EAST WEST UNIVERSITY

CSE412
Software Engineering

Nishat Tasnim Niloy

Lecturer

Department of Computer Science and Engineering

Faculty of Science and Engineering

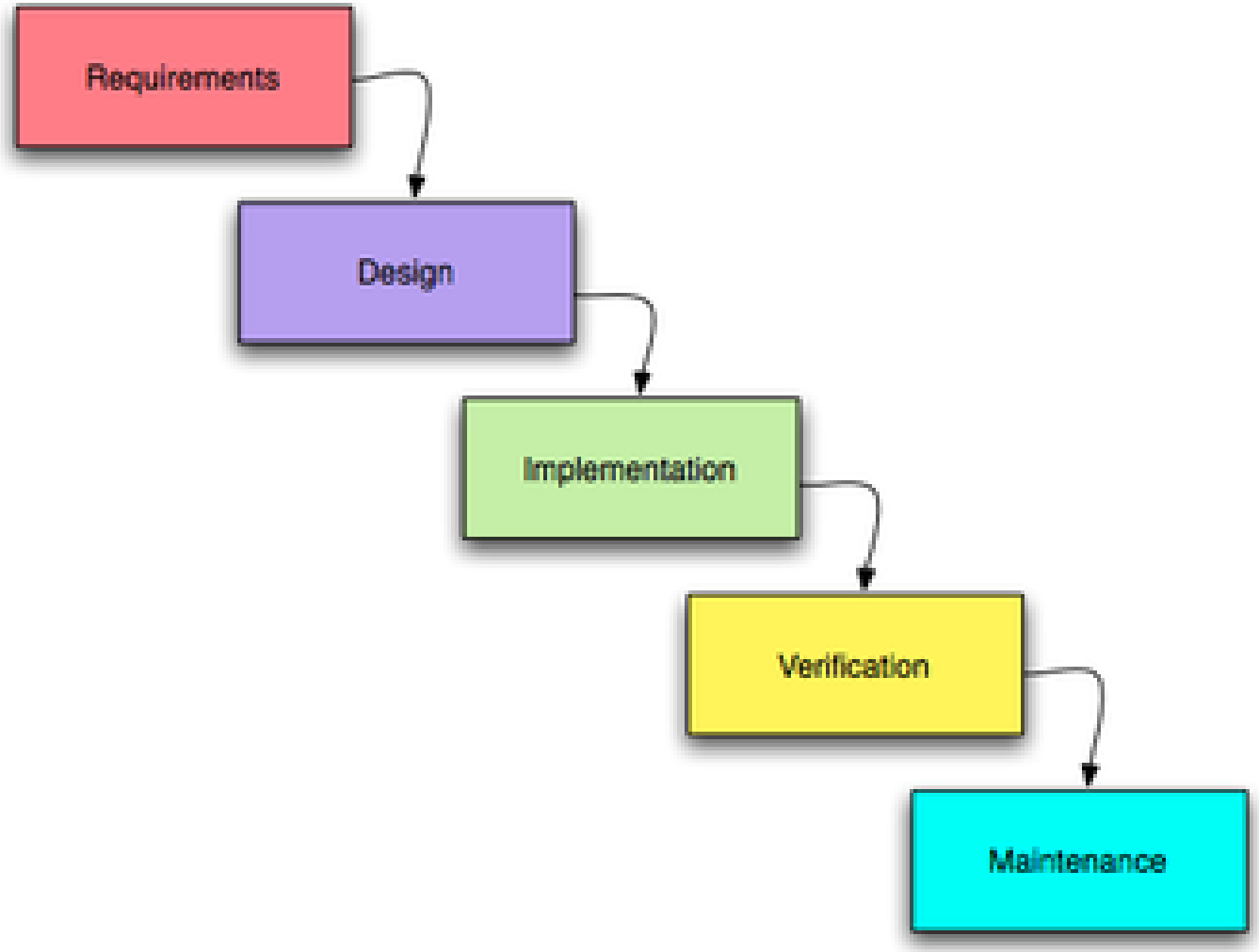
Topic 3

Source Code Management and version control system

Images for version controlling chapter is taken from: Image: http://svnbook.red-bean.com/

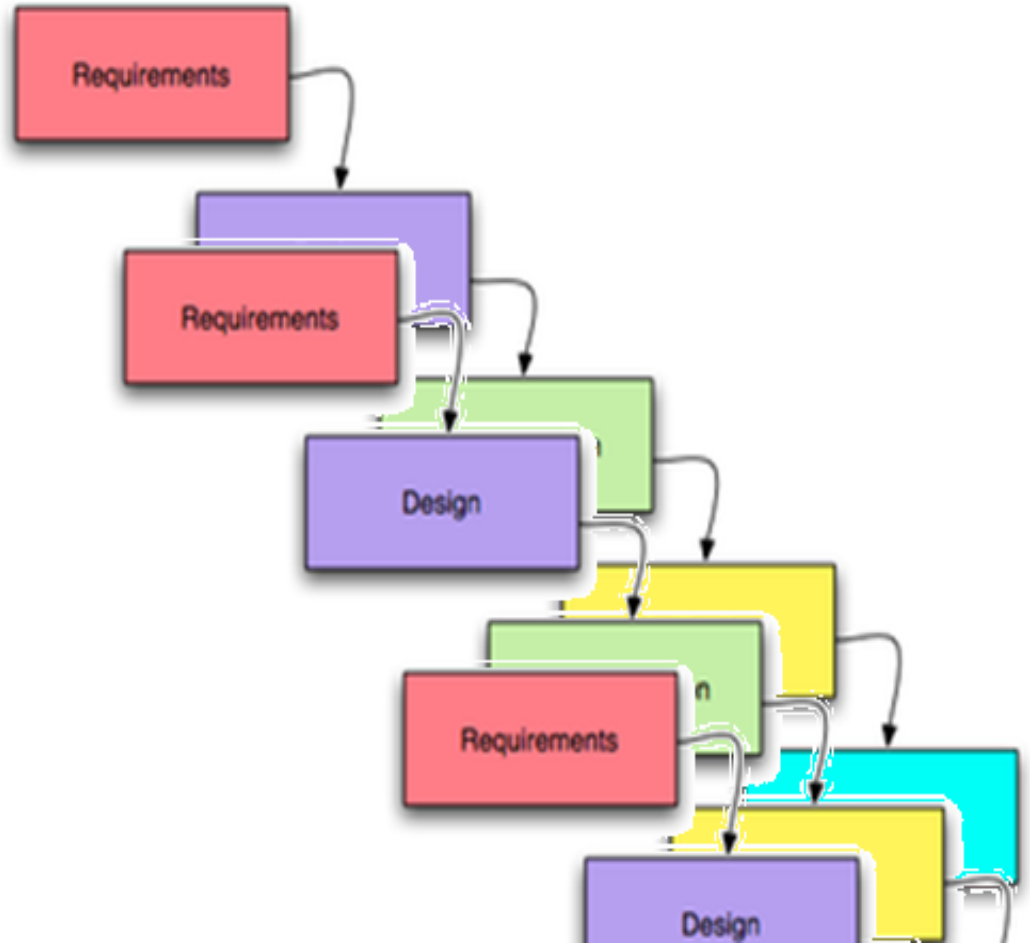
The stages of developing a software application

- Requirements Analysis
- High-level Design
- Low-level Design
- ***Implementation***
- Unit Test
- ***Integration***
- ***System Test***
- Deploy
- Maintain



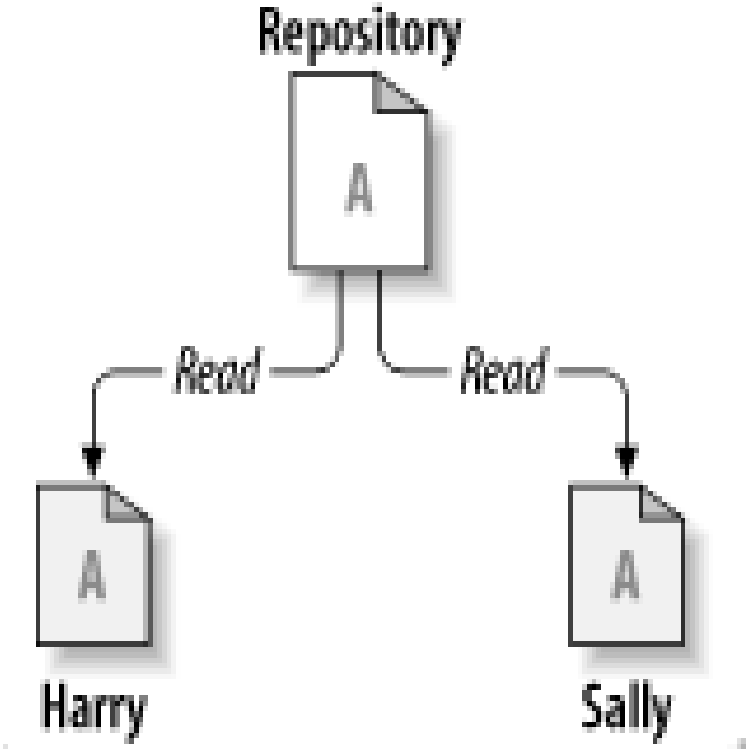
In many cases, multiple projects run concurrently

- New features are typically being added to the next version
- While at the same time, defects need to be corrected in existing releases
- All team members work on the same code and develop their respective parts
- An engineer may be responsible for an entire class
- Or just a few methods within a particular class



Sharing Files Among Developers On A Team Project

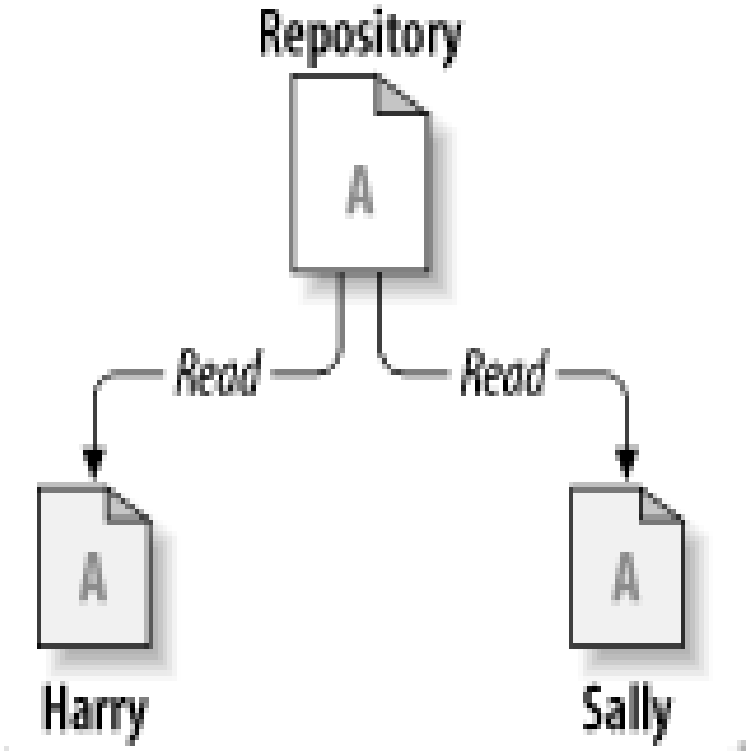
- Shared files can be kept in some type of **Repository** where they can be accessed and worked on by multiple individuals



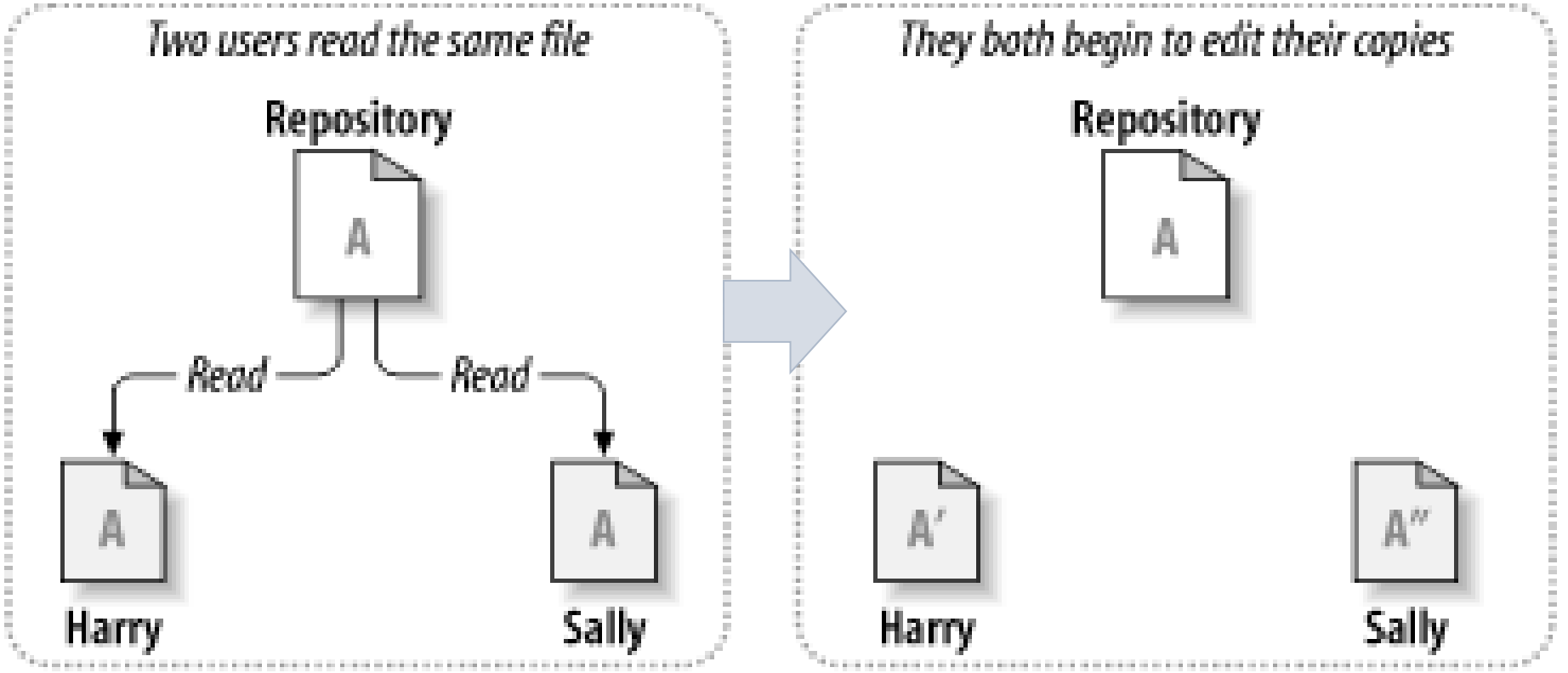
Harry and Sally get their own respective local **Working Copies** of the Master File in the Repository

What might you use for a Repository?

- Would any of these work?
- USB/SD drive
- Private network drive
  - E.g., shared “X:” drive
- Cloud drive
  - Google drive
  - Dropbox
  - iCloud

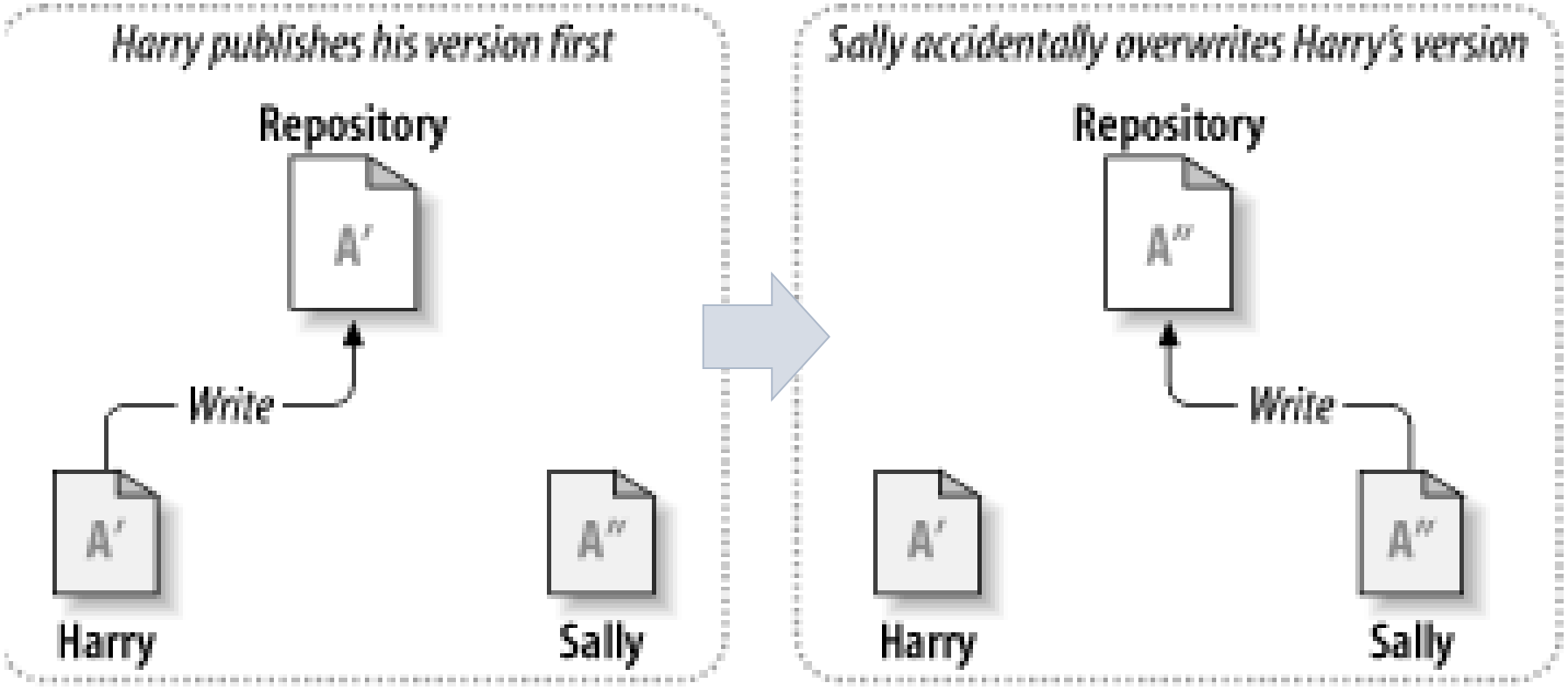


Sharing files can lead to big problem



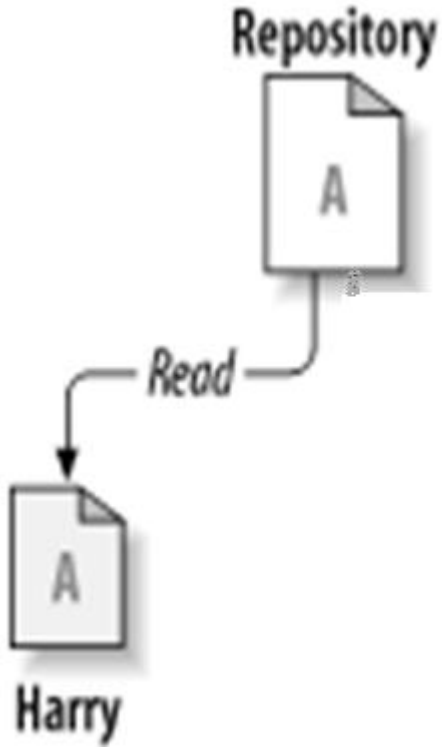
Adds some method **Modifies some other method**

The problem to avoid



Harry's changes are still held locally…

Harry’s changes are lost



Harry’s local file gets overwritten with Sally’s latest changes.

The Issue: Effective management of software artifacts, especially code

Some acronyms:

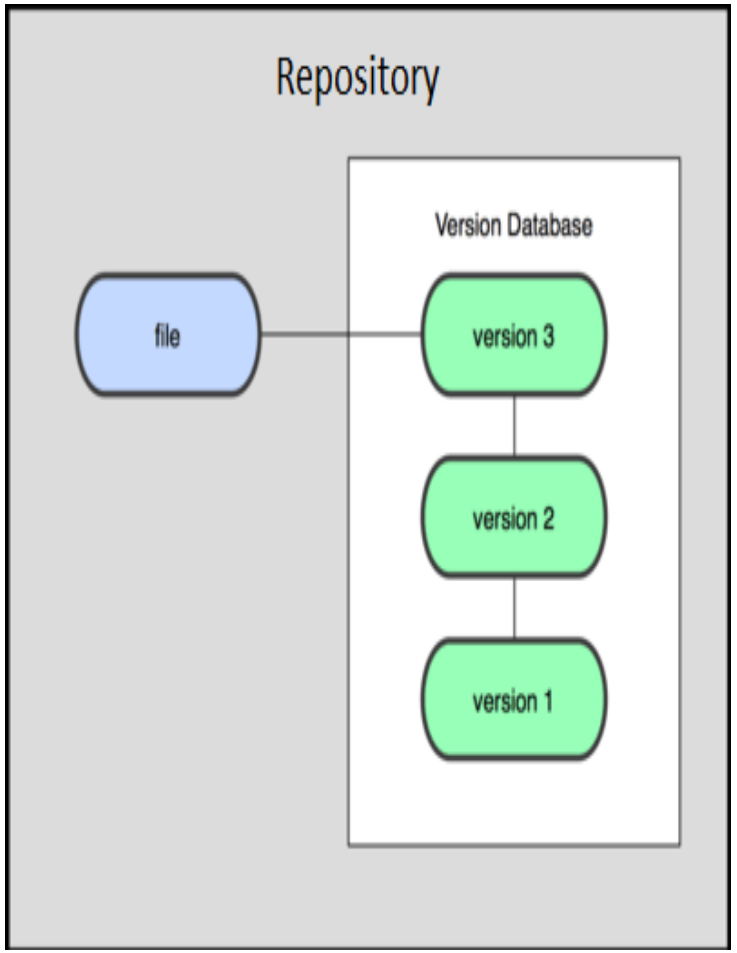
- SCM – Source Code Management
- SCCM – Source Code Configuration Management
- RCS – Revision Control System
- VCS – Version Control System
- DVCS - Distributed Version Control Systems

Revision/Version History might help

- …but USB/SD, private network drives do not maintain revision history

- Cloud drives (e.g., **Google Drive**, **Dropbox**, **iCloud/Time Machine**) maintain a (limited) history of the various revisions of a file
  - Google Drive: keeps last 100 revisions or last 30 days
    - Supported in “Classic” mode only – going away?

- If you detect a collision, you can manually merge the differences and resynchronize your work
  - Might be workable for two people – what about 5 or 10?



Any other disadvantages to Google Drive/Dropbox/iCloud

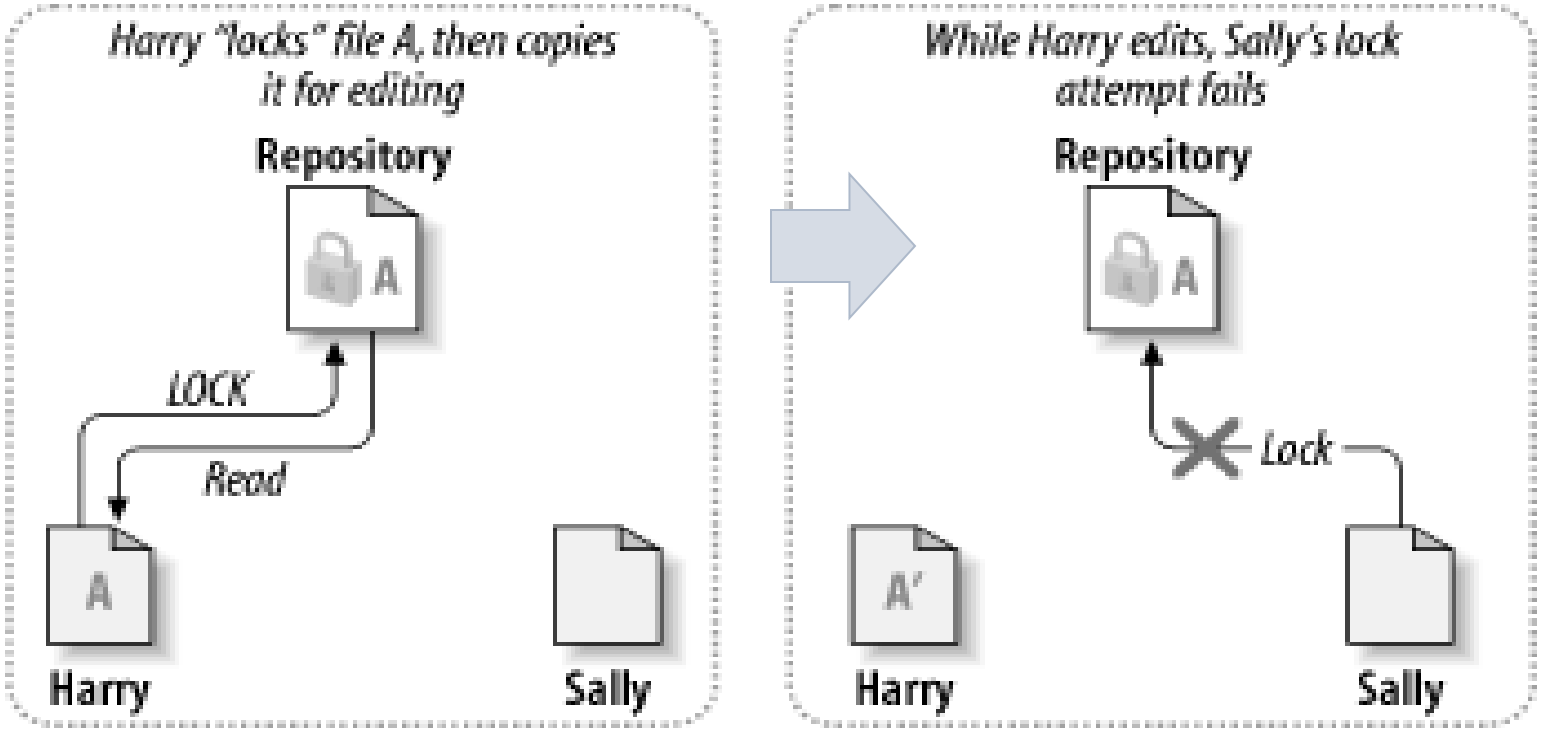
- Security?
- Support (bug fixes and new features)?
- Ownership of stored information?
- Cost?
- Ease of Use?
- Acceptance?

Features of version control systems

- All changes to a file are tracked
  - Version histories show **who**, **what**, **when**
- Changes can be reverted (rewound to any earlier version)
- Changes between revisions are easy to identify

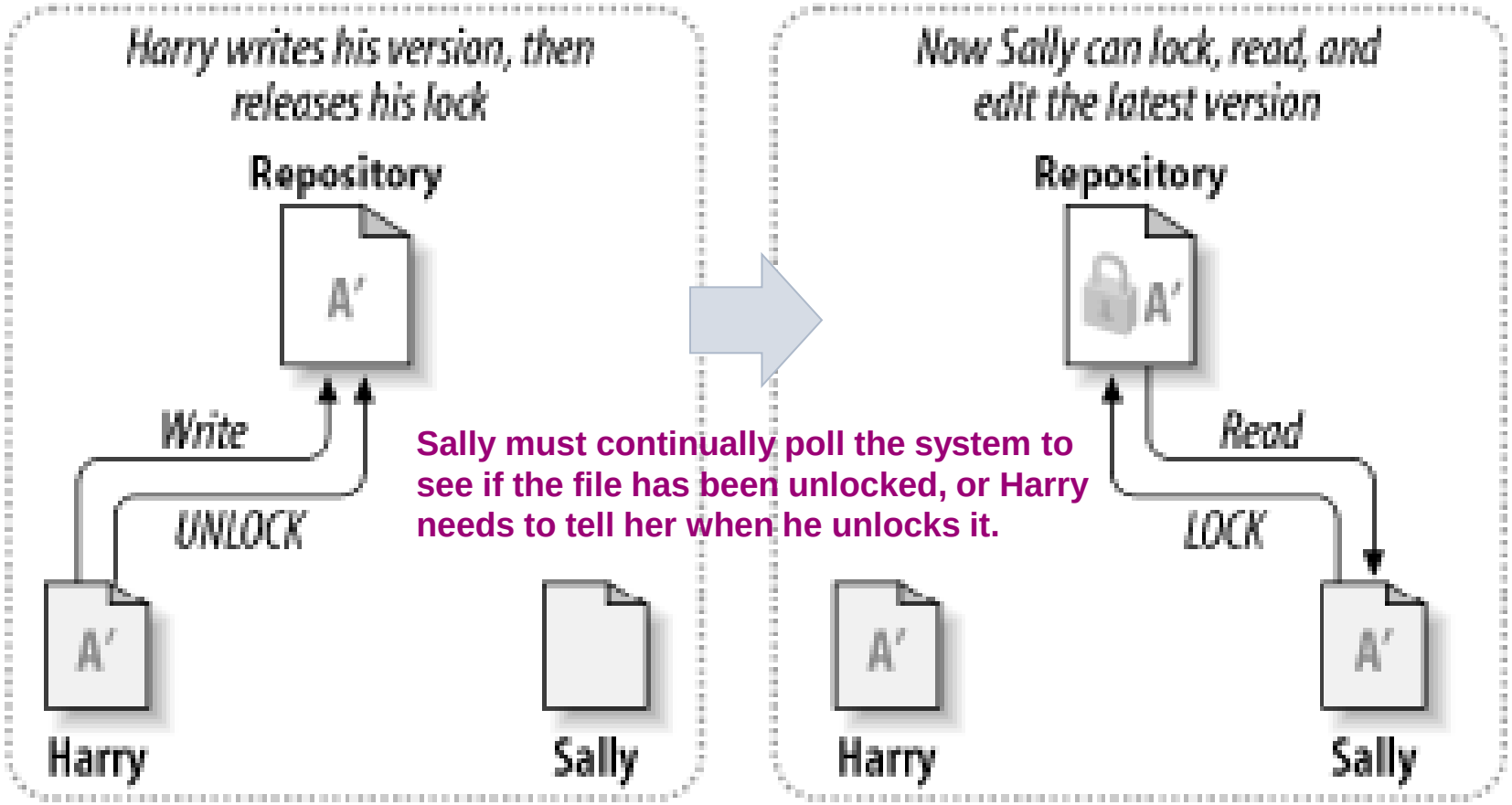
The Lock-Modify-Unlock solution

Early systems followed this model (RCS, SourceSafe)



Sally must wait until Harry releases the lock, although she can retrieve a "readonly" version of the file in the meantime.

Lock-Modify-Unlock only allows a single user to edit the file at a time



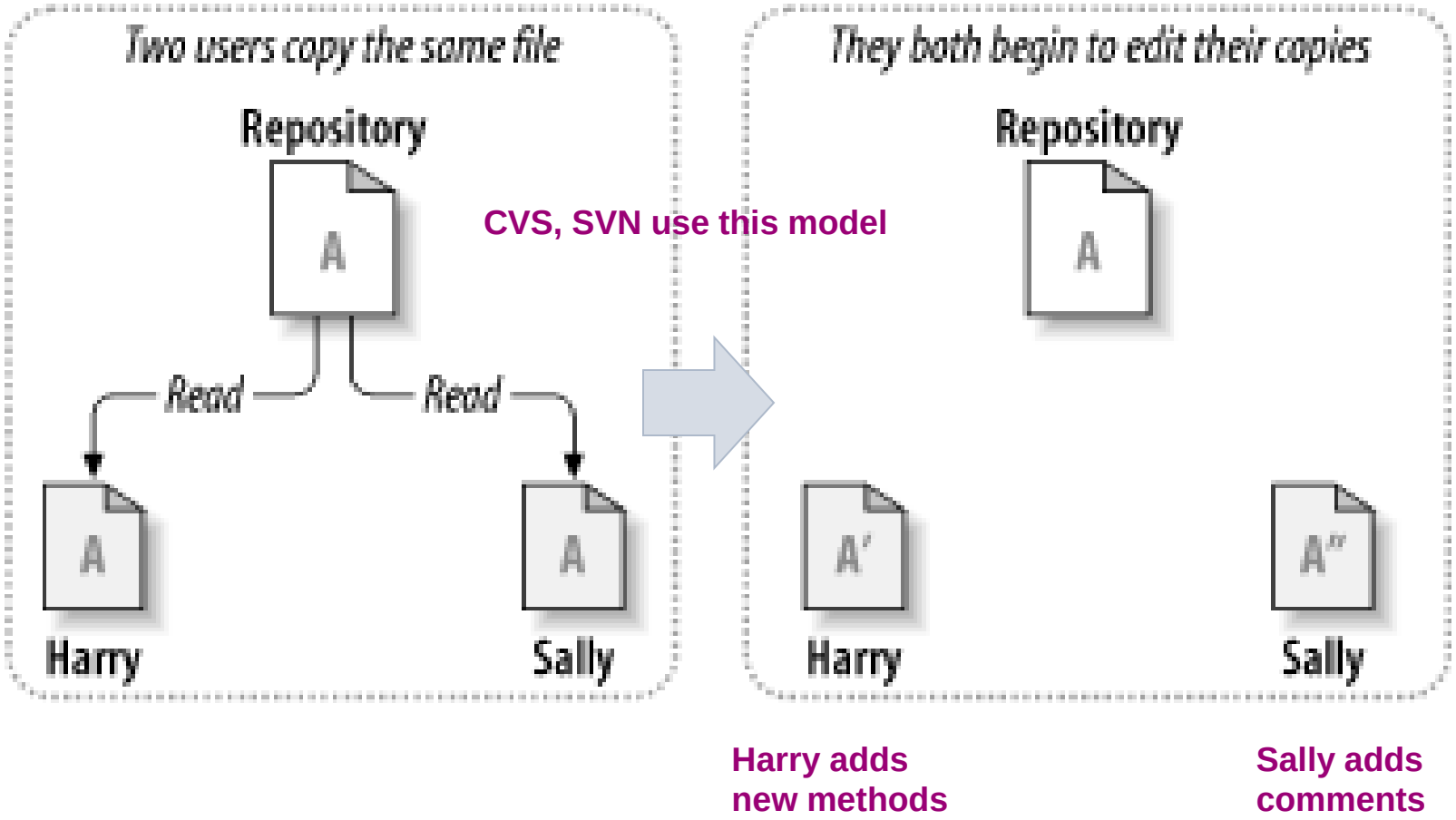
Sally must continually poll the system to see if the file has been unlocked, or Harry needs to tell her when he unlocks it.

Lock-Modify-Unlock has certain drawbacks

- Harry has to remember to release his lock before Sally (or anyone else) can acquire the lock in order to edit

- Sally has to wait for Harry to finish before editing (editing must be done sequentially)
  - Harry might be adding some new methods (takes a long time)
  - Harry might forget to release the lock before going home (or on vacation!)

The Copy-Modify-Merge solution allows concurrent editing

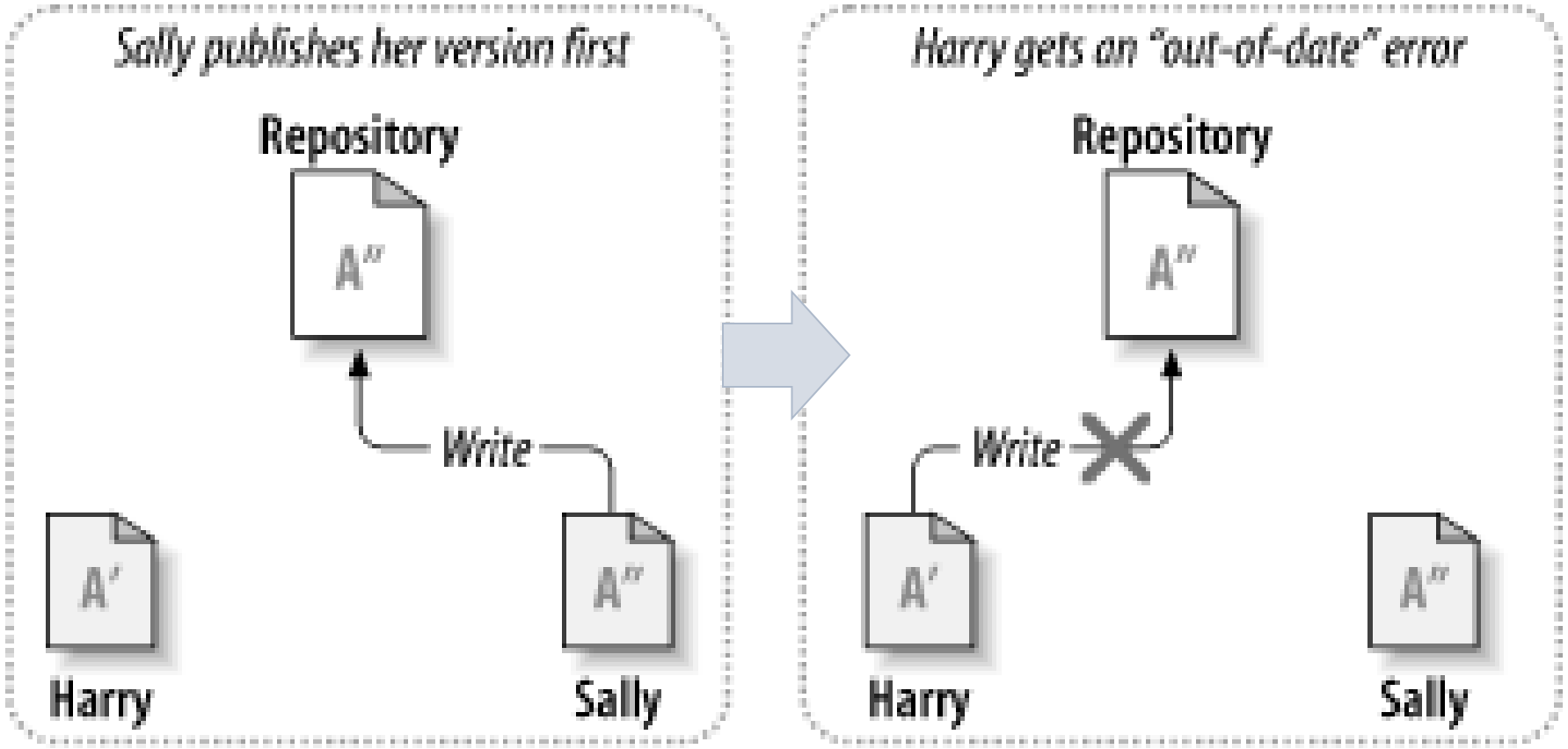


CVS, SVN use this model

Harry adds new methods

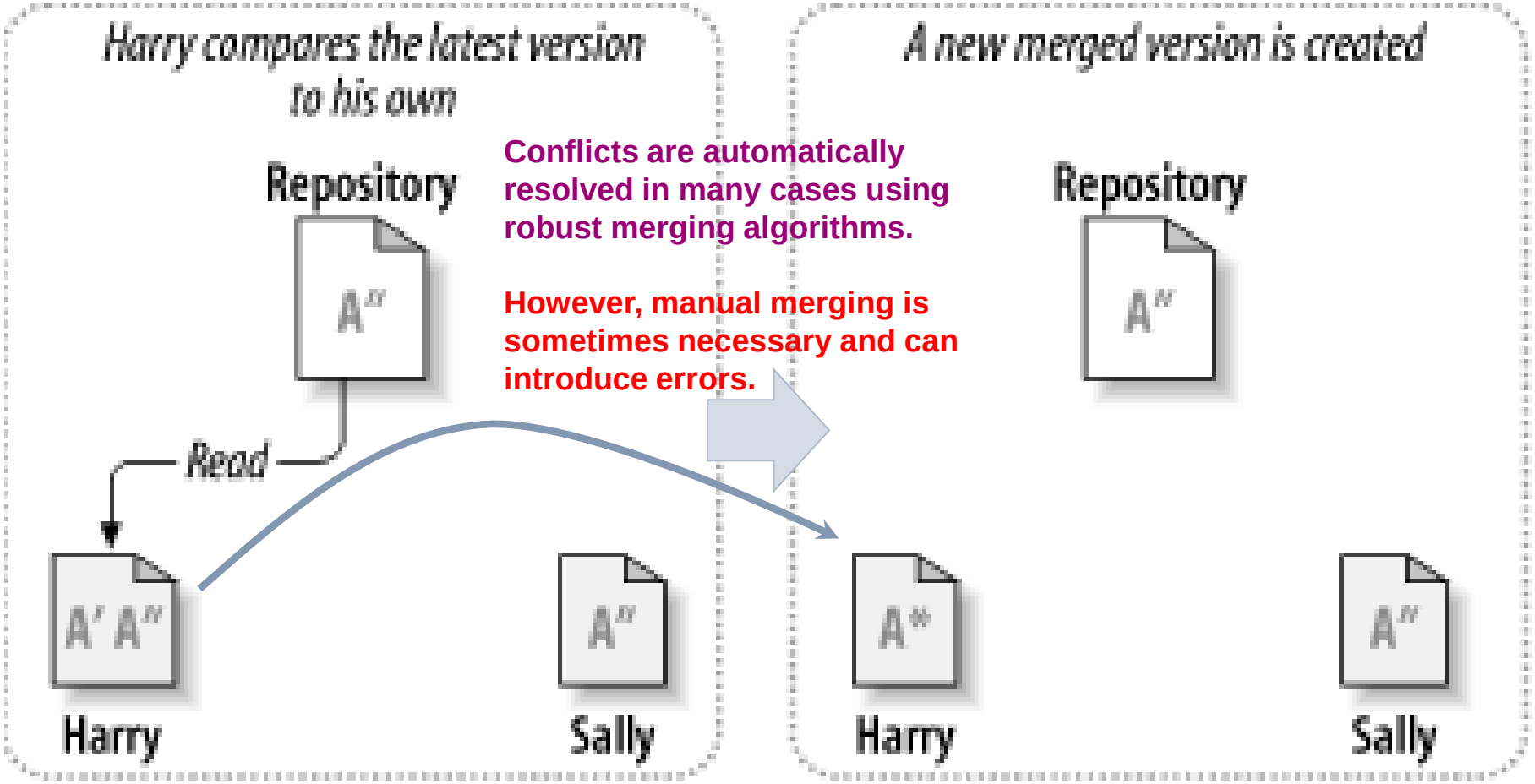
Sally adds comments

The Repository recognizes conflicts



Harry is prevented from writing

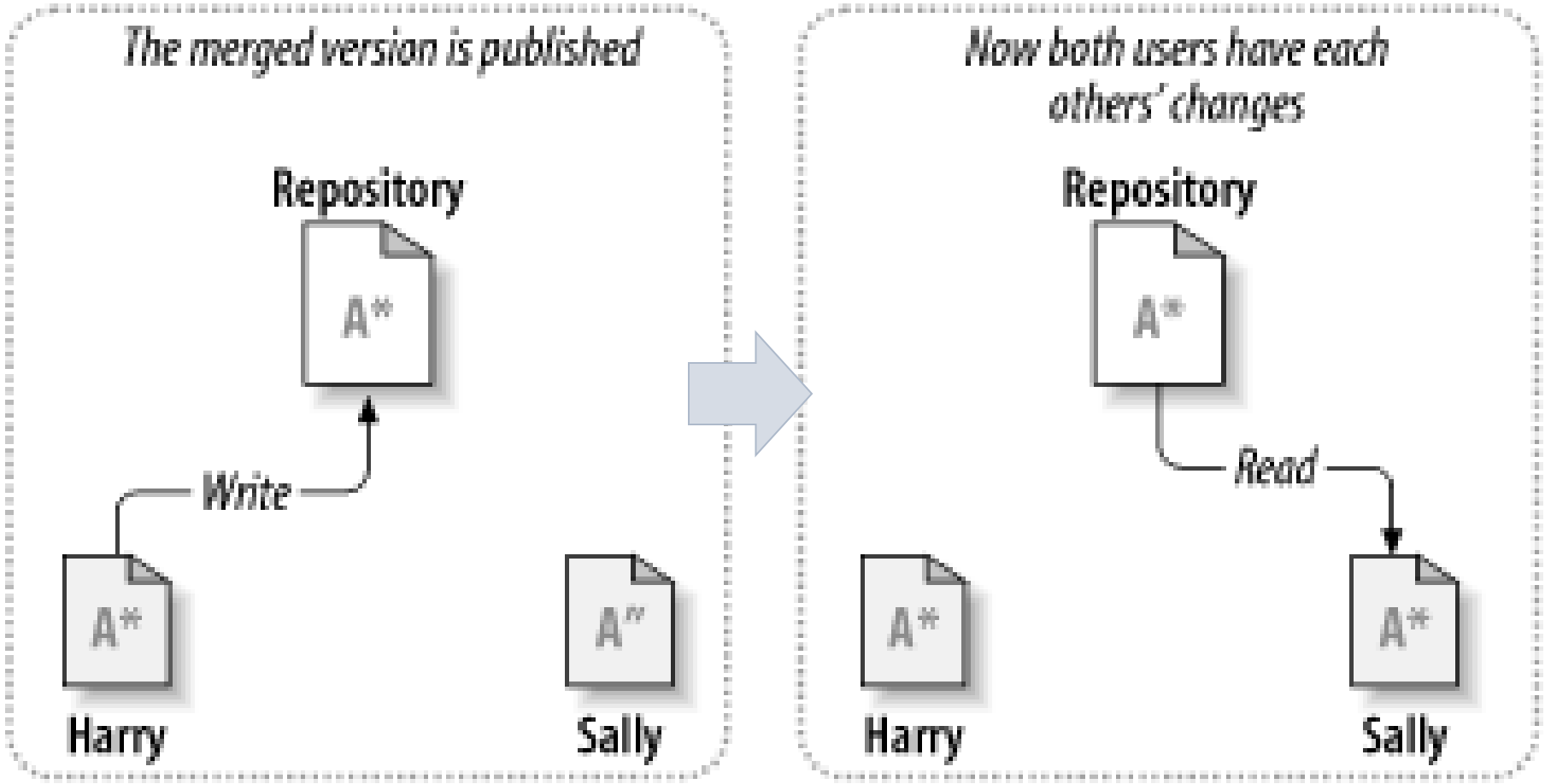
Conflicting versions of the files are merged



Conflicts are automatically resolved in many cases using robust merging algorithms.

However, manual merging is sometimes necessary and can introduce errors.

The Repository keeps both users synchronized



Copy-Modify-Merge is generally easy to use and manage

- Users can work in parallel
- Most of the time, concurrent changes don't overlap
  - People generally don't edit exactly the same code simultaneously, so merging is done automatically in many cases
  - Amount of time spent resolving conflicts is nearly always less than the time that would be spent waiting for a lock to be released

Is Lock-Modify-Unlock ever needed anymore?

When two or more people need to work on the same file, the simultaneous changes may not be mergeable in all cases:

- MS Office documents
- Image documents
- Other binary files

Source Code Management (SCM)

- The Problem: you and a couple partners are working on a project. How do you handle the source code?
- The hard way: you designate a guy to maintain the golden copy. As you finish code you give it to him, and he places it in a directory on his laptop
  - Forces a guy to do this manually
  - What if two people are working on the same file?
  - What if you want to go back to an older version of a file?
  - What if you want to maintain multiple “forks” or versions of the program?

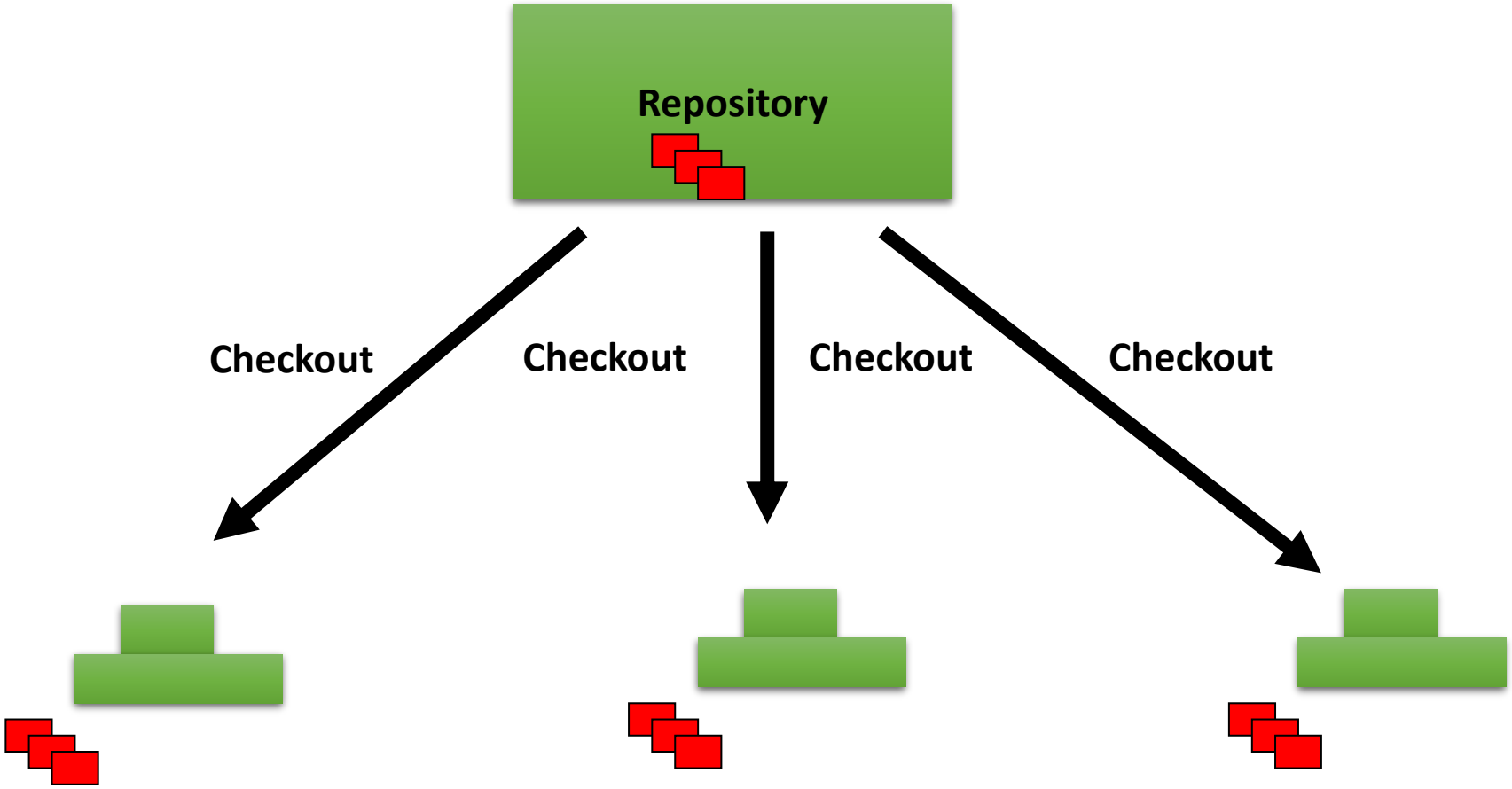
Source Code Management

- These are the problems source code management is intended to solve. Effectively it is a database for source code that eases the problems of
  - Multiple people in many places working on the same code
  - Retrieving old versions of files
  - Keeping logs about what changed in a file
  - Integrating changed code
  - Generating release builds

Source Code Management

- What it is NOT:
  - A replacement for communication
  - A replacement for management
  - A substitute for other tools, such as bug tracking, mailing lists, and distribution
  - It has no understanding of the semantics of your program; as far as it's concerned, it's all just bytes

Concept of Version Controlling



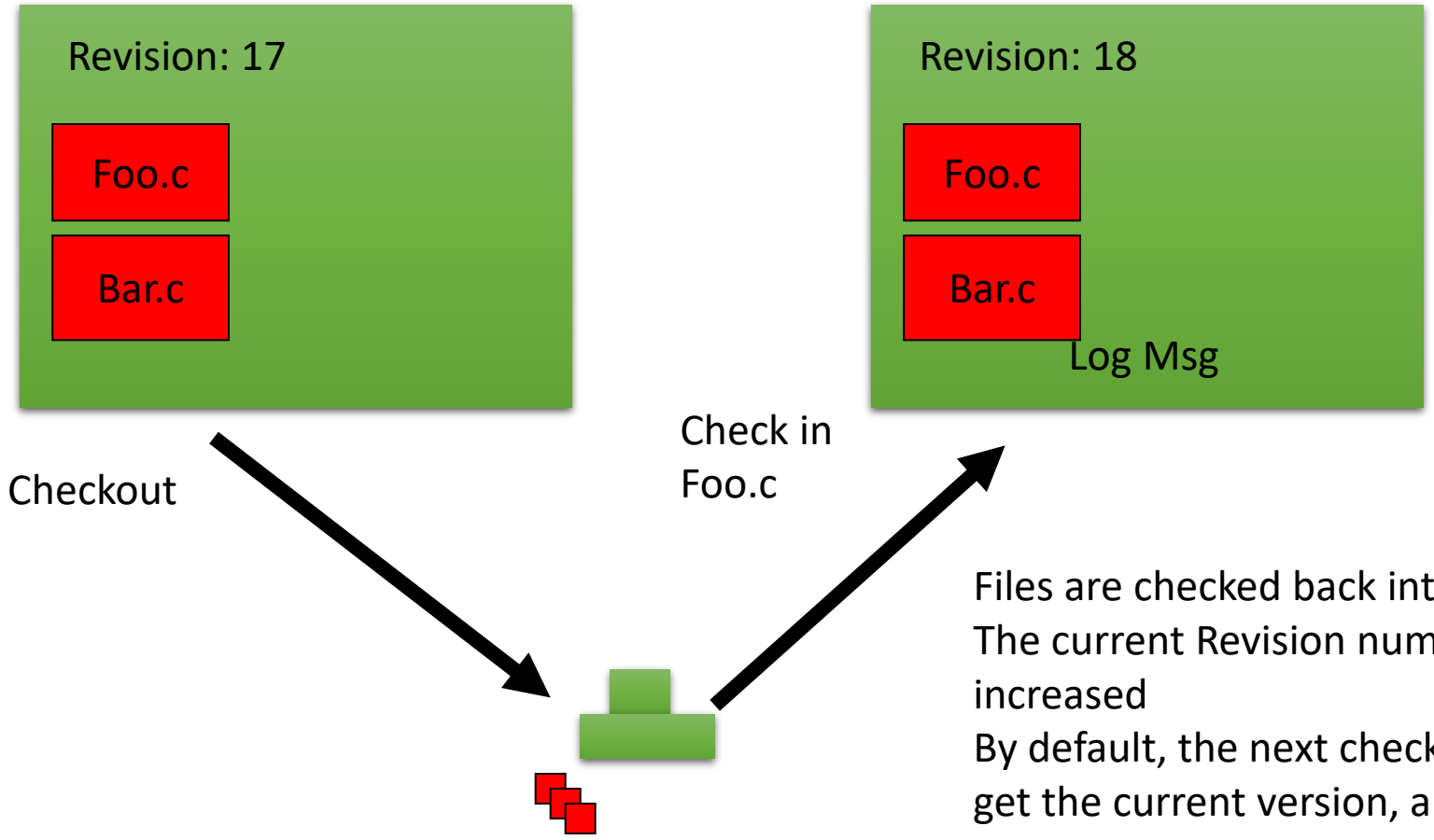
Checkout

- Programmers contact the central database and retrieve a copy of the project files
- They don’t have to do it all at once; they can contact the database individually and without coordination
- The database gives them a local copy of the files that they can modify in any way they want.
- What if the programmers messes up? He can simply delete all the files on his computer and check them out again.

Checking Files In

- Eventually the programmer does something useful and wants to put the file back in the database , so he checks it in.
- This causes the version number of the repository to be increased by one, e.g., from 17 to 18
- The “repository version” is a number assigned to all the files collectively in a project whenever anything is changed. If you have five source code files in a project, and check in one modified file, the repository revision will be increased by one, and the new repository revision will refer to all of the files collectively. Checking in five files at once will also increase the repository version by one
- Earlier versions of the files can still be retrieved from the database if needed This allows you to back up if you made a mistake or need to see an old version of the file.
- The check in process also allows you to make a log entry for that file, like “Fixed bug that caused intermittent spontaneous human combustion”

Check In



Files are checked back into the database .
The current Revision number of the repository is increased
By default, the next checkout by a client will get the current version, and you can retrieve any Repository version

Updates

- Conceptually that's what happens, but the reality is more complex. What happens if other people are working with the database too?

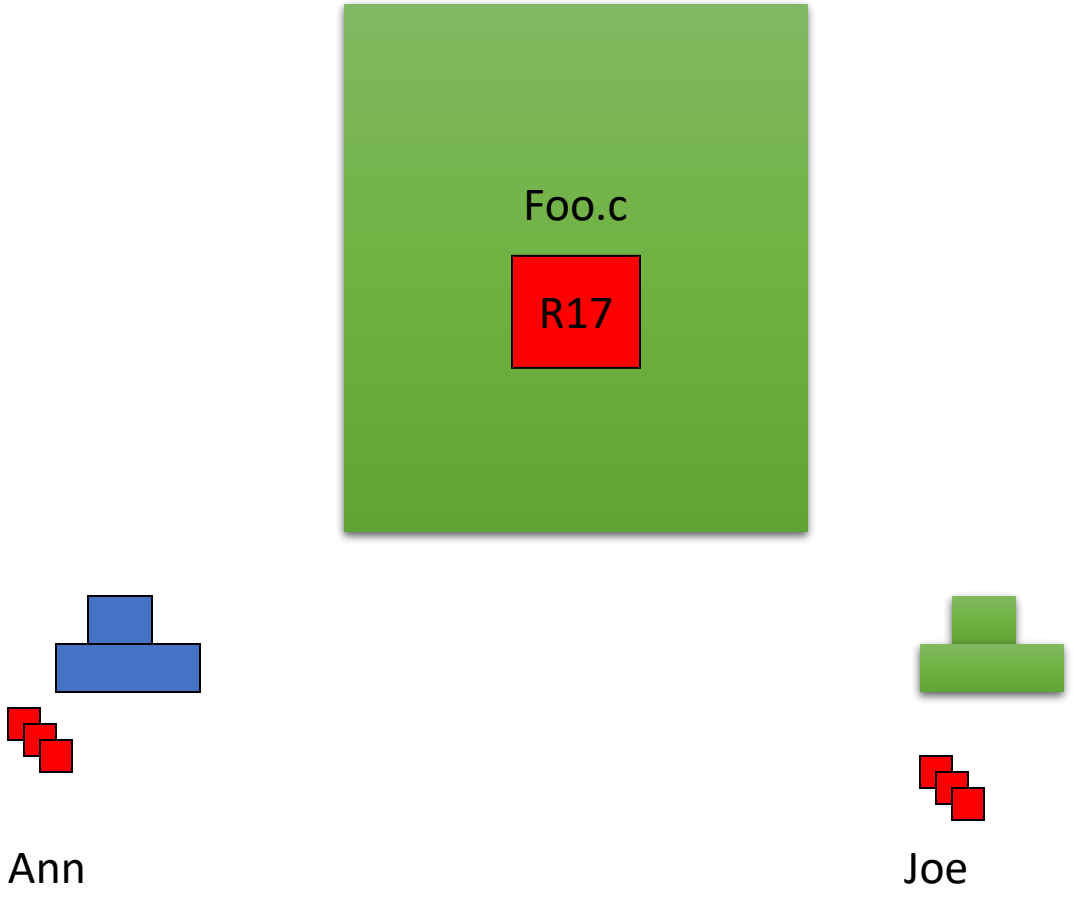
Updates

- The process of updating a file to the database consists of three steps:
  - Get current version of file from database
  - Merge any changes between the database version and the local version
  - Commit the file back to the database

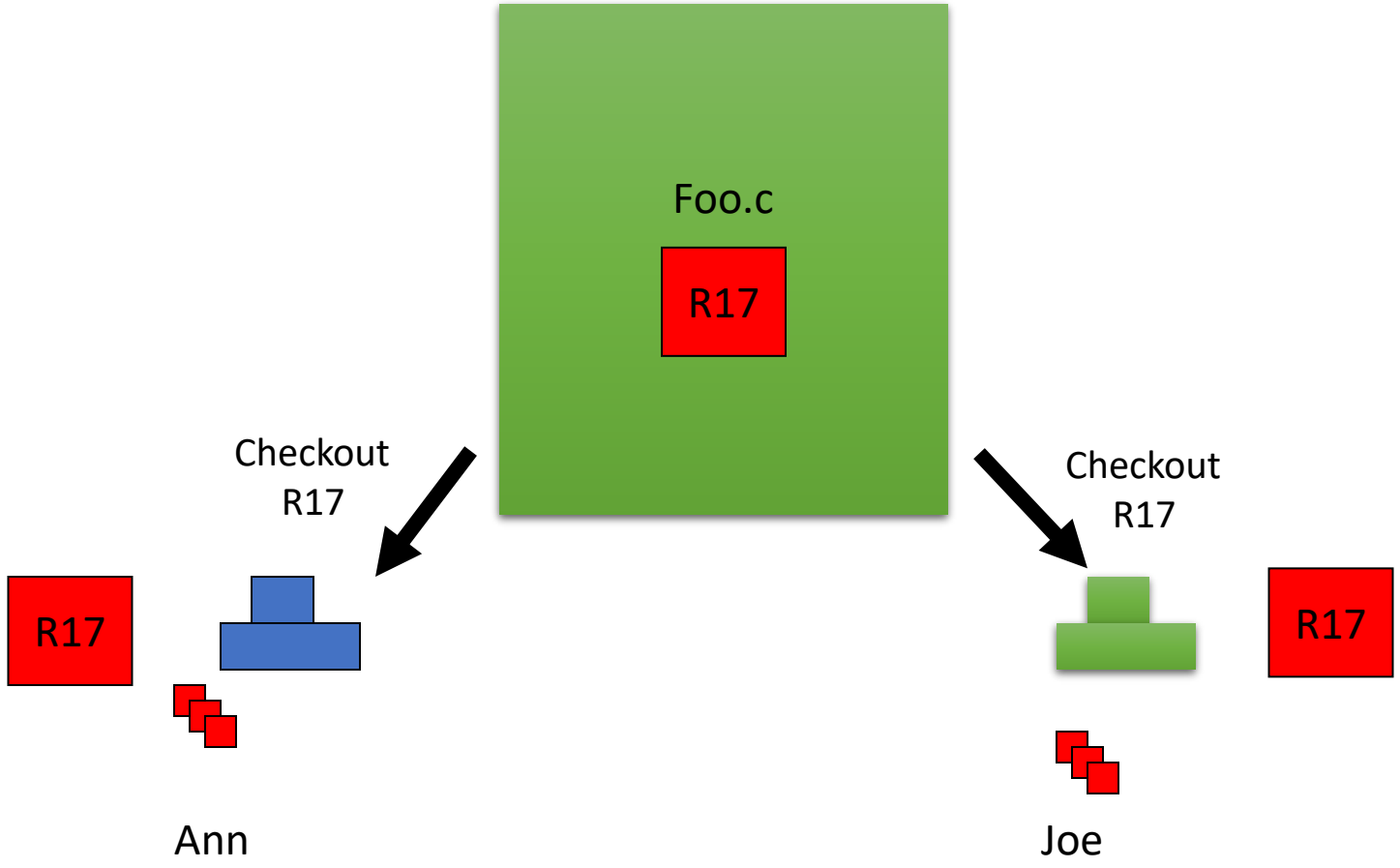
Example

- Ann and Joe check out files from database
- Ann modifies a file and checks it back in
- Joe modifies his file, but the file now has two changes: Ann’s and his.
- He retrieves the current file from the database, causes his changes to be merged with Ann’s, and checks the file back in.

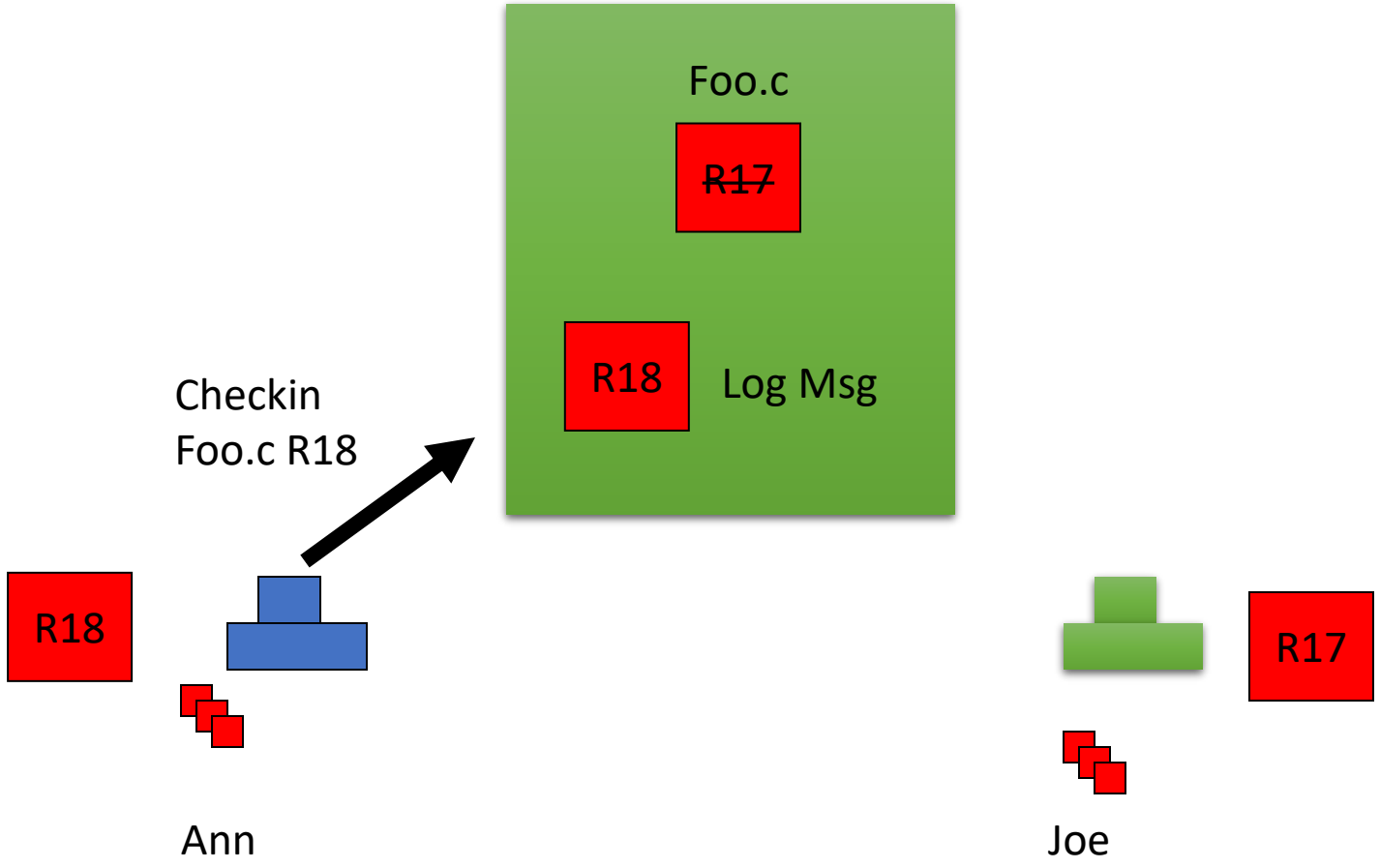
Example:



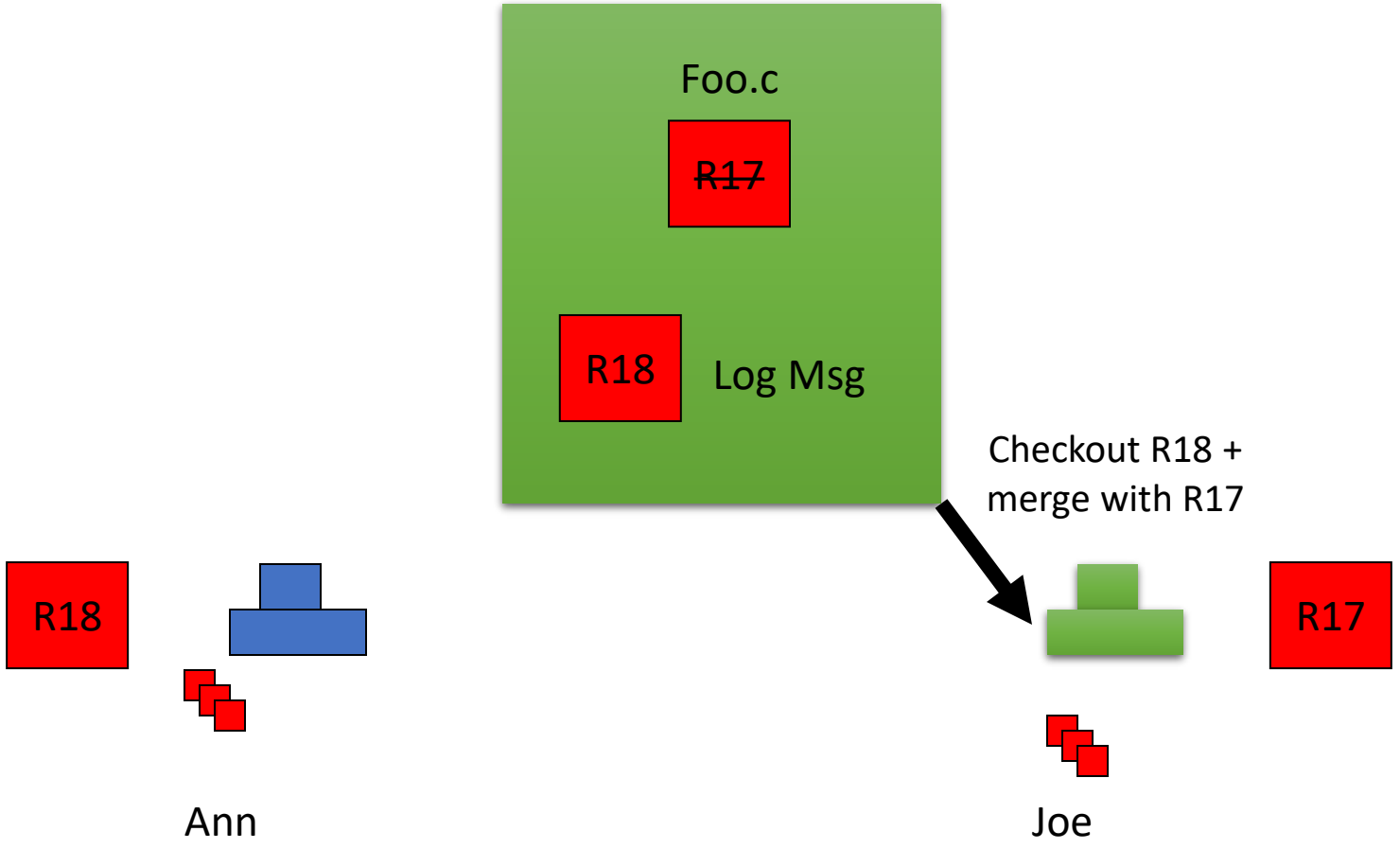
Example: Step 1



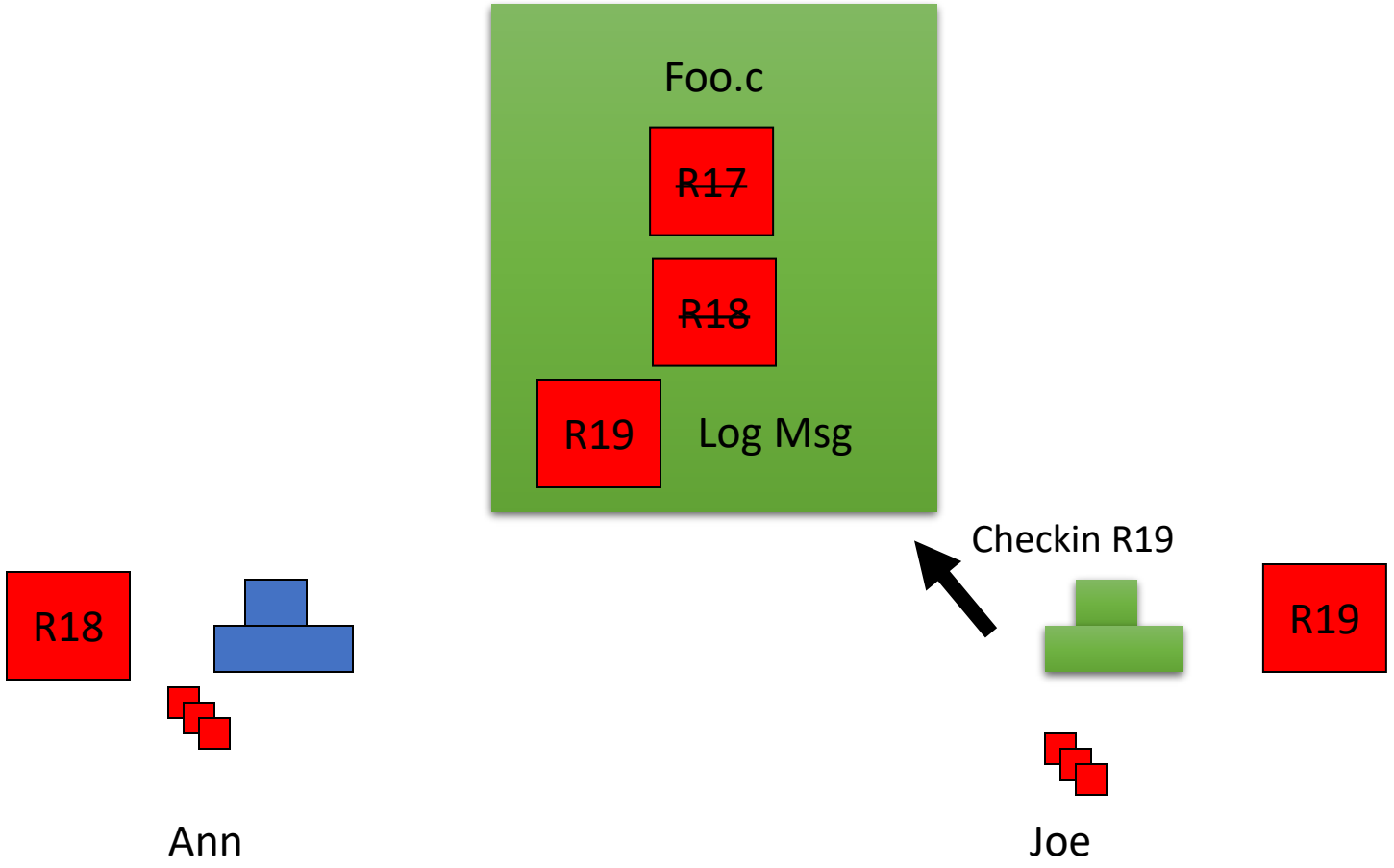
Example: Step 2



Example: Step 3



Example: Step 4



Concurrent Changes

- What if Ann modified a file and checked it in, and Joe wants to check in his changes?
  - Best case: they modified different areas of the file. In this case the Update operation will automatically merge in the changes
  - Worse case: the two programmers modified the same lines of code. In this case you have to manually resolve the changes

What Goes In?

- When you create your repo, what do you check in?
- Do NOT check in files that are created from other files in the build process--.class, .o, editor backup files, etc.
- Do check in source files (.java, .c, .cc), text files, xml files, image files
- Binary files are OK so long as they are specified as binary during checkin
- You should be able to check out, type “make”, and have a functioning program.

SCM Programs

- There are several programs that do version control, both free and commercial. The concepts are generally the same—you check out code from a central database to work on—but the details are different.
- **Free**: Concurrent Versions System (CVS), Subversion, git
- **Commercial**: BitKeeper, Microsoft SourceSafe, Metroworks, Perforce, ClearCase

Summary

- Source code management is good. If your program isn’t under SCM, it probably isn’t “in control”.
- It’s useful even for small projects with single developers