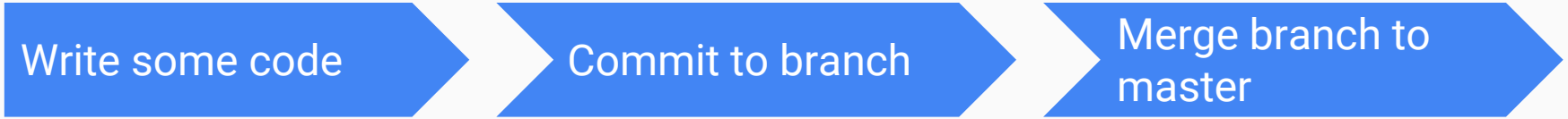
Git Rebase Work Flow

How to minimize time wasted when committing code

Typical Git Commit Flow



Self explanatory: new feature is defined and coding begins

Code is updated in the Git repo as a series of commits

Feature is completed, code is finalized, and it is brought back into master

Git usage based on number of developers

Solo Developer

- Commit directly to master, maybe a branch every now and then

- Almost no conflicts

- Does not require co-ordination between teammates, because you have none

- No wait time between merges

2-4 Developers

- Start to use branching strategy

- Start to see conflicts

- Some wait time between branch merges

- Git history begins to get complex but still manageable

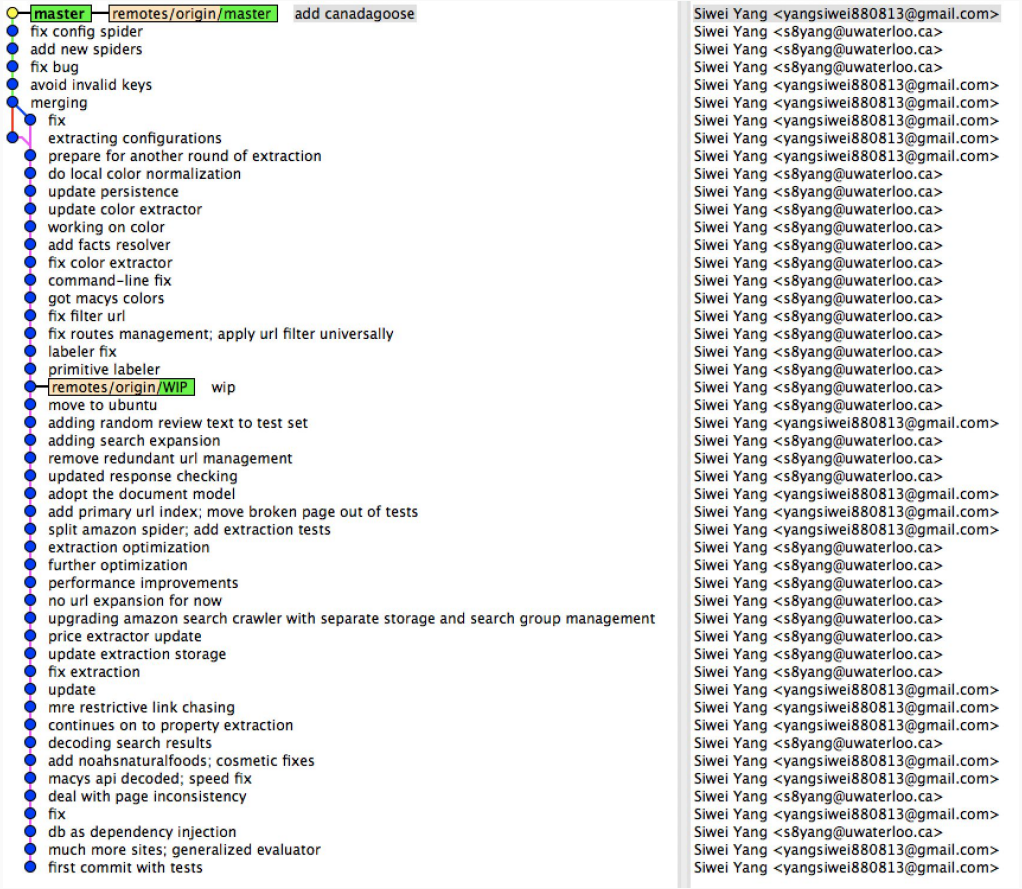
5+ Developers

- Lots of branches clutter git history

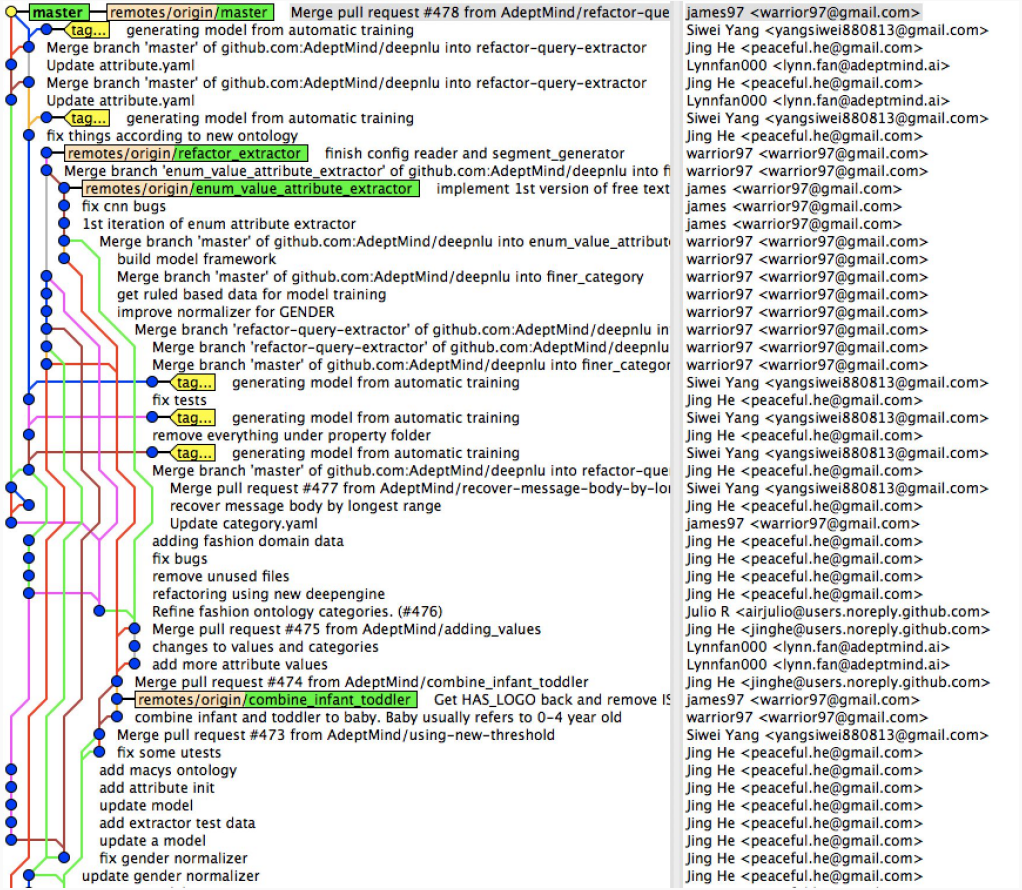
- Conflicts begin to rise quickly, and each teammate spends a lot of time resolving or maybe 1 developer dedicated to merging (usually senior)

- If one dev dedicated to merge, person becomes bottleneck and this process won't scale

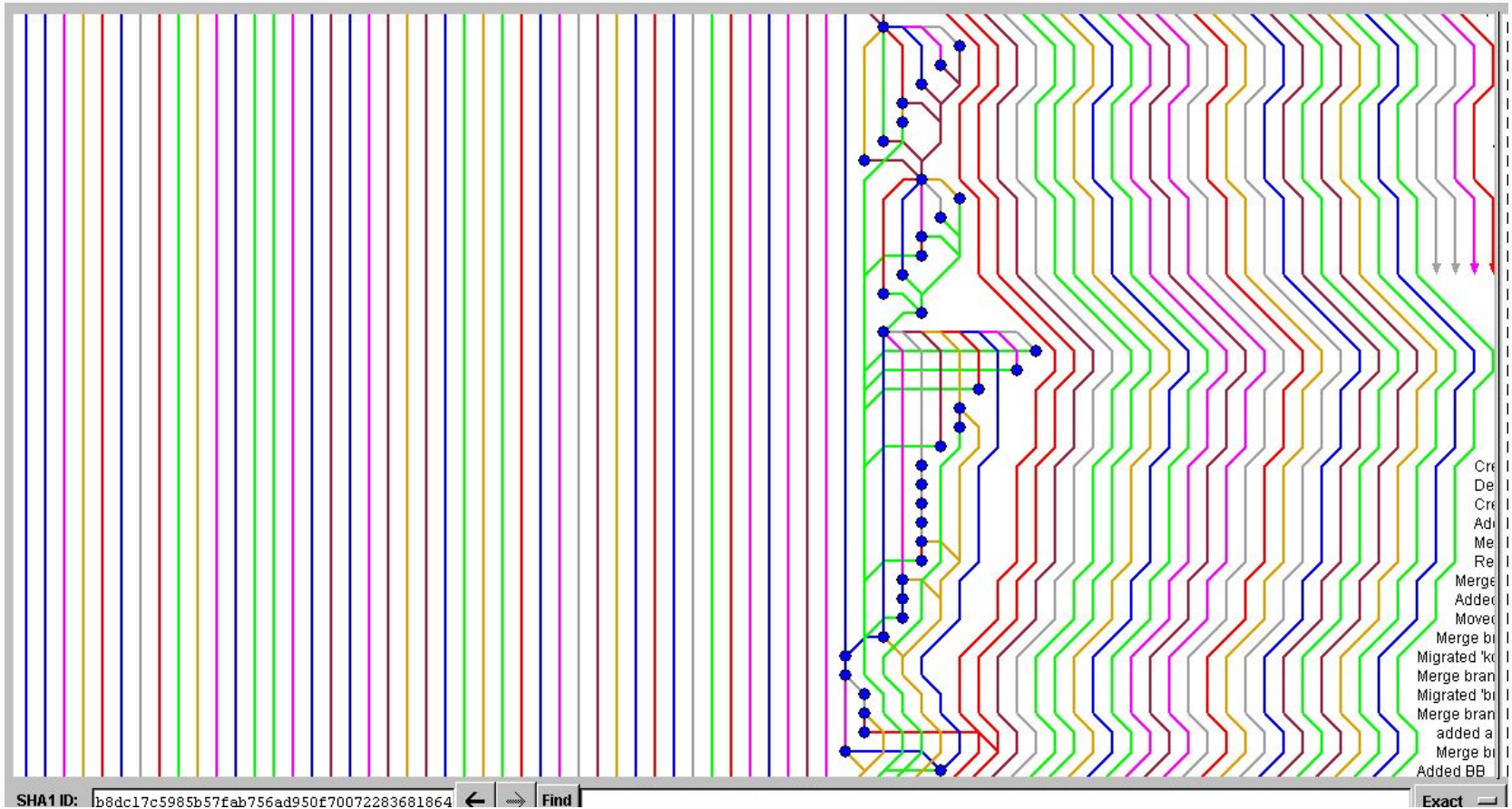
Solo Developer Git History -- `general_spider` Repository



Git Merge History for 2-4 Developers -- `deepnlu` repository

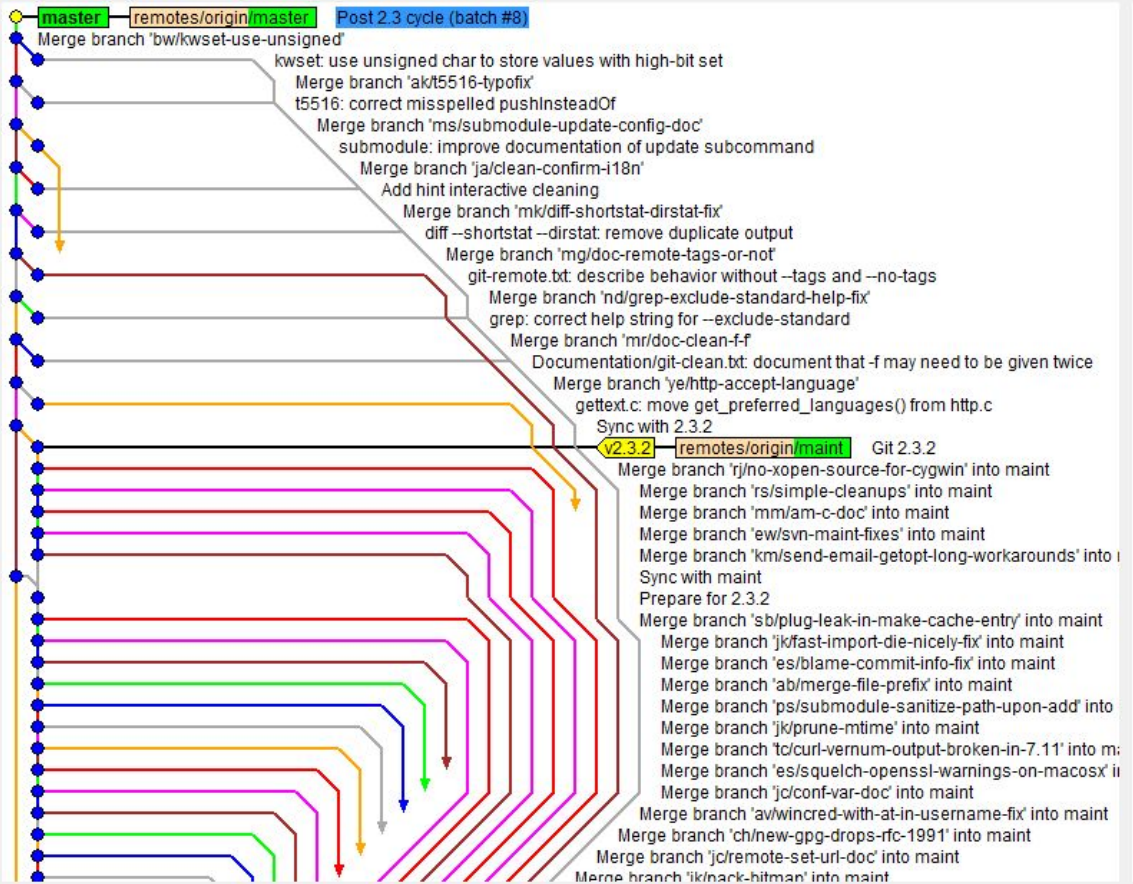


Potential Chaos Scenario With 5+ Developers



Wat.

Git Merge -- Time consumption approaches infinite



Solution

Use
`git pull --rebase origin master`
or
`git rebase -i origin/master`

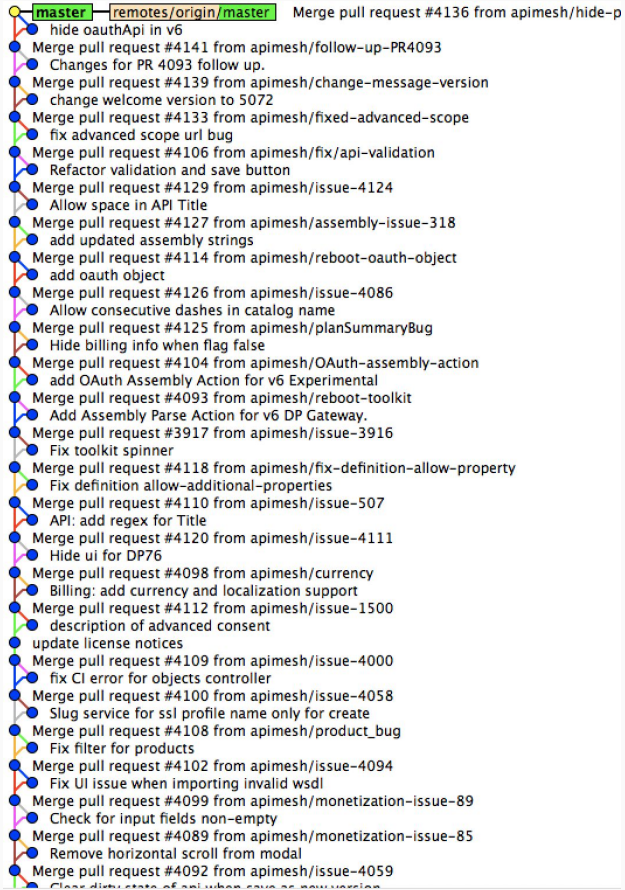
- Linear history based on merge

- Easy to handle product release, backporting, and versioning

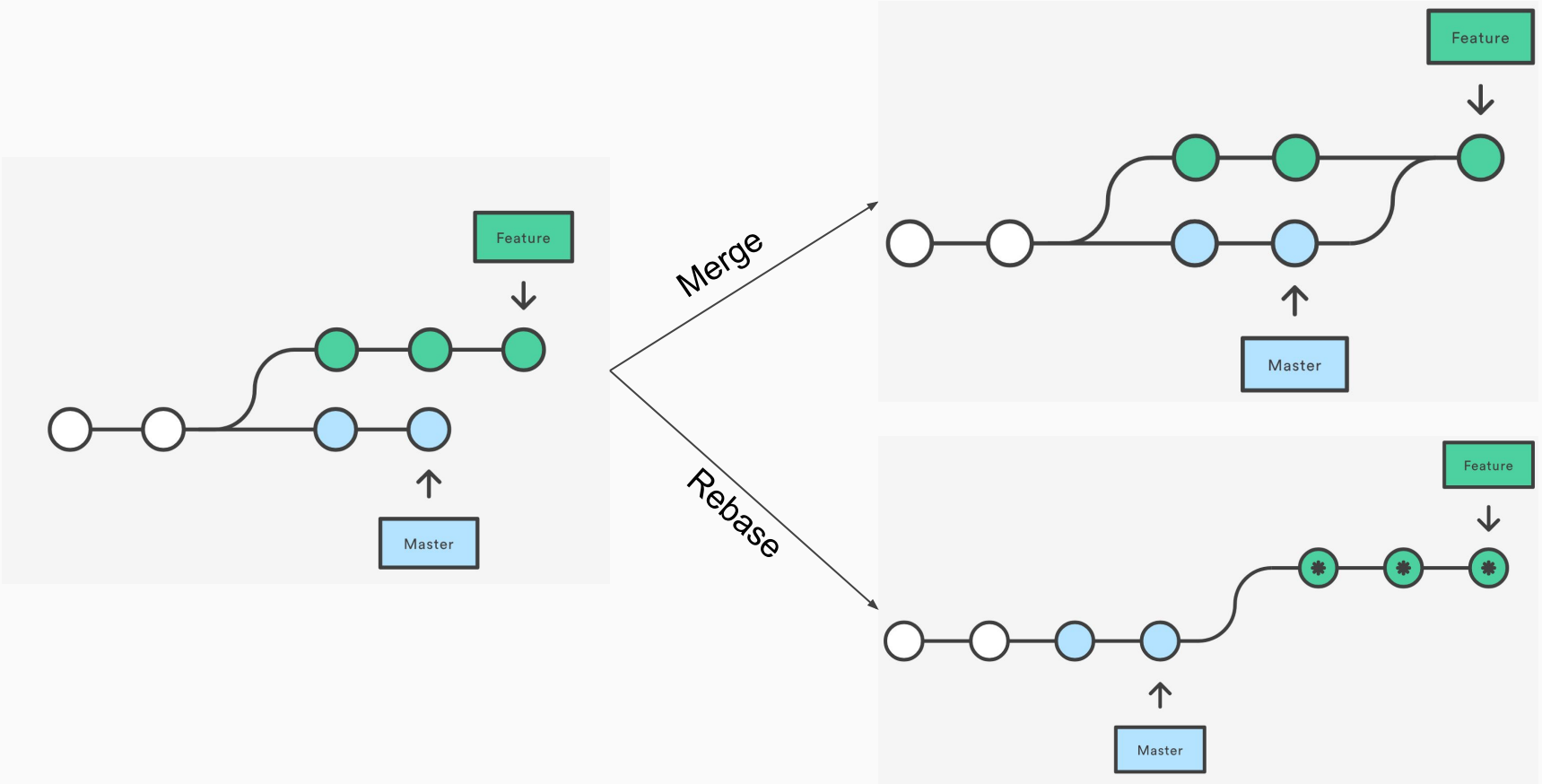
- Conflicts resolved by single developer, everyone else gets for free (most of the time)

- Scales to infinite developers, proven by Open-Source community

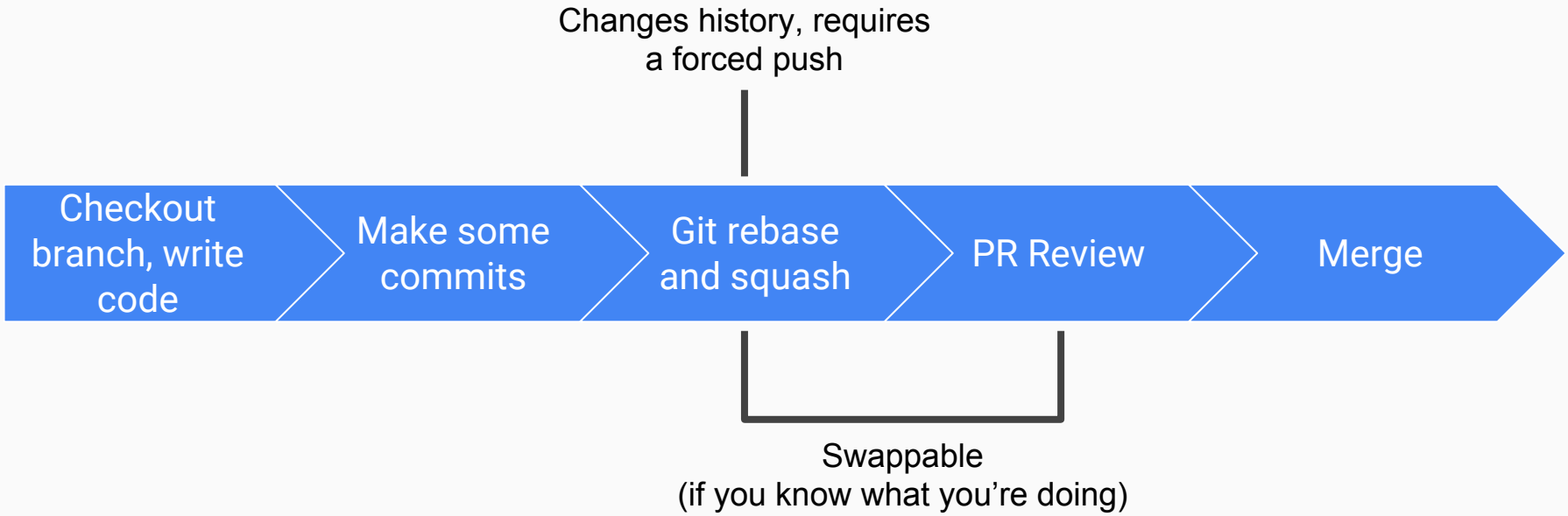
Git Rebase Work-Flow: A Team Effort



What is Git Rebase?



Git Rebase Flow



TEAM SPIRIT

(Do some demos?)