Rebuilding the airplane at 10 000m

Continuous Deployment with Jenkins and Gerrit

R. Tyler Croy
tyler@linux.com

Hello and thanks for coming. I'm R. Tyler Croy, and today I'm going to talk to you about two things at the same time!

I'm going to tell you how we rebuilt our engineering organization "mid-flight" for Continuous Deployment and at the same time, I'm going to tell you how you can too with Jenkins and Gerrit

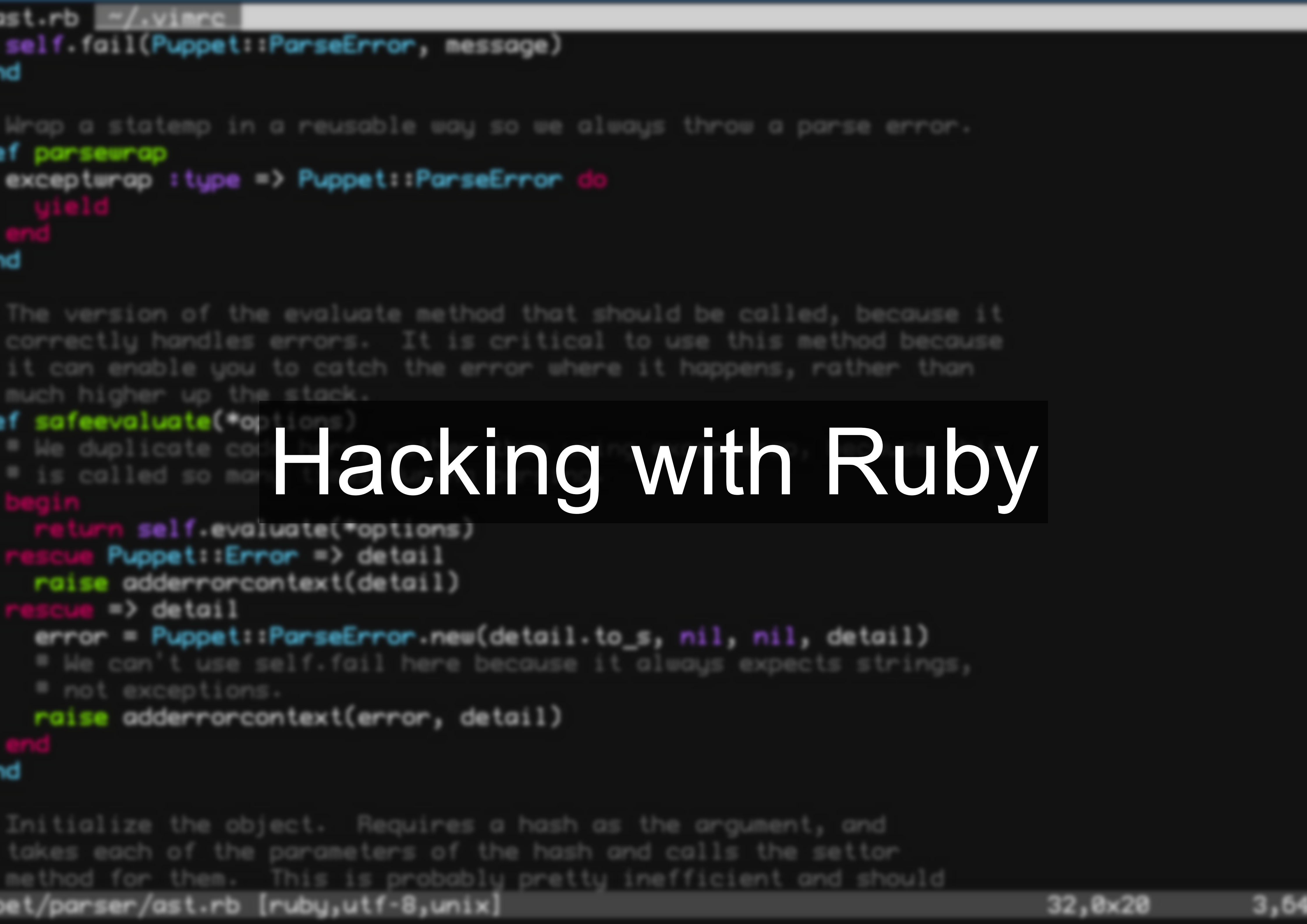
I work for Lookout

First of all, I work at Lookout Mobile Security, if you're an Android user you might already be familiar with some of our products.

If you're not familar with us, we are primarily known for our security app on Android.

My job at Lookout primarily involves ->

Hacking with Ruby



```
ast.rb  ~/.vimrc
 self.fail(Puppet::ParseError, message)
nd

 Wrap a statemp in a reusable way so we always throw a parse error.
ef parsewrap
 exceptwrap :type => Puppet::ParseError do
   yield
 end
nd

 The version of the evaluate method that should be called, because it
 correctly handles errors.  It is critical to use this method because
 it can enable you to catch the error where it happens, rather than
 much higher up the stack.
ef safeevaluate(*op
 # We duplicate co
 # is called so man
 begin
   return self.evaluate(*options)
 rescue Puppet::Error => detail
   raise adderrorcontext(detail)
 rescue => detail
   error = Puppet::ParseError.new(detail.to_s, nil, nil, detail)
   # We can't use self.fail here because it always expects strings,
   # not exceptions.
   raise adderrorcontext(error, detail)
 end
nd

 Initialize the object.  Requires a hash as the argument, and
 takes each of the parameters of the hash and calls the settor
 method for them.  This is probably pretty inefficient and should
et/parser/ast.rb [ruby,utf-8,unix]                          32,0x20        3,6
```

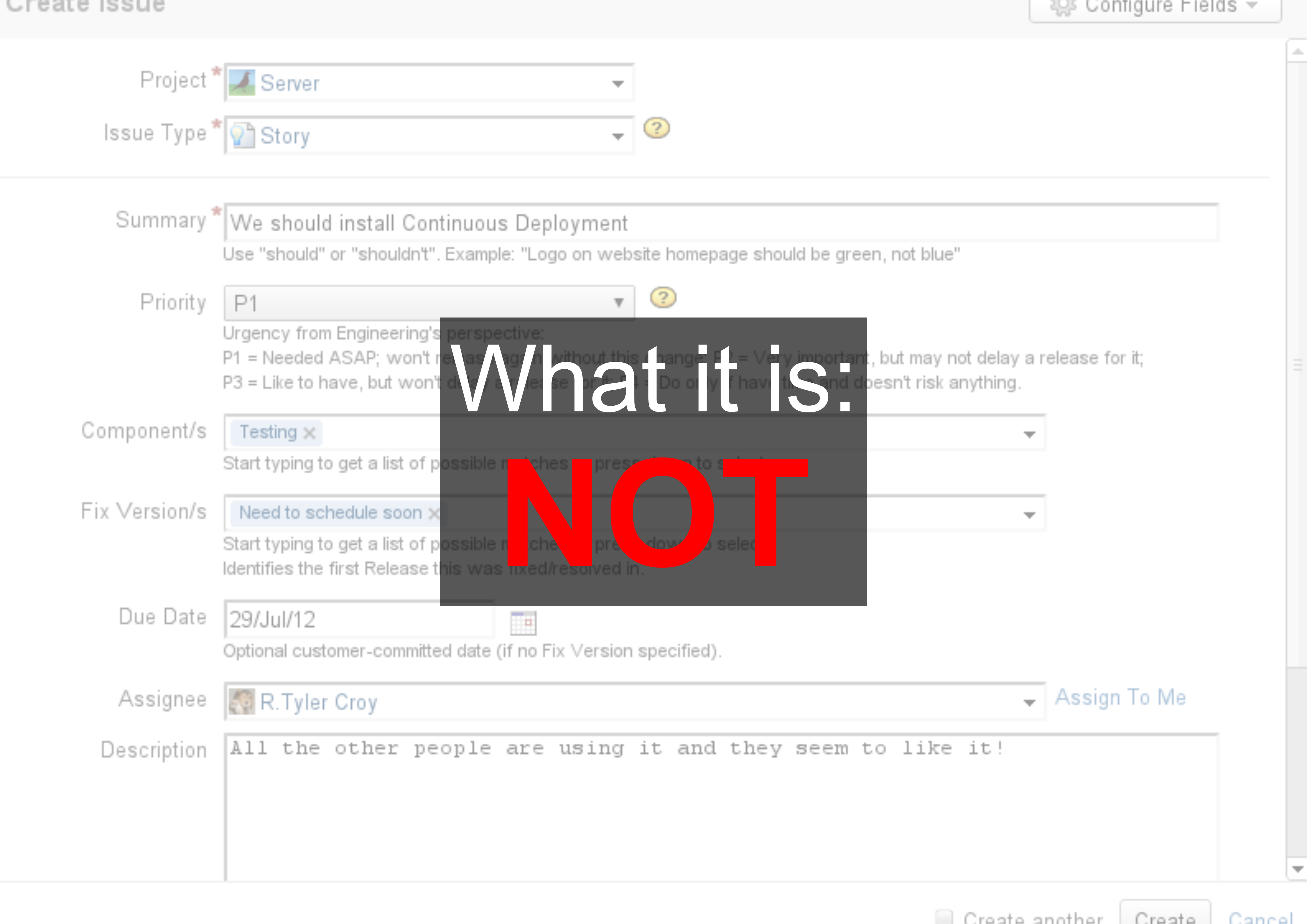
Hacking with Ruby, you see while we have a nice fancy Java-based Android application, we also have a *large* server-backend which handles device notifications, backups, analysis and much more.

That entire backend is written in Ruby, and can benefit from ->

Let's talk about: **continuous deployment**

Continuous Deployment.

Before I talk too much about what it *is*, I'd like to talk about ->



Create Issue

Project* Server

Issue Type* Story

Summary* We should install Continuous Deployment

Use "should" or "shouldn't". Example: "Logo on website homepage should be green, not blue"

Priority P1

Urgency from Engineering's perspective:
P1 = Needed ASAP; won't release again without this change; P2 = Very important, but may not delay a release for it;
P3 = Like to have, but won't delay a release for it; P4 = Do only if have time and doesn't risk anything.

Component/s Testing ×

Start typing to get a list of possible matches or press down to select.

Fix Version/s Need to schedule soon ×

Start typing to get a list of possible matches or press down to select.
Identifies the first Release this was fixed/resolved in.

Due Date 29/Jul/12

Optional customer-committed date (if no Fix Version specified).

Assignee R.Tyler Croy

Description

```
All the other people are using it and they seem to like it!
```

What it is:

NOT

What Continuous Deployment is NOT.

Above all else, it is not something you do *once* and then you're finished
and you can move on. Continous Deployment is a process and mind-set you and
your team stick with

"Release *everything* as soon as possible!"

Continuous Deployment doesn't mean you release EVERYTHING as soon as it's committed. Nor does it mean you must deploy every single commit.

(photo by thomen: <http://www.flickr.com/photos/thomen/364890522/>)

"Great! No need for a QA team"

One of the interesting things I've discovered at Lookout and at other organizations, is that Continuous Deployment, and some of the practices involved in it is that it will free up the QA team to *do their jobs*.

Good QA engineers are most useful when they're exploring, hunting for bugs. Having QA engineers running through test plans every day of the week is boring, slow and is a good candidate for replacement with automated testing tools

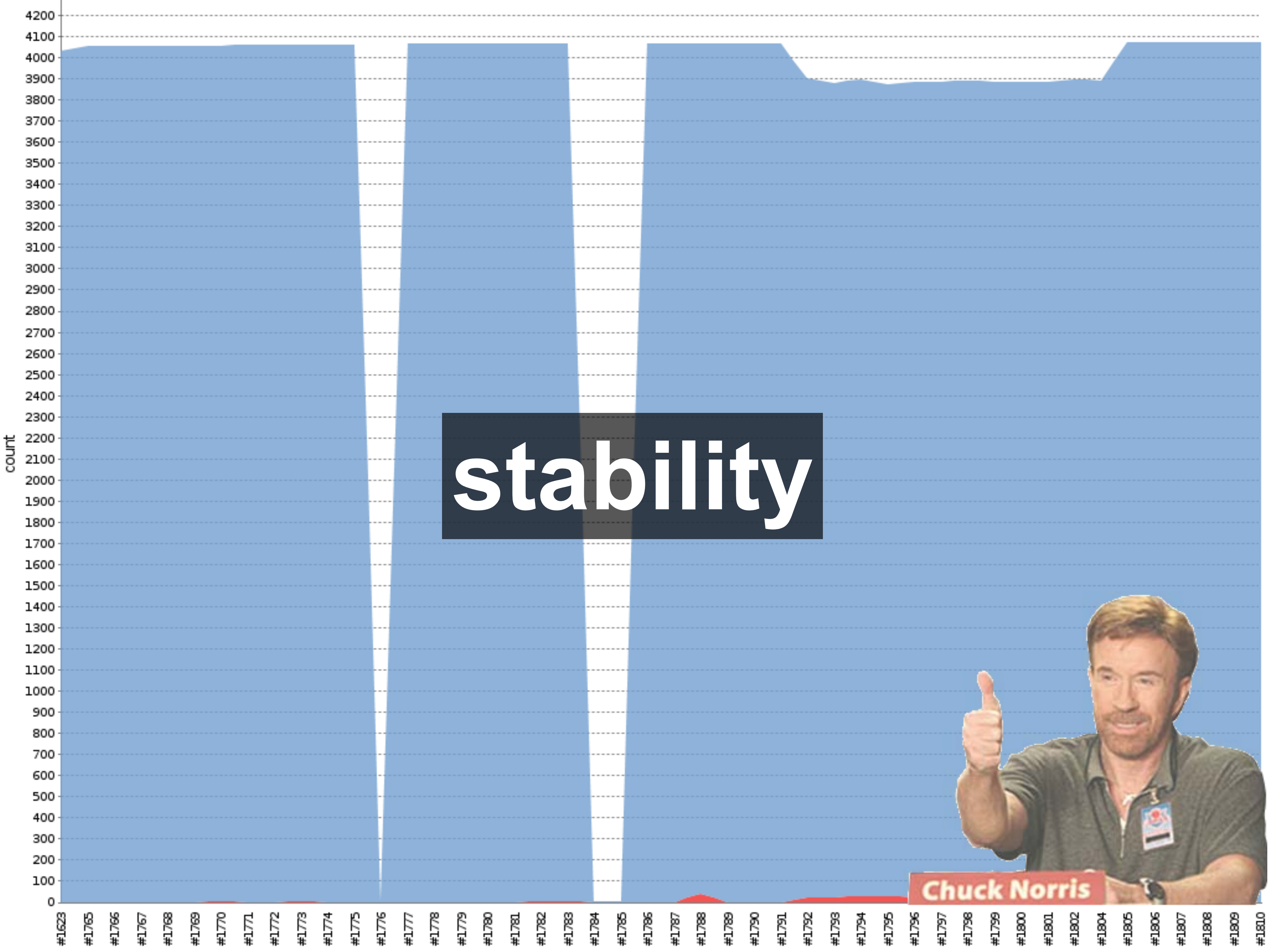
"Our users will be our testers!"

Lastly, I don't think Continuous Deployment means you can offload testing to your users.

To some extent this will be inevitable, as changes are more rapidly deployed, but I believe you should try everything you can to avoid your users experiencing issues because of bugs you've introduced

Continuous Deployment is about

Continous Deployment, in my opinion, is all about ->


4200
4100
4000
3900
3800
3700
3600
3500
3400
3300
3200
3100
3000
2900
2800
2700
2600
2500
2400
2300
2200
2100
2000
1900
1800
1700
1600
1500
1400
1300
1200
1100
1000
900
800
700
600
500
400
300
200
100
0
count
#1623
#1765
#1766
#1767
#1768
#1769
#1770
#1771
#1772
#1773
#1774
#1775
#1776
#1777
#1778
#1779
#1780
#1781
#1782
#1783
#1784
#1785
#1786
#1787
#1788
#1789
#1790
#1791
#1792
#1793
#1794
#1795
#1796
#1797
#1798
#1799
#1800
#1801
#1802
#1804
#1805
#1806
#1807
#1808
#1809
#1810
Chuck Norris

stability

stability. It is about being able to deploy changes ->

Faster
with
More Confidence

Continuous Deployment is about releasing changes faster, and with more confidence in the changes you're preparing the release.

In order to make that happen, it is important to have *good* procedures for rapid deployment and an excellent feedback loop from production. These two factors, above anything else will enable you to ship code rapidly and maintain quality.

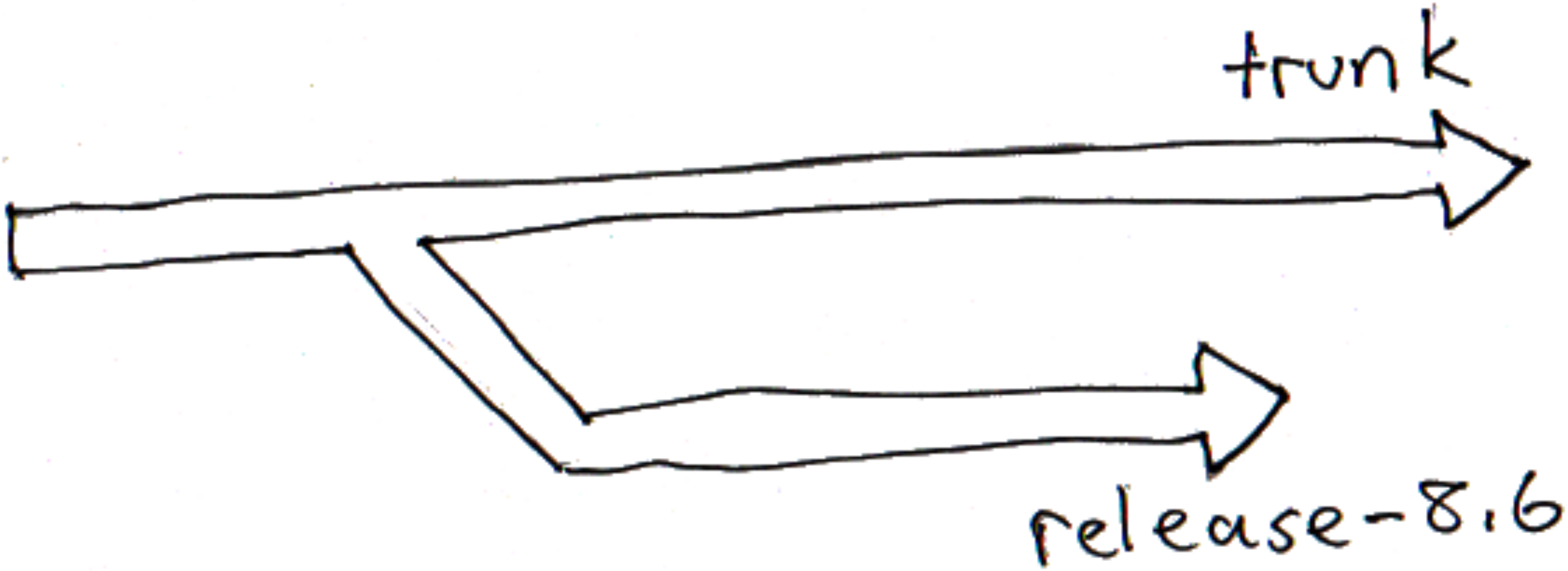
Continuous Deployment is

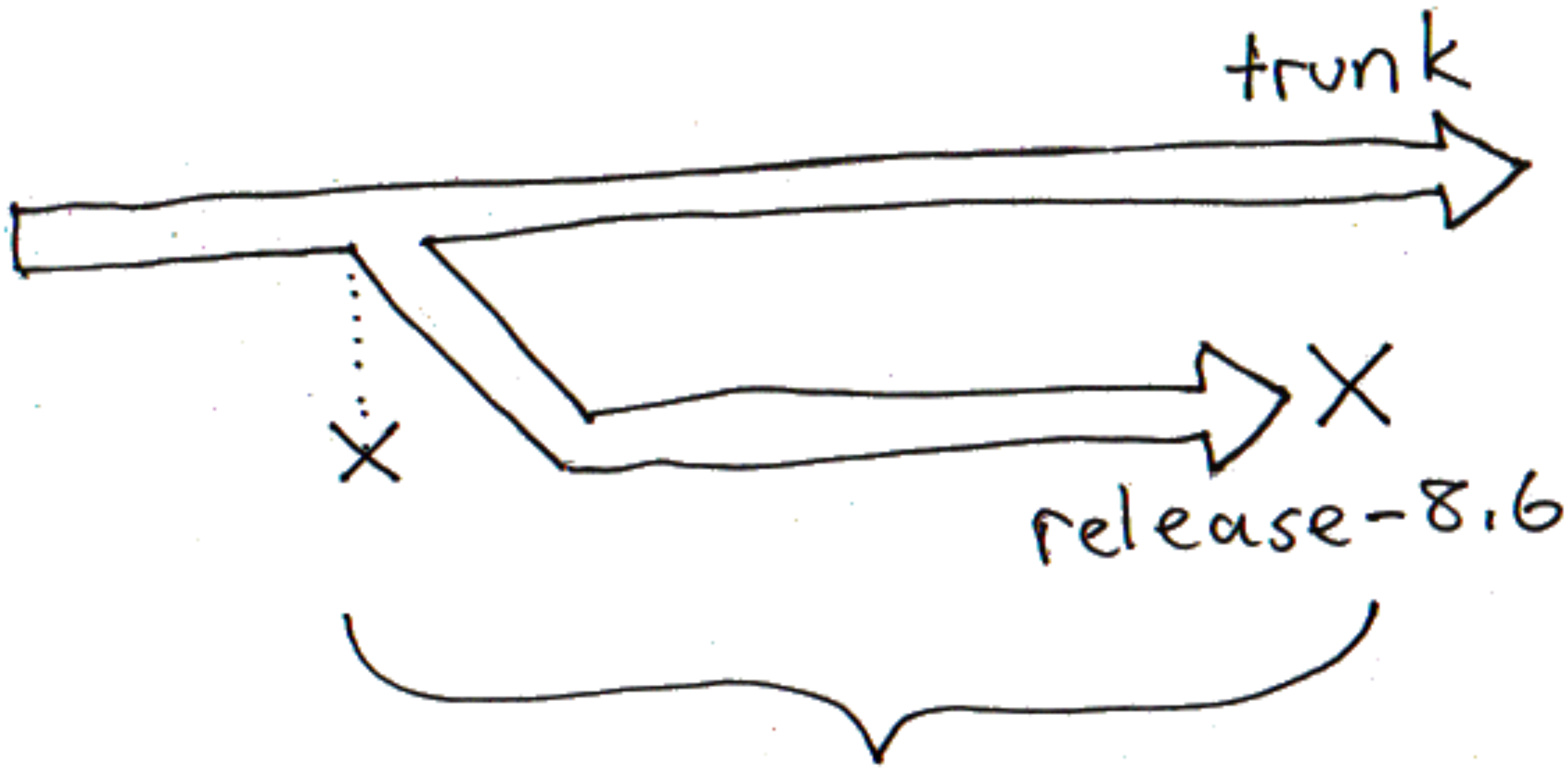
GOOD

Continuous Deployment is GOOD for your organization. Even if you don't end up rapidly deploying your software, the practice of *striving* for continuous deployment will help improve so many other parts of your development process

Meanwhile at Lookout

Subversion branches for releases





10-18 days per release branch

manual code review

TODO: Fill out a good description of code review pre-git+gerrit

very little automation

Sadness with Numbers

36%

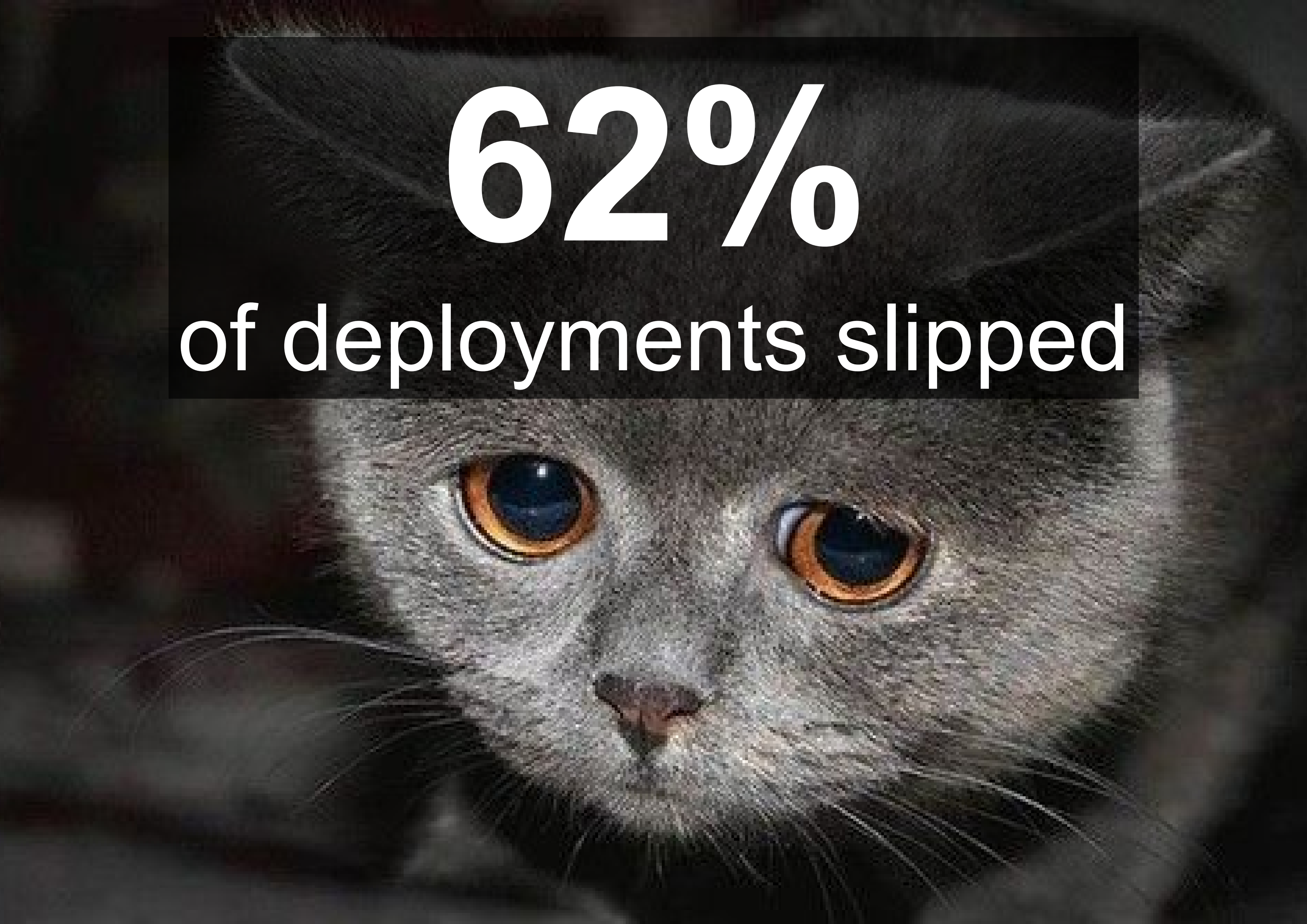
of deployments failed

68

commits per deployment

62%

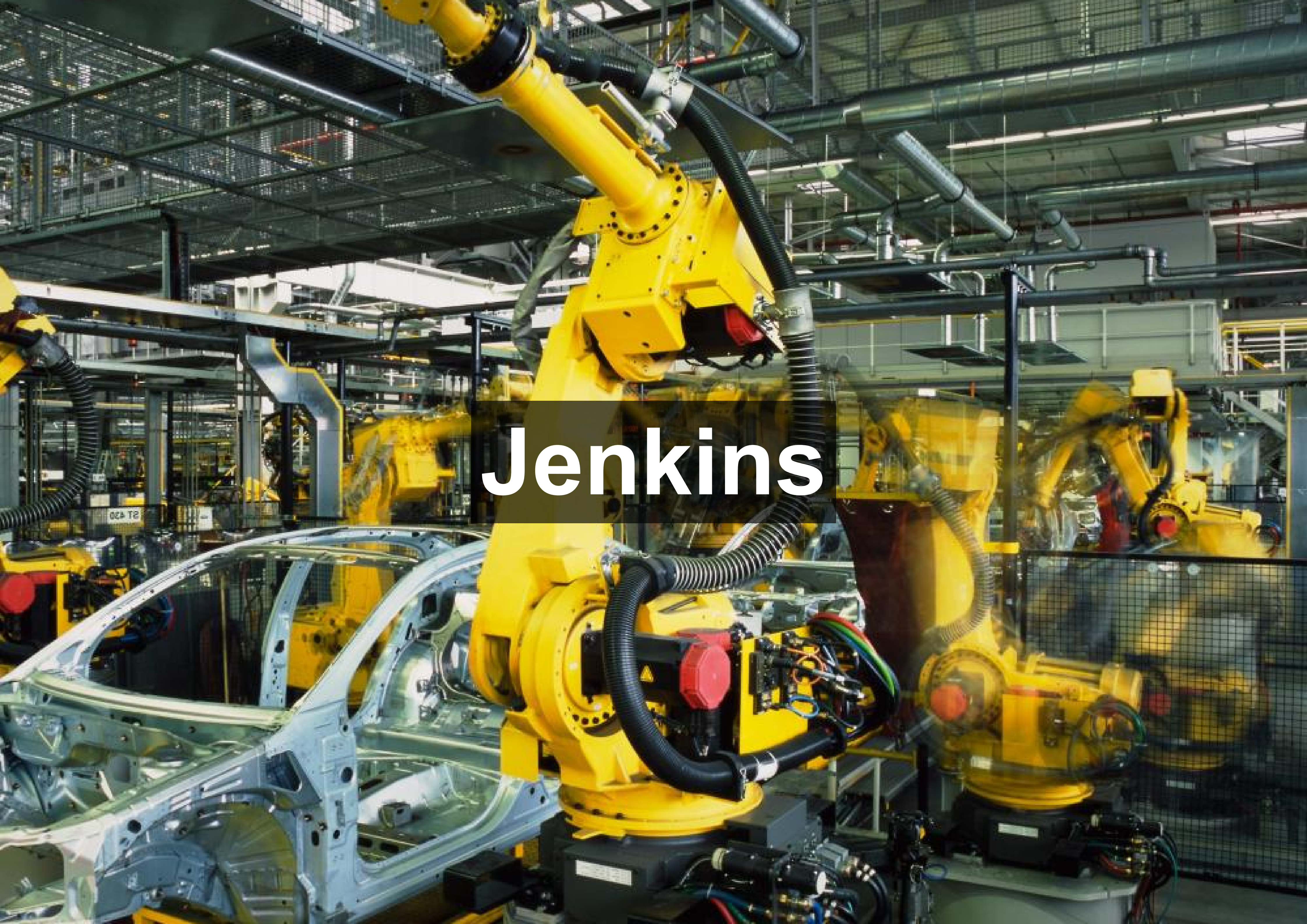
of deployments slipped



Let's **fix** this.

Step One: **Automate**

Jenkins



Before we used a tool called Bitten, I won't tell you too much about Bitten, but it's not a great tool and we had a number of issues with it:

* Practically zero developer insight into the test/build process
* All the tests ran on *one* build machine which was hand-crafted by the Operations team for the task

We installed Jenkins and started to work on migration "jobs" over to Jenkins.

"Why don't our tests pass?"

The first major issue we had was that we noticed that we had tests that didn't actually *pass* reliably. Previously this was hidden from us, but after running the tests after every commit with Jenkins, we noticed that we had some technical debt in the test suite

Never stop automating.

TODO: Need to dive into how automation is an on-going commitment

Step Two

Better tools, processes

The second important step we took towards continuous deployment was to use
better tools and processes than we were using

(I don't like SVN)

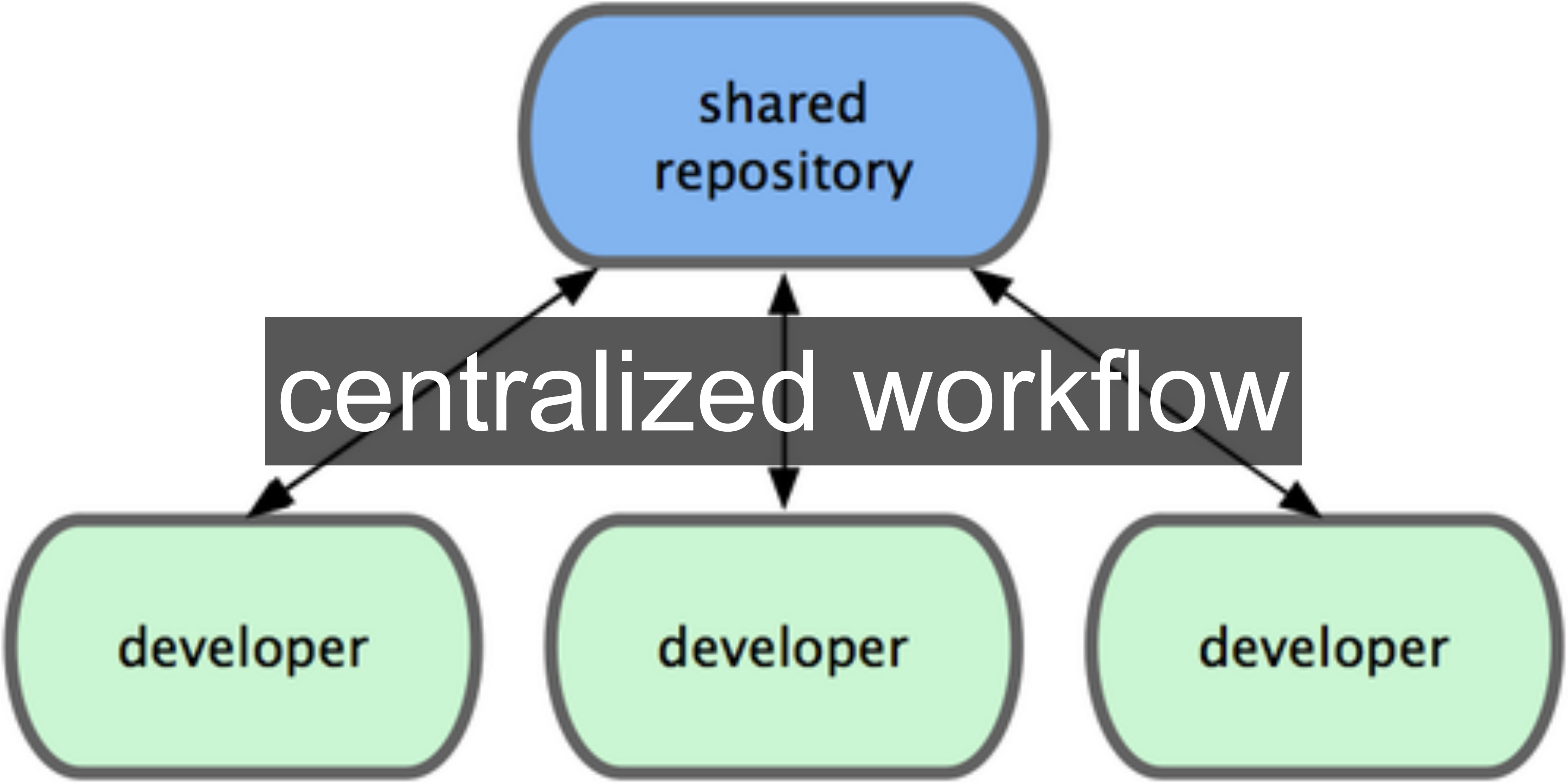
I'm not going to rant against Subversion here, if you like it, that's fine. There are ways to accomplish continuous deployment with Subversion. At Lookout however, I viewed it as one of the things standing in our way.

So we got rid of Subversion and instead opted for ->

Git

USAG 374N/35
SOLA
AV 30 cm
BAHCO ergo 8071 MADE IN SWEDEN
ORBIS
MADE IN FRANCE
PH2 x 100
C 311
Wiha SoftFinish
3,0 x 75
C 302
Wiha SoftFinish
PB SWISS TOOLS
PB 6510 R-30
USAG 382F 300

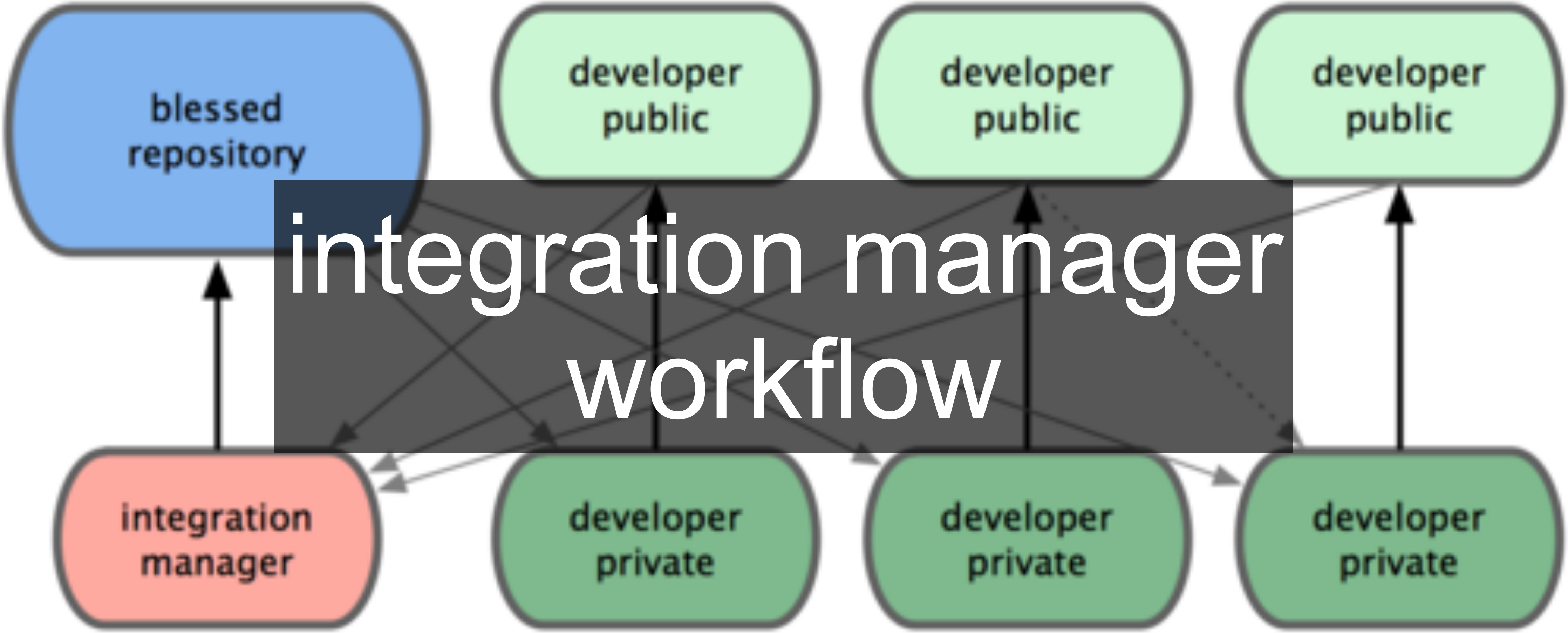
Git can be used a number of different ways, there's the familiar ->



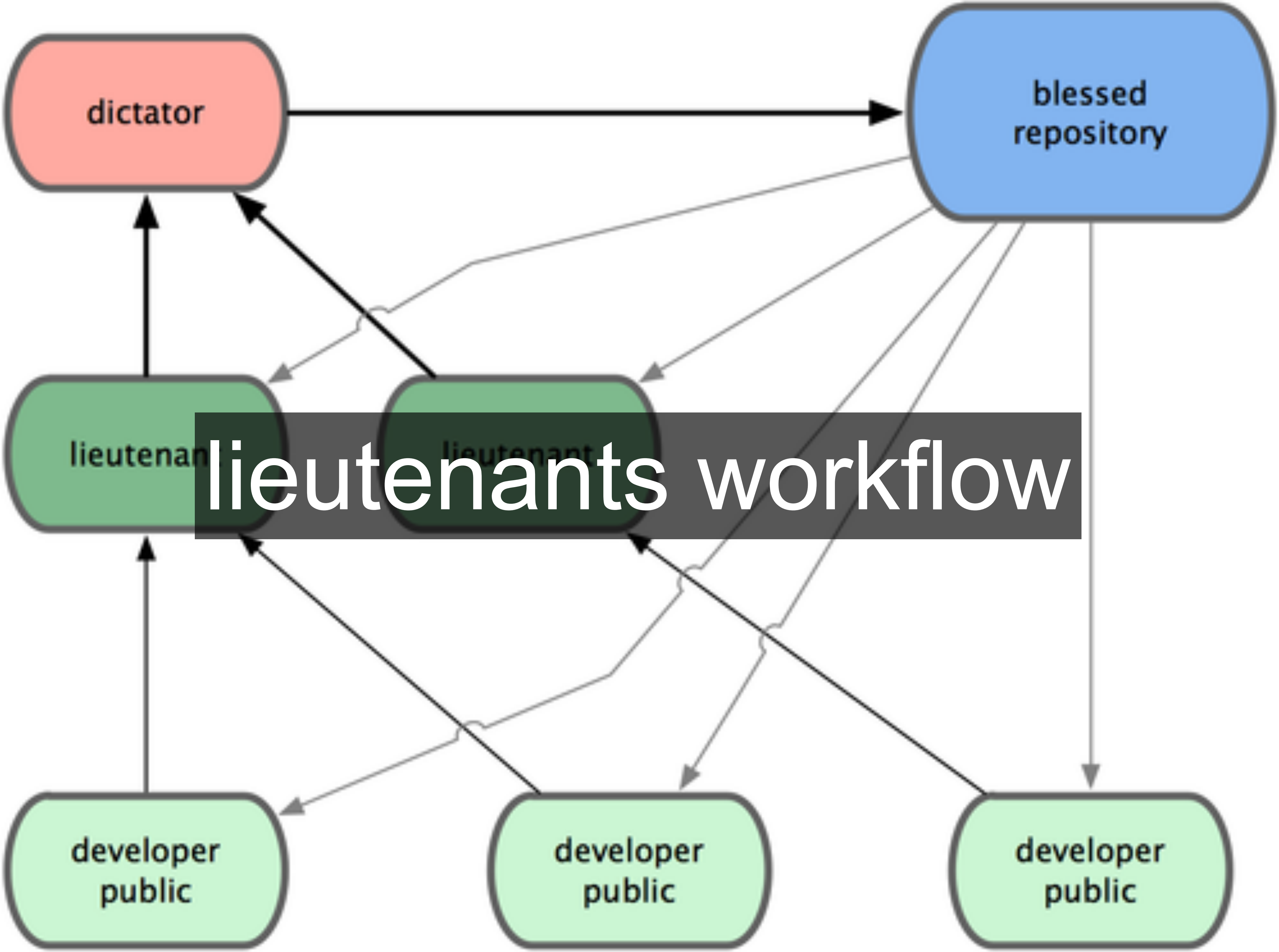
centralized workflow

This is more common for smaller companies

integration manager workflow



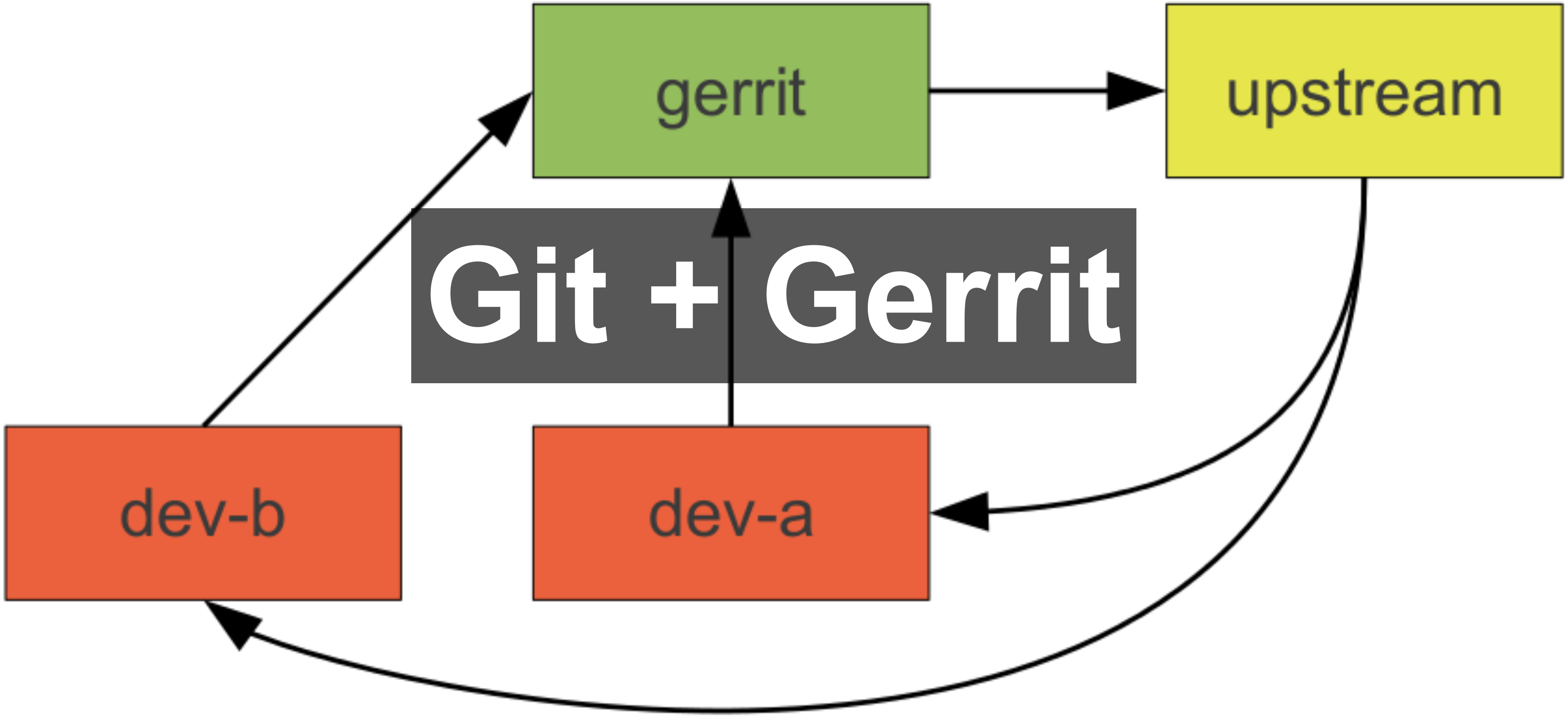
This is common for GitHub-based projects



lieutenants workflow

This is how the Linux kernel is developed

Git + Gerrit



Git and Gerrit

Documentation

Tyler Croy <rtyler.croy@mylookout.com>

Watched Changes Starred Changes

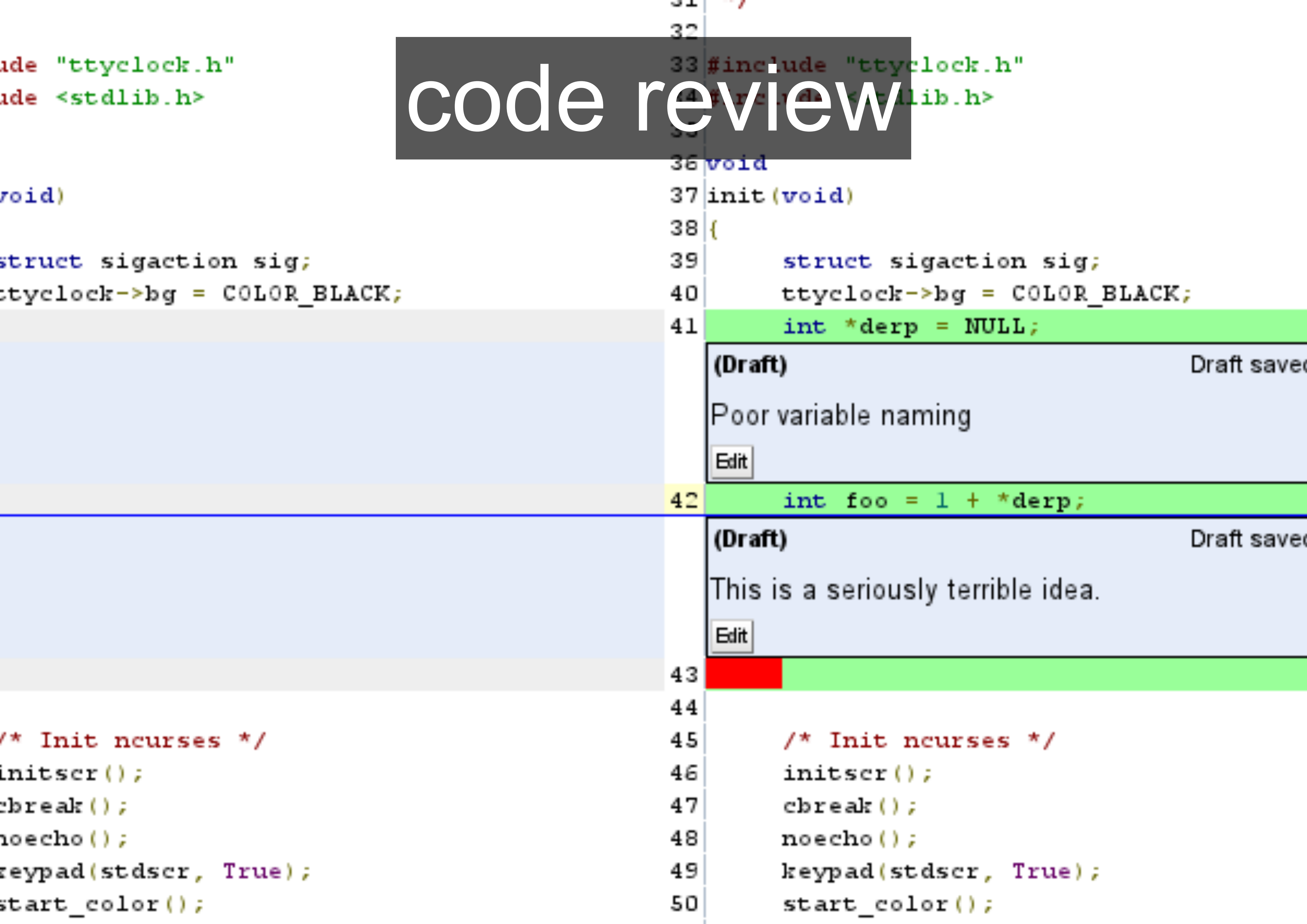
Dashboard for Tyler Croy

ject	Owner	Project	Branch
	Tyler Croy	flexd	master (frontend-errorlog)
	Tyler Croy	flexd	master (frontend-errorlog)
Tyler Croy			
		flexd	master
		flexd	master
		flexd	master
		flexd	master
(MERGED)		flexd	master (in-app-recurring)
(MERGED)		flexd	master (in-app-recurring)
(MERGED)		gems/newsclippings	master
(MERGED)		gems/newsclippings	master
(MERGED)		gems/newsclippings	master
(MERGED)		gems/newsclippings	master
(MERGED)	Tyler Croy	flexd	deploy-2012.07.19
(MERGED)	Tyler Croy	flexd	master
(MERGED)	Tyler Croy	flexd	deploy-2012.07.18
(MERGED)	Tyler Croy	flexd	deploy-2012.07.17
(MERGED)	Tyler Croy	flexd	master (SVR-7576)

Gerrit

Gerrit is a Git-based code review tool

code review



```
 */

#include "ttyclock.h"
#include <stdlib.h>

void
init(void)
{
     struct sigaction sig;
     ttyclock->bg = COLOR_BLACK;
     int *derp = NULL;
```

(Draft) Draft saved

Poor variable naming

Edit

```
     int foo = 1 + *derp;
```

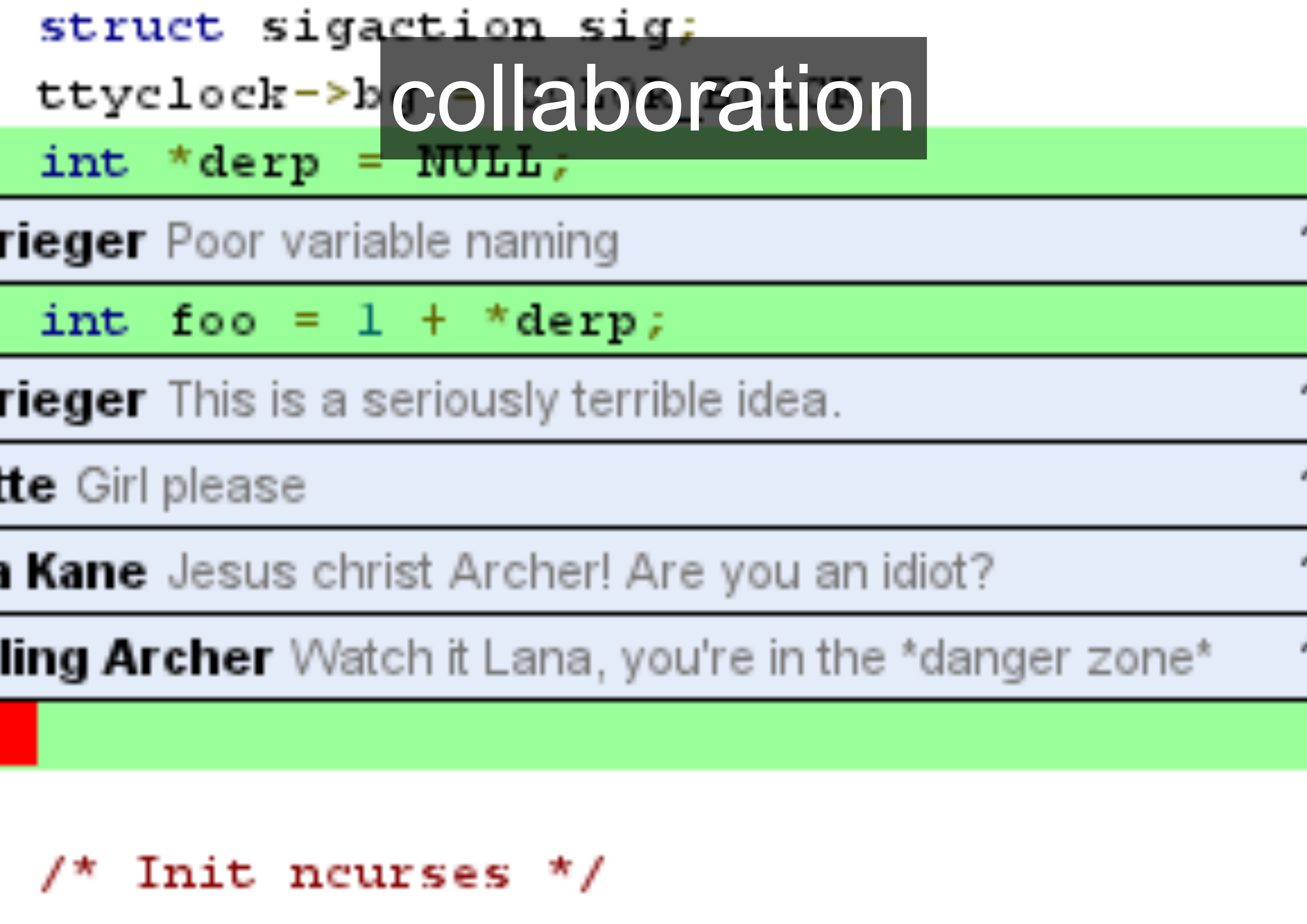
(Draft) Draft saved

This is a seriously terrible idea.

Edit

```


     /* Init ncurses */
     initscr();
     cbreak();
     noecho();
     keypad(stdscr, True);
     start_color();
```

```
ude "ttyclock.h"
ude <stdlib.h>

void)

struct sigaction sig;
ttyclock->bg = COLOR_BLACK;

/* Init ncurses */
initscr();
cbreak();
noecho();
eypad(stdscr, True);
start_color();
```



collaboration

```
struct sigaction sig;
ttyclock->bg = COLOR_BLACK;
int *derp = NULL;
```

rieger Poor variable naming

```
int foo = 1 + *derp;
```

rieger This is a seriously terrible idea.

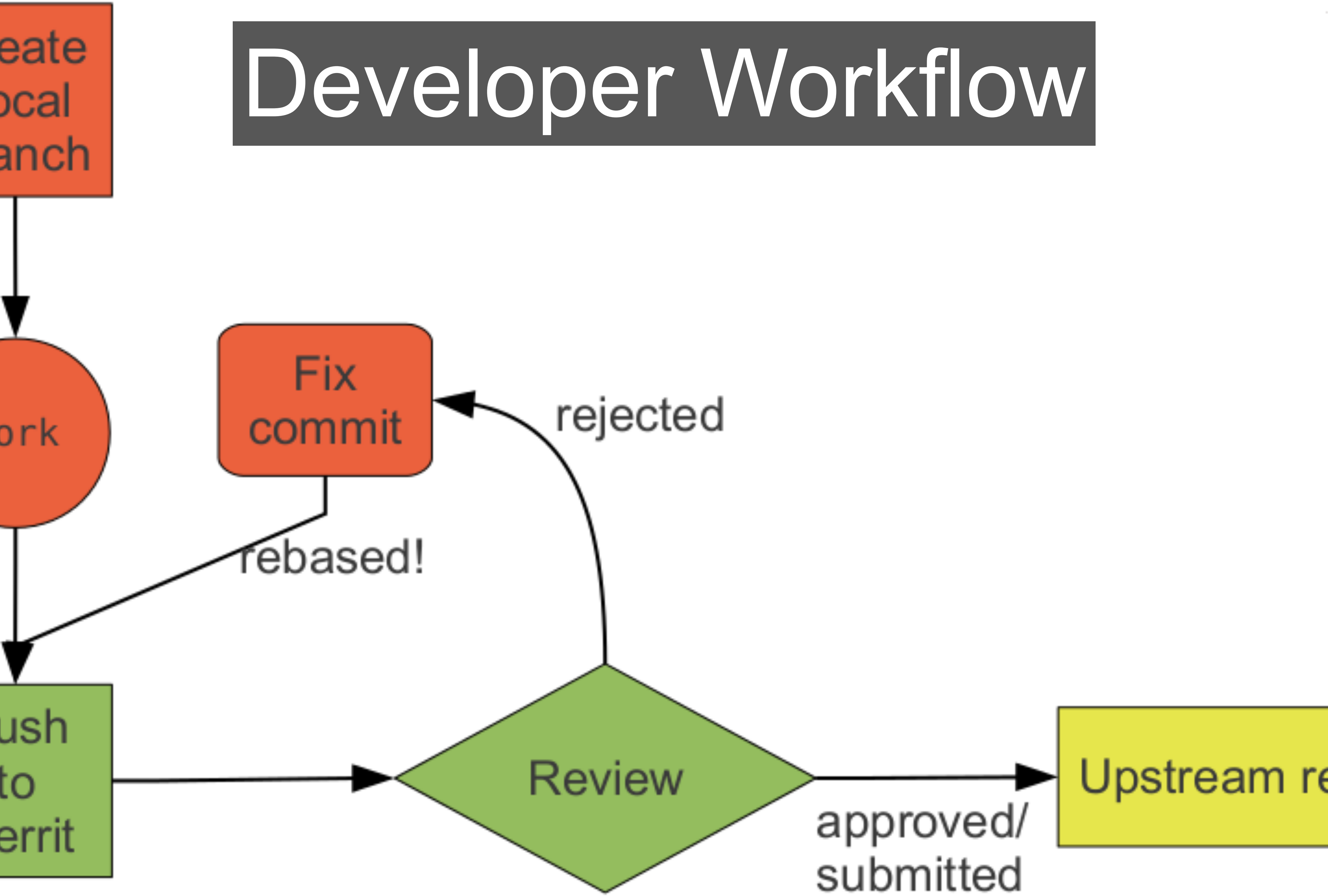
tte Girl please

a Kane Jesus christ Archer! Are you an idiot?

ling Archer Watch it Lana, you're in the *danger zone*

```
/* Init ncurses */
```

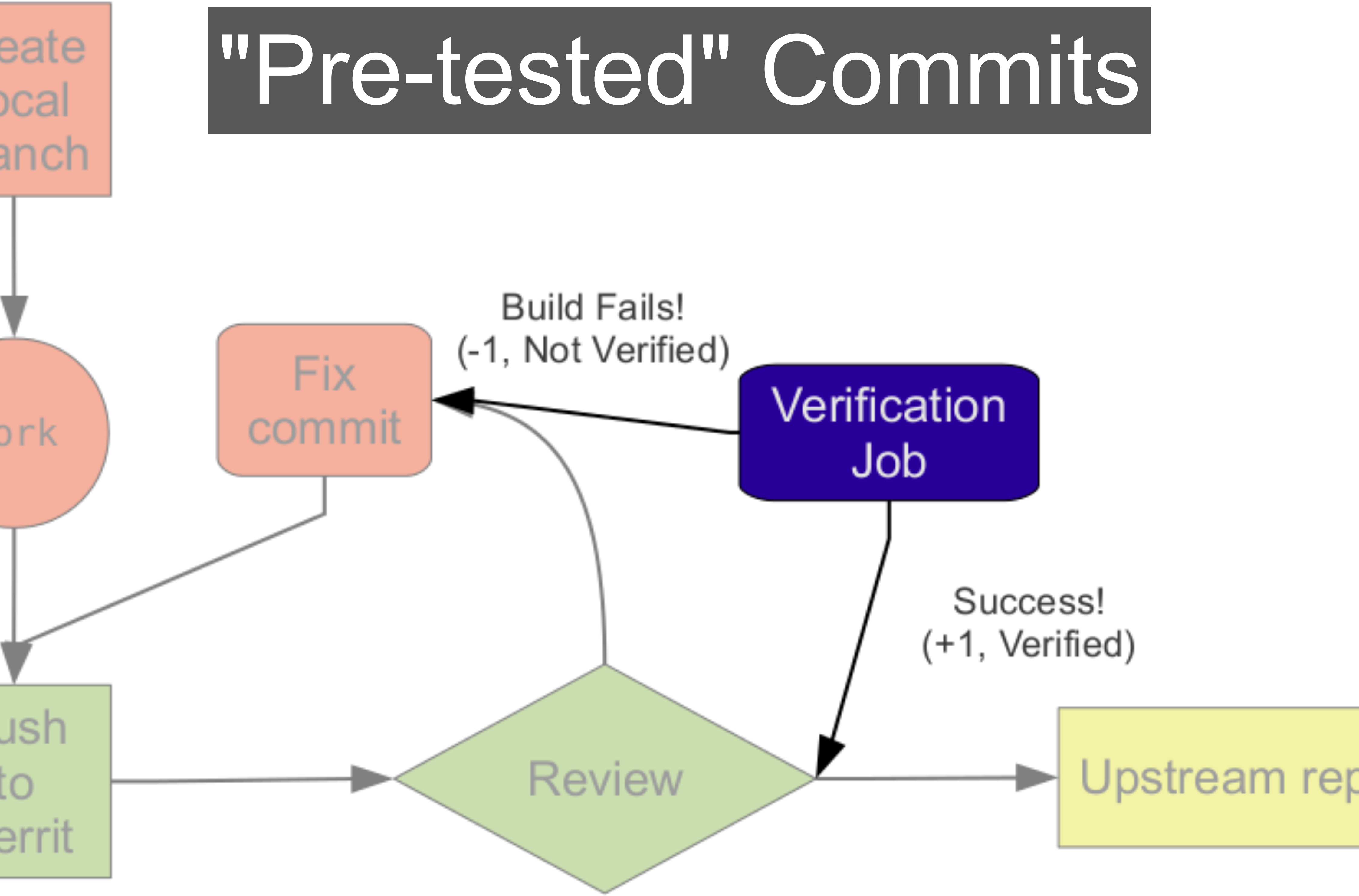
Developer Workflow



What this means for an individual developer is that they can iterate on their code in Gerrit, based on feedback from their colleagues.

Once the code is all polished up, it can then be integrated into the "main" repository

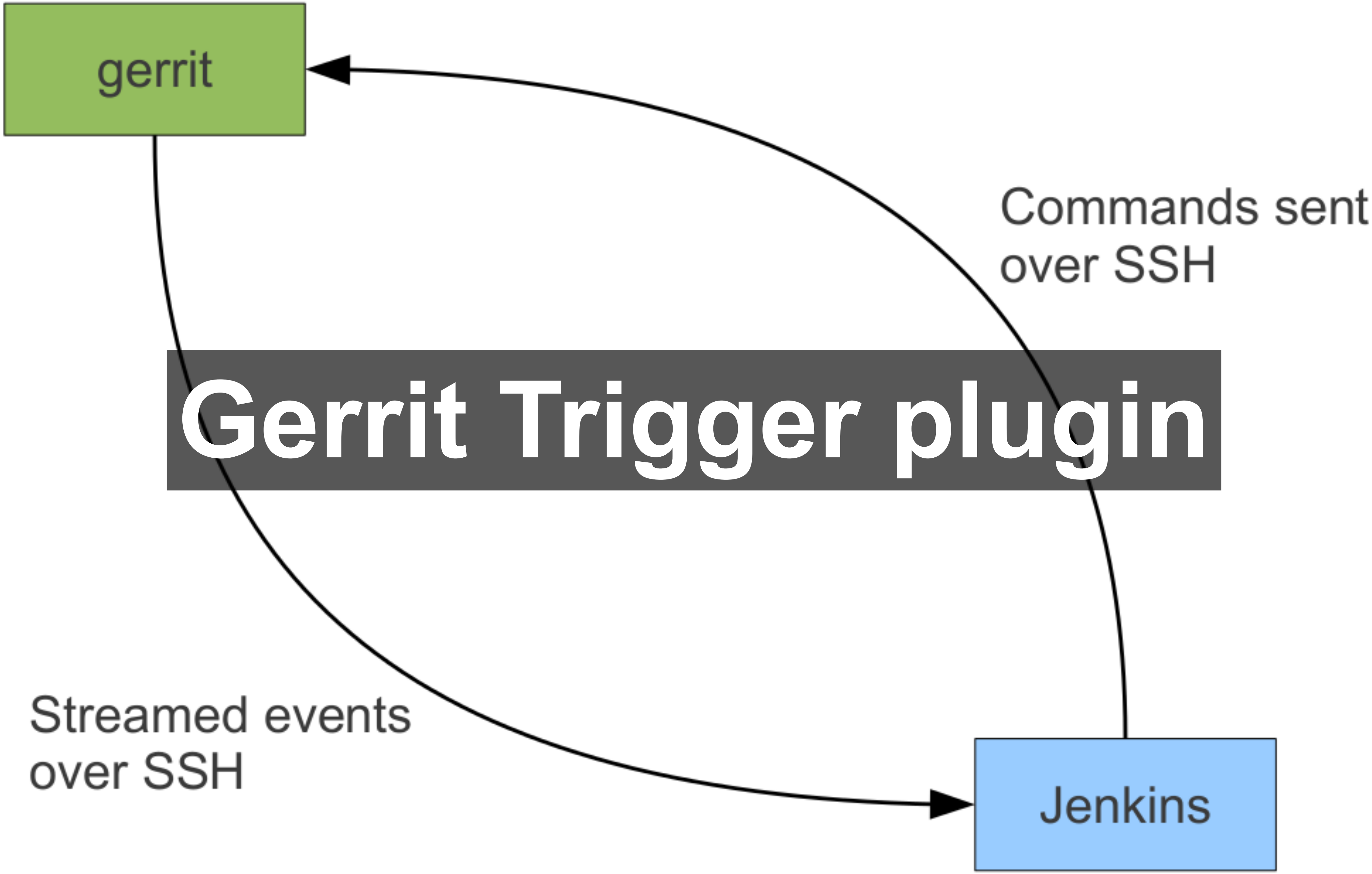
"Pre-tested" Commits



An integral part of our Git + Gerrit workflow involved pre-testing commits.

The whole concept behind "pre-testing" a commit is that only changes which have passed the "tests" will be allowed to be integrated or merged.

Gerrit Trigger plugin



gerrit

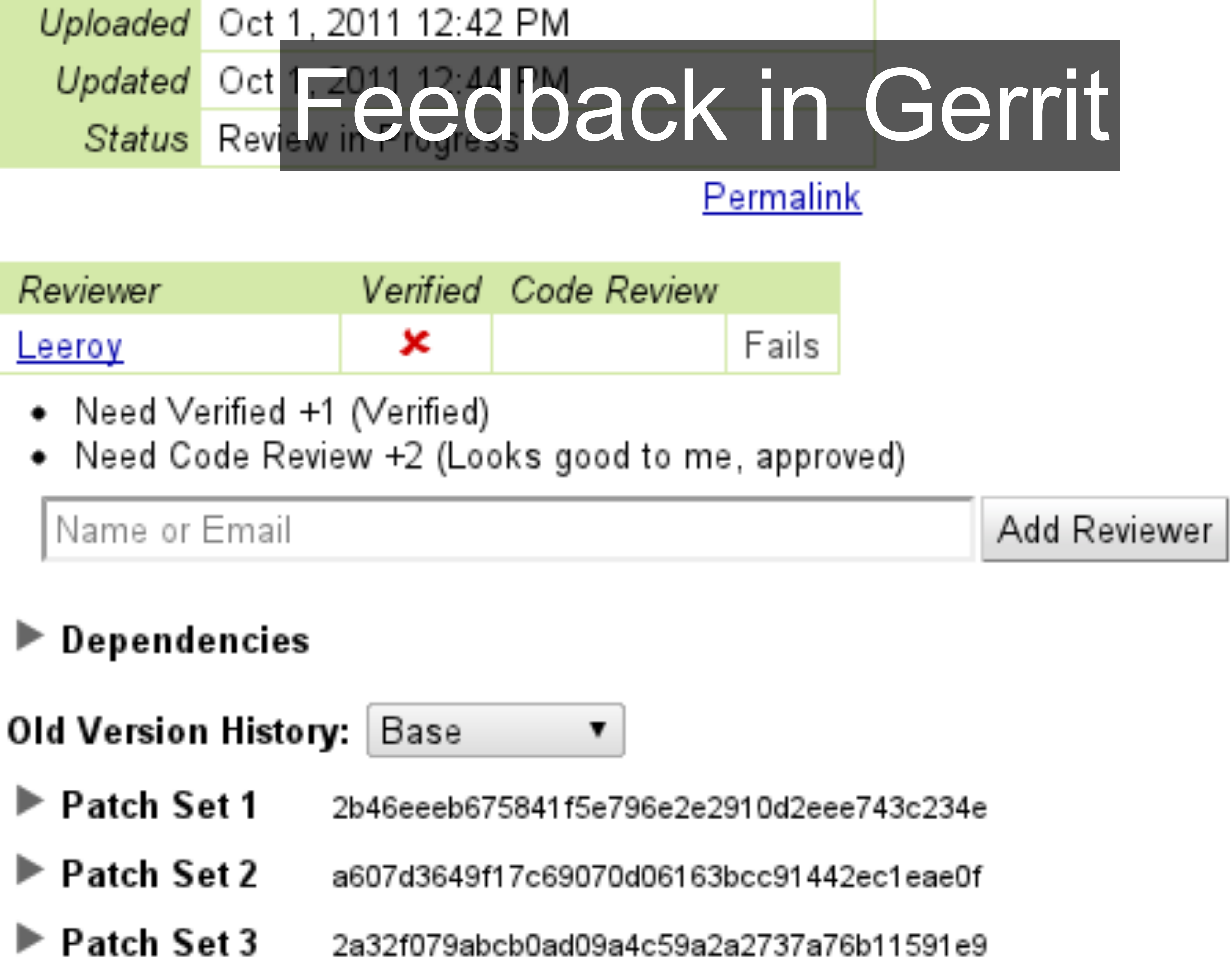
Commands sent
over SSH

Streamed events
over SSH

Jenkins

Gerrit Trigger plugin

Feedback in Gerrit



Uploaded	Oct 1, 2011 12:42 PM
Updated	Oct 1, 2011 12:44 PM
Status	Review in Progress

Permalink

Reviewer	*Verified*	*Code Review*	
Leeroy	✗		Fails

- Need Verified +1 (Verified)
- Need Code Review +2 (Looks good to me, approved)

▶ Dependencies

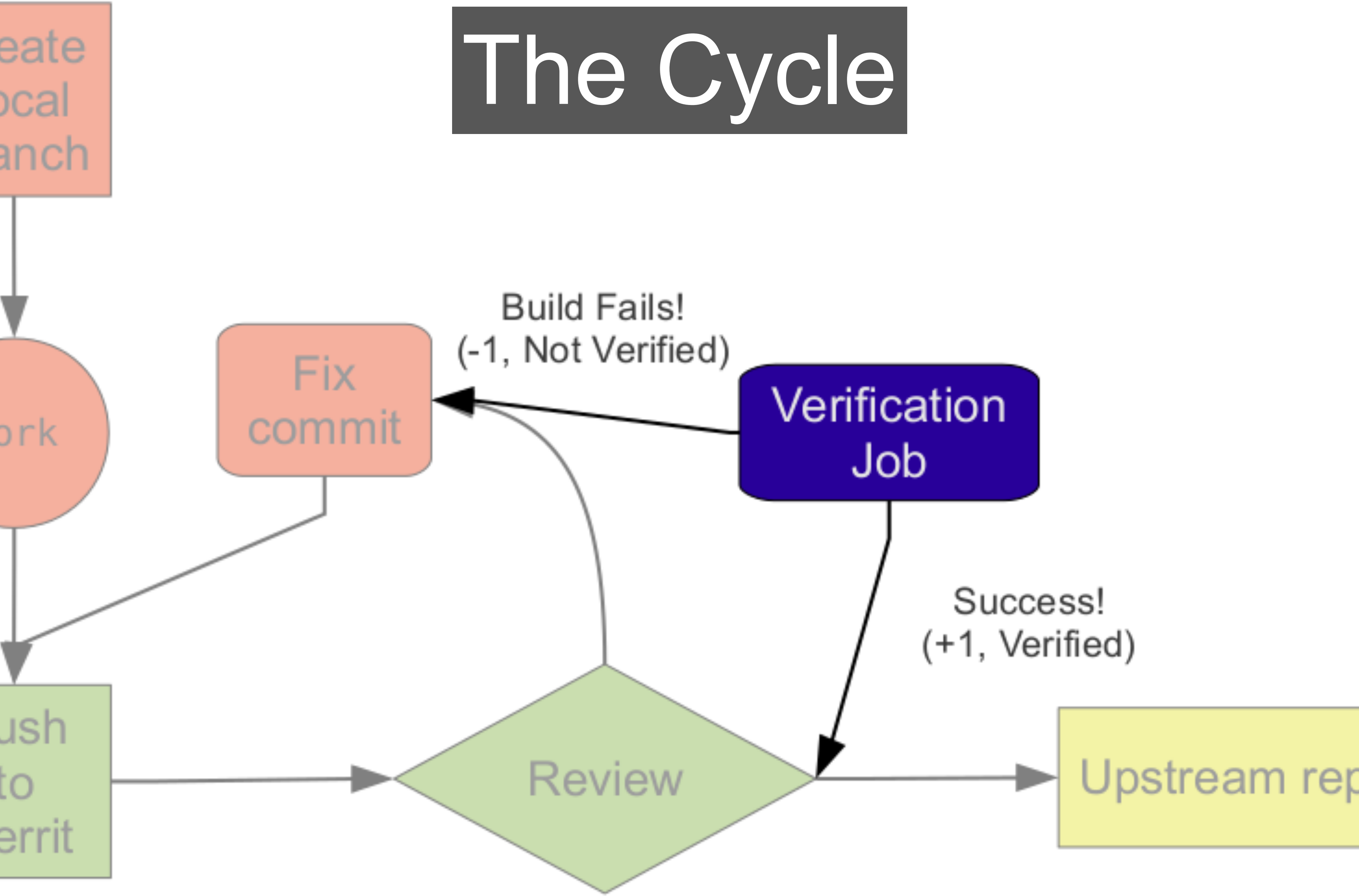
Old Version History: Base ▼

▶ Patch Set 1 2b46eeeb675841f5e796e2e2910d2eee743c234e

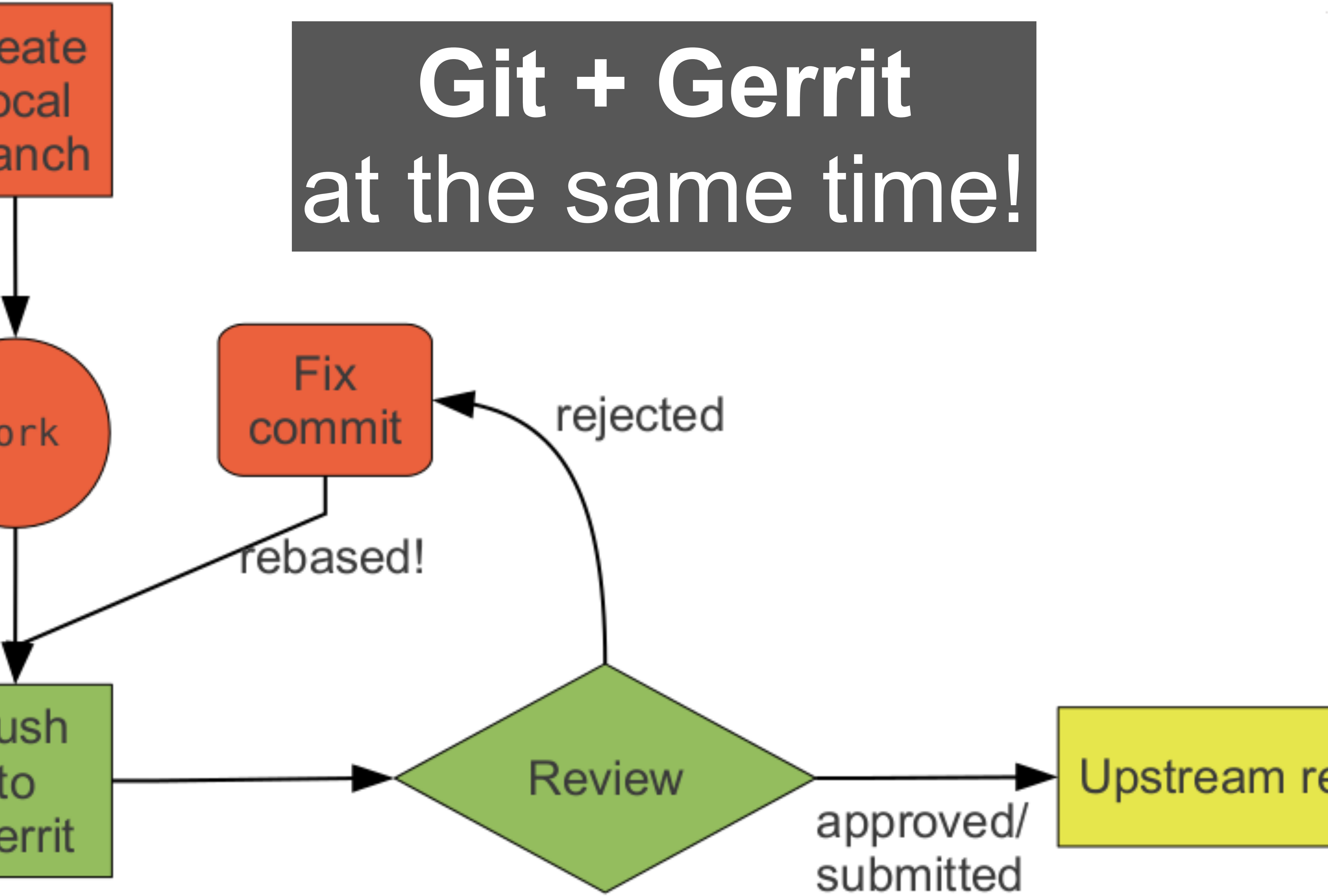
▶ Patch Set 2 a607d3649f17c69070d06163bcc91442ec1eae0f

▶ Patch Set 3 2a32f079abcb0ad09a4c59a2a2737a76b11591e9

The Cycle


eate
ocal
anch
ork
Fix commit
Build Fails!
(-1, Not Verified)
Verification Job
Success!
(+1, Verified)
ush
to
errit
Review
Upstream rep

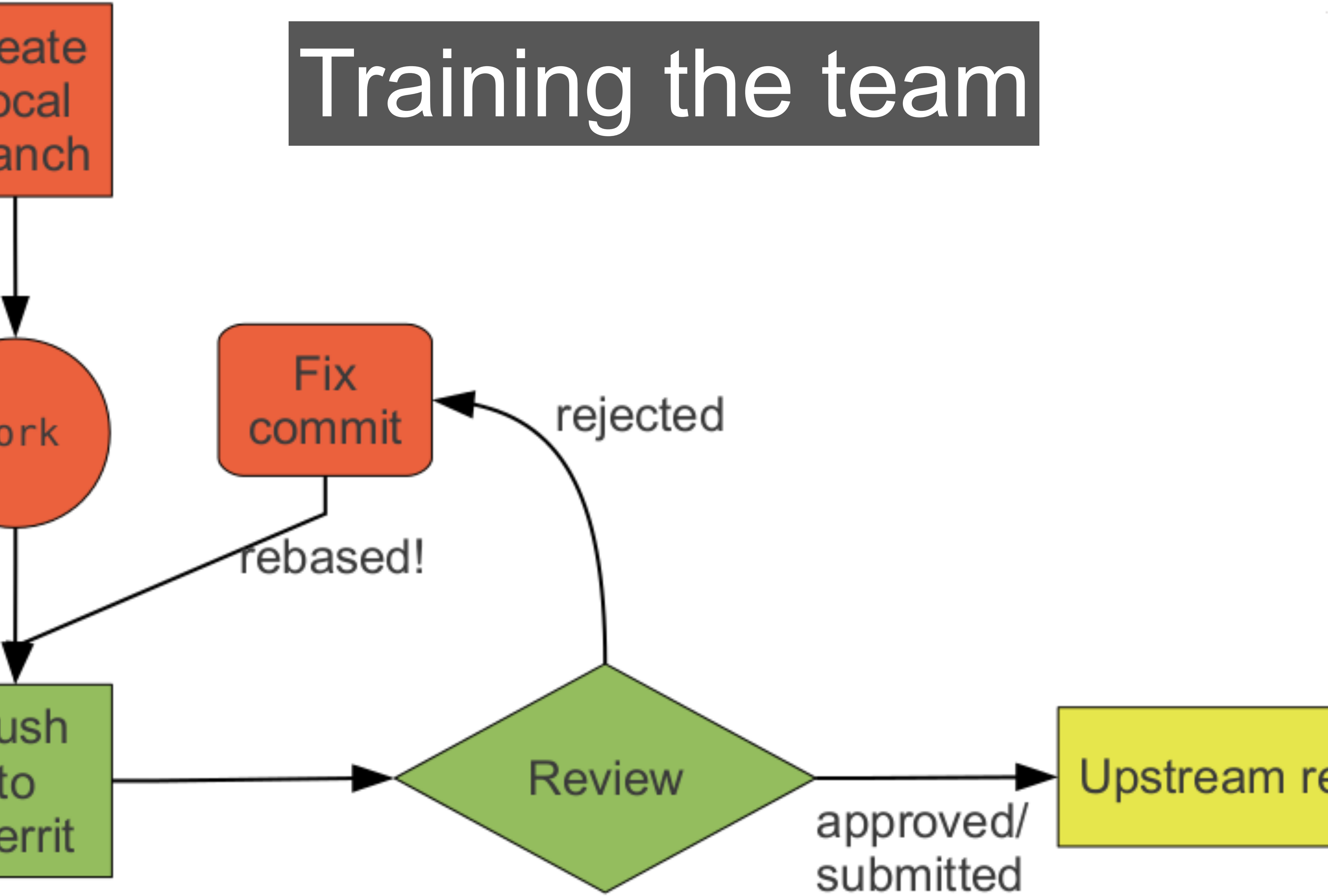
Git + Gerrit at the same time!



TODO: Perhaps this should go after the pre-tested commits bit
We switched from Subversion directly to Git and Gerrit, all at once.

Instead of introducing Git as a separate tool to developers, we introduced at the same time so developers never learned a Git-based workflow that *didn't* involve Gerrit at its core.

Training the team



We scheduled 3 different 1 hour training sessions with various groups of engineers in order to provide a hands-on walk-through of the Git + Gerrit workflow

This included a fully set-up "demo" project to use for experimentation of creating commits, code reviewing them, verifying them with Jenkins and finally merging them into the "master" bracnh

During the course of these training sessions, we used the feedback and common problems encountered by engineers to fill out a "getting started" wiki page which new hires now use to come up to speed with Git + Gerrit.

"We need more builders!"

TODO: Make this less out of place
When we first started moving things into Jenkins we had 3 slaves that were properly configured for running our tests. As we started using pre-tested commits with Gerrit and Jenkins, we *very* quickly realized that we needed

In the very early stages of this, we used the same hand-crafted VM base image with VMWare ESX, and then spun up multiple new machines.

This process has changed a bit since then, which I'll dig into in a moment

Recap

TODO: Diagram and cover "the world thus far"

Step Three
Automate *Everything*

Deploying the test environment

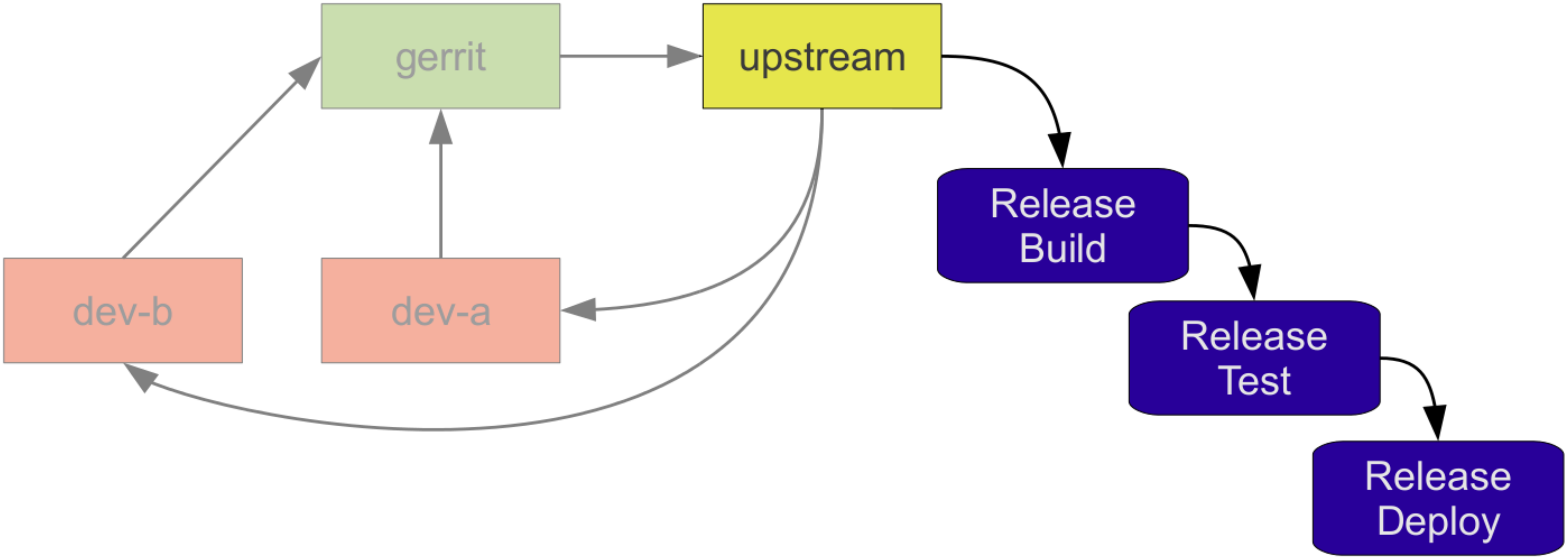
Once the deployment of our test environment was managed through Jenkins,
we created pipelines with Jenkins, chaining off of a successful
deployment to the test environment.

TODO: Discuss selenium testing/SI testing after QA deploy

New kinds of tests!

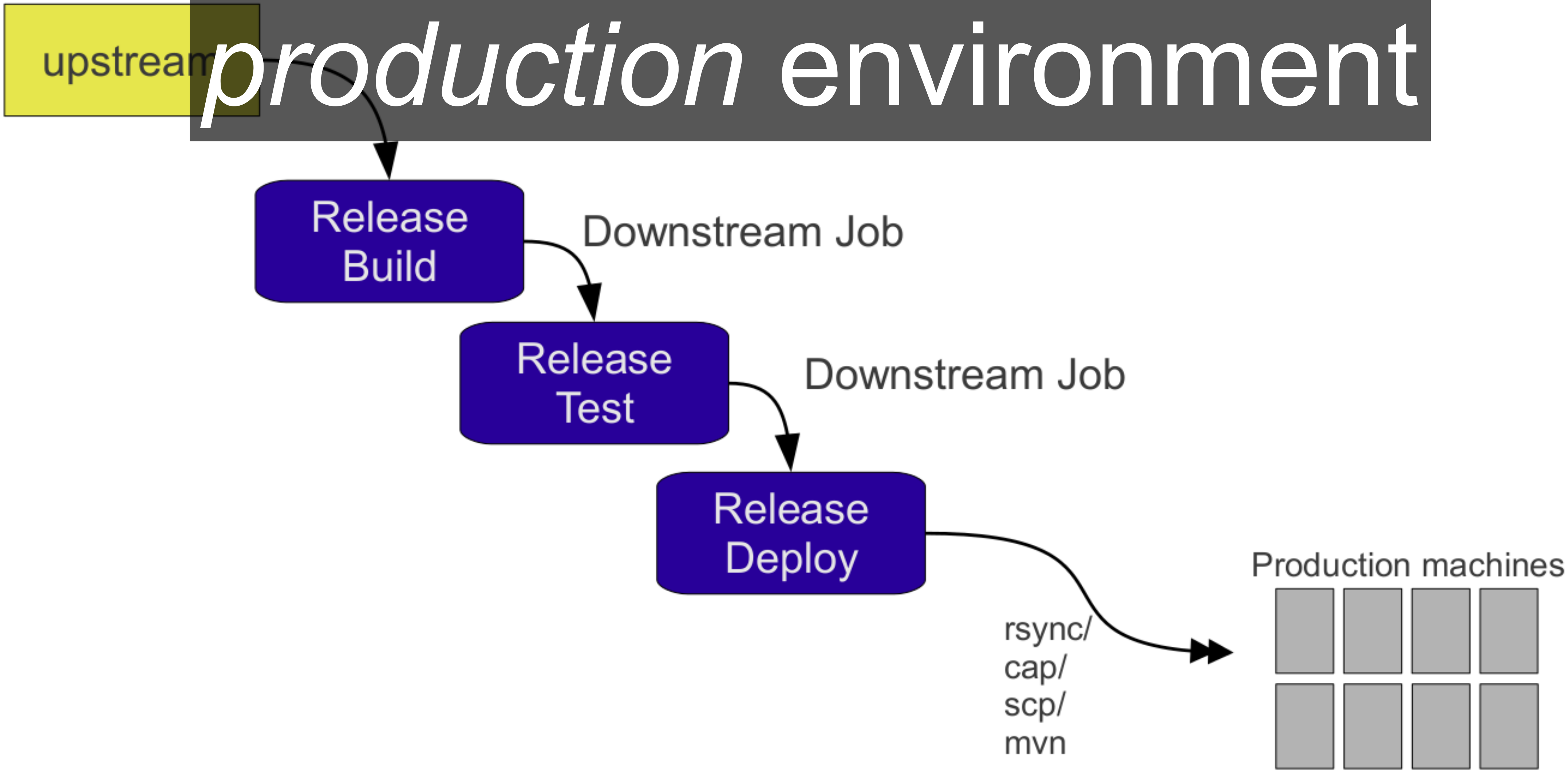
TODO: Discuss Jasmine/Selenium tests with Sauce Labs

Automating **deployment**



First automating with `infra_update_faithful2` then moving into the `infra_deploy_qa` territory. Finally automating the actual deployment of production

Deploying to the *production* environment



TODO: Discuss the use of build promotions to stage, deploy and finally mark the deployment as successful

Step Four
More Robots

OpenStack
and the
jclouds plugin

TODO: Discuss the investment in OpenStack for adding more build capacity.
Looking forward to the work done with the jclouds plugin

Slave Management *with* Puppet

TODO: discuss Puppetizing the slave machines

TODO

(*nobody's perfect*)

* Haven't automated the rollback of production
* Not nearly enough stages/production clusters to run Selenium tests per-commit
* Not enough real-time feedback from production
* No production acceptance testing
* Tests take a LONG time
* Still only doing a single daily deployment, facing the same issues we did before with 2 week deployment, but on a smaller scale

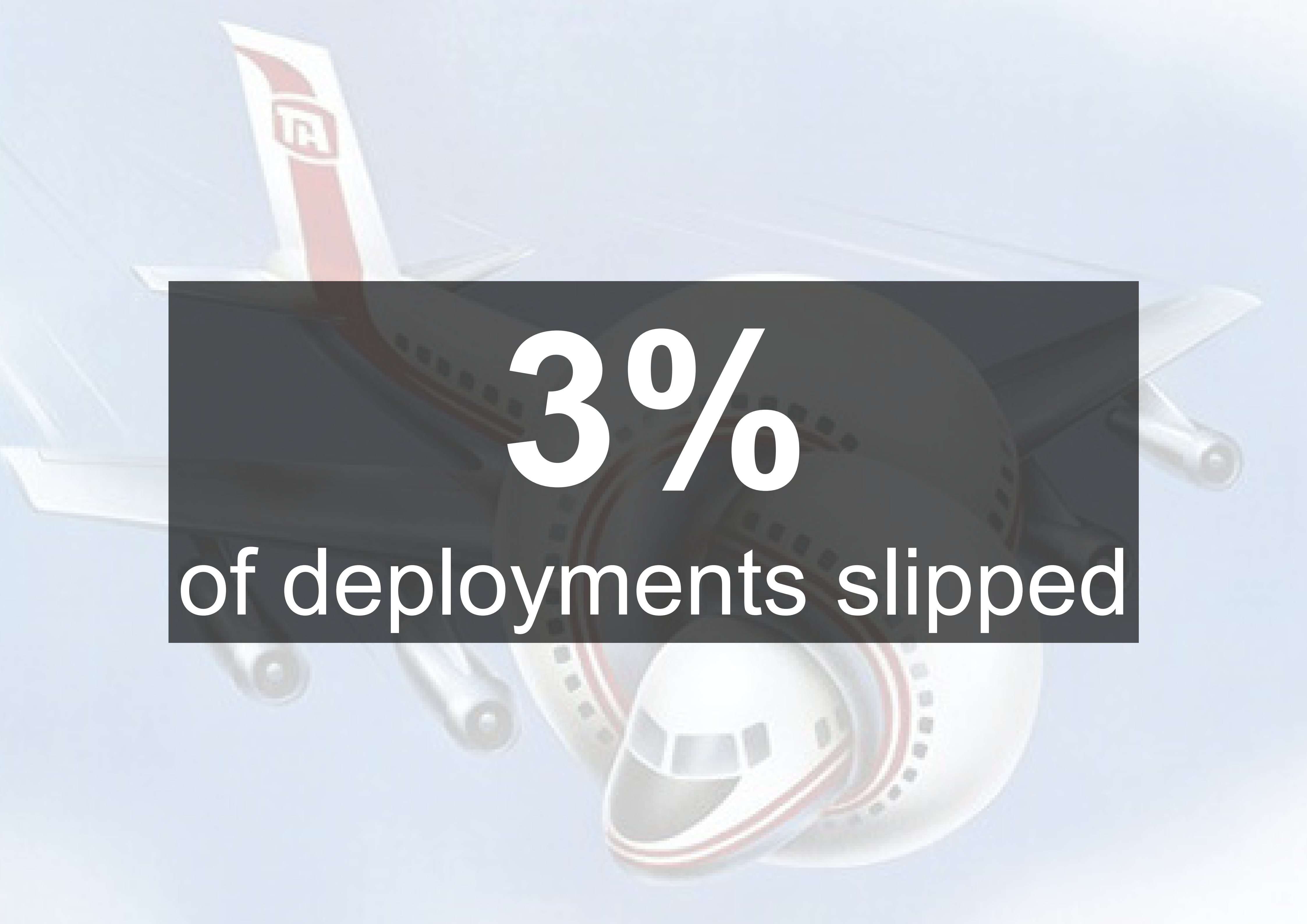
Happiness with Numbers

2%

of deployments failed

14

commits per deployment



3%

of deployments slipped

(*that* ***rocks!***)

Questions?