Why Git{,Hub} is cool

Daniel Halperin

4 September 2012

What is Git?

- "Git is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency."
  - Website: [1] http://git-scm.com/
- Replacement for SVN, CVS, SCCS, ...

Reason #1

- Git is completely distributed
- Works perfectly well with no network
- Works perfectly well with no server configuration
- (Actually, everything is a server ;)

Demo #1

- Use git to do revision control for any little tiny project you work on

```
git init
touch file.txt
git add file.txt
git commit -s
<write a good shortlog and log>
```

Demo #2

- Run this command: `svn log`

  `<ping server>`
  `<communicate with server>`
  `<wait>`

- Oh, it worked. Finally.

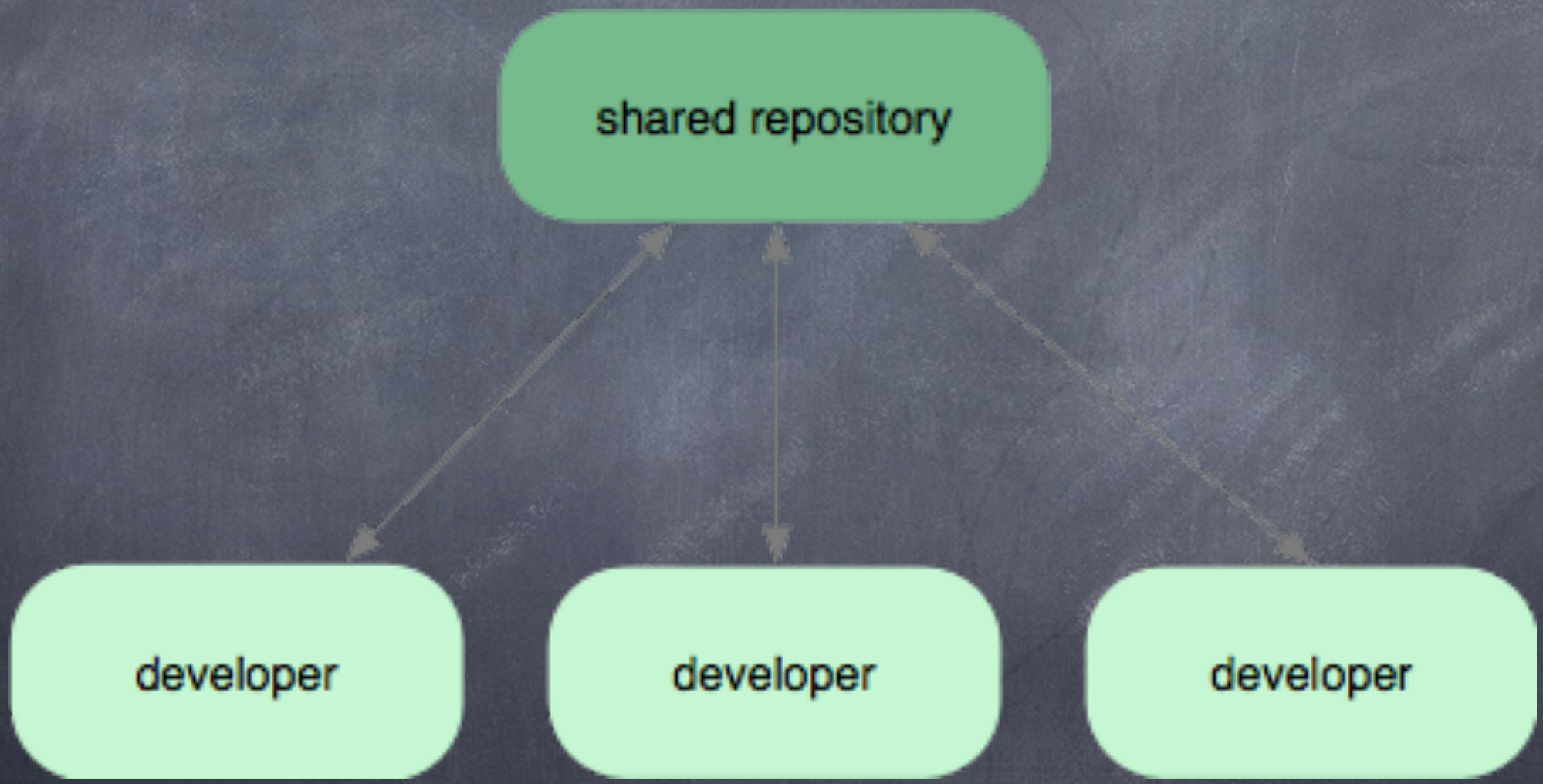
- This is always "instantaneous": `git log`

Reason #2

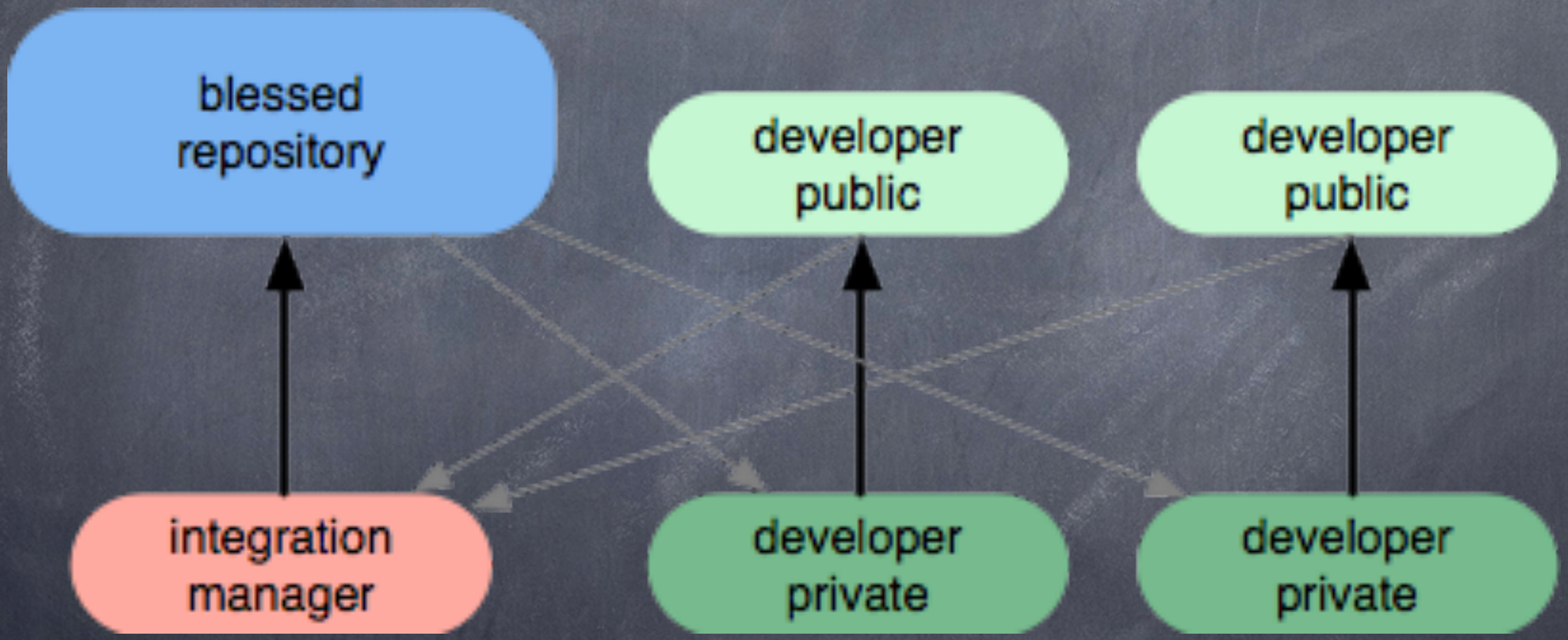
- GitHub is awesome
  - None of us, ever, have to administer anything
  - Lots of great features
  - We all have 100% of state on our computers, so trivial FT in emergencies
- Demo #3: let's go look at it

Basic workflow: Branching and Merging [1]

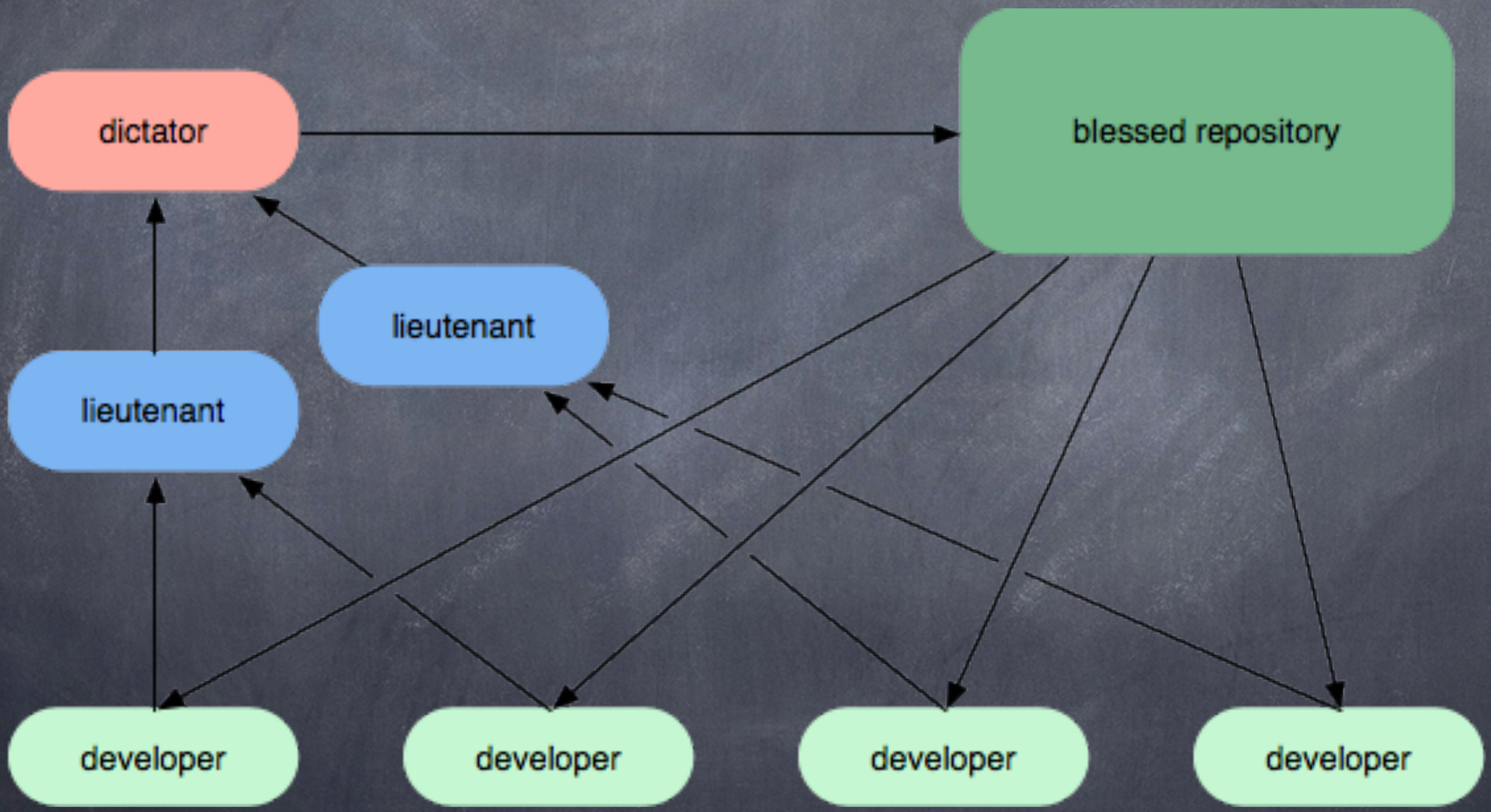
More workflows [1]: SVN-style (centralized)



More workflows [1]: Integration Manager



More workflows [1]: Dictator and Lieutenants



Case Study: Adding a feature

- Usually, this includes many steps:

  1. Refactoring some existing code to make room
  2. Implementing the core
  3. (e.g.) Making some user interface changes
  4. Adding documentation
  5. Adding tests

- SVN-style: code everything up and then commit a huge patch!

Issues with SVN-style

- SVN-style: code everything up and then commit a huge patch!
  - Hard for developer to separate changes from different steps
  - Instead, create partial-commits that may not compile and affect everyone.
  - If you introduce a bug, hard to find because `svn diff` gives ENTIRE delta.
- Impossible to properly review

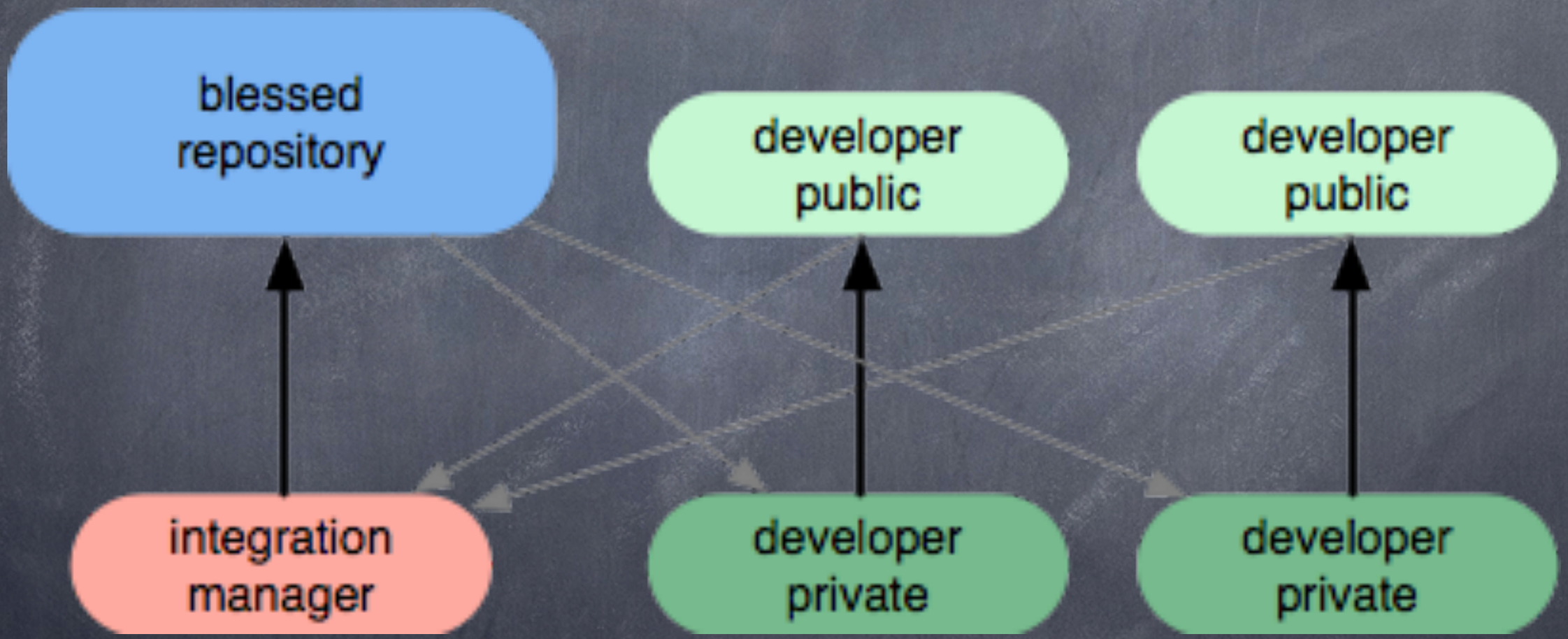
Git-style

- A commit should include one "logical change"
- The project should compile and pass all tests before and after every commit
- A feature comprises many commits, batch-submitted via a single "pull request"
- Project collaborators review and comment on the commit
- The commit is eventually merged once it "LGTM"s

Demo #4: Development branch and pull request

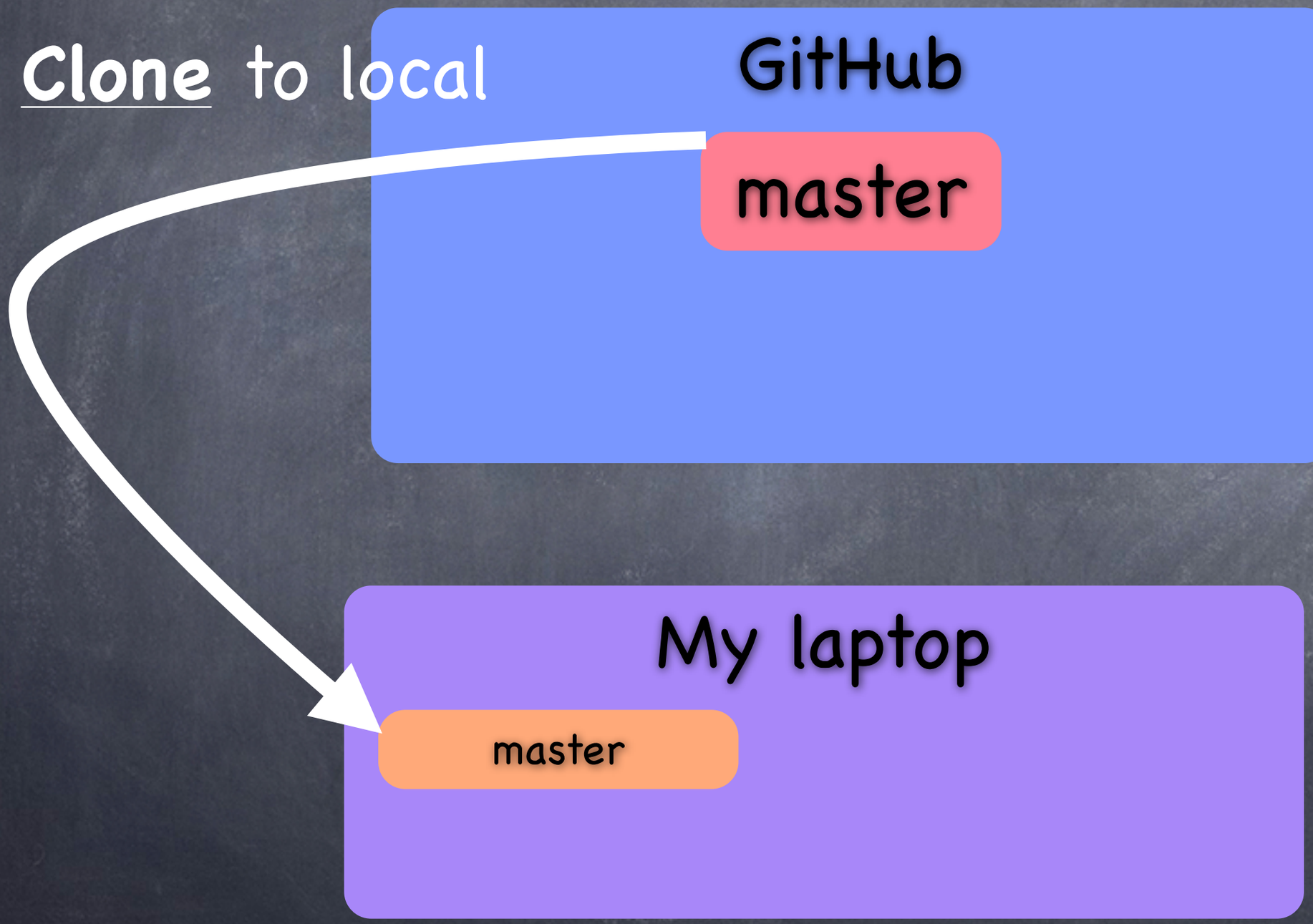
- I updated some comments.
- Make a new branch, commit locally.
- Push this new branch to the server.
- Submit a pull request.
- Get comments, eventually merge it to master.

Integration Manager workflow [1]

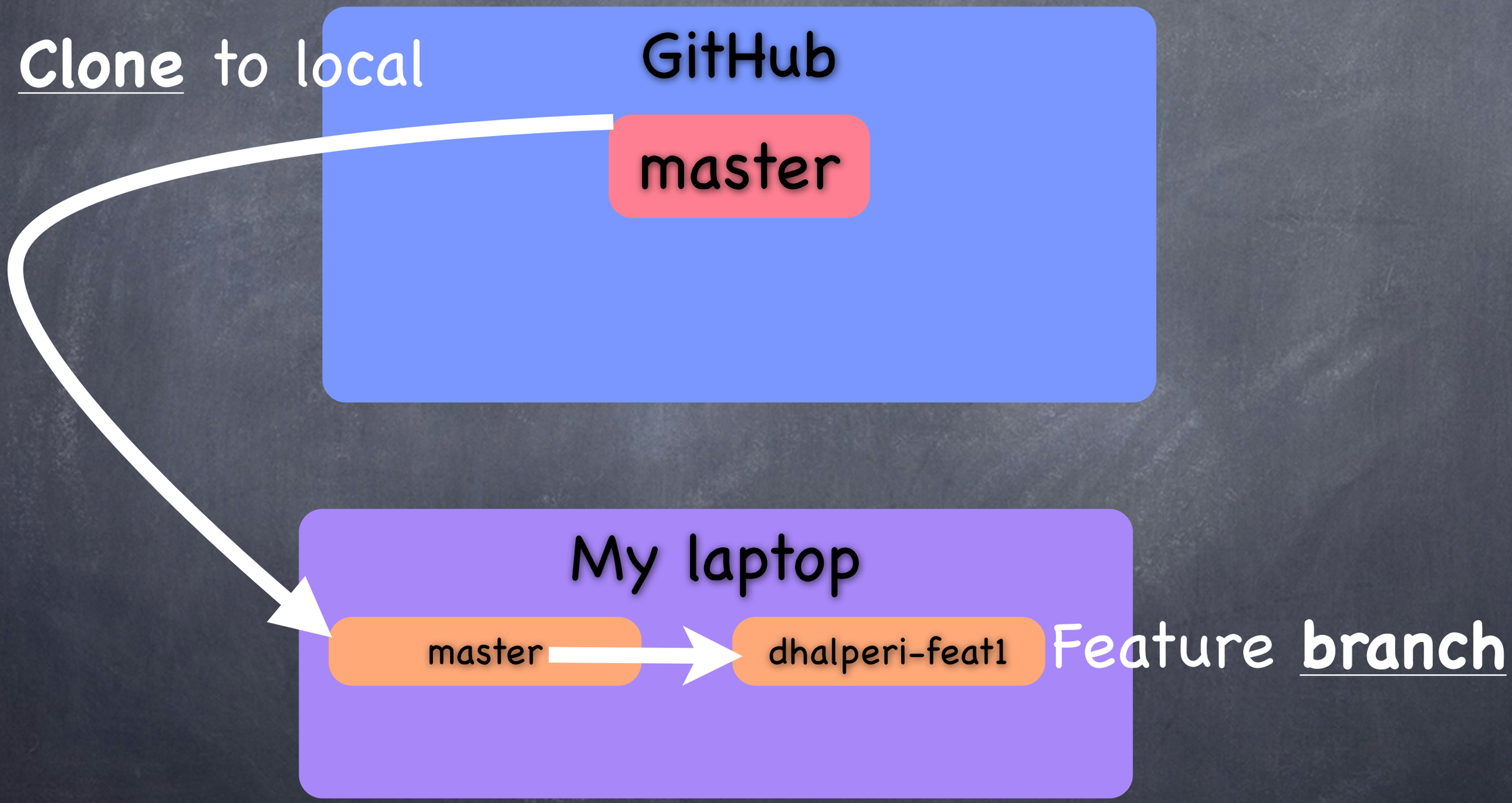


"Small Lab" workflow

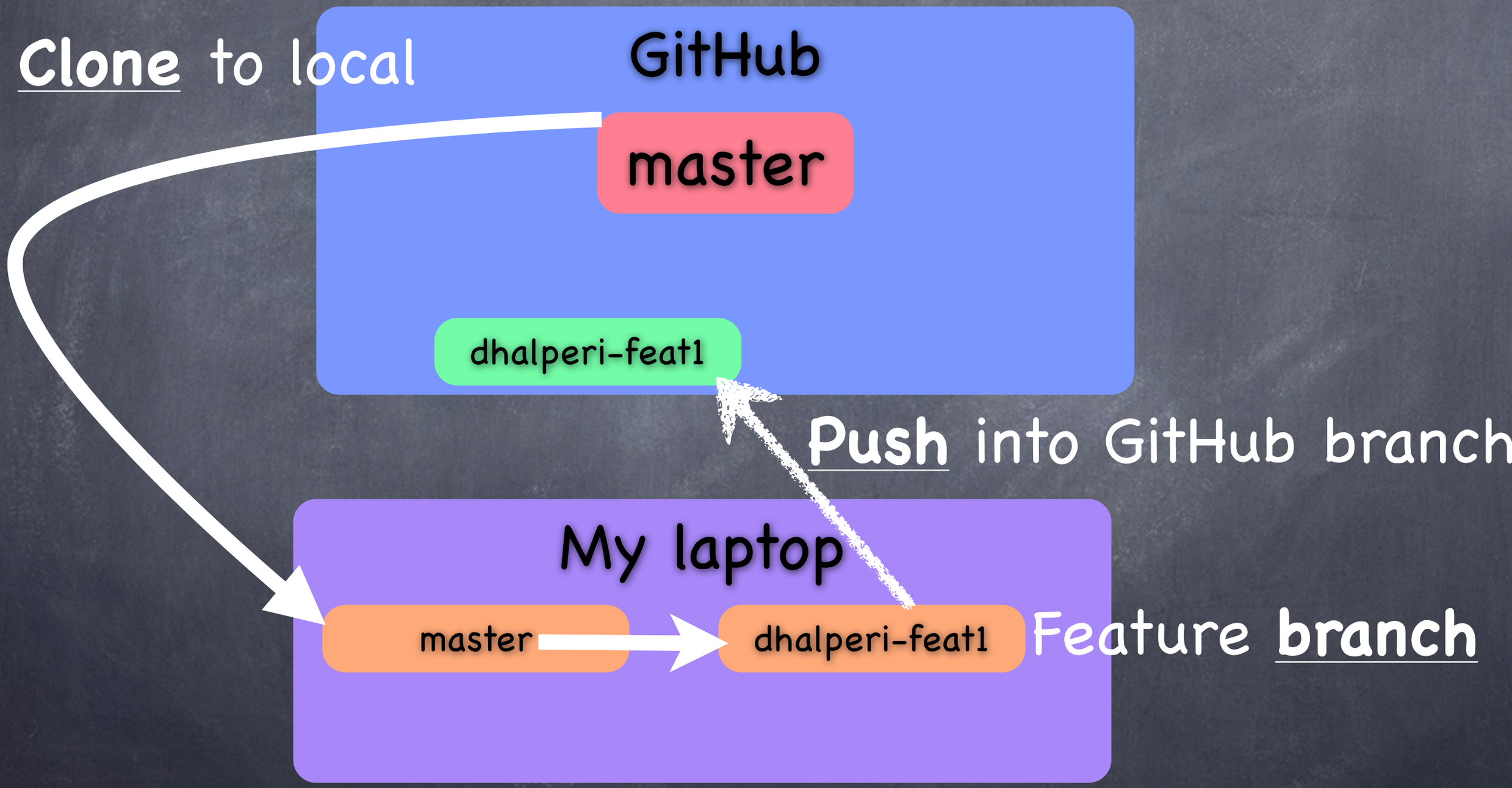
"Small Lab" workflow



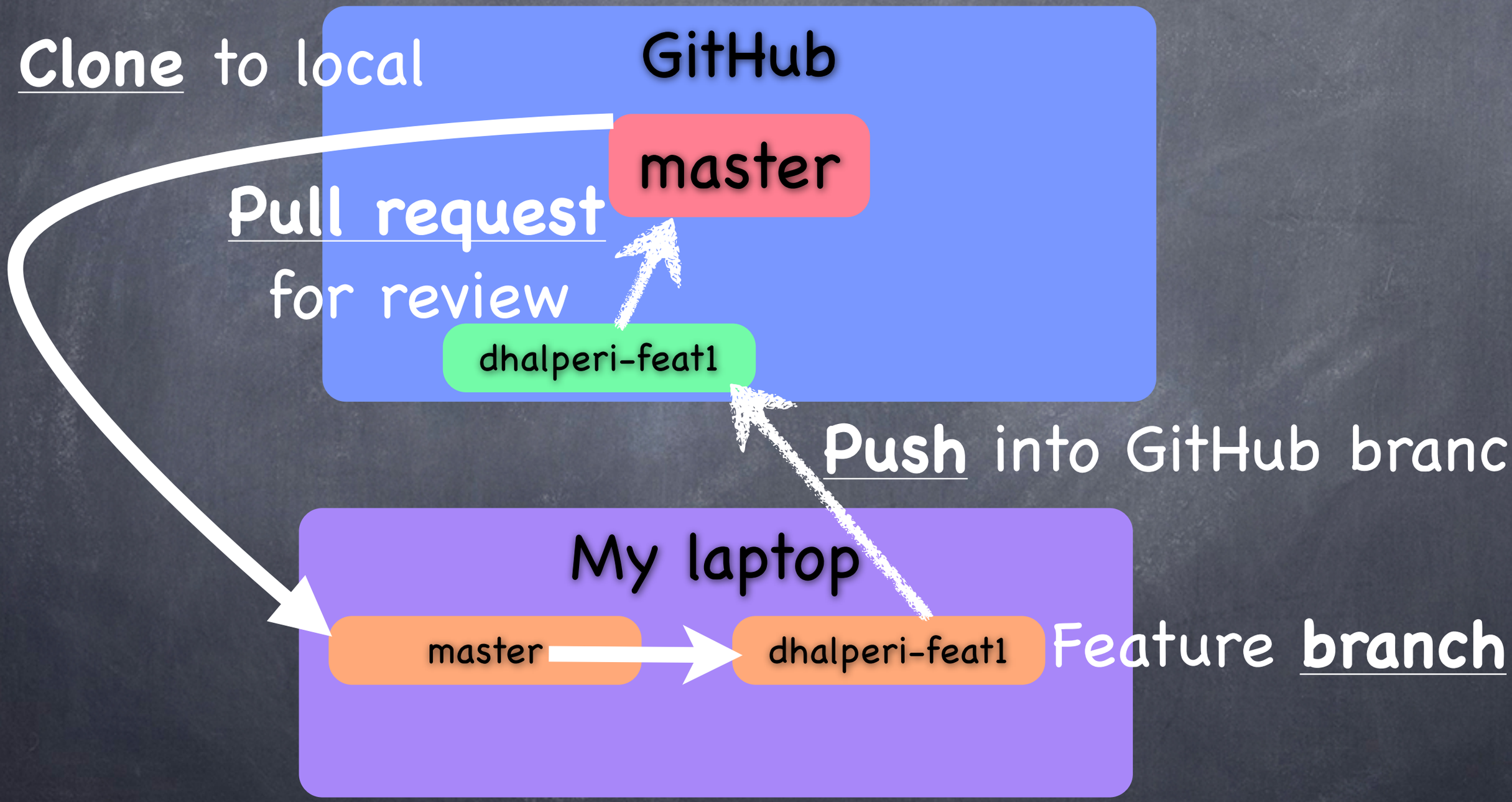
"Small Lab" workflow



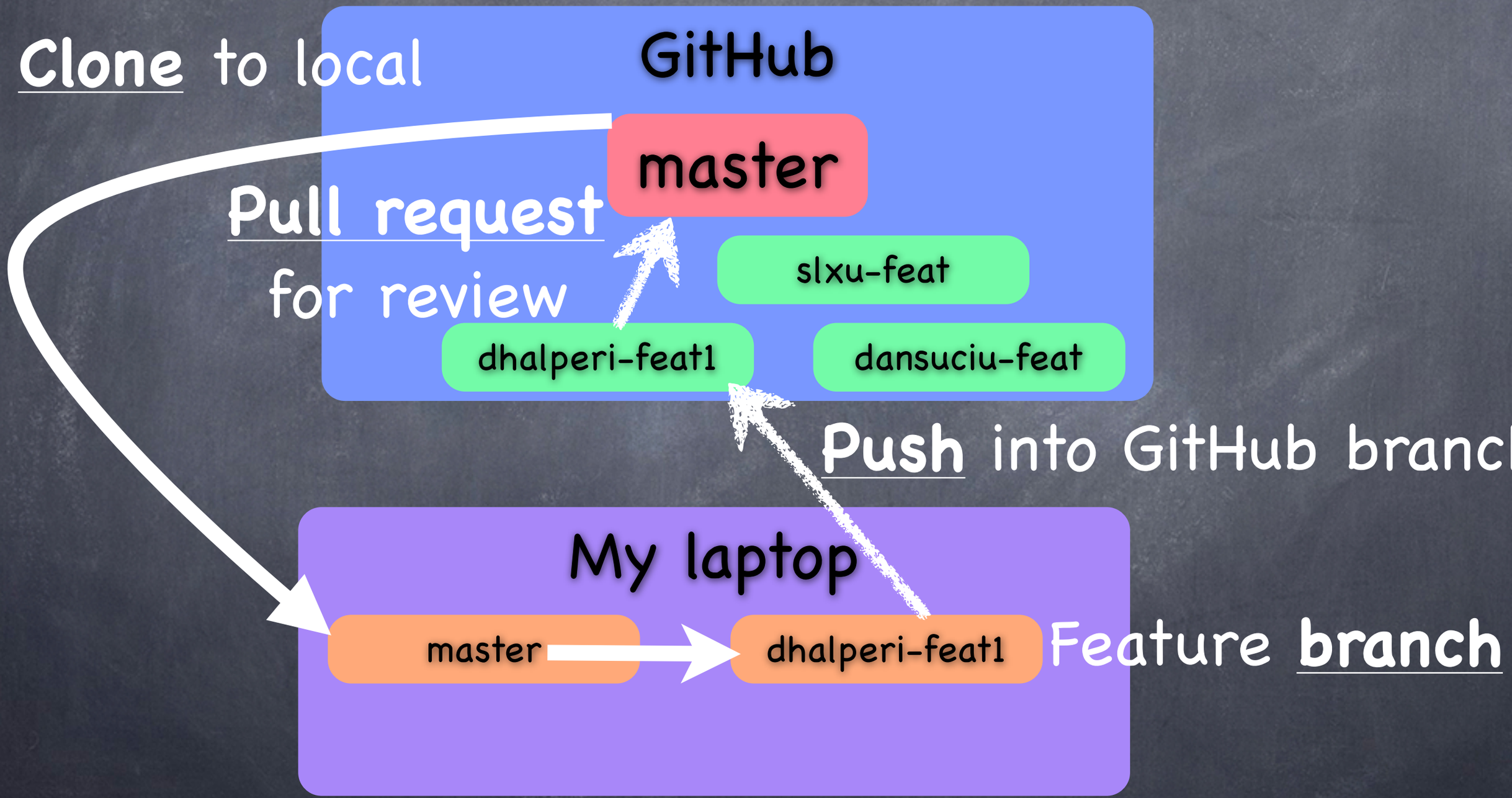
"Small Lab" workflow



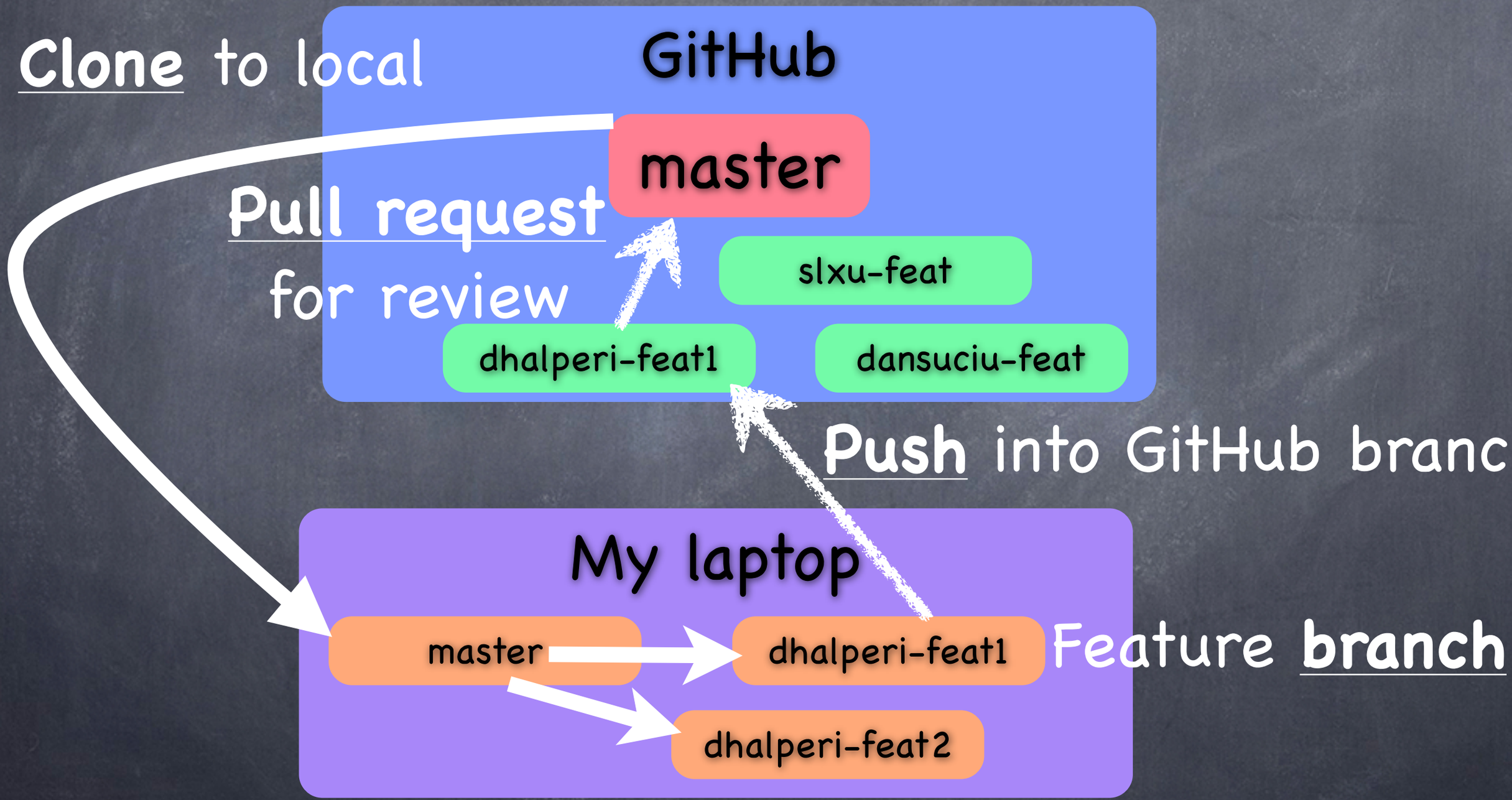
"Small Lab" workflow



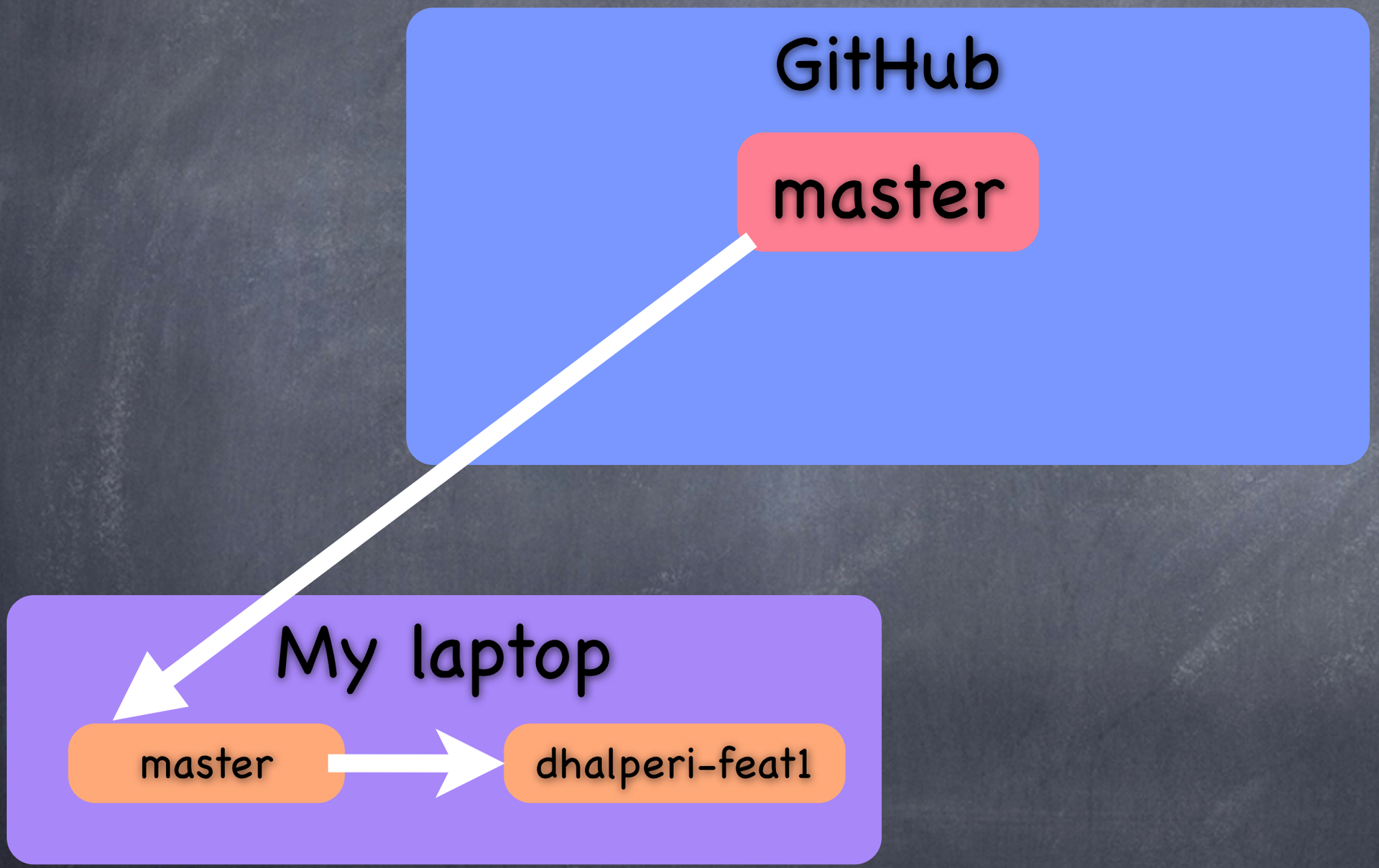
"Small Lab" workflow



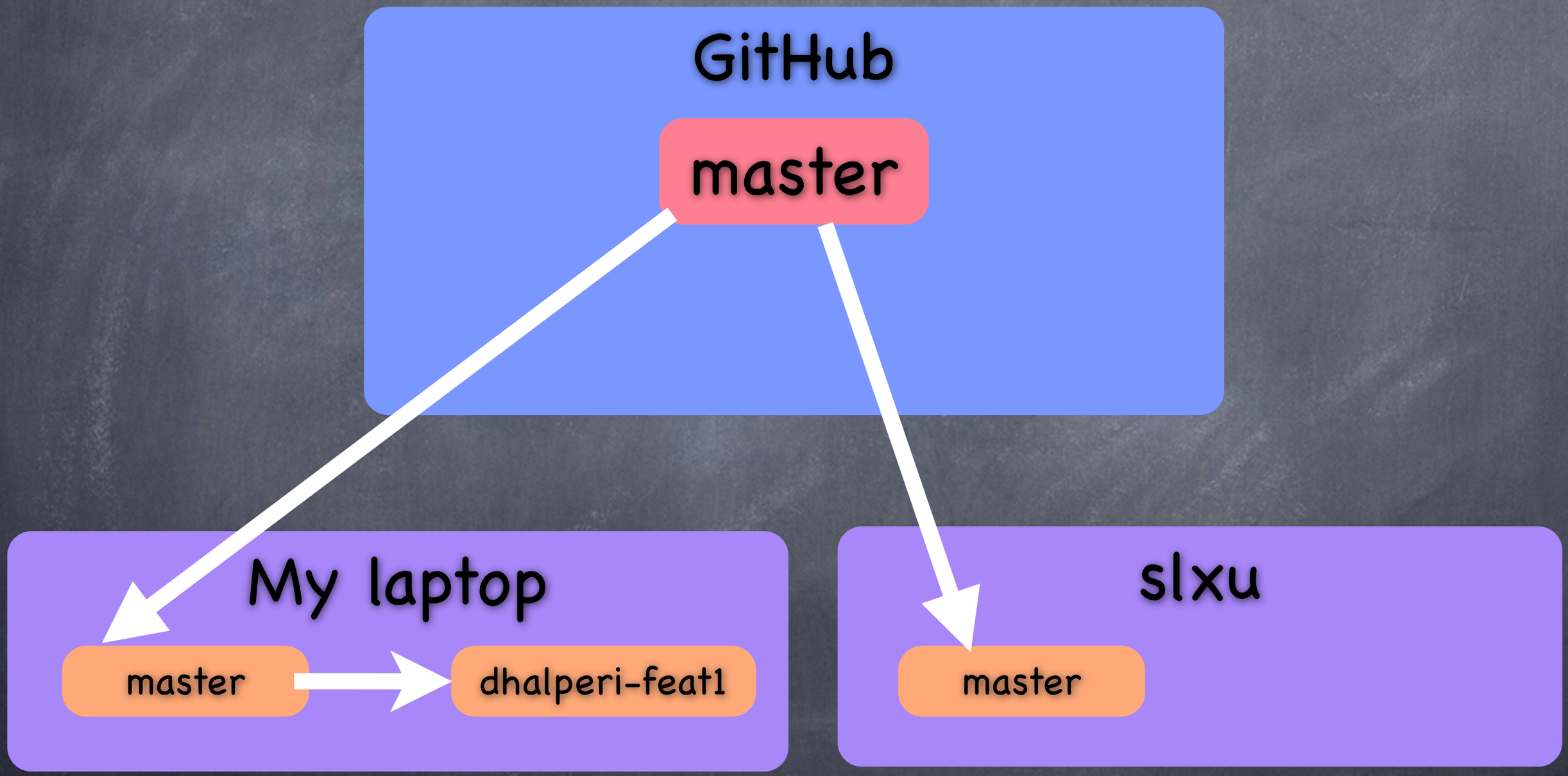
"Small Lab" workflow



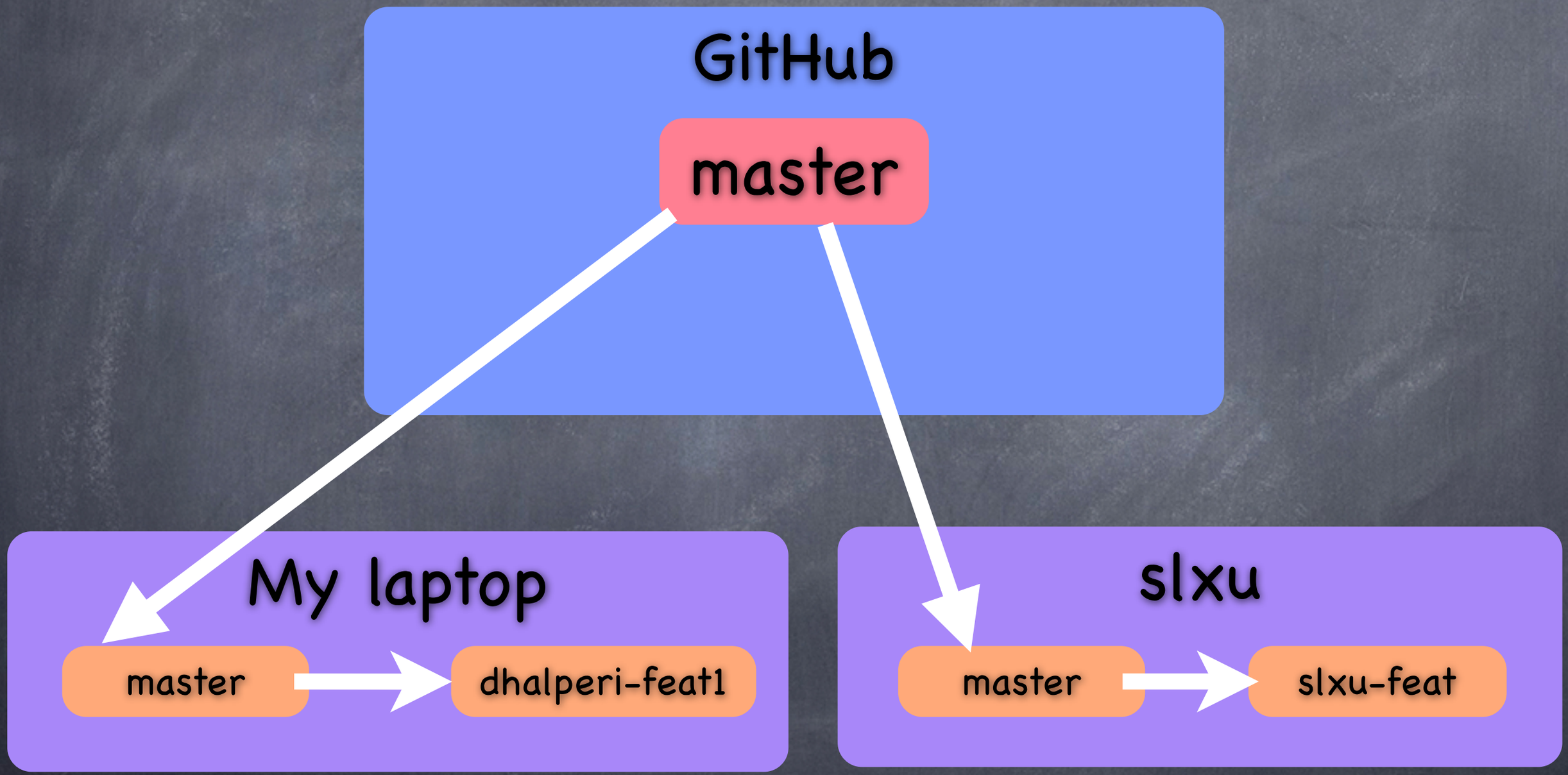
Conflicts happen!



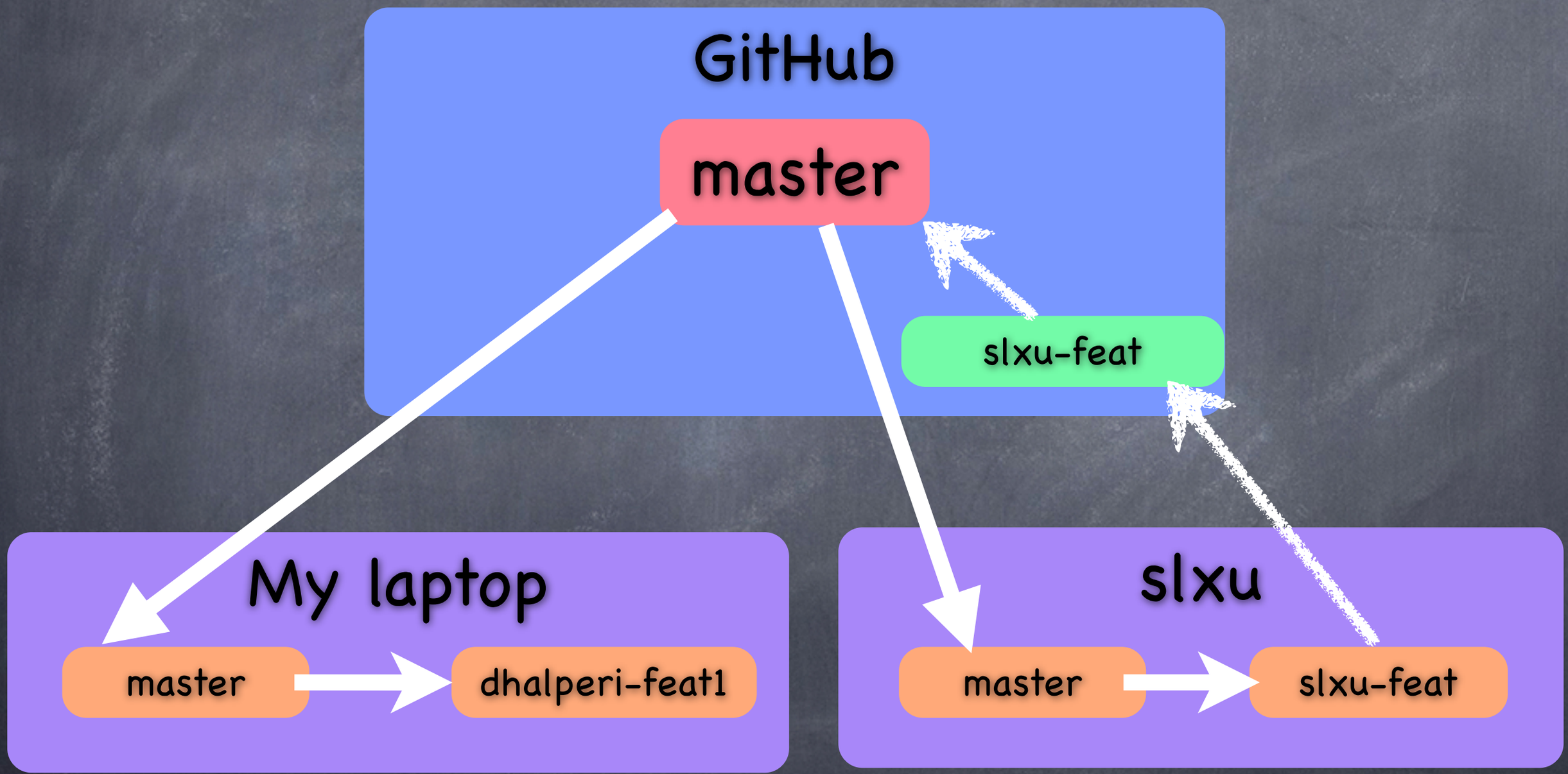
Conflicts happen!



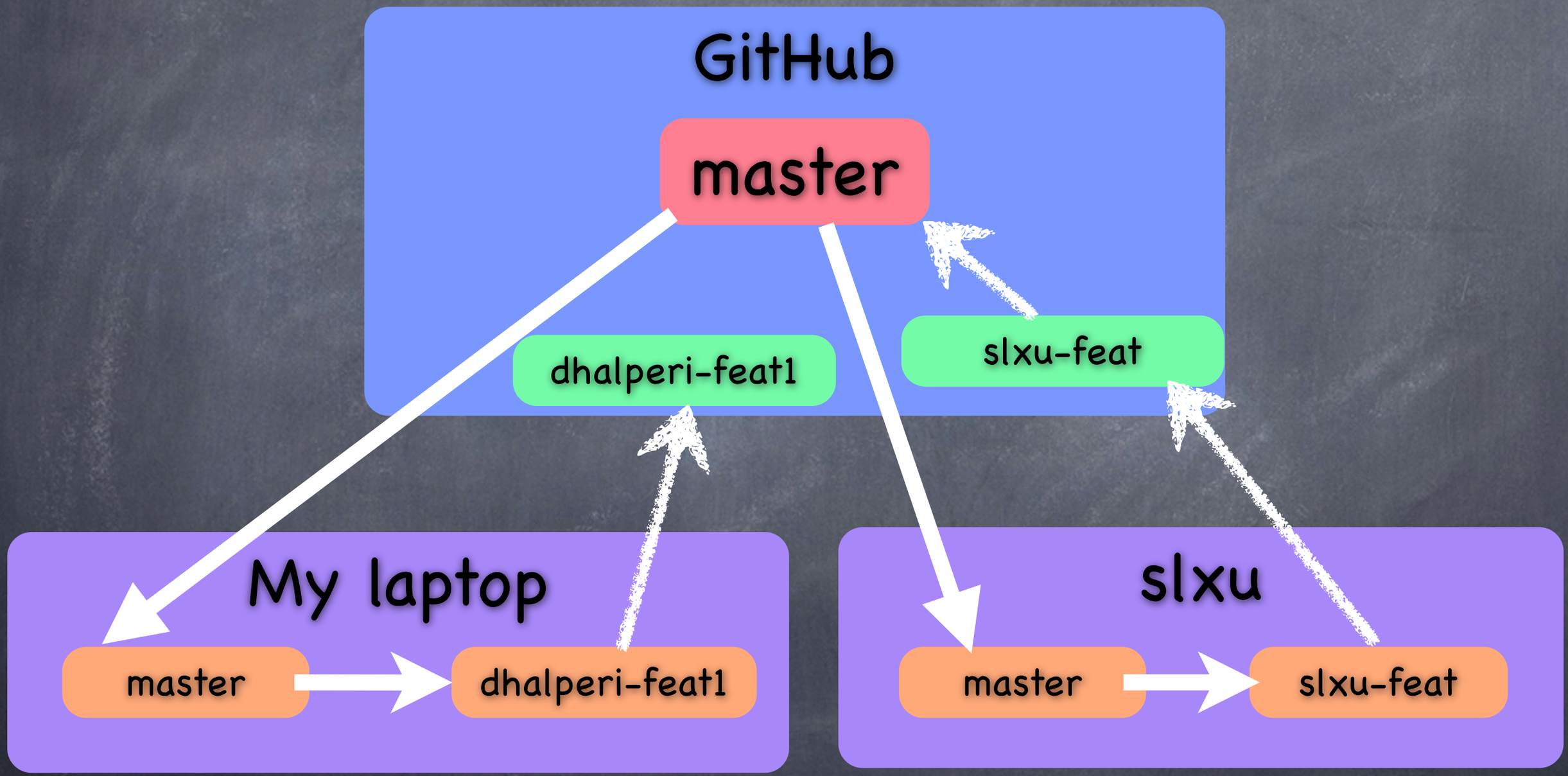
Conflicts happen!



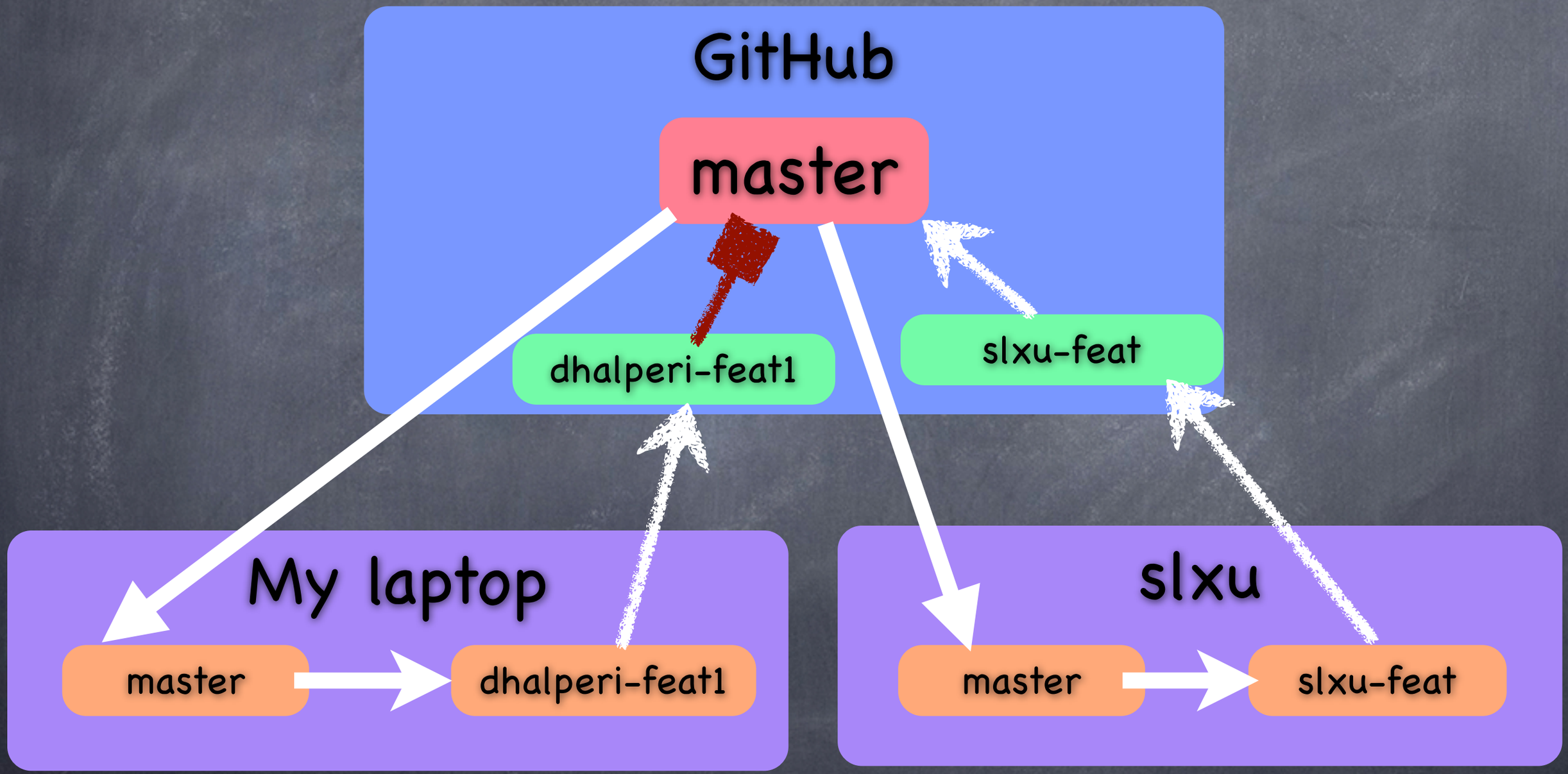
Conflicts happen!



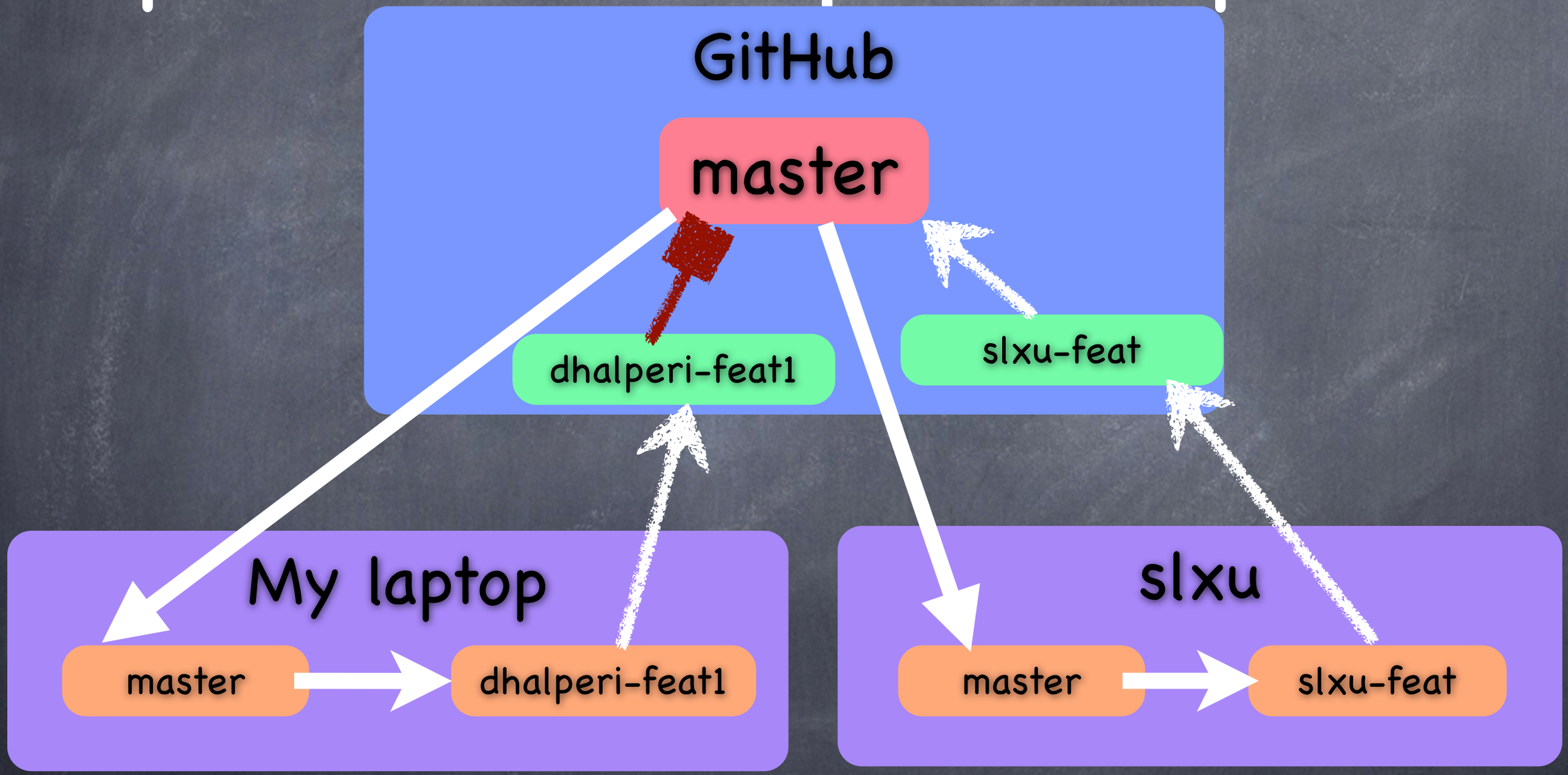
Conflicts happen!



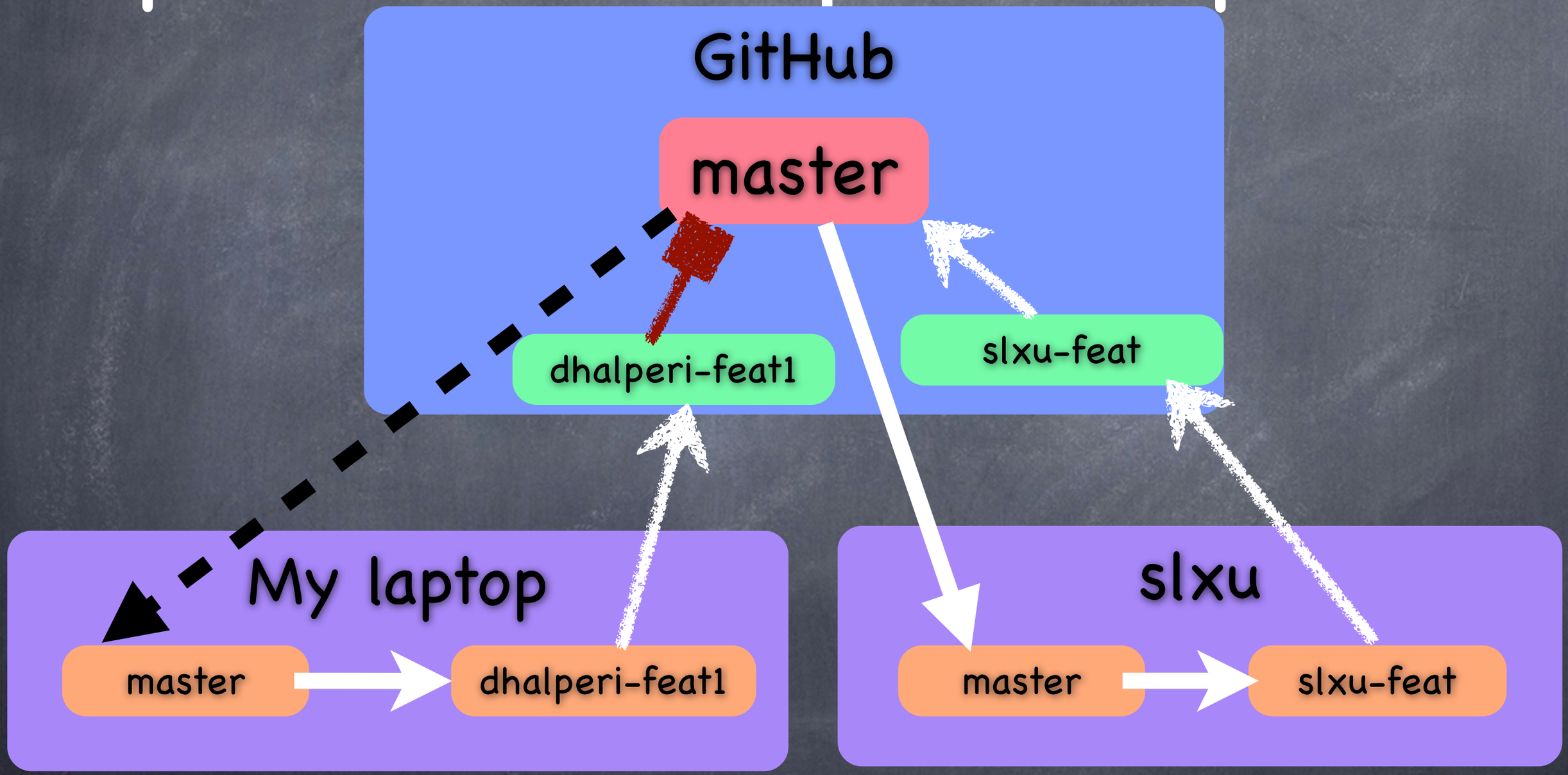
Conflicts happen!



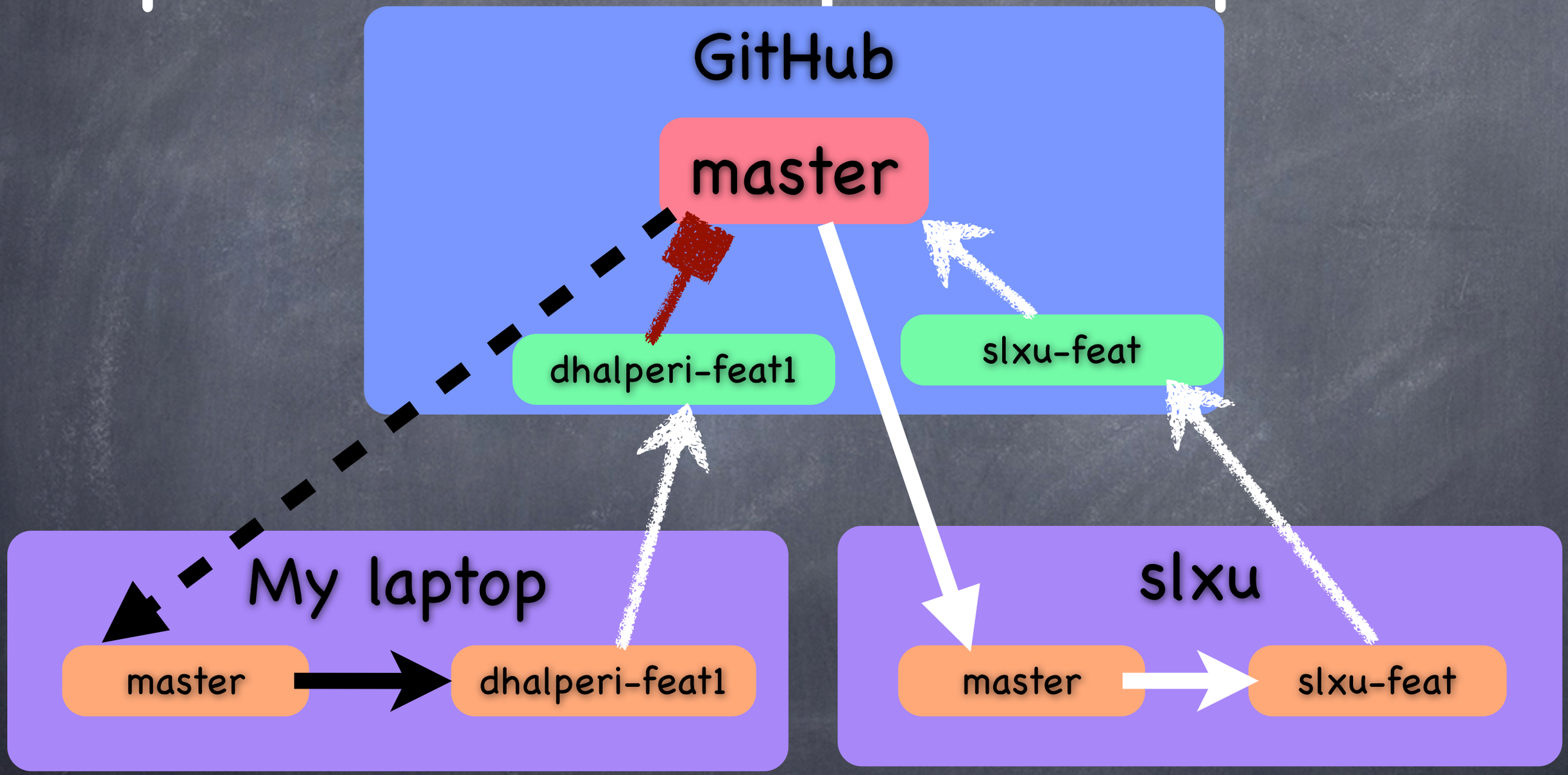
(1) Pull and merge, then re-push and re-pull-request



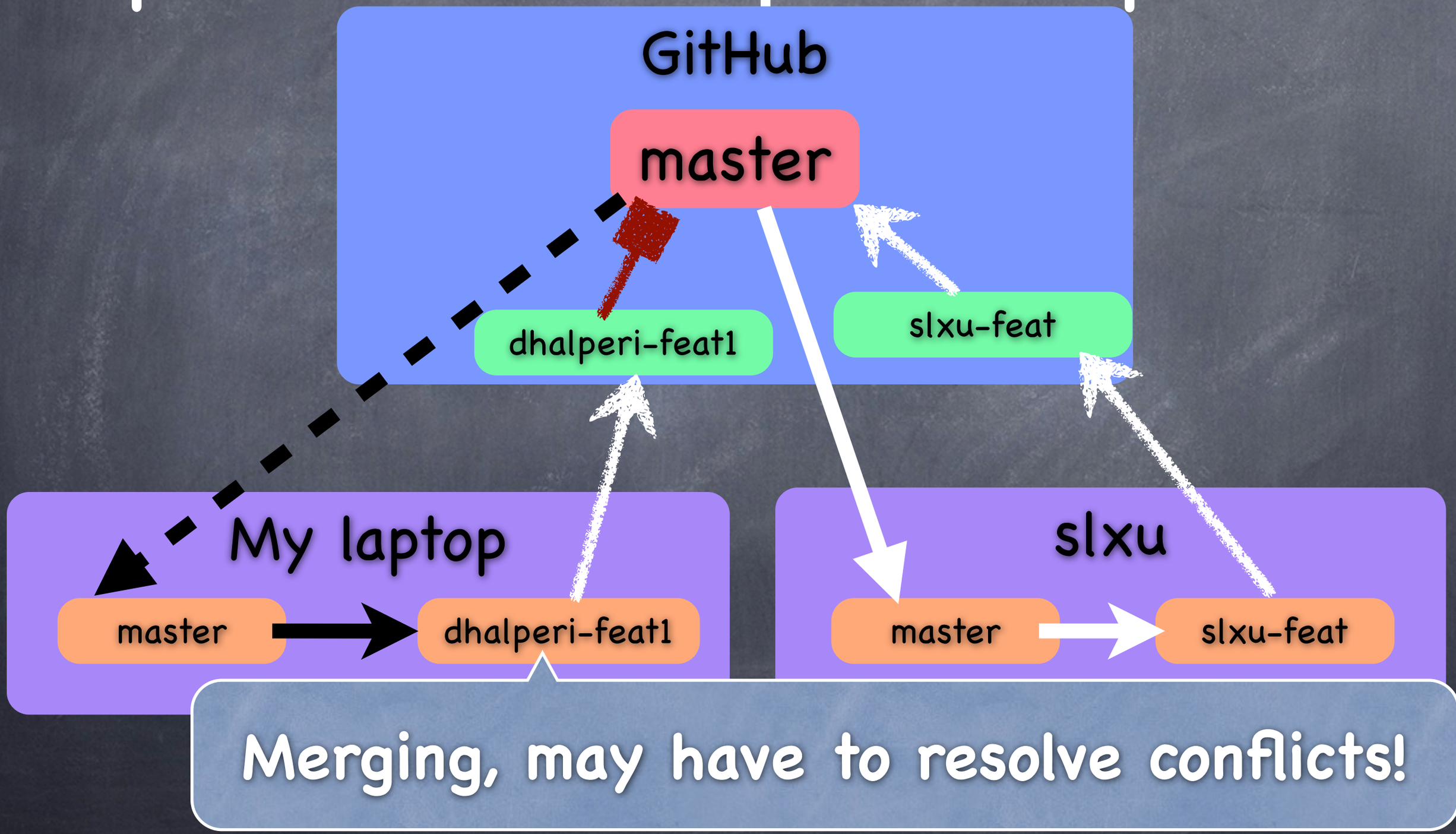
(1) Pull and merge, then re-push and re-pull-request



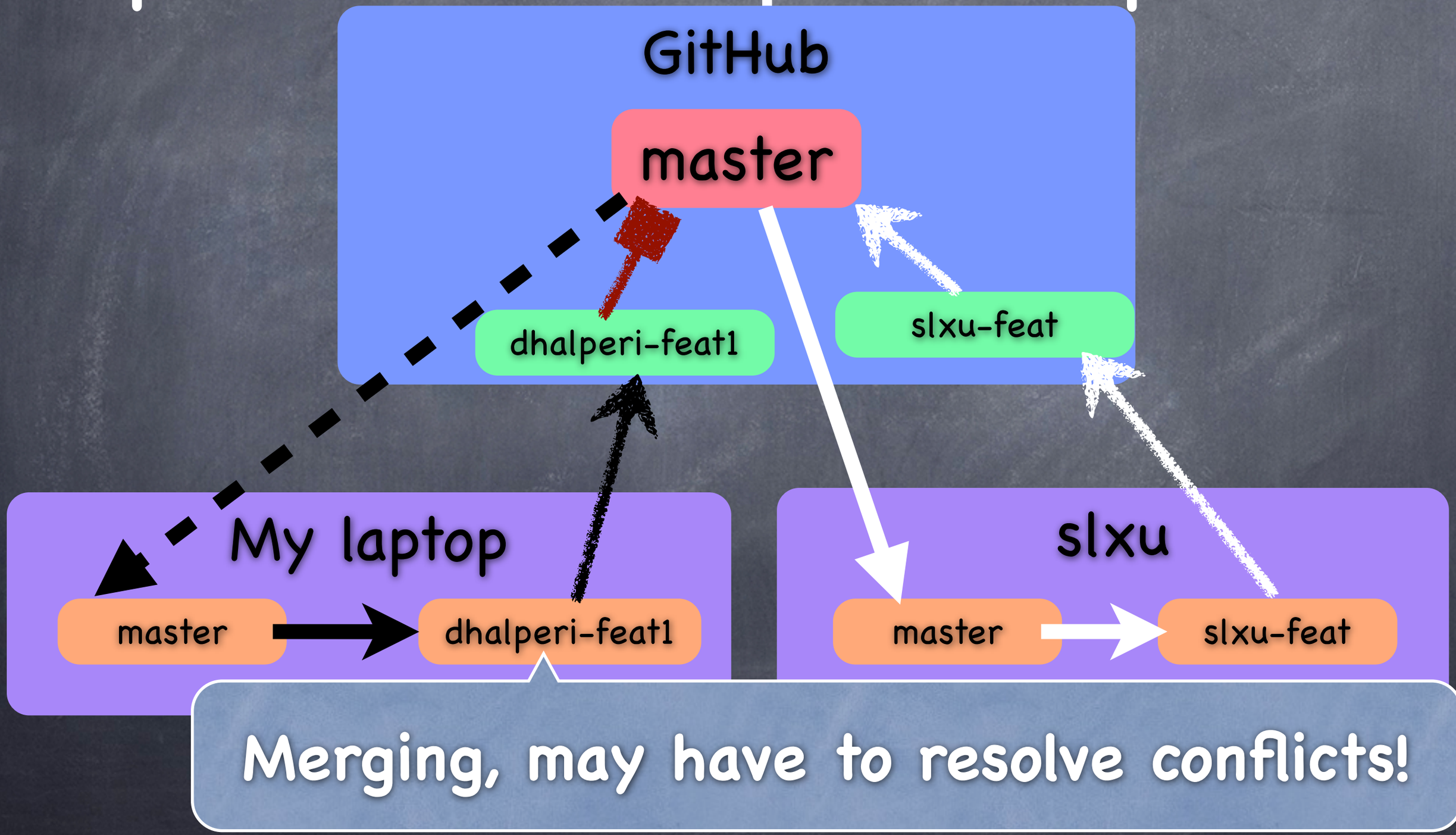
(1) Pull and merge, then re-push and re-pull-request



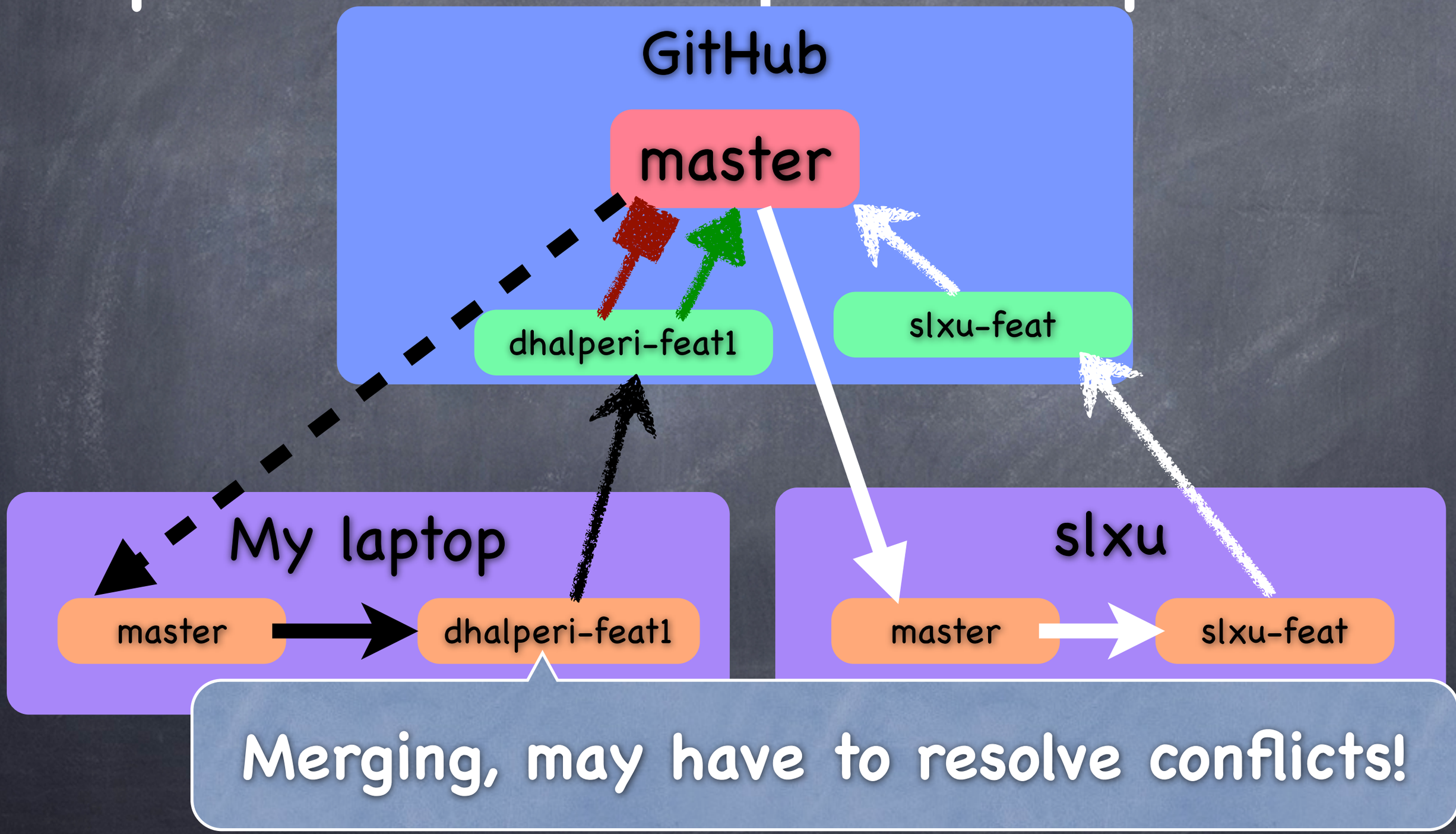
(1) Pull and merge, then re-push and re-pull-request



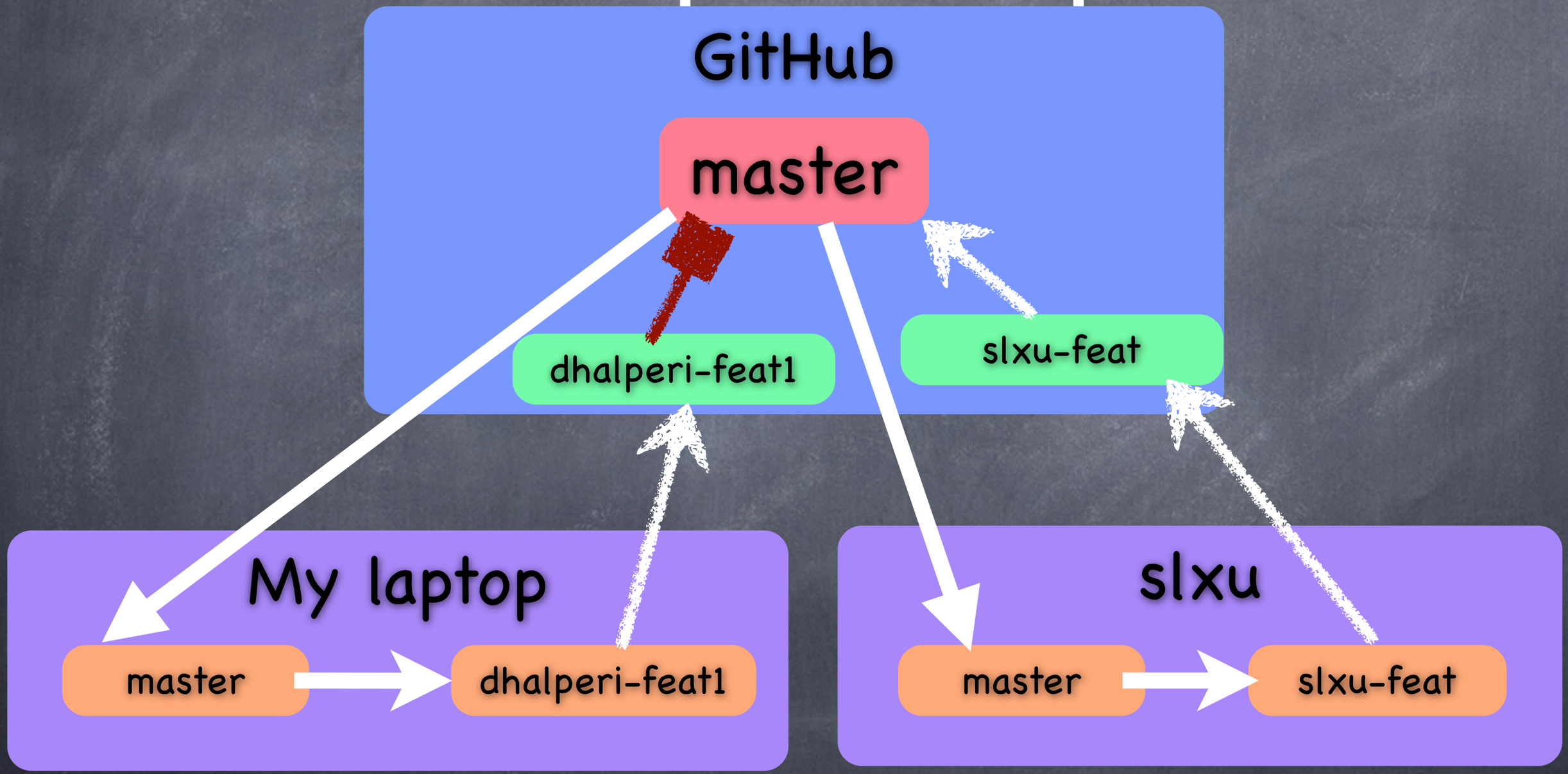
(1) Pull and merge, then re-push and re-pull-request



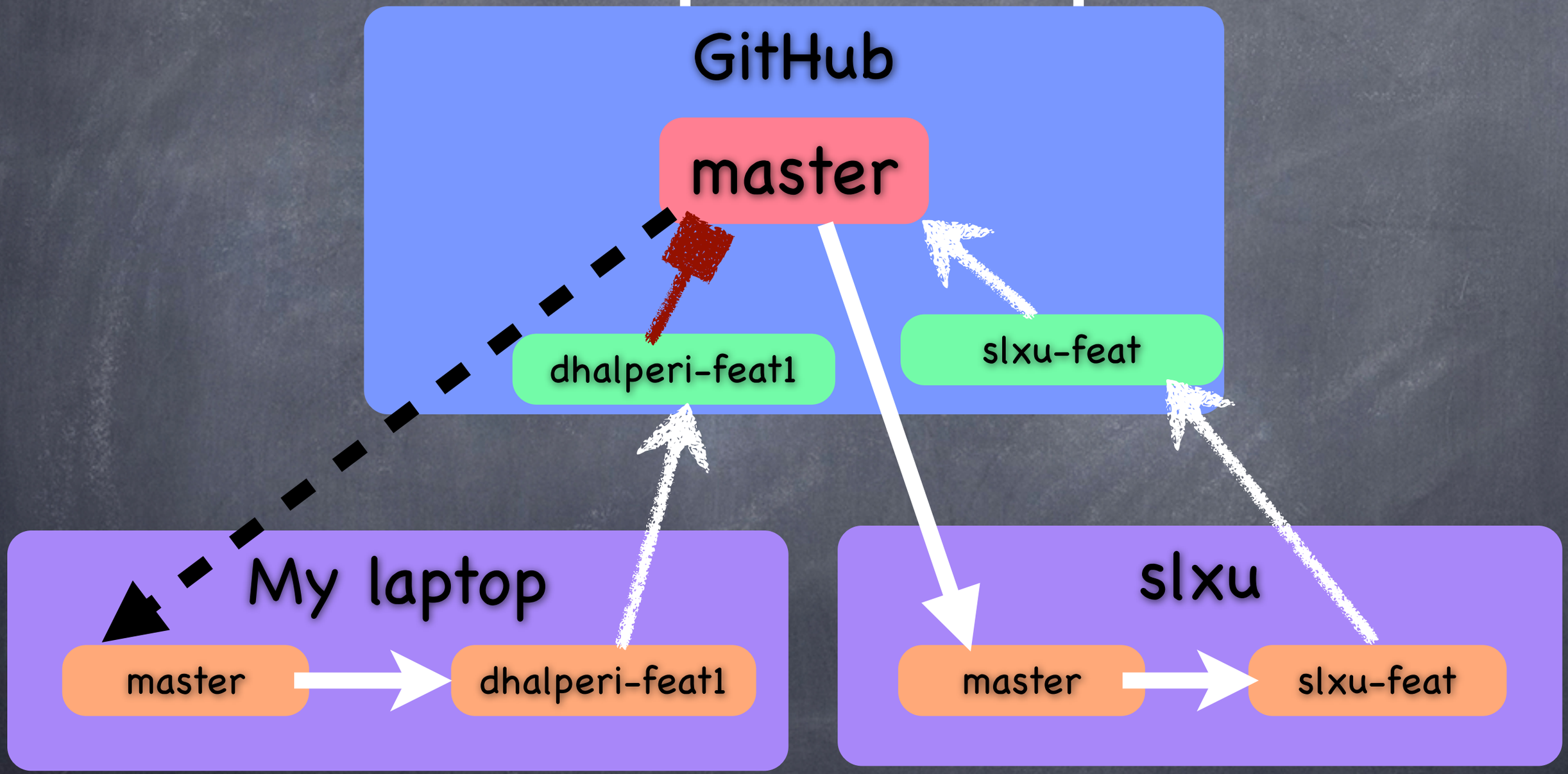
(1) Pull and merge, then re-push and re-pull-request



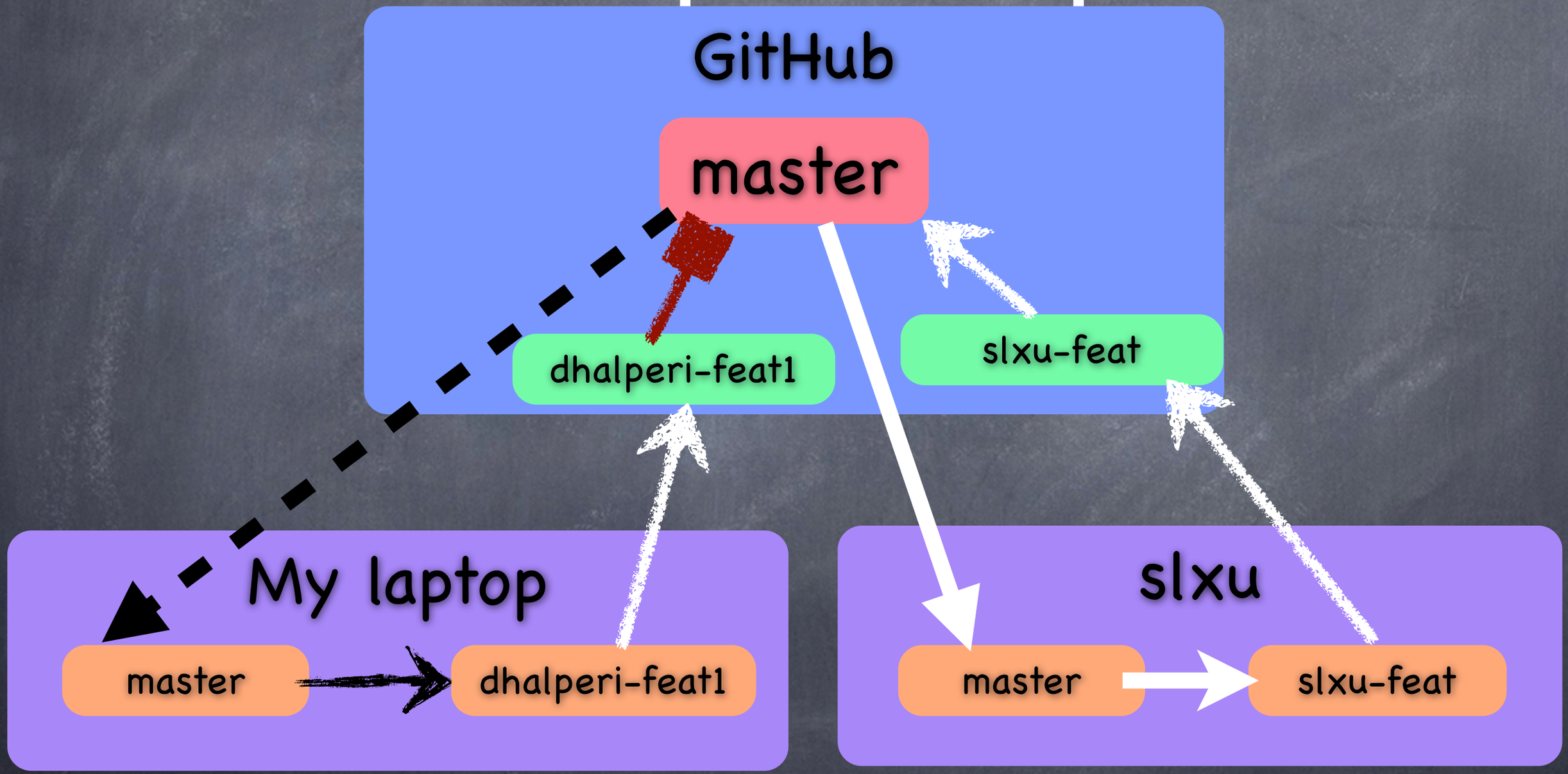
(2) Rebase, then re-push and re-pull-request



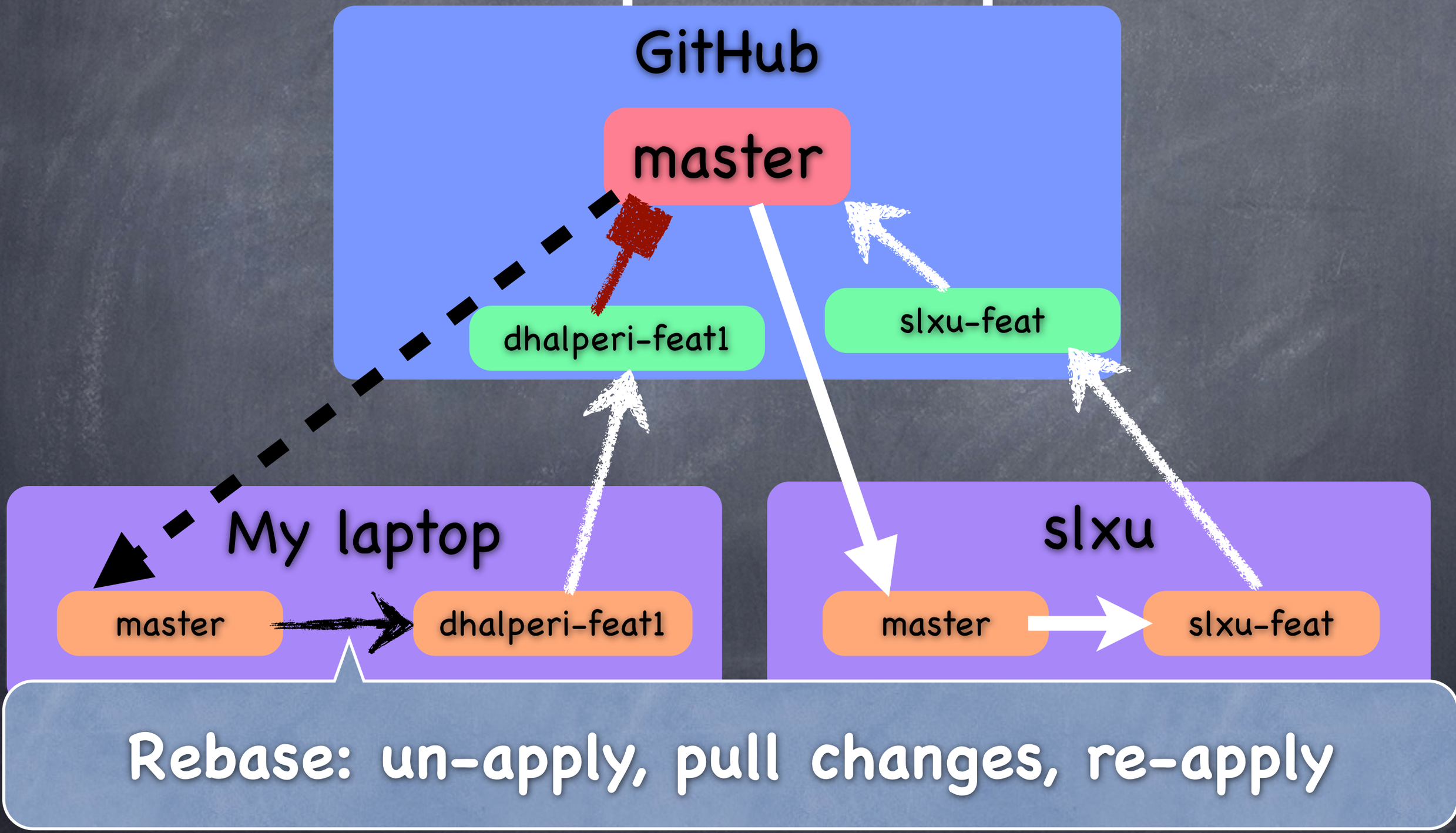
(2) Rebase, then re-push and re-pull-request



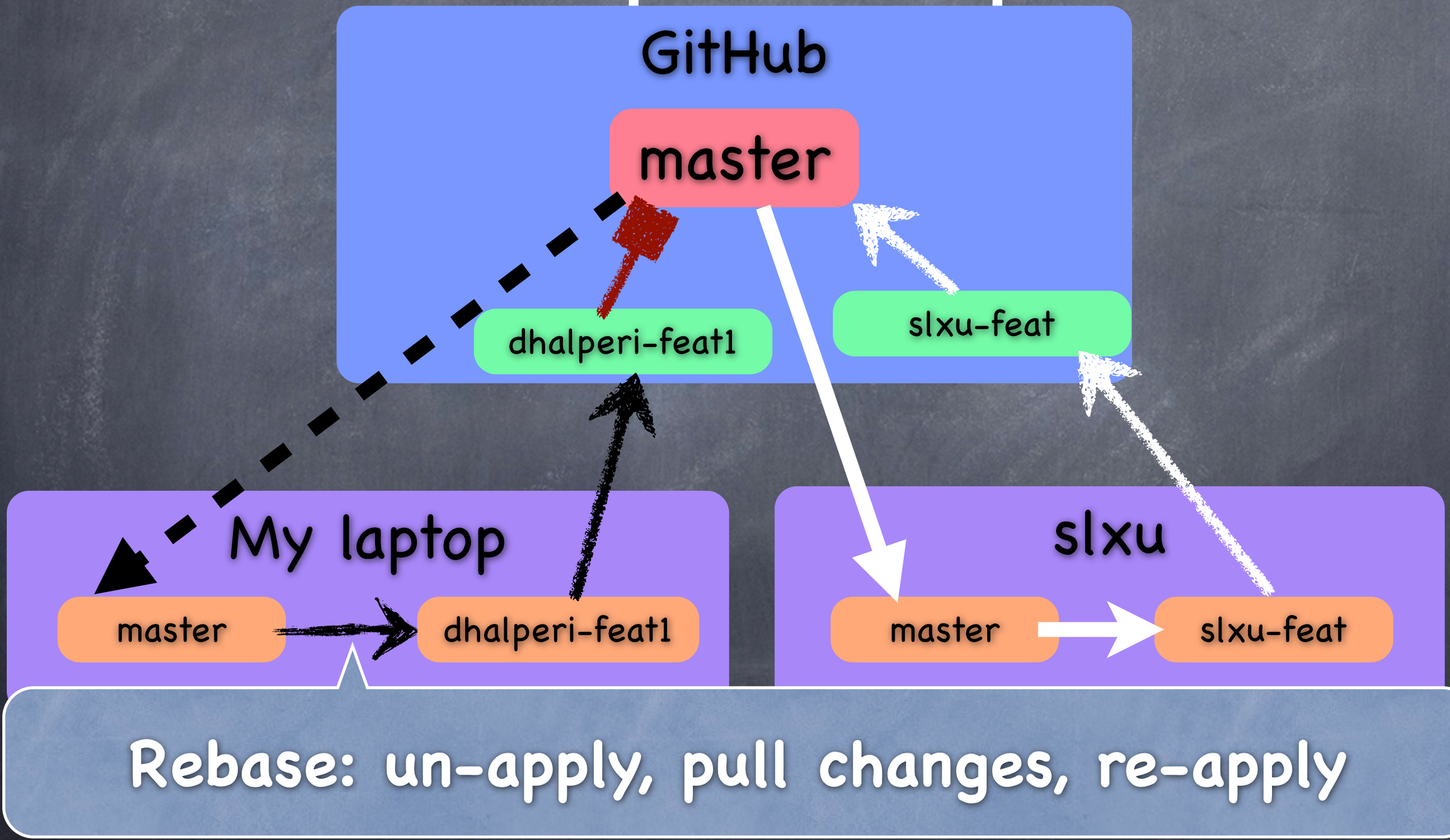
(2) Rebase, then re-push and re-pull-request



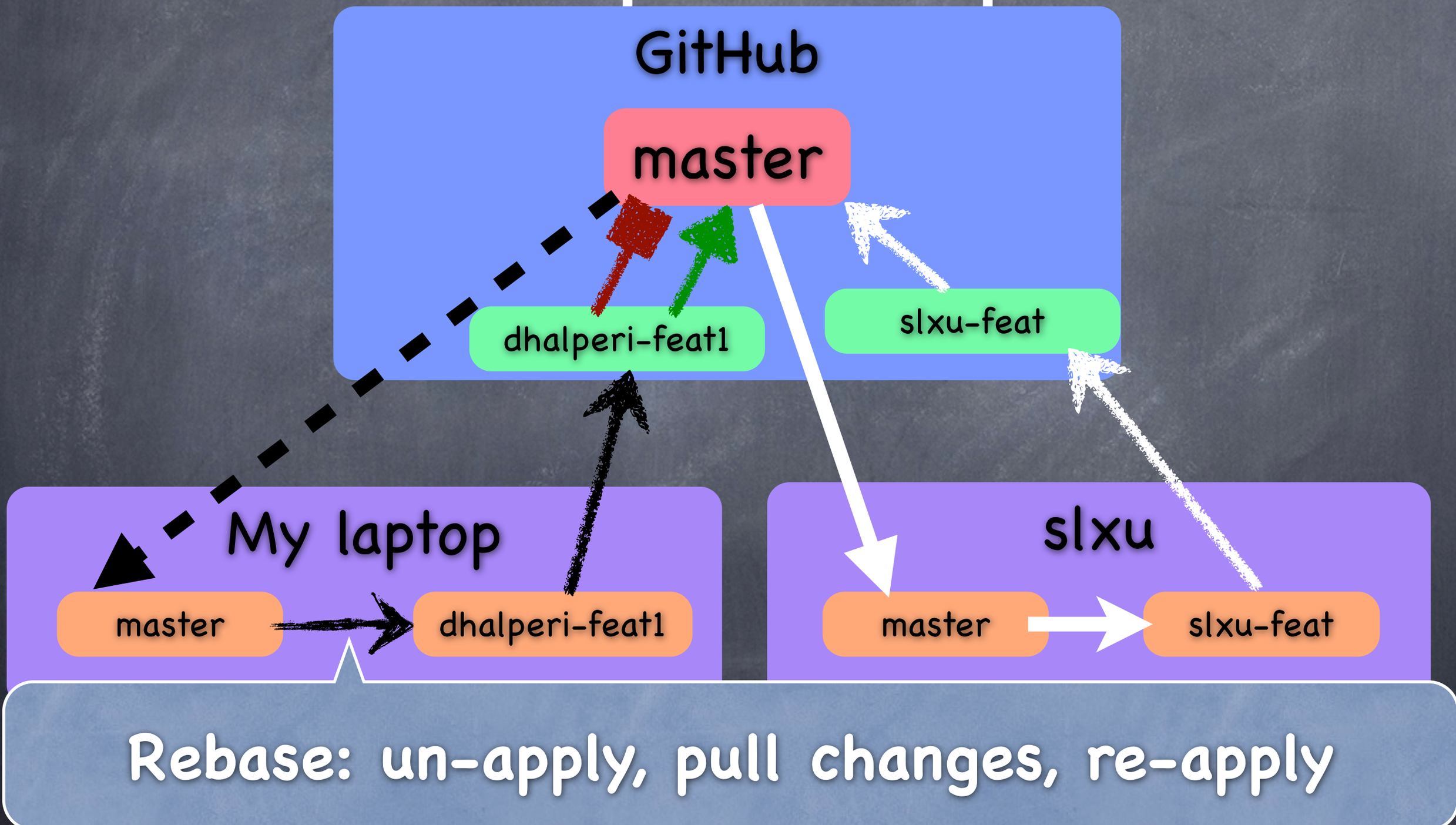
(2) Rebase, then re-push and re-pull-request



(2) Rebase, then re-push and re-pull-request



(2) Rebase, then re-push and re-pull-request



Key principles

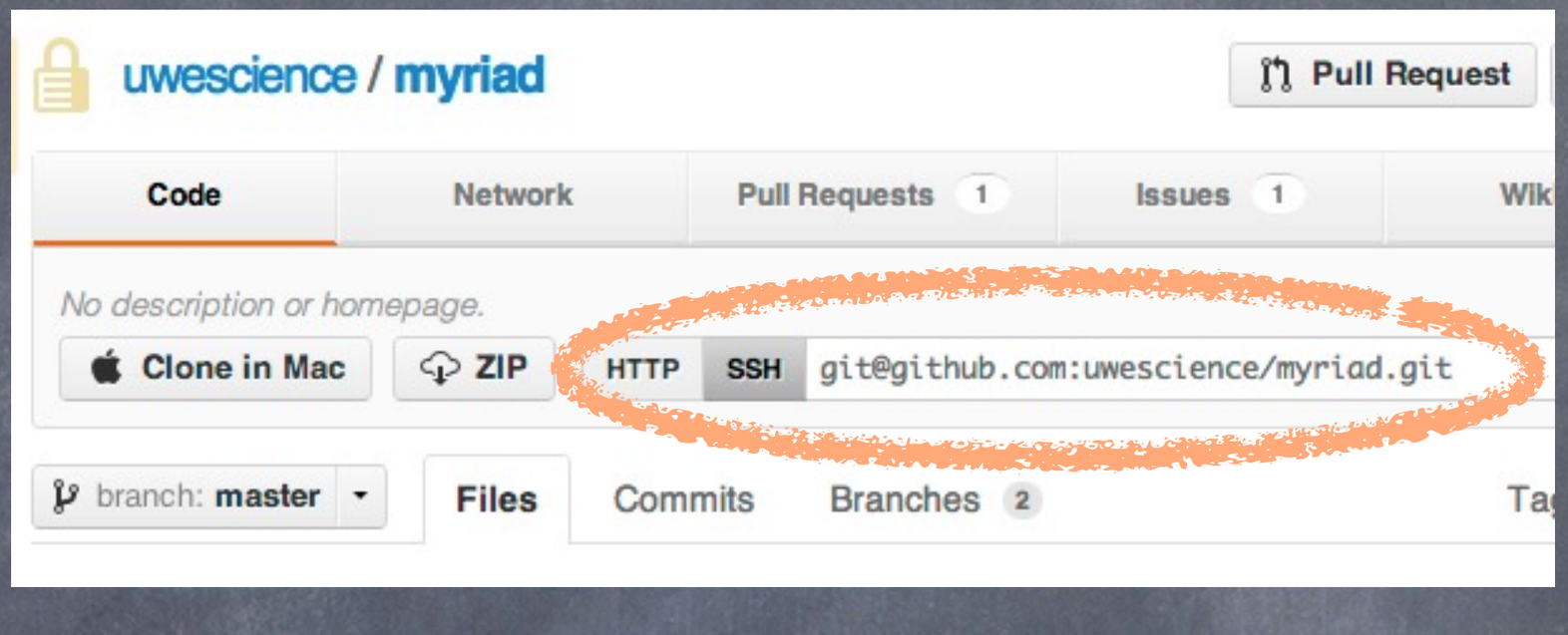
- Make your code easy to review
  - 1 change / commit, good shortlog and log messages
  - Lots of commits, batched into 1 pull request
- Minimize surprises / minimize conflicts
  - Work in a feature branch -> your master can always pull GitHub master w/no conflicts
    - Rebase and/or merge conflicts in your feature branch instead
  - Never push directly to master
    - Push to GitHub branch, then submit pull request
    - All code reviewed before it reaches master branch

Summary

- Git, in my opinion, has much better workflow than SVN
- Distributed >> Centralized
- GitHub has tons of crazy features
  - Commit emails, other simple things
  - Nets Lab has continuous integration testing going w/Jenkins
- What else do you want to know?

Translating from SVN to Git

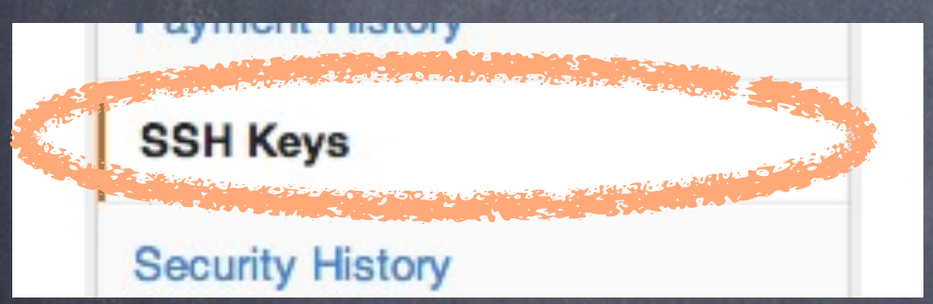
Translate: svn checkout

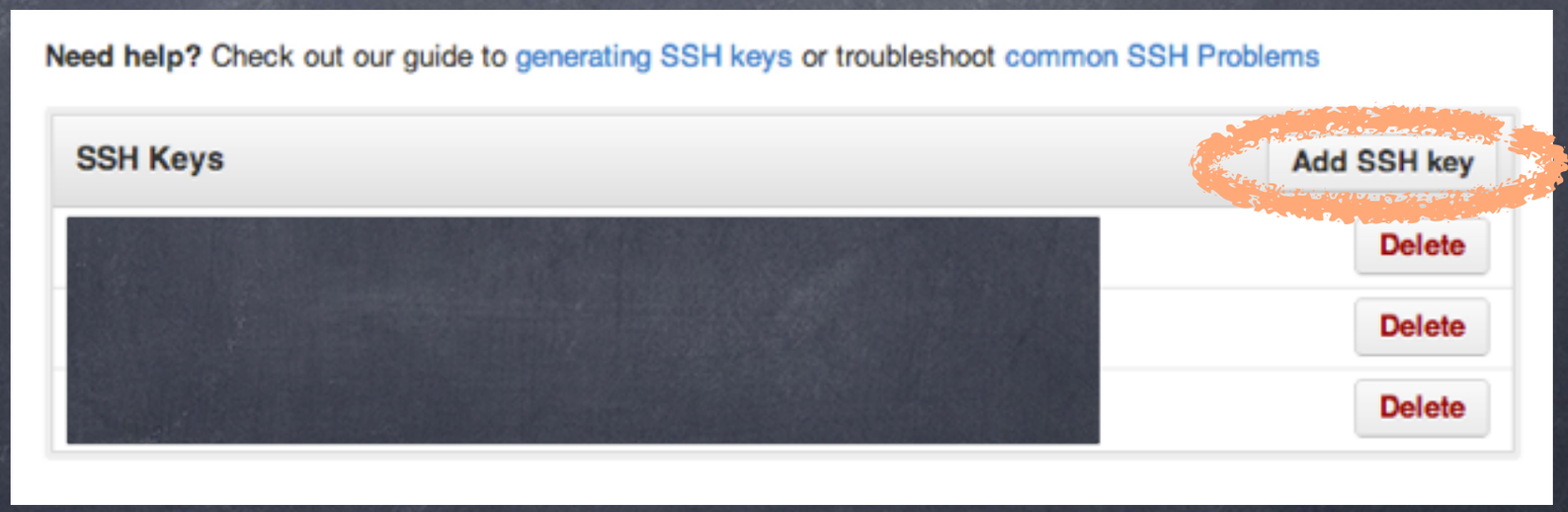


- find repository URL
  - e.g., from GitHub
- `git clone URL [name.git]`
- `git clone git@github.com:uwescience/myriad.git`

Git clone: HTTPS or SSH

- I prefer SSH w/keys
  - In particular, I don't even know my github password. Password reuse is bad!





Translate: svn update

- Fetch and merge all new changes from server
  - `git pull [remote] [branch]`
  - `git pull` =(usually)= `git pull origin master`
- Can do this separately (some would say you should)
  - `git fetch [origin] [branch]`
  - `git merge [origin/branch]`
- For more info: http://longair.net/blog/2009/04/16/git-fetch-and-merge/

Translate: svn commit

- Push to server all local changes, including:
  - tracked files that differ
  - added/deleted files
  - property changes (e.g., make executable)
  - more?

- git makes these two phases:
  - 1) stage changes with `git {add,rm,mv,commit}`
  - 2) generate one or more commits this way
  - 3) send them all to the server with `git push`

Translate: svn status

- What's happening locally

```
dhalperi@dhm temp % git status
# On branch master
# 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:   apple.tex
#
# Untracked files:
#   (use "git add <files>..." to include in what will be committed)
#
#       pear.tex
no changes added to commit (use "git add" and/or "git commit -a")
```

- Tells you which branch is active
- Staged changes: will be committed on `git commit`
- Unstaged changes: diff, but will not be committed
- Any untracked files (no version control)

svn revert: Here there be dragons!

- `svn revert`: undo any local, uncommitted changes
- `git reset --hard HEAD`: `svn revert`
- `git reset HEAD`: just unstages everything, leaves files alone
- `git stash`: take a diff and store it in a local temp place called the stash
- I generally prefer the latter combo, throwing changes away scares me.

svn revert:
Here there be dragons!

- `git revert <commit_id>`: create a new patch that is the reverse of <commit_id>, apply it to HEAD, and `git commit`
- Not `svn revert`!

Translation

SVN	GIT
checkout	clone
update	pull
commit	add &
	commit &
	push
branch	branch
diff	diff
log	log
status	status

SVN	GIT
revert	reset
--	revert