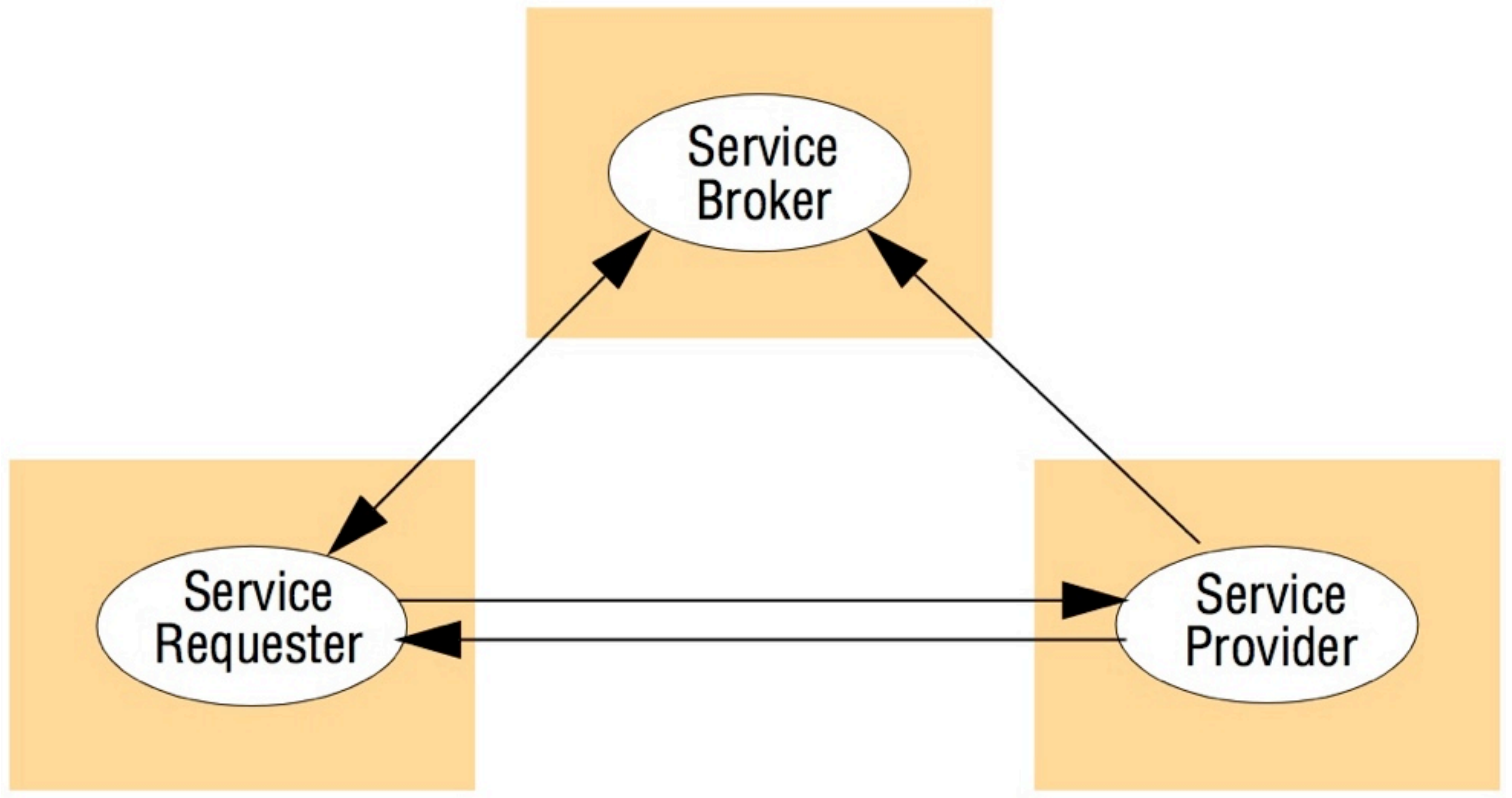
## Thin clients and compute servers



The trend in distributed computing is towards moving complexity away from the end-user device towards services in the Internet

This is most apparent in the move towards cloud computing

## The web service architectural pattern for middleware



This middleware pattern is replicated in many areas of distributed systems, for example with the registry in Java RMI and the naming service in CORBA

## Categories of middleware

| Category | subcategory | Example systems |
|---|---|---|
| Distributed objects | Standard | RM-ODP |
| | Platform | CORBA |
| | Platform | JavaRMI |
| Distributed components | Lightweight components | Fractal |
| | Lightweight components | OpenCOM |
| | Application servers | SUN EJB |
| | Application servers | CORBA component model |
| | Application servers | JBoss |
| Publish-subscribe systems | | CORBA event service |
| | | Scribe |
| | | JMS |
| Message queues | | Websphere MQ |
| | | JMS |
| Web services | Web services | Apache Axis |
| | Grid services | The Globus Toolkit |
| Peer-to-peer | Routing overlays | Pastry |
| | Routing overlays | Tapestry |
| | Application-specific | Sqirrel |
| | Application-specific | Oceanstore |
| | Application-specific | Ivy |
| | Application-specific | Gnutella |

## RESTful architectures

RESTful architectures view a distributed system as a collection of resources, individually managed by components. Resources may be added, removed, retrieved, and modified by (remote) applications.

1 Resources are identified through a single naming scheme

2 All services offer the same interface

3 Messages sent to or from a service are fully self-described

4 After executing an operation at a service, that component forgets everything about the caller

Basic operations

| Operation | Description |
|---|---|
| PUT | Create a new resource |
| GET | Retrieve the state of a resource in some representation |
| DELETE | Delete a resource |
| POST | Modify a resource transferring to a new state |

## Example: Amazon Simple Storage Service S3

Objects (i.e. files) are placed into buckets (i.e. directories). Buckets cannot be placed into buckets. Operations on `ObjectName` in bucket `BucketName` require the following identifier:

`http://BucketName.s3.amazonaws.com/ObjectName`

All operations are carried out by sending HTTP requests:

- Create a bucket/object: `PUT` , along with the URI
- Listing objects: `GET` on a bucket name
- Reading an object: `GET` on a full URI

https://docs.aws.amazon.com/en_us/AmazonS3/latest/dev/UsingBucket.html

## Ordering events in a distributed system

The execution of activities in a distributed system can be described in terms of events and their ordering, despite the lack of accurate clocks

We are often interested in knowing whether an event (sending or receiving a message) at one process occurred before, after or concurrently with another event at another process

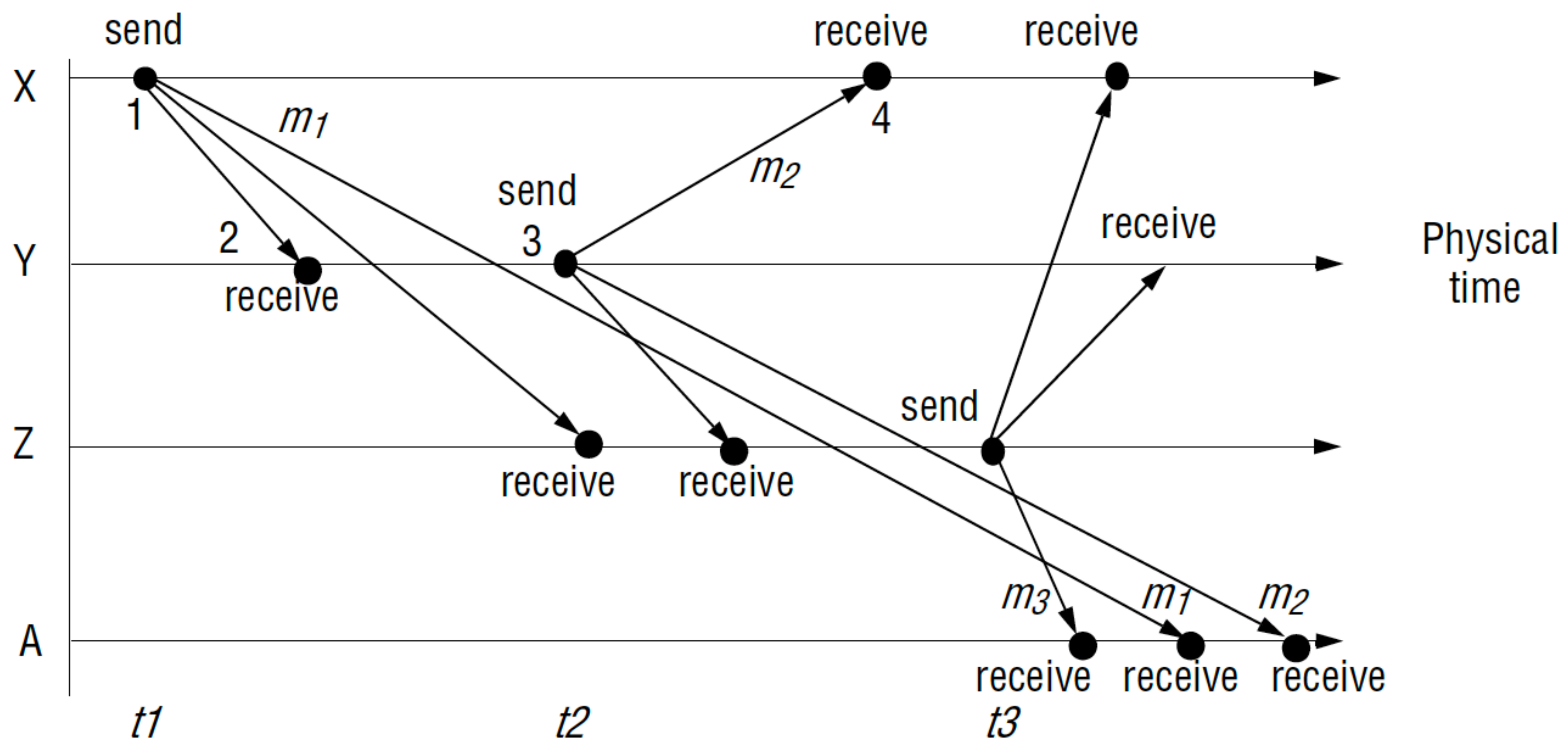
Example: consider the following set of exchanges between a group of email users, X, Y, Z and A, on a mailing list:

1. User X sends a message with the subject «Meeting.»
2. Users Y and Z reply by sending a message with the subject «Re: Meeting.»

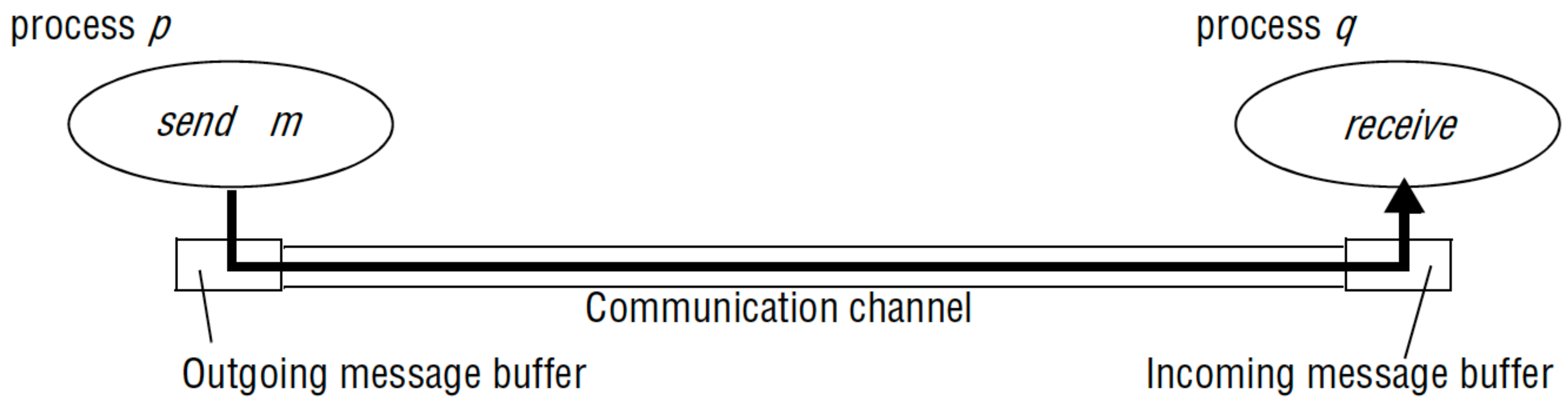
In real time, X’s message is sent first, and Y reads it and replies; Z then reads both X’s message and Y’s reply and sends another reply, which references both X’s and Y’s messages.

But due to the independent delays in message delivery, the messages may be delivered in a wrong order  (see next slide)

## Real-time ordering of events



## Processes and channels



A communication channel produces an **omission failure** if it does not transport a message from p 's outgoing message buffer to q 's incoming message buffer. This is known as 'dropping messages' and is generally caused by lack of buffer space at the receiver or at an intervening gateway, or by a network transmission error, detected by a checksum carried with the message data

## Omission and arbitrary failures

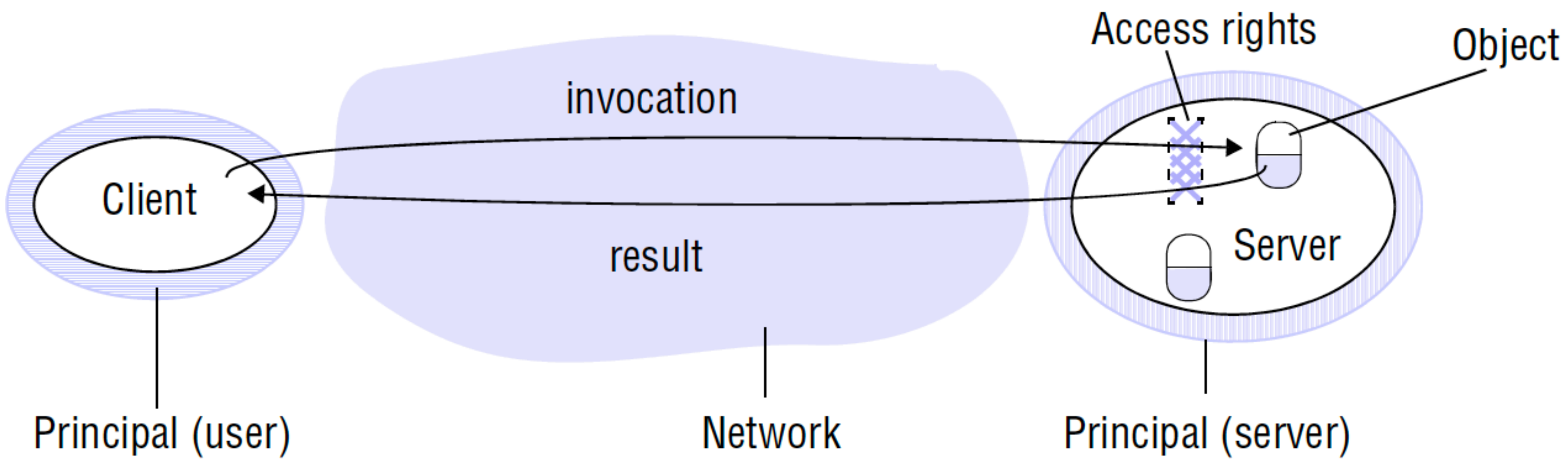
| *Class of failure* | *Affects* | *Description* |
|---|---|---|
| Fail-stop | Process | Process halts and remains halted. Other processes may detect this state. |
| Crash | Process | Process halts and remains halted. Other processes may not be able to detect this state. |
| Omission | Channel | A message inserted in an outgoing message buffer never arrives at the other end’s incoming message buffer. |
| Send-omission | Process | A process completes a *send*, but the message is not put in its outgoing message buffer. |
| Receive-omission | Process | A message is put in a process’s incoming message buffer, but that process does not receive it. |
| Arbitrary (Byzantine) | Process or channel | Process/channel exhibits arbitrary behaviour: it may send/transmit arbitrary messages at arbitrary times, commit omissions; a process may stop or take an incorrect step. |

## Timing failures

| *Class of Failure* | *Affects* | *Description* |
|---|---|---|
| Clock | Process | Process’s local clock exceeds the bounds on its rate of drift from real time. |
| Performance | Process | Process exceeds the bounds on the interval between two steps. |
| Performance | Channel | A message’s transmission takes longer than the stated bound. |

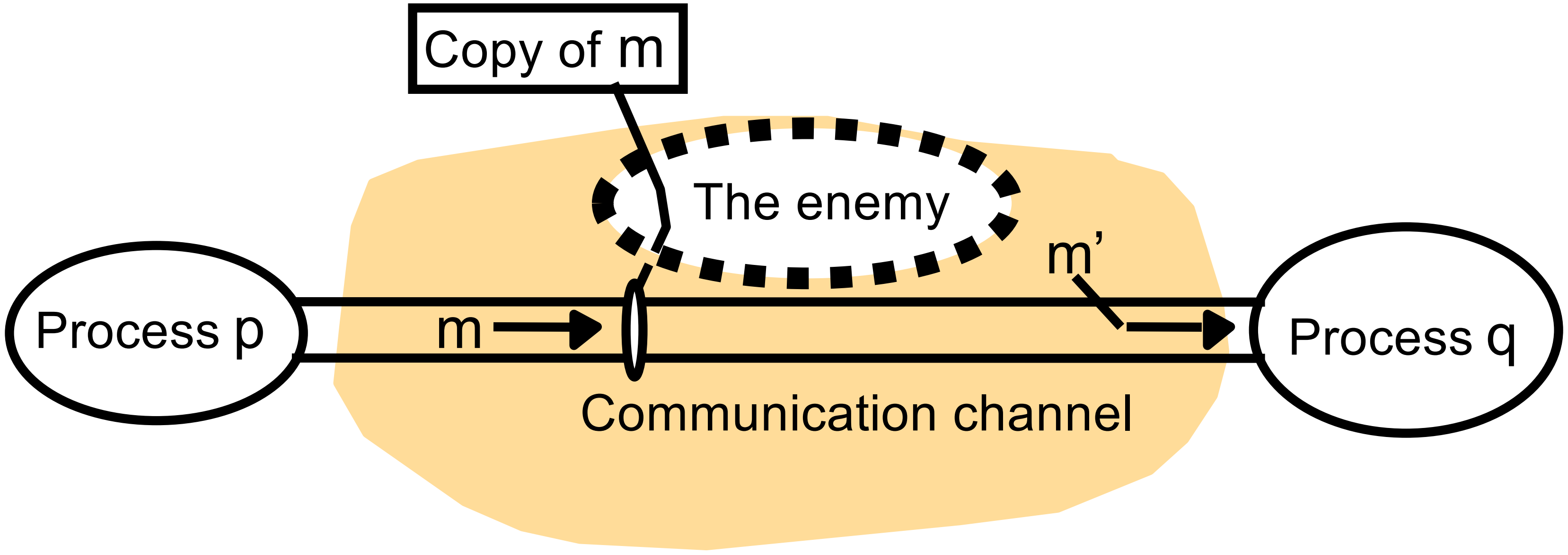
Timing failures are applicable in synchronous distributed systems where time limits are set on process execution time, message delivery time and clock drift rate.
Any one of these failures may result in responses being unavailable to clients within a specified time interval.

## Objects and principals



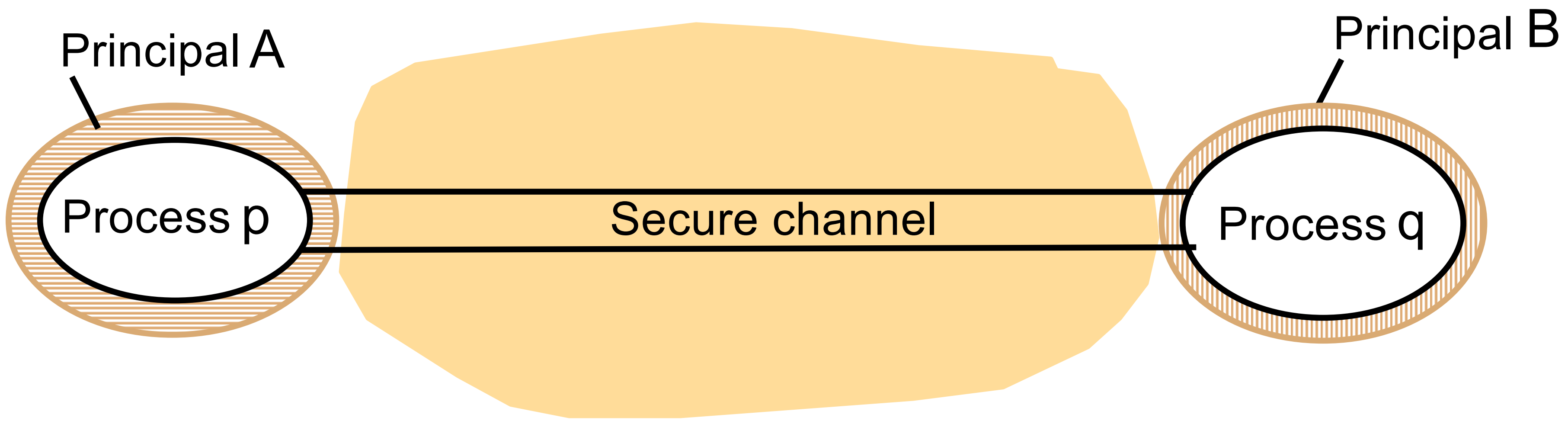
- A server manages a collection of objects on behalf of some users, who can run client programs that call the server to perform operations on the objects.
- The server carries out the operation specified in each call and answers the client.
- Objects are to be used in different ways by different users.
- For example, some objects may hold a user’s private data, such as their mailbox, and other objects may hold shared data such as web pages.
- To support this, access rights specify who is allowed to perform the operations of an object – for example, who is allowed to read or to write its state.

## The enemy



To model and study security threats, we postulate an enemy (adversary) that is capable of sending any message to any process and reading or copying any message sent on a channel between a pair of processes

## Secure channels



A secure channel (encrypted, authenticated) has the following properties:

- Each of the processes knows reliably the identity of the principal on whose behalf the other process is executing.
- A secure channel ensures the privacy and integrity (protection against tampering) of the data transmitted across it.
- Each message includes a physical or logical timestamp to prevent messages from being replayed or reordered

## Conclusions

Distributed systems can be studied via their architectural abstractions and models

Client server and peer to peer are the most important architectural patterns for distributed systems

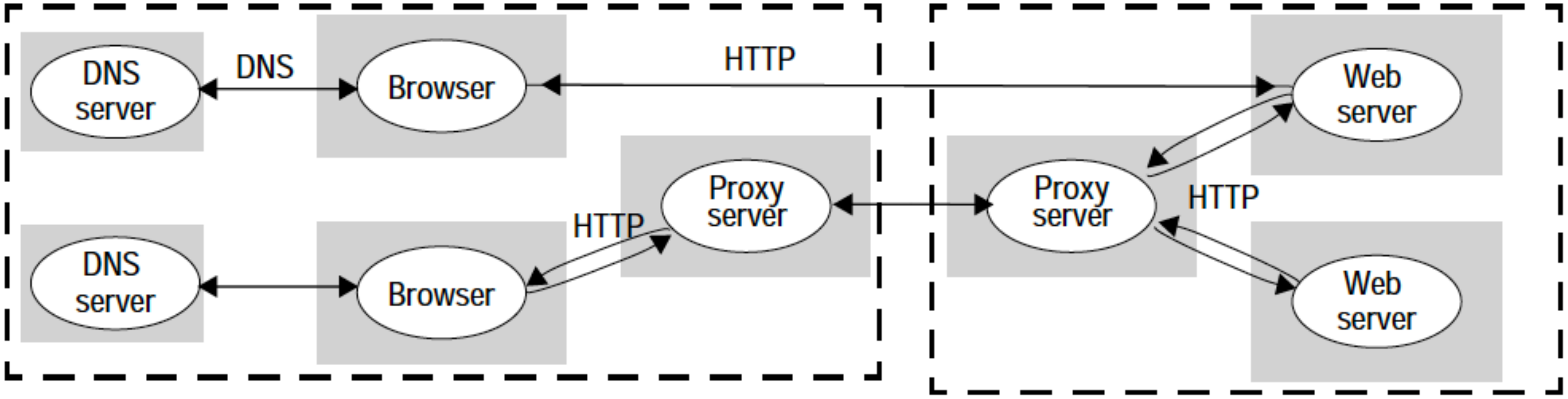
## Exercises

1. Describe the software architecture of the major Internet services for

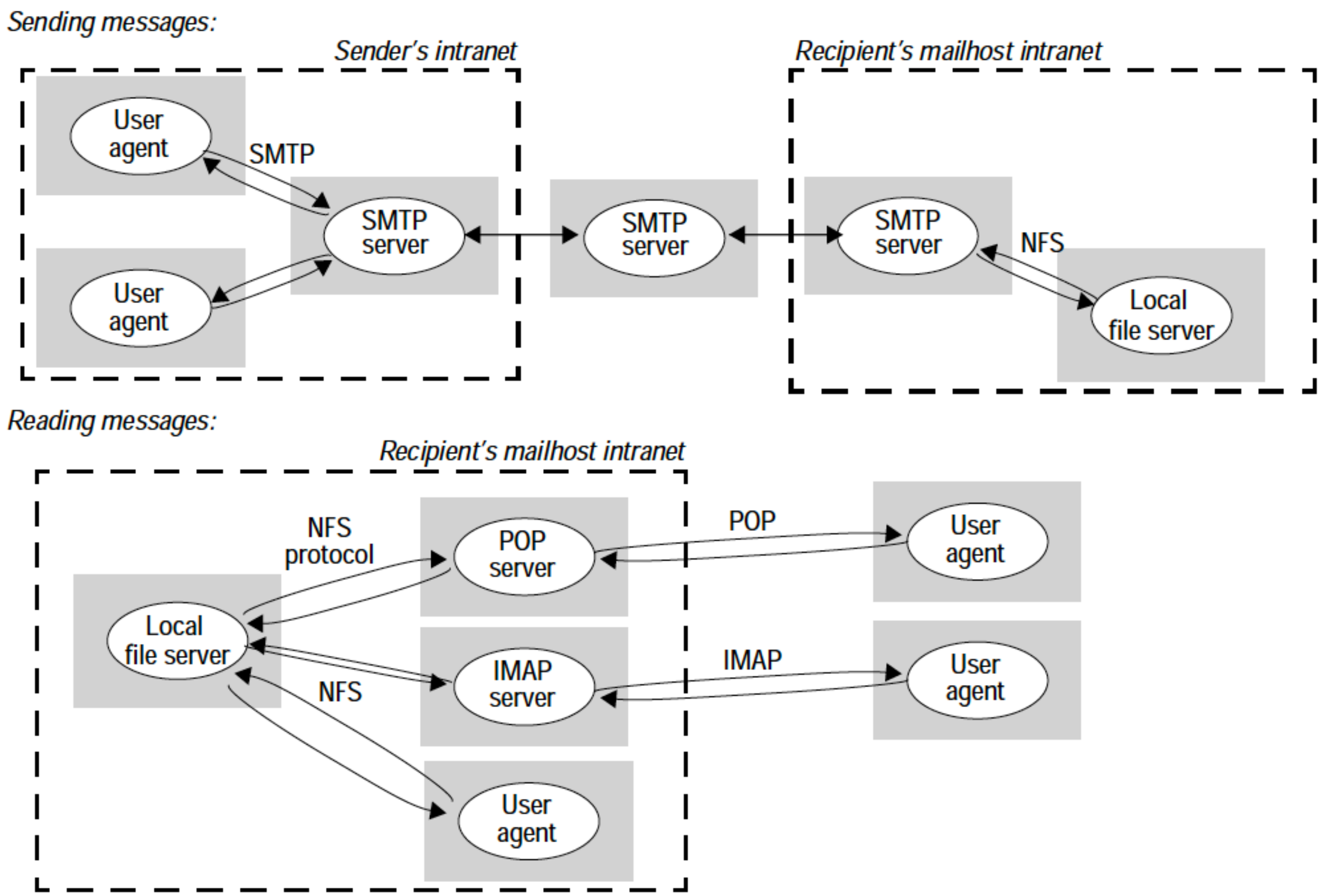
- Web,
- Mail
- news

2. Describe how servers cooperate in providing these services
3. How do the services exploit the partitioning and/or replication (or caching) of data amongst servers?

## Web

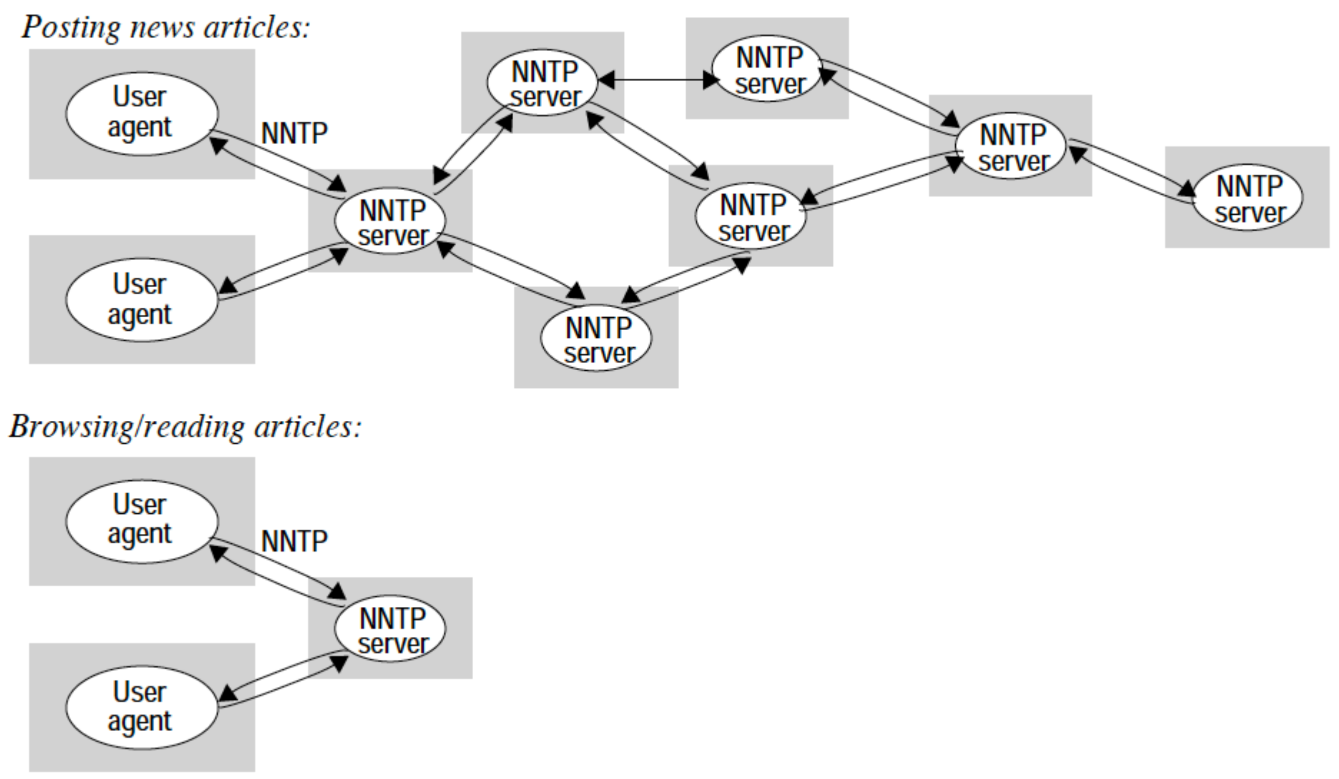

DNS server
DNS
Browser
HTTP
Web server
Proxy server
HTTP
Proxy server
HTTP
DNS server
Browser
Web server

# Mail



Sending messages:
Sender's intranet
User agent
SMTP
SMTP server
User agent
SMTP server
Recipient's mailhost intranet
SMTP server
NFS
Local file server
Reading messages:
Recipient's mailhost intranet
NFS protocol
POP server
POP
User agent
Local file server
IMAP server
IMAP
User agent
NFS
User agent

## Netnews


Posting news articles:
User agent
NNTP
User agent
NNTP server
NNTP server
NNTP server
NNTP server
NNTP server
NNTP server
NNTP server
Browsing/reading articles:
User agent
NNTP
User agent
NNTP server

## Server cooperation

Web: Web servers cooperate with Proxy servers to minimize network traffic and latency. Responsibility for consistency is taken by the proxy servers - they check the modification dates of pages frequently with the originating web server.

Mail: SMTP servers do not necessarily hold mail delivery routing tables to all destinations. Instead, they simply route messages addressed to unknown destinations to another server that is likely to have the relevant tables.

Netnews: All NNTP servers cooperate to provide the newsfeed mechanism.

## Replication

Web: Web page masters are held in a file system at a single server. The information on the web as a whole is therefore partitioned amongst many web servers.

Replication is not a part of the web protocols, but a heavily-used web site may provide servers with identical copies of the relevant file system . HTTP requests can be multiplexed amongst the identical servers using the DNS load sharing mechanism. In addition, web proxy servers support replication through the use of cached replicas of recently-used pages and browsers support replication by maintaining a local cache of recently accessed pages.

Mail: Messages are stored only at their destinations. That is, the mail service is based mainly on partitioning, although a message to multiple recipients is replicated at several destinations.

Netnews: Each group is replicated only at sites requiring it