

Software Engineering

Version Control with git

Aims of today's Lecture

- Know what configuration management is.
- Know how (distributed) version control systems work.
- Know development workflows for teams with VCSs.



new.psd



newfinal.psd



newfinalfinal.psd



newfinalestfinal.psd



newfinalestfinal
forsure.psd

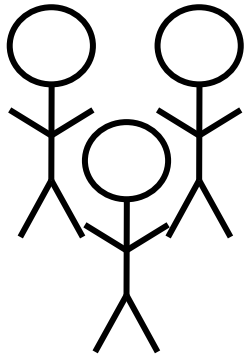


newfinalestfu#%
this\$h%tfinal.psd

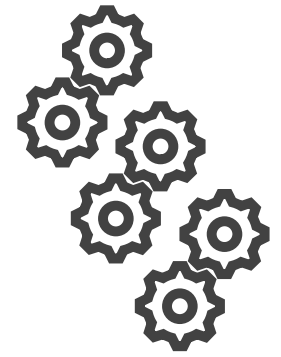
Software Configuration Management & **Version Control**

Challenges in Software Development

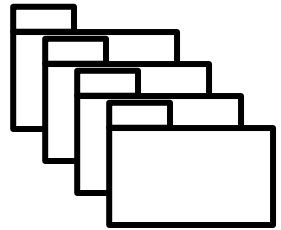
Many developers



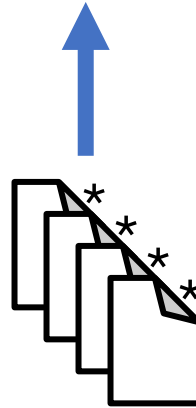
Many requirements



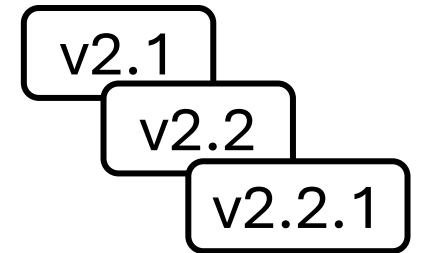
We need
coordination.



Many projects



Many changes



Many versions

SCM Functionalities

There are four main aspects of SCM:

- Version control (VCS): Manage code changes in sequential and parallel development, allow collaborative development.
- Change management: Track and plan change requests.
- Build management: Collect, compile and link code and libraries into executable software.
- Release management: Prepare new releases and manage old releases.

In this lecture, we focus on VCS. The later ones follow in lectures for process models, management and deployment.

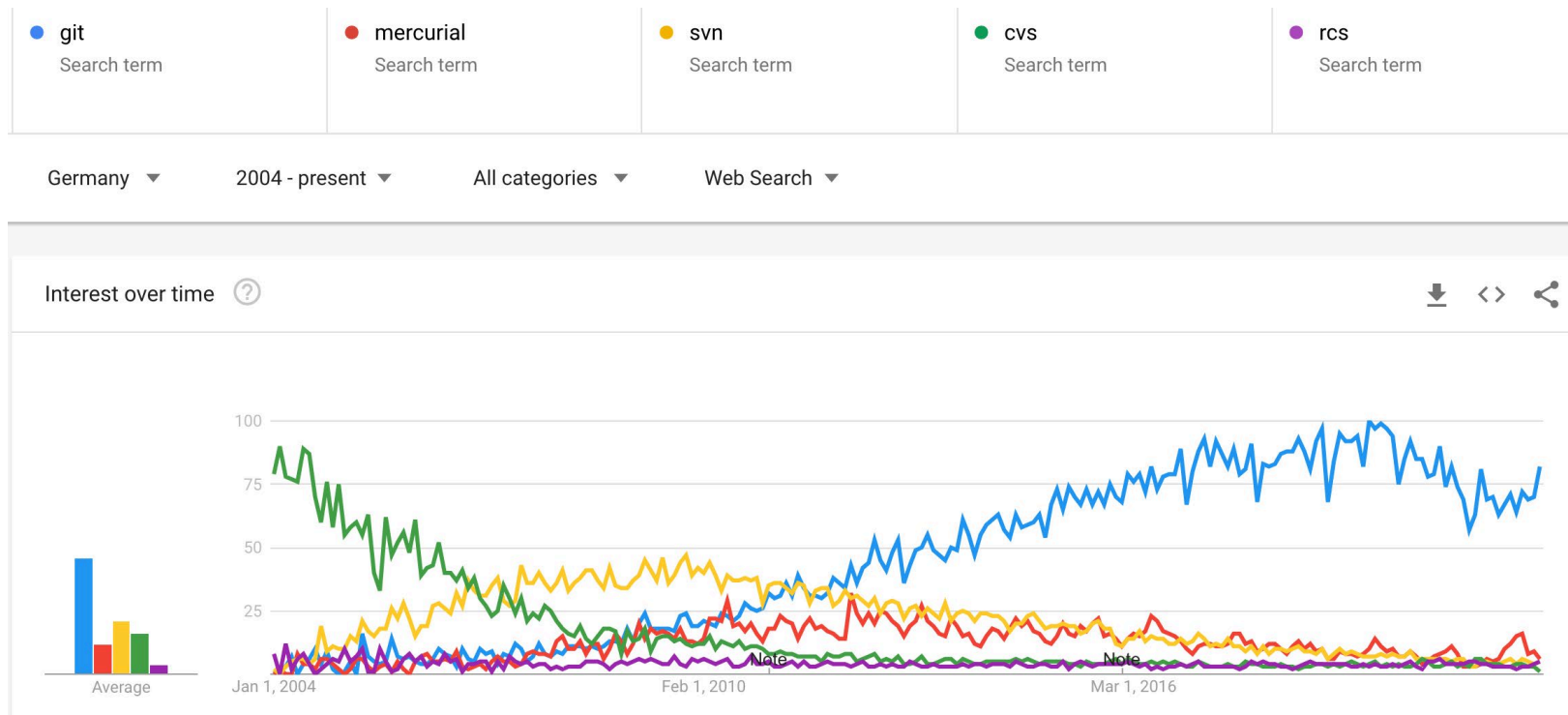
Why Version Control?

We want (and need to):

- know who has changed what, when, why.
- go back to older version.
- share changes.
- work in parallel and manage integration.
- manage versions and variants.

Common VCS Tools

- Local only: SCCS (1972), RCS (1982)
- Centralized: CVS (1986), SVN (2000)
- Decentralized: git (2005), mercurial (2005)





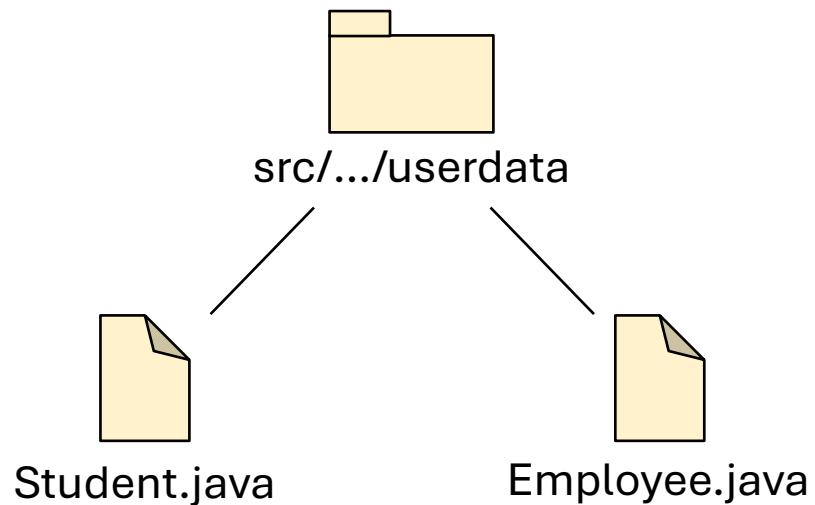
git

git Basics

Working Directory and Repository

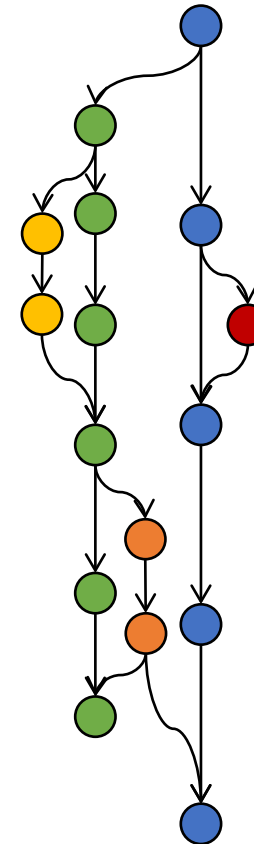
Working Directory

Contains "current state" of a developer's version of the project.

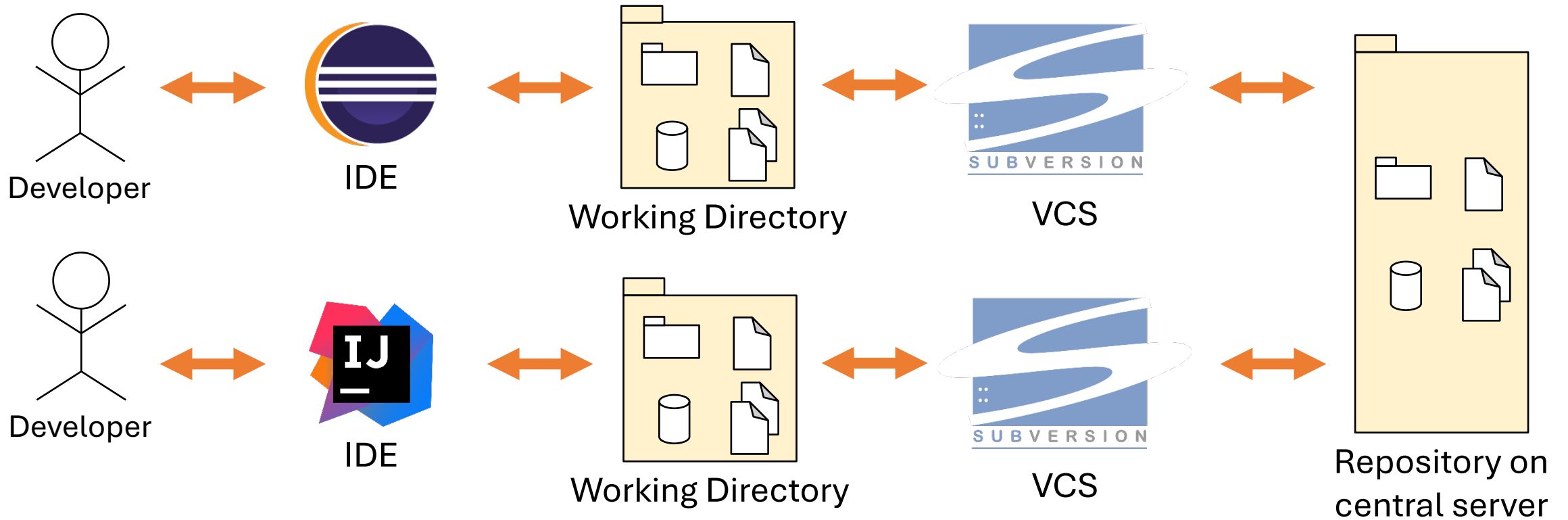


Repository

Contains all versions and all variants of the project.



CVS und SVN: Centralized VCS



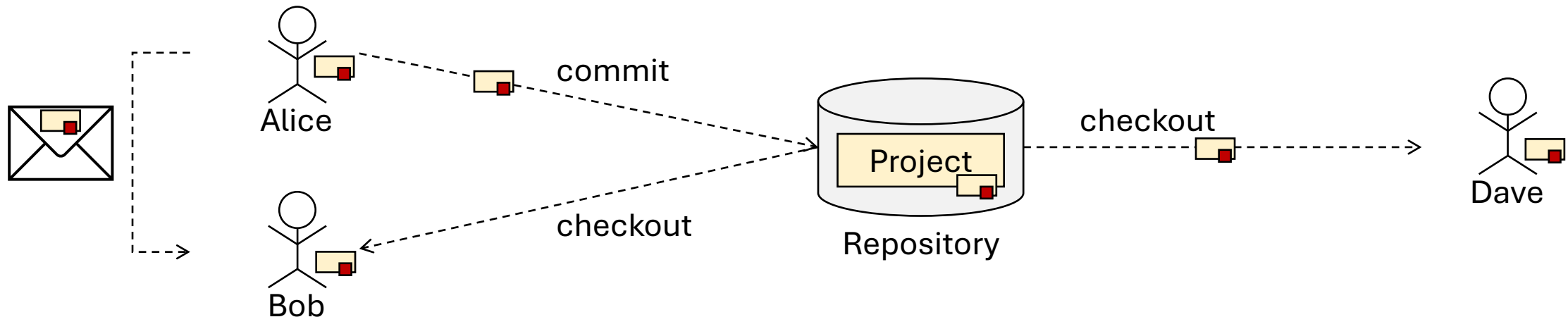
Principle: One central repository ("official version"). Each dev has their own working directory with copies of the project (may contain new, temporary, unfinished,... versions)

Exchanging Intermediate State in Centralized VCS

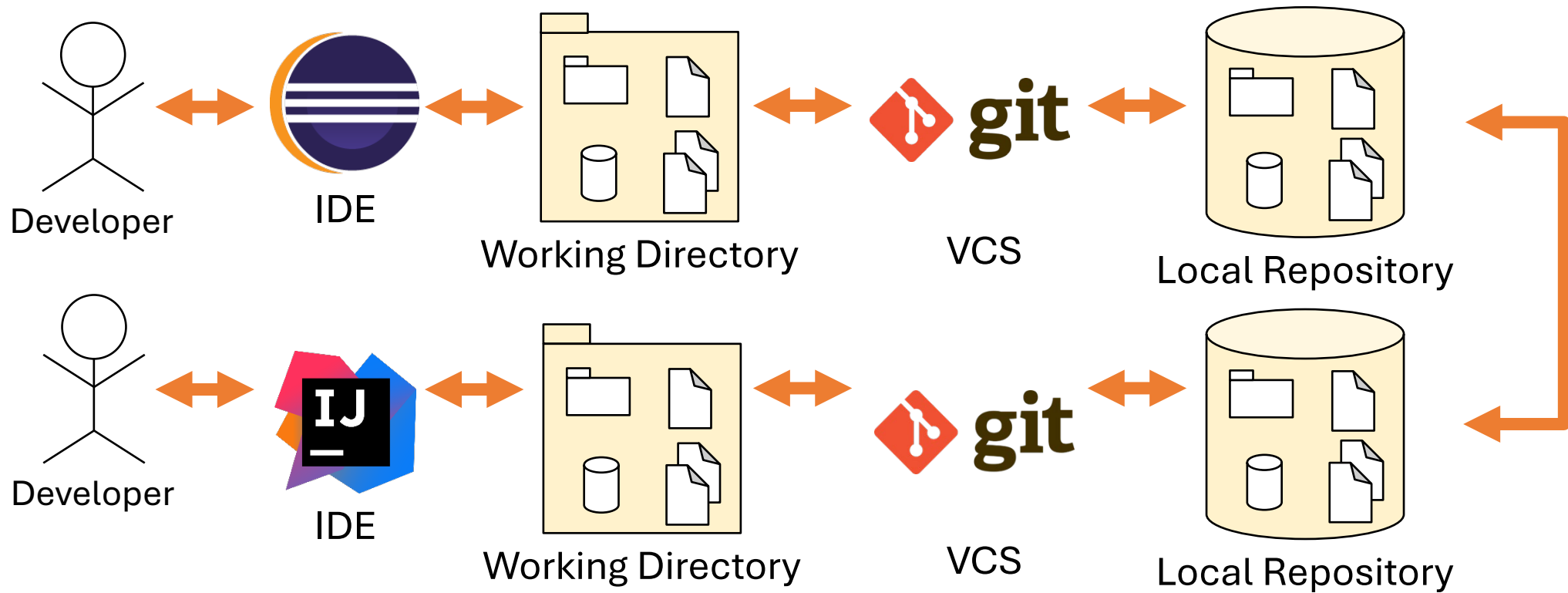
Alice works on a feature of the project and is ~half-way through. Bob works on another feature that uses Alice's feature, so he asks her to send him her current state.

There are two possibilities:

- Commit to repository.
- Send via e-mail.

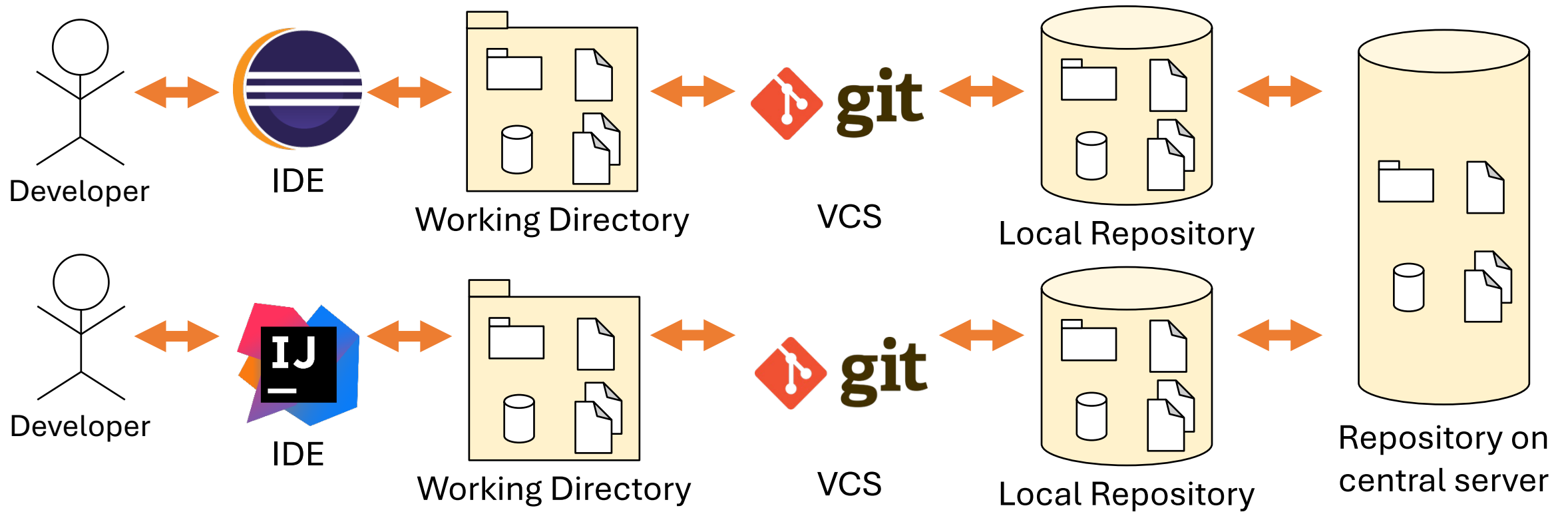


git: Decentralized VCS



Principle: Each developer has their own local repository. There is not "the one" repository. Developers can exchange their work by pushing changes to their colleagues' repositories.

git: Centralized-Decentralized VCS



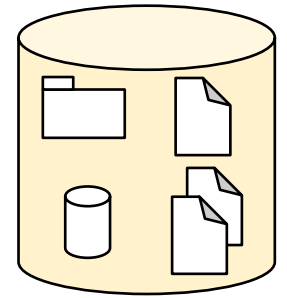
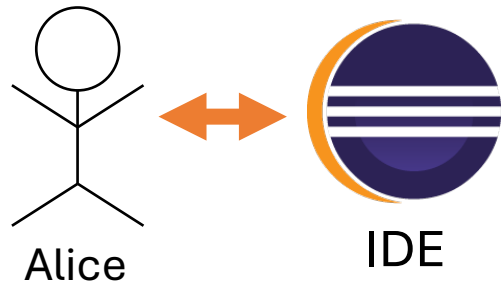
Principle: Each developer has a local repository but there is one central repository every developer pushes their changes to.

Remote Repositories

Every repository is treated equivalent. That means, every repository can be cloned, and one can push/pull to/from every repository. Prerequisite is that the targeted repository is set as a remote.

- remote = different, known repository, specified via url or path.
- origin = remote repository from which the local repository was cloned.
- fetch = get information about new remote branches and changes without integrating them.
- tracking = mapping of local and remote branches.
 - Push/pull always operate on branches that track a remote branch.

Initial Situation



Repository on
central server

Cloning a Project



```
git clone <address>
```

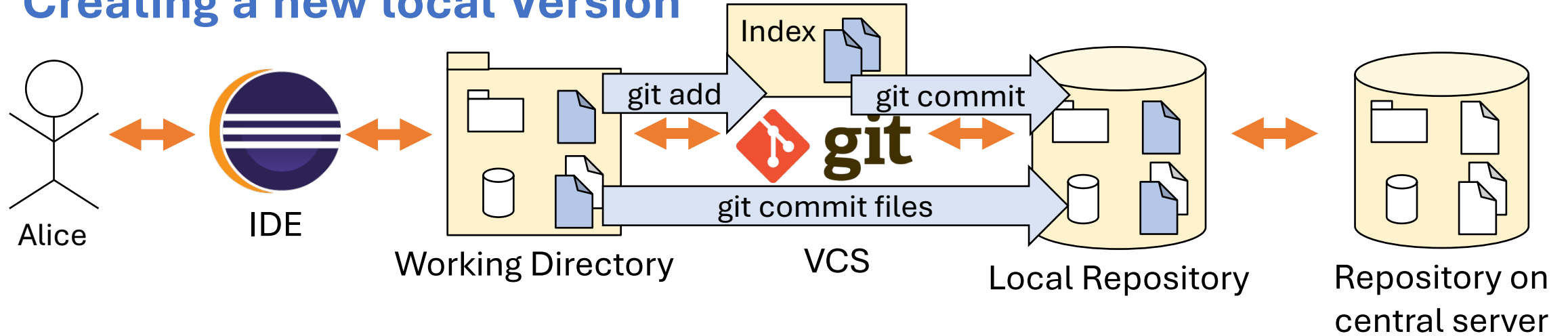
Create a local repository with a copy of the current version from the origin.

Making a Change



The developer changes (create new files, delete existing files, change content of files) some files in their working directory.

Creating a new local Version



```
git add <file names>
```

```
git commit <commit message>
```

```
git commit <commit message> <file names>
```

The developer selects which changes should be integrated in the repository. They add several changes to the index and then create a commit with these changes.

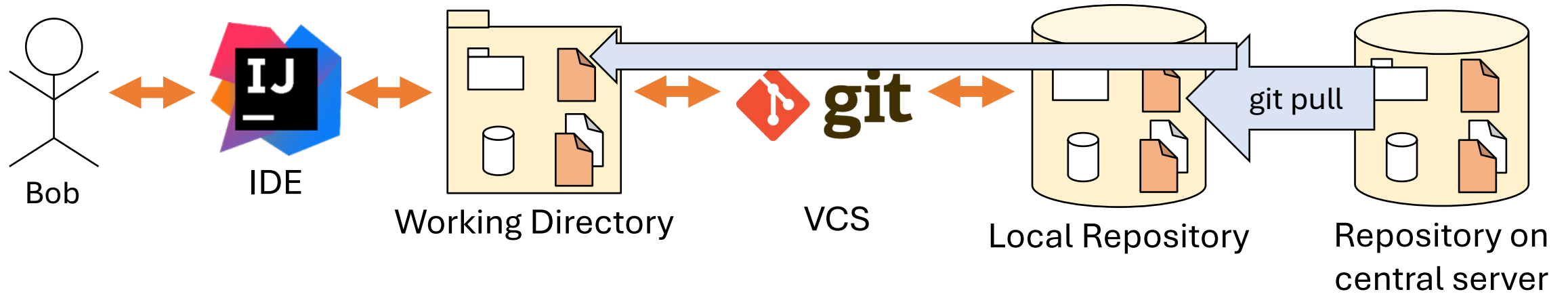
Transfer current Version from Local to Remote Repository



```
git push <remote>
```

The developer pushes the latest commits to the remote/origin repository.

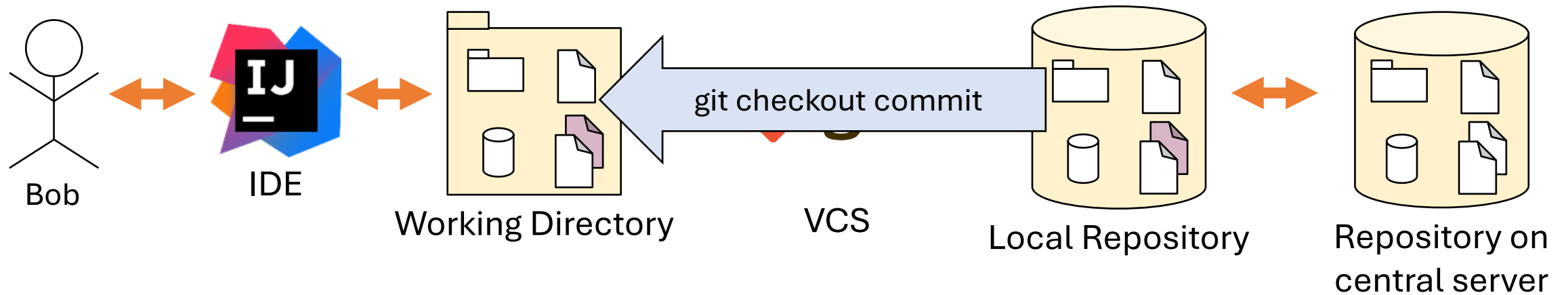
Get current Version from Remote Repository



```
git pull <remote>
```

The developer pulls the latest changes from the remote/origin repository into their local repository and working directory.

Get specific Version from Local Repository



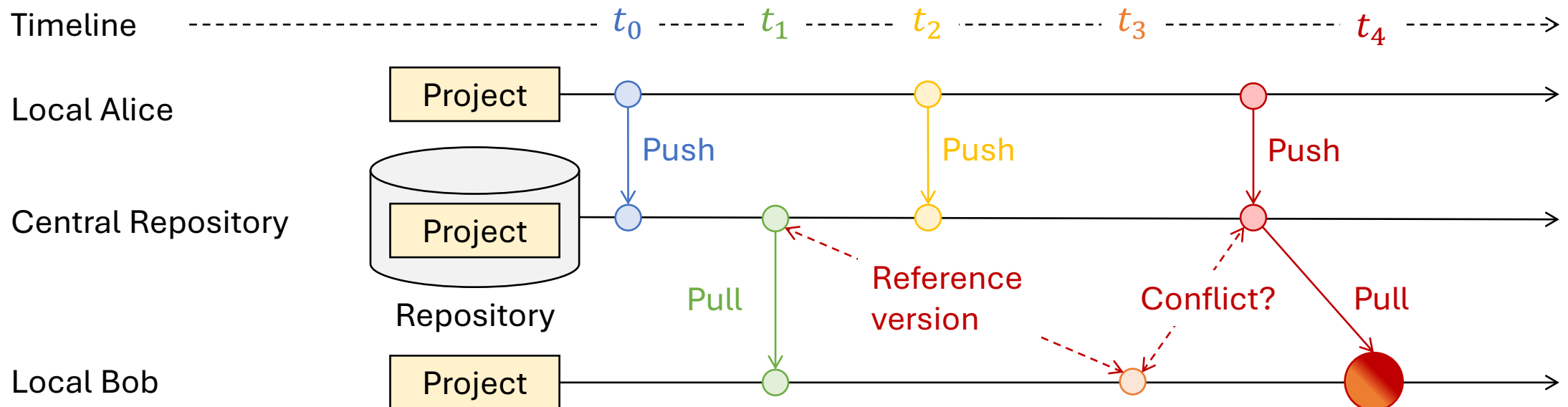
```
git checkout commit <commit id>
```

The developer load the version of a certain commit into the working directory.

Problem: Simultaneous Local and Remote Changes

What happens when we have changes in our local repository and pull from remote, where new changes happened as well?

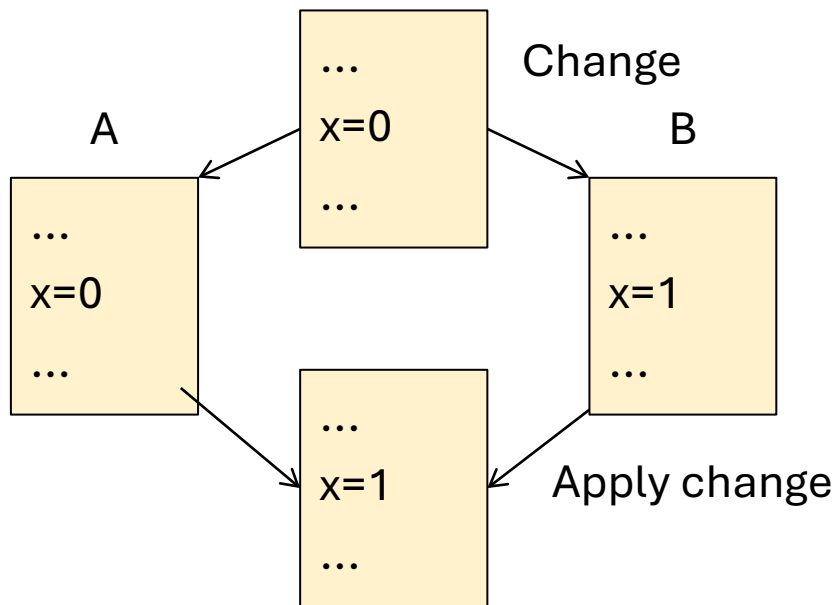
Git tries to perform an automatic merge. If there are changes to the same lines of a text-based file, automatic merging is not possible, and the developer needs to decide how to merge.



Automatic 3-Way-Merge

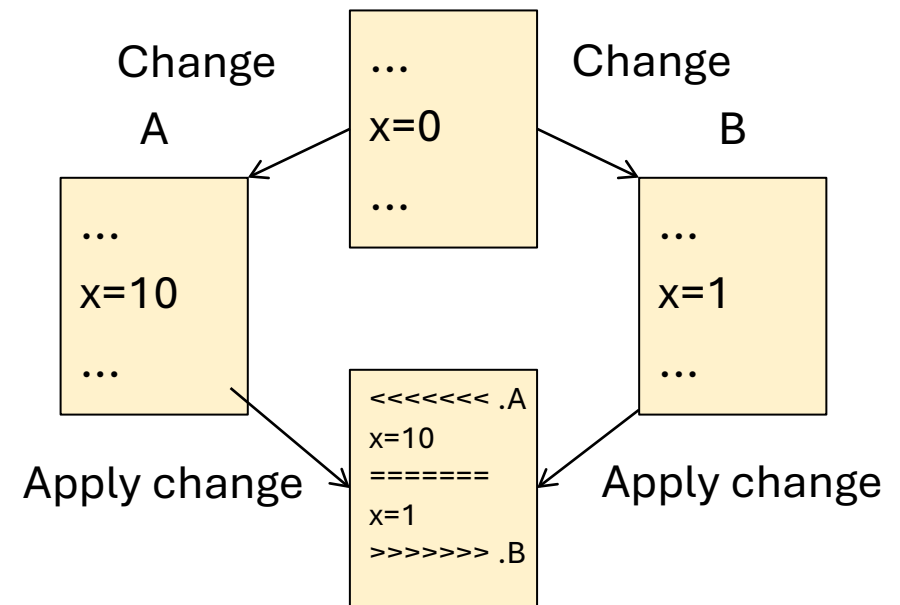
Automatic Merging is based on the reference version (newest common ancestor)

Successful Merge



Dev does not need to do anything.

Merge Conflict



Dev must choose what to keep.

Side-by-Side View in IDE

```
f357c425d89805ce258e2bd71ed5c6a7eab0b4d9 Local
13 13
14 14
15 15
16 16
17 17
18 18
19 19
20 20
21 21
22 22
23 23
24 24
25 25
26 26
27 27
28 28
29 29
30 30
31 31
32 32
33 33
34 34
35 35

// print banner
System.out.println("*****");
System.out.println("*** Customer owes ***");
System.out.println("*****");

// calculate outstanding
while (e.hasMoreElements()) {
    Order each = (Order) e.nextElement();
    outstanding += each.getAmount();
}

// print details
System.out.println("name"+ _name);
System.out.println("amount"+ outstanding);
}

private static class Order {
    public double getAmount() {
        return 1;
    }
}

// print banner
System.out.println("*****");
System.out.println("*** Customer owes ***");
System.out.println("*****");

// calculate outstanding
while (e.hasMoreElements()) {
    Order each = e.nextElement();
    outstanding += each.getAmount();
}

// print details
System.out.println("name"+ _name);
System.out.println("amount"+ outstanding);
}

private static class Order {
    public double getAmount() {
        return 1;
    }
}
```


Parallel Development in "Branches"

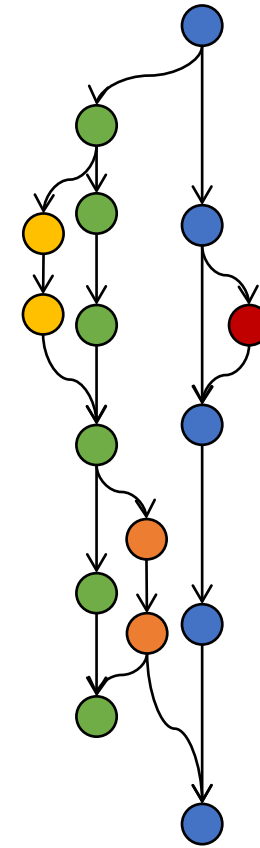
Idea:

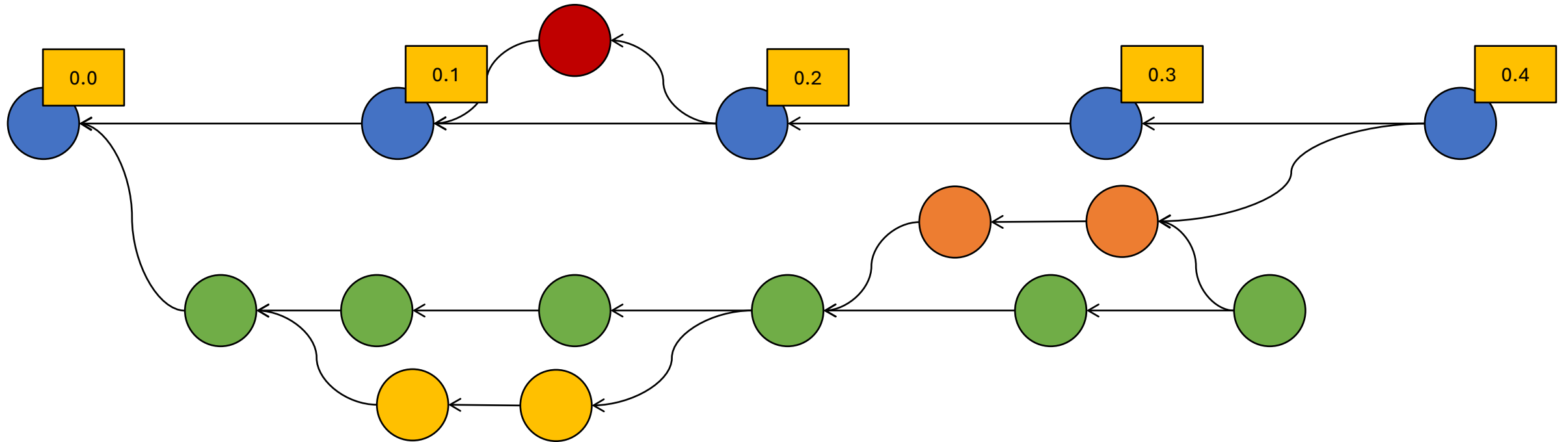
- Versions = sequential development line.
- Variants = parallel development lines.

Application examples:

- Bug fixes for current release while working on next release in parallel.
- Work on different features in parallel.

A branch allows to build a variant of the system. At some point in the repository's history, the development splits up, creating the branch. At some point, we want to integrate the changes of the branch again. For this, we merge our branch into the target branch.

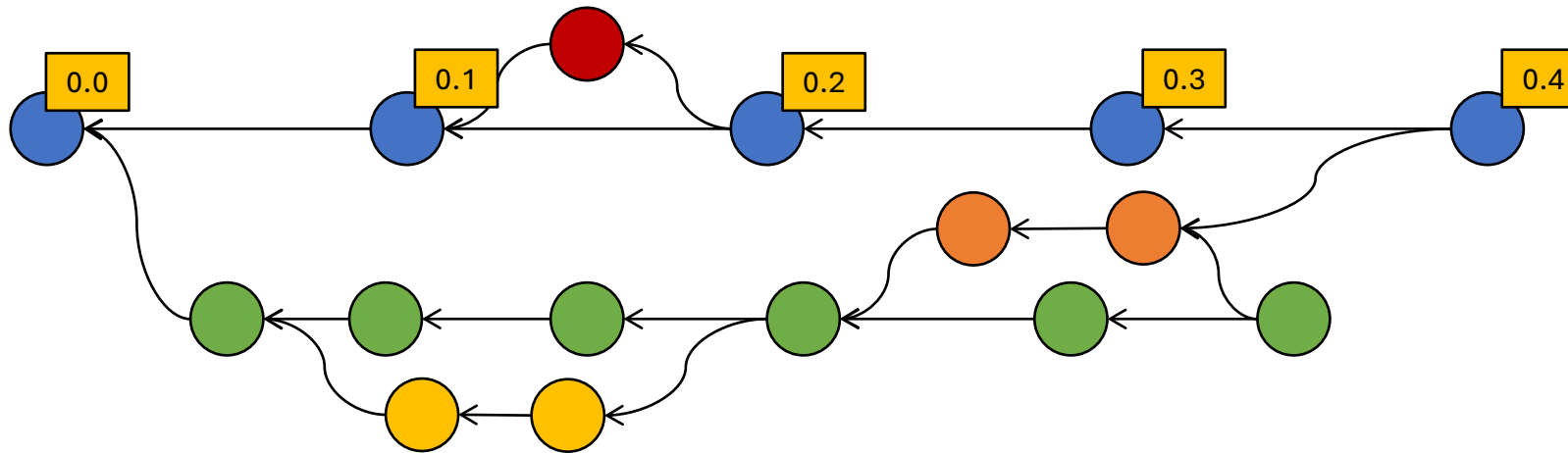




Git Good Habits

Feature Branch Flow

- Main/Master only for deployable releases.
- Development as main development branch.
- Each new feature gets its own branch.
- For upcoming release: create release branch.
- Maintenance/Hotfix Branches may directly merge into Main.



Sensible Commits

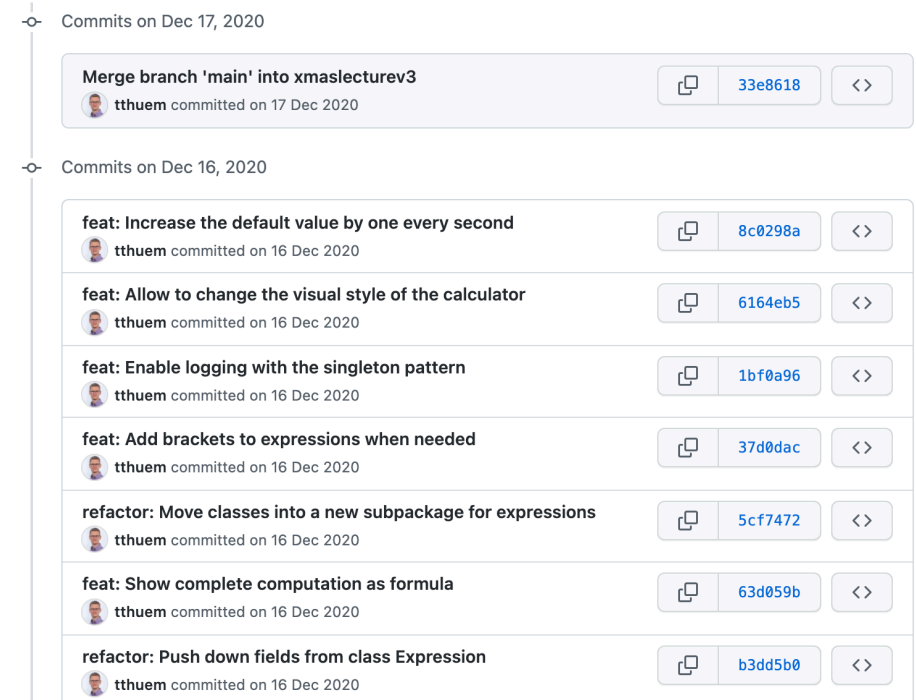
- Commits should be fine-grained, e.g., a bugfix, a feature,...
- First pull, then push. Do not force push.
- Integrate into the main development branch regularly, but only tested code.
- Commit messages should be descriptive.



	COMMENT	DATE
○	CREATED MAIN LOOP & TIMING CONTROL	14 HOURS AGO
○	ENABLED CONFIG FILE PARSING	9 HOURS AGO
○	MISC BUGFIXES	5 HOURS AGO
○	CODE ADDITIONS/EDITS	4 HOURS AGO
○	MORE CODE	4 HOURS AGO
○	HERE HAVE CODE	4 HOURS AGO
○	AAAAA	3 HOURS AGO
○	ADKFJSLKDFJSDKLFJ	3 HOURS AGO
○	MY HANDS ARE TYPING WORDS	2 HOURS AGO
○	HAAAAAANDS	2 HOURS AGO

AS A PROJECT DRAGS ON, MY GIT COMMIT MESSAGES GET LESS AND LESS INFORMATIVE.

vs.



Commits on Dec 17, 2020		
Merge branch 'main' into xmaslecturev3	33e8618	<>
tthuem committed on 17 Dec 2020		
Commits on Dec 16, 2020		
feat: Increase the default value by one every second	8c0298a	<>
tthuem committed on 16 Dec 2020		
feat: Allow to change the visual style of the calculator	6164eb5	<>
tthuem committed on 16 Dec 2020		
feat: Enable logging with the singleton pattern	1bf0a96	<>
tthuem committed on 16 Dec 2020		
feat: Add brackets to expressions when needed	37d0dac	<>
tthuem committed on 16 Dec 2020		
refactor: Move classes into a new subpackage for expressions	5cf7472	<>
tthuem committed on 16 Dec 2020		
feat: Show complete computation as formula	63d059b	<>
tthuem committed on 16 Dec 2020		
refactor: Push down fields from class Expression	b3dd5b0	<>
tthuem committed on 16 Dec 2020		

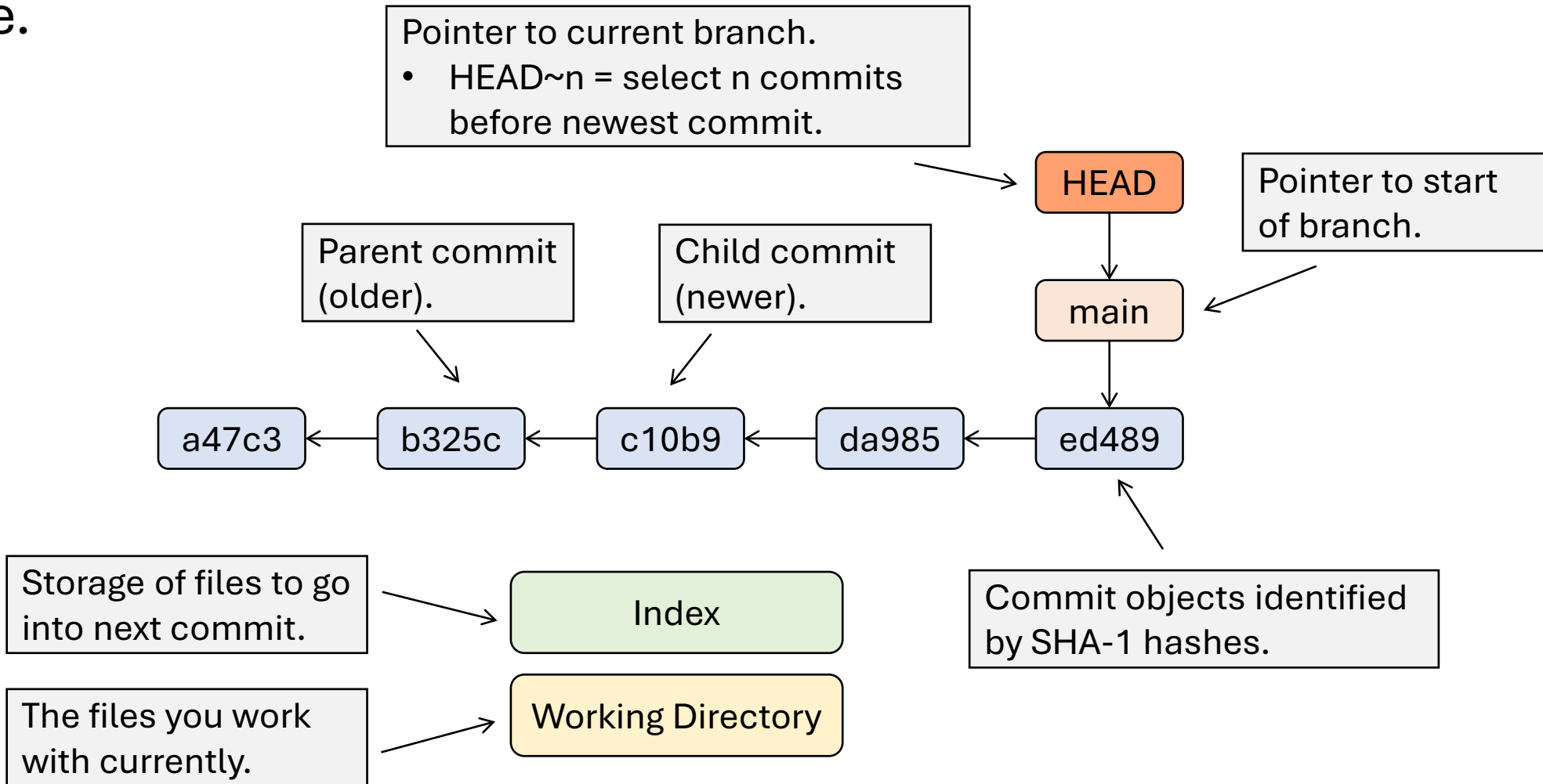


git Commands "under the hood"

You do not need to know these things for the exam, but they *might* be helpful when working with git.

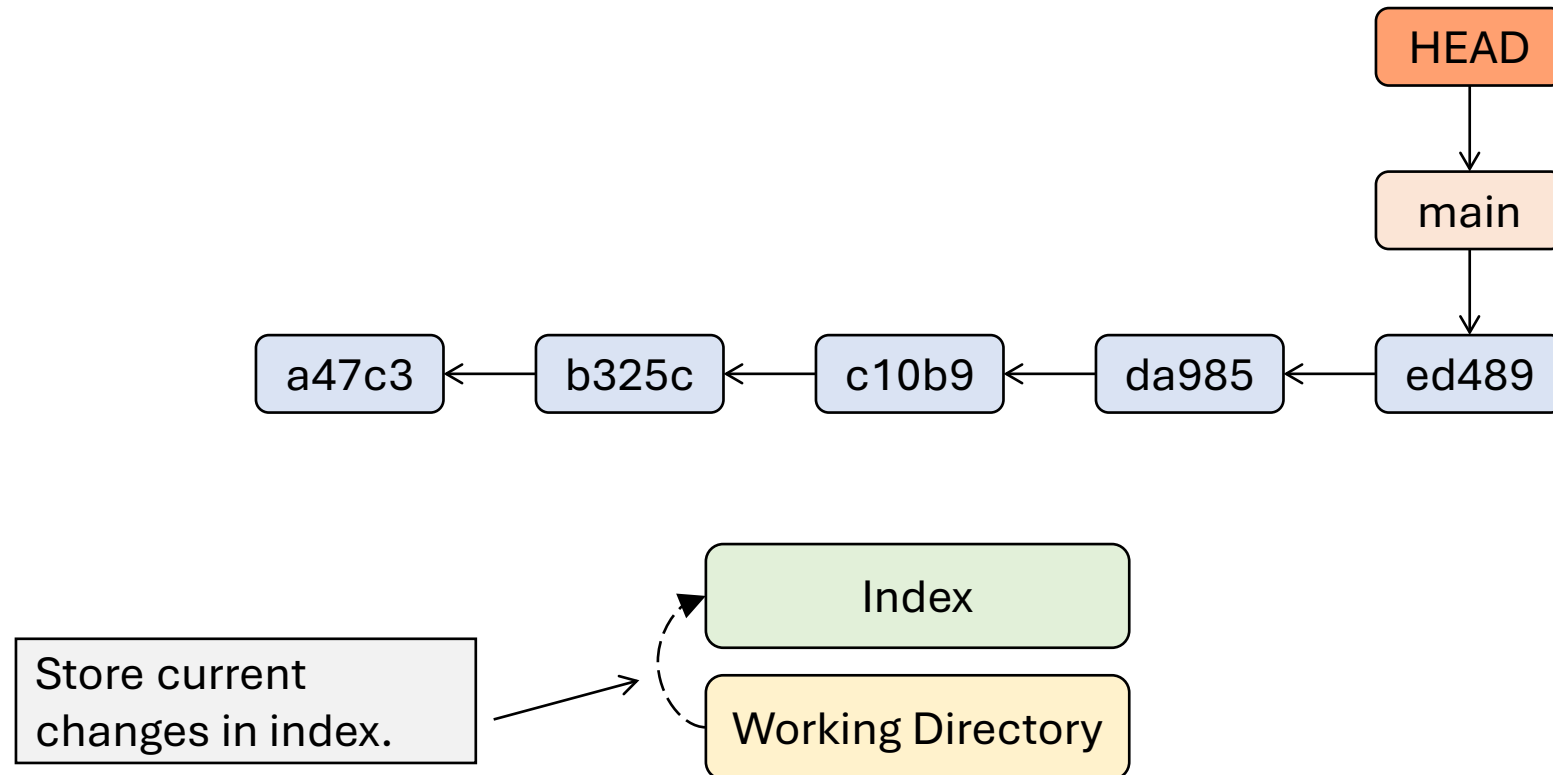
Git Implementation

A commit is a snapshot of the files under version control at a given time.



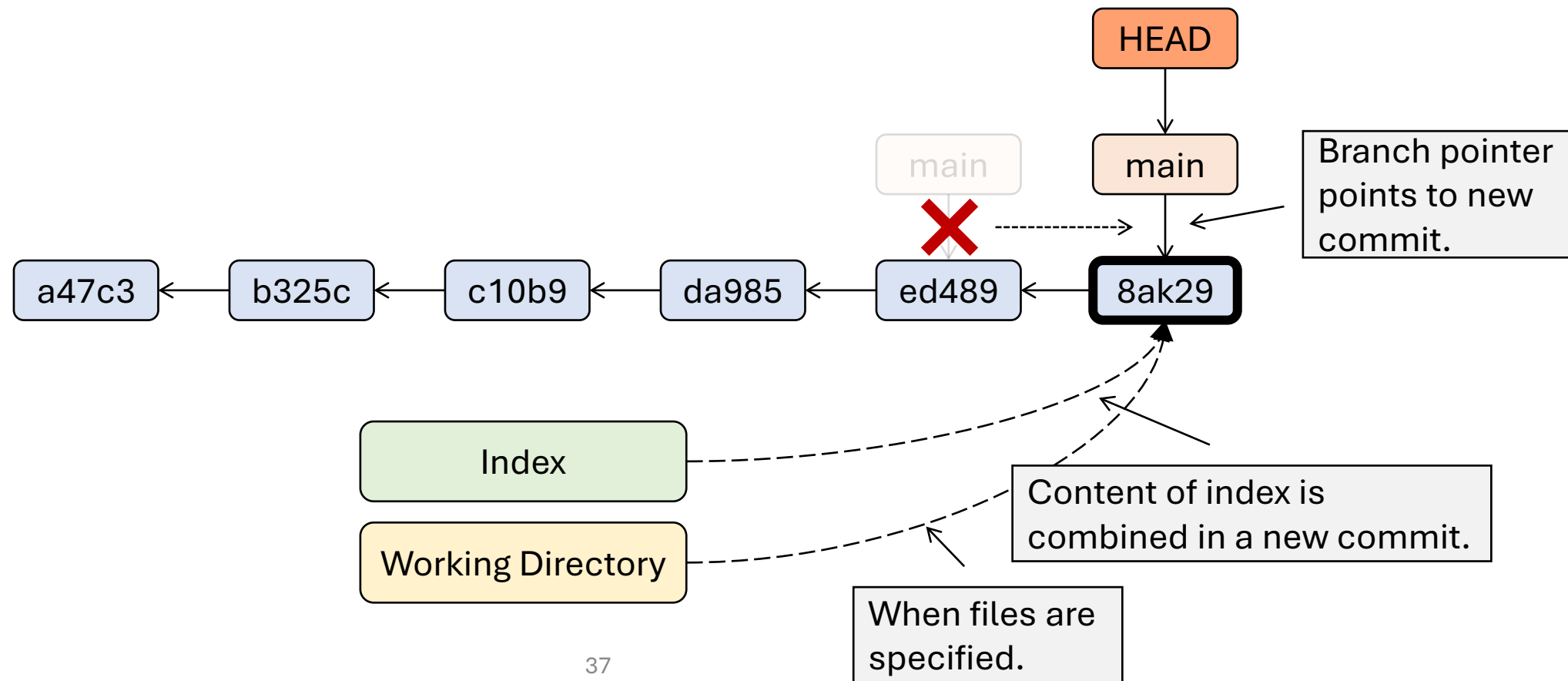
git add

Remember changes that are due to be stored in the repository in the next commit.



git commit

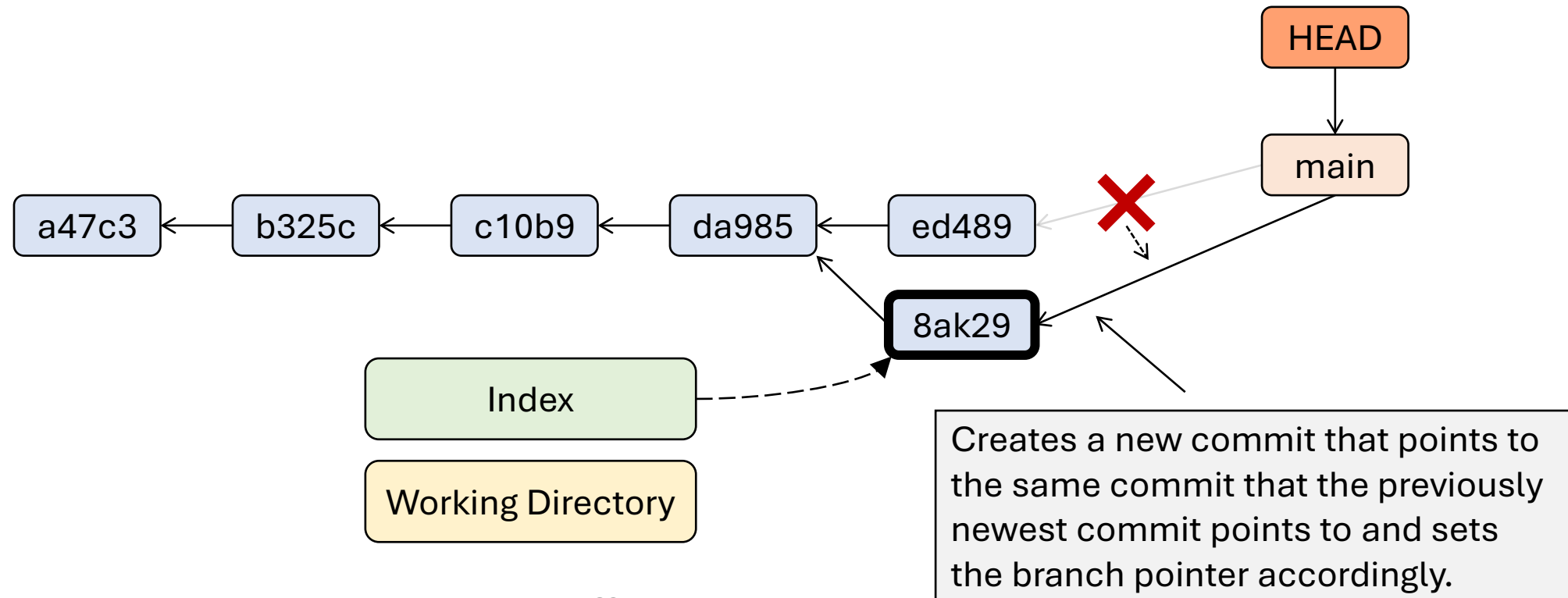
Creates a new commit object from the content of the index that points to the previous newest commit and sets the HEAD and branch pointer to this commit.



git commit -- amend

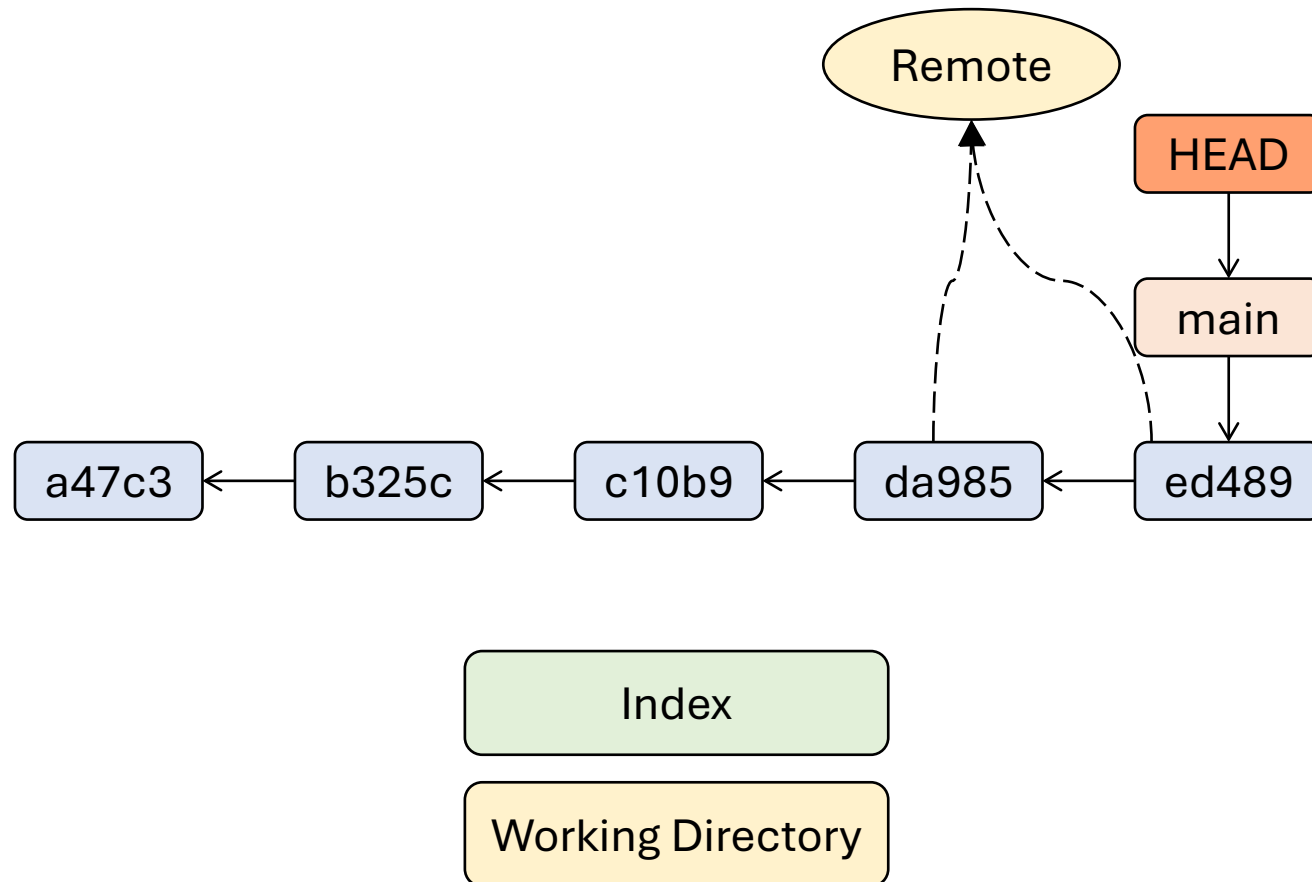
The `-- amend` option replaces the previous commit.

Intention: Committed too much, too few, the wrong things,...



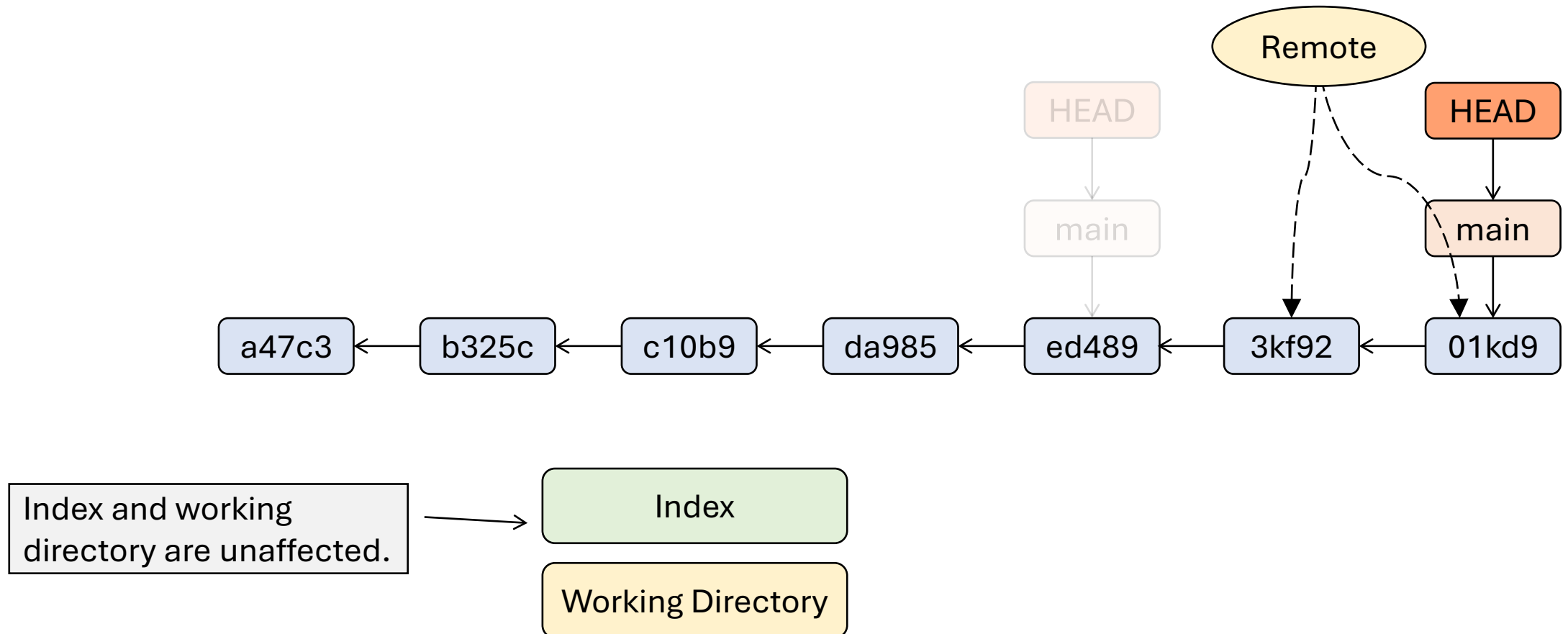
git push

Current state of local repository is loaded to the remote repository.



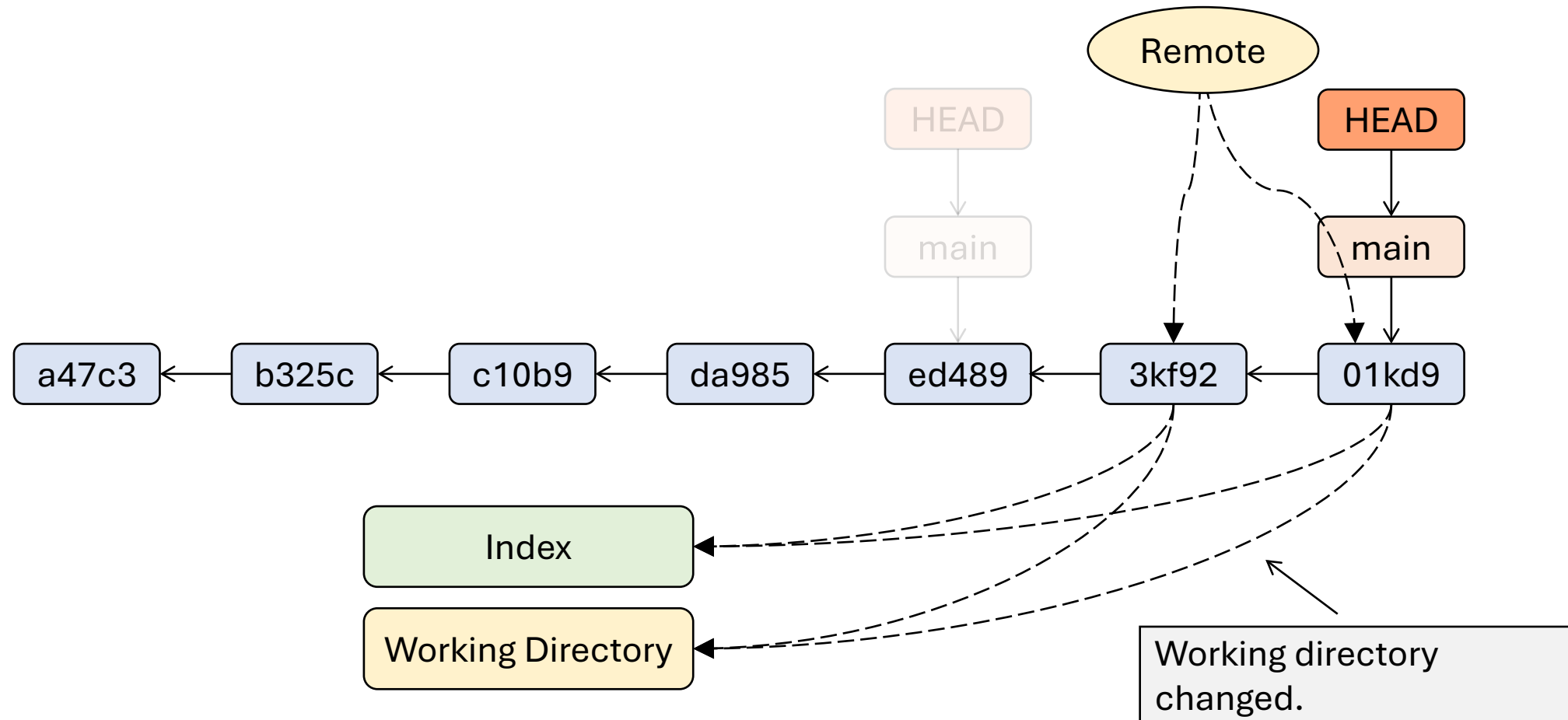
git fetch

Fetch loads the current state of the remote repository for all tracked branches and information about untracked branches.



git pull

Pull is a fetch + merge. The changes loaded from remote are also merged with the current state in the working directory and index.



Branch, Checkout, Merge

Typical process:

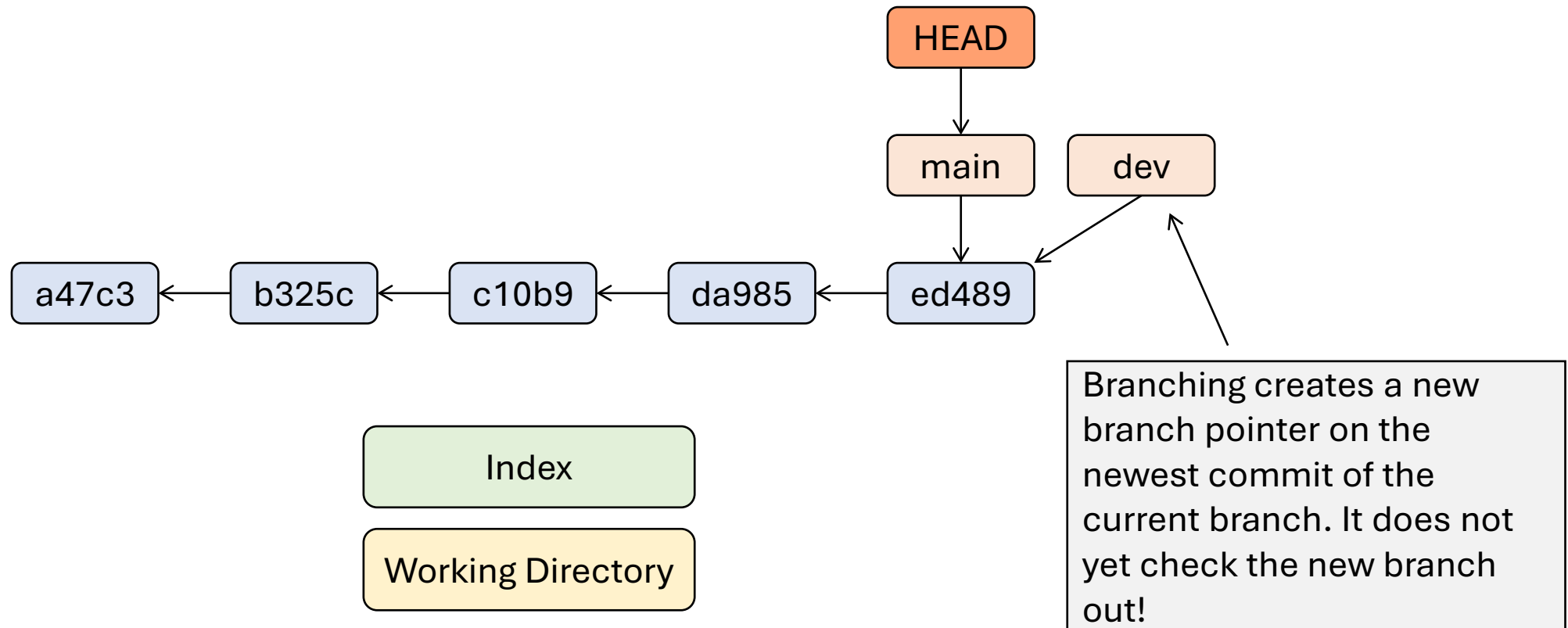
1. Create branch.
2. Switch to the branch.
3. Implement functionalities in multiple commits.
4. Merge back.

We can merge arbitrary branches. To do so,...

- Switch to target branch via git checkout
- Merge the source branch into it via git merge

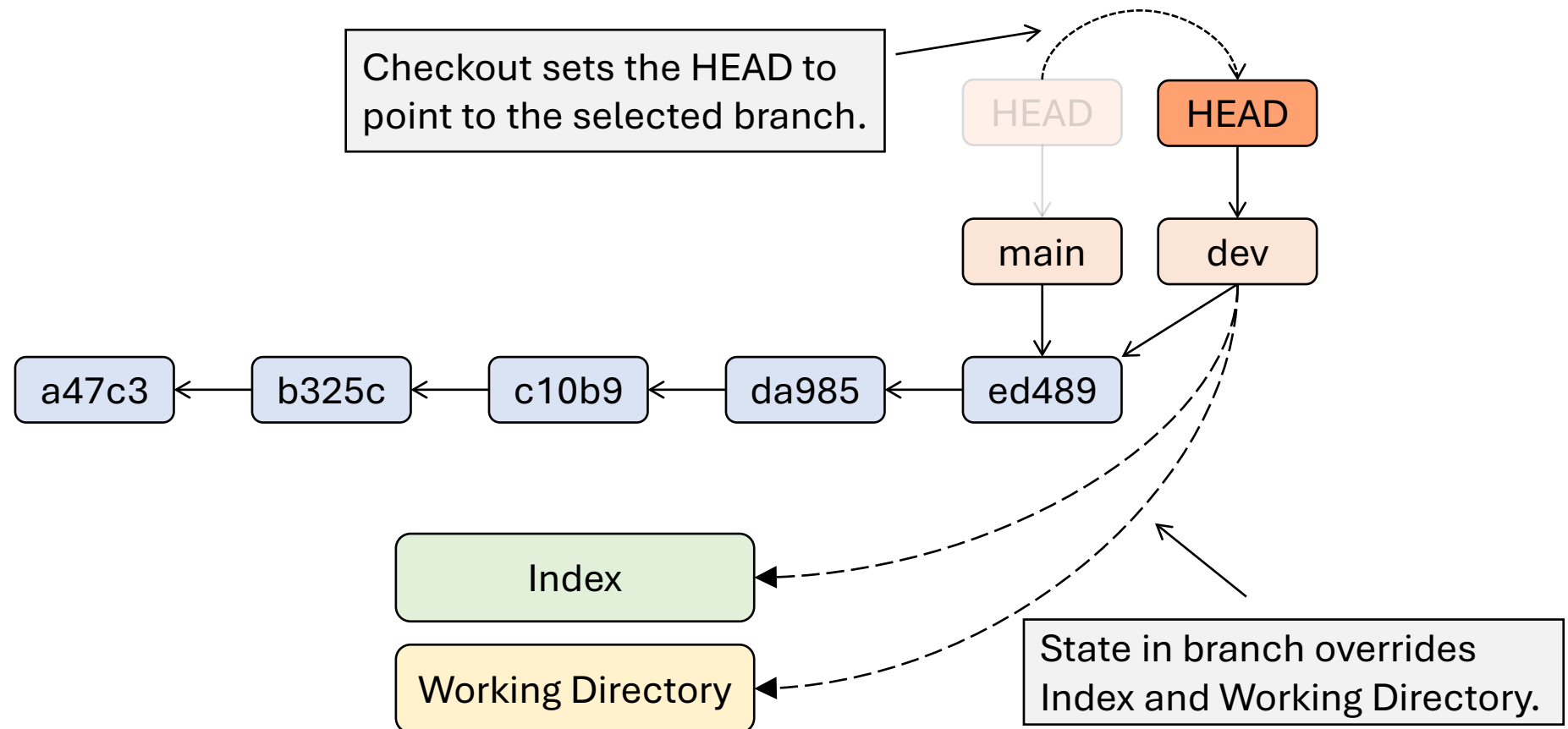
git branch

A branch is a sequence of commits. A branch is implemented as a pointer to the latest commit of this sequence.



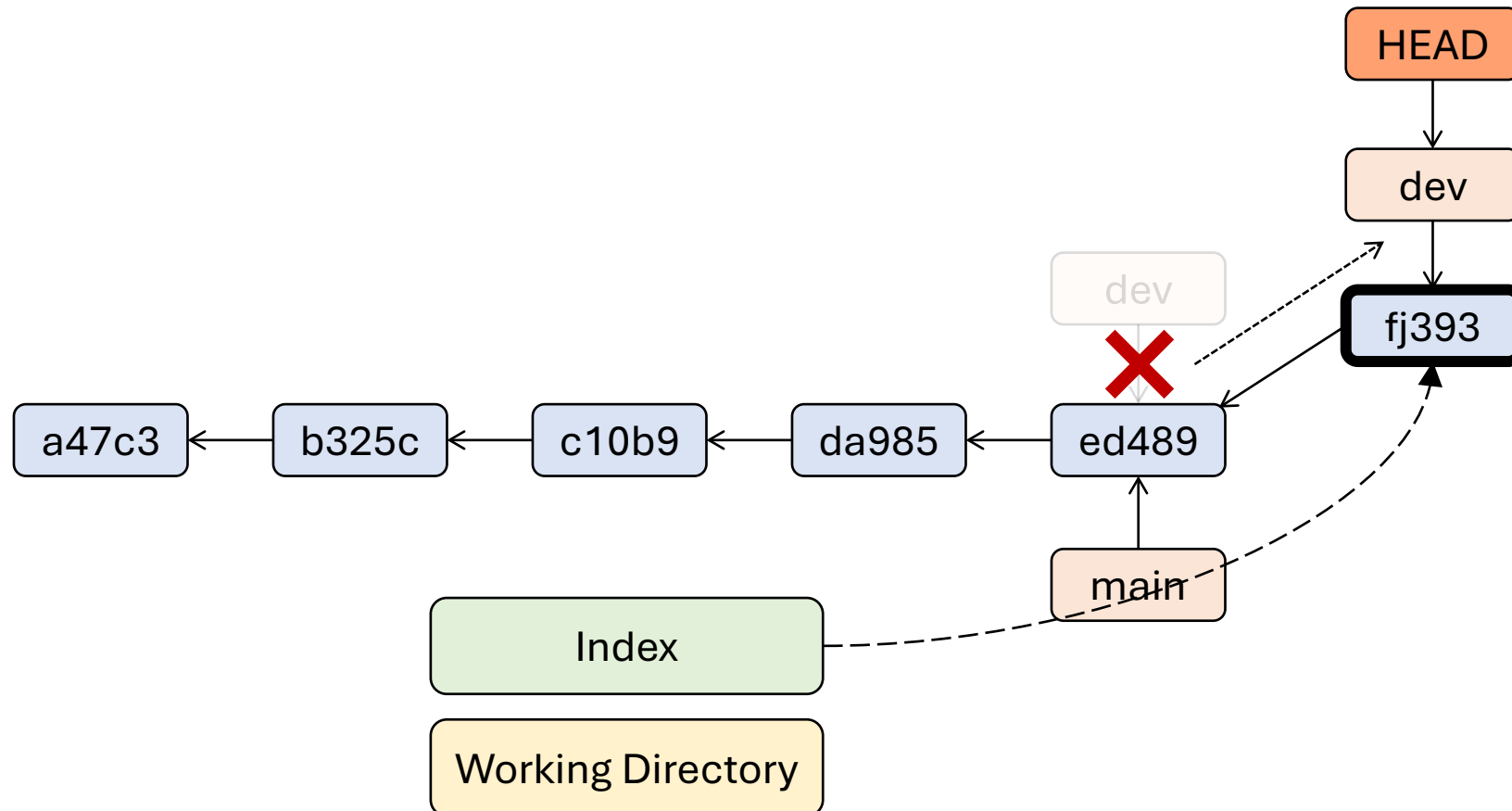
git checkout branch

Checking out a branch changes the HEAD to point to the desired branch.



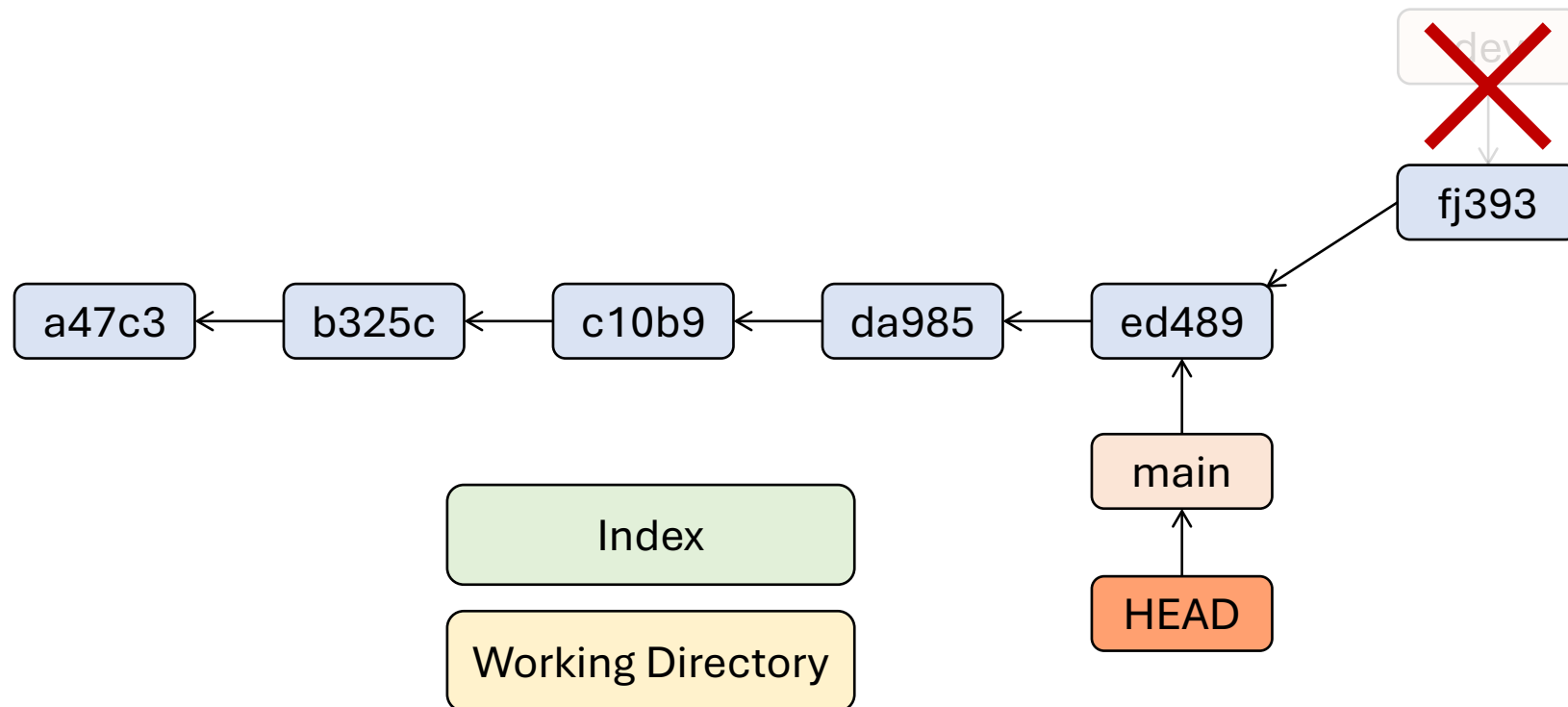
git commit on Branch

Committing on a branch works the usual way and leaves the former branch (that the current one branched off of) as is.



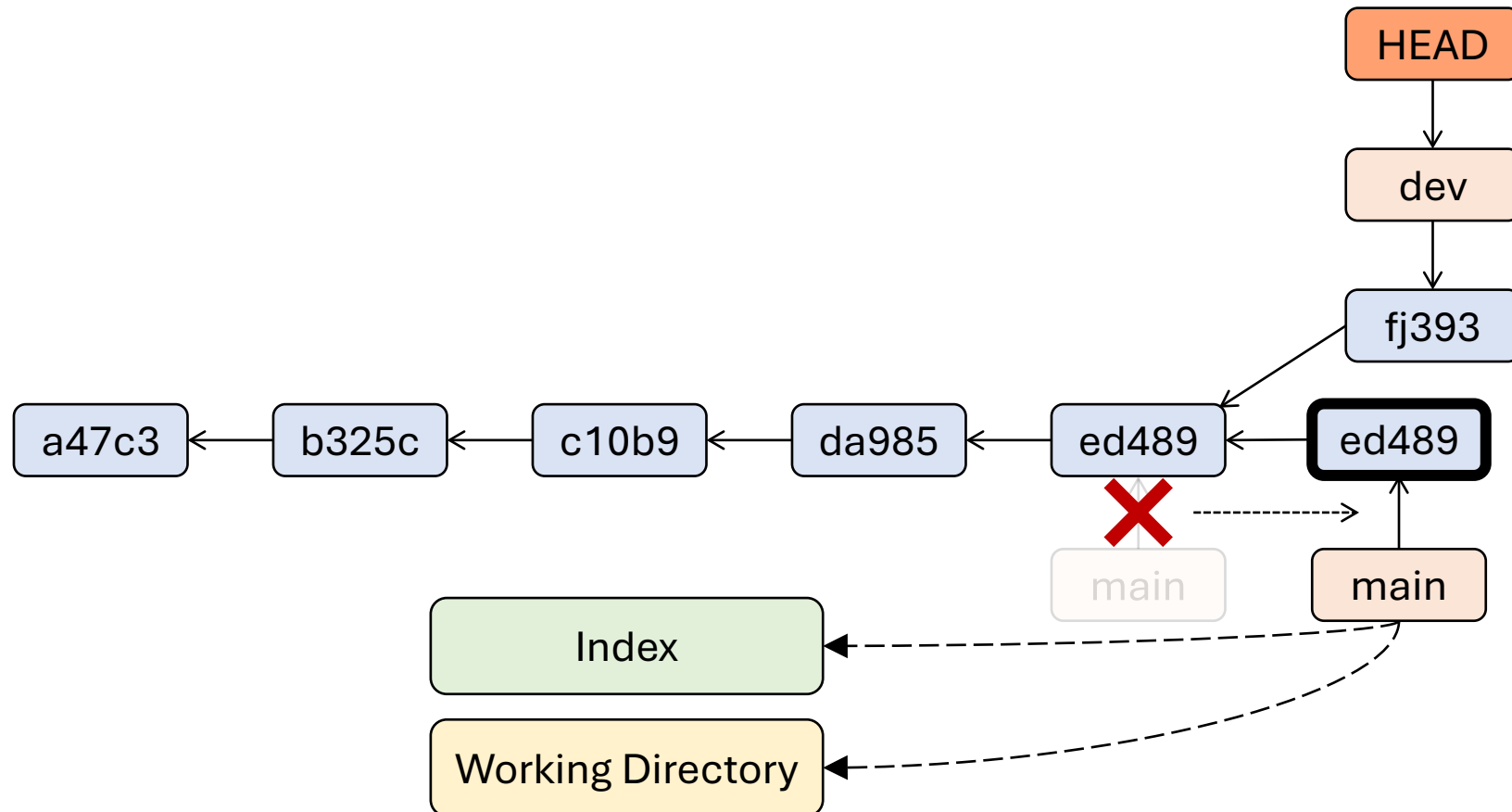
git branch -d

Deletes the specified branch.



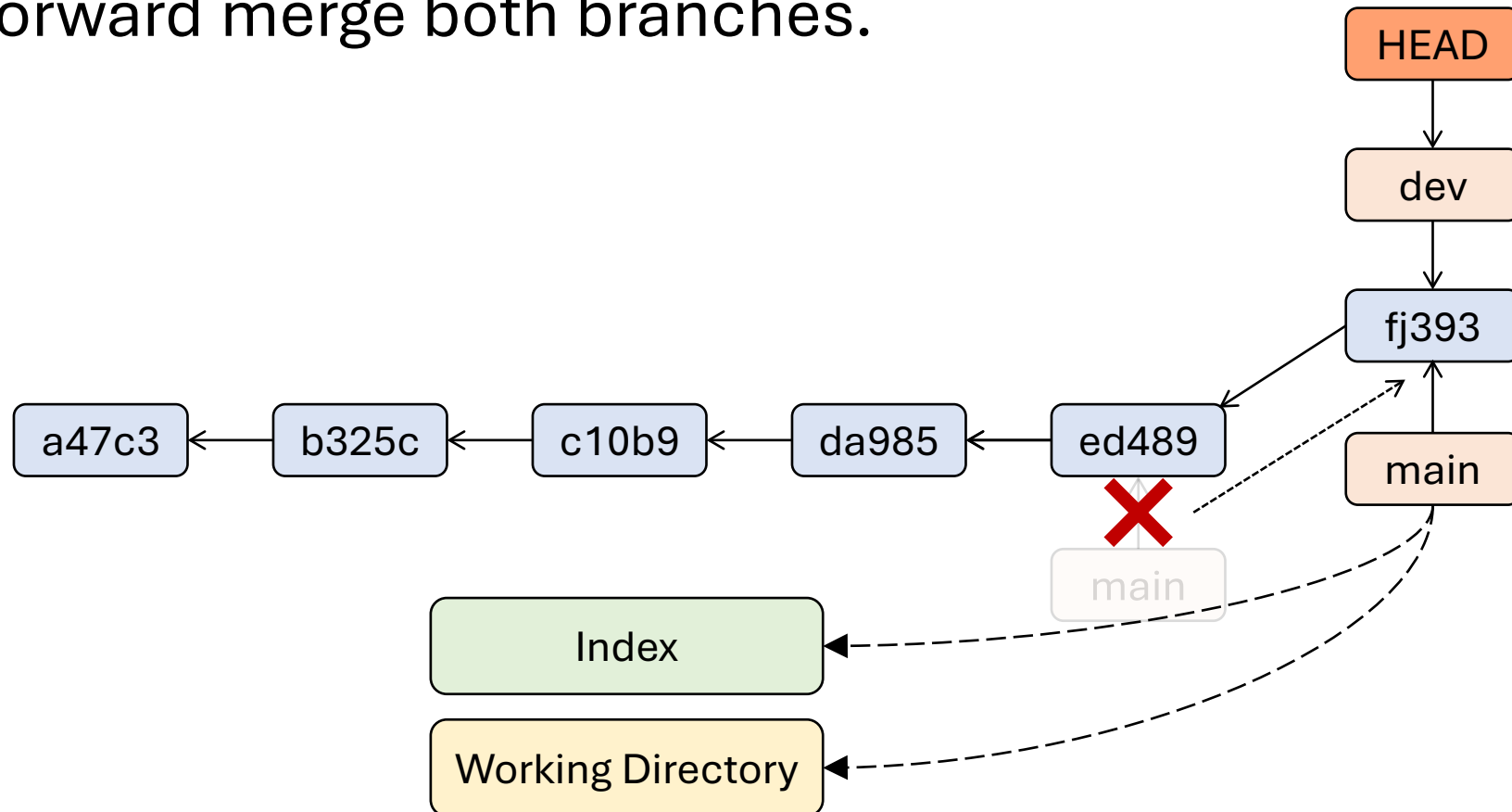
git commit on Branch

Committing on the former branch then leads to the parallel structure.



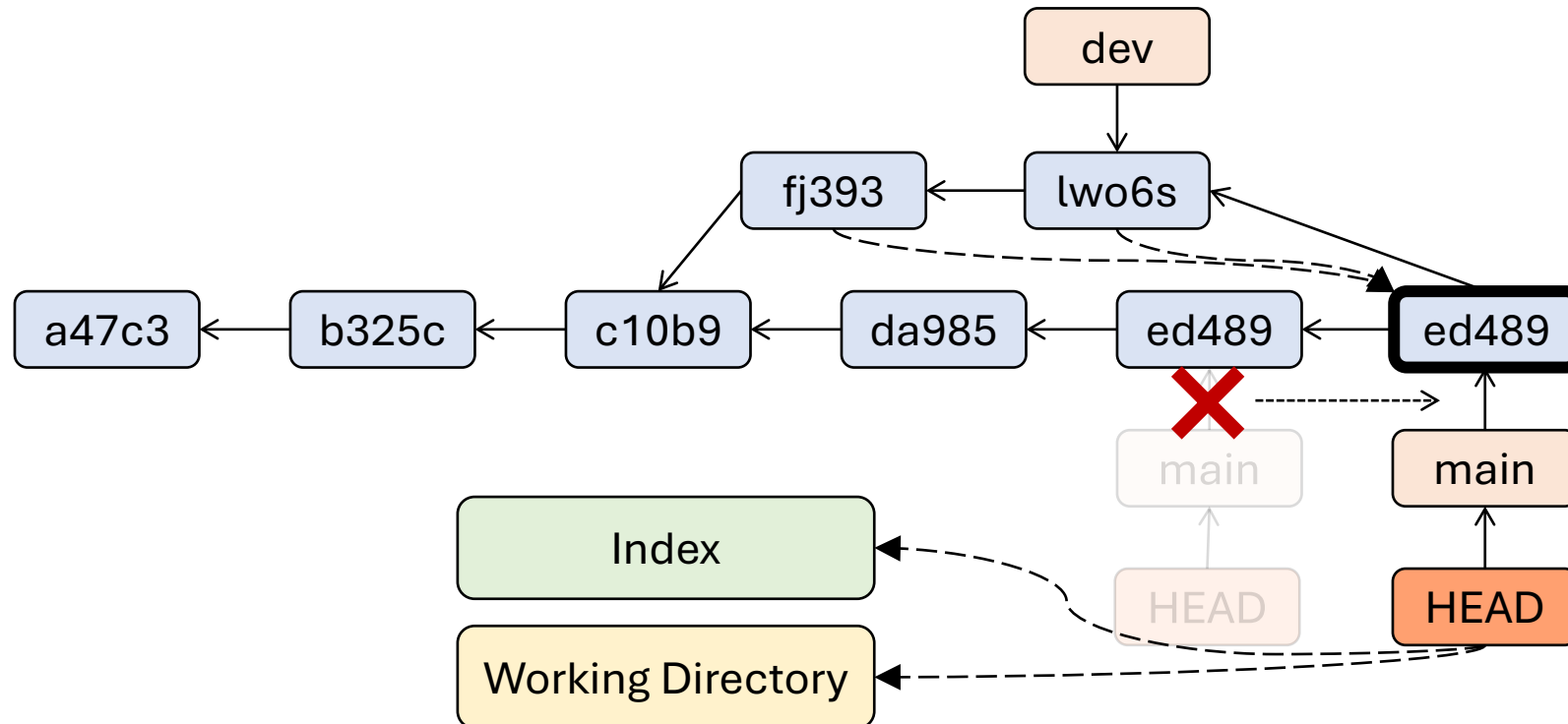
git merge Fast-Forward

If the target branch is a subset of the source branch (there is no newer commits on the target branch after the split point), then we can fast-forward merge both branches.



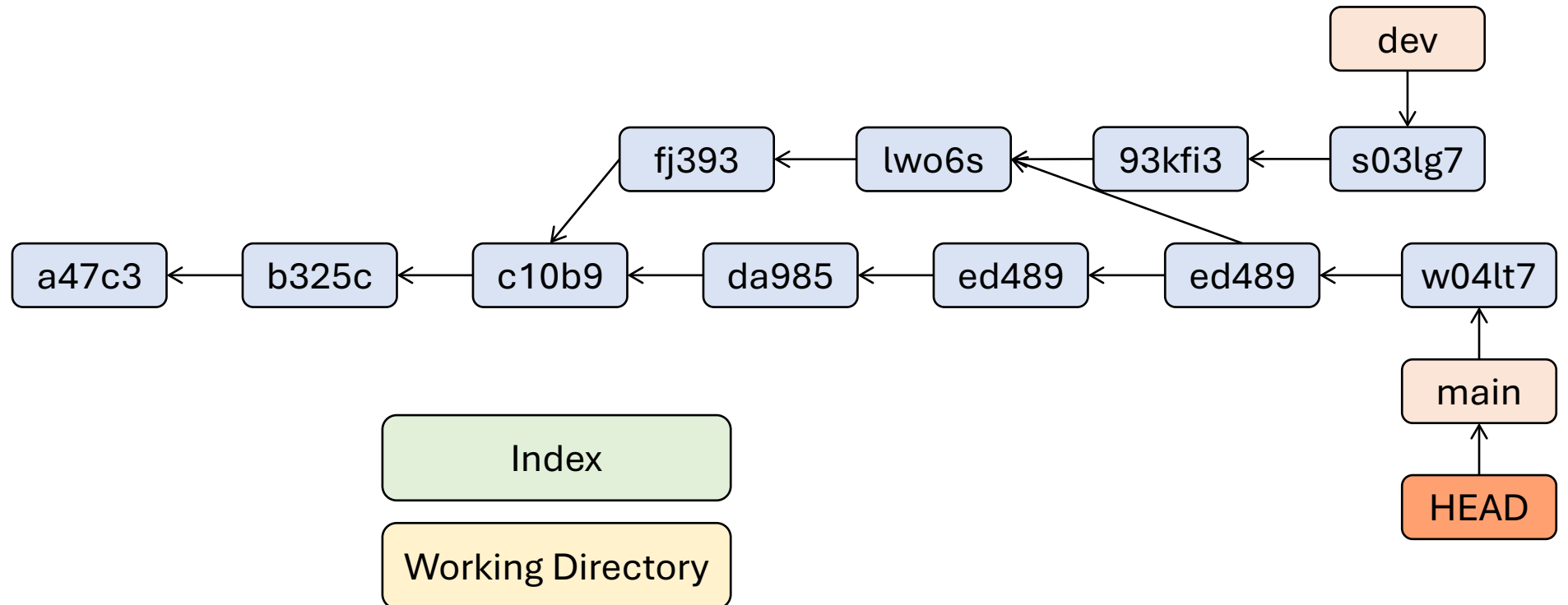
git merge

Merging two branches creates a merge commit in the current branch that point to two parent commits.



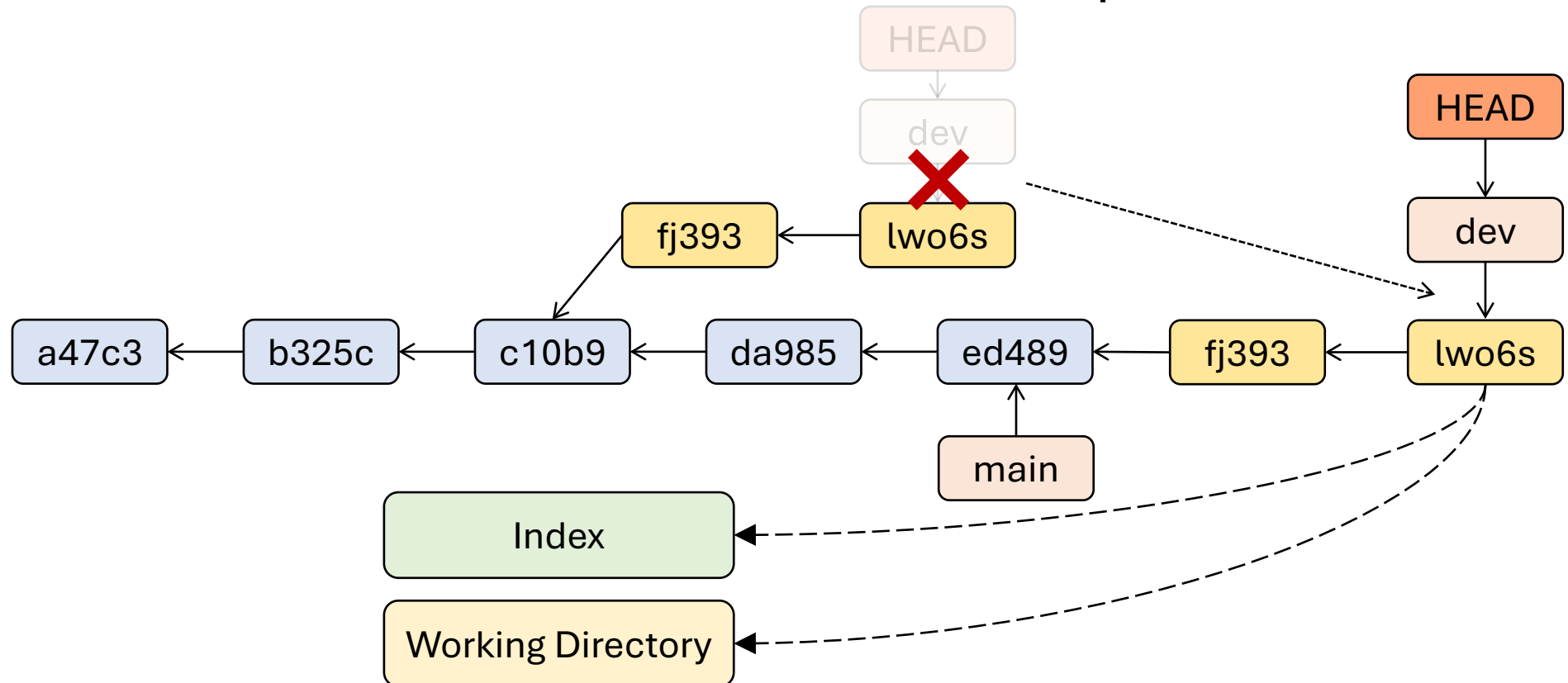
Branches co-exist after Merging

Merging a branch neither means that the branch does not exist anymore, nor does it move the original's pointer.



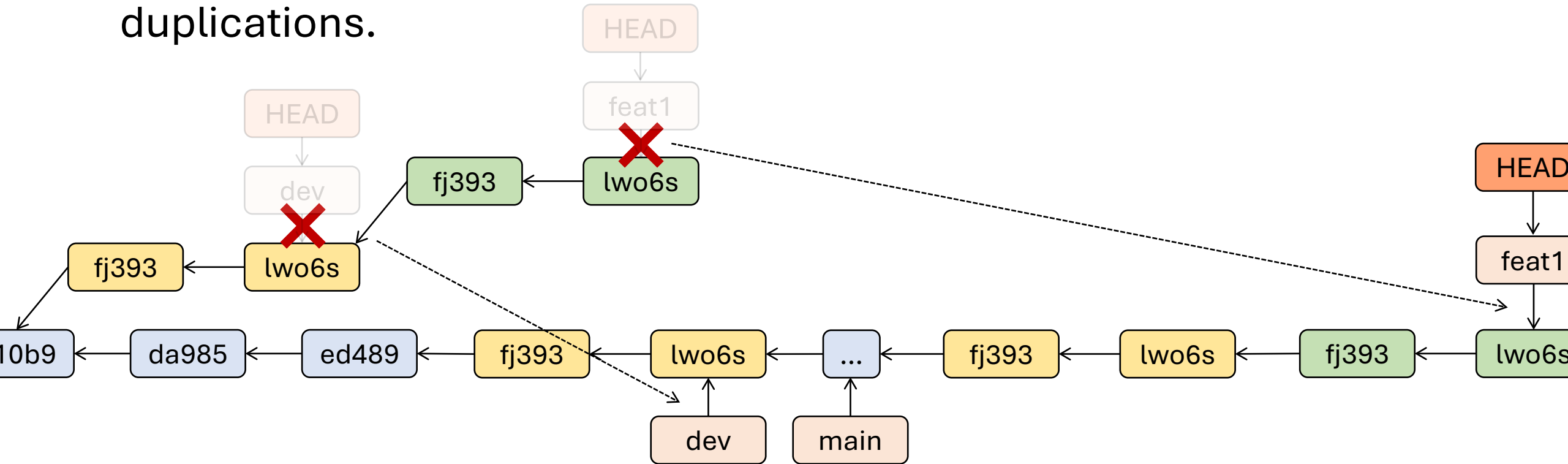
git rebase

Rebase, as an alternative to merge commits, does not create a new merge commit but append all commits of the source branch to the target branch one-on-one, and move the branches pointer.



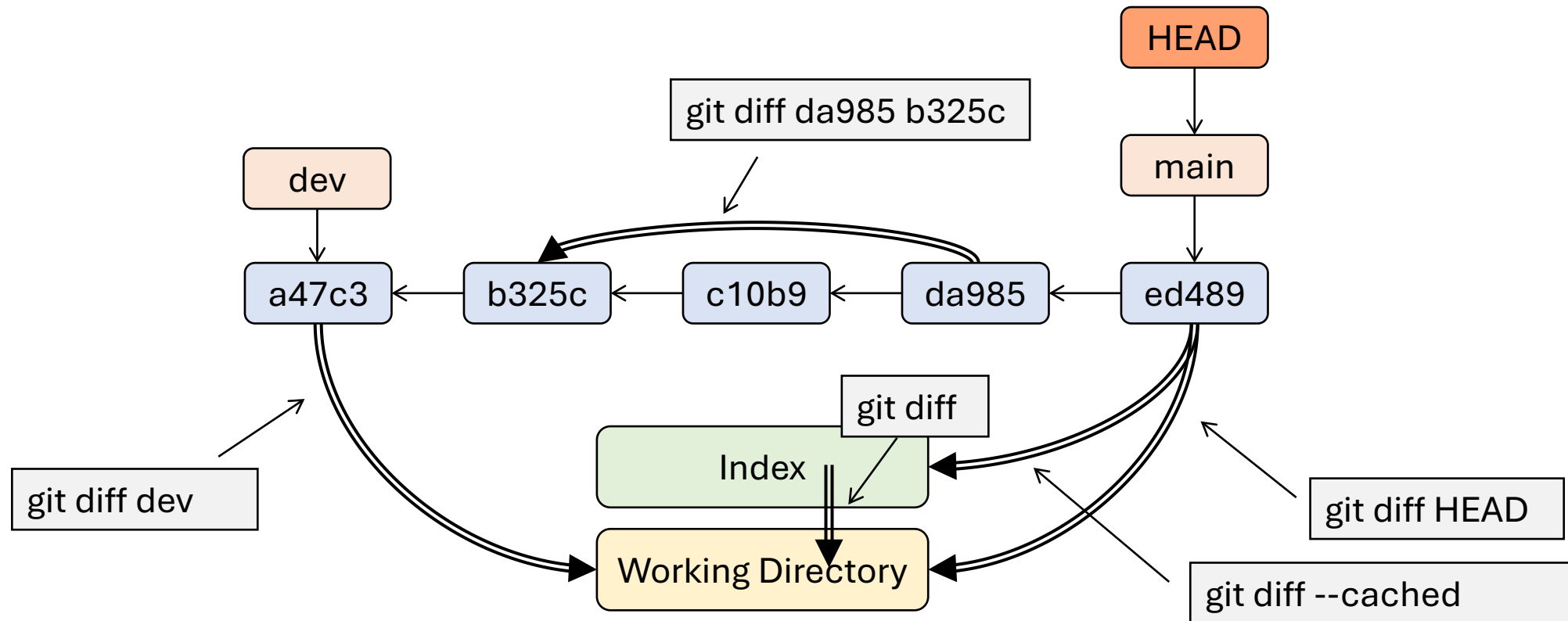
git rebase

Rebasing loses the information of parallel development and causes duplications.



git diff

git diff can be used to view differences.

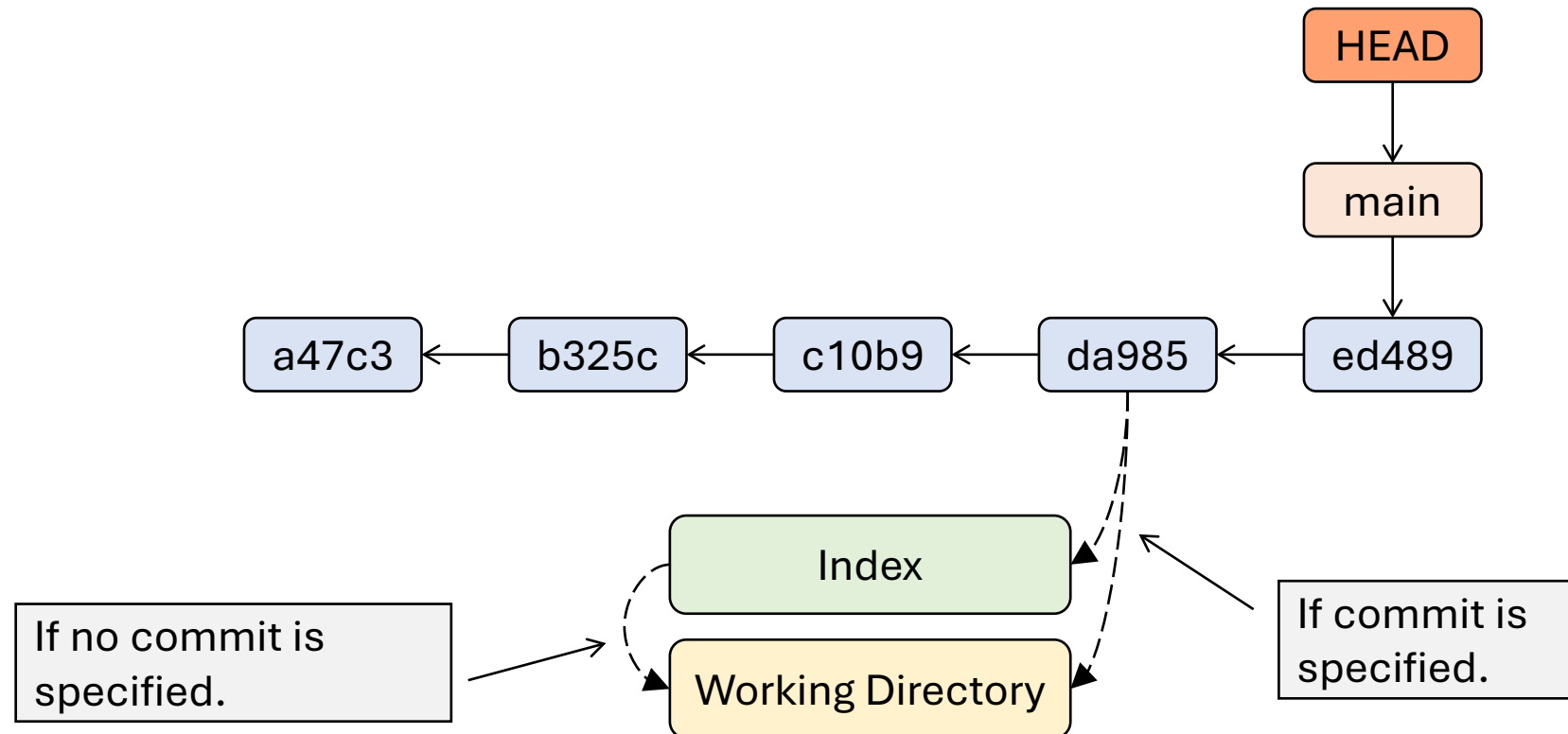


Kind of Undos in git

- Drop changes in working directory.
- Remove changes from index.
- Drop commits.
- Mixings of the former.

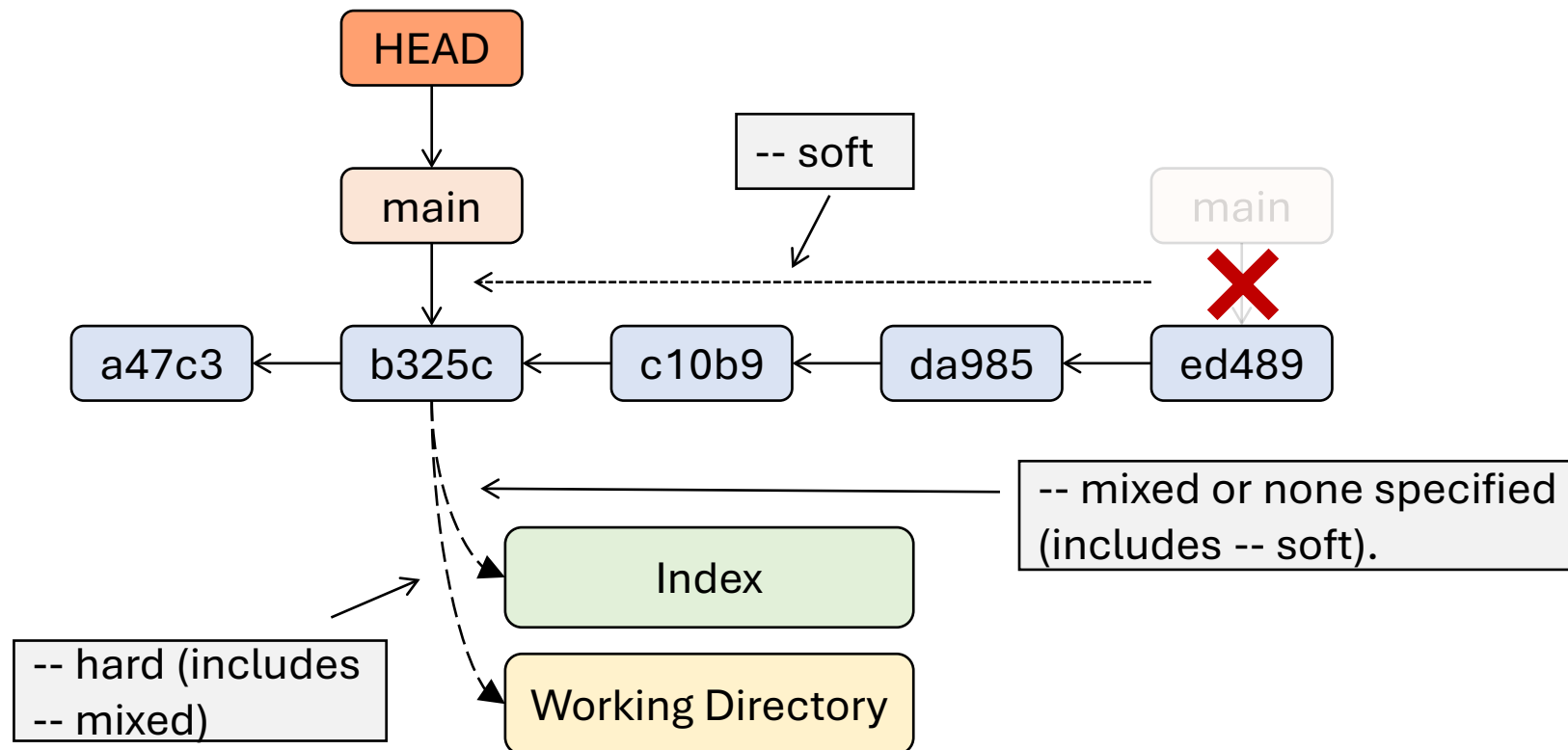
git checkout commit [paths]

Set content of working directory to content of specified commit.



git reset [--option] commit

Moves branch pointer and HEAD to specified commit.



Other Git Commands

Git status: list status of files in index and which are not tracked yet.

Git branch (nothing more): list branches.

Git log: show all commits in current branch.

Git remote add <alias> <path>: Add a remote repository.

Git stash: Save current modifications before changing branches

- Git stash pop: get top-most changes back.
- Git stash drop: Remove top-most changes.

*„It is easy to shoot your foot off with git,
but also easy to revert to a previous foot and merge it with your current leg.“*

- Jack William Bell