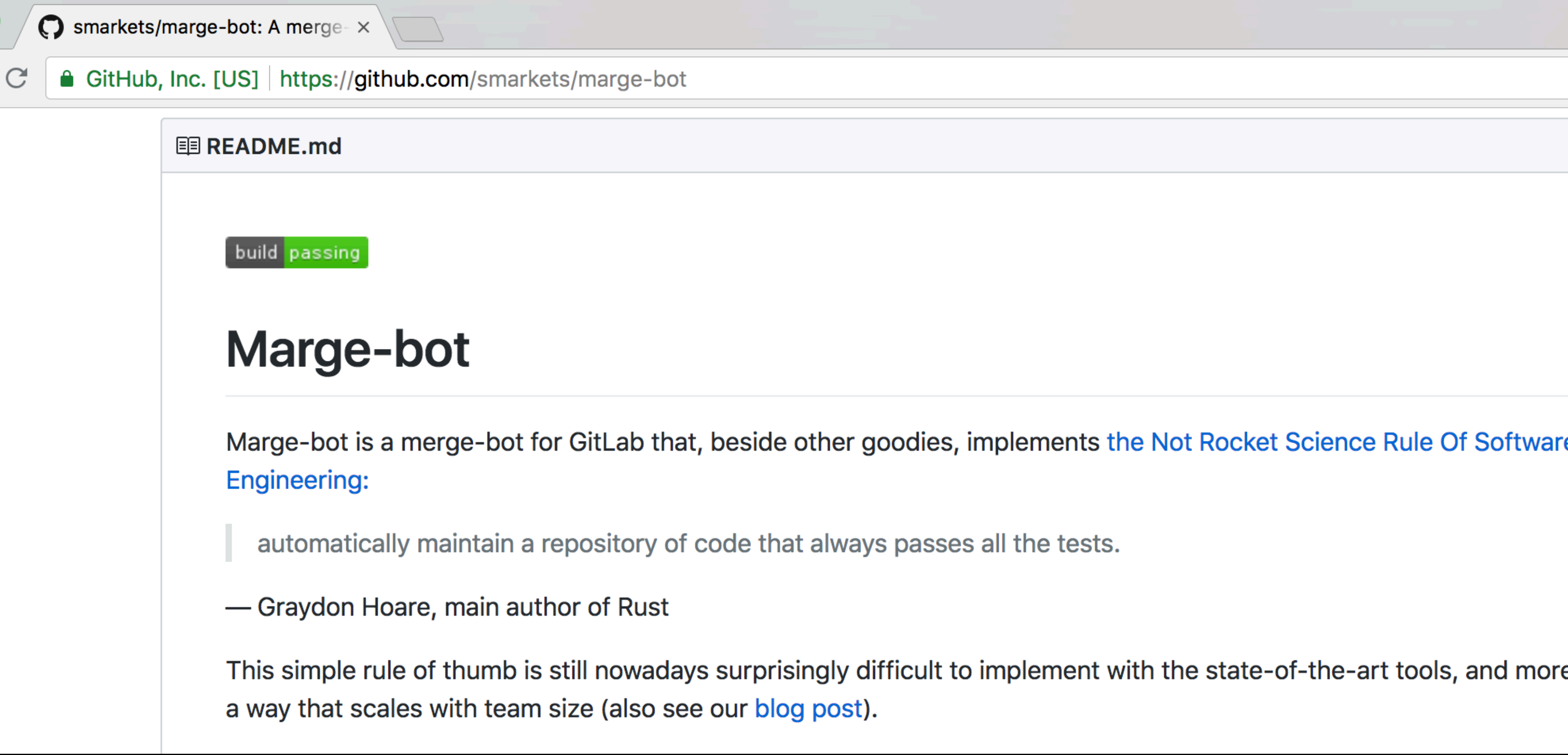
smarkets

Marge-bot: better Git’ing with python

Europython 2018

Alexander Schmolck
Mika Bostrom


smarkets/marge-bot: A merge

GitHub, Inc. [US] https://github.com/smarkets/marge-bot

README.md

build passing

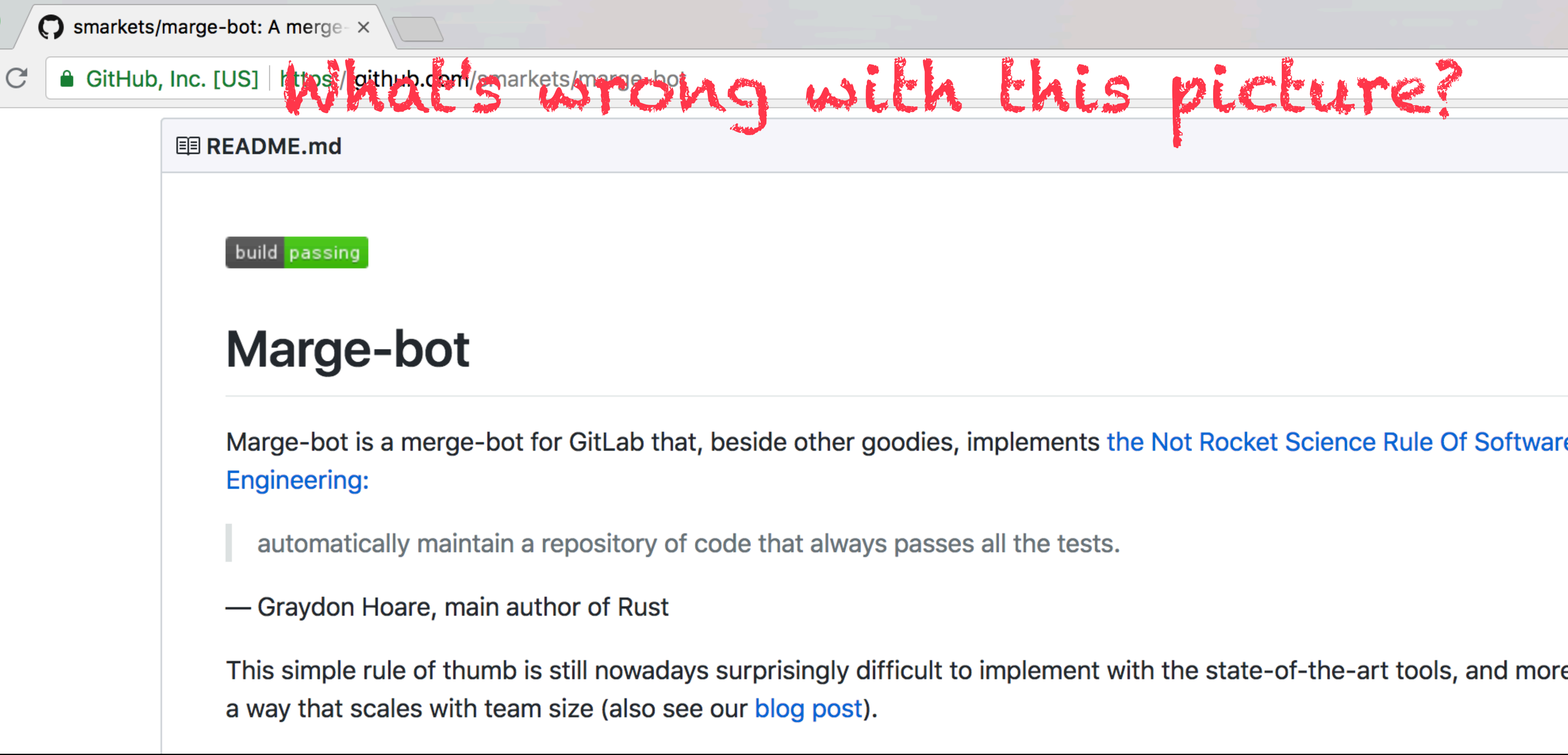
Marge-bot

Marge-bot is a merge-bot for GitLab that, beside other goodies, implements the Not Rocket Science Rule Of Software Engineering:

> automatically maintain a repository of code that always passes all the tests.

— Graydon Hoare, main author of Rust

This simple rule of thumb is still nowadays surprisingly difficult to implement with the state-of-the-art tools, and more a way that scales with team size (also see our blog post).


smarkets/marge-bot: A merge
GitHub, Inc. [US]
https://github.com/smarkets/marge-bot
README.md
build passing

What's wrong with this picture?

Marge-bot

Marge-bot is a merge-bot for GitLab that, beside other goodies, implements the Not Rocket Science Rule Of Software
Engineering:

> automatically maintain a repository of code that always passes all the tests.

— Graydon Hoare, main author of Rust

This simple rule of thumb is still nowadays surprisingly difficult to implement with the state-of-the-art tools, and more
a way that scales with team size (also see our blog post).

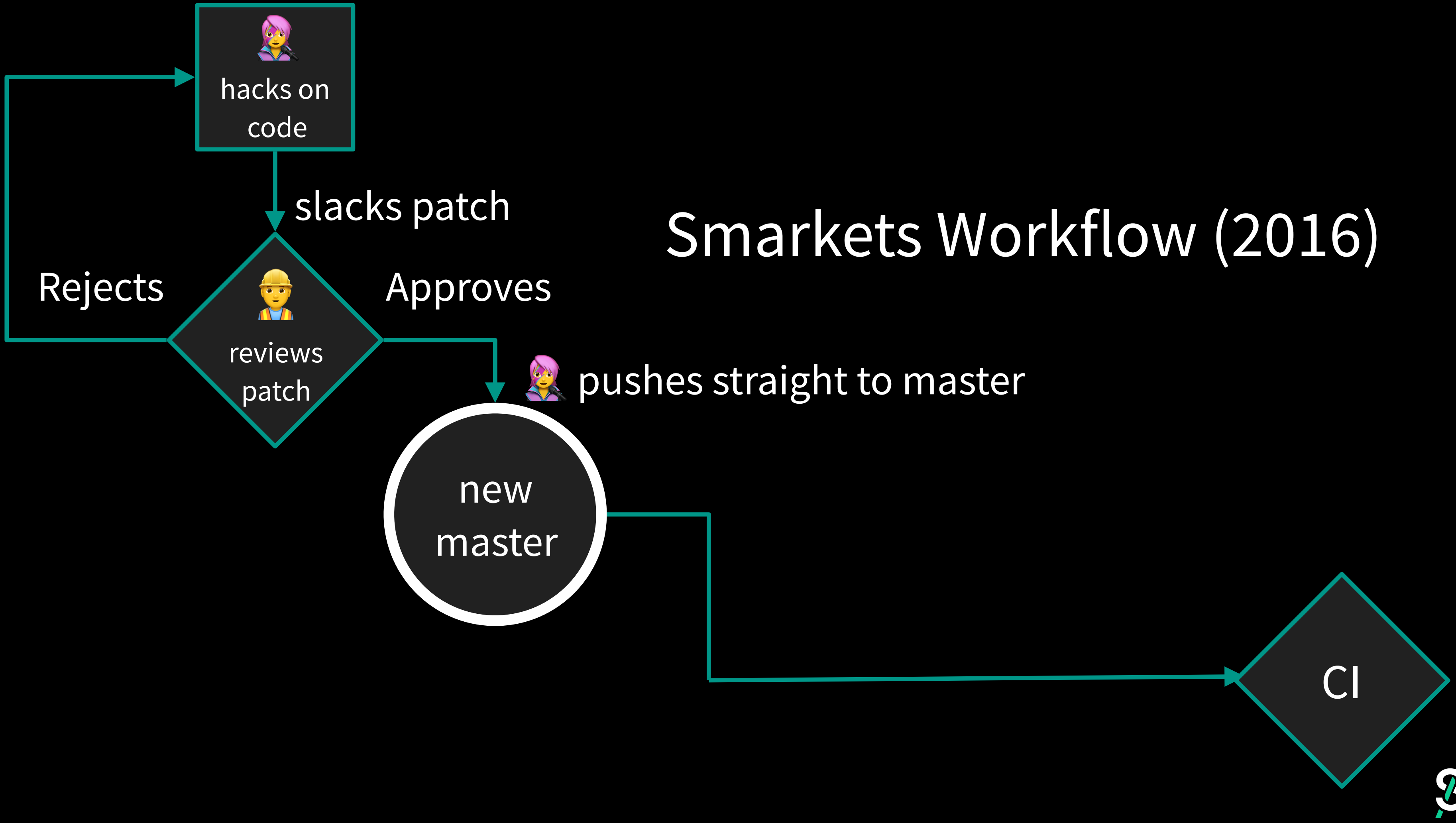
Outline

- The typical CI setup or going from green PRs to red master
- The fix, conceptually
- The fix, in practice: achieving familiarity, usability and scalability
- More nice stuff:
    - auditing, usable git bisect & git revert
- **Audience takeaways:**
    - Flexible & free out-of-the box solution for Gitlab (~3 mins setup)
    - Solutions, DIY or otherwise if you're not on Gitlab (e.g. Github)
    - Some useful Git workflow ~~and Python~~ patterns

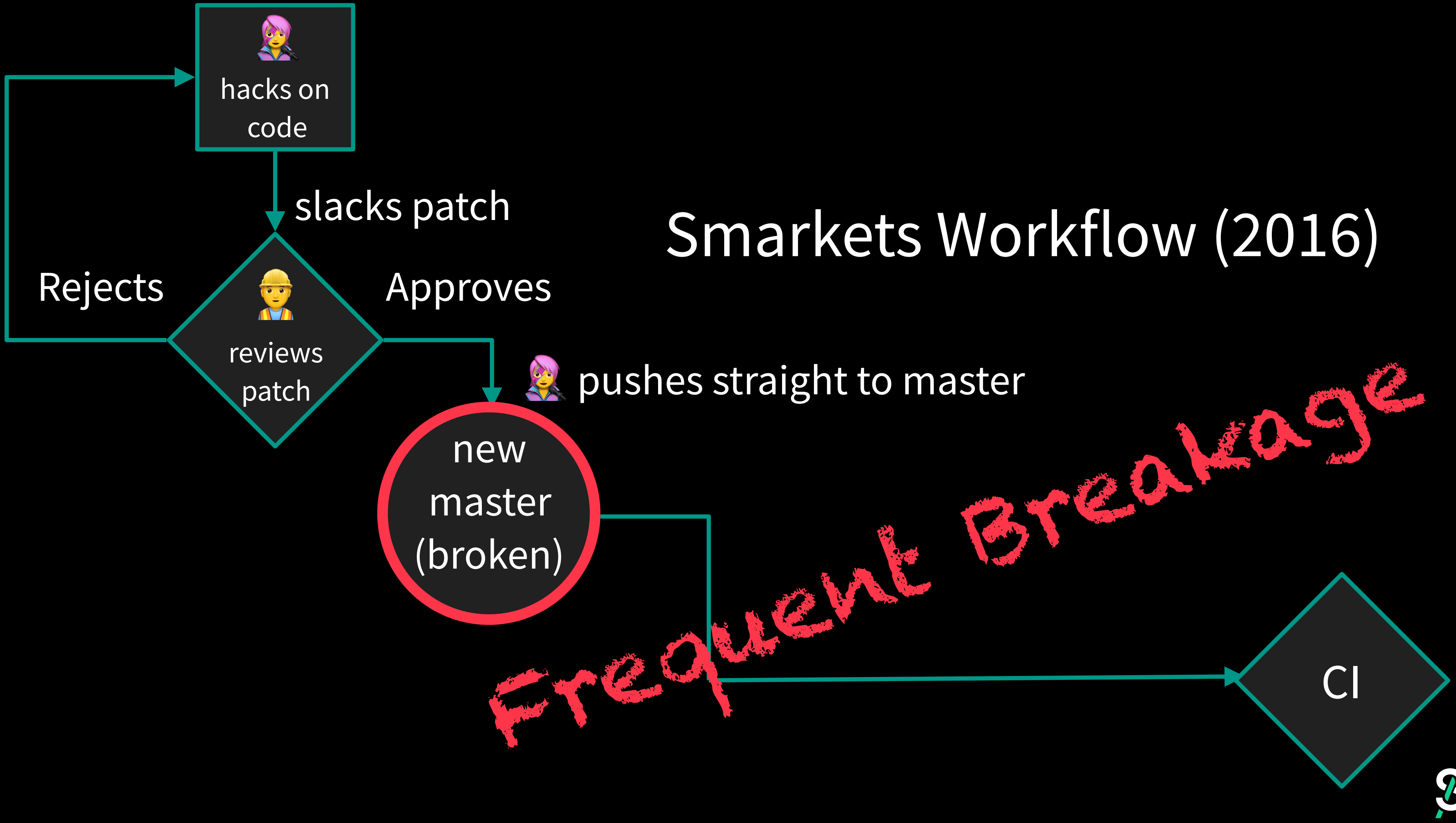
Why broken master is bad

- 🏝 subsequent PRs are built on sand
- 🚢 bad ships to prod more likely
- 🏃retracing your steps becomes hard (e.g. bisect)

Smarkets Workflow (2016)



Smarkets Workflow (2016)

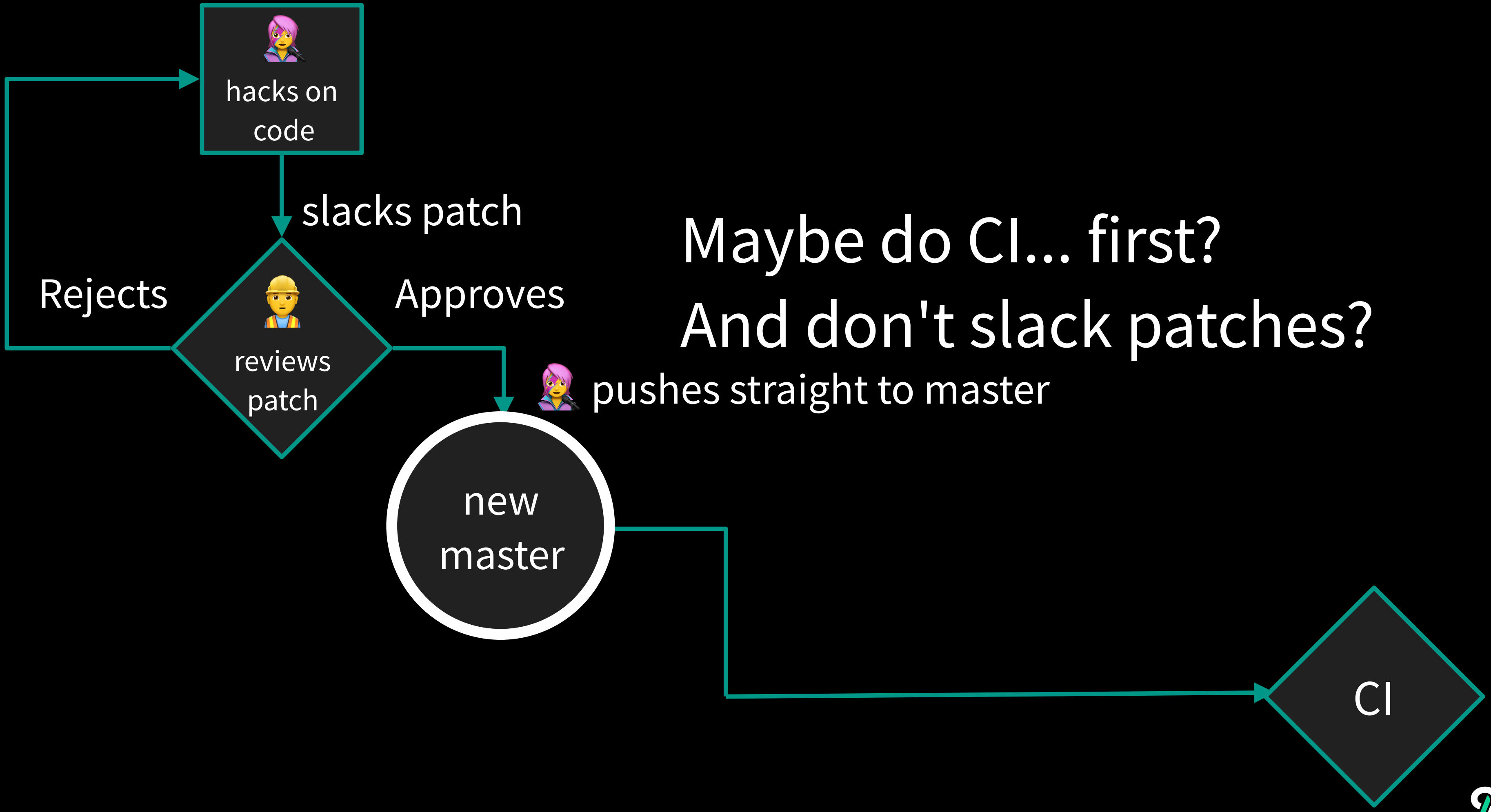


Master can be **broken** because

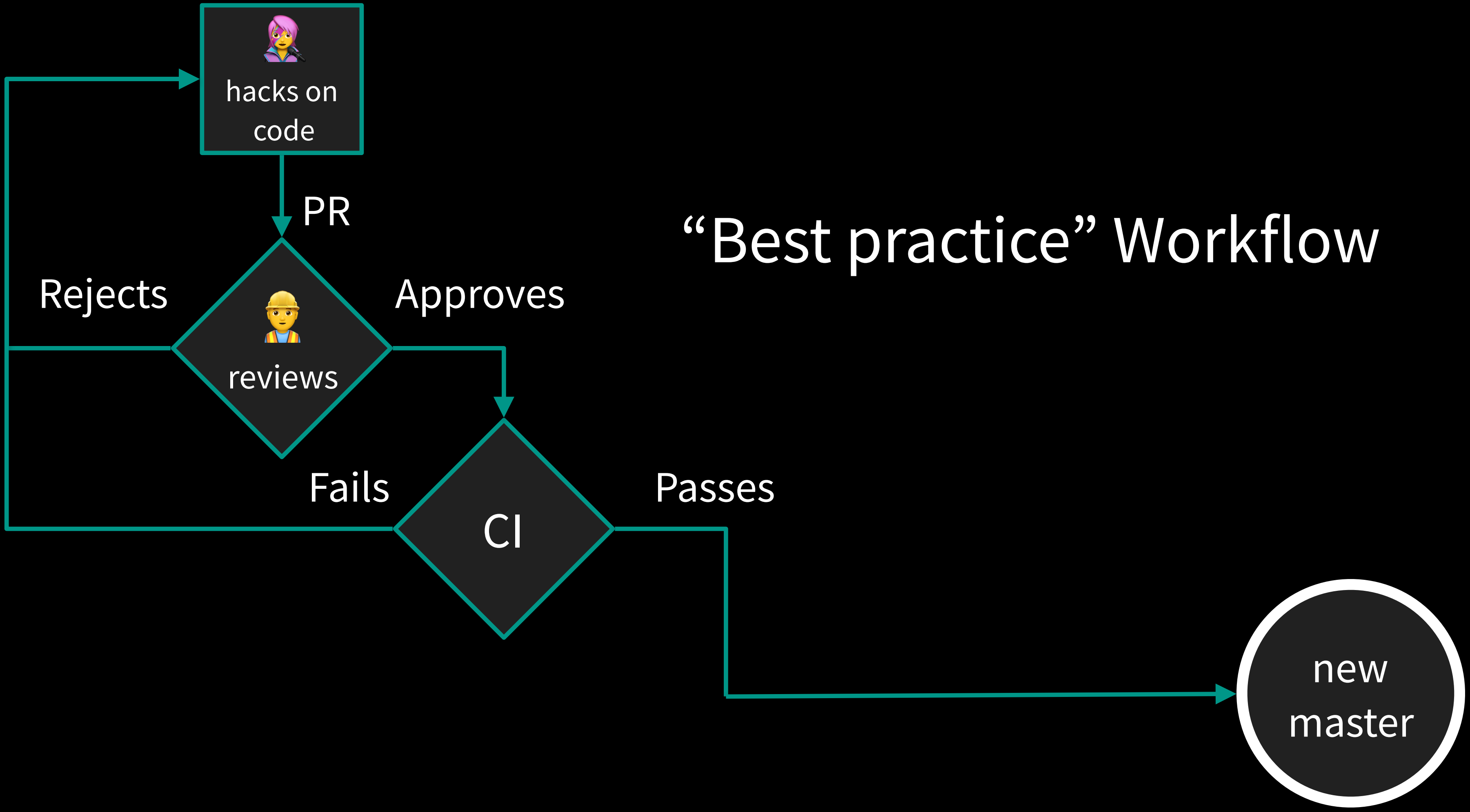
- bad workflows
  (spoiler: 🤖marge-bot can fix that for you!)
- “flaky builds”
  (non-determinism is harder to fix)

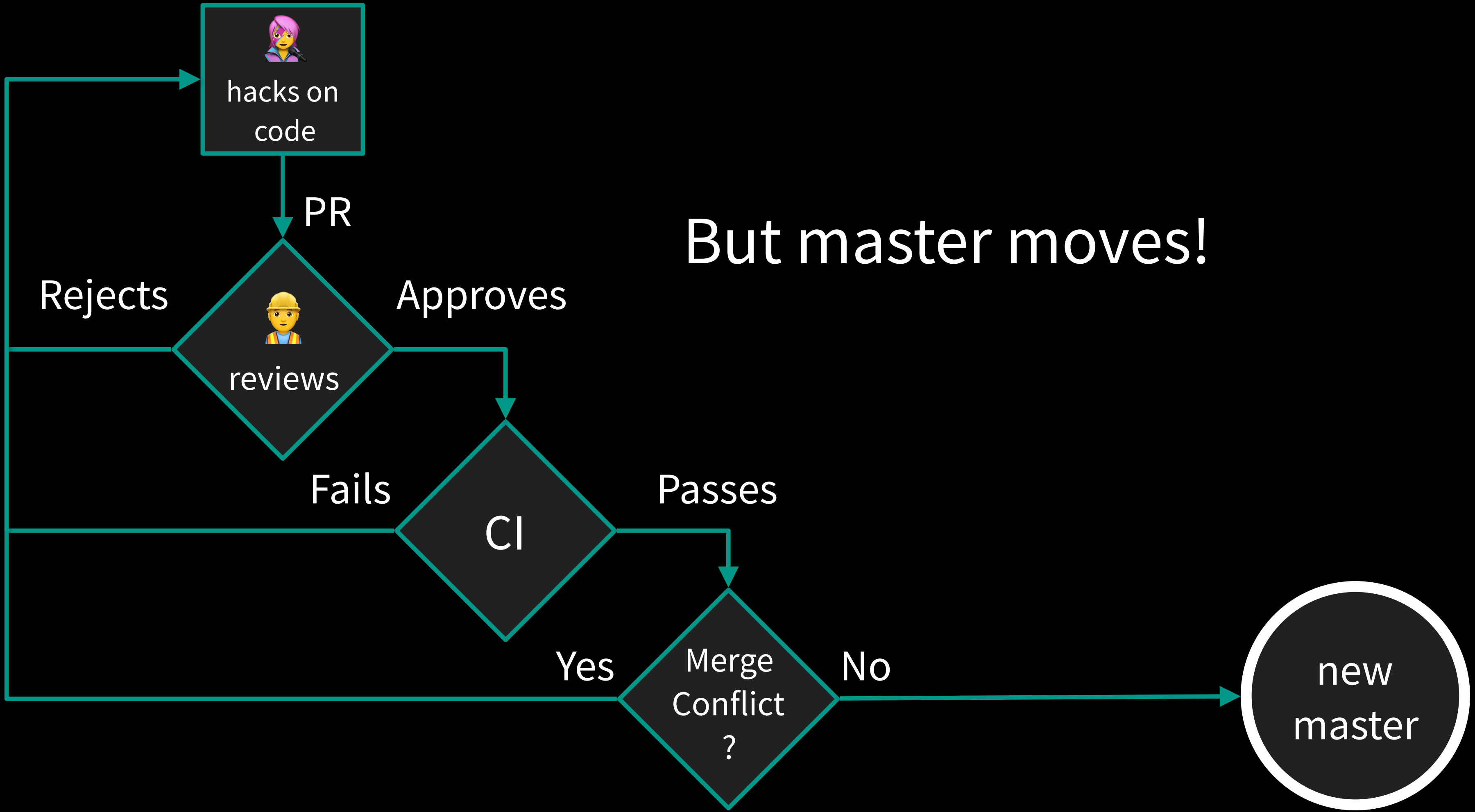
Maybe do CI... first?

And don't slack patches?



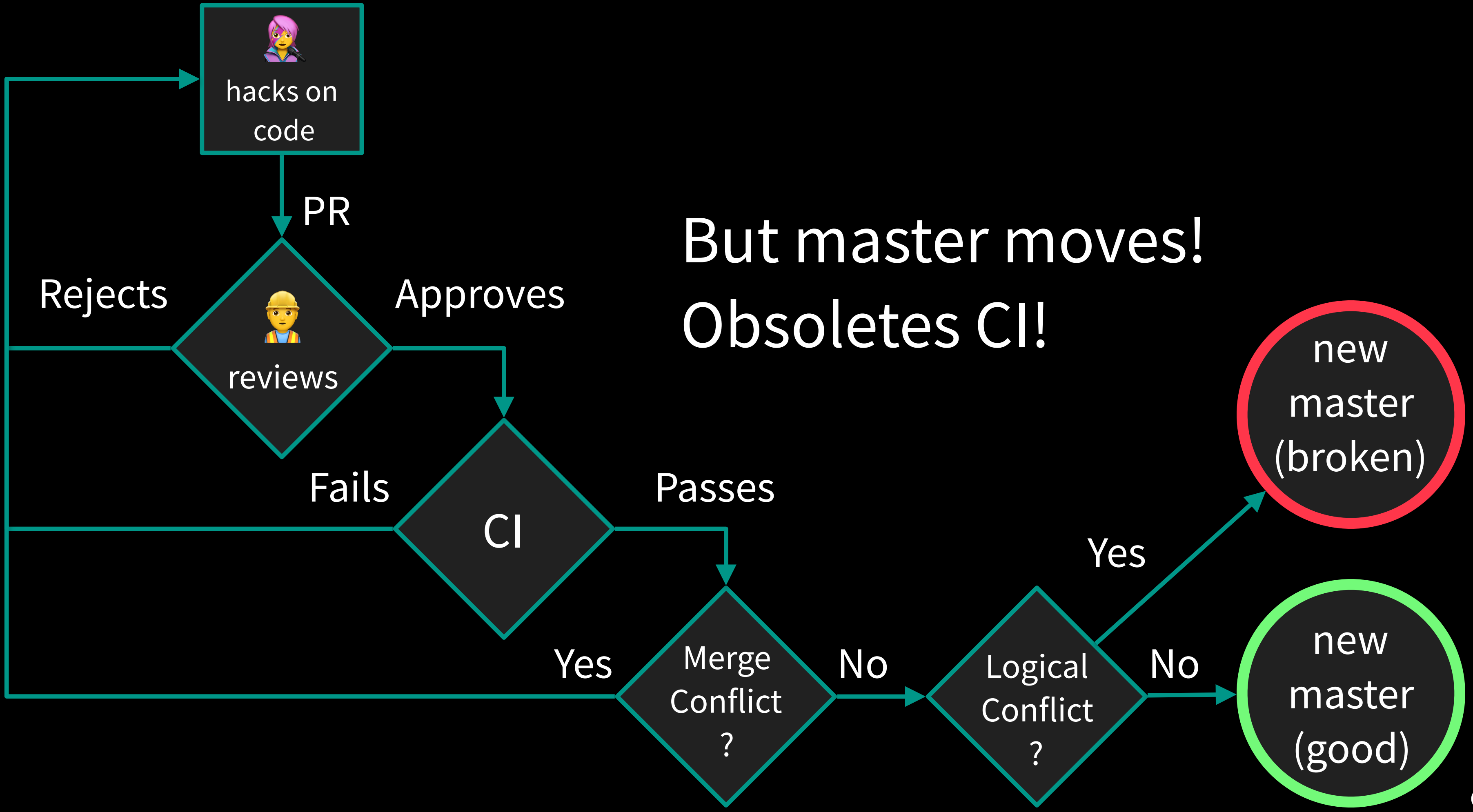
“Best practice” Workflow



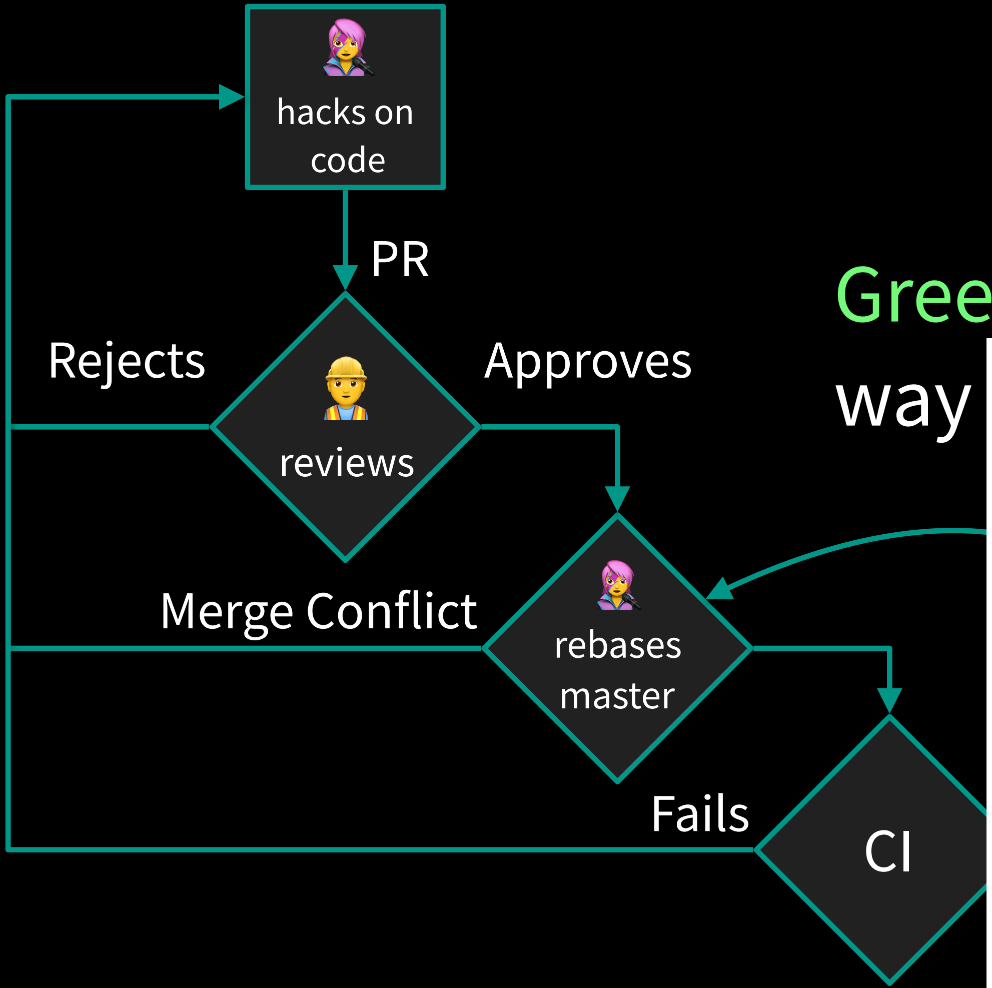


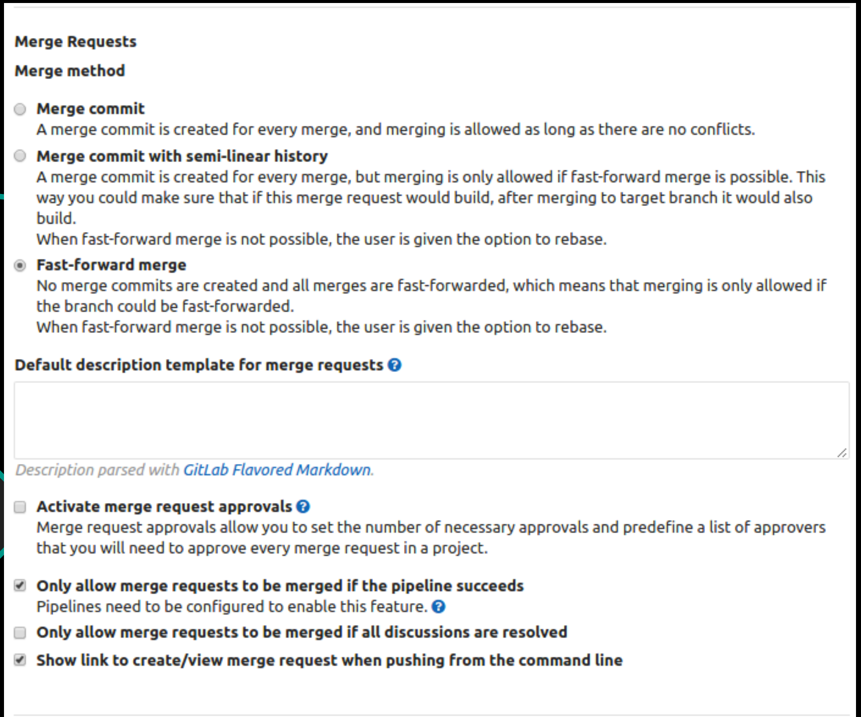
But master moves!

But master moves!
Obsoletes CI!



Green master the hard way





Merge Requests

Merge method

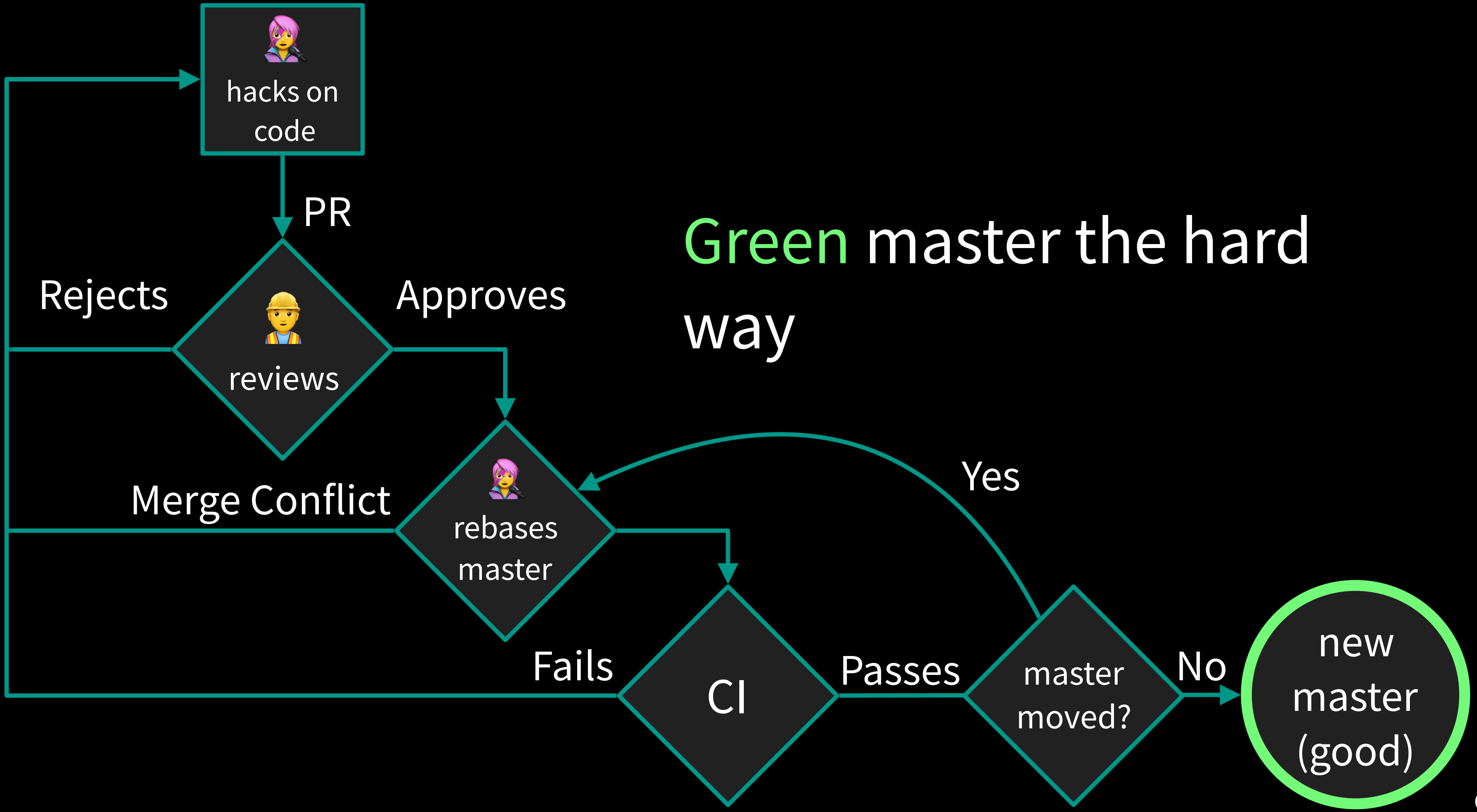
- [ ] **Merge commit**
  A merge commit is created for every merge, and merging is allowed as long as there are no conflicts.
- [ ] **Merge commit with semi-linear history**
  A merge commit is created for every merge, but merging is only allowed if fast-forward merge is possible. This way you could make sure that if this merge request would build, after merging to target branch it would also build.
  When fast-forward merge is not possible, the user is given the option to rebase.
- [✓] **Fast-forward merge**
  No merge commits are created and all merges are fast-forwarded, which means that merging is only allowed if the branch could be fast-forwarded.
  When fast-forward merge is not possible, the user is given the option to rebase.

Default description template for merge requests

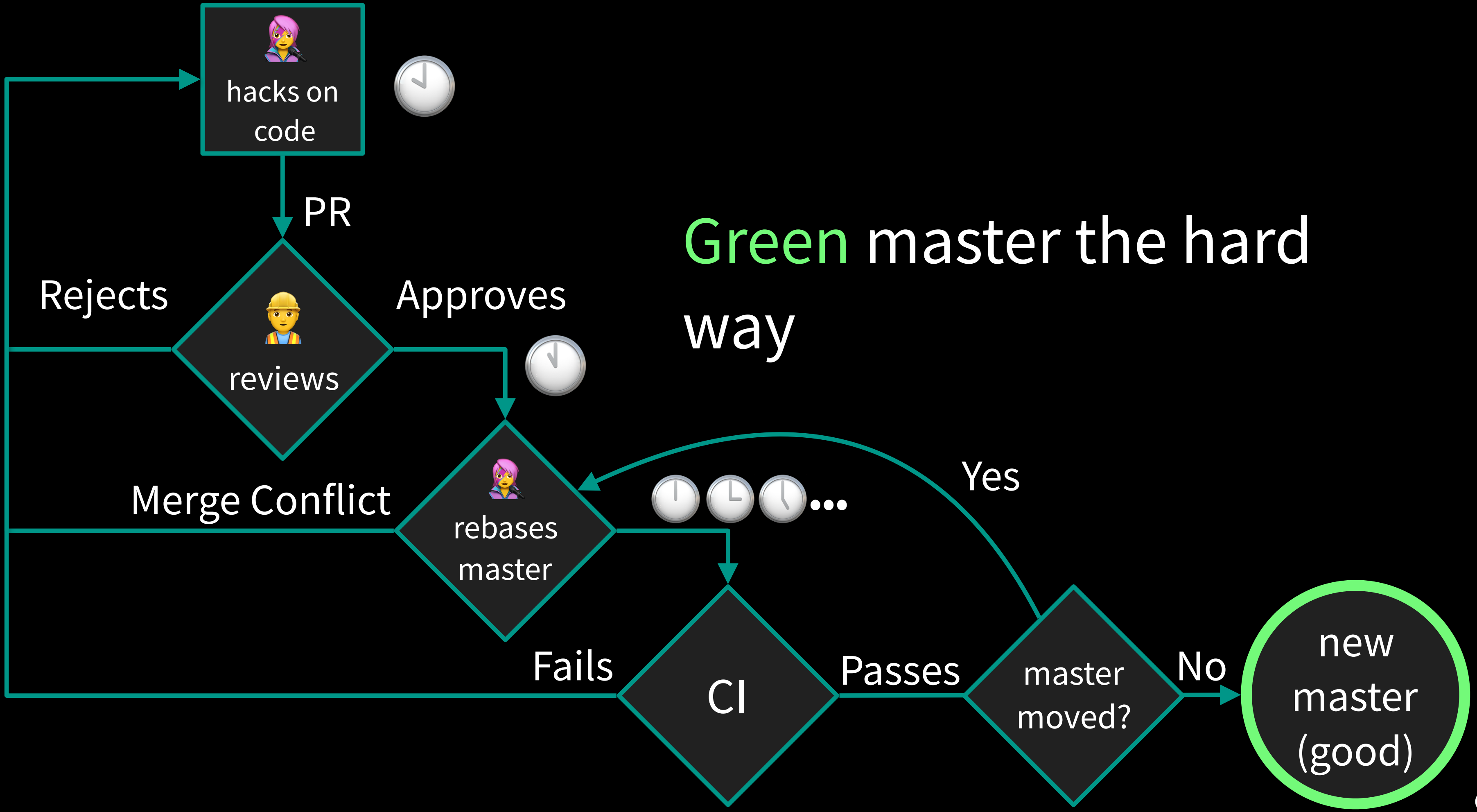
Description parsed with GitLab Flavored Markdown.

- [ ] **Activate merge request approvals**
  Merge request approvals allow you to set the number of necessary approvals and predefine a list of approvers that you will need to approve every merge request in a project.
- [✓] **Only allow merge requests to be merged if the pipeline succeeds**
  Pipelines need to be configured to enable this feature.
- [ ] **Only allow merge requests to be merged if all discussions are resolved**
- [✓] **Show link to create/view merge request when pushing from the command line**

Green master the hard way



Green master the hard way



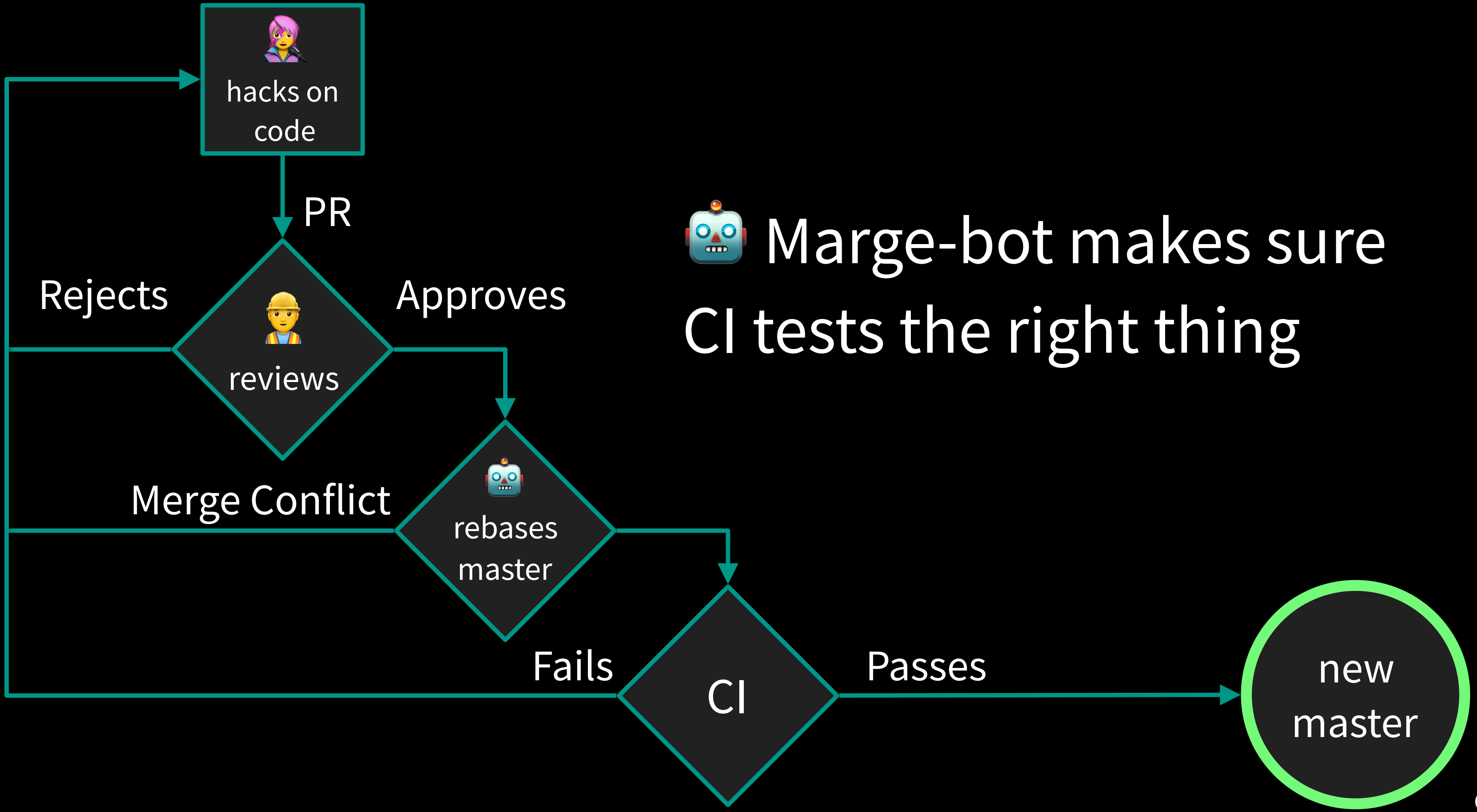
Sad 👩‍🎤

Sad

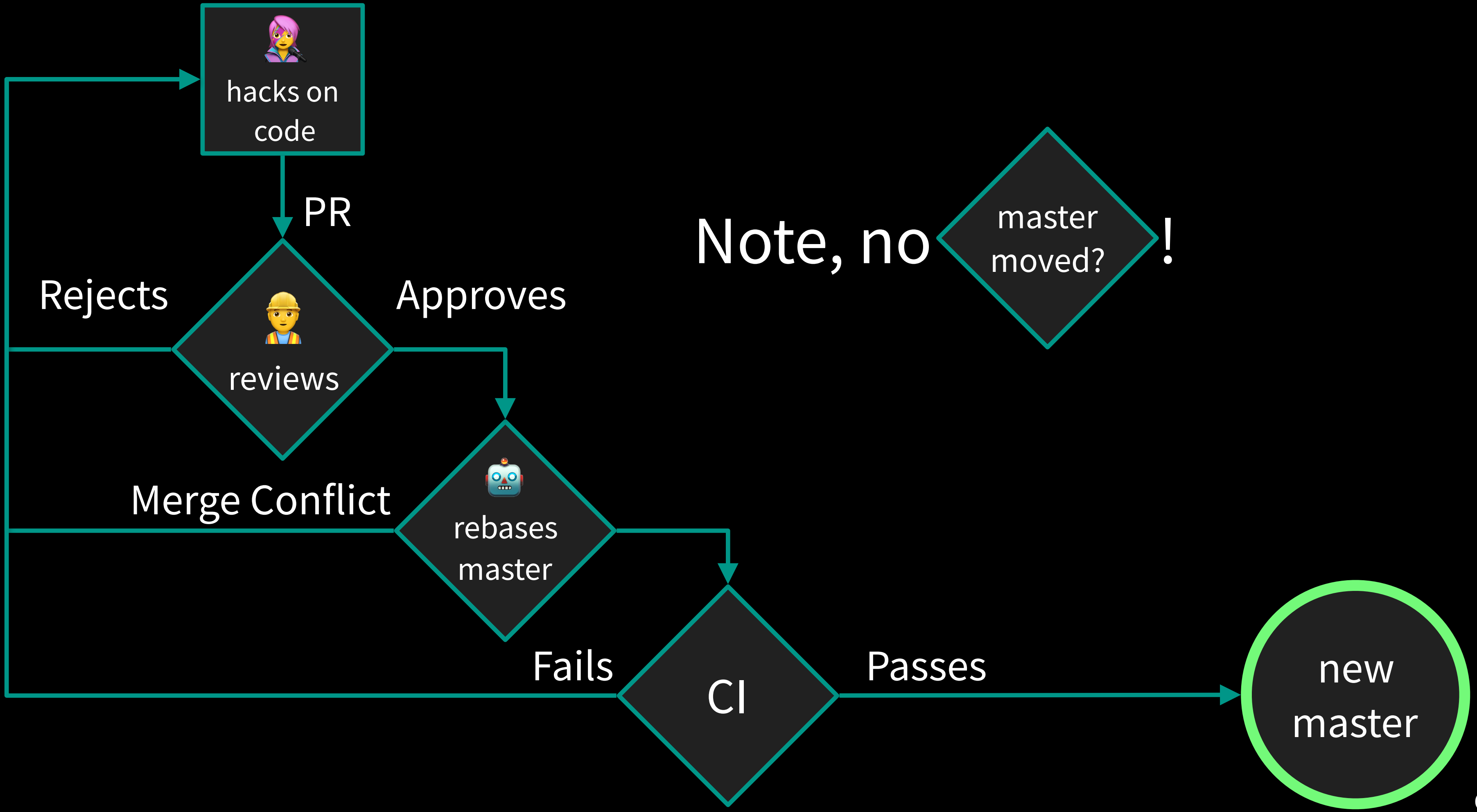
Sad 👩‍🎤

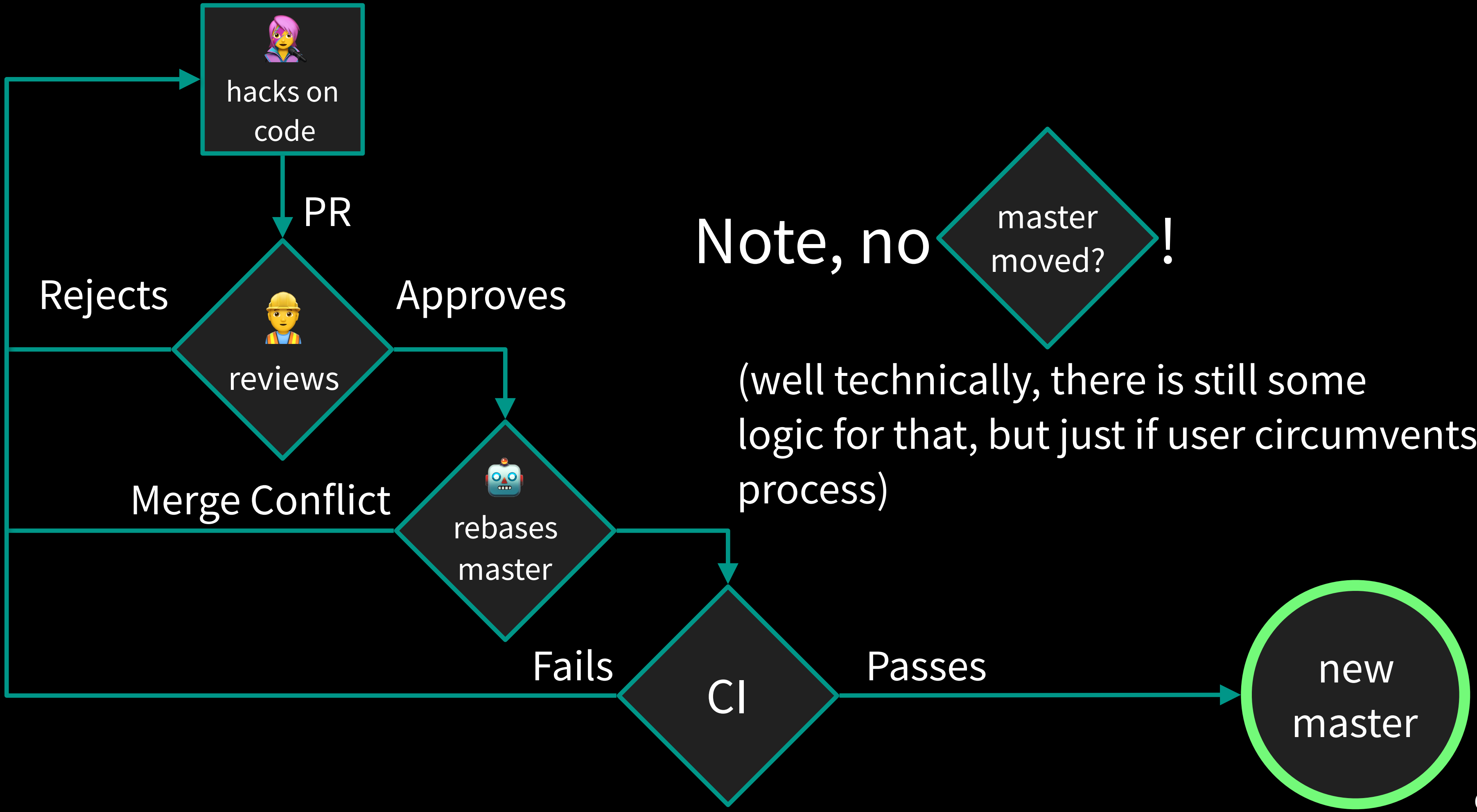
writes 🤖

This is the story of this 🤖



🤖 Marge-bot makes sure CI tests the right thing


hacks on code
PR
Rejects
reviews
Approves
Merge Conflict
rebases master
Fails
CI
Passes
new master
Note, no
master moved?
!



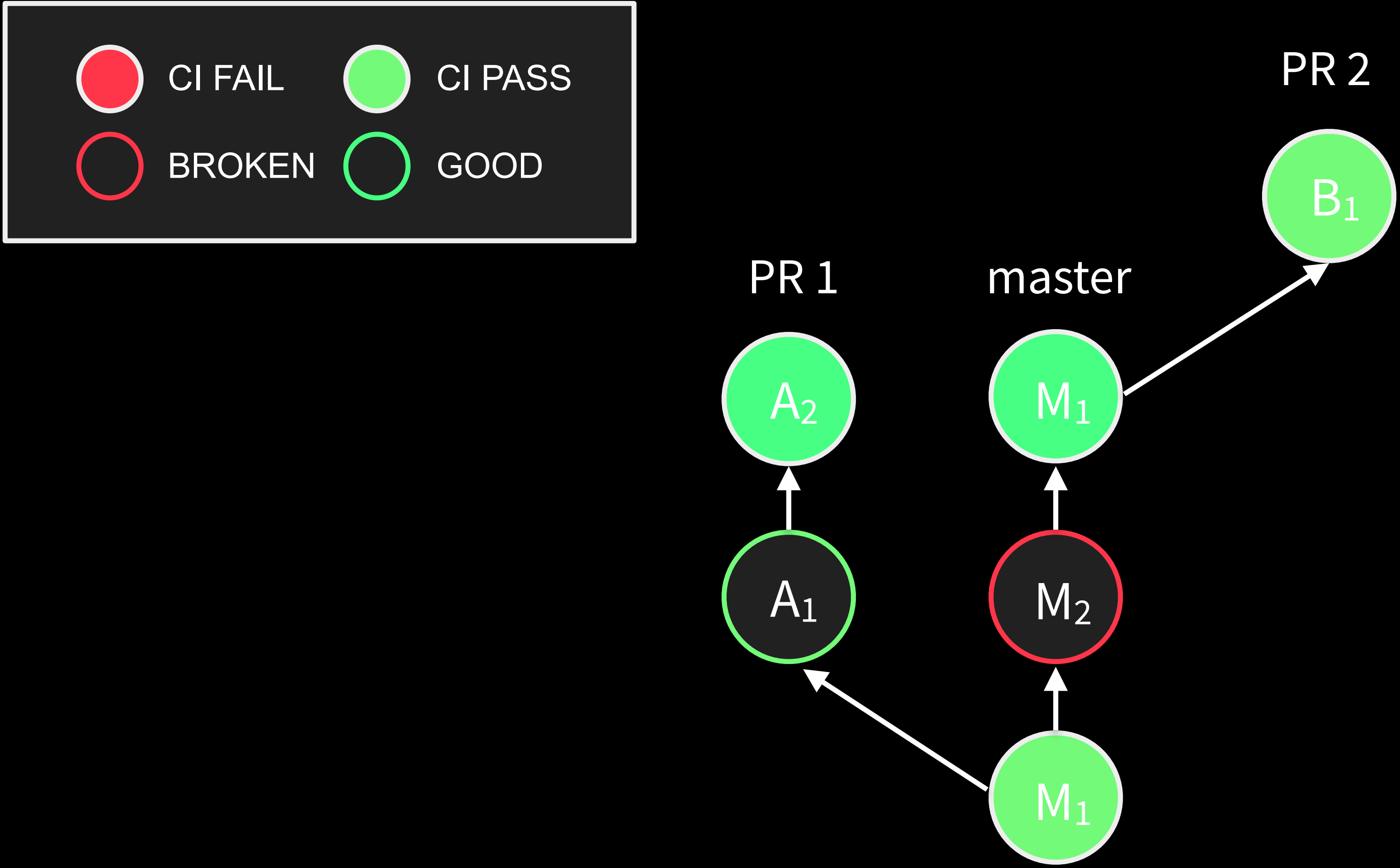
Note, no

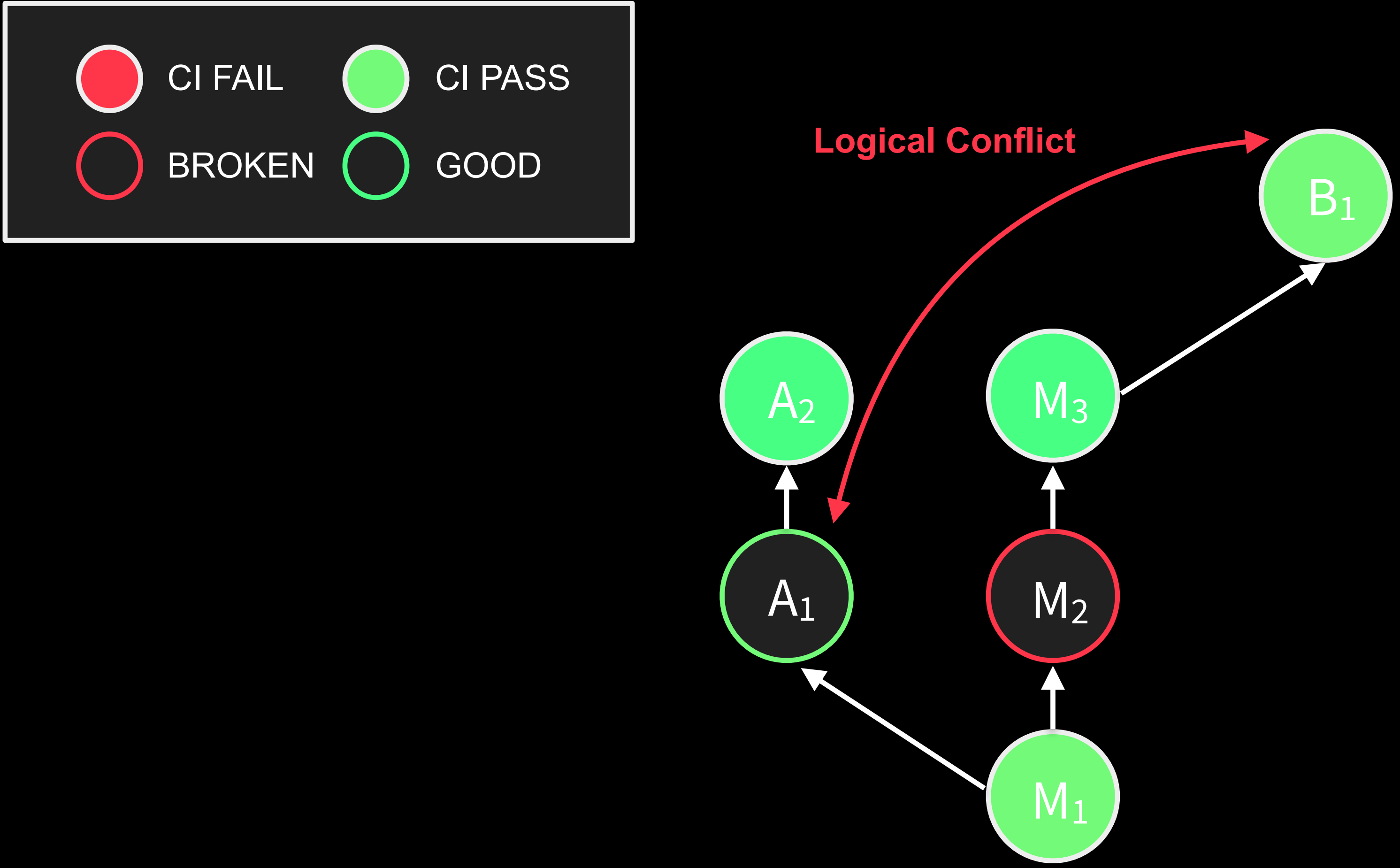
master moved?

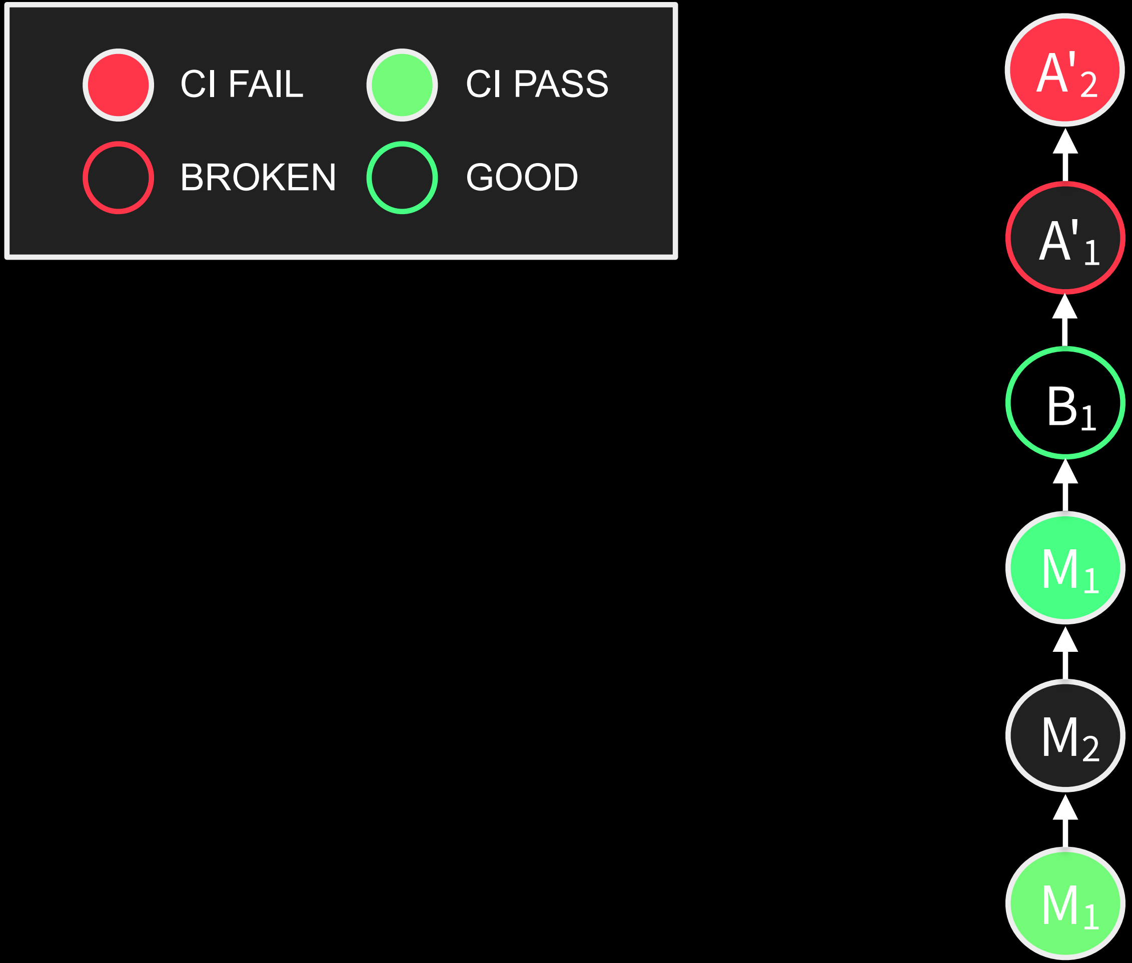
!

(well technically, there is still some logic for that, but just if user circumvents process)

How green PRs can break master


CI FAIL
CI PASS
BROKEN
GOOD
PR 2
B_1
PR 1
master
A_2
M_1
A_1
M_2
M_1


CI FAIL
CI PASS
BROKEN
GOOD
Logical Conflict
B_1
A_2
M_3
A_1
M_2
M_1


CI FAIL
CI PASS
BROKEN
GOOD
A'_2
A'_1
B_1
M_1
M_2
M_1

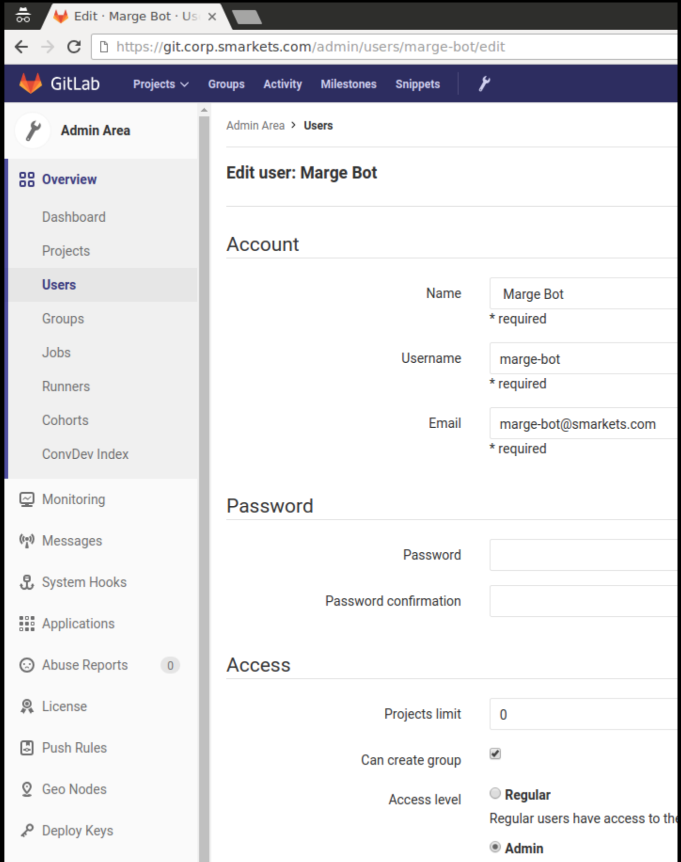
Going from Green PRs to Red master

- **New wine into old skins**: PR 1 changes a function’s API and all its call-sites, PR 2 introduces a new call site
- **Outdated coverage**: PR1 improves test coverage, PR2 changes the API
- **Fragile Baseclass**: PR1 makes internal changes to how Klass is implemented; PR2 adds some functionality to SubKlass that breaks because e.g. Klass.f no longer calls Klass.g internally.

None of these will cause **merge** conflicts or (feature-branch) CI failures; you’ll find out there is a **logical** conflict after successful merge to master.

Running Marge-bot (in 3mins or less)

1. Create marge-bot ssh key and gitlab account/token



Edit user: Marge Bot

Account

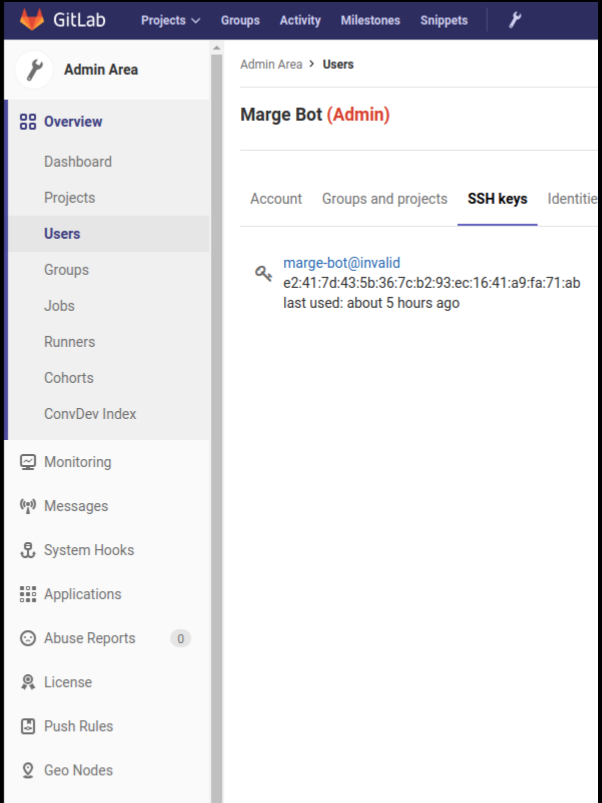
Name	Marge Bot * required
Username	marge-bot * required
Email	marge-bot@smarkets.com * required

Password

Password	
Password confirmation	

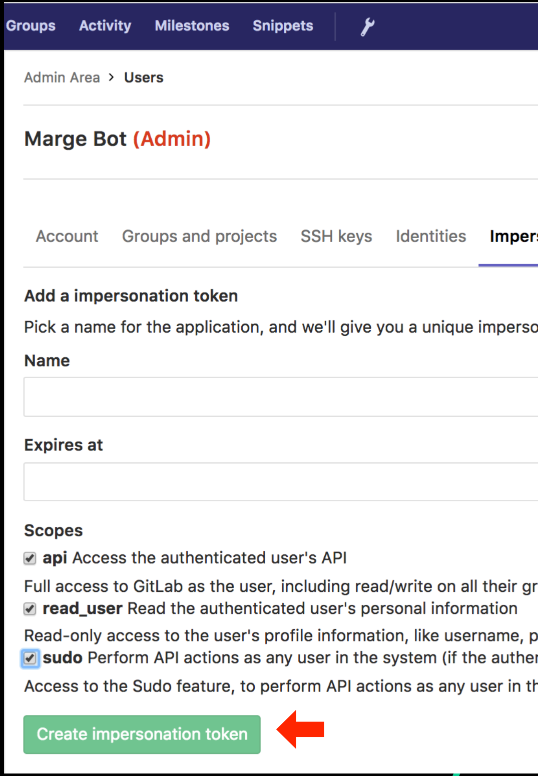
Access

Projects limit	0
Can create group	✓
Access level	Regular Regular users have access to the [✓ Admin]



Marge Bot (Admin)

marge-bot@invalid
e2:41:7d:43:5b:36:7c:b2:93:ec:16:41:a9:fa:71:ab
last used: about 5 hours ago



Marge Bot (Admin)

Add a impersonation token

Pick a name for the application, and we'll give you a unique imperso

Name

Expires at

Scopes

✓ **api** Access the authenticated user's API

Full access to GitLab as the user, including read/write on all their gr

✓ **read_user** Read the authenticated user's personal information

Read-only access to the user's profile information, like username, p

✓ **sudo** Perform API actions as any user in the system (if the auther

Access to the Sudo feature, to perform API actions as any user in th

Running Marge-bot (in 3mins or less)

1. Create marge-bot ssh key and gitlab account/token
2.

```
docker run --restart=on-failure \
  --env-file=<(
     echo MARGE_AUTH_TOKEN="$(cat marge-bot.token)";
     echo MARGE_SSH_KEY="$(cat marge-bot-ssh-key)") \
  smarkets/marge-bot \
  --gitlab-url='http://your.gitlab.instance.com'
```

Running Marge-bot (in 3mins or less)

1. Create marge-bot ssh key and gitlab account/token
2.

```
docker run --restart=on-failure \
  --env-file=<(
     echo MARGE_AUTH_TOKEN="$(cat marge-bot.token)";
     echo MARGE_SSH_KEY="$(cat marge-bot-ssh-key)") \
  smarkets/marge-bot \
  --gitlab-url='http://your.gitlab.instance.com'
```

3. Add marge-bot as a user (dev or master) to your project(s)

Demo of assigning PR to marge-bot

Conceptual Fix is simple

- Maintain a queue of pull requests
- Before merging a PR
  - rebase latest master into PR branch
  - wait for CI to pass
- Profit!
  - master will be always green (unless some tests are flaky)

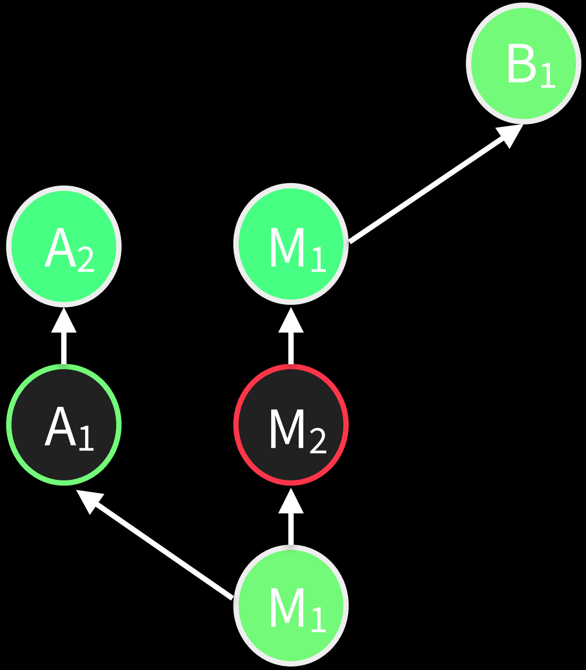
Making it work practice (Usability/Familiarity)

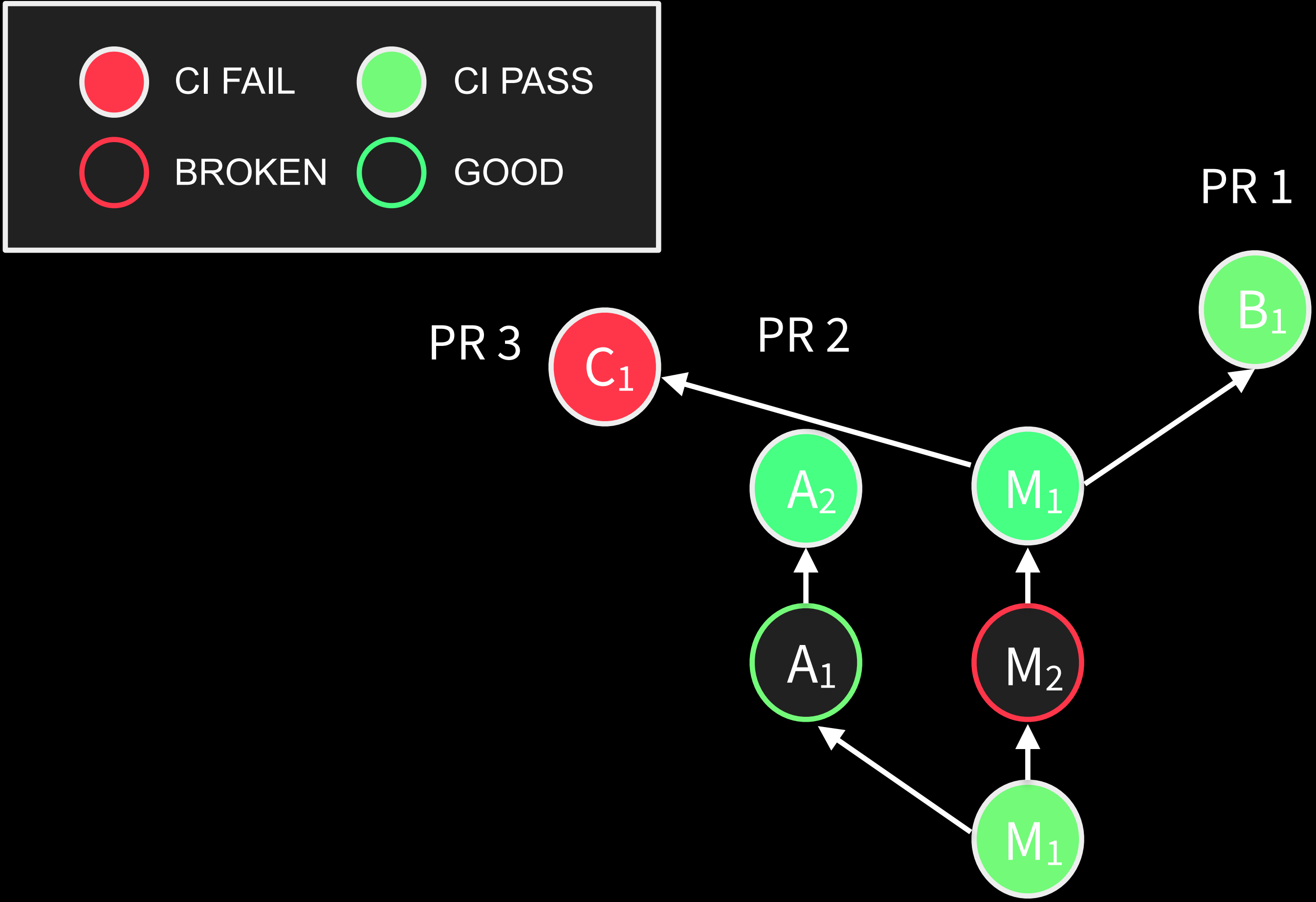
- **Familiarity**
  - use normal gitlab flow, but assign to Marge-bot rather than pressing “merge after CI passes” (will make sure CI passed and branch has been reviewed)
  - Fair amount of behind-the-scenes work to bend Gitlab API to our will (not designed with this use case in mind; race conditions, need)
- **Usability**
  - Gitlab user name is “` marge-bot`” (initial space, sorts first in list of users, so quick to assign to)
  - Marge leaves comments telling you if there is a problem (CI failed, no approval, conflict...)
  - we have a Slack channel that shows the Merge queue maintained by Marge (so place in queue/ETA easy to find

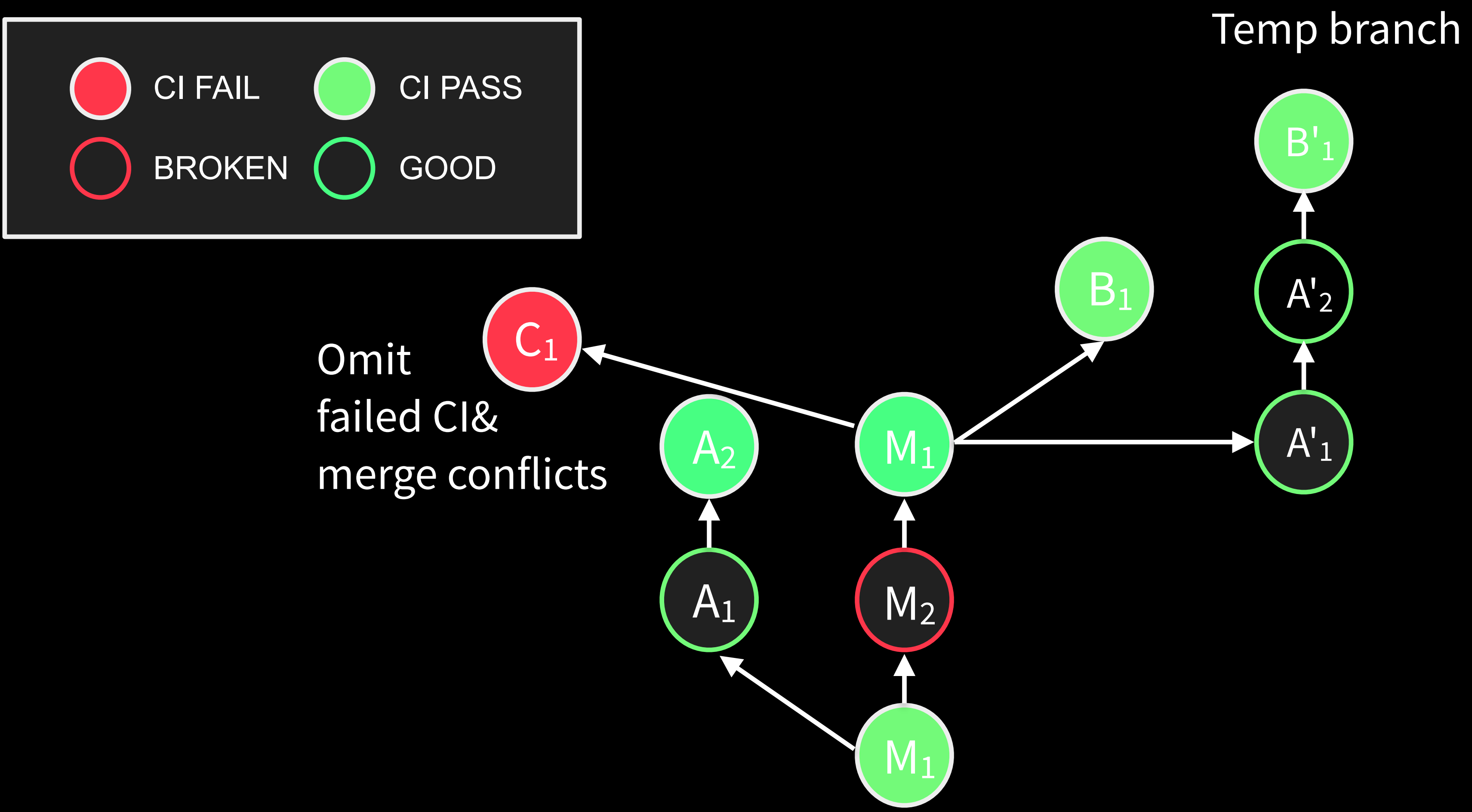
Making it work practice (Scalability)

- **Scalability** (via Batching)
  - rebasing all open PRs on merge/rebase to master and re-running CI works fine for up to a dozen devs and CI tests that take a few mins
  - beyond that too much load on gitlab and CI build slaves
  - solution: create a “synthetic” batch merge request of top of queue PRs that have passed branch CI already; make synthetic branch top of queue
    - omit PRs that cause merge conflicts
    - if tests pass, merge all individual PRs (bypassing CI)
    - if tests fail, split
    - in either case, throw away “synthetic” branch

●	CI FAIL	●	CI PASS
○	BROKEN	○	GOOD




CI FAIL
CI PASS
BROKEN
GOOD
PR 1
PR 2
PR 3
B_1
C_1
A_2
M_1
A_1
M_2
M_1



CI FAIL
CI PASS
BROKEN
GOOD
Temp branch
B'1
A'2
A'1
B1
C1
Omit failed CI& merge conflicts
A2
M1
A1
M2
M1

That’s basically it!

Green Master!

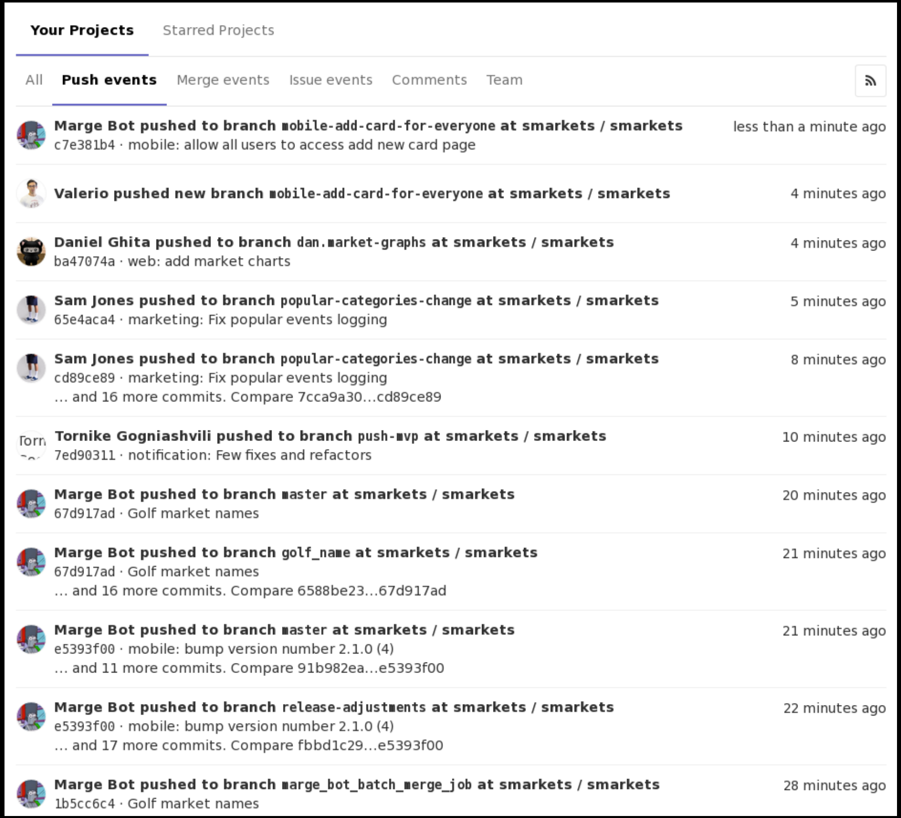
But wait
THERE'S MORE!

Our workflow requirements at Smarkets

Productivity Requirements

Code must flow:

- 70 devs, 11 teams, ~130 services
- Commits every few mins
- ~10 ships to prod/day
- want to preserve velocity



Your Projects   Starred Projects

All   Push events   Merge events   Issue events   Comments   Team

Marge Bot pushed to branch ■obile-add-card-for-everyone at smarkets / smarkets — less than a minute ago
c7e381b4 · mobile: allow all users to access add new card page

Valerio pushed new branch ■obile-add-card-for-everyone at smarkets / smarkets — 4 minutes ago

Daniel Ghita pushed to branch dan.■arket-graphs at smarkets / smarkets — 4 minutes ago
ba47074a · web: add market charts

Sam Jones pushed to branch popular-categories-change at smarkets / smarkets — 5 minutes ago
65e4aca4 · marketing: Fix popular events logging

Sam Jones pushed to branch popular-categories-change at smarkets / smarkets — 8 minutes ago
cd89ce89 · marketing: Fix popular events logging
… and 16 more commits. Compare 7cca9a30…cd89ce89

Tornike Gogniashvili pushed to branch push-■vp at smarkets / smarkets — 10 minutes ago
7ed90311 · notification: Few fixes and refactors

Marge Bot pushed to branch ■aster at smarkets / smarkets — 20 minutes ago
67d917ad · Golf market names

Marge Bot pushed to branch golf_na■e at smarkets / smarkets — 21 minutes ago
67d917ad · Golf market names
… and 16 more commits. Compare 6588be23…67d917ad

Marge Bot pushed to branch ■aster at smarkets / smarkets — 21 minutes ago
e5393f00 · mobile: bump version number 2.1.0 (4)
… and 11 more commits. Compare 91b982ea…e5393f00

Marge Bot pushed to branch release-adjust■ents at smarkets / smarkets — 22 minutes ago
e5393f00 · mobile: bump version number 2.1.0 (4)
… and 17 more commits. Compare fbbd1c29…e5393f00

Marge Bot pushed to branch ■arge_bot_batch_■erge_job at smarkets / smarkets — 28 minutes ago
1b5cc6c4 · Golf market names

Audits & Auditors

- We’re in a regulated industry
- Requirements vary by country
- Goal #1: meet **all** at once
- Goal #2: don’t cripple workflow

Regulatory Requirements

Auditors may want to know:

- Who wrote this code and when?
- Who signed it off?
- How was it validated?
- Why was it needed?
- When was it deployed and by whom?
- Who approved the deployment?

In fact fact **you** might well want to know these things, even if you’re not audited!

Regulatory Requirements

Auditors may want to know:

- Who wrote this code and when? — Git-out of the box
- Who signed it off?
- How was it validated? — Gitlab + Marge-bot (git trailers)
- Why was it needed?
- When was it deployed and by whom? — Our Ship tool + git notes
- Who approved the deployment?

In fact fact **you** might well want to know these things, even if you’re not audited!

Common thread: compliance centred around git

- Cryptographic
- Familiar
- Flexible
- Platform agnostic

Git workflows

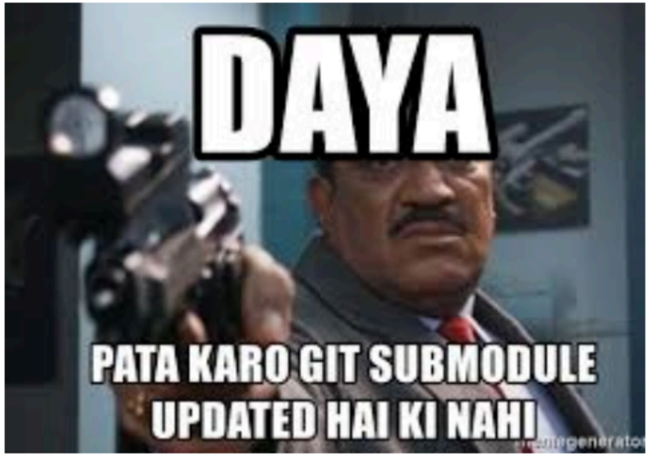
Theory vs Practice

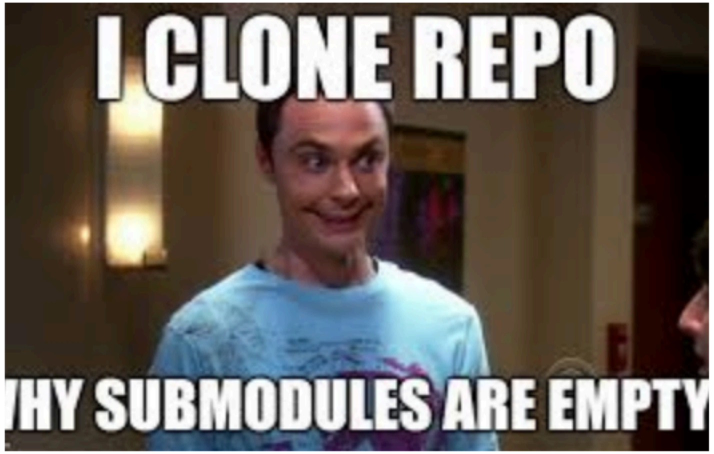
Theory

- Main repo as a collection of **subrepos** of independently developed (micro-)services/libs is a **Good Idea**
  - Macro-view of all that will run in prod in the main repo
  - Micro-view of individual services in subrepo
    - just your service’s history/code, no need to check out GiBs etc.
    - can simplify and speed up CI
- **Merging** (feature) branches (into master) is a **Good Idea**
  - Macro-view of features on master
  - Micro-view of implementation steps in branch
  - history is inviolate!

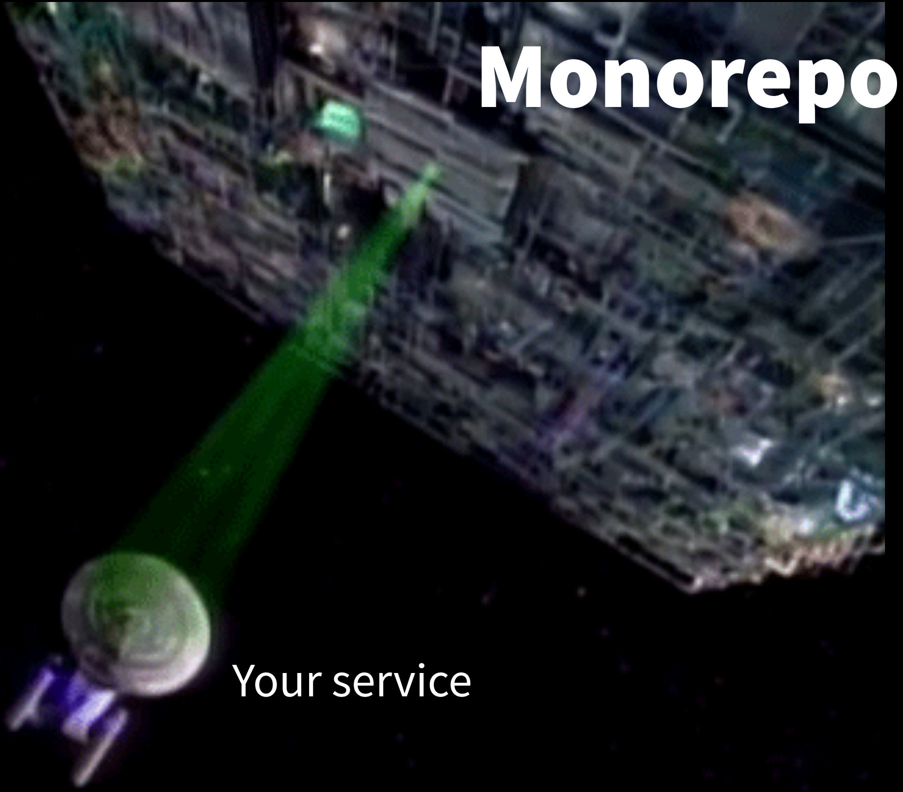
Practice

Google Image Search for Git submodules

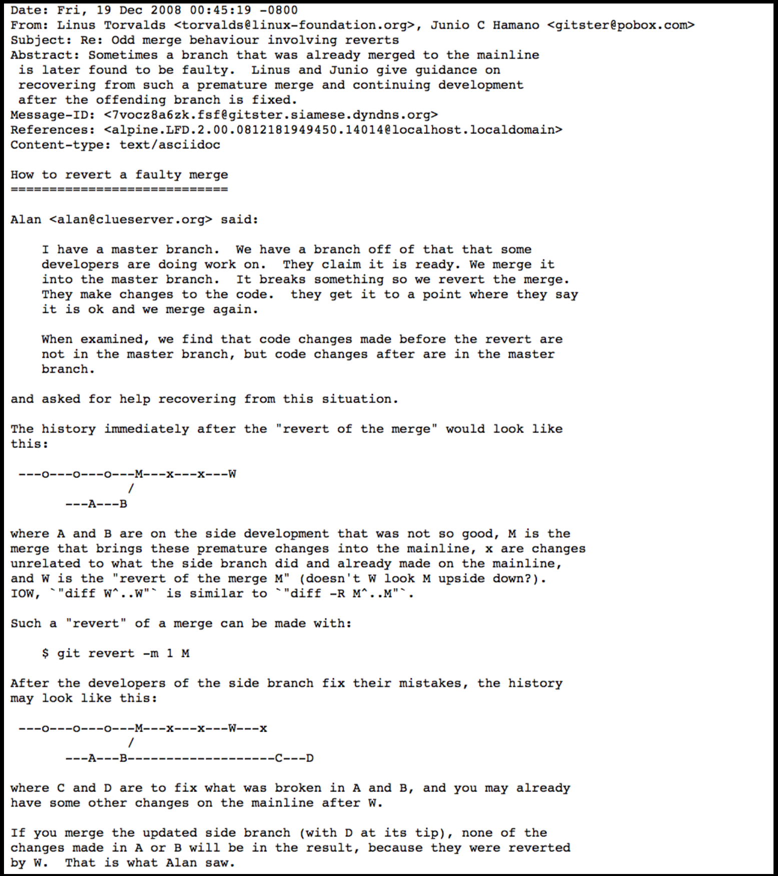




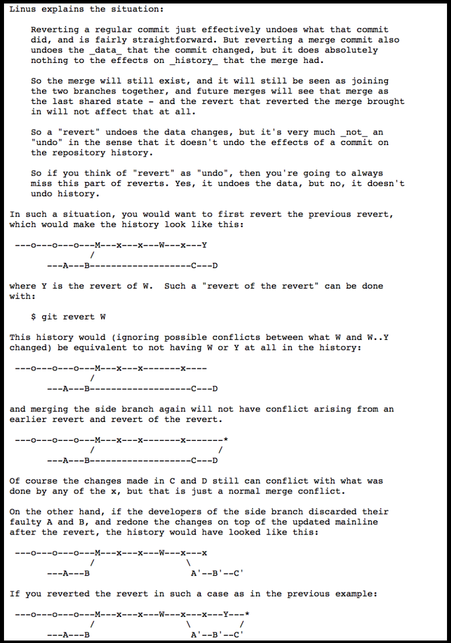
Just go Monorepo, don’t resist assimilation



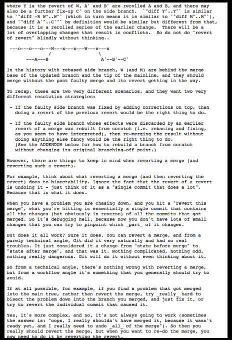
🐧Reverting PRs the Linus way



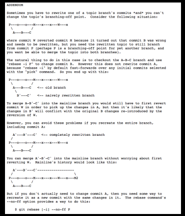
```
Date: Fri, 19 Dec 2008 00:45:19 -0800
From: Linus Torvalds <torvalds@linux-foundation.org>, Junio C Hamano <gitster@pobox.com>
Subject: Re: Odd merge behaviour involving reverts
Abstract: Sometimes a branch that was already merged to the mainline
 is later found to be faulty.  Linus and Junio give guidance on
 recovering from such a premature merge and continuing development
 after the offending branch is fixed.
Message-ID: <7vocz8a6zk.fsf@gitster.siamese.dyndns.org>
References: <alpine.LFD.2.00.0812181949450.14014@localhost.localdomain>
Content-type: text/asciidoc

How to revert a faulty merge
============================

Alan <alan@clueserver.org> said:

    I have a master branch.  We have a branch off of that that some
    developers are doing work on.  They claim it is ready. We merge it
    into the master branch.  It breaks something so we revert the merge.
    They make changes to the code.  they get it to a point where they say
    it is ok and we merge again.

    When examined, we find that code changes made before the revert are
    not in the master branch, but code changes after are in the master
    branch.

and asked for help recovering from this situation.

The history immediately after the "revert of the merge" would look like
this:

 ---o---o---o---M---x---x---W
               /
       ---A---B

where A and B are on the side development that was not so good, M is the
merge that brings these premature changes into the mainline, x are changes
unrelated to what the side branch did and already made on the mainline,
and W is the "revert of the merge M" (doesn't W look M upside down?).
IOW, `"diff W^..W"` is similar to `"diff -R M^..M"`.

Such a "revert" of a merge can be made with:

    $ git revert -m 1 M

After the developers of the side branch fix their mistakes, the history
may look like this:

 ---o---o---o---M---x---x---W---x
               /
       ---A---B-------------------C---D

where C and D are to fix what was broken in A and B, and you may already
have some other changes on the mainline after W.

If you merge the updated side branch (with D at its tip), none of the
changes made in A or B will be in the result, because they were reverted
by W.  That is what Alan saw.
```



```
Linus explains the situation:

    Reverting a regular commit just effectively undoes what that commit
    did, and is fairly straightforward. But reverting a merge commit also
    undoes the _data_ that the commit changed, but it does absolutely
    nothing to the effects on _history_ that the merge had.

    So the merge will still exist, and it will still be seen as joining
    the two branches together, and future merges will see that merge as
    the last shared state - and the revert that reverted the merge brought
    in will not affect that at all.

    So a "revert" undoes the data changes, but it's very much _not_ an
    "undo" in the sense that it doesn't undo the effects of a commit on
    the repository history.

    So if you think of "revert" as "undo", then you're going to always
    miss this part of reverts. Yes, it undoes the data, but no, it doesn't
    undo history.

In such a situation, you would want to first revert the previous revert,
which would make the history look like this:

 ---o---o---o---M---x---x---W---x---Y
               /
       ---A---B-------------------C---D

where Y is the revert of W.  Such a "revert of the revert" can be done
with:

    $ git revert W

This history would (ignoring possible conflicts between what W and W..Y
changed) be equivalent to not having W or Y at all in the history:

 ---o---o---o---M---x-------x----
               /
       ---A---B-------------------C---D

and merging the side branch again will not have conflict arising from an
earlier revert and revert of the revert.

 ---o---o---o---M---x-------x-------*
               /                   /
       ---A---B-------------------C---D

Of course the changes made in C and D still can conflict with what was
done by any of the x, but that is just a normal merge conflict.

On the other hand, if the developers of the side branch discarded their
faulty A and B, and redone the changes on top of the updated mainline
after the revert, the history would have looked like this:

 ---o---o---o---M---x---x---W---x---x
               /                 \
       ---A---B                   A'--B'--C'

If you reverted the revert in such a case as in the previous example:

 ---o---o---o---M---x---x---W---x---x---Y---*
               /                 \         /
       ---A---B                   A'--B'--C'
```



```
where Y is the revert of W, A' and B' are rerolled A and B, and there may
also be a further fix-up C' on the side branch.  `"diff Y^..Y"` is similar
to `"diff -R W^..W"` (which in turn means it is similar to `"diff M^..M"`),
and `"diff A'^..C'"` by definition would be similar but different from that,
because it is a rerolled series of the earlier change.  There will be a
lot of overlapping changes that result in conflicts.  So do not do "revert
of revert" blindly without thinking..

 ---o---o---o---M---x---x---W---x---x
               /                 \
       ---A---B                   A'--B'--C'

In the history with rebased side branch, W (and M) are behind the merge
base of the updated branch and the tip of the mainline, and they should
merge without the past faulty merge and its revert getting in the way.

To recap, these are two very different scenarios, and they want two very
different resolution strategies:

 - If the faulty side branch was fixed by adding corrections on top, then
   doing a revert of the previous revert would be the right thing to do.

 - If the faulty side branch whose effects were discarded by an earlier
   revert of a merge was rebuilt from scratch (i.e. rebasing and fixing,
   as you seem to have interpreted), then re-merging the result without
   doing anything else fancy would be the right thing to do.
   (See the ADDENDUM below for how to rebuild a branch from scratch
   without changing its original branching-off point.)

However, there are things to keep in mind when reverting a merge (and
reverting such a revert).

For example, think about what reverting a merge (and then reverting the
revert) does to bisectability. Ignore the fact that the revert of a revert
is undoing it - just think of it as a "single commit that does a lot".
Because that is what it does.

When you have a problem you are chasing down, and you hit a "revert this
merge", what you're hitting is essentially a single commit that contains
all the changes (but obviously in reverse) of all the commits that got
merged. So it's debugging hell, because now you don't have lots of small
changes that you can try to pinpoint which _part_ of it changes.

But does it all work? Sure it does. You can revert a merge, and from a
purely technical angle, Git did it very naturally and had no real
troubles. It just considered it a change from "state before merge" to
"state after merge", and that was it. Nothing complicated, nothing odd,
nothing really dangerous. Git will do it without even thinking about it.

So from a technical angle, there's nothing wrong with reverting a merge,
but from a workflow angle it's something that you generally should try to
avoid.

If at all possible, for example, if you find a problem that got merged
into the main tree, rather than revert the merge, try _really_ hard to
bisect the problem down into the branch you merged, and just fix it, or
try to revert the individual commit that caused it.

Yes, it's more complex, and no, it's not always going to work (sometimes
the answer is: "oops, I really shouldn't have merged it, because it wasn't
ready yet, and I really need to undo _all_ of the merge"). So then you
really should revert the merge, but when you want to re-do the merge, you
now need to do it by reverting the revert.
```



```
ADDENDUM

Sometimes you have to rewrite one of a topic branch's commits *and* you can't
change the topic's branching-off point.  Consider the following situation:

 P---o---o---M---x---x---W---x
  \         /
   A---B---C

where commit W reverted commit M because it turned out that commit B was wrong
and needs to be rewritten, but you need the rewritten topic to still branch
from commit P (perhaps P is a branching-off point for yet another branch, and
you want be able to merge the topic into both branches).

The natural thing to do in this case is to checkout the A-B-C branch and use
"rebase -i P" to change commit B.  However this does not rewrite commit A,
because "rebase -i" by default fast-forwards over any initial commits selected
with the "pick" command.  So you end up with this:

 P---o---o---M---x---x---W---x
  \         /
   A---B---C   <-- old branch
    \
     B'---C'   <-- naively rewritten branch

To merge A-B'-C' into the mainline branch you would still have to first revert
commit W in order to pick up the changes in A, but then it's likely that the
changes in B' will conflict with the original B changes re-introduced by the
reversion of W.

However, you can avoid these problems if you recreate the entire branch,
including commit A:

   A'---B'---C'  <-- completely rewritten branch
  /
 P---o---o---M---x---x---W---x
  \         /
   A---B---C

You can merge A'-B'-C' into the mainline branch without worrying about first
reverting W.  Mainline's history would look like this:

   A'---B'---C'------------------
  /                              \
 P---o---o---M---x---x---W---x---M2
  \         /
   A---B---C

But if you don't actually need to change commit A, then you need some way to
recreate it as a new commit with the same changes in it.  The rebase command's
--no-ff option provides a way to do this:

    $ git rebase [-i] --no-ff P
```

...

https://github.com/git/git/blob/master/Documentation/howto/revert-a-faulty-merge.txt

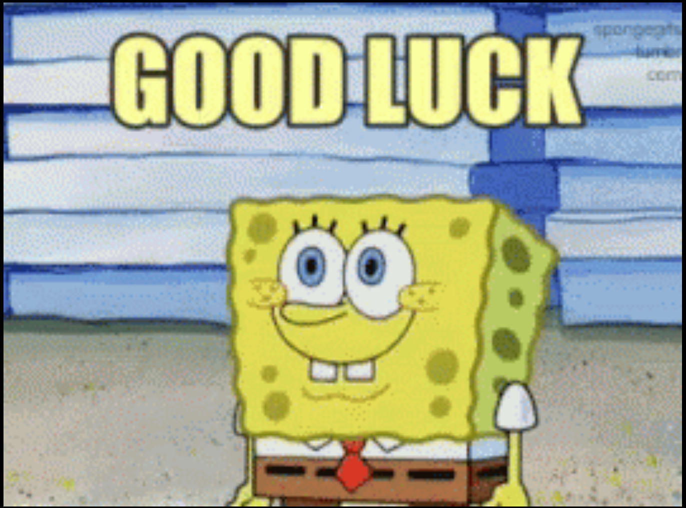
🤖 Reverting PRs the marge-bot way

```
git revert-mr 123
```

Which one would you rather do when prod is broken?

Now *which* PR actually broke this? (plain git way)

- I know, I'll use `git bisect run` to find out



- Only look at merge commits to master? **Needs 3rd party tool.**
- Feature branch commits will often not have passed CI, will get false positives.

Now *which* PR actually broke this? (marge-bot way)

```
git bisect-run-tested ./test.sh
```

Only runs on commits that passed CI
(marge-bot can add Tested-by: trailer to last commit of PR if you have mandatory CI switched on in Gitlab)

How does it work?

Recap: bare bones marge setup

```
docker run --restart=on-failure \
  --env-file=<(
      echo MARGE_AUTH_TOKEN="$(cat marge-bot.token)";
      echo MARGE_SSH_KEY="$(cat marge-bot-ssh-key)") \
  smarkets/marge-bot \
  --gitlab-url='http://your.gitlab.instance.com'
```

More --args, more features!

```
docker run --restart=on-failure \
  --env-file=<(
     echo MARGE_AUTH_TOKEN="$(cat marge-bot.token)";
     echo MARGE_SSH_KEY="$(cat marge-bot-ssh-key)") \
  smarkets/marge-bot \
  --gitlab-url='http://your.gitlab.instance.com' \
  --batch \             Try to optimistically batch several PRs for faster CI
  --add-part-of \       Adds a "Part-of: <PR-url>" Trailer to commits
  --add-tested \        Adds a "Tested-by: <PR-url>" Trailer to final commit in PR
  --add-reviewers \     Adds a "Reviewed-by: <guy who approved PR>" Trailer to commits
  --impersonate-approvers     Re-approve after rewriting commits to add Trailers
```

Bare bones commit

```
commit 5ddc293ec55408ecc101eacf6495421a16182633
Author: Jaime Lennox <jaime.lennox@smarkets.com>
Date:   Mon Jul 23 11:46:41 2018 +0100

    marge-bot: bump to version 0.7.0

    There's a new version of Marge available, so let's update our in-house
    version to match.
```

Optional Features: Audit with --add-reviewers

```
commit 146891a956fd35cf8ab6445d7ec76fddf4230925
Author: Jaime Lennox <jaime.lennox@smarkets.com>
Date:   Mon Jul 23 11:46:41 2018 +0100

    marge-bot: bump to version 0.7.0

    There's a new version of Marge available, so let's update our in-house
    version to match.

    Reviewed-by: Tornike Gogniashvili <tornike.gogniashvili@smarkets.com>
```

Optional Features: Bisect with --add-tested

```
commit ca582b7ab9f03f496509291b1fa2e8f768a76f05
Author: Jaime Lennox <jaime.lennox@smarkets.com>
Date:   Mon Jul 23 11:46:41 2018 +0100

    marge-bot: bump to version 0.7.0

    There's a new version of Marge available, so let's update our in-house
    version to match.

    Reviewed-by: Tornike Gogniashvili <tornike.gogniashvili@smarkets.com>
    Tested-by: <https://git.corp.smarkets.com/smarkets/smarkets/merge_requests/9727>
```

Optional Features: Revert with --add-part-of

```
commit 088bf8546b73b559322a8744e867cf8949fe6225
Author: Jaime Lennox <jaime.lennox@smarkets.com>
Date:   Mon Jul 23 11:46:41 2018 +0100

    marge-bot: bump to version 0.7.0

    There's a new version of Marge available, so let's update our in-house
    version to match.

    Reviewed-by: Tornike Gogniashvili <tornike.gogniashvili@smarkets.com>
    Tested-by: <https://git.corp.smarkets.com/smarkets/smarkets/merge_requests/9727>
    Part-of: <https://git.corp.smarkets.com/smarkets/smarkets/merge_requests/9727>
```

Then it's just a bunch of git aliases!

```
git config --global alias.bisect-run-tested \
    'f() { git bisect run /bin/sh -c
             "if !(git log -1 --format %B
                 | fgrep -q \"Tested-by: Marge Bot\");
              then exit 125;
              else "$@"; fi"; }; f'

git config --global alias.mr-revs \
    '!f() { git log --grep "^Part-of.*/""$1"">" --pretty="%H"; }; f'
git config --global alias.mr-url \
    '!f() { git log -1 --grep "^Part-of.*/""$1"">" --pretty="%b" |
      grep "^Part-of.*/""$1"">"  | sed "s/.*<\\(.*\\)>/\\1/"; }; f'
git config --global alias.revert-mr \
    '!f() { REVS=$(git mr-revs "$1"); URL="$(git mr-url "$1")";
        git revert --no-commit $REVS;
        git commit -m "Revert <$URL>$(echo;echo; echo "$REVS"
               | xargs -I% echo "This reverts commit %.")"; }; f'
```

What if you can't use Marge-bot (not on Gitlab)?

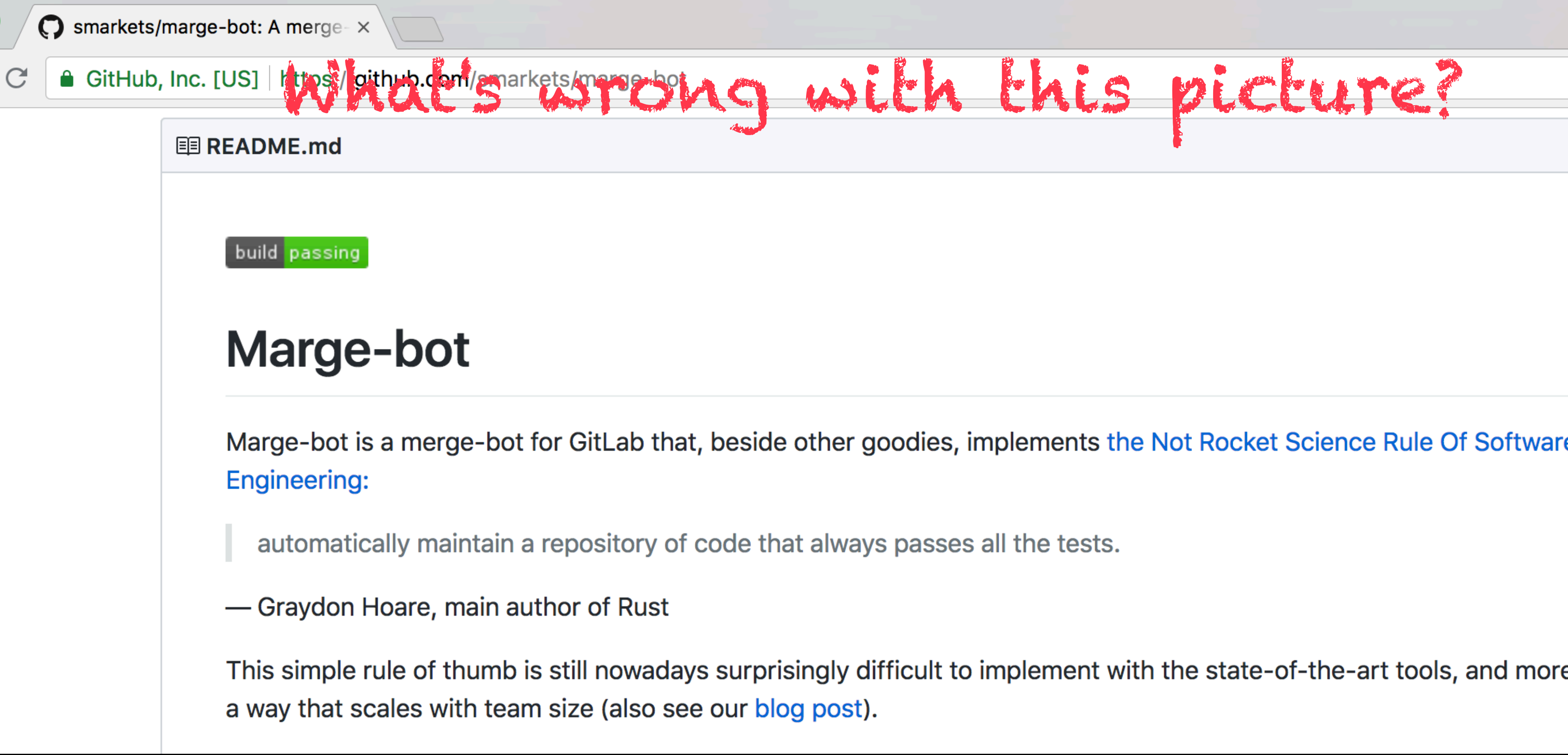
What if you can't use Marge-bot (not on Gitlab)?

- At least now you know what you're missing ;)
- If you don't need something general, roll-your-own is often pretty easy
    - we more or less did that at my last 3 employers
- Also, we welcome PRs to https://github.com/smarkets/marge (github backend would be cool, and probably straightforward!)
- Similar tools might exists; e.g. for github there's also Rust's homu

Summary

- A good PR workflow runs tests against “future” master not just the feature branch
- good ≠ common (but now you know you want it and how to get it!)
- https://github.com/smarkets/marge-bot will do it for gitlab, the way you want (merge or rebase-based)
- Marge-bot can also add Trailers to show who Reviewed commit and what PR
  - Combines best of Merge and Rebase based workflows (e.g. you can still see what PR commits belonged to)
  - Extra perks:
    - git bisect that actually works (at PR level)
    - git revert that actually works (at PR level)
- In practice: Monorepo and rebasing PRs > subrepos and merging PRs (usually)

What's wrong with this picture?


smarkets/marge-bot: A merge
GitHub, Inc. [US]
https://github.com/smarkets/marge-bot
README.md
build passing

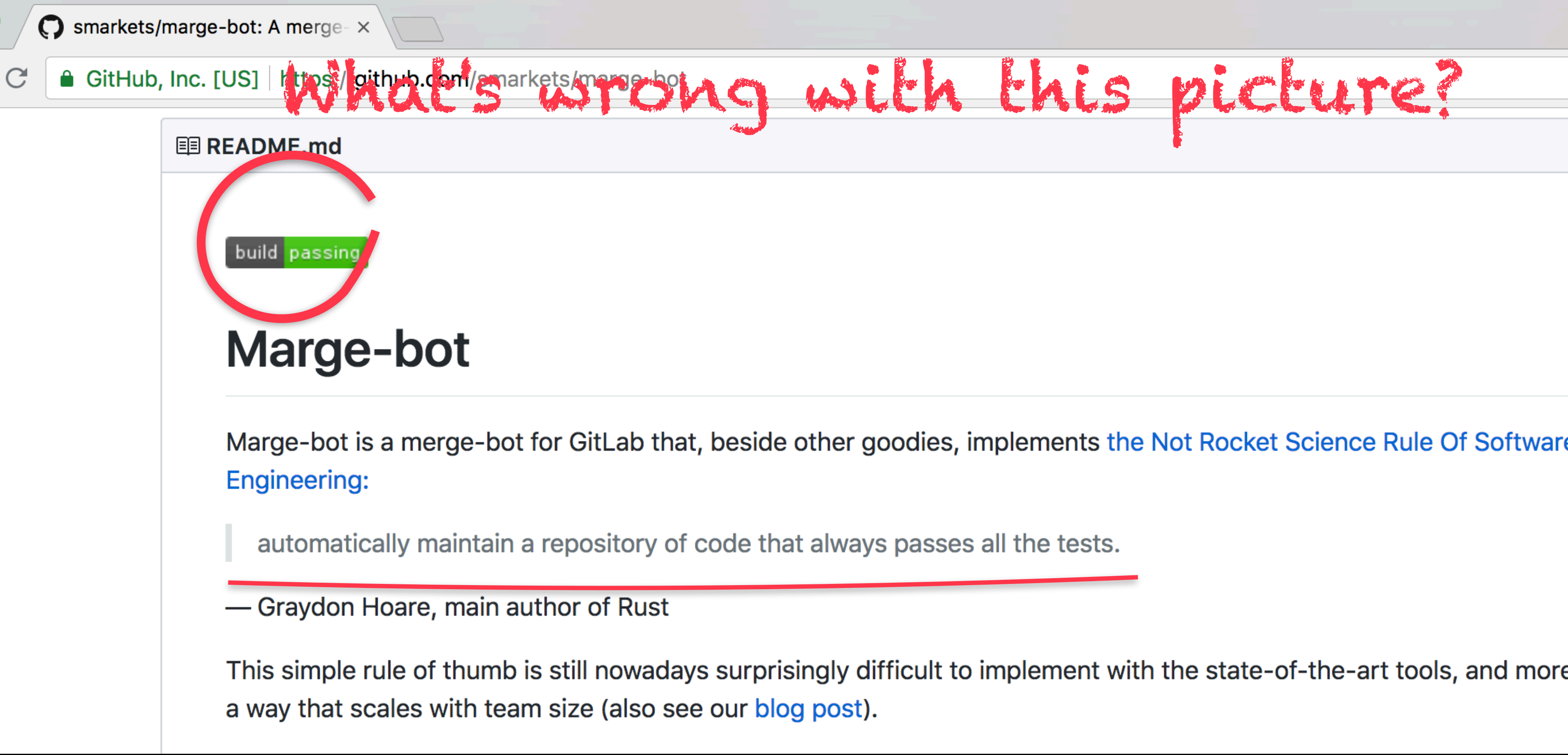
Marge-bot

Marge-bot is a merge-bot for GitLab that, beside other goodies, implements the Not Rocket Science Rule Of Software Engineering:

> automatically maintain a repository of code that always passes all the tests.

— Graydon Hoare, main author of Rust

This simple rule of thumb is still nowadays surprisingly difficult to implement with the state-of-the-art tools, and more a way that scales with team size (also see our blog post).



smarkets/marge-bot: A merge- ×

GitHub, Inc. [US] https://github.com/smarkets/marge-bot

What's wrong with this picture?

README.md

build passing

Marge-bot

Marge-bot is a merge-bot for GitLab that, beside other goodies, implements the Not Rocket Science Rule Of Software Engineering:

> automatically maintain a repository of code that always passes all the tests.

— Graydon Hoare, main author of Rust

This simple rule of thumb is still nowadays surprisingly difficult to implement with the state-of-the-art tools, and more
a way that scales with team size (also see our blog post).

Credits

- Daniel Gorin (initial design & implementation)

- Jaime Lennox (batching, slack integration, general maintenance)

- Alexander Schmolck (trailers, nix-based build, general maintenance)
- Marian Rusu (batching, build & CI improvements)

- all our users who submitted PRs, suggestions and bug reports

Extra Material

Gitlab Wish list

- `Commit message changes should not trigger CI`
- `Rebasing the target branch into the source branch should not reset approvals`
- `Getting approver email should not require excessive perms`
- Merge API should
    - take optional expected hash of master (don't merge if it doesn't match)
    - allow overriding CI trigger and force merge (e.g. “trust me, it's tested” for batch mode)
- PRs from forks are kinda broken
    - CI is on fork (which probably hasn't the right setup)

Implementation details...

... that I think worked well

Architecture

- keeping it simple:
  - stateless
  - no concurrency
  - crash on network or Gitlab failures (HTTP 50x)
    - `rely on` `` `docker run --restart=on-failure` or systemd ``
- Advantages:
  - simple! (no messy python-style concurrency, linear log of actions easy to grok)
- Disadvantages:
  - extra requests/slower (e.g. cloning big repo again after crash); not an issue for us so far
- yield-on-sleep and retrying requests might be good complexity/benefit trade-off

ConfigArgParse: mix and match --args, ENV and config files

- Marge accepts most options as env-var --arg or yaml config option
- Various benefits
  - quickly override option in config file for test (e.g. new auth token)
  - config file best for complex setup, command line args sufficient for vanilla setup
- Can still customize:
  - we disallow passing secrets as commandline args for security reasons

Derive from namedtuples for lightweight biz logic classes

```python
class Version(namedtuple('Version', 'release edition')):
    @classmethod
    def parse(cls, string):
        release_string, edition = string.split('-', 1)
        release = tuple(int(number) for number in release_string.split('.'))
        return cls(release=release, edition=edition)

    @property
    def is_ee(self):
        return self.edition == 'ee'
```

(constructor, repr, updated-copy and enforced immutability for free)

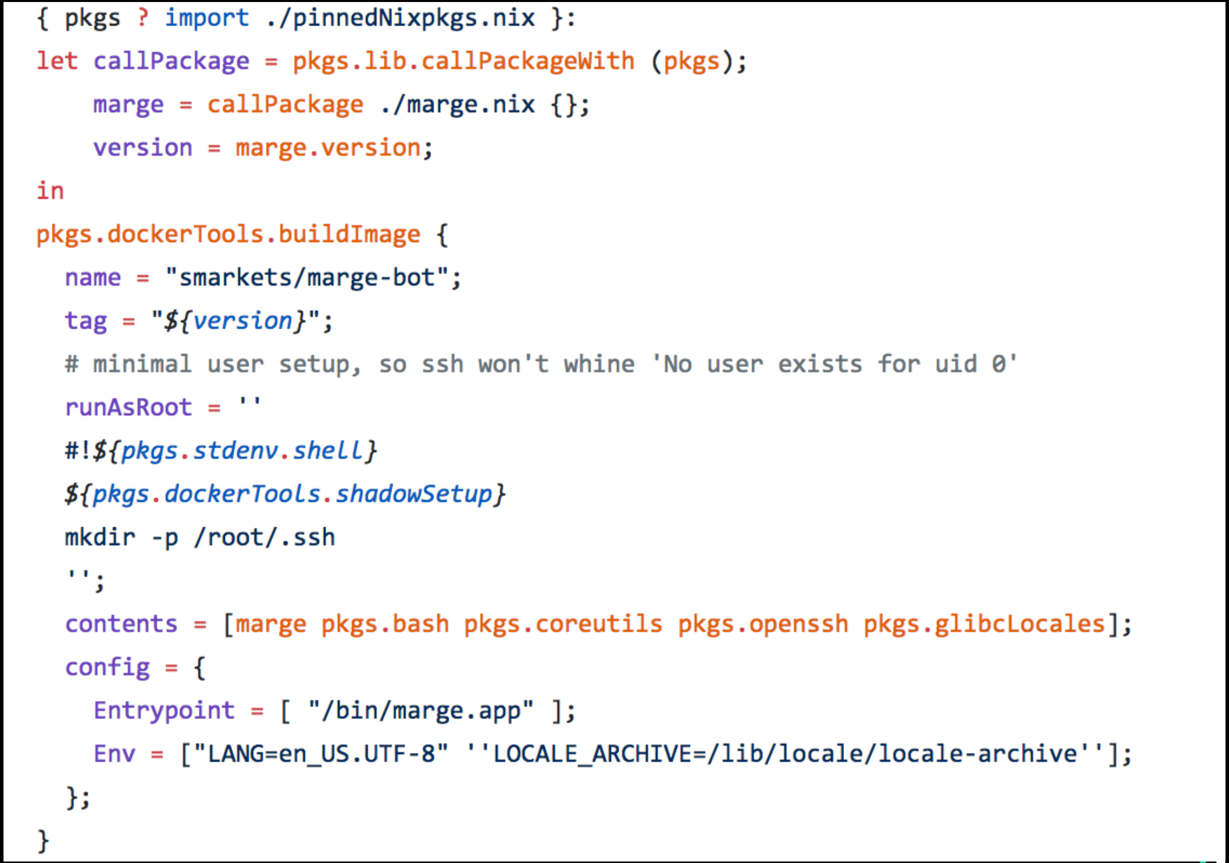
Better mocking with state machines

```
class MockLab(object):
    def __init__(self, gitlab_url=None):
        self.api = api = ApiMock('...', initial_state='initial')

        api.add_transition(GET('/version'), Ok({'version': '9.2.3-ee'}))
        # [...]
        api.add_transition(
            GET('/projects/1234/repository/commits/%s' % rewritten_sha),
            Ok(commit_after_pushing),
            from_state='pushed', to_state='passed',
        )
        api.add_transition(
            GET('/projects/1234/repository/commits/%s' % rewritten_sha),
            Ok(_commit(id=rewritten_sha, status='success')),
            from_state=['passed', 'merged'],
        )
```

(Nicer than typical mocking, IMO, but learning curve has kept it out of recent code)

nix: Completely reproducible (docker image) builds (w/o docker)

- fast, correct, small (pick 3)
- Travis supports it
- drives CI and images
- nix-shell --pure = virtualenv on steroids
  - also handles non-py deps, e.g. git, ssh
  - actually works



```
{ pkgs ? import ./pinnedNixpkgs.nix }:
let callPackage = pkgs.lib.callPackageWith (pkgs);
    marge = callPackage ./marge.nix {};
    version = marge.version;
in
pkgs.dockerTools.buildImage {
  name = "smarkets/marge-bot";
  tag = "${version}";
  # minimal user setup, so ssh won't whine 'No user exists for uid 0'
  runAsRoot = ''
  #!${pkgs.stdenv.shell}
  ${pkgs.dockerTools.shadowSetup}
  mkdir -p /root/.ssh
  '';
  contents = [marge pkgs.bash pkgs.coreutils pkgs.openssh pkgs.glibcLocales];
  config = {
    Entrypoint = [ "/bin/marge.app" ];
    Env = ["LANG=en_US.UTF-8" ''LOCALE_ARCHIVE=/lib/locale/locale-archive''];
  };
}
```