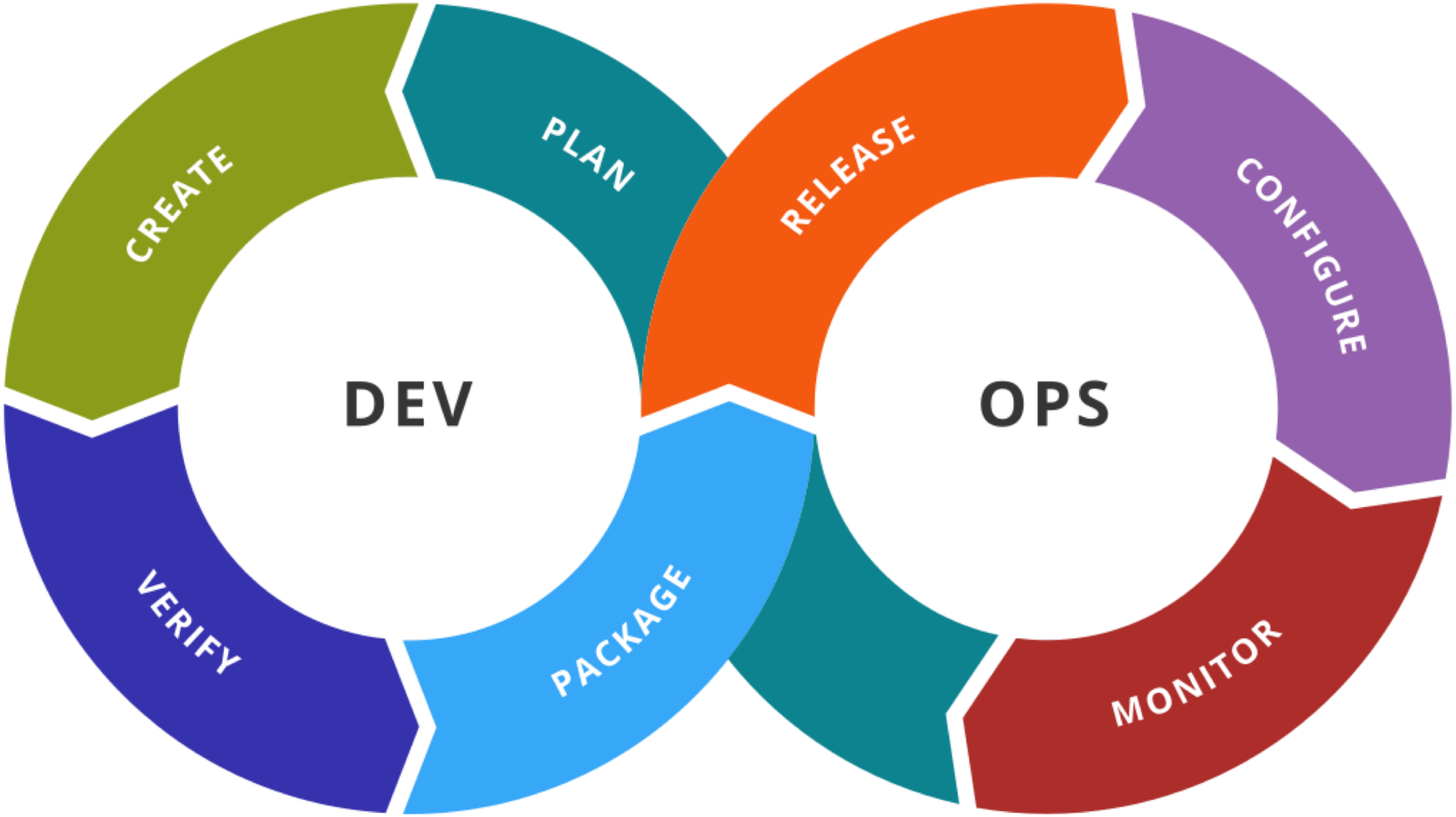
Principles of Software Construction: Objects, Design, and Concurrency

DevOps continued and Ethics

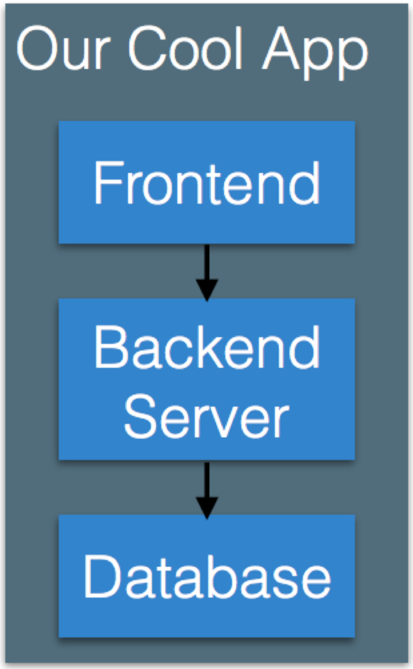
Michael Hilton Bogdan Vasilescu

Administrivia

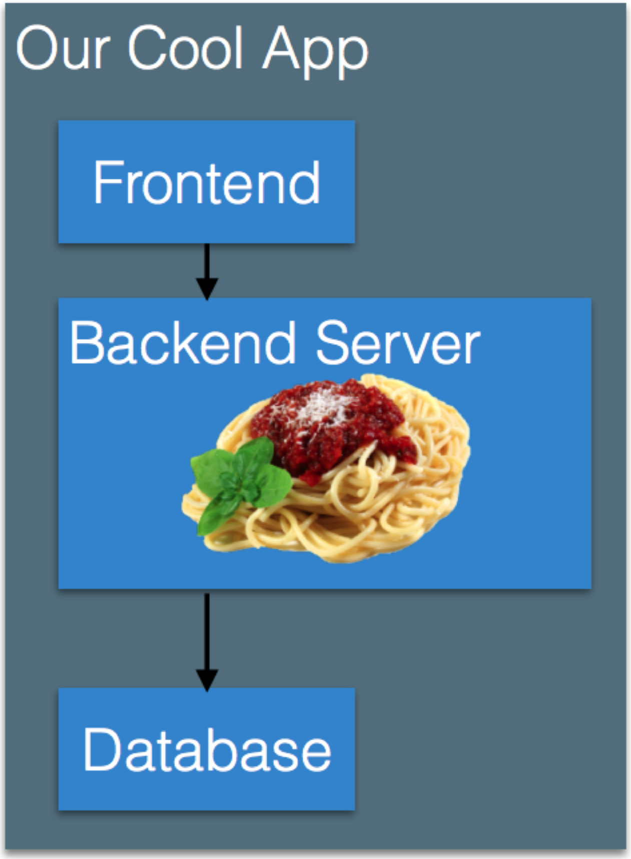
- Final Exam: Monday, May 6, 2019 05:30 p.m. - 08:30 p.m.
  - **LOCATION: GHC 4401**
  - Review Session Saturday, May 4th, 1-3pm in NSH 3305


CREATE
PLAN
RELEASE
CONFIGURE
DEV
OPS
VERIFY
PACKAGE
MONITOR

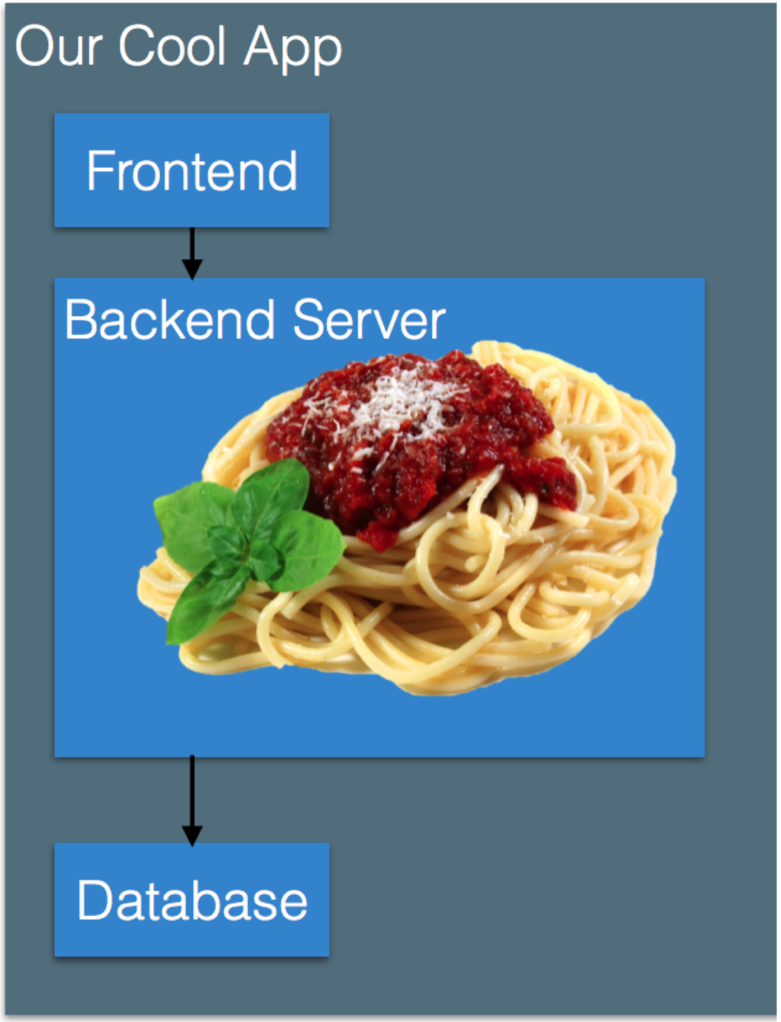
Simple Layers App



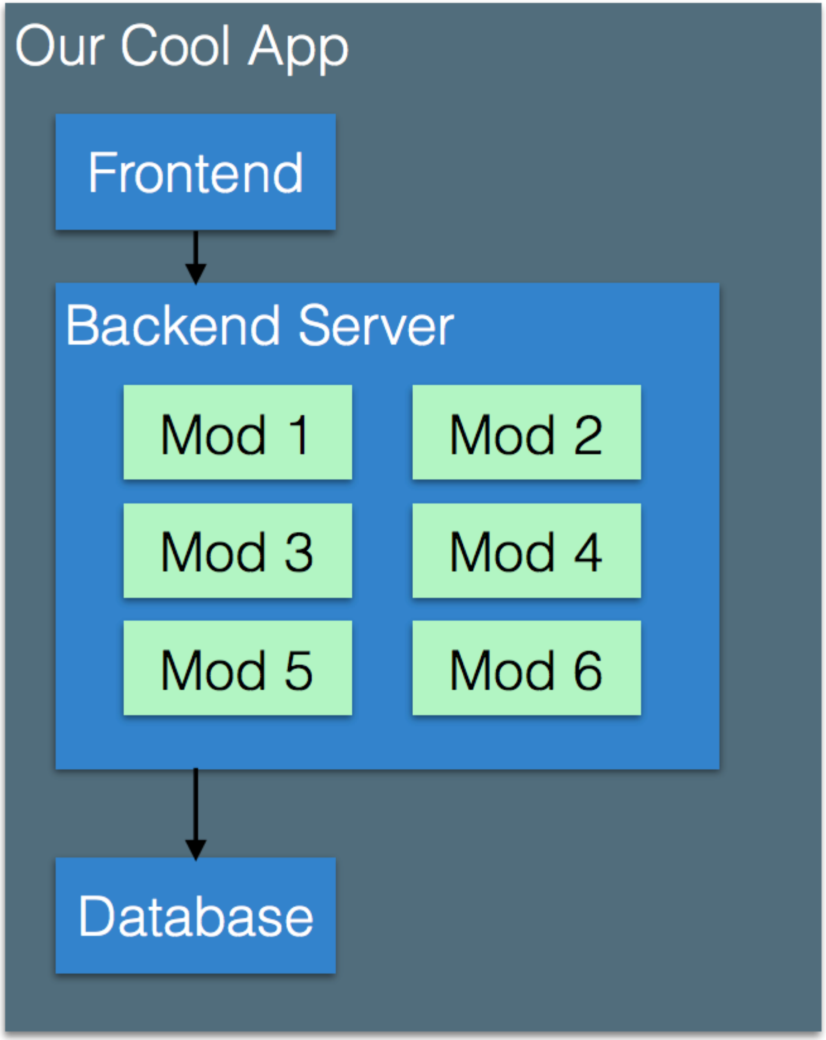
More functionality



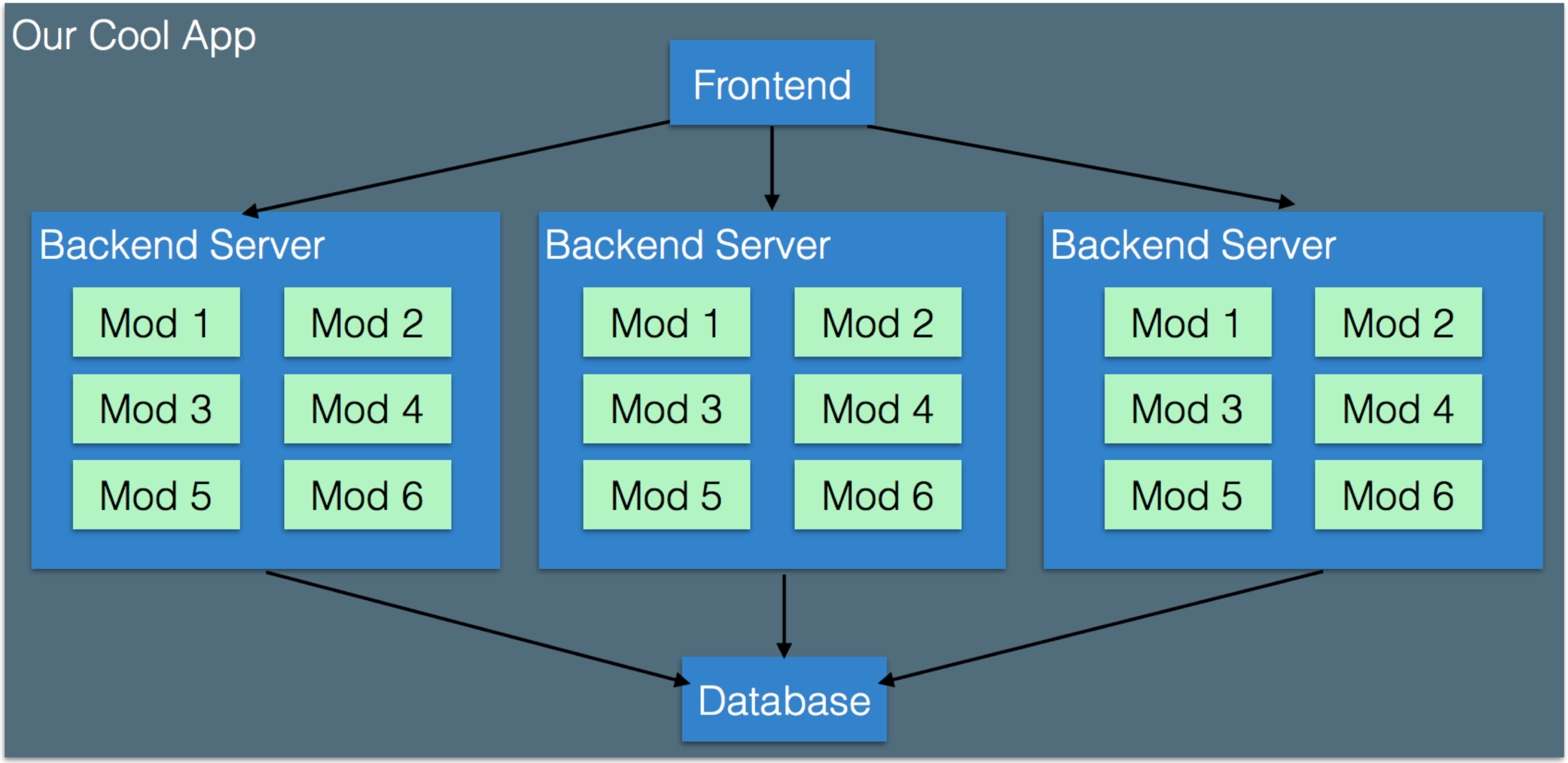
Even more functionality



Organize our backend

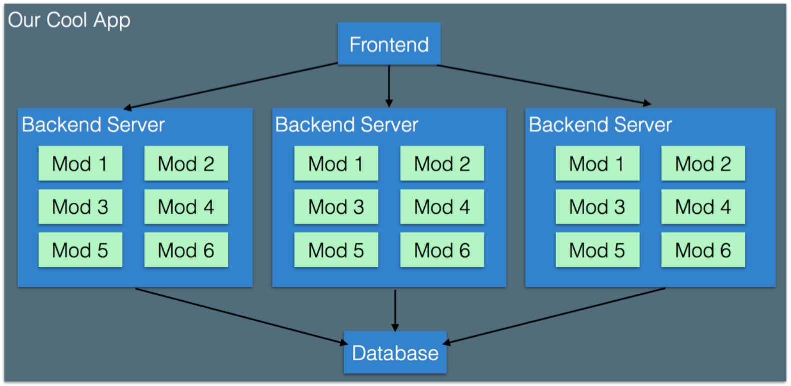


How to scale?

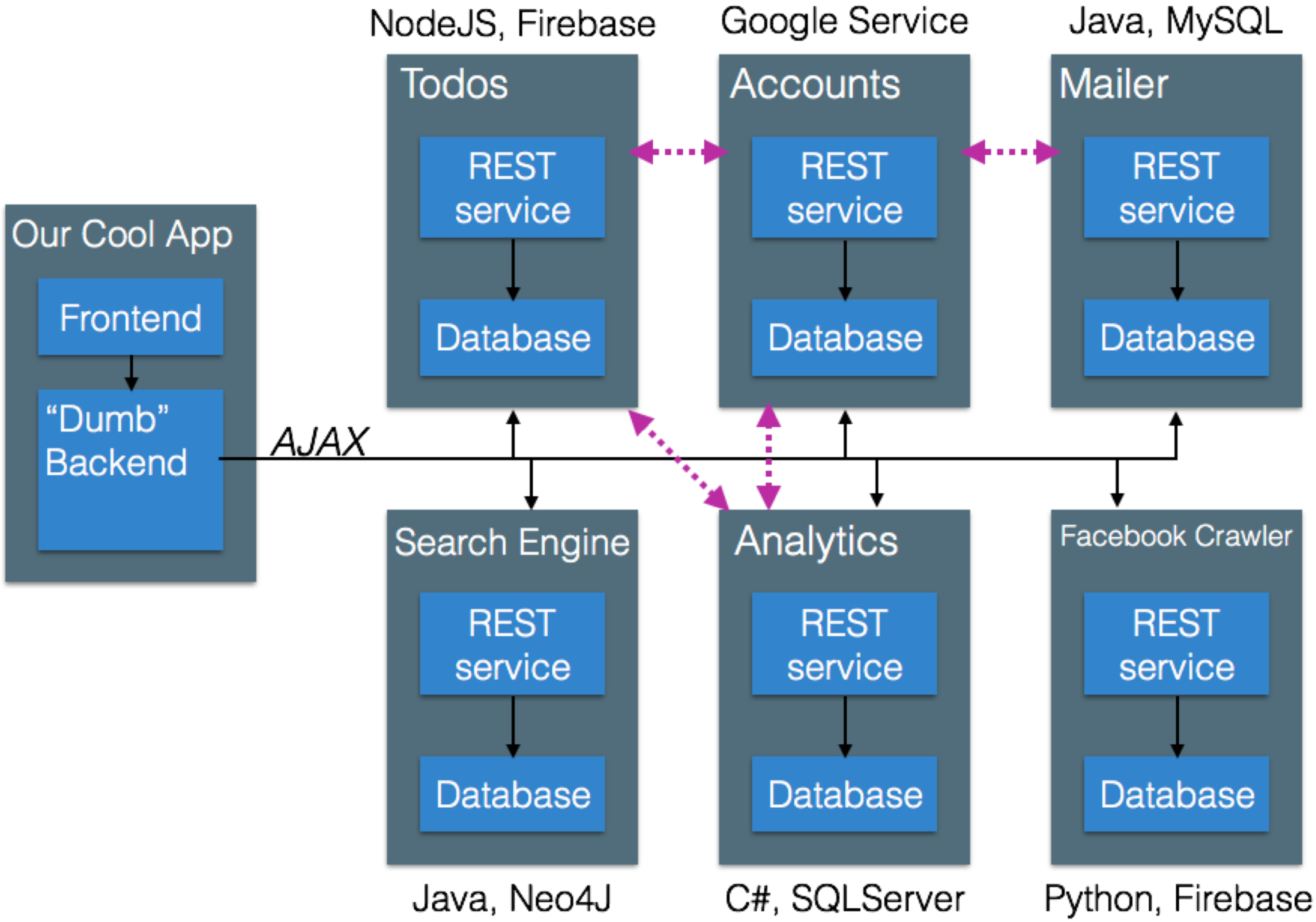

Our Cool App
Frontend
Backend Server
Mod 1
Mod 2
Mod 3
Mod 4
Mod 5
Mod 6
Backend Server
Mod 1
Mod 2
Mod 3
Mod 4
Mod 5
Mod 6
Backend Server
Mod 1
Mod 2
Mod 3
Mod 4
Mod 5
Mod 6
Database

Monolith

- What happens when we need 100 servers?
- What if we don't use all modules equally?
- How can we update individual models?
- Do all modules need to use the same DB, language, runtime, etc?

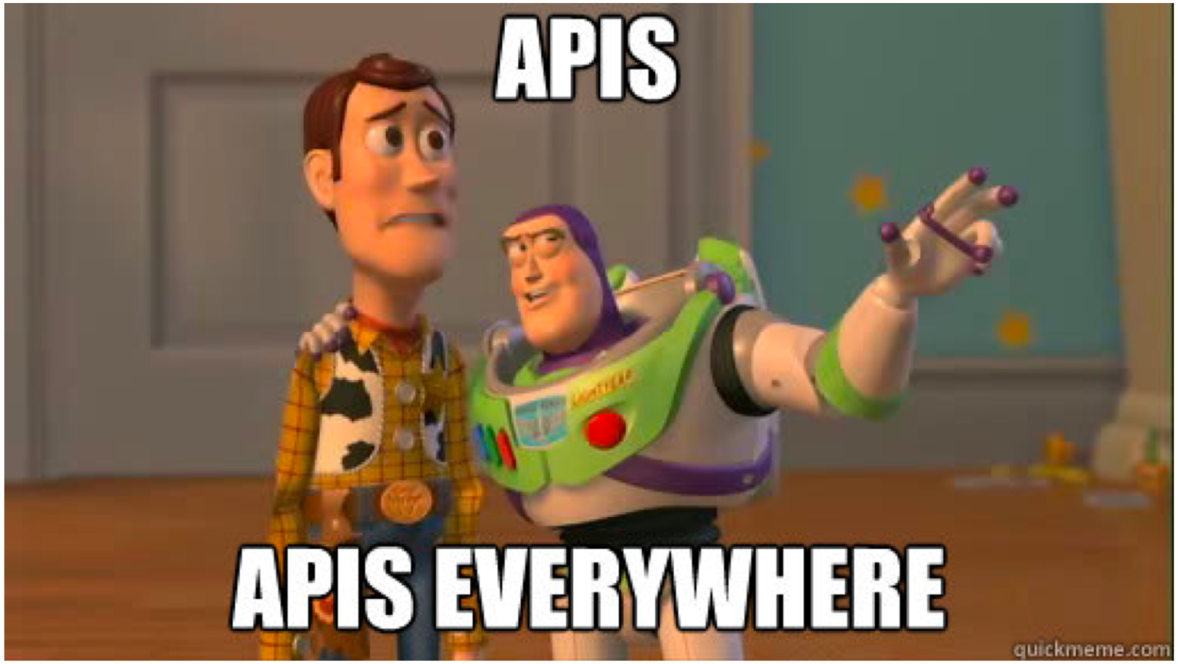


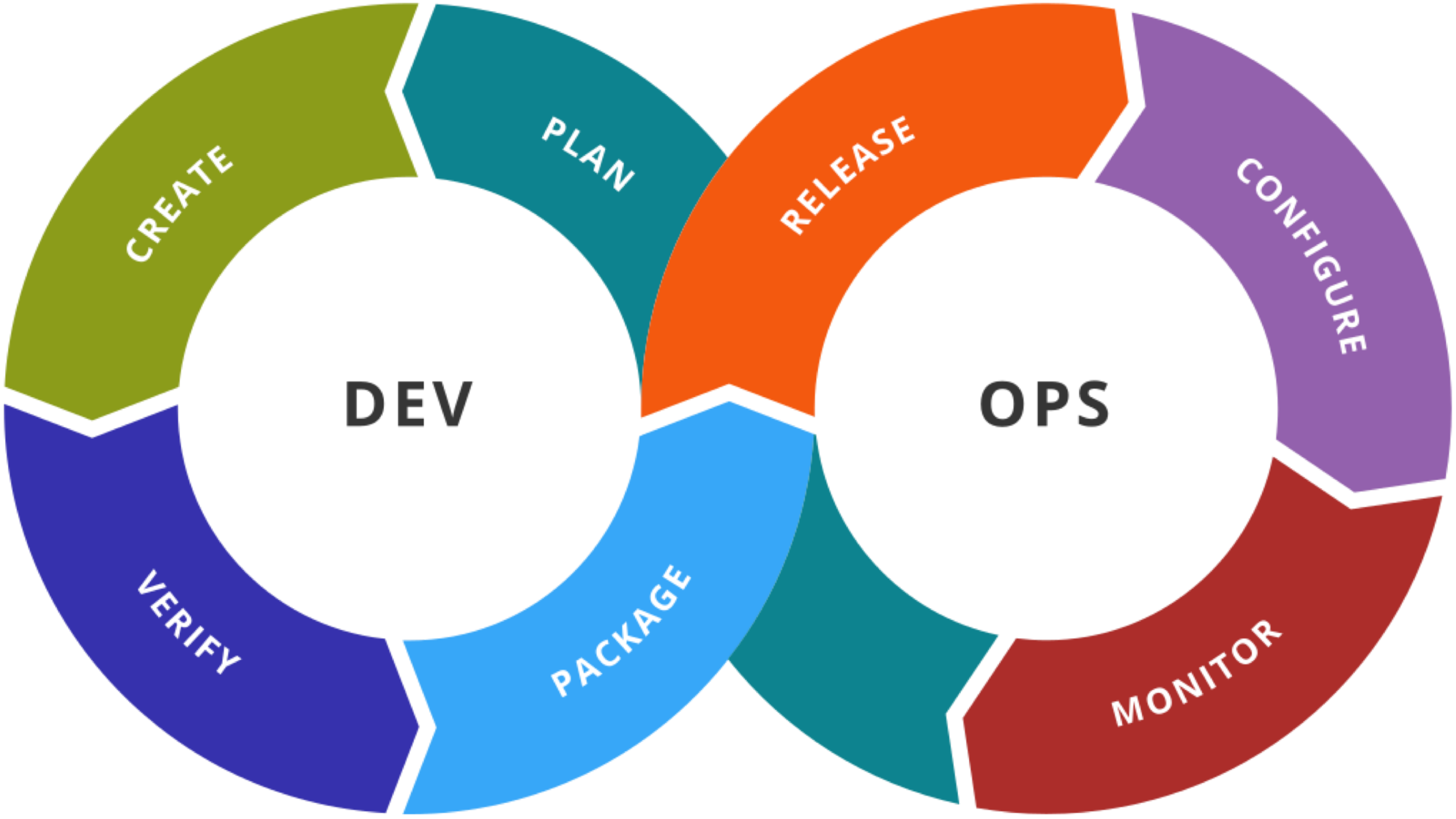
Microservices



Microservice costs

- Distribution
- Eventual Consistency
- Operational complexity
- Leads to more API design decisions



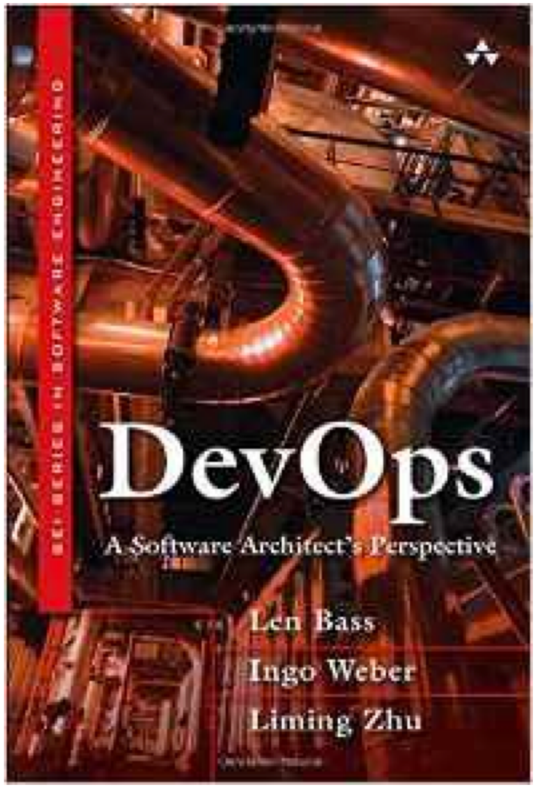

CREATE
PLAN
DEV
VERIFY
PACKAGE
RELEASE
CONFIGURE
OPS
MONITOR

Why DevOps?

- Developers and Operations don’t have the same goals
  - Devs want to push new features
  - Ops wants to keep the system available (stable, tested, etc.)s
- Poor communication between Dev and Ops
- Limited capacity of operations staff
- Want to reduce time to market for new features
- Reduce “Throw it over the fence” syndrome

DevOps Definition

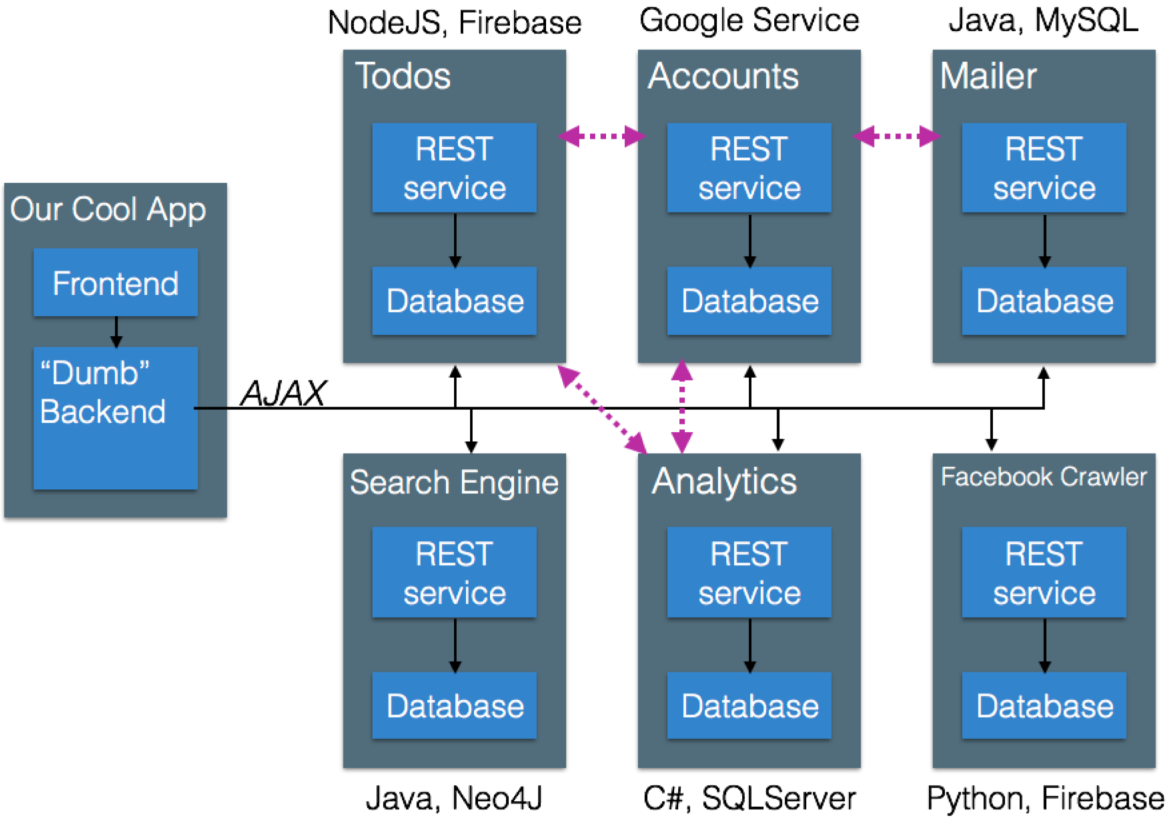
- “DevOps is a set of practices intended to reduce the time between committing a change to a system and the change being placed into normal production, while ensuring high quality.”


SEI SERIES IN SOFTWARE ENGINEERING
DevOps
A Software Architect's Perspective
Len Bass
Ingo Weber
Liming Zhu

What are implications of DevOps?

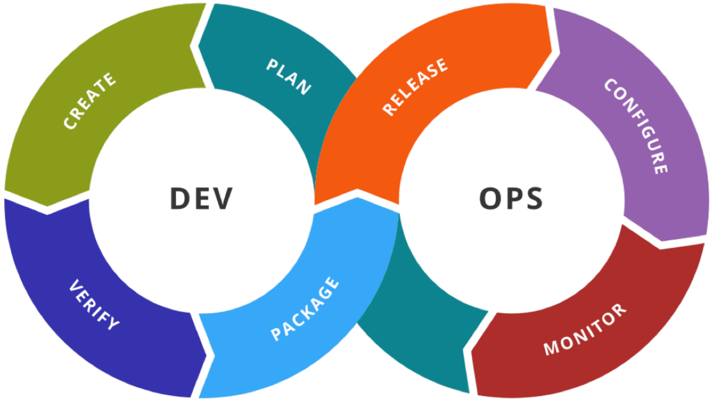
- Quality of the code must be high
  - Testing
- Quality of the build & delivery mechanism must be high
  - Automation & more testing
- Time is split:
  - From commit to deployment to production
  - From deployment to acceptance into normal production
- Goal-oriented definition
  - May use agile methods, continuous deployment (CD), etc.
  - Likely to use tools
- Achieving it starts before committing

Microservices rely on DevOps



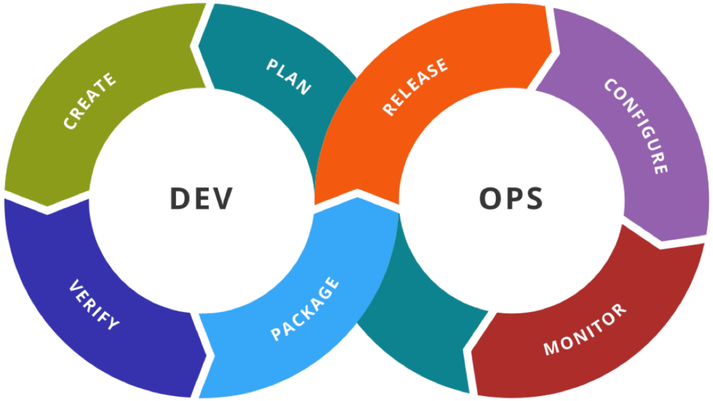
DevOps Toolchain

- Code — code development and review, source code management tools, code merging
- Build — continuous integration tools, build status
- Test — continuous testing tools that provide feedback on business risks
- Package — artifact repository, application pre-deployment staging



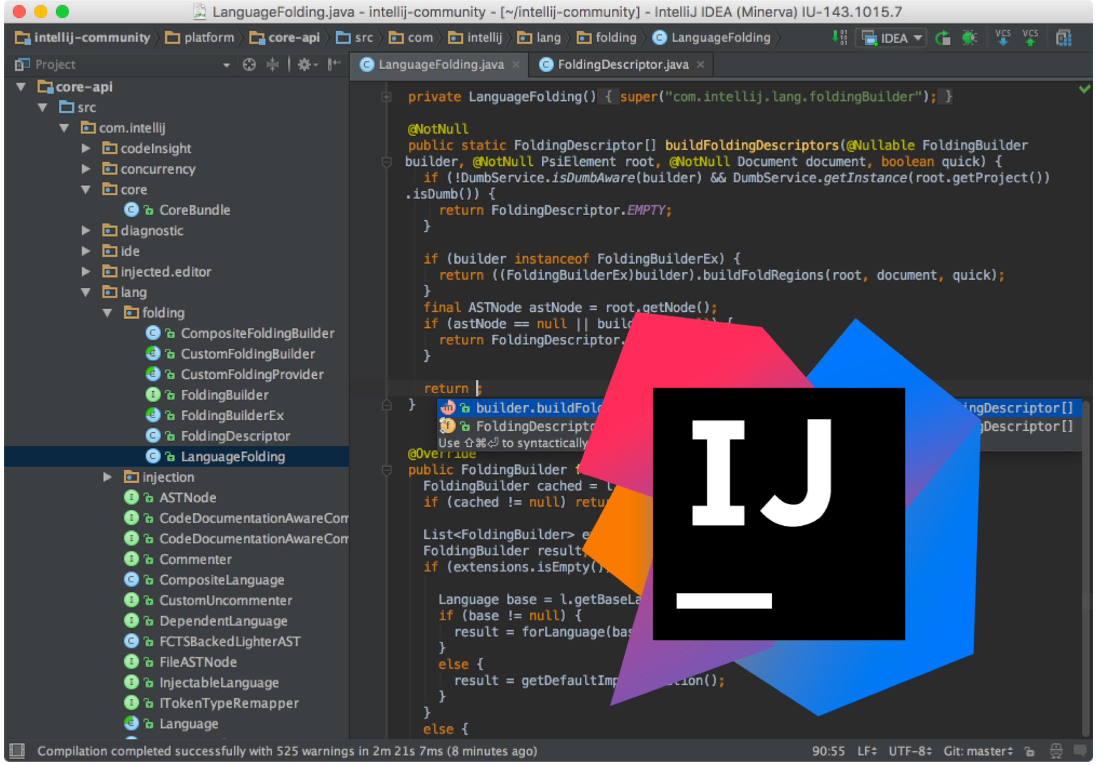
DevOps Toolchain continued

- Release — change management, release approvals, release automation
- Configure — infrastructure configuration and management, Infrastructure as Code tools
- Monitor — applications performance monitoring, end–user experience



DevOps Toolchain - Code

- **Code development** and review
- Source code management tools
- Code merging



LanguageFolding.java - intellij-community - [~/intellij-community] - IntelliJ IDEA (Minerva) IU-143.1015.7

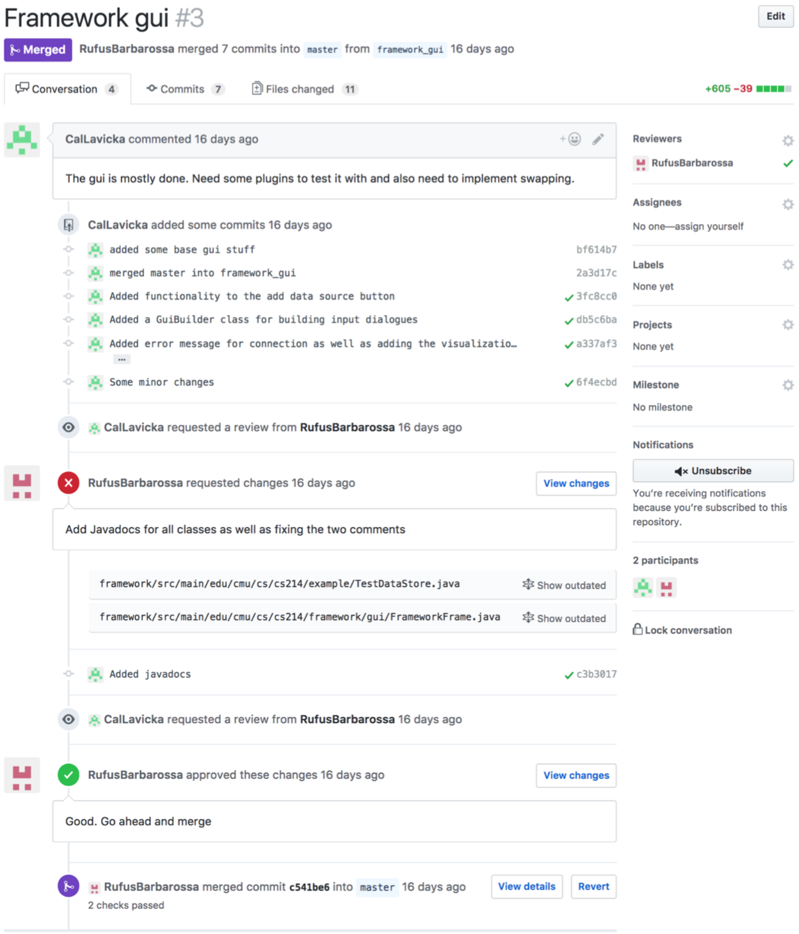
```
  private LanguageFolding() { super("com.intellij.lang.foldingBuilder"); }

  @NotNull
  public static FoldingDescriptor[] buildFoldingDescriptors(@Nullable FoldingBuilder
builder, @NotNull PsiElement root, @NotNull Document document, boolean quick) {
    if (!DumbService.isDumbAware(builder) && DumbService.getInstance(root.getProject())
.isDumb()) {
      return FoldingDescriptor.EMPTY;
    }

    if (builder instanceof FoldingBuilderEx) {
      return ((FoldingBuilderEx)builder).buildFoldRegions(root, document, quick);
    }
    final ASTNode astNode = root.getNode();
    if (astNode == null || buil
      return FoldingDescriptor.
    }

    return ;
  }    builder.buildFol                                          ingDescriptor[]
       FoldingDescripto                                          Descriptor[]
       Use ⇧⌘⏎ to syntactically
  @Override
  public FoldingBuilder f
    FoldingBuilder cached = l
    if (cached != null) retu

    List<FoldingBuilder> e
    FoldingBuilder result
    if (extensions.isEmpty()

      Language base = l.getBaseLa
      if (base != null) {
        result = forLanguage(bas
      }
      else {
        result = getDefaultImp      tion();
      }
    }
    else {
```

Compilation completed successfully with 525 warnings in 2m 21s 7ms (8 minutes ago)

90:55 LF⇕ UTF-8⇕ Git: master⇕

DevOps Toolchain - Code

- Code development and **review**
- Source code management tools
- Code merging



Framework gui #3

Merged RufusBarbarossa merged 7 commits into `master` from `framework_gui` 16 days ago

CalLavicka commented 16 days ago

The gui is mostly done. Need some plugins to test it with and also need to implement swapping.

CalLavicka added some commits 16 days ago

```
added some base gui stuff                                                  bf614b7
merged master into framework_gui                                           2a3d17c
Added functionality to the add data source button                        ✓ 3fc8cc0
Added a GuiBuilder class for building input dialogues                    ✓ db5c6ba
Added error message for connection as well as adding the visualizatio…   ✓ a337af3
Some minor changes                                                       ✓ 6f4ecbd
```

CalLavicka requested a review from **RufusBarbarossa** 16 days ago

RufusBarbarossa requested changes 16 days ago

Add Javadocs for all classes as well as fixing the two comments

```
framework/src/main/edu/cmu/cs/cs214/example/TestDataStore.java
framework/src/main/edu/cmu/cs/cs214/framework/gui/FrameworkFrame.java
```

```
Added javadocs                                                           ✓ c3b3017
```

CalLavicka requested a review from **RufusBarbarossa** 16 days ago

RufusBarbarossa approved these changes 16 days ago

Good. Go ahead and merge

RufusBarbarossa merged commit **c541be6** into `master` 16 days ago

2 checks passed

Reviewers: RufusBarbarossa ✓

Assignees: No one—assign yourself

Labels: None yet

Projects: None yet

Milestone: No milestone

Notifications

You're receiving notifications because you're subscribed to this repository.

2 participants

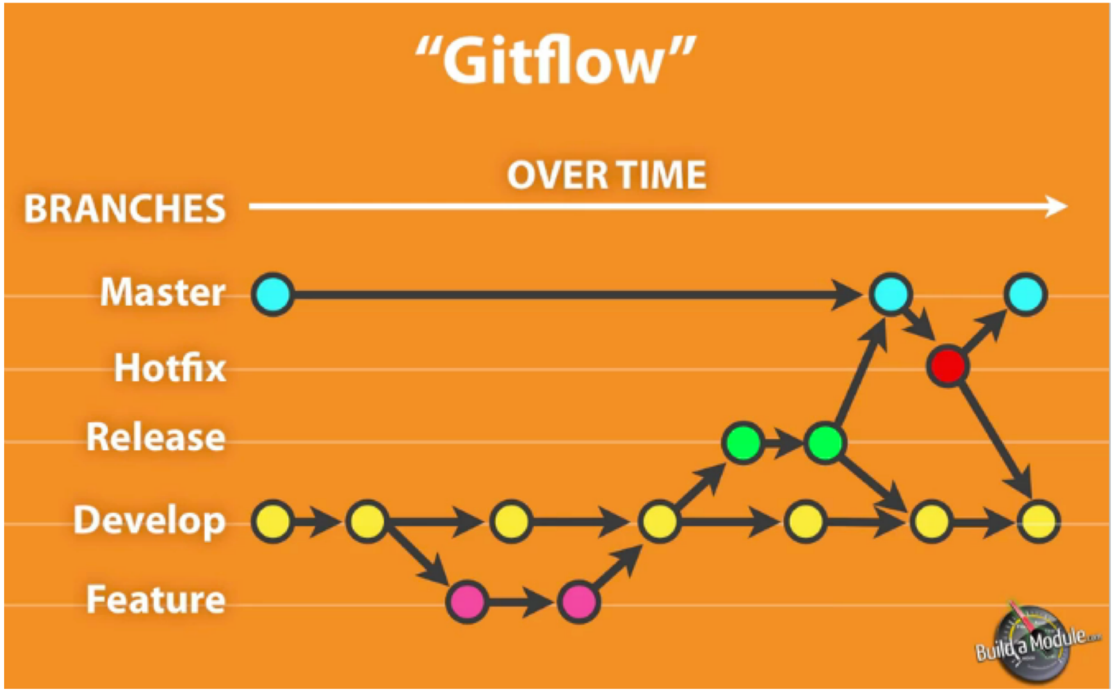
More on Code Review in 17-313

DevOps Toolchain - Code

- Code development and review
- **Source code management tools**
- Code merging

DevOps Toolchain - Code

- Code development and review
- Source code management tools
- **Code merging**



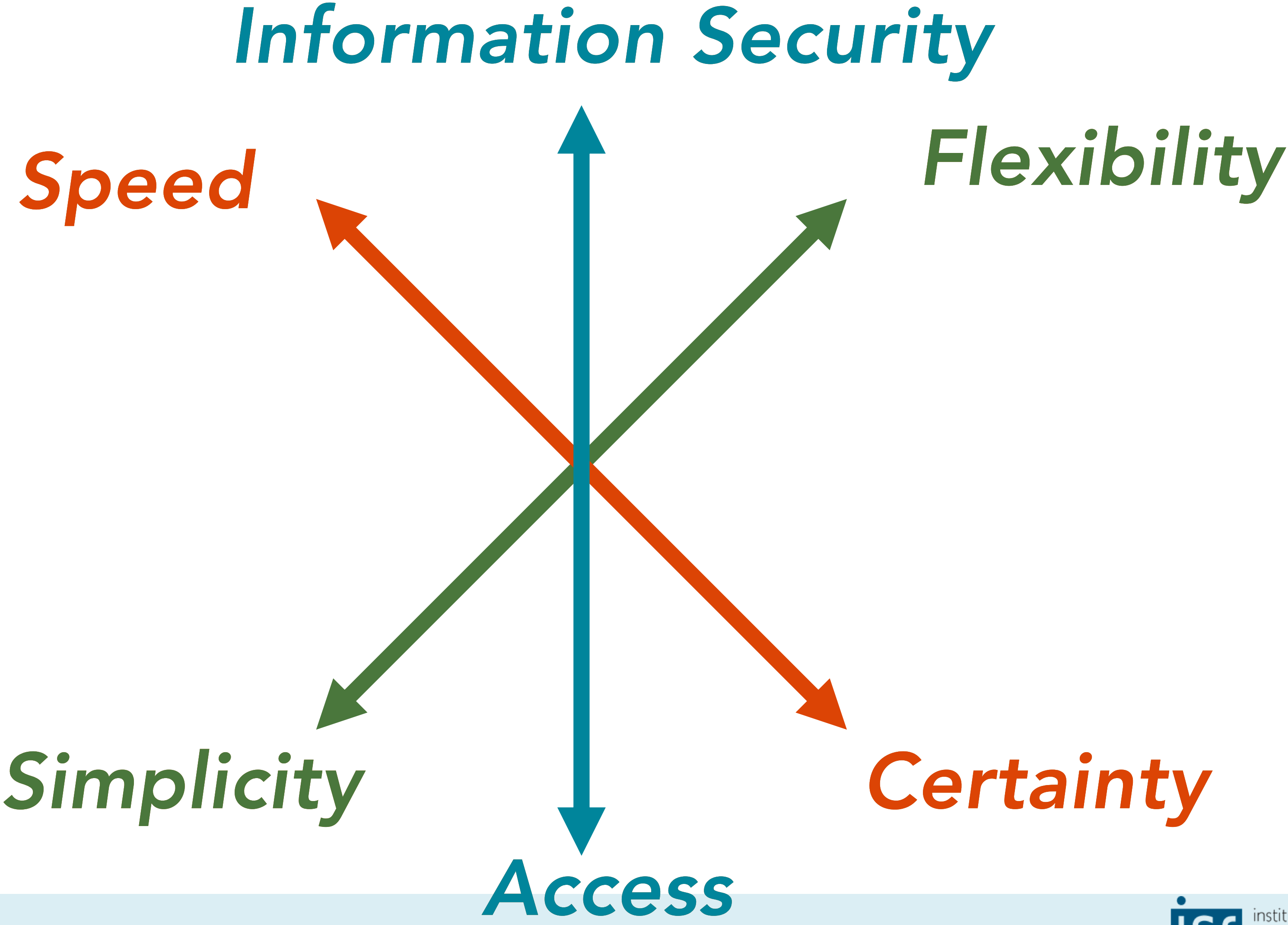
DevOps Toolchain - Build

- **Continuous integration tools**
- Build status

Example

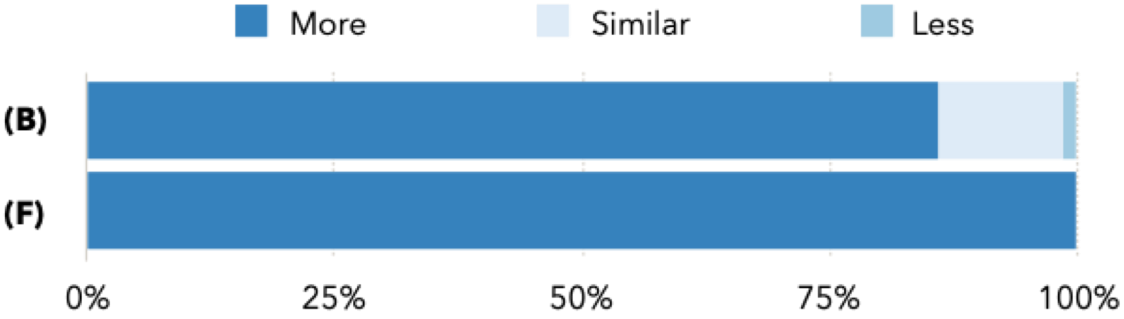
- Create Pull Request
- GitHub tells Travis CI build is mergeable
- It builds and passes tests
- All is well Travis updates PR
- PR is merged

Findings


Information Security
Speed
Flexibility
Simplicity
Certainty
Access

Experiences

Do developers on projects with CI give (**more**/similar/less) value to automated tests?

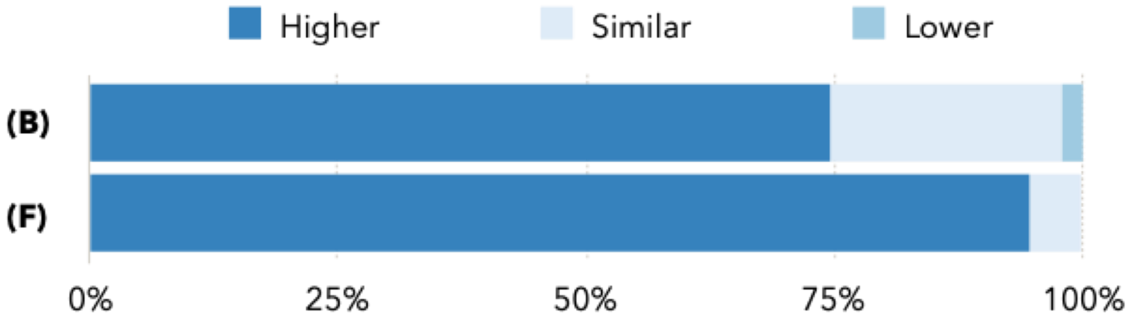


(B)road (F)ocused

Experiences

Do developers on projects with CI give (**more**/similar/less) value to automated tests?

Do projects with CI have (**higher**/similar/lower) test quality?



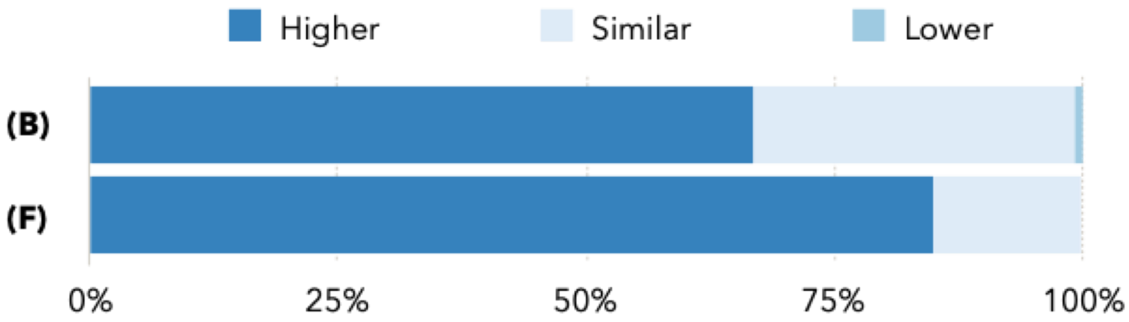
(B)road (F)ocused

Experiences

Do developers on projects with CI give (**more**/similar/less) value to automated tests?

Do projects with CI have (**higher**/similar/lower) test quality?

Do projects with CI have (**higher**/similar/lower) code quality?



(B)road (F)ocused

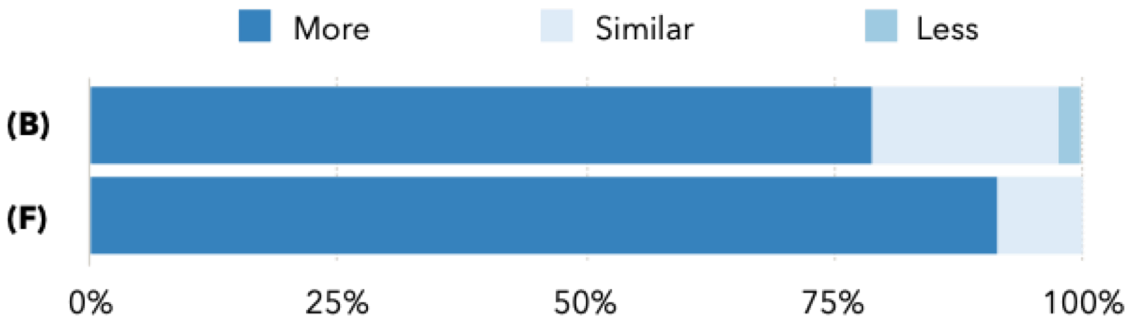
Experiences

Do developers on projects with CI give (**more**/similar/less) value to automated tests?

Do projects with CI have (**higher**/similar/lower) test quality?

Do projects with CI have (**higher**/similar/lower) code quality?

Are developers on projects with CI (**more**/similar/less) productive?



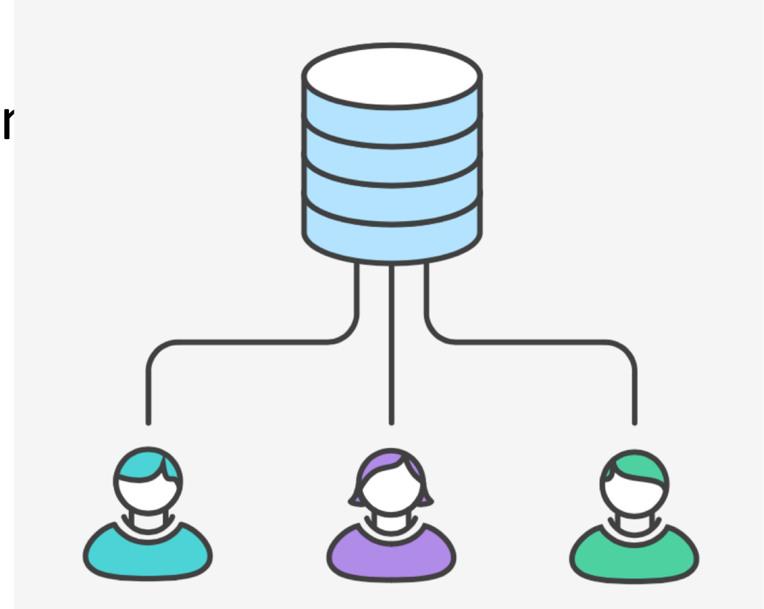
(B)road (F)ocused

BRANCH WORKFLOWS

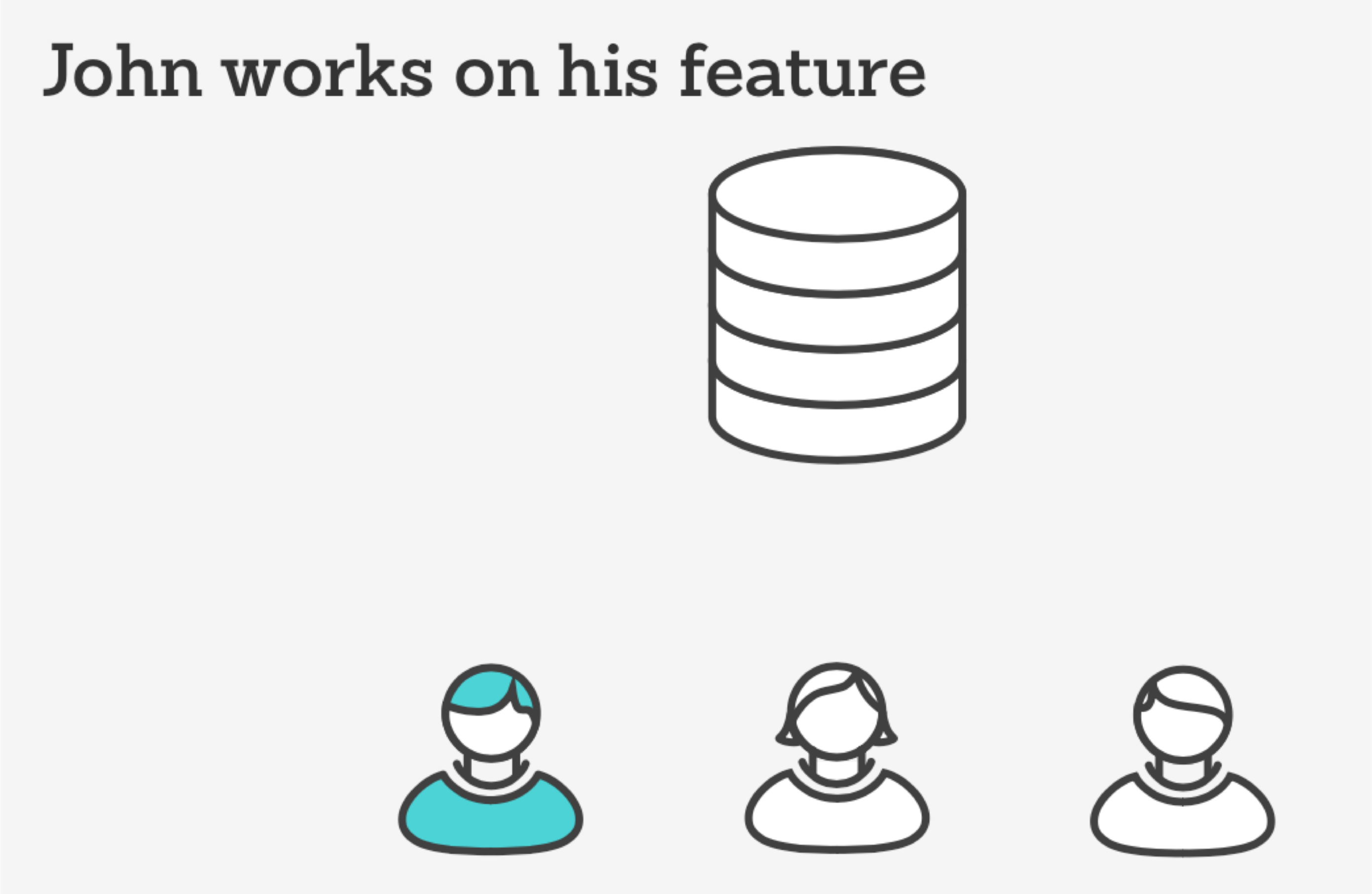
https://www.atlassian.com/git/tutorials/comparing-workflows

1. Centralized workflow

- Central repository to serve as the single point-of-entry for all changes to the project
- Default development branch is called r
  - all changes are committed into master
  - doesn’t require any other branches

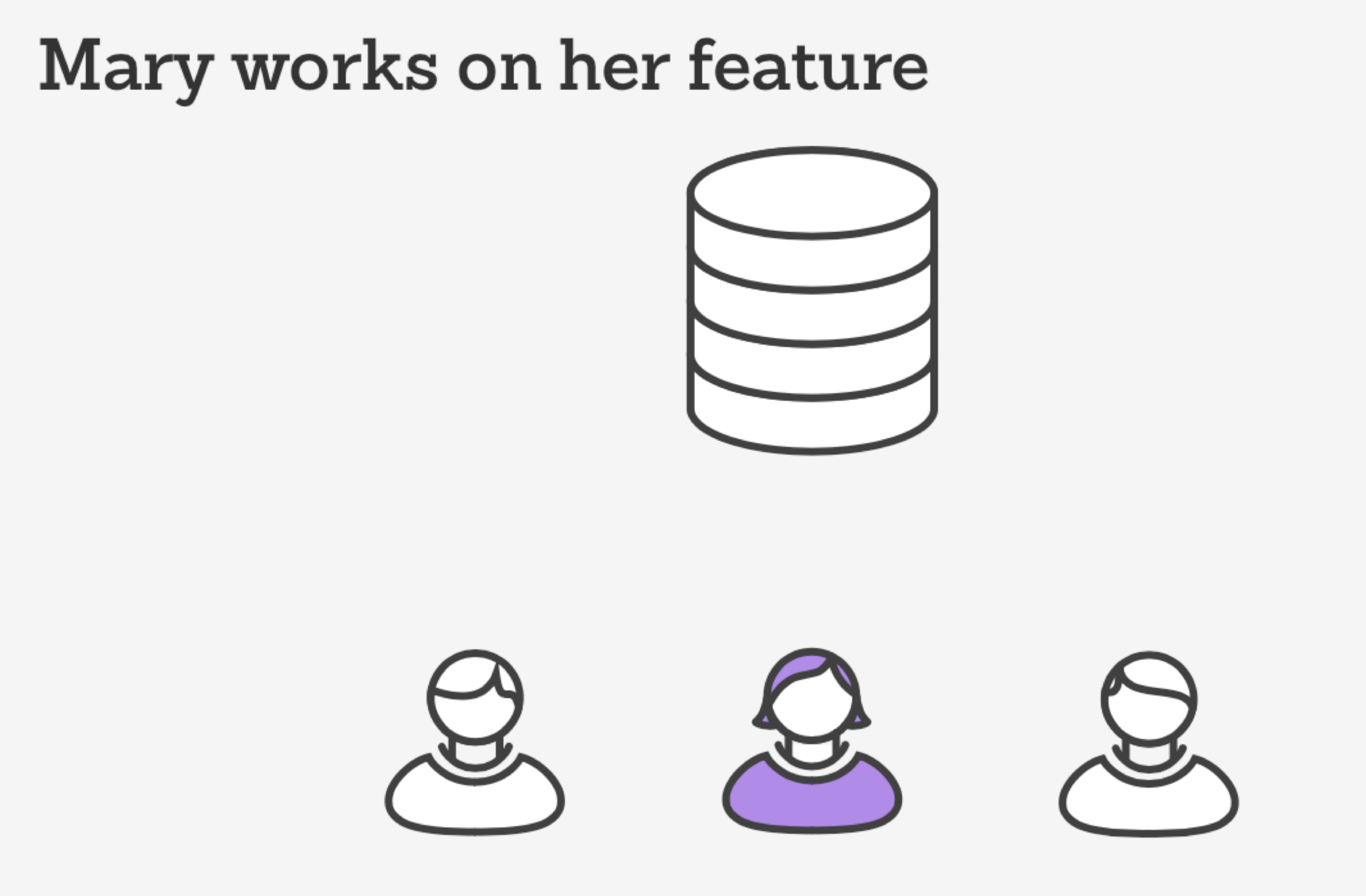


Example



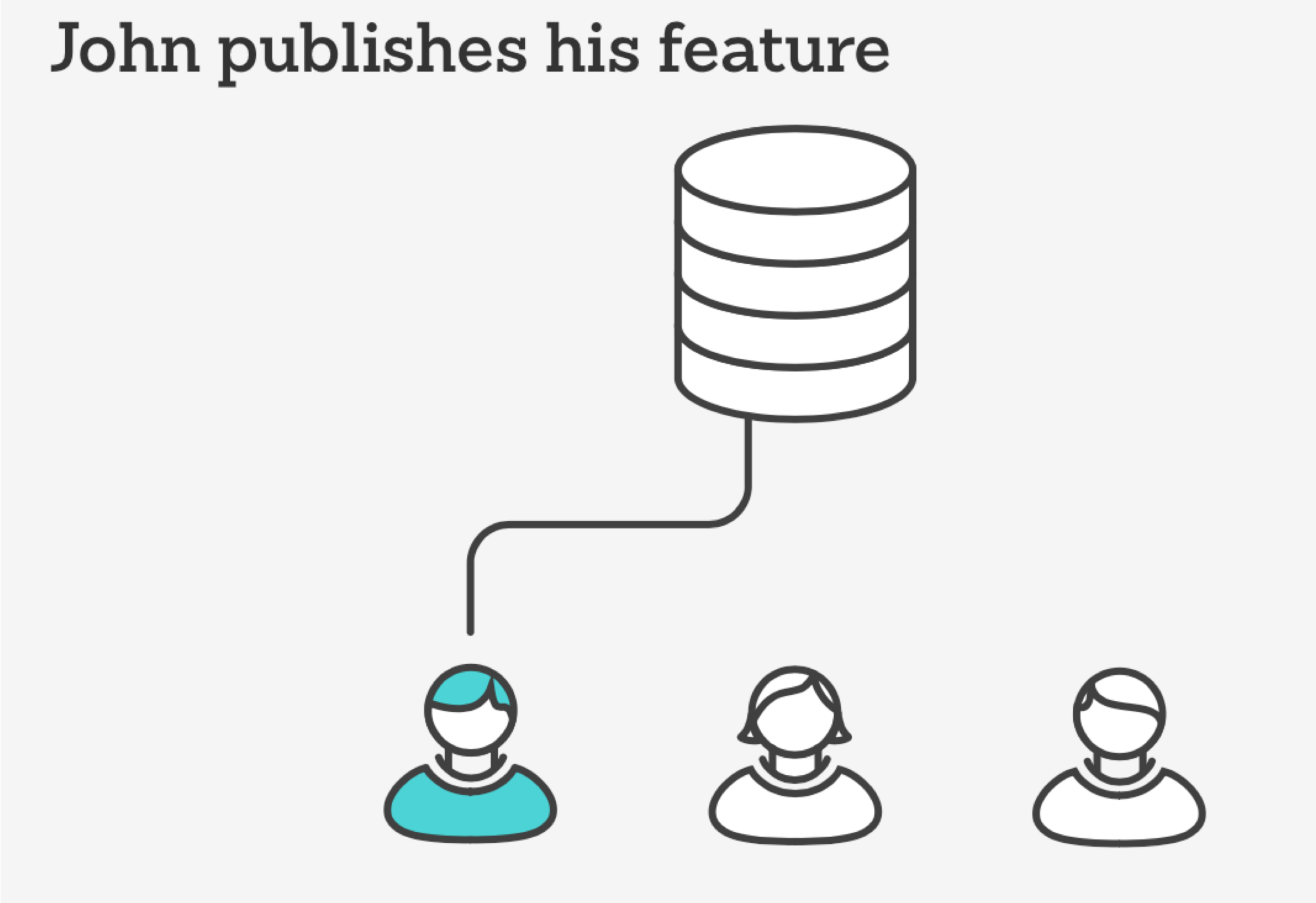
John works on his feature

Example



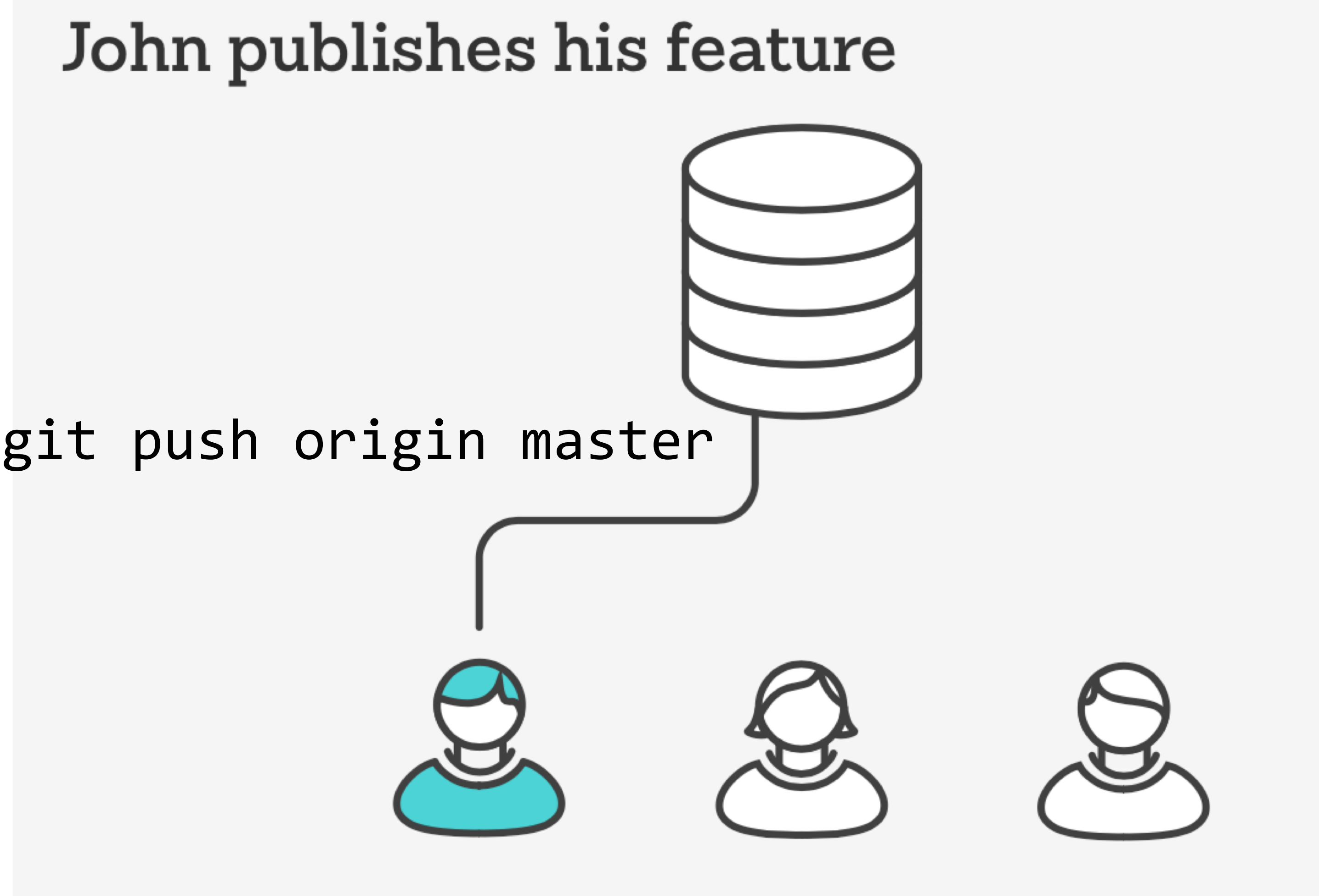
Mary works on her feature

Example



John publishes his feature

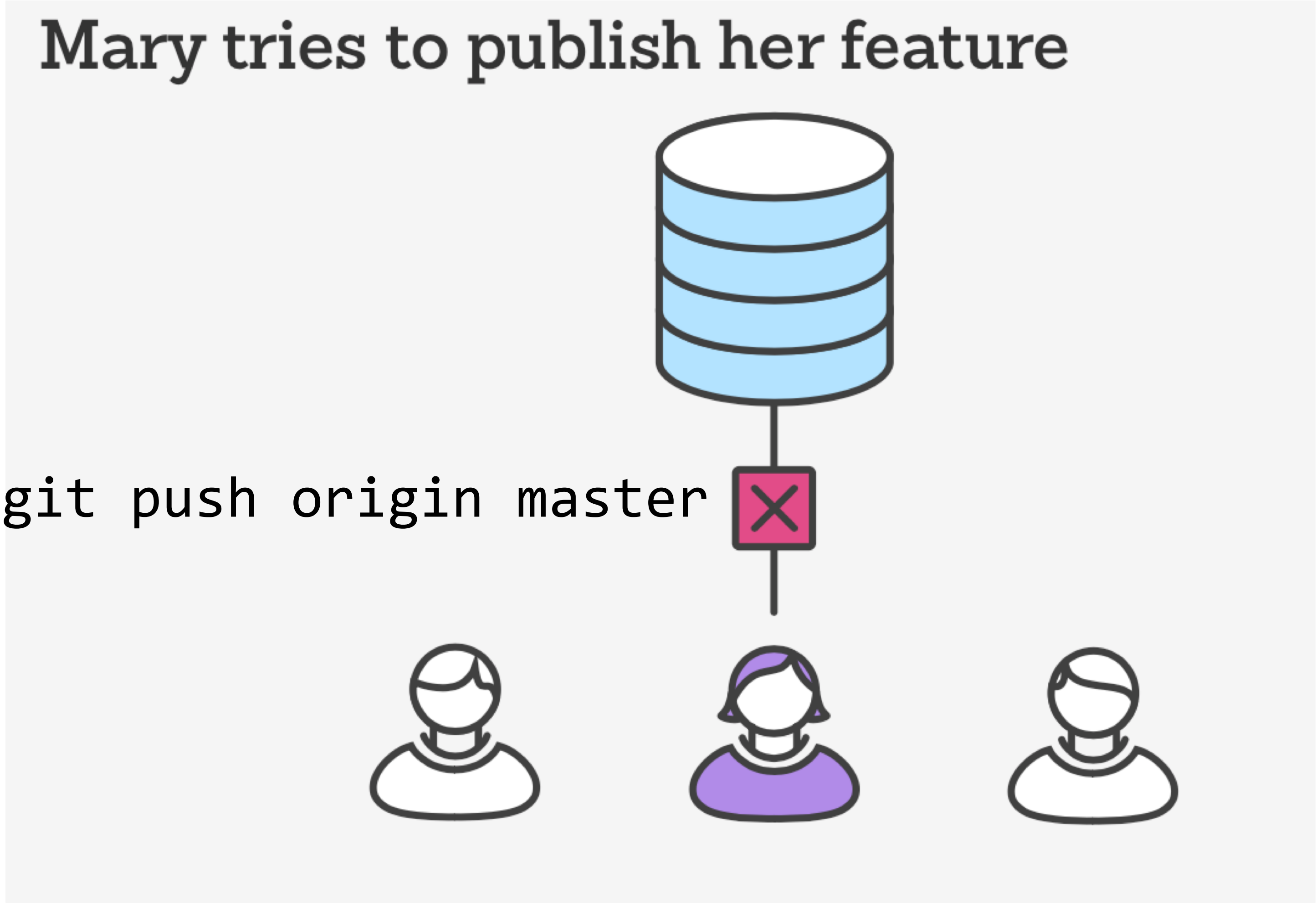
Example



John publishes his feature

```
git push origin master
```

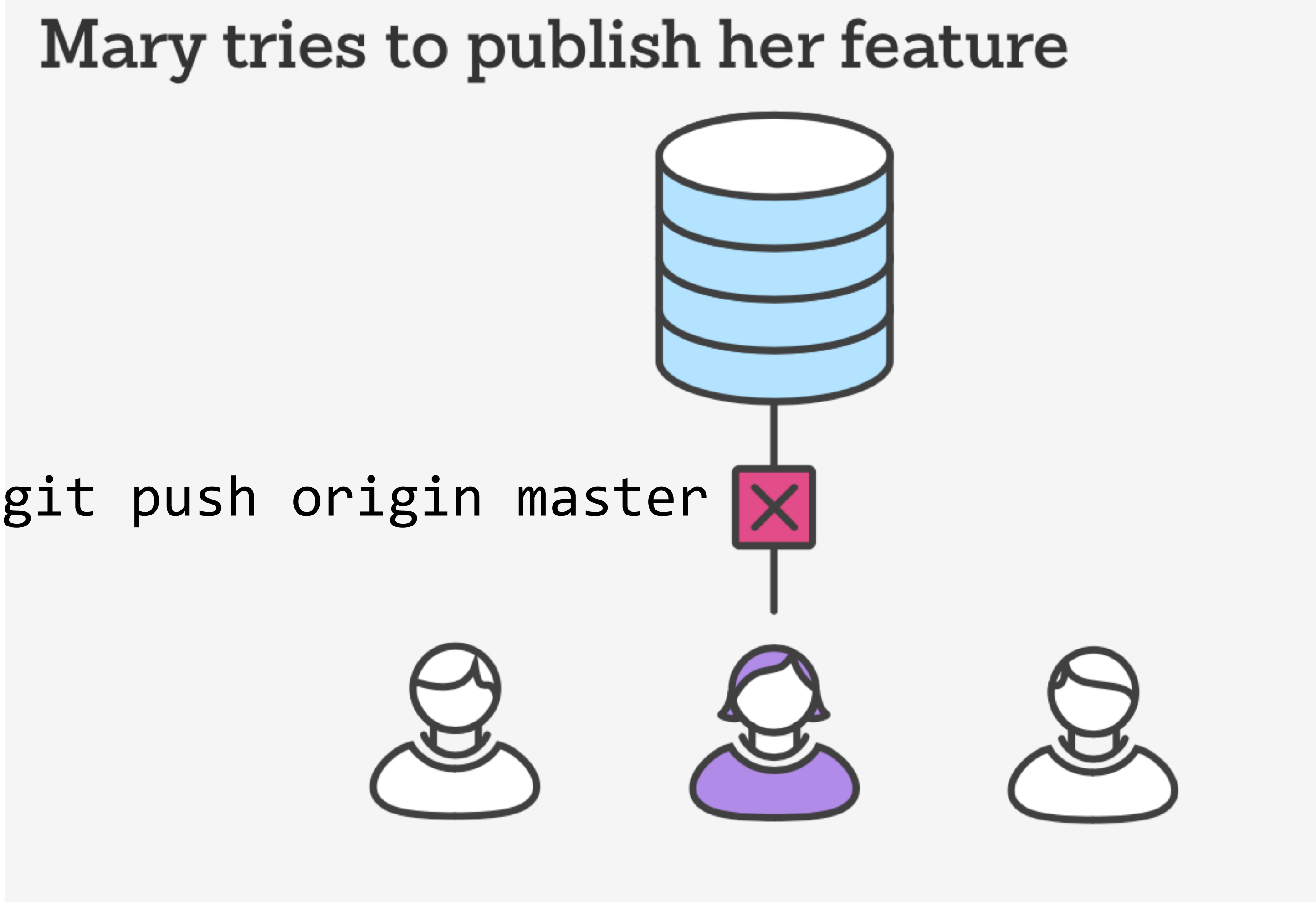
Example



Mary tries to publish her feature

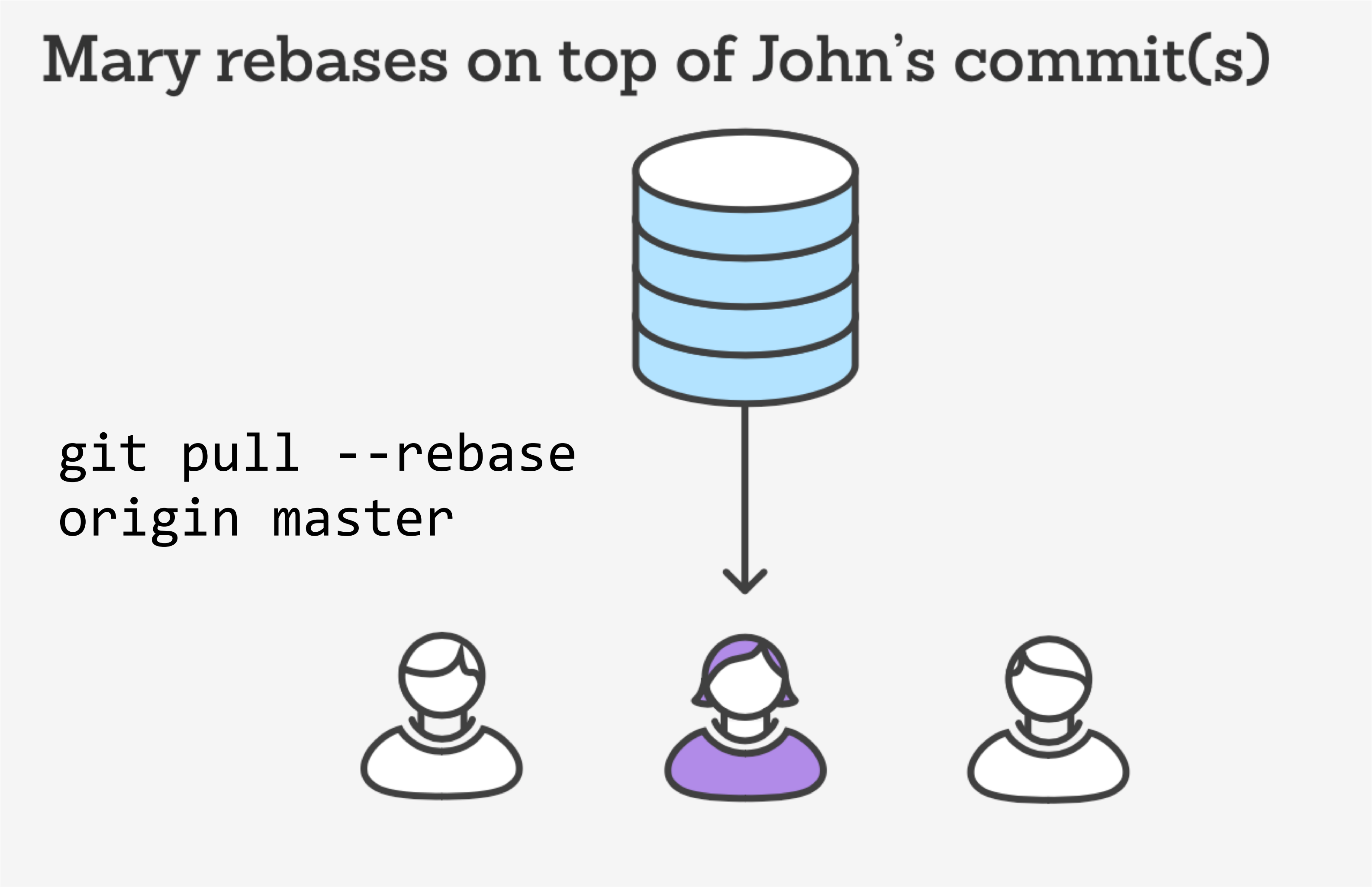
```
git push origin master
```

```
error: failed to push some refs to '/path/to/repo.git' hint:
Updates were rejected because the tip of your current branch is
behind hint: its remote counterpart. Merge the remote changes
(e.g. 'git pull') hint: before pushing again. hint: See the
'Note about fast-forwards' in 'git push --help' for details.
```



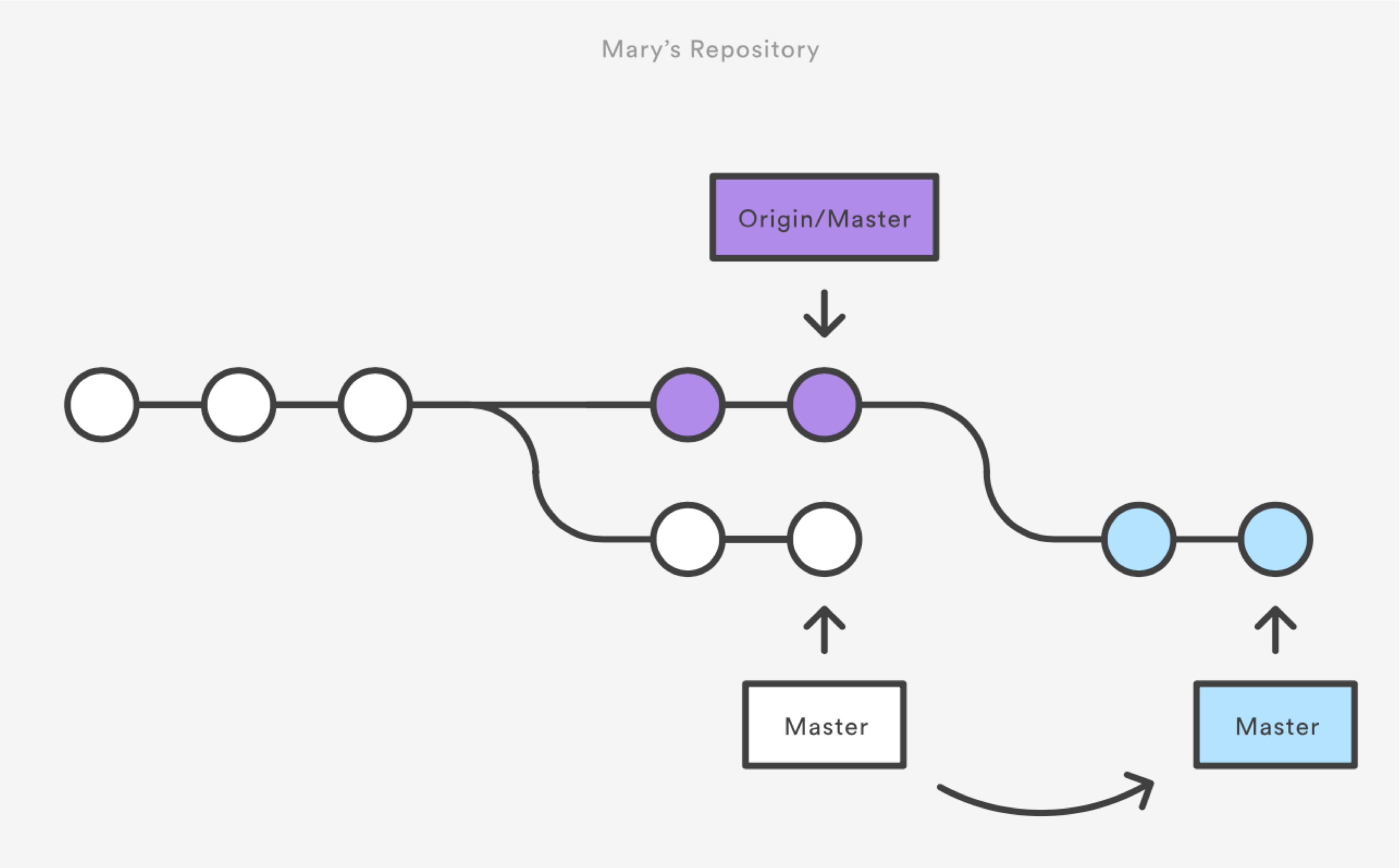
```
git push origin master
```

Example

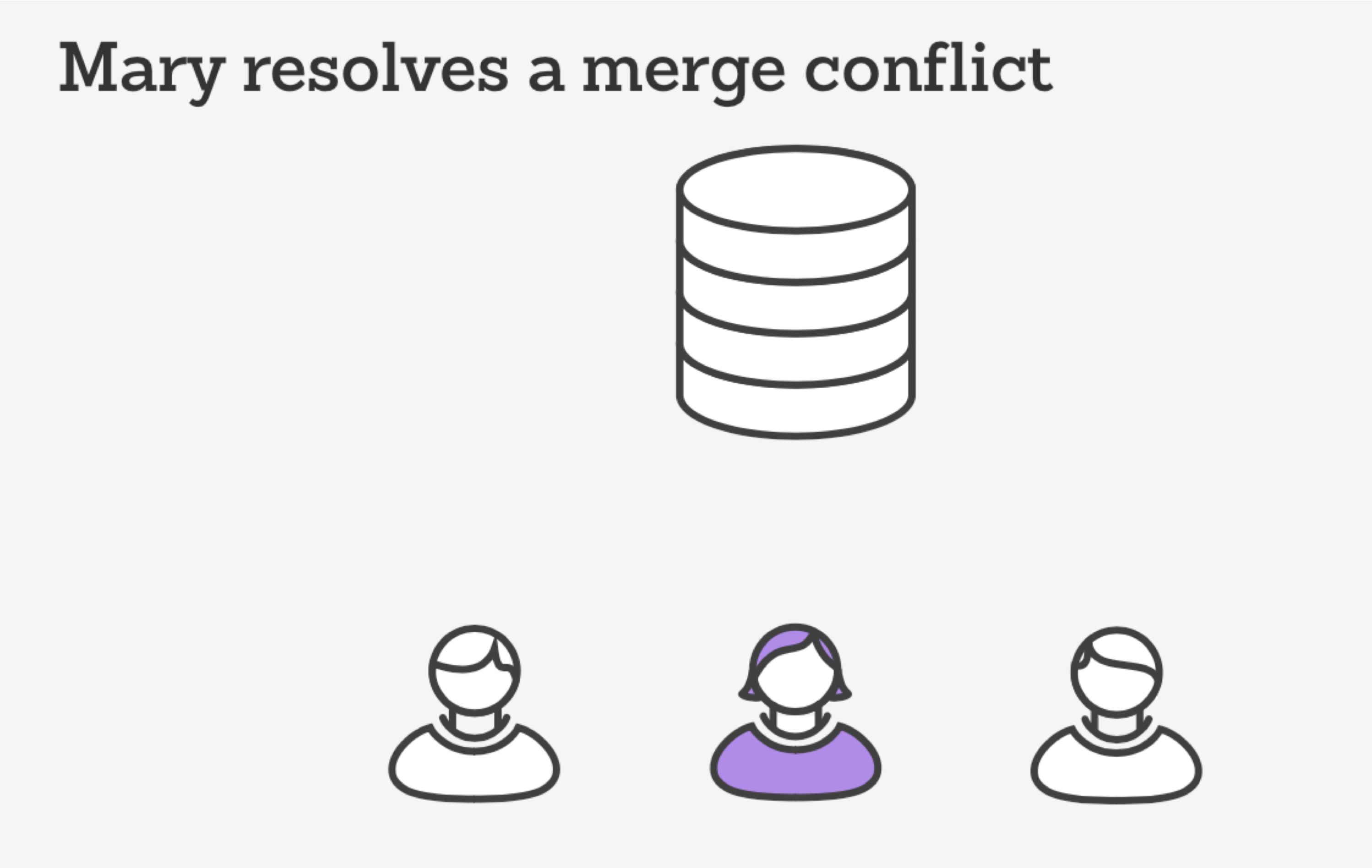


Mary rebases on top of John's commit(s)

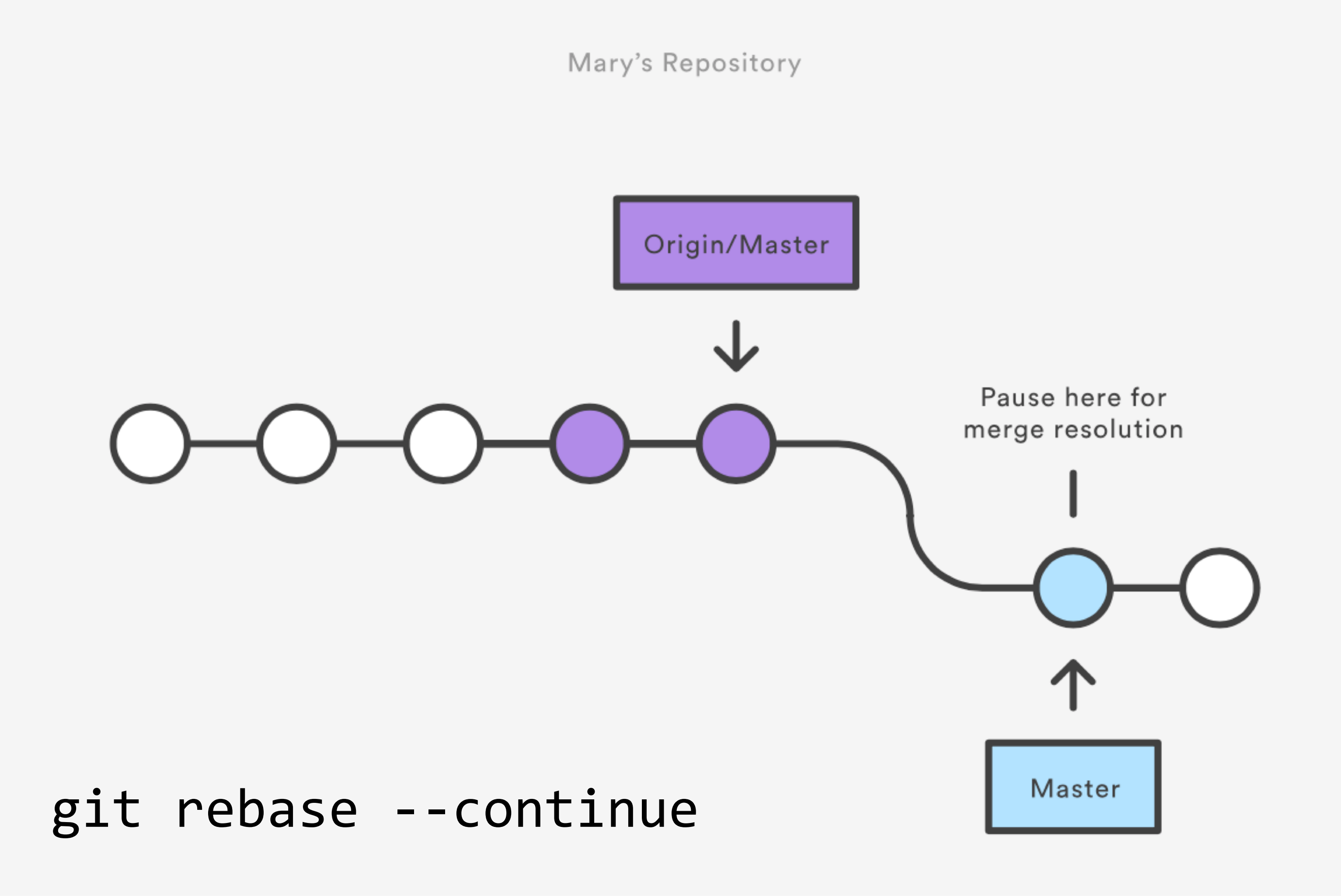
```
git pull --rebase
origin master
```


Mary's Repository
Origin/Master
Master
Master

Example

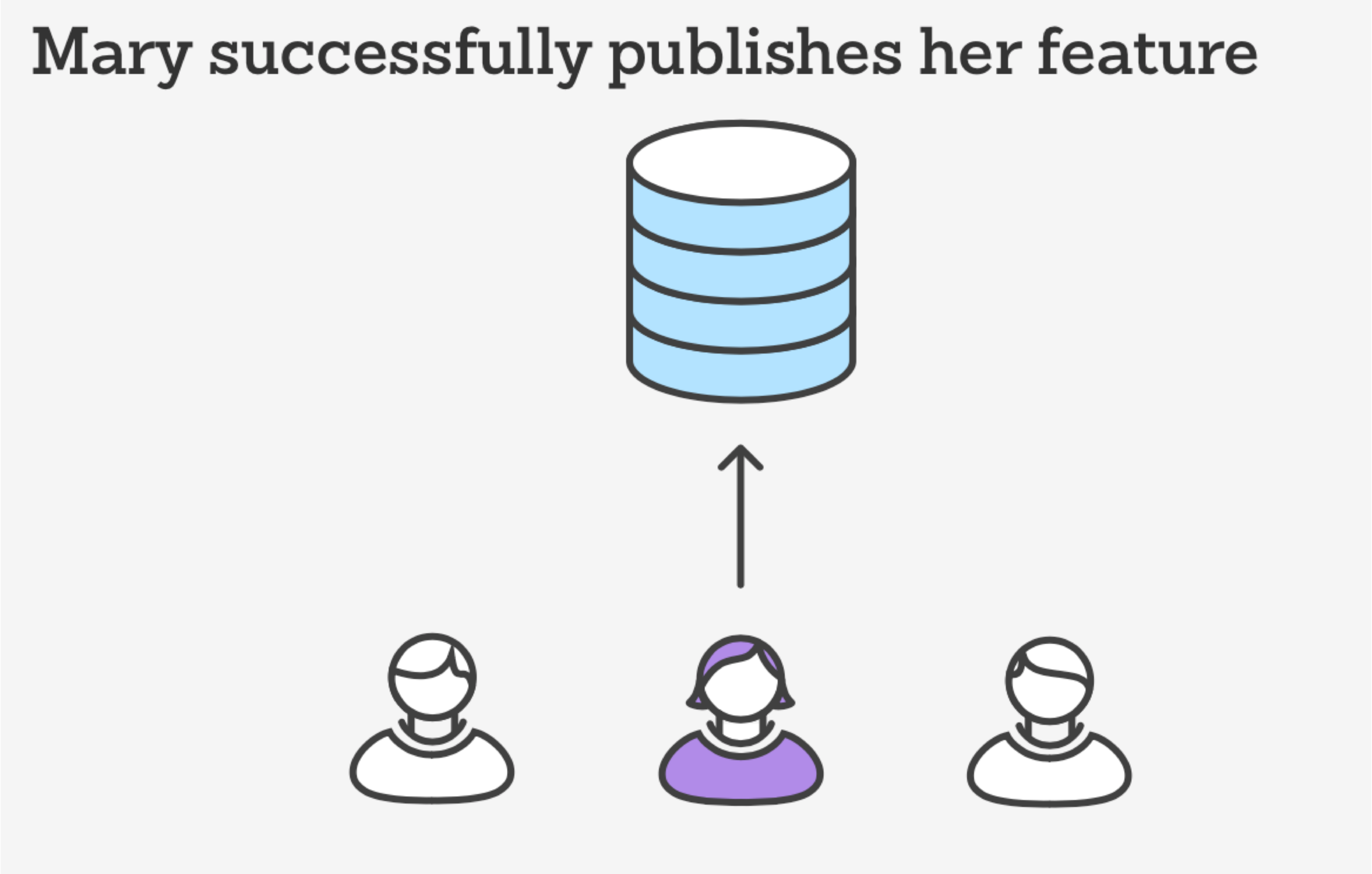


Example



```
git rebase --continue
```

Example

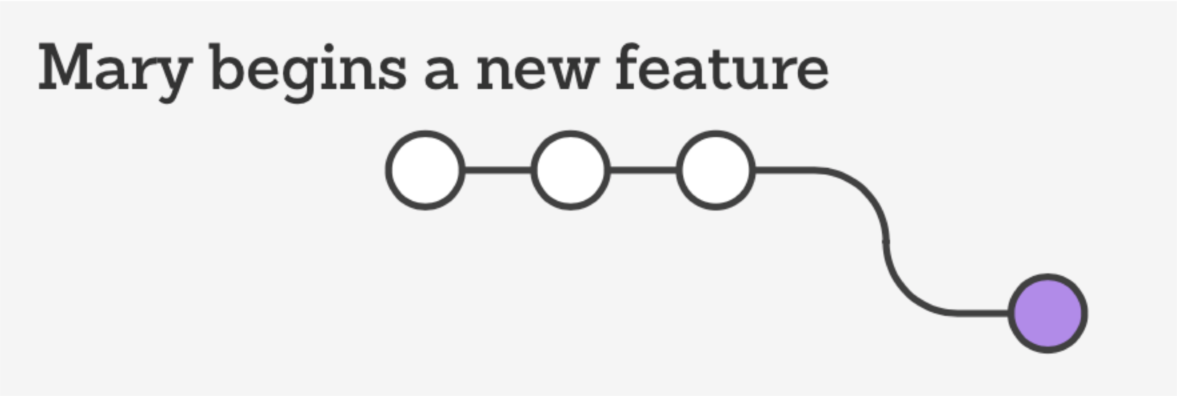


Mary successfully publishes her feature

2. Git Feature Branch Workflow

- *All* feature development should take place in a dedicated branch instead of the master branch
- Multiple developers can work on a particular feature without disturbing the main codebase
  - master branch will never contain broken code (enables CI)
  - Enables pull requests (code review)

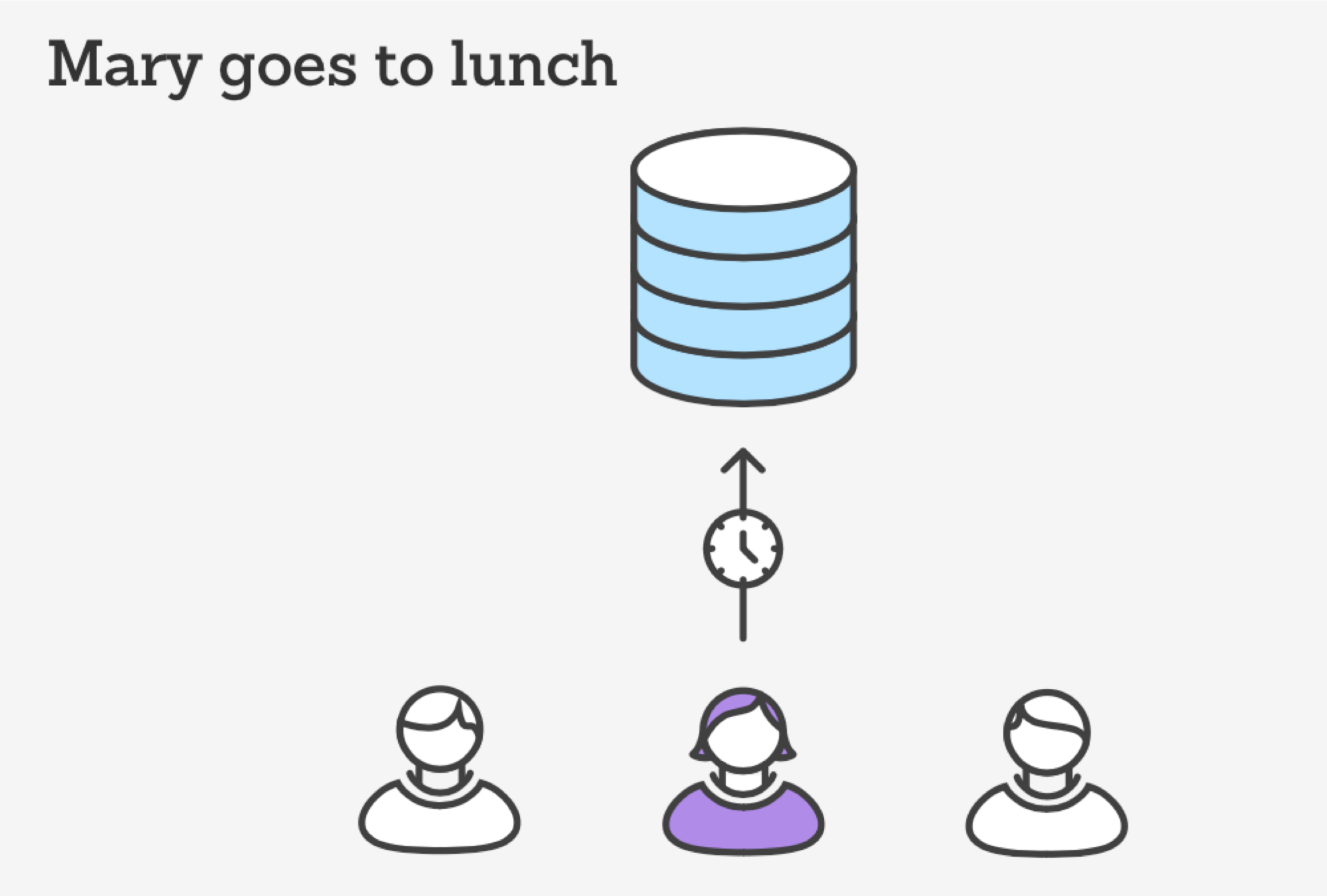
Example



Mary begins a new feature

```
git checkout -b marys-feature master

git status
git add <some-file>
git commit
```

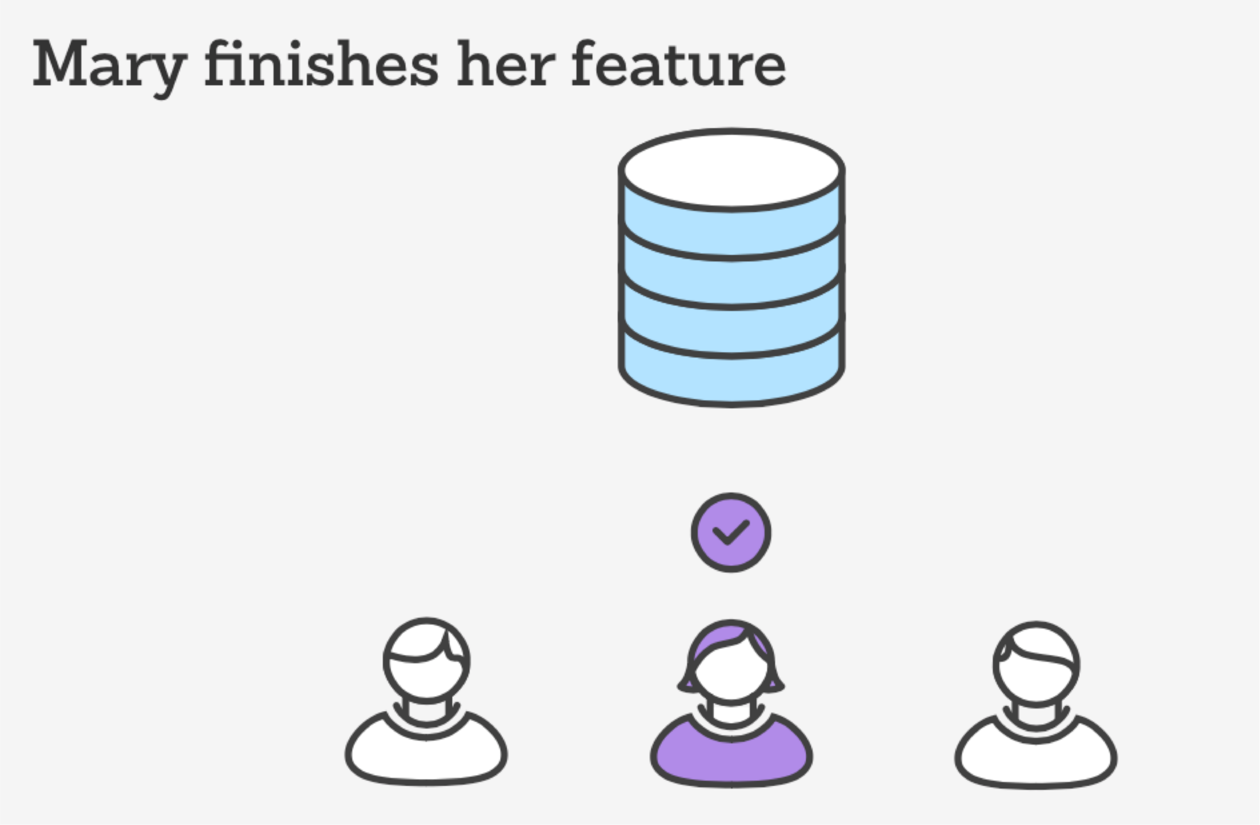
Example



Mary goes to lunch

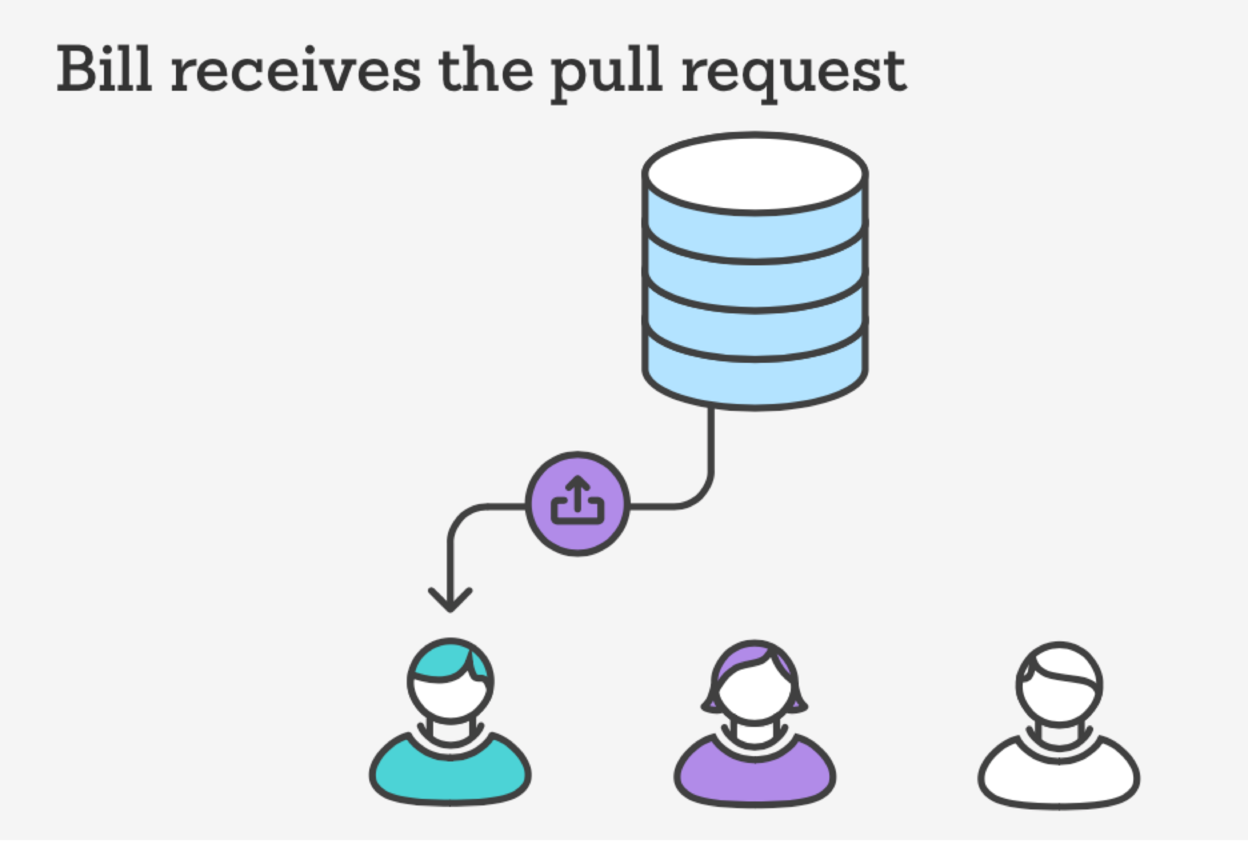
```
git push -u origin marys-feature
```

Example

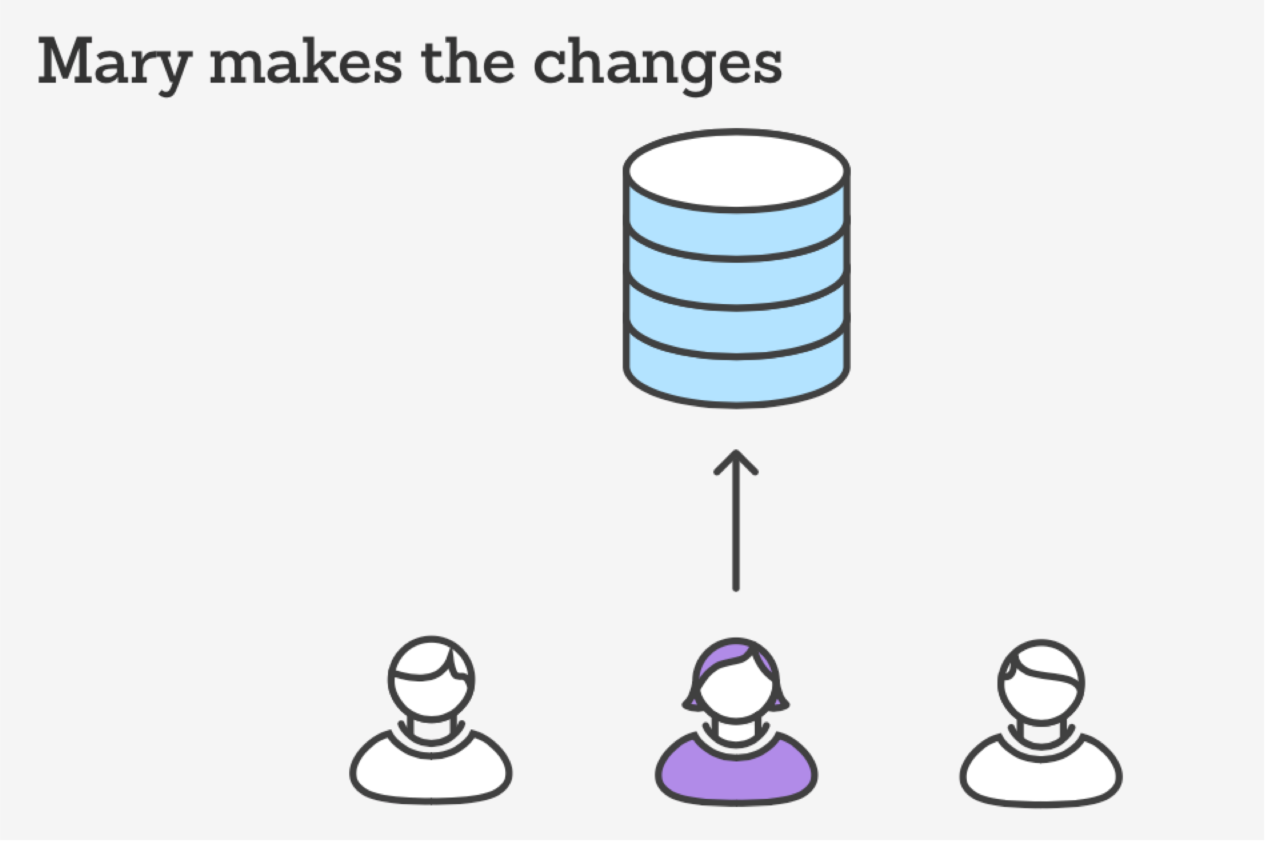


```
git push
```

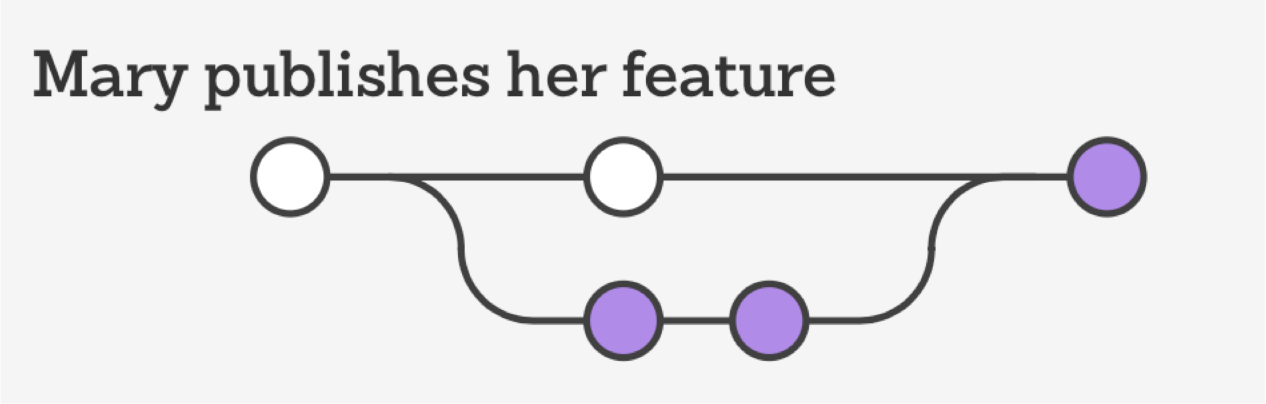
Example



Example



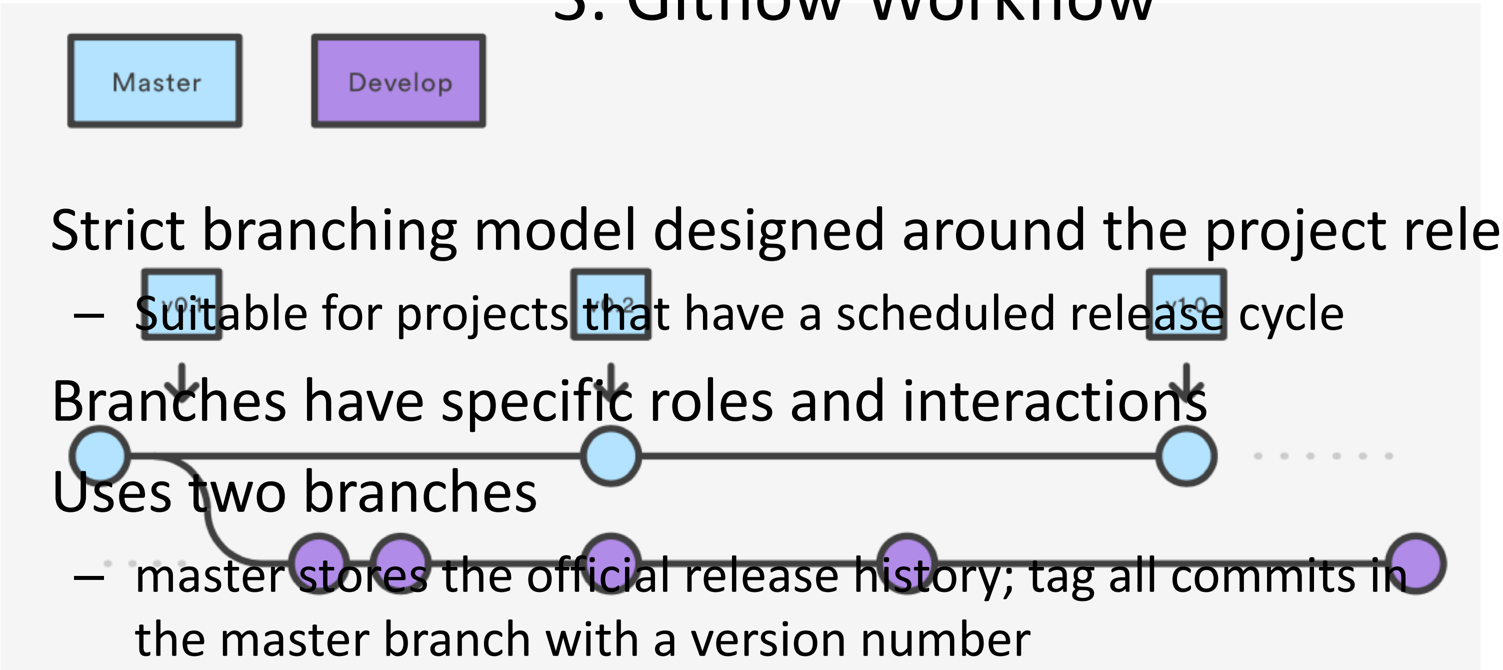
Example - Merge pull request



Mary publishes her feature

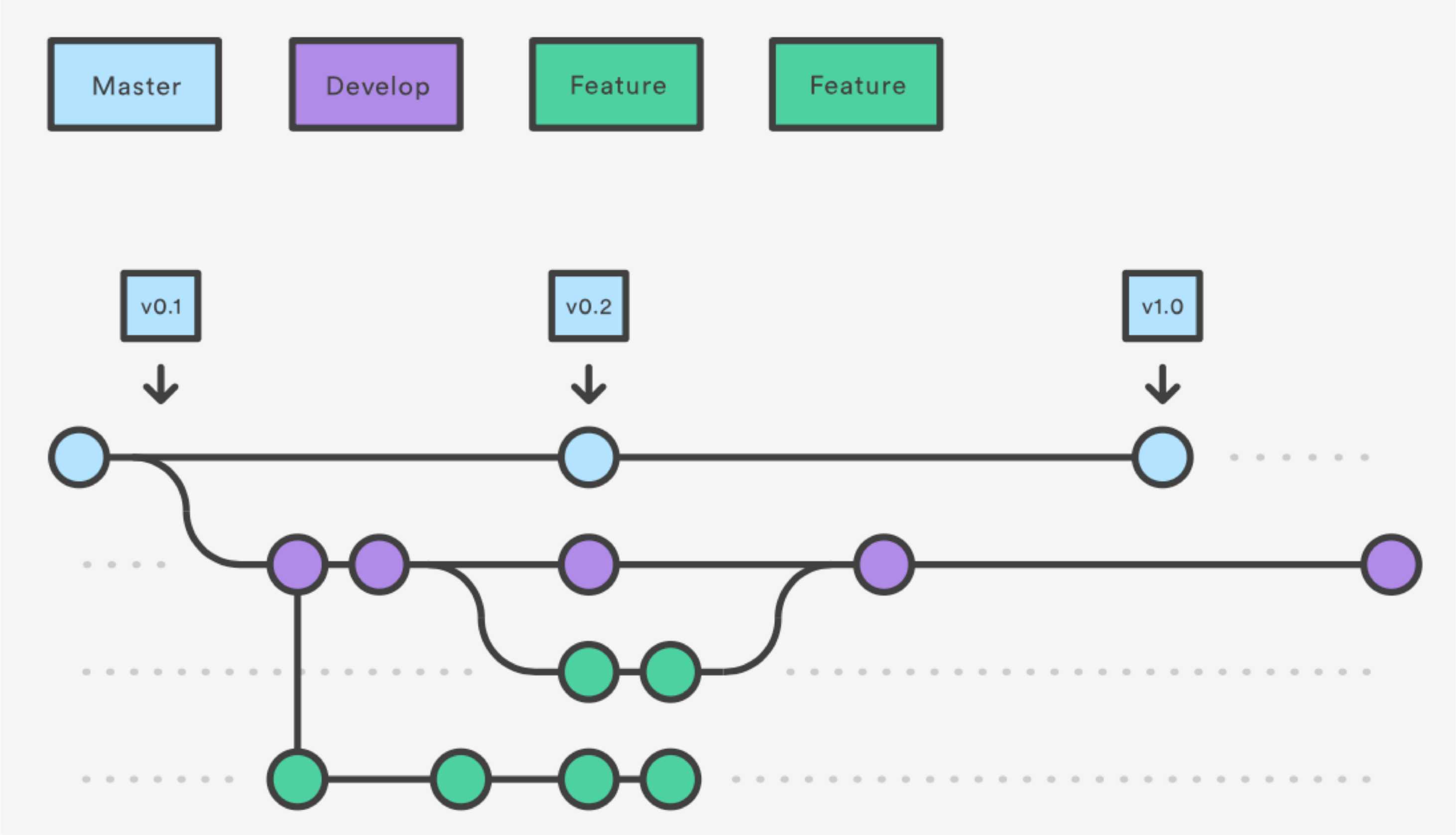
```
git checkout master
git pull
git pull origin marys-feature
git push
```

3. Gitflow Workflow

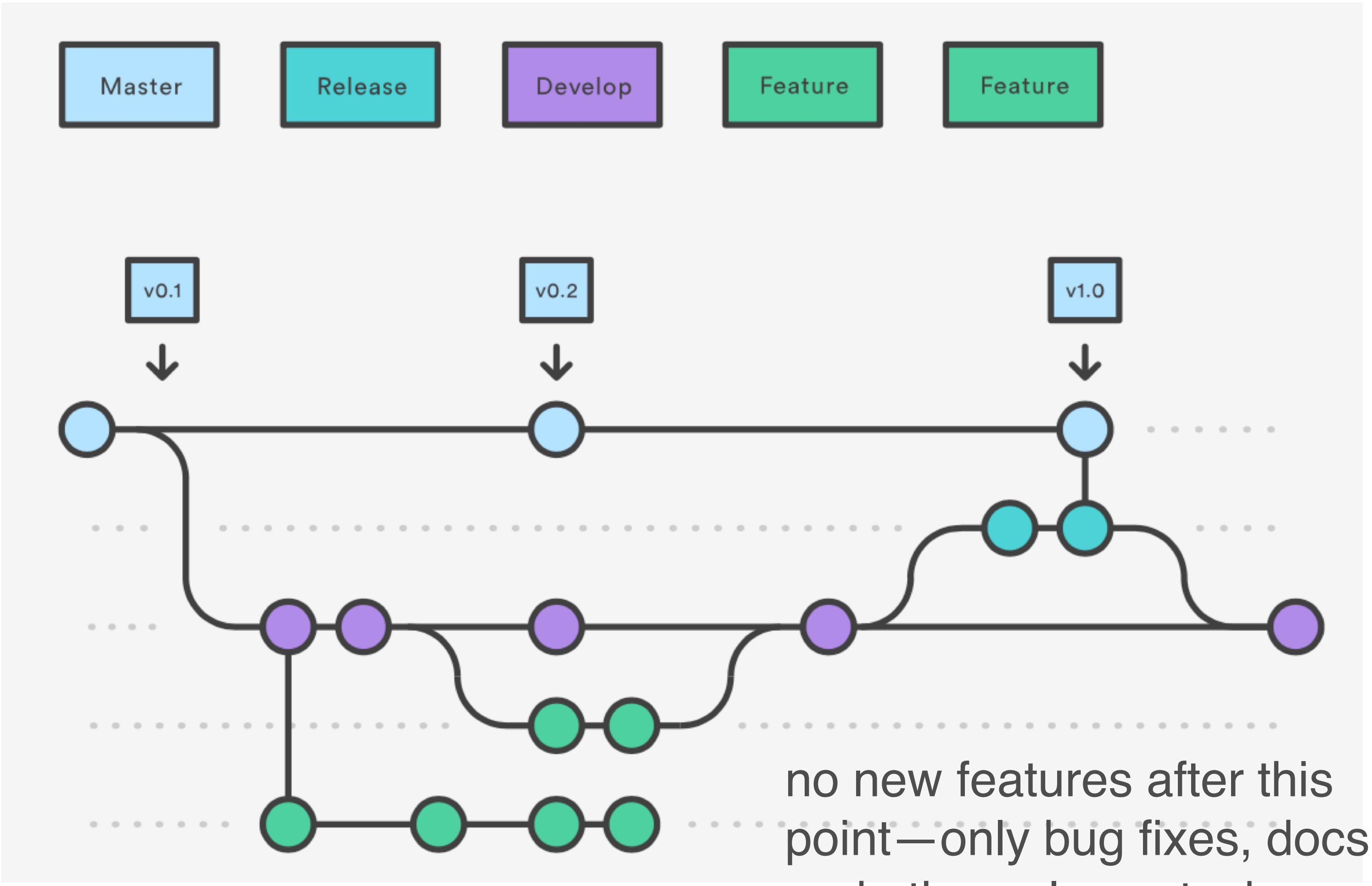


- Strict branching model designed around the project release
  - Suitable for projects that have a scheduled release cycle
- Branches have specific roles and interactions
- Uses two branches
  - master stores the official release history; tag all commits in the master branch with a version number
  - develop serves as an integration branch for features

GitFlow feature branches (from develop)



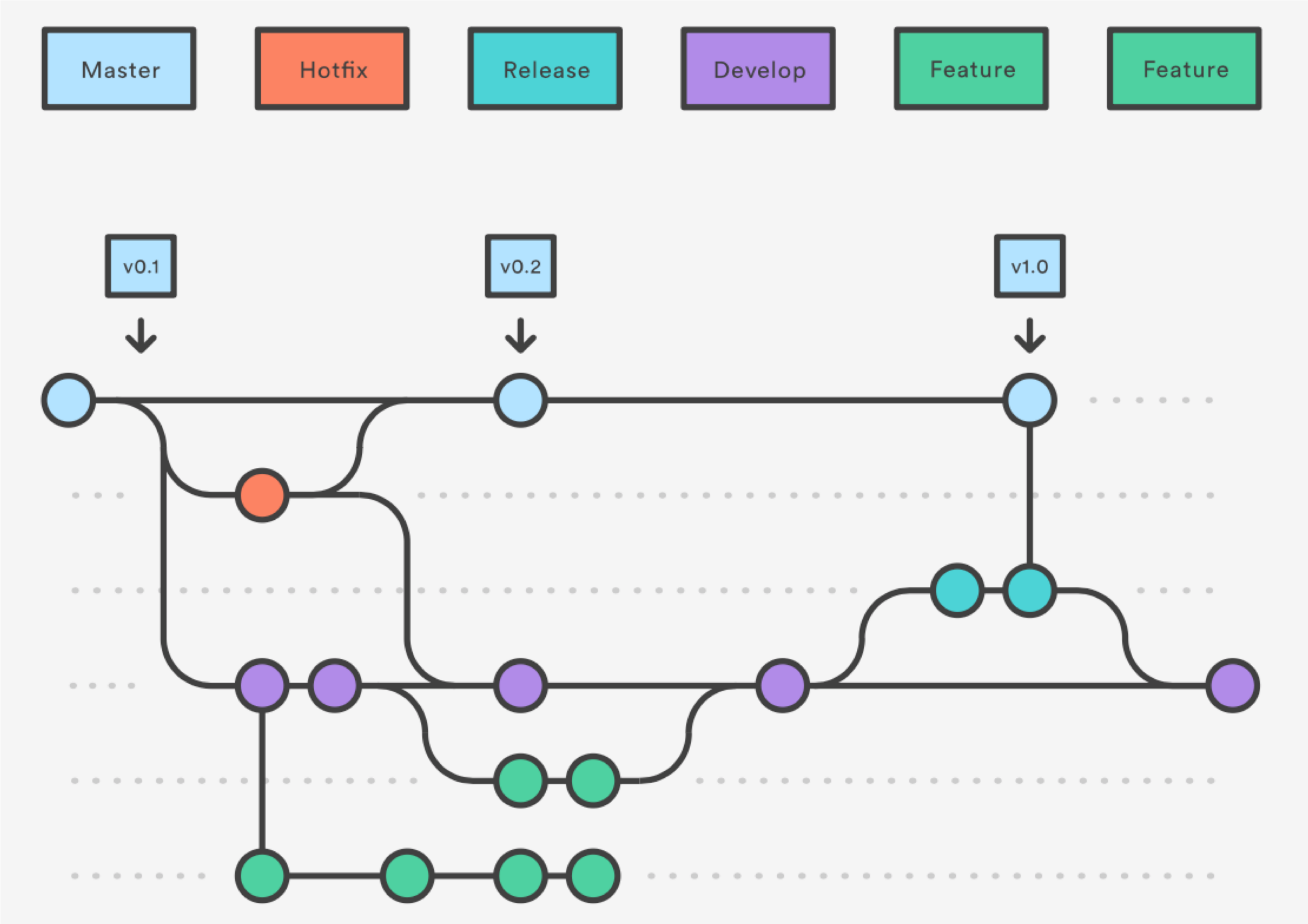
GitFlow release branches (eventually into master)



no new features after this point—only bug fixes, docs, and other release tasks

GitFlow hotfix branches

used to quickly patch production releases



Summary

- Version control has many advantages
  - History, traceability, versioning
  - Collaborative and parallel development
- Collaboration with branches
  - Different workflows
- From local to central to distributed version control

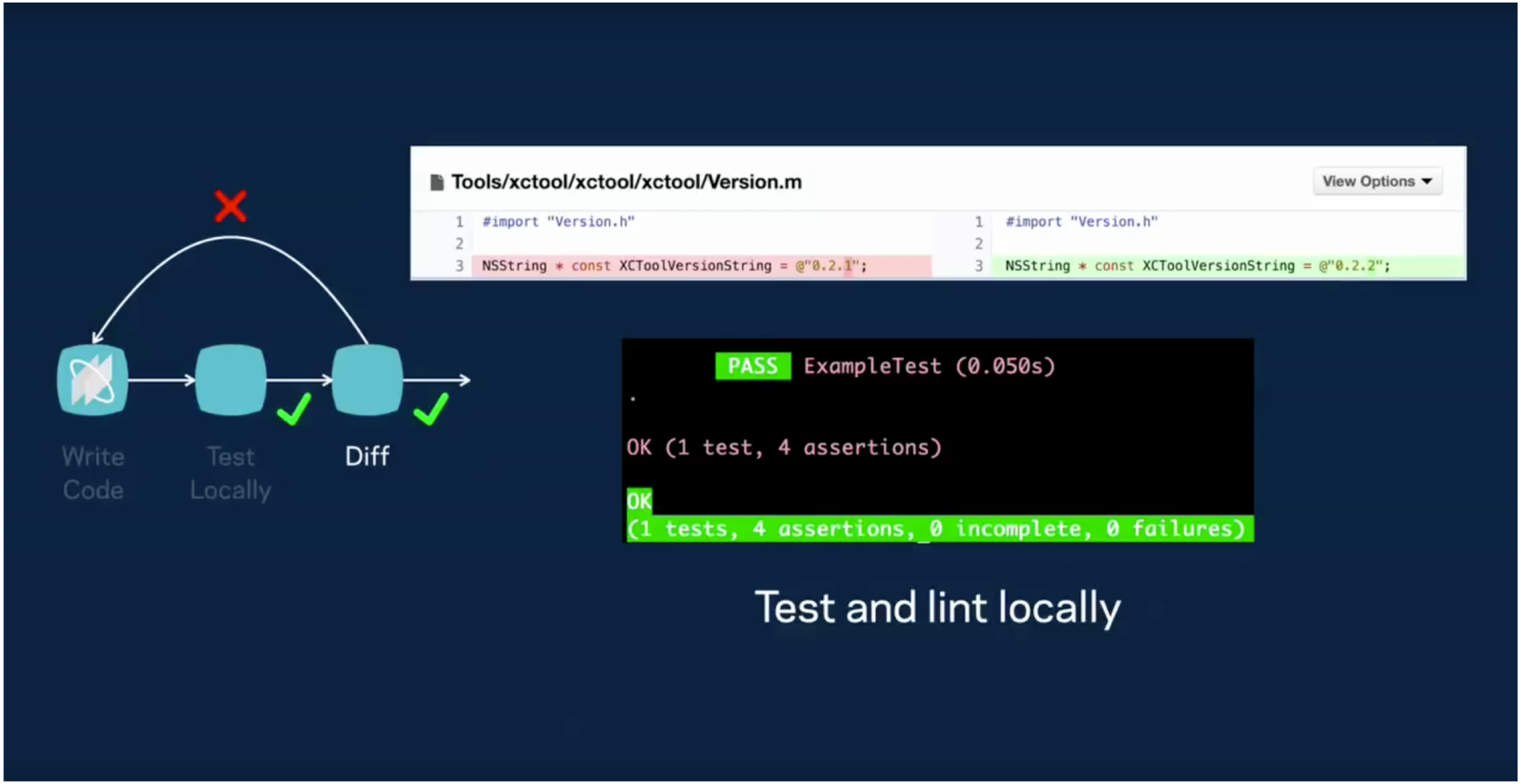
DEVELOPMENT AT SCALE

Releasing at scale in industry

- Facebook: https://atscaleconference.com/videos/rapid-release-at-massive-scale/
- Google: https://www.slideshare.net/JohnMicco1/2016-0425-continuous-integration-at-google-scale
  - https://testing.googleblog.com/2011/06/testing-at-speed-and-scale-of-google.html
- Why Google Stores Billions of Lines of Code in a Single Repository: https://www.youtube.com/watch?v=W71BTkUbdqE
- F8 2015 - Big Code: Developer Infrastructure at Facebook's Scale: https://www.youtube.com/watch?v=X0VH78ye4yY

Pre-2017 release management model at Facebook

Diff lifecycle: local testing



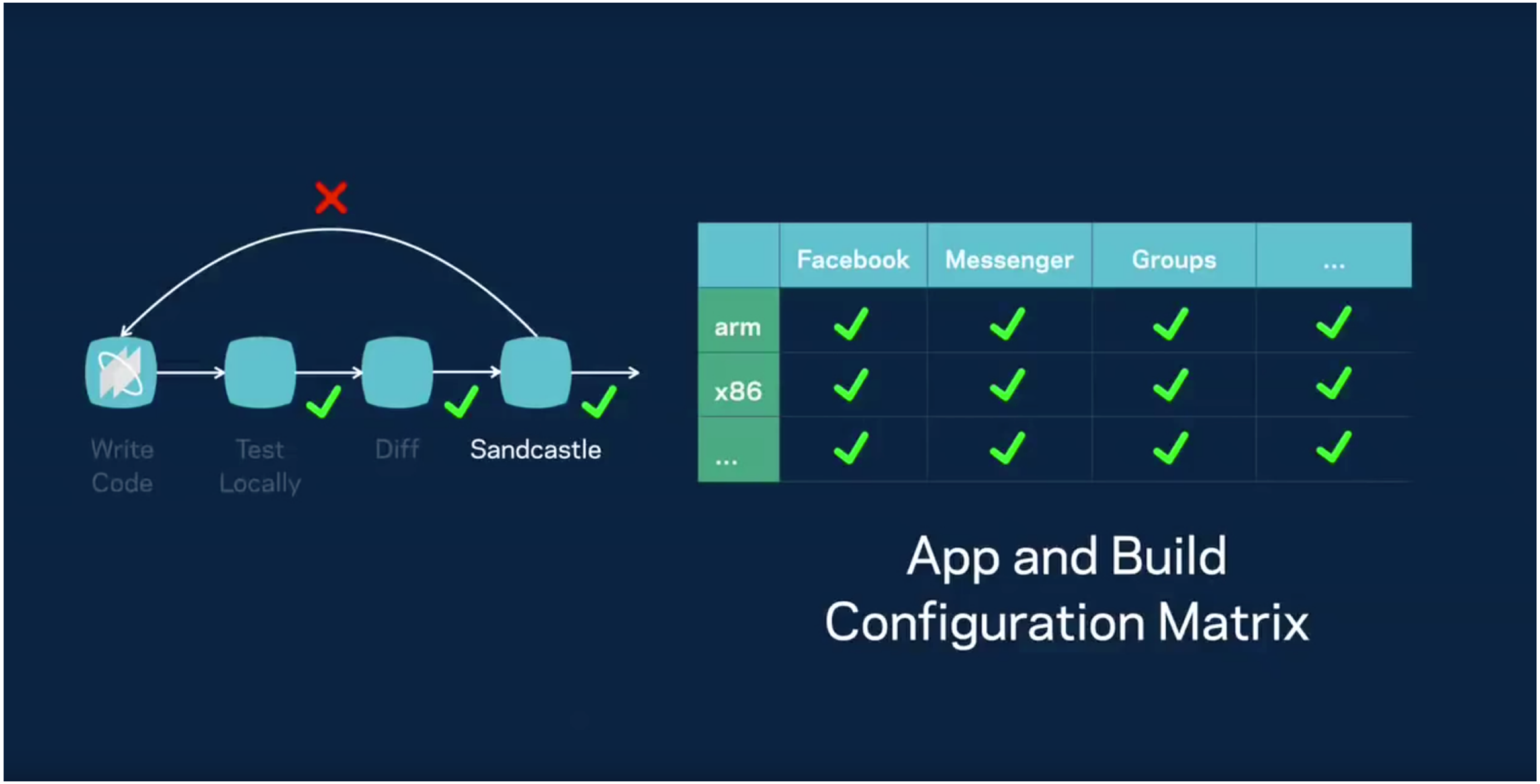
```
#import "Version.h"

NSString * const XCToolVersionString = @"0.2.1";
```

```
#import "Version.h"

NSString * const XCToolVersionString = @"0.2.2";
```

```
        PASS  ExampleTest (0.050s)
.

OK (1 test, 4 assertions)

OK
(1 tests, 4 assertions, 0 incomplete, 0 failures)
```

Test and lint locally

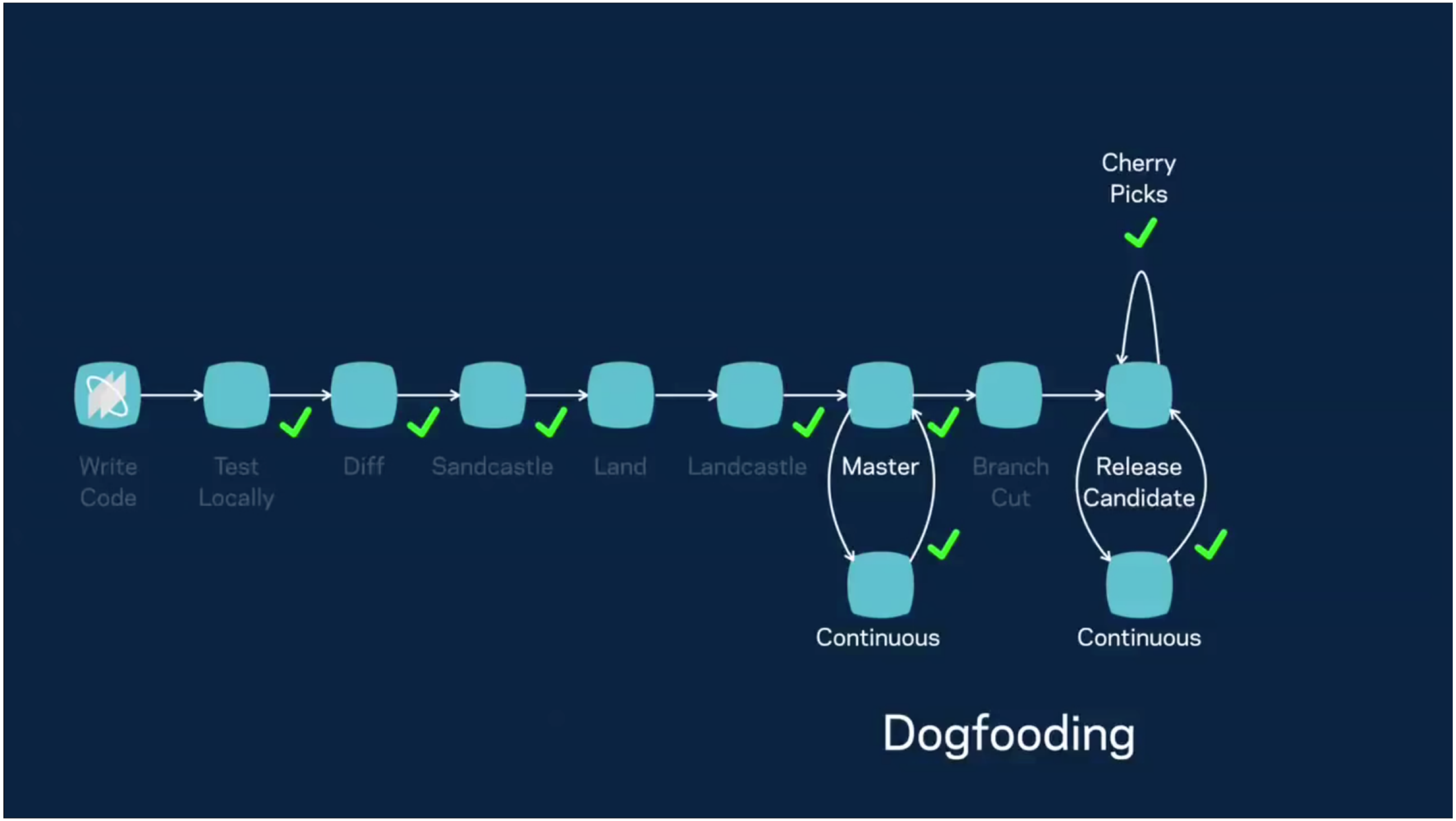
Diff lifecycle: CI testing (data center)



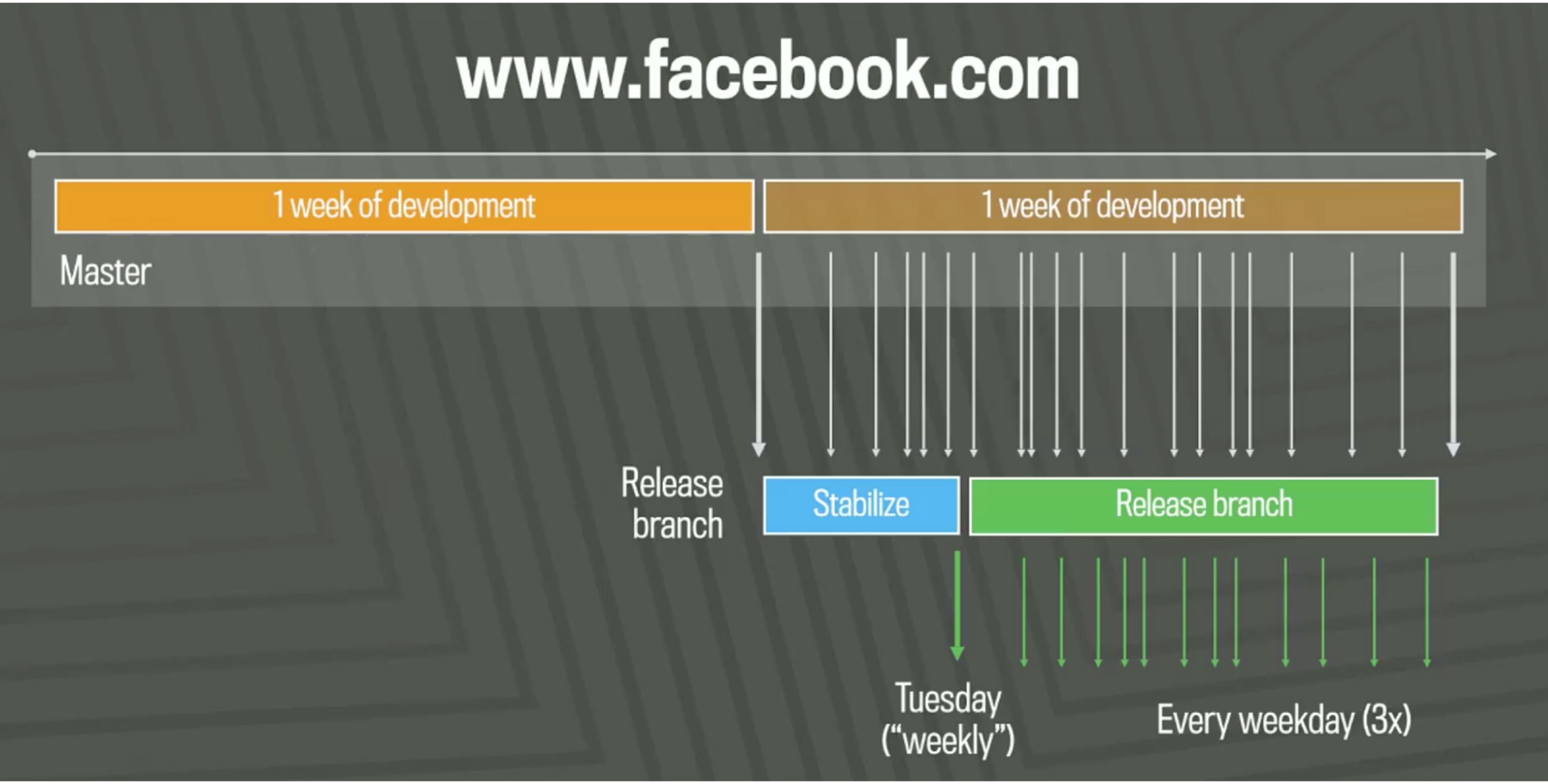
	Facebook	Messenger	Groups	...
arm	✓	✓	✓	✓
x86	✓	✓	✓	✓
...	✓	✓	✓	✓

App and Build Configuration Matrix

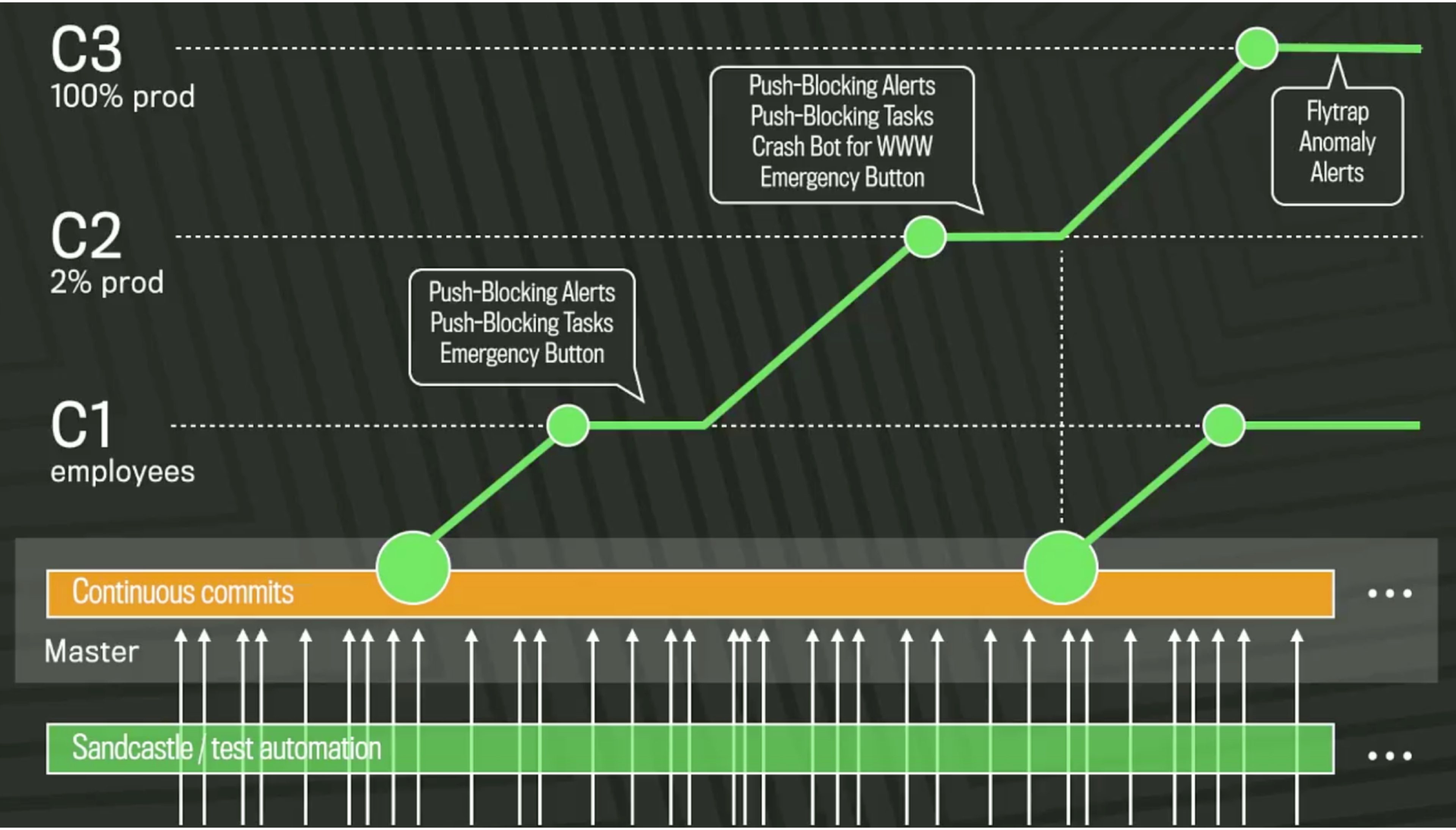
Diff lifecycle: diff ends up on master



Release every two weeks



Quasi-continuous push from master (1,000+ devs, 1,000 diffs/day); 10 pushes/day



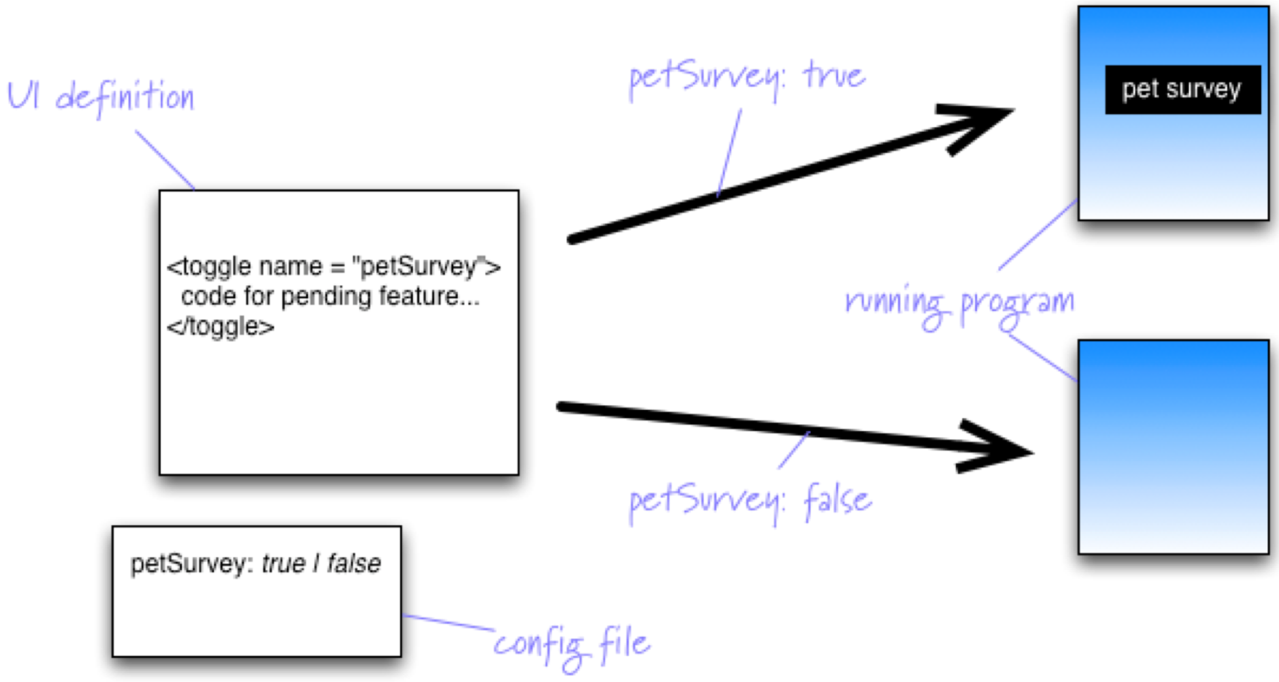
Aside: Key idea – fast to deploy, slow to release

Dark launches at Instagram

- **Early**: Integrate as soon as possible. Find bugs early. Code can run in production about 6 months before being publicly announced (“dark launch”).
- **Often**: Reduce friction. Try things out. See what works. Push small changes just to gather metrics, feasibility testing. Large changes just slow down the team. Do dark launches, to see what performance is in production, can scale up and down. *"Shadow infrastructure" is too expensive, just do in production.*
- **Incremental**: Deploy in increments. Contain risk. Pinpoint issues.

Aside: Feature Flags

Typical way to implement a dark launch.



http://swreflections.blogspot.com/2014/08/feature-toggles-are-one-of-worst-kinds.html
http://martinfowler.com/bliki/FeatureToggle.html

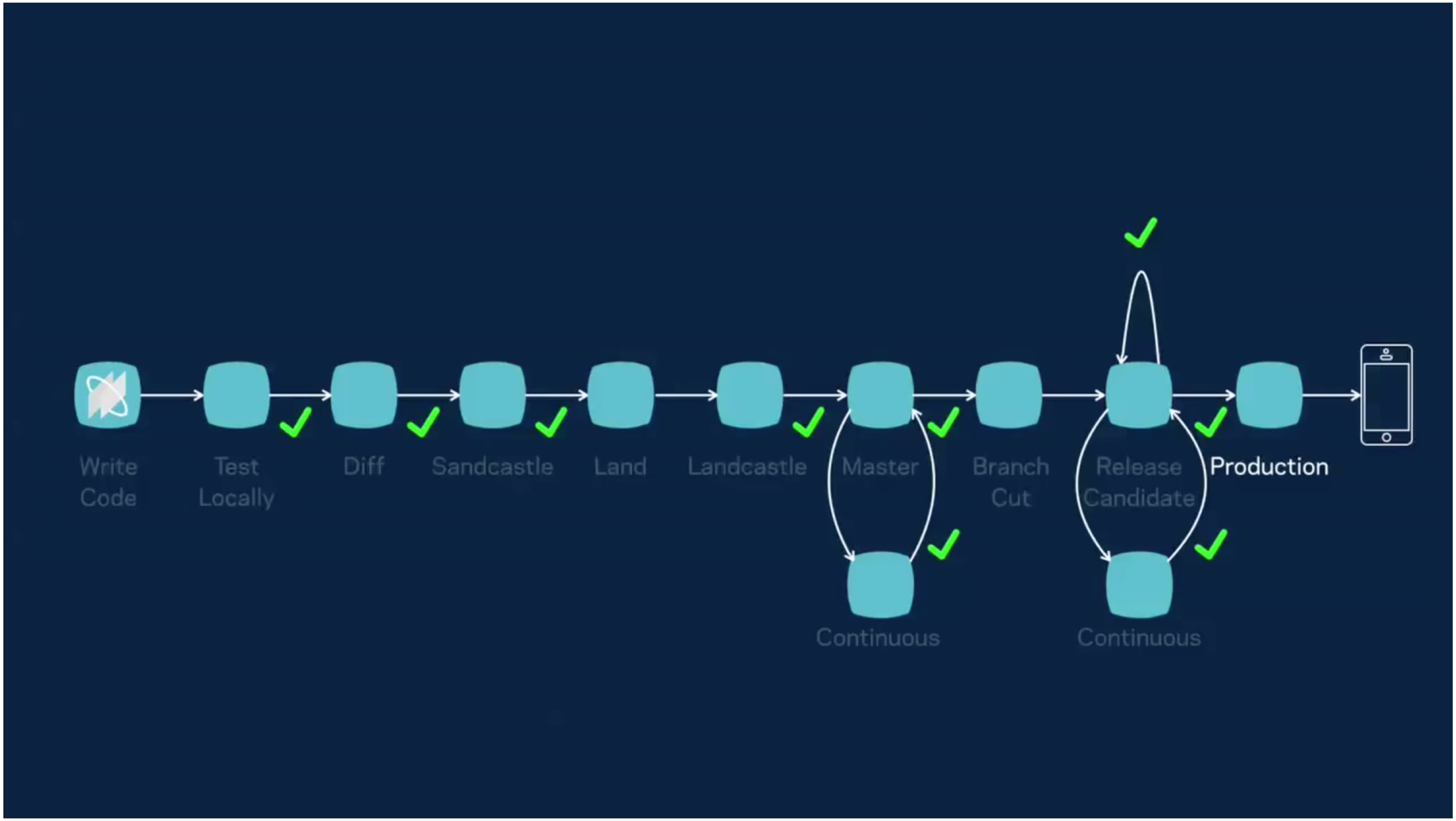
Issues with feature flags

Feature flags are “technical debt”

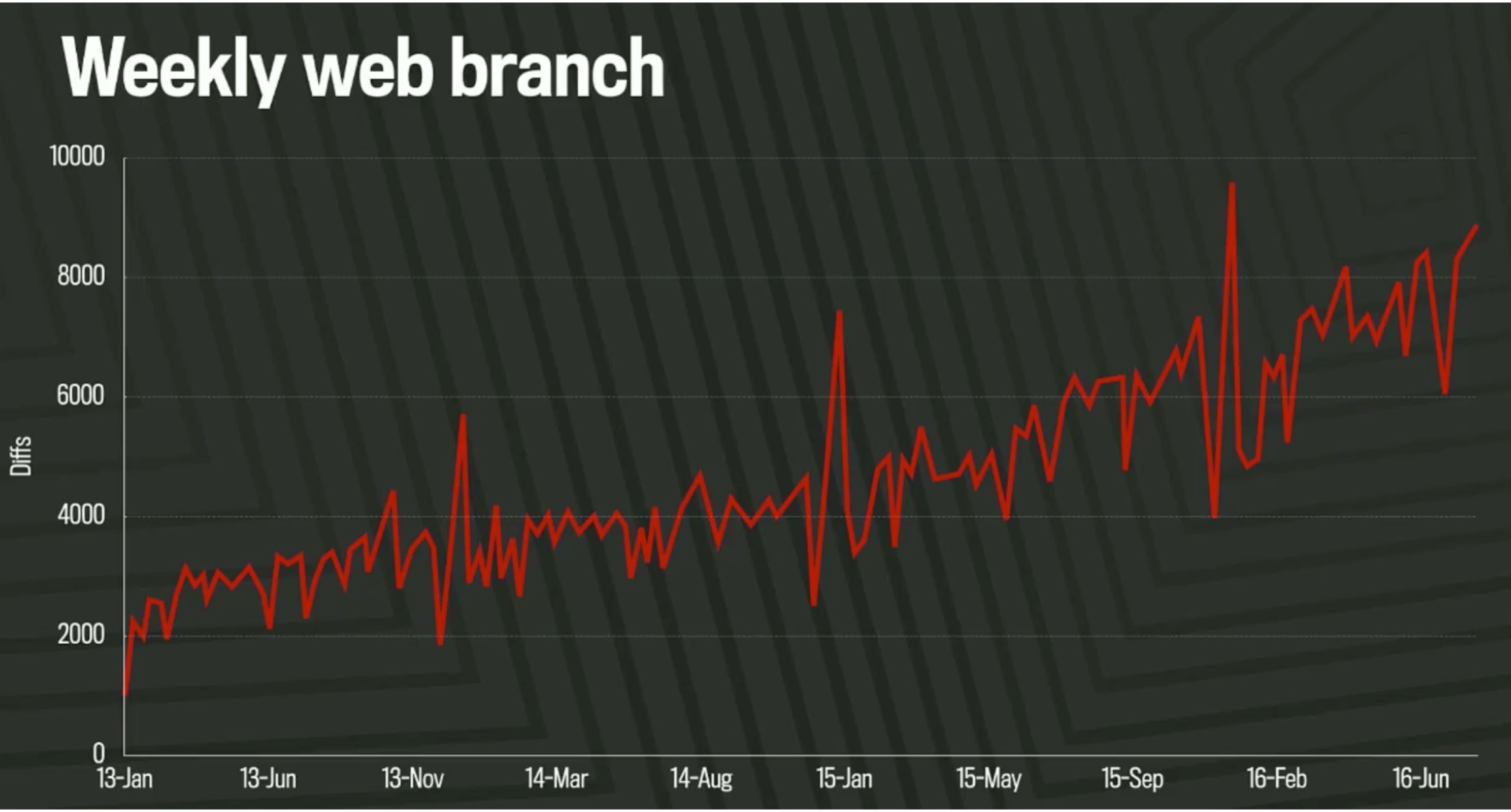
Example: financial services company went bankrupt in 45 minutes.

http://dougseven.com/2014/04/17/knightmare-a-devops-cautionary-tale/

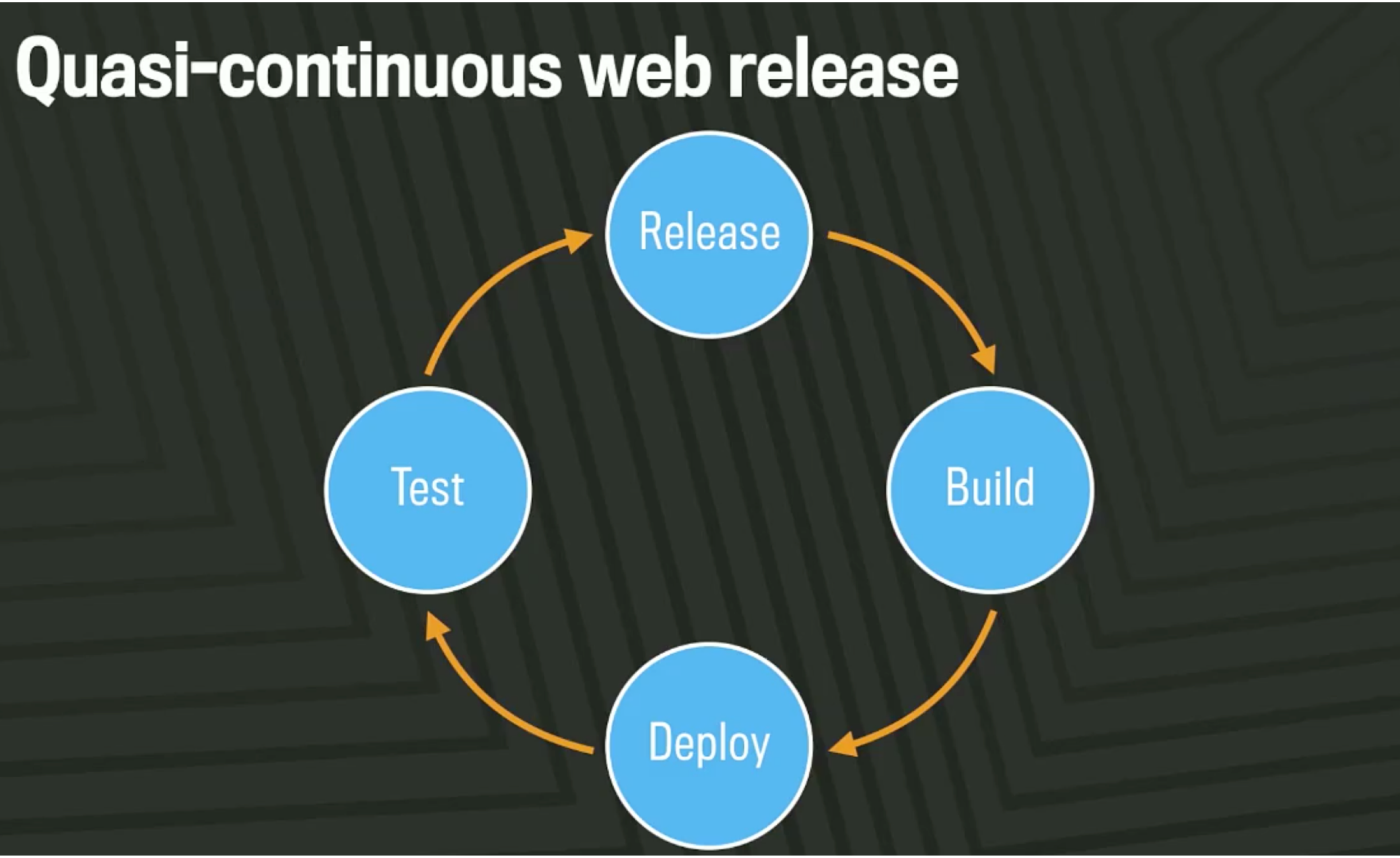
Diff lifecycle: in production



What’s in a weekly branch cut? (The limits of branches)

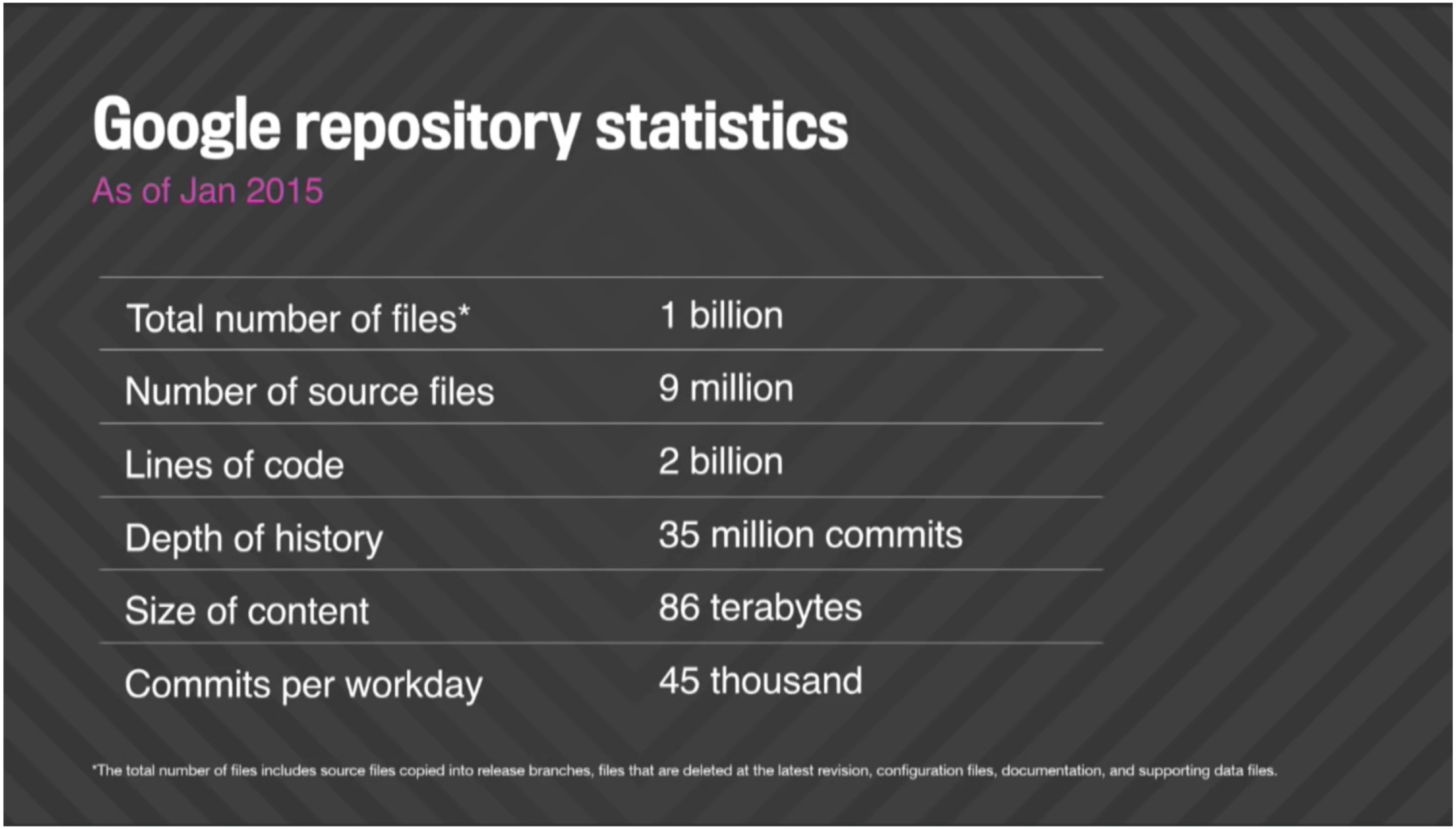


Post-2017 release management model at Facebook



Quasi-continuous web release

Google: similar story. HUGE code base



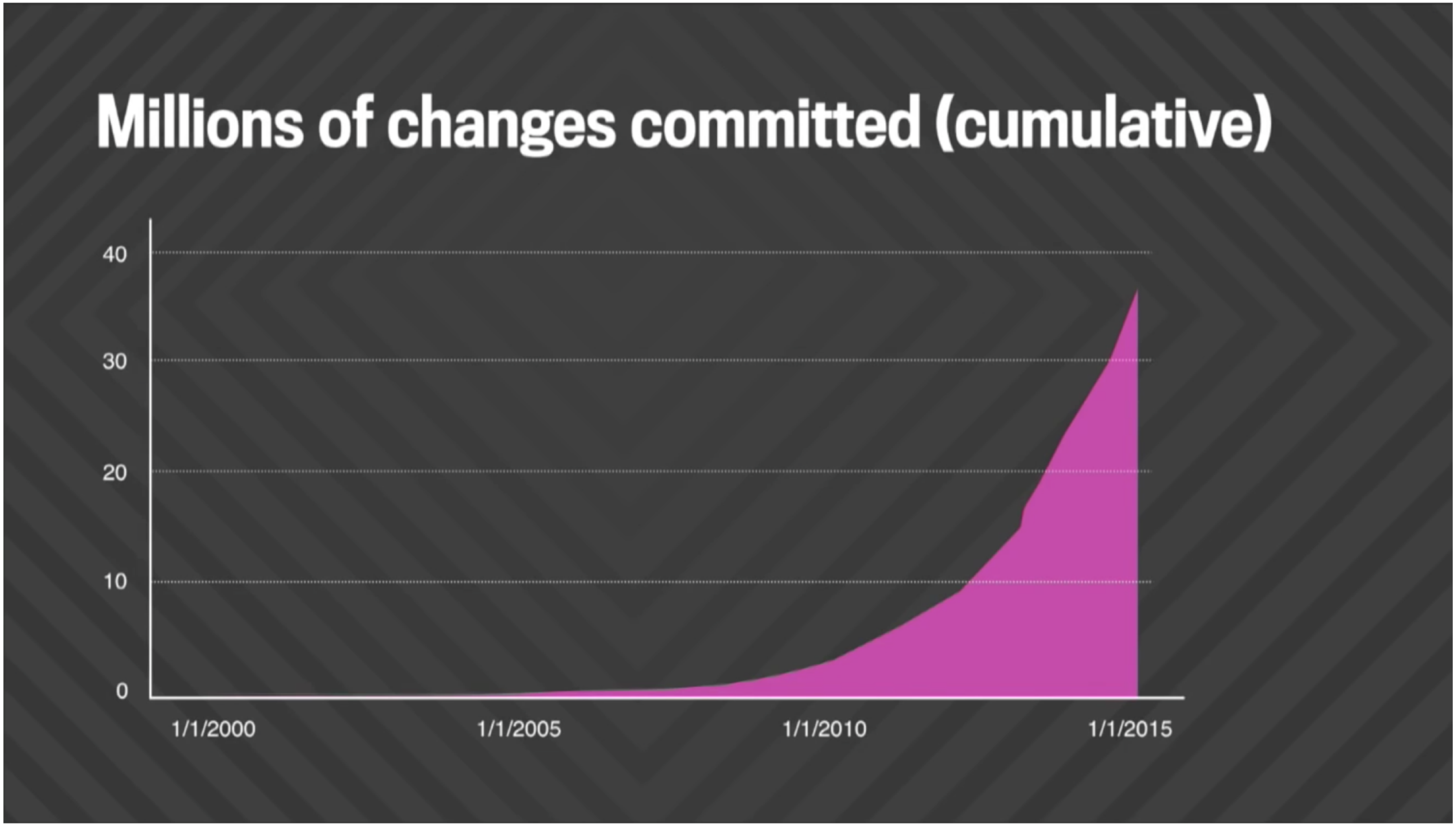
Google repository statistics

As of Jan 2015

Total number of files*	1 billion
Number of source files	9 million
Lines of code	2 billion
Depth of history	35 million commits
Size of content	86 terabytes
Commits per workday	45 thousand

*The total number of files includes source files copied into release branches, files that are deleted at the latest revision, configuration files, documentation, and supporting data files.

Exponential growth



2016 numbers

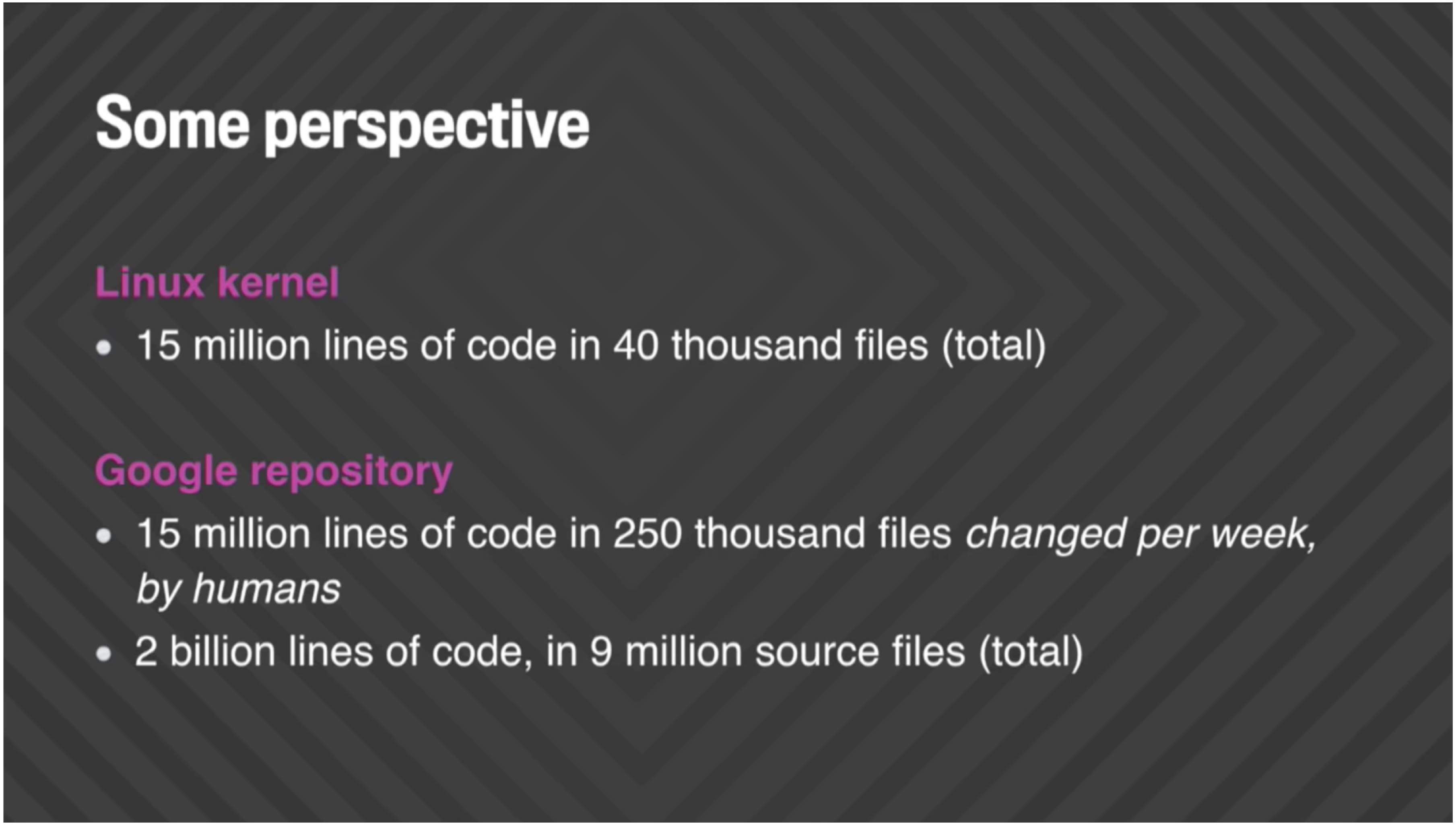
Google Speed and Scale

- >30,000 developers in 40+ offices
- 13,000+ projects under active development
- 30k submissions per day (1 every 3 seconds)

- All builds from source
- 30+ sustained code changes per minute with 90+ peaks
- 50% of code changes monthly
- 150+ million test cases / day, > 150 years of test / day
- Supports continuous deployment for all Google teams!

Google Confidential and Proprietary

Google code base vs Linux kernel code base



Some perspective

Linux kernel

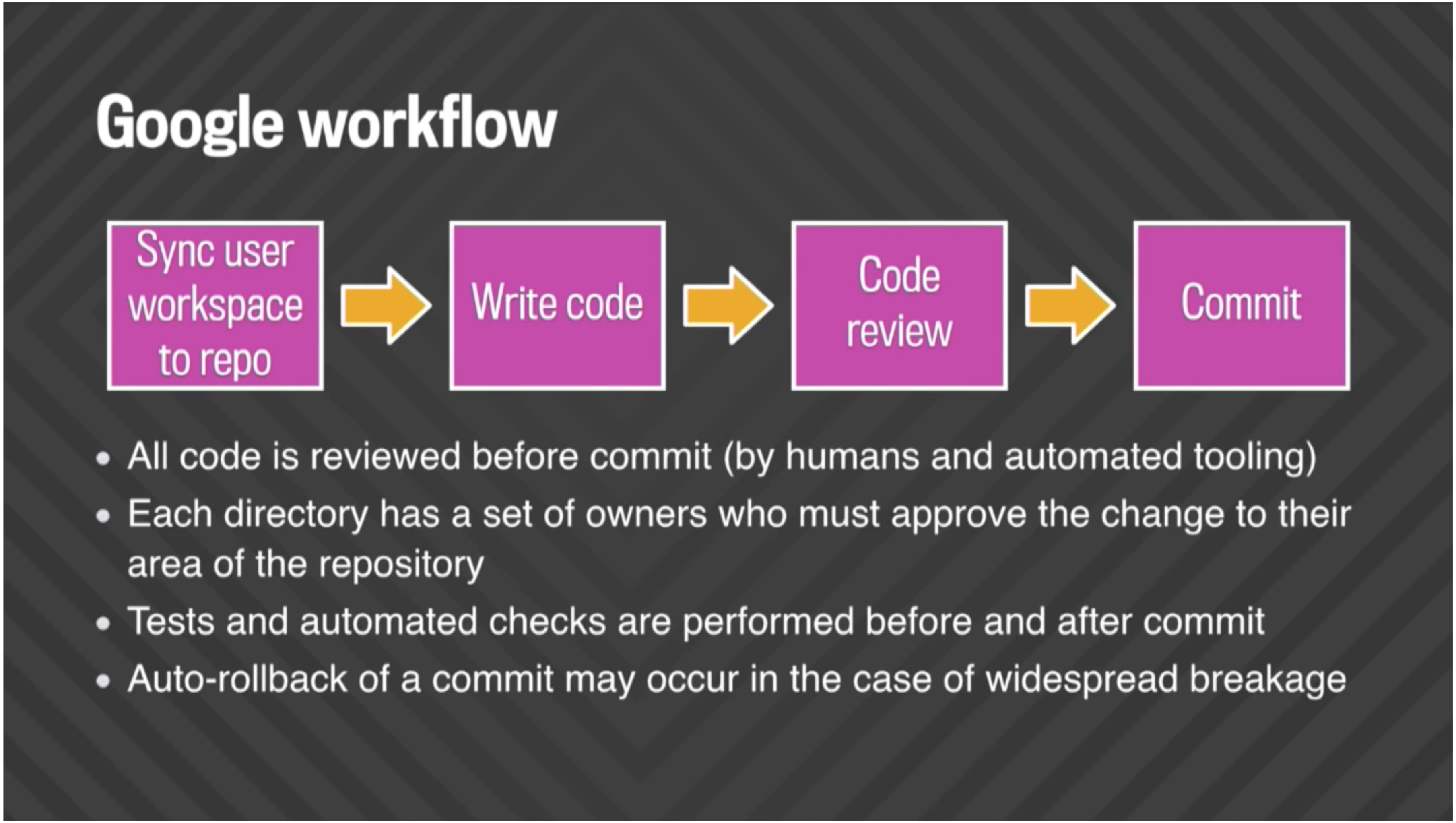
- 15 million lines of code in 40 thousand files (total)

Google repository

- 15 million lines of code in 250 thousand files *changed per week, by humans*
- 2 billion lines of code, in 9 million source files (total)

How do they do it?

1. Lots of (automated) testing

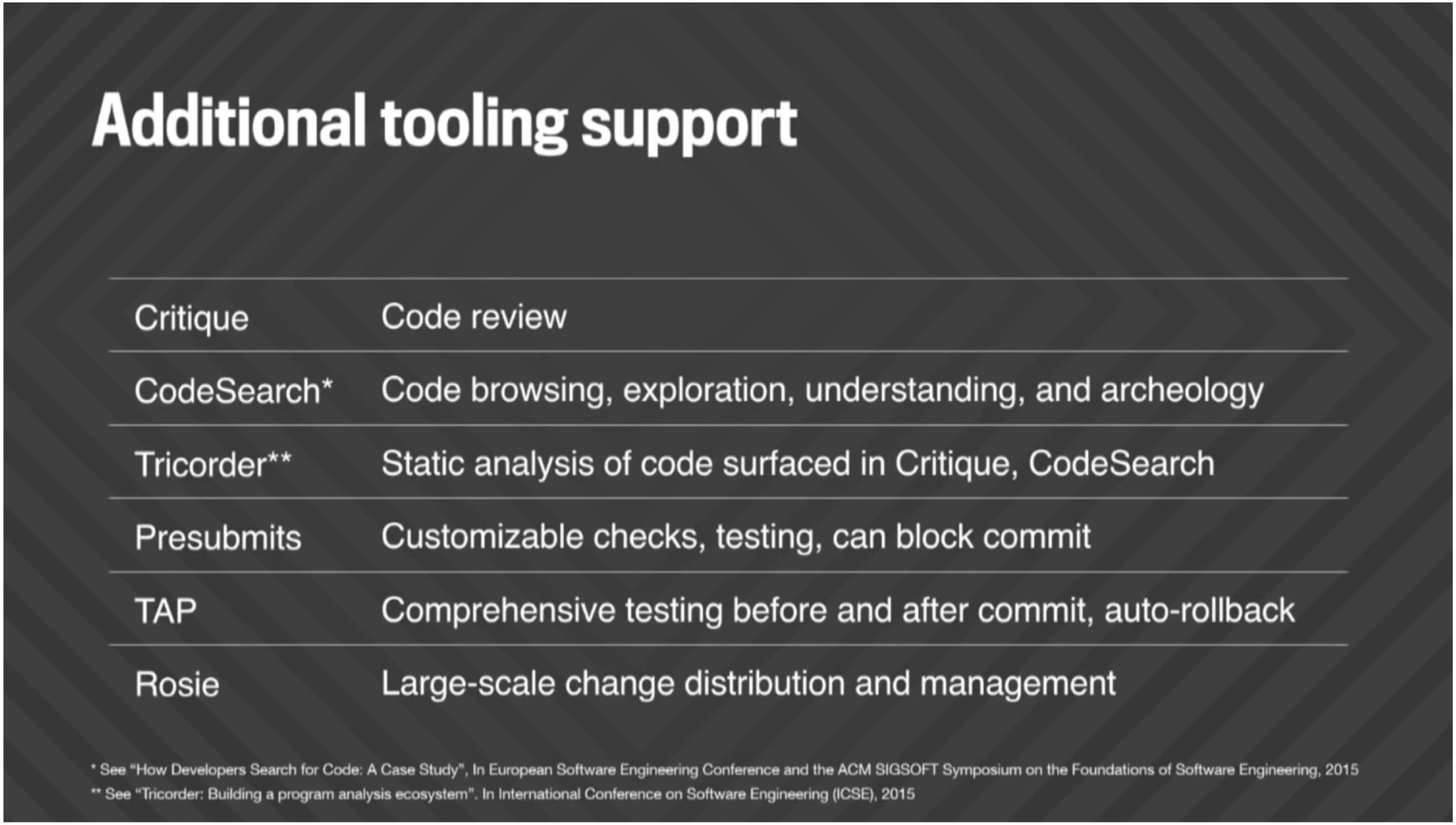


Google workflow

Sync user workspace to repo → Write code → Code review → Commit

- All code is reviewed before commit (by humans and automated tooling)
- Each directory has a set of owners who must approve the change to their area of the repository
- Tests and automated checks are performed before and after commit
- Auto-rollback of a commit may occur in the case of widespread breakage

2. Lots of automation



Additional tooling support

Critique	Code review
CodeSearch*	Code browsing, exploration, understanding, and archeology
Tricorder**	Static analysis of code surfaced in Critique, CodeSearch
Presubmits	Customizable checks, testing, can block commit
TAP	Comprehensive testing before and after commit, auto-rollback
Rosie	Large-scale change distribution and management

* See "How Developers Search for Code: A Case Study", In European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering, 2015

** See "Tricorder: Building a program analysis ecosystem". In International Conference on Software Engineering (ICSE), 2015

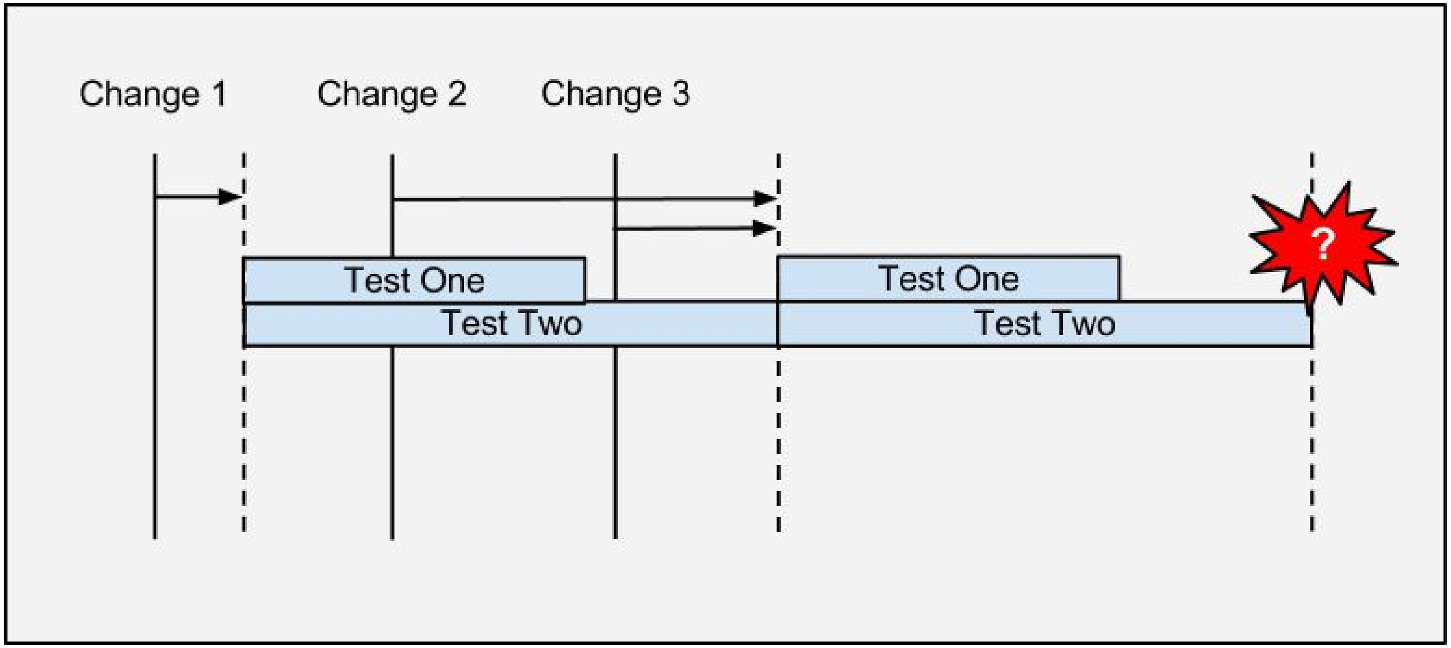
3. Smarter tooling

- Build system
- Version control
- …

3a. Build system

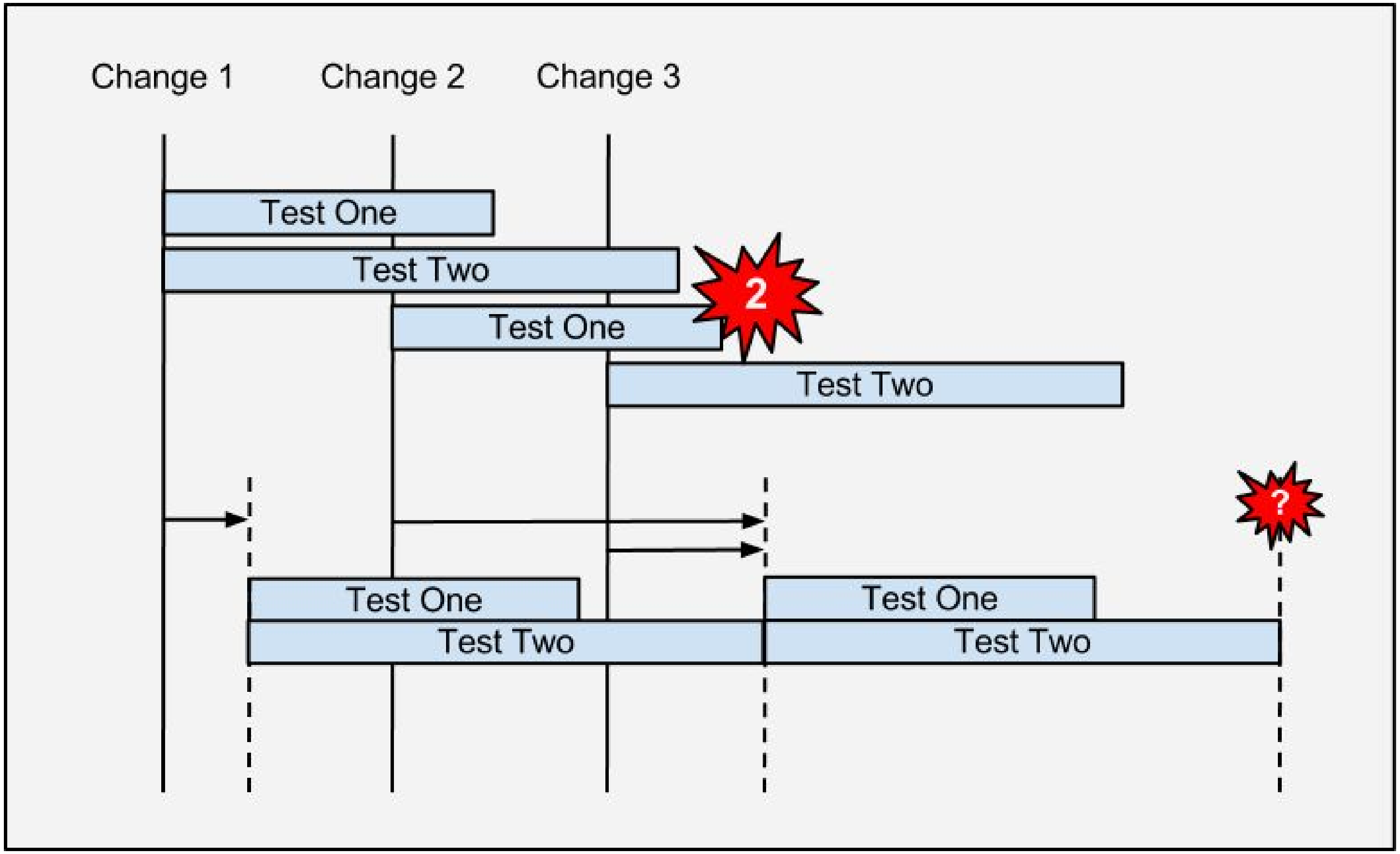
Google Standard Continuous Build System

- Triggers builds in continuous cycle
- Cycle time = longest build + test cycle
- Tests many changes together
- Which change broke the build?



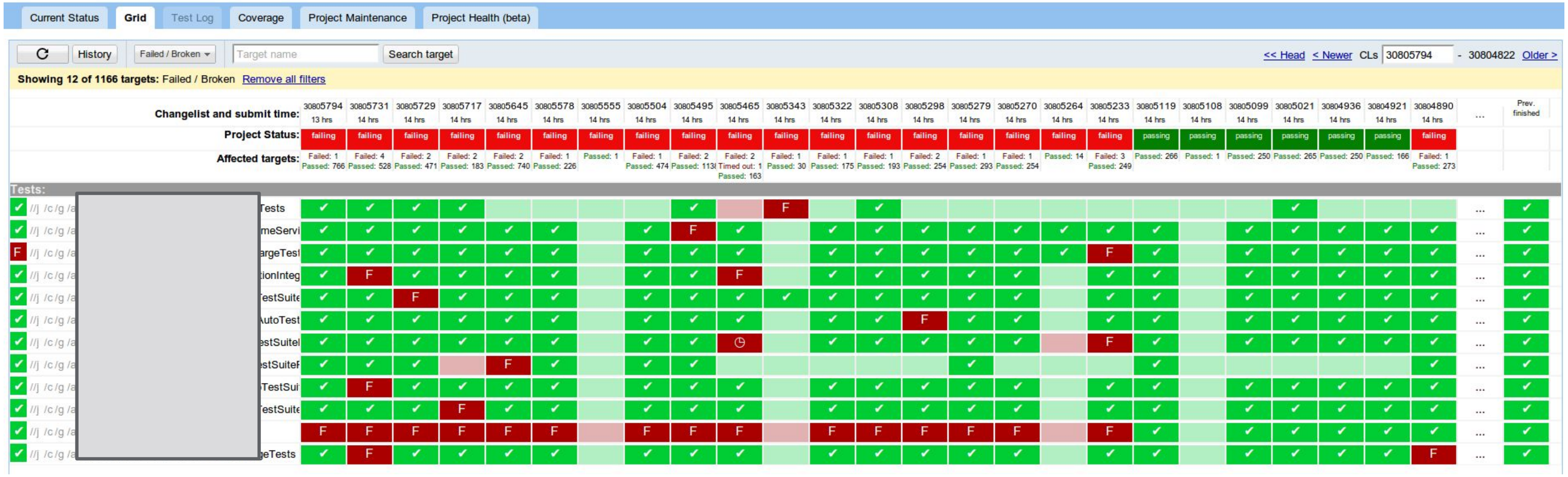
Google Continuous Build System

- Triggers tests on every change
- Uses fine-grained dependencies
- Change 2 broke test 1



Google Confidential and Proprietary

Google Continuous Integration Display



Showing 12 of 1166 targets: Failed / Broken Remove all filters

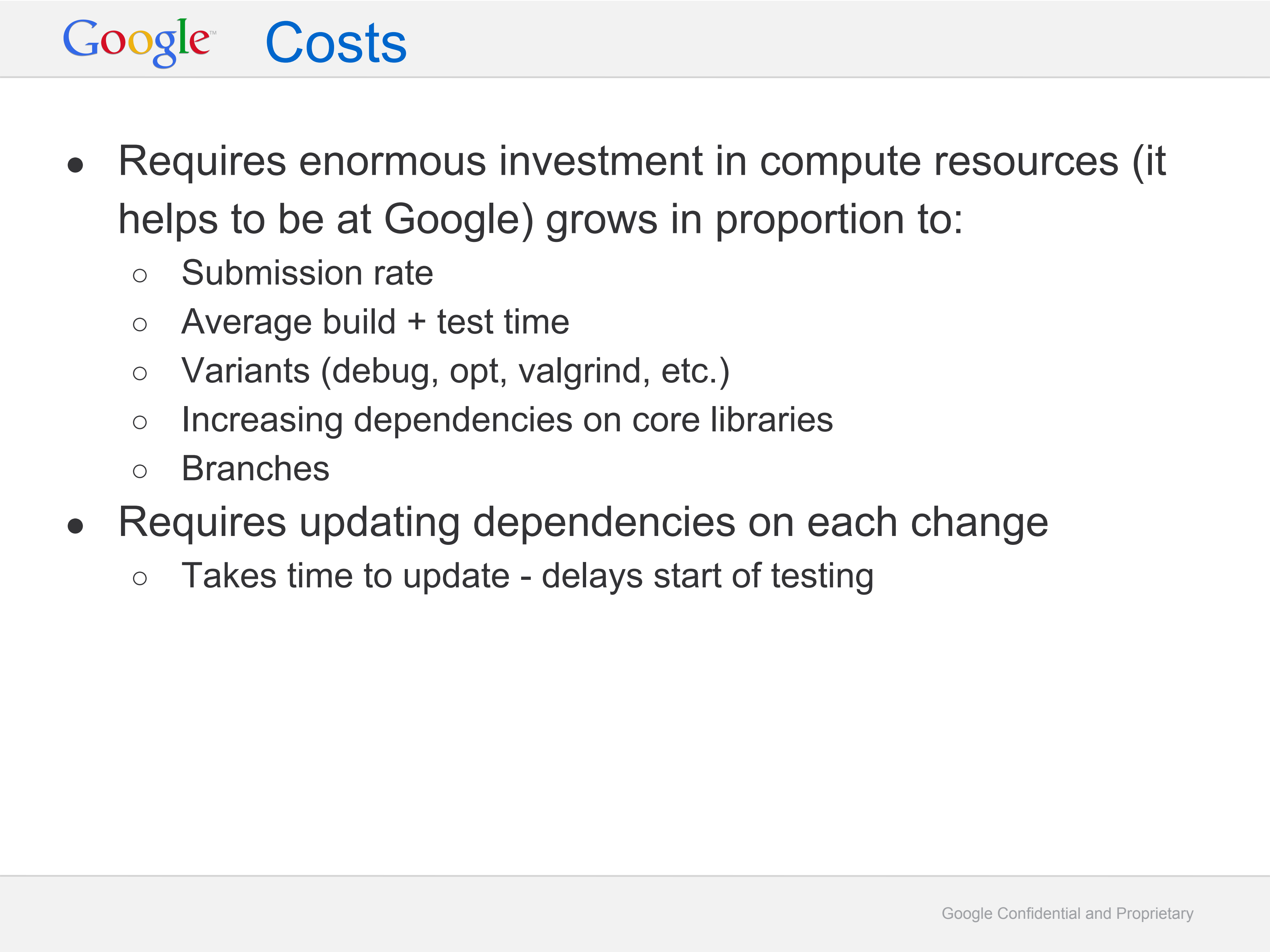
	Changelist and submit time:	30805794 13 hrs	30805731 14 hrs	30805729 14 hrs	30805717 14 hrs	30805645 14 hrs	30805578 14 hrs	30805555 14 hrs	30805504 14 hrs	30805495 14 hrs	30805465 14 hrs	30805343 14 hrs	30805322 14 hrs	30805308 14 hrs	30805298 14 hrs	30805279 14 hrs	30805270 14 hrs	30805264 14 hrs	30805233 14 hrs	30805119 14 hrs	30805108 14 hrs	30805099 14 hrs	30805021 14 hrs	30804936 14 hrs	30804921 14 hrs	30804890 14 hrs	...	Prev. finished
	Project Status:	failing	failing	failing	failing	failing	failing	failing	failing	failing	failing	failing	failing	failing	failing	failing	failing	failing	failing	passing	passing	passing	passing	passing	passing	failing		
	Affected targets:	Failed: 1 Passed: 766	Failed: 4 Passed: 528	Failed: 2 Passed: 471	Failed: 2 Passed: 183	Failed: 2 Passed: 740	Failed: 1 Passed: 226	Passed: 1	Failed: 1 Passed: 474	Failed: 2 Passed: 113	Failed: 2 Timed out: 1 Passed: 163	Failed: 1 Passed: 30	Failed: 1 Passed: 175	Failed: 1 Passed: 193	Failed: 2 Passed: 254	Failed: 1 Passed: 293	Failed: 1 Passed: 254	Passed: 14	Failed: 3 Passed: 249	Passed: 266	Passed: 1	Passed: 250	Passed: 265	Passed: 250	Passed: 166	Failed: 1 Passed: 273		
Tests:																												
✓	//j /c/g /a [redacted] Tests	✓	✓	✓	✓					✓		F		✓									✓				...	✓
✓	//j /c/g /a [redacted] meServi	✓	✓	✓	✓	✓	✓		✓	F	✓		✓	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓	...	✓
F	//j /c/g /a [redacted] argeTest	✓	✓	✓	✓	✓	✓		✓	✓	✓		✓	✓	✓	✓	✓	✓	F	✓		✓	✓	✓	✓	✓	...	✓
✓	//j /c/g /a [redacted] ionInteg	✓	F	✓	✓	✓	✓		✓	✓	F		✓	✓	✓	✓	✓		✓	✓		✓	✓	✓	✓	✓	...	✓
✓	//j /c/g /a [redacted] TestSuite	✓	✓	F	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓		✓	✓		✓	✓	✓	✓	✓	...	✓
✓	//j /c/g /a [redacted] AutoTest	✓	✓	✓	✓	✓	✓		✓	✓	✓		✓	✓	F	✓	✓		✓	✓		✓	✓	✓	✓	✓	...	✓
✓	//j /c/g /a [redacted] estSuitel	✓	✓	✓	✓	✓	✓		✓	✓	◷		✓	✓	✓	✓	✓		F	✓		✓	✓	✓	✓	✓	...	✓
✓	//j /c/g /a [redacted] estSuiteF	✓	✓	✓		F	✓		✓	✓						✓				✓						✓	...	✓
✓	//j /c/g /a [redacted] TestSui	✓	F	✓	✓	✓	✓		✓	✓	✓		✓	✓	✓	✓	✓		✓	✓		✓	✓	✓	✓	✓	...	✓
✓	//j /c/g /a [redacted] TestSuite	✓	✓	✓	F	✓	✓		✓	✓	✓		✓	✓	✓	✓	✓		✓	✓		✓	✓	✓	✓	✓	...	✓
✓	//j /c/g /a [redacted]	F	F	F	F	F	F		F	F	F		F	F	F	F	F		F	✓		✓	✓	✓	✓	✓	...	✓
✓	//j /c/g /a [redacted] geTests	✓	F	✓	✓	✓	✓		✓	✓	✓		✓	✓	✓	✓	✓		✓	✓		✓	✓	✓	✓	F	...	✓

Google Confidential and Proprietary

Google

Benefits

- Identifies failures sooner
- Identifies culprit change precisely
  - Avoids divide-and-conquer and tribal knowledge
- Lower compute costs using fine grained dependencies
- Keeps the build green by reducing time to fix breaks
- Accepted enthusiastically by product teams
- Enables teams to ship with fast iteration times
  - Supports submit-to-production times of less than 36 hours for some projects



Costs

- Requires enormous investment in compute resources (it helps to be at Google) grows in proportion to:
  - Submission rate
  - Average build + test time
  - Variants (debug, opt, valgrind, etc.)
  - Increasing dependencies on core libraries
  - Branches
- Requires updating dependencies on each change
  - Takes time to update - delays start of testing

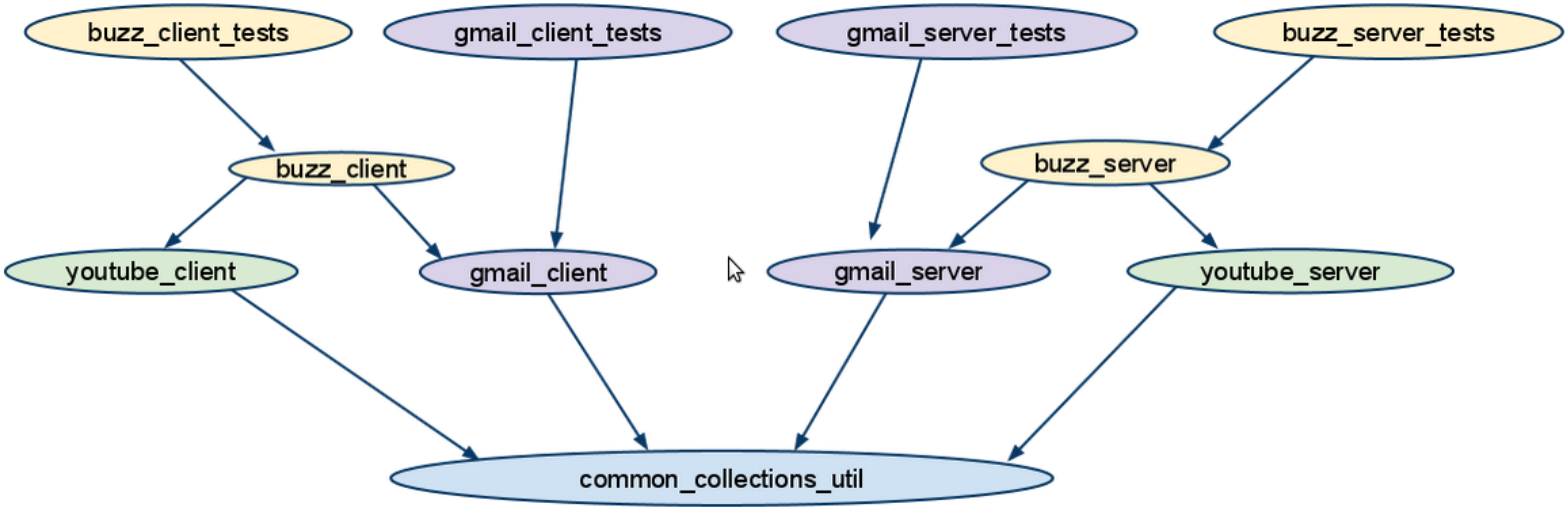
Which tests to run?

GMAIL

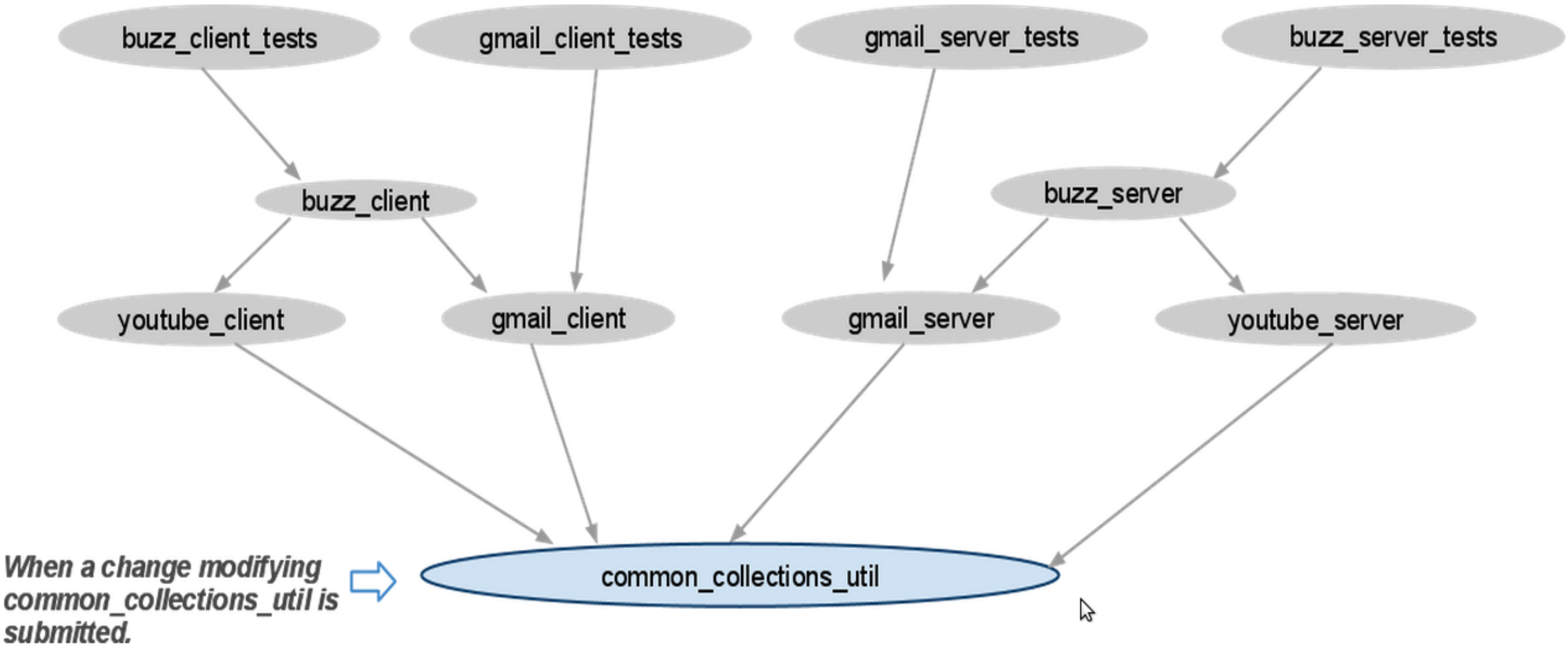
Test Target:
name: //depot/gmail_client_tests
name: //depot/gmail_server_tests

BUZZ

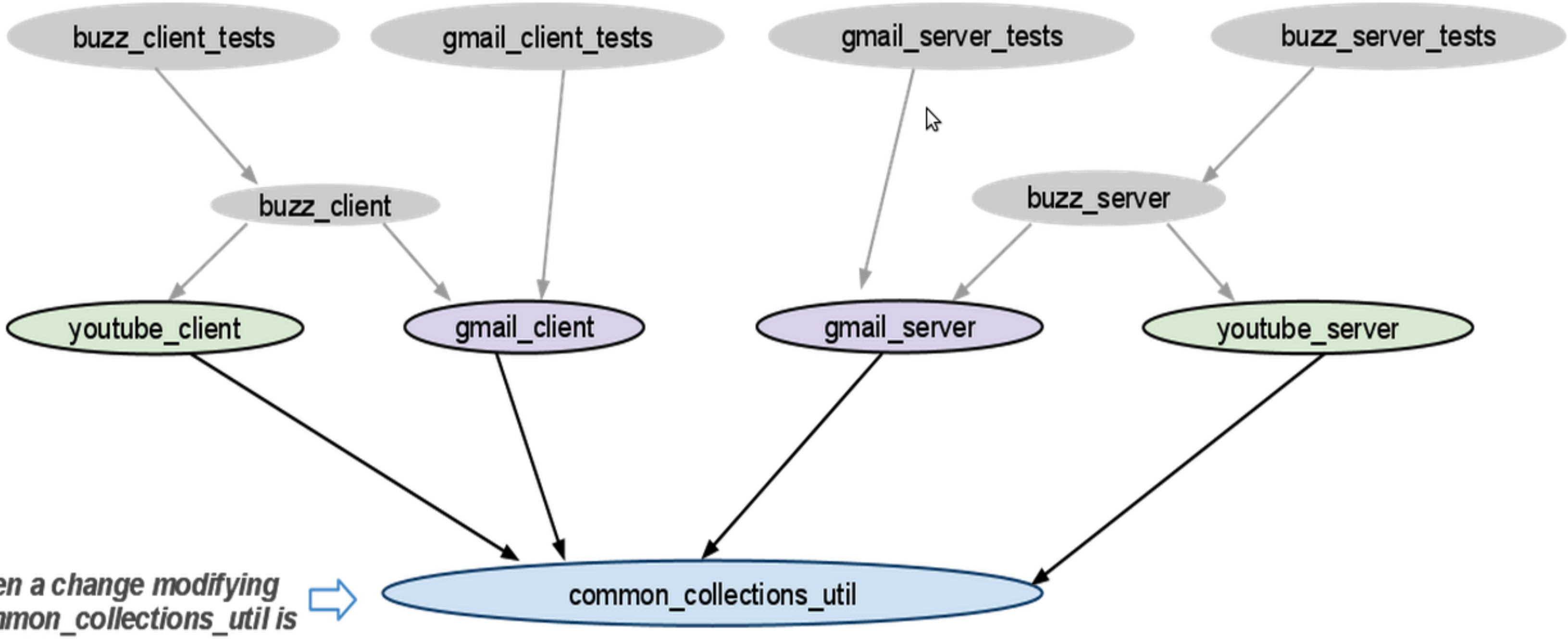
Test targets:
name: //depot/buzz_server_tests
name: //depot/buzz_client_tests



Scenario 1: a change modifies common_collections_util

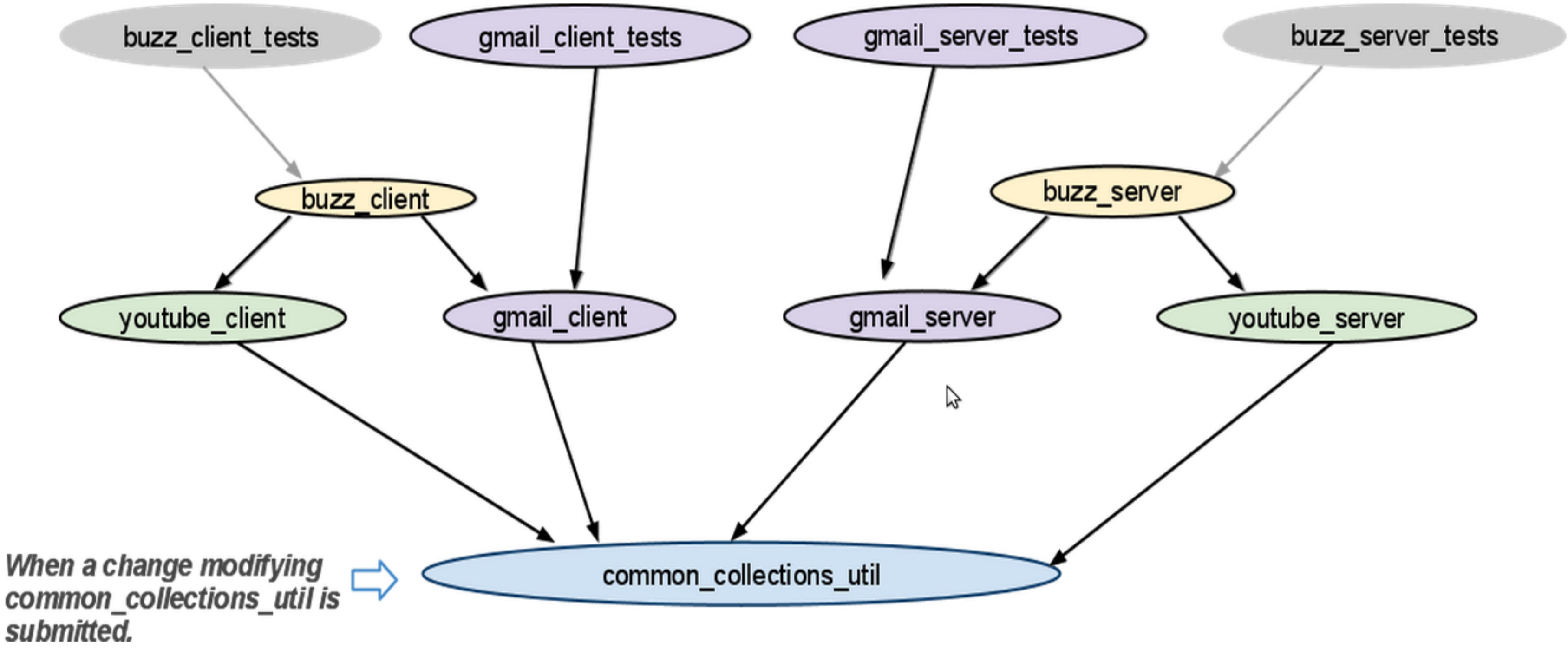


Scenario 1: a change modifies common_collections_util

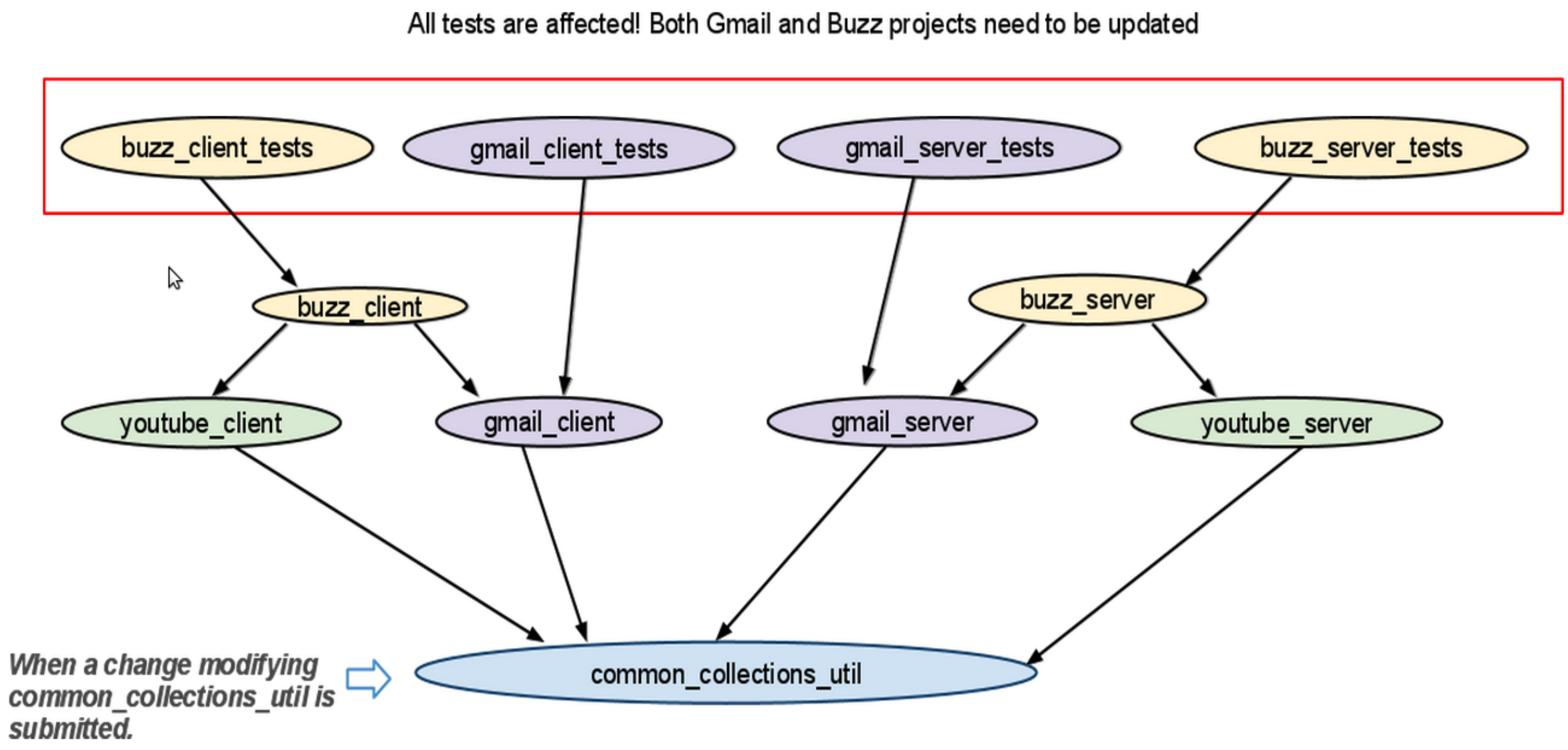


When a change modifying common_collections_util is submitted.

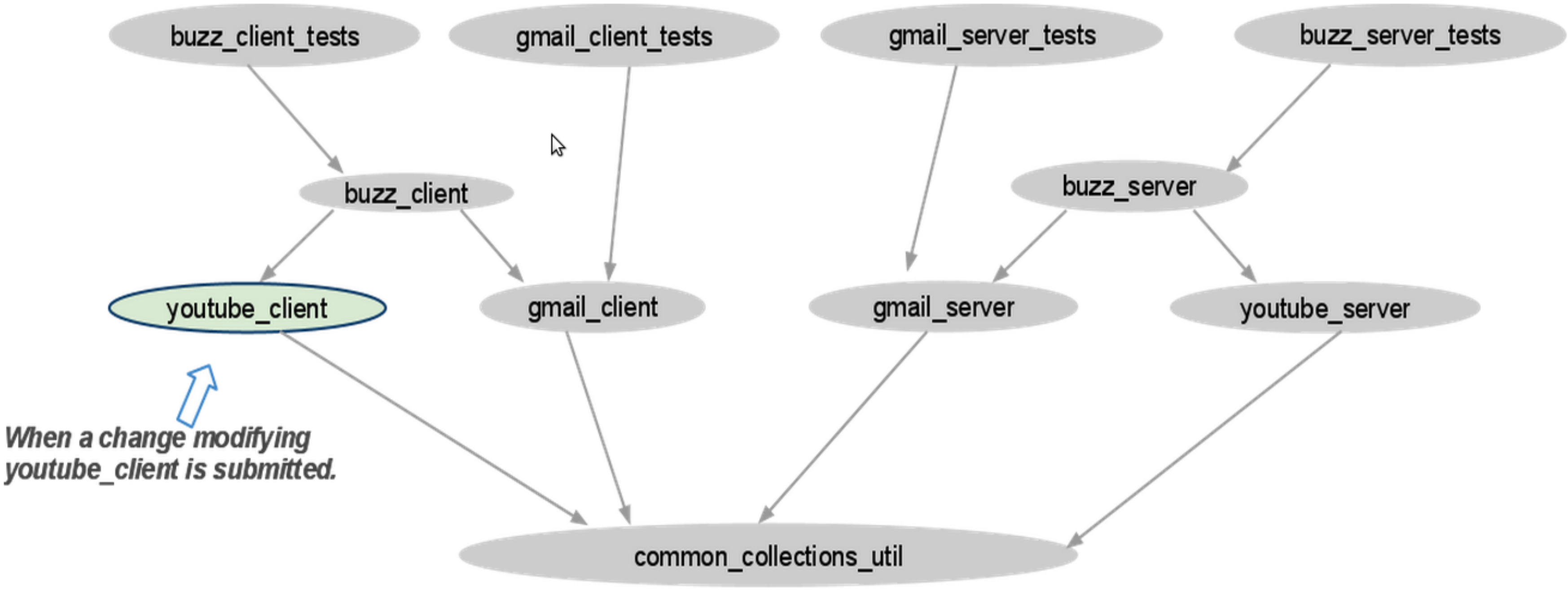
Scenario 1: a change modifies common_collections_util



Scenario 1: a change modifies common_collections_util

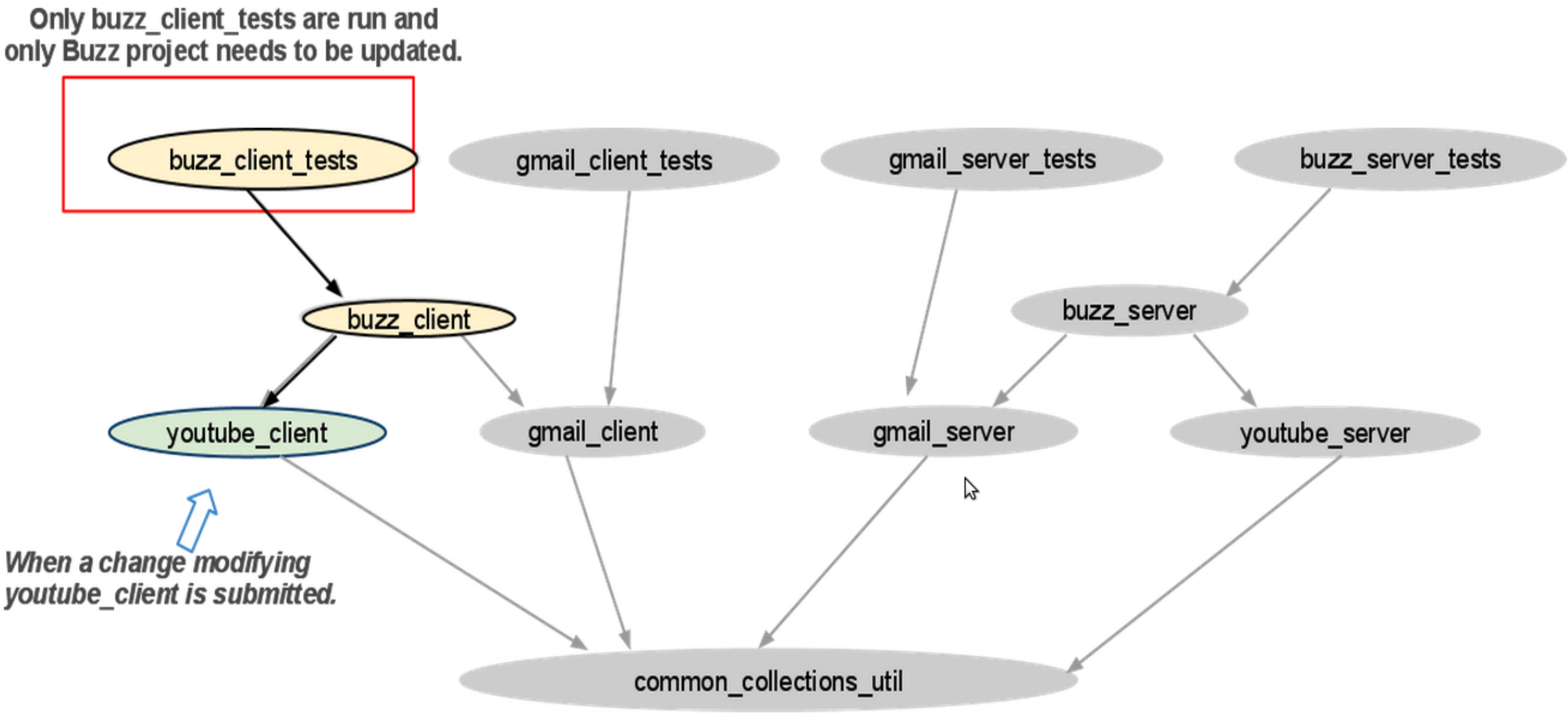


Scenario 2: a change modifies the youtube_client



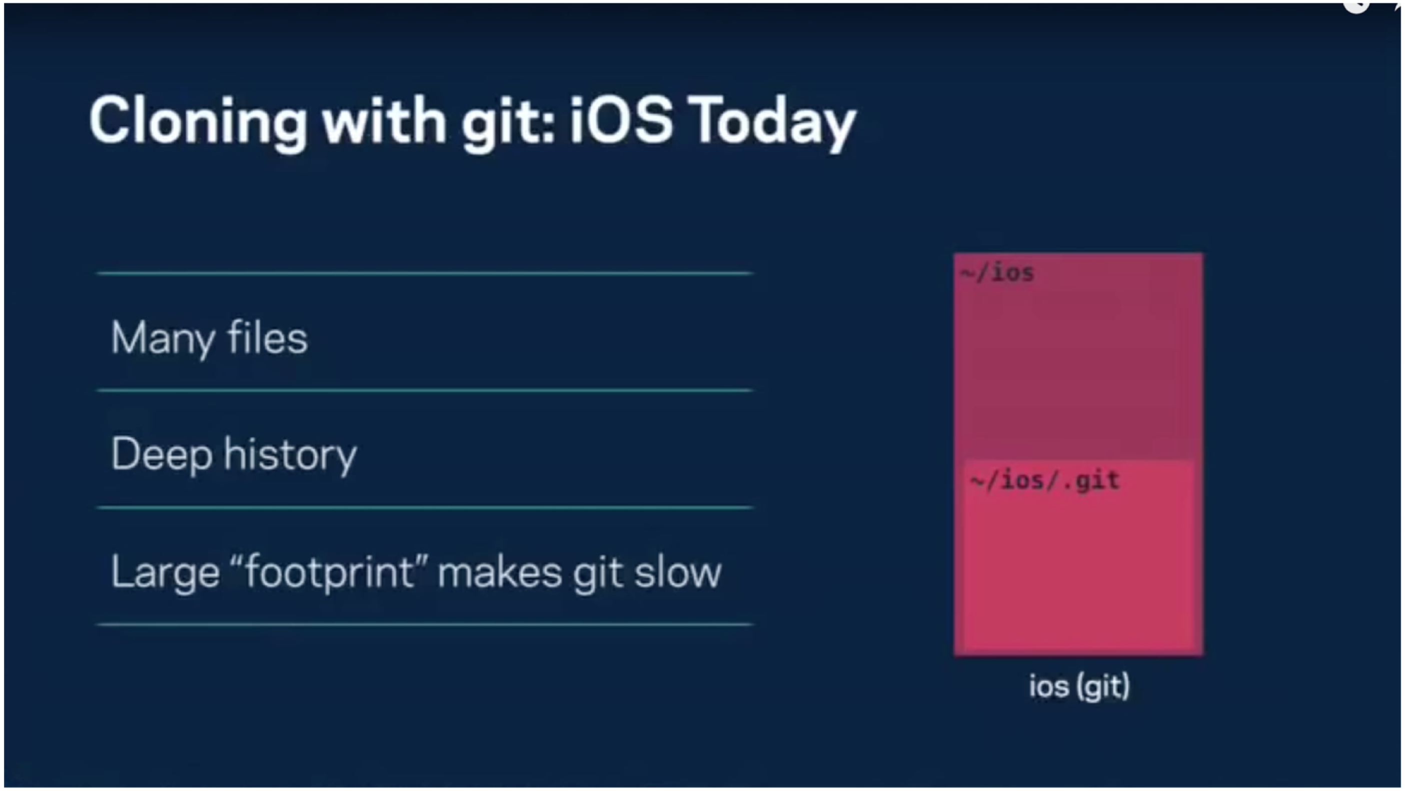
When a change modifying youtube_client is submitted.

Scenario 2: a change modifies the youtube_client



3b. Version control

- Problem: even git can get slow at Facebook scale
  - 1M+ source control commands run per day
  - 100K+ commits per week

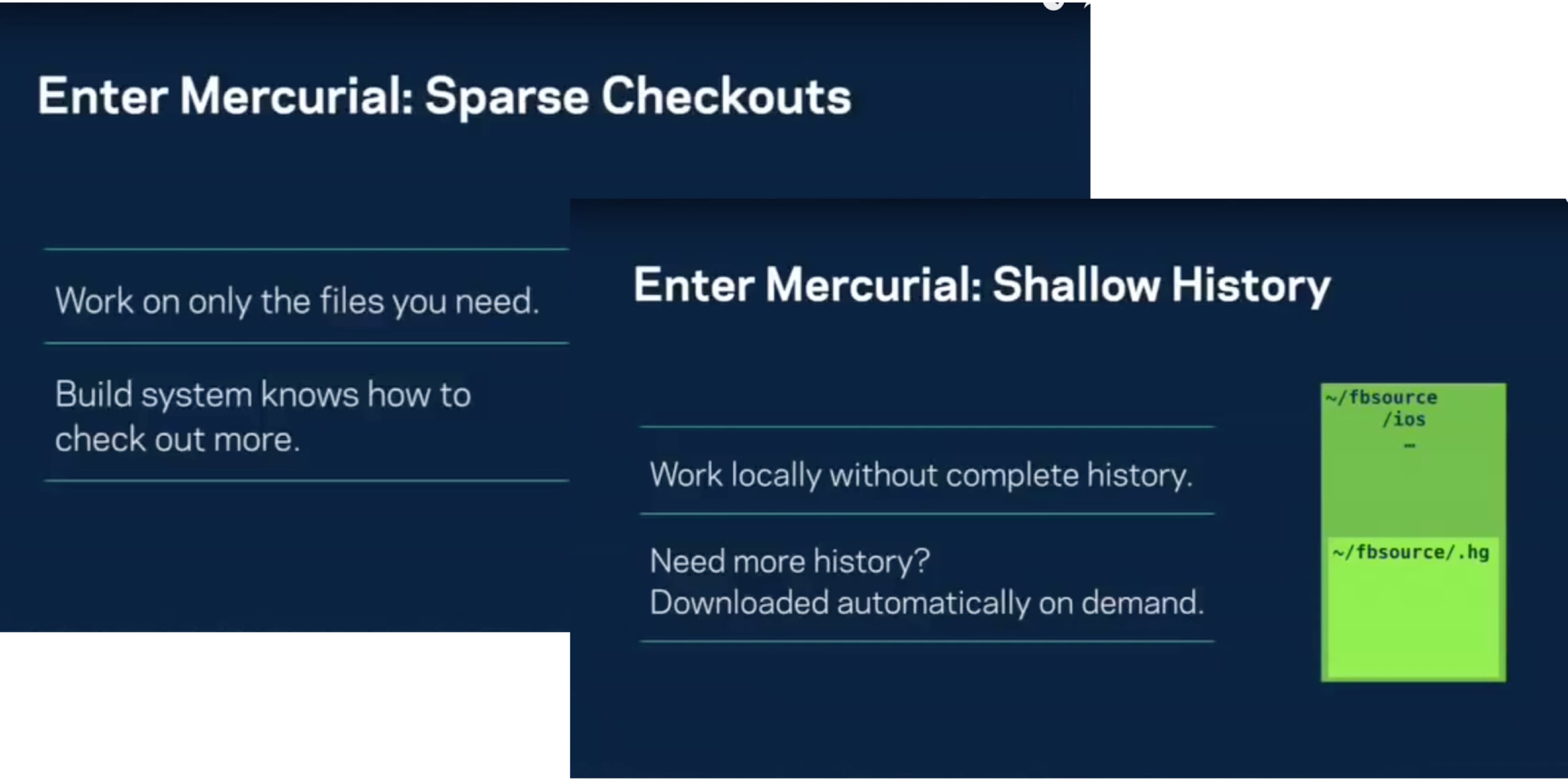


Cloning with git: iOS Today

- Many files
- Deep history
- Large “footprint” makes git slow

3b. Version control

- Solution: redesign version control



Enter Mercurial: Sparse Checkouts

Work on only the files you need.

Build system knows how to check out more.

Enter Mercurial: Shallow History

Work locally without complete history.

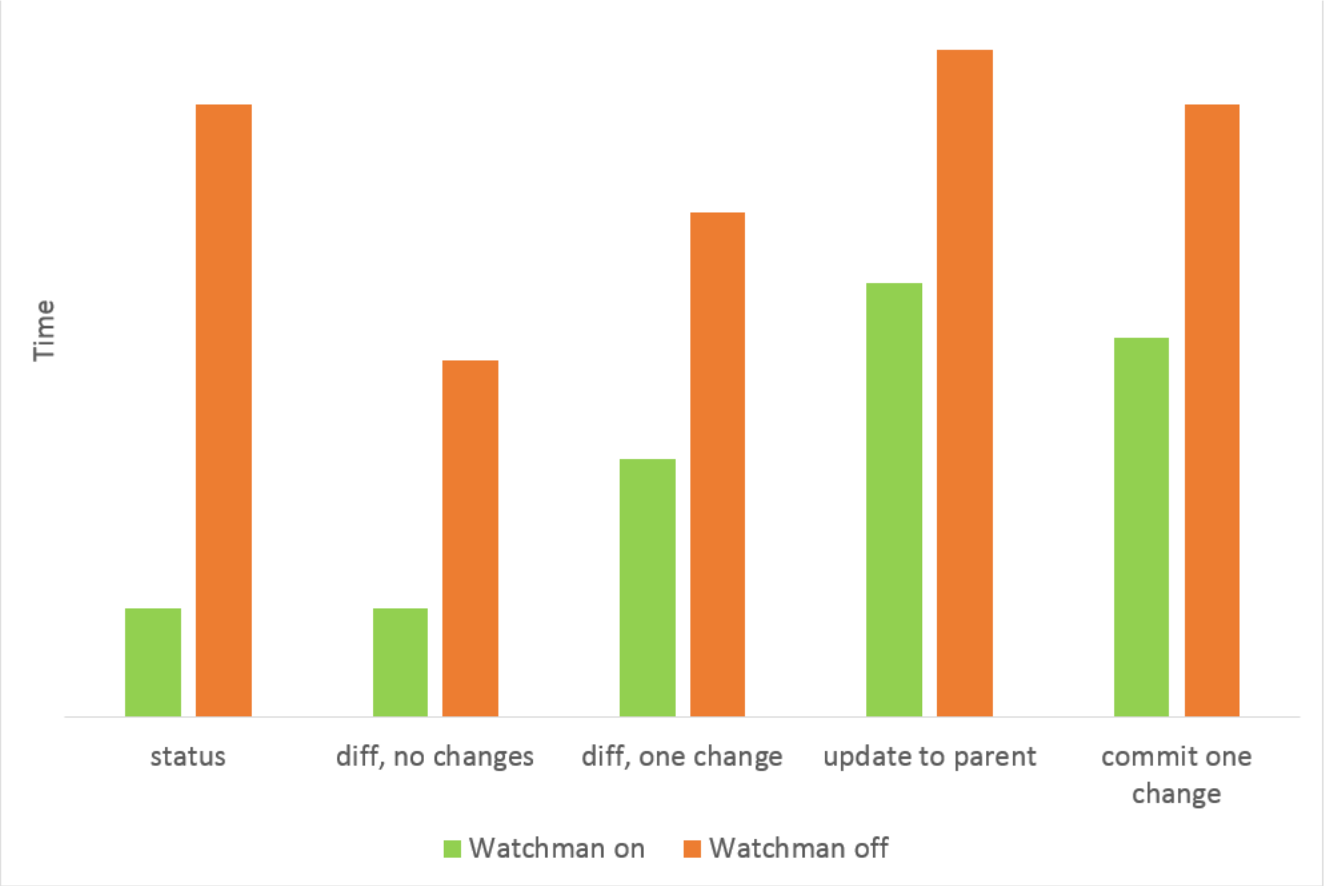
Need more history?
Downloaded automatically on demand.

3b. Version control

- Solution: redesign version control
  - Query build system's file monitor, Watchman, to see which files have changed

3b. Version control

- Solution: redesign version control
  - Query build system's file monitor, Watchman, to see which files have changed → **5x faster "status" command**



3b. Version control

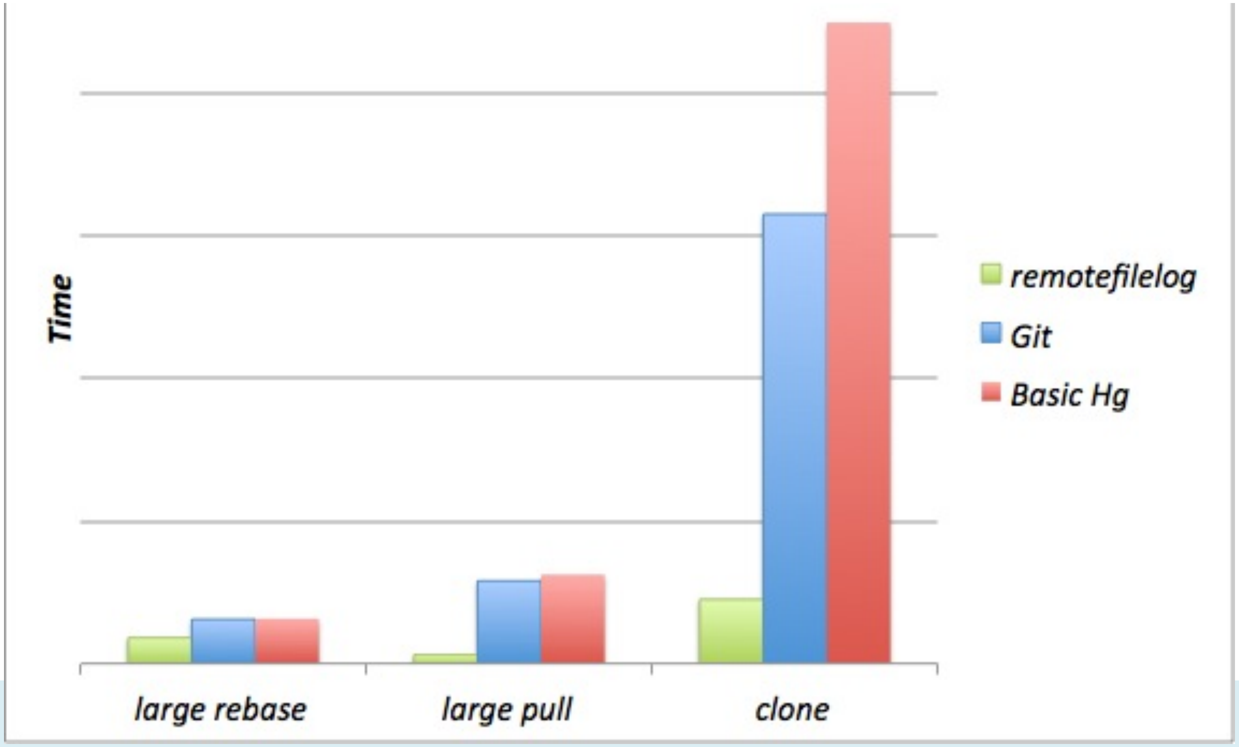
- Solution: redesign version control
  - Sparse checkouts??? (remember, git is a distributed VCS)

3b. Version control

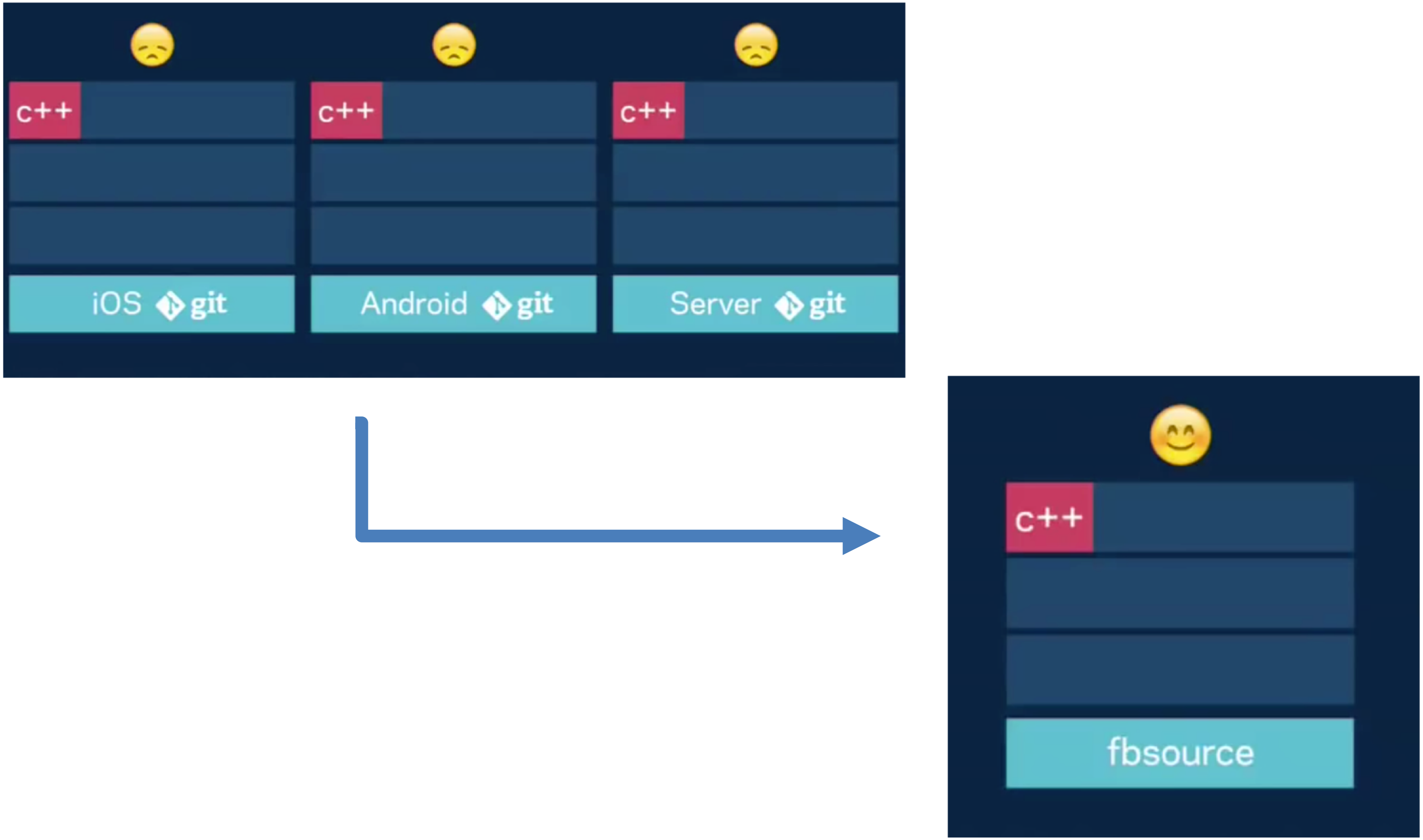
- Solution: redesign version control
  - Sparse checkouts:
  - Change the clone and pull commands to download only the commit metadata, while omitting all file changes (the bulk of the download)
  - When a user performs an operation that needs the contents of files (such as checkout), download the file contents on demand using existing memcache infrastructure

3b. Version control

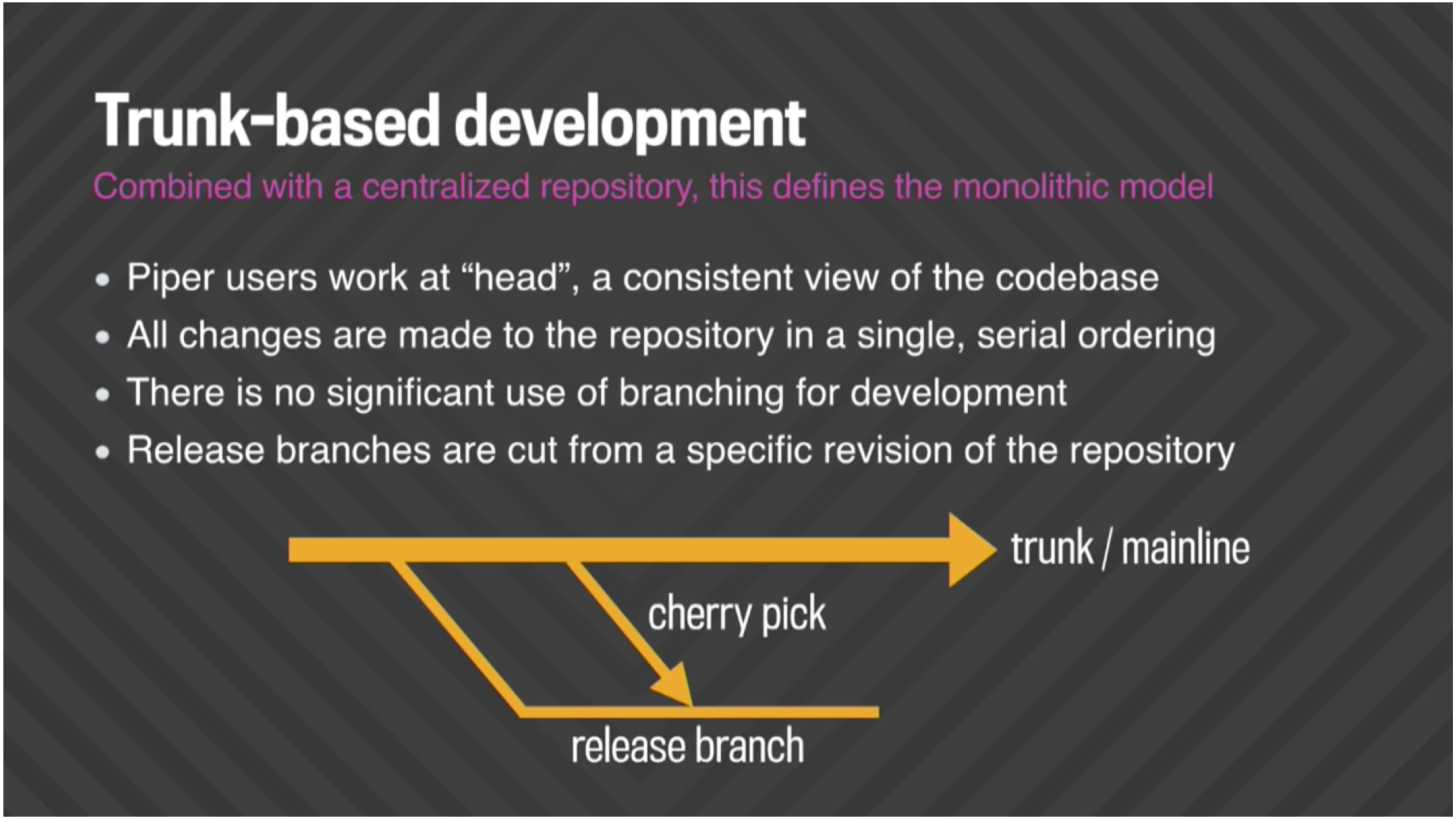
- Solution: redesign version control
  - Sparse checkouts → **10x faster clones and pulls**
  - Change the clone and pull commands to download only the commit metadata, while omitting all file changes (the bulk of the download)
  - When a user performs an operation that needs the contents of files (such as checkout), download the file contents on demand using existing memcache infrastructure



4. Monolithic repository



Monolithic repository – no major use of branches for development



Trunk-based development

Combined with a centralized repository, this defines the monolithic model

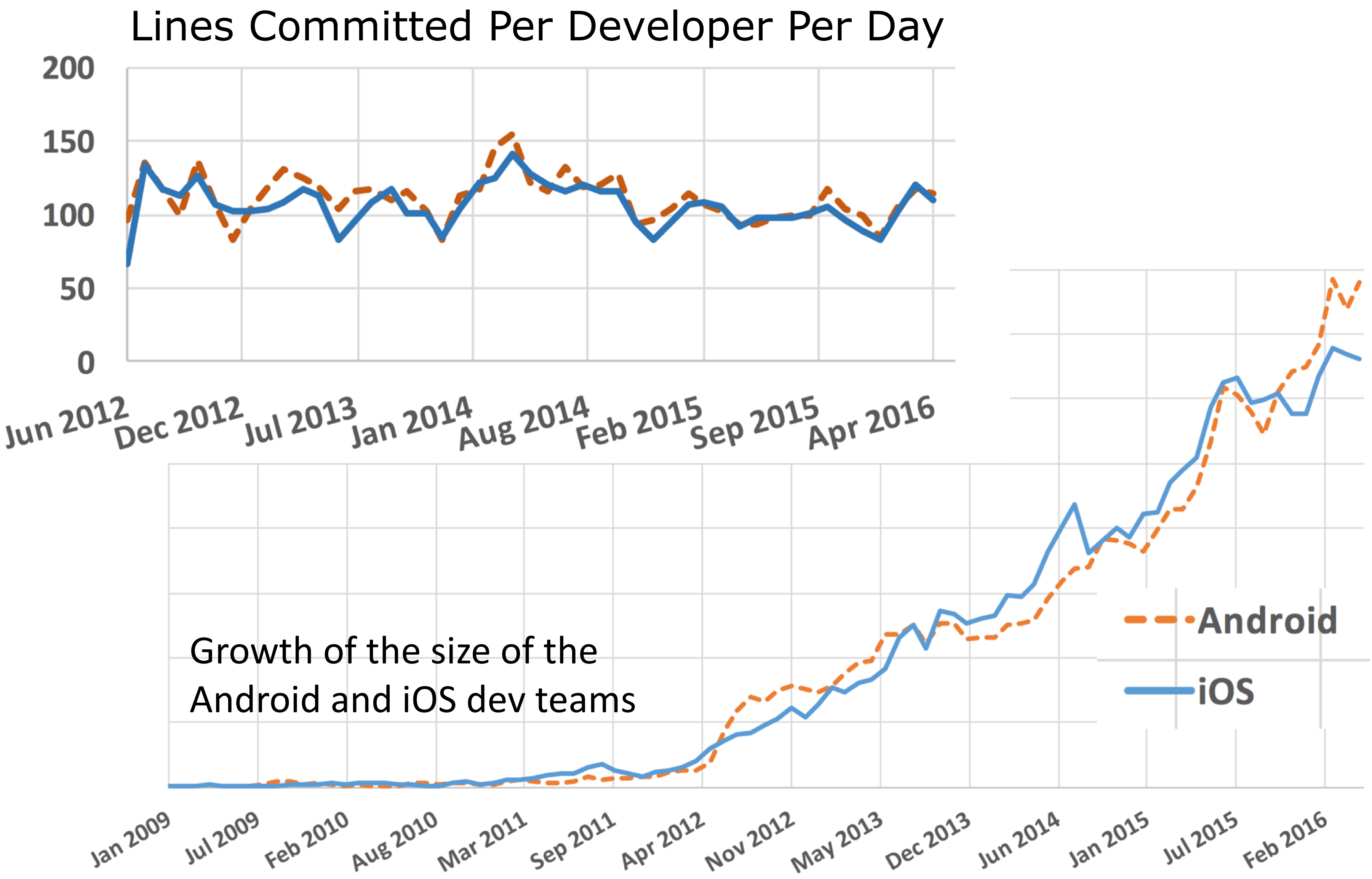
- Piper users work at “head”, a consistent view of the codebase
- All changes are made to the repository in a single, serial ordering
- There is no significant use of branching for development
- Release branches are cut from a specific revision of the repository

trunk / mainline

cherry pick

release branch

Did it work? Yes. Sustained productivity at Facebook



Lines Committed Per Developer Per Day

Growth of the size of the Android and iOS dev teams

MONOREPO VS MANY REPOS

A recent history of code organization

- A single team with a monolithic application in a single repository

…

- Multiple teams with many separate applications in many separate repositories
- Multiple teams with many ~~separate applications~~ **microservices** in many separate repositories
- A single team with many microservices in many repositories

…

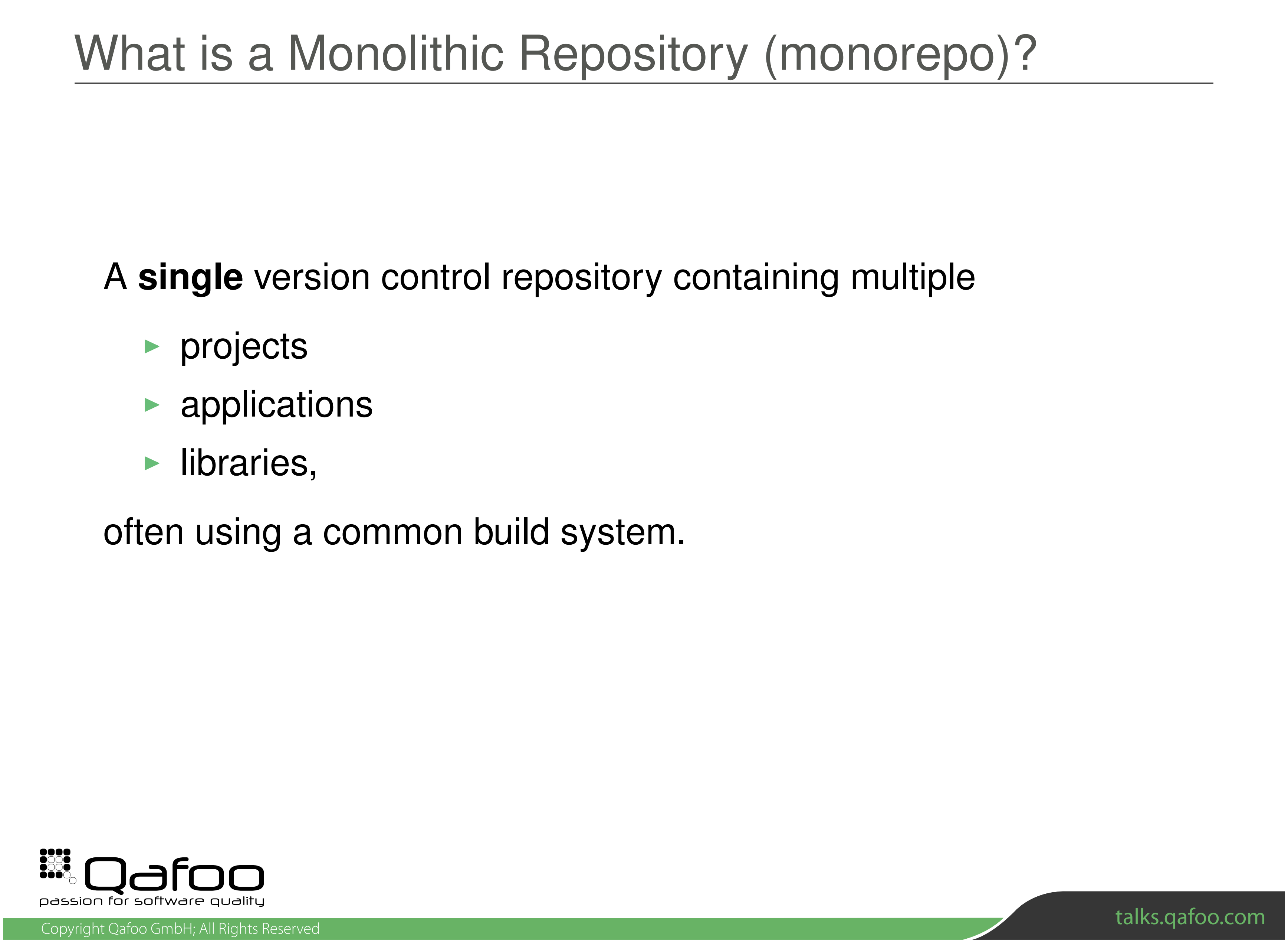
- Many teams with many applications in one big **Monorepo**

What is a monolithic repository (monorepo)?

- A **single** version control repository containing multiple
  - Projects
  - Applications
  - Libraries
- often using a common build system.

History of Version Control

Before Git/Mercurial we all used Subversion and monorepos where widespread.



What is a Monolithic Repository (monorepo)?

A **single** version control repository containing multiple

- projects
- applications
- libraries,

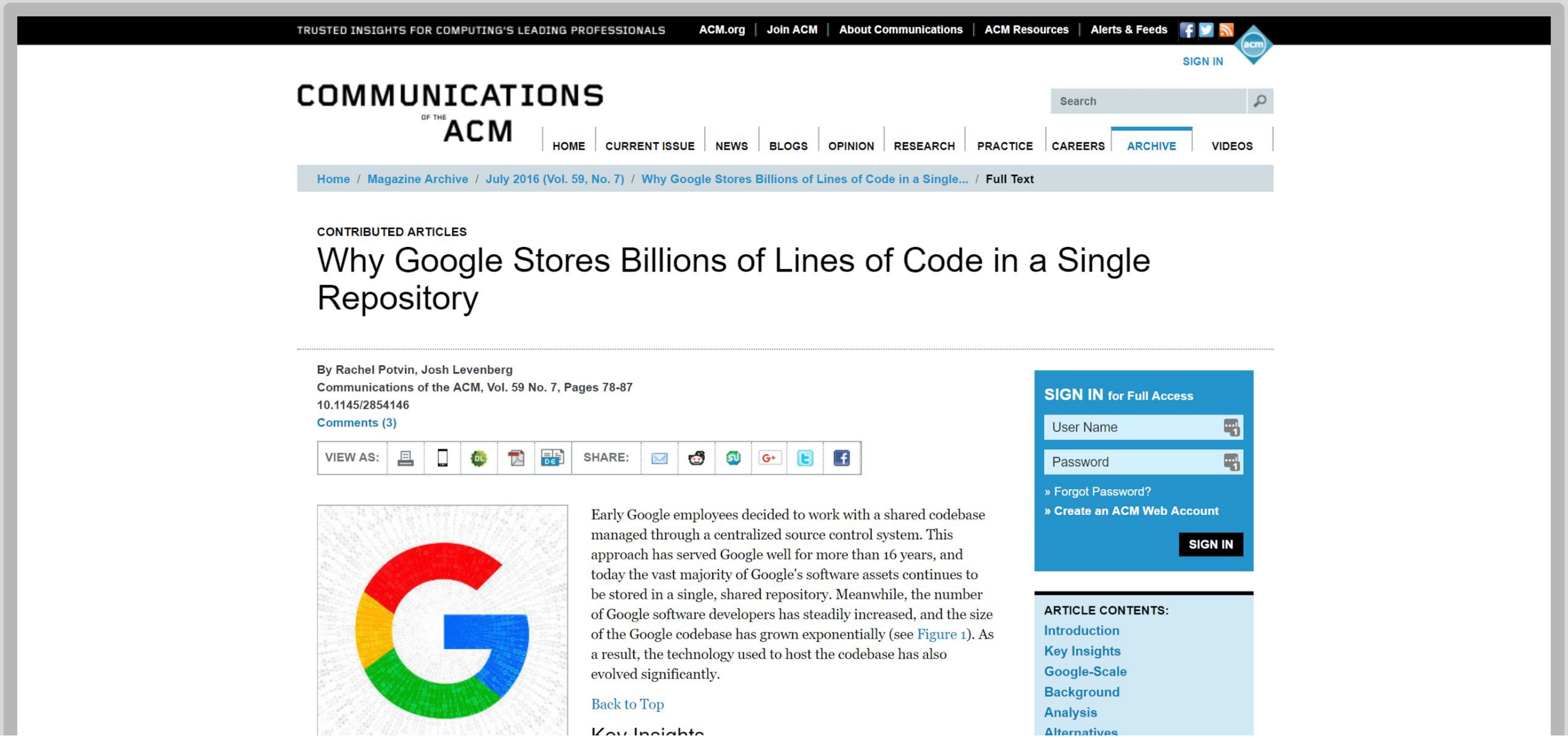
often using a common build system.

talks.qafoo.com

2015 talk by Benjamin Eberlei

Monorepos in industry

Google (computer science version)



TRUSTED INSIGHTS FOR COMPUTING'S LEADING PROFESSIONALS

ACM.org | Join ACM | About Communications | ACM Resources | Alerts & Feeds

SIGN IN

COMMUNICATIONS OF THE ACM

Search

HOME | CURRENT ISSUE | NEWS | BLOGS | OPINION | RESEARCH | PRACTICE | CAREERS | ARCHIVE | VIDEOS

Home / Magazine Archive / July 2016 (Vol. 59, No. 7) / Why Google Stores Billions of Lines of Code in a Single... / Full Text

CONTRIBUTED ARTICLES

Why Google Stores Billions of Lines of Code in a Single Repository

By Rachel Potvin, Josh Levenberg
Communications of the ACM, Vol. 59 No. 7, Pages 78-87
10.1145/2854146
Comments (3)

VIEW AS: SHARE:

Early Google employees decided to work with a shared codebase managed through a centralized source control system. This approach has served Google well for more than 16 years, and today the vast majority of Google's software assets continues to be stored in a single, shared repository. Meanwhile, the number of Google software developers has steadily increased, and the size of the Google codebase has grown exponentially (see Figure 1). As a result, the technology used to host the codebase has also evolved significantly.

Back to Top

Key Insights

SIGN IN for Full Access

User Name

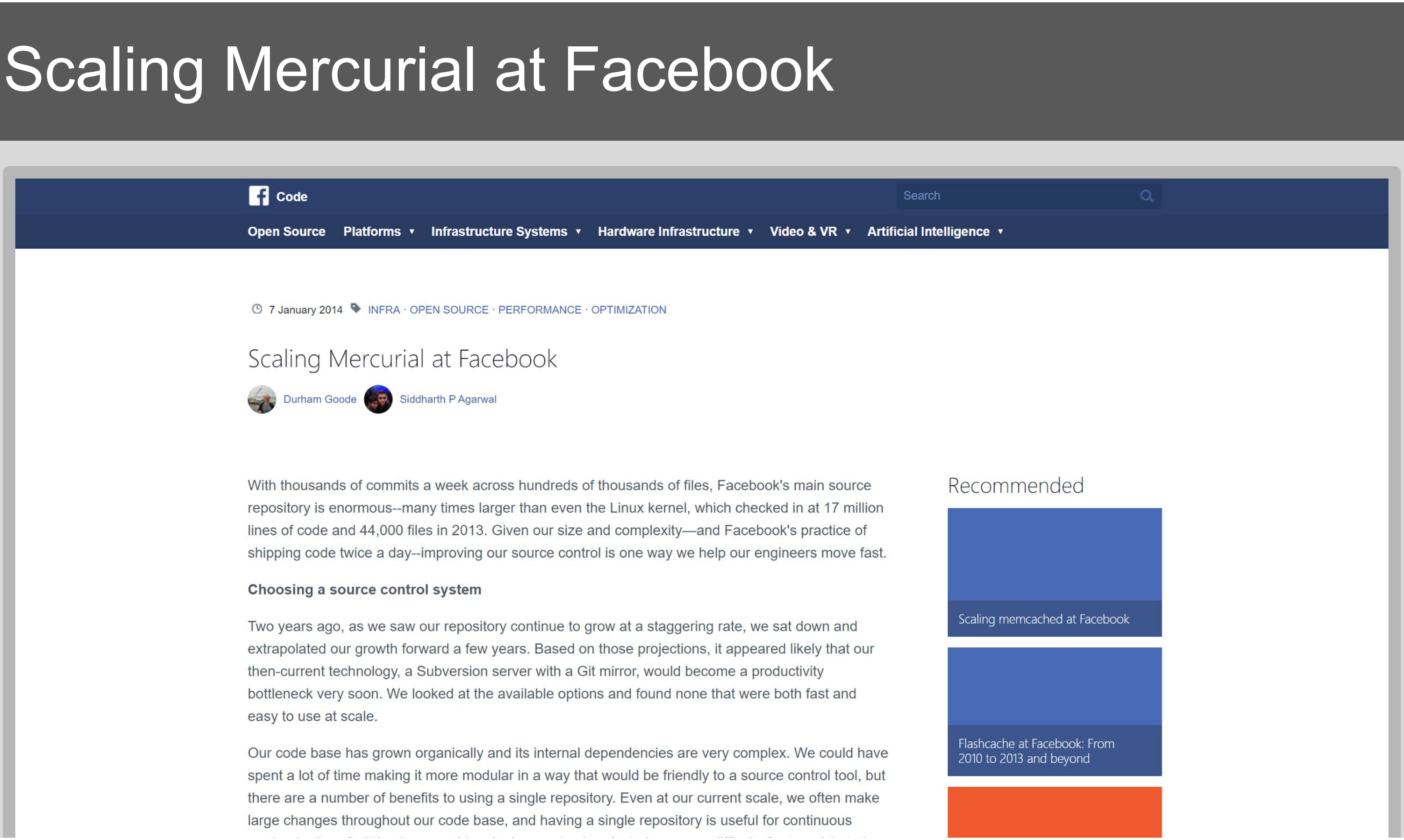
Password

» Forgot Password?
» Create an ACM Web Account

SIGN IN

ARTICLE CONTENTS:
Introduction
Key Insights
Google-Scale
Background
Analysis
Alternatives

Monorepos in industry



Scaling Mercurial at Facebook

7 January 2014 INFRA · OPEN SOURCE · PERFORMANCE · OPTIMIZATION

Scaling Mercurial at Facebook

Durham Goode Siddharth P Agarwal

With thousands of commits a week across hundreds of thousands of files, Facebook's main source repository is enormous--many times larger than even the Linux kernel, which checked in at 17 million lines of code and 44,000 files in 2013. Given our size and complexity—and Facebook's practice of shipping code twice a day--improving our source control is one way we help our engineers move fast.

Choosing a source control system

Two years ago, as we saw our repository continue to grow at a staggering rate, we sat down and extrapolated our growth forward a few years. Based on those projections, it appeared likely that our then-current technology, a Subversion server with a Git mirror, would become a productivity bottleneck very soon. We looked at the available options and found none that were both fast and easy to use at scale.

Our code base has grown organically and its internal dependencies are very complex. We could have spent a lot of time making it more modular in a way that would be friendly to a source control tool, but there are a number of benefits to using a single repository. Even at our current scale, we often make large changes throughout our code base, and having a single repository is useful for continuous

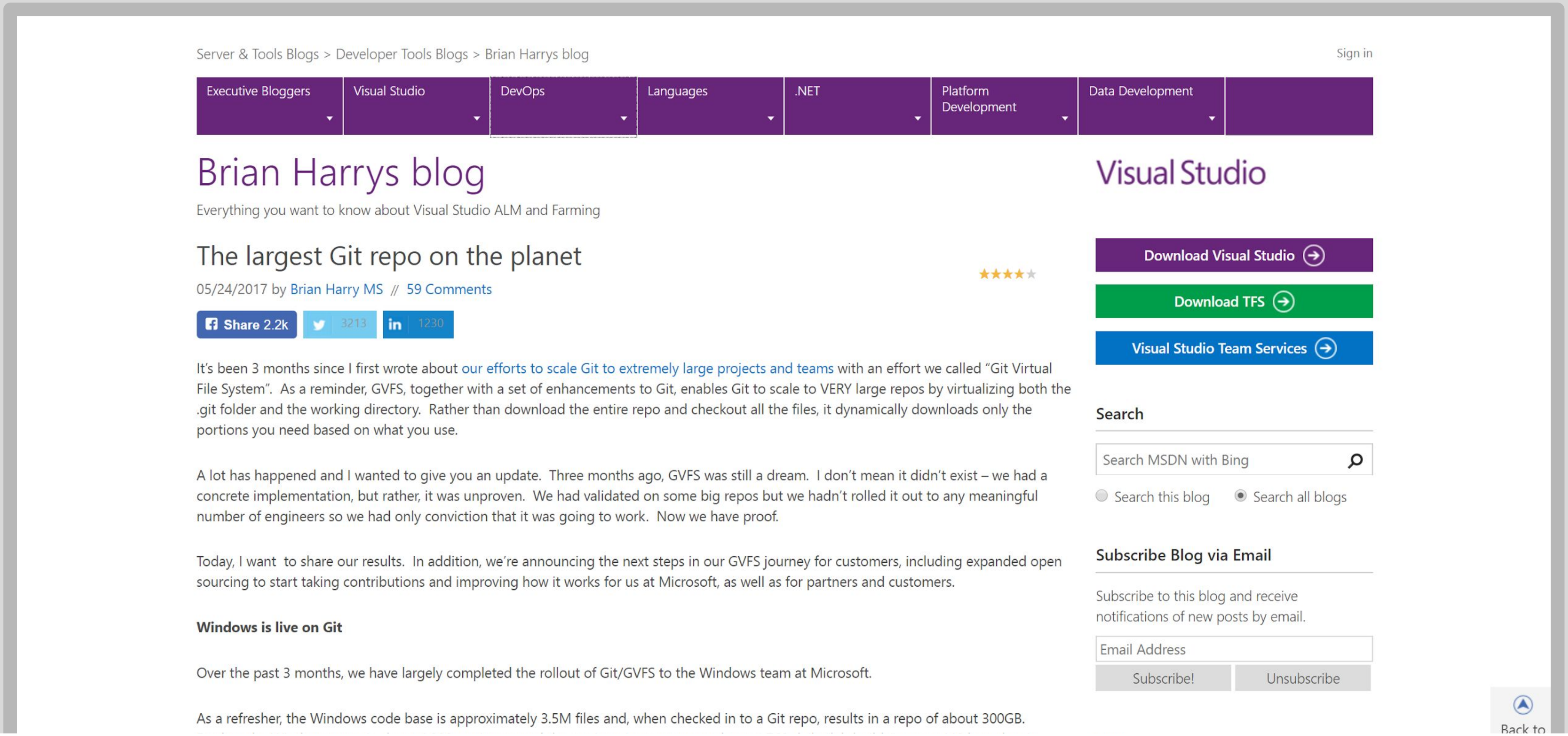
Recommended

Scaling memcached at Facebook

Flashcache at Facebook: From 2010 to 2013 and beyond

Monorepos in industry

Microsoft claim the largest git repo on the planet



Brian Harrys blog

Everything you want to know about Visual Studio ALM and Farming

The largest Git repo on the planet

05/24/2017 by Brian Harry MS // 59 Comments

★★★★☆

It's been 3 months since I first wrote about our efforts to scale Git to extremely large projects and teams with an effort we called "Git Virtual File System". As a reminder, GVFS, together with a set of enhancements to Git, enables Git to scale to VERY large repos by virtualizing both the .git folder and the working directory. Rather than download the entire repo and checkout all the files, it dynamically downloads only the portions you need based on what you use.

A lot has happened and I wanted to give you an update. Three months ago, GVFS was still a dream. I don't mean it didn't exist – we had a concrete implementation, but rather, it was unproven. We had validated on some big repos but we hadn't rolled it out to any meaningful number of engineers so we had only conviction that it was going to work. Now we have proof.

Today, I want to share our results. In addition, we're announcing the next steps in our GVFS journey for customers, including expanded open sourcing to start taking contributions and improving how it works for us at Microsoft, as well as for partners and customers.

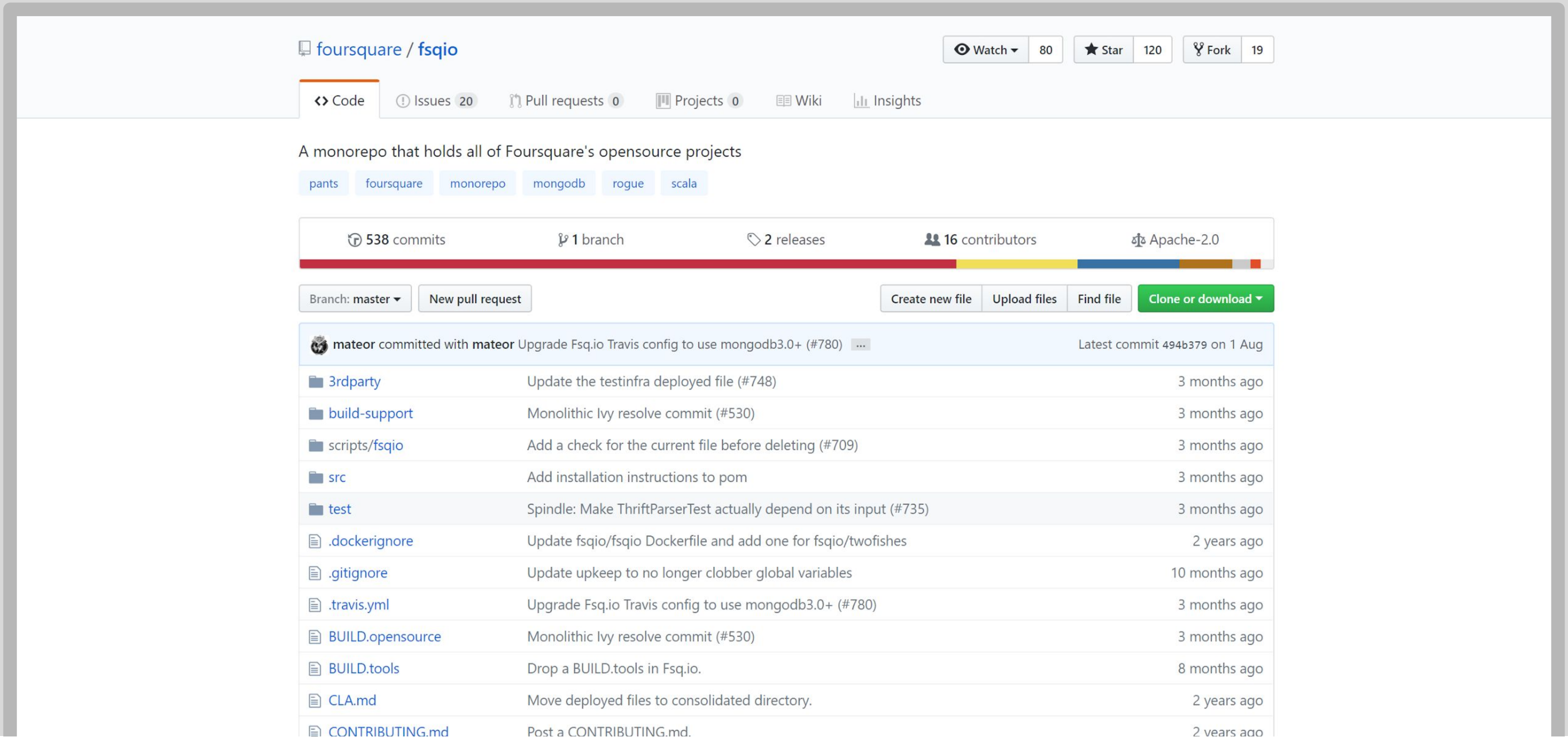
Windows is live on Git

Over the past 3 months, we have largely completed the rollout of Git/GVFS to the Windows team at Microsoft.

As a refresher, the Windows code base is approximately 3.5M files and, when checked in to a Git repo, results in a repo of about 300GB.

Monorepos in open-source

foresquare public monorepo



foursquare / fsqio

A monorepo that holds all of Foursquare's opensource projects

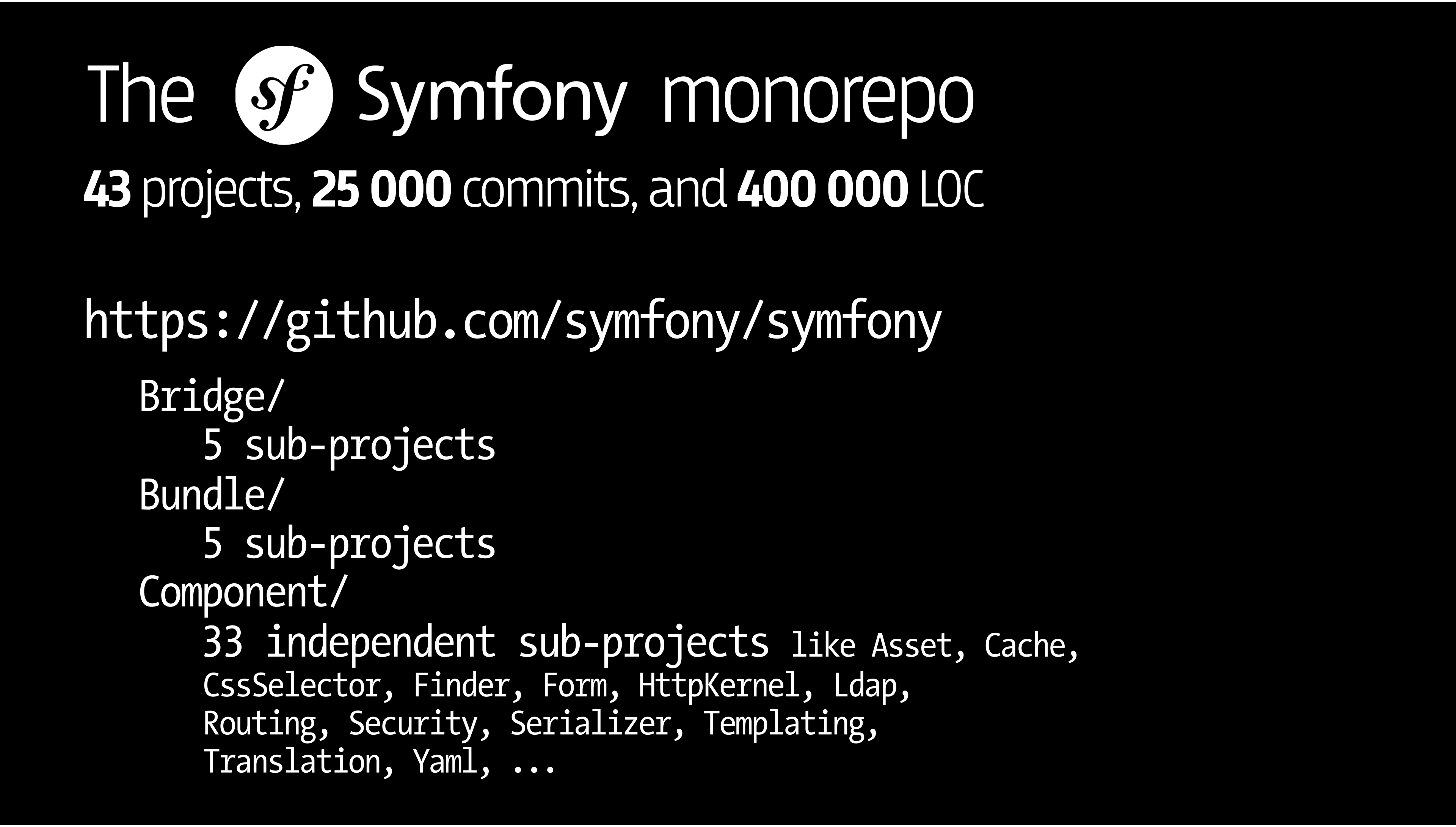
538 commits · 1 branch · 2 releases · 16 contributors · Apache-2.0

mateor committed with mateor Upgrade Fsq.io Travis config to use mongodb3.0+ (#780) … Latest commit 494b379 on 1 Aug

3rdparty	Update the testinfra deployed file (#748)	3 months ago
build-support	Monolithic Ivy resolve commit (#530)	3 months ago
scripts/fsqio	Add a check for the current file before deleting (#709)	3 months ago
src	Add installation instructions to pom	3 months ago
test	Spindle: Make ThriftParserTest actually depend on its input (#735)	3 months ago
.dockerignore	Update fsqio/fsqio Dockerfile and add one for fsqio/twofishes	2 years ago
.gitignore	Update upkeep to no longer clobber global variables	10 months ago
.travis.yml	Upgrade Fsq.io Travis config to use mongodb3.0+ (#780)	3 months ago
BUILD.opensource	Monolithic Ivy resolve commit (#530)	3 months ago
BUILD.tools	Drop a BUILD.tools in Fsq.io.	8 months ago
CLA.md	Move deployed files to consolidated directory.	2 years ago
CONTRIBUTING.md	Post a CONTRIBUTING.md.	2 years ago

2016 talk by FABIEN POTENCIER

Monorepos in open-source



The Symfony monorepo

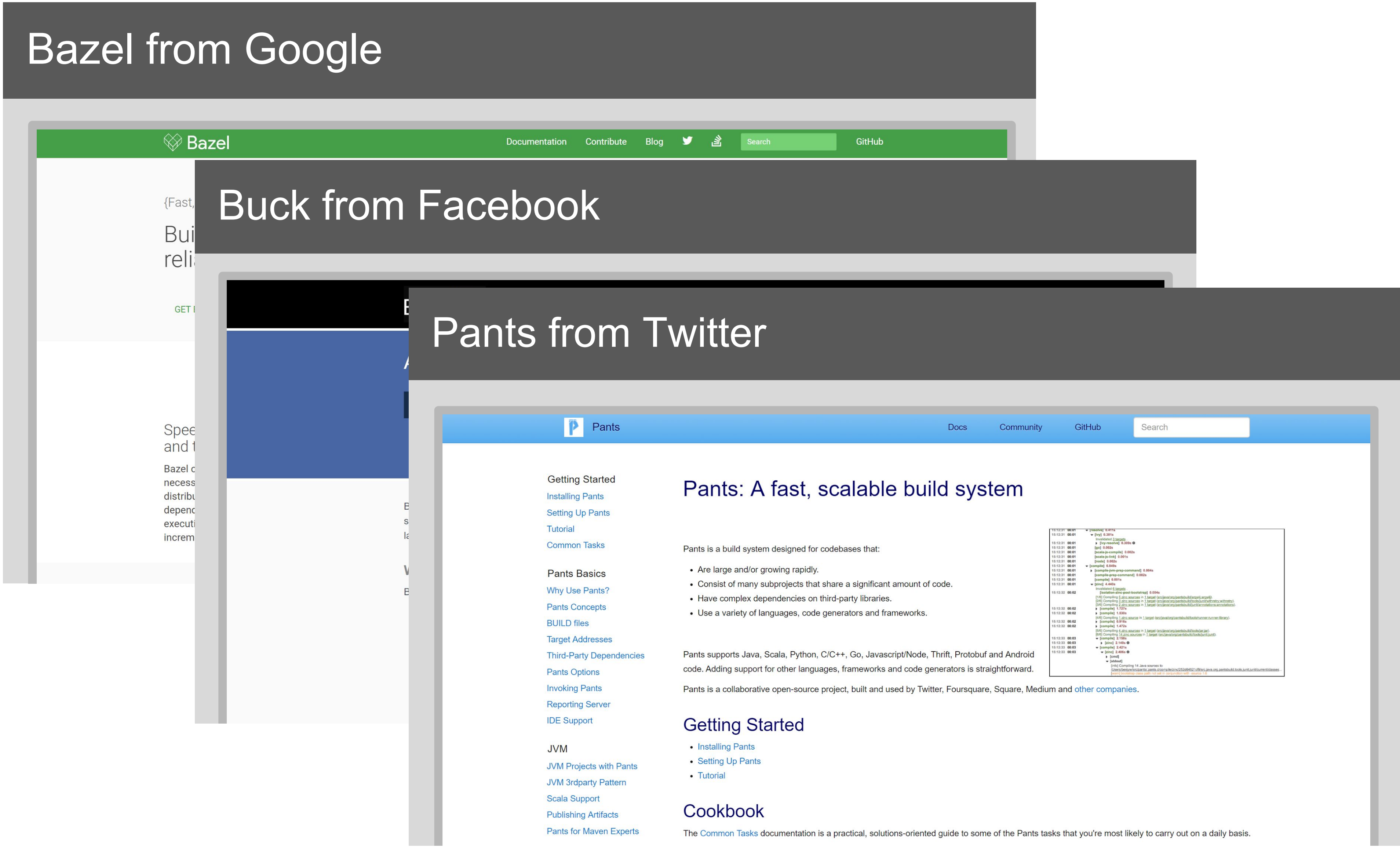
43 projects, **25 000** commits, and **400 000** LOC

https://github.com/symfony/symfony

```
Bridge/
    5 sub-projects
Bundle/
    5 sub-projects
Component/
    33 independent sub-projects like Asset, Cache,
    CssSelector, Finder, Form, HttpKernel, Ldap,
    Routing, Security, Serializer, Templating,
    Translation, Yaml, ...
```

2016 talk by FABIEN POTENCIER

Common build system



Bazel from Google

Bazel Documentation Contribute Blog Search GitHub

{Fast,

Bui
reli

GET

Spee
and

Bazel c
necess
distribu
depend
executi
increm

Buck from Facebook

B

A

B
s
la

W

B

Pants from Twitter

Pants Docs Community GitHub Search

Getting Started
Installing Pants
Setting Up Pants
Tutorial
Common Tasks

Pants Basics
Why Use Pants?
Pants Concepts
BUILD files
Target Addresses
Third-Party Dependencies
Pants Options
Invoking Pants
Reporting Server
IDE Support

JVM
JVM Projects with Pants
JVM 3rdparty Pattern
Scala Support
Publishing Artifacts
Pants for Maven Experts

Pants: A fast, scalable build system

Pants is a build system designed for codebases that:

- Are large and/or growing rapidly.
- Consist of many subprojects that share a significant amount of code.
- Have complex dependencies on third-party libraries.
- Use a variety of languages, code generators and frameworks.

Pants supports Java, Scala, Python, C/C++, Go, Javascript/Node, Thrift, Protobuf and Android code. Adding support for other languages, frameworks and code generators is straightforward.

Pants is a collaborative open-source project, built and used by Twitter, Foursquare, Square, Medium and other companies.

Getting Started

- Installing Pants
- Setting Up Pants
- Tutorial

Cookbook

The Common Tasks documentation is a practical, solutions-oriented guide to some of the Pants tasks that you're most likely to carry out on a daily basis.

Some advantages of monorepos

High Discoverability For Developers

- Developers can read and explore the whole codebase
- `grep`, IDEs and other tools can search the whole codebase
- IDEs can offer auto-completion for the whole codebase
- Code Browsers can links between all artifacts in the codebase

Code-Reuse is cheap

Almost zero cost in introducing a new library

- Extract library code into a new directory/component
- Use library in other components
- Profit!

Refactorings in one commit

Allow large scale refactorings with one single, atomic, history-preserving commit

- Extract Library/Component
- Rename Functions/Methods/Components
- Housekeeping (phpcs-fixer, Namespacing, ...)

passion for software quality

Another refactoring example

- Make large backward incompatible changes easily... especially if they span different parts of the project

- For example, old APIs can be removed with confidence
  - Change an API endpoint code **and** all its usages in **all** projects in **one** pull request

Some more advantages

- Easy continuous integration and code review for changes spanning several projects
- (Internal) dependency management is a non-issue
- Less context switching for developers
- Code more reusable in other contexts
- Access control is easy

Some downsides

- Require collective responsibility for team and developers
- Require trunk-based development
  - Feature toggles are technical debt (recall financial services example)
- Force you to have only one version of everything
- Scalability requirements for the repository
- Can be hard to deal with updates around things like security issues
- Build and test bloat without very smart build system
- Slow VCS without very smart system
- Permissions?

Summary

- Configuration management
  - Treat infrastructure as code
  - Git is powerful
- Release management: versioning, branching, …
- Software development at scale requires a lot of infrastructure
  - Version control, build managers, testing, continuous integration, deployment, …
- It’s hard to scale development
  - Move towards heavy automation (DevOps)
- Continuous deployment increasingly common
- Opportunities from quick release, testing in production, quick rollback