
Gah!!!

KEEP CALM USE AND STASH

http://www.keepcalm-o-matic.co.uk/p/keep-calm-and-use-git-reflog/

```
git status
```

git status

```
# On branch getCommits
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       new file:   src/com/looselytyped/galore/bla.clj
#
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
#
#       modified:   .gitignore
#       modified:   src/com/looselytyped/galore/commit.clj
#       modified:   src/com/looselytyped/galore/fs.clj
#
```

```
git status
```

```
# On branch getCommits
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       new file:   src/com/looselytyped/galore/bla.clj
#
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
#
#       modified:   .gitignore
#       modified:   src/com/looselytyped/galore/commit.clj
#       modified:   src/com/looselytyped/galore/fs.clj
#
```

```
git stash
```

```
git status
```

```
# On branch getCommits
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       new file:   src/com/looselytyped/galore/bla.clj
#
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
#
#       modified:   .gitignore
#       modified:   src/com/looselytyped/galore/commit.clj
#       modified:   src/com/looselytyped/galore/fs.clj
#
```

```
git stash
```

```
git status
```

```
git status
```

```
# On branch getCommits
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       new file:   src/com/looselytyped/galore/bla.clj
#
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
#
#       modified:   .gitignore
#       modified:   src/com/looselytyped/galore/commit.clj
#       modified:   src/com/looselytyped/galore/fs.clj
#
```

```
git stash
```

```
git status
```

```
# On branch master
nothing to commit, working directory clean
```


Yay!!!

Stash

Stores the index and working tree state

Resets to the current commit!

Stash

Is just like a commit

Does not participate in the log

Stash

```
git stash

git stash save <message>
```

Stash

```
git stash list

git stash apply/pop
```

Quiz Time!

This is your "git status"

```
› git status
On branch master
Changes to be committed:
        new file:   src/HelloWorld.spec.js
Changes not staged for commit:
        modified:   src/HelloWorld.js
Untracked files:
        src/lib.js
```

- [ ] What will "git stash save 'some message'" do? What will "git status" report after stashing?

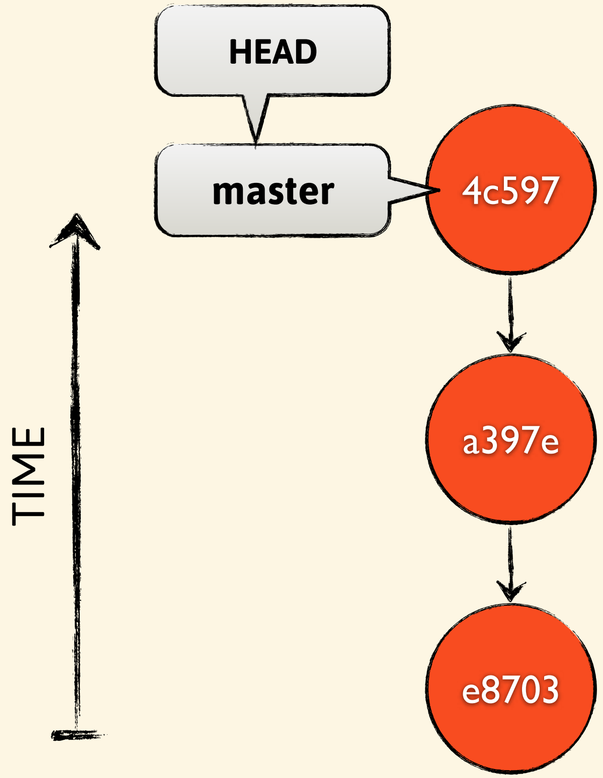
Exercise

- Edit some committed files, add a few new files, and add some of the new files to the index
- Stash your work using "git stash save 'Some message'" — Check "git status"
- Perform a "git stash apply" followed by a "git stash list"
- Then perform a "git stash pop" followed by a "git stash list" - What's the difference?

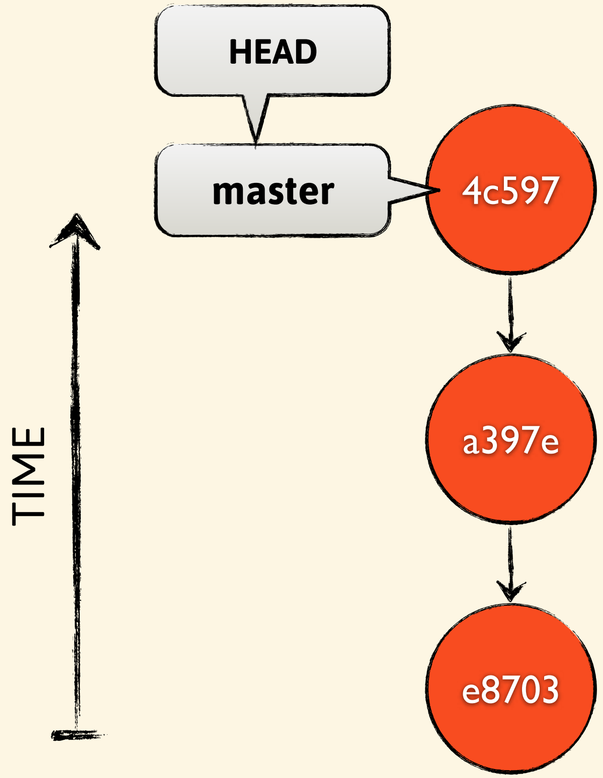
Quiz Time!

- [ ] Can you think of a better workflow than using stashes?

A better workflow

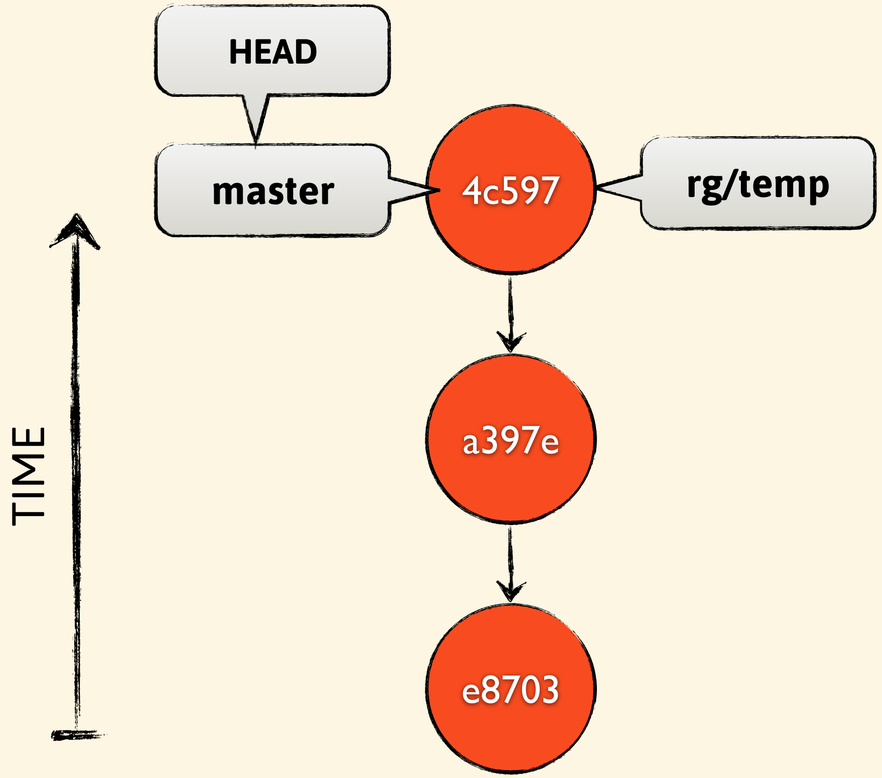


A better workflow



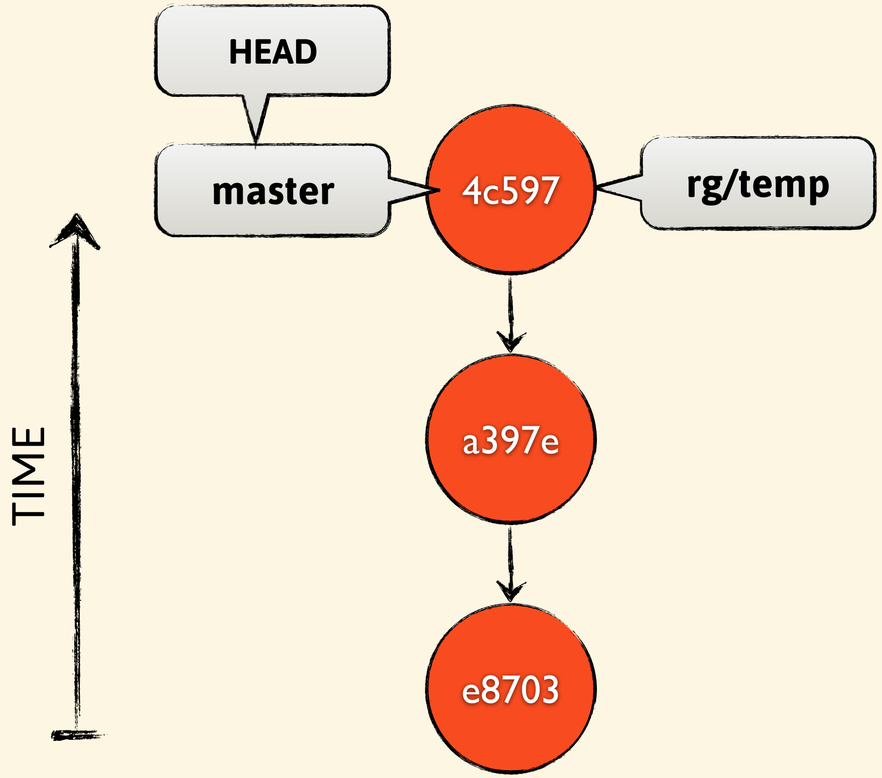

```
(master) > git branch rg/temp
```

A better workflow



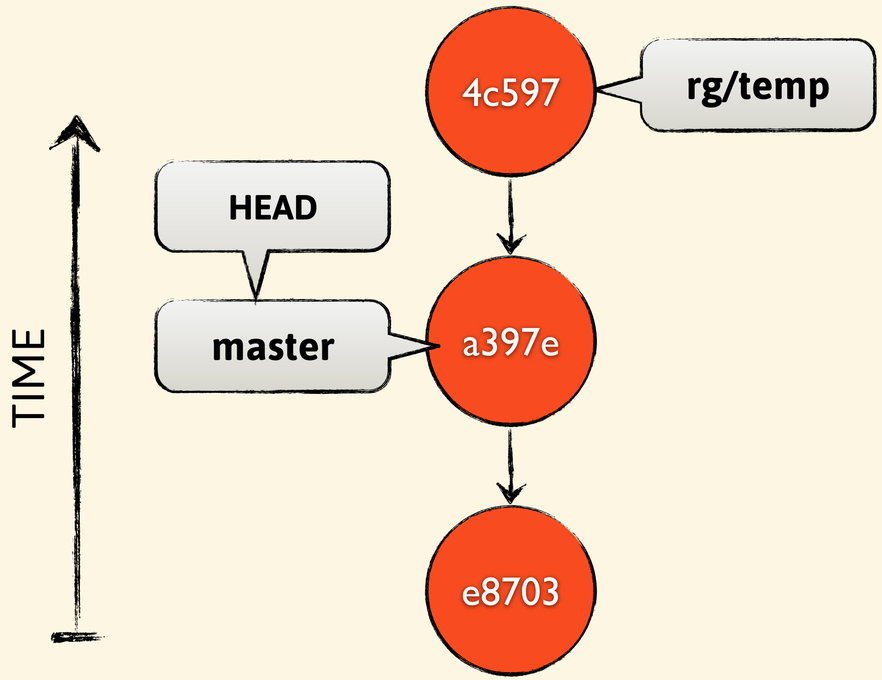

```
(master) > git branch rg/temp
```

A better workflow




```
(master) > git branch rg/temp
(master) > git reset HEAD~1
```

A better workflow




```
(master) > git branch rg/temp
(master) > git reset HEAD~1
```

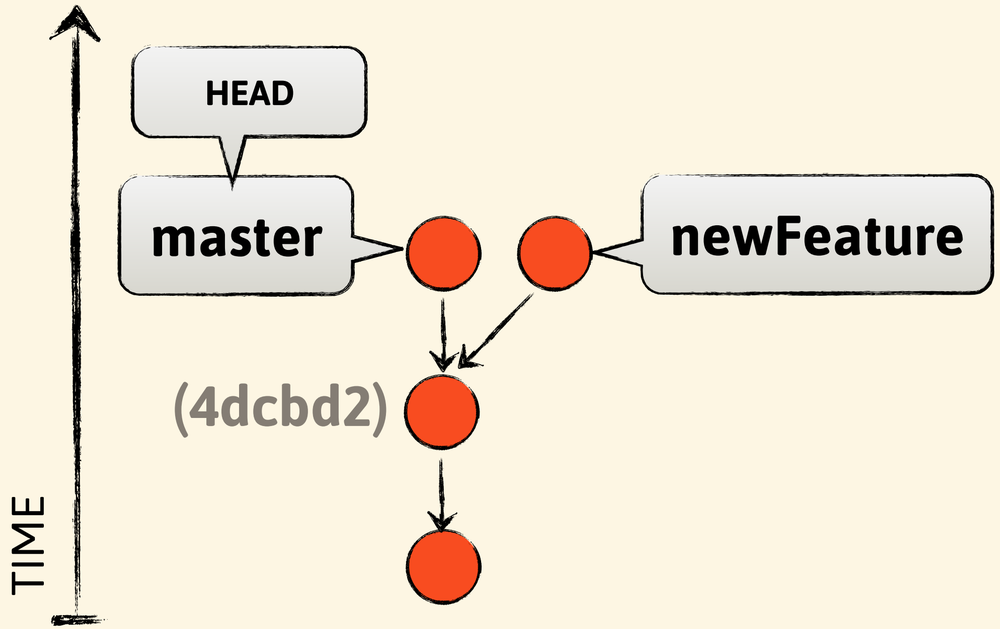
Undo

SORTA …

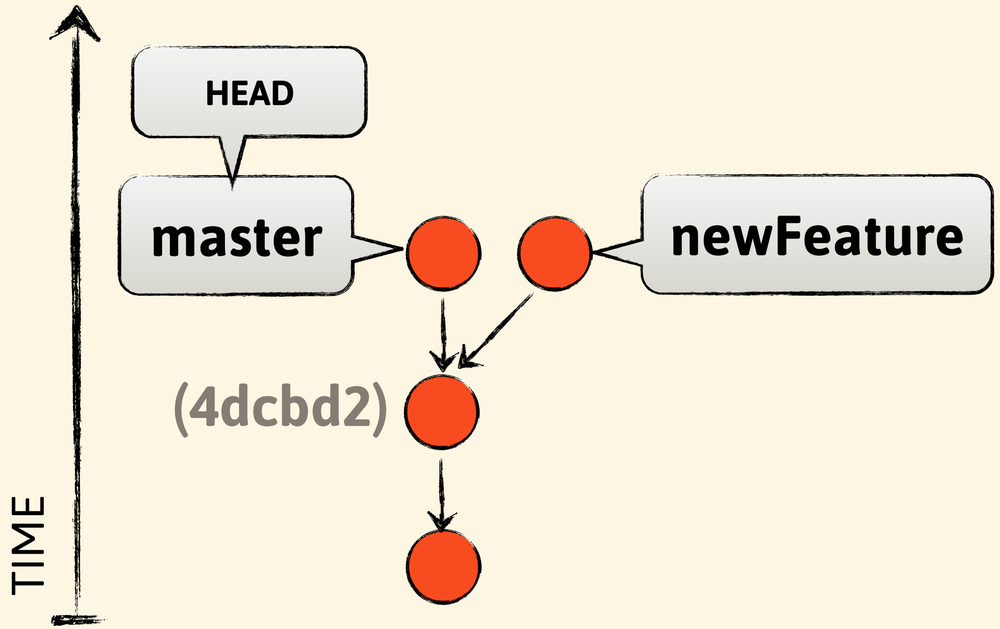
Reset

Moves the HEAD and branch

Reset --soft

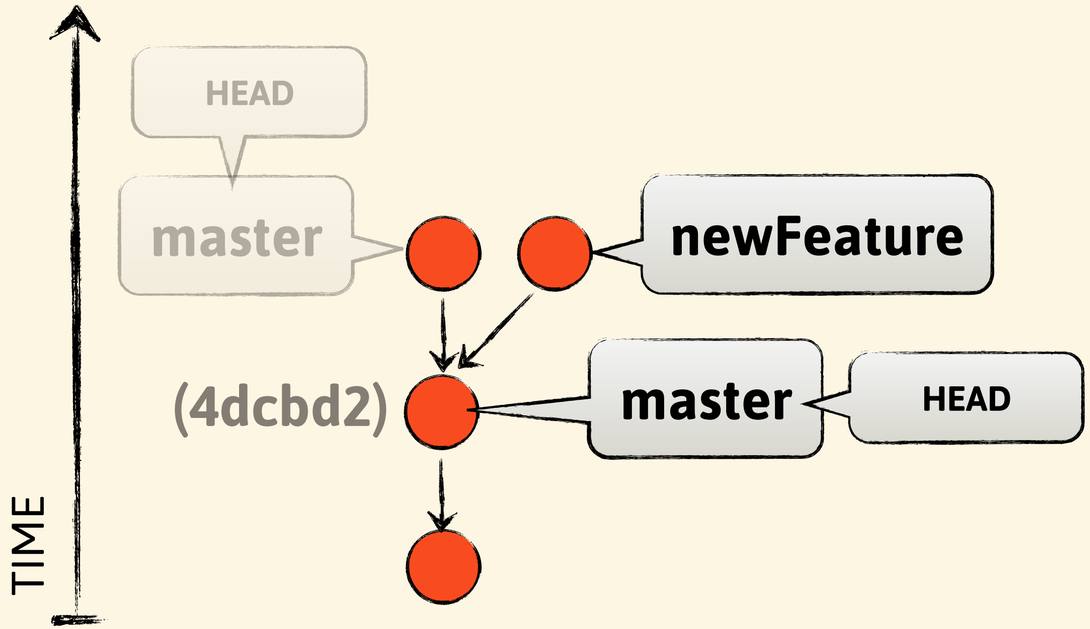


Reset --soft




```
(master) > git reset --soft 4dcbd2
```

Reset --soft




```
(master) > git reset --soft 4dcbd2
```

Quiz Time!

- [ ] Do you think it's prudent to use "git reset" on integration branches?

Exercise

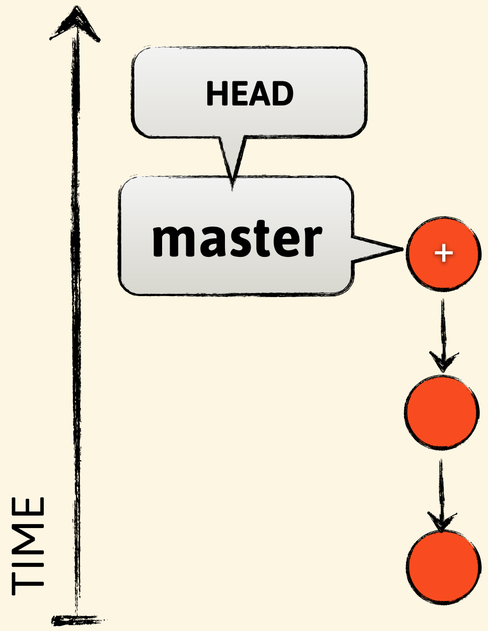
- Create two files in your repository — Let's call them A.txt, and B.txt
- Add A.txt to the index (Be sure to check your status here)
- Perform a "git reset HEAD~1"
  - Check "git status" — Explain what you see
- Use reflog to set your branch back to where it was (git reset <previous_sha>)
- Perform a "git reset --hard HEAD~1"
  - Check "git status" — Explain what you see

NEVER RESET PUBLIC COMMITS

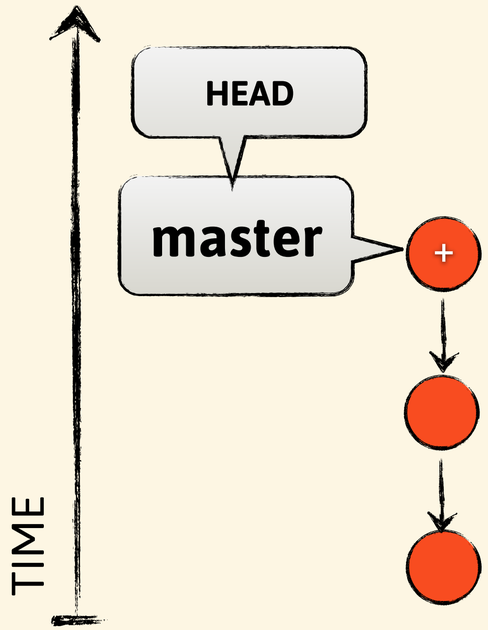
Revert

Undos the commit and create a new commit

Revert

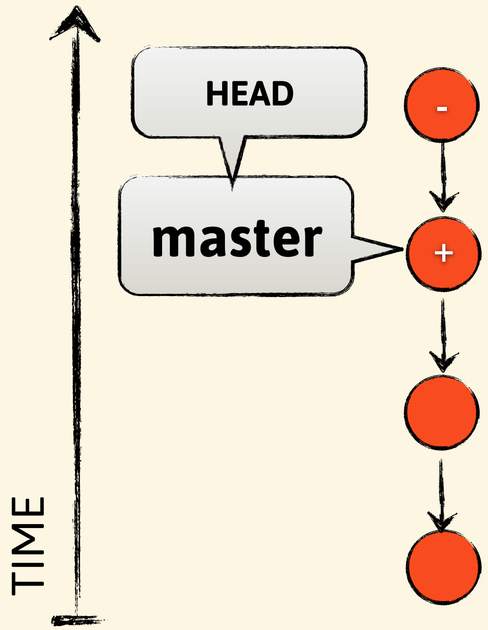


Revert



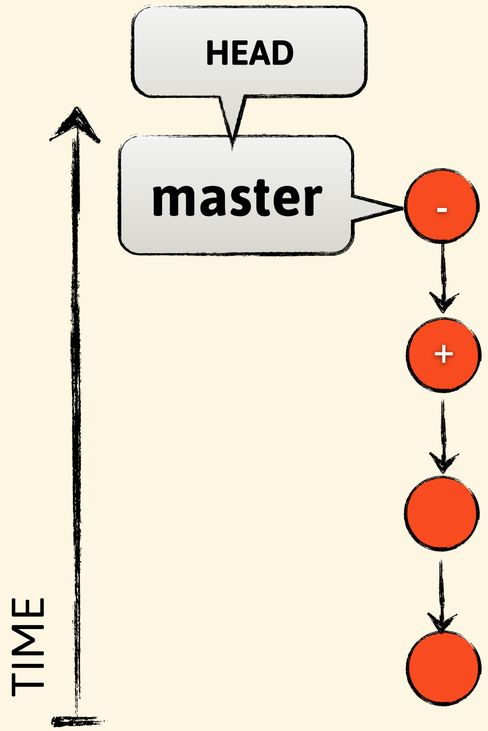

```
(master) > git revert HEAD
```

Revert




```
(master) > git revert HEAD
```

Revert




```
(master) > git revert HEAD
```

Exercise

- Perform a "git --no-pager log --oneline" and find a commit
- Perform a "git show <commit-sha>" to see what the delta was
- Perform a "git revert <commit-sha>" followed by a "git --no-pager show HEAD" to see if the delta is the opposite of the original commit

Merge

Happily, forever

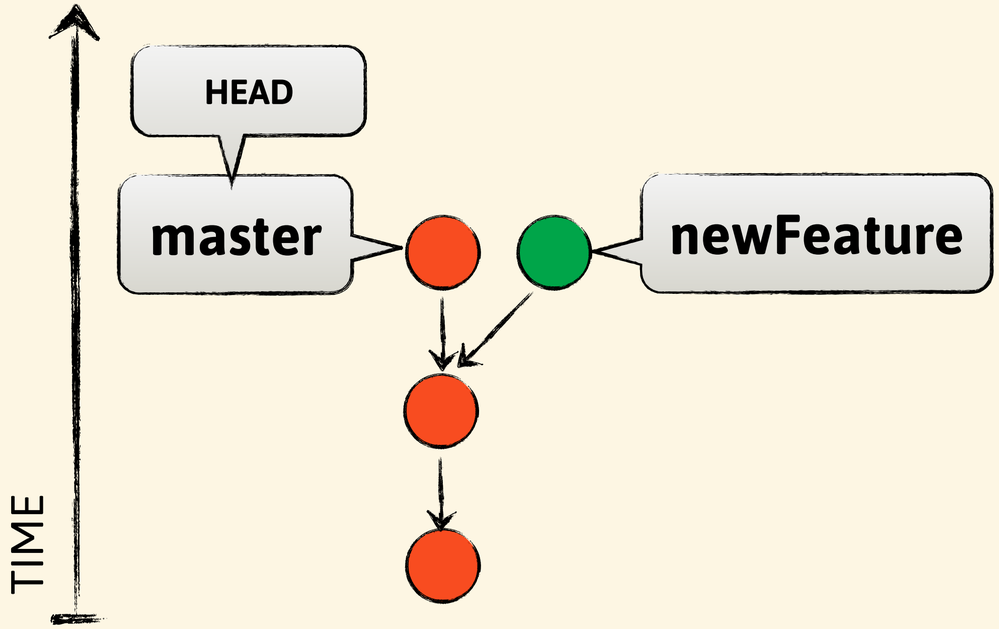
Merge

Joins two or more commits

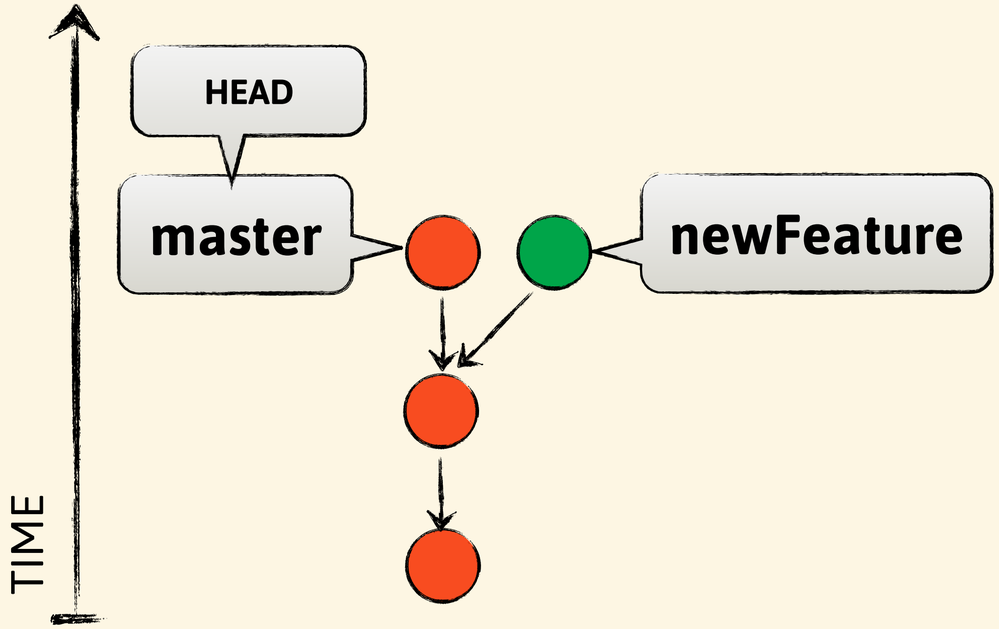
Merge

(May) Create a child commit

Merge

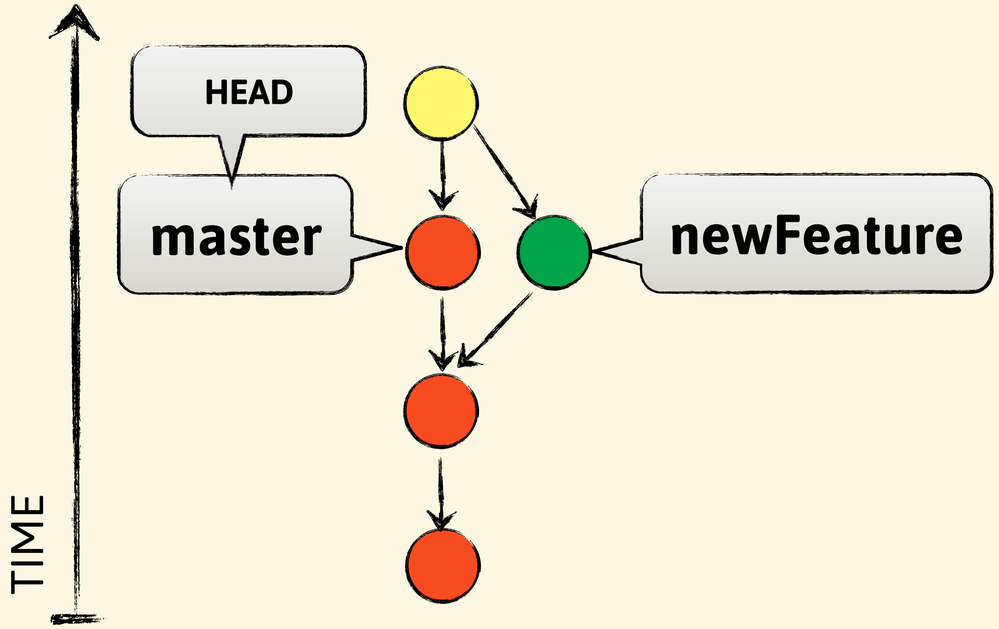


Merge



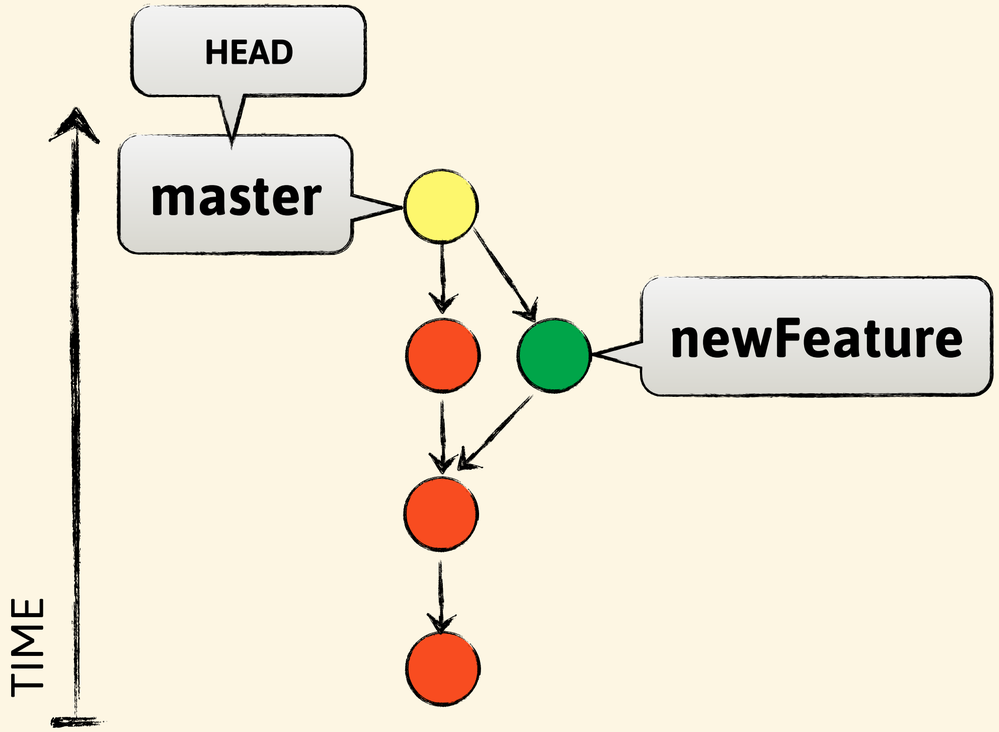

```
(master) > git merge newFeature
```

Merge



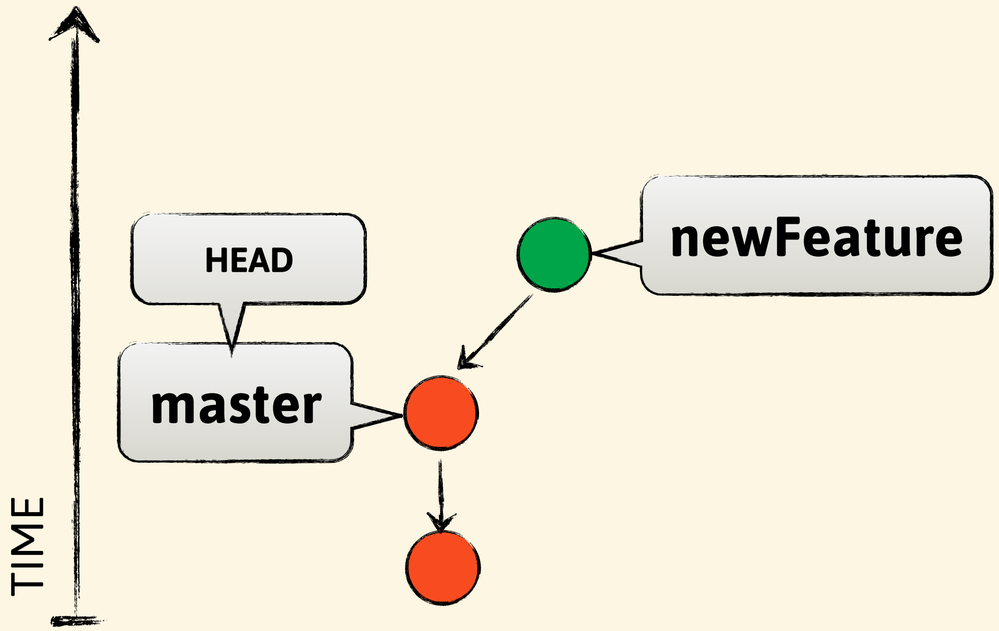

```
(master) > git merge newFeature
```

Merge

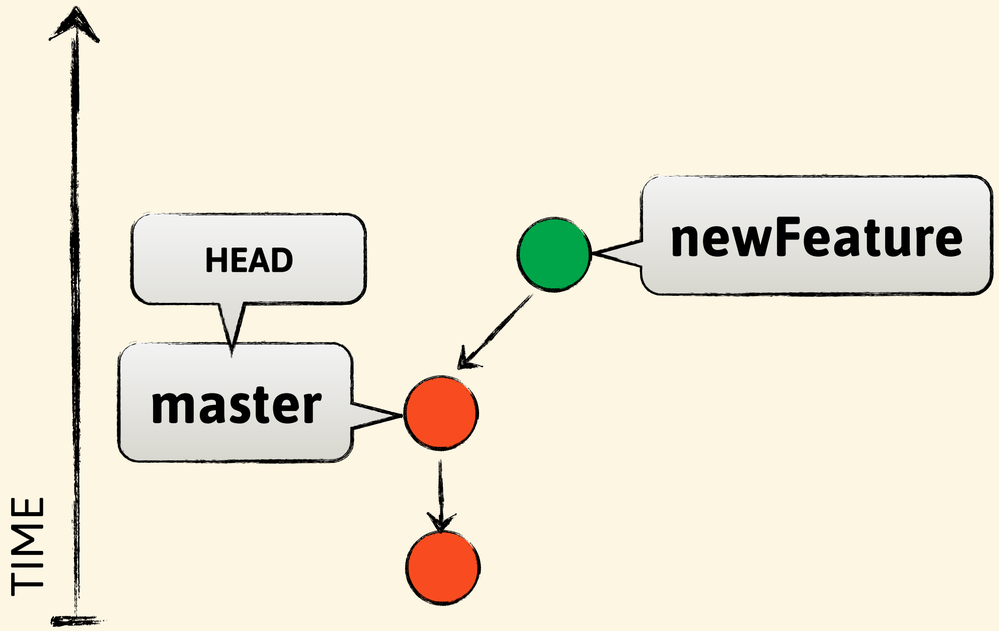



```
(master) > git merge newFeature
```

(Fast-forward) Merge

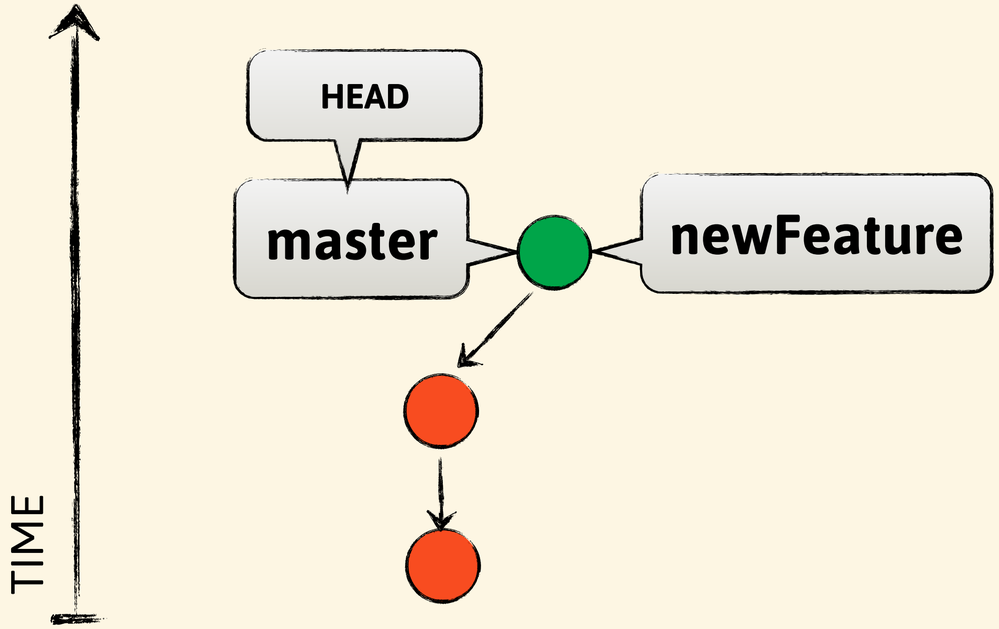


(Fast-forward) Merge




```
(master) > git merge newFeature
```

(Fast-forward) Merge




```
(master) > git merge newFeature
```

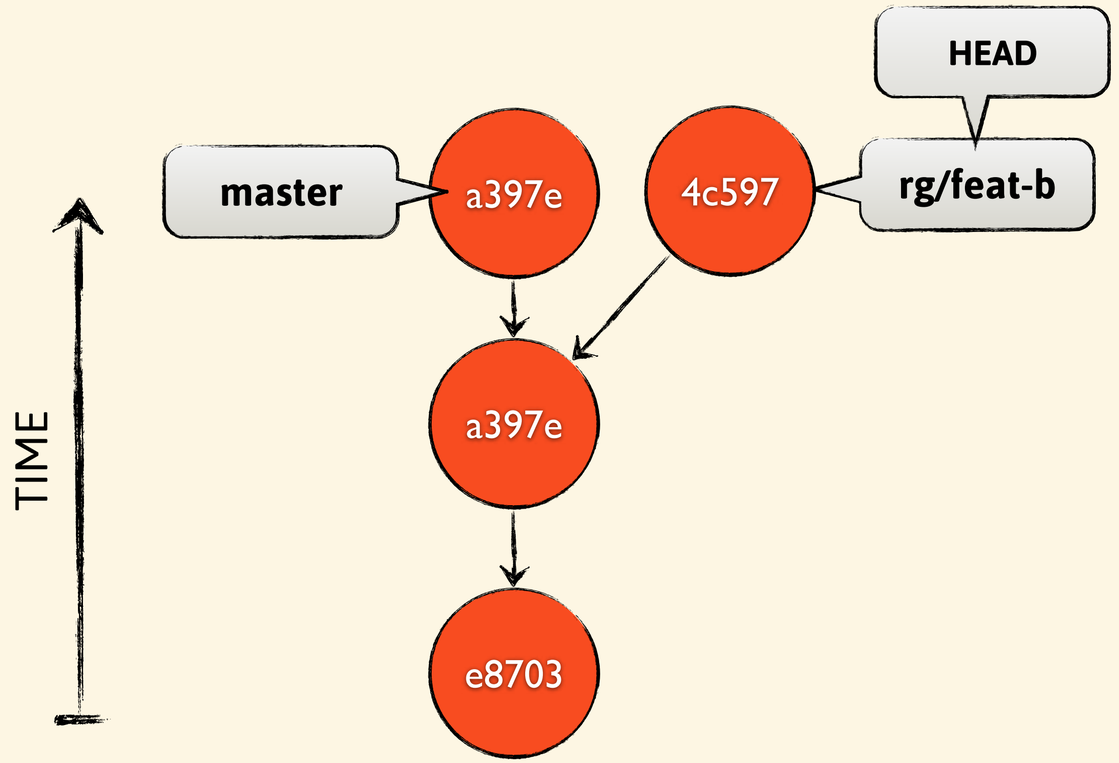
USE MERGE FOR PUBLIC HISTORY

?

Quiz Time!

Suppose your DAG looks like this. Will "git merge master" be a fast-forward merge? Explain your answer.

- ☐ Yes
- ☐ No



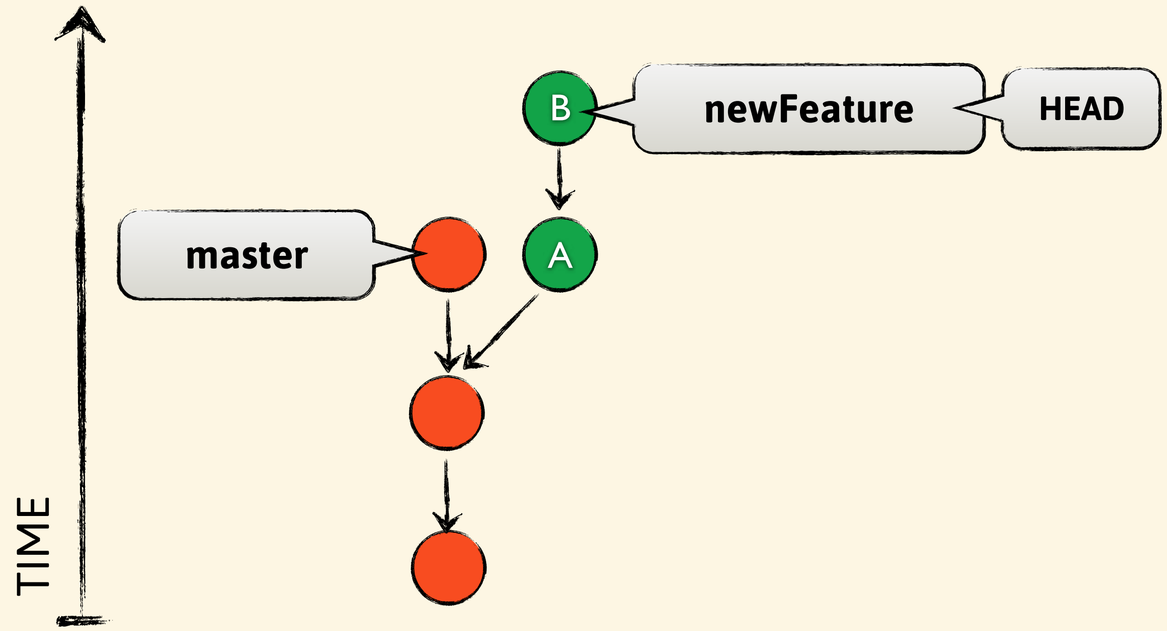
Rebase

The swiss army knife

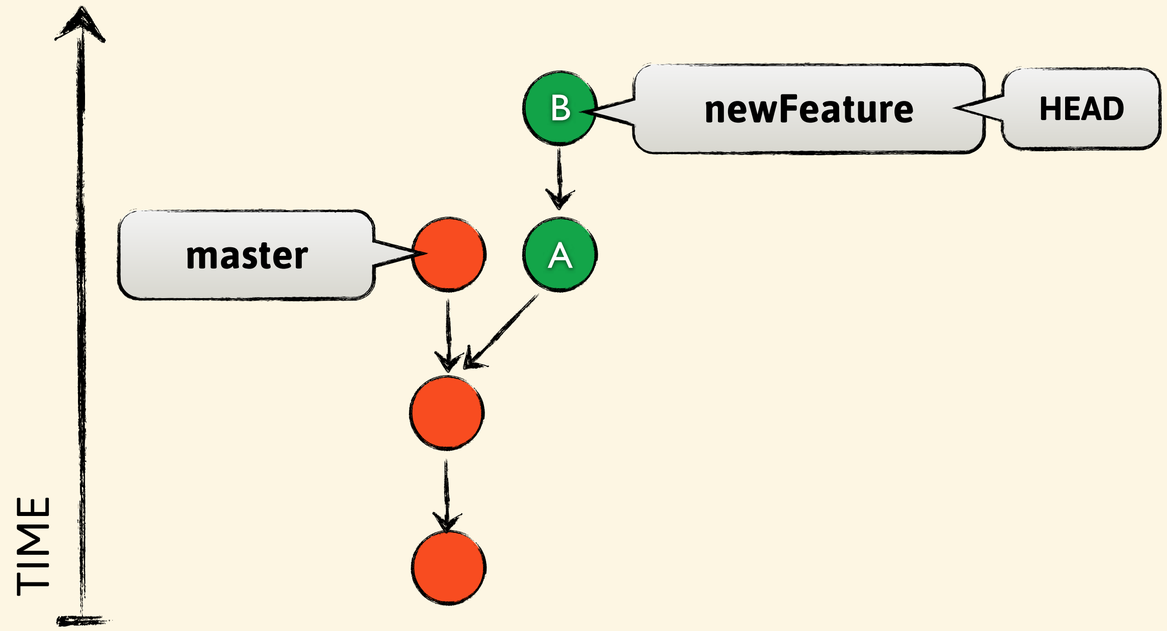
Rebase

Relocates a branch to a new parent

Rebase

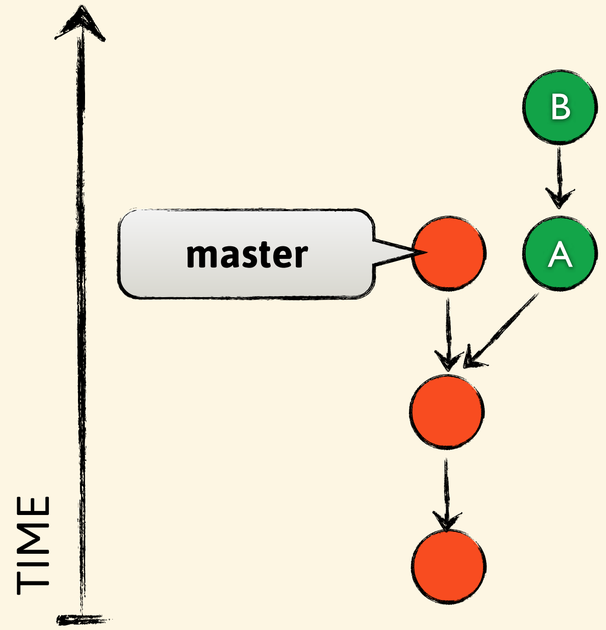


Rebase



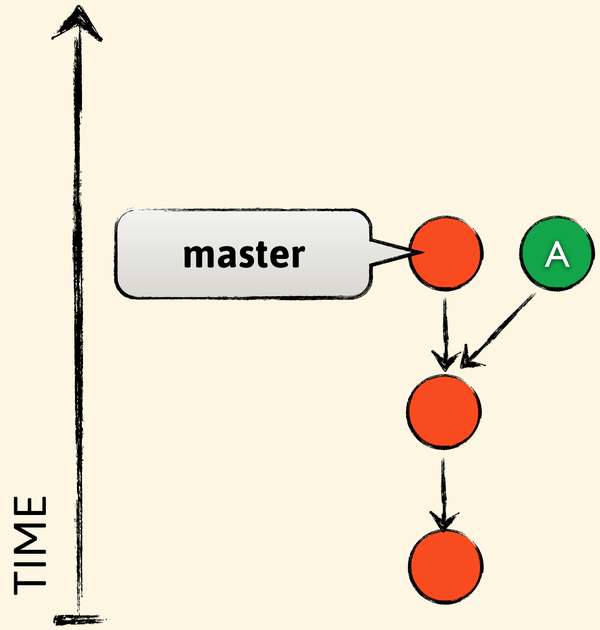

```
(newFeature) > git rebase master
```

Rebase



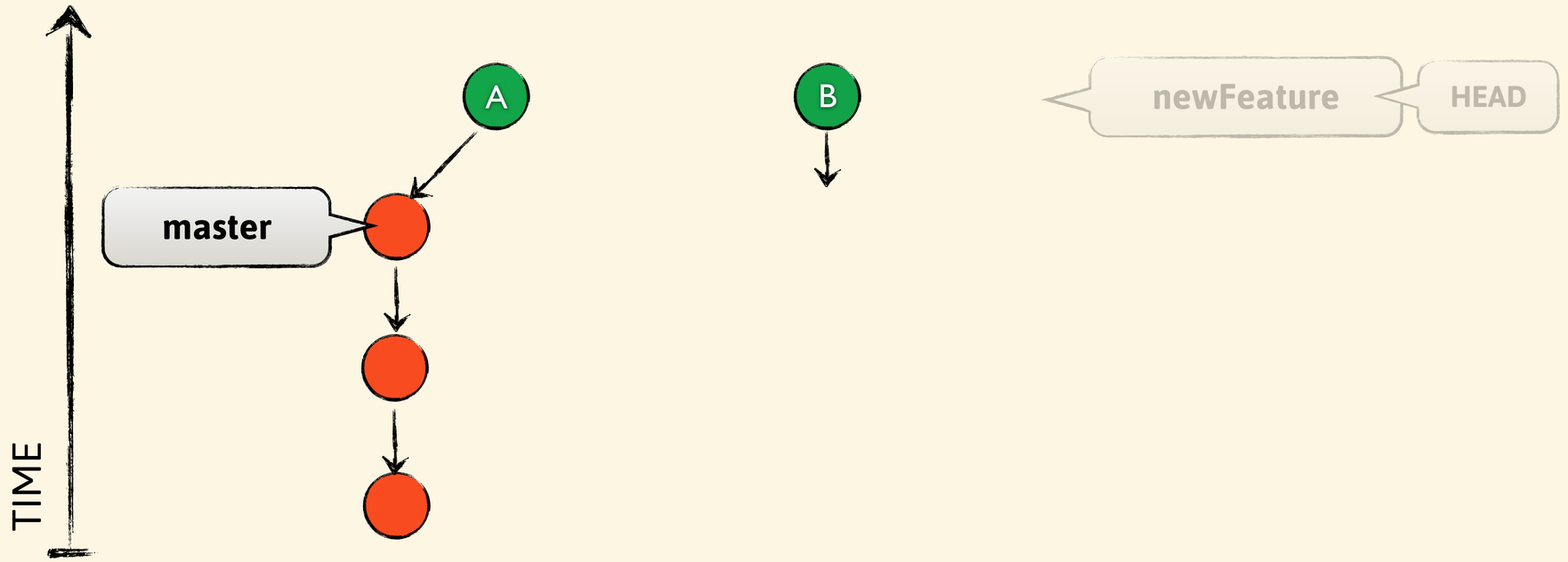

```
(newFeature) > git rebase master
```

Rebase



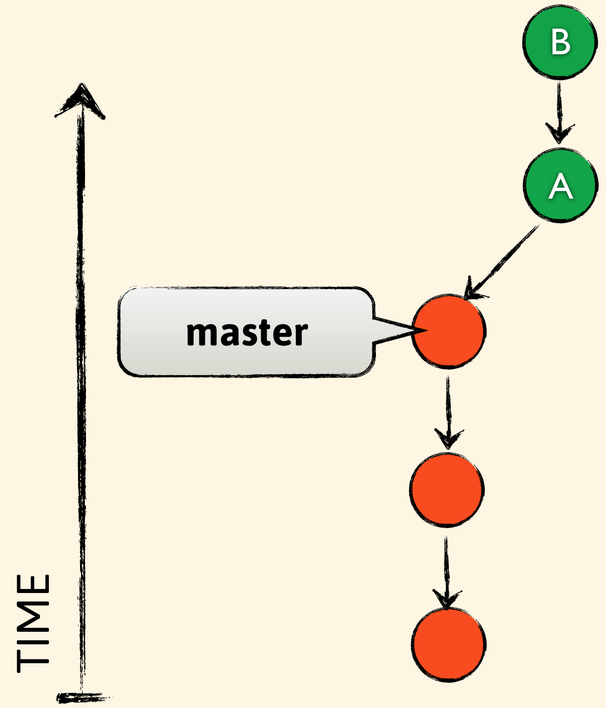

```
(newFeature) > git rebase master
```

Rebase



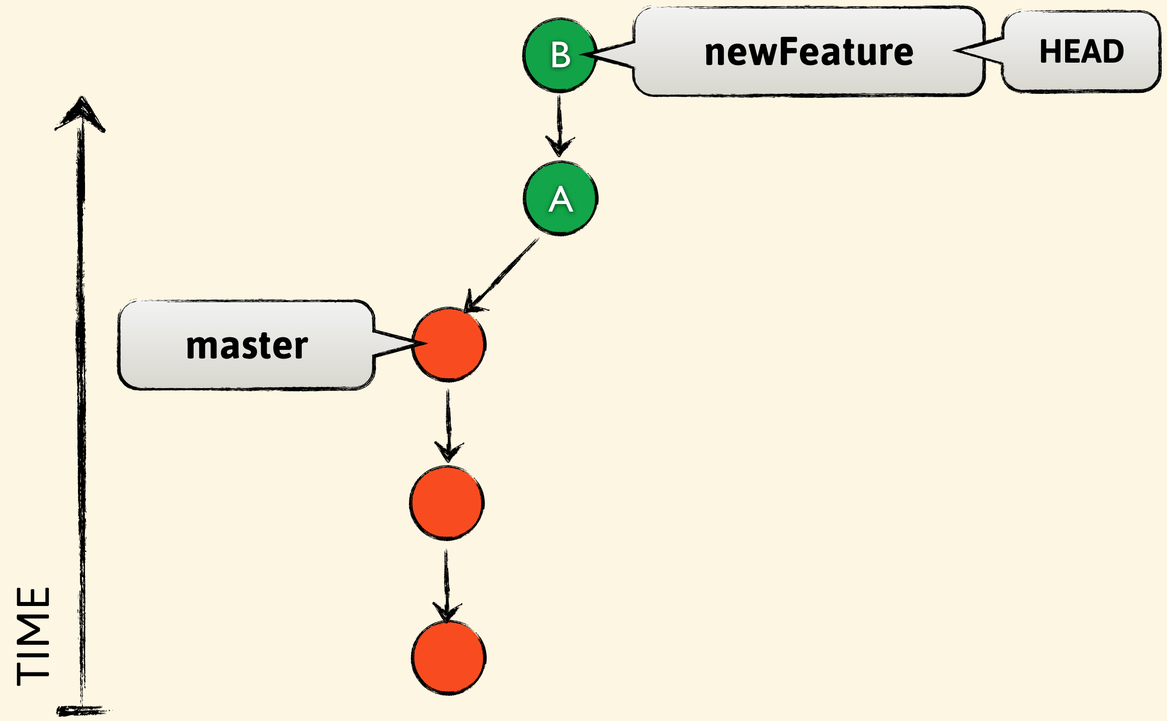

```
(newFeature) > git rebase master
```

Rebase




```
(newFeature) > git rebase master
```

Rebase

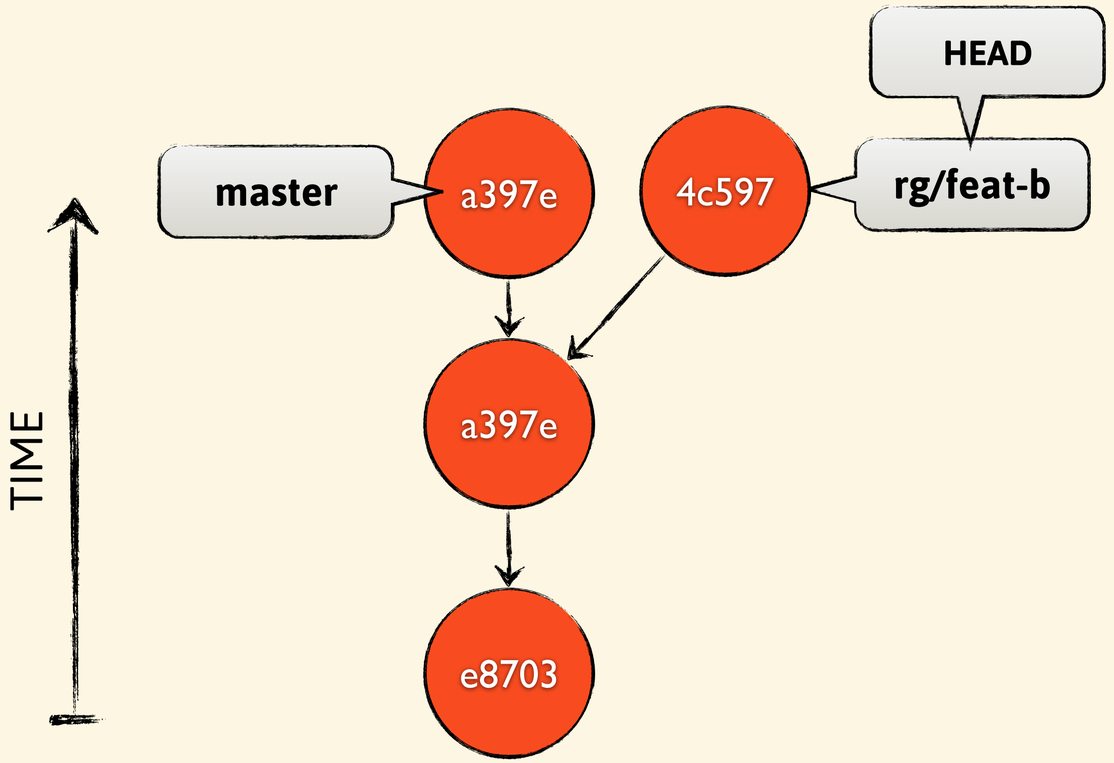



```
(newFeature) > git rebase master
```

USE **REBASE** FOR **PRIVATE COMMITS**

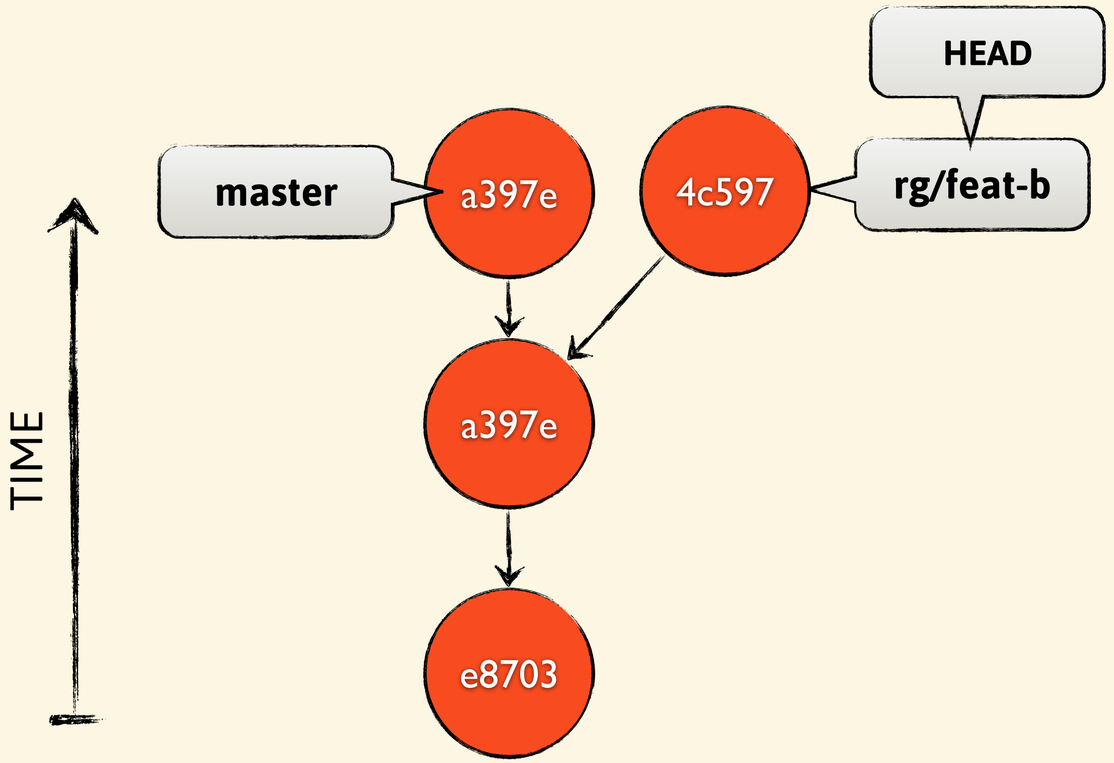
Exercise

Make it so that your DAG looks like this. Then perform a "git rebase master". Observe "git --no-pager log --oneline -n 10"



Exercise

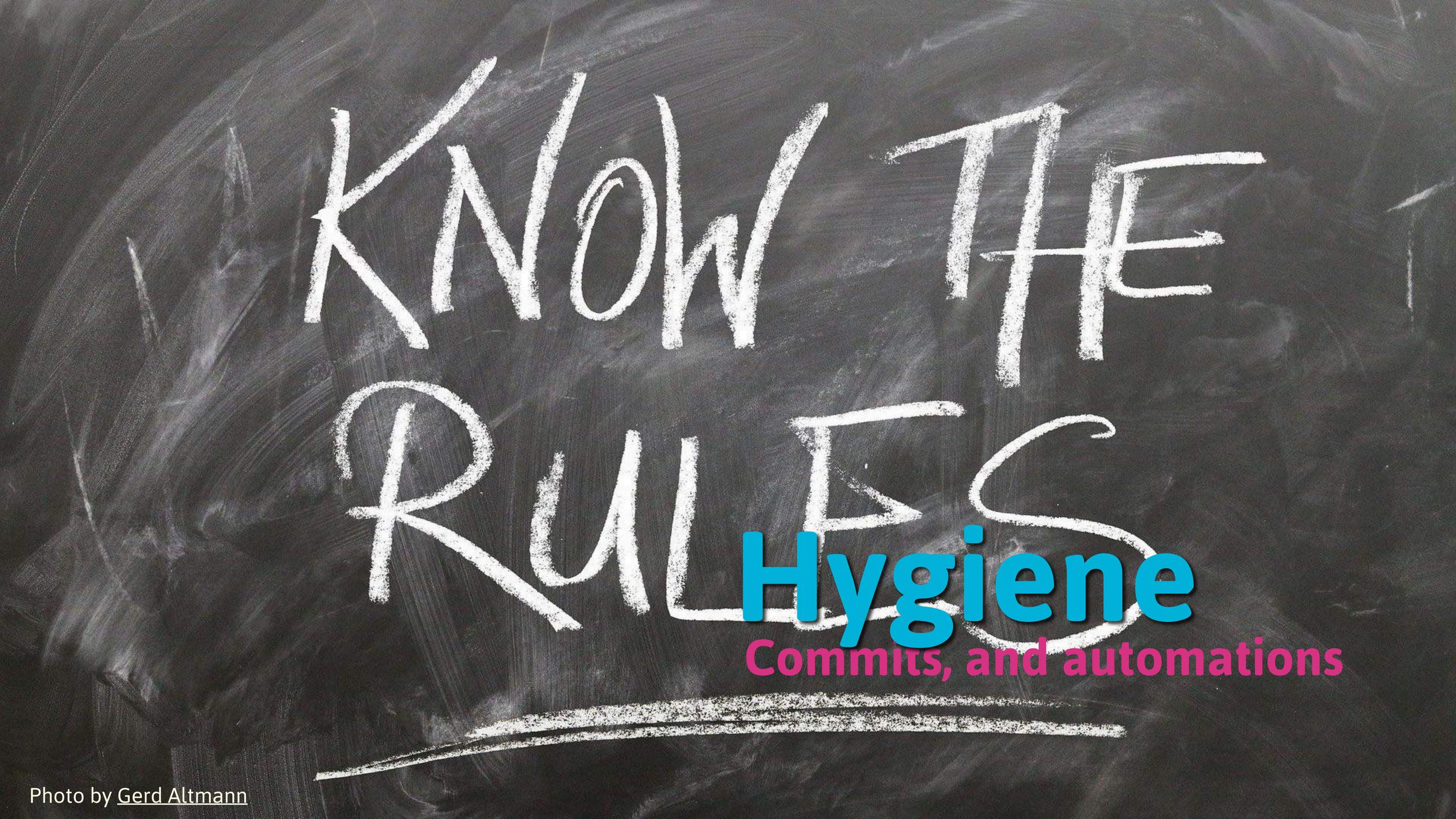
Make it so that your DAG looks like this. Then perform a "git rebase master". Observe "git --no-pager log --oneline -n 10"



Exercise

- Switch to any of your feature branches (or create one) - Add one or more commits
- Attempt a "git rebase -i master"
  - Try editing the message of one of your commits
- See the results of "git --no-pager log --oneline -n 10"

- Hints
  - Git will stop the rebase at the commit you are editing. You can use "git commit --amend" to edit the commit message
  - If you get in a bad state you can use "git rebase --abort"



Hygiene

Commits, and automations

Photo by Gerd Altmann

"**commit messages** to me are **almost as important** as the **code** change itself

- Linus Torvalds

```
1786586747 (origin/g3) fix(core): Refresh transplanted views at insertion point only (#35968)
45c09416ed refactor(ngcc): move `PathMappings` to separate file to avoid circular dependency (#36626)
4779c4b94a fix(ngcc): handle `ENOMEM` errors in worker processes (#36626)
793cb328de fix(ngcc): give up re-spawing crashed worker process after 3 attempts (#36626)
966598cda7 fix(ngcc): support recovering when a worker process crashes (#36626)
772ccf0d9f feat(ngcc): support reverting a file written by `FileWriter` (#36626)
ff6e93163f refactor(ngcc): keep track of transformed files per task (#36626)
dff5129661 refactor(ngcc): notify master process about transformed files before writing (#36626)
e367593a26 refactor(ngcc): support running callback before writing transformed files (#36626)
16039d837e refactor(ngcc): rename `TaskQueue#markTaskCompleted()` to `markAsCompleted()` (#36626)
4665c35453 feat(ngcc): support marking an in-progress task as unprocessed (#36626)
```

```
<type>: <subject>
<BLANK LINE>
<body>
<BLANK LINE>
<footer>
```

```
<type>: <short summary>
  │            │
  │            └─⫸ Summary in present tense. Not capitalized. No period at the end.
  │
  └─⫸ Commit Type: build|ci|docs|feat|fix|perf|refactor|style|test
```

Inspired by https://github.com/angular/angular/blob/master/CONTRIBUTING.md#commit

10.0.0-next.4 (2020-04-29)

Bug Fixes

- **compiler:** normalize line endings in ICU expansions (#36741) (70dd27f), closes #36725
- **core:** attempt to recover from user errors during creation (#36381) (3d82aa7), closes #31221
- **core:** handle synthetic props in Directive host bindings correctly (#35568) (f27deea), closes #35501
- **language-service:** disable update the `[@angular](https://github.com/angular)/core` module (#36783) (dd049ca)
- **localize:** include legacy ids when describing messages (#36761) (47f9867)
- **ngcc:** recognize enum declarations emitted in JavaScript (#36550) (89c5890), closes #35584

Features

- **router:** allow CanLoad guard to return UrlTree (#36610) (00e6cb1), closes #26521 #28306

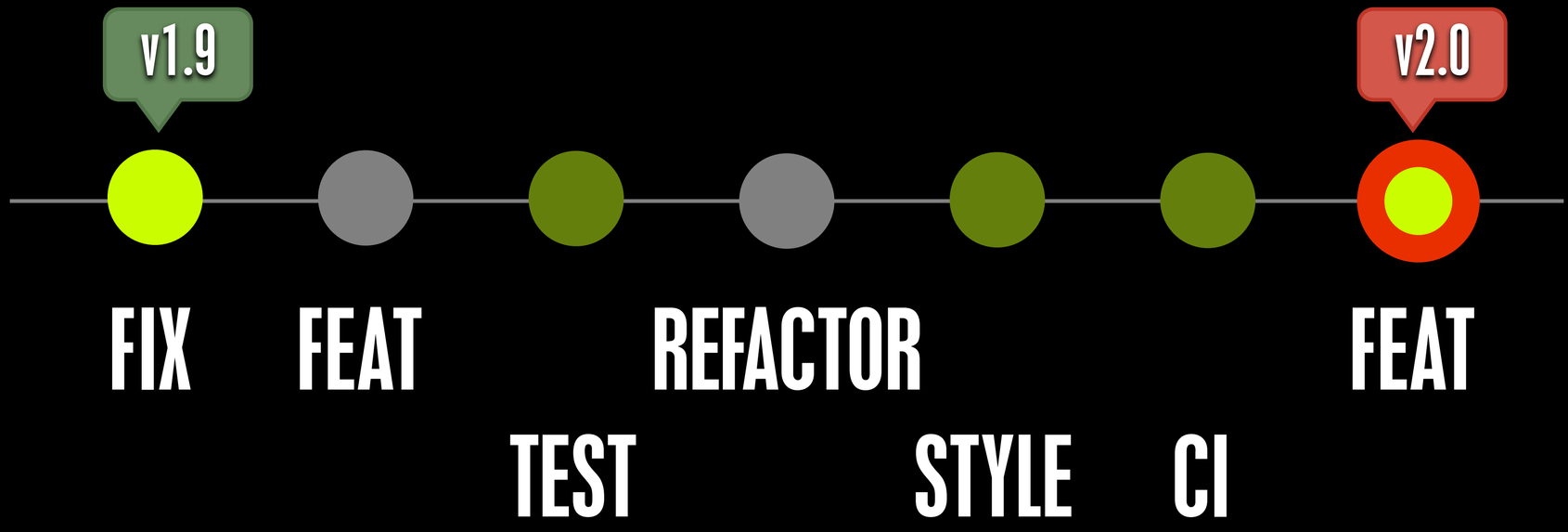
9.1.4 (2020-04-29)

Bug Fixes

- **core:** attempt to recover from user errors during creation (#36381) (d743331), closes #31221
- **core:** handle synthetic props in Directive host bindings correctly (#35568) (0f389fa), closes #35501
- **language-service:** disable update the `[@angular](https://github.com/angular)/core` module (#36783) (d3a77ea)
- **localize:** include legacy ids when describing messages (#36761) (aa94cd5)
- **ngcc:** recognize enum declarations emitted in JavaScript (#36550) (c440165), closes #35584

10.0.0-next.3 (2020-04-22)

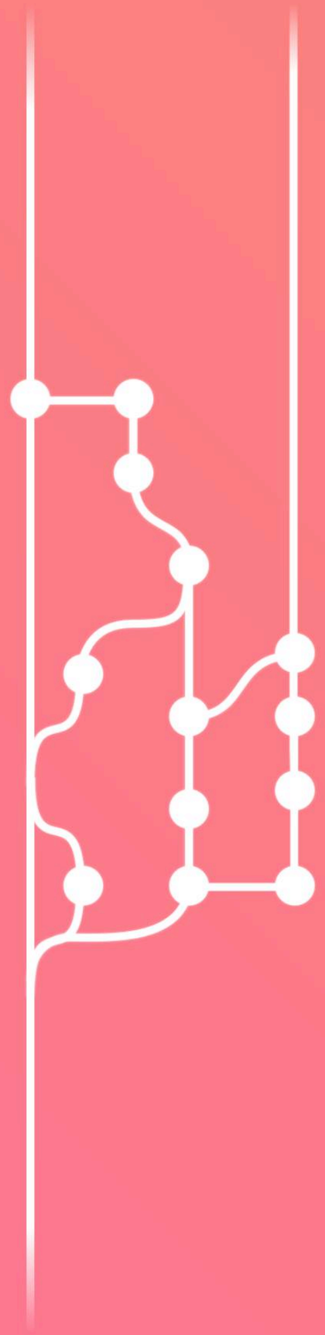
GIT BISECT



Conventional Commits

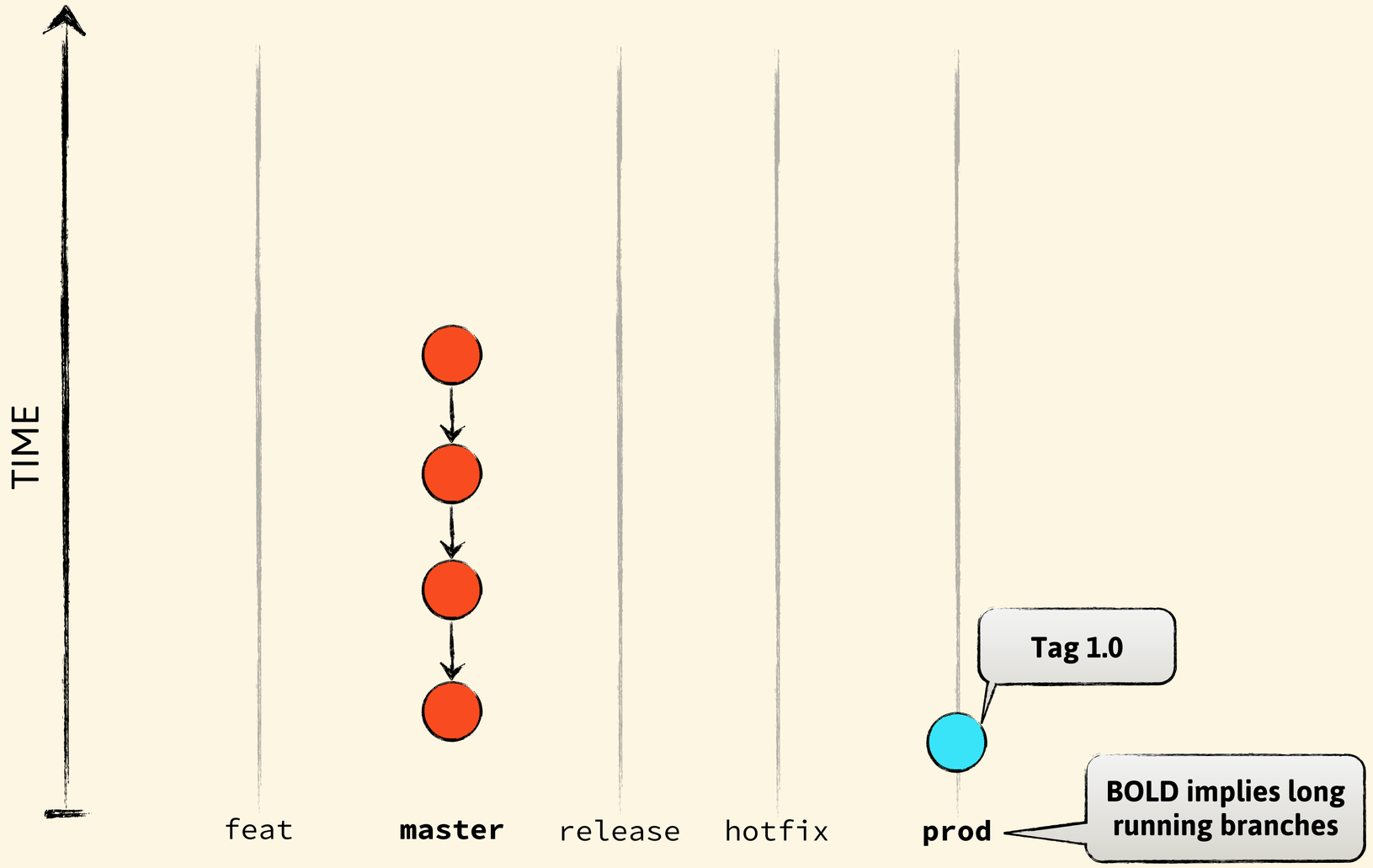
A specification for adding human and machine readable meaning to commit messages

Quick Summary | Full Specification | Contribute

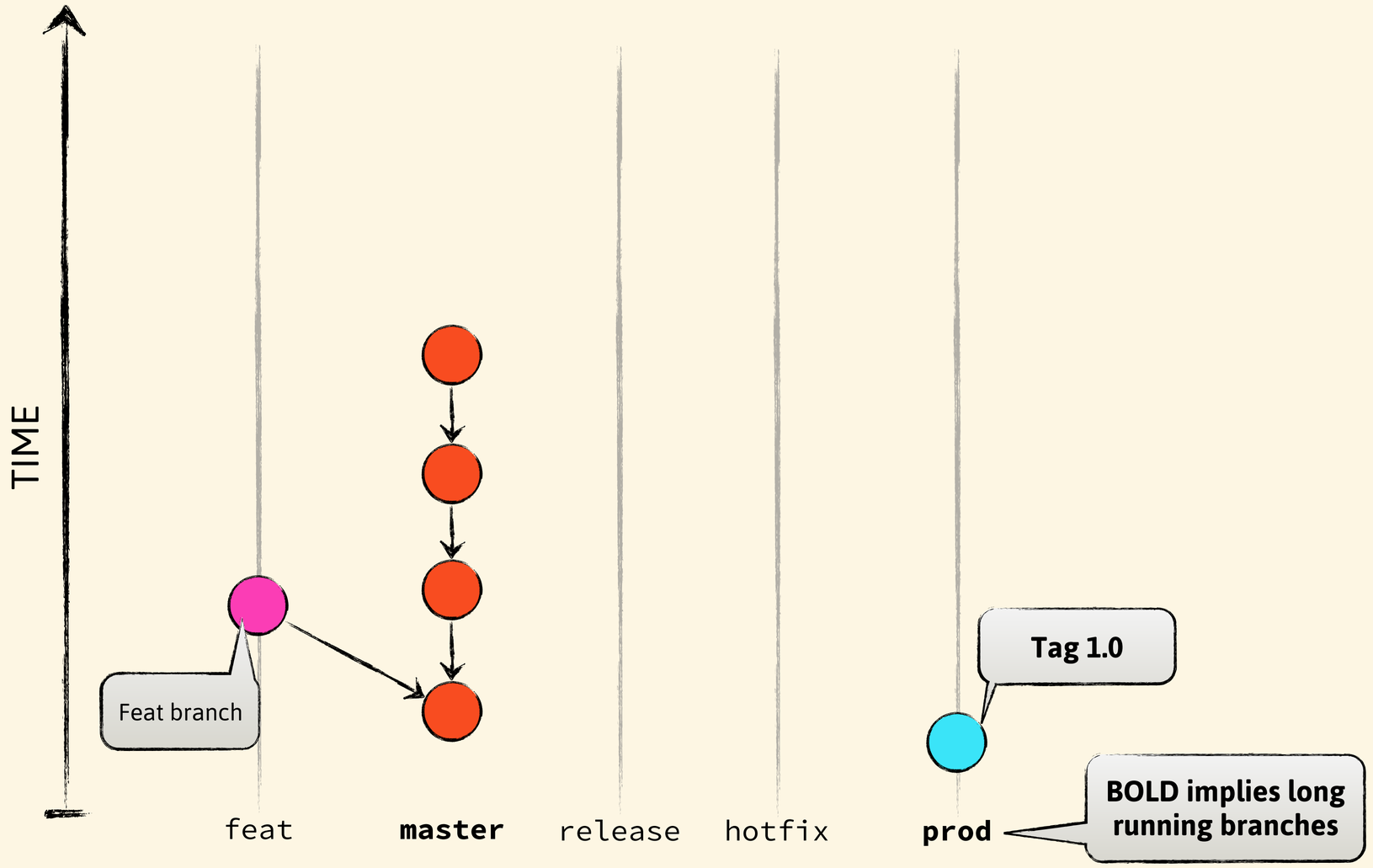



Workflows
Strategies for your team

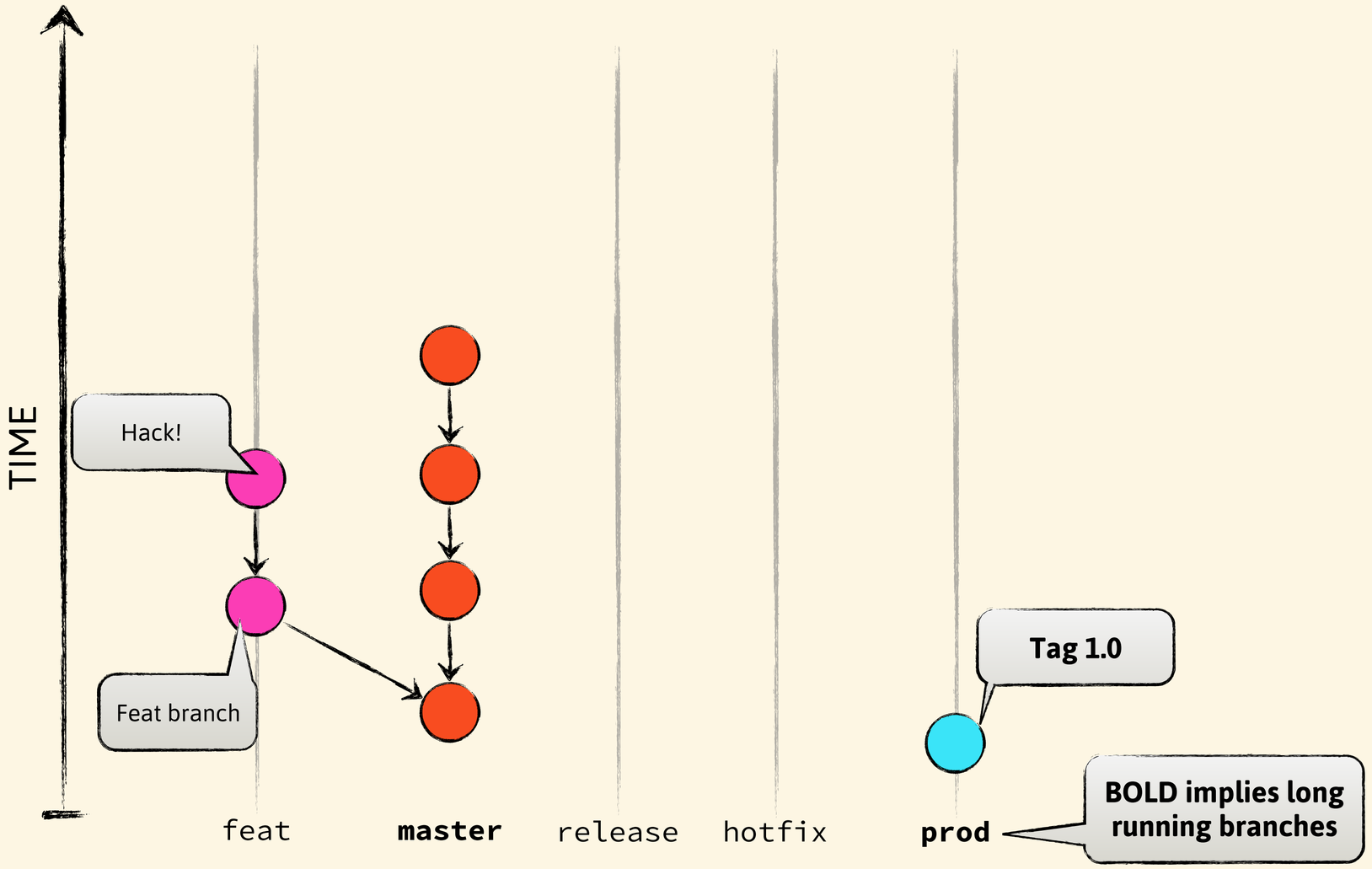
Gitflow


TIME
feat
master
release
hotfix
prod
Tag 1.0
BOLD implies long running branches

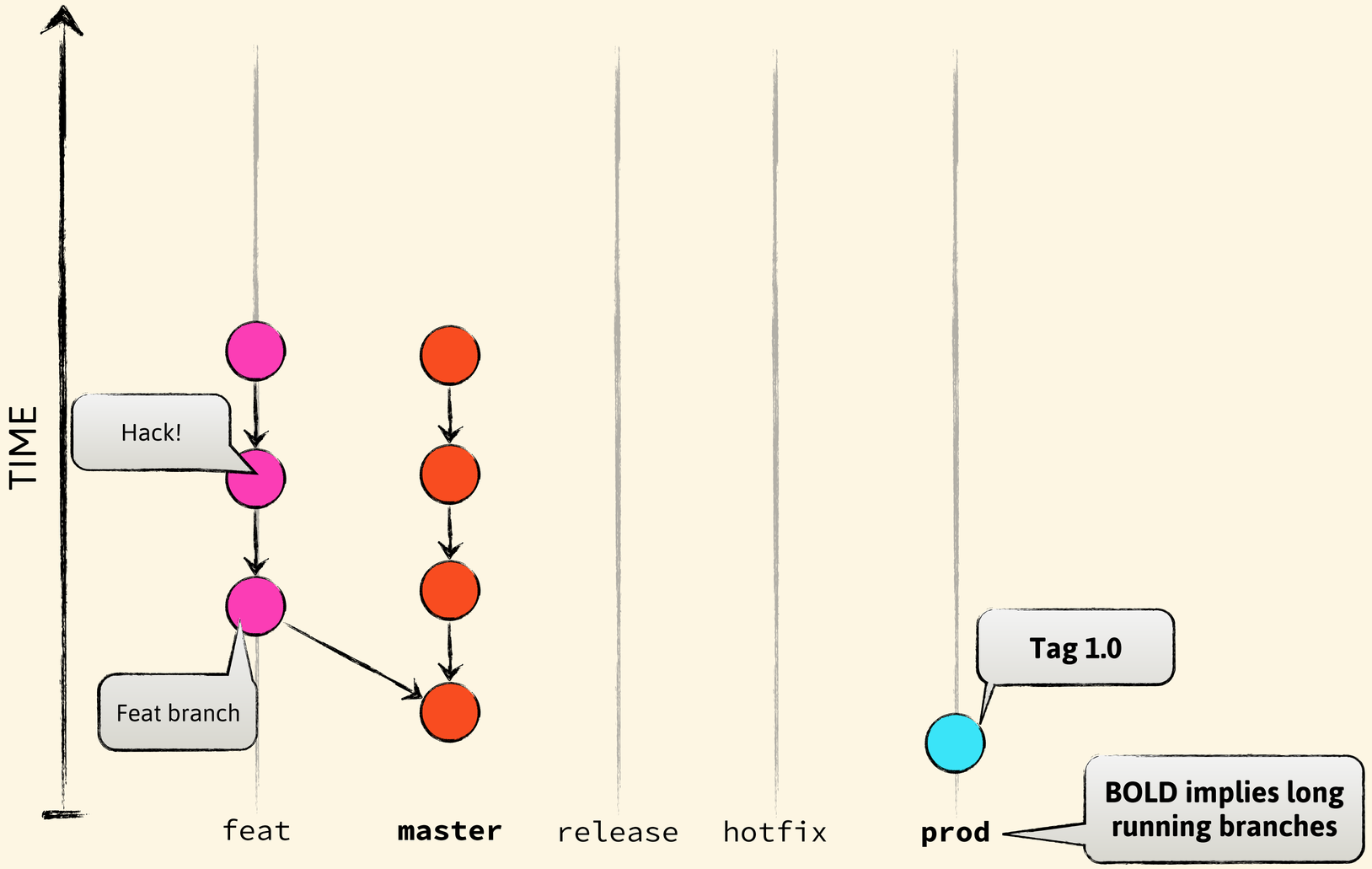
Gitflow


TIME
Feat branch
Tag 1.0
BOLD implies long running branches
feat
master
release
hotfix
prod

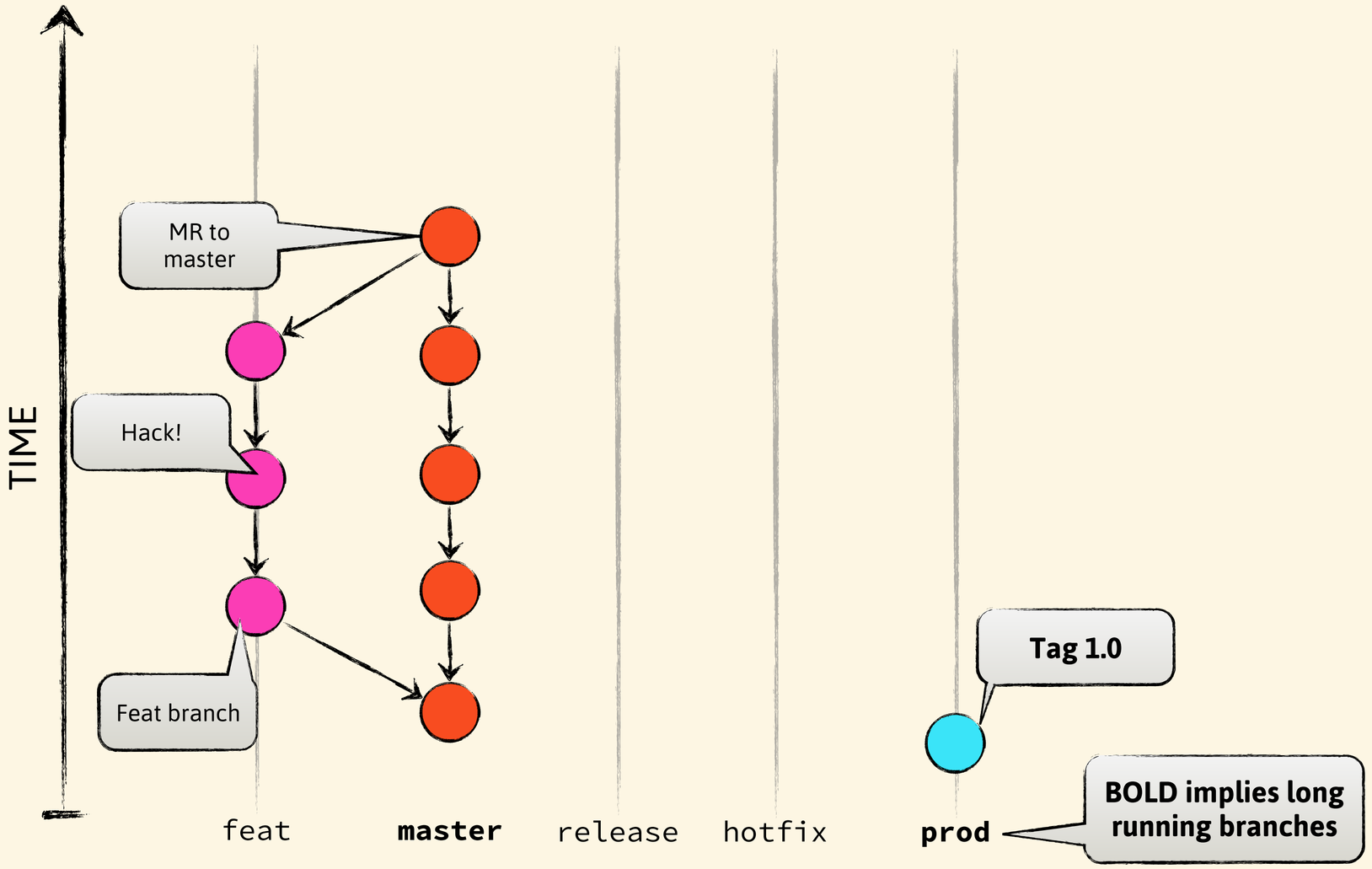
Gitflow


TIME
Hack!
Feat branch
Tag 1.0
BOLD implies long running branches
feat
master
release
hotfix
prod

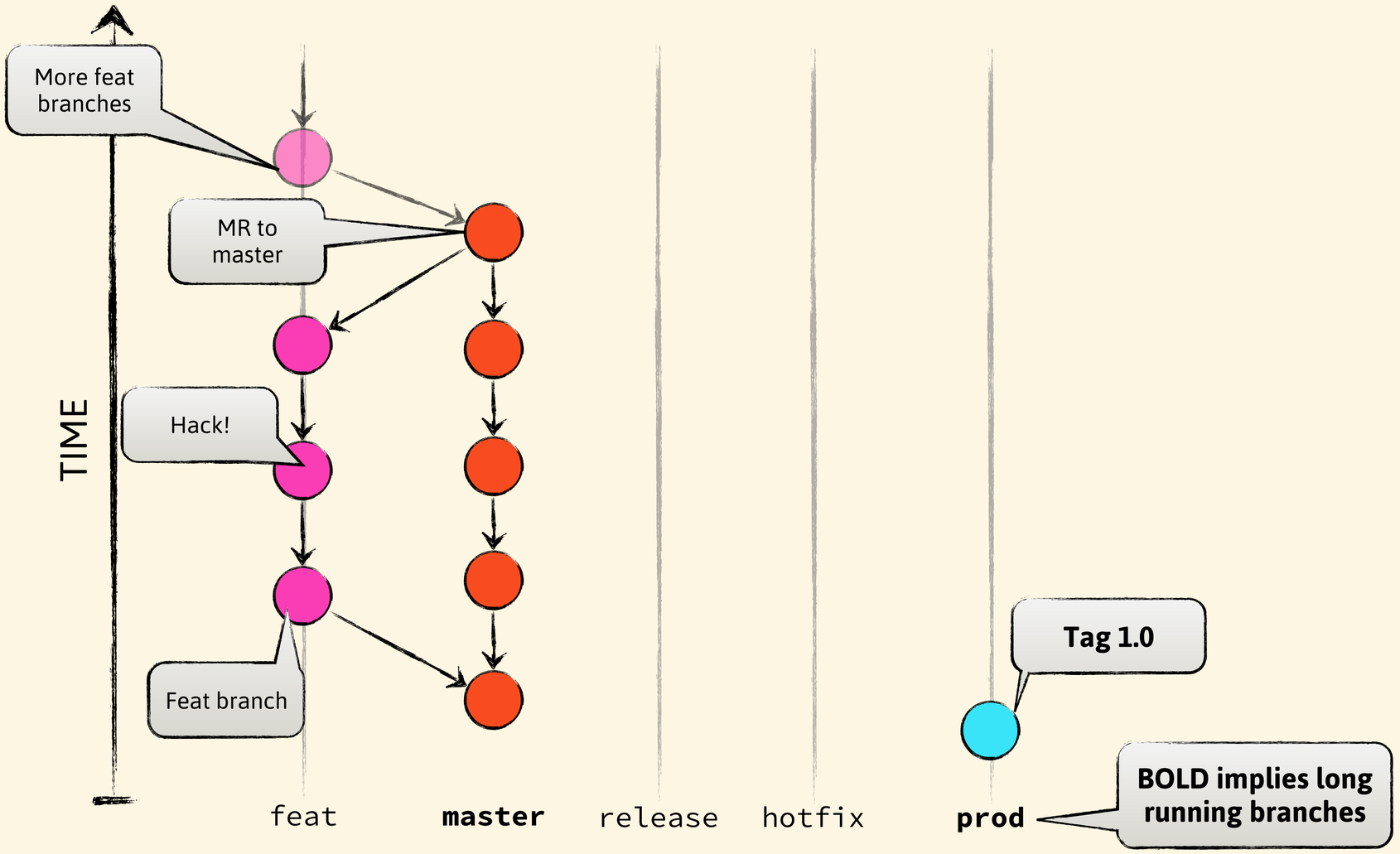
Gitflow


TIME
Hack!
Feat branch
Tag 1.0
BOLD implies long running branches
feat
master
release
hotfix
prod

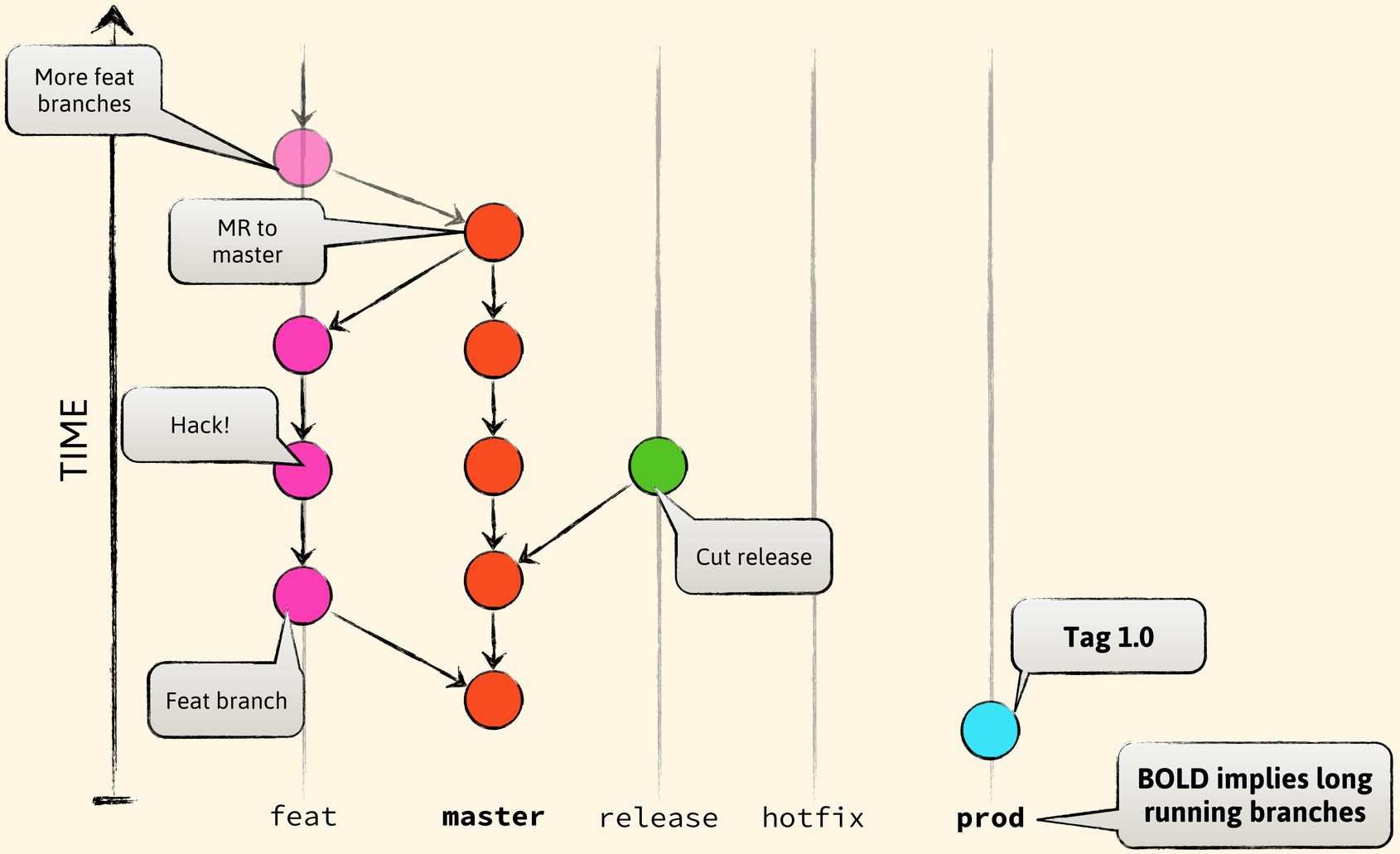
Gitflow


TIME
MR to master
Hack!
Feat branch
Tag 1.0
BOLD implies long running branches
feat
master
release
hotfix
prod

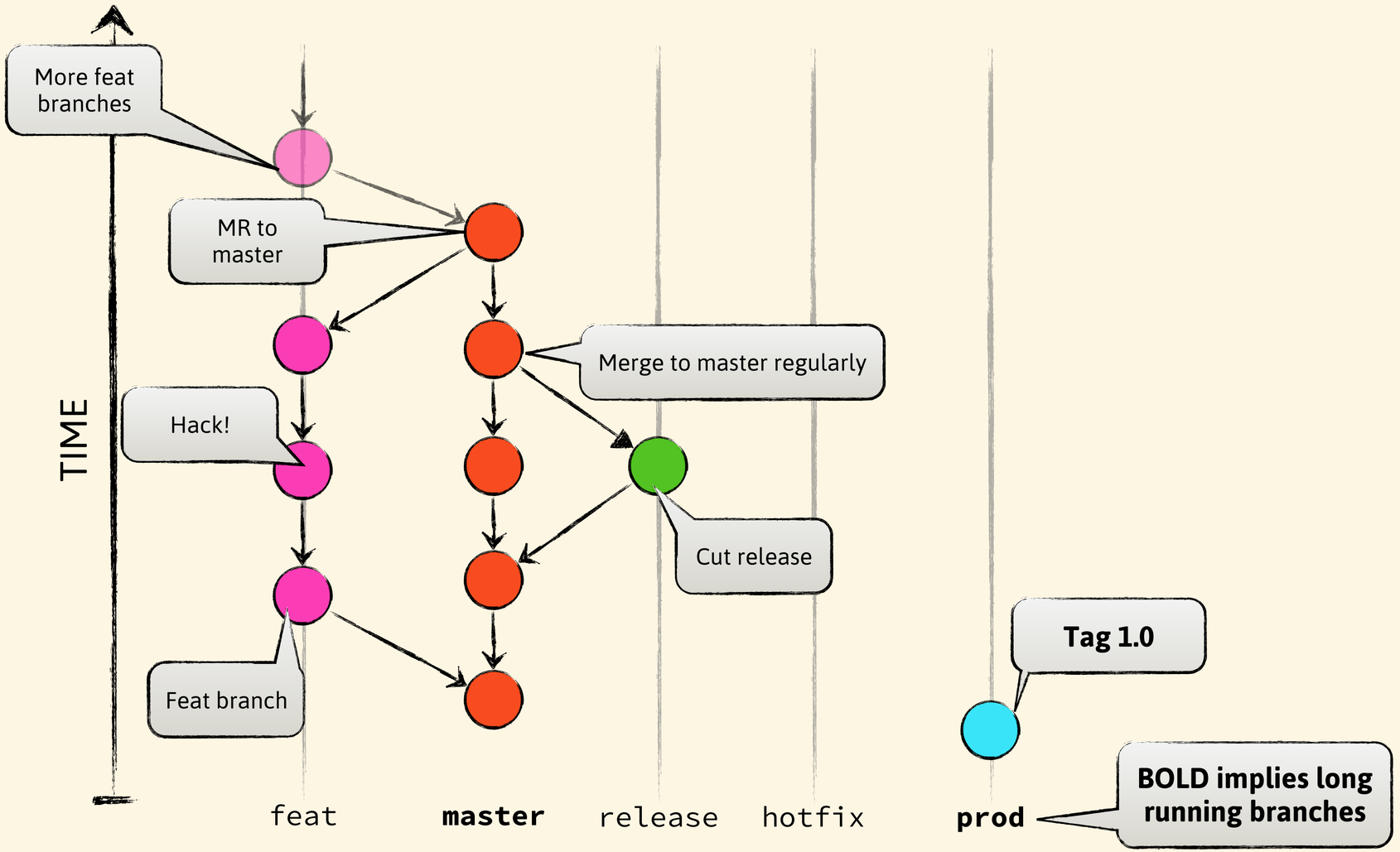
Gitflow


More feat branches
MR to master
Hack!
Feat branch
TIME
feat
master
release
hotfix
prod
Tag 1.0
BOLD implies long running branches

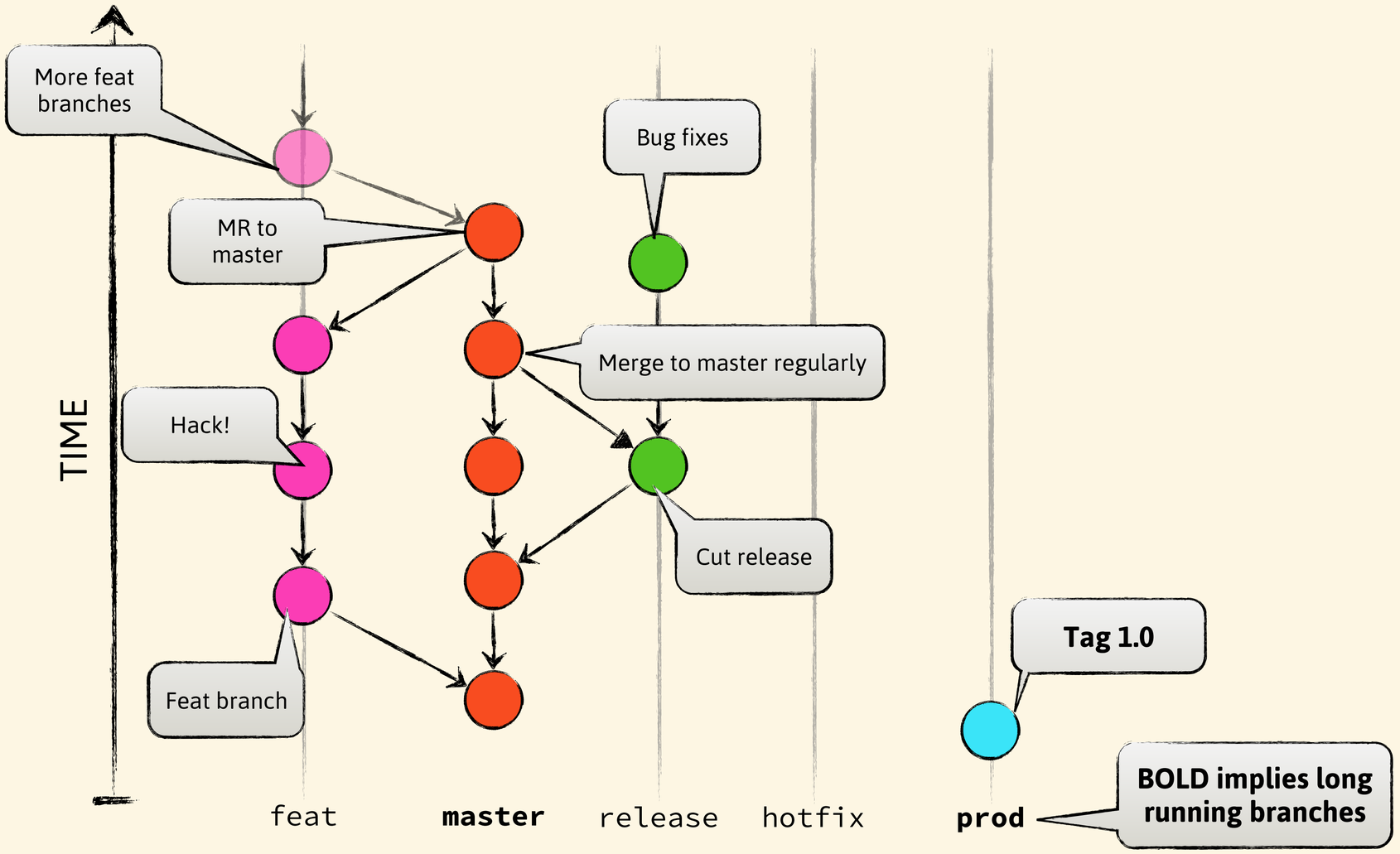
Gitflow


More feat branches
MR to master
Hack!
Feat branch
Cut release
Tag 1.0
BOLD implies long running branches
TIME
feat
master
release
hotfix
prod

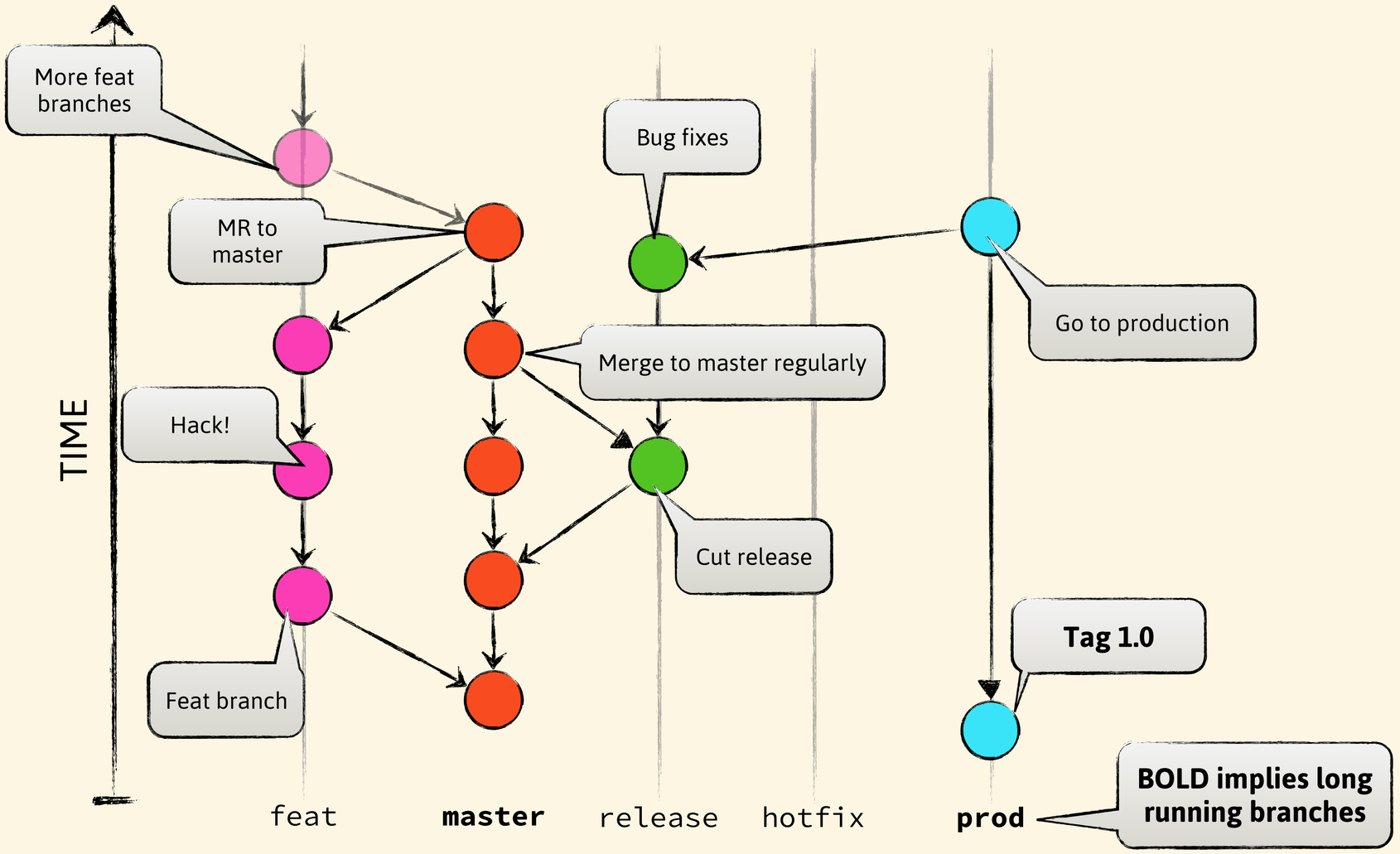
Gitflow


More feat branches
MR to master
Merge to master regularly
Hack!
Cut release
Feat branch
TIME
Tag 1.0
BOLD implies long running branches
feat
master
release
hotfix
prod

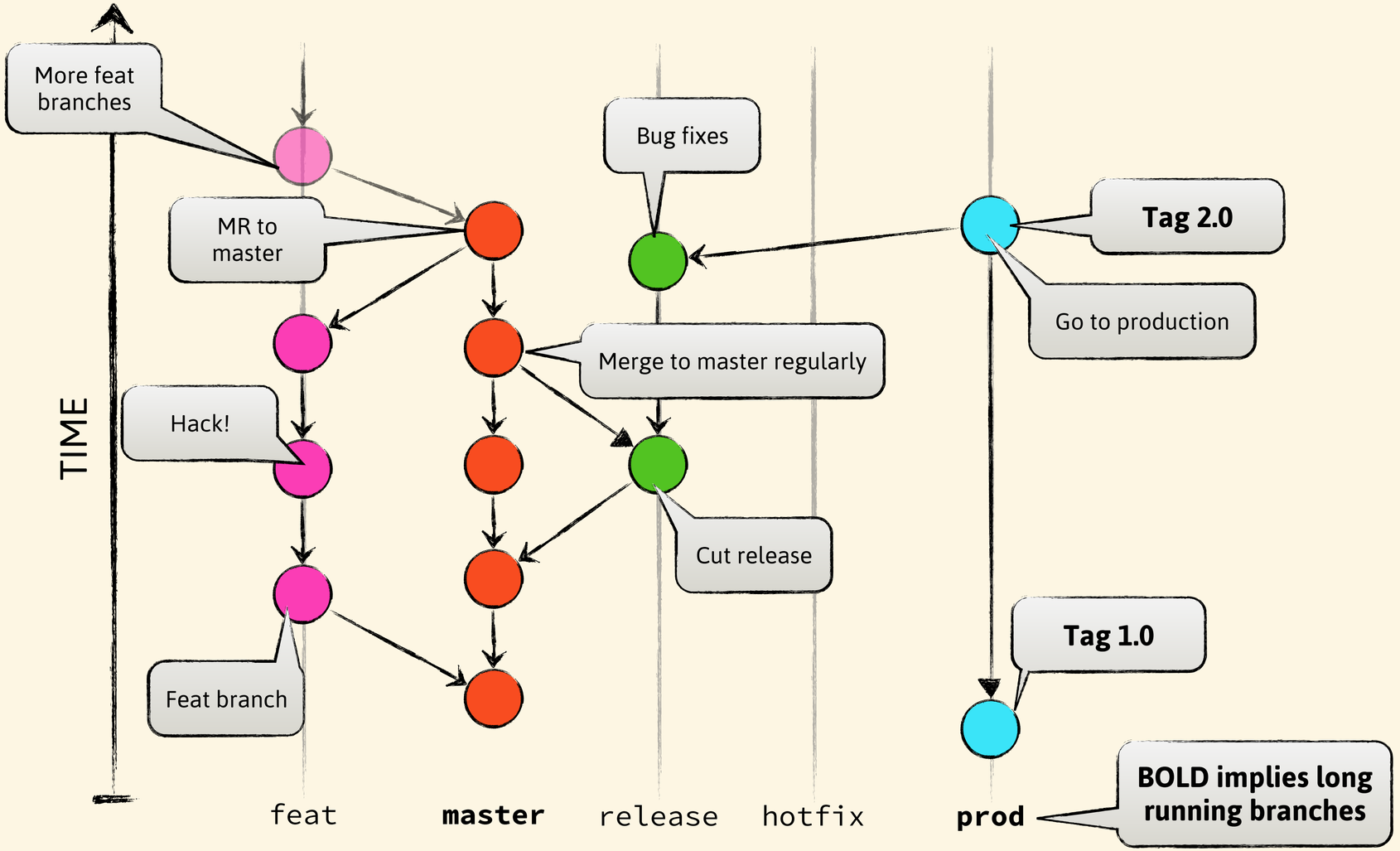
Gitflow


More feat branches
MR to master
Hack!
Feat branch
Bug fixes
Merge to master regularly
Cut release
TIME
feat
master
release
hotfix
Tag 1.0
prod
BOLD implies long running branches

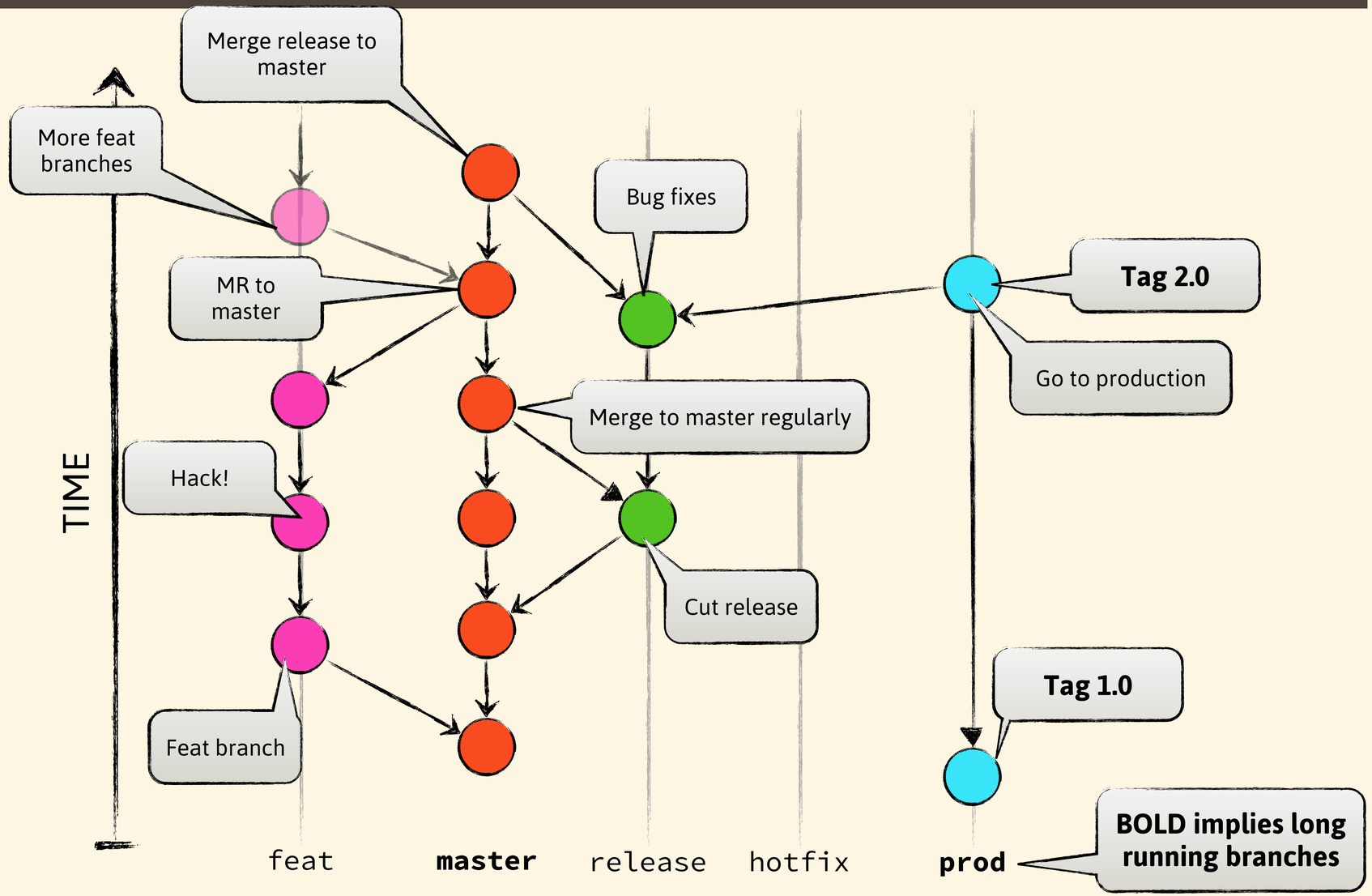
Gitflow


More feat branches
MR to master
Hack!
Feat branch
Bug fixes
Merge to master regularly
Cut release
Go to production
Tag 1.0
BOLD implies long running branches
TIME
feat
master
release
hotfix
prod

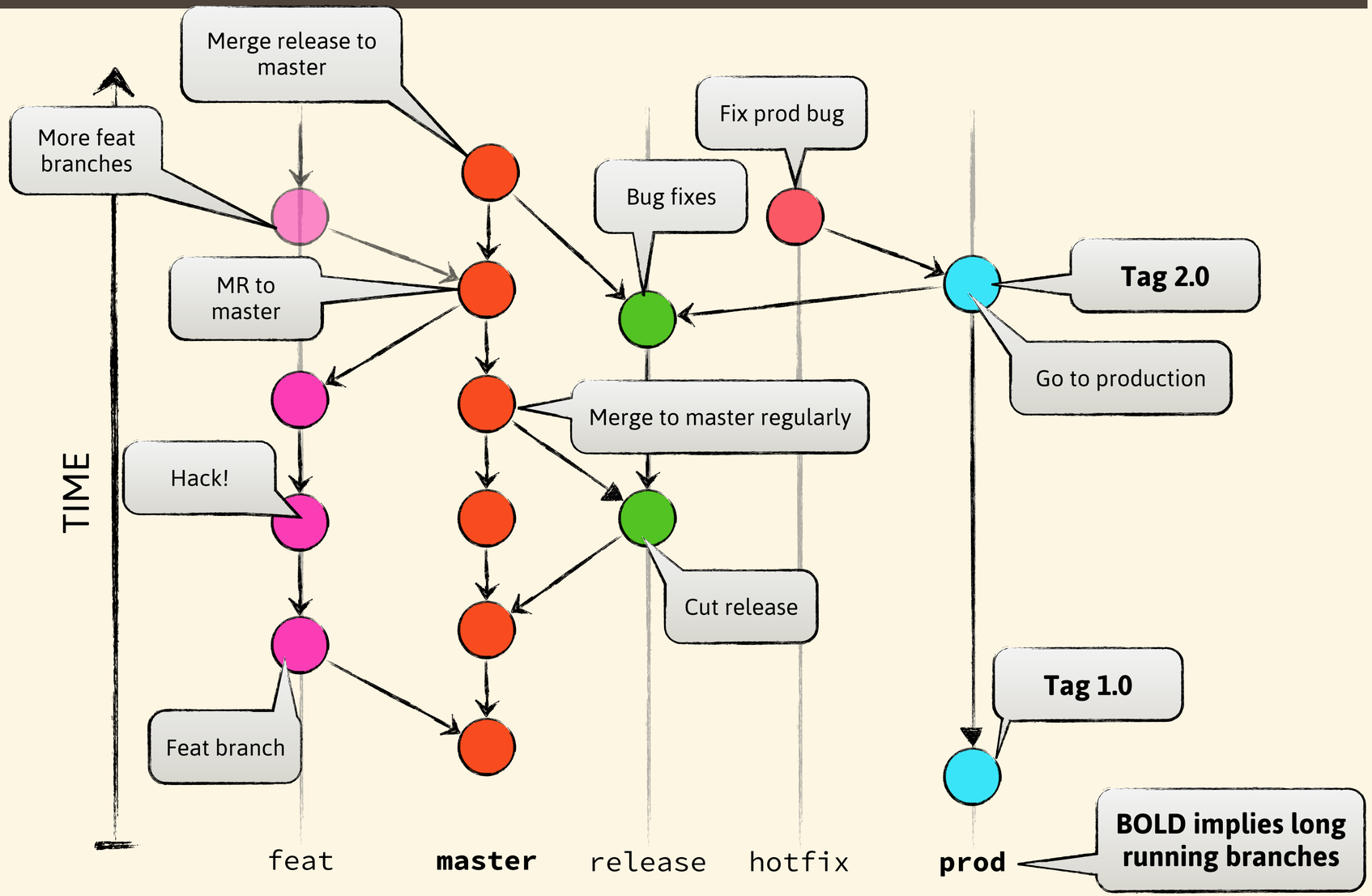
Gitflow


More feat branches
MR to master
Hack!
Feat branch
Bug fixes
Merge to master regularly
Cut release
Tag 2.0
Go to production
Tag 1.0
BOLD implies long running branches
TIME
feat
master
release
hotfix
prod

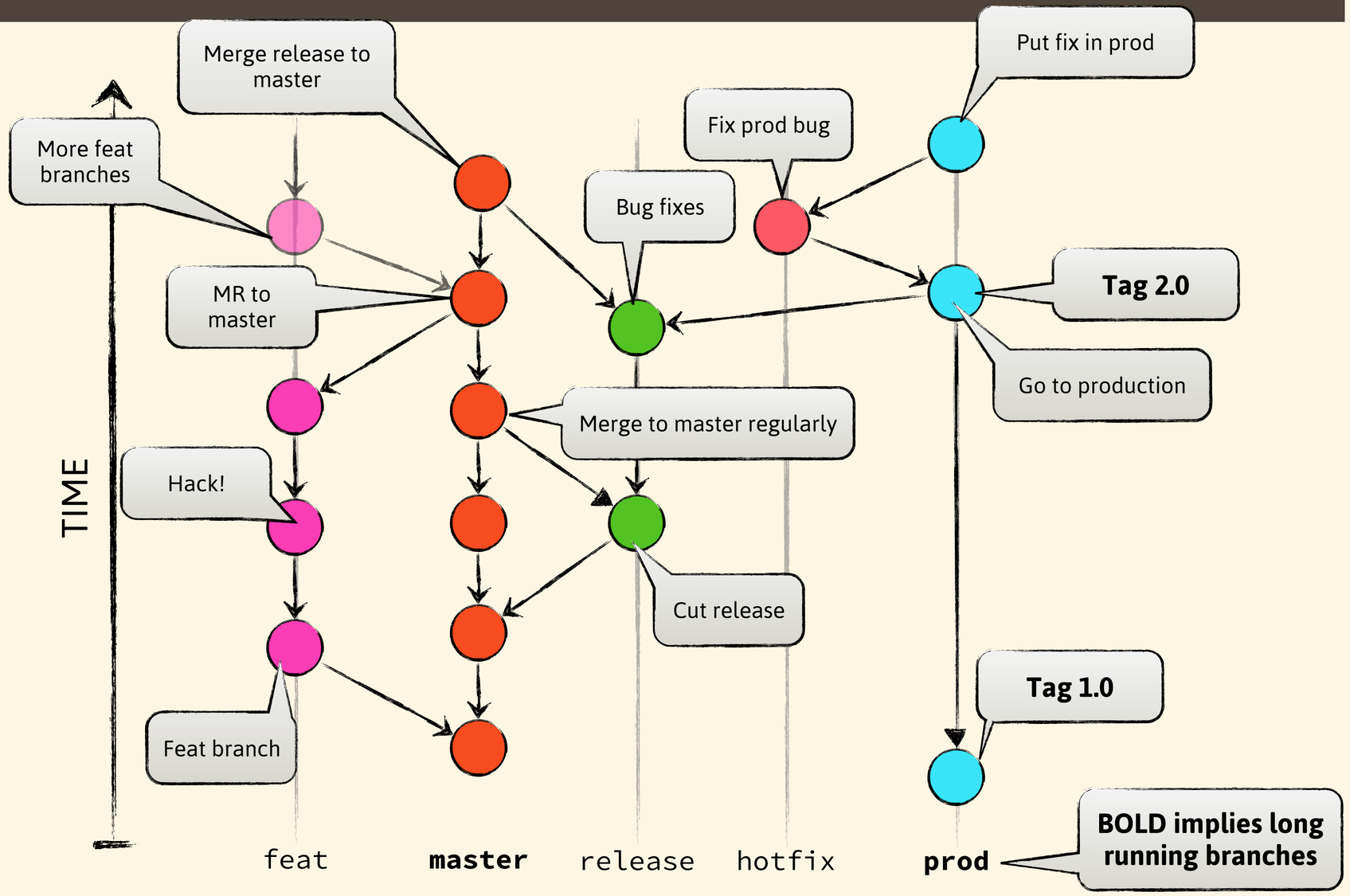
Gitflow


Merge release to master
More feat branches
Bug fixes
MR to master
Tag 2.0
Go to production
Merge to master regularly
Hack!
TIME
Cut release
Tag 1.0
Feat branch
BOLD implies long running branches
feat
master
release
hotfix
prod

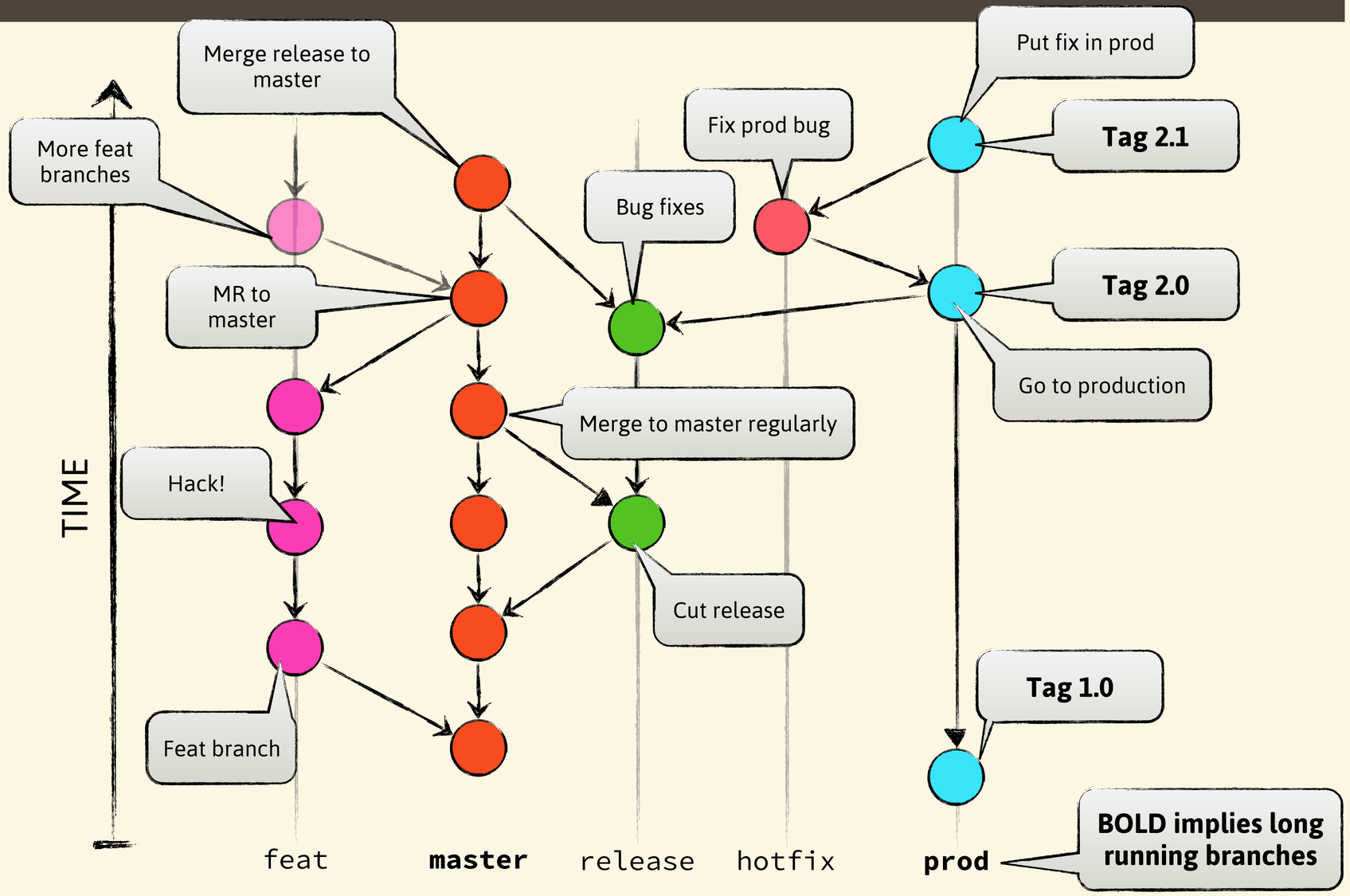
Gitflow


Merge release to master
More feat branches
Fix prod bug
Bug fixes
MR to master
Tag 2.0
Go to production
Merge to master regularly
Hack!
TIME
Cut release
Tag 1.0
Feat branch
BOLD implies long running branches
feat
master
release
hotfix
prod

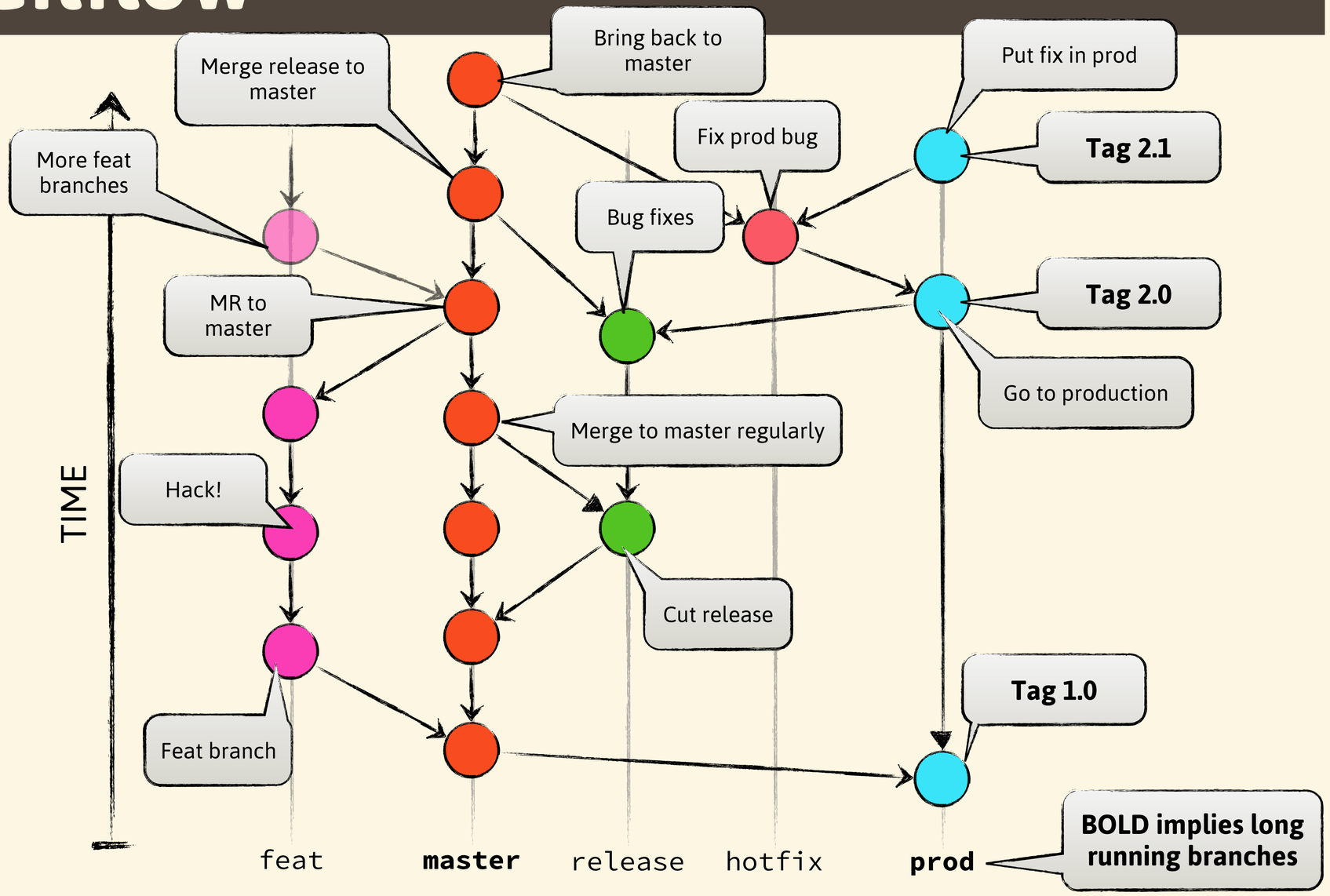
Gitflow


Merge release to master
Put fix in prod
Fix prod bug
More feat branches
Bug fixes
MR to master
Tag 2.0
Go to production
Merge to master regularly
Hack!
TIME
Cut release
Tag 1.0
Feat branch
BOLD implies long running branches
feat
master
release
hotfix
prod

Gitflow


Merge release to master
More feat branches
MR to master
Hack!
Feat branch
TIME
Fix prod bug
Bug fixes
Merge to master regularly
Cut release
Put fix in prod
Tag 2.1
Tag 2.0
Go to production
Tag 1.0
feat
master
release
hotfix
prod
BOLD implies long running branches

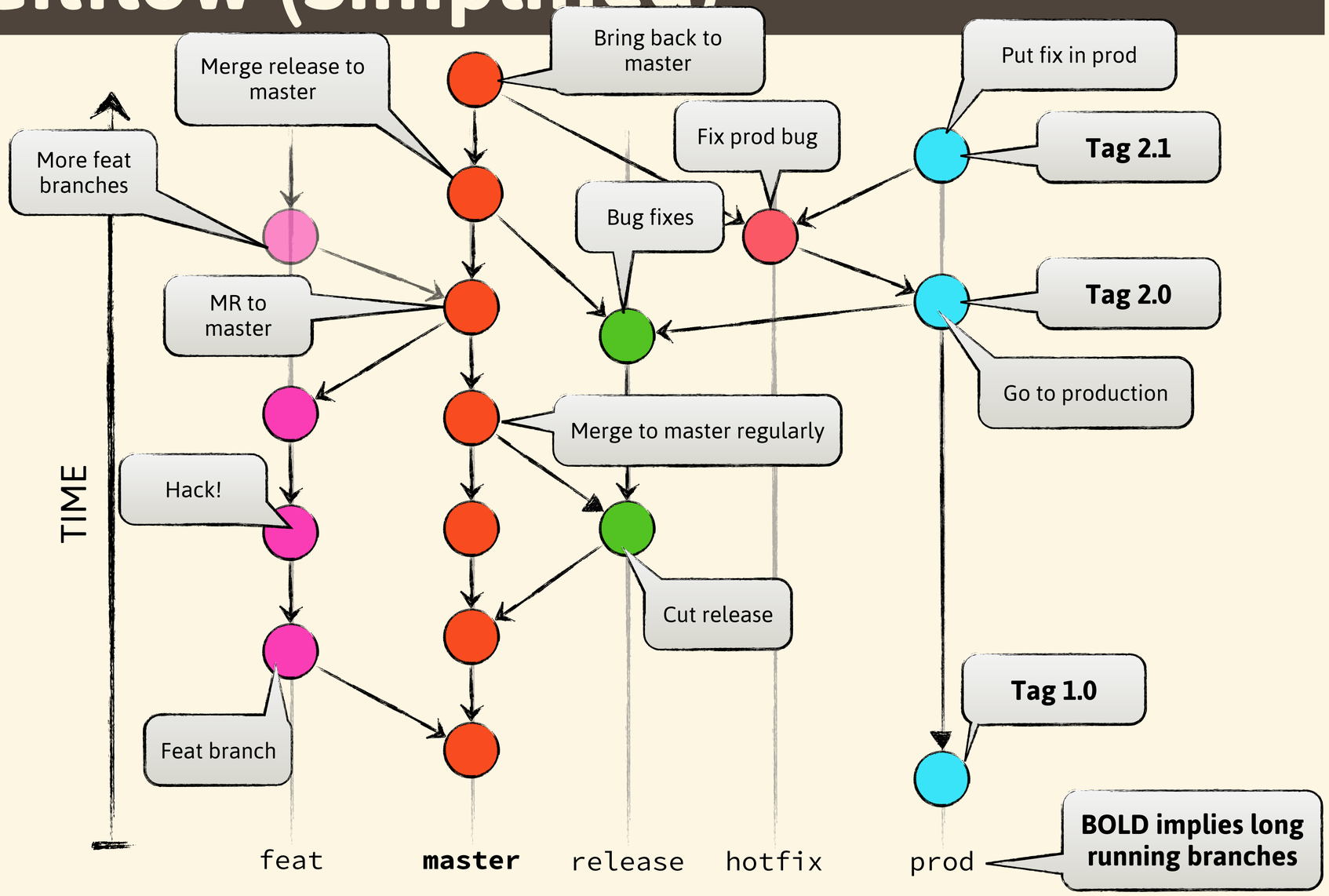
Gitflow


Merge release to master
Bring back to master
Put fix in prod
More feat branches
Fix prod bug
Tag 2.1
Bug fixes
MR to master
Tag 2.0
Go to production
Merge to master regularly
Hack!
TIME
Cut release
Tag 1.0
Feat branch
BOLD implies long running branches
feat
master
release
hotfix
prod

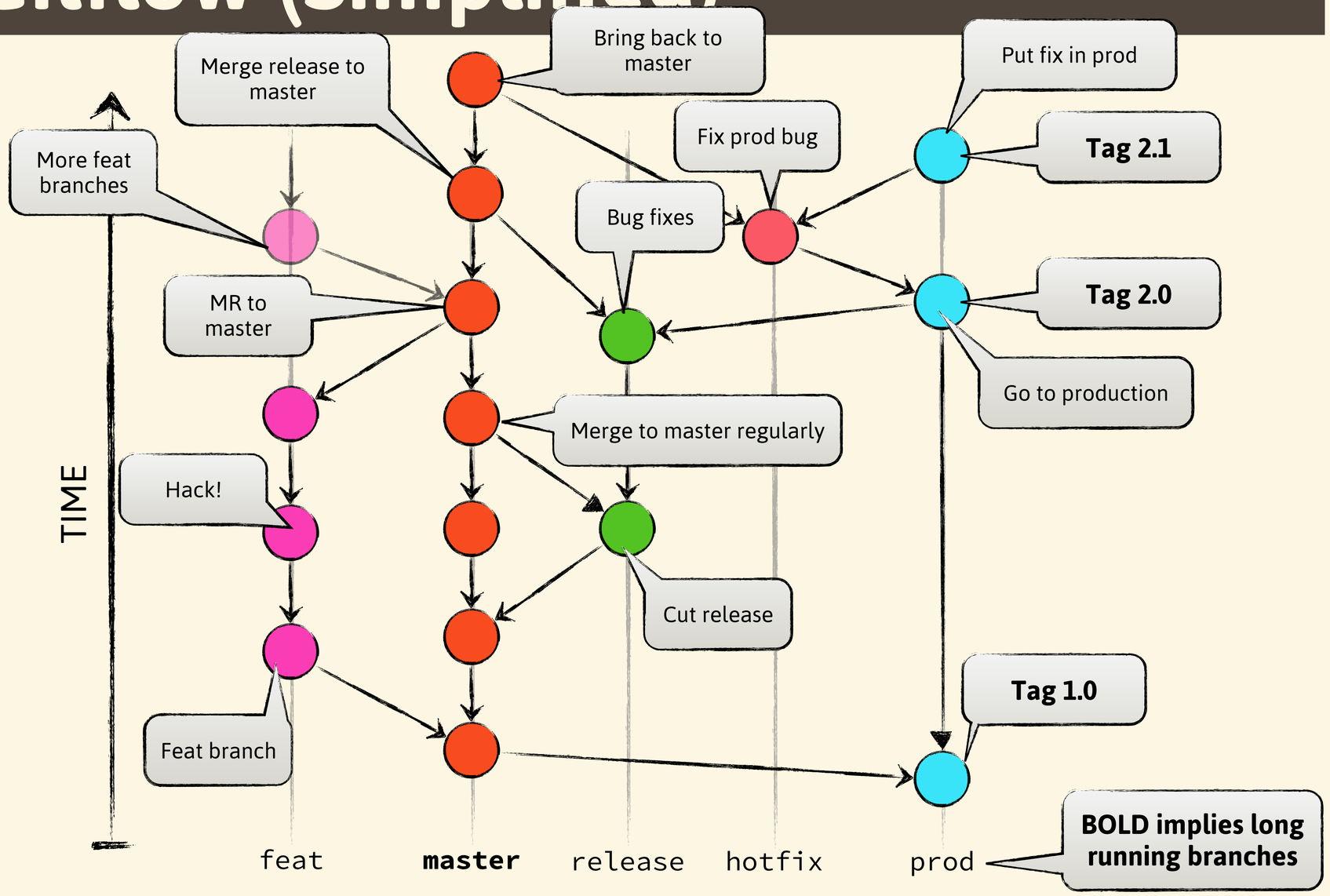
Gitflow

- Pros
  - Adapts well to large teams
  - Allows for a traditional "promote to production" model
  - Third party tooling available
- Cons
  - Encourages multiple long running branches
  - *May* encourage "big-bang" releases

Gitflow (Simplified)



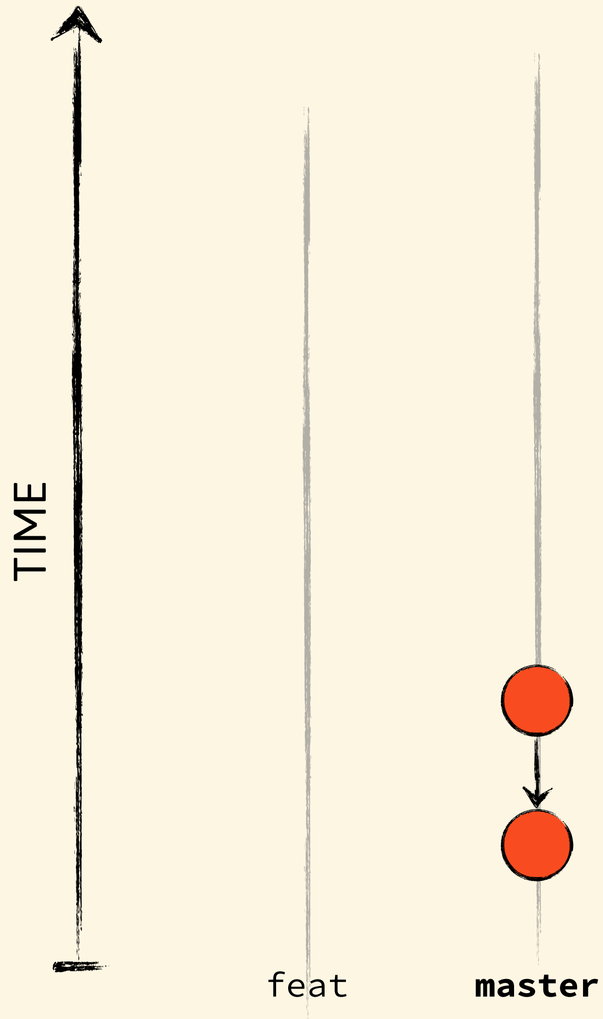
Gitflow (Simplified)



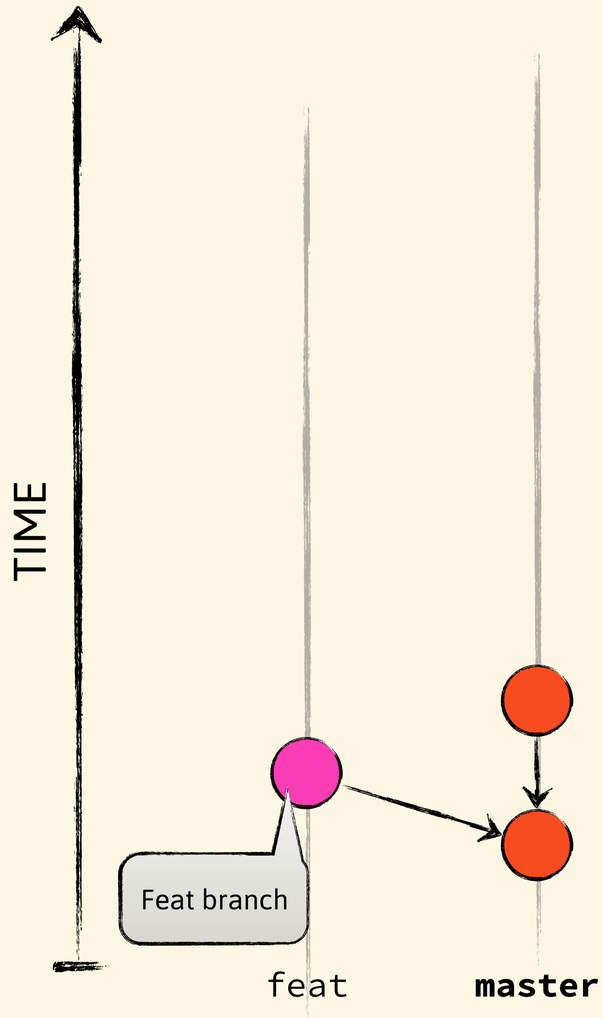
Gitflow (Simplified)

- Pros
  - Same as traditional Gitflow
  - Only **one** long running branch
- Cons
  - Same as Gitflow

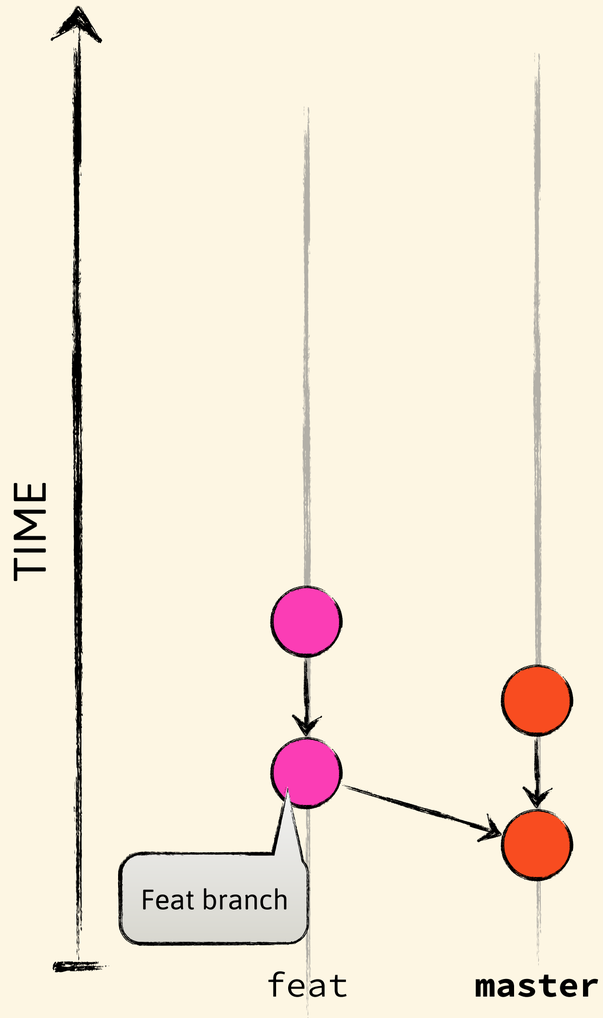
Trunk-based (CD model)



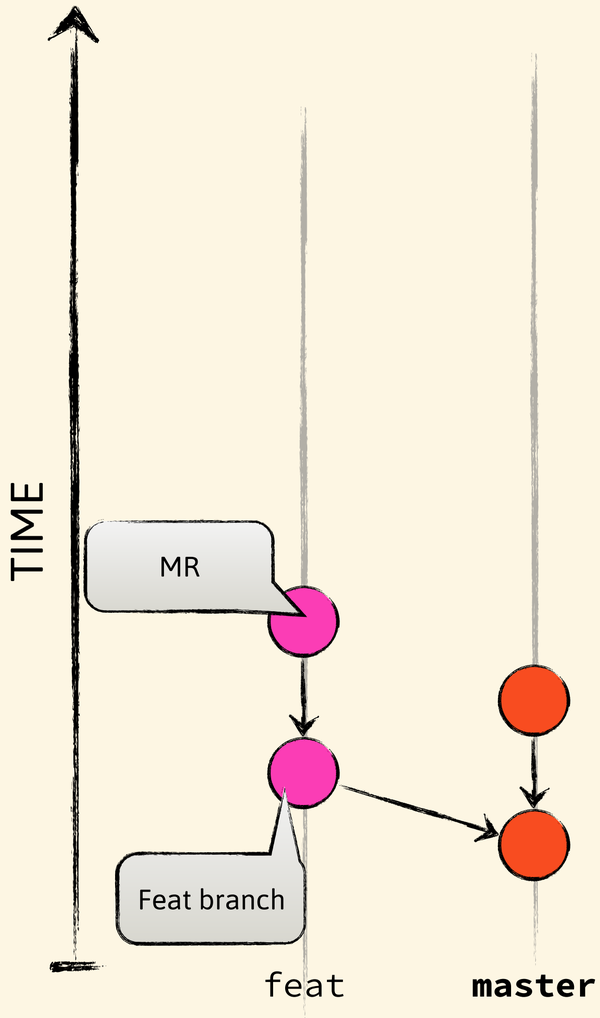
Trunk-based (CD model)



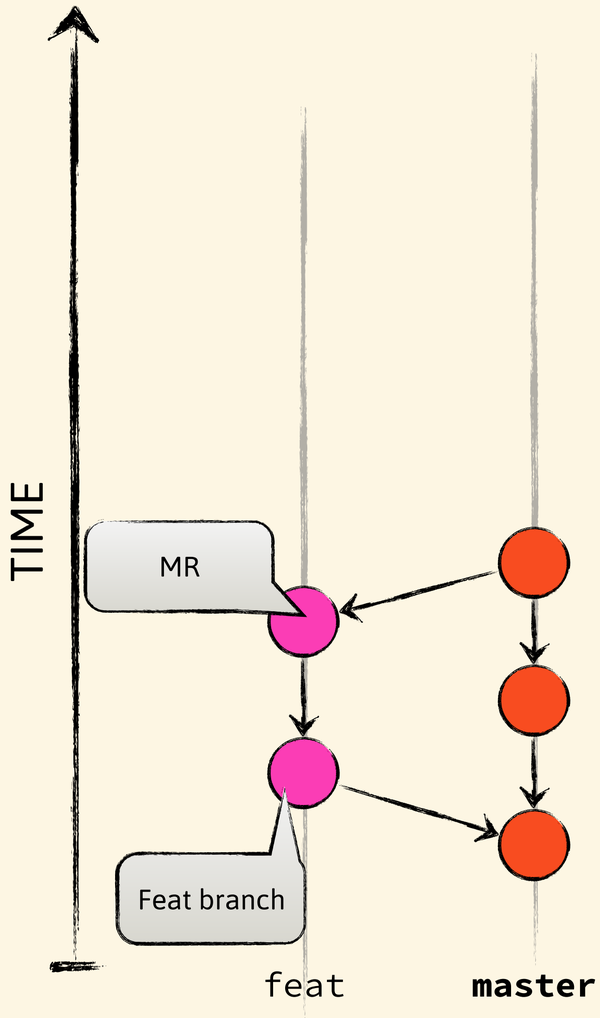
Trunk-based (CD model)



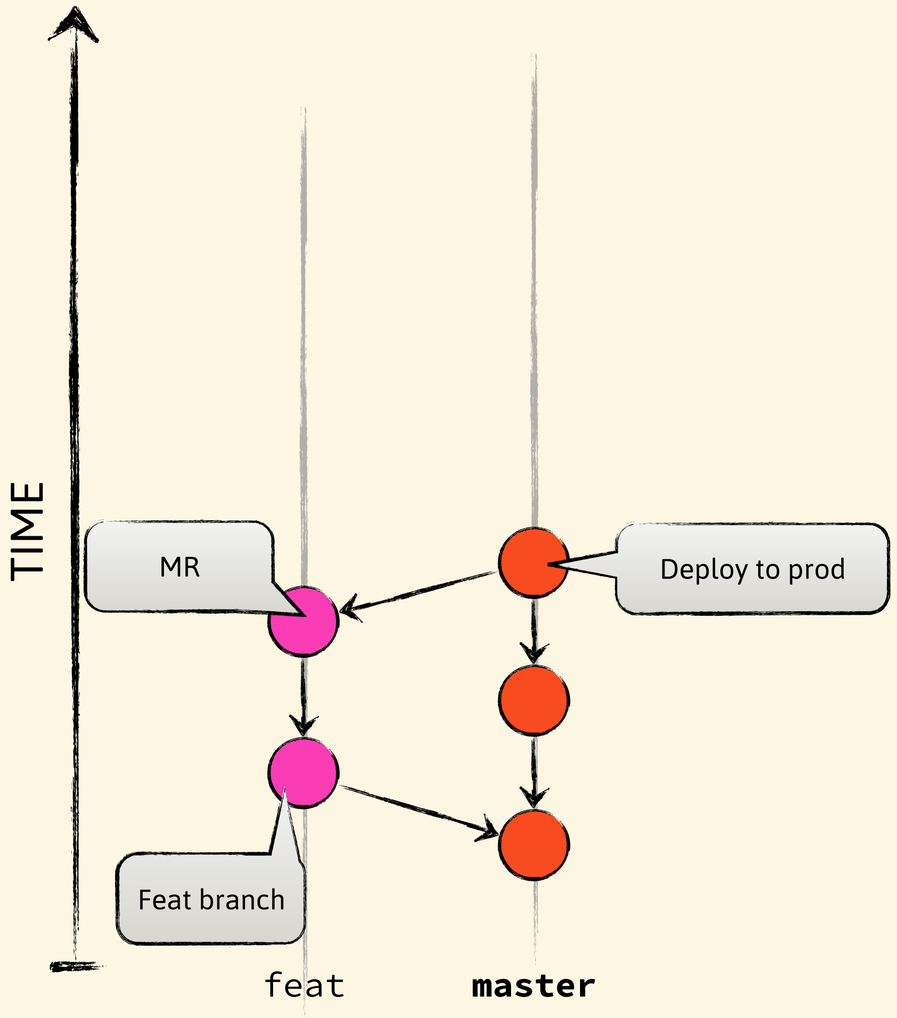
Trunk-based (CD model)



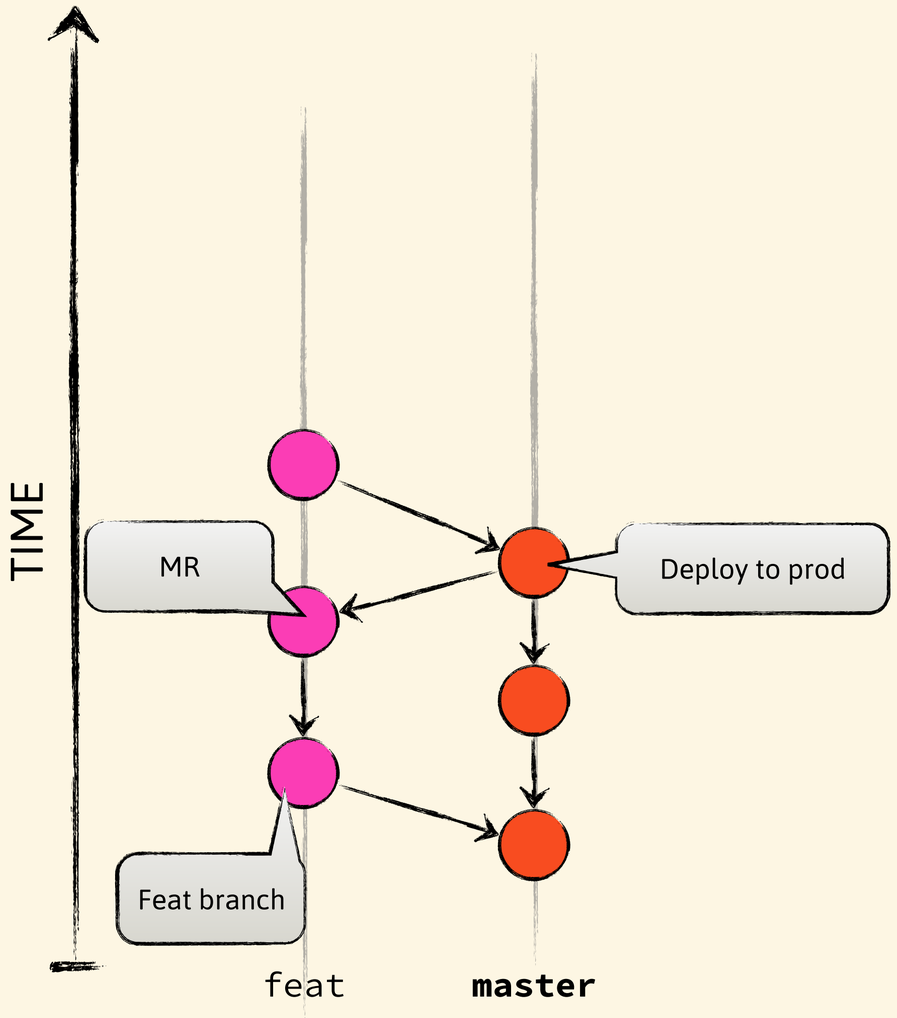
Trunk-based (CD model)



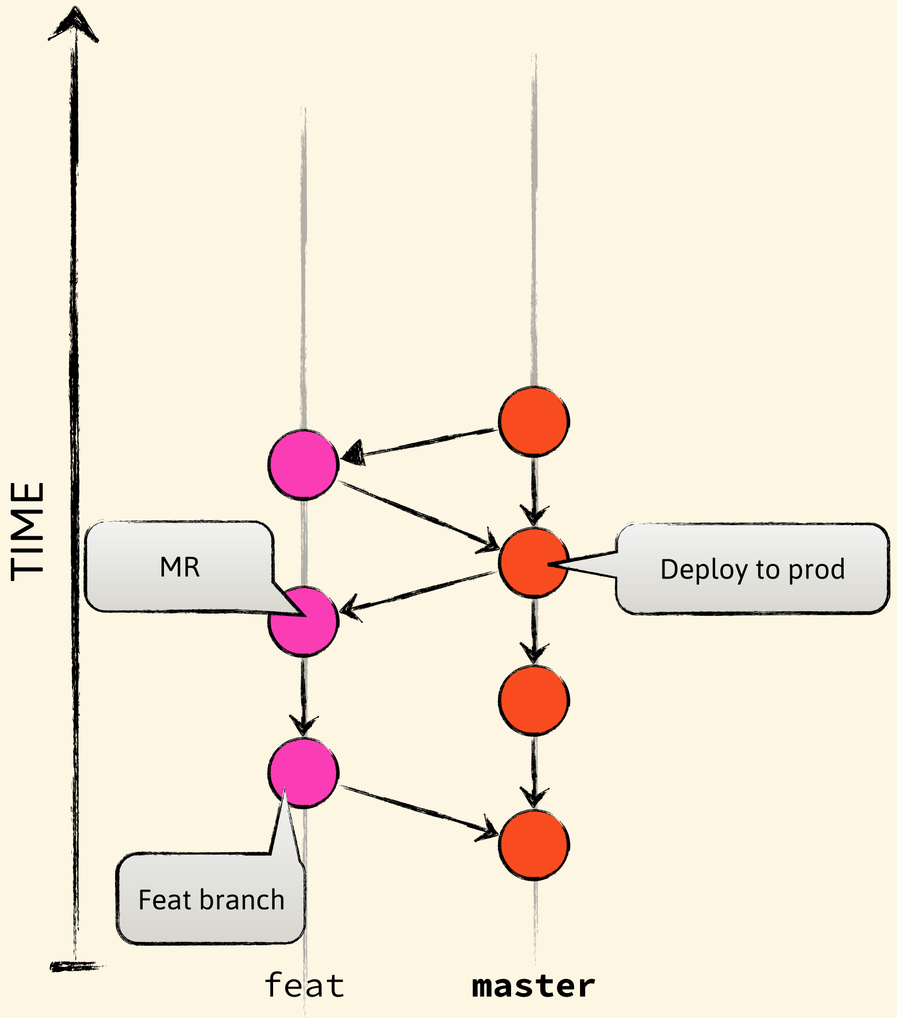
Trunk-based (CD model)



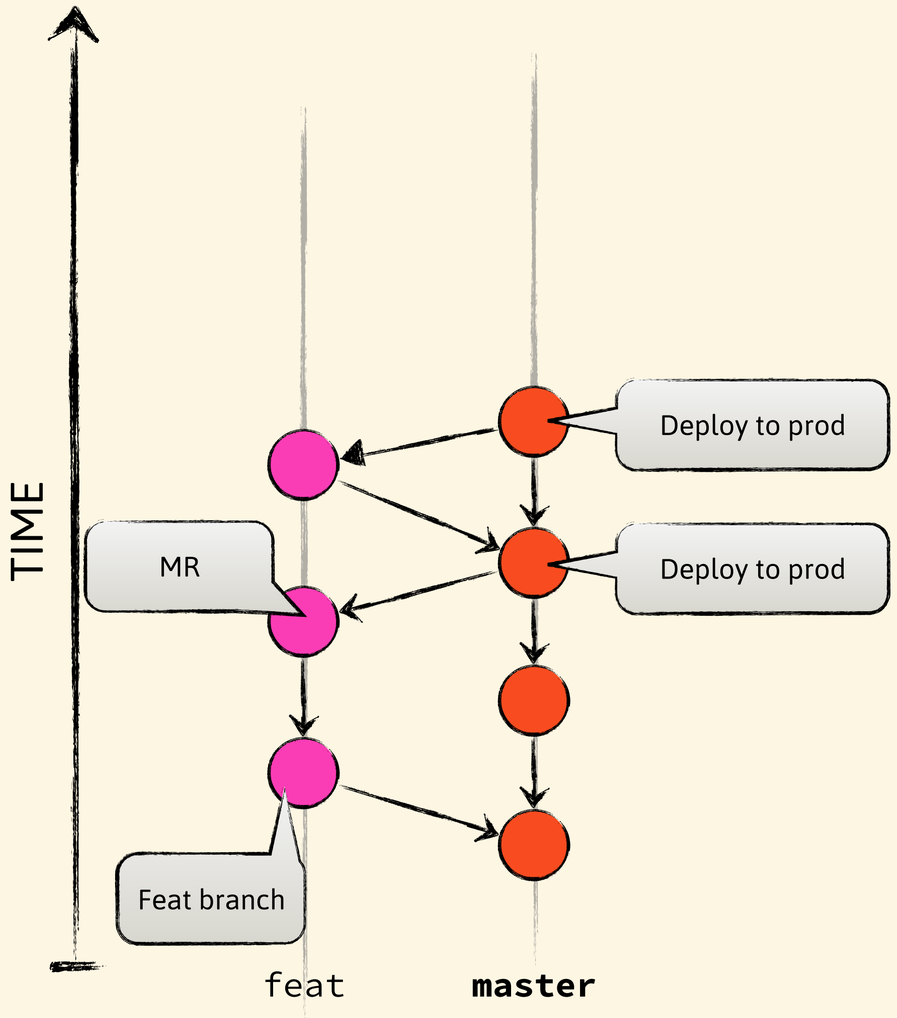
Trunk-based (CD model)



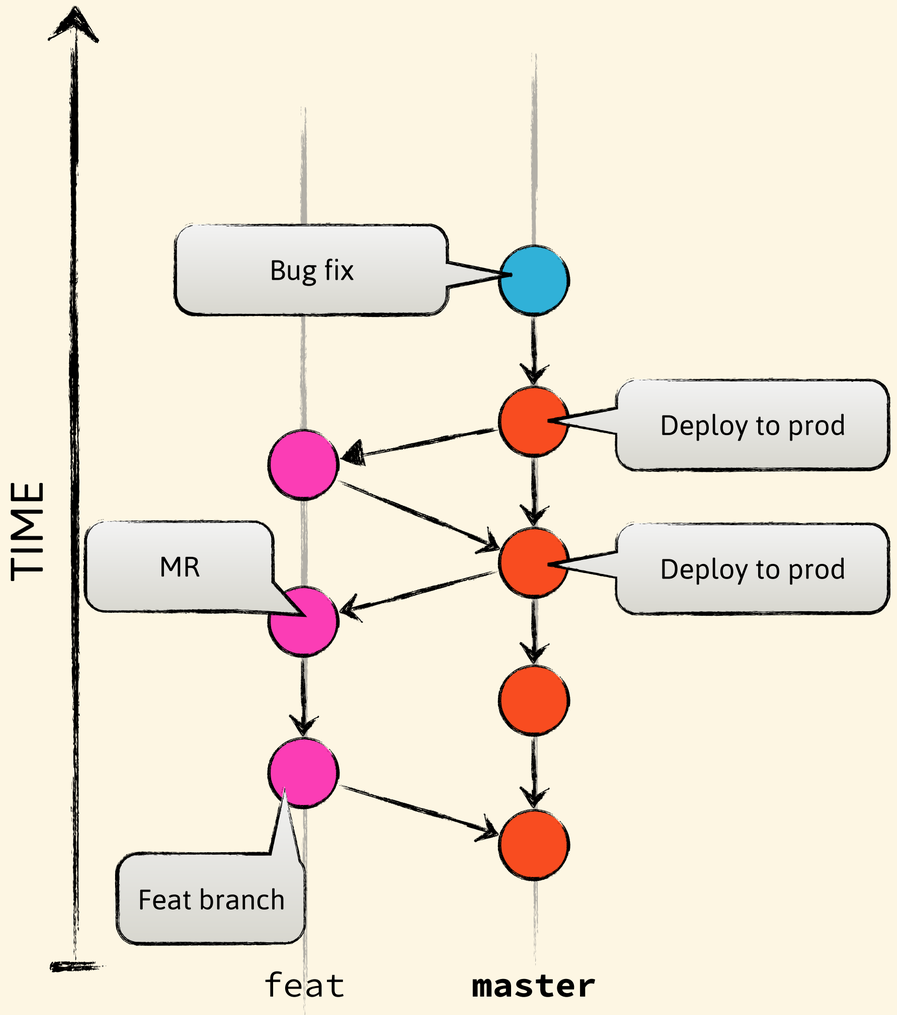
Trunk-based (CD model)



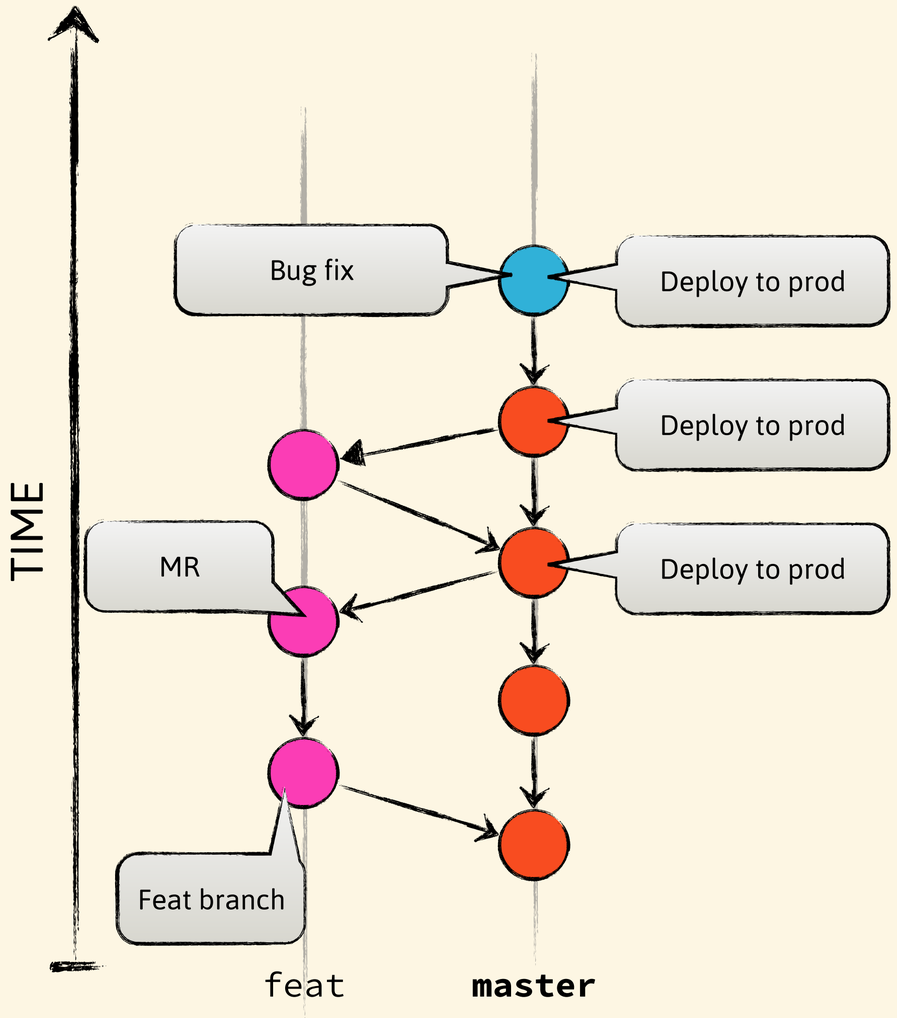
Trunk-based (CD model)



Trunk-based (CD model)



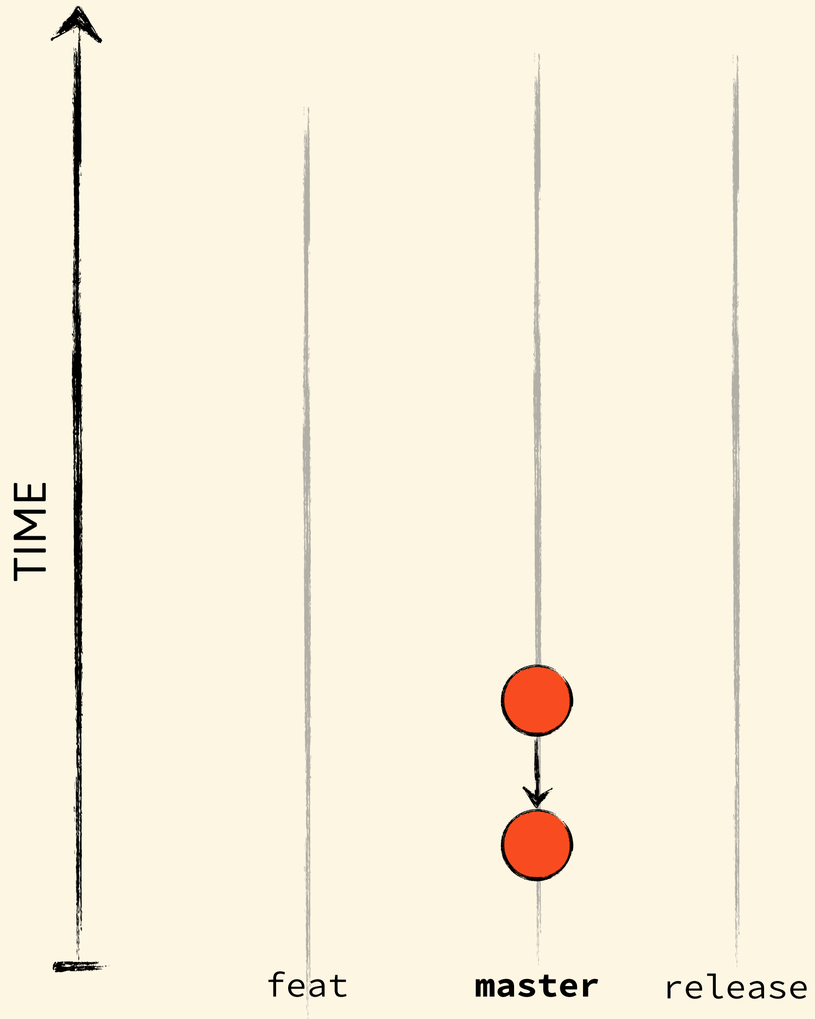
Trunk-based (CD model)



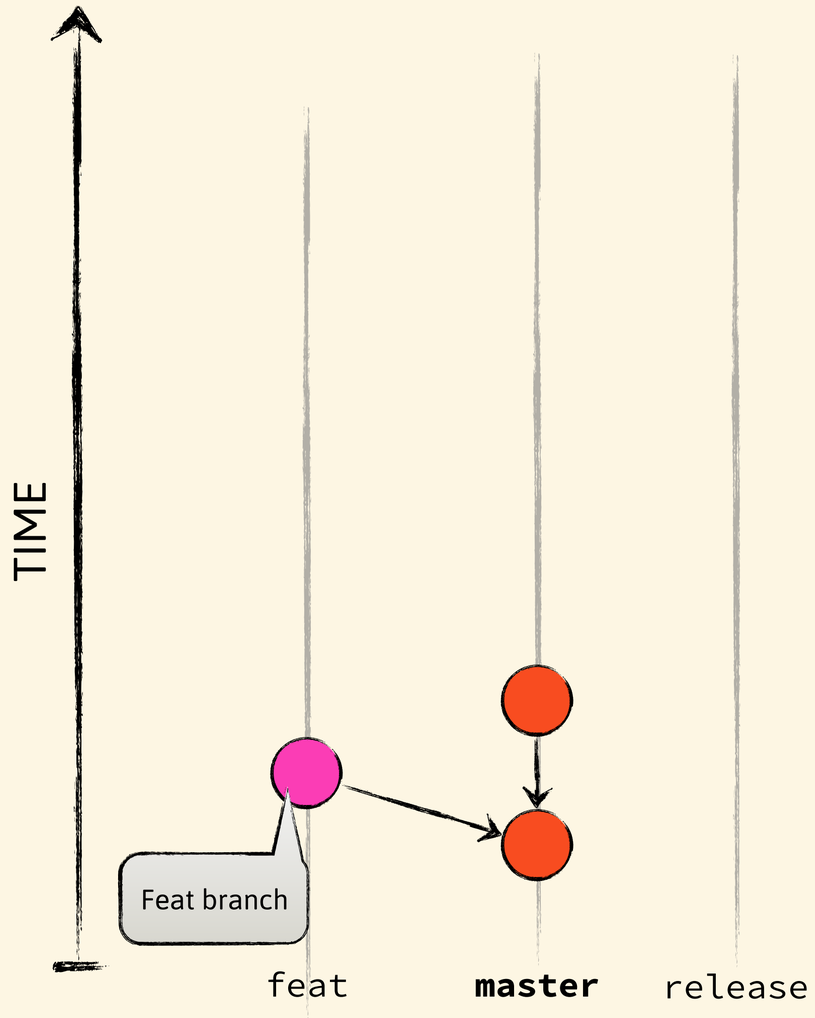
Trunk-based (CD model)

- Pros
  - Extremely simple and lightweight model
  - Allows for very small lead times
  - Lends very well to Continuous Delivery (and if possible, Continuous Deployments)
- Cons
  - Simple does not imply easy — Necessitates concepts like "branching by abstraction" and feature-toggles
- See Trunk Based Development and Branch by Abstraction

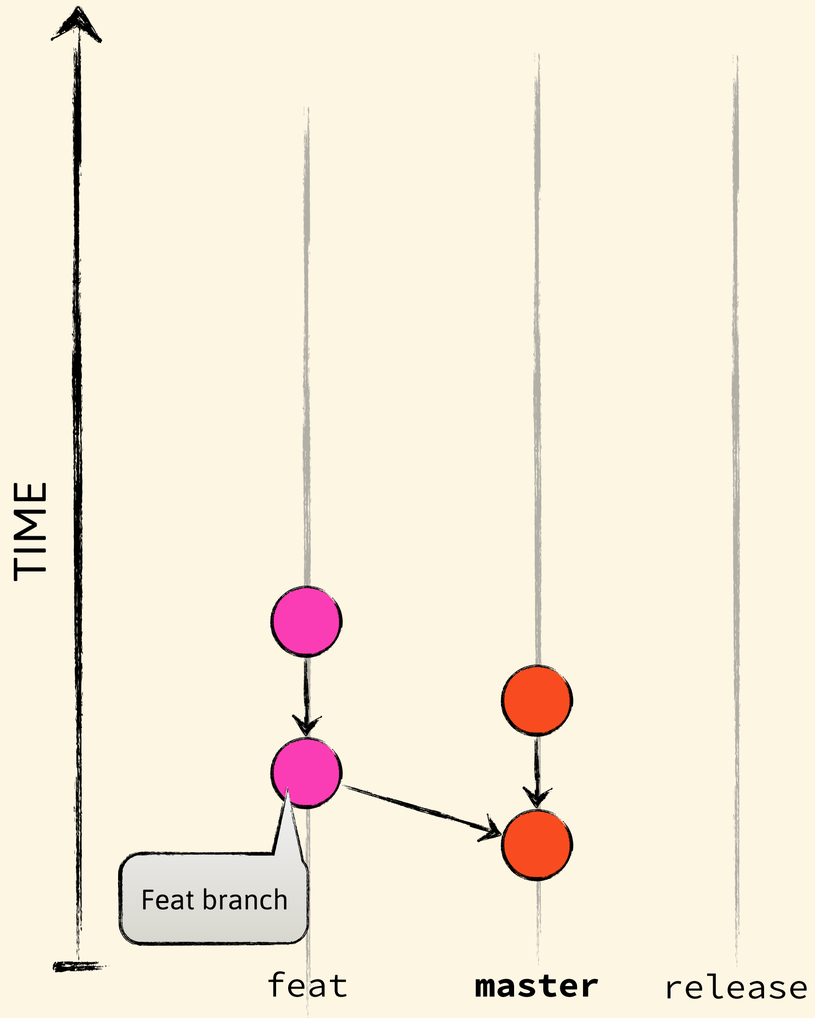
Trunk-based (Branch for release)



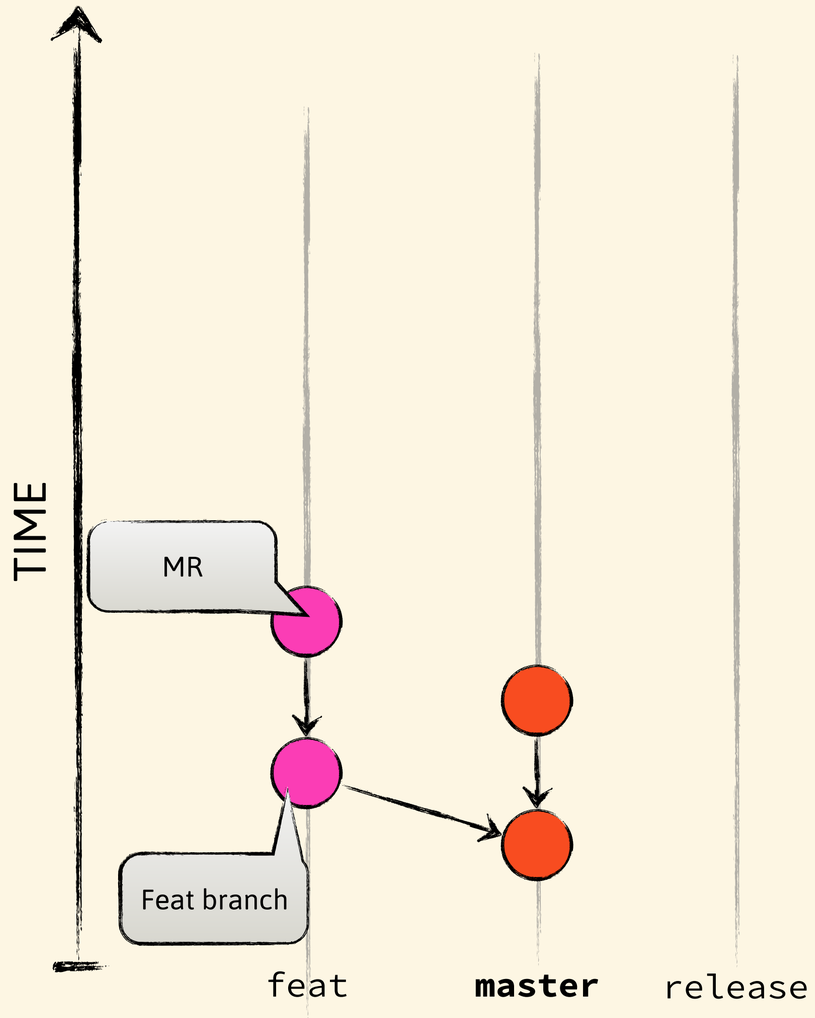
Trunk-based (Branch for release)



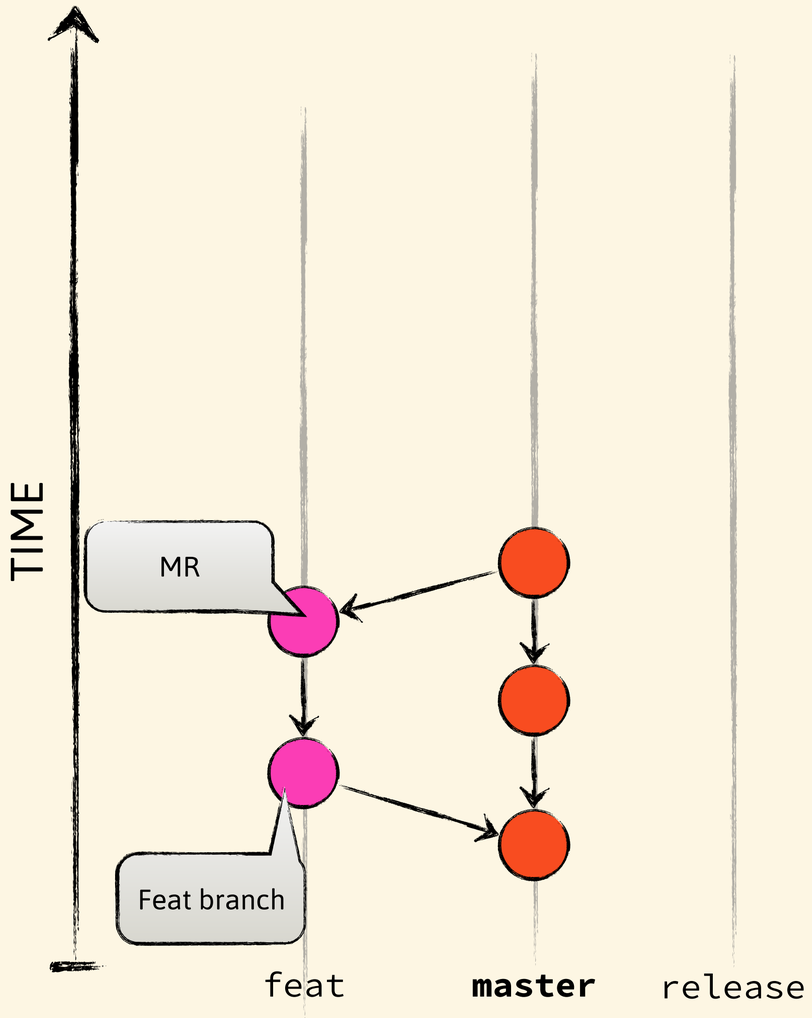
Trunk-based (Branch for release)



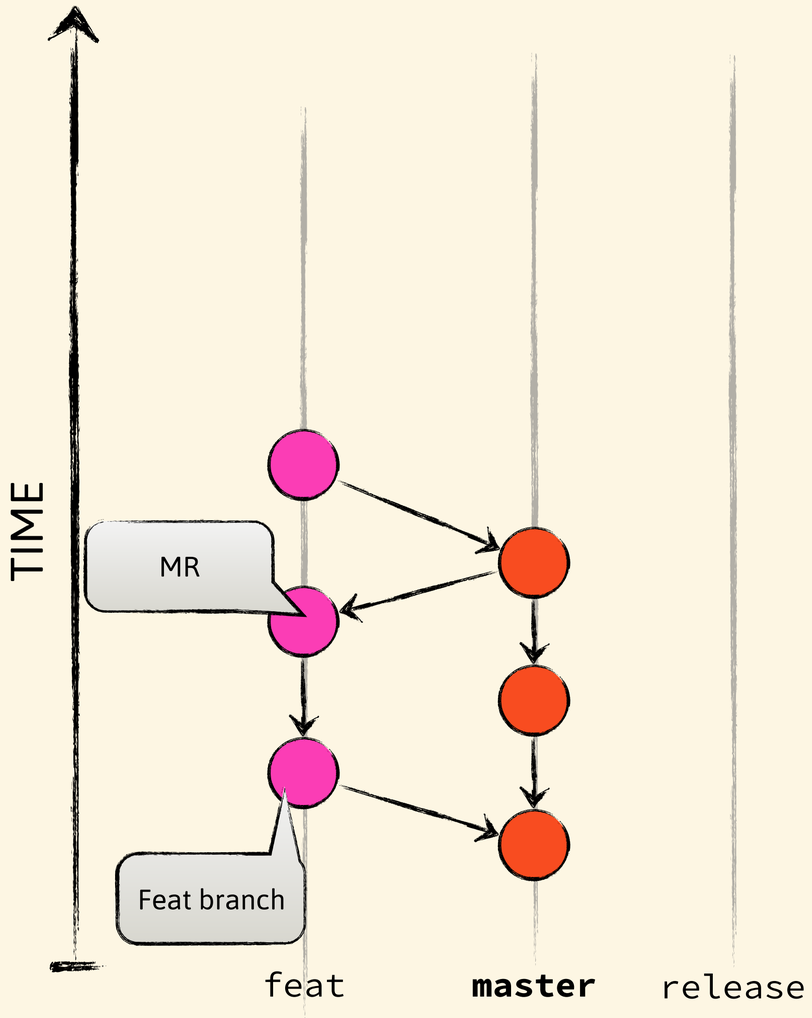
Trunk-based (Branch for release)



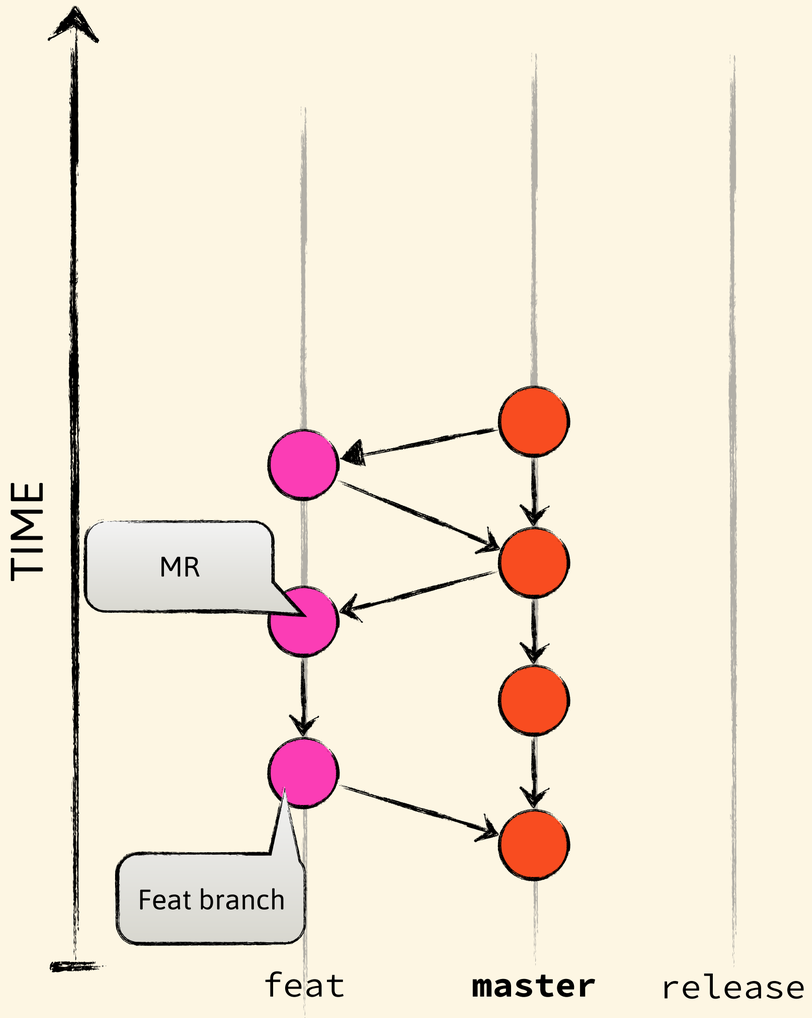
Trunk-based (Branch for release)



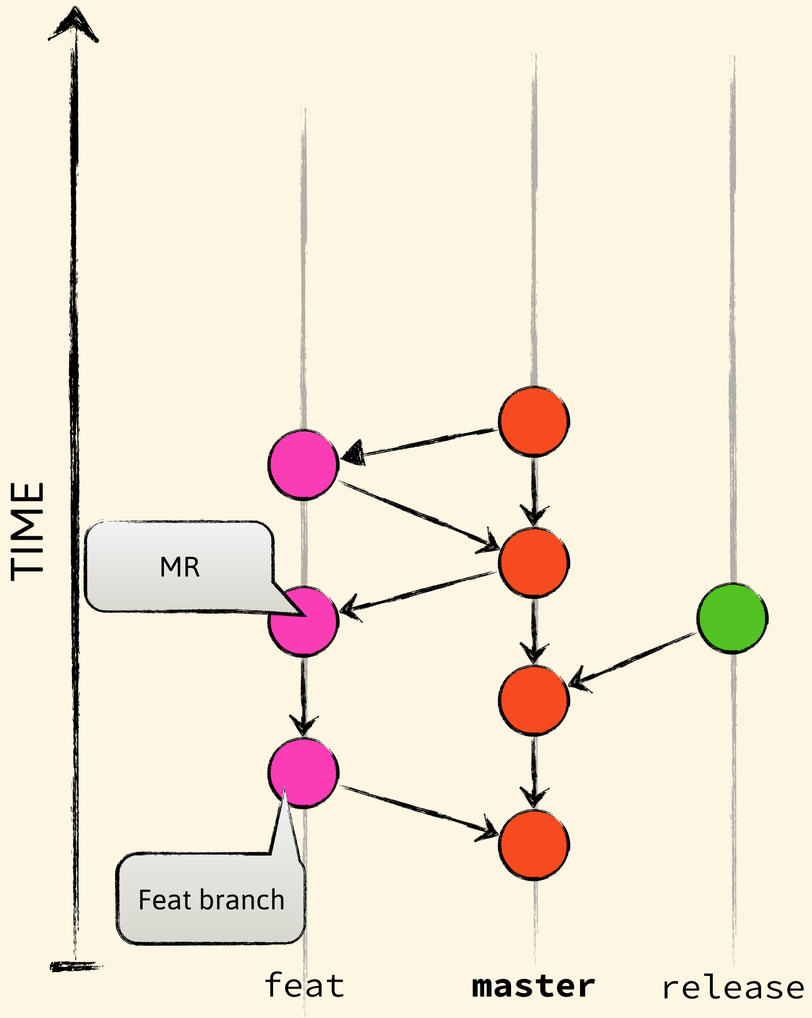
Trunk-based (Branch for release)



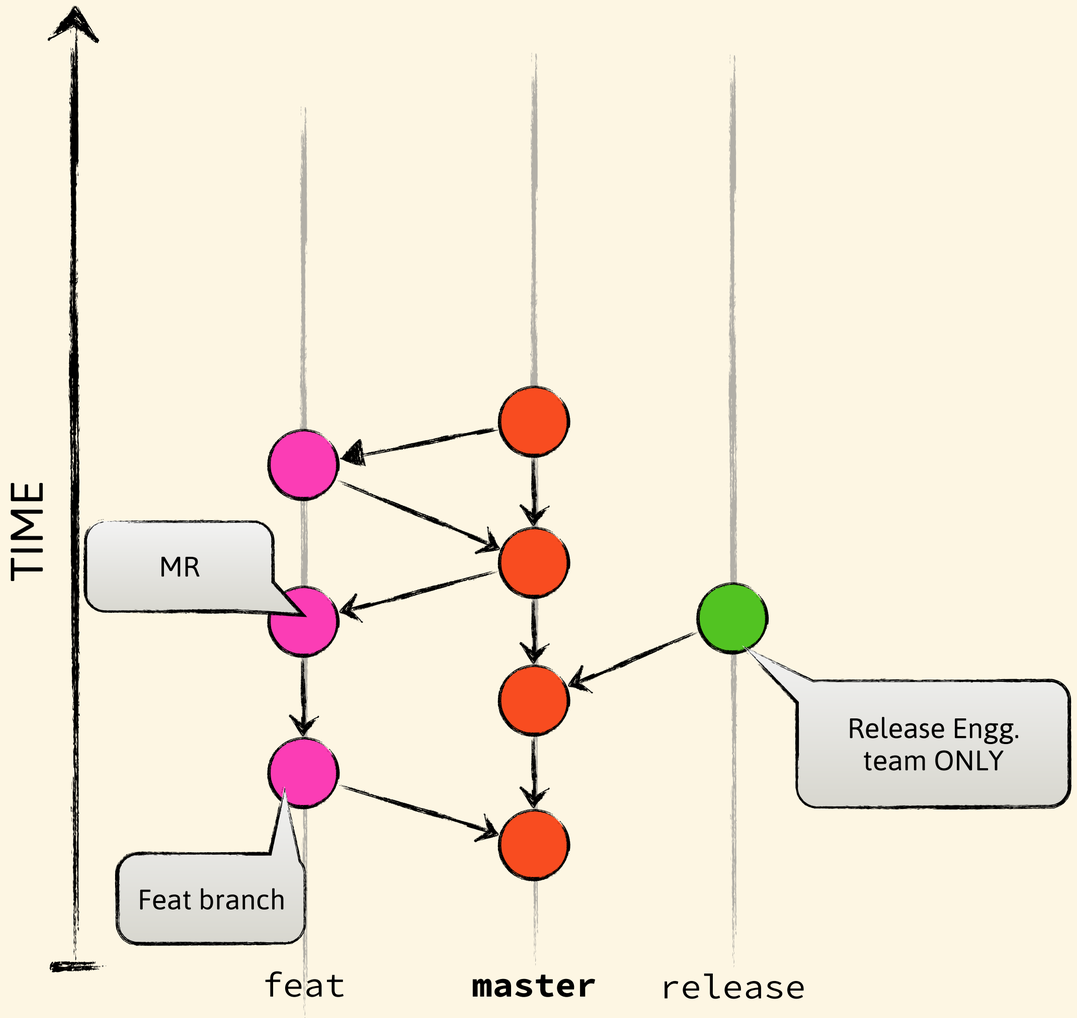
Trunk-based (Branch for release)



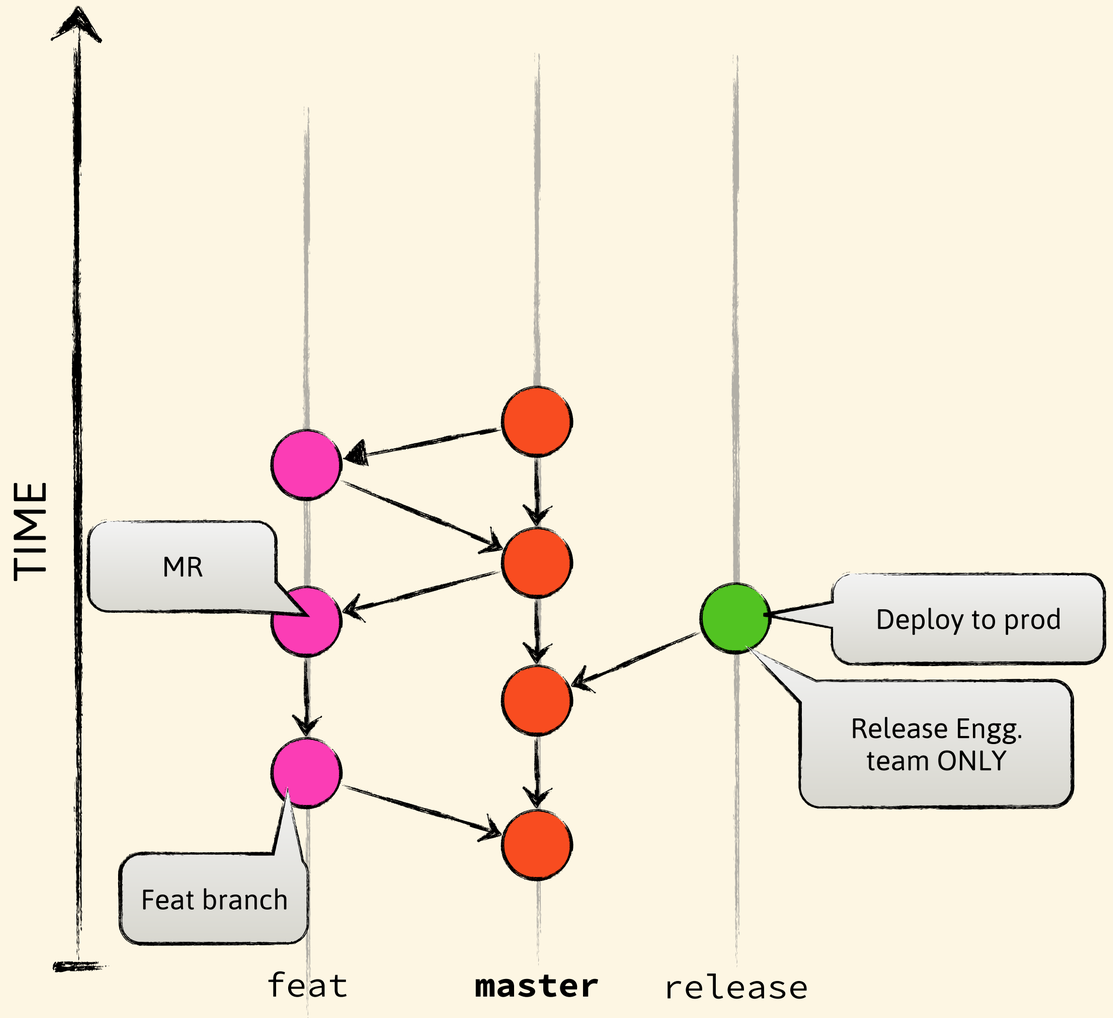
Trunk-based (Branch for release)



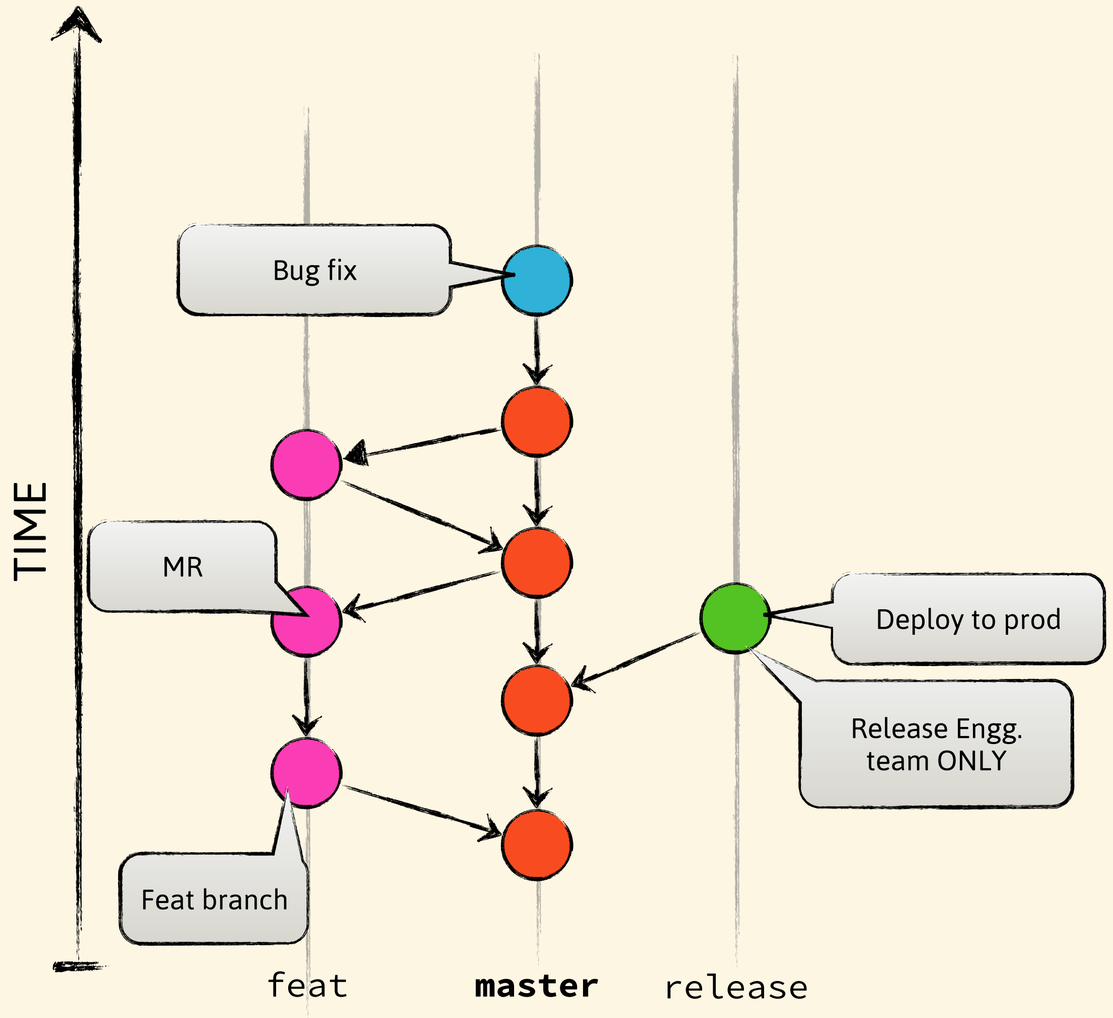
Trunk-based (Branch for release)



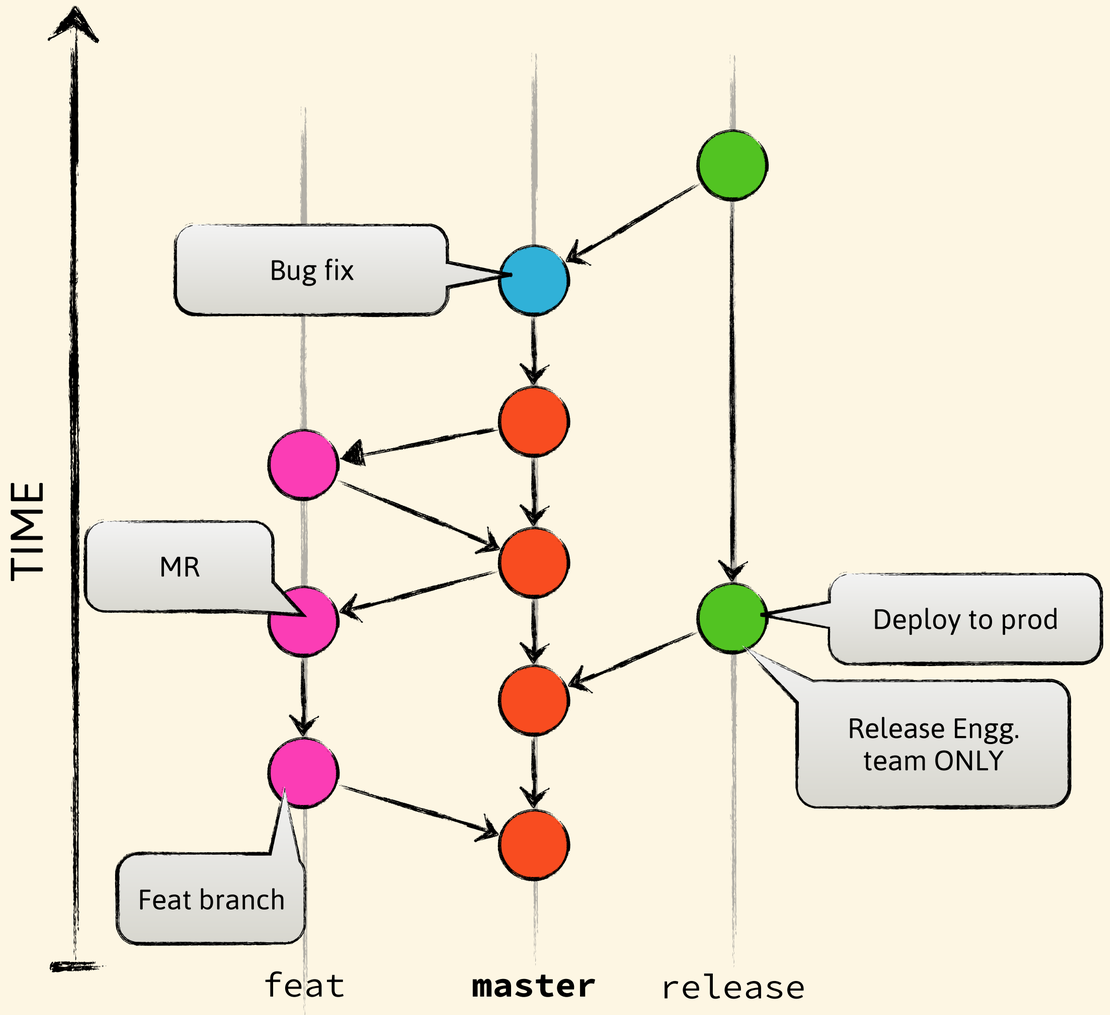
Trunk-based (Branch for release)



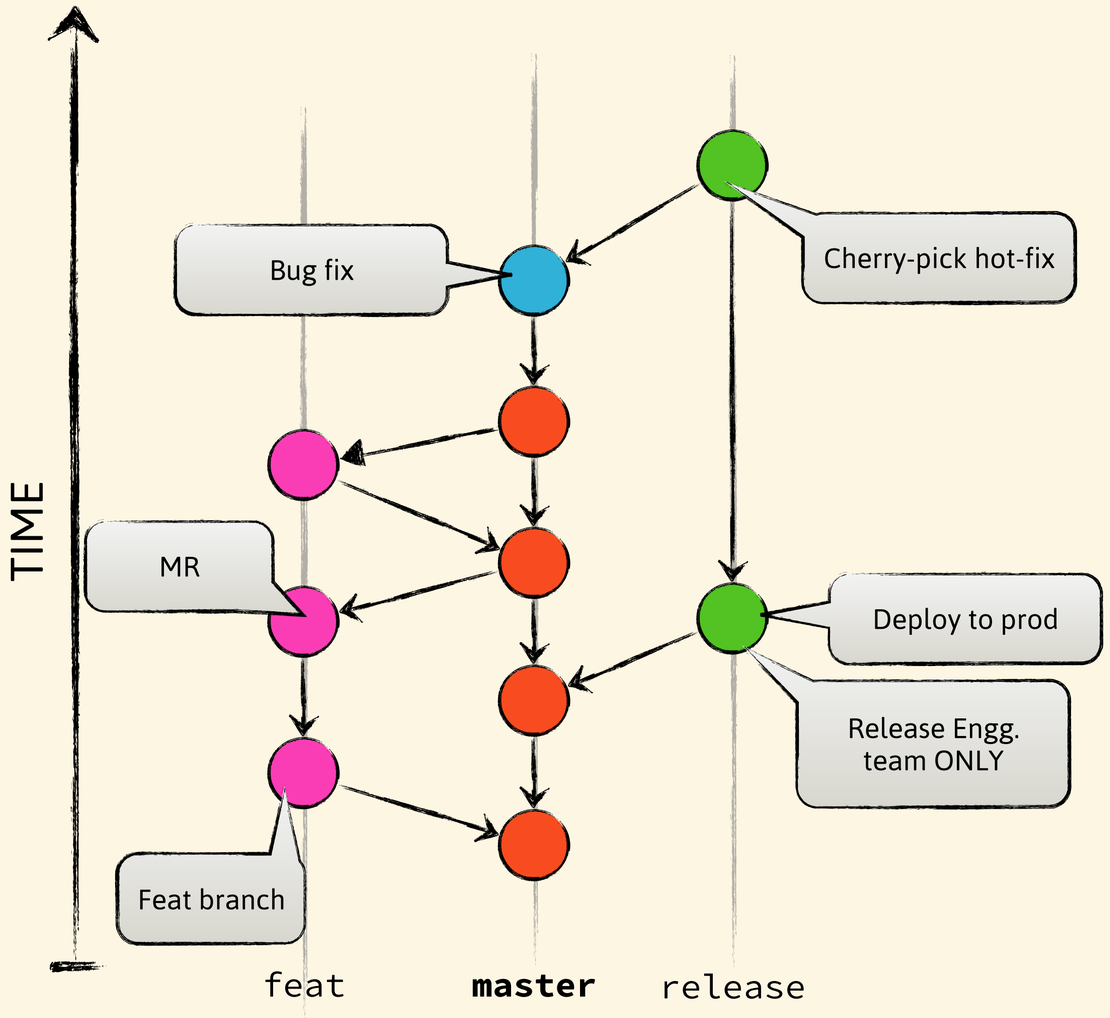
Trunk-based (Branch for release)



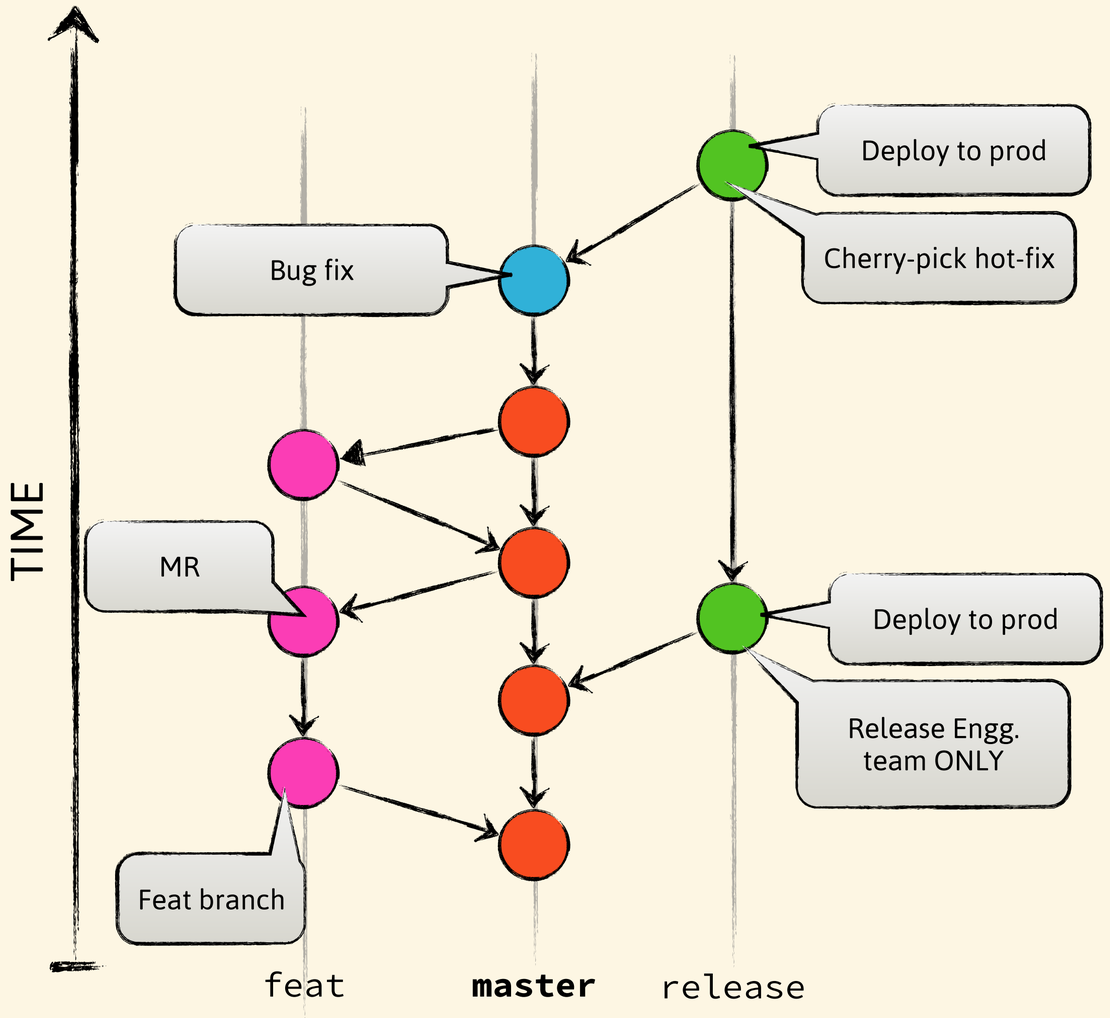
Trunk-based (Branch for release)



Trunk-based (Branch for release)



Trunk-based (Branch for release)



Trunk-based (Branch for release)

- Pros
  - Same as Trunk-Based
  - Scales for **very** large teams
- Cons
  - Same as Trunk-Based
  - Hot-fixes need to be cherry-picked
- See Trunk Based Development and Branch by Abstraction

THANKS

Credits

- Asap font
- Theme blatantly stolen from Git's homepage

Resources

- Pro Git by Scott Chacon
- Git from the bottom up by John Wiegley
- Git for Computer Scientists by Tommi Virtanen
- Git Core Concepts by Ted Naleid