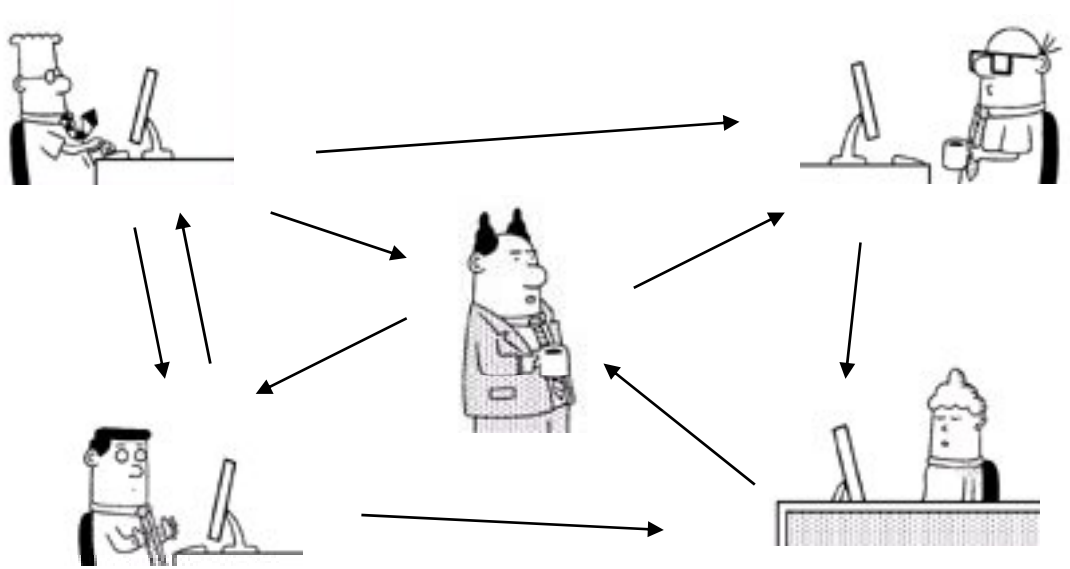
# Revision control systems (RCS)

and

# Subversion

# Problem area

- Software projects with multiple developers need to coordinate and synchronize the source code

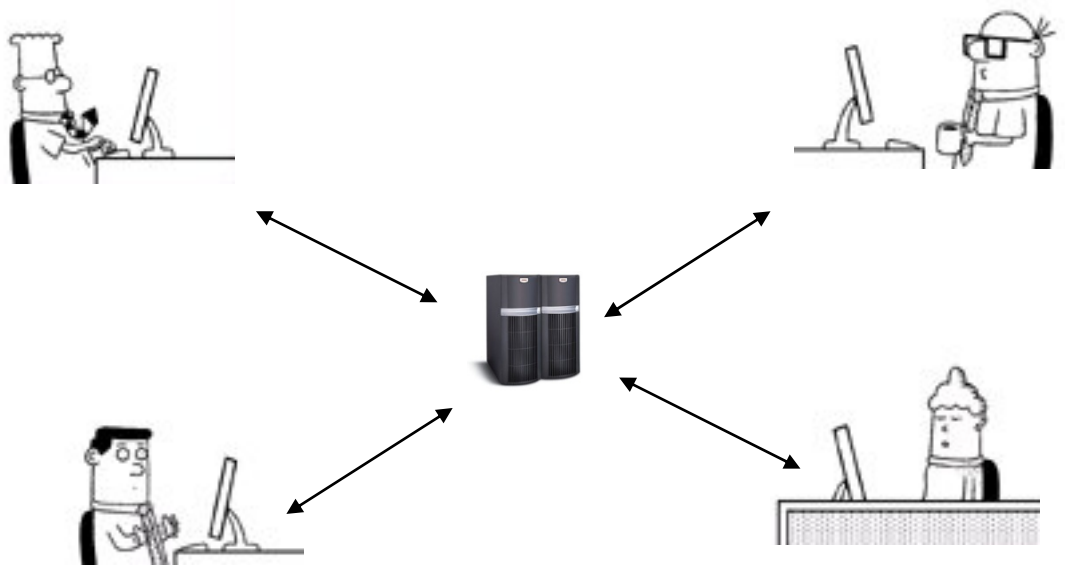


# Approaches to version control

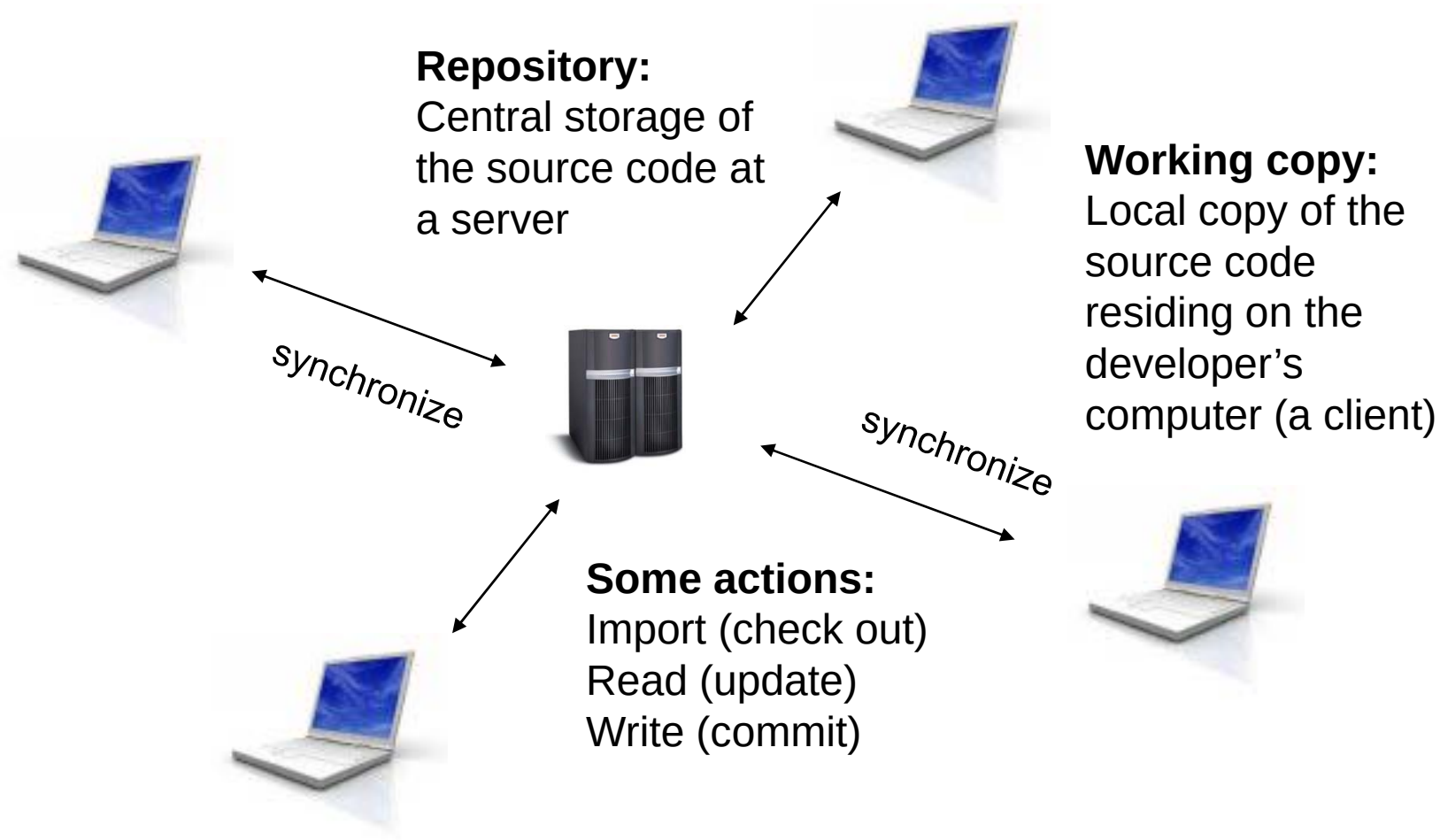
- Work on same computer and take turns coding
  - Nah...
- Send files by e-mail or put them online
  - Lots of manual work
- Put files on a shared disk
  - Files get overwritten or deleted and work is lost, lots of direct coordination
- In short: Error prone and inefficient

# The preferred solution

- Use a revision control system (like Subversion)
- RCS - software that allows for multiple developers to work on the same codebase in a coordinated fashion
- Can manage any sorts of files
- Alternatives are Bazaar, Git, Mercurial

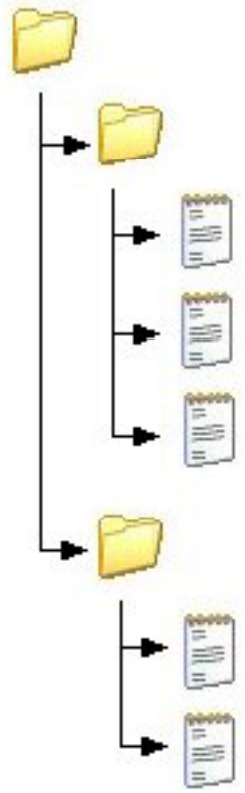


# How it works


**Repository:**
Central storage of the source code at a server
**Working copy:**
Local copy of the source code residing on the developer's computer (a client)
synchronize
synchronize
**Some actions:**
Import (check out)
Read (update)
Write (commit)

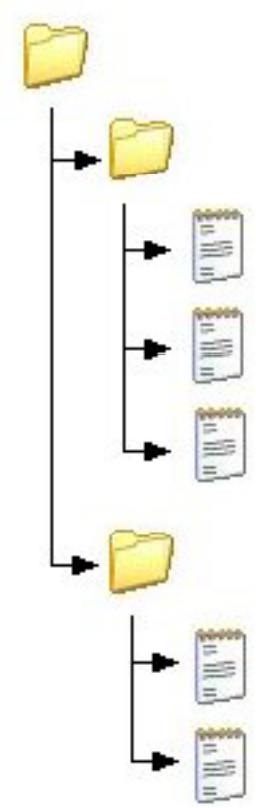
# The repository

- A central store of data
- Stores information in a virtual filesystem tree
- Remembers every change ever written to it
- Clients can check out an independent, private copy of the filesystem called a *working copy*
- Clients connect to the repository and read or write to the filesystem



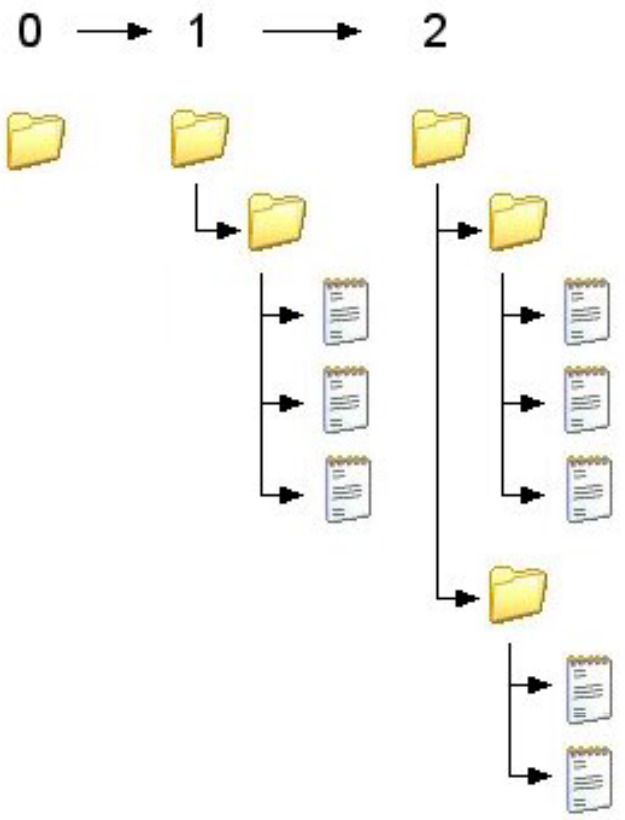
# Working copies

- Ordinary directory tree
- Each directory contains an administrative directory named *.svn*
- Changes are not incorporated or published until you tell it to do so
- A working copy corresponds to a subtree of the repository



# Revisions

- Every commit creates a new *revision*, which is identified by a unique revision number
- Every revision is remembered by the RCS and forms a revision history
- Every revision can be checked out independently
- The current revision can be roll-backed to any revision
- Commits are *atomic*



# Work cycle

**Initial check out:**
The developer checks out the source code from the repository

**1) Development:**
The developer makes changes to the working copy

Client

**2) Update:**
The developer receives changes made by other developers and syncronizes his local working copy with the repository

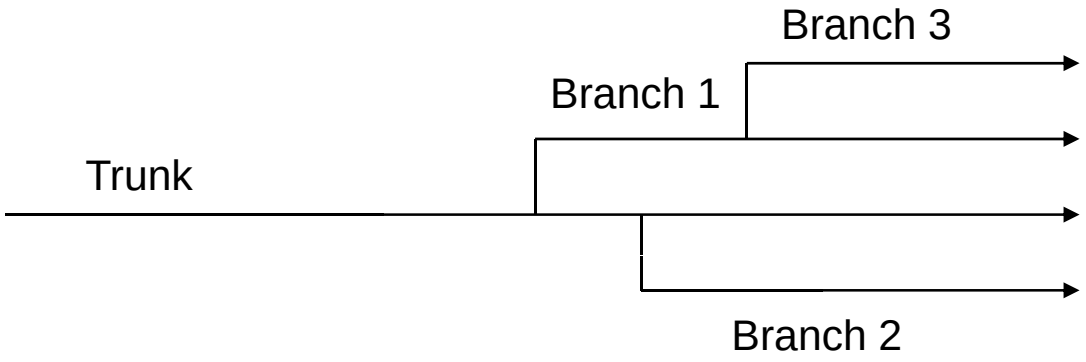
Repository

**3) Resolve conflicts:**
When a developer has made local changes that won't merge nicely with other changes, conflicts must be manually resolved

**4) Commit:**
The developer makes changes and writes or merges them back into the repository

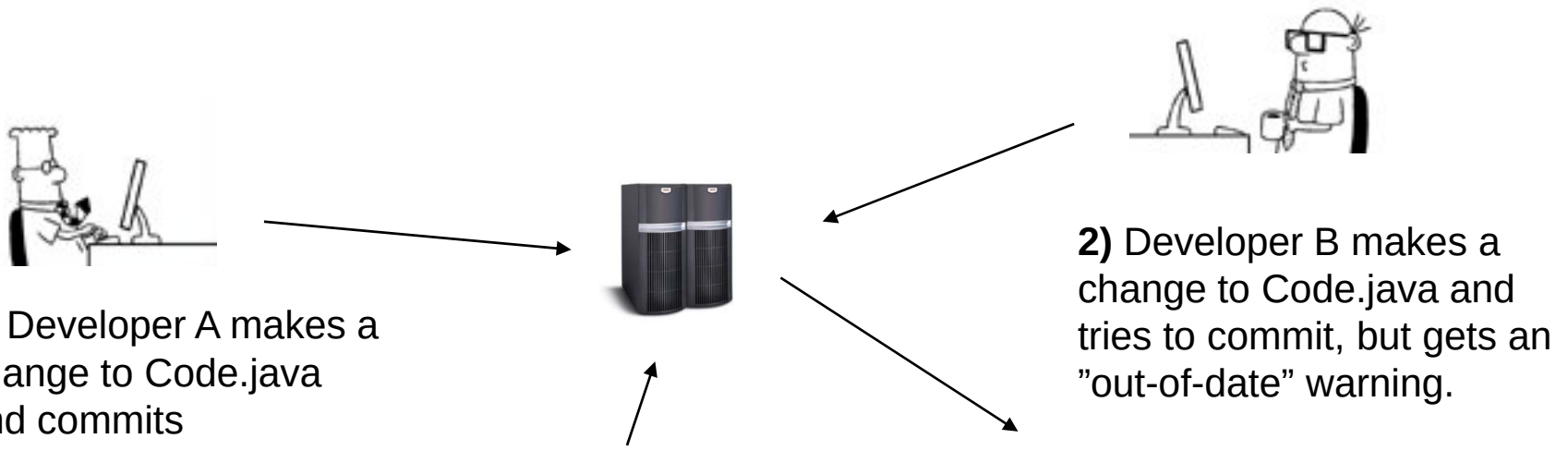
# Trunk and Branches

- Trunk is the original main line of development
- A branch is a copy of trunk which exists independently and is maintained separately
- Useful in several situations:
  - Large modifications which takes long time and affects other parts of the system (safety, flexibility, transparency)
  - Different versions for production and development
  - Customised versions for different requirements



# Conflicts

- Arises if several developers edit the same part of a file
- Solution in Subversion: ”Copy-modify-merge”



**1)** Developer A makes a change to Code.java and commits

**2)** Developer B makes a change to Code.java and tries to commit, but gets an ”out-of-date” warning.

**3)** Developer B updates his working copy. He will be noticed that Code.java is in a state of conflict.

**4)** Developer B edits and resolves the conflicts, and commits the file back in the repository

# Conflicts

- Changes that do not overlap are merged automatically
- 4 solutions are provided in conflict situations:
  - Use ”mine” version – the developers local copy
  - Use ”their” version – the copy in the repository
  - Use ”base” version – the file before you started editing
  - Use the original file with conflict markers and edit the conflict manually before comitting
- Subversion must be told that the conflict is *resolved*
  - Will remove the temporary files and let you commit

# Advantages of RCS

- Concurrent development by multiple developers
- Possible to roll-back to earlier versions if development reaches a dead-end
- Allows for multiple versions (branches) of a system
- Logs useful for finding bugs and monitoring the development process
- Works as back-up

# Good practises

- Update, build, test, *then* commit
  - Do not break the checked in copy
- Update out of habit before you start editing
  - Reduce your risk for integration problems
- Commit often
  - Reduce others risk for integration problems
- Check changes (diff) before committing
  - Don’t commit unwanted code in the repo
- Do not use locking
  - Obstructs collaboration

# What to add to the repository

- Source code including tests
- Resources like configuration files

- What to *not* add:
  - Compiled classes / binaries (target folder)
  - IDE project files
  - Third party libraries

- Add sources, not products (generated files)!

# Subversion online commands

- Checkout a working copy:
  - $ svn checkout http://svn.example.com/scm
- Update a working copy:
  - $ svn update
- Commit your changes:
  - $ svn commit –m ”a log message”
- Create a branch
  - $ svn copy http://svn.example.com/scm/trunk http://svn.example.com/scm/branches/my-branch

# Subversion offline commands

- Add a file to the working copy:
  - $ svn add Code.java
- Delete a file from the working copy:
  - $ svn delete Code.java
- Move a file:
  - $ svn move Code.java dir/Code.java
- Compare working copy with repository on file-level:
  - $ svn status
- Compare working copy with repository on code-level:
  - $ svn diff
- Revert a file to the state from last commit
  - $ svn revert Code.java

# Create a repository

/home/projects $ **svnadmin create assignment1**

/home/projects/assignment1

Repository

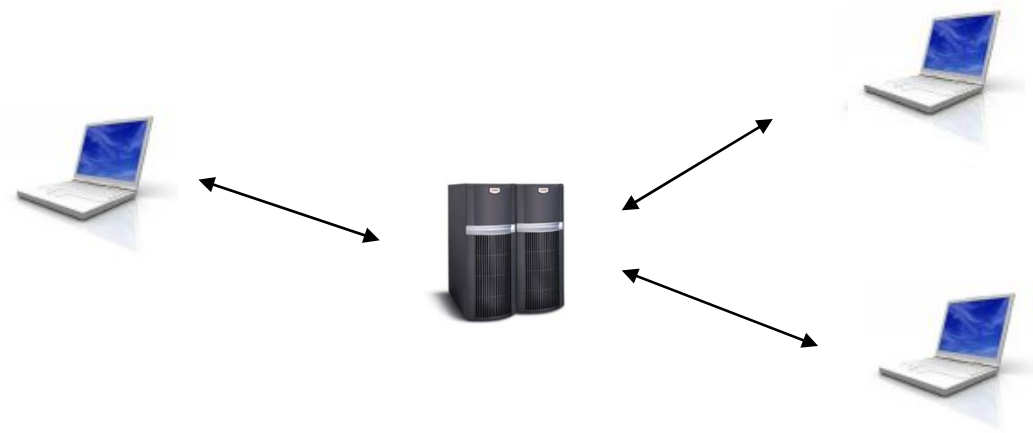
/myhome/assignment1 $ **svn checkout svn+ssh://username@ svn.server.url/home/projects/assignment1**

/myhome/assignment1/...

Client

# Summary

- Revision control systems enable multiple developers to work on the same code base
- Subversion uses a client/server system with a repository and working copies
- Every commit generates a new revision, which can be checked out independently
- Projects have a trunk version and might have multiple branches



# Resources

- ”Version control with Subversion”
  - Free PDF book online
  - http://svnbook.red-bean.com/
- Subversion home page
  - http://subversion.tigris.org/
- Subversion help command
  - $ svn help <command>
- TortoiseSVN – Graphical user interface for Subversion
  - http://tortoisesvn.tigris.org