Configuration Management

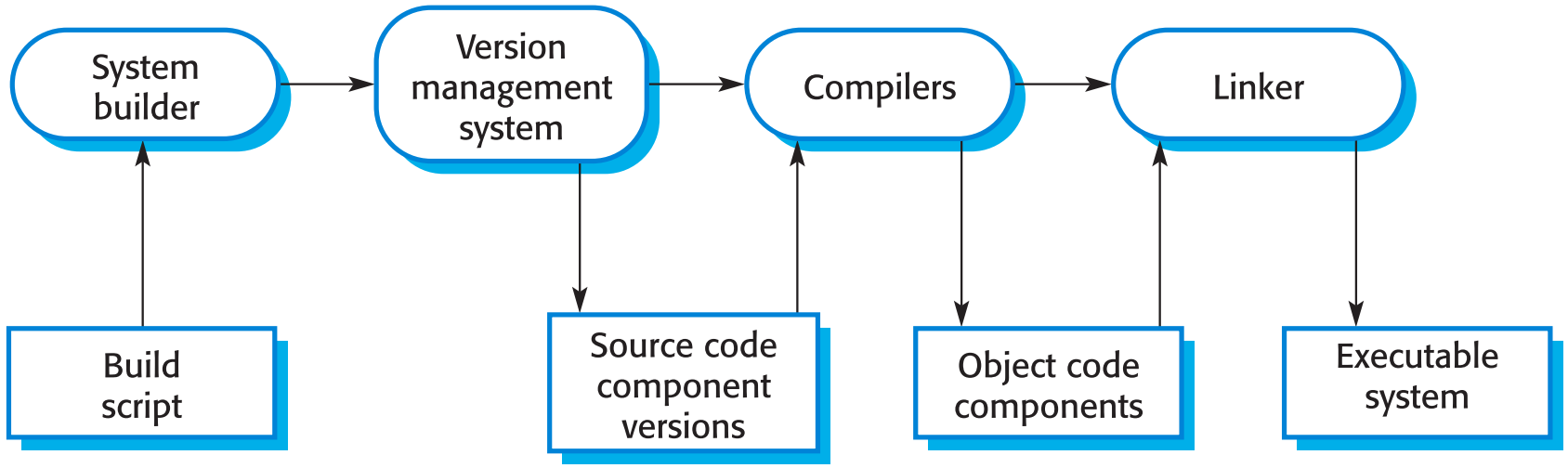
Luciano Baresi
Politecnico di Milano

Credits: Leonardo Mariani (University of Milano Bicocca)

CM: Some definitions

- A discipline applying technical and administrative direction and surveillance to: identify and document the functional and physical characteristics of a configuration item, control changes to those characteristics, record and report change processing and implementation status, and verify compliance with specified requirements. [IEEE Standard Glossary]
- On any team project, a certain degree of confusion is inevitable. The goal is to minimize this confusion so that more work can get done. The art of coordinating software development to minimize this particular type of confusion is called configuration management. Configuration management is the art of identifying, organizing, and controlling modifications to the software being built by a programming team. The goal is to maximize productivity by minimizing mistakes. [W. Babich]

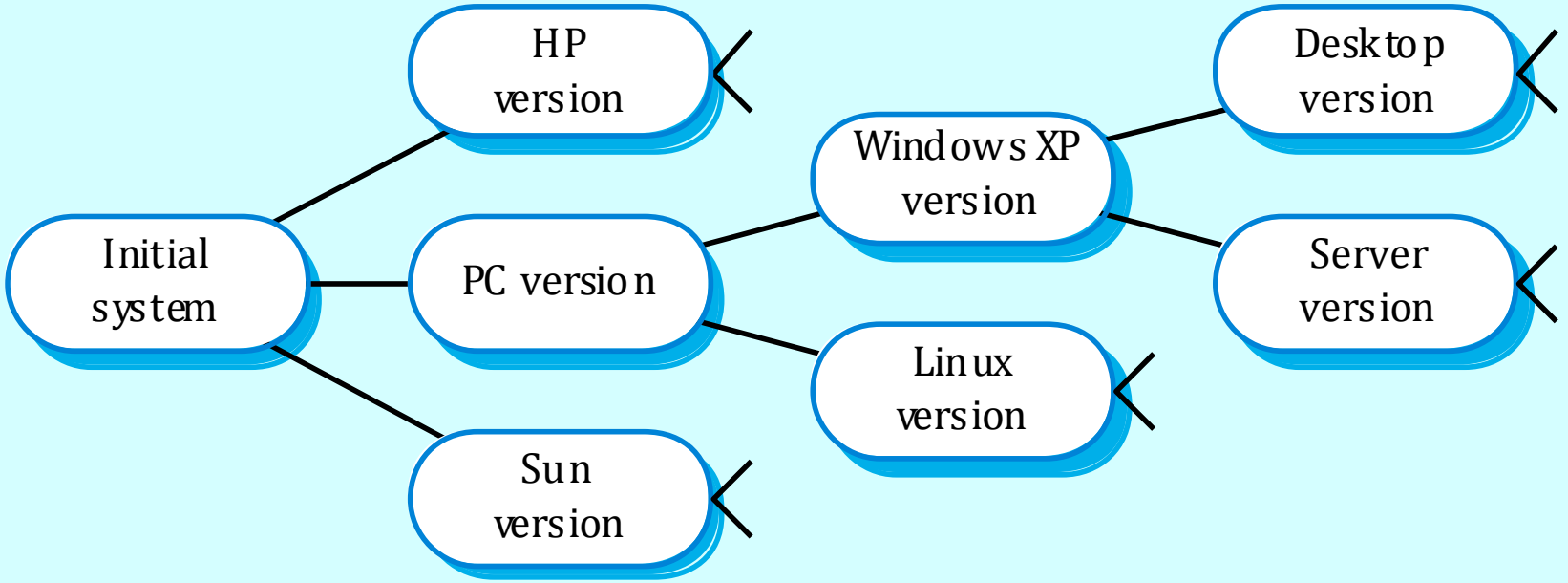
System building



Configuration Management

- CM is concerned with managing evolving software systems:
  - control the costs and effort
    - procedures + standards
    - part of the quality process

System families



from Sommerville “Software Engineering”

CM standards

- based on a set of standards which are applied within an organisation
  - how items are identified,
  - how changes are controlled
  - how new versions are managed
- Standards may be based on external CM standards (e.g., IEEE standard for CM)
- Products to be managed?
  - specifications, designs, programs, test data, user manuals…

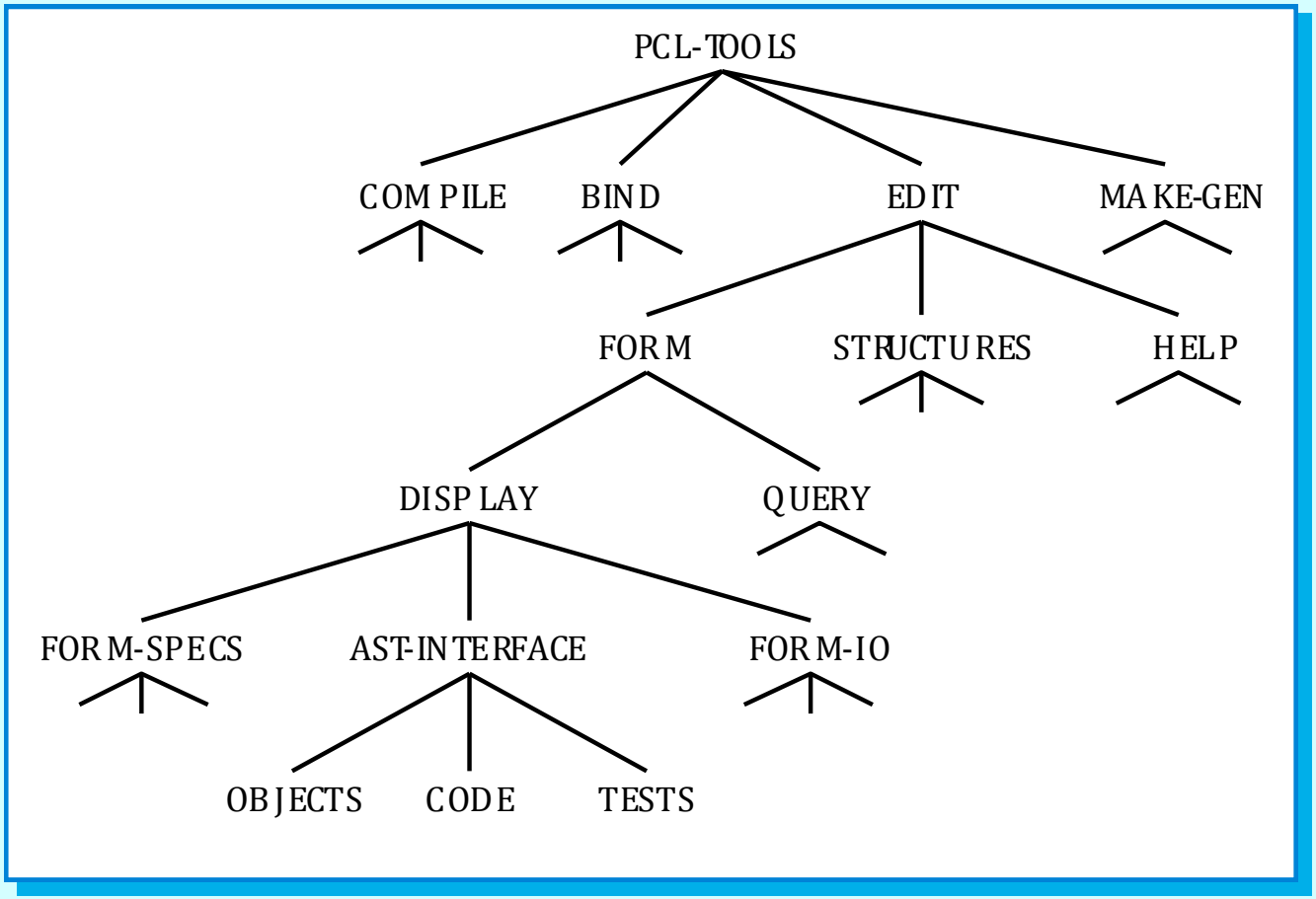
The CM plan

- Defines the types of documents to be managed and a document naming scheme
- Defines who takes responsibility for the CM procedures and creation of baselines
- Defines policies for change control and version management
- Defines the CM records which must be maintained
- Describes the tools which should be used to assist the CM process and any limitations on their use
- Defines the process of tool use
- Defines the CM database used to record configuration information

Configuration item identification

- Large projects typically produce thousands of documents which must be uniquely identified
- Some of these documents must be maintained for the lifetime of the software
- Document naming scheme should be defined so that related documents have related names
- A hierarchical scheme with multi-level names is probably the most flexible approach
  - PCL-TOOLS/EDIT/FORMS/DISPLAY/AST-INTERFACE/CODE

Configuration hierarchy



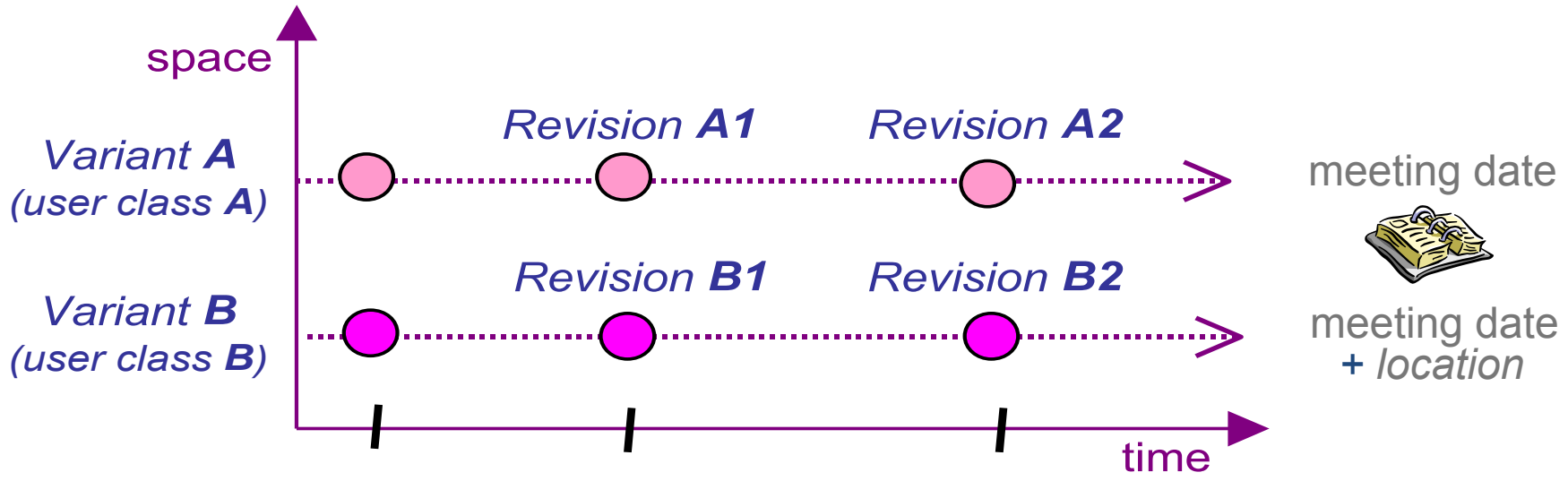
from Sommerville “Software Engineering”

Traceability

From A. van Lamsweerde
“Requirements Engineering”

Features, revisions, variants

- Feature = change unit
  - **functional/non-functional**: sets of functional/non-functional reqs
  - **environmental**: assumptions, constraints, work procedures, etc
- Feature changes yield new system version
  - **revision**: to correct, improve single-product version
  - **variant**: to adapt, restrict, extend multi-product version
    => commonalities + variations at variation points



A wide variety of changes

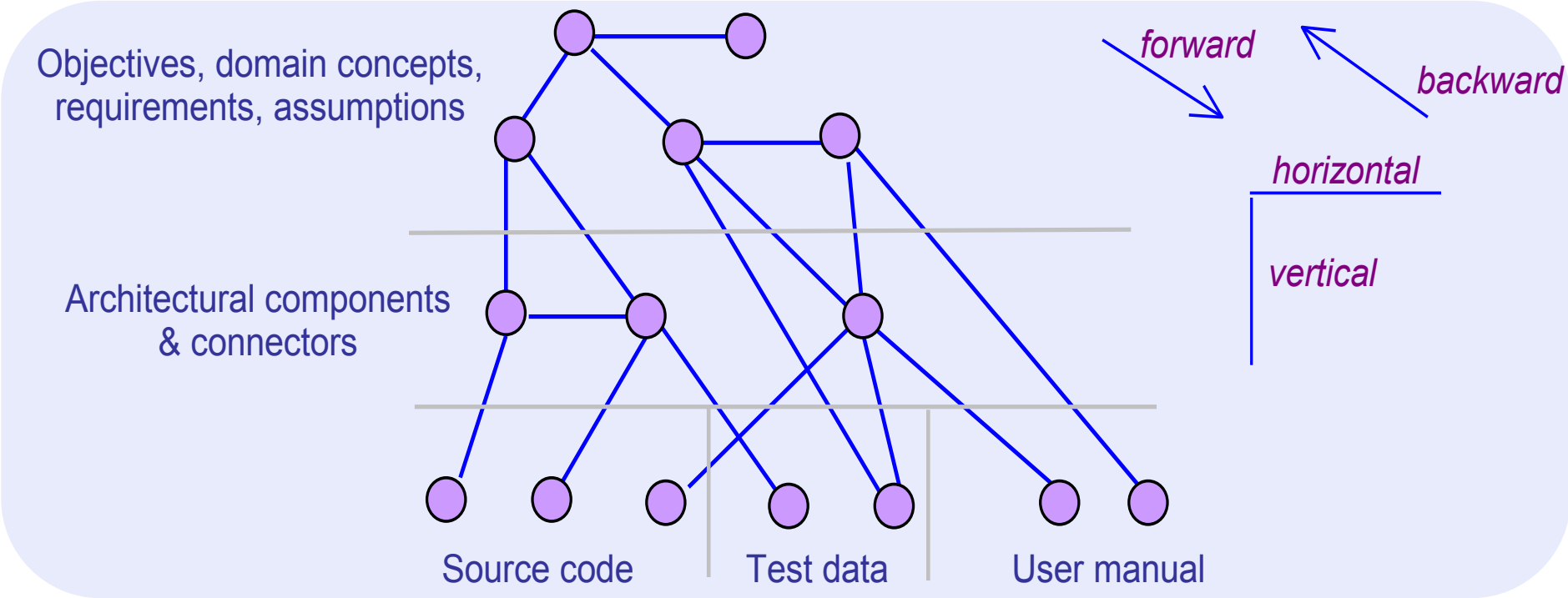
Cause	Change type	Version type
errors & flaws	corrective	revision
better understanding	corrective extension	revision
new functionality	extension	revision, variant
improved feature	ameliorative	revision
new users/usage	adaptative	variant
other ways of doing	adaptative	variant
new regulation	adaptative	revision
alternative regulation	adaptative	variant
organizational change	adaptative	revision
new priority/constraint	adaptative	revision

Evolution support requires traceability management

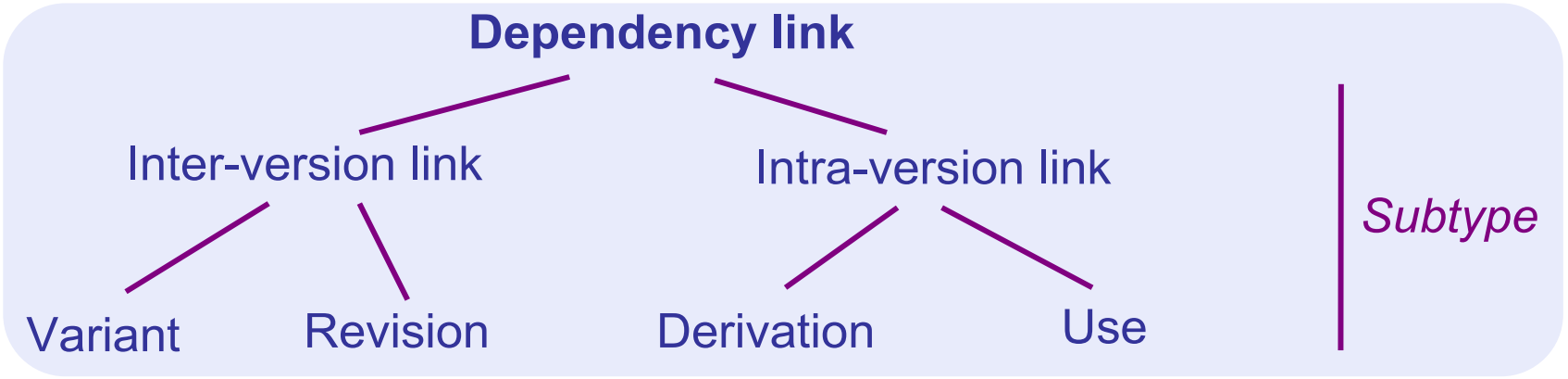
- An item is traceable if we can fully figure out
  - WHERE it comes from, WHY it is there
  - WHAT it will be used for, HOW it will be used
- Traceability management (TM), roughy
  - identify, document, retrieve the rationale & impact of items
- Objectives of traceability
  - assess impact of proposed changes
  - easily propagate changes to maintain consistency

TM relies on traceability links among items

- To be identified, recorded, retrieved
- Bidirectional: for accessibility from
  - source to target (forward traceability)
  - target to source (backward traceability)
- Within same phase (horizontal) or among phases (vertical)



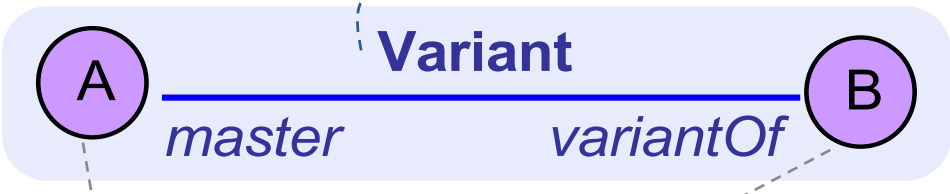
A taxonomy of traceability link types



What are the types of links traced by Configuration Management?

Inter-version traceability: *variant*, *revision* links

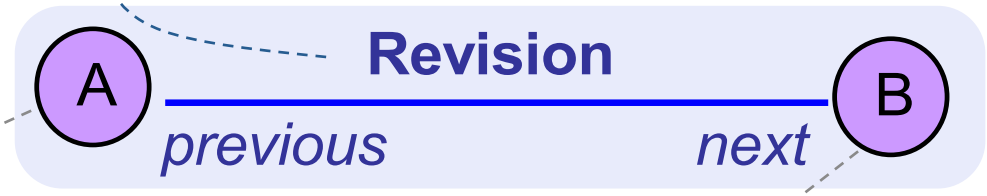
***B** has all features of **A** + specific ones*



common reqs for meeting scheduling

specific reqs for handling important participants

B overrides features of A, adds/removes some, keeps all others



reqs for optimal date to fit exclusion constraints

reqs for optimal date to fit exclusion & preference constraints

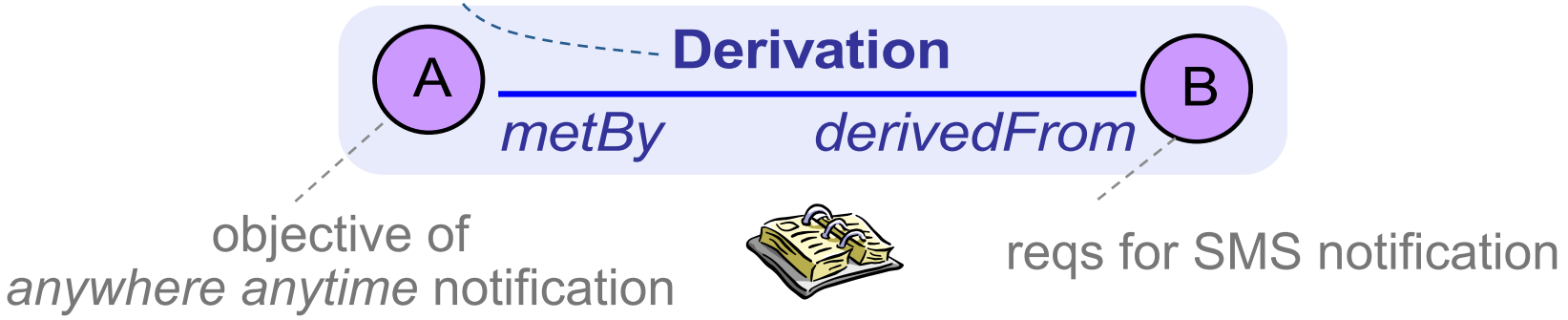
+ link annotation with configuration management info: *date, author, status, rationale*

Our focus: how to (semi-)automatically trace inter-version links

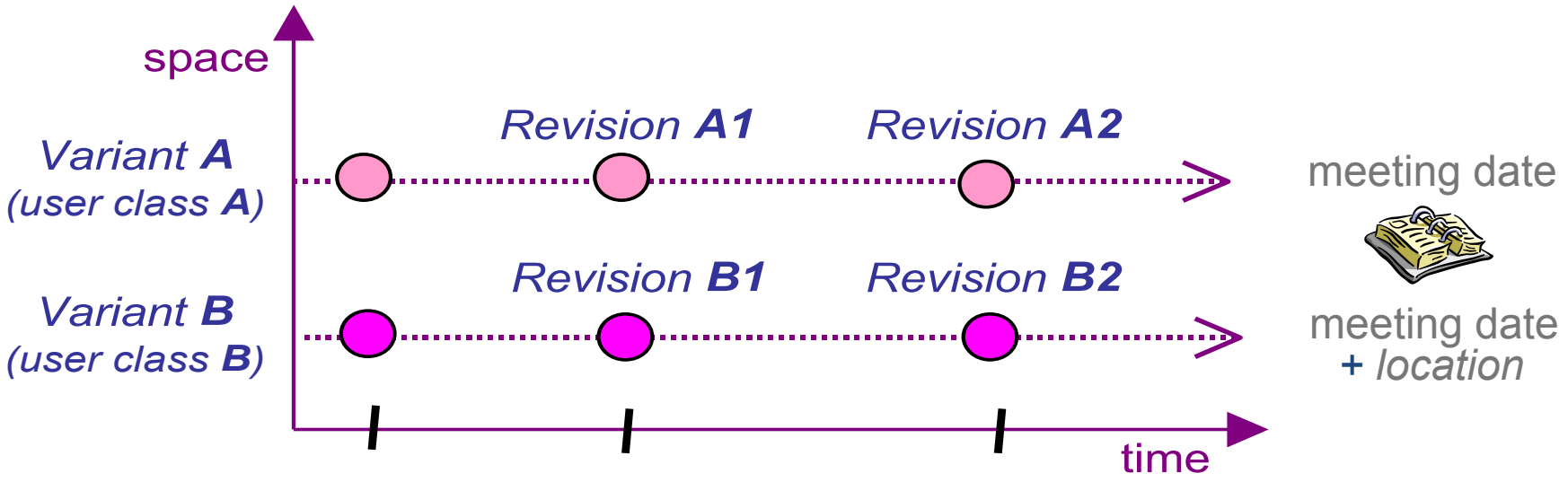
Intra-version traceability: *use*, *derivation* links

*changing **A** makes **B** incomplete, inconsistent, inadequate or ambiguous*

B is built from A under constraint that A must be met



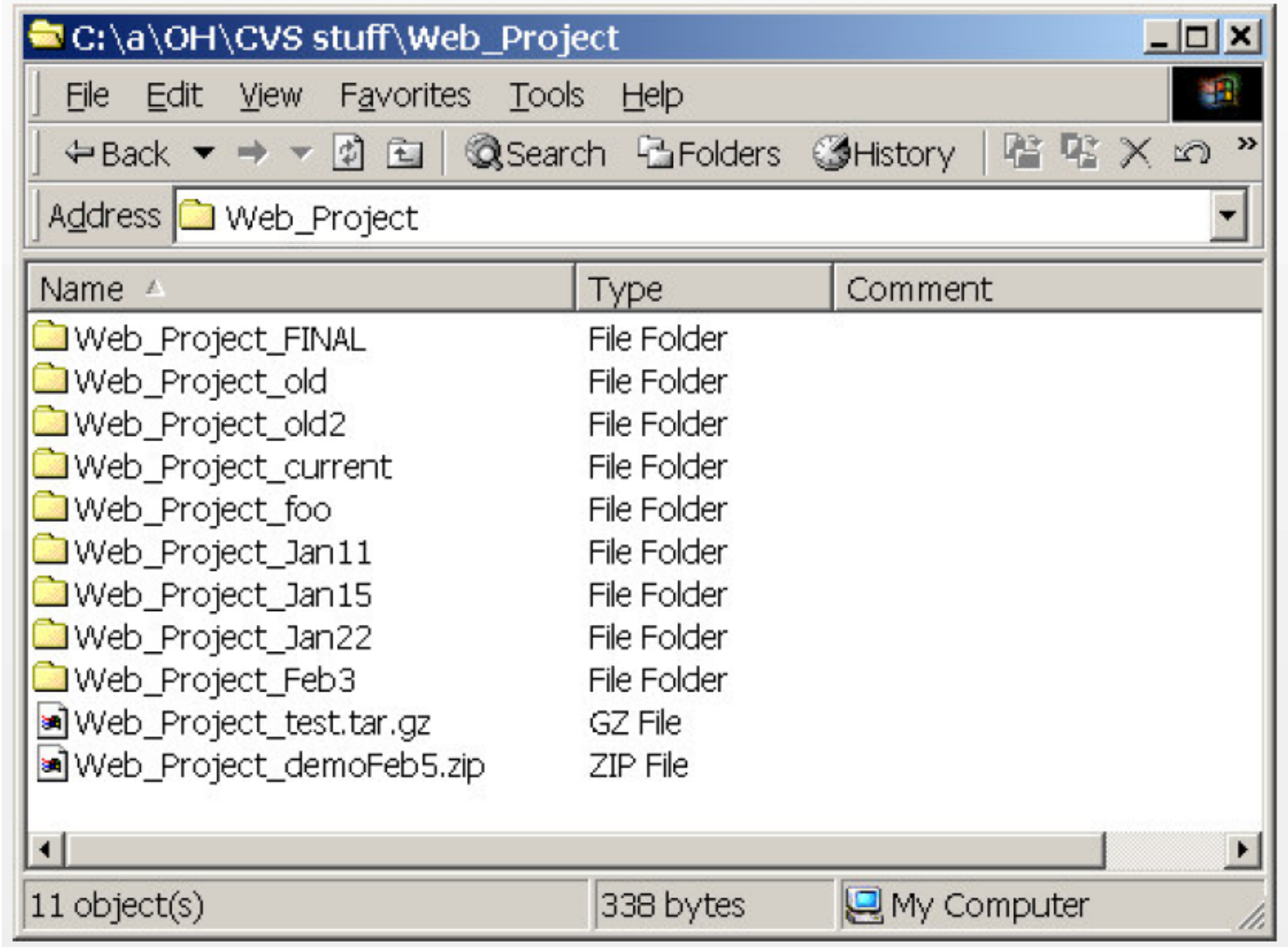
When the Information About Revisions and Variants is Useful?



CASE for CM

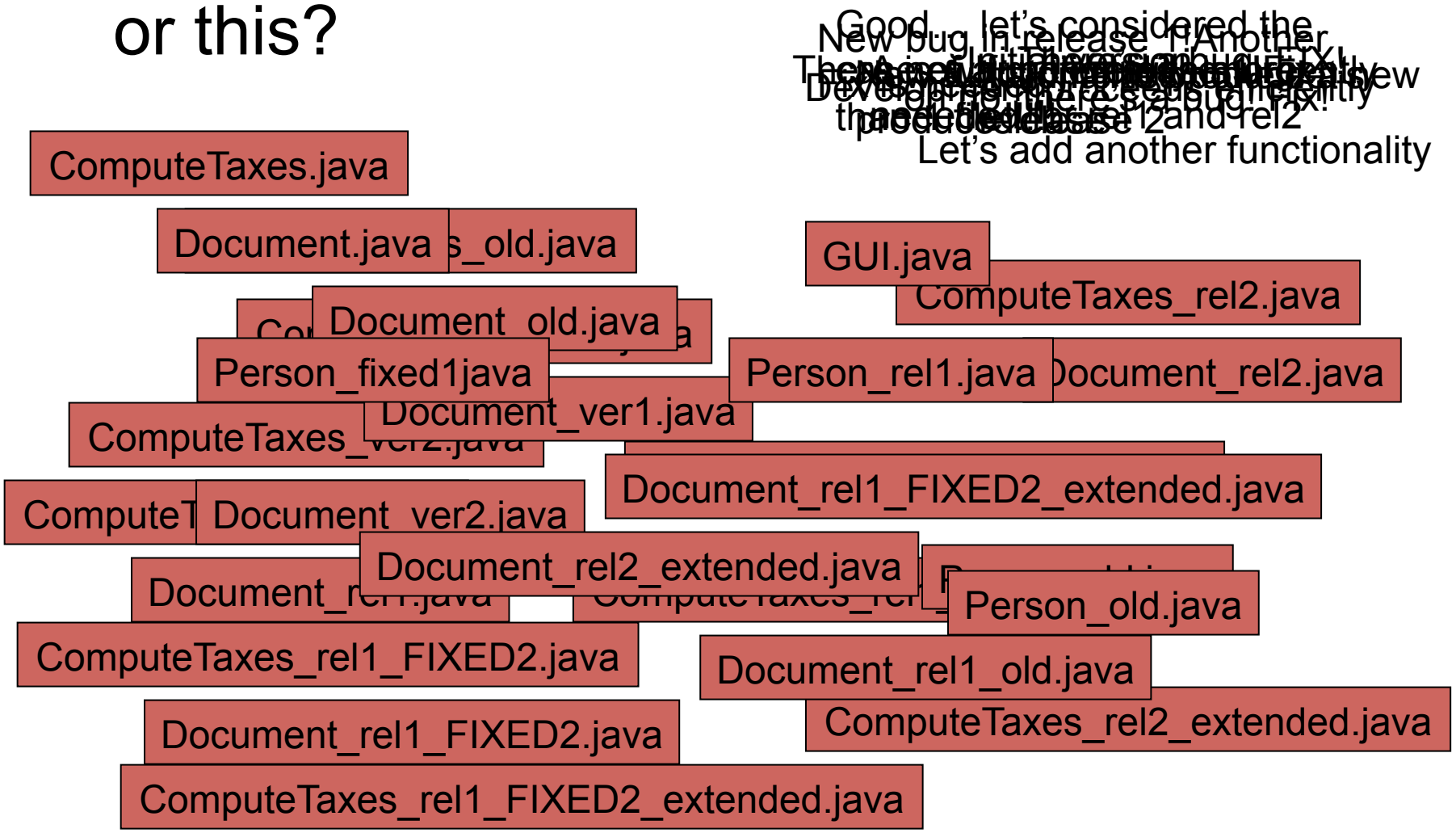
Let’s start by thinking of a world without version control…

Version Management



Version Management

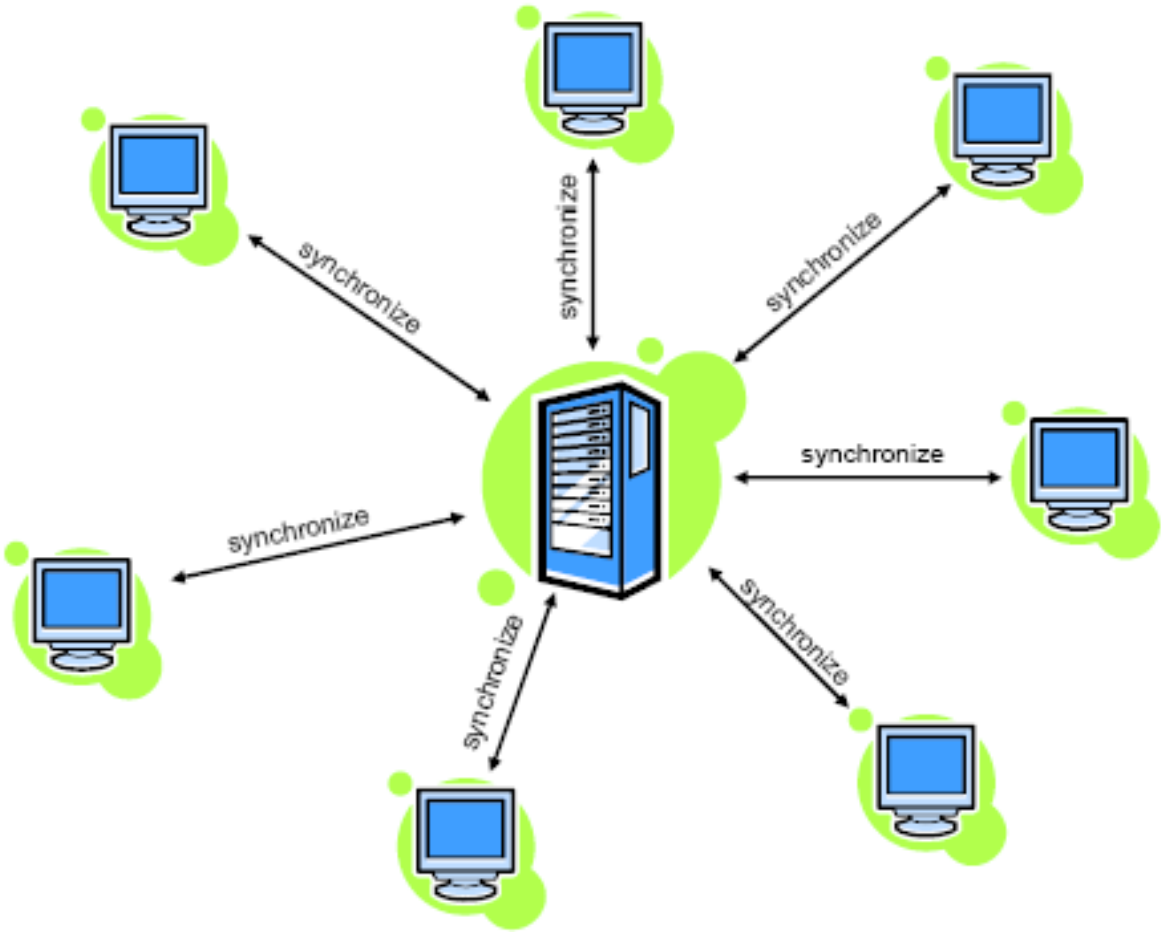
or this?



Coordination

- How to coordinate the activity of multiple developers?
  - Use one PC?
  - Send emails?
  - Use a shared folder?

General Scenario



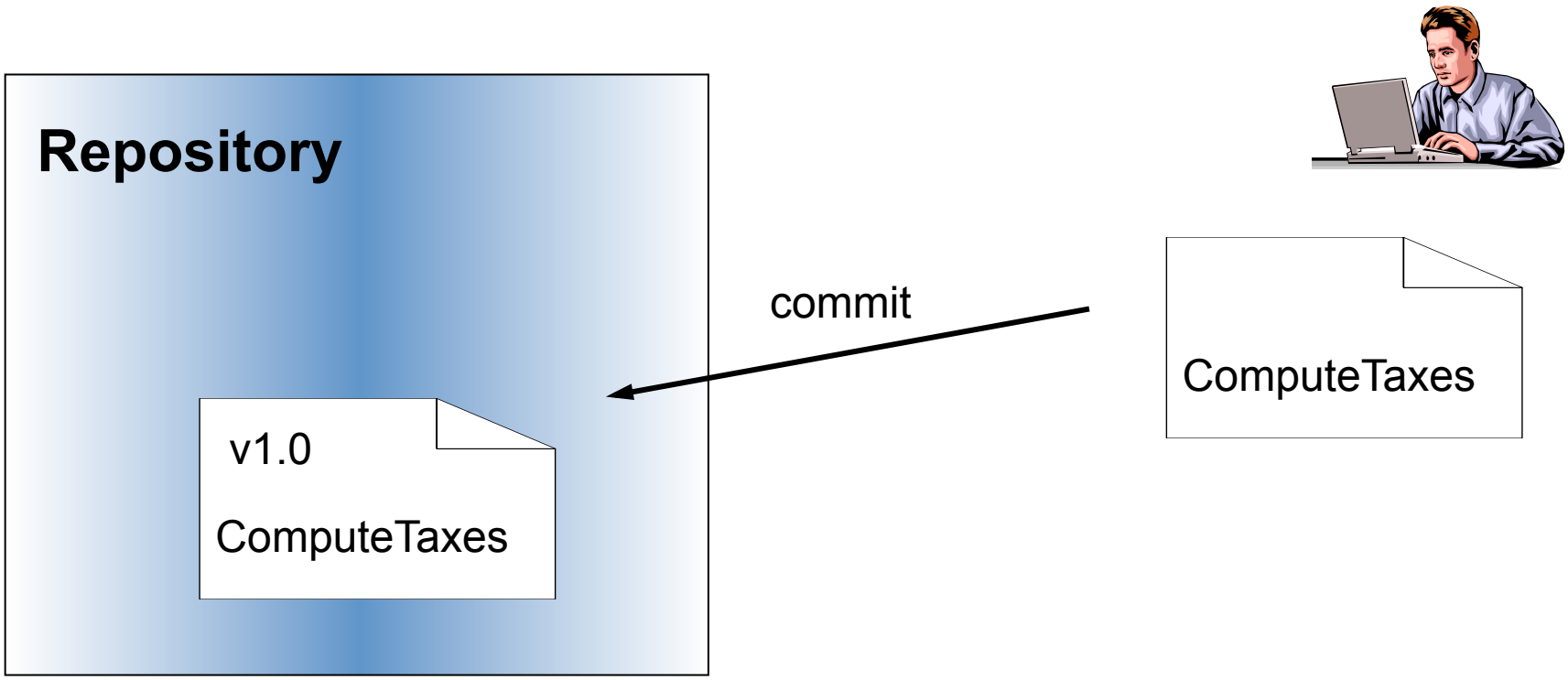
Single User Scenario

produce the initial version

Repository

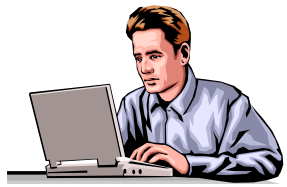
ComputeTaxes

Single User Scenario



Single User Scenario

extend!



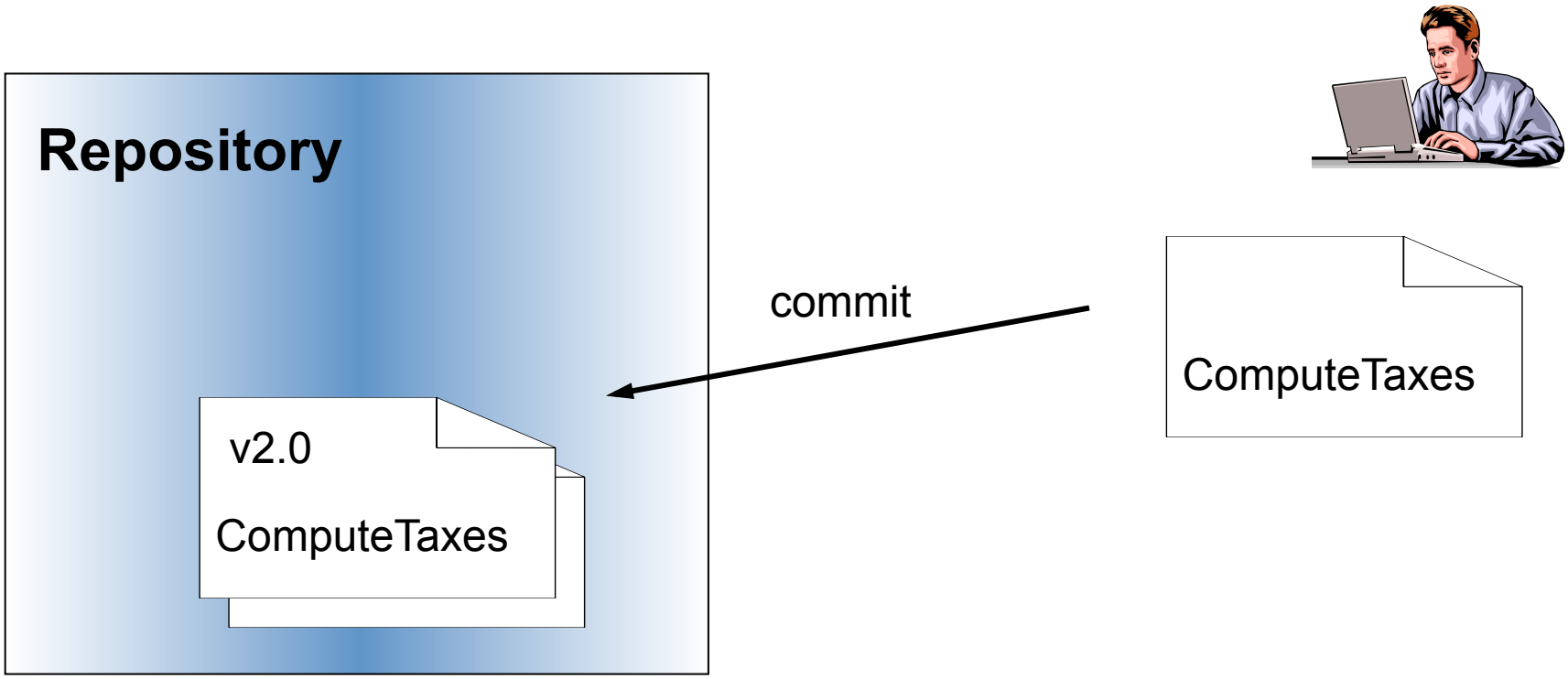
Repository

v1.0

ComputeTaxes

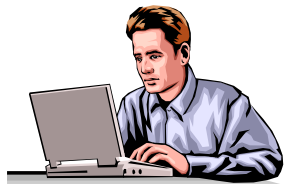
ComputeTaxes

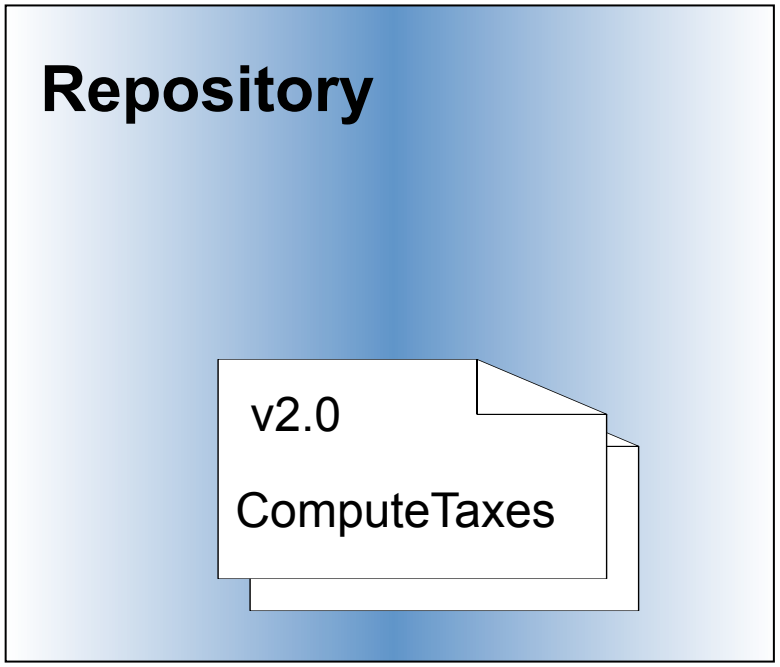
Single User Scenario

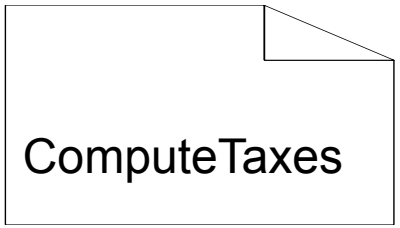


Single User Scenario

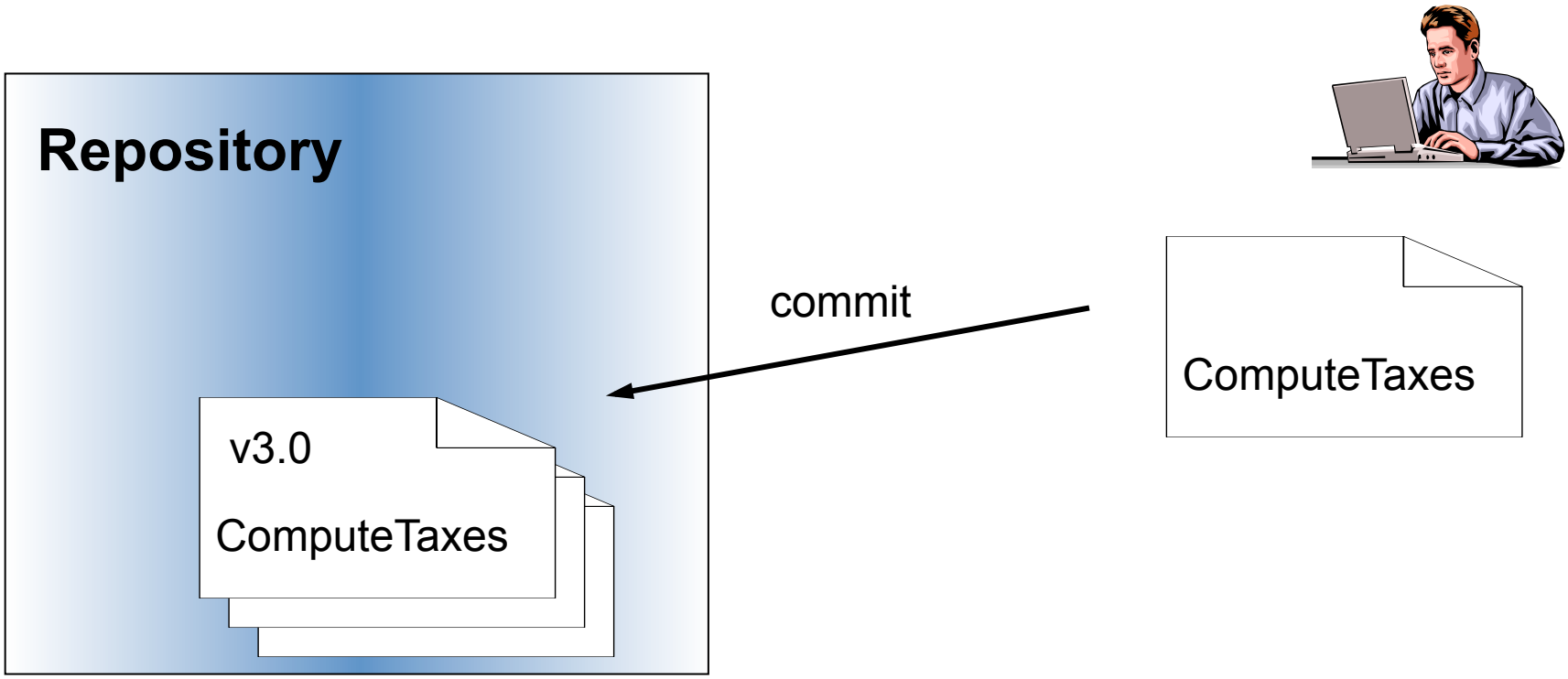
extend!





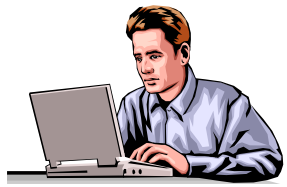


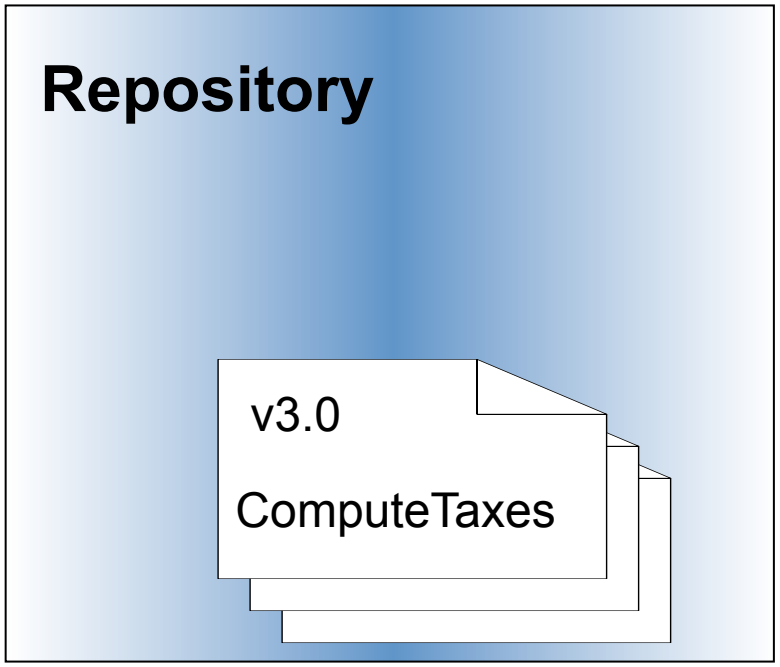
Single User Scenario



Single User Scenario

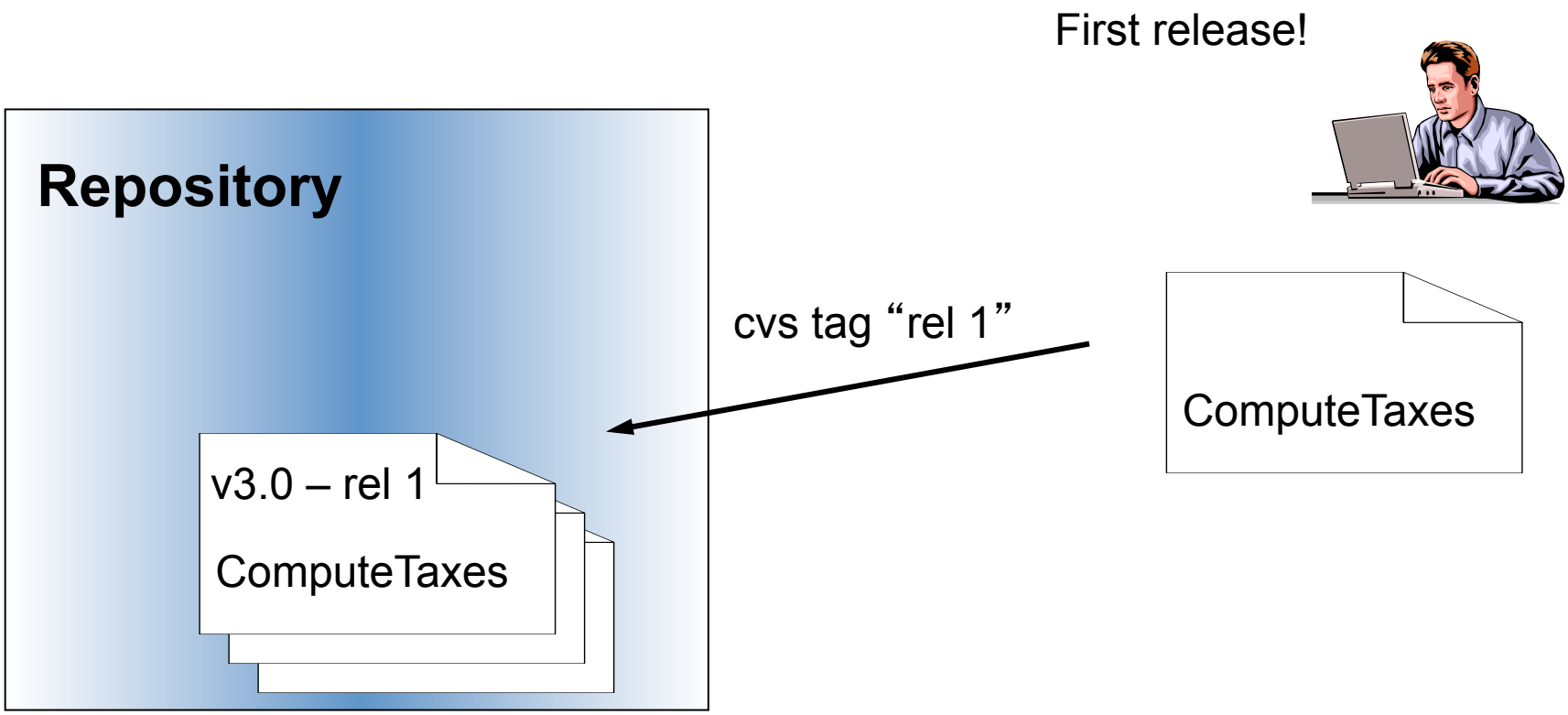
First release!



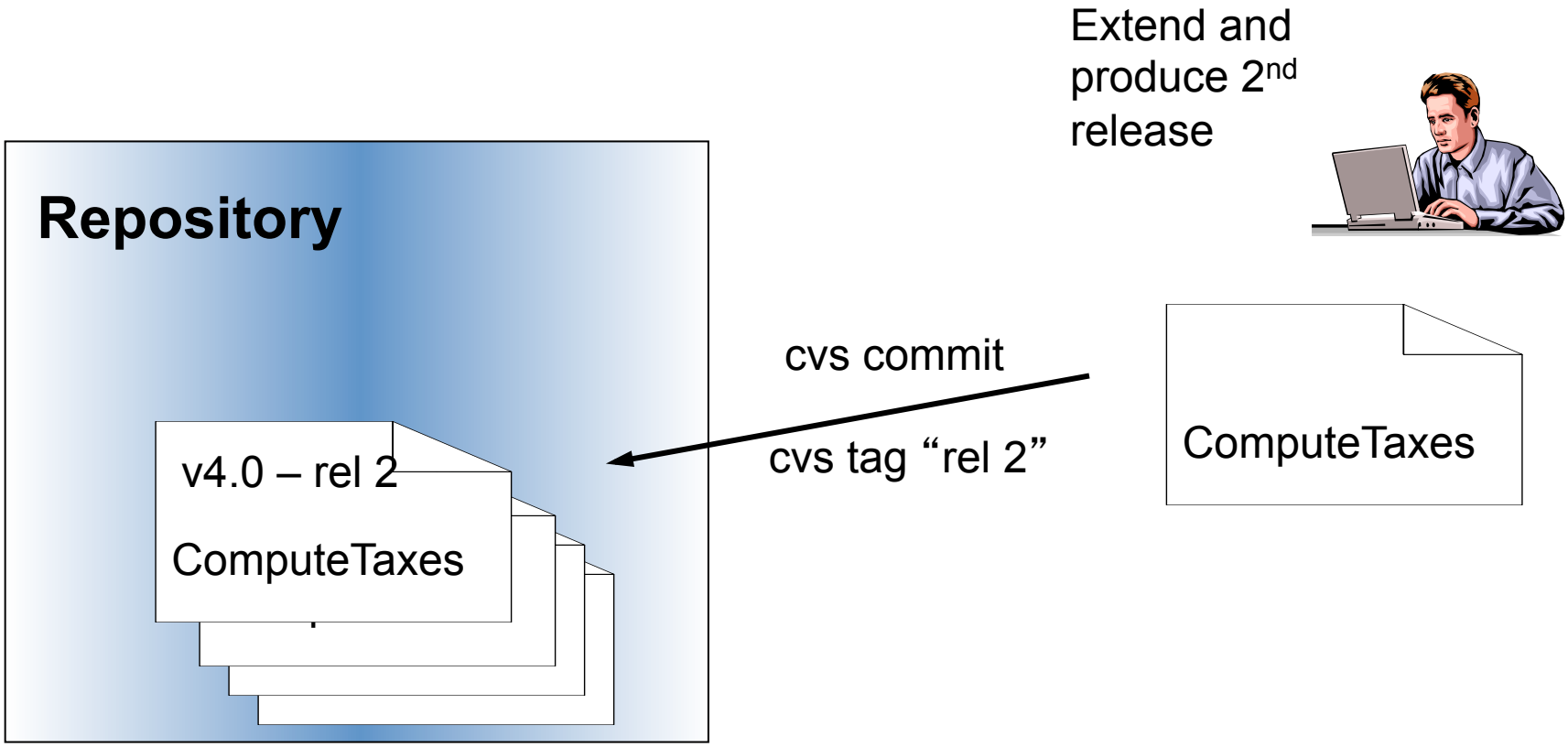


ComputeTaxes

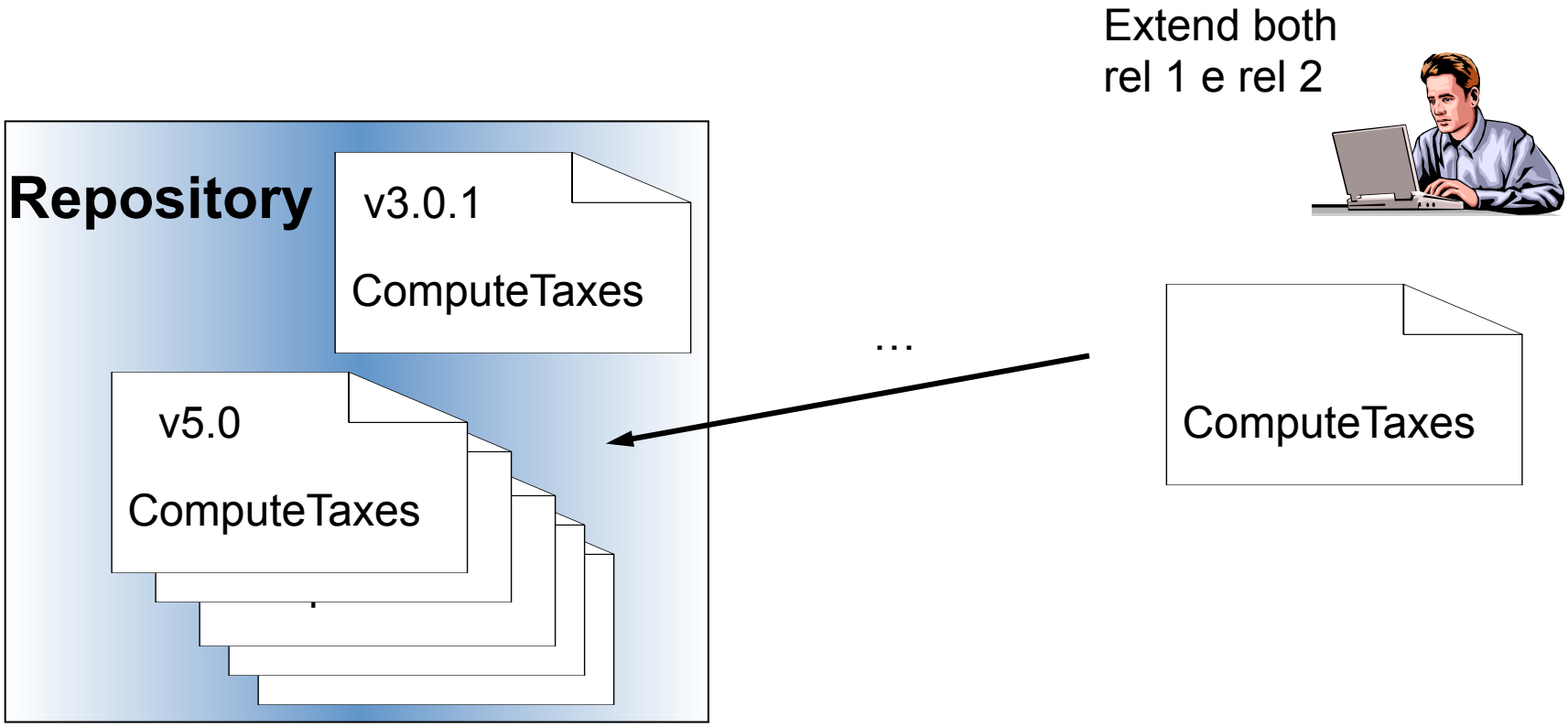
Single User Scenario



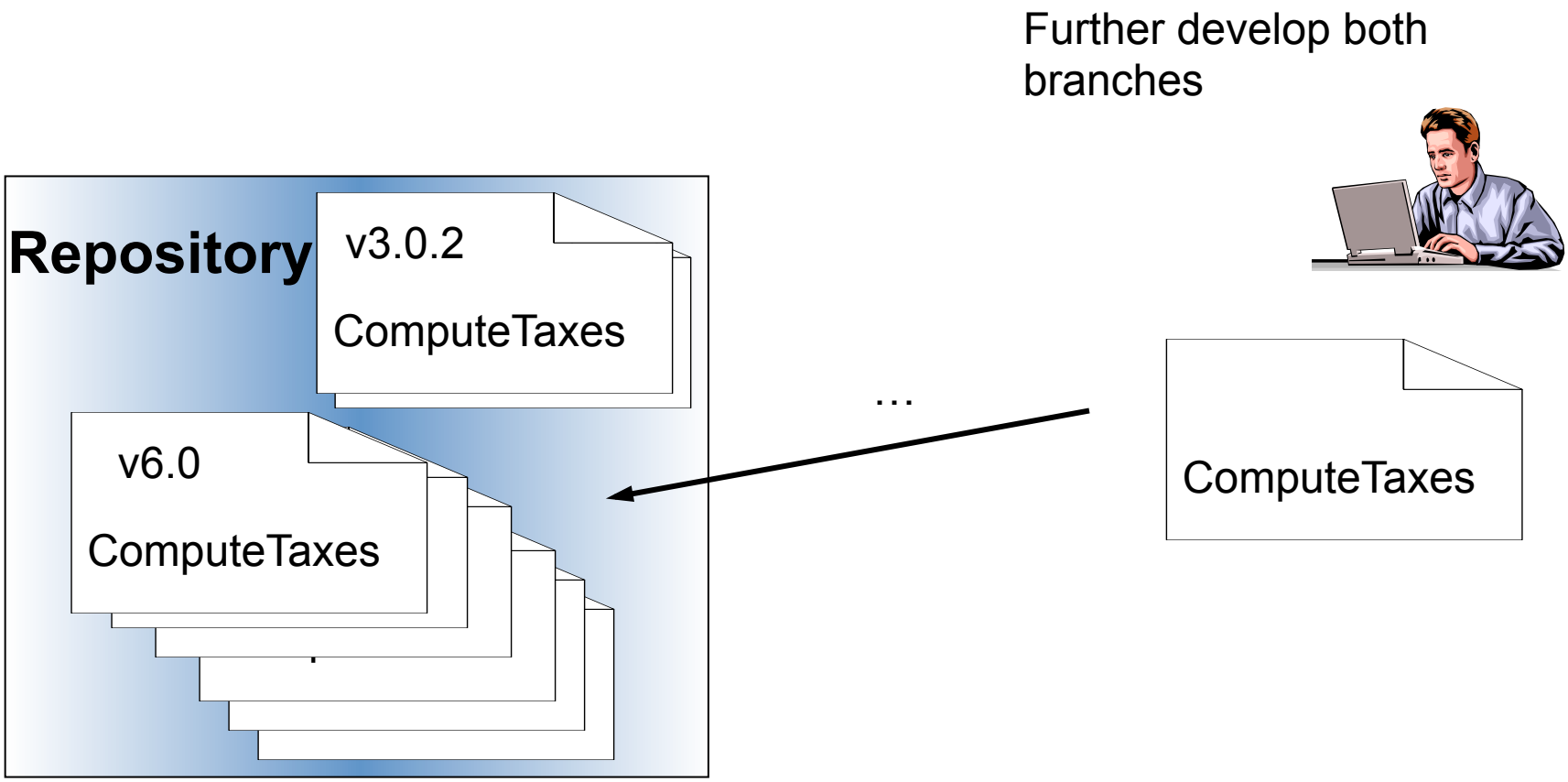
Single User Scenario



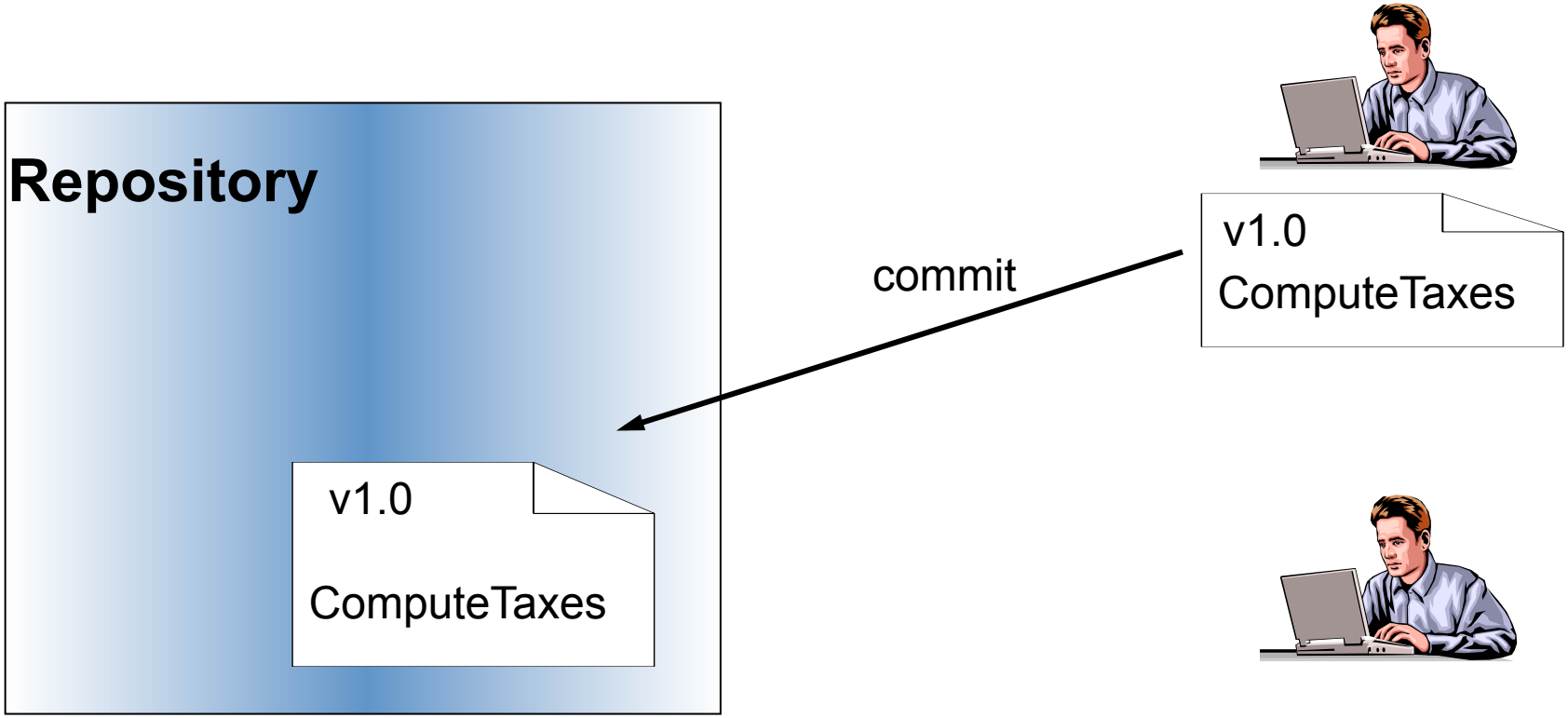
Single User Scenario



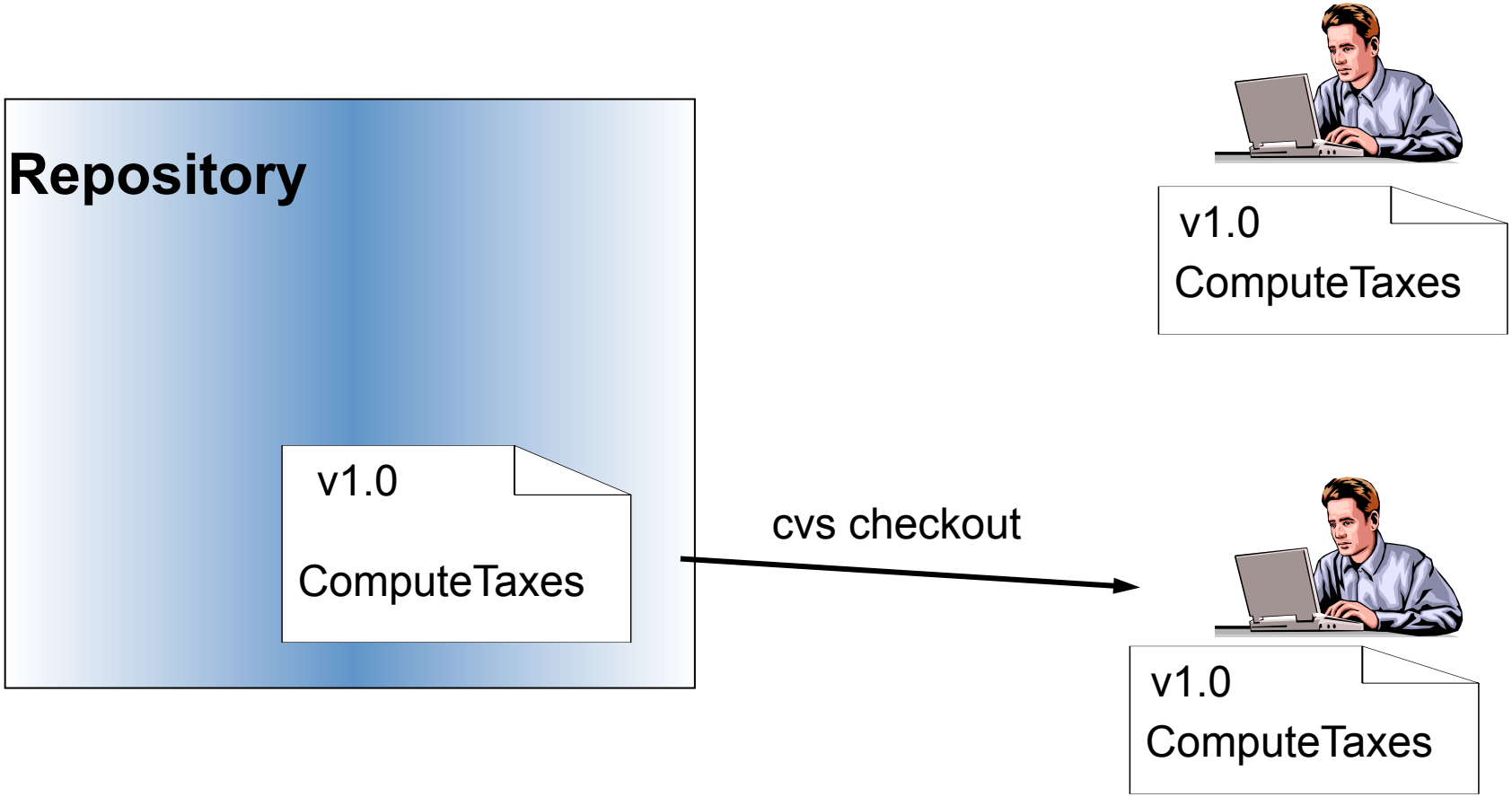
Single User Scenario



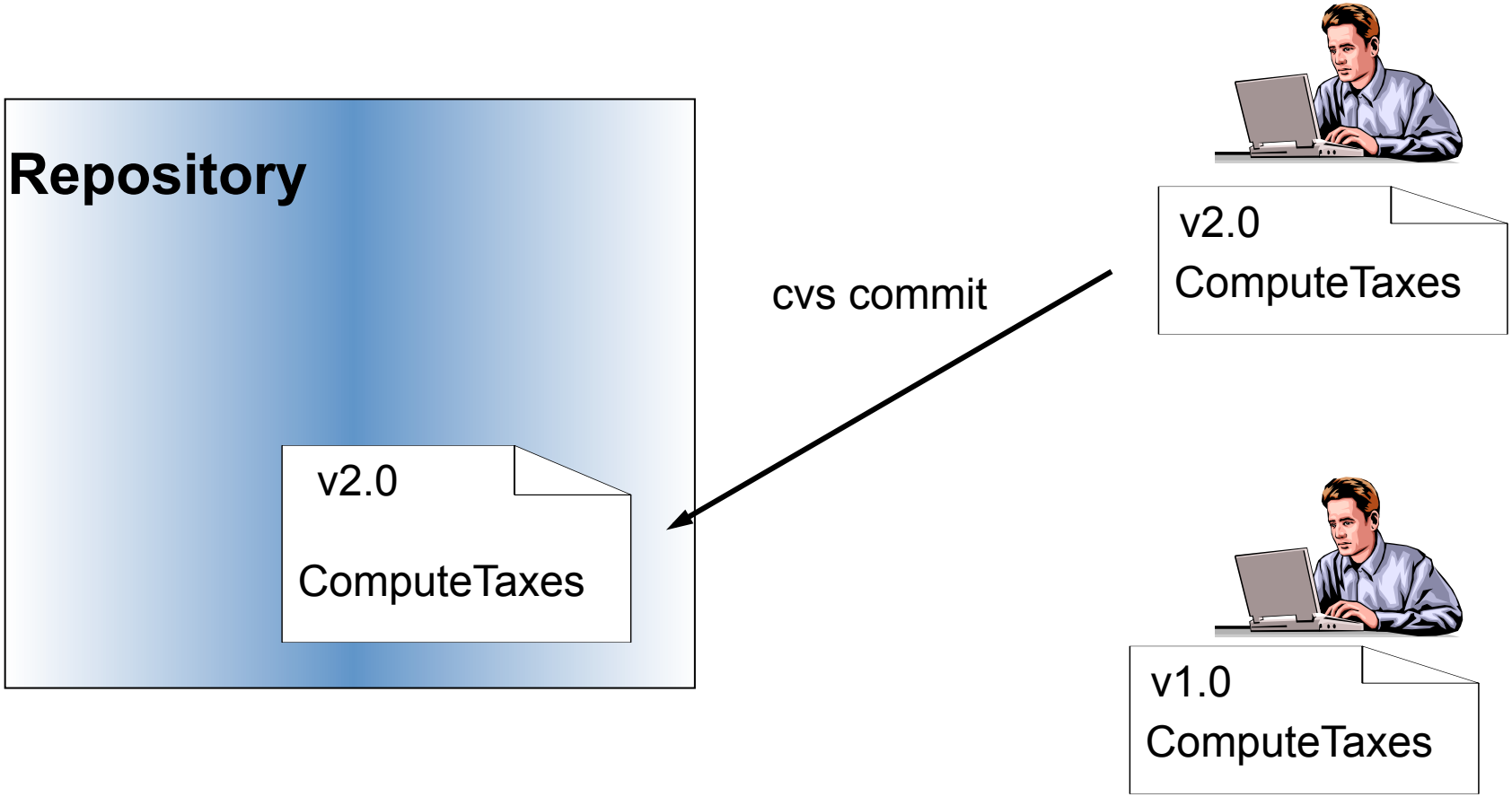
Multi User Scenario



Multi User Scenario

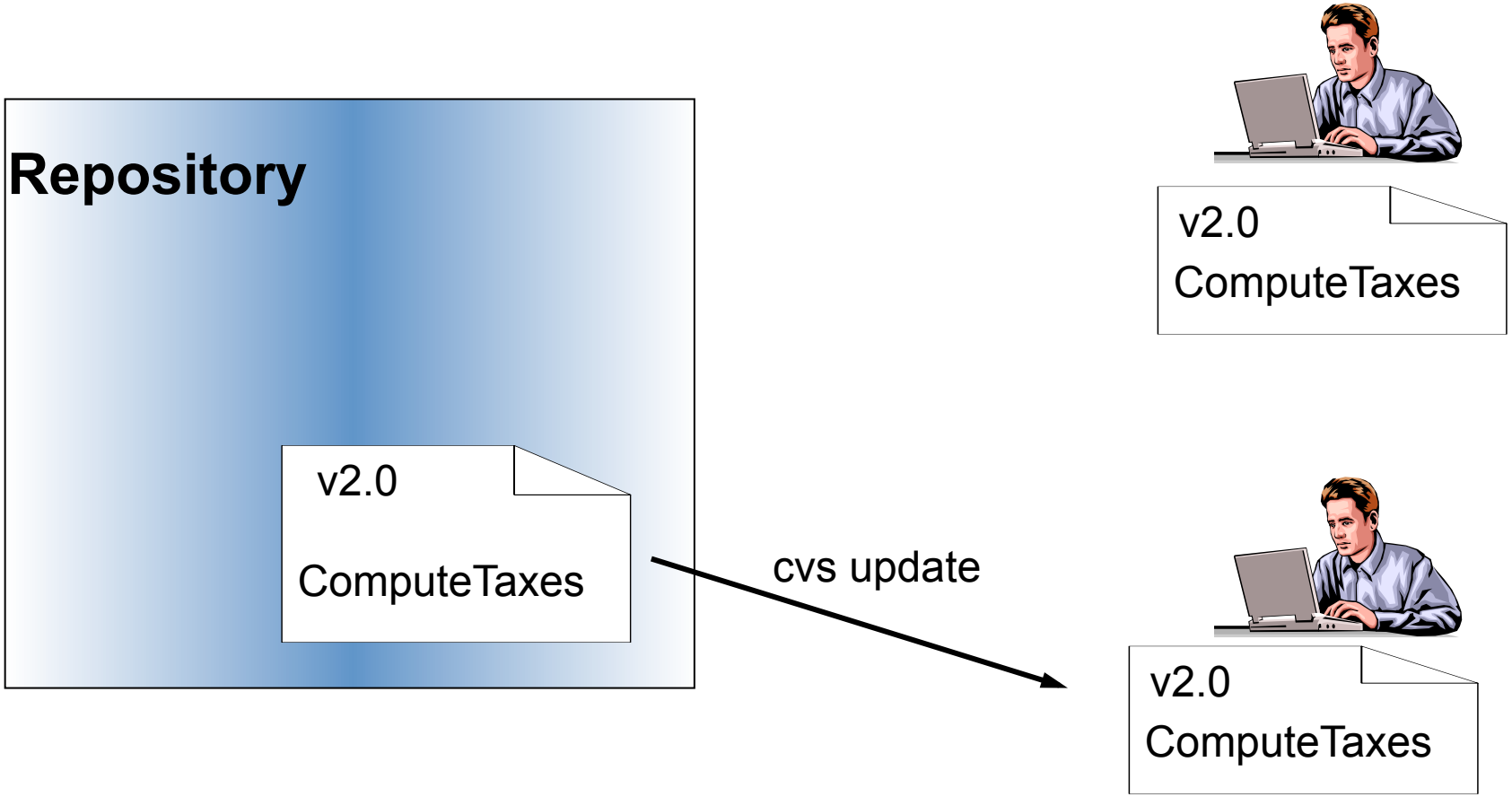


Multi User Scenario



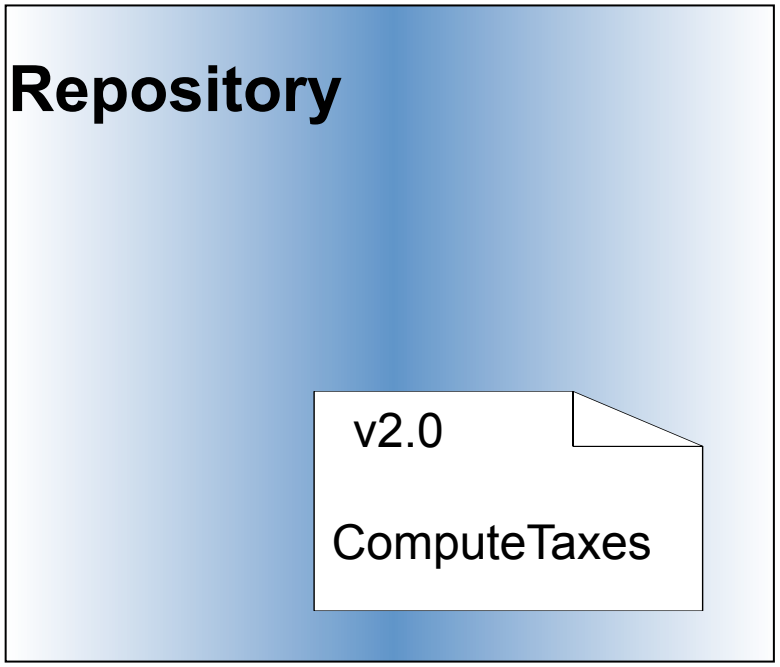
Multi User Scenario

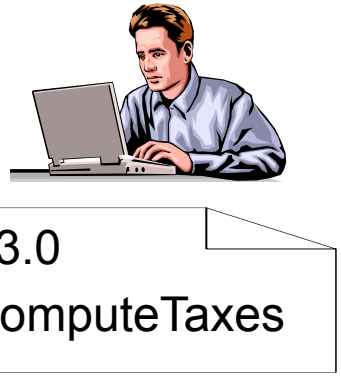
When starting a new working session, the user has to execute an update first!

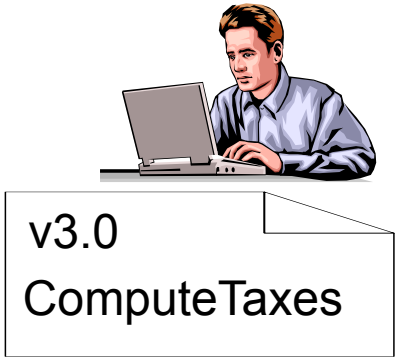


Multi User Scenario

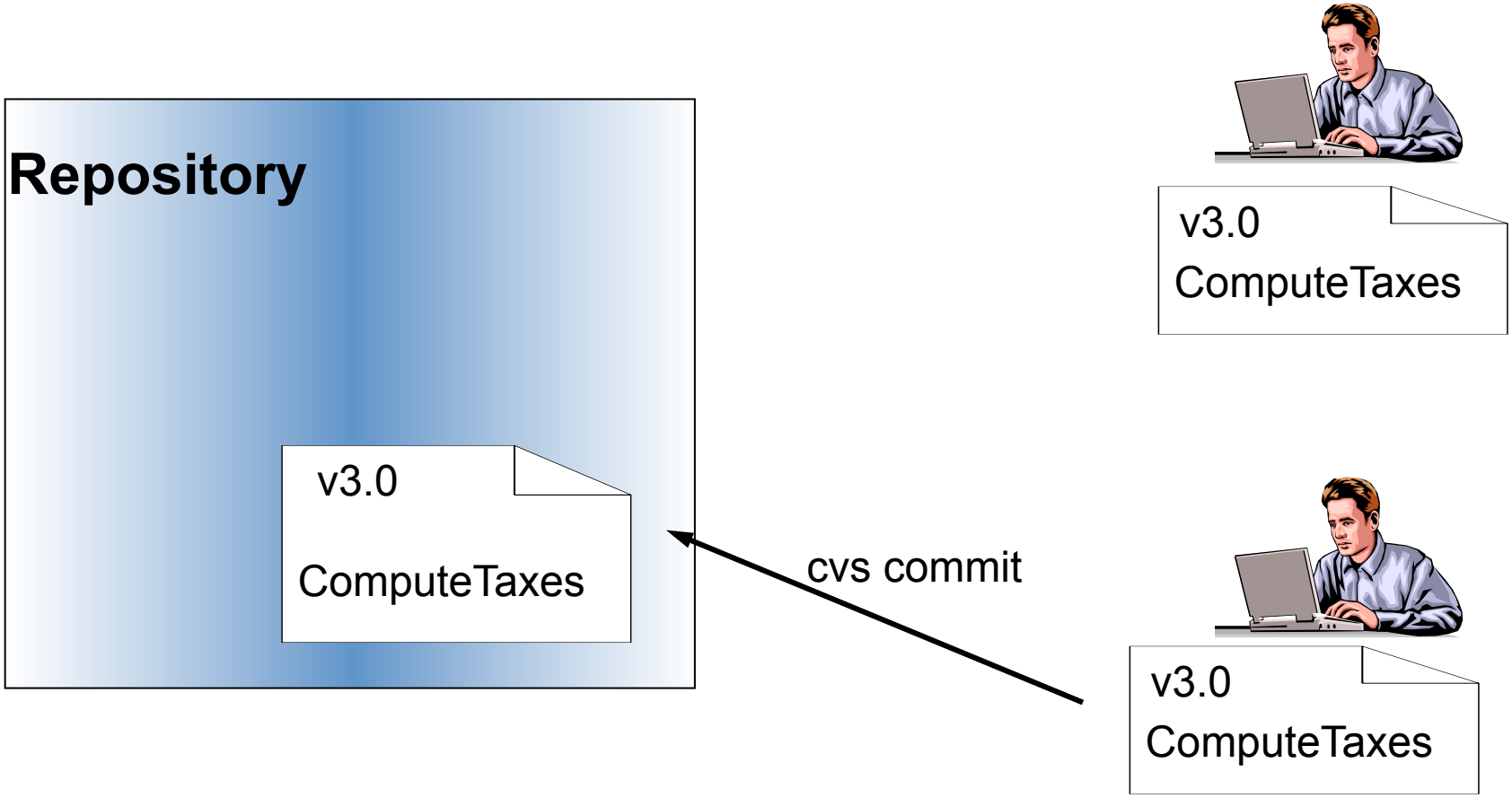
Developers work in parallel





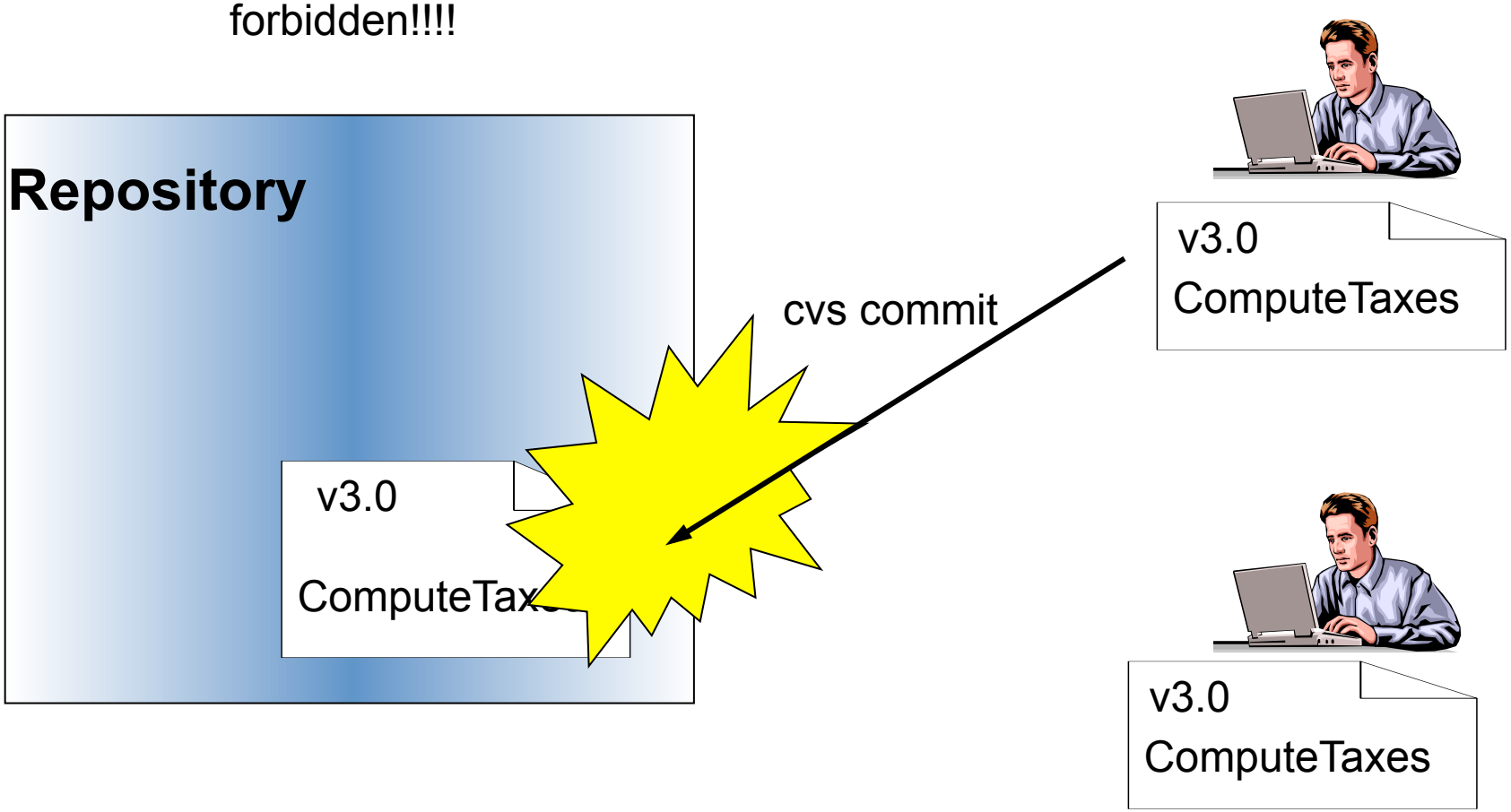


Multi User Scenario

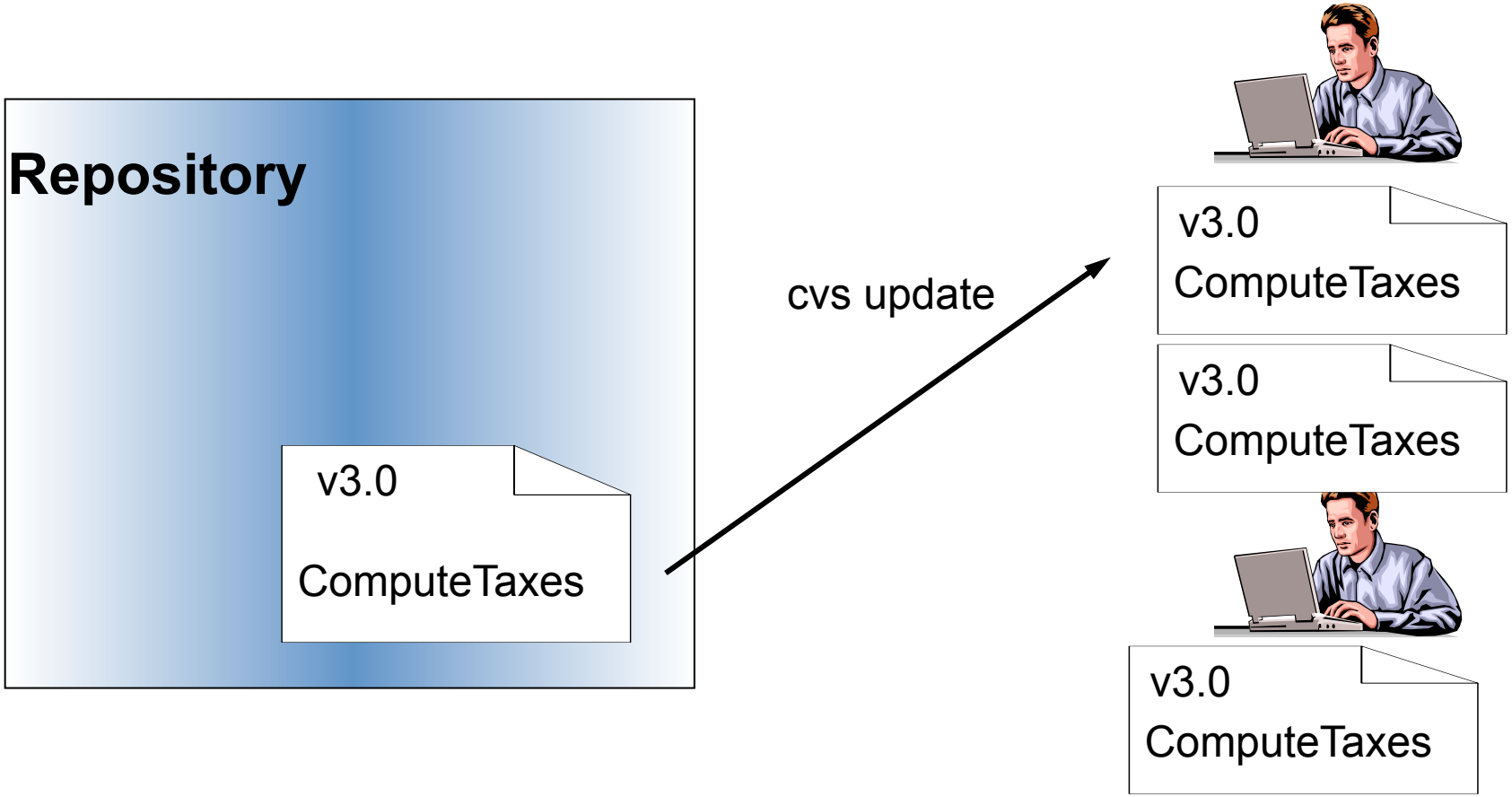


Multi User Scenario

There is an attempt to overwrite the changes implemented by another developer – operation forbidden!!!!

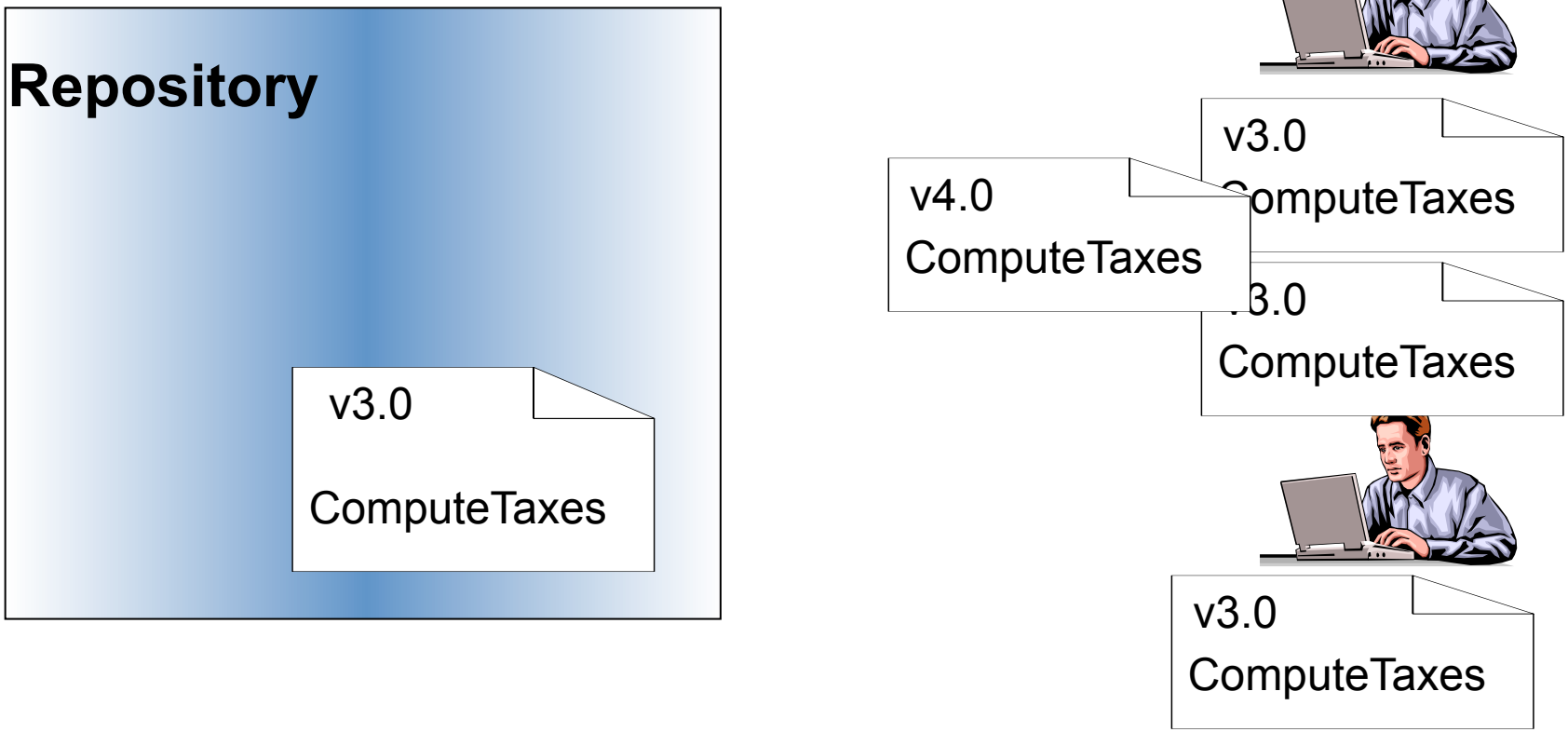


Multi User Scenario

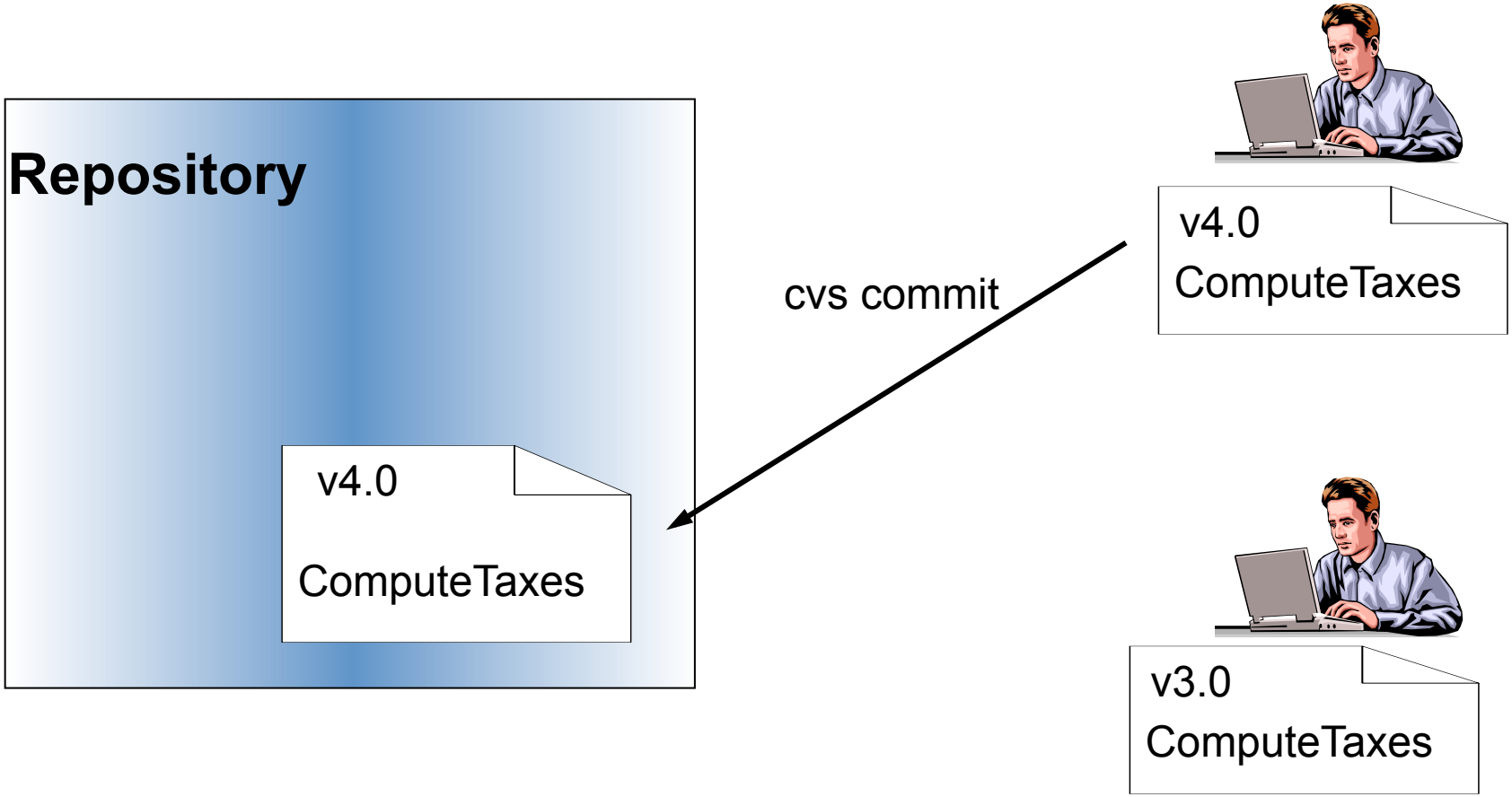


Multi User Scenario

Manual resolution of
the conflicts

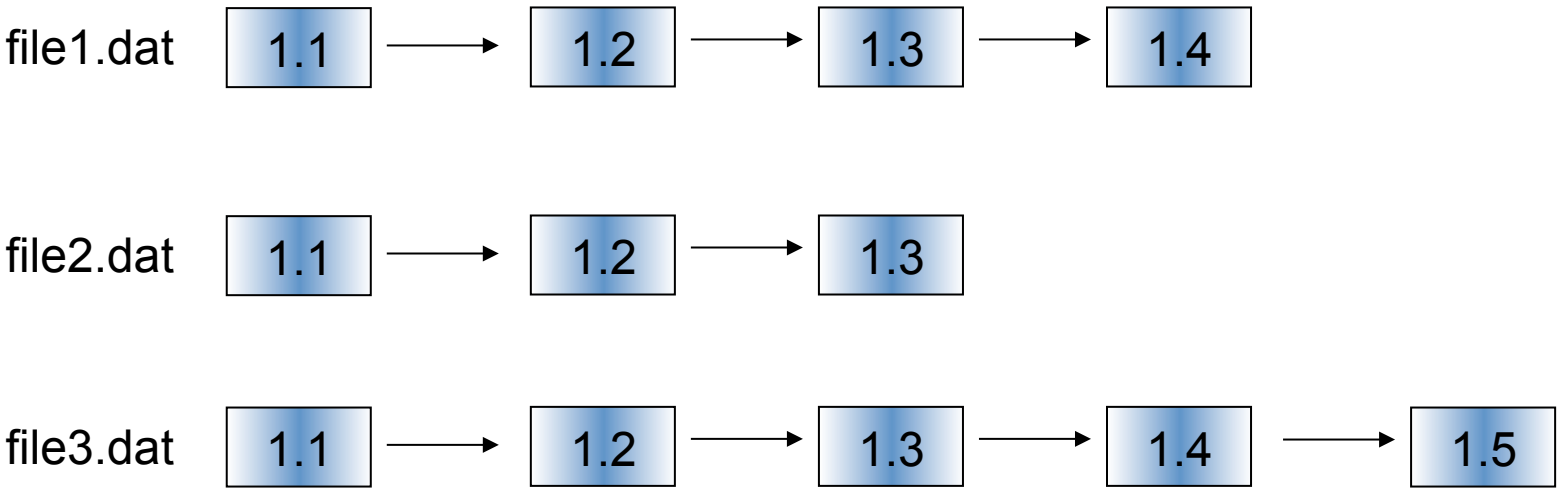


Multi User Scenario



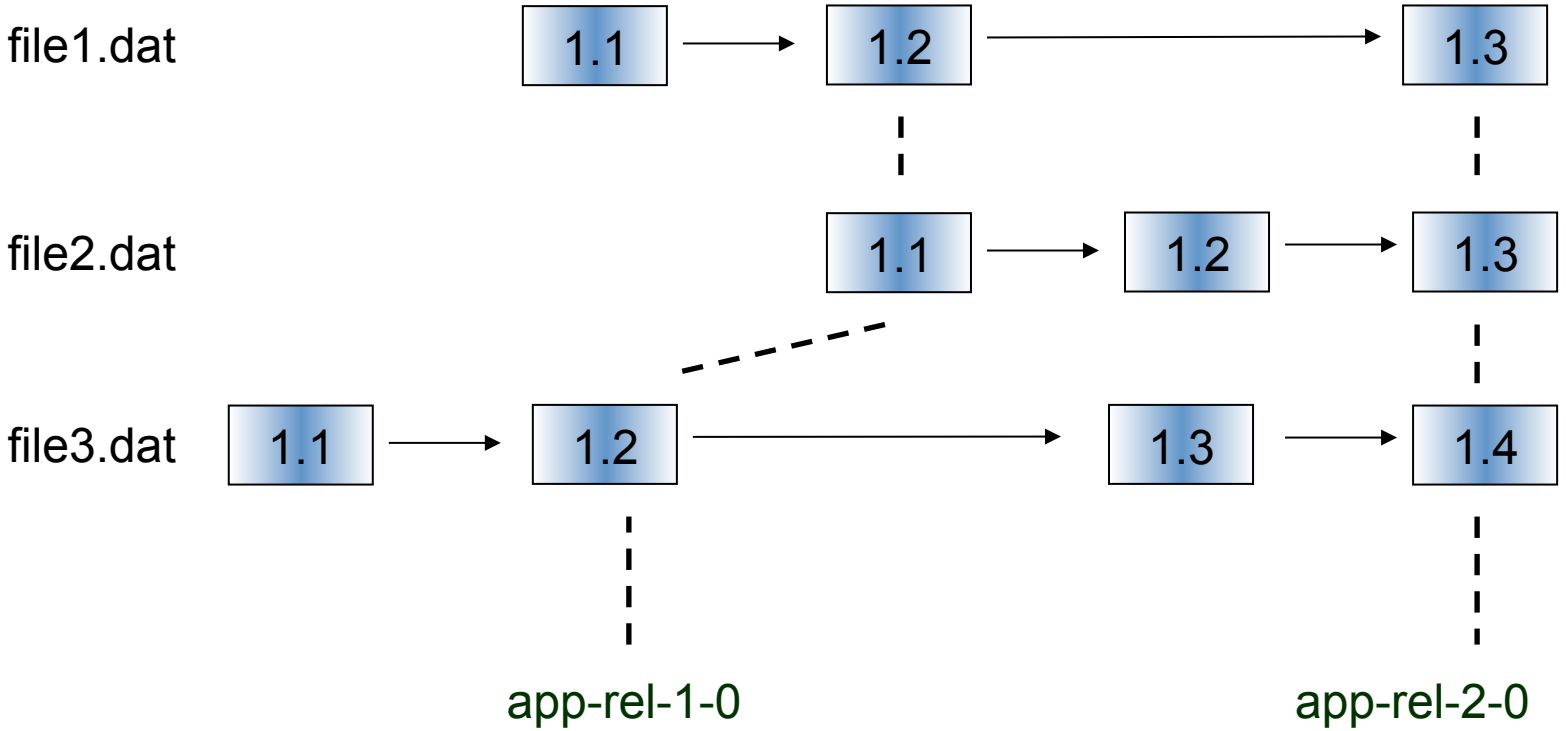
Structure of the Repository

- A same file occurs in multiple revisions



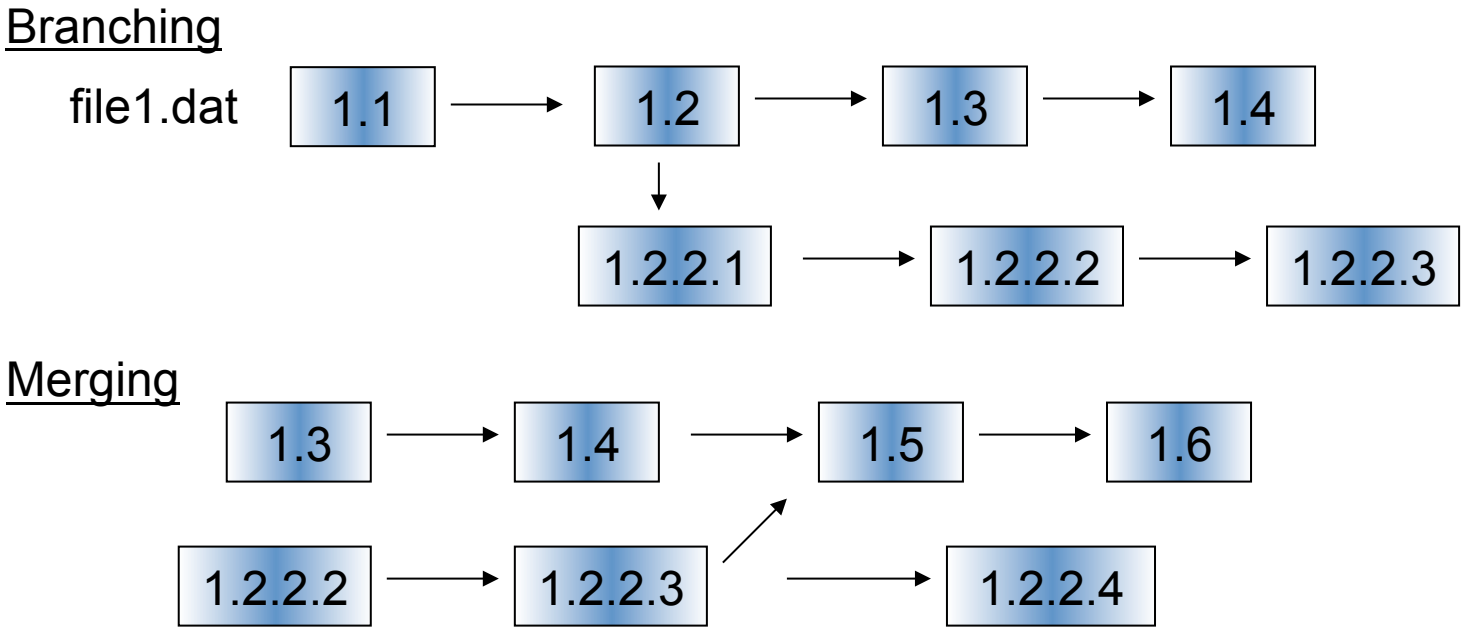
Structure of the Repository

- A coherent and consistent collection of files existing at a given time might be relevant
  - Versions identify these collections of files
  - Each version has a name (tag)



Structure of the Repository

- A project is usually organized into multiple development branches
  - The main branch is usually called head or trunk or master
  - Branches can be created and merged



(free) Tools

CVS

CVS

- CVS = Concurrent Version System
  - To trace versions of documents (and files), and
  - To support concurrent and distributed activity
- File (revision)
- Module (version)
- Repository

- Working/local copy

Concurrent Development

- Concurrent activity might generate conflicts
- Update before Commit to reveal conflicts
  - Conflicts must be resolved by the developer who has to commit changes
  - This rule might affect the behavior of the developers…
- When a change (e.g., to code) can be committed?

Subversion

Subversion

- More recent than CVS
- Simpler than CVS
- Overcome some limitations of CVS
- There are anyway pros and cons

Subversion vs CVS

- Pros SVN
  - Atomic commit
  - SVN implementations perform typically better than CVS implementations
  - SVN efficiently stores binary files
- Pros CVS
  - SVN has a unique version number for the whole repository, while CVS assigns version numbers to the individual files
  - SVN “simulates” tags
  - in case of “disasters”, the CVS repository is easily readable (text files), while SVN is not

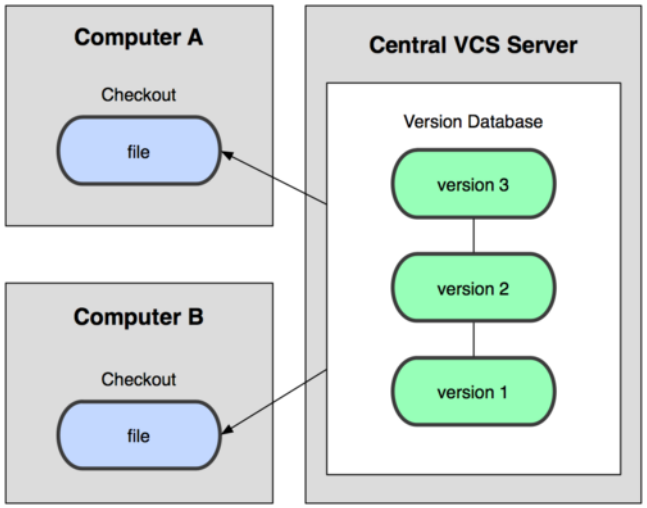
Git

Git

- Efficient, modern, distributed version control system
  - Advanced branching mechanisms
- Many hosting services available online
- GitHub (github.com)
  - Hosting service
  - Developers community
  - Web-based interface
  - Access control
  - Collaboration features (including wikis, etc.)

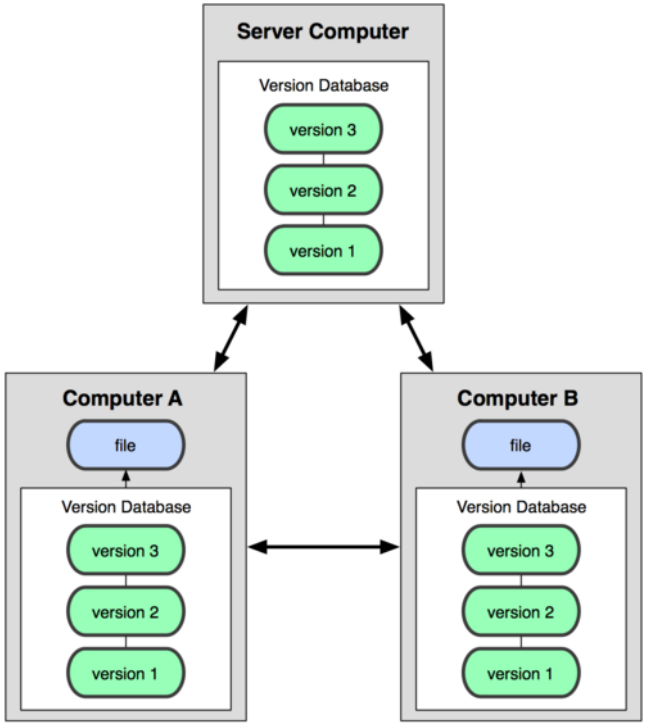
Git uses a distributed model

Centralized Model



(CVS, Subversion, Perforce)

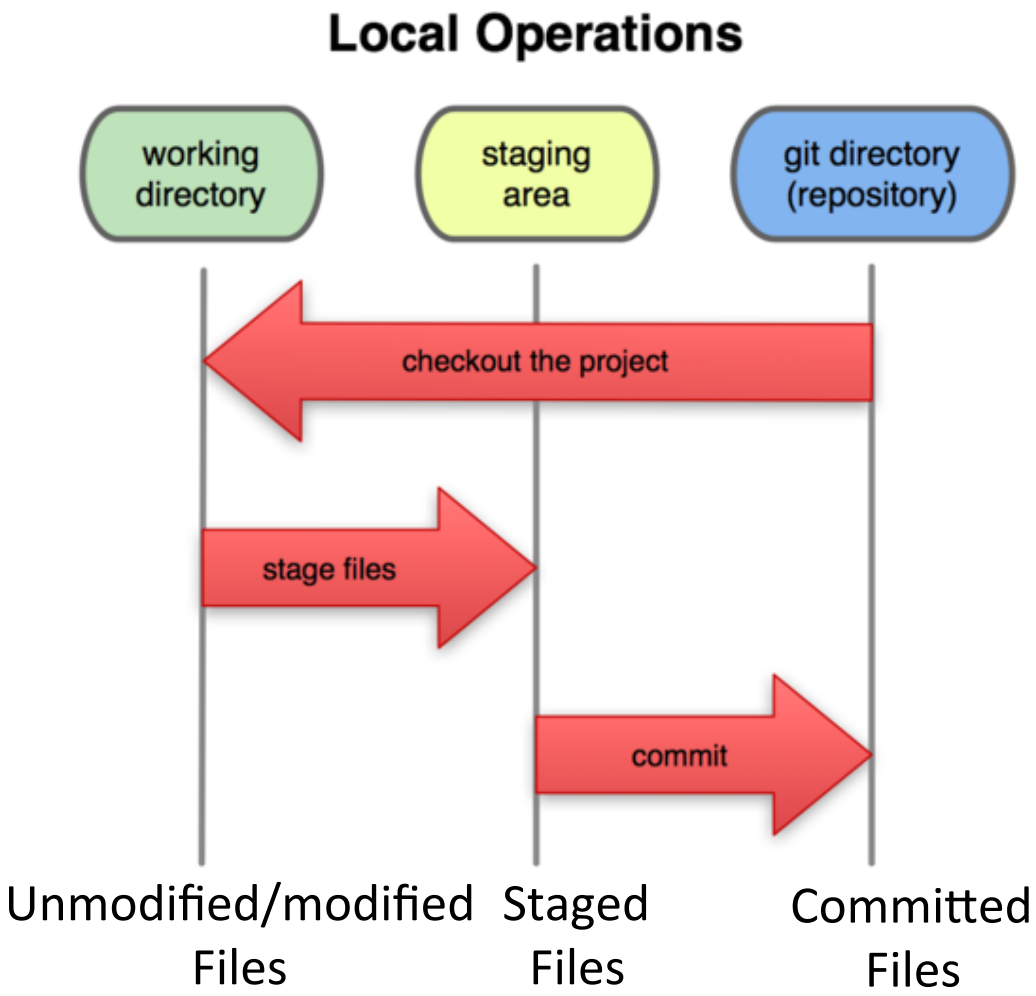
Distributed Model



(Git, Mercurial)

Result: Many operations are local

A Local Git project has three areas



Basic Workflow

- Basic Git workflow
  - Modify files in your working directory
  - Stage files, adding snapshots of them to your staging area
  - Do a commit, which takes the files as they are in the staging area and stores that snapshot permanently to your Git directory
- Notes:
  - If a particular version of a file is in the git directory, it is considered committed
  - If it is modified but has been added to the staging area, it is staged
  - If it was changed since it was checked out but has not been staged, it is modified

Aside: So what is github?

- GitHub.com is a site for online storage of Git repositories
- Many open source projects use it, such as the Linux kernel
- You can get free space for open source projects or you can pay for private projects

- Question: Do I have to use github to use Git?
- Answer: No!
- you can use Git completely locally for your own purposes, or you or someone else could set up a server to share files, or you could share a repo with users on the same file system

From Version Control to Continuous Integration

- When a new version is ready, a number of quality control activities can be executed
  - Testing
  - Analysis
  - …
- Why not executing them regularly?
  - Or after every commit?

Continuous Integration Policy

- A time (say 5pm) for delivery of system components is agreed
- A new version of a system is built from these components by compiling and linking them
- This new version is tested using pre-defined tests
  - See the second part of the lecture for information about testing and analysis
- Faults that are discovered during testing are documented and returned to the system developers

Maven

Project management and comprehension tool

Build Tools Retrospective

- One level above ant
- Make → ant → Maven
- (Assembly → C → C++)

Make	makefile	target
Ant	build.xml	target
Maven	project.xml maven.xml	goals

Desired Features

- Dependency management
- Versioning
- Compile Java code, build jars
- Execute tests and report results, fail build on failed tests
- Run quality-check tools (PMD, Findbugs, Checkstyles)
- File generation (XmlBeans, Xsl, Velocity, AspectJ)
- Property expansion / token substitution
- Build vs. deploy vs. release
- Full control when needed
- Cross-platform
- IDE Support
- Documentation / Support

Objectives

- Make the development process visible or transparent
- Provide an easy way to see the health and status of a project
- Decreasing training time for new developers
- Bringing together the tools required in a uniform way
- Preventing inconsistent setups
- Providing a standard development infrastructure across projects
- Focus energy on writing applications