Version Control Software

28 AUGUST, 2018

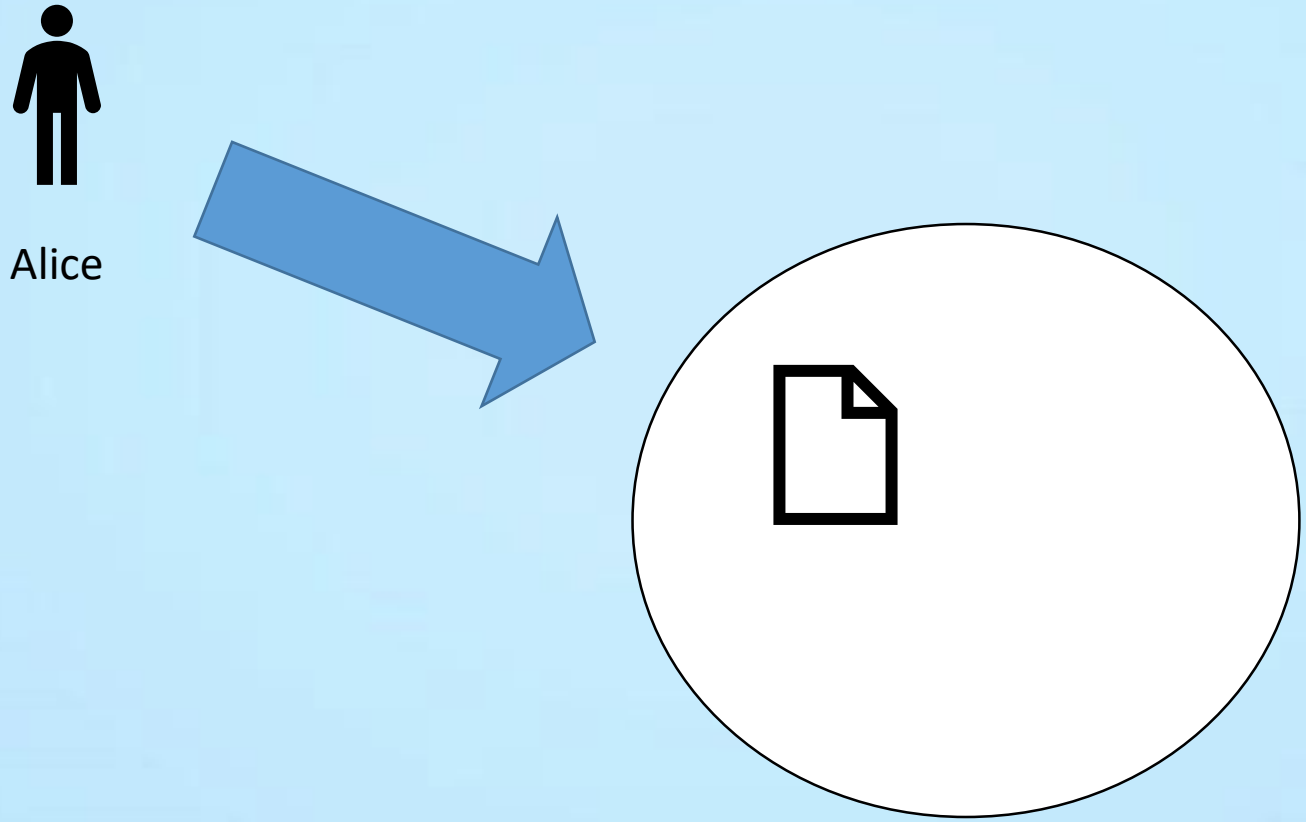
YEVHENIIA KASHCHEIEVA

AGENDA

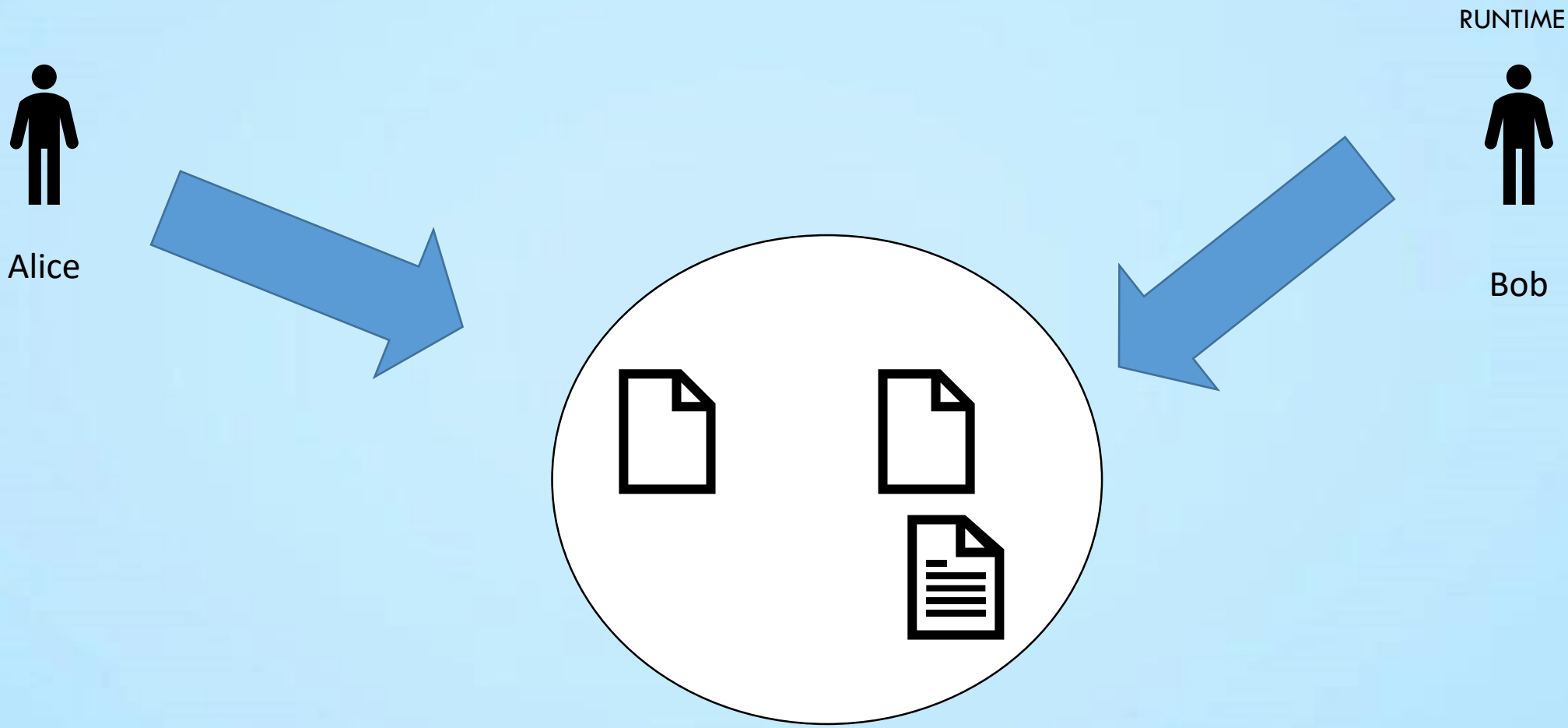
1. Introduction to .NET
2. VCS
3. GIT
4. Language-Integrated Query (LINQ)
5. Q&A

WHY DO WE NEED A VCS?

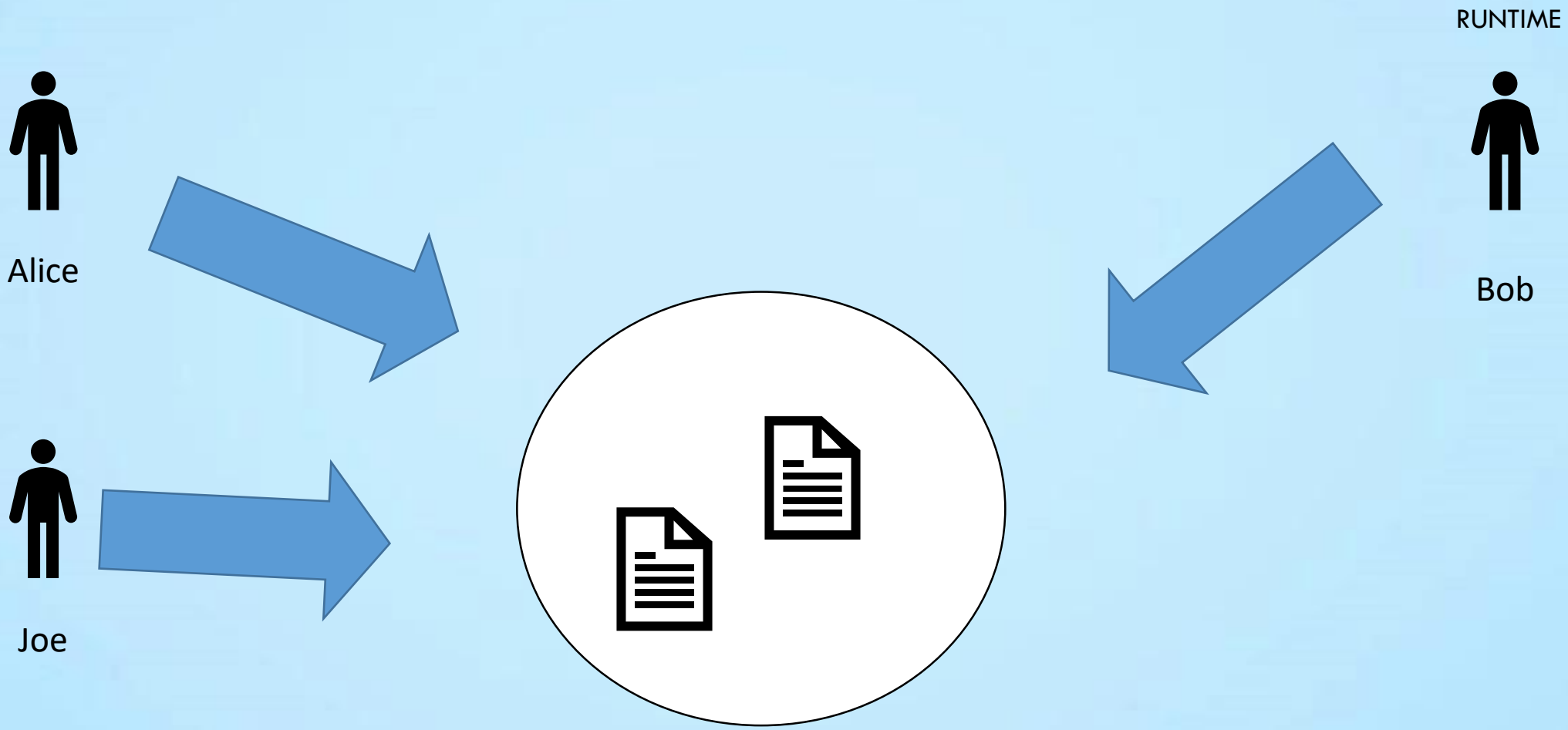
RUNTIME



WHY DO WE NEED A VCS?



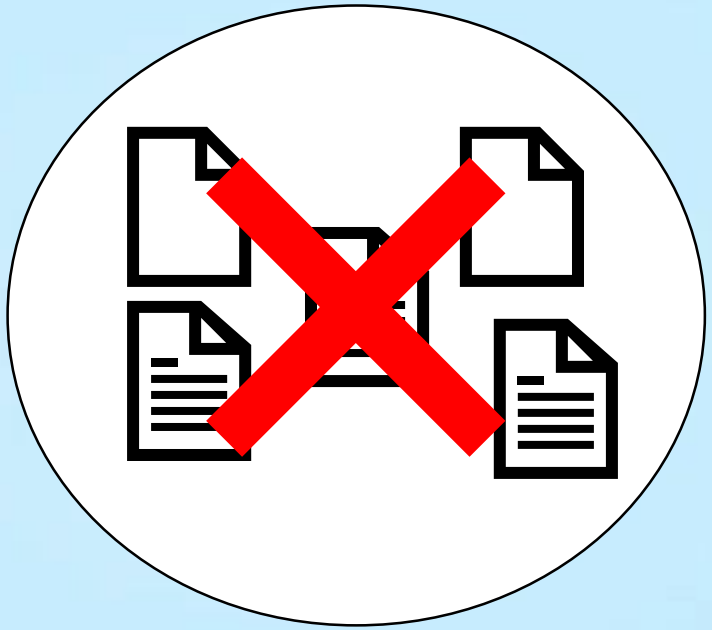
WHY DO WE NEED A VCS?



WHY DO WE NEED A VCS?

RUNTIME

Alice



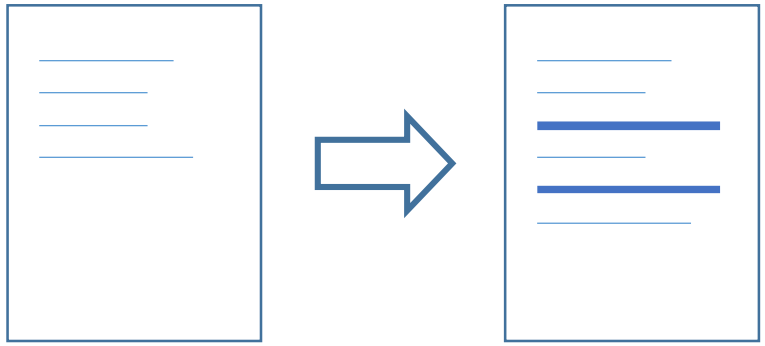
Joe

Who replaced the files? When ?

Bob

WHY DO WE NEED A VCS?

VCSs Track File Changes



Code is organized within a repository.

VCSs Tell Us:

- Who made the change?
  - So you know whom to blame
- What has changed (added, removed, moved)?
  - Changes within a file
  - Addition, removal, or moving of files/directories
- Where is the change applied?
  - Not just which file, but which version or branch
- When was the change made?
  - Timestamp
- Why was the change made?
  - Commit messages

Basically, the Five W's

BRIEF HISTORY OF VERSION CONTROL SOFTWARE

- First Generation – Local Only
  - SCCS – 1972
    - Only option for a LONG time
  - RCS – 1982
    - For comparison with SCCS, see this 1992 forum link
- Second Generation – Centralized
  - CVS – 1986
    - Basically a front end for RCS
  - SVN – 2000
    - Tried to be a successor to CVS
  - Perforce – 1995
    - Proprietary, but very popular for a long time
- Second Generation (Cont.)
  - Team Foundation Server – 2005
    - Microsoft product, proprietary
    - Good Visual Studio integration
- Third Generation – Decentralized
  - BitKeeper – 2000
  - GNU Bazaar – 2005
    - Canonical/Ubuntu
  - Mercurial – 2005
  - Git – 2005
  - Team Foundation Server – 2013


git

GIT

- Created by Linus Torvalds, creator of Linux, in 2005
  - Came out of Linux development community
  - Designed to do version control on Linux kernel

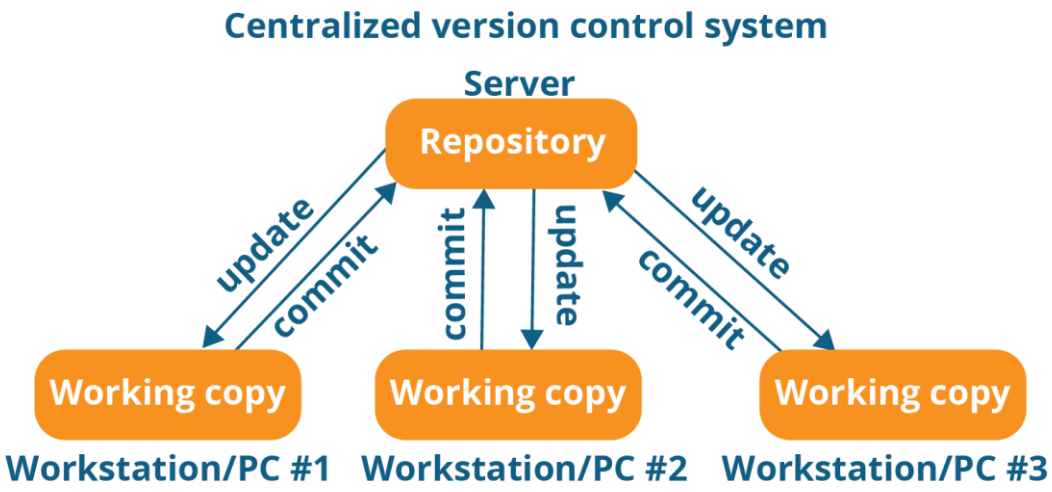
- Goals of Git:
  - Speed
  - Support for non-linear development (thousands of parallel branches)
  - Fully distributed
  - Able to handle large projects efficiently

INSTALLING/LEARNING GIT

- Git website: *http://git-scm.com/*
- Free on-line book: *http://git-scm.com/book*
  - Reference page for Git: *http://gitref.org/index.html*
  - Git tutorial: *http://schacon.github.com/git/gittutorial.html*
  - Git for Computer Scientists: *http://eagain.net/articles/git-for-computer-scientists/*
- At command line: (where verb = config, add, commit, etc.) – git help verb

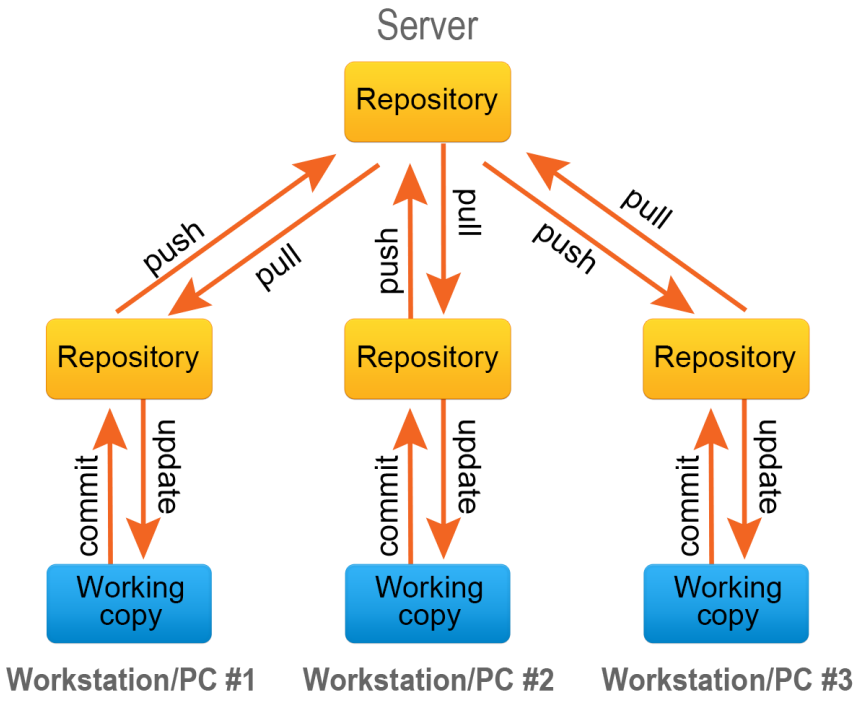
CENTRALIZED VCS

- In Subversion, CVS, Perforce, etc. A central server repository (repo) holds the "official copy" of the code
  - the server maintains the sole version history of the repo
- You make "checkouts" of it to your local copy
  - you make local modifications
  - your changes are not versioned
- When you're done, you "check in" back to the server
  - your check-in increments the repo's version

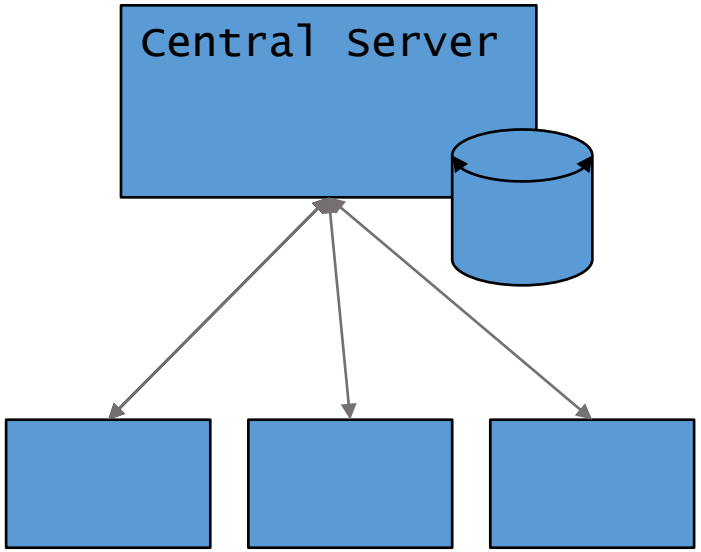


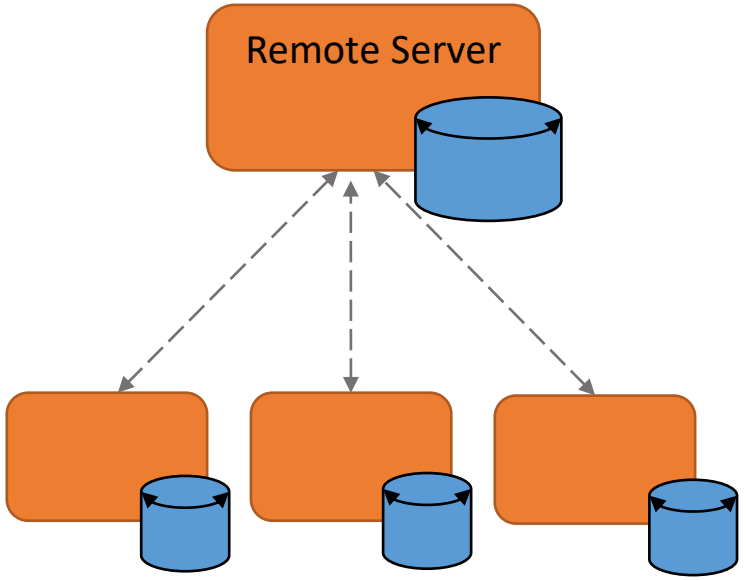
DISTRIBUTED VCS (GIT)

- In git, mercurial, etc., you don't "checkout" from a central repo
  - you "clone" it and "pull" changes from it
- Your local repo is a complete copy of everything on the remote server
  - yours is "just as good" as theirs
- Many operations are local:
  - check in/out from local repo
  - commit changes to local repo
  - local repo keeps version history
- When you're ready, you can "push" changes back to server

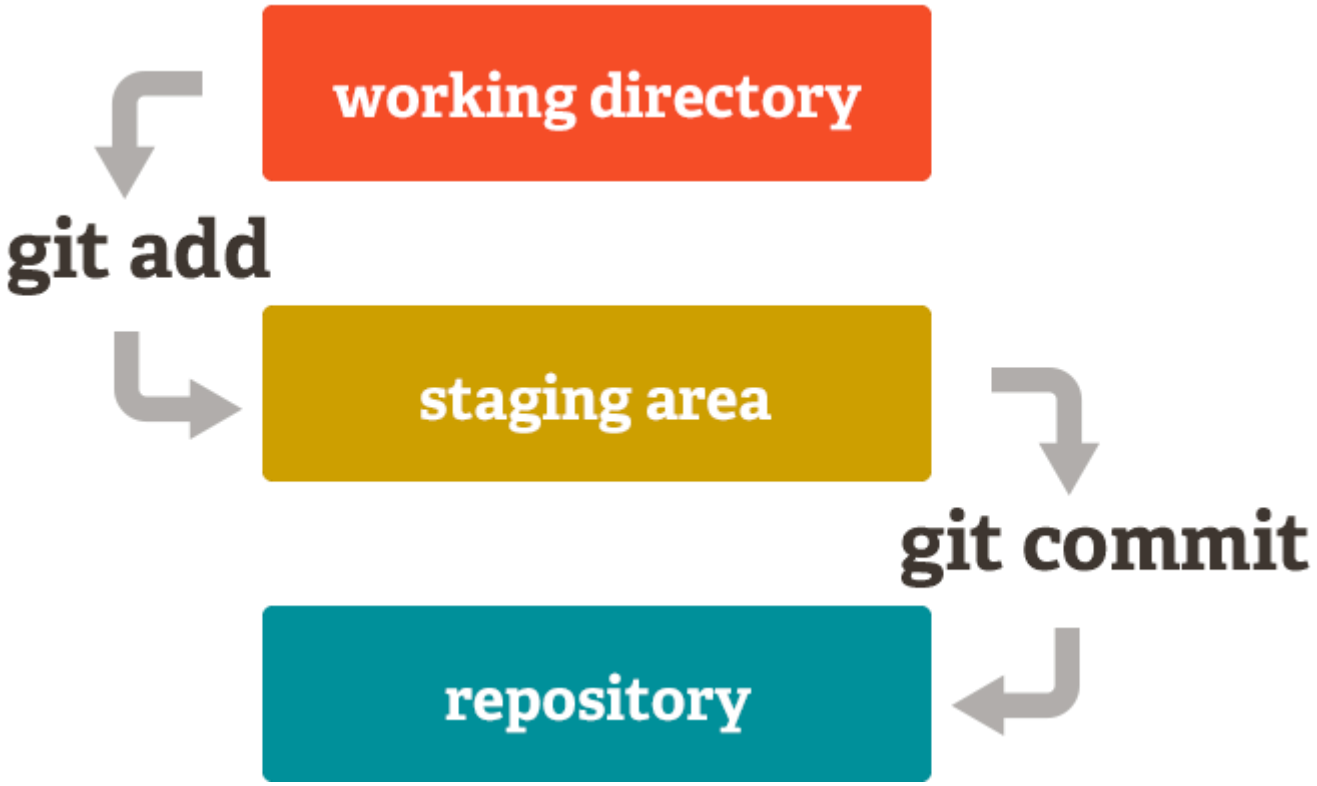


CENTRALIZED VC VS. DISTRIBUTED VC





BASIC GIT MODEL LOCALLY



TERMINOLOGY

- Branch
  - A history of successive changes to code
  - A new branch may be created at any time, from any existing commit
  - May represent versions of code
    - Version 1.x, 2.x, 3.x, etc.
  - May Represent small bugfixes/feature development
  - Branches are cheap
    - Fast switching
    - Easy to "merge" into other branches
    - Easy to create, easy to destroy
  - See this guide for best practices
- Commit
  - Set of changes to a repository's files
  - More on this later
- Tag
  - Represents a single commit
  - Often human-friendly
    - Version Numbers
- A Repository may be created by:
  - Cloning an existing one (git clone)
  - Creating a new one locally (git init)

WHAT IS A COMMIT?

- Specific snapshot within the development tree
- Collection of changes applied to a project's files
  - Text changes, File and Directory addition/removal, chmod
- Metadata about the change
- Identified by a SHA-1 Hash
  - Can be shortened to approx. 6 characters for CLI use
    - (e.g., "git show 5b16a5")
  - HEAD – most recent commit
  - ORIG_HEAD – after a merge, the previous HEAD
  - <commit>~n – the **n**th commit before <commit>
    - e.g., 5b16a5~2 or HEAD~5
  - master@{01-Jan-2018} – last commit on master branch before January 1, 2018

```
commit d6424449ced0e33af1c2b89e35ed40e2c00a29d1
Author: Nathan Grebowiec <njgreb@gmail.com>
Date:   Fri Sep 19 06:50:41 2014 -0500

    use querySelectorAll() in all cases

    since we aren't using the HTML LiveCollection returned by
    getElementsByTagName() there is no reason to not just use
    querySelectorAll() in all cases.

commit 5b16a579a2e7c052f56f867623c301eda762fab1
Author: John Heroy <johnheroy@gmail.com>
Date:   Thu Sep 18 22:01:31 2014 -0700

    Remove type=text/javascript in example <script> tags

commit 58e11bd4d899fd9943231b55424a954e6398f7a3
Author: Jose Joaquin Merino <jomerinog@gmail.com>
Date:   Thu Sep 18 21:18:44 2014 -0700

    Fix capitalisation and specificity of parameters

    Layout changes:
    - randLoadIncrement makes references to previous 'load increme
put former after the latter.
```

TERMINOLOGY

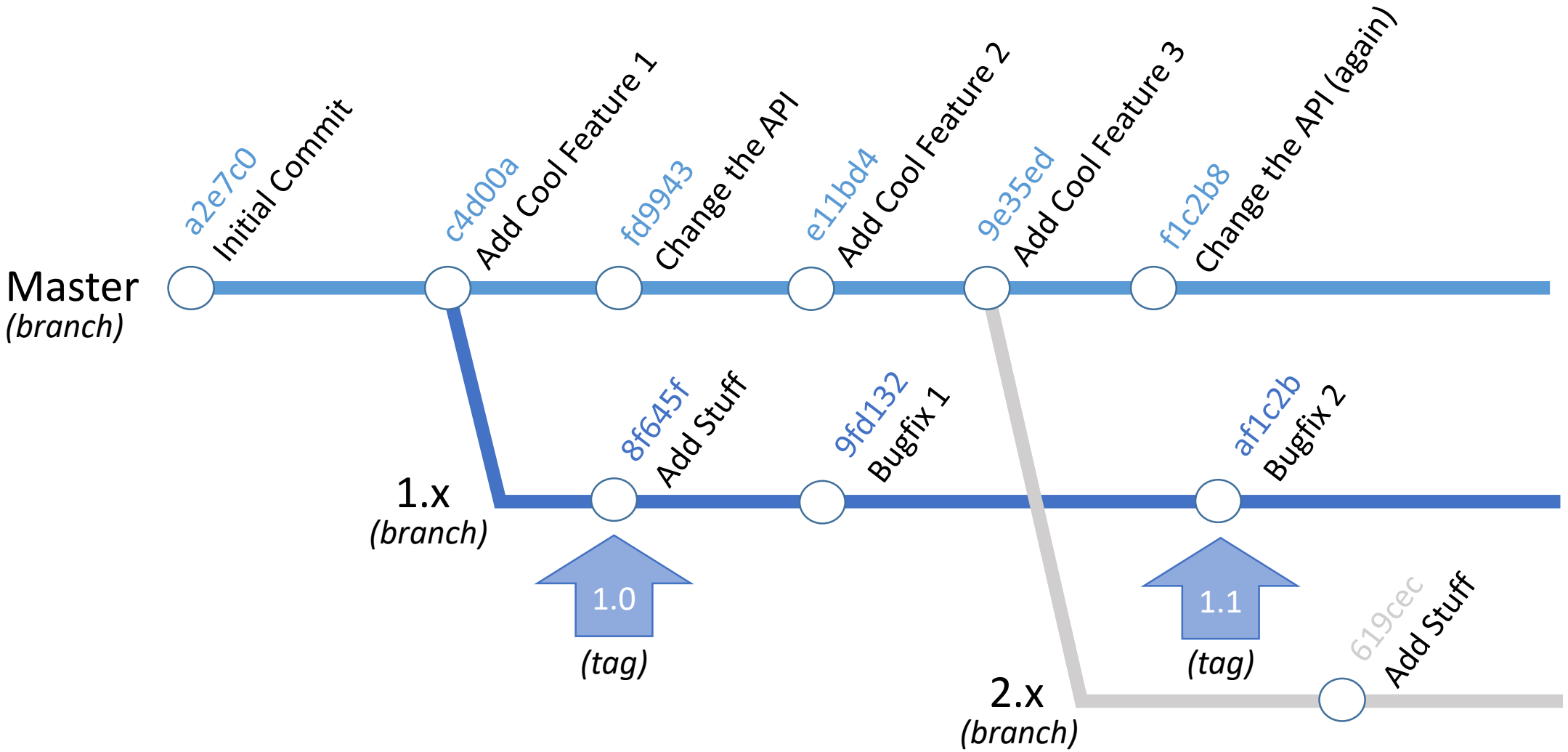
- Working Files
  - Files that are currently on your File System
- The Stage (also called the "index")
  - Staging is the first step to creating a commit
  - The stage is what you use to tell Git which changes to include in the commit
  - File changes must be "added" to the stage explicitly
  - Only changes that have been staged will be committed
- Checkout
  - Replace the current working files with those from a specific branch or commit

- Use "git diff" to see which changes exist.
- Use "git add" to tell Git that a file's changes should be added to the stage.
- Use "git status" to see the changes in your working files, as well as what changes have been staged for the commit.
- Use "git commit" to convert all staged changes into a commit.
  - git commit -m "my commit message"
  - git commit -m "my commit message" **-a**
    - Will automatically stage all files tracked by the repo & add them to the commit.
    - **Please don't do this.**

PRACTICAL GUIDELINES

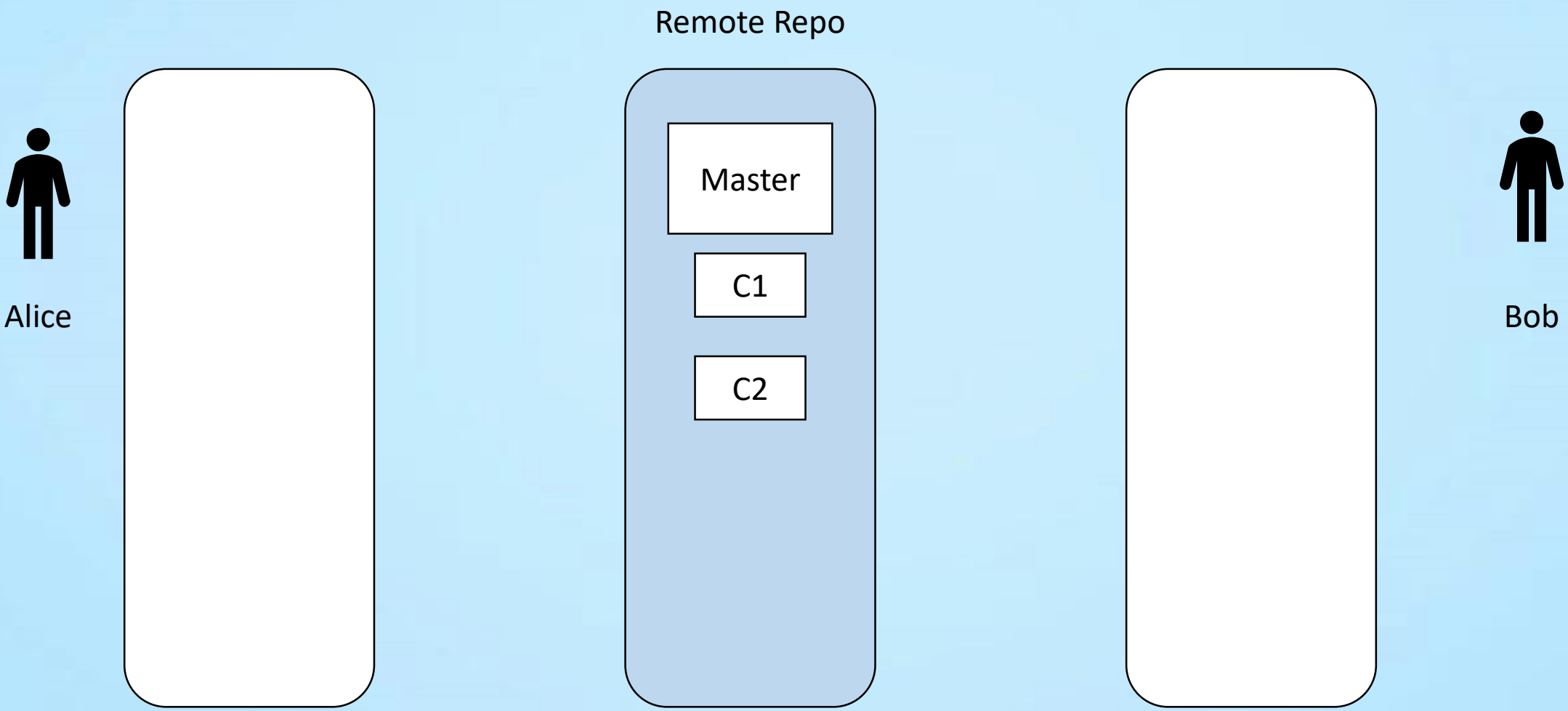
- Do's
  - Make small, incremental commits (within reason) are good. Avoid monolithic commits at all cost.
  - Use a separate branch for each new bug/feature request.
  - Write nice commit messages. Otherwise, the commit log is useless.
  - Use a .gitignore to keep cruft out of your repo.
  - Search engines are your friend. Someone else has had the same question/mistake/situation as you, and they have already asked the question online. It's probably on StackOverflow.
- Don'ts
  - Do not commit commented-out debug code.
    - It's messy. It's ugly. It's unprofessional.
  - Do not mix your commits. (e.g., Don't commit two bugfixes at the same time.)
  - Do not commit sensitive information (passwords, database dumps, etc.)
  - Do not commit whitespace differences, unless it is specifically needed.
  - Do not commit large binaries.

BRANCHES, COMMITS, AND TAGS, OH MY!

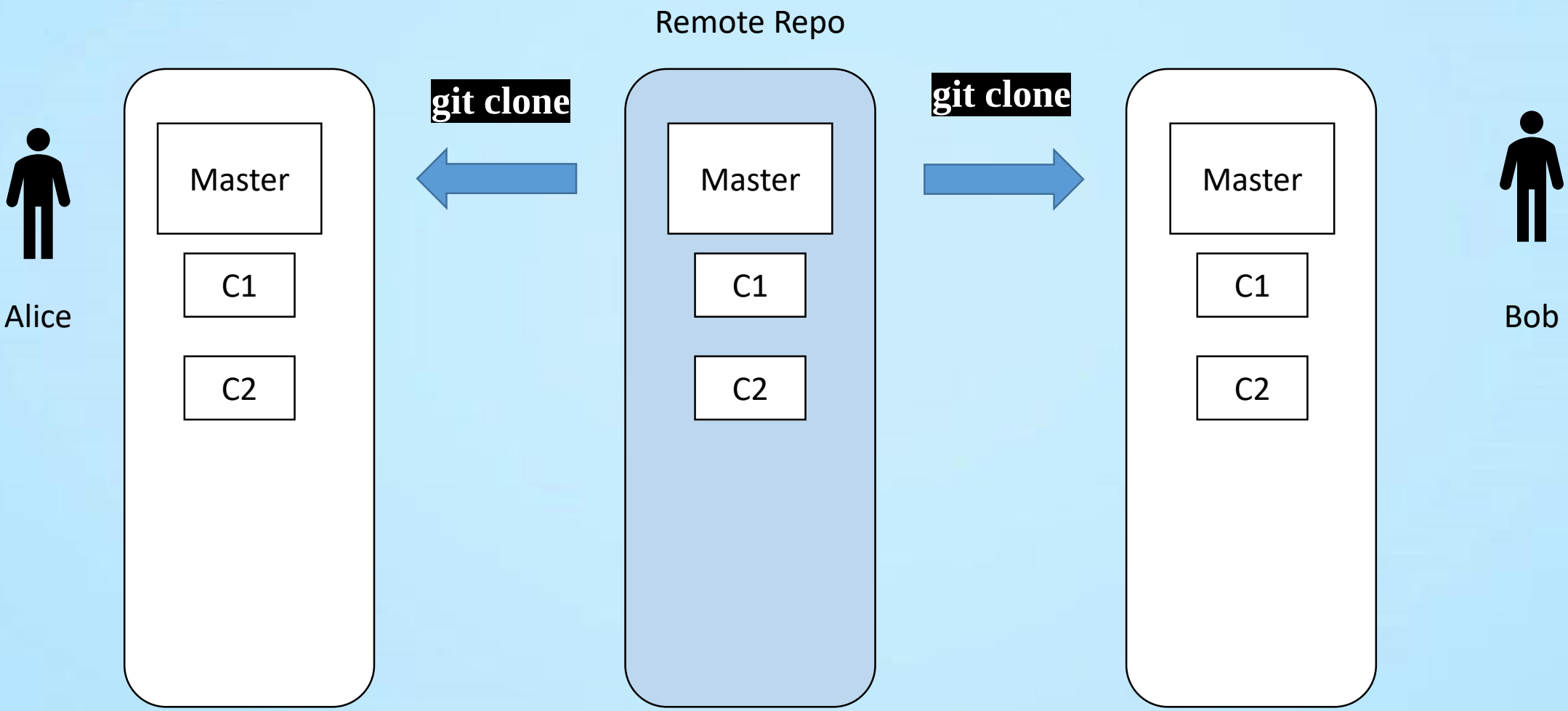



DEMO

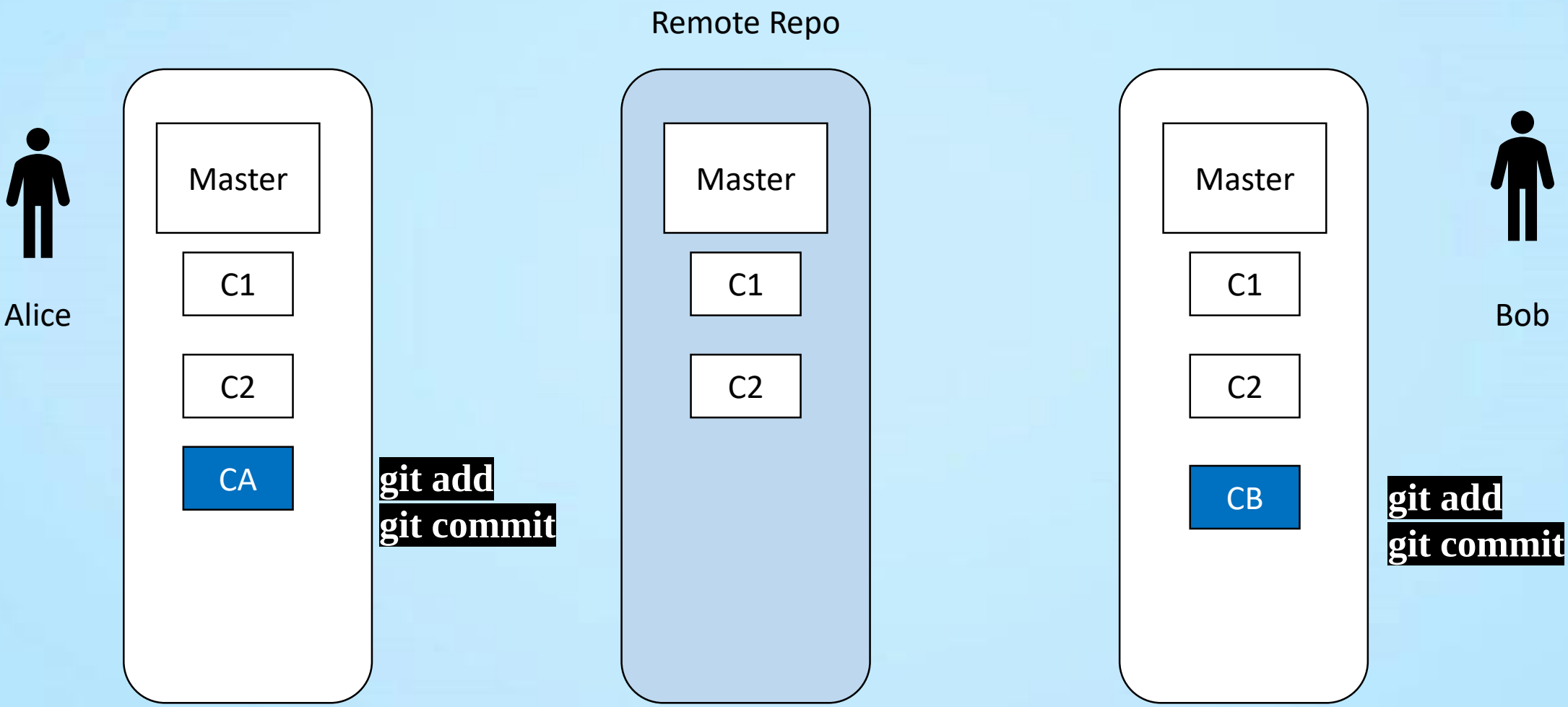
COLLABORATE



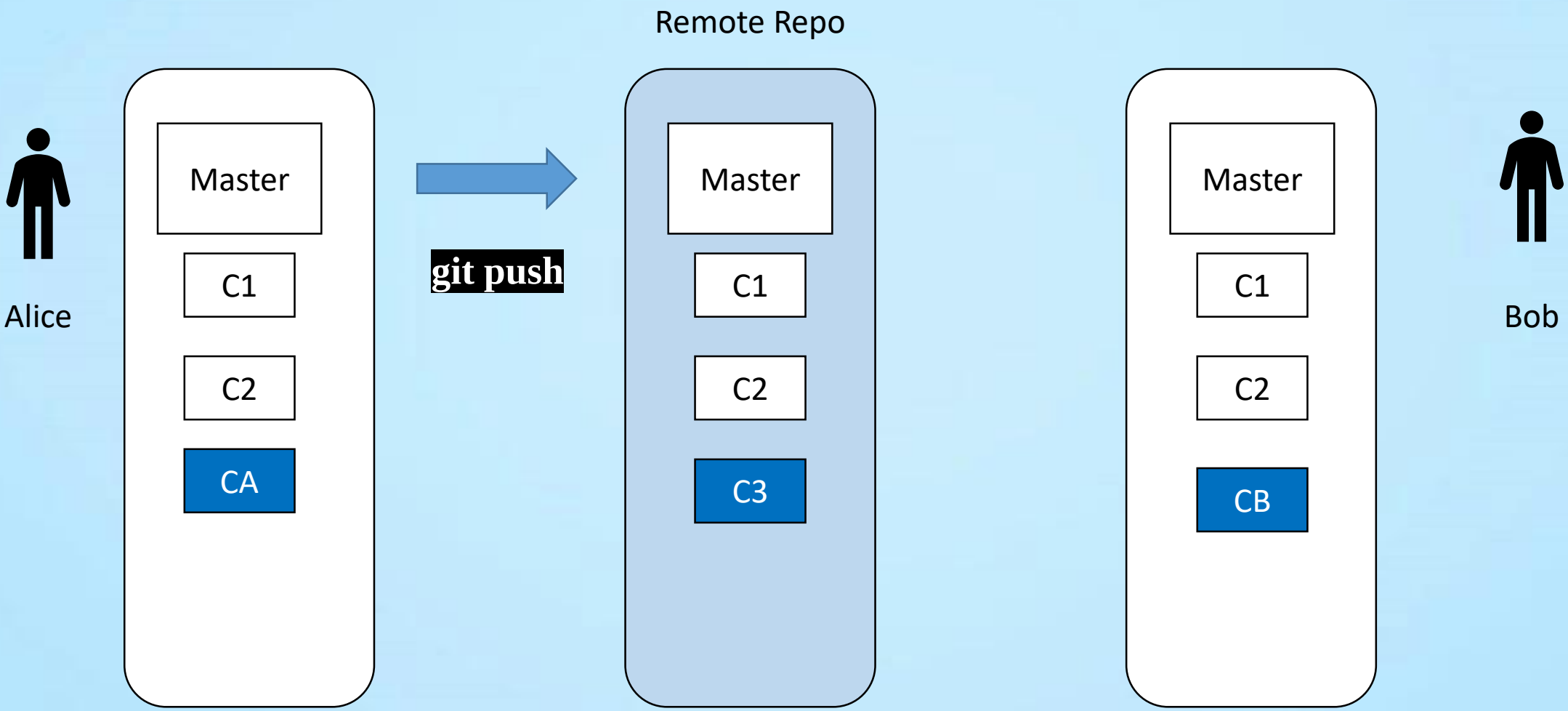
COLLABORATE



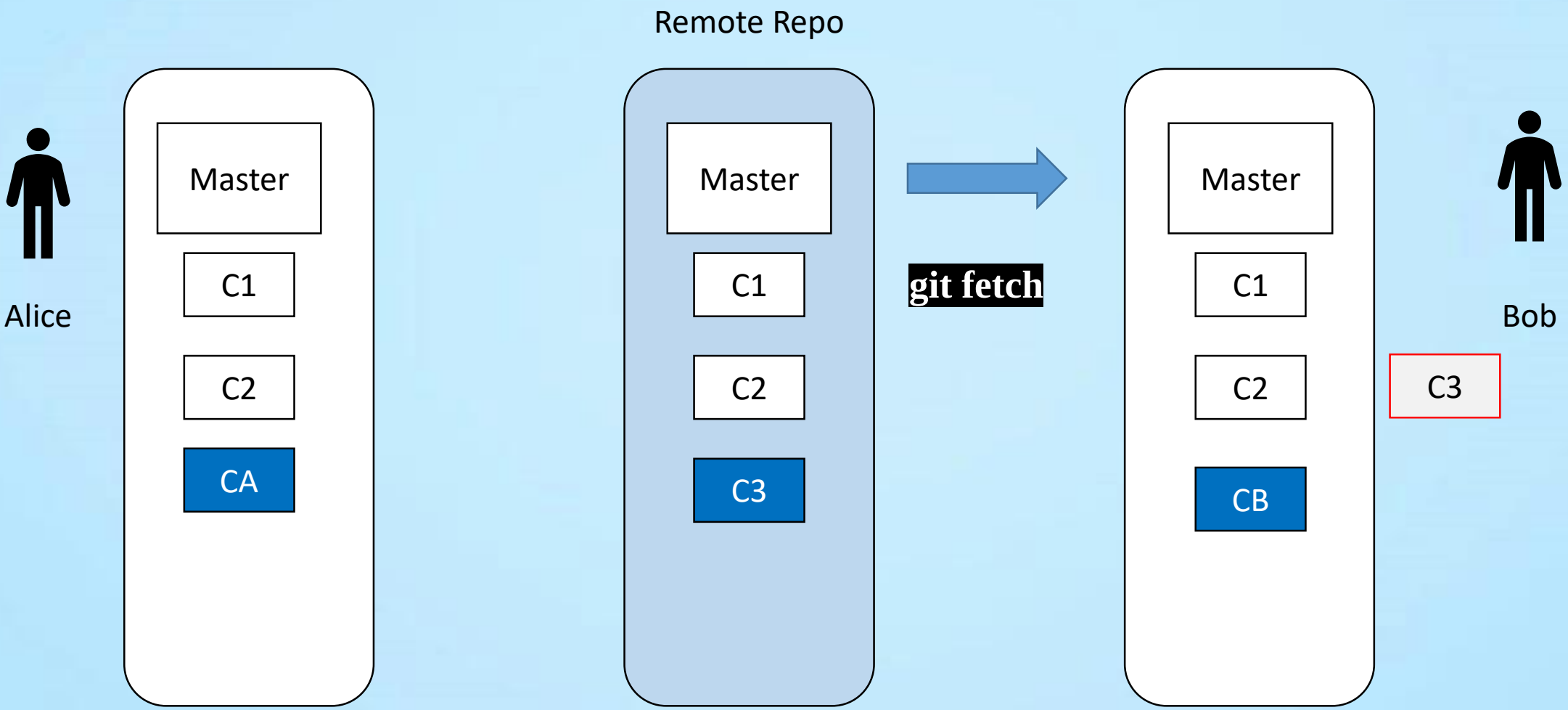
COLLABORATE



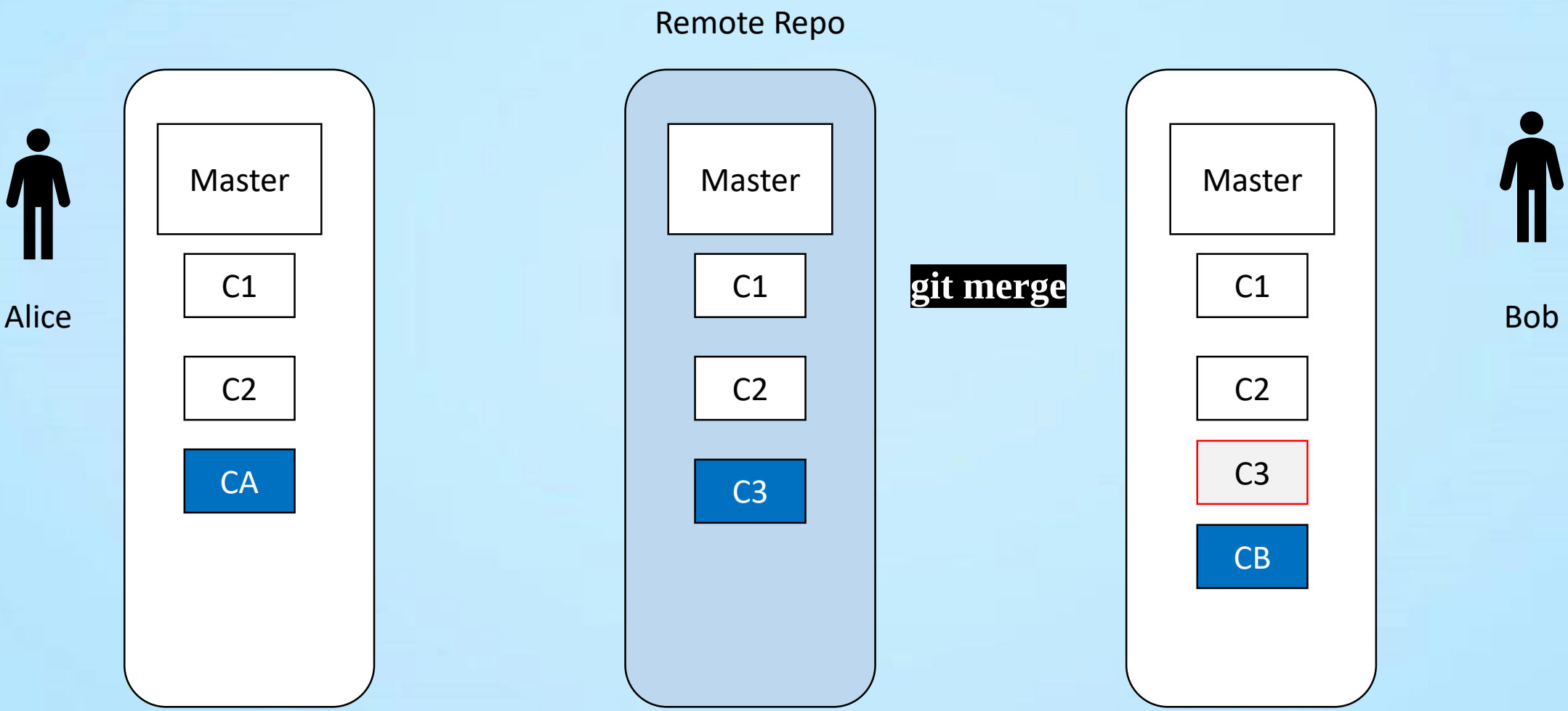
COLLABORATE



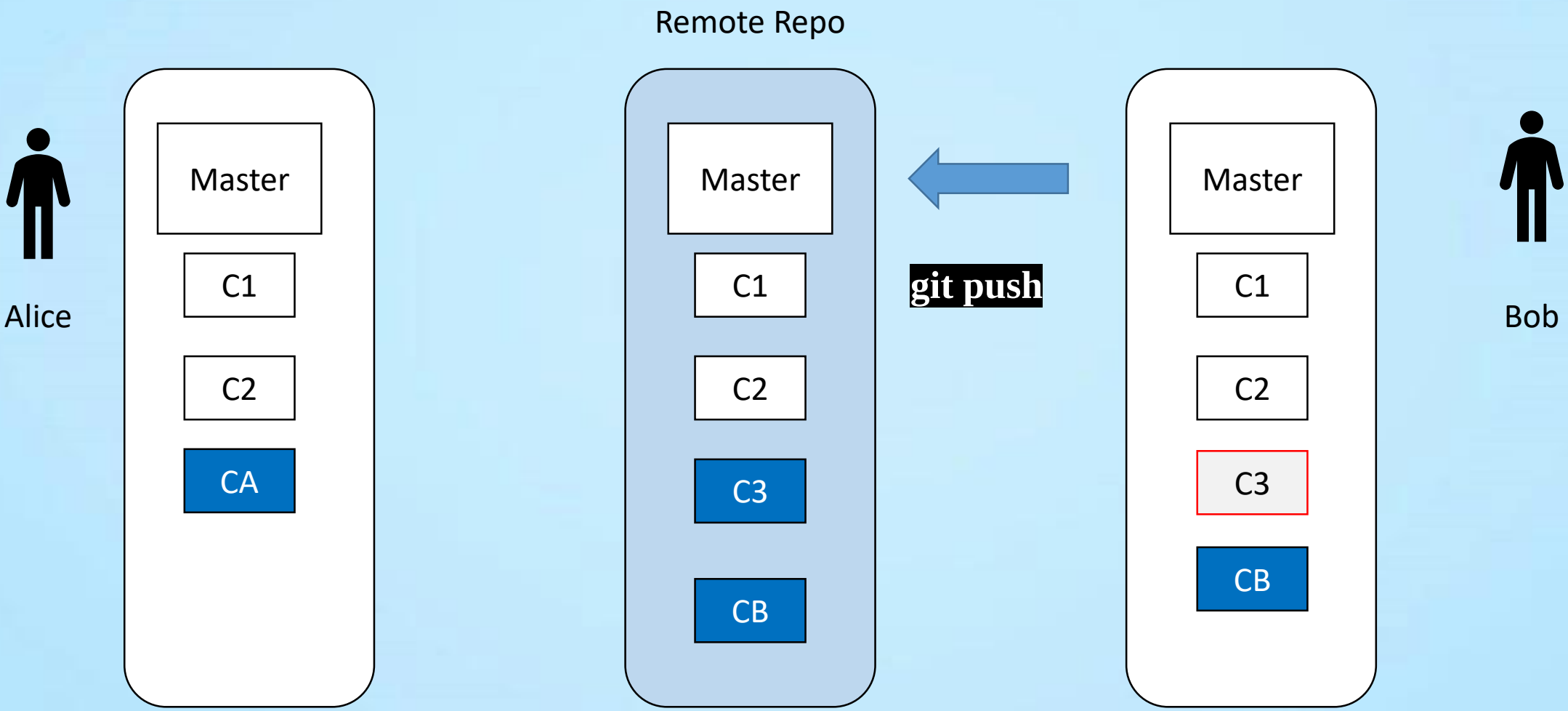
COLLABORATE



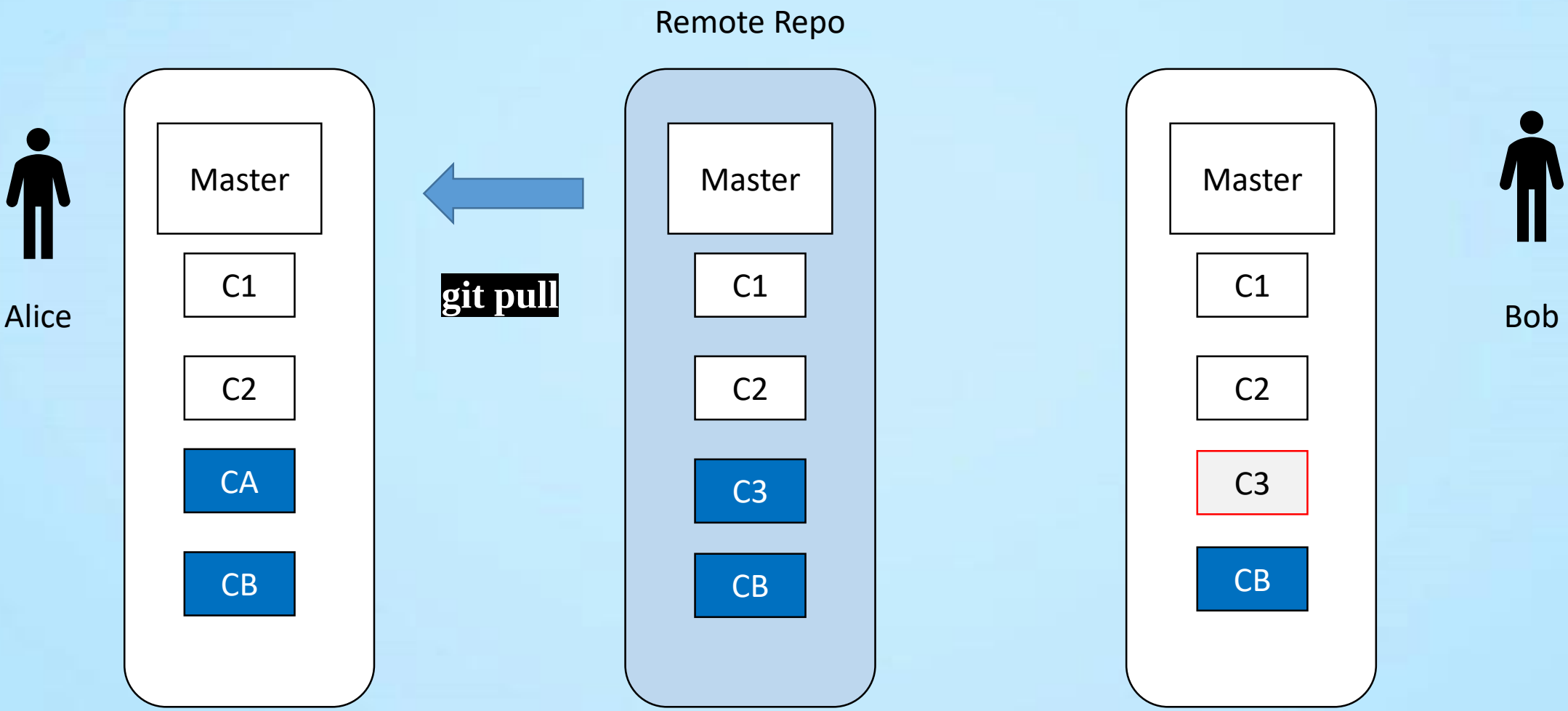
COLLABORATE



COLLABORATE



COLLABORATE



GIT COMMANDS

INITIAL GIT CONFIGURATION

- Set the name and email for Git to use when you commit:
  - `git config --global user.name "Bugs Bunny"`
  - `git config --global user.email bugs@gmail.com`
  - You can call git config –list to verify these are set.
- Set the editor that is used for writing commit messages:
  - `git config --global core.editor nano`
    - (it is vim by default)

CREATING A GIT REPO

Two common scenarios: (only do one of these)

- To create a new local Git repo in your current directory:
  - `git init`
    - This will create a .git directory in your current directory.
    - Then you can commit files in that directory into the repo.
  - `git add filename`
  - `git commit -m "commit message"`

CREATING A GIT REPO

- To clone a remote repo to your current directory:
  - `git clone url localDirectoryName`

    This will create the given local directory, containing a working copy of the files from the repo, and a .git directory (used to hold the staging area and your actual local repo) 12 Git command

ADD AND COMMIT A FILE

- The first time we ask a file to be tracked, and every time before we commit a file, we must add it to the staging area:
  - `git add Hello.java Goodbye.java`
    - Takes a snapshot of these files, adds them to the staging area.
    - In older VCS, "add" means "start tracking this file." In Git, "add" means "add to staging area" so it will be part of the next commit.
- To move staged changes into the repo, we commit:
  - `git commit –m "Fixing bug #22"`
- To undo changes on a file before you have committed it:
  - `git reset HEAD -- filename` (unstages the file)
  - `git checkout -- filename` (undoes your changes)
  - All these commands are acting on your local version of repo.

VIEWING/UNDOING CHANGES

- To view status of files in working directory and staging area:
  - `git status` or `git status -s` (short version)
- To see what is modified but unstaged:
  - `git diff`
- To see a list of staged changes:
  - `git diff --cached`
- To see a log of all changes in your local repo:
  - `git log` or `git log --oneline` (shorter version)

```
1677b2d Edited first line of readme
258efa7 Added line to readme
0e52da7 Initial commit
```

  - `git log -5` (to show only the 5 most recent updates), etc.

BRANCHING AND MERGING

- Git uses branching heavily to switch between multiple tasks.
- To create a new local branch:
  - `git branch name`
- To list all local branches: (* = current branch)
  - `git branch`
- To switch to a given local branch:
  - `git checkout branchname`
- To merge changes from a branch into the local master:
  - `git checkout master`
  - `git merge branchname`

MERGE CONFLICTS

- The conflicting file will contain <<< and >>> sections to indicate where Git was unable to resolve a conflict:

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

- Find all such sections, and edit them to the proper state (whichever of the two versions is newer / better / more correct).


THANK YOU!