Key to notes:
*:notes in slides
d: definition
w: Wikipedia page

Distributed Systems

Chapter 1

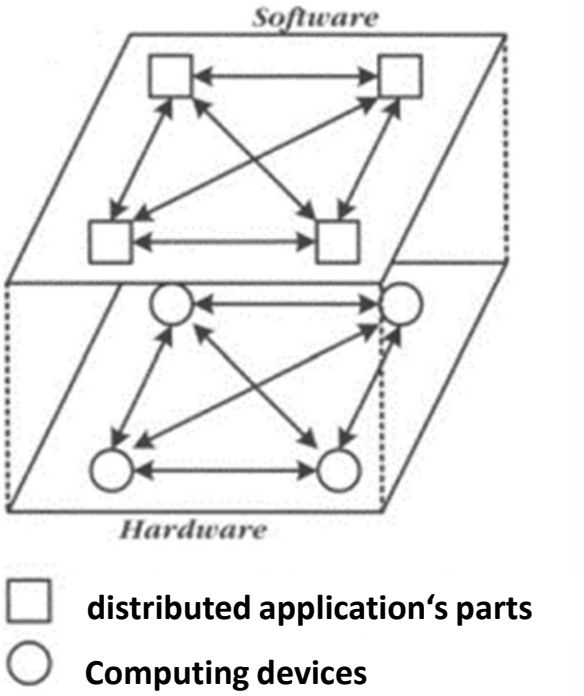
1.1: Introduction and concepts: archiecture of distributed systems

- What does a **distributed system** consist of?
  - a distributed system is a system that consists of multiple independent computing devices* that are linked together through a telecommunication network [w].

  - the computing devices in a distributed system work together to provide a common service called a **distributed application.**

*: *computing devices* or *computing systems* or just *computers* that are involved in a distributed system are also reffered to as : **active components**

Picture 1.1: structure of a distributed system



Definition of a distributed system

So what is a **distributed system** [w]?

[d]A distributed system is the combination of the physical distributed system (networked computers) and the distributed application that this system operates.

Advantages and disadvantages of distributed systems:

Advantages:

- several users can share **the access** to the software and hardware at the same time.
- Increased **reliability***
- **Faster access** to data can be achieved by distributing** it.
- Flexibility*** of the distributed application

*E.g.: implementation of a 2 or more computing systems that do identical tasks (redundancy)
**E.g.: distributed data around the globe instead of a central database in global operations companies
***E.g.: new or modified applications can be implemented on existing distributed systems.

Disadvantages:

- More components results in more failure points that can affect the functionality of the distributed system and thus **reduces fail-safety**.
- **new security risks ,**The transfer of data between individual computing devices over the network can risk the sniffing of the data by unauthorized third parties
- **Consistency problems**, introduced by having multiple saves and copies of the data in redundant data storages.
- **Fluctuating performance** driven by fluctuating workloads.
- difficult **Maintenance and upkeeping.**

1.2: Emergence of distributed systems

- The development of distributed systems mostly depends on the developments in **operating systems** [w]**, computing systems and networks**, which together forms the **fundamental technologies** distributed systems are based on.

Uniprocessor systems

- The first computing [d] systems only had a single CPU (**uniprocessor systems**[w*]) that were operated by simple operating systems. These operating systems allowed [d] the sequential processing of different tasks and applications (**single tasking)**
- Thereby , for example a shorter task must wait for the processing of a much longer task.
- Each computing system was working individually and computers weren't networked

*: modernly referred to as: single-core based systems

A step towards multi-tasking systems [W]

- Soon enough, It was acknowledged that sequential processing of tasks isn't efficient and leads to prolonged waiting delays for individual users.
- As a result , A so called multitasking operating systems were developed that can do a more of quasi-parallel* task processing.
- These operating systems are based on methods for managing the computer resources and distributing them on multiple tasks.

*Quasi-parallel: nearly parallel , almost parallel or have some parallel-like characteristics

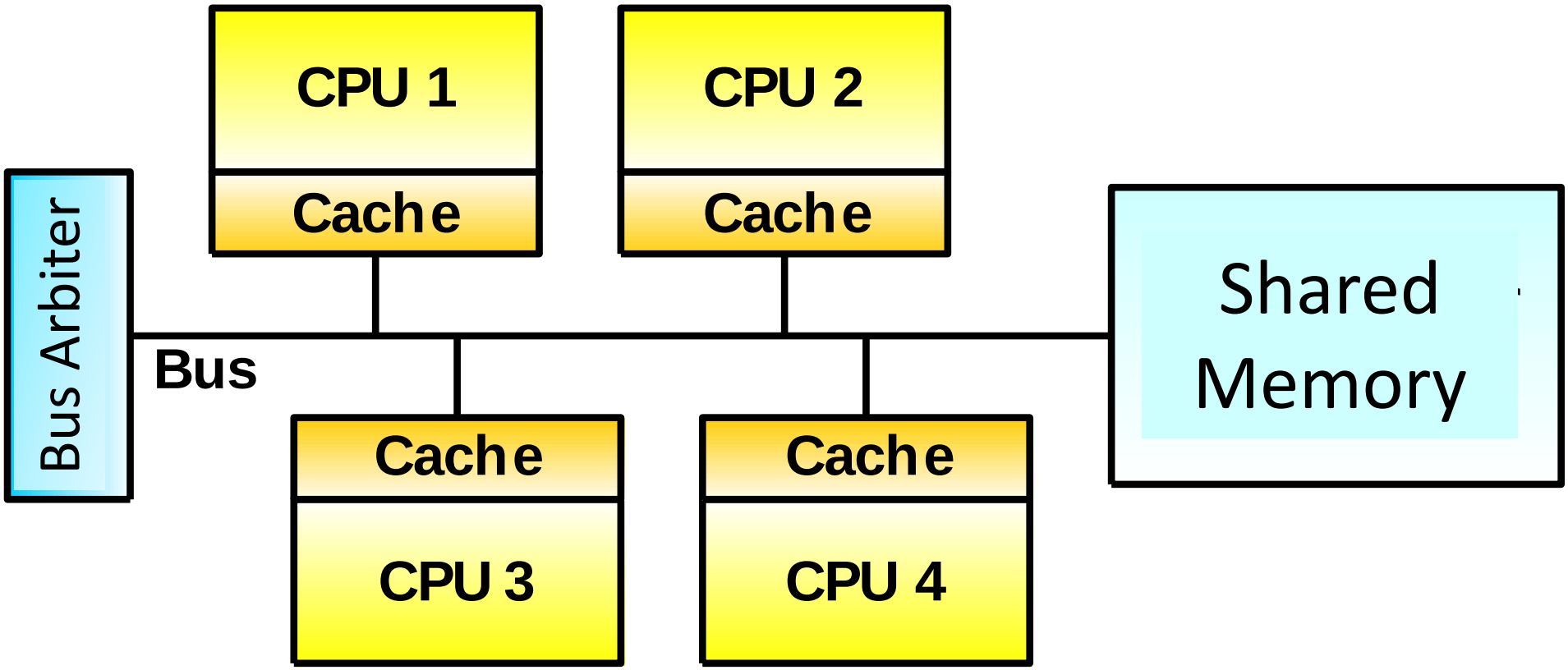
Multi-tasking operating systems

- For example, the computing time of the CPU can be distributed over several jobs using a time slicing method*.
- A task is assigned to the CPU for a defined period of time (time slice) and get executed on it.
- At the end of the time slice the operating system pauses the task and may work on another task.

* Also with a process called preemption : Wikipedia

Multiprocessor Systems [w]

- To gain further efficiency, computer systems that are [d] based on several CPUs were developed.
- Thereby these [d] computer systems can **truly** process multiple tasks in parallel , by distributing them on multiple CPUs, these systems can be called **multiprocessor Systems**.
- The CPUs in these systems shares common memory [w], over which Data can be exchanged between CPUs.
- Such systems were easy to implement but lacked stability.



Picture 1.2: schematic structure of a multiprocessor system

Important components

- CPU [w]: A distinction is made between symmetrical and asymmetrical multiprocessor systems. In a symmetrical multiprocessor system, all CPUs have equal rights, i.e. code of the operating system can be executed on any CPU. An asymmetric multiprocessor system has a privileged CPU, the so-called master, which is exclusively allowed to run the operating system code. All other CPUs are used to process user tasks. A power of two is often used for the number of CPUs used (2, 4 , 8, 16, 32 , etc.)
- Cache [w]: The cache is an intermediate memory to compensate for the speed difference between the CPU and the memory.
- Bus [w]: The connection lines between the CPUs, the bus arbiter (or arbitrator) and the shared memory is called the bus. The bus is shared by all CPUs and can therefore become a bottleneck.
- Bus Arbiter [w]: The arbiter* [1]controls access to the shared bus. It implements logic that [2]prevents multiple CPUs from accessing the shared bus at the same time.
- Memory [w]: is used by the CPUs to [1]store data and for [2]inter-process communication.

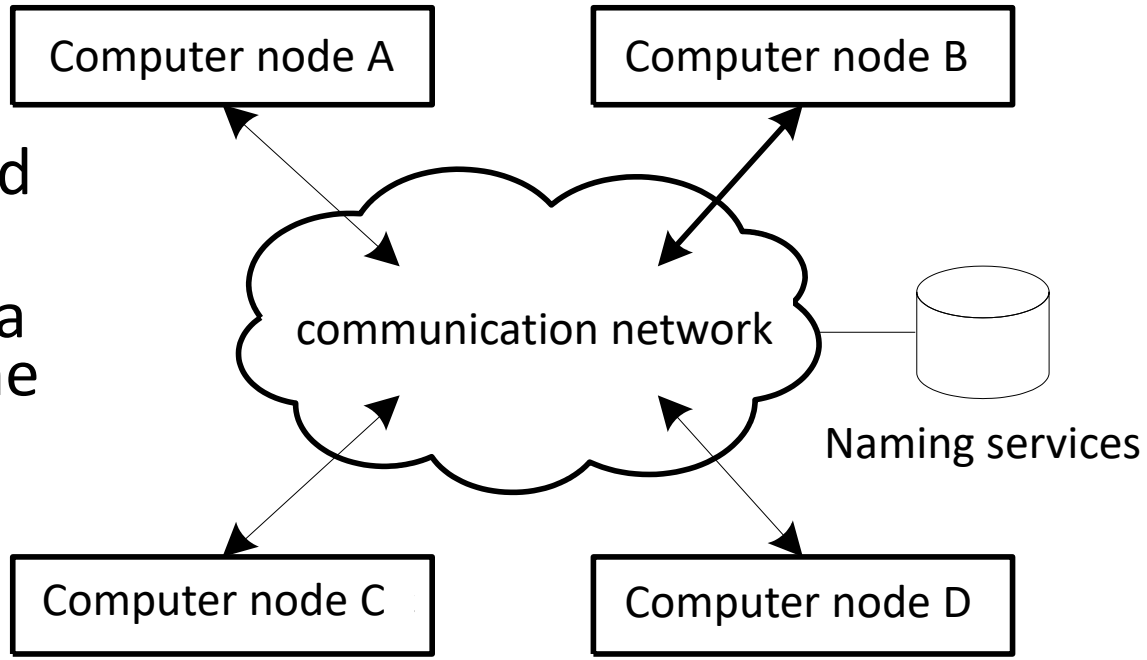
*Bus Arbiter: also referred to as Bus master , or Bus arbitrator .The Job it does is called Bus arbitration.

Drawbacks and alternatives: The Multicomputer

- Unfortunately, the production of multiprocessor systems was *relatively expensive*, which is why people quickly looked for alternative ways to *increase performance*.
- An obvious idea was to link the existing uniprocessor systems to form a combined system in order to achieve higher performance.
- d When several independent uniprocessor systems are linked together, this is referred to as a **multicomputer.**

Multicomputer systems

- Multicomputer systems can be seen as the archetype* of distributed systems.
- The essential feature of a multicomputer is that the individual computers **do not have a shared memory**.
- *Communication* can therefore only be realized via the **exchange of messages**.



Picture 1.3: structure diagram of a multicomputer

* archetype: a typical example of an architectural model Or one of the most primitive applied concept of it

Structure of a message:

- A message always consists of a
  1- sender address
  2- recipient address
  3- an envelope
  4- the actual content (payload)
- The envelope serves to protect or encrypt the content (analogue: ordinary envelope).
- The sender and recipient address (analogue: postal address) are used to convey the message from the queue to the destination.

Naming Service

- Communication via messages requires the use of a naming service. A naming service implements the mapping of the sender and receiver addresses to physical addresses of the computer systems.
- Consequently, a naming service works like a directory inquiry service, which assigns a person to their address.

Important components of a multicomputer:

1. Computer systems:
   - [d] when the system have all it's computers of the same type (in term of performance and internal architecture),the system can be called a **homogeneous multicomputer system**.
   - [d] When the system have different types of computers in it , it can be called **heterogeneous multicomputer system**.
2. Communication network(s):
   - the communication network can be a [1]standard or [2]proprietary network.
   - In order for a computer to communicate over a network , it's operating system must [d] have special networking functionalities. These operating systems were *historically* called **network operating systems*.**

* The term have changed it's meaning with time, more about it : Wikipedia

1.3 Structure of distributed systems: distributed and programming paradigms

- Each distributed system is based on a specific distribution paradigm or (distribution structure). This paradigm specifies which (logical) units have to be distributed.
- Also each distributed system have a programming paradigm (implementation technique). This paradigm defines how a developer would implement the distributed applications and what basic functionality he can fall back on.

What to distribute?

Distribution paradigm (structure):

- Data [w]
  - Documents (Files)
- Functionalities
  - Subroutines [w](Procedures)
  - Objects [w]
  - Computational resources [w]

Programming paradigm (implementation technique):

- Communication over sockets [w]
- remote procedure call (RPC) [w]
- Remote Method invocation (RMI) [w]
- Message passing [w] (message-based communication)

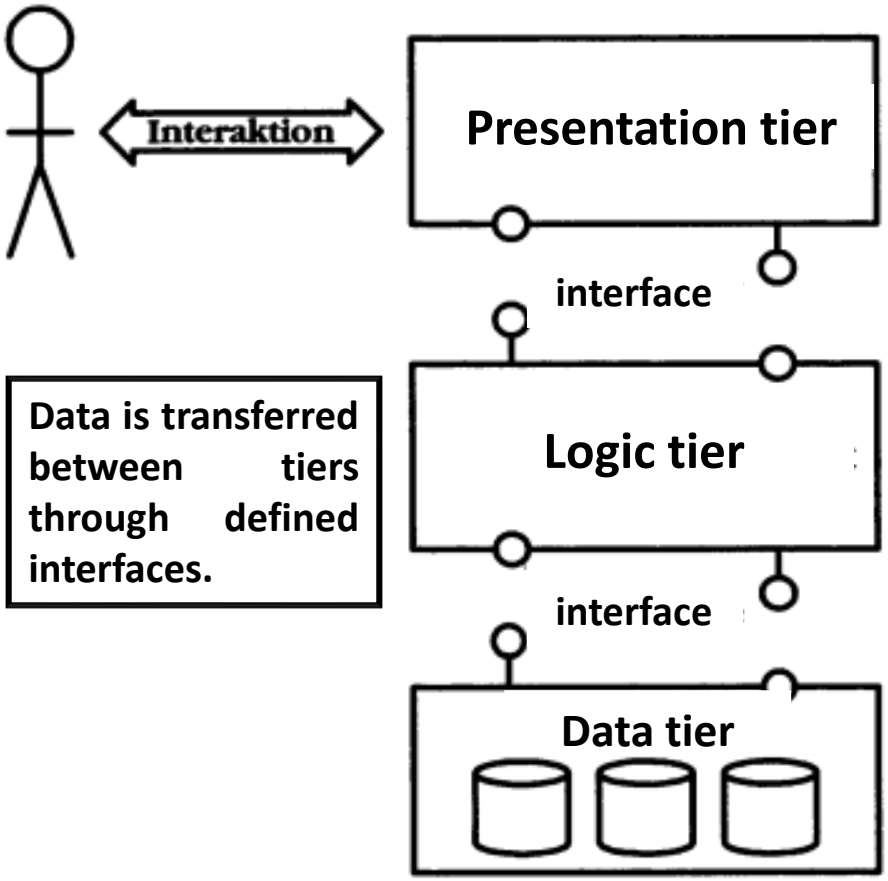
Distribution paradigm	Programming paradigm: *Sockets*	*RPC*	*RMI*	Messages
Data	x	–	–	o
Procedures	o	x	–	o
Objects	o	–	x	o
Computation	o	–	–	x

Table 1.1 shows the relationship between the distribution paradigm and the programming paradigm.
x in a field means that distribution and programming paradigm fit very well together and are often used in combination.
0 means that distribution and programming paradigm can be combined, but a certain additional effort is necessary.
- on the other hand advises against the combination of distribution and programming paradigm. In principle, however, all distribution paradigms can be implemented with any programming paradigm.

1.4: composition of distributed applications

e.g. : 3-tier-architecture

- a distributed application is subdivided into several logical tiers [w]:
  1. **Presentation tier:** implements a graphical front-end(GUI) for the user and thus allows the operation of the distributed application.
  2. **Logic tier***: implements the logic for the distributed application's functions.
  3. **Data tier**: is the persistent Storage of the application's data.



*: can also be : Business service tier or/and application tier

Picture 1.4 : 3-tier-architecture

N-tier-architecture

- For efficient implementation of distributed applications:
  - The logical tiers are distributed across several computer systems.
  - In practice the three logical tiers are distributed over three computers , which is referred to as **three-tier-architecture.**
  - Other distributions are also possible:
    - When using an n-tier architecture with more than 3 tiers , the logic tier is further *subdivided*.

N-tier-architecture

- Depending on how much **functionality** is provided by the computer system a distinction is made between thin clients and thick clients:
  - **Thin clients**: only the user interface and data is shown on thin clients , while the program logic is mostly provided by other components , in the is case thick clients (heavily dependent on the distributed system) .
  - **Thick clients**: thick clients takes over the programs' logic (in addition of the presentation of data)(have the ability to operate on it's own partially or by whole).

1.5: transparency [W] in distributed systems

- An important characteristic of distributed systems is the implementation of various **transparencies.**
- **Transparency :** hiding some aspect of the distributed system from the user.
- Different transparencies:
  - Location Transparency
  - Migration Transparency
  - Replication Transparency
  - Fault Transparency
  - Performance Transparency
  - Scaling Transparency
  - Concurrency Transparency
  - Access Transparency
  - Relocation Transparency

Transparencies in distributed systems:

Note: Resources : can refer to data , objects, or other software or hardware components.

1. Location transparency
   - The location of resources and other system components are not of user concern and thus may not be visible.
2. Migration transparency
   - Objects and resources can migrate between computer systems, this process is hidden from the user
3. Replication transparency
   - Replicated resources appear as a single
4. Scaling Transparency
   - A system should be able to grow with ease when required
   - A system should also be capable to scale down to small environments where required.

Transparencies in distributed systems:

5. Fault Transparency:
   - If software or hardware failures occur, these should be hidden from the user
6. Concurrency and Relocation transparency:
   - resources should be accessible while they are relocating or moving between computer systems
   - Resources should be accessible by multiple users and applications without interference
7. Performance Transparency:
   - the distribution of tasks on different computer systems (load balancing)
   - And the configuration of the system should not be apparent to the user in term of performance

Transparencies in distributed systems:

8. Access Transparency:

- Hiding the diversity in operating systems and computer systems

Resources
(in German)

- Günther Bengel: *Grundkurs Verteilte Systeme (3. Auflage)*; vieweg 2004; http://www.vts.fh-mannheim.de
- George Coulouris, Jean Dollimore, Tim Kindberg: *Verteilte Systeme – Konzepte und Design (3., überarbeitete Auflage)*; Pearson Studium 2002
- Schahram Dustdar, Harald Gall, Manfred Hauswirth: *Software-Architekturen für Verteilte Systeme*; Springer 2003
- Andrew S. Tanenbaum: *Computernetzwerke (3. überarbeitete Auflage)*; Pearson Studium 2002; http://www.cs.vu.nl/~ast
- Andrew Tanenbaum, Marten van Steen: *Verteilte Systeme – Grundlagen und Paradigmen*; Pearson Studium 2003
- Michael Weber: *Verteilte Systeme*; Spektrum Akademischer Verlag 1998

Resources
(in German)

- In "Middleware in Java" you will find repetition, consolidation, programming tasks to the presented topics. To consolidate and deepen the learned, it is recommended to work on the tasks.