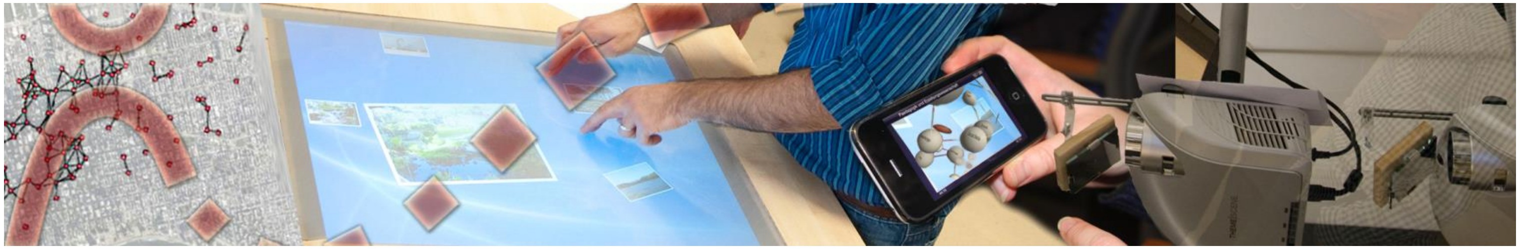
Telecooperation Lab
Prof. Dr. Max Mühlhäuser

# Telekooperation 1: Exercise WS15/16

Michael Stein, MSc.
michael.stein@tk.informatik.tu-darmstadt.de
Jens Heuschkel, MSc.
jens.heuschkel@tk.informatik.tu-darmstadt.de

***Copyrighted material – for TUD student use only***



# TK1 – EXERCISE

- **Organization**
- Solutions Theory Assignment 1.
- Introduction Theory Assignment 2.

# Consultation Hours

- Document with information regarding the consultation hours is available on Moodle
- The consultation hour is perfectly suited to clarify open questions regarding the exercises
- Contact information is available for questions via e-mail

# Plagiarism

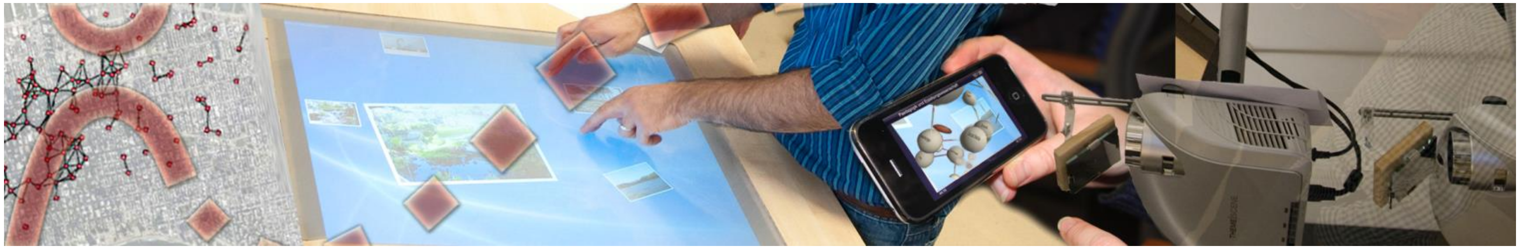
- We can‘t give you points for exact copies
- Please use your own words!

- Copies from the internet **without** marking it will give 0 points
- **Exact** copies from the slides will also give 0 points

- Citations are allowed, but you have to mark it

More information:

***https://www.informatik.tu-darmstadt.de/de/sonstiges/plagiarismus/***



# TK1 – EXERCISE

- Organization
- **Solutions Theory Assignment 1.**
- Introduction Theory Assignment 2.

# Task 1.1: Basic Problems

- Task 1.1: Describe the three basic problems of distributed systems.

# Task 1.1: Basic Problems

Basic Problem #1: Global State Not Accessible (without unacceptable slowdown)

- no synchronized global variables, no global shared memory
- message / agent travelling A → B: out-dated state of A arriving at B

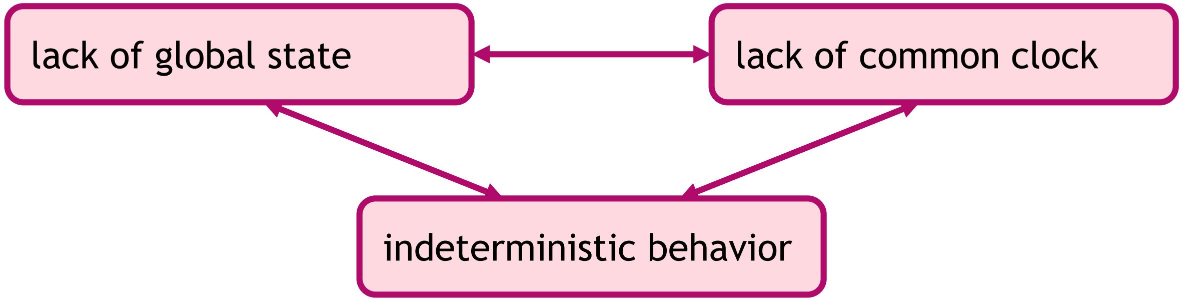
Basic Problem #2: Clocks Not 100% synchronized

Events $E_A$, $E_B$ at A, B with recorded times $t(E_A) < t(E_B)$:

- May have happened at $t(E_A) > t(E_B)$!!
- When is it safe to „believe" $t(E_A) < t(E_B)$ ?
- How to find out which is true? if undecidable: does distinction matter?

Basic Problem #3: Indeterminism – multiple execution of same system may yield different results

- Race conditions (messages from different senders, different threads) …
- … plus 'erroneous' underlying computer network →„correct program" has unpredictable result!!



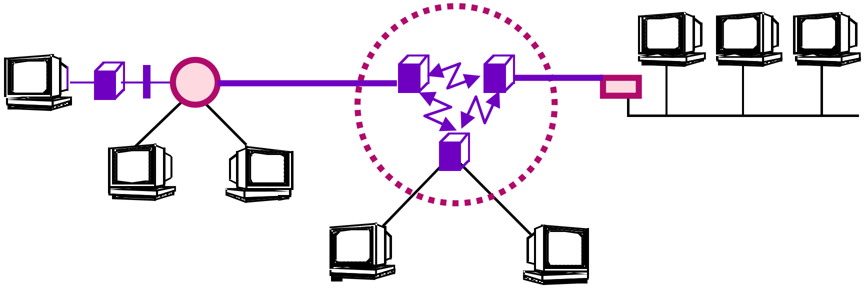
# Task 1.2: Abstraction Levels

Task 1.2: Describe the four abstraction levels of distributed systems.

# Task 1.2: Abstraction Levels

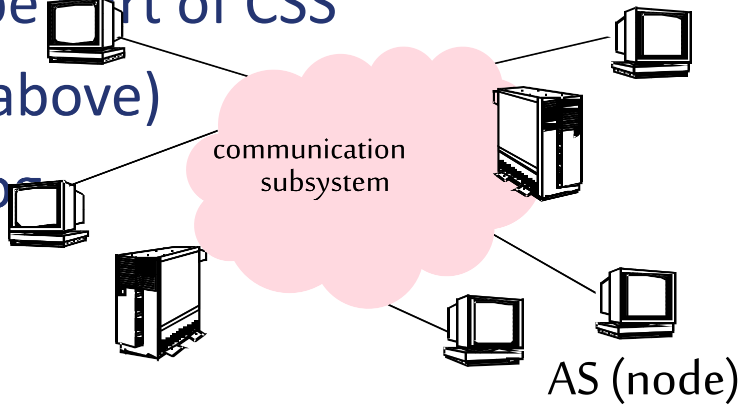
Level 1: physical configuration - seems irrelevant for DS:

  - Object of SysOp people
  - For DisProg people, should be abstracted from
- but:
  - ownership → cost (public net?), security, ... !
  - bandwidth etc. → performance
  - reliability?



## Level 2: logical configuration - „the“ CompNet abstraction!

- CSS = „cloud“, classes of ASs; AS may be part of CSS
- sometimes, abstraction too high (see above)
- usually, abstraction too low for DistProg



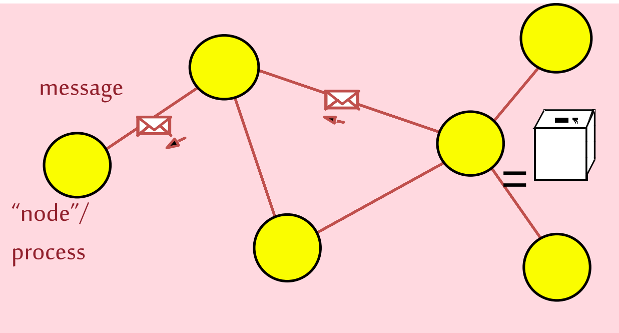
# Task 1.2: Abstraction Levels

## Level 3: process network (logical distribution): abstracts from real distribution

(≈ Distributed Program DistProg):

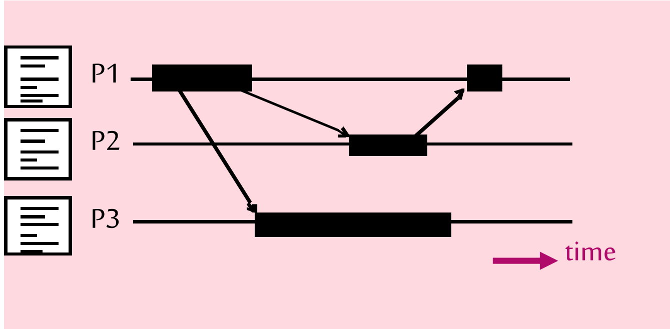
processes (objects, agents) exchange messages

- e.g., processes reached via mailboxes
- state distribution (no global view), see below
- no common time (no exact global "clock"), see below
- reliability, indeterminism assumed?
  depends further on abstraction & underlying support



## Level 4: Distributed Algorithm - abstracts from

- Target environment (→ reliability, performance, ...)
  not necessarily from reliability / performance issues
- Target process configuration i.e. # of processes (well, ought to…)
  not necessarily from interconnection / topology issues!
- Implementation language, platform, lifecycle



# Task 1.3: Transparency

- Task 1.3: Define the term „transparency“. Furthermore, name and describe four real world examples of different types of transparency.

# Task 1.3: Transparency

- Definition:

  **Transparency = „Concealment** from the user and the application programmer of the **separation of components** in a distributed system, so that the **system is perceived as a whole** rather than as a collection of independent components.

  - Note: „transparency“ is about „hiding something“ in English
  - We introduce 8 forms of transparency below, but literature varies → there are more!

1. Access transparency
   - Local & remote resources accessed using identical op’s
   - **Example:** A graphical file explorer user interface, which is the same for local and remote files
2. Location transparency
   - Resources accessed w/o knowledge of their physical/network location
   - **Example:** Web resource names, URL (does not contain Internet address)
3. Concurrency transparency
   - Several processes operate concurrently using shared resources without interference between them
   - **Example:** Distributed file system. Changes to a file by one client should not interfere with the operation of other clients simultaneously accessing or changing the same file.

# Task 1.3: Transparency

4. Replication transparency
   - Multiple instances of resources used (→ reliability, performance) w/o knowledge of replicas by users & programmers
   - **Example:** Distributed file system. A file may be represented by several copies at different locations without the user or application programmer being aware of that
5. Failure transparency
   - Concealment of faults, allowing users & programs to complete tasks despite HW/SW failure
   - **Example:** eMail is eventually delivered (even if communication link or server fails)
6. Mobility transparency
   - Movement of resources/clients w/o affecting users & programs
   - **Example:** Mobile phone users move between cells during a call
7. Performance transparency
   - Local/remote op (exec, data access) don't differ by orders of magnitude (most persistent problem!)
   - allows system to reconfigure to improve performance as loads vary
   - **Example:** Distributed file system. Client programs continue to perform satisfactorily while the load on the service varies within a specified range
8. Scaling transparency
   - Allows system/applications to expand in scale without change to system structure or application algorithms
   - **Example:** Distributed file system. The service is incrementally expanded to deal with larger network size

# Task 1.4: “Programming” Abstractions

Task 1.4: Describe the four principles of programming abstractions with respect to distributed software development

# Task 1.4: “Programming” Abstractions

4 principles for abstractions wrt. Distributed Software Development

1. **Distributed operating system approach**
   - Support for distributed programming is part of operating system
   - Pro: Quite general solution, *parallel* programming paradigm
   - Con: Needs wide-scale adoption of the same system
     - Large systems always heterogeneous
2. **Distributed database approach**
   - Same as above, except OS replaced by a database system
   - Pro: Allows for all DB features (semantics, …), isolated *sequential* prog’s
   - Con: Independent applications with shared database
   - Con: Many distributed algorithms hard to realize in this case
3. **Protocol approach** for dedicated purposes (Xwindow etc.)
   - Standardized protocols for connecting to servers (e.g., HTTP)
   - Pro: Open, global; ~ sequential prog’s (+ callback threads)
   - Con: Limited to standard functionalities
4. **In this lecture: Distributed Programming ‘Language’ approach**
   - the only one that is wide spread – up to now!