Distributed Systems

Introduction

Prof. Dr. Oliver Hahm

Frankfurt University of Applied Sciences
Faculty 2: Computer Science and Engineering
oliver.hahm@fb2.fra-uas.de
https://teaching.dahahm.de

13.04.2023

Agenda

- Motivation and History
  - Semiconductor Technology
  - Communication Technology
  - System Technology
- Basic Concepts of Distributed Systems
  - Basic Concepts
  - Types of Transparency
  - Design Principles
  - Operating System Support (LOS - NOS - DOS)
  - Overview

Agenda

- Motivation and History
  - Semiconductor Technology
  - Communication Technology
  - System Technology
- Basic Concepts of Distributed Systems
  - Basic Concepts
  - Types of Transparency
  - Design Principles
  - Operating System Support (LOS - NOS - DOS)
  - Overview

Motivation

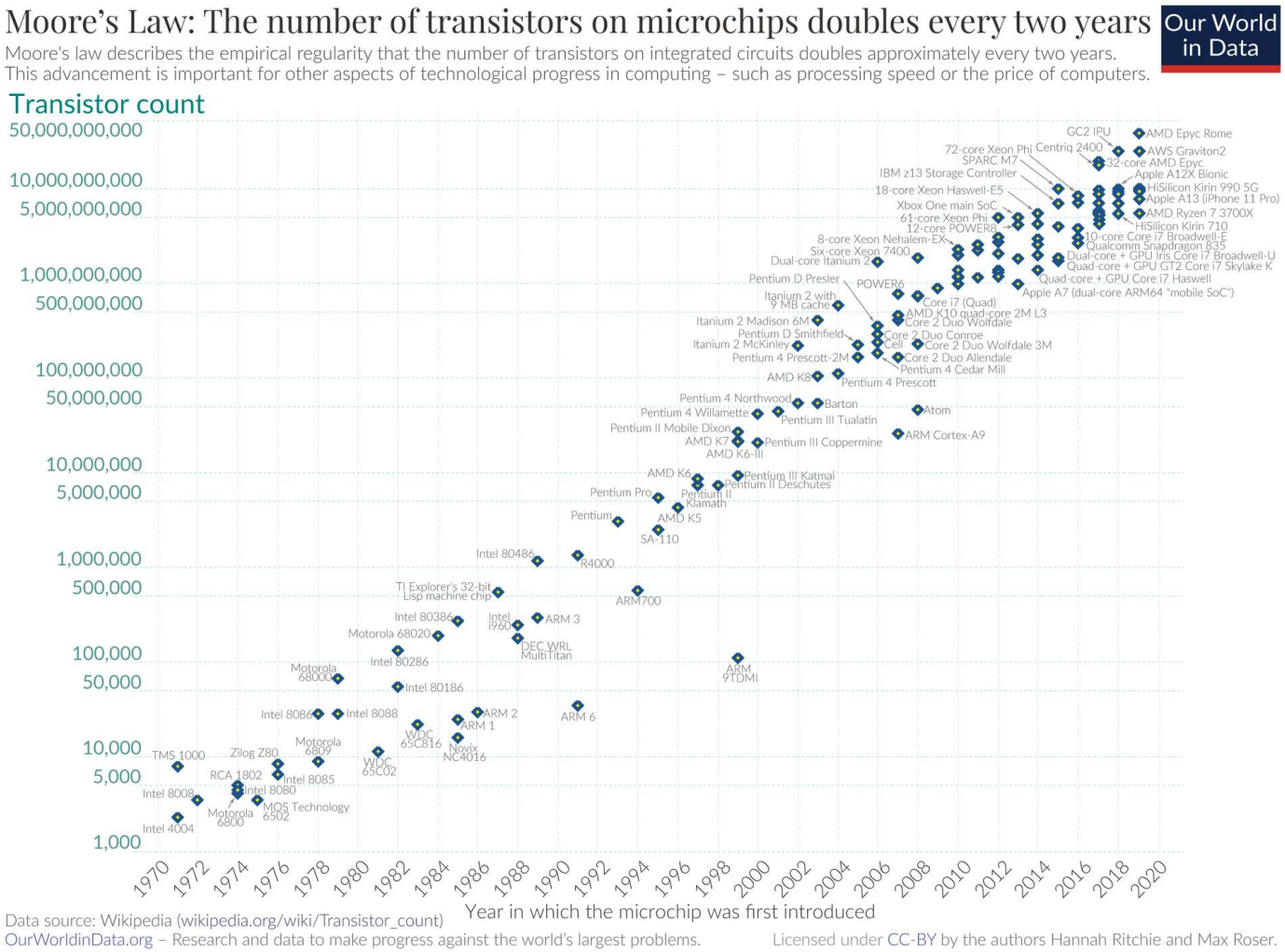
Why do we need distributed systems?

Agenda

Semiconductor Technology: Performance and Costs

- Memory chips:
  - 1973: 4 kB,
  - 1993: 16 MB,
  - 2012: 16 GB
  - 2021: 512 GB (Samsung DDR5 DRAM)
- Moores's law (1965): The number of transistors in an integrated circuit (IC) doubles about every two years
- The costs per transistor function decrease to one tenth every four years
- In 1999 Bell Labs predicted the end of silicon technology development: silicon oxide as insulation layer would have reached the strength of four atoms and cannot be further reduced (short-circuit)
- New technologies: Z-RAMs, MRAMs, FeRAMs (non-volatile), ...

Evolution of CPU Complexity



https://upload.wikimedia.org/wikipedia/commons/0/00/Transistor_Count_and_Moore%27s_Law_-_2011.svg

Agenda

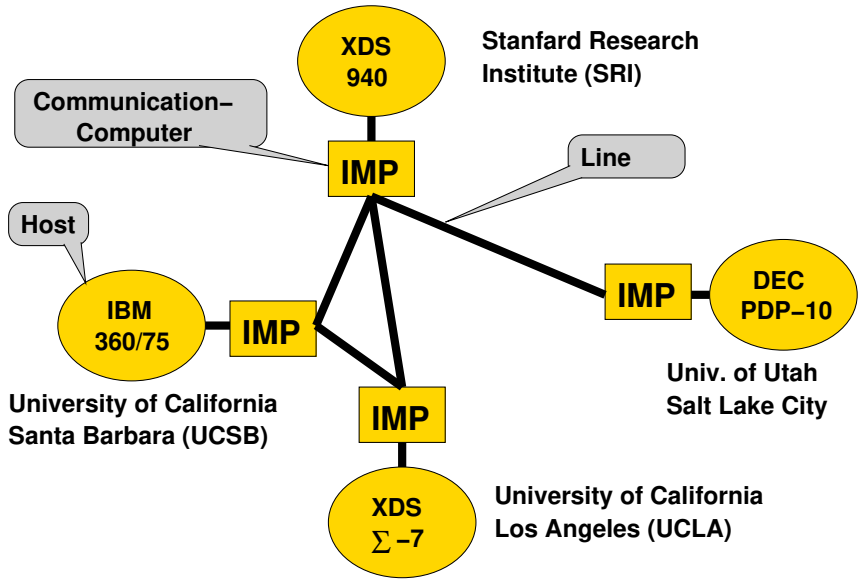
The ARPANET

1957 Foundation of the Advanced Research Projects Agency (ARPA) by the US Dept of Defense (DoD) in response to *Sputnik*

1962 The idea of the 'Internet' as 'tool to create critical mass of intellectual resources' (Licklider, Taylor)

1967 Plan for the ARPANET was published
Main architects: *Vinton Cerf, Bob Kahn*

1969 First Request for Comments (RFC) and first functioning network, rented 50 kBit/sec lines, Interface Message Processors by BBN



Graphic by courtesy of Prof. Dr. Roland Kaiser, Hochschule RheinMain

First Internet Protocols

1972 First public demo (remote login) using the Network Control Protocol (NCP)
main use: terminal sessions, file transfer, Electronic Mail

1974 Basics of TCP/IP written on paper by Cerf/Kahn (IP=Internet Protocol, TCP=Transmission Control Protocol), standardization in the following years

1982 Transition towards IP version 4 (IPv4) [1]

from 1983 Dissemination of TCP/IP due to Berkeley UNIX 4.2 BSD, source code publicly available

[2]

Author: Gorthmog, CC BY-SA 4.0

[2] deprecated, but still widely used

The World-Wide Web (WWW)

from 1970 Work about *hypertext systems* (i.e., distributed network of node documents connected by pointers with rudimentary navigation options) by Ted Nelson (Project Xanadu)

1990 Proposal of a hypertext project at *CERN* in Geneva by Tim Berners-Lee and Robert Cailliau: cradle of the world wide web

1992 Publication of an open version of a web server and browser (Unix based) by CERN, by the end of the year about 50 web servers are online

1993 Marc Andreessen, Eric Bira (NCSA, Univ. of Illinois) publish the first version of the Mosaic browser, later they found Netscape

1994 The WWW is not yet a topic for Microsoft. Bill Gates: '*... an Internet Browser is a trivial piece of software. There are at least 30 companies that have written creditable Internet browsers, so that's nothing...*'

1995 Windows 95 is released, including the *Internet Explorer* ...

1996 First *search engines* with a site-scoring algorithm, e.g., Google search

1998 Start of the dot-com boom

2004 Start of *Web 2.0* brought up blogs and RSS as well as services like Facebook or Twitter

2011 The Websocket protocol is standardized, providing communication channels "over HTTP"

Ubiquitous Networks

1982 A Coca-Cola vending machine was *connected to the Internet* at Carnegie Mellon University
1995 The first specification of IPv6 is published
1996 Hewlett-Packard and Nokia release the OmniGo 700LX and the 9000 Communicator, first smartphone predecessors
1997 Kristofer S. J. Pister, Joe Kahn, and Bernhard Boser (Berkeley) preset a research project proposal called *Smart Dust*
1999 Kevin Ashton (P&G) coined the term Internet of Things
2001 Wikipedia goes online
2004 Facebook is founded
2007 Apple releases the first iPhone
2014 The IETF working group *CORE* publishes a first specification about the Constrained Application Protocol (CoAP)

Agenda

Today's Classes of Computer Systems

- Personal Computer (PC, Desktop), Workstations
- Server, Mainframes
  - Highly reliable processing of mass data
  - High to ultra-high performance I/O-units
  - Server provide services in computer networks
- Supercomputer
  - Variety of processors/nodes
  - very high processing performance
  - Example: numerical calculations for weather forecasting
- Embedded Computer
  - Part of machines, devices, or facilities
  - The computing unit remains in the background compared to the (main) functionality of the surrounding system
  - Cyber-Physical System

Current Development

- Today's computer become more and more powerful and they have an increasingly better price-to-performance ratio, but this is achieved only by gradual improvements of known techniques
  - Processors
    - Reduced development cycles due to improved design tools
    - Focus on processors with Intel instruction sets for office usage
    - various μController types for embedded Systems (ARM, MIPS, RISC V ...)
    - Multicore processors
  - Systems
    - Increased use of systems with many nodes
    - e.g., blade server, HPC cluster
  - Networks
    - Increasing data rate
    - Manifold quality of service (QoS) requirements
    - Mobile nodes

Current Development (2)

- Virtualization
  - Virtual machines (VMs)
  - Memory virtualization (Software Defined Storage)
  - Virtual networks (Software Defined Networks, SDNs)
- Virtual infrastructures (Cloud Computing)
- Internet of Things, Industry 4.0/Industrial Internet
- Big Data

Agenda

- Motivation and History
  - Semiconductor Technology
  - Communication Technology
  - System Technology
- Basic Concepts of Distributed Systems
  - Basic Concepts
  - Types of Transparency
  - Design Principles
  - Operating System Support (LOS - NOS - DOS)
  - Overview

What could be a distributed system?

Agenda

Distributed Systems – A definition

A **Distributed System** is

- a collection of autonomous computing systems (nodes),
- coupled over a logical network, and
- appearing to its users as a single coherent system.

Distributed Systems – A definition

A **Distributed System** is

- a collection of autonomous computing systems (nodes),
- coupled over a logical network, and
- appearing to its users as a single coherent system.

Lemma:

- We need a network (i.e., connect the nodes)
- Communication is usually based on some kind of **middleware** providing a consistent access to the nodes and a common semantics for operations and results
- Independent nodes may behave *erratically* and we need some mechanism to manage those

More Definitions

Coulouris

"A system in which hardware or software components located at **networked computers** communicate and coordinate their actions only by **message passing**."

Leue

"A system that consists of a collection of two or more **independent** computers which coordinate their processing through the exchange of synchronous or asynchronous **message passing**."

Tanenbaum

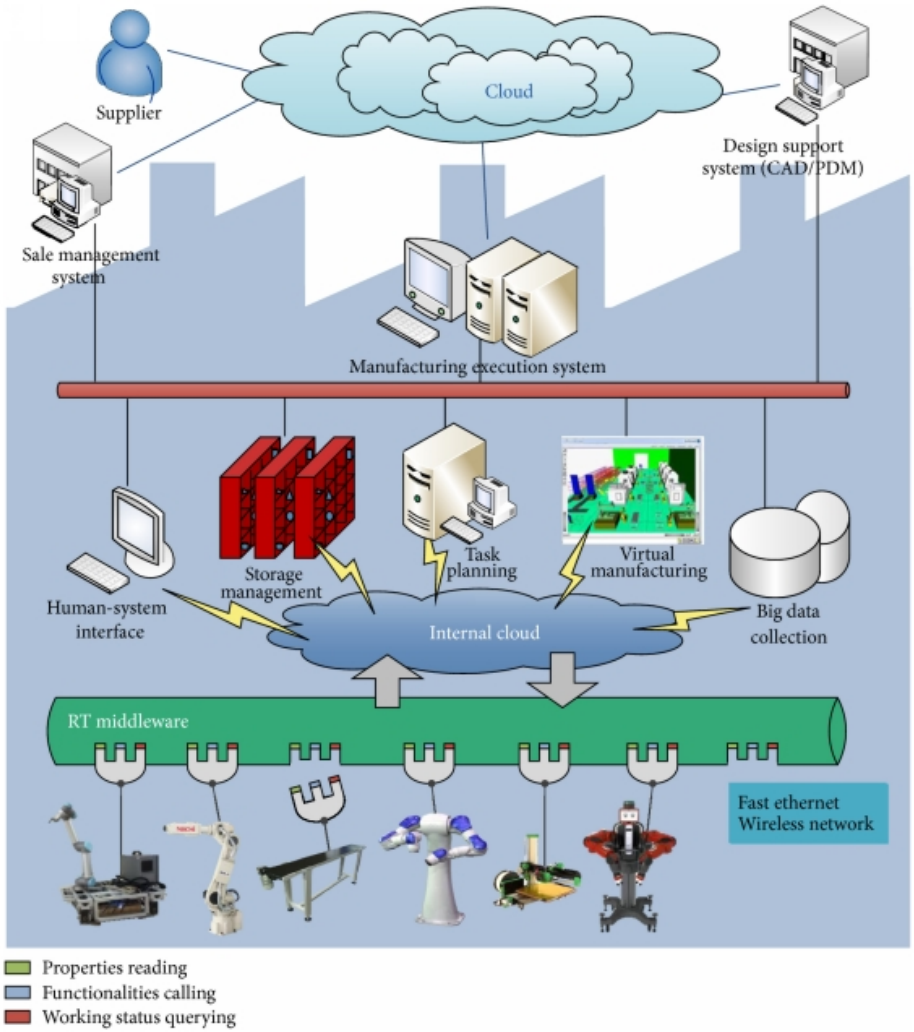
"A distributed system is a collection of **independent** computers that **appear to the users** of the system as a single computer."

Brainstorming

Can you think of an example for a distributed system?

Examples for Distributed Systems

- The **Domain Name System** (DNS)
- **Core Router infrastructure** on the Internet
- **Peer-to-peer** network applications (like BitTorrent for filesharing)
- **Automated production** lines
- Amazon Web Services (AWS) **cloud** solution
- **Internet of Things** (IoT)



https://commons.wikimedia.org/wiki/File:System-architecture-of-the-smart-factory.jpg

Basic Concepts of Distributed Systems

- Strong Coupling: Two software components are called **strongly coupled**, if they communicate with each other by sharing common resources, i.e.,
  - shared typed objects
  - shared memory segments
- Loose Coupling: Two software components are called **loosely coupled**, if they communicate with each other by message passing (increased autonomy of the components)
- Analogously, there are corresponding paradigms at the level of application programming paradigms that are based on sharing or message passing.

Distributed Program/Distributed System

- A **distributed program** consists of a set of loosely coupled software components that cooperate (by message passing) with respect to a common problem solution
- A distributed program contains
  - a distributed state
    (in the respective software components)
  - distributed control/coordination, to accomplish joint problem solving
- A **distributed system** is a computing system that executes a distributed program

Computer Networks vs. Distributed Systems

Computer Network

The autonomous computers are explicitly visible (and have to be explicitly addressed)

Computer Networks vs. Distributed Systems

Computer Network

The autonomous computers are explicitly visible (and have to be explicitly addressed)

Distributed System

Existence of multiple autonomous computers is not visible to the users ($\Rightarrow$ **transparency**)

Computer Networks vs. Distributed Systems

Computer Network

The autonomous computers are explicitly visible (and have to be explicitly addressed)

Distributed System

Existence of multiple autonomous computers is not visible to the users ($\Rightarrow$ **transparency**)

- But many issues have to be tackled for both
- Every distributed system relies on services provided by a computer network

Agenda

- Motivation and History
  - Semiconductor Technology
  - Communication Technology
  - System Technology
- Basic Concepts of Distributed Systems
  - Basic Concepts
  - Types of Transparency
  - Design Principles
  - Operating System Support (LOS - NOS - DOS)
  - Overview

Transparency (User Perspective)

- Transparency, in contrast to common usage, means the **invisibility** of a property of a multi-computer system.
- Common types of transparency:
  - **Location transparency**: enables resources to be accessed without knowledge of their physical or network location, esp. the location is not part of its name
  - **Access transparency**: Enables local and remote resources to be accessed using identical operations
  - **Migration** or **mobility transparency**: The component can be moved without changing the user interface
  - **Replication transparency**: Enables multiple instances of resources to be used to increase reliability and performance without knowledge of the replicas used by users

Types of Transparency (Developer Perspective)

- More types of transparency:
  - **Concurrency transparency**: Enables several processes to operate concurrently using shared resources without interference between them
  - **Scaling transparency**: Allows the system and applications to expand in scale without change to the system structure or the application algorithms
  - **Performance transparency**: Allows the system to be reconfigured to improve performance as loads vary.
  - **Failure transparency**: Enables the concealment of faults, allowing users to complete their tasks despite the failure of hardware or software components

How transparent are modern Distributed Systems?

- Transparency helps to simplify the management and programming of the system, since the aspect in question does not need to be considered by the user of the system.
- A distributed system should, if possible, realize all the specified transparency types in order to achieve as uniform a system view as possible
- Perfect distributed systems that abstract from all aspects do not currently exist
- The support of individual transparency types (e.g., location transparency) is advanced

Agenda

Principle: Robustness

Robustness of a distributed system requires

- availability to objects in the distributed system,
- topology-invariance of objects's in the distributed system and their access,
- fail-safe behavior of the objects.

A robust distributed system depends

- on a well-chosen and qualified architecture,
- nodes in the distributed system which perform well, i.e., posses only very few SW bugs,
- redundancy and fail-over mechanisms.

↪ Here, we define **Robustness** relative to the user experience (externally) and not to the system behavior (internally).

Robustness versus Failure

Failures in a distributed system may happen

- for the *entire* distributed systems (not usable any more)
- for a few specific nodes and objects (partial unusable components).

In the last case, some components fail, while others still stay intact. Thus, we may consider:

- *Detecting* failures (identifying component and perhaps reason),
- *Marking* failures (making it visible to others),
- *Tolerating* failures (while employing a work-around),
- *Recovery* from failures (bring up to usual operation), and perhaps
- setting up *Redundancy*.

$\hookrightarrow$ These actions are vital for a robust distributed system.

Principle: Scalability

A distributed system behaves **scalable**, if it is operational even in spite

- the number of users,
- the number of nodes, or
- the numbers of objects (resources)

increase significantly.

In practice, the **scalabilty** of a distributed system depends on

- the number of users and processes in an IT system, restricted by memory and computing power,
- the physical distance between nodes, introducing latency in information exchange, and
- the domain-model of the distributed system, confining the administrative growth.

How to scale?

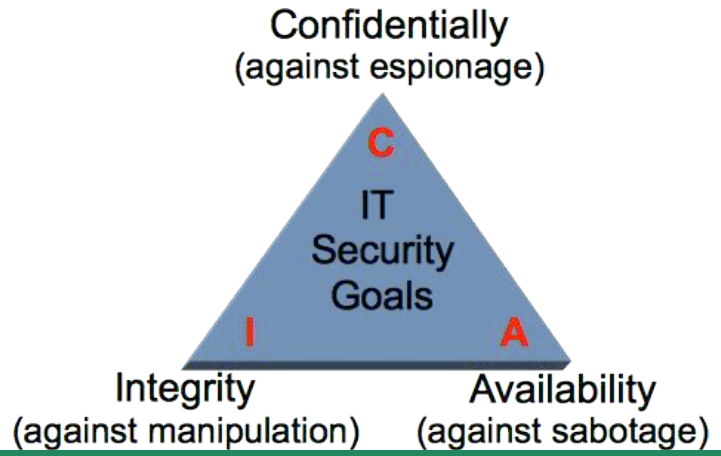
There not a single unique answer, how to scale, but a lot of recipes depending on the problem to solve:

- Use **asynchronous** communication
- Separate handler for incoming requests
- Keep information local ($\Rightarrow$ latency is minimal)
- Cache as much as possible (local replication of data)
- Organize data **hierarchical** ($\rightarrow$ DNS)
- Reduce name lookups for resources; instead use an algorithmic scheme
- Move computation to clients

$\hookrightarrow$ Data replication leads to inconsistencies among the different copies of a data set. Hence, global synchronization of objects and **time** on the nodes is required. However, strict synchronization and thus long-distance coherence is almost impossible.

Principle: Security

Security is crucial for distributed systems
IT Security targets the following goals ⇒



A distributed system meets the **IT Security** goals, if it provides

- confidential access and storage of data (by means of en/de-cryption),
- integrity for data-in-rest and data-in-flight (and data-in-computation),
- availability of resources even under critical circumstances and failure conditions.

Common failure conditions for distributed systems may be triggered from the outside:

- (Distributed) *Denial of Service* (DDoS)
- *Malware* infection.

Security versus Ease of Use

Where is the problem with **IT Security** and user acceptance?

- A common opinion is, that (IT) security is too complicated to handle by the user but **usable** security is required
- As a result, IT security concepts have been developed to be opaque (invisible) to the user and works 'automatically'.
- **Transport Layer Security** (**TLS**) (formerly known as **SSL** is a popular and wide-spread approach to establish a secure communication channel.
- Any **IT Security** requires that the user of the system is *informed* about the current security level; otherwise may act irresponsible.

$\hookrightarrow$ In order to realize security, compromises with other quality features of software development are always necessary. However, usability is a key criteria to implement security.

Principle: Openness

An open distributed system provides

- an uniform communication mechanism (interoperability),
- well defined and published APIs (Application Program Interface),
- publishes interfaces to enable remote access,
- permitting the use of the shared resources (objects),
- allows access independently from specific hardware, (computer) languages, and from heterogeneous sources (clients/users),
- and is well tested and verified regarding these requirements.

↪ Thus, **openness** is not restricted to internal use, but to public access, which of course makes a distributed system vulnerable. In turn, this requires particular means for **robustness** and **security**.

Homogeneity and Heterogeneity

Distributed systems consist of a vast variety of heterogeneous components; moreover, different understanding of the shared **objects** due to

- different hardware platforms (big vs. little endian),
- different computing Languages (Java, C, Python),
- different integration mechanisms (middleware).

[3] see: `http://pubs.opengroup.org/onlinepubs/9699919799/`

Homogeneity and Heterogeneity

Distributed systems consist of a vast variety of heterogeneous components; moreover, different understanding of the shared **objects** due to

- different hardware platforms (big vs. little endian),
- different computing Languages (Java, C, Python),
- different integration mechanisms (middleware).

Some thoughts:

- In order to provide **openness** a qualified abstraction layer is required and proprietary solutions need to be avoided. A solid foundation to realize **openness** is the `POSIX`[3] standard, to be obeyed.
- On the other hand, homogeneity often yields a restricted view to the problem and is subject of inefficient legacy solutions which tend to simultaneously crash in case of a problem.

[3] see: `http://pubs.opengroup.org/onlinepubs/9699919799/`

Agenda

LOS (Local Operating System)

- Common OS for a single node (without support for distributivity)
- Examples:
  - IBM MVS,
  - UNIX System III,
  - DOS, Windows 3.1,
  - ...

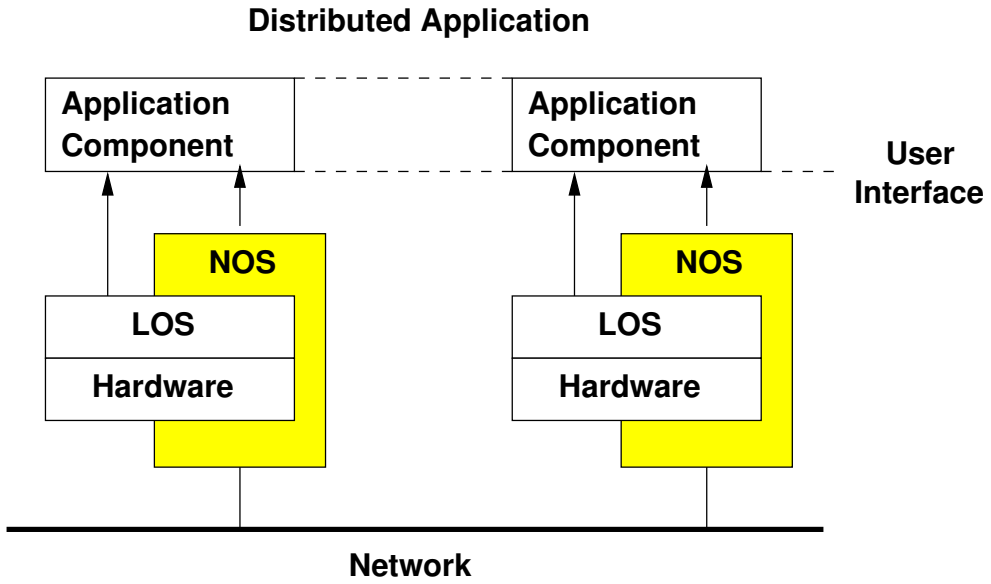
NOS (Network Operating System)

- OS extension of various LOS' for a multi-computer system to provide certain functions wrt.
  - File system,
  - Protection (user management),
  - remote program execution

  on a system level, more or less transparent
  Examples:
  - Novell NetWare,
  - MS Windows for Workgroups and basically all versions of Windows since Windows 98,
  - UNIX Yellow Pages (NIS) und Network File System (NFS)
  - Linux

NOS (2)

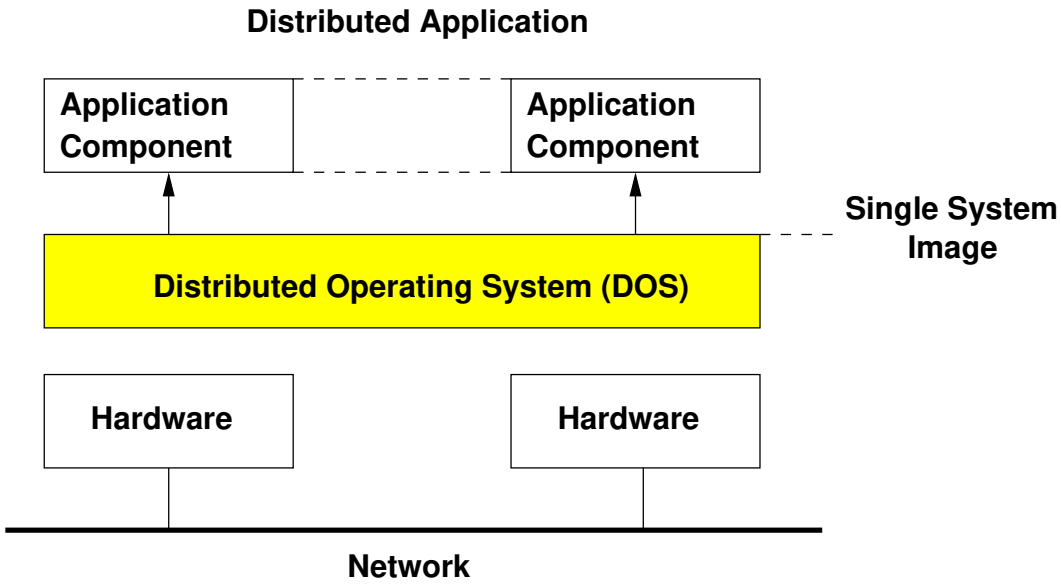
- Basic structure of a NOS:



- The underlying LOS may be the same or different
  Examples:
  - Netware Client for DOS, NT, ...
  - NFS Client for UNIX, NT, ...

DOS (Distributed Operating System)

- A distributed operating system is a basic OS which
  - provides a unified system view of a multi-computer system to its users
  - is based on algorithms that run under distributed control and exchange of messages in order to implement transparency



Agenda

Topics of this Lecture

Some topics that will be covered in this lecture:

- Network and Concurrent Programming
- Communication Patterns
- Remote Invocation
- Directory Services
- Security
- Global State and Time
- Fault Tolerance
- Distributed Filesystems
- Middleware
- Distributed Debugging
- Service Discovery
- Web Services and REST
- Coordination and Transactions
- Internet of Things
- Information Centric Networking

Important takeaway messages of this chapter

- Physical limits in semiconductor technologies require new approaches to boost performance
- The ubiquity of the Internet makes distributed systems increasingly important
- The underlying distributed nature of the components remains invisible to the user and programmer of a distributed system ($\rightarrow$ transparency)