Distributed Version Control

David Grellscheid

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

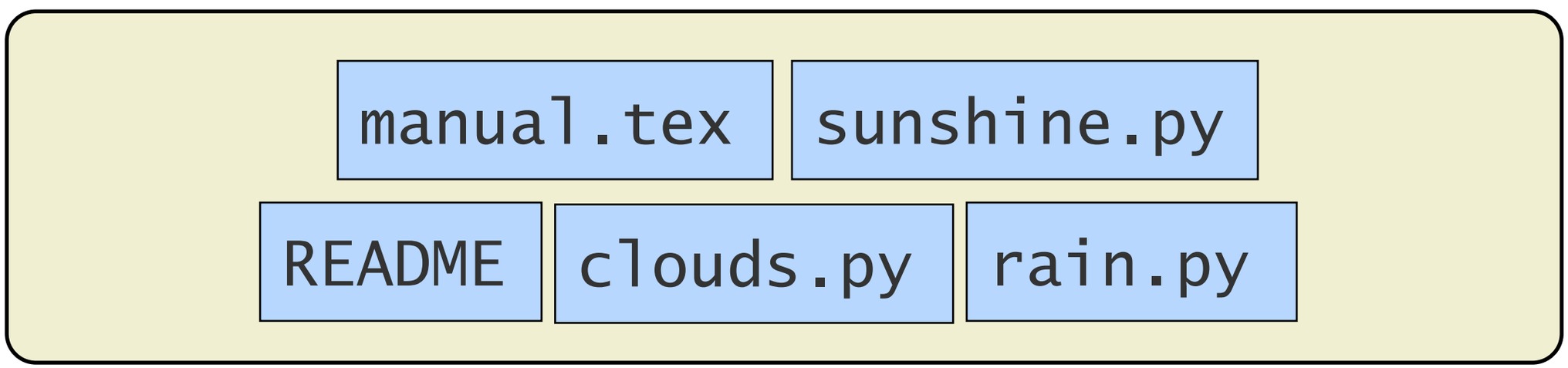
* Mercurial has built-in `hg serve`
* Many other GUI frontends available, try a few to find something you like
* Github / Bitbucket additionally give full project management tools, but keep a second repo elsewhere!

Terminology

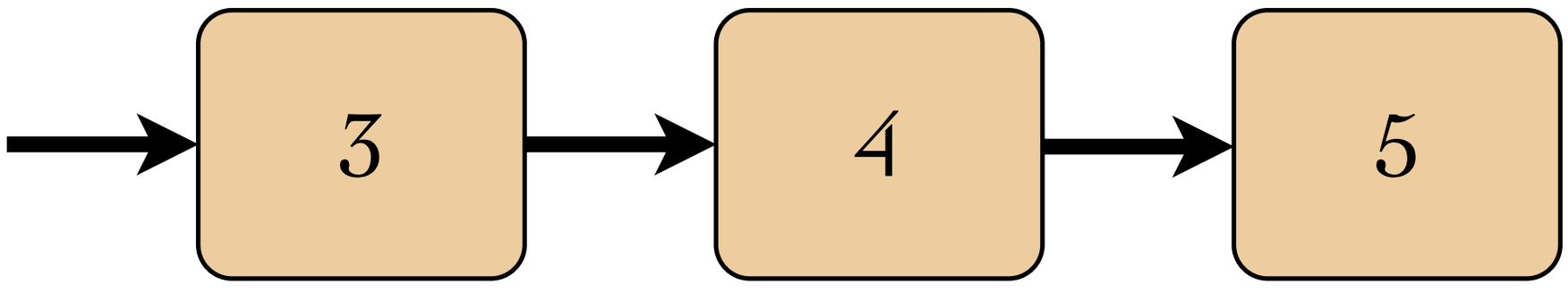
* Working copy
* Repository (local, remote)
* checkout, checkin — update, commit
* pull, push
* branch, merge

Working directory

one version of the project, visible to be edited

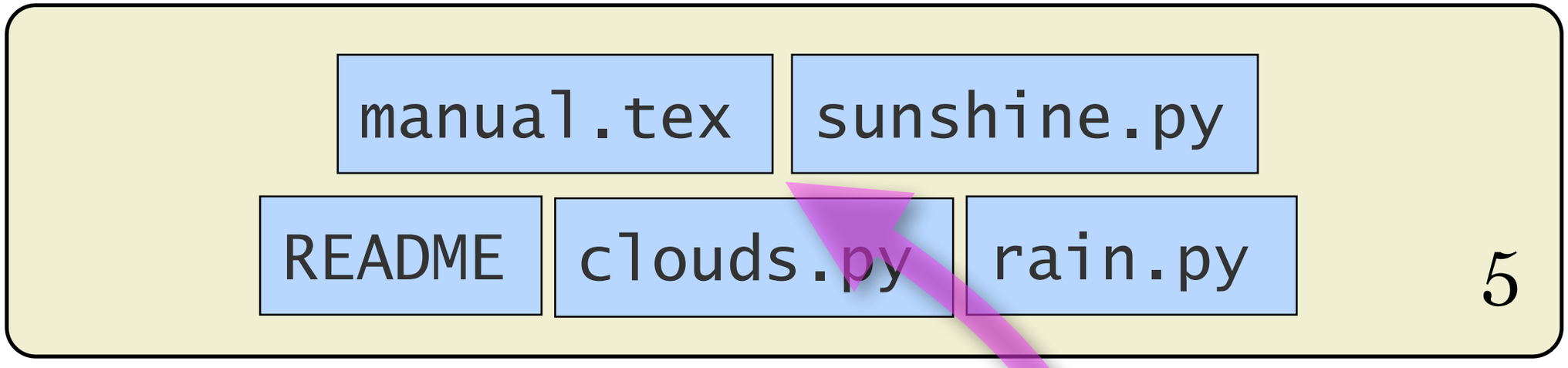


is based on **one** version in the repository,
usually the last one

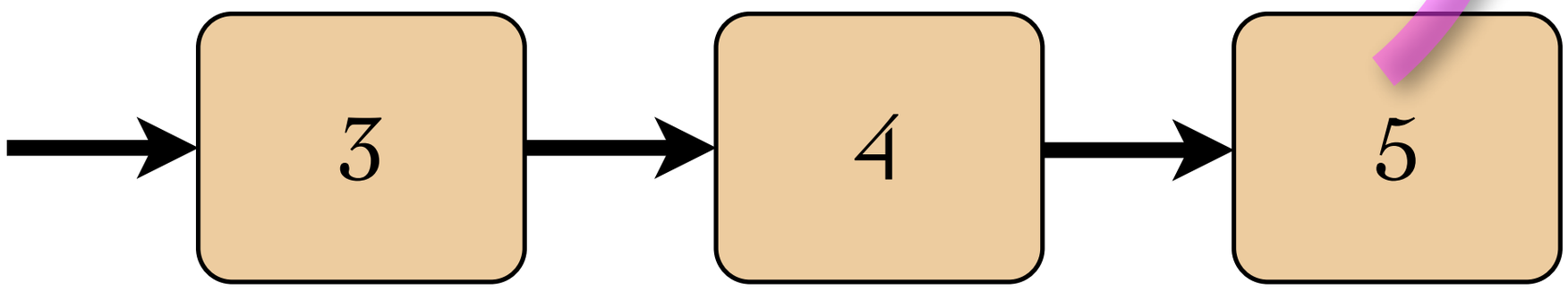


Working directory

one version of the project, visible to be edited

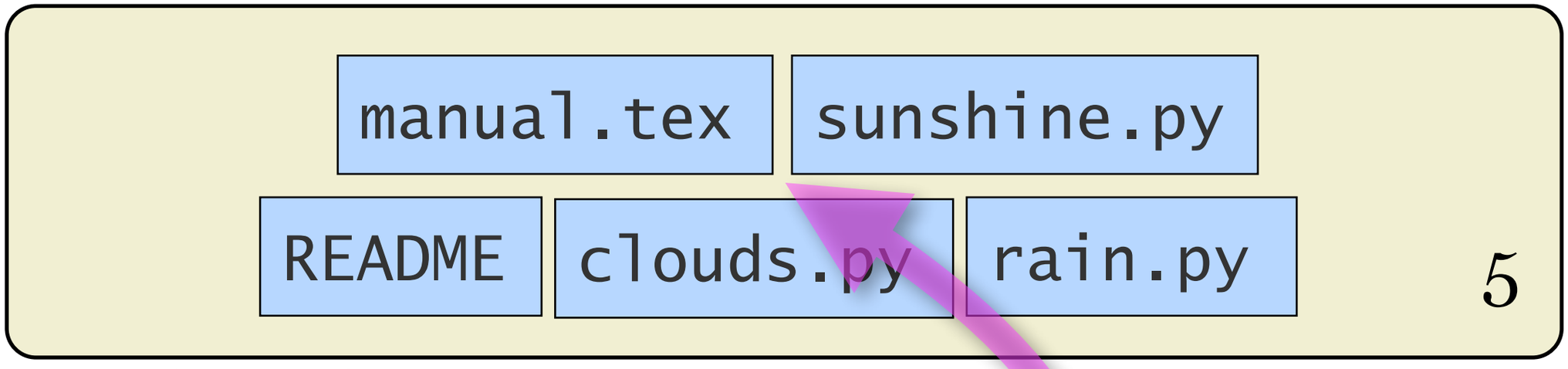


is based on **one** version in the repository,
usually the last one

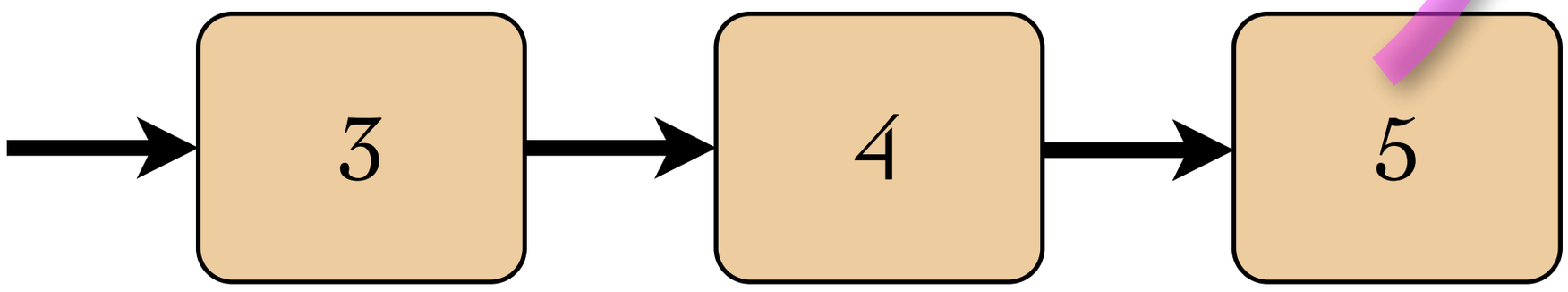


Working directory

one version of the project, visible to be edited



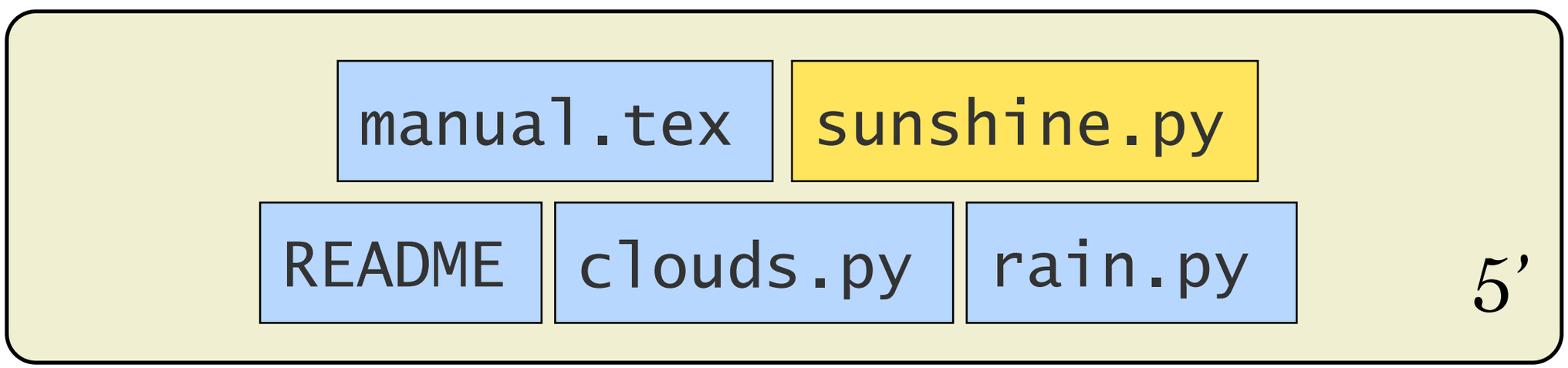
is based on **one** version in the repository,
usually the last one



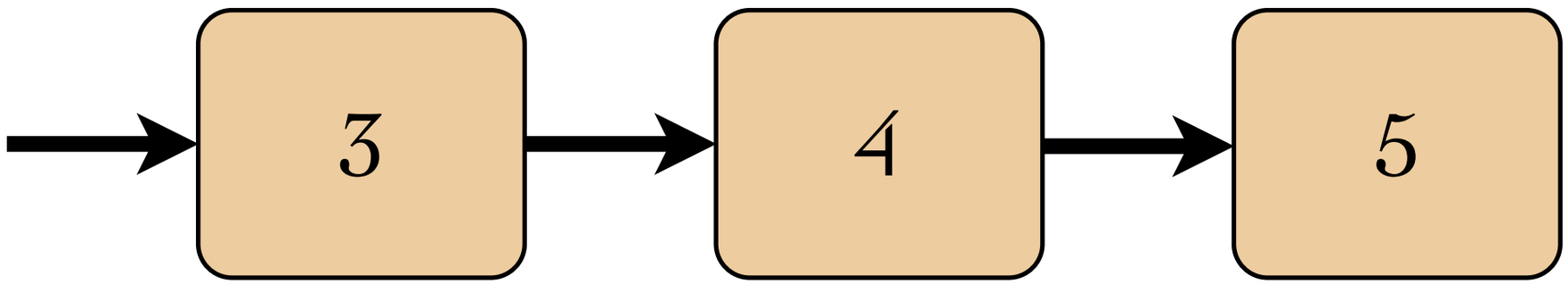
each WD has its own copy of the full repository!

Changes

working copy can be changed as usual

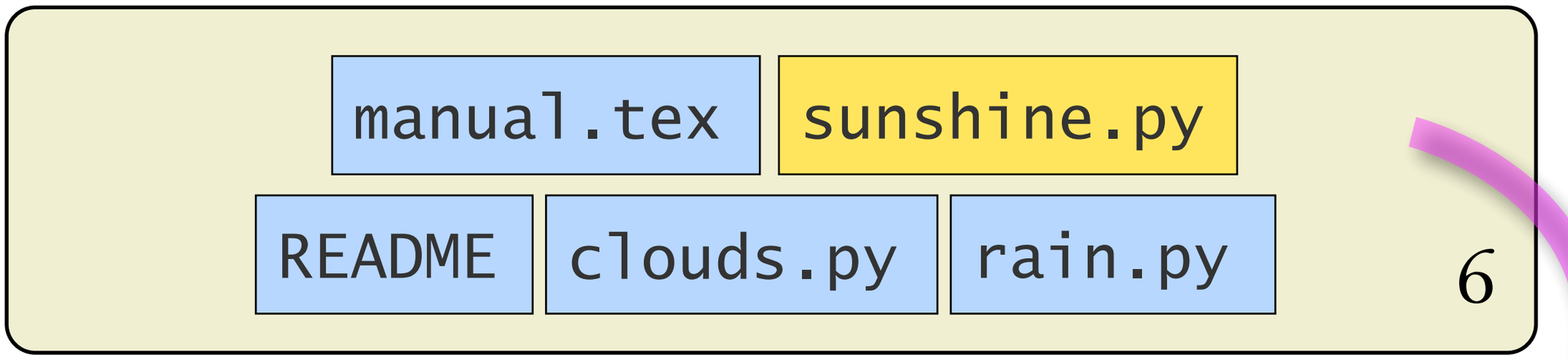


changes are then checked in / committed as a new version



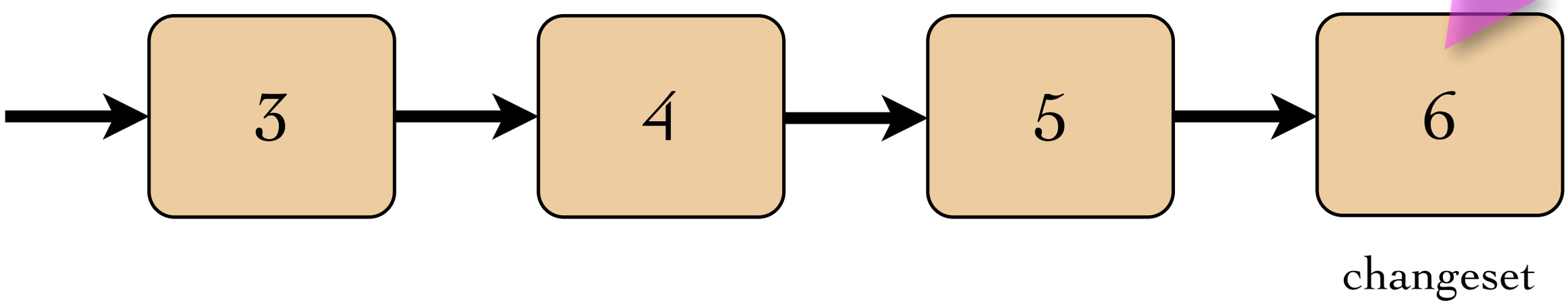
Changes

working copy can be changed as usual



6

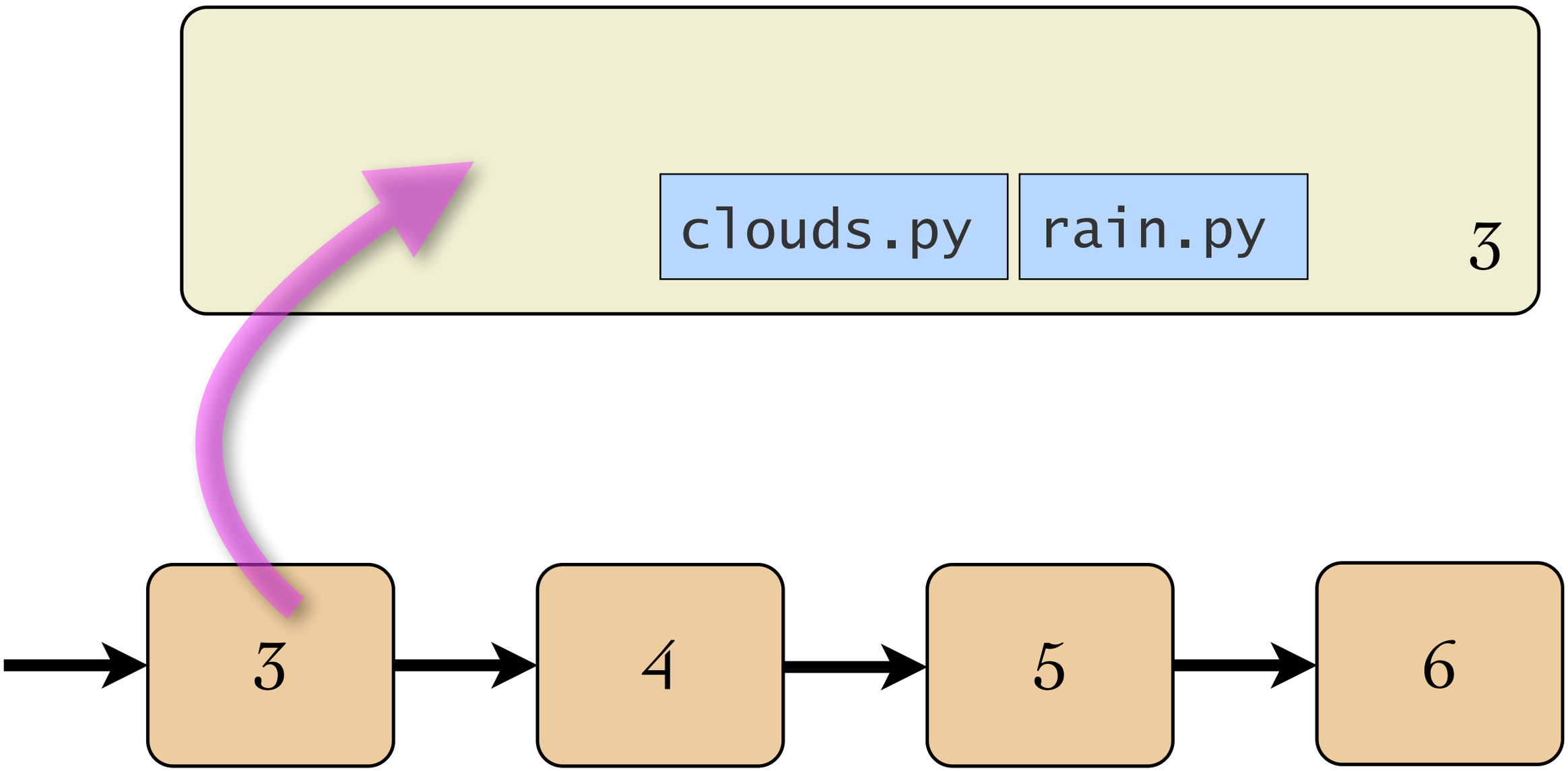
changes are then checked in / committed as a new version



changeset

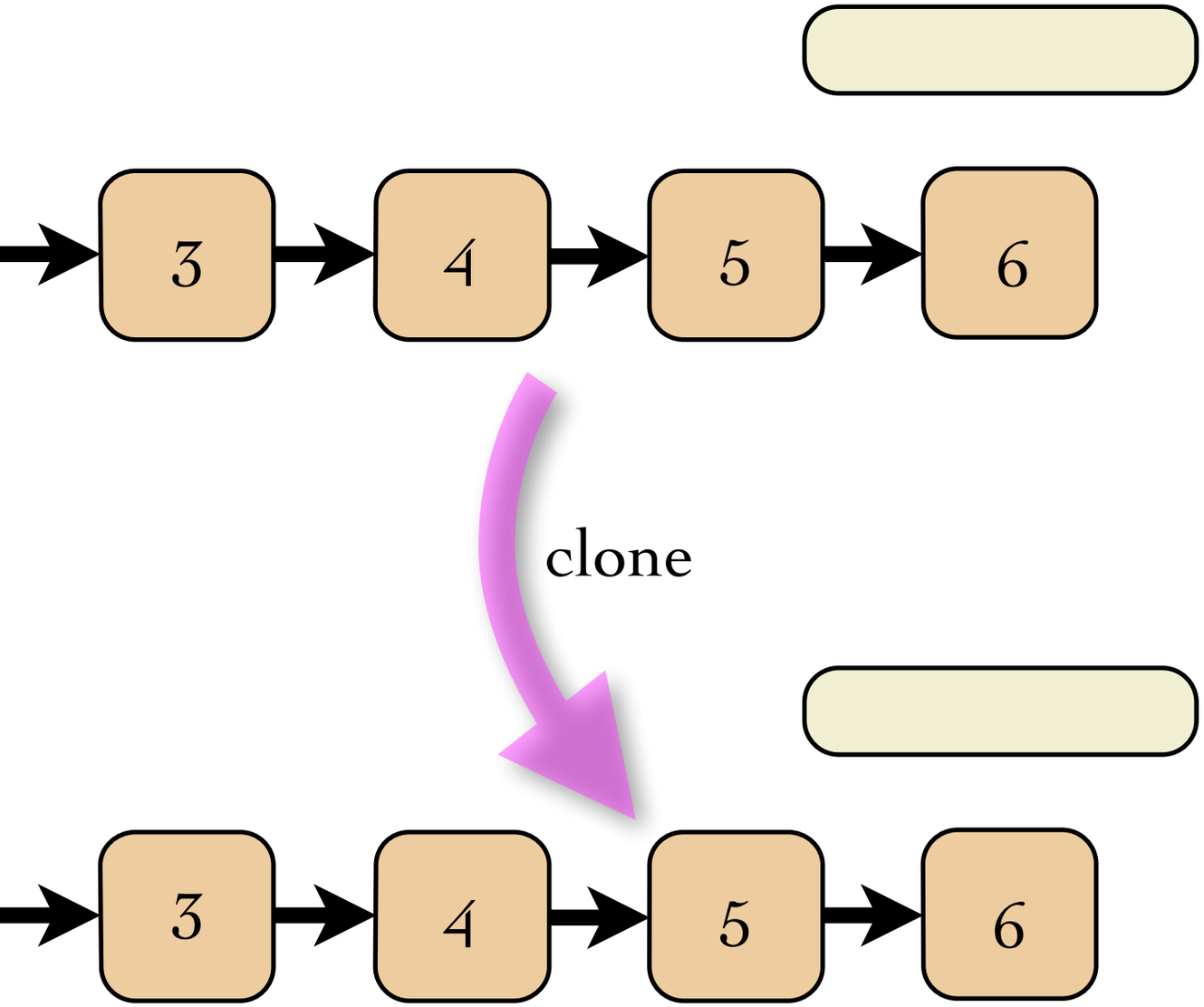
Backtracking

older versions can be looked at at any time



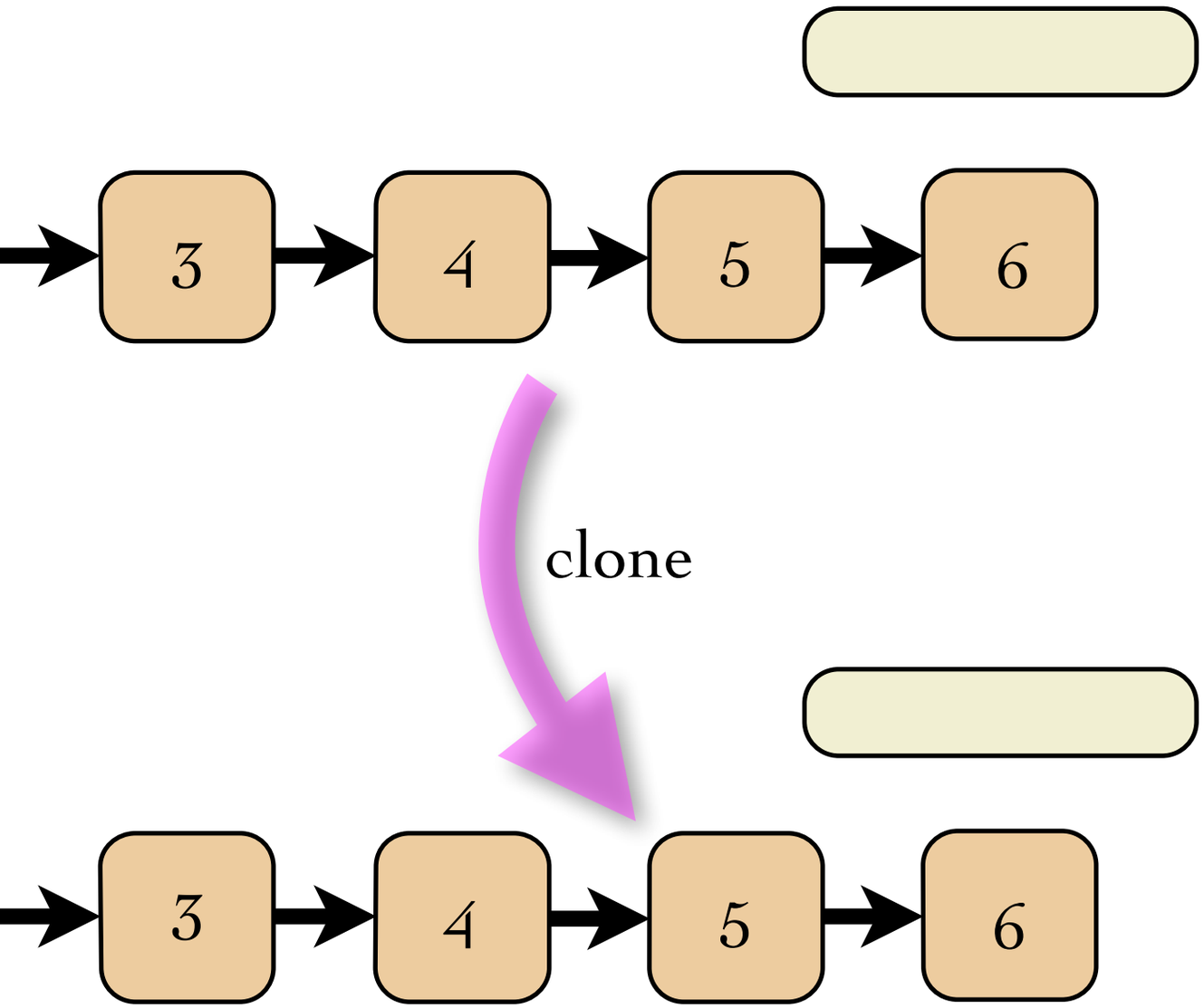
Remote work

clone repositories to other locations / devices / people



Remote work

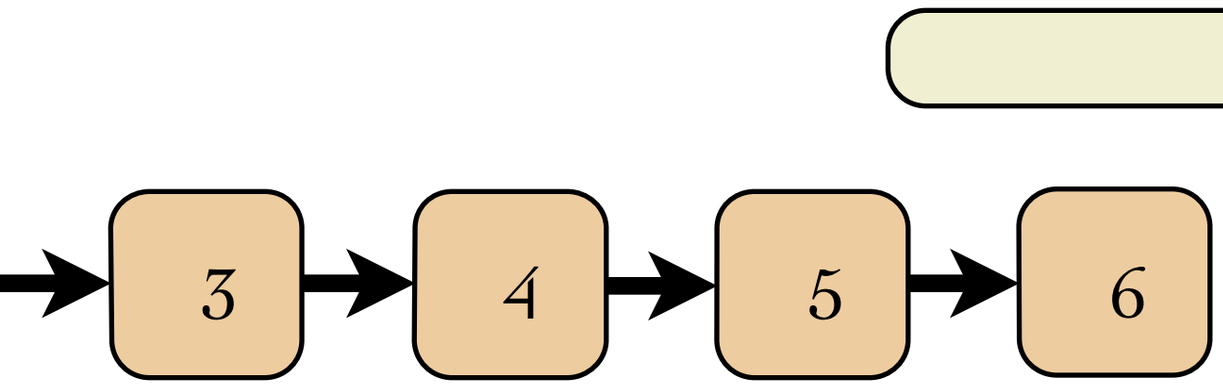
clone repositories to other locations / devices / people

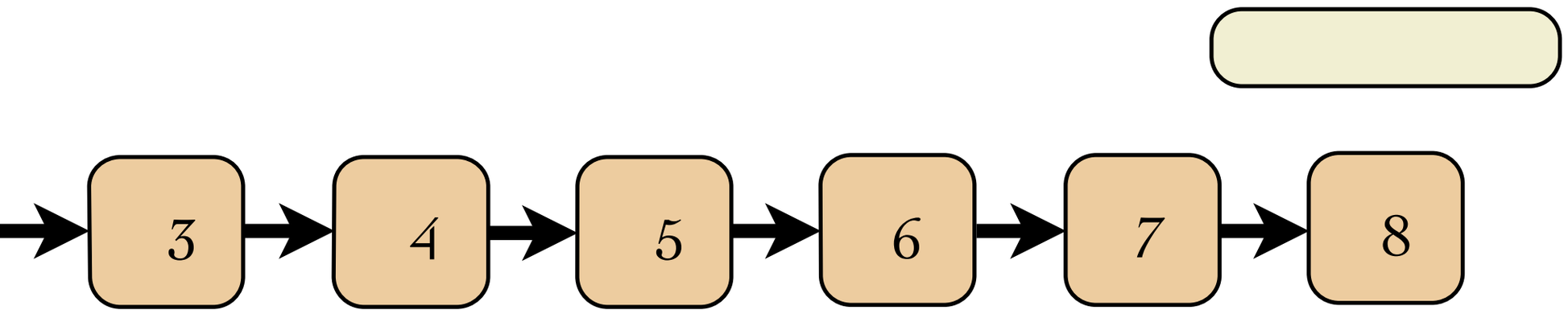


each clone has all features,
not distinguishable

Remote work

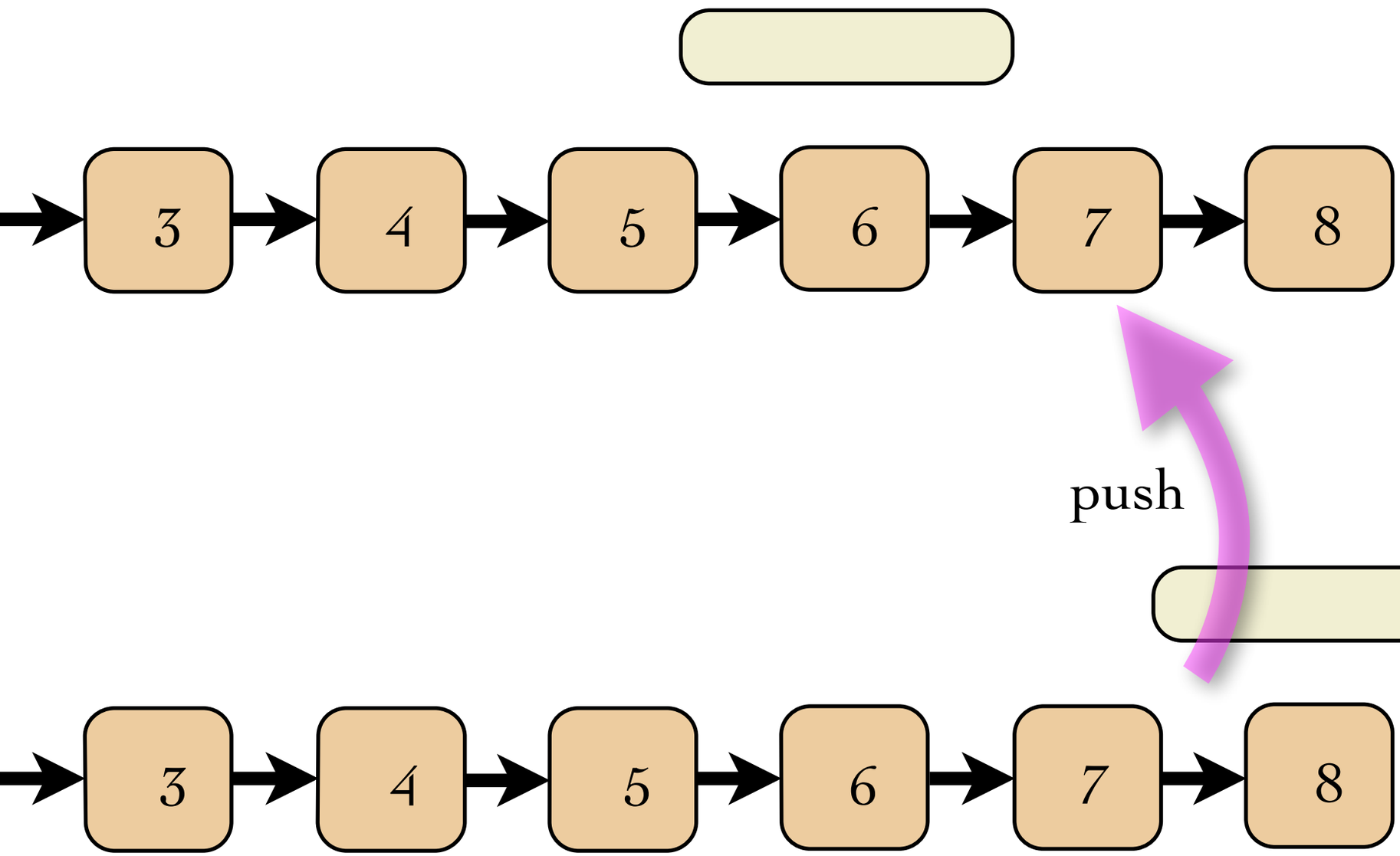
development can happen anywhere





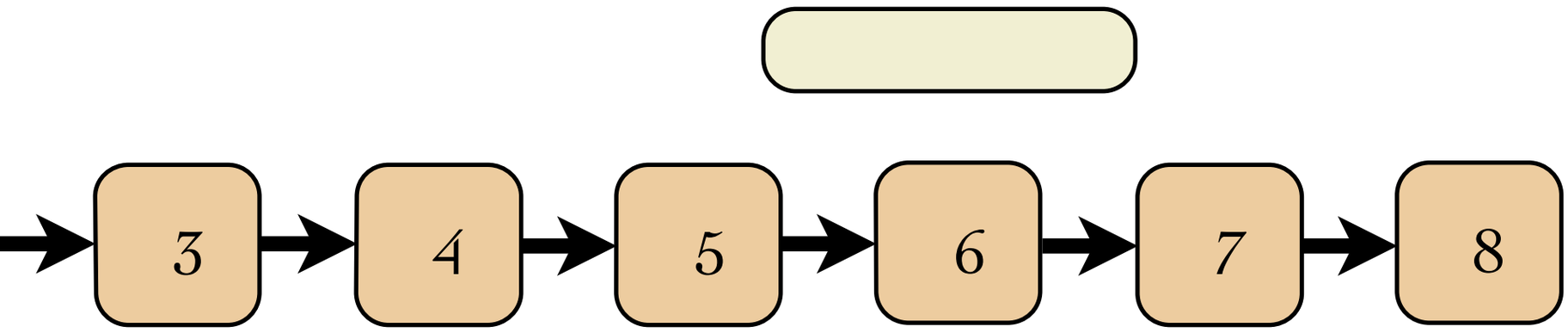
Remote work

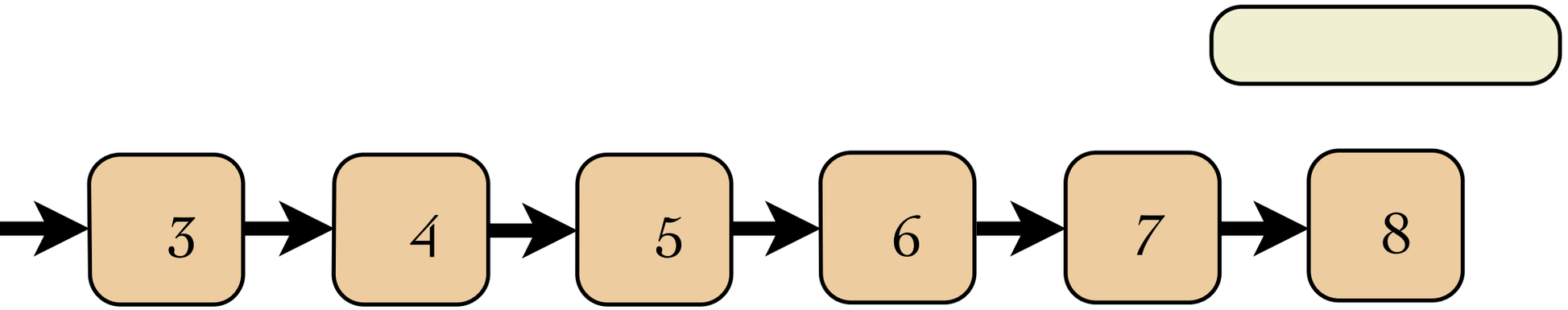
development can happen anywhere



Remote work

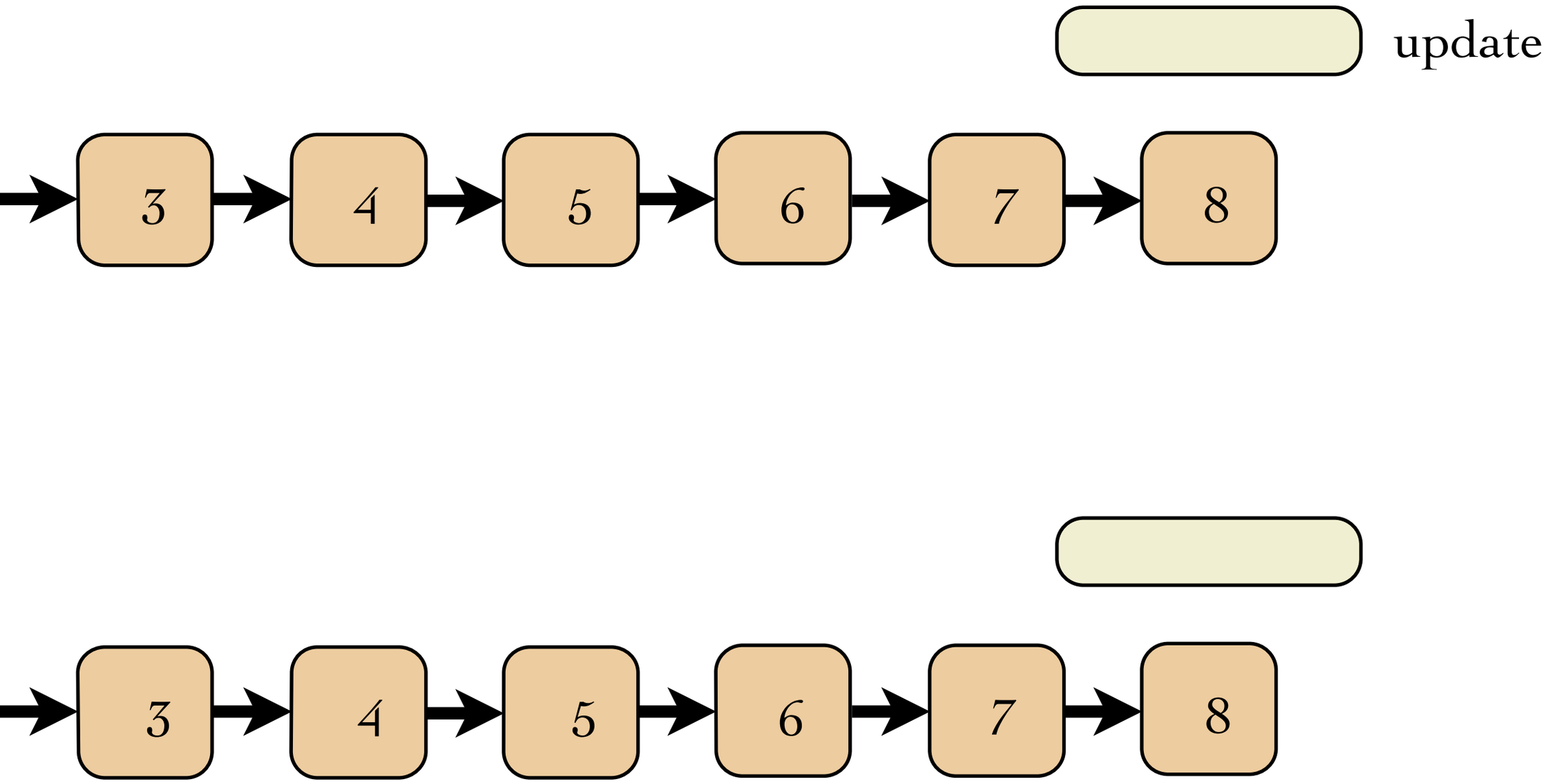
development can happen anywhere





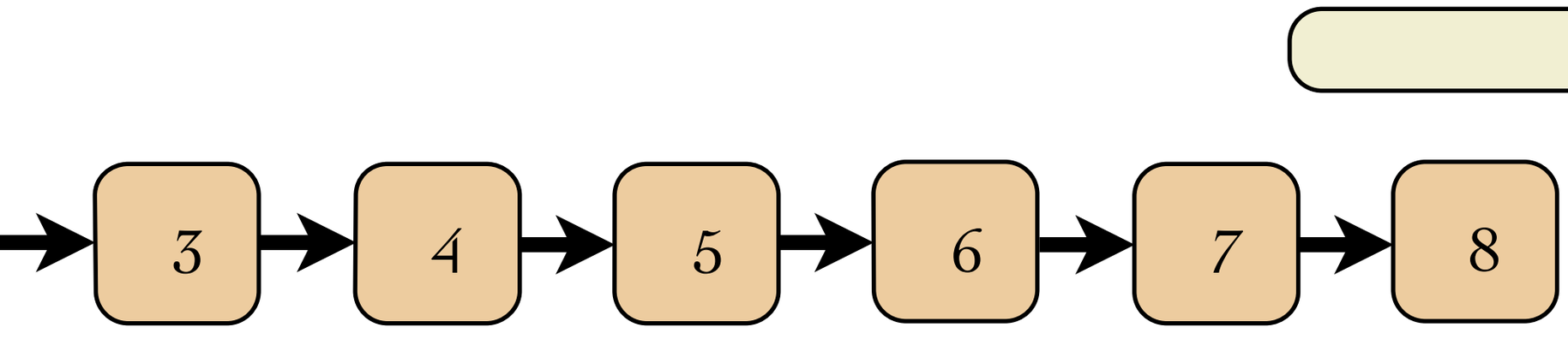
Remote work

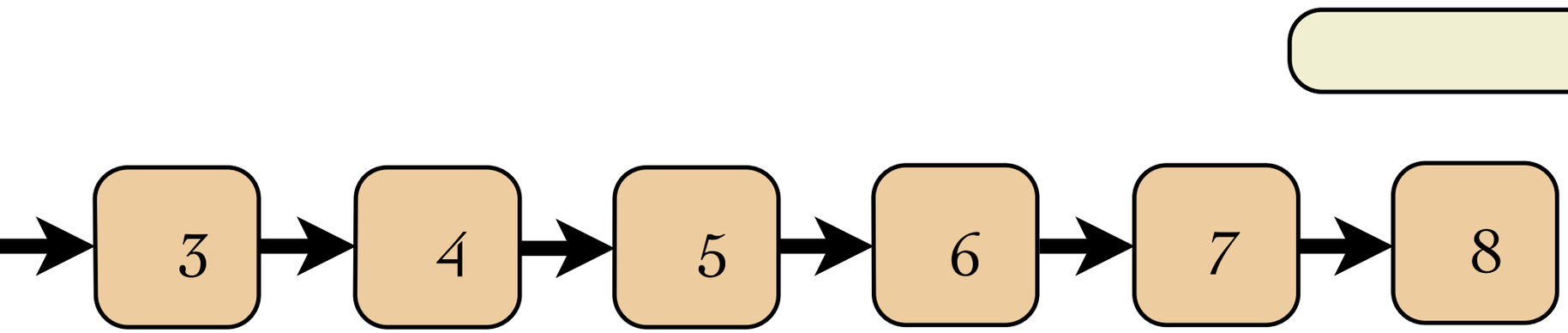
development can happen anywhere



Remote work

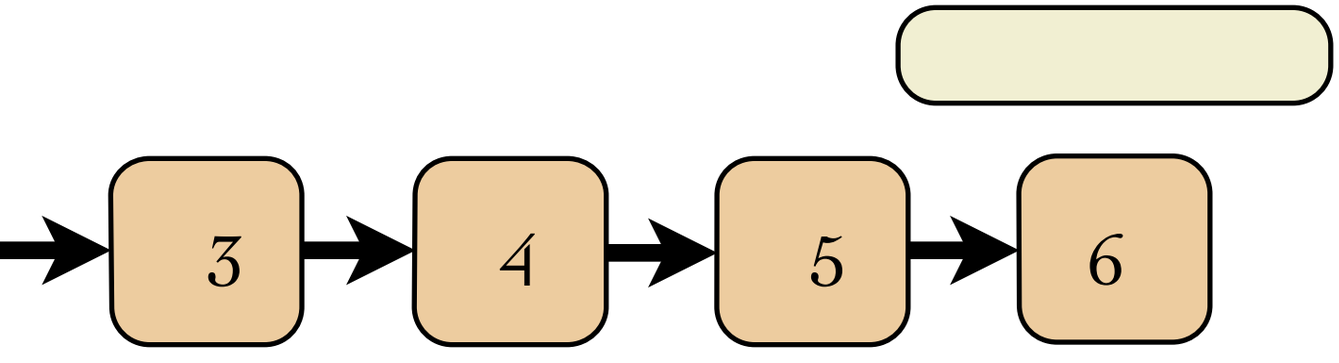
development can happen anywhere

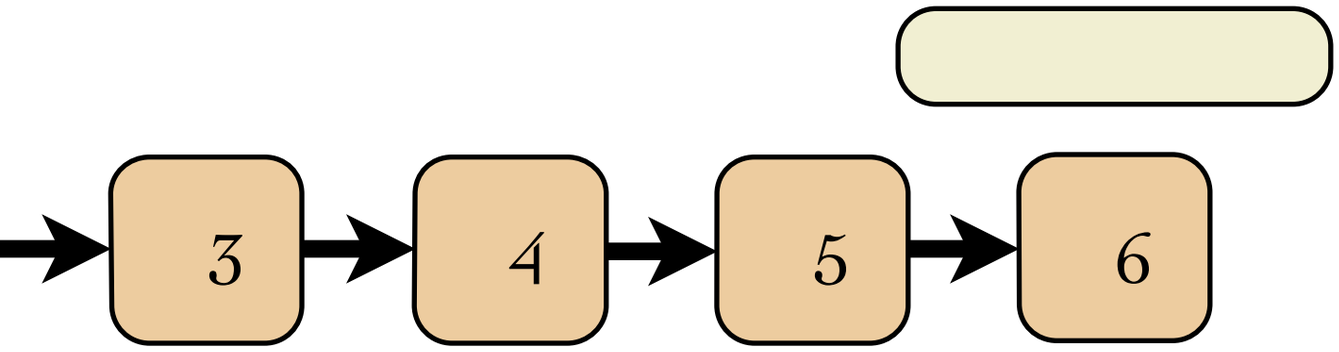




Branches

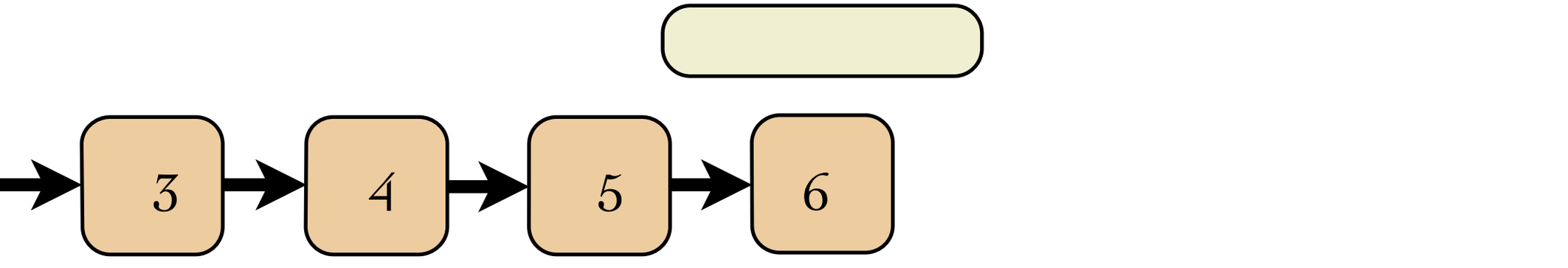
unlike in SVN, branches are a natural feature

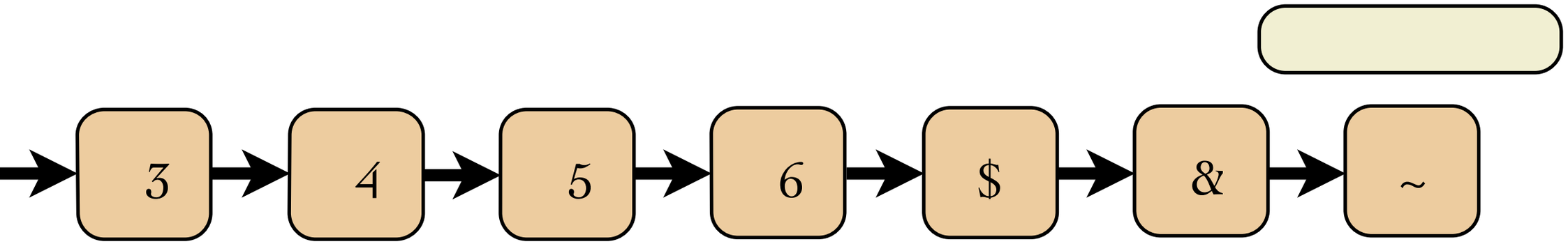




Branches

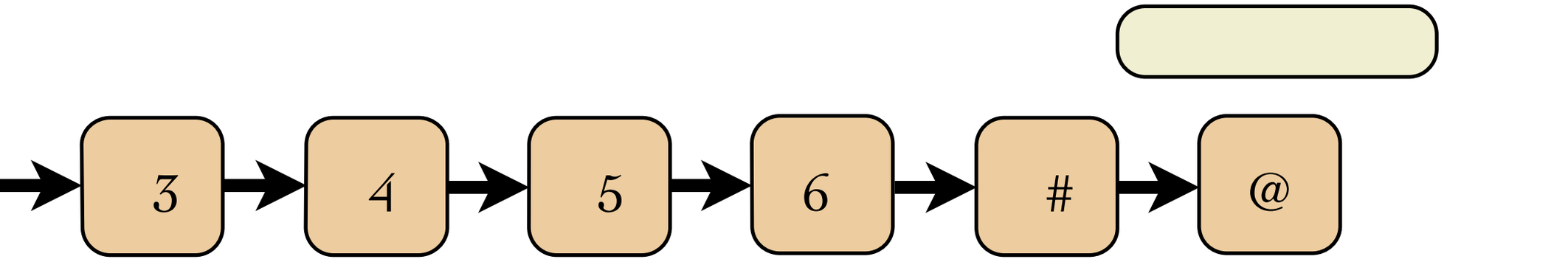
unlike in SVN, branches are a natural feature

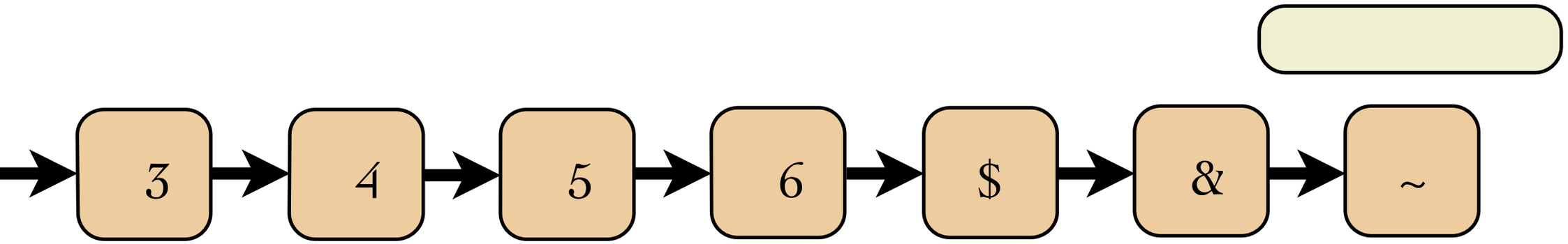




Branches

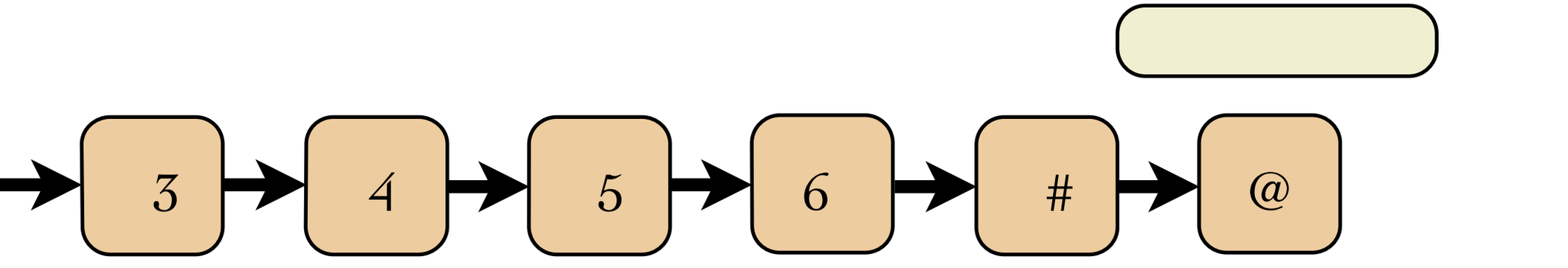
unlike in SVN, branches are a natural feature



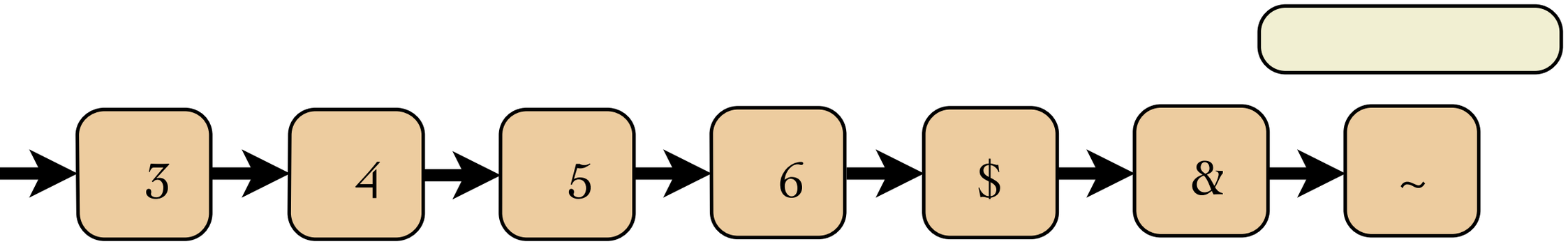


Branches

unlike in SVN, branches are a natural feature

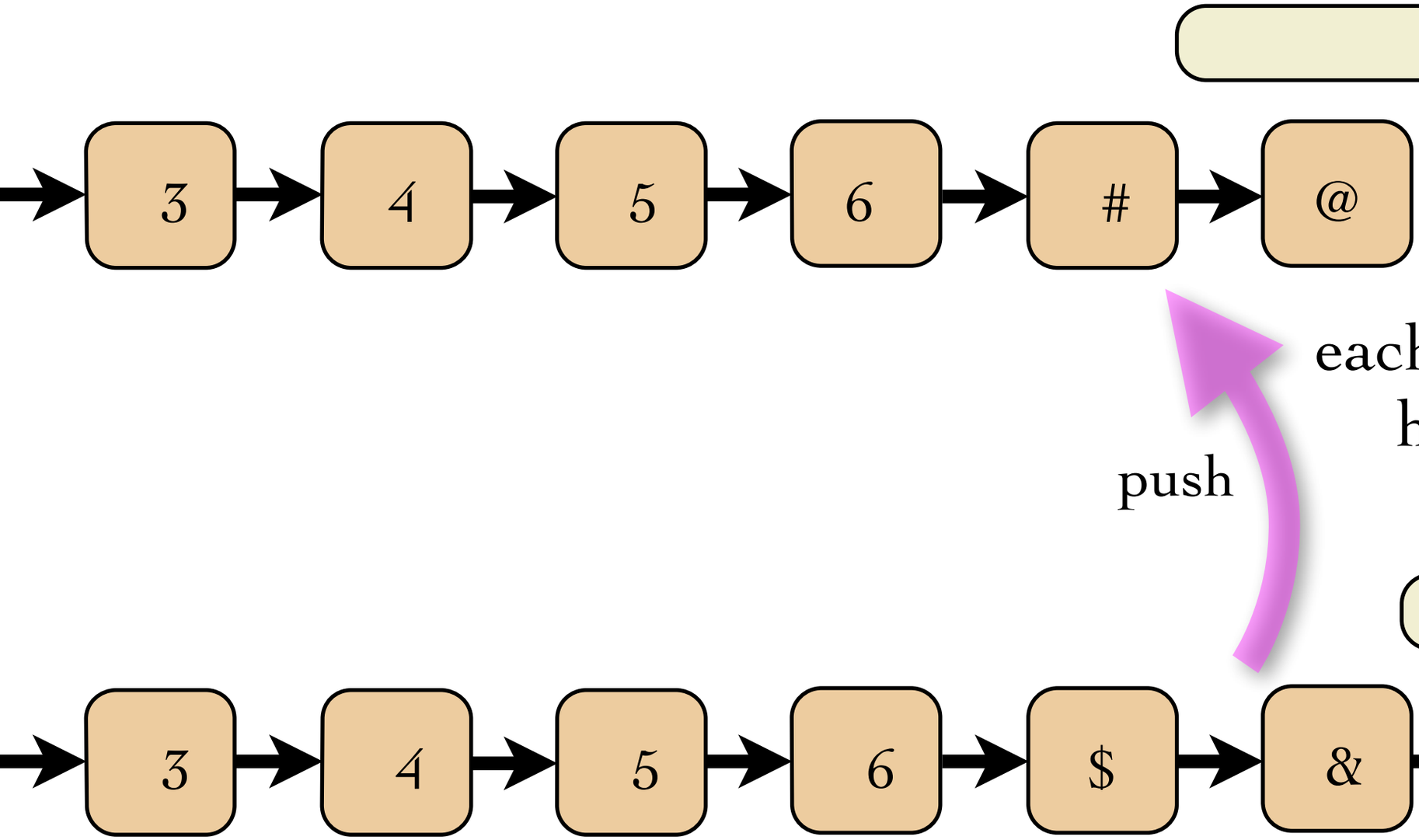


each distinct changeset
has a unique ID!



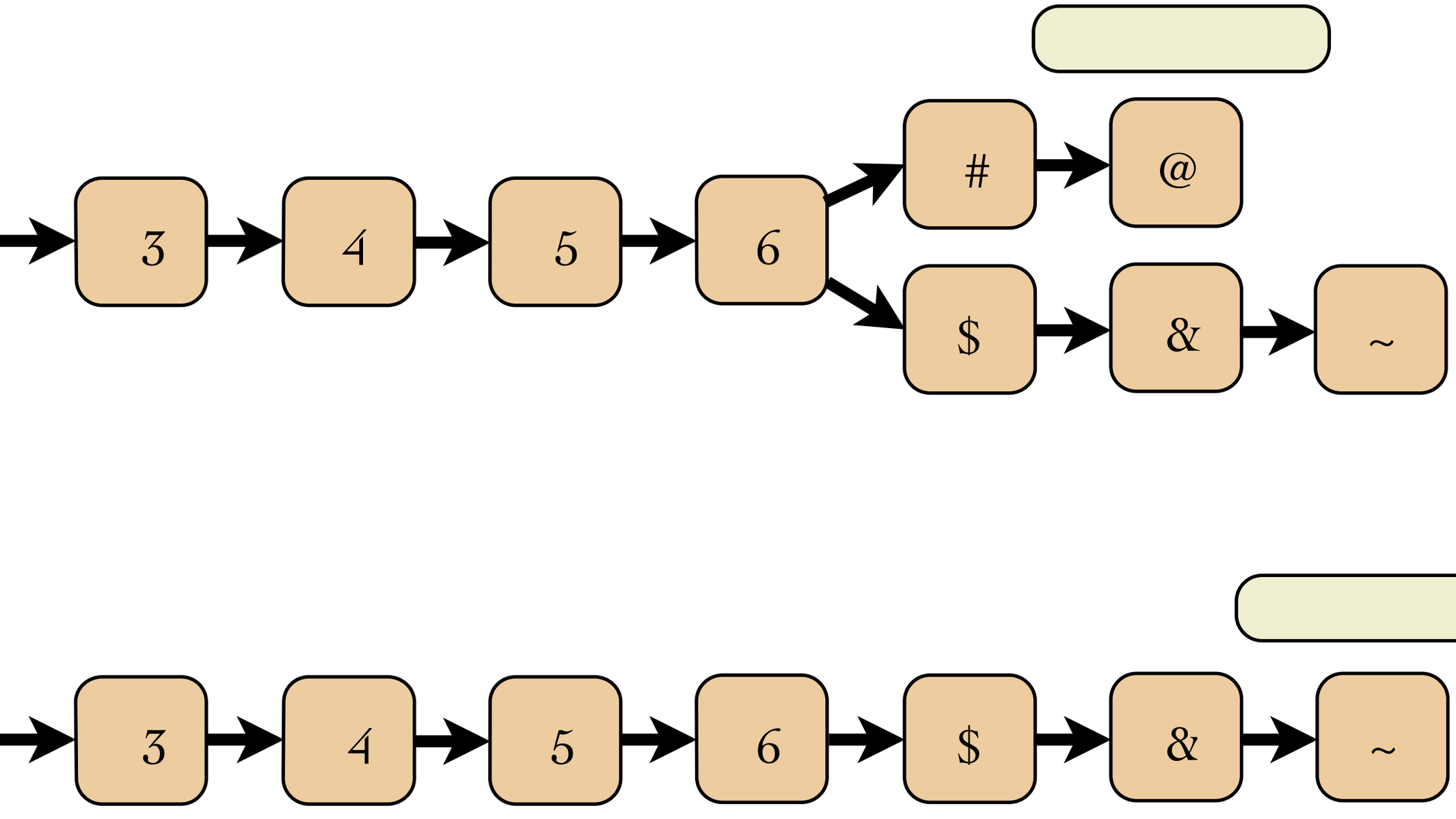
Branches

unlike in SVN, branches are a natural feature



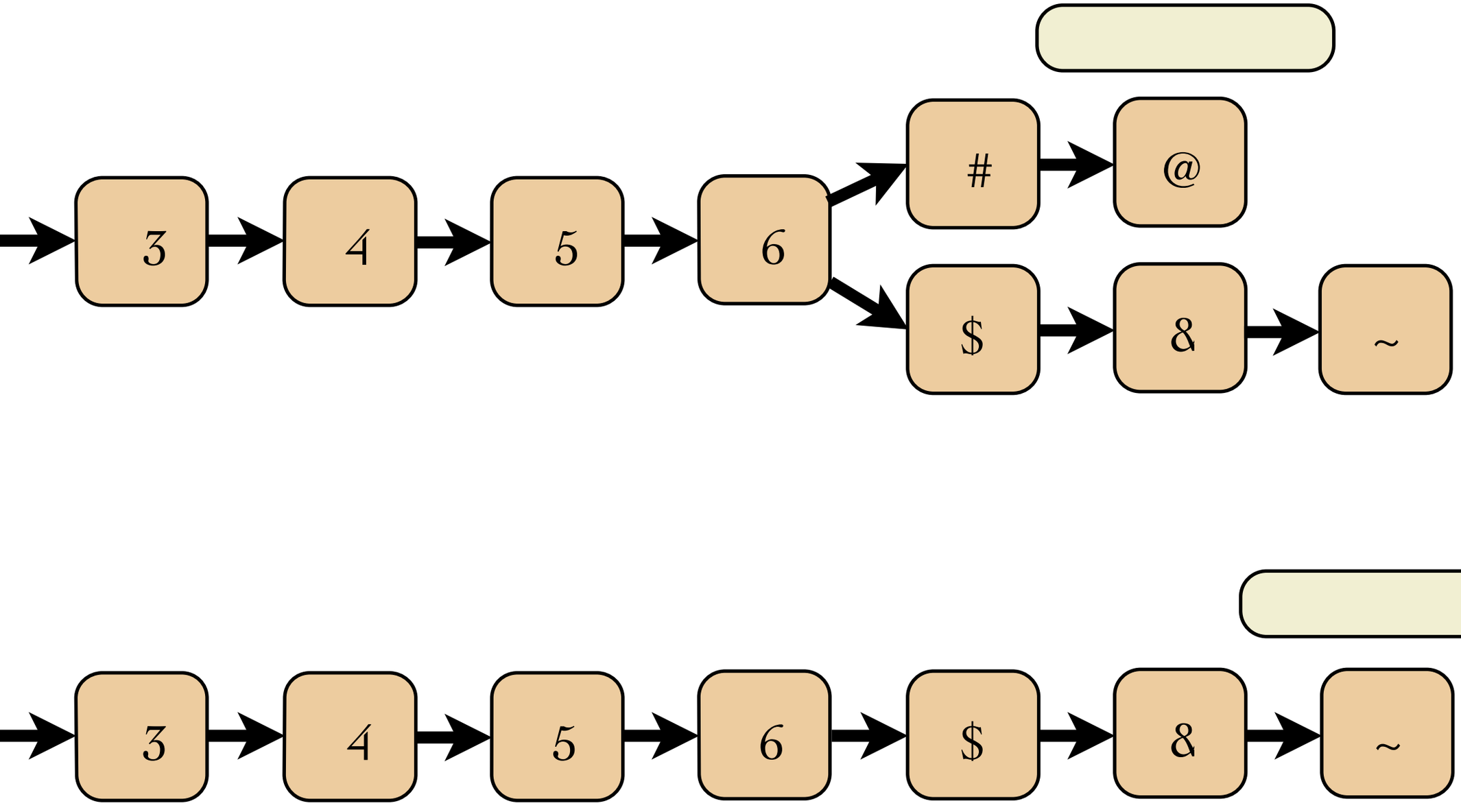
Branches

unlike in SVN, branches are a natural feature



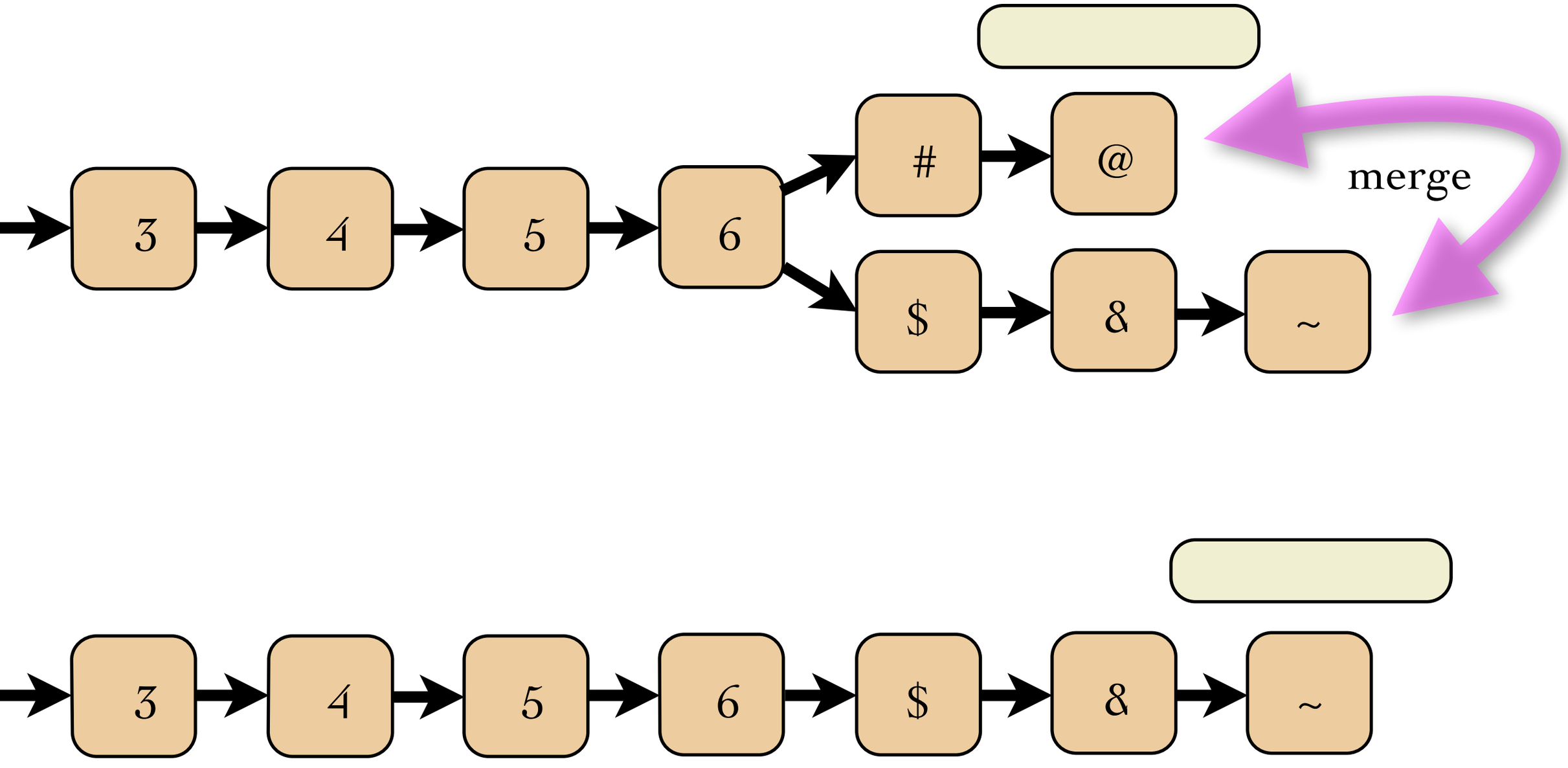
Merging

merging is automatic as far as possible



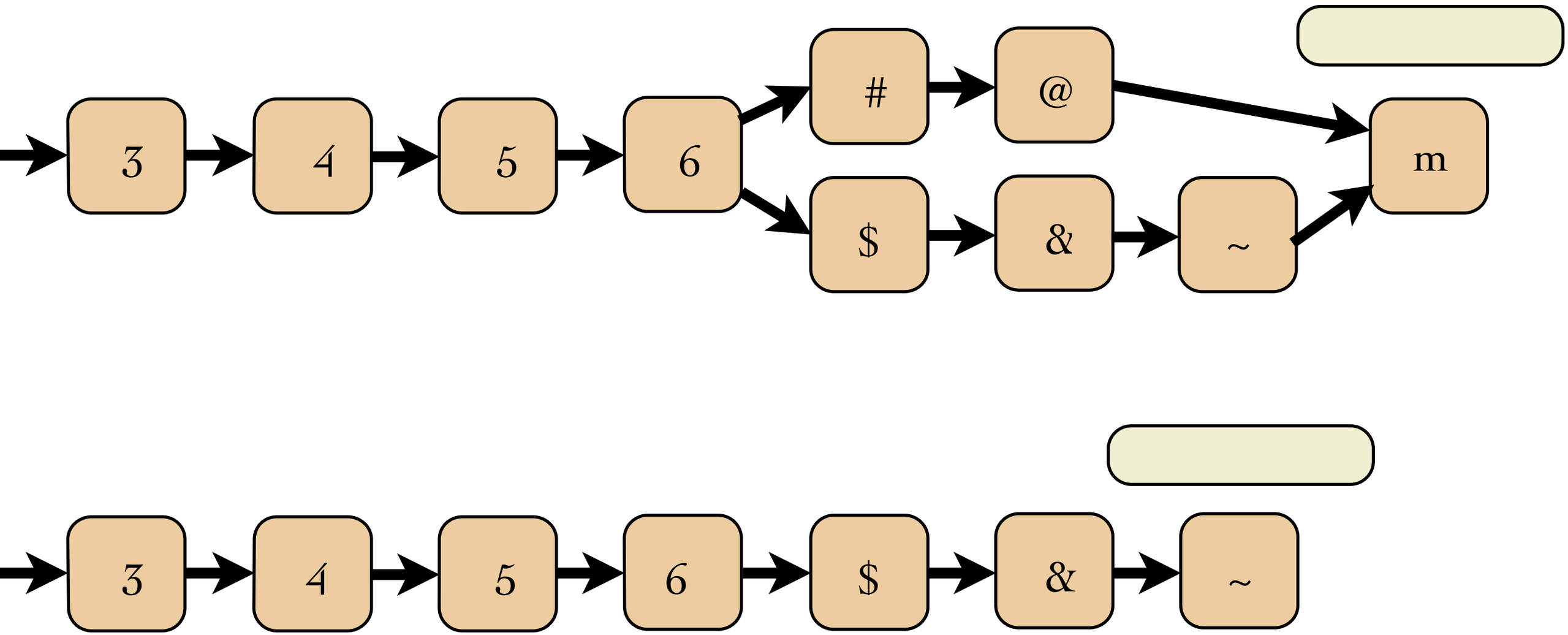
Merging

merging is automatic as far as possible



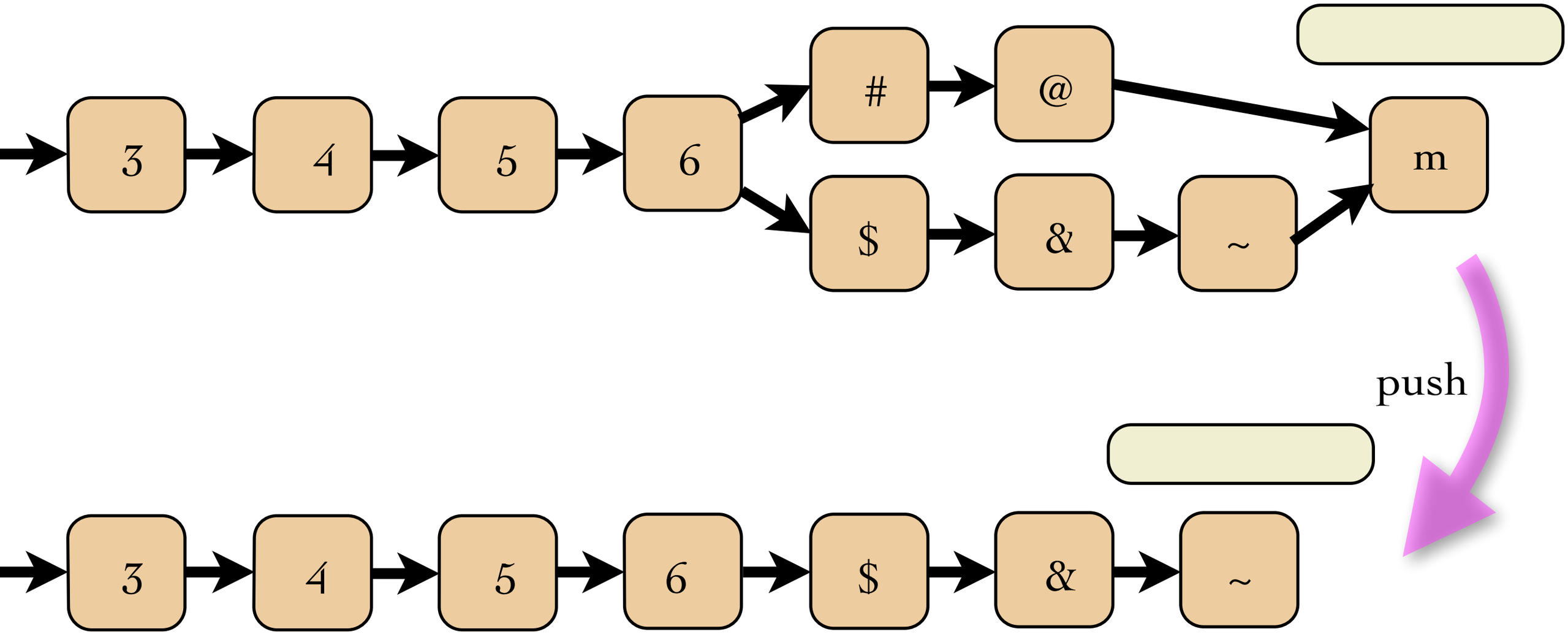
Merging

only need to fix some conflicts by hand



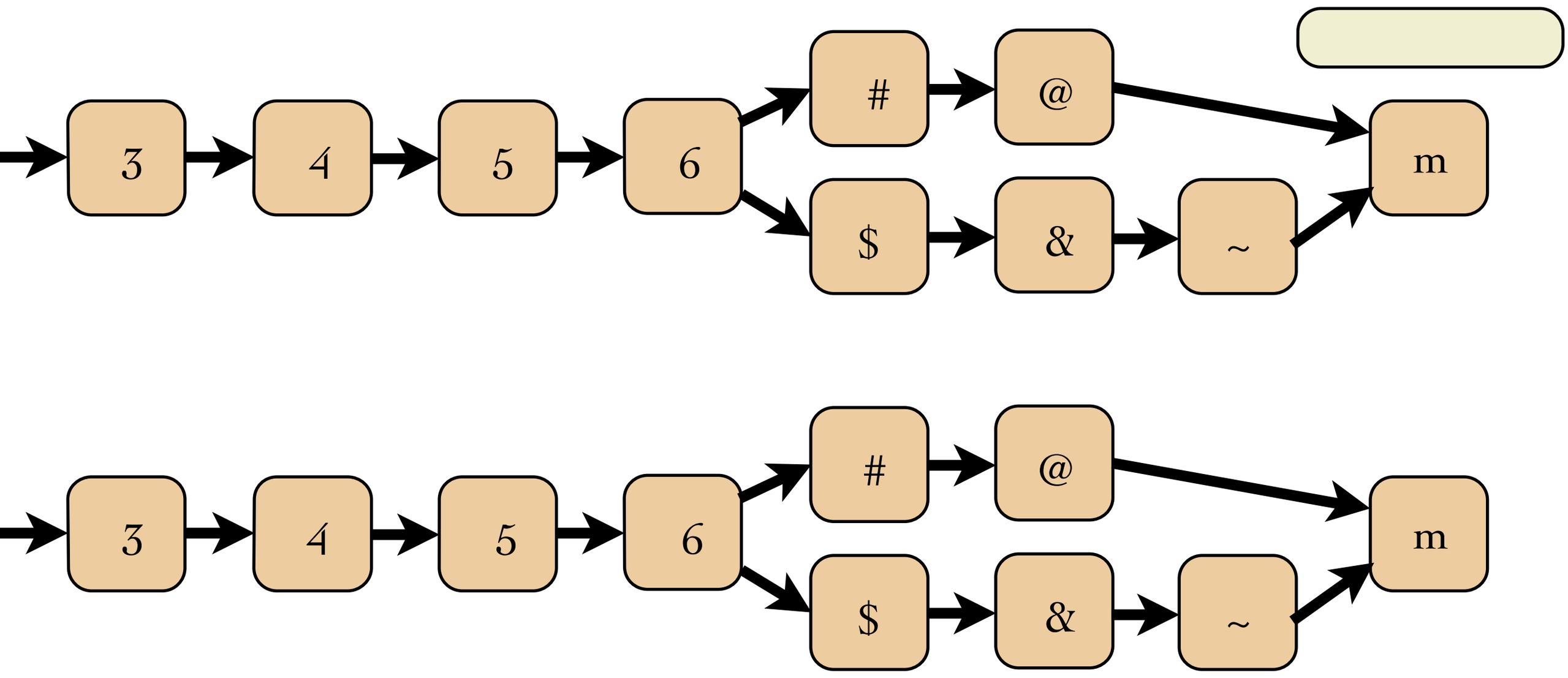
Merging

only need to fix some conflicts by hand



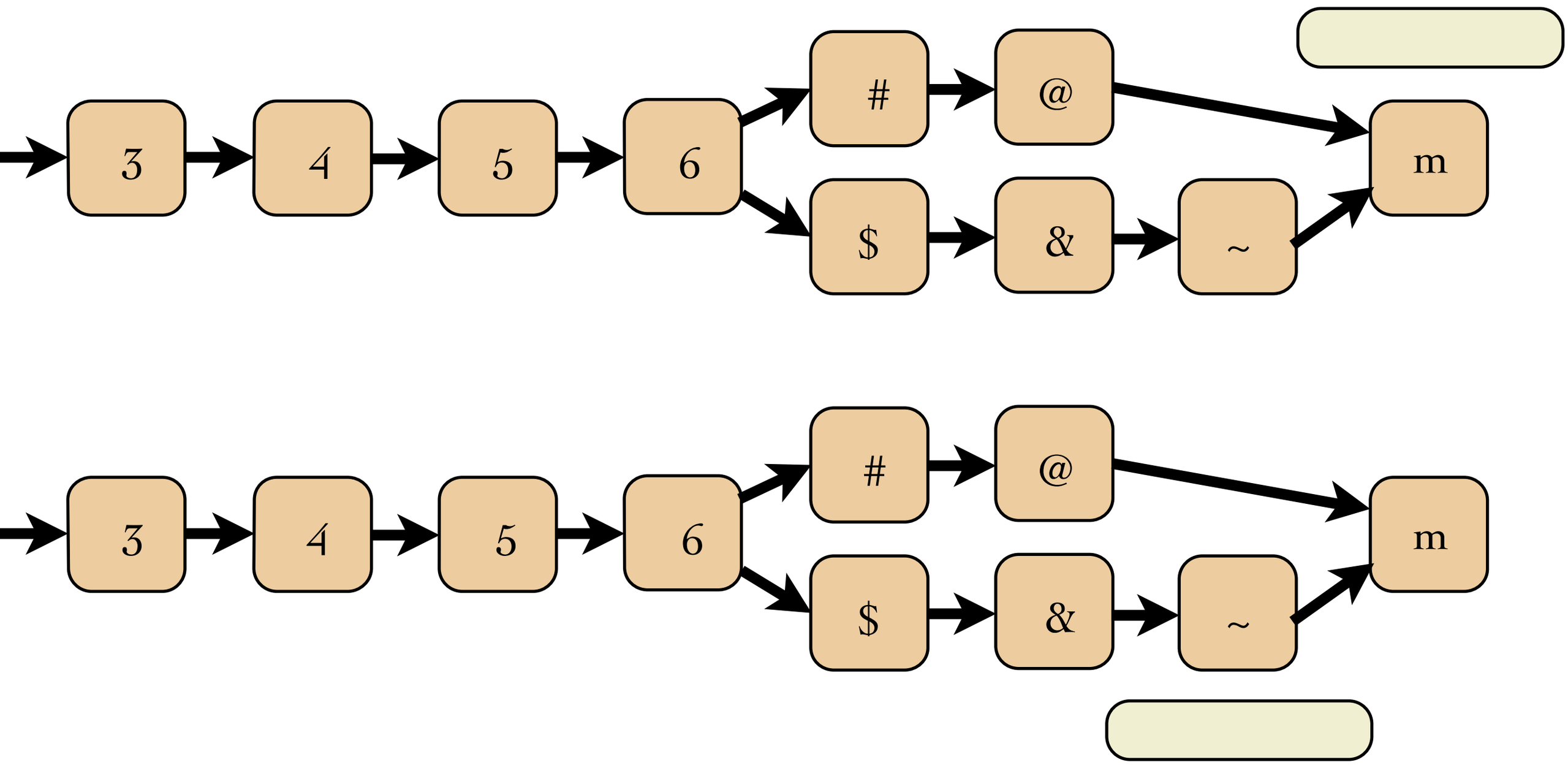
Merging

consistent state after push



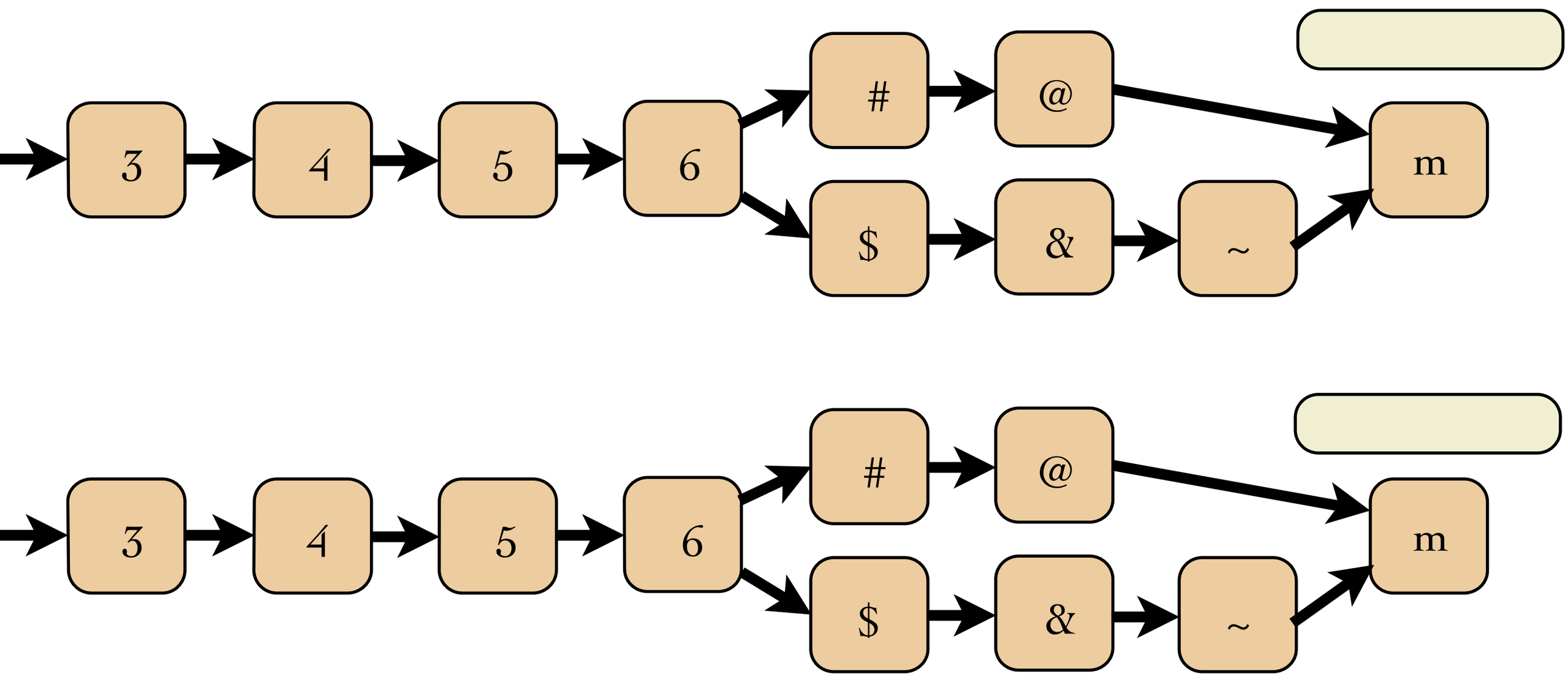
Merging

consistent state after push



Merging

consistent state after push

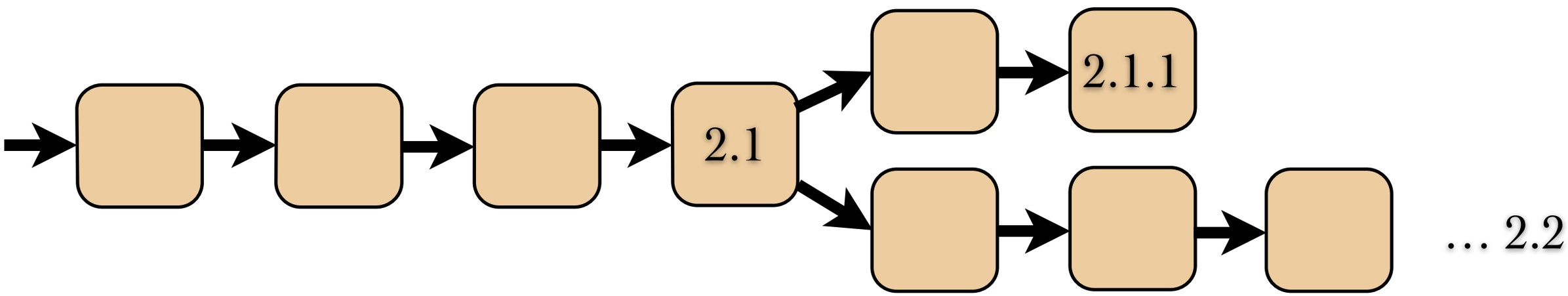


Typical usage patterns

* 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

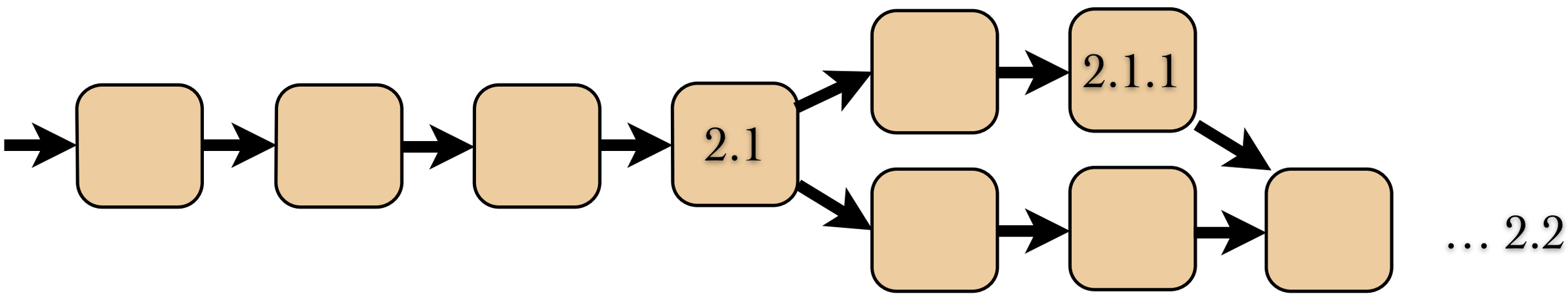
Typical usage patterns

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



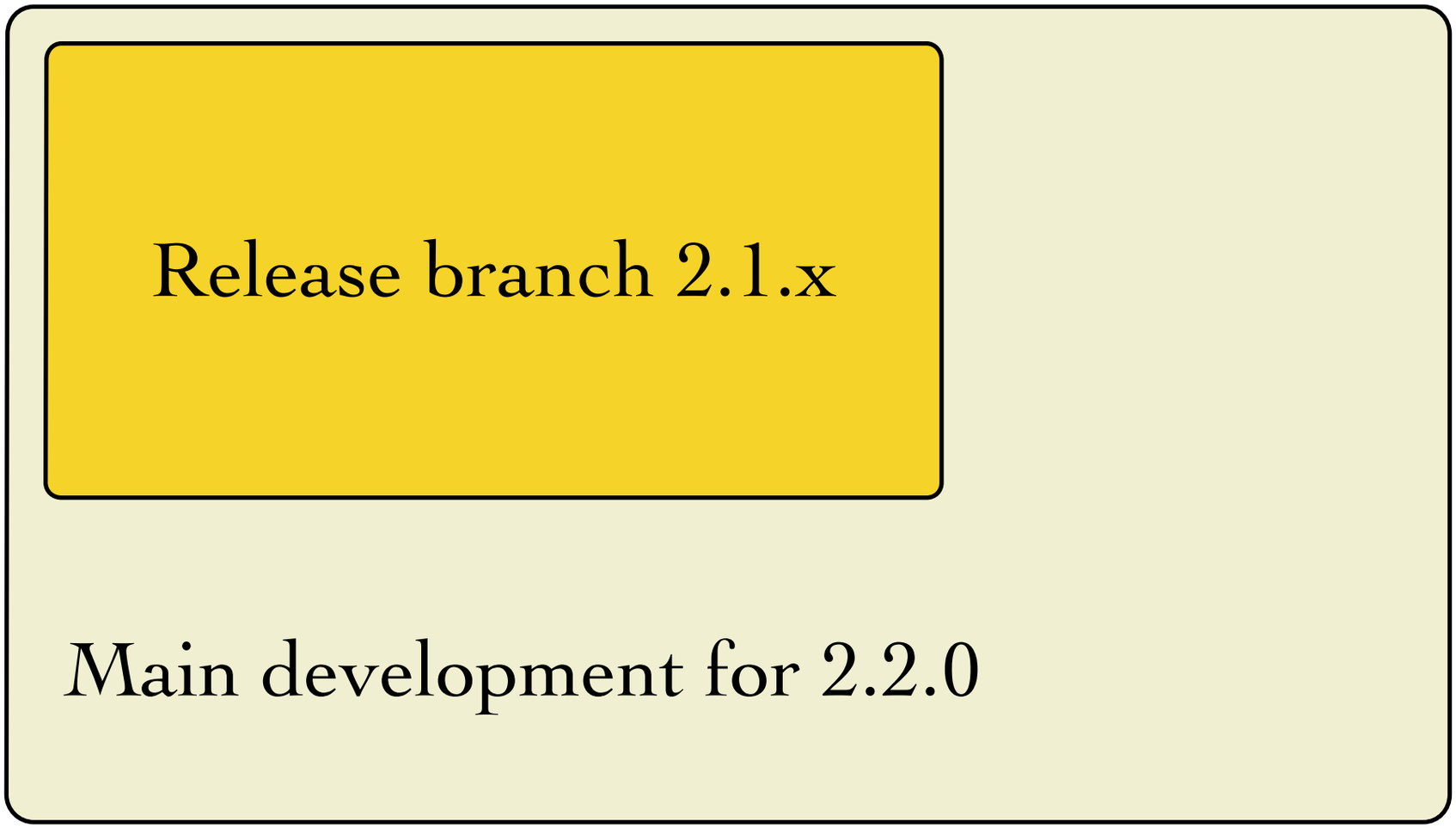
Typical usage patterns

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



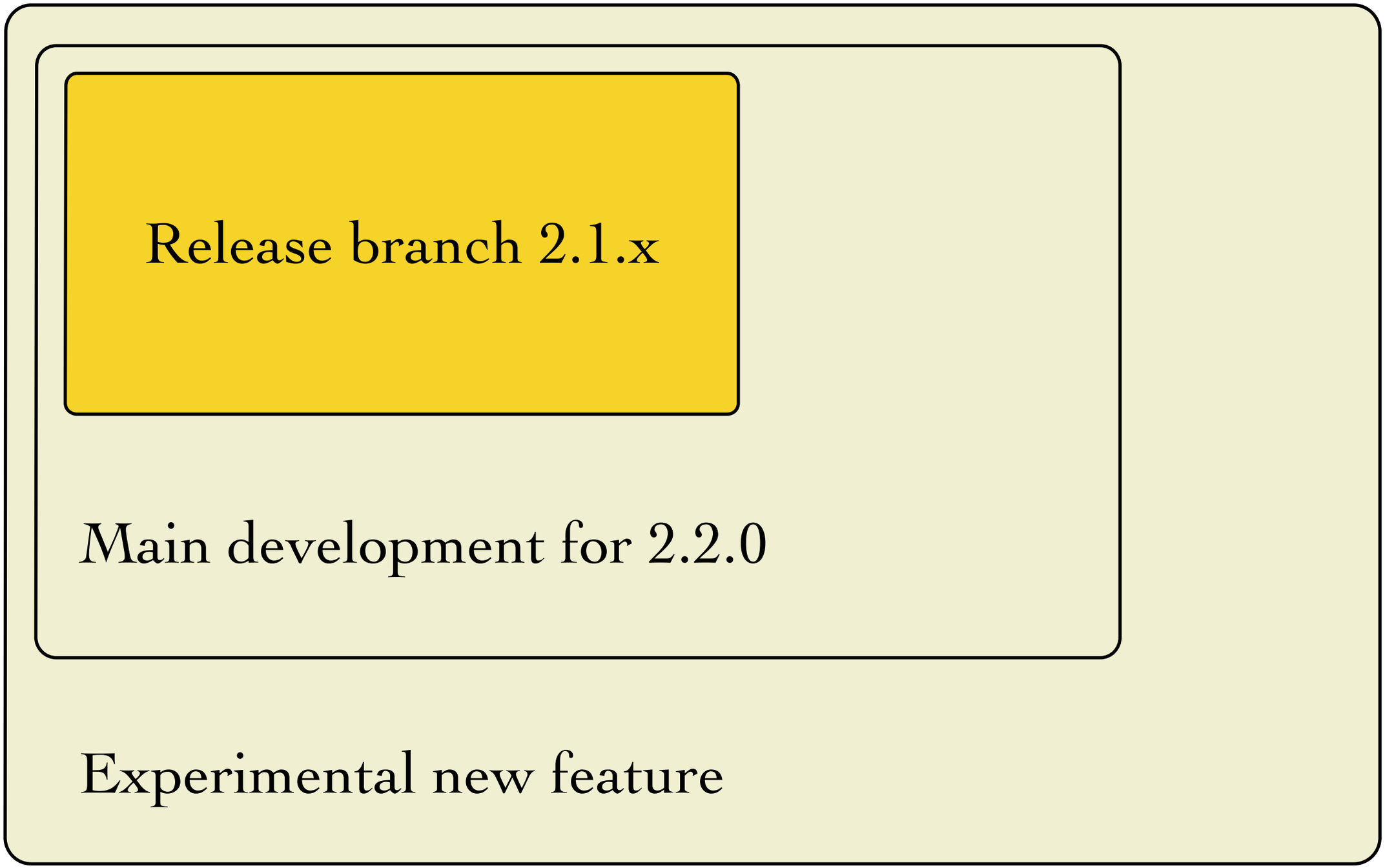
Which branch for which checkin?

* Pick the innermost appropriate branch



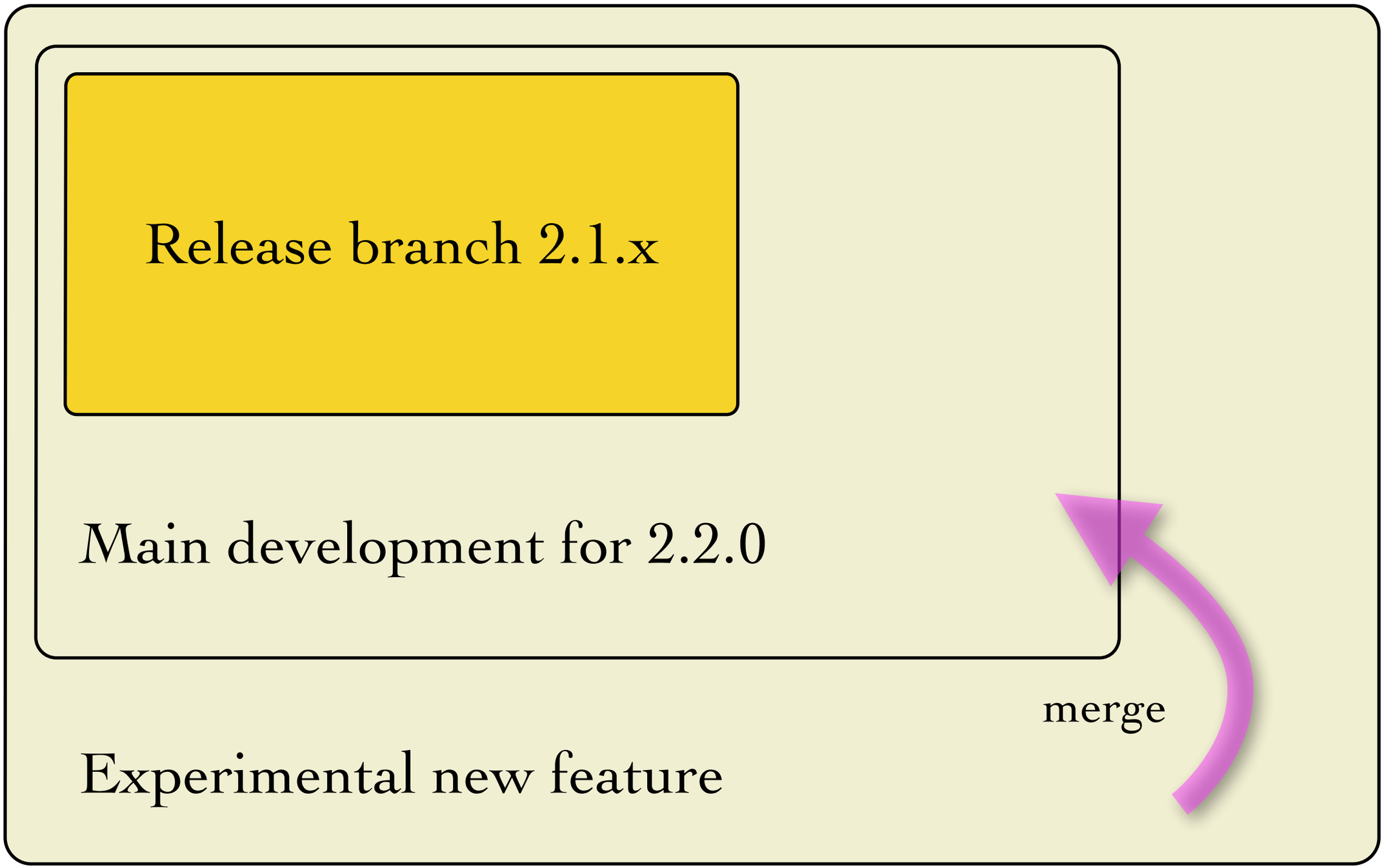
Which branch for which checkin?

* Pick the innermost appropriate branch



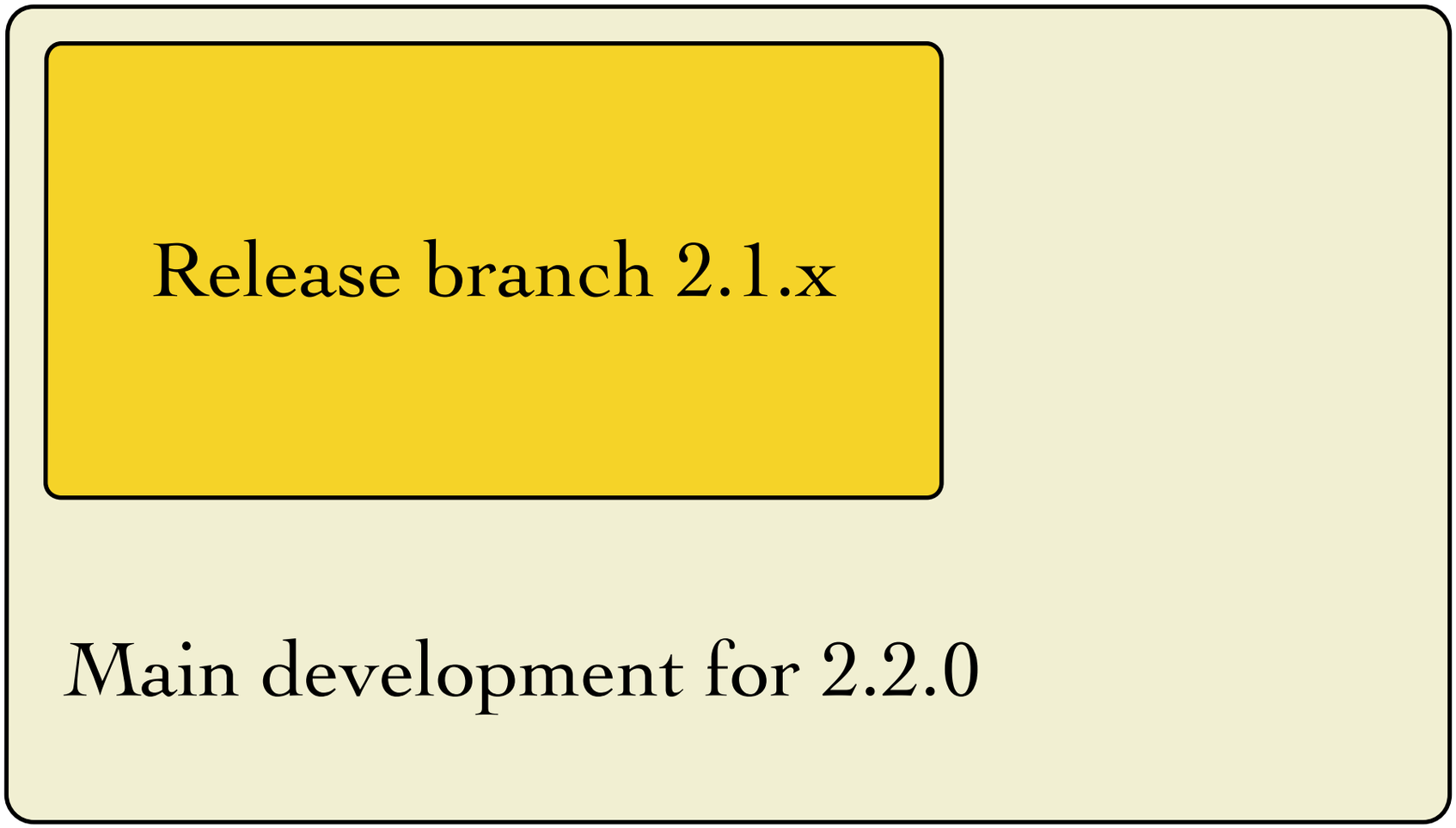
Which branch for which checkin?

* Pick the innermost appropriate branch



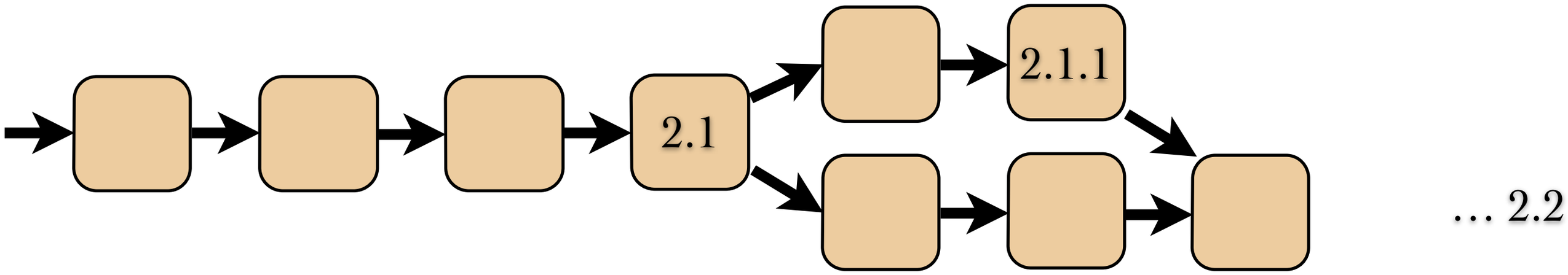
Which branch for which checkin?

* Pick the innermost appropriate branch



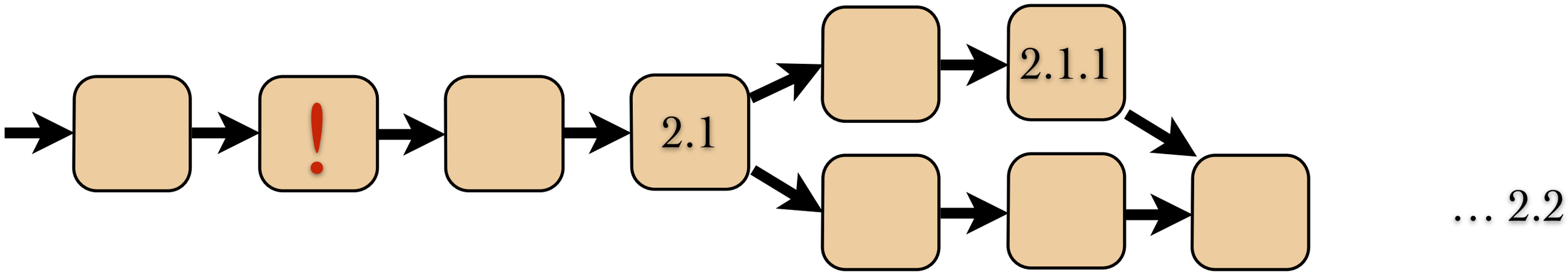
Typical usage patterns

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



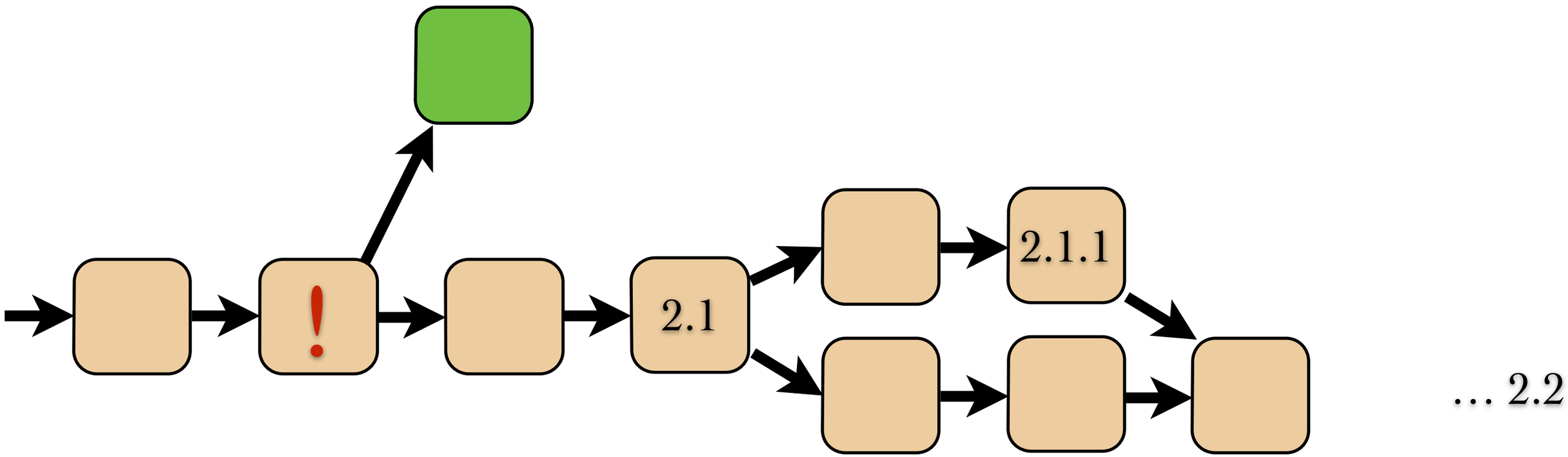
Typical usage patterns

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



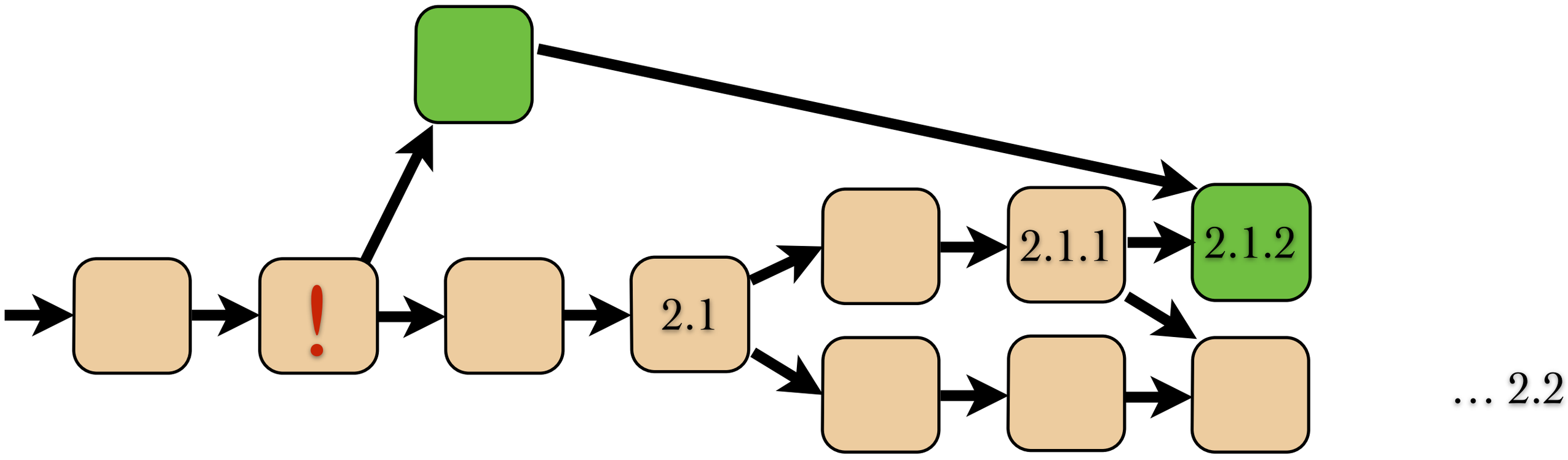
Typical usage patterns

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



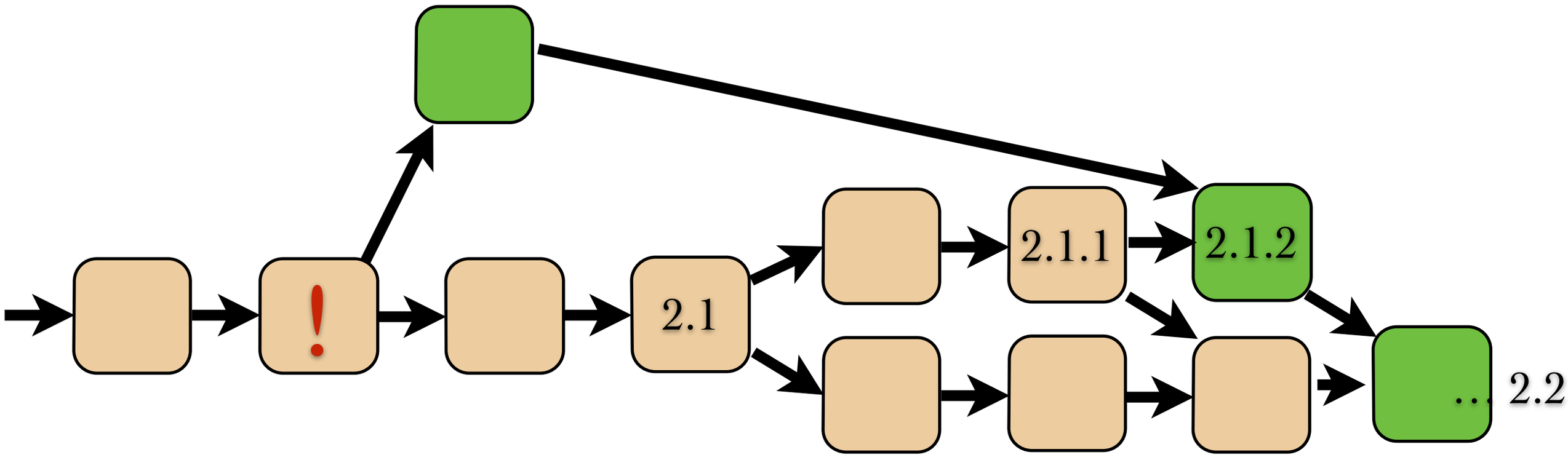
Typical usage patterns

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!

Summary

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

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!