

WTF is GIT?

a humble talk by Sergey Arkhipov
in MERA on 15 Nov 2011

WTF is GIT?

Agenda

1. What is GIT?
2. GIT vs ClearCase
3. Fully distributed
4. GIT workflow
5. Musthave commands
6. Credits

What is GIT?

What is GIT?

Git is an open source,
distributed version control
system designed for speed
and efficiency.

What is GIT?

Git is an open source,
distributed version control
system **designed for speed
and efficiency.**

Snapshots, not patches!

What is GIT?

Snapshots, not patches!



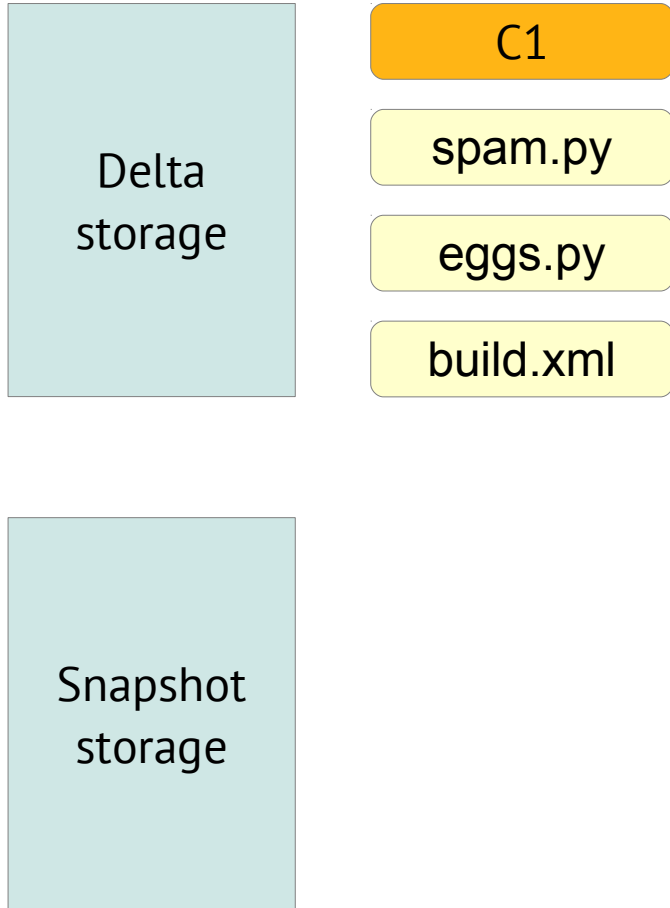
Delta
storage

The diagram consists of two light blue rectangular boxes stacked vertically on the left side of the slide. The top box is labeled 'Delta storage' and the bottom box is labeled 'Snapshot storage'. There are no lines or arrows connecting the two boxes.

Snapshot
storage

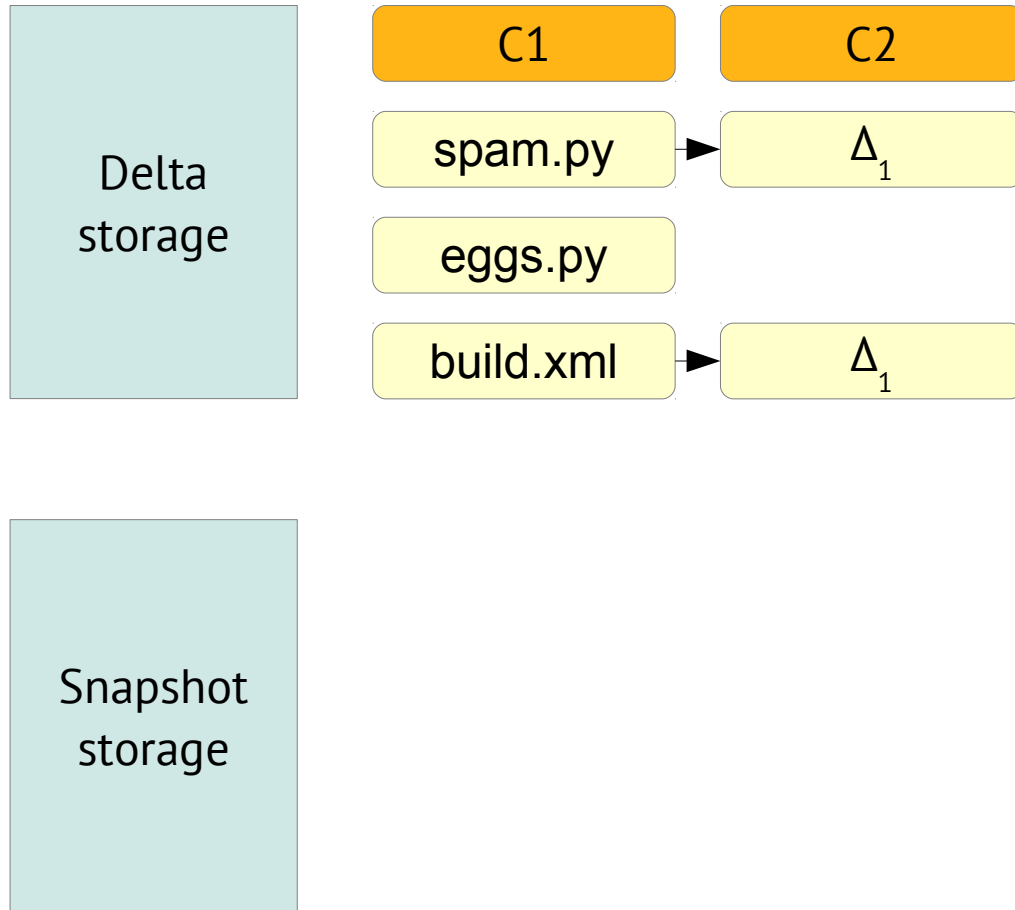
What is GIT?

Snapshots, not patches!



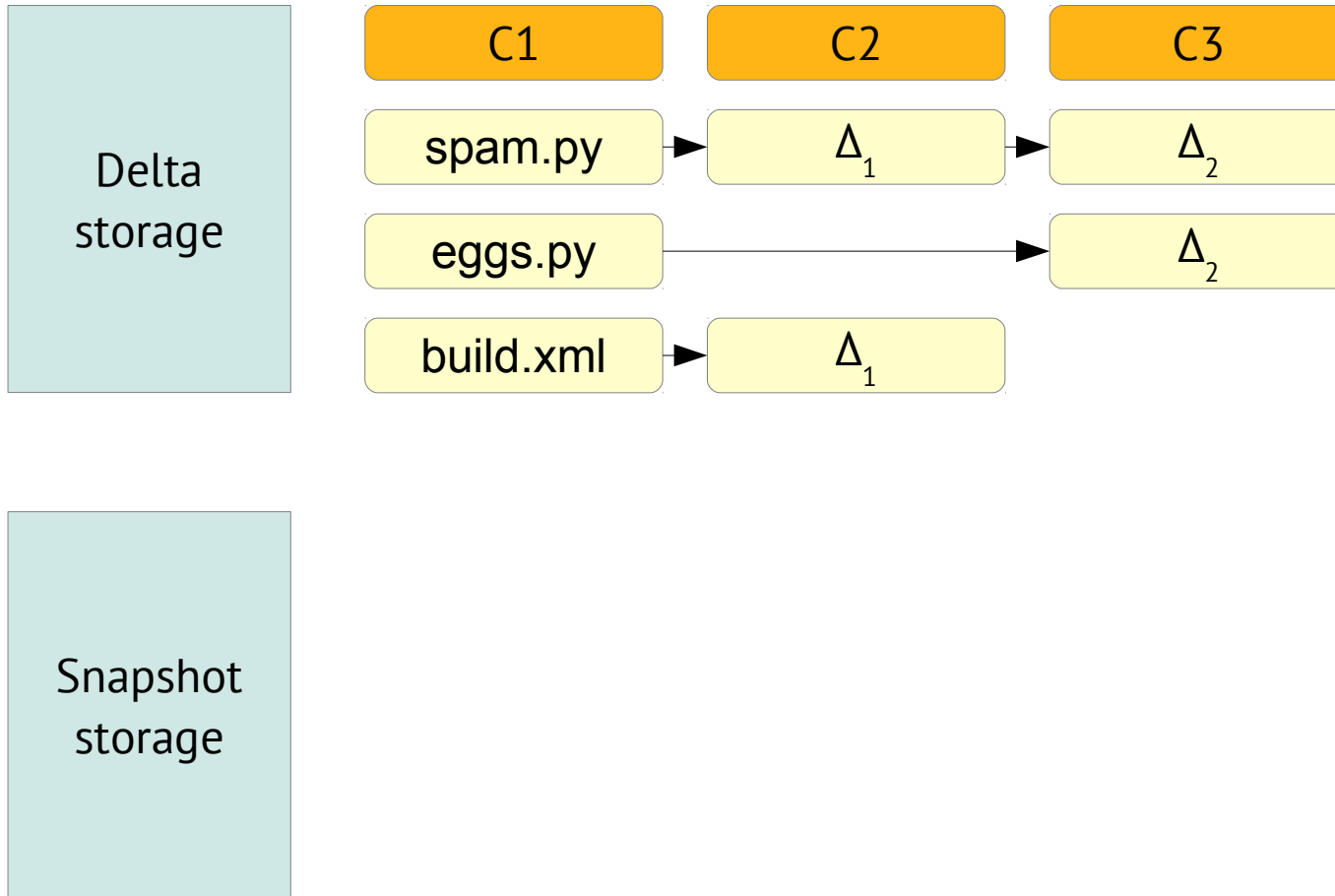
What is GIT?

Snapshots, not patches!



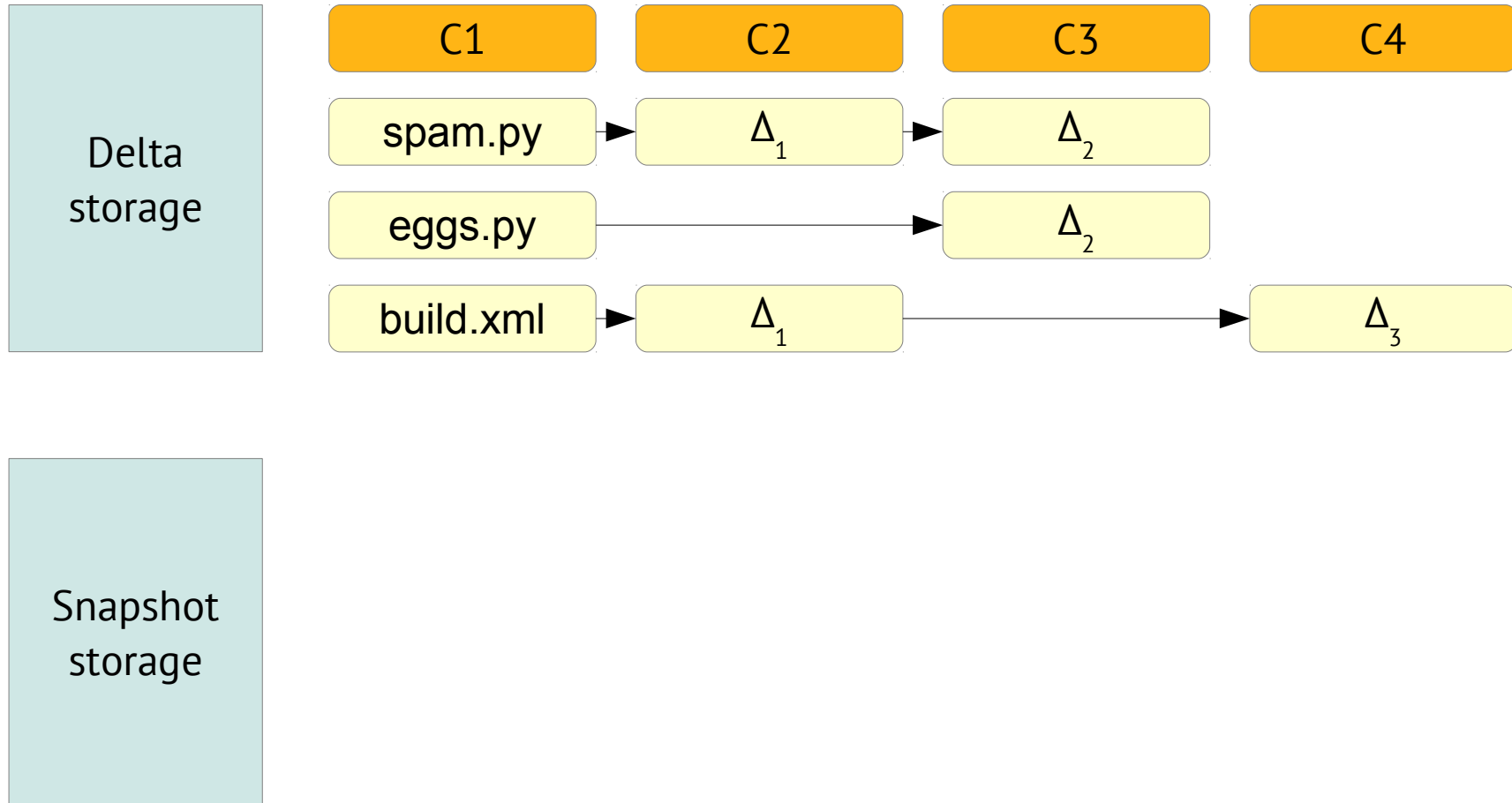
What is GIT?

Snapshots, not patches!



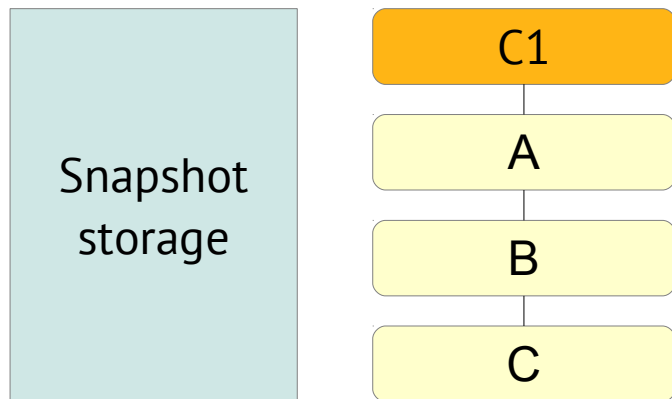
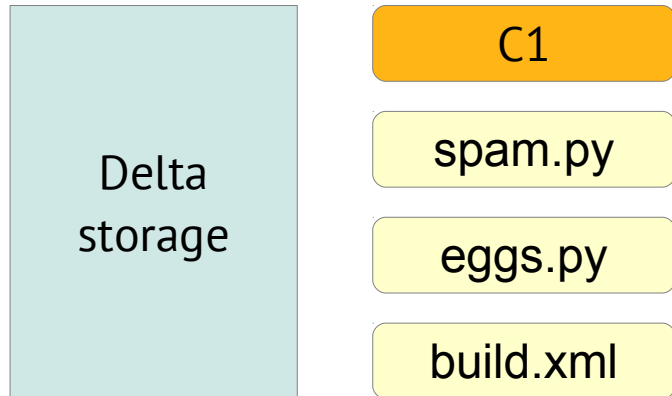
What is GIT?

Snapshots, not patches!



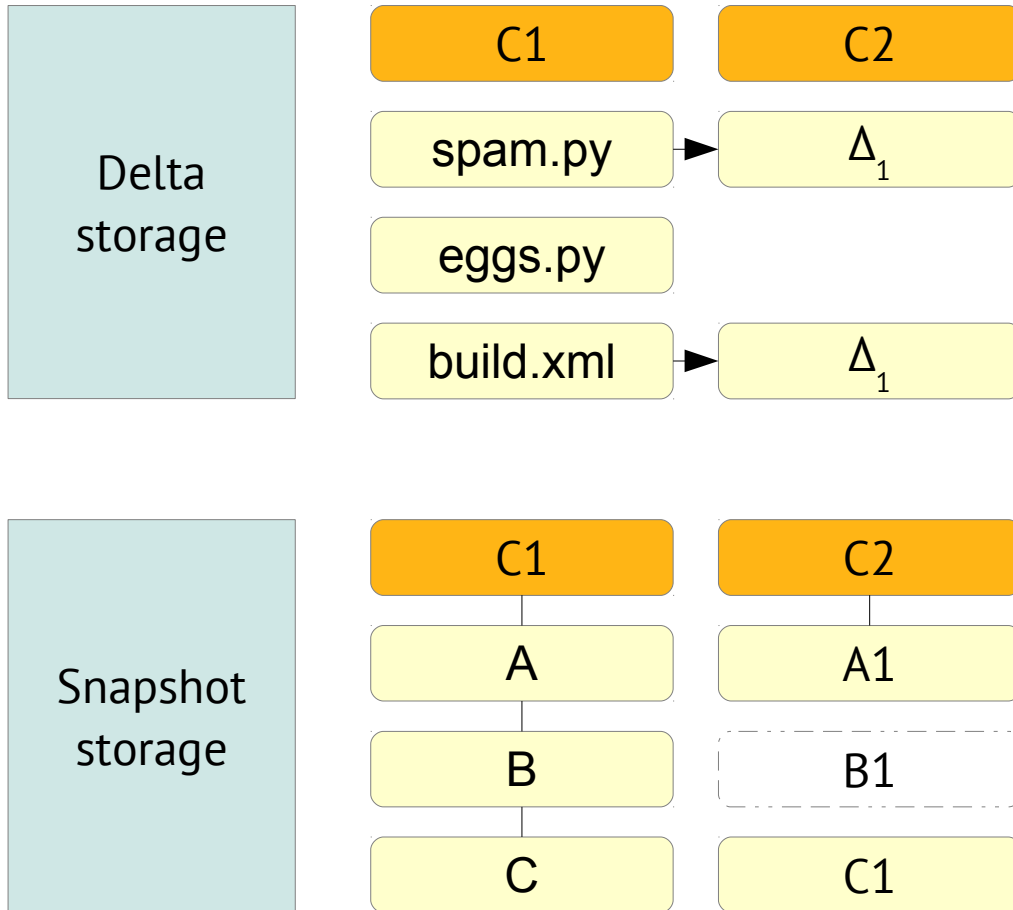
What is GIT?

Snapshots, not patches!



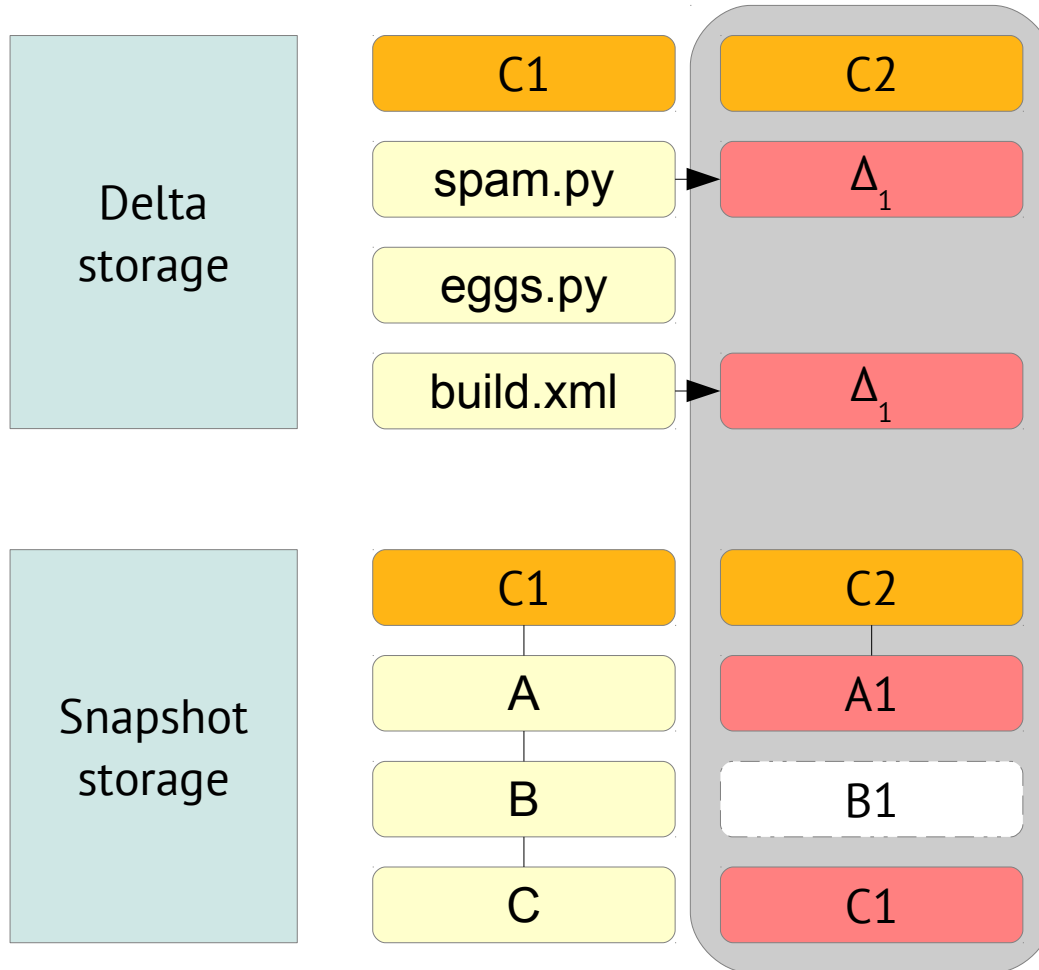
What is GIT?

Snapshots, not patches!



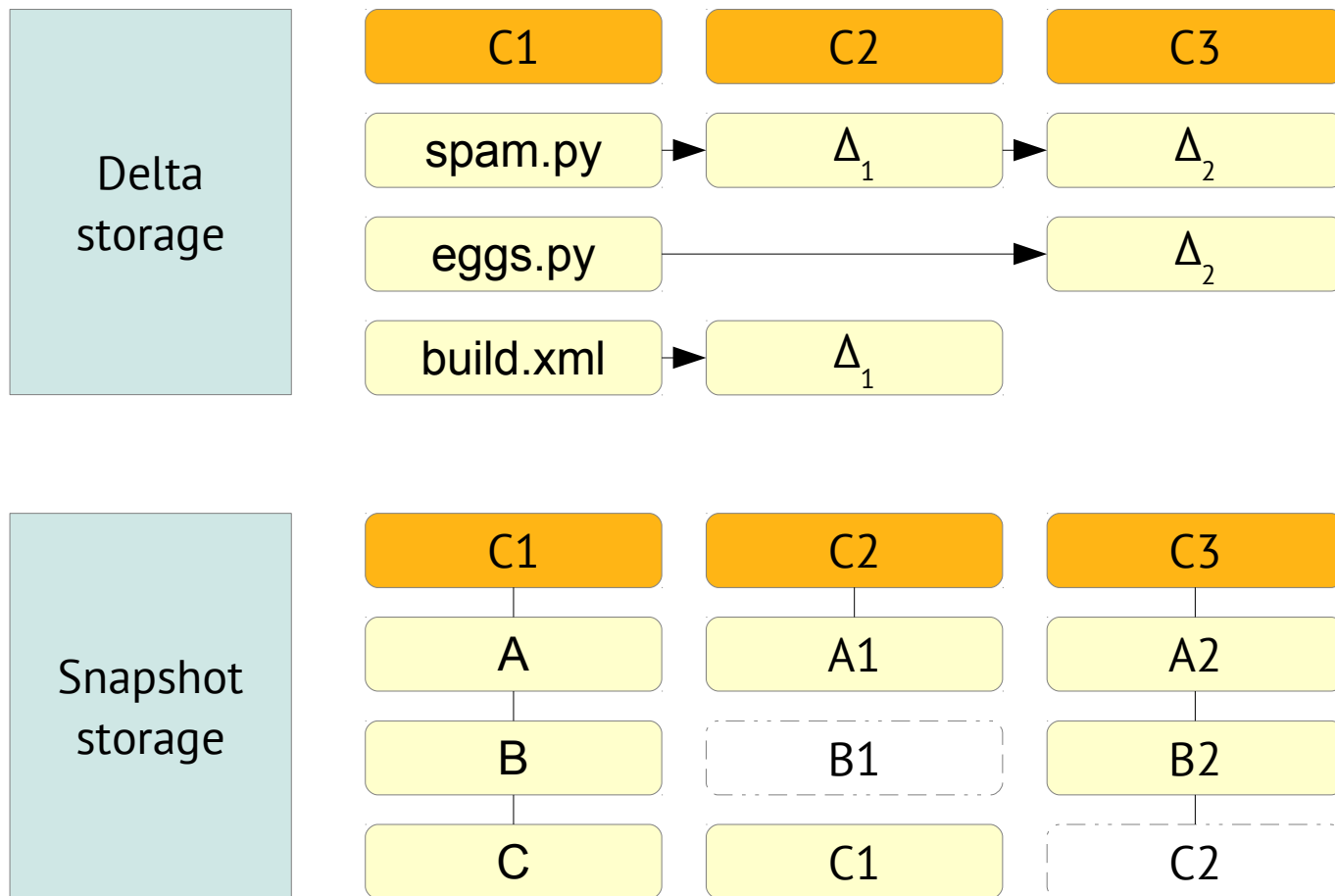
What is GIT?

Snapshots, not patches!



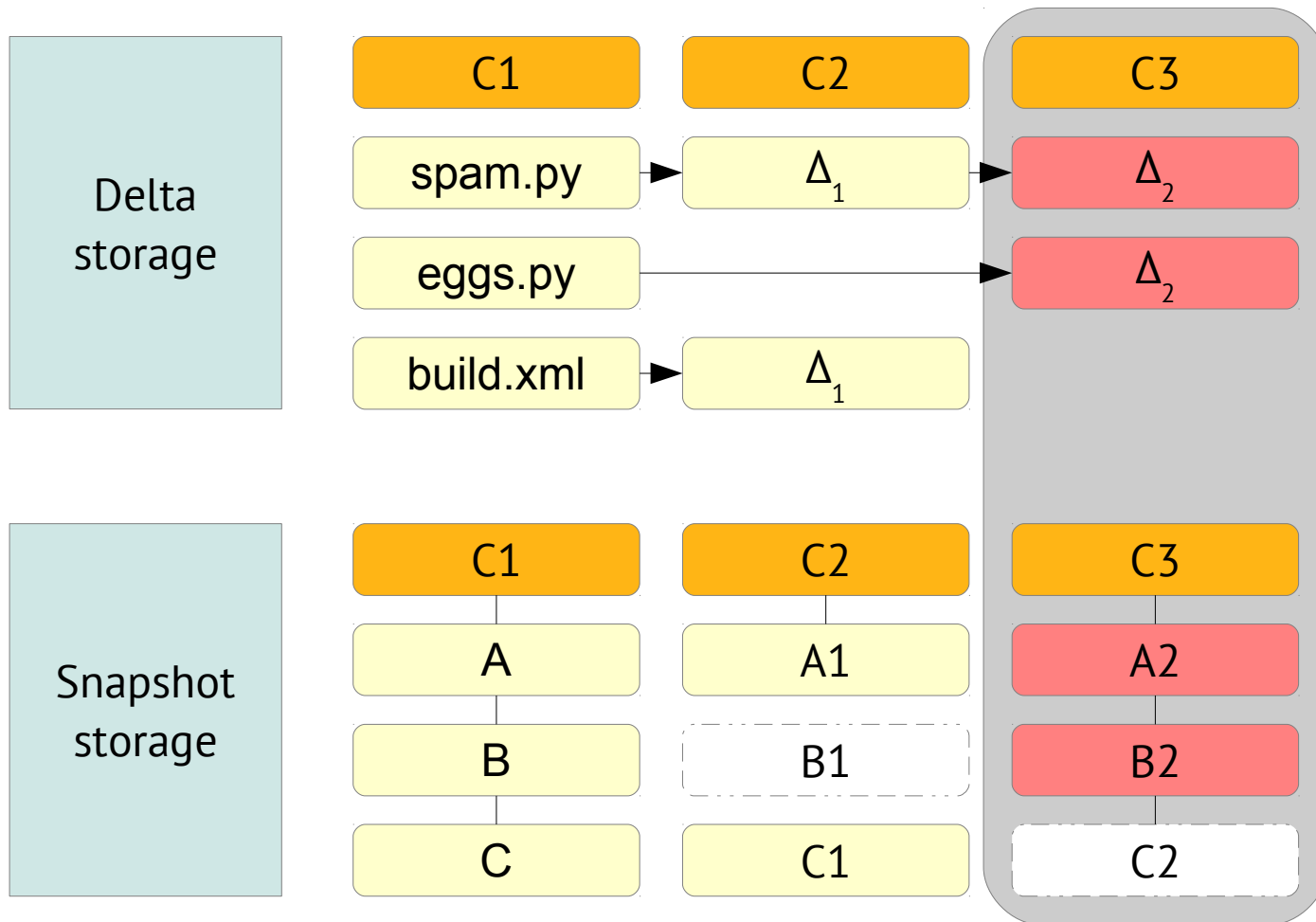
What is GIT?

Snapshots, not patches!



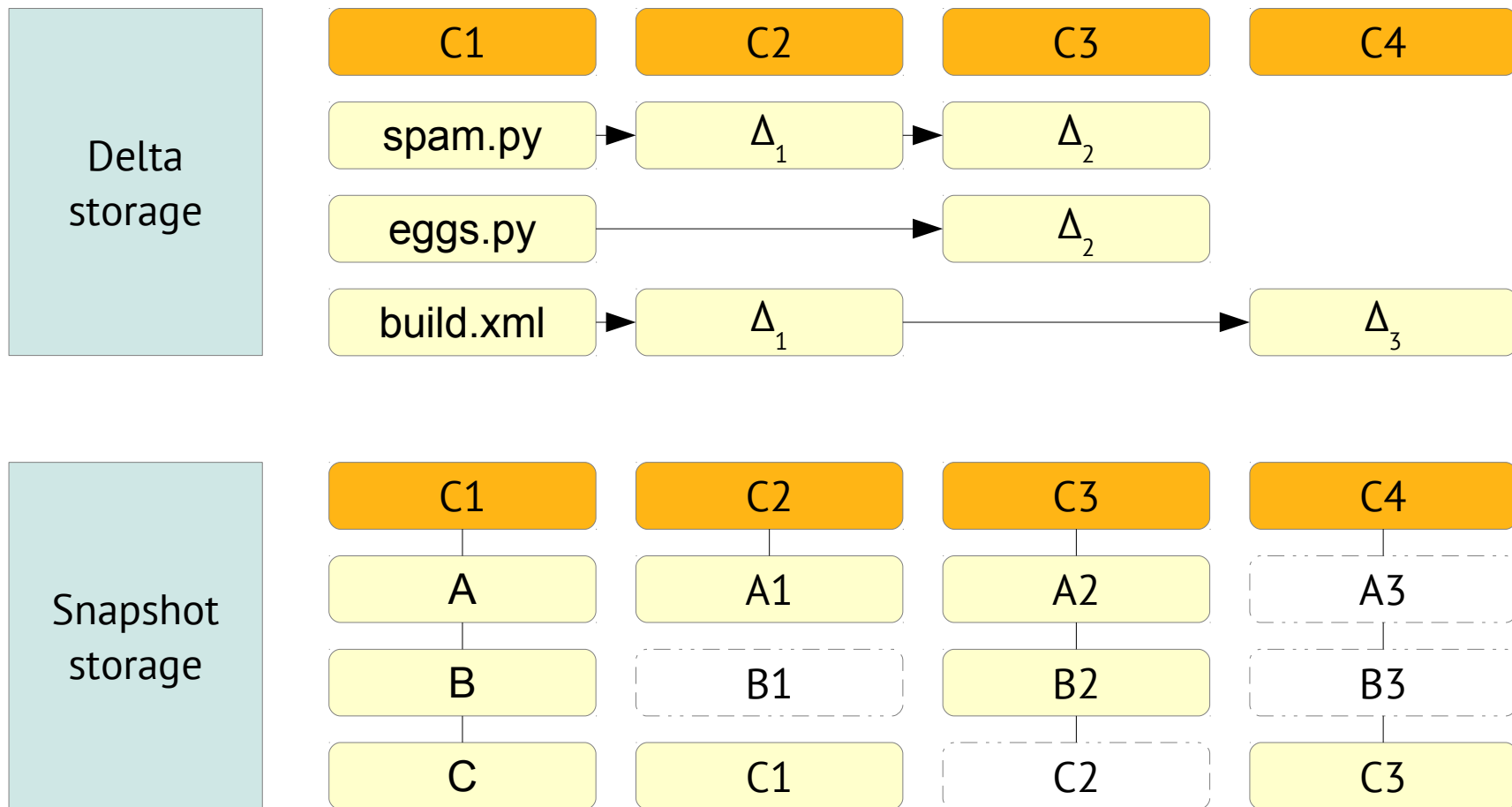
What is GIT?

Snapshots, not patches!



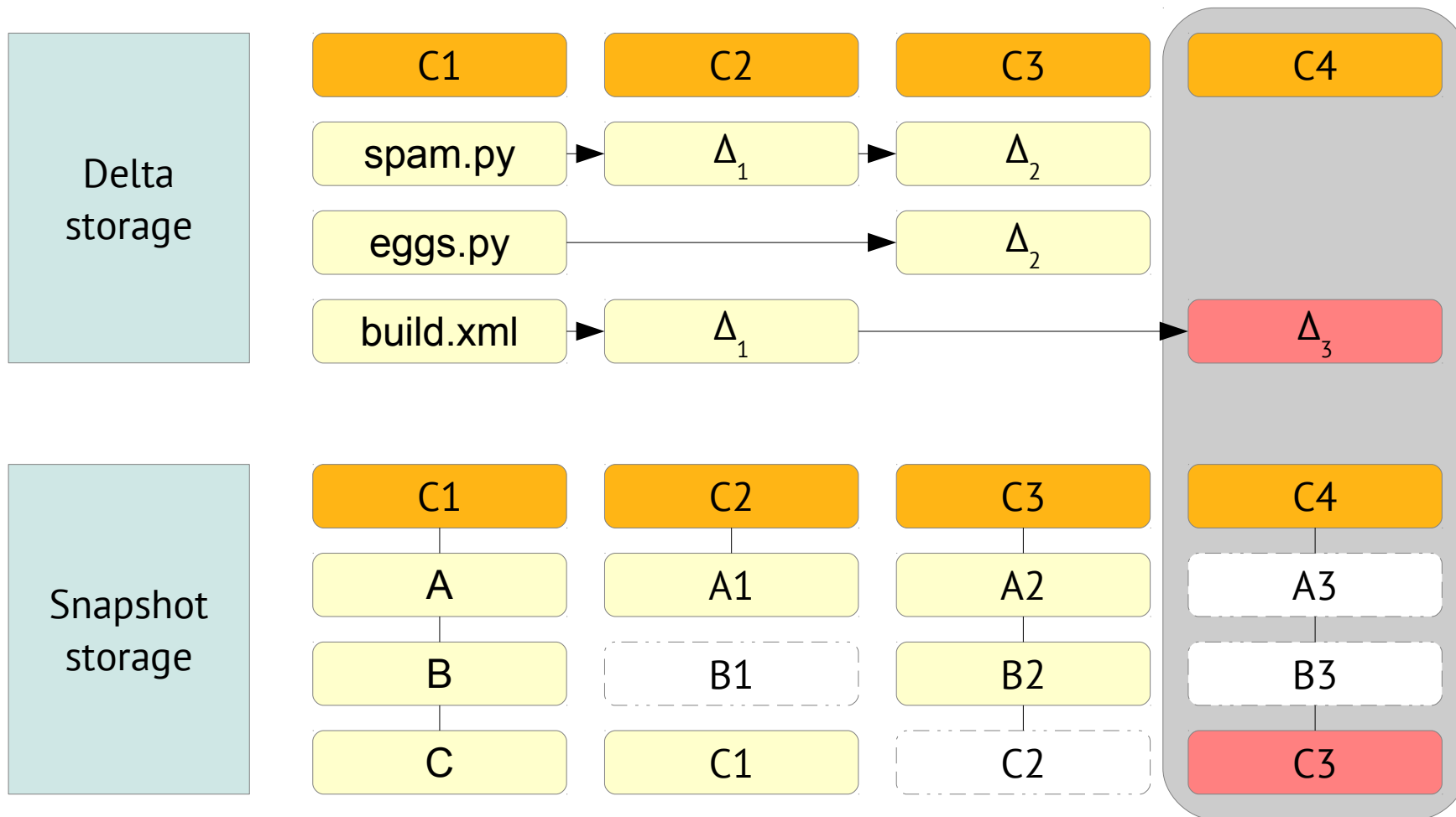
What is GIT?

Snapshots, not patches!



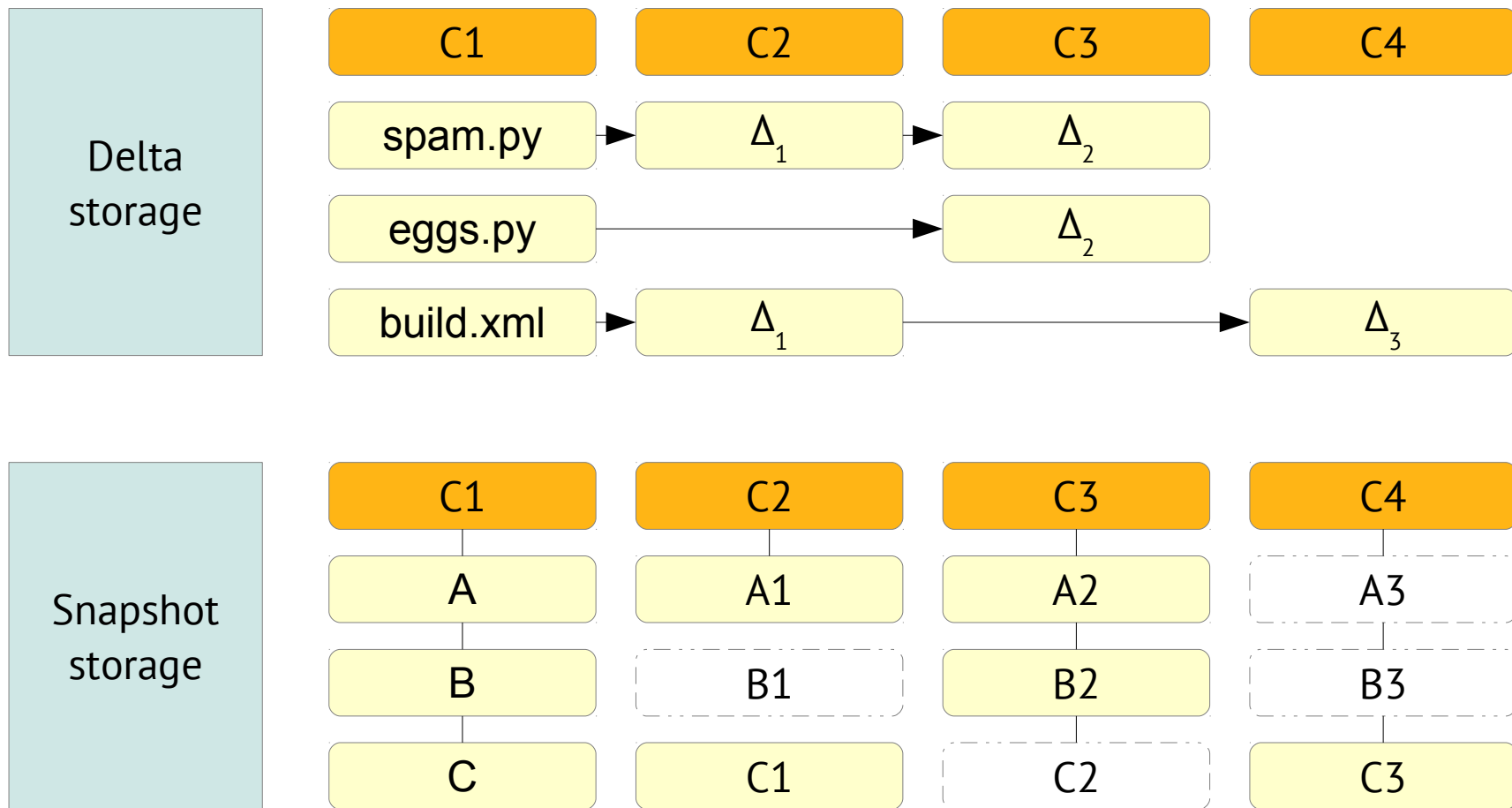
What is GIT?

Snapshots, not patches!



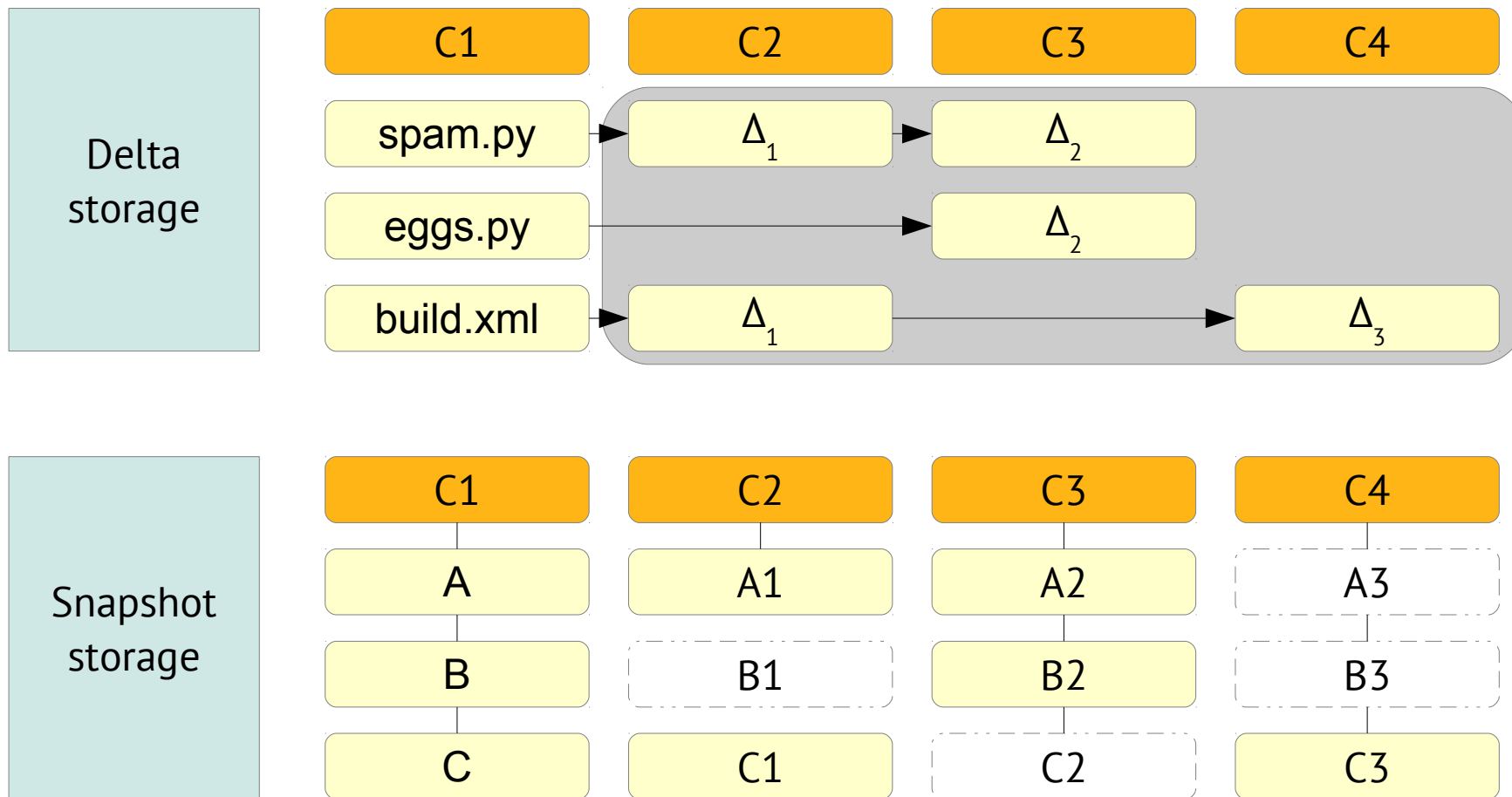
What is GIT?

Snapshots, not patches!



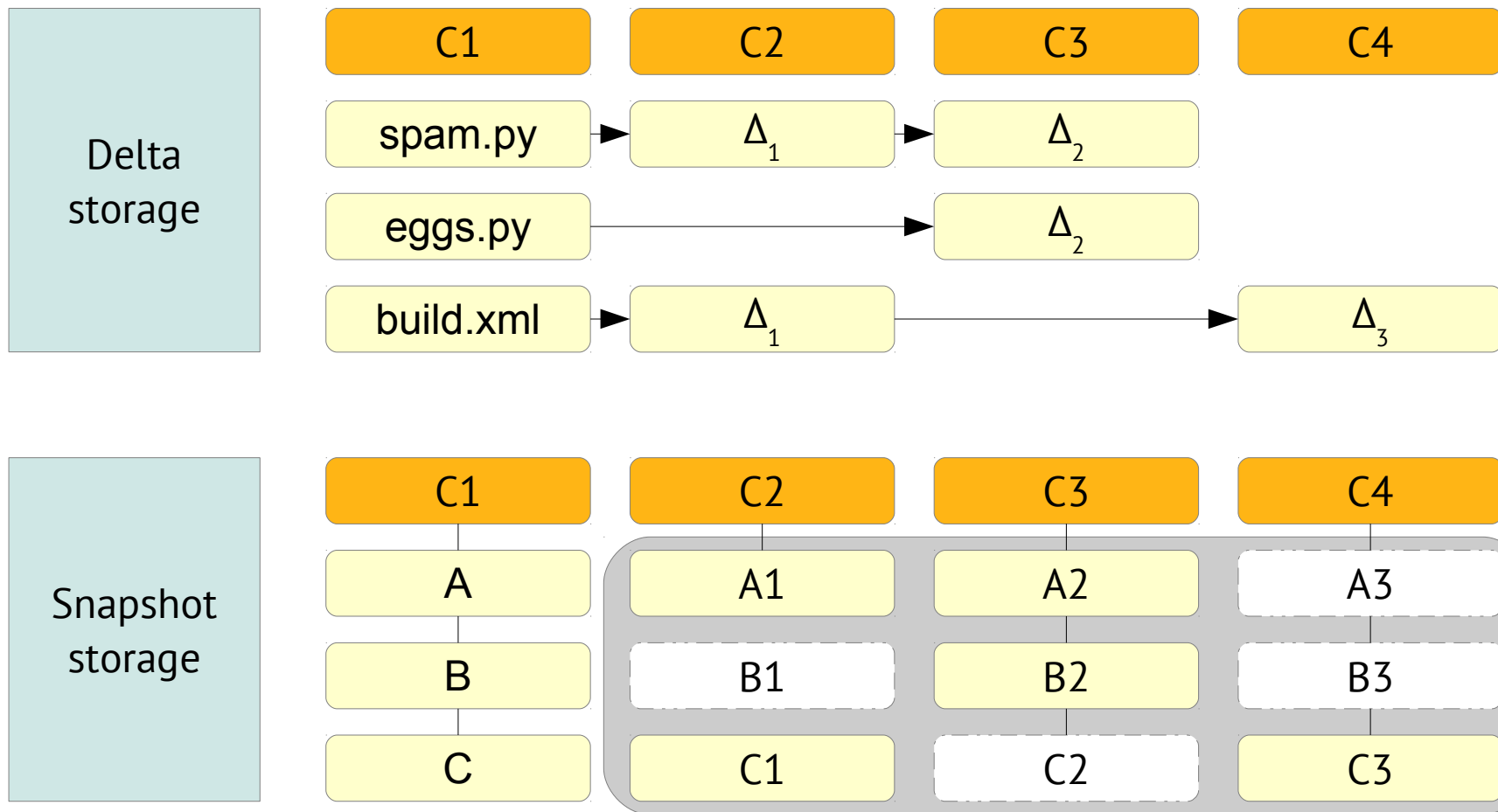
What is GIT?

Snapshots, not patches!



What is GIT?

Snapshots, not patches!



OK.

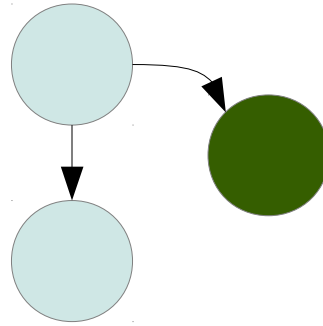
But how is this different to
our beloved ClearCase?

GIT vs ClearCase

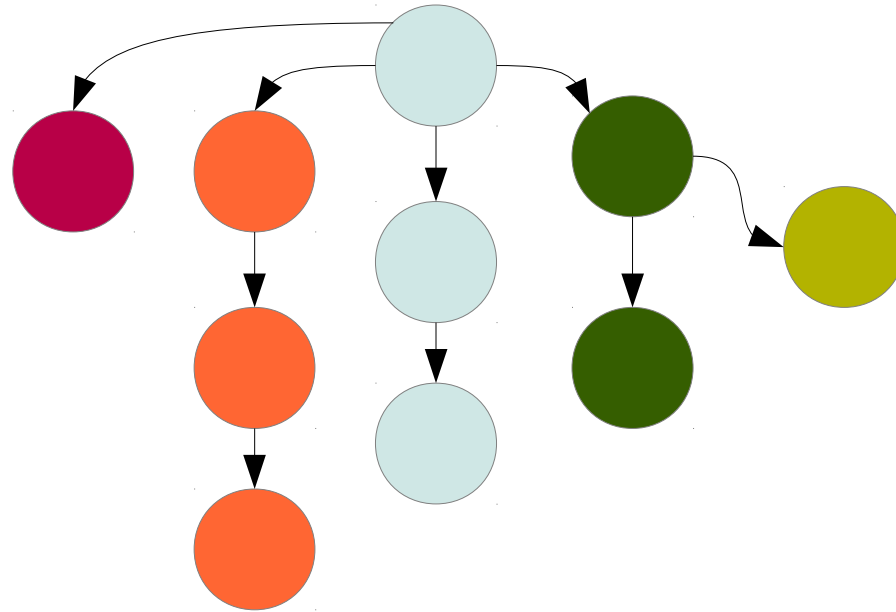
GIT vs ClearCase



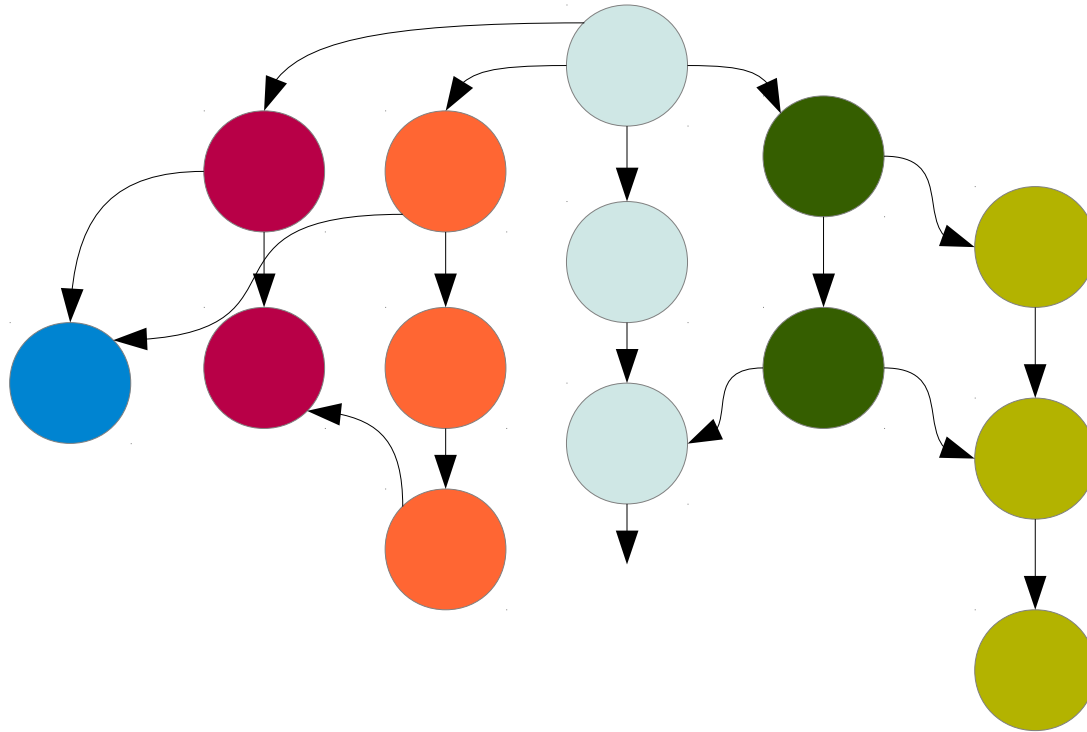
GIT vs ClearCase



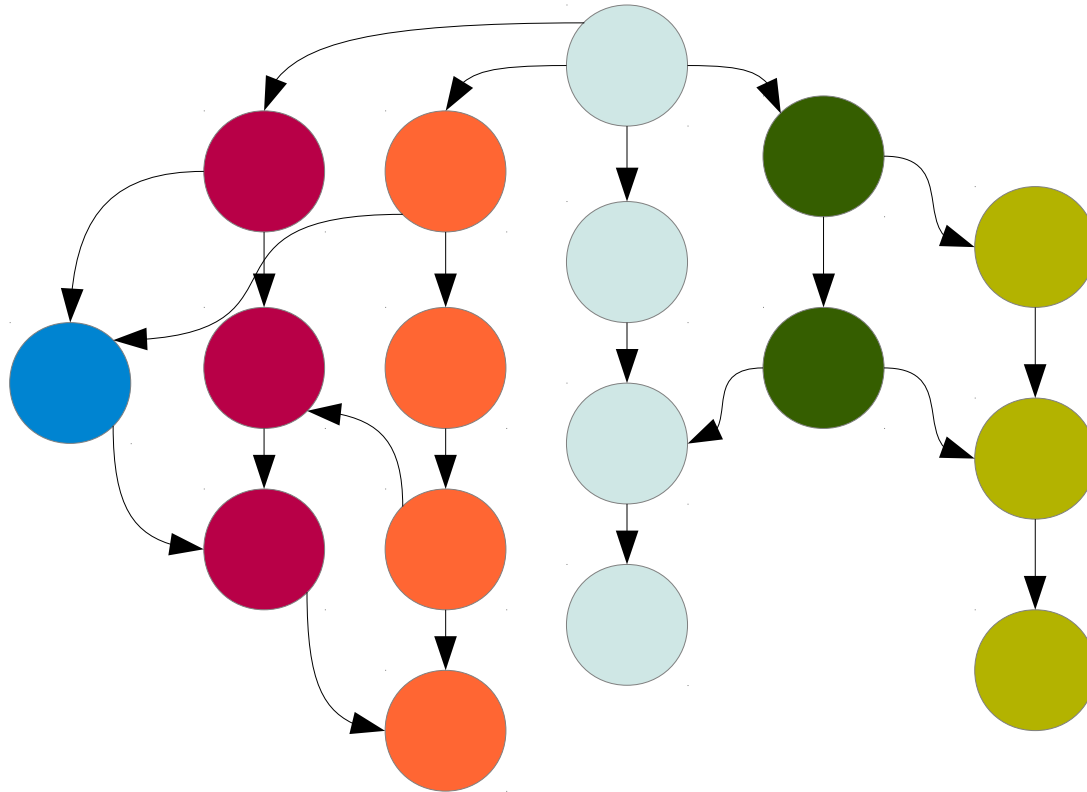
Git vs ClearCase



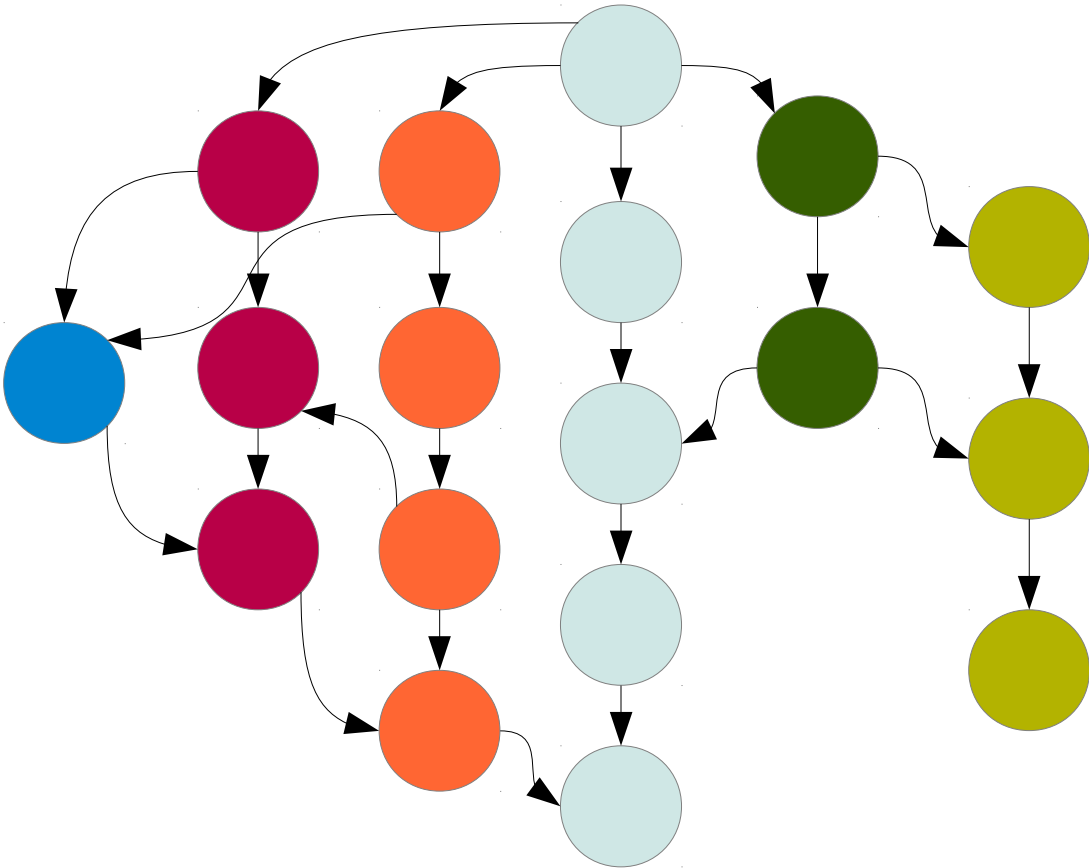
Git vs ClearCase



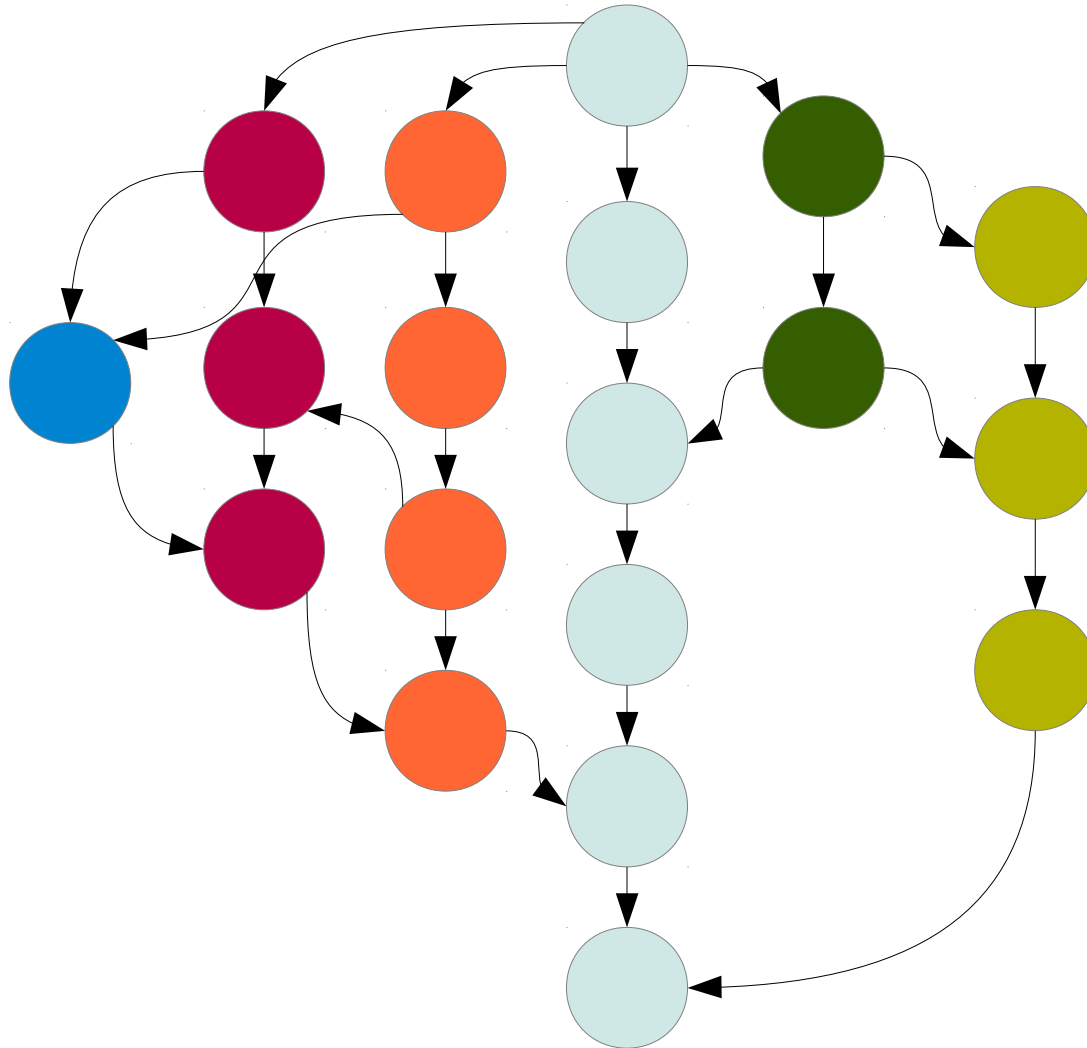
GIT vs ClearCase



GIT vs ClearCase

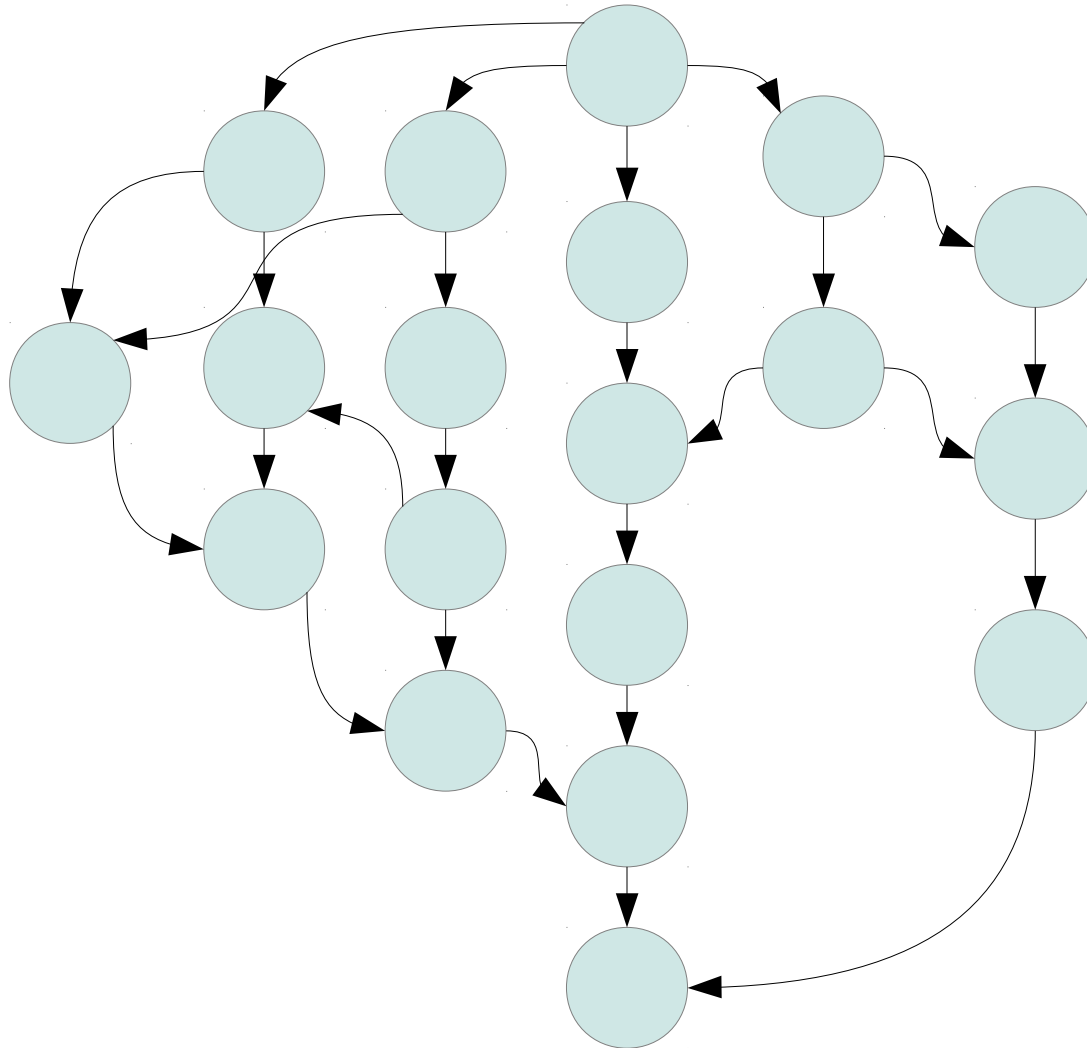


Git vs ClearCase



Everything looks great, but...

Git vs ClearCase



After 1 year...

GIT vs ClearCase

Where did Ivan
introduce his correction?



GIT vs ClearCase

How could I give
my beautiful patch to Nikolay?



GIT vs ClearCase

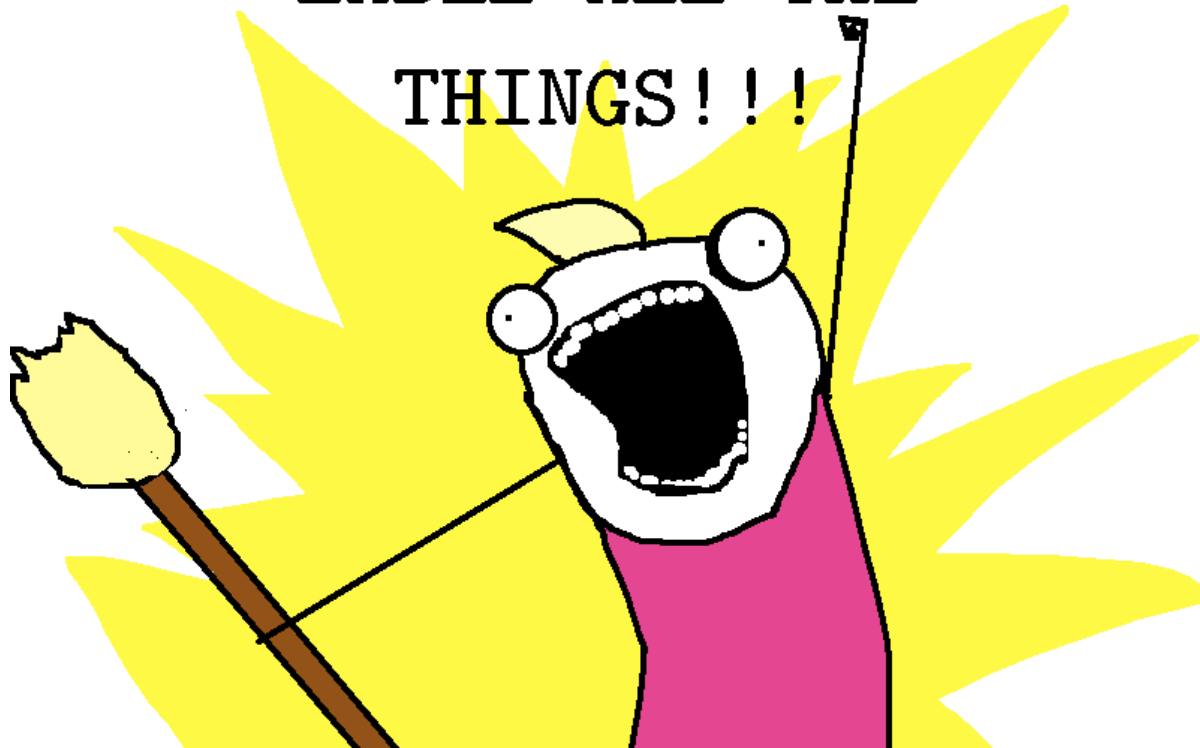
How should I manage
all the things?



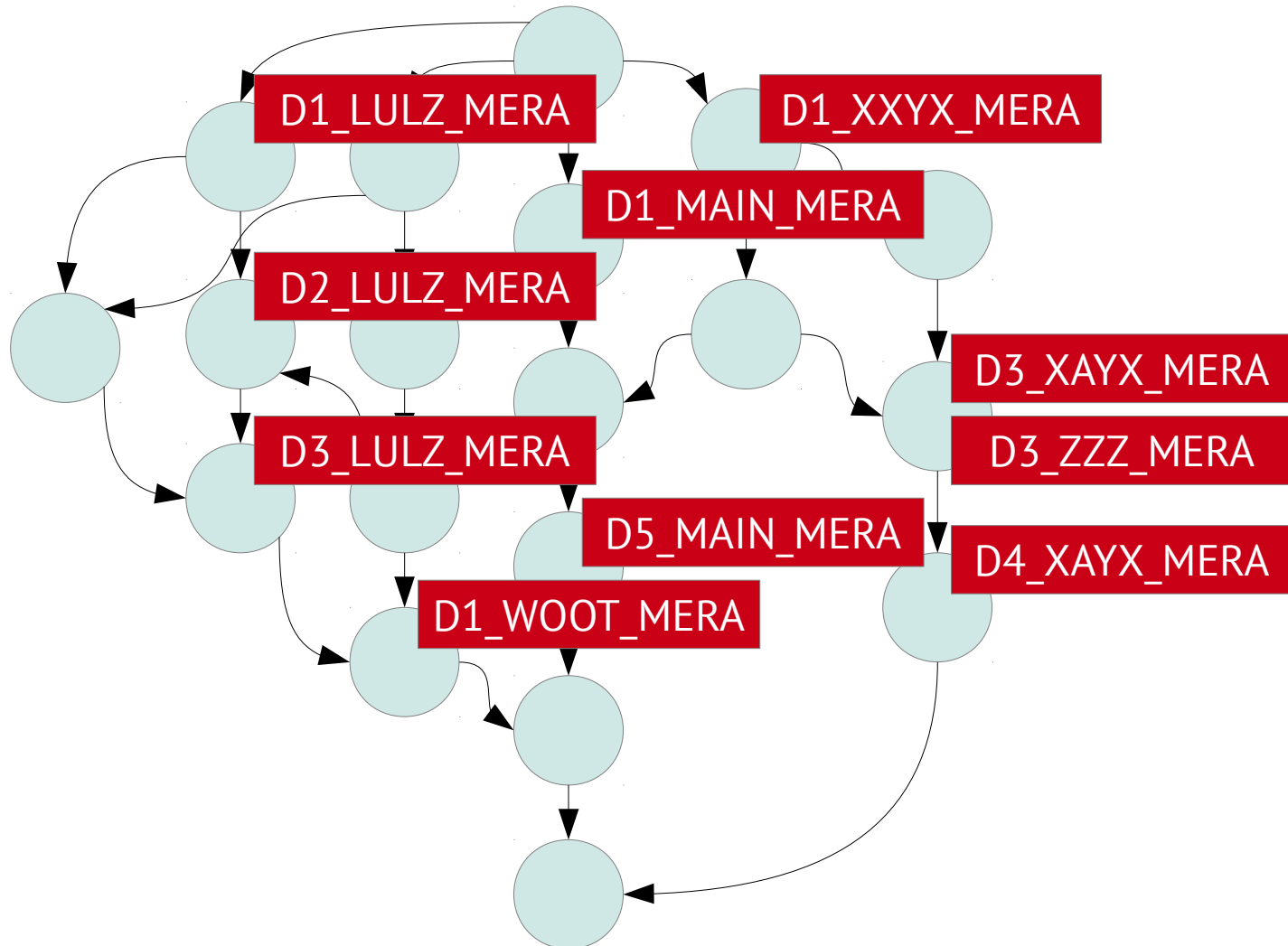
?

GIT vs ClearCase

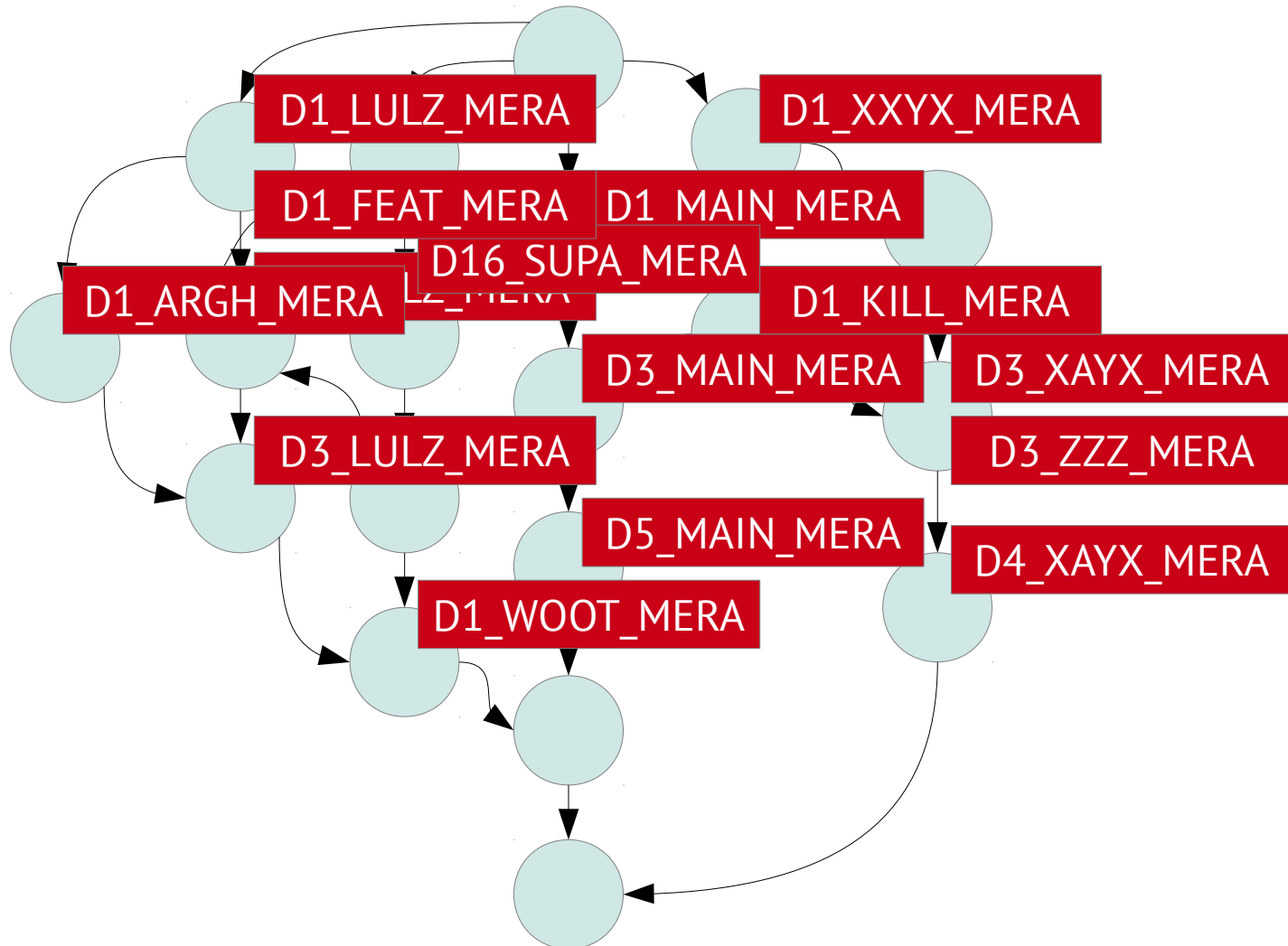
LABEL ALL THE
THINGS!!!



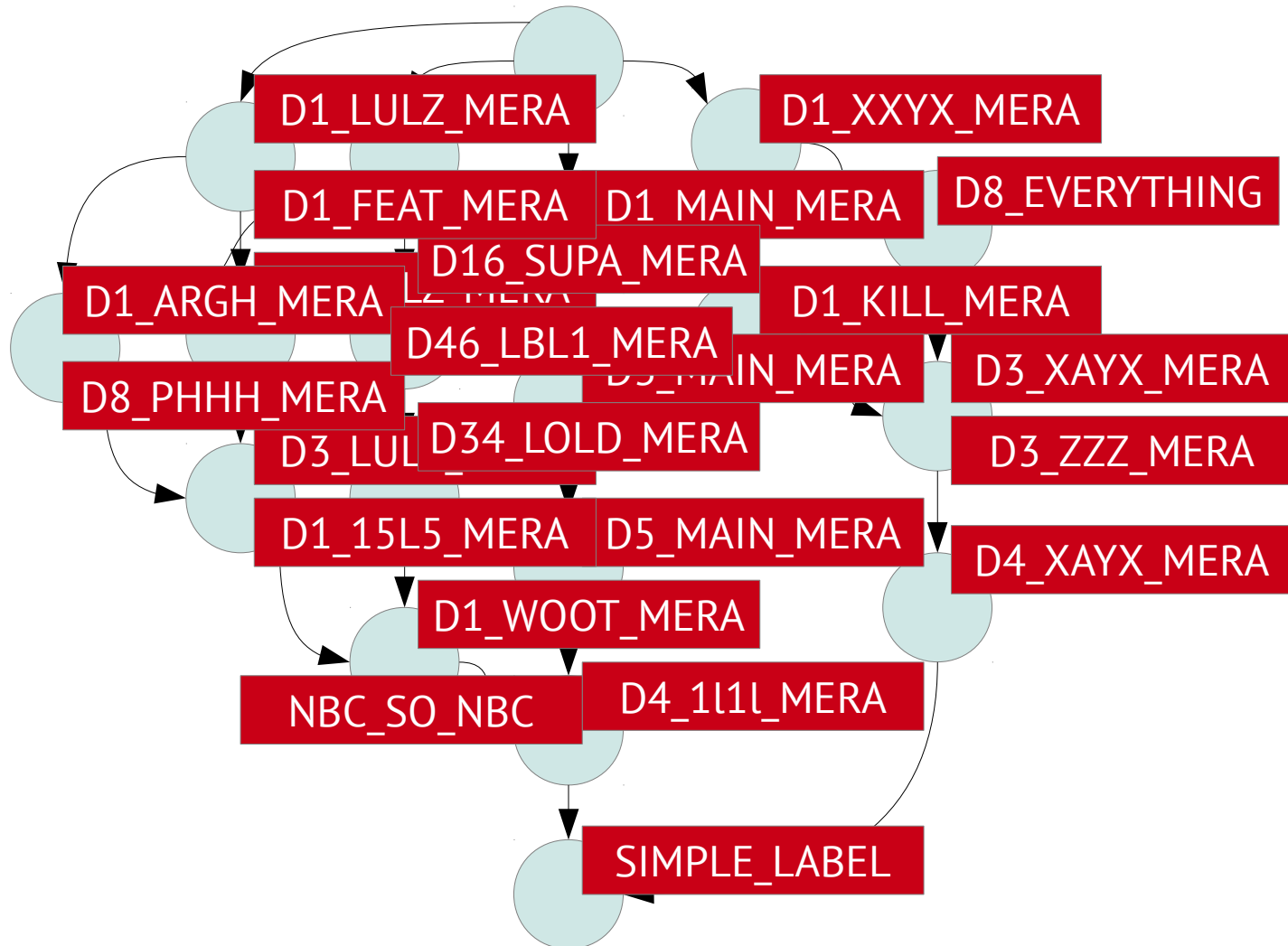
Git vs ClearCase



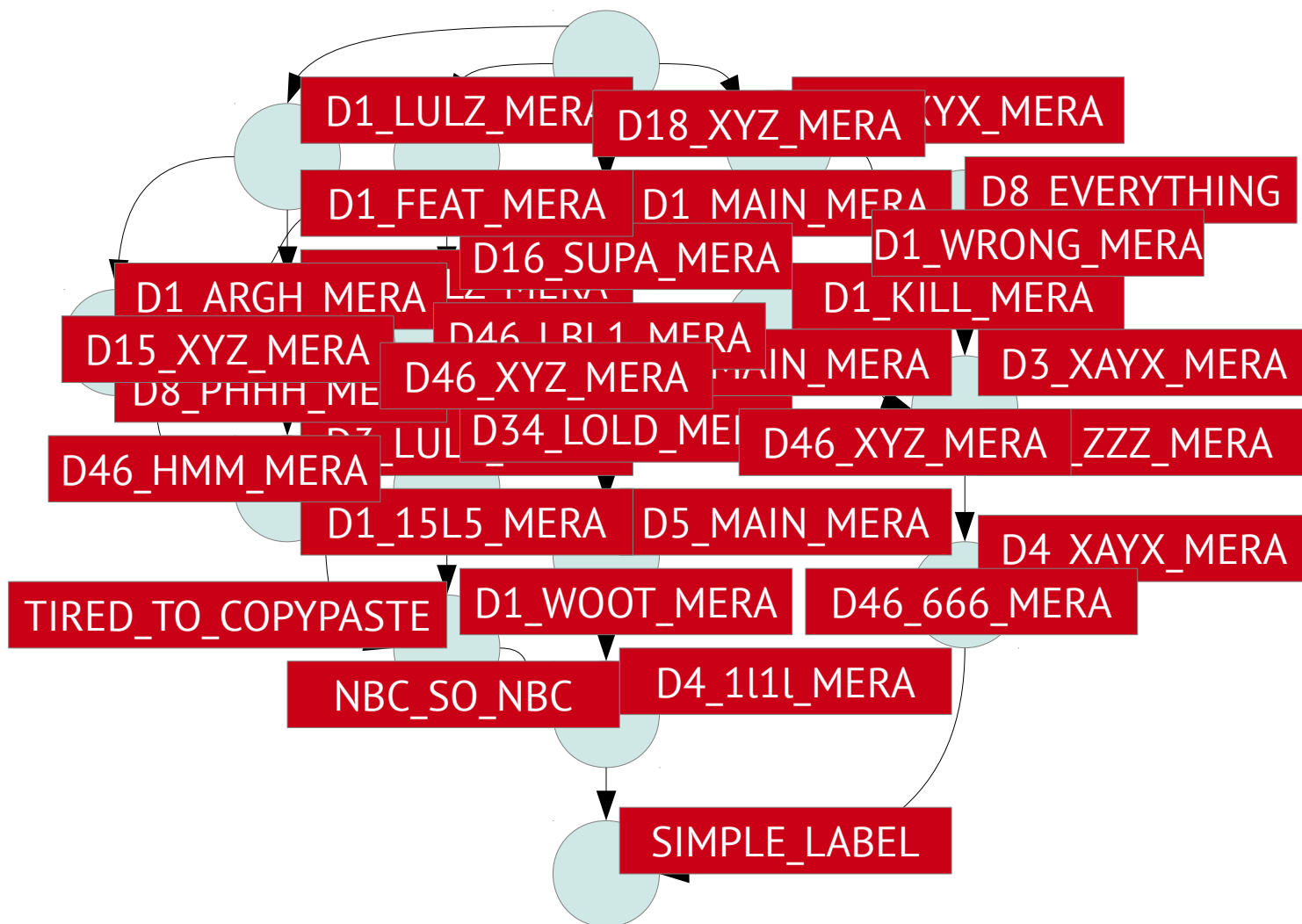
Git vs ClearCase



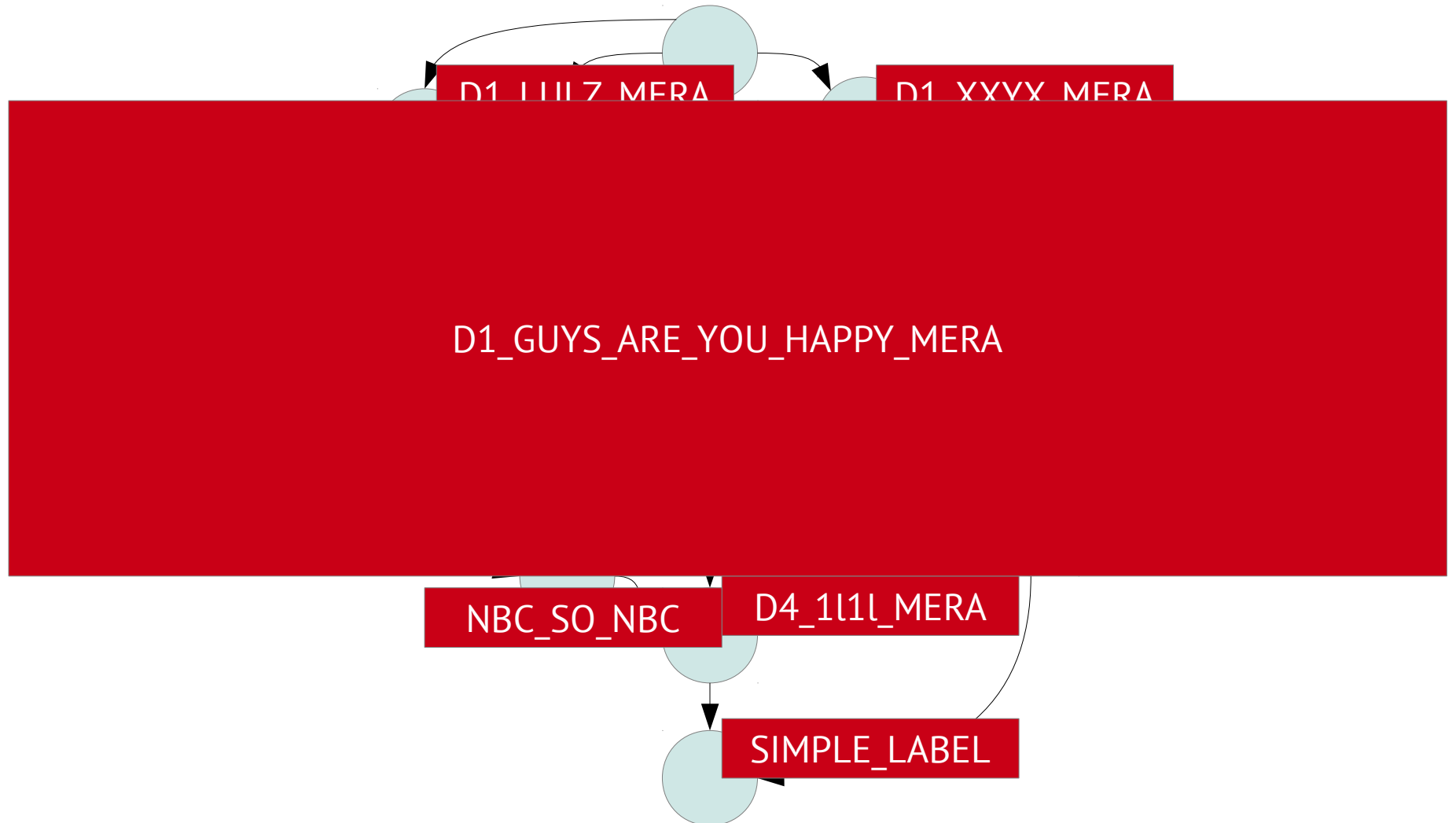
Git vs ClearCase



Git vs ClearCase



Git vs ClearCase



Are you really happy?

GIT vs ClearCase



Ordinary ClearCase user

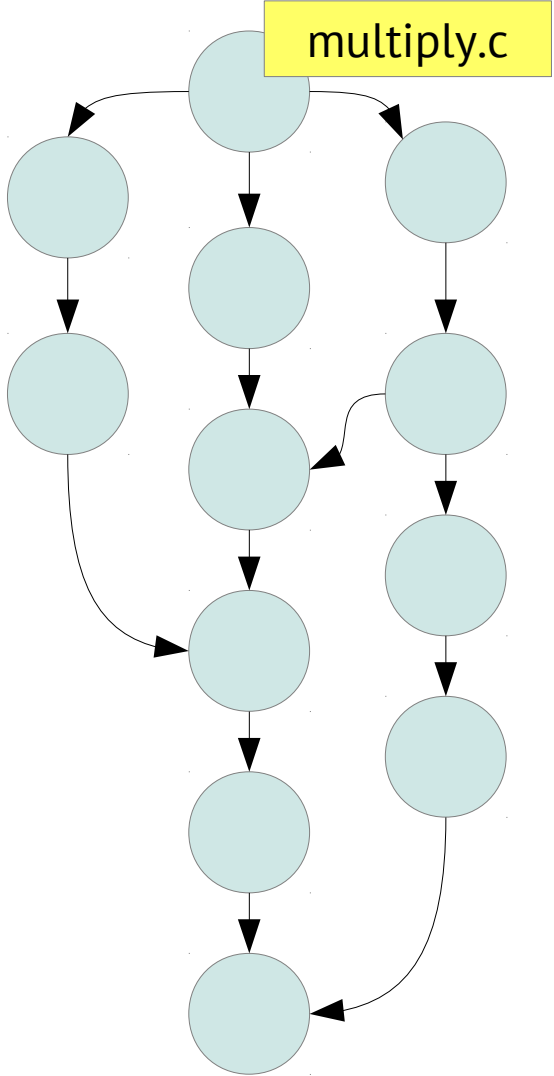
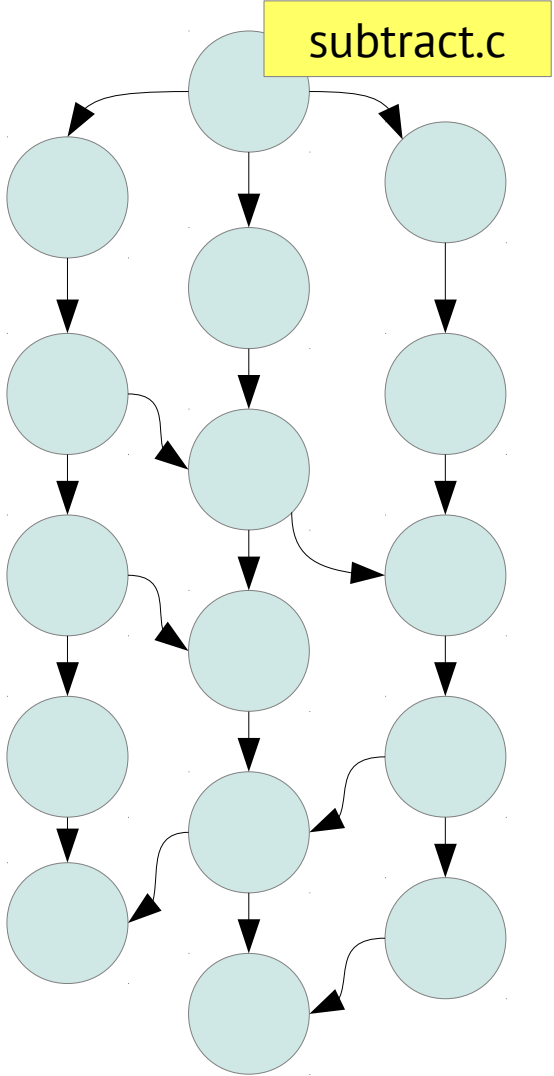
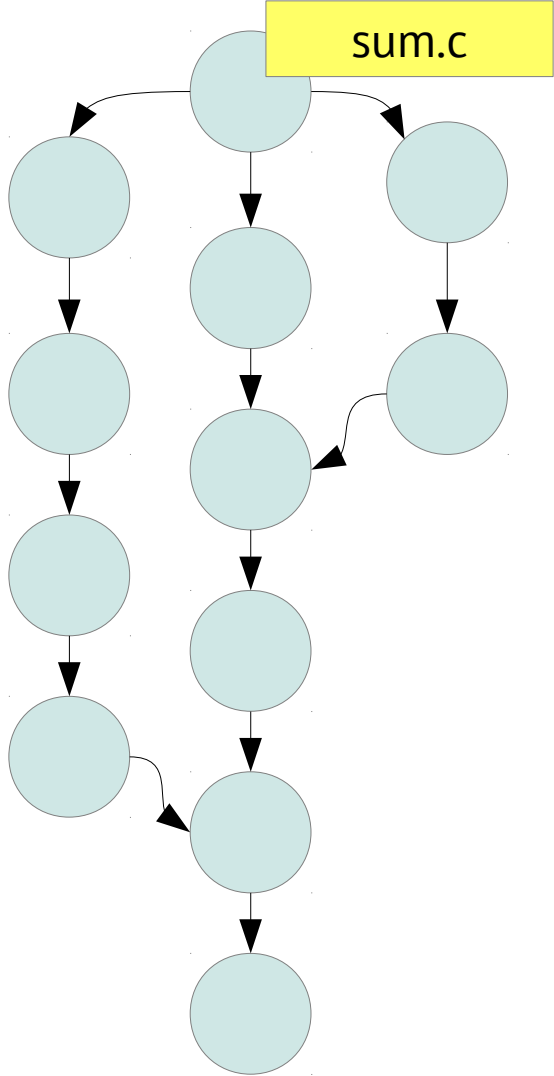
GIT vs ClearCase



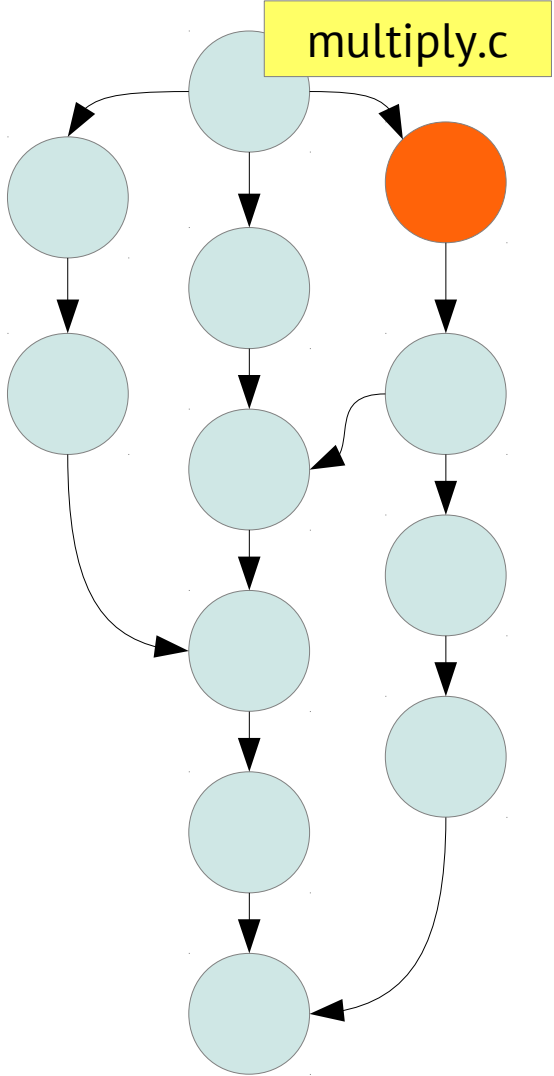
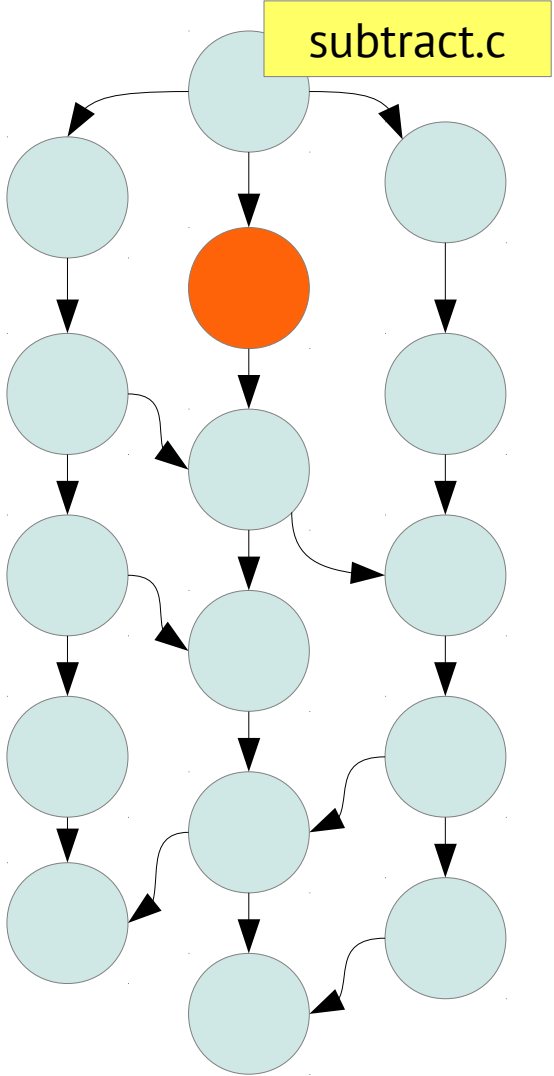
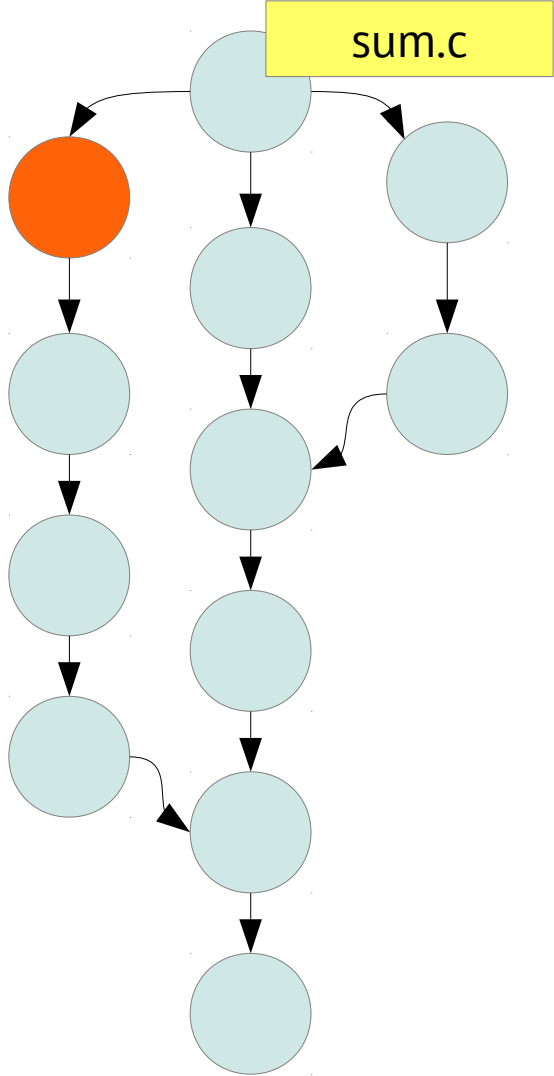
Ordinary GIT user

Have you ever realized
what your ClearCase view
really is?

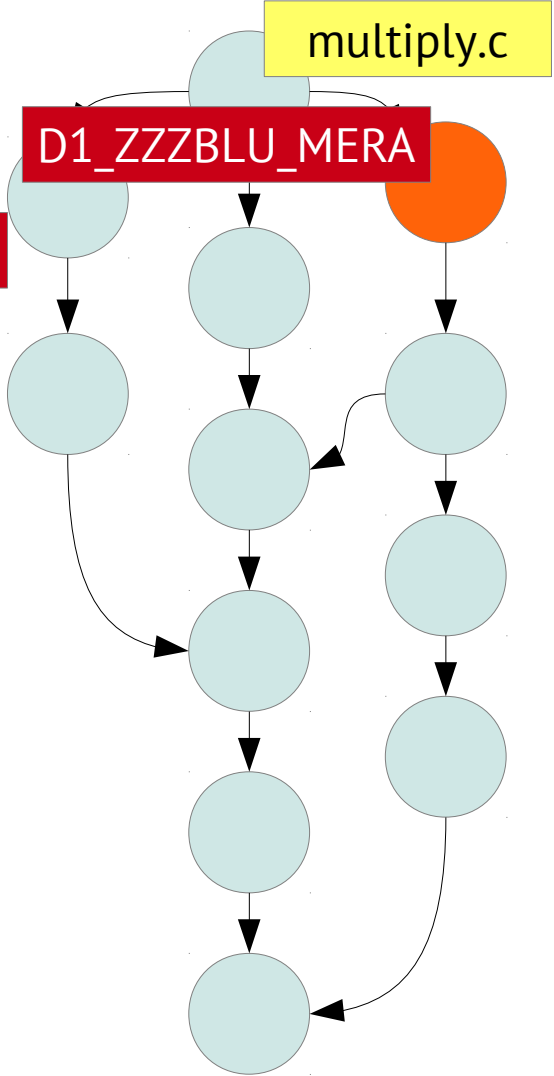
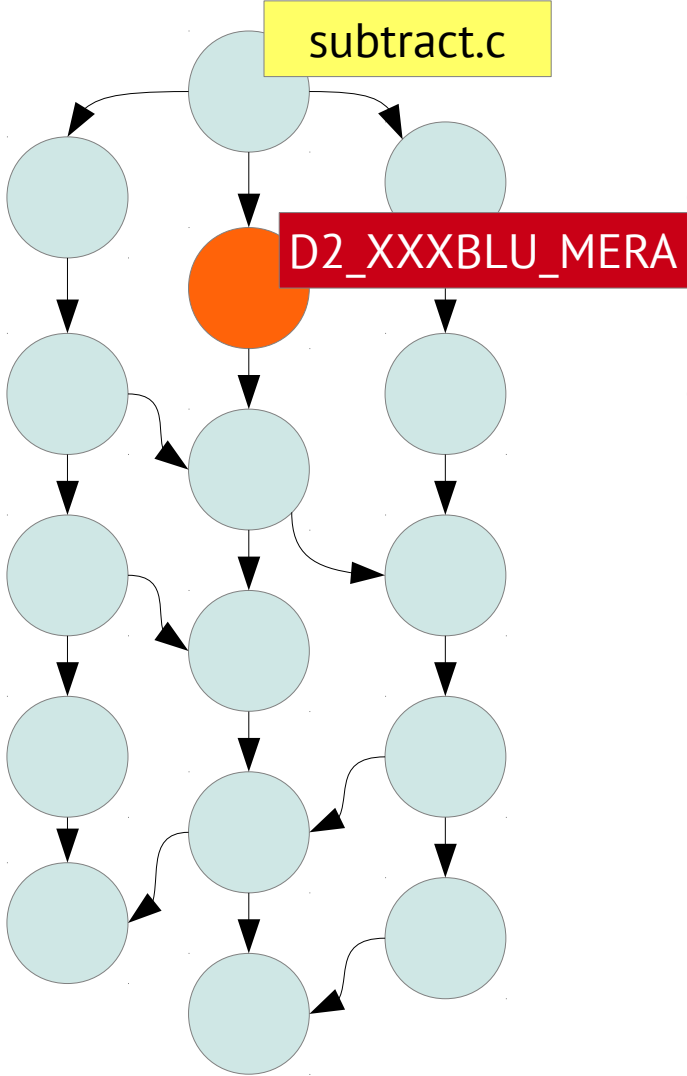
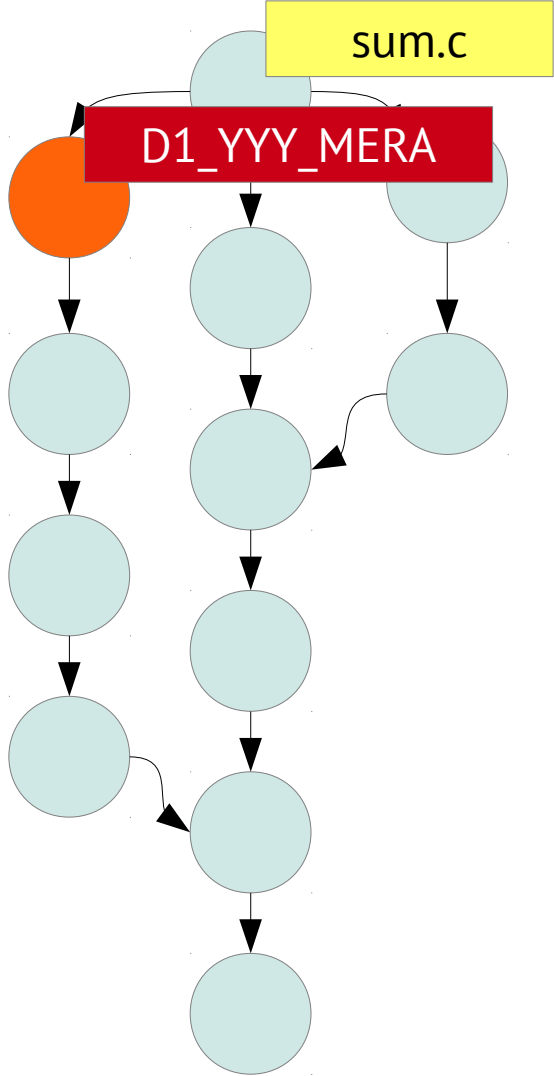
Git vs ClearCase



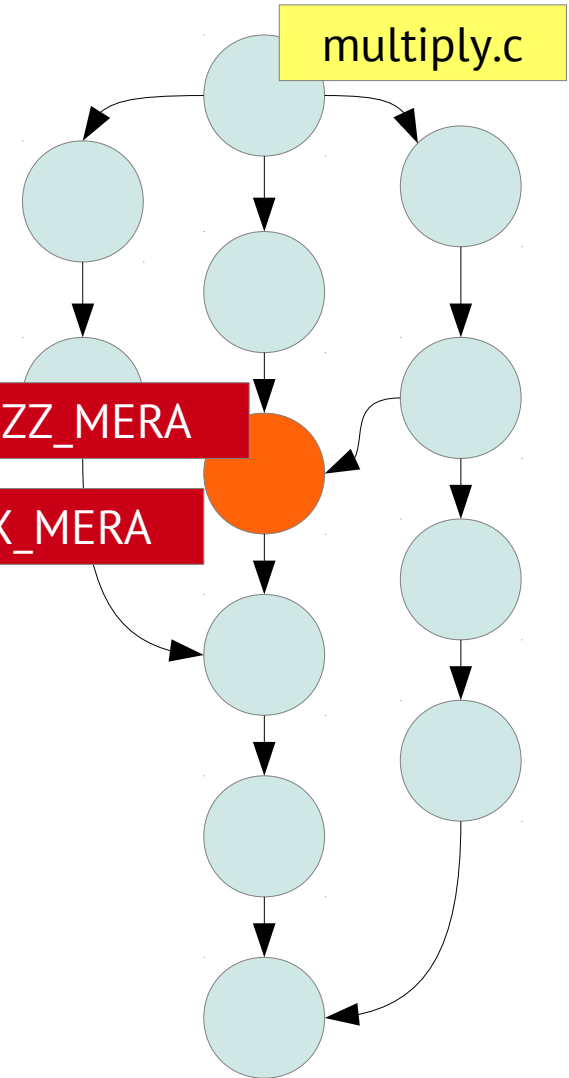
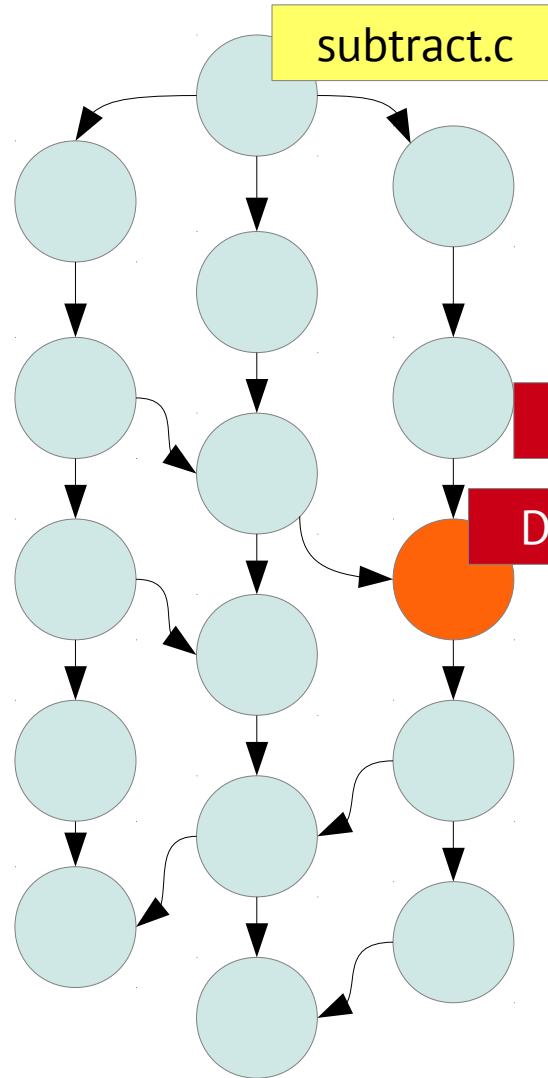
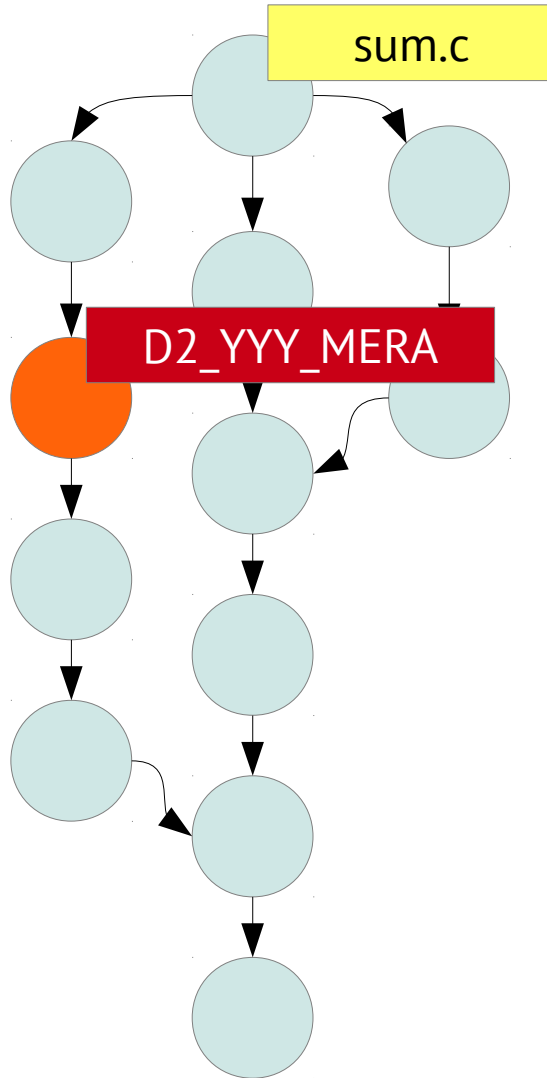
Git vs ClearCase



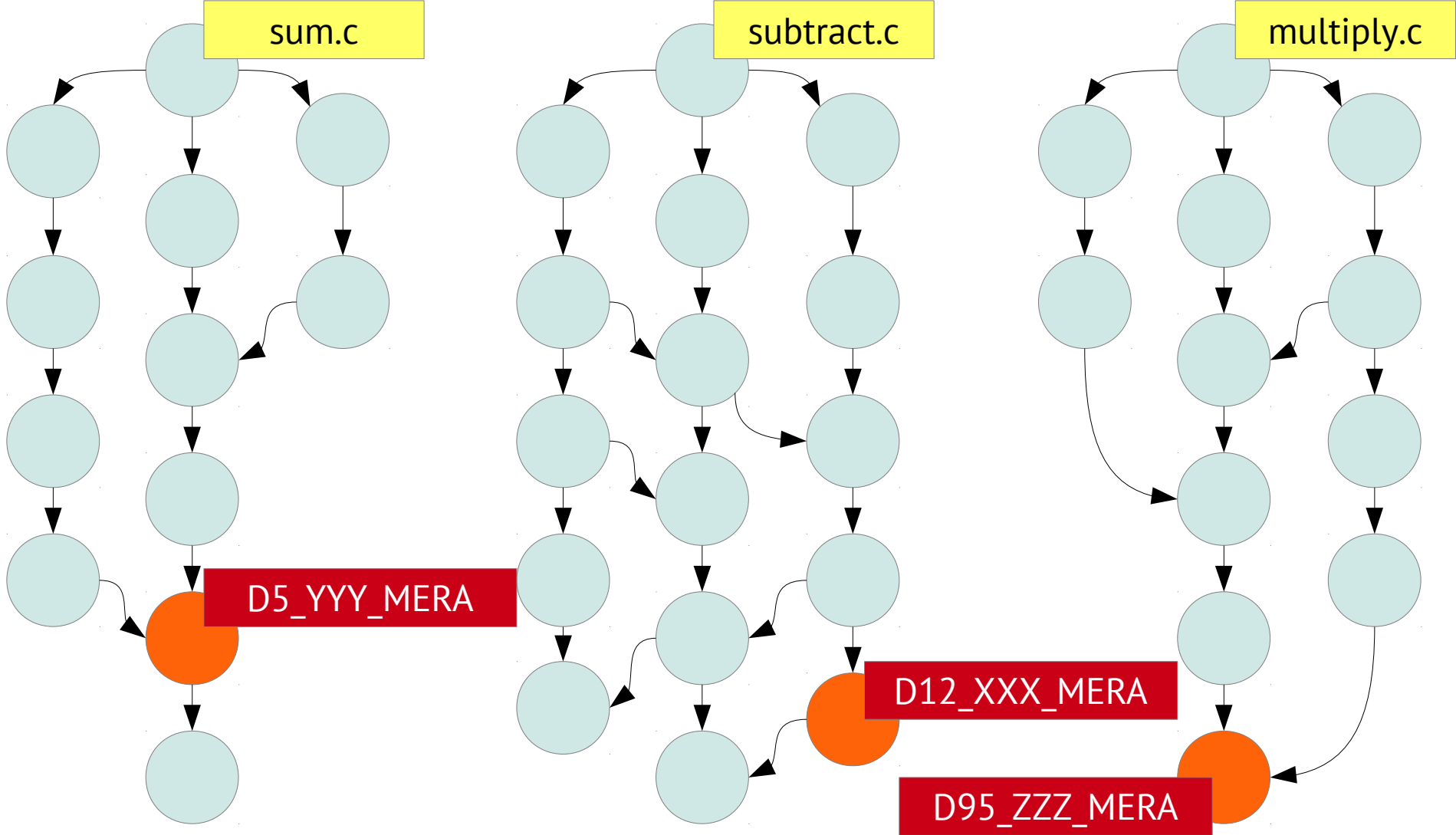
Git vs ClearCase



GIT vs ClearCase

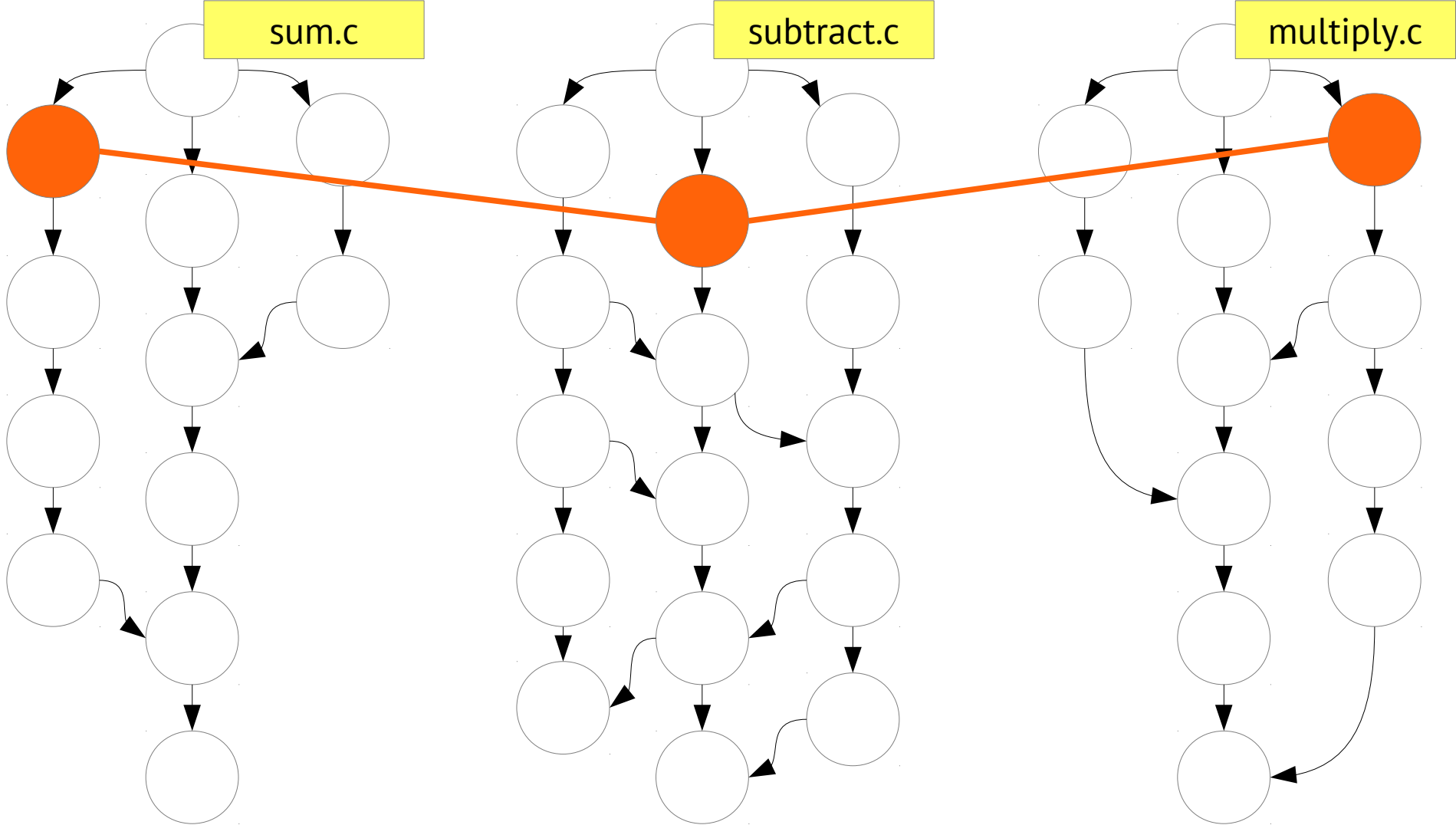


Git vs ClearCase

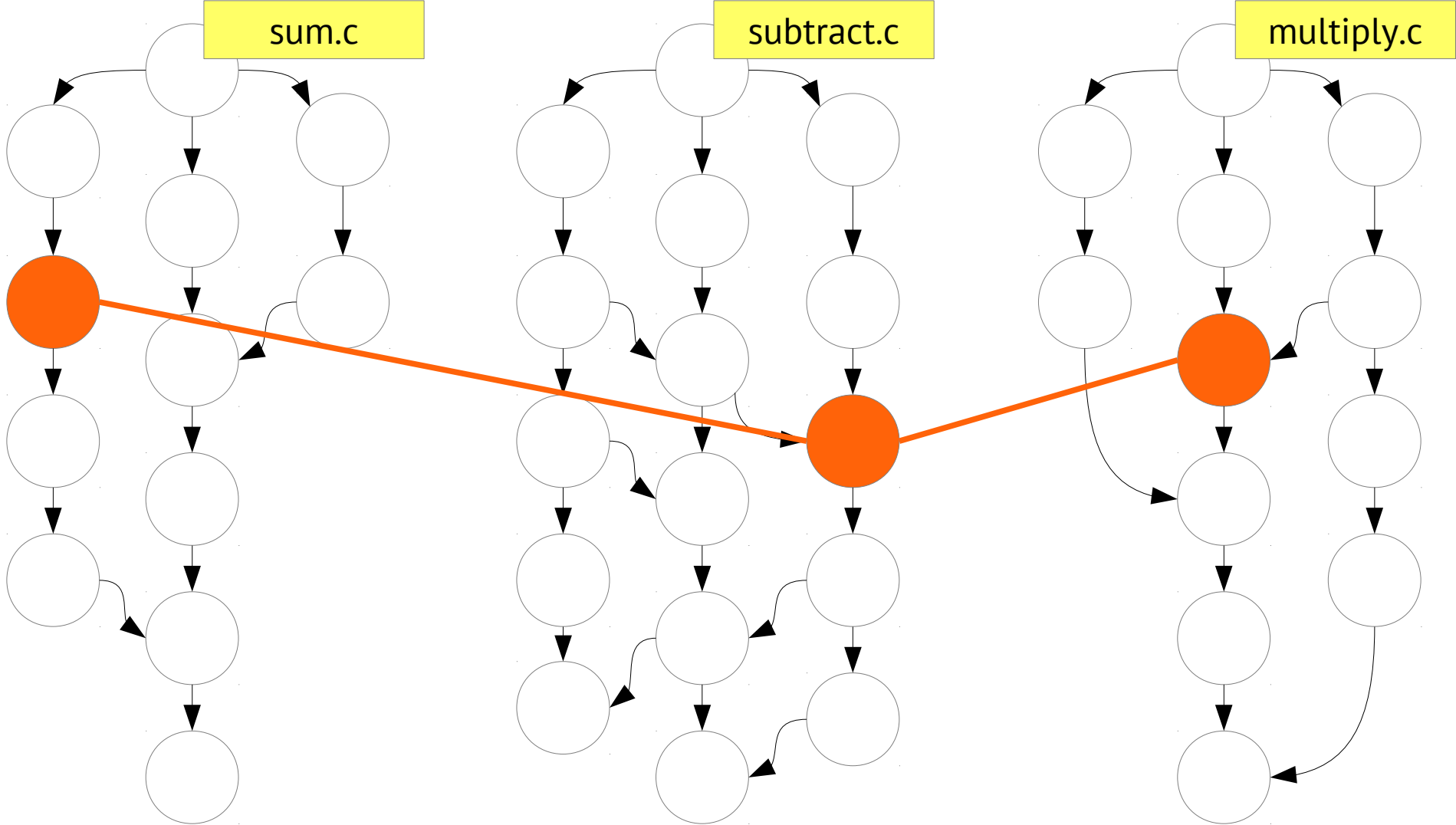


Any ideas about evolution of
your view?

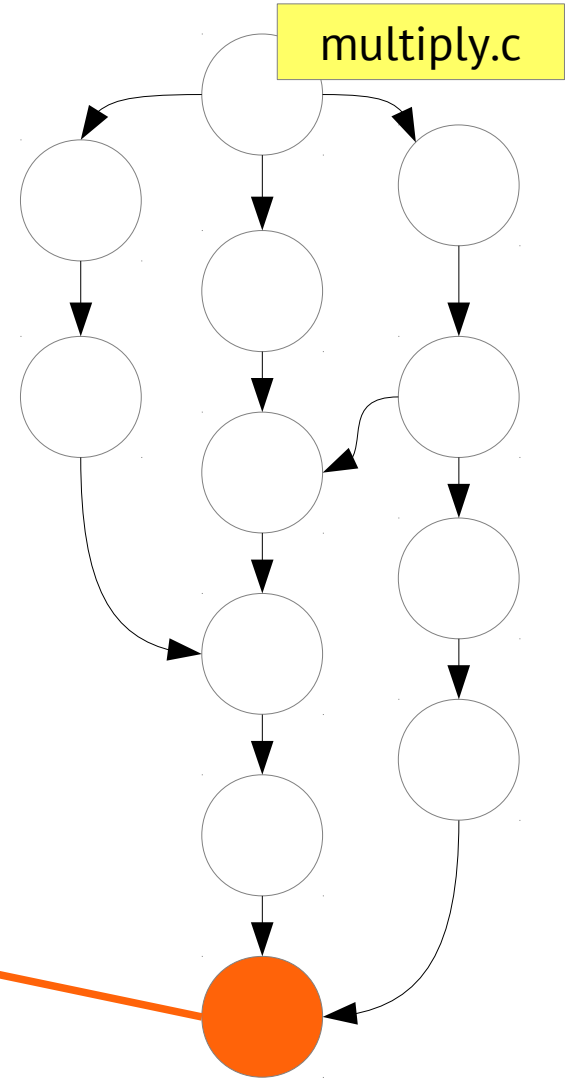
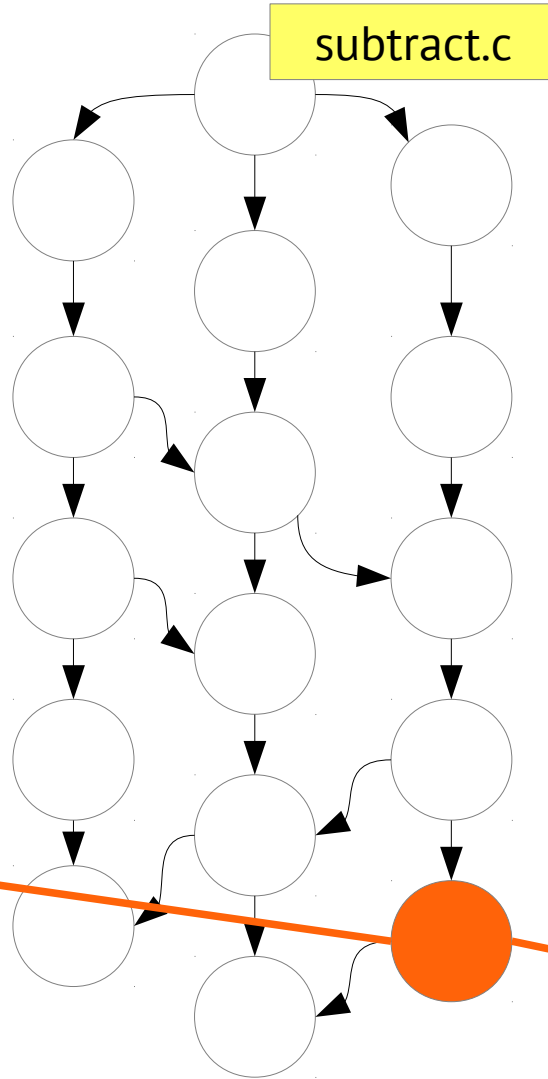
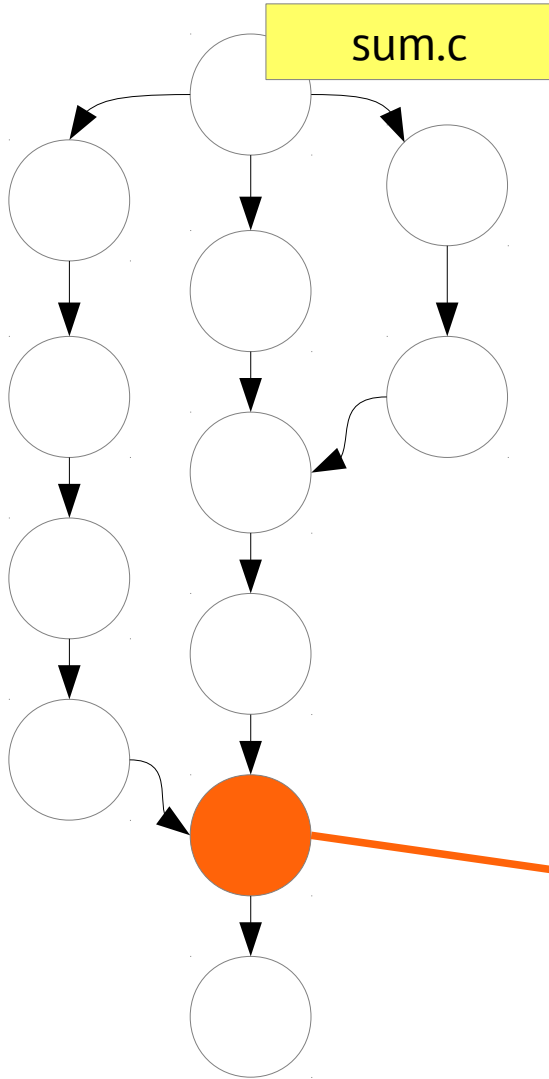
Git vs ClearCase



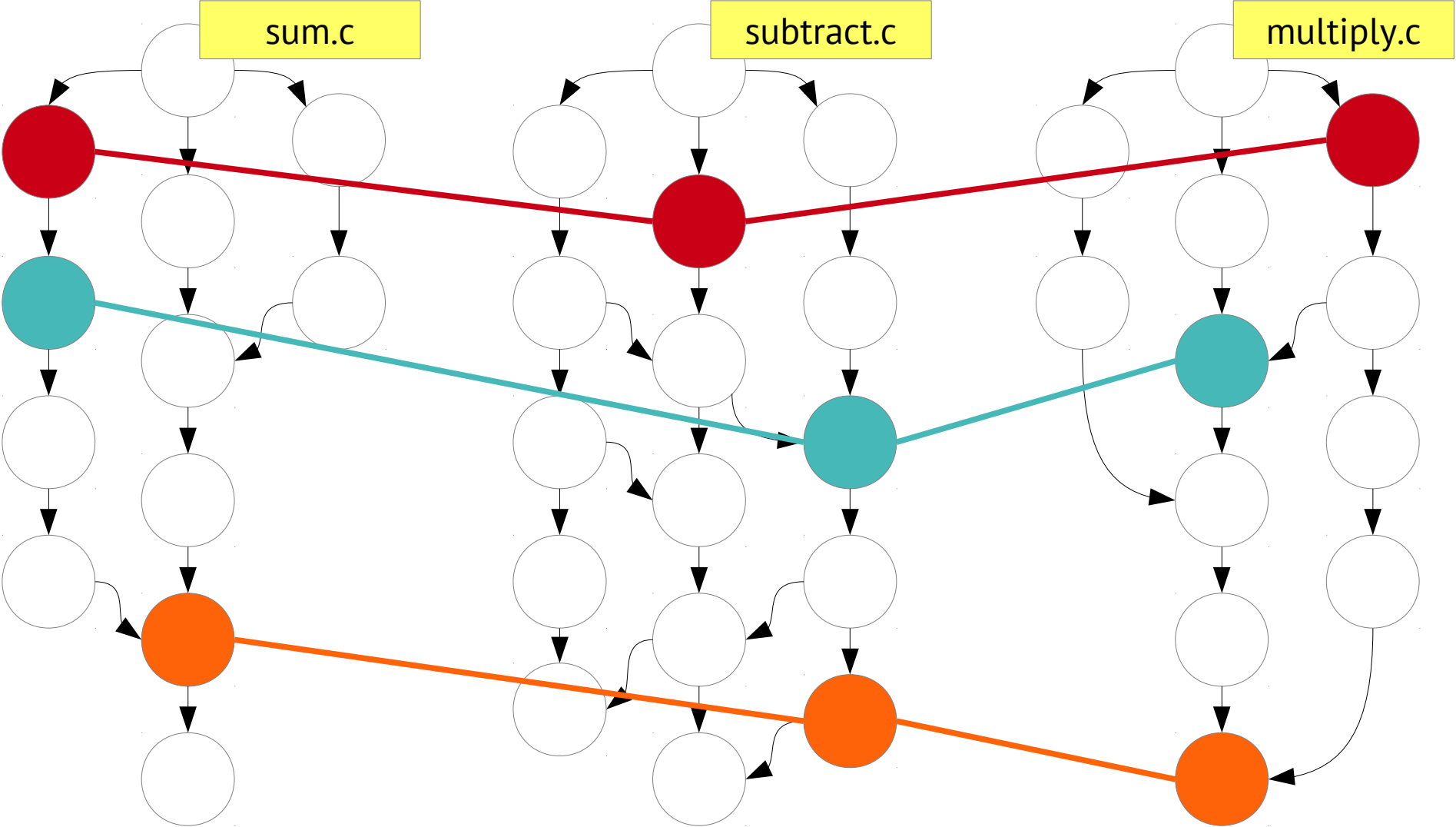
Git vs ClearCase



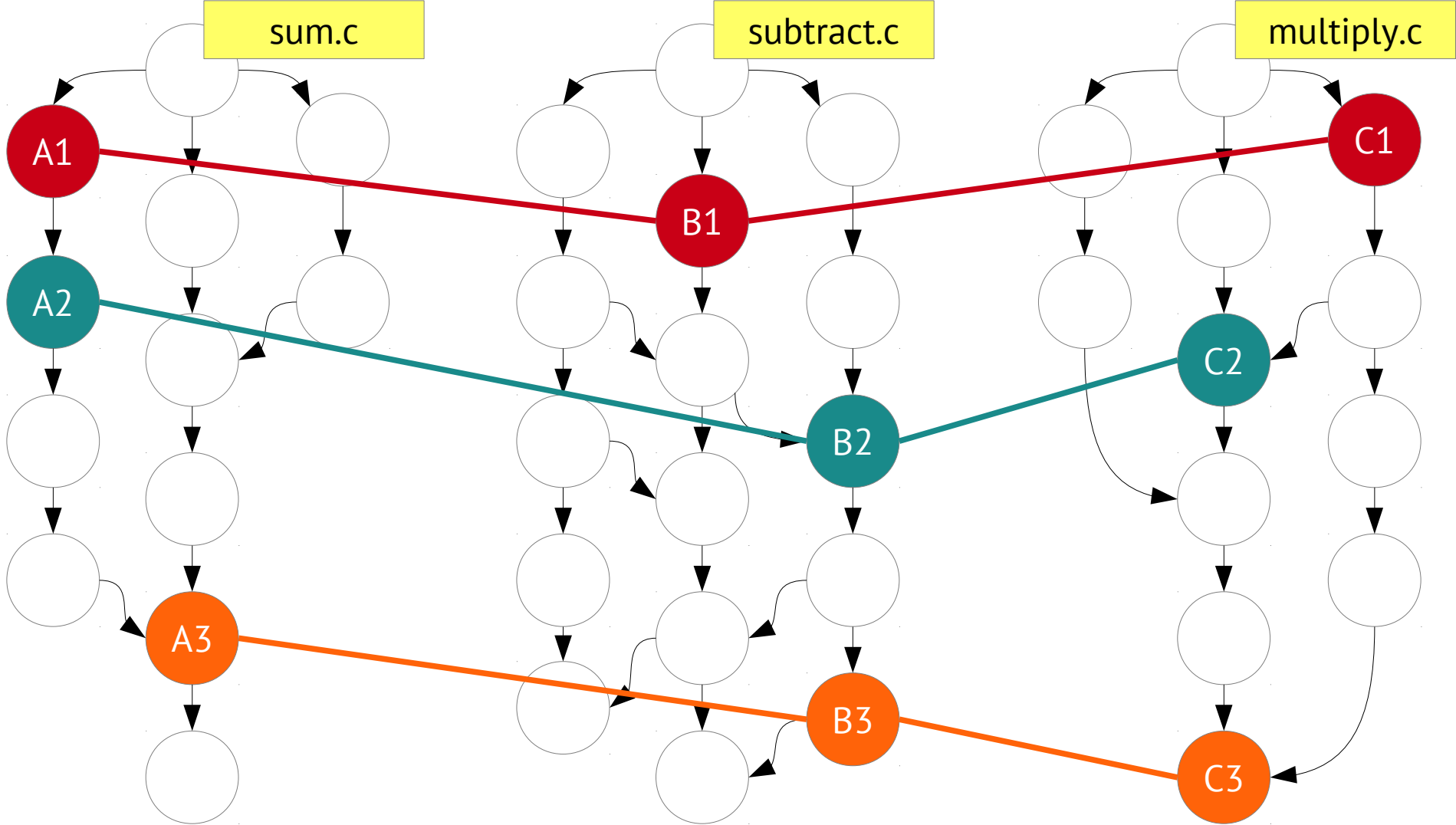
GIT vs ClearCase



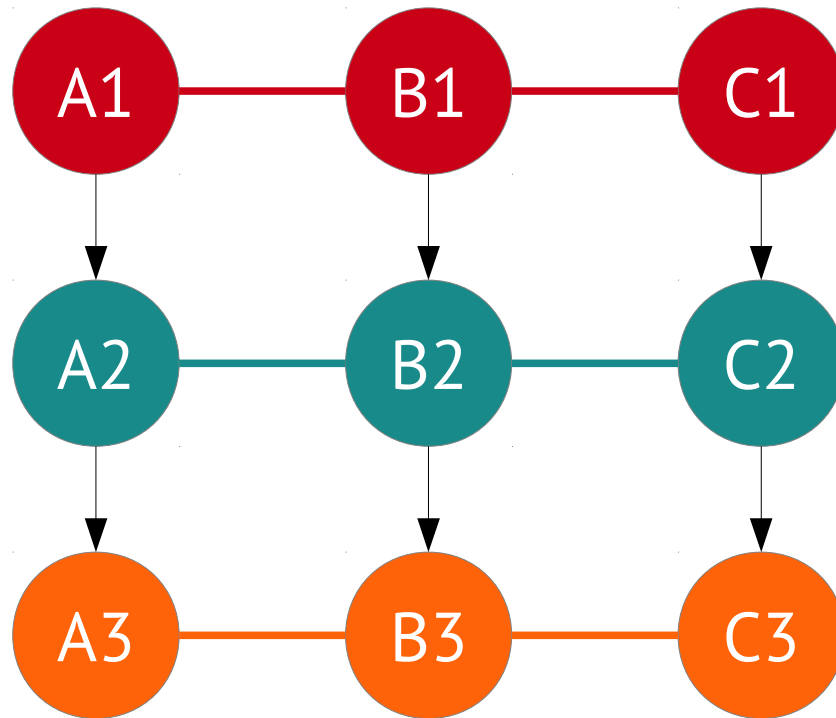
GIT vs ClearCase



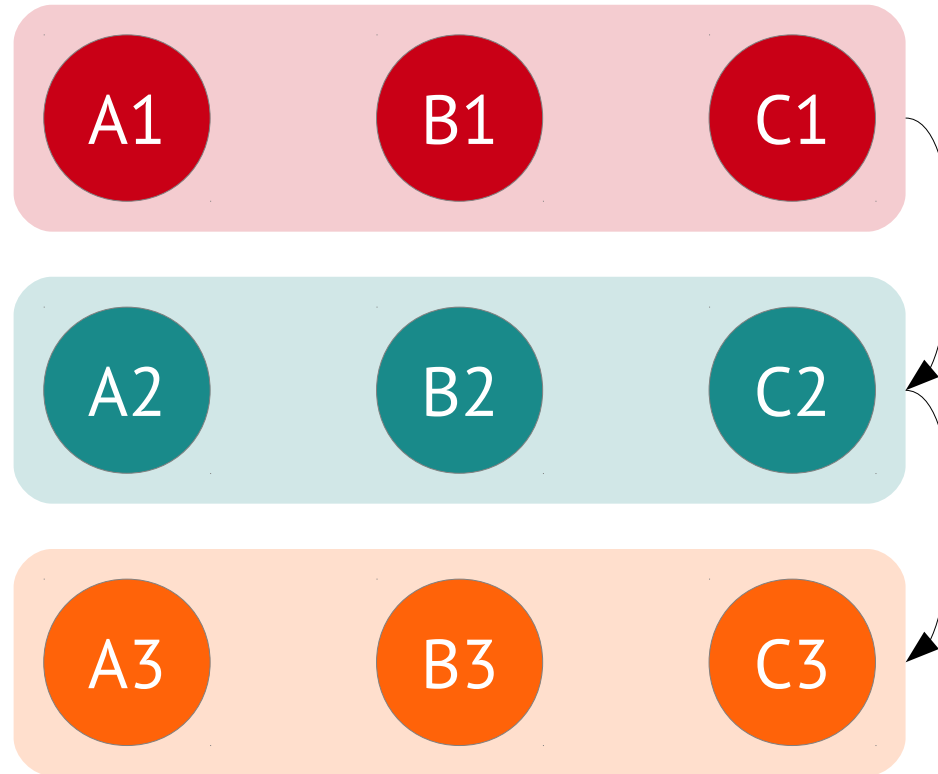
Git vs ClearCase



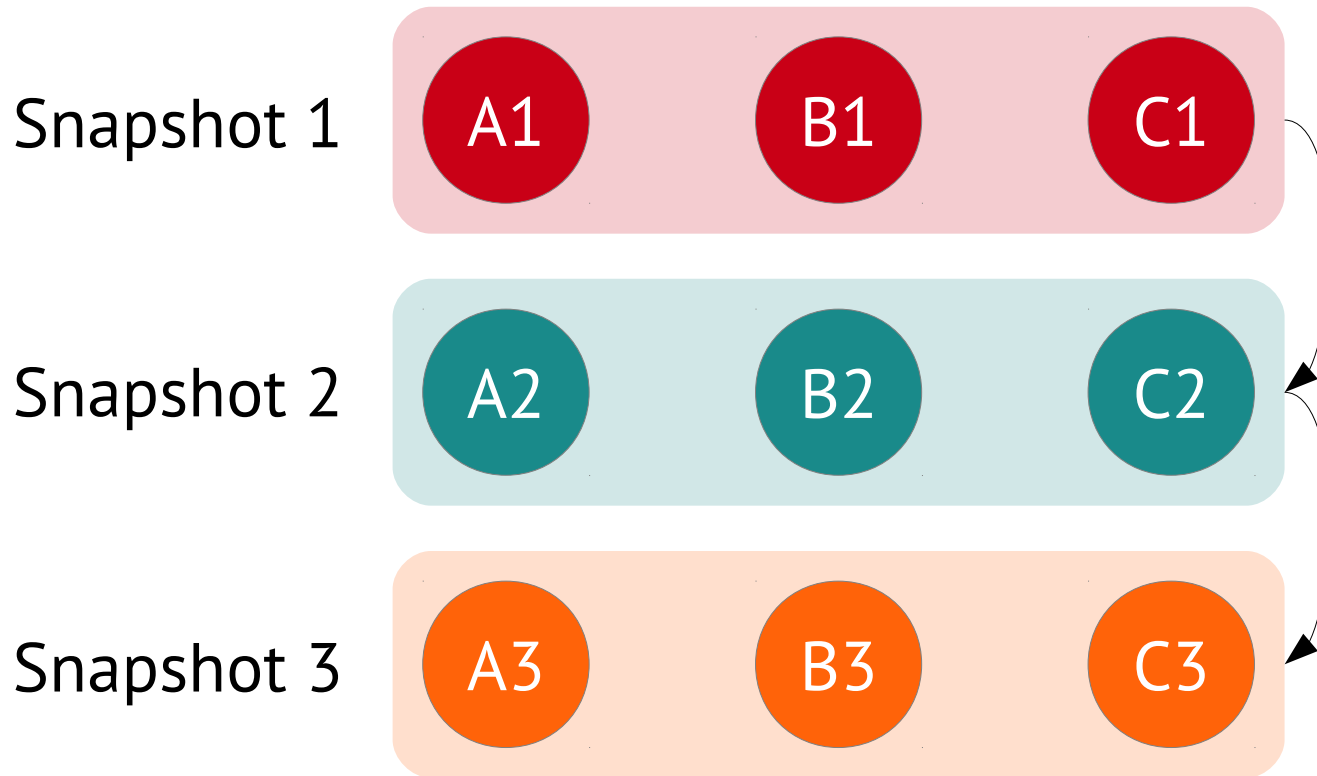
Git vs ClearCase



Git vs ClearCase



Git vs ClearCase

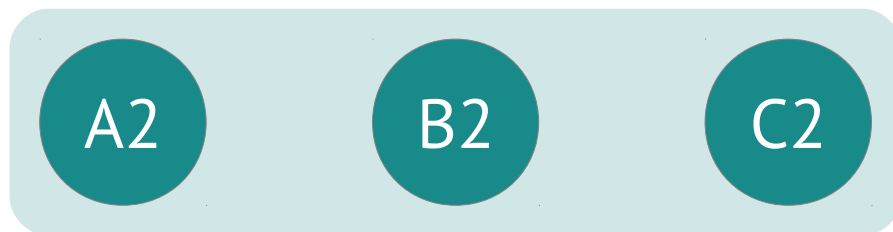


Git vs ClearCase

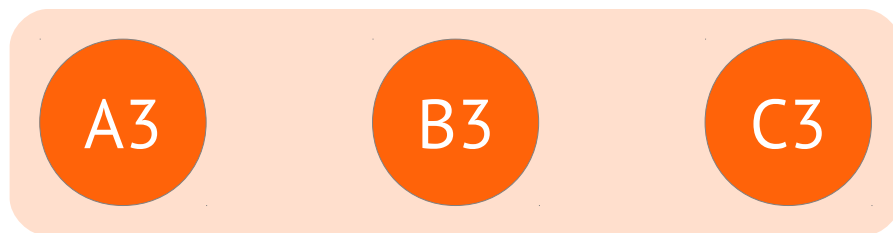
Snapshot ₁



Snapshot ₂



Snapshot ₃



Do not lie yourself.
You do not deal with labels.
You deal with snapshots.

GIT vs ClearCase

- 'Baseline' means 'snapshot' in GIT terms.
- But to deliver you correct snapshots there is the team of masterminds who supports label system and makes centralized system redundant.
- Each config spec includes over 9000 different labels to create 1 snapshot.

Over 9000 labels
to create 1 snapshot

GIT vs ClearCase

- And if you want to extend your snapshot, you need to add much more additional rules.
- Each rule defines how your local copy of ClearCase will have to request information about certain file or diff by network.
- Each operation is performed over the network which causes mortal delays. Everything is very slow.
- Without network you cannot even open file or build something.
- You need to explain your ClearCase by unreadable labels and awkward rules time after time which snapshot you need instead of straight using it.

GIT vs ClearCase

GIT have no such restrictions by design

- GIT initially operates snapshots, there is no need in any explicit determining which set of labels you need.
- Everything is local. You may pull changes from master repository and continue your work even in aircraft, with branches and whole development story. You may check earliest version even you lost in Tundra without any internet signs.
- Your repository is your own game. You can develop in any way you want. Just send your changes into central repository at time.
- Operations are really rapid and easy. If you want to select LSV or branch, you do not need to ask central repository for proper set of rules. Just use it!

Examples

GIT vs ClearCase

Choosing baseline

ClearCase way

\$ select_config CPP_DESIGN

waiting for 2 minutes

selecting latest baseline

having a cup of tea

excluding components

Pushing 'Set' button

waiting for 2 minutes

closing application

waiting for 1 minute

working

Git way

\$ git checkout BASELINE

working

GIT vs ClearCase

Branching

ClearCase way

```
$ ct mkbrtype hardcore_branch
```

```
$ ctedcs
```

Typing something like

```
element * CHECKEDOUT
```

```
element <path> .../hardcore_branch/LATEST
```

```
element <path> /main/cppdev/LATEST  
-mkbranch hardcore_branch
```

```
$ ct co -nc <path>/<file>
```

Working if noone checkedout a file

Git way

```
$ git checkout -b hardcore_branch
```

working

GIT vs ClearCase

Building your software

ClearCase

```
$ cat ./cc_build.sh
```

```
ct_set_needed_cs -version  
<LSV> -baseline <BASELINE>
```

```
cd <path to one our famed  
component>
```

```
<build command>
```

GIT

```
$ cat ./git_build.sh
```

```
git checkout BASELINE
```

```
cd <path to one our famed  
component>
```

```
<build command>
```

GIT vs ClearCase

Building your software

ClearCase

```
$ time -p ./cc_build.sh >  
/dev/null
```

```
real 1053
```

```
user 266
```

```
sys 78.72
```

GIT

```
$ time -p ./git_build.sh >  
/dev/null
```

```
real 58.28
```

```
user 21.20
```

```
sys 35.14
```

GIT vs ClearCase

Building your software

ClearCase

```
$ time -p ./cc_build.sh >  
/dev/null
```

real 1053

user 266

sys 78.72

GIT

```
$ time -p ./git_build.sh >  
/dev/null
```

real 58.28

user 21.20

sys 35.14

17.5 minutes vs 1 minute!

Do you really like ClearCase?

Or you just have not used
something completely different
before?

GIT vs ClearCase

- GIT support your **actual way** of developing software **by design**. You do not need to juggle tons of labels being in a jail of ClearCase ancient restrictions.
- ClearCase does not allow to use modern convenient ways of developing 'out of box'. It provides you with hardcore methods for hacking it only.
- Everytime you need something from ClearCase, you need to modify your config spec which already contains text with size around a two volumes of 'War and Peace' book instead of telling it what you really want to get.

**You do not use ClearCase to simplify your job,
everytime you appeal to it to get anything.**

Fully distributed

What is GIT?

Git is an open source,
distributed version control
system designed for speed
and efficiency.

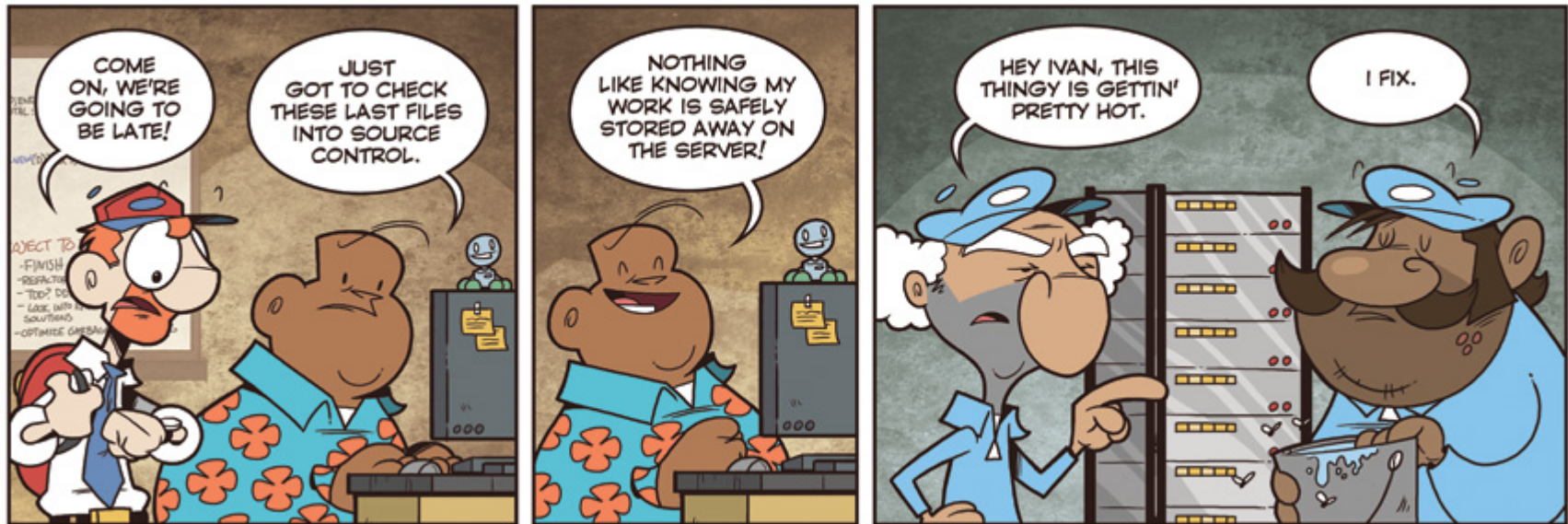
Fully distributed

(almost) everything is local

which means

- everything is fast;
- every clone is backup;
- work offline;
- every known remote repo is branch;

Fully distributed



Fully distributed

Integration
repository

Trunk
repository

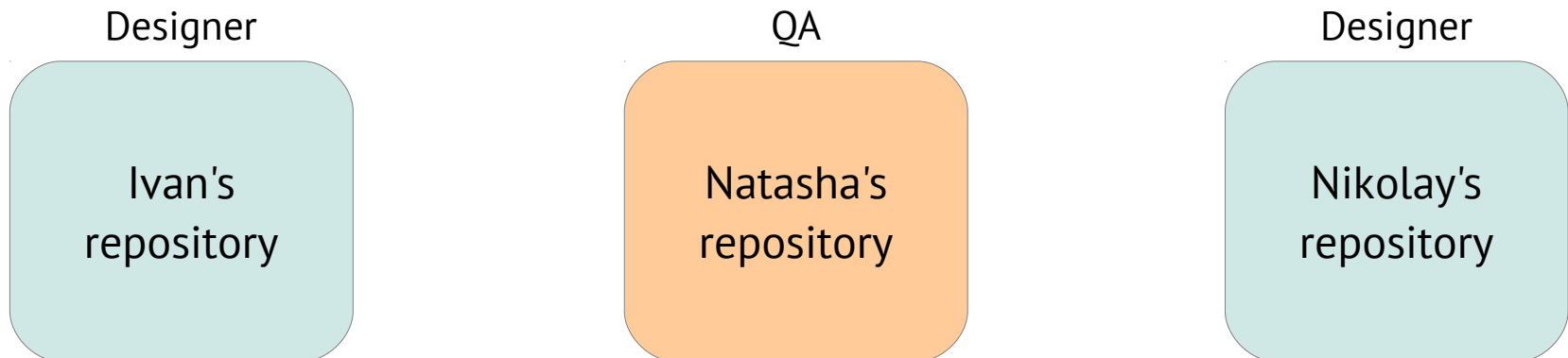
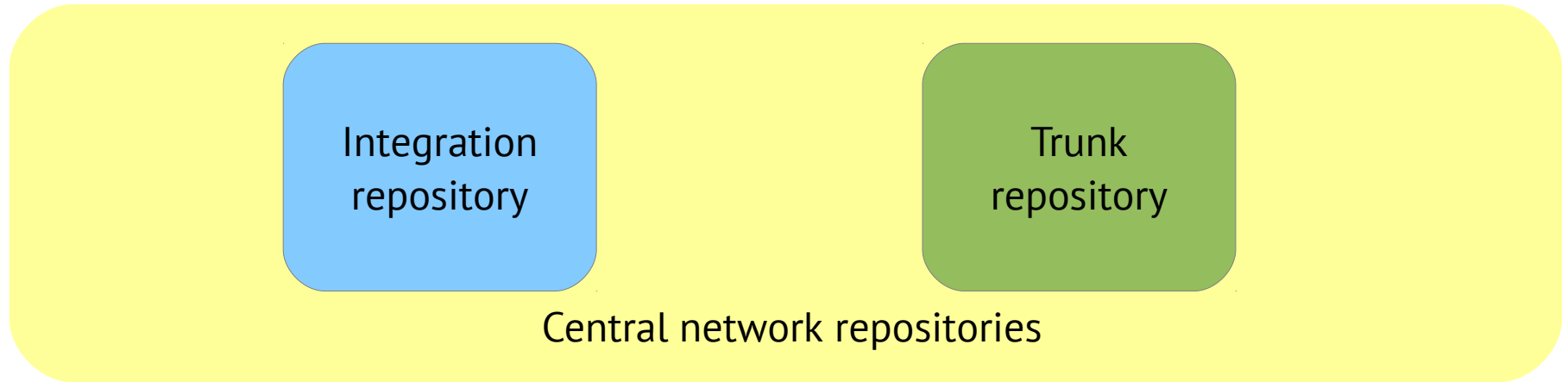
Central network repositories

Ivan's
repository

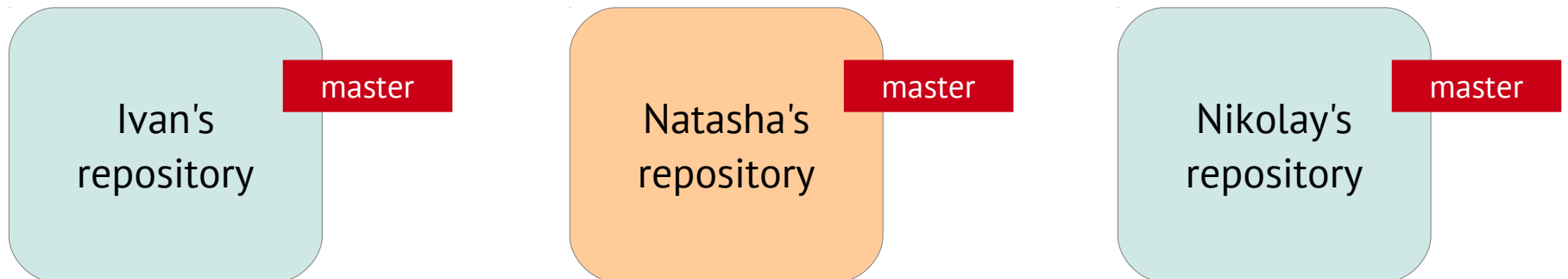
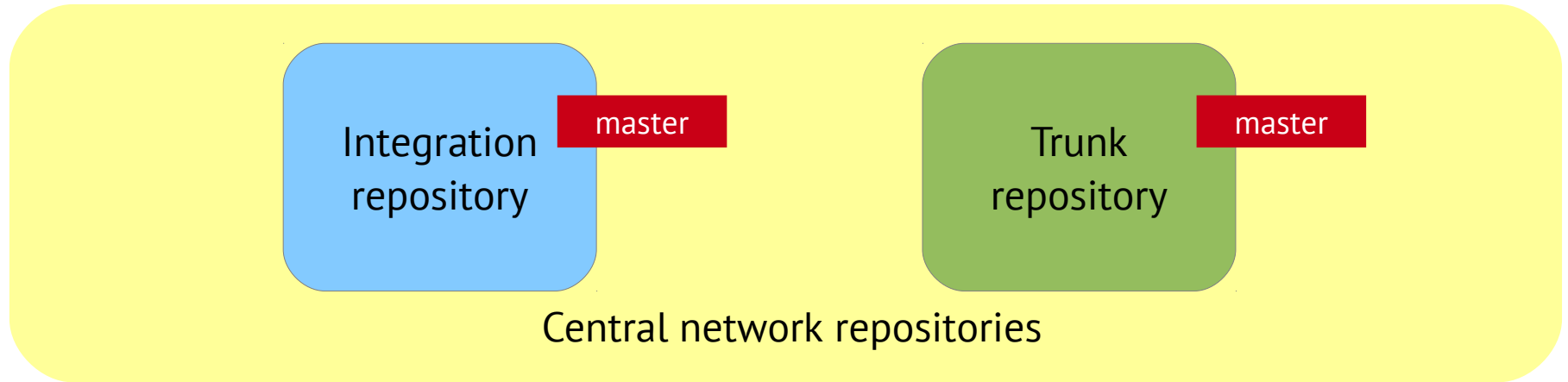
Natasha's
repository

Nikolay's
repository

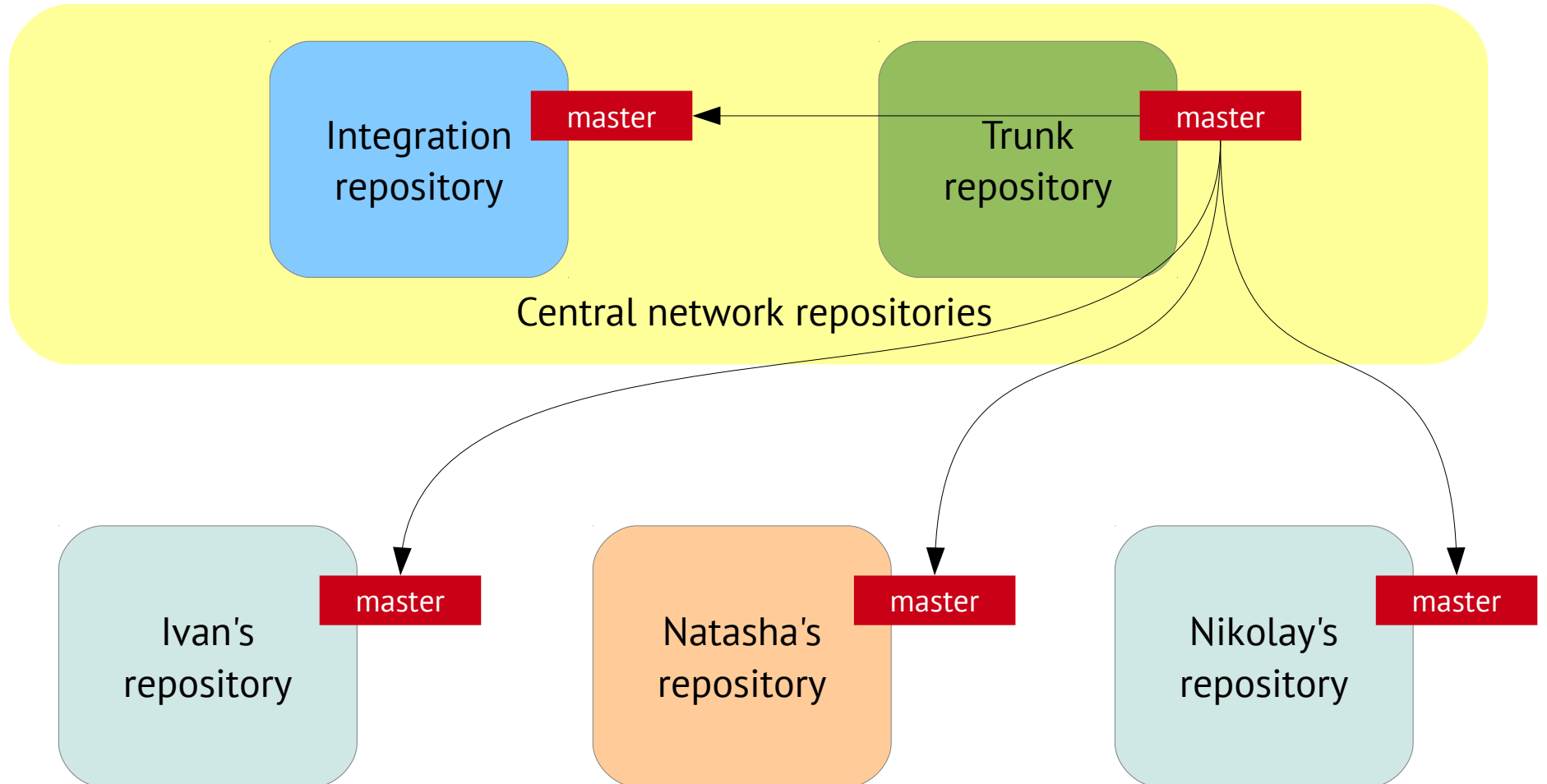
Fully distributed



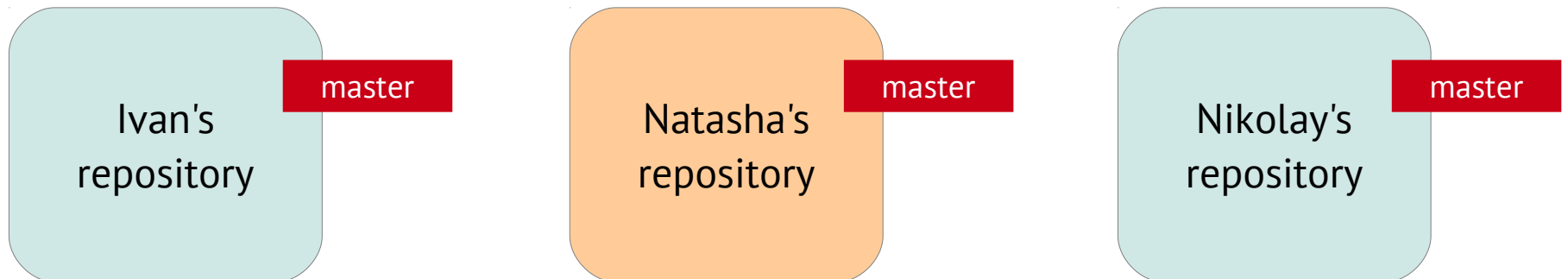
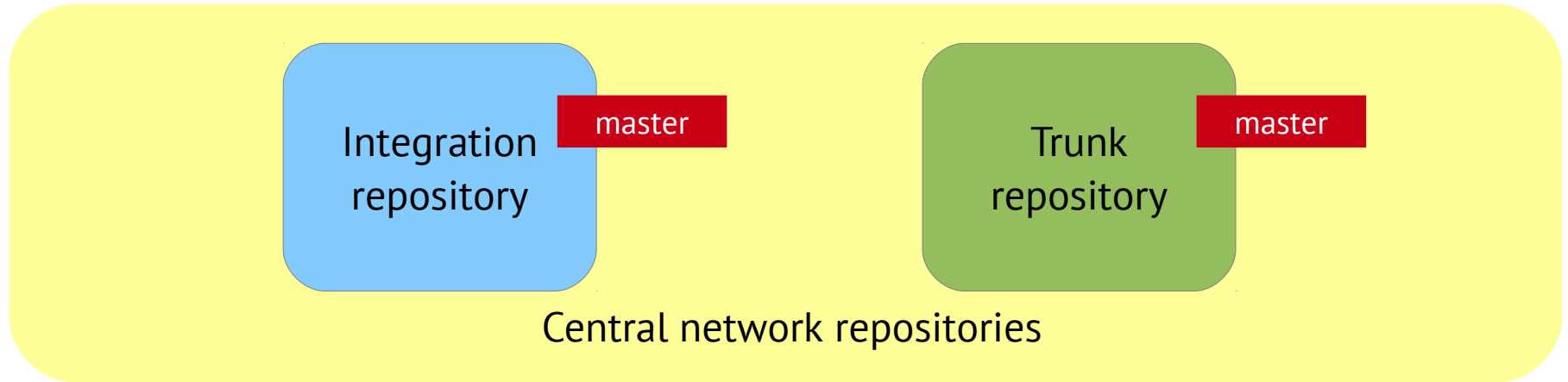
Fully distributed



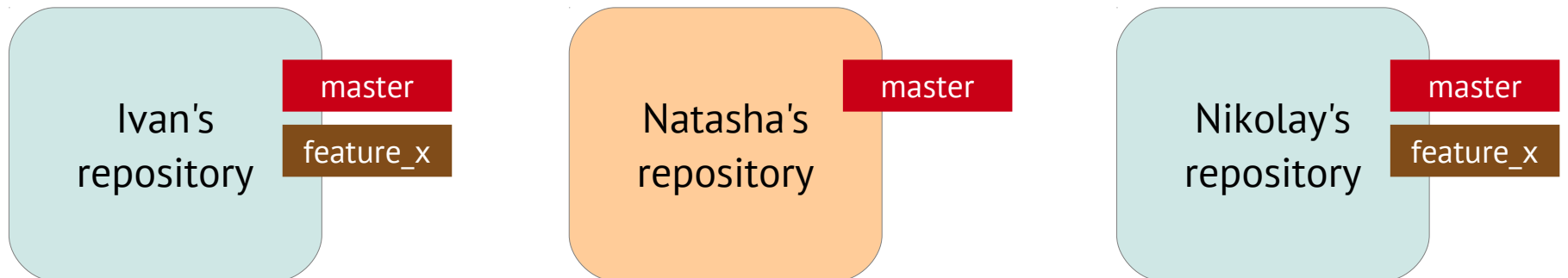
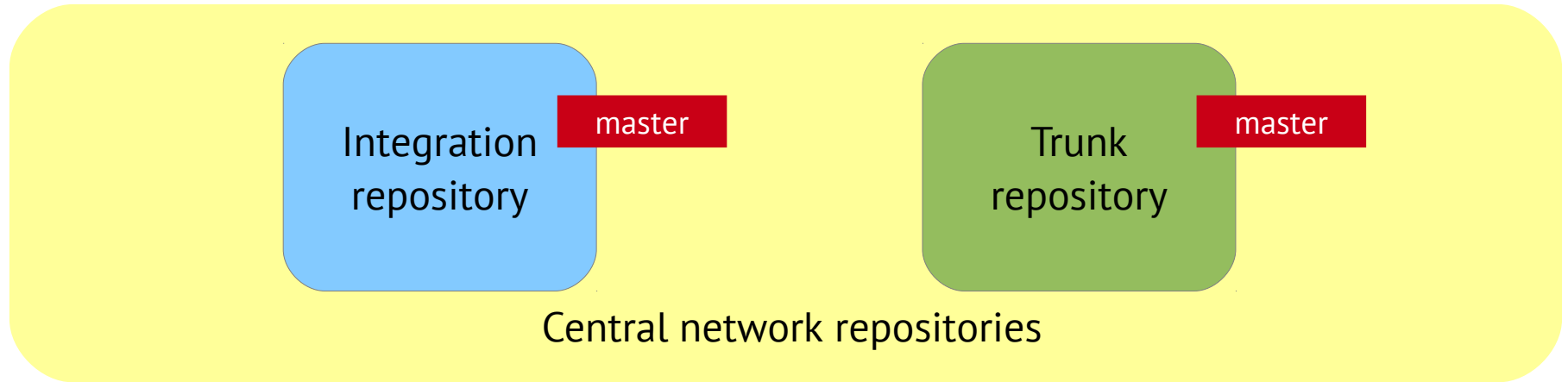
Fully distributed



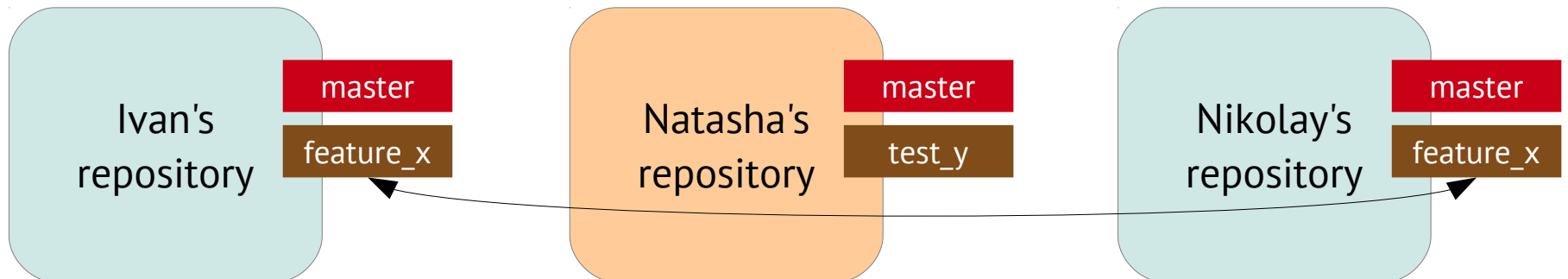
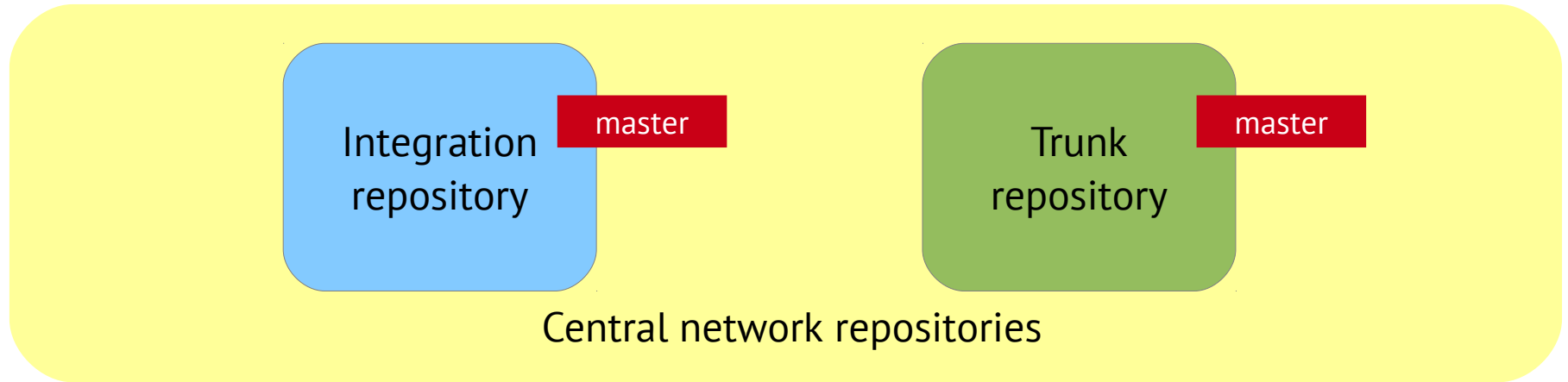
Fully distributed



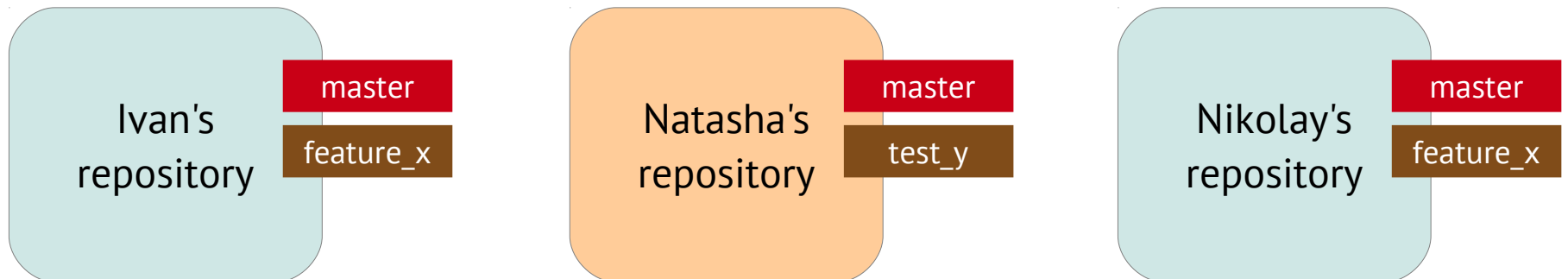
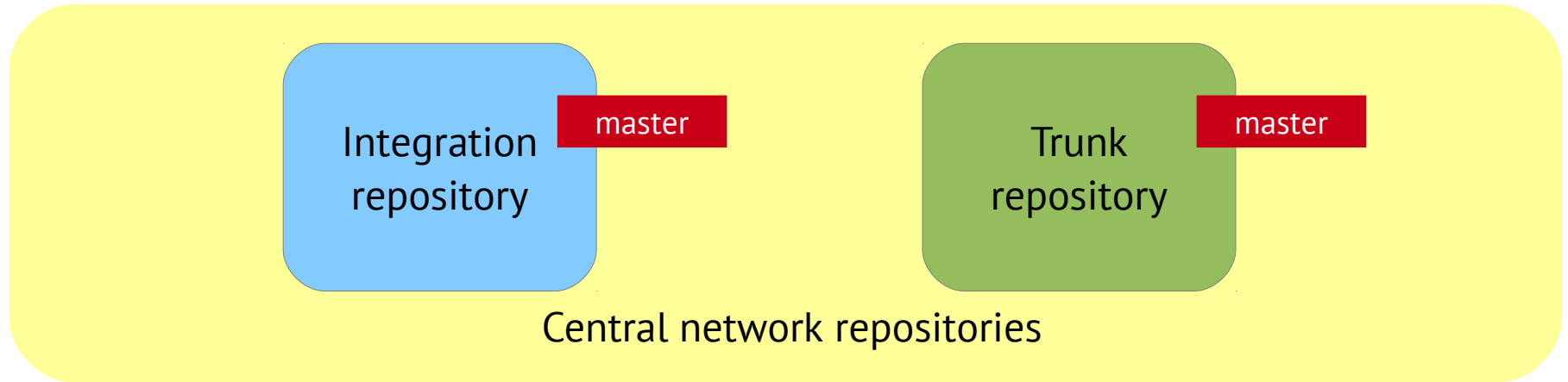
Fully distributed



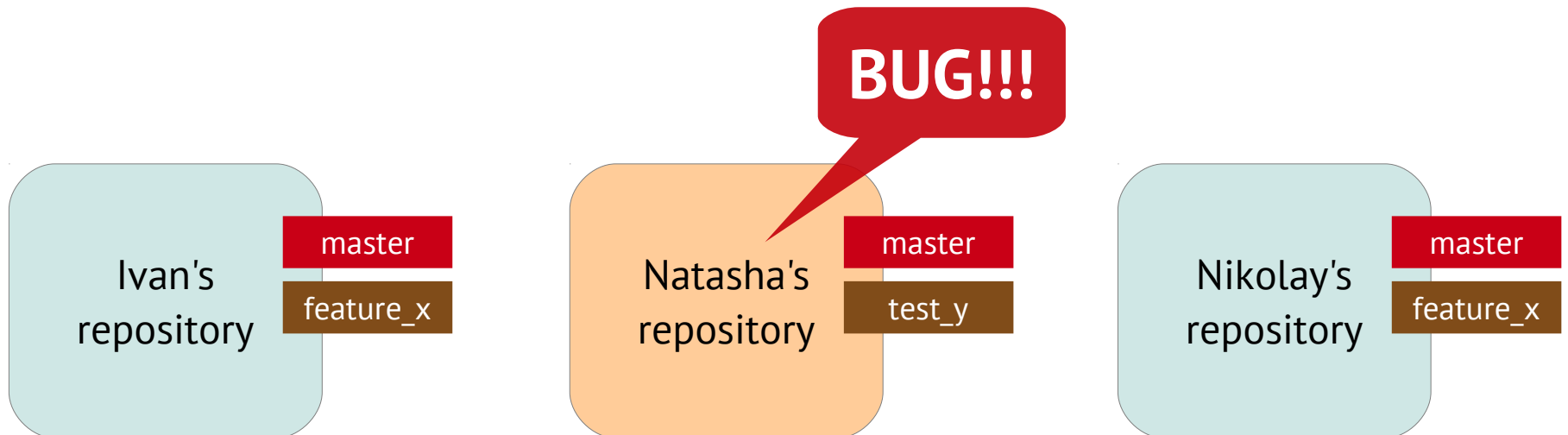
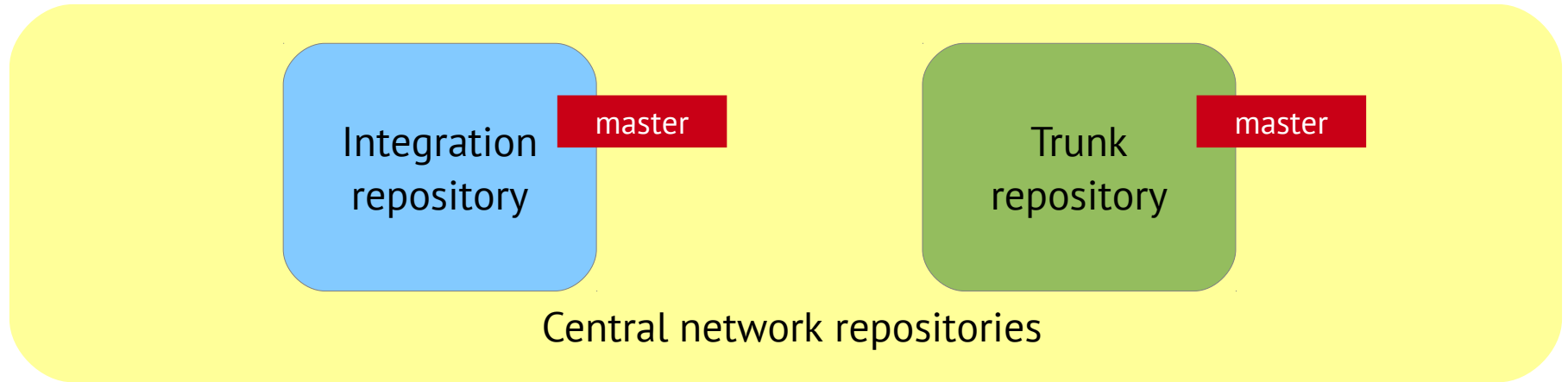
Fully distributed



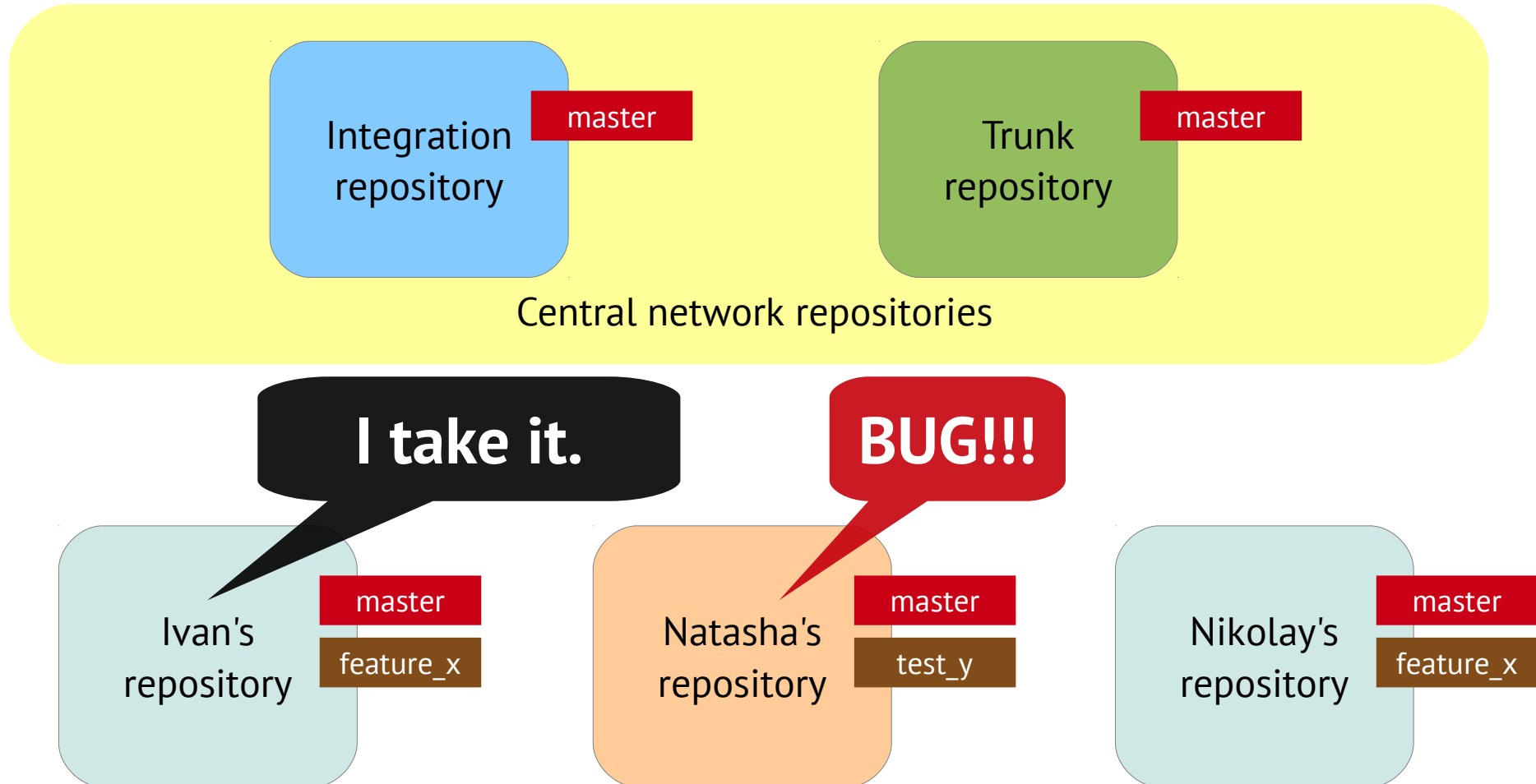
Fully distributed



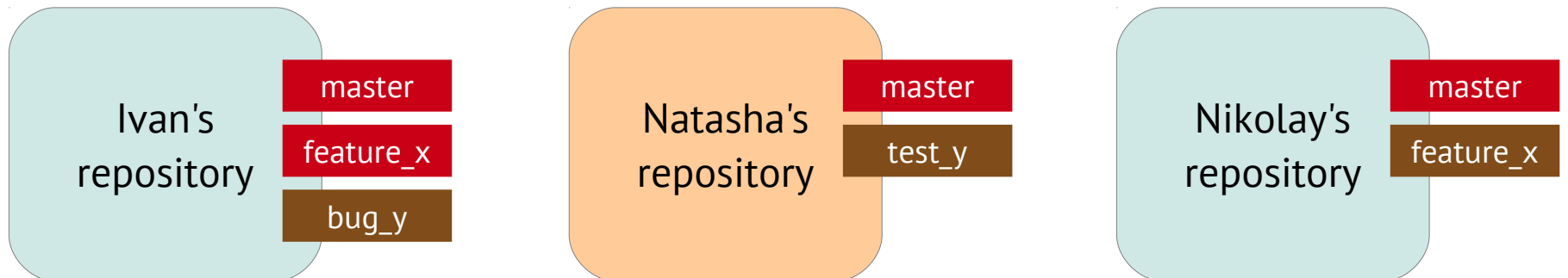
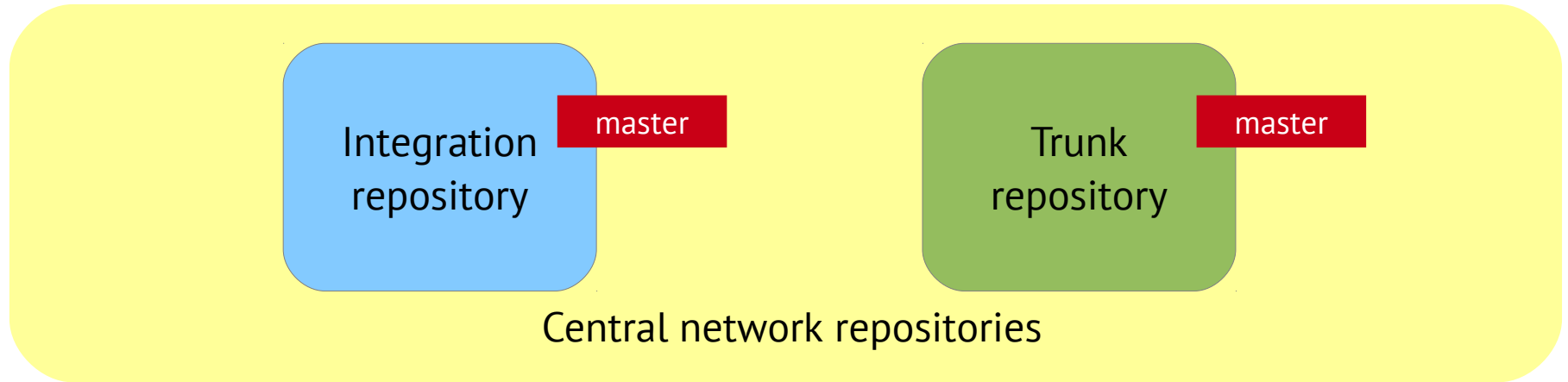
Fully distributed



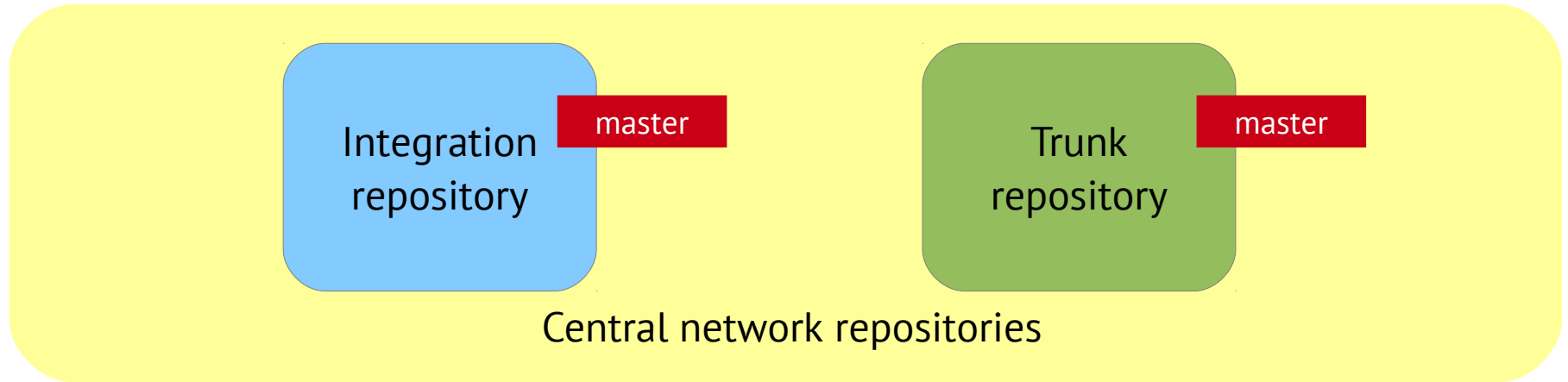
Fully distributed



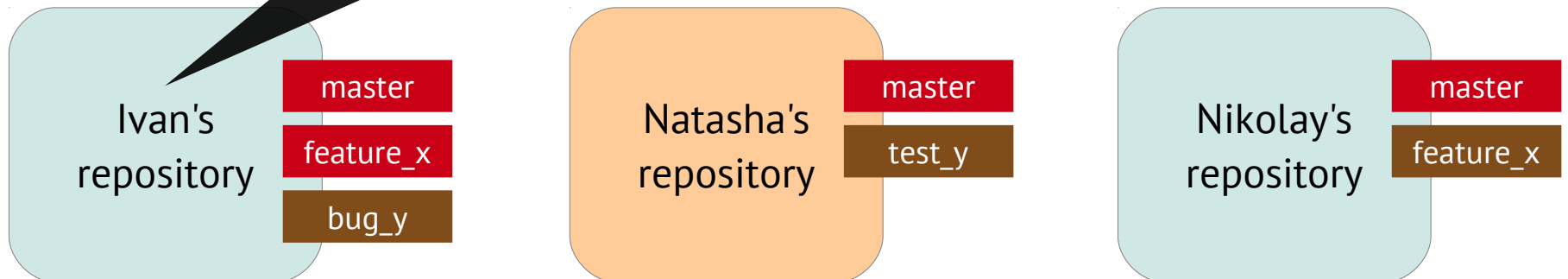
Fully distributed



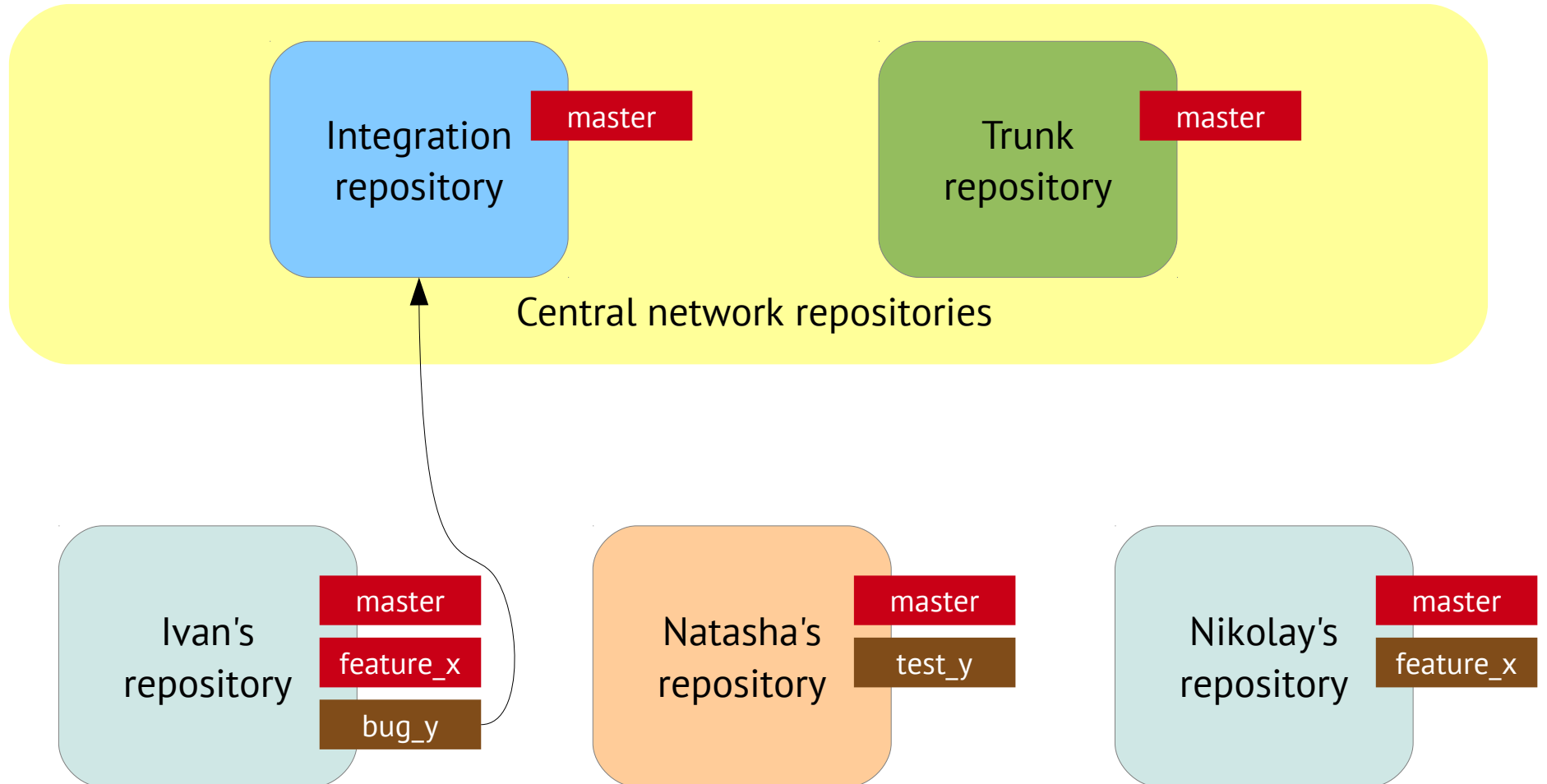
Fully distributed



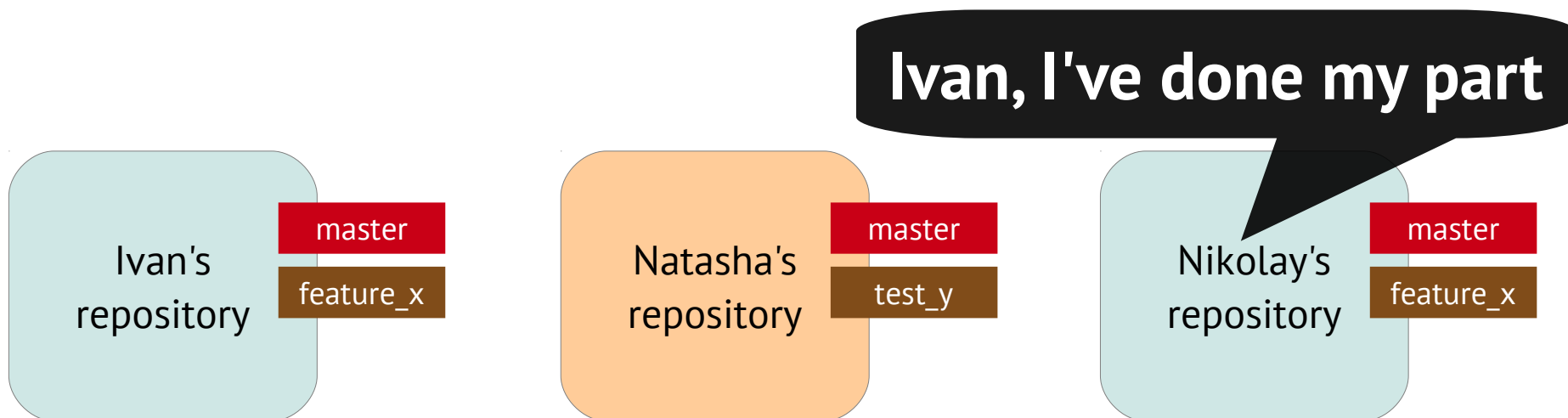
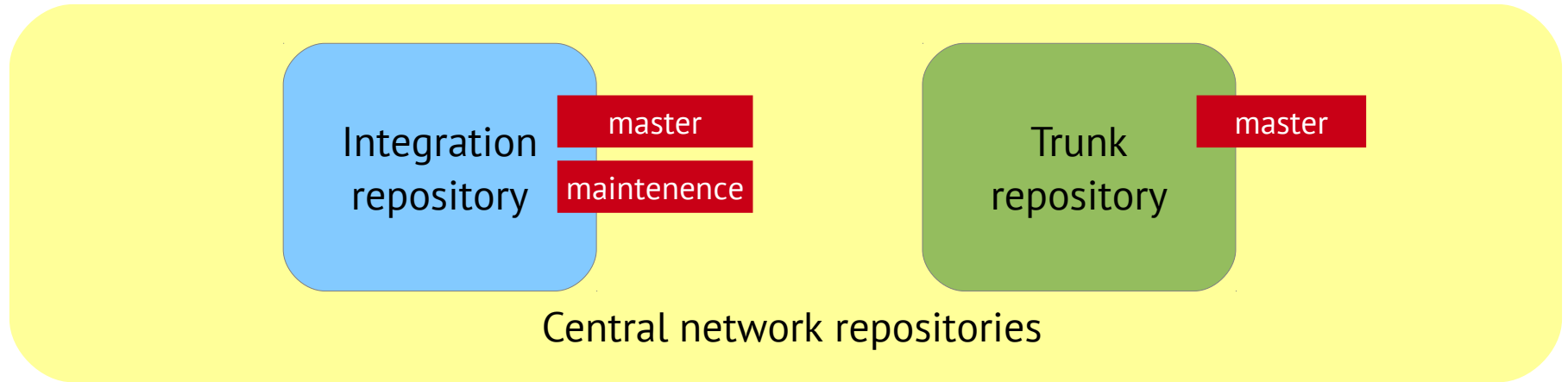
Ready!



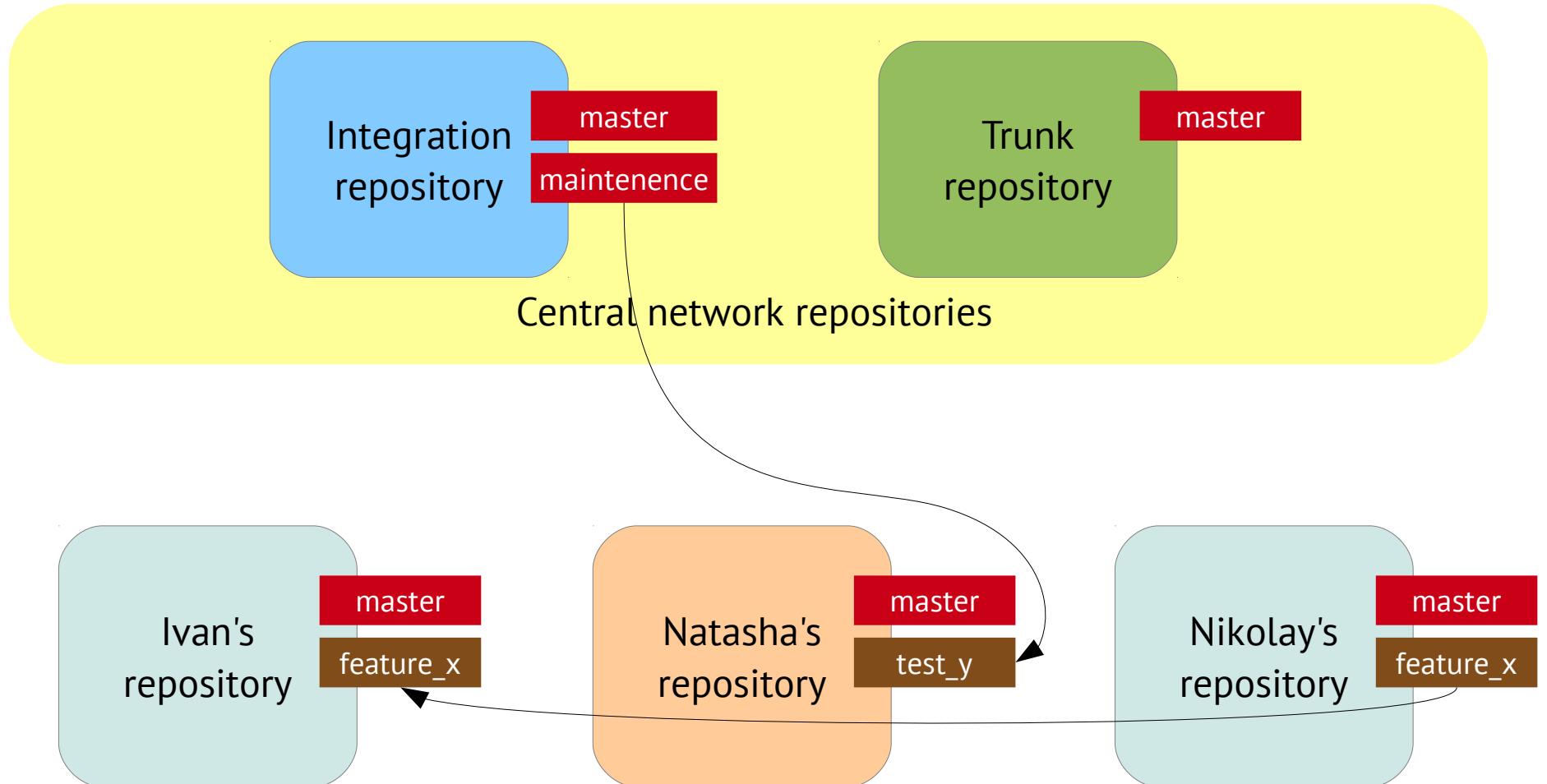
Fully distributed



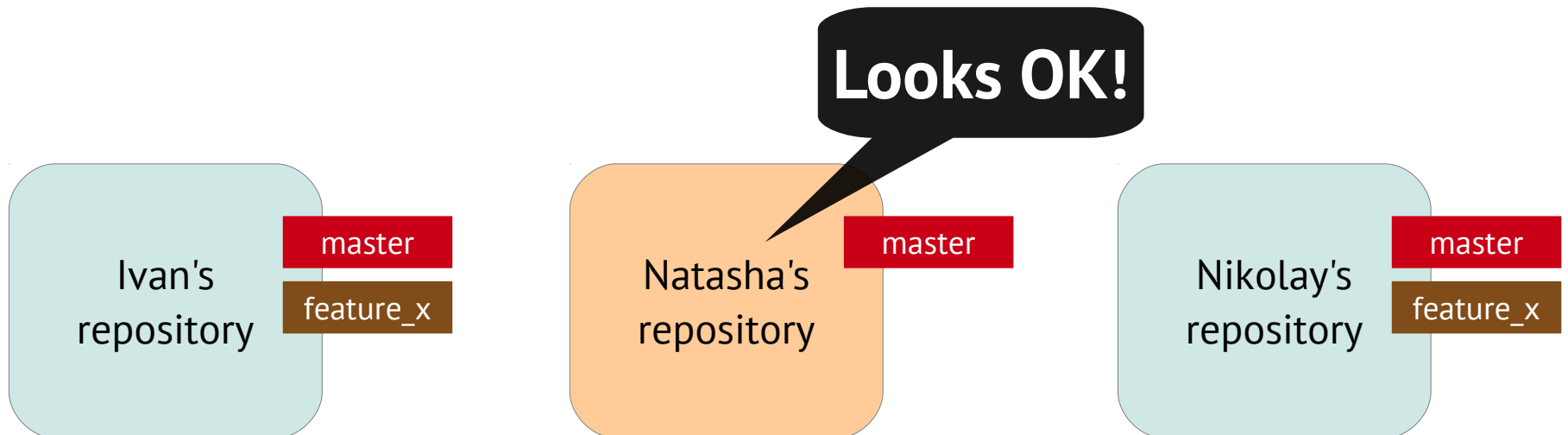
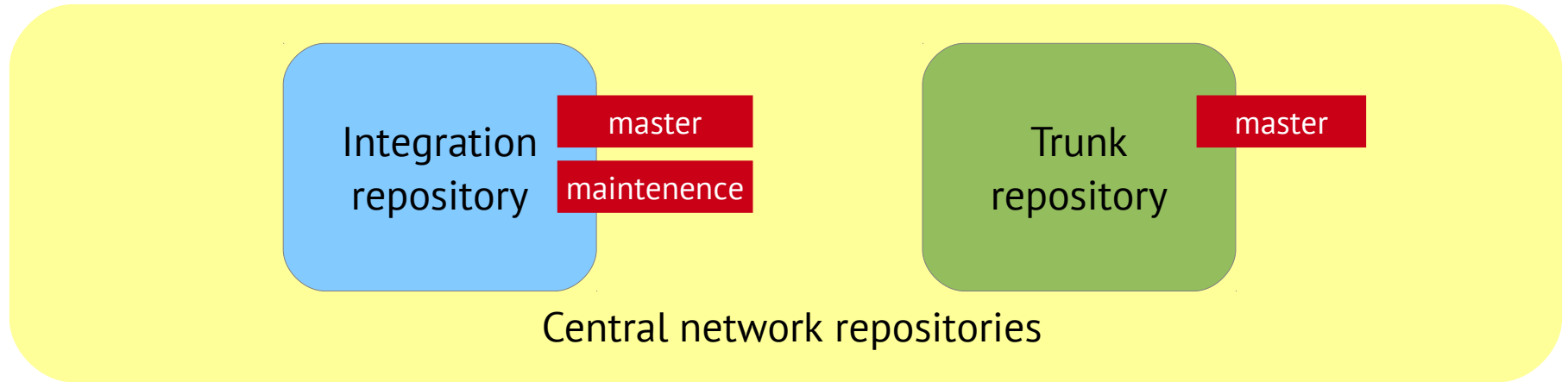
Fully distributed



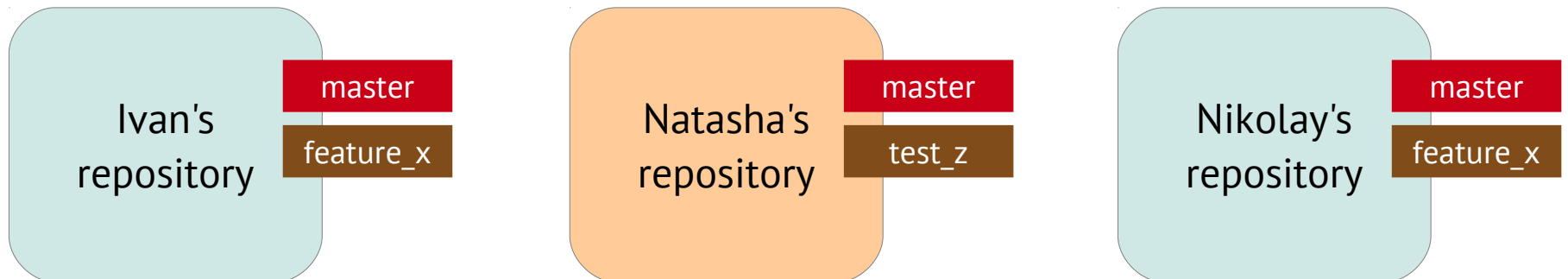
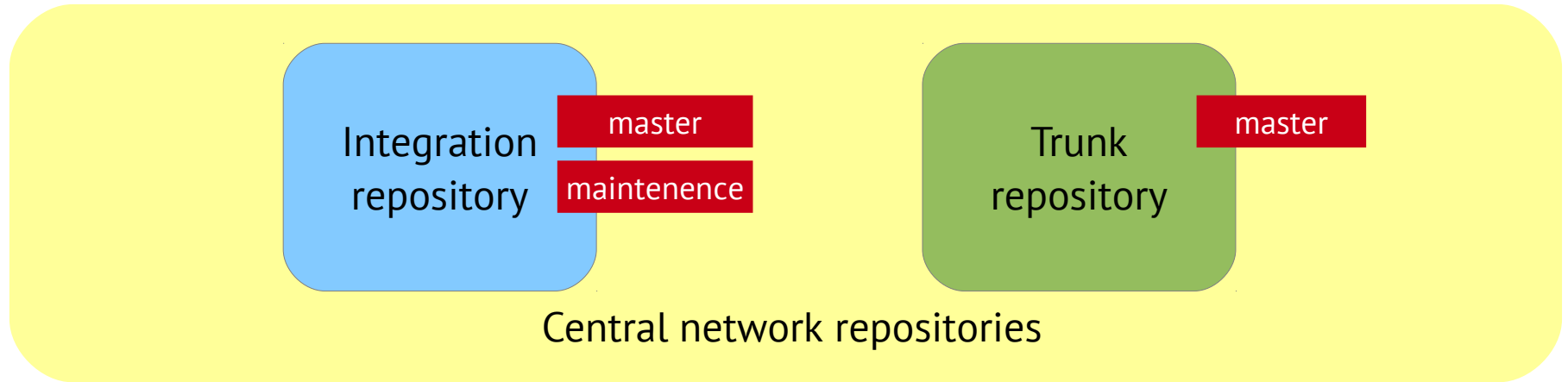
Fully distributed



Fully distributed

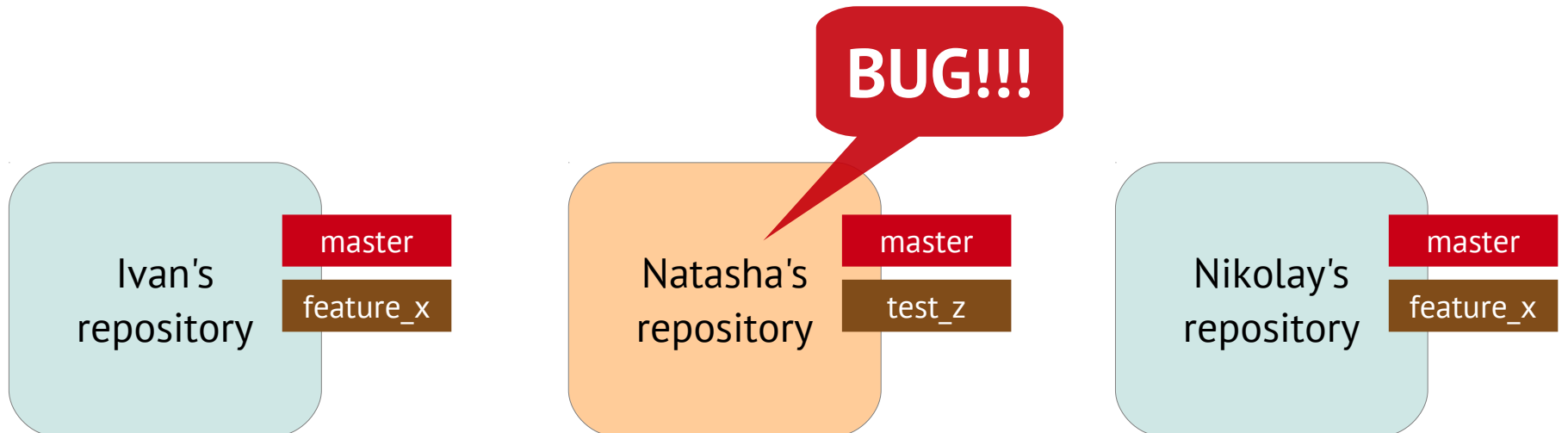


Fully distributed



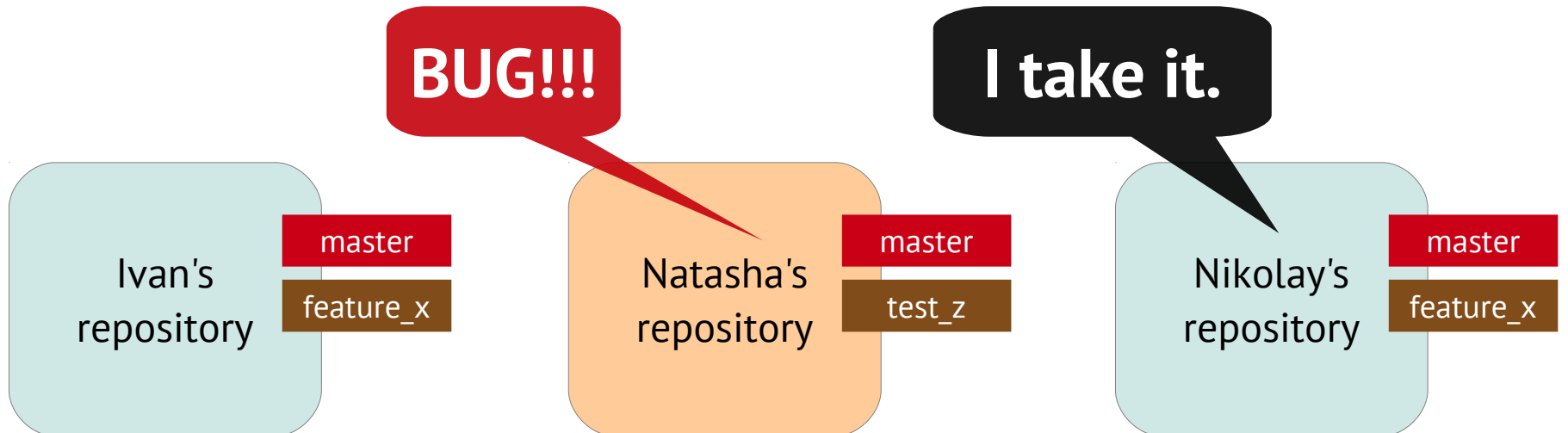
Fully distributed

Network down



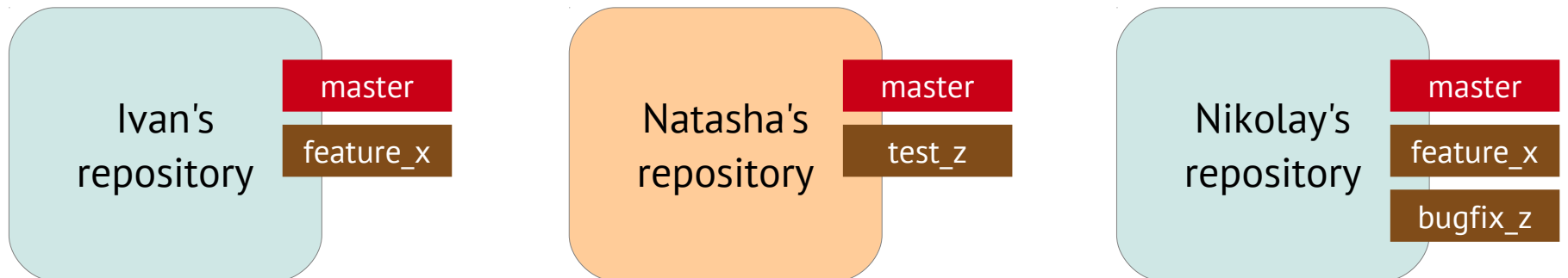
Fully distributed

Network down



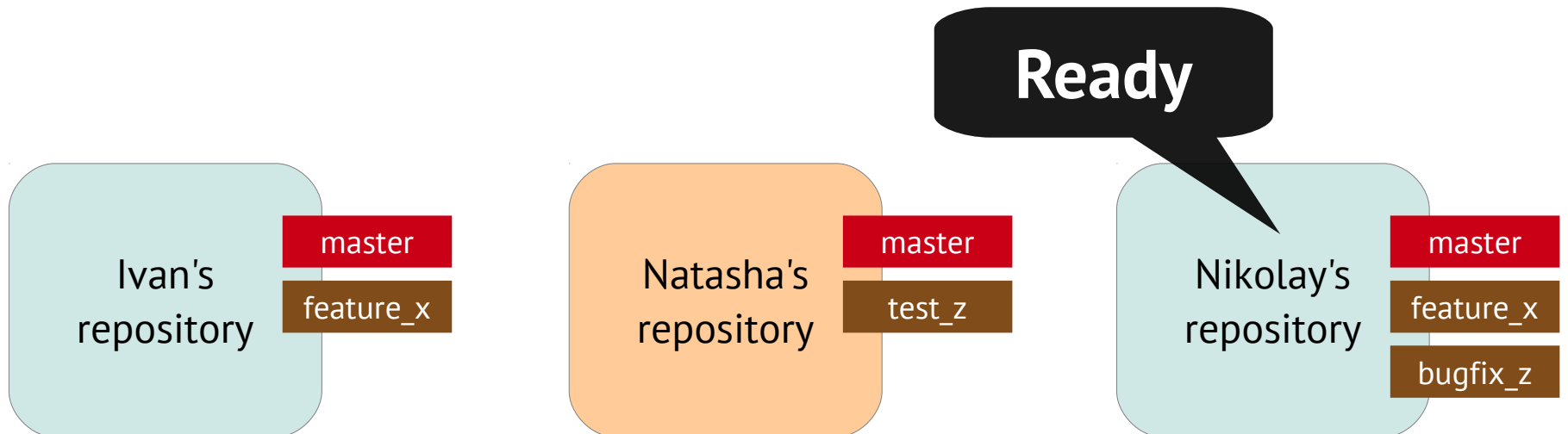
Fully distributed

Network down



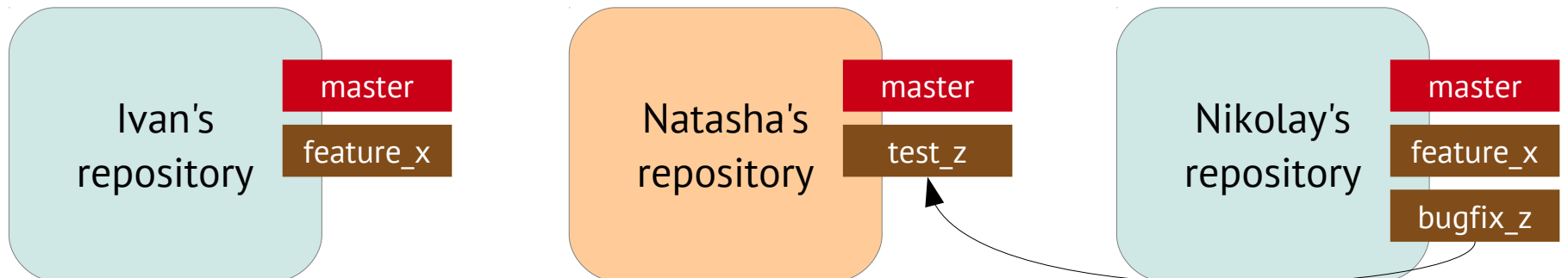
Fully distributed

Network down



Fully distributed

Network down



Fully distributed

Network down

Looks OK!

Ivan's
repository

master

feature_x

Natasha's
repository

master

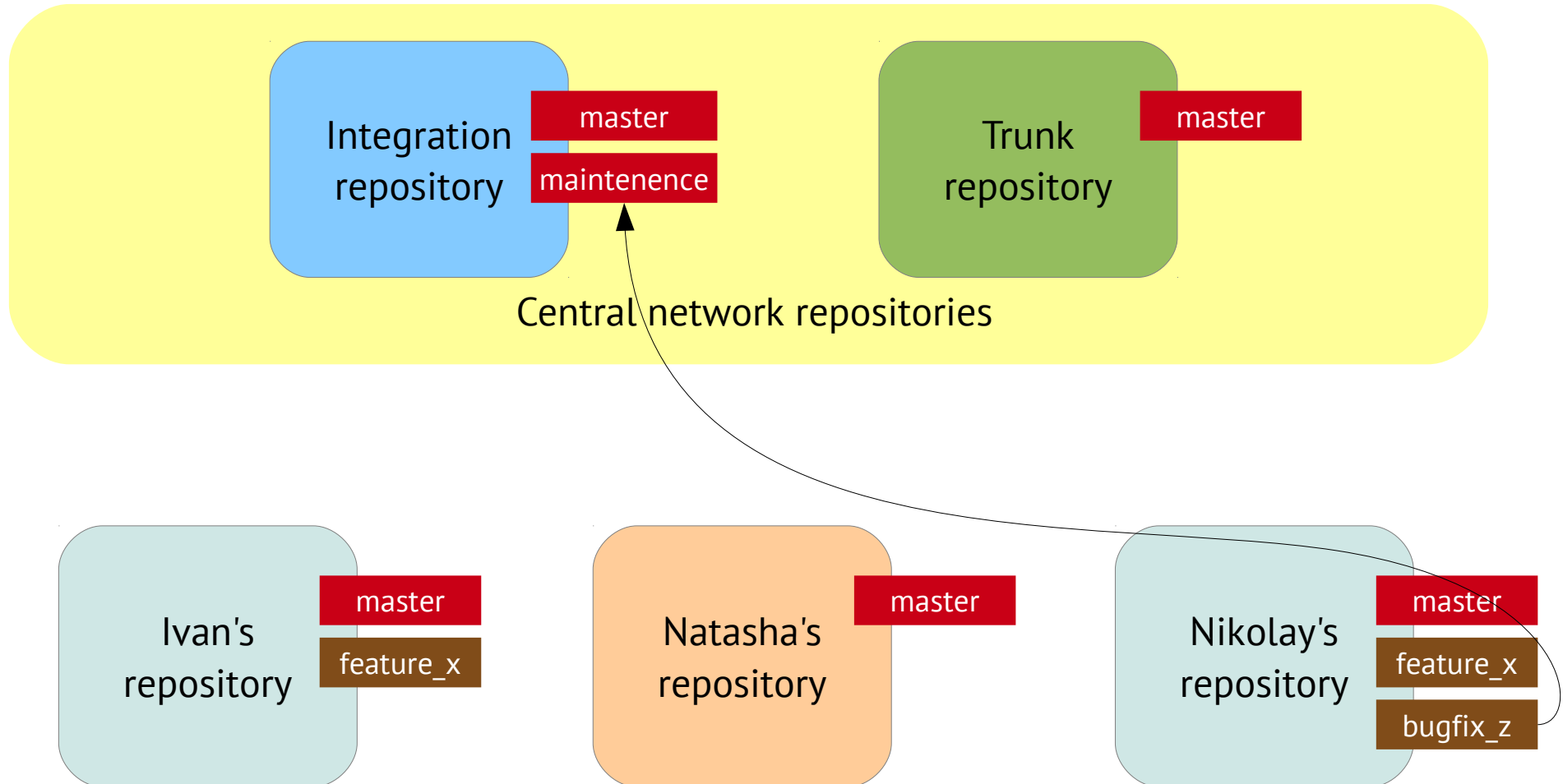
Nikolay's
repository

master

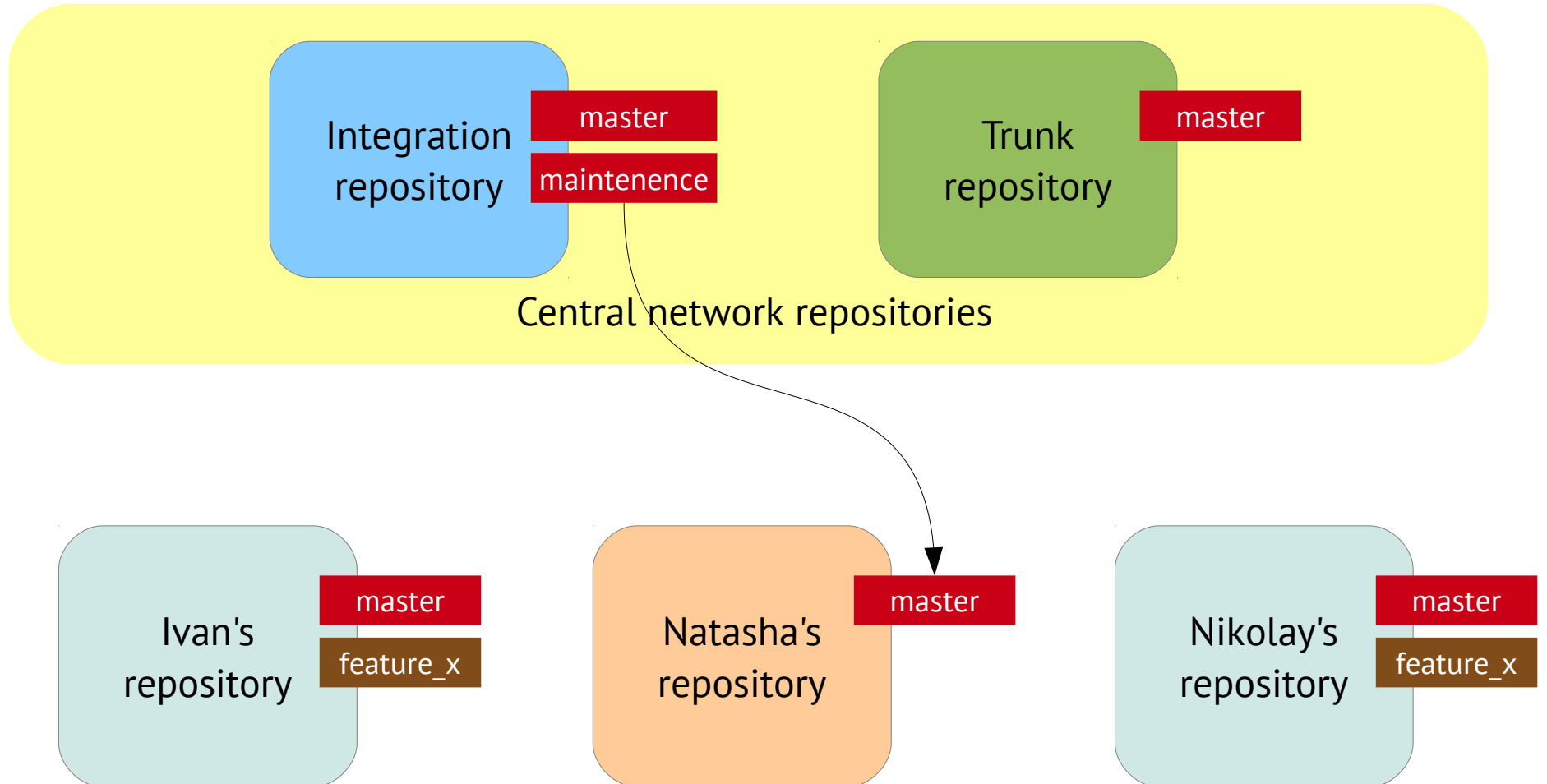
feature_x

bugfix_z

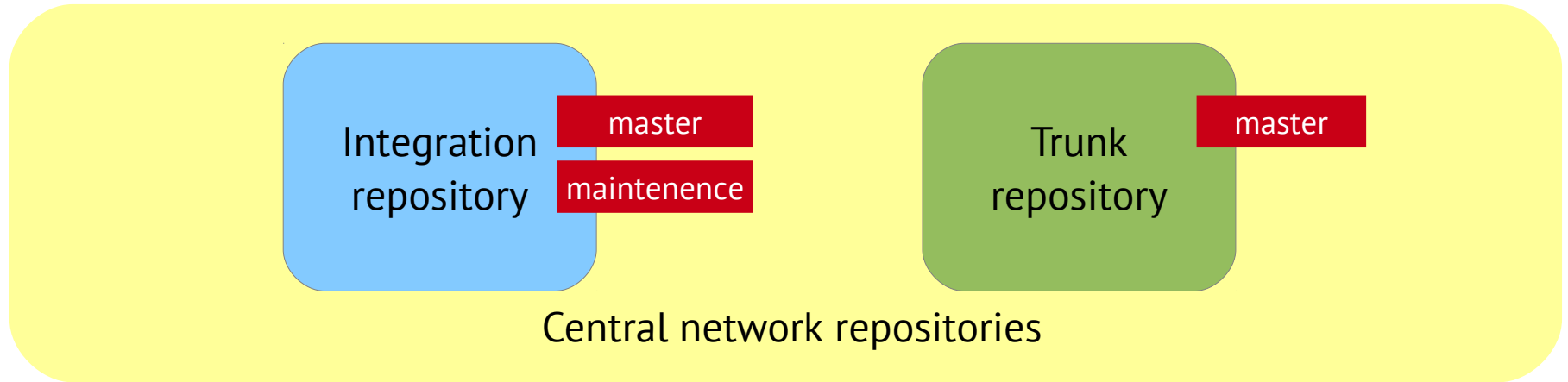
Fully distributed



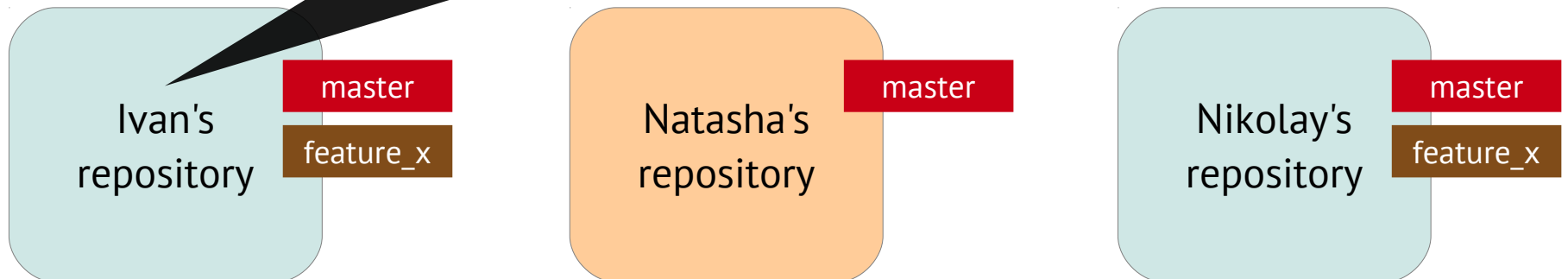
Fully distributed



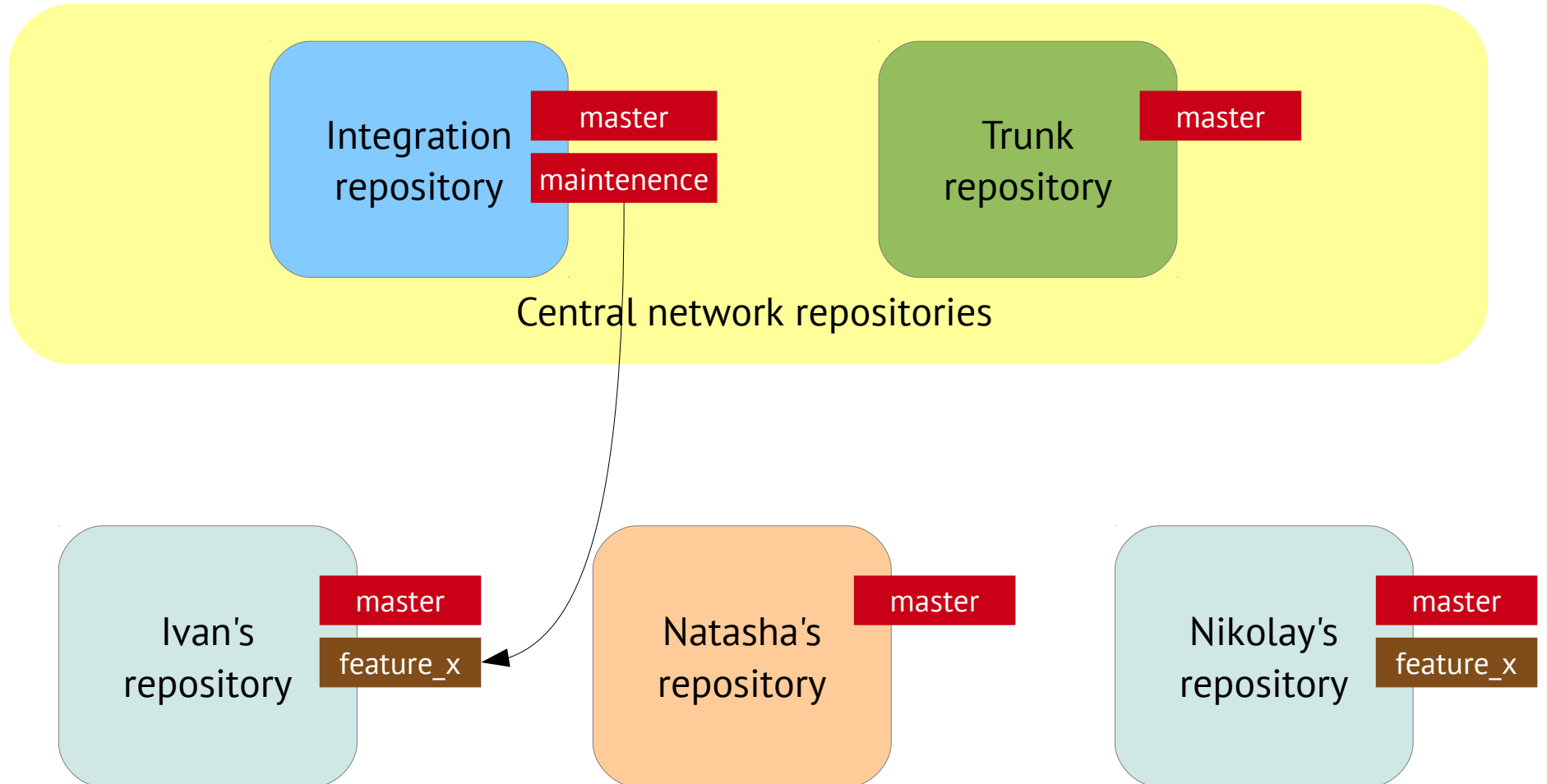
Fully distributed



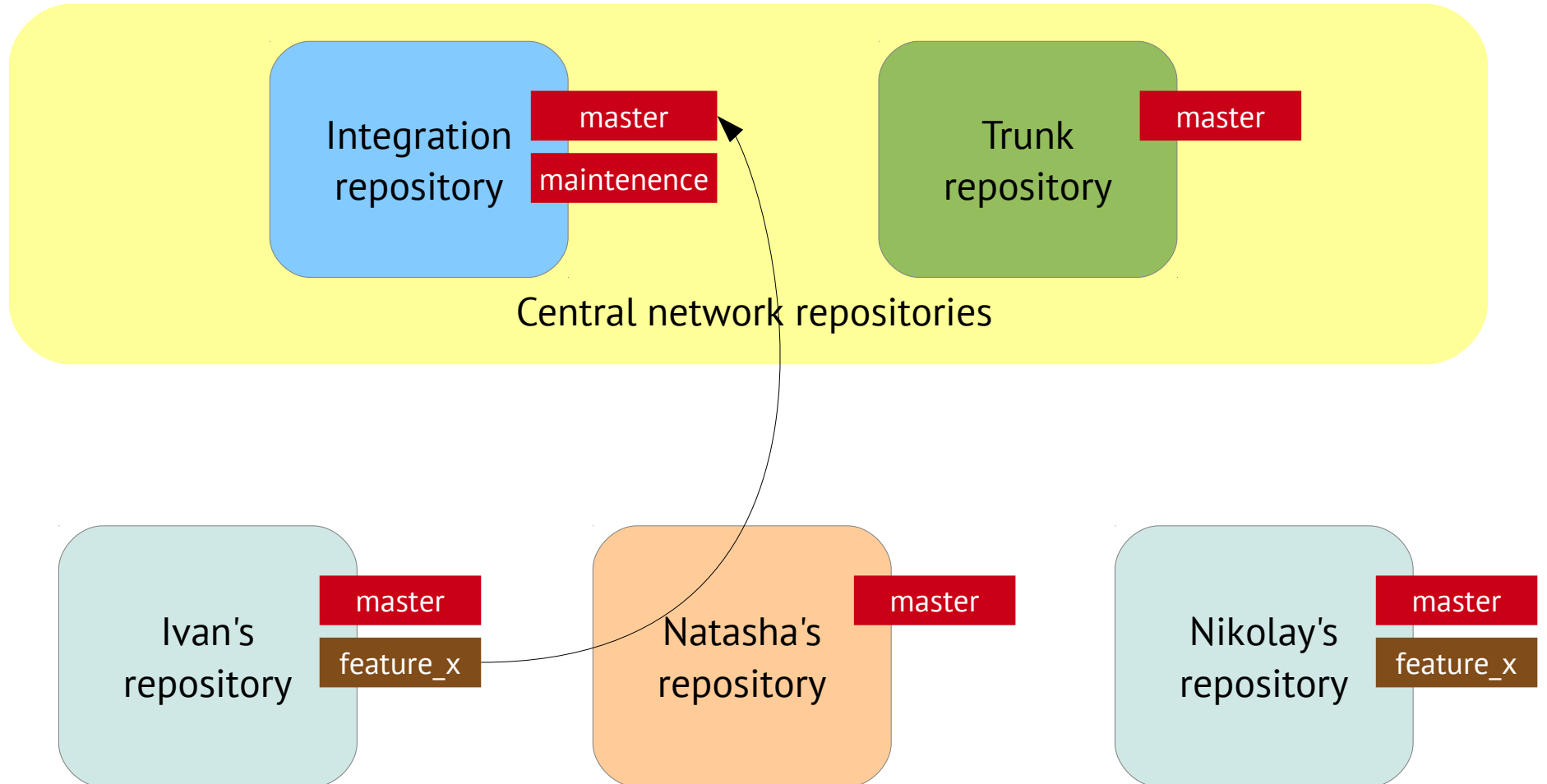
Ready. Need to rebase.



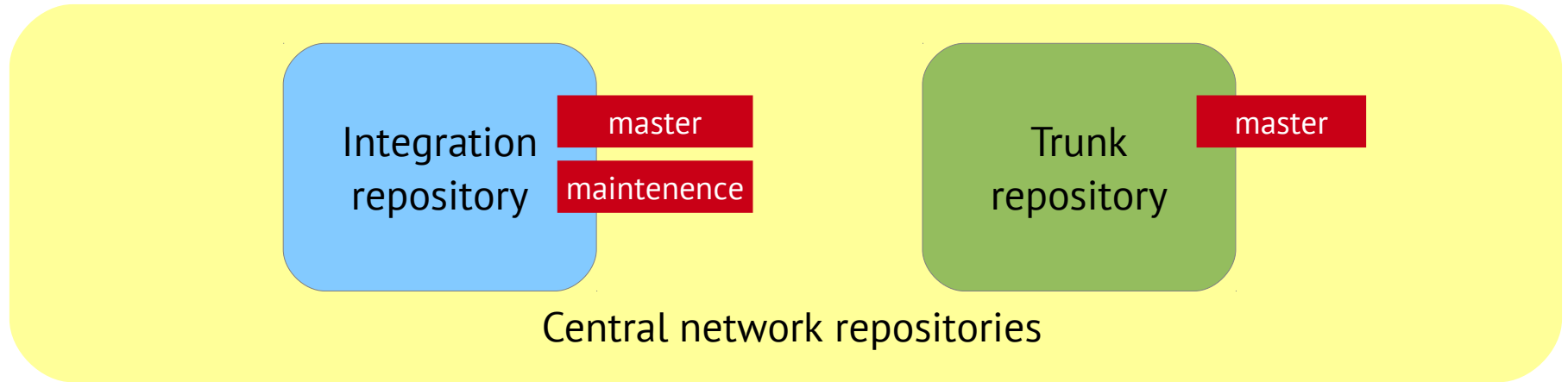
Fully distributed



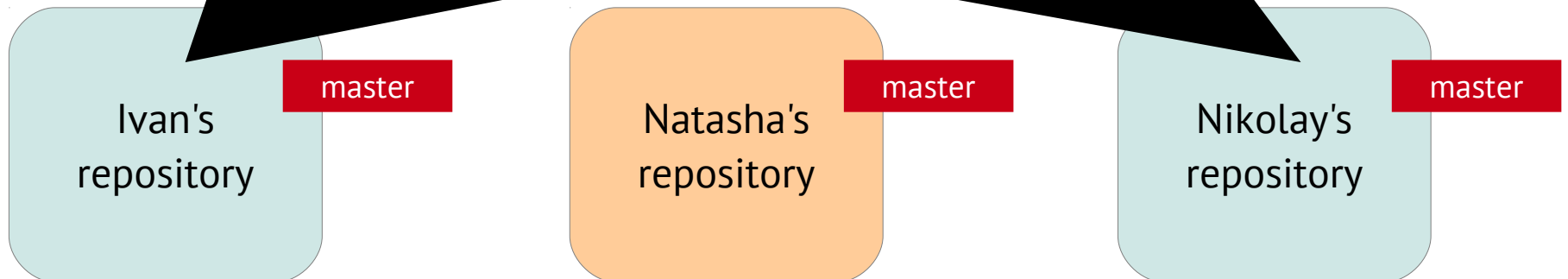
Fully distributed



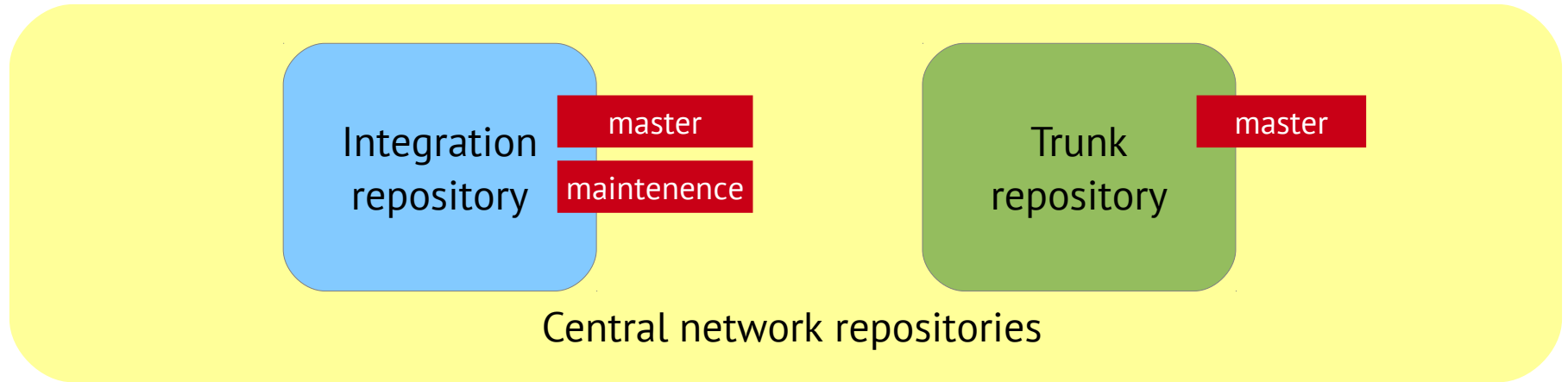
Fully distributed



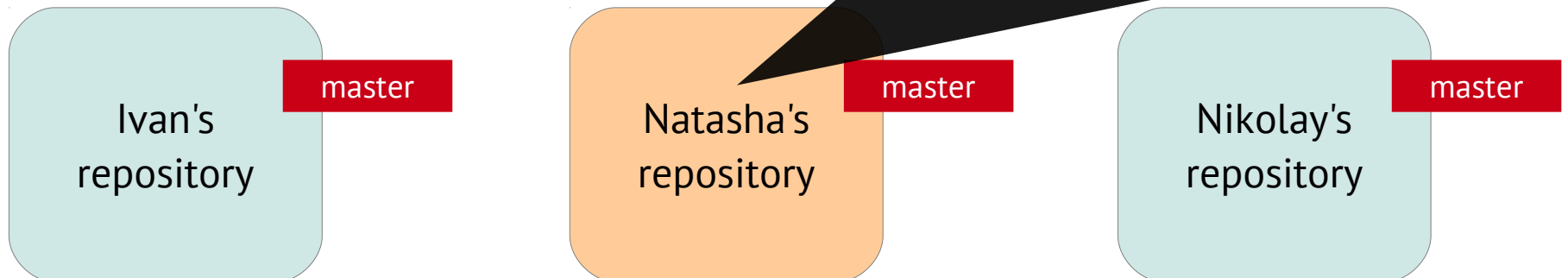
New feature is ready!



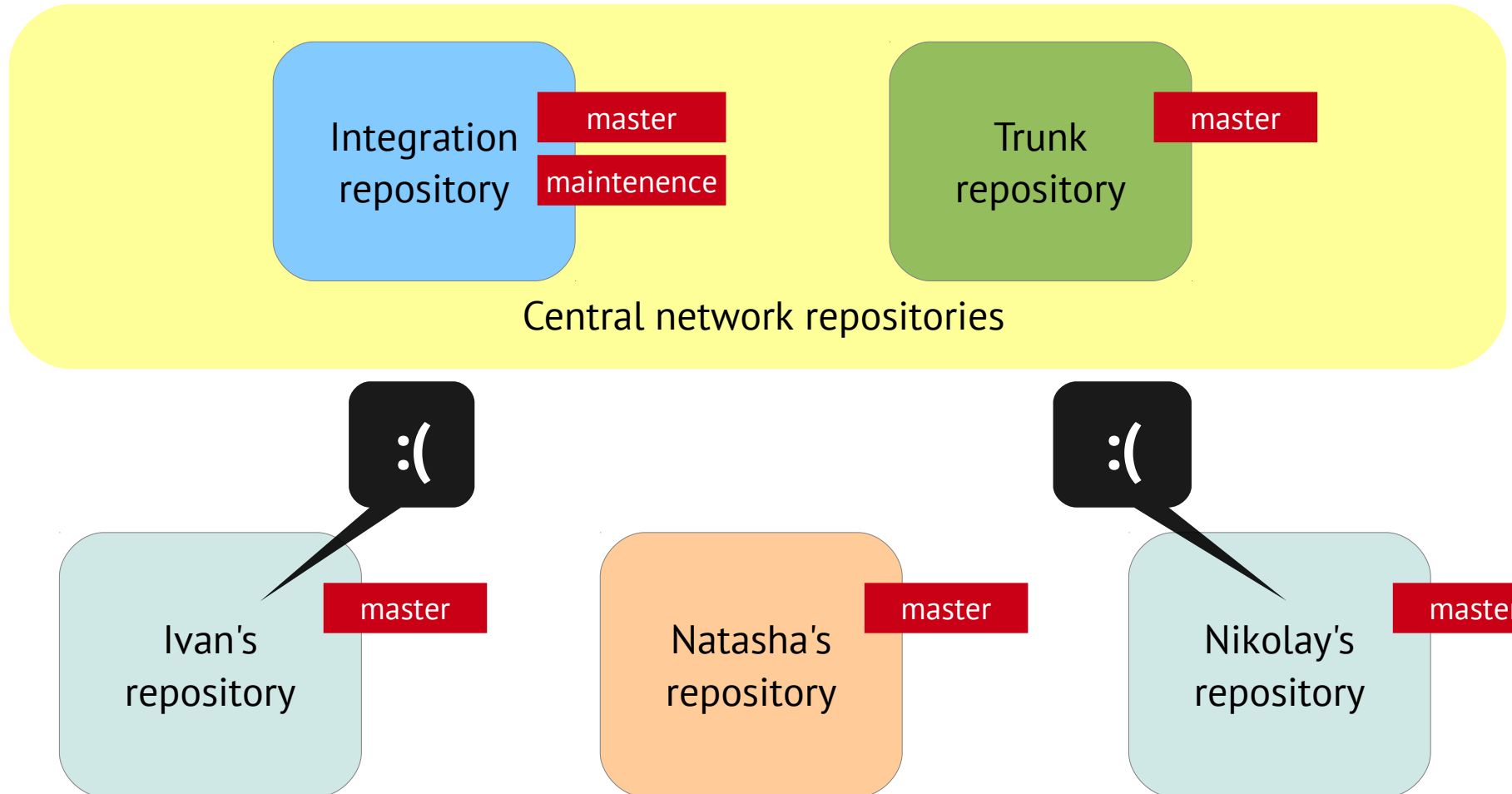
Fully distributed



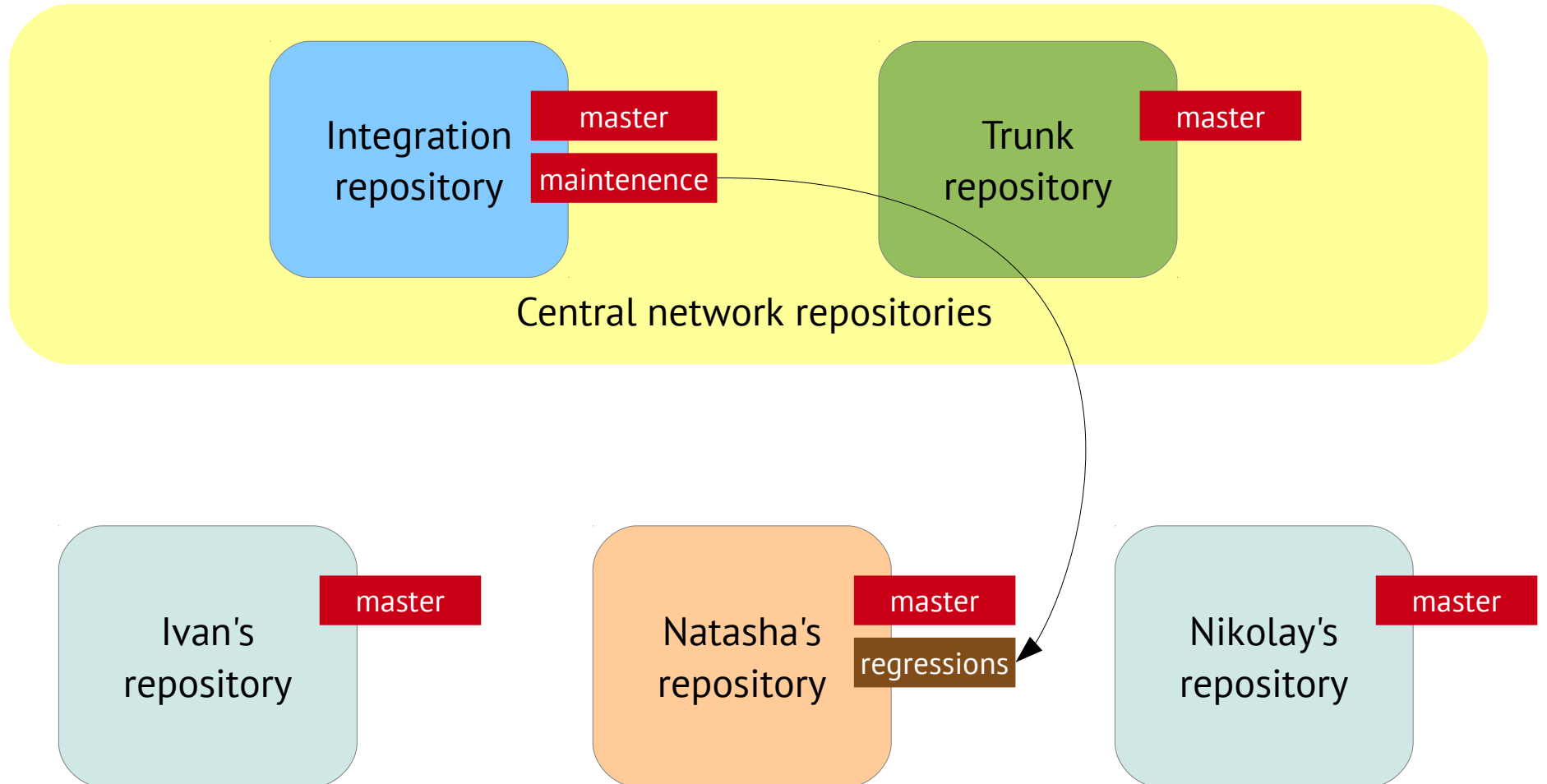
Have you said that new bugs are ready?



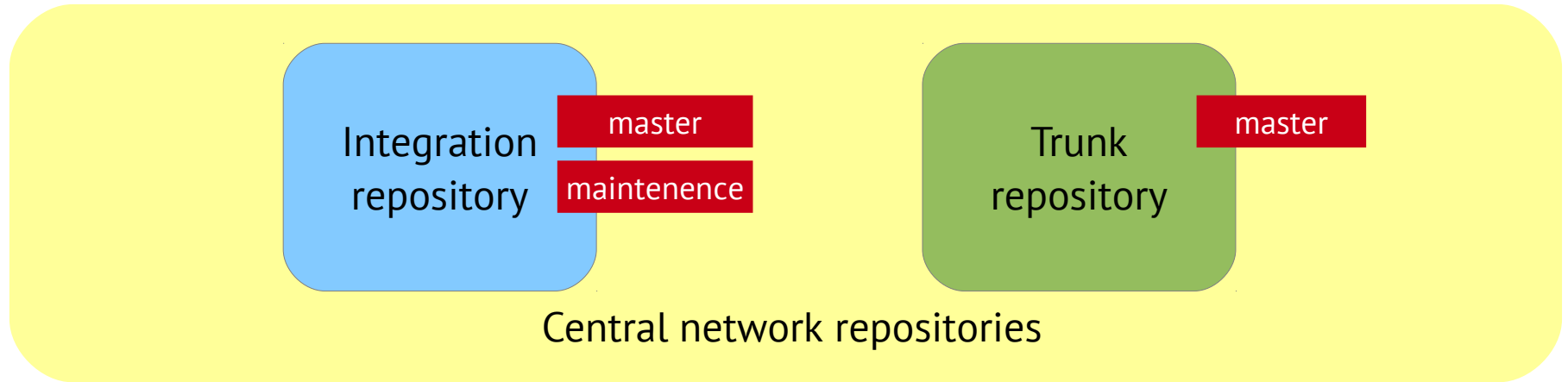
Fully distributed



Fully distributed



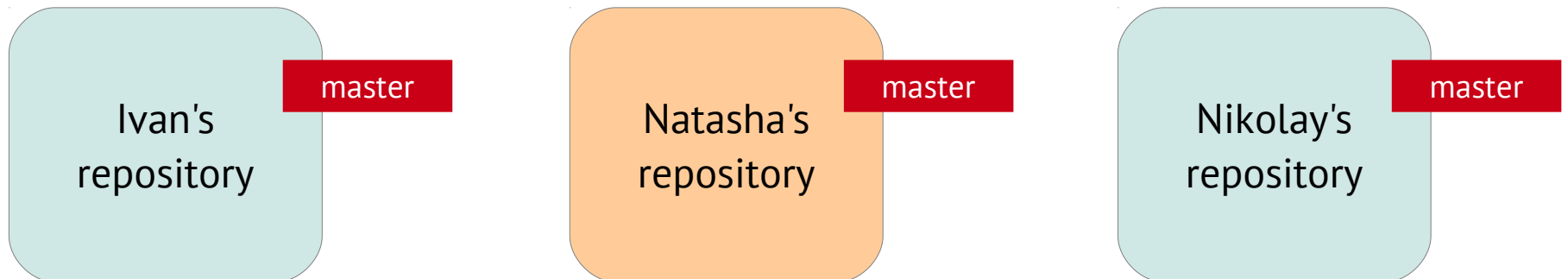
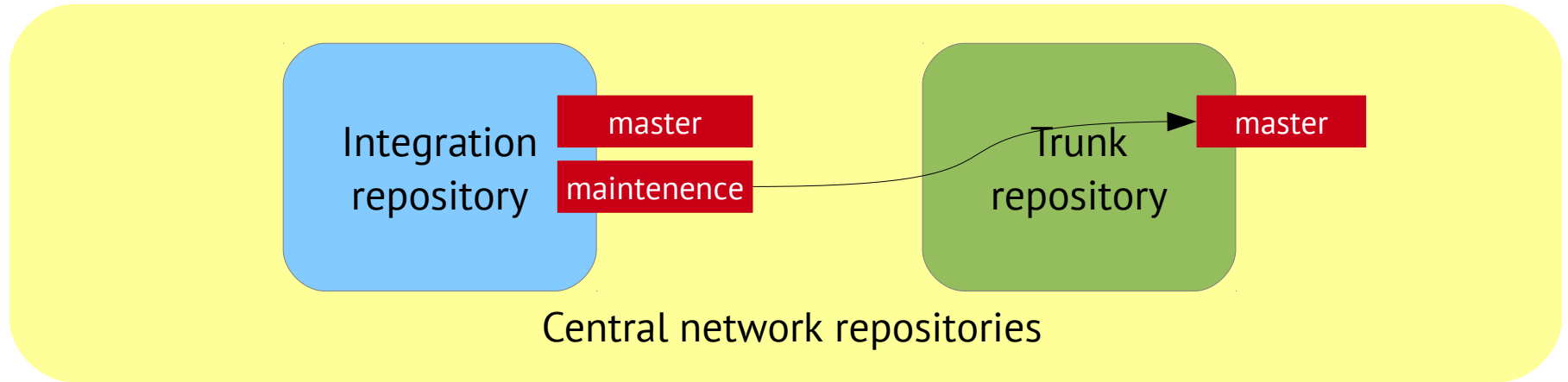
Fully distributed



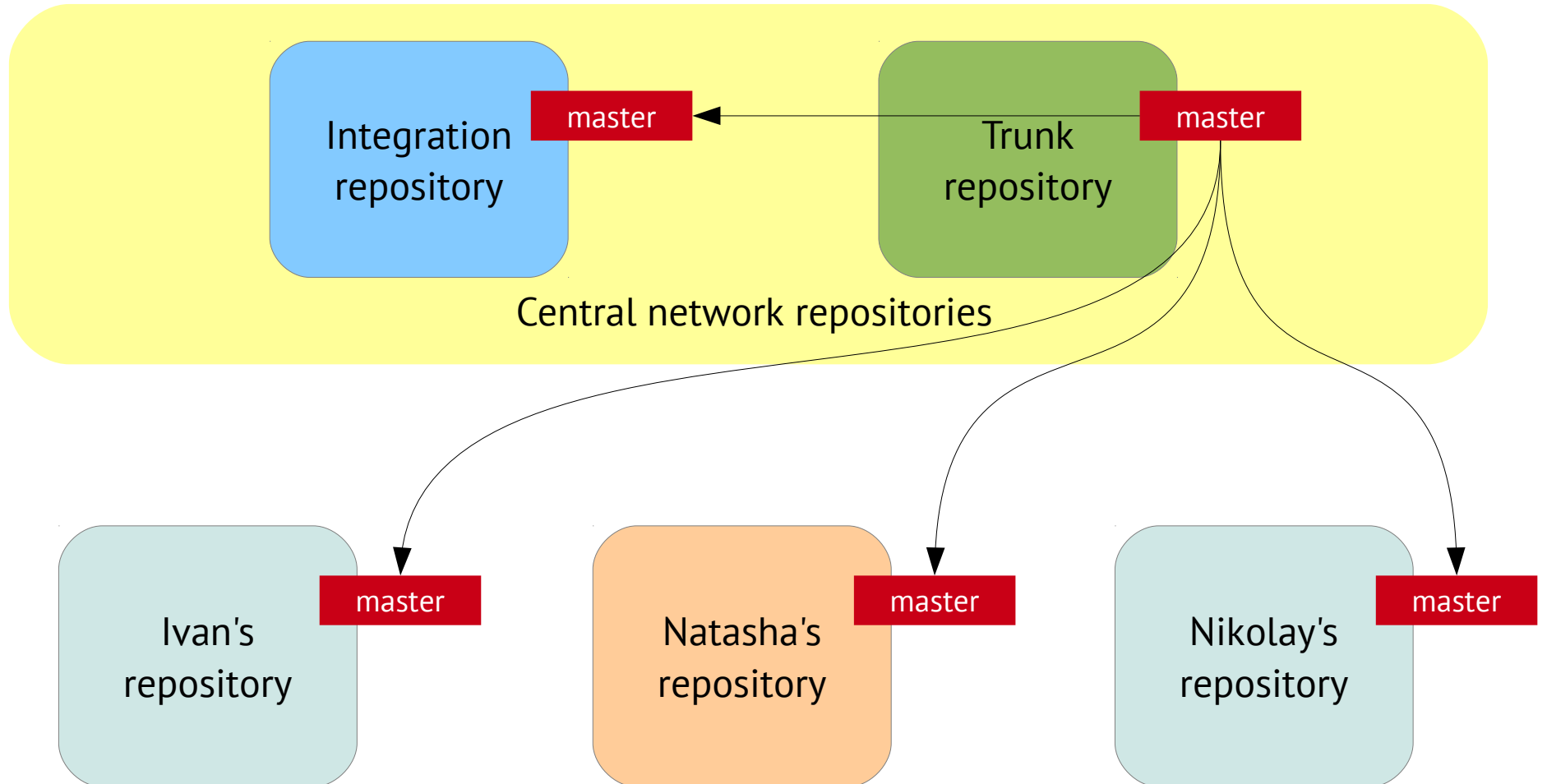
Old feature looks done



Fully distributed



Fully distributed



GIT workflow

GIT workflow

- **Create your branch**
- Edit files
- Stage your changes
- Commit the changes
- Rebase your branch from master
- Merge to master
- Push to central repository

Branches

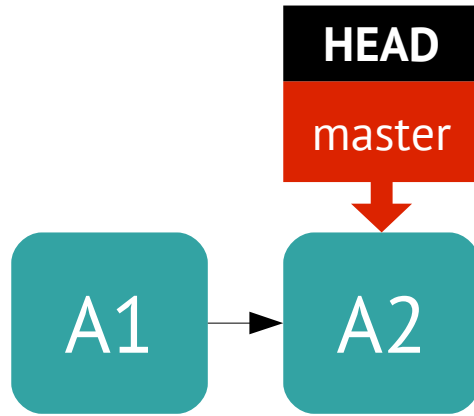
GIT workflow

Branching in details



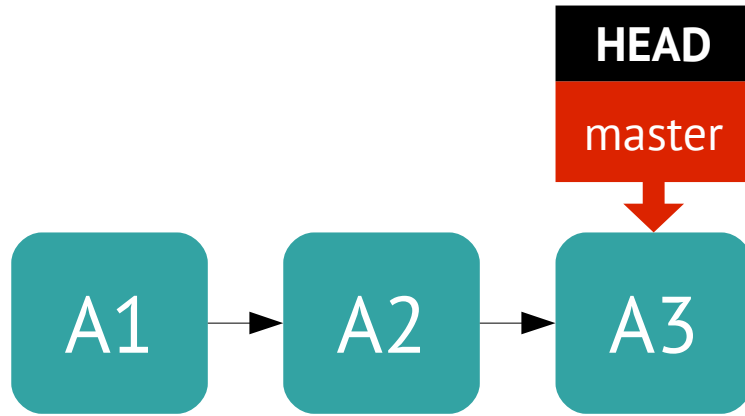
GIT workflow

Branching in details



GIT workflow

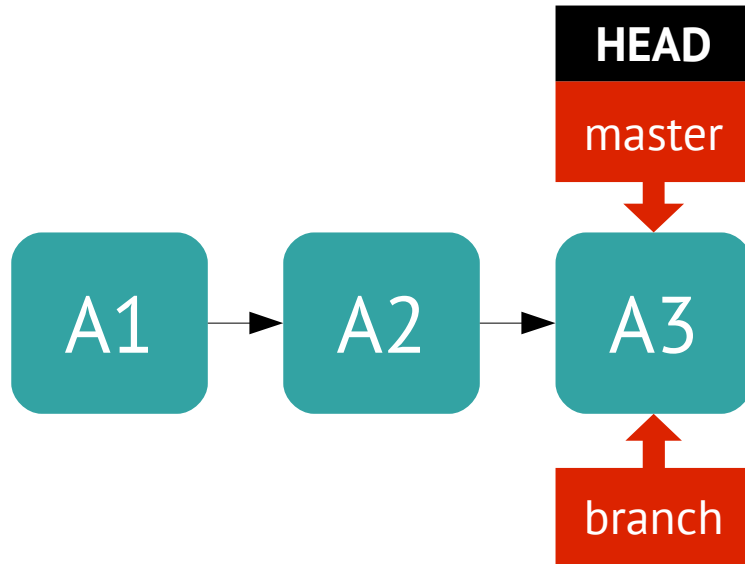
Branching in details



git branch branch

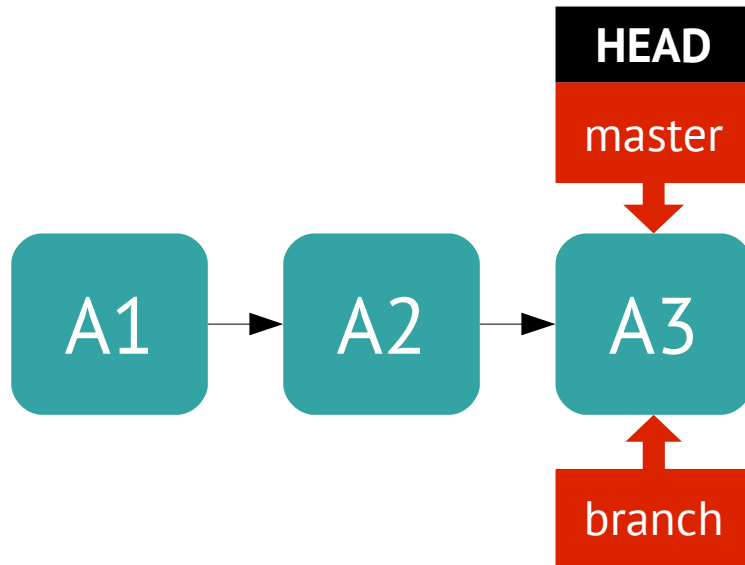
GIT workflow

Branching in details



GIT workflow

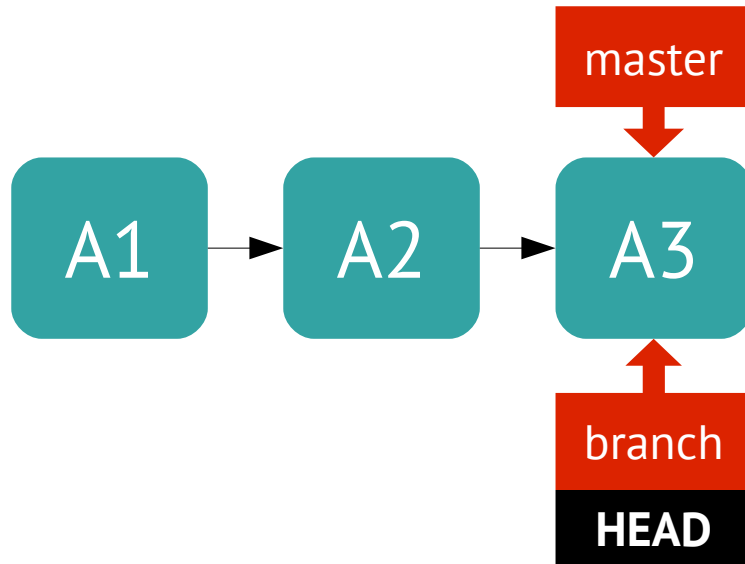
Branching in details



git checkout branch

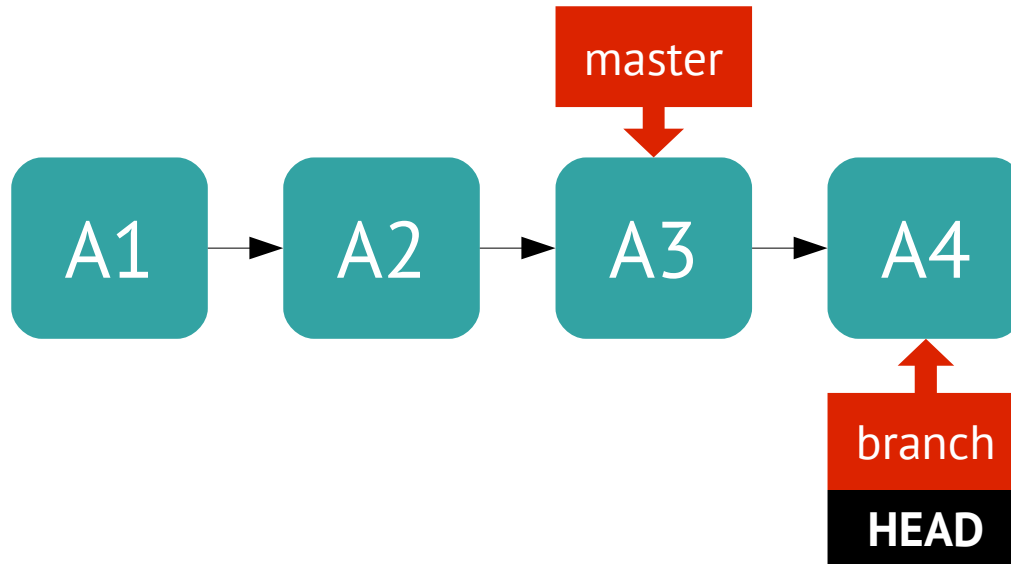
GIT workflow

Branching in details



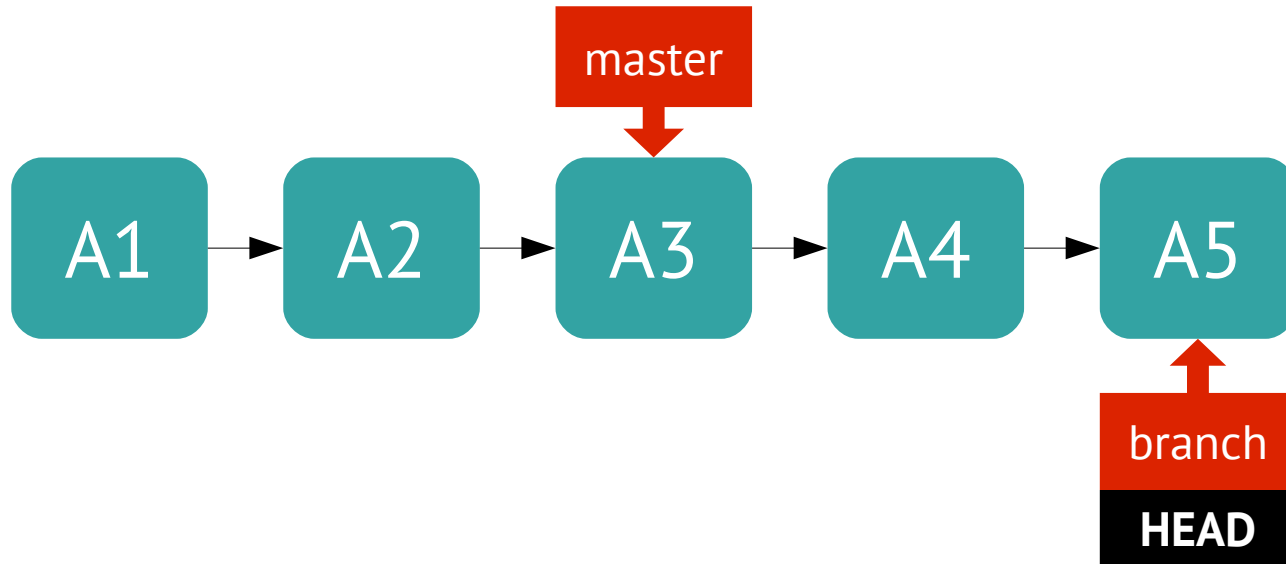
GIT workflow

Branching in details



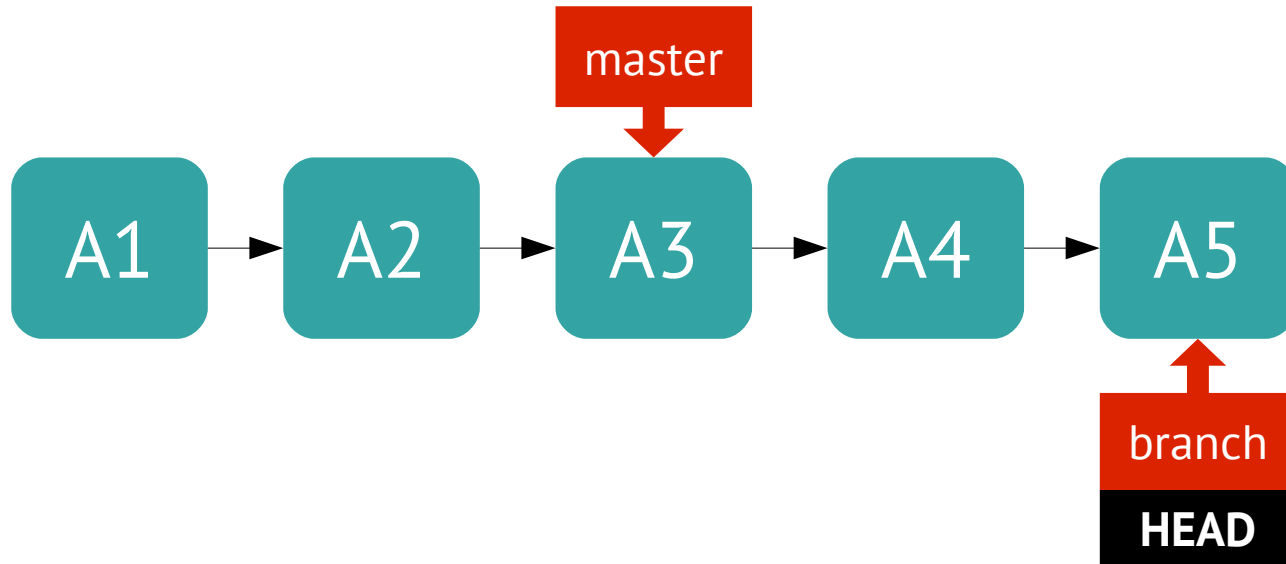
GIT workflow

Branching in details



GIT workflow

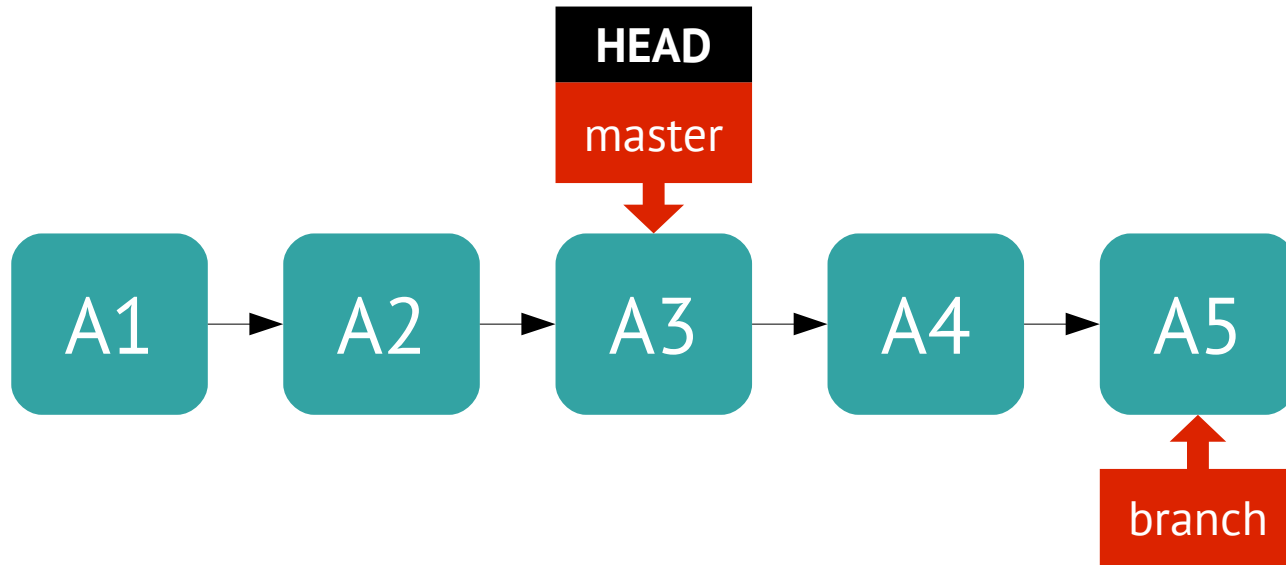
Branching in details



git checkout master

GIT workflow

Branching in details



What's up, folks?

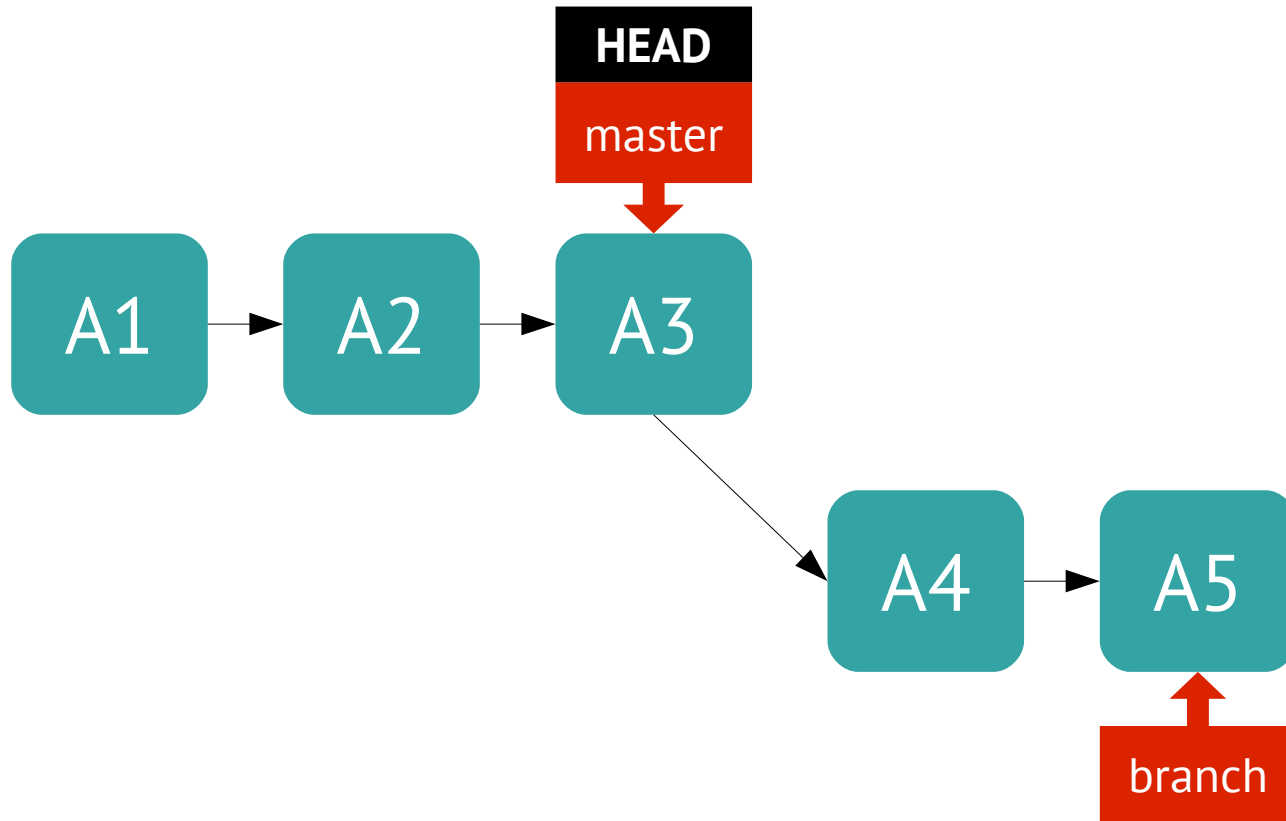
This presentation
contains 255 slides
and looks like that
you have reached
the midway.

Congrats!



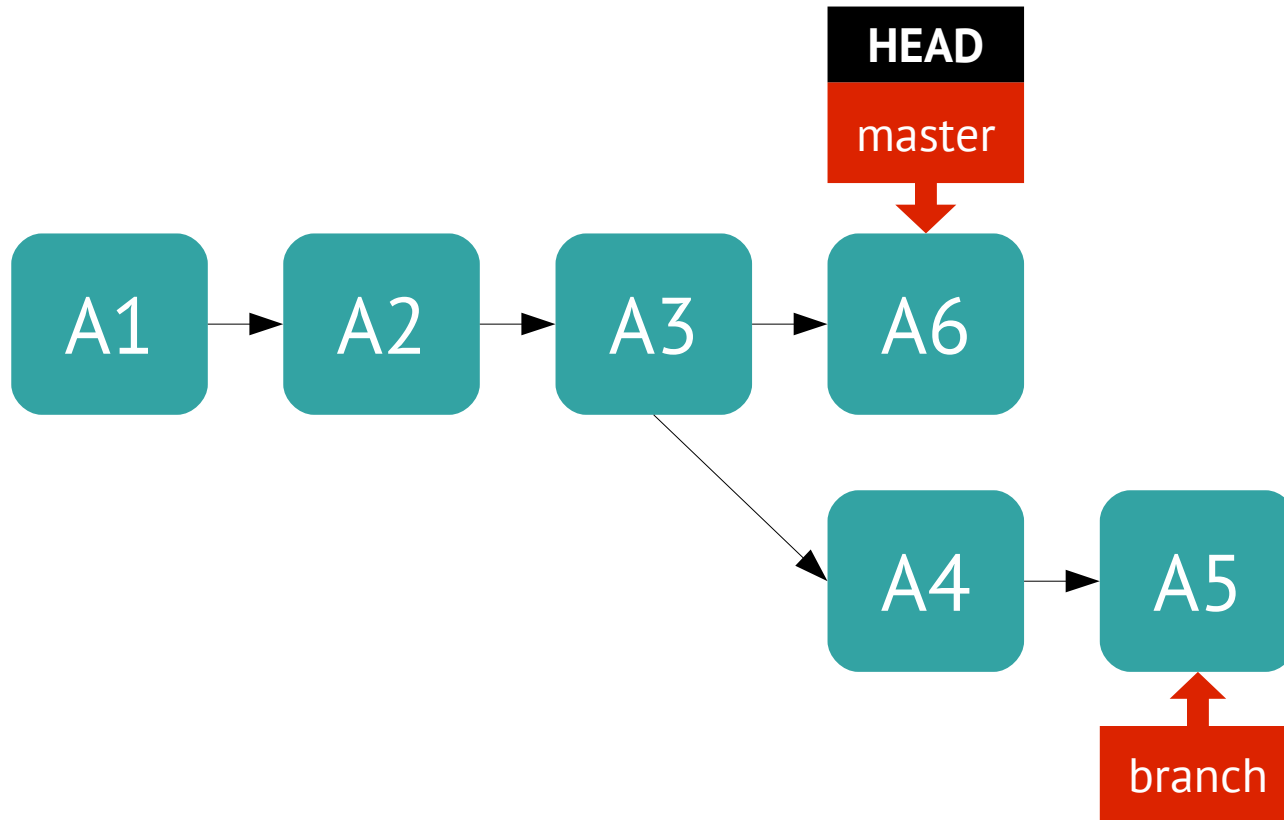
GIT workflow

Branching in details



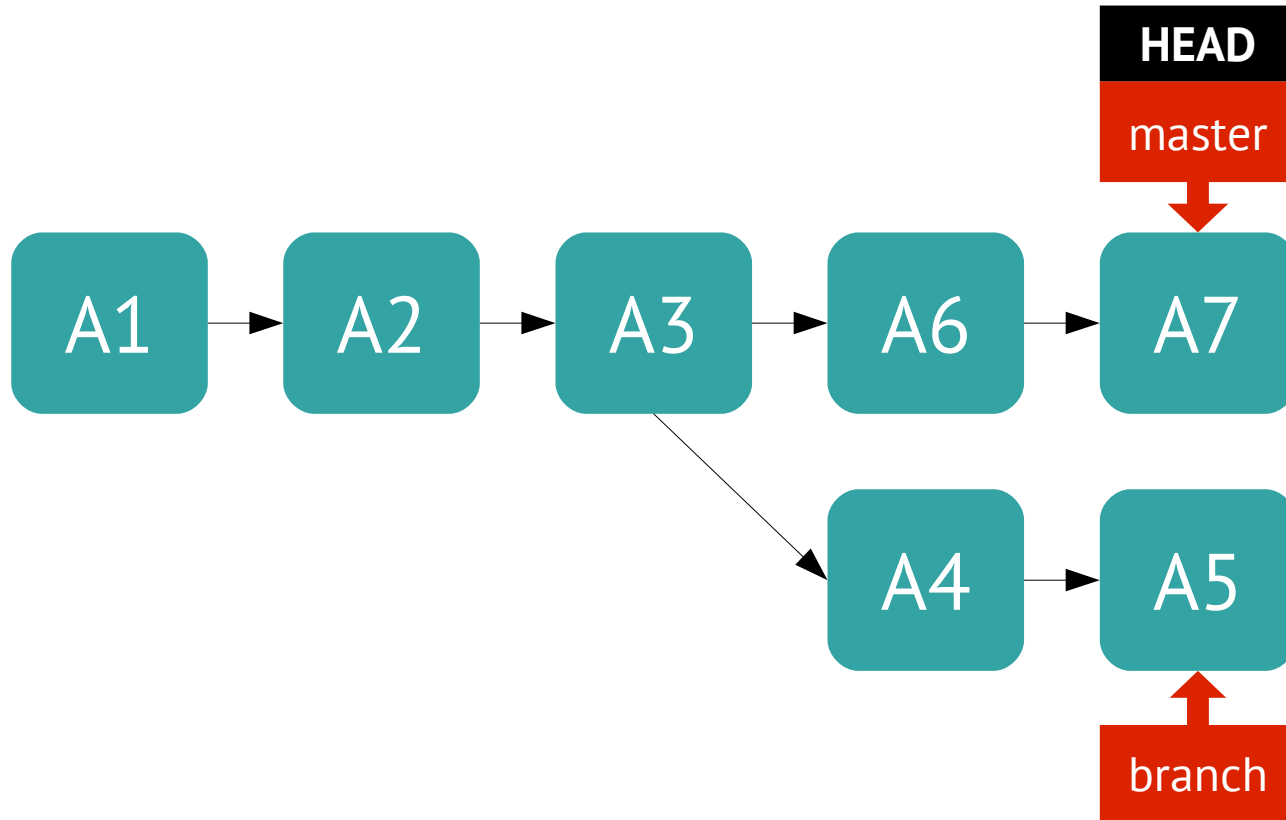
GIT workflow

Branching in details



GIT workflow

Branching in details



GIT workflow

Branching in details

- Branch pointer always points on the head of a branch;
- Creating branch = creating pointer to commit. 41 bytes on your disk;
- Removing branch is removing pointer.
- Obsolete commits is removed by GIT's GC;
- HEAD meta-tag points to a current active branch;
- One task = one branch.

GIT workflow

Branching in details

- Branch pointer always points on the head of a branch;
- Creating branch = creating pointer to commit. 41 bytes on your disk;
- Removing branch is removing pointer.
- Obsolete commits is removed by GIT's GC;
- HEAD meta-tag points to a current active branch;
- **One task = one branch.**

GIT workflow

- ~~Create your branch~~
- **Edit files**
- **Stage your changes**
- **Commit the changes**
- Rebase your branch from master
- Merge to master
- Push to central repository

Staging and committing

GIT workflow

Staging and committing



The diagram illustrates the Git workflow for staging and committing. It consists of a large yellow rounded rectangle containing three stacked, rounded rectangular boxes. The top box is red and labeled 'Working directory'. The middle box is purple and labeled 'Index'. The bottom box is green and labeled 'Repository'. Arrows indicate the flow from the Working directory to the Index, and then from the Index to the Repository.

Working directory

Index

Repository

GIT workflow

Staging and committing

Working directory

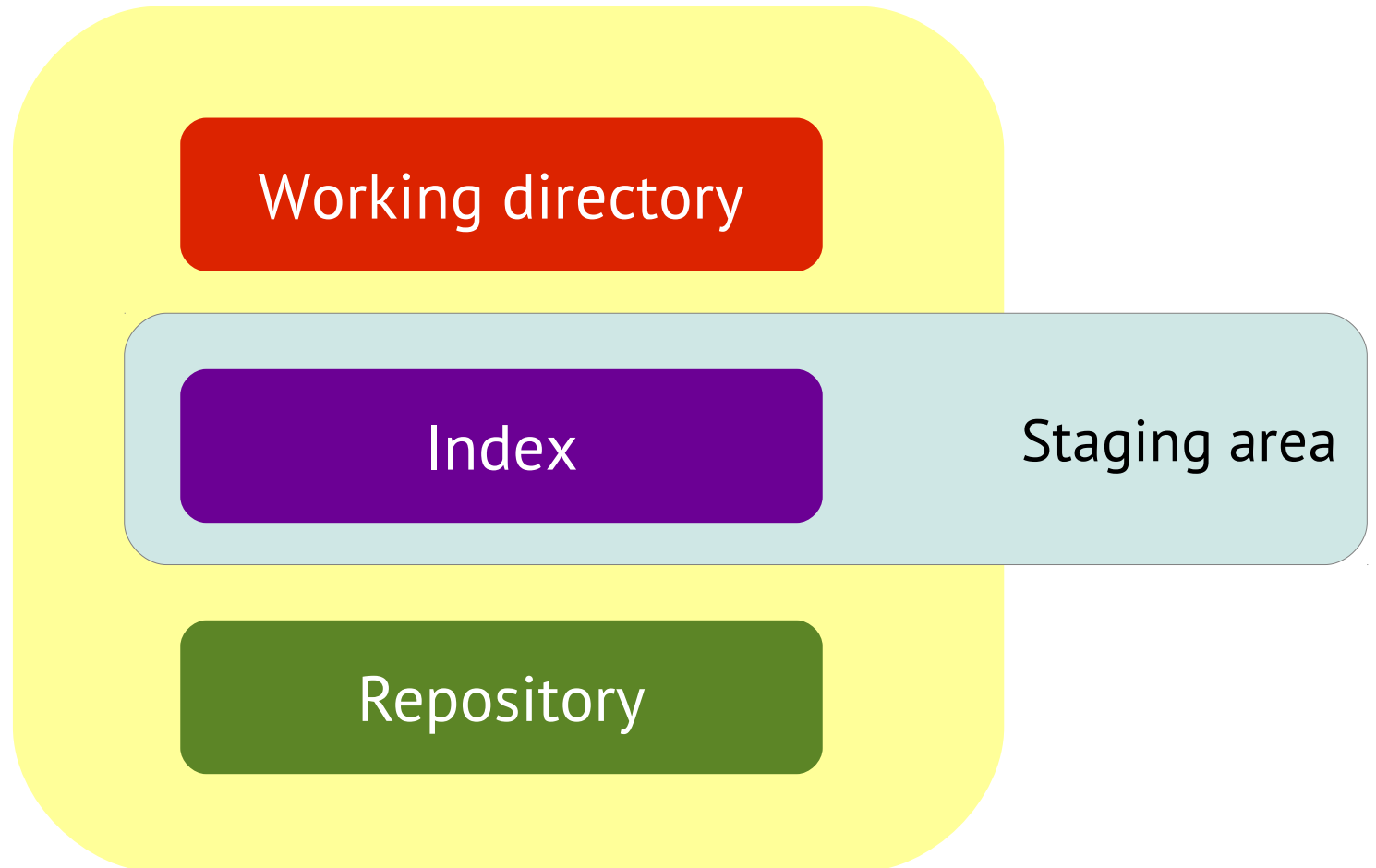
A working copy
of your project

Index

Repository

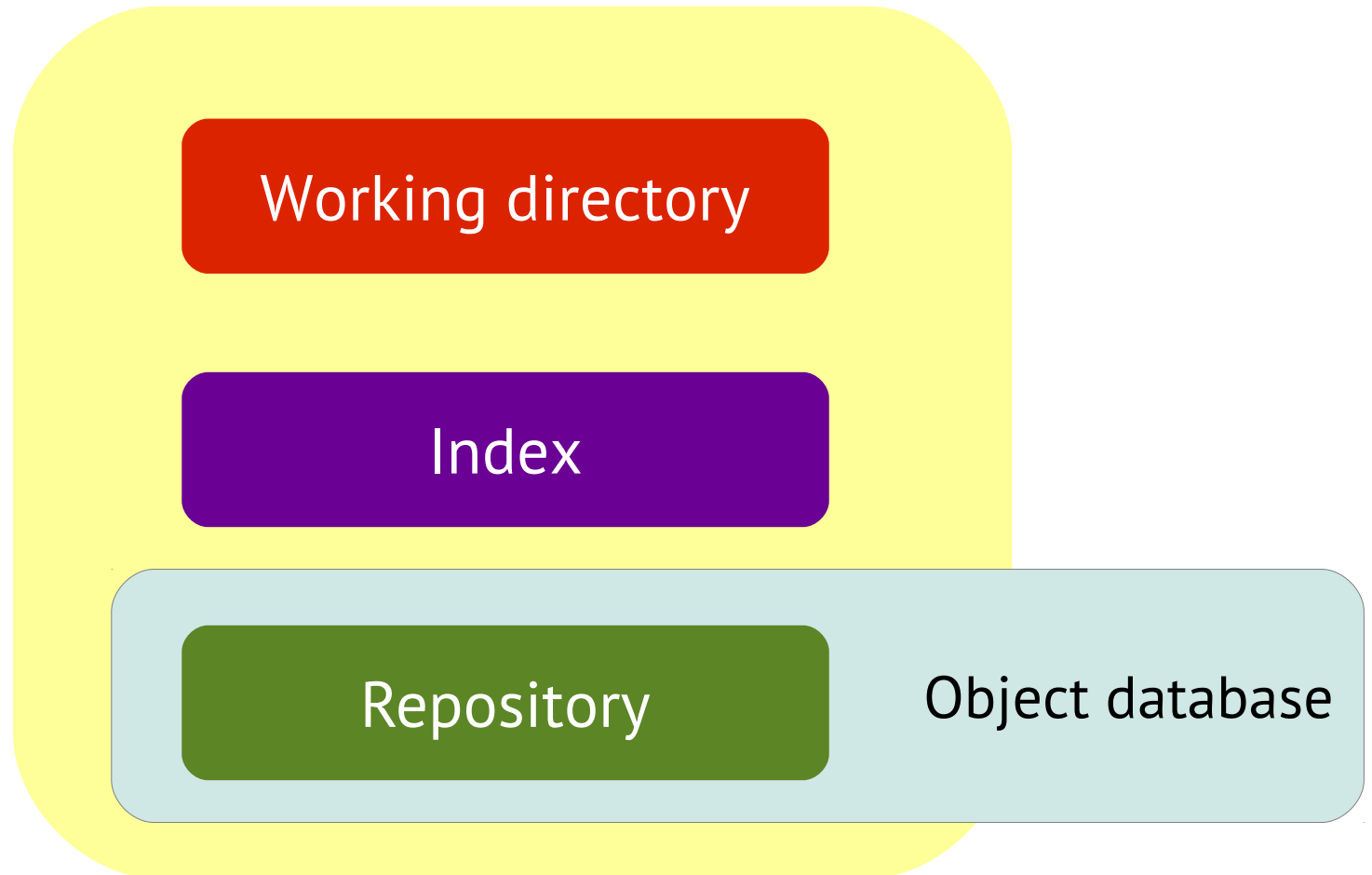
GIT workflow

Staging and committing



GIT workflow

Staging and committing



GIT workflow

Staging and committing

\$ git status

On branch master

#

Initial commit

#

nothing to commit (create/copy files and use "git add" to track)

GIT workflow

Staging and committing

```
$ git status
```

```
# On branch master
```

```
#
```

```
# Initial commit
```

```
#
```

```
nothing to commit (create/copy files and use "git add" to track)
```

GIT workflow

Staging and committing

```
$ emacs helloworld.c
```

```
$ cat helloworld.c
```

```
/* This is a classic 'hello world' app */
```

```
int
```

```
main (int argc, char** argv) {
```

```
    puts("Hello, brave new world!");
```

```
    return 0;
```

```
}
```

GIT workflow

Staging and committing

\$ git status

On branch master

#

Initial commit

#

Untracked files:

(use "git add <file>..." to include in what will be committed)

#

helloworld.c

nothing added to commit but untracked files present (use "git add" to track)

GIT workflow

Staging and committing

```
$ git status
```

```
# On branch master
```

```
#
```

```
# Initial commit
```

```
#
```

```
# Untracked files:
```

```
# (use "git add <file>..." to include in what will be committed)
```

```
#
```

```
# helloworld.c
```

```
nothing added to commit but untracked files present (use "git add"  
to track)
```


GIT workflow

Staging and committing

```
$ git add helloworld.c
```

```
$ git status
```

```
# On branch master
```

```
#
```

```
# Initial commit
```

```
#
```

```
# Changes to be committed:
```

```
#   (use "git rm --cached <file>..." to unstage)
```

```
#
```

```
#   new file:   helloworld.c
```

```
#
```

GIT workflow

Staging and committing

```
$ git add helloworld.c
```

```
$ git status
```

```
# On branch master
```

```
#
```

```
# Initial commit
```

```
#
```

```
# Changes to be committed:
```

```
#   (use "git rm --cached <file>..." to unstage)
```

```
#
```

```
#   new file:   helloworld.c
```

```
#
```

GIT workflow

Staging and committing

```
$ git commit -m "Useful comment"
```

```
[master (root-commit) 23409d6] Useful comment
```

```
1 files changed, 7 insertions(+), 0 deletions(-)
```

```
create mode 100644 helloworld.c
```

GIT workflow

Staging and committing

```
$ emacs helloworld.c
```

```
$ cat helloworld.c
```

```
/* This is a classic 'hello world' app */
```

```
#include <stdio.h>
```

```
int
```

```
main (int argc, char** argv) {
```

```
    puts("Hello, brave new world!");
```

```
    return 0;
```

```
}
```

GIT workflow

Staging and committing

\$ git status

On branch master

Changes not staged for commit:

(use "git add <file>..." to update what will be committed)

(use "git checkout -- <file>..." to discard changes in working directory)

#

modified: helloworld.c

#

no changes added to commit (use "git add" and/or "git commit -a")

GIT workflow

Staging and committing

\$ git status

On branch master

Changes not staged for commit:

(use "git add <file>..." to update what will be committed)

(use "git checkout -- <file>..." to discard changes in working directory)

#

modified: helloworld.c

#

no changes added to commit (use "git add" and/or "git commit -a")

GIT workflow

Staging and committing

\$ git status

On branch master

Changes **not staged** for commit:

(use "git add <file>..." to update what will be committed)

(use "git checkout -- <file>..." to discard changes in working directory)

#

modified: helloworld.c

#

no changes added to commit (use "git add" and/or "git commit -a")

GIT workflow

Staging and committing

\$ git diff

```
--- a/helloworld.c
+++ b/helloworld.c
@@ -1,5 +1,7 @@
    /* This is a classic 'hello world' app */

#include <stdio.h>
+
int
main (int argc, char** argv) {
    puts("Hello, brave new world!");
```


GIT workflow

Staging and committing

\$ git diff

```
--- a/helloworld.c
+++ b/helloworld.c
@@ -1,5 +1,7 @@
  /* This is a classic 'hello world' app */
```

```
+#include <stdio.h>
+
```

```
int
main (int argc, char** argv) {
    puts("Hello, brave new world!");
```

GIT workflow

Staging and committing

```
$ git add helloworld.c
```

```
$ git status
```

```
# On branch master
```

```
# Changes to be committed:
```

```
#   (use "git reset HEAD <file>..." to unstage)
```

```
#
```

```
#   modified:   helloworld.c
```

```
#
```

GIT workflow

Staging and committing

```
$ emacs helloworld.c
```

```
$ cat helloworld.c
```

```
/* This is a classic 'hello world' app */
```

```
#include <stdio.h>
```

```
int
```

```
main (int argc, char** argv) {
```

```
    printf("Hello, brave new world!\n");
```

```
    return 0;
```

```
}
```

GIT workflow

Staging and committing

\$ git status

On branch master

Changes to be committed:

(use "git reset HEAD <file>..." to unstage)

#

modified: helloworld.c

#

Changes not staged for commit:

(use "git add <file>..." to update what will be committed)

(use "git checkout -- <file>..." to discard changes in working directory)

#

modified: helloworld.c

#

GIT workflow

Staging and committing

\$ git status

On branch master

Changes to be committed:

(use "git reset HEAD <file>..." to unstage)

#

modified: helloworld.c

#

Changes not staged for commit:

(use "git add <file>..." to update what will be committed)

(use "git checkout -- <file>..." to discard changes in working directory)

#

modified: helloworld.c

#

GIT workflow

Staging and committing

\$ git diff

```
--- a/helloworld.c
```

```
+++ b/helloworld.c
```

```
@@ -4,6 +4,6 @@
```

```
int
```

```
main (int argc, char** argv) {
```

```
-    puts("Hello, brave new world!");
```

```
+    printf("Hello, brave new world!\n");
```

```
    return 0;
```

```
}
```

GIT workflow

Staging and committing

\$ git diff

```
--- a/helloworld.c
```

```
+++ b/helloworld.c
```

```
@@ -4,6 +4,6 @@
```

```
int
```

```
main (int argc, char** argv) {
```

```
-    puts("Hello, brave new world!");
```

```
+    printf("Hello, brave new world!\n");
```

```
    return 0;
```

```
}
```

WTF?

GIT workflow

Staging and committing



The diagram illustrates the Git workflow for staging and committing. It consists of a large yellow rounded rectangle containing three stacked, rounded rectangular boxes. The top box is red and labeled 'Working directory'. The middle box is purple and labeled 'Index'. The bottom box is green and labeled 'Repository'. Arrows indicate the flow from the Working directory to the Index, and then from the Index to the Repository.

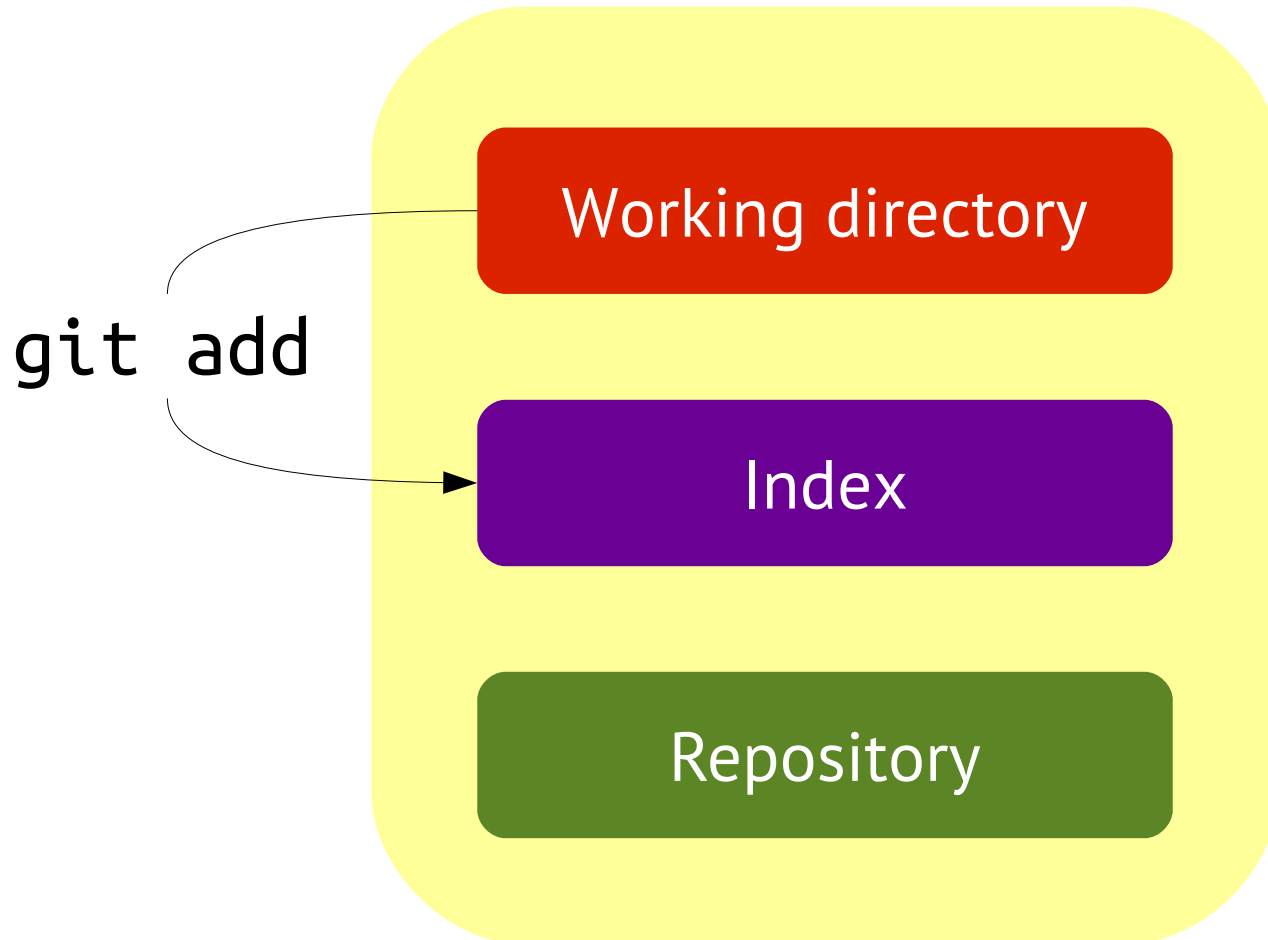
Working directory

Index

Repository

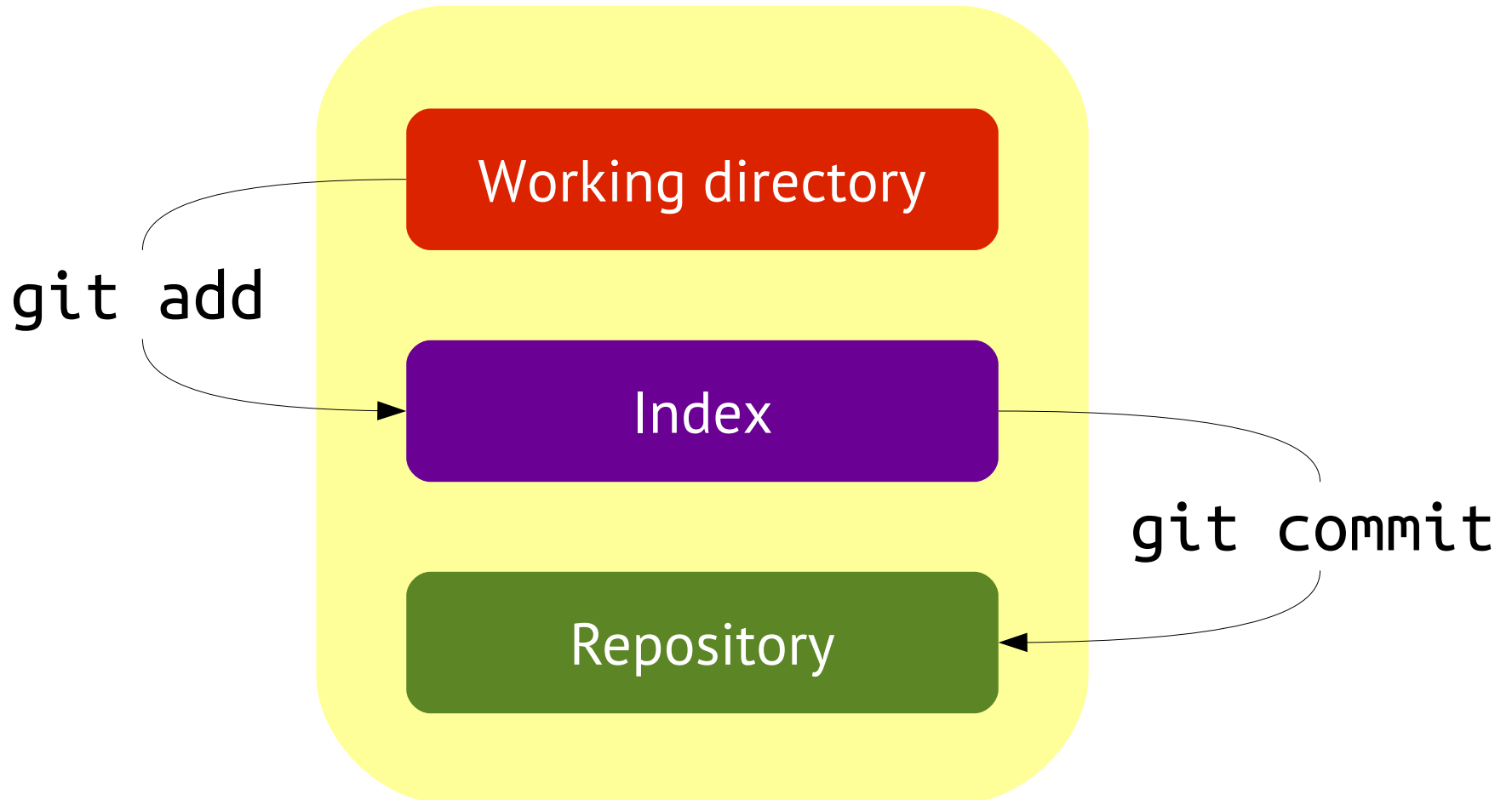
GIT workflow

Staging and committing



GIT workflow

Staging and committing



GIT workflow

Staging and committing

\$ git status

On branch master

Changes to be committed:

(use "git reset HEAD <file>..." to unstage)

#

modified: helloworld.c

#

Changes not staged for commit:

(use "git add <file>..." to update what will be committed)

(use "git checkout -- <file>..." to discard changes in working directory)

#

modified: helloworld.c

#

GIT workflow

Staging and committing

\$ git status

On branch master

Changes **that are staged**:

(use "git reset HEAD <file>..." to unstage)

#

modified: helloworld.c

#

Changes **that are not staged**:

(use "git add <file>..." to update what will be committed)

(use "git checkout -- <file>..." to discard changes in working directory)

#

modified: helloworld.c

#

GIT workflow

Staging and committing

\$ git status

On branch master

Changes to be committed:

(use "git reset HEAD <file>..." to unstage)

#

modified: helloworld.c

#

Changes not staged for commit:

(use "git add <file>..." to update what will be committed)

(use "git checkout -- <file>..." to discard changes in working directory)

#

modified: helloworld.c

#

Staged

GIT workflow

Staging and committing

\$ git status

On branch master

Changes to be committed:

(use "git reset HEAD <file>..." to unstage)

#

modified: helloworld.c

#

Changes not staged for commit:

(use "git add <file>..." to update what will be committed)

(use "git checkout -- <file>..." to discard changes in working directory)

#

modified: helloworld.c

#

Not staged

GIT workflow

Staging and committing

You have to stage a file
after you edit it

GIT workflow

Staging and committing

You have to stage a file
after you edit it

GIT workflow

Staging and committing

You have to stage a file
after you edit it

GIT workflow

Staging and committing

```
$ git commit -a
```

```
Hello world of a new generation.
```

```
# Please enter the commit message for your changes. Lines starting
```

```
# with '#' will be ignored, and an empty message aborts the commit.
```

```
# On branch master
```

```
# Changes to be committed:
```

```
#   (use "git reset HEAD <file>..." to unstage)
```

```
#
```

```
#       modified:   helloworld.c
```

```
#
```

GIT workflow

Staging and committing

```
[master 4bbbef9] Hello world of a new generation.
```

```
1 files changed, 3 insertions(+), 1 deletions(-)
```

GIT workflow

Staging and committing

[master **4bbbef9**] Hello world of a new generation.

1 files changed, 3 insertions(+), 1 deletions(-)

GIT workflow

Staging and committing

4bbbef9

GIT workflow

Staging and committing

4bbbf94bbbf9b703c8197ec34cb90382c7b8fd5046a9a

GIT workflow

Staging and committing

```
$ tree -a
```

```
.
├── .git
│   ├── branches
│   ├── COMMIT_EDITMSG
│   ├── config
│   ├── description
│   ├── HEAD
│   ├── hooks
│   │   ├── applypatch-msg.sample
│   │   └── ...
│   ├── index
│   ├── info
│   │   └── exclude
│   ├── logs
│   │   ├── HEAD
│   │   └── refs
│   │       └── heads
│   │           └── master
│   ├── objects
│   │   ├── 23
│   │   │   └── 409d6895e7cb65306d4ed7d0c558ced6c82753
│   │   └── ...
│   ├── info
│   │   └── pack
│   └── refs
│       ├── heads
│       │   └── master
│       └── tags
└── helloworld.c
```


GIT workflow

Staging and committing

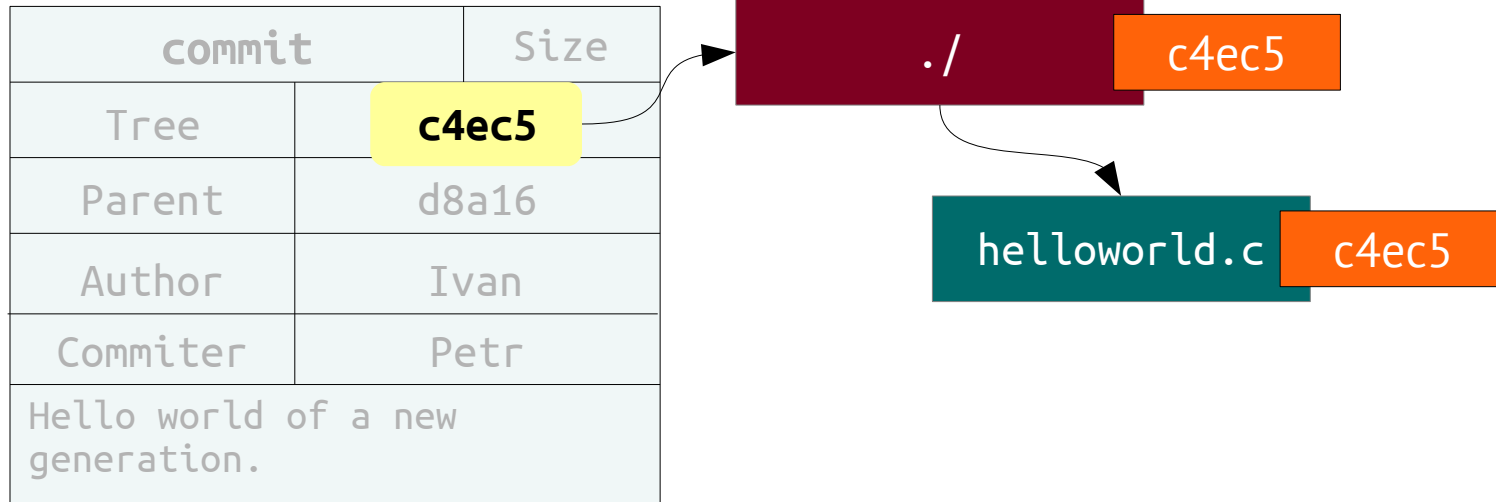
4bbbef94bbbef9b703c8197ec34cb90382c7b8fd5046a9a

commit		Size
Tree	c4ec5	
Parent	d8a16	
Author	Ivan	
Committer	Petr	
Hello world of a new generation.		

GIT workflow

Staging and committing

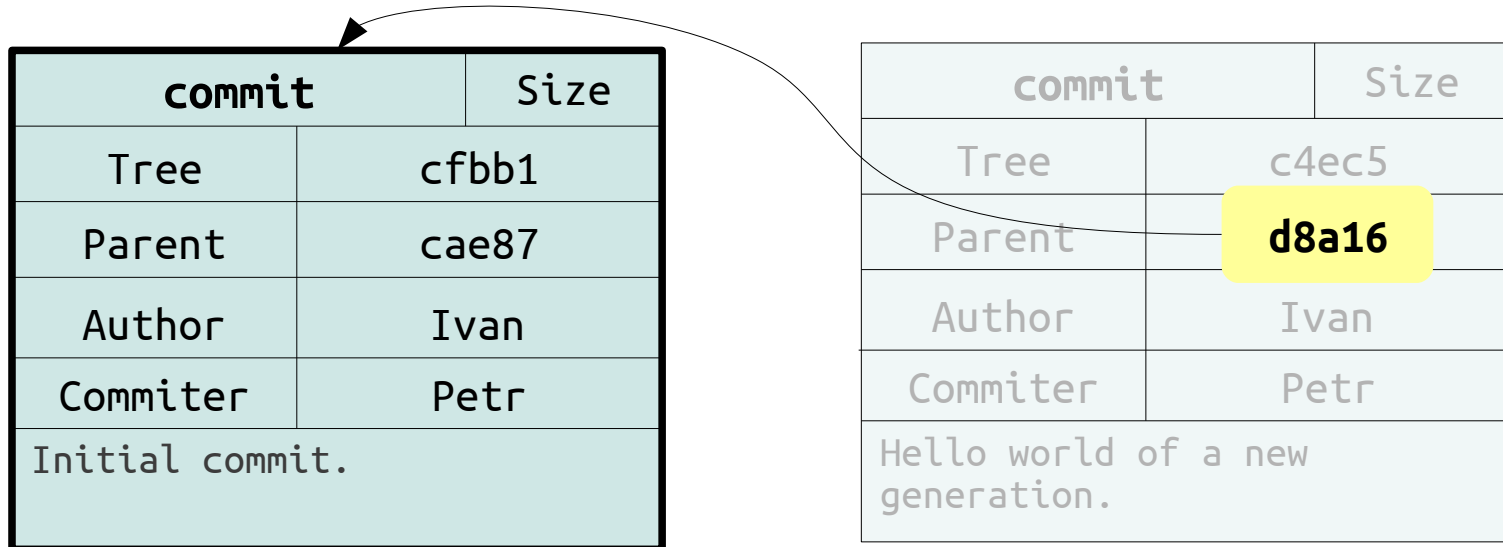
4bbbf94bbbf9b703c8197ec34cb90382c7b8fd5046a9a



GIT workflow

Staging and committing

4bbbef94bbbef9b703c8197ec34cb90382c7b8fd5046a9a



GIT workflow

Staging and committing

Repository

c1e	fa8	135
47b	aba	a5b
ccc	cfe	c0d
0aa	17f	ffe
9b4	400	813

GIT workflow

Staging and committing

Repository

c1e	fa8	135
47b	aba	a5b
ccc	cfe	c0d
0aa	17f	ffe
9b4	400	813



```
git checkout 47b
```

GIT workflow

Staging and committing

Repository

c1e	fa8	135
47b	aba	a5b
ccc	cfe	c0d
0aa	17f	ffe
9b4	400	813

git checkout 47b

GIT workflow

Staging and committing

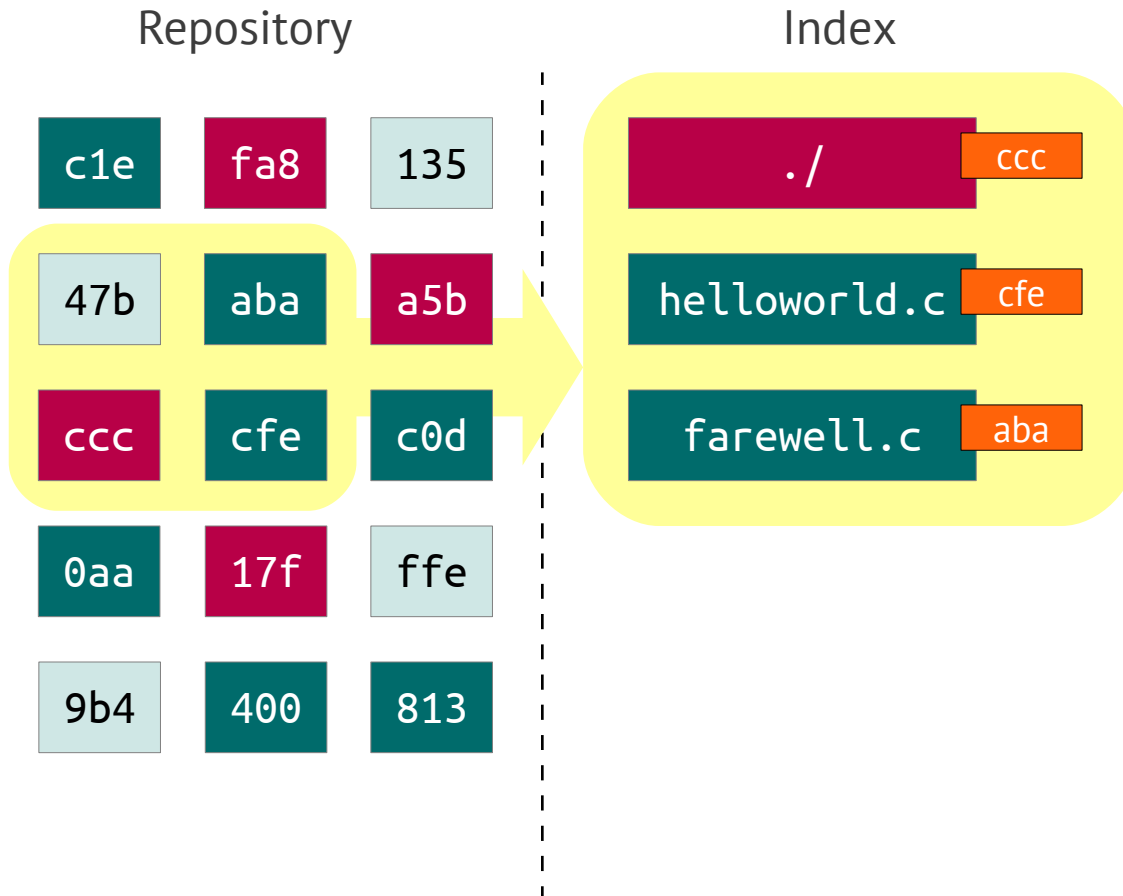
Repository

c1e	fa8	135
47b	aba	a5b
ccc	cfe	c0d
0aa	17f	ffe
9b4	400	813

git checkout 47b

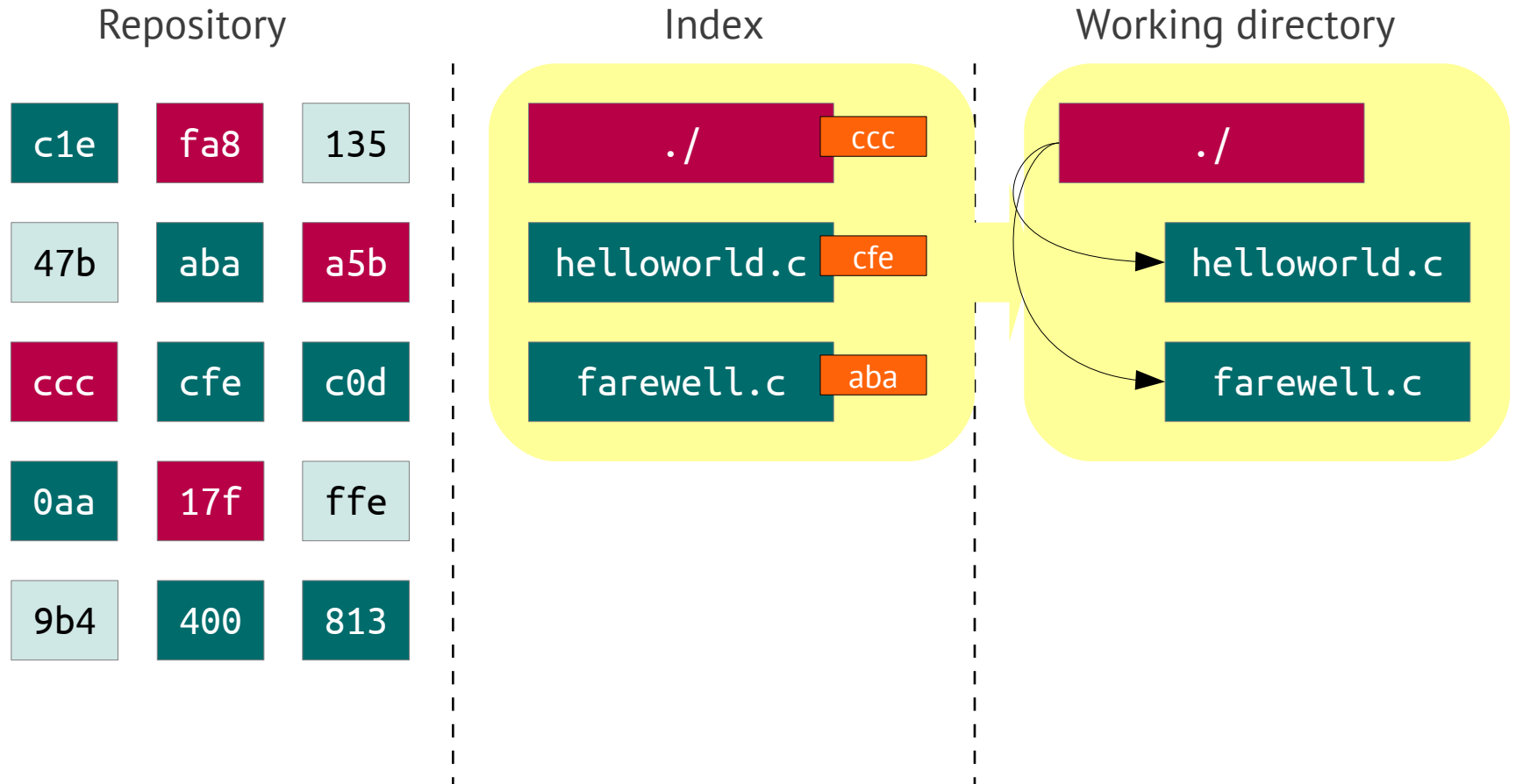
GIT workflow

Staging and committing



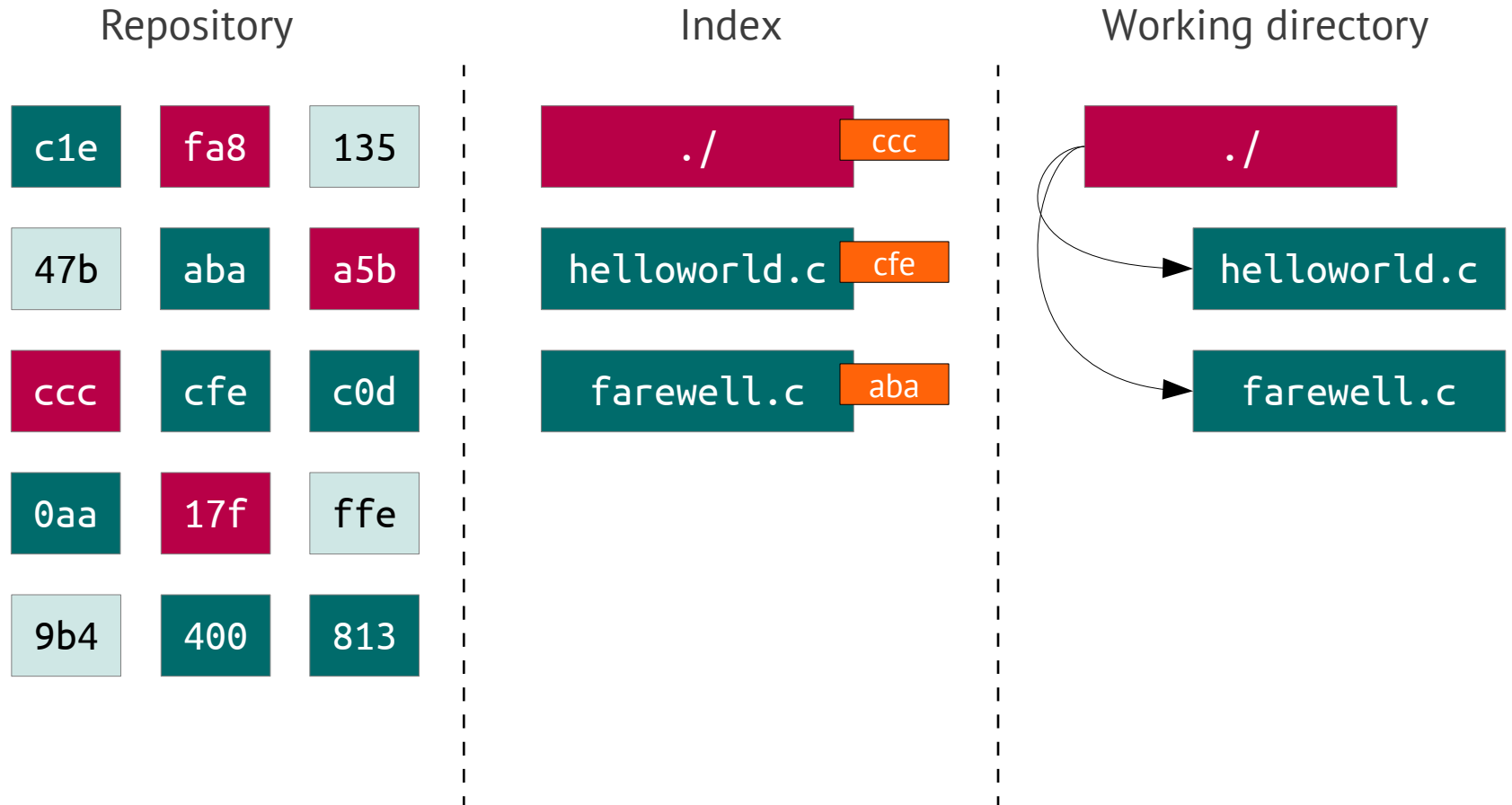
GIT workflow

Staging and committing



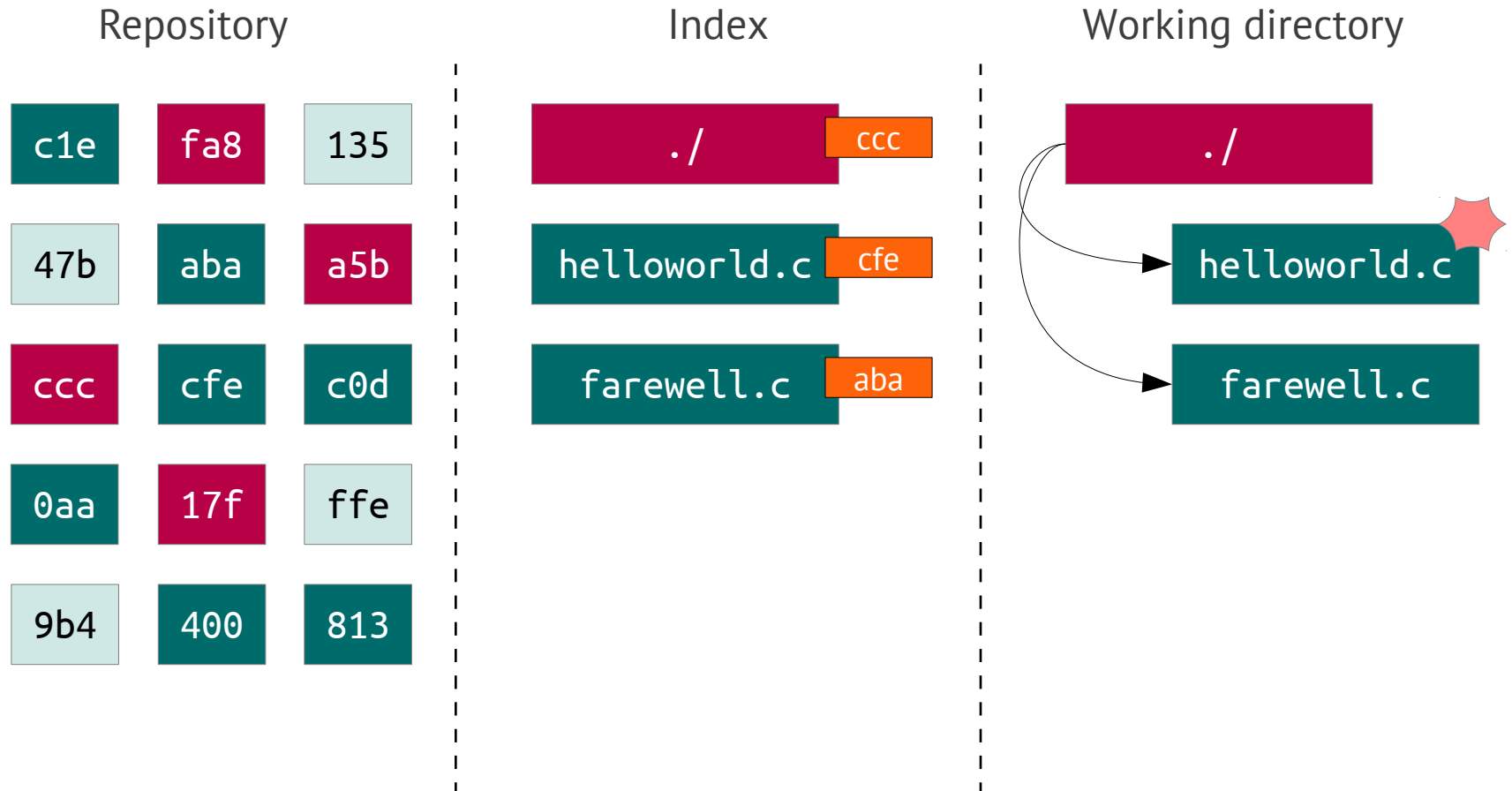
GIT workflow

Staging and committing



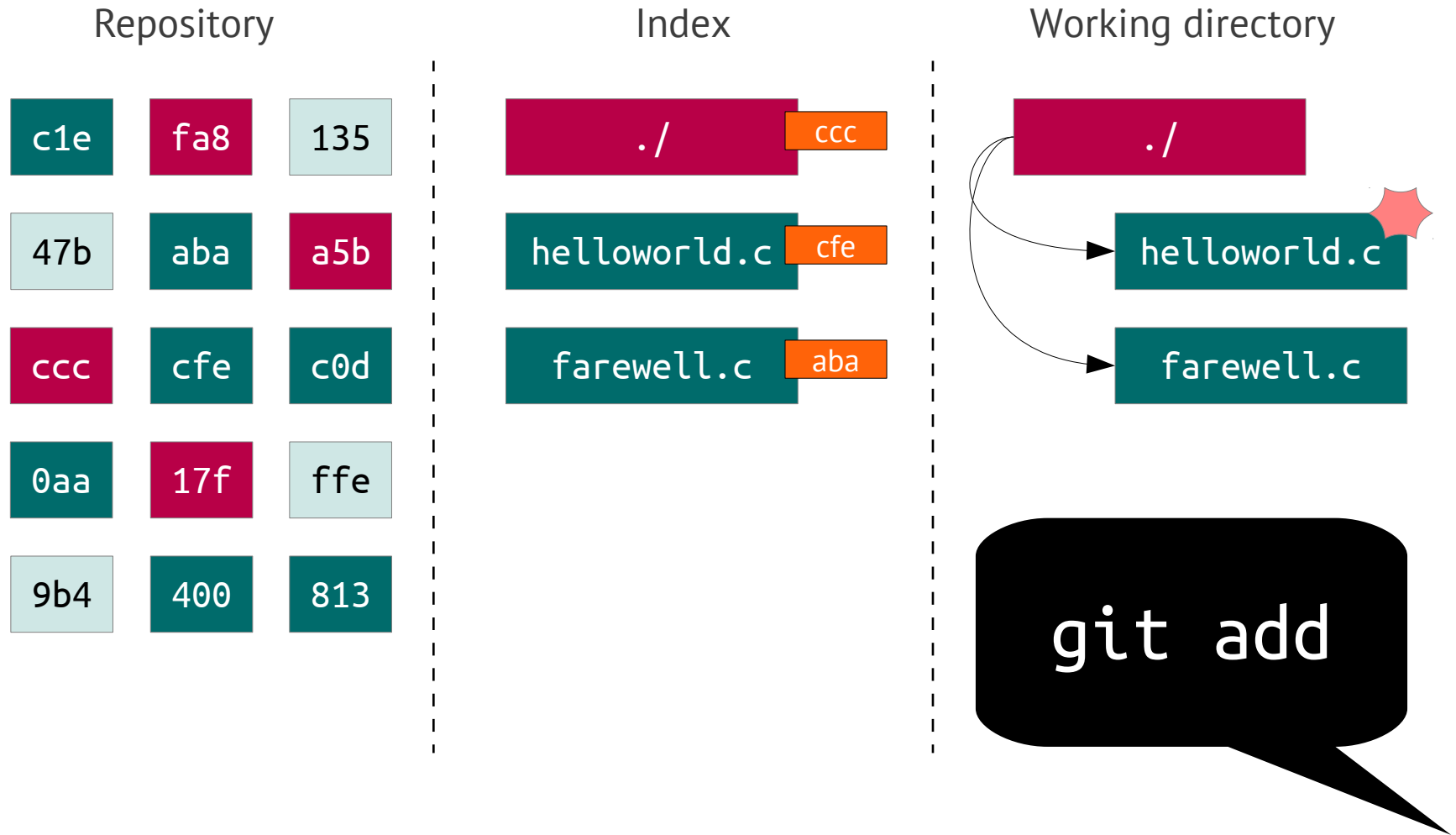
GIT workflow

Staging and committing



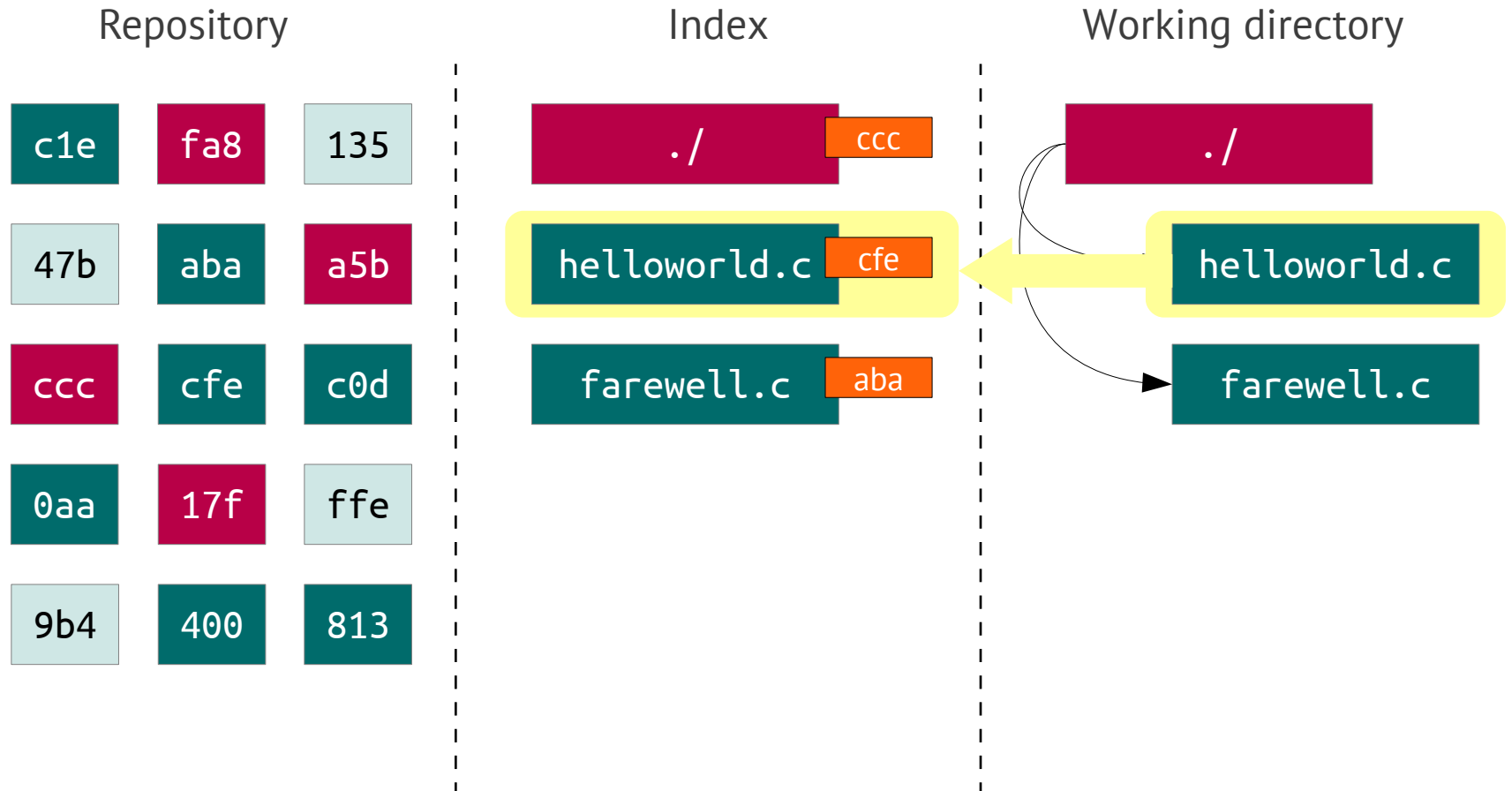
GIT workflow

Staging and committing



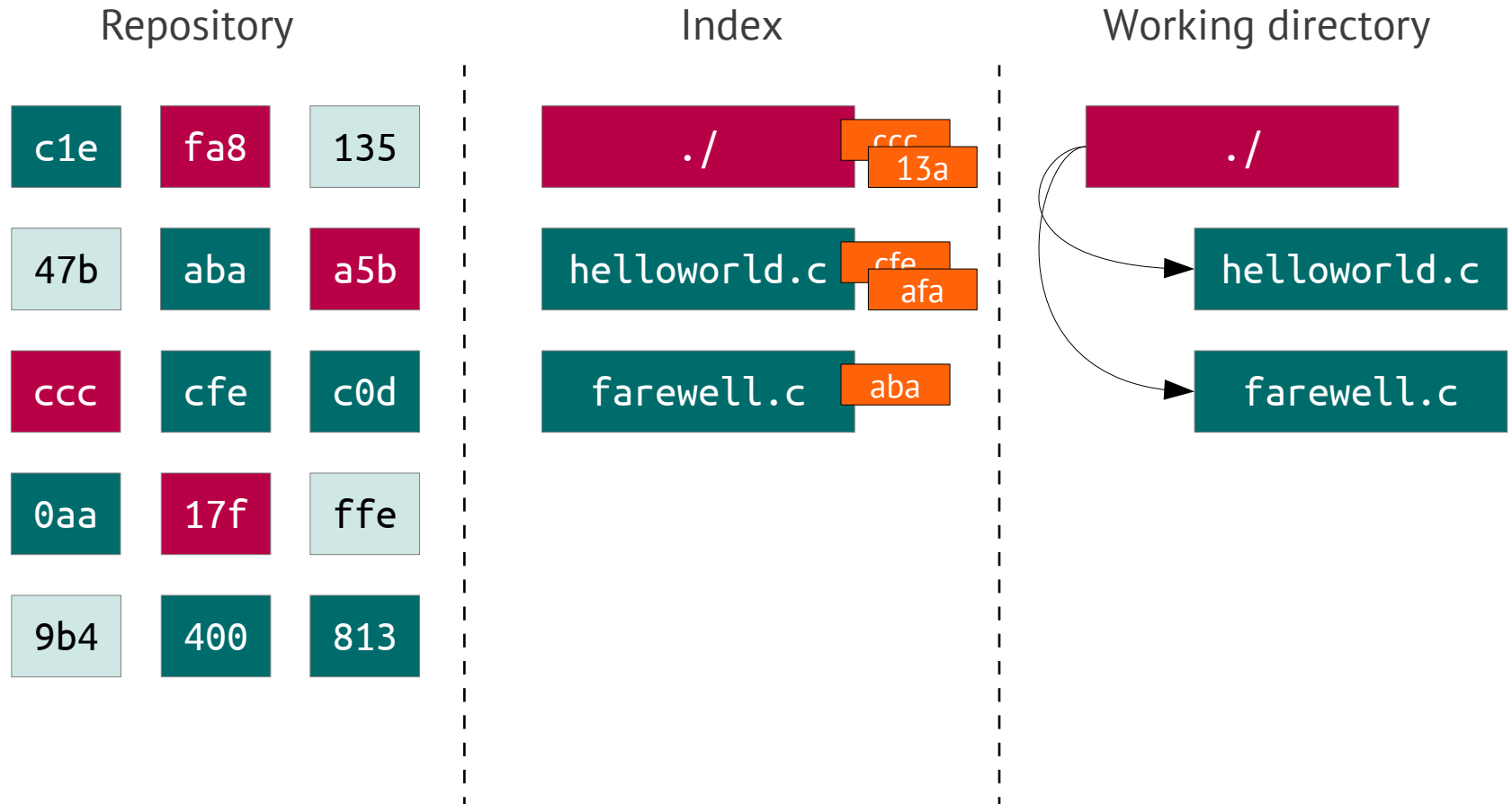
GIT workflow

Staging and committing



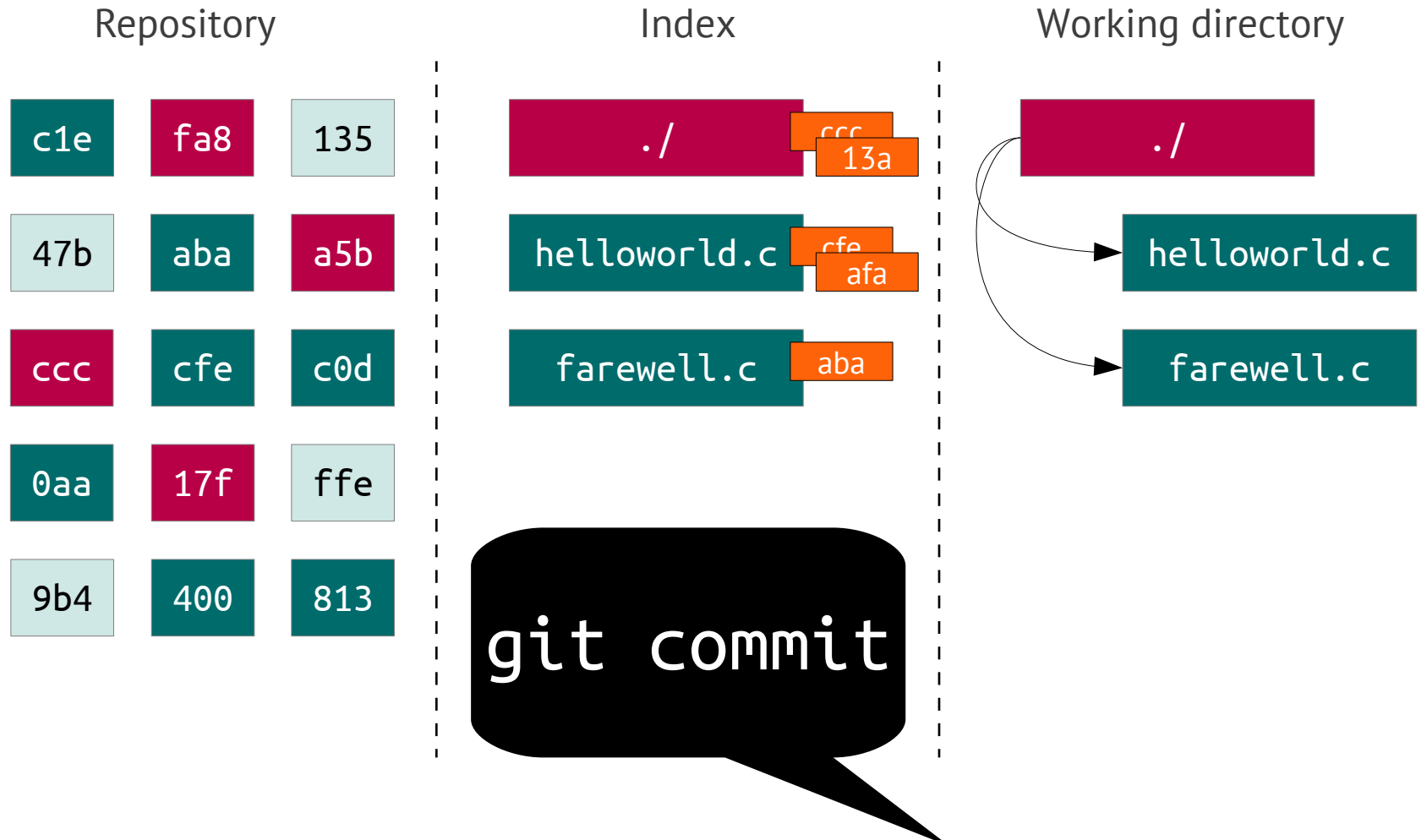
GIT workflow

Staging and committing



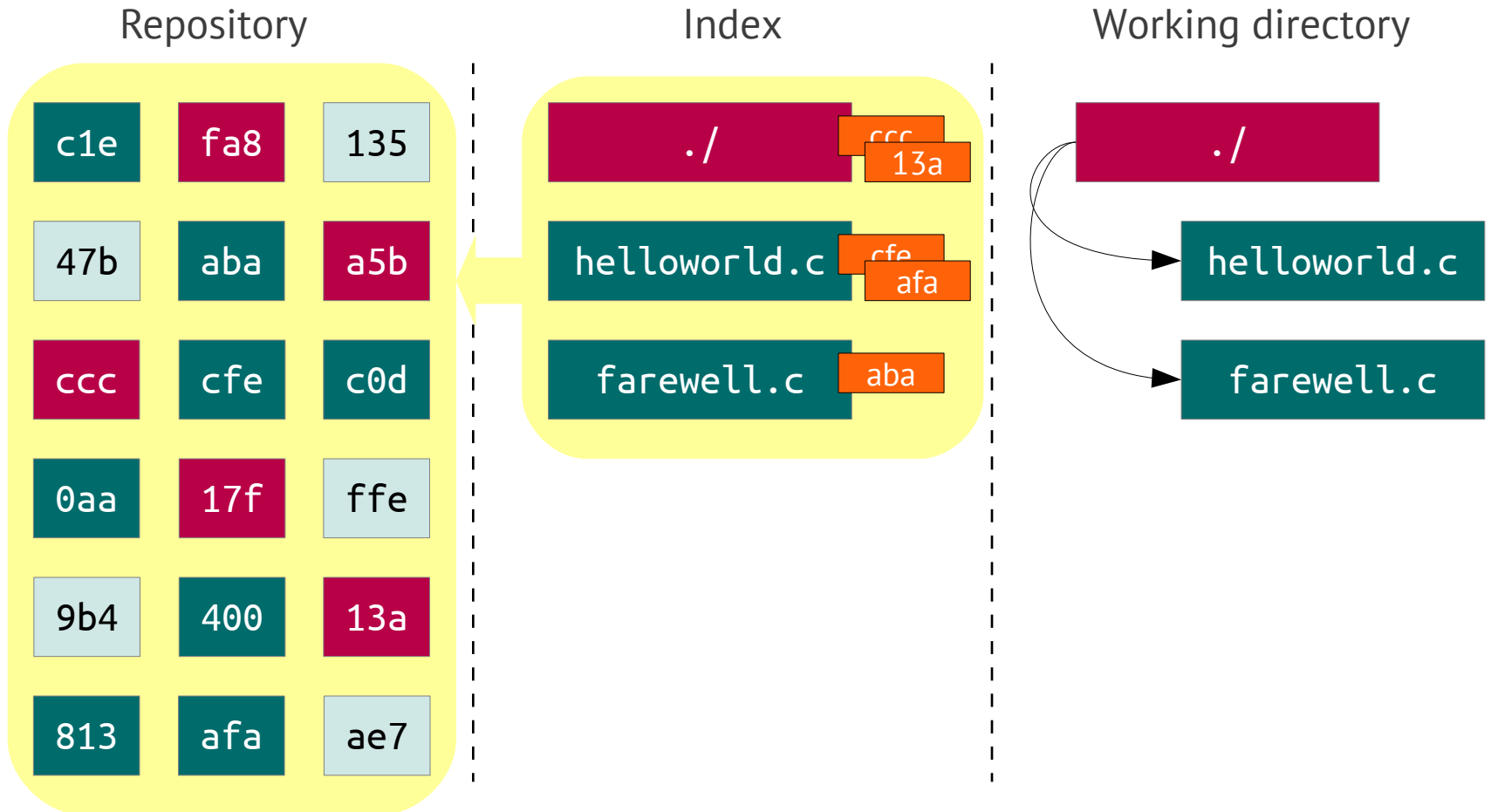
GIT workflow

Staging and committing



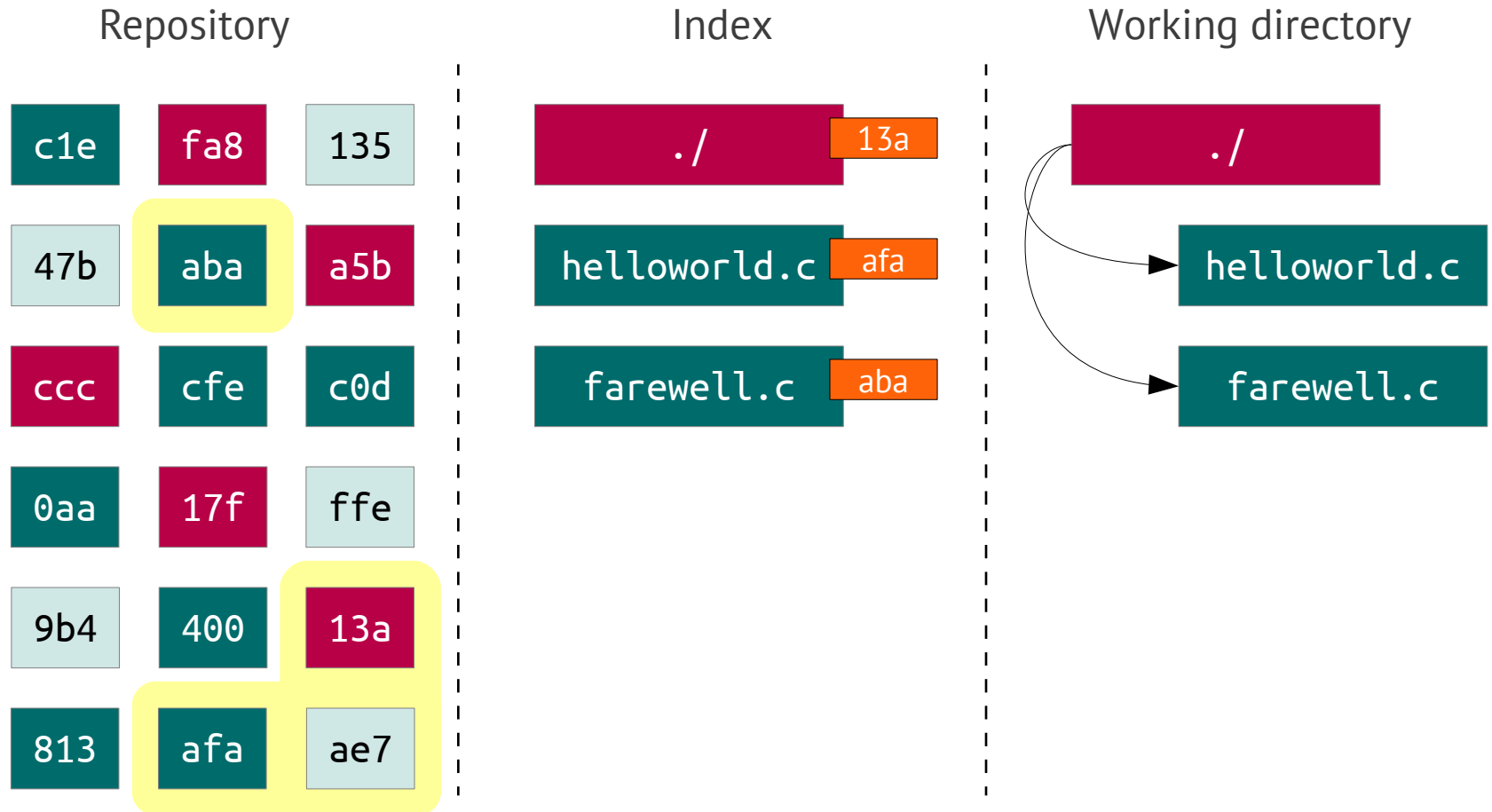
GIT workflow

Staging and committing



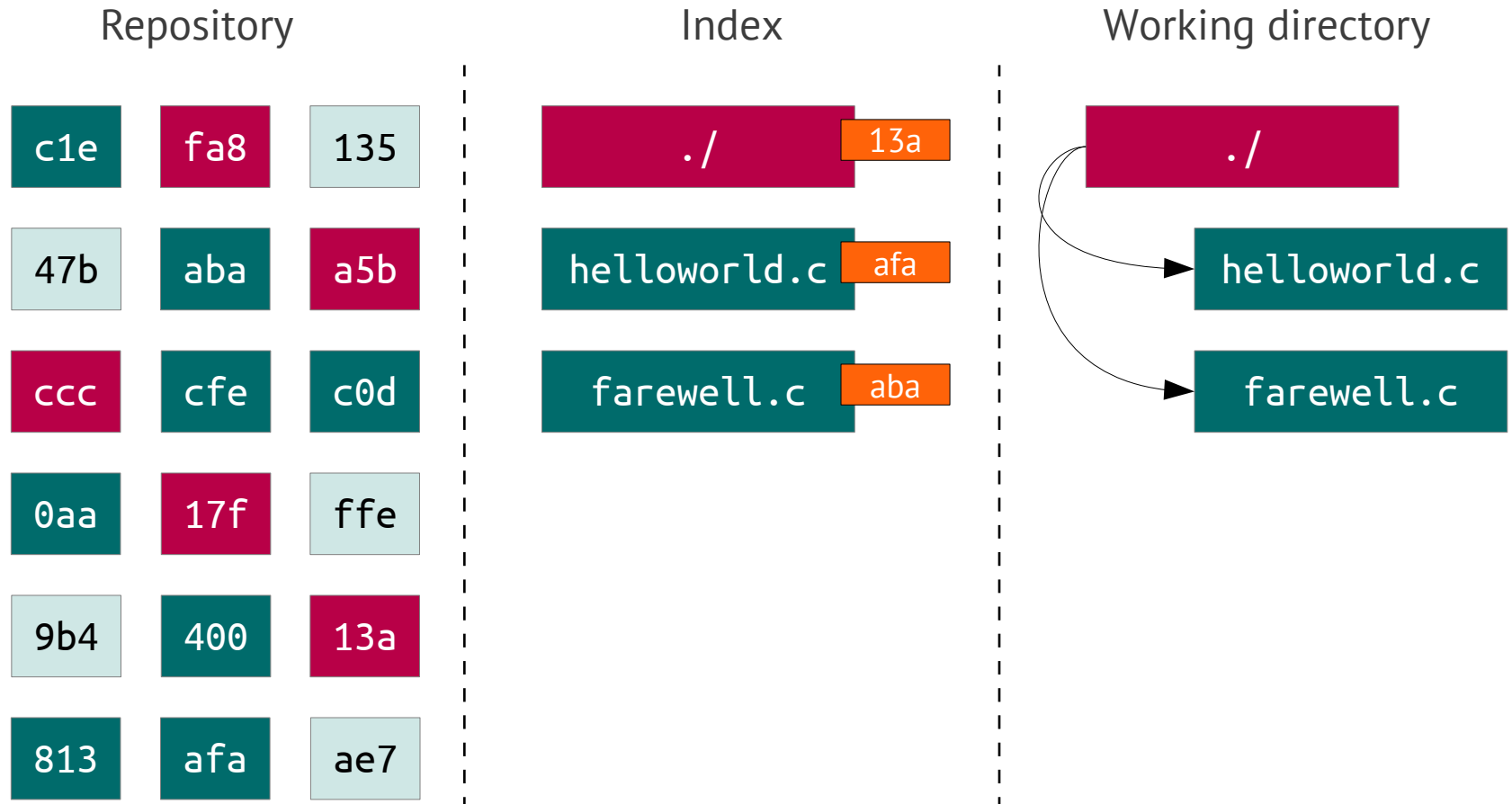
GIT workflow

Staging and committing



GIT workflow

Staging and committing



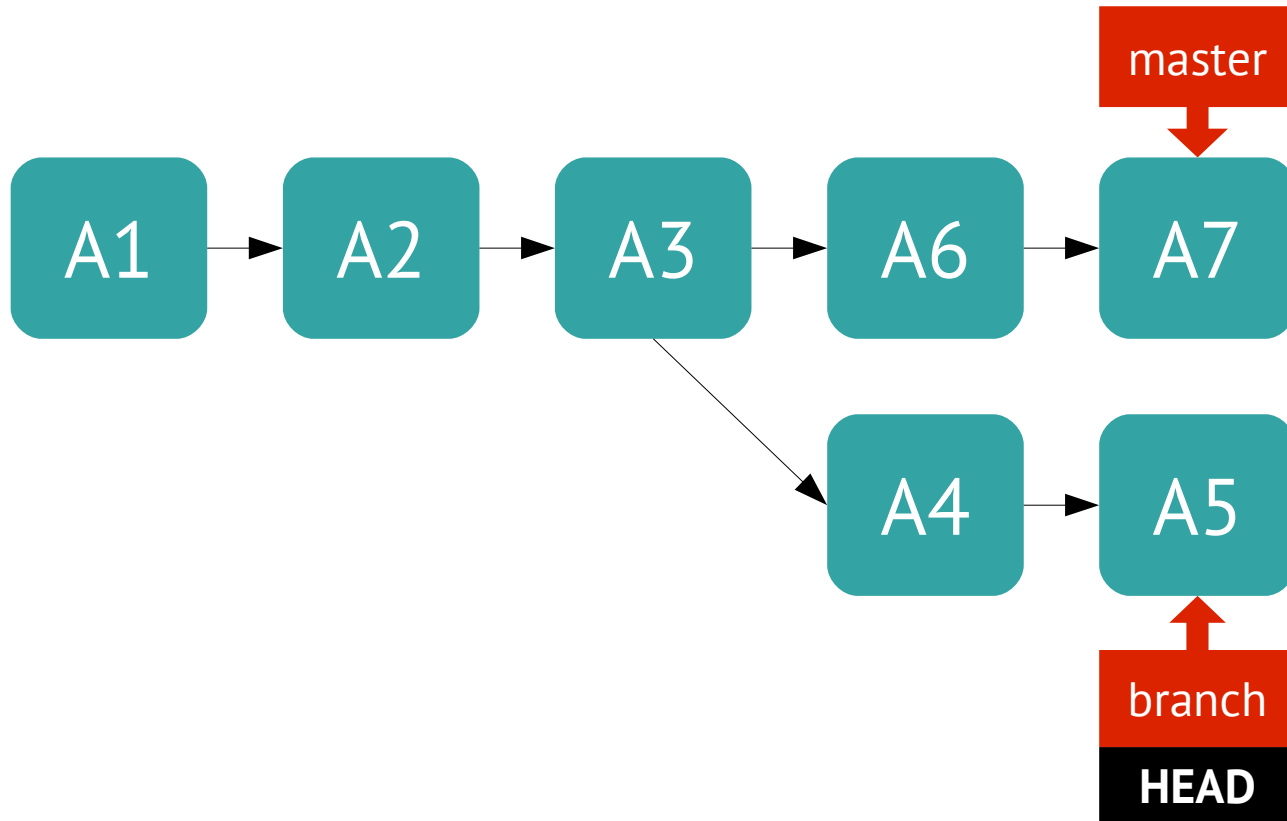
GIT workflow

- ~~Create your branch~~
- ~~Edit files~~
- ~~Stage your changes~~
- ~~Commit the changes~~
- **Rebase your branch from master**
- Merge to master
- Push to central repository

Rebasing

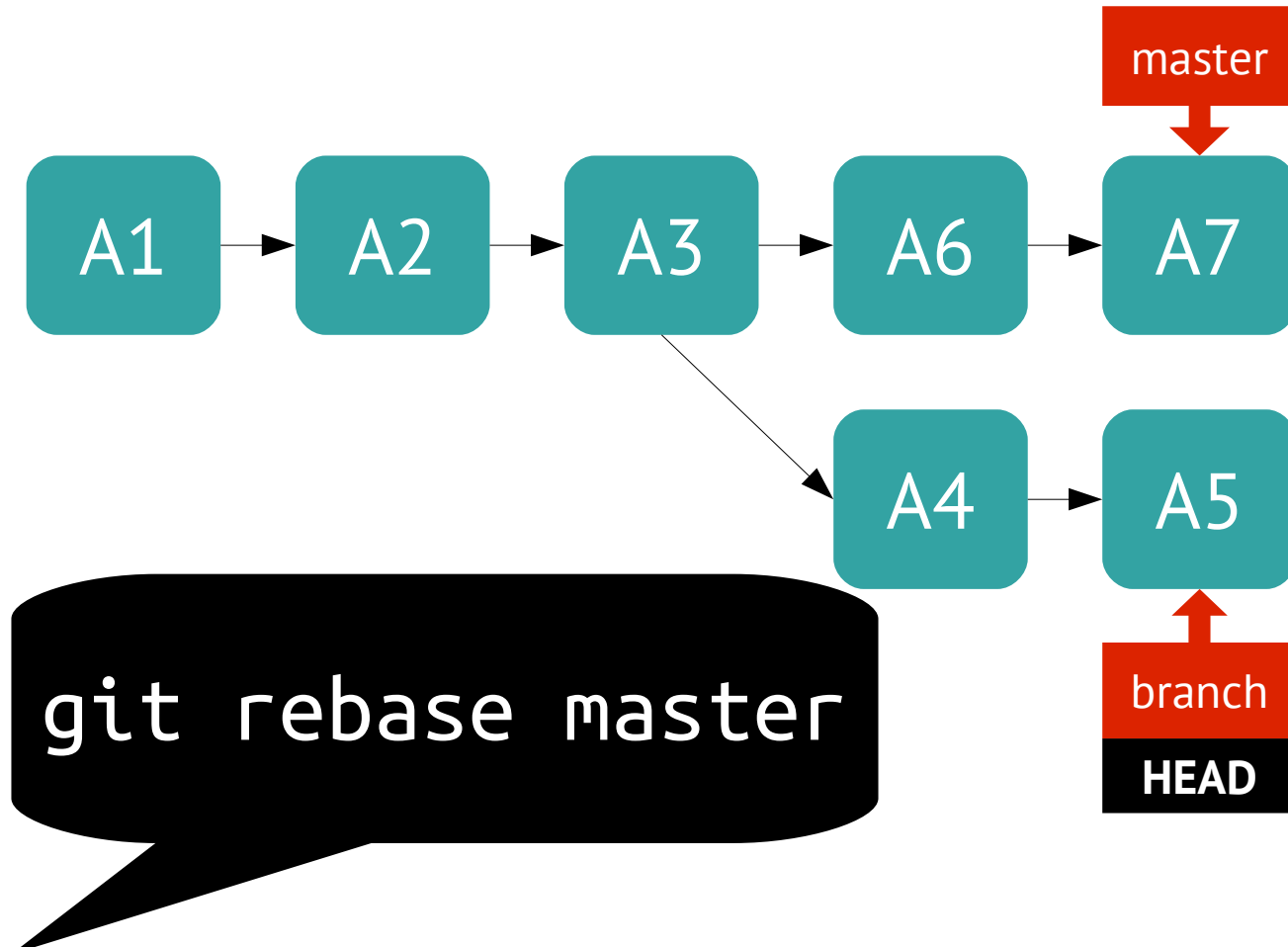
GIT workflow

Rebasing



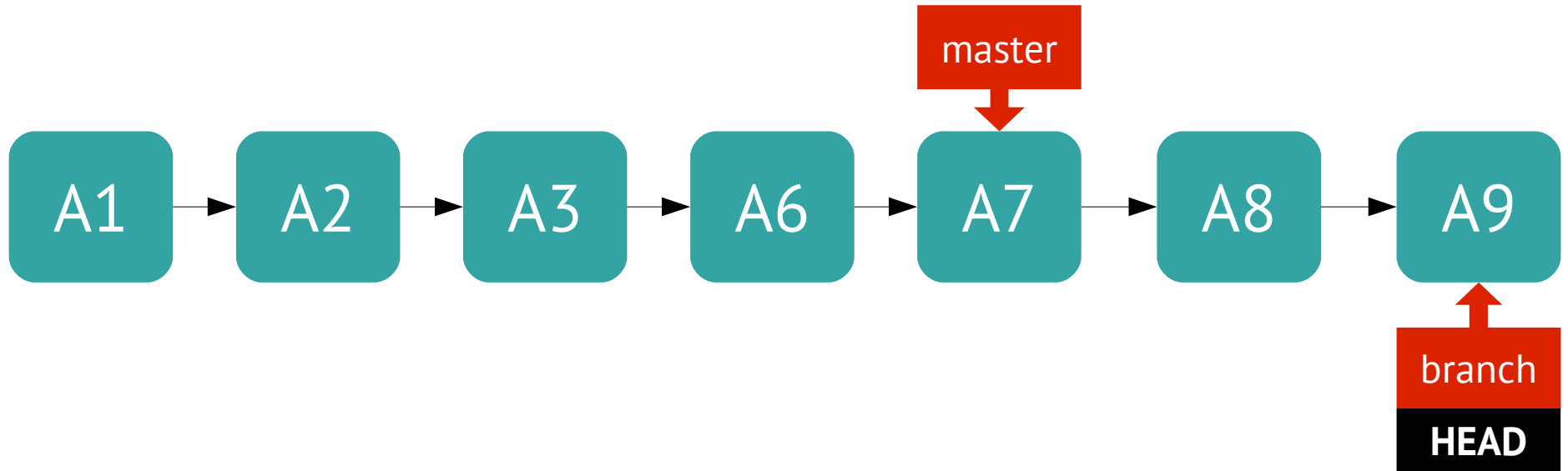
GIT workflow

Rebasing



GIT workflow

Rebasing



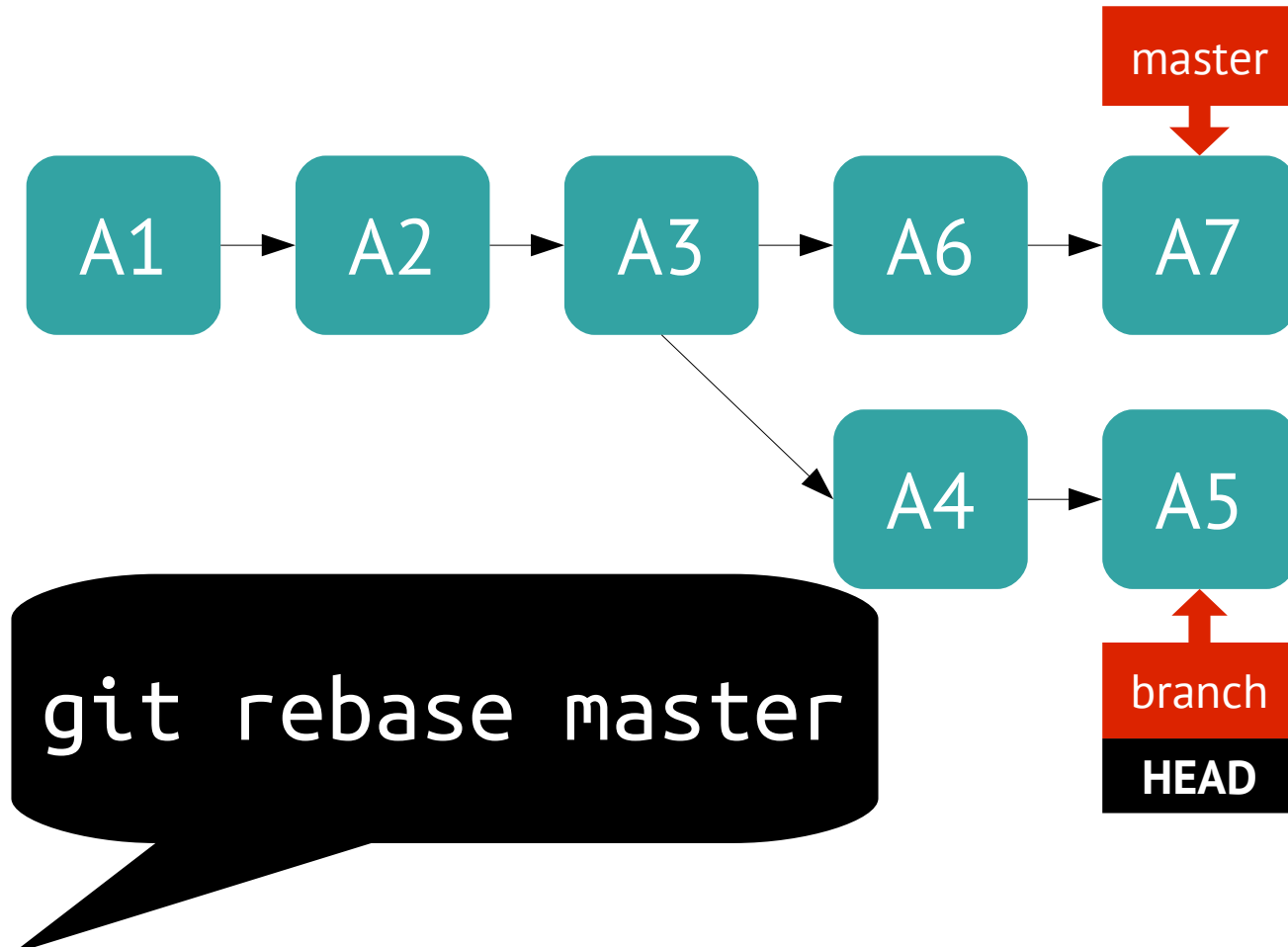
Huh?

Where is my A4 and A5?

What is happening?

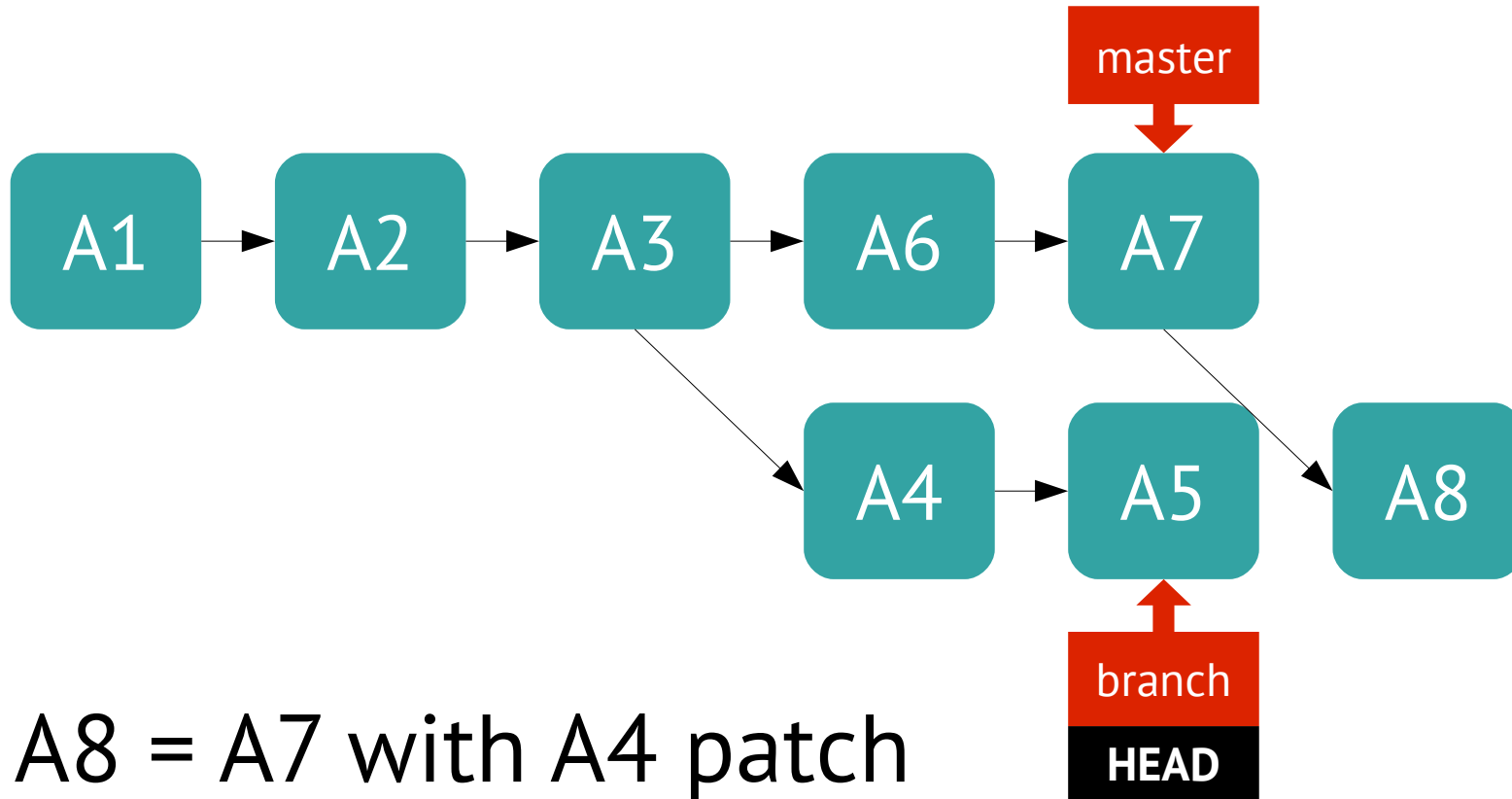
GIT workflow

Rebasing



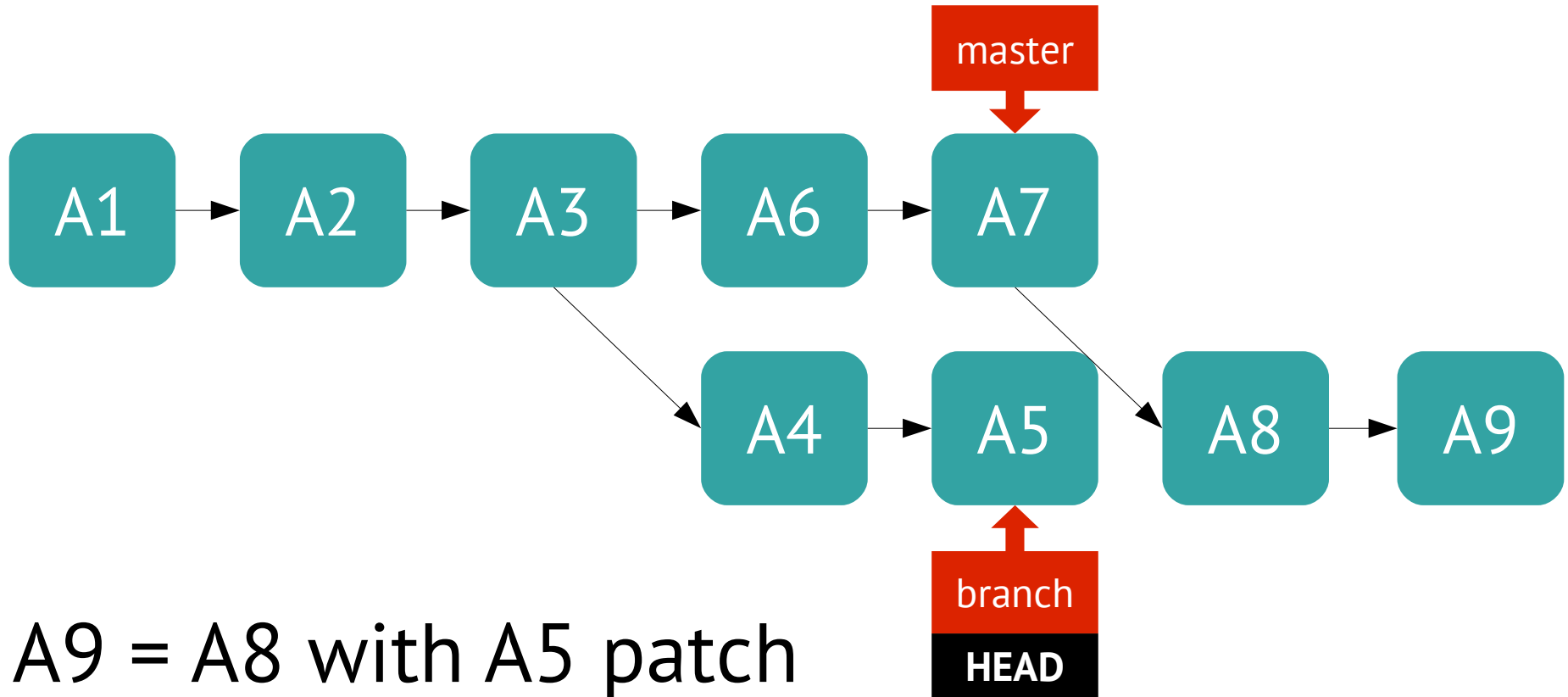
GIT workflow

Rebasing



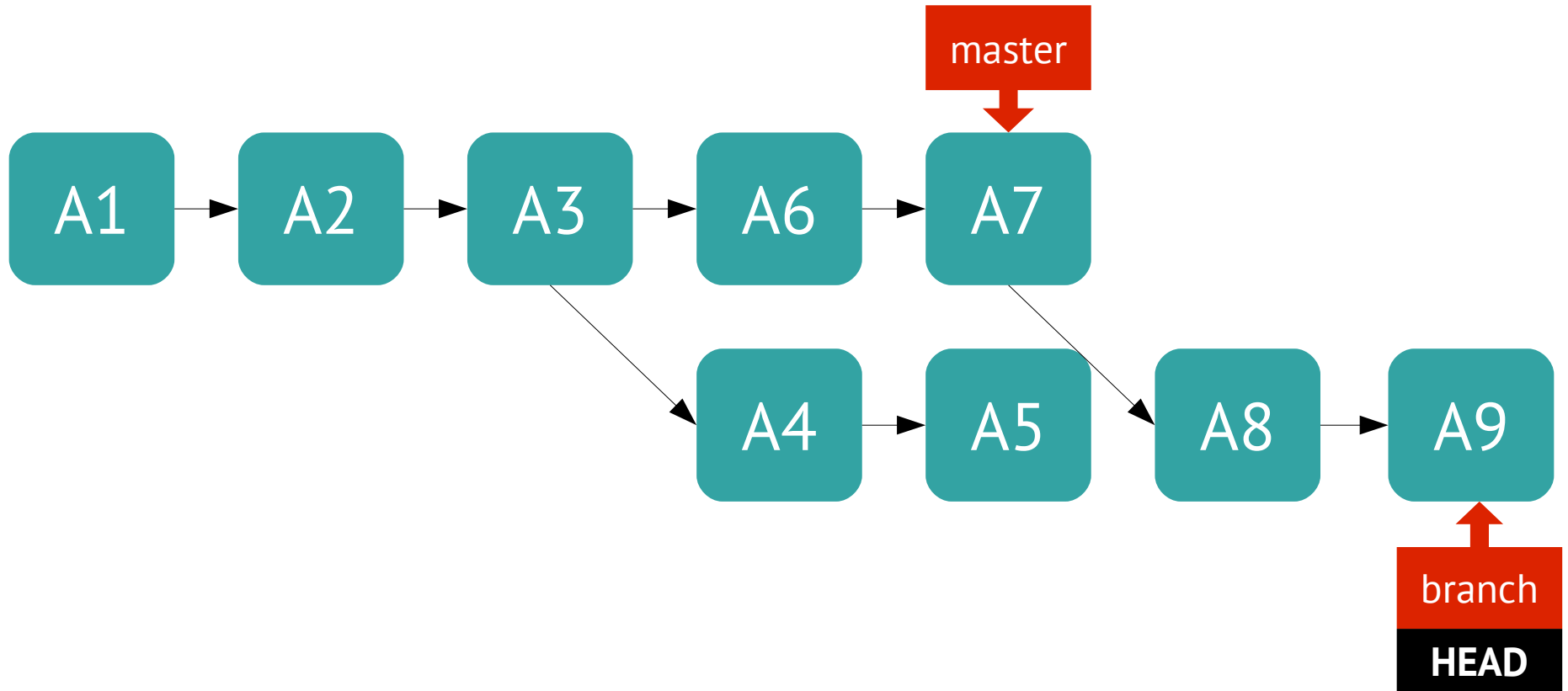
GIT workflow

Rebasing



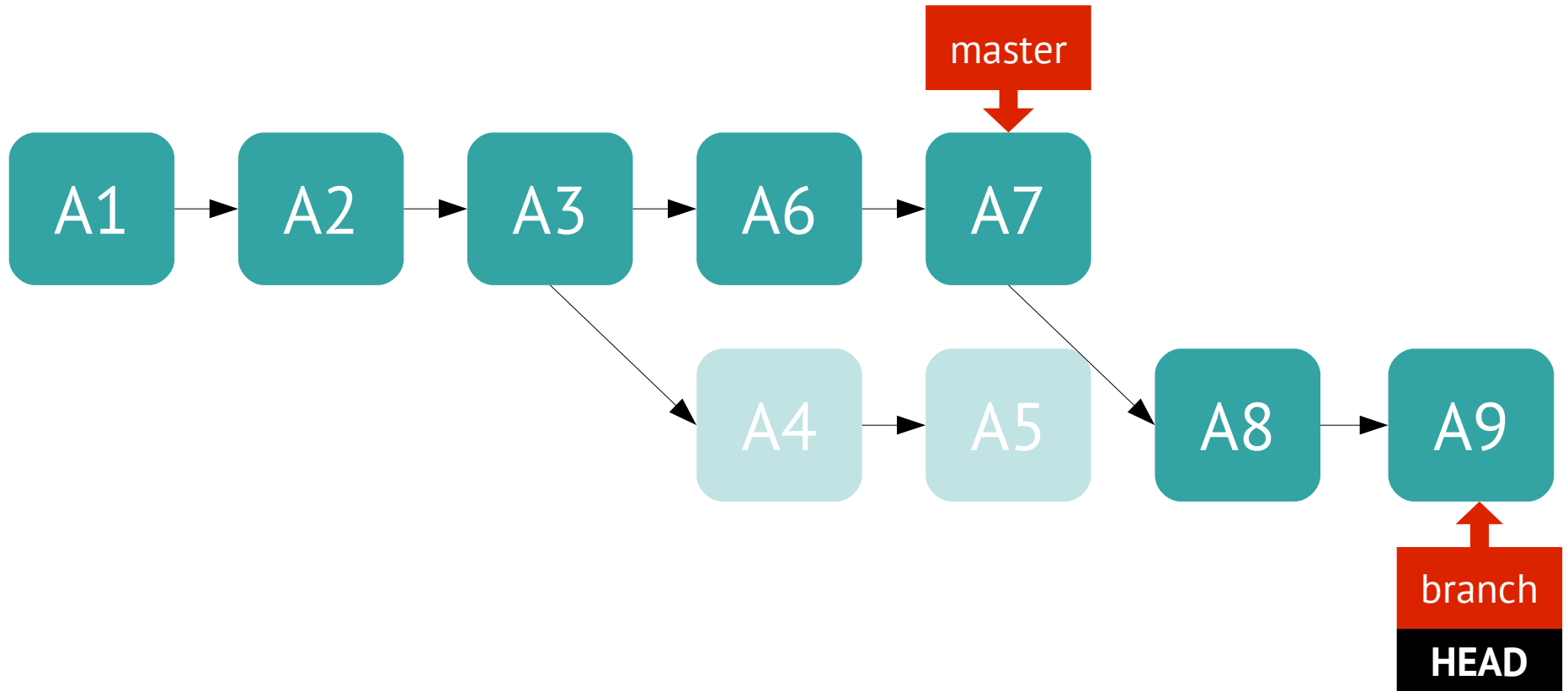
GIT workflow

Rebasing



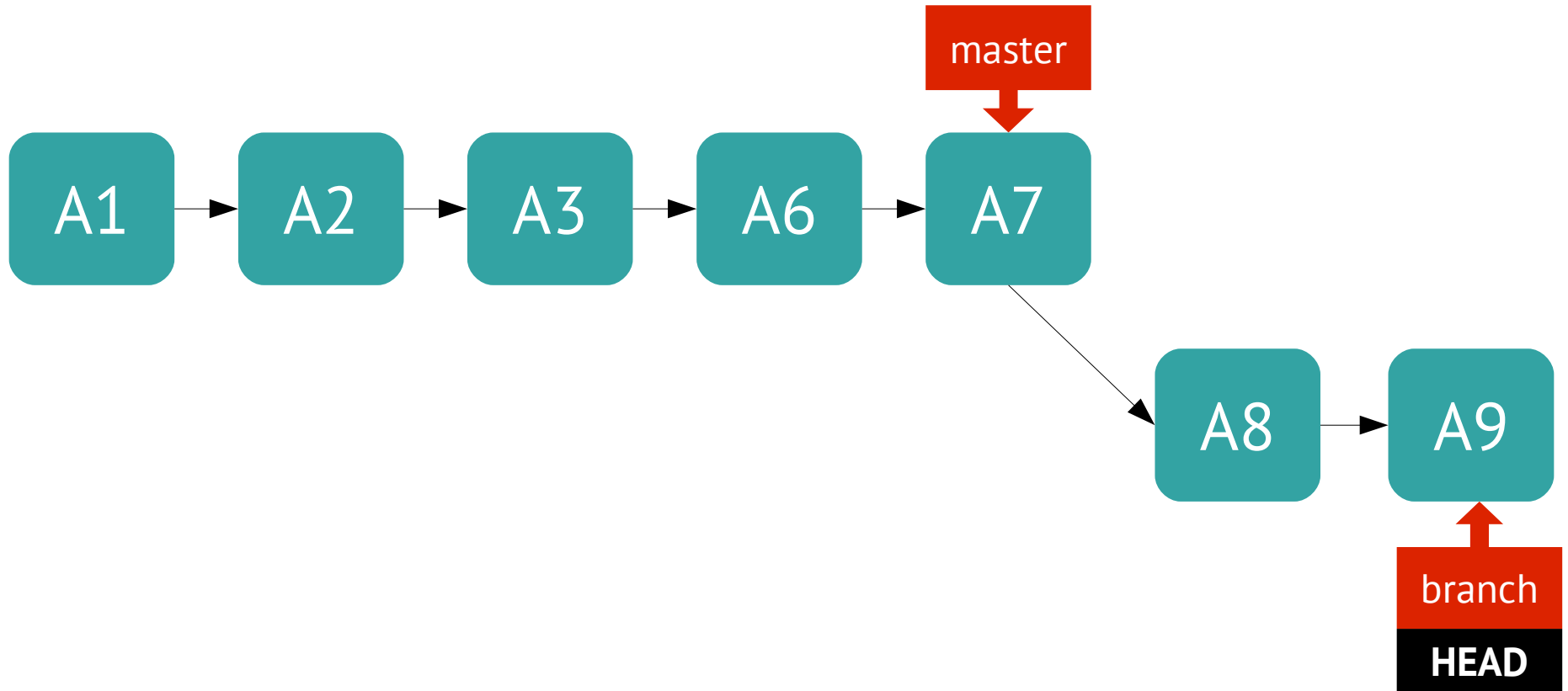
GIT workflow

Rebasing



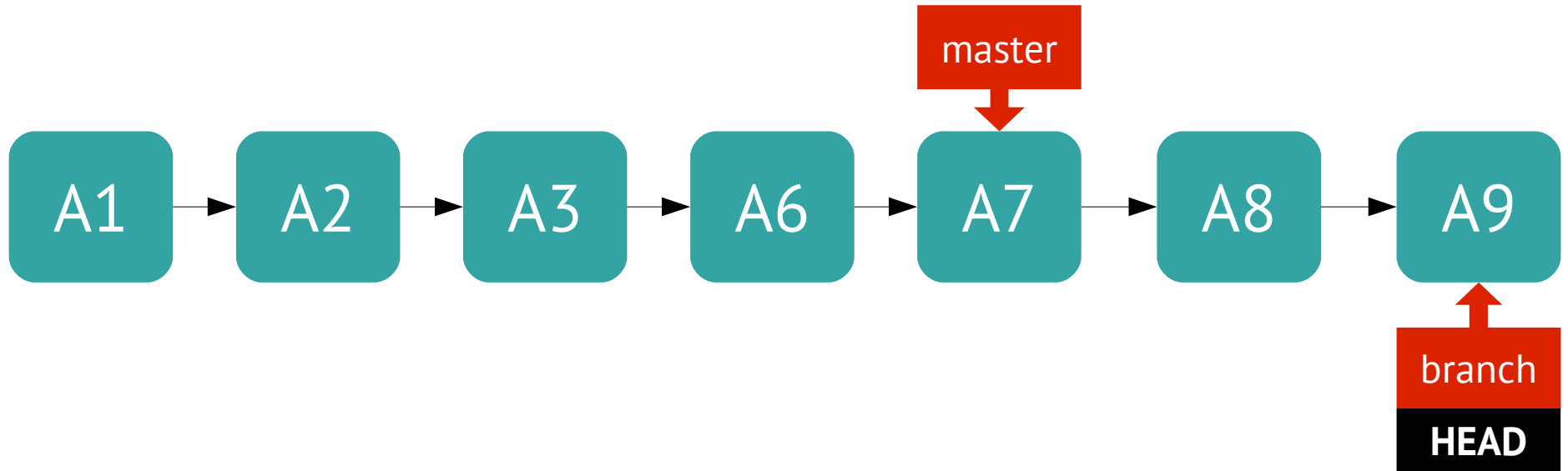
GIT workflow

Rebasing



GIT workflow

Rebasing



GIT workflow

Rebasing

Never rebase when you are on a
shared branch

GIT workflow

Rebasing

**Never rebase when you are on a
shared branch**

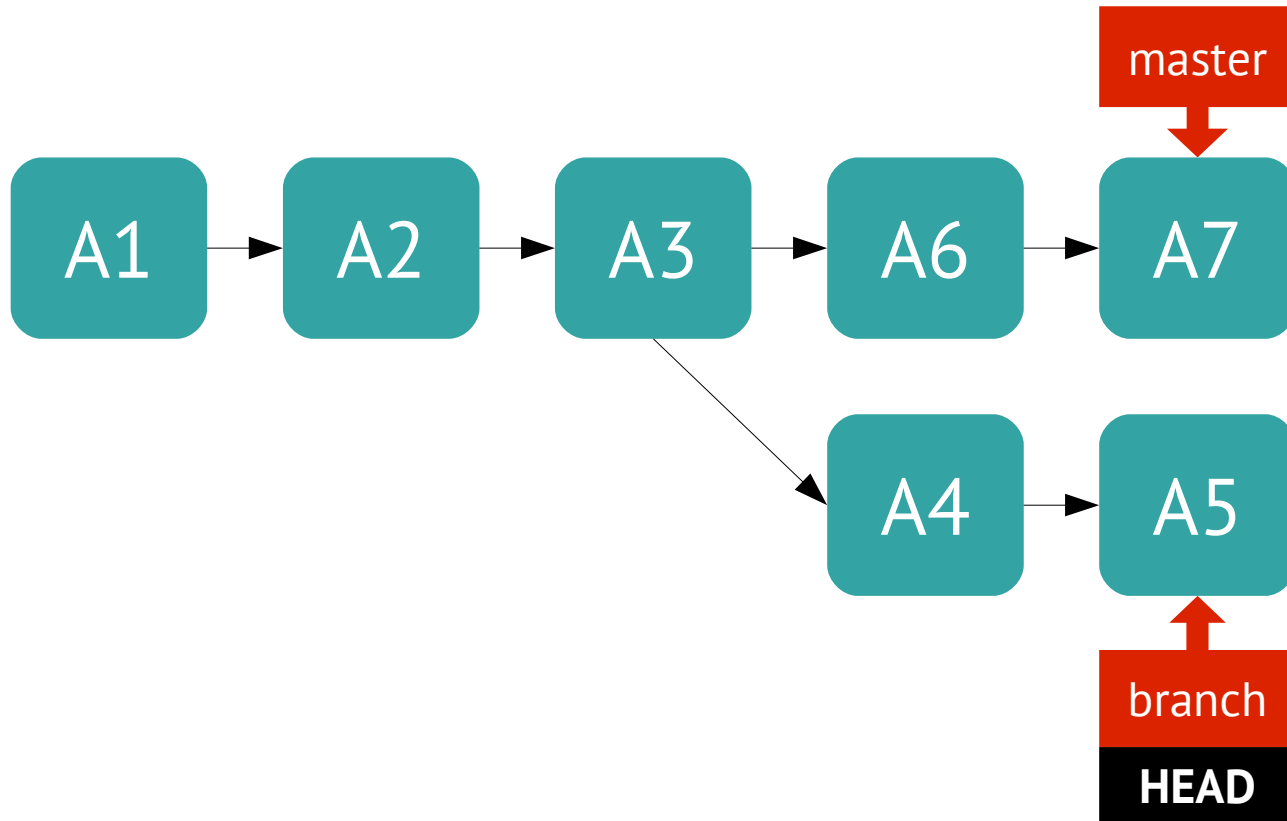
GIT workflow

Rebasing

Never rebase a shared branch

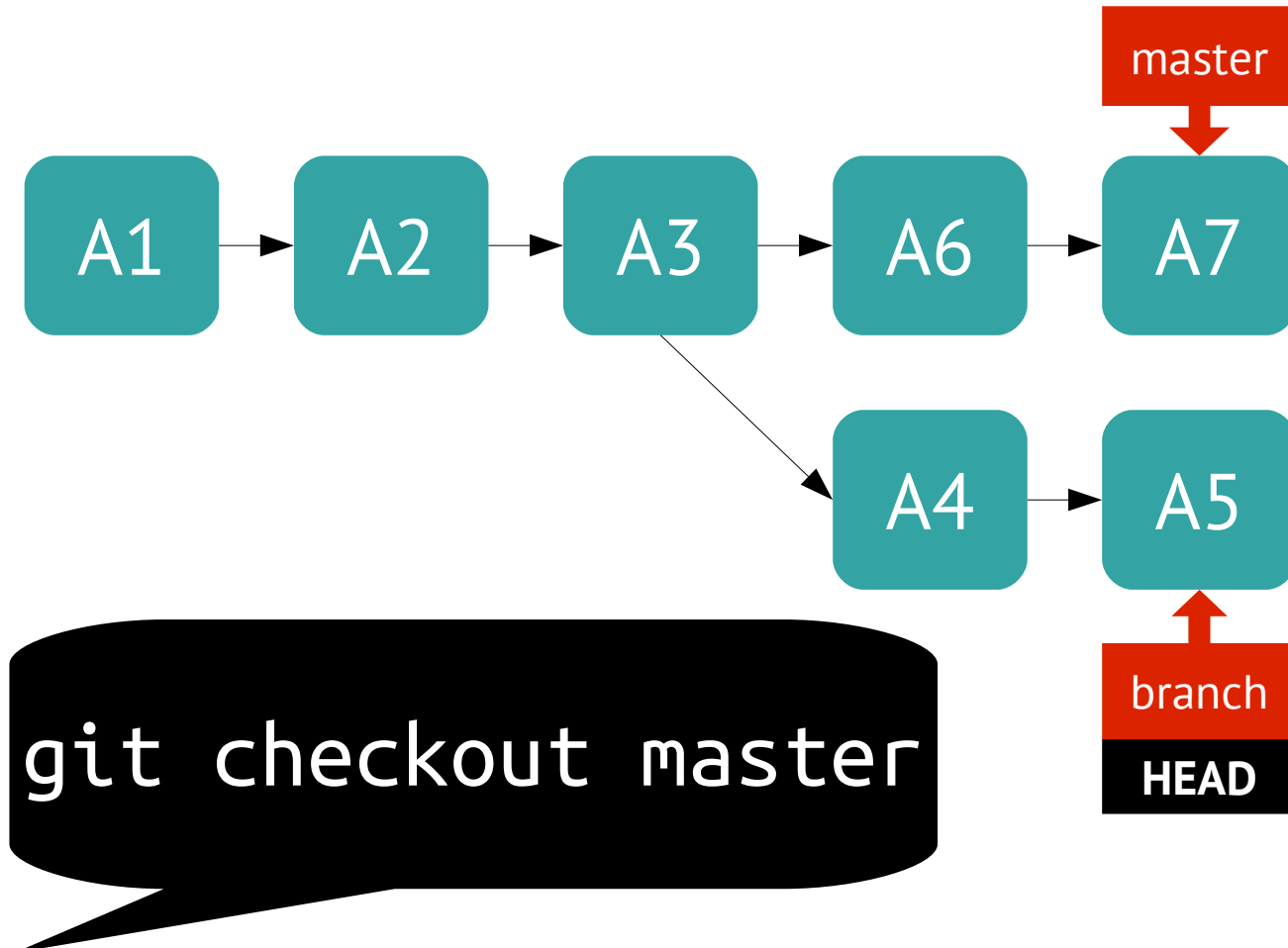
GIT workflow

Rebasing



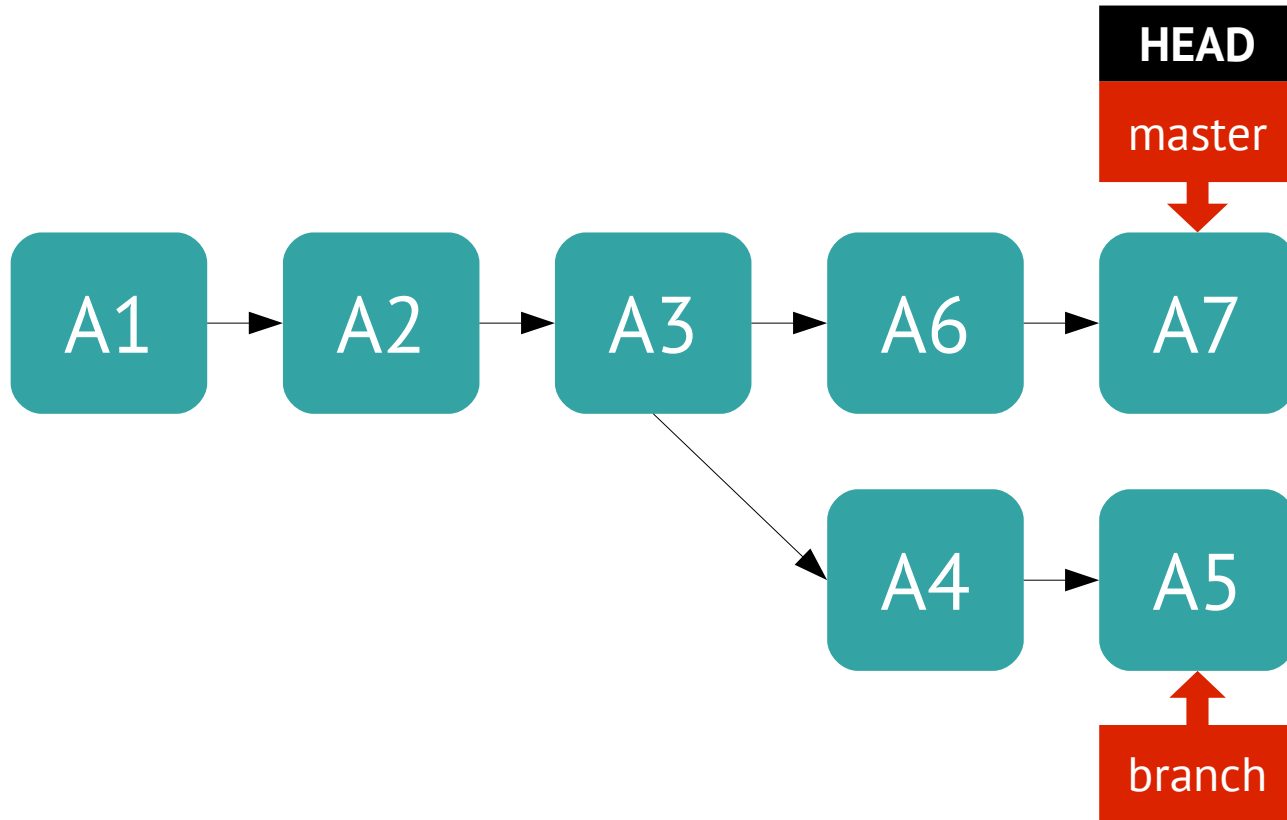
GIT workflow

Rebasing



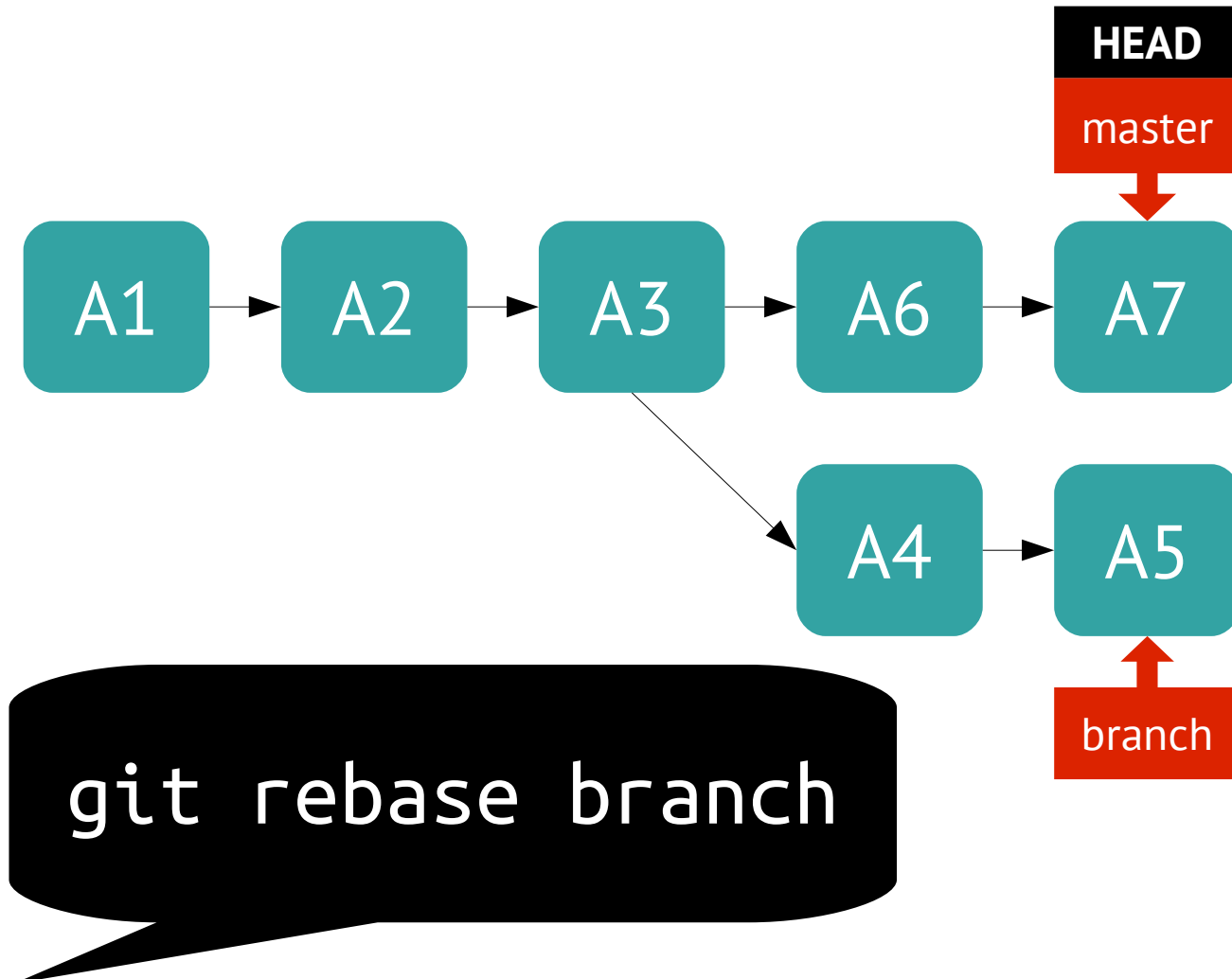
GIT workflow

Rebasing



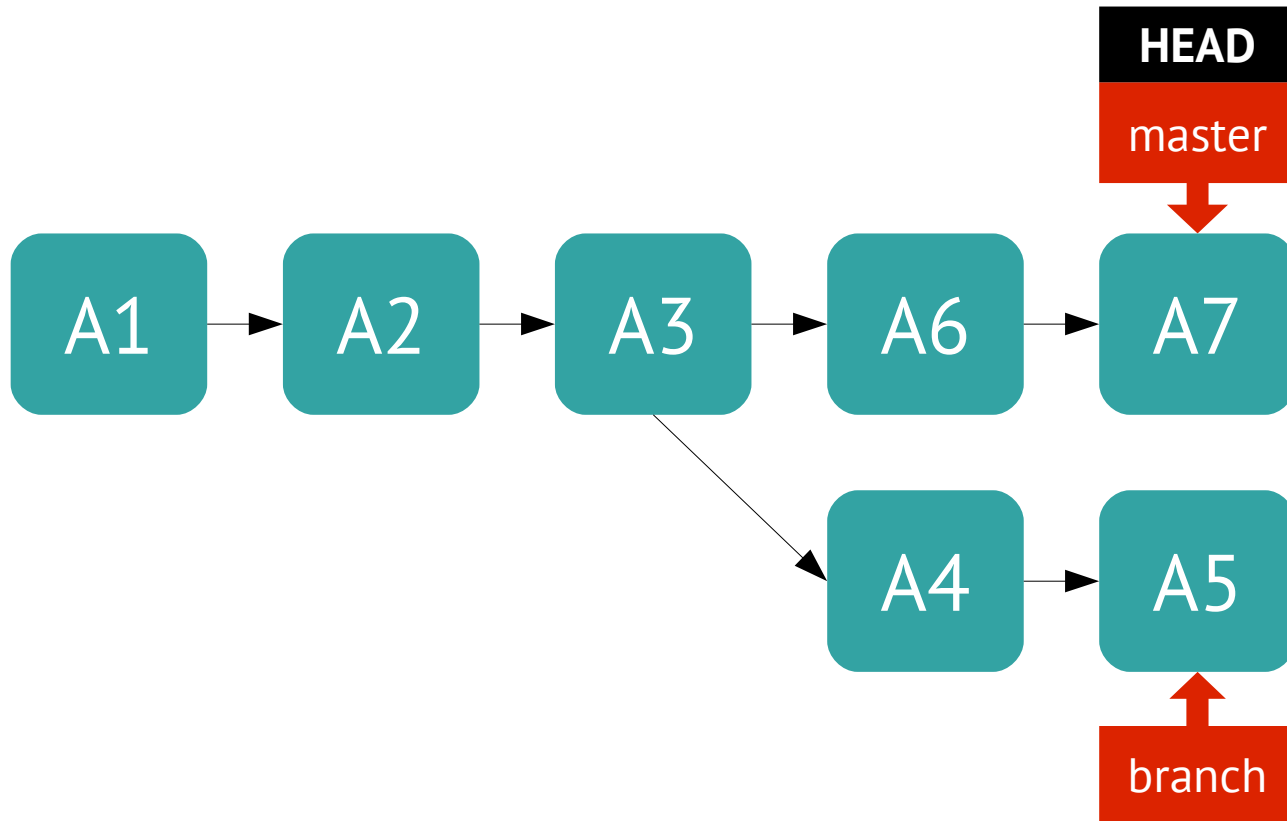
GIT workflow

Rebasing



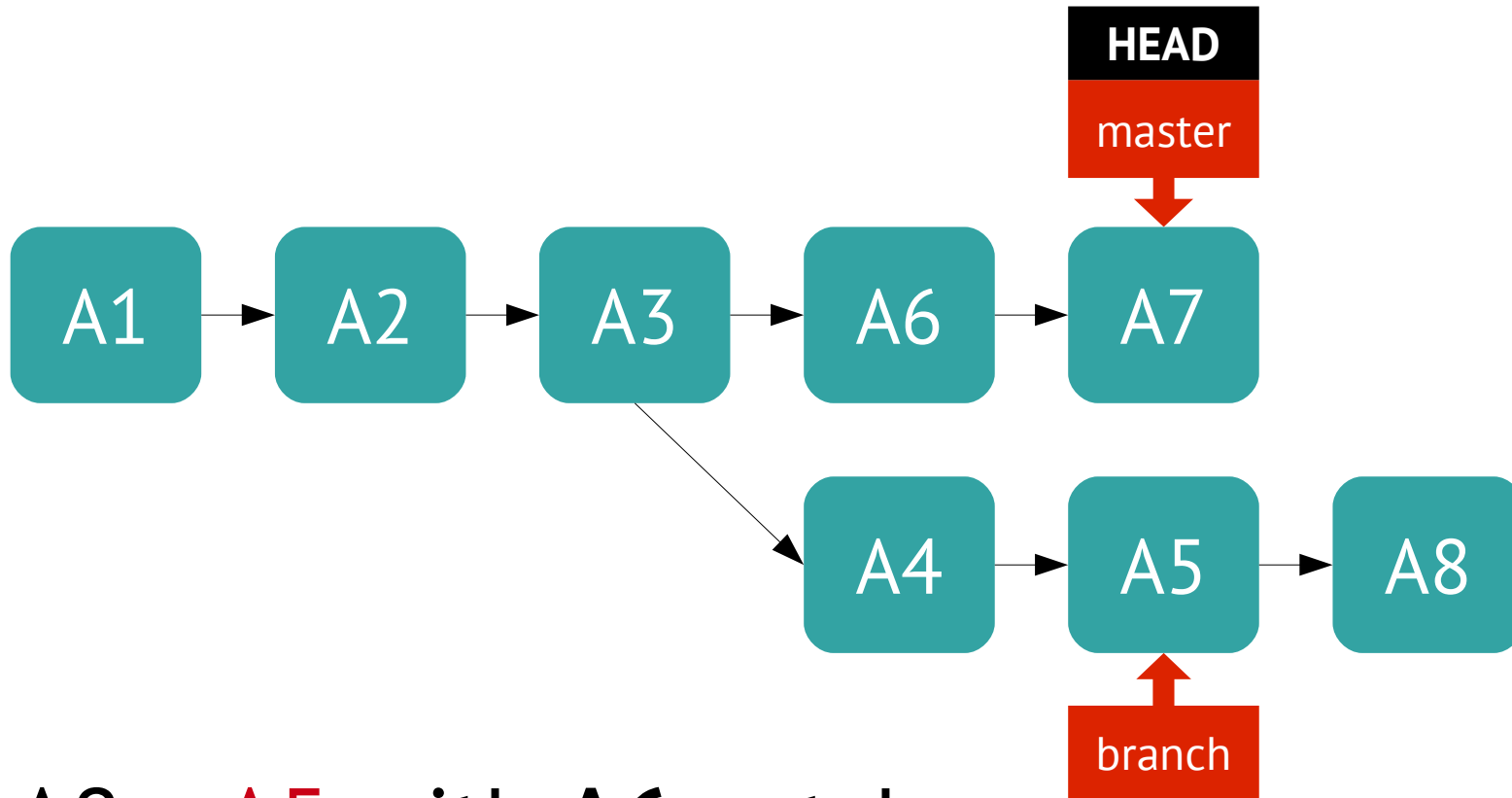
GIT workflow

Rebasing



GIT workflow

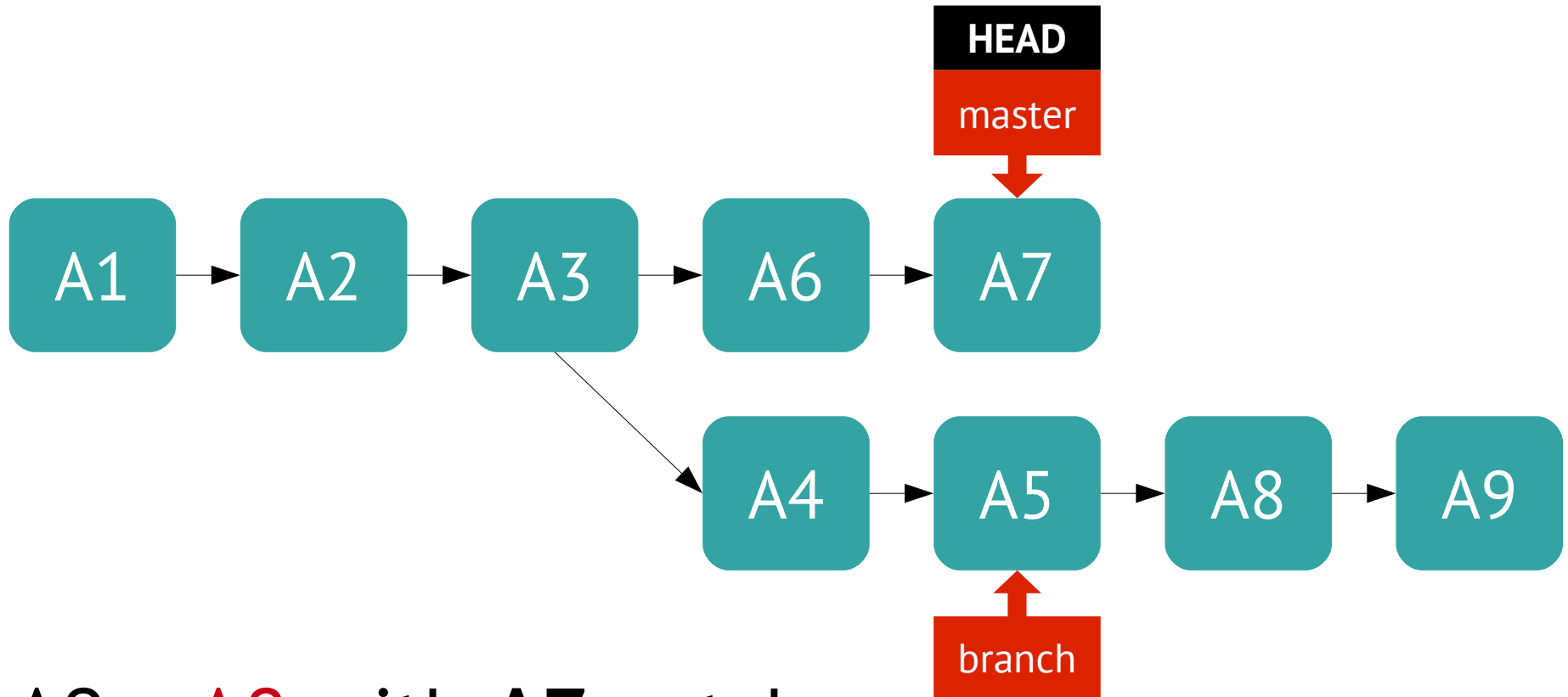
Rebasing



A8 = **A5** with **A6** patch

GIT workflow

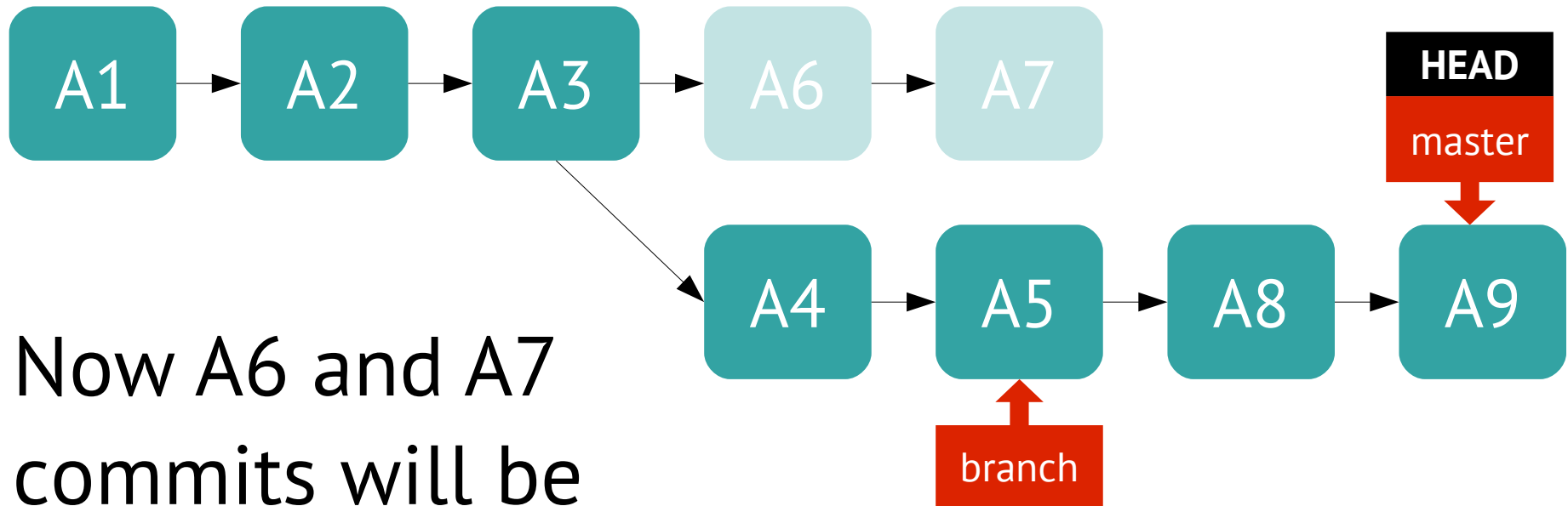
Rebasing



A9 = **A8** with **A7** patch

GIT workflow

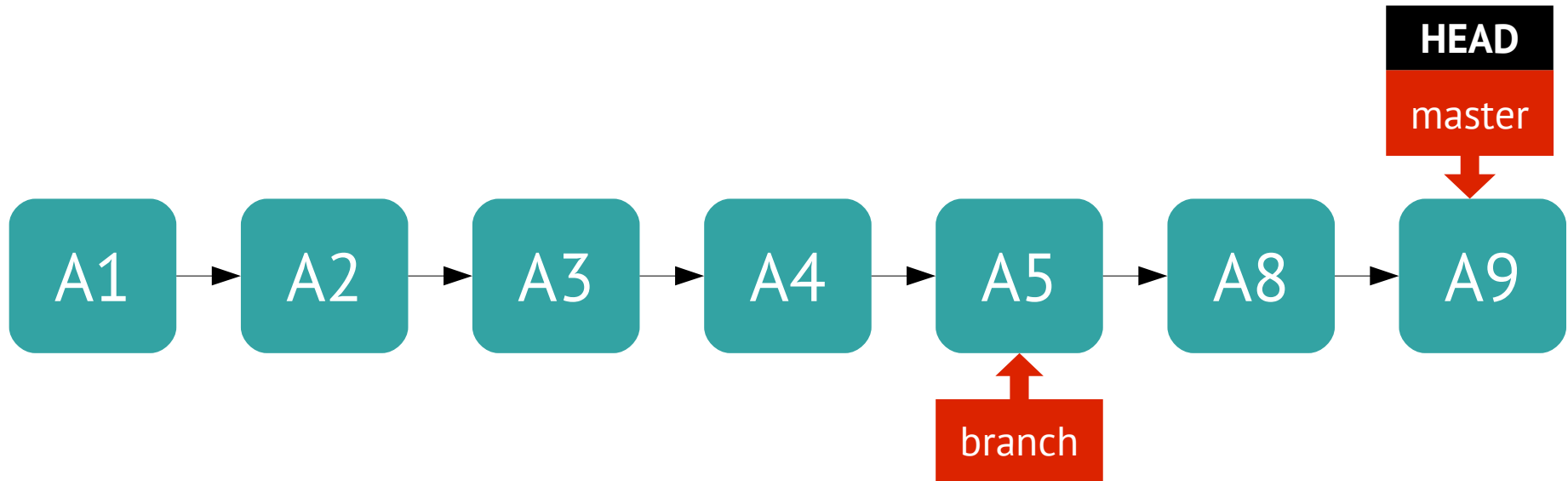
Rebasing



Now A6 and A7
commits will be
deleted. Forever.

GIT workflow

Rebasing



You have broken common branch. If you push it somewhere, it will break other's ones.

GIT workflow

Rebasing

**Never rebase
a shared branch,
you idiot!**

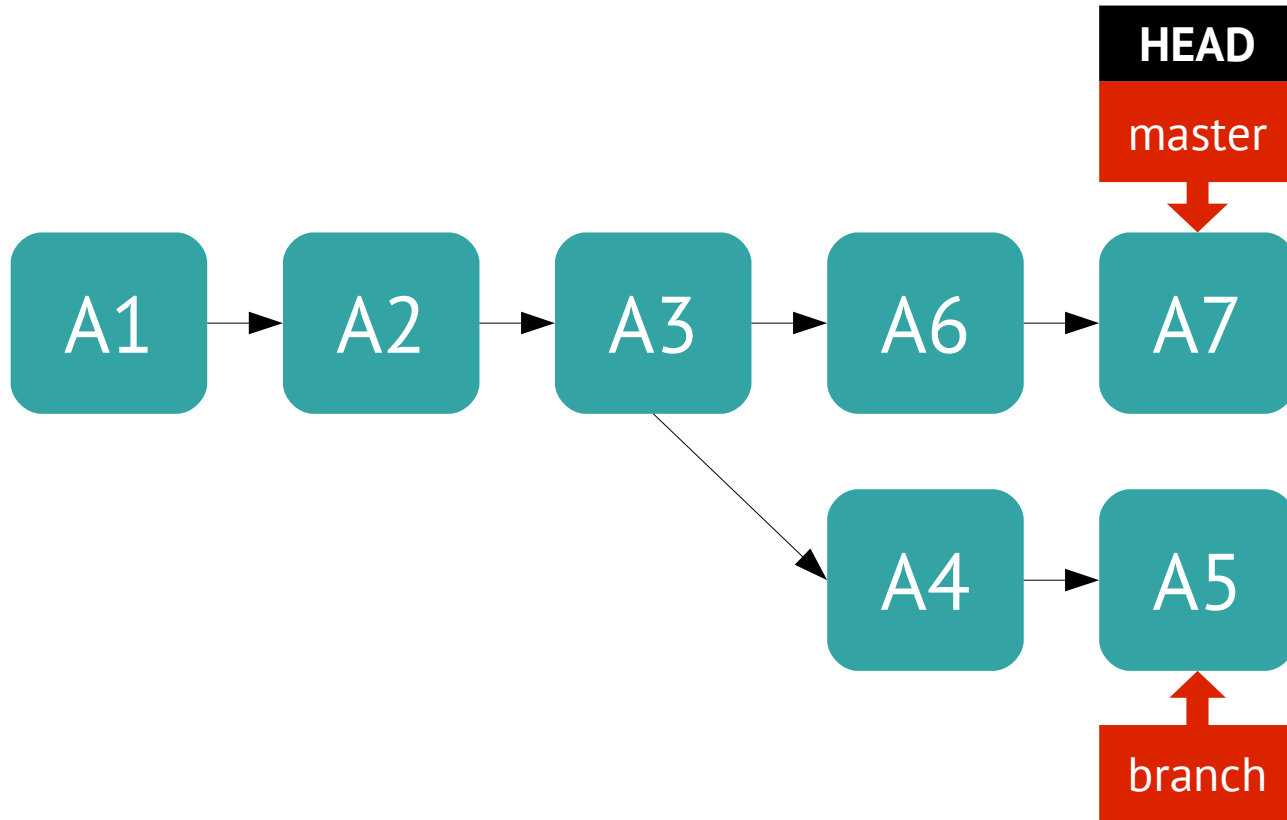
GIT workflow

- ~~Create your branch~~
- ~~Edit files~~
- ~~Stage your changes~~
- ~~Commit the changes~~
- ~~Rebase your branch from master~~
- **Merge to master**
- Push to central repository

Merges

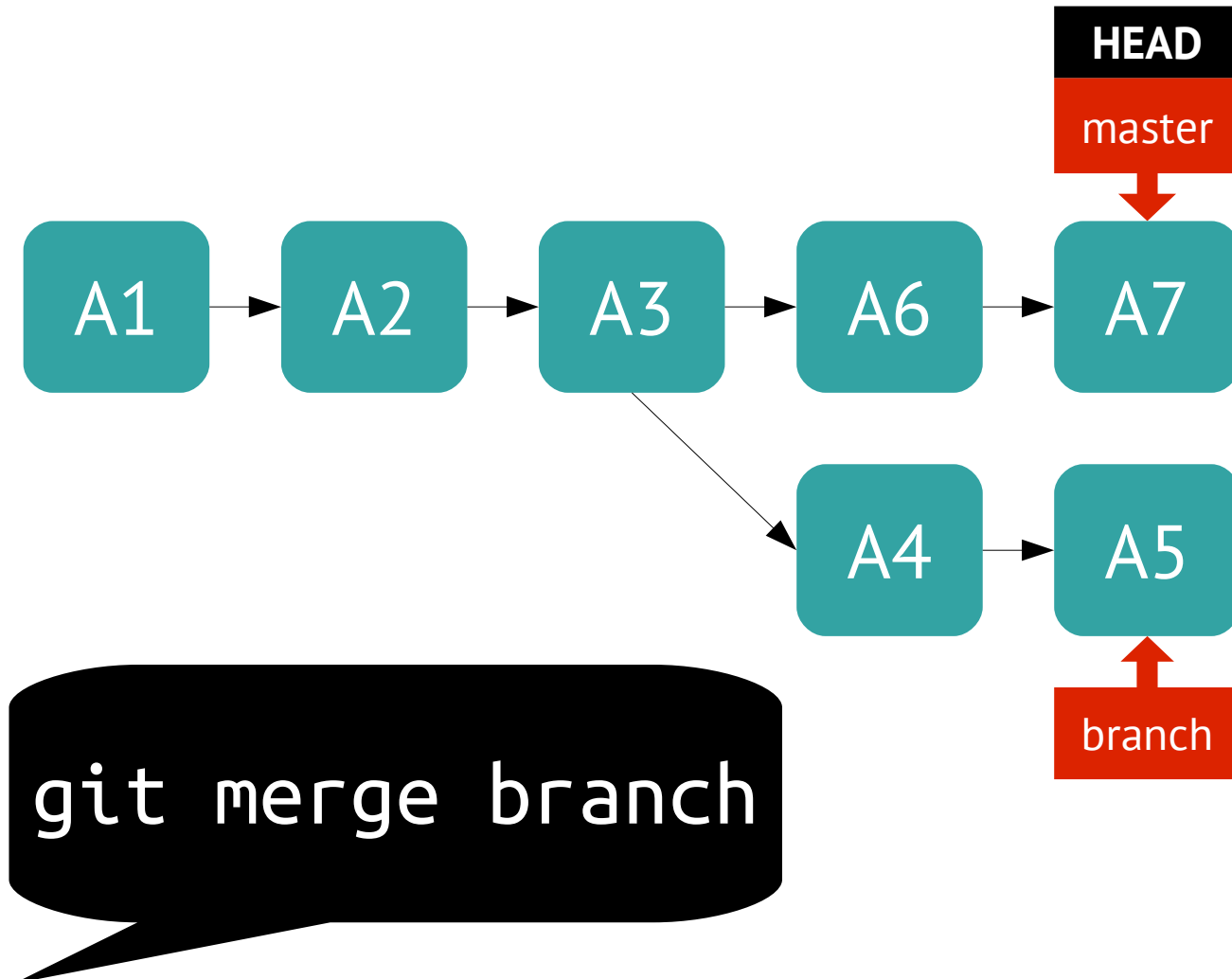
GIT workflow

Merges



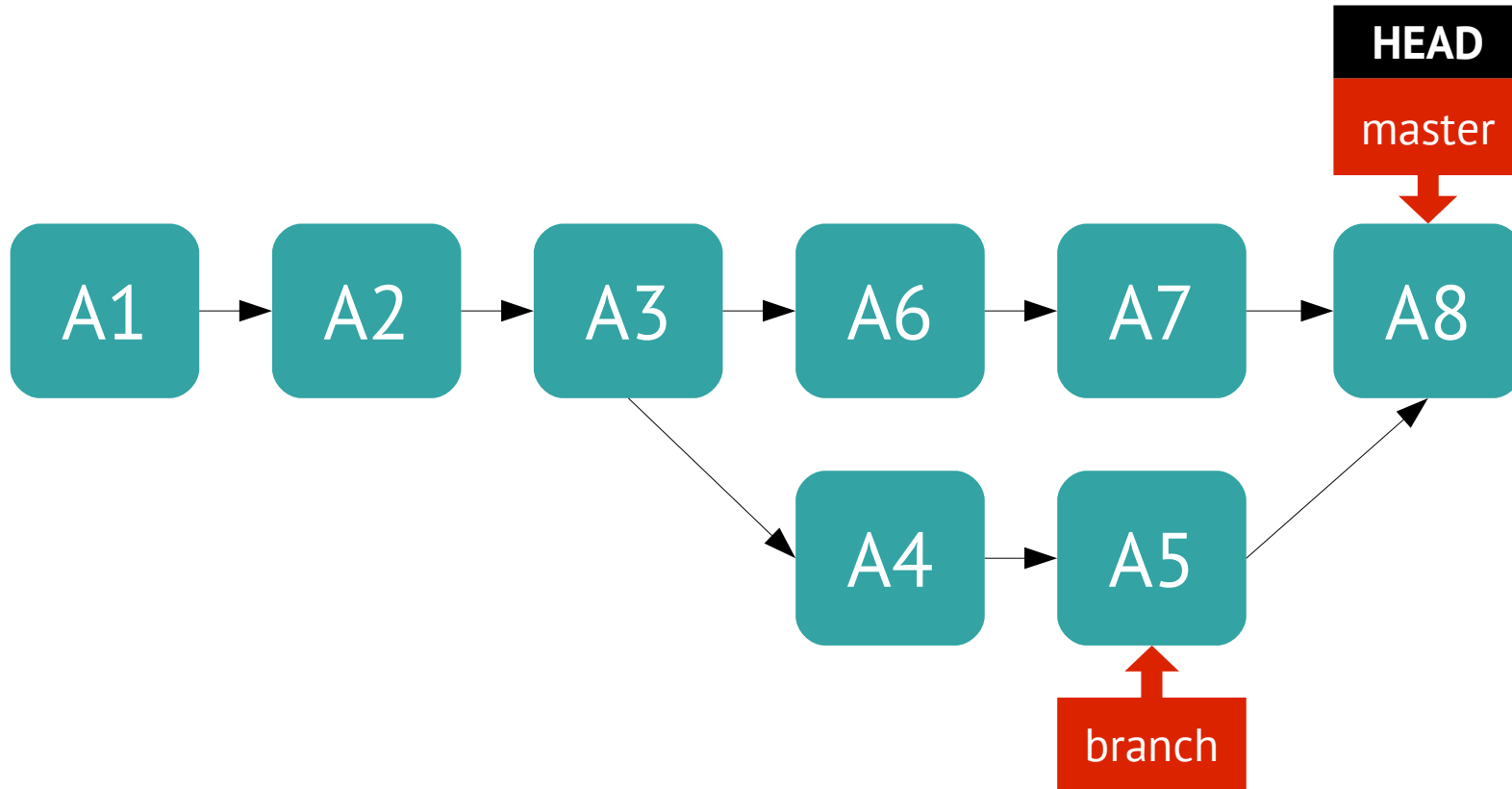
GIT workflow

Merges



GIT workflow

Merges



Have you really thought
that GIT have
complicated merges?

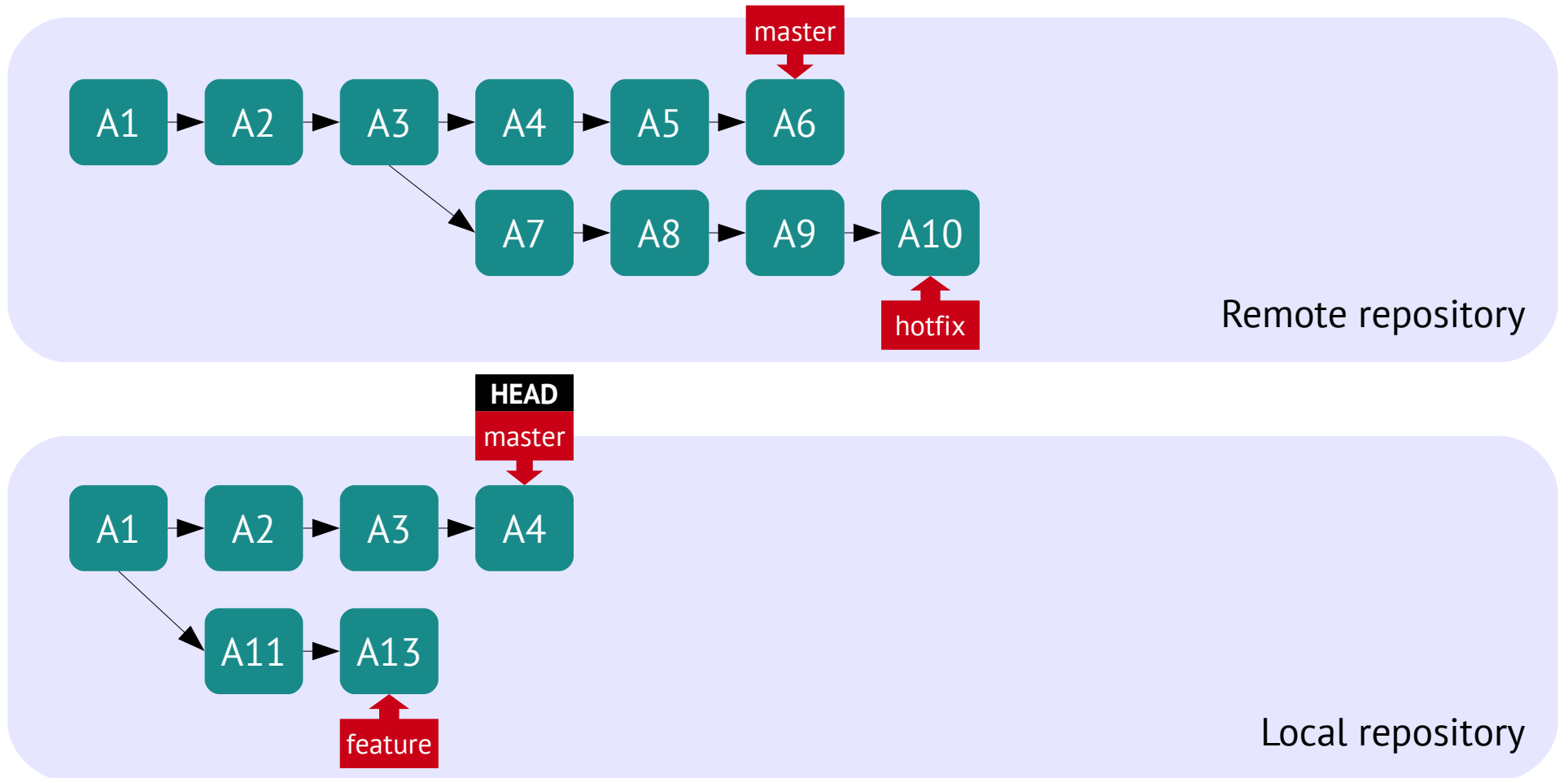
GIT workflow

- ~~Create your branch~~
- ~~Edit files~~
- ~~Stage your changes~~
- ~~Commit the changes~~
- ~~Rebase your branch from master~~
- ~~Merge to master~~
- **Push to central repository**

Push and pull

GIT workflow

Push and pull



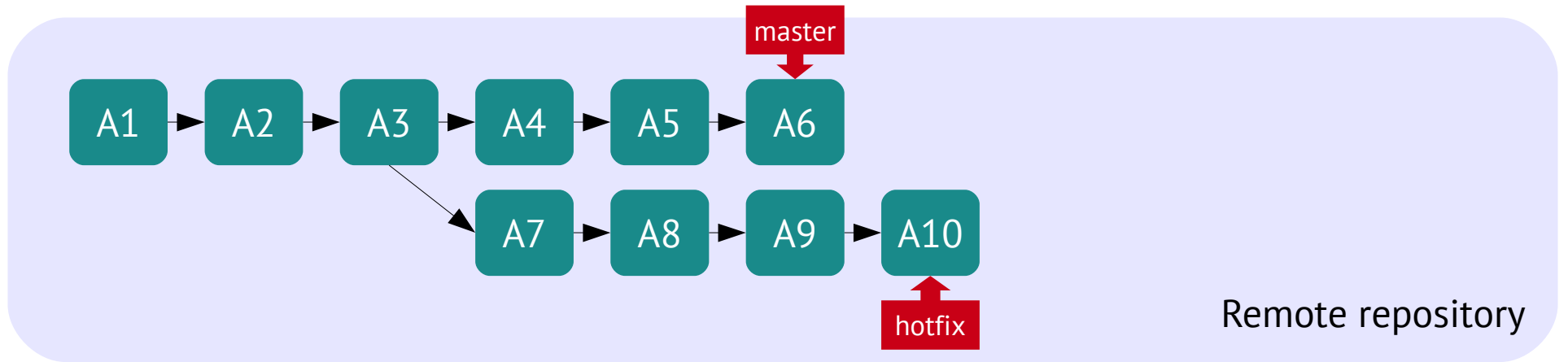
GIT workflow

Push and pull

```
git fetch
```

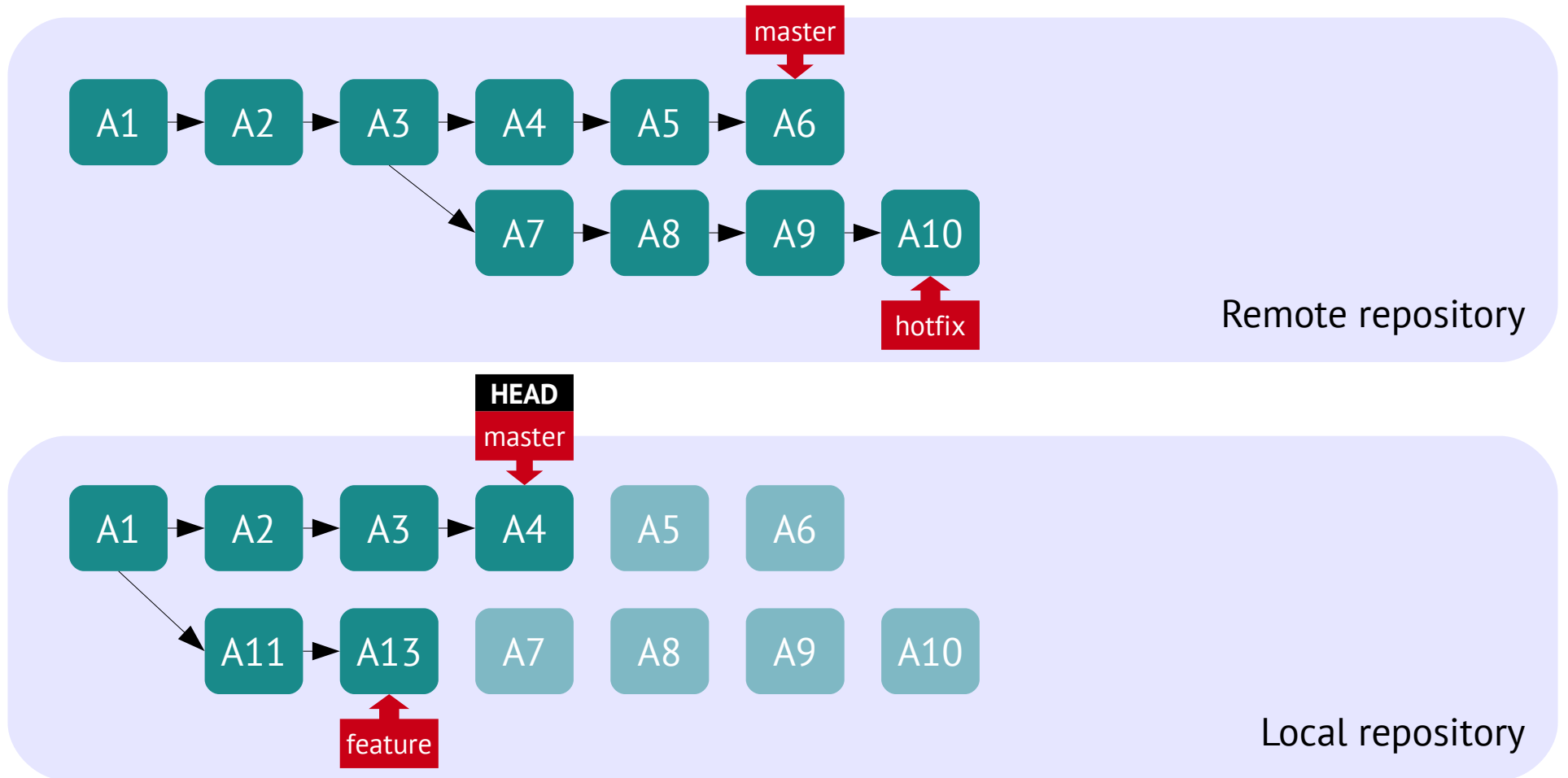
GIT workflow

Push and pull



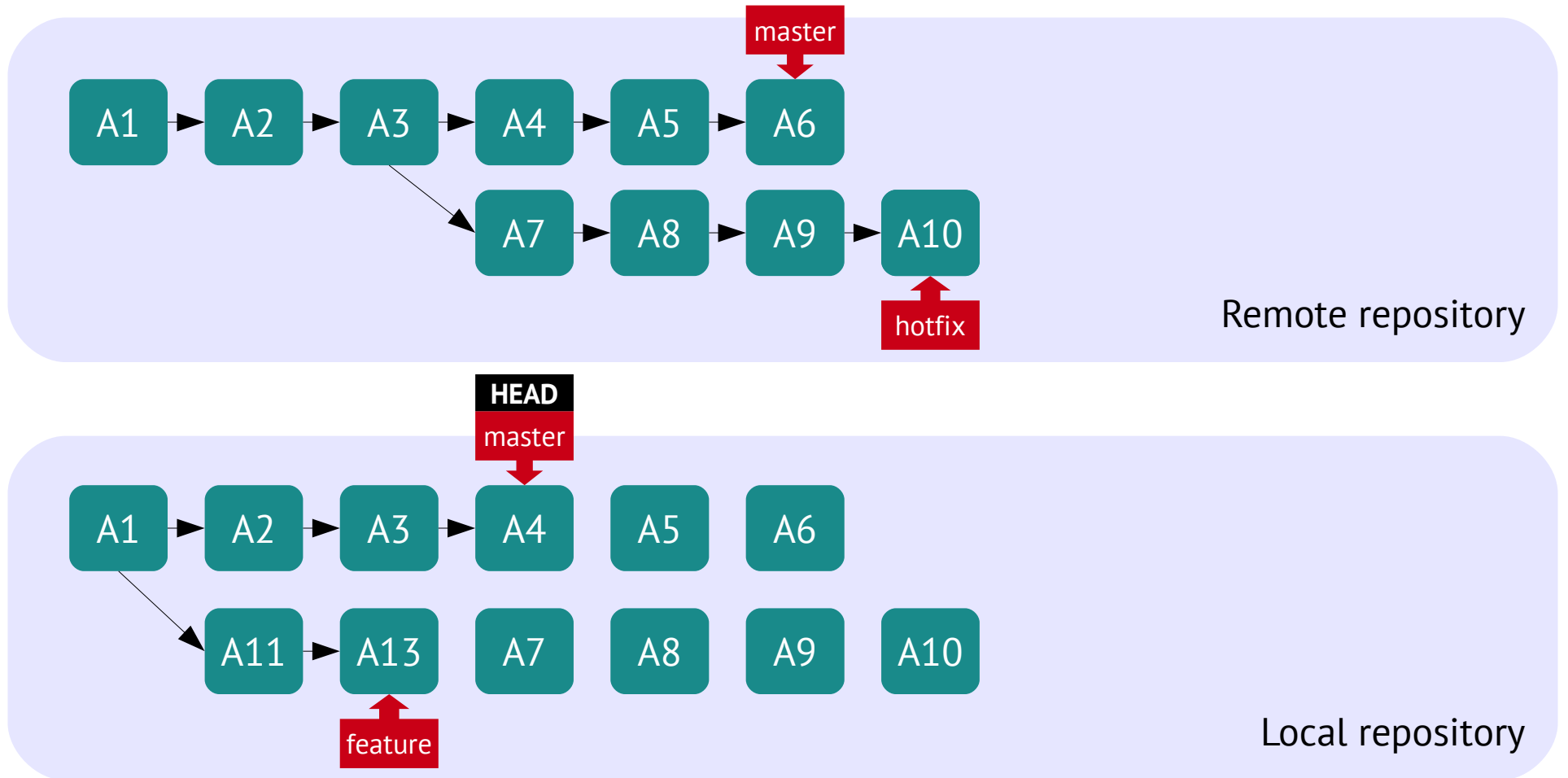
GIT workflow

Push and pull



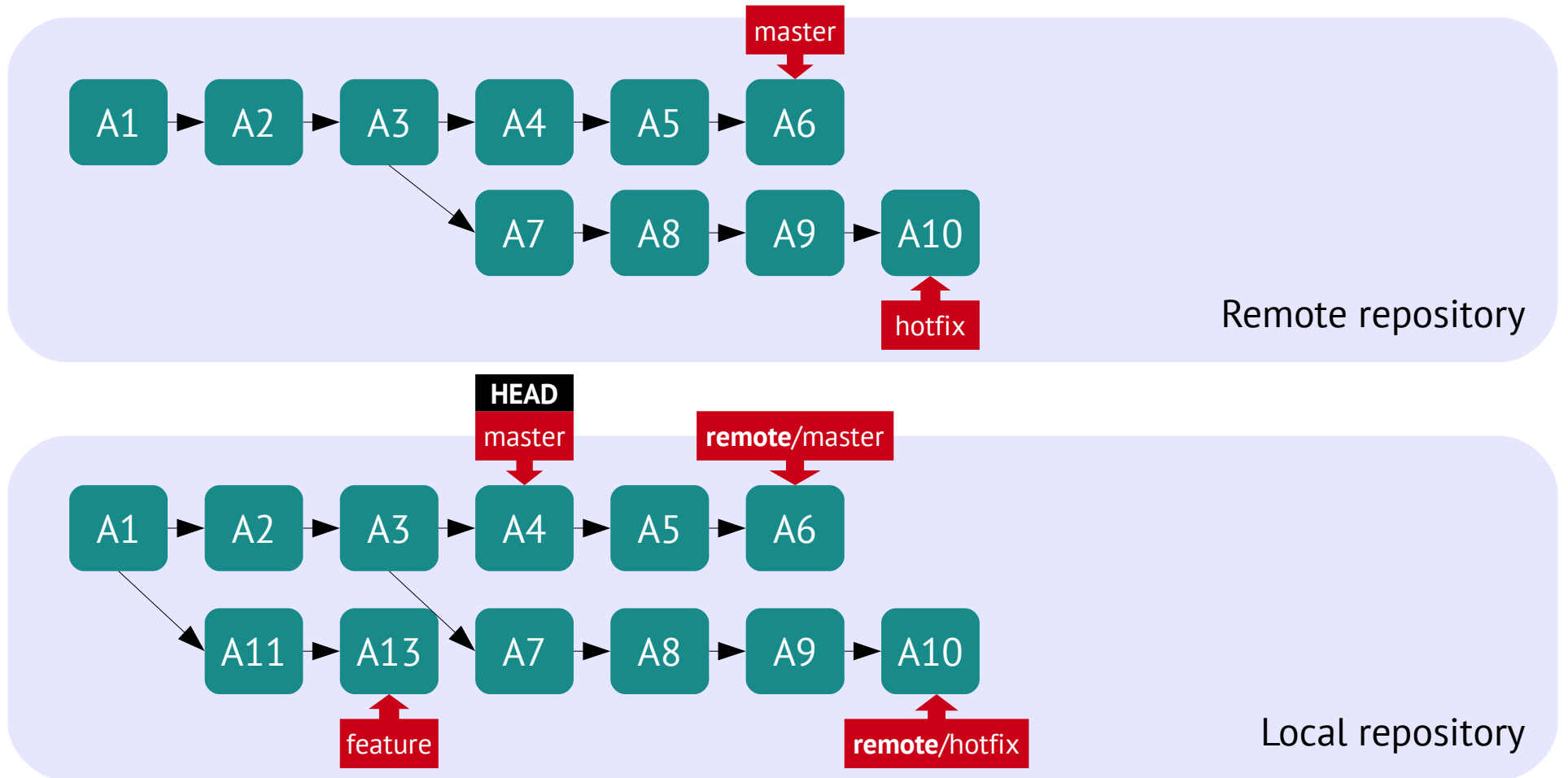
GIT workflow

Push and pull



GIT workflow

Push and pull



GIT workflow

Push and pull

```
git pull
```

GIT workflow

Push and pull

$\text{pull} = \text{fetch} + \text{merge}$

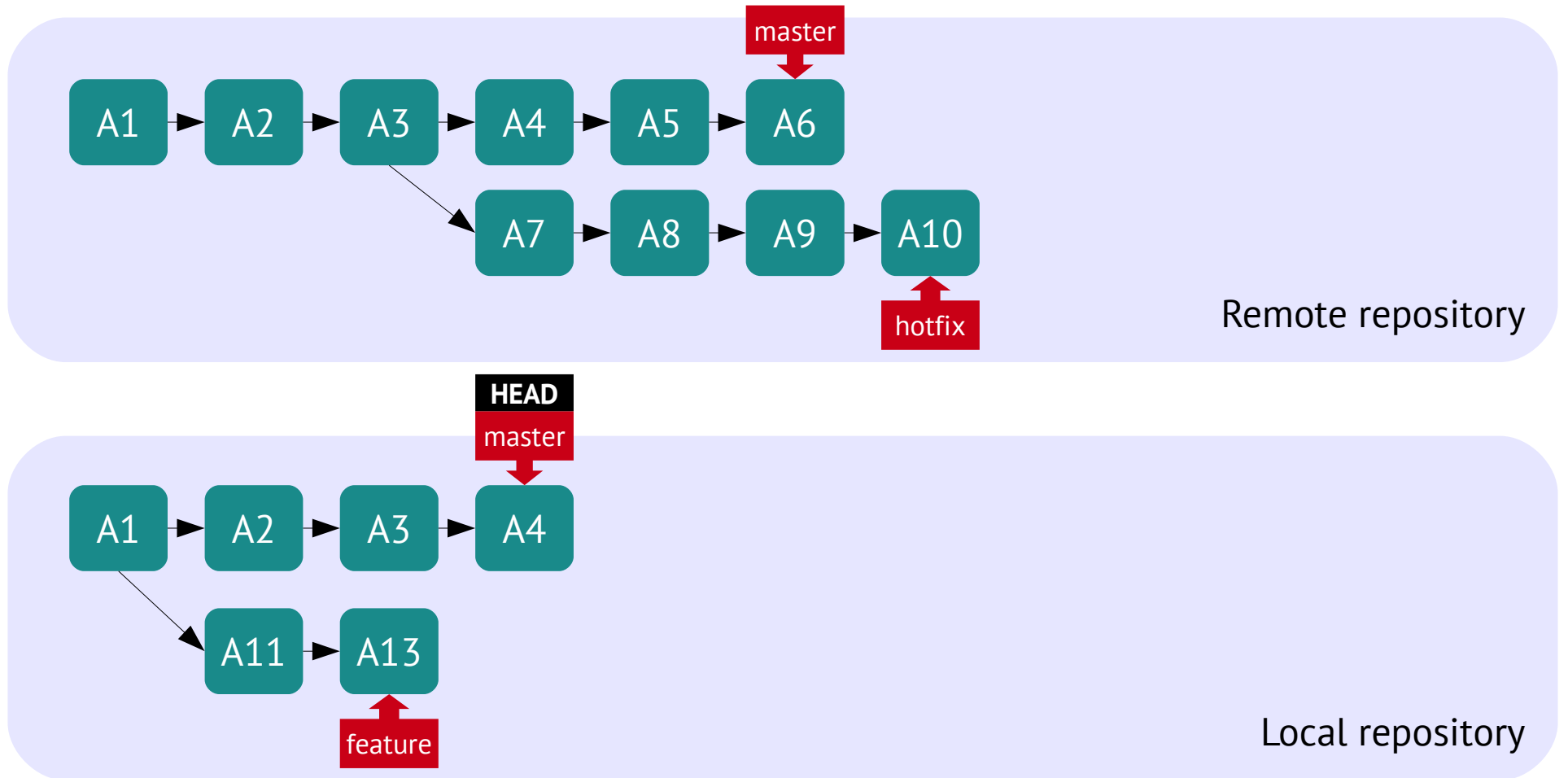
GIT workflow

Push and pull

```
git push
```

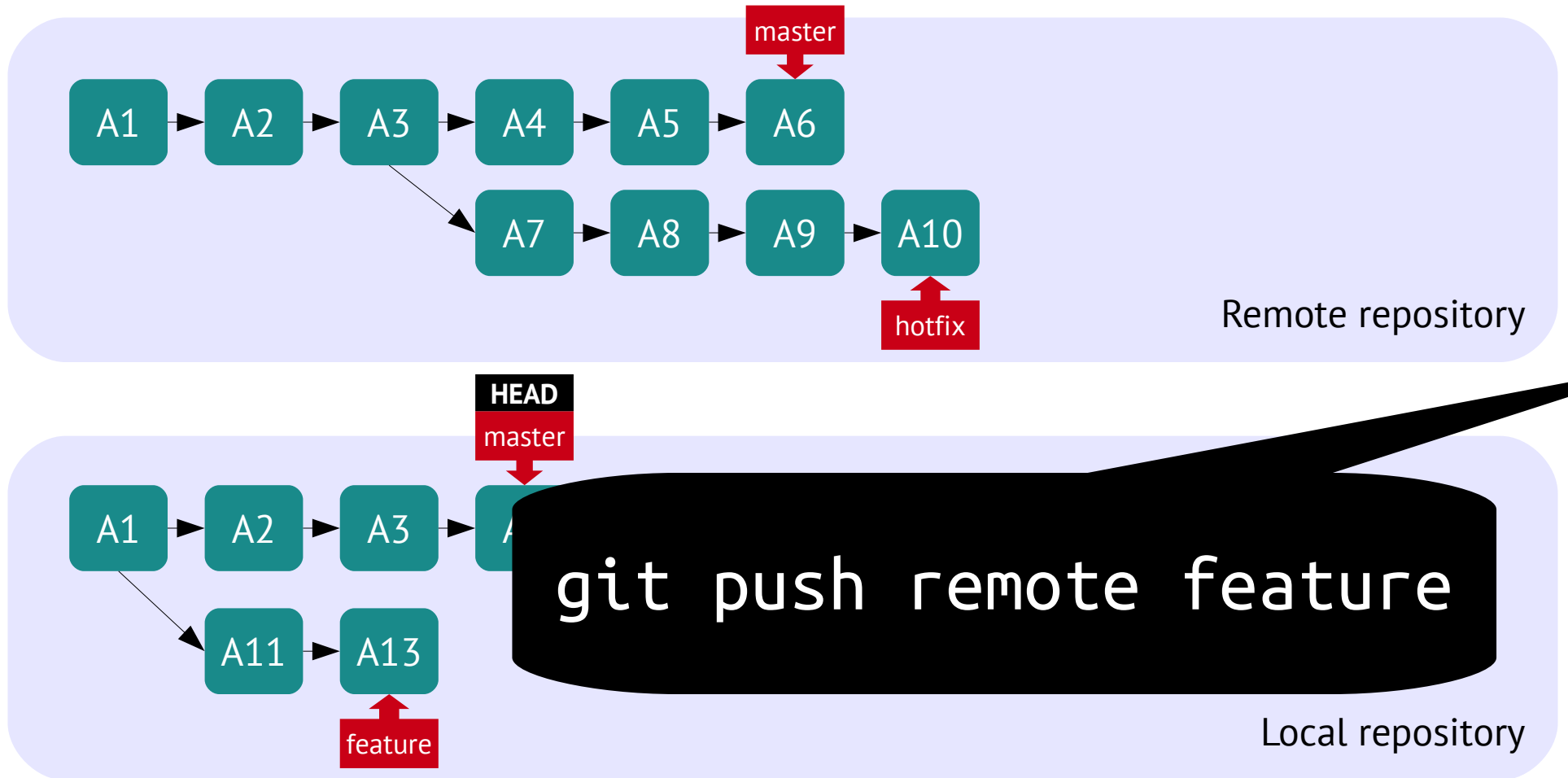
GIT workflow

Push and pull



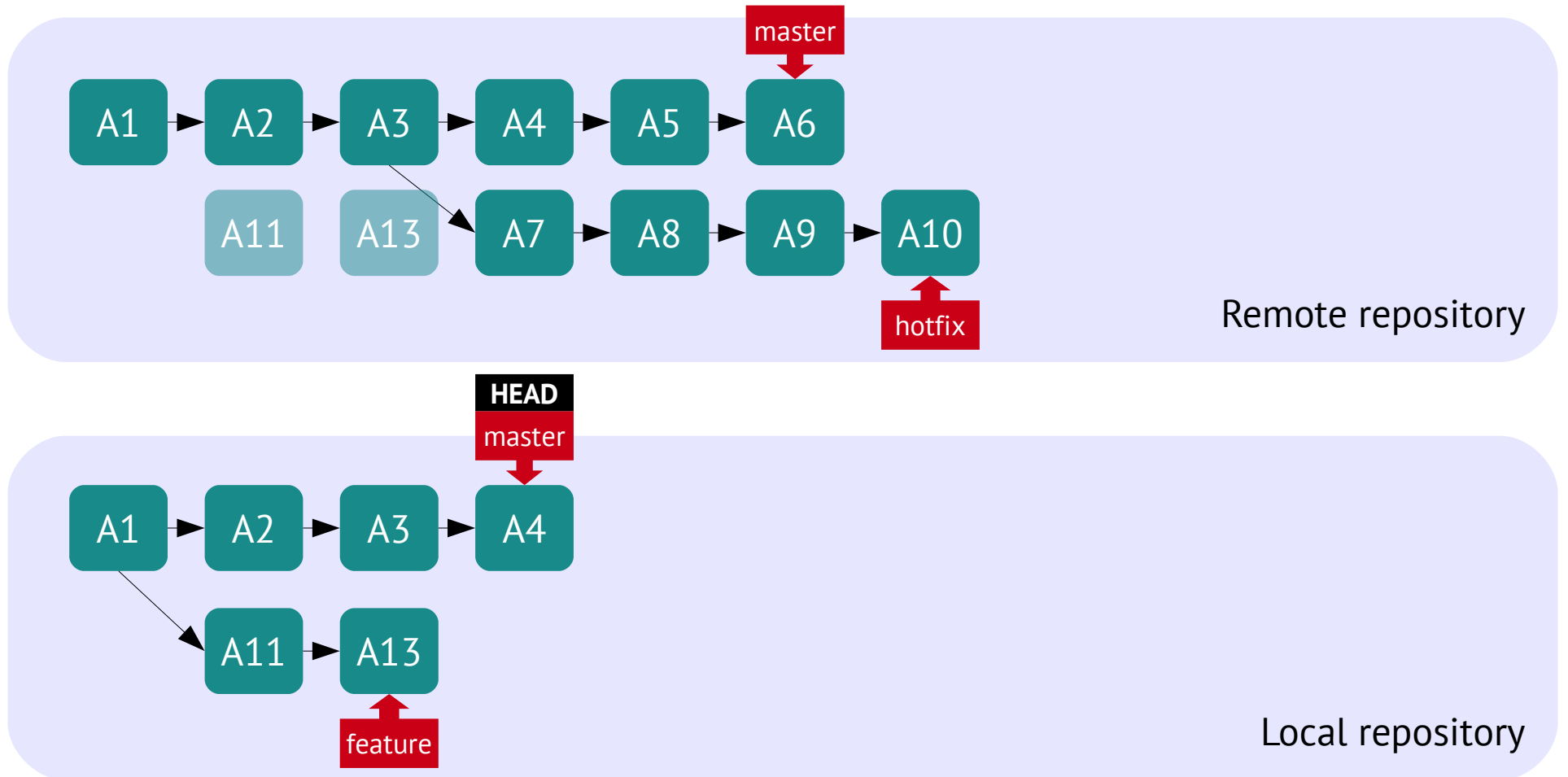
GIT workflow

Push and pull



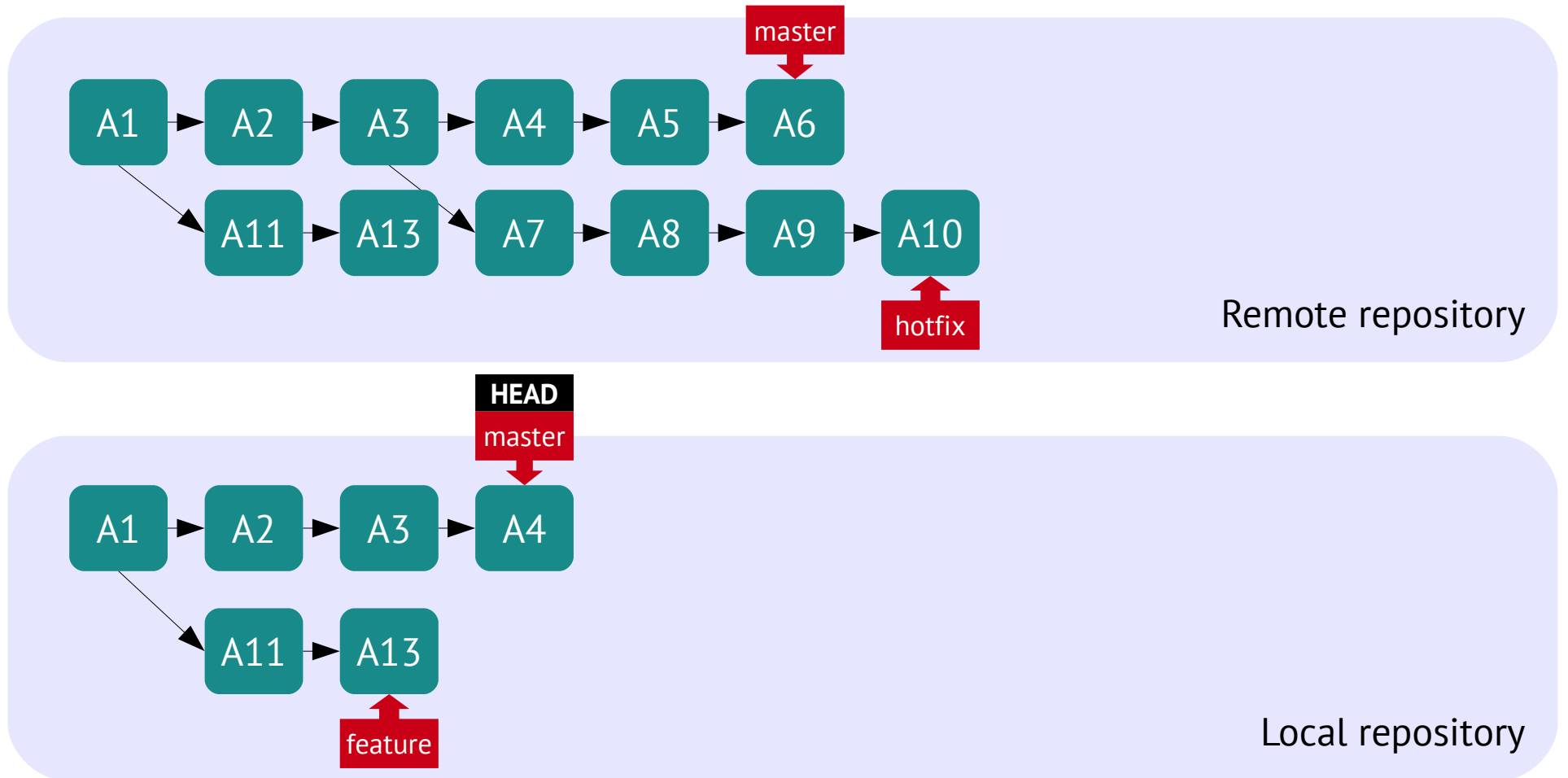
GIT workflow

Push and pull



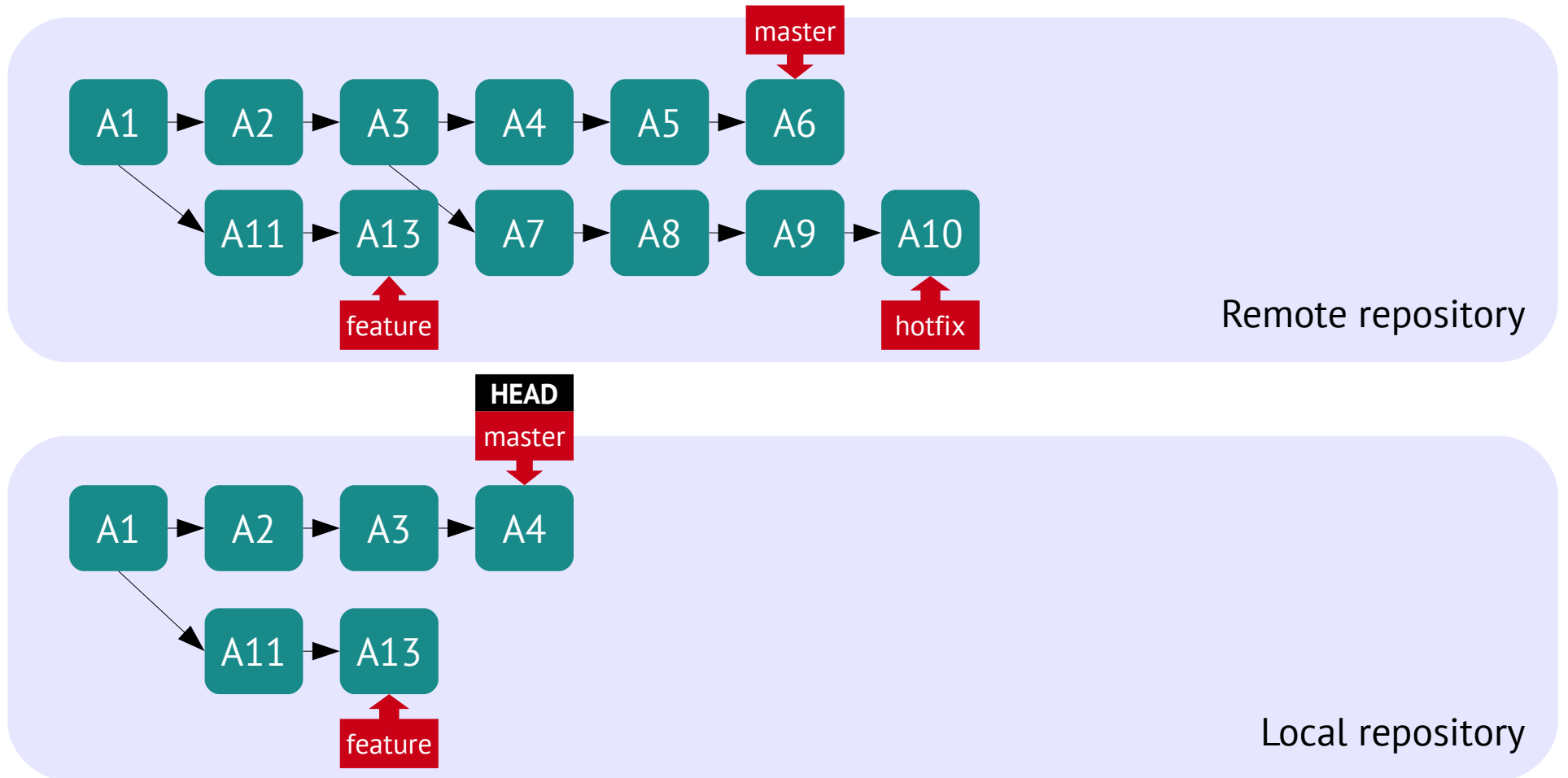
GIT workflow

Push and pull



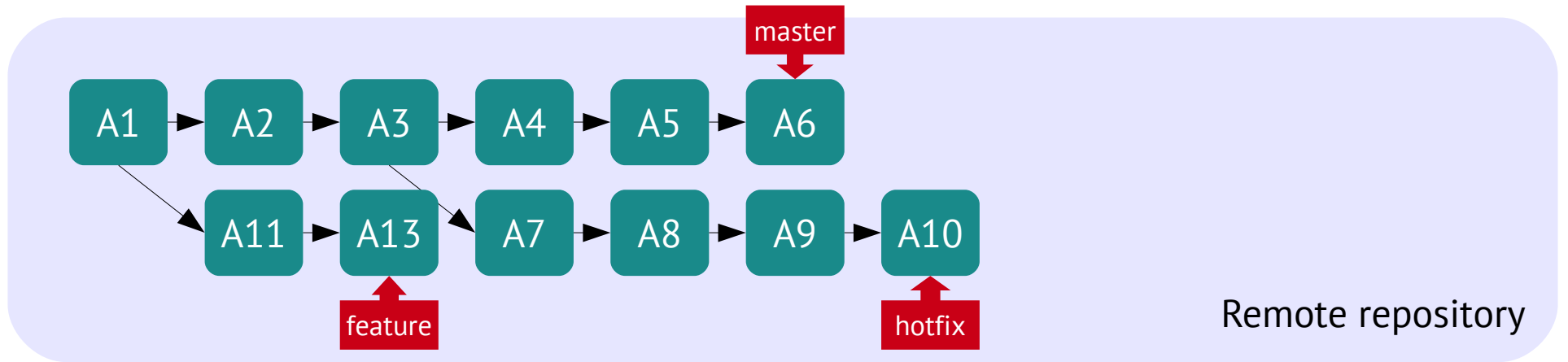
GIT workflow

Push and pull



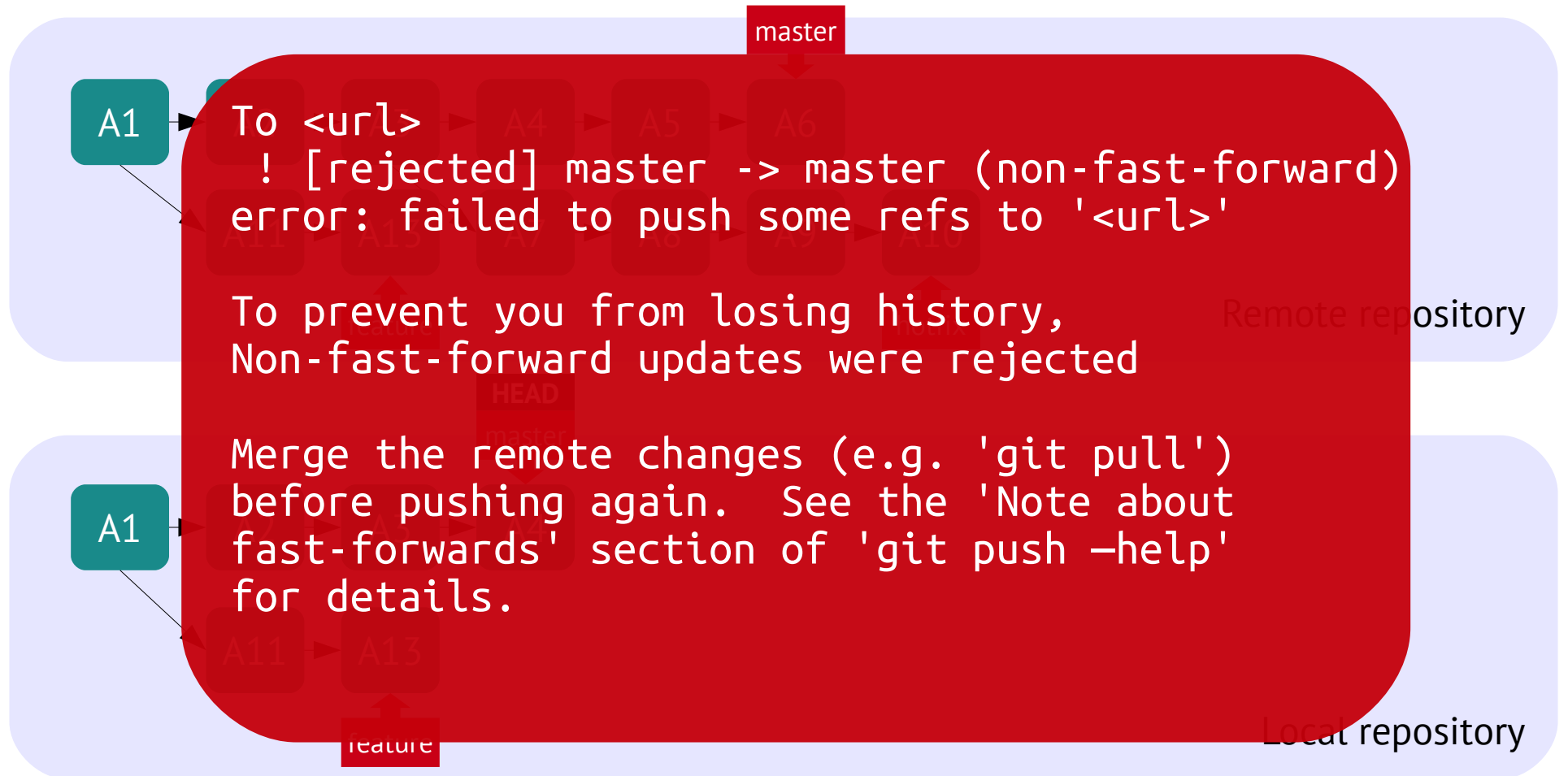
GIT workflow

Push and pull



GIT workflow

Push and pull



GIT workflow

Push and pull

master

A1

To <url>
! [rejected] master -> master (non-fast-forward)
error: failed to push some refs to '<url>'

**To prevent you from losing history,
Non-fast-forward updates were rejected**

A1

Merge the remote changes (e.g. 'git pull')
before pushing again. See the 'Note about
fast-forwards' section of 'git push -help'
for details.

feature

Local repository

GIT workflow

Push and pull

`git push` is about
fast-forward merges

GIT workflow

Push and pull

Always pull before push!

GIT workflow

Push and pull

Always **pull** before **push**!

GIT workflow

- ~~Create your branch~~
- ~~Edit files~~
- ~~Stage your changes~~
- ~~Commit the changes~~
- ~~Rebase your branch from master~~
- ~~Merge to master~~
- ~~Push to central repository~~

There are only 5 slides left.
Cheer up!

Must-have commands

Must-have commands

`git diff`

`git status`

`git clone`

`git merge`

`git add`

`git log`

`git commit`

`git push`

`git branch`

`git pull`

`git commit`

`git fetch`

12
commands

Credits

Credits

Git Community Book

ProGit

Help on GitHub

Linus Torvalds on GIT

StackOverflow GIT tag

GIT Magic

GIT Immersion

GITCasts

GIT Wizardry

GIT Manual

GIT official site

GIT Reference

that's it!