ΕΠΛ 602:Foundations of Web Technologies

System Architectures

Lecture Outline

- Distributed system models
- The end-to-end argument for system design
- Basic system models and technologies for the Web

Why Models?

- Models are intended to provide an abstract, simplified and consistent description of a relevant aspect of distributed system design.

  - Descriptive
    - Provide a common vocabulary for use when describing systems
  - Guidance
    - Identify key areas in which services are required
  - Prescriptive
    - Define standard protocols and APIs to facilitate creation of interoperable systems and portable applications

System Models

Three different levels:

- **Physical Model**

Capture the hardware composition of a system in terms of the computer devices and their interconnecting networks

- **Architectural Model**

provide a high-level view of the distribution of functionality between components and the relationships between them

- **Fundamental Model**

vertical views representing some key aspects of distributed systems in an abstract way (e.g., failure or interaction models)

Physical Models

Basic definition (reminder) A distributed system is defined as one in which hardware or software components located at networked computers communicate and coordinate their actions only by passing messages.

Representation of the underlying hardware components that abstracts away from specific details of technologies

Three generation:

1. Early distributed systems
2. Internet-scale distributed systems
3. Contemporary distributed systems

Physical Models

Three generation:

Early distributed systems

Late 1970s- early 1980s, Ethernet

10-100 nodes + local area networking + Limited Internet

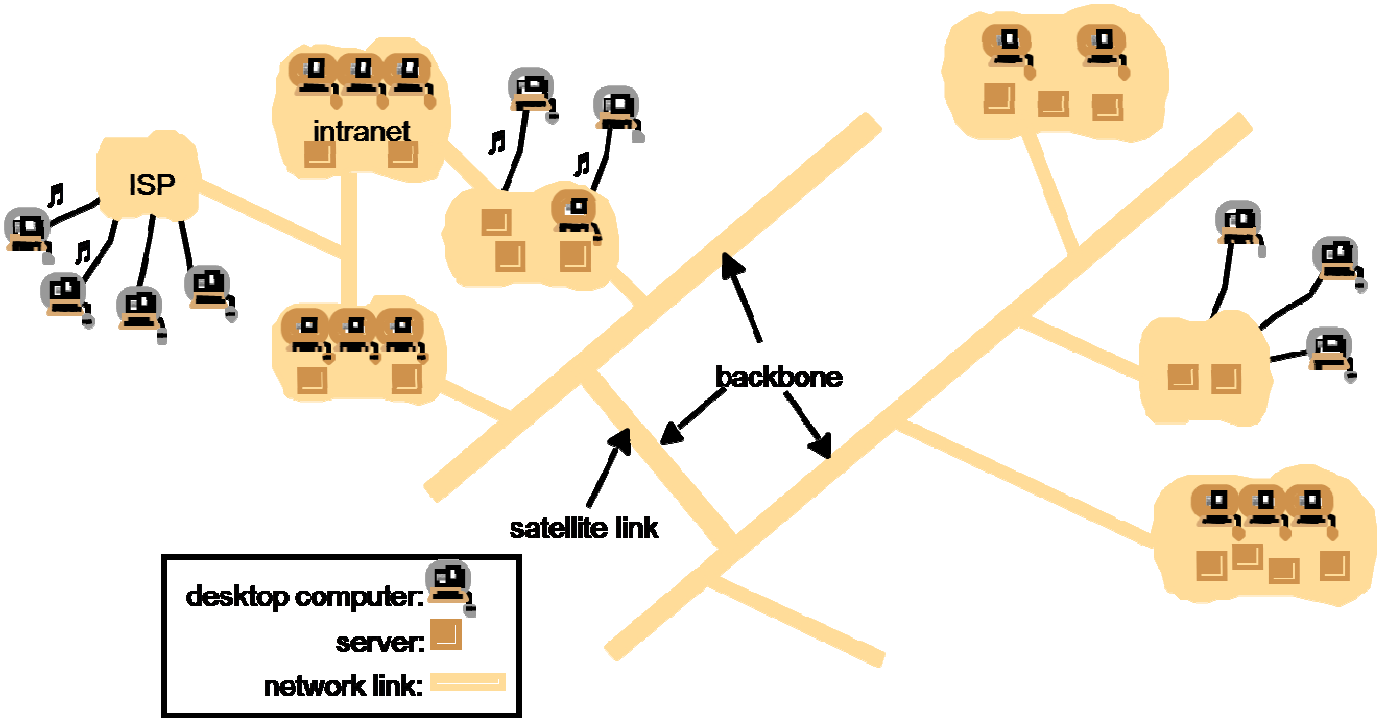
Services: shared local printers and file servers

Physical Models

Internet-scale distributed systems

1990s (Internet, Google 1996)

Network of networks



Physical Models

Contemporary distributed systems

Static (one physical location)
- mobile computing (discovery, spontaneous interoperation)

Discrete (not embedded in other physical entities)
- sensors, smart home

Autonomous (independent of other computers)
- cloud computing

Physical Models

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

Architectural Models

Architectural Models

- What are the entities that are communicating
- How they communicate (communication paradigms)
- What roles and responsibilities?
- How are they mapped on the physical distributed infrastructure (placement)

Architectural Models: what are the entities?

1. Processes

From a **system perspective**: processes coupled with appropriate interprocess communication paradigms

Nodes
Threads

Interface to Internet-level transport protocols UDP and TCP – sockets
UDP: message passing
TCP: two-way stream (producer/consumer paradigm)

Architectural Models: what are the entities?

2. Objects

From a **programming perspective**: objects

A computation consists of a number of interacting objects
Objects are accessed via interfaces (IDL)

Components

+dependencies between objects (assumptions made about other components/interfaces – contract), non-functional properties such as security + deployment strategies)

RPC (remote procedure call), RMI (extension to local method invocation that allows an object living in one process to invoke the methods of an object living in another process)
Higher-level programming abstractions: CORBA (interoperability)
Component-based: Enterprise JavaBeans, Fractal

Architectural Models: what are the entities?

3. Web services

Web services are distributed web applications that provide discrete functionality and expose it in a well-defined manner over standard Internet protocols to other web applications

(W3C)

A software application identified by a URI, whose interfaces and bindings are capable of being defined, described and discovered as XML artifacts. A web service supports direct interactions with other software agent s using XML-based message exchanges via Internet protocols

Architectural Models: what are the entities?

- Processes (nodes, threads)
- Objects (components)
- Web services

Architectural Models: Entities

- Embody a flow of control
  - E.g. process, thread
- Characterized by state, which includes:
  - Private data
  - State of execution
  - Bindings to other components (code and resource)

Architectural Models: how they communicate?

- Interprocess communication

Relatively low-level support offered (e.g., message-passing primitives, direct access to the API offered by the Internet (socket programming) and support for multicast)

Architectural Models: how they communicate?

- Remote invocation
  - **request-reply protocols** (underlying primitive): a pairwise exchange of messages from server to client (1st message: encoding of the operation to be executed by the server + arguments, 2nd message: encoded result (example: HTTP)
  - **remote procedure call (RPC):** servers offer a set of operations through a service interface and clients call these operations directly as if local
  - **remote method invocation (RMI):** an object can invoke a method in a remote object + object identity and pass objects as parameters

Architectural Models: how they communicate?

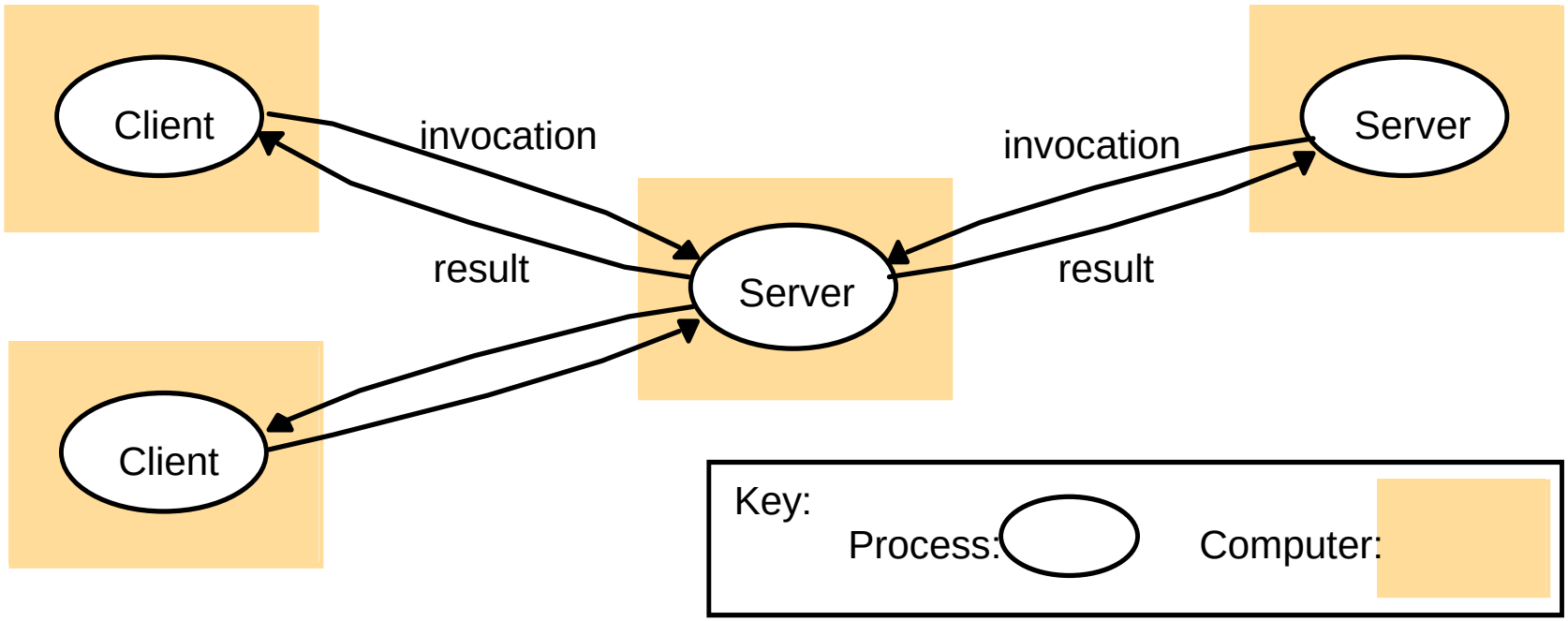
- Indirect communication
  - **group communication**: one to many, recipients elect to receive messages send to a group by joining the group
  - **publish/subscribe:** or distributed event-based systems (a matching service)
  - **message queues:** point-to-point service, producers send messages to a queue and consumers receive messages (or get notified)
  - **tuple spaces:** processes place arbitrary items of structured data (aka tuples) in a persistent tuple space; other process can read or remove tuples from the tuple space by specifying patterns of interest
  - **distributed shared memory (DSM):** an abstraction for sharing data between processes that do not share physcical memort

Architectural Models: summary so far

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

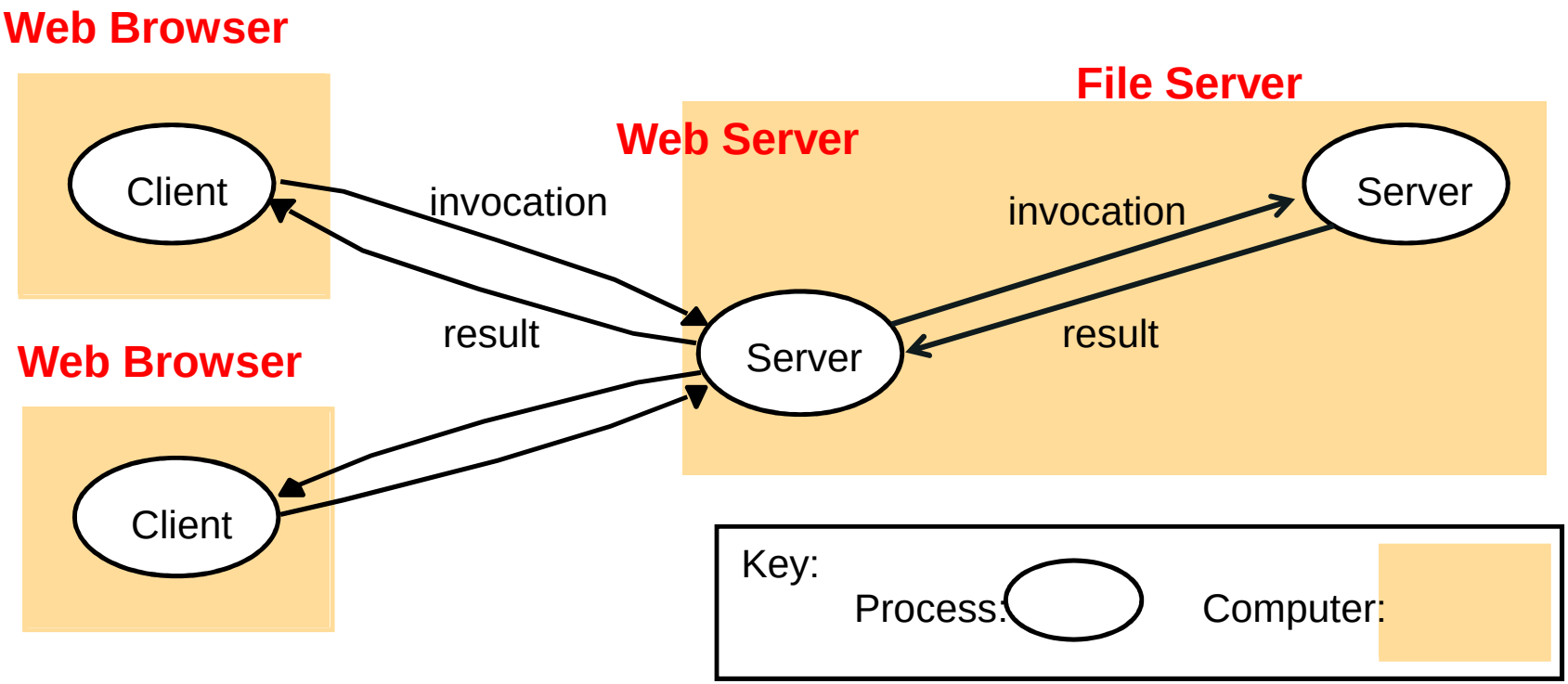
Architectural Models: roles?

Client-server



Architectural Models: roles?

Client-server

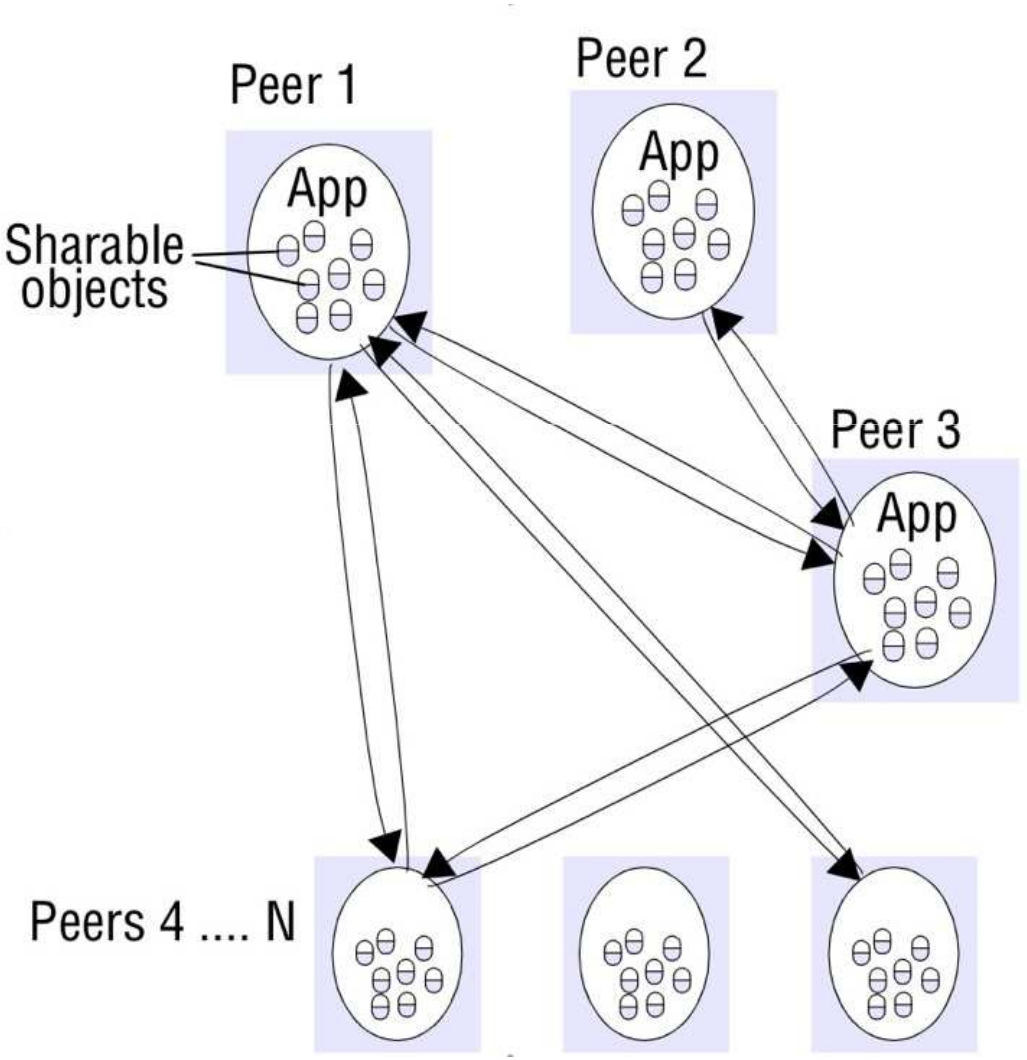


Architectural Models: roles?

Peer-to-peer

Address scalability,

Exploit resources (both data and hardware) at the edge of the network (users)



Architectural Models: placement

Where to place the entities (e.g., objects or services) in terms of machines or processes within machines

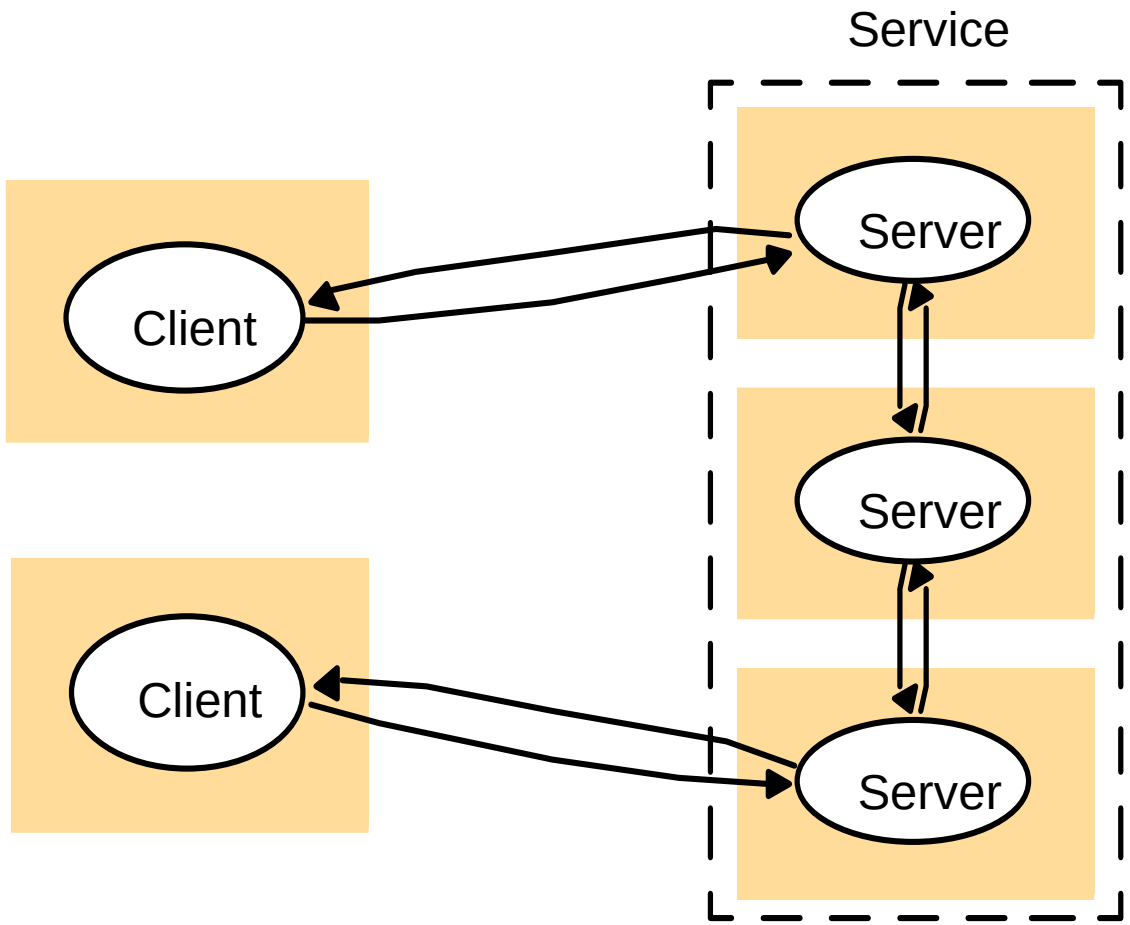
Issues: performance, security, reliability/availability
Current load, communication patterns, quality of communication, etc

Architectural Models: placement

Mapping of services to multiple servers

Partition (example: web servers)
or
Replicate (example: Sun NIS, computers on a LAN access authentication data

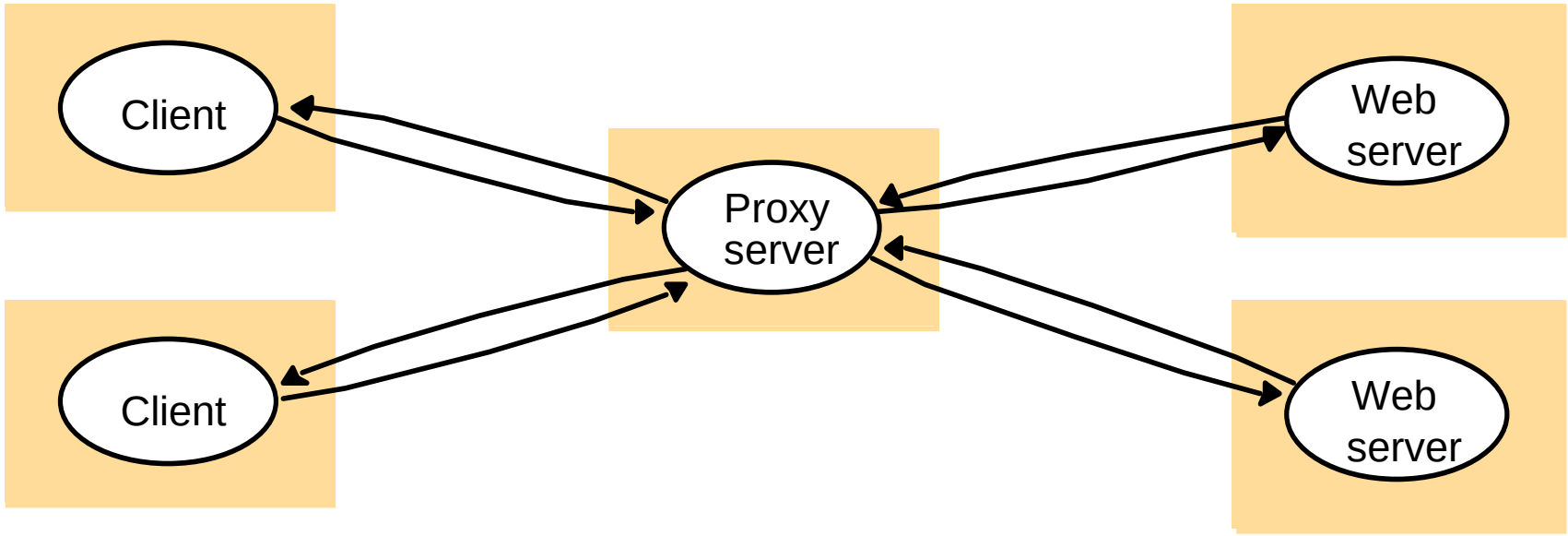
Cluster (search engines)



Architectural Models: placement

Caching

1. **Web browsers** maintain a cache of recently visited web pages and other web resources in their local file system and a special HTTP request to check with the server
2. **Web proxy servers** provide a shared cache (other roles: filtering, firewalls)

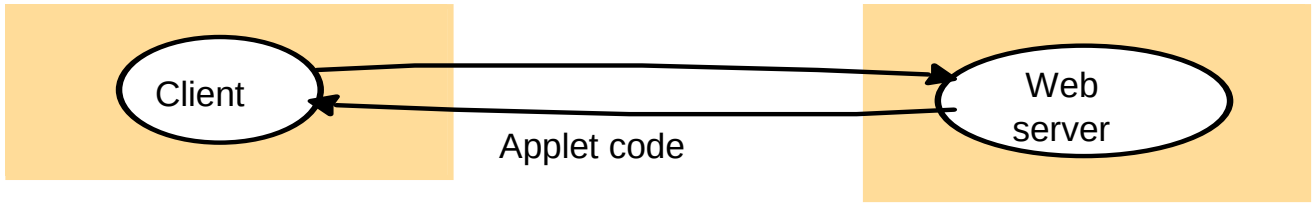


Architectural Models: placement

Mobile code

(running code locally vs remotely): applets

a) client request results in the downloading of applet code



b) client interacts with the applet

+ interactive response time

+ require downloading additional functionality (push, applet that receives updates)

- security

Architectural Patterns

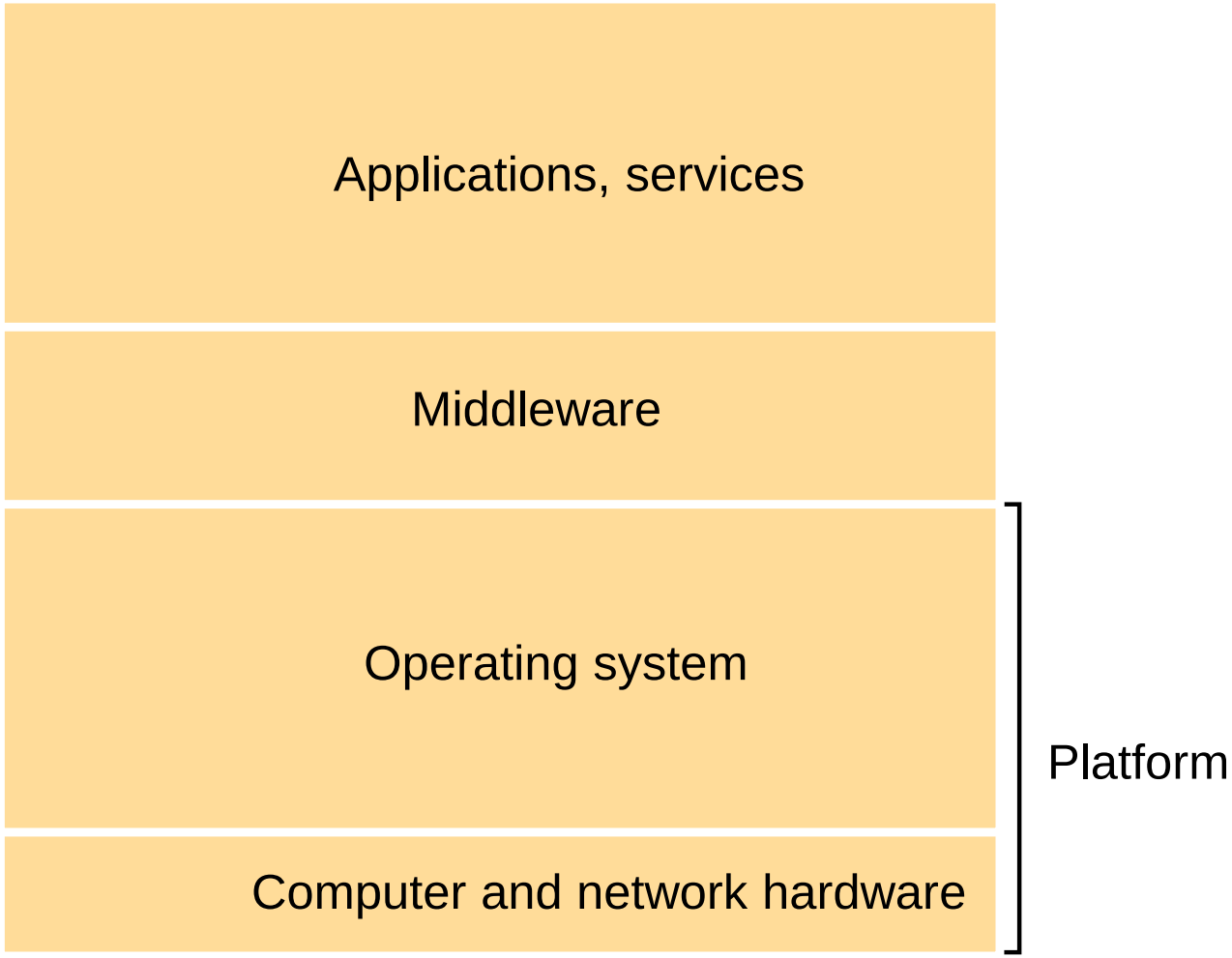
- Structures that have shown to work well

Architectural Patterns: Layering

A complex system is partitioned into a number of layers with a given layer making use of services offered by the layers below

Each layer offers a software abstraction, with higher layers unaware of implementation details of any other layers below them

Architectural Patterns: Layering



Architectural Patterns: Tiering

Complementary to layering

Layering: vertical organization into layers of abstractions

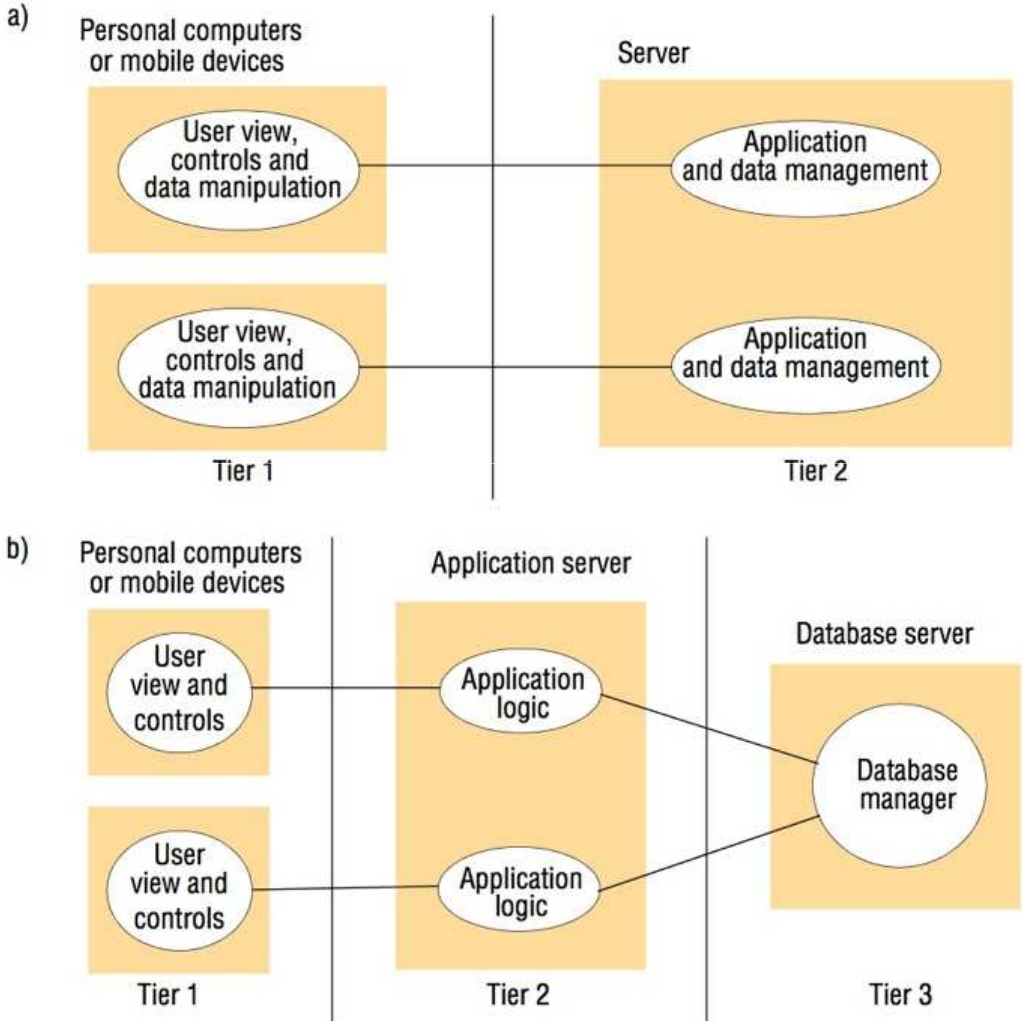
Tiering: organize functionality in a given layer and place it into appropriate servers (and physical nodes)

Architectural Patterns: Tiering

Example: presentation logic, application logic, data logic

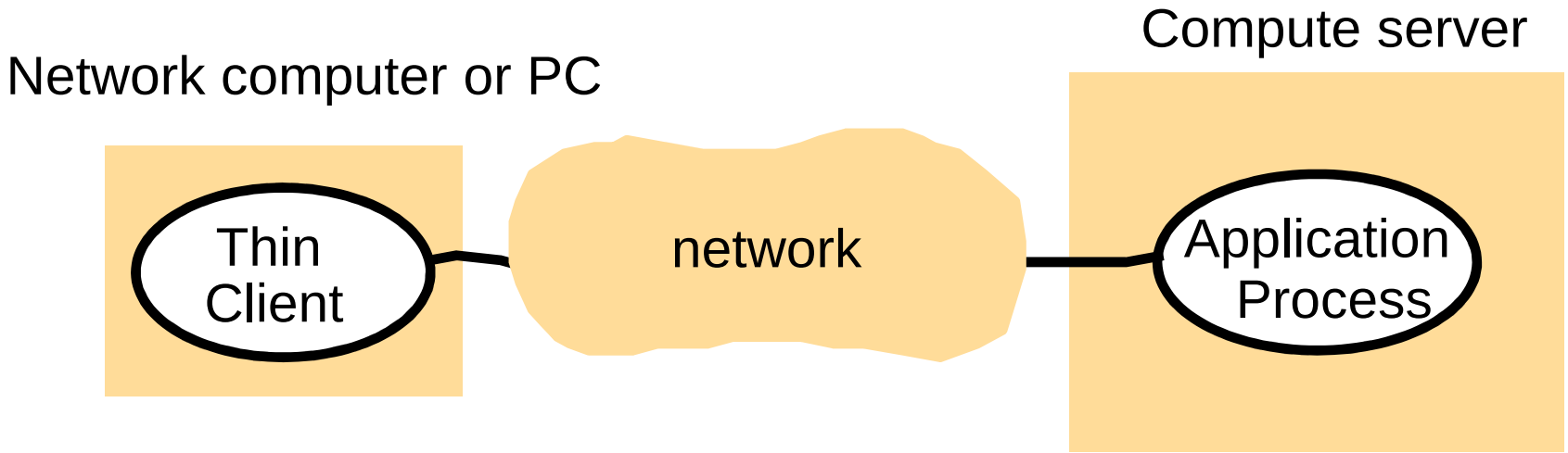
Generalizes to n-tier

See Ajax later today



Architectural Patterns: Thin clients

Thin client refers to a software layer that supports a user interface that is local to the user while executing applications program, or accessing services on a remote computer



Recent trend move complexity away from the end-user device towards services in the Internet (cloud computing)

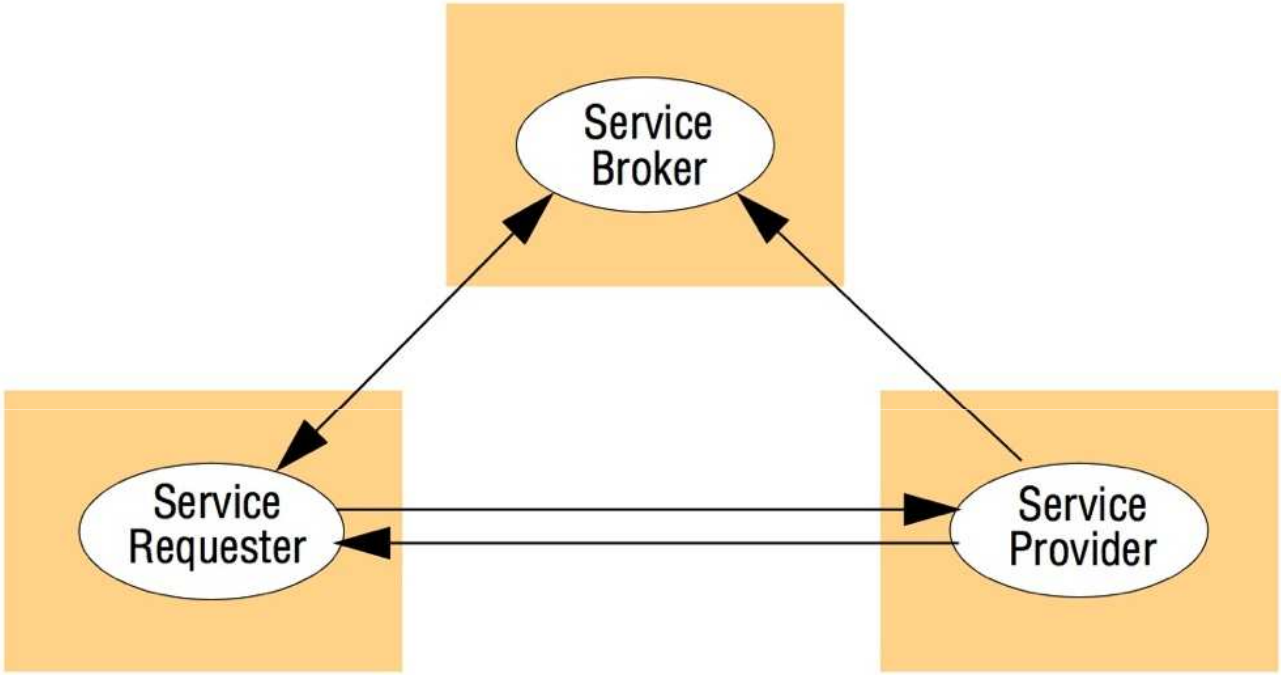
+ simple local devices (e.g., smart phones)
- Interactive graphical activities

Architectural Patterns: Proxy

Example 1
To support location transparency in RPC or RMI
A proxy is created in the local address space to represent the remote object; offers exactly the same interface; programmer makes calls to this proxy

Example 2:
Mobile computing

Architectural Patterns: Brokers



Registry in Java RMI
Naming services in CORBA, Web services

Architectural Patterns: Reflection

In a reflective system, standard service interfaces are available at the base level, but a **meta-level interface** provides access to the components and their parameters that are involved in the realization of the services

Techniques at the metalevel:
Intercept incoming messages (or invocations) to dynamically discover the interface offered by a given object (introspection) and to discover and adapt the underlying architecture of the system (intercession)

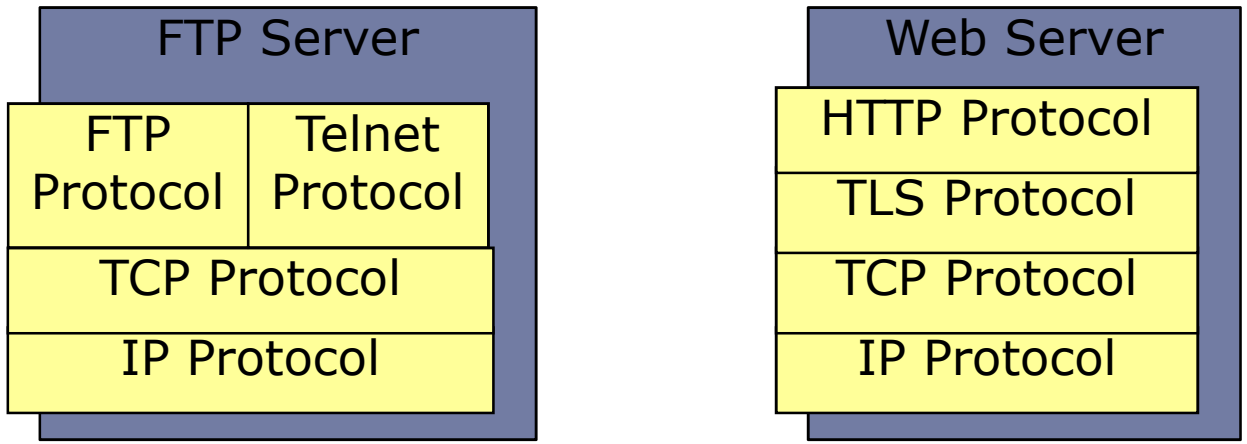
Lecture Outline

- Distributed system models
  - Physical Model
  - Architectural Model
    - Basic elements (entities, communication paradigms, roles, placement)
    - Patterns (layering, multiple-tiers, thin clients, proxies, brokers, reflection
    - Protocols, Middleware
  - Fundamental Models

Architectural Models: Protocols

- Implementation of a protocol that defines a set of capabilities
  - Protocol defines interaction with service
  - All services require protocols
  - Not all protocols are used to provide services (e.g. IP)

- Examples: FTP and Web servers



Architectural Models: Middleware

Middleware is a layer of software whose purpose is to mask heterogeneity and provide a convenient programming model to application programmers

Programming abstractions
+ infrastructure services

Architectural Models: Middleware

Major categories:	*Subcategory*	*Example systems*
Distributed objects (Chapters 5, 8)	Standard	RM-ODP
	Platform	CORBA
	Platform	Java RMI
Distributed components (Chapter 8)	Lightweight components	Fractal
	Lightweight components	OpenCOM
	Application servers	SUN EJB
	Application servers	CORBA Component Model
	Application servers	JBoss
Publish-subscribe systems (Chapter 6)	-	CORBA Event Service
	-	Scribe
	-	JMS
Message queues (Chapter 6)	-	Websphere MQ
	-	JMS
Web services (Chapter 9)	Web services	Apache Axis
	Grid services	The Globus Toolkit
Peer-to-peer (Chapter 10)	Routing overlays	Pastry
	Routing overlays	Tapestry
	Application-specific	Squirrel
	Application-specific	OceanStore
	Application-specific	Ivy
	Application-specific	Gnutella

Challenges

- Heterogeneity
  - See middleware
- Openness
  - See APIs
- Security
  - Encryption, types of attacks
- Scalability
  - In adding new nodes/users and service access
- Failure Handling
  - Deal with multiple types of failures
- Concurrency
  - Safe concurrent access
- Transparency

Design requirements

- Performance
  - Responsiveness, throughput, load balancing
- Quality of service (QoS)
  - time-critical applications (real-time apps)
  - guarantee certain level of quality delivered by the deadline
  - allocation of computation and communication resources
- Caching and replication
  - web caching: server provides expiration time
- Dependability
  - fault tolerance: redundancy, recovery
  - security

Sidestep: Course contract revised

Course: General

Διάλεξη:

Δευτέρα, 18.00-21.00, ΧΩΔ02 104

Ευαγγελία Πιτουρά, Επισκέπτρια Αναπληρώτρια Καθηγήτρια

Γραφείο: FST 01 B117

Ώρες Γραφείου: Τρίτη 13.30-15.00

Email: pitoura-AT-cs.ucy.ac.cy **or** –AT-cs.uoi.gr

Εργαστήριο:

Παρασκευή: 19:30-21, ΘΕΕ 01

Χριστόφορος Παναγιώτου, Ειδικό Εκπαιδευτικό Προσωπικό

Γραφείο: FST 02 B176

Email: panchris-AT-cs.ucy.ac.cy

Ιστοσελίδα Μαθήματος:

http://www.cs.uoi.gr/~pitoura/courses/epl602

Course: Content

Περιγραφή Μαθήματος

Θέματα Αρχιτεκτονικής και Σχεδιασμού Παγκόσμιου Ιστού Πληροφοριών, Ανασκόπηση Πρωτοκόλλων Διαδικτύου (TCP, IP, DNS), Σχεδιασμός Πρωτοκόλλων Παγκόσμιου Ιστού (HTTP), Εναποθήκευση Πληροφορικών Ιστού (Web Caching), Χαρακτηρισμός και Μοντελοποίηση Ιστού (Web Characterization), Δίκτυα Μετάδοσης Πληροφοριών (Content Distribution Networks), Ομότιμα Δίκτυα (Peer-to-Peer Networks).

Στόχοι Μαθήματος

Βασικός στόχος του μαθήματος είναι η κατανόηση των μοντέλων, αρχιτεκτονικών αλγορίθμων και πρωτοκόλλων σχετικών με το web ως ένα κατανεμημένο σύστημα μεγάλης πολυπλοκότητας. απόκτηση ικανότητας επίλυσης διαφόρων προβλημάτων με προγραμματισμό.

Συγκεκριμένα:

- Θεμελίωση βασικών αρχών κατανεμημένου υπολογισμού και υπολογισμού στο web.
- Μελέτη γλωσσών, πλαισίων και ενδιάμεσου λογισμικού για προγραμματισμό στο web
- Ανάπτυξη μια εφαρμογής στο web

Course: Content

Ενδεικτική Ύλη

Το περιεχόμενο του μαθήματος θα καλύψει ενδεικτικά τα παρακάτω θέματα σε σχέση με το web:

1. Μοντέλα και Αρχιτεκτονικές
2. Βασικά Πρωτόκολλα
3. Επικοινωνία, Πρότυπα, Διαδικασίες
4. Αναπαράσταση Δεδομένων
5. Υπηρεσίες Διαδικτύου
6. Κοινωνικά Δίκτυα
7. Υπερκείμενα (overlay) Δίκτυα και δίκτυα ομότιμων κόμβων
8. Μηχανές Αναζήτησης

Τρόποι Διδασκαλίας

Η διδασκαλία του μαθήματος στηρίζεται σε διαλέξεις (3 ώρες ανά εβδομάδα) και στο εργαστήριο (1 + ½ ώρα ανά εβδομάδα). Οι διαφάνειες από τις διαλέξεις θα είναι διαθέσιμες στην ιστοσελίδα με σκοπό να βοηθήσουν τον φοιτητή στη μελέτη της σχετικής ύλης.

Course: Sources

Βιβλιογραφία

Δεν υπάρχει *ένα* βασικό βιβλίο για το μάθημα.

Για τις βασικές αρχές κατανεμημένου υπολογισμού και υπολογισμού στο web, θα χρησιμοποιηθεί το βιβλίο

- George Coulouris, Jean Dollimore, Tim Kindberg and Gordon Blair, *Distributed Systems: Concepts and Design,* Addison Wesley, 5th Edition, May 2011

Για θέματα προγραμματισμού εφαρμογών web, θα χρησιμοποιηθεί υλικό από τα βιβλία:

- Leon Shklar and Rich Rosen, Web Application Architecture: Principles, Protocols and Practices, 2nd Edition, Willey 2009
- Paul Deitel, *Internet & World Wide Web: How to Program*, 4th Edition, Pearson 2009

Για θέματα διαχείρισης δεδομένων

- Serge Abiteboul, Ioana Manolescu, Philippe Rigaux, Marie-Christine Rousset and Pierre Senellart, Web Data Management, Cambridge University Press 2011.

Θα χρησιμοποιηθούν επίσης σχετικά άρθρα από τη διεθνή βιβλιογραφία
Τέλος, οι σημειώσεις του μαθήματος (διαφάνειες) θα είναι διαθέσιμες στην ιστοσελίδα του μαθήματος.

Course: Requirements

Αξιολόγηση

Στόχος είναι η ενεργή συμμετοχή του φοιτητή στο μάθημα και η ουσιαστική κατανόηση της ύλης του μαθήματος. Για το στόχο αυτό η αξιολόγηση θα γίνει ως εξής.

Εβδομαδιαίες Ασκήσεις 30%
Ομαδική Εργασία Εξαμήνου 35%
Γραπτή Ενδιάμεση Εξέταση 20%
Γραπτή Τελική Εξέταση 25%

Τα ποσοστά αθροίζουν σε 110 (περιλαμβάνεται ένα θετικό bonus).

Course: Requirements

Εβδομαδιαίες Ασκήσεις 30%

Στόχος είναι η κατανόηση της ύλης που διδάσκεται σε κάθε διάλεξη.

Οι ασκήσεις **θα ανακοινώνονται την Τρίτη στην ιστοσελίδα του μαθήματος**, θα καλύπτουν την ύλη της διάλεξης της Δευτέρας και θα παραδίδονται στο μάθημα της επόμενης εβδομάδας.

Αργοπορημένες ασκήσεις δε θα γίνονται δεκτές.

Κάποιες από αυτές τις εργασίες μπορεί να είναι ομαδικές.

Course: Requirements

Ενδιάμεση Εξέταση 20%

Η ενδιάμεση γραπτή εξέταση υπολογίζεται ότι θα διεξαχθεί τέλη Φεβρουαρίου (**εβδομάδα 20-24 Φεβρουαρίου**).

Η ακριβής ημερομηνία θα ανακοινωθεί σε μεταγενέστερο στάδιο.

Τελική Εξέταση 25%

Η τελική εξέταση θα καλύπτει την ύλη που δεν καλύφτηκε στην ενδιάμεση εξέταση

Course: Requirements

Ομαδική Εργασία Εξαμήνου 35%

Αφορά την ανάπτυξη μιας εφαρμογής με χρήση κάποιων από των τεχνολογιών που καλύφτηκαν στο μάθημα.

Ενδεικτικά θέματα θα ανακοινωθούν την πρώτη εβδομάδα του Φεβρουαρίου στην ιστοσελίδα του μαθήματος.

Η εργασία θα είναι σε ομάδες έως 3 ατόμων.

Ο σχεδιασμός της εφαρμογής θα παρουσιαστεί σε διάλεξη πριν το Πάσχα και η τελική εφαρμογή στην τελευταία διάλεξη του εξαμήνου.

Σημειώνεται ότι στις ομαδικές εργασίες κάθε φοιτητής θα βαθμολογείται ατομικά, δηλαδή, δε είναι απαραίτητο όλα τα μέλη να πάρουν τον ίδιο βαθμό.

Course: General

1. Κάθε φοιτητής δικαιούται να παρακολουθεί τις διαλέξεις και τα εργαστήρια χωρίς ενοχλήσεις και αδικαιολόγητες διακοπές. Παρακαλούνται λοιπόν όλοι να διαφυλάξουν το δικαίωμα αυτό, σεβόμενοι τον χρόνο ενάρξεως και λήξεως των μαθημάτων, την καθαριότητα των αμφιθεάτρων και των εργαστηριακών χώρων και γενικώς την ακαδημαϊκή ελευθερία.
2. Οι φοιτητές καλούνται να σεβαστούν τους κανόνες πνευματικής ιδιοκτησίας αναφορικά με την αντιγραφή και χρήση λογισμικού και την φωτοαντιγραφή βιβλίων.
3. Η απουσία από εξέταση και η καθυστέρηση παράδοσης εργασιών γίνονται αποδεκτές μόνο σε έκτακτες περιστάσεις και κατόπιν προηγουμένης συνεννοήσεως με τον καθηγητή. Ο καθηγητής δεν υποχρεούται να δώσει εξετάσεις σε άτομα που απουσίασαν αδικαιολόγητα από μία εξέταση. Η καθυστερημένη παράδοση εργασιών συνεπάγεται βαθμολογική ποινή, ασχέτως της ποιότητας της παραδεδομένης εργασίας.
4. Ενστάσεις στα αποτελέσματα εξετάσεων και στην βαθμολογία εργαστηριακών ασκήσεων γίνονται δεκτές βάσει των κανονισμών του Πανεπιστημίου.
5. **Η αντιγραφή ή η προσπάθεια αντιγραφής μεταξύ φοιτητών σε εξετάσεις ή εργασίες, απαγορεύεται αυστηρά.** Τυχούσες αντιγραφές θα συνεπάγονται την αποπομπή των αναμεμιγμένων φοιτητών από την τάξη, τον μηδενισμό του βαθμού τους στις εν λόγω εξετάσεις ή εργασίες και την καταγγελία τους στο Συμβούλιο του Τμήματος για την εφαρμογή περαιτέρω πειθαρχικών κανόνων.

End of Sidestep

Fundamental Models

- Fundamental properties in processes and communication, shared among different architectures discussed previously

- Interaction
- Failures
- Security

Interaction model: issues

- sequential vs. distributed algorithms (1) *timing* (rate it proceeds and timing of transmission cannot be predicted), (2) *distributed state* (each process its own private state)

Two significant factors:

1. performance of communication channels
   - latency: transmission, access, os
   - bandwidth
   - jitter: variation among messages
2. clocks and timing events
   - clock drift (deviation from a perfect reference clock)
   - Synchronization (GPS, accuaracy 1 microsec)

Interaction Model: types

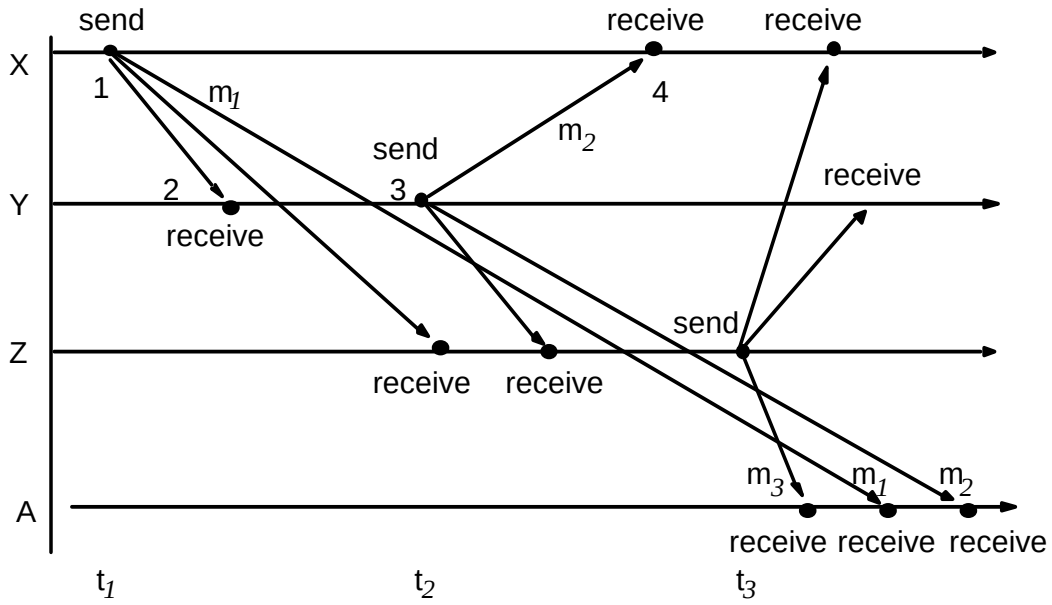
- synchronous distributed systems
  1. lower and upper bounds for execution of a step
  2. message transmission in bounded time
  3. clock drift rate is bounded

  - Time-outs: failures can be detected when bounds are exceeded
  - accomplished by allocating sufficient resources
- asynchronous distributed systems (e.g, the Internet)
  - no bounds on process speed, message delay, clock drift rate
  - failures are harder to detect
  - performance can't be guaranteed
  - consequence of resource sharing

  Simple solutions (browsers allow users to do other things while waiting)

Interaction Model: event ordering

- Relative ordering might be more important than exact time
- logical clocks--ordering events without physical clocks



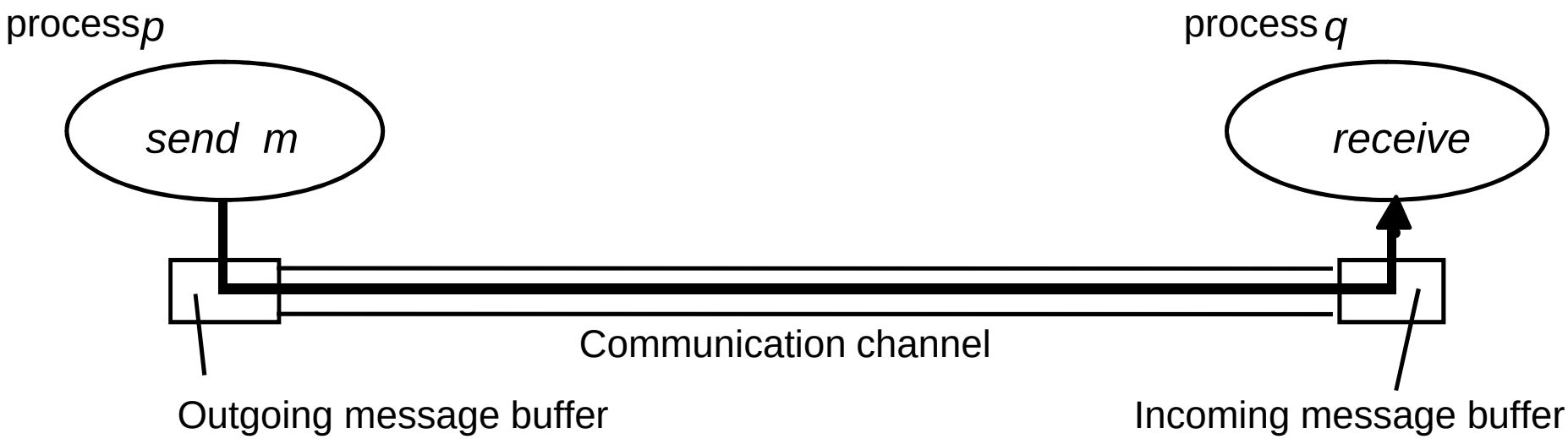
Users X, Y, Z and A
X: Meeting
Y, Z: Re: Meeting

User A:

1. From Z: Re: Meeting
2. From X: Meeting
3. From Y: Re: Meeting

Failure Model

- Failure of processes or communication channels



Failure Model

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.

Failure Model: Timing failures

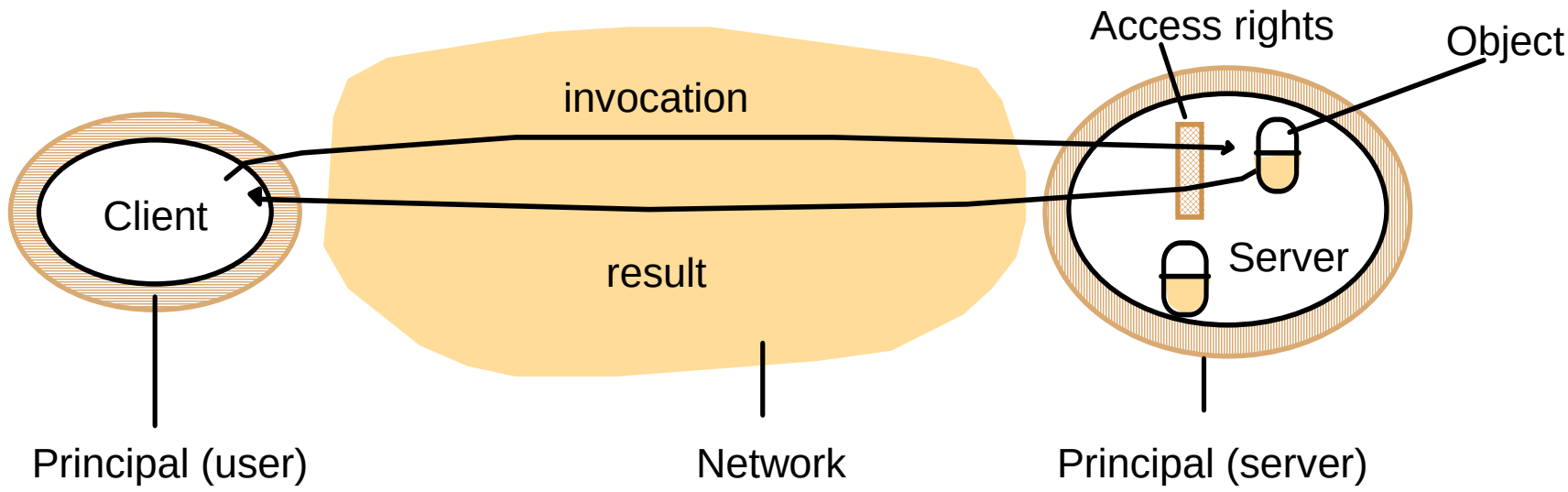
Class of Failure	*Affects*	*Description*
Clock	Process	Process's local clock exceeds the bounds on its rate of drift from real time.
Performance	Process	Process exceeds the bounds on the interval between two steps.
Performance	Channel	A message's transmission takes longer than the stated bound.

Failure Model

- masking failures
  - hiding--use another server to respond
  - converting it into more acceptable--drop the packet if it is corrupted
- reliable one-to-one communication
  - validity: eventually delivered
  - integrity: content not corrupted or duplicated

Security Model

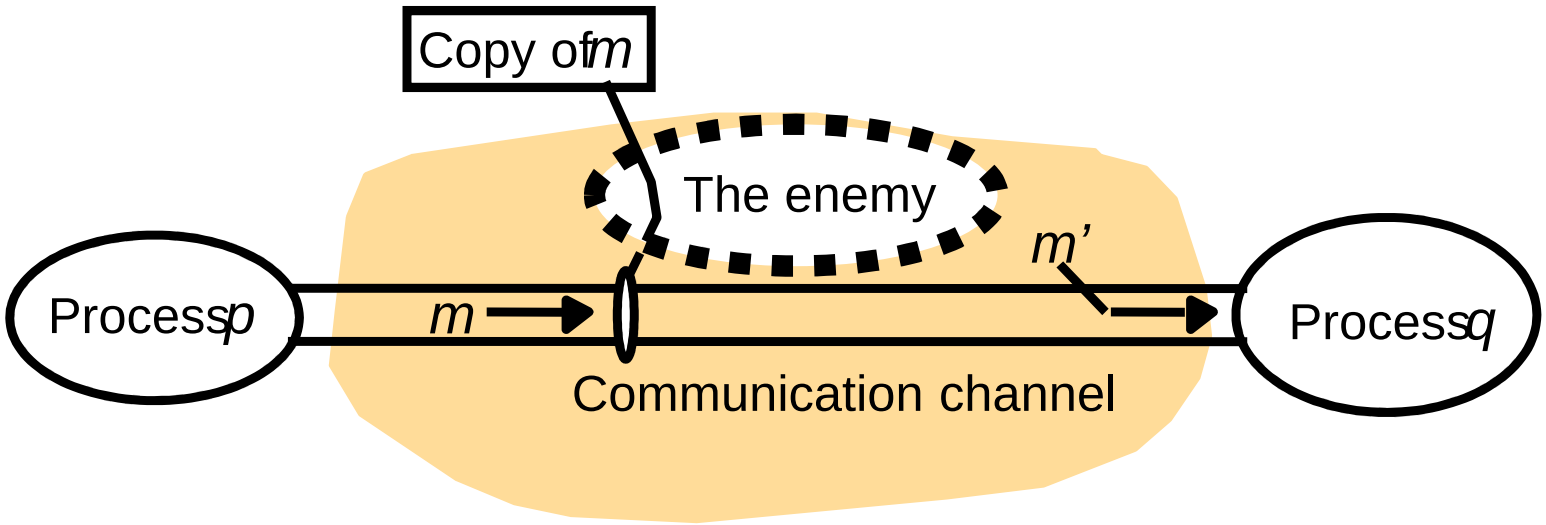
- Protecting objects:
  - authorization (access rights to principals)
  - authentication (identity of parties/principals)



Security Model

Enemy or adversary

Attack: Threat to processes & communication channels



Security Model

- Threat to processes (no reliable knowledge of the source of a message) & communication channels (an enemy can copy, alter of inject messages)
- Denial of service
- Cryptography: science of keeping messages secret
  - encryption: process of scrambling a message to hide its content
  - secret keys--large numbers that are difficult to guess
  - authentication--encrypt the identity, check the decrypted identity
  - secured channels--authentication, privacy/integrity, time stamp to prevent replaying and reordering

The end-to-end argument

Question

- How to partition the functionality/roles between the various components in a client-server system

Introduction

- Choosing the proper boundaries between functions is a primary activity of the computer system designer.

- In systems involving communication:
  - Modular boundary around communication subsystem
  - Firm interface between communication subsystem and the rest of the system.

Introduction

- Where do we implement the system functionality:
  - In the communication subsystem
  - In the clients of the communication subsystem
  - As a joint venture
  - Redundantly

End-to-end argument

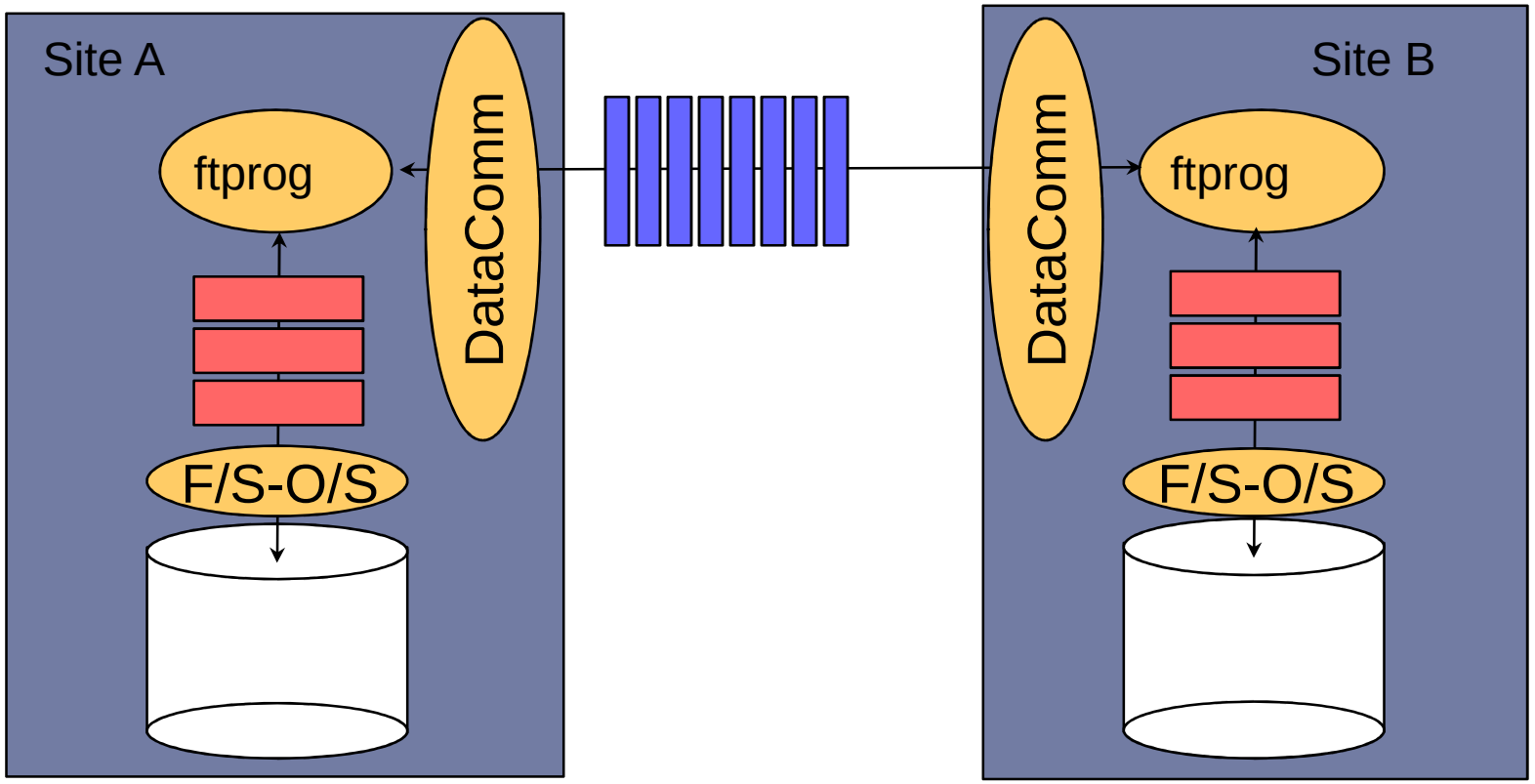
- "The function in question can be completely and correctly be implemented only with the knowledge and help of the application standing at the endpoints of the communication system".

- Therefore, providing that questioned function as a feature of the communication subsystem itself is not possible
  - Sometimes, an incomplete version of the function be provided by the communication subsystem may be useful as a performance enhancement.

Careful File Transfer

- Move file from computer A to computer B without damage.
- Steps taken:
  1. **At host A**, the file transfer program *calls the file system* to read the file from disk. The f/s passes the file to the file Xfer program in fixed-sized blocks chosen to be disk format independent.
  2. **At host A**, the ftprog *asks the data communication system* to transmit the file using some communication protocol that involves splitting the data into packets. The packet size is typically different from the file block size and the disk track size.
  3. The data communication network moves packets from A to B.
  4. **At host B**, the *data communication program* removes packets from the protocol and hands the contained data to a second part of the data transfer application operating on B.
  5. **At host B** the file Xfer program *asks the file system* to write the received data on the disk of host B.

Careful File Transfer



Threats to transaction

1. Hardware faults in the disk storage result to *reading* incorrect data.
2. File system software or file transfer program or data communication system make a mistake in buffering and copying the data of the file either at A or B.
3. Hardware processor or local memory have transient error while doing buffering and copying at A or B.
4. Communication system drops or changes bits in a packet or deliver a packet more than once.
5. Either of the hosts may crash part way through the transaction after performing an unknown amount of the transaction.

Dealing with threats

- Reinforce each of the steps along the way using duplicate copies, time-out and retry, carefully located redundancy for error detection, crash recovery, etc.
- Systematic countering of threat (2) requires writing correct programs…
  - Not all programs are written by the file-transfer programmer.
- Brute-force using tri-modular redundancy (do everything 3 times!) for the whole process…

End-to-end Check and Retry

- Suppose that a checksum is stored with each file, reducing the possibility of error to an acceptably negligible value.
- Then, transfer file from A to B.
- FT Application at B reads the file back to its memory, computes the checksum and sends the value back to A for comparison.
- If comparison fails, retry…

What if?

- The communication subsystem provides internally a guarantee for reliable data transmission, through:
  - Selective redundancy in the form of packet checksums
  - Sequence number checking
  - Internal retry mechanisms
- We can lower the probability of dropped bits to a very small number, and eliminate threat (4).
- Henceforth, we achieve a reduction of the frequency of retries by the file transfer application.
- What is the effect on the correctness of the file-transfer outcome?
  - Still in need to counter the other threats at the end-level

Conclusion

- To achieve careful file transfer, the application program that performs the transfer must supply a file-transfer-specific, end-to-end reliability guarantee.

- For the communication subsystem to go out of its way to be extraordinarily reliable does not reduce the burden on the application program to ensure reliability.

Performance Aspects

- So, should lower levels play no part in obtaining reliability?
- Consider a somewhat unreliable network, dropping a message in each hundred messages sent.
- What is the effect of this, as we transmit files of increasing size?
  - The probability that all packets of a file arrive correctly decreases exponentially with the file length (prove this).
  - So, the expected time to transmit the file grows exponentially with the file length.
- Performance of the file transfer application hurts!

Performance Trade offs

- The amount of effort to be put into reliability measures within the data communication system is an *engineering trade-off* based on performance, rather than a requirement for correctness.

(-) If communication subsystem is too unreliable, the file transfer application performance suffers.

(+) If communication subsystem is beefed up with internal reliability measures, those measures have a performance cost:

- Lost bandwidth to redundant data
- Added delay from waiting for internal consistency checks to complete
- And, after all, the end-to-end consistency check is still required, no matter how reliable the communication system becomes.

Performance Trade offs

- Using performance to justify placing functions in a low-level subsystem must be done carefully.
- Sometimes, the same or better performance can be achieved at the high level.
- Performing the function at the low level may:
  - be more efficient if the function can be performed with minimum perturbation of the machinery already included in the low-level.
  - Cost more because:
    - Most applications using the low-level subsystem do not need the function.
    - The low-level subsystem does not have adequate information, like the higher levels, to do the job efficiently.

Other Examples of the e2e Argument (skip)

- Acknowledgment of message delivery:
  - The communication network can easily return an ack to the sender, whenever a message is delivered to a recipient.
  - This is not very helpful for applications, since:
    - The application wants to know if the target host *acted* on the message (receipt does not directly translate to action). So, the application needs end-to-end ack.

Other Examples of the e2e Argument (skip)

- Secure transmission of data: if the data communication system performs encryption-decryption:
  - It will need to manage the keys.
  - Data will be vulnerable while passing from the communication system to the application.
  - Authenticity of the message must still be checked by the application. If the application performs the encryption, it can also do the authentication checks and keeps its keys.
  - Is the encryption of data by the communication subsystem necessary?

Other Examples of the e2e Argument (skip)

- Secure data transmission
  - Is the network trusted to manage the required encryption keys
  - the data will be vulnerable
  - Still, the app must do authentication!
  - Yet, some network encryption may be useful
    - Why?

- Duplicate Message Suppression
  - The app may accidentally originate duplicates!

Identifying the ends

- The end-to-end argument is *not* an absolute rule but rather a guideline:
- Suppose we use a communication subsystem to send voice packets:
  - To support two people carrying a real-time conversation (delays are disruptive, do not hide failures!)
  - To transport a speech message (to be listened to later).
- What are the choices we have?

Future of the end-to-end argument (Tussle in Cyberspace)

- The end-to-end argument provides:
  - Innovation
  - reliability
  - In the end, we have a "transparent" network
- This is threatened nowadays by:
  - Loss of trust (e.g., firewalls)
  - ISP control desires
  - 3rd parties wish to observe data flow
  - Caching, mirroring, etc
- Improve performance of today's apps in favor of new ones?

Future of the end-to-end argument (2)

- The end-to-end argument is still valid:
  - But needs redefinition in today's world…
- Evolution of existing apps is inevitable
- Keep the net open and transparent for new apps
- Cope with the loss of transparency

Web architecture

Web 101

Client/Server Model Client = Web browser/Server = Web server

1. Markup language for formatting hypertext documents (**HTML**)
2. A uniform notation schema for addressing accessible resources over the Internet (**URL**)
3. A protocol for transporting messages over the network (**HTTP**)

HTTP protocol: how requests and responses are transmitted and processed build on top of Core Internet Protocols (TCP/IP) (Layering)
✓*stateless*

Browser <-> DNS Server(s); Browser <-> Server

- Entities: Processes
- Request-Response Communication Paradigm
- Proxy (between clients and servers)
- Caching (at the browser and the proxy – server decides whether to cache + expiration)
- Replicated Servers (partition: each web server each own data
- Broker (DNS)

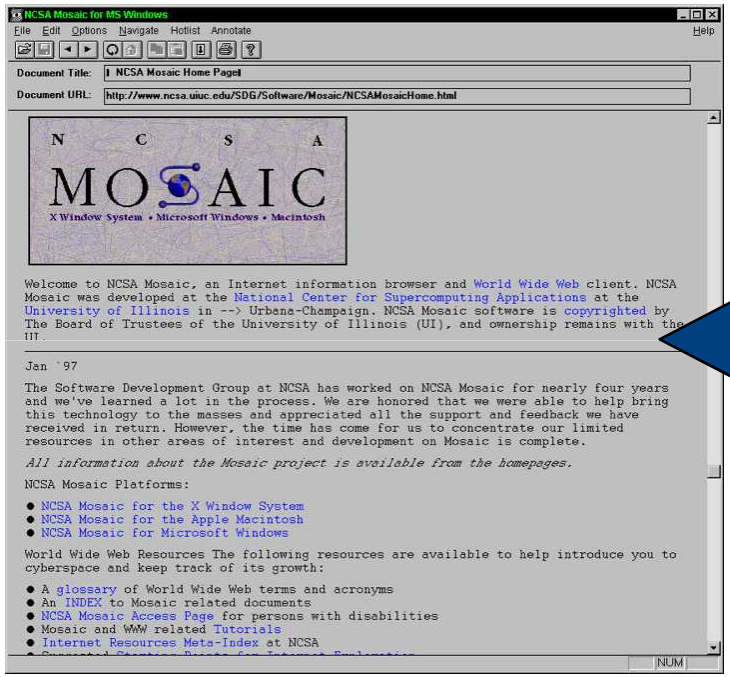
HTML and beyond

HTML a simple markup language to enable cross-referencing in documents through hyperlinks

Cascading Style Sheets (CSS) a mechanism for controlling the style for HTML rendering

XML is a meta-language for defining specialized mark-up languages
DTD, XML Schema
APIs for accessing XML parse trees, XSL, XQuery
XHTML: reformulation of HTML as an XML application
XHTML Mobile Profile

From Web pages to Web applications



The browser gradually became a “platform”

From Web pages to Web applications

Dynamic web
From building a web site -> design a web application

Web application: a client/server application that uses a web browser as its client program

Delivers interactive services through web servers distributed over the Internet or an intranet

A web application can present dynamically tailored content based on request parameters, traced user behaviors and security consideration

Example: online shopping cart

Web browsers

1. Generate and submit requests
2. Accept responses from web servers
3. Render the result

Cookies: a mechanism for maintaining state in browsers across multiple HTTP requests

JavaScript: to support event-handlers – custom code that executes when a browser event occurs

AJAX: a set of programming techniques that enable browsers to communicate asynchronously with web servers
Common uses: code injection (a background request is sent to the server to fetch content – that causes discrete updates to the content displayed in the browser or the data stored on the server)

Web servers

1. Virtual hosting: if the web server is providing service for multiple domains, determine the target domain
2. Address mapping: whether the request is for static or dynamic content
3. Authentication

Delivery of dynamic content

CGI: the CGI mechanism assumes that when a request to execute a CGI script arrives at the serve, a new “process” is spawned to execute a particular application program

Servlet API and JSP: Servlet areJava programs that have access to information in HTTP requests. JSP processors generate Java classes that extend the base class that implements the Servlet interface

Web services

Web services are distributed web applications that provide discrete functionality and expose it in a well-defined manner over standard Internet protocols to other web applications

Client -> web application

SOAP is an XML-based application layer protocol for constructing and processing web services requests and responses + Web Service Definition Language (WSDL), Universal Description, Discovery and Integration (UDDI) for registering and discovering web services

REST an architectural pattern as an alternative to SOAP
Items of interest on the web identified by their URL as resources, not static pages but calls to web applications
When accessed such resources return their representations that can be though of as the browser state

From Web from Web2.0

Incorporating applications that support user generated content, on-line communities and collaborative mechanisms for updating on-line content

Site visitors contribute information ranging from reviews and ratings to personal journals (blogs) to news.

Sophisticates web application technology incorporating *user authentication*, *access control* and *content management services*

Question?