Git lab 3

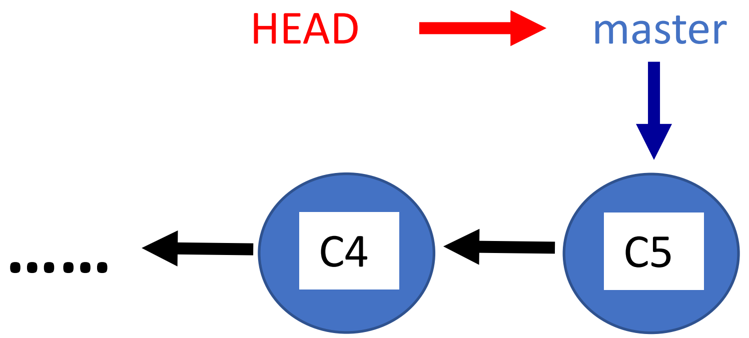
4 Creating new branches

Key features of VCS

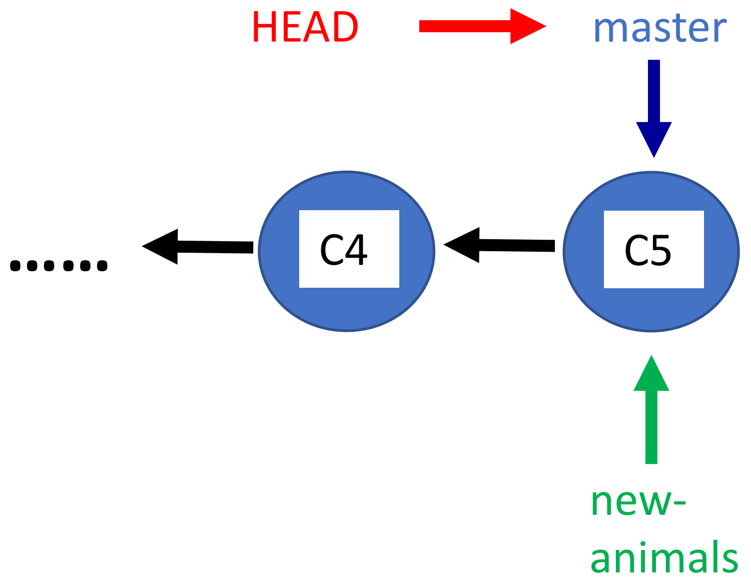
- ❖ Supports code development in parallel
  - Allows devs to collaborate simultaneously on the same code base to implement different features or perform a bug fix
- ❖ Ensures that these work streams are kept isolated and independent of each other
  - When completed, these work streams can then be integrated back into the main code base
  - Any conflicts are resolved appropriately without accidental overwrites

Branch features

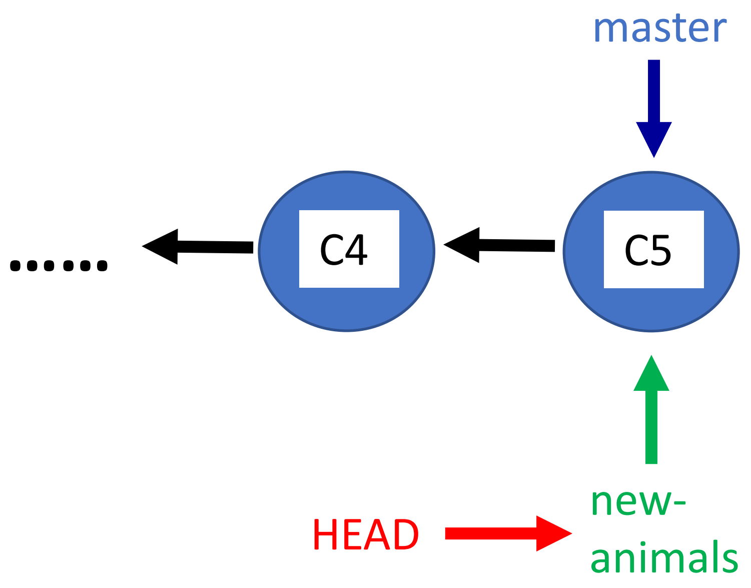
- ❖ Branches provide context isolation
  - Allows devs to work independently and simultaneously on a common code base to implement new features or create bug fix
- ❖ The active branch is pointed to by HEAD
  - New commits are only added to the active branch, other branches remain unchanged
  - We can switch between branches by moving HEAD to point to a different branch


HEAD
master
......
C4
C5

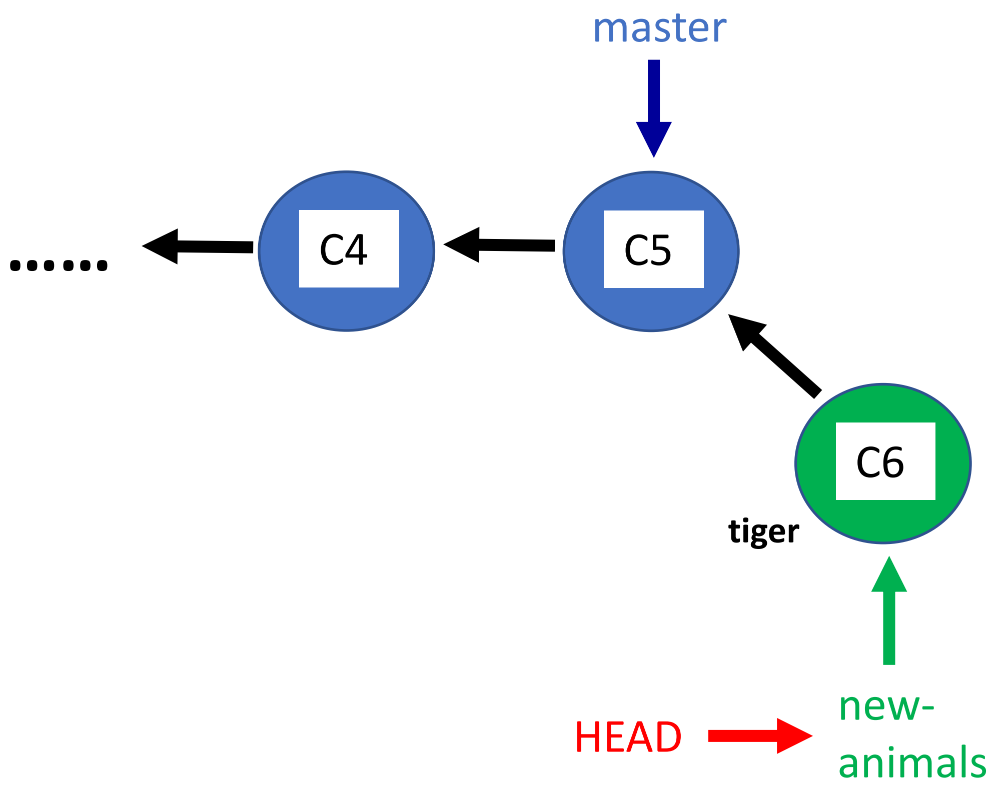
```
git branch new-animals
```



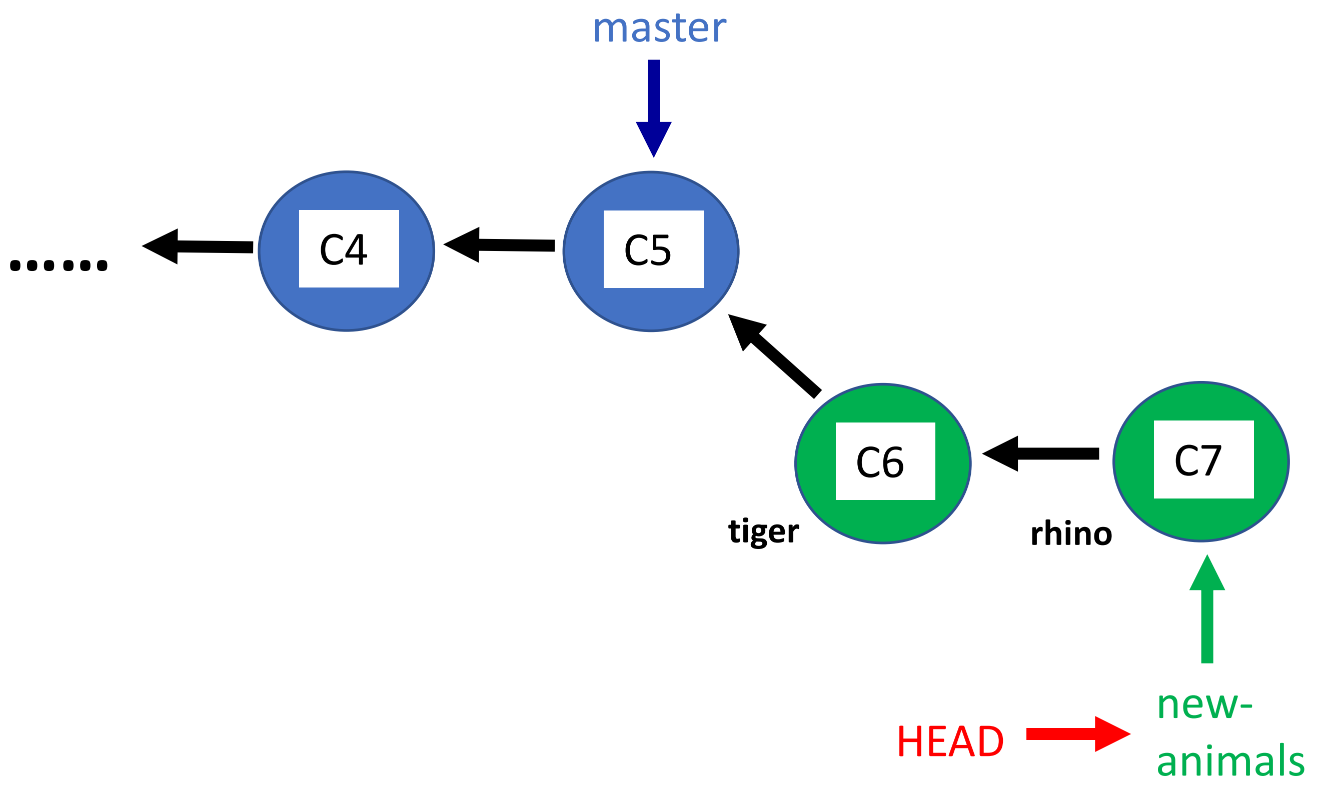
```
git checkout new-animals
```



```
git commit -am "Added a new tiger"
```



```
git commit -am "Added a new rhino"
```



git

Git lab 3

5 Switching between branches

Last In First Out (LIFO) / Stack

git stash

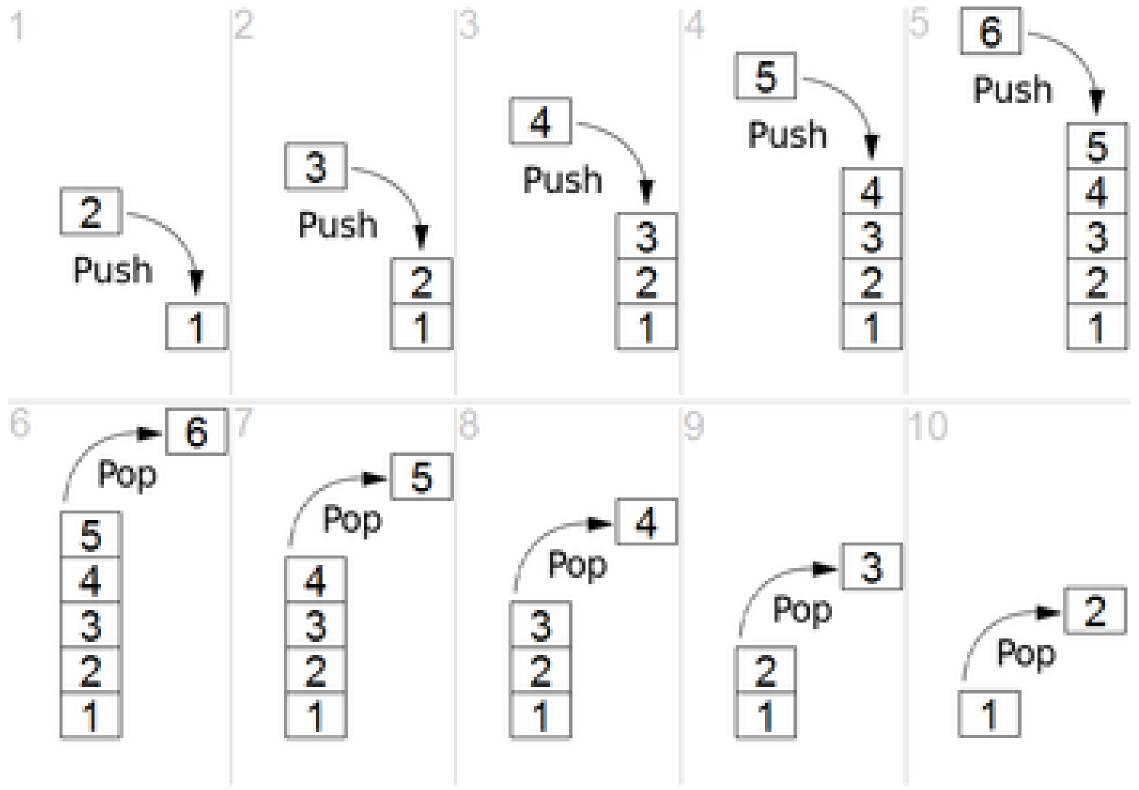
Saves both staged and unstaged changes as an entry in a LIFO stack

git stash pop

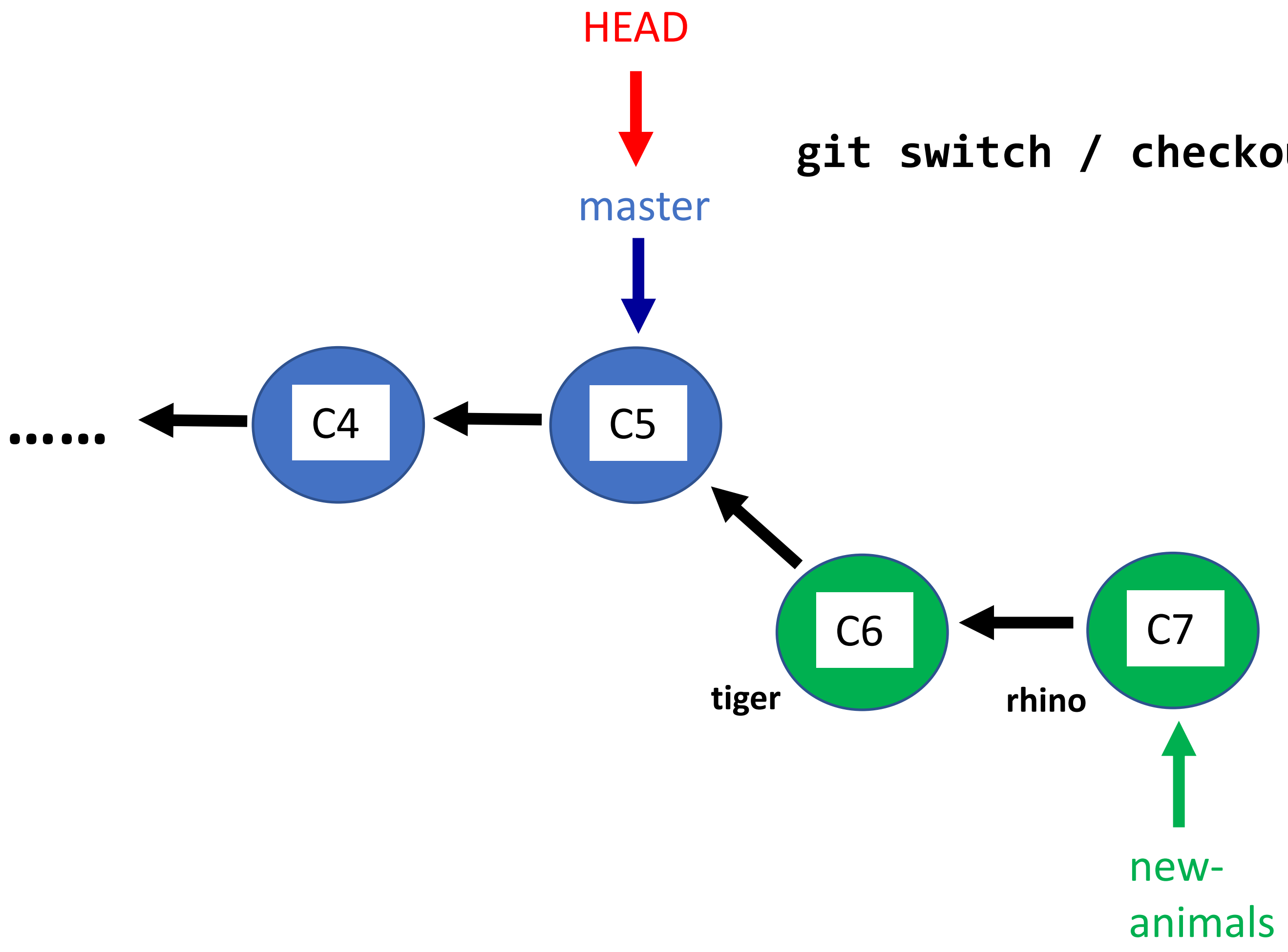
Removes the latest entry from the stack and applies to working directory

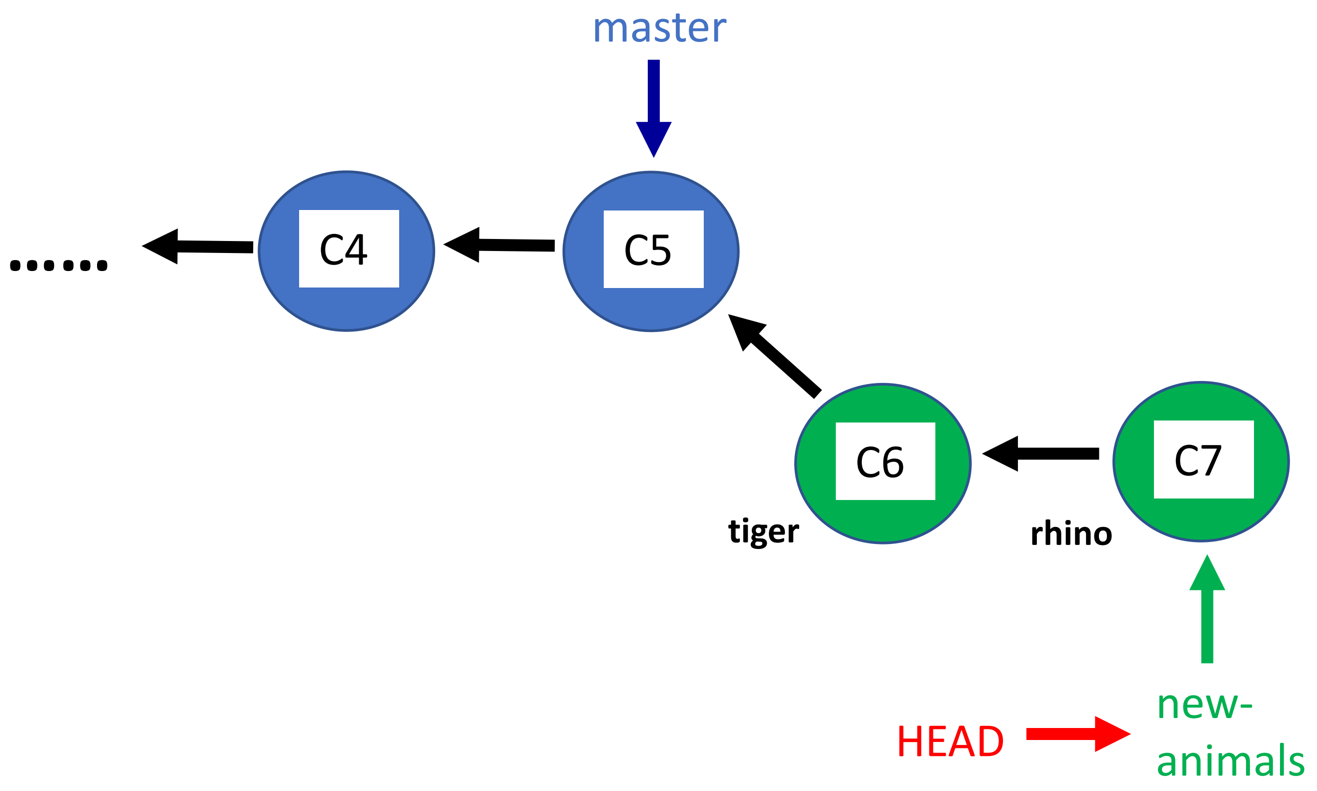
git stash apply

Applies the latest entry from the stack to working directory but keeps it in the stack

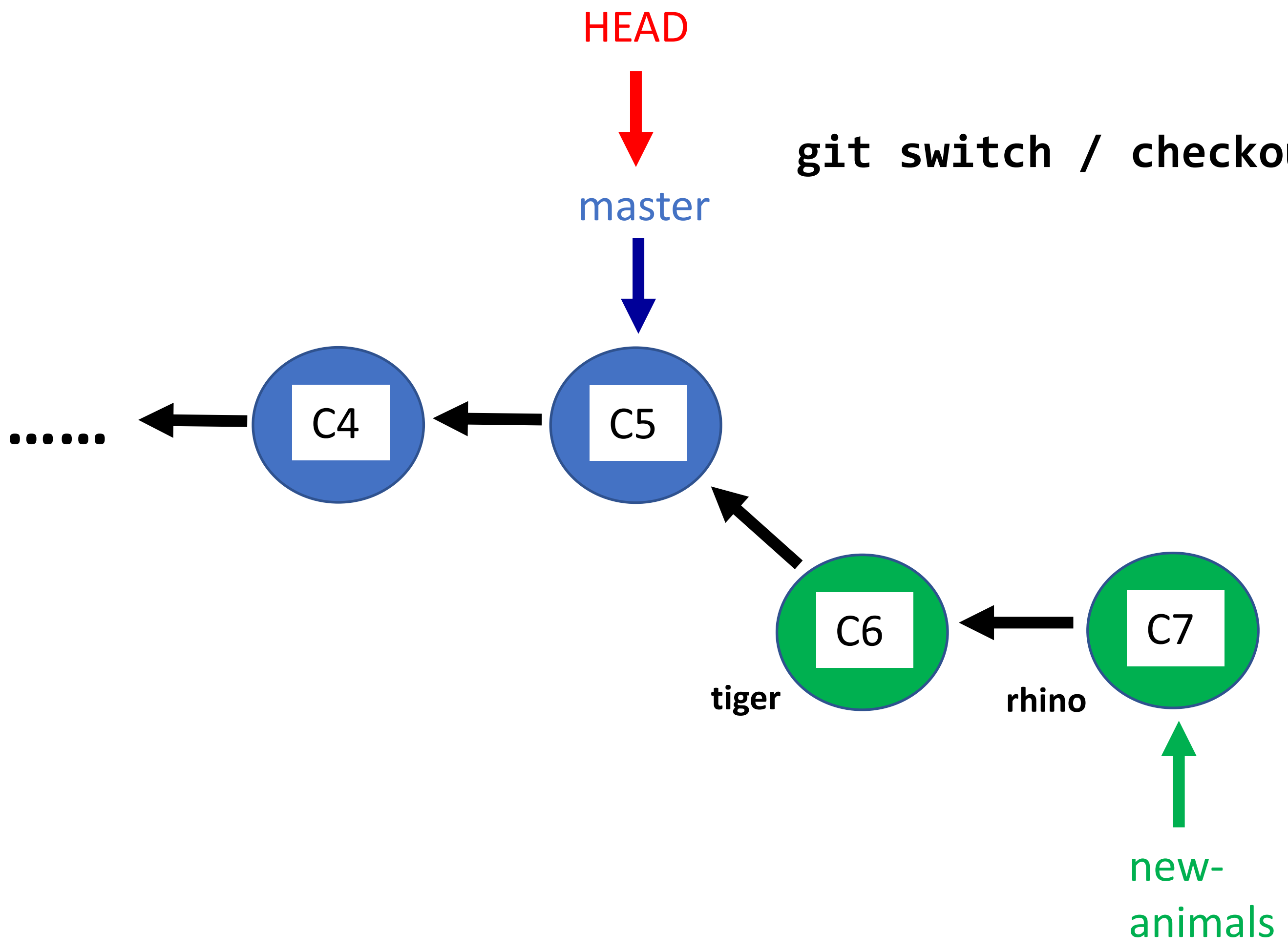


```
git switch / checkout master
```

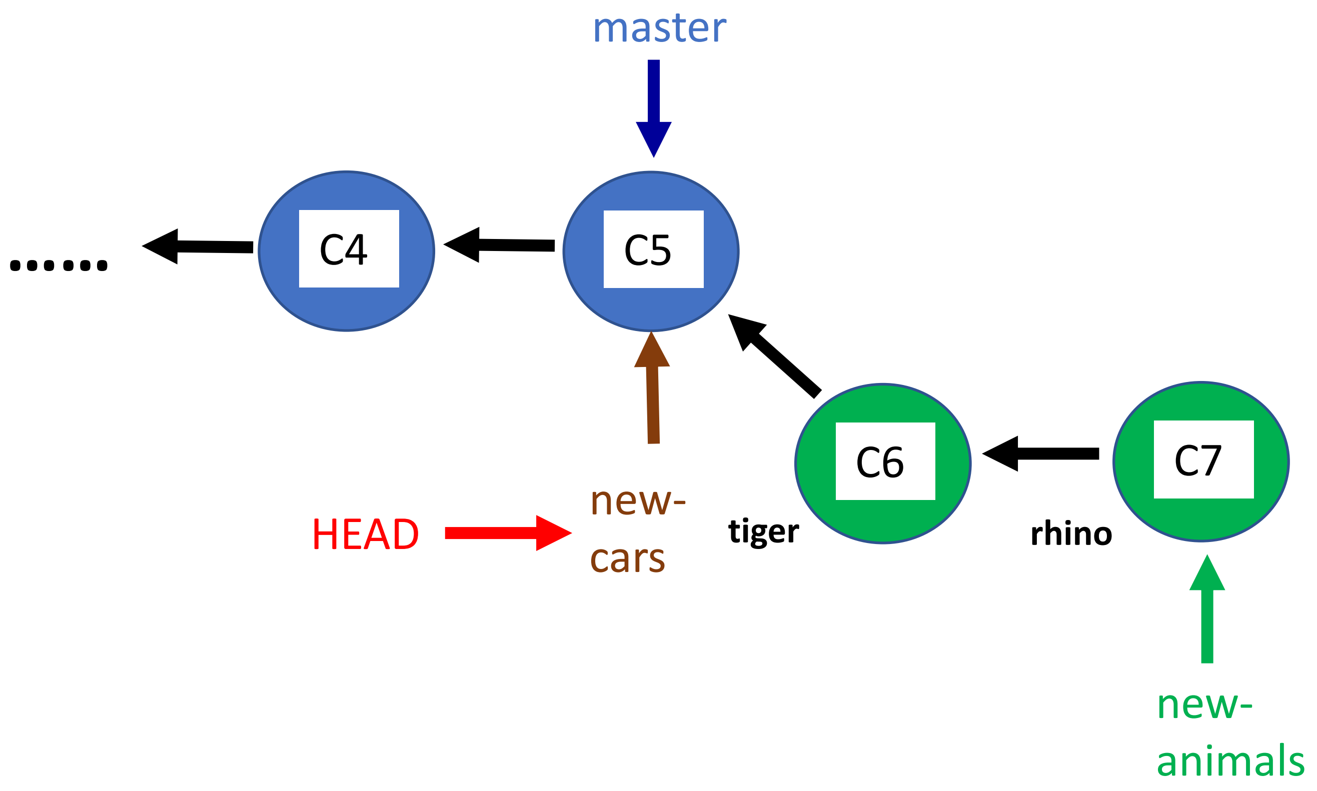


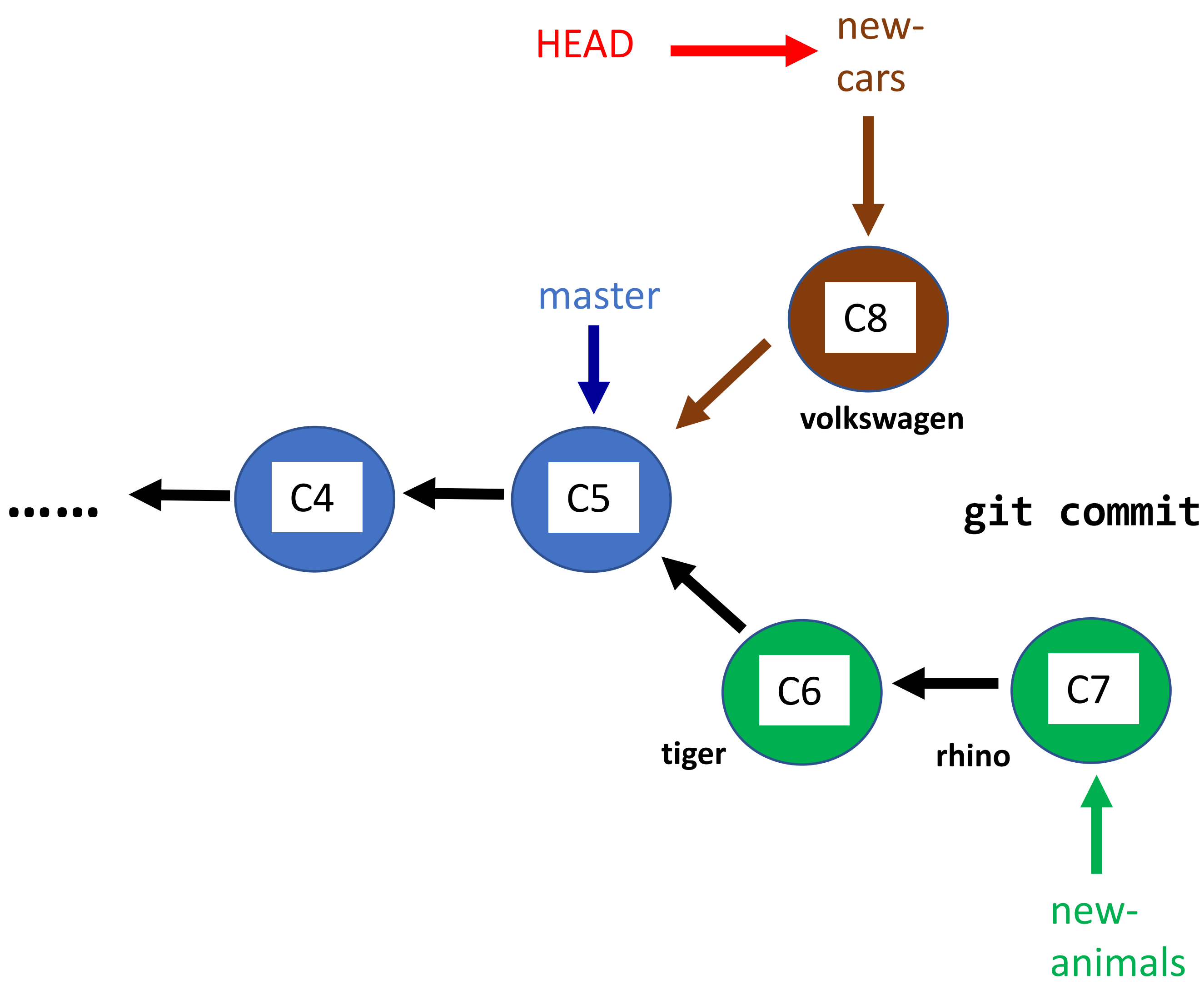


```
git switch / checkout new-animals
```

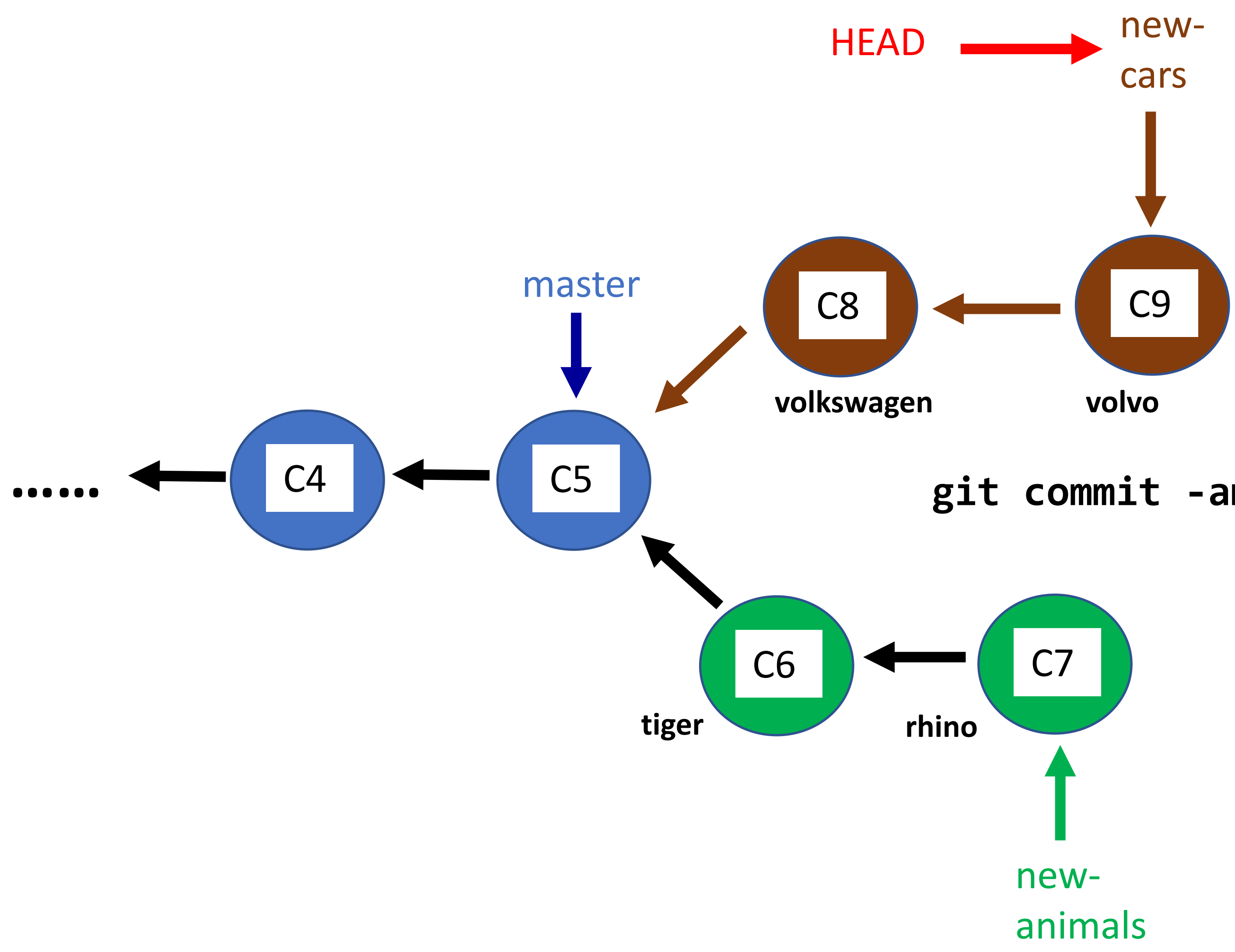


```
git switch / checkout master
git checkout -b new-cars
```





```
git commit -am "Added a new volkswagen"
```

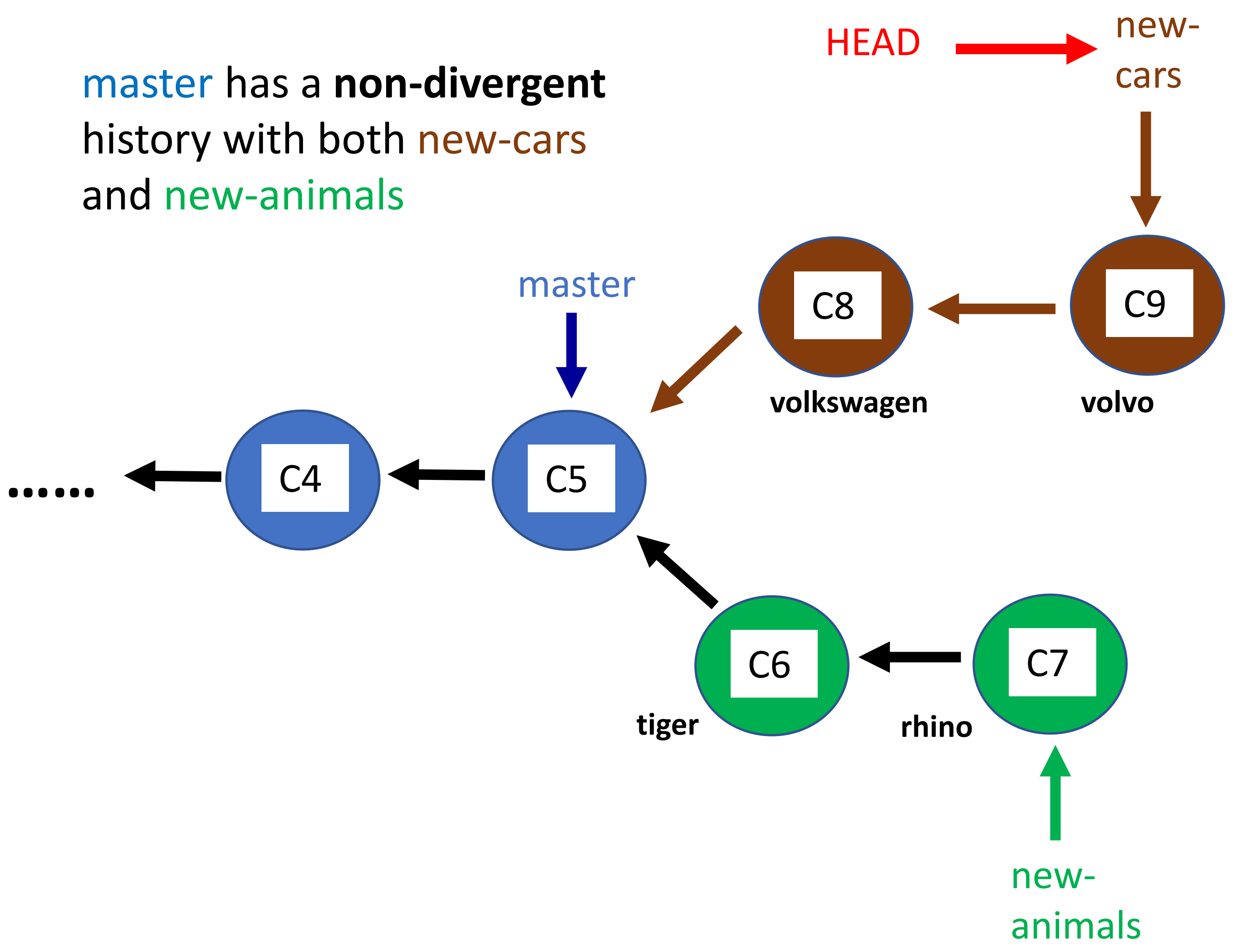


```
git commit -am "Added a new volvo"
```

Divergent vs non-divergent branches

- When Branch A (master) is a direct ancestor of Branch B (new-cars/new-animals)
  - All commits in commit history of branch A is also present in branch B
  - Branch A has a non-divergent history with branch B
- When Branch C (new-cars) or Branch D (new-animals) are NOT direct ancestors of each other
  - Branch C has a divergent history with Branch D

master has a **non-divergent** history with both new-cars and new-animals

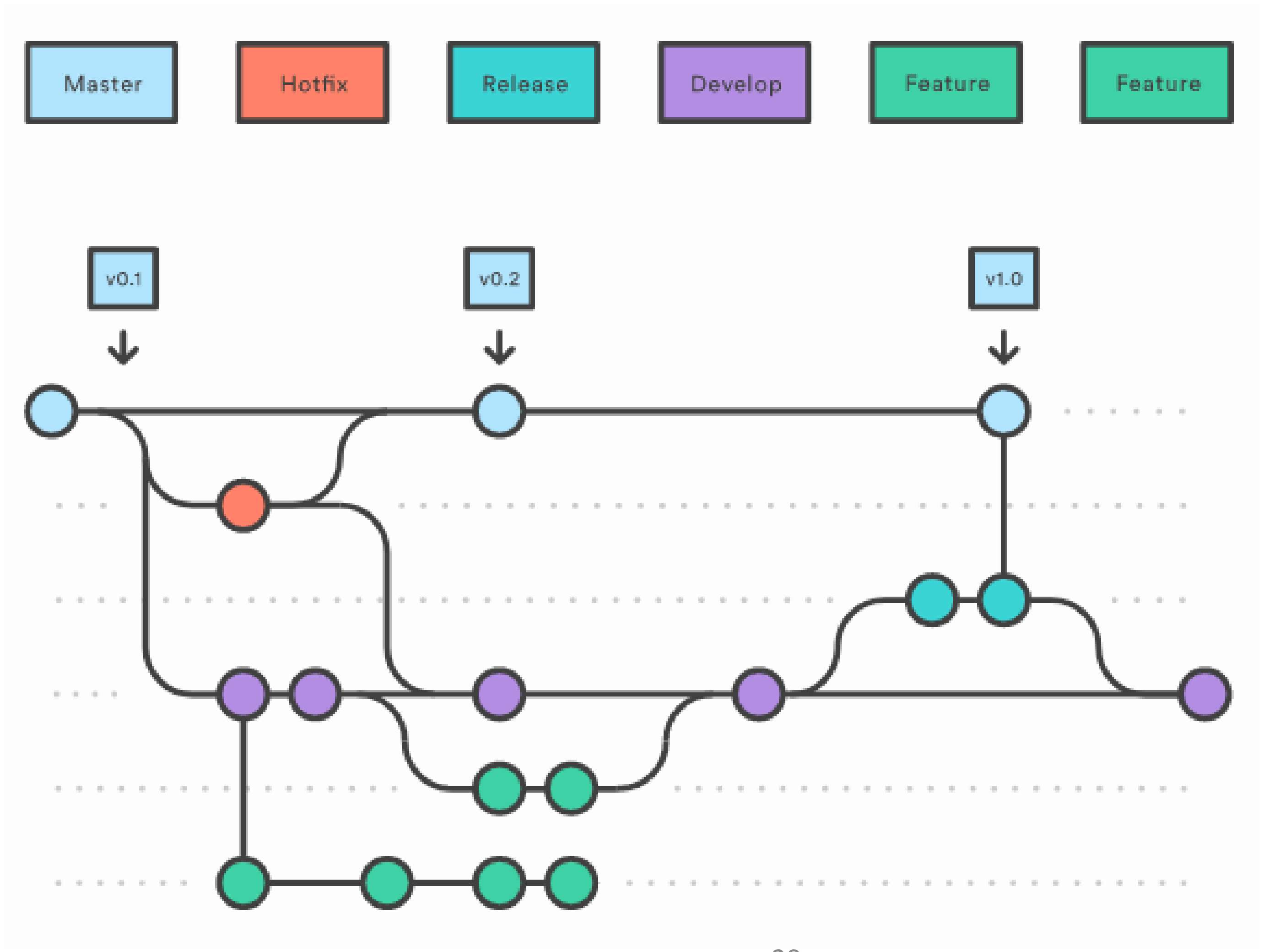


new-cars has a **divergent** history with new-animals

Git lab 3

6 Working with tags

Complex branch workflow



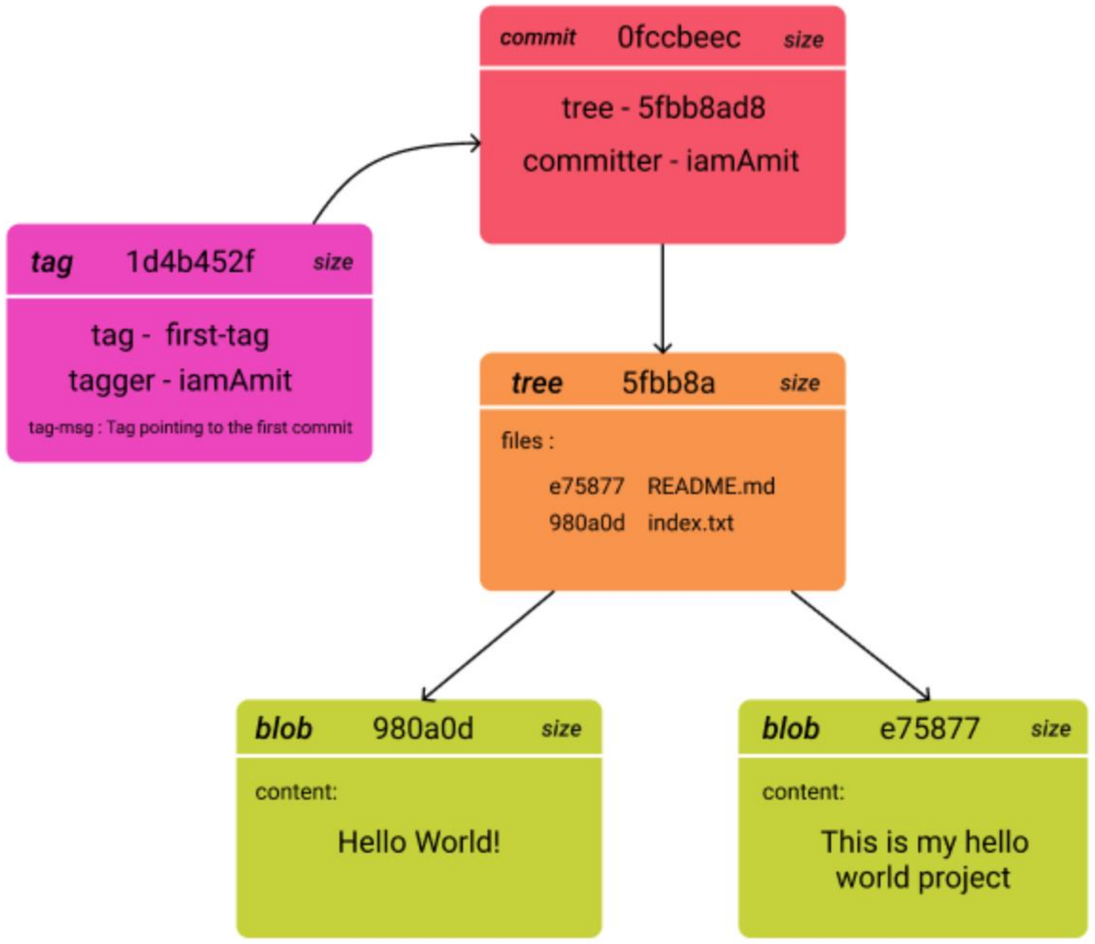
Git object model

- ❖Git architecture involves a structure that consists of 4 different types of objects:
  - These are: blob, tree, commit and tag
  - These different objects also have a size and content
- ❖The structure becomes a “file-system” that runs on top of the native file system.

Git object model: Tag

- Tag
  - Used to mark a specific commit as being special
  - For e.g. to mark a commit as a production release, testing release, etc
- Contains additional metadata
  - Name of tagger
  - Tag message
  - Date of tag creation

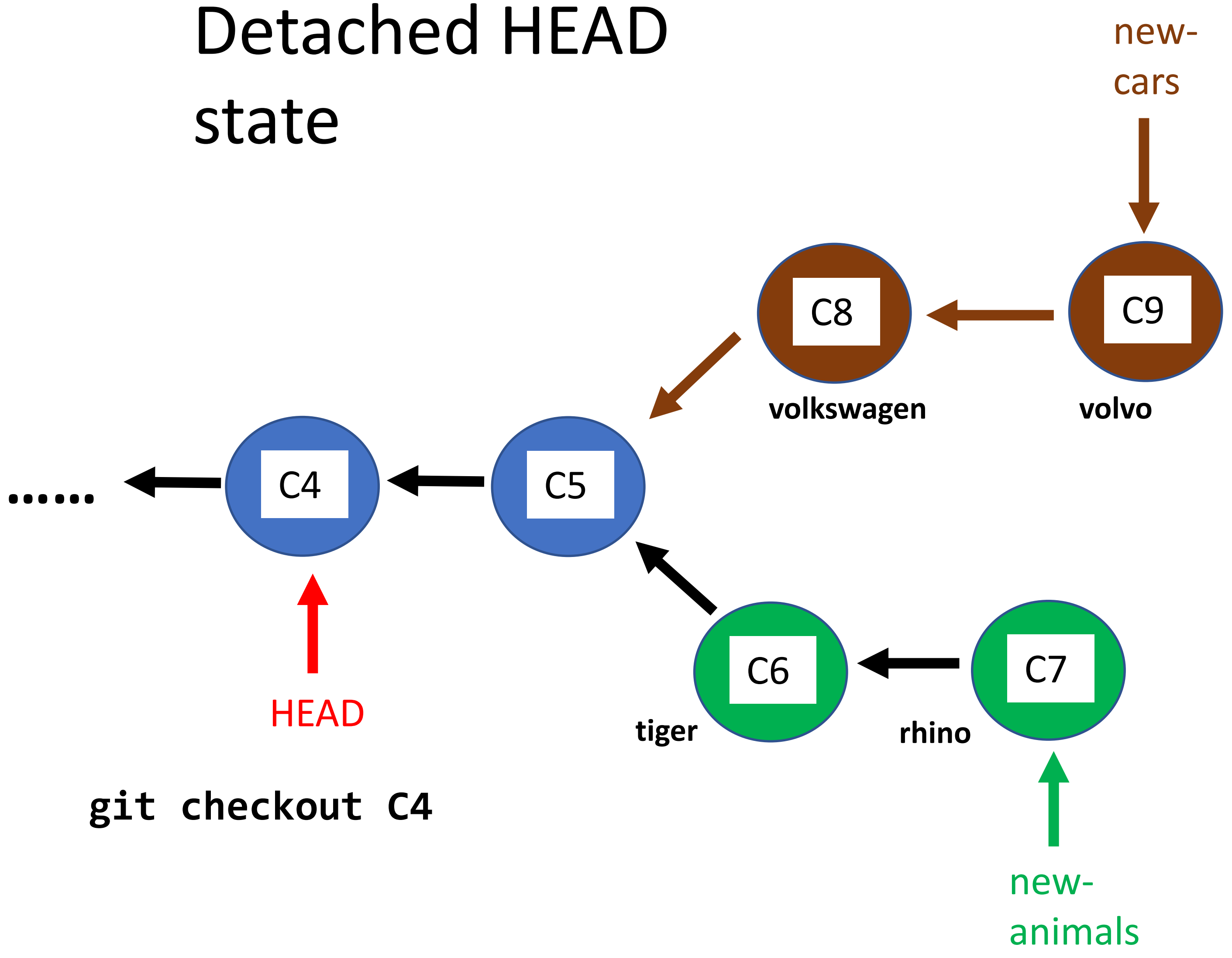
Git object model: Tag



Git lab 3

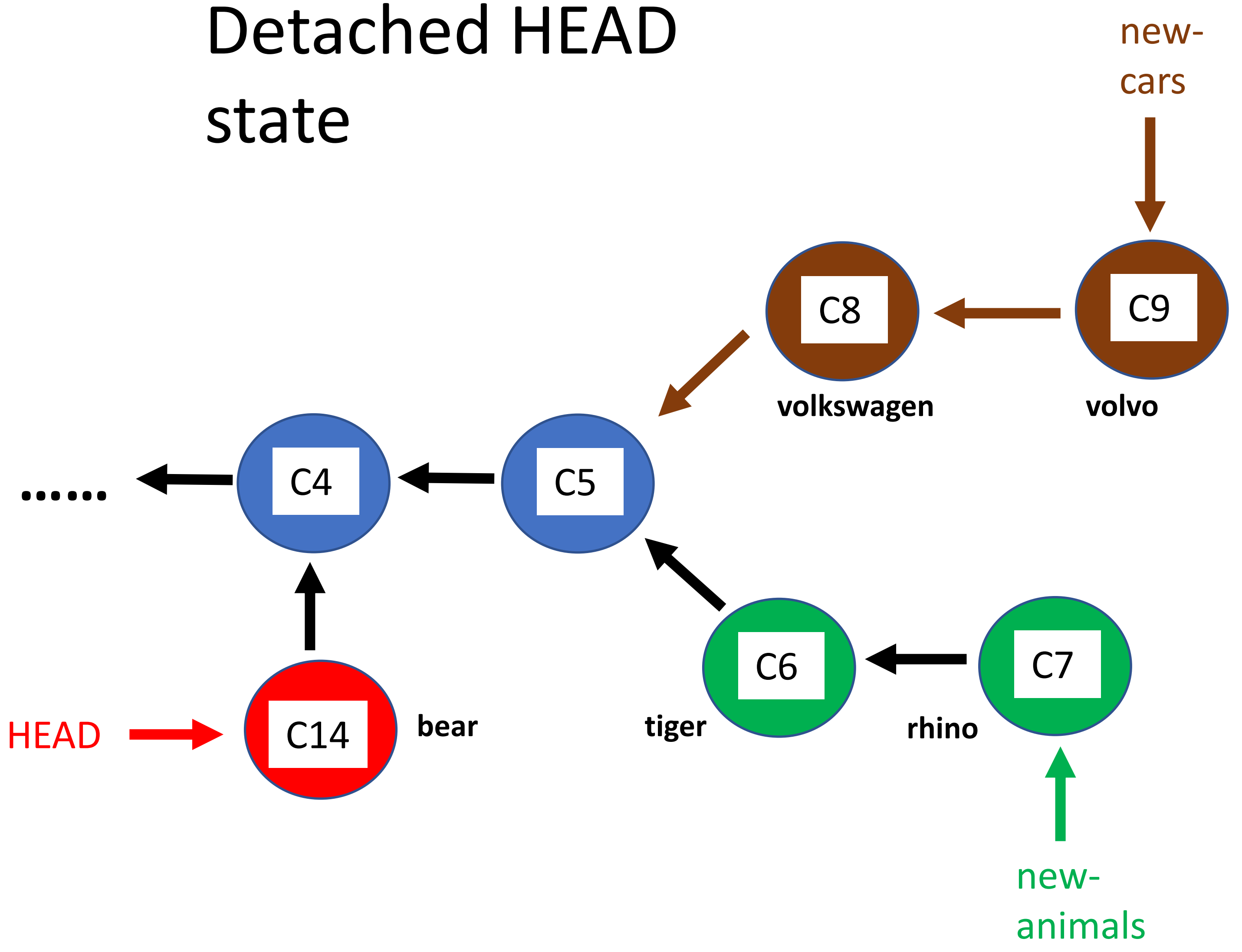
7 Working with untracked commits

Detached HEAD state



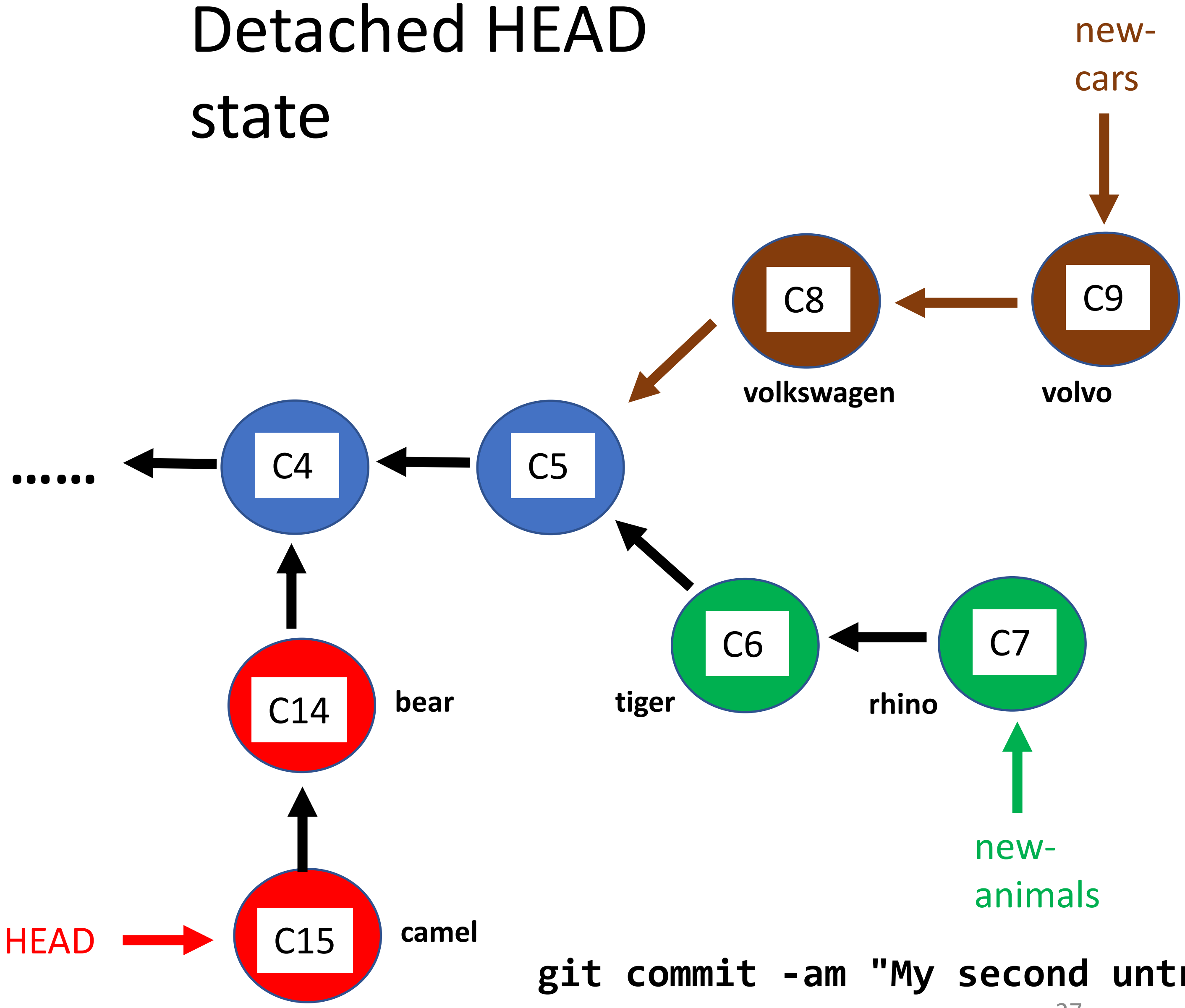
```
git checkout C4
```

Detached HEAD state



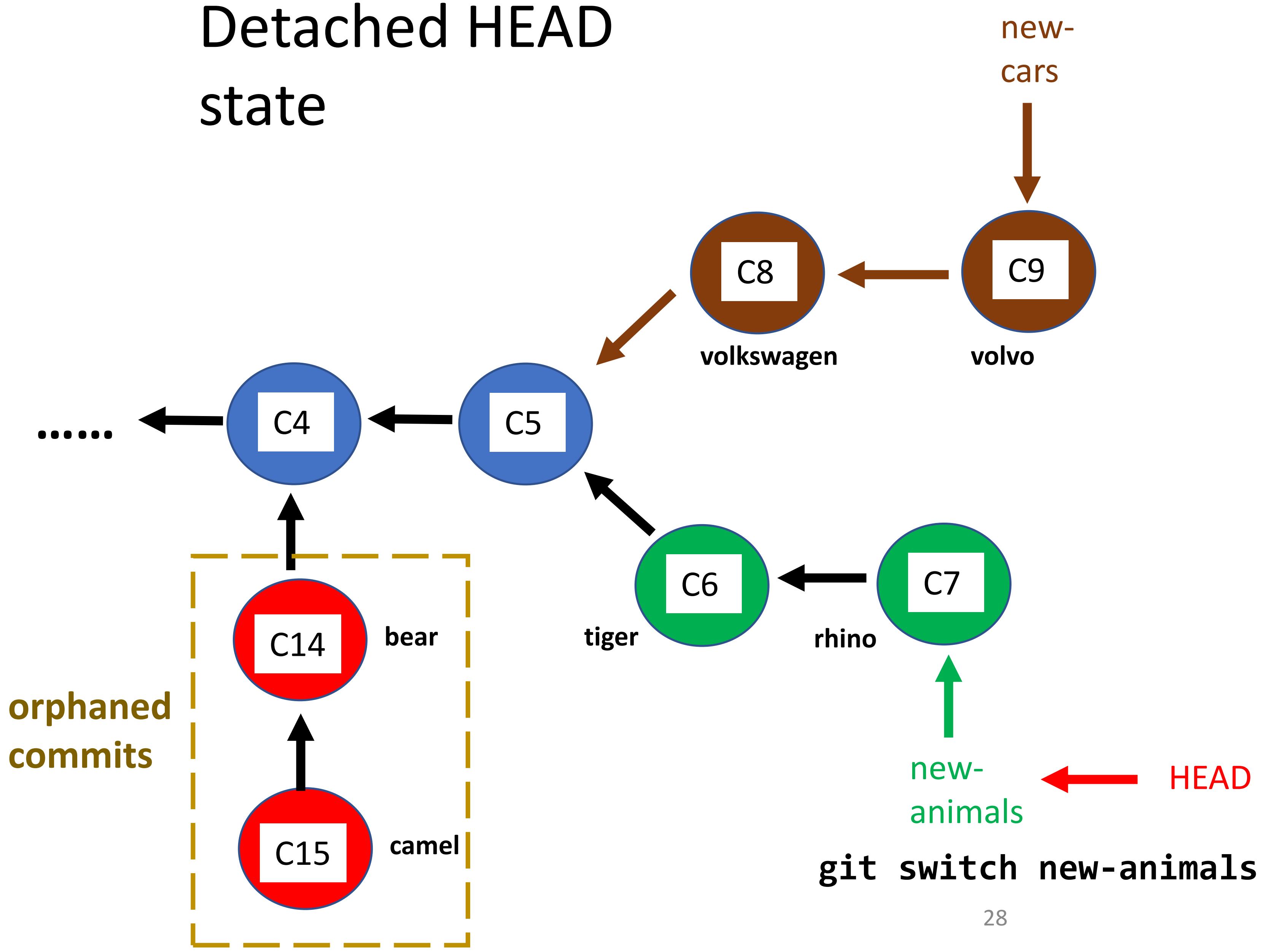
```
git commit -am "My first untracked commit"
```

Detached HEAD state



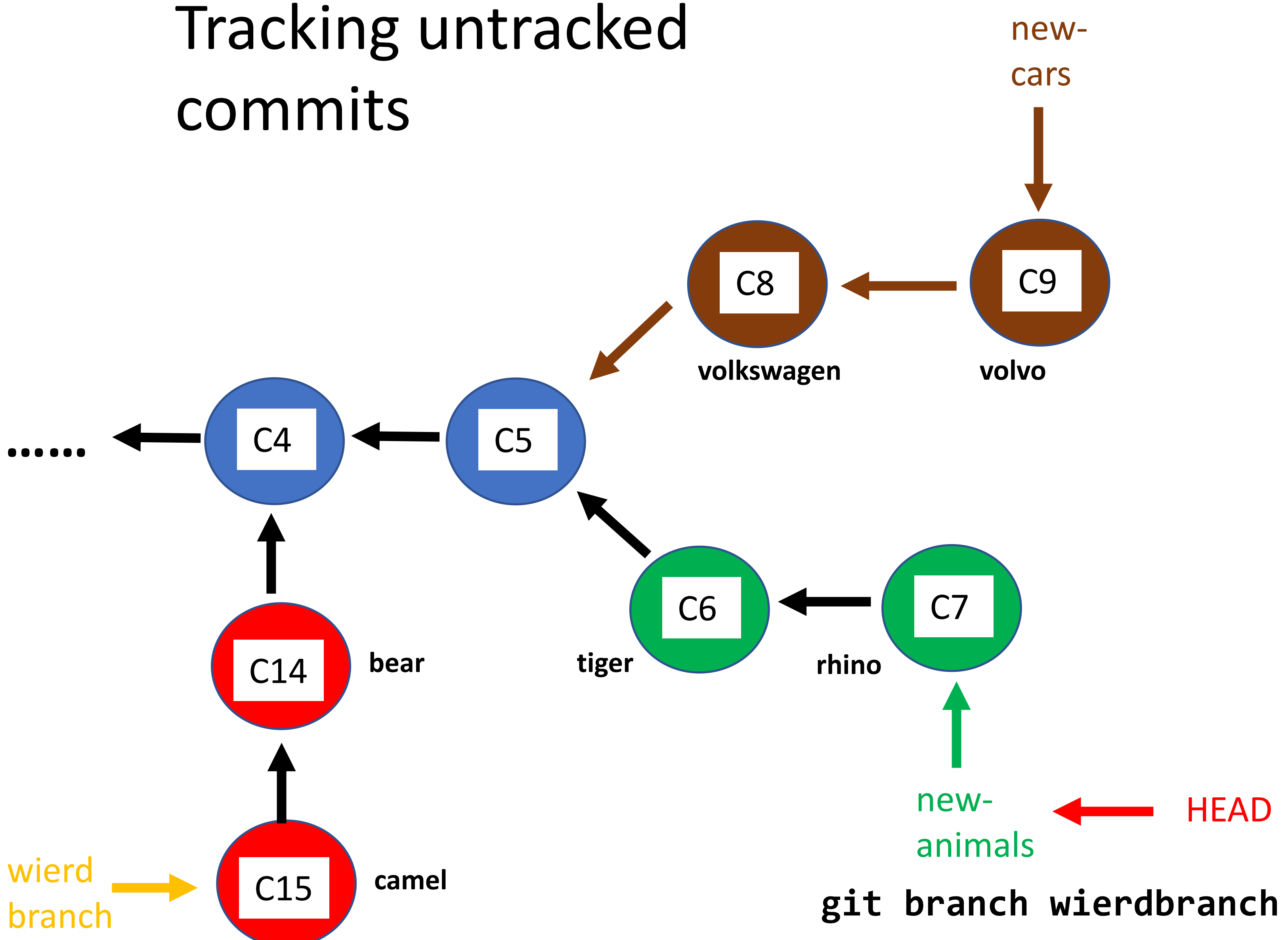
```
git commit -am "My second untracked commit"
```

Detached HEAD state



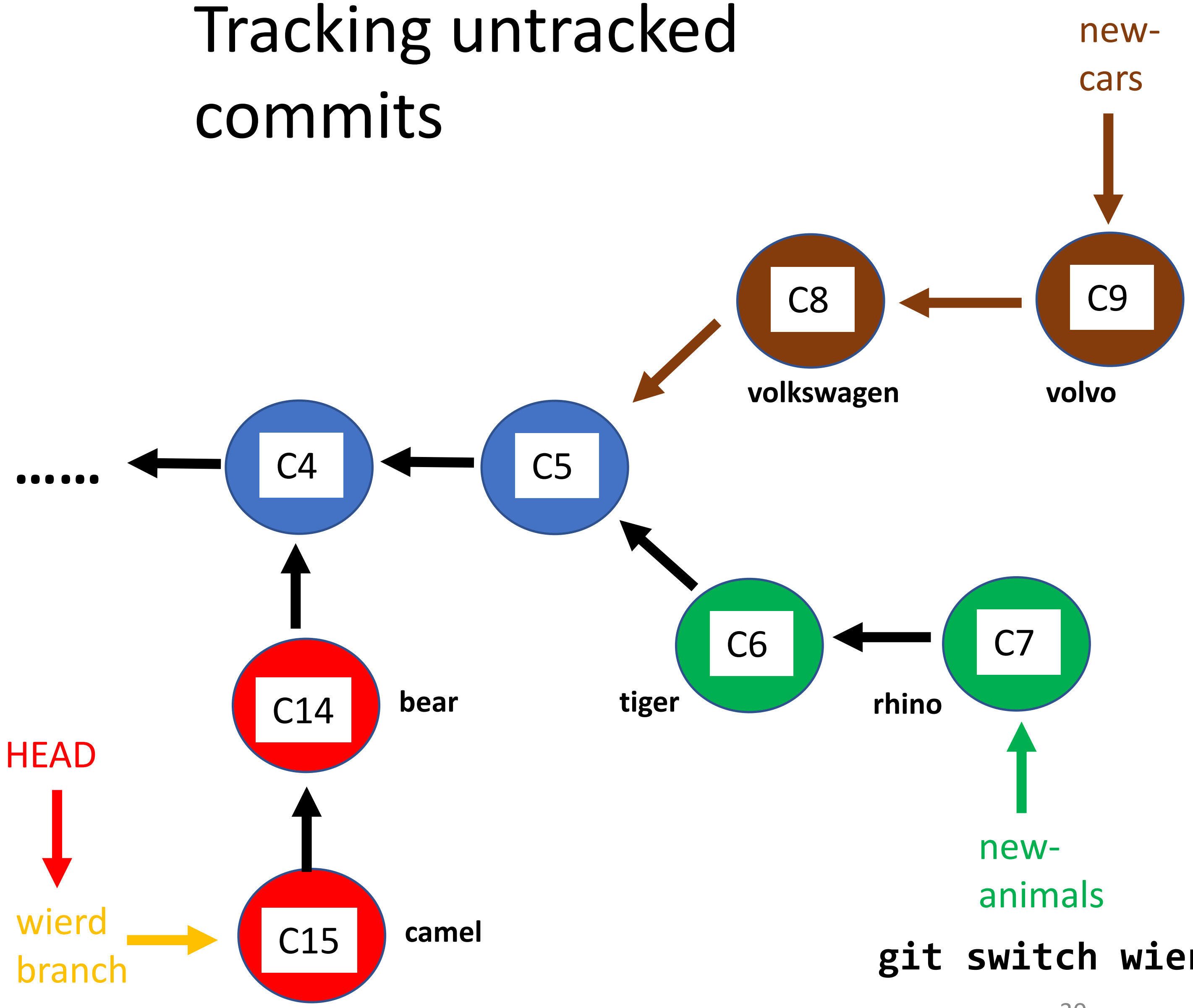
```
git switch new-animals
```

Tracking untracked commits



```
git branch wierdbranch C15
```

Tracking untracked commits



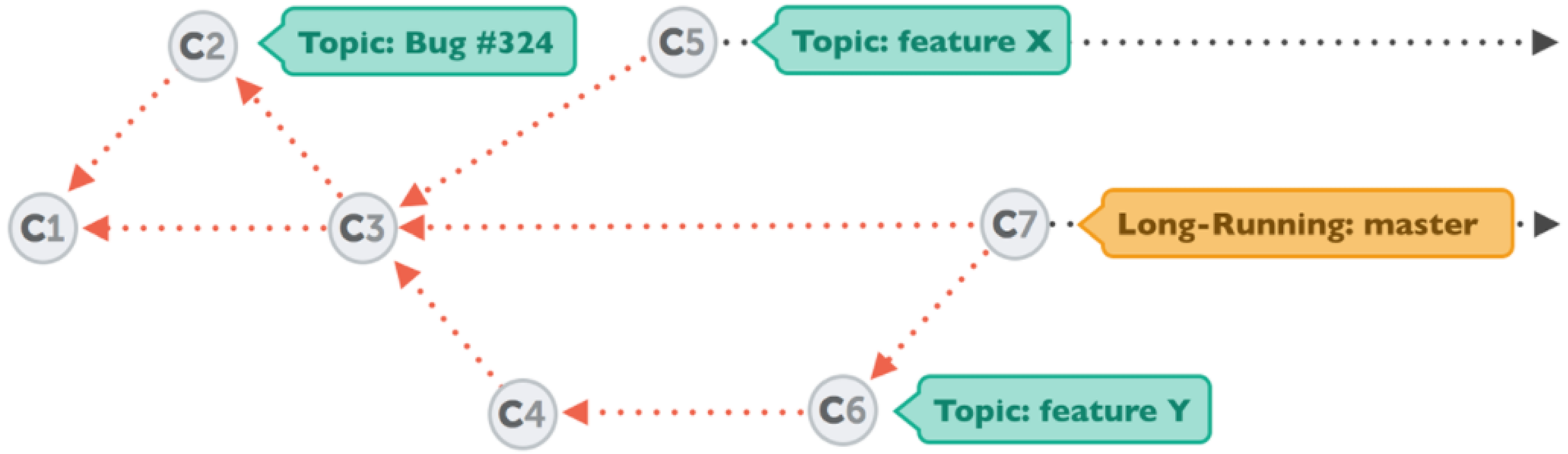
```
git switch wierdbranch
```

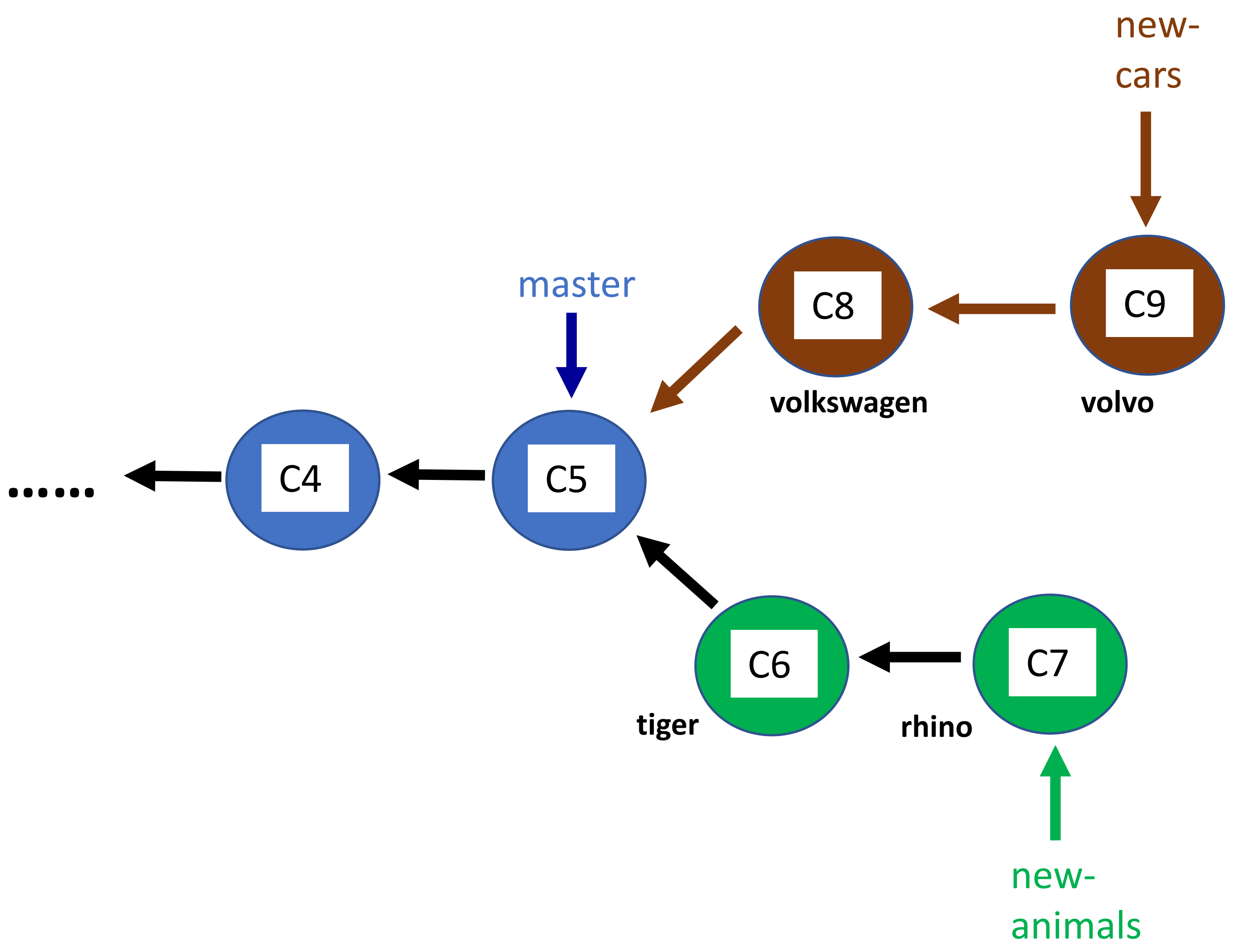
Git lab 3

8 Integrating branches with a fast-forward merge

Using branches in Git

When a particular line of work in a side branch is complete, we can merge it back into the main / master branch

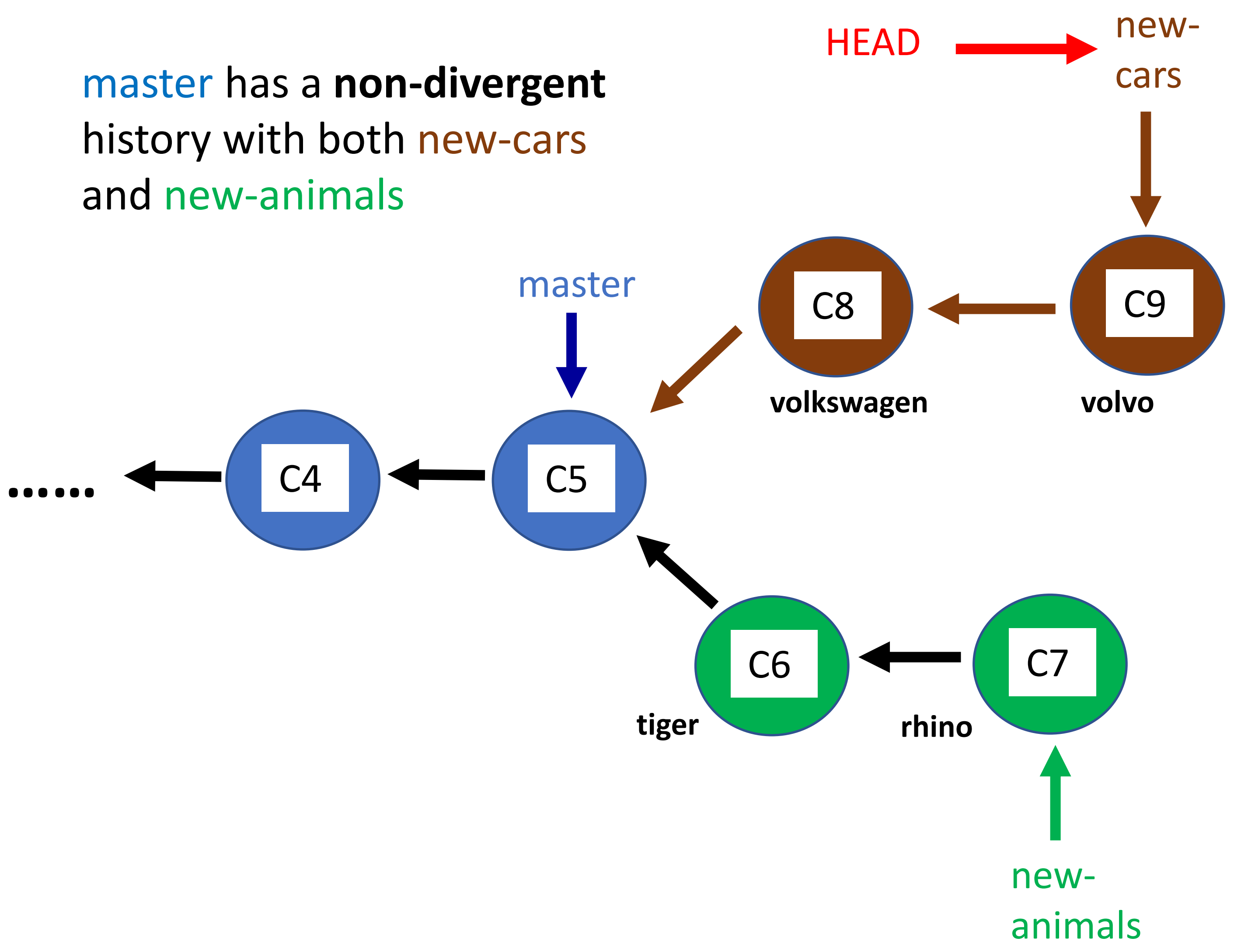



new-
cars
master
C8
C9
volkswagen
volvo
......
C4
C5
C6
C7
tiger
rhino
new-
animals

Integrating branches (2 general strategies)

- Merge
  - The most commonly used
  - Fast forward merge (non-divergent) or 3-way merge (divergent)
- Rebase
  - More complex
  - Use for specific situations

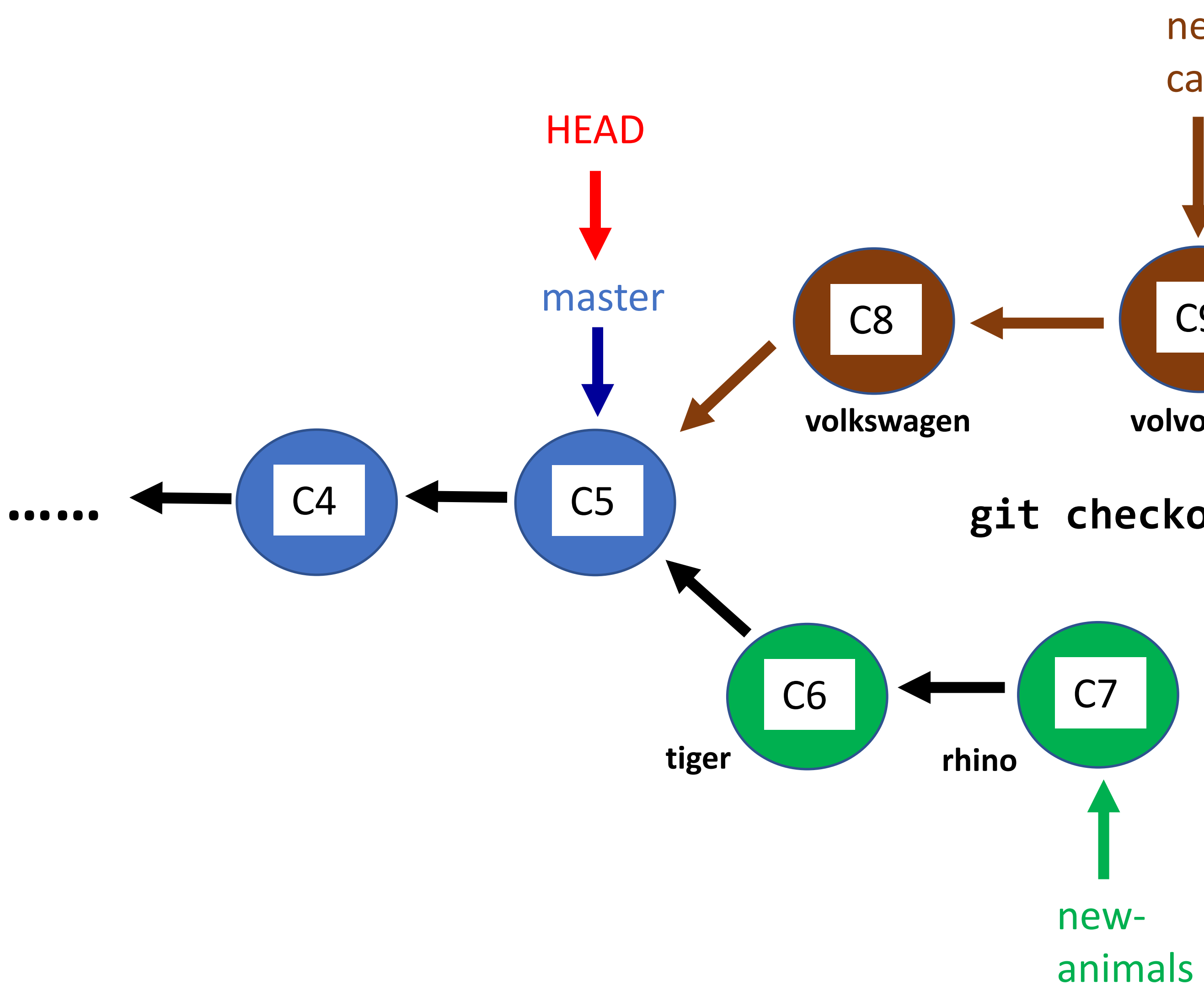
master has a **non-divergent** history with both new-cars and new-animals



new-cars has a **divergent** history with new-animals

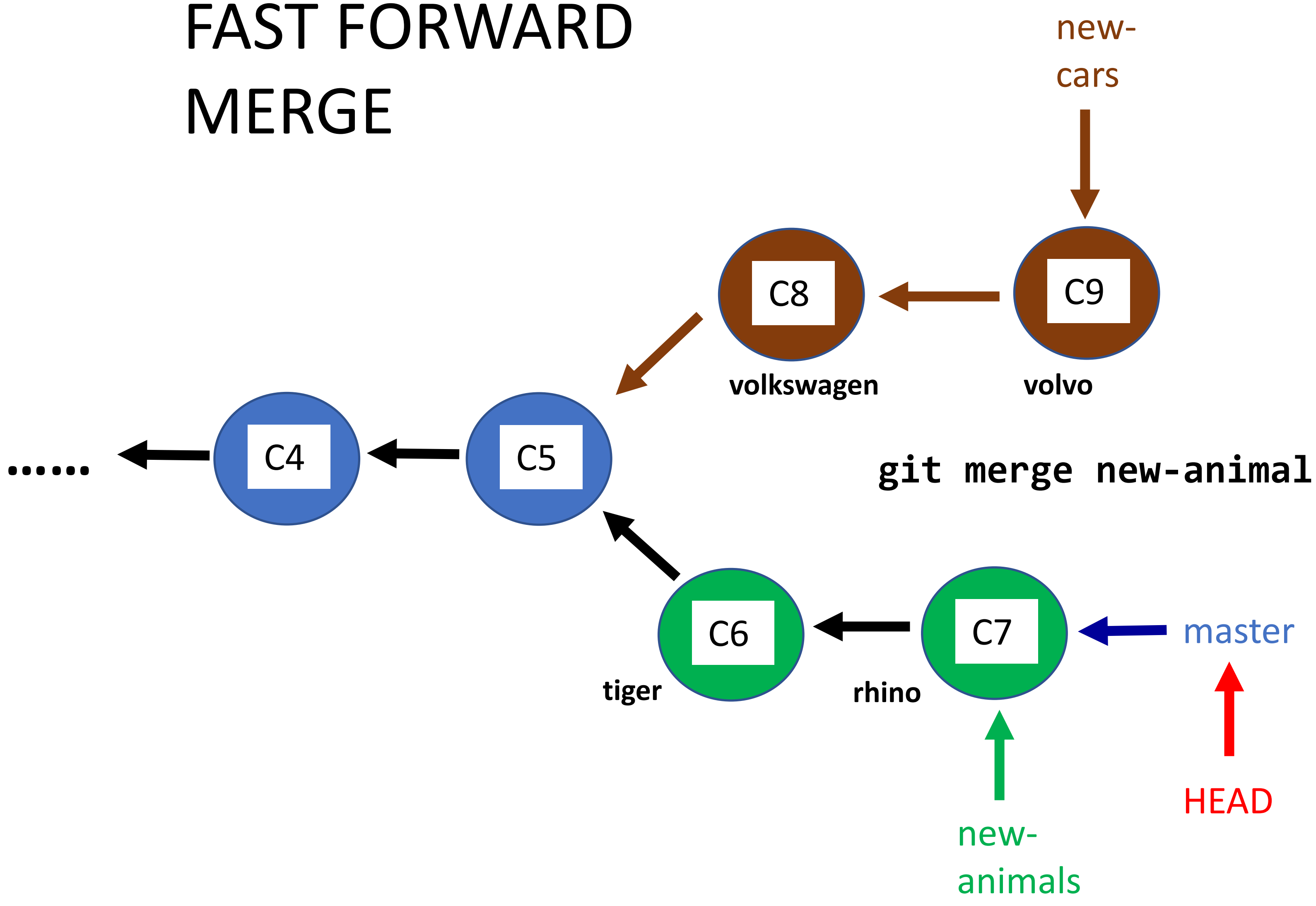
Fast forward merge

- When the branches being merged have a non-divergent history
  - Aligns both branch pointers at the commit which the most advanced branch points at
  - Does not result in the creation of an additional commit



```
git checkout master
```

FAST FORWARD MERGE

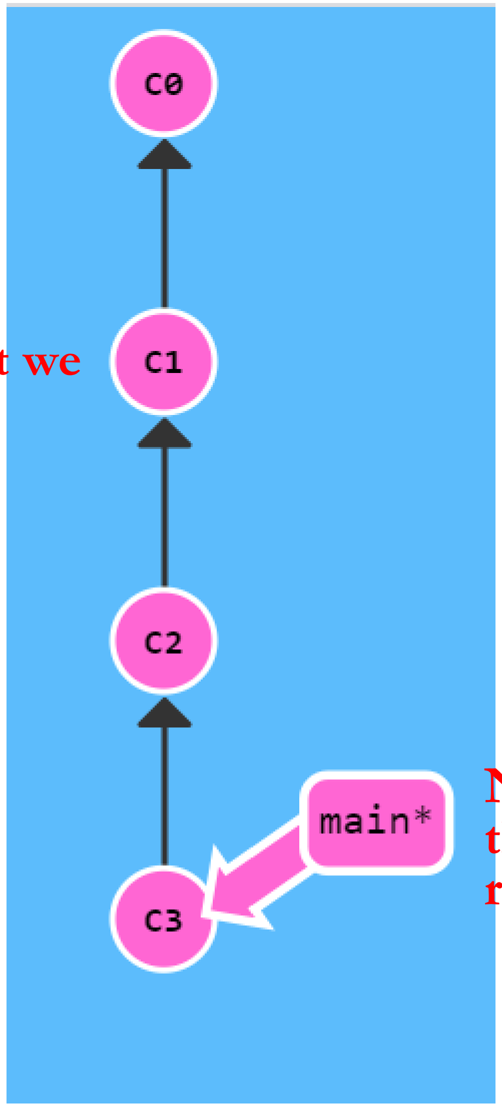


```
git merge new-animals
```

Git lab 3

9 Dropping a commit and reverting to a previous one

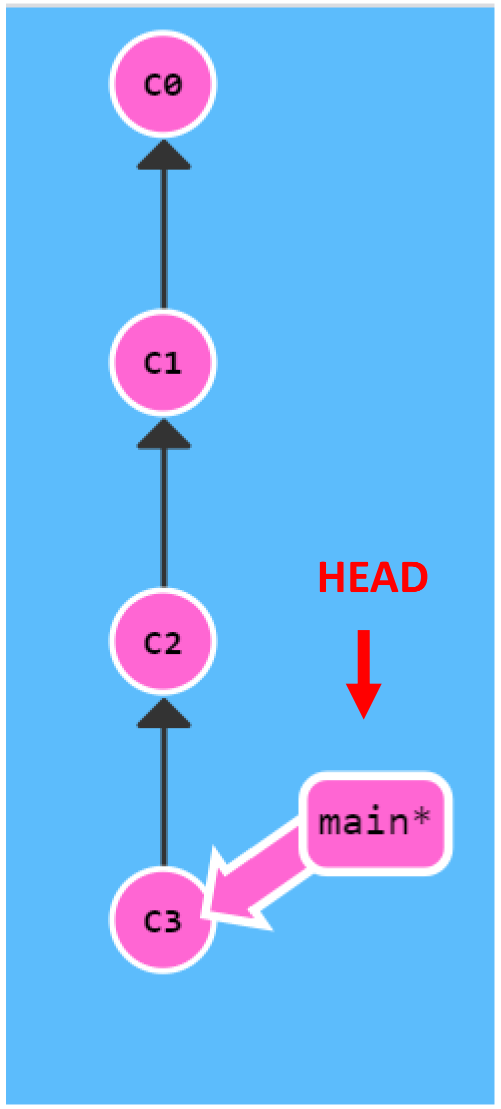
Reverting to an older commit



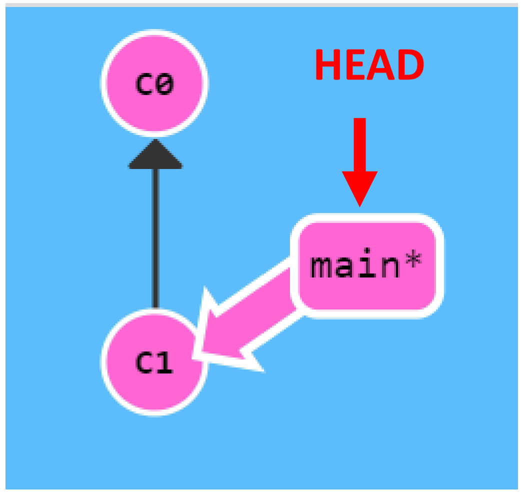
Assume that we know this commit contains the code base version that we want to rebuild from

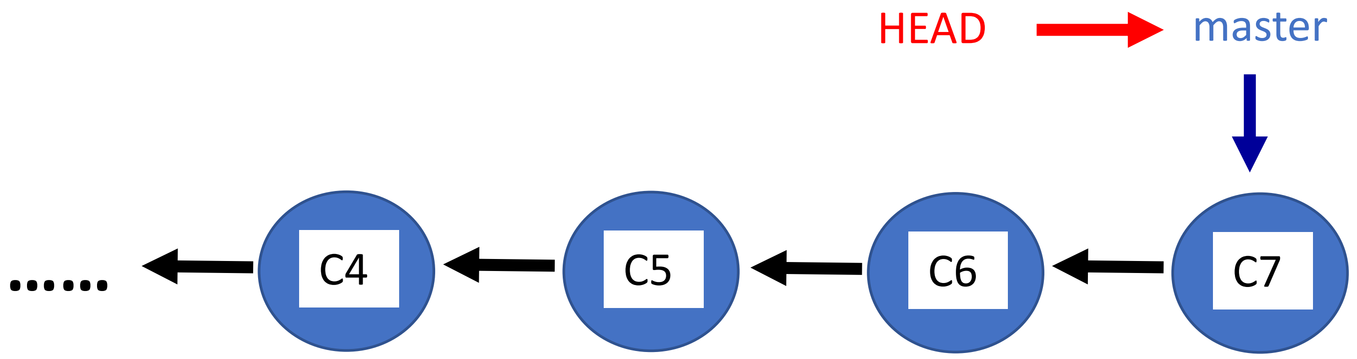
New features introduced through C2 and C3 no longer required

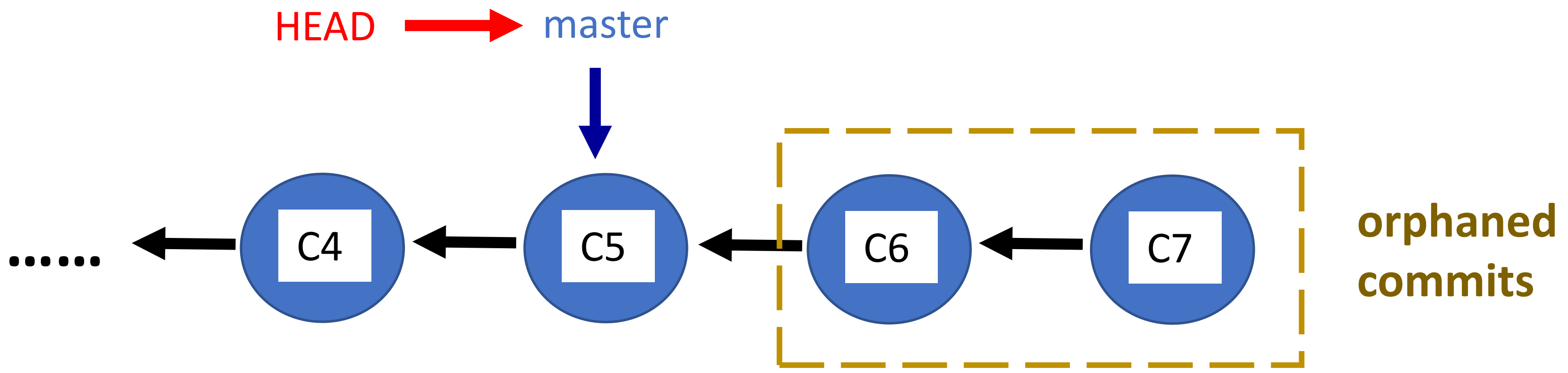
Permanent reversion to older commit



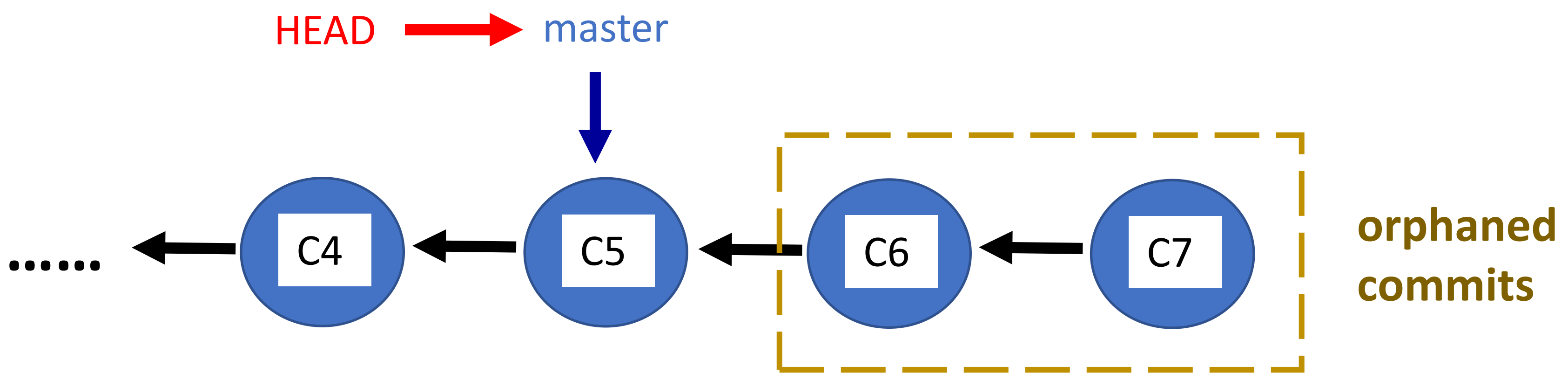
```
git reset --hard C1
```




HEAD
master
......
C4
C5
C6
C7

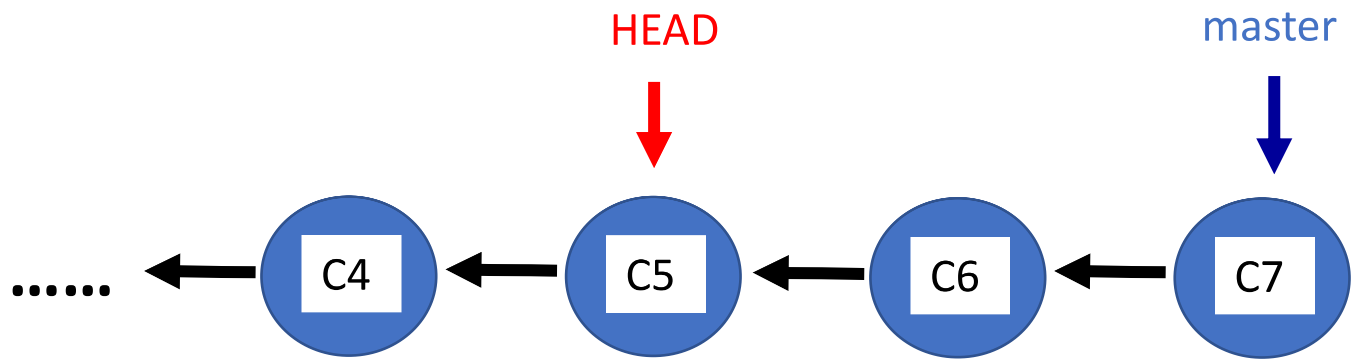


```
git reset --hard C5
```



```
git reset --hard C5
```

VS



```
git checkout C5
```

Deleting orphaned commits

- To delete a commit permanently
  - It must be removed from the Git repo (.git folder)
- First step is to make the commit an orphaned commit
  - Example commands: **git reset --hard, git rebase -i,** etc
- When the orphaned commit remains inaccessible for a certain period of time
  - Git's Garbage Collector (GC) will delete it from the Git repo permanently
  - Can configure this period of time or force GC to run immediately

Git lab 3

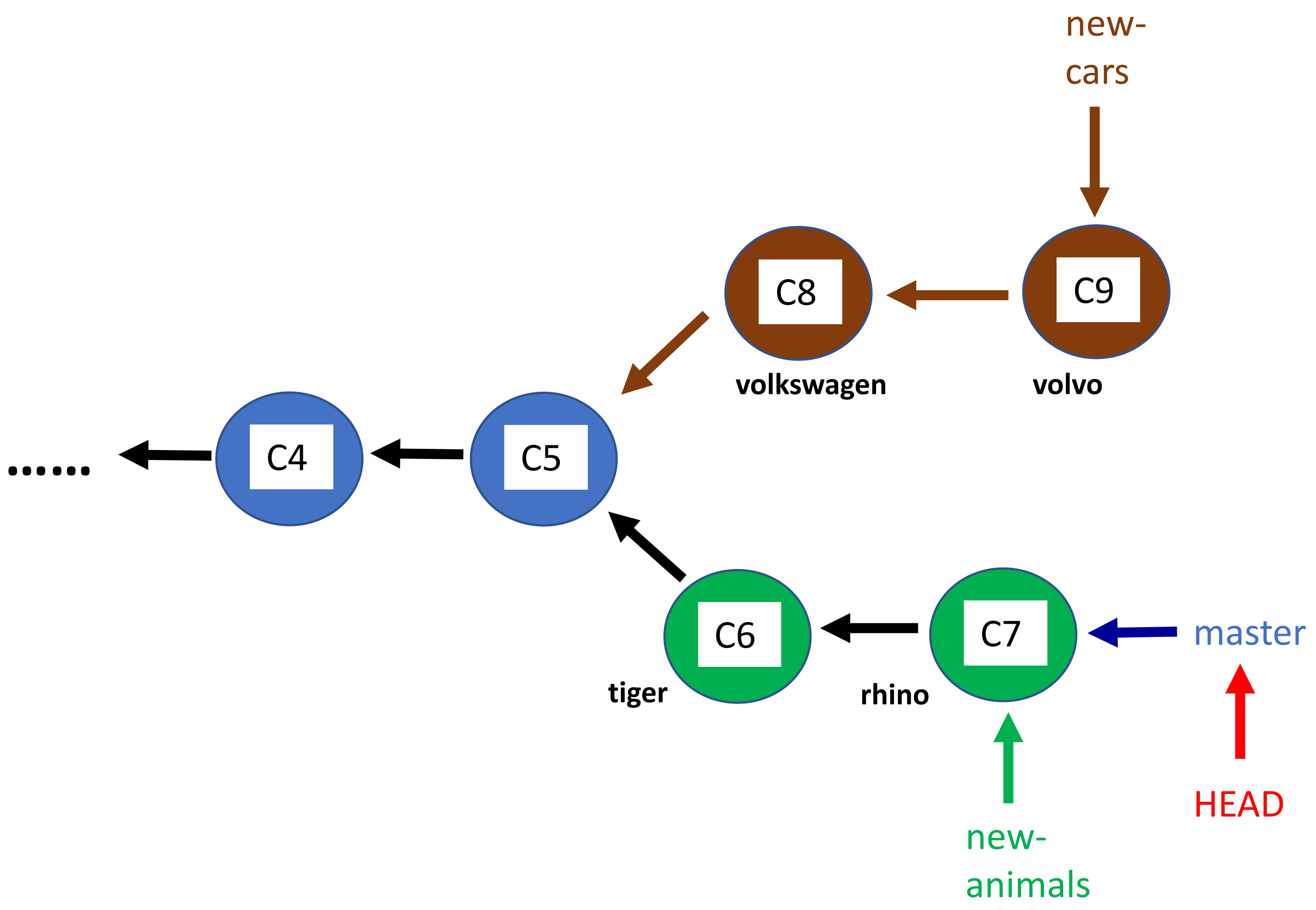
10 Integrating branches with a 3-way merge

Integrating branches (2 general strategies)

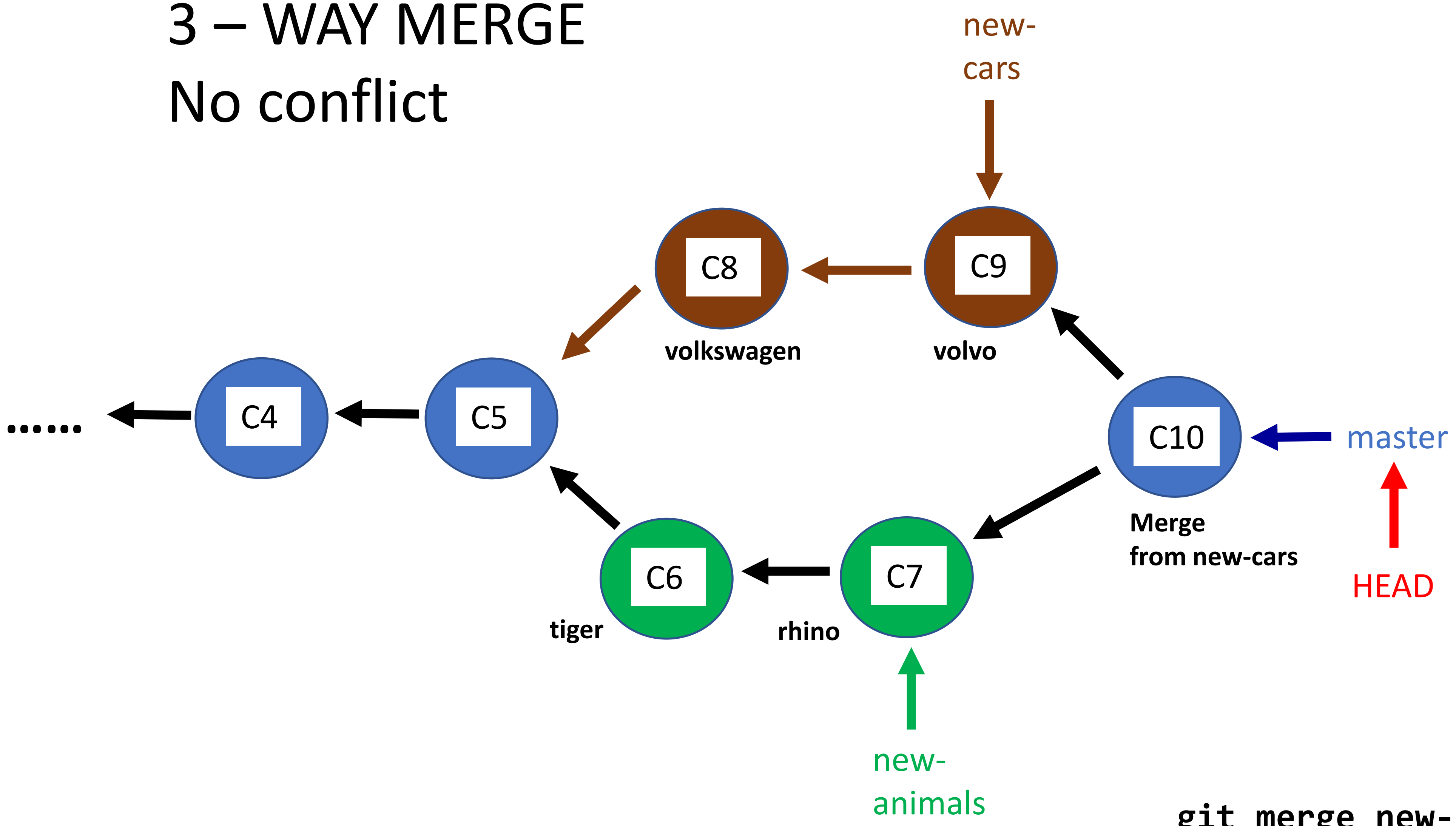
- ❖ Merge
  - The most commonly used
  - Fast forward merge (non-divergent) or 3-way merge (divergent)
- ❖ Rebase
  - More complex
  - Use for specific situations

3 way merge

- ❖ When the branches being merged have a divergent history
  - Results in a creation of an additional merge commit, which will now have two parent commits
  - Can also cause a merge conflict which will now need to be resolved manually


new-
cars
C8
C9
volkswagen
volvo
......
C4
C5
C6
C7
master
tiger
rhino
HEAD
new-
animals

3 – WAY MERGE
No conflict



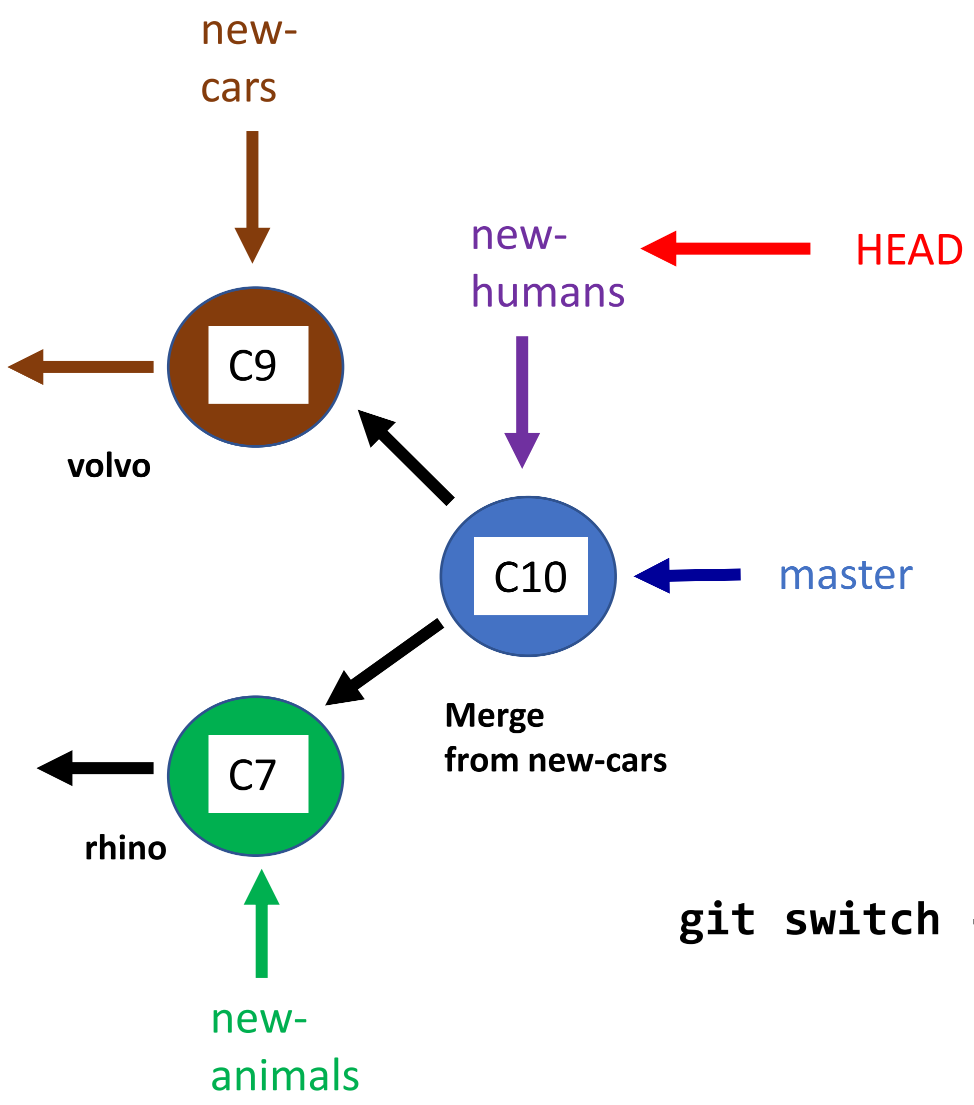
```
git merge new-cars
```

Merge strategies

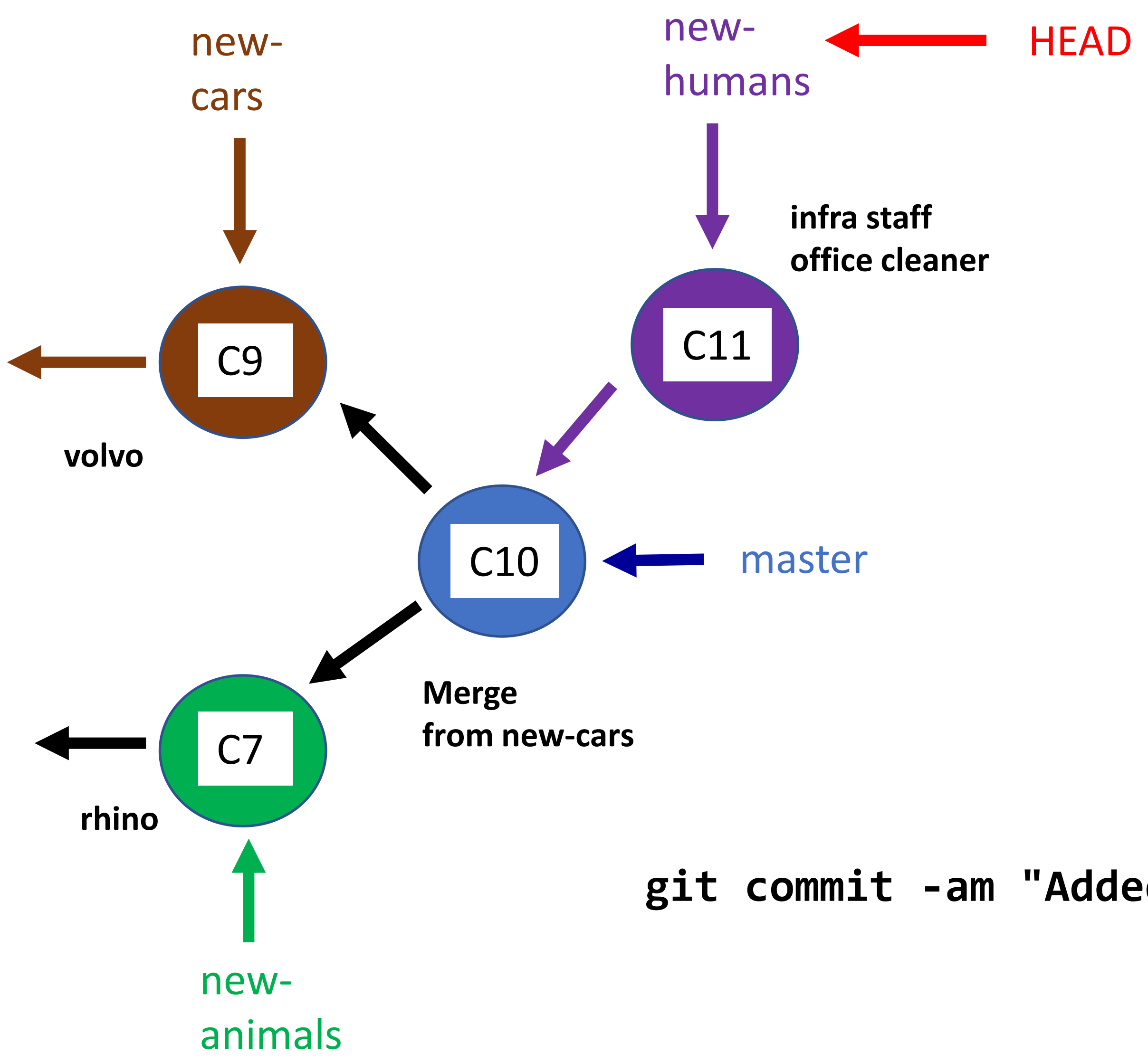
- ❖ Strategy that Git uses to locate common base commit between two divergent branches that are being merged
- ❖ Examples
  - Ort / recursive (default)
  - Ours
  - Octopus
  - Resolve
  - Subtree

Git lab 3

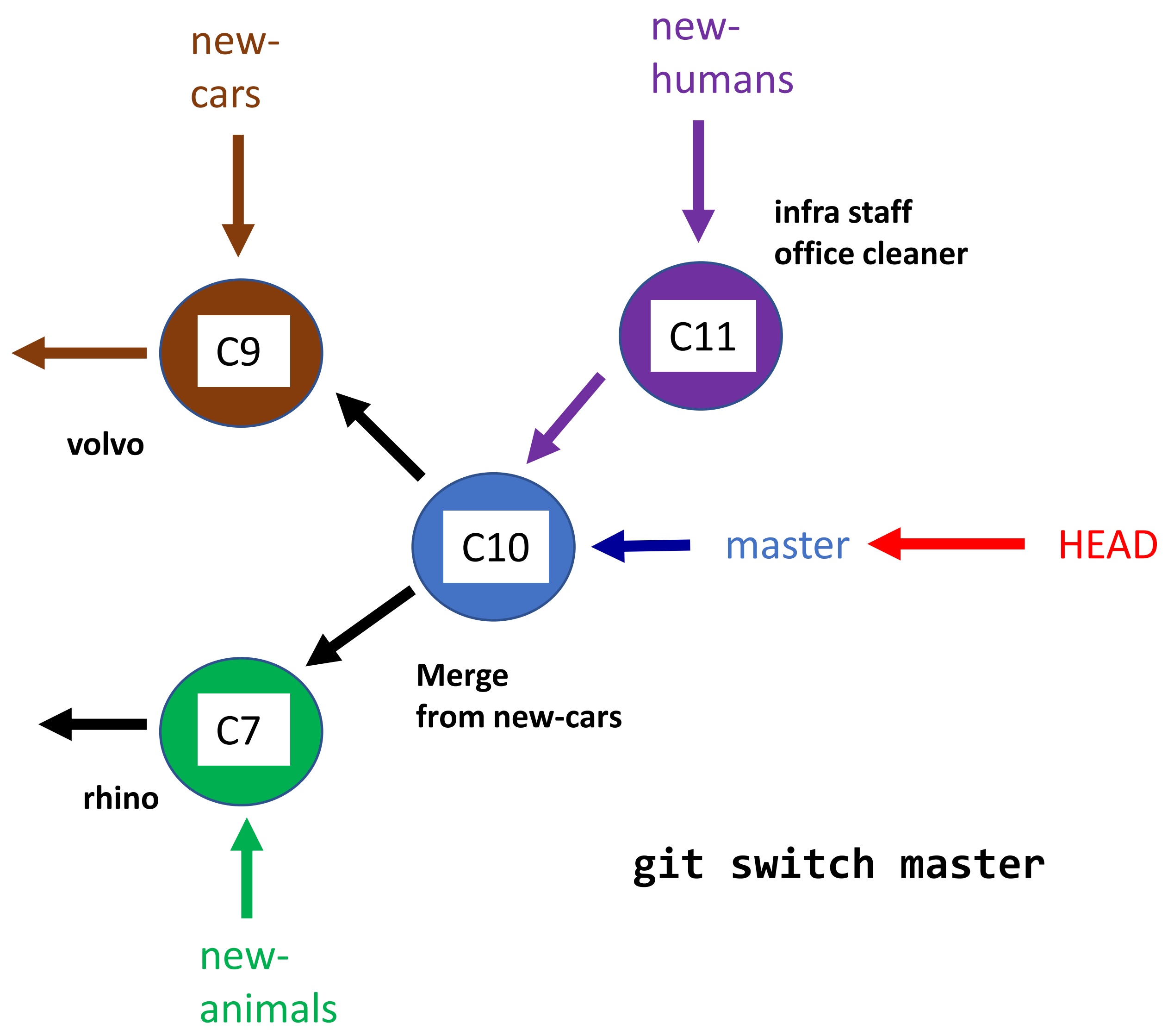
11 Resolving a merge conflict



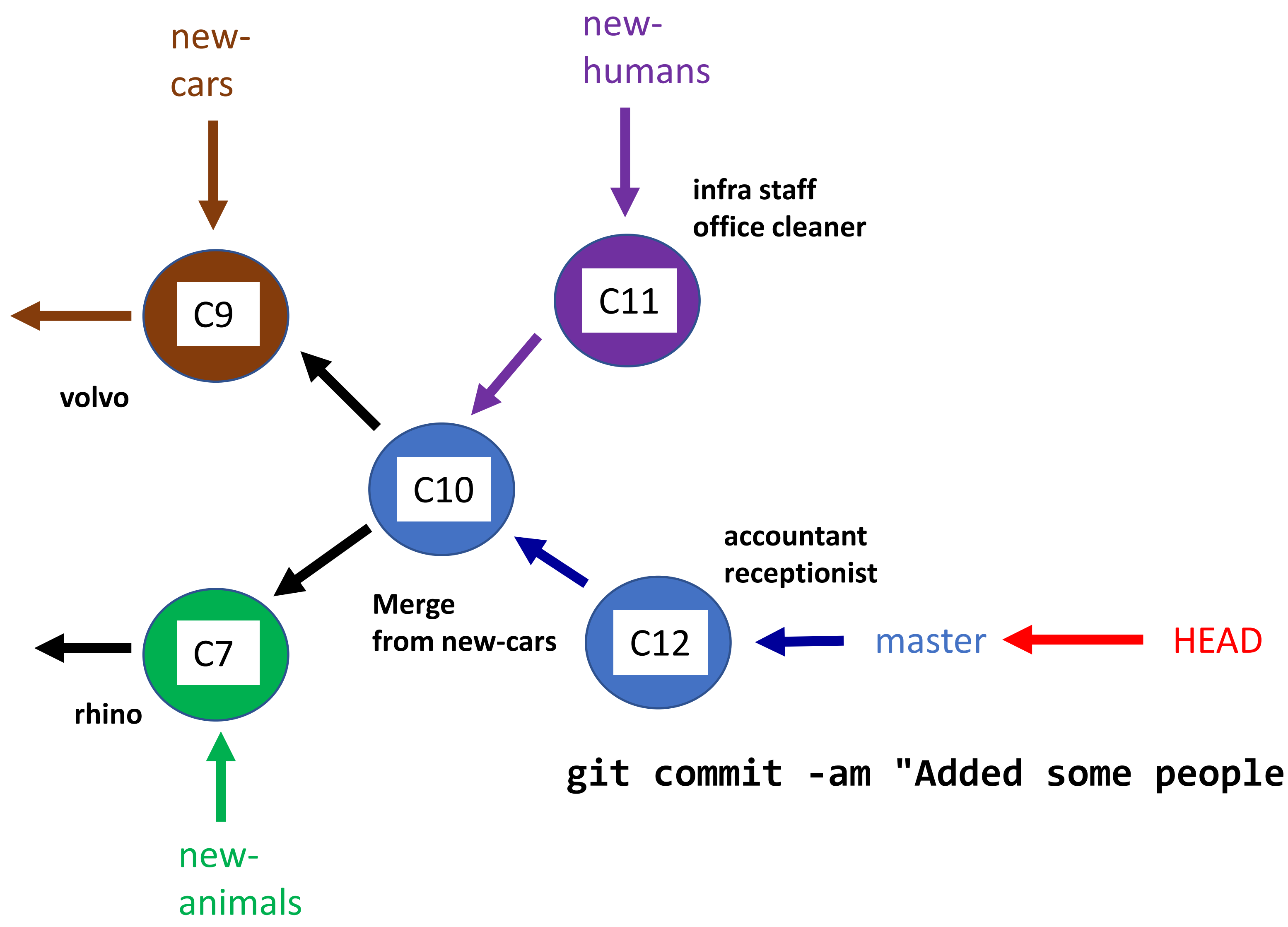
```
git switch -c new-humans
```



```
git commit -am "Added some folks from new-humans"
```



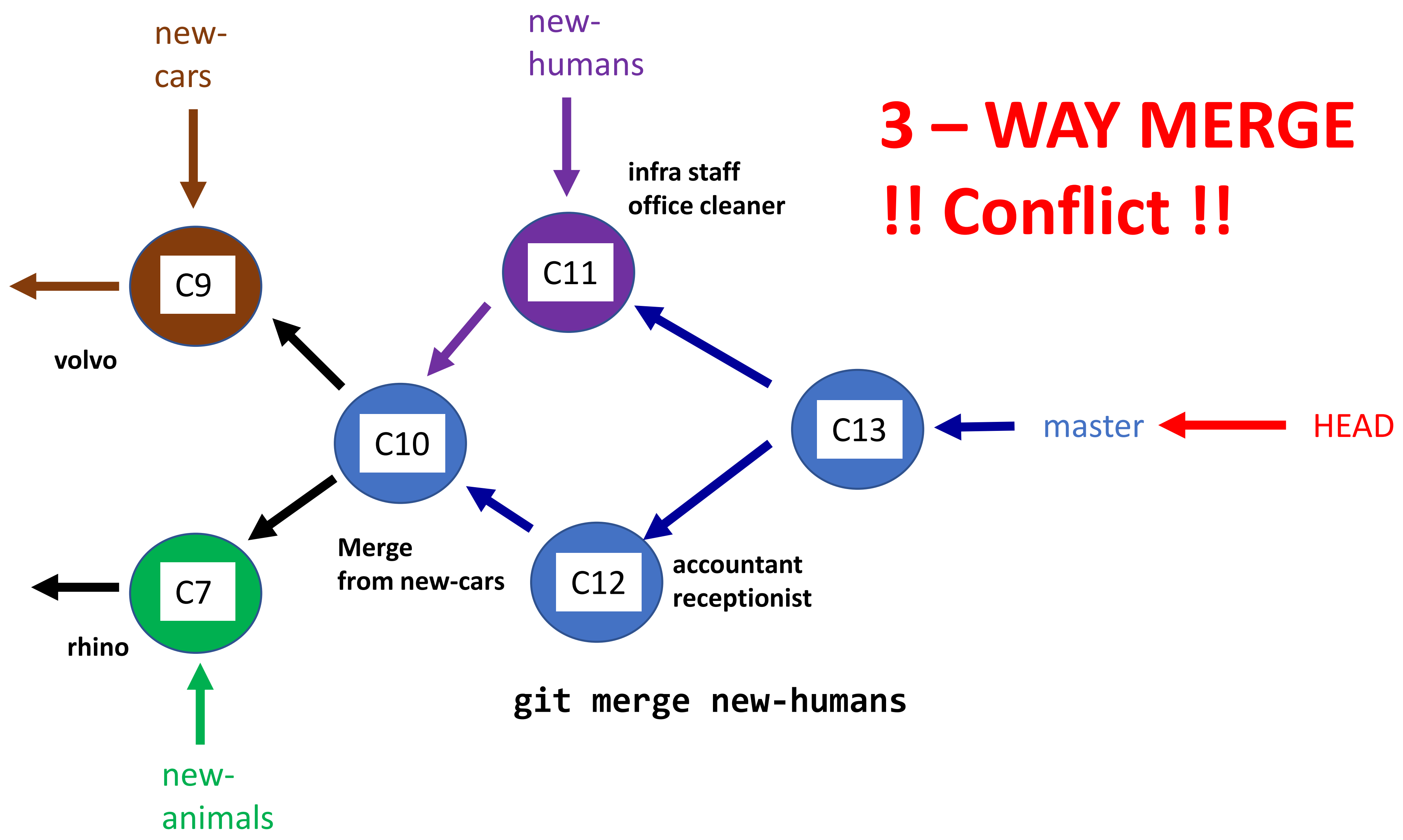
```
git switch master
```



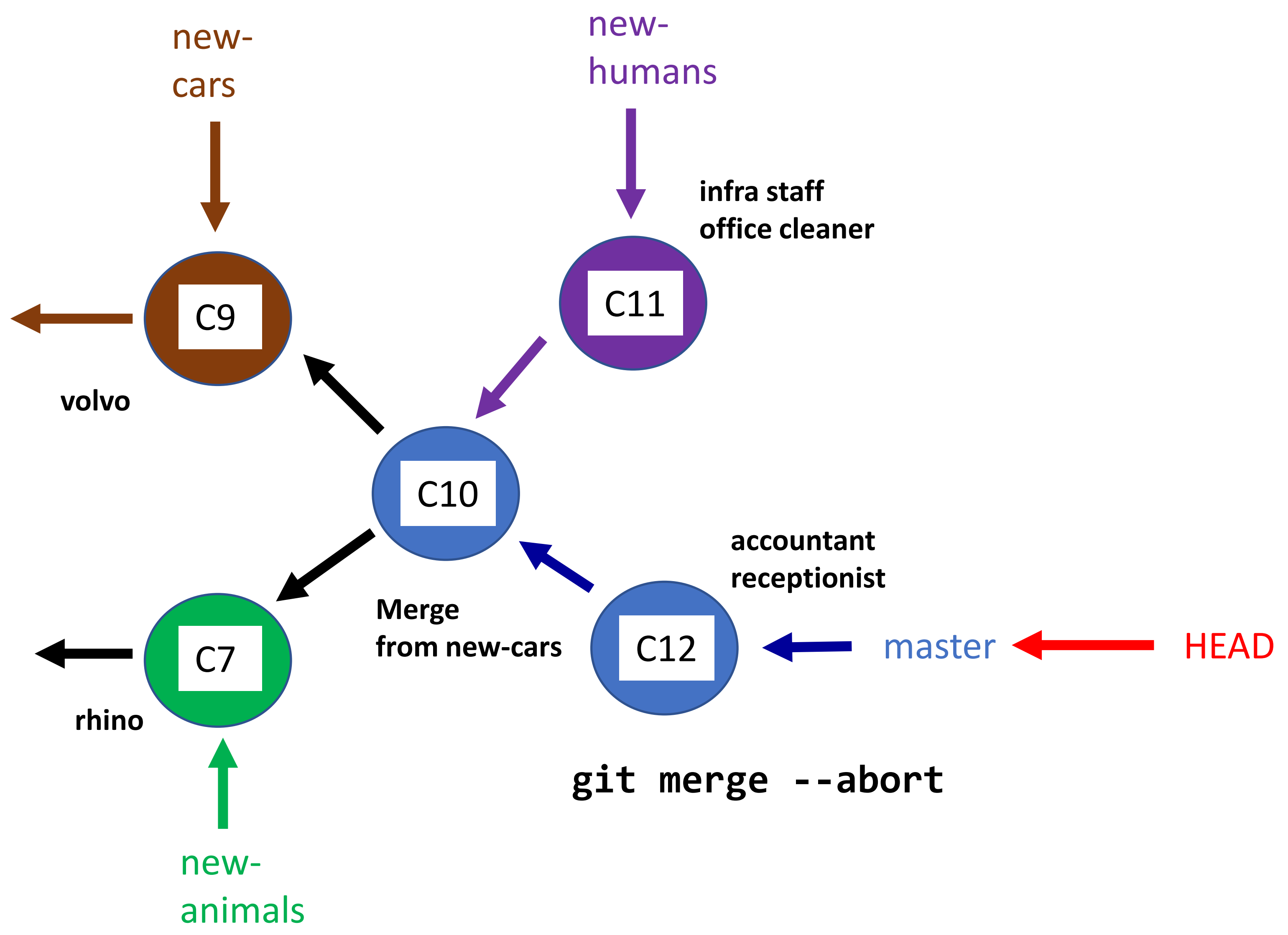
```
git commit -am "Added some people from master"
```

3 – WAY MERGE

!! Conflict !!



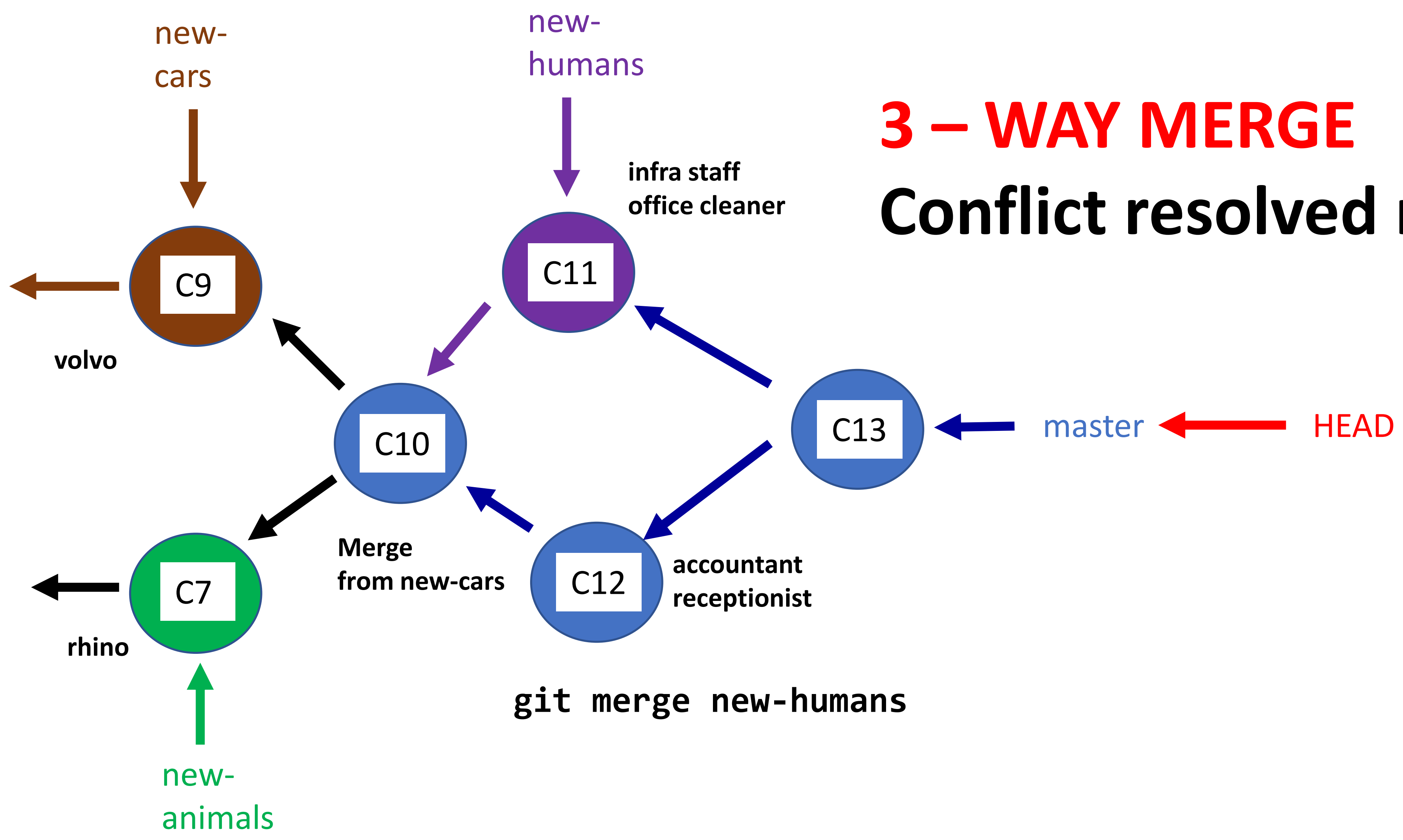
`git merge new-humans`



```
git merge --abort
```

3 – WAY MERGE

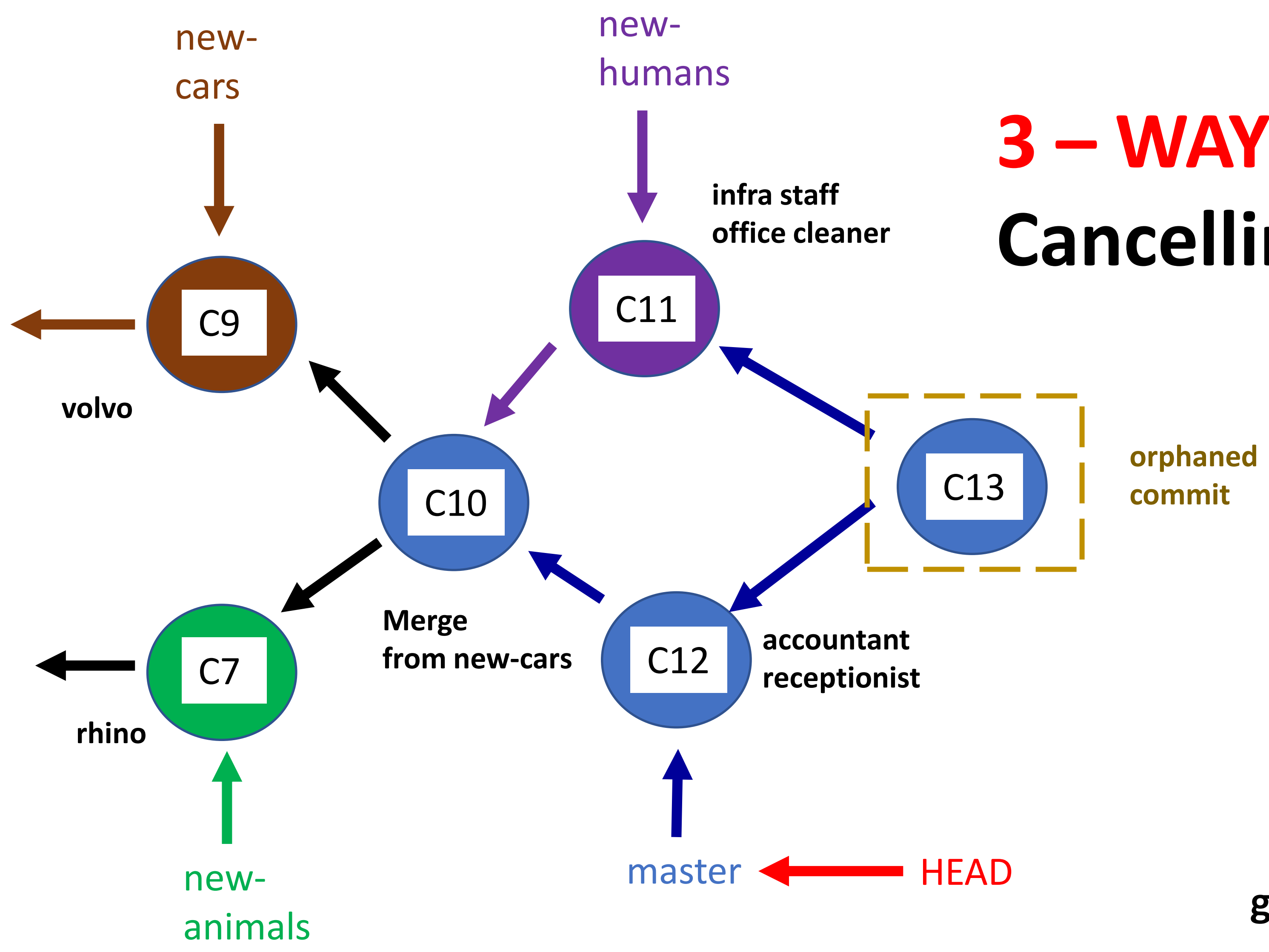
Conflict resolved manually



```
git merge new-humans
```

3 – WAY MERGE

Cancelling the merge



```
git reset --hard HEAD~1
```