Advanced Software Development: Git

CPSC 501: Advanced Programming Techniques
Fall 2022

Jonathan Hudson, Ph.D
Assistant Professor (Teaching)
Department of Computer Science
University of Calgary

Friday, September 23, 2022

UNIVERSITY OF CALGARY

Not an acronym

Not an acronym

- (from the source code read-me)
- "git" can mean anything, depending on your mood.
  - random three-letter combination that is pronounceable, and not actually used by any common UNIX command. The fact that it is a mispronunciation of "get" may or may not be relevant.
  - stupid. contemptible and despicable. simple. Take your pick from the dictionary of slang.
  - "global information tracker": you're in a good mood, and it actually works for you. Angels sing, and a light suddenly fills the room.
  - "g0d@mn idiotic truckload of sh*t": when it breaks

The Rise of Git

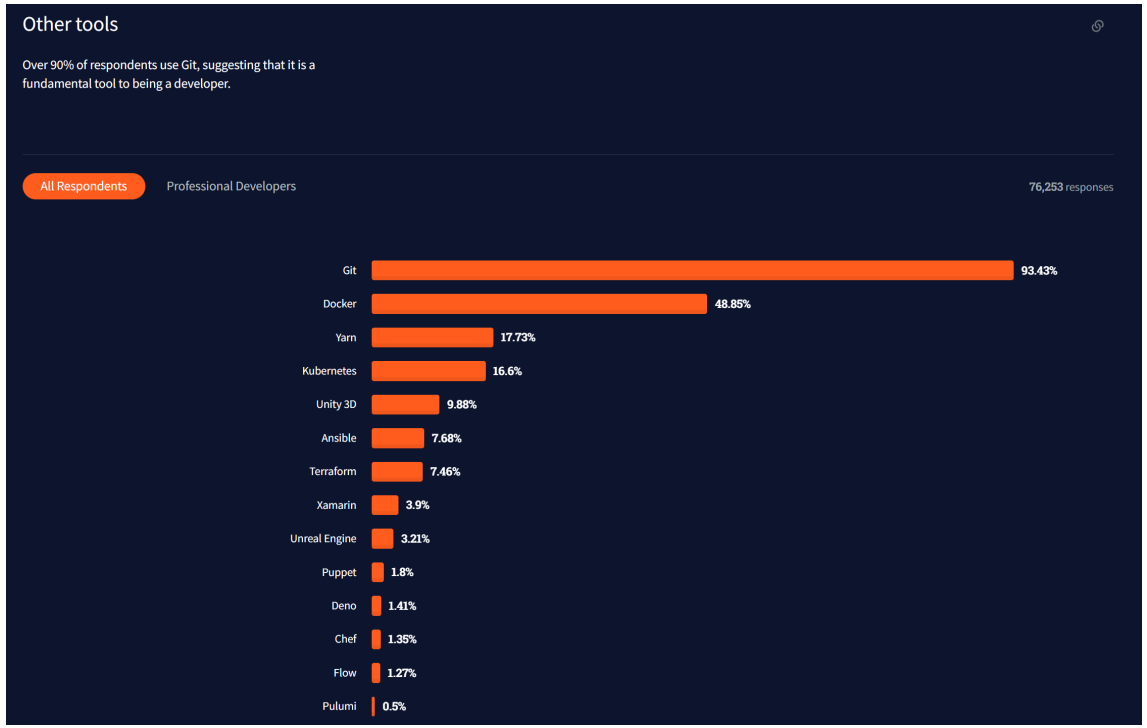
- *Git* is the most popular implementation of a **distributed version control system**.
- Development started in 2005 by **Linus Torvalds.**
  - Linux kernel source host dispute with BitKeeper [had been using for linux kernel since 2002]
  - Same reason resulted in another DVCS -> Mercurial
- It is used by many popular open source projects as well as many commercial organizations.

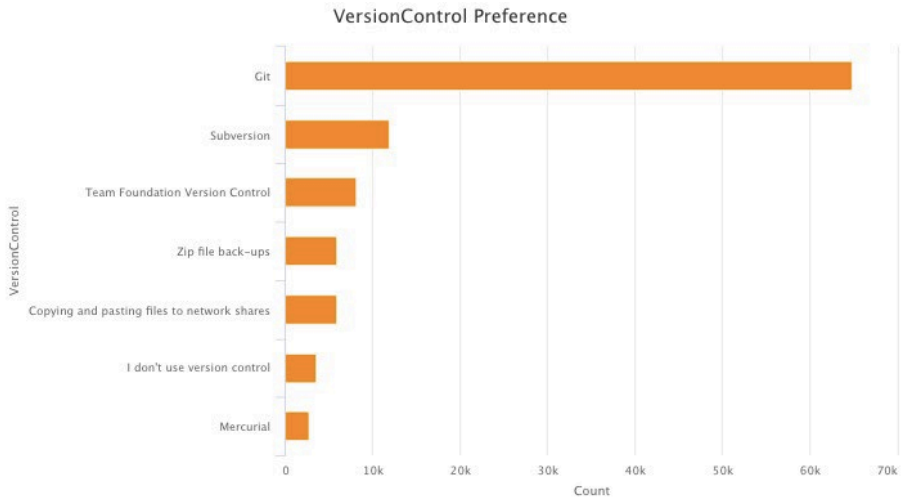
The Rise of Git - Goal

1. Take Concurrent Versions System (CVS) as an example of what not to do; if in doubt, make the exact opposite decision.
2. Support a distributed, BitKeeper-like workflow. (‘He’s dead Jim’ -> BitKeeper)
3. Include very strong safeguards against corruption, either accidental or malicious.

Why Git?

- Git's the most popular version control system in the industry.
- https://insights.stackoverflow.com/survey/2021
- Most other popular VCS are similar to Git





Source: Stackoverflow 2018 survey

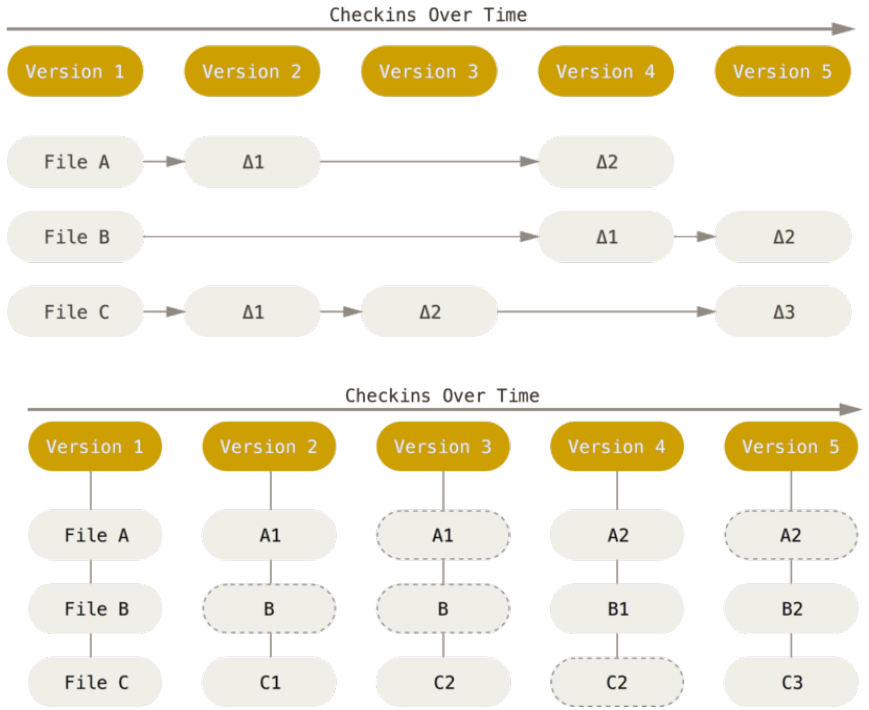
UNIVERSITY OF CALGARY

What is Git?

- Git is distributed
- i.e. there is generally a remote repo. (like the single svn one) and a local repo on your own machine
  - SVN required repo to be only local, or only remote
  - Git lets each developer have their own version of repo
  - Each developer can make changes and make commits to own repo and periodically push/pull from remote to bring together development
  - Frees programmer, code on a plane and still do multiple local commits

Tracking changes?

- Not a delta-based system where only changes are tracked
- Git has a snapshot per commit
  - Old files are not-resaved if they haven't changed
  - But model allows tools to think of each image as a full file system at a time



https://git-scm.com/book/en/v2

This is how you do it

Git: *New* Version Control Terminology

SHA

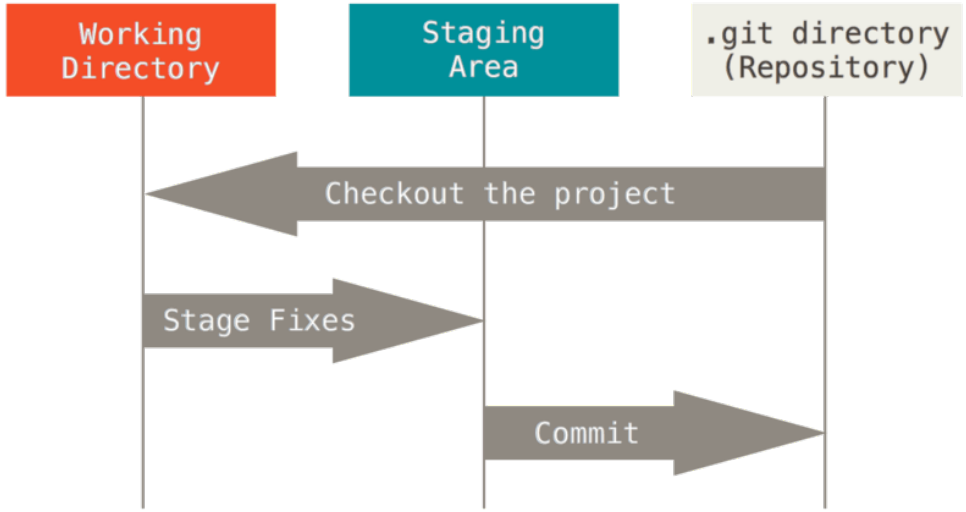
- A SHA is basically an ID number for each commit.
- Ex. E2adf8ae3e2e4ed40add75cc44cf9d0a869afeb6
- Instead of version numbering (SVN)

Staging Area

- You can think of the staging area as a prep table where Git will take the next commit.
- Files on the Staging Index are ready to be added to the repository.

Git: Getting Started

- Three trees of Git
  - Repo (.git)
    - Where the metadata and object database is stored
  - Staging area/Index
    - Proposed next commit snapshot
  - Working directory
    - Where your IDE is working on project



https://git-scm.com/book/en/v2

- Example install
- https://git-scm.com/book/en/v2/Getting-Started-Installing-Git
- Most IDEs have it integrated in GUI form

Git: Status of Files

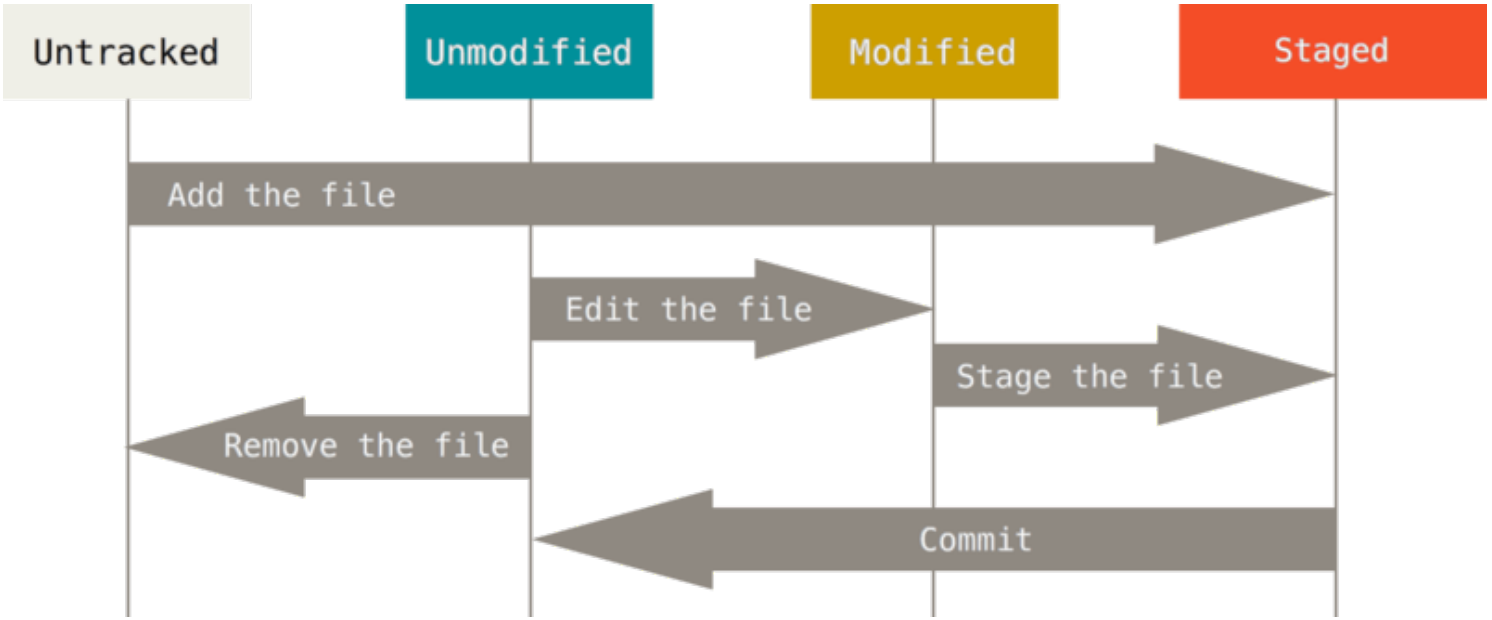


Figure 8. The lifecycle of the status of your files

https://git-scm.com/book/en/v2

Git: Basic Commands

- **git init** – *Initialize a Git repository/working directory*
  - ***git init NAME***
- **git status** – *Status of your working directory*
  - ***git status***
- **git add <filename>** or **git add .** *(for all files in your working directory)*
- **git commit** – *Stash changes in your working directory to repo*
- **git log** – *View your commit history*

- **git clone** – *Create an identical copy (example is pulling from GitHub/Gitlab)*

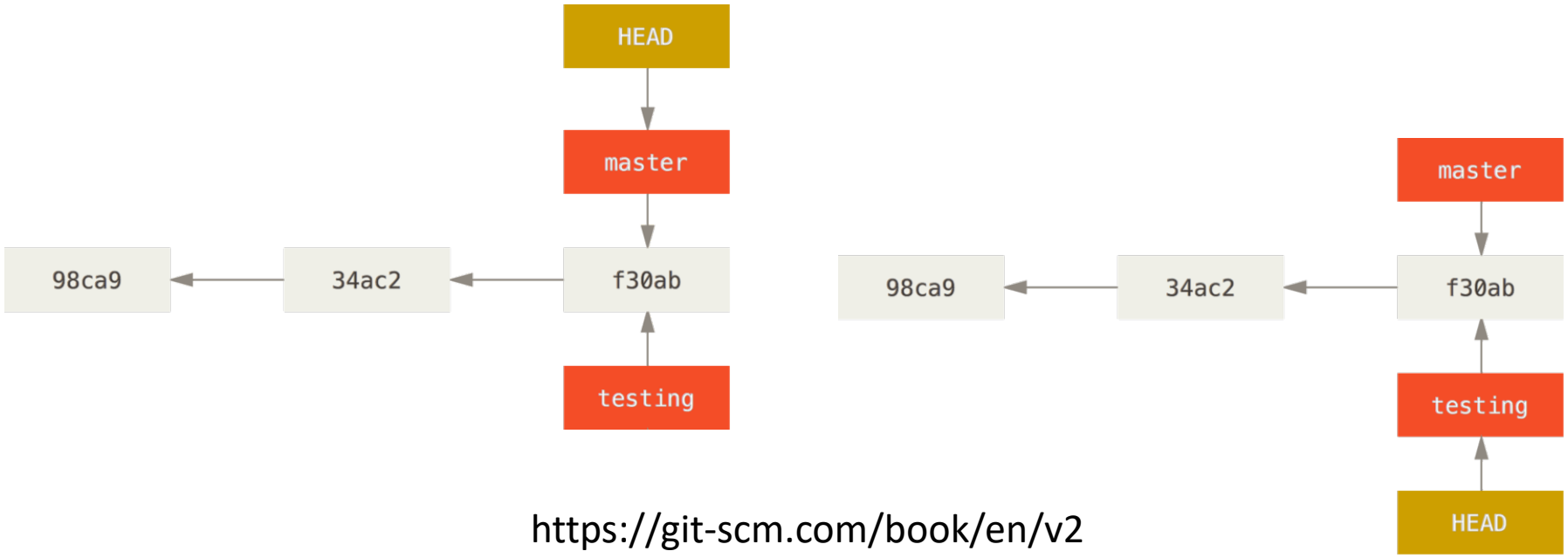
Branching

Git: Commit Tree -> Branching

- When you checkout a branch name you switch HEAD to it

  git branch testing

  git checkout testing



https://git-scm.com/book/en/v2

Git: Commit Tree -> Branching

- Then when you commit you add to that branch only

  git add test.rb

  git commit -a -m 'made a change'

https://git-scm.com/book/en/v2

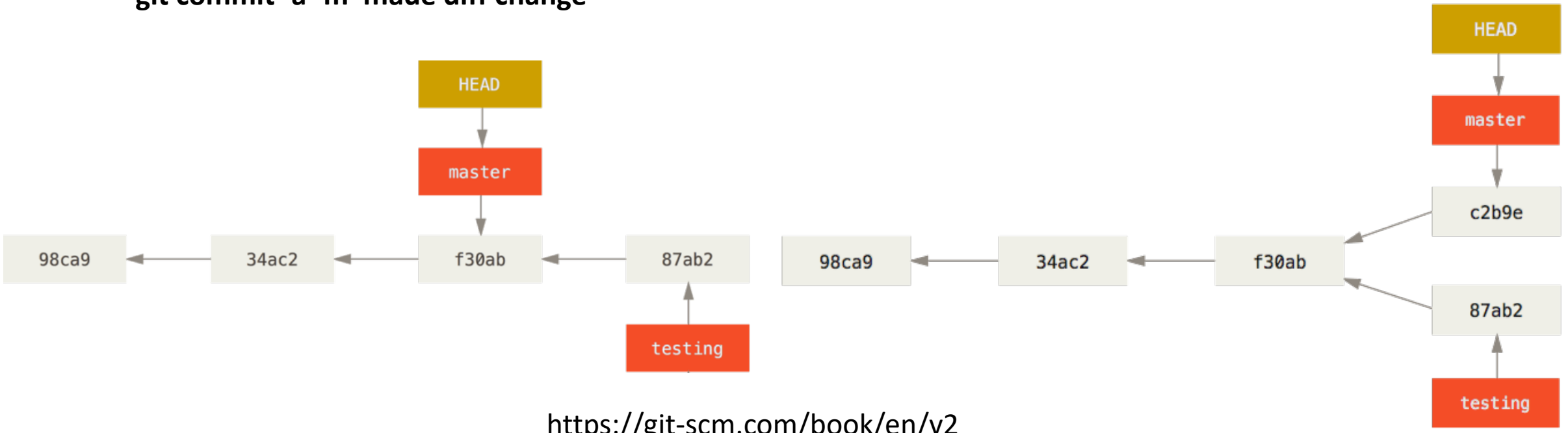
Git: Commit Tree -> Branching

- You can move back to master and commit to it, now you have parallel dev.

  git checkout master

  git add x.java

  git commit -a -m 'made diff change'

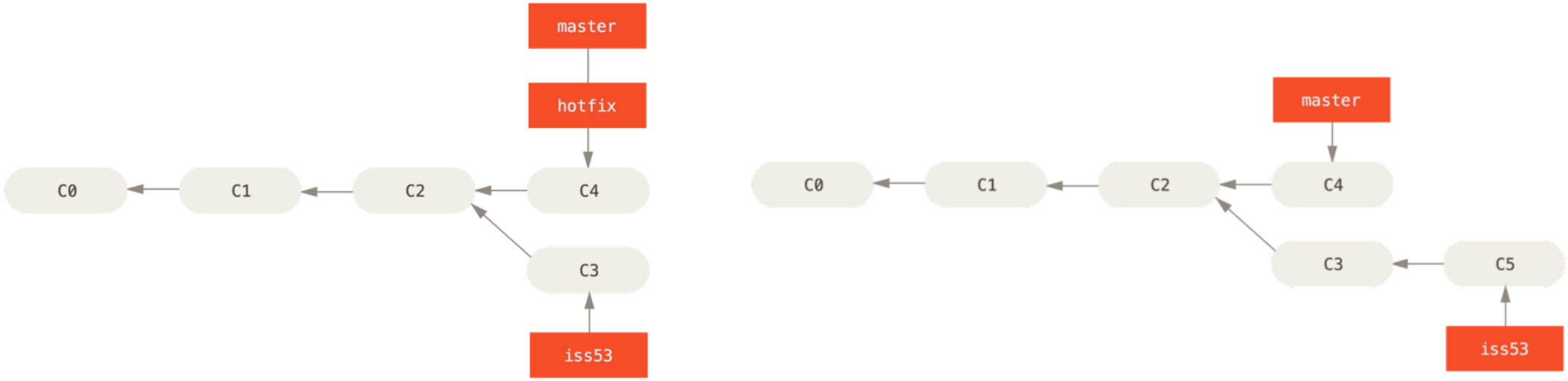


https://git-scm.com/book/en/v2

Merging

Git: Commit Tree -> Merging

- Let’s consider you were working on issue 53, then switched back to hotfix master, then you went back to work on issue 53. Now you are done, how do introduce changes back to mainline



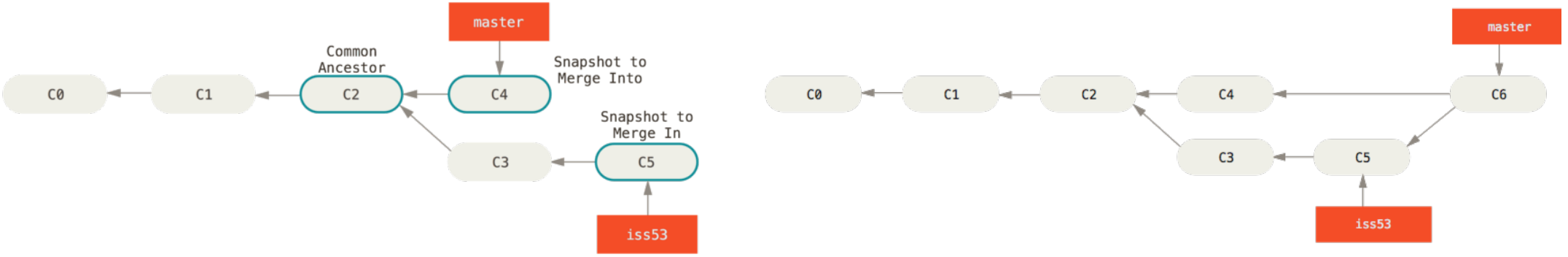
https://git-scm.com/book/en/v2

Git: Commit Tree -> Merging

- Move to master, then merge iss53 into it (results in new commit)

  git checkout master

  git merge iss53



https://git-scm.com/book/en/v2

Git: Branching and Merging

- Why this is cool?
  - Non-linear development

```
clone the code that is in ‘master’
create a branch for issue #53 ‘iss53’
work for 10 minutes

someone asks for a hotfix for issue #102
checkout ‘master’
create a branch ‘hotfix’
fix the issue
checkout ‘master’, merge ‘hotfix’
push ‘master’

checkout ‘iss53’ and keep working
…
checkout ‘master’, merge ‘iss53’
```

Git: Conflicts

- Conflicts happen generally in 2 ways

1. Auto-mergeable
    1. The git tool fits pieces together and creates merge without involvement of user
2. Fix by hand
    1. Merge will fail
    2. Leave file in directory with conflict-resolution markers
    3. Fix these yourself (or use tool like IDE)
    4. Git add these back into attempt to merge
    5. Then merge

```
<<<<<<< HEAD:index.html
<div id="footer">contact : email.support@github.com</div>
=======
<div id="footer">
 please contact us at support@github.com
</div>
>>>>>>> iss53:index.html
```

```
<div id="footer">
 please contact us at support@github.com
</div>
```

Remotes

GitHub, UofC GitLab

- It’s a hosting medium/website for your Git repositories
- Offers powerful collaborative abilities
- A good indicator of what you code/how much you code/quality of your code

Git: Working with a remote repository

- Remote?

The common central repository

By default, remote name is **origin** and default branch is **main** *(previously master)*.

A cloned repo with generally have its remote set to origin (it is possible to add more than one remote)

Add a new remote

git remote add <shortname> <url>

How to access GitHub/UofC GitLab

- Access on https://github.com/ or https://gitlab.cpsc.ucalgary.ca

- Example repo on GitHub
    - https://github.com/intley/version-control-workshop

- Get a clone link
    - https://github.com/intley/version-control-workshop.git

git clone <repo url>

Git: Remote Commands

- push your changes into the remote repository

  git push origin master

- Pull latest changes from remote (harmless and lets you view before pull)

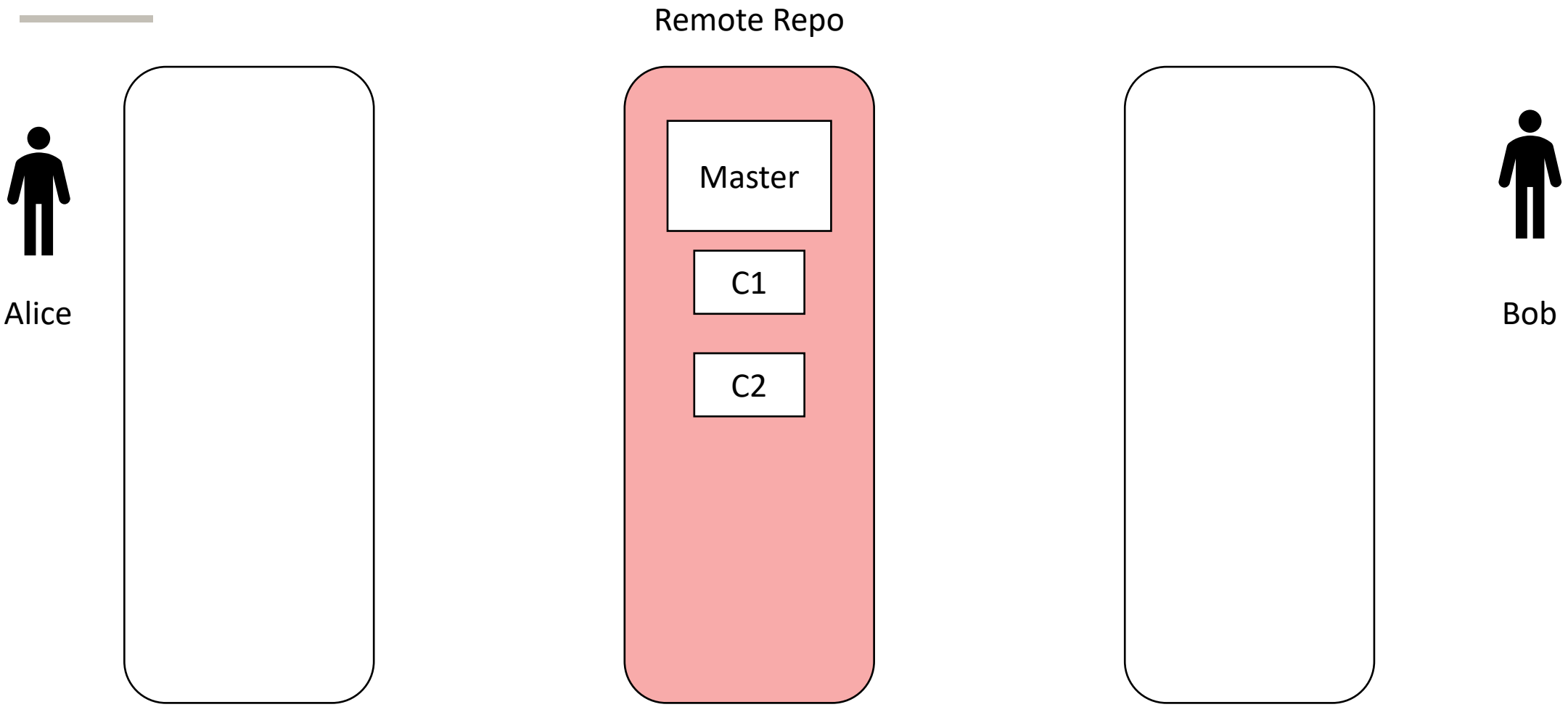
  git fetch origin

- Pull your latest changes from the remote to master and integrate (merge conflicts may occur)

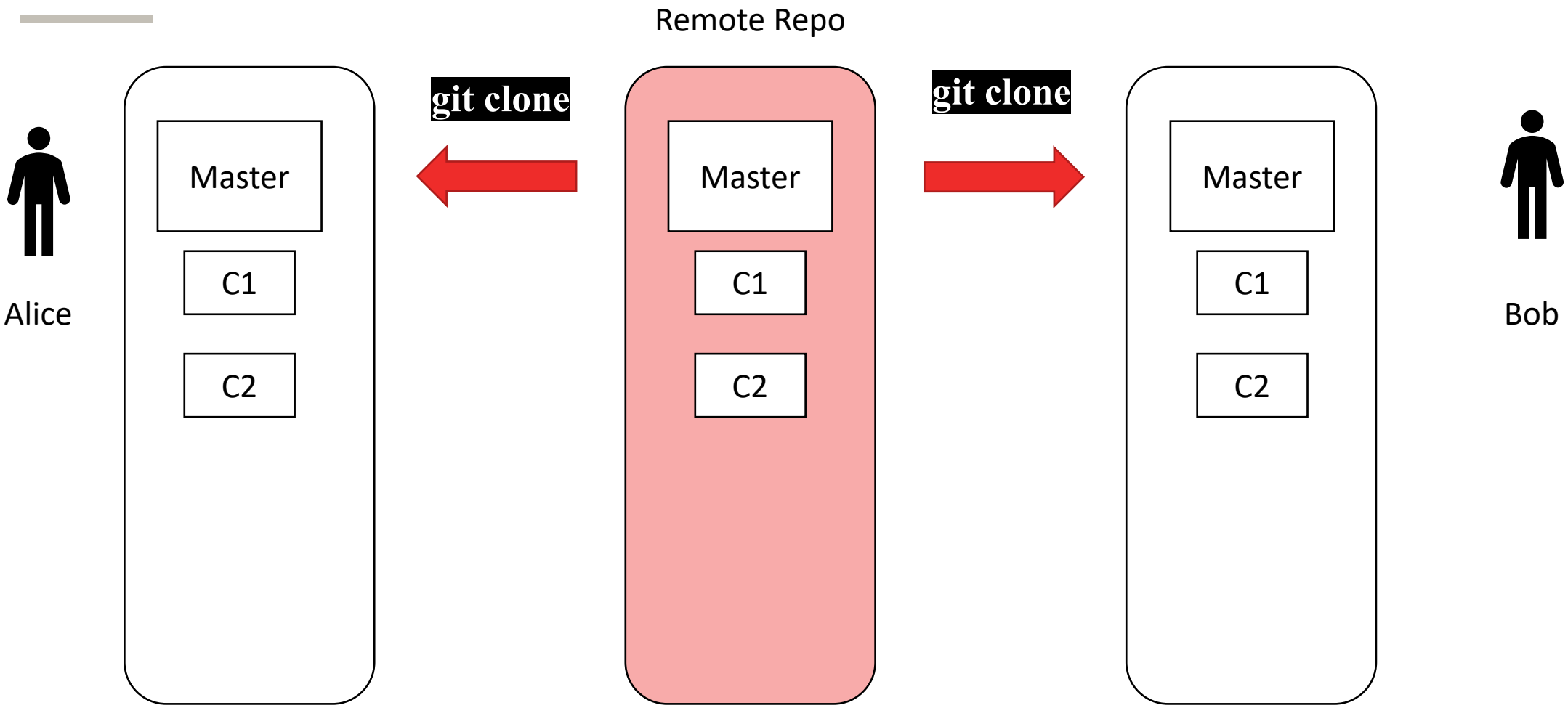
  git pull origin master

Collaborate

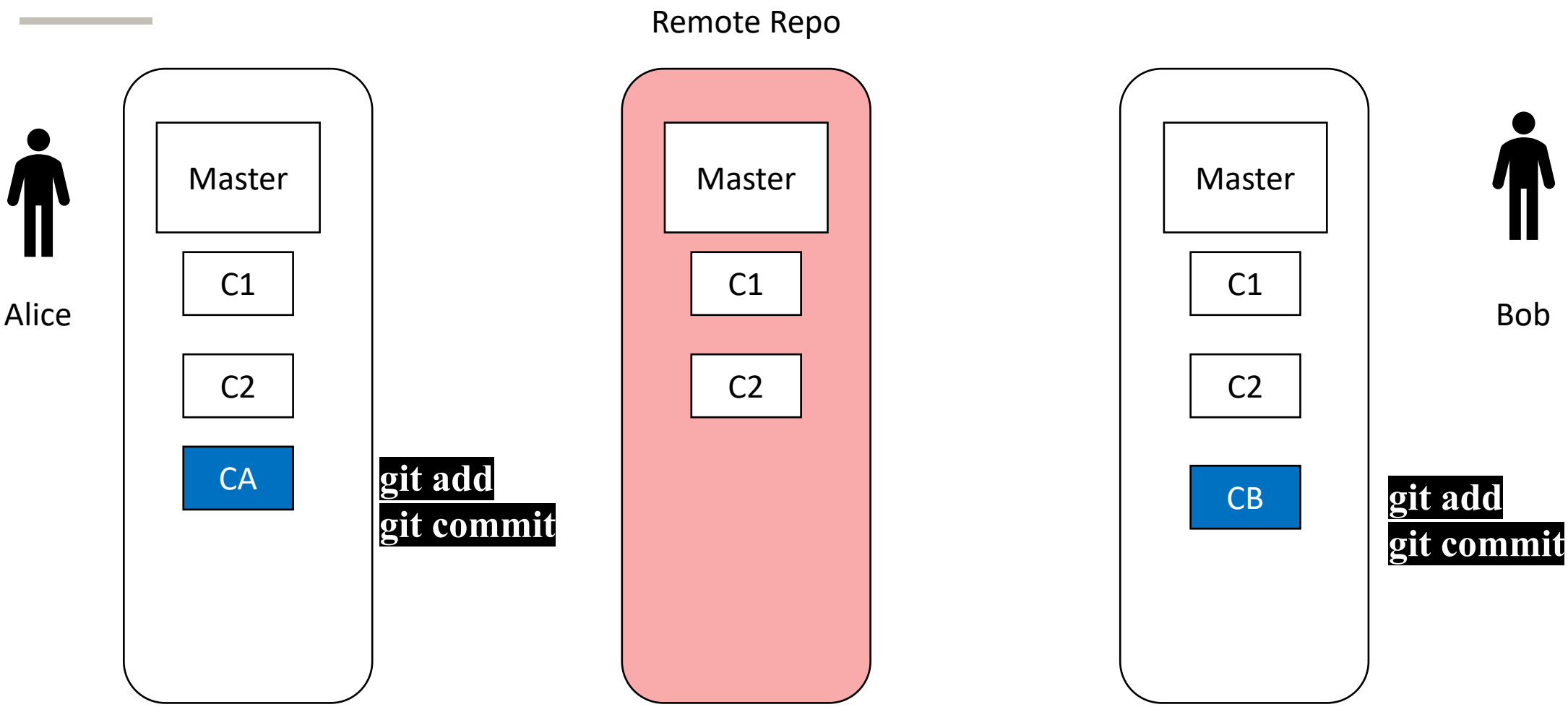
Git: Collaborate



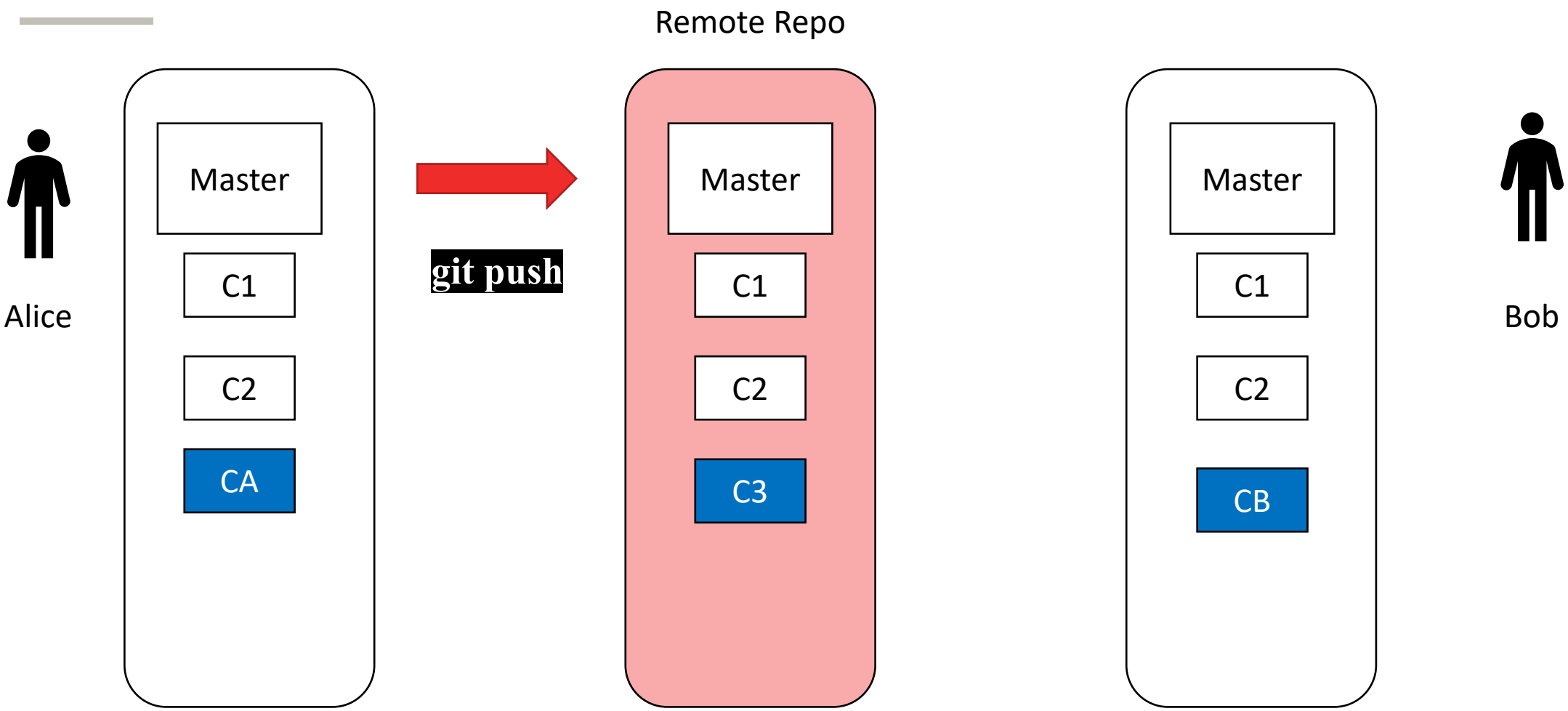
Git: Collaborate



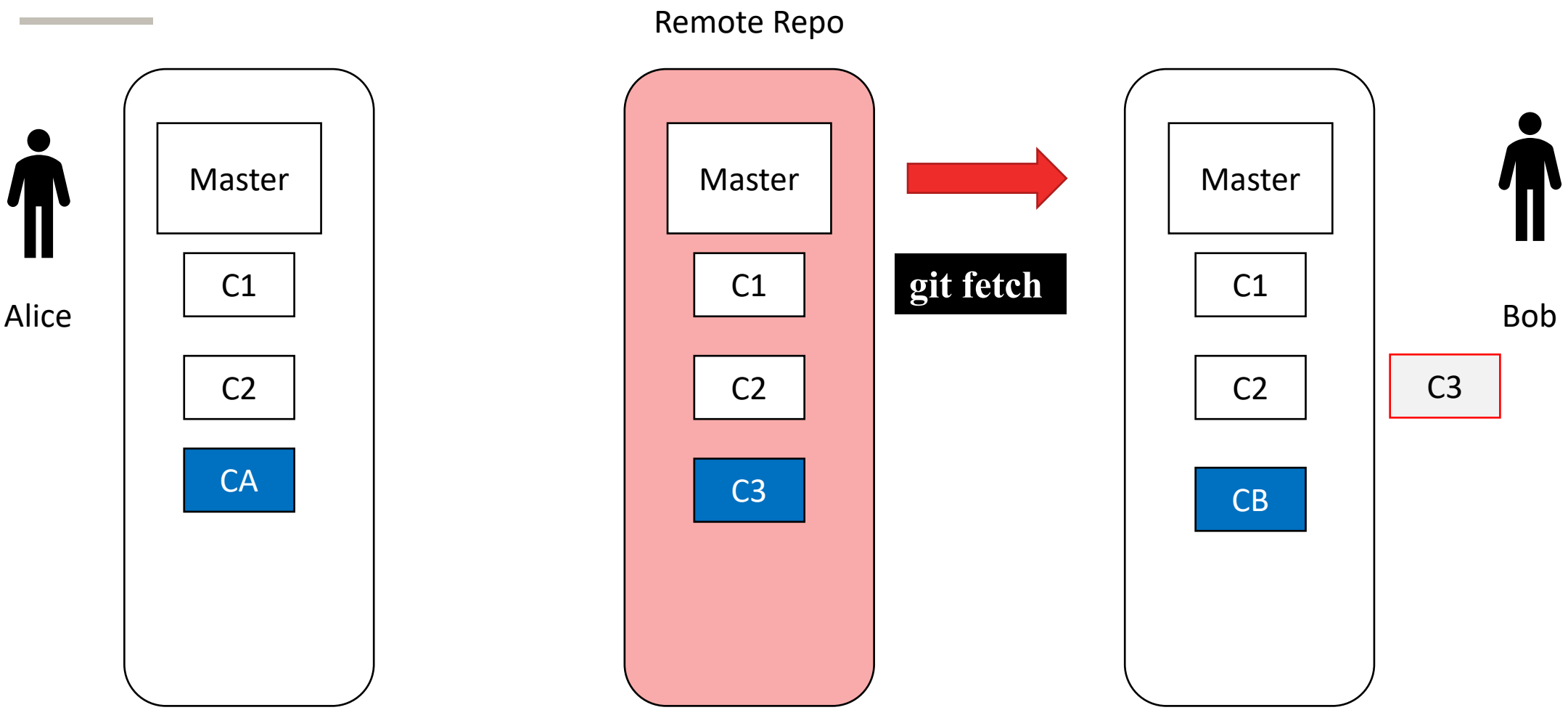
Git: Collaborate



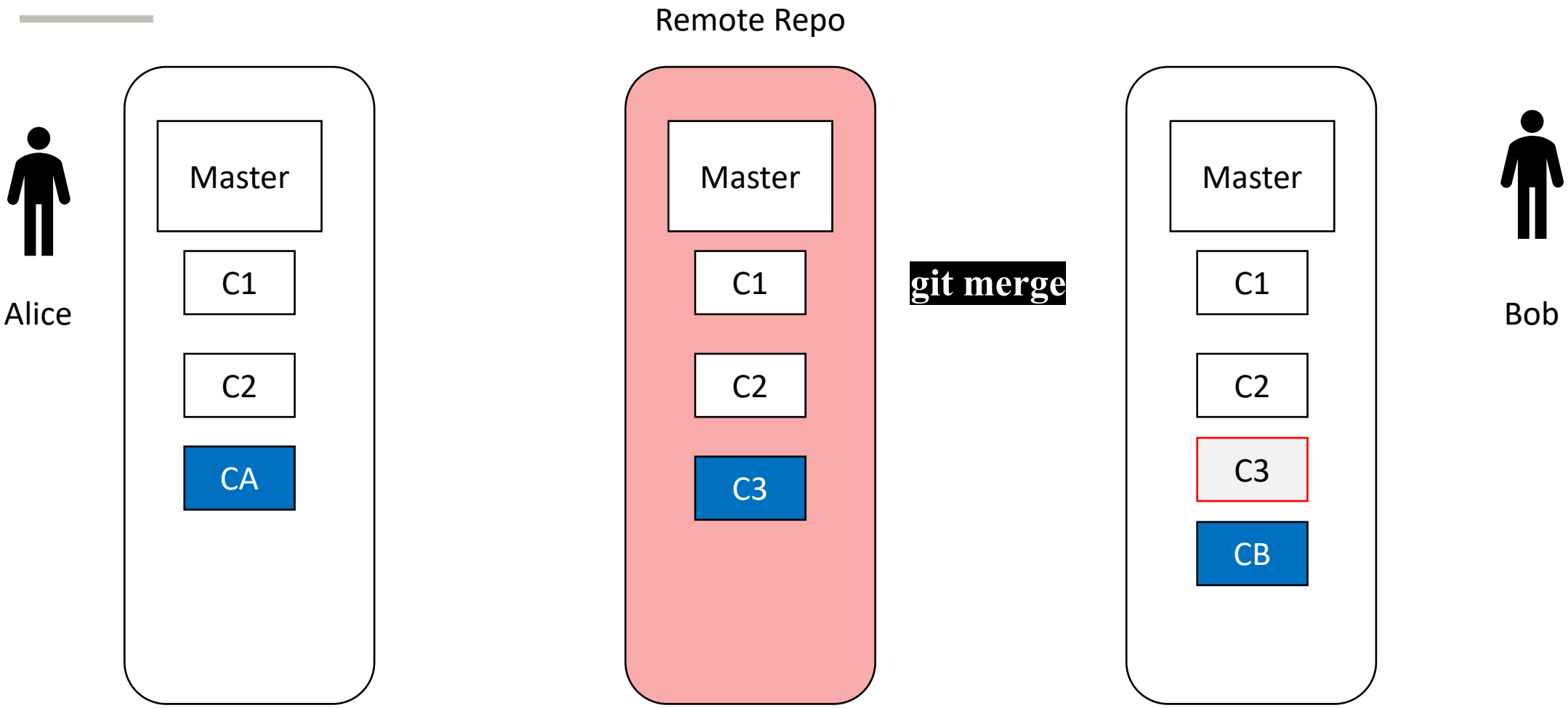
Git: Collaborate



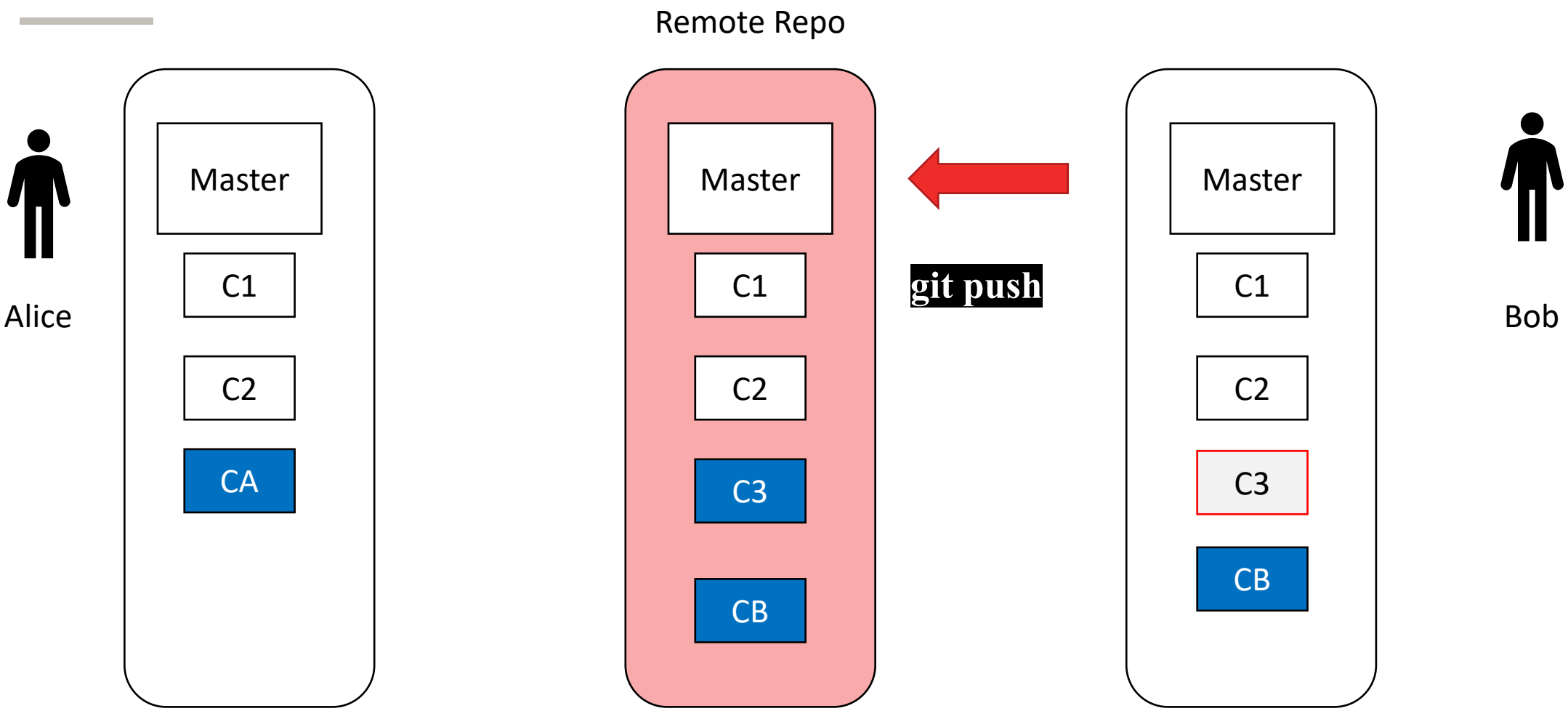
Git: Collaborate



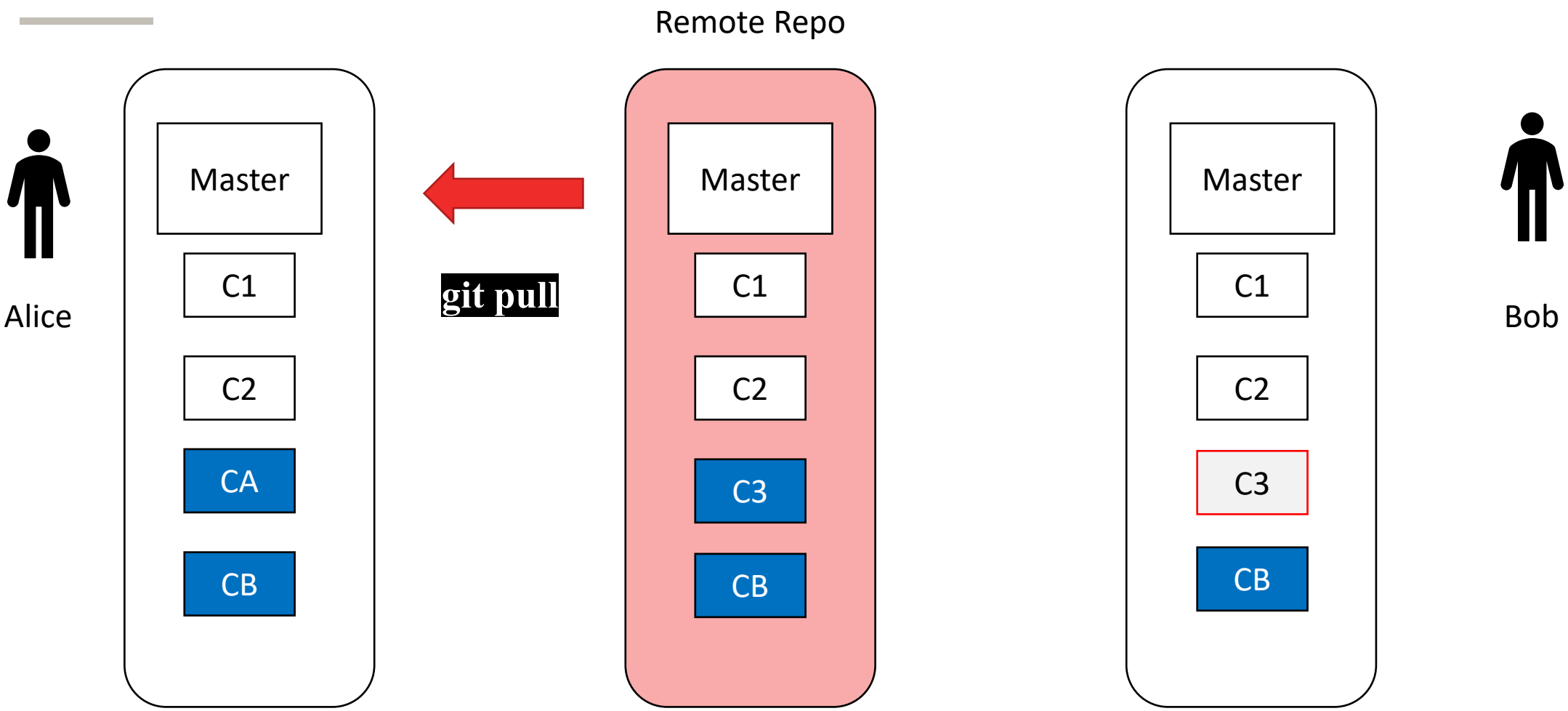
Git: Collaborate



Git: Collaborate



Git: Collaborate



Onward to … comparison.

Jonathan Hudson
jwhudson@ucalgary.ca
https://pages.cpsc.ucalgary.ca/~jwhudson/

UNIVERSITY OF CALGARY