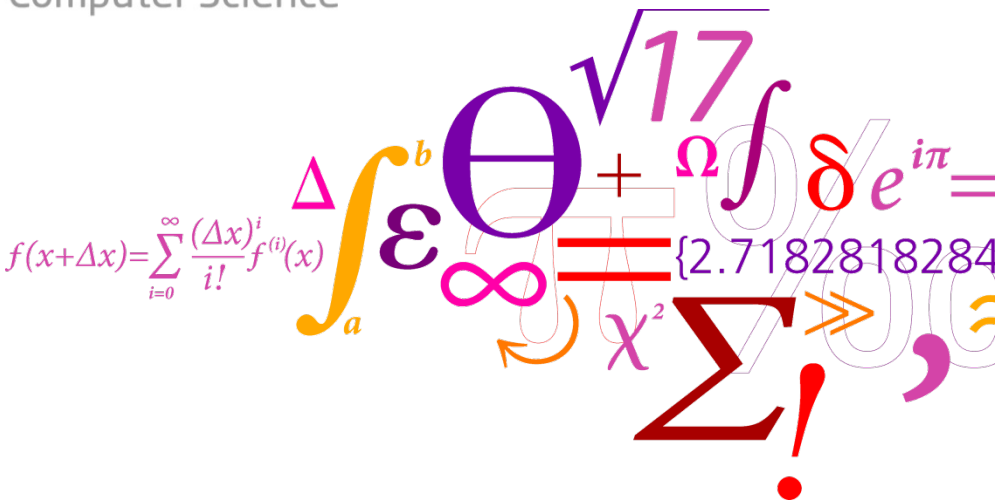
Software Engineering 2

A practical course in software engineering

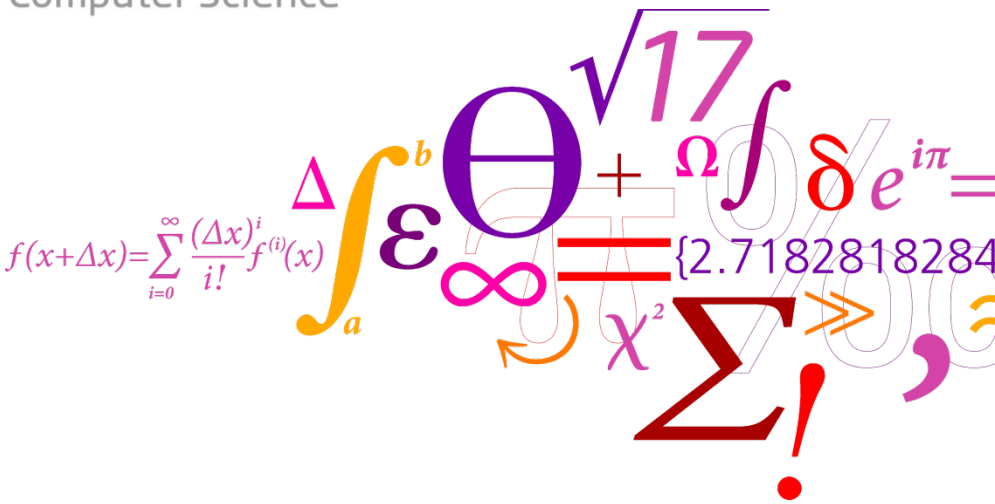
Ekkart Kindler

DTU Compute
Department of Applied Mathematics and Computer Science



IV. Working Together

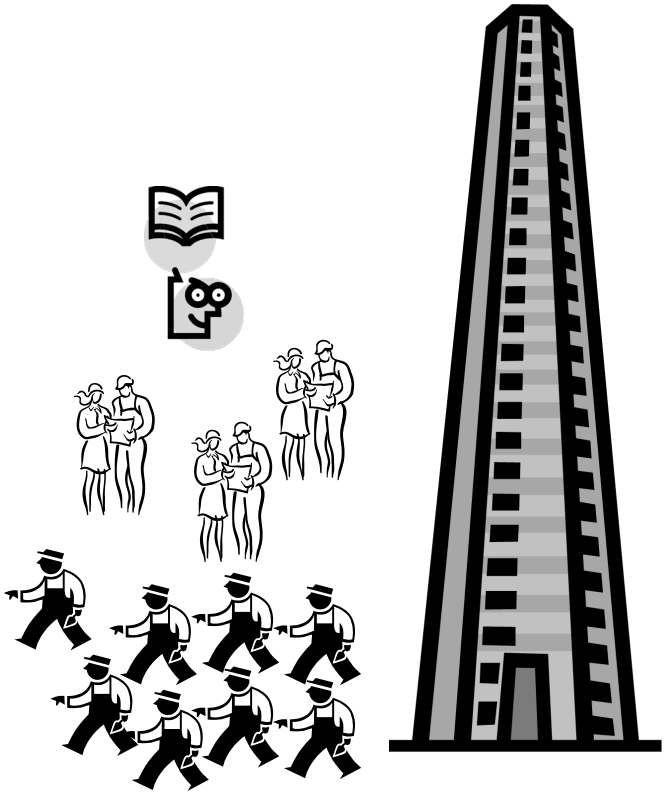
DTU Compute
Department of Applied Mathematics and Computer Science



Working together

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

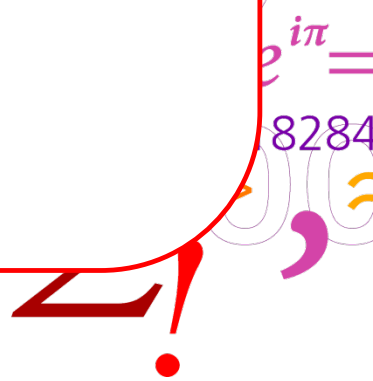
- Management
- Process Models
- Version Management Systems
- Collaborative Development Environments



Parts of this course are based on:

Helmut Balzert: Lehrbuch der Software-Technik:
Software-Entwicklung.
Spektrum Akademischer Verlag 1996.

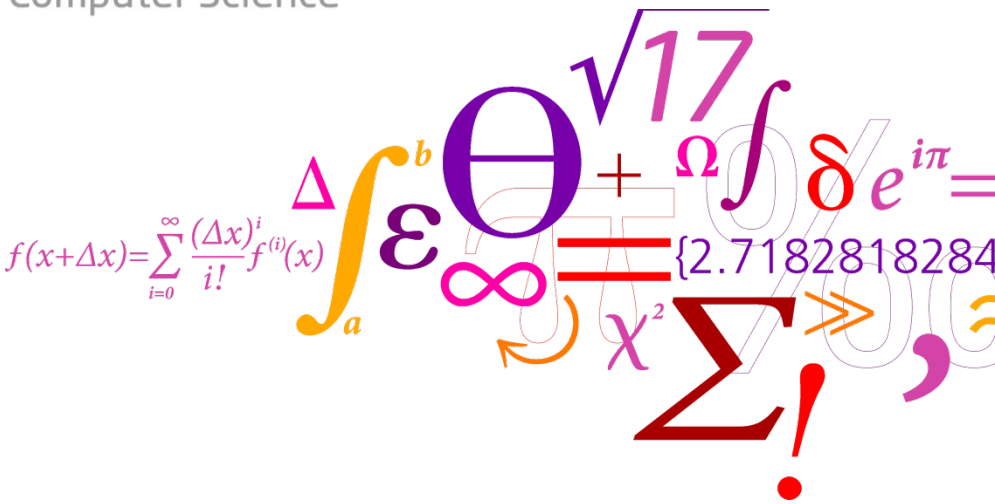
Helmut Balzert: Lehrbuch der Software-Technik:
Software-Management.
Spektrum Akademischer Verlag 1998.



IV. Working together

IV.1. Management

DTU Compute
Department of Applied Mathematics and Computer Science



Definition

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

(Software) Management

Planning, organizing, leading, monitoring and controlling the software development process.

Management Goals

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

Define goals and make sure that they will be achieved

General goals:

- Increase productivity

  decrease development costs

- Increase quality

  Increase product value

In short:

Make a profit and increase it!

Short- vs. Long-term Goals

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

Software Management requires

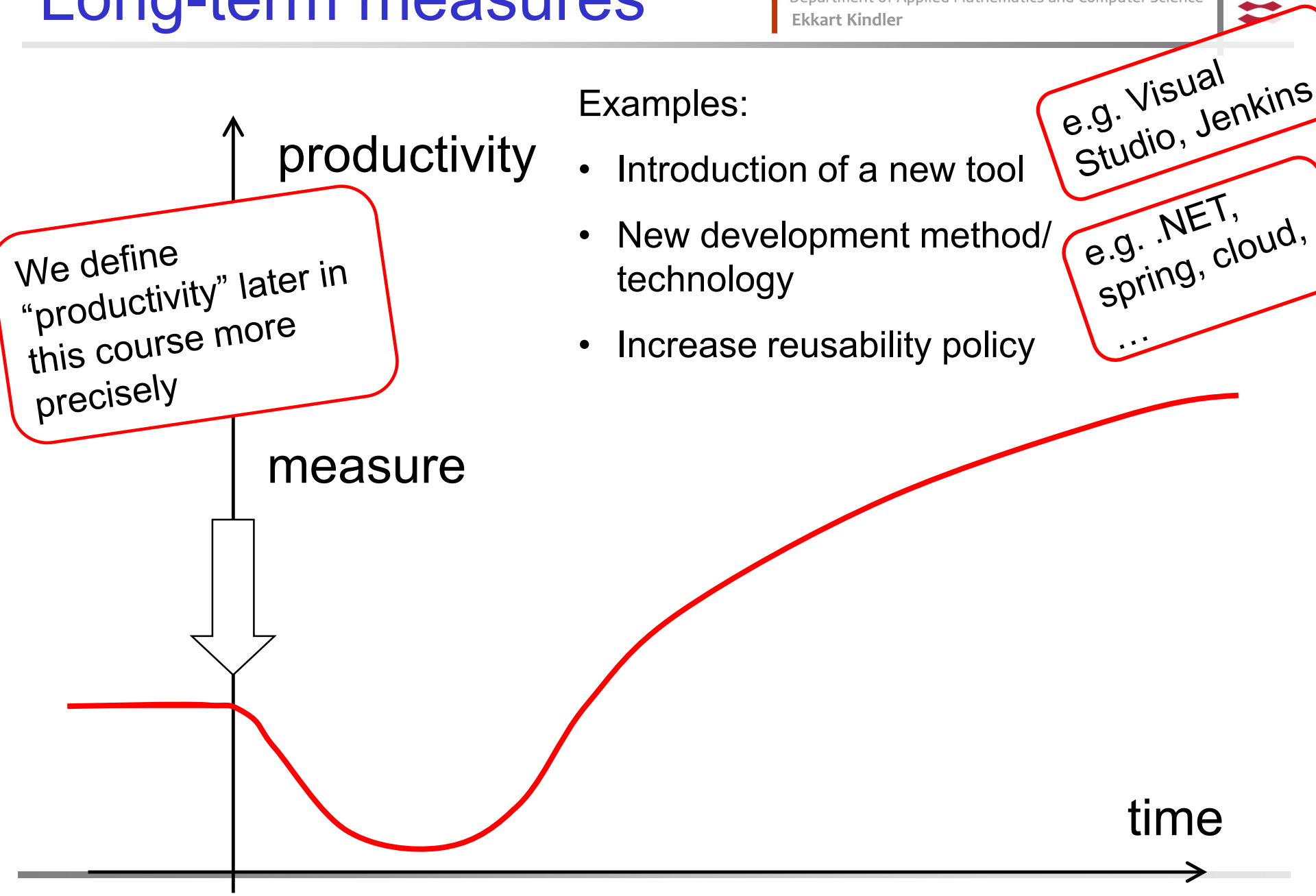
- short-term
  (within a project or even within a phase)

  and
- long-term
  (spanning more projects)

measures

Long-term measures

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler



Examples:

- Introduction of a new tool
- New development method/ technology
- Increase reusability policy

e.g. Visual Studio, Jenkins

e.g. .NET, spring, cloud, …

Short- vs. Long-term

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

Short-term measures and long-term measures often have opposite effects:

- short-term measures result in long-term loss in productivity
  (e.g. quick hack for finishing a project; programming language)
- long-term measures result in short-term loss of productivity
  (frustration, learning curve)

Management

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

- **Planning**
  - set goal
  - set dates
  - define course of action
  - allocate resources
  - …
- **Organization**
  - assign tasks
  - define organisation structures
  - assign responsibilities
  - …

Management

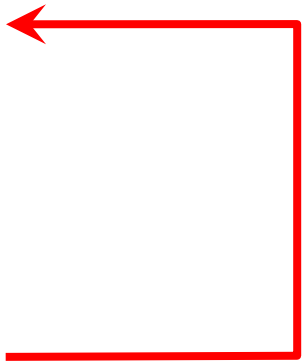
DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

- **Leading**
  - lead and motivate team members
  - improve communication
  - solve conflicts
  - …
- **Monitoring and controlling**
  - check progress
  - identify problems (early)
  - produce relief
  - …

Management cycle

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

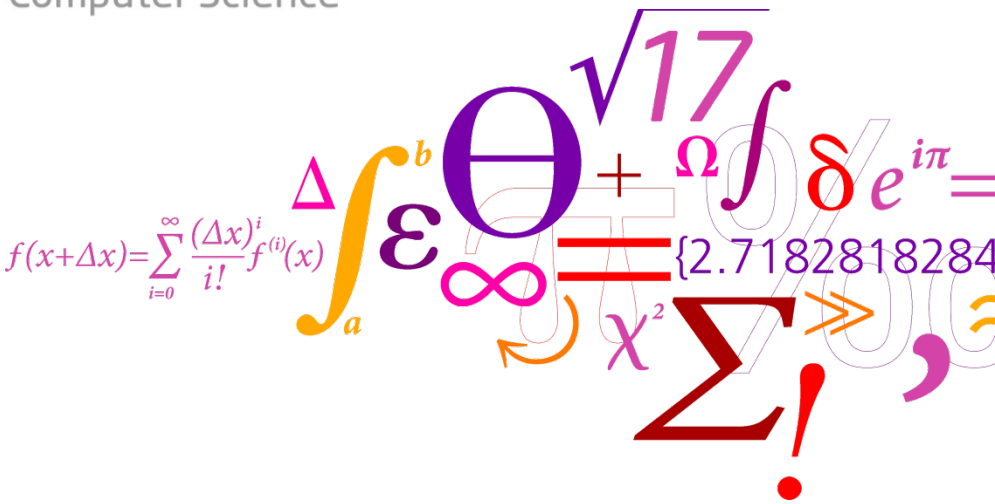
- Planning
- Organization
- Leading
- Monitoring & controlling



IV. Working together

IV.2. Coordination

DTU Compute
Department of Applied Mathematics and Computer Science



Mechanisms

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

- Mutual agreement
- Instruction

- Standardization of procedures
- Standardization of the product
- Standardization of the qualification the team members

- Socialisation

Works for small groups of people who know each other well (from other projects)

Clear Responsibilities

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

- Having many discussions is very good

But, they are useless, unless

- the **results** are **put on record**; and
- **problems** are **clearly stated**,
- it is fixed
  - **what** should be done
  - by **whom**
  - by **when**
- and kept **track of**.

Put agenda and minutes for every meeting to the repository (in particular group meetings)

Communication

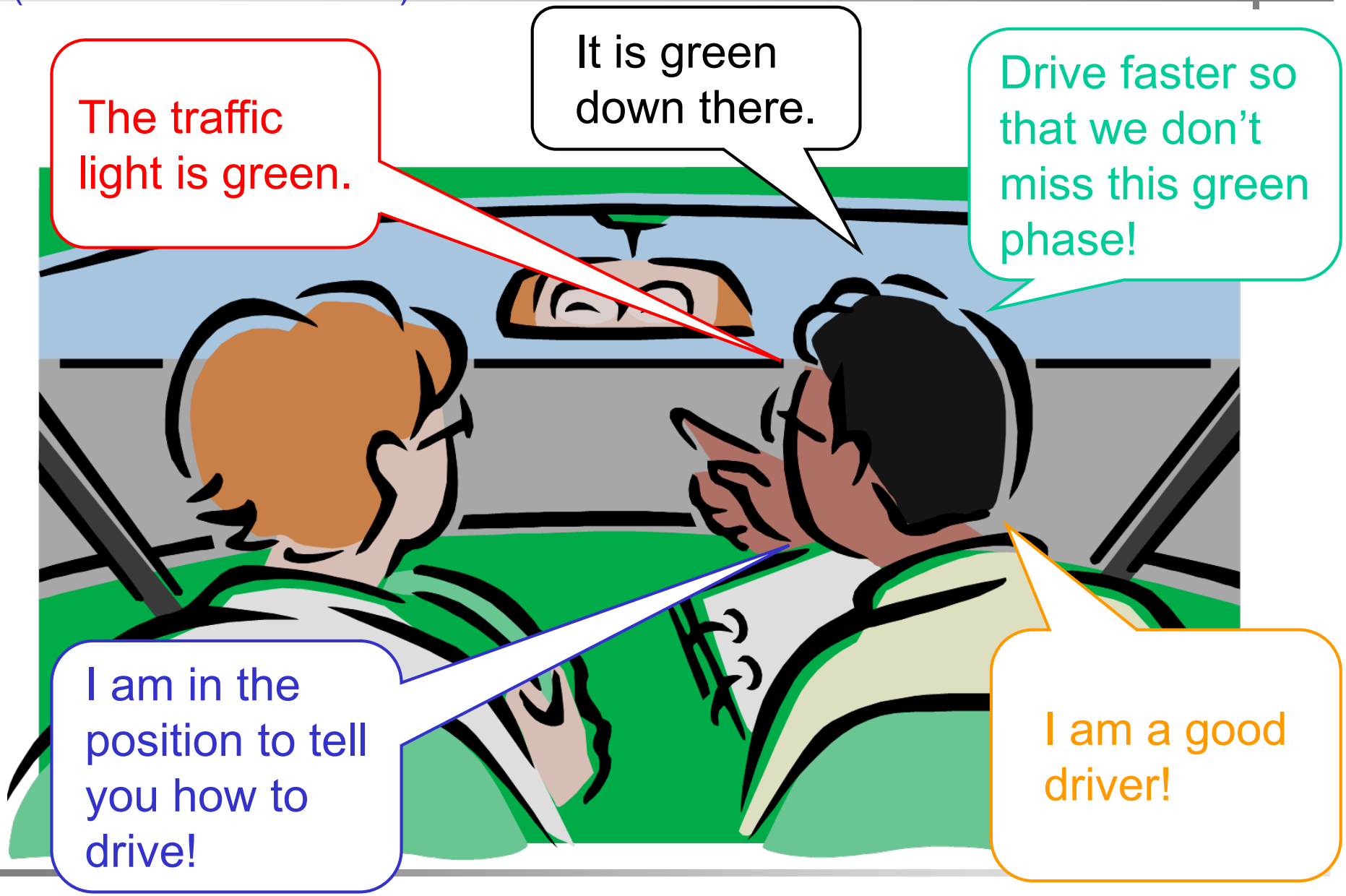
Coordination and leadership require

communication

Four sides of a statement

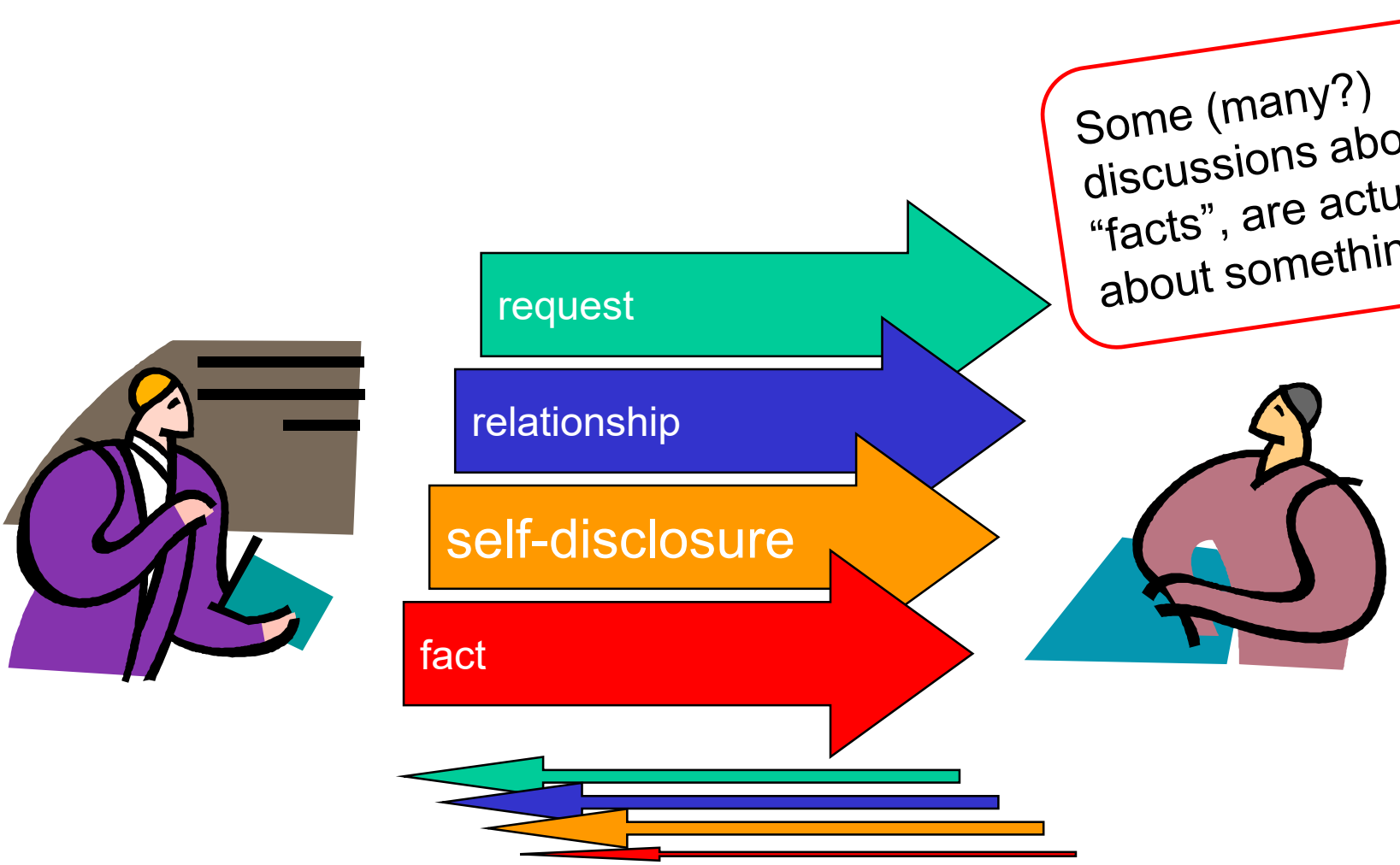
(after Schulz von Thun)

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler



Four sides of a statement
(after Schulz von Thun)

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler



Good Leadership

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

- instruct team members
- delegate responsibilities
- motivate team members
- coordinate activities
- stir and improve communication
- identify and solve conflicts
- …

Meta-level discussions (but not too often)!

Rules for Leaders

Not all are relevant for our short project!

- sets challenging (but achievable) goals
- assigns tasks and required capabilities for team members
- explains tasks and projects coherently and thoroughly
- defines (checkable) performance criteria
- identifies sources of problems so that team members can adjust

Rules after:
Derschka / Balzert II

Rules for Leaders

- expresses more approval than criticism
- supports team members
- communicates high personal expectations in an individual way
- puts emphasis on human relations
- gives team members the chance to find and correct their own mistakes
- includes team members when making decisions

Rules for Leaders

- rewards and conveys team members with respect to innovation and willingness to carry risks
- regularly discusses the performance with his team members
- rewards excellent performance

Rules for Leaders

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

- Organizes meetings such that cooperation is encouraged
- Shows personal commitment

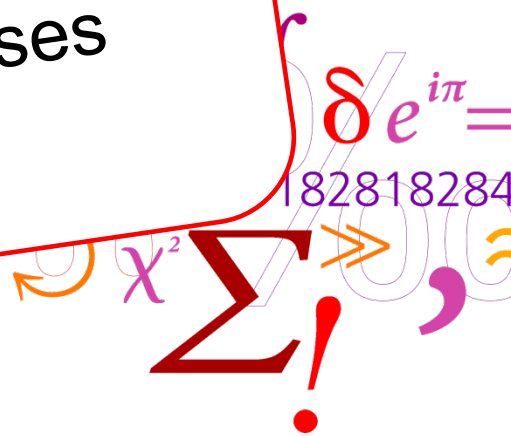
Makes these rules his/her rules and does not apply them schematically.

IV. Working together

IV.3. Process models

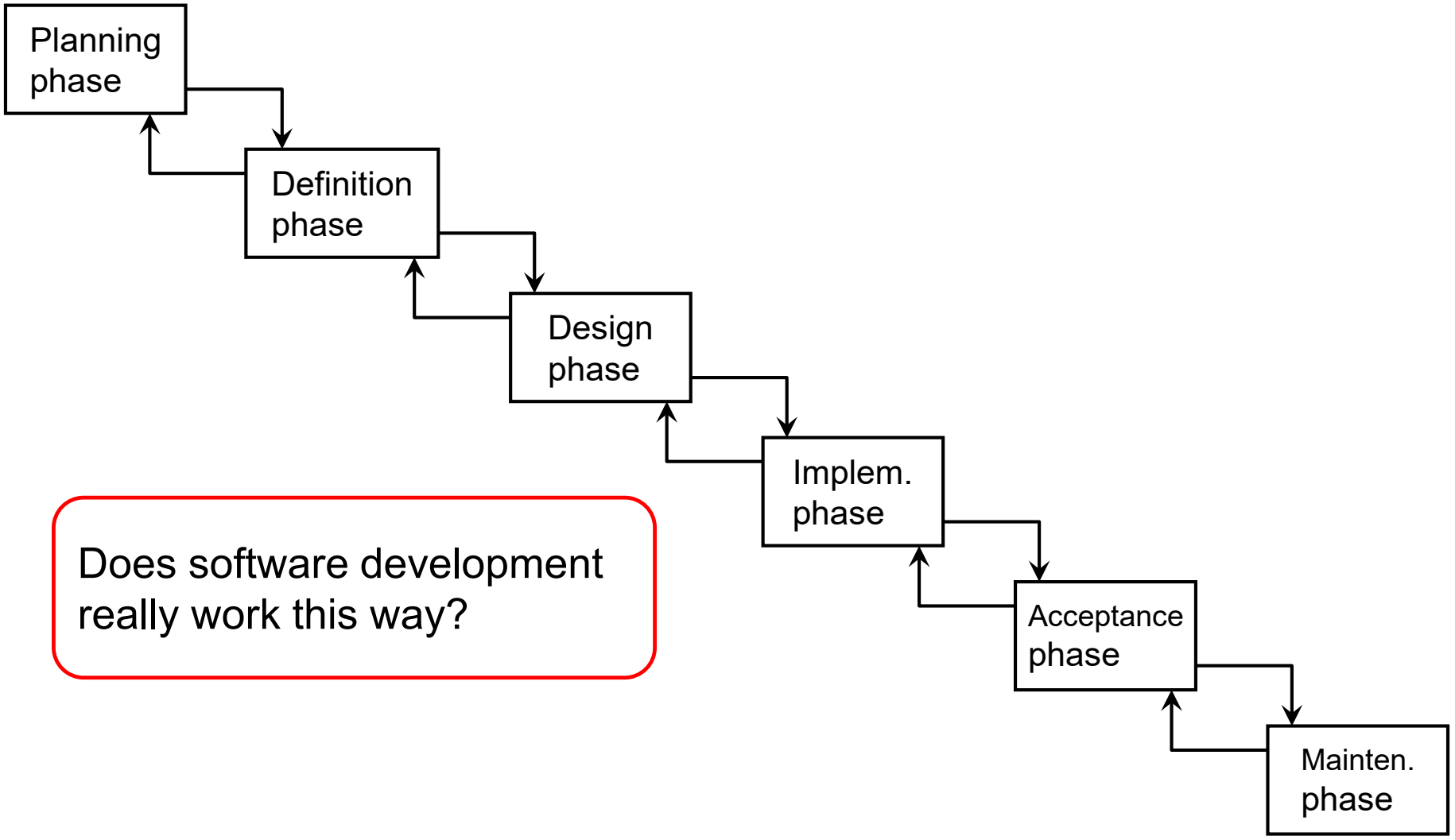
DTU Compute
Department of Ap

Coordination mechanism:
Standardization of processes



Waterfall Model

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler



Does software development really work this way?

Process Models

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

- **Process models** are the distilled experience of project plans of successful software projects
- they are idealized and abstract from (many) details
- this is their strength (and maybe also their weakness)

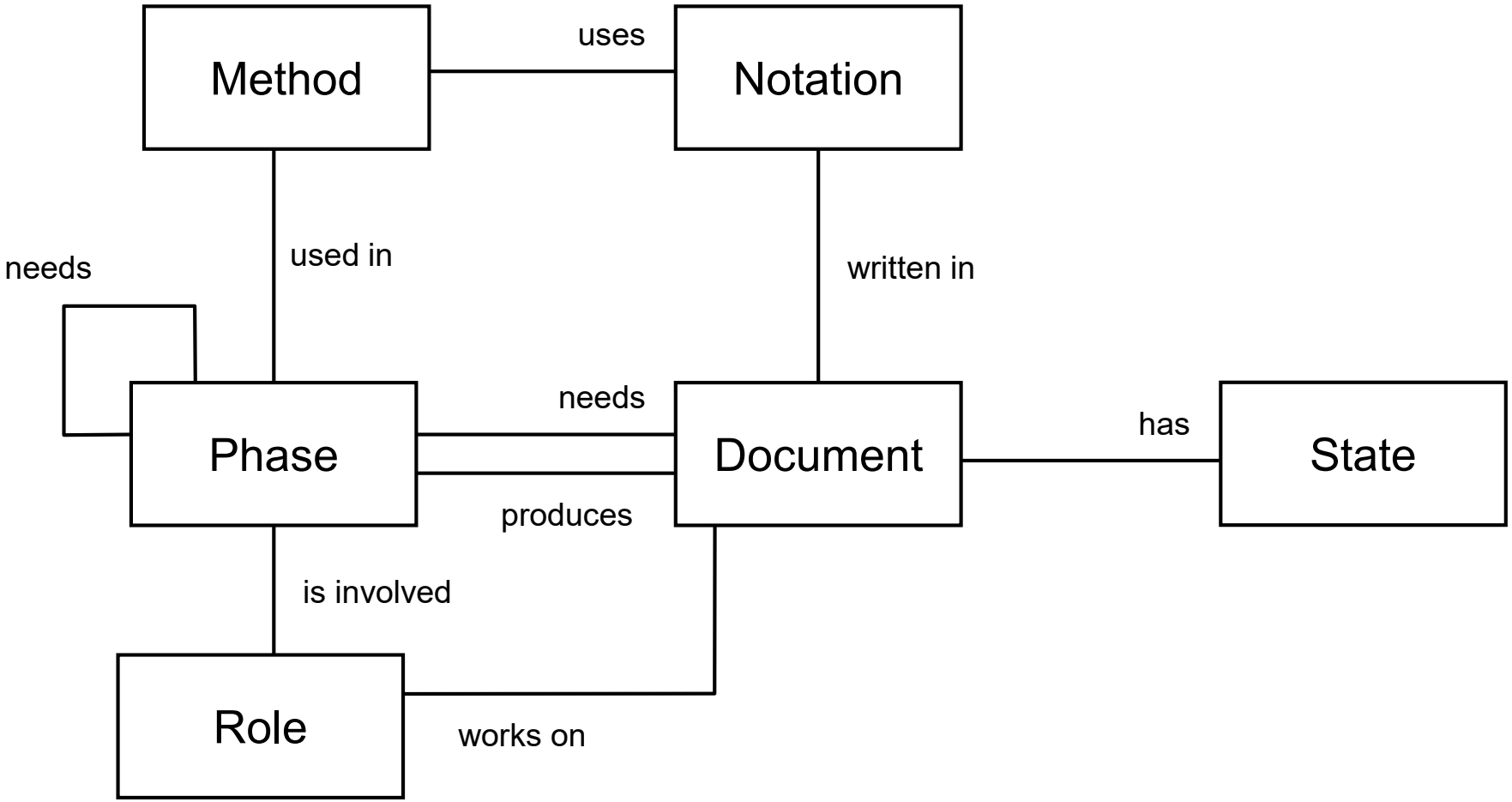
- A **project plan** is a refined, extended, modified and more concrete process model

The waterfall model is extremely simple.
Still it has its values!

Even simpler: Why → What → How

Meta model

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler



Project plan

- In a project plan, phases are split into tasks and task might be further split into sub tasks
- Moreover more concrete information is added:
  - begin / end
  - roles of team members and assignment to concrete tasks
  - effort for each task
  - time spent of each member on the assigned tasks

In agile, the plan is mostly the (prioritized) user stories and tasks for the next release!

Project plans

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

- There are many different notations for such plans
  - Gantt diagrams
  - Many tools (MS Project etc.)
- It is easy to analyse such plans:
  - timing
  - critical paths (buffer)
  - load on team and every team member
  - planning costs
  - controlling
  - …

In agile, adjust for every sprint, no explicit long-term planning and analysis.

Milestones

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

- in order to monitor the progress, the project plan defines **milestones**
- a milestone is a set of documents/artefacts (including code) that must be provided at a specific time and in a specified state
- typically, the documents at the end of a phase or task or made a milestone

How to chose milestones

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

- Verifiable
  - It must be verifiable whether the milestone is reached or not
    ("some document is 75% finished" is **NOT** a milestone)
- Manageable
  - The required documents can be produced in a reasonable amount of time
    (weeks or months, not years)

In agile, even shorter 2-3 weeks!

- Regular
  - The milestones should be in regular intervals

Not a milestone

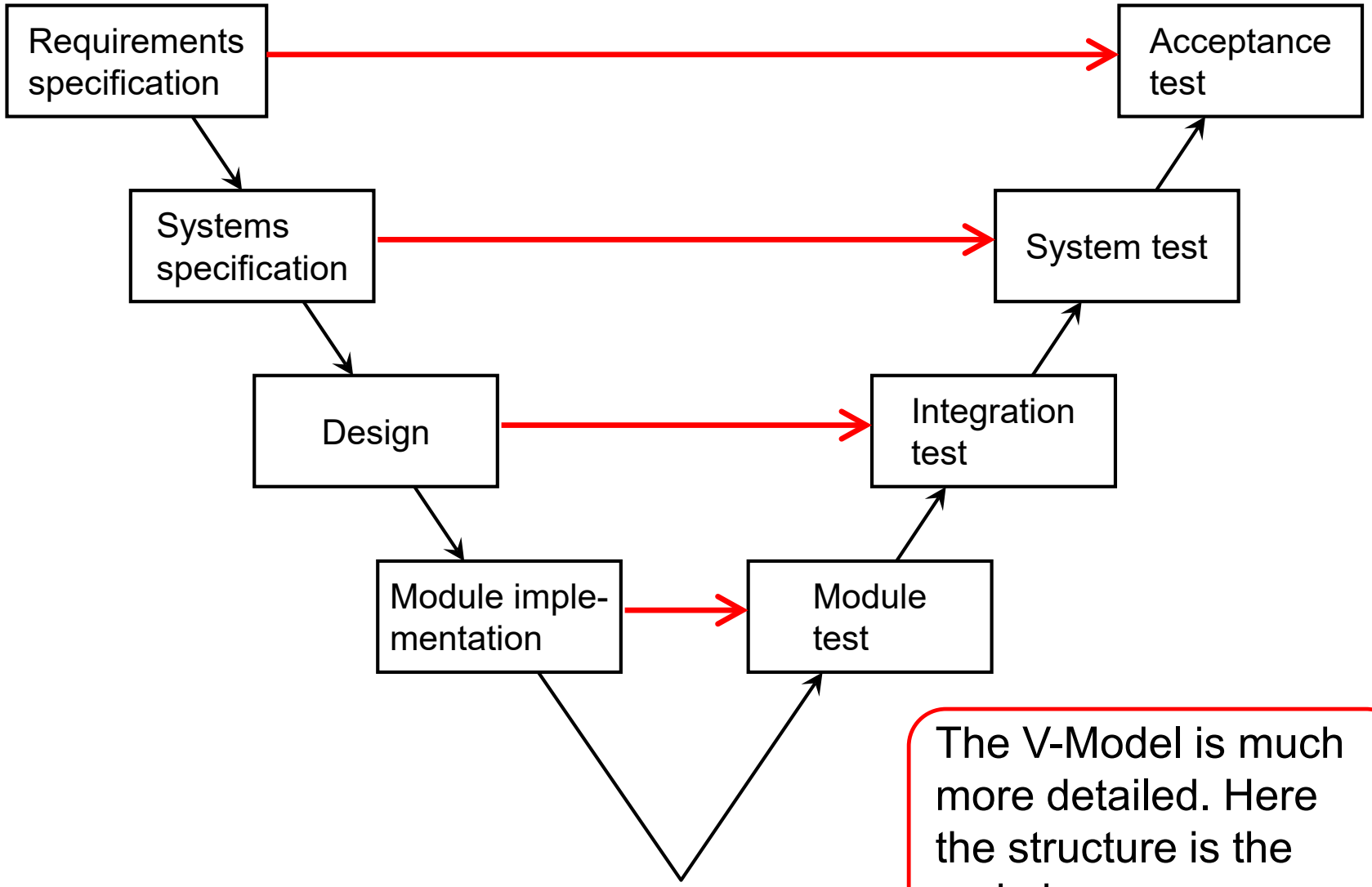
DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

- A data is not a milestone!
  But, a milestone has a date!

- A task is not a milestone!
  But, task may contribute to a milestone (in form of some artefact, which is deliverd)

The V-Model (rough idea)

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler



The V-Model is much more detailed. Here the structure is the main issue.

Other models

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

- Prototype models
- Evolutionary or incremental models
- Spiral model
- …

These models have their advantage (dependent on type and size of the project). Sometimes, however, they are used as bad excuse for not planning. (But, that is not the fault of these models).

Idea:

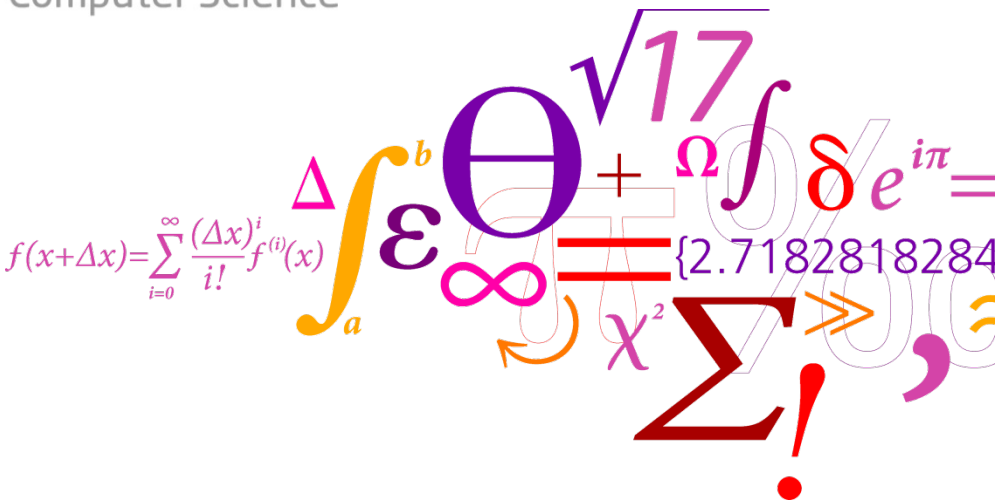
- Stepwise development of product
- Early prototypes (“executable milestones”)
- “Maintenance as part of the development”

In our project: Agile!

IV. Working together

IV.4. Version Management

DTU Compute
Department of Applied Mathematics and Computer Science



Motivation

Situation:

- Software consists of many different documents (requirements, specification, design, implementation, documentation, handbook, …)
- Software development is team work

That is the very definition of software.

Motivation

Consequently:

- Many different people access (and possibly change) the same set of documents
- Often, different people work on the same document concurrently (independently of each other at the same time)

Motivation

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

Result: Typical problems / questions:

- Where is the up to date version?
- Who has the last working version?
- Where is the version from September 25, 2020?
- Who has the version that we presented recently at the meeting with our customer?

Motivation

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

Result: Typical problems / questions:

- Who has the documentation that corresponds to the current implementation?
- Who has undone my changes from yesterday? And why?
- All my documents are lost!
- …

Motivation

Simple “Solutions”:

- Shared file system
- Conventions and rules for naming files (e.g. append: version number and date)
- Policies for changing documents
- Appoint persons responsible for documents

Motivation

Simple "solutions" are not:

- conventions will be violated
- coordination is inefficient and might cause long delays
- variants and different configurations need to be managed by hand
- …

→mental capacity should not be wasted with these "details"

Conventions need to be enforced!

Motivation

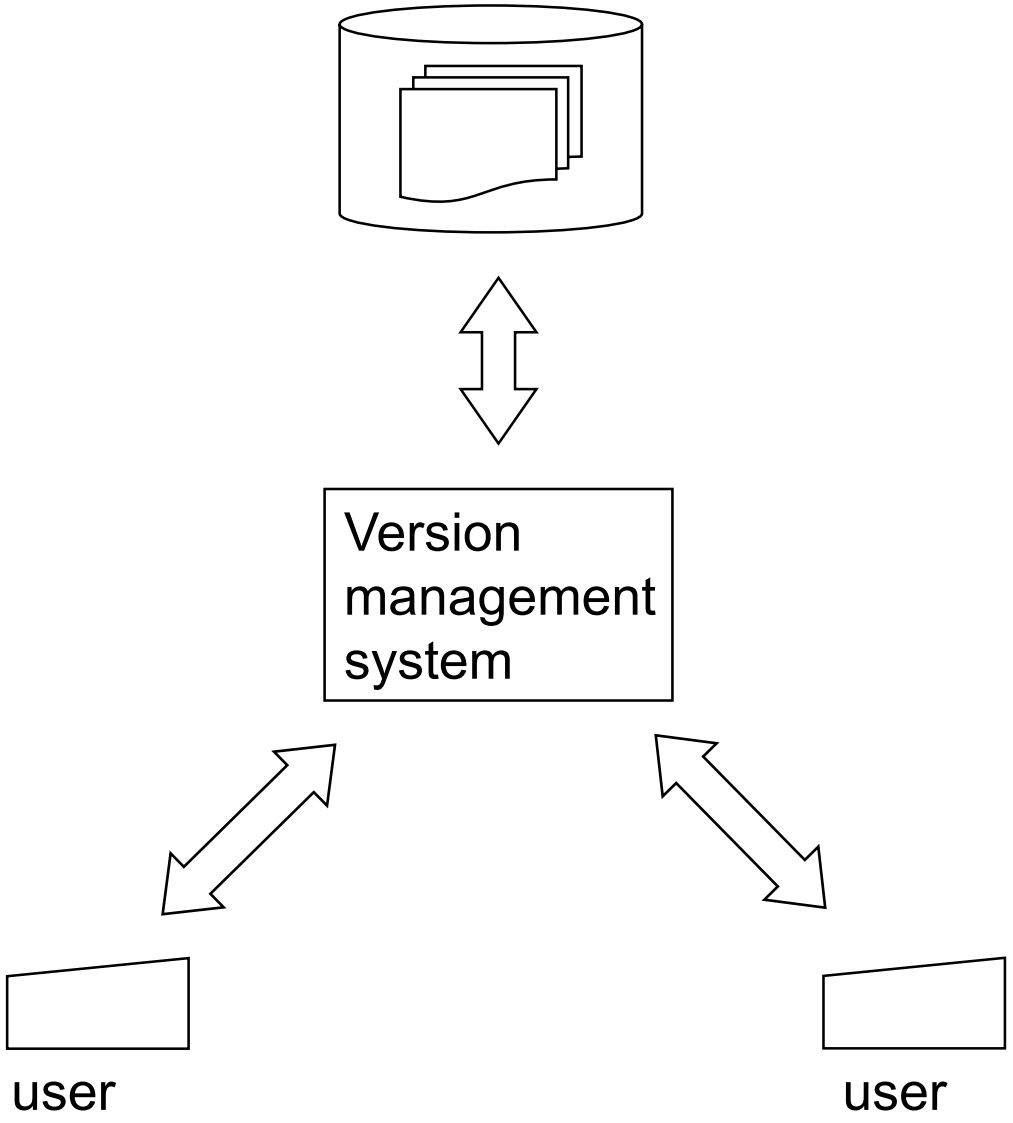
Version and Configuration Management Systems solve all these problems (almost) without any extra effort

- when applied properly and
- with good policies

They have even some extra benefits (e.g. simple backups).

Version Management Systems

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler



documents in hierarchical structure

coordinates accesses and modifications (maintains consistency)

Access, modification, and creation of documents

Mechanisms

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

Pessimistic mechanisms:

A document can never be accessed and changed by two different people at the same time (→ locking / checkout / checkin).

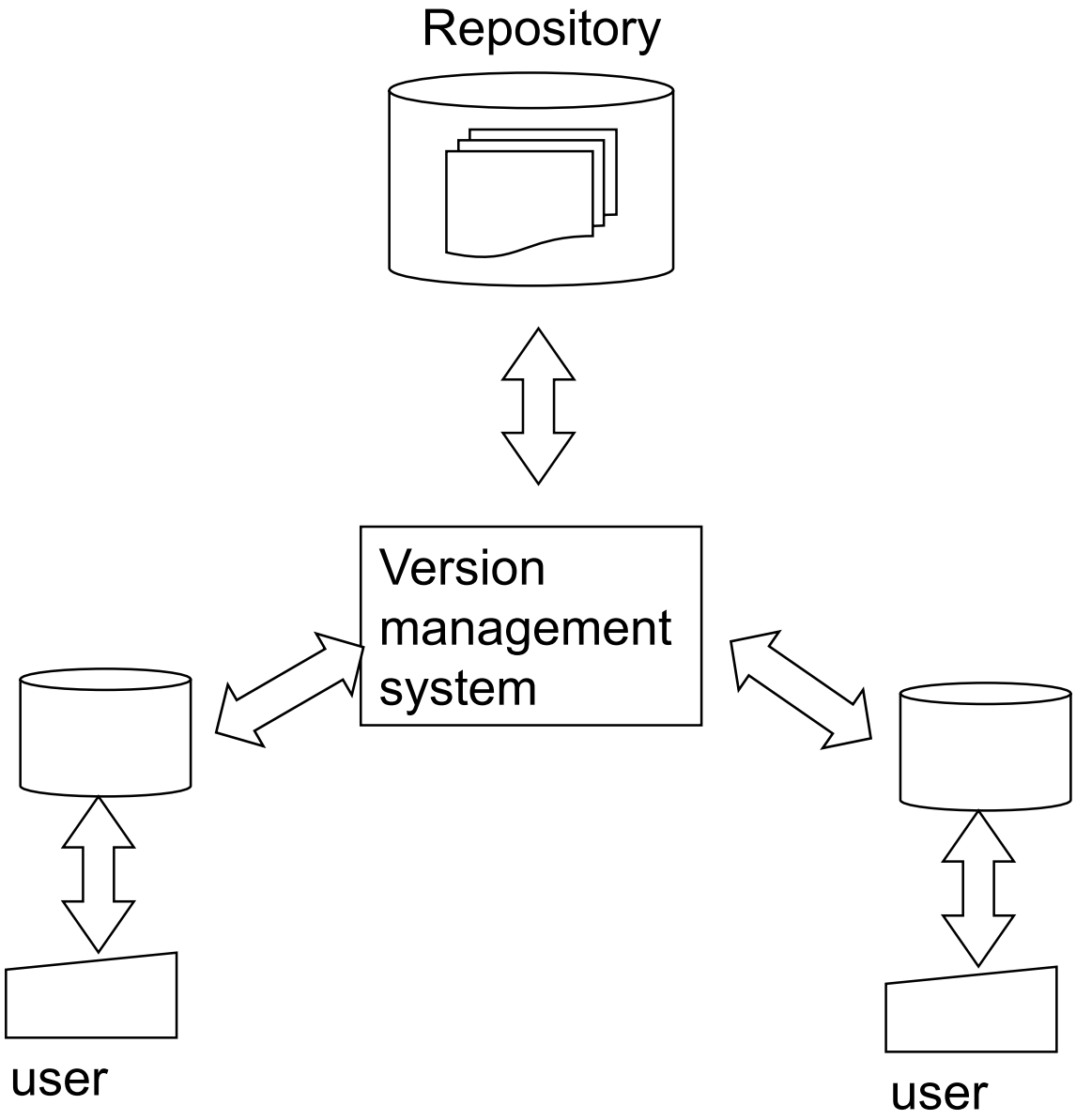
Optimistic mechanism:

A document can be changed by different people at the same time; the system detects and integrates the different changes (→ commit / merge / update).

Problem: This cannot always be done fully automatically.

Optimistic Approach

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler



Local copy of the current state of the repository (working copy)

Optimistic Approach

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

- An **`update`** updates the user's working copy with the information from the repository
- A **`commit`** brings the changes of the working copy to the repository (with a new version number)

These exist as command line commands as well as in the Eclipse GUI (Team).

Optimistic Approach

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

- The user can change files in his/her local working copy at any time (no locks).

- The working copy and the repository are synchronized with the commands **`update`** und **`commit`** (for all files or selected set of files or a single file).

Problem

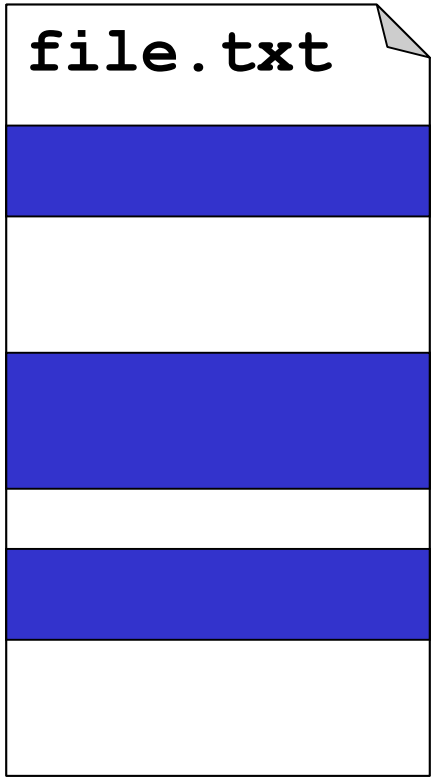
DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

- What happens, when a user executes an **`update`** and there are changes in the local working copy already?

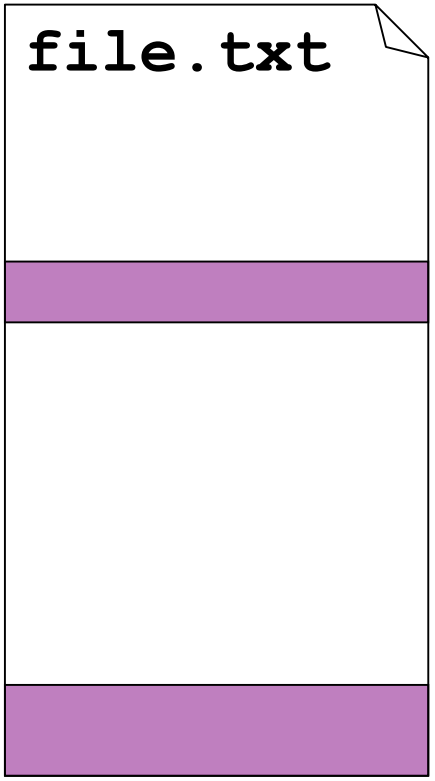
→**Merge** of the changes in the repository (since the last update) and the changes in the working copy.

Merge: Scenario 1

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler



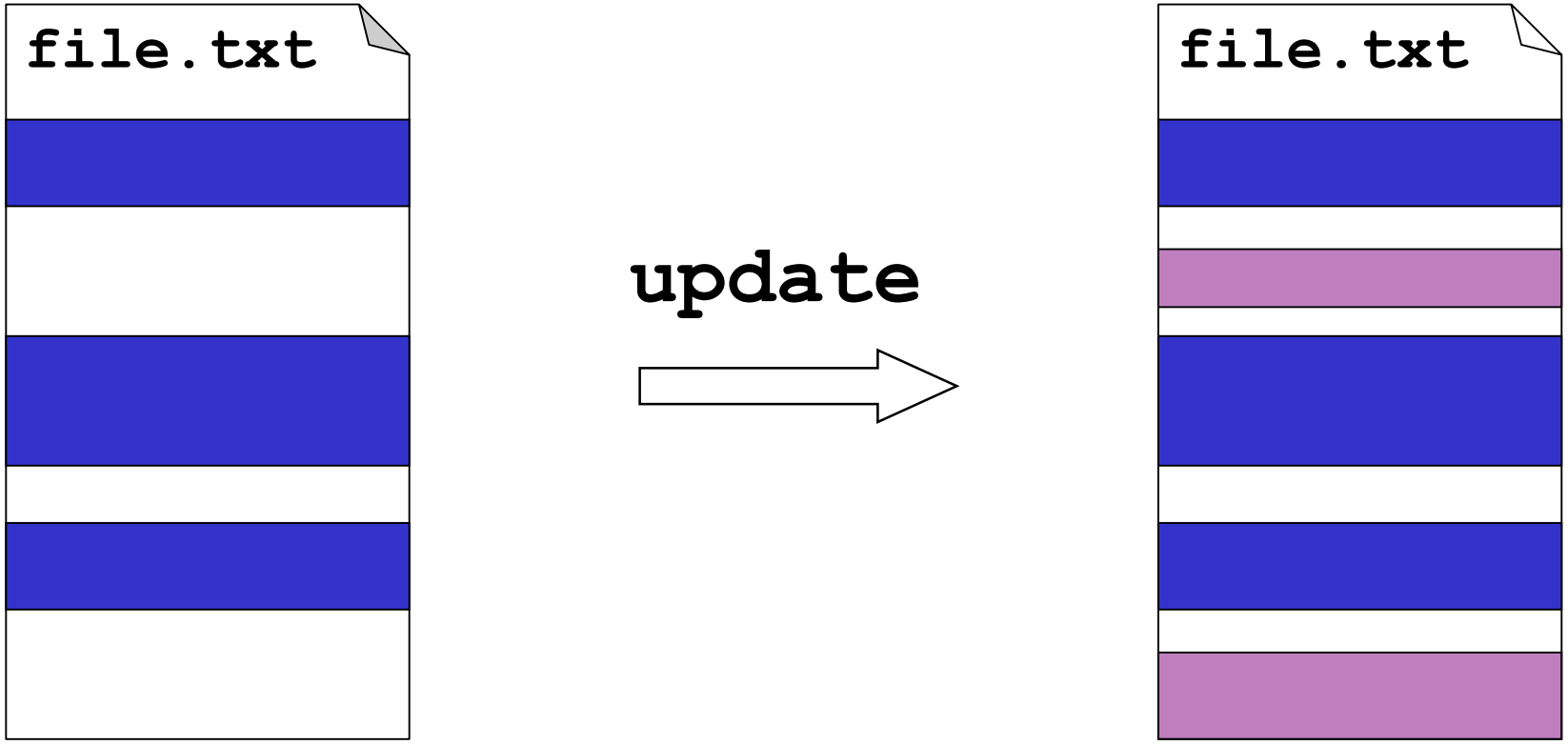
Changes in repository
(by a commit of a different user)



Changes in working copy

Merge: Scenario 1

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

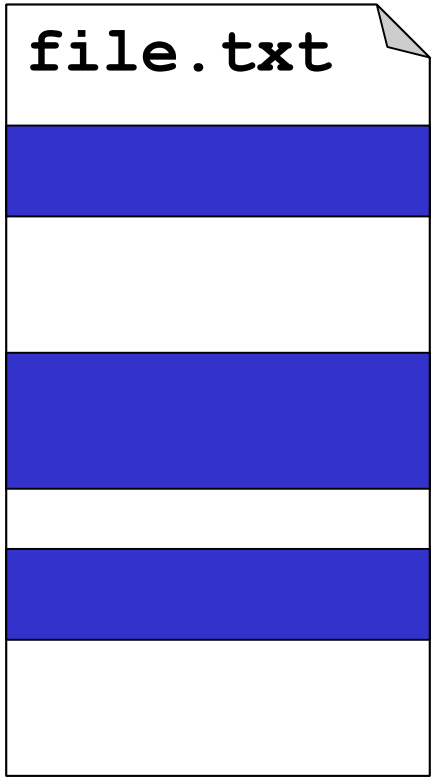


Changes in repository

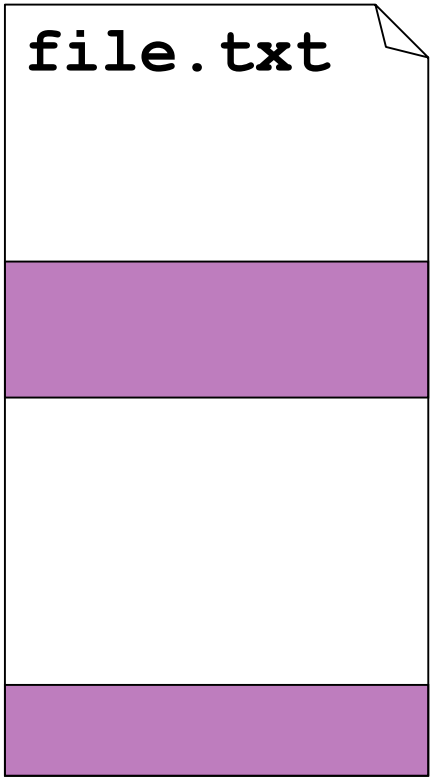
Updated working copy

Merge: Scenario 2

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler



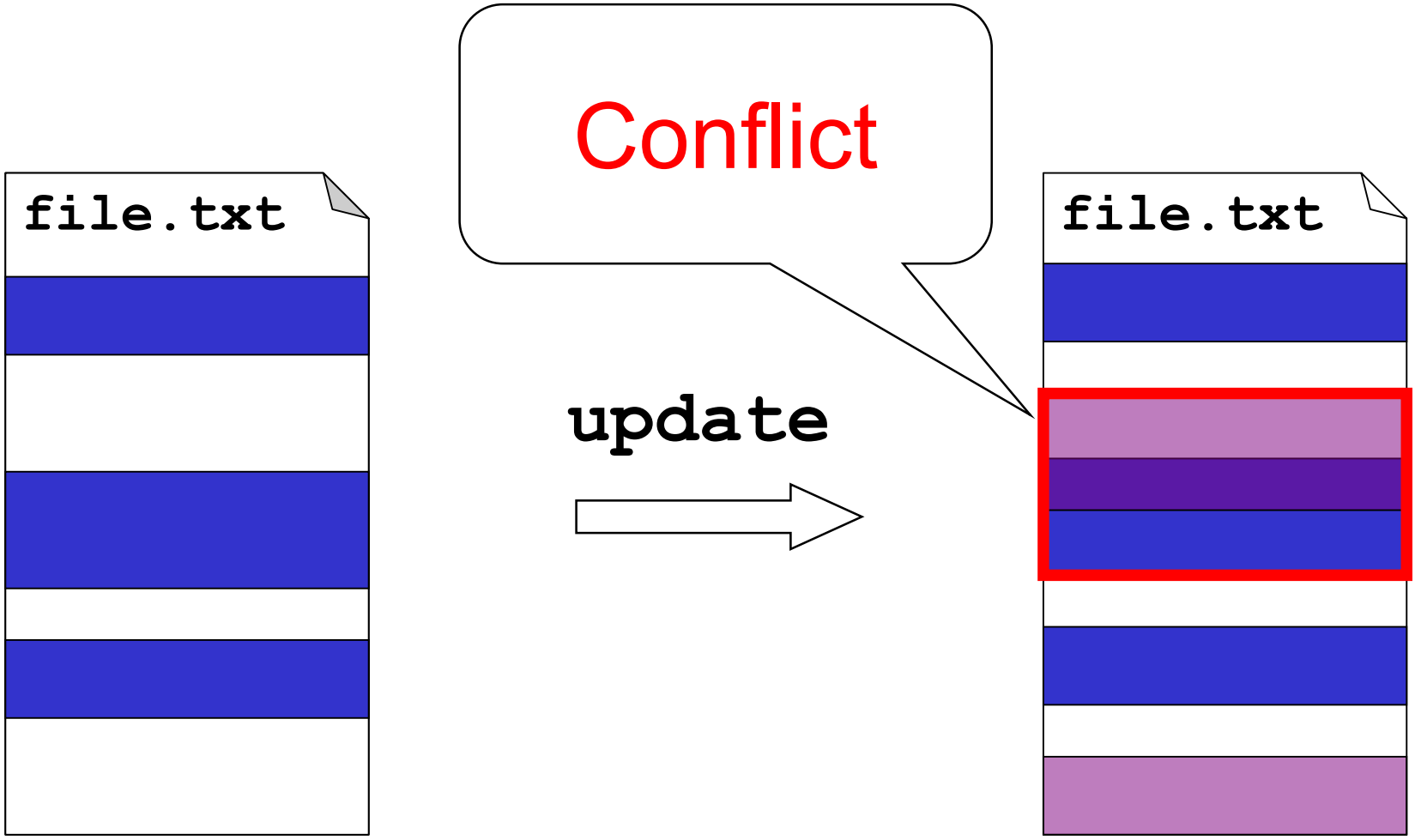
Changes in repository



Changes in working copy

Merge: Scenario 2

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler



Changes in repository

Changes in working copy

Conflicts

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

- Conflicts will be indicated in the files in the working copy:

```
<<<<<<
   Change 1
------
   Change 2
>>>>>>
```

- Conflicts must be resolved manually by the user in his working copy.

Remarks: Binary Files

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

- Merges make sense in text files only!
- Binary files should not be merged.
- Which files are binary must be made explicit to the version management system.

SVN and git make some good guesses!

- MS Word documents are binary for most version management systems

Problems

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

- What happens, when the user executes a **`commit`** without getting all changes from the repository first?

→ **Impossible!!**

→ By this strategy conflicts only occur in working copies!

→ There is always a user responsible for resolving it.

→ Before committing, a user needs to execute an update (and if necessary resolve the conflict).

Comparison

Pessimistic approach:

\+ no conflicts

\-- no concurrent work on the same object
(for long files or „forgotten“ check-ins this is very problematic)

Optimistic Approach:

\- conflicts (fortunately very rare)

\++ concurrent work possible
(the last one needs to clean up the mess)

Comparison

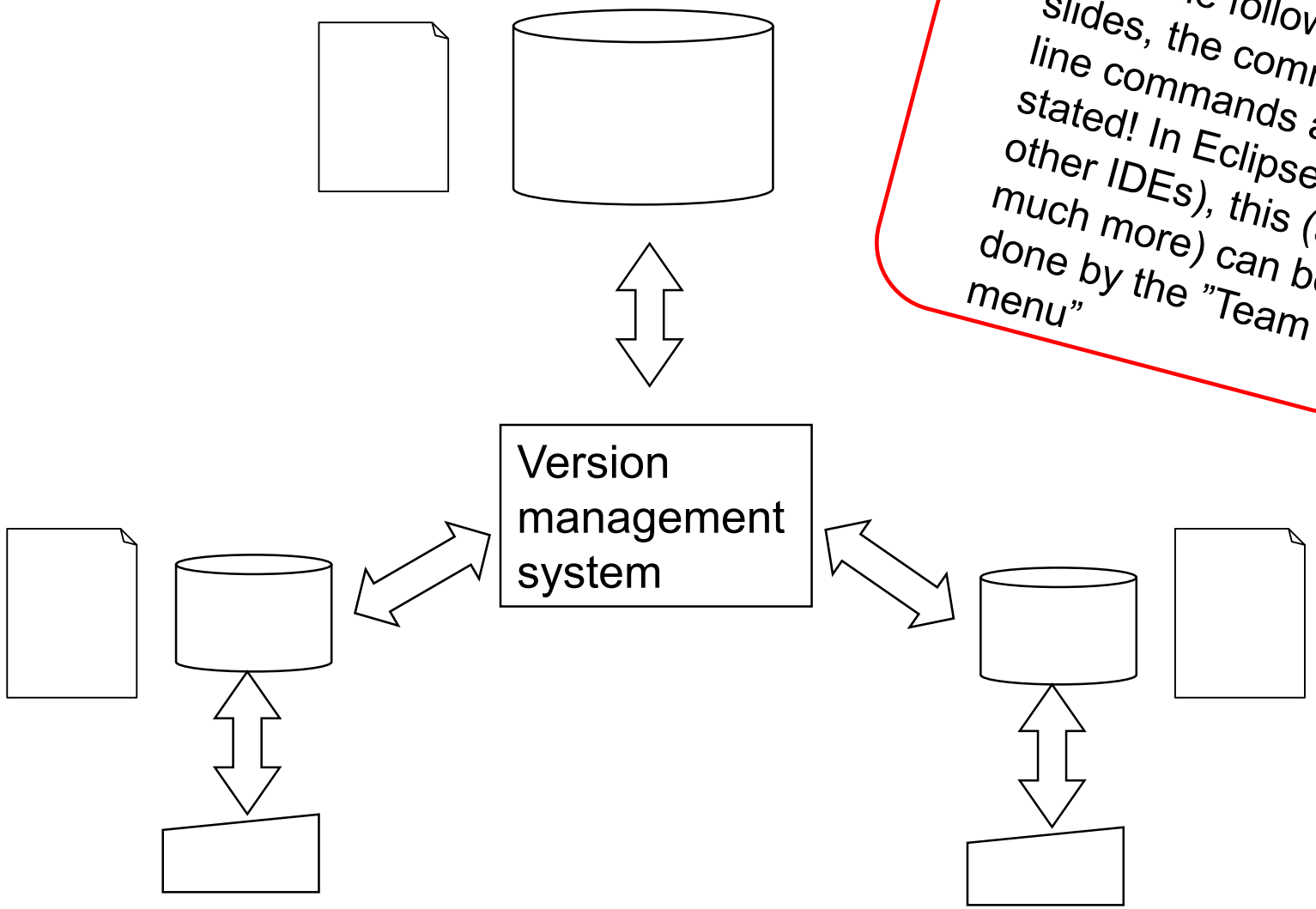
For typical software projects (big teams working concurrently), optimistic consistency mechanisms have turned out to be more appropriate:

- concurrent work
- in combination with responsibilities, conflicts are rare

Typical scenario

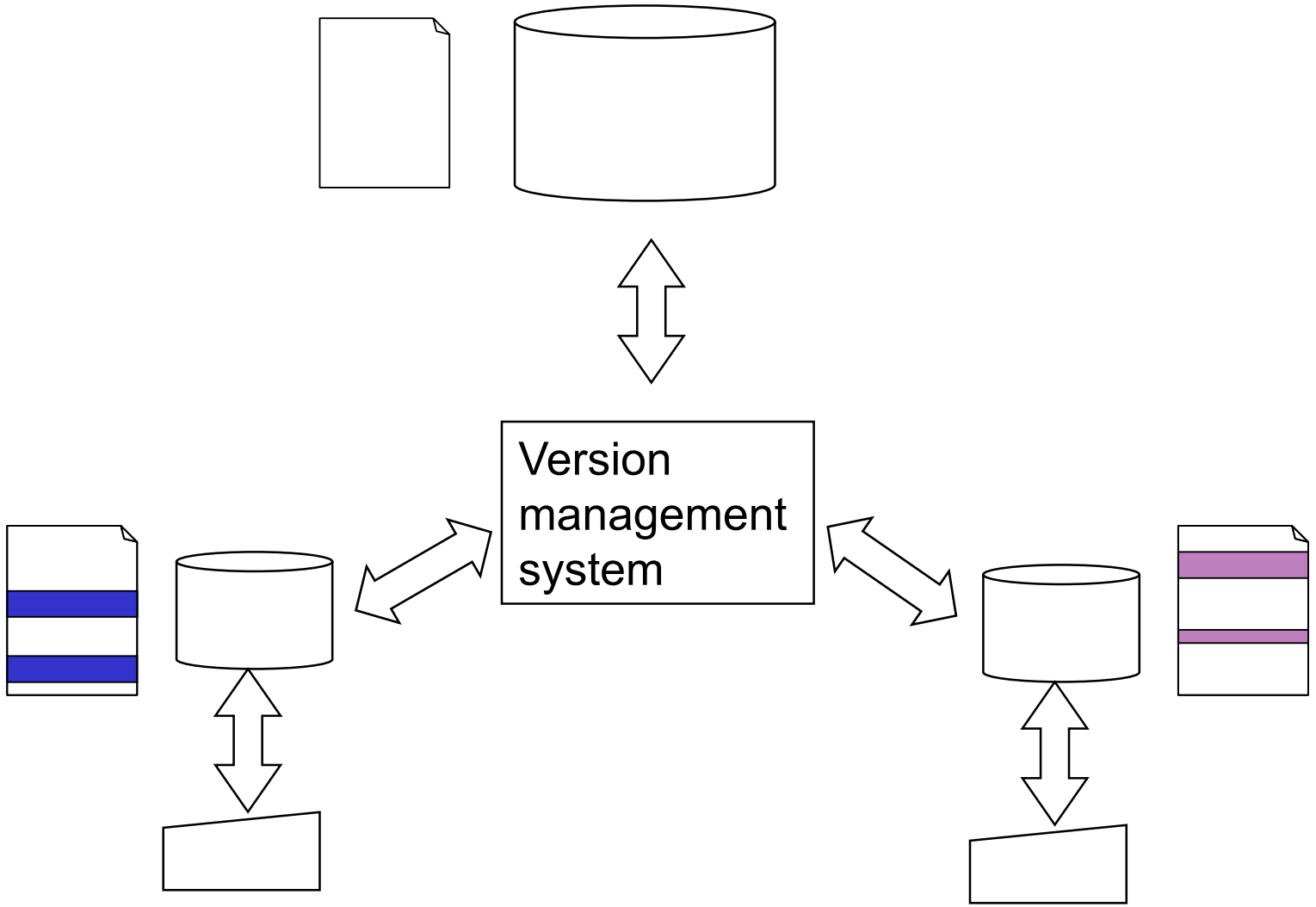
DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

Note: On the following slides, the command line commands are stated! In Eclipse (or other IDEs), this (and much more) can be done by the "Team menu"



Typical scenario

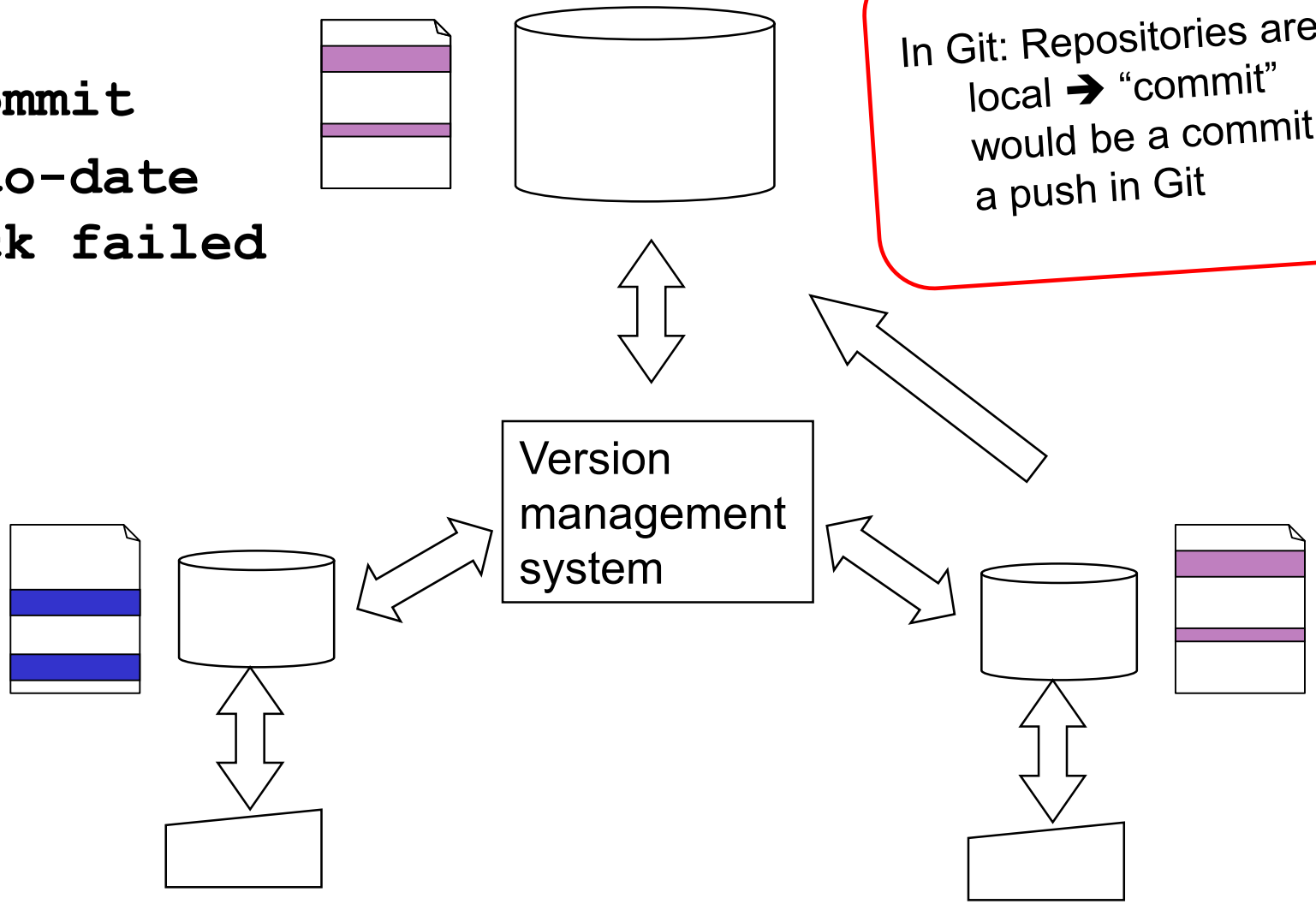
DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler



Typical scenario

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

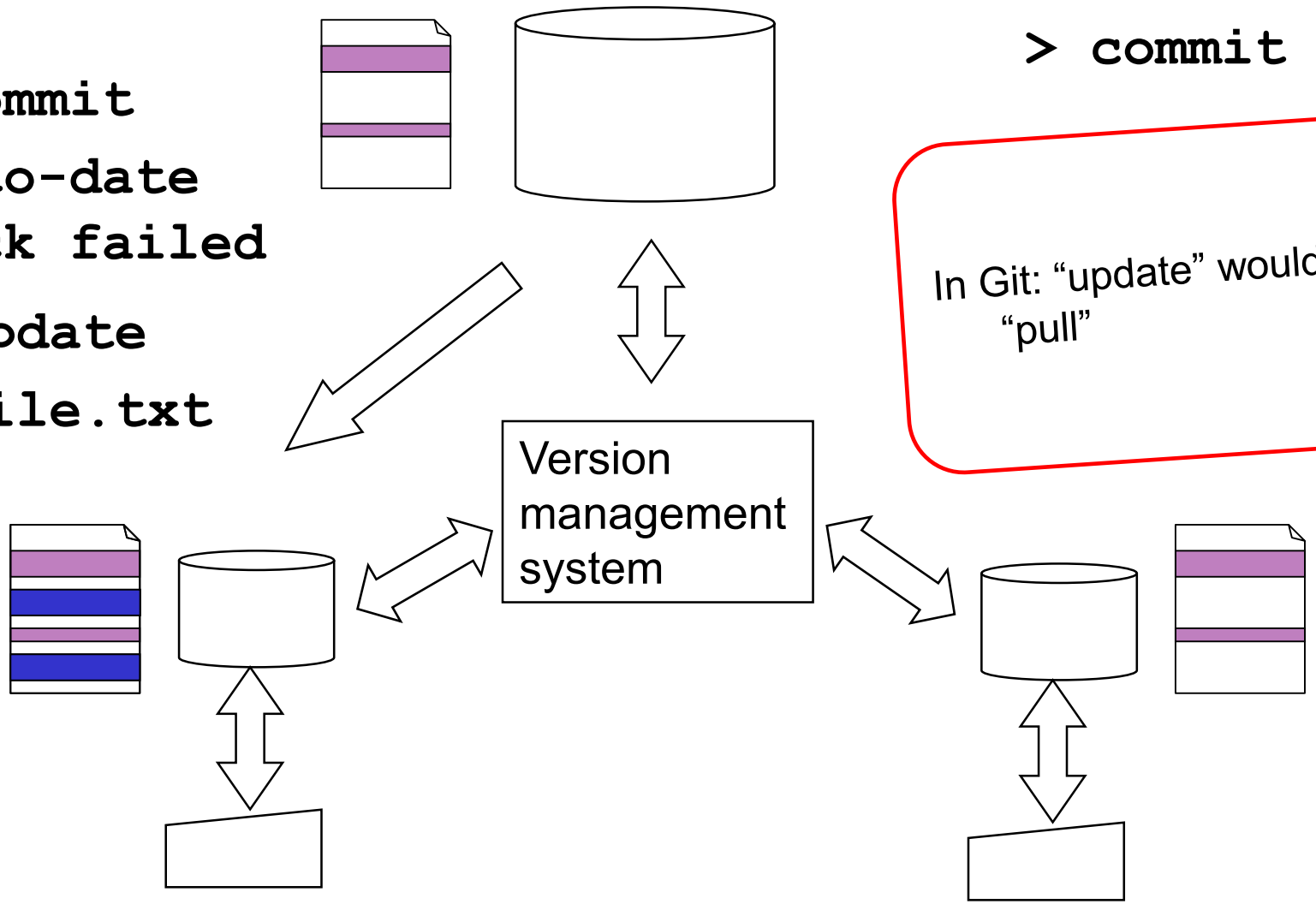
```
> commit
up-to-date
check failed
```



Typical scenario

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

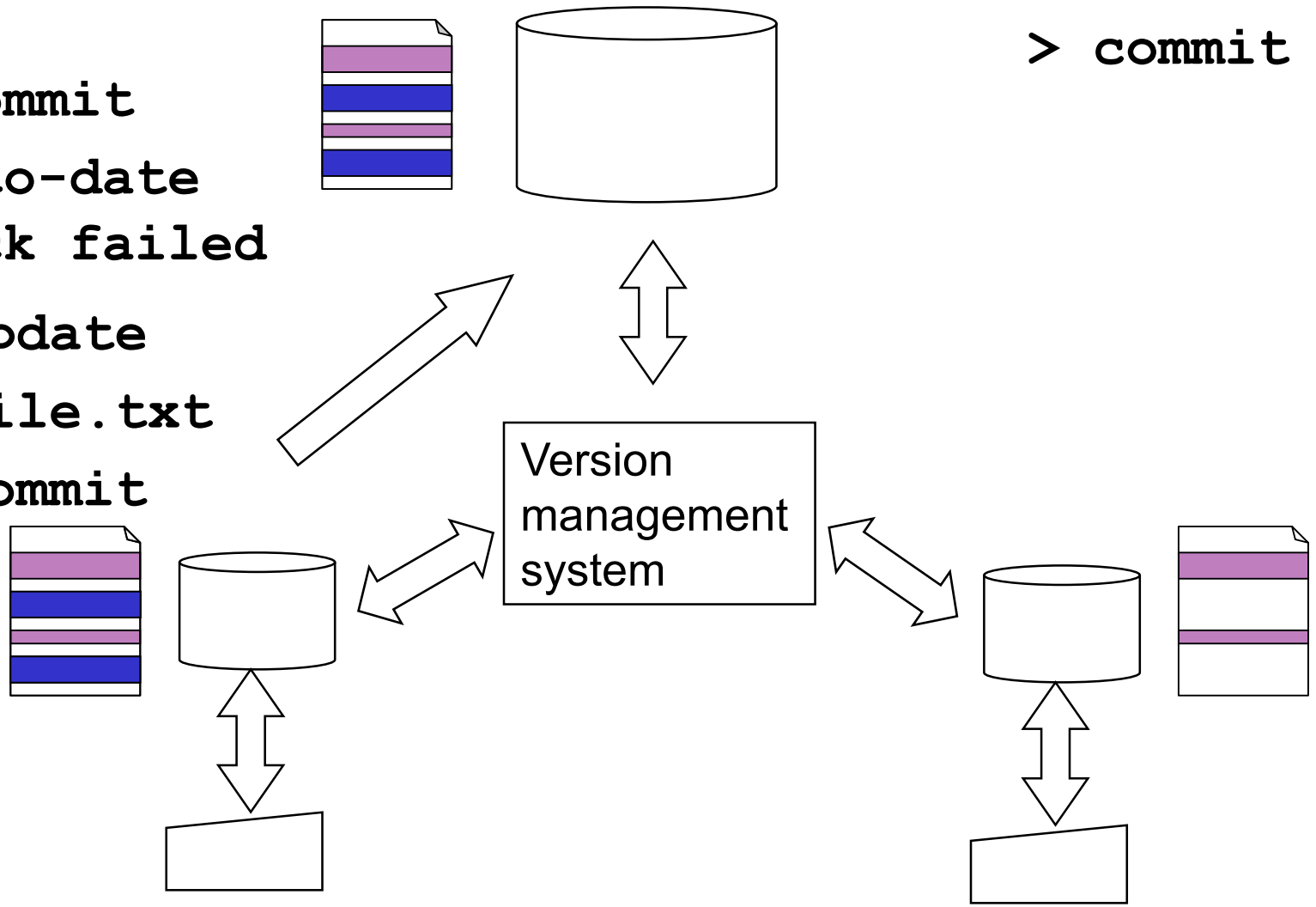
```
> commit
up-to-date
check failed
> update
M file.txt
```



Typical scenario

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

```
> commit
up-to-date
check failed
> update
M file.txt
> commit
```



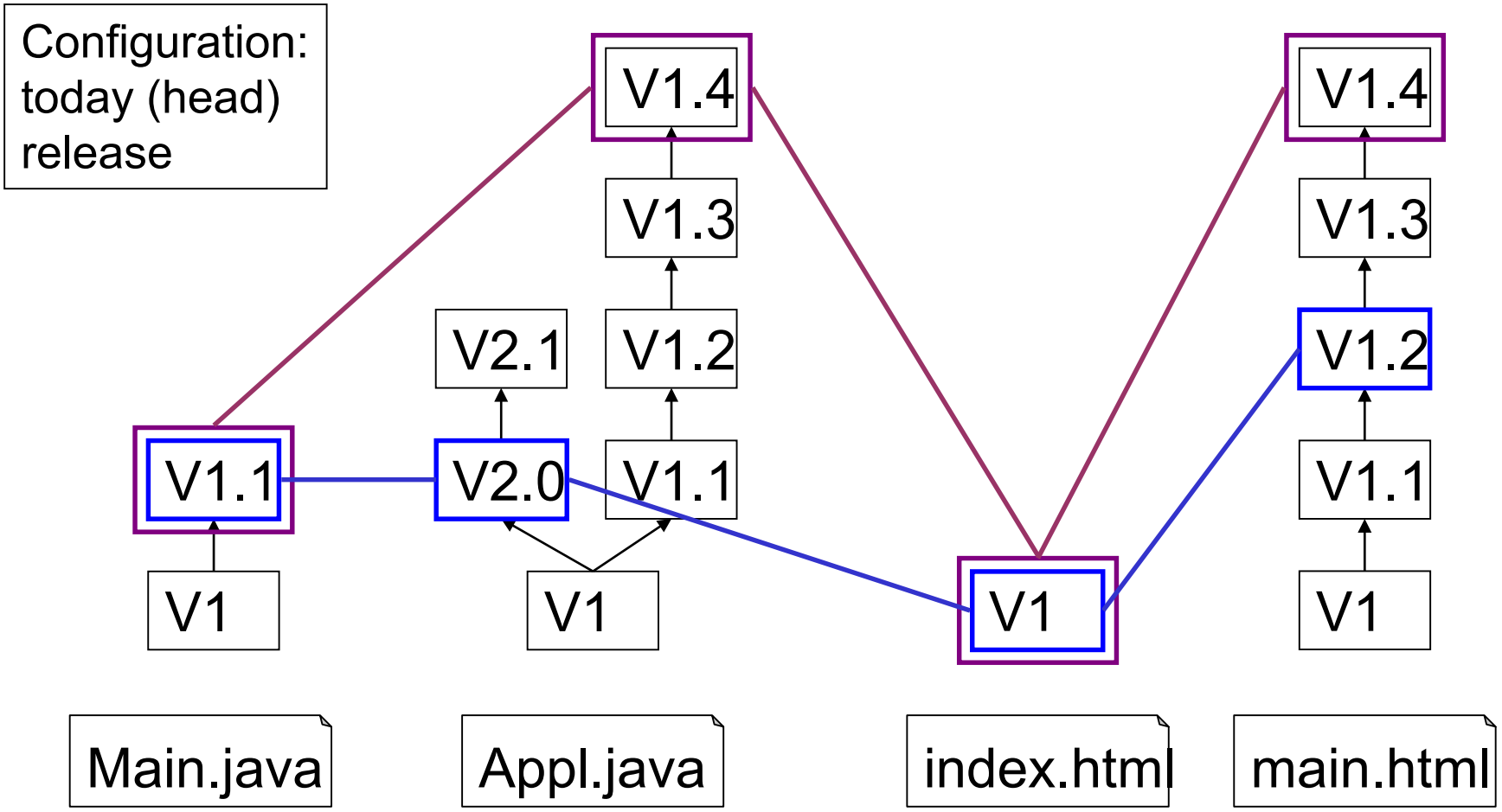
More Concepts

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

- every `commit` automatically creates a new version of the file with a new version number
- we can retrieve earlier versions of a file and go back to earlier versions
- we can compare different versions of a file
- we can define different configurations (branches) of the same project
- we can define releases (a set of related versions of files)

Configurations and releases

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler



More Concepts

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

- every `commit` can (**and should**) be accompanied by a brief comment what changes were made (and why)
- Users can automatically be notified by changes
- Change history shows who made which changes at which time (for a single file or a complete project).

More Concepts

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

- you can “tag” some versions (in SVN this is done by “SVN copying” a file to a specific “tags” directory
- …

In Git: there is an explicit notion of tag (“immutable branch”); all submissions of code should be explicitly tagged (e.g. “Final submission”).

Guideline

Which documents should be in a repository?

All
documents and files that belong to the software and the development process

This includes, build and configuration scripts (but bew of checking in passwords).

which cannot be automatically reproduced from other files or documents (by **all** other users)

Remember: Software is more than the code. Also documents should be in the repository!

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

- A **decentralized** version of a repository
- Concepts:
  - **Commit objects**:
    a set of files with changes with respect to **parent** commit objects
  - **References** to commit objects: in particular HEAD (symbolic reference to head of current branch)
- **Local repository**:
  - Commit, checkout, merge, branch
- **Remote repositories**:
  - Remote repository reference (typically: origin)
  - Clone, tracking, fetch, pull (= fetch+merge), push

Commit is in local repository only!

Git

- Git stores versions (**commit objects**) in snapshots of the current files (as opposed to deltas)
- Each **commit** (object) has its checksum, which is used for identifying it
- One commit can have one or more **parents** (merge)
- Commits can be **referenced** (in particular HEAD)
- Explicit notion of **tags** (lightweight and annotated): basically tags are branches that cannot change

- Files in working directory need to be staged (added to index) for the commit

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

- **Branch:** create a new branch in a repository; e.g. for some experiments, which should not go to the master branch

- **Merge**: Merge all changes made of two branches (based on the common ancestor).

  - Merge might be fast-forward (easy case)
  - Merge might require manual intervention (in case of conflicts)

DTU Compute
Department of Applied
Ekkart Kindler

- **Cherry Pick** one or several commits: Apply all changes (!) of the picked commits to a branch

Use with care

- Useful for applying selected changes (e.g. from a special feature branch) to the master branch
- But, this requires that all changes of a commit "belong" to a desried change/feature

- **Rebase**: Can be considered as a special form of cherry-picking

Pull request / Merge request

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

- Makes changes (from some branch/repository) available, and asks some people for reviewing, changing, extending, and then merging these changes (into some other branch/repository)

- The other branch is often protected (typically the master branch)
- GitHub and Bitbucket call this a ”pull request”; GitLab calls it ”merge request” ☹
- Don’t confuse this with a ”git pull” or a ”git merge” command!

Git: Links

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

Git Concepts

- https://www.sbf5.com/~cduan/technical/git/
- https://git-scm.com/book/en/v2

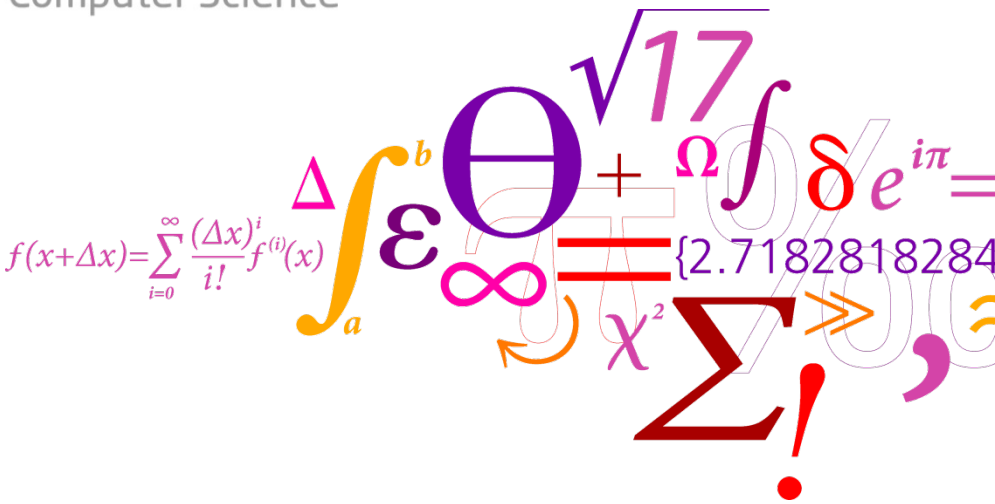
Practical use:

- GitHub: https://guides.github.com/activities/hello-world
- eGit: https://wiki.eclipse.org/EGit/User_Guide
- Workflow: https://www.atlassian.com/git/tutorials/comparing-workflows

IV. Working together

IV.5. Collaborative Development Environments (CDE)

DTU Compute
Department of Applied Mathematics and Computer Science



CDEs: Idea

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

Streamline communication among different stakeholders in a development process – often web interface on top of a version management system

Example: Redmine, gforge, …

Concepts

- Users
- Roles (responsibilities, skills)
- Tasks (development, bug fix, …)
- Task tracking (live cycle of a task)
- Communication (smoothly integrated)
  - notifications
  - more or less structured discussions (wiki, news, email, …)

Note: This year, we do not use an explicit CDE (Redmine). But, GitHub, GitLab, … provide some basic CDE functionality and Jenkins for CI/CD.