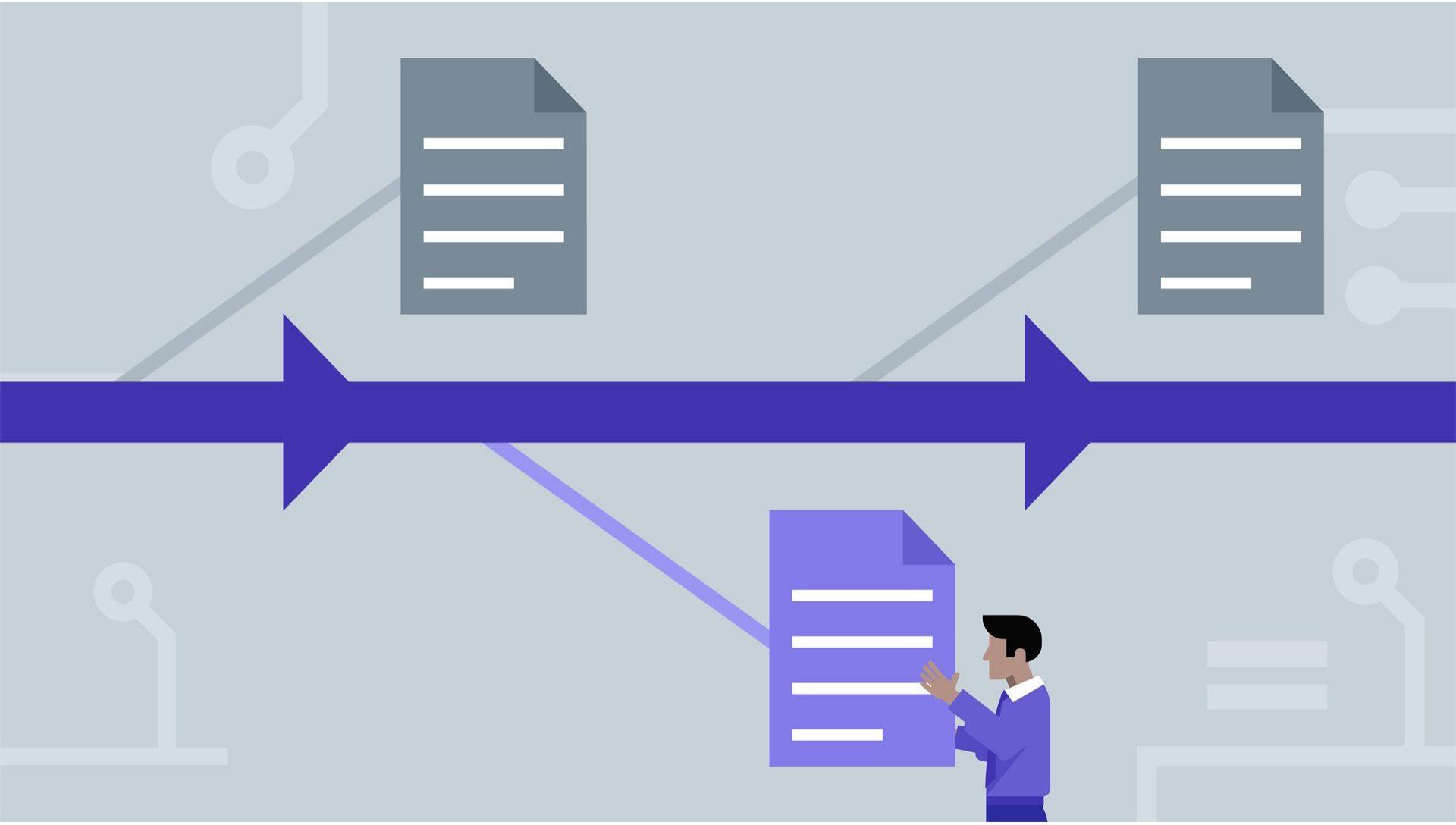
Collaborating on Code

Agenda

- Version Control
- Development Process
- Licensing

Version Control



Version Control

- Deal with software changes
- Track who did what
- Find a particular change (version)
- Manage multiple releases

What is Git?

Git is a *distributed* version control system

Stores revisions of files over time

Four problems to motivate you

Change Control

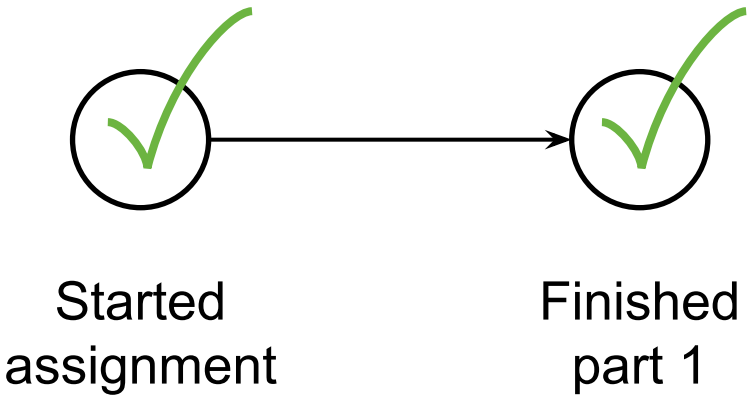
Someone else changed something.

What did they do?

Everything's suddenly broken.

What changed?

Changes are Discrete

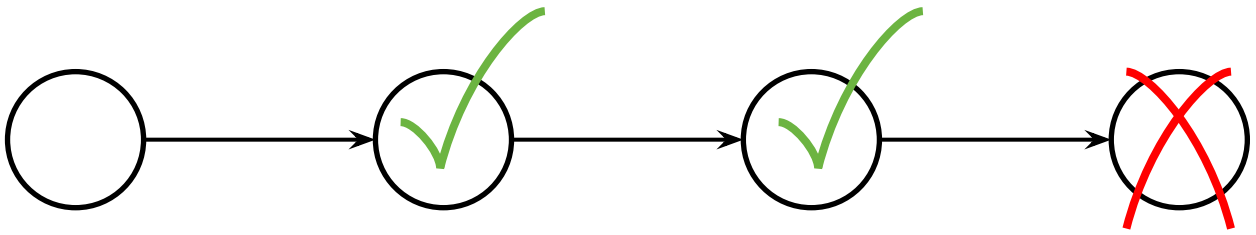


Reversion

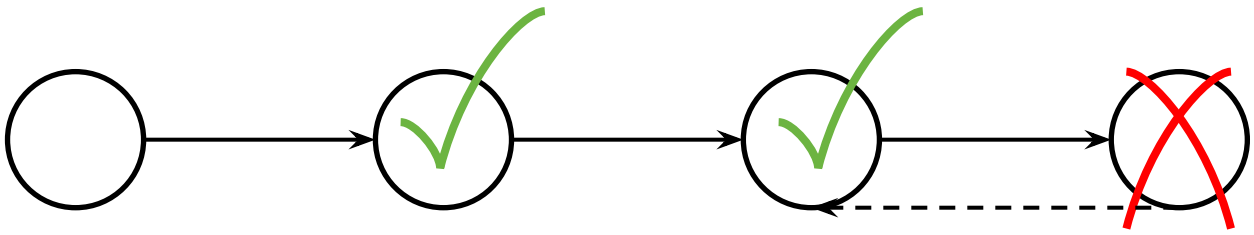
What if the change I’m about to make turns out not to be a good idea?

How can I undo those changes?

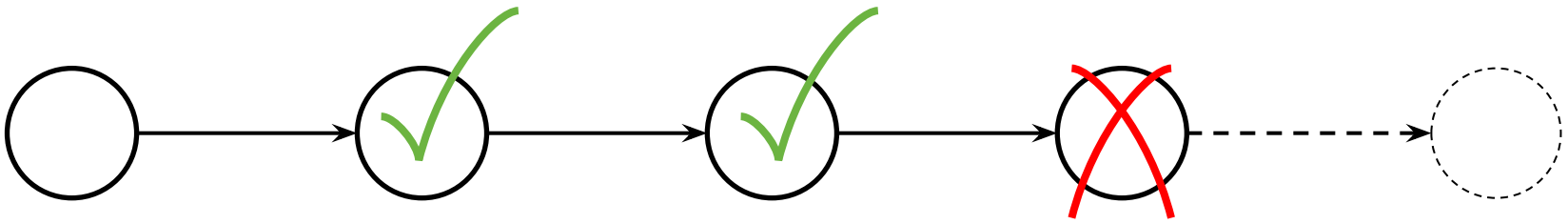
Reversion



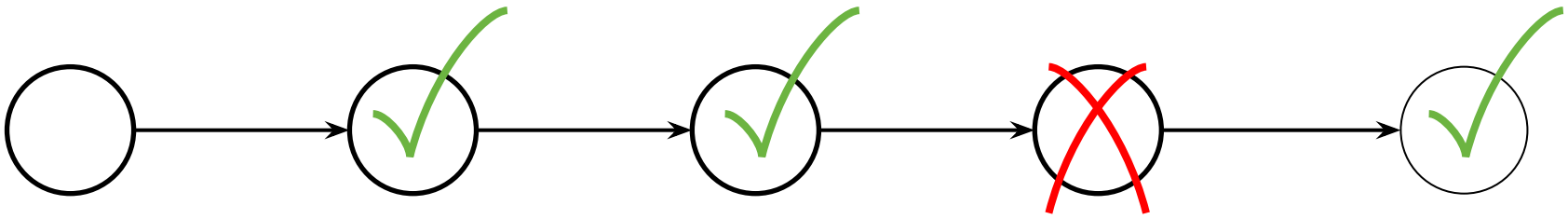
Reversion



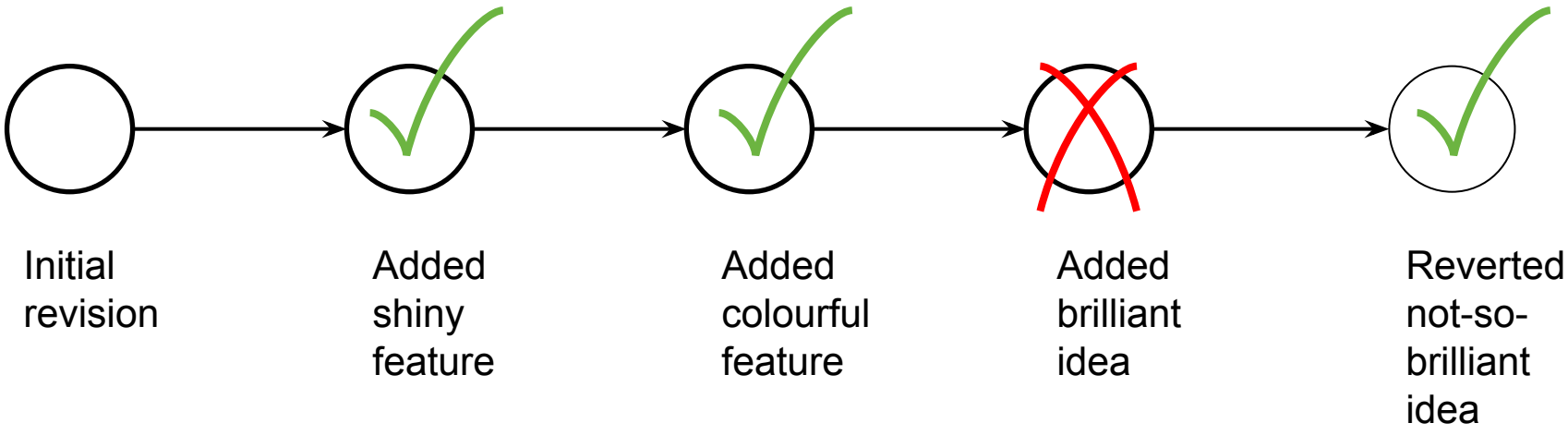
Reversion



Reversion



History

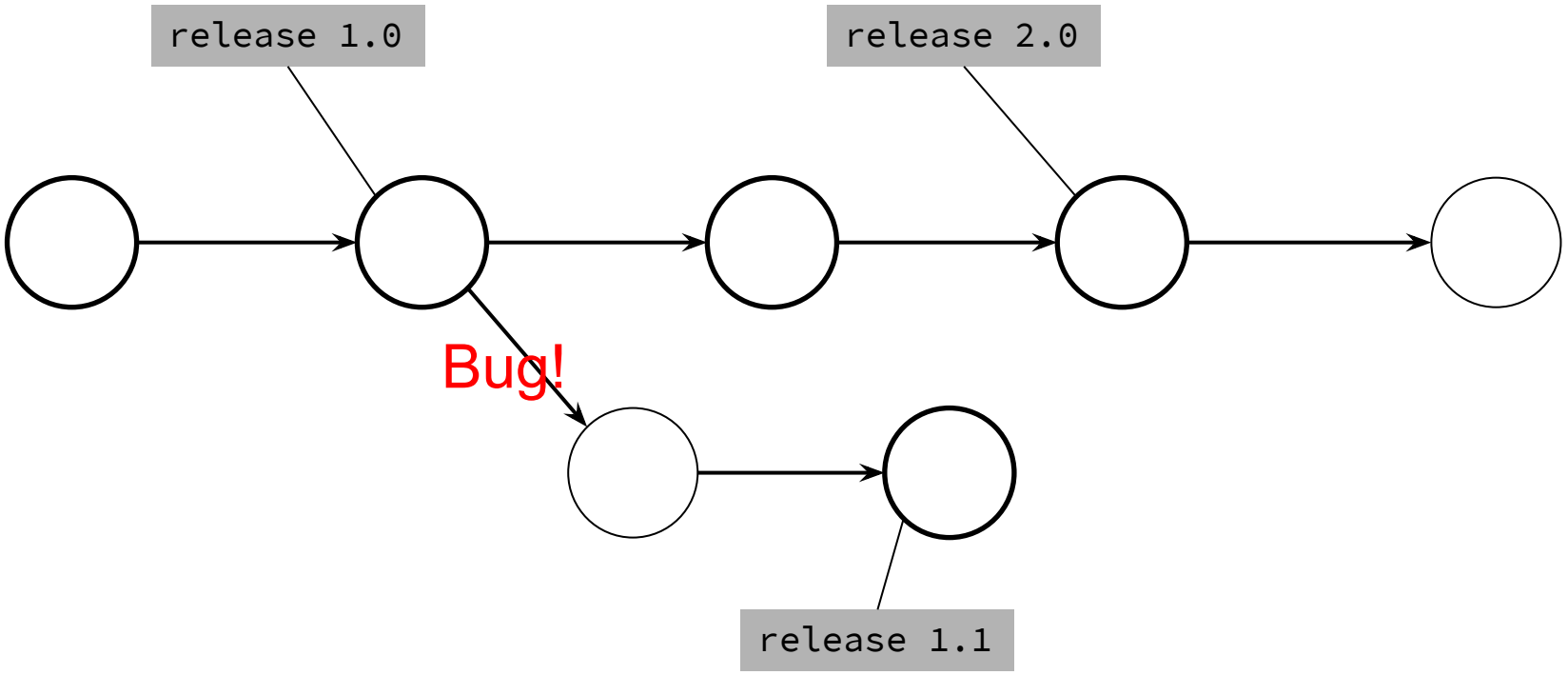


Change log

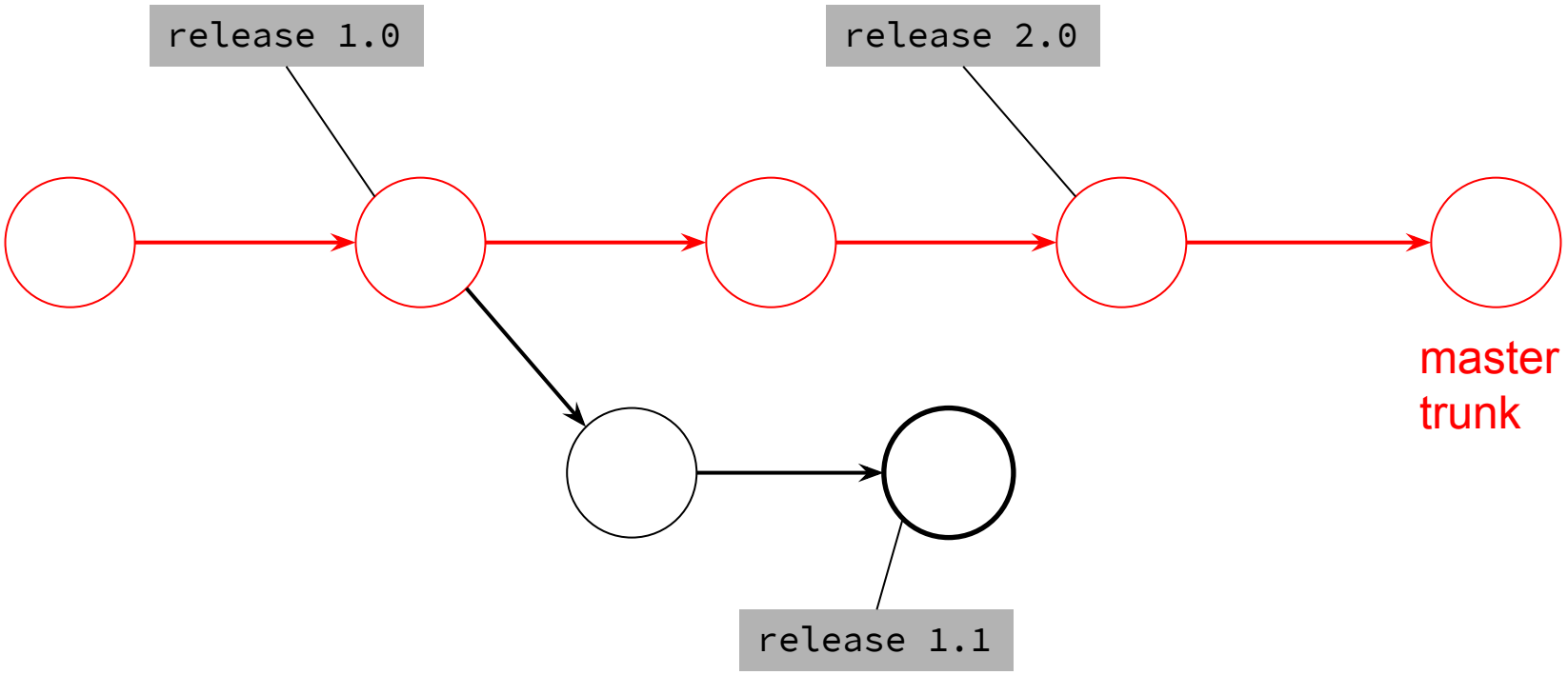
Branching

I have to maintain the version I sold to Alistair as well as developing the new feature to sell to Barbara

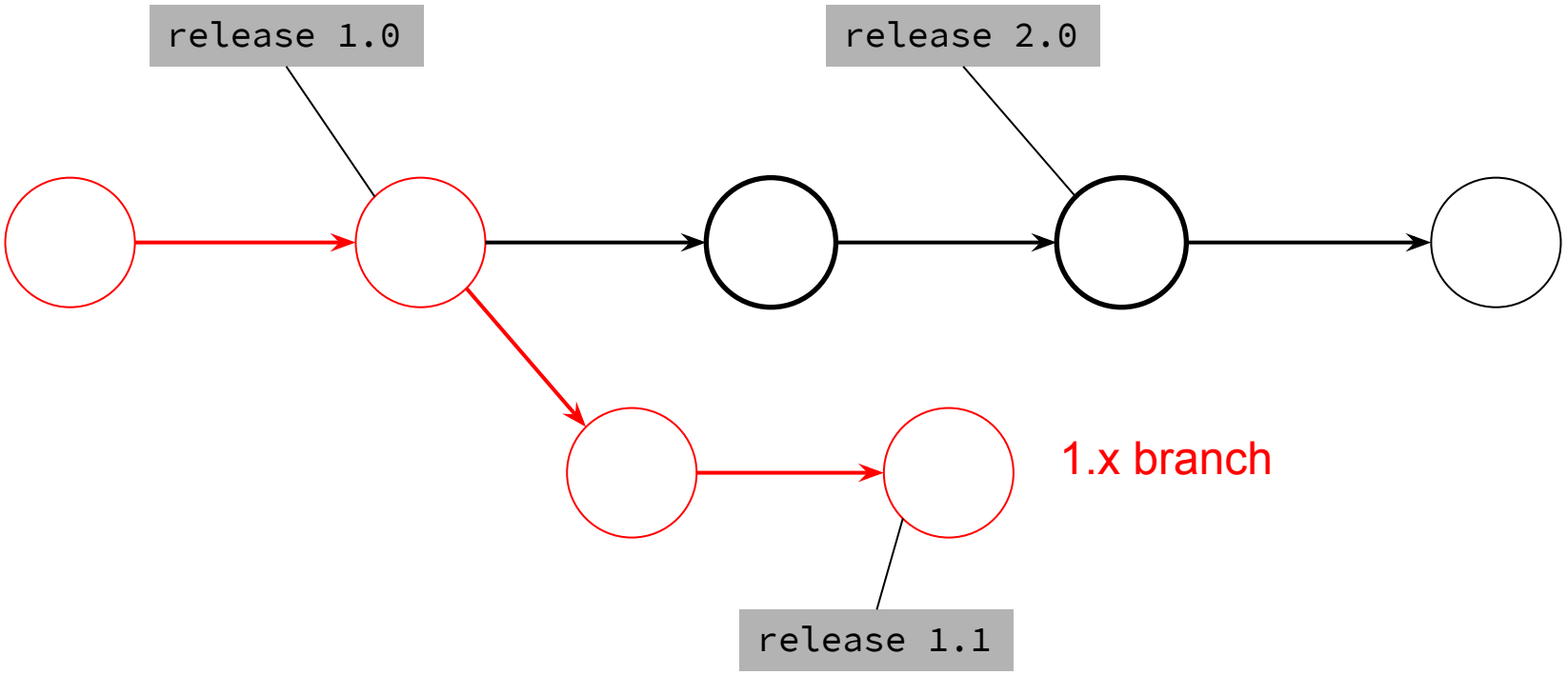
Branching



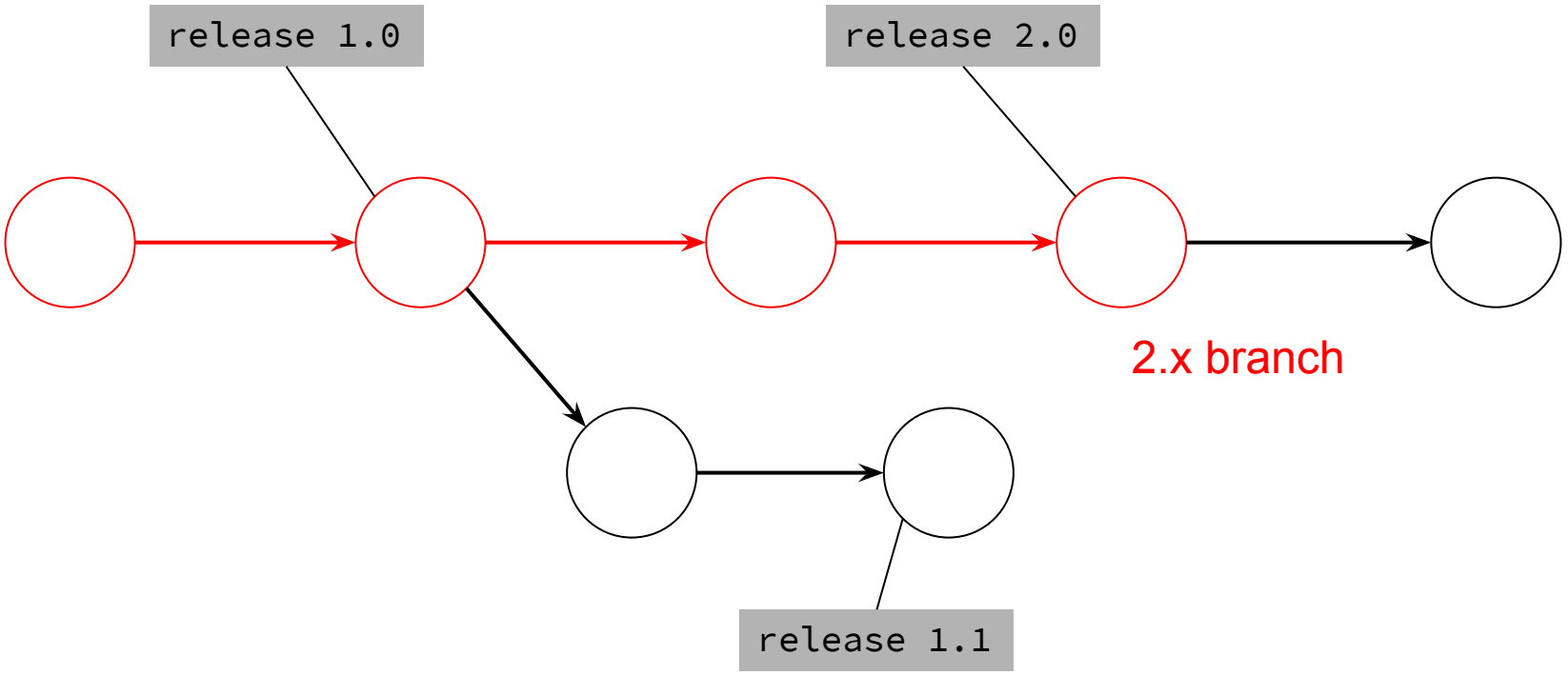
Branching



Branching



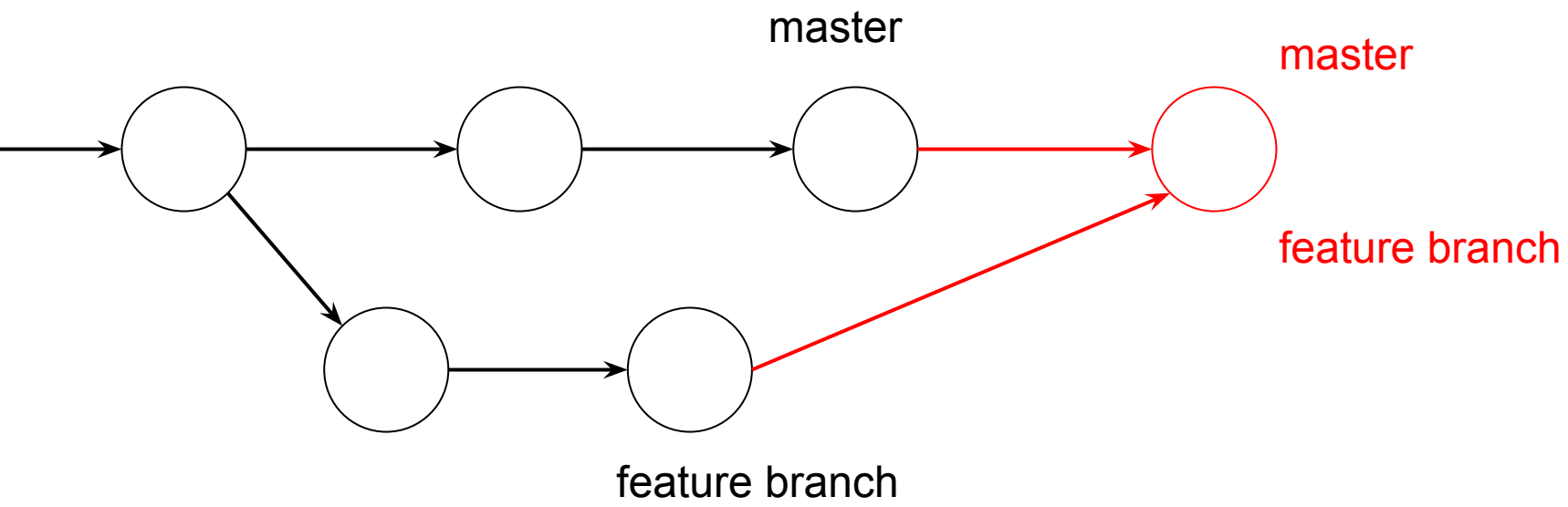
Branching



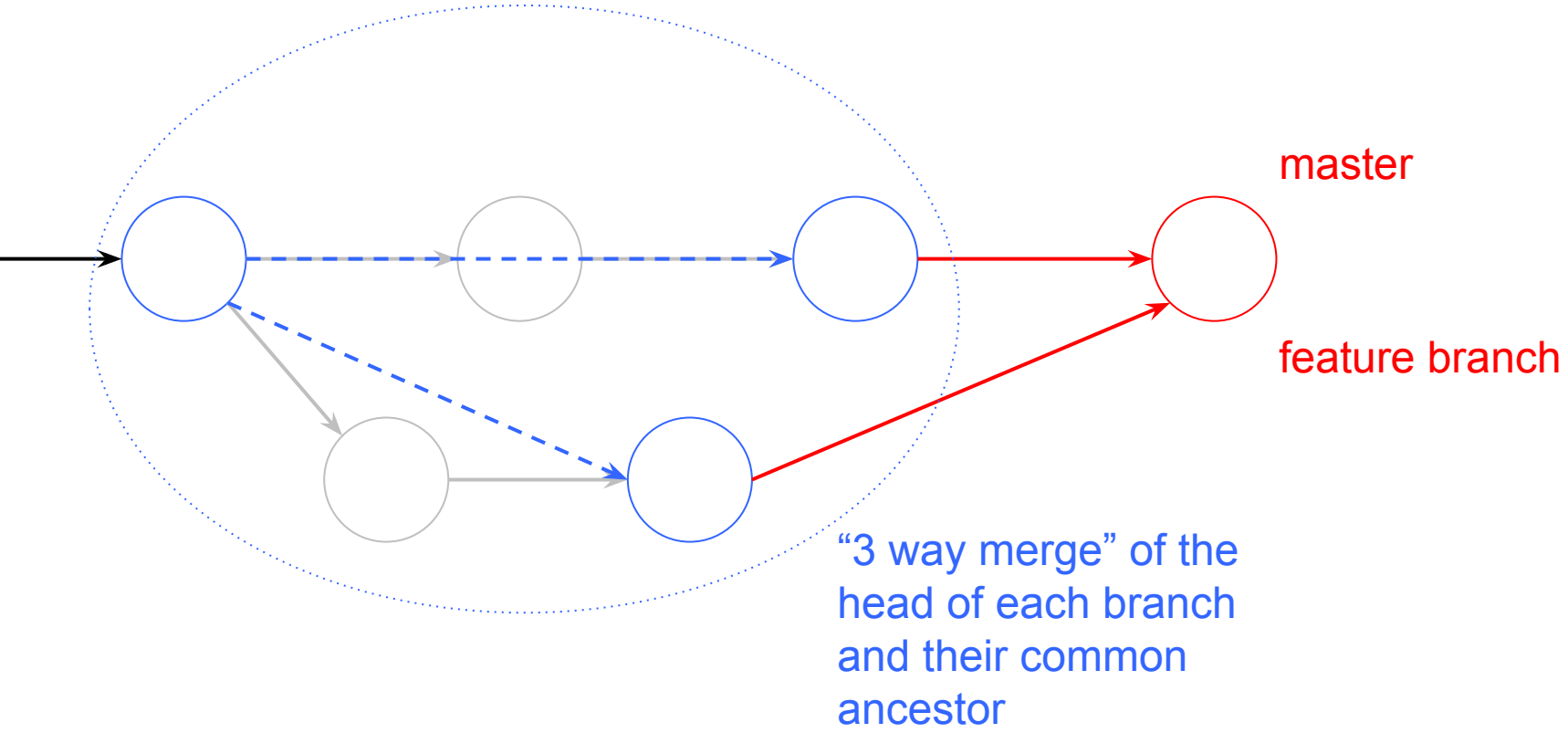
Merging

Now Alistair wants some of the stuff
I did for Barbara too

Merging



Merging


master
feature branch
“3 way merge” of the head of each branch and their common ancestor

Local Version Control

Revision history is on your local machine

Lose it, and you’re toast

Distributed Version Control

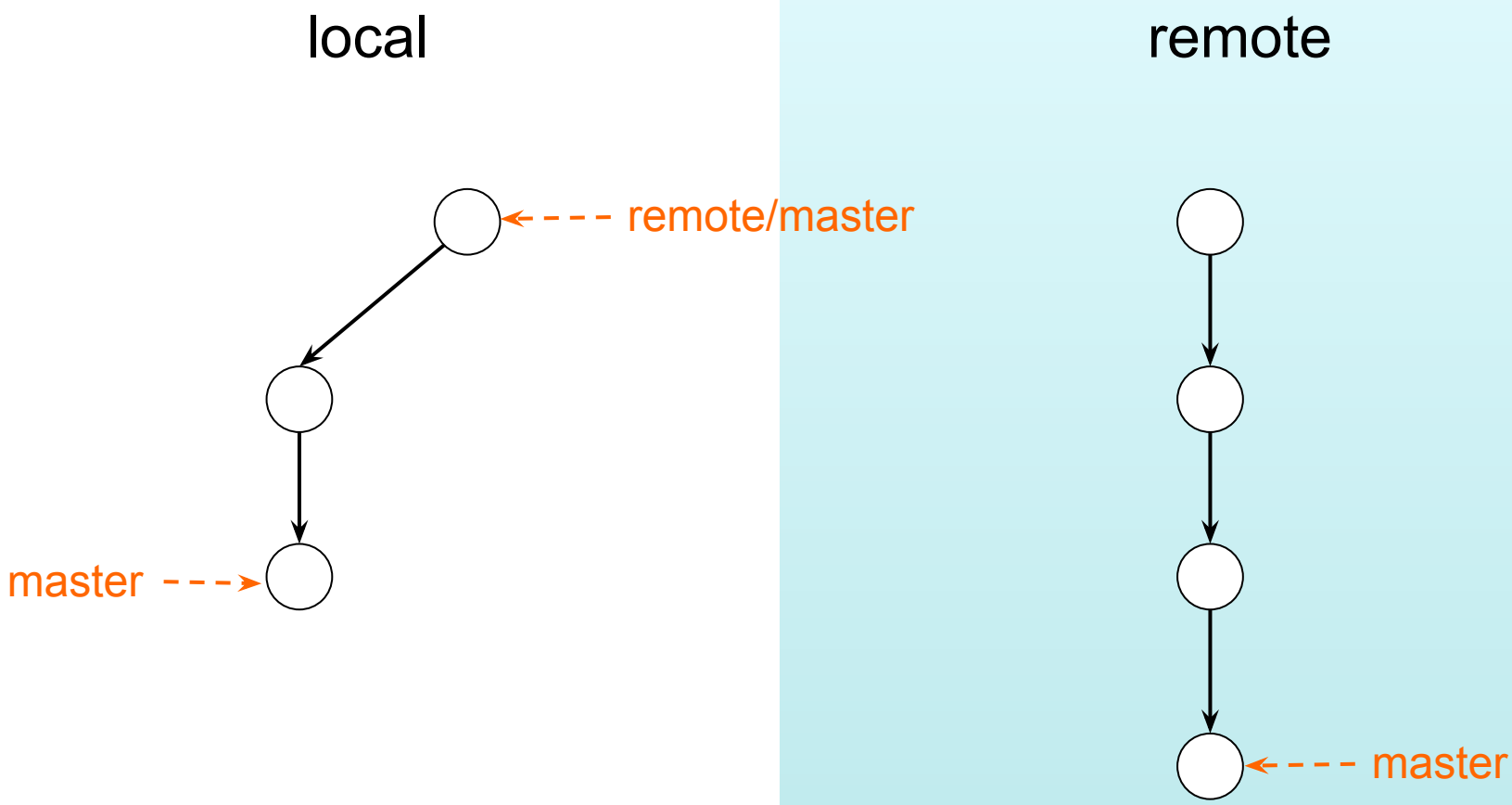
You have a revision history on your local machine

So does the server

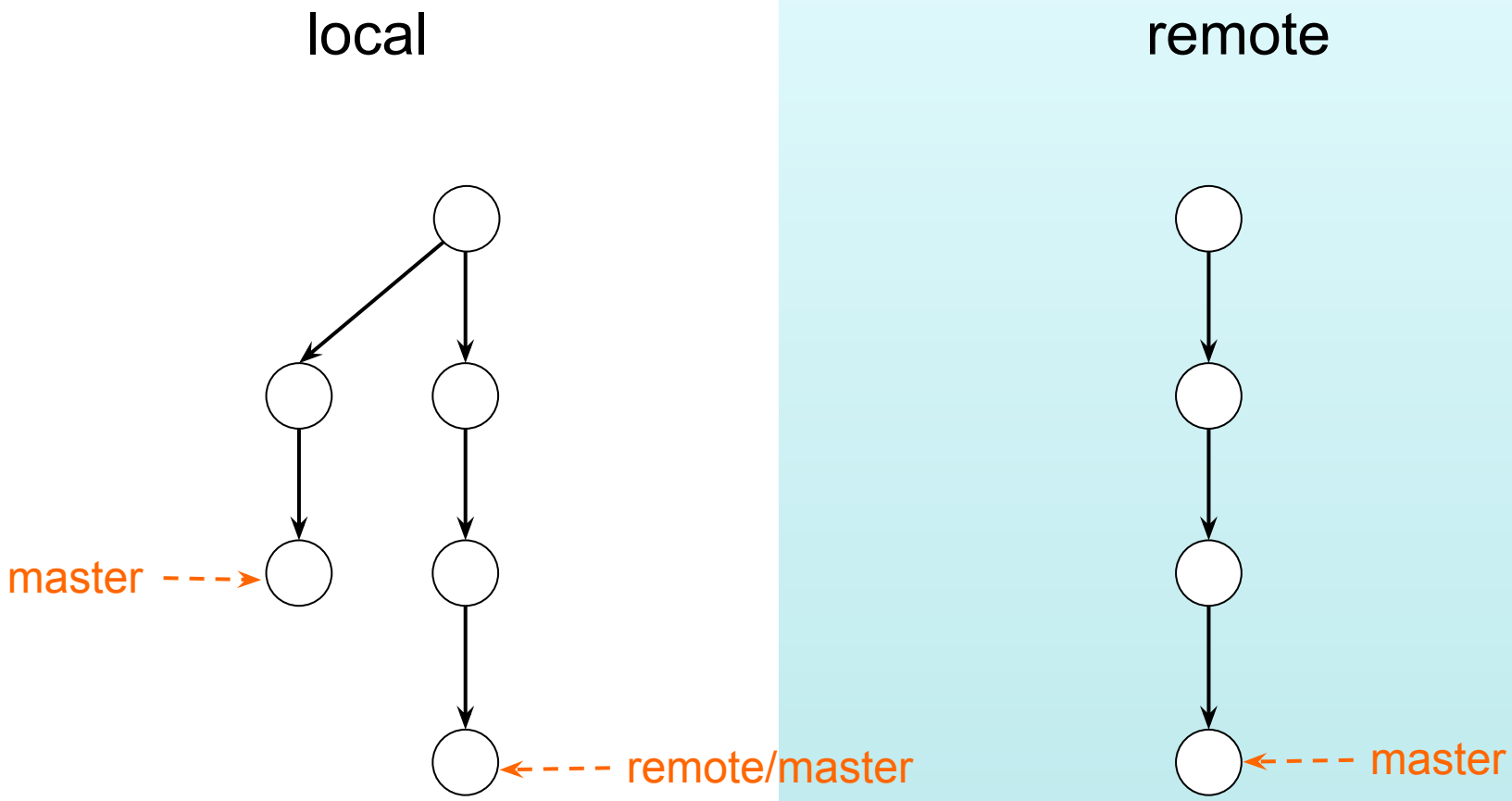
So does every developer

You push and pull commits to each other to keep your repositories synchronised

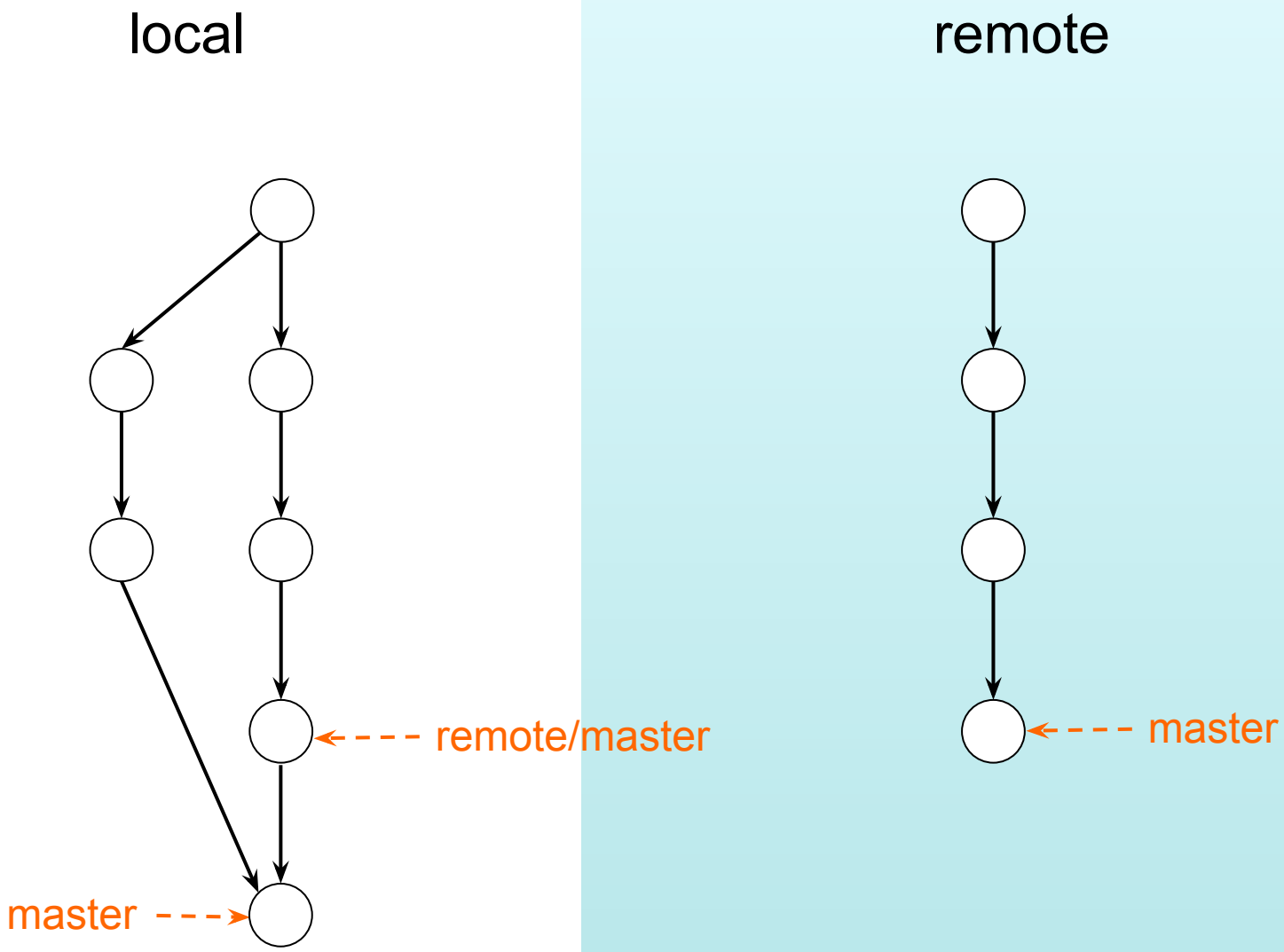
Pull = fetch + merge



Pull = fetch + merge



Pull = fetch + merge



Merging

Sometimes requires manual intervention

This is more likely, the more changes have taken place since the branches were last merged

What makes merge conflicts happen?

Merge conflicts occur when both branches have made different changes to the same locations in the same files

What might make merge conflicts likely?

Possibility #1

If two people work on the same task at the same time
(and don’t know it)

They will probably both change the same code in the same files
(slightly differently)

And merge conflicts will ensue

Solution #1

We need a way of telling each other what we are working on

“Issue Management”

Issues
(sometimes called tasks/jobs)

Discrete problem or task related to the project

We can track

- Status
  - open, closed, ...
- Description of the problem
- Assignee (who’s responsible)
- Severity
- Discussions related to the problem or solution
- Is it linked to a milestone?

GitHub Issues

GitHub issues also let you

- Mention issue numbers in commit messages
  - “This should help us work out the reason for #4.”
  - “This has fixed the memory leak. We can close #4.”
- Commit will show up in the history for the issue
- https://github.com/UQdeco2800/minesim/issues?q=is%3Aissue+

Possibility #2

Suppose your code has one function that does *everything*

Every change will involve changing that function

Even if two people work on different tasks,
they will still have to modify that function

Merge conflicts will ensue

Solution #2

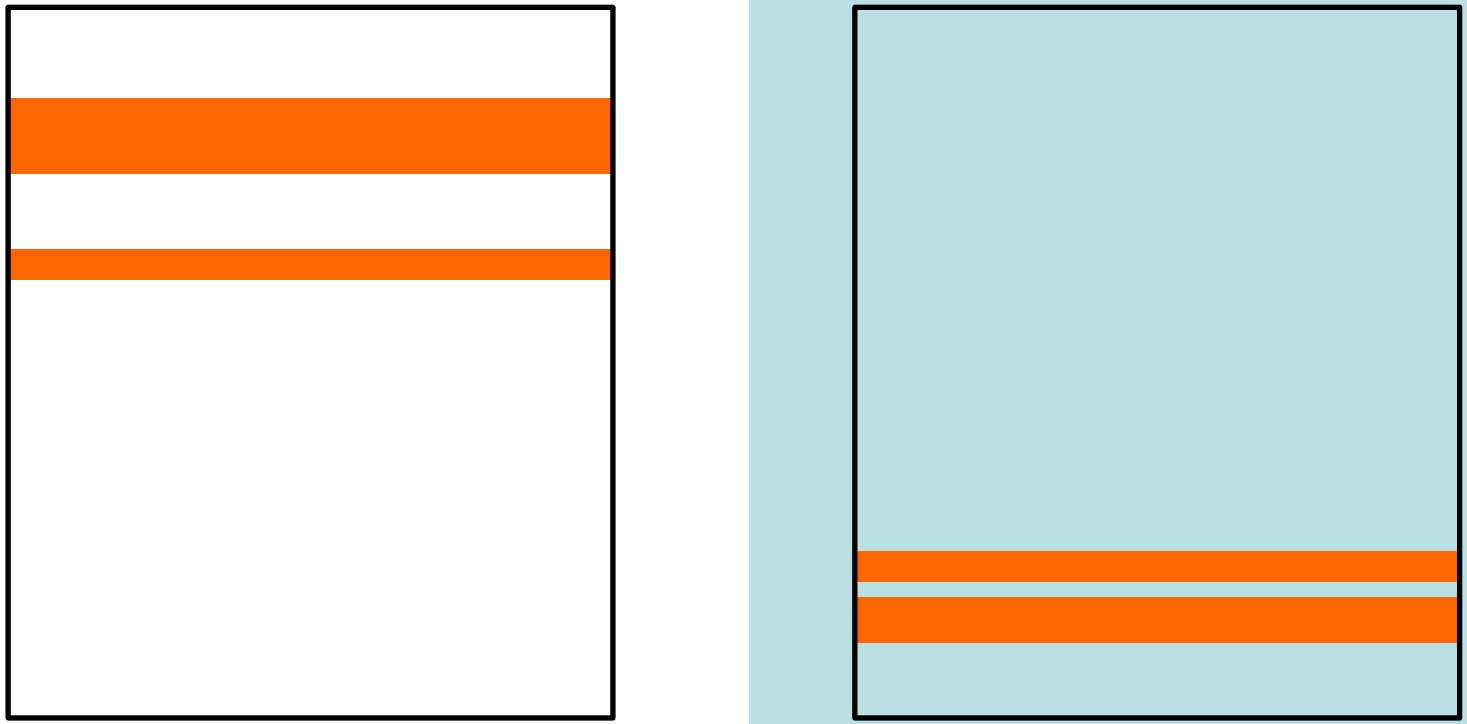
Write your program so that each part has distinct and coherent responsibilities

“Good design!”

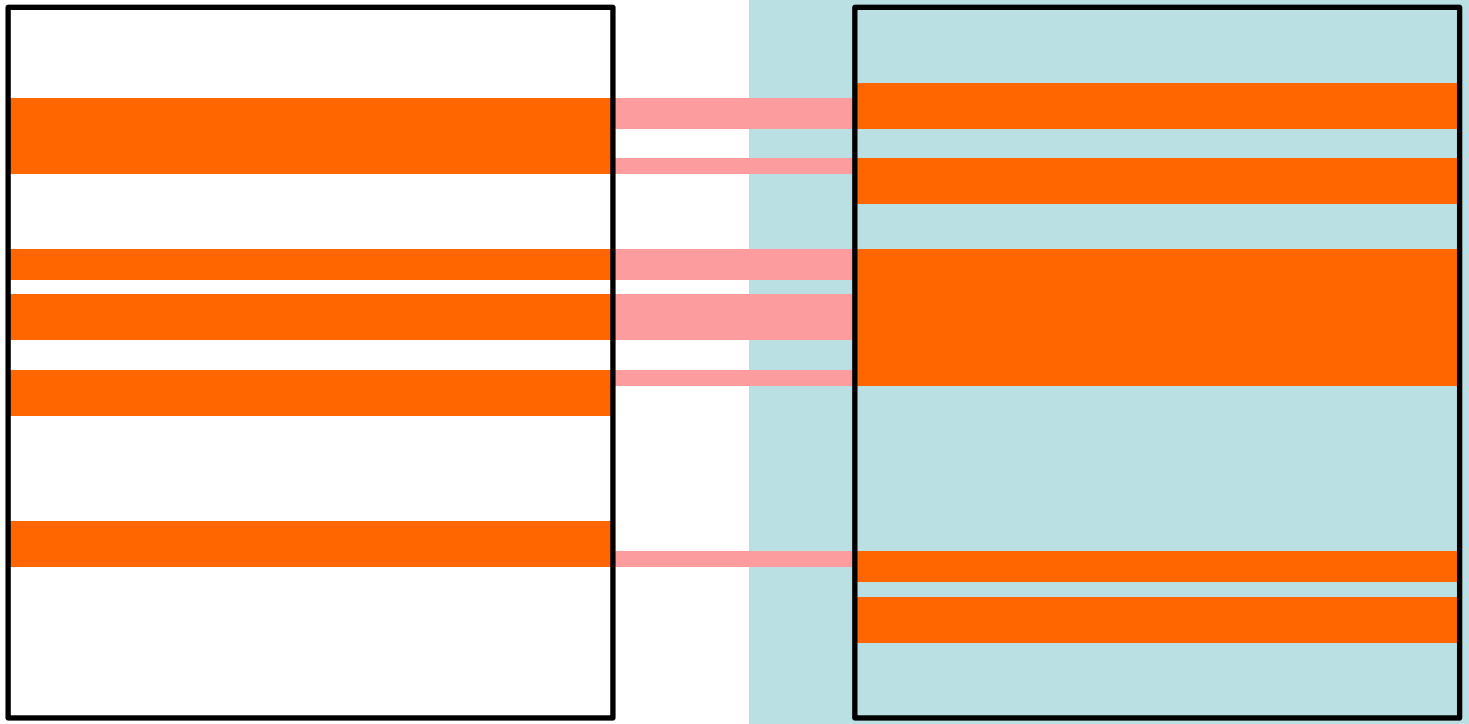
Possibility #3

Bigger changes are more likely to conflict
than smaller changes

Possibility #3



Possibility #3



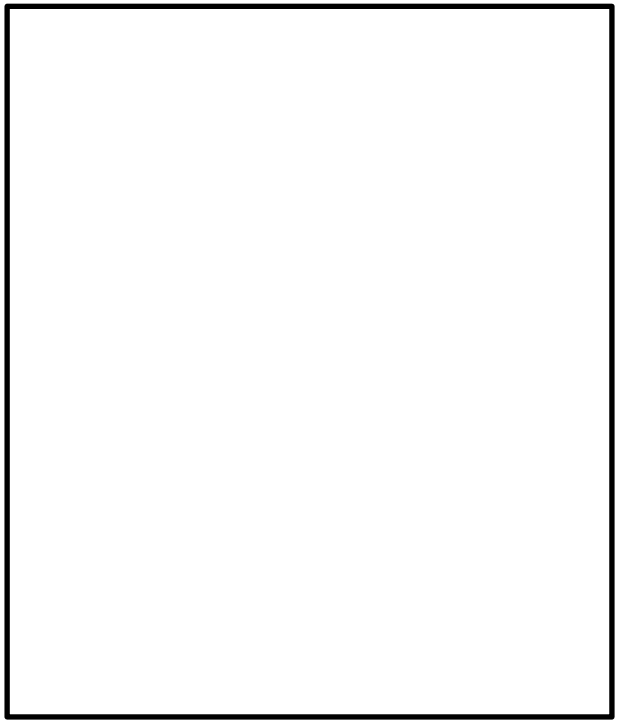
Solution #3

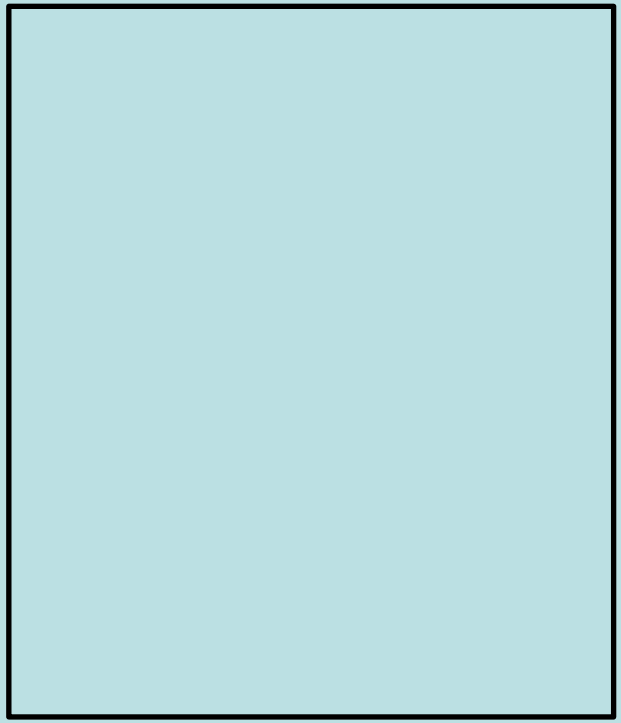
Commit *small*, *meaningful*, *coherent* changes

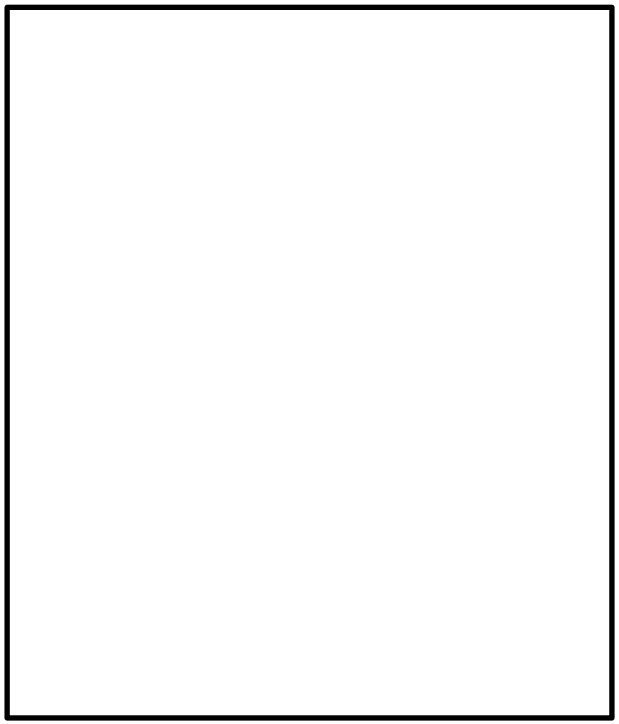
(Not *“here’s my last month’s work in one big lump”*)

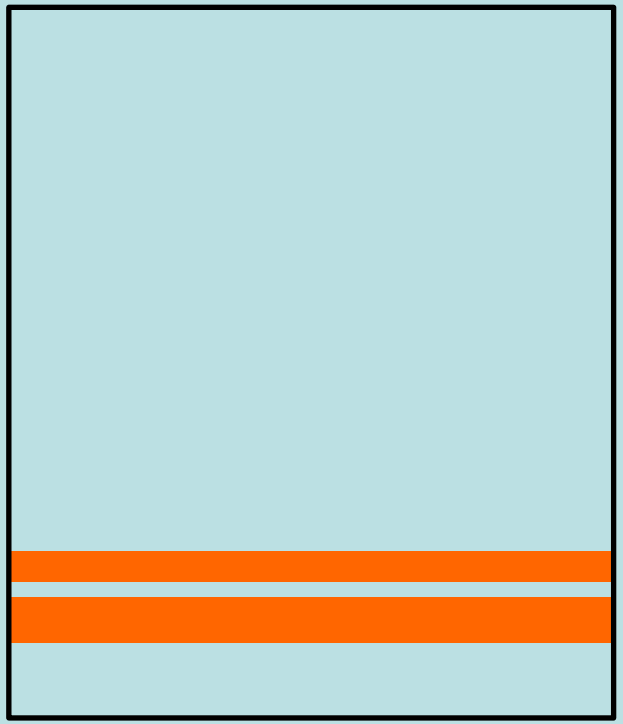
Possibility #4

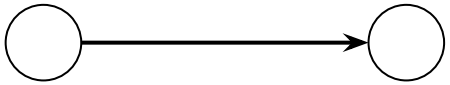
Every unmerged commit makes
your merge bigger and harder

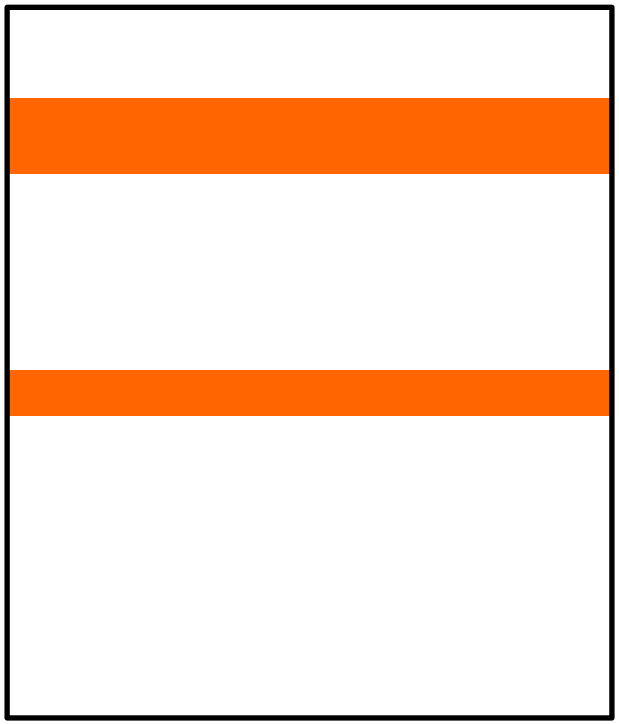


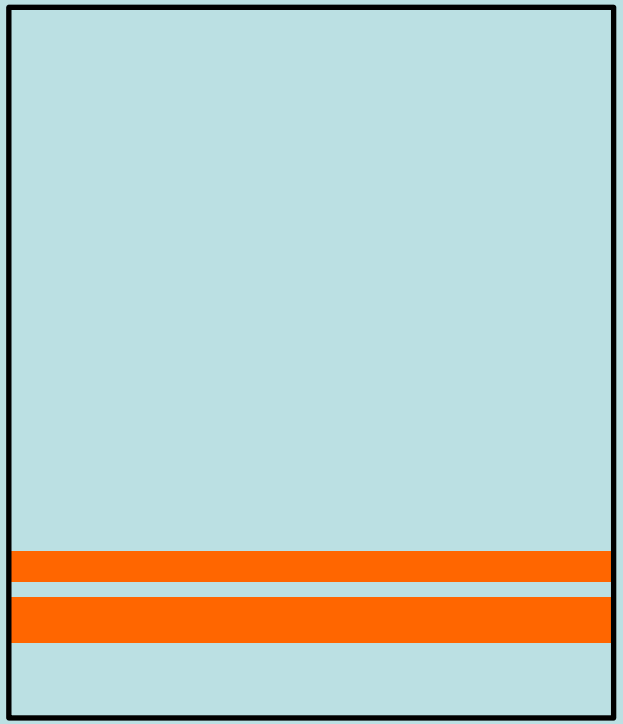


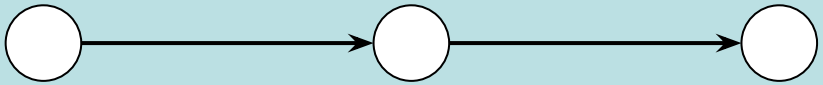


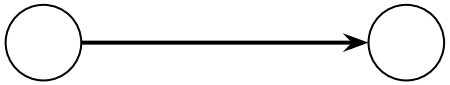


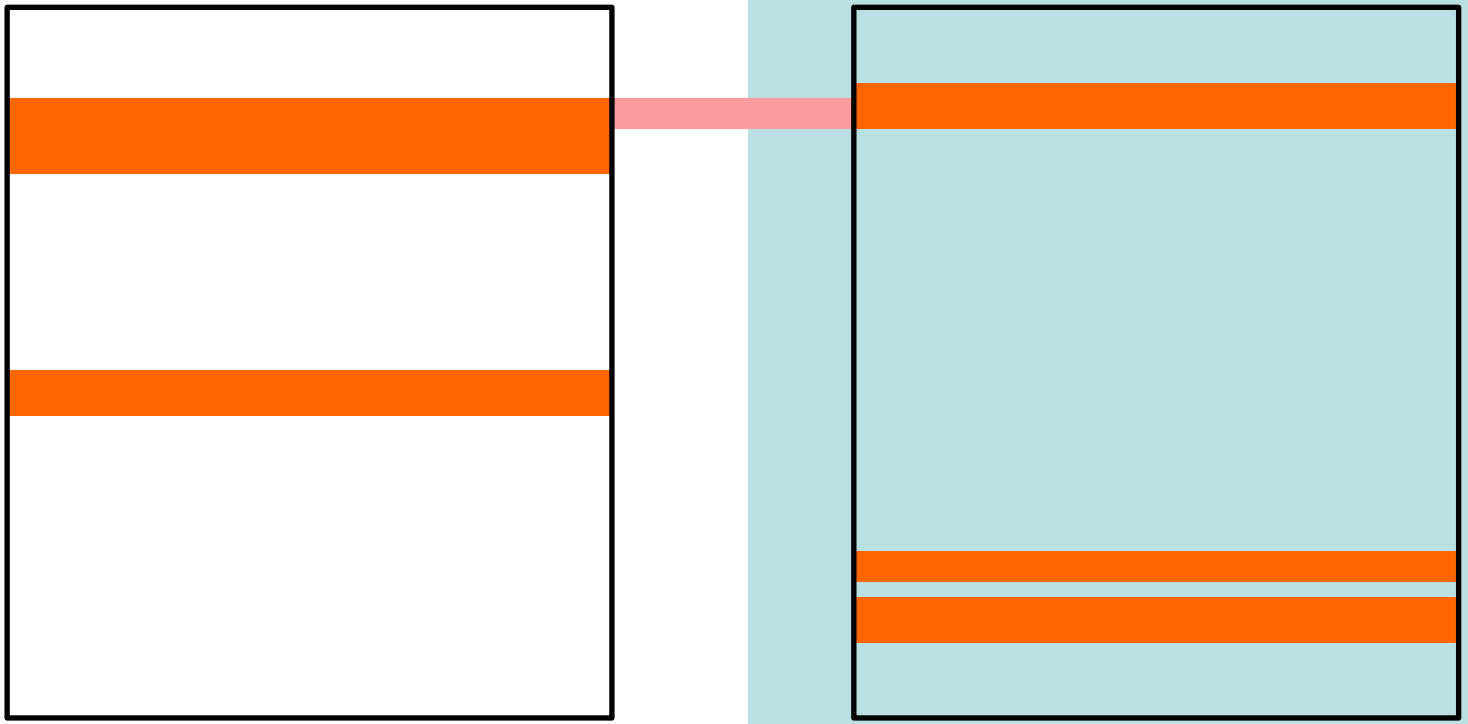


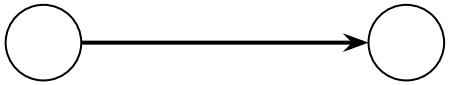


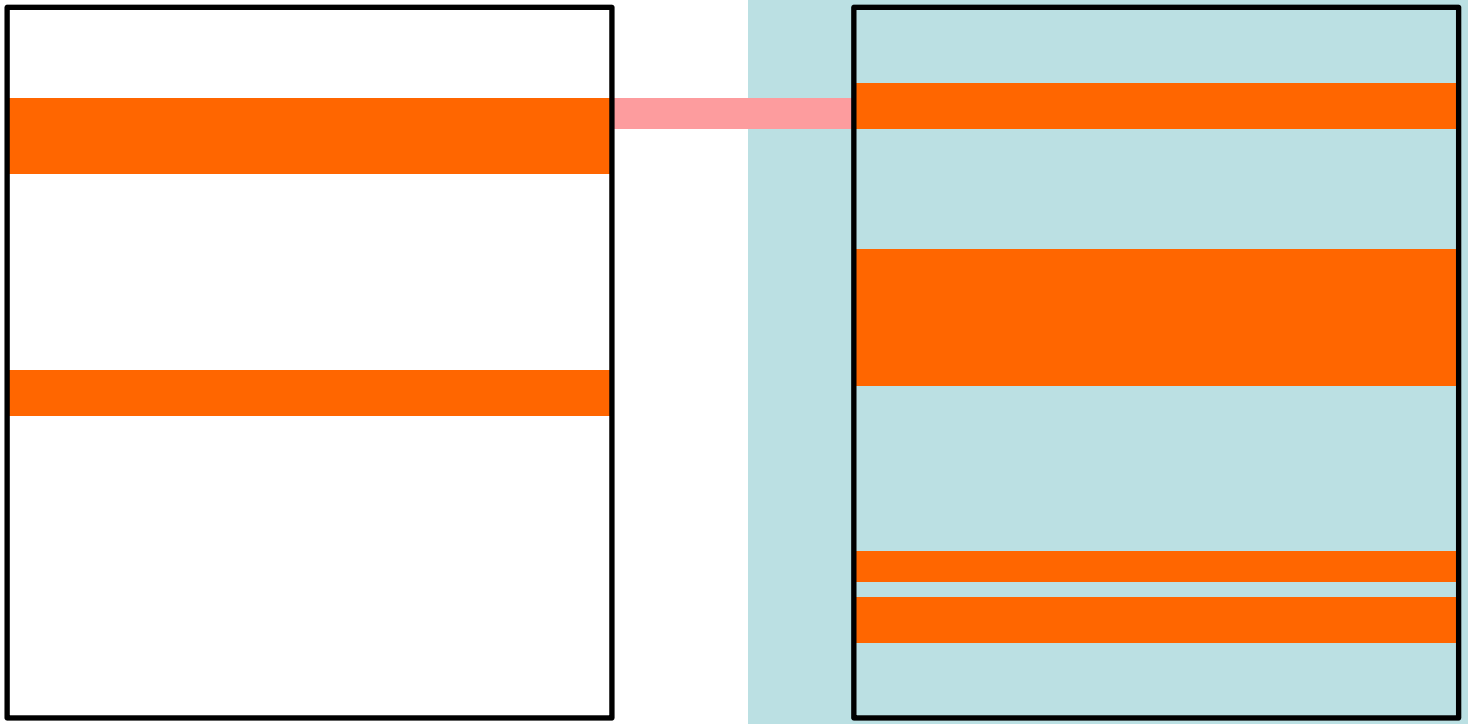


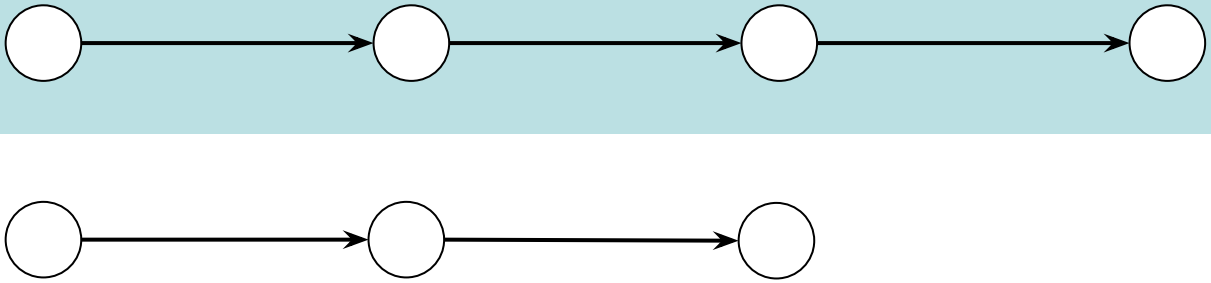


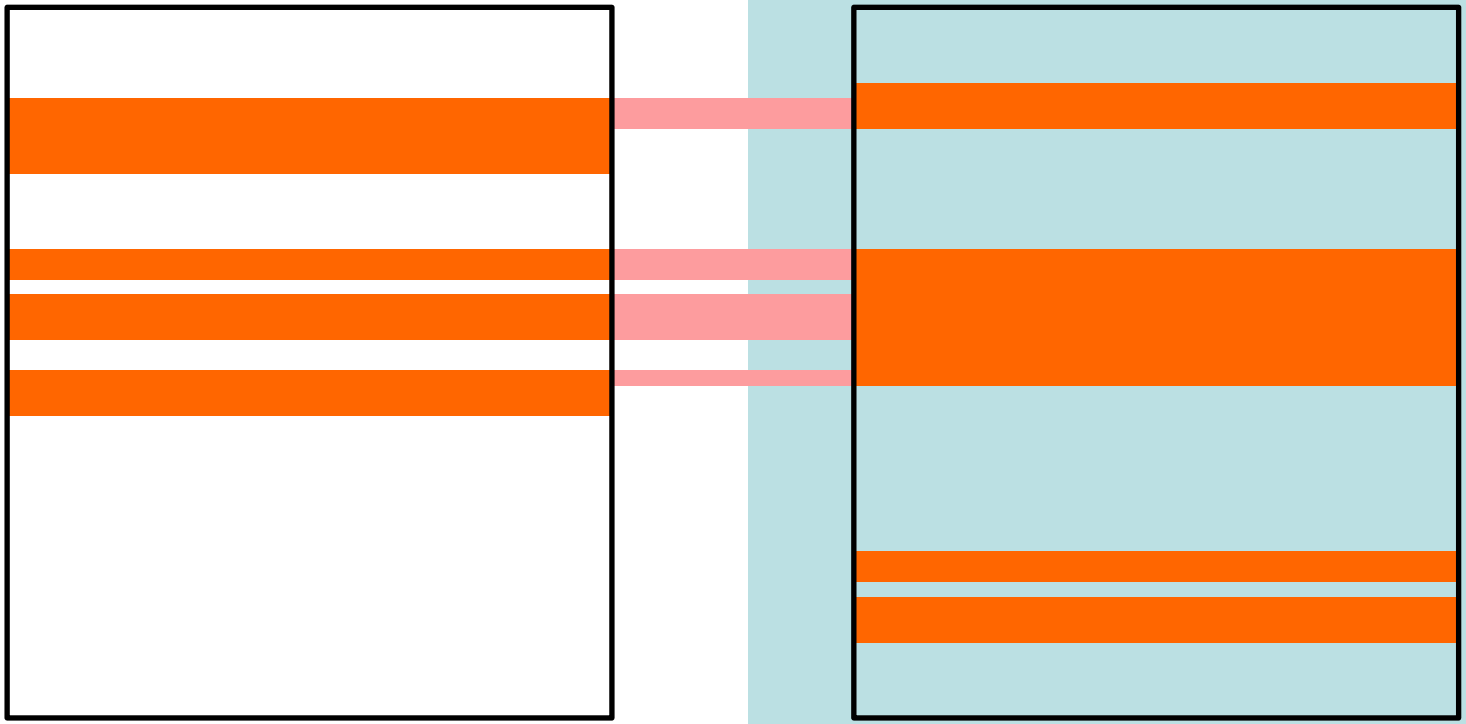


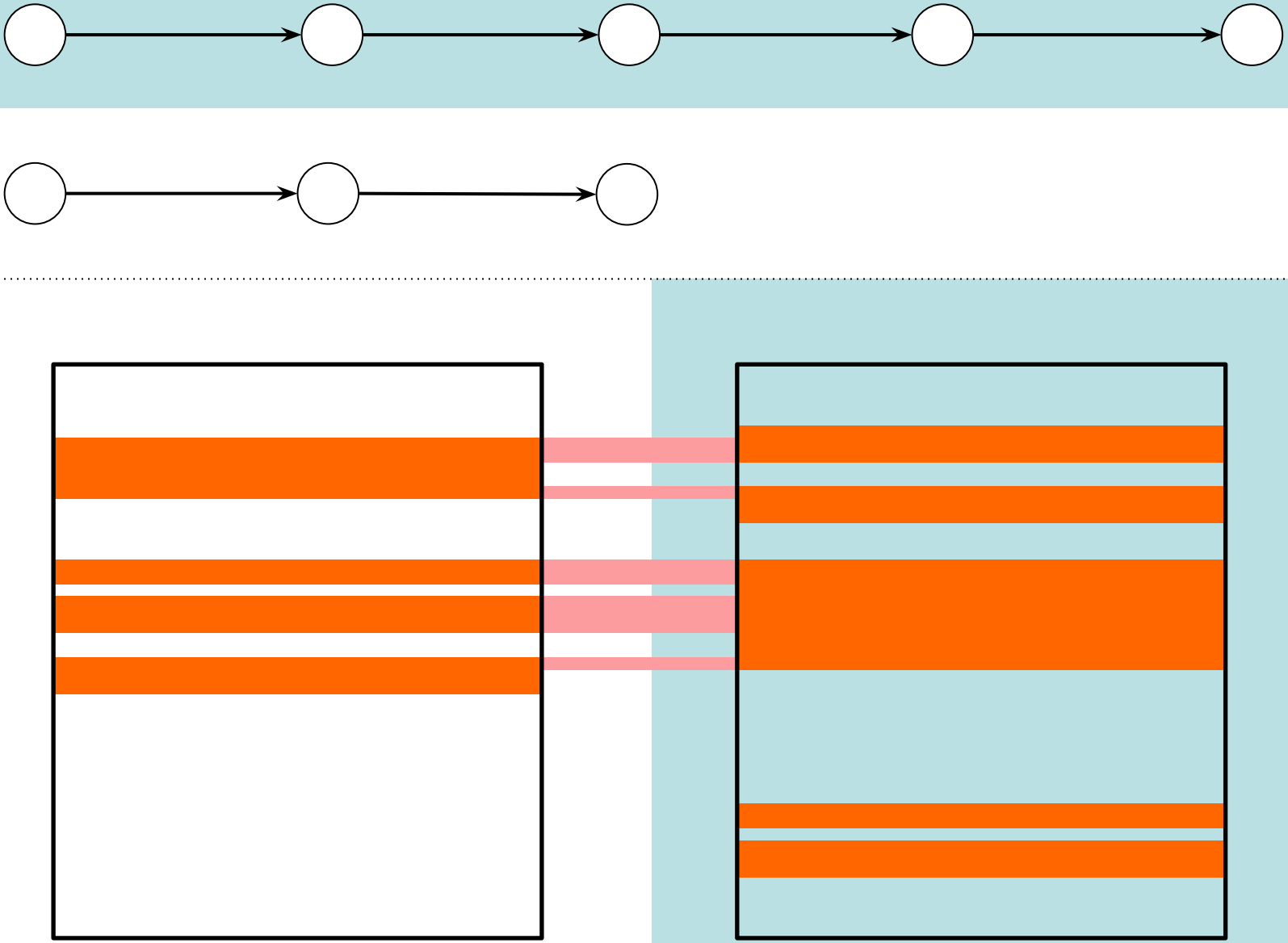


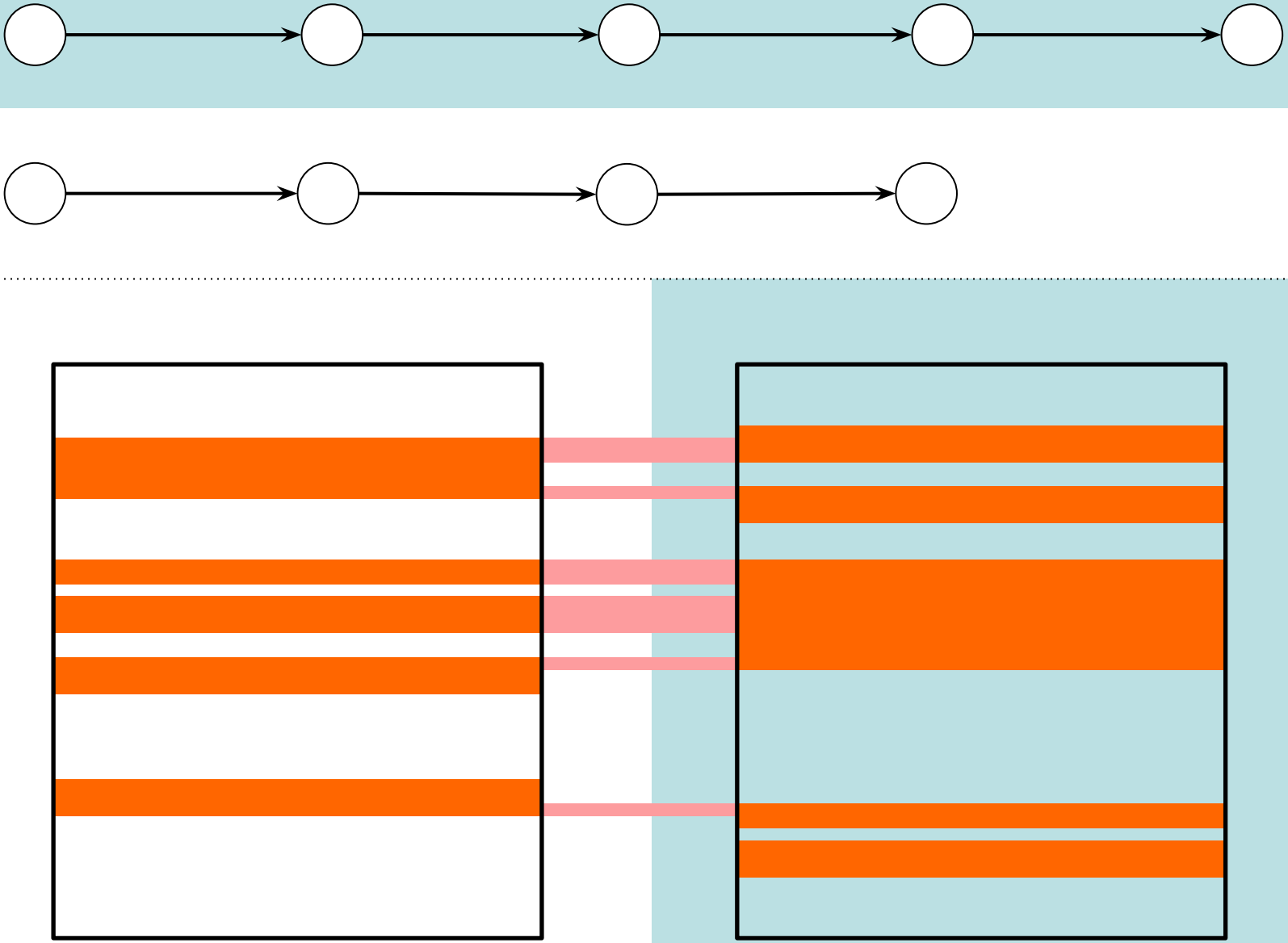






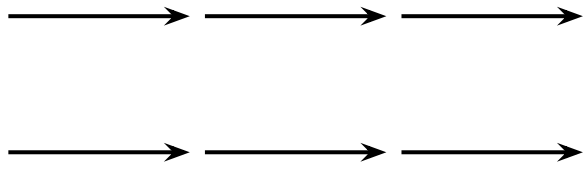




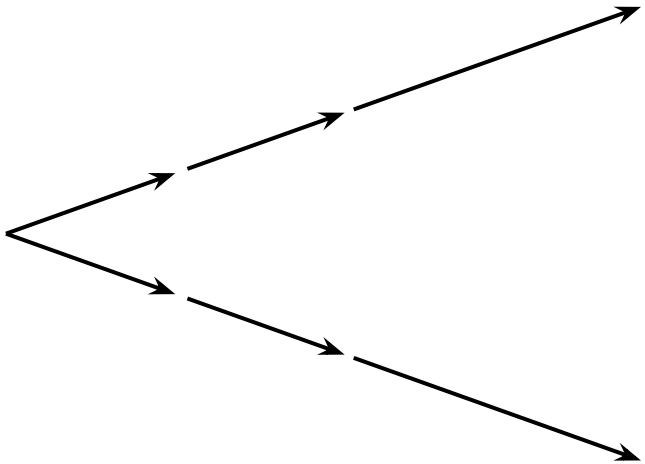


With each unmerged commit your code and master diverge

not parallel



but divergent



“Integration Hell”

When **integrating** your work is as much effort as **coding** your work

Solution #4

Merge changes from origin/master into your development often

Push your changes to origin/master regularly

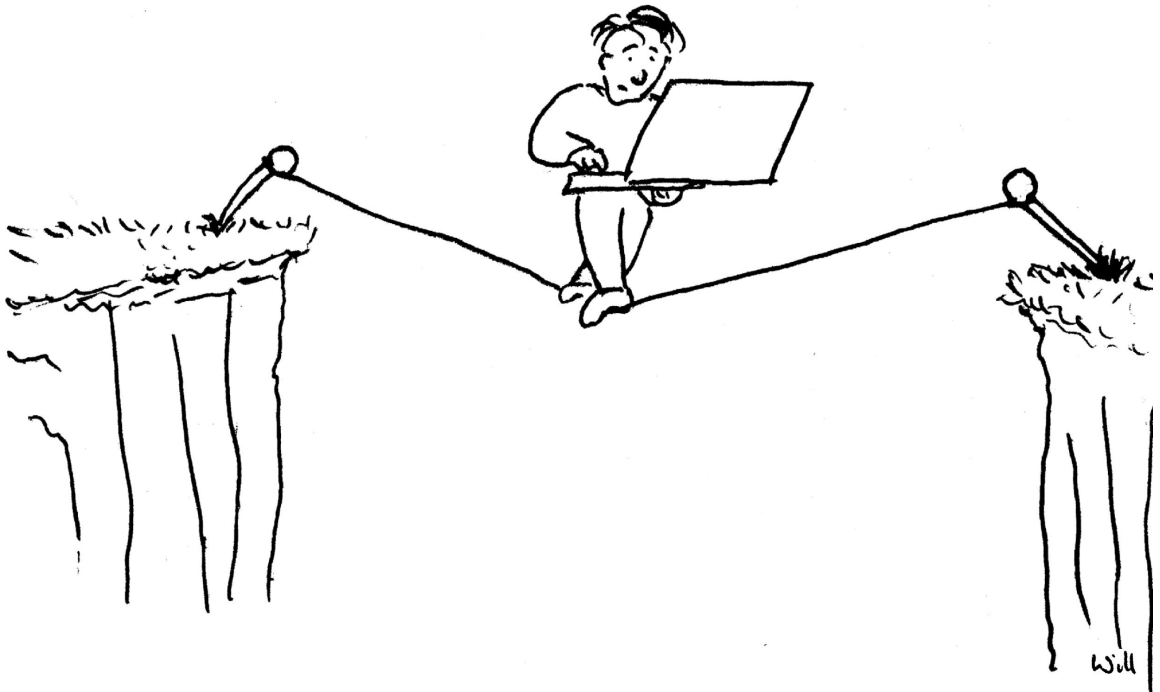
“Continuous Integration”

Git Resources

- https://edge.edx.org/courses/course-v1:UQx+GIT100x+2020/about
- https://edge.edx.org/courses/course-v1:UQx+GIT200x+2020/about
- https://www.atlassian.com/git/tutorials
- https://git-scm.com/download/
- https://www.sourcetreeapp.com/
- https://github.com/
- https://gitlab.com/
- https://bitbucket.org/
- …

Continuous Integration

Championed by Martin Fowler as part of
“Extreme Programming”



Continuous Integration Team Responsibilities

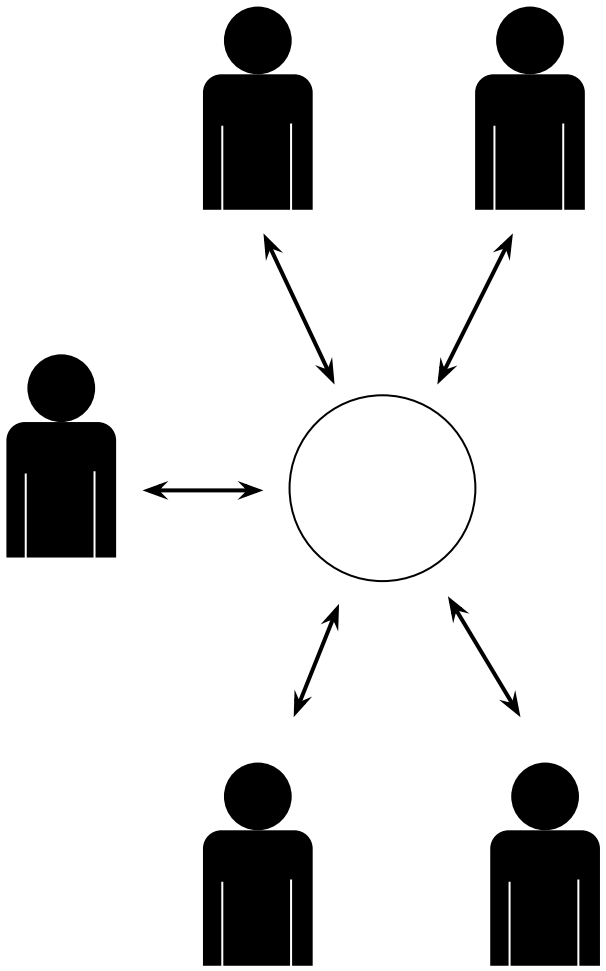
Merge frequently

Don’t push broken code

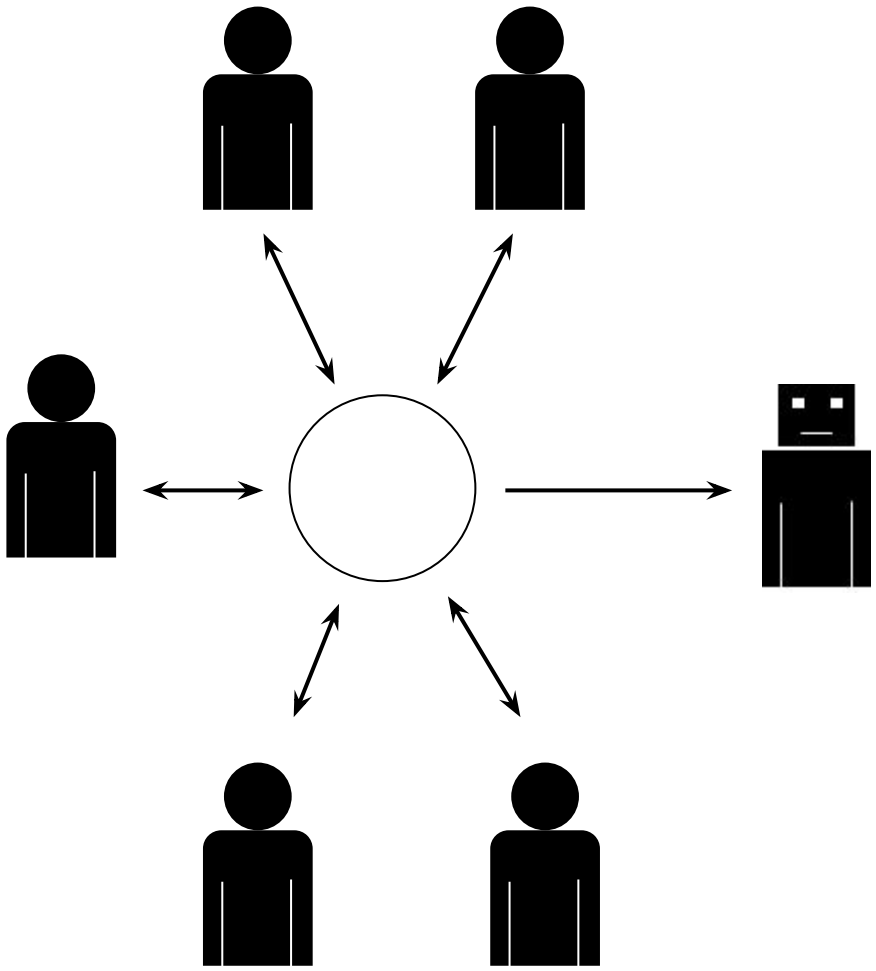
Don’t push untested code

Don’t push when the build is broken

If the build is broken, fix it

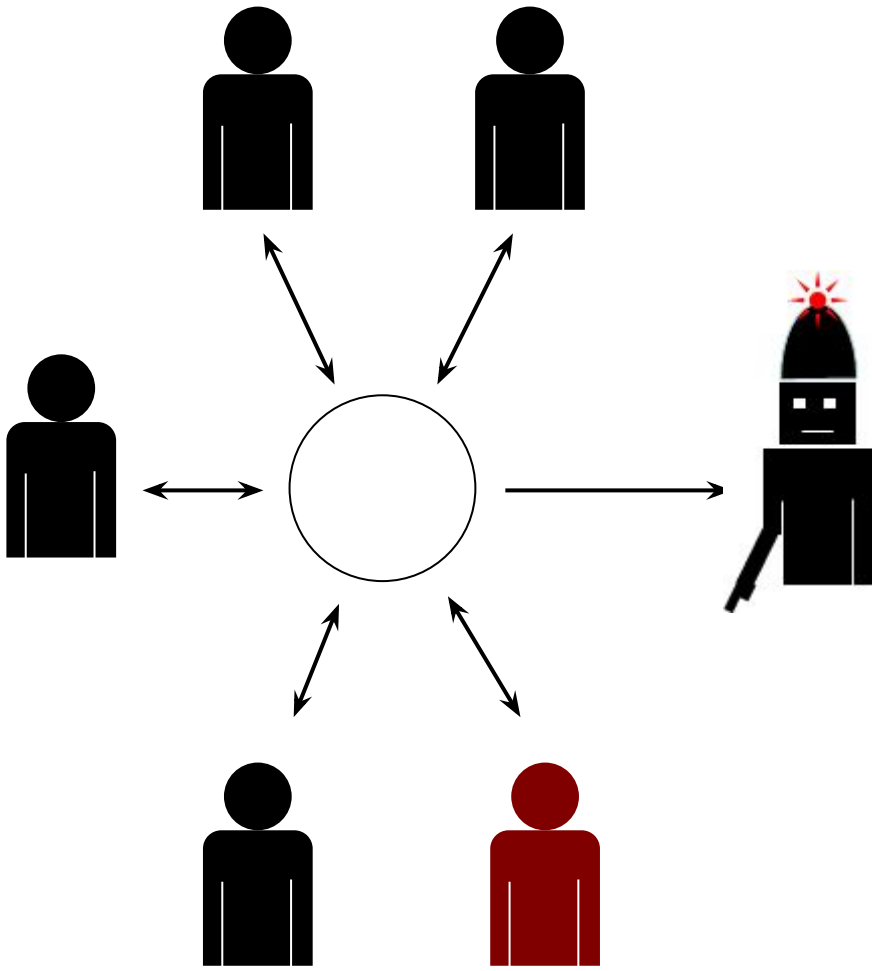


Everyone merges with the same repository



Everyone merges with the same repository

A robot butler checks out, builds, and tests the code on every push to verify the build is not broken



Everyone merges with the same repository

A robot butler checks out, builds, and tests the code on every push to verify the build is not broken

And detects when someone breaks the build!

- Maintain a single source repository
- Automate the build
- Make the build self-testing
- Every commit should build on an integration machine
- Keep the build fast
- Test in a clone of the production environment
- Make it easy for anyone to get the latest executable
- Everyone can see what’s happening
- Automate deployment

- Maintain a single source repository
- Automate the build
- Make the build self-testing
- Every commit should build on an integration machine
- Keep the build fast
- Test in a clone of the production environment
- Make it easy for anyone to get the latest executable
- Everyone can see what's happening
- Automate deployment

Git

- Maintain a single source repository
- Automate the build
- Make the build self-testing
- Every commit should build on an integration machine
- Keep the build fast
- Test in a clone of the production environment
- Make it easy for anyone to get the latest executable
- Everyone can see what's happening
- Automate deployment

A single command to build everything.

- e.g. **gradle**

- Maintain a single source repository
- Automate the build
- Make the build self-testing
- Every commit should build on an integration machine
- Keep the build fast
- Test in a clone of the production environment
- Make it easy for anyone to get the latest executable
- Everyone can see what's happening
- Automate deployment

Write automated tests to verify that parts of the code do what they should.

- e.g. **JUnit** to write tests
- Tests are executed by **Gradle**

Run the tests *before* you push code to master.

- ***Don't push broken code!***

- Maintain a single source repository
- Automate the build
- Make the build self-testing
- Every commit should build on an integration machine
- Keep the build fast
- Test in a clone of the production environment
- Make it easy for anyone to get the latest executable
- Everyone can see what's happening
- Automate deployment

Jenkins builds and tests the code after every push to master.

- Maintain a single source repository
- Automate the build
- Make the build self-testing
- Every commit should build on an integration machine
- Keep the build fast
- Test in a clone of the production environment
- Make it easy for anyone to get the latest executable
- Everyone can see what's happening
- Automate deployment

Under ten minutes!

Development Process

Development Process

Many options …

- agile
- lean
- plan-driven
- …

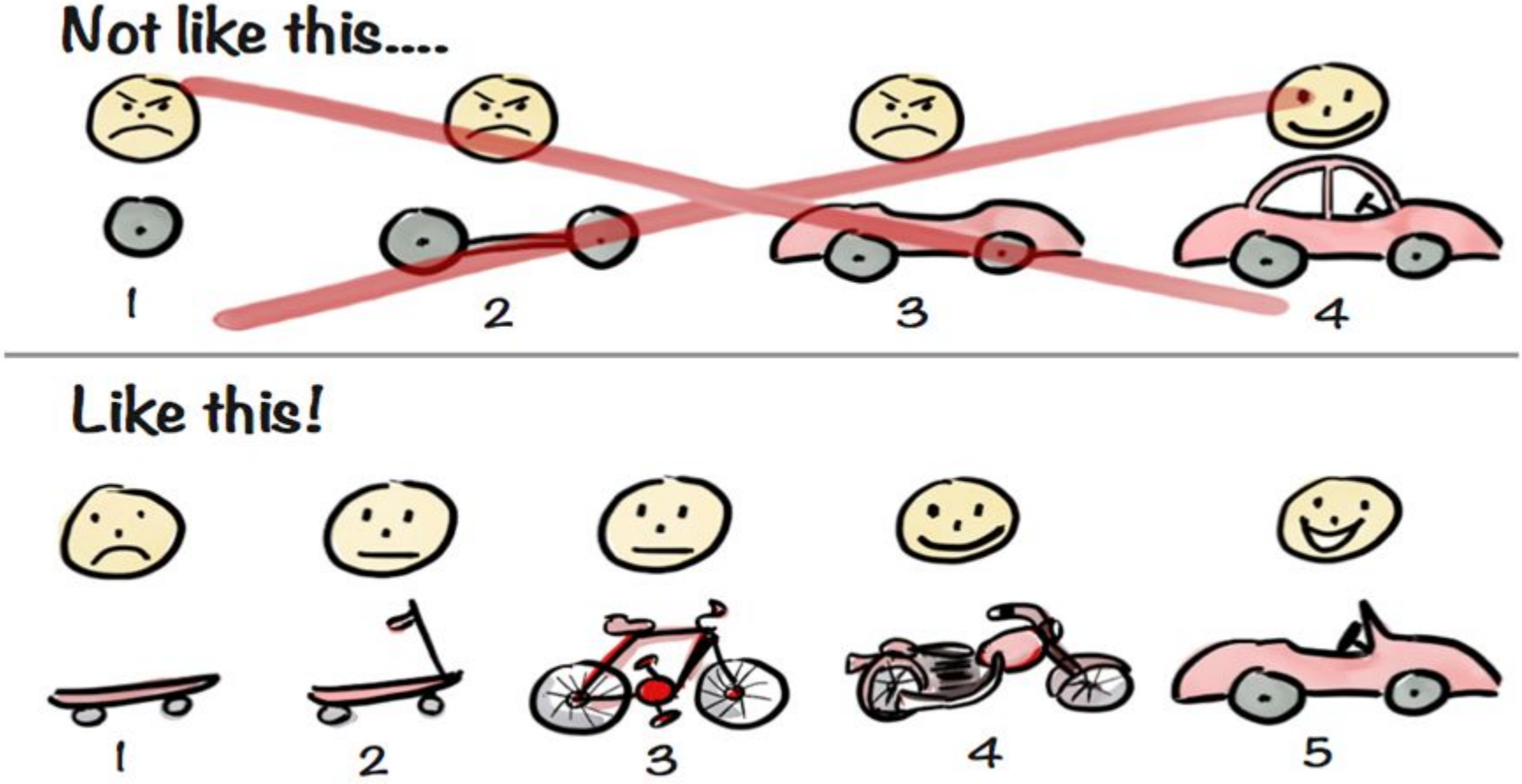
Scrum

- https://www.mountaingoatsoftware.com/agile/scrum

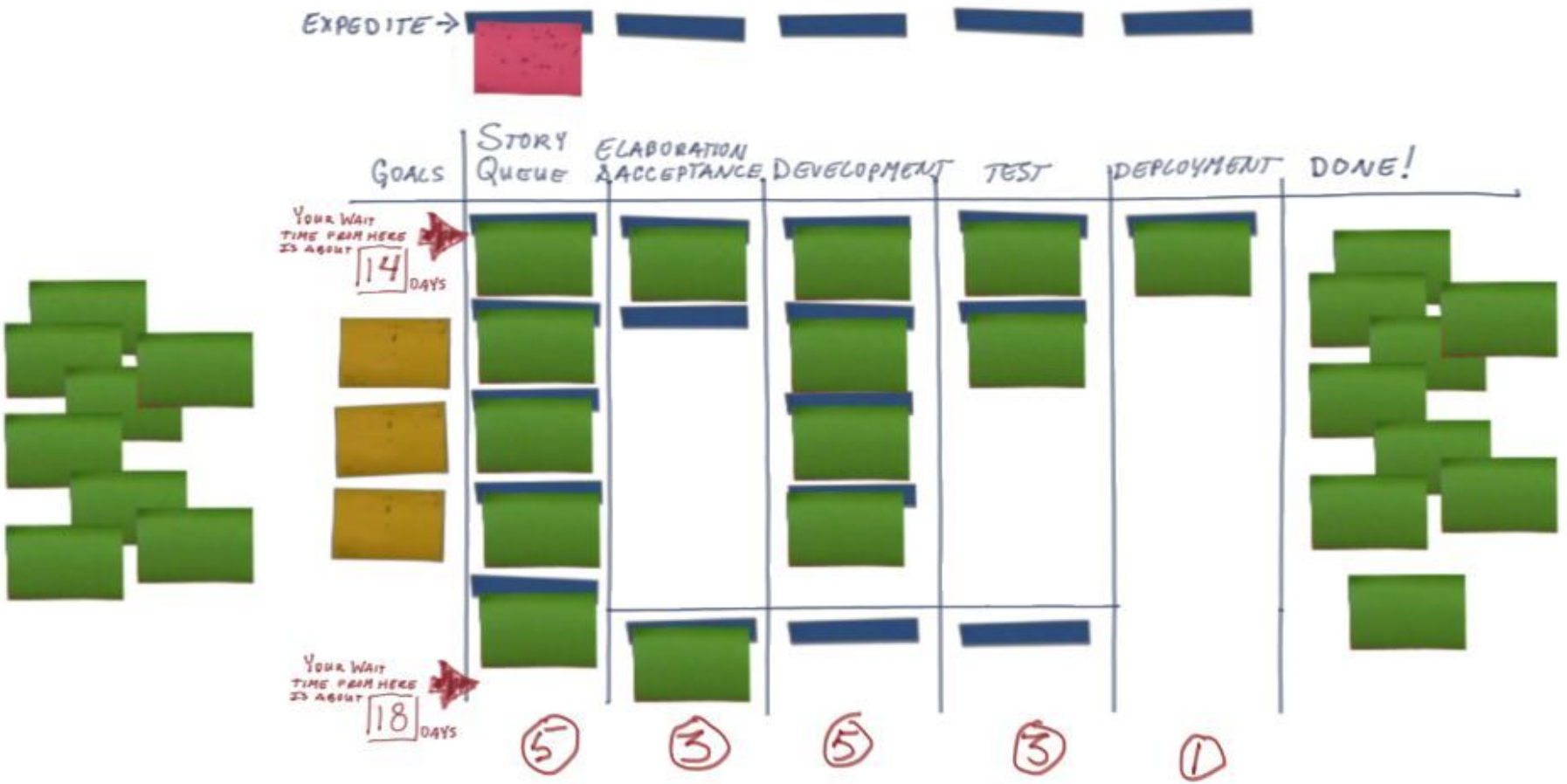
Kanban

- https://www.planview.com/resources/articles/what-is-kanban/
- https://www.atlassian.com/agile/kanban

Incremental Delivery



Kanban Board



Visual Task Tracking

- GitHub Projects
- Trello
  - https://trello.com/

Licensing

Licensing

- Who can use it?
- Who can change it?
- Who can distribute it?
- What restrictions are appropriate?

- https://choosealicense.com/
- https://creativecommons.org/