

# System Models for Distributed Systems

IN5020/9020 Autumn 2023  
Lecturer: Amir Taherkordi

August 28, 2023

UiO  University of Oslo

## Outline

- Introduction
- System Models
  - Physical Models
  - Architectural Models
  - Fundamental Models
- Summary

## System Models




- Purpose:

To illustrate/describe **common properties** and **design choices** for distributed systems in a **single descriptive model**

- Three types of models

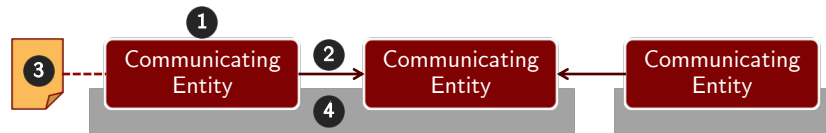
- **Physical models:** capture the *hardware* composition of a system in terms of computers and other *devices* and their interconnecting *network*
- **Architectural models:**
  - *software architecture:* the main components of the system + their roles + how they interact
  - *system architecture:* how they are deployed in an underlying network of computers
- **Fundamental models:** *formal* description of the *properties* that are *common* to architectural models.
  - Three fundamental models: **interaction** models, **failure** models and **security** models

## Physical Models

Distributed Systems	Early	Internet-scale	Contemporary
	 <b>LAN (1970s)</b>	 <b>Internet (1980s–1990s)</b>	 <b>Cloud computing (2000s)</b>
<b>Scale</b>	Small (10-100)	Large	Ultra-large
<b>Heterogeneity</b>	Limited (typically relatively homogeneous configurations)	Significant in terms of platforms, languages and middleware	Added dimensions introduced including radically different styles of architecture
<b>Openness</b>	Not a priority	Significant priority with range of standards introduced	Major challenge with existing standards: not yet able to embrace complex systems
<b>Quality of Service</b>	Not a priority	Significant priority with range of services introduced	Major challenge with existing services: not yet able to embrace complex systems

## Architectural Models

- To master the complexity of distributed systems: crucial that they are properly organized
- Concern the logical organization of distributed systems into:



1. Communicating entities

- **Objects**
- **Components**
- **Web services**

Object-based DS

Distributed Components

Web-Based Systems

2. Communication paradigms

- **Interprocess communication (IPC)**
- **Remote invocation**
- **Indirect communication**

Communication Paradigms

3. Roles and responsibilities

4. Placement Strategies

## Indirect Communication Example

- Distributed shared data space

Information sharing through a database-like distributed system called **MIDAS Data Space**

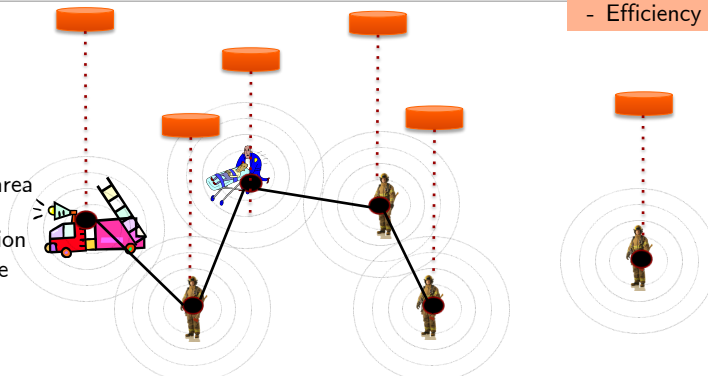
Application

Select, Insert, ...

**Implementation challenges:**

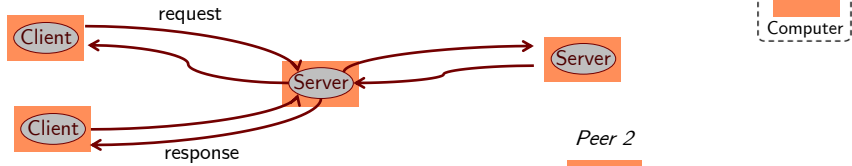
- Availability
- Fault-tolerance
- Scalability
- Consistency
- Efficiency

Emergency area without communication infrastructure

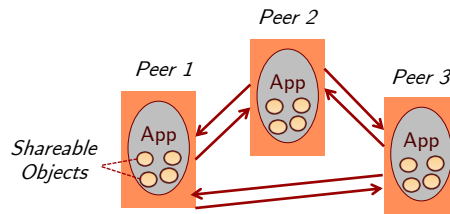


## Roles and Responsibilities

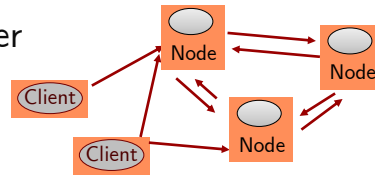
- Component view of client-server model



- Peer-to-peer



- Hybrids: P2P and Client/server



IN5020, IFI/UiO

7

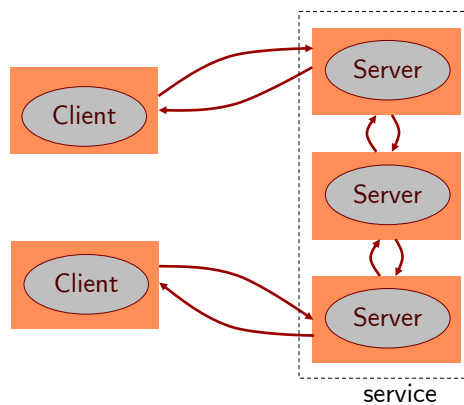
## Placement Strategies - 1

- Multiple server processes:

- service realized as a number of server-processes
- several access points

- Typical services

- LDAP or NIS

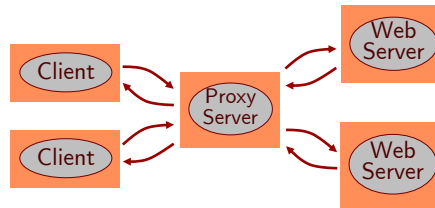


IN5020, IFI/UiO

8

## Placement Strategies - 2

- Client/server model with proxy-server
- Proxy server: cache that is shared between several clients
  - Cache: stores recently-used data objects that are closer to the client than the original objects themselves.
  - The proxy is often a load balancer
- Three types of cache
  - Content distribution network (CDN)
    - caches content for closest physical location
    - e.g. Cloudflare
  - Server cache
    - Typical caches recurring database calls.
  - Client cache
    - Browser cache
    - App downloads data that it needs to function without connection to server

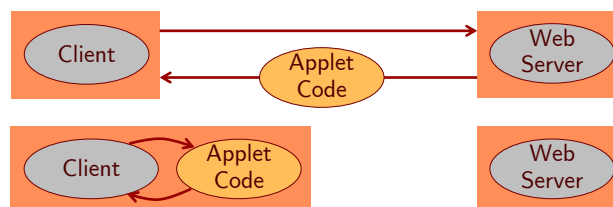
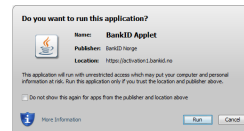


<https://www.uio.no/matnat/ifi/> - Translate this page  
**IN5020 – Distribuerte systemer - Universitetet i Oslo**  
 Tema inkluderer fundamentale modeller for distribuerte system, distribuert mellomvare med fokus på objektbasert mellomvare og komponentbaserte arkitekturer, ...  
 You've visited this page many times. Last visit: 8/28/22

This is Google's cache of <https://www.uio.no/studier/emner/matnat/ifi/IN5020/>. It is a snapshot of the page as it appeared on 25 Aug 2022 14:03:52 GMT. The [current page](#) could have changed in the meantime. [Learn more.](#)

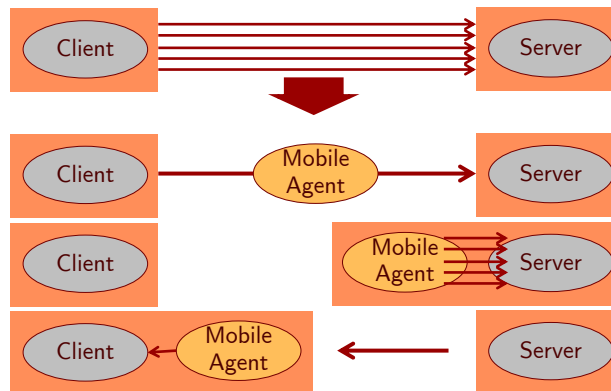
## Placement Strategies - 3

- Mobile code
  - Examples
    - Mobile apps
    - Programs/apps/websites etc. is downloaded, stored locally and able to run much of the functionality.
      - SSR – Server side rendering
  - Virtual machines
    - Downloads a java VM to run on client



## Placement Strategies - 4

- An agent takes care of handling services or data.
  - WebCrawlers
  - WebWorkers
    - Used in mobile app (both native and web)
  - Computer viruses/worms
  - Program (code + data) that migrates between computers and executes a task on behalf of someone.



IN5020, IFI/UiO

11

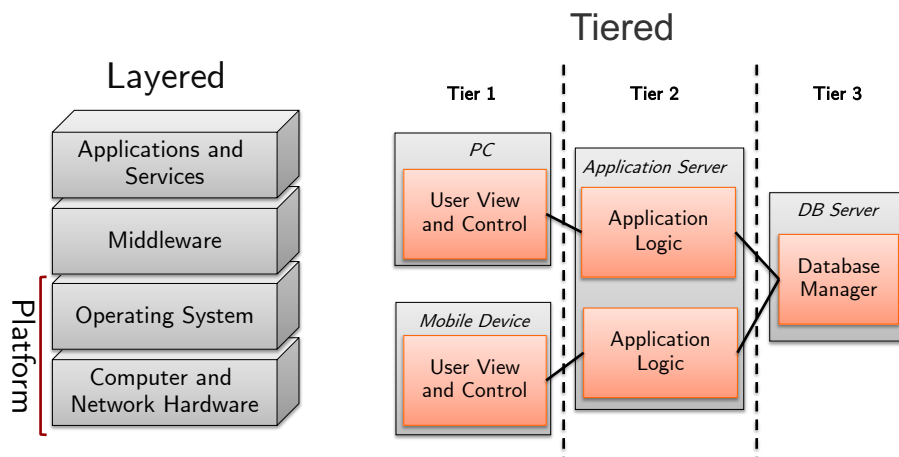
## Architectural Patterns – 1

- Build on more primitive architectural elements
- Not necessarily complete solution, but reusable by designers for some problems
- Recurring structures that have been shown to work well
  - Layering Architecture
  - Tiered Architecture
  - Thin Clients (Cloud Clients)
  - Among other patterns: Proxy, Brokerage and Reflection

IN5020, IFI/UiO

12

## Architectural Patterns – 2

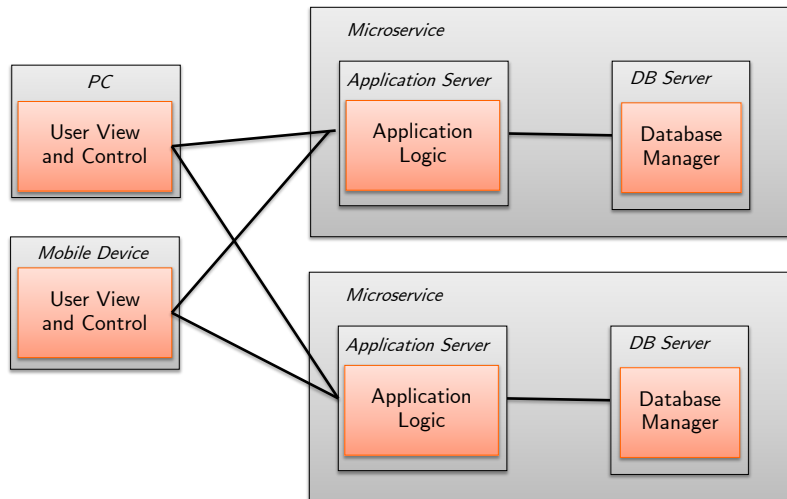


## Middleware Solutions

- Types of *Middleware Solutions*
  - Distributed objects
  - Distributed components
  - Publish-subscribe systems
  - Message queues
  - Web services
  - Peer-to-peer
- Limitations:
  - Dependability aspects
  - End-to-end argument
  - Context-aware and adaptive solutions

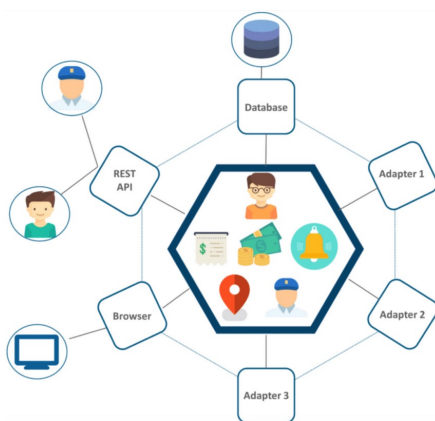
## Architectural Patterns – 3

### ■ Microservices

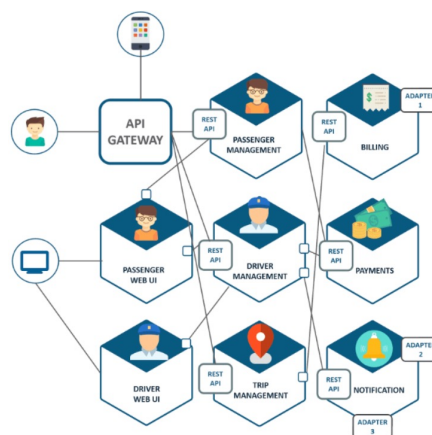


## Example from Uber

### Uber's monolithic arch.



### Uber's microservices arch.





## Architectural Patterns – 4

### ■ Thin Clients

- Move complexity from end-user devices to server side



- For example:
  - Virtual Network Computing (VNC): graphical desktop sharing system to remotely control another computer
  - Remote gaming

## Fundamental Models

- Properties shared by all architectural models
  - Processes **communicate** by sending messages across a network
  - Leads to requirements of **performance, reliability, and security**
- Fundamental models
  - focus on a **particular aspect** of a system's behavior
  - **abstract** over unnecessary/irrelevant details (like hardware details)
  - used to address questions like
    - what are the most **important entities** in the system?
    - how do they **interact**?
    - what are the **characteristics** that affect their **individual and collective behaviour**?
- The purpose of fundamental models
  - to make explicit all **relevant assumptions** about the modeled system
  - to find out what is generally **feasible and not feasible** under the **given** assumptions

## Fundamental Models

- Most common:
  - Interaction model
    - processes, messages, coordination (synchronization and ordering)
    - must reflect that messages are subject to delays, and that delay limits exact **coordination** and maintenance of global time
  - Failure model
    - **defines and classifies** failures that can occur in a DS
    - basis for analysis of **effects** of failures and for design of **fault-tolerant** systems
  - Security model
    - defines and classifies security attacks that can occur in a DS
    - basis for **analysis of threats** to a system and for design of systems that are able to **resist them**

## Interaction Model - Two Significant Factors

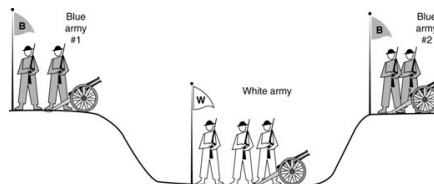
- Performance of communication:
  - **Latency**: delay between the start of the transmission and the beginning of reception
  - **Bandwidth**: Total amount of information that can be transmitted
  - **Jitter**: Variation in the time taken to deliver a series of messages: relevant for multimedia data
- Computer Clocks:
  - Each computer: its own clock
  - Two processes running on different computers: timestamps?
  - Even reading at the same time: different timestamps!
  - **Clock drift**:
    - rate for deviation from reference clock
    - How to correct time: from GPS or reference computer in the network

## Interaction Model - Two Variants

- **Synchronous** distributed systems
  - the time to **execute** each step of a process: **known** lower and upper bounds
  - each message **transmitted** over a channel is received within a **known** bounded time
  - local **clock's drift** rate from real time has a **known** bound
- **Asynchronous** distributed systems
  - the time to execute each step of a process can take arbitrarily long
  - each message transmitted over a channel can be received after an arbitrarily long time
  - local clock's drift rate from real time can be arbitrarily large

## Significance of Syn. vs Asyn. DS

- Many coordination problems have a solution in synchronous distributed systems, but not in asynchronous
  - e.g. "The two army problem" or "Agreement in Pepperland" (see [Coulouris])



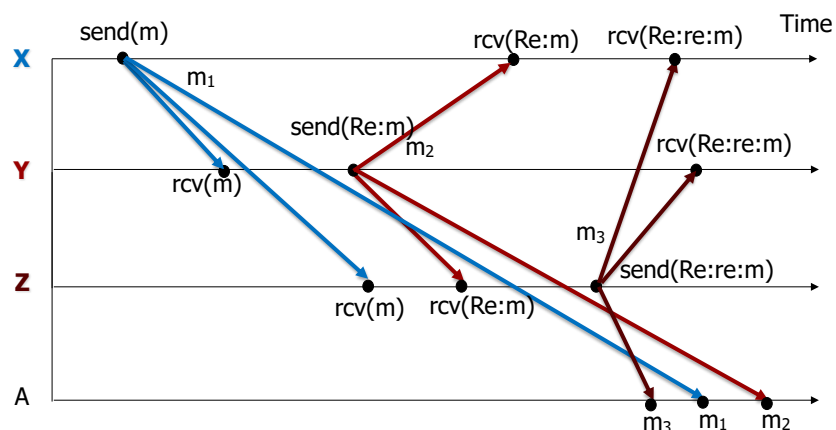
- Often we assume synchrony even when the underlying distributed system in essence is asynchronous
  - **Internet** is in essence asynchronous but we use timeouts in protocols over Internet to detect failures
  - based on **estimates of time limits**
  - but: design based on **time limits** that can not be guaranteed, will generally be **unreliable**

## Interaction Model - Ordering of Events

- Distributed coordination protocols:
  - the need for ordering of events in time (“happened before”-relationship)
  - events: sending and receiving messages
  - e.g. update of replicated data must generally be done in the same order in all replica
- Difficult to use physical clocks in computers for coordination (e.g. clock values in messages)
  - limited time resolution and ticks with different rates (clock drift)
  - basic properties of message exchange limit the accuracy of the synchronization of clocks in a DS [Lamport 78]

## Example

- E-mail exchange



## Interaction Model - Logical Clocks

- Logical clock: describing **logical ordering of events** even without accurate clocks
- Principle of “happened before”
  - $A \rightarrow B$  reads “A happened before B”
  - If A and B happen in the same process, then they occur in the order observed by that process:  $A \rightarrow B$
  - if A is sending of a message by one process and B is the receipt of the same message by another process, then  $A \rightarrow B$
- Happened-before relationship
  - is derived by **generalizing** the **two** relationships **above** such that if A, B and C are events and  $A \rightarrow B$  and  $B \rightarrow C$ , then  $A \rightarrow C$
- Logical clocks extends the idea above
  - more later in the course

Time and Coordination in DS

## Failure Model

- Is a definition of in which **way failures may occur** in distributed systems
- Provides a basis for understanding the **effects** of failures
- Failure model of a service: enables construction of a *new* service that **hides the faulty behavior** of the service it builds upon
  - example: TCP on top of IP
    - IP: unreliable datagram service
    - TCP: reliable byte-stream service hiding the unreliability of IP

## Failure Model - Specifications

- A way to describe failures
- One approach: classifying failure types (Cristian, 1991) (Hadzilacos & Toueg, 1994)
  - **Omission** failures
  - **Arbitrary** failures
  - **Timing** failures
- System model:



IN5020, IFI/UiO

27

## Failure Model - Omission Failures

- A process or channel fails to perform actions that it is supposed to do

Failure class	Affects	Description
<b>Fail-stop</b>	Process	Process halts and remains halted. Other processes may detect this state.
<b>Crash</b>	Process	Process halts and remains halted. Other processes <b>may not</b> be able to detect this state.
<b>Omission</b>	Channel	A message inserted in an outgoing message buffer never arrives in the other end's incoming buffer.
<b>Send omission</b>	Process	A process completes a <i>send</i> -operation, but the message is <b>not put into the outgoing</b> message buffer.
<b>Receive omission</b>	Process	A message is put into a process's incoming message buffer, but the <b>process does not receive it</b> .

IN5020, IFI/UiO

28

## Failure Model - Arbitrary Failures (Byzantine Failures)

- **Process or channel** may exhibit **arbitrary behavior** when failing
  - send/receive arbitrary messages at arbitrary intervals
  - a process may halt or perform “faulty” steps
  - a process may omit to respond now and then
- By adopting a byzantine failure model:
  - we can build “**ultra-reliable**” systems: handle HW failures, and provide guaranteed response times
    - control systems in airplanes
    - patient monitoring systems
    - robot control systems
    - control systems for nuclear power plants

## Failure Model - Timing Failures

- Applicable in synchronous distributed systems
  - responses not available to clients in a **specified time interval**
  - timing guarantees: guaranteed access to resources when they are needed
- Examples:
  - control and monitoring systems, multimedia systems

Failure class	Effects	Description
Clock	Process	Process's local clock <b>exceeds</b> the bounds on its <b>rate of drift</b> from real time
Performance	Process	Process <b>exceeds</b> the <b>bounds</b> on the <b>interval</b> between two processing steps
Performance	Channel	A message's transmission takes <b>longer</b> than the <b>stated bounds</b>

## Failure Model - Masking Failures

- Masking a failure by
  - **hiding** it all together or
    - e.g. message retransmission: hiding omission failures
  - **converting** it into a more acceptable type of failure
    - e.g. checksums for masking corrupted messages: in fact an arbitrary failure => an omission failure
- To mask some communication omission failures
  - Reliable 1-to-1 communication
    - Defined in terms of:
      - **Validity**: Any message in the outgoing message buffer is eventually delivered to the incoming message buffer
      - **Integrity**: The message received is identical to the one sent, and no messages are delivered twice
    - Threats:
      - Retransmission with no duplicate detection
      - Malicious injection of messages

## Summary

- Three types of system models
  - **Physical models**: capture the hardware composition of a system in terms of computers and other devices and their interconnecting network
  - **Architecture models**: defines the components of the system, the way they interact, and the way they are deployed in a network of computers
    - Architectural elements (entities, communication paradigms)
    - Architectural patterns (layering, tiered)
    - Middleware solutions
  - **Fundamental models**: formal description of the properties that are common to all architecture models
    - Interaction models
    - Failure models
    - Security models (not covered in this course)