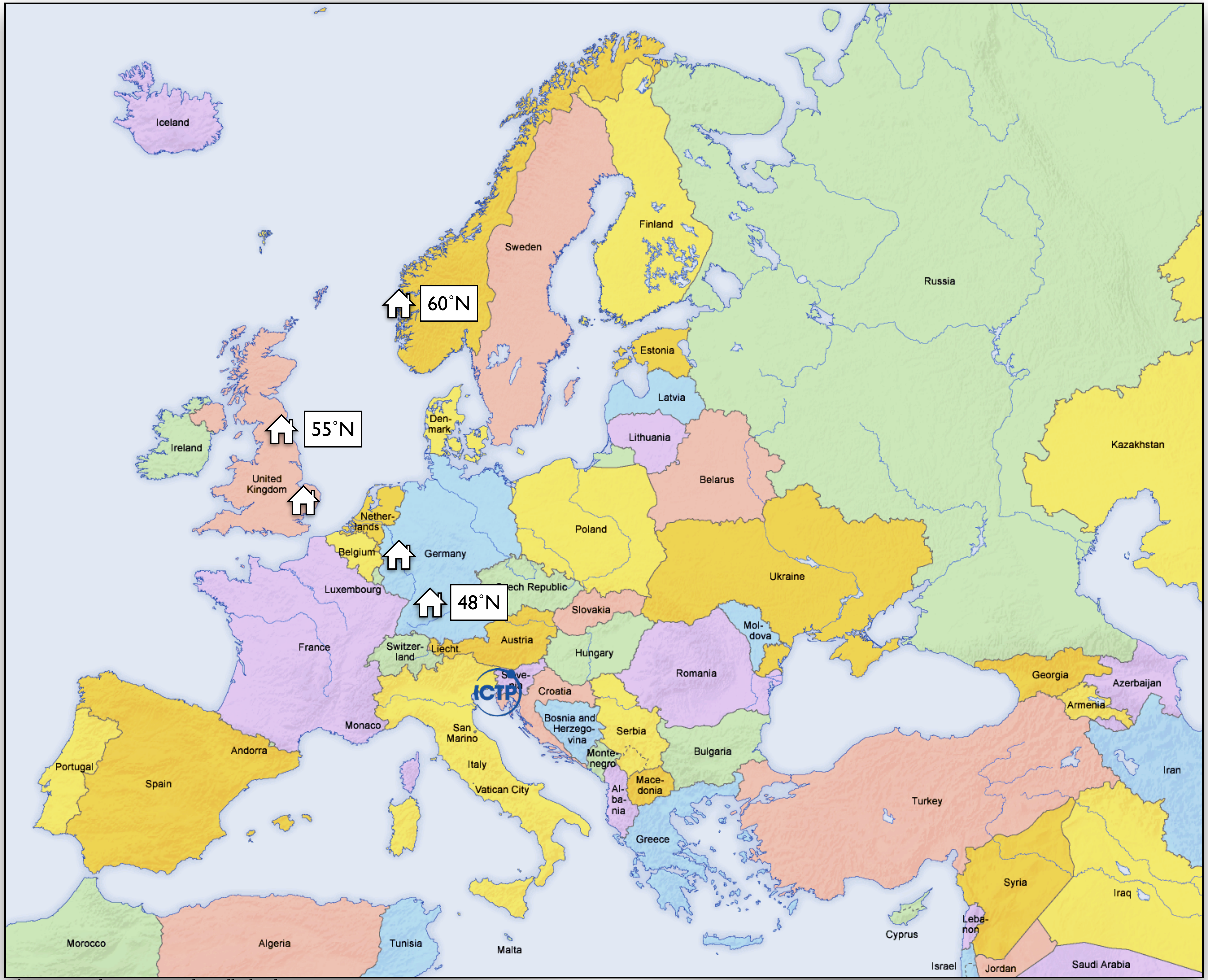
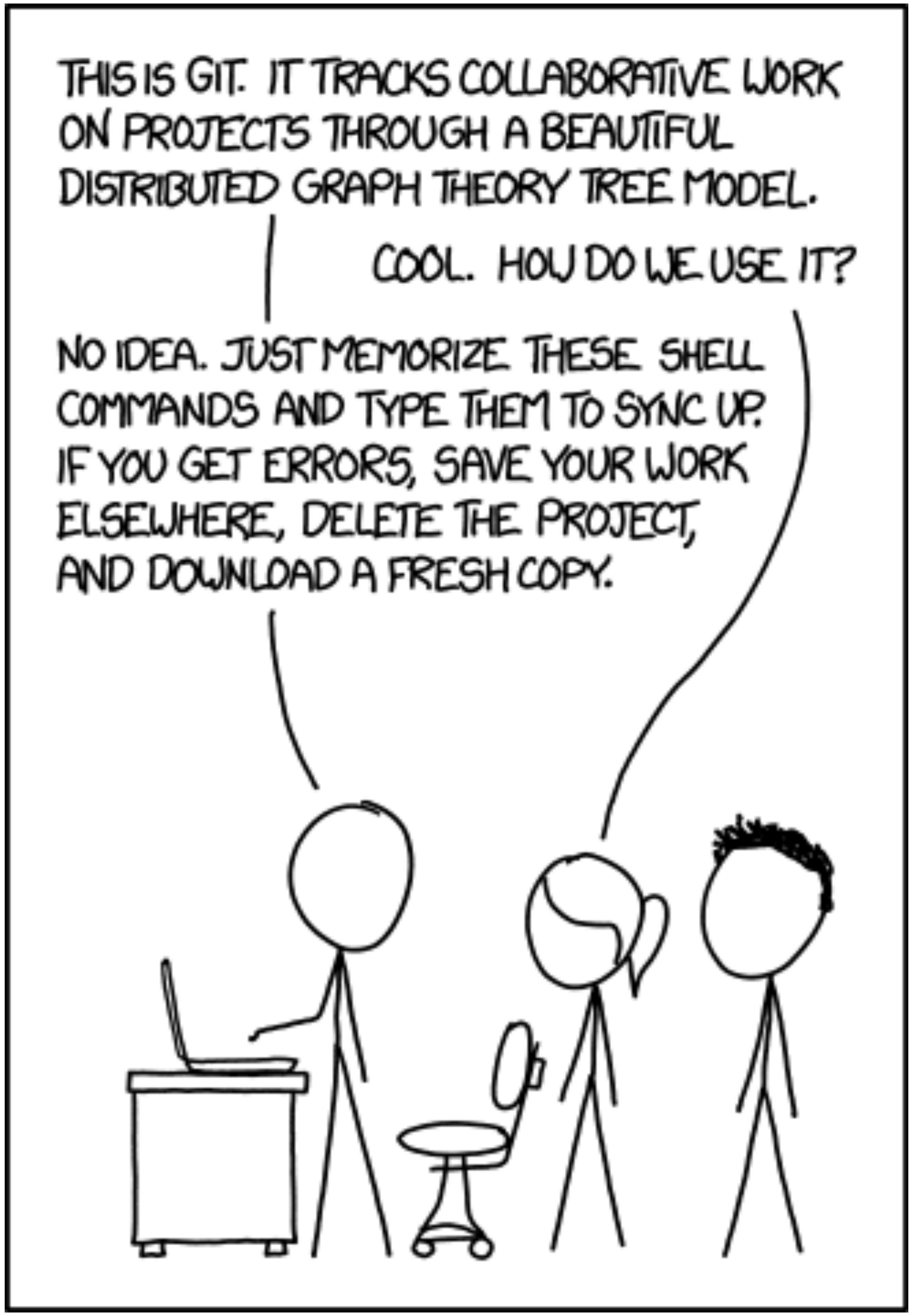
Distributed Version Control

David Grellscheid

UNIVERSITETET I BERGEN


60°N
55°N
48°N
ICTP
Iceland
Ireland
United Kingdom
Netherlands
Belgium
Luxembourg
Germany
France
Switzerland
Liecht.
Monaco
Andorra
Portugal
Spain
Den-mark
Norway
Sweden
Finland
Russia
Estonia
Latvia
Lithuania
Belarus
Poland
Czech Republic
Slovakia
Austria
Hungary
Ukraine
Mol-dova
Romania
Slove-nia
Croatia
Bosnia and Herzego-vina
Serbia
Monte-negro
Al-ba-nia
Mace-donia
Bulgaria
Greece
San Marino
Italy
Vatican City
Malta
Turkey
Georgia
Armenia
Azerbaijan
Kazakhstan
Iran
Syria
Iraq
Cyprus
Leba-non
Israel
Jordan
Saudi Arabia
Morocco
Algeria
Tunisia


THIS IS GIT. IT TRACKS COLLABORATIVE WORK ON PROJECTS THROUGH A BEAUTIFUL DISTRIBUTED GRAPH THEORY TREE MODEL.
COOL. HOW DO WE USE IT?
NO IDEA. JUST MEMORIZE THESE SHELL COMMANDS AND TYPE THEM TO SYNC UP. IF YOU GET ERRORS, SAVE YOUR WORK ELSEWHERE, DELETE THE PROJECT, AND DOWNLOAD A FRESH COPY.

xkcd 1597

Version management

In any project, there are tedious bookkeeping tasks that need to be done:

* Backup of consistent project snapshots
* Documentation of changes
* Sharing of changes
* Distributed development by multiple people and/or in multiple locations
* Bug tracing through development history

Task automation

The bookkeeping can be done

- by hand: copy things into renamed files
- locally: keep versions in a DB of some form
- centrally: a server keeps the DB, clients do the work
- distributed: each client holds a full copy of the DB

Tool history

Only showing commonly used free/OSS tools:

- local:
  RCS (1982)
- central:
  CVS (1990), SVN (2000)
- distributed:
  Darcs (2002), Bazaar, Git, Mercurial (all 2005)

Frontends and visualization

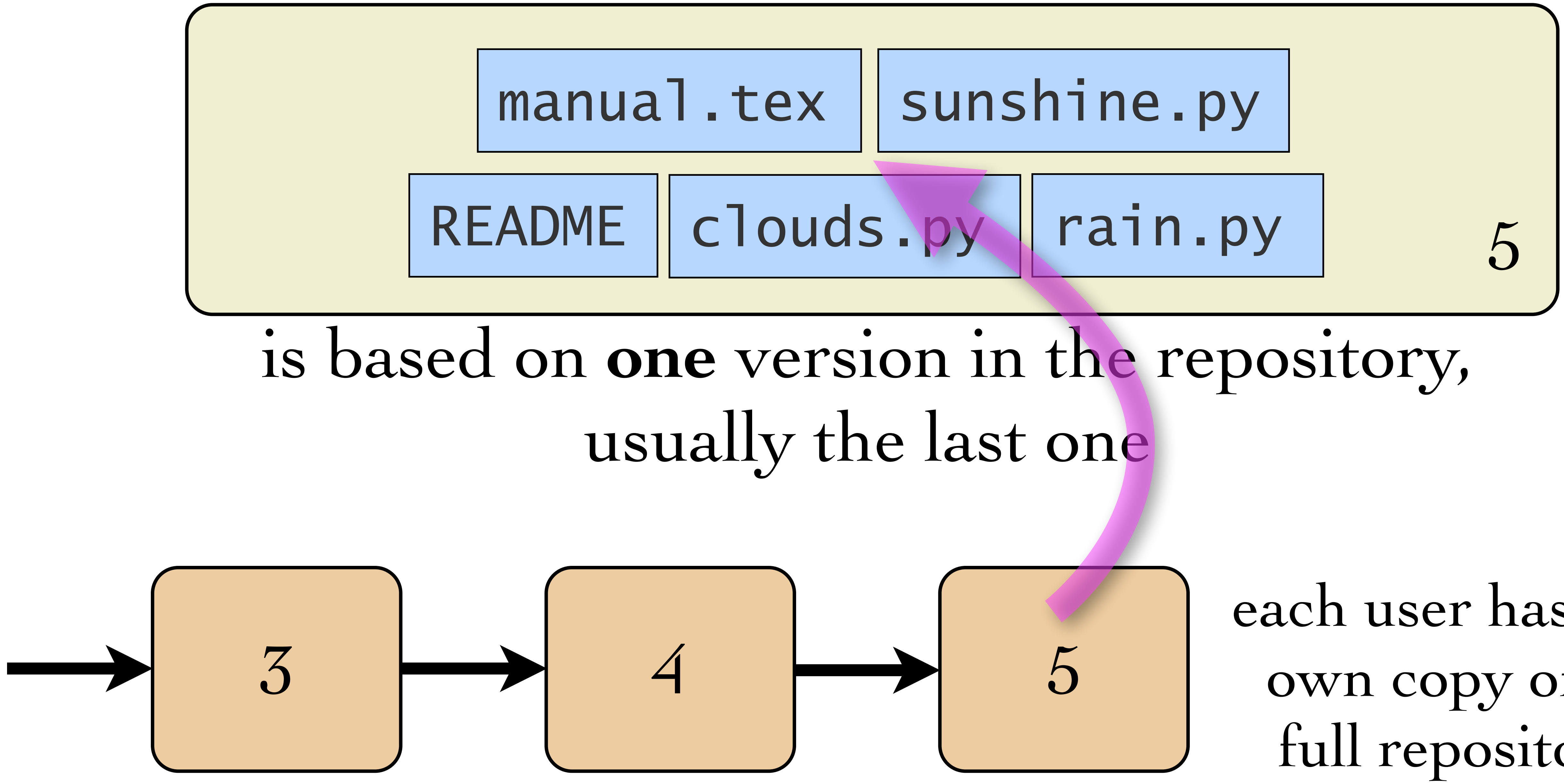
- Mercurial has built-in `hg serve`
- Many other GUI frontends available,
  try a few to find something you like
- Github / Bitbucket / Gitlab additionally give full project management tools, but keep a second repo elsewhere!

Terminology

* Working copy
* Repository (local, remote)
* checkout, checkin — update, commit
* pull, push
* branch, merge

Working directory

one version of the project, visible to be edited

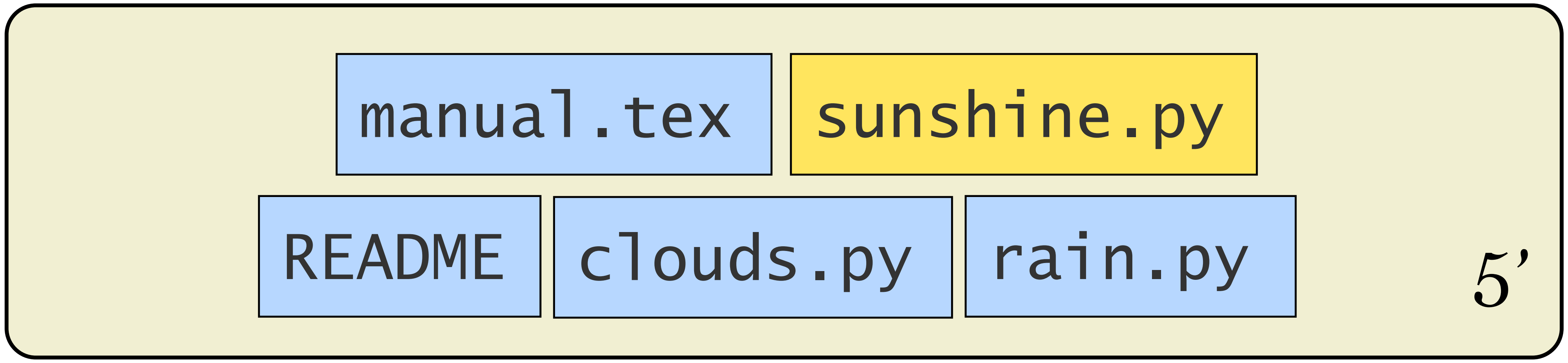


is based on **one** version in the repository, usually the last one

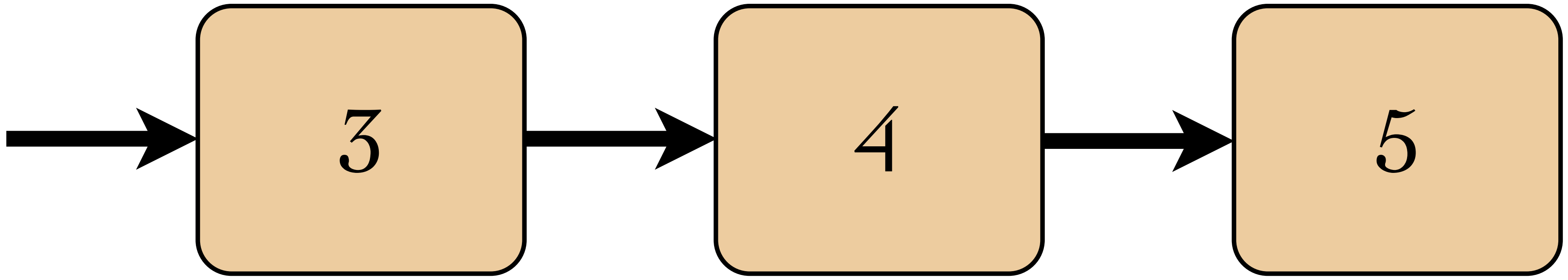
each user has their own copy of the full repository!

Changes

working copy can be changed as usual

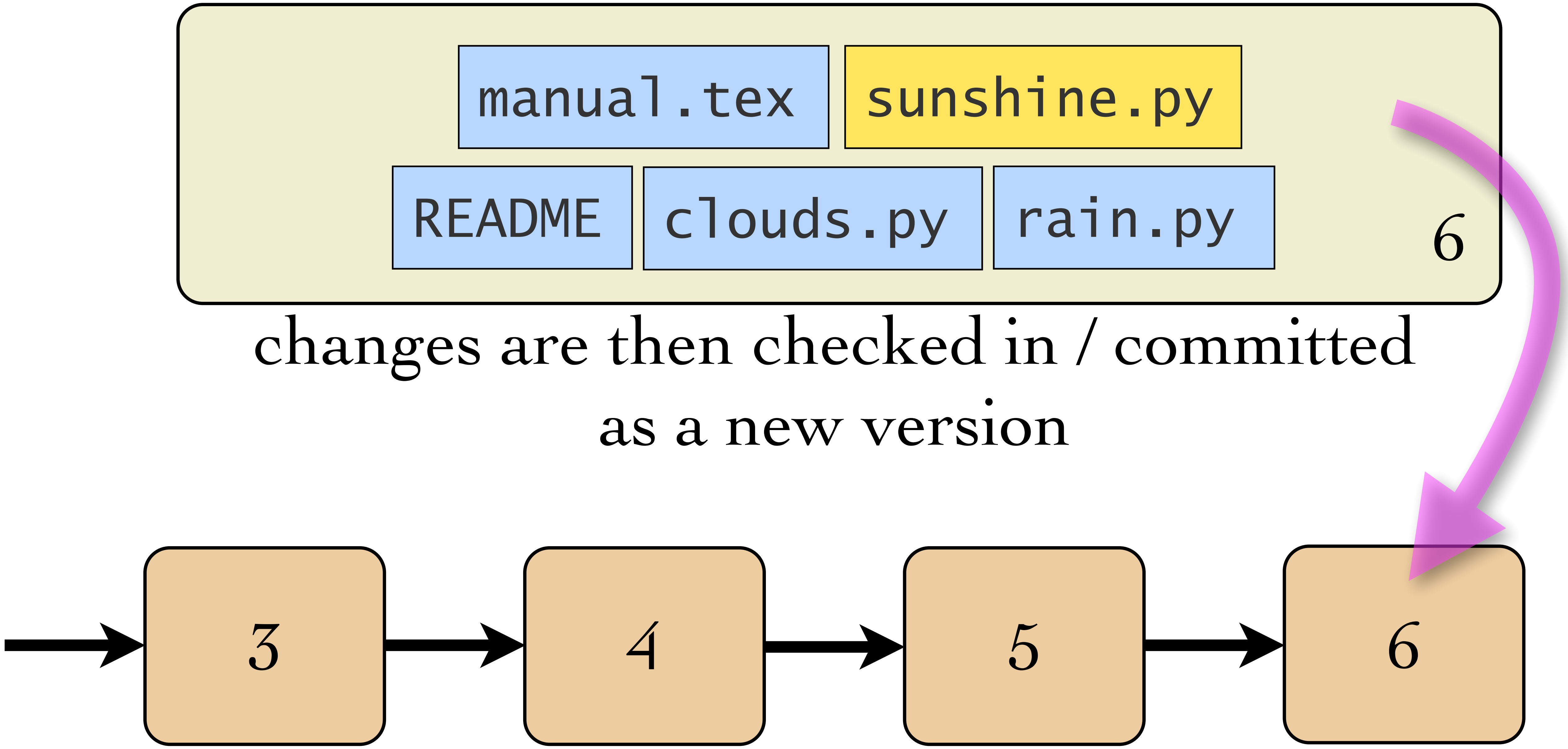


changes are then checked in / committed as a new version



Changes

working copy can be changed as usual

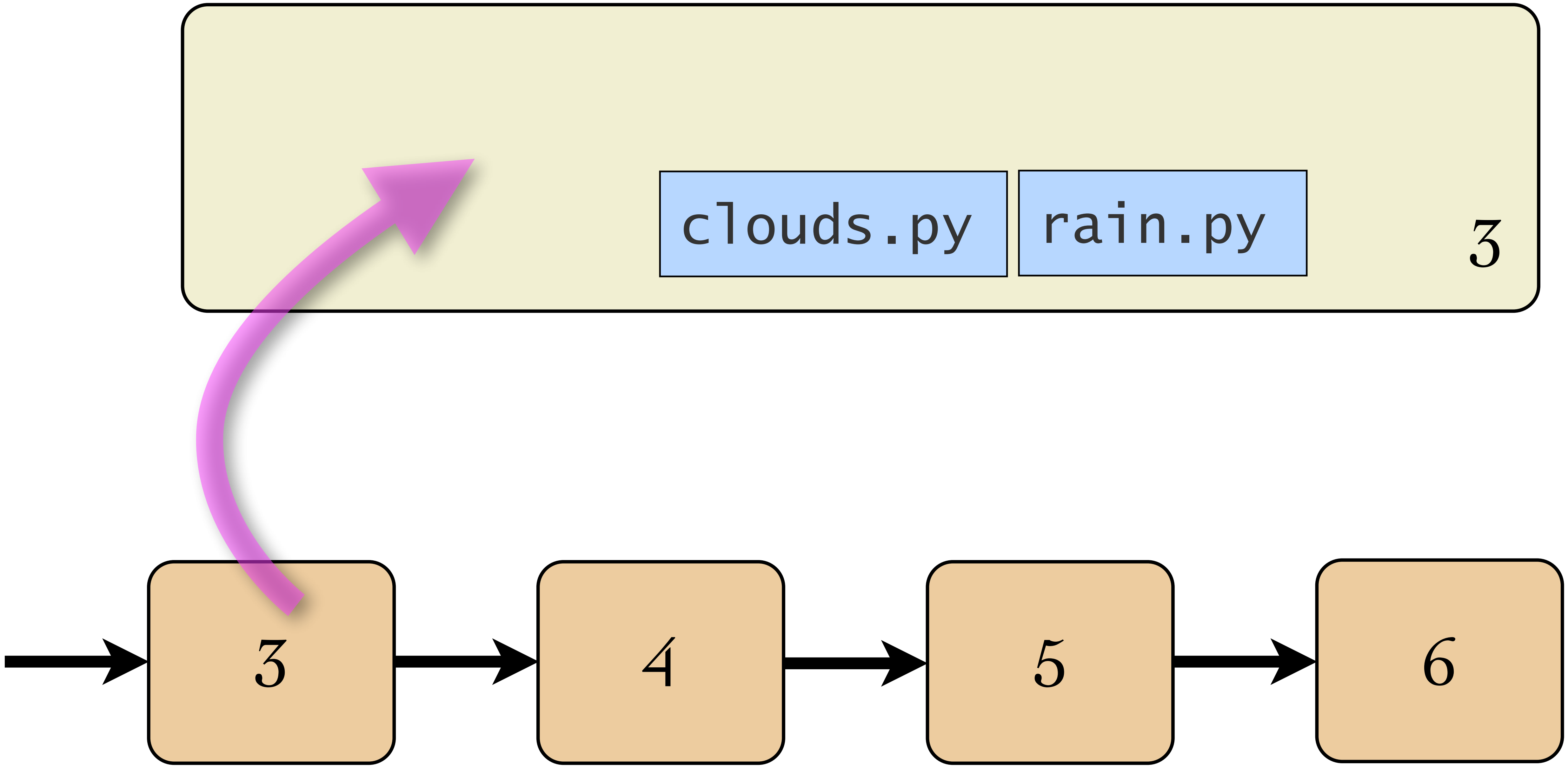


changes are then checked in / committed as a new version

changeset

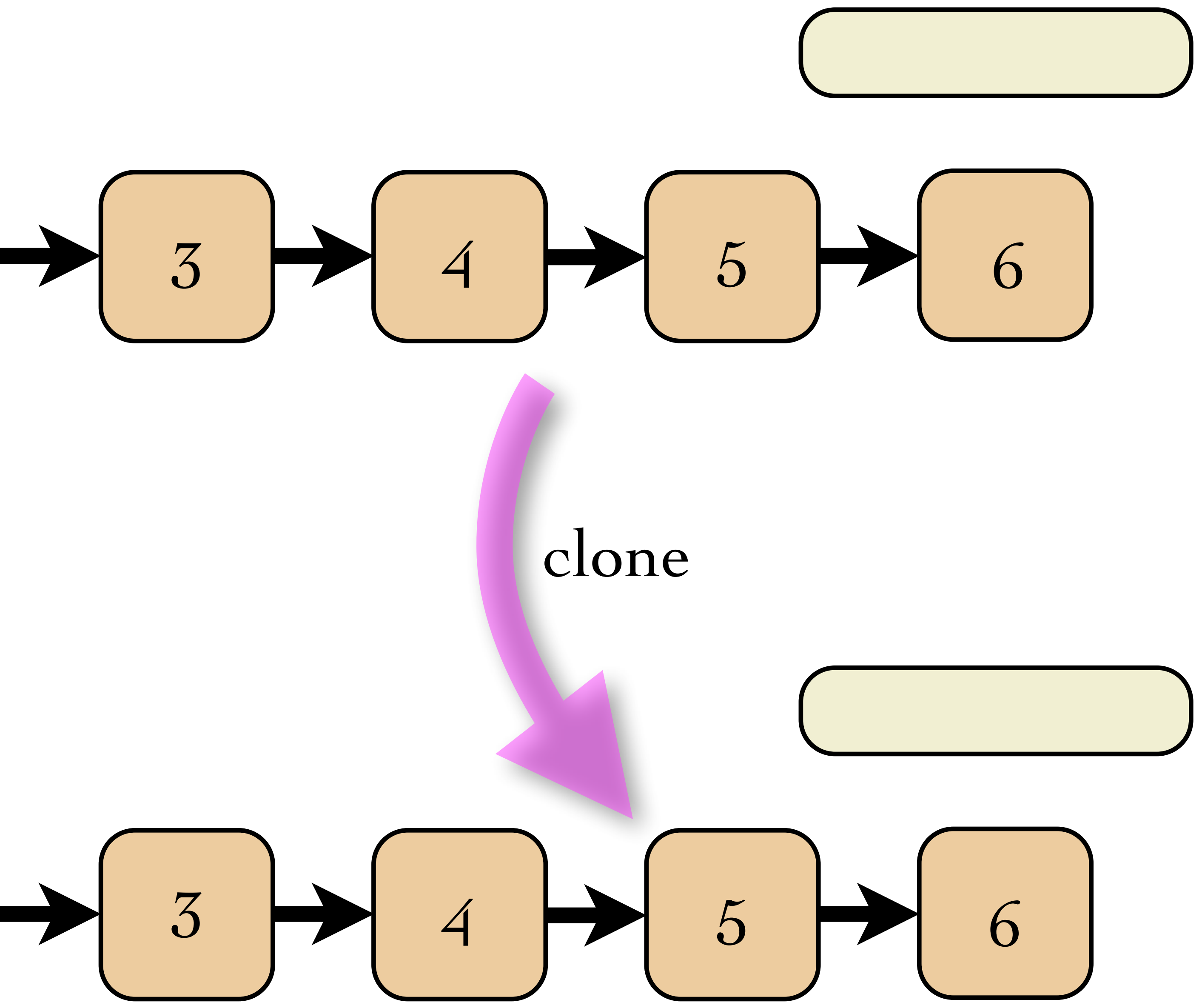
Backtracking

older versions can be looked at at any time



Remote work

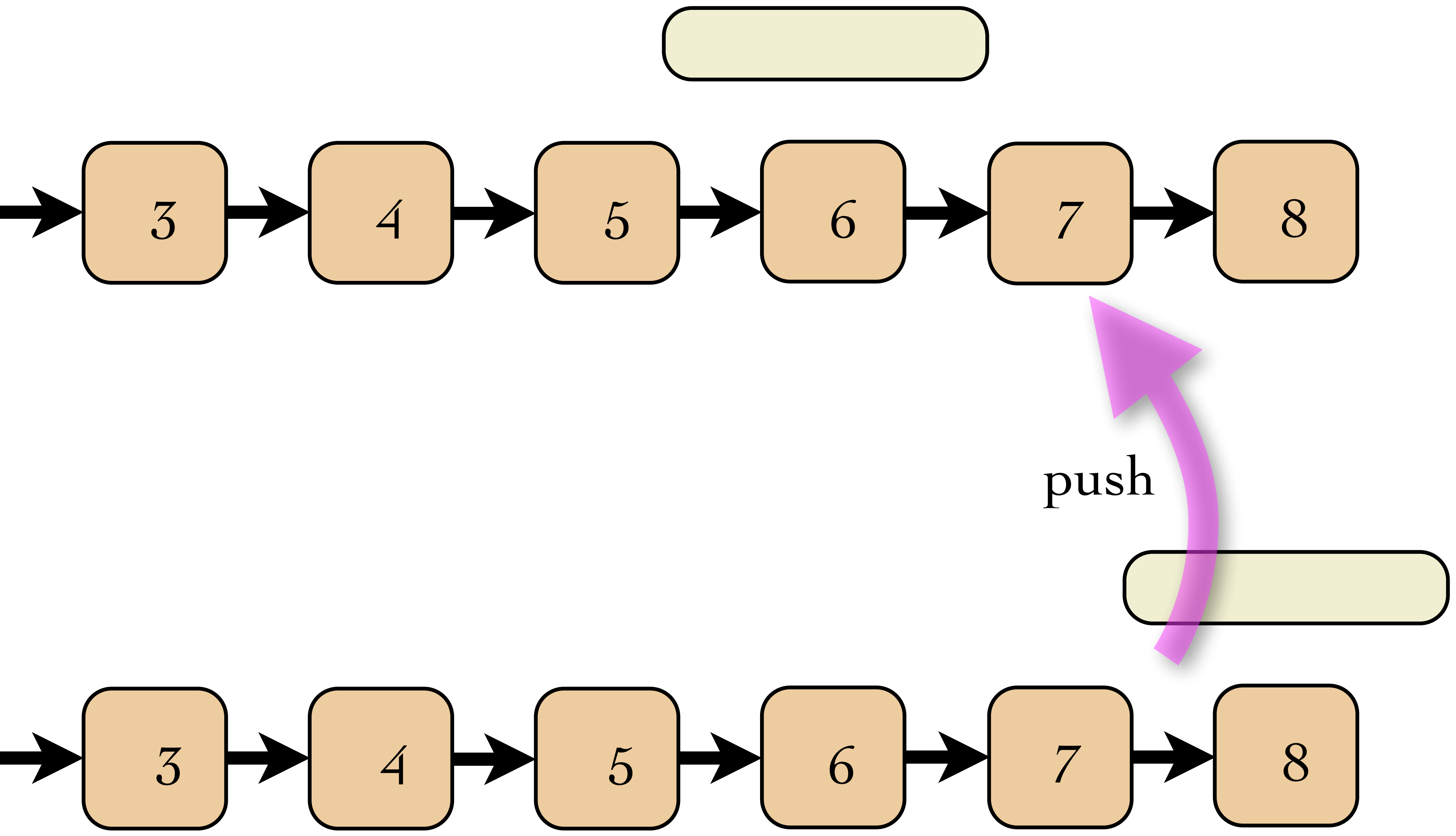
clone repositories to other locations / devices / people



each clone has all features,
not distinguishable

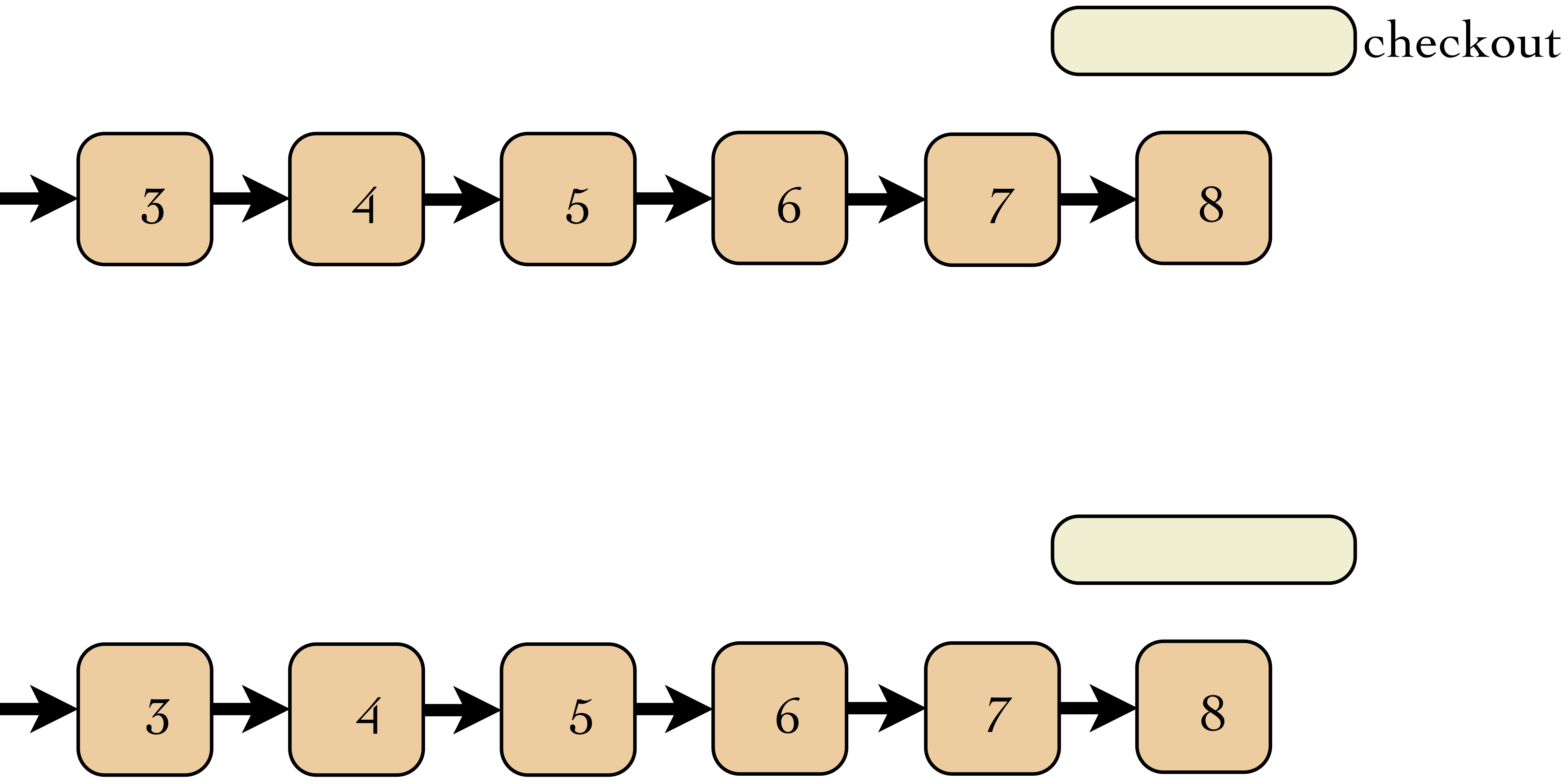
Remote work

development can happen anywhere



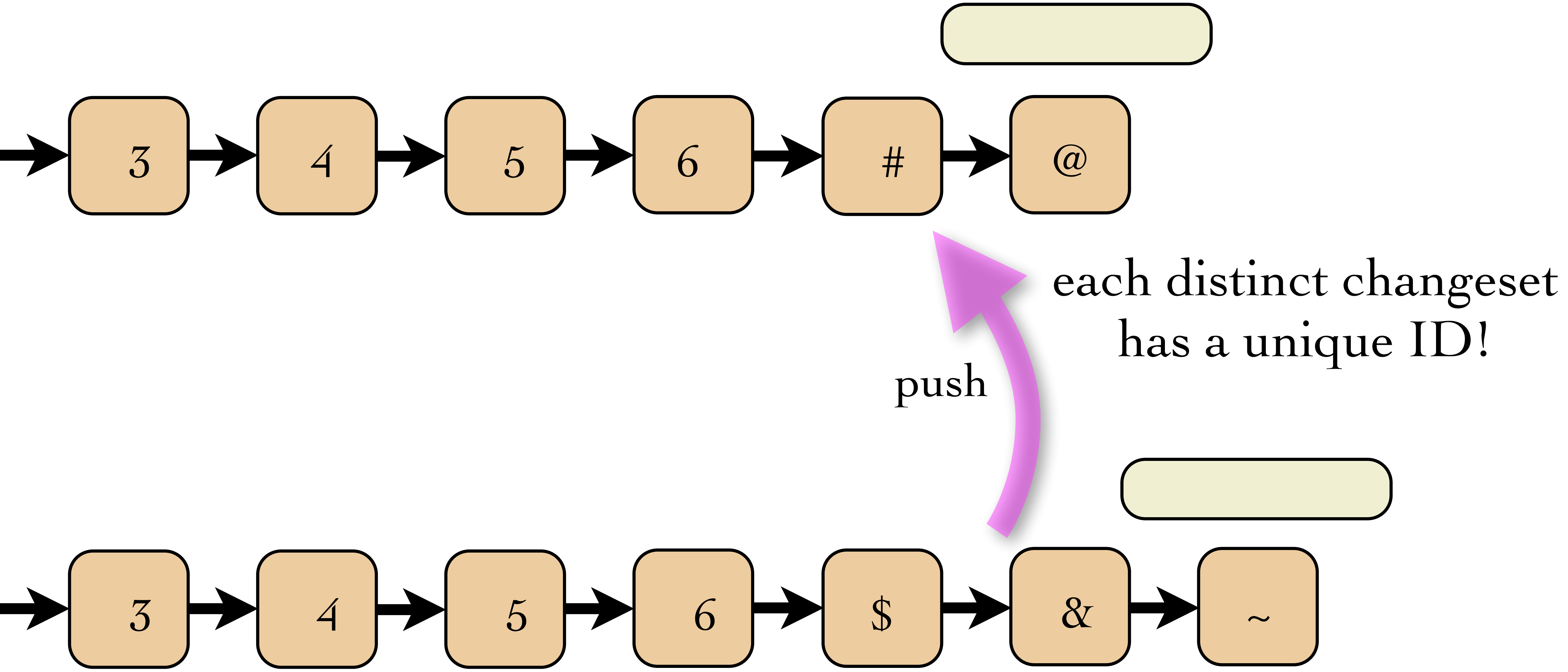
Remote work

development can happen anywhere



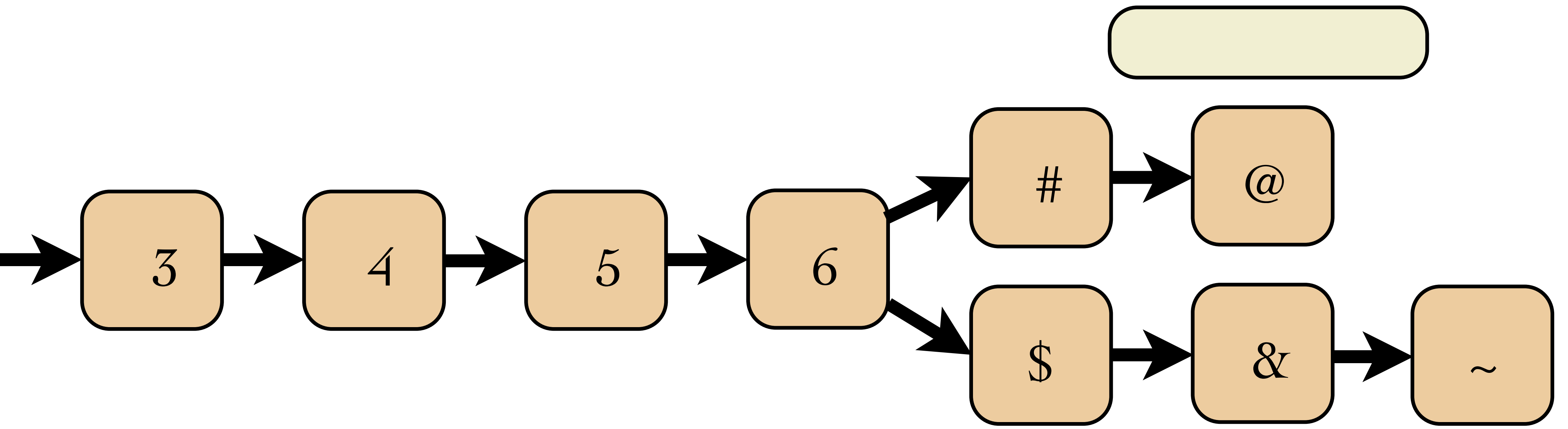
Branches

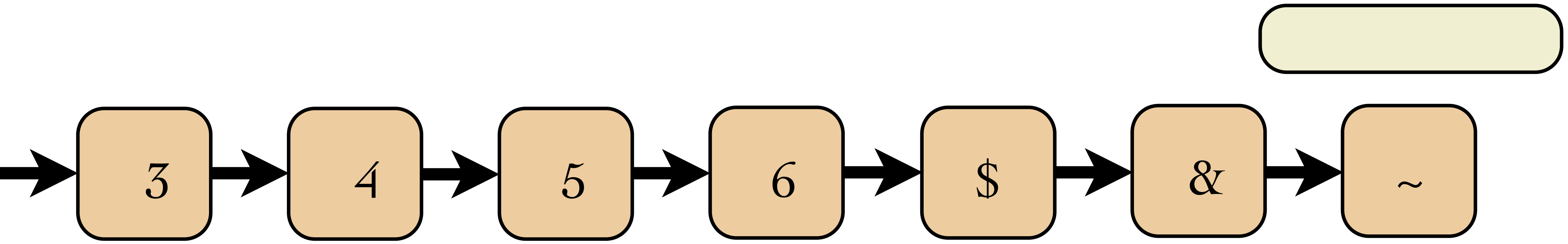
unlike in SVN, branches are a natural feature



Branches

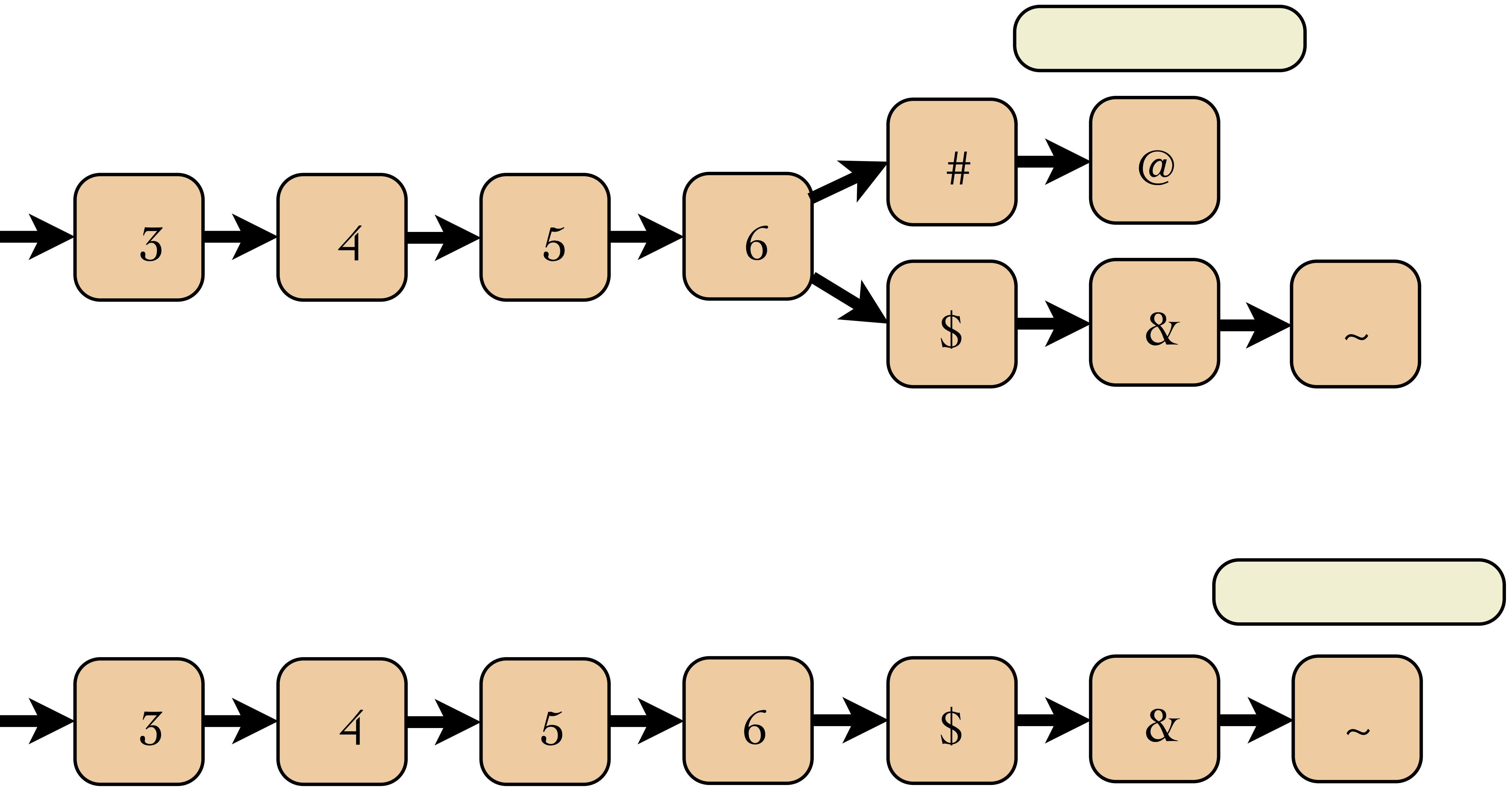
unlike in SVN, branches are a natural feature





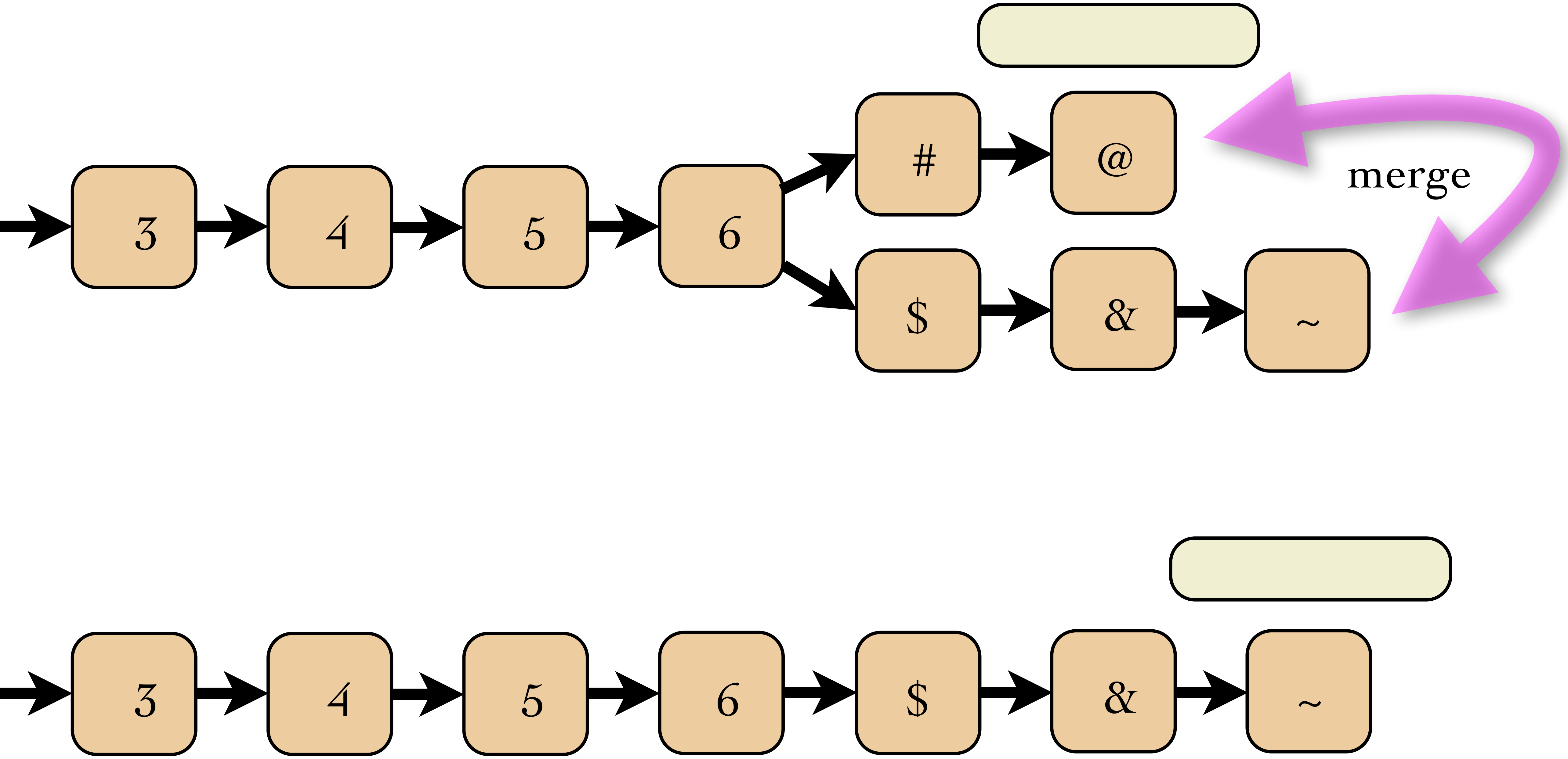
Merging

merging is automatic as far as possible



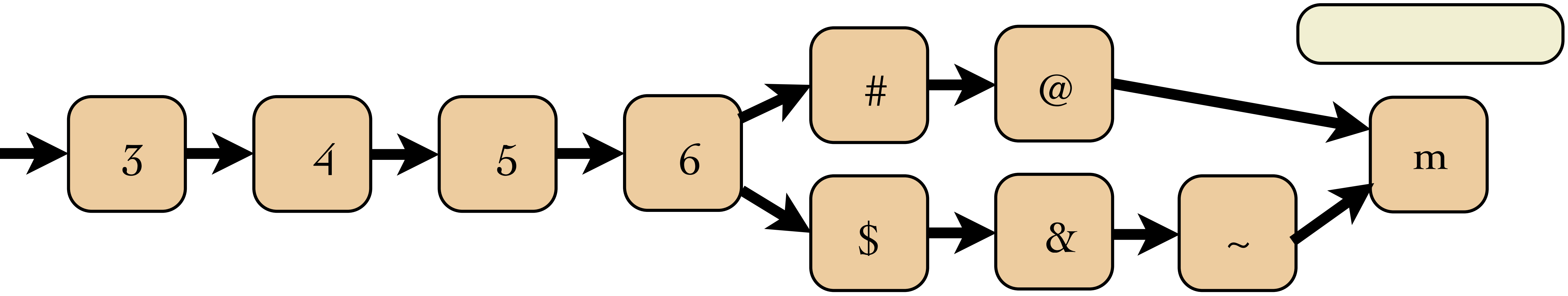
Merging

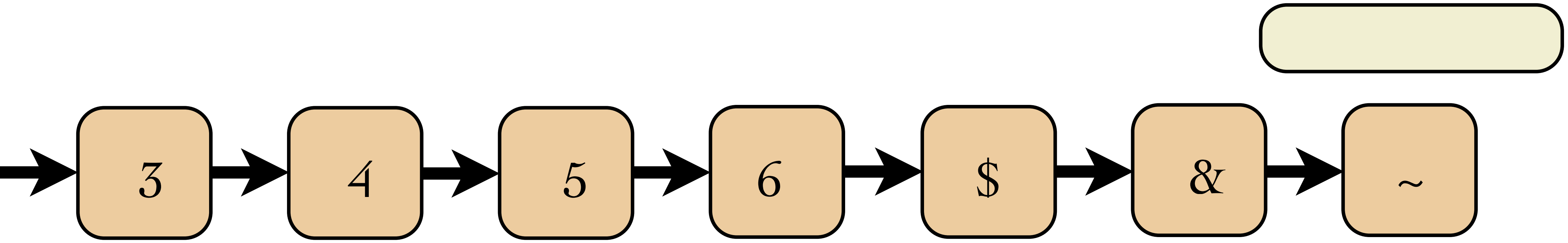
merging is automatic as far as possible



Merging

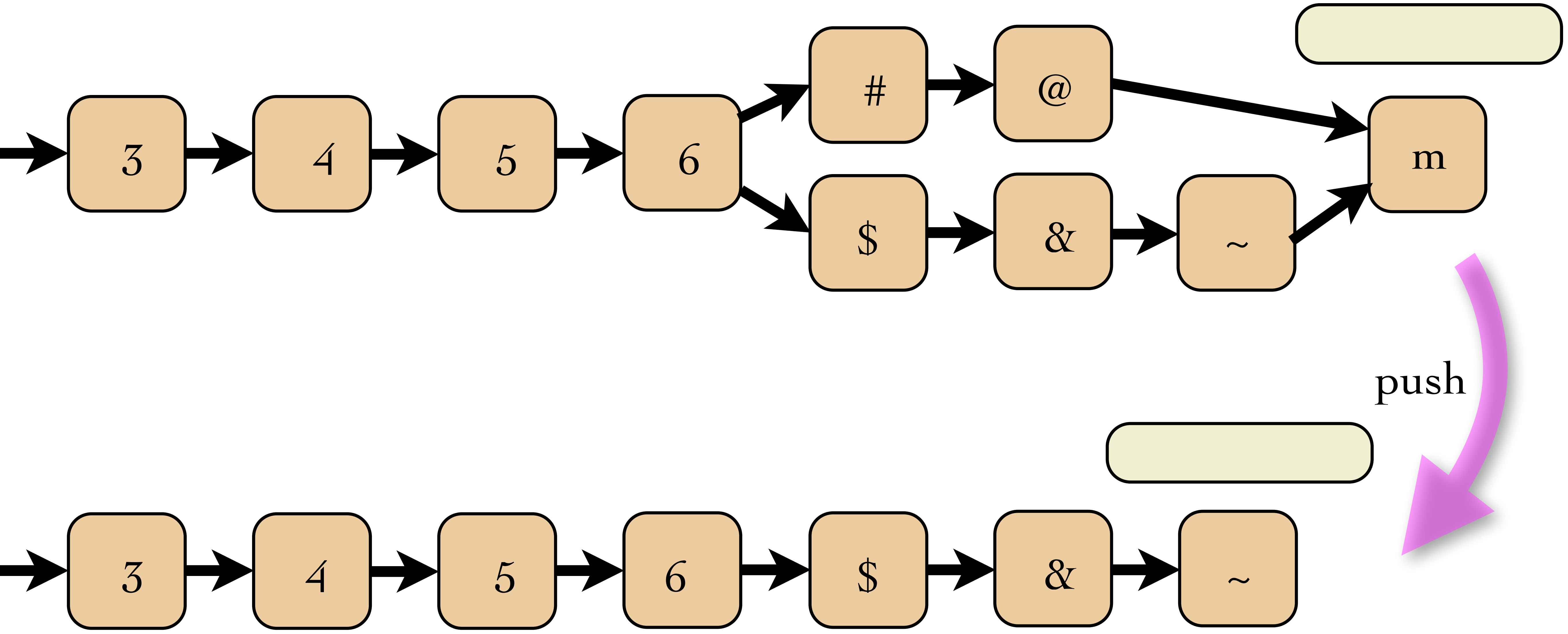
only need to fix some conflicts by hand





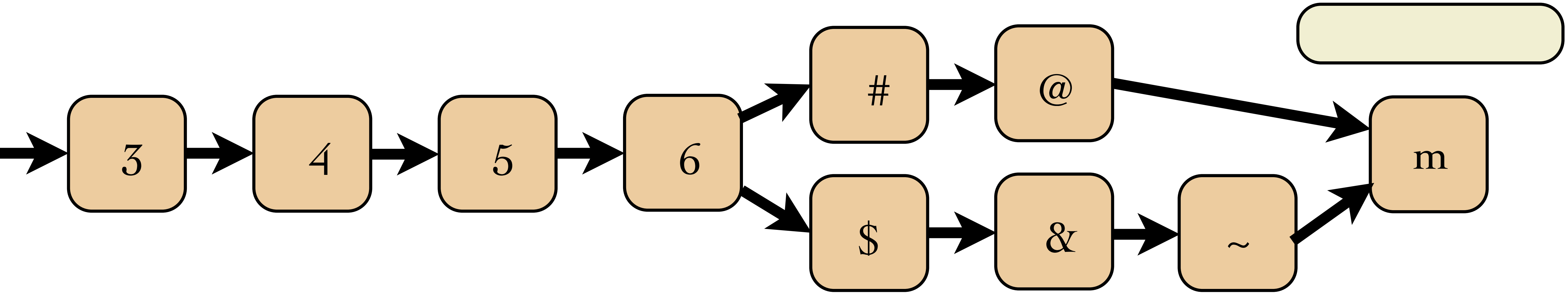
Merging

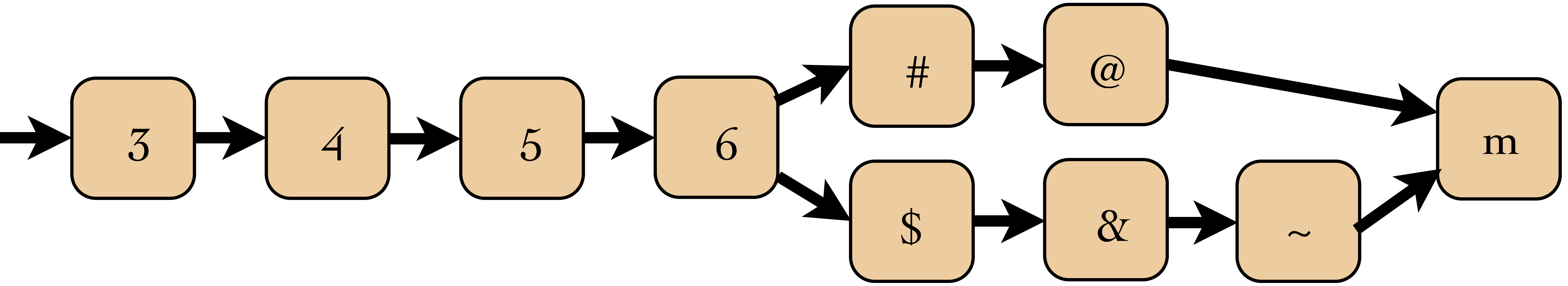
only need to fix some conflicts by hand



Merging

consistent state after push



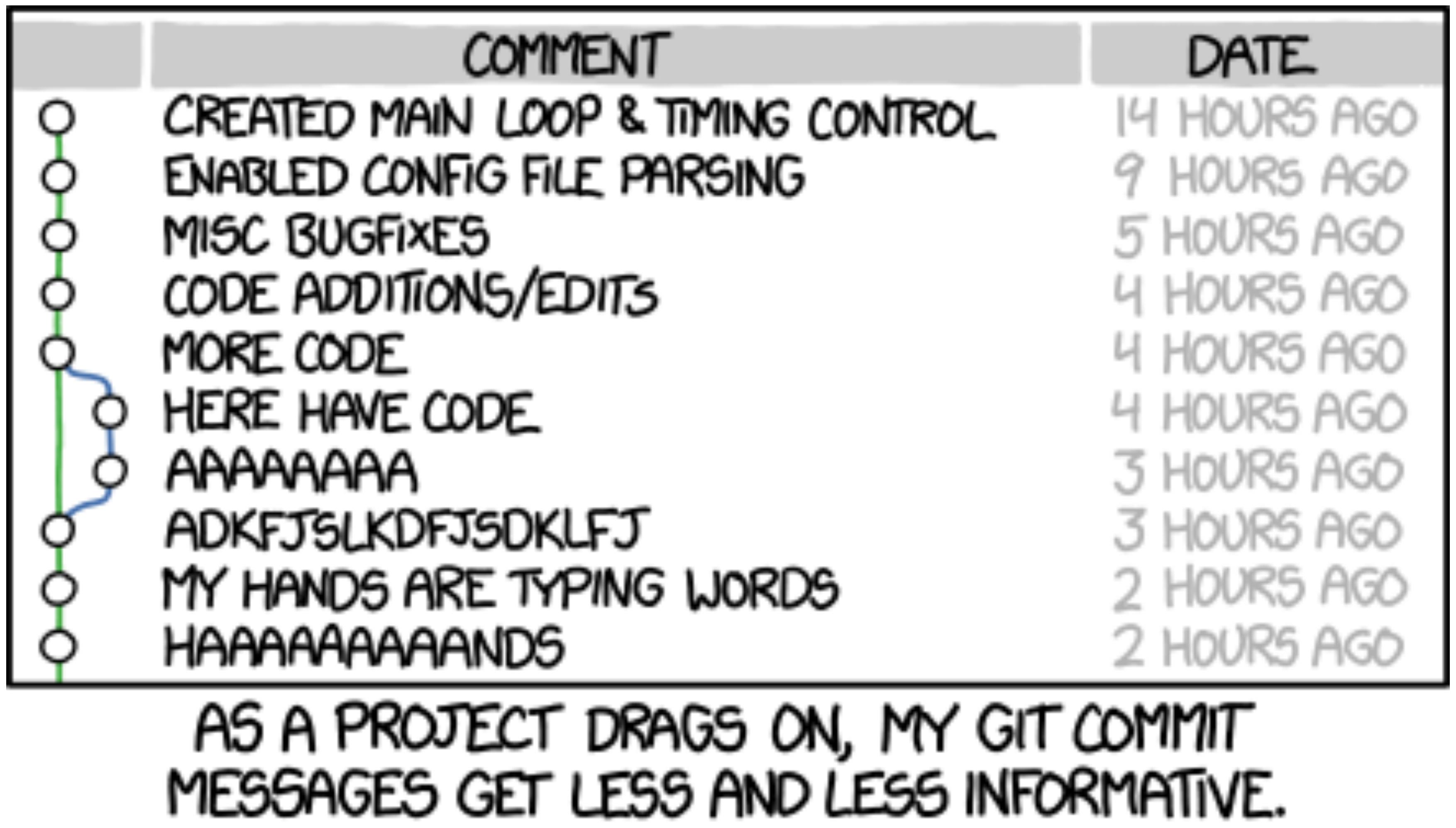


Merging

consistent state after push

$$\begin{tikzcd}
 & & & & \# \arrow[r] & @ \arrow[drr] & & \\
3 \arrow[r] & 4 \arrow[r] & 5 \arrow[r] & 6 \arrow[ur] \arrow[dr] & & & & m \\
 & & & & \$ \arrow[r] & \& \arrow[r] & \sim \arrow[ur] &
\end{tikzcd}$$

$$\begin{tikzcd}
 & & & & \# \arrow[r] & @ \arrow[drr] & & \\
3 \arrow[r] & 4 \arrow[r] & 5 \arrow[r] & 6 \arrow[ur] \arrow[dr] & & & & m \\
 & & & & \$ \arrow[r] & \& \arrow[r] & \sim \arrow[ur] &
\end{tikzcd}$$



	COMMENT	DATE
○	CREATED MAIN LOOP & TIMING CONTROL	14 HOURS AGO
○	ENABLED CONFIG FILE PARSING	9 HOURS AGO
○	MISC BUGFIXES	5 HOURS AGO
○	CODE ADDITIONS/EDITS	4 HOURS AGO
○	MORE CODE	4 HOURS AGO
○	HERE HAVE CODE	4 HOURS AGO
○	AAAAAAAA	3 HOURS AGO
○	ADKFJSLKDFJSDKLFJ	3 HOURS AGO
○	MY HANDS ARE TYPING WORDS	2 HOURS AGO
○	HAAAAAAAANDS	2 HOURS AGO

AS A PROJECT DRAGS ON, MY GIT COMMIT MESSAGES GET LESS AND LESS INFORMATIVE.

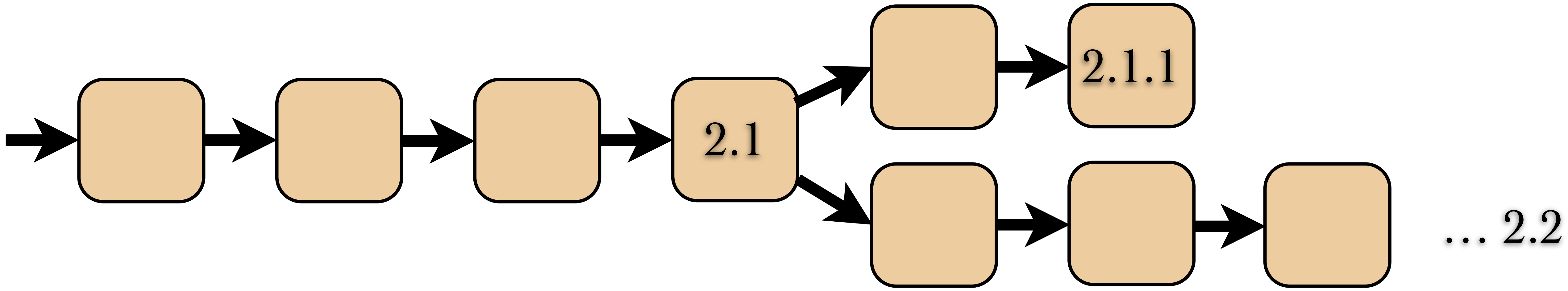
xkcd 1296

Typical usage patterns

- Technically, every repo is the same
- By agreement, designate one repo as “central”
- Tarballs or other distributions are built *only* from there
- Interaction with central repo: only `push/pull` from developers, no direct commits
- Can pipeline even more: put code-review, tester, build manager repos in between

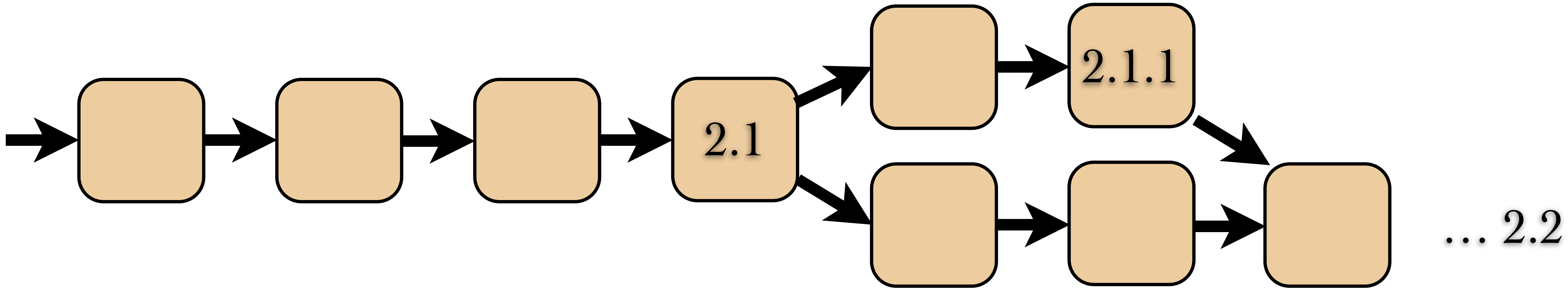
Typical usage patterns

* Think about release lifetime
* Typically, separate release and feature branches
* Carefully put checkins in the right place!



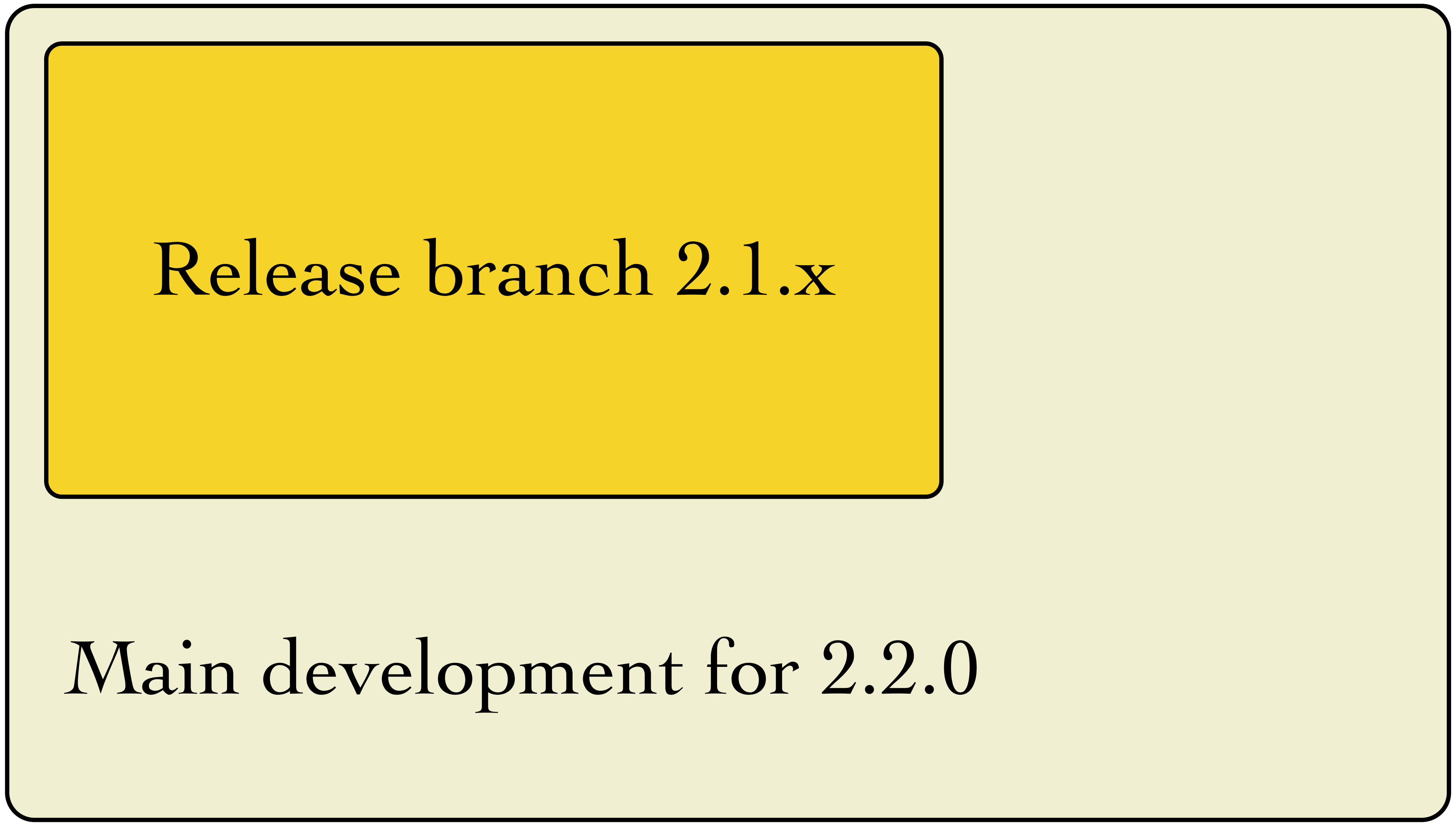
Typical usage patterns

- Think about release lifetime
- Typically, separate release and feature branches
- Carefully put checkins in the right place!



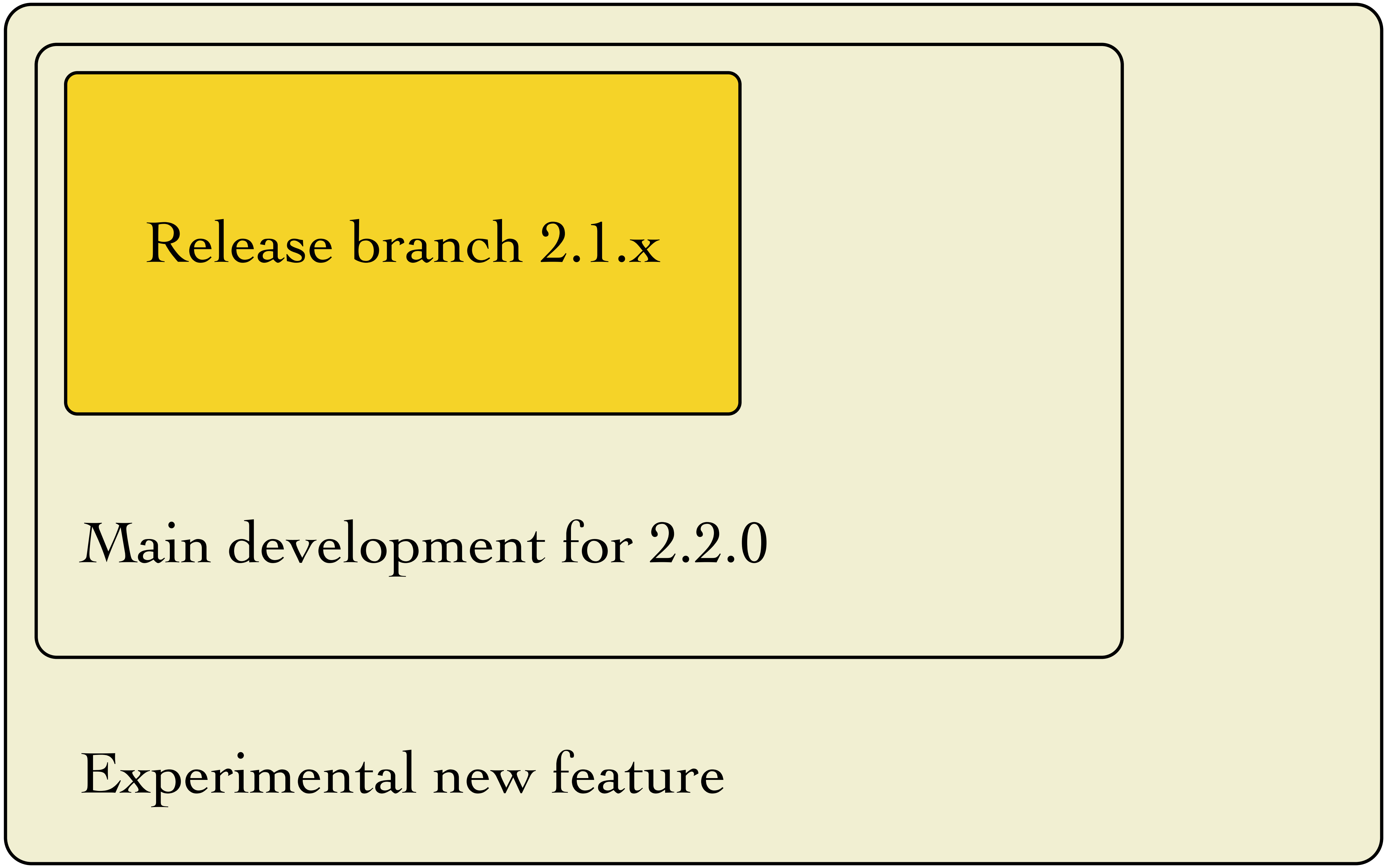
Which branch for which checkin?

- Pick the innermost appropriate branch



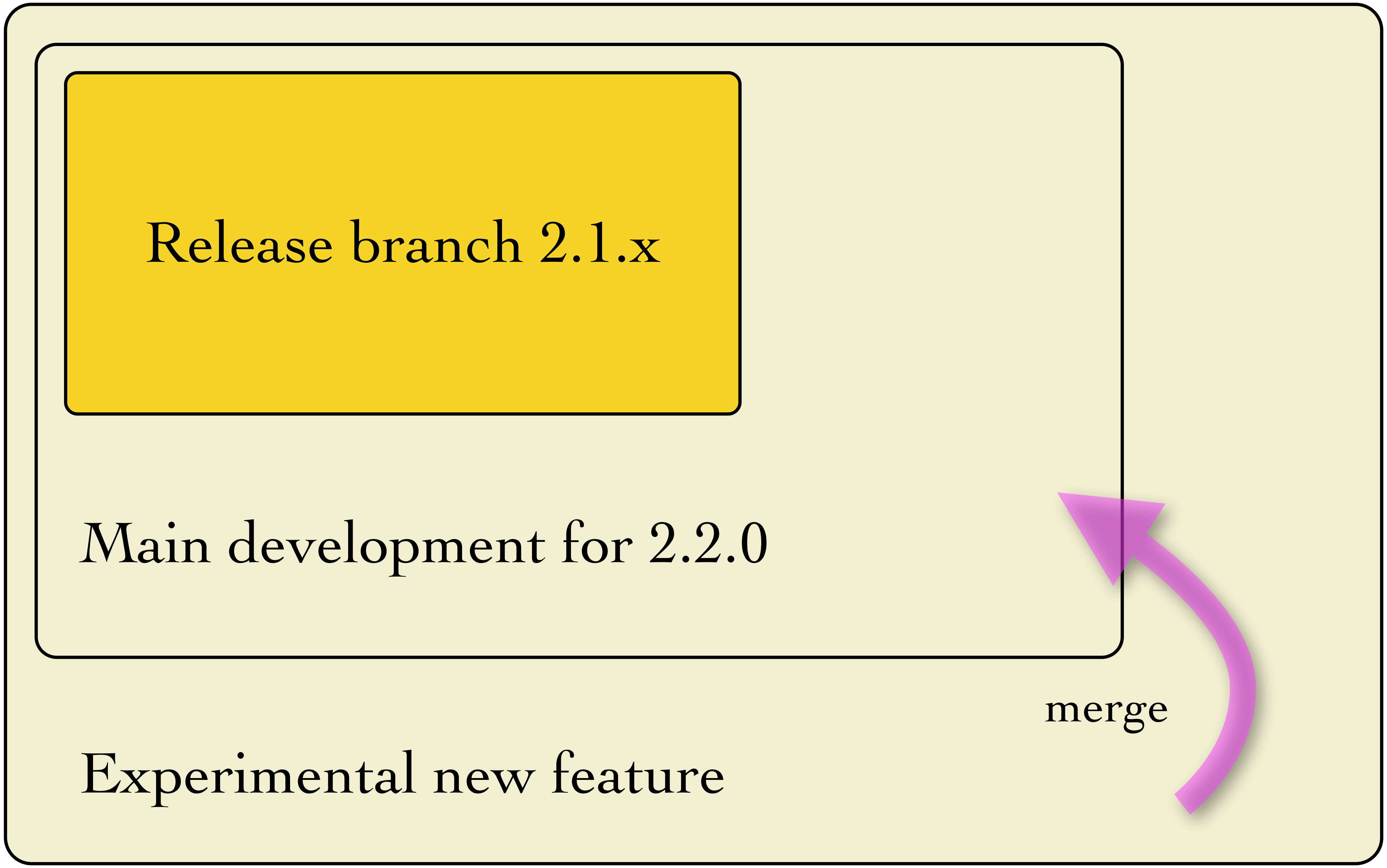
Which branch for which checkin?

* Pick the innermost appropriate branch



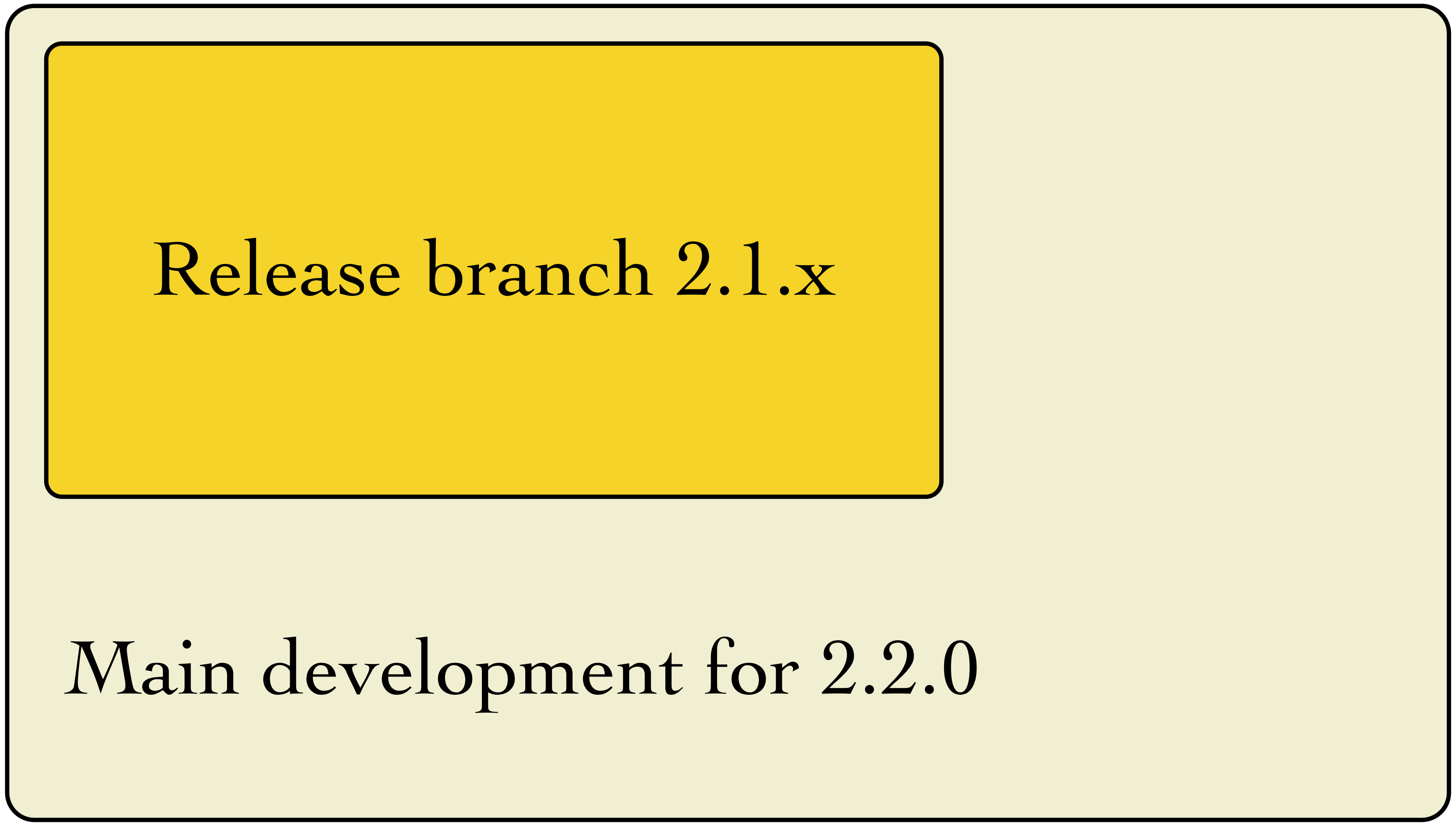
Which branch for which checkin?

* Pick the innermost appropriate branch



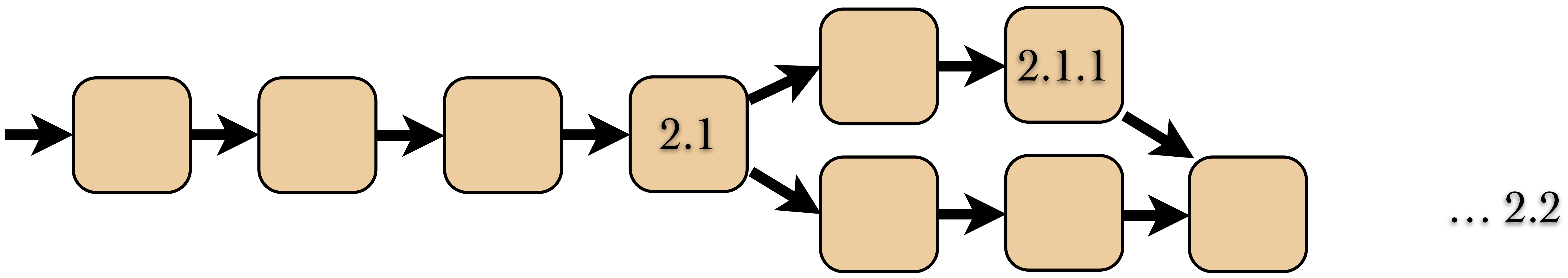
Which branch for which checkin?

- Pick the innermost appropriate branch



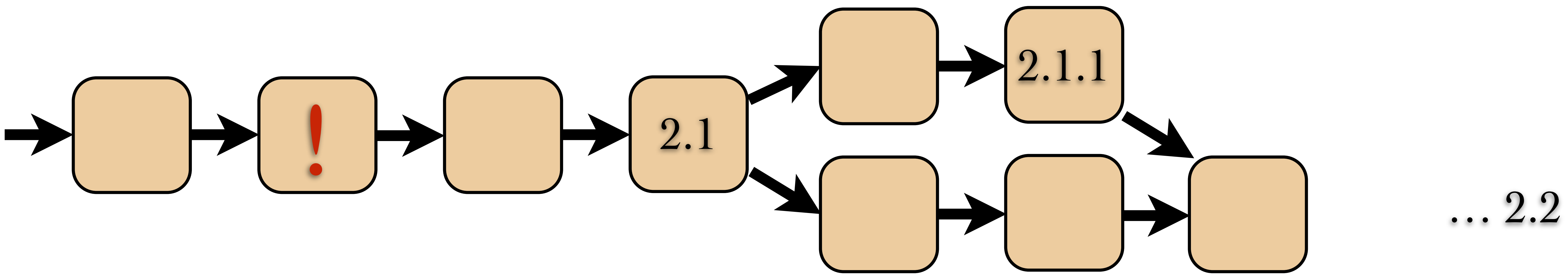
Typical usage patterns

If a bugfix is localizable, fix it **there!**



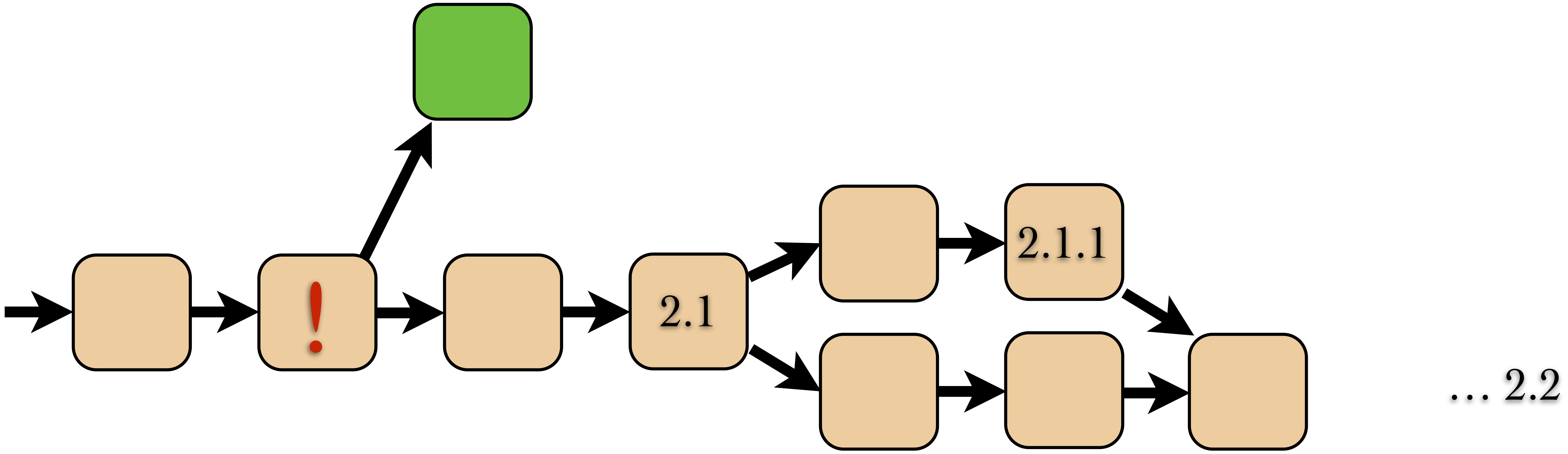
Typical usage patterns

If a bugfix is localizable, fix it **there!**



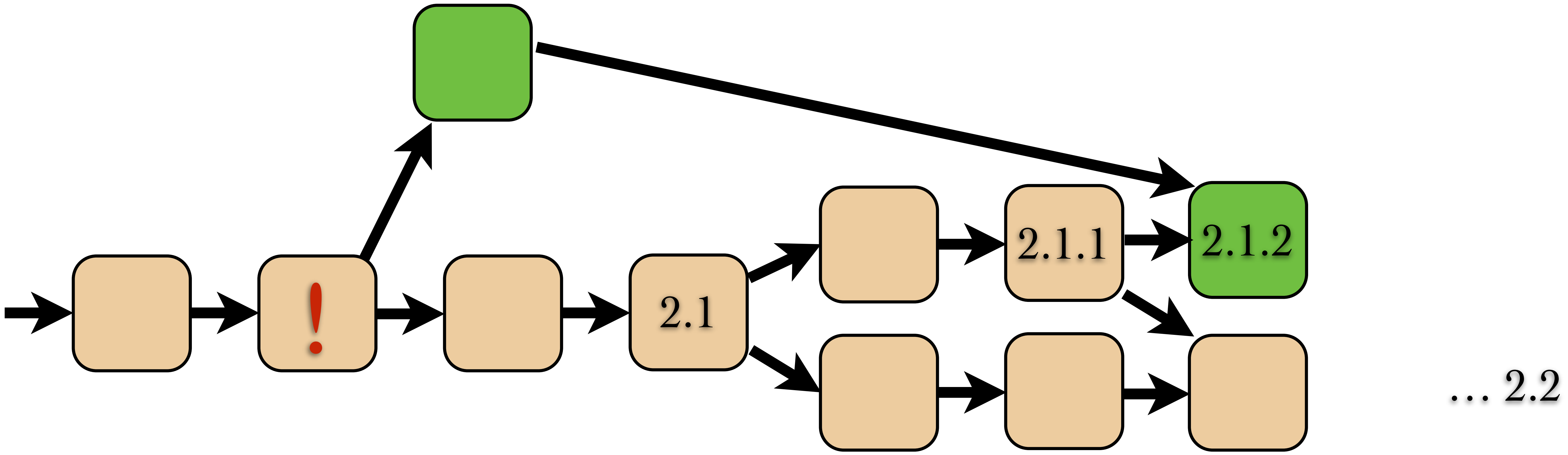
Typical usage patterns

If a bugfix is localizable, fix it **there!**



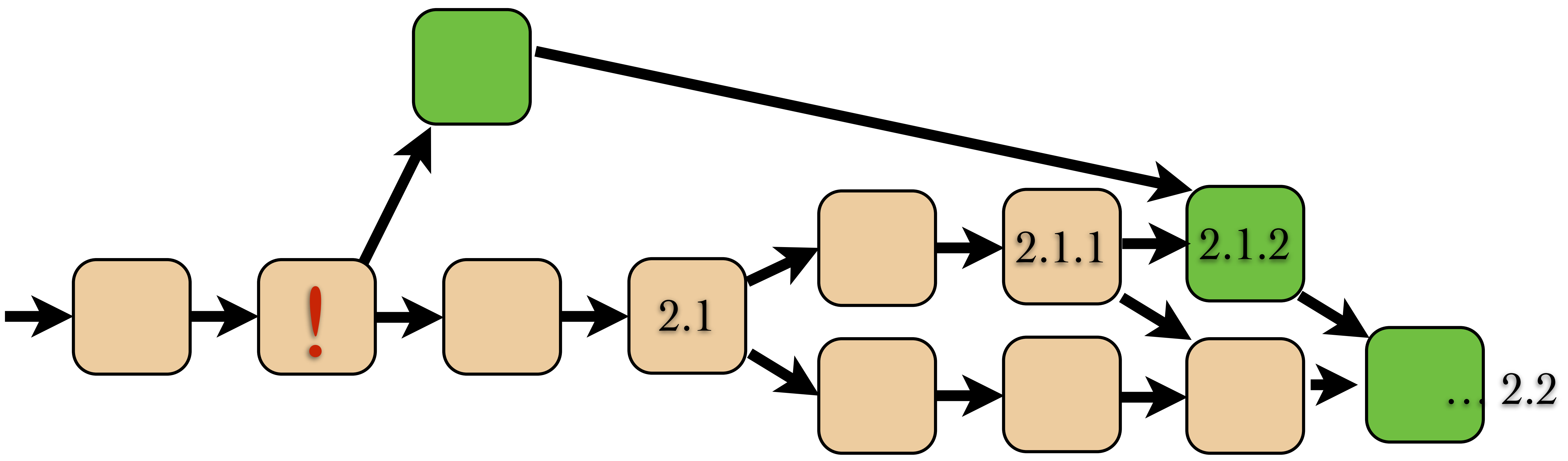
Typical usage patterns

If a bugfix is localizable, fix it **there!**



Typical usage patterns

If a bugfix is localizable, fix it **there!**



Summary

Distributed VCS are so easy to use that there's no reason not to do so!

Don't even think of writing outside version control!

- Configure your git settings:
    - $ git config --global user.name "[name]"
    - $ git config --global user.email "[email address]"
    - $ git config --global color.ui auto
- More options:
    - $ git config --global core.editor “editor name”
    - $ git config --global -e
    - $ git config -h
    - $ git config --help

- Initiate git in a directory:
    - $ git init
- Make some files:
    - $ git status
    - $ git add
- Committing to changes:
    - $ git commit
    - $ git commit -a

- Now make a change and go through the process again!

- Try:
  - `$ git status -s`
- Add a file to the staging environment
  - `$ git diff`
- Try:
    - `$ git diff -h`
    - `$ git diff --stat`

- Looking at the changes
  - $ git log
  - $ git log -3
  - $ git log -p
  - $ git log --stat --summary
  - $ git log --follow [file]
  - $ git log --oneline
  - $ git log --after 2017-07-04
  - $ git log —author=“ali”
  - $ git log —grep=" word of phrase to search”

- $ git show 1b2e1d63ff (some identifier)
- $ git show HEAD
- $ git show HEAD~1
- $ git show HEAD~2:file1.txt

- $ rm file2.txt
- $ git add file2.txt

$ git rm file2.txt

$ git commit -m “”

- Rename files
  - $ mv file3.txt main.cpp
  - $ git add file3.txt
  - $ git add main.cpp

$ git mv file3.txt main.cpp

- Made changes to file1.txt
  - $ git restore file1.txt
- Added changes to file1.txt to the staging area
  - $ git restore --staged file1.txt
  - $ git restore file1.txt
- Restoring deleted files
  - $git rm file1.cpp; $ git commit
  - $ git restore --source=HEAD~1 file1.cpp
  - [more options] $ git restore -h