PVG (EDAF45) - lecture 3: Konfigurationshantering

Ulf Asklund/Lars Bendix

SCM
EDAN10

Department of Computer Science
Lund Institute of Technology
Sweden

© Lars Bendix - Lund Institute of Technology

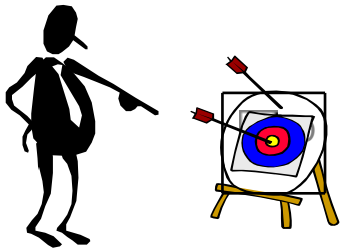
Agenda…

- CM-grunder
  - CM finns ”överallt”
  - Varför CM?
  - Olika målgrupper för CM
  - Traditionell CM
    - CM-aktiviteter
  - CM för utvecklare
    - CM för XP
    - Versionshanteringsmodeller
      - CVS
      - git

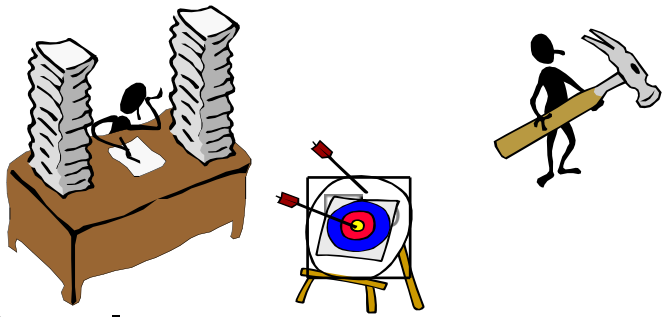
What is SCM?

Software Configuration Management:
is the discipline of organising, controlling and managing the development and evolution of software systems. (IEEE, ISO,...)

The goal is to maximize productivity by minimizing mistakes. (Babich)



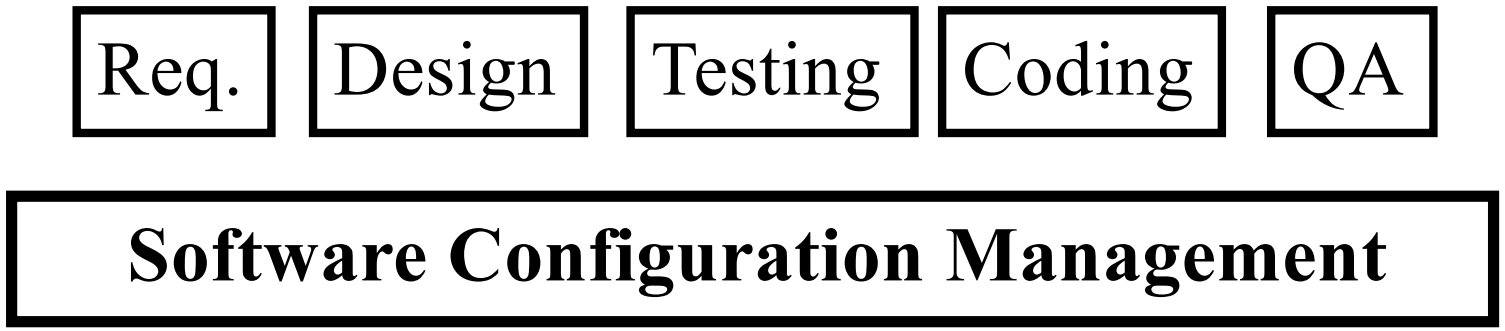
Management



Developer

Building on sand?

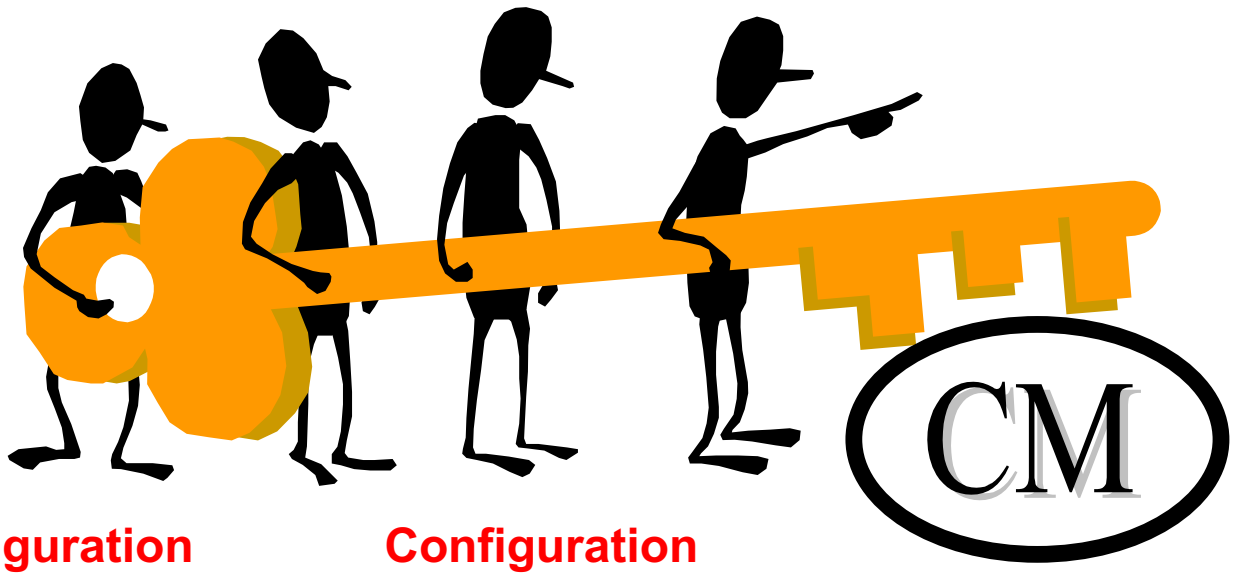
CM is a CMM level 2 key process area



CM activities

Configuration Status Accounting

Configuration Identification

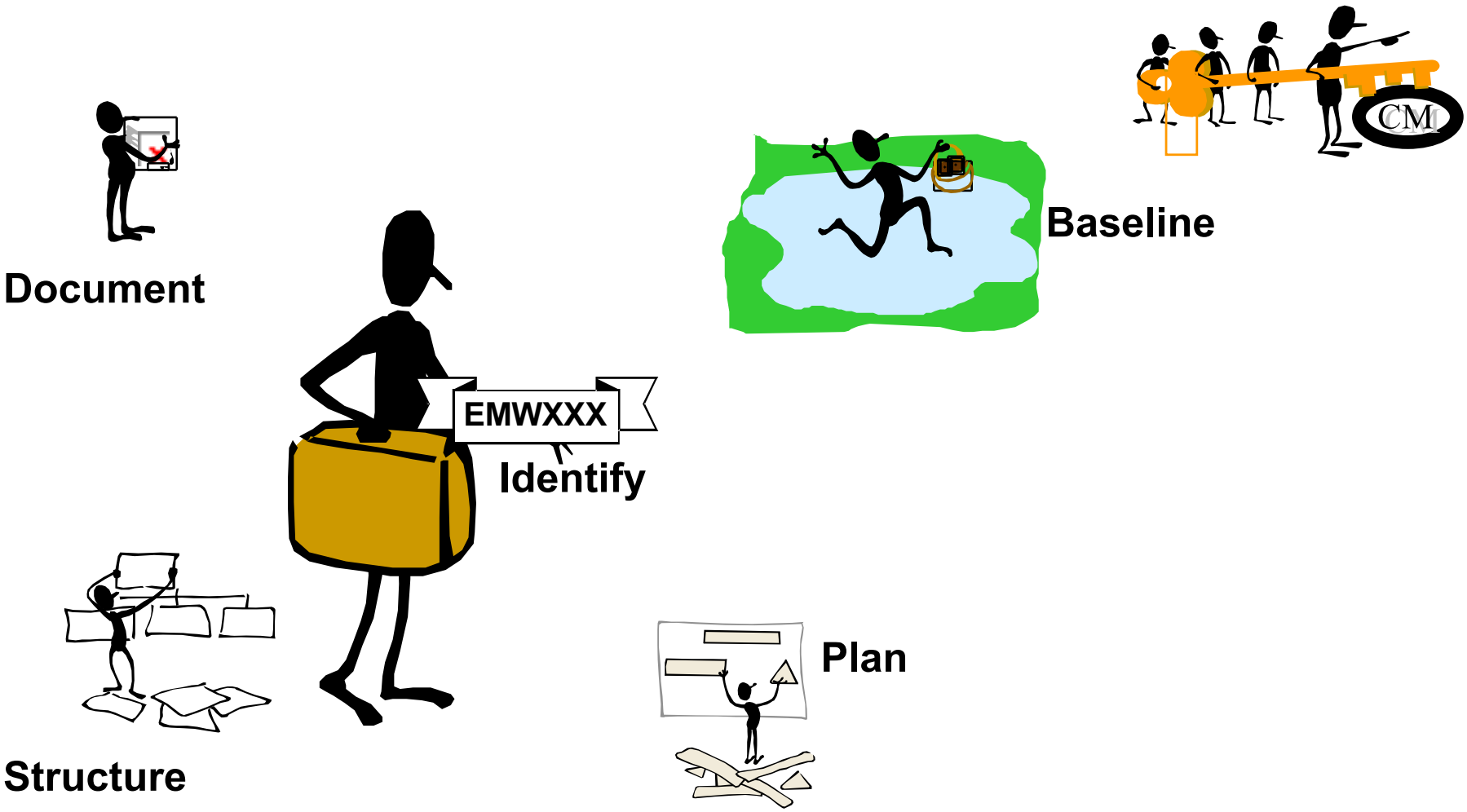


Configuration Auditing

Configuration Control

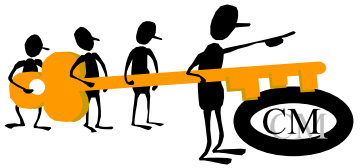
Release Management

CM - Configuration Identification

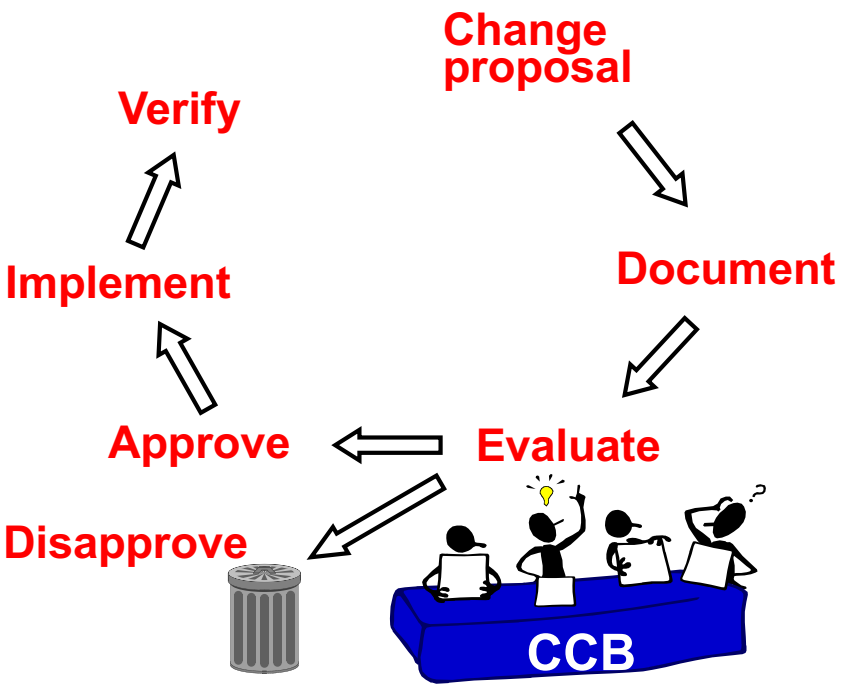


CM - Configuration Control

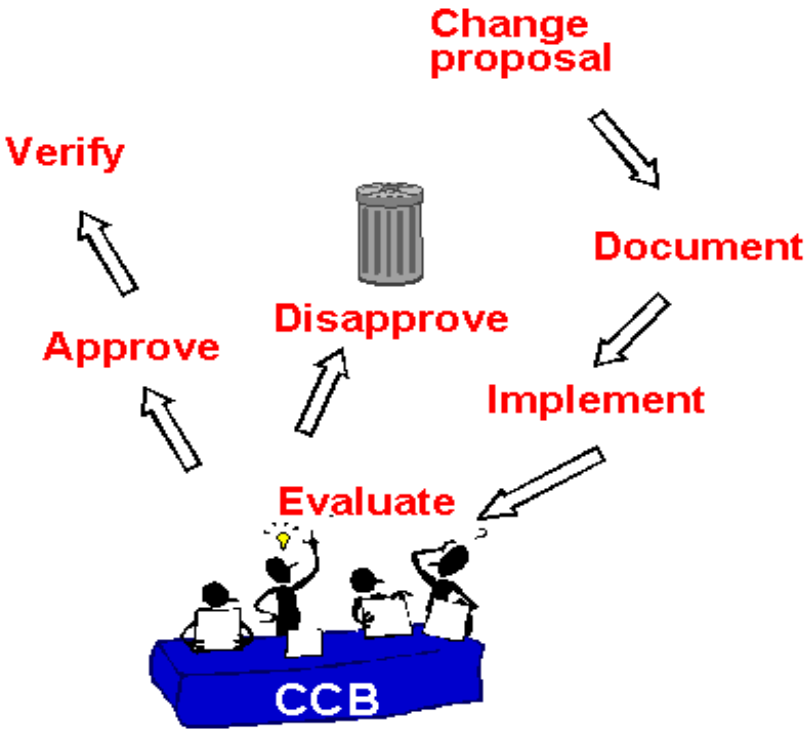
- Change management
- Traceability



Configuration Control

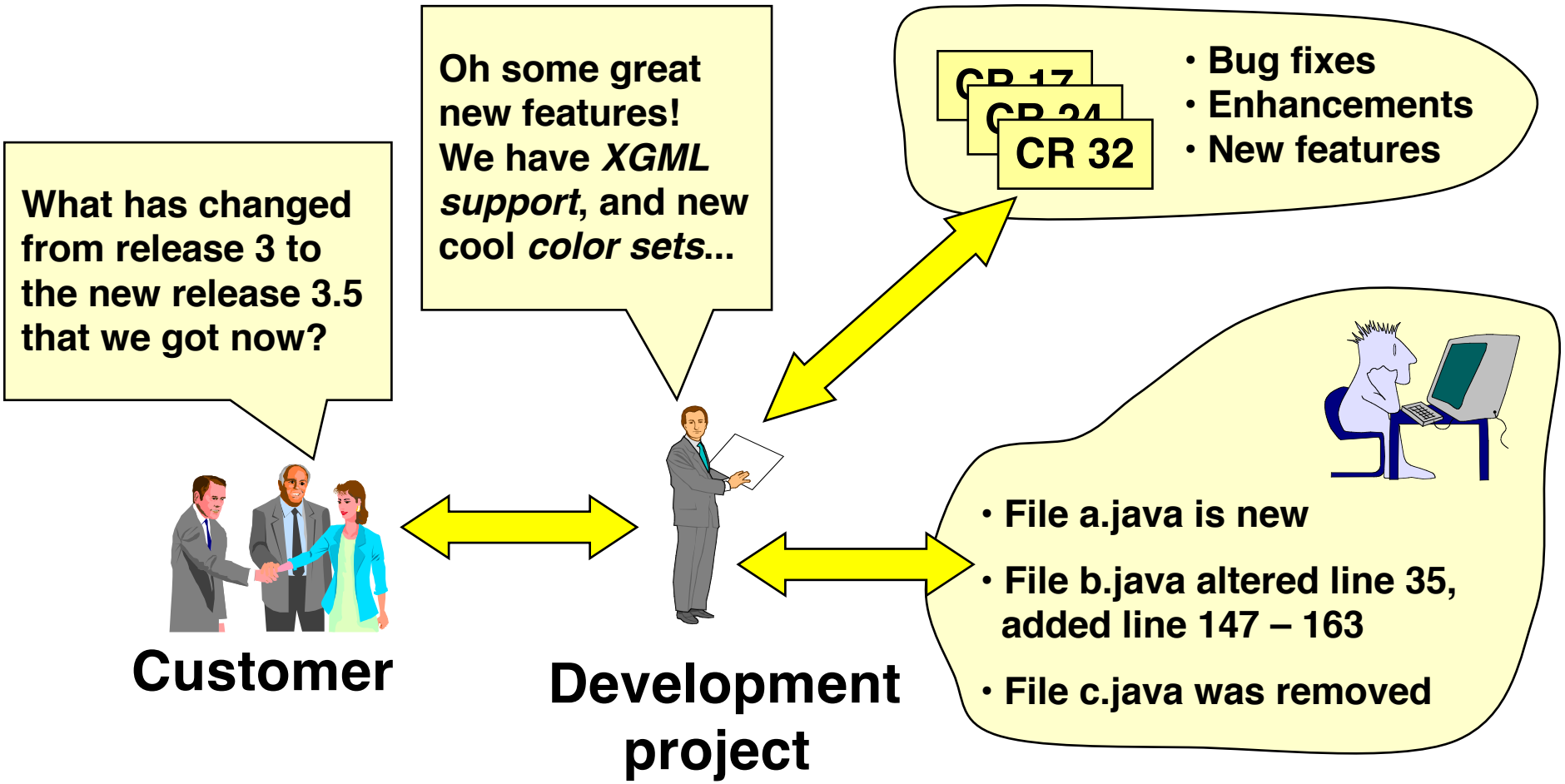


Change Request (CR) Process

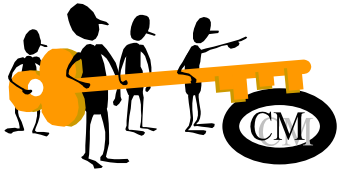


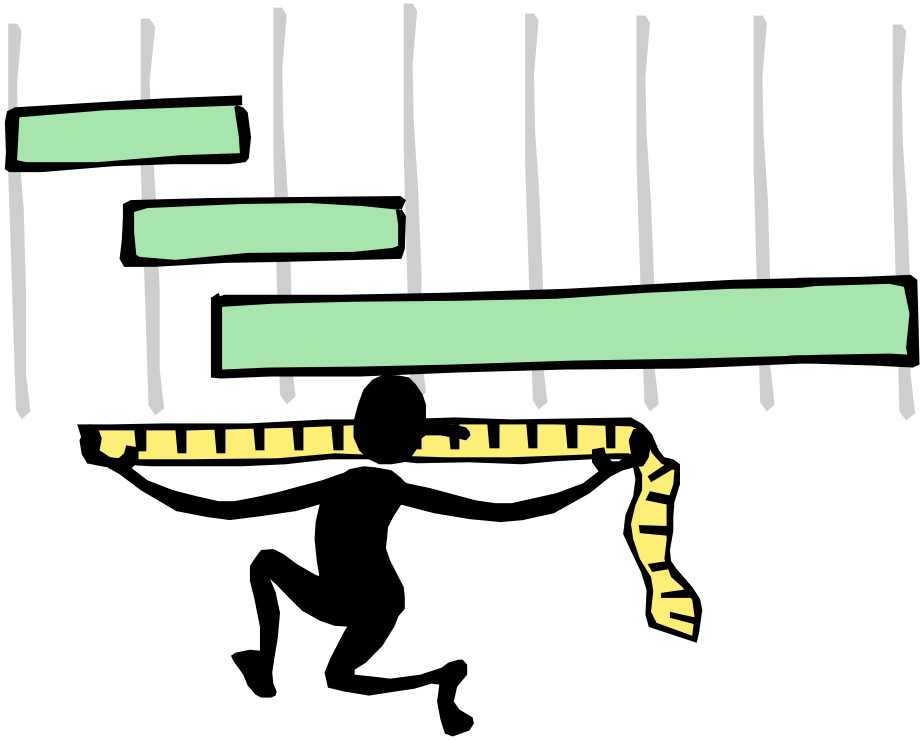
Change process for OSS

The goal of CM - Change tracking



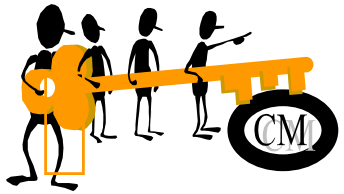
CM - Configuration Status Accounting





Status Accounting

CM - Configuration Auditing



Physical Audit

Functional Audit

Release Management

“Which configurations does this customer have?”

“Did we deliver a consistent configuration?”

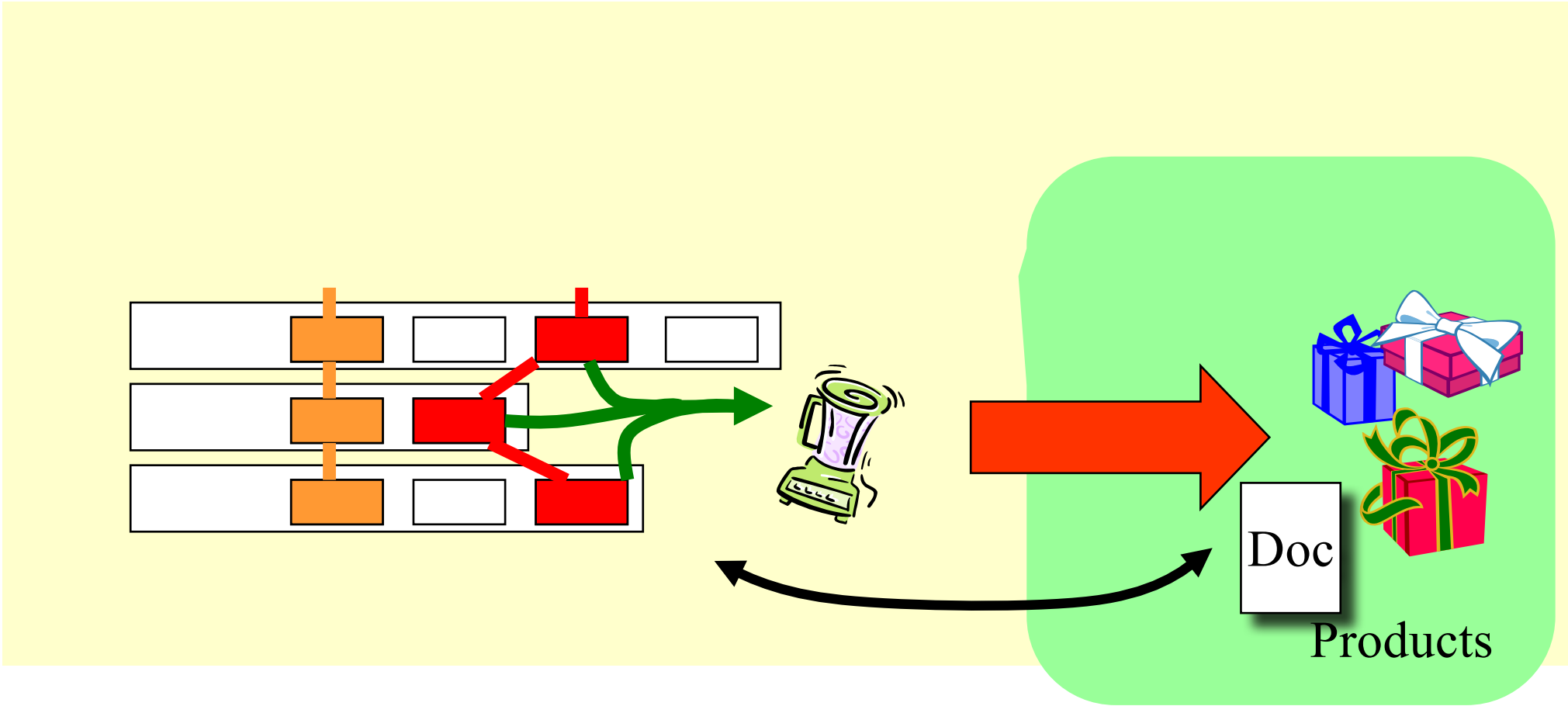
“Did the customer modify the code”

“How exactly was this delivery configuration produced?”

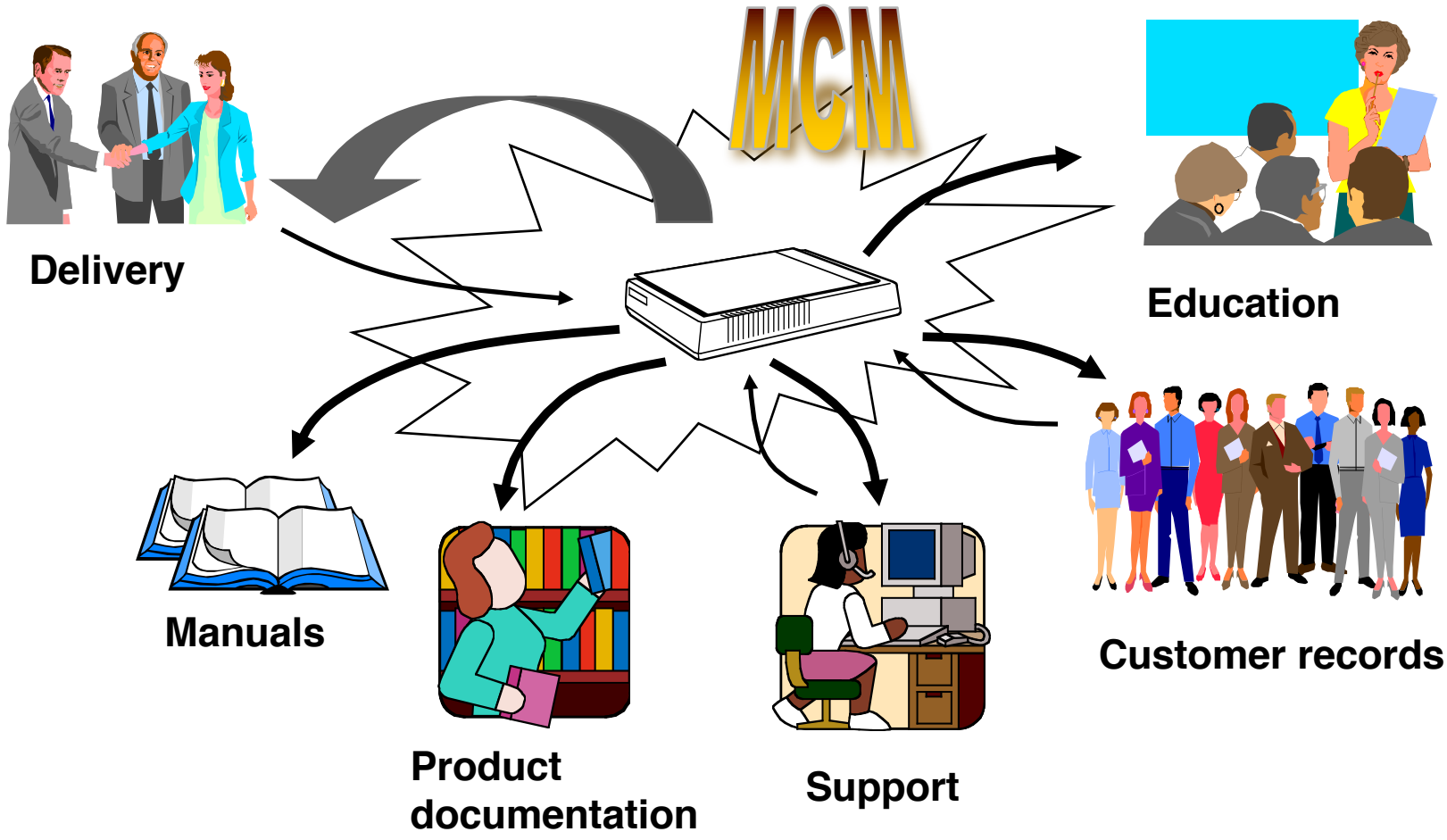
“Were all regression tests performed on this system delivery version?”

“Did we deliver an up-to-date binary version to the customer?”

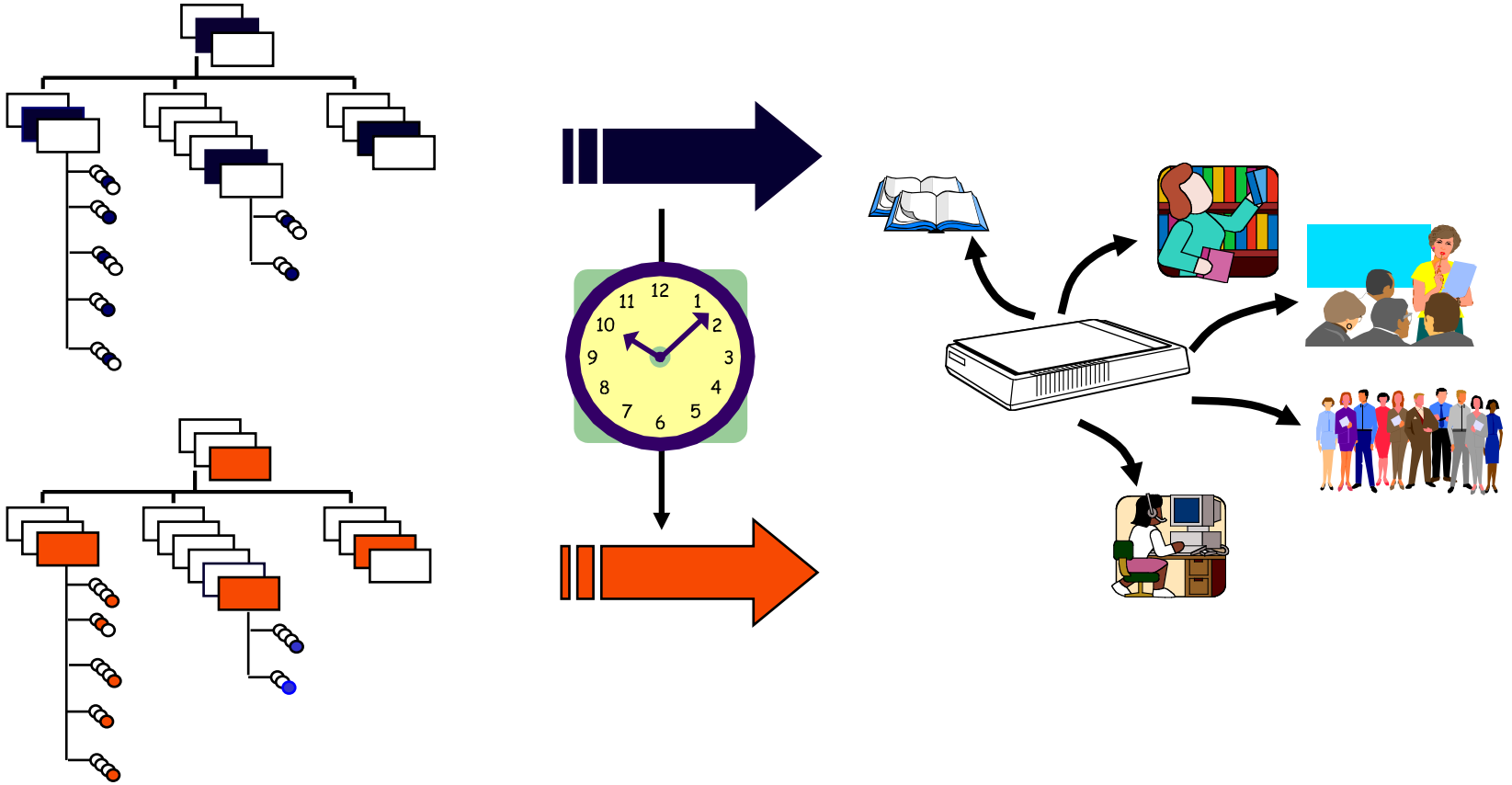
Release Management



The need for CM - The maintenance phase



The goal of CM - Software deployment



SCM for XP development

Support and help for:

- handling source code
- collective ownership
- simple integration
- painless refactoring
- ease of testing
- effortless releasing
- handling document(ation)

© Lars Bendix - Lund Institute of Technology

How does a programmer spend his time?

- 50 % interacting with other team members
- 30 % working alone (pair-programming??)
- 20 % non-productive activities

Common heritage is the reason:

- sharing things
- memory/history
- communication
- co-ordination

© Lars Bendix - Lund Institute of Technology

Problems of co-ordination

Shared data

Double maintenance

Simultaneous update

© Lars Bendix - Lund Institute of Technology

Co-ordination

Working in isolation:

- local dynamicity
- global stability
- problem:
  - multiple maintenance

Working in group:

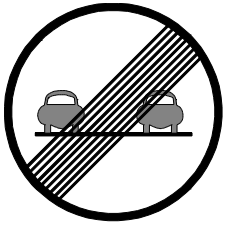
- global dynamicity
- problems:
  - shared data
  - simultaneous update

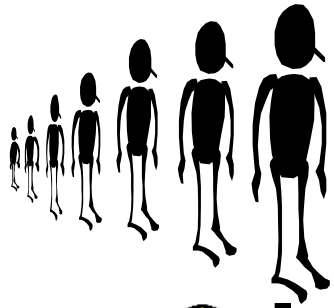
© Lars Bendix - Lund Institute of Technology

Configuration Management
Strategies - Models - Tools

Concurrent development

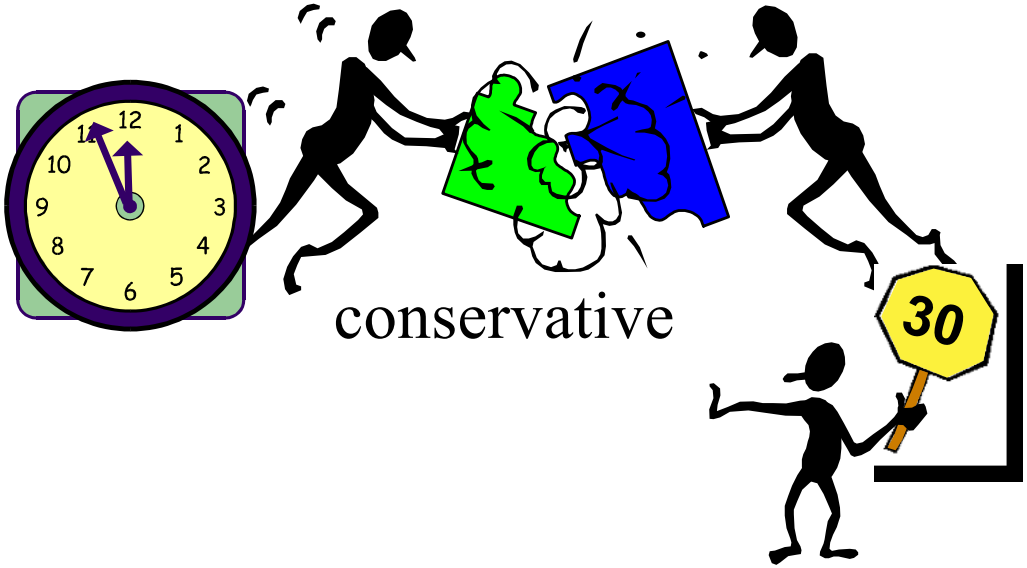
optimistic



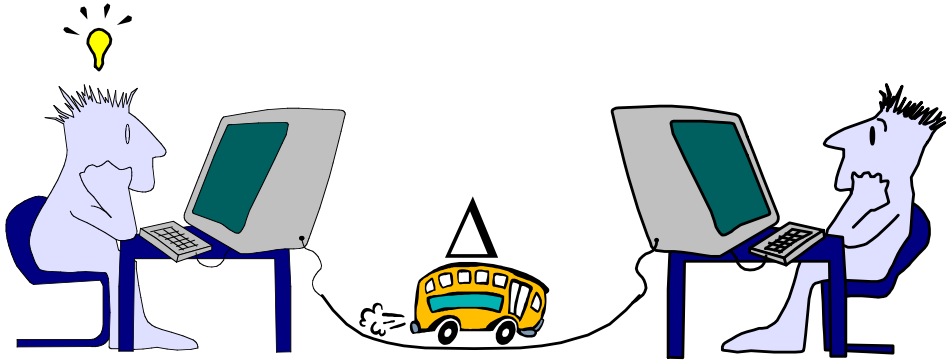


conservative

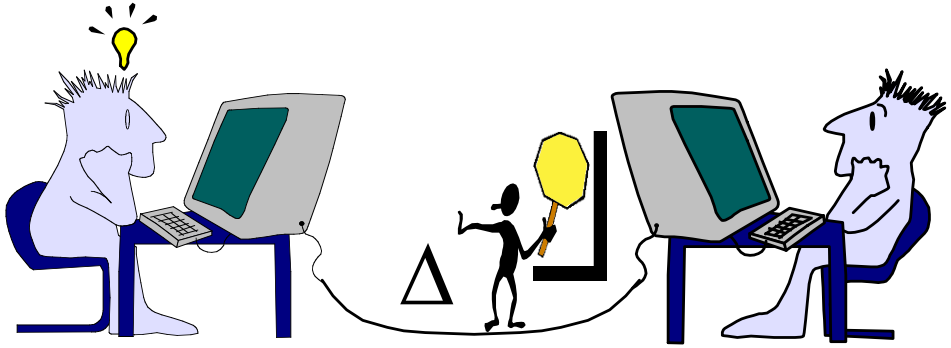
Developing Strategy



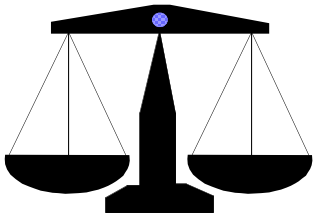
Update Strategy



optimistic



conservative

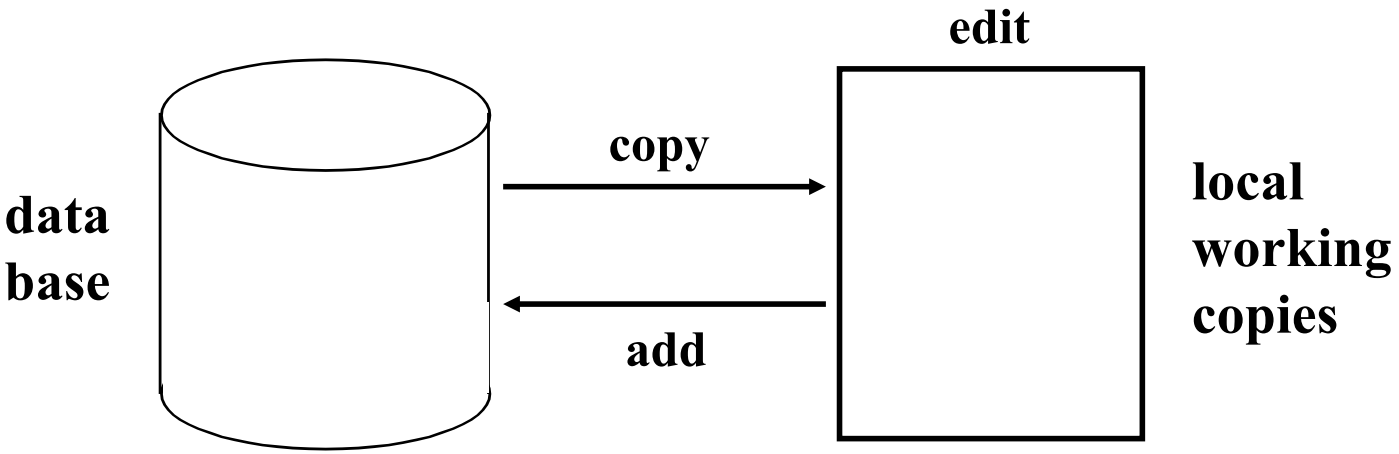


early integration

work in peace (isolation)

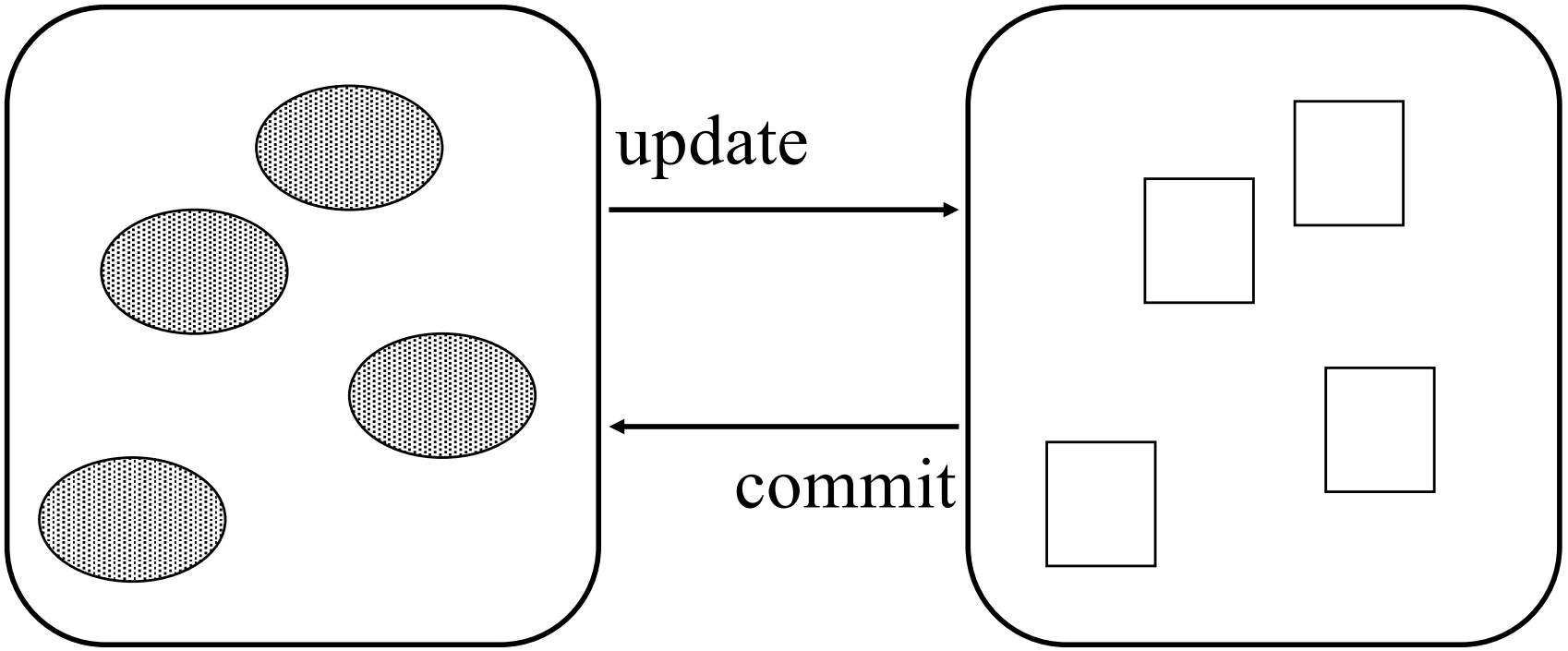
Immutability principle

Principle: components are **immutable**



© Lars Bendix - Lund Institute of Technology

Working



© Lars Bendix - Lund Institute of Technology

Copy/merge work model

Can we *lock* the things we want to work on? NO!

So we **copy** everything to our workspace...
...and everyone else copy to their workspaces...
⇒ double maintenance !!

o

Fortunately ”update” has a built-in **merge** facility:

- We first merge *from* the repository *into* the workspace
- Then we check and fix problems
- Finally we commit (add) to the repository

© Lars Bendix - Lund Institute of Technology

Quotes from XP'ers

- Overall CVS (and CM) was a HUGE help for the project.
- The version history was a real life saver.
- CVS made it possible for 12 people to work on the same code at the same time.
- CVS rules!
- It would have been impossible to merge different people's work without it.
- CVS sucks!
- Branching made releasing much easier.
- We tagged the releases – it served it's purpose.

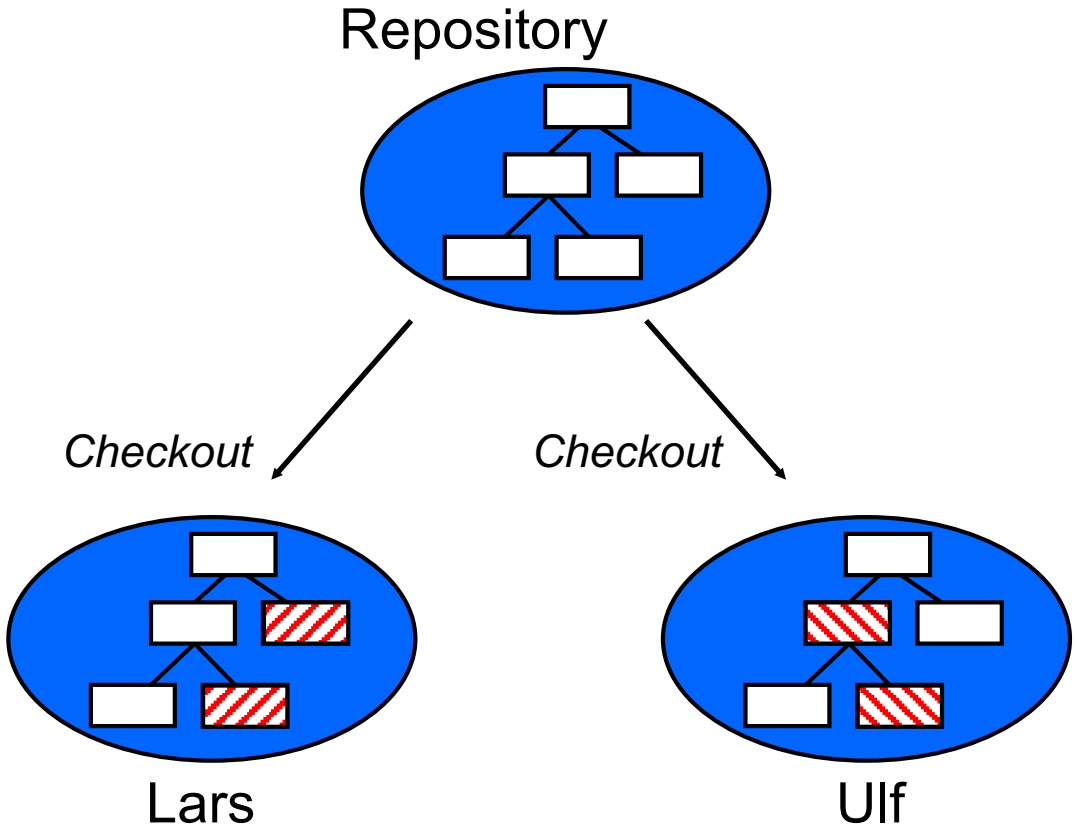
© Lars Bendix - Lund Institute of Technology

So how is CM used?

- update-commit
- merge – merge – merge
- no versioning, diff, tag, …
- change log only to identify people

© Lars Bendix - Lund Institute of Technology

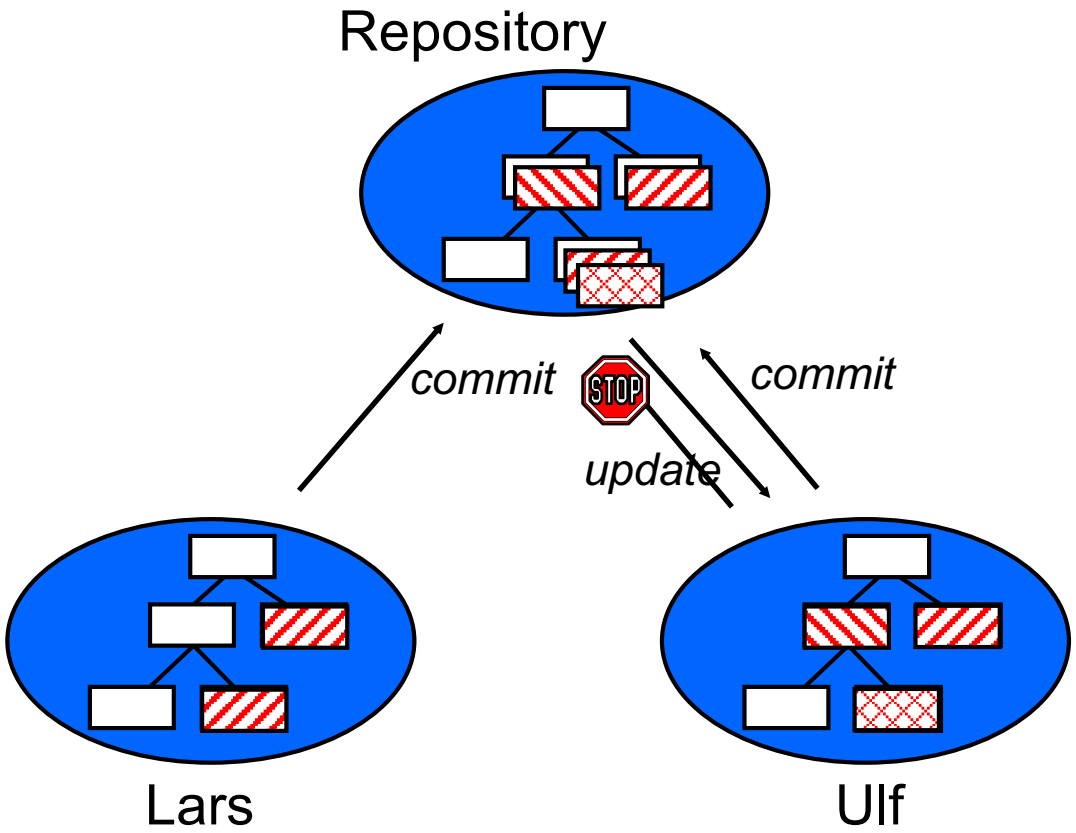
Long transactions I



workspace creation

© Lars Bendix - Lund Institute of Technology

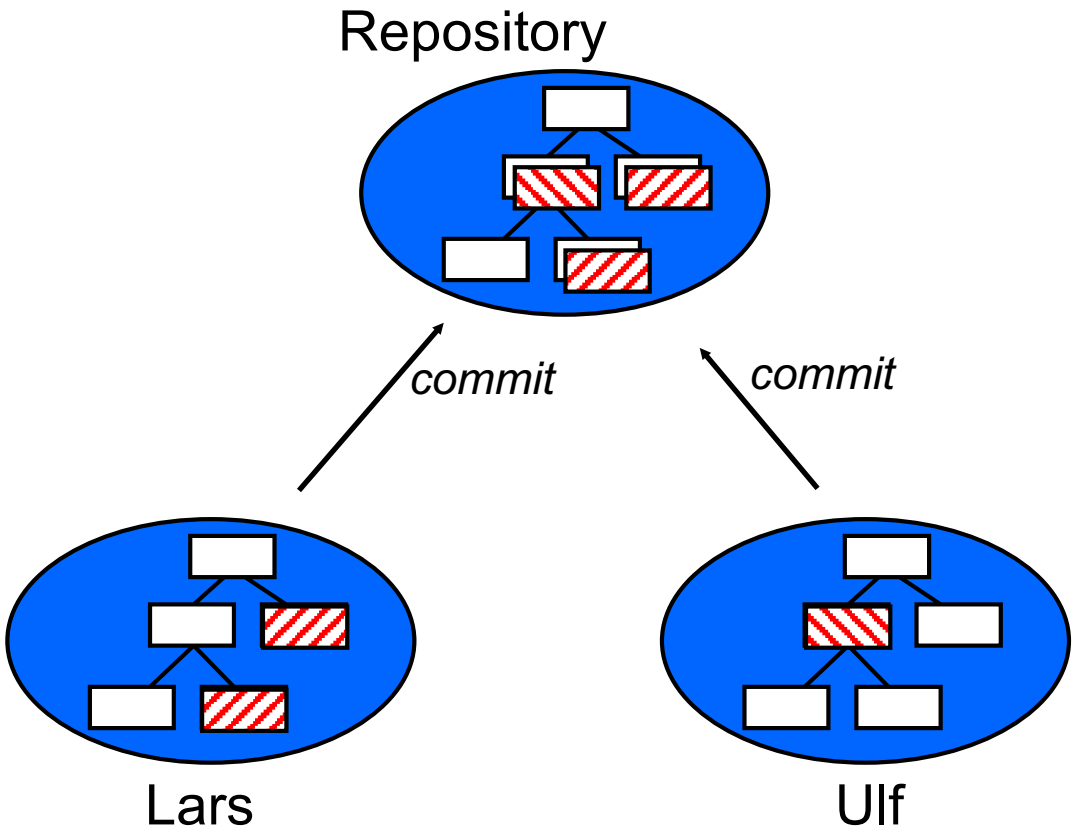
Long transactions II



workspace usage (termination)

© Lars Bendix - Lund Institute of Technology

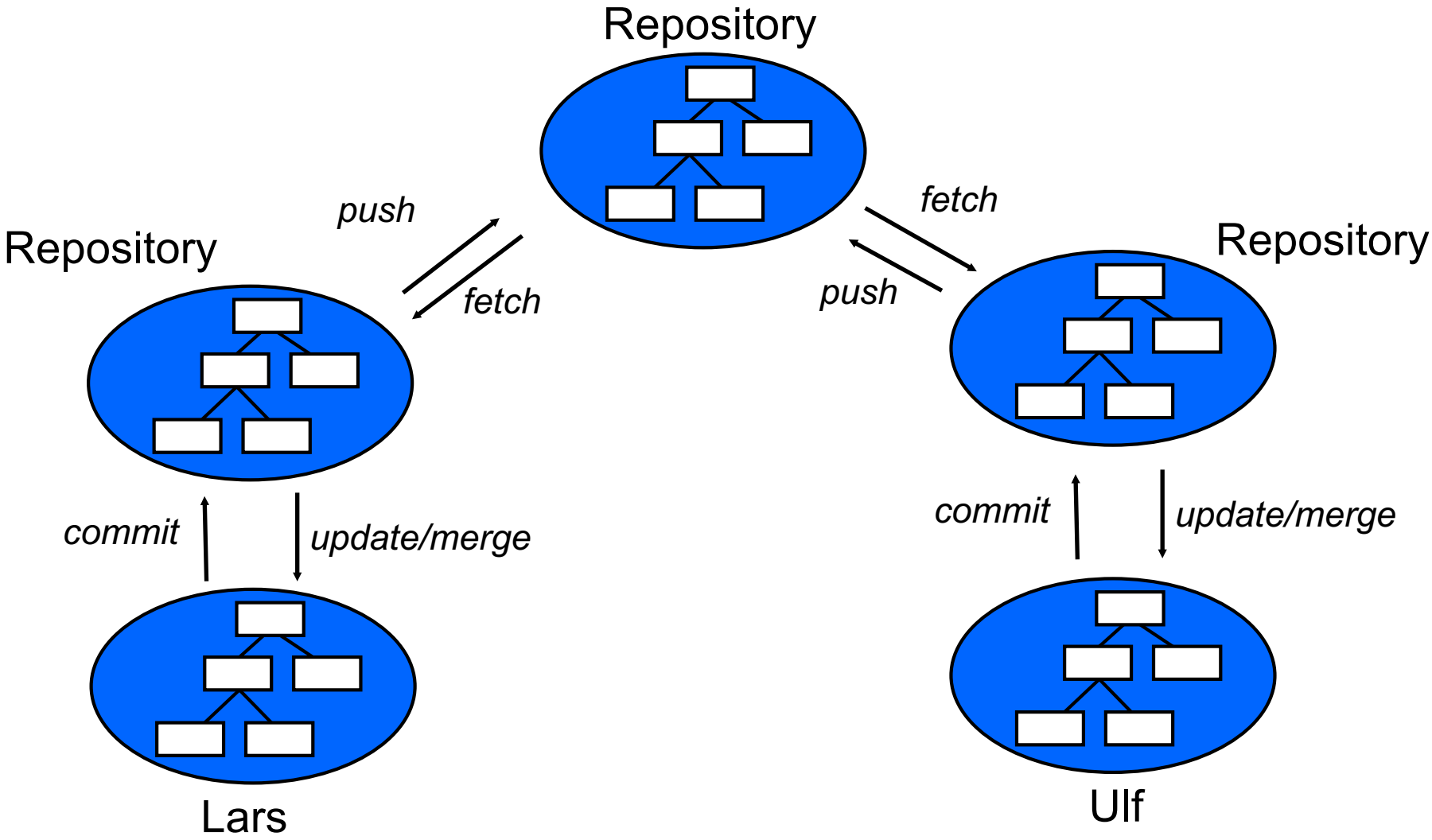
Unfortunately :-(



NO ***strict*** long transactions - so...

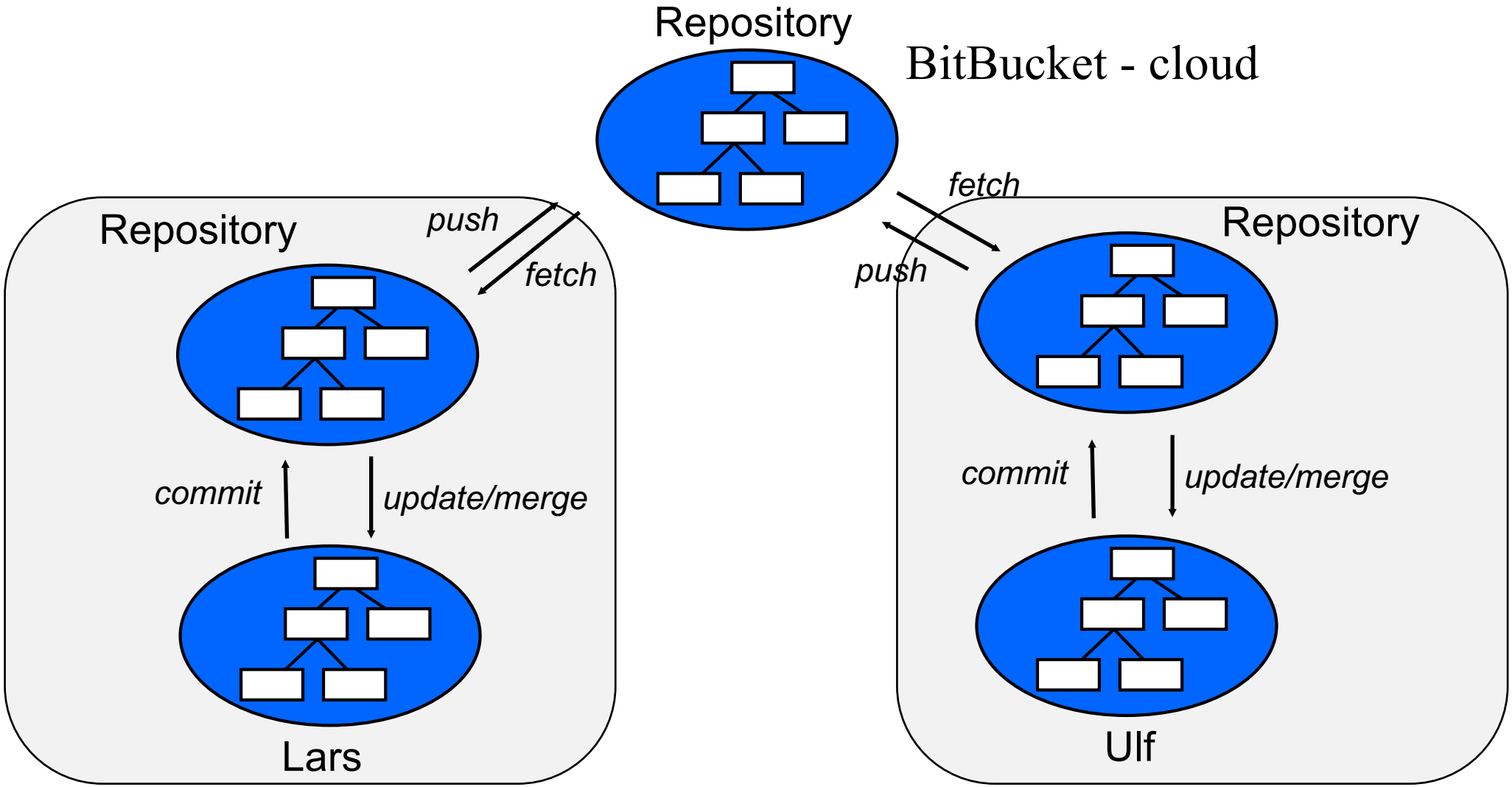
© Lars Bendix - Lund Institute of Technology

Distributed version control



Yes, ***strict*** long transactions (push/pull)

git - BitBucket



Yes, ***strict*** long transactions (push/pull)

Extreme programming

SCM-related practices:

- collective ownership (developer)
- continuous integration (developer)
- refactoring (coding)
- small releases (business)
- planning game (business/developer)
- test-driven development (developer)

© Lars Bendix - Lund Institute of Technology

Collective code ownership

Goal: to spread the responsibility for the code to the team

How/why:

- from individual (pair) to team ownership
- reinforces code review (and readability)
- enables refactoring

Requires:

- team spirit
- frequent integration

SCM dangers:

- huge merge conflicts

© Lars Bendix - Lund Institute of Technology

Integrate continually I

Goal: to reduce the impact of adding new features

How/why:

- ”download” & ”upload” integration
- run tests; update (merge); re-run tests; commit
- ***all*** components must be in repository
- integration machine/responsibility/how often?
- keeps everyone in synchronisation
- keeps the project releasable all the time

© Lars Bendix - Lund Institute of Technology

Integrate continually II

Requires:

- collective source code repository
- short tasks

SCM dangers:

- huge merge conflicts
- false positives

© Lars Bendix - Lund Institute of Technology

Refactor mercilessly

Goal: to find the code's optimal design

How:

- before & after a task, think about refactoring
- changes the structure, but *not* the behaviour
- break out code; remove duplications; …

Requires:

- collective code ownership
- coding standards

SCM dangers:

- big-bang refactorings

© Lars Bendix - Lund Institute of Technology

Release regularly

Goal: to return the customer's investment often

Why/when/how:

- two-way feedback
- at the end of each iteration (daily?)
- clean machine principle
- automating and optimising the release

Requires:

- continuous integration

SCM dangers:

- manual process takes time
- a broken release :-(

© Lars Bendix - Lund Institute of Technology

Play the Planning Game

Goal: to schedule the most important work

Why/how:

- to maximize the value of features produced
- divides planning responsibilities (what/how)
- developers estimate user stories
- developers split stories up into tasks

Requires:

- active customer
- mutual respect

SCM dangers:

- sloppy estimates and work break-down

© Lars Bendix - Lund Institute of Technology

XP process

1. Always start with all of the "released" code.
2. Write tests that correspond to your tasks.
3. Run all unit tests.
4. Fix any unit tests that are broken.
5. When all unit tests run, your local changes become release candidates.
6. Release candidate changes are integrated with the currently released code.
7. If the released code was modified, compare the differences and integrate them with your changes.
8. Rerun tests, fix, rerun tests, fix, rerun ….
9. When the unit tests run, release all of your code, making a new official version.

© Lars Bendix - Lund Institute of Technology

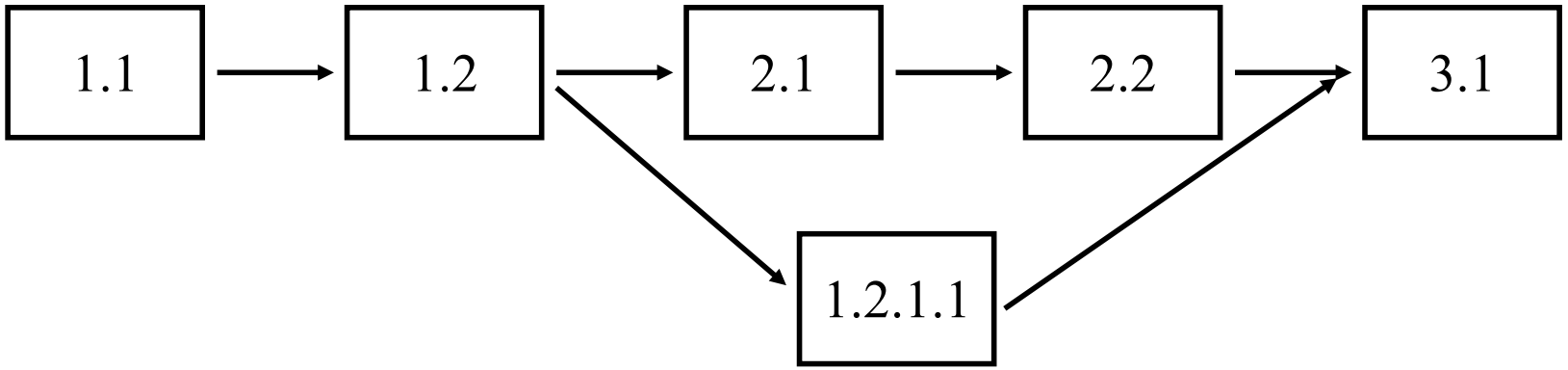
Diffing and merging

Visualisation of differences:

- diff

Merging of branches:

- merge
- always control manually that things went well



© Lars Bendix - Lund Institute of Technology