TDDD25
Distributed Systems

Fundamentals of Distributed Systems

Christoph Kessler

IDA
Linköping University
Sweden

LINKÖPING UNIVERSITY

Agenda

1. What is a Distributed System?
2. Examples of Distributed Systems
3. Advantages and Disadvantages
4. Design Issues with Distributed Systems
5. Course Topics

What is a Distributed System?

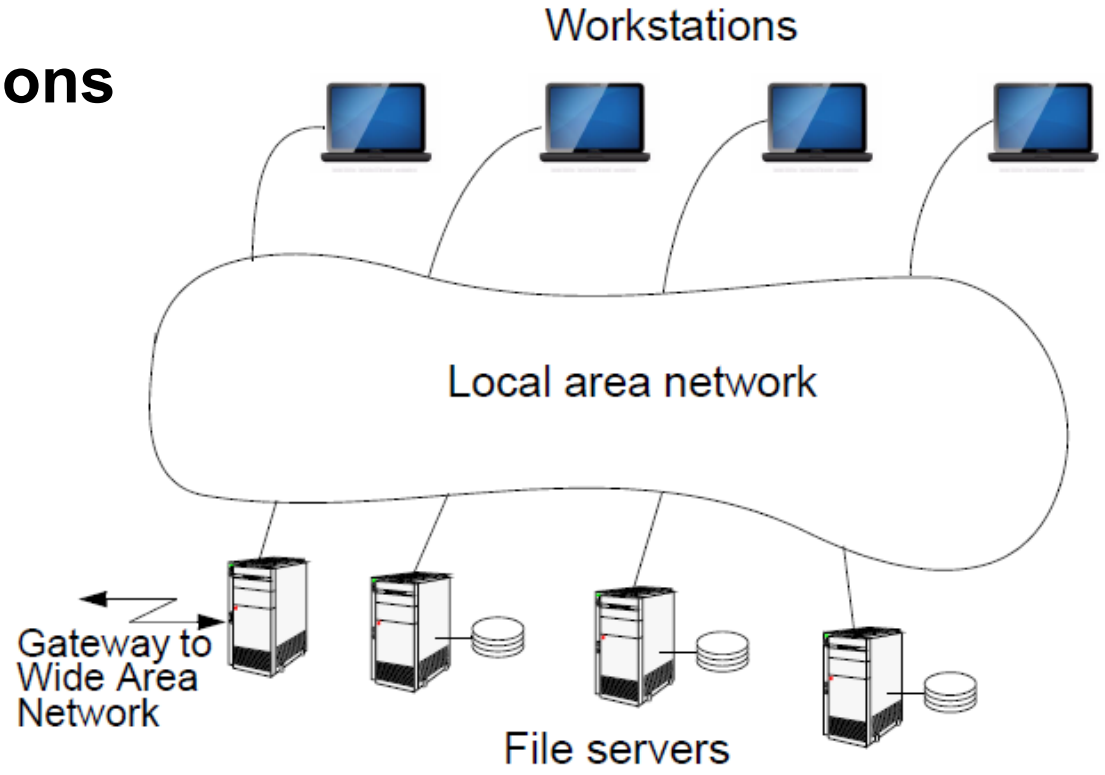
- **A *distributed system* is a collection of autonomous computers linked by a computer network that appear to the users of the system as a single computer.**

Some comments:

- System architecture: The machines are ***autonomous***; this means they are computers which, in principle, could work independently.
- The user's perception: the distributed system is perceived as a ***single*** system solving a certain problem (even though, in reality, we have several computers placed in different locations).
- By running a ***distributed system software***, the computers are enabled to:
  - coordinate their activities
  - share resources: hardware, software, data.

Examples of Distributed Systems (1)

Network of Workstations



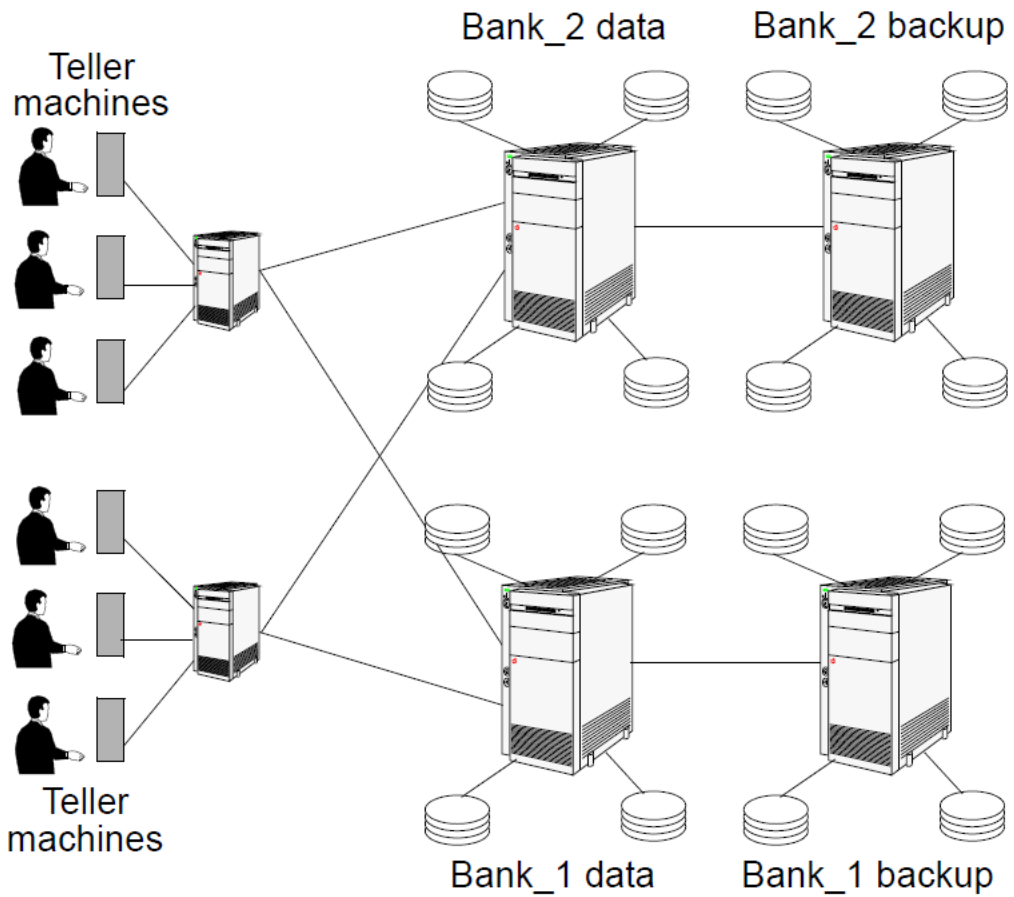
Personal workstations + servers not assigned to specific users.

- **Single file system,** with all files accessible from all machines in the same way and using the same path name.
- For a certain command, the system can look for the best place (workstation) to execute it.

Examples of Distributed Systems (2)

Automated banking system
(with teller machines, client devices)

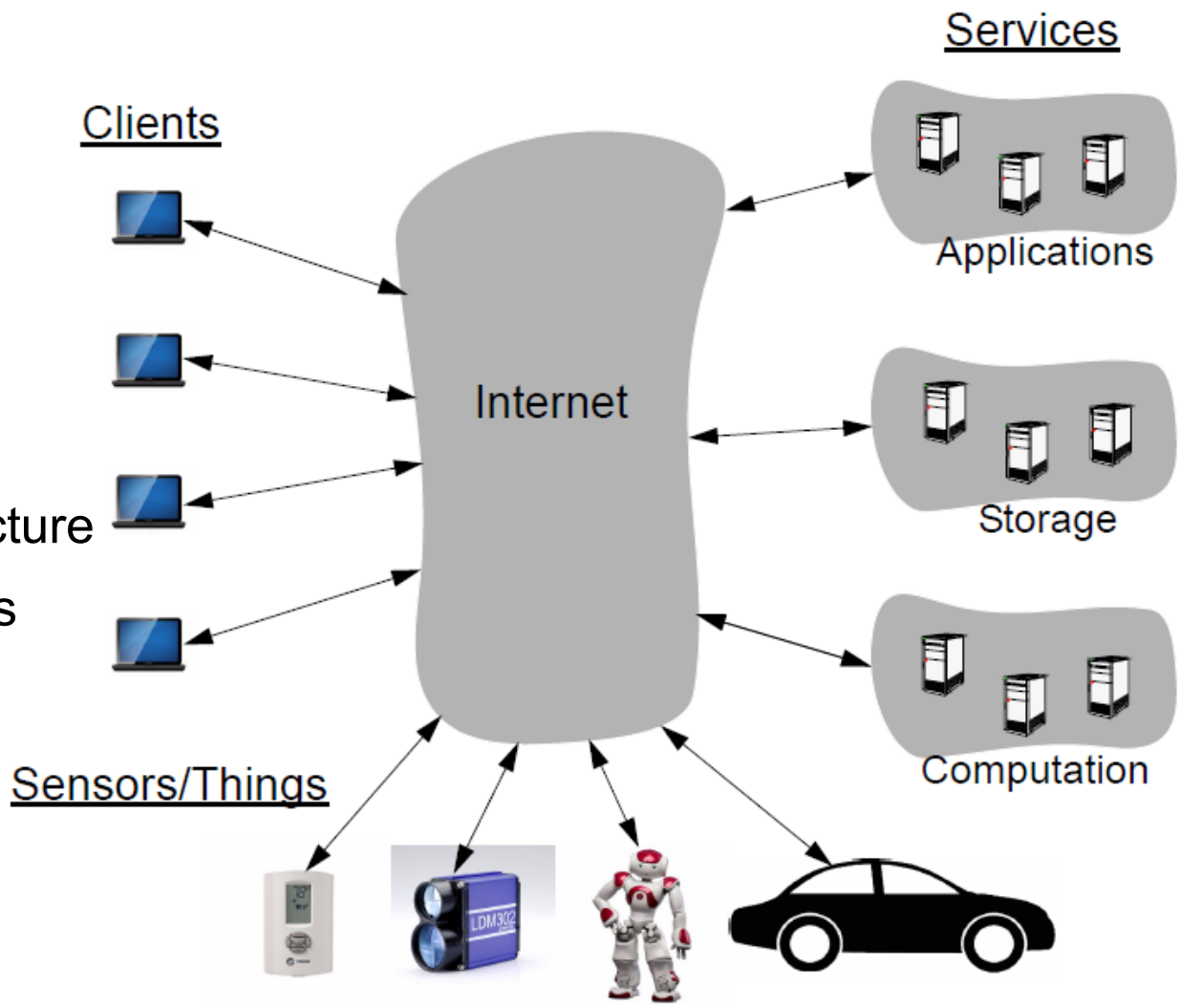
- **Primary requirements:** security and reliability
- **Consistency of replicated data**
- **Concurrent transactions** (operations which involve accounts in different banks, simultaneous access from several users, etc.)
- **Fault tolerance**



Examples of Distributed Systems (3)

The Cloud and IoT

- Computing as a utility (service)
  - Application
  - Storage
  - Computation
  - Platform / Infrastructure
- Pay on per-usage basis
- Main concerns:
  - Scaling
  - Performance
  - Security / Privacy
  - Reliability

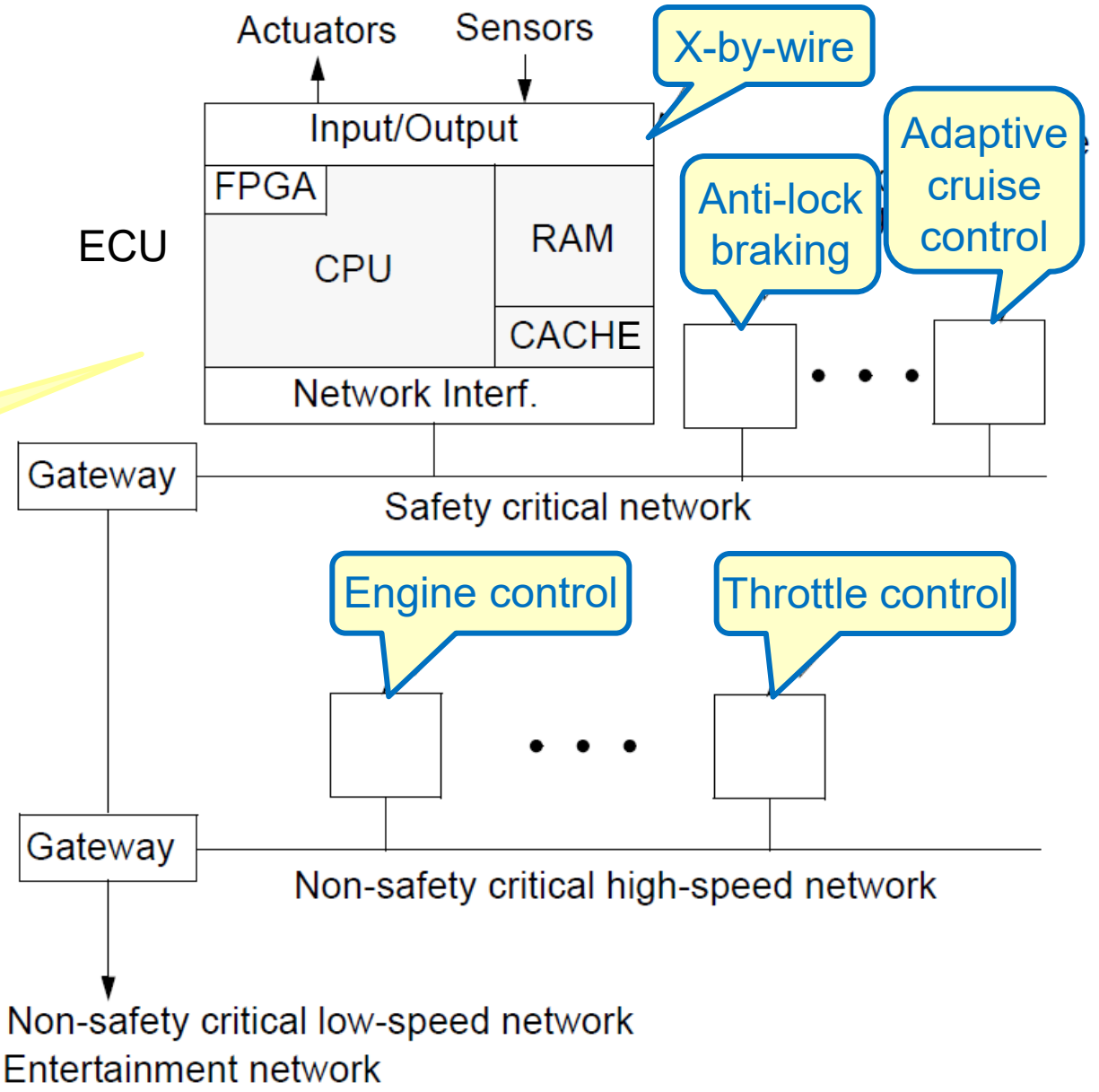


Examples of Distributed Systems (4)

Automotive system

- A distributed, embedded, real-time system

A modern high-end car has >100 ECUs



Examples of Distributed Systems (5)

Distributed Real-Time Systems

- Synchronization of physical clocks
- Scheduling with hard time constraints
- Real-time communication
- Fault tolerance

Why do we need them?

Advantages of Distributed Systems

- **Performance**
  - Very often, a collection of computers can provide higher performance (and better price/performance ratio) than a centralized computer.
    - E.g., use computers with specialized hardware for faster computing of some tasks.
    - E.g., run computations closer to where the input data is generated / stored
- **Distribution**
  - Many applications involve, by their nature, spatially separated machines (banking, commercial, automotive system).
- **Reliability** (**fault tolerance**)
  - If some machine crashes, the system can survive.
- **Incremental growth** (**scaling**)
  - As requirements on processing power grow, new machines can be added incrementally.
- **Sharing of data/resources**
  - Shared data is essential to many applications (e.g., banking, computer-supported cooperative work, reservation systems)
  - Other resources can be also shared (e.g., expensive printers).
- **Communication**
  - facilitates human-to-human communication.

Disadvantages of Distributed Systems

- **Difficulties of developing distributed software**
  - How should operating systems, programming languages and applications look like?
- **Networking problems**
  - several problems are created by the network infrastructure, which have to be dealt with:
    - Loss of messages
    - Overloading
    - ...
- **Security problems**
  - Sharing generates the problem of data security.

Design Issues with Distributed Systems

Issues that arise specifically from the distributed nature of the application

- Transparency
- Communication
- Performance and scalability
- Heterogeneity
- Openness
- Reliability and fault tolerance
- Security

Design Issues with Distributed Systems

Issues that arise specifically from the distributed nature of the application

- **Transparency**
- Communication
- Performance and scalability
- Heterogeneity
- Openness
- Reliability and fault tolerance
- Security

Transparency

- How to achieve the **single-system view**?
  - How to create the illusion for the user
    that the collection of machines is a "simple" computer?

Transparency (1)

- **Access transparency**
  - Local and remote resources are accessed using *identical operations*.
- **Location transparency**
  - Users cannot tell *where* hardware and software resources (CPUs, files, databases) are located
    - The *name* of the resource should not encode the *location* of the resource.
- **Migration (mobility) transparency**
  - Resources should be free to *move* from one location to another without having their names changed.
- **Replication transparency**
  - The system is free to make additional *copies* of files and other resources (for purpose of performance and/or reliability), without the users noticing.

Transparency (2)

- **Concurrency transparency**
  - The users will not notice the existence of *other users* in the system (even if they access the same resources).
- **Failure transparency**
  - Applications should be able to complete their task despite failures occurring in certain components of the system.
- **Performance transparency**
  - *Load variation* should not lead to performance degradation.
  - This could be achieved by *automatic reconfiguration* as response to changes of the load; it is difficult to achieve.

Design Issues with Distributed Systems

Issues that arise specifically from the distributed nature of the application

- Transparency
- **Communication**
- Performance and scalability
- Heterogeneity
- Openness
- Reliability and fault tolerance
- Security

Communication

Components of a distributed system must communicate in order to interact.

This implies support at two levels:

- **Networking infrastructure**
  - Interconnections and network software
- **Appropriate communication primitives and models**
  - **Communication primitives**
    - send
    - receive

    } Message passing
    - remote procedure call (RPC)
  - **Communication models**
    - client-server communication
      - implies a message exchange between two processes: the process that requests a service and the one that provides it;
    - group multicast
      - the target of a message is a set of processes, which are members of a given group.

Design Issues with Distributed Systems

Issues that arise specifically from the distributed nature of the application

- Transparency
- Communication
- **Performance and scalability**
- Heterogeneity
- Openness
- Reliability and fault tolerance
- Security

Performance and Scalability

Several factors influence the **performance** of a distributed system:

- The performance of involved individual computers (e.g., workstations, servers).
- The speed of the communication infrastructure.
- Extent to which reliability (fault tolerance) is provided
  - Replication and preservation of coherence imply large overheads.
- Flexibility in workload allocation
  - For example, idle processors (workstations) could be allocated automatically to a user's task.

Scalability

- The system should remain efficient even with a significant increase in the number of users and resources connected:
  - cost of adding resources should be reasonable;
  - performance loss with increased number of users and resources should be controlled;
  - *software* resources should not run out (e.g. number of bits allocated to addresses, number of entries in tables, etc.)

Design Issues with Distributed Systems

Issues that arise specifically from the distributed nature of the application

- Transparency
- Communication
- Performance and scalability
- **Heterogeneity**
- **Openness**
- Reliability and fault tolerance
- Security

Heterogeneity

Distributed applications are typically **heterogeneous**:

- **Different hardware**
  - mainframes, workstations, PCs, servers, mobile devices …
  - CPU types, accelerators, memory hierarchies …
- **Different system software**
  - UNIX/Linux, MS Windows, IBM OS/2, …, Android/iOS/…, Real-time OSs, …, file systems, executable formats, etc.;
- **Unconventional devices**
  - teller machines, telephone switches, robots, cars, manufacturing systems, etc.;
- **Diverse networks and protocols**
  - Ethernet, FDDI, ATM, TCP/IP, Novell Netware, Infiniband, etc.

The solution:

- **Middleware**, an additional software layer to mask heterogeneity

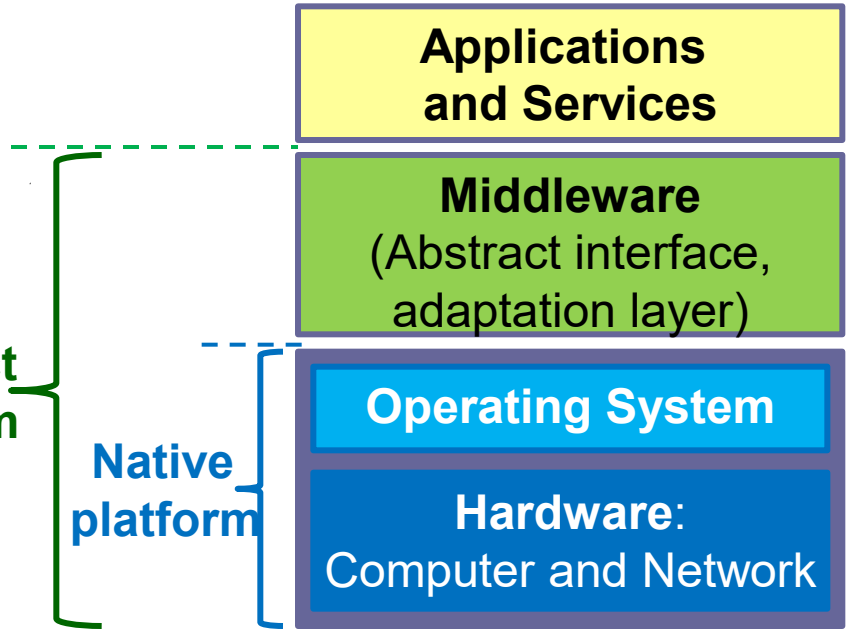
Openness

An important feature of distributed systems is **openness and flexibility**:

- Every service is equally accessible to every client (local or remote).
- It is easy to implement, install and debug new services.
- Users can write and install their own services.
- Portability of applications and services.

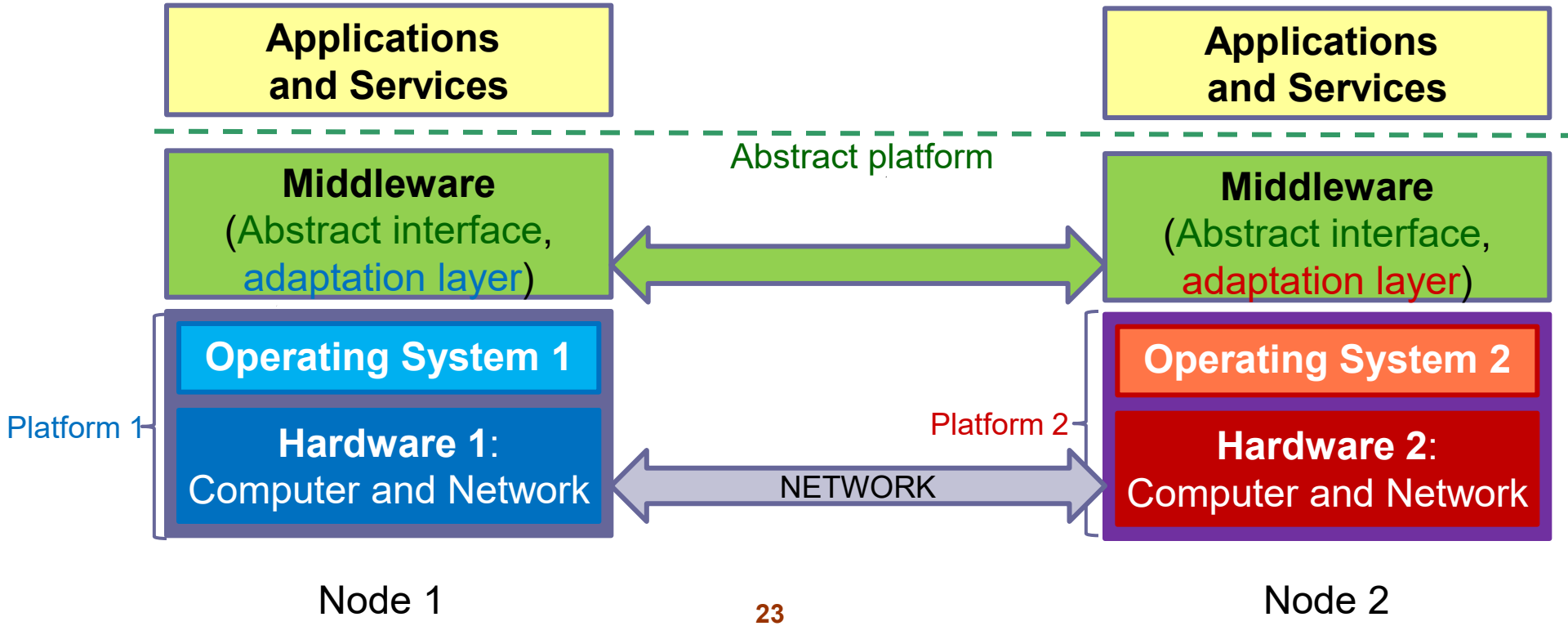
Key aspect of openness:

- ***Standard*** interfaces and protocols
  - e.g., XML, Internet protocols, HTTP, etc.
- Support of heterogeneity
  - by adequate ***middleware***, like CORBA, MPI, JVM, ...
- ***Layer-based*** software architecture:
  - Platforms could be stacked



Openness

- **Middleware**
  (abstract platform interface and its implementation(s))
  - creates a ***portable*** platform for programming and execution atop a distributed system with its heterogeneous platforms



Design Issues with Distributed Systems

Issues that arise specifically from the distributed nature of the application

- Transparency
- Communication
- Performance and scalability
- Heterogeneity
- Openness
- **Reliability and fault tolerance**
- **Security**

Reliability and Fault Tolerance

One of the main goals of building distributed systems is improved **reliability**.

Availability: If machines go down, the system should still work with the reduced amount of resources.

- There should be a very small number of critical resources (single points of failure);
  - ***critical resources***: resources which *have* to be up in order for the distributed system to work.
- Key pieces of hardware and software (critical resources) should be ***replicated***
  - if one of them fails, another one takes up - redundancy.
- Data on the system must not be lost, and copies stored *redundantly* on different servers must be kept ***consistent***.
  - The more copies are kept, the better the availability, but keeping consistency becomes more difficult.

Reliability and Fault Tolerance

- Reliable systems need to have a high degree of availability; in order to achieve this, they *need to be fault tolerant*.

- **Fault tolerance**:
  the system has to detect faults and act in a reasonable way:
  - **mask** the fault: continue to work with possibly reduced performance but without loss of data/information.
  - **fail gracefully**: react to the fault in a predictable way and possibly stop functionality for a short period, but without loss of data/information.

Security

Security of information resources implies:

1. **Confidentiality**
   - Protection against disclosure to unauthorised person
2. **Integrity**
   - Protection against alteration and corruption
3. **Availability**
   - Keep the resource accessible

Distributed systems allow communication between programs/users/resources on different computers.

Security risks associated with free access.

The appropriate use of resources by different users needs to be guaranteed!

Course Topics
at a Glance

The course focus is on understanding fundamental concepts, not on programming / learning concrete APIs

- **Basics**
  - Introduction ✓
  - Models of Distributed Systems
  - Communication in Distributed Systems
- **Middleware**
  - Distributed Heterogeneous Applications and CORBA
  - Peer-to-Peer Systems
- **Theoretical Aspects and Distributed Algorithms**
  - Time and State in Distributed Systems
  - Distributed Mutual Exclusion
  - Election and Agreement
- **Distributed Data and Fault Tolerance**
  - Replication
  - Recovery and Fault Tolerance
- **Distributed Real-Time Systems**

Course Topics (1): Basics, Middleware

- **Models of distributed systems**
- **Communication in Distributed Systems**
  - Message passing and the client/server model
  - Remote Procedure Call
  - Group Communication
  - Publish-Subscribe Systems
- **Distributed Heterogeneous Applications and CORBA**
  - Heterogeneity in distributed systems
  - Middleware
  - Objects in distributed systems
  - The CORBA approach
- **Peer-to-peer systems**
  - Basic design issues
  - The Napster file sharing system
  - BitTorrent

Course Topics (2): Theory

- **Time and State in Distributed Systems**
  - Time in distributed systems
  - Logical clocks and Vector clocks
  - Causal ordering of messages
  - Global states and state recording
- **Distributed Mutual Exclusion**
  - Mutual exclusion in distributes systems
  - Non-token-based algorithms
  - Token based algorithms
  - Distributed elections

Course Topics (3):
Distributed Data and Fault Tolerance

- **Replication**
  - Motivation for replication
  - Consistency and ordering
  - Total and causal ordering
  - Update protocols and voting
- **Recovery and Fault Tolerance**
  - Transaction recovery
  - Checkpointing and recovery
  - Fault tolerance in distributed systems
  - Hardware and software redundancy
  - Byzantine agreement

Course Topics (4):
Distributed Real-Time Systems

- Physical Clocks and Clock Synchronization
- Real-Time Scheduling
- Real-Time Communication

Acknowledgments

- Most of the slide contents is based on a previous version by Petru Eles, IDA, Linköping University.