Software Version Control

CSCI 4140: Open-Source Software Project Development

Prof. Hong Xu

http://course.cse.cuhk.edu.hk/~csci4140/

Based on slides by Jonathan Aldrich, Charlie Garrod and Ruth Anderson

Why Version Control?

- Scenario 1 (individual):
  - You had a working implementation for Homework #1 yesterday
  - You made a lot of improvements to optimize its performance, UI, etc.
  - You haven’t gotten them to work yet
- You need to submit your homework in 5 minutes

Why Version Control? (Cont.)

- Scenario 2 (individual):
  - You had a working implementation
  - You changed one little thing and your program broke
    - You change your program back
      - It still breaks
- What’s wrong?

Why Version Control? (Cont.)

- Scenario 3 (individual):
  - You accidentally deleted one file with 1K LoC
  - Your hard drive failed one day before the deadline

Why Version Control? (Cont.)

- Scenario 4 (team):
  - How do you share code with your teammates?
    - Email
    - Shared network directory
    - rsync
    - FTP Server / Dropbox / Google Drive
    - Google Docs / Microsoft Office 365

Why Version Control? (Cont.)

- Scenario 5 (team):
  - You changed one file of the program
  - Your teammate makes different changes to the same file
  - How do you merge the changes?
- Scenario 6 (team):
  - You changed one file of the program and uploaded it
  - Your teammate makes different changes to the same file and overwrite the one you wrote

The *diff* utility

- A tool that
  - Helps you spot the differences (changes) between two files
  - Generates a “patch” to apply the differences to one file
- Requires the older version of the file

Solution: Version Control

- You use version control in lots of software
  - The undo button in text/photo/spreadsheet editors, etc.
- A Version Control System (VCS) is a software that
  - Tracks multiple versions (history) of files (code, resource, etc.) in a project
  - Manages and displays the differences between versions
  - Attaches comments with each change
  - Allows undo of a change / rollback to an earlier version
  - Supports parallel development

Repository

- A central location that storing and managing the history of all files
- Common operations include
  - *Check in*: adding a new file to the repository
  - *Check out*: fetching a copy of a file from the repository to your local workspace
    - You don’t edit files directly in the repository
    - All modifications are local until they are committed
  - *Commit*: checking in a modified version of files that were checked out
  - *Revert*: undoing changes to files that were checked out
  - *Update*: fetching copies of the latest version of all files that have been committed by other users

Repository Location

- A repository can be created on your local machine or on a remote server
  - A local repository is suitable for temporarily managing a personal and small-scale project
- A robust repository is preferred in most situations
  - On a machine that allows 24/7 access
    - Authorized user can access it anytime from anywhere
  - On a machine that provides storage redundancy
    - Accidental hard drive failure is no more a worry

History of Files

- Record changes or snapshots of the project
  - Supports undo
  - Implies backups
- Record who performed what changes
- Label cohesive development steps
  - Explains rationale
  - Helps understand the changes
  - Allows tracing in history and to other development artifacts (bug trackers, CVEs)

Rationale and Traceability

index : kernel/git/torvalds/linux.git

master switch

Linux kernel source tree — Linus Torvalds

about summary refs log tree commit diff stats — log msg search

author	Linus Torvalds <torvalds@linux-foundation.org>	2018-01-07 11:42:57 -0800
committer	Linus Torvalds <torvalds@linux-foundation.org>	2018-01-07 11:42:57 -0800
commit	b84449dc14d274a3f3c78cd734b702ca31aa4dd1 (patch)	
tree	ab61e05babc155ed3b17c9d5d7898e2b860f84bd	
parent	9cfd403a7cee59aeb197ddb99eeca1bb590fee74 (diff)	
parent	310d82784fb4d60c80569f5ca9f53a7f3bf1d477 (diff)	
download	linux-b84449dc14d274a3f3c78cd734b702ca31aa4dd1.tar.gz	

diff options

context: 3
space: include
mode: unified

Merge branch 'parisc-4.15-3' of git://git.kernel.org/pub/scm/linux/kernel/git/deller/parisc-linux

```
Pull parisc fixes from Helge Deller:

 - Many small fixes to show the real physical addresses of devices
   instead of hashed addresses.

 - One important fix to unbreak 32-bit SMP support: We forgot to 16-byte
   align the spinlocks in the assembler code.

 - Qemu support: The host will get a chance to sleep when the parisc
   guest is idle. We use the same mechanism as the power architecture by
   overlaying the "or %r10,%r10,%r10" instruction which is simply a nop
   on real hardware.

* 'parisc-4.15-3' of git://git.kernel.org/pub/scm/linux/kernel/git/deller/parisc-linux:
  parisc: qemu idle sleep support
  parisc: Fix alignment of pa_tlb_lock in assembly on 32-bit SMP kernel
  parisc: Show unhashed EISA EEPROM address
```

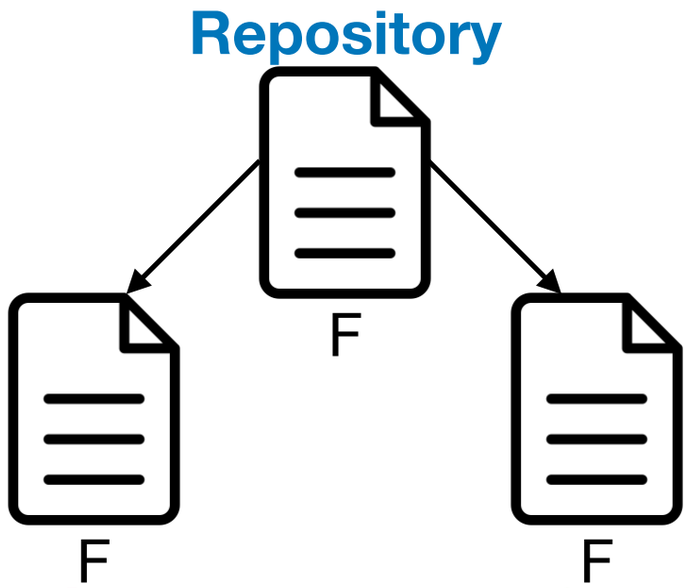
Which Files to Manage?

- All source code and resource files (plain-text files are preferred)
  - C/C++/Java/HTML code
  - Build/Configuration files
  - Images, styles, binaries
  - Documentation
- Excluding intermediary / generated files
  - Object files, class files, etc.
- Most VCS have a mechanism to exclude certain files

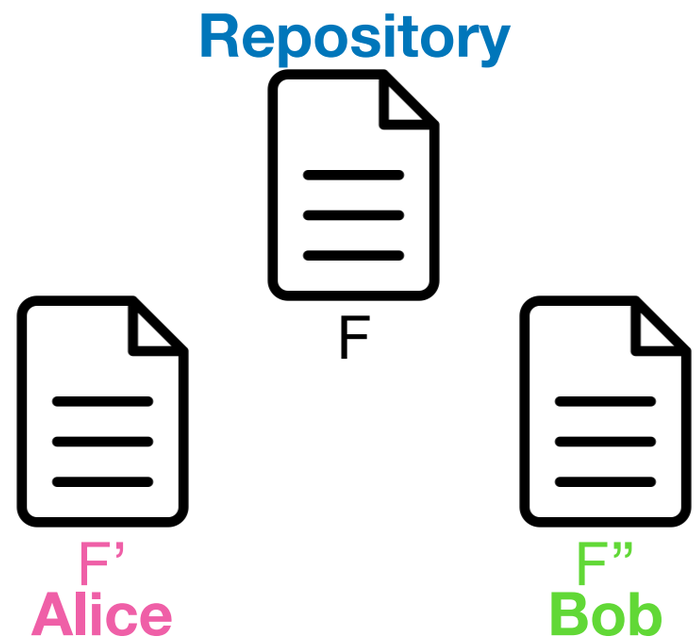
Concurrent Modifications

- Allowing concurrent modification is challenging
  - Conflicts may occur
  - Accidental overwriting
- Common strategies
  - Uses locks to prevent concurrent modification
  - Detects and resolves conflicts

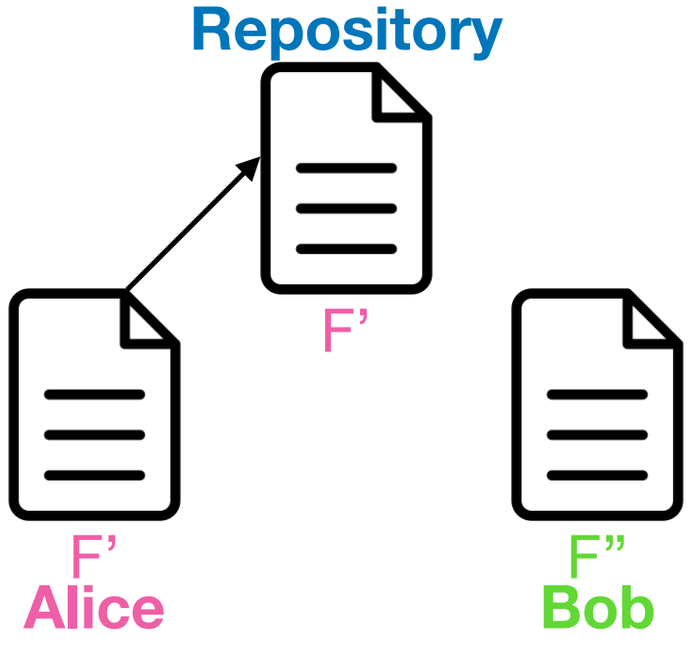
Change Conflicts



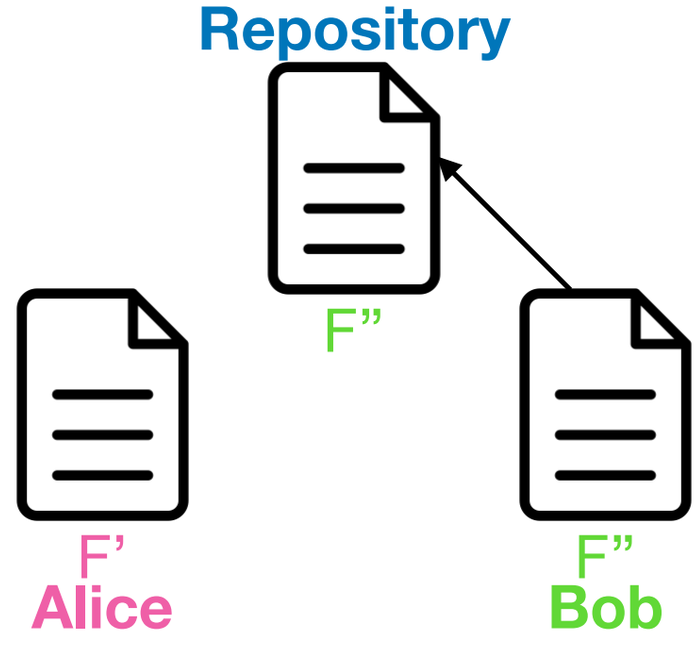
Two users read the same file



They edit their own copies

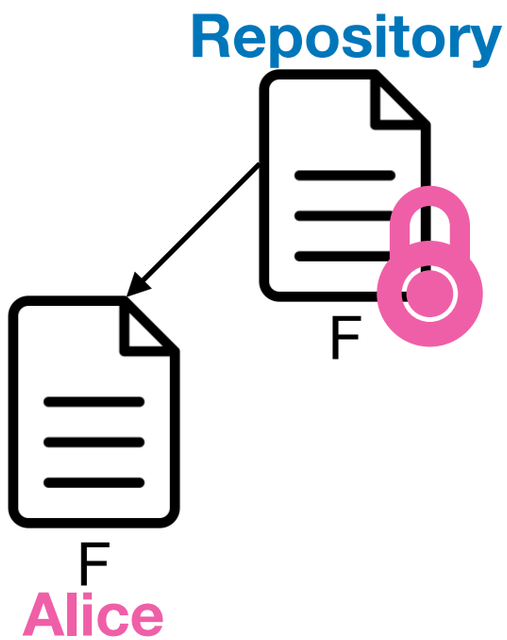


Alice publishes her copy first

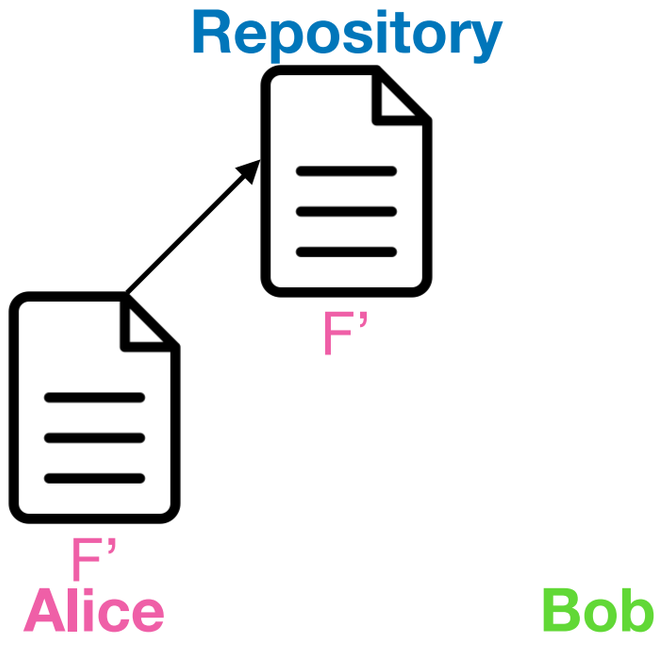


Bob publishes his copy later

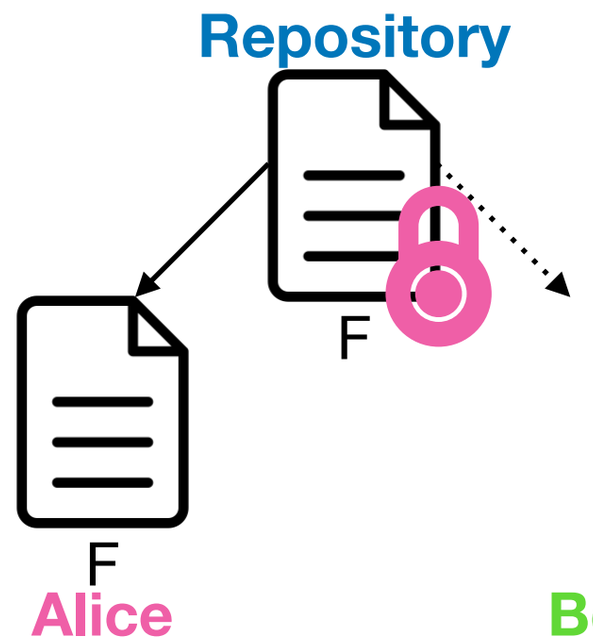
Locking Files



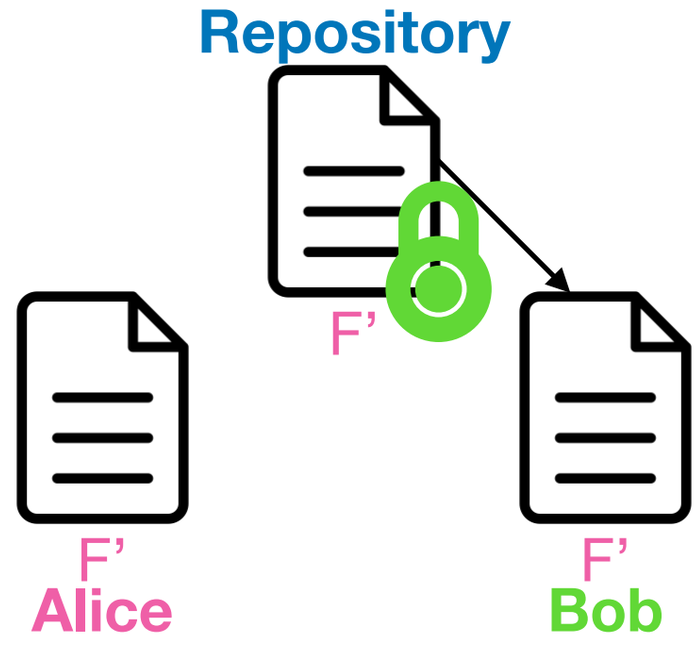
Alice locks and reads file F



Alice publishes her copy and releases the lock



Bob fails to acquire the lock, while Alice edits

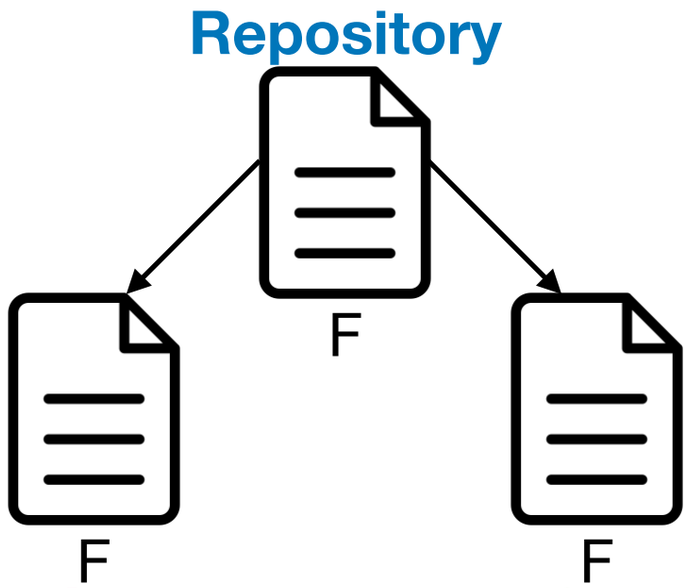


Bob can lock and edit the latest version

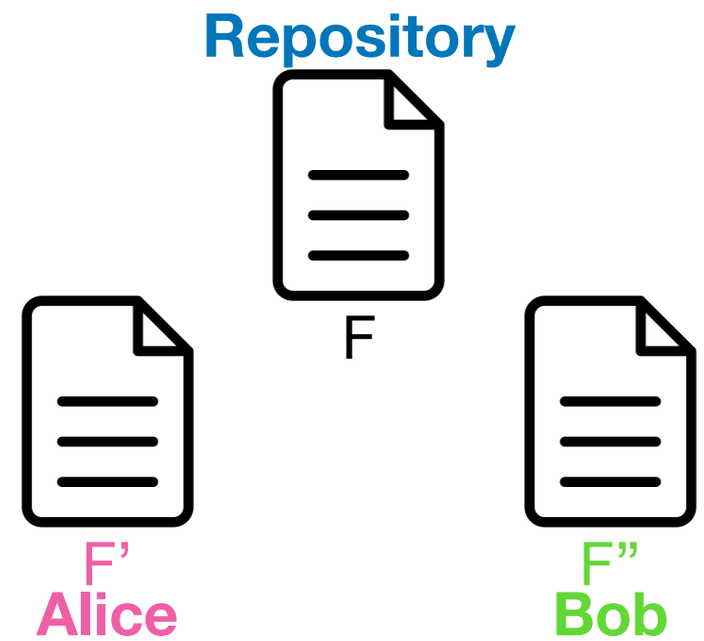
Problems Regarding Locking

- How to lock?
  - Announcement vs. central system
- When to unlock?
  - Automatically release after each commit
  - Manual release
- Blocks parallel development on the same file

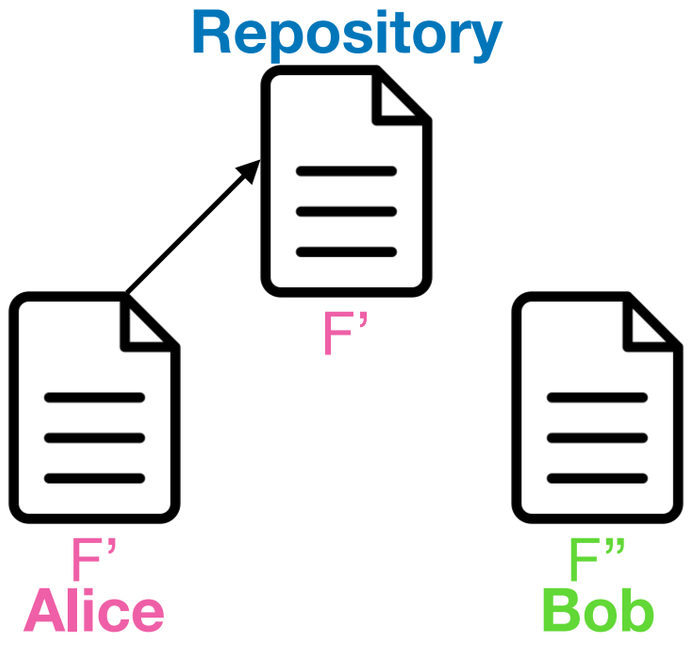
Detecting Conflicts



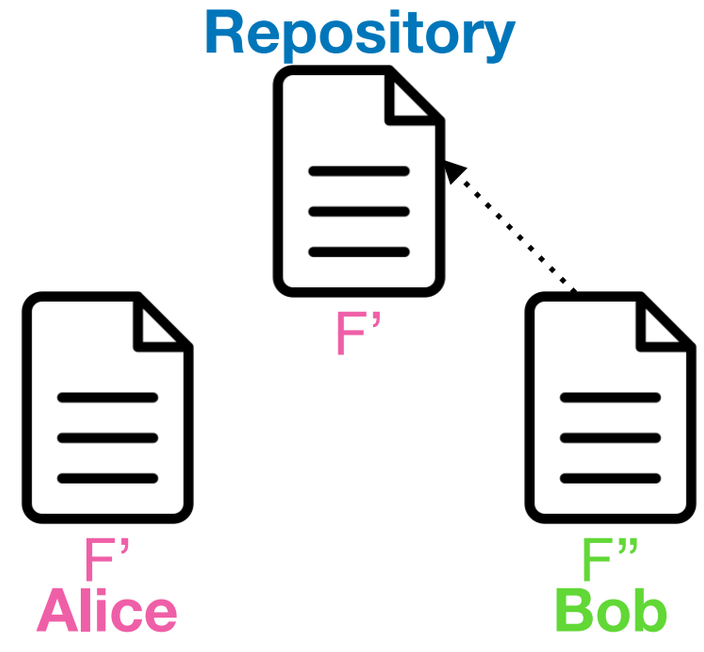
Two users read the same file



They edit their own copies

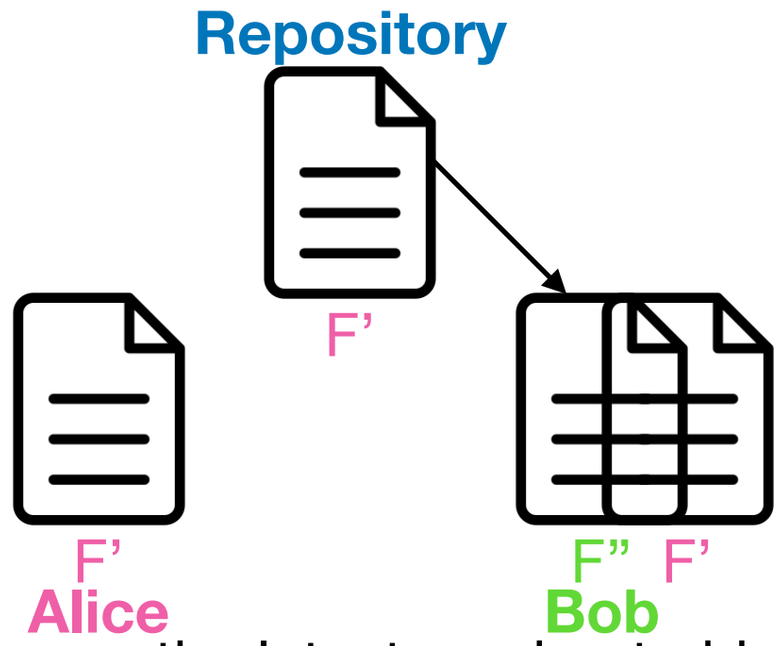


Alice publishes her copy first

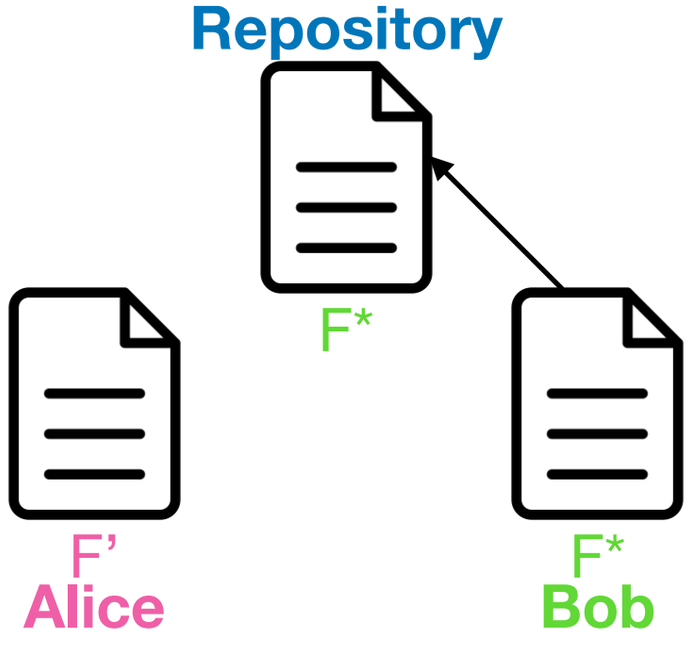


Bob gets a conflict error

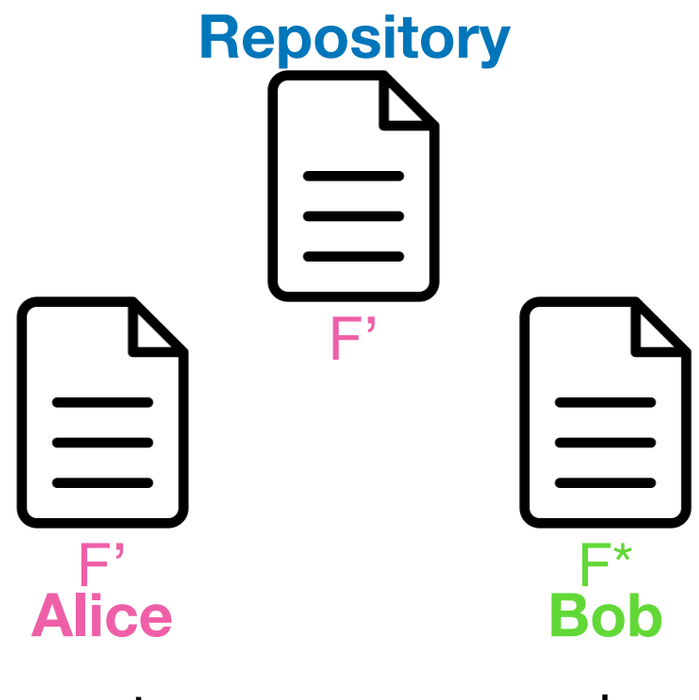
Resolving Conflicts



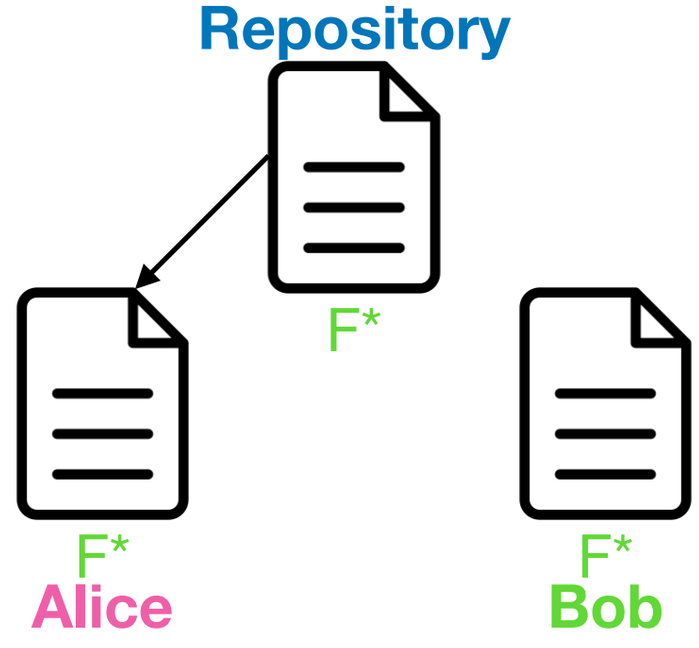
Bob compares the latest version to his own



Bob publishes the merged version



Bob creates a new merged version



Alice also gets Bob’s change now

Resolving Conflicts - Merging

- Merging is required if there are conflicts on changes of the same file
- Conflicts can be resolved automatically if the changes are non-overlapping
- User intervention is required if the changes on the same file are overlapping
  - The VCS shows the difference between two changes and asks the user to manually repair the conflict
    - Combining the changes in some way
    - Selecting one change in favor of the other
    - Reverting both changes
- Resolving conflicts on binary files can be difficult
  - The user usually has to discard one version

Branches

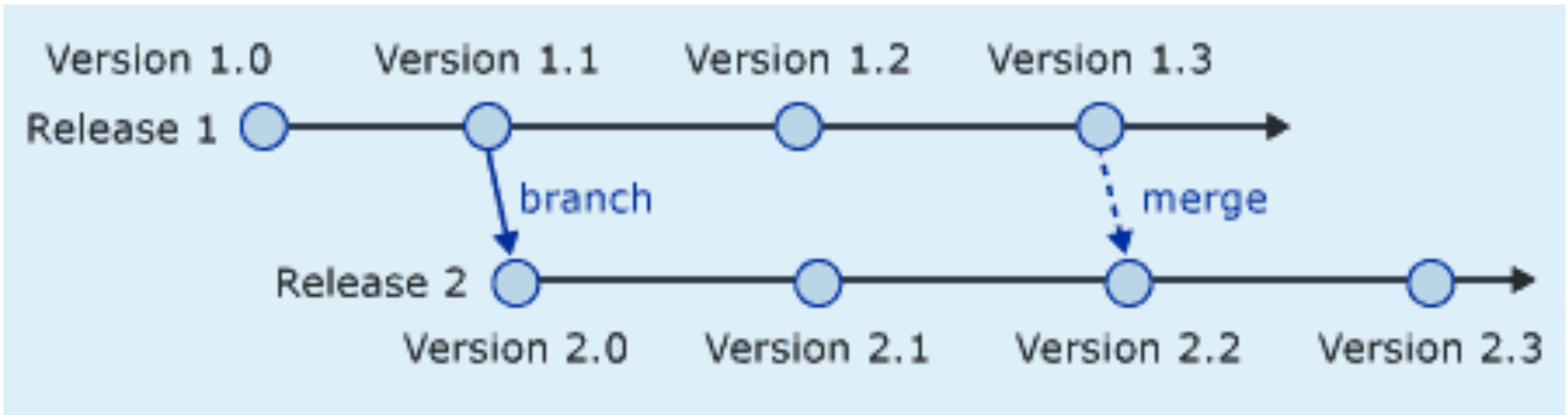
- Branch: A new copy of the files in a repository
- Branches can be
  - Developed independently
  - Given their own version number in the VCS
  - Merged later
- Branches are often used to explore experimental development or isolate development activities
  - Main branch for development or maintenance
  - New branches for experimental features or nontrivial maintenance work

Variants and Revisions

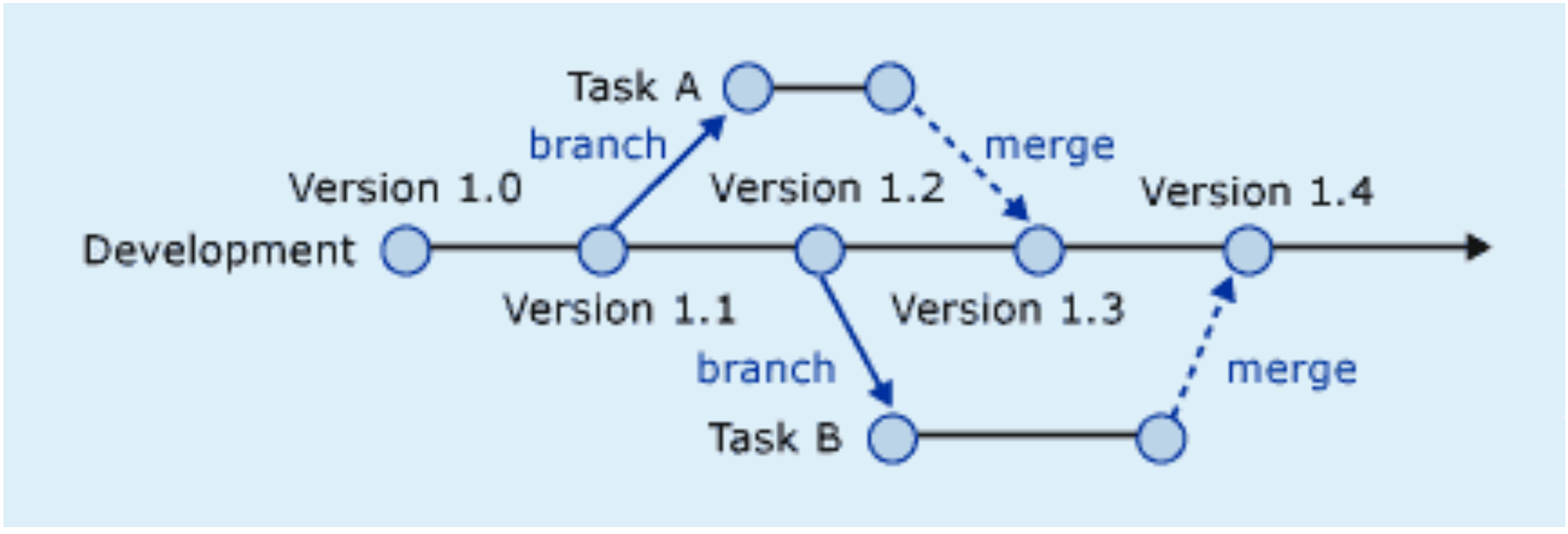
- **Revision** replaces prior revision
- **Variant** coexists with other variants
- Both revision and variant are described by **version**
- **Release:** Published and named version

Branches (Cont.)

- Branches per Release



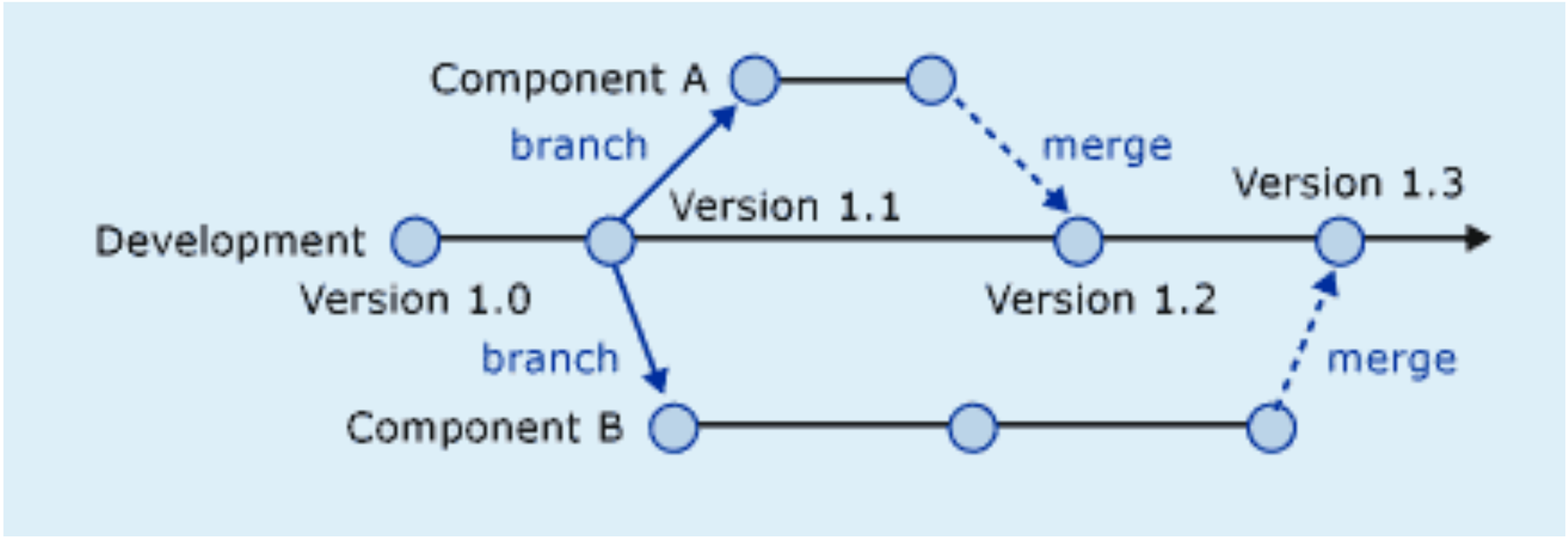
- Branches per Task



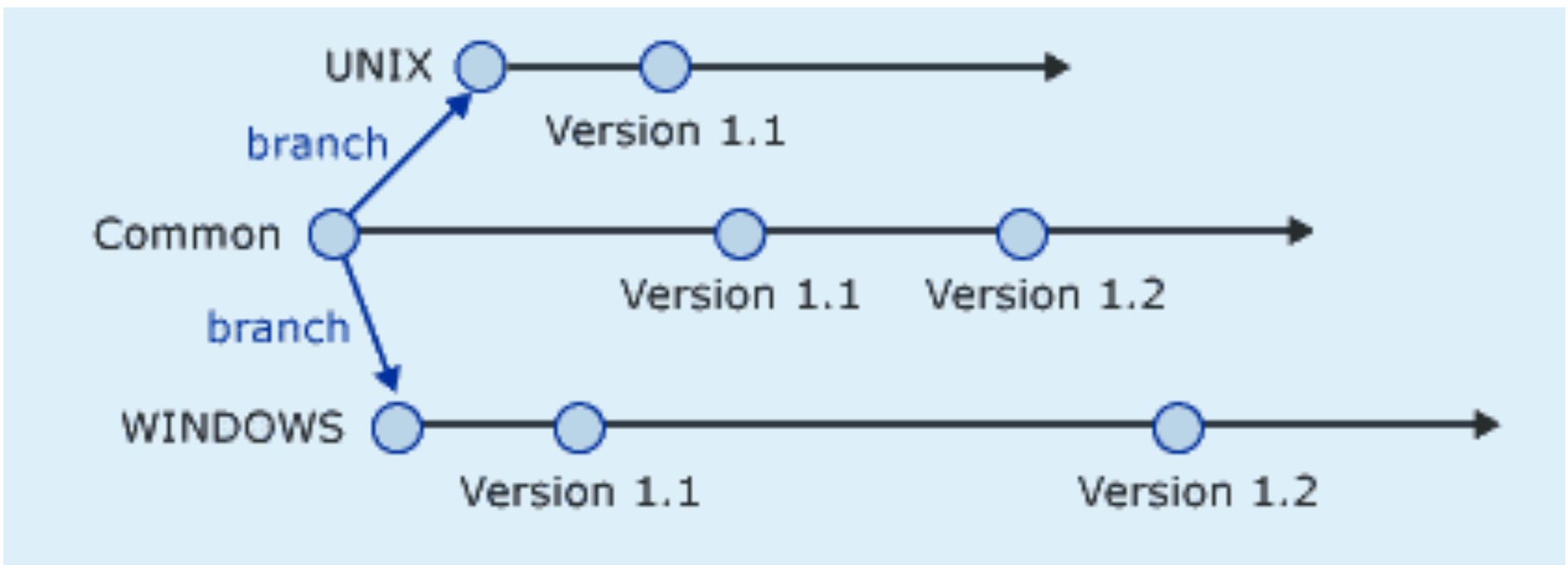
https://blog.codinghorror.com/software-branching-and-parallel-universes/

Branches (Cont.)

- Branches per Component



- Branches per Technology



https://blog.codinghorror.com/software-branching-and-parallel-universes/

Semantic Versioning for Releases

- Given a version number MAJOR.MINOR.PATCH, increment:
  - MAJOR version when you make incompatible API changes;
  - MINOR version when you add functionality in a backwards-compatible manner;
  - PATCH version when you make backwards-compatible bug fixes

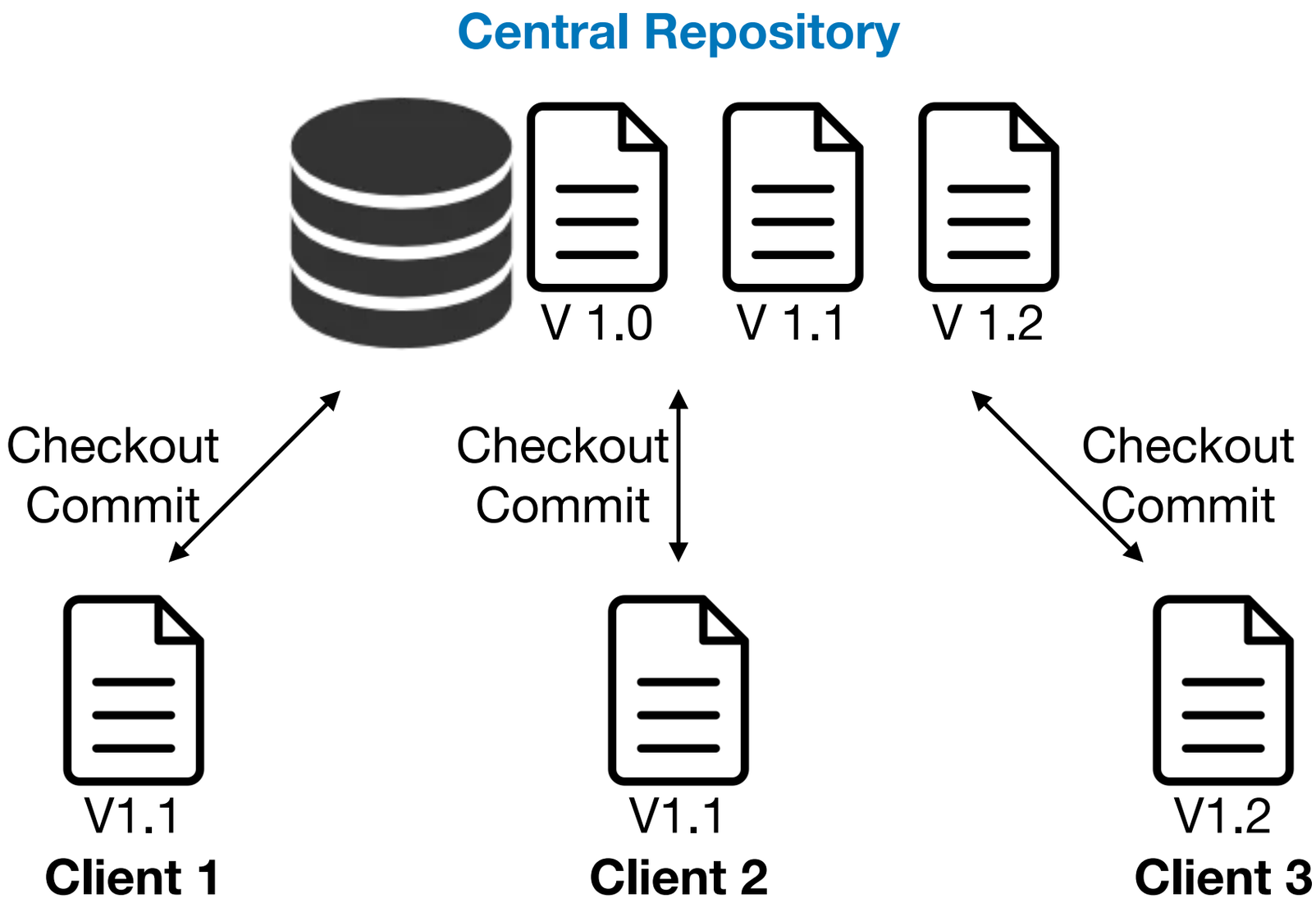
Classes of Version Control Systems

- Local version control
  - Local history of files: SCCS (1970s), RCS (1982)
- Central version control
  - Examples: CVS (1990), SVN (2004), Perforce, Visual SourceSafe
- Distributed version control
  - Examples: Git (2005), Mercurial, Bitkeeper, ClearCase

Centralized VCS

- A central master server maintains the “official copy” of the files
  - All past versions are stored on the central master server
  - Clients synchronize with the central server (update, commit)
- Clients can make “checkouts” of the central repository to own local copy
  - You can make local modifications
  - Your changes are not versioned
- Clients make “checkins” to write changes back to the server
  - Your checkin increments the repo’s version

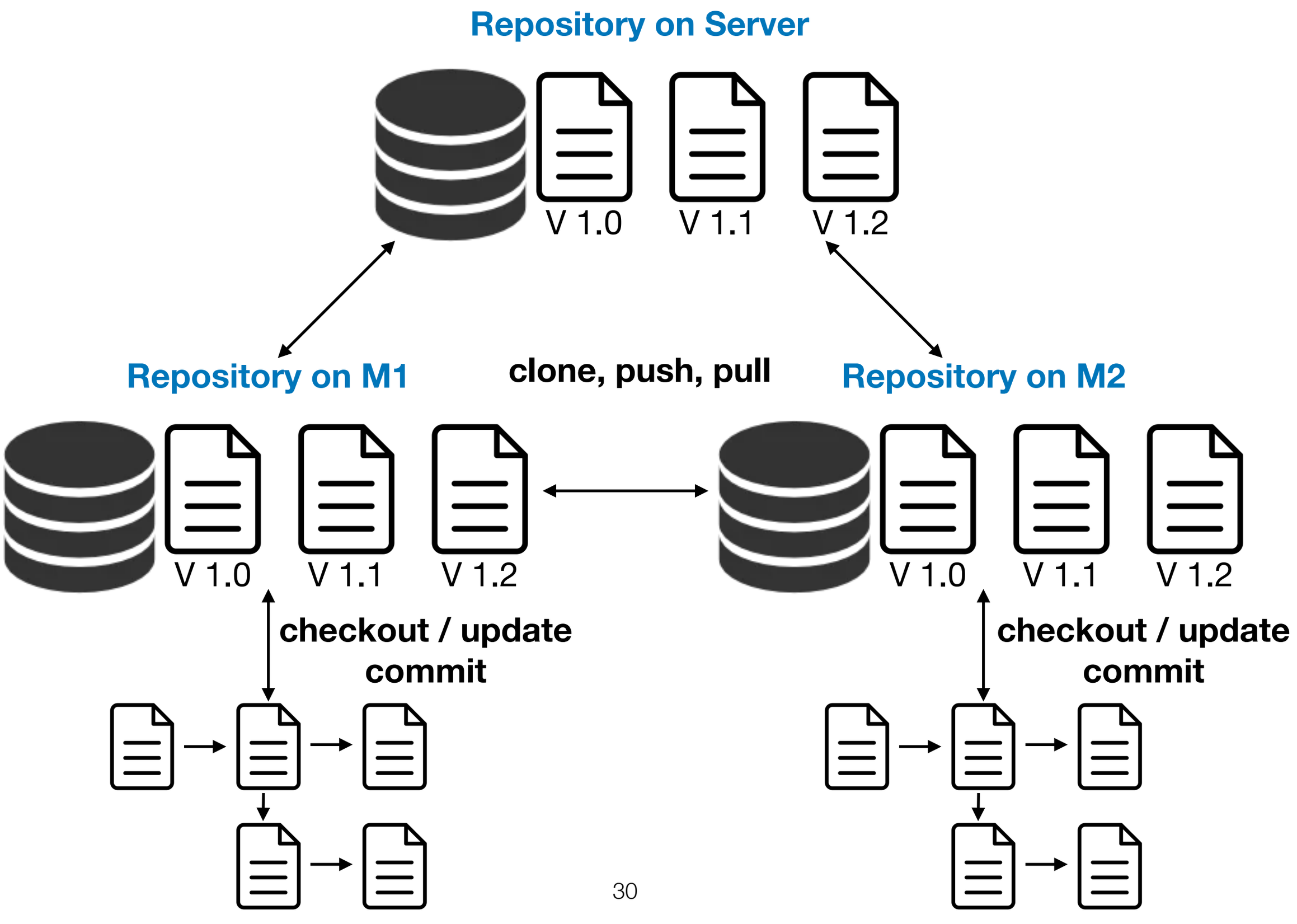
Centralized VCS (Con.)



Distributed VCS

- Clients and the server maintain multiple repositories
  - The central server is optional
  - The local repository on a client machine is a complete copy of the repository on the remote server
- Clients “clone” and “pull” changes from the server repository
- Clients can work on their local copies independently
  - Checkin and checkout from/to local repo.
  - Commit changes to local repo.
- Clients can “push” (publish) changes back to server

Distributed VCS (Con.)



Distributed Versions of Git

- Each modification to the central repository increments the version number of the overall repository in Subversion
- Versions are not globally coordinated/sorted in Git
  - Each user has a copy of the repository
  - Changes are committed locally to their local copy of the repository before being pushed to the remote repository
  - Changes are managed with unique IDs through hashes
  - Each commit is associated with a unique *SHA-1 hash*

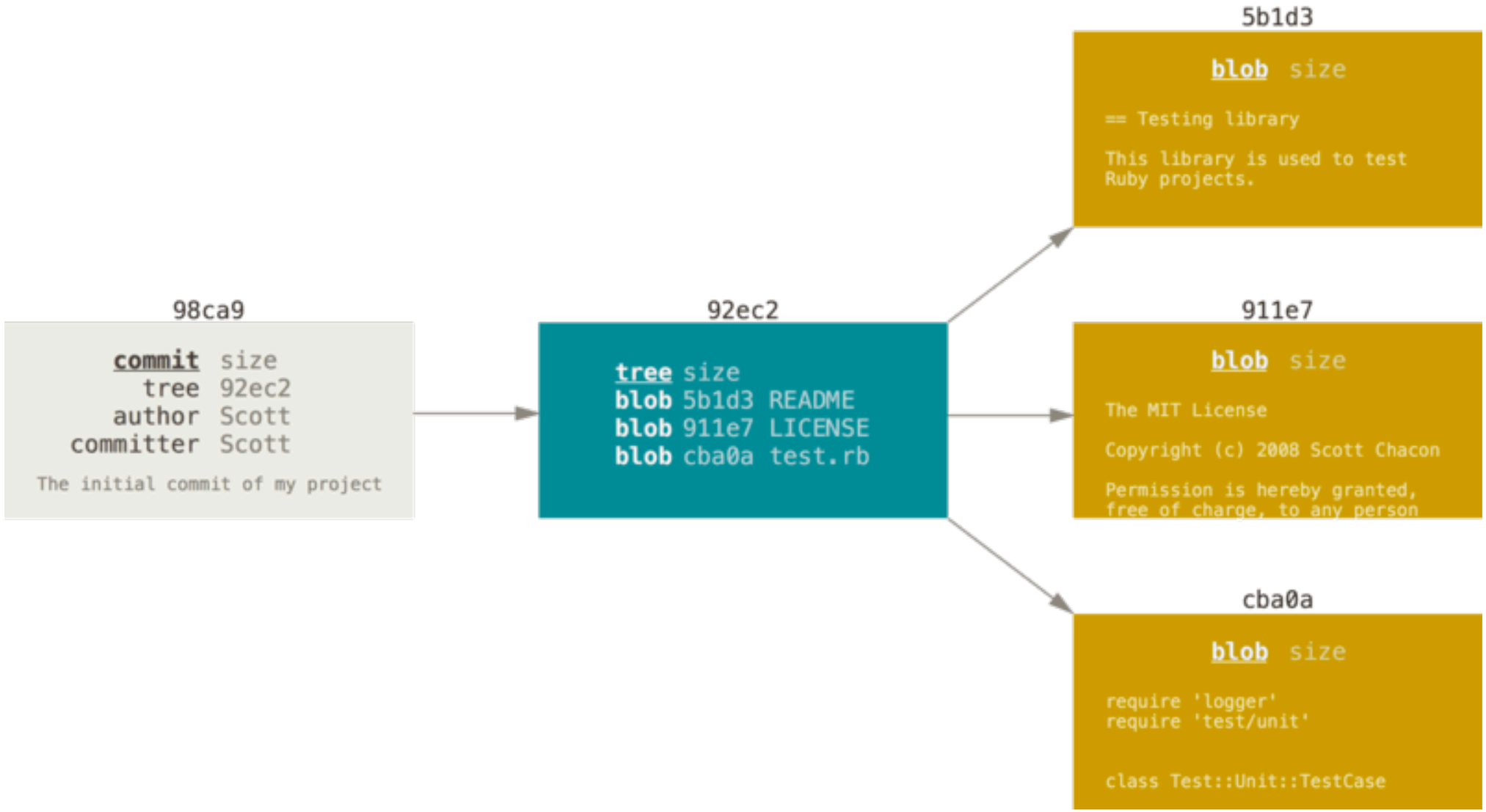
```
commit 10adf23fedece40f5003d2cfb3f2044a331ccfec
Author: Hong Xu <hongxu@cuhk.edu.hk>
Date:   Wed Jan 6 15:38:22 2021 +0800

    two principles

commit 132a3e98ab027e07d61b2ee522070a1c951cf0c5
Author: Hong Xu <hongxu@cuhk.edu.hk>
Date:   Tue Jan 5 21:35:39 2021 +0800

    misc in sec4
```

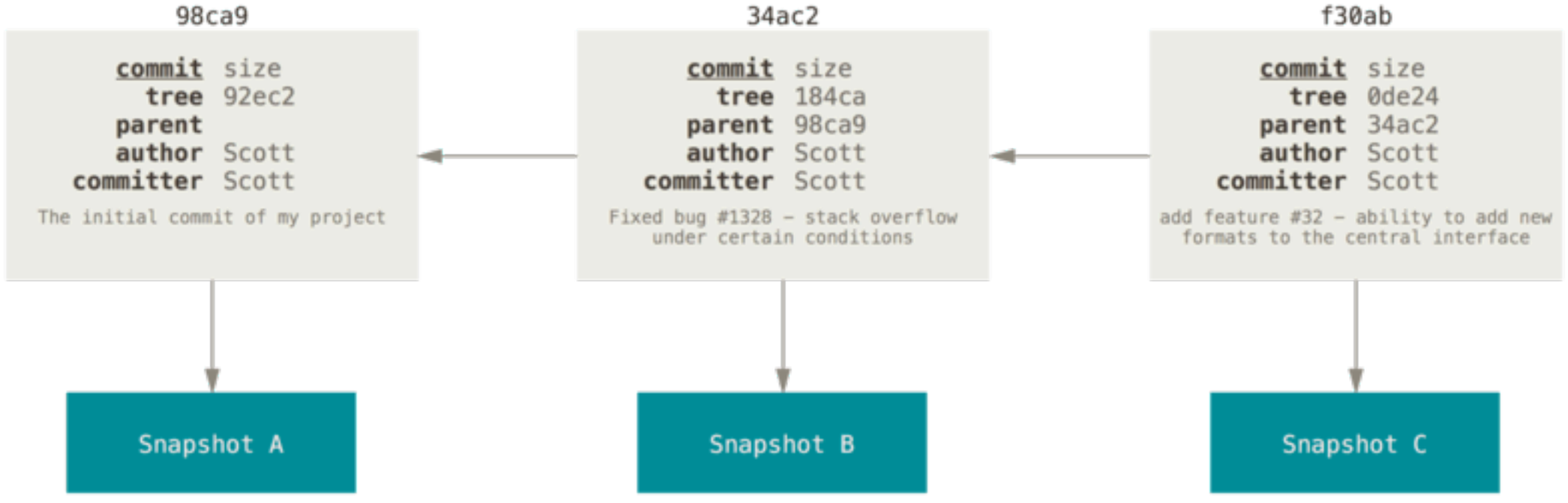
What Are In A Git Commit?



A commit tree

Relationships Among Commits

- Relationships among changes are tracked in a successor graph
  - Possible to merge select changes
  - Possible to rewrite the history as long as not shared remotely



Commits and their parents

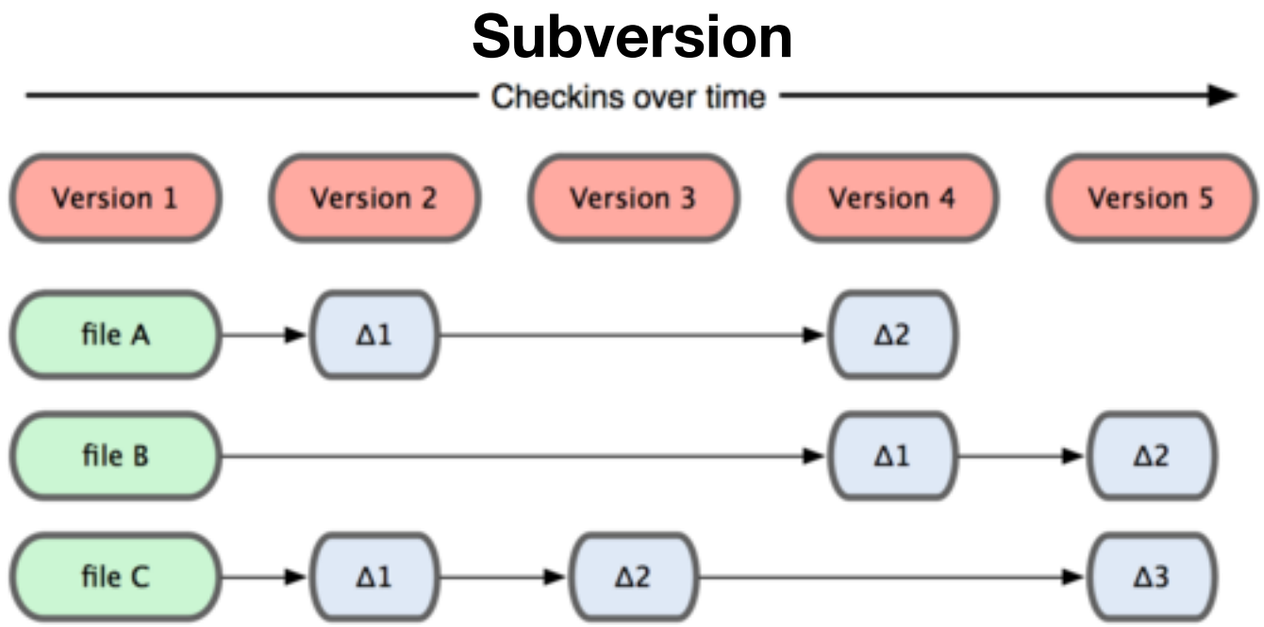
Git Snapshots

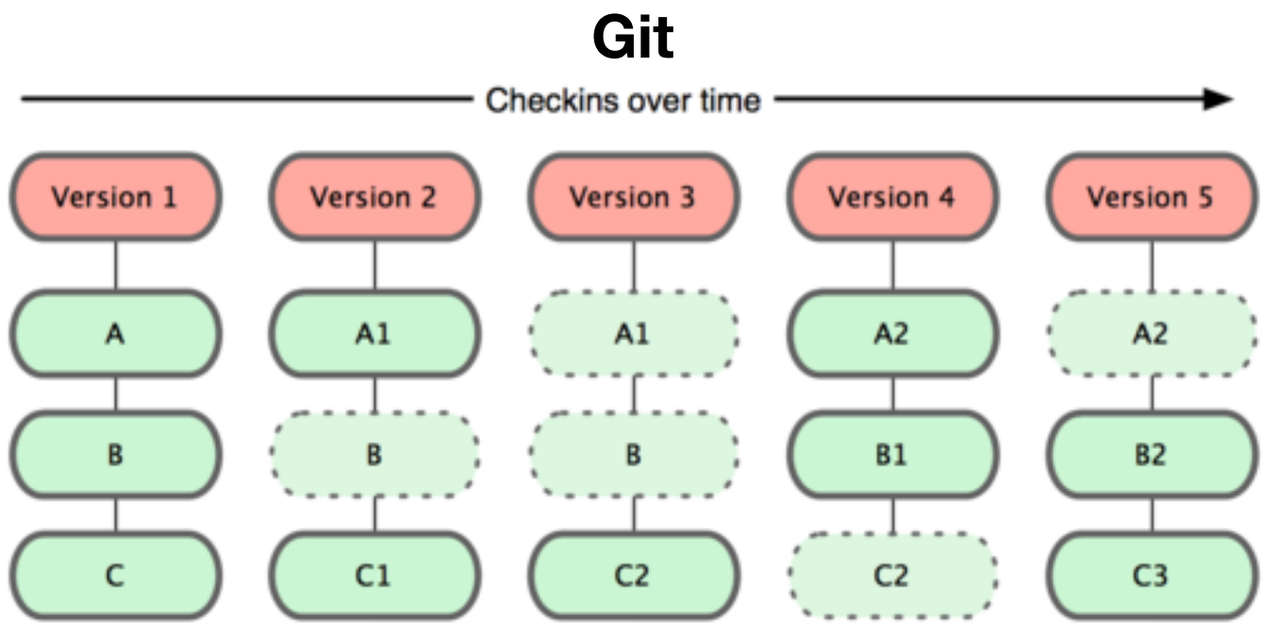
- Some centralized VCS, e.g., SVN, tracks version data (differences) on each individual file.

- Git keeps "snapshots" of the entire project

  - Each version of the project includes a copy of every file in the project

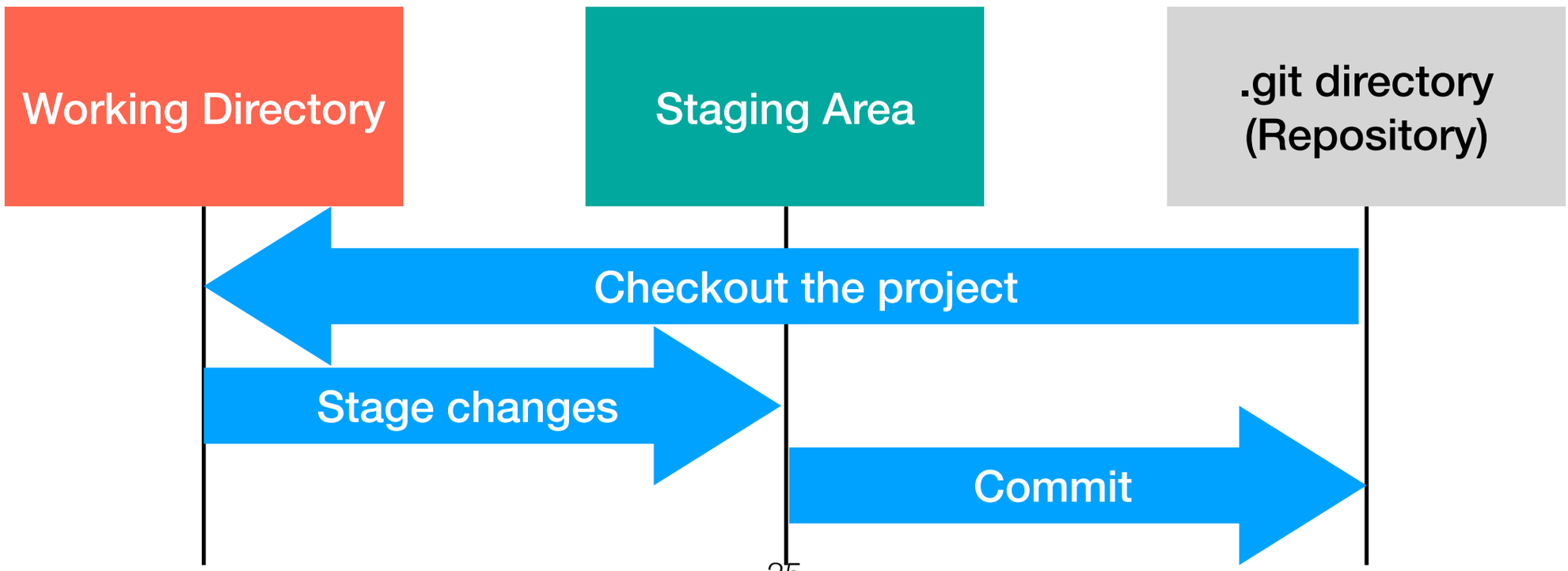
  - Faster but requires more storage





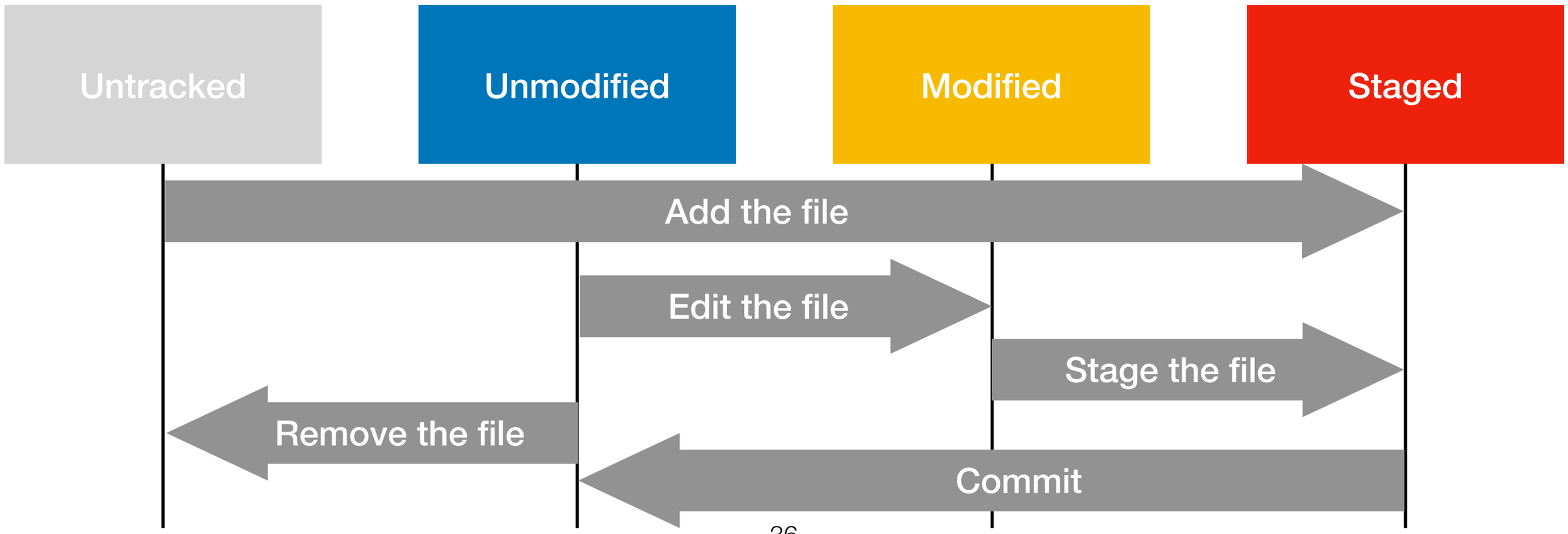
Local Git Areas

- In your local copy, files can be
  - In your local .git repository (*committed*)
  - Checked out and modified, but not yet committed (*working copy*)
  - In a “staging area”
    - Staged files are ready to be committed.
    - A commit saves a snapshot of all stages files.



Basic Git Workflow

- **Add** new files to be tracked
- **Modify** files in your working directory
- **Stage** files, adding snapshots of them to the staging area
- **Commit** changes, which stores the snapshot permanently to your .git repository



Working with Remotes

- Adding a remote repository
- Cloning a complete copy of a remote repository
  - Normal checkout and commit operations are local
- Fetching and pulling from your remotes to get missing versions
  - *Fetch* only downloads the data without merging it with your local copy
  - *Pull* downloads and merges the remote branch into your current branch
- Pushing your local changes to your remote repository

Branching and Merging in Git

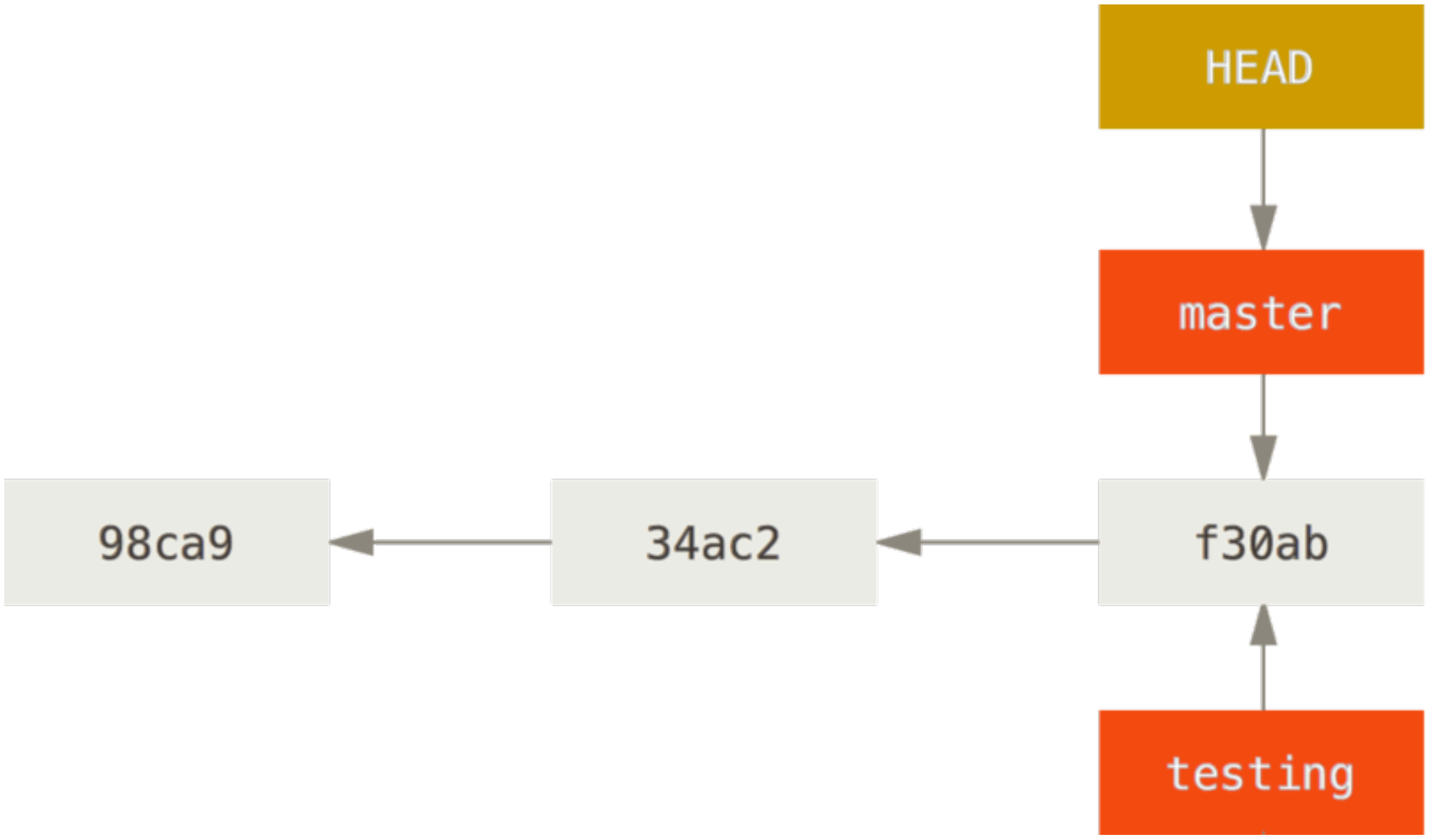
- Git uses branching heavily to switch between multiple tasks
- When you merge two branches, the conflicting files will contain <<< and >>> sections to indicate locations where Git cannot resolve the conflicts automatically

```
<<<<<<< HEAD:index.html
<div id="footer">todo: message here</div>
=======
<div id="footer">
    thanks for visiting our site
  </div>
  >>>>>>> SpecialBranch:index.html
```

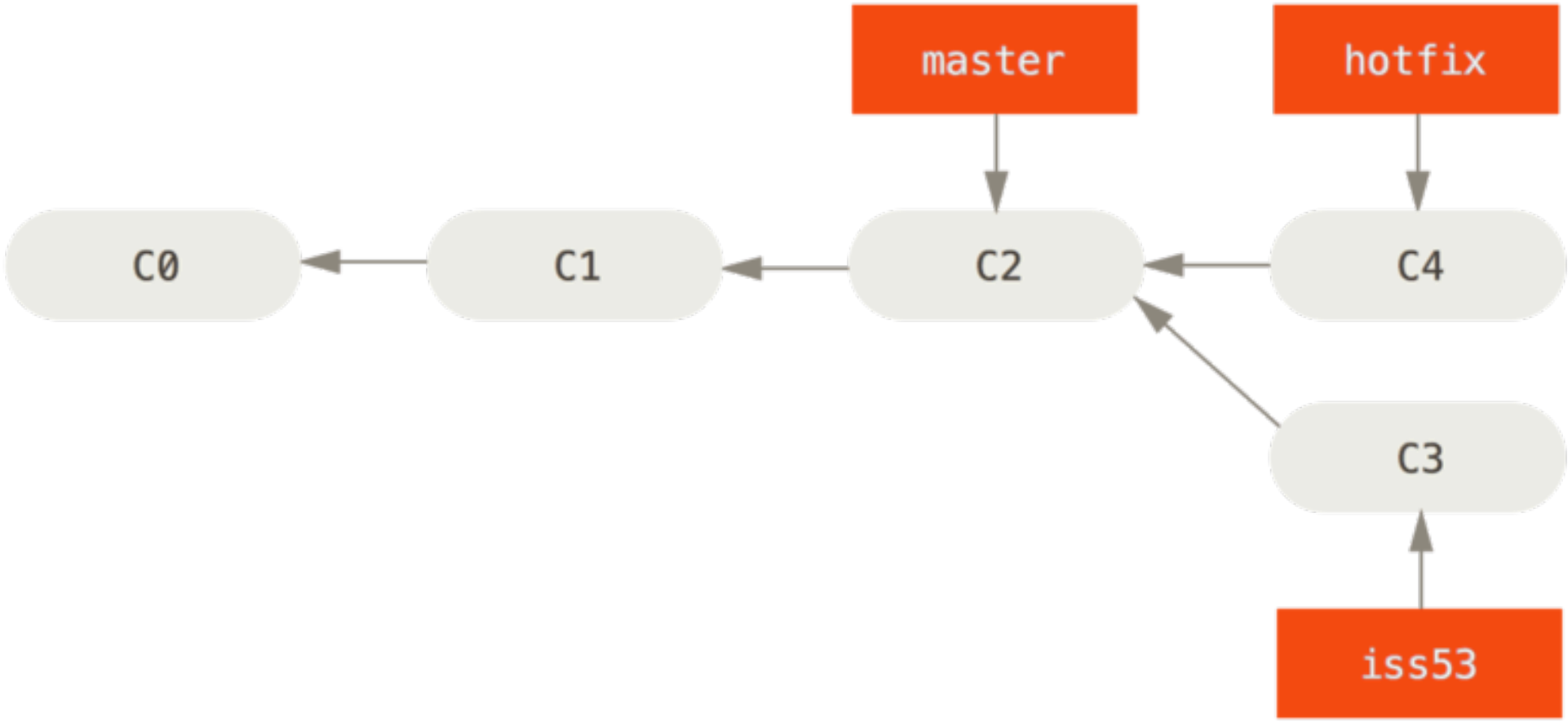
Branch 1's version

Branch 2's version

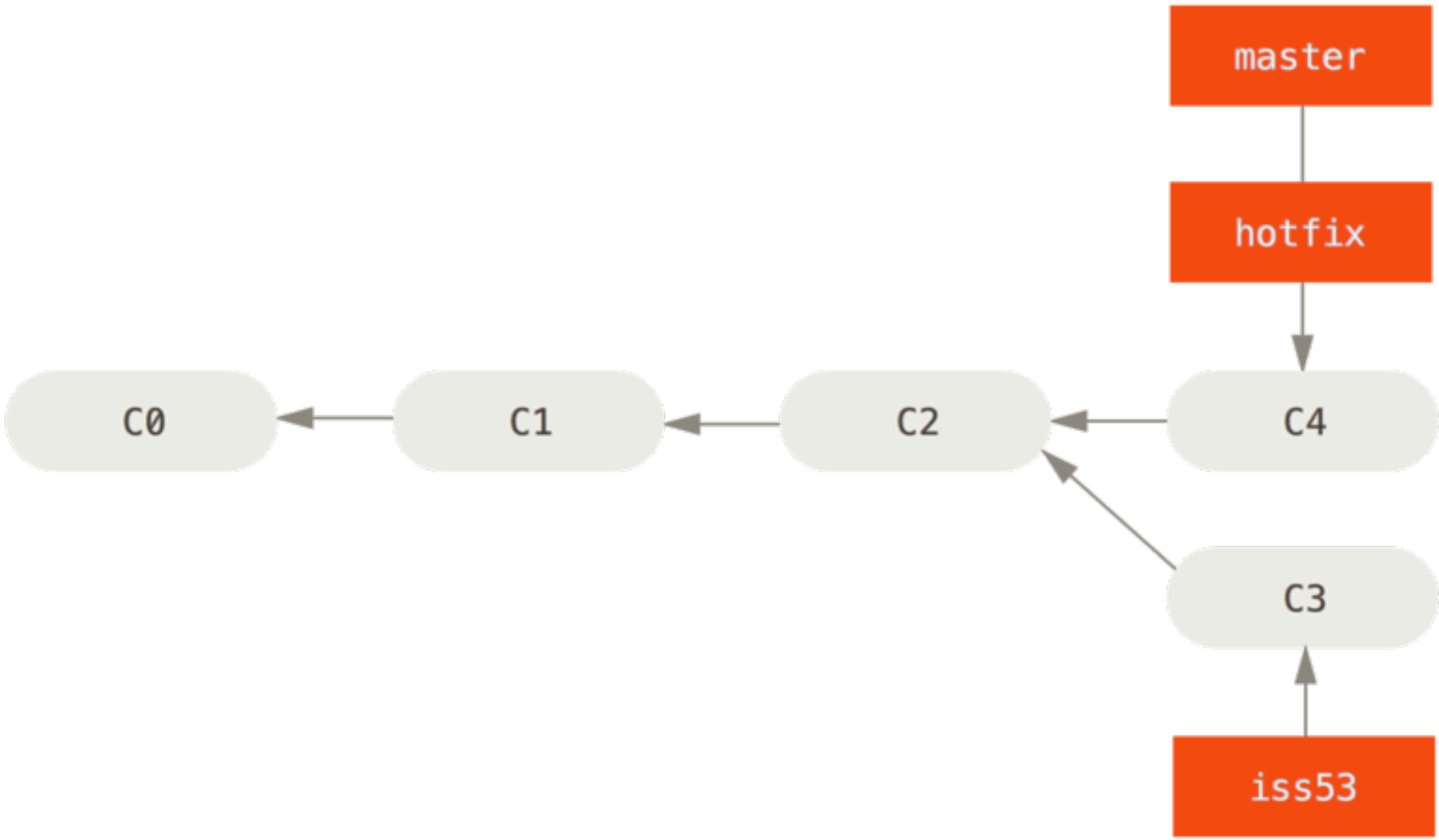
Git Branches - HEAD Pointer



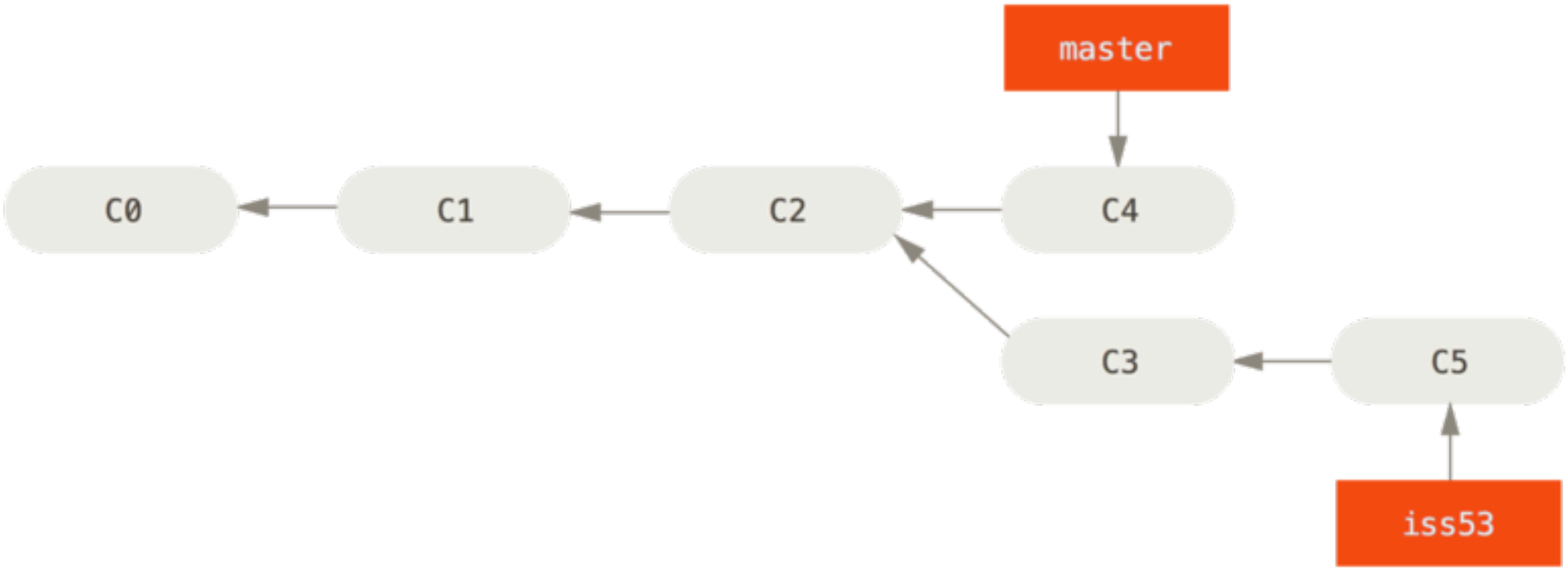
Use Branches for Different Tasks



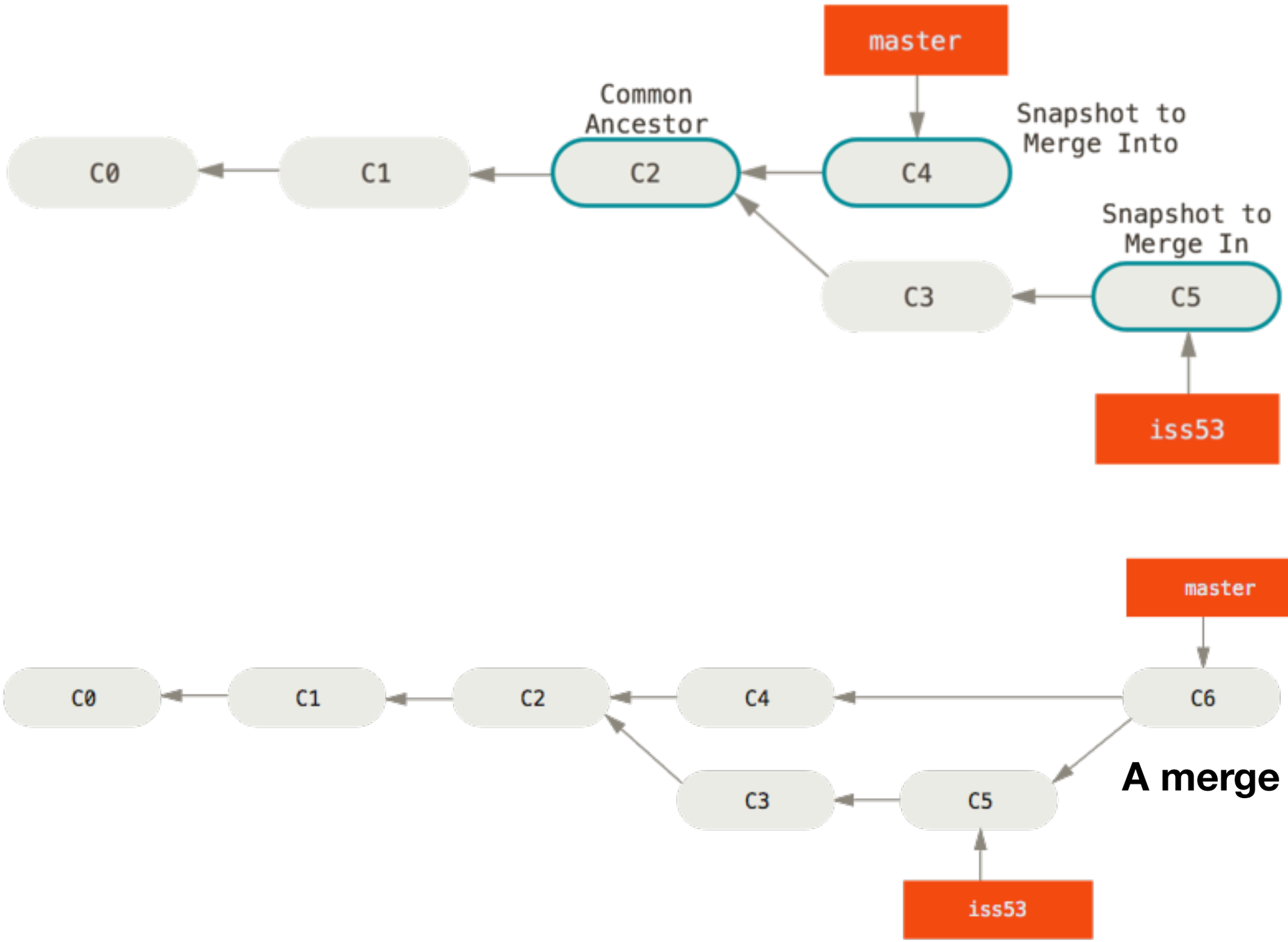
Fast-Forwarding Merge in Git



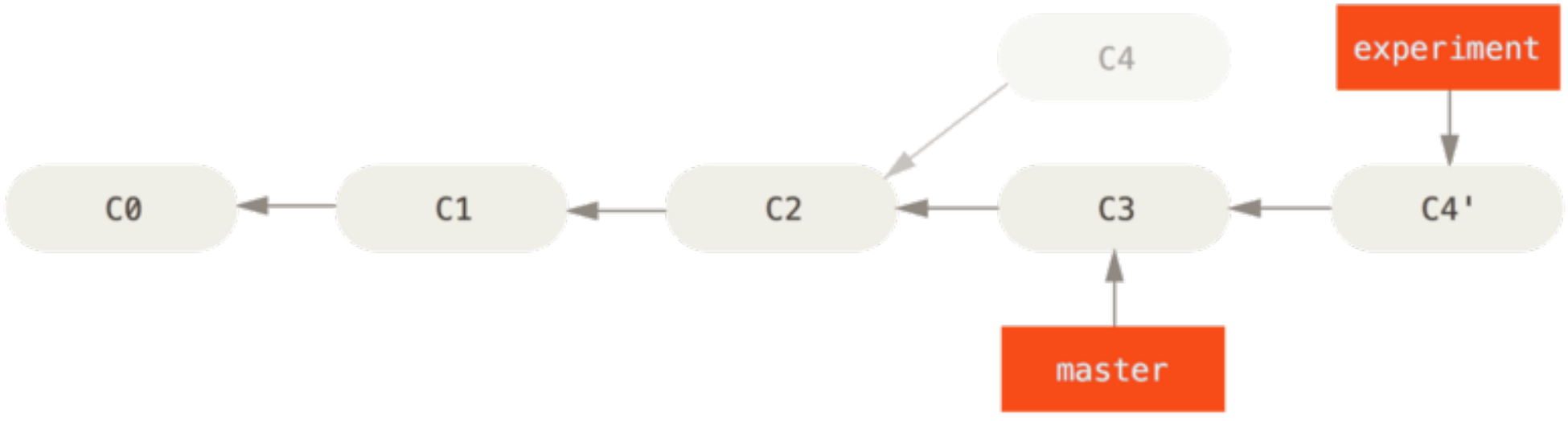
Diverging Development With Git Branches



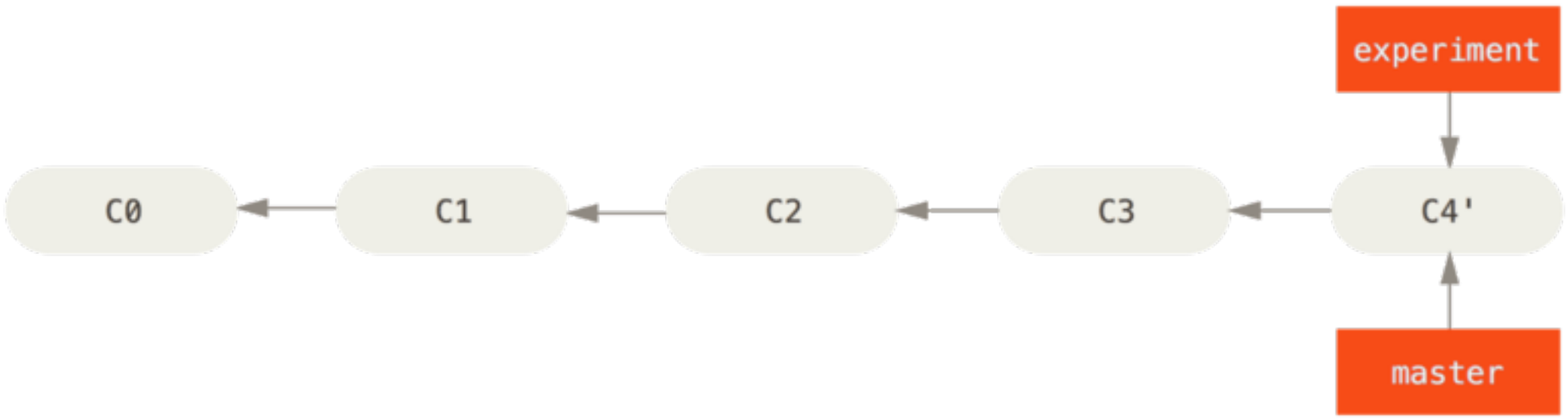
Three-Way Merge in Git



Git Rebasing

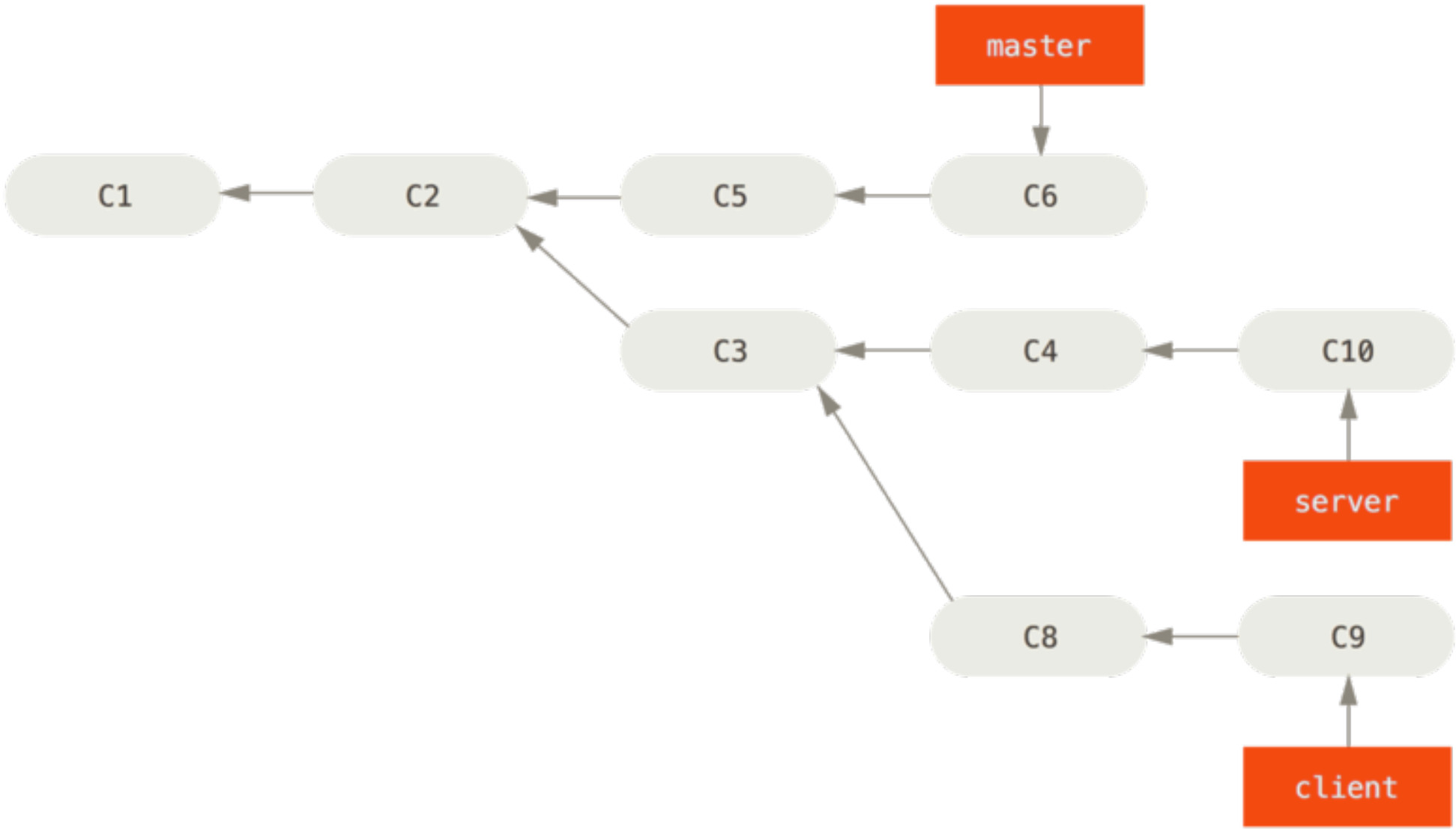


Rebasing the change introduced in C4 onto C3



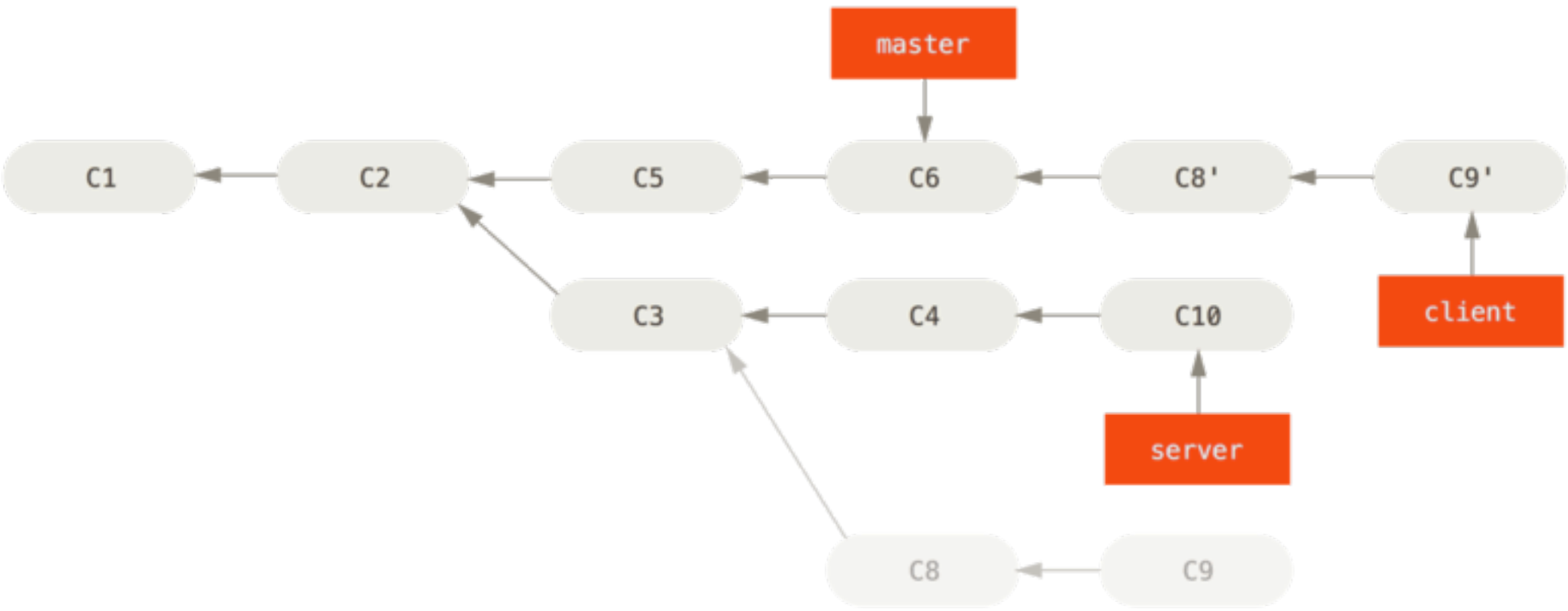
Fast-forwarding the master branch

Git Rebasing --onto



A history with a topic branch off another topic branch

Git Rebasing --onto (Cont.)



Rebasing a topic branch off another topic branch

GitHub

- GitHub.com is a website for hosting remote Git repositories
  - Many open-source projects use it, e.g., Linux kernel
  - Open-source projects can be hosted for free
  - Private repositories are free and unlimited since Jan 7, 2019
    - Educational plan: https://education.github.com/
- Question: Do you have to use GitHub to use Git?
  - Answer: No. Git can be used locally.
  - There are alternatives to GitHub, e.g., BitBucket, GitLab, SourceForge, etc.
  - You can set up your own server that speaks the Git protocol to host repositories

Learn More About Git

- Tutorial Session
  - This week too
- The Pro Git book, written by Scott Chacon and Ben Straub
  - Available online: https://git-scm.com/book/en/v2/