# [320] Reproducibility (Dependencies)

Department of Computer Sciences
University of Wisconsin-Madison

# Reproducibility

**Big question:** *will my program run on someone else's computer?*

**Things to match:**

1. Hardware ← a program must fit the CPU; `python.exe` will do this, so `program.py` won't have to
2. Operating System ← we'll use Ubuntu Linux on virtual machines in the cloud
3. Dependencies ← today: versioning

# Dependency Versions

this program "depends" on pandas

program.py

```
import os, sys, json
import pandas

import pandas

print("Pandas Version:", pandas.__version__)

# code that uses pandas
```

you can check a module version

behavior depends on which release was installed

```
pip install pandas
```

**or**

```
pip install pandas==0.25.1
```

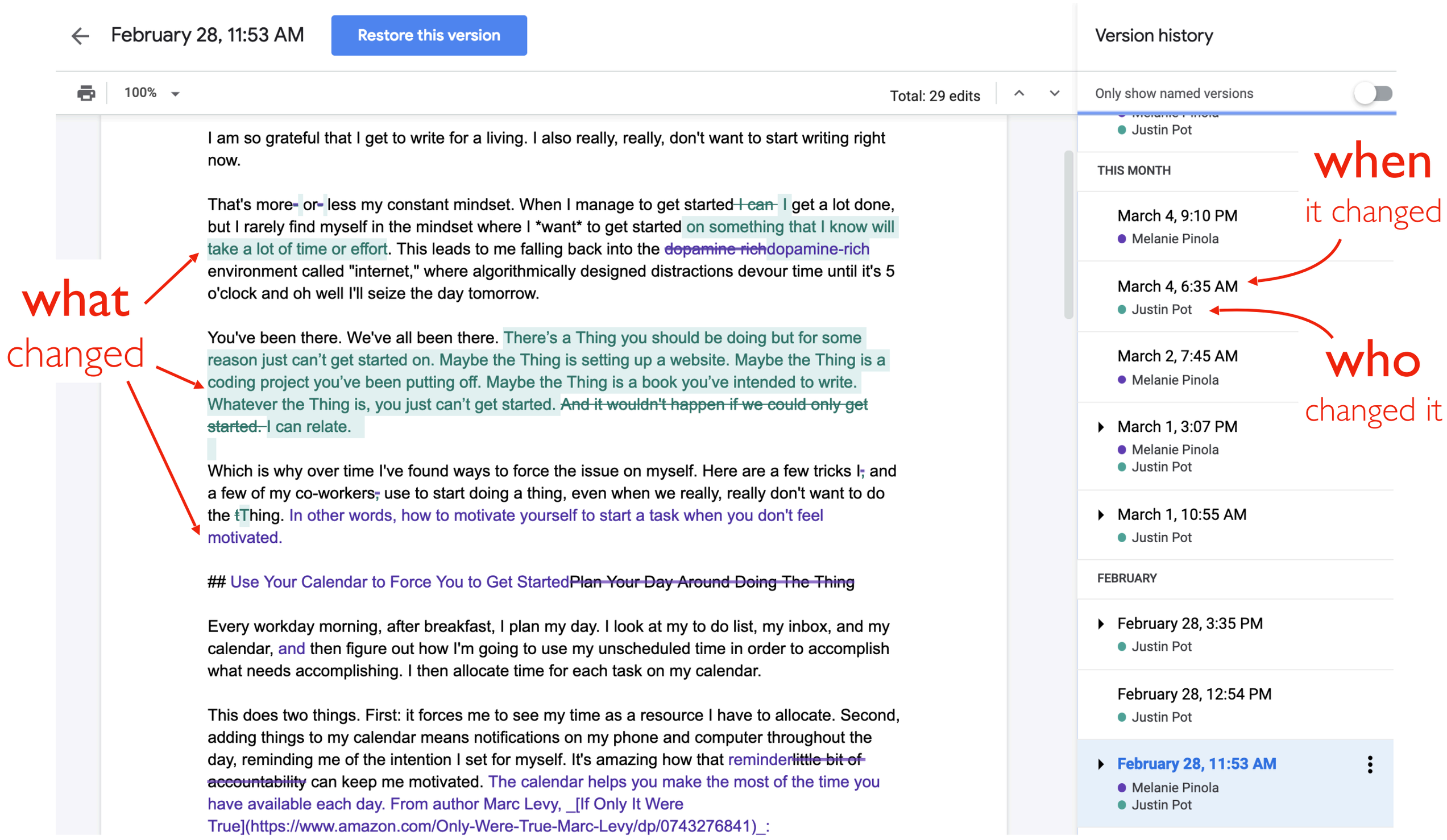
**or**

```
pip install pandas==0.24.0
```

**or...**

# Versioning: motivation and basic concepts

# Many tools auto-track history (e.g., Google Docs)




https://zapier.com/apps/google-docs/tutorials/google-docs-revision-history

# Version Control Systems (VCS)

**Useful for many kinds of projects**

- code, papers, websites, etc
- manages all files for same project (maybe thousands) in a repository

**Explicit snapshots/checkpoints, called *commits***

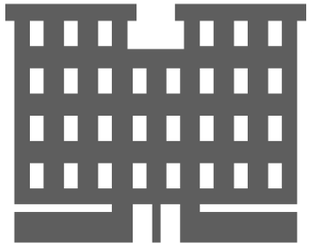
- users manually run commands to preserve good versions

Explicit commit messages

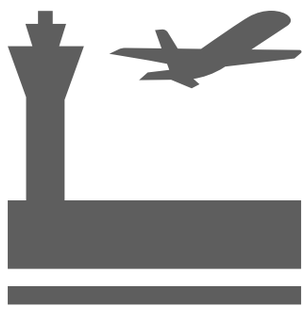
- who, what, when, why

**Work can *branch* out and be *merged* back**

- people can work offline
- can get feedback before merging
- humans need to resolve ***conflicts*** when versions being merged are too different



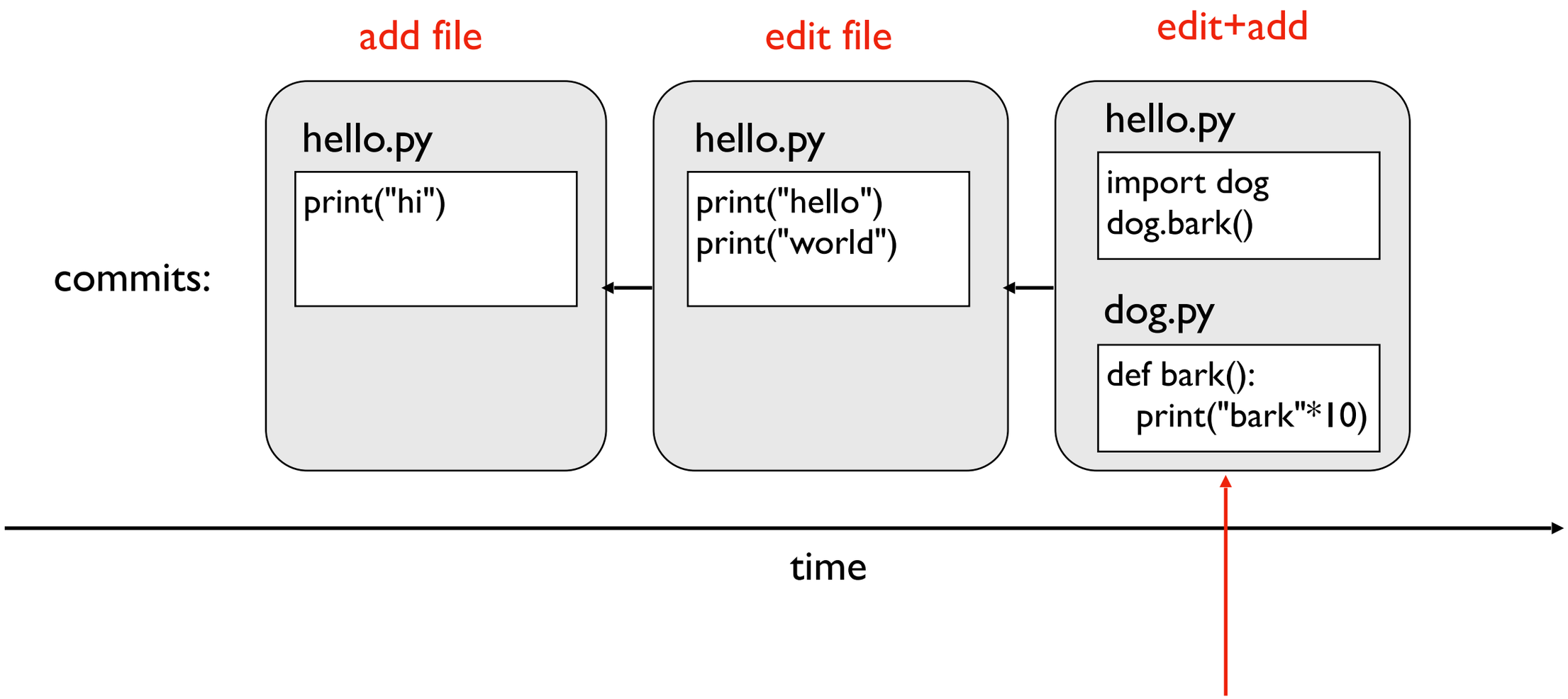
**partner A** working on `hw.py` at school



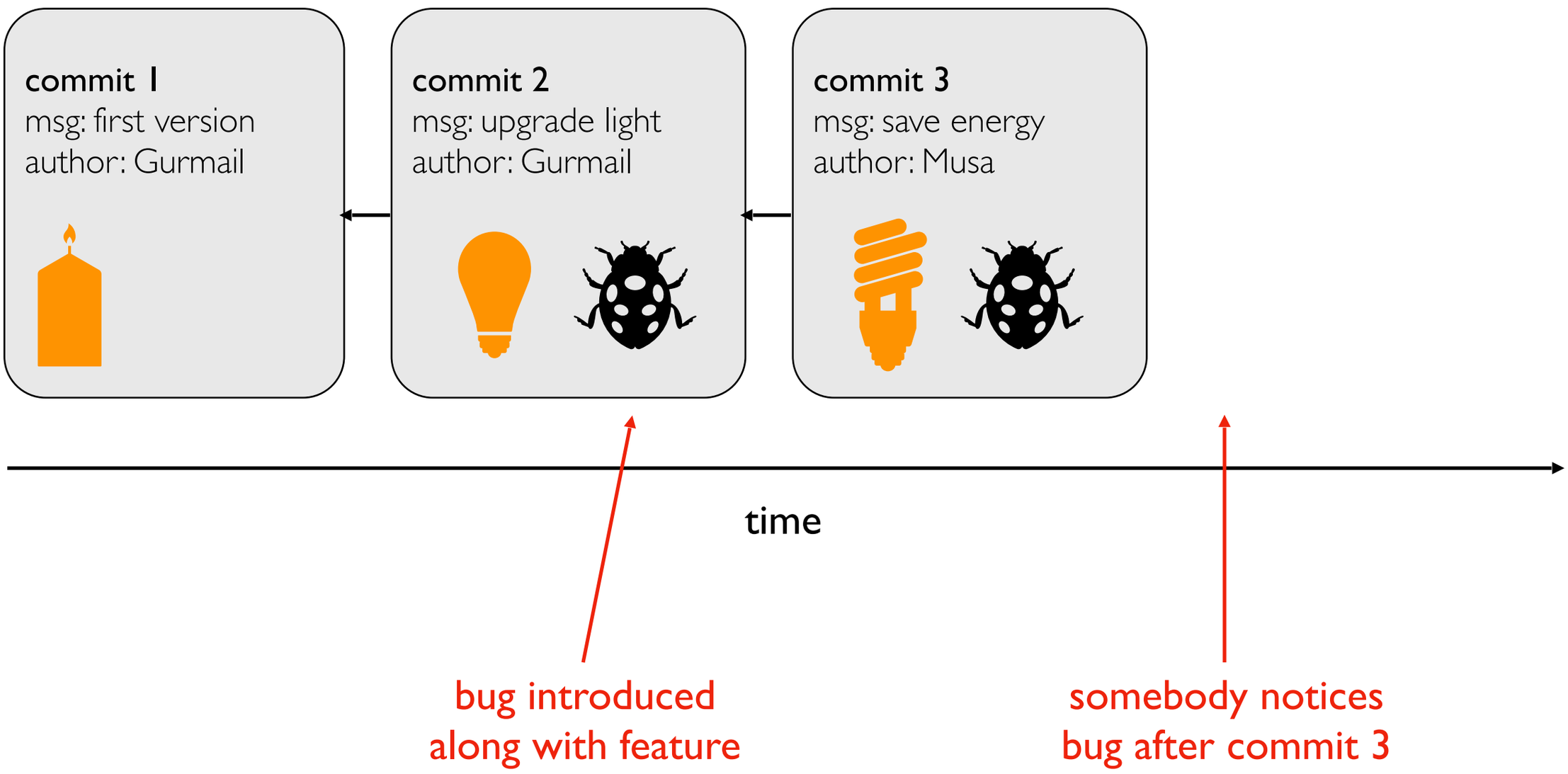
**partner B** also working on `hw.py`, without wifi

**what happens when the plane lands?**

# Example

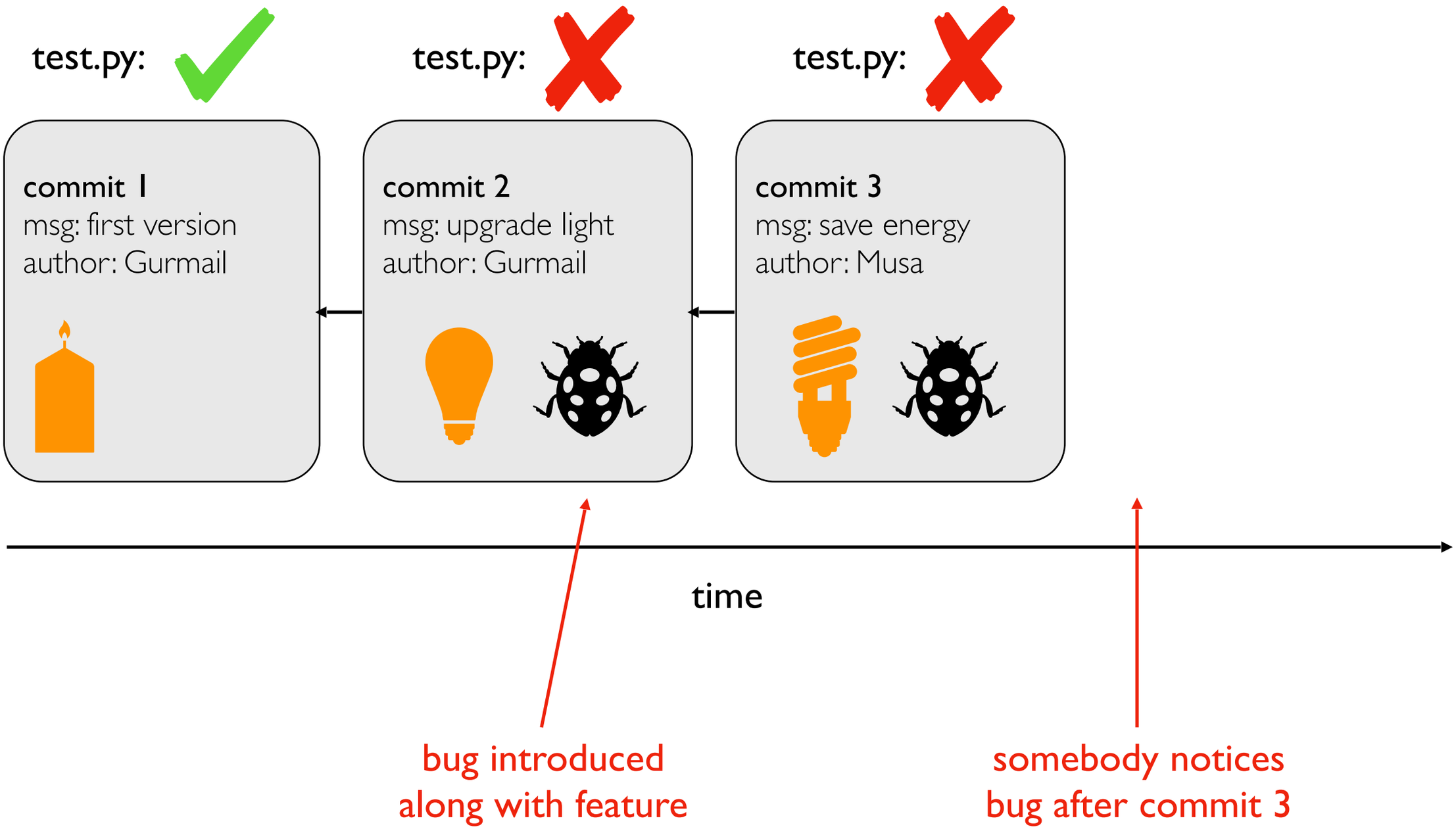

add file
edit file
edit+add
commits:
hello.py
print("hi")
hello.py
print("hello")
print("world")
hello.py
import dog
dog.bark()
dog.py
def bark():
print("bark"*10)
time
at any point in time,
you just see one version
of the files on your computer

# Use case 1: troubleshooting discovered bug

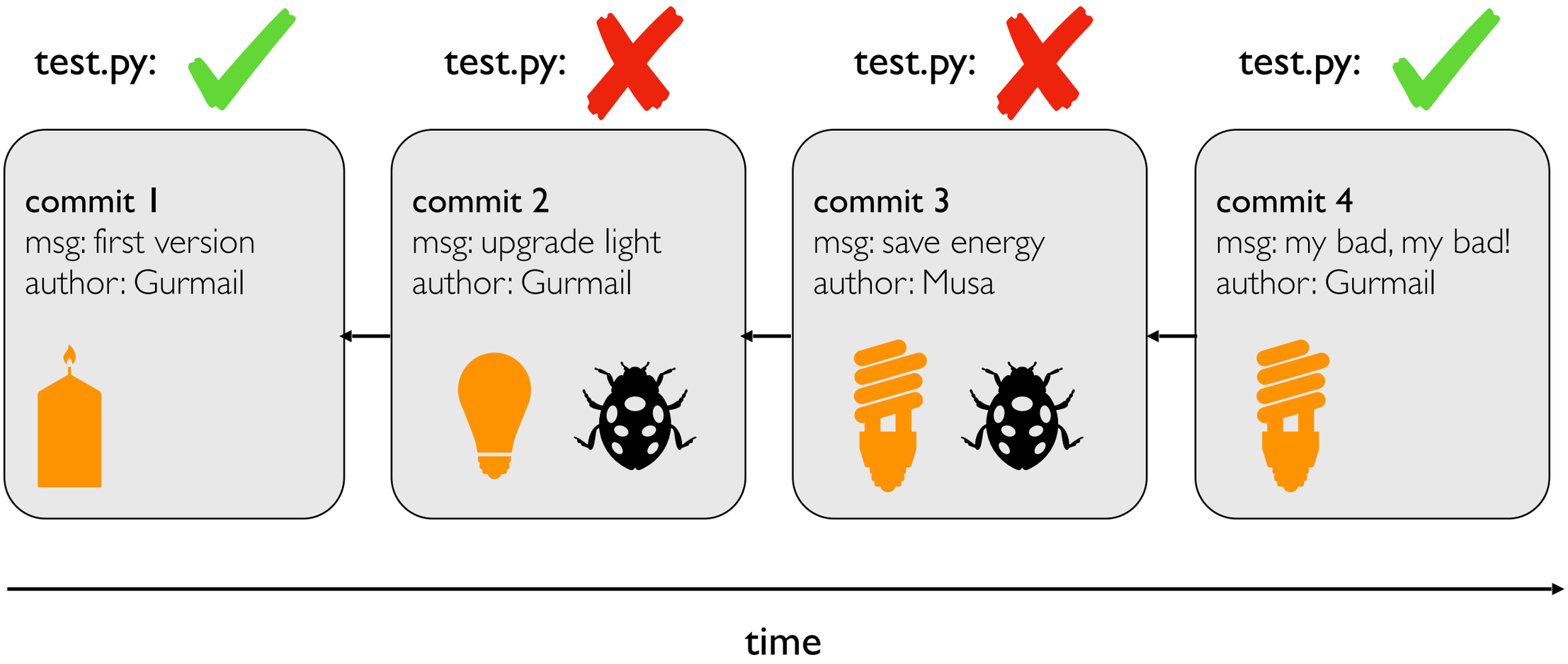



who will get blamed?

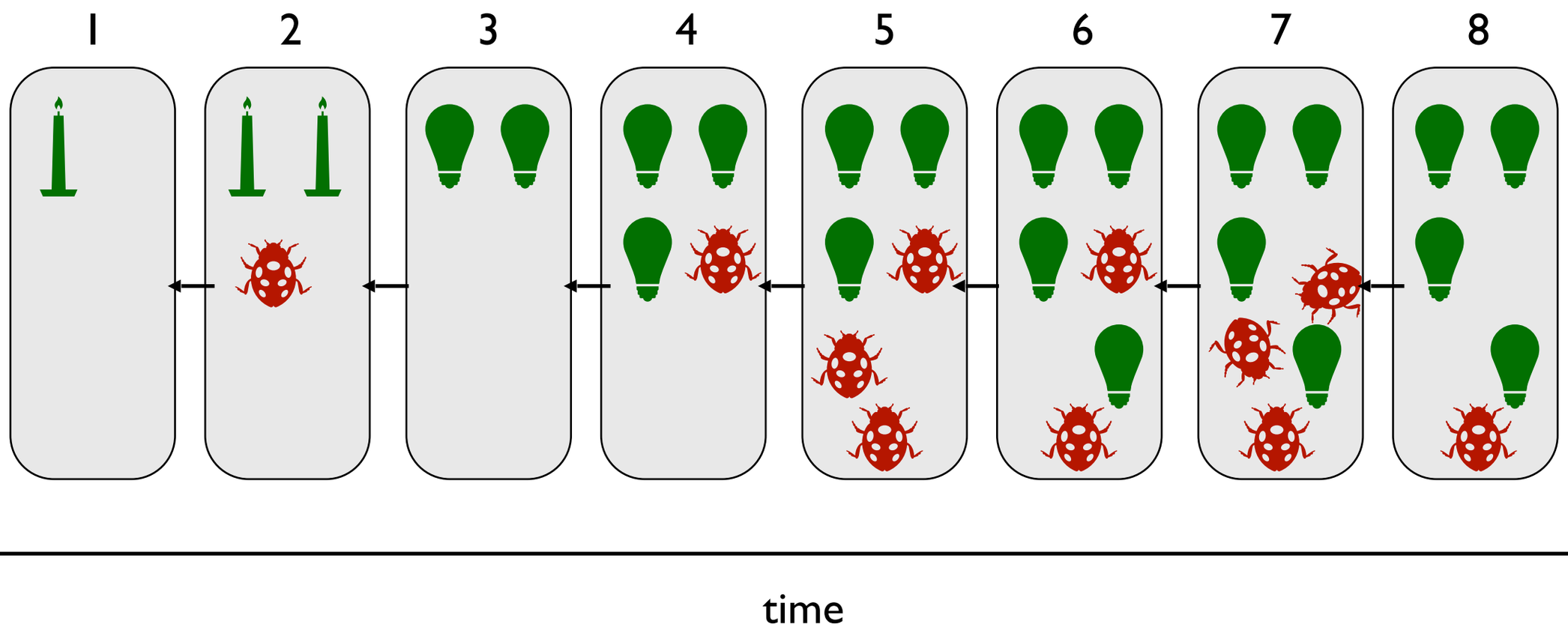
# Use case 1: troubleshooting discovered bug



# Use case 1: troubleshooting discovered bug

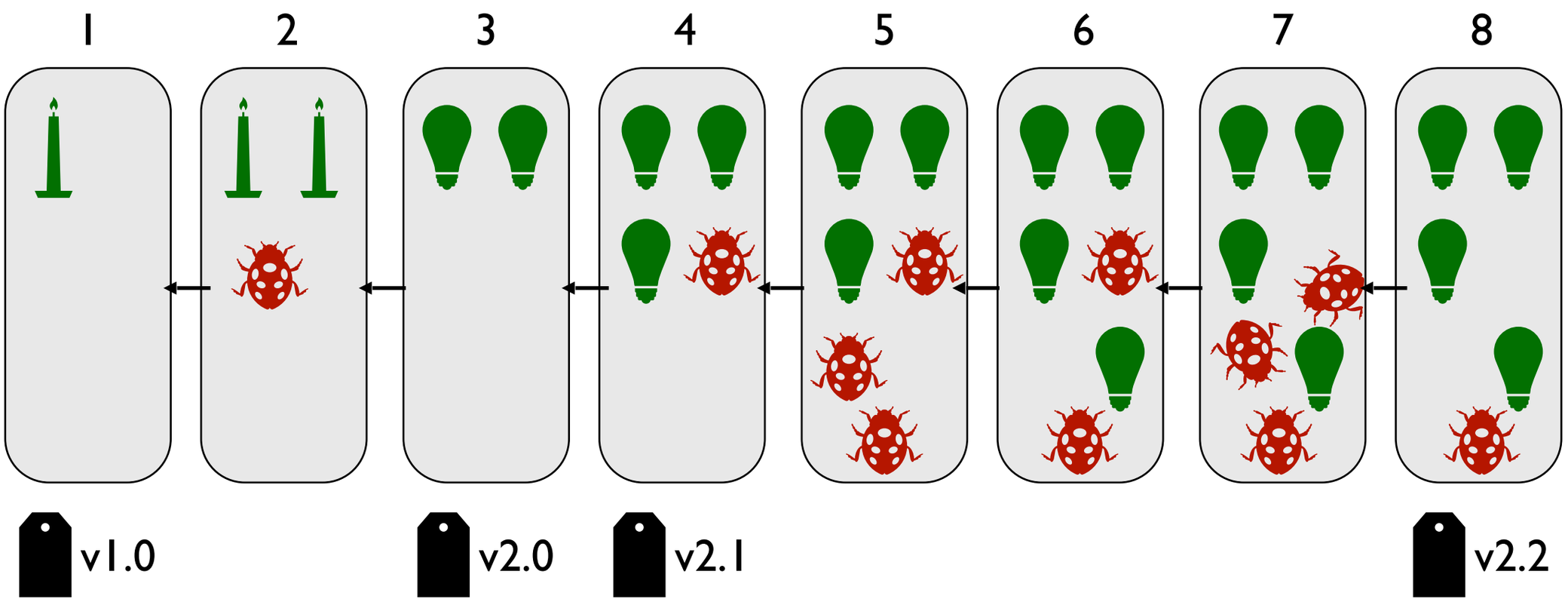


# Use case 2: versioned releases




which version would you use?

# Use case 2: versioned releases

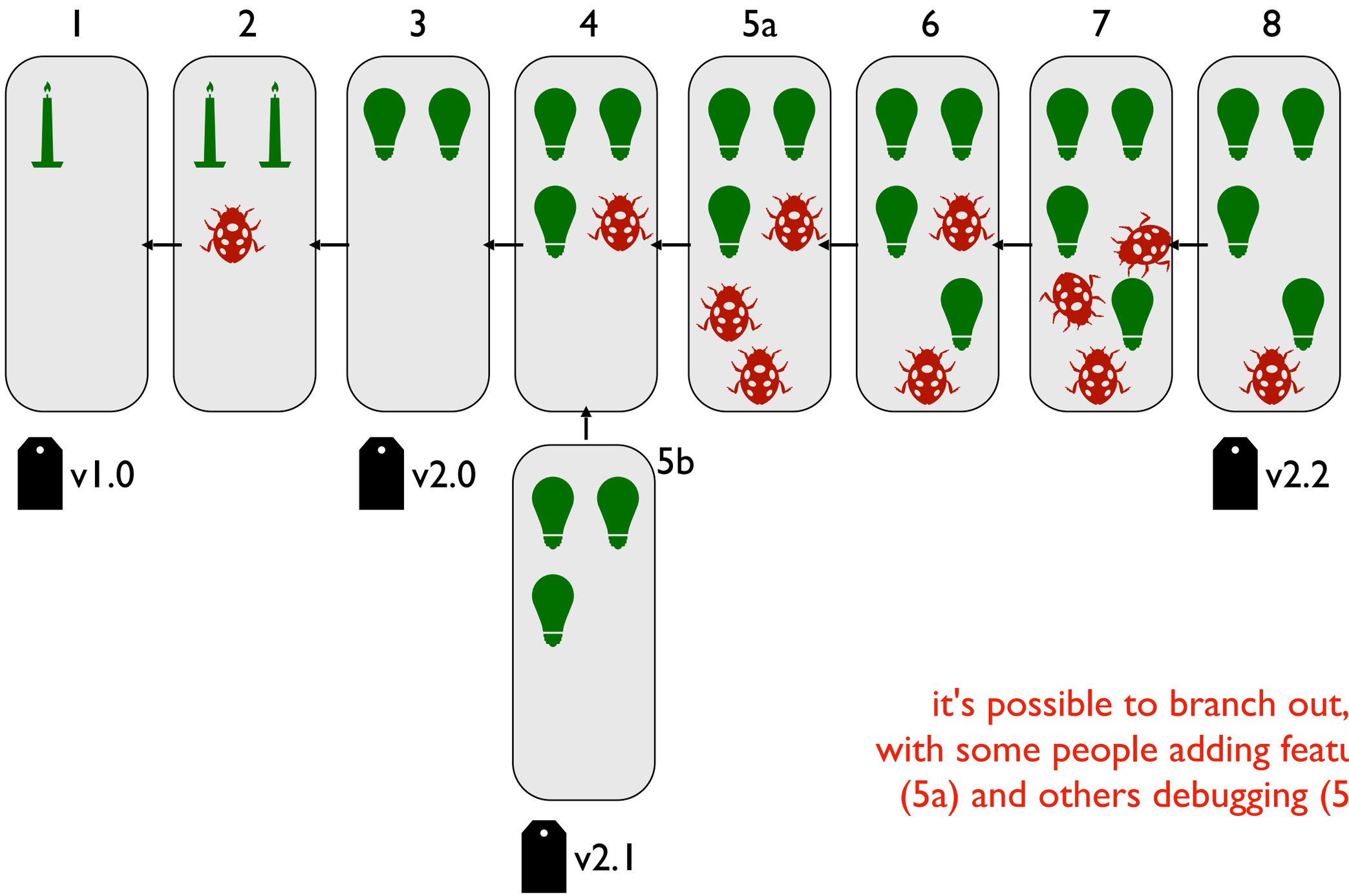



tag "good" commits to create releases

https://pypi.org/project/pandas/#history

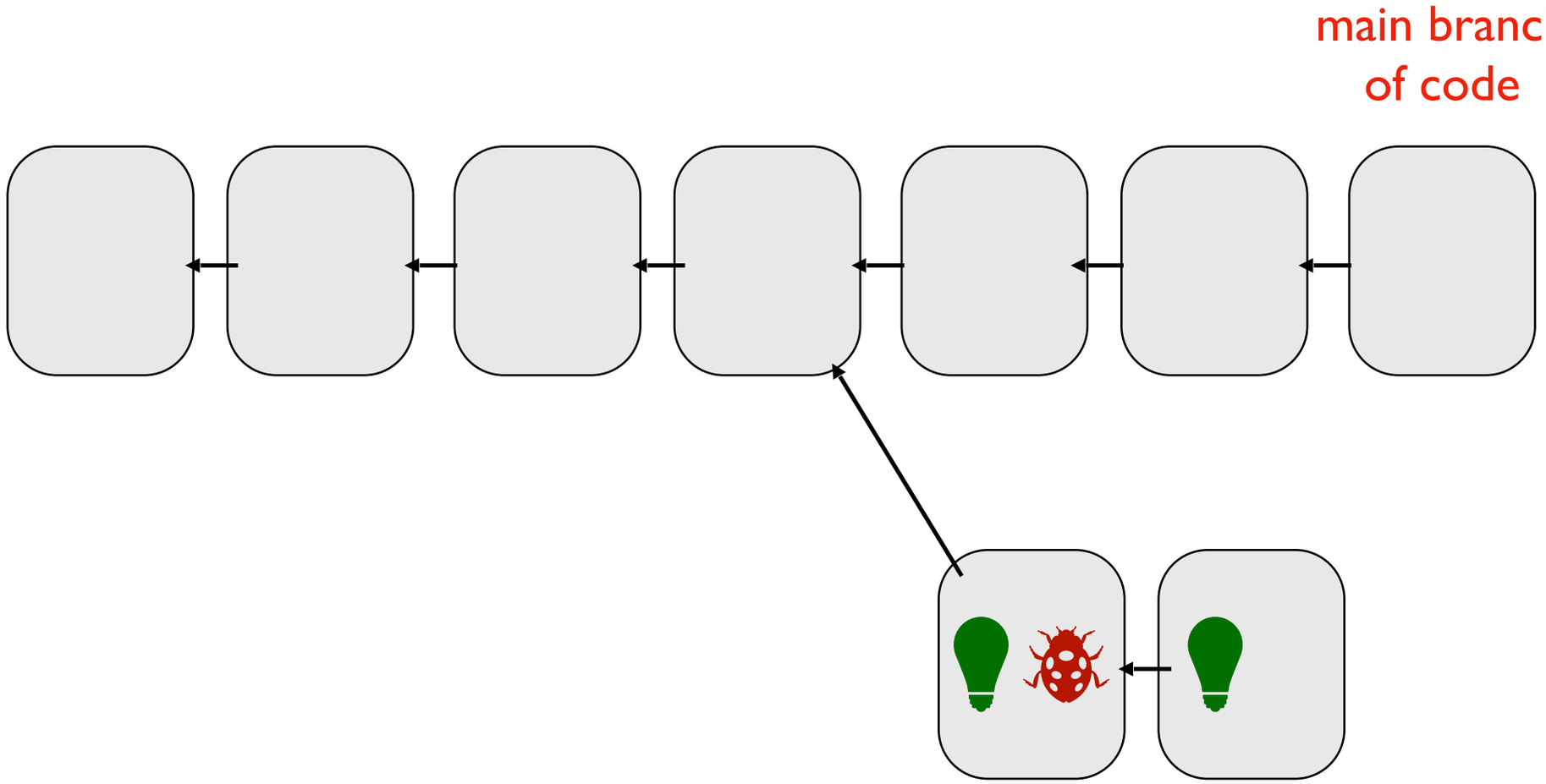
https://github.com/pandas-dev/pandas/releases

# Use case 2: versioned releases

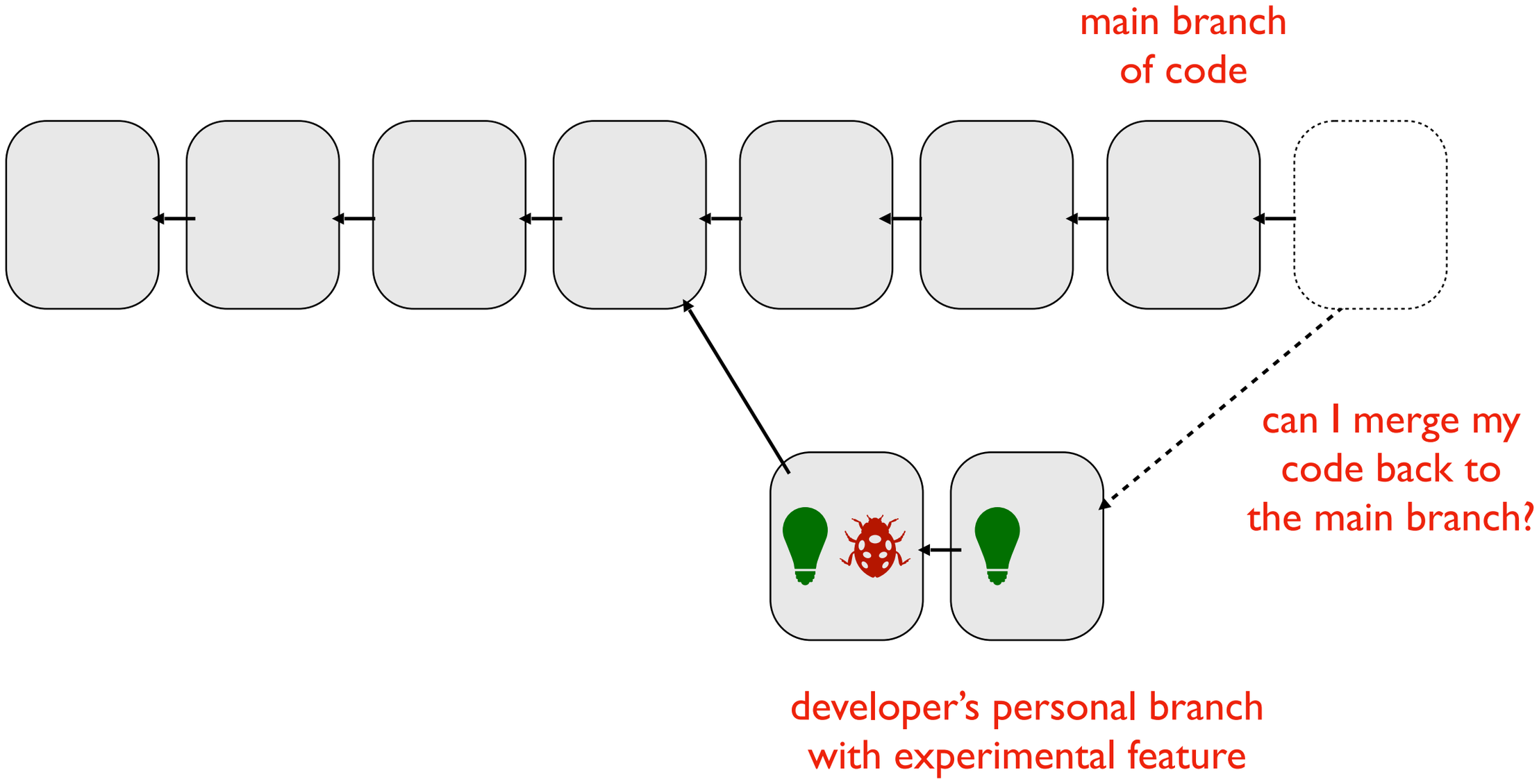



it's possible to branch out,
with some people adding features
(5a) and others debugging (5b)

# Use case 3: feedback



# Use case 3: feedback



# git

# Version Control System Tools

**tools**

svn

git

Mercurial

TeamFoundation

**git providers**

GitLab (don't delay, signup for a free GitLab account today, it is necessary for this class)

BitBucket

GitHub:

Linus Torvalds developed git to manage Linux as a BitKeeper replacement

https://www.linuxjournal.com/content/25-years-later-interview-linus-torvalds

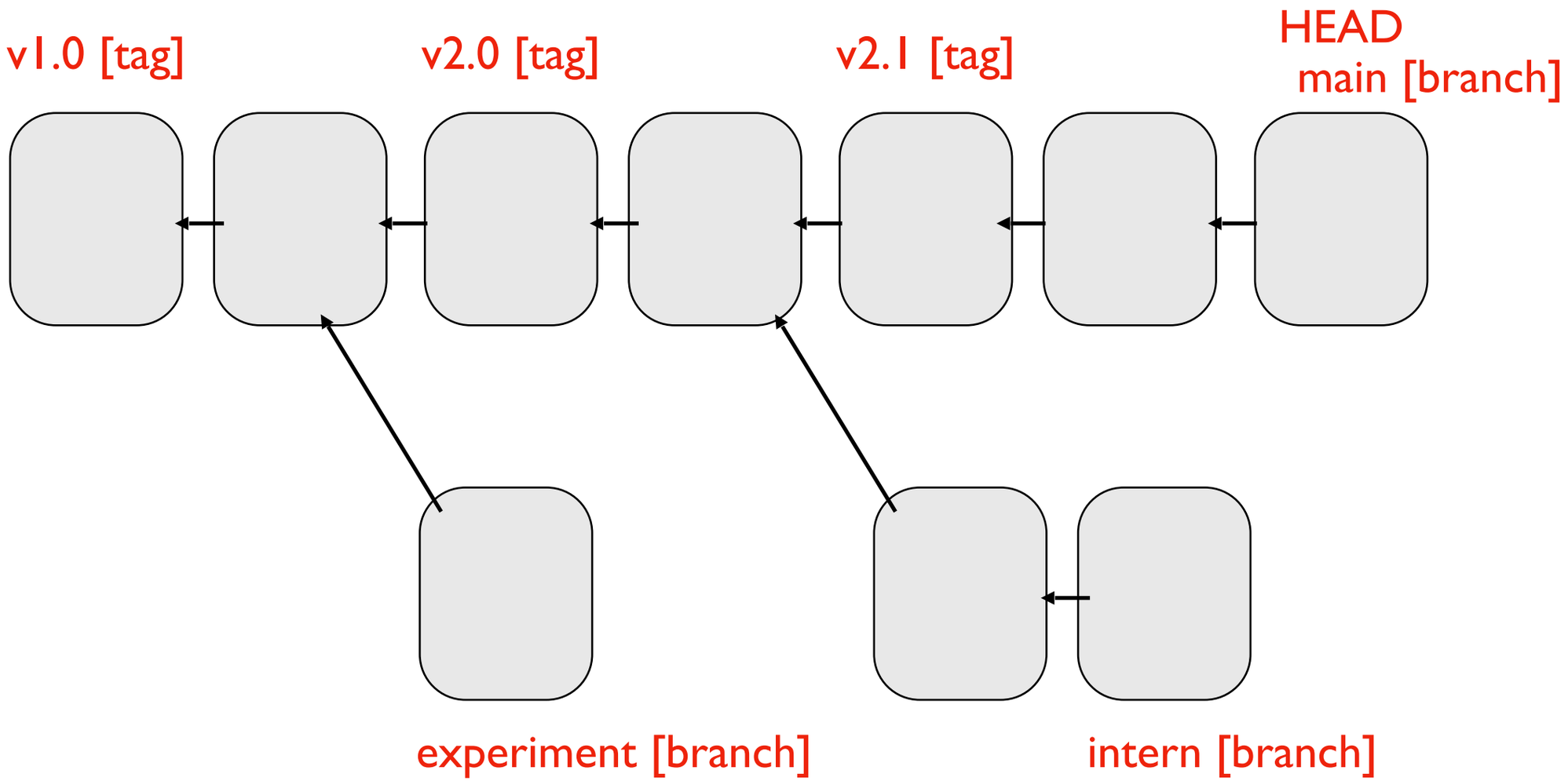
# Git Demos

Activities:

- connect to a VM via SSH
- copy ("clone") a GitLab repo to the VM
- view history
- switch between versions
- write ("commit") new versions

# HEAD, Branches, and Tags

**Remembering commit numbers is a pain! Various kinds of labels can serve as easy-to-remember aliases**




**HEAD:** wherever you currently are (only one of these)
**tag:** label tied to a specific commit number
**branch:** label tied to end of chain (moves upon new commits)

# HEAD, Branches, and Tags

## What branch are we on?

```
git branch
```

## Create new branch

```
git branch branchname
```

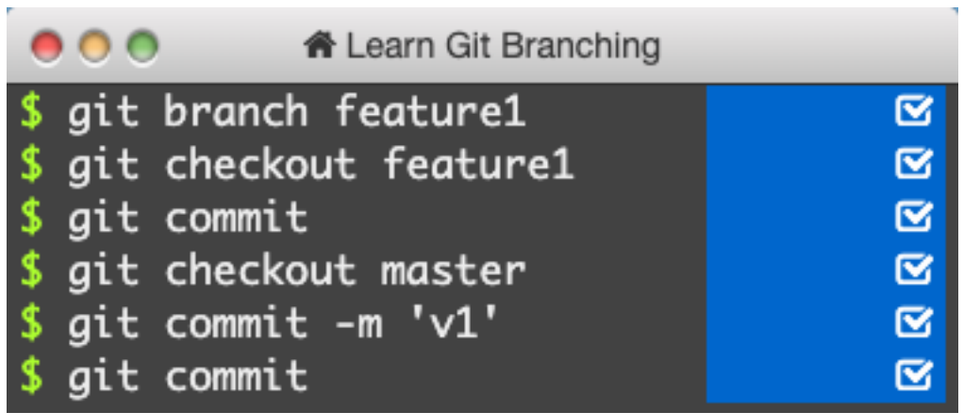
## Switch branch

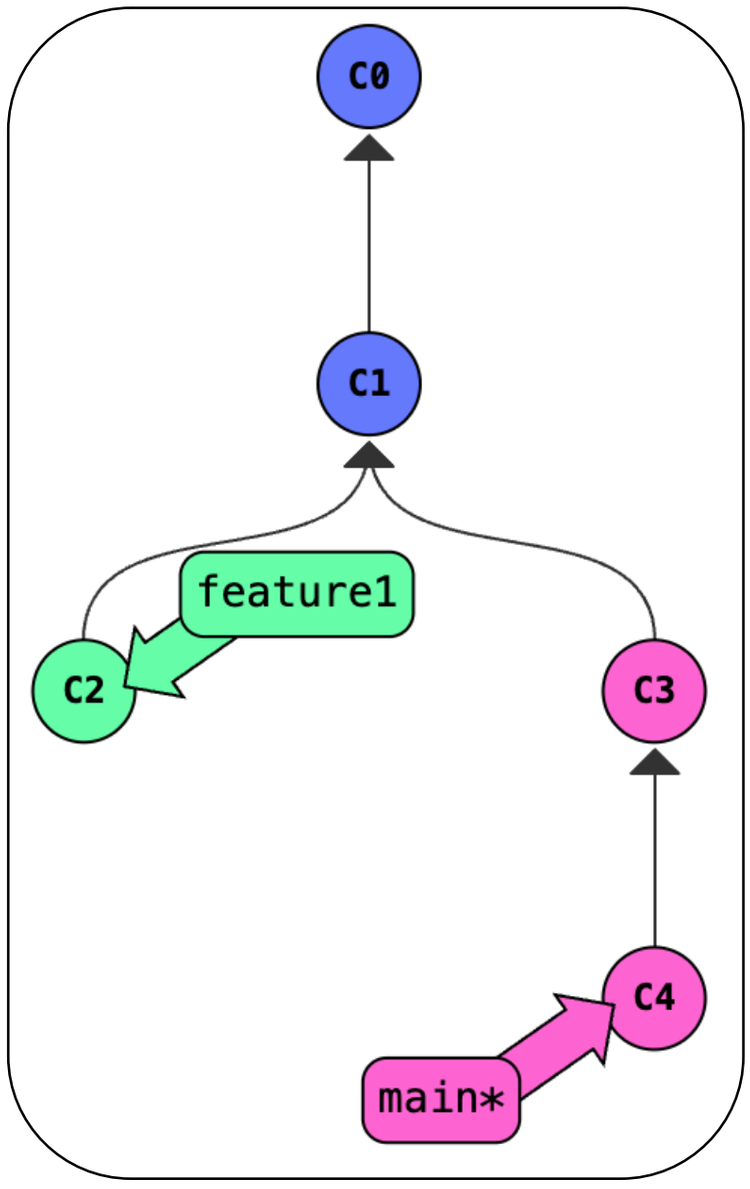
```
git checkout branchname
```

# Branching and Merging

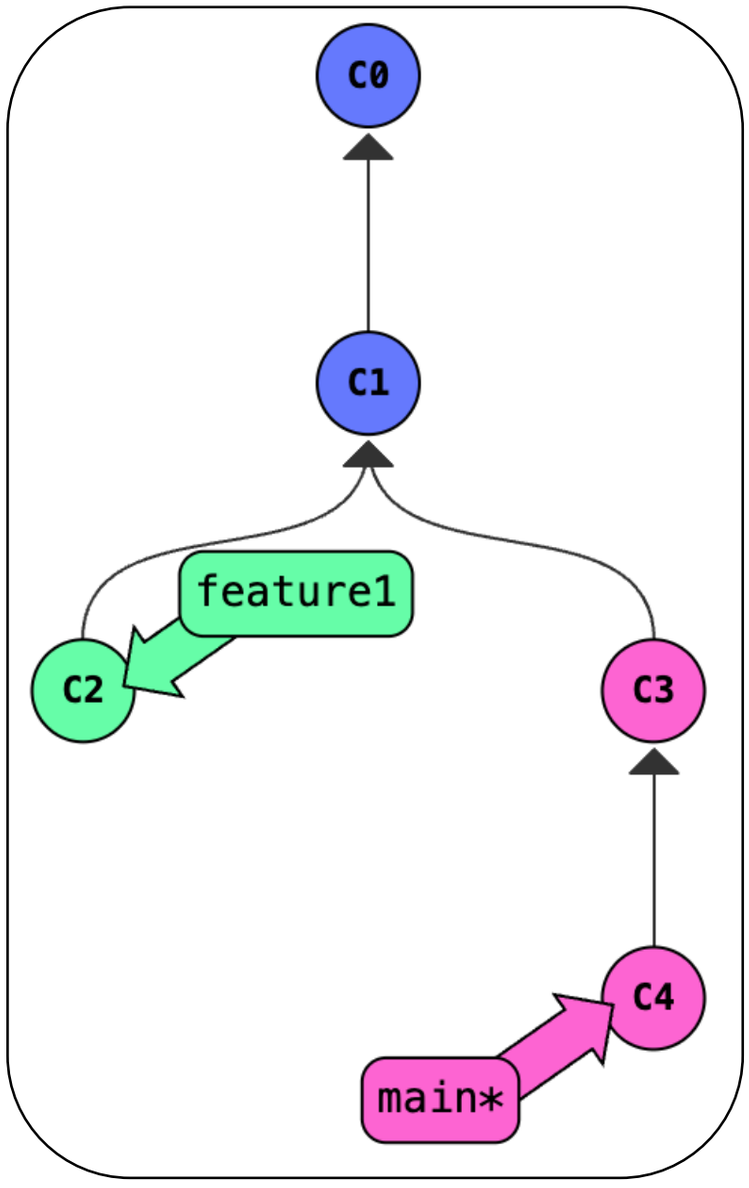
# Practice Branching

**Git equivalent of PythonTutor:**
https://learngitbranching.js.org/?NODEMO



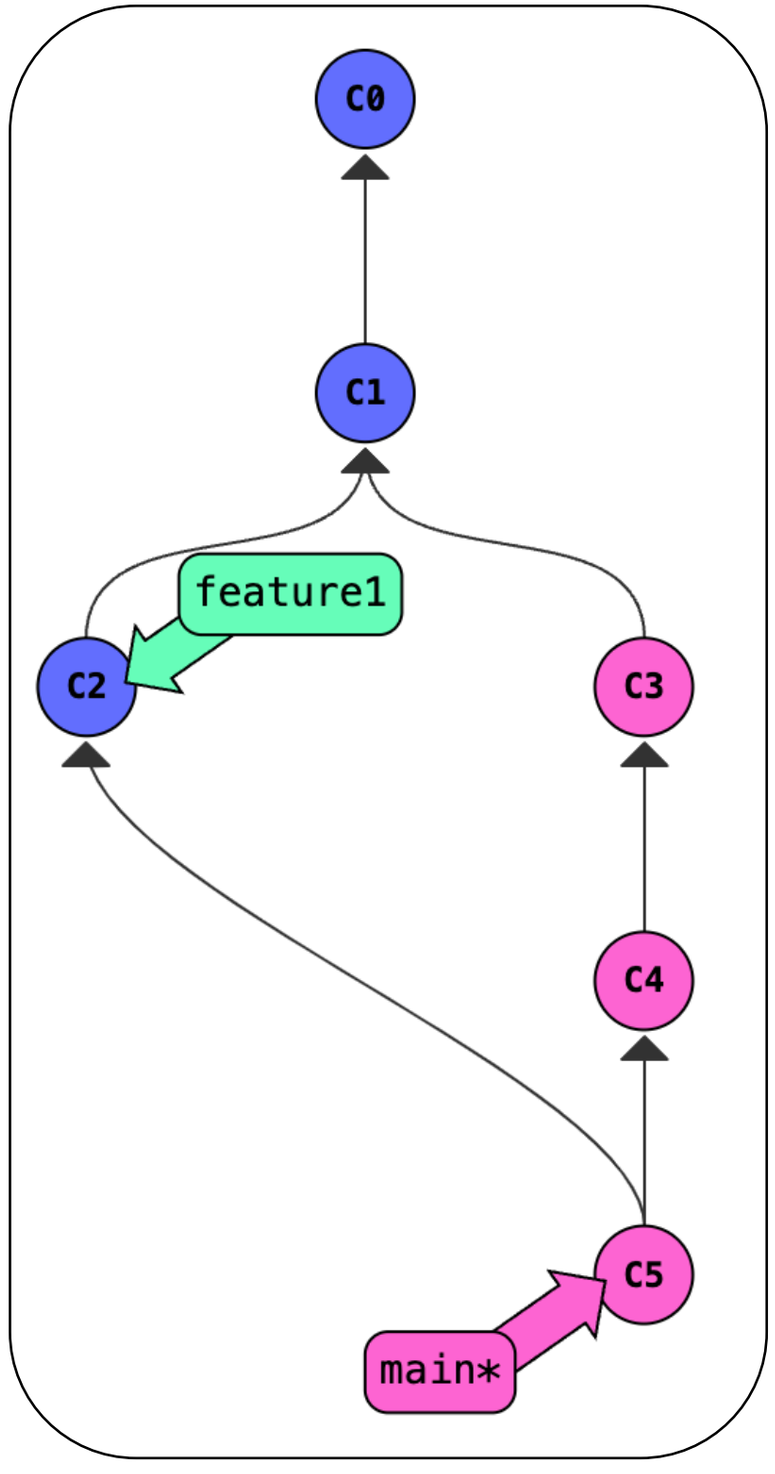
# Merging without Conflicts




## Switch branch

```
git merge frombranch
```

add whatever is there to the current branch

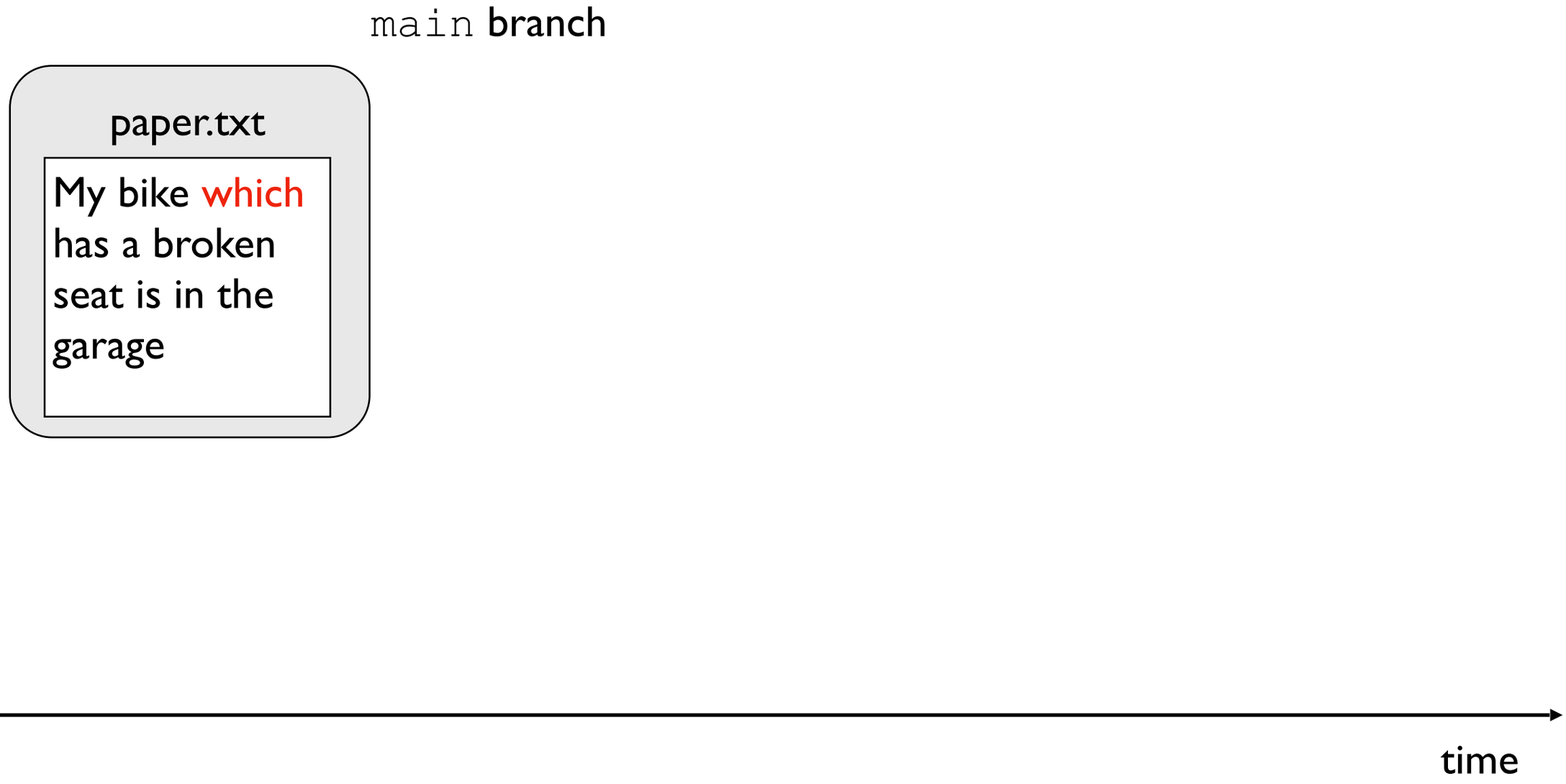



tip (or learn vim):

```
export EDITOR=nano
```

# Merging with Conflicts

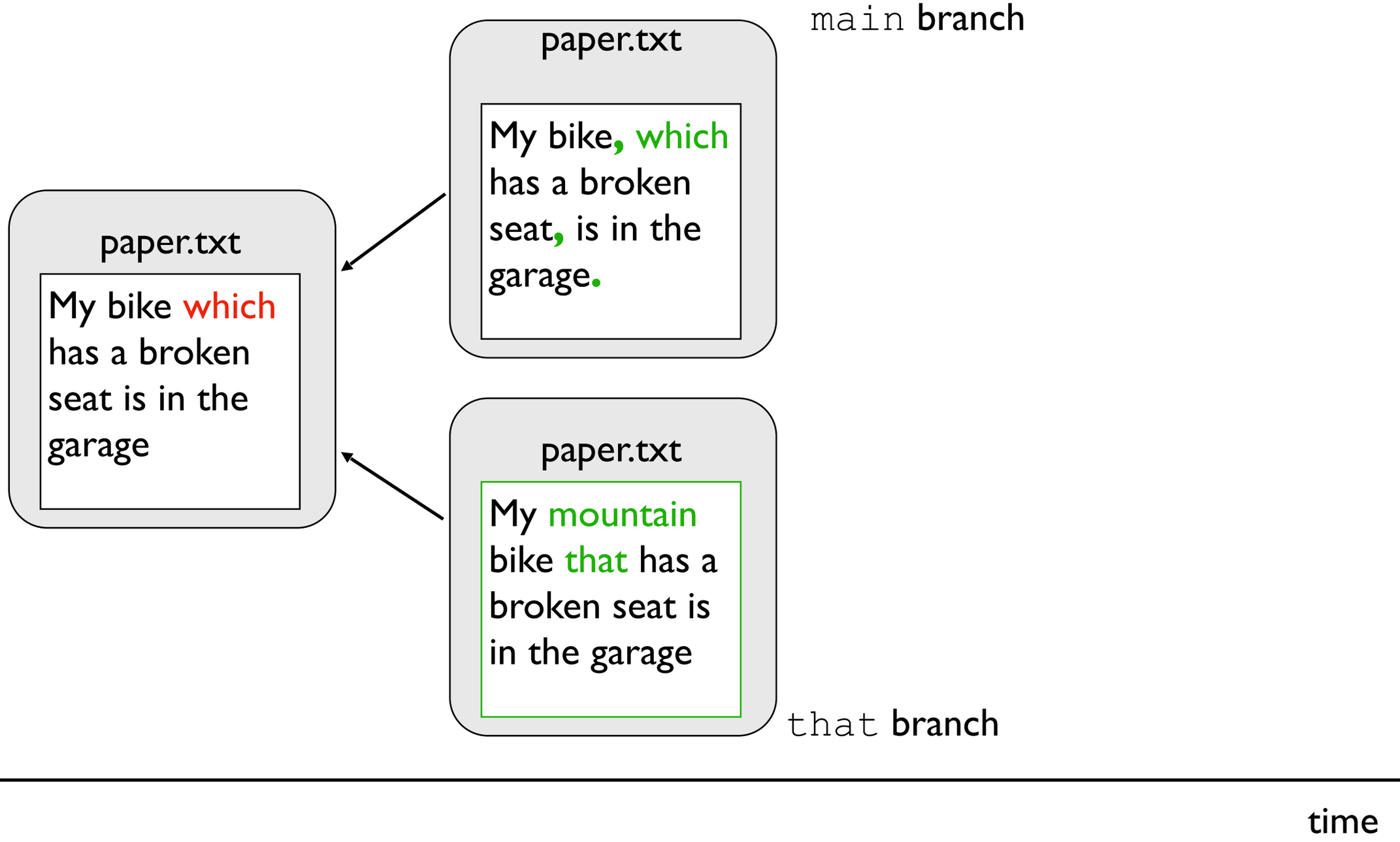
What happens when two people try to fix the same issue, in two different (incompatible) ways?




https://www.grammarly.com/blog/which-vs-that/

# Merging with Conflicts

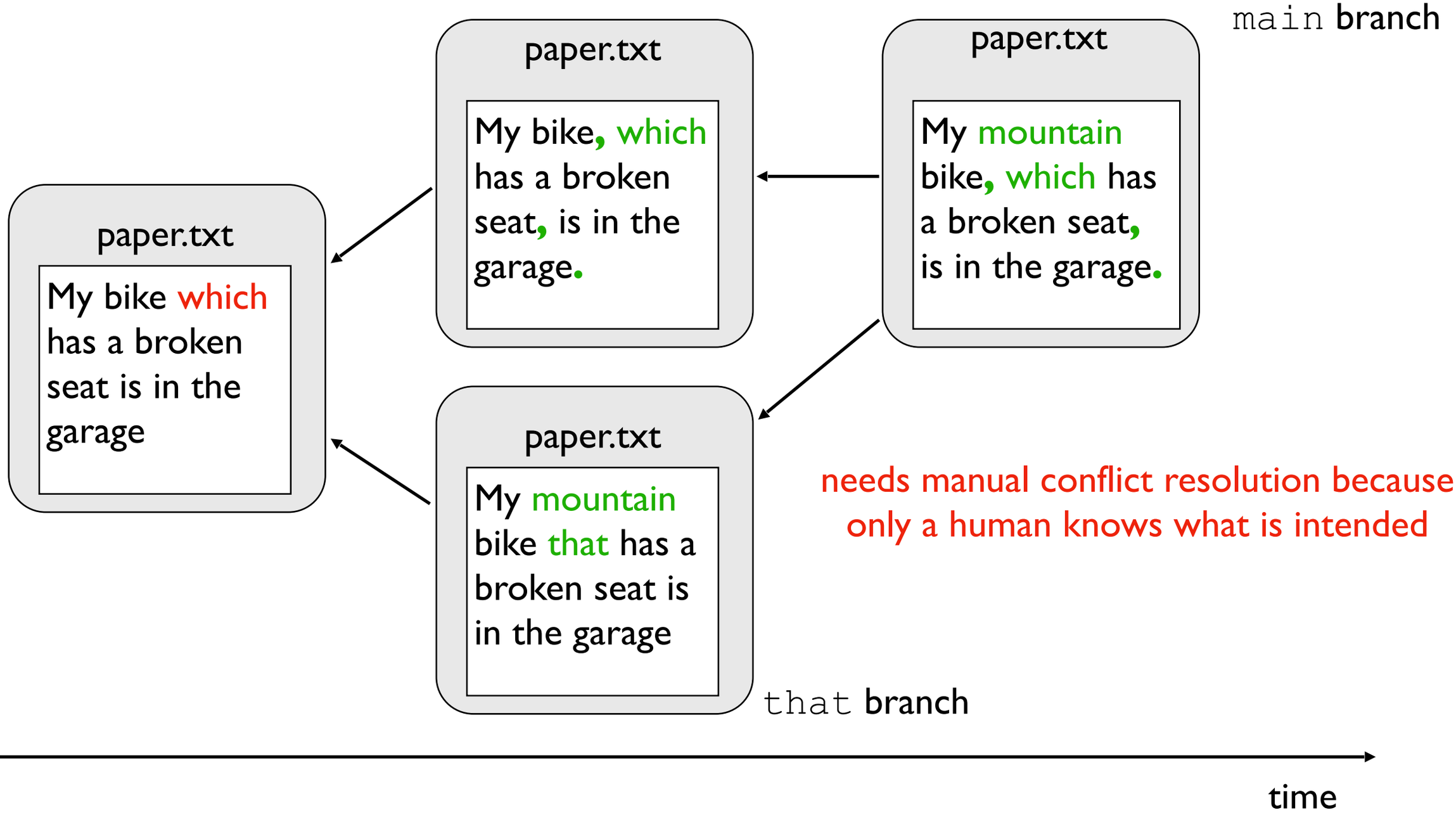
What happens when two people try to fix the same issue, in two different (incompatible) ways?




https://www.grammarly.com/blog/which-vs-that/

# Merging with Conflicts

What happens when two people try to fix the same issue, in two different (incompatible) ways?




https://www.grammarly.com/blog/which-vs-that/

# Summary of Terms

commit: a snapshot of files at a point in time

HEAD: a convenient label for the current commit

tag: a long-term label associated with a commit

branch: a label attached to a commit that re-attaches to new commits

merge: to combine changes on another branch into the current branch

conflict: differences that cannot automatically be merged

# Challenges (figure out how to reproduce):