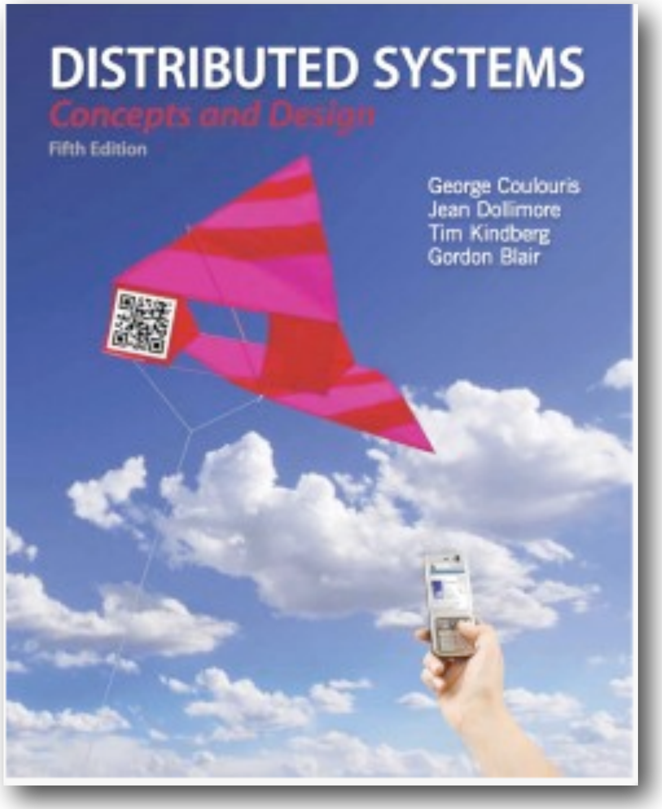
Distributed Systems: Models and Design

Nicola Dragoni
Embedded Systems Engineering
DTU Compute

1. Architectural Models
2. Interaction Model
3. Design Challenges
4. Case Study: Design of a Client-Server System



Architectural vs Fundamental Models

- Systems that are intended for use in real-world environments should be designed to function correctly in the widest possible range of circumstances and in the face of many possible difficulties and threats

- An architectural model is concerned with the placement if its components and the relationships between them

  - client-server systems

  - peer-to-peer systems

- Fundamental models are concerned with a more abstract description of the properties that are common in all of the architectural models

Architectural Models

- The architecture of a system is its structure in terms of separately specified components and their interrelationships

- 4 **fundamental building blocks** (and 4 key questions):

  1. **Communicating entities**: *what are the entities that are communicating in the distributed system?*

  2. **Communication paradigms**: *how do these entities communicate, or, more specifically, what communication paradigm is used?*

  3. **Roles and responsibilities**: *what (potentially changing) roles and responsibilities do these entities have in the overall architecture?*

  4. **Placement**: *how are these entities mapped on to the physical distributed infrastructure (i.e., what is their placement)?*

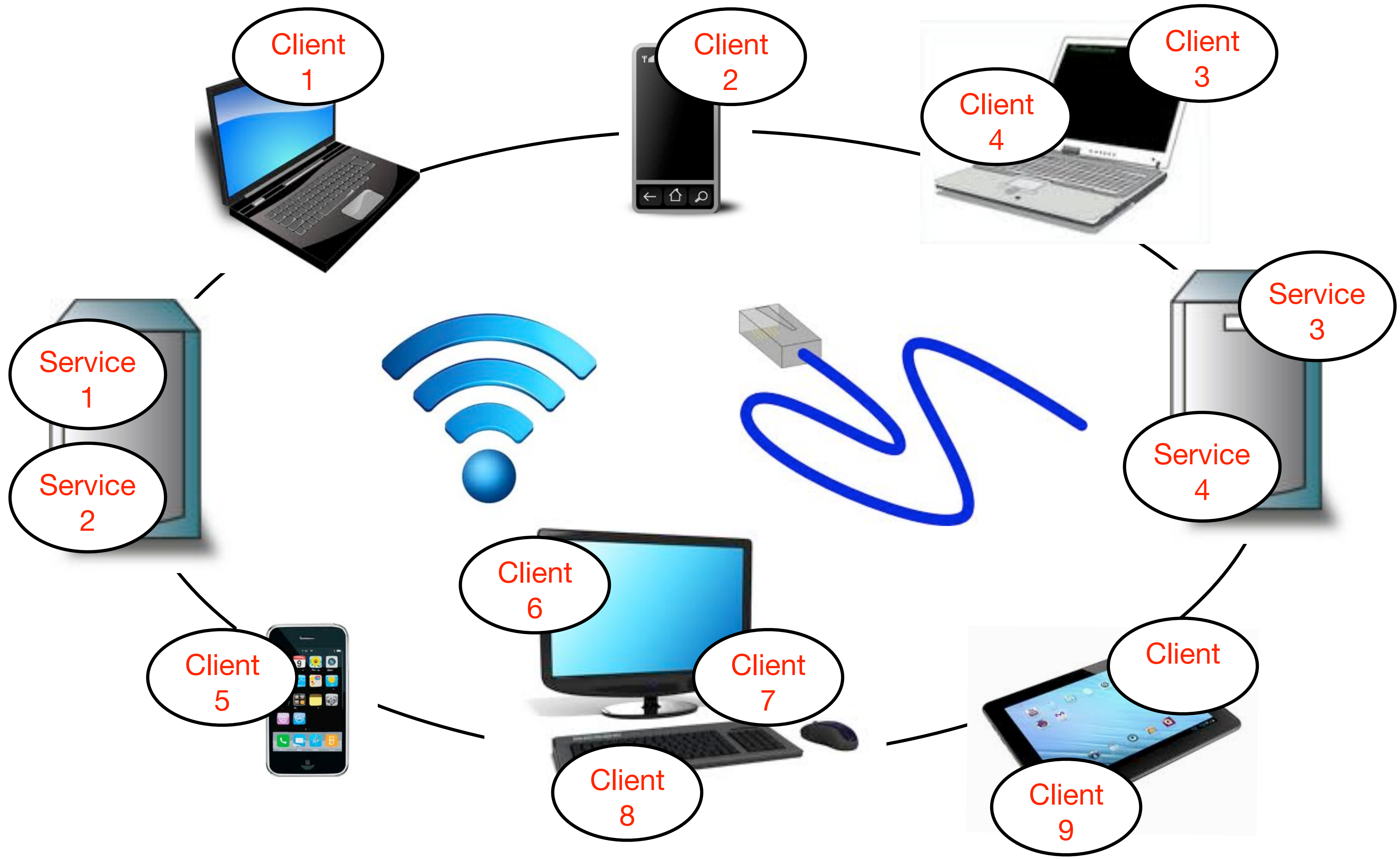
Communicating Entities

What are the entities that are communicating in the distributed system?

Communicating Entities

- **System perspective**:
  - communicating entities are processes
  - distributed system: processes coupled with appropriate interprocess communication paradigms
  - two caveats:
    - in some environment, such as sensor networks, the underlying operating systems may not support process abstractions, and hence the entities that communicate in such systems are nodes
    - in most distributed environments, processes are supplemented by threads (lightweight processes), so, strictly speaking, it is threads that are endpoints of communication

Processes VS Devices



Communicating Entities

- **Programming perspective**:
  - more problem-oriented abstractions have been proposed, such as distributed objects, multi-agent systems, Web services, …
  - distributed objects:
    - introduced to enable and encourage the use of object-oriented approaches in distributed systems
    - computation consists of a number of interacting objects representing natural units of decomposition for the given problem domain
    - objects are accessed via interfaces, with an associated interface definition language providing a specification of the methods defined on an object

Communication Paradigms

How do entities communicate in a distributed systems?

(What communication paradigm is used?)

Communication Paradigms

- 3 types of **communication paradigm** (cont.):
  - interprocess communication
    low level support for communication between processes in the distributed system, including message-passing primitives, socket programming, multicast communication
  - remote invocation
    most common communication paradigm, based on a two-way exchange between communicating entities and resulting in the calling of a remote operation (procedure or method)

Communication Paradigms

- 3 types of **communication paradigm** (cont.):
  - indirect communication
    communication is indirect, through a third entity, allowing a strong degree of decoupling between senders and receivers, in particular:
    - space uncoupling: senders do not need to know who they are sending to
    - time uncoupling: senders and receivers do not need to exist at the same time

    Key techniques include: group communication, publish subscribe systems, message queues, tuple spaces, distributed shared memory (DSM)

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

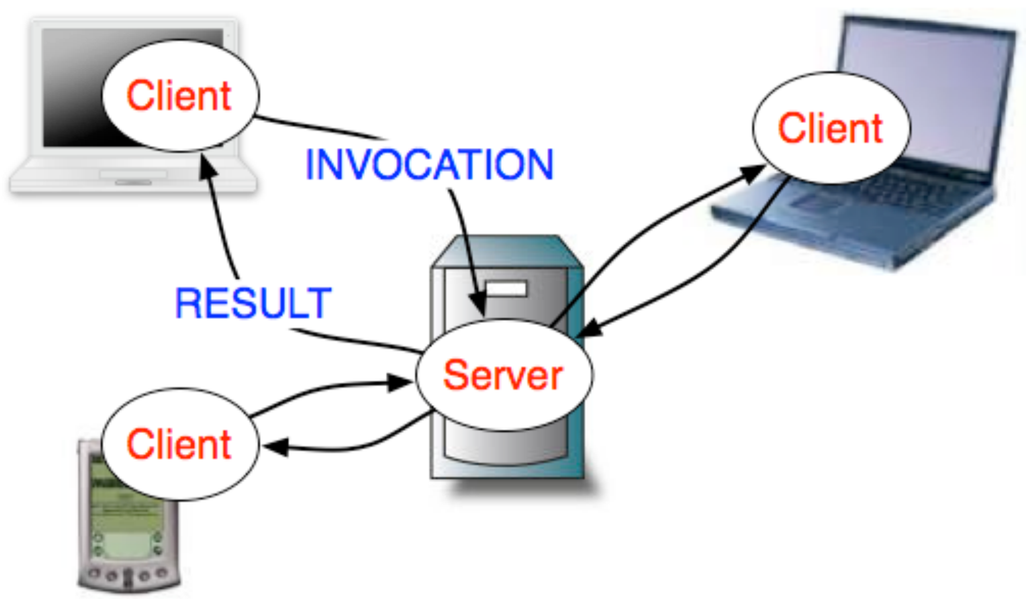
Roles & Responsibilities

What (potentially changing) roles and responsibilities do these entities have in the overall architecture?

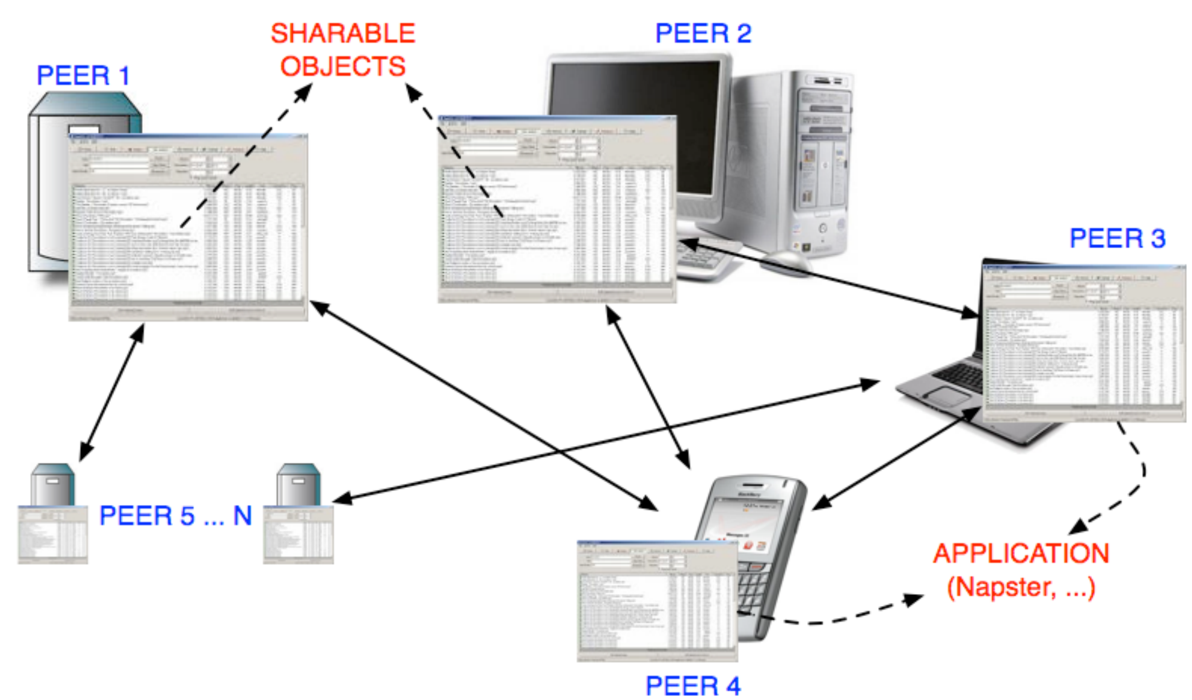
Roles & Responsibilities

- 2 architectural styles stemming from the role of individual processes

client-server

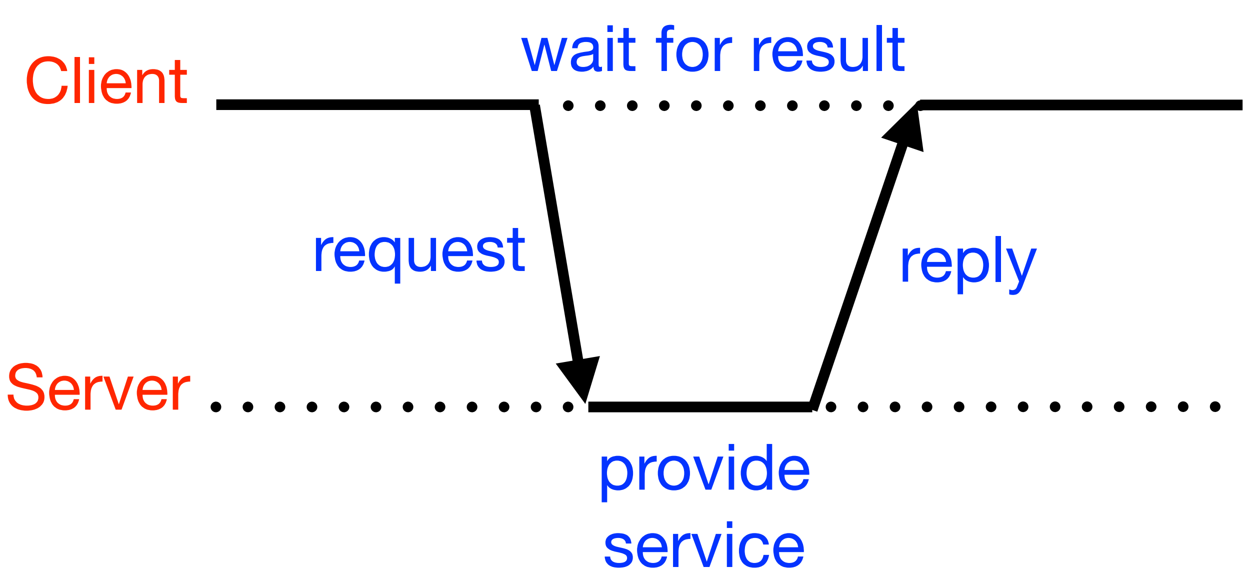


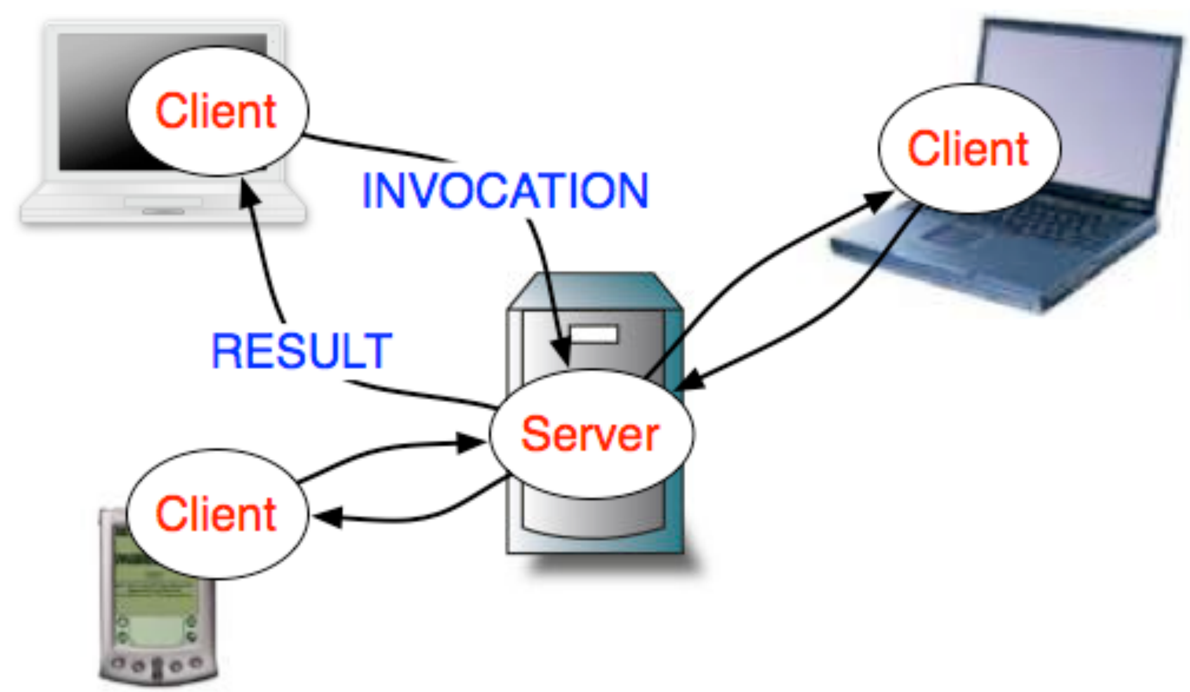
peer-to-peer (P2P)



Client-Server Architectural Style

- Processes divided into two (possibly overlapping) groups:
  - Server: process implementing a specific service (file system service, database service, ...)
  - Client: process that requests a service from a server by sending it a request ad subsequently waiting for the server's reply
- Request-reply protocol



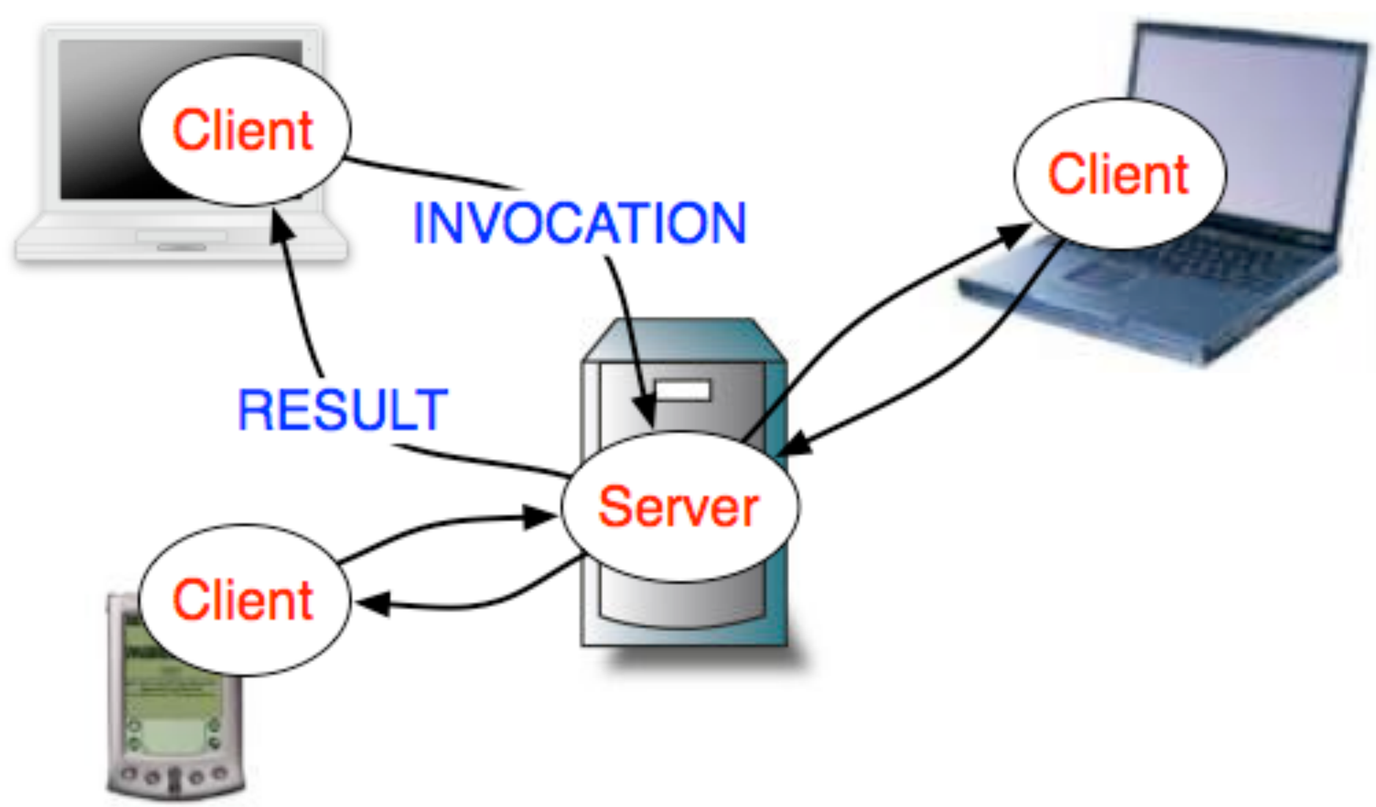


Client-Server Interaction

- Requests are sent in messages from clients to a server
  - When a client sends a request for an operation to be carried out, we say that the client invokes an operation upon the server

- Replies are sent in messages from the server to the clients

- Remote invocation: a complete interaction between a client and a server (from the point when the client sends its request to when it receives the server's response)



Example: The Web as Client-Server Resource Sharing System

- The World Wide Web is an evolving and open system for publishing and accessing resources and services across the Internet

- For instance, through Web browsers (clients) users can
  - retrieve and view documents of many types
  - listen to audio streams
  - view video streams
  - and in general interact with an unlimited set of services

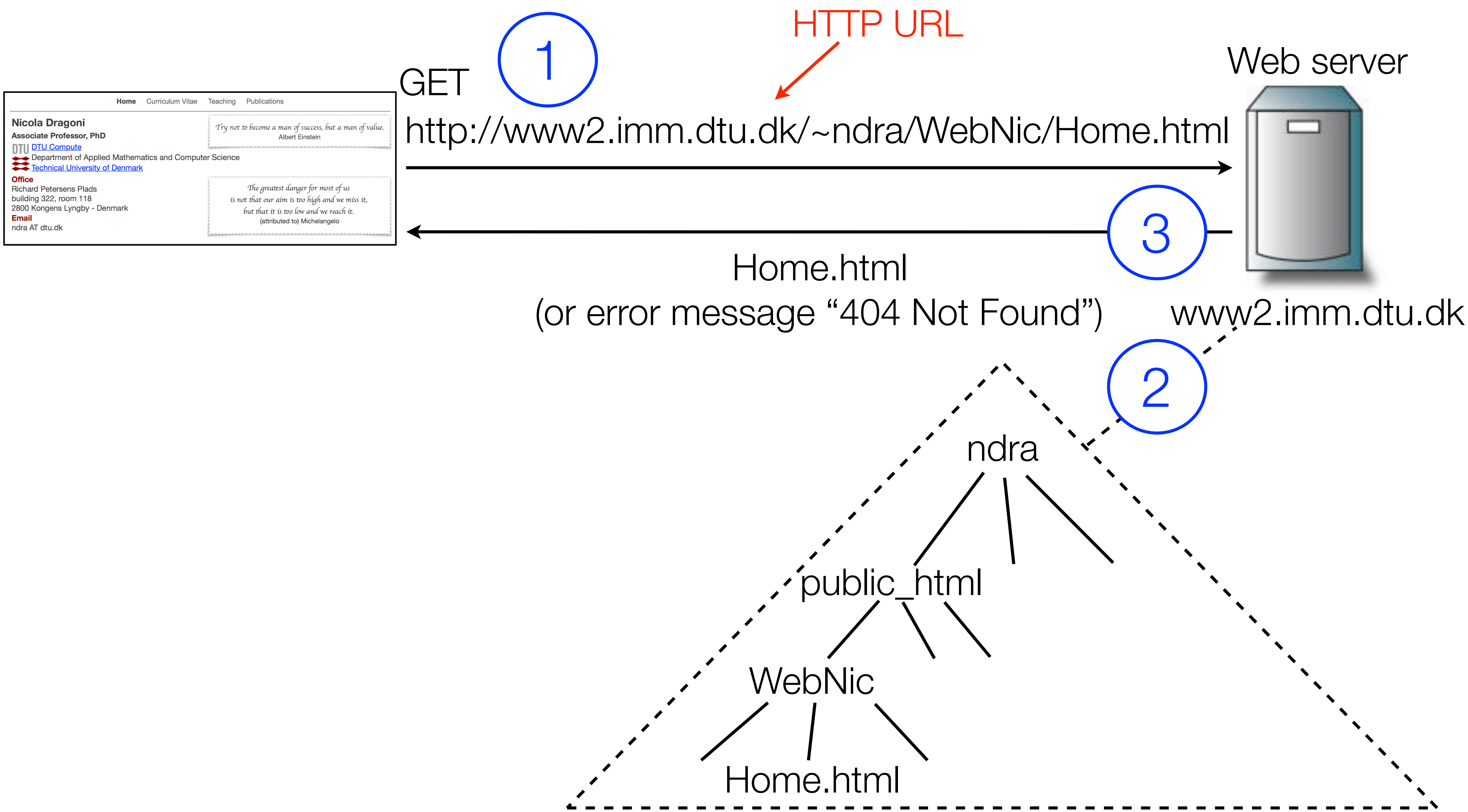
[Web] Main Technological Components

1. The HyperText Markup Language (HTML) is a language for specifying the contents and layout of pages as they are displayed by Web browsers

2. Uniform Resource Locators (URLs) which identify documents and other resources stored as part of the Web

3. A client-server system architecture, with standard rules for interaction (the HyperText Transfer Protocol - HTTP) by which browsers and other clients fetch documents and other resources from Web servers

Web Browser and Web Server Example



On the Client and Server Role...

- **A process can be both a client and a server**, since servers sometimes invoke operations on other servers
- The terms "client" and "server" apply only to the roles played in a single request
- But in general they are distinct concepts:
  - clients are active and server are passive (reactive)
  - server run continuously, whereas clients last only as long as the applications of which they form a part

On the Client-Server Role: Examples

- Example 1: a Web server is often a client of a local file server that manages the files in which the web pages are stored
- Example 2: Web servers and most Internet services are clients of the DNS service (which translates Internet Domain names to network addresses)
- Example 3: search engine
  - Server: it responds to queries from browser clients
  - Client: it runs (in the background) programs called *web crawlers* that act as clients of other web servers

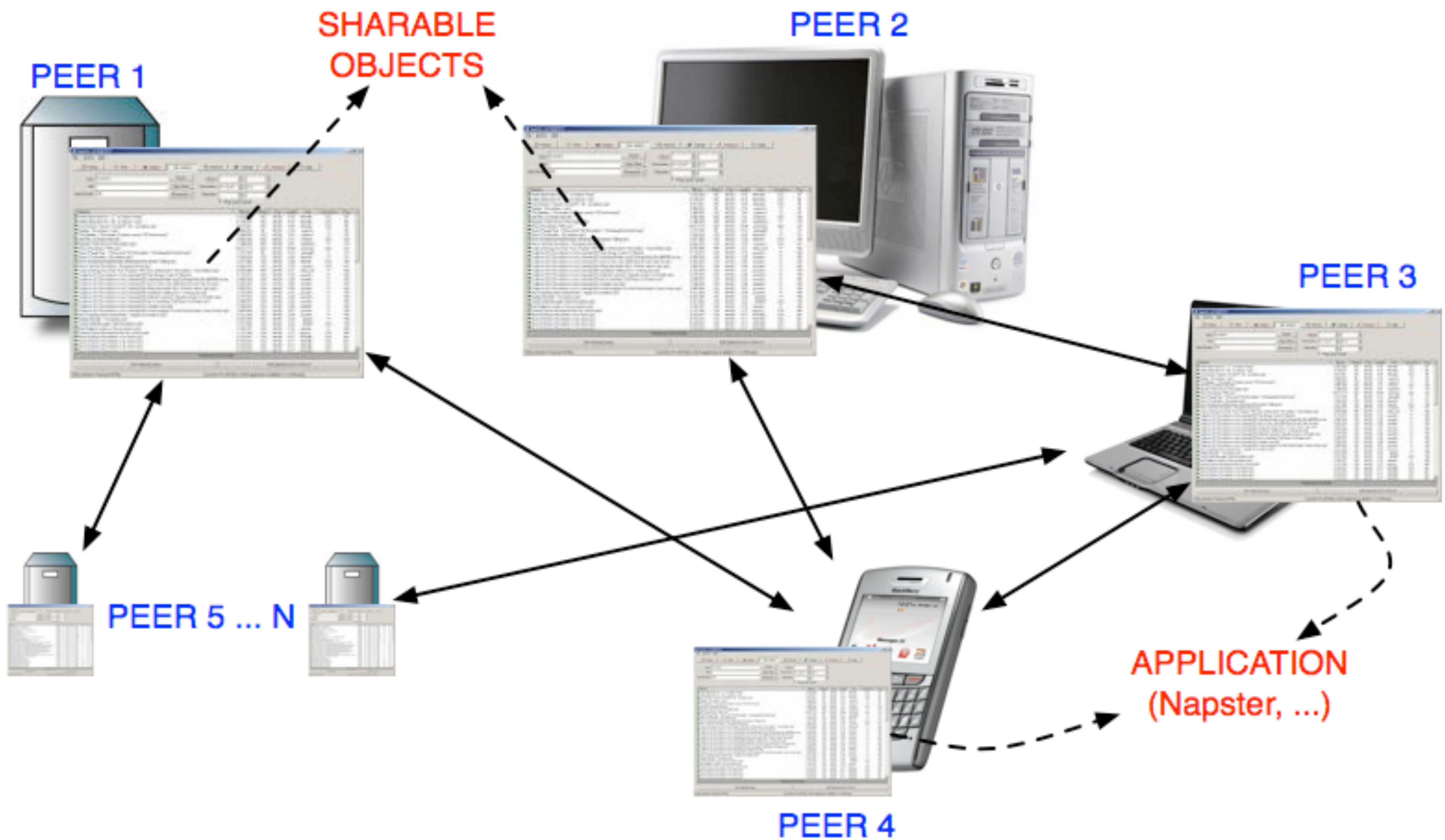
Architectural Style: Peer-to-Peer (P2P)

- All the processes involved in a task or activity play similar roles, interacting *cooperatively* as peers without any distinction between client and server processes or the computers that they run on

- In practical terms, all peers run the same program and offer the same set of interfaces to each other

The aim of the P2P architecture is to exploit the resources (both data and hardware) in a large number of participating computers for the fulfilment of a given task or activity

Distributed Application Based on a P2P Architecture



Placement

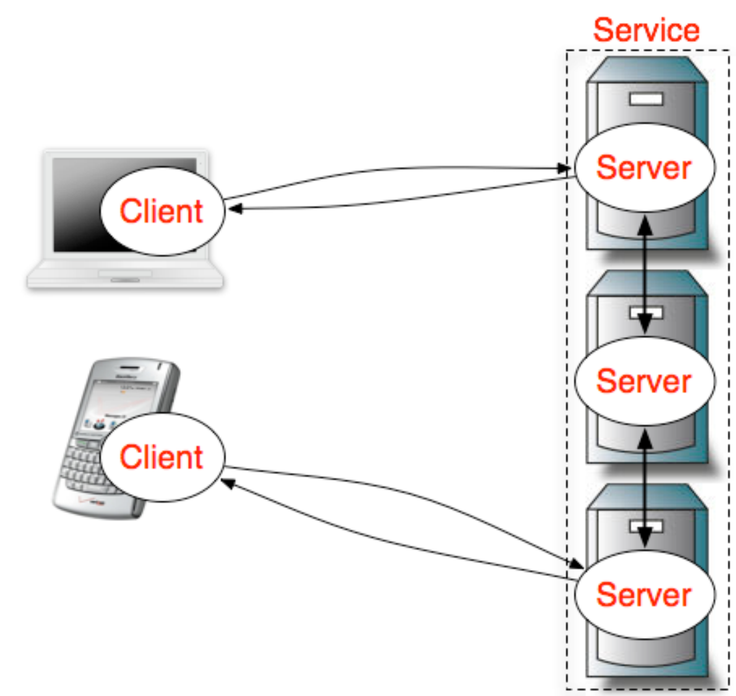
How are entities mapped on to the physical distributed infrastructure
(i.e., what is their placement)?

Placement

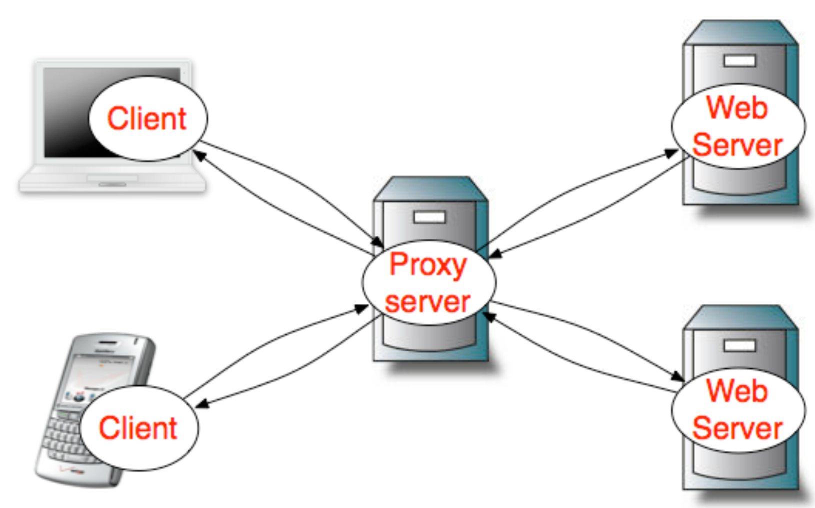
- Physical distributed infrastructure usually consists of a potentially large number of machines interconnected by a network of arbitrary complexity
- Placement is crucial in terms of determining the properties of the distributed system, such as performance, reliability and security
- Placement need to take into account several aspects (machines, reliability, communication, ...) and there are few universal guidelines to obtaining an optimal solution!

Placement Strategies

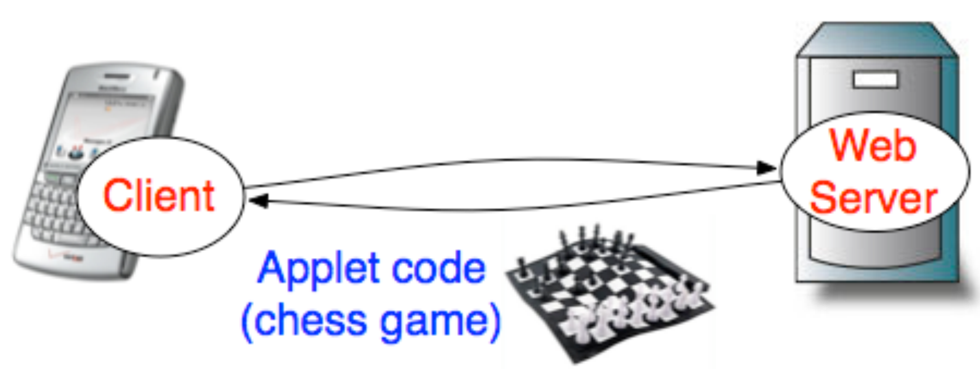
- Mapping of services to multiple servers



- Proxy server and caches



- Mobile code



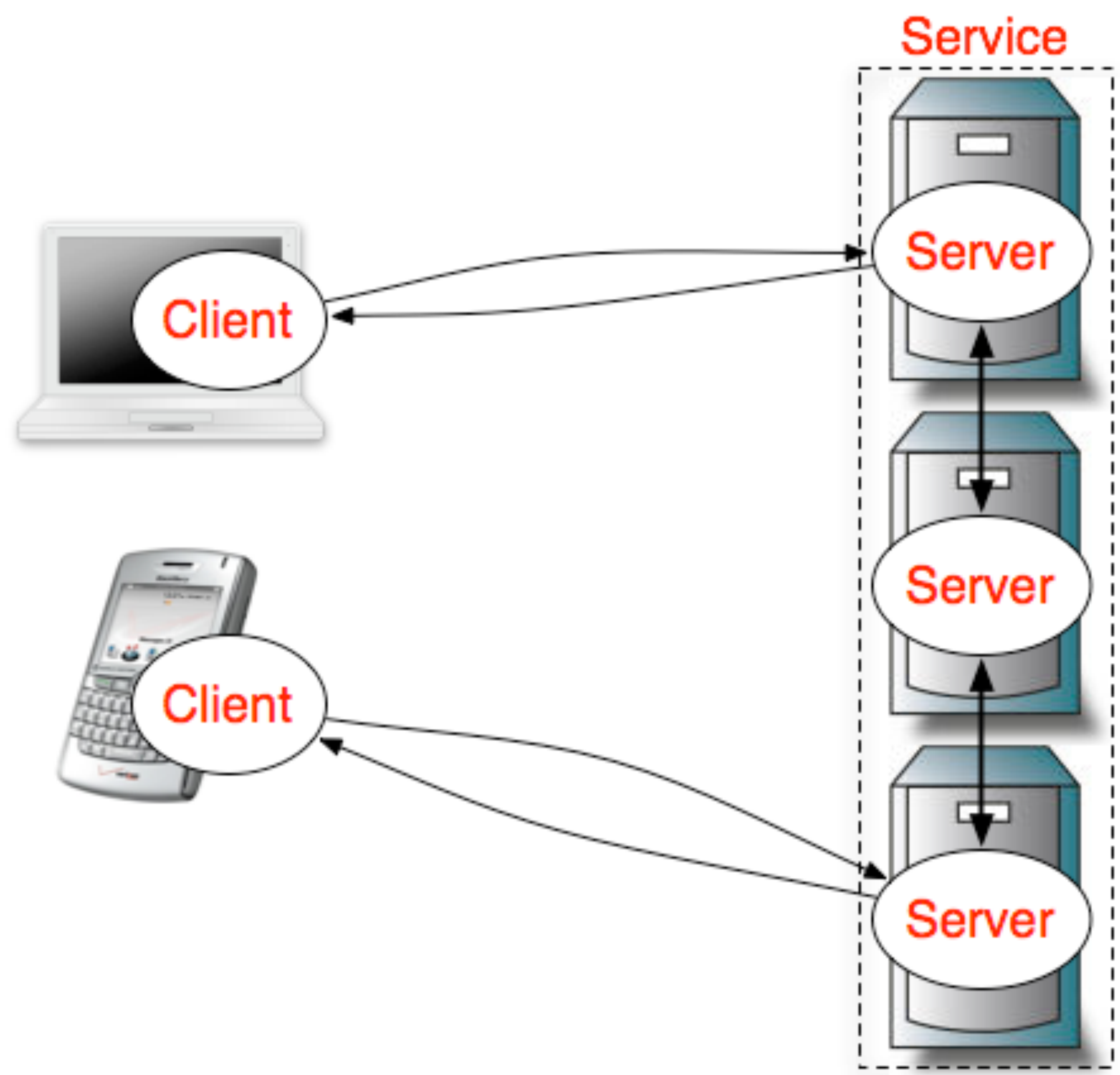
Placement Strategy: Service Provided by Multiple Servers

- Services may be implemented as several server processes in separate host computers interacting as necessary to provide a service to client processes

- The servers may:

  1) partition the set of objects on which the service is based and distributed them between themselves (e.g. Web servers)

  2) they may maintain replicated copies of them on several hosts (e.g. SUN Network Information Service (NIS)).



Placement Strategy: Proxy Servers and Caches

- A cache is a store of recently used data objects that is closer to one client or a particular set of clients than the objects themselves

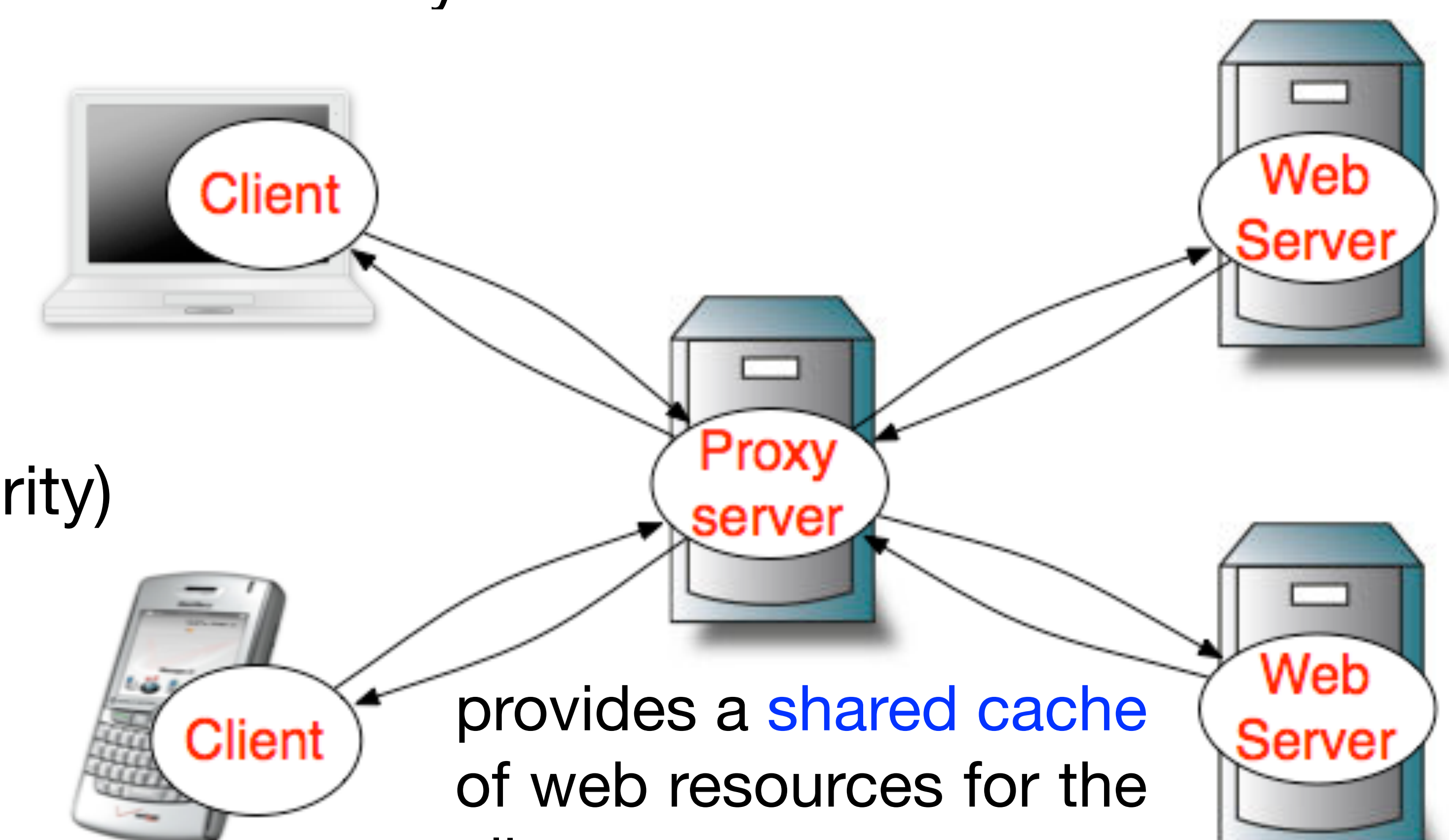
- Example 1: Web browsers maintain a cache of recently visited pages and other web resources in the client's local file system

- Example 2: Web proxy server

  Purpose:

  1. To keep machines behind it anonymous (mainly for security)

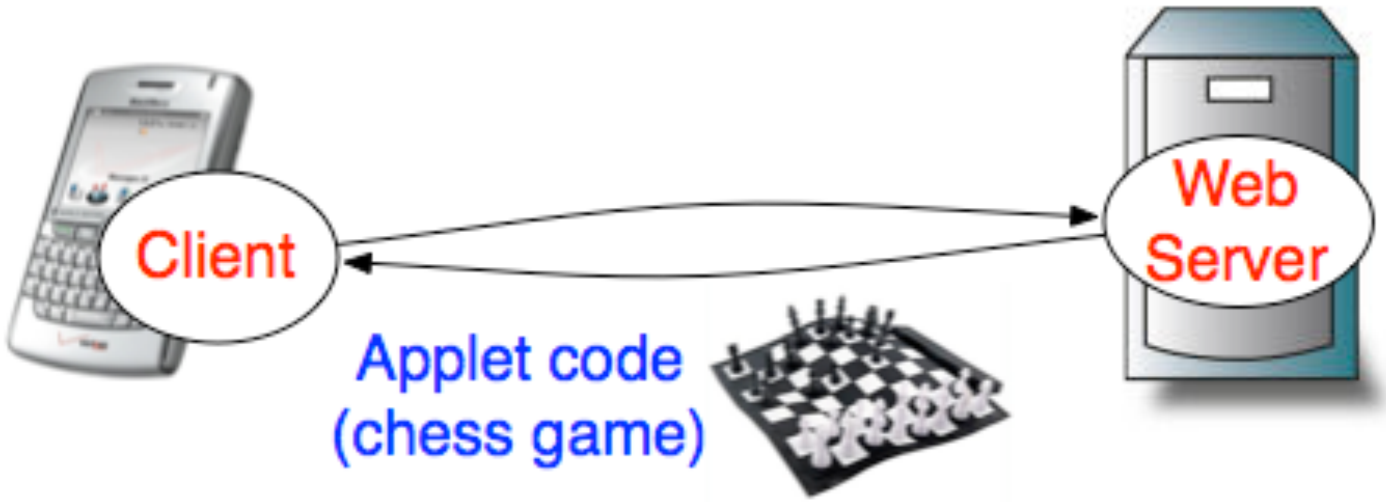
  2. To speed up access to a resource (via caching)



provides a shared cache of web resources for the clients

Placement Strategy: Mobile Code

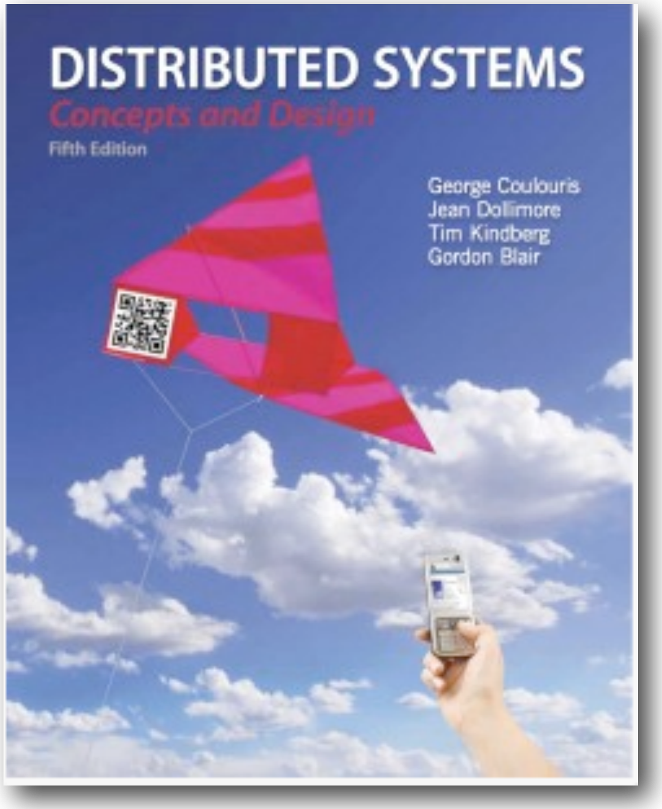
A) Client request results in the downloading of applet code



B) Client interacts with the applet

An advantage of running the downloaded code *locally* is that it can give good interactive response since it does not suffer from the delays or variability of bandwidth associated with network communication

Interaction Model

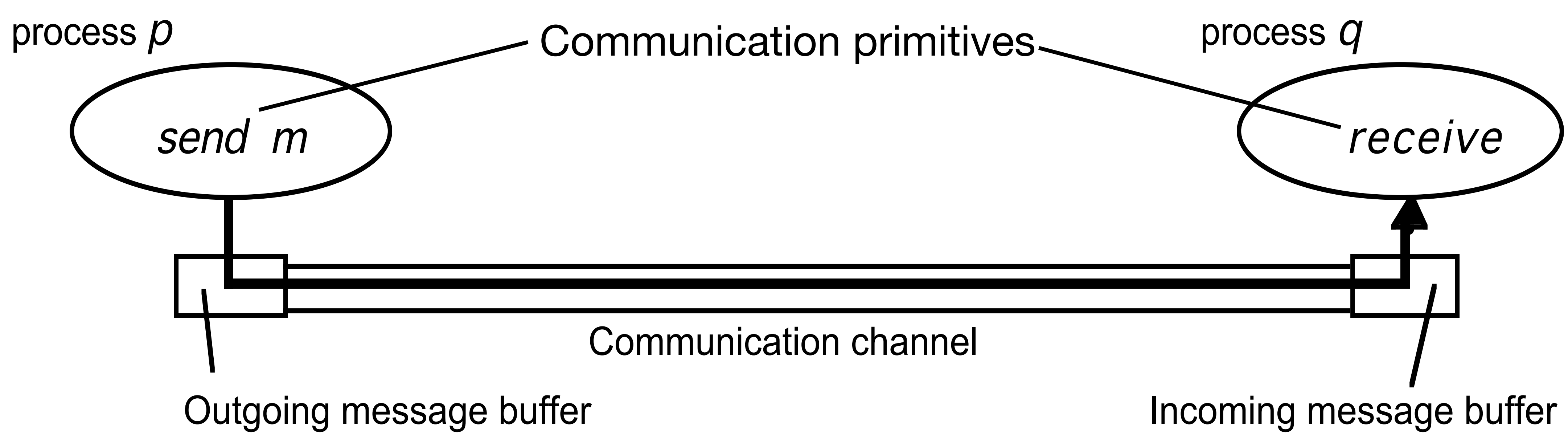


1. Architectural Models
2. Interaction Model
3. Design Challenges
4. Case Study: Design of a Client-Server System

Some Assumptions on Interacting Processes

- The rate at which each process proceeds **cannot** in general be predicted
- The timing of the transmission of messages **cannot** in general be predicted
- Each process has its own state, consisting of the set of data that it can access and update, including the variables in its program
- The state belonging to each process is completely private (that is, it cannot be accessed or updated by any other processes)

Processes and Communication Channels



- A process *p* performs a *send* by inserting the message m in its outgoing message buffer
- The communication channel transports *m* to q's incoming message buffer
- Process *q* performs a *receive* by taking m from its incoming message buffer and delivering it
- Outgoing/incoming message buffers are typically provided by the operating systems

Factors Affecting Interacting Processes

- Communication performance

- It is impossible to maintain a single global notion of time

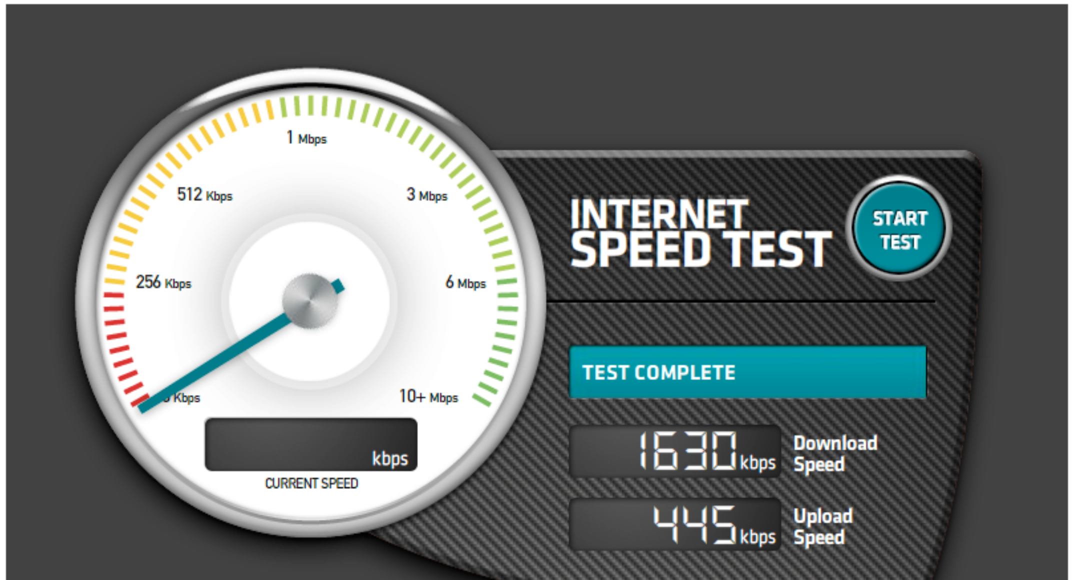
Performance of Communication Channels: Latency

- Latency: the delay between the start of a message's transmission from one process and the beginning of its receipt by another

- The latency includes:
  - The time taken for the first of a string of bits transmitted through the network to reach its destination
  - The delay in accessing the network, which increases significantly when the network is heavily loaded
  - The time taken by the operating system communication services at both the sending and receiving processes, which varies according to the current load of the operating systems

Performance of Communication Channels: Bandwidth

- The bandwidth of a computer network is the total amount of information that can be transmitted over it in a given time
- Usually expressed in bit/s or multiples of it (kbit/s, Mbit/s, etc)
- When a large number of communication channels are using the same network, they have to share the available bandwidth



http://www.bandwidthplace.com/

56 kbit/s	Modem / Dialup
1.5 Mbit/s	ADSL Lite
1.544 Mbit/s	T1
10 Mbit/s	Ethernet
11 Mbit/s	Wireless 802.11b
44.736 Mbit/s	T3
54 Mbit/s	Wireless-G 802.11g
100 Mbit/s	Fast Ethernet
155 Mbit/s	OC3
300 Mbit/s	Wireless-N 802.11n
622 Mbit/s	OC12
1000 Mbit/s	Gigabit Ethernet
2.5 Gbit/s	OC48
9.6 Gbit/s	OC192
10 Gbit/s	10 Gigabit Ethernet

No Global Clock!!

- No single global notion of correct time.

Computer Clocks and Timing Events

- Each computer in a distributed system has its own internal clock, which can be used by local processes to obtain a value of the current time
- Therefore, two processes running on different computers can associate timestamps with their events
- However, **even if two processes read their clocks at the same time, their local clocks may supply different time values**
- This is because computer clocks drift from perfect time and, more importantly, their drift rates differ from one another
- Clock drift rate: rate at which a computer clock deviates from a perfect reference clock

Variants of the Interaction Model

- In a distributed system it is hard to set time limits on the time taken for process execution, message delivery or clock drift

- Two opposite extreme positions provide a pair of simple models:
  - Synchronous distributed systems: strong assumption of time
  - Asynchronous distributed systems: no assumptions about time

Synchronous Distributed System

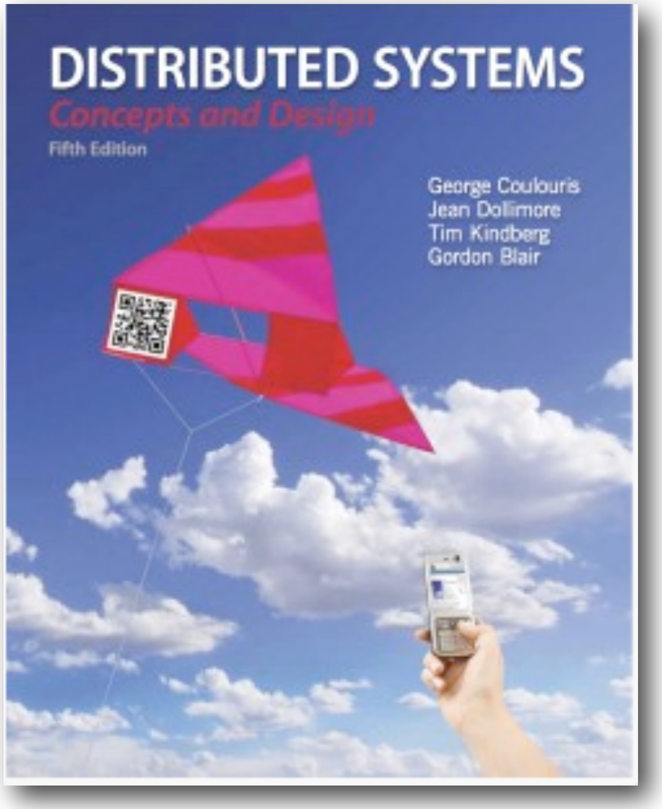
- A distributed system in which the following bounds are defined:
  - the time to execute each step of a process has known lower and upper bounds
  - each message transmitted over a channel is received within a known bounded time
  - each process has a local clock whose drift rate from real time has a known bound

Asynchronous Distributed System

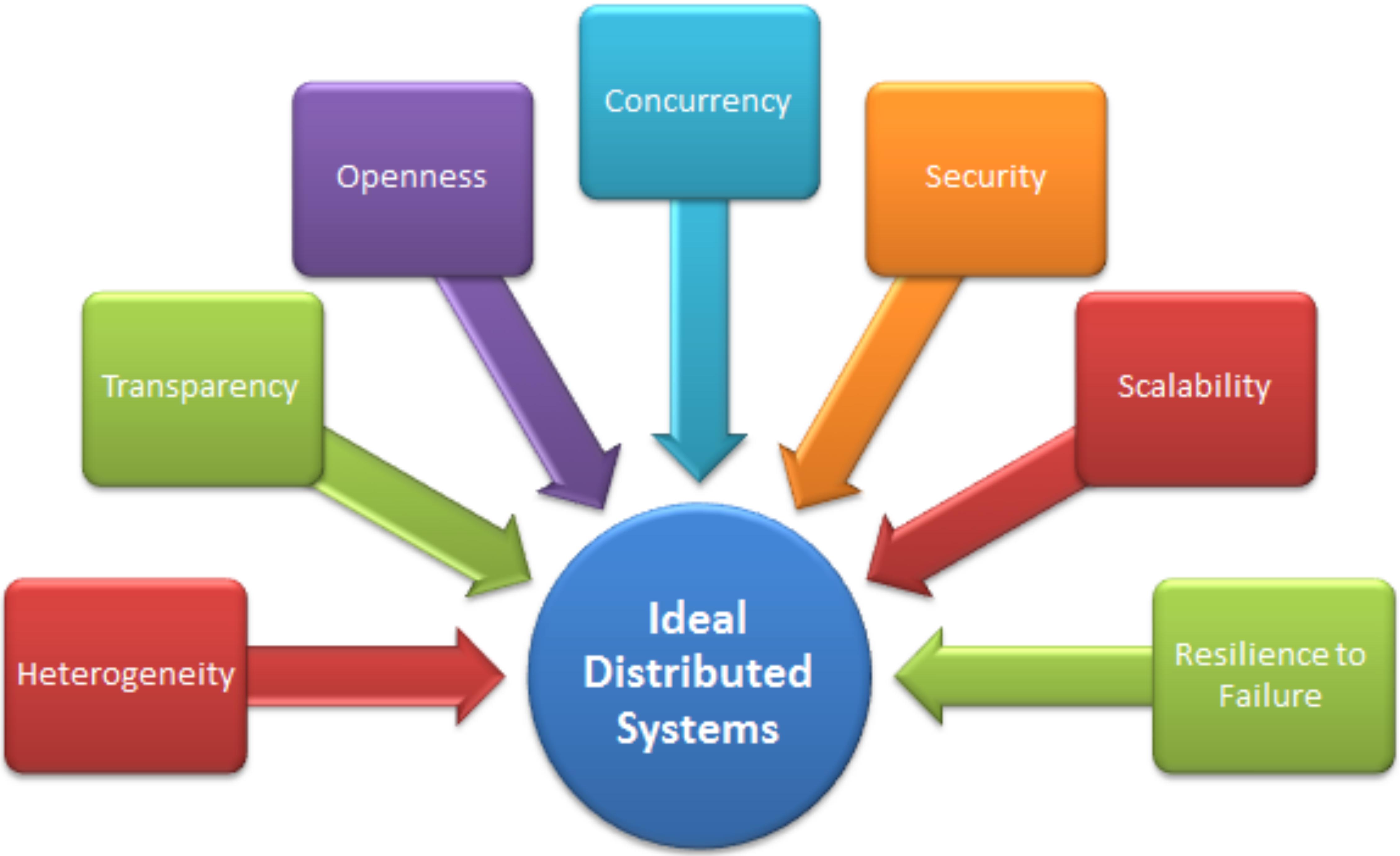
- A distributed system in which there are no bounds on:
  - process execution speeds: each step may take an arbitrarily long time
  - message transmission delays: a message may be received after an arbitrarily long time
  - clock drift rates: the drift rate of a clock is arbitrary
- This exactly models the Internet, in which there is no intrinsic bound on server or network load and therefore on how long it takes, for example, to transfer a file using ftp, or to receive an email message
- *Any solution that is valid for an asynchronous distributed system is also valid for a synchronous one*. **Why? What about the contrary?**

Design Challenges

1. Architectural Models
2. Interaction Model
3. Design Challenges
4. Case Study: Design of a Client-Server System



Design Challenges for Distributed Systems

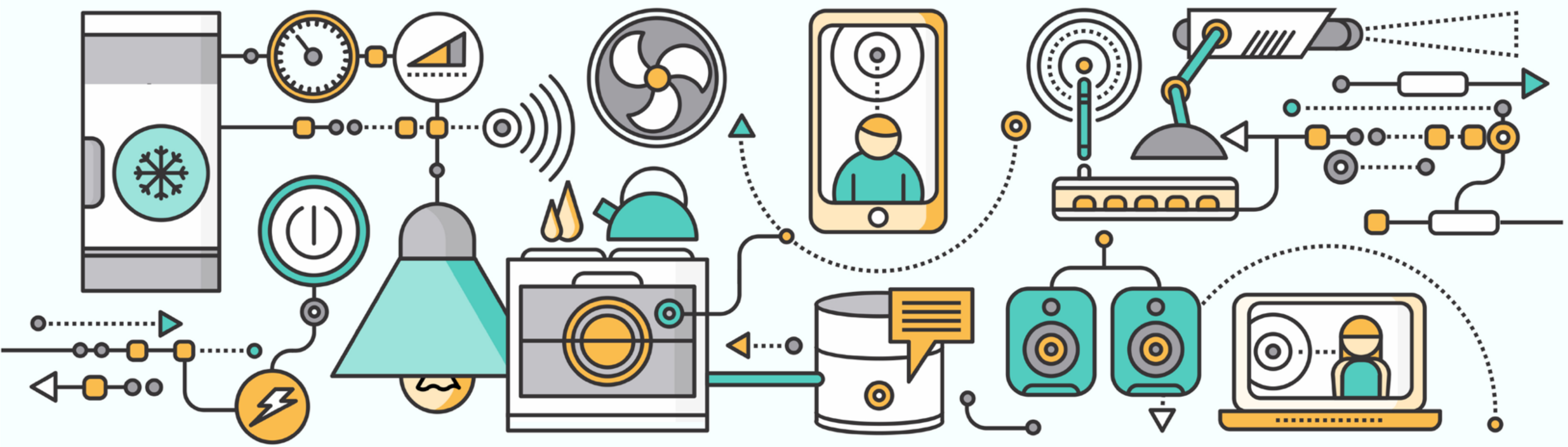


Heterogeneity of Components

- Heterogeneity (i.e., variety and difference) applies to the following:
  - networks
  - computer hardware
  - operating systems
  - programming languages
  - implementations by different developers

Heterogeneity can be addressed by means of:
- protocols (such as Internet protocols)
- middleware (software layer that provides a programming abstraction)

Example: Internet of Things (IoT)



- IoT is a complex, heterogeneous, distributed systems
  - Enormous differences in resources across tiers
  - Many programming languages, OSes, platforms, networks, …
  - Specialized hardware

Openness

- The openness of a computer system is the characteristic that determines whether the system can be *extended* and *re-implemented* in various ways

- In distributed systems it is determined primarily by the degree to which new resource sharing services can be added and be made available for use by a variety of client programs

- Open distributed systems may be extended
    - at the hardware level by the addition of computers to the network
    - at the software level by the introduction of new services and the re-implementation of old ones

Scalability

- A system is scalable if it will remain effective when there is a significant increase in the number of resources and the number of users
- The Internet provides an illustration of a distributed system in which the number of computers and services has increased dramatically

Date	*Computers*	*Web servers*	*Percentage*
1993, July	1,776,000	130	0.008
1995, July	6,642,000	23,500	0.4
1997, July	19,540,000	1,203,096	6
1999, July	56,218,000	6,598,697	12
2001, July	125,888,197	31,299,592	25
2003, July	~200,000,000	42,298,371	21
2005, July	353,284,187	67,571,581	19

Example

Challenge: preventing software resources running out

Example: Internet IP addresses (computer addresses in the Internet)

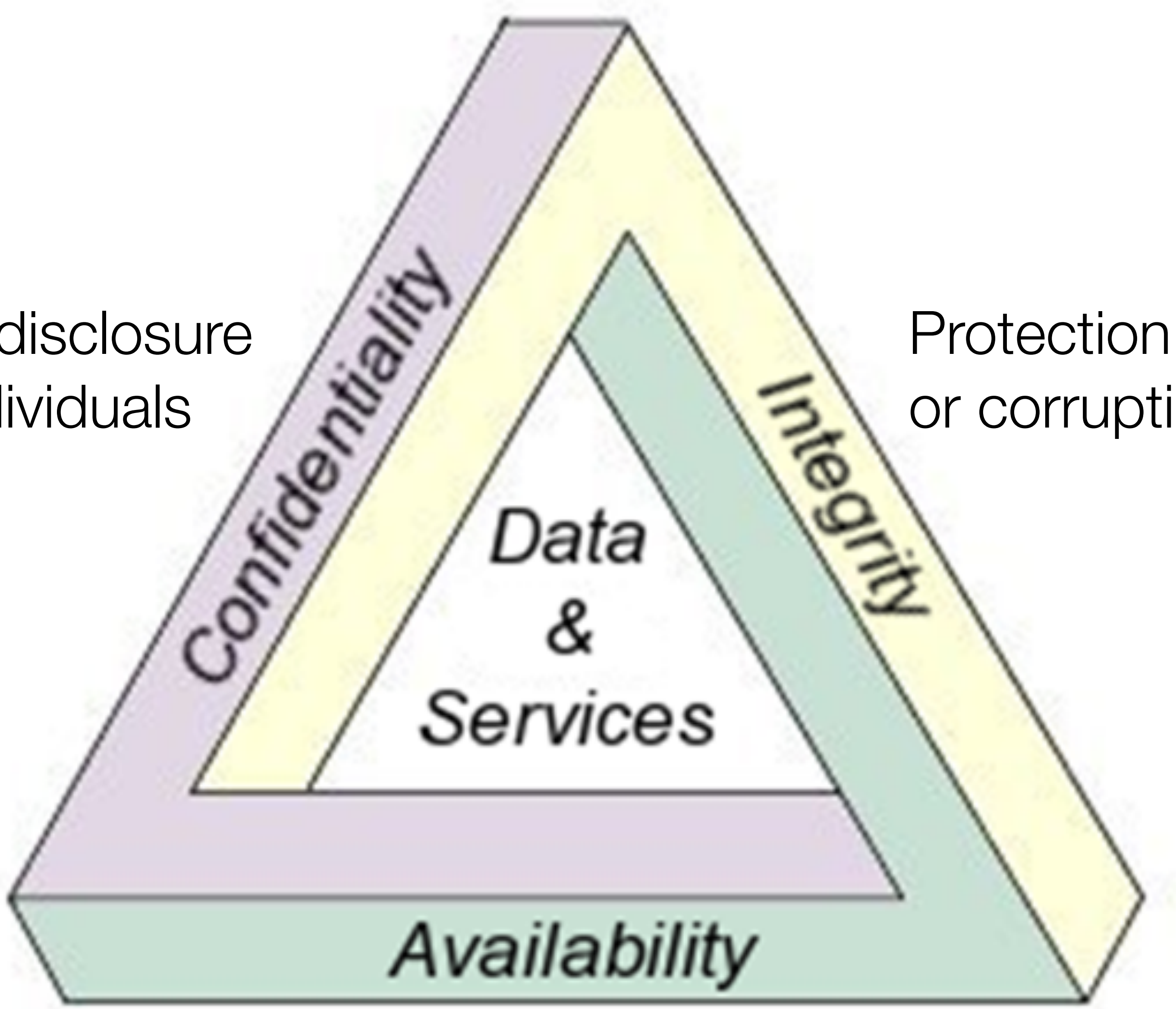
- In the late 1970s, it was decided to use 32 bits, but the supply of available Internet addresses is running out
- For this reason, a new version of the protocol with 128-bit Internet addresses is being adopted and this will require modifications to many software components

- How to solve this problem? Not easy!
  - It is difficult to predict the demand that will be put on a system years ahead
  - Over-compensating for future growth may be worse than adapting to a change when we are forced to (for instance, larger Internet addresses will occupy extra space in messages and in computer storage)

Another example: **Cloud Computing**

Security

Protection against disclosure to unauthorized individuals



Protection against alteration or corruption

Protection against interference with the means to access the resources

Security Challenge: Denial of Service Attack

- A bad guy may wish to disrupt a service for some reason:
  - he bombards the service with such a large number of pointless requests that the serious users are unable to use it

- On August 6, 2009, Twitter was shut down for hours due to a DoS attack:

THU AUG 6TH

twitter

Ongoing denial-of-service attack 21 hours ago

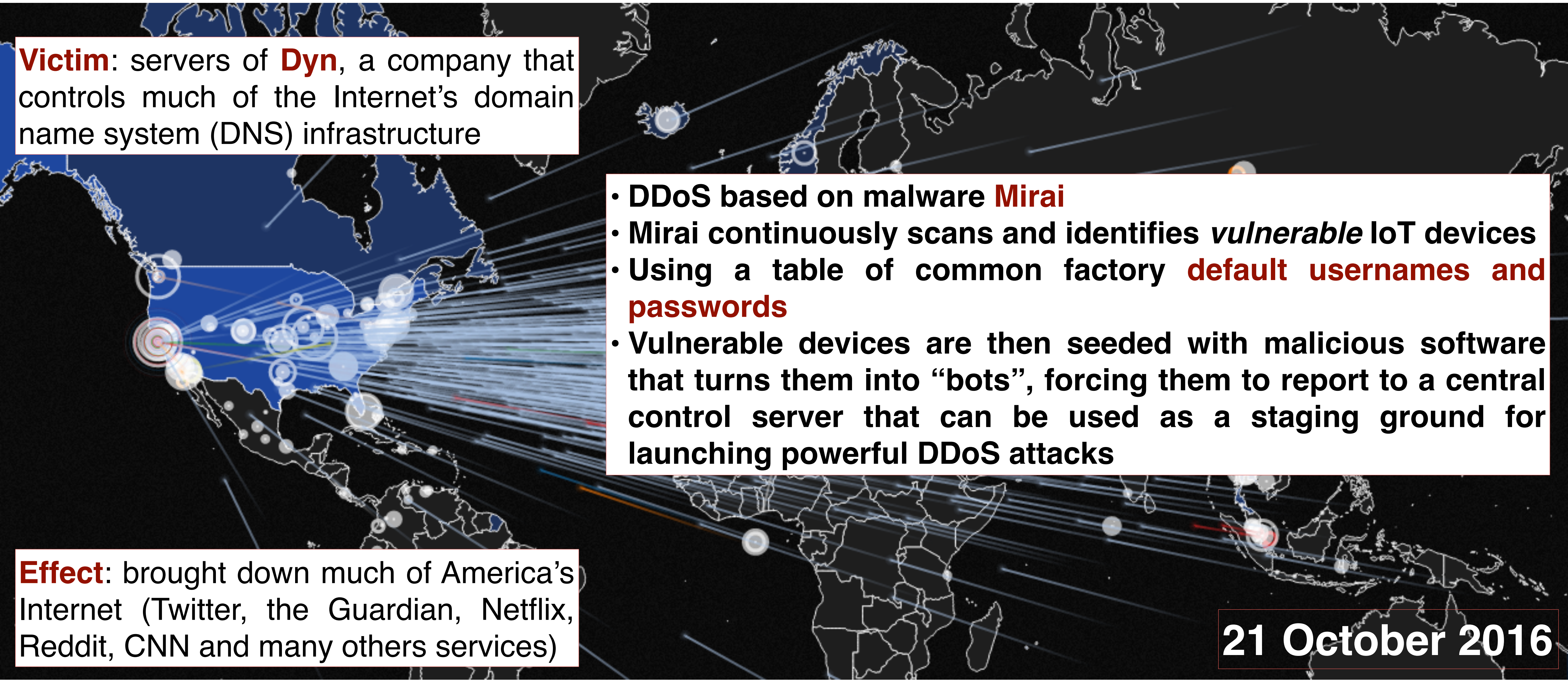
We are defending against a denial-of-service attack, and will update status again shortly.

Update: the site is back up, but we are continuing to defend against and recover from this attack.

Update (9:46a): As we recover, users will experience some longer load times and slowness. This includes timeouts to API clients. We're working to get back to 100% as quickly as we can.

Update (4:14p): Site latency has continued to improve, however some web requests continue to fail. This means that some people may be unable to post or follow from the website.

Largest DDoS Attack Ever (~1.2Tbps)



Victim: servers of **Dyn**, a company that controls much of the Internet's domain name system (DNS) infrastructure

- **DDoS based on malware Mirai**
- **Mirai continuously scans and identifies *vulnerable* IoT devices**
- **Using a table of common factory default usernames and passwords**
- **Vulnerable devices are then seeded with malicious software that turns them into "bots", forcing them to report to a central control server that can be used as a staging ground for launching powerful DDoS attacks**

Effect: brought down much of America's Internet (Twitter, the Guardian, Netflix, Reddit, CNN and many others services)

21 October 2016

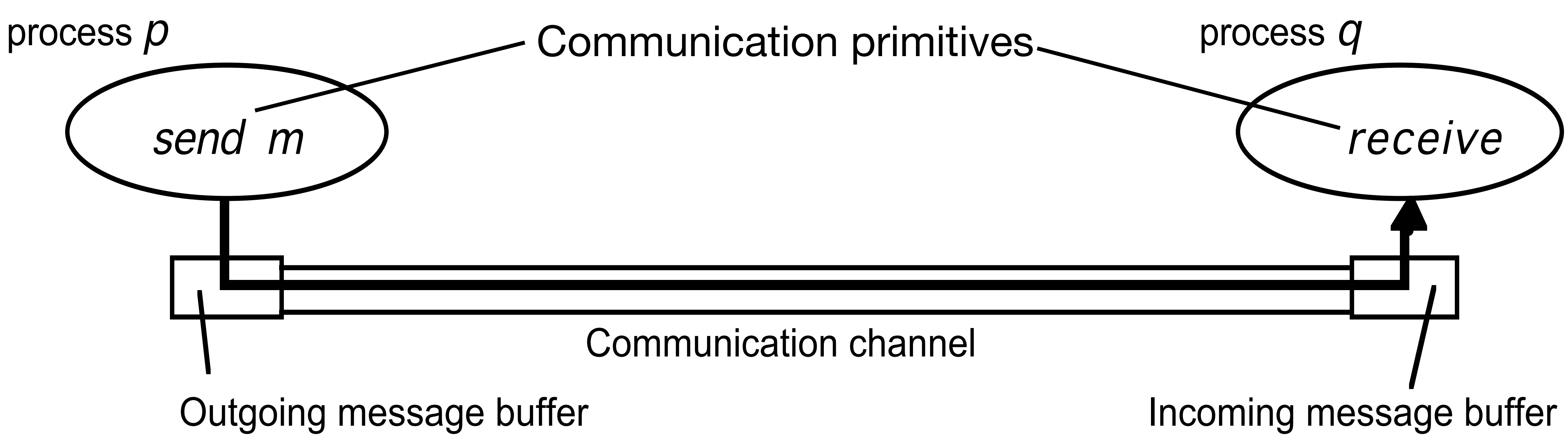
Failure Handling

- Computer systems *sometimes* fail
- When faults occur in hardware or software, programs may produce incorrect results or they may stop before they have completed the intended computation
- Failures in distributed systems are partial:
  - any process, computer or network may fail independently of the others
  - some components fail while others continue to function
- Therefore the handling of failures in distributed systems is particularly difficult

Failure Model

- The failure model defines the ways in which failures may occur in order to provide an understanding of the effects of failures
- Example of taxonomy of failures [Hadzilacos and Toueg, 1994]:
  - Omission failures: a process or communication channel fails to perform actions that it is supposed to do
  - Arbitrary failures: any type of error may occur
  - Timing failures: applicable in synchronous distributed systems

[Failure Model] Omission Failures



Class of failure	Affects	Description
Crash	Process	Process halts prematurely and remain halted.
Omission	Channel	A msg inserted in an outgoing msg 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.

[Failure Model] Arbitrary Failures

- The term *arbitrary* or *Byzantine* failure is used to describe the worst possible failure semantics, in which any type of error may occur
- Arbitrary failure of a process: the process arbitrarily omits intended processing steps or takes unintended processing steps
- Communication channel arbitrary failures: message contents may be corrupted or non-existent messages may be delivered or real messages may be delivered more than once

Class of failure	**Affects**	**Description**
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.

[Failure Model] Timing Failures

- Timing failures are applicable in synchronous distributed systems, where time limits are set on process execution time, message delivery time and clock drift rate

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.

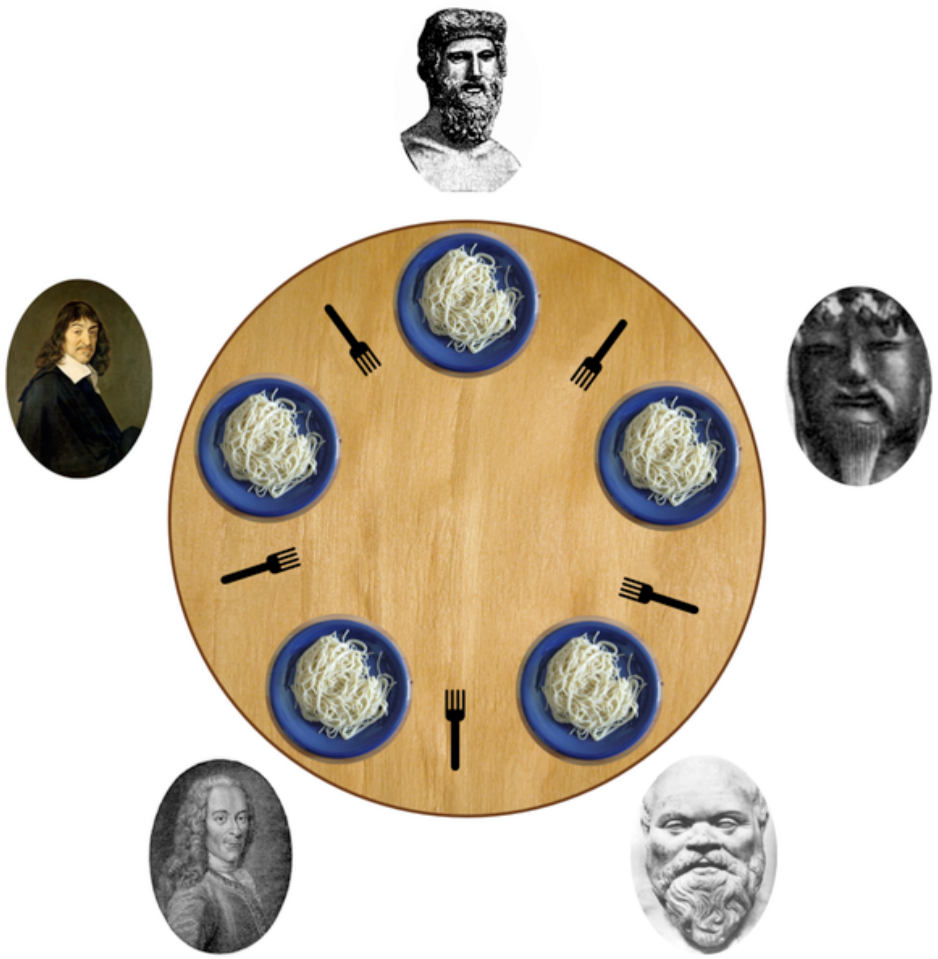
- In an asynchronous distributed systems, an overloaded server may respond too slowly, but *we cannot say that it has a timing failure since no guarantee has been offered*

Concurrency

- Both services and applications provide resources that can be shared by different clients in a distributed system

- There is therefore a possibility that several clients will attempt to access a shared resource at the same time

- Each resource (servers, Web resources objects in applications, ...) must be designed to be safe in a concurrent environment



Dining Philosophers Problem

Transparency

- Transparency: the concealment from the user and the application programmer of the separation of components in a distributed system, so that the system is perceived as a whole rather than a collection of independent components

- Aim: to make certain aspects of distribution invisible to the application programmer so that they need only be concerned with the design of their particular application

- The ANSA Reference Manual and the International Organization for Standardization's Reference Model for Open Distributed Processing (RM-ODP) identify 8 forms of transparency

Transparencies

Network transparency

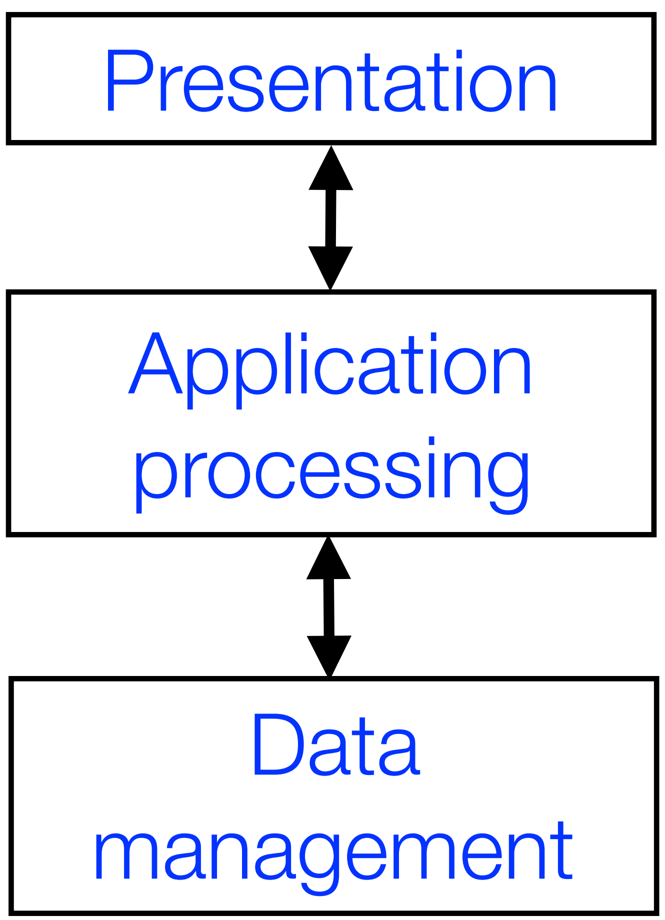
Access Transparency	Enables local and remote resources to be accessed using identical operations
Location Transparency	Enables resources to be accessed without knowledge of their physical or network location (for example, which building or IP address)
Concurrency Transparency	Enables several processes to operate concurrently using shared resources without interference between them
Replication Transparency	Enables multiple instances of resources to be used to increase reliability and performance without knowledge of the replicas by users or application programmers
Failure Transparency	Enables the concealment of faults, allowing users and application programs to complete their tasks despite the failure of hardware or software components
Mobility Transparency	Allows the movement of resources and clients within a system without affecting the operation of users or programs
Performance Transparency	Allows the system to be reconfigured to improve performance as loads vary
Scaling Transparency	Allows the system and applications to expand in scale without change to the system structure or the application algorithms

PROBLEM

Design of a Client-Server System for Banking

Problem: Design of a Client-Server System

- Input: an informal description of an application (e.g., banking application)



concerned with user interface

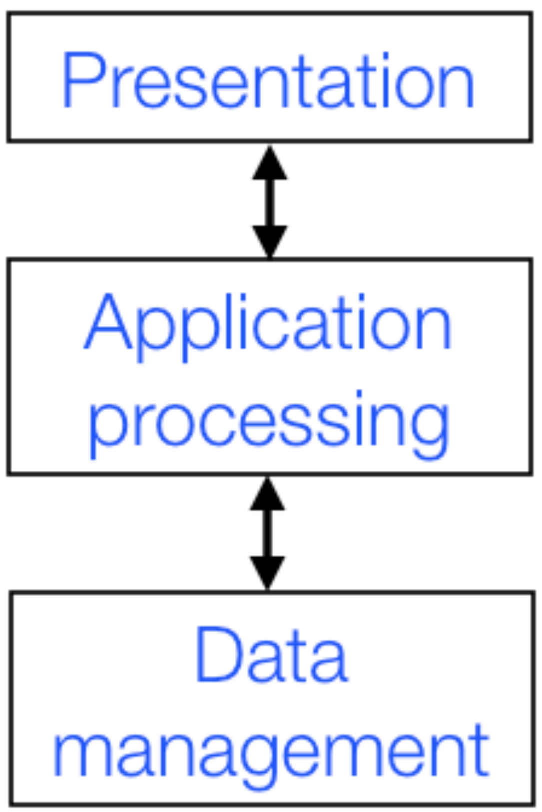
concerned with the detailed application-specific processing associated with the application

concerned with the persistent storage of the application (typically a DBMS)

- Output: client-server implementation of the application

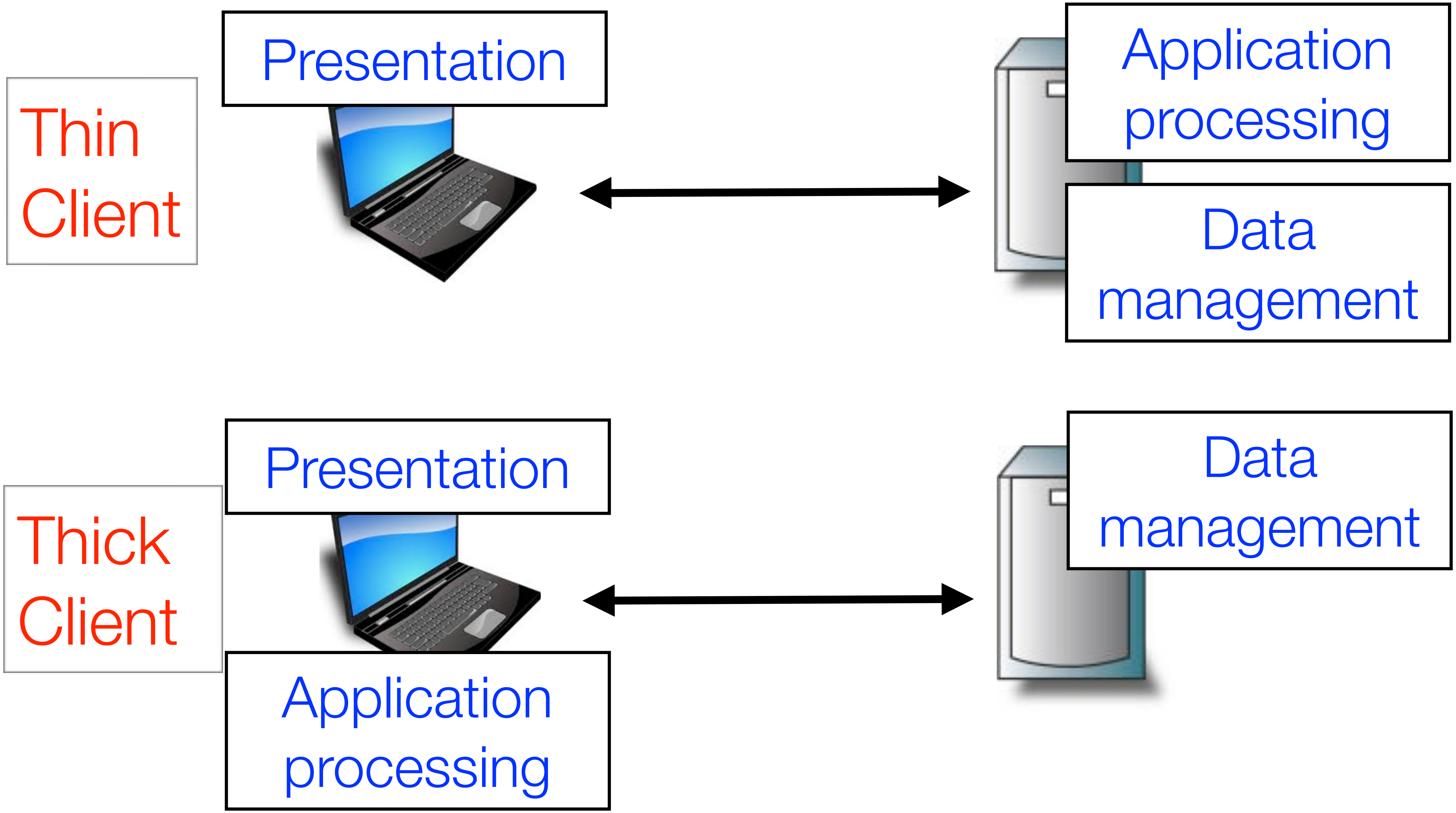
Solution 1: Two-Tier Client-Server Architecture

- Application organized as a server and a set of clients
- Two kinds of machines: client machines and server machines

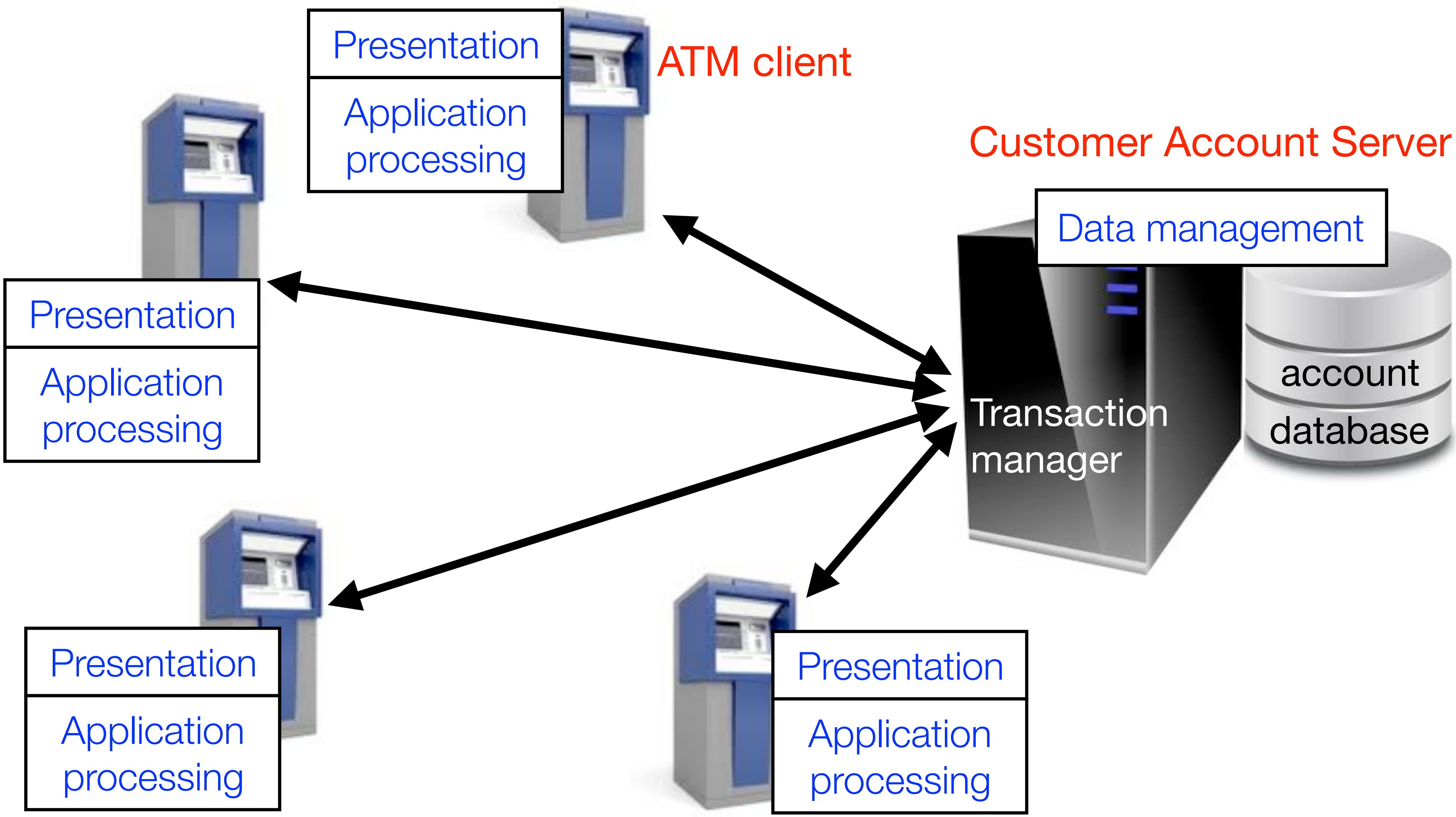


how to map
3 application layers
into a 2-tier architecture?

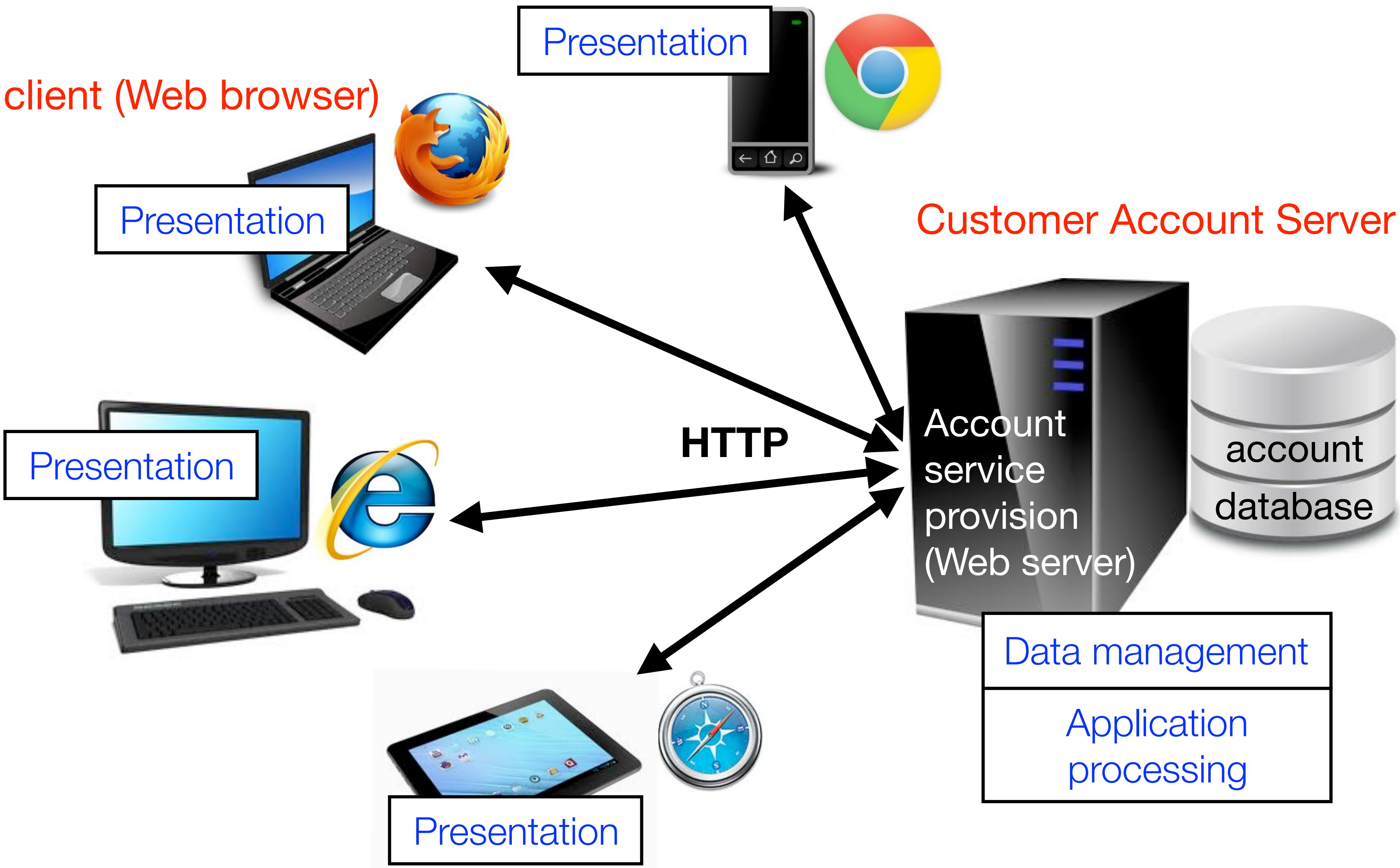
Thin VS Thick Client Model



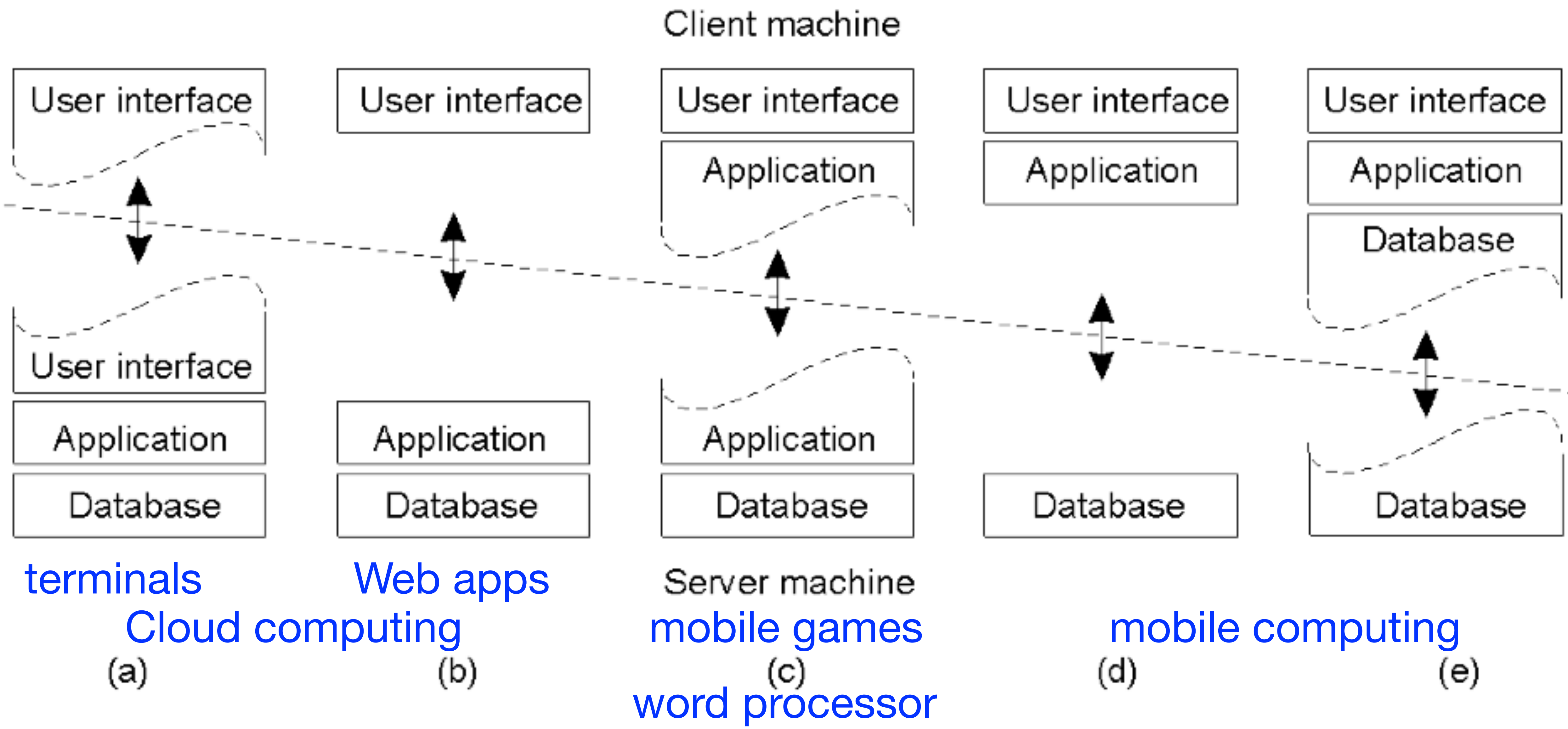
Example of Thick Client: ATM Banking System



Examples of Thin Client: Internet Banking System

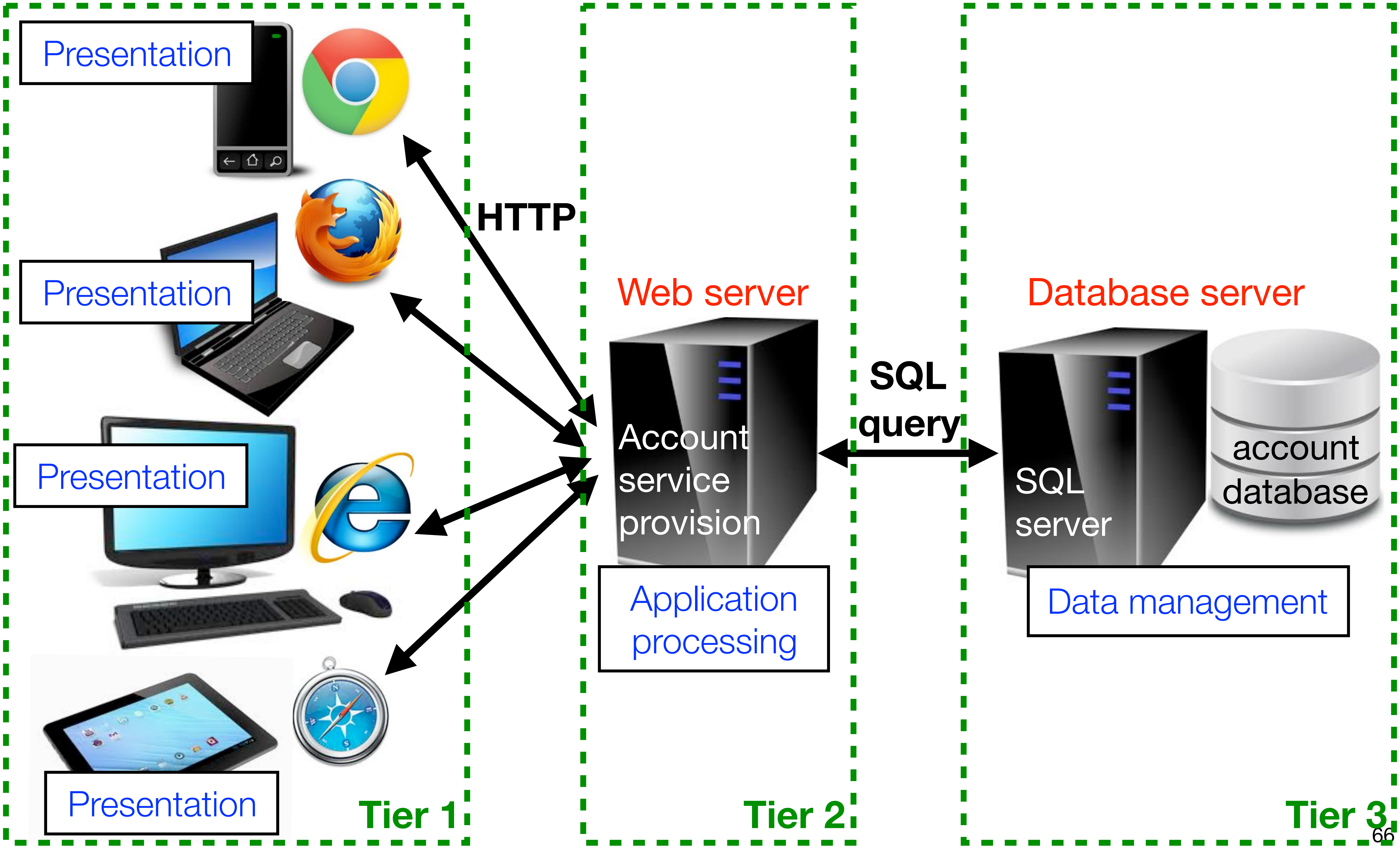


Alternative Two-Tier Client Server Organizations



Thin Client | Hybrid Client | Thick Client

Internet Banking System... in Practice



Thin or Thick? Thin

- Devices significantly enhanced with a plethora of networked services
- Access to legacy systems
- System management and administration
  - from admin perspective: system maintenance, security
  - from user perspective: not hassle with administrative aspects or constant upgrades
- More security
- Green IT (power saving --> cost saving)

pros

cons

- Heavy processing load on both server and network
- Less client-perceived performance (in highly interactive graphical activities such as CAD and image processing)
- Need to be always connected

Thin or Thick? Thick

- Better client-perceived performance (especially, in terms of image & video processing)
- (Partly) available offline
- Distributed computing (no single point of failures)
- Devices are becoming ever faster and cheaper:
  what is the point of off-loading computation on a server when the client is amply capable of performing it without burdening the server or forcing the user to deal with network latencies?

pros

cons

- System management and related costs
- Having more functionality on the client makes client-side software more prone to errors and more dependent on the client's underlying platform

Use of Client–Server Architectural Patterns

Two-tier client-server architecture with thin clients

- Legacy system applications that are used when separating application processing and data management is impractical; clients may access these as services
- Computationally intensive applications such as compilers with little or no data management
- Data-intensive applications (browsing and querying) with non-intensive application processing (example: browsing the Web)

Two-tier client-server architecture with fat clients

- Applications where application processing is provided by off-the-shelf software (e.g., Microsoft Excel) on the client
- Applications where computationally intensive processing of data (e.g., data visualization) is required
- Mobile applications where internet connectivity cannot be guaranteed
- Some local processing using cached information from the database is therefore possible

Multi-tier client-server architecture

- Large-scale applications with hundreds or thousands of clients
- Applications where both the data and the application are volatile
- Applications where data from multiple sources are integrated