Welcome!

Virtual tutorial starts at 15:00 GMT

Please leave feedback afterwards at:
www.archer.ac.uk/training/feedback/online-course-feedback.php

archer

epcc

THE UNIVERSITY OF EDINBURGH

Introduction to using Version Control with Git

ARCHER Virtual Tutorial

25/07/2018

Reusing this material

archer

epcc

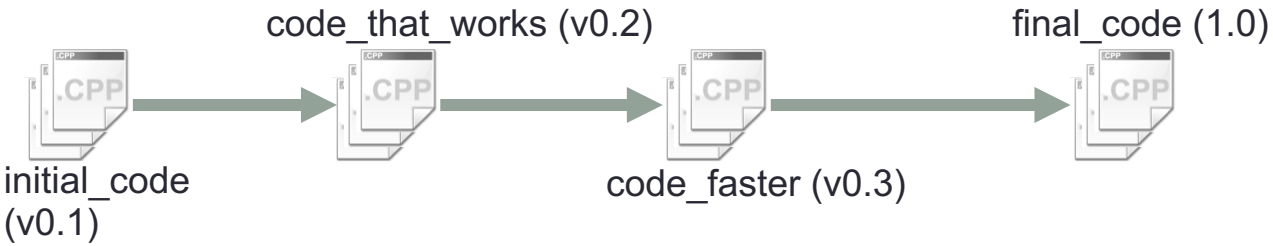
THE UNIVERSITY OF EDINBURGH

Outline

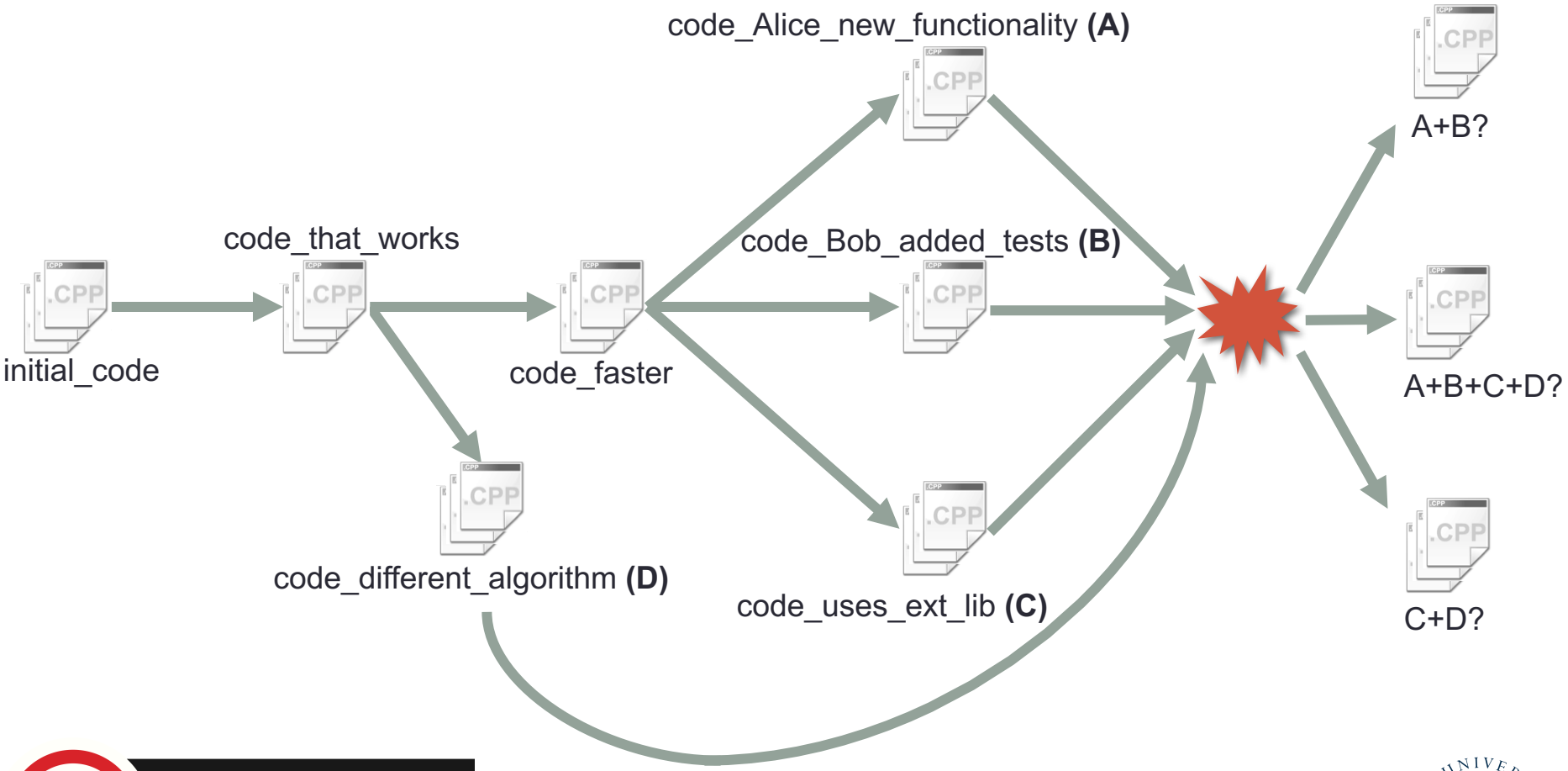
- Why version control?
- Why Git?
- Getting started with Git
  - Concepts
  - Basic workflow
  - Gotchas / subtleties to keep in mind
- Git vs other version control tools
- Hosting & additional features (GitHub & GitLab)
  - Forking & pull requests, issue tracking, tests
- Distributed workflows

Why version control?

Why version control?



Why version control?



Why version control?

Version control tools:

- Provide a framework to record meaningful information about versions in a systematic way

- Allow you to revisit / recover past versions (history!)
  - snapshot of a set of files captures interdependencies between files

- Automate the tracking of changes between versions

Why version control?

Version control tools:

- Facilitate reproducible research
  - Mark results with the exact version of code used to generate them, and make versions public

- Allow for easy duplication and synchronisation of files across multiple locations
  - Avoid error-prone manual transferring of file versions
  - Work on different machines
  - Help maintain backups of your data

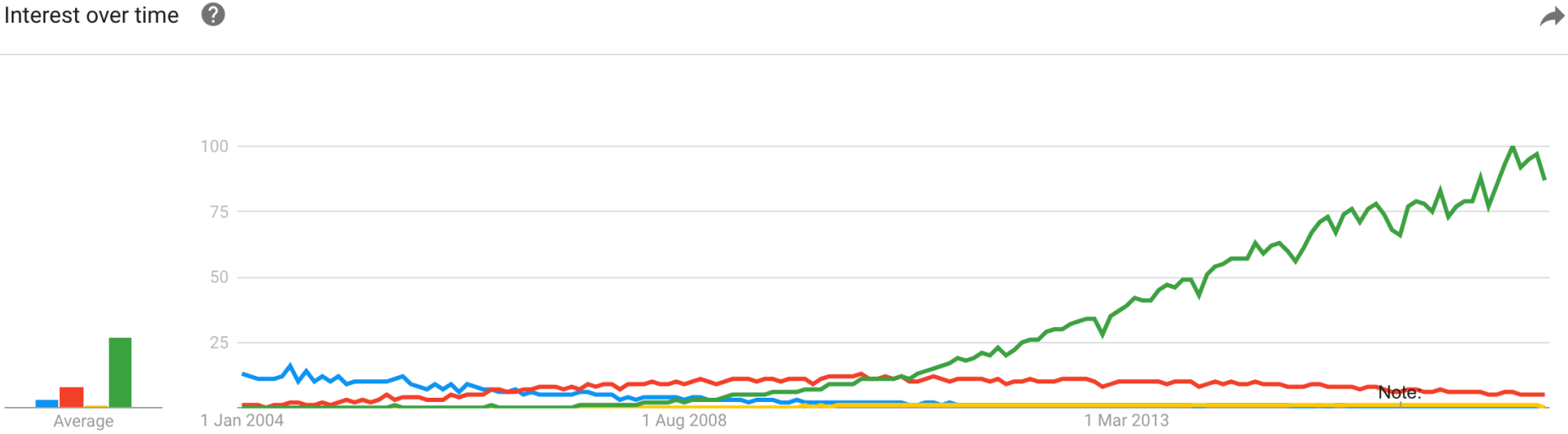
Why version control?

Version control tools:

- Enable collaborative work on one or more of the same files at the same time
  - identify and combine contributions from different authors
- Are crucial for sustainable software development work

Why Git?

Why Git?



- Source: Google Trends
- cvs svn mercurial git

Why Git?

- Git dominates (open source) software development, and software for computational science used in academia
- Many existing projects use Git – you are very likely to encounter it
- Most popular online hosting and collaborative tools for software development target Git
  - GitHub, GitLab

Getting started with Git

Git concepts

Git concepts

- Repository (“repo”)
  - A complete archive containing the full history of all recorded versions of files (i.e. all snapshots taken), stored as the differences (“deltas”) between successive versions
  - Stored on your local machine (unlike Subversion/svn and cvs repositories)
  - Created using “git init”, or downloaded using “git clone”
- Commit (*verb*) (“git commit”)
  - To record a snapshot of the current state of one or more files in the repository
- Commit (*noun)*
  - A snapshot of the kind described above

Git concepts

- Branch *(noun)*
  - A series of successive snapshots (“commits”) of one or more files stored in the repository
  - All repositories contain at least one branch (the master branch)
- Branch *(verb)* (“git branch”)
  - Creation of a new branch as an offshoot from the current branch
  - New branch is initialised with copies of the current versions of all files in the current branch
  - Typically done in order to initiate development of a new feature
  - Changes isolated in new branch – no other branches affected
- Merge (“git merge”)
  - Combine two branches by merging one into the other
  - Often done to integrate new feature developed on separate branch into the main codebase (“master” or “dev” branch) following successful testing

Git concepts

- Working tree / working directory
  - Your local view of the files in the repository
  - Shell only shows files in one branch at any given time (GUI tools can expand this view)
  - Differs from the latest commit on the relevant branch in repository if you've changed, added or removed files
- Check out ("git checkout")
  - Switch the working directory view from one branch to another
- Log ("git log")
  - A human-readable record of which files in the repository were changed when, including (hopefully) meaningful comments by the author who made the changes

Git concepts – interacting with remote repos

- Push (“git push”)
  - Upload the commits stored in your local repository to a corresponding remote repository
  - Default behaviour is to push the current branch to corresponding remote “upstream” branch
  - Will not work if there are conflicts, will then need to pull first (see below) and push once conflicts resolved

- Pull (“git pull”)
  - Download commits from a remote repository and merge them into your local repository
  - Default behaviour is to pull from the remote branch corresponding to the current branch, and merge into the current branch
  - Actually consists of two sub-commands: “git fetch” and “git merge”
  - Any conflicts become apparent and will need to be resolved

Getting started with Git

Basic workflows

Basic workflows – private repo

To create and maintain your own local repository:

- git init
- git add file1 file2
- git commit –m “Added initial versions of files”

- Modify/add new files and repeat:
- git add modified_files newfiles
- git commit –m “Description of changes”

- Etc.

Basic workflows – shared repo

To develop a new feature on a shared software project as a member of the development team:

- Clone project repository locally (assume repository lives on GitHub or GitLab):
  - git clone repository_URL
- Create a new branch for your feature:
  - git branch myFeature
- Switch the local working directory view to that branch:
  - git checkout myFeature
- Modify/add files and commit these
- Push changes to the original remote (origin) repo:
  - git push

Basic workflows – shared repo

Once feature has been developed and tested successfully, want to integrate it into the main code for general release:

- Need to perform the merge locally
- First need to make sure our destination branch (e.g. “master”) is up to date:
  - git checkout master
  - git pull
- Then merge our feature branch into master:
  - git merge my_feature
- After resolving any conflicts, push the result to share with collaborators / the world:
  - git push

- Note: it is risky to merge directly into master branch, as conflict resolution may get messy. Most development workflows merge into intermediary development branches instead

Getting started with Git

Gotchas / subtleties to keep in mind

Gotchas / subtleties to keep in mind

- Git has the notion of the “index”
  - A staging area (cache) listing files that are due to be committed
  - Existing tracked files that are modified need to be added explicitly to the index every time they are modified to make sure they are included in a commit
  - Show using “git status”
- When in doubt, orient yourself with `git status`
  - This shows the current branch, the index, and how the current state of the working tree compares to the repository
- When you create a new branch locally that you intend to push to a remote repository, you will need to use the --set-upstream or --track option (see the relevant man page, `git branch --help` or `git push --help`)
- HEAD
  - This is a reference to the most recent commit on whatever branch is currently checked out

Git vs other version control tools

Git vs other version control tools

- Git is an example of distributed / decentralised version control (same as Mercurial) – everybody has a local repository
- Older tools (CVS, Subversion/svn) are centralised – there is only one copy of the repository, stored on a server – everybody has local working copies
- Git has a much greater emphasis on / efficiency using branches → more complex workflows

Distributed version control

Each user has their own repository copy stored locally

Central server is optional (in practice often useful)

Alice

Carol

Bob

Dave

archer

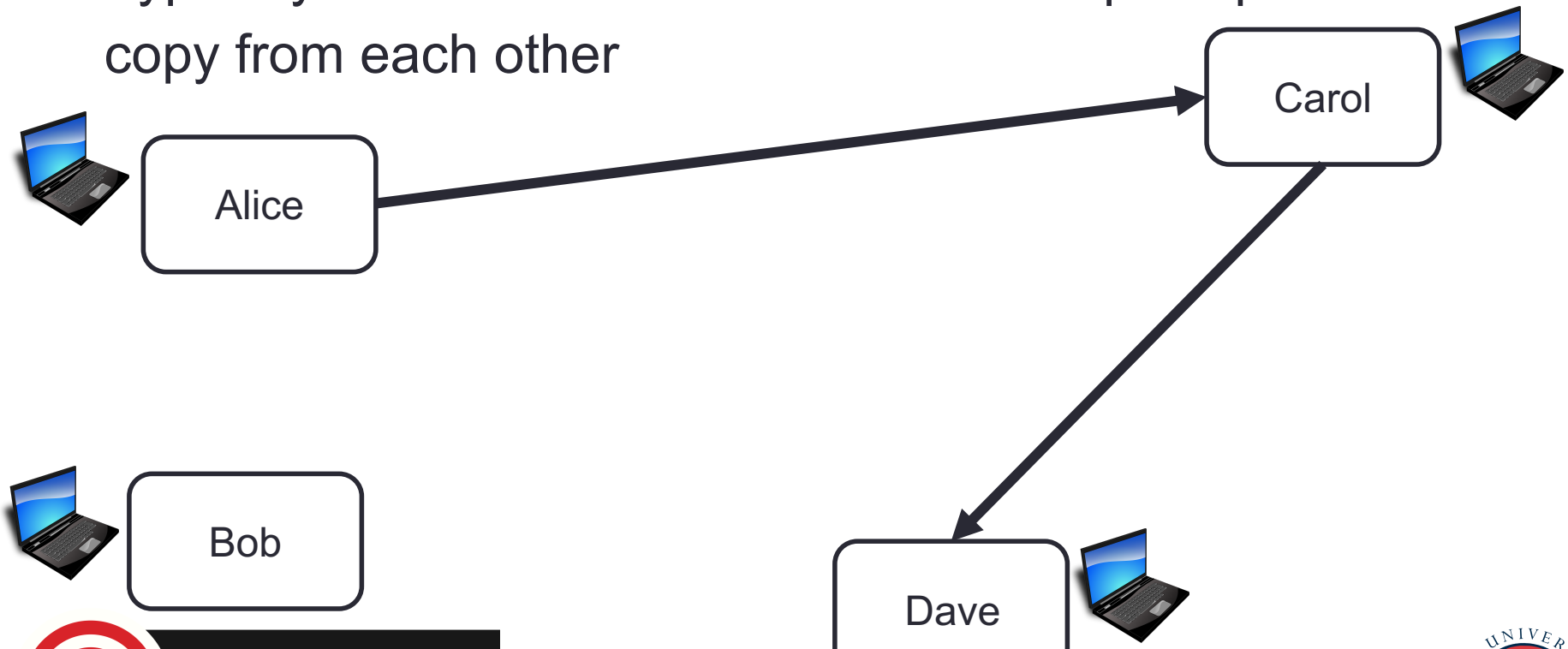
epcc

THE UNIVERSITY OF EDINBURGH

Distributed version control

New users clone, i.e. copy, an existing repository

- Typically from a central server but can in principle copy from each other



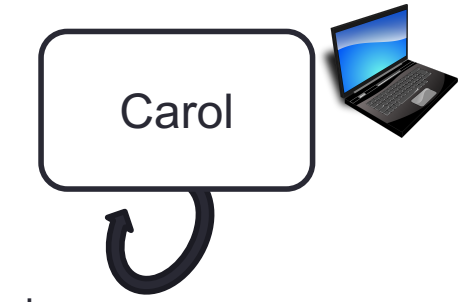
archer

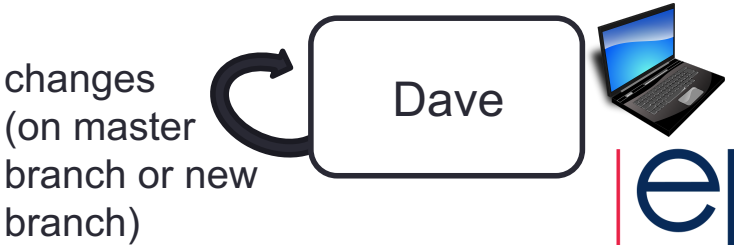
epcc

THE UNIVERSITY OF EDINBURGH

Distributed version control

Users make changes in their working copy and commit this to their local repository ➔ repositories diverge



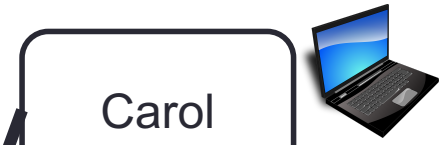


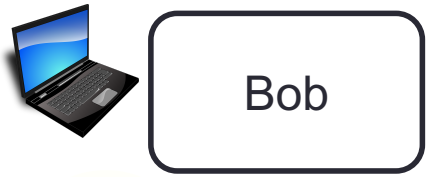
epcc

THE UNIVERSITY OF EDINBURGH

Distributed version control

To combine content from different repositories someone has to fetch other people's changes into their working copy and perform merges there





Dave pulls from Carol, merges his and Carol's changes

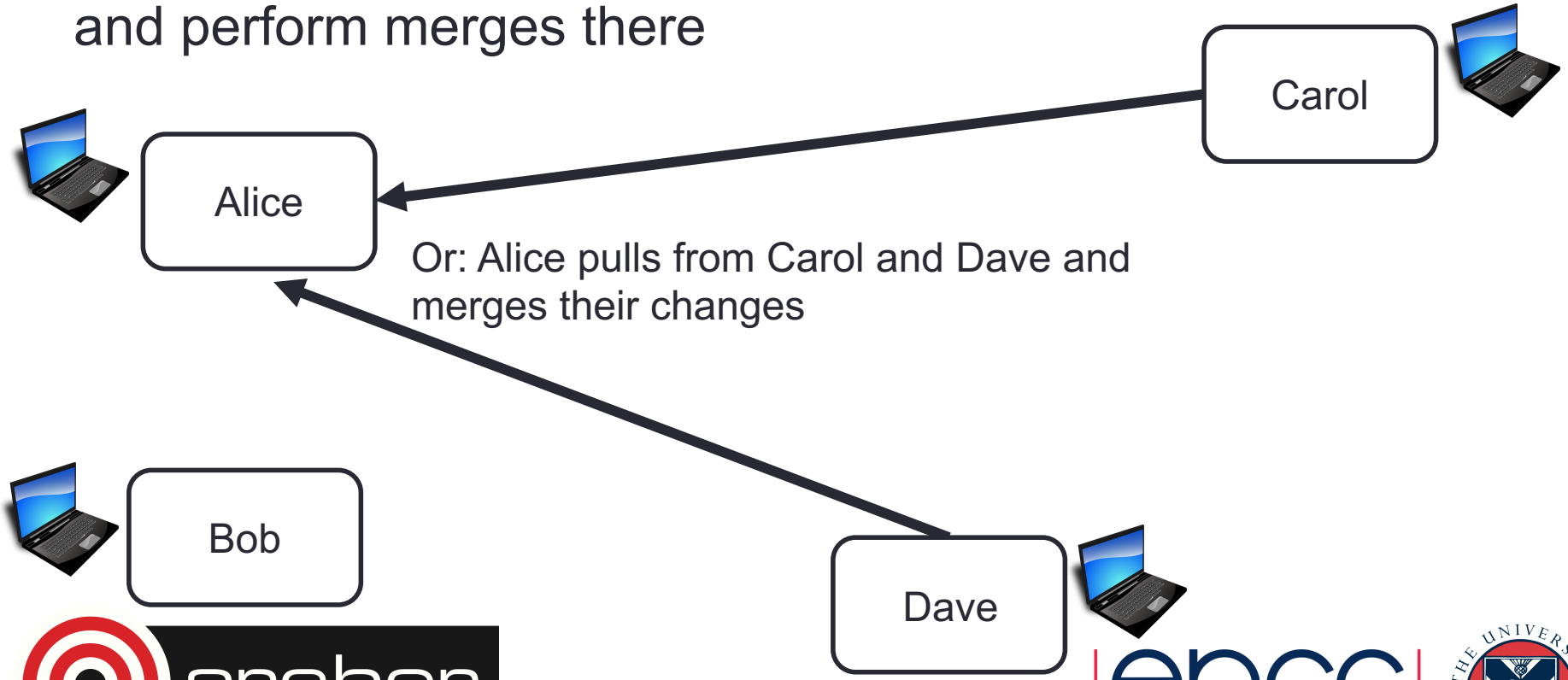
archer

epcc

THE UNIVERSITY OF EDINBURGH

Distributed version control

To combine content from different repositories someone has to fetch other people's changes into their working copy and perform merges there



archer

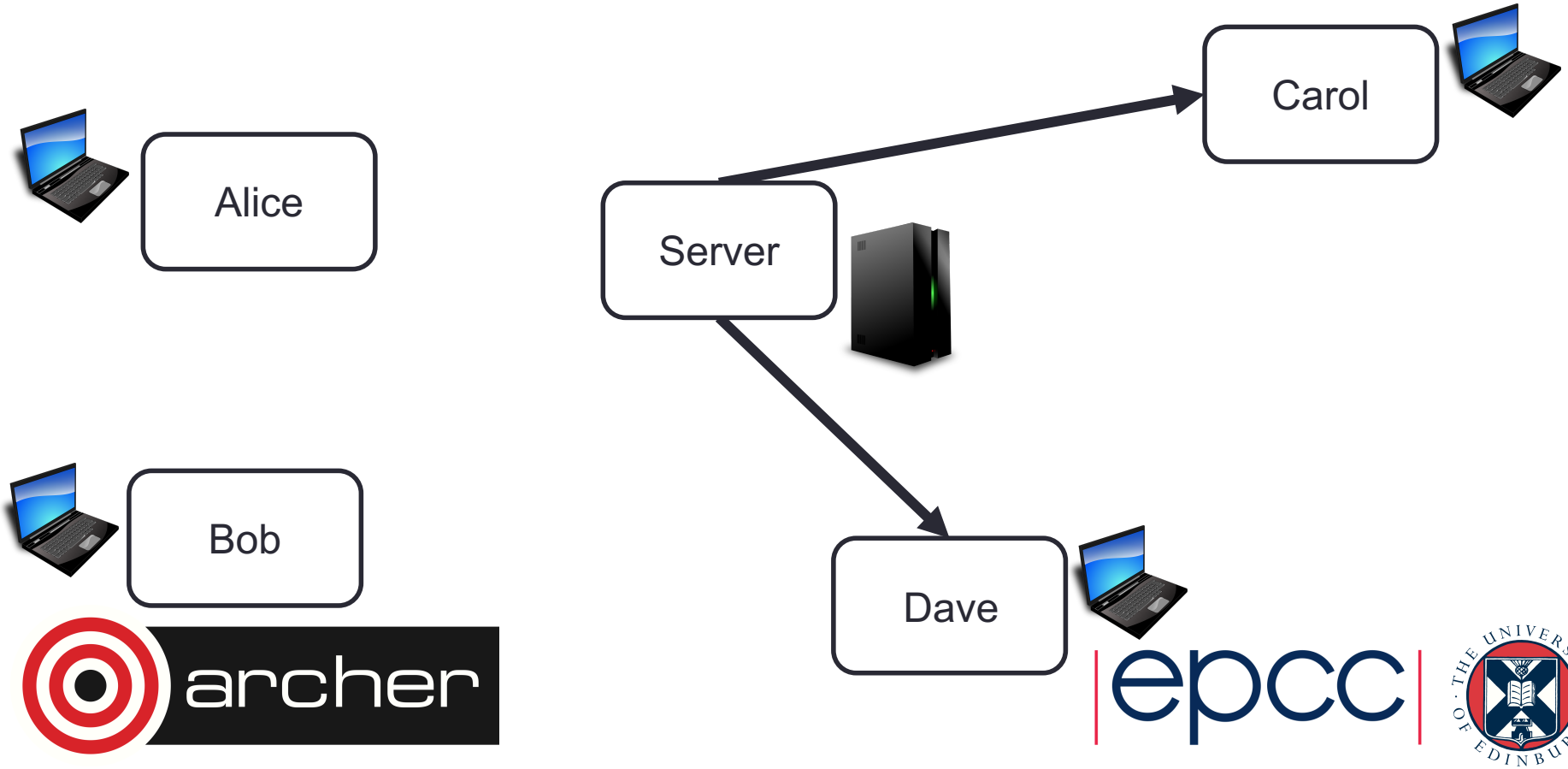
epcc

THE UNIVERSITY OF EDINBURGH

Distributed version control

Often use a central server for convenience:

- Clone



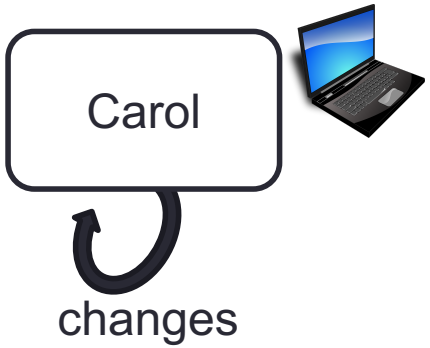
archer

epcc

THE UNIVERSITY OF EDINBURGH

Distributed version control

- Commit local changes



Distributed version control

- Push changes to server repository

Alice

Carol

Server

Bob

Dave

archer

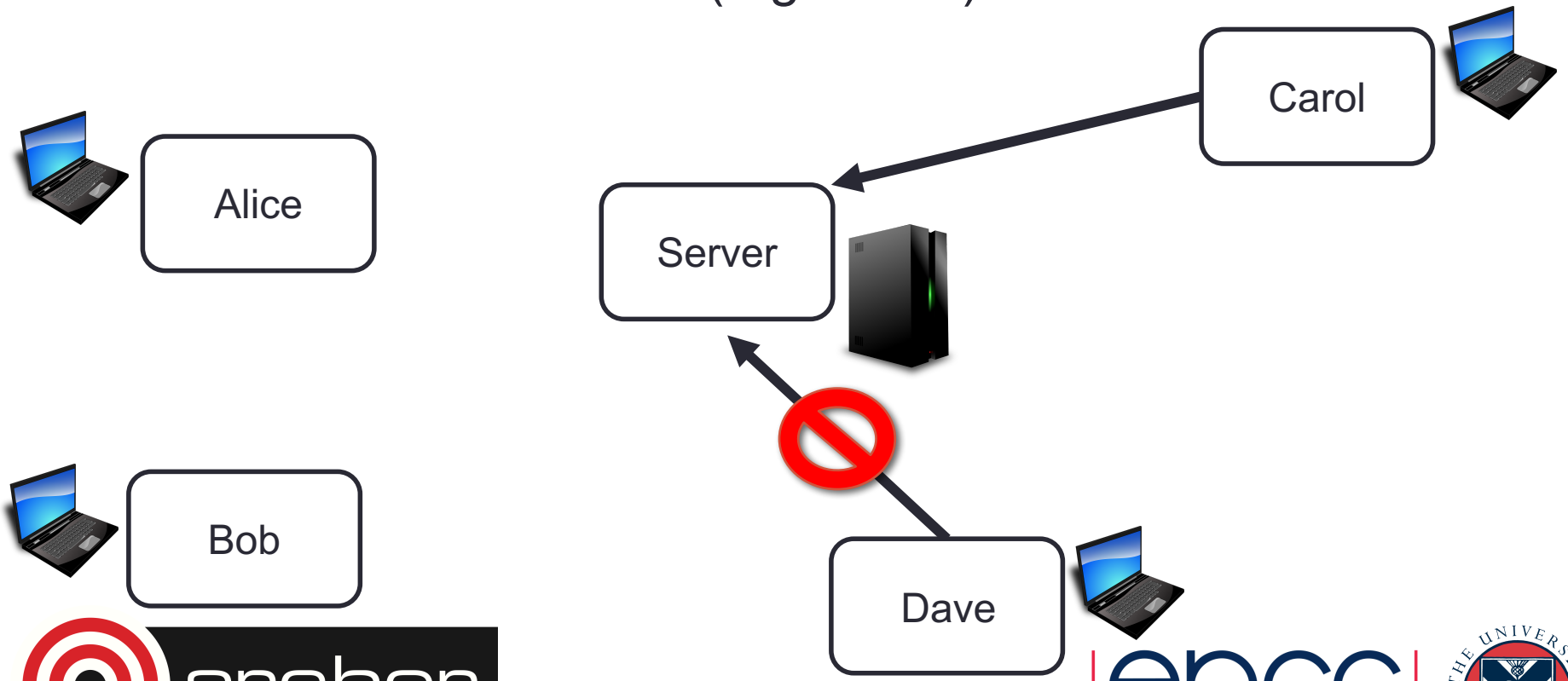
epcc

THE UNIVERSITY OF EDINBURGH

Distributed version control

If changes were made to master branch:

- First to commit to server (e.g. Carol) “wins”



archer

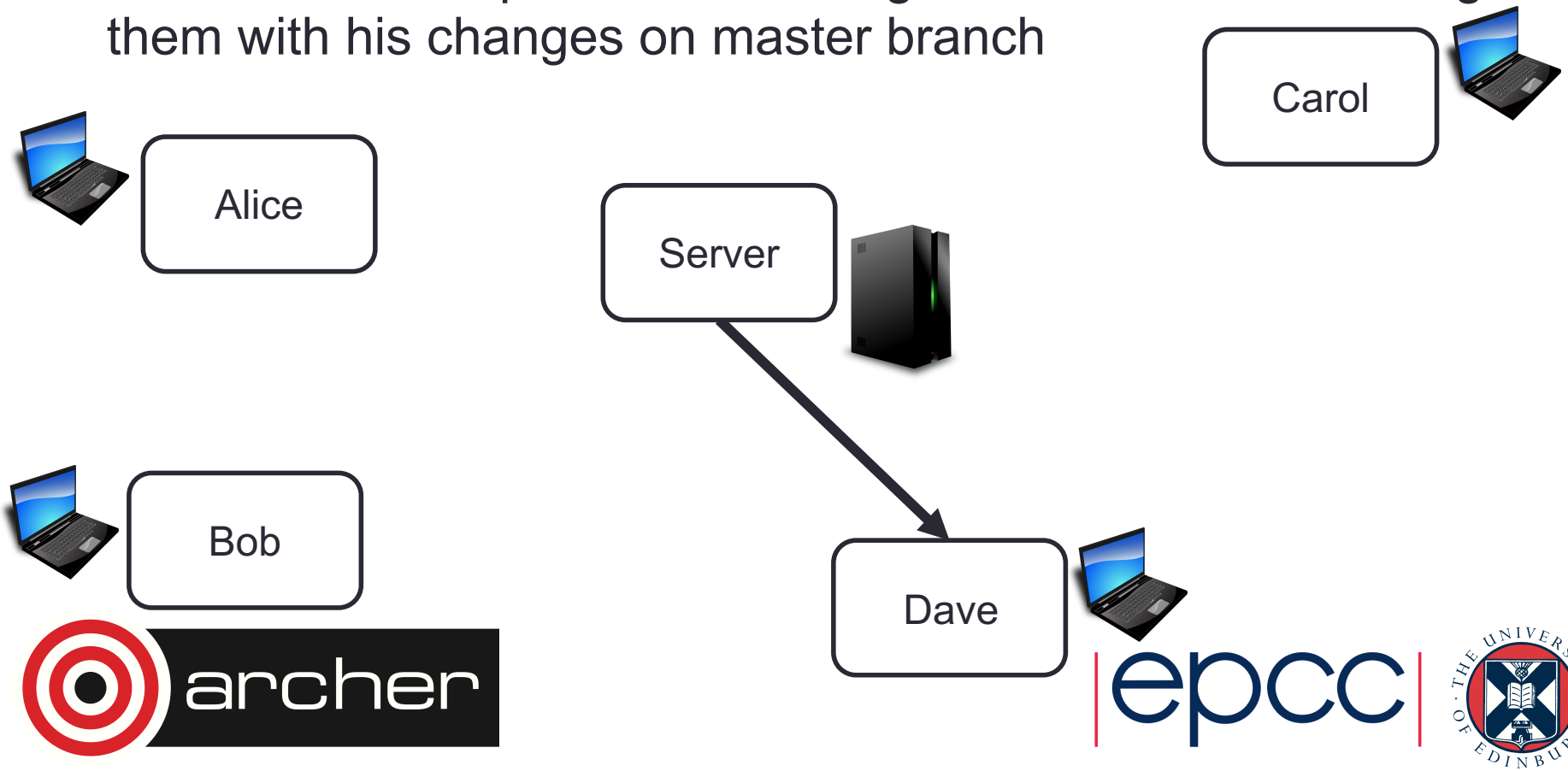
epcc

THE UNIVERSITY OF EDINBURGH

Distributed version control

If changes were made to master branch:

- Dave has to first pull Carol's changes from server and merge them with his changes on master branch



archer

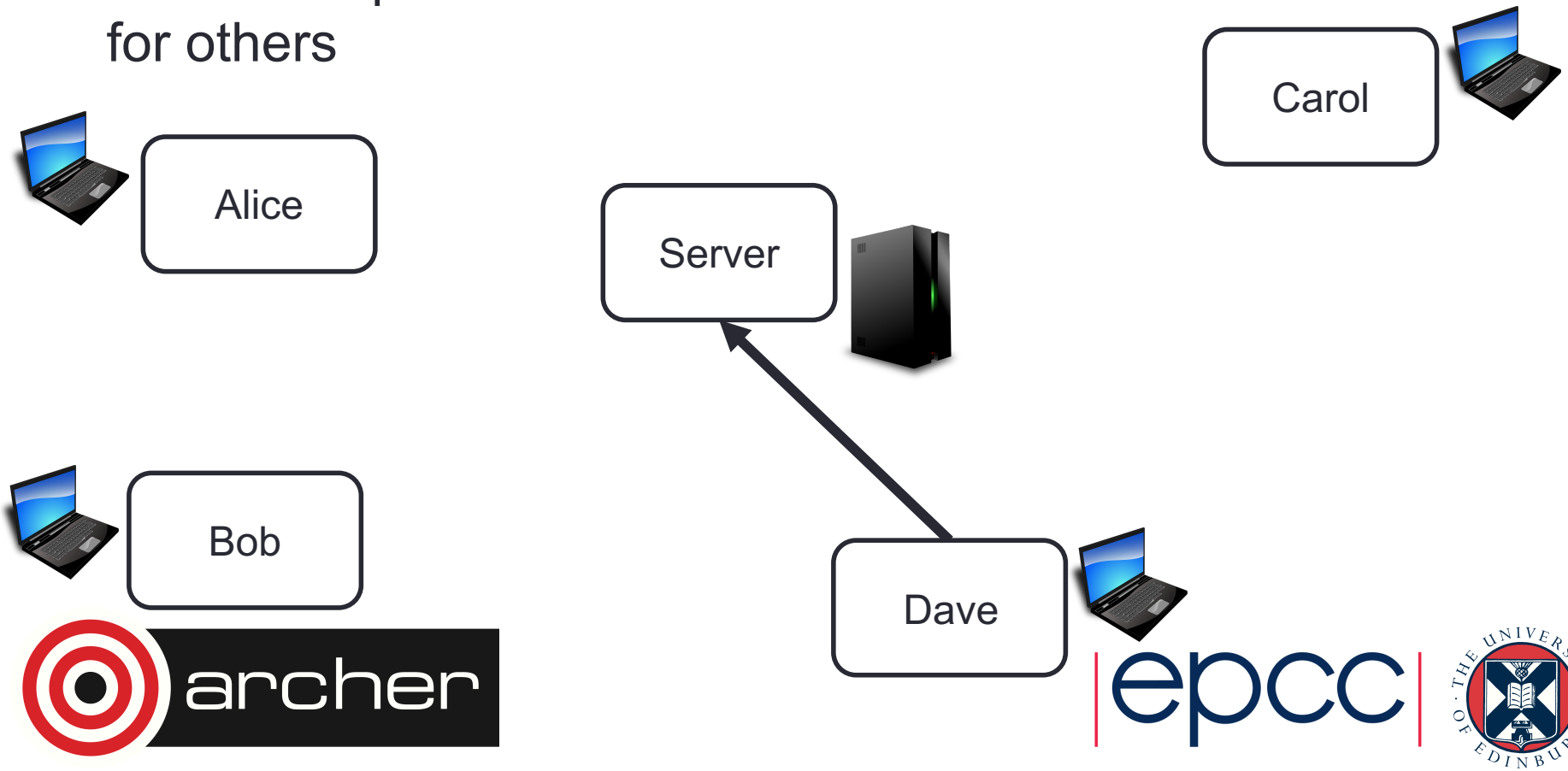
epcc

THE UNIVERSITY OF EDINBURGH

Distributed version control

If changes were made to master branch:

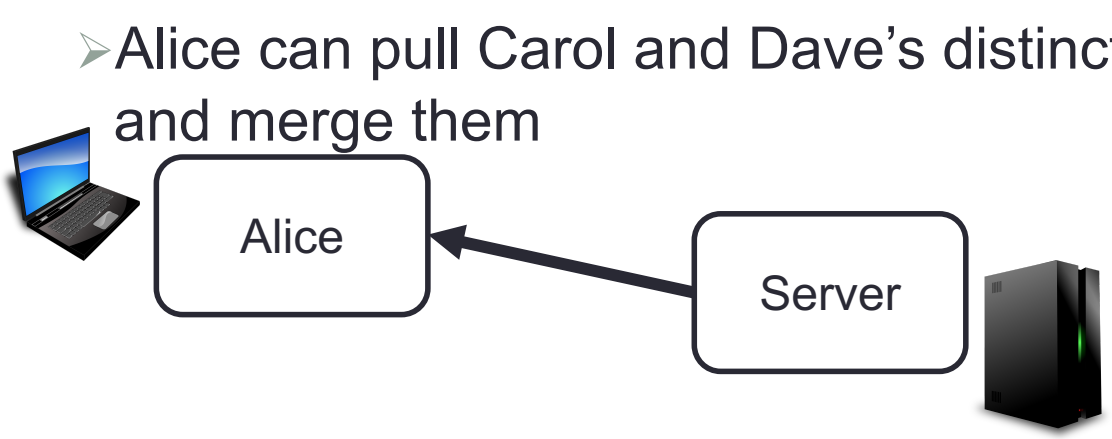
- Dave then pushes result back to master branch on server for others



Distributed version control

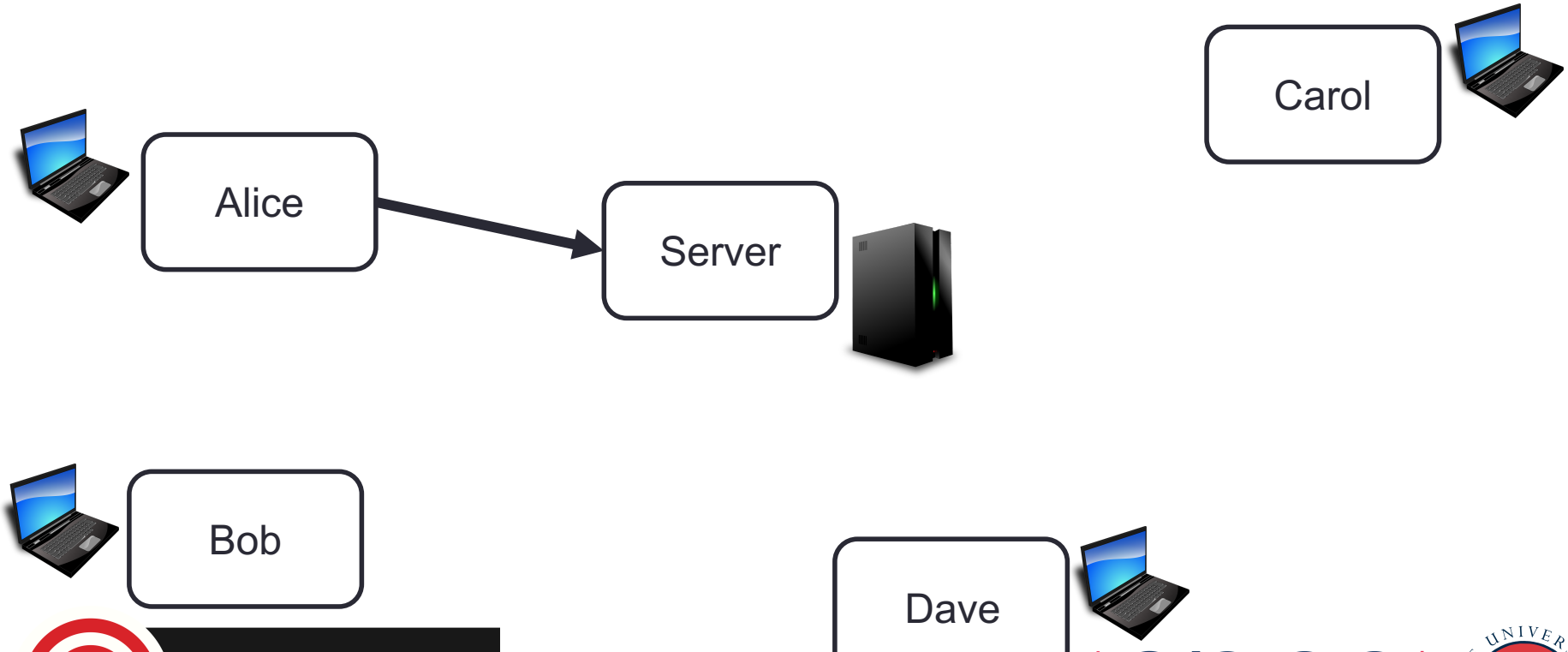
If changes were made to new branches:

- Carol and Dave push to different branches on server
- Alice can pull Carol and Dave's distinct branches and merge them



Distributed version control

- Alice then pushes result back to a single branch on server (can be master branch or another branch) for others



Distributed version control

- Don’t need to be online to commit changes
- Changes can be committed privately
  - Encourages committing early on
  - Encourages branching to commit e.g. experimental code
- Full revision history (log & past versions) available locally
- Can adopt workflows other than centralised for combining content from contributors
- Many common operations are faster because no communication with server needed

Hosting & additional features

Hosting & additional features

Distributed version control systems became very popular over the past ~8 years (Git born 2005)

A number of websites (GitHub, GitLab, Bitbucket, …) have helped fuel this trend and exploit the potential of distributed version control.

GitHub *et al*. offer repository hosting and management and additional features that facilitate collaborative software development

Hosting & additional features

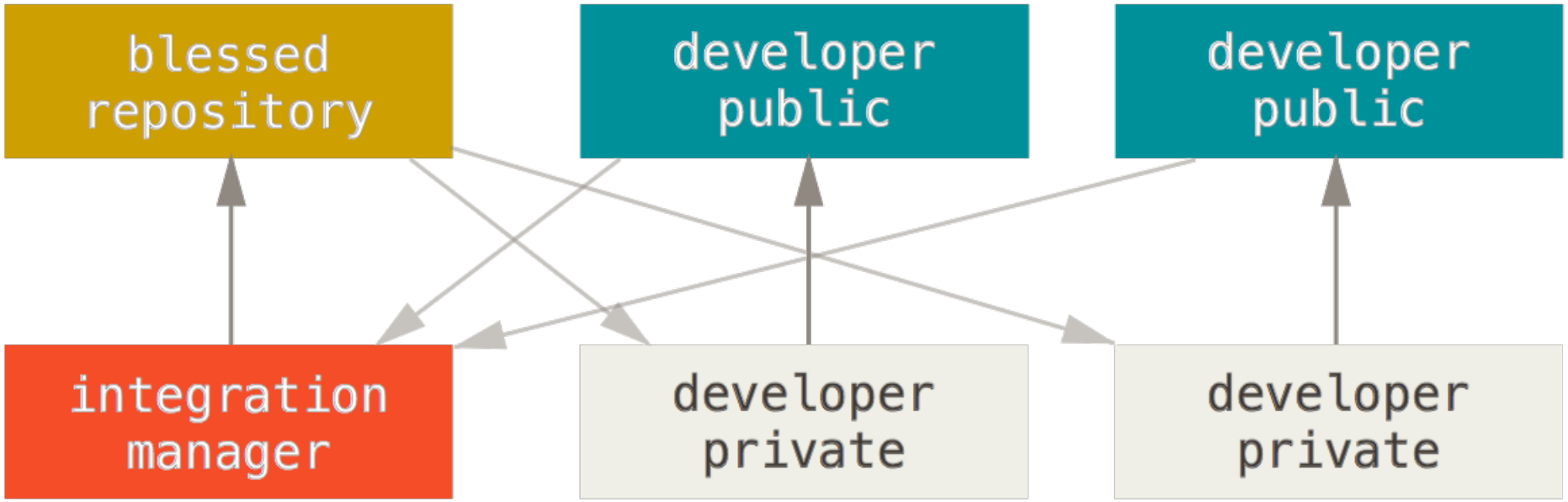
Aditional features:

- Issue tracker to track and discuss bugs, feature requests etc. tightly integrated with version control workflow
- “Pull request” mechanism allowing developers to clone (fork) a repository, make changes, then suggest to the original owner that these changes are integrated into the parent repository
  - Useful for integrating contributions from unknown developers – no need to give write access to original project repository
- Can set up pipelines / jobs that run server-side after a commit is pushed – e.g. for continuous integration testing

GitLab site-installable web-based repository management frameworks offer similar features.

Distributed workflows

Integration manager workflow:



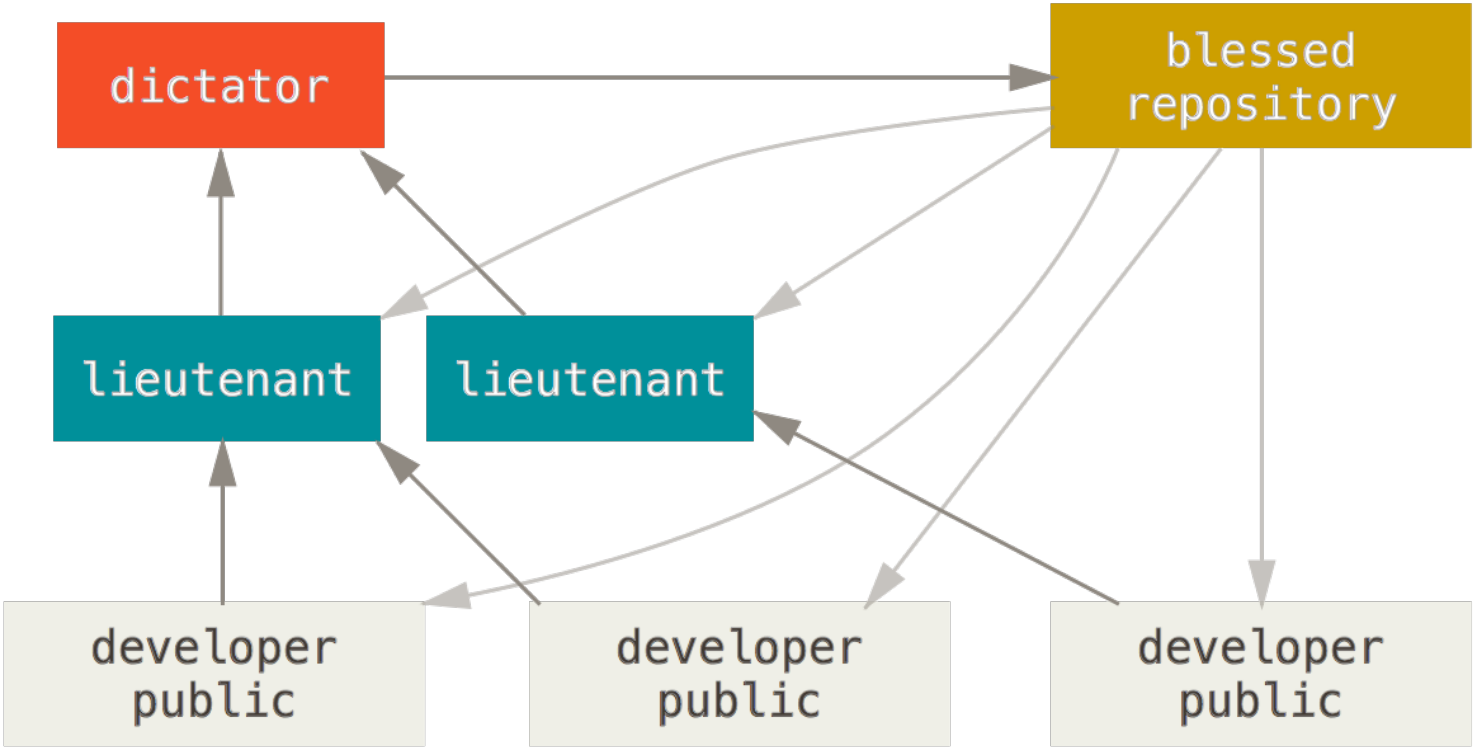
archer

epcc

THE UNIVERSITY OF EDINBURGH

Distributed workflows

"Dictator and lieutenants" workflow:



Final words

- Version control systems are not a magic bullet, but a powerful tool
- You still need to think and decide how to manage your work
- When working collaboratively, need to communicate

References

- https://oer.gitlab.io/oer-on-oer-infrastructure/Git-introduction.html
- https://backlog.com/git-tutorial/
- Quick Introduction to Version Control with Git and Github:
  - https://doi.org/10.1371/journal.pcbi.1004668

http://www.archer.ac.uk/training/

- Face-to-face courses
  - timetable, information and registration
  - material from all past courses
- Virtual tutorials
  - timetable plus slides and recordings from past courses
  - please leave feedback on previous tutorials after viewing material

Goodbye!

Thanks for attending

Please leave feedback at:
www.archer.ac.uk/training/feedback/online-course-feedback.php

archer

epcc

THE UNIVERSITY OF EDINBURGH