Cloud computing and distributed systems

Chapter #2
System Models

Zeynep Yücel

Ca' Foscari University of Venice
zeynep.yucel@unive.it
yucelzeynep.github.io

System models

- Real-world functionality across conditions.
- Common properties and design problems in distributed systems.
- Explore descriptive models.
- Physical models of hardware.
- Architectural models of tasks.
- Fundamental models for analysis:
  - Interaction models for communication.
  - Failure models for operations.
  - Security models for protection.

Physical models

From the early days

- Physical model represents hardware elements.
- Baseline Model: Interconnected nodes.
- Distributed Systems: Emergence in 1970s-1980s with Ethernet.
- Internet-Scale Systems: Evolved in 1990s with Internet's growth.

Physical models

To current day

- Contemporary Systems: Mobile and cloud computing.
- Distributed Systems of Systems: Ultralarge-scale systems.

Physical models

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

Architectural models

- Architecture defines system structure.
- Good design meets current and future needs.
- Key concerns: reliability, adaptability, manageability, and cost-effectiveness.
- Major models:
  - Client-server models
  - Peer-to-peer approaches
  - Distributed objects/components
  - Event-based systems.

Architectural models

- Three sections:
  - Analyze architectural elements.
  - Explore composite patterns.
  - Review middleware platforms.
- We will ask the questions:
  - What entities communicate?
  - How do they communicate?
  - What are the roles and responsibilities?
  - How is the physical infrastructure mapping?
- Trade-offs in choices.

Architectural models

What entities communicate? From a system perspective

- Typically processes communicate (via interprocess communication).
- Some nuances
  - In primitive environments, nodes communicate.
  - Incorporation of threads in many systems.
- Modeling at the process level or thread level

Architectural models

What entities communicate? From a programming perspective

- Programming perspective: problem-oriented abstraction
  - Objects:
    - Distributed object-based approaches: object-oriented design
    - Problem decomposition: objects via interfaces
  - Components:
    - Developed for distributed objects: problem-solving
    - Similar to objects: tools for distributed systems
    - Clear interfaces and dependencies: easier development
  - Web services:
    - Relation to objects/components: encapsulation, interface access
    - W3C definition: software application, URI, XML, via Internet

Architectural models

How do the entities communicate? Three communication paradigms

- Three communication paradigms:
  - Interprocess communication.
  - Remote invocation.
  - Indirect communication.

Architectural models

How do the entities communicate? 1. Interprocess communication

- Interprocess communication (IPC): Methods for processes.
- Key aspects:
  - Low-level support.
  - Message-passing primitives.
  - Socket programming.
  - Support for multicast communication.

Architectural models

How do the entities communicate? 2. Remote invocation

- Remote invocation in distributed systems
  - Request-reply protocols
    - Client-server message exchange model
  - Remote Procedure Calls (RPC)
    - Invoke remote procedures as local
  - Remote Method Invocation (RMI)
    - RPC extension for distributed objects

Architectural models

How do the entities communicate? 3. Indirect communication

- Request-reply or remote communication: Direct communication and simultaneous presence.
- Indirect communication: Decoupling.
  - Space decoupling: unknown receivers.
  - Time decoupling: asynchronous presence.
- Key techniques for indirect communication:
  - Group communication
  - Publish-subscribe systems
  - Message queues
  - Tuple spaces
  - Distributed shared memory

Architectural models

How do the entities communicate? 3. Indirect communication – Key techniques

- Group Communication: One-to-many messaging.
- Publish-Subscribe Systems: Producers to consumers.
- Message Queues: Point-to-point communication.
- Tuple Spaces: Structured data storage.
- Distributed Shared Memory (DSM): Read/write abstraction.

Architectural models

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

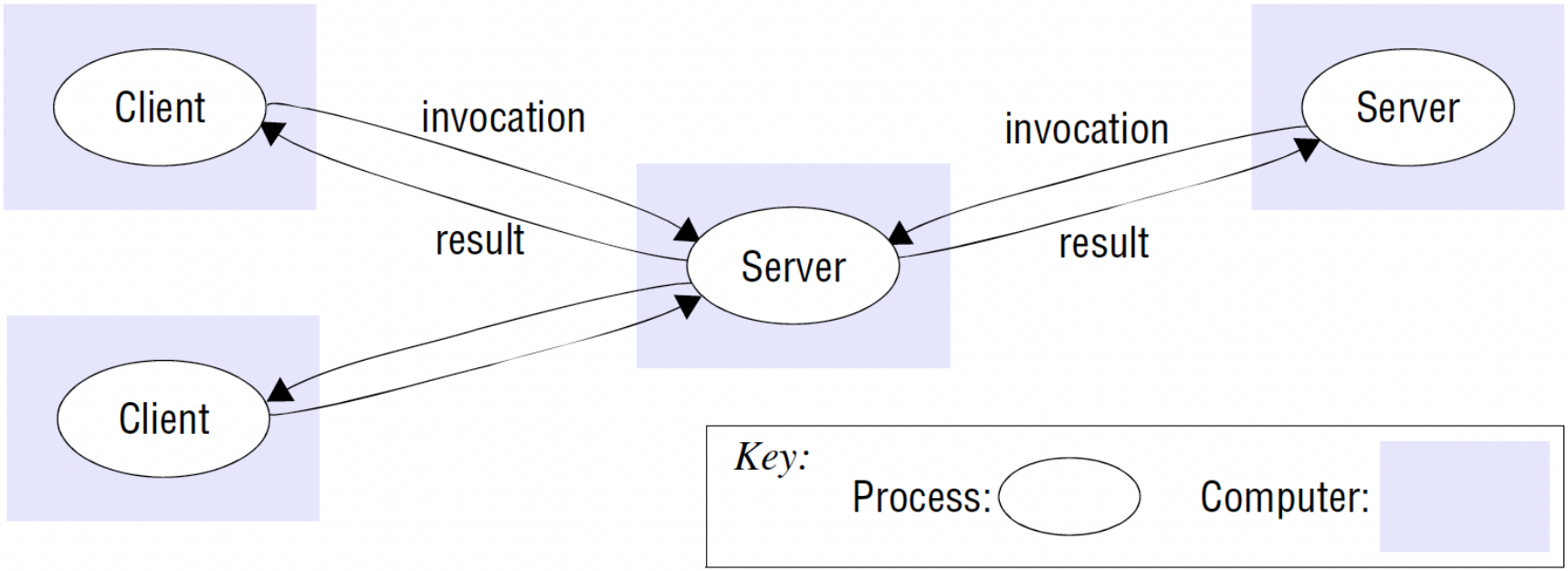
Roles and responsibilities

Client-server

- Distributed system roles, client-server, peer-to-peer etc.
  - Client communicates server and access shared resources.
  - Servers can also be clients.
    - Search engine responds queries.
    - Web crawlers access servers.

Roles and responsibilities

Client-server



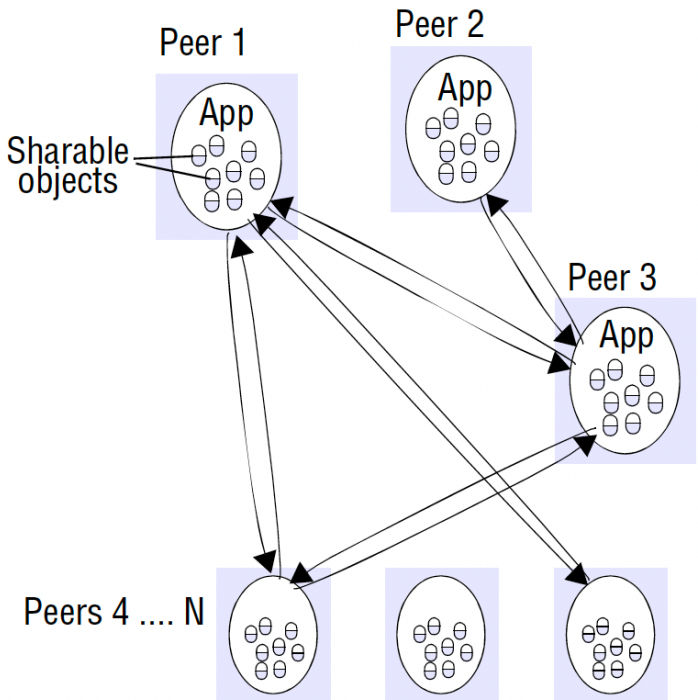
Roles and responsibilities

Peer-to-peer

- Distributed systems roles define architecture.
  - Peer-to-peer: equal interactions.
  - Uniform programs, shared interfaces.
  - Client-server: scalability limitations.
  - Peer-to-peer: leverage with user resources.
  - Collective data and hardware utilization.
  - Resources increase with users.
  - Effective peer-to-peer resource management.
  - Examples: Napster, BitTorrent.

Roles and responsibilities

Peer-to-peer



- Numerous peer processes.
- Distributed storage and processing.
- Object replication for resilience.
- Complex architecture compared to client-server.

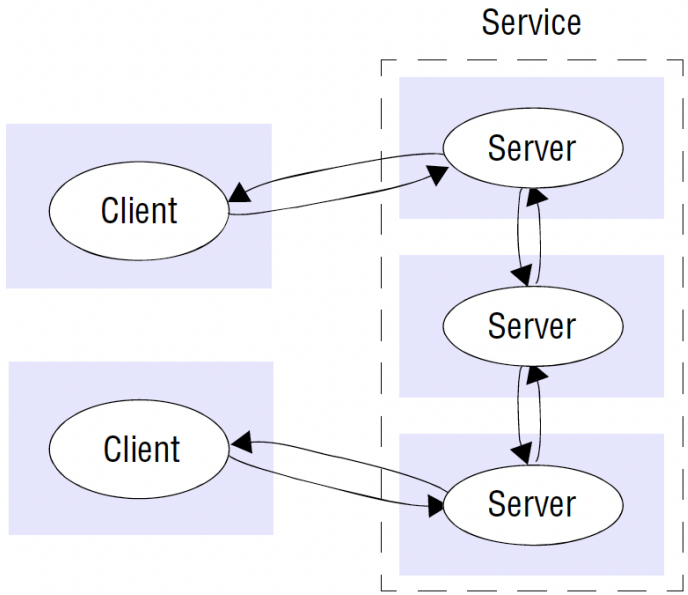
Placement

How is the physical infrastructure mapping?

- Placement aligns entities with infrastructure.
- Strategies include:
  - Multiple server mapping.
  - Caching.
  - Mobile code.
  - Mobile agents.

Placement

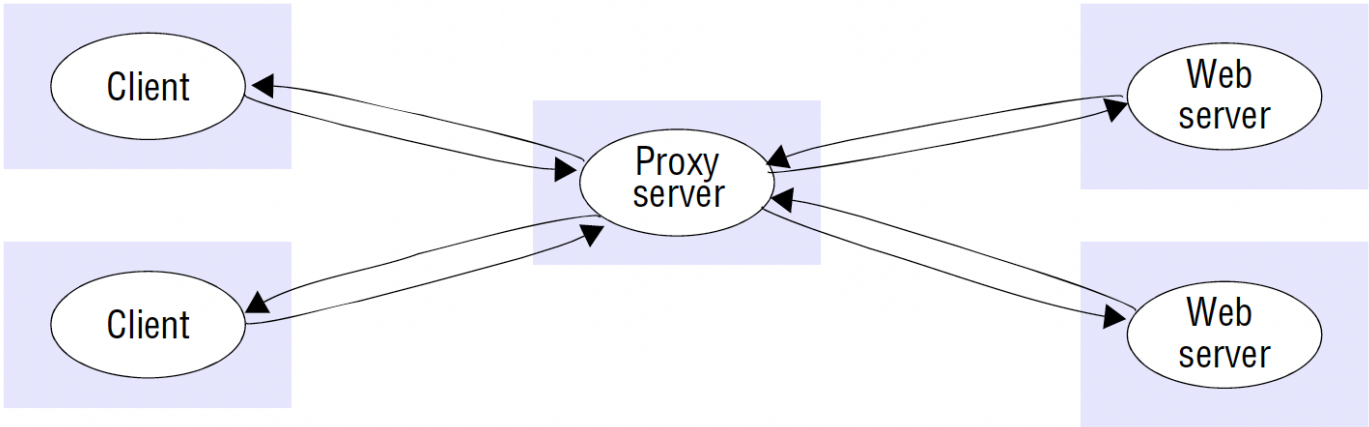
Mapping of services to multiple servers



- Multiple server processes interact.
- Data partitioning or replication.
  - Web: partitioned server resources.
  - NIS: replicated user authentication.
  - Clusters: integrated processing approach.

Placement

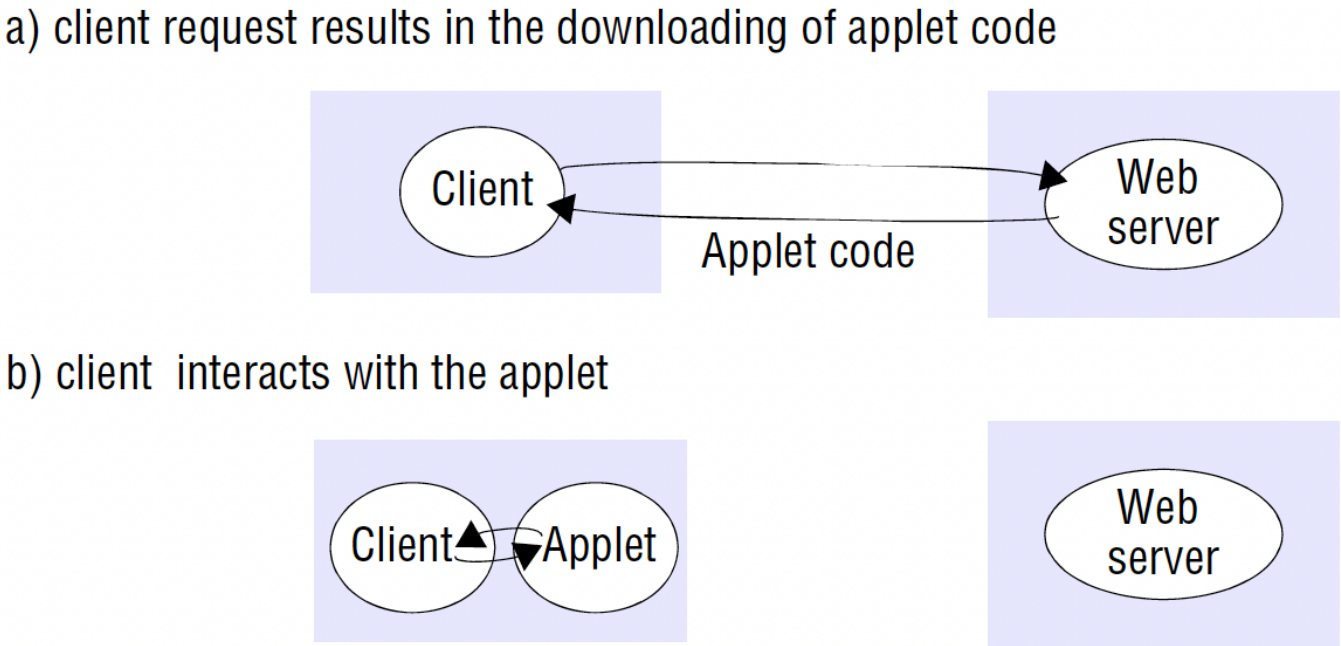
Caching



- Storing recent data objects.
- New objects are added and may replace older ones.
- Check cache before fetching.
- E.g. Web browsers store pages.
- Caches can be local/shared (at a proxy).
- Proxy servers enhance performance.

Placement

Mobile code



- Mobile code, e.g. downloaded browser applets.
- Local execution: interactive response improvement.
- Push model: server-initiated interactions.
- Stockbroker applet: price notifications, automation.
- Security risks: limit resource access.

Placement

Mobile agents

- Mobile agents: programs (code and data) moving between computers.
- Reduce costs: local invocations.
- Applicability limitations:
  - Security risks: identification necessary.
  - Vulnerability: access denial issues.
  - Alternative methods: web crawlers.

Architectural patterns

Layering

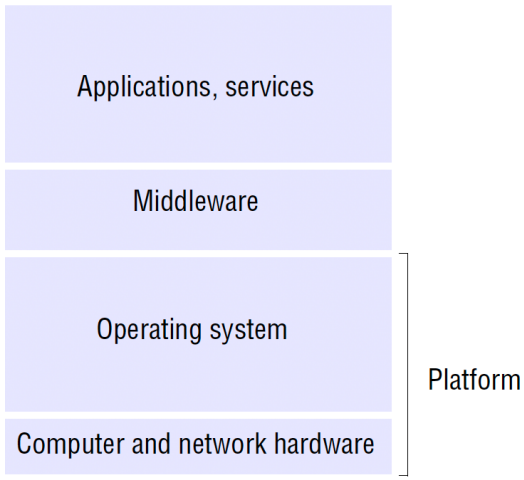


Figure: Software and hardware service layers.

- Layering: abstraction in complex systems.
- In distributed systems: vertical service organization.
- Platforms: foundational hardware/software layers.
- Middleware: interaction simplification, programming support.

Architectural patterns

Tiered architectures

- Tiered architectures complement layering.
- Layering organizes services vertically; tiering organizes functionality within those.
- Functional decomposition of two- and three-tiered architectures:
  - Presentation Logic: user interaction.
  - Application Logic: business tasks.
  - Data Logic: persistent storage.

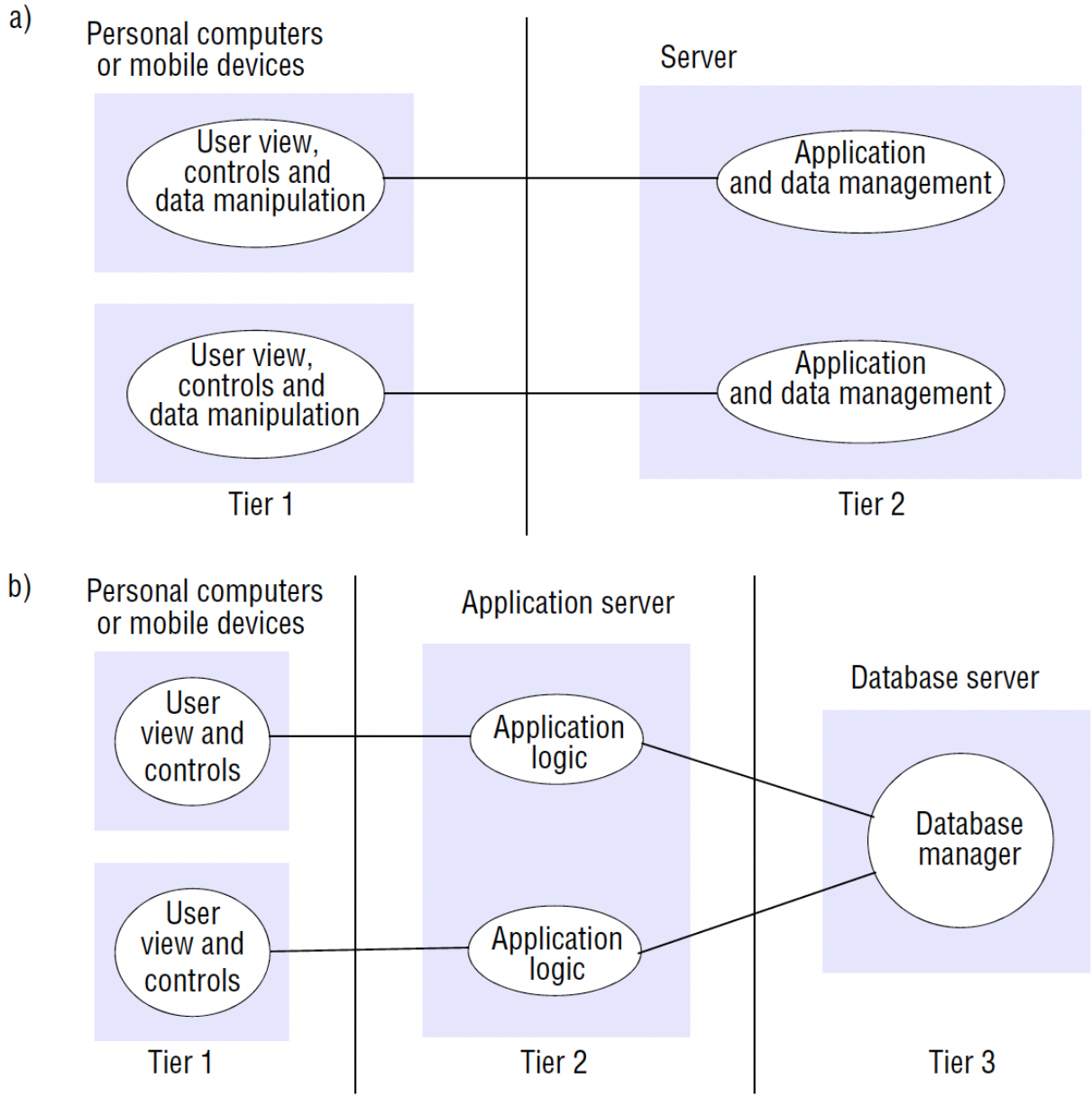
Architectural patterns

Tiered architectures

- Two-tier architecture: client-server split.
  - Application logic divided: client-server.
  - Reduces latency; complicates shared access.
- Three-tier architecture: each logical component to a physical server.
  - Enhances maintainability; increases complexity.
- N-tier solutions: generalized architecture concept.

Architectural patterns

Tiered architectures



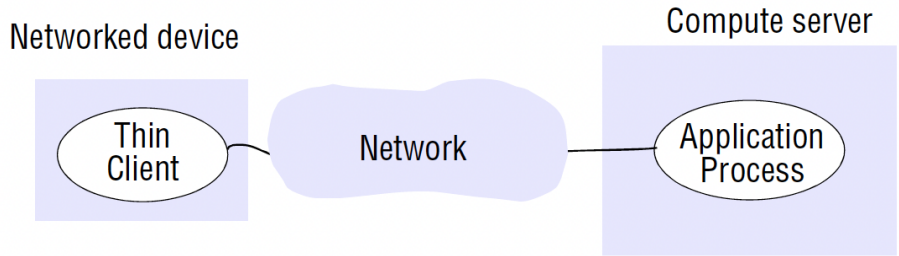
Architectural patterns

The role of AJAX

- AJAX: efficient communication between front-end and back-end.
- JavaScript: user interface, application logic handled within browser.
- Direct data requests, selective updates on current page.
- Managing internet latency and for responsive applications.

Architectural patterns

Thin client



- Complexity shifts to services.
- Thin client: local interface while accessing remote applications.
- Enhances simple devices with services.
  - Advantage: enhanced local capabilities.
  - Disadvantage: latency in tasks.
- VNC: simple protocol interaction.

Architectural patterns

Other commonly occurring patterns

- Proxy pattern: location transparency in remote calls.
  - Same interface: same interface as remote object, no awareness of distribution.
- Brokerage pattern: interoperability, service matching.
  - Some service programmer tasks: discovery, coordination, balancing, abstraction.

Architectural patterns

Associated middleware solutions

- Middleware provides programming abstraction.
- Promotes interoperability and portability.

Categories of 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

Architectural patterns

Limitations of middleware

- Middleware does not address all dependability issues.
  - Large Email transfer on unreliable networks.
  - TCP error detection insufficient.
- *End-to-end argument*:
  Some communication-related functions can be completely and reliably implemented only with the knowledge and help of the application standing at the end points of the communication system. Therefore, providing that function as a feature of the communication system itself is not always sensible.

Fundamental models

- Different system models share certain fundamental properties:
  - Processes communicate via messages.
  - Focus on performance, reliability, security.
- The purpose of building fundamental models is making
  - Explicit relevant assumptions.
  - Generalizations of possibilities.

Fundamental models

What to be captured in fundamental models

- Highlight essential system ingredients.
- Some key aspects of distributed systems:
  - Interaction: Message delays, communication.
  - Failure: Hardware, software, network faults.
  - Security: Modularity, openness, threats.

Fundamental models

Interaction model

- Single process, clear states.
- Distributed systems, unpredictable timing.
- Complex interactions in systems:
  - Servers cooperate for services.
  - Peers achieve common goals.

Fundamental models

Interaction model

- Simple programs, sequential algorithms.
  - Single process execution.
  - Behavior depends on algorithms.
- Distributed systems, multiple processes.
  - Defines steps for processes.
  - Includes message transmission.

Fundamental models

Interaction model

- Describing all states is challenging.
- Private states per process.
- Key factors affecting interactions:
  - Limit due to communication performance.
  - No global time.

Fundamental models

Interaction model - Performance of communication channels

- Characteristics of computer network communication: latency, bandwidth, jitter.
  - Latency: Delay in message.
    - Transmission latency.
    - Network access delay.
    - OS communication.
  - Bandwidth: Information volume limits.
    - Shared bandwidth affects performance.
  - Jitter: Variability of delivery time.
    - Affects multimedia data consistency.

Fundamental models

Interaction model - Computer Clocks and Timing Events

- Each computer has internal clock.
- Synchronized time eventually lost.
- Clocks drift.
  - Drift rates differ among clocks.
- Different timestamps assigned by processes.

Fundamental models

Interaction model - Two variants of the interaction model

- Establishing time limits is challenging.
- Two approaches:
  - Synchronous distributed systems
    - Fixed time intervals assumption.
  - Asynchronous distributed systems
    - No timing assumptions.

Fundamental models

Interaction model - Variant 1: Synchronous

- Defined bounds for execution and delays.
- Accuracy of bounds is challenging.
- Lack of guaranteed values affects reliability.
  - Still, modeling provides insights into behavior.
  - Timeouts detect process failures.
- Predictable resources and known drift required.

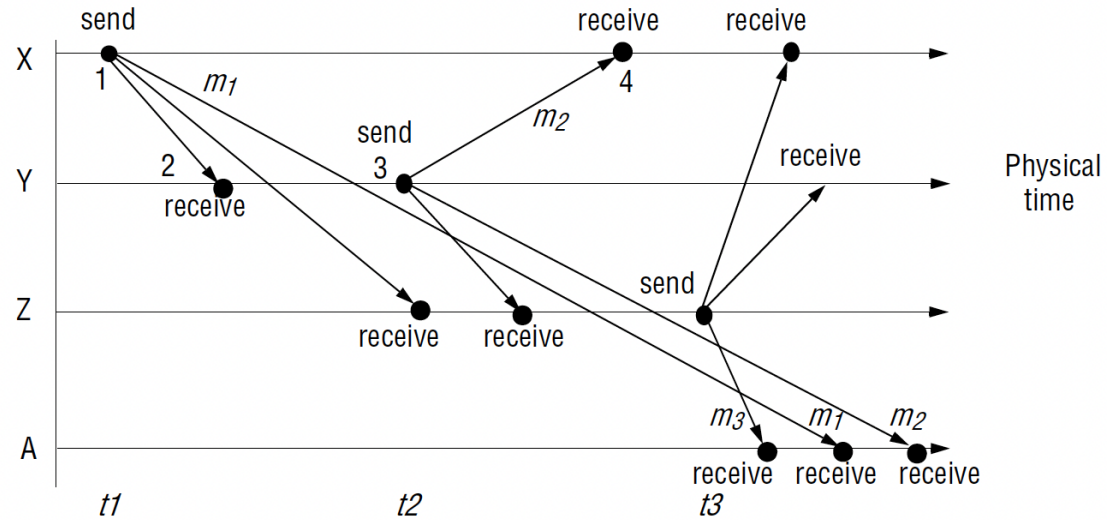
Fundamental models

Interaction model - Variant 2: Asynchronous

- No limits on execution speeds.
- Models operations with arbitrary durations.
- Real-world systems often asynchronous.
- Some design problems require time aspects.
- Synchronous model needed for deadlines.

Fundamental models

Interaction model - Event ordering



		Inbox:
Item	*From*	*Subject*
23	Z	Re: Meeting
24	X	Meeting
25	Y	Re: Meeting

- Users X, Y, Z, A communicate
  - X's message subject: Meeting.
  - Y, Z reply: Re:Meeting.
- X sends message; Y replies; Z replies referencing both.
- Message delivery delays cause order confusion for A.

Fundamental models

Interaction model - Event ordering

- System execution represented by events.
- Logical time models event order.
  - Message sent before received.
    - X sends m1 before Y receives m1.
    - Y sends m2 before X receives m2.
    - Y receives m1 before sending m2.
- Logical time assigns numbers (order) to events.

Fundamental models

Failure model

- Processes or channels may fail.
- Failure model defines failure types.
- Failures include:
  - Omission failures.
  - Arbitrary failures.
  - Timing failures.

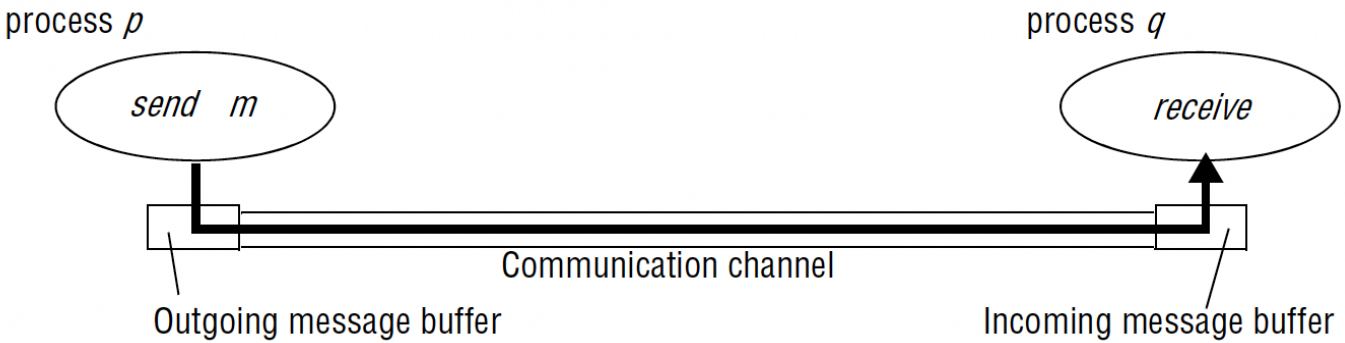
Failure model

Failure model - Omission failures

- 1. Process omission failures
  - Process crashes permanently.
  - Assumption: processes either work correctly or halt completely.
  - Detect crashes, e.g. via timeouts.
    - Timeouts in asynchronous systems problematic.
  - Detecting failures in asynchronous systems challenging.
  - Fail-stop indicates clear crashes.

Failure model

Failure model - Omission failures



- 2. Communication omission failures
  - Communication Primitives: *send*, *receive*.
    - Sending Process: Message insertion.
    - Receiving Process: Message retrieval.
  - Omission Failure: Message not transported.
    - Buffer space issues.
    - Network errors (e.g. detected by checksums).

Failure model

Failure model - Arbitrary failures

- Arbitrary or Byzantine failure: worst-case system errors (any type of error).
  - E.g. wrong data values
- Arbitrary failures in processes: Omitting or incorrectly executing processing steps
  - Hard to detect
- Arbitrary failures in communication channels: message corruption, loss, duplication.
  - Rare with safeguards

Failure model

Failure model - Timing failures

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 apply to synchronous systems.
- Asynchronous systems: cannot talk of timing failure.
- Timing crucial for multimedia channels.

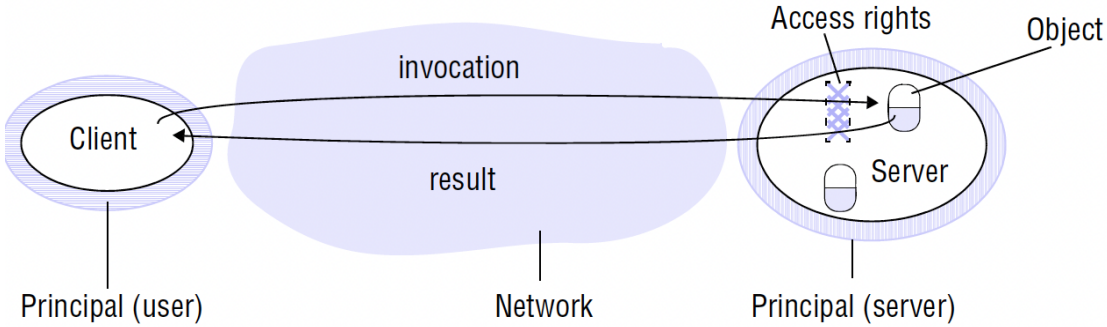
Failure model

Failure model - Masking and reliable communication

- Understanding failures help design.
- Services hide or transform failures.
- Definition of reliable communication:
  - Validity: Message delivery assurance.
  - Integrity: Message consistency and no-duplication ensured.
    - Protocols use sequence numbers.
    - Extra security for malicious threats.

Failure model

Security model - Protection of objects



- Protect processes, objects, channels from unauthorized access.
- Let users invoke operations on objects.
- Access rights determine user operations.
- Principal: An entity with authority to make requests or perform actions.
- Server responsible for verifying principal's identity, client may validate it.

Failure model

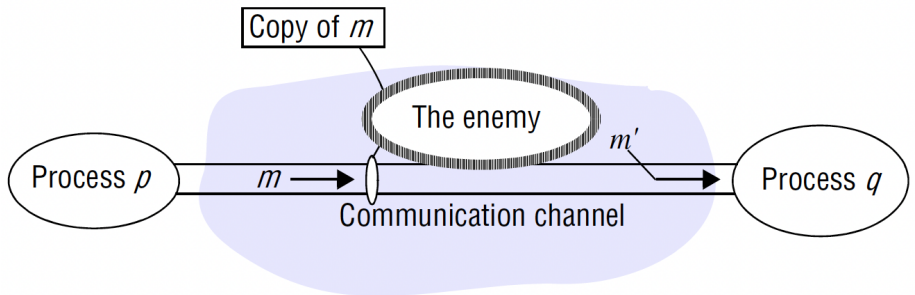
Security model - Securing processes and their interactions

- Messages vulnerable to attacks.
  - Network openness exposes interfaces.
- Sensitive tasks at risk.
- Integrity compromised by breaches.

Failure model

Security model - Enemy

- Imagine an enemy sending, reading, or copying messages between computers.



- It may intercept messages.
- Computer legitimate in the network or not.
- Threats to processes and threats to channels.

Failure model

Security model - Enemy - Threats to processes

- Processes cannot necessarily identify senders.
- Sender's address can be faked.
- Risks for servers and clients.
  - Servers misled by forged requests.
  - Clients receive false responses.

Failure model

Security model - Enemy - Threats to communication channels

- When messages are in the channel, enemy can copy, change, or add
- Harm privacy and accuracy
- Message interception
- Message replay attacks
- Secure channels with cryptography and authentication.

Failure model

Security model - Enemy - Defeating Security Threats

- Secret sharing, identity verification.
  - Encryption: Message scrambling, secret keys.
- Authentication: Identity confirmation, encrypted messages.
  - Basic authentication: encrypted verification, sender identity.
- A secure channel features:
  - Identity verification, access rights.
  - Data privacy, integrity protection.
  - Timestamping, avoid replaying.

Failure model

Security model - Enemy - Other threats

- Denial of Service (DoS):
  - Overwhelming system with numerous requests.
  - Prevent legitimate users from accessing services.
- Mobile Code:
  - Programs from other locations.
  - Potentially malicious code.

Failure model

Security model - A good security model

- Security methods can be costly.
- Good model manages costs.
- Analyze threats from various sources.
- Create threat model to assess risks.

Discussion topic

The host computers used in peer-to-peer systems are often simply desktop computers in users' offices or homes. What are the implications of this for the availability and security of any shared data objects that they hold and to what extent can any weaknesses be overcome through the use of replication?

Discussion topic

Consider a hypothetical car hire company and sketch out a three-tier solution to the provision of their underlying distributed car hire service. Use this to illustrate the benefits and drawbacks of a three-tier solution considering issues such as performance, scalability, dealing with failure and also maintaining the software over time.

Discussion topic

Consider two communication services for use in asynchronous distributed systems. In service A, messages may be lost, duplicated or delayed and checksums apply only to headers. In service B, messages may be lost. delayed or delivered too fast for the recipient to handle them, but those that are delivered arrive order and with the correct contents. Describe the classes of failure exhibited by each service. Classify their failures according to their effect on the properties of validity and integrity. Can service B be described as a reliable?