Advanced School on PARALLEL COMPUTING

Configuration Management

Leonardo Mariani

University of Milano Bicocca

mariani@disco.unimib.it

CINECA

Configuration Management

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

CM standards

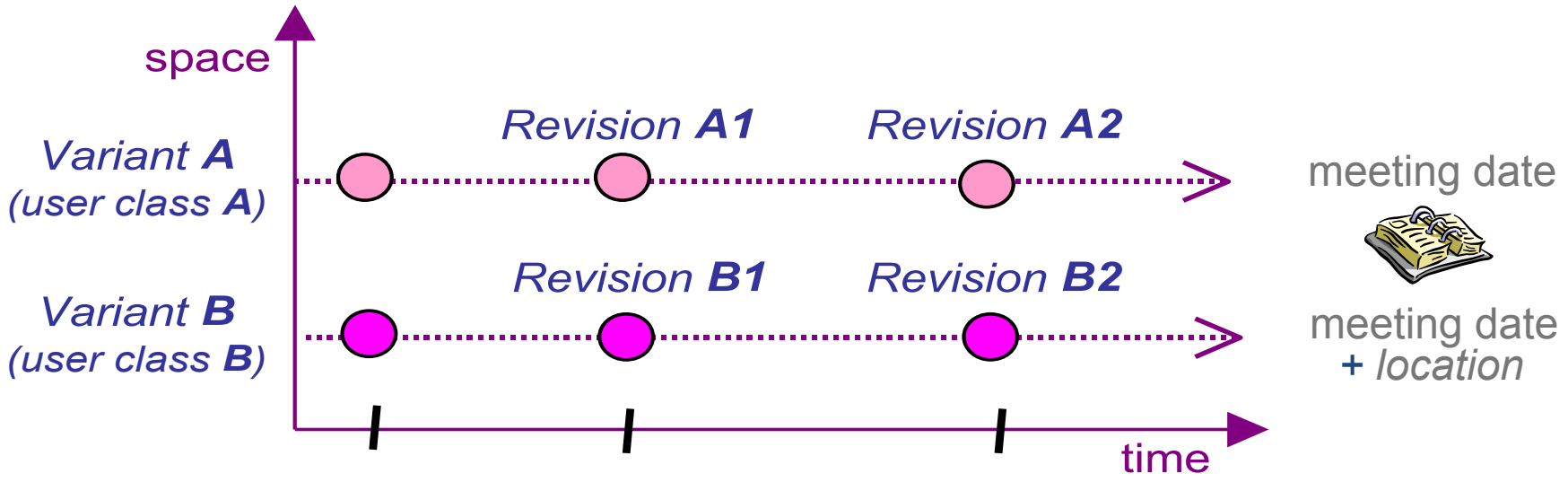
- based on a set of standards which are applied within an organisation
  - how items are identified,
  - how changes are controlled
  - how new versions are managed
- Standards may be based on external CM standards (e.g., IEEE standard for CM)
- Products to be managed?
  - specifications, designs, programs,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

From A. van Lamsweerde “Requirements Engineering”

TRACEABILITY

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

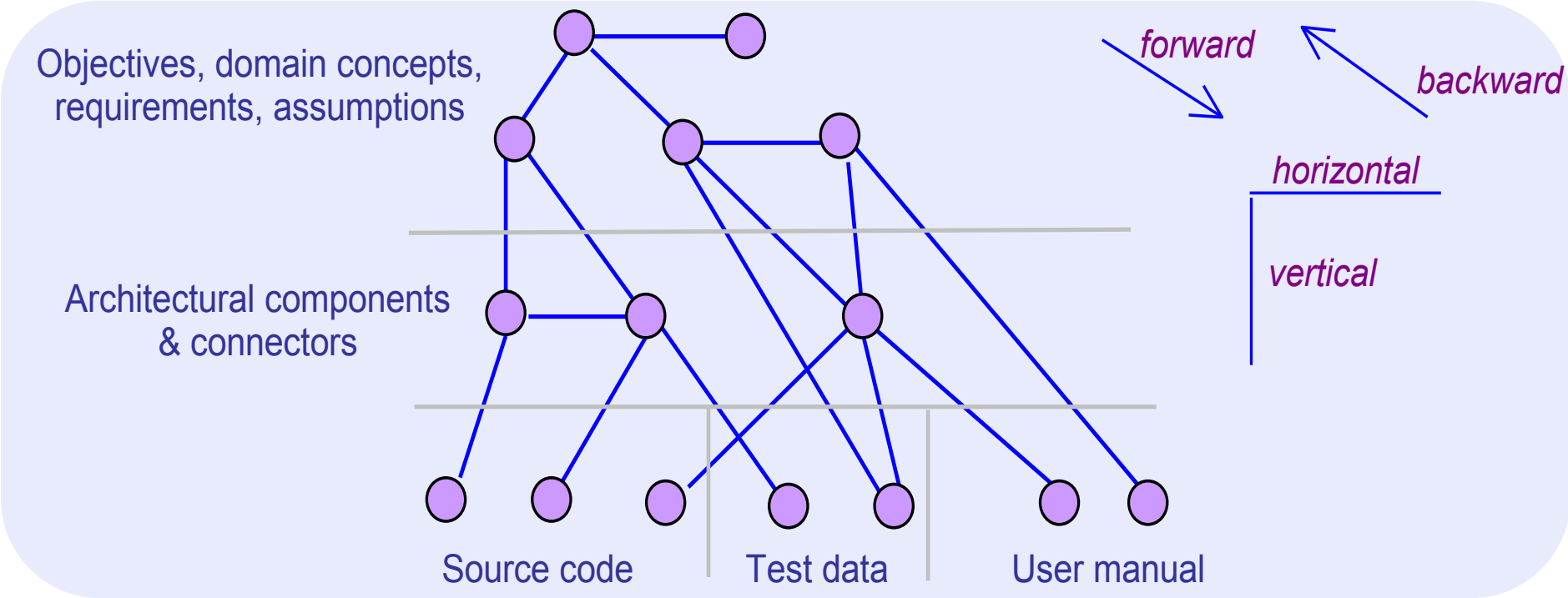


Evolution support requires traceability management

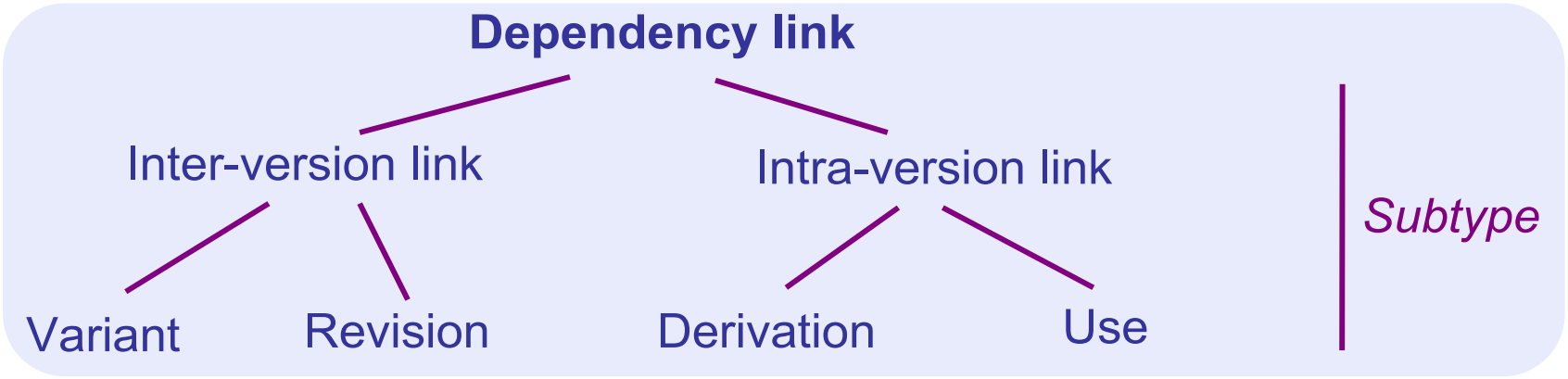
- An item is **traceable** if we can fully figure out
  - WHERE it comes from, WHY it is there
  - WHAT it will be used for, HOW it will be used
- Traceability management (TM), roughy
  - identify, document, retrieve the rationale & impact of items
- Objectives of traceability
  - assess **impact** of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



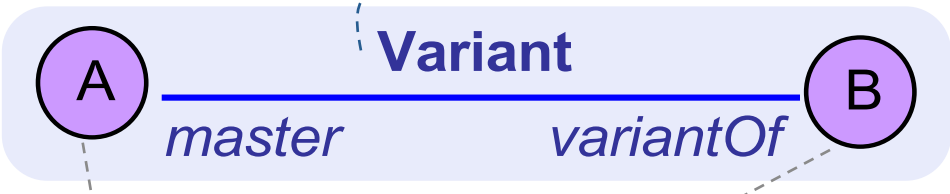
A taxonomy of traceability link types



What are the types of links traced by Configuration Management?

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

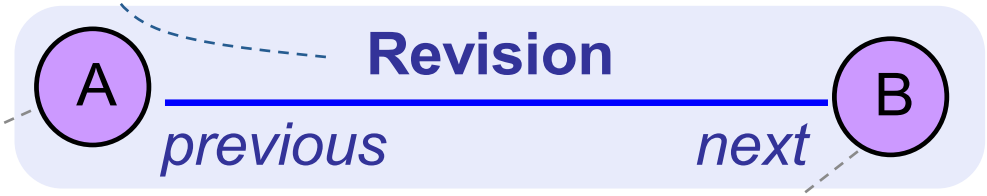
***B** has all features of **A** + specific ones*



common reqs for meeting scheduling

specific reqs for handling important participants

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



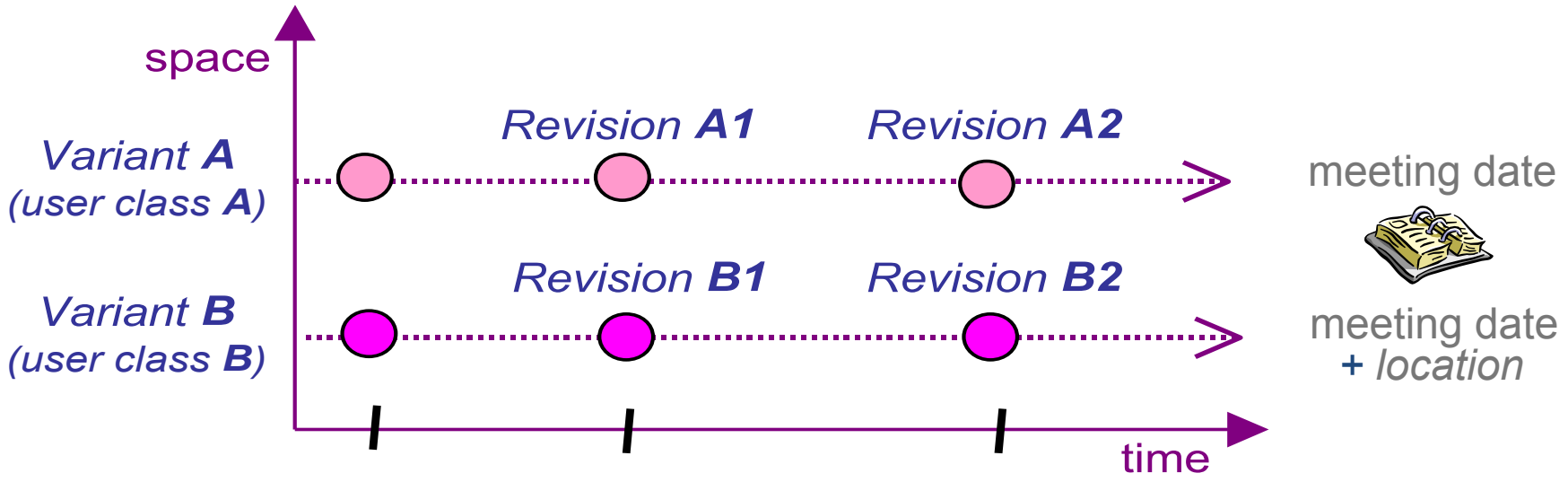
reqs for optimal date to fit exclusion constraints

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

When the Information About Revisions and Variants is Useful?

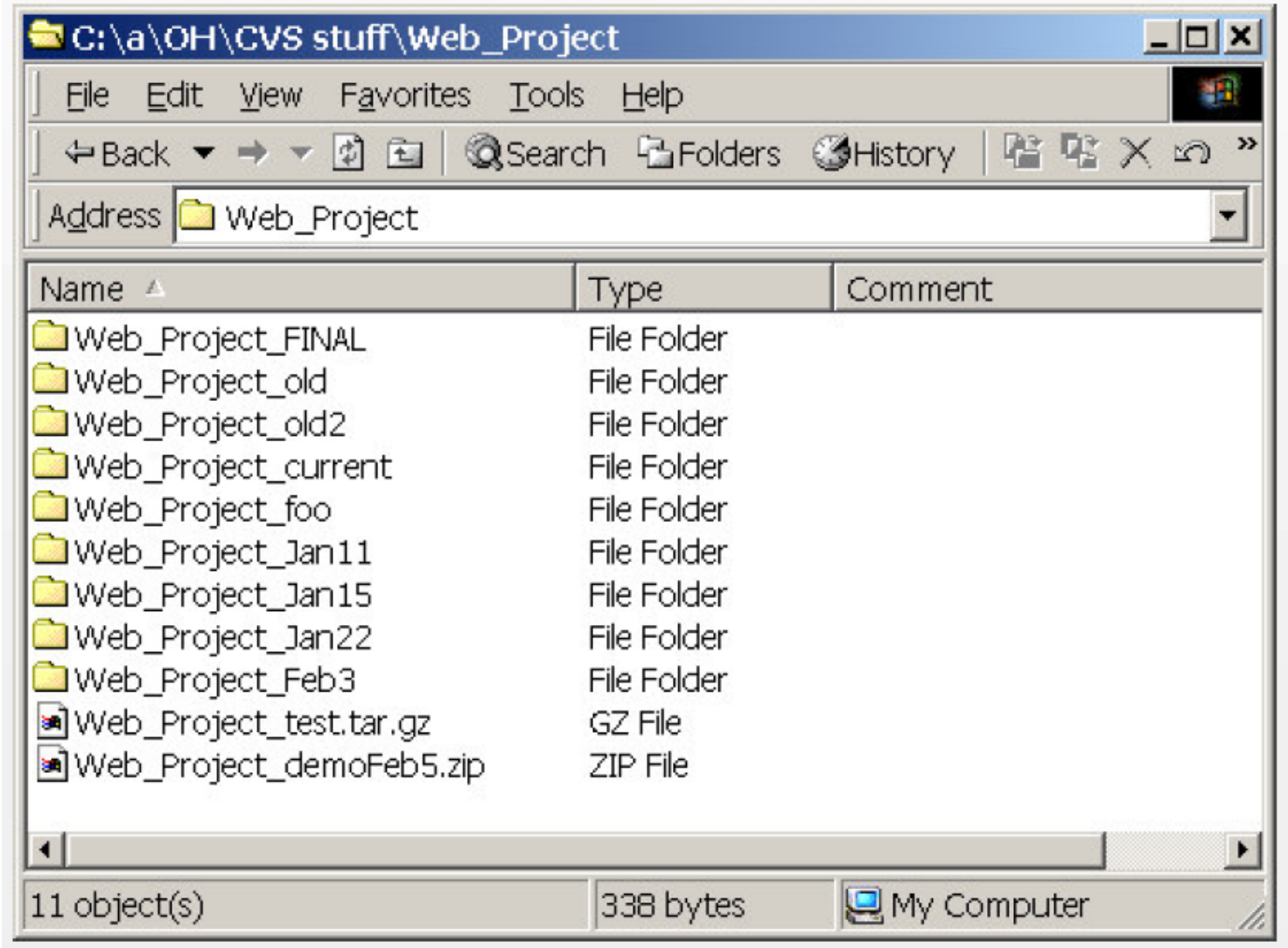


CASE for CM

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

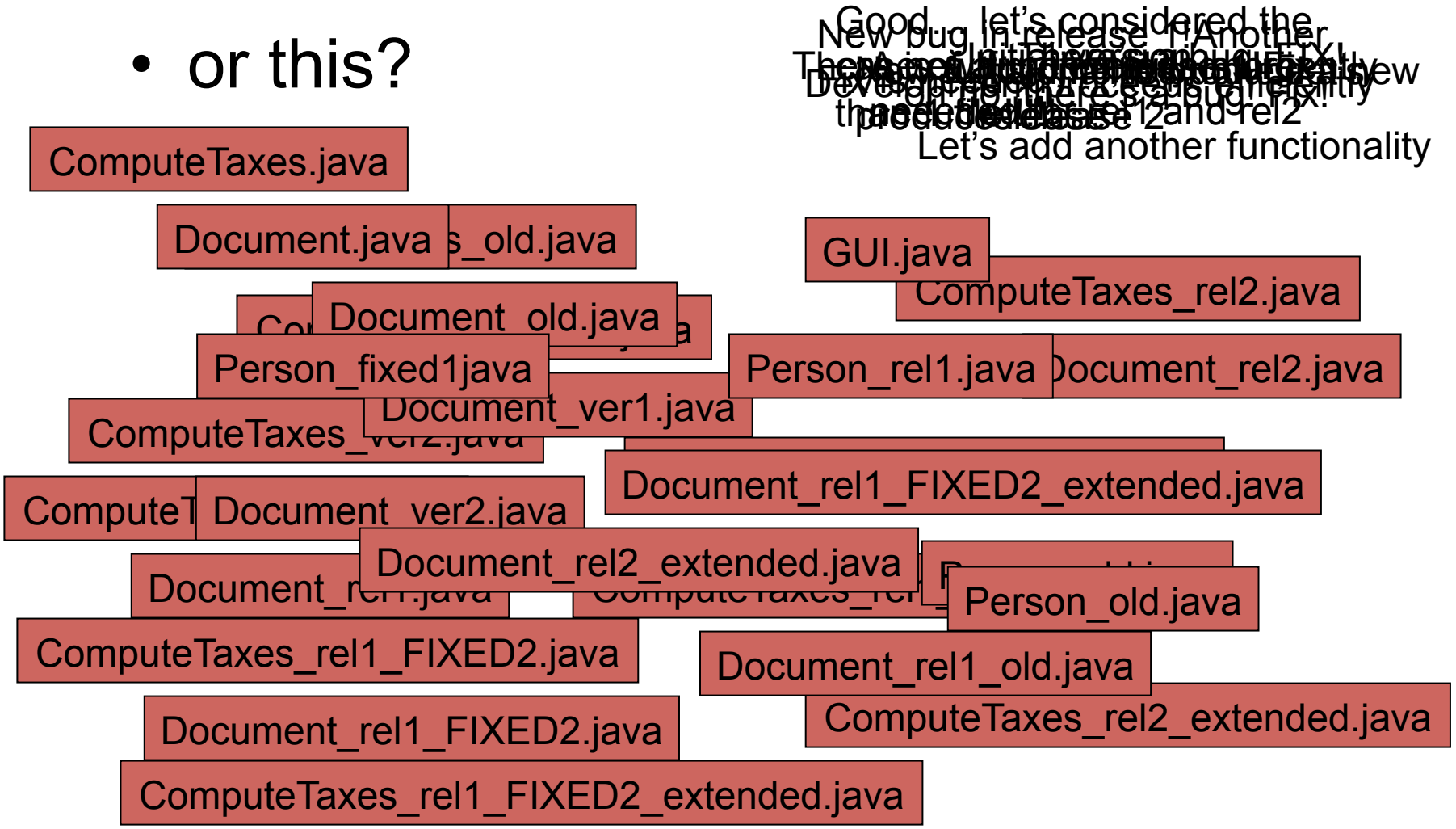
Version Management

Have you ever had something like that in you file system?



Version Management

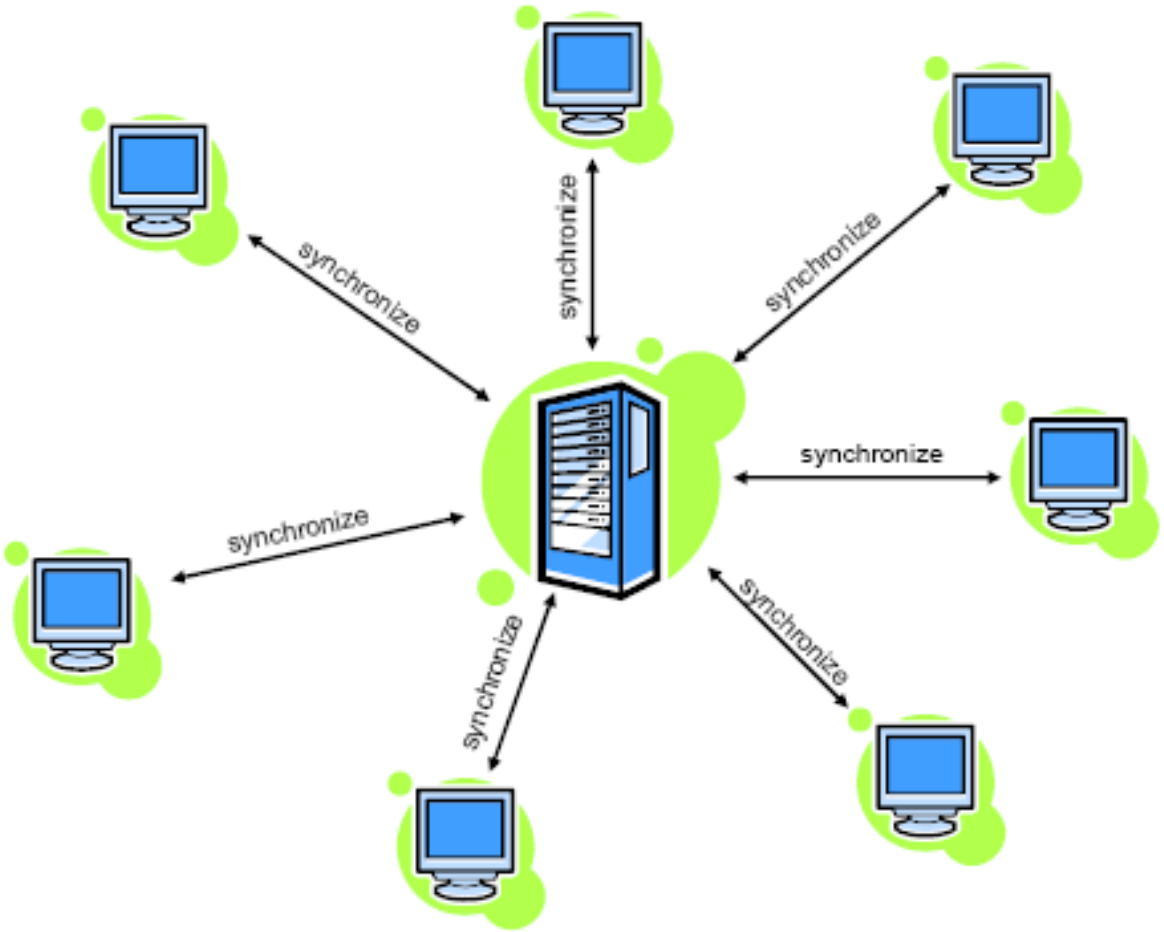
- or this?



Coordination

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

General Scenario



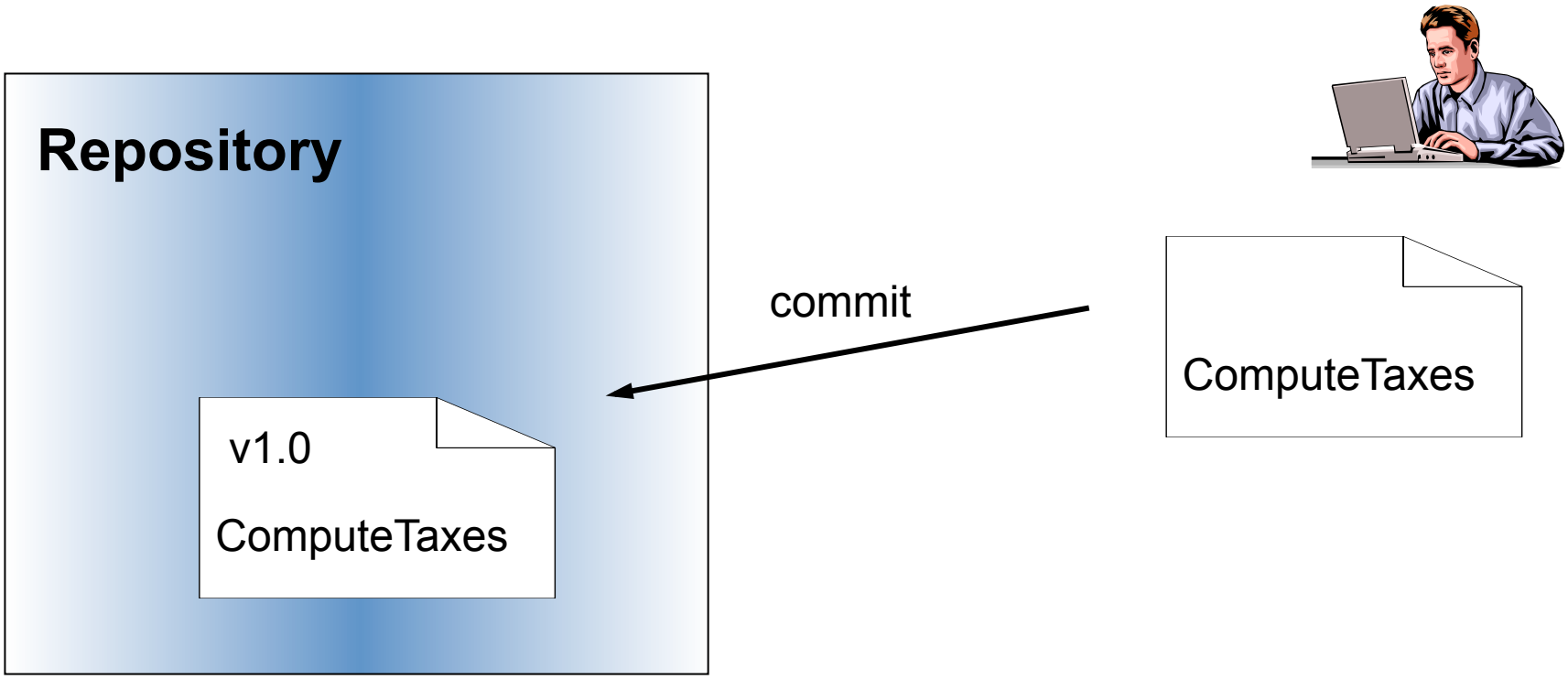
Single User Scenario

produce the initial version

Repository

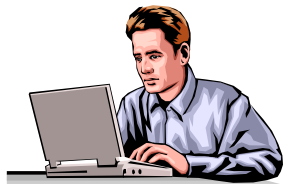
ComputeTaxes

Single User Scenario



Single User Scenario

extend!



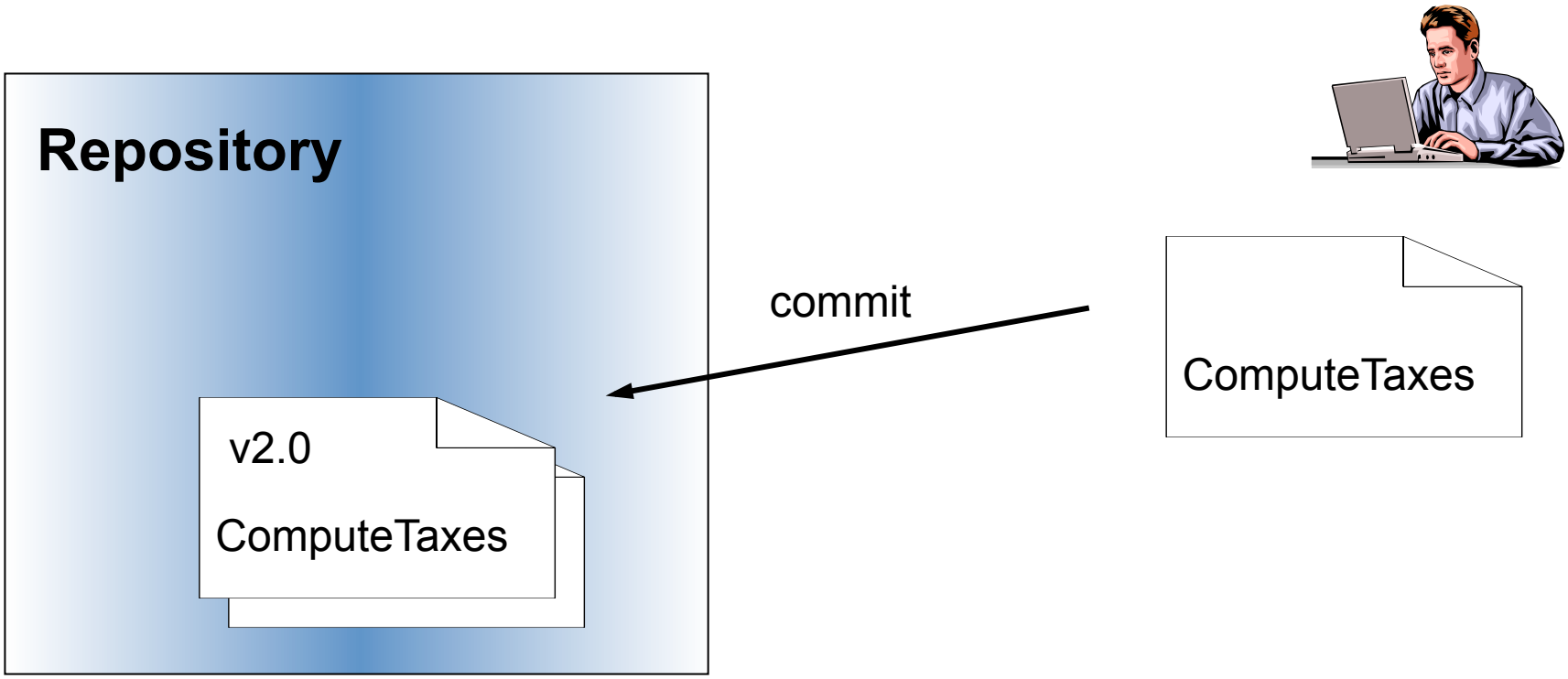
Repository

v1.0

ComputeTaxes

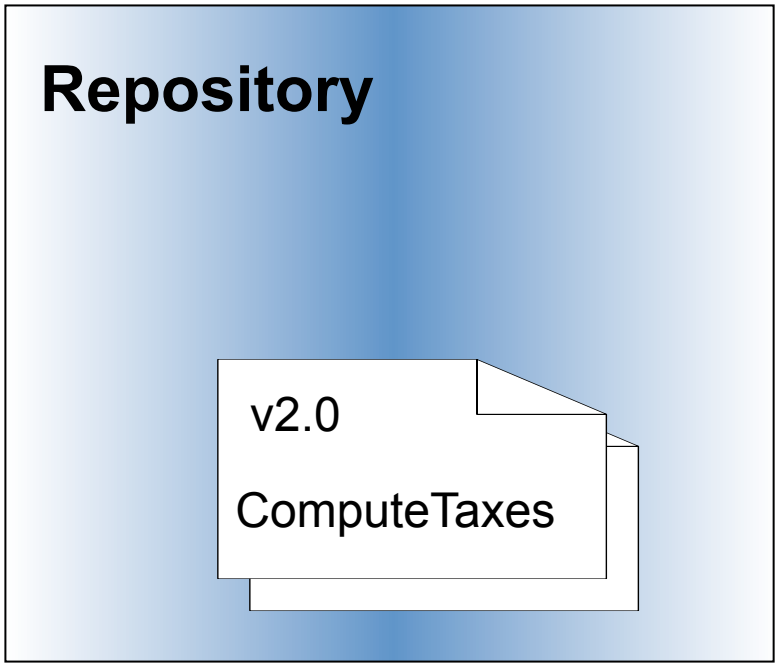
ComputeTaxes

Single User Scenario



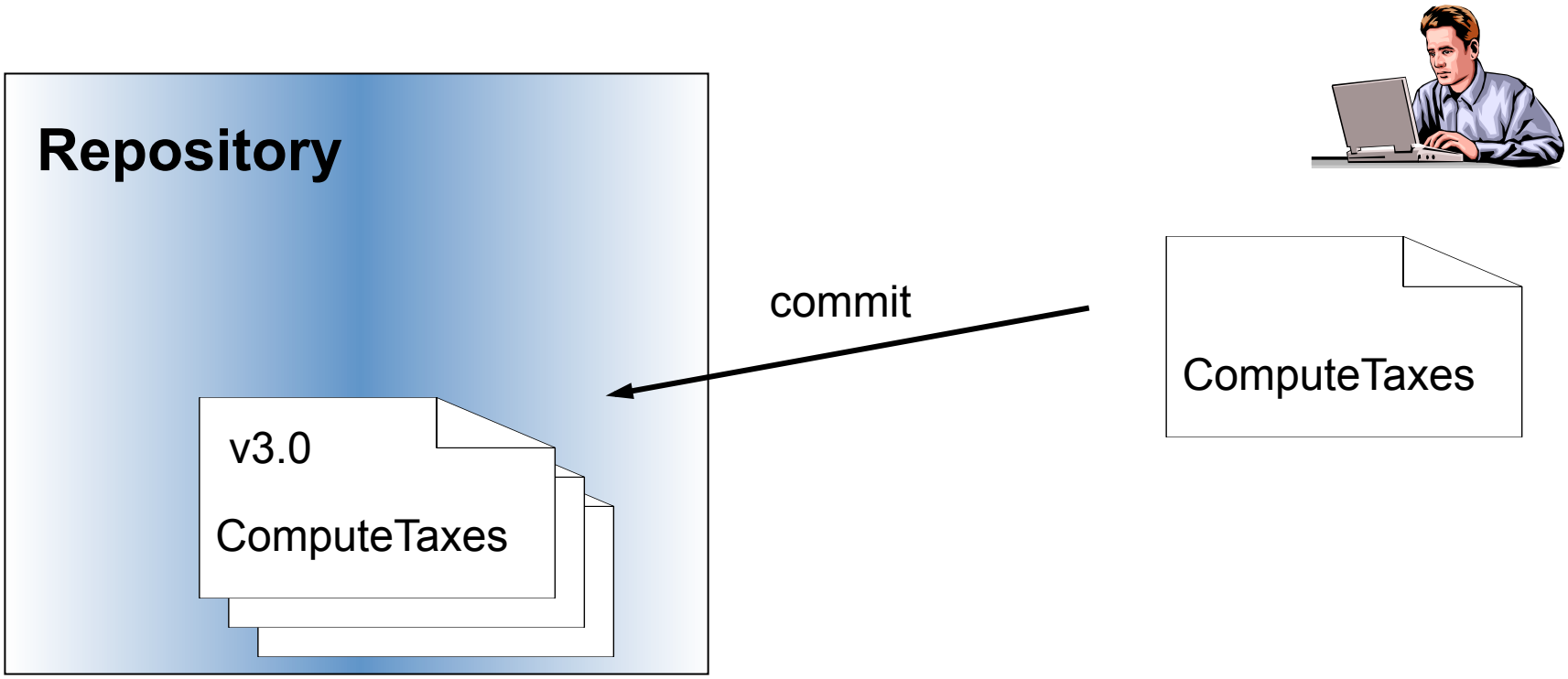
Single User Scenario

extend!



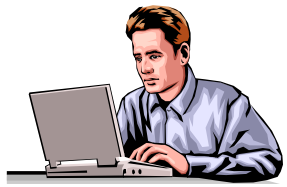
ComputeTaxes

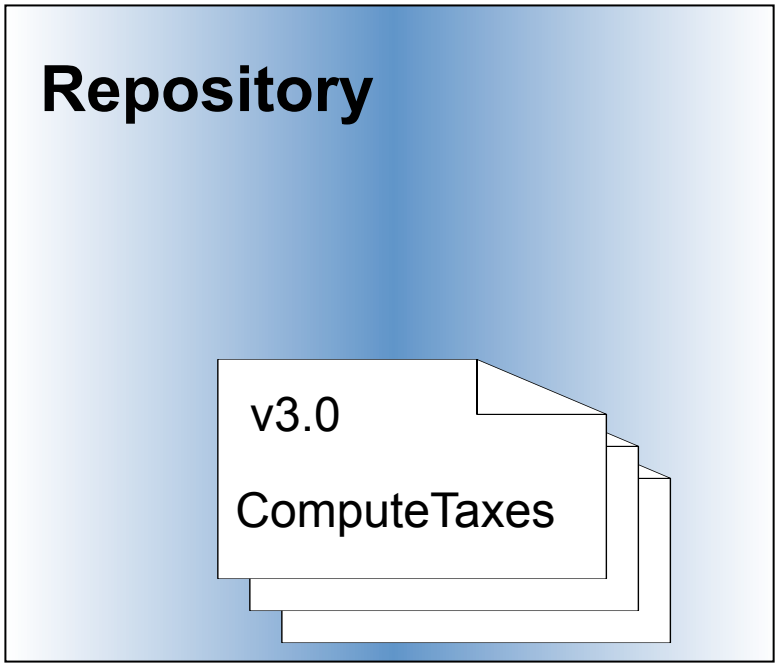
Single User Scenario



Single User Scenario

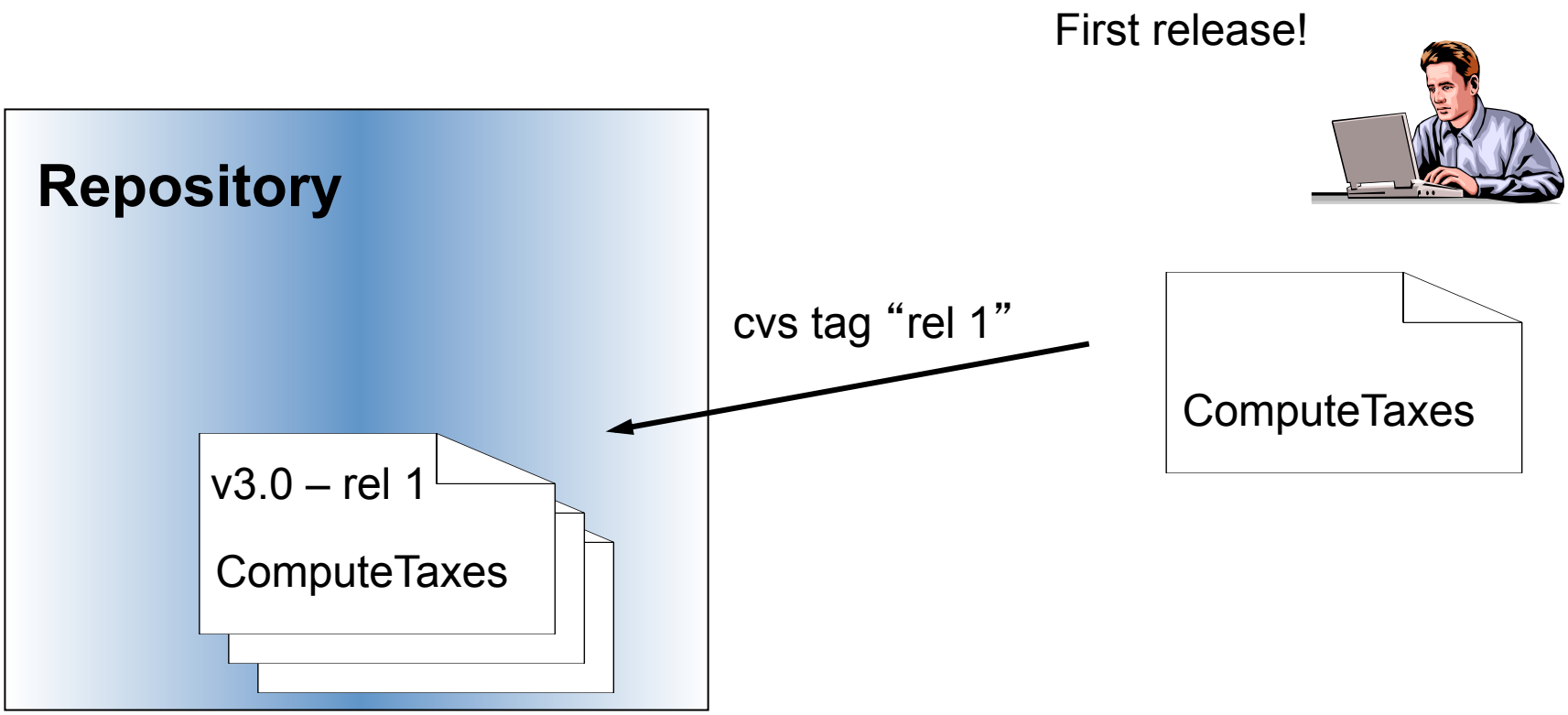
First release!



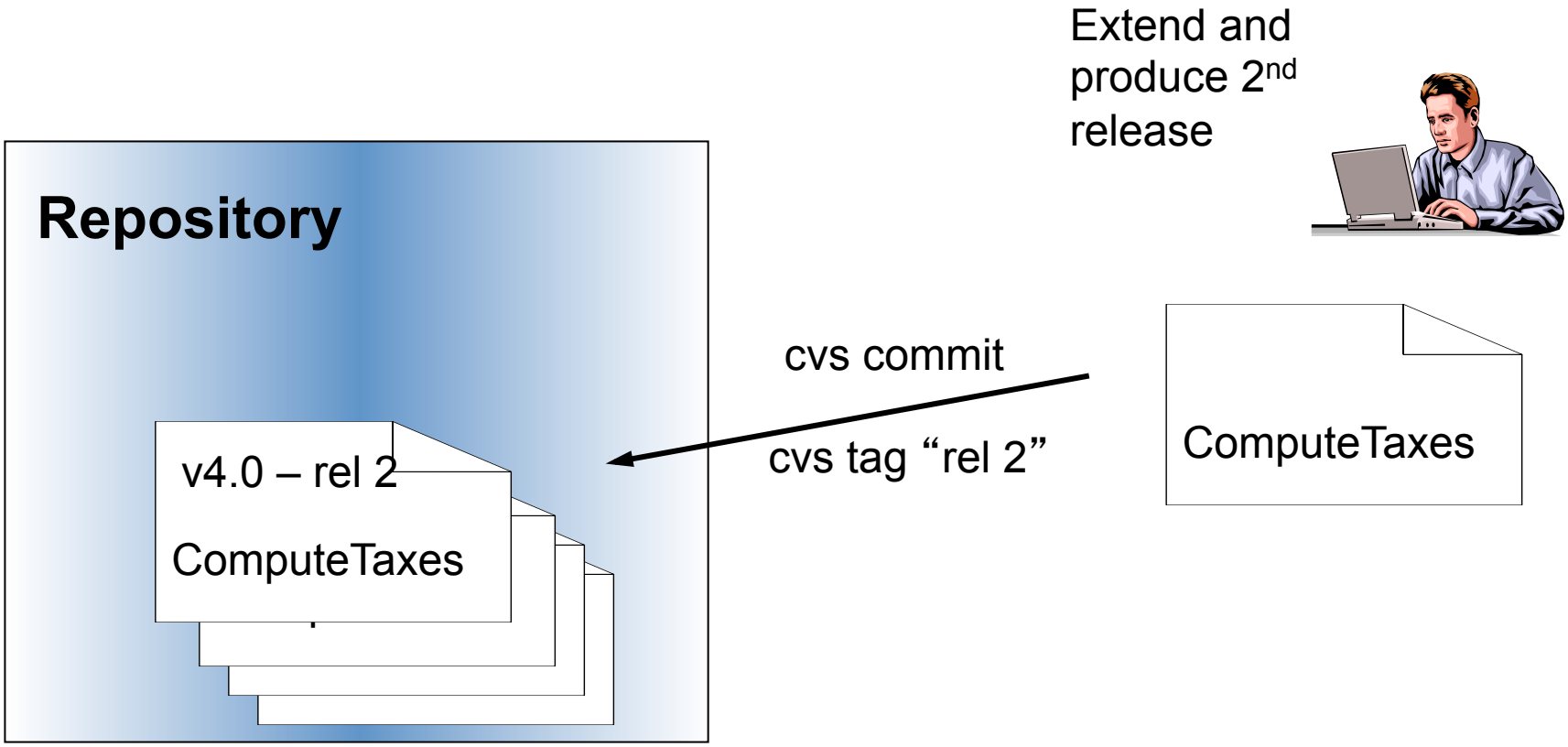


ComputeTaxes

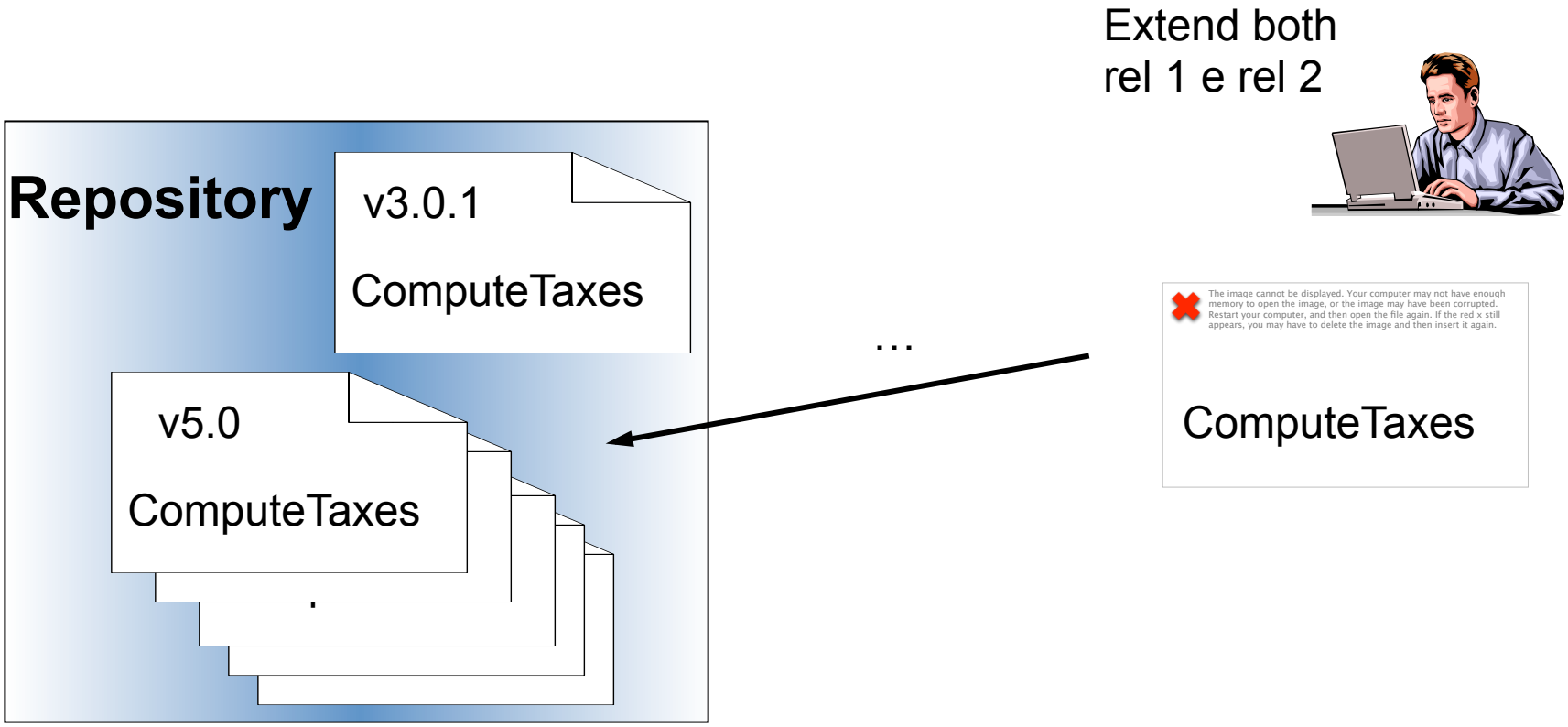
Single User Scenario



Single User Scenario

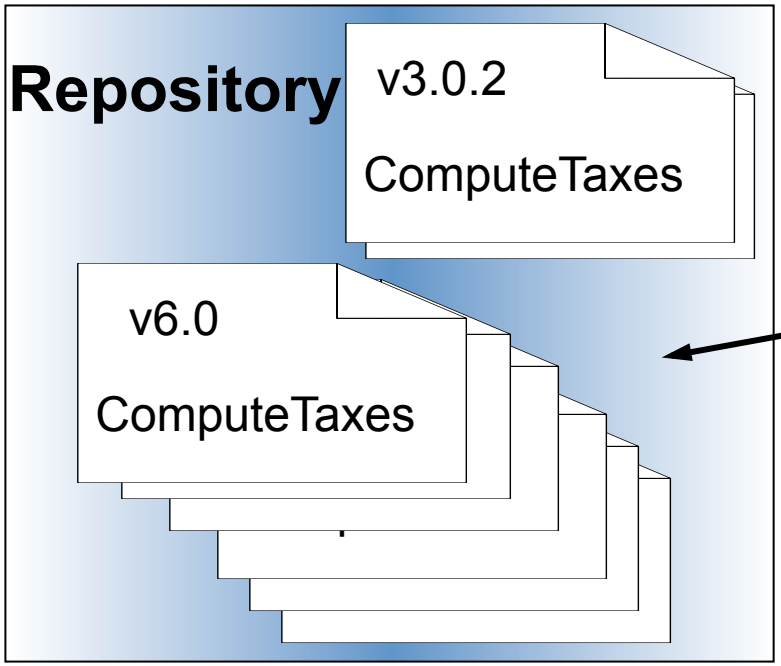


Single User Scenario



Single User Scenario

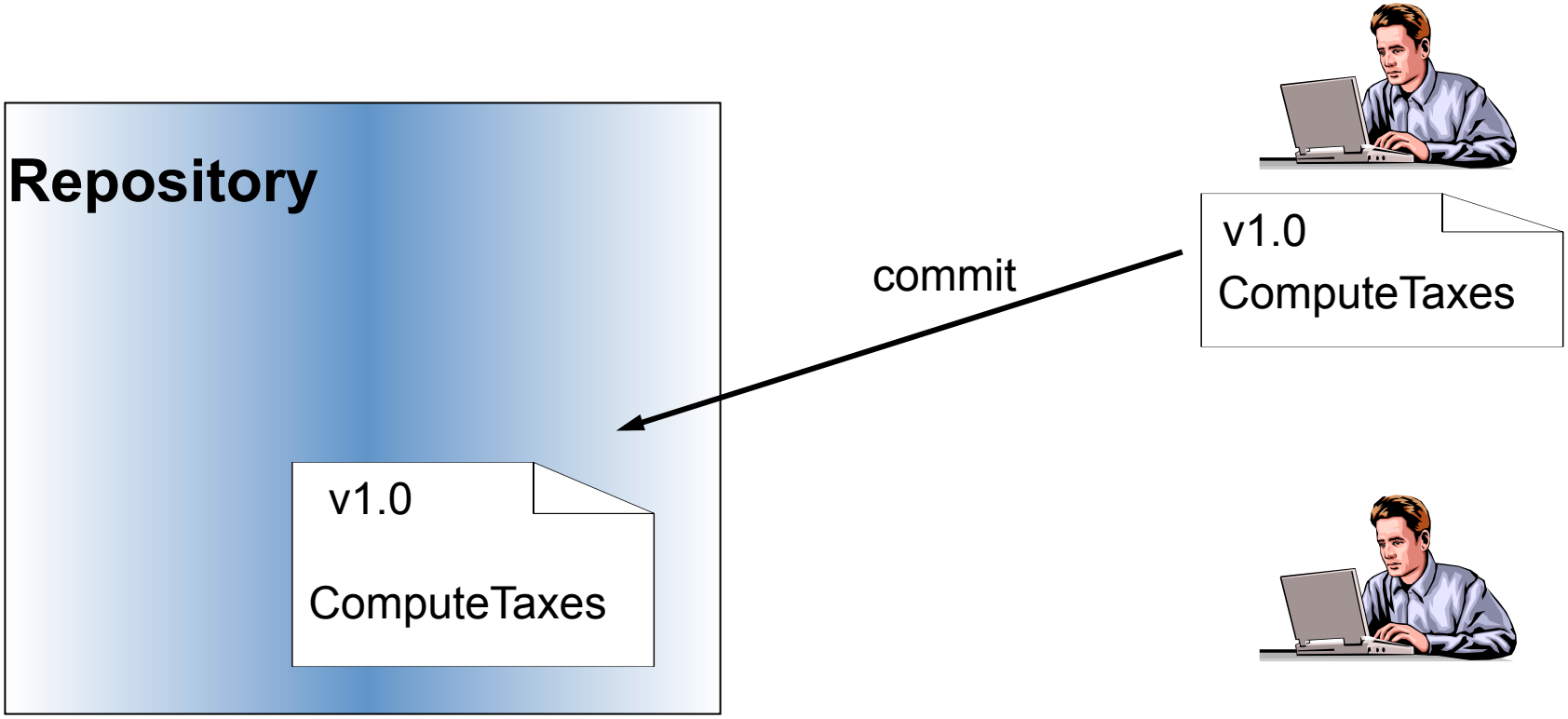
Further develop both branches



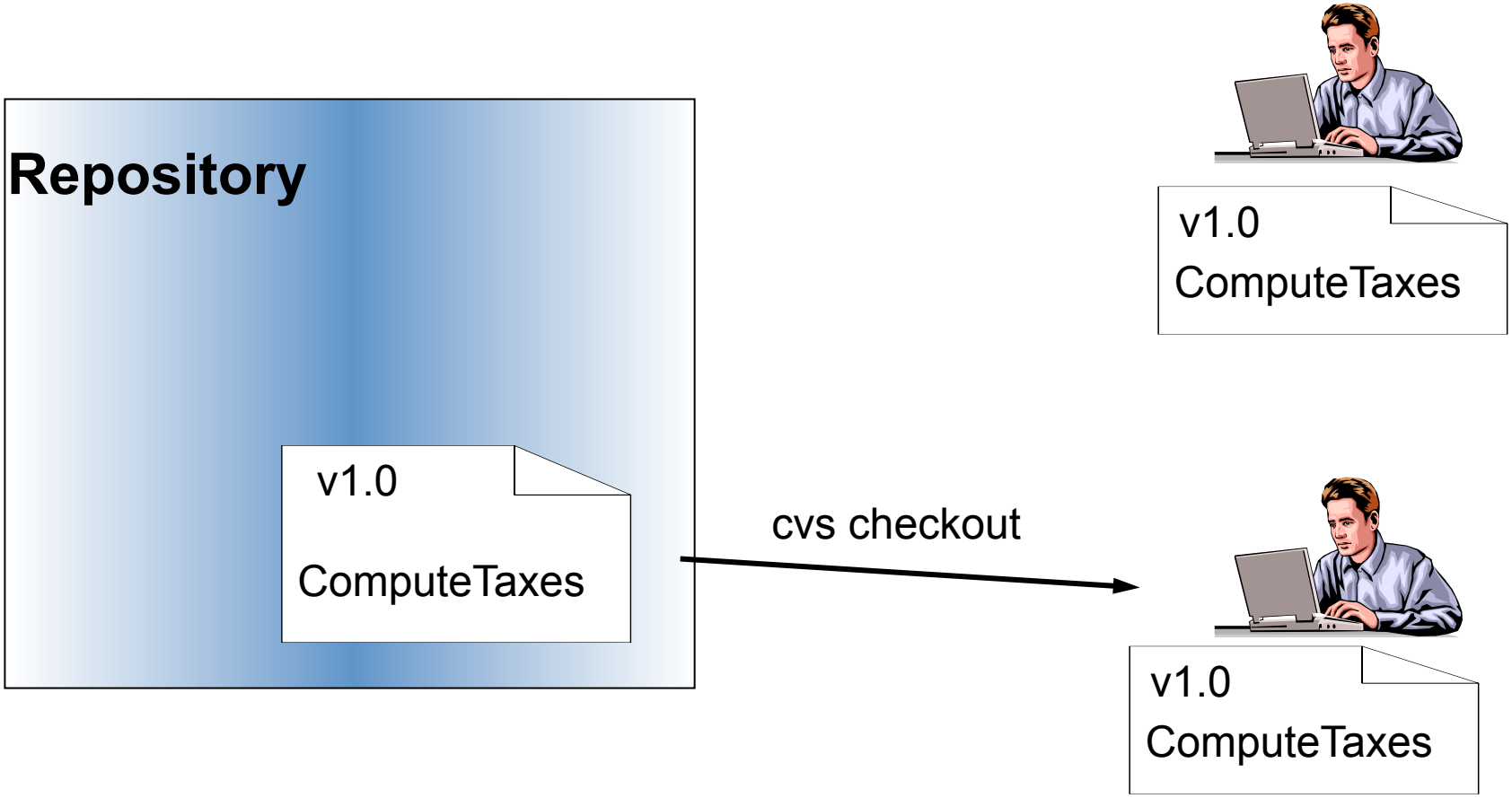
…

ComputeTaxes

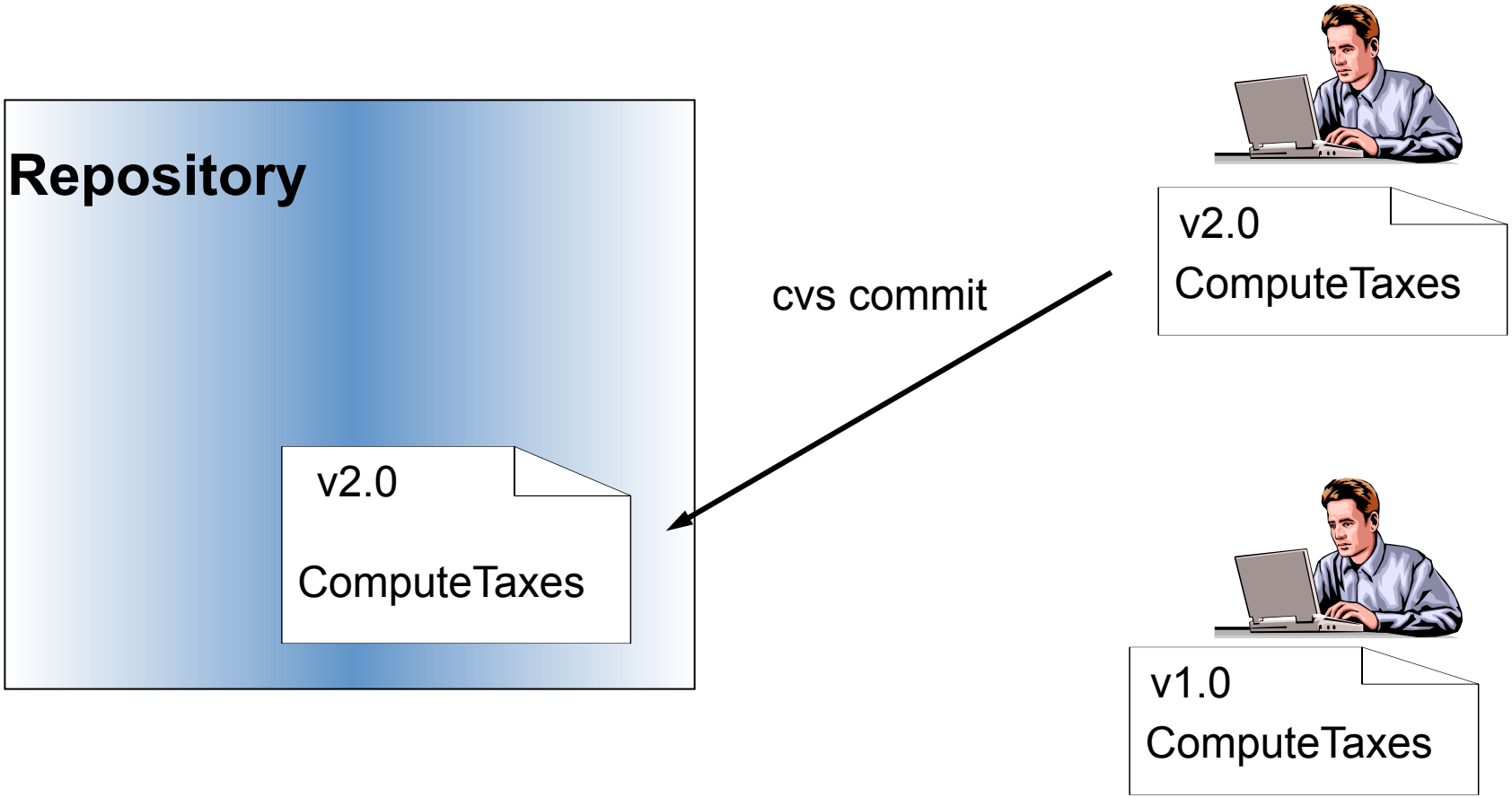
Multi User Scenario



Multi User Scenario

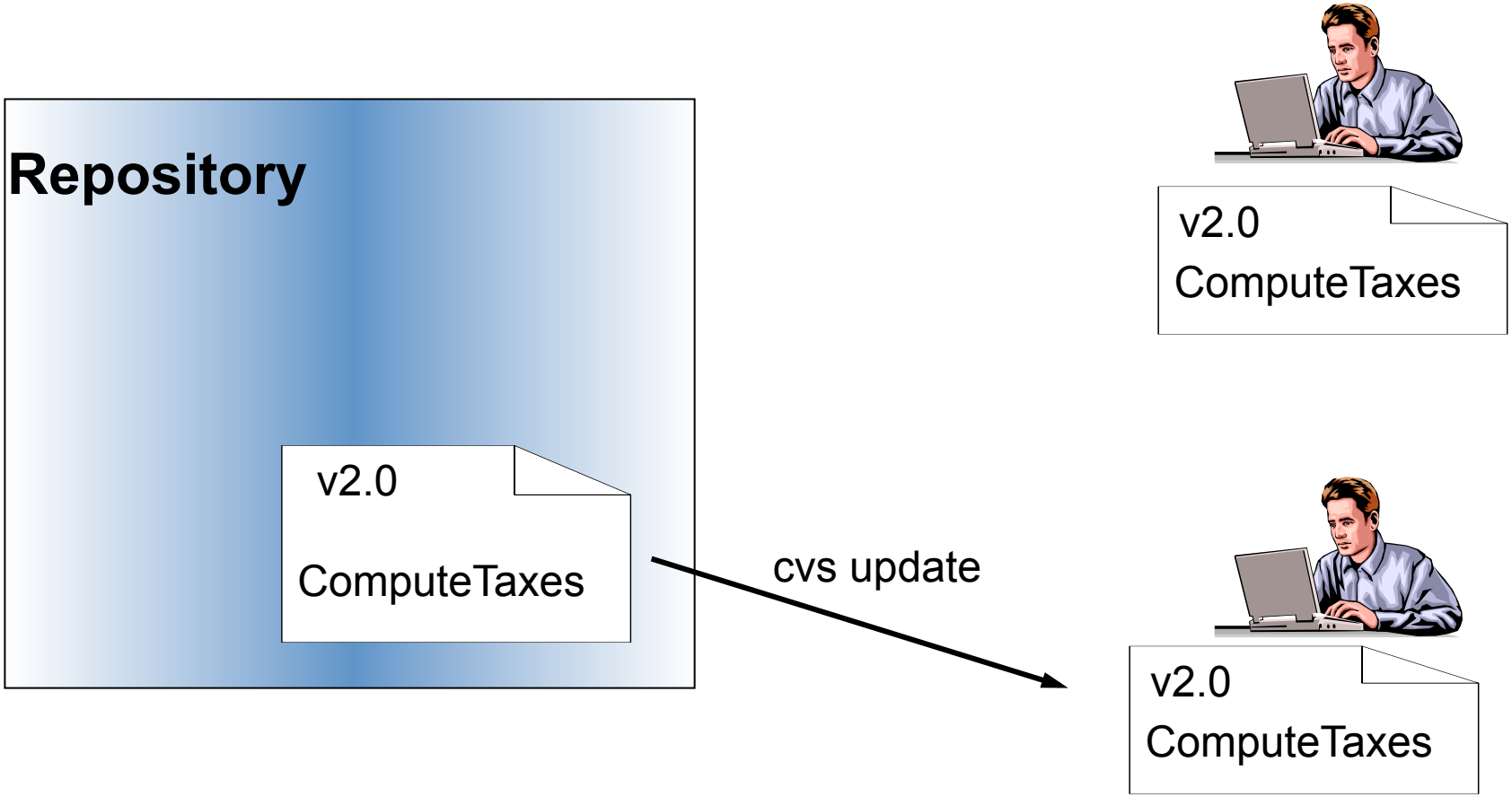


Multi User Scenario



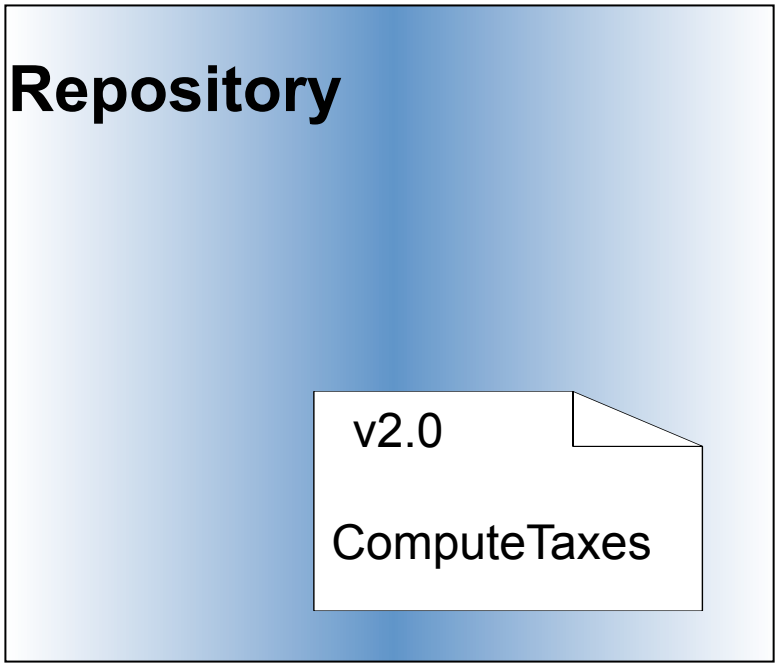
Multi User Scenario

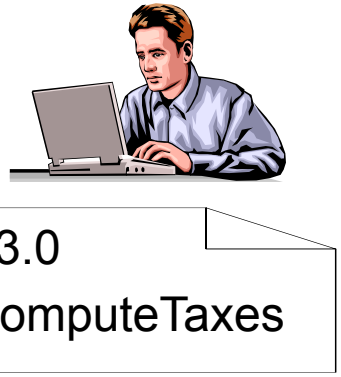
When starting a new working session, the user has to execute an update first!

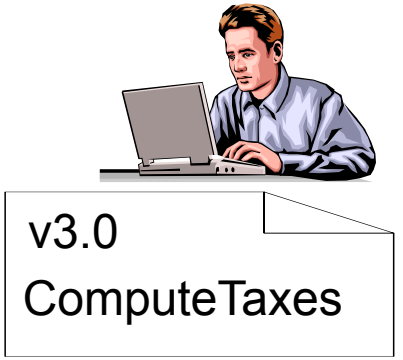


Multi User Scenario

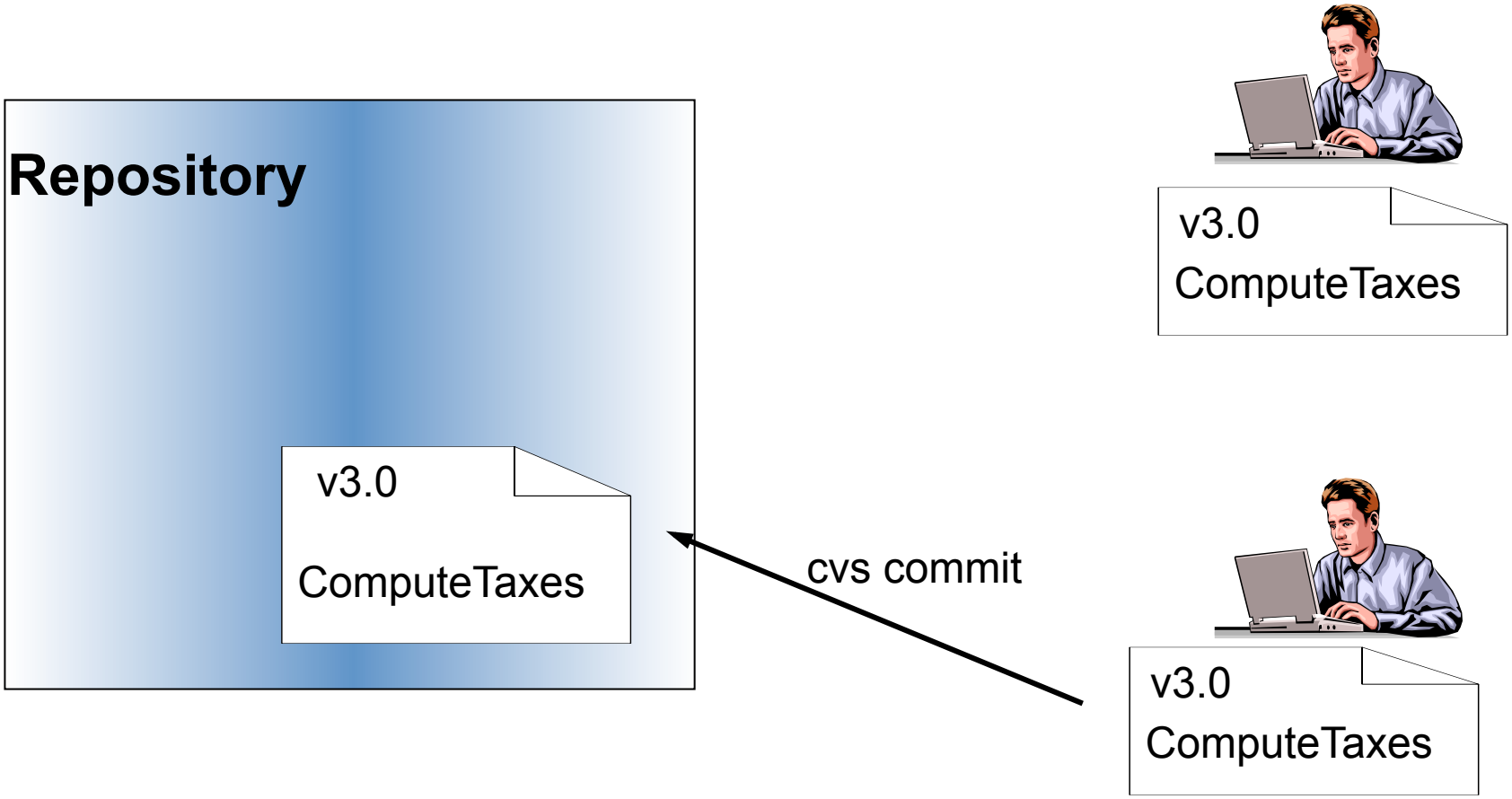
Developers work in parallel





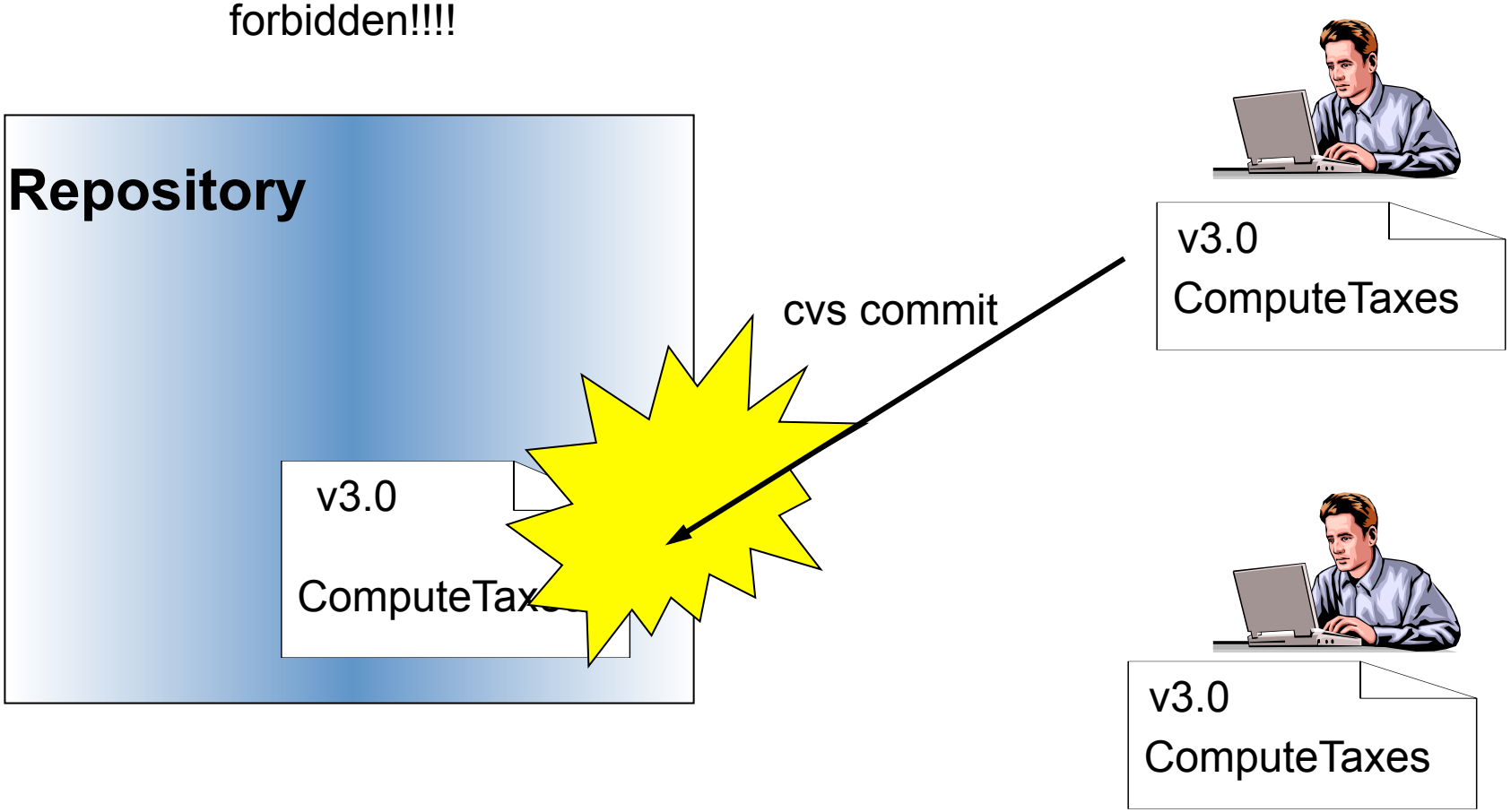


Multi User Scenario

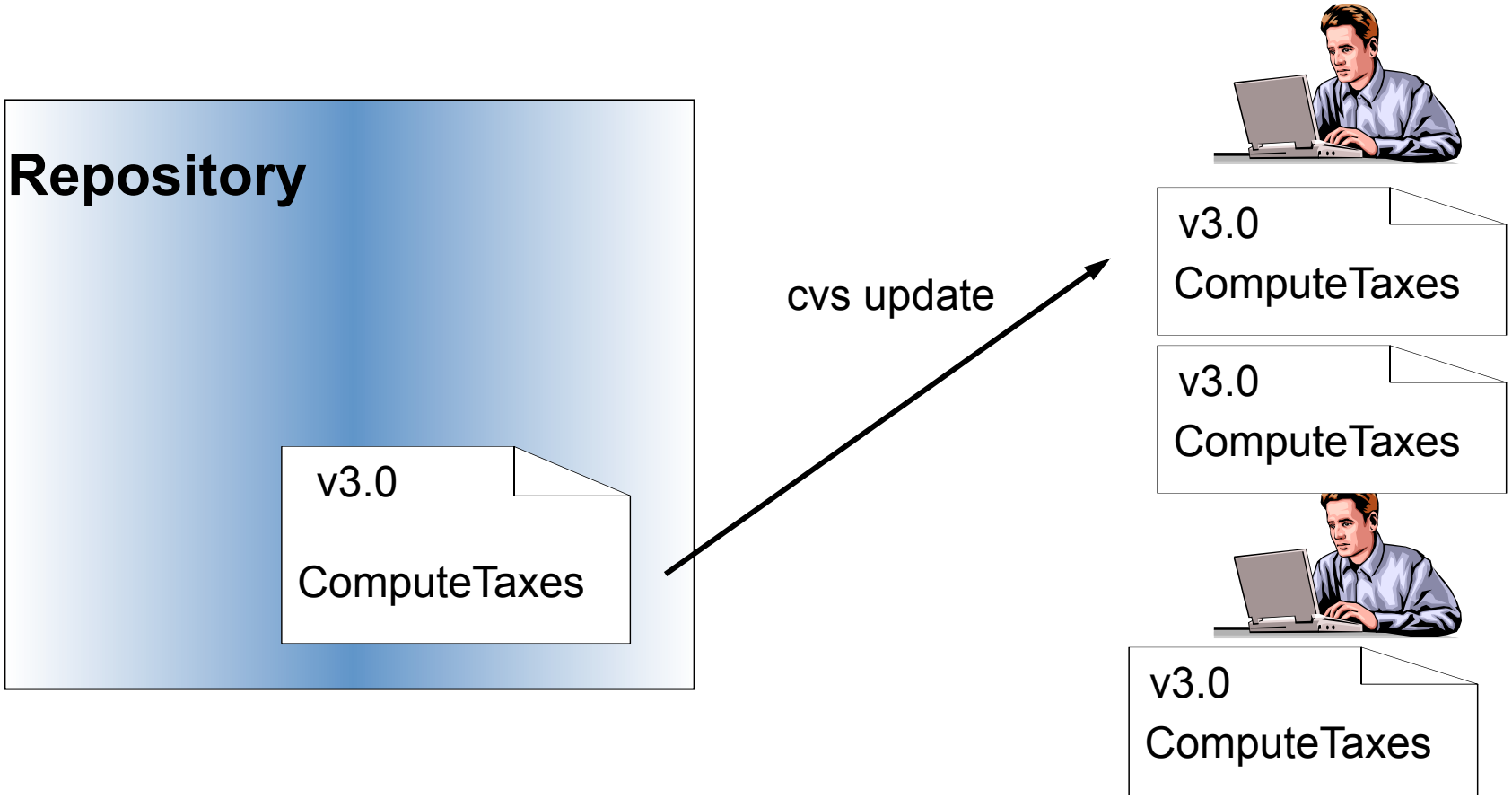


Multi User Scenario

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

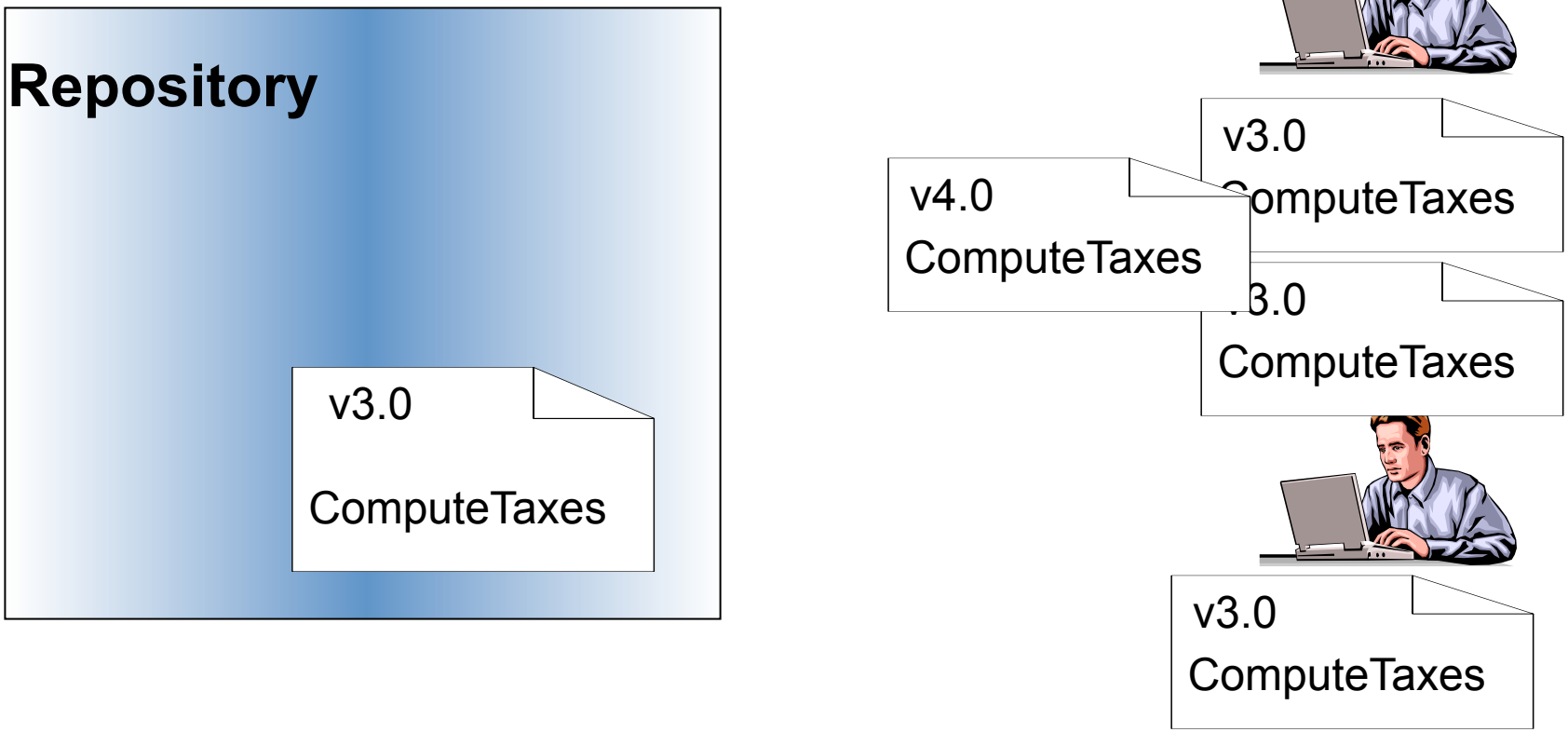


Multi User Scenario

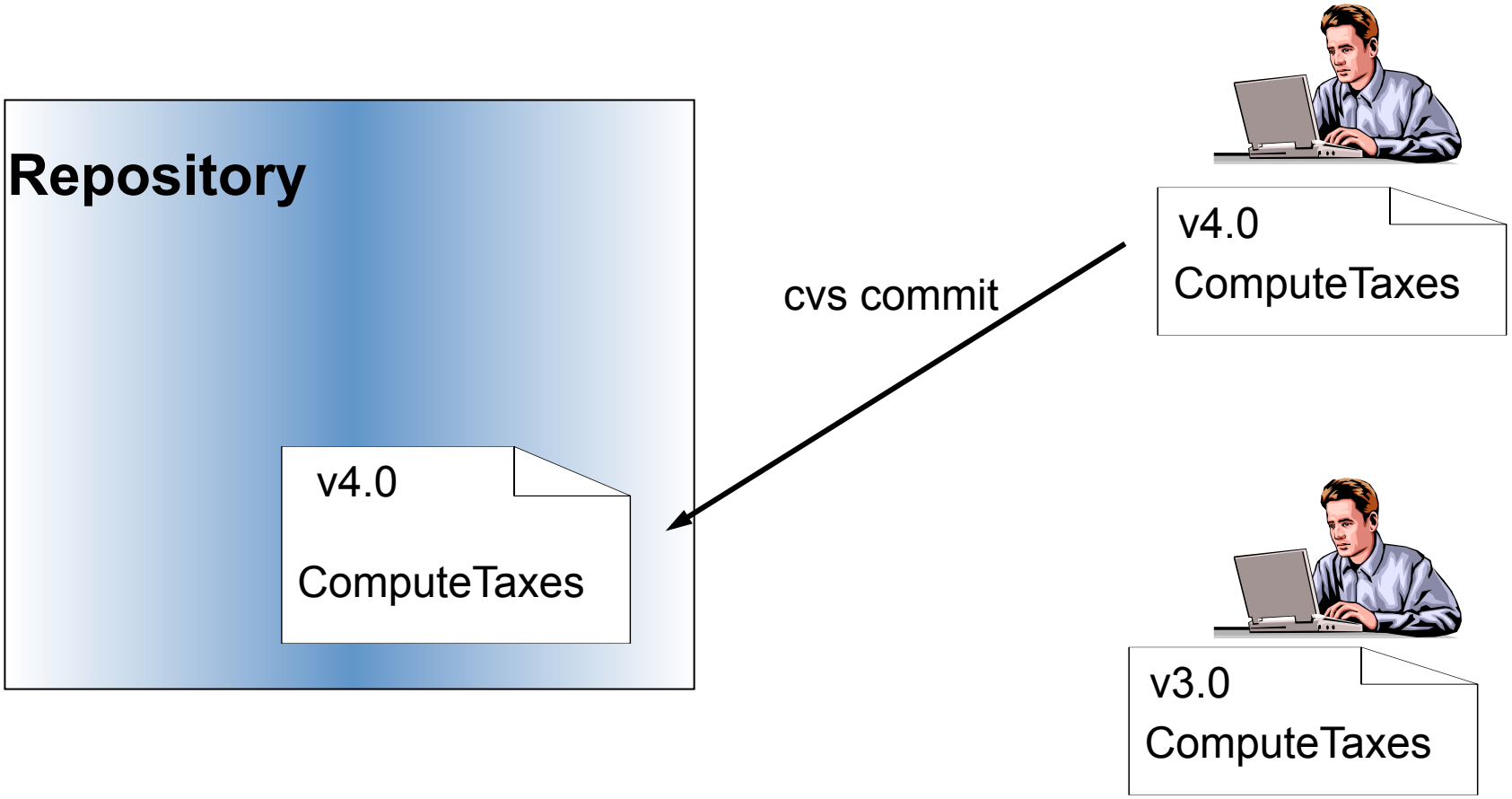


Multi User Scenario

Manual resolution of
the conflicts

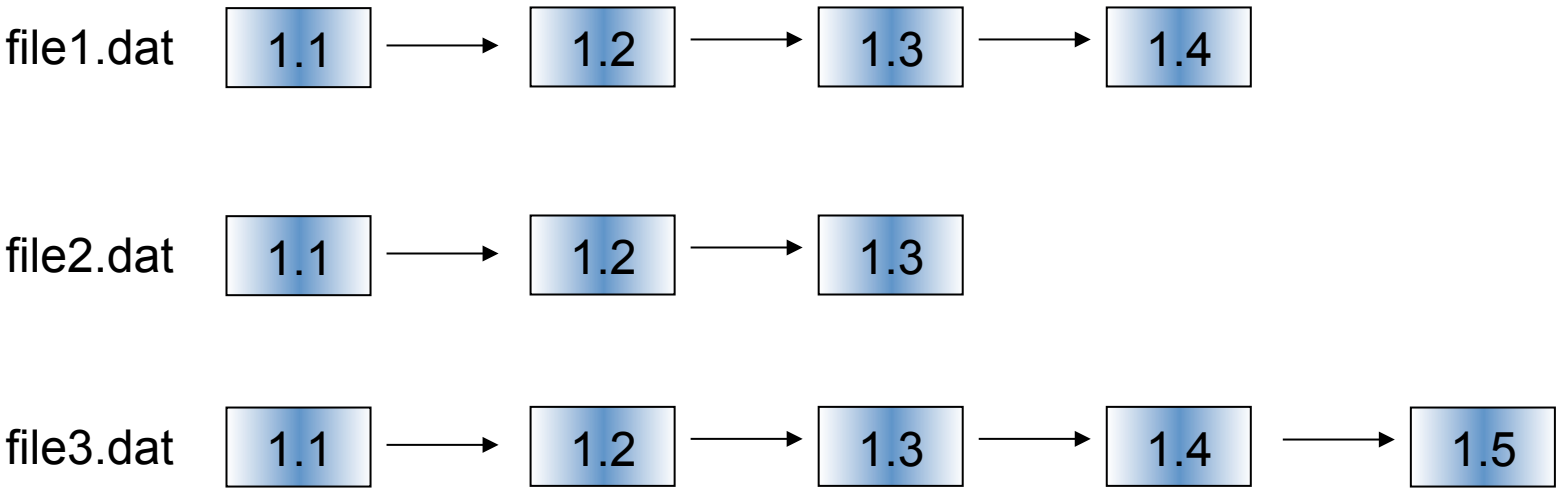


Multi User Scenario



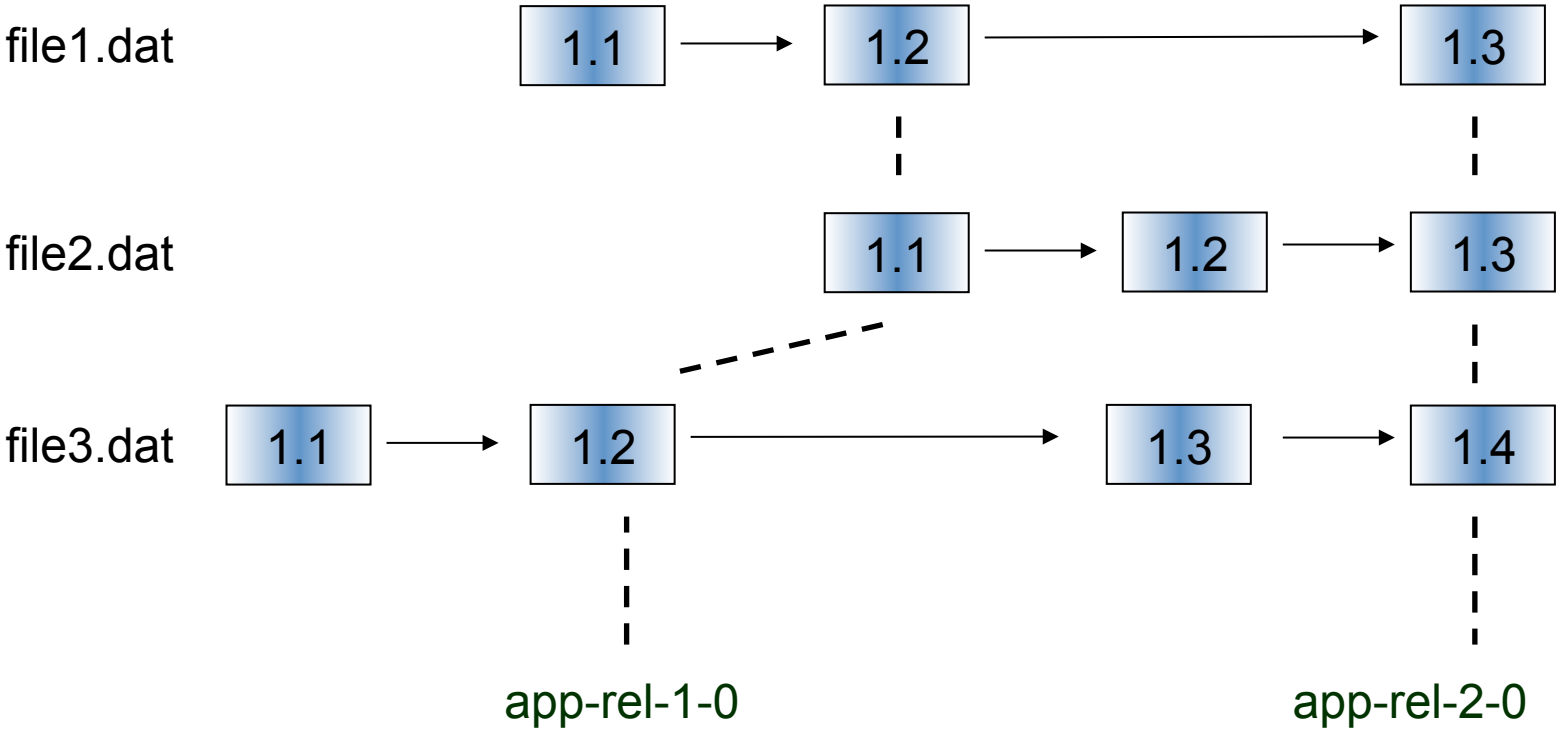
Structure of the Repository - Revisions

- A same file occurs in multiple revisions



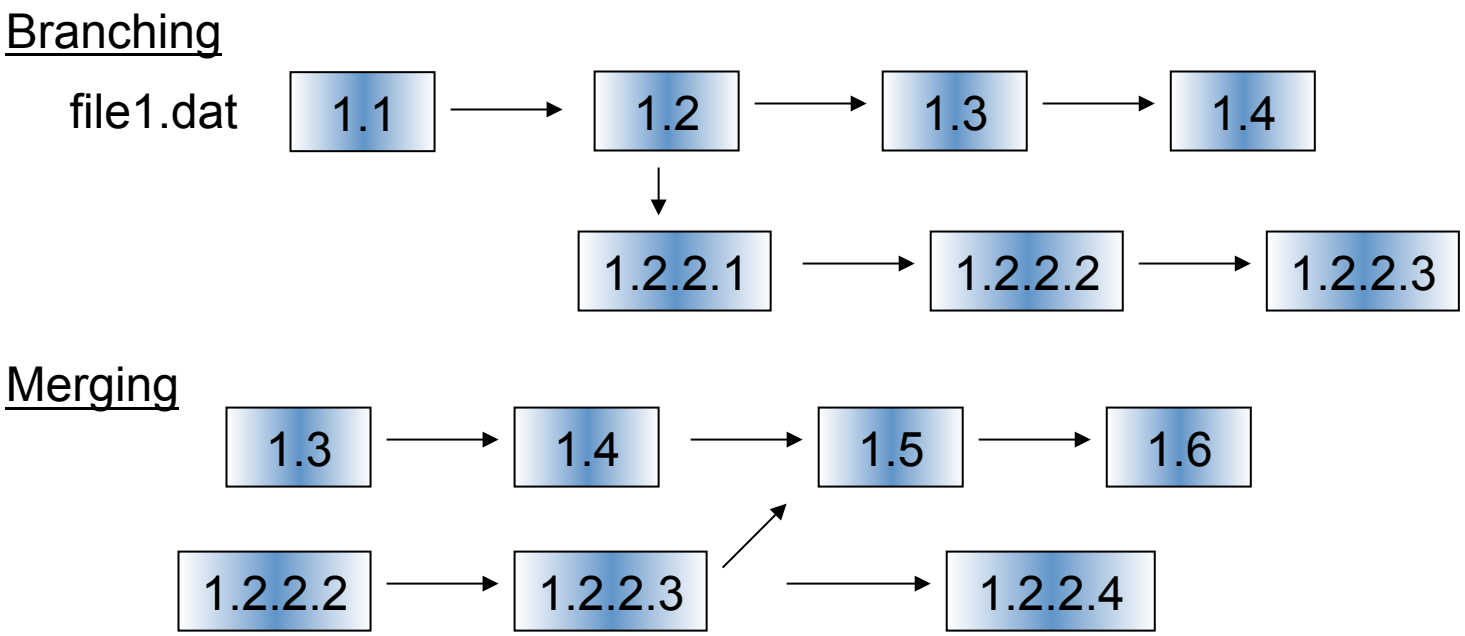
Structure of the Repository - Versions

- 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 - Branches

- 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

DEMO

- CVS by example…

SVN vs CVS

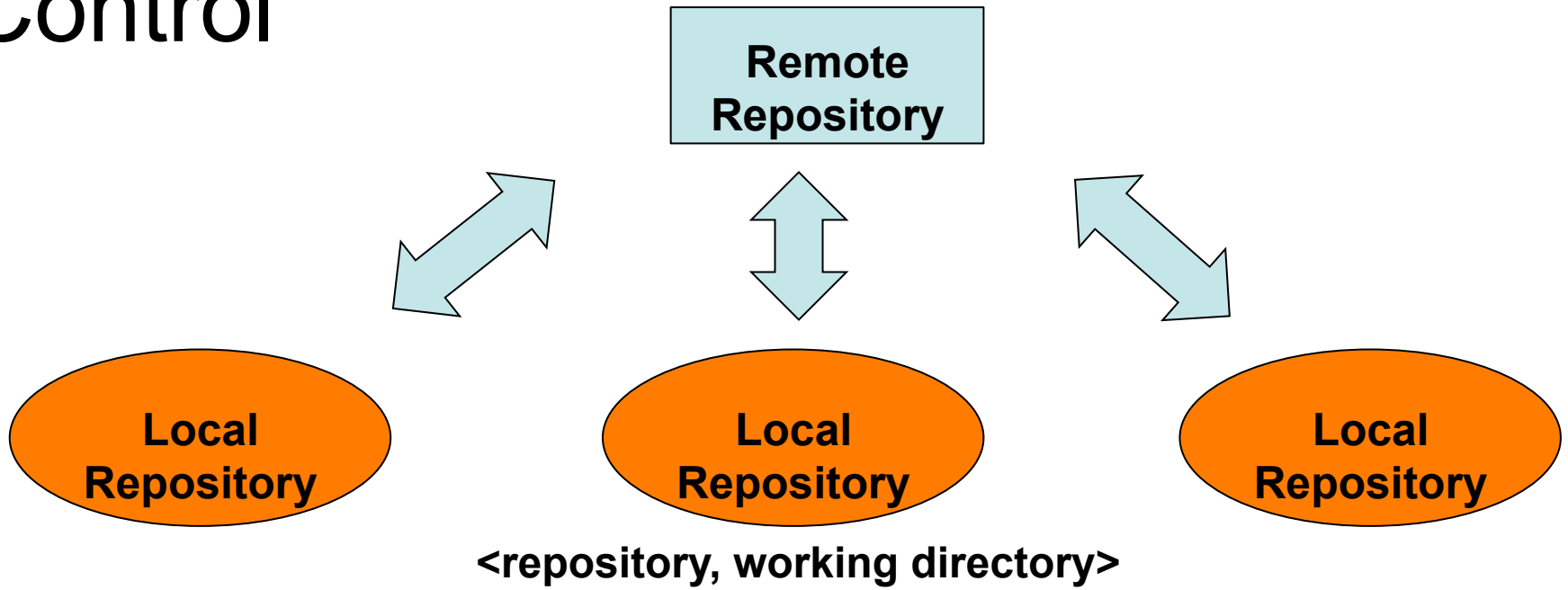
Pros SVN

- Atomic commit
- SVN implementations perform typically better than CVS implementations
- SVN efficiently stores binary files
- (show GUI)

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

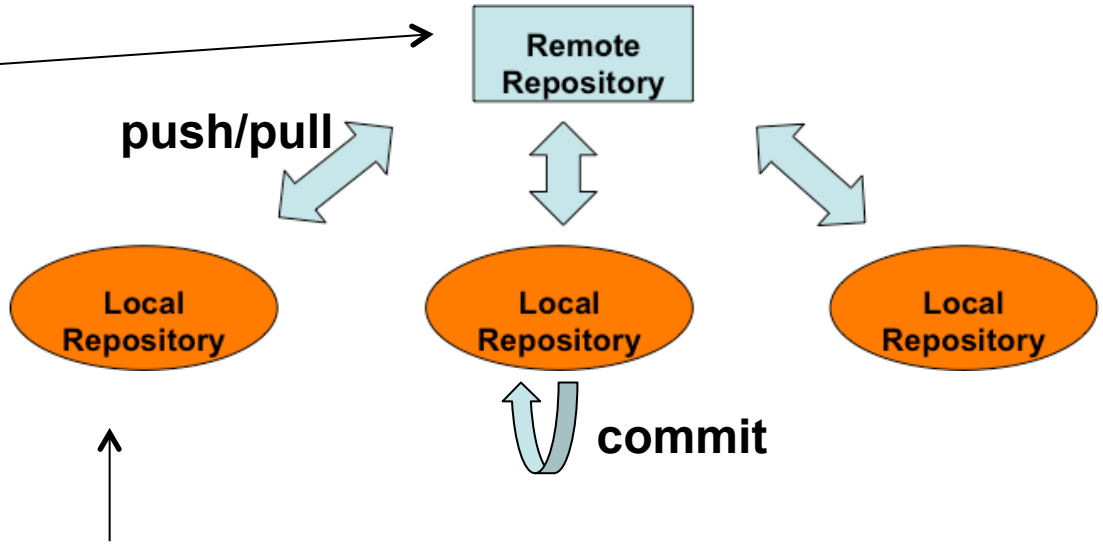
GIT: From Client-Server to Distributed Version Control



GIT

- Distributed version control
- Used by many popular open source projects

Bare repository: shared by multiple developers

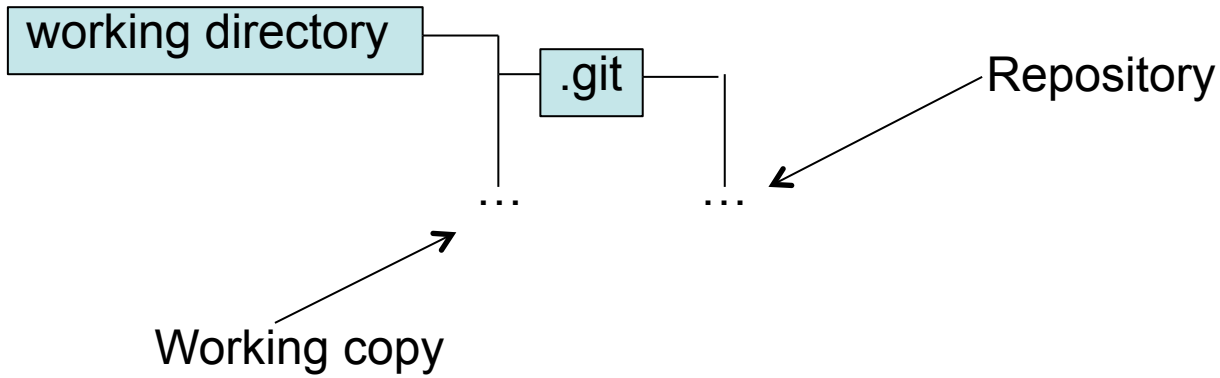


Non-Bare repository: single developer repository

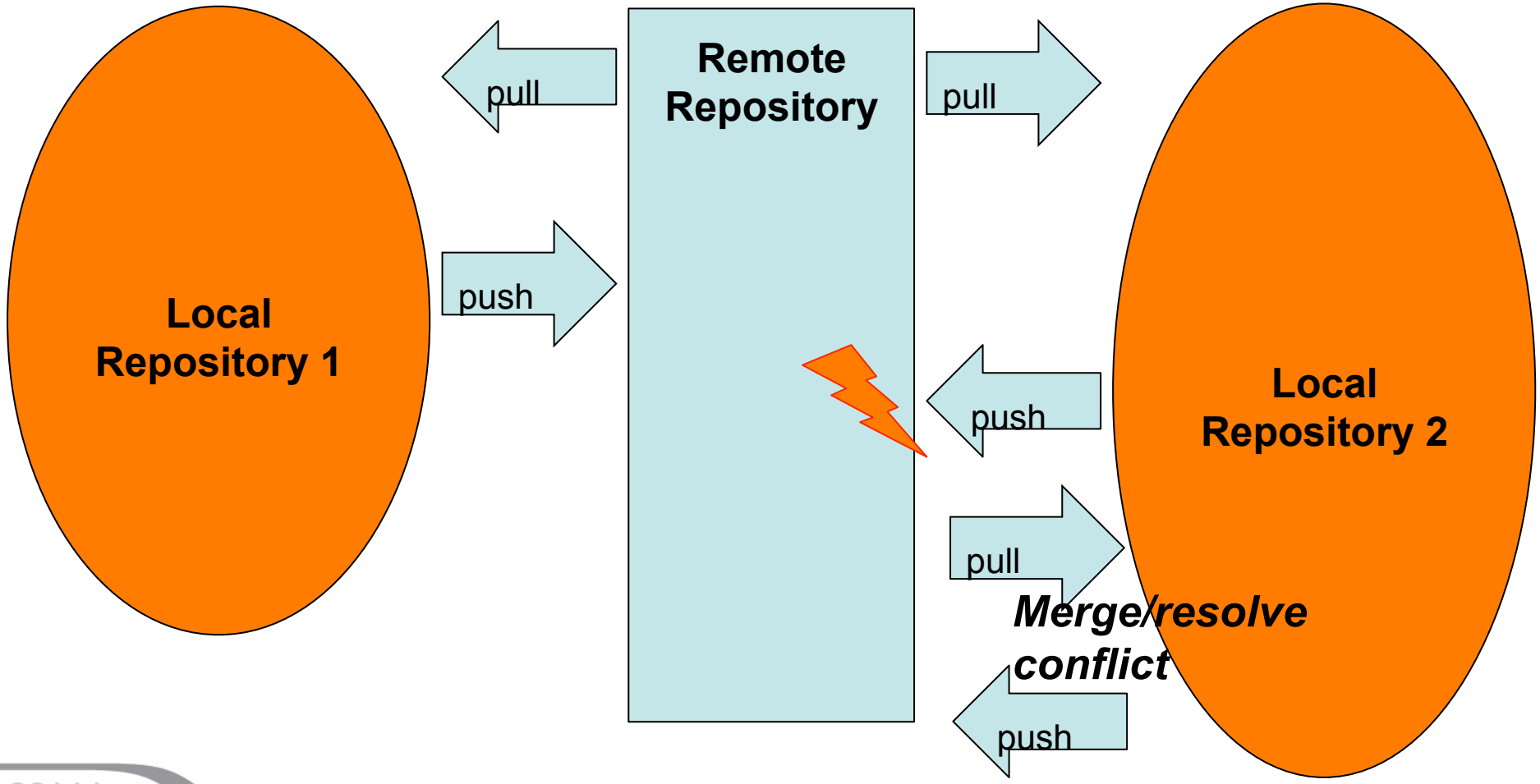
- There is almost nothing you cannot do locally

Create a GIT Local Repository

- Go into working directory
- Type
  - git init
  - git clone git://…



Conflicts


Local Repository 1
pull
push
Remote Repository
pull
push
pull
Merge/resolve conflict
push
Local Repository 2

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.)

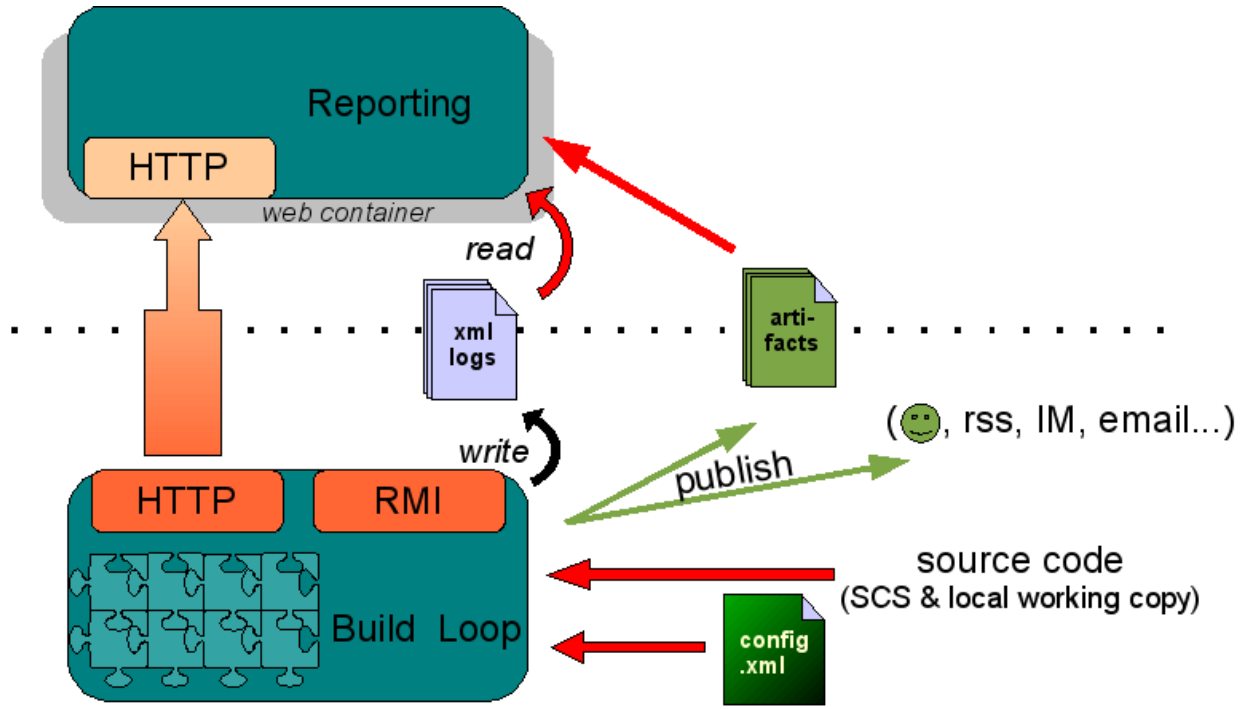
From Version Control to Continuous Integration

- 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?

Example: 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
  - Next part about testing and analysis
- Faults that are discovered during testing are documented and returned to the system developers

CruiseControl Build Loop



Take Home

- User Version Control Software
  - Even if you do not work in a team

- Consider continuous integration tools