

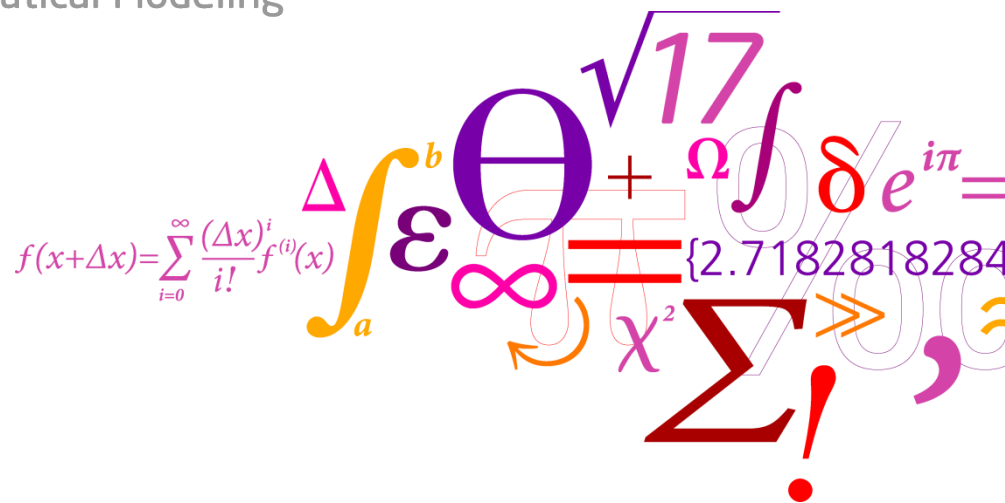
Software Engineering 2

A practical course in software engineering

Ekkart Kindler

DTU Informatics

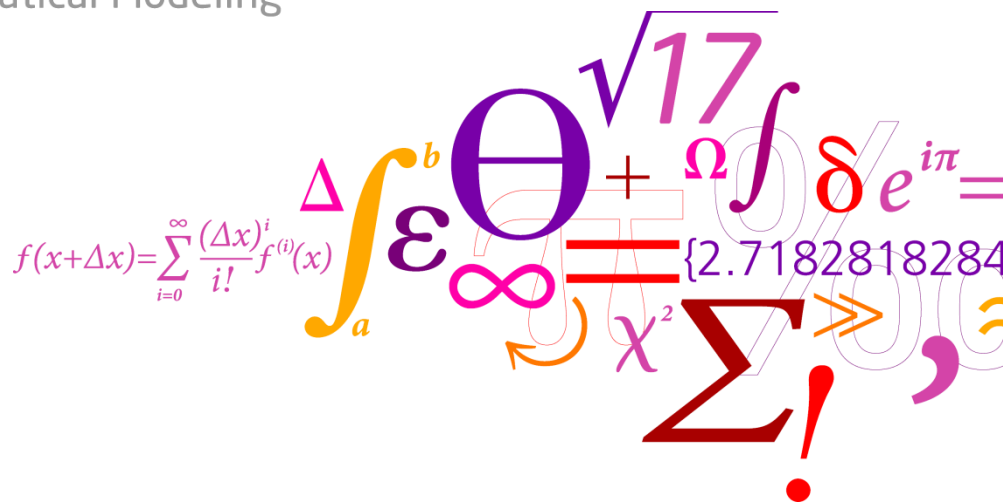
Department of Informatics and Mathematical Modeling



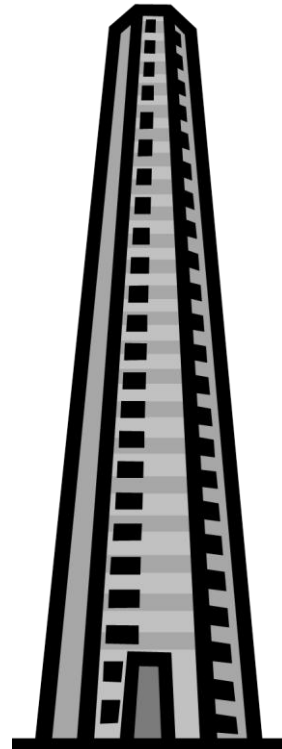
IV. Working Together

DTU Informatics

Department of Informatics and Mathematical Modeling



- 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.

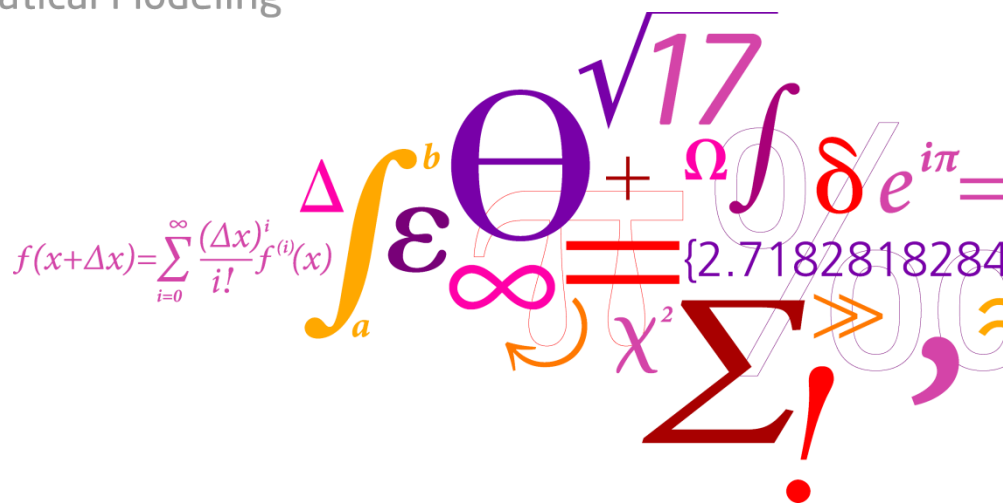


IV. Working together

IV.1. Management

DTU Informatics

Department of Informatics and Mathematical Modeling



(Software) Management

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

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!

Software Management requires

- short-term
(within a project or even within a phase)
and
- long-term
(spanning more projects)

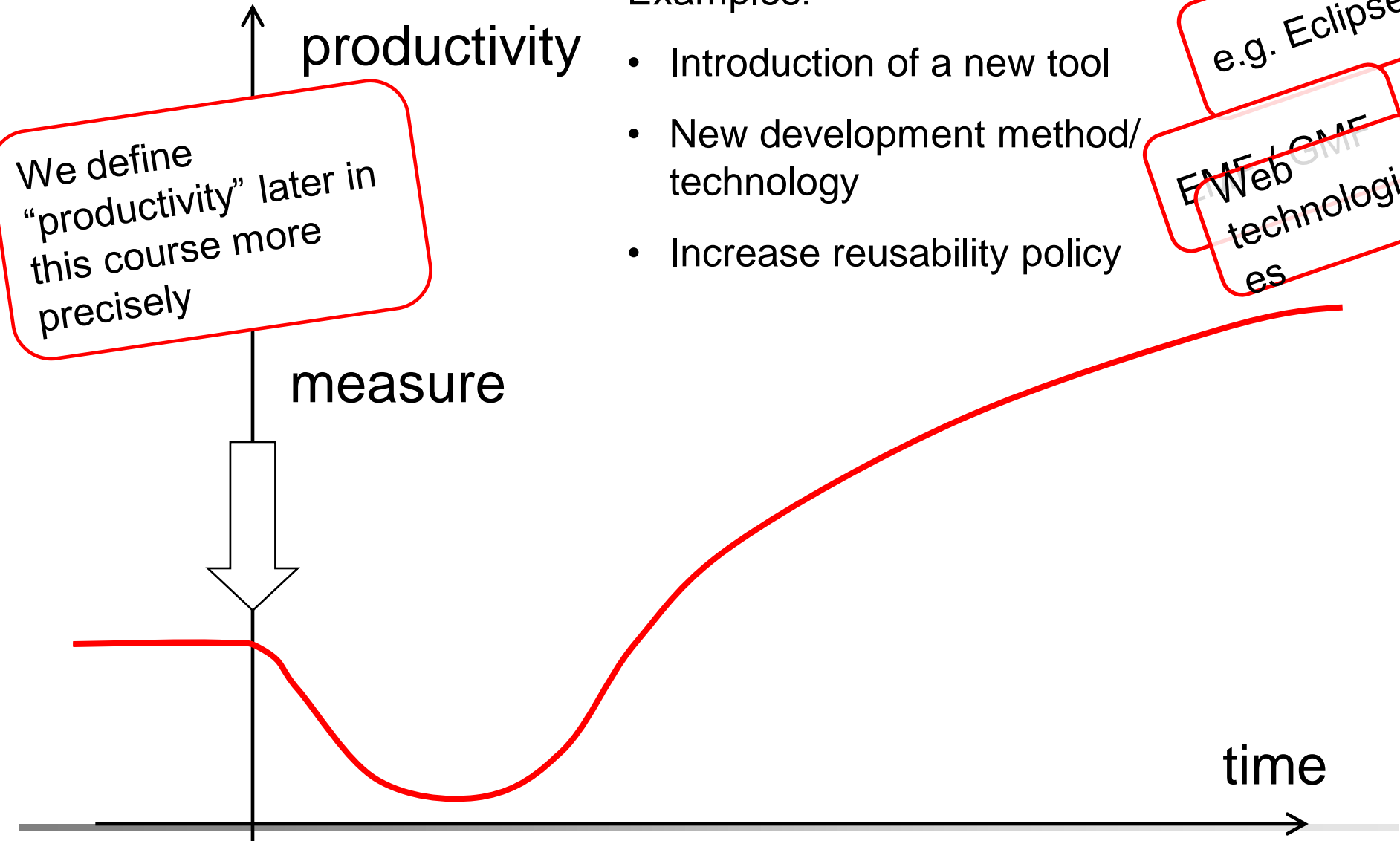
measures

Examples:

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

e.g. Eclipse

EMF / GME
Web
technologies



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)

■ Planning

- set goal
- set dates
- define course of action
- assign resources
- ...

■ Organization

- assign tasks
- define organisation structures
- assign responsibilities
- ...

■ **Leading**

- lead and motivate team members
- improve communication
- solve conflicts
- ...

■ **Monitoring and controlling**

- check progress
- identify problems (early)
- produce relief
- ...

- Planning
- Organization
- Leading
- Monitoring & controlling

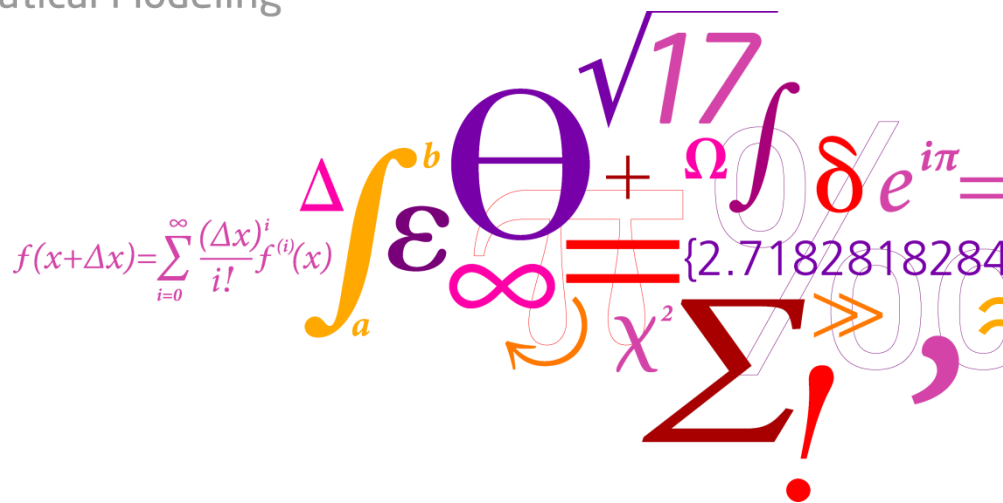


IV. Working together

IV.2. Coordination

DTU Informatics

Department of Informatics and Mathematical Modeling



- Mutual agreement
- Instruction

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

- Socialisation

- 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**.

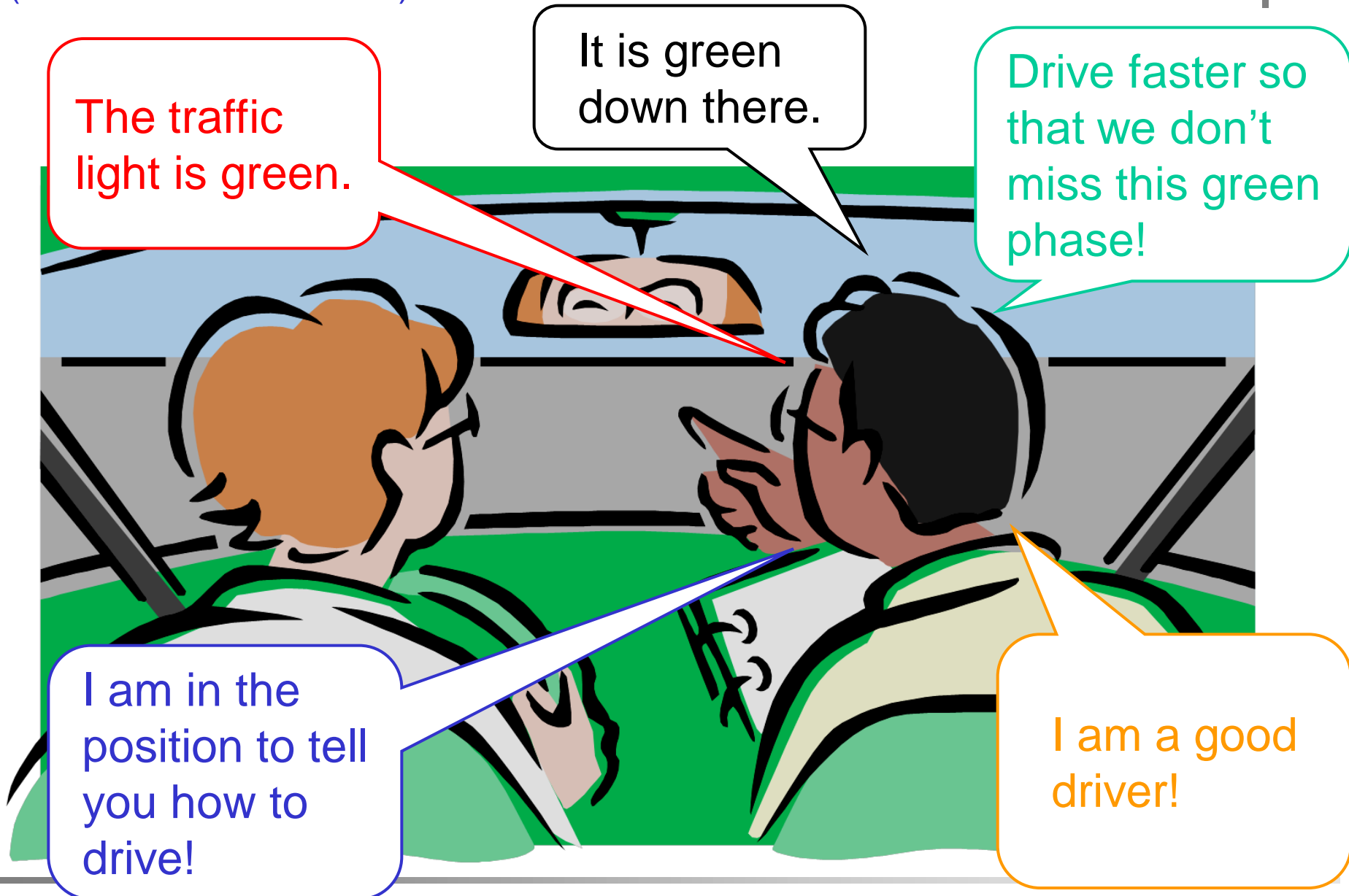
Put agenda and minutes
for every meeting to the
repository (in particular
group meetings)

Coordination and leadership require

communication

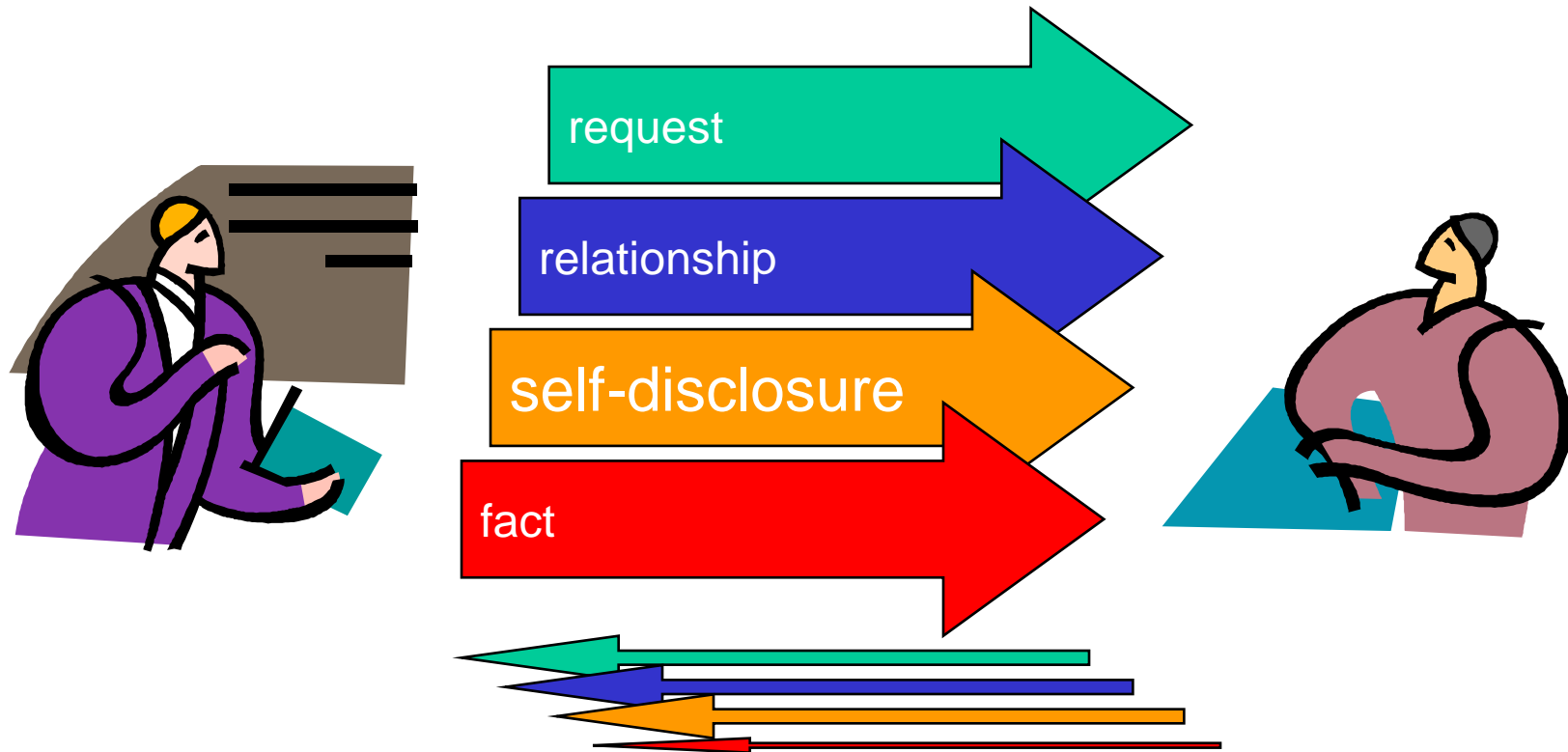
Four sides of a statement

(after Schulz von Thun)



Four sides of a statement

(after Schulz von Thun)



- 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)!

- 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

- 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

- 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

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

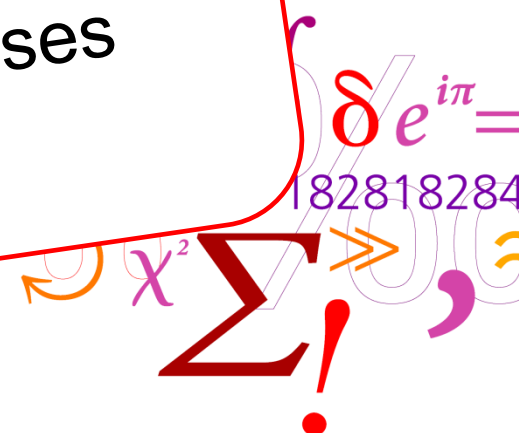
Made these rules
his/her rules and
does not apply them
schematically.

IV. Working together

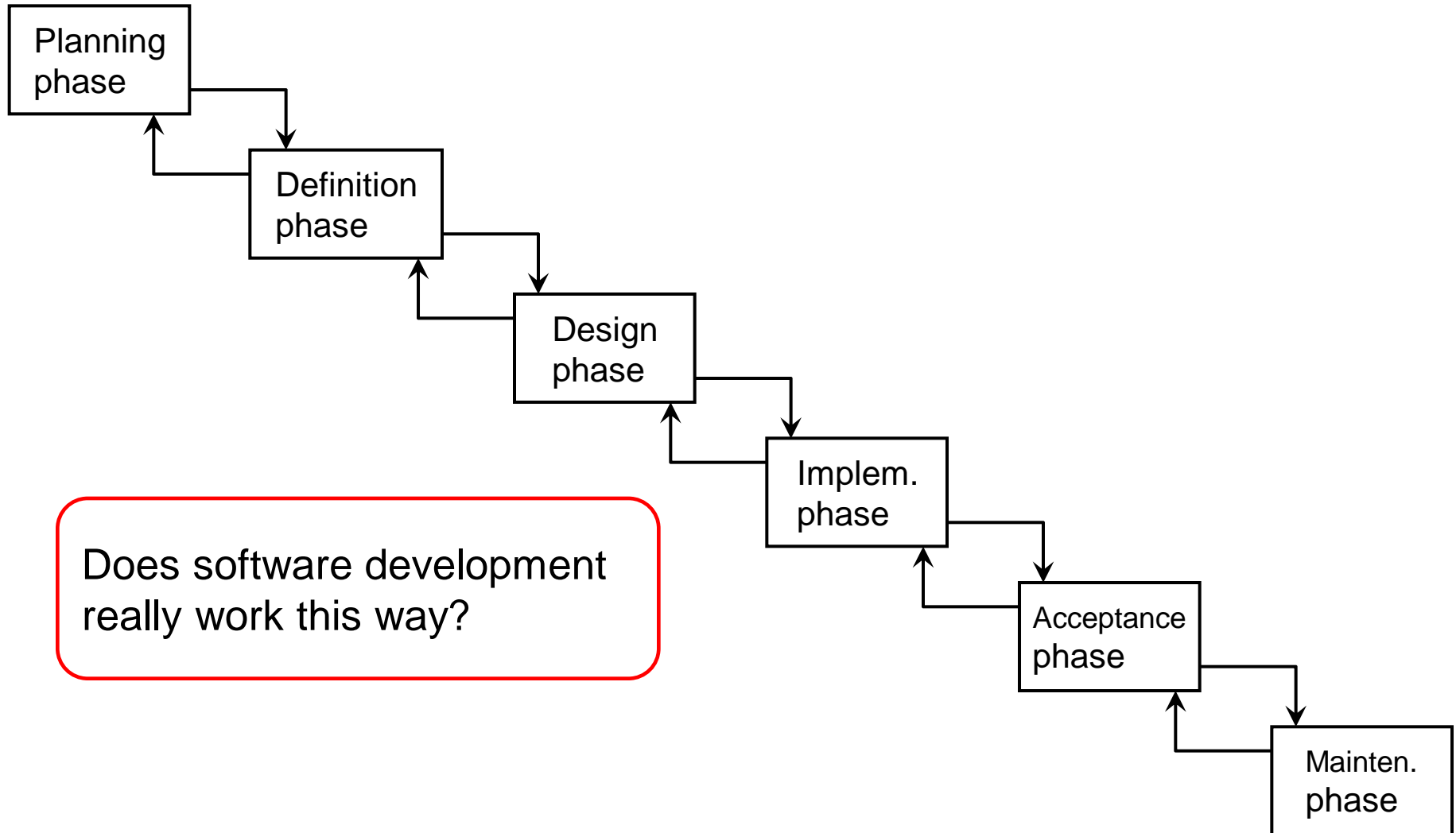
IV.3. Process models

DTU Informatics
Department of In

Coordination mechanism:
Standardization of processes

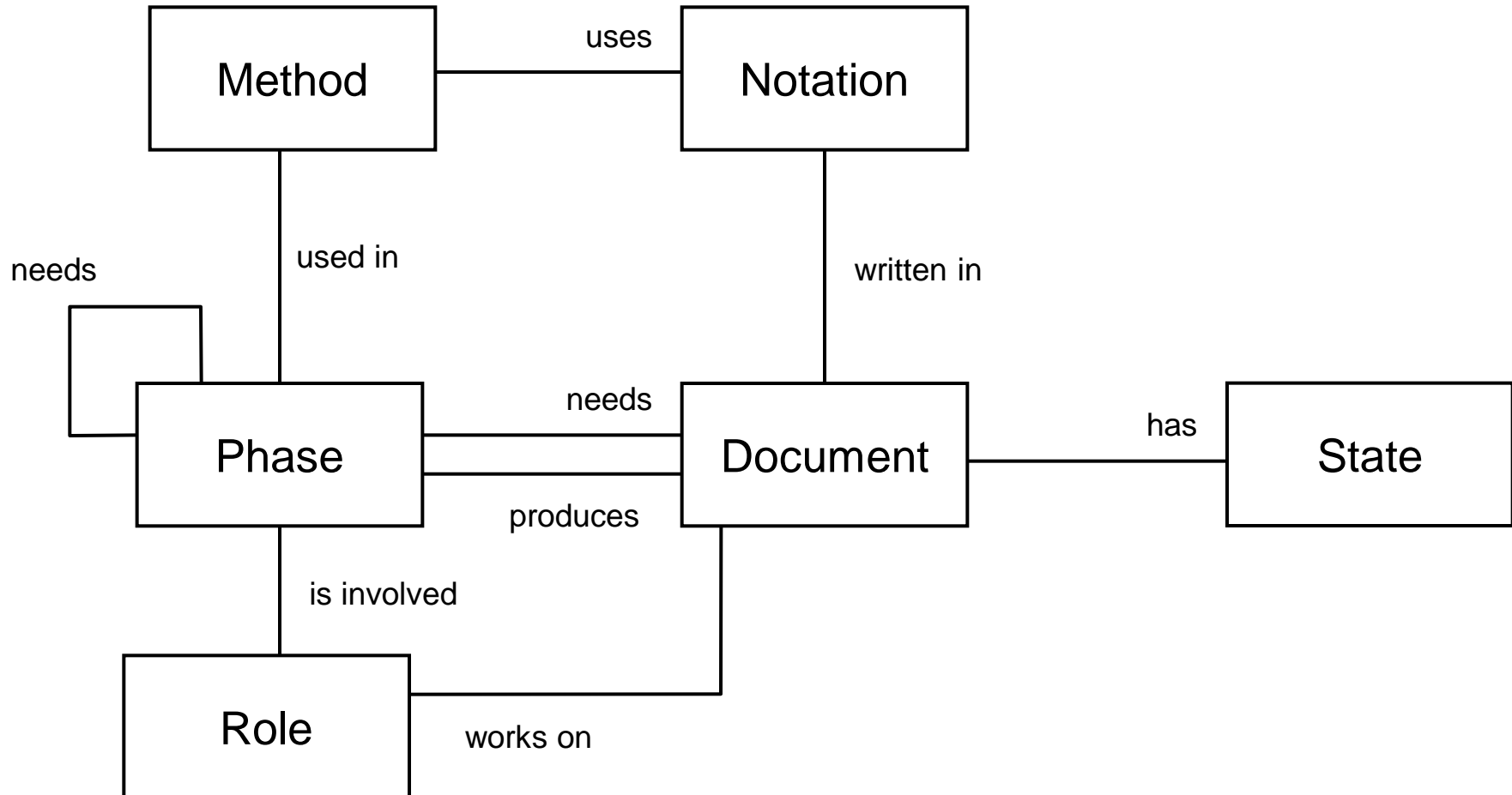


Waterfall Model



- **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!



- 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

- There are many different notations for such plans

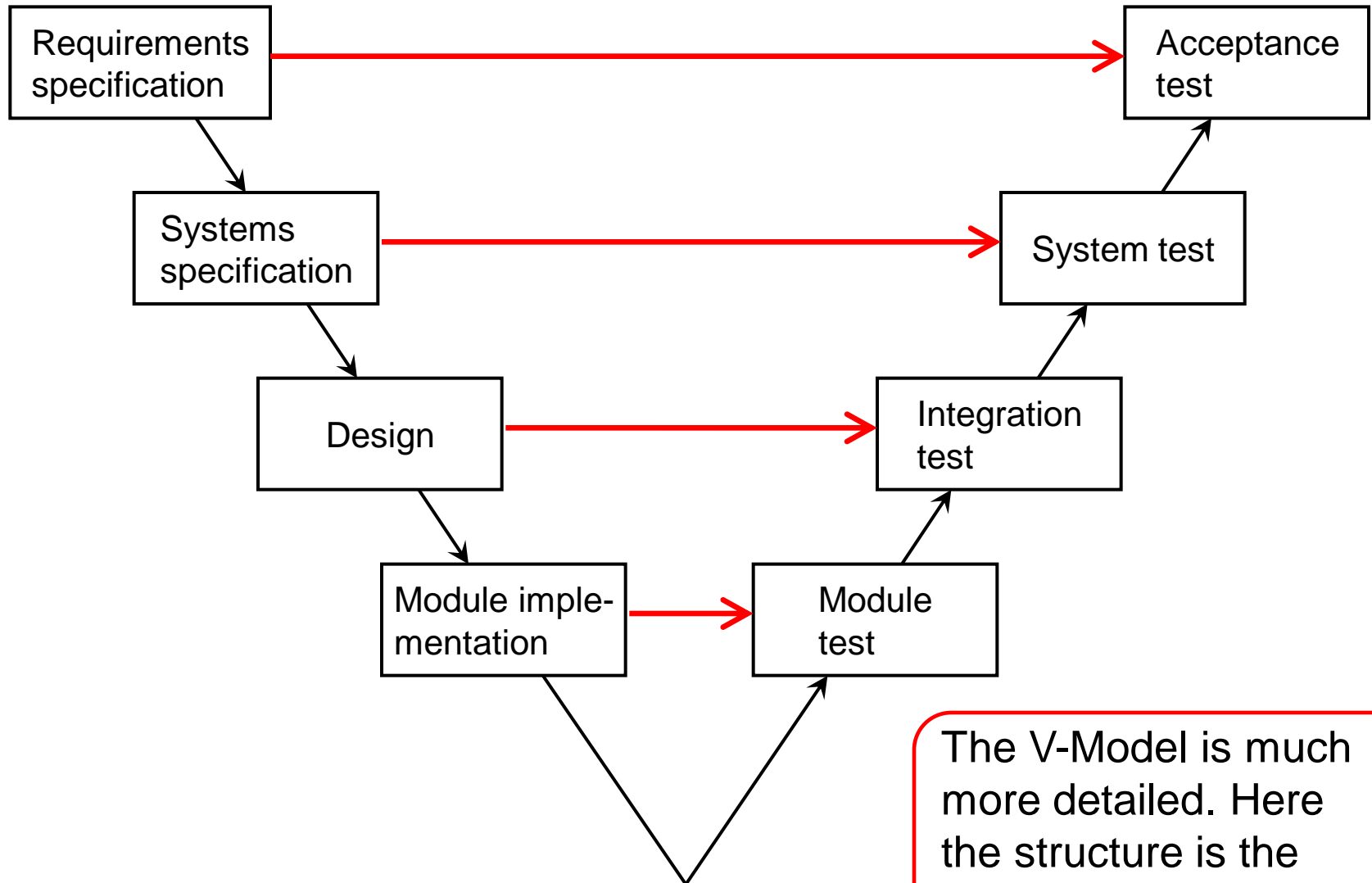
- Gantt diagrams
- Many tools (MS Project etc.)

- It is easy to analyse such plans:
 - timing
 - critical paths (buffer)
 - load on team and every team member
 - planning costs
 - controlling
 - ...

- in order to monitor the progress, the project plan defines **milestones**
- a milestone is a set of documents 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

- 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)
- Regular
 - The milestones should be in regular intervals

The V-Model (rough idea)



- 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:

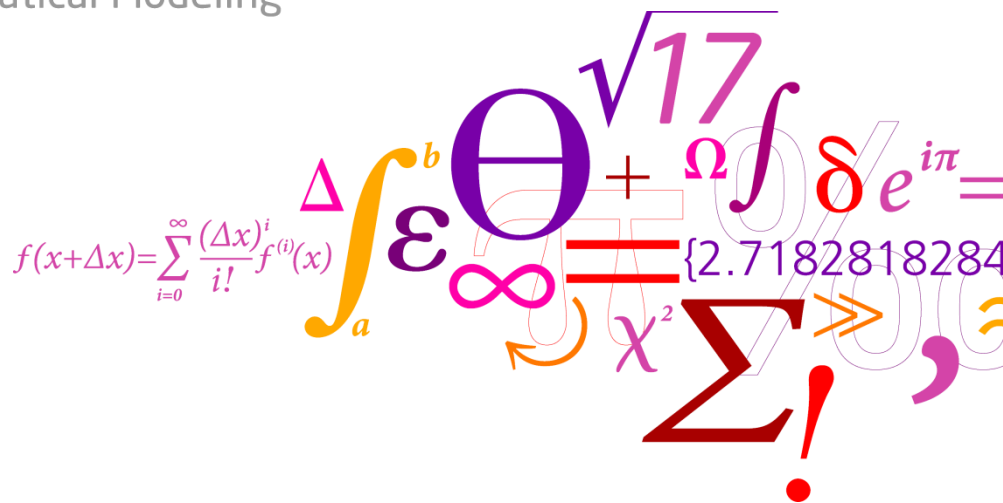
- Stepwise development of product
- Early prototypes (“executable milestones”)
- “Maintenance as part of the development”

IV. Working together

IV.4. Version Management

DTU Informatics

Department of Informatics and Mathematical Modeling



$$f(x+\Delta x) = \sum_{i=0}^{\infty} \frac{(\Delta x)^i}{i!} f^{(i)}(x)$$

$\int_a^b \varepsilon \Theta + \Omega \int \delta e^{i\pi} = \{2.7182818284\}$

∞ , χ^2 , Σ , \gg , $!$

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.

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)

Result: Typical problems / questions:

- Where is the up to date version?
- Who has the last working version?
- Where is the version from August 10, 2007?
- Who has the version that we presented recently at the meeting with our customer?

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!
- ...

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

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”

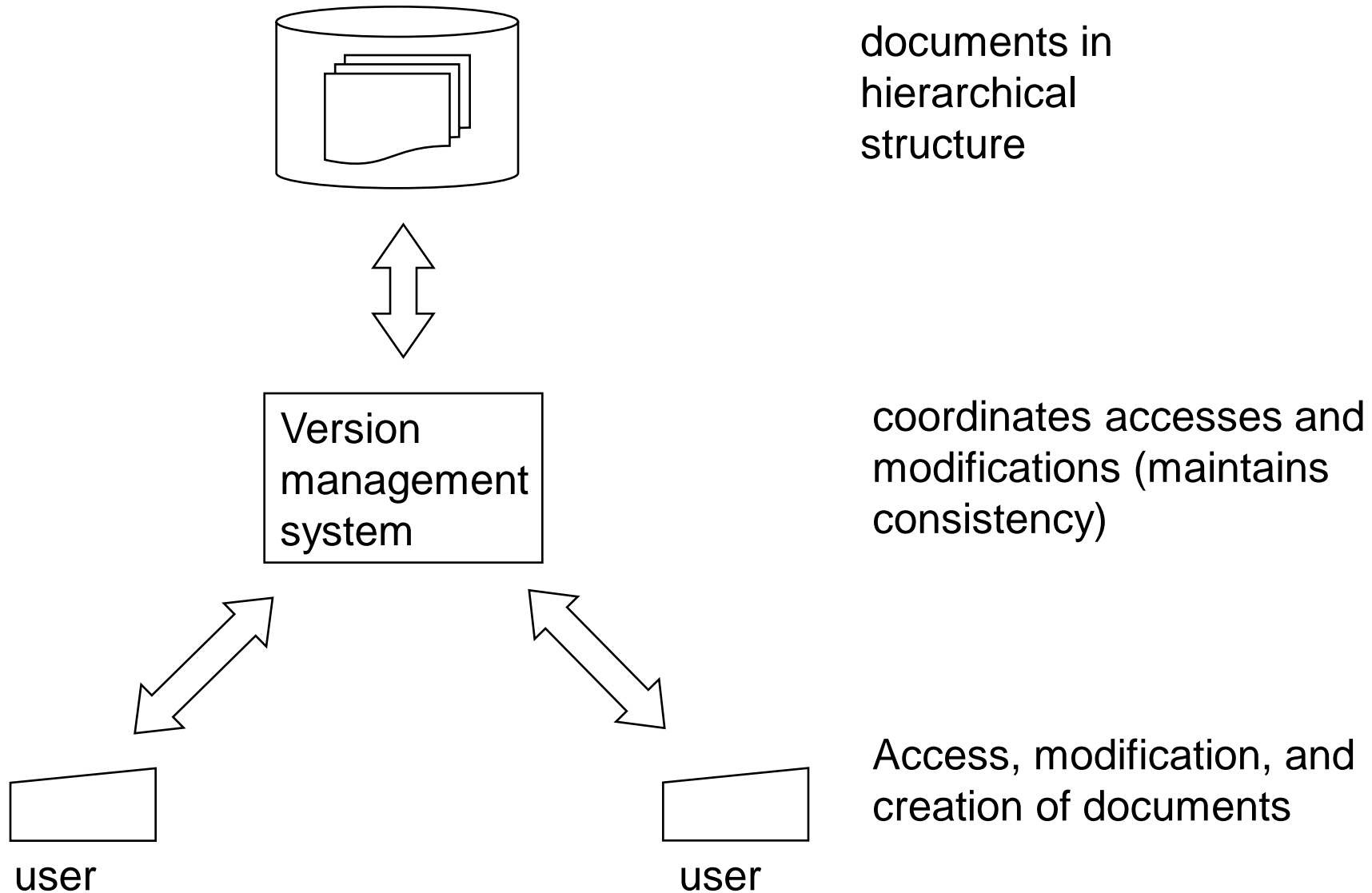
Conventions need to be enforced!

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).



Pessimistic mechanisms:

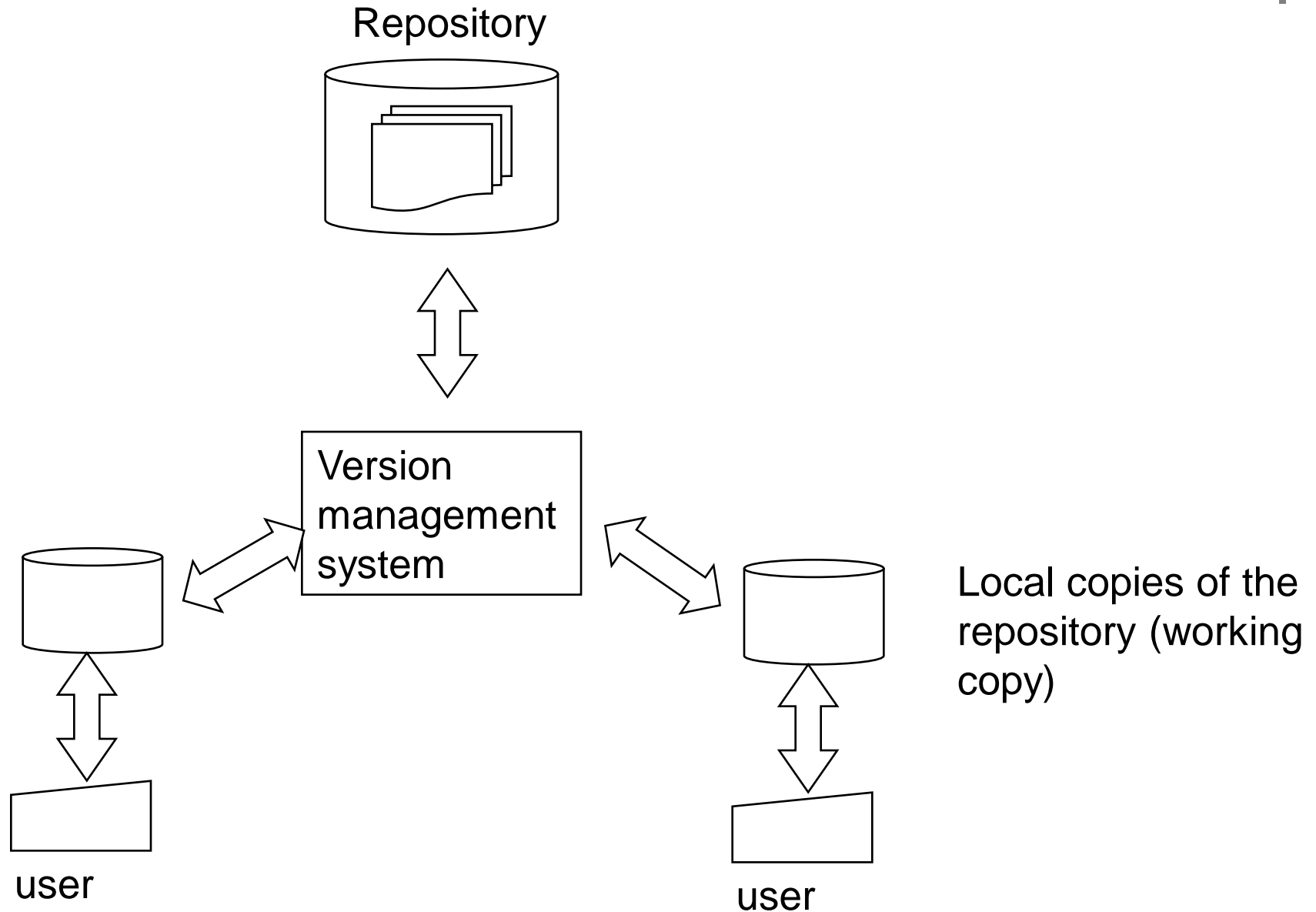
A document can never be accessed and changed by two different people at the same time (→ locking / checkout / checkin).

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



- 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)

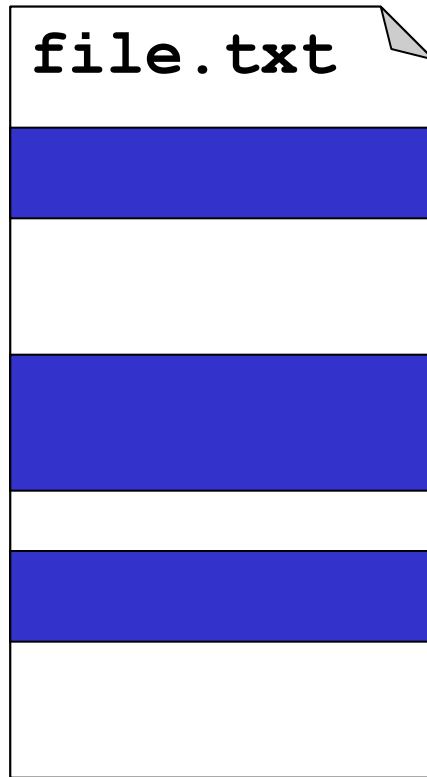
These exist as command line commands as well as in the Eclipse GUI (Team).

- 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).

- 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

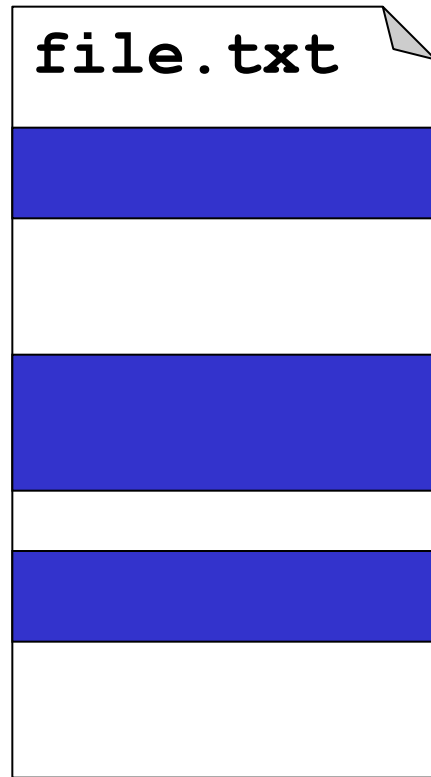


Changes in repository
(by a commit of a different user)



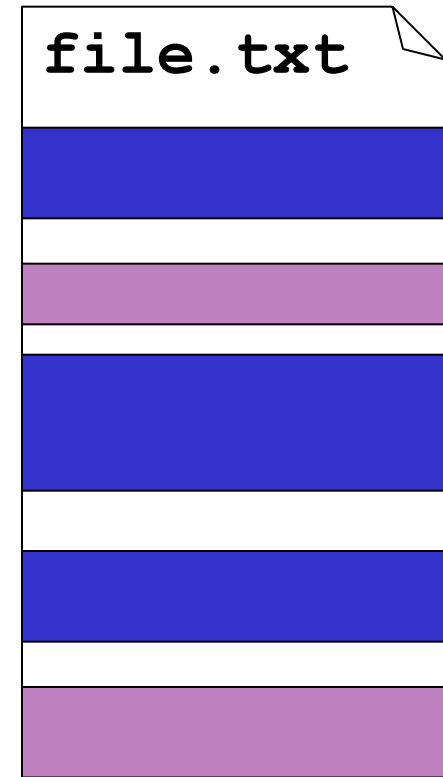
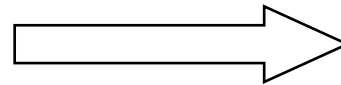
Changes in working copy

Merge: Scenario 1



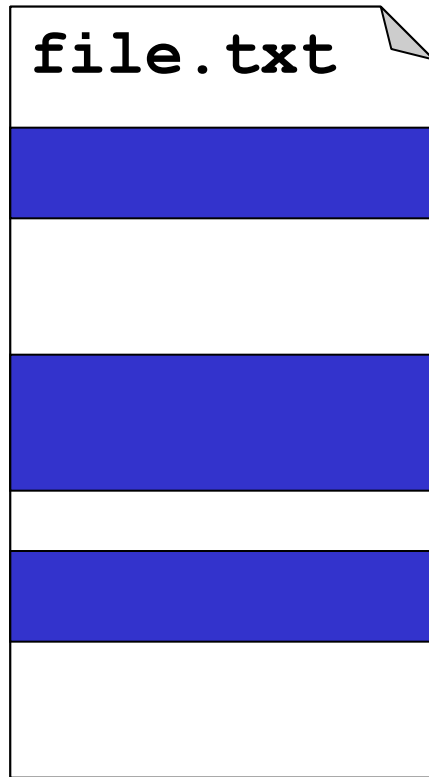
Changes in repository

update

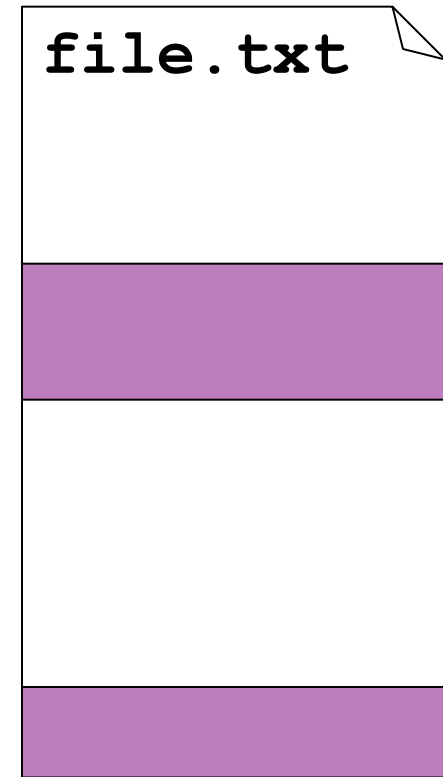


Updated working copy

Merge: Scenario 2

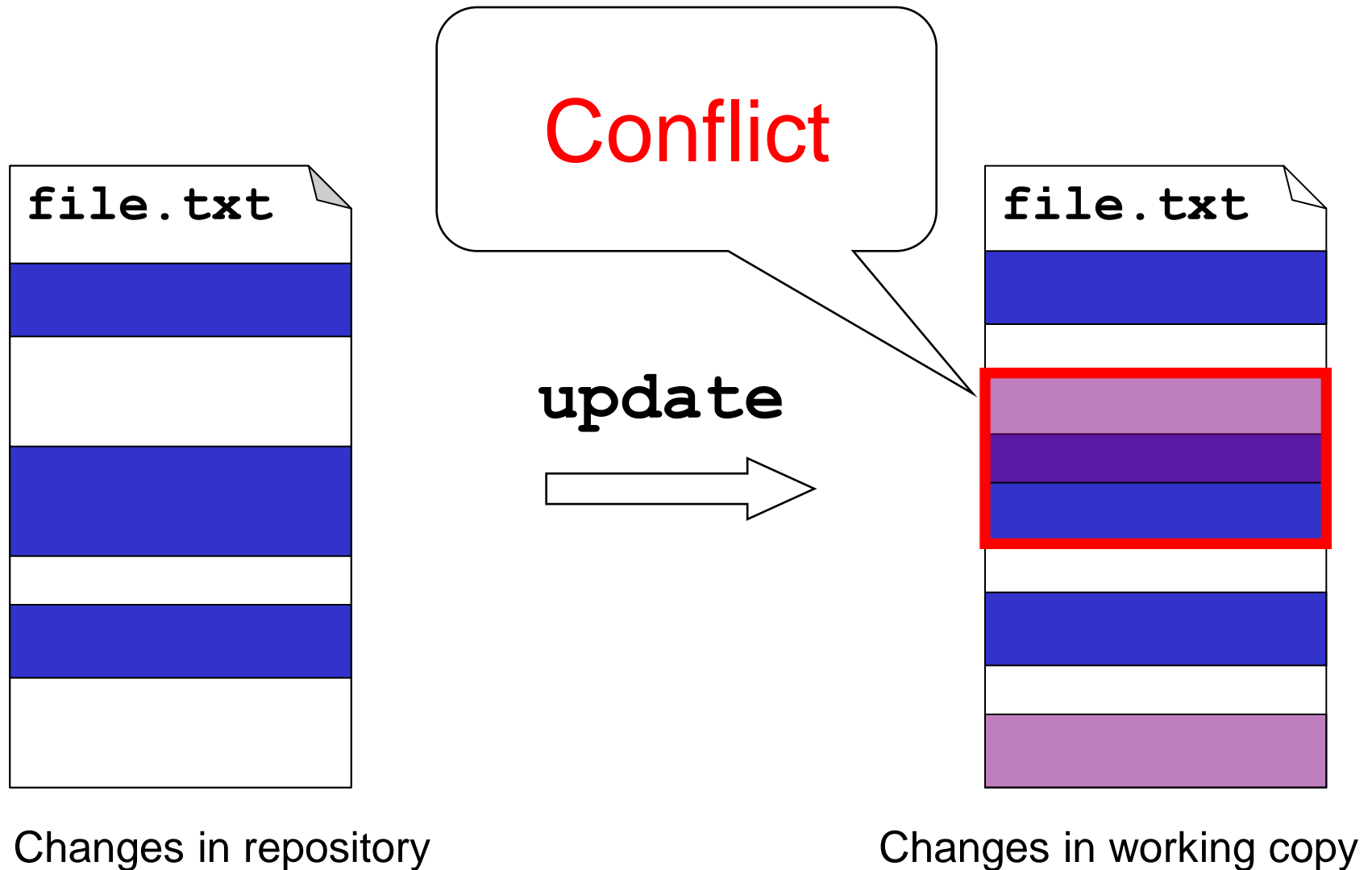


Changes in repository



Changes in working copy

Merge: Scenario 2



- **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.

- 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 makes some good guesses!

- MS Word documents are binary for most version management systems

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

→ **Impossible!!**

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

- By this strategy conflicts only occur in working copies!
- There is always a user responsible for resolving it.

Pessimistic approach:

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

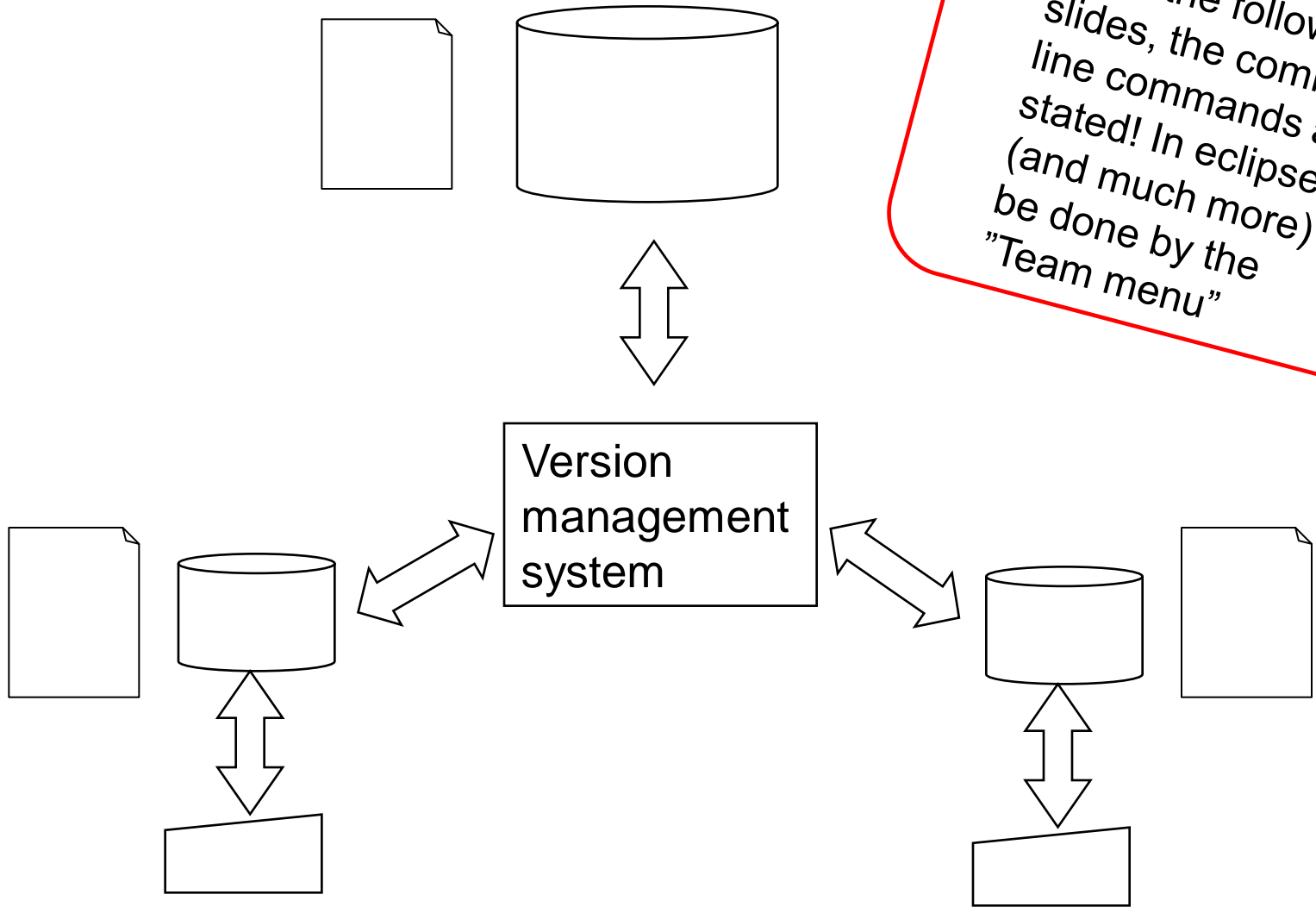
Optimistic Approach:

- conflicts (fortunately very rare)
- ++ concurrent work possible
(the last one needs to clean up the mess)

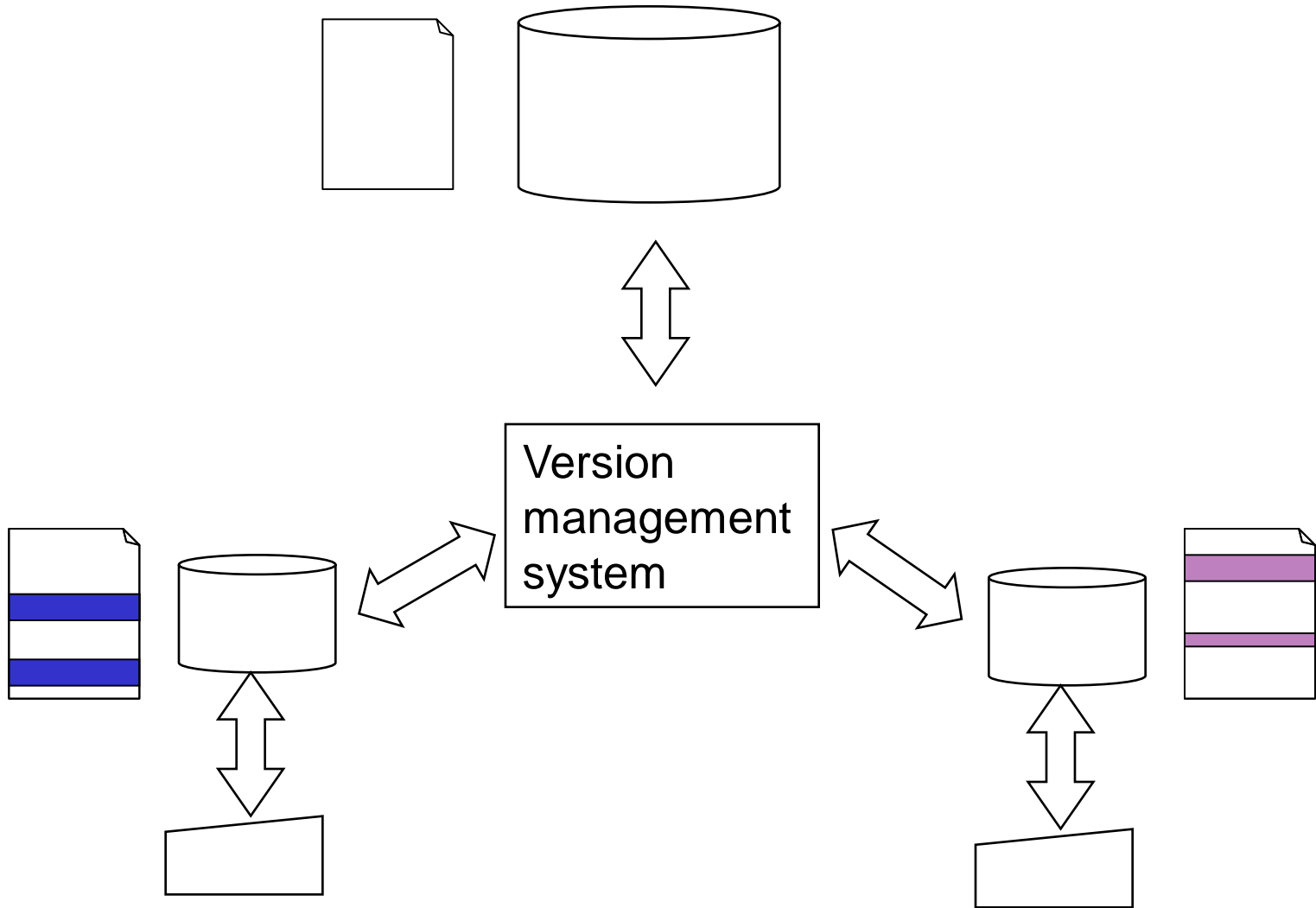
For typical software projects (big teams working concurrently), optimistic consistency mechanisms have turned out to be more appropriate:

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

Typical scenario



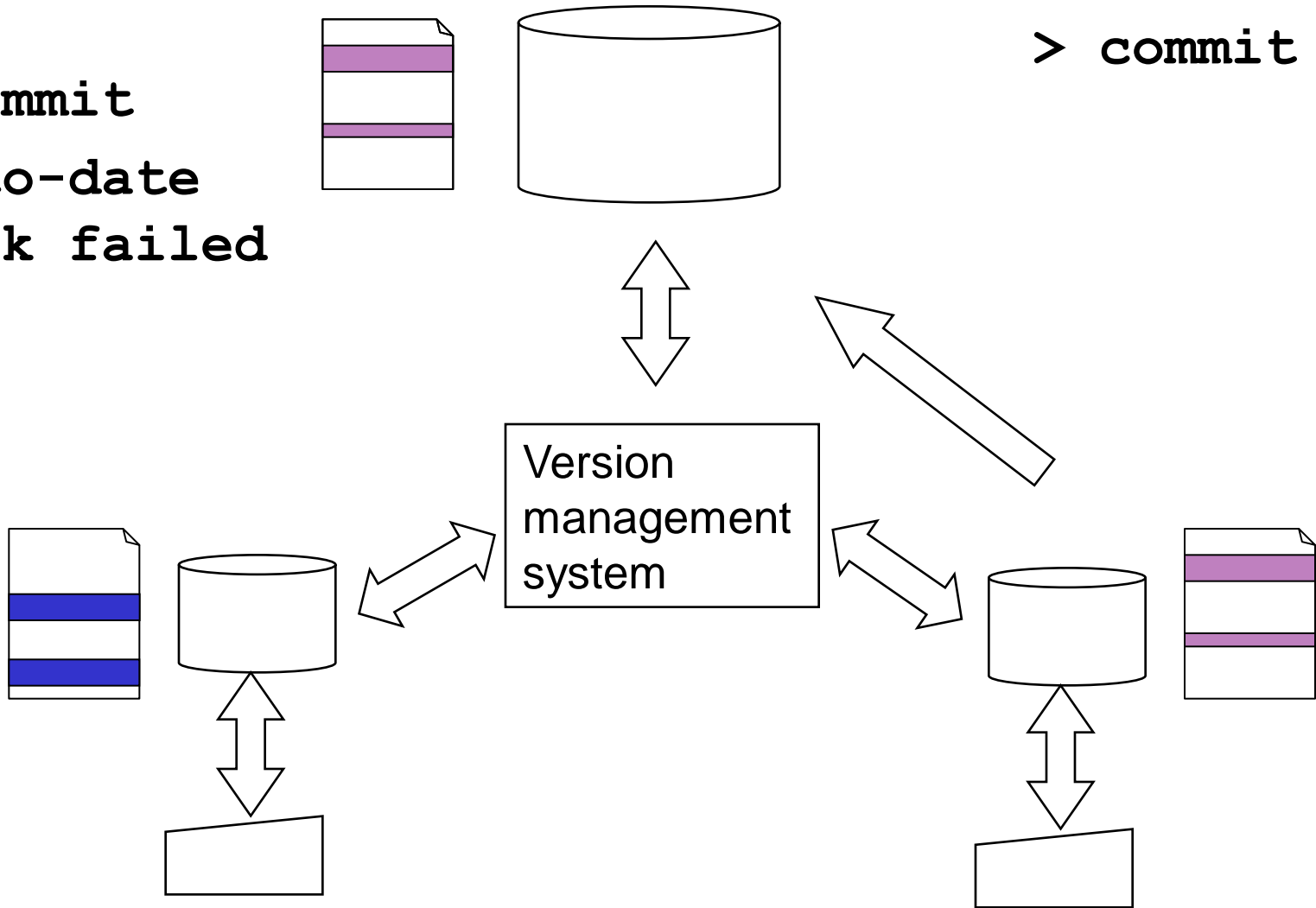
Note: On the following slides, the command line commands are stated! In eclipse, this (and much more) can be done by the "Team menu"



Typical scenario

> commit
up-to-date
check failed

> commit

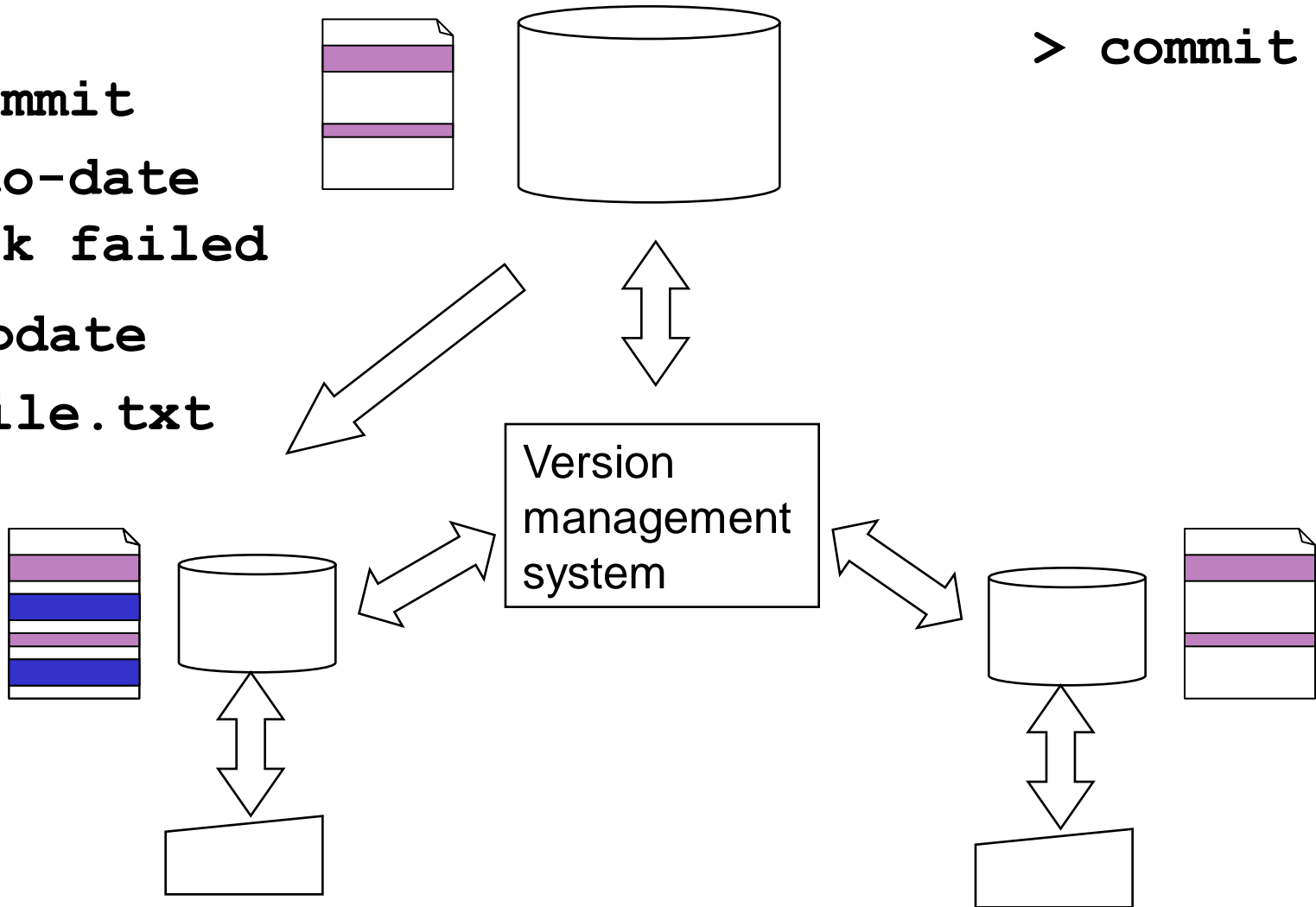


Typical scenario

> commit
up-to-date
check failed

> update
M file.txt

> commit

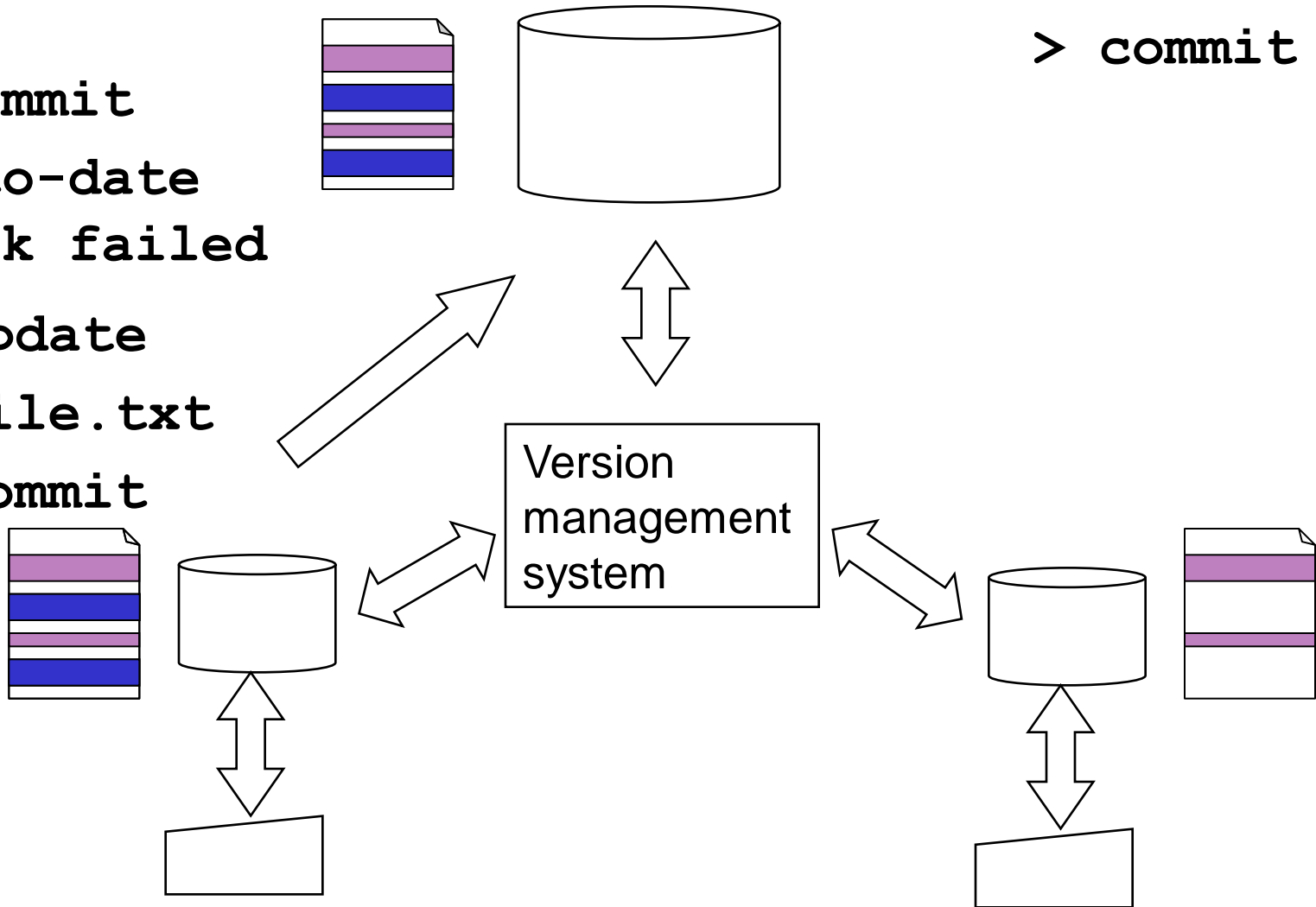


Typical scenario

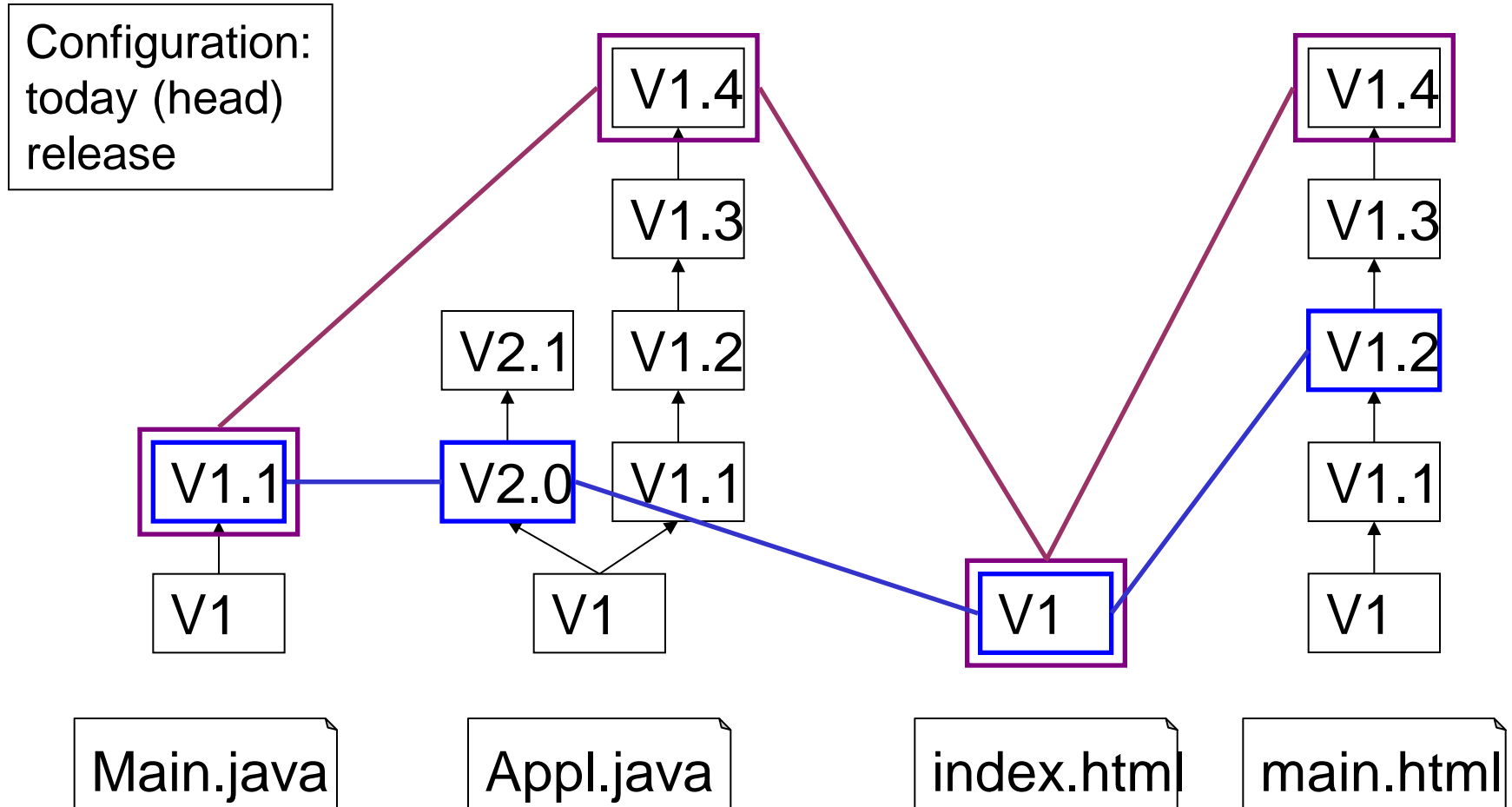
> commit
up-to-date
check failed

> update
M file.txt
> commit

> commit



- 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)



- 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).

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

Which documents should be in a repository?

All

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

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

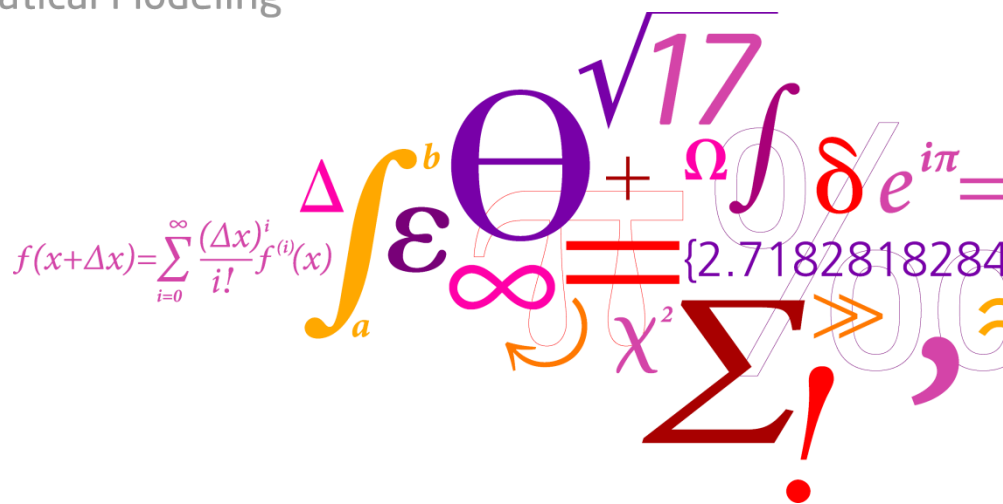
Remember: Software is more than the code. Also documents Should be in the repository!

IV. Working together

IV.5. Collaborative Development Environments (CDE)

DTU Informatics

Department of Informatics and Mathematical Modeling



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

Example: gforge, ...

- 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, ...)

You will have a Redmine server available shortly.
See <http://www.redmine.org/>