O'REILLY®
oscon

Deep Dive into Git

Edward Thomson
@ethomson

oscon.com
#oscon

“Talking about version control and **trying to make it interesting**... is like talking about sex and **trying to make it boring**.”

- Eric Sink, Version Control by Example

How Not to Learn Git

"It's easy!
It's just a directed acyclic graph!"

- Emma Jane Hogbin Westby
(on Anti-Patterns for Teaching Git)

Edward Thomson @ethomson

Software Engineer, **GitHub**
Maintainer, **libgit2** (www.libgit2.org)
Author, **Git for Visual Studio** (www.gitforvisualstudio.com)

Why care about how Git works?

Goals for Creating Git

Fast

Support a Distributed Workflow

Provide safeguards against accidental data corruption

Take the CVS project as an example of what *not* to do

If that doesn't fix it, git.txt contains the phone number of a friend of mine who understands git. Just wait through a few minutes of 'It's really pretty simple, just think of branches as...' and eventually you'll learn the commands that will fix everything.

xkcd: http://xkcd.com/1597/

Git Has Some Strange Commands

```
git rebase --onto origin/master HEAD~5 HEAD
git rerere forget foo/bar.c
git checkout branch
git checkout -b newbranch
git checkout -- foo/bar.c
git checkout origin/master changed_file.c
```

Git Has Some Strange Messages

```
foo.c: needs update

warning: LF will be replaced by CRLF

warning: CRLF will be replaced by LF

By default, updating the current branch in a non-
bare repository is denied

fatal: https://foo.com/info/refs not found: did you
run git update-server-info on the server?

You are in 'detached HEAD' state.
```

Git Has Some Strange Documentation

Forward-port local commits to the updated upstream head

Join two or more development histories together

Reveal all downstream commits below the sent upstream paths

Update remote refs along with associated objects

Git Has Some Strange Documentation

Git Man Page Generator:

http://git-man-page-generator.lokaltog.net

The Git commands are simply a leaky abstraction over the data storage.

What is a Distributed Version Control System

Open Source developers are geographically distributed

What is Distributed Version Control?

Not a classic “centralized” version control system

- Many operations must be performed against the server: get, revert, query history
- Even getting the status of your local working folders may talk to the server

What is Distributed Version Control?

Not a classic “centralized” version control system

- Many operations must be performed against the server: get, revert, query history
- Even getting the status of your local working folders may talk to the server

Distributed Version Control requires no central server

- Each user has an independent copy of the repository, including history
- Operations work against the local repository, including “commit”
- Users publish their changes independently of committing them
- Two independent changes must be merged to ensure logical consistency

The Git Repository

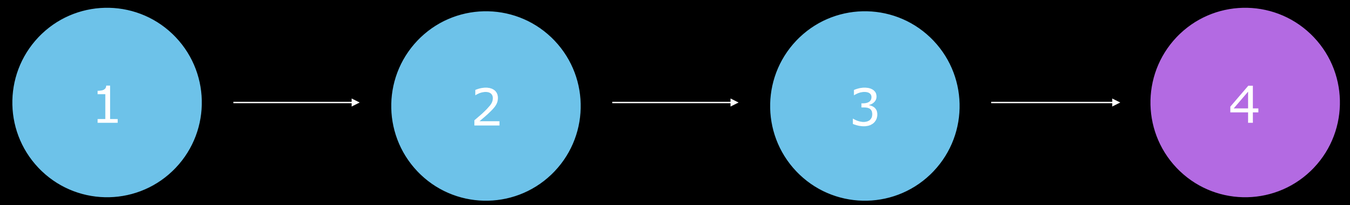
History

My repository

Alice's repository

History

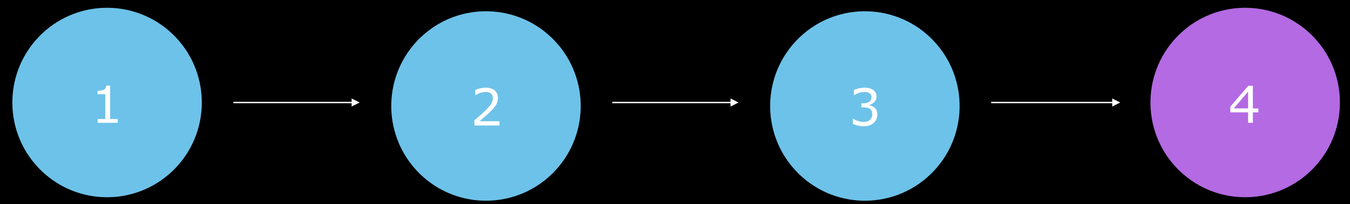
My repository



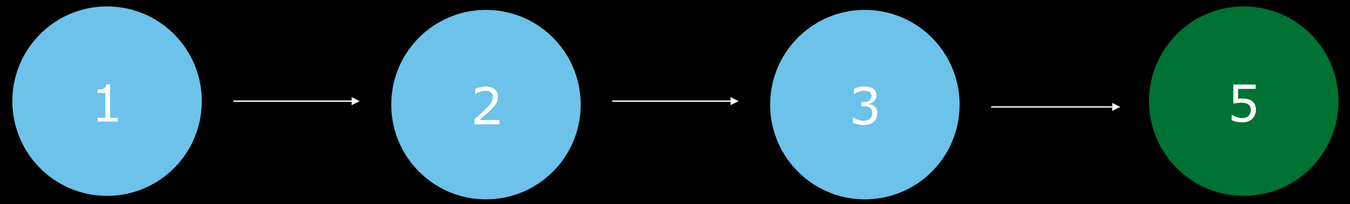
Alice's repository

History

My repository

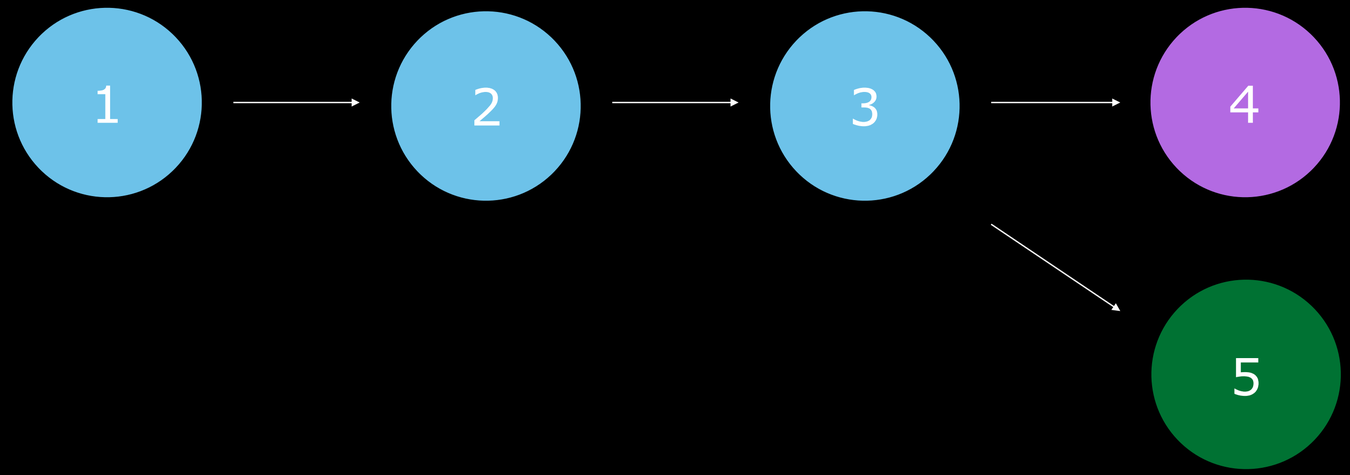


Alice's repository

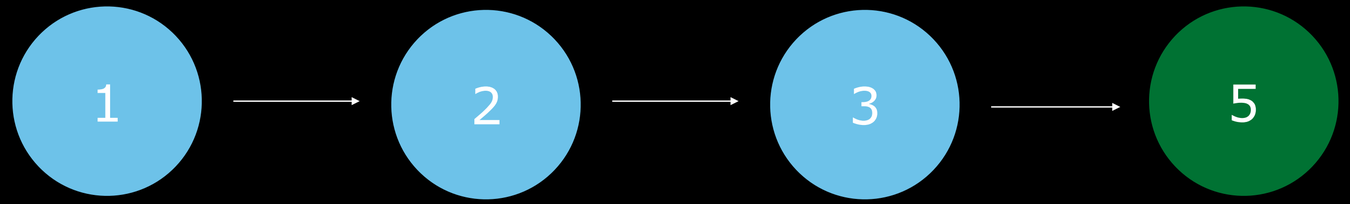


History

My repository

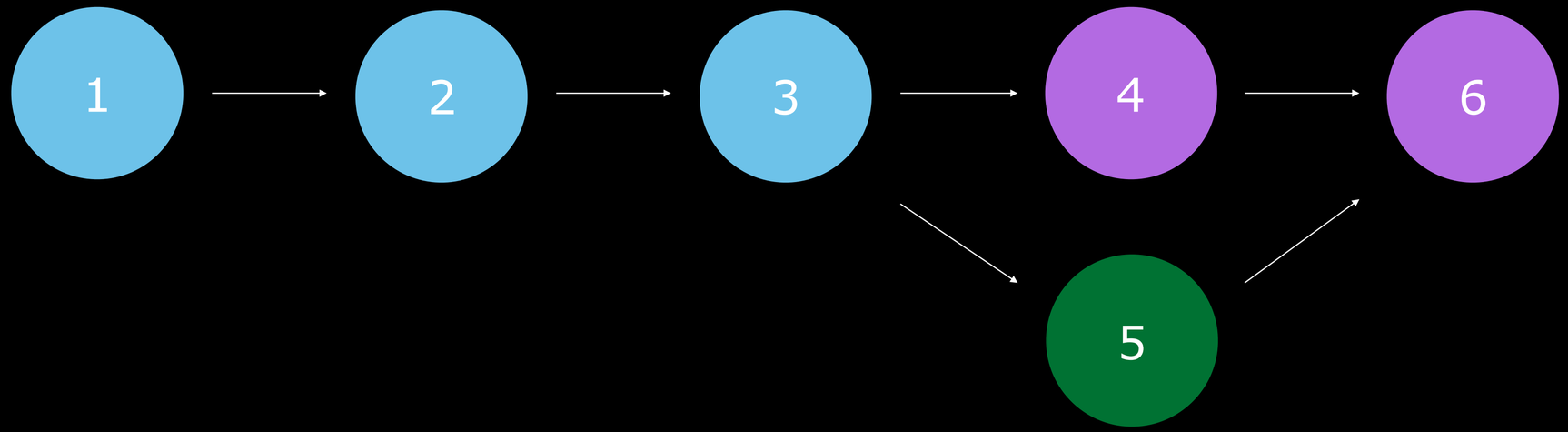


Alice's repository

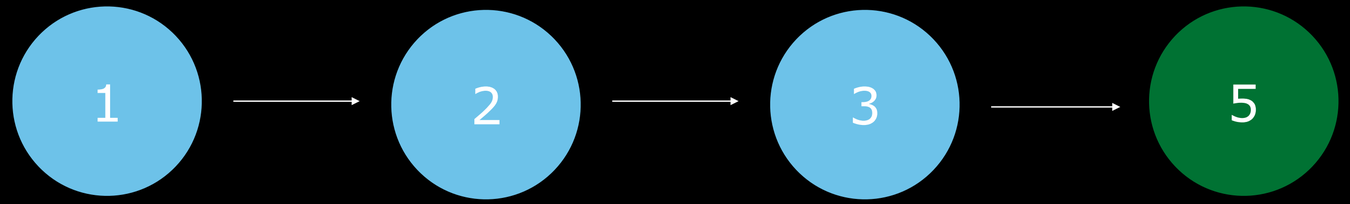


History

My repository



Alice's repository



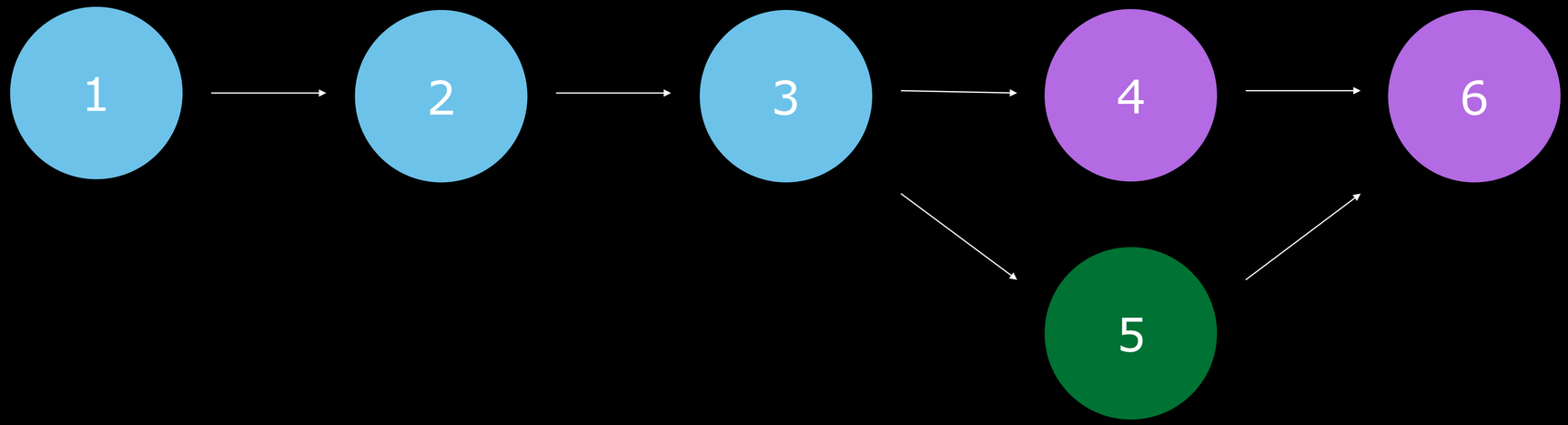
Demo

How History is Modeled

History

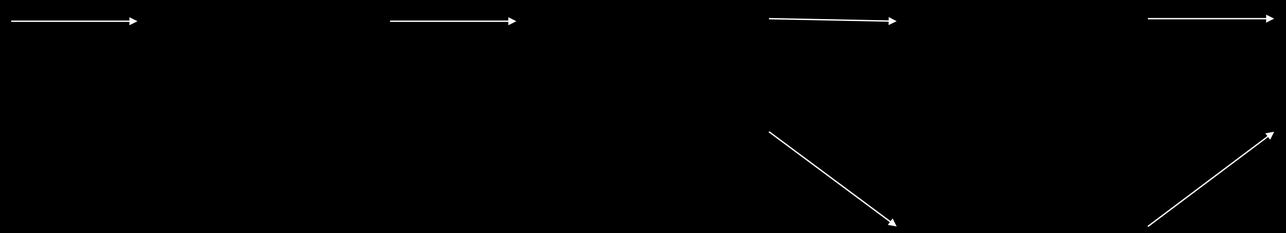
Git represents history in a Directed Acyclic Graph

Called the "DAG" or simple the "graph"



Git represents history in a Directed Acyclic Graph

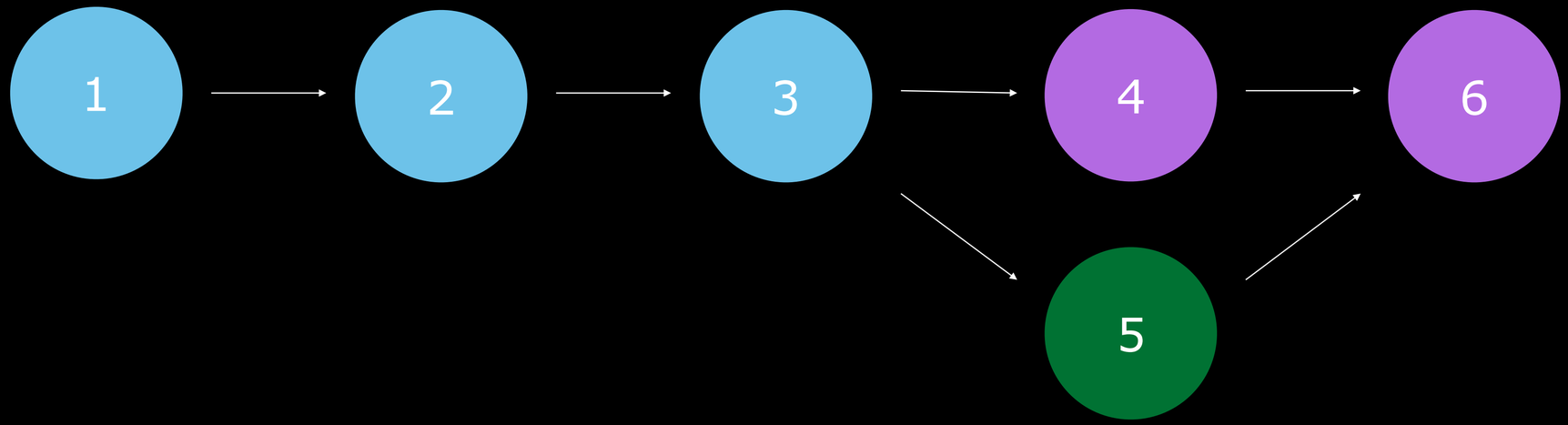
Called the “DAG” or simple the “graph”



History

Git represents history in a Directed Acyclic Graph

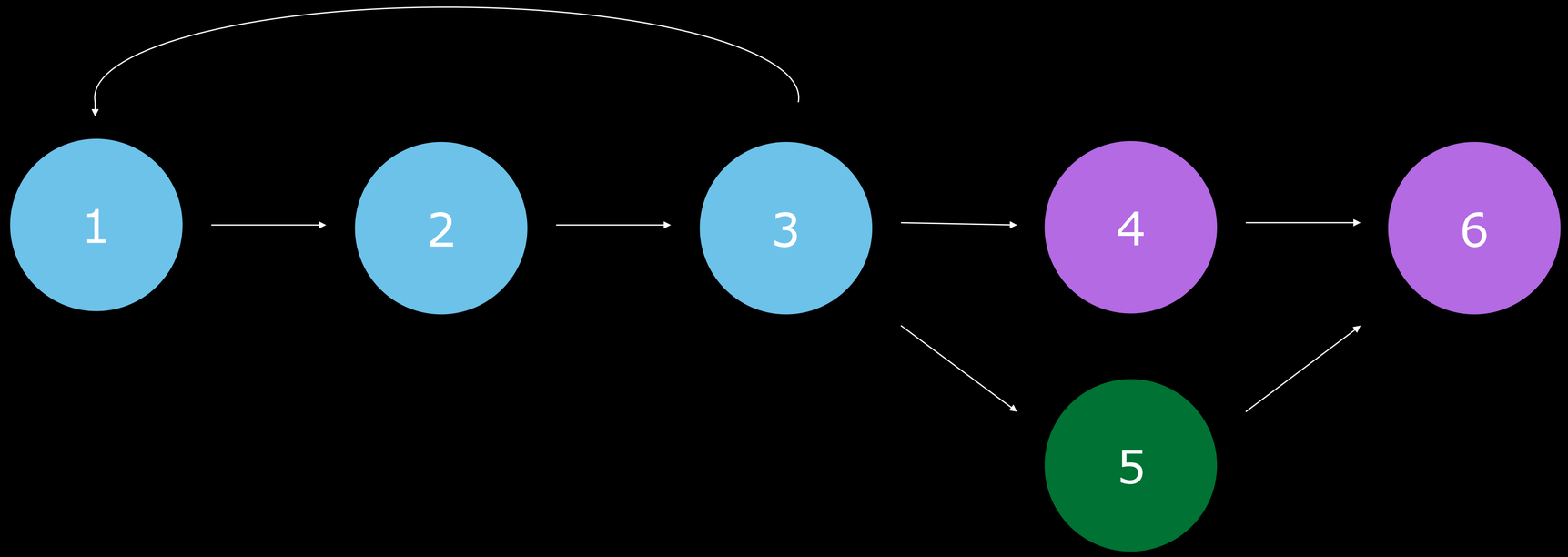
Called the "DAG" or simple the "graph"



History

Git represents history in a Directed Acyclic Graph

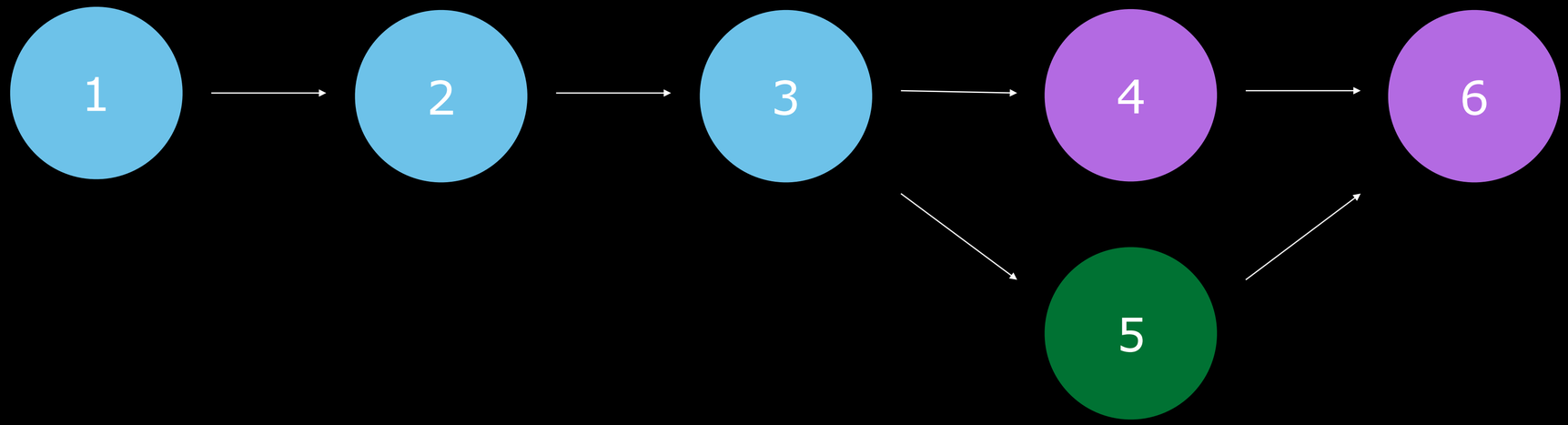
Called the "DAG" or simple the "graph"



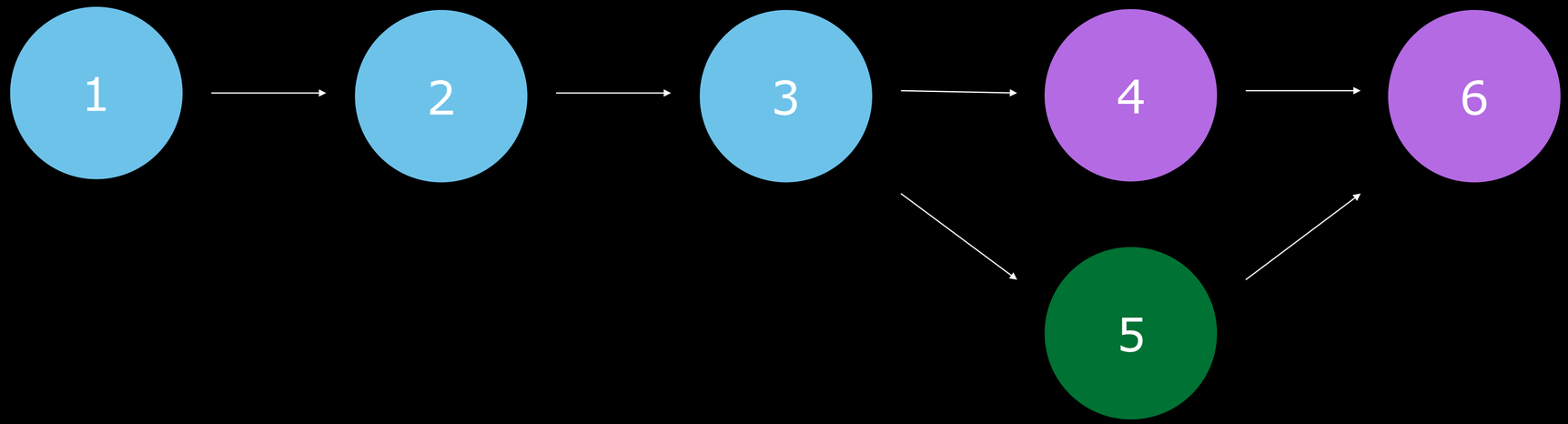
History

Git represents history in a Directed Acyclic Graph

Called the "DAG" or simple the "graph"



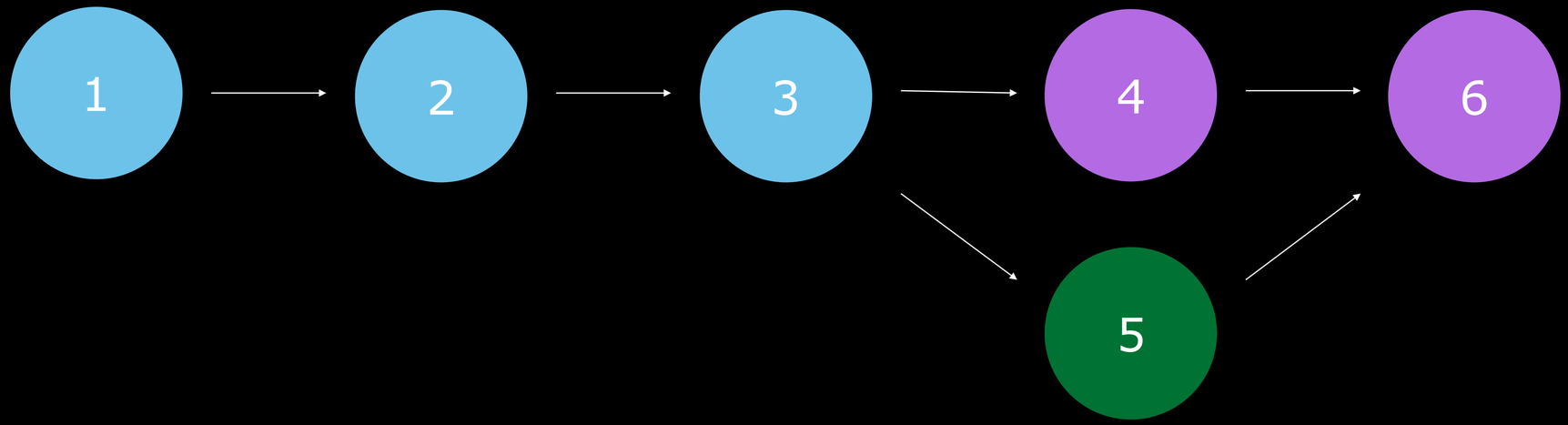
How are commit IDs generated?



History

How are commit IDs generated?

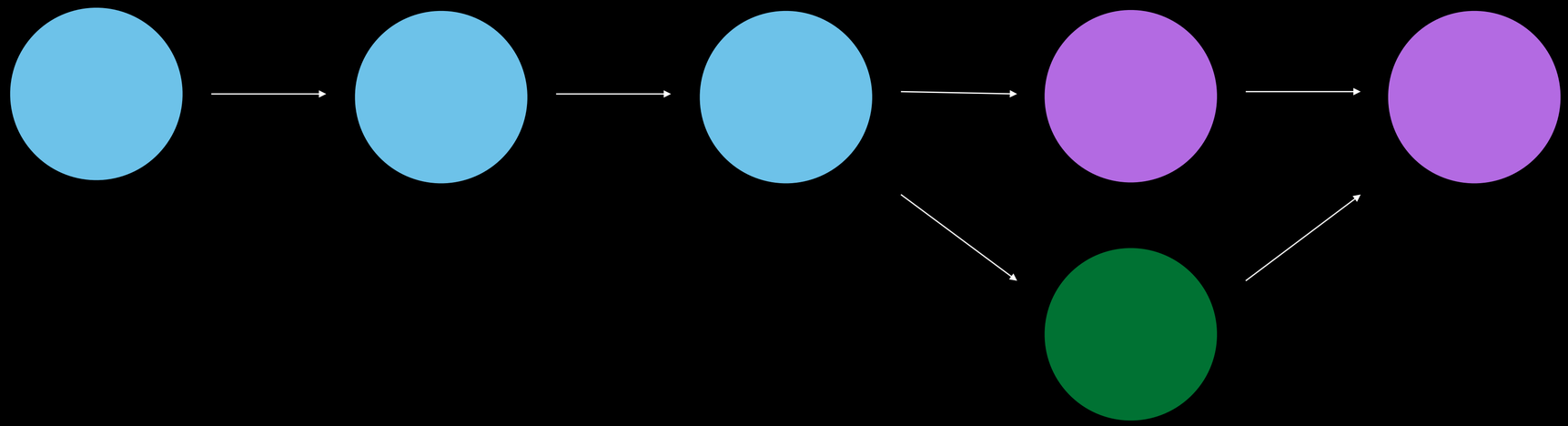
- Can't be monotonically increasing (without a server to arbitrate)



History

How are commit IDs generated?

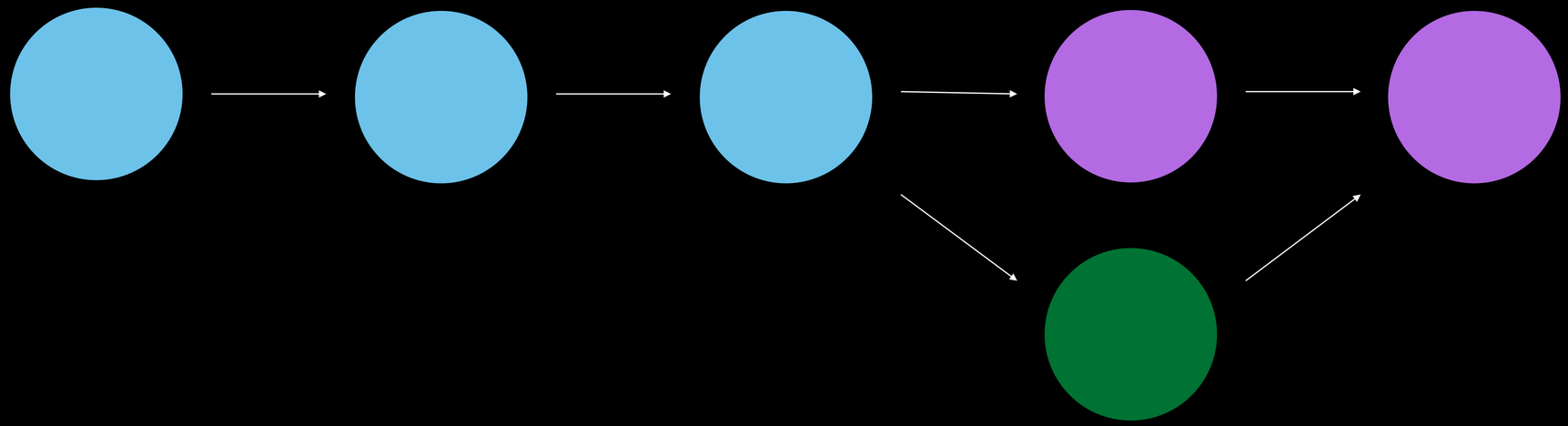
- Can't be monotonically increasing (without a server to arbitrate)



History

How are commit IDs generated?

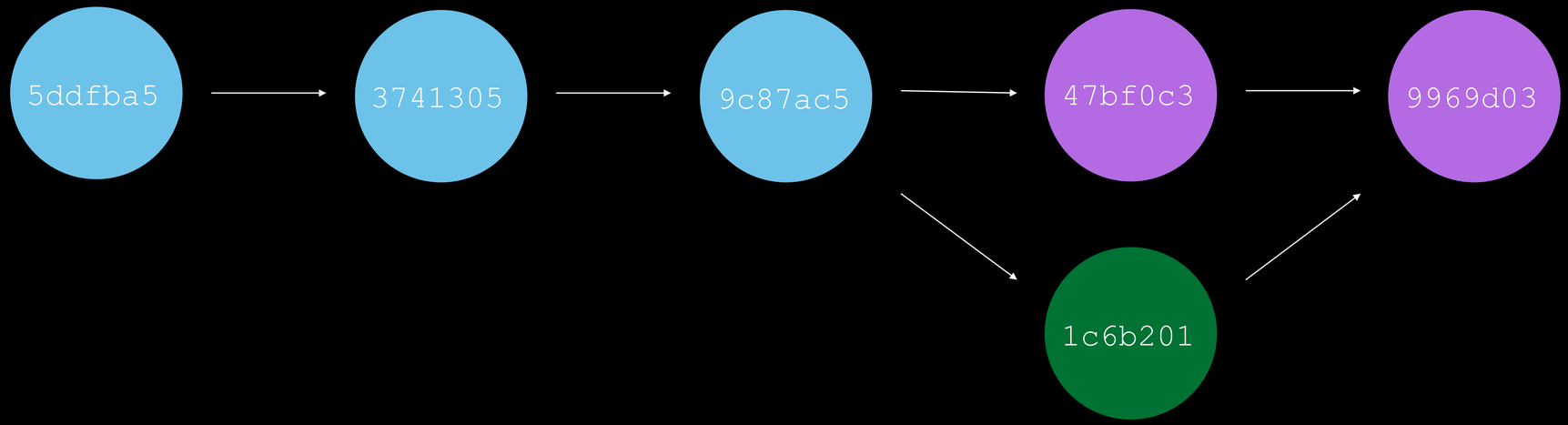
- Can't be monotonically increasing (without a server to arbitrate)
- SHA1 hash of the contents of the commit



History

How are commit IDs generated?

- Can't be monotonically increasing (without a server to arbitrate)
- SHA1 hash of the contents of the commit



Commits

Represent the entire repository

- Contain a pointer to the repository tree at that particular version
- To see what changed from the previous commit, you must run a *diff*

Commits

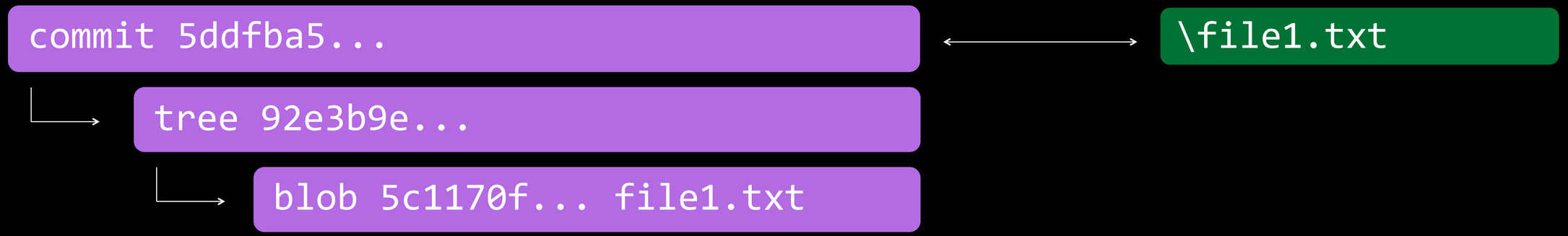
Represent the entire repository

- Contain a pointer to the repository tree at that particular version
- To see what changed from the previous commit, you must run a *diff*

Created from **trees** and **blobs**

- Trees represent a directory, blobs represent a file
- Together can represent an entire filesystem

Commits


commit 5ddfba5...
\file1.txt
tree 92e3b9e...
blob 5c1170f... file1.txt

Commits


commit 5ddfba5...
tree 92e3b9e...
blob 5c1170f... file1.txt
\file1.txt
commit 3741305...
tree f2a25c9...
blob 5c1170f... file1.txt
blob 3eac351... file2.txt
\file1.txt
\file2.txt

Commits


commit 5ddfba5...
tree 92e3b9e...
blob 5c1170f... file1.txt
\file1.txt
commit 3741305...
tree f2a25c9...
blob 5c1170f... file1.txt
blob 3eac351... file2.txt
\file1.txt
\file2.txt

Commits


commit 5ddfba5...
\file1.txt
tree 92e3b9e...
blob 5c1170f... file1.txt
commit 3741305...
\file1.txt
\file2.txt
tree f2a25c9...
blob 5c1170f... file1.txt
blob 3eac351... file2.txt

Demo

How Objects are Stored

Branches

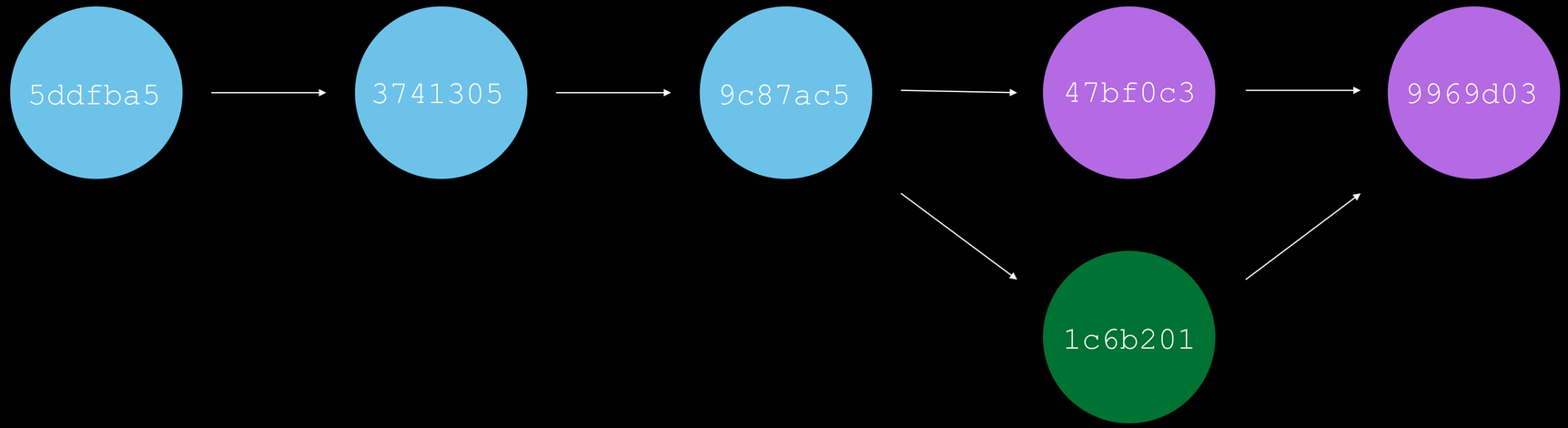
Exist at the repository level

- A branch applies to the entire repository
- Unlike (most) centralized tools where branches exist inside the repository

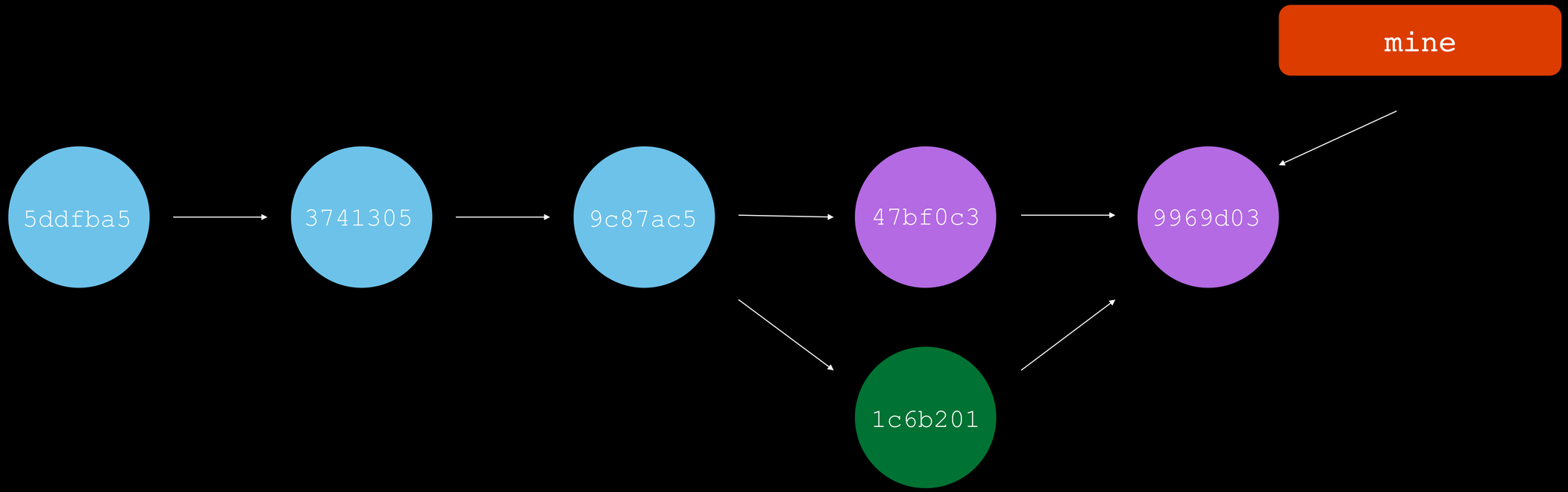
Exceptionally lightweight

- Implemented as a pointer to a commit in the graph
- Exist only in the local repository until they're explicitly shared
- Encourages feature branches

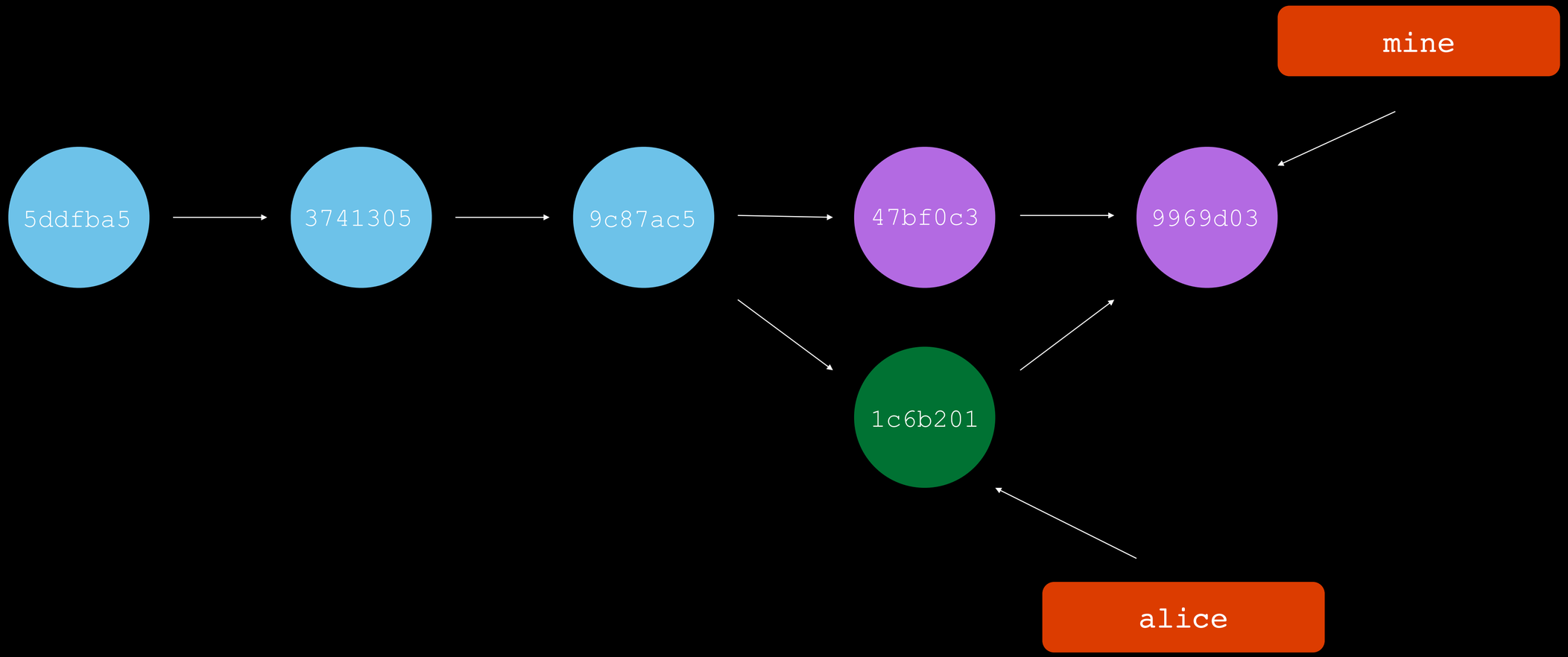
Branches



Branches



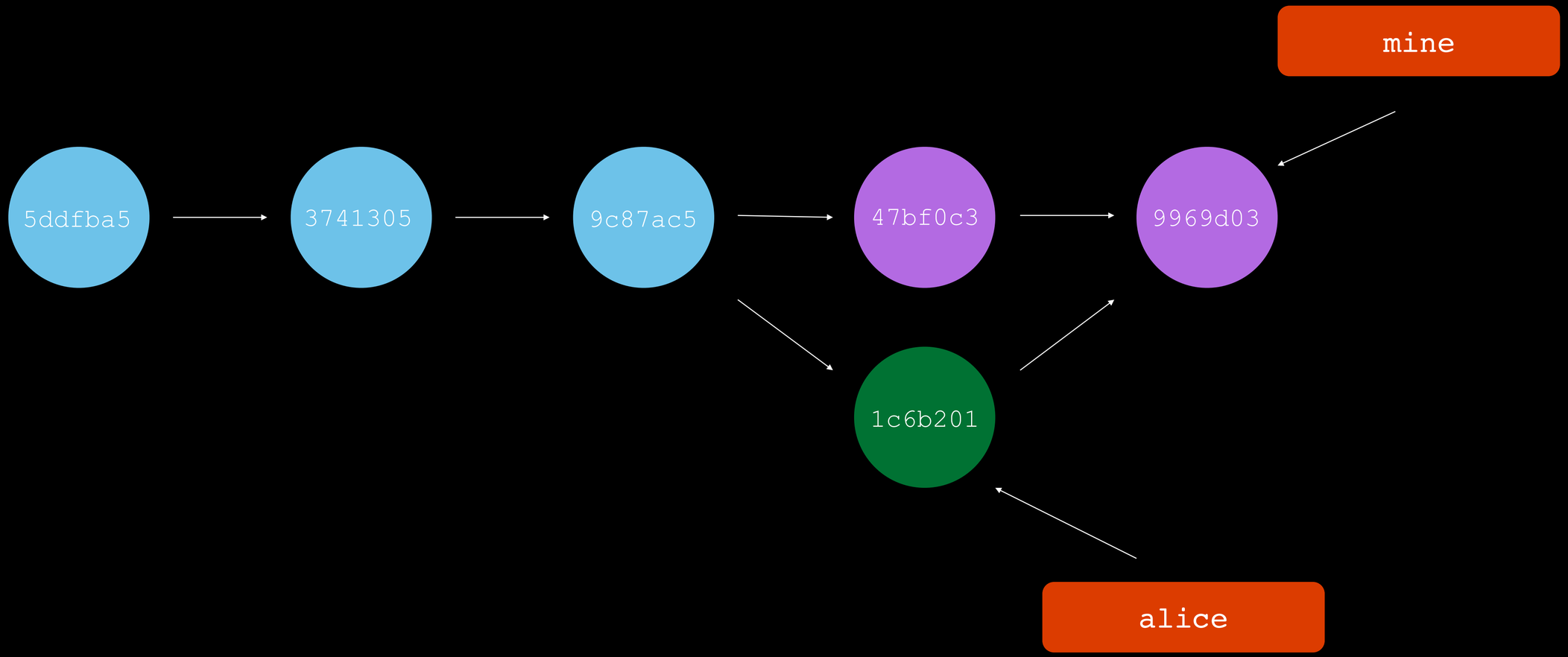
Branches


mine
5ddfba5
3741305
9c87ac5
47bf0c3
9969d03
1c6b201
alice

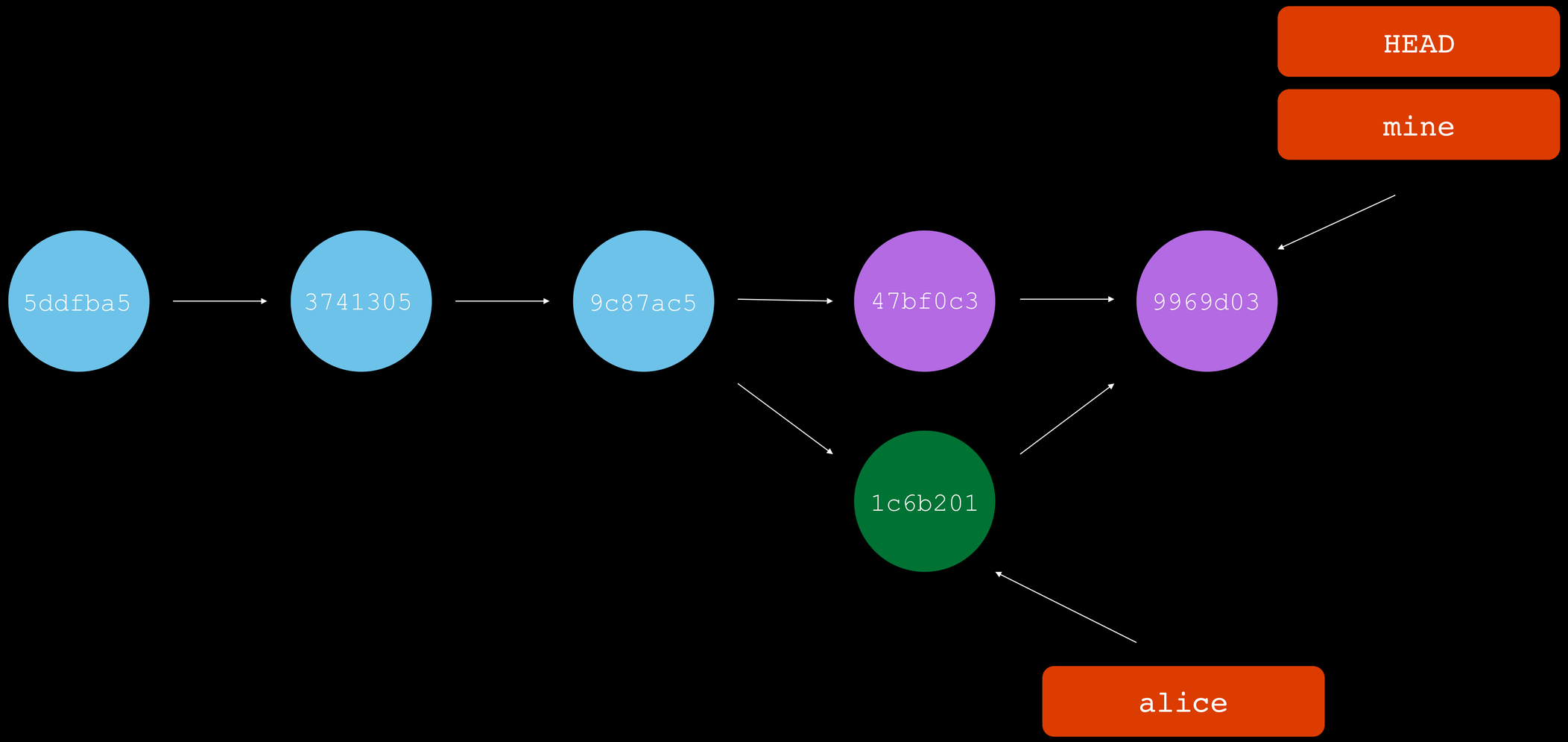
Demo

How Branches are Stored

Branches


mine
5ddfba5
3741305
9c87ac5
47bf0c3
9969d03
1c6b201
alice

Branches

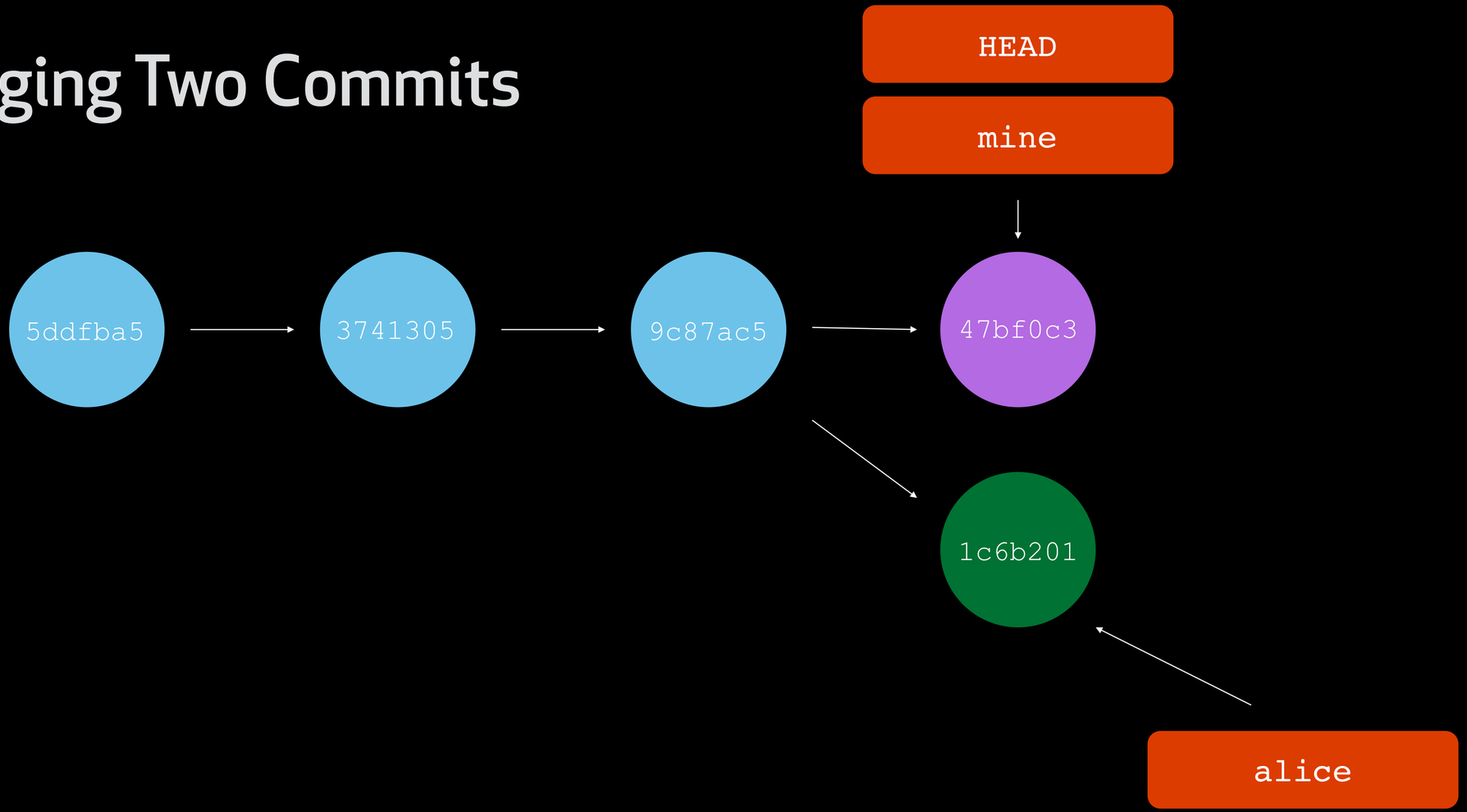

HEAD
mine
5ddfba5
3741305
9c87ac5
47bf0c3
9969d03
1c6b201
alice

Demo

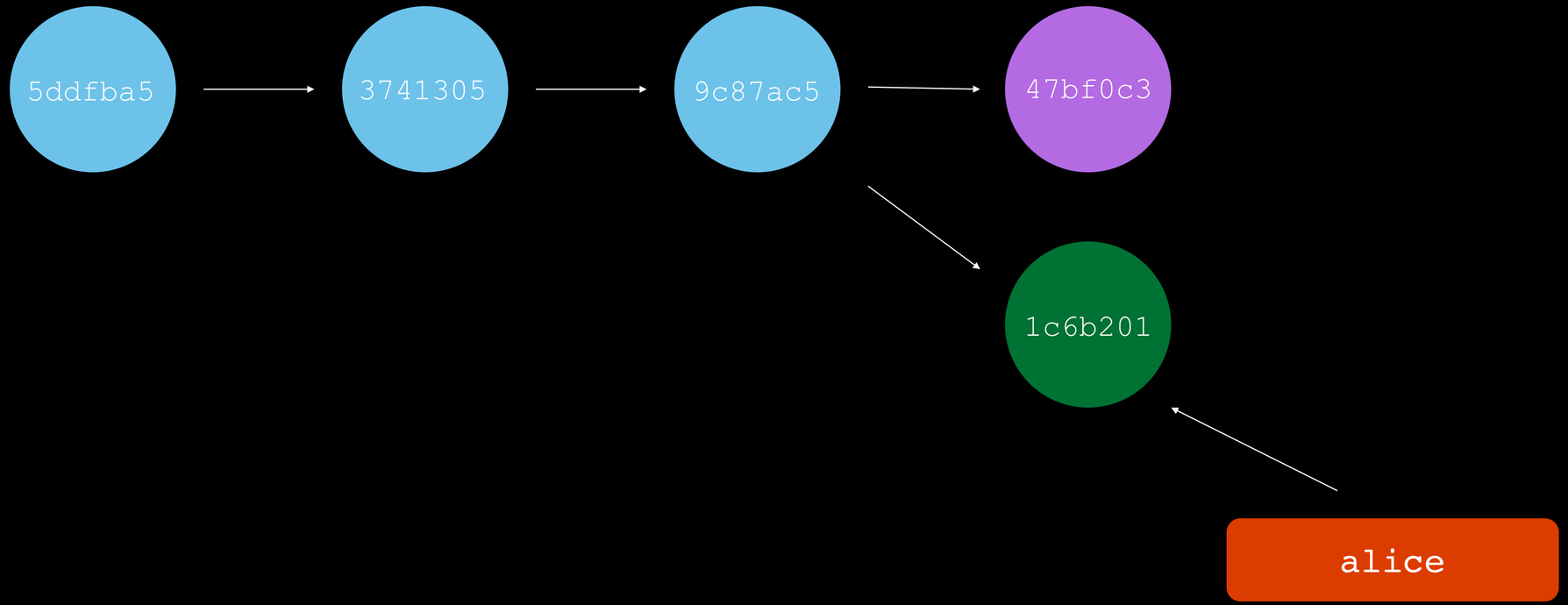
How the Current Branch is Stored

Merging Two Commits

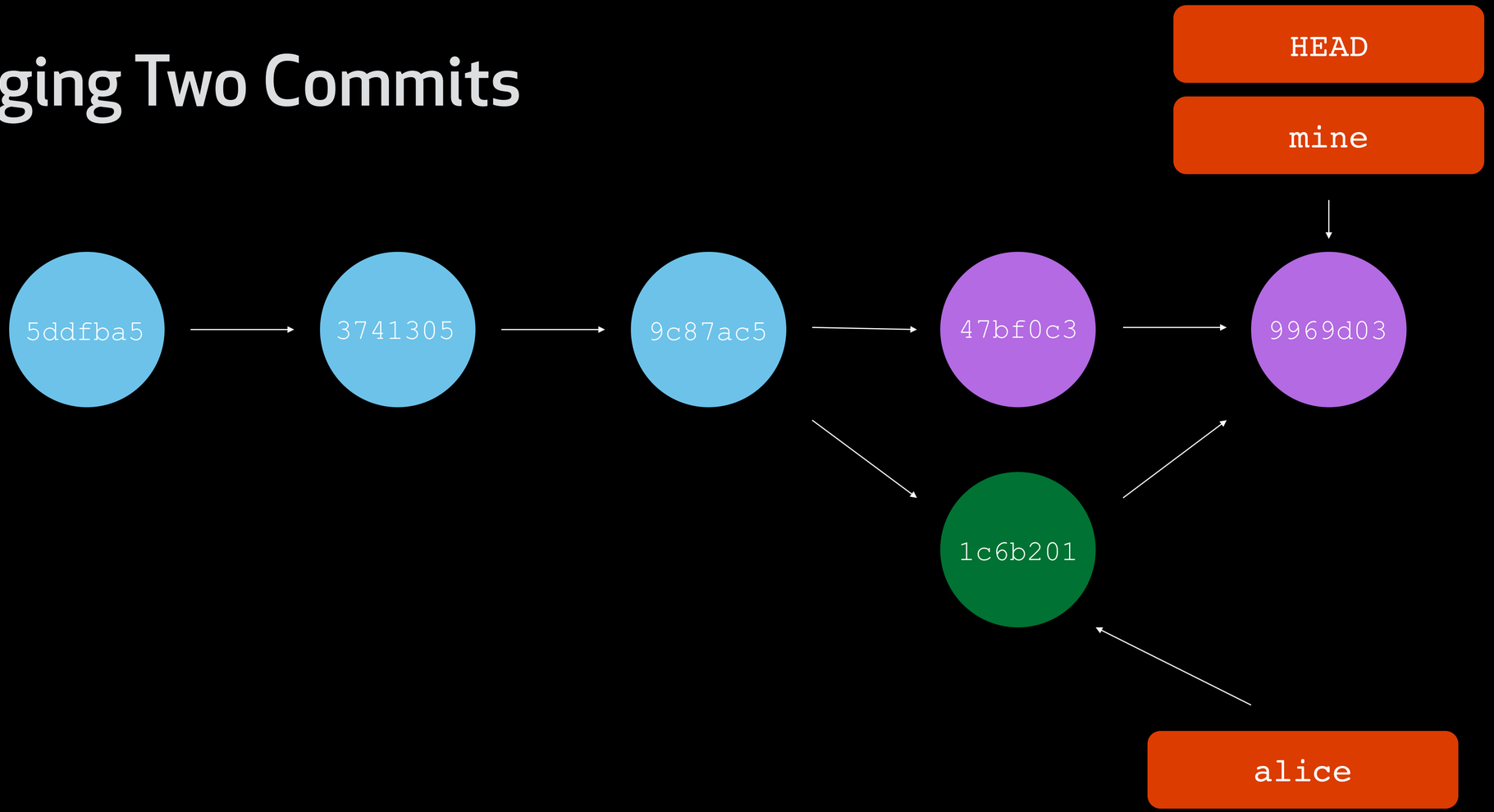
Merging Two Commits



Merging Two Commits



Merging Two Commits



Demo

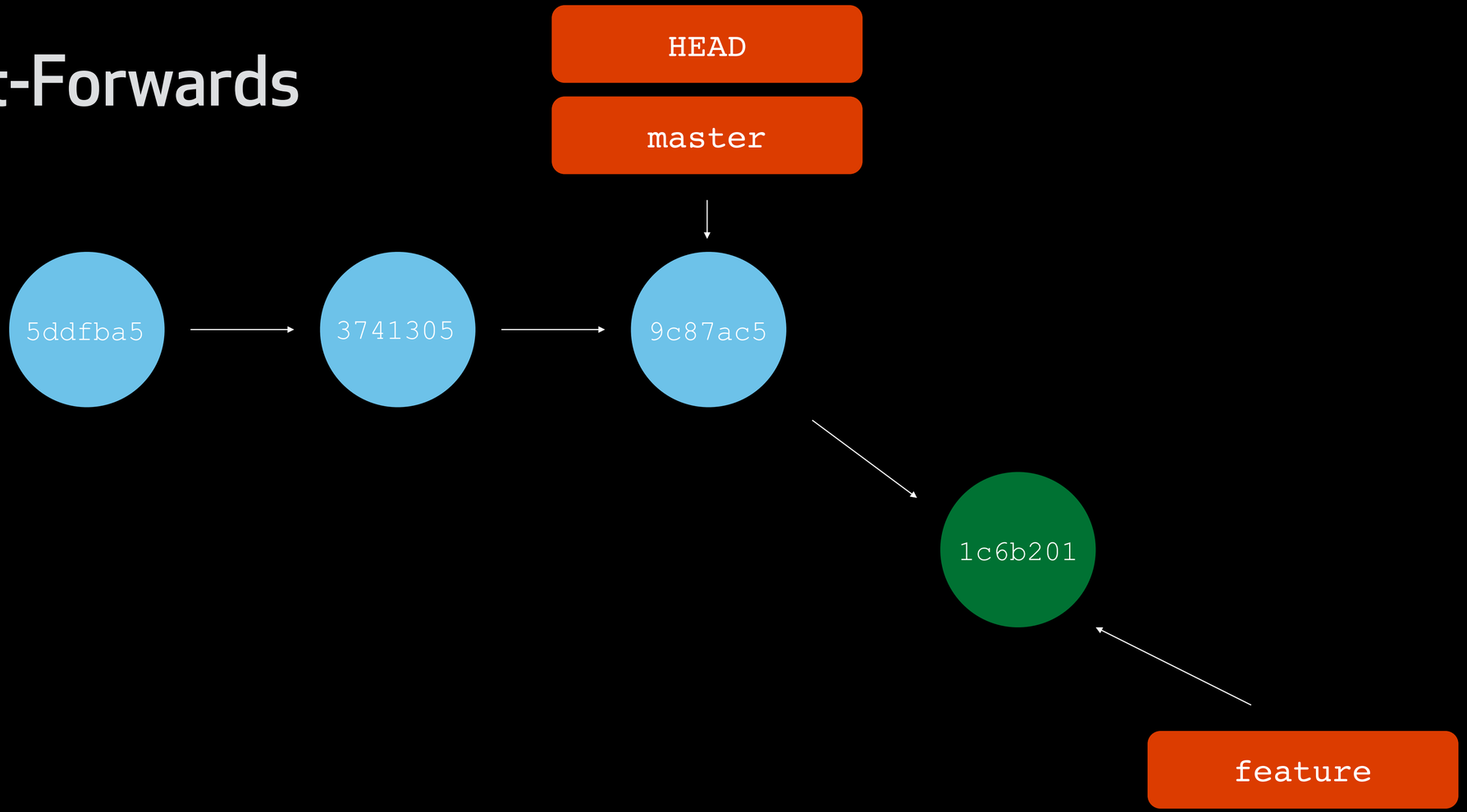
Merging Two Commits

Merges

Fast-Forwards

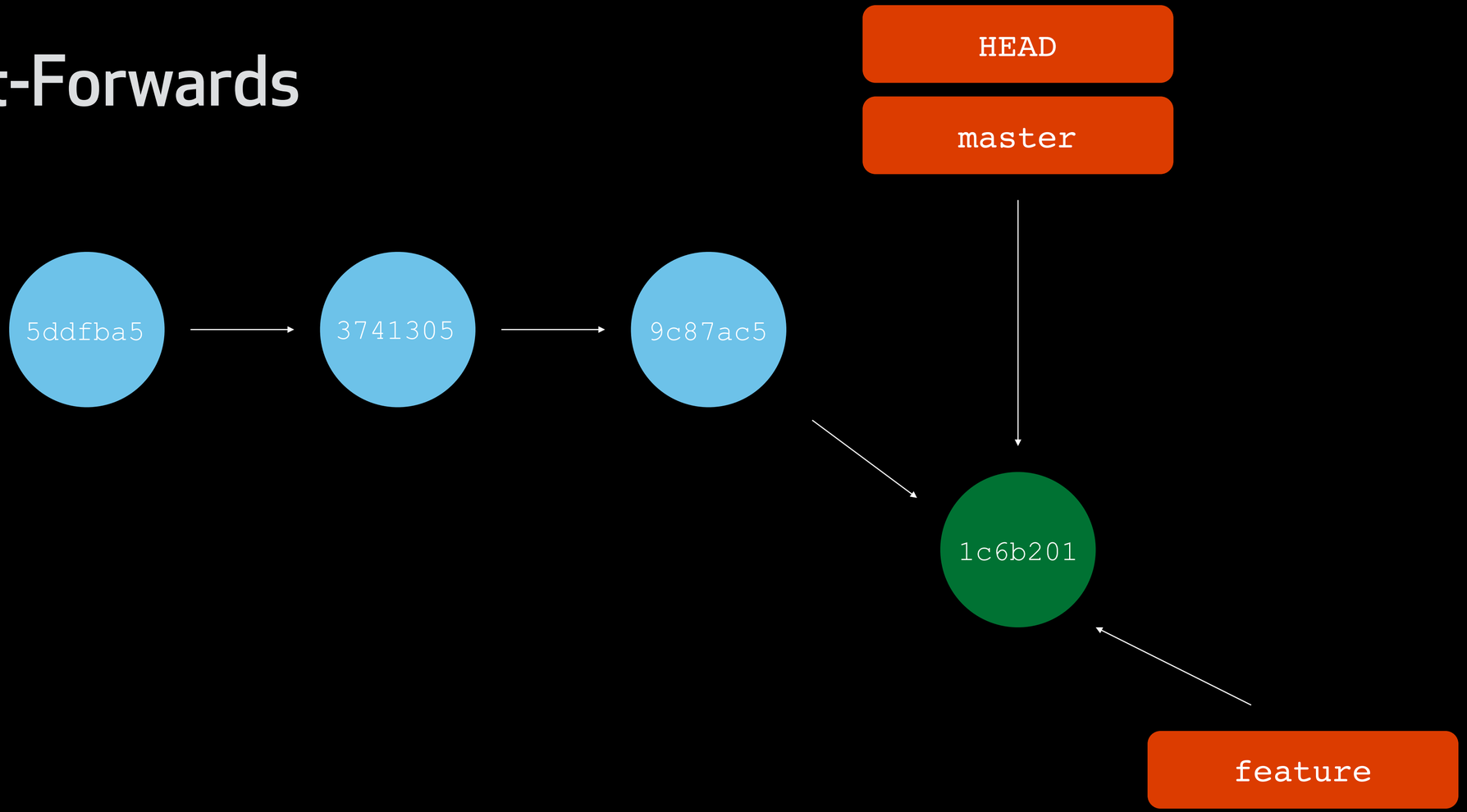
Merges

Fast-Forwards



Merges

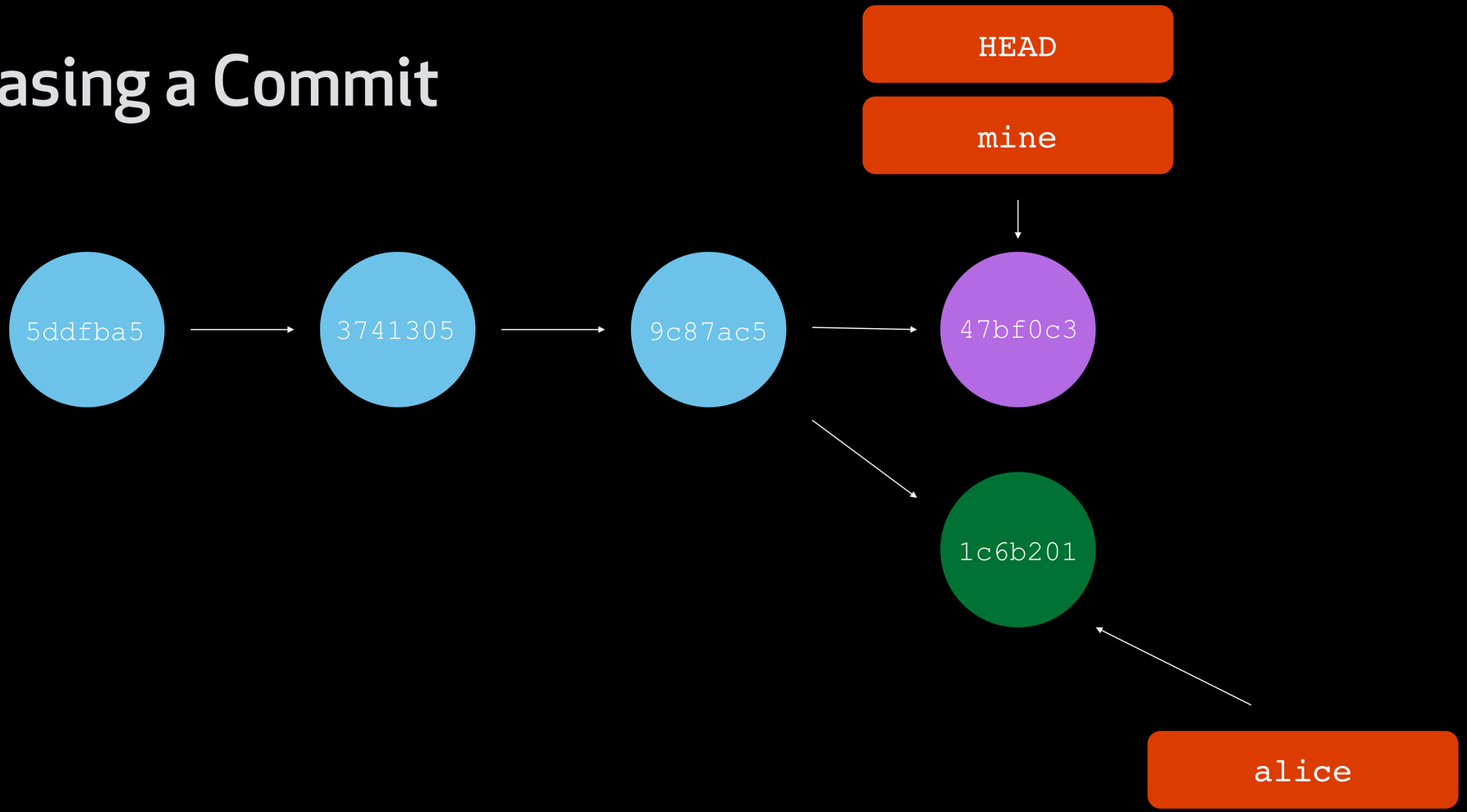
Fast-Forwards



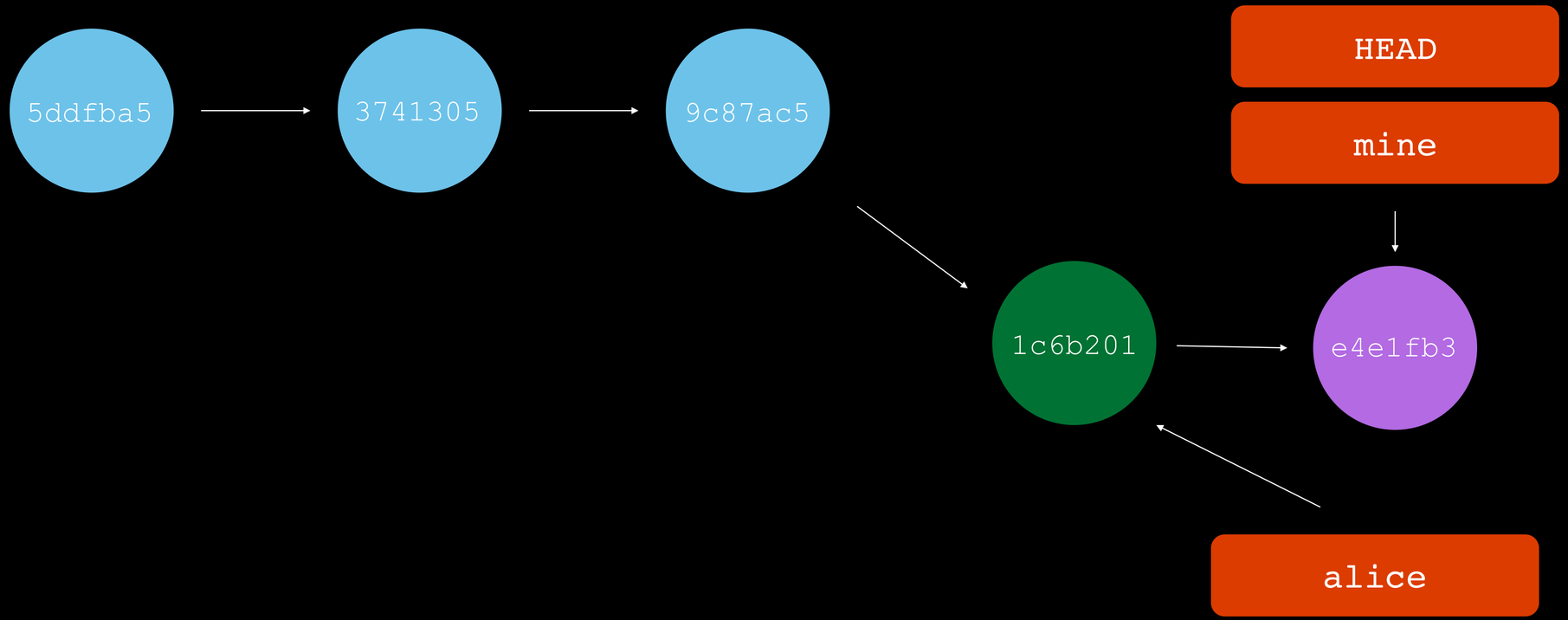
Demo

Fast-Forward Merges

Rebasing a Commit



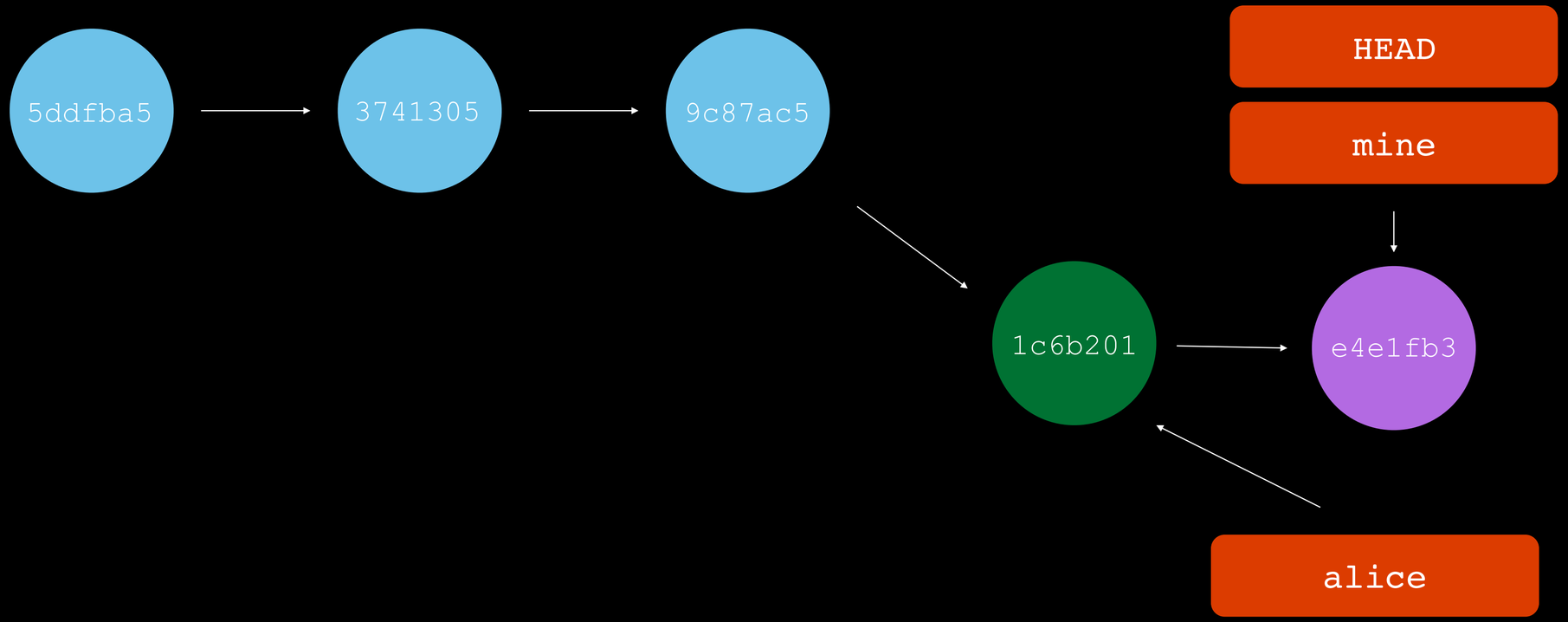
Rebasing a Commit



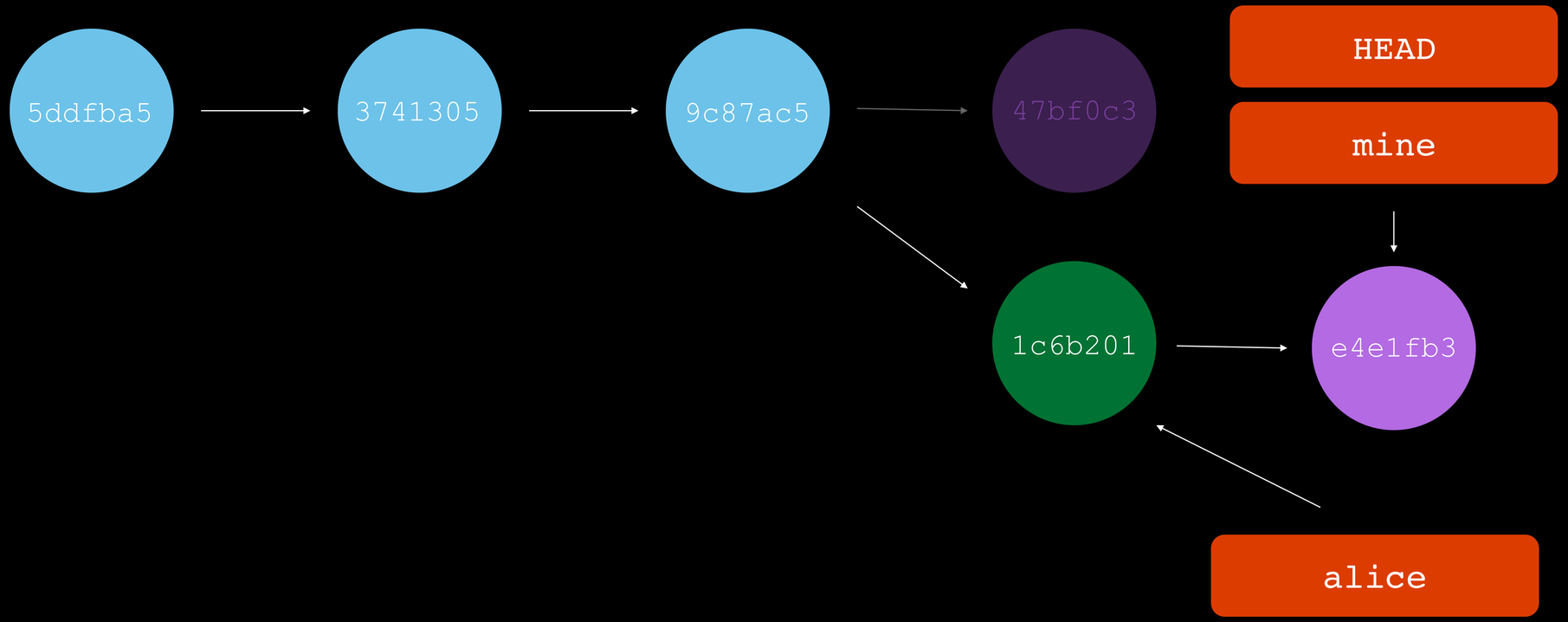
Demo

Rebasing a Commit Onto a New Branch

Rebasing a Commit



Rebasing a Commit



The Git commands are simply a leaky abstraction over the data storage.

Thank you!

Edward Thomson
Twitter, GitHub: **@ethomson**
Author, **Git for Visual Studio** (www.gitforvisualstudio.com)