ETH Zürich

Chair of Software Engineering

Software Architecture

Bertrand Meyer, Carlo A. Furia, Martin Nordio

ETH Zurich, February-May 2011

Lecture 9: Configuration management

About your future

- You will never work alone.
- Other people will mess up your code.
- You will mess up other people's code.
- You will never write a (major) program from scratch.
- The software you will work on was already there when you joined the company.
- The software you will work on will still be there when you leave the company.

Configuration management – the long story

"Configuration management is unique identification, controlled storage, change control, and status reporting of selected intermediate work products, product components, and product during the life of a system."

(Anne Mette Jonassen Hass: "Configuration Management; Principles and Practice")

Configuration management – the short story

Configuration management is about the role of

TIME

in software development.

It is the task of tracking and controlling changes.

Ten key SCM activities

1. Accessing and retrieving software
2. Retrofitting changes across the development life cycle
3. Migrating changes across the development life cycle
4. Managing the compile and build process
5. Managing the distribution of changes
6. Obtaining approvals and sign-offs
7. Managing software change requests
8. Coordinating communication between groups
9. Obtaining project status
10. Tracking bugs and fixes

(Software Configuration Management, Jessica Keyes)

Change management

- CM has to record
  - WHICH document was changed.
  - WHAT was changed.
  - WHO has done the change.
  - WHEN the change was made.

- The history of the changes should be visible.
- It should be possible to undo changes.
- It should be possible to view the version of a document at a certain point in time.

Three parts of SCM

1. Version control systems
   a) Local version control
   b) Centralized version control
   c) Distributed version control
2. Build management systems
3. Bug and issue tracking systems

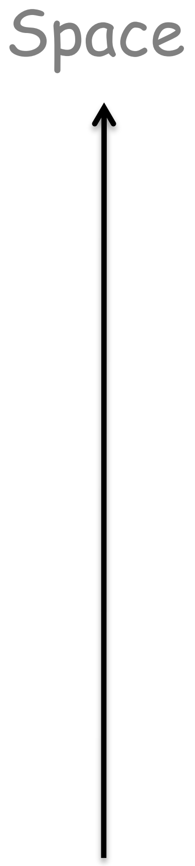
Three parts of SCM

1. Version control systems
   a) Local version control
   b) Centralized version control
   c) Distributed version control

Versions/Revisions

- Versions (also called revisions) give a unique time-dependent identification to each document.
- Deciding for proper version names is the basis of successful software configuration management.
- Version numbers can have multiple levels (within one project).
- Examples:
  - Versions of documents:
    - Example: REQDOC-20100103-R4
  - Versions of source code:
    - 1.1, 1.2, 1.3, 1.4, 1.4.1, ...
  - Versions of binary builds:
    - 610, 611, 612, ...

Space



root_class.e

readme.txt

gui/

gui/main_window.e

gui/dialog_window.e

net/

net/ftp_protocol.e

...

A system that stores and organizes documents over space is called a file system.

Space and time

root_class.e

readme.txt

gui/

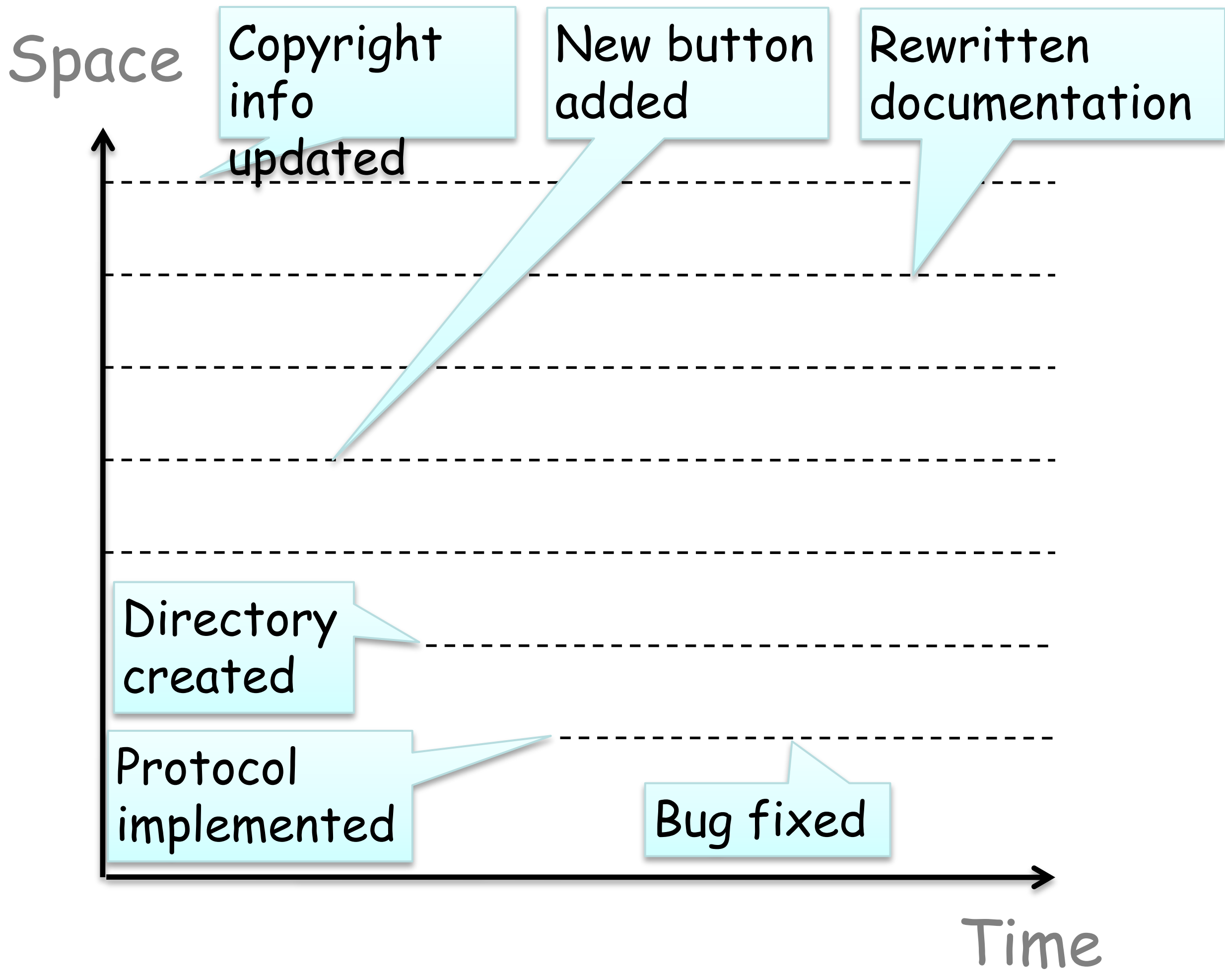
gui/main_window.e

gui/dialog_window.e

net/

net/ftp_protocol.e

...



A system that stores and organizes documents over space and time is called a repository.

Version control systems (VCS) manage…

- Source code revisions and variants
- Binary versions of the software (builds)
- Requirements and analysis documents
- Design documents, UML diagrams
- Test cases and test results
- System configuration information
- …

VCS are about…

- ... knowing who has done what to which document in the past.
- ... enabling different people at different locations to work on the same set of documents at the same time.
- ... going back to an old version of the documents in the case that the path taken was not good.
- ... tracking the quality of the software over time and to stop software regression.

Local version control

- Each document under independent version control
  - File-level granularity
- Versions of the document are stored in the document itself
- Limited support for branching/merging
- Mostly useful in single-user scenarios, or with serialized access to files.

Local version control

Examples:

- Microsoft Word's track change
- SCCS (Source Code Control System)
  - First version control software
  - First version (in SNOBOL): 1972, Bell Labs, Marc Rochkind
  - File format still used by other VCS
- RCS (Revision Control System)
  - Free and improved version of SCCS
  - First version: 1982, Purdue University, Walter Tichy
  - Today part of the GNU project
  - Improvements over SCCS:
    - Automated storing, retrieval, logging and identification of revisions
    - Store deltas ("diffs")
    - Locking of files

Lock-modify-write

- Also called “lock modify unlock”
- File is locked by a user for modification
- System prohibits changes by other users when locked
- Only approach possible with binary files (and other formats with complex structure)

Advantage: No conflicts

Drawback: Wait until file is unlocked

Based on the philosophy of pessimistic version management

What a session with RCS looks like

1. Create a file
2. Check-in: `ci test.txt`
   - creates `test.txt,v`
   - destroys `test.txt`
3. Check-out: `co test.txt`
   - generates back test.txt (in the latest version)
   - write-protected by default!
4. Do your edits on the working copy `test.txt`
   - after removing write-protection
5. Check-in changes: `ci test.txt`
   - fails because you need a lock to check in!
6. Retroactively lock file: `rcs -l test.txt`
7. Have a look at `test.txt,v`

Centralized version control

- Client/server system with access control
  - Server stores versions of a whole project and history
  - Clients check out a local working copy
- Examples:
  - CVS (Concurrent Versions System)
    - First versions: 1986
    - Open source
    - Developed from RCS
    - No locking by default (but explicit locking available)
  - SVN (Subversion)
    - First versions: 2000
    - Open source, maintained by the Apache foundation
    - "A better CVS"

CVS

- Best known VCS tool.

  http://www.cvshome.org

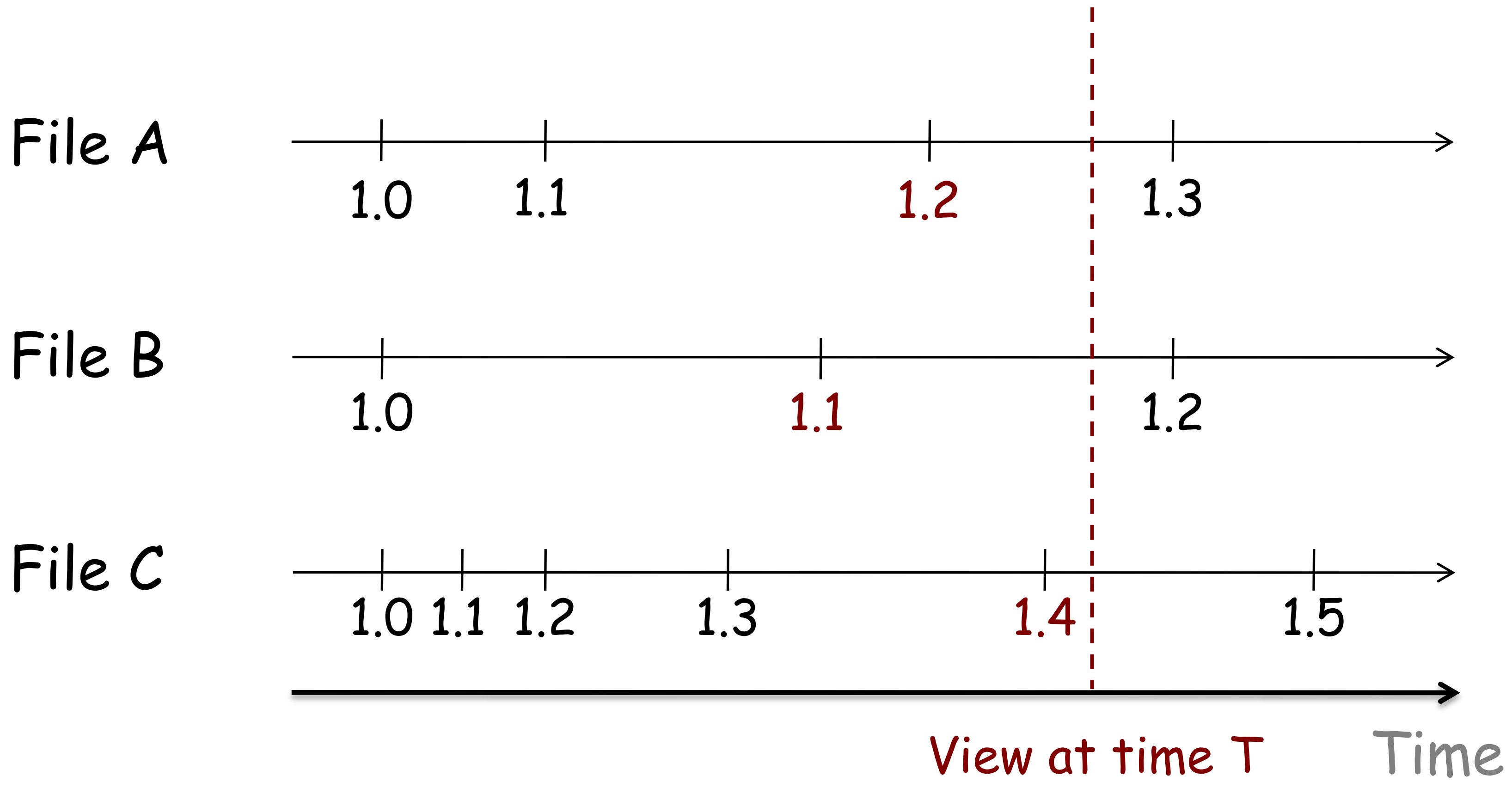
- Key to most open-source development projects and used in many companies.
- Based on RCS (Revision Control System)
  - extended to handle multiple files collectively
- Command-line program
- Graphical User Interfaces:
  - WinCVS (Windows) / GCVS (Unix)
  - Integration into Eclipse and many other IDEs

CVS

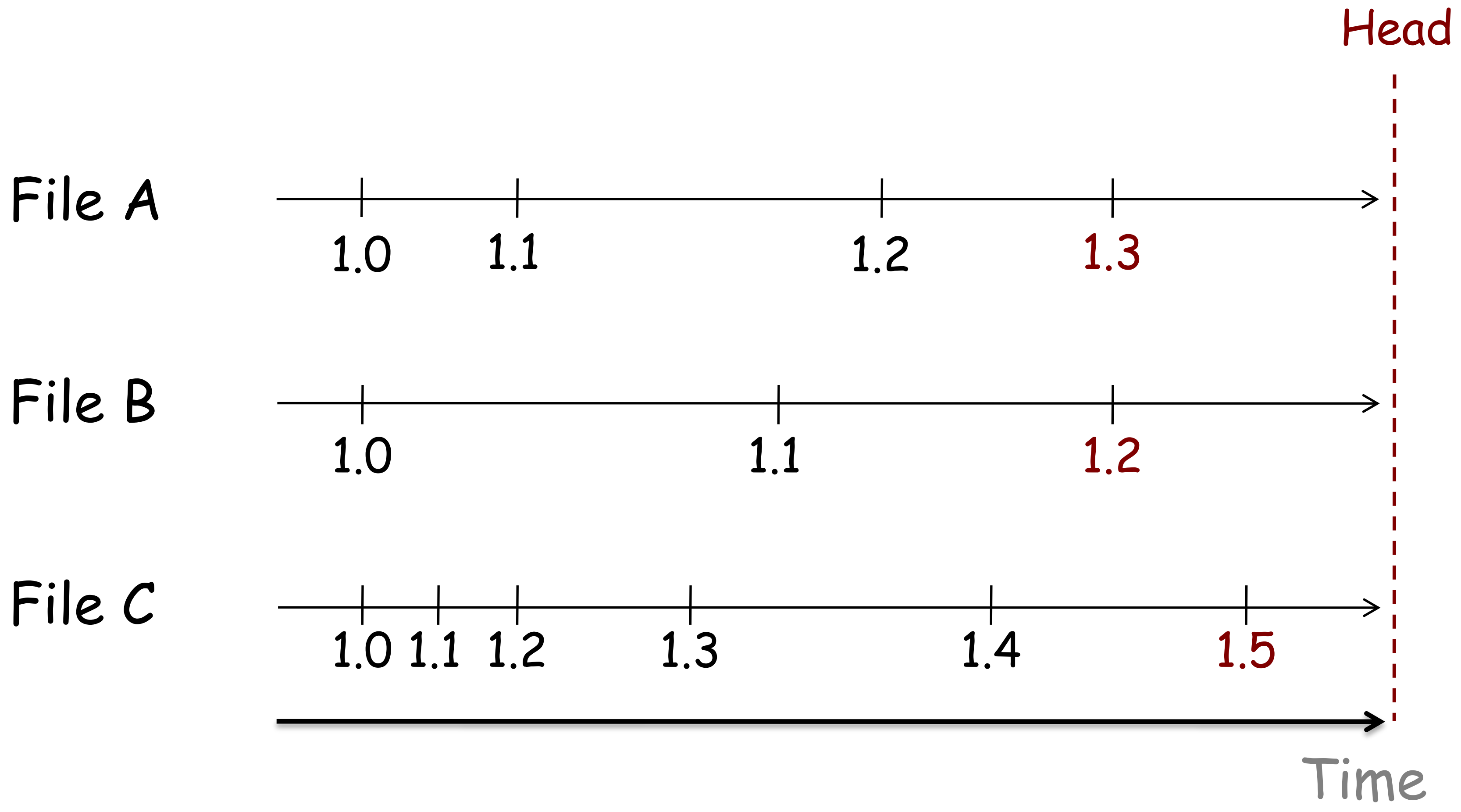
Main features of CVS:

- Central repository
- Checkout creates a copy of the files on the local machine.
- File-based versioning
- Distributed work over the Internet with PSERVER (insecure) or SSH (secure) protocols.
- Can work with binary and ASCII encoded files, but is not very practical with binary files (no diffs).
- Supports primitive conflict resolving commands.

Views

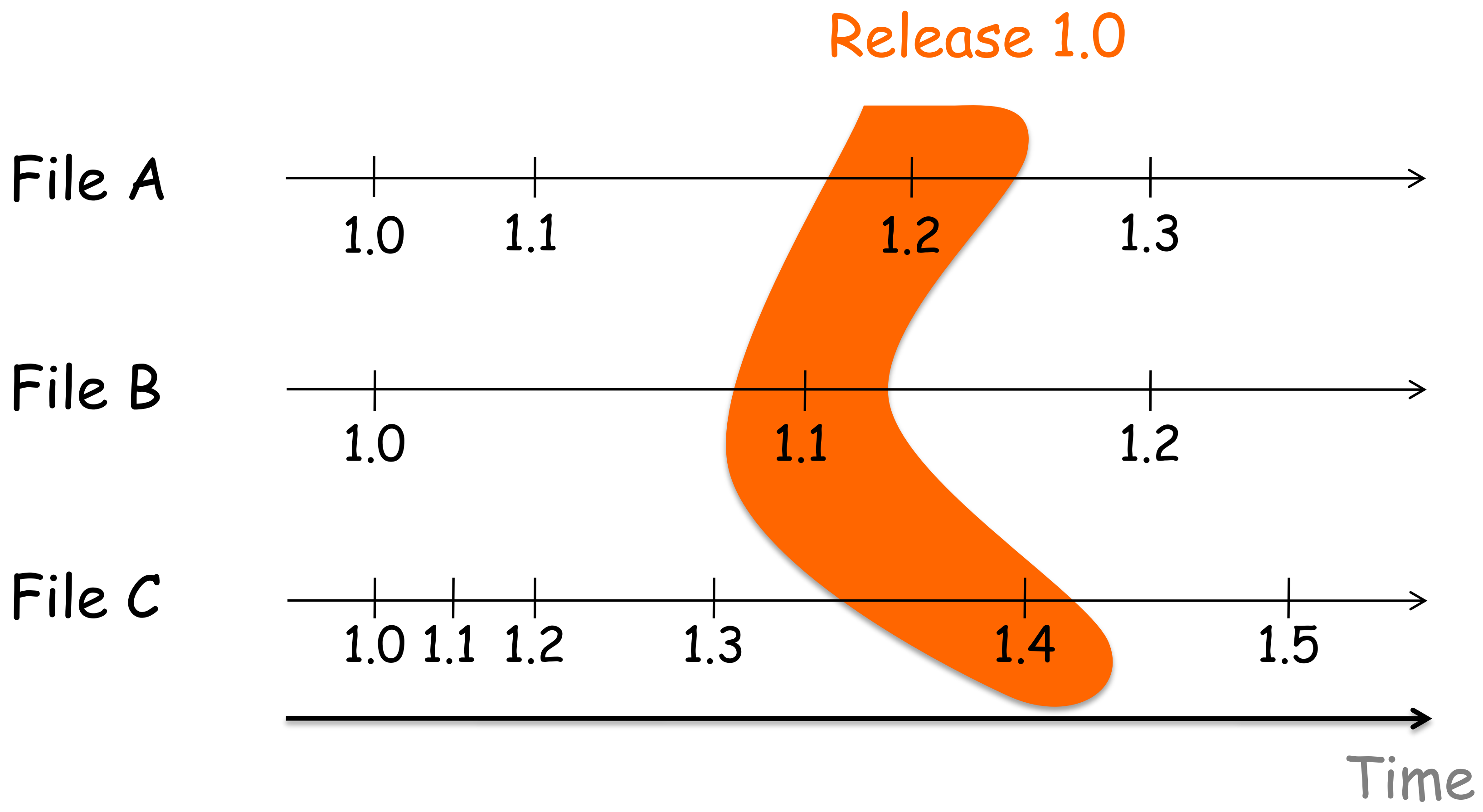

File A
1.0
1.1
1.2
1.3
File B
1.0
1.1
1.2
File C
1.0
1.1
1.2
1.3
1.4
1.5
View at time T
Time

Head/Tip



Head = View at the "latest" time

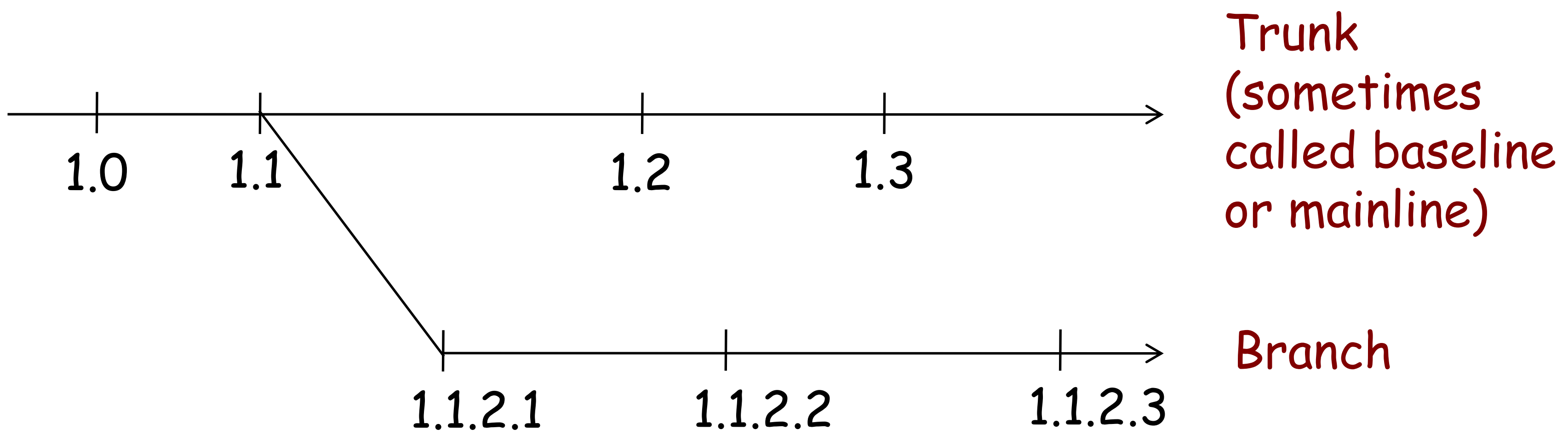
Tagging/labeling



Tag/Label = Important snapshot in time

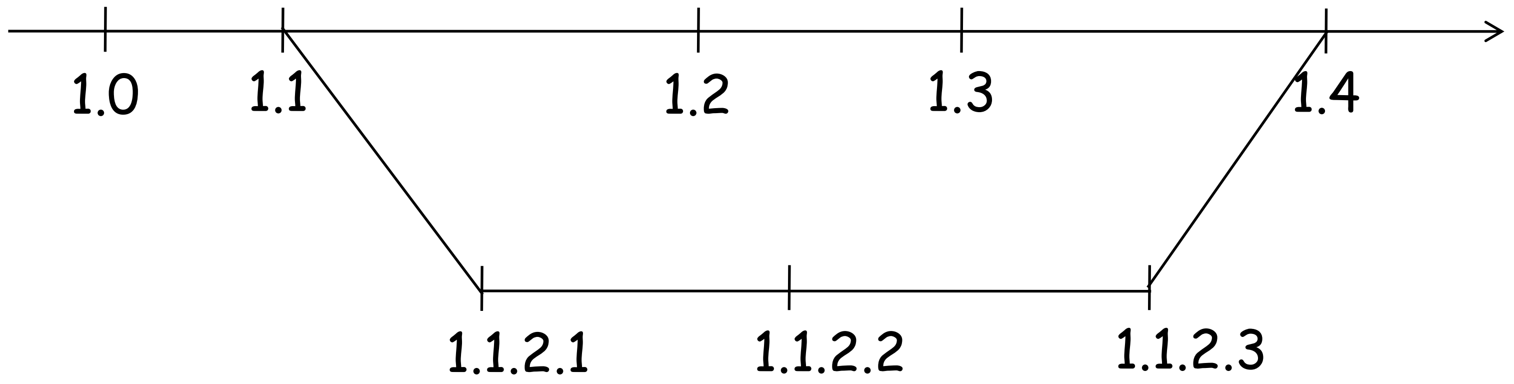
Branching

Creating multiple variants of a set of documents is called branching (or forking)



Merging

Joining variants that were developed independently for some time is called merging

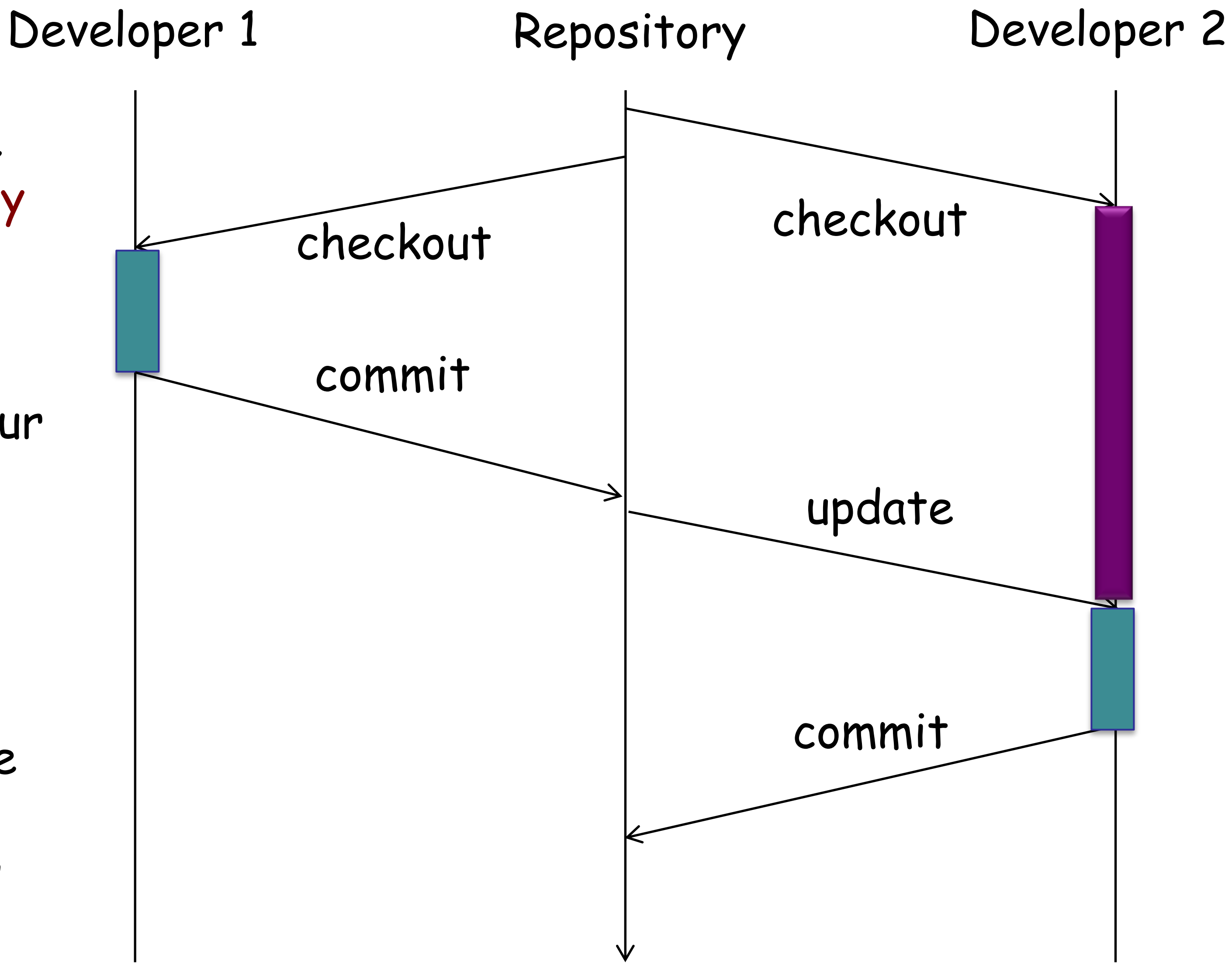


Distributed development

checkout: create local working copy of repository

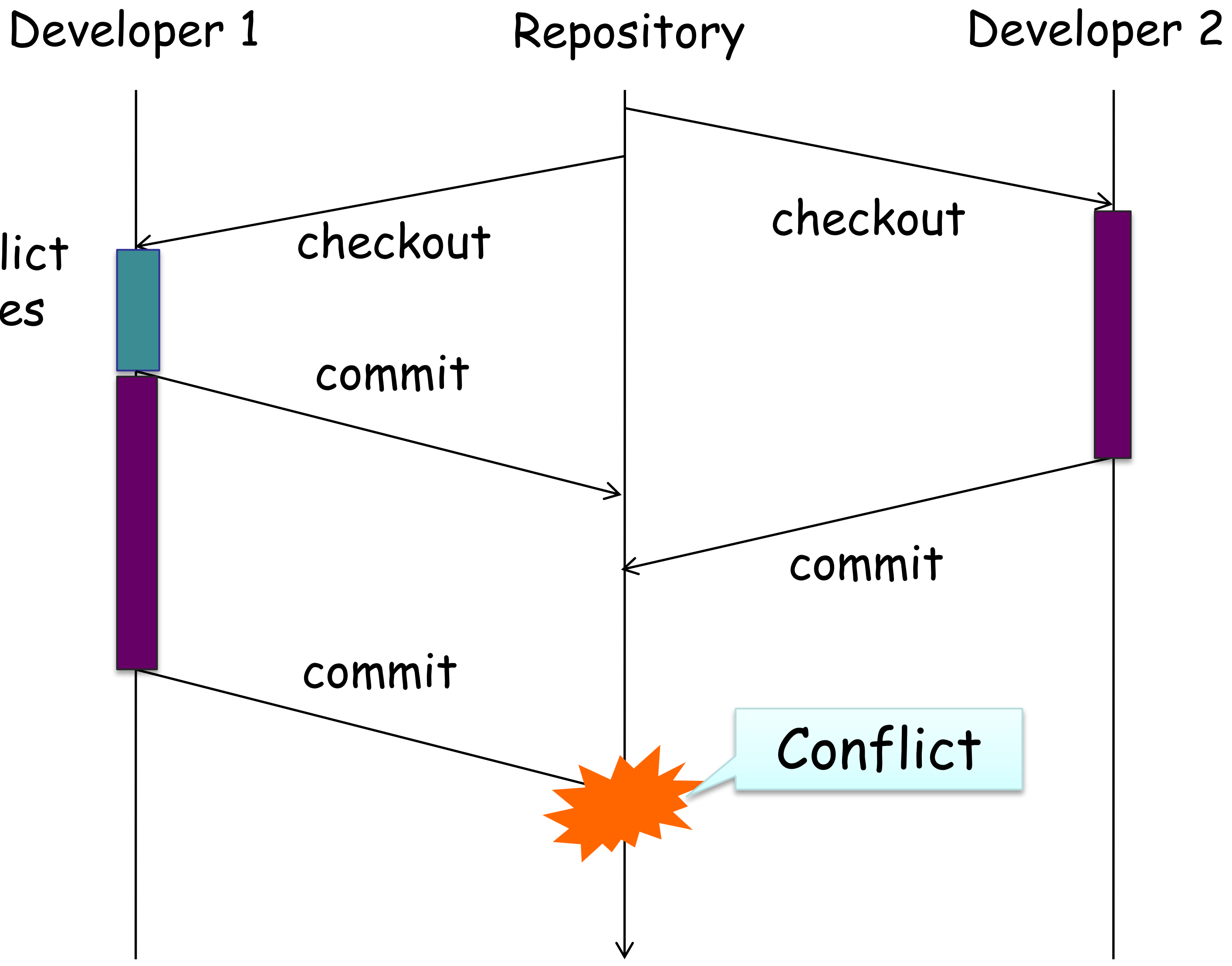
commit: write your changes to the repository

update: merge changes from the repository into the local working copy



Distributed development

resolve: user intervention to address a conflict between changes on the same document



Copy modify merge

- Allows simultaneous modifications of a document
- Changes need to be merged automatically or manually

Advantage: Simultaneous editing

Disadvantage:

Not applicable to binary files

Risks of conflicts

Based on the philosophy of optimistic version management

Used by all modern centralized (and distributed) version control systems

What is wrong with CVS?

- File-based management
  - What happens when you rename files?
  - When you change the directory structure?
- No atomic commits
  - A commit operation crashing may corrupt the repository
- Network protocols are problematic
  - PSERVER is not encrypted
  - RSH is obsolete
  - SSH lacks anonymous access
  - Too much communication between client and server
- Inefficient storage of binary files
- Not designed for heavy branching

SVN (Subversion): references

Available at:

http://subversion.tigris.org/

Book on SVN at:

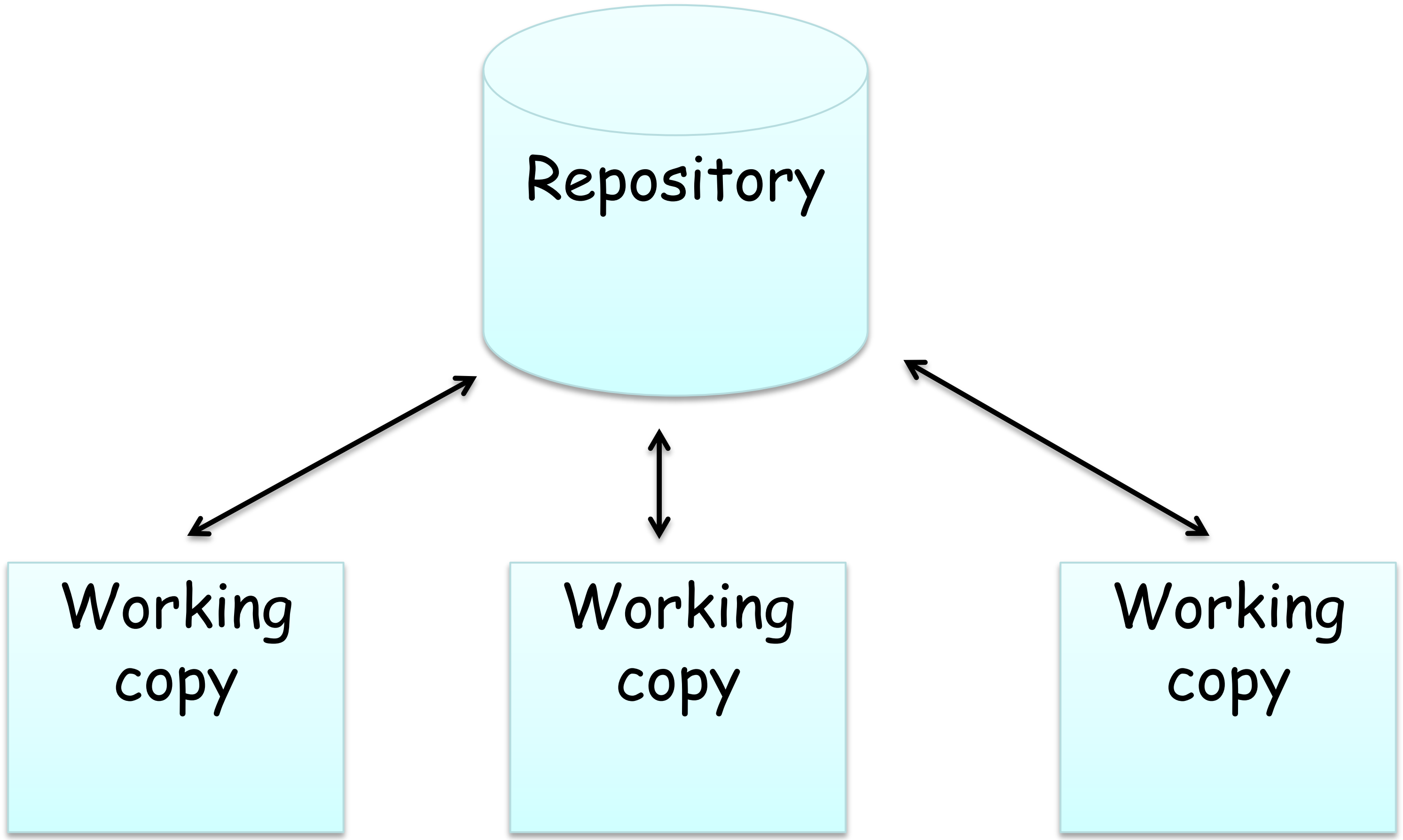
http://svnbook.red-bean.com

"Subversion is meant to be a better CVS, so it has most of CVS's features. Generally, Subversion's interface to a particular feature is similar to CVS's, except where there's a compelling reason to do otherwise."

-- Subversion Homepage

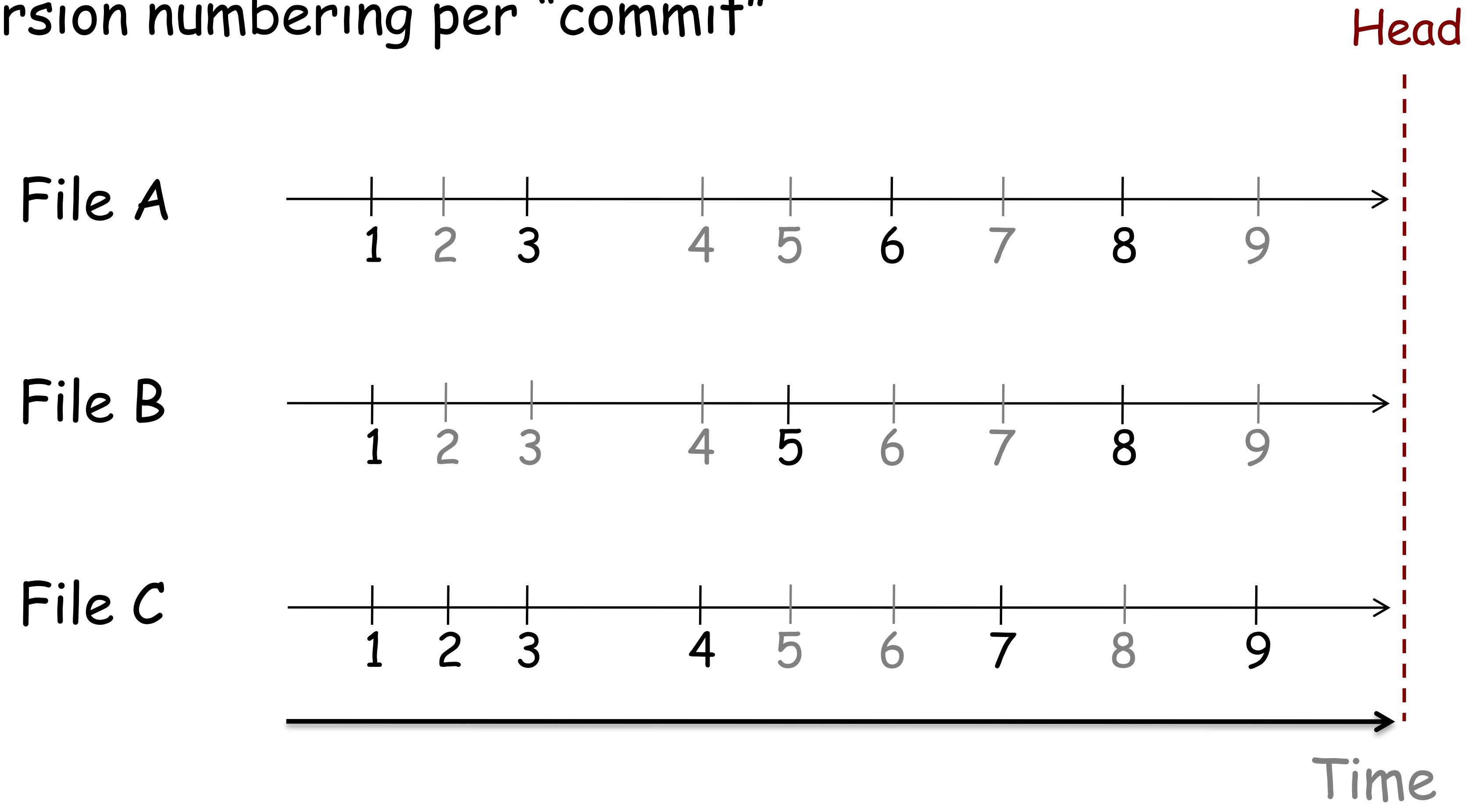
Working copies

- Subversion uses a client server architecture
- Every developer works on his/her "working copy"

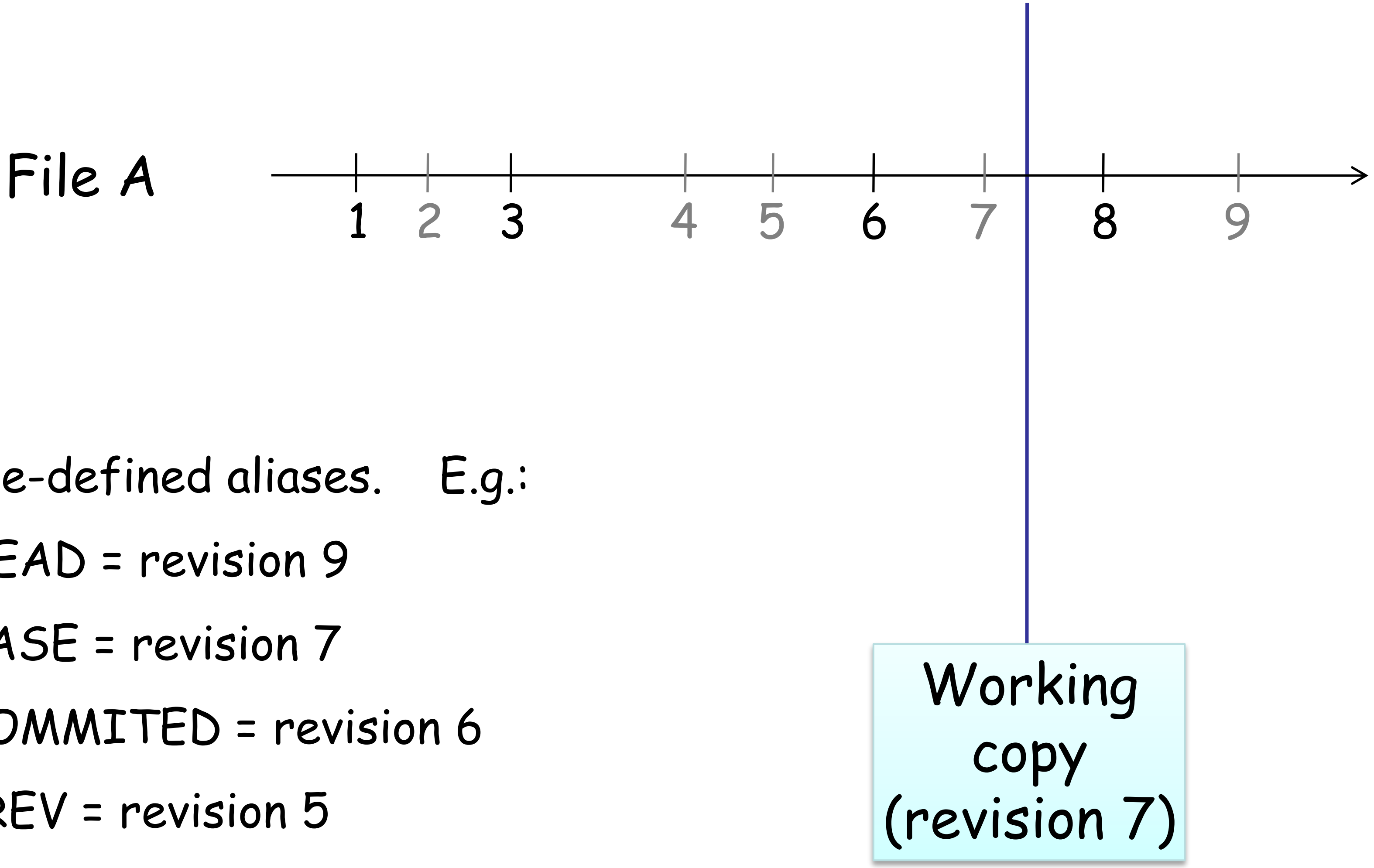


Version numbering SVN

Version numbering per "commit"



HEAD, BASE, COMMITTED, and PREV



Pre-defined aliases. E.g.:

HEAD = revision 9

BASE = revision 7

COMMITED = revision 6

PREV = revision 5

Directories and changes

- SVN tracks tree structures, not just file contents
- You may move files (move = copy + delete)
- You may add/delete/copy/move directories, but the changes in the repository only take place after committing
- Directories have version numbers
- Branching means copying the directory structure into a new place in the repository (using a “diff” copy)

Subversion Commands

- svn checkout
- svn update / svn revert
- svn commit
- svn info / svn log / svn status
- svn add / svn delete / svn move / svn mkdir
- svn copy
- svn diff
- svn merge / svn resolved
- svn cat / svn list / svn blame
- svn export / svn import / svn switch

What’s wrong with SVN?

"Subversion is meant to be a better CVS, so it has most of CVS's features."

“[...] my hatred of CVS has meant that I see Subversion as being the most pointless project ever started. The slogan of Subversion for a while was "CVS done right", or something like that, and if you start with that kind of slogan, there's nowhere you can go. There is no way to do CVS right.”

-- Linus Torvalds

What’s wrong with SVN?

- Rename/move not behaving robustly in all situations
- Misses some “super-user” management features that would come handy

  E.g.: permanently remove all versions older than X
- Centralized version control has issues with large development communities and/or with highly cooperative development
  - How to enforce different access levels?
  - To branch or not to branch?
  - Commit-update: how frequently?

Distributed version control systems (DVCS)

- Every working copy is a repository with version history
  - a backup of the code base
  - a potential branch
- Synchronization uses the exchange of patches (change-sets) with unique ids between peers
  - “Pull” changes from other repositories
  - “Push” changes to other repositories

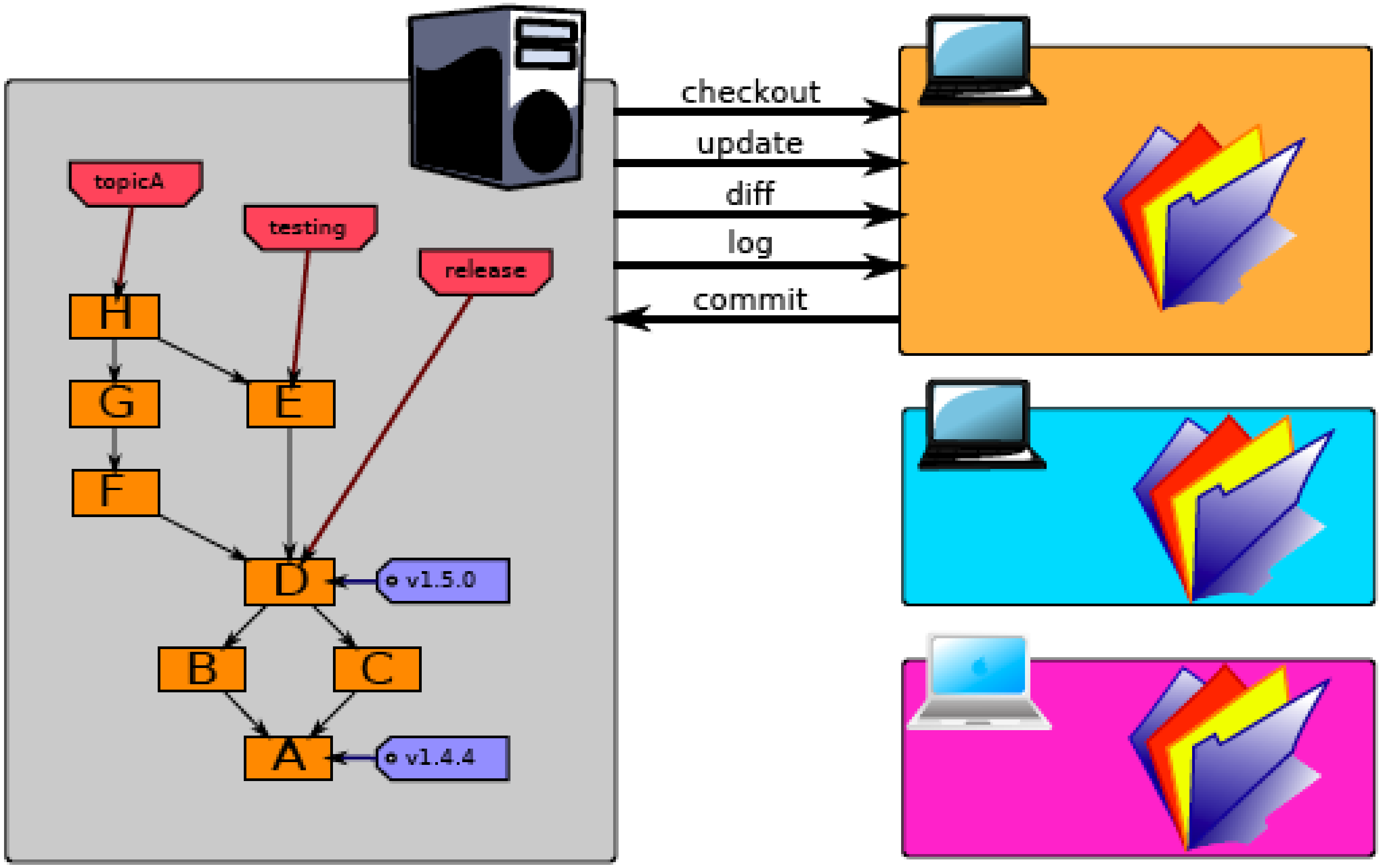
Distributed version control systems (DVCS)

- Usually one copy is sanctioned as main development branch
- Different branches are merged based on the "reliability rank" of the committer
  - Based on the committer's history of how reliable a committer he/she's been
  - Merging is usually not painful
  - Can do selective merge of some parts only
- Joining the project is easy
  - No need for formal approval
  - New project members have to work their ways up the committers' hierarchy
- Working off-line less risky

Examples of distributed VCS

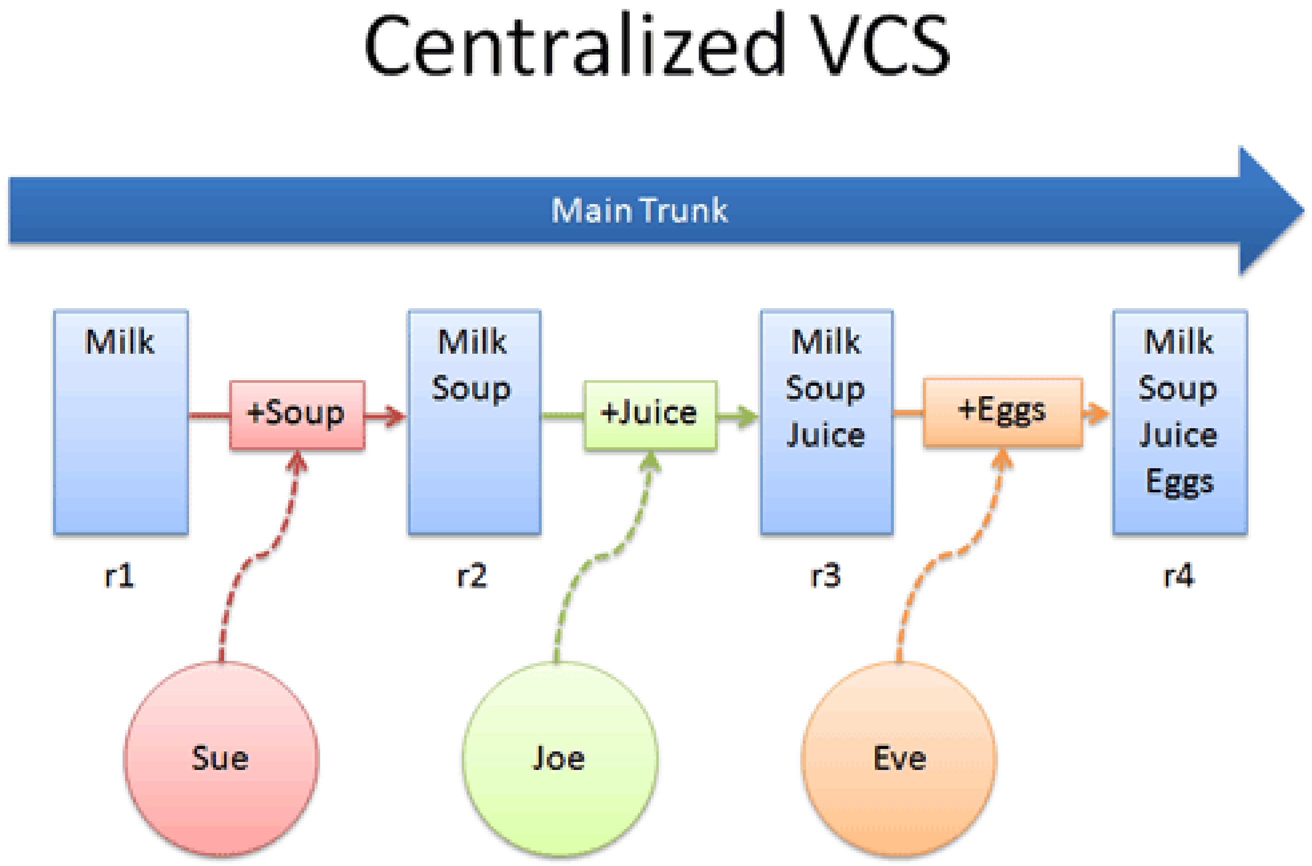
- Bitkeeper (proprietary since 2005)
- Git (Linus Torvalds, ~2005, free)
- Mercurial (~ 2005, free)

Centralized VCS



Git the basics, Bart Trojanowski, http://excess.org/article/2008/07/ogre-git-tutorial/

Workflow centralized VCS



http://betterexplained.com/articles/intro-to-distributed-version-control-illustrated/

Operations

Bootstrap
- init
- checkout
- switch branch

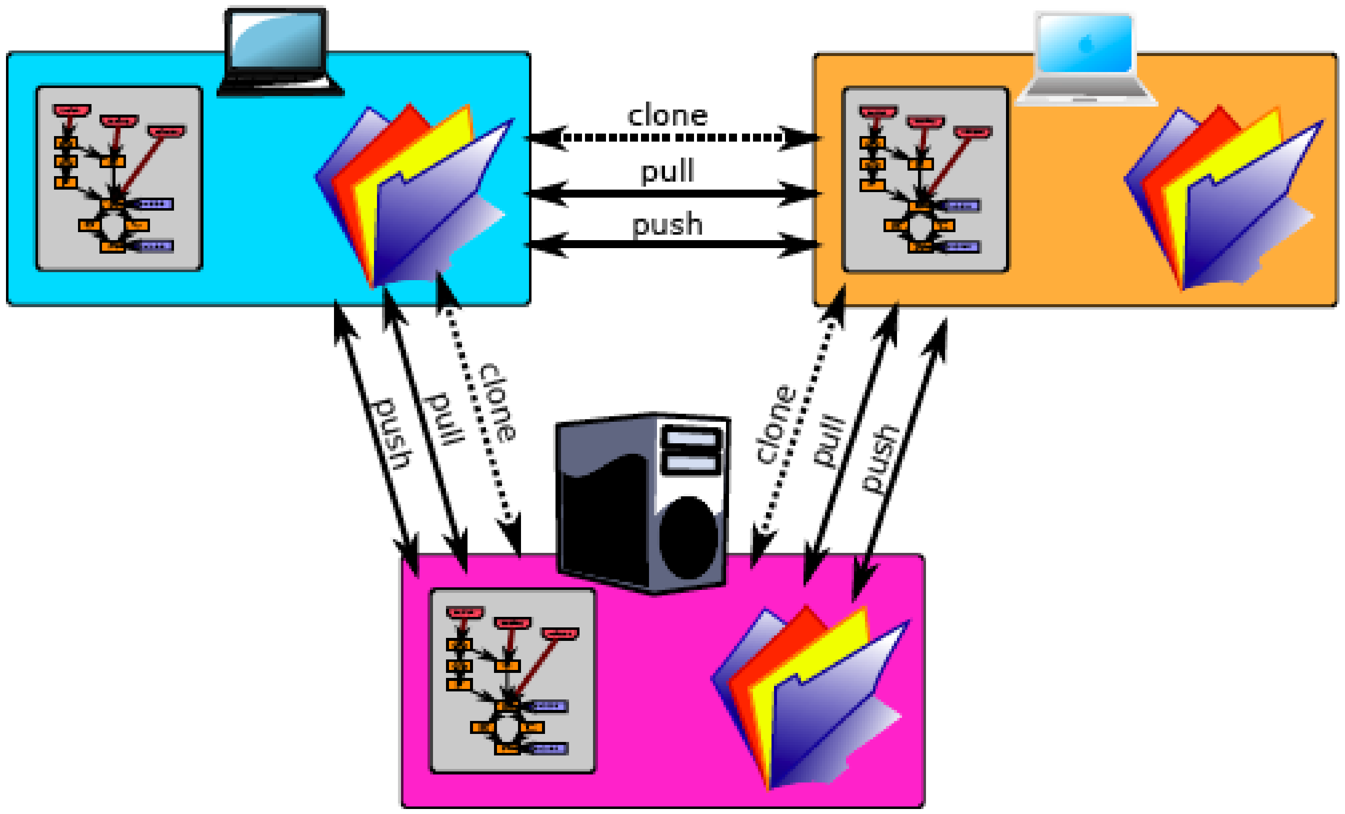
Modify
- add, delete, rename
- commit

Information
- status
- diff
- log

Reference
- tag
- branch

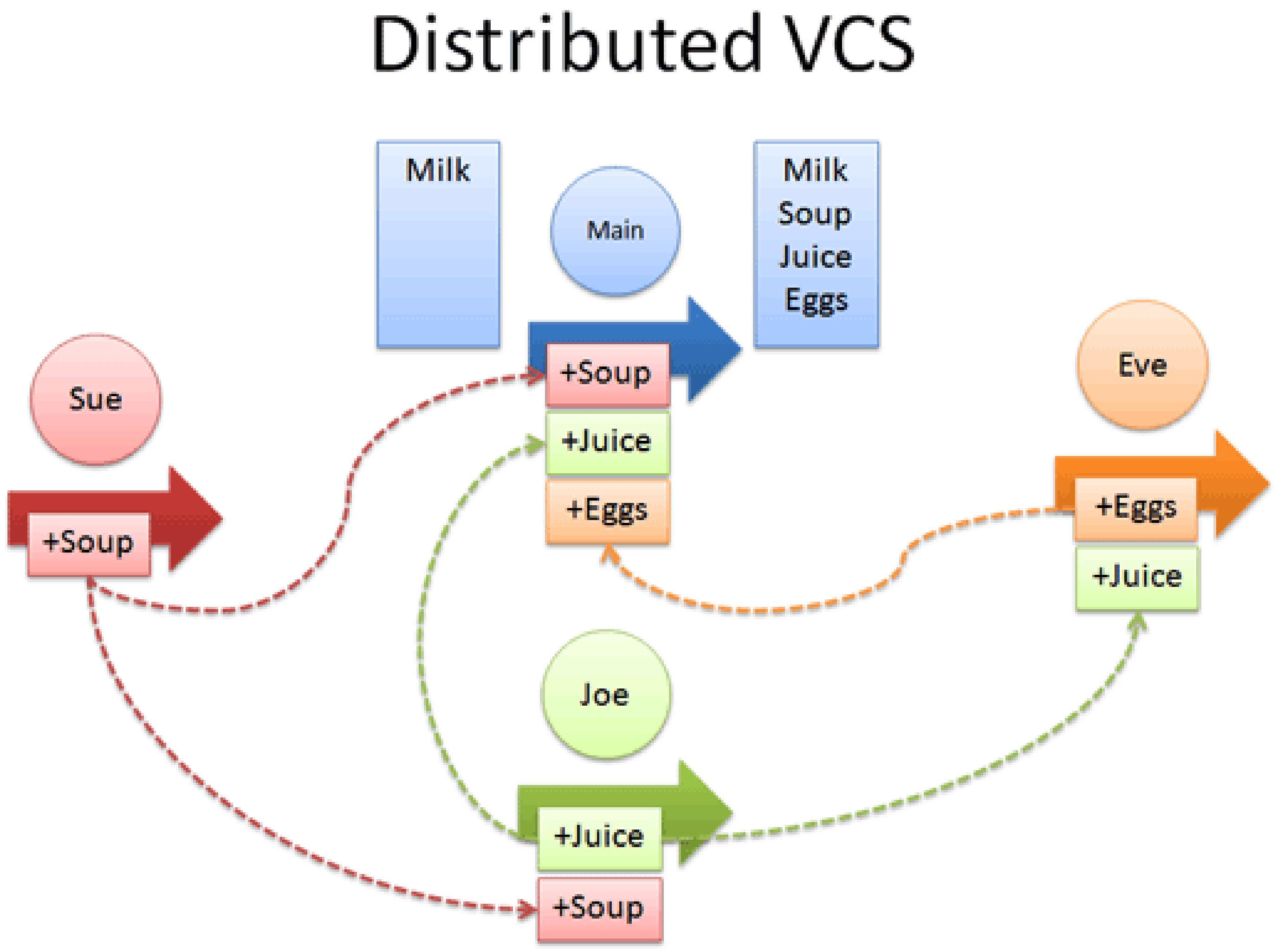
Decentralized
- clone
- pull, fetch
- push

Distributed VCS: peer-to-peer architecture



Git the basics, Bart Trojanowski, http://excess.org/article/2008/07/ogre-git-tutorial/

Workflow distributed VCS



http://betterexplained.com/articles/intro-to-distributed-version-control-illustrated/

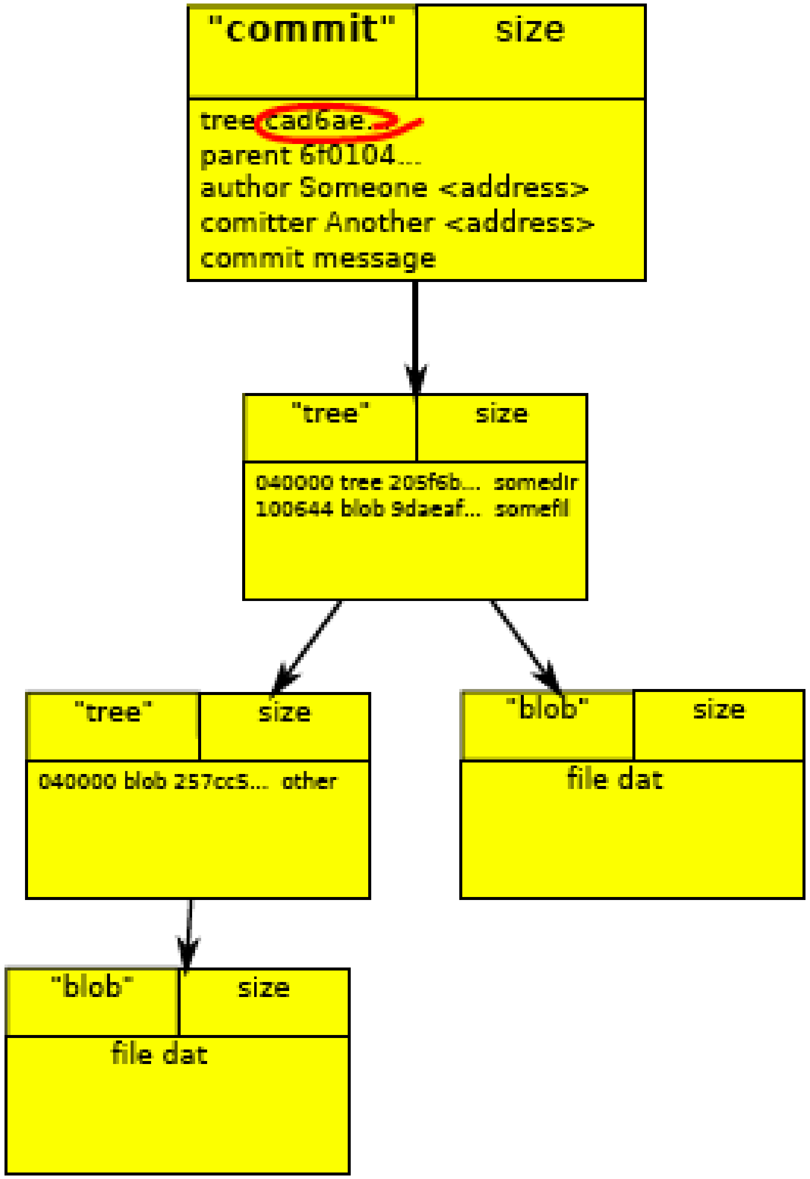
Git: main features

- Radically embraces the distributed model of VCS
  - heavy branch/merge
  - multiple merge algorithms
  - no explicit storage of revision history
    - change history recovered dynamically
    - a file rename is a change like any other
- Performance is a main concern
  - scalability to large projects, with many developers
- Resilience to errors and malicious attacks
- Modular design
  - Collection of tools similar to GNU/Linux systems
- "Packaged" files
  - store multiple objects in files efficiently

Objects in Git

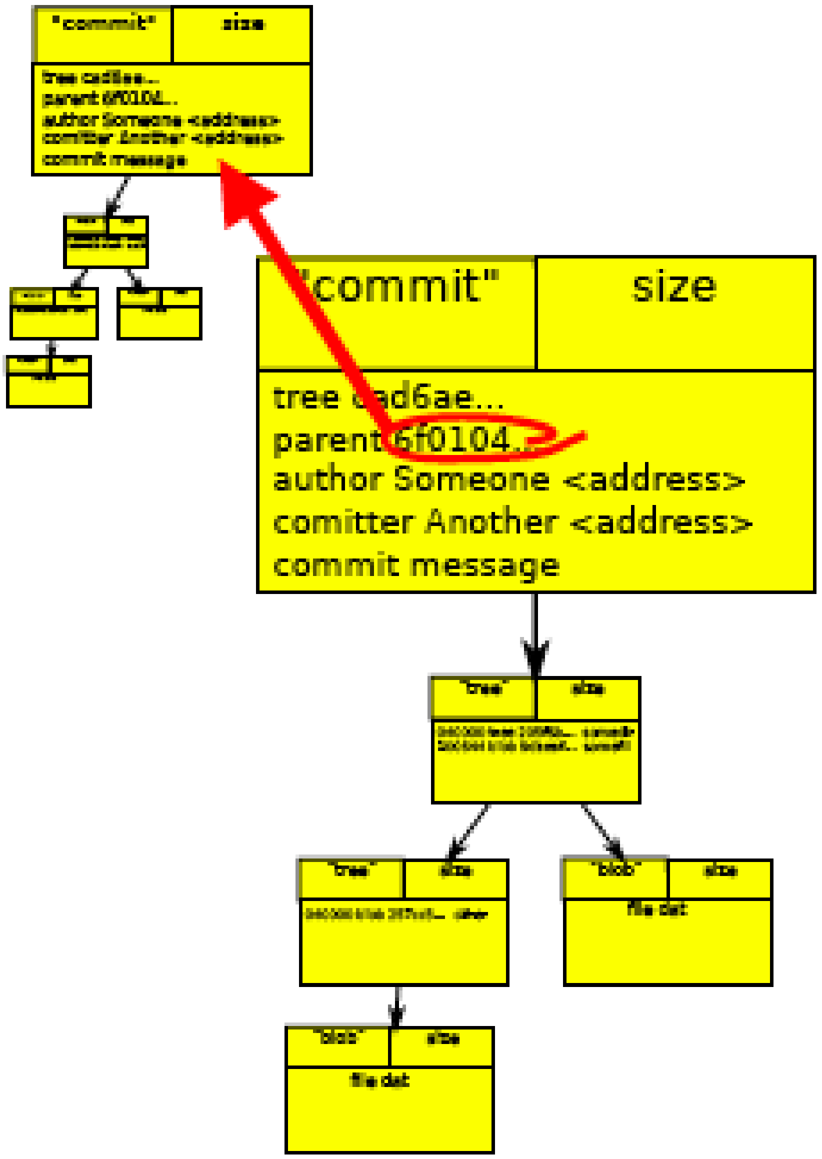
Object types:

- Blob
  - content of a file
- Tree
  - directory structure
- Commit
  - tree + history
- Tag
  - reference and label to another container

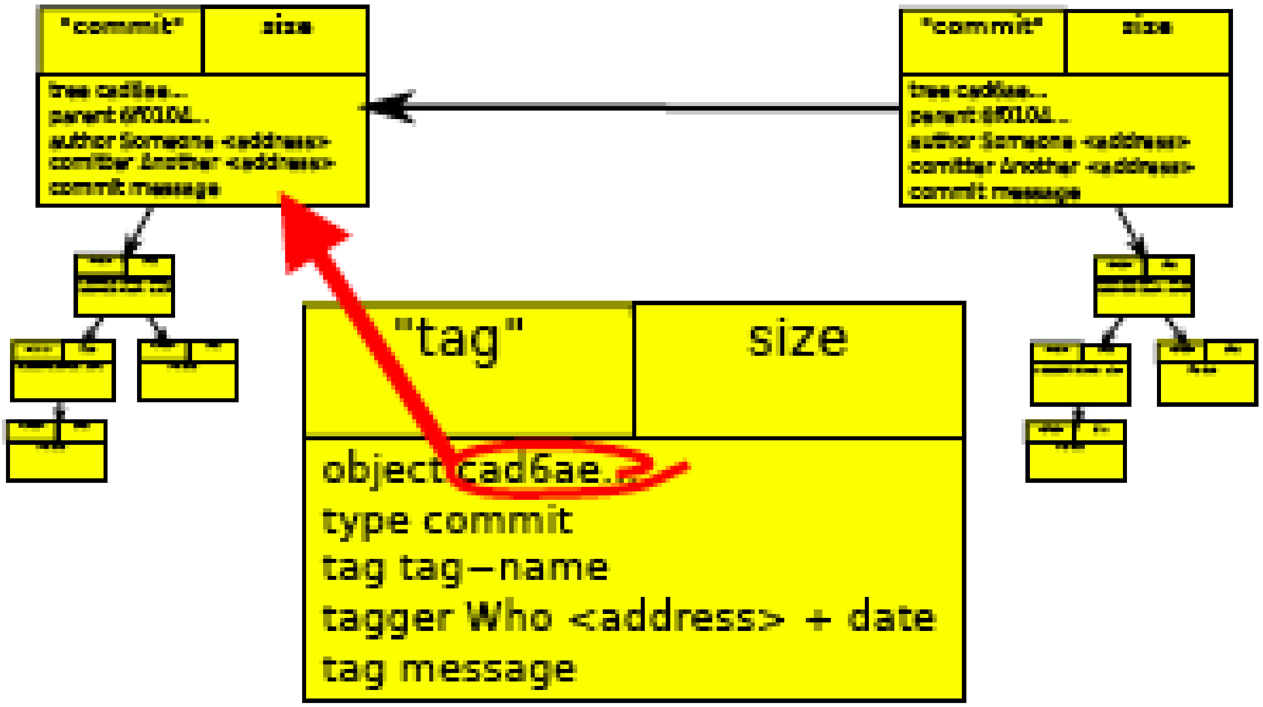


Git the basics, Bart Trojanowski, http://excess.org/article/2008/07/ogre-git-tutorial/

Objects in Git

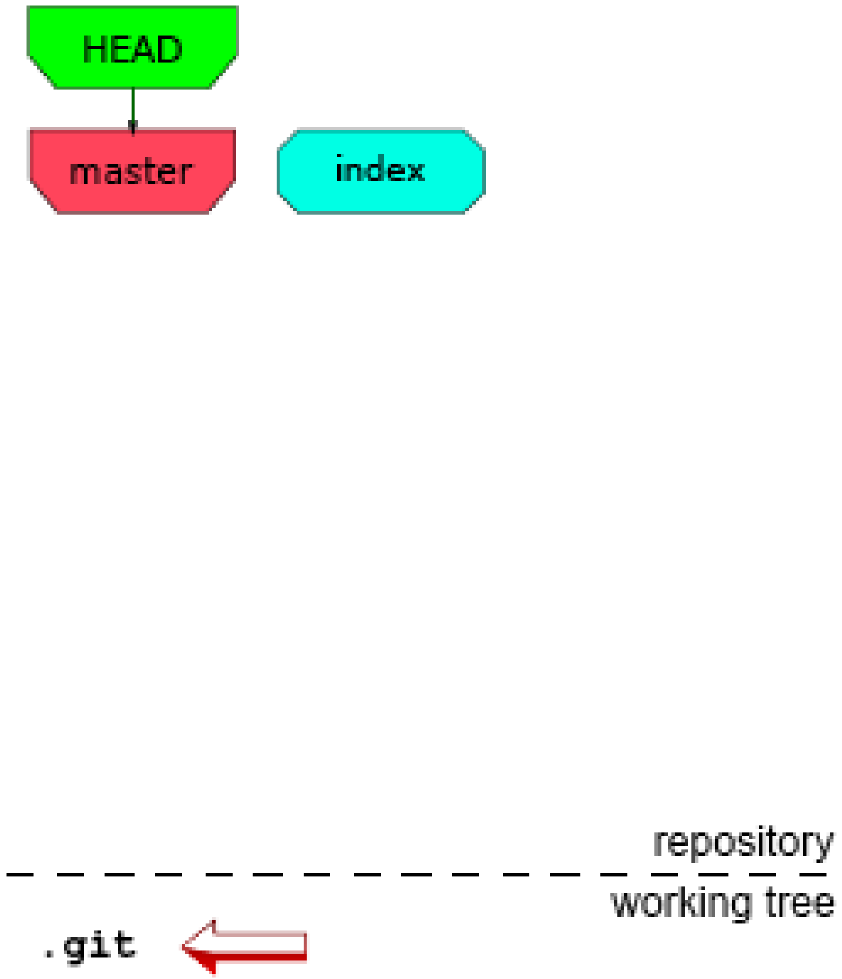


Objects in Git



Git example – bootstrapping

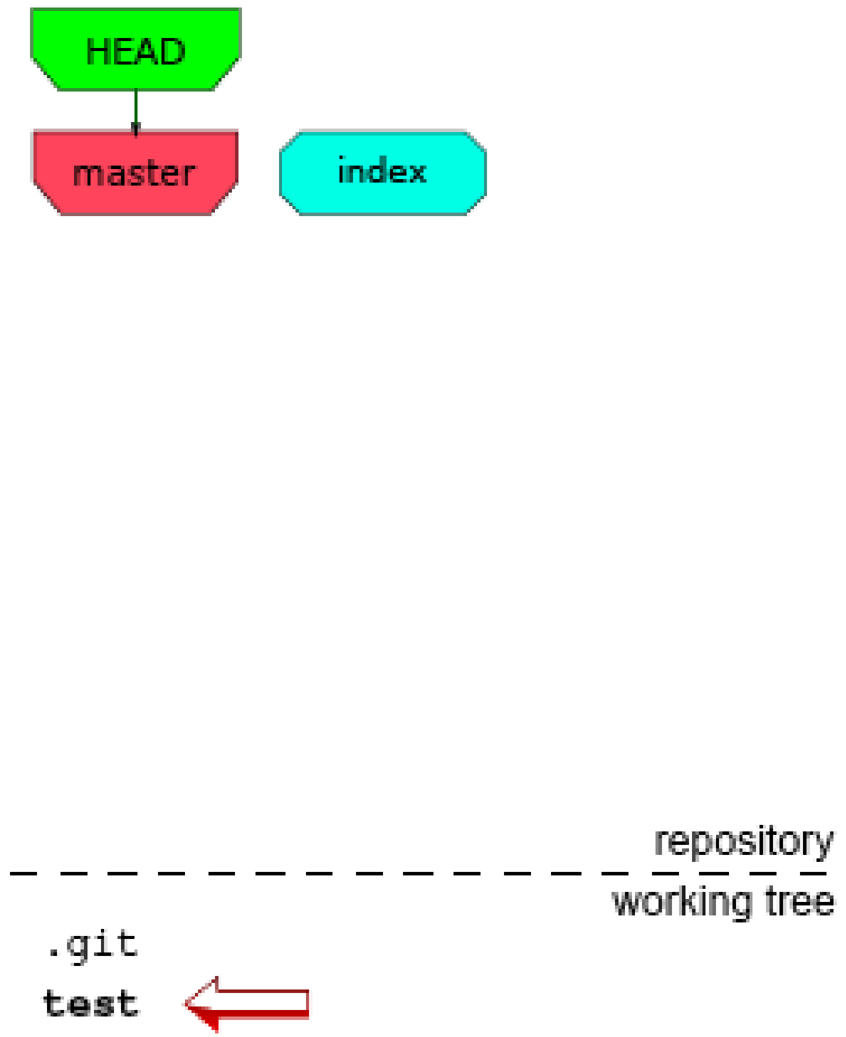
```
$ mkdir project
$ cd project
$ git init
```



Git the basics, Bart Trojanowski, http://excess.org/article/2008/07/ogre-git-tutorial/

Git example – bootstrapping

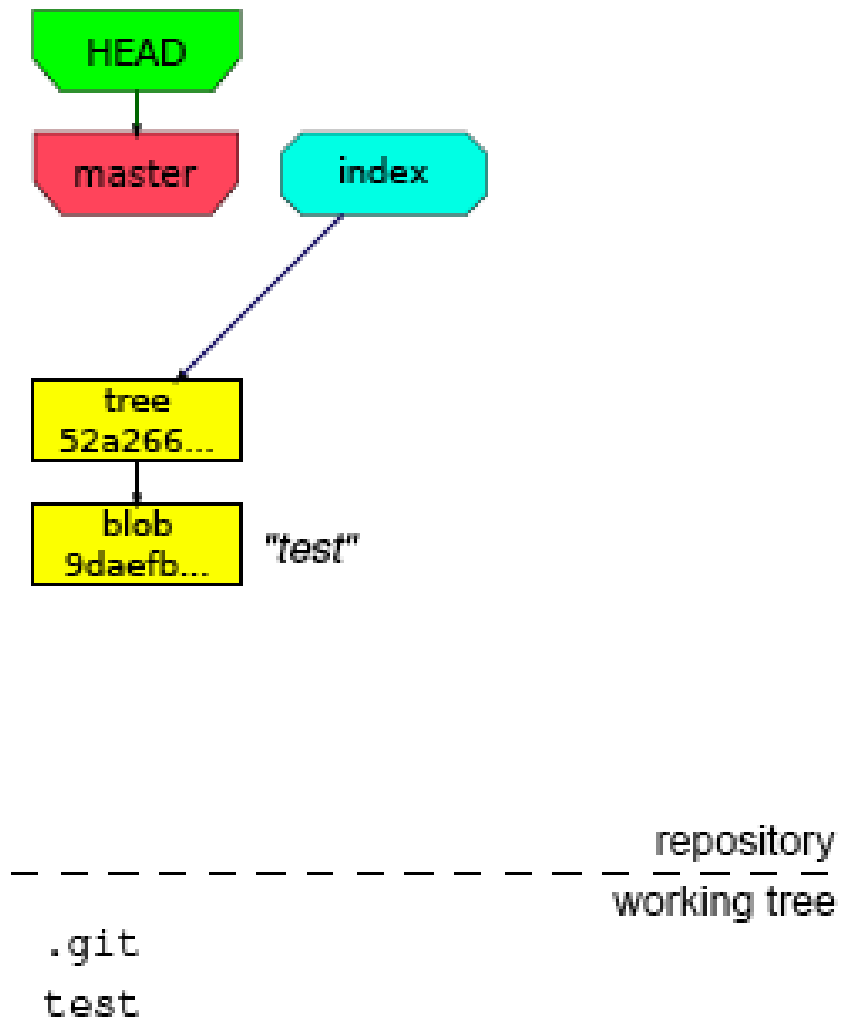
```
$ echo test > test
```



Git the basics, Bart Trojanowski, http://excess.org/article/2008/07/ogre-git-tutorial/

Git example – do work

```
$ echo test > test
$ git add test
```



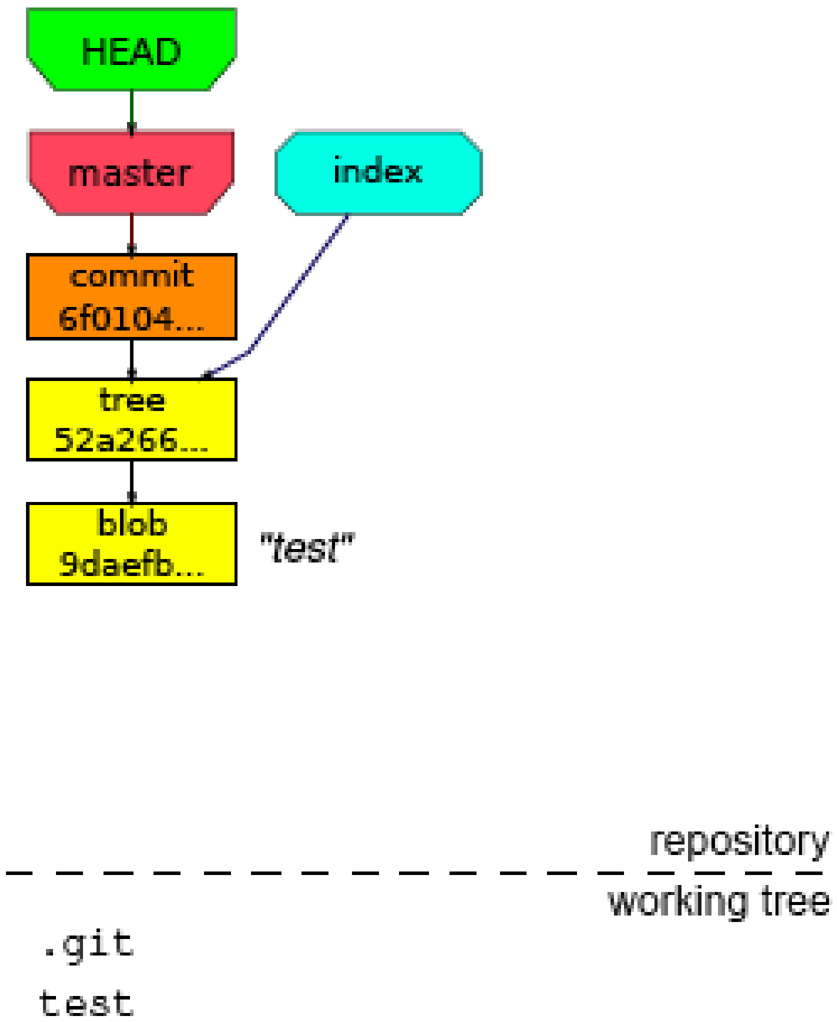
Git the basics, Bart Trojanowski, http://excess.org/article/2008/07/ogre-git-tutorial/

Git example – commit

```
$ echo test > test
$ git add test


$ git commit -m‘‘test’’
Created initial commit 6f01040:  test
1 files changed, 1 insertions(+),
0 deletions(-)

create mode 100644 test
```



Git the basics, Bart Trojanowski, http://excess.org/article/2008/07/ogre-git-tutorial/

Remote repositories

- To get a local copy

```
git clone remote_url
```

- To fetch and merge remote changes

```
git pull
```

- To publish local changes to remote repo

```
git pull
```

→ Always pull before push

Advantages/disadvantages of DVCS

Advantages:

- Everyone has a local sandbox
- Works offline
- Fast diffs, commits, reverts
- Handles changes well
- Easy branching and merging
- Less management (no server)

Disadvantages:

- Not a backup
- No real "latest version"
- No real revision numbers
  - (you can still use change numbers or tags)

Three parts of SCM

2. Build management systems

Build activities

- Compiling and linking multi-package application
- Creating installers
- Related activities
  - deployment
  - release documentation
  - ...

Build activities

Compiling and linking multi-package application

- manually invoke the compiler
- compilation scripts
- build automation

Make

Make (first release: 1977, Stuart Feldman)

- call compiler/linker in right order according to dependencies
- incremental compilation of the minimal number of modules changed
- a Makefile specifies the targets (files to be compiled) and the dependencies

Automake

- generate platform-specific Makefiles from a higher-level description

Make-like tools

- e.g.: Apache Ant for Java

Makefile basic structure

```
target1:        component1.1   component1.2   component1.3 ...
        command1.1 to build target 1
        command1.2 to build target 1
        ...

target2:        component2.1   component2.2
        command2.1 to build target 2
        command2.2 to build target 2
        ...
```

components can include references to other targets

first target built by default

Makefile: example

```
application:  foobar.c
    gcc foobar.c -o application

foobar.txt: foo.txt bar.txt
    cat foo.txt bar.txt > foobar.txt

foobar.c: foobar.txt
    mv foobar.txt foobar.c
```

Make in action

- `$> make targetx`

- Semantics:
  - Recursively make all the other targets targetx depends upon
  - If any of targetx's dependencies are newer than targetx or if targetx doesn't exist
    - (re)generate targetx by executing the associated commands
  - Otherwise
    - done

Binary builds

There are different levels of binary builds:

- Daily (Nightly) Builds
- Integration Builds
- Stable Builds
- Releases
  - Alpha Release
  - Beta Release
  - Release Candidate
  - Official Release

current, but not reliable

$\updownarrow$

not current, but reliable

Three parts of SCM

3. Bug and issue tracking systems

Bug tracking

Bug-Tracking is the term for an infrastructure that captures and manages bug-reports in a given project.

The number and quality of bugs is normally a judgment for the release-quality of software.

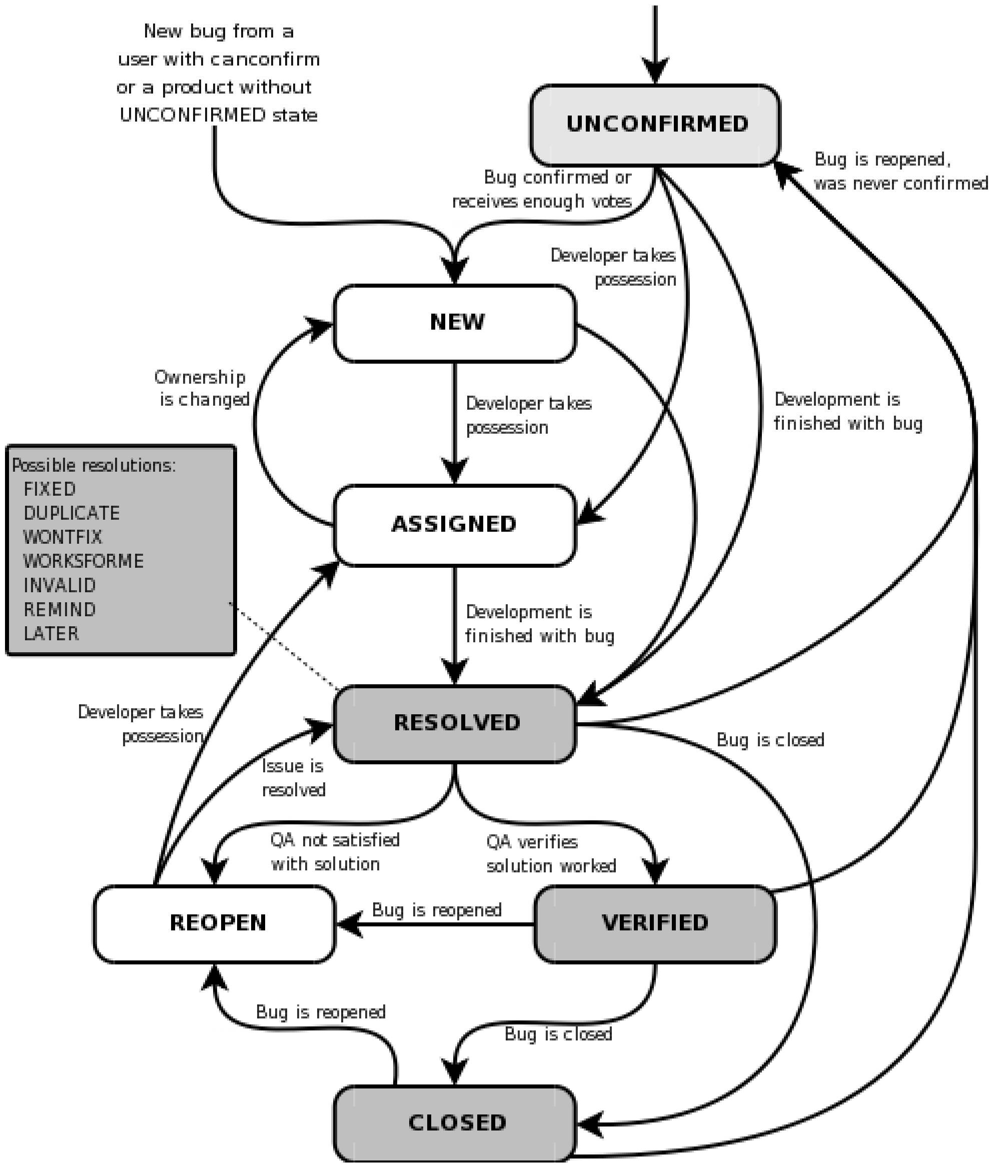
Bugzilla

Web-based bug tracking system

- First release: 1998 (open source, for Mozilla project)
- Written in Perl (originally in Tcl)

- Tracks software defects
  - when they are first acknowledged
  - who is responsible for them
  - when they are fixed
  - ...

A Bugzilla bug's life


New bug from a user with canconfirm or a product without UNCONFIRMED state
UNCONFIRMED
Bug confirmed or receives enough votes
Bug is reopened, was never confirmed
Developer takes possession
NEW
Ownership is changed
Developer takes possession
Development is finished with bug
Possible resolutions:
FIXED
DUPLICATE
WONTFIX
WORKSFORME
INVALID
REMIND
LATER
ASSIGNED
Development is finished with bug
Developer takes possession
RESOLVED
Issue is resolved
Bug is closed
QA not satisfied with solution
QA verifies solution worked
REOPEN
Bug is reopened
VERIFIED
Bug is reopened
Bug is closed
CLOSED

From the official Bugzilla documentation

Three parts of SCM

1. Version control systems
   a) Local version control
   b) Centralized version control
   c) Distributed version control
2. Build management systems
3. Bug and issue tracking systems

CM Terminology

- Repository, Commit, Update, Checkout, Head
- Branch/Fork, Merge
- Conflicts, Resolving Conflicts
- Tag, Label
- Pull, push, fetch
- Release, Build
- Test Cases, Test Suite, Regression Tests

Tools for CM

Software Configuration Management (SCM) Tools

- RCS, CVS, Subversion, Git, Mercurial, Monotone, ArK, tla (FreeSoftware)
- ClearCase, BitKeeper, SourceSafe (Commercial)

Bug-Tracking Tools

- Bugzilla (FreeSoftware)
- Origo issue tracker

Build Tools

- Make (FreeSoftware)
- Xenofarm (FreeSoftware)
- Tinderbox (FreeSoftware)