# Mining GitHub for fun and profit

Georgios Gousios // **@gousiosg**
TU Delft

# api.github.com

## Entities

- static view
- interlinked
- current state

## Events

- dynamic view
- generated by user actions
- affect current entity state
- can be browsing roots

Events

CreateEvent

PushEvent

WatchEvent

.

.

.

ForkEvent

Events

CreateEvent

PushEvent

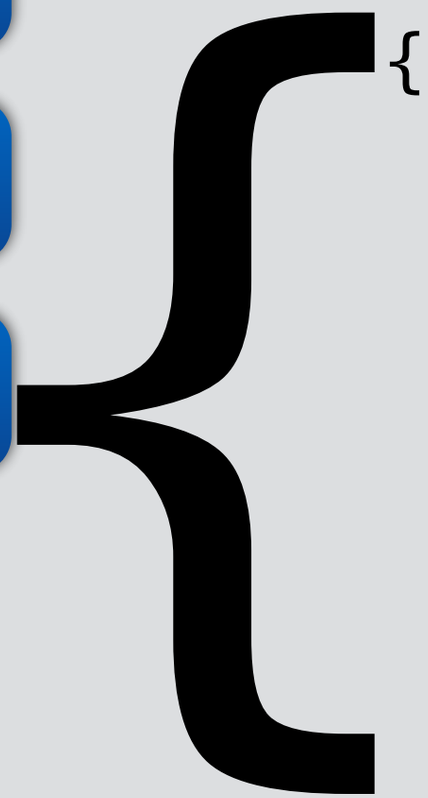
WatchEvent

.

.

.

ForkEvent




```
{
  "type": "WatchEvent",
  "payload": {...},
  "public": true,
  "repo": {...},
  "created_at": "2012-05-28T12:42
  "id": "1556481024",
  "actor": {"login": "Sarukhan"}
}
```

Entities

users

repositories

organizations

•

•

•

issues

/users/:user

/user/repos

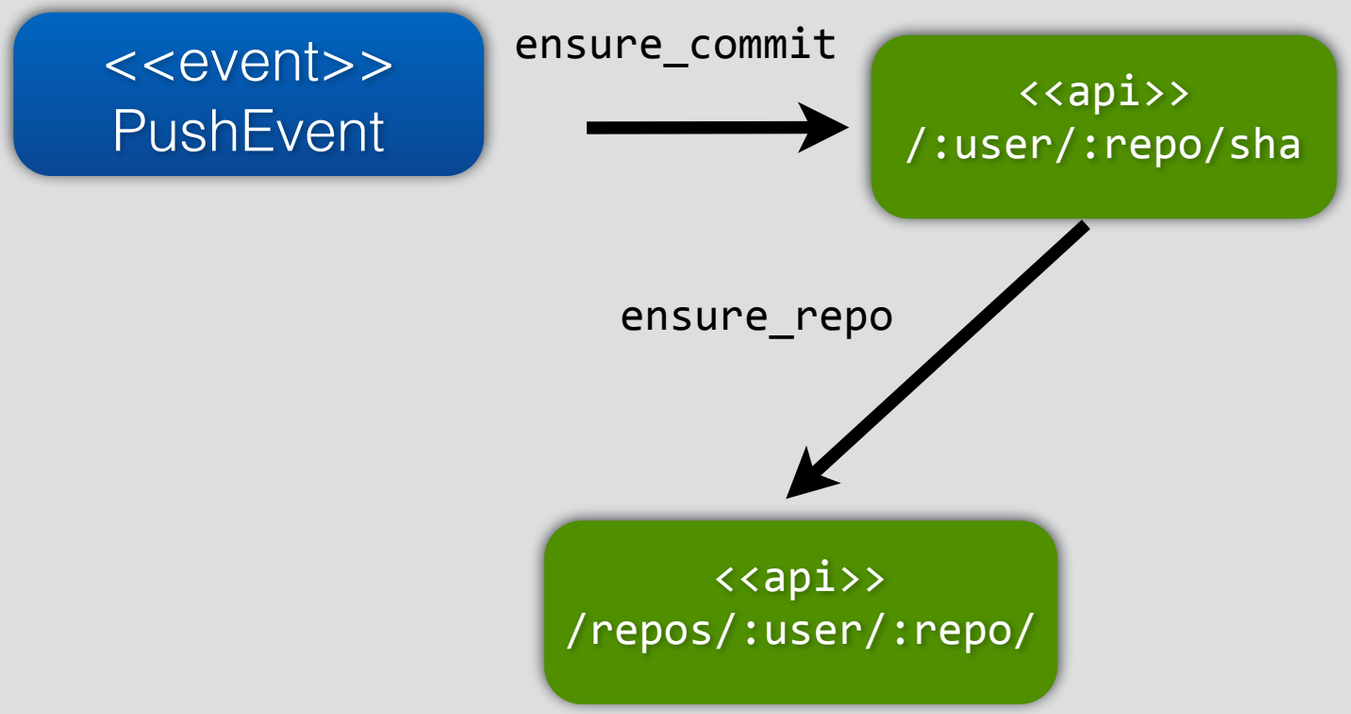
/orgs/:org

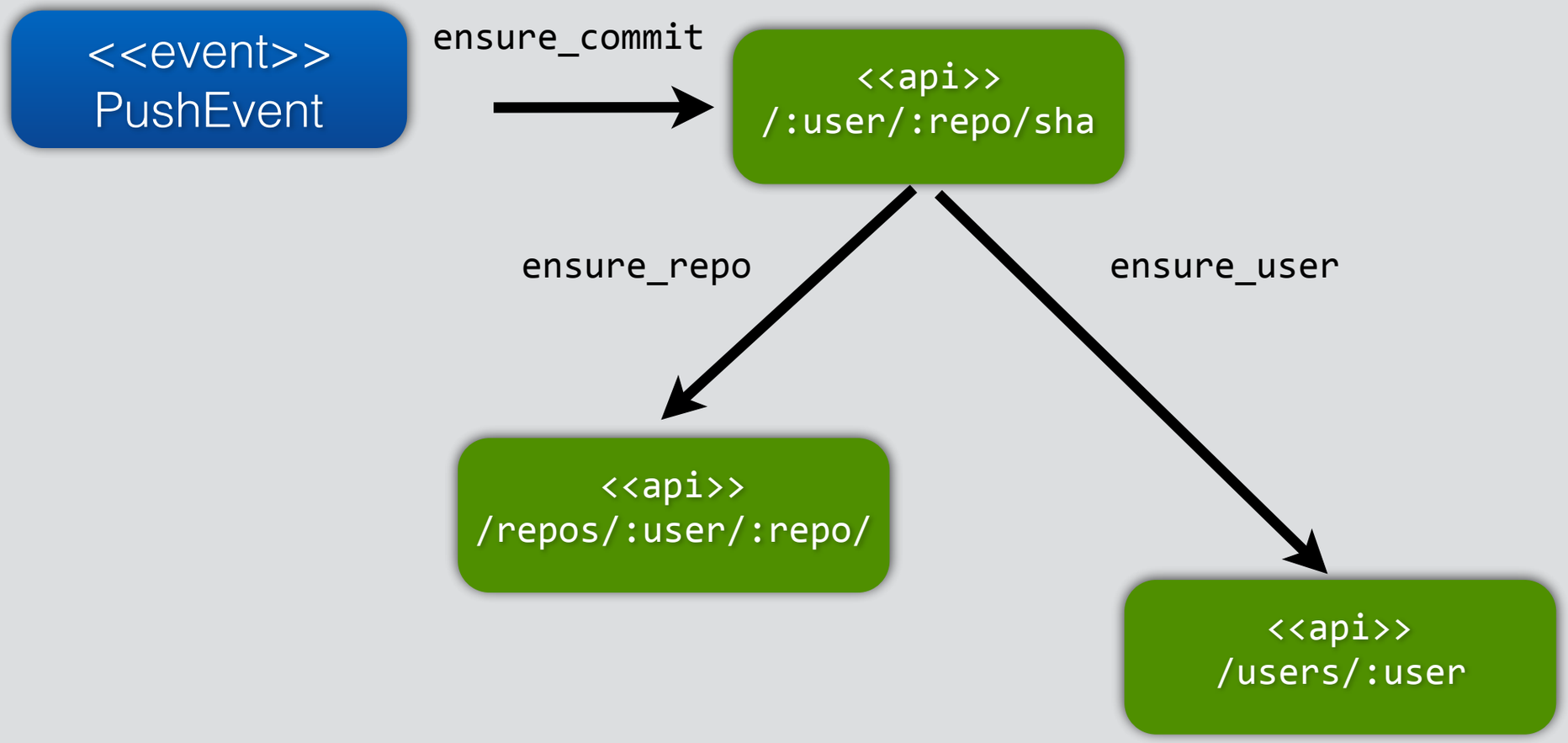
/repos/:user/:repo/issues

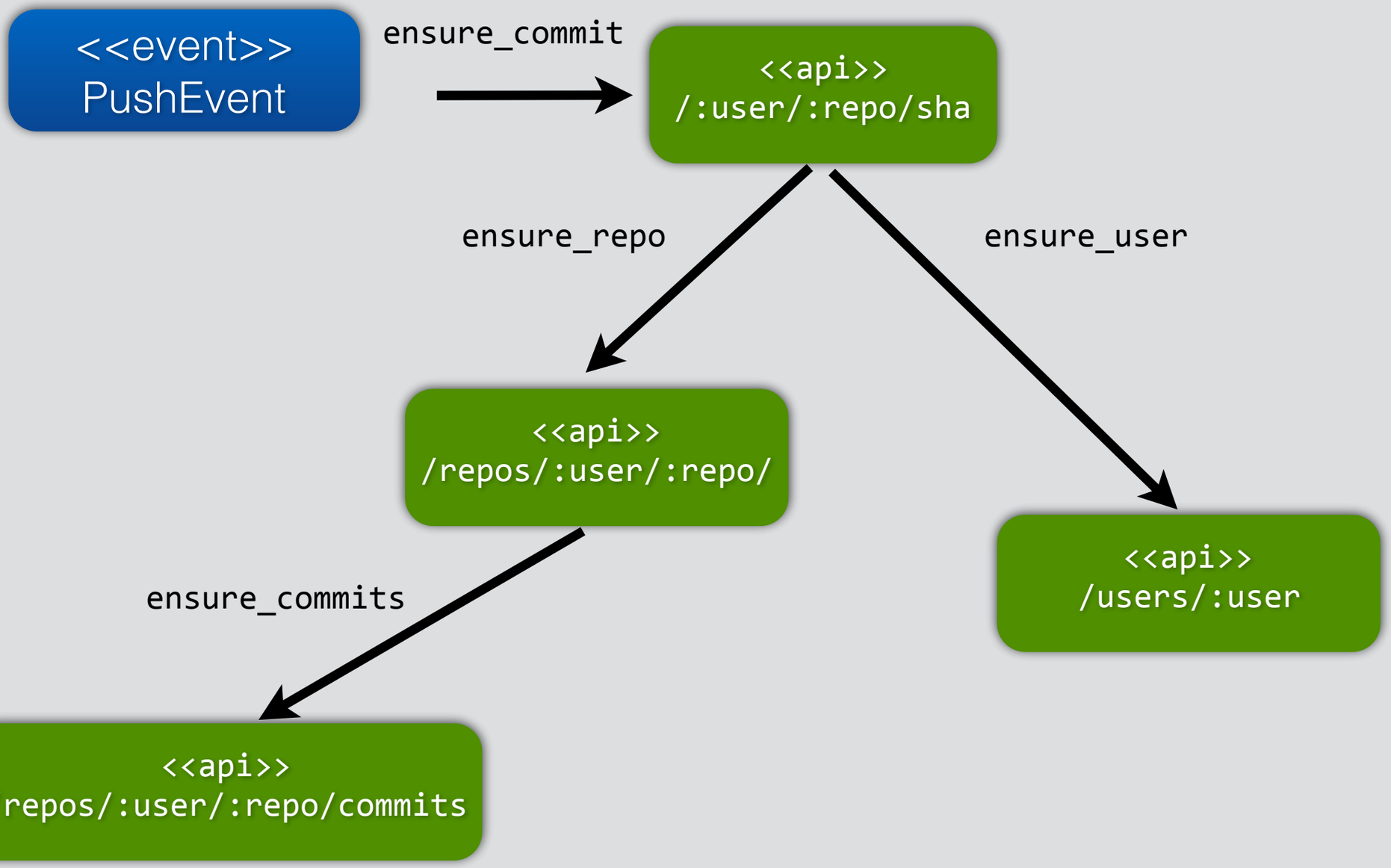

```
{
  "type": "User",
  "public_gists": 10,
  "login": "gousiosg",
  "followers": 64,
  "name": "Georgios Gousios",
  "public_repos": 20,
  "created_at": ...,
  "id": 386172,
  "following": 16,
}
```

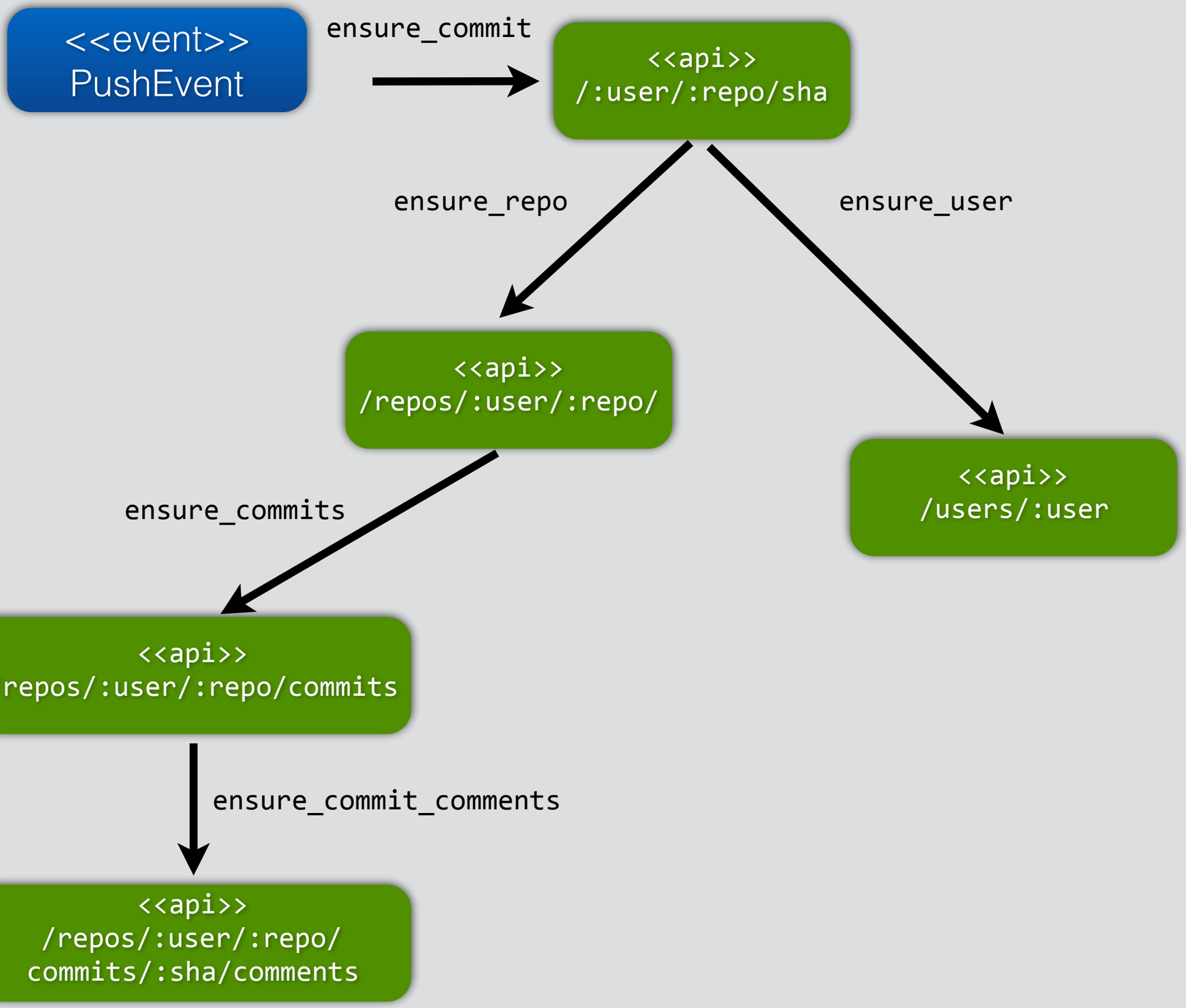
<<event>>
PushEvent

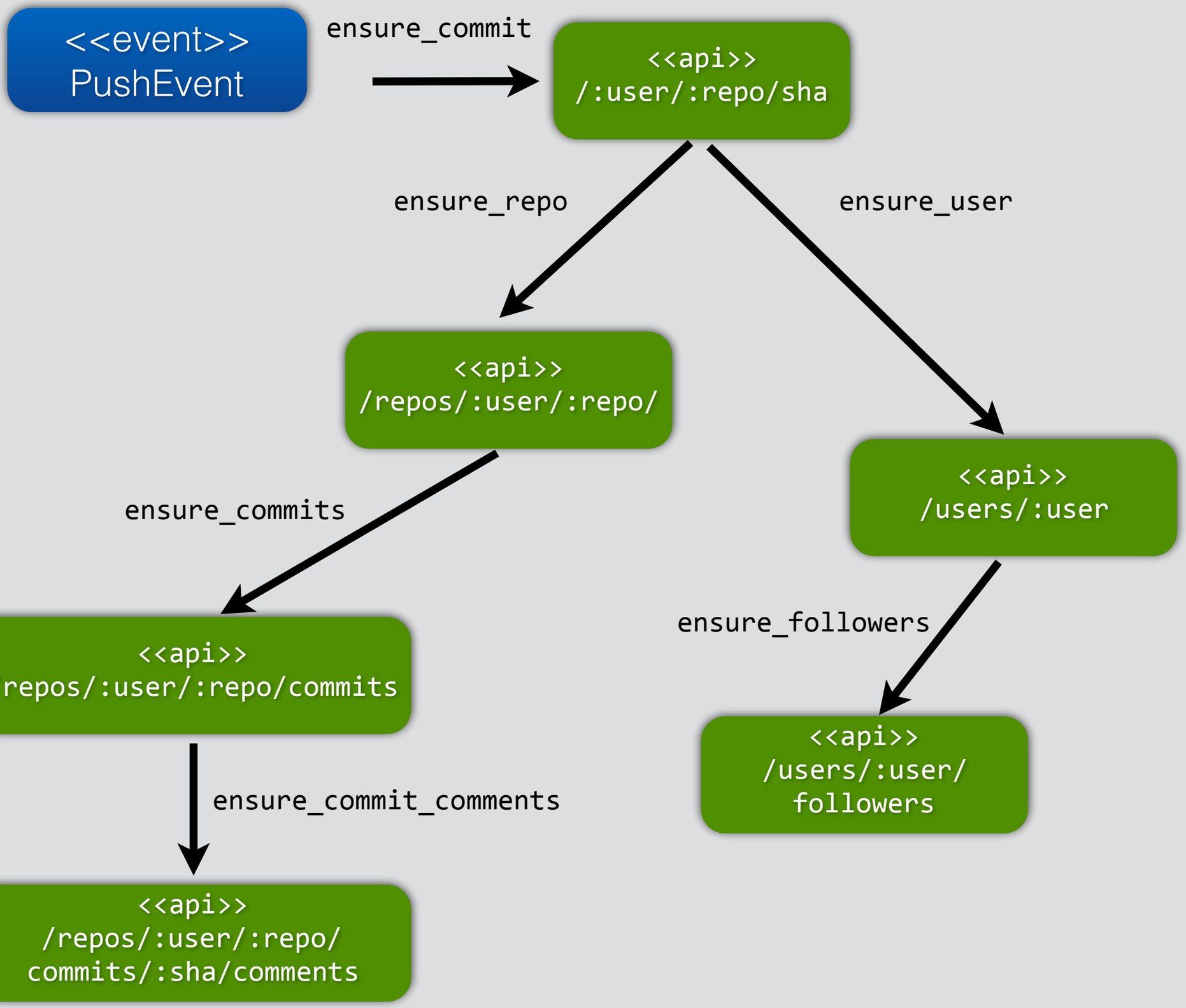

<<event>>
PushEvent
ensure_commit
<<api>>
/:user/:repo/sha

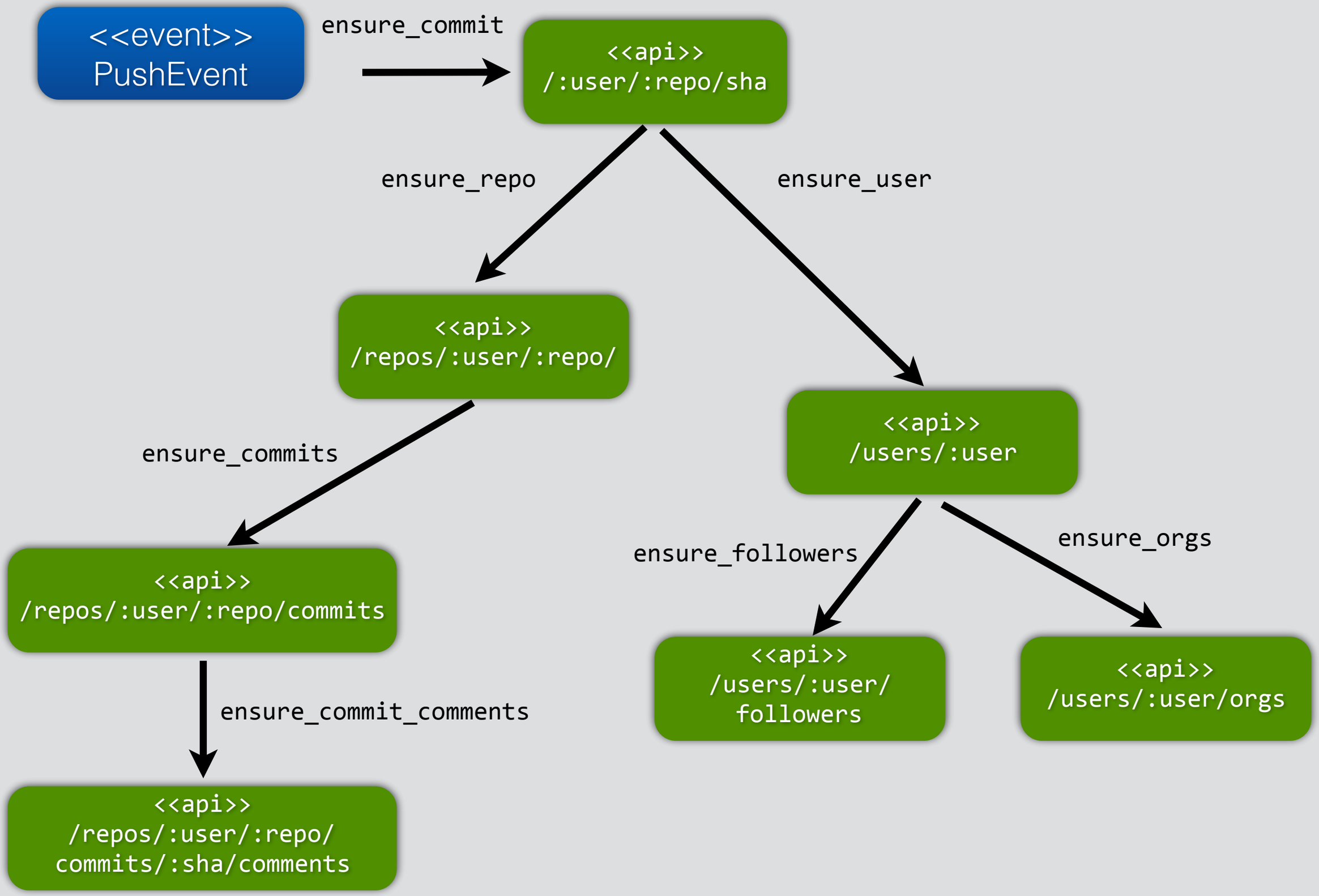

<<event>>
PushEvent
ensure_commit
<<api>>
/:user/:repo/sha
ensure_repo
<<api>>
/repos/:user/:repo/

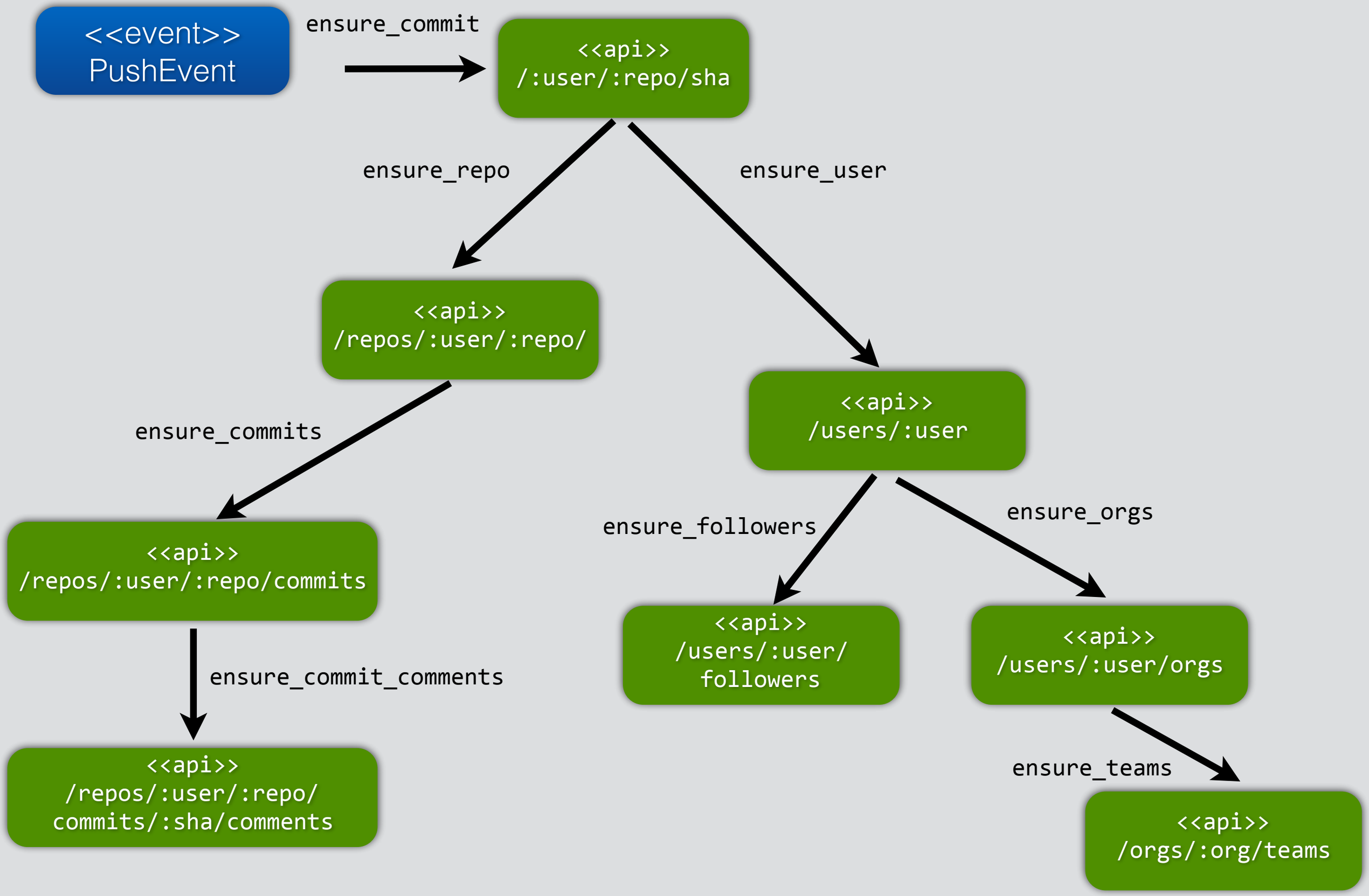

<<event>>
PushEvent
ensure_commit
<<api>>
/:user/:repo/sha
ensure_repo
ensure_user
<<api>>
/repos/:user/:repo/
<<api>>
/users/:user

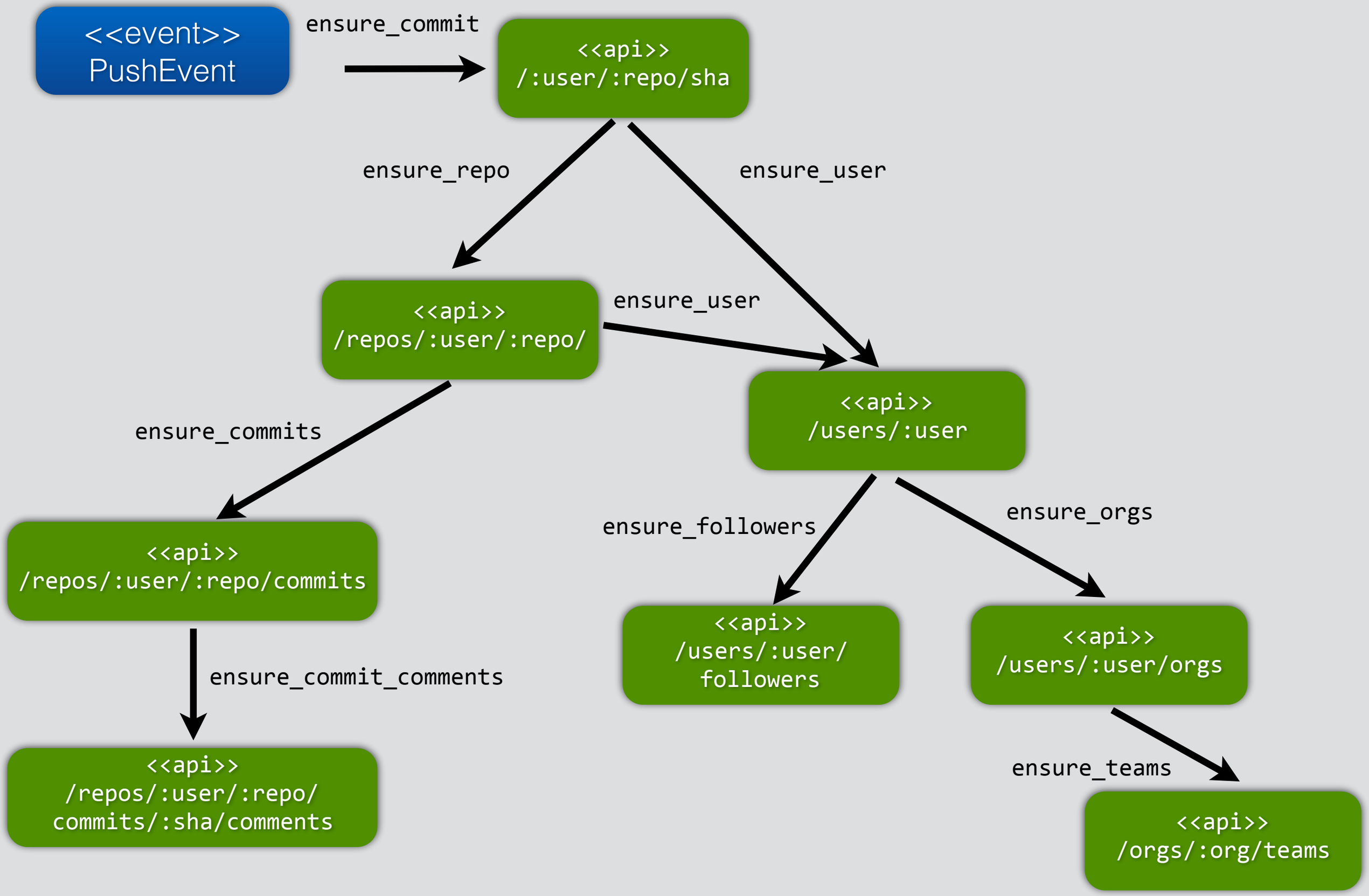

<<event>>
PushEvent
ensure_commit
<<api>>
/:user/:repo/sha
ensure_repo
ensure_user
<<api>>
/repos/:user/:repo/
ensure_commits
<<api>>
/users/:user
<<api>>
/repos/:user/:repo/commits

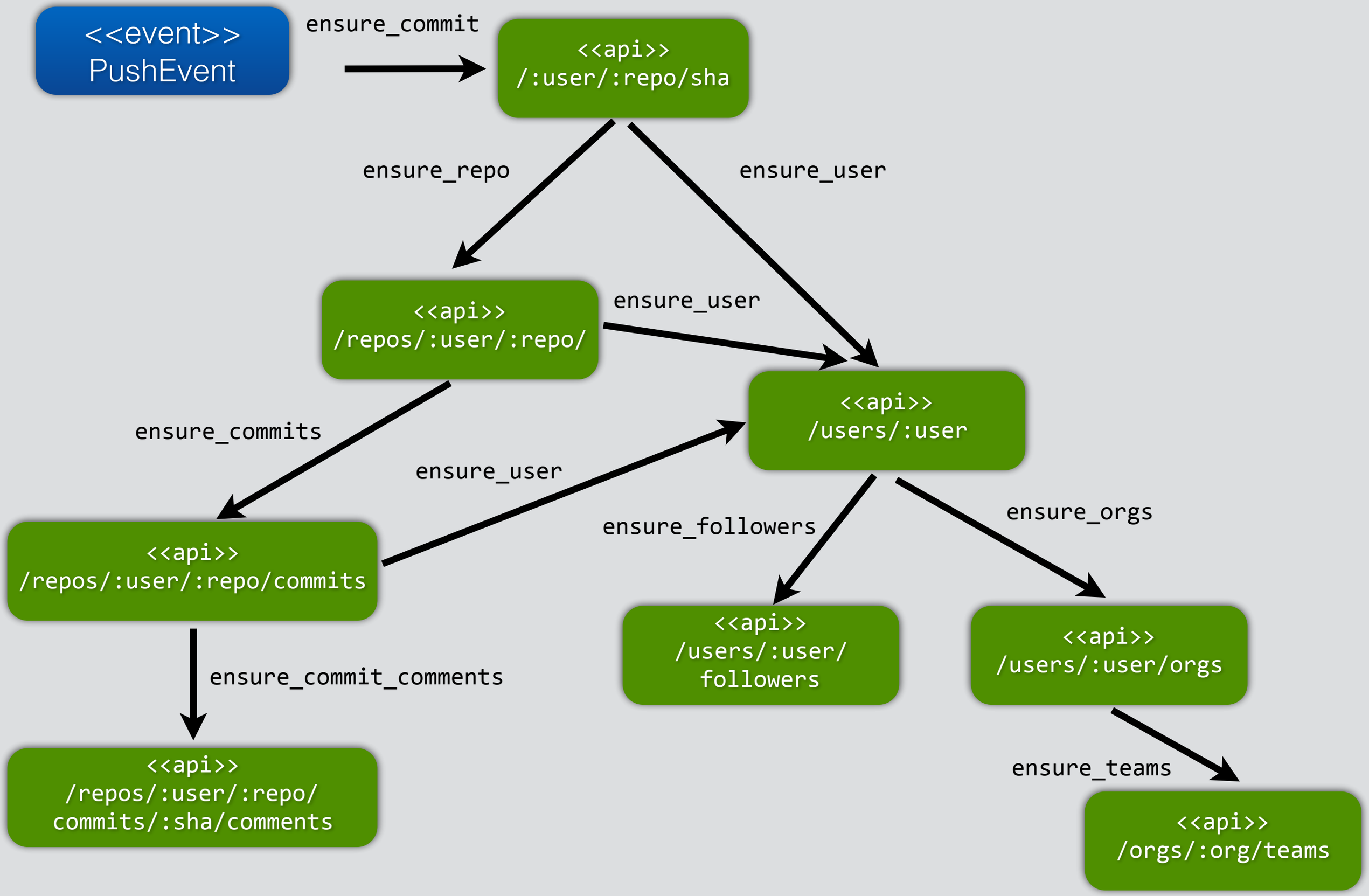

<<event>>
PushEvent
ensure_commit
<<api>>
/:user/:repo/sha
ensure_repo
ensure_user
<<api>>
/repos/:user/:repo/
ensure_commits
<<api>>
/users/:user
<<api>>
/repos/:user/:repo/commits
ensure_commit_comments
<<api>>
/repos/:user/:repo/
commits/:sha/comments

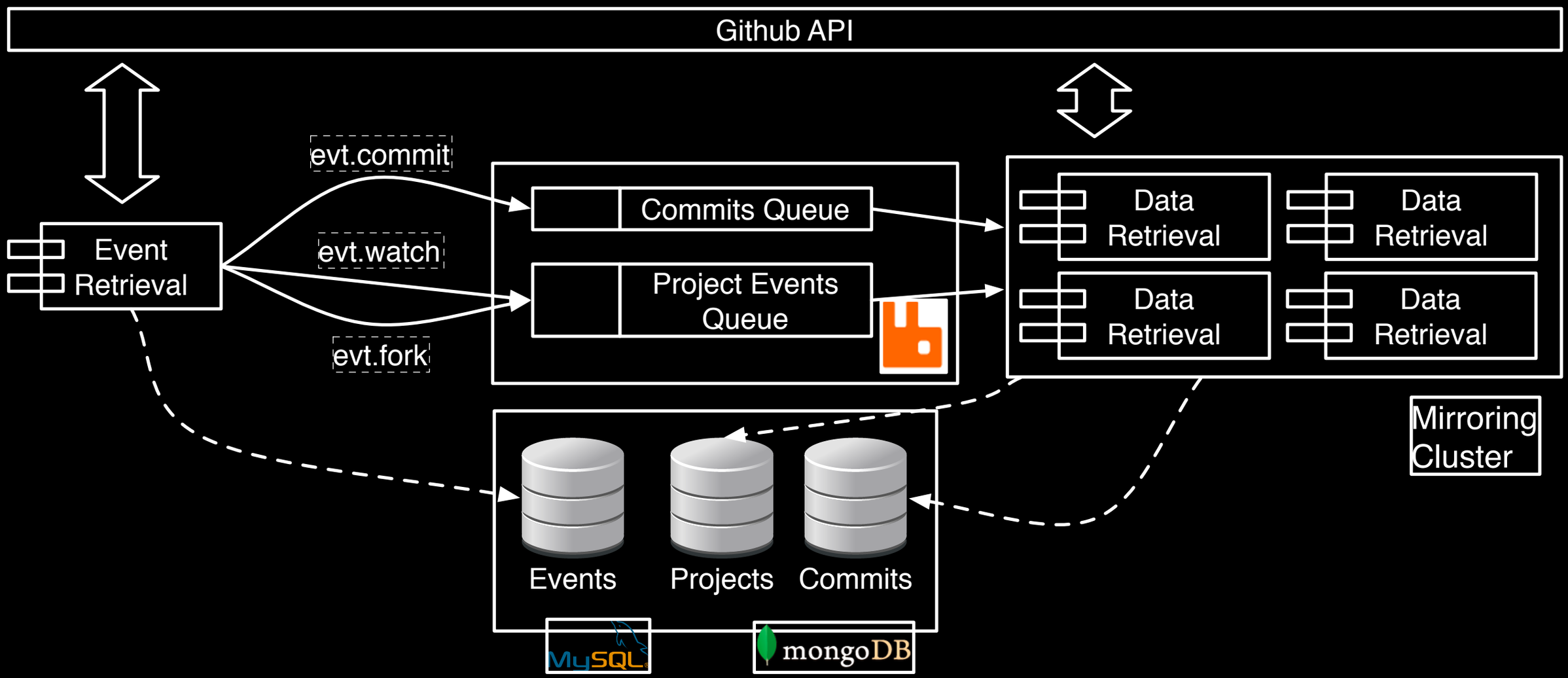

<<event>>
PushEvent
ensure_commit
<<api>>
/:user/:repo/sha
ensure_repo
ensure_user
<<api>>
/repos/:user/:repo/
ensure_commits
<<api>>
/users/:user
<<api>>
/repos/:user/:repo/commits
ensure_followers
ensure_commit_comments
<<api>>
/users/:user/
followers
<<api>>
/repos/:user/:repo/
commits/:sha/comments


<<event>>
PushEvent
ensure_commit
<<api>>
/:user/:repo/sha
ensure_repo
ensure_user
<<api>>
/repos/:user/:repo/
<<api>>
/users/:user
ensure_commits
<<api>>
/repos/:user/:repo/commits
ensure_followers
ensure_orgs
ensure_commit_comments
<<api>>
/users/:user/
followers
<<api>>
/users/:user/orgs
<<api>>
/repos/:user/:repo/
commits/:sha/comments

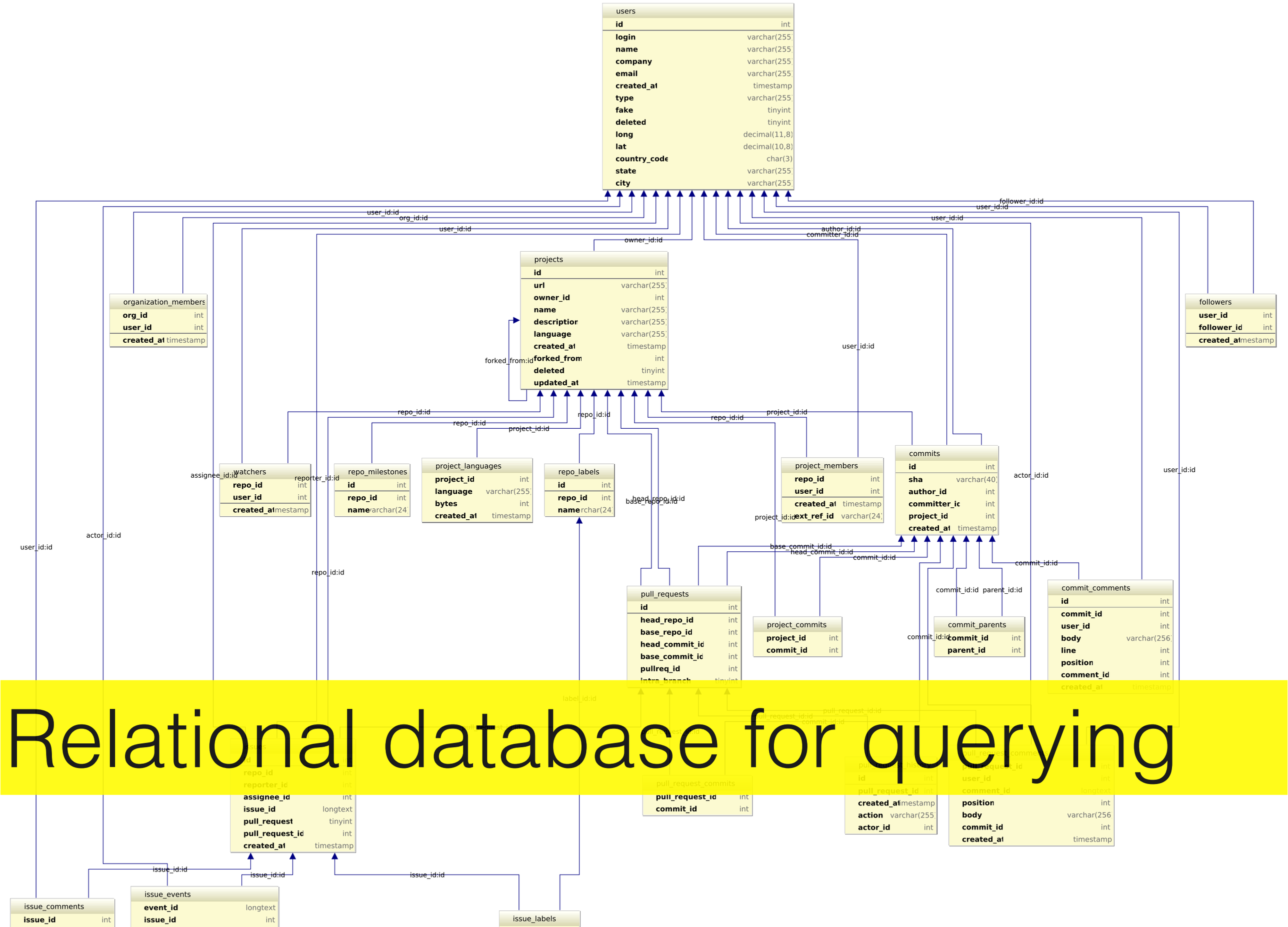

<<event>>
PushEvent
ensure_commit
<<api>>
/:user/:repo/sha
ensure_repo
ensure_user
<<api>>
/repos/:user/:repo/
ensure_commits
<<api>>
/repos/:user/:repo/commits
ensure_commit_comments
<<api>>
/repos/:user/:repo/
commits/:sha/comments
<<api>>
/users/:user
ensure_followers
ensure_orgs
<<api>>
/users/:user/
followers
<<api>>
/users/:user/orgs
ensure_teams
<<api>>
/orgs/:org/teams


<<event>>
PushEvent
ensure_commit
<<api>>
/:user/:repo/sha
ensure_repo
ensure_user
<<api>>
/repos/:user/:repo/
ensure_user
ensure_commits
<<api>>
/users/:user
<<api>>
/repos/:user/:repo/commits
ensure_followers
ensure_orgs
ensure_commit_comments
<<api>>
/users/:user/
followers
<<api>>
/users/:user/orgs
<<api>>
/repos/:user/:repo/
commits/:sha/comments
ensure_teams
<<api>>
/orgs/:org/teams


<<event>>
PushEvent
ensure_commit
<<api>>
/:user/:repo/sha
ensure_repo
ensure_user
<<api>>
/repos/:user/:repo/
ensure_user
ensure_commits
ensure_user
<<api>>
/users/:user
<<api>>
/repos/:user/:repo/commits
ensure_followers
ensure_orgs
ensure_commit_comments
<<api>>
/users/:user/
followers
<<api>>
/users/:user/orgs
<<api>>
/repos/:user/:repo/
commits/:sha/comments
ensure_teams
<<api>>
/orgs/:org/teams




# Distributed data processing




# Relational database for querying

users

repositories

organizations

issues

/users/:user

/user/repos

/orgs/:org

/repos/:user/:repo/issues


```
{
  "type": "User",
  "public_gists": 10,
  "login": "gousiosg",
  "followers": 64,
  "name": "Georgios Gousios",
  "public_repos": 20,
  "created_at": ...,
  "id": 386172,
  "following": 16,
                }
```


mongoDB

# MongoDB as query-able cache

Come in

WE'RE

OPEN

Open from the beginning

# MySQL database dumps

New! As of MySQL dump `mysql-2015-09-25`, we are distributing CSV files (one file per table) instead of `mysqldump` based backups. The provided arc
directory including a restore script and instructions on how to do the restore. See more information here.

You can also query MySQL. It is always loaded with the latest dump.

- 2016-02-01 (34969 MB)
- 2016-01-16 (34178 MB)
- 2016-01-08 (33942 MB)
- 2015-09-25 (32273 MB)
- 2015-08-07 (31537 MB)
- 2015-06-18 (33476 MB)
- 2015-04-01 (25075 MB)
- 2015-01-04 (16583 MB)
- 2014-11-10 (14418 MB)
- 2014-08-18 (11485 MB)
- 2014-04-02 (7013 MB)
- 2014-01-02 (5646 MB)
- 2013-10-12 (4312 MB)

# MongoDB database dumps

The MongoDB backups come in different formats:

- Bi-montly and by collection, from August 2015 up to Nov 2015
- Daily from 2015-12-01, includes all collections

To restore the full database, you need to download, extract and restore all of them.

# Periodic dumps of DBs

Daily dumps

2016-02-08, 2016-02-07, 2016-02-06, 2016-02-05, 2016-02-04, 2016-02-02, 2016-02-01, 2016-01-31, 2016-01-30, 2016-01-29, 2016-01-28, 2016-01-27, 2016-01-24, 2016-01-23, 2016-01-22, 2016-01-21, 2016-01-20, 2016-01-19, 2016-01-18, 2016-01-17, 2016-01-16, 2016-01-15, 2016-01-14, 2016-01-13, 2016-01-09, 2016-01-08, 2016-01-07, 2016-01-06, 2016-01-05, 2016-01-04, 2016-01-03, 2016-01-02, 2016-01-01, 2015-12-31, 2015-12-30, 2015-12-29, 2015-12-26, 2015-12-25, 2015-12-24, 2015-12-23, 2015-12-22, 2015-12-21, 2015-12-20, 2015-12-19, 2015-12-18, 2015-12-17, 2015-12-16, 2015-12-15, 2015-12-09, 2015-12-08, 2015-12-07, 2015-12-06, 2015-12-05, 2015-12-04, 2015-12-03, 2015-12-02, 2015-12-01,

DBLite

Database Explorer

localhost

- localhost
  - information_schema
  - ghtorrent
    - Tables
      - commit_comments
      - commit_parents
      - commits
      - followers
      - issue_comments
      - issue_events
      - issue_labels
      - issues
      - organization_members
      - project_commits
      - project_members
      - projects
      - pull_request_comments
      - pull_request_commits
      - pull_request_history
      - pull_requests
      - repo_labels
      - repo_milestones
      - schema_info
      - users
      - watchers
    - Views
    - Stored Procs
    - Functions
  - test

SQL*

```
1  select month(p.created_at) as month, year(p.created_at) as year, count(*) as contributing
2  from projects p
3  where exists (
4      select *
5      from pull_requests pr
6      where pr.head_repo_id = p.id
7  )
8  group by p.id, month, year
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
```

Conn: localhost | DB: ghtorrent | Table Structure | History | 1. Result

Export | Show SQL | Refresh | Auto Refresh

| month | year | contributing |
|---|---|---|
| 8 | 2011 | 1 |
| 6 | 2010 | 1 |
| 4 | 2011 | 1 |
| 7 | 2012 | 1 |
| 8 | 2012 | 1 |
| 10 | 2010 | 1 |
| 8 | 2012 | 1 |

Page 1 of 7487

# Query relational DB online

# Querying MySQL programmatically

To connect to the MySQL programmatic endpoint, you need a MySQL client (command line, graphical or program library) and SSH installed on your machine.

## Connection details

To obtain access, please send us your public key as described here.

1. When we contact you back, you will be able to setup an SSH tunnel with the following command: `ssh -L 3306:web.ghtorrent.org:3306 ghtorrent@web.ghtorrent.org`. Keep in mind that no shell will be allocated in the open SSH session.
2. You will then be able to connect to our server using the command: `mysql -u ght -h 127.0.0.1 ghtorrent` (user name: ght, no password, database: ghtorrent).

Here is an example session:




```
####
# on terminal session 1
$ ssh -L 3306:web.ghtorrent.org:3306 ghtorrent@web.ghtorrent.org
PTY allocation request failed on channel 2
#####
# on a different terminal
$ mysql -u ght -h 127.0.0.1 ghtorrent
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A



Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 1004
```

Query MySQL programmatically

# Querying MongoDB

While the GHTorrent project offers downloadable versions of the MongoDB raw dataset, downloading and restoring them to MongoDB can be very time consuming. For this reason, we have created a publicly available version of the data as they are collected by our main MongoDB server. The only prerequisite is to have a MongoDB client (command line, graphical or program library) and SSH installed on your machine.

## Obtaining access

1. Send us you public SSH key (usually in `~/.ssh/id_rsa.pub`). To create one, use `ssh-keygen`. We need your API key *as a file attachment*, because various email clients tend to break key strings at arbitraty locations.
2. When we contact you back, you will be able to setup an SSH tunnel with the following command: `ssh -L 27017:dutihr.st.ewi.tudelft.nl:27017 ghtorrent@dutihr.st.ewi.tudelft.nl`. Keep in mind that no shell will be allocated in the open SSH session.
3. You will then be able to connect to our server using the command: `mongo -u ghtorrentro -p ghtorrentro github`.

Here is an example session:

```
####
# on terminal session 1
$ ssh -L 27017:dutihr.st.ewi.tudelft.nl:27017 ghtorrent@dutihr.st.ewi.tudelft.nl
Password for tunneled port channel 2

#####
# on a different terminal
$ mongo -u ghtorrentro -p ghtorrentro github
MongoDB shell version: 3.0.3
```



Query MongoDB programmatically

# Streaming updates from GHTorrent

## Connection details

To obtain access, please send us your public key as described here.

## Declaring queues

Our queue server, RabbitMQ implements the AMQP protocol. Some familiarity with it is necessary before using the streaming service. The RabbitMQ getting started page is a very good starting point with lots of examples in many languages.

The streaming service uses topic exchanges and concequently message-based routing (see here for details). To start receiving messages, a client needs to:

1. connect to the server
2. declare a queue
3. bind the declared queue to the default exchange with routing key

The following examples are in Ruby.

## Connecting to the server

Assuming your connection works as described above, you should have port 5672 listening on localhost. You should connect and define the `ght-streams` exchange (if you define other exchnages, you will receive no messages as nobody will post there).

```
require 'bunny'
conn = Bunny.new(:host => "127.0.0.1", :port => 5672,
                 :username => "tbd", :password => "tbd")
conn.start
ch   = conn.create_channel
exchange = ch.topic('ght-streams')
```

## Declaring a queue

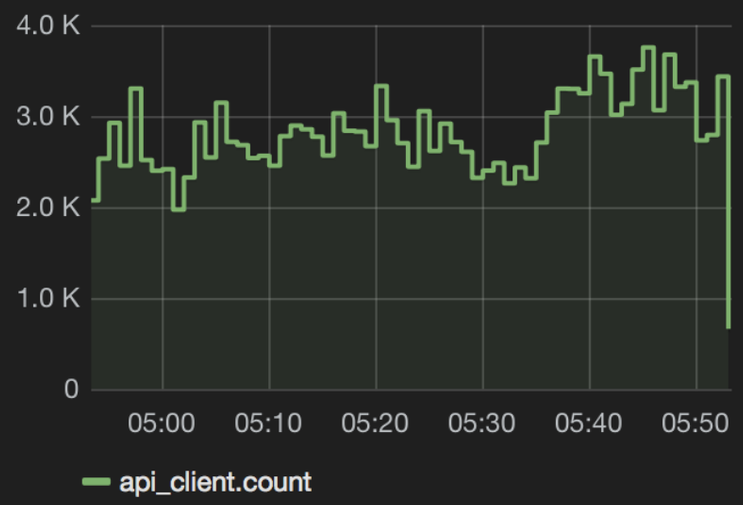

Streaming updates

dashboard.ghtorrent.org/dashboard/db/ghtorrent-stats

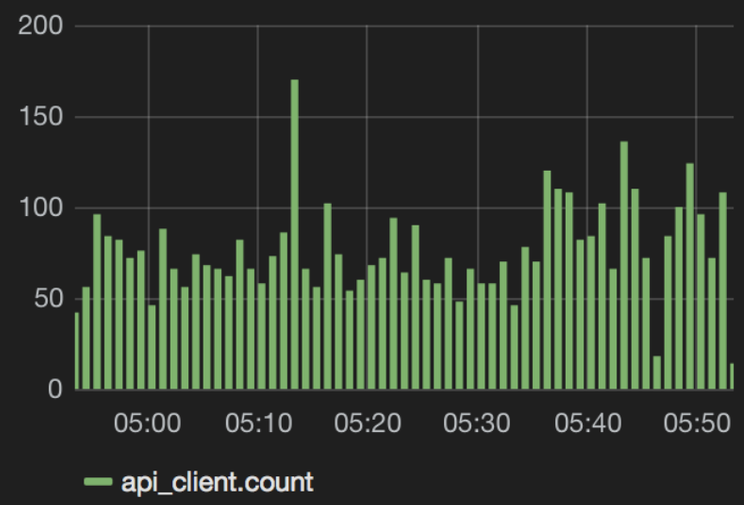
GHTorrent stats

Zoom Out | Last 1 hour | Refresh every 1m

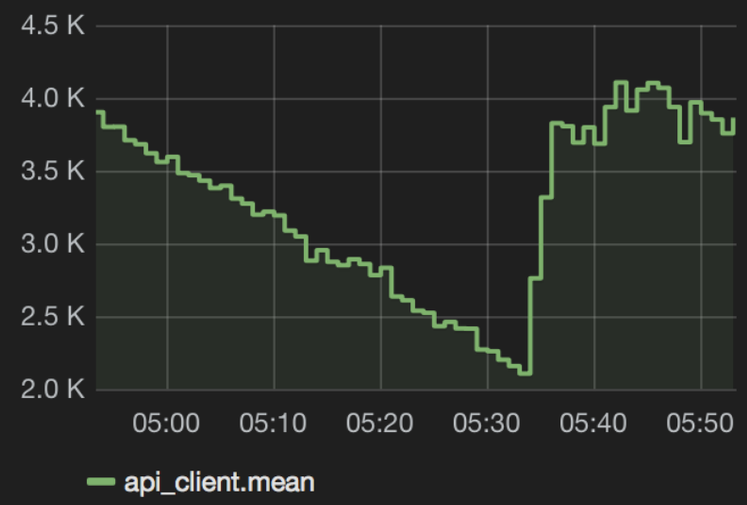
## API requests per minute



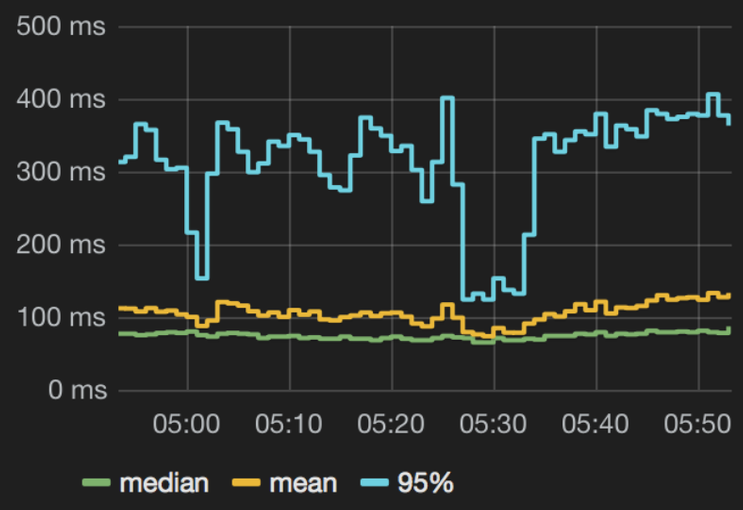

## API errors per minute



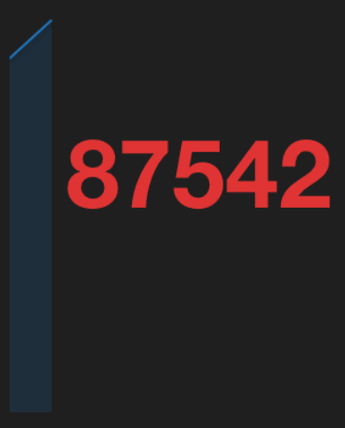

## Remaining API Quota




## API response time (median, mean, 95%)

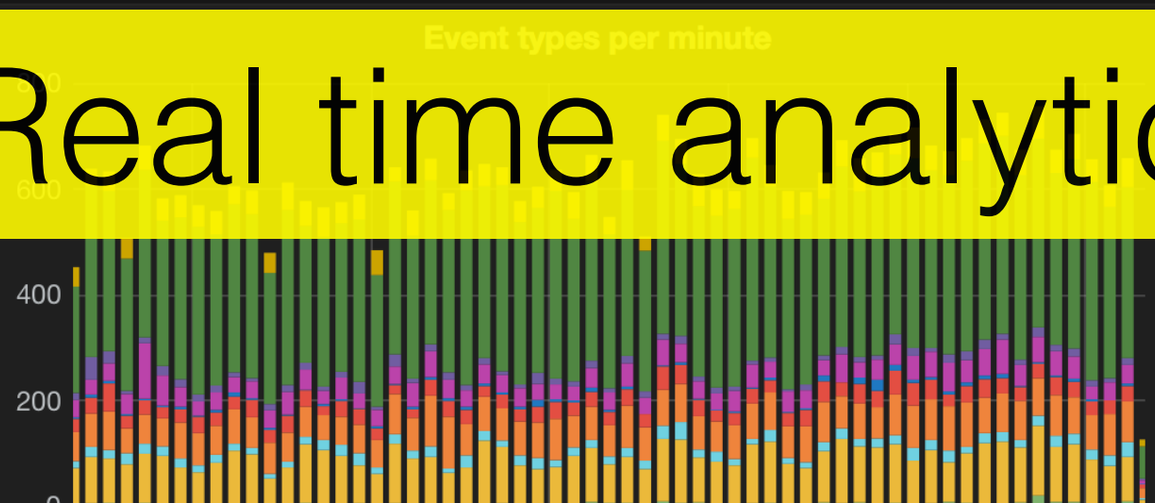



## Total requests last hour

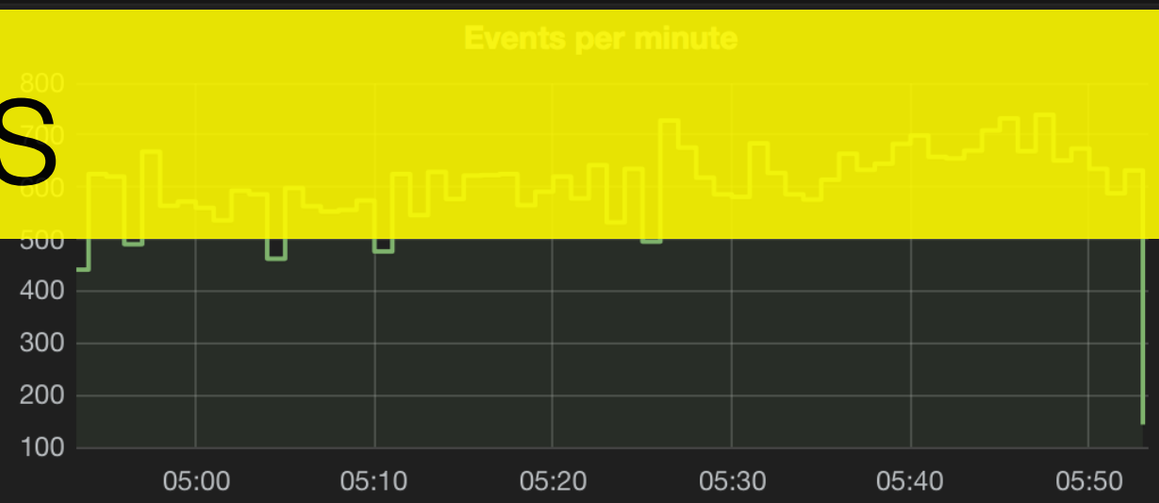



# Real time analytics

## Event types per minute




## Events per minute



```
1  select country_code, count(*)
2      from users
3      where fake is false
4          and deleted is false
5          and country_code is not null
6      group by country_code
7      order by count(*) desc
8
9
```

Conn: rs0 | DB: ghtorrent | Table Structure | History | 1. R

Export | Show SQL | Refresh | Auto Refresh

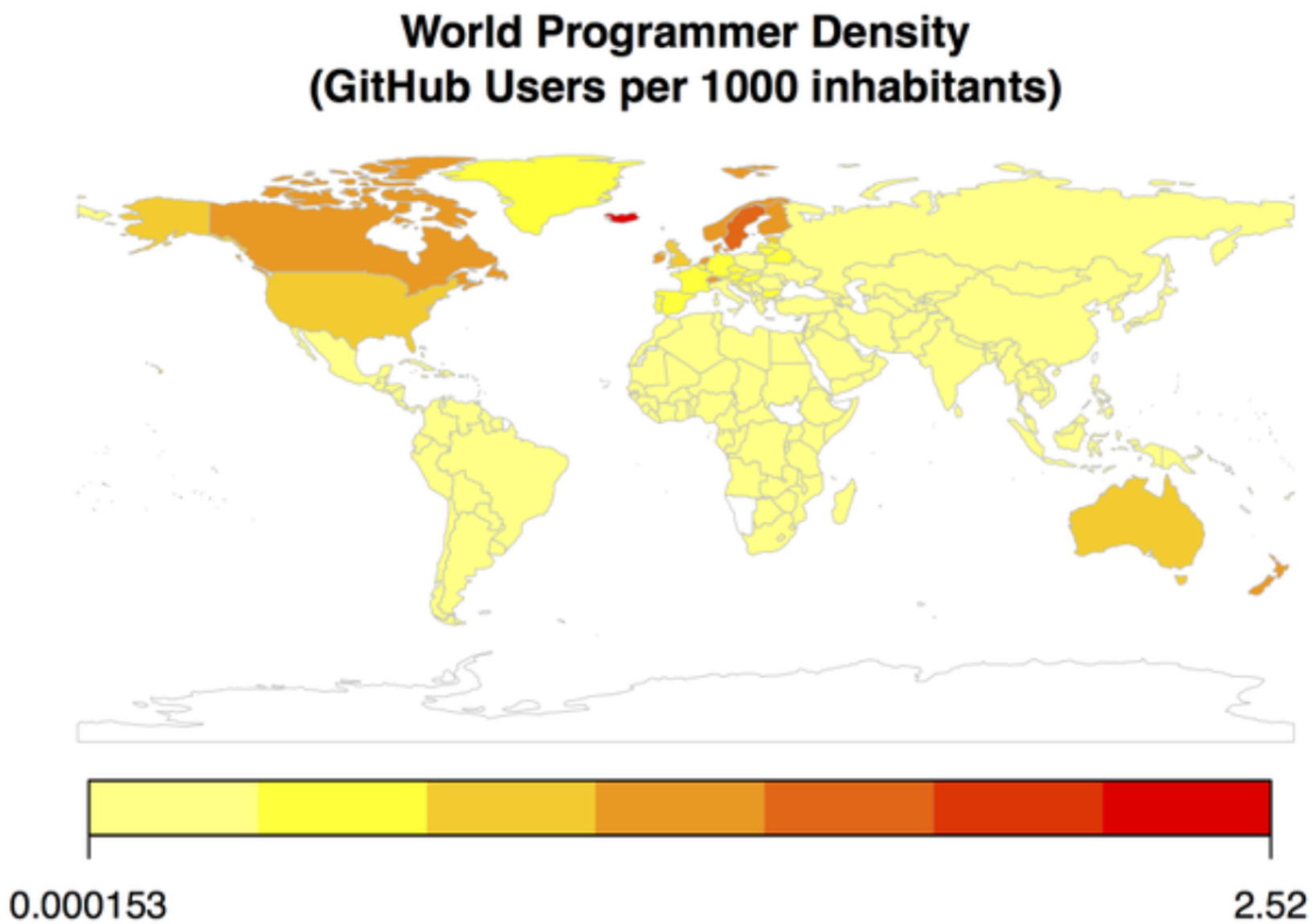
| country_code | count(*) |
|---|---|
| us | 283211 |
| gb | 57550 |
| cn | 50819 |
| in | 46204 |
| de | 45473 |
| ca | 34510 |
| br | 33529 |
| fr | 33003 |
| ru | 24441 |

# User geolocation

**Georgios Gousios**
@gousiosg

Follow

World programmer density map, calculated by geocoding all @github users that declared a location (w/ @ghtorrent)




RETWEETS 381 LIKES 187

SQL*

```
1  select u.login, p.name, p.language, pl.language as main_language,
2      pl.bytes, pl.created_at as date_measured
3      from project_languages pl, projects p, users u
4      where u.id = p.owner_id
5      and pl.project_id = p.id
6      and p.name = 'scala'
7      and u.login = 'scala';
8
9
```

Connection | DB: ghtorrent | Table: project_langu | History | **1. Result**

Export | Show SQL | Refresh | Auto Refresh

| login | name | language | main_language | bytes | date_measured |
|---|---|---|---|---|---|
| scala | scala | Scala | scala | 15934585 | 2015-10-29 10:46:57 |
| scala | scala | Scala | java | 1023702 | 2015-10-29 10:46:57 |
| scala | scala | Scala | javascript | 241109 | 2015-10-29 10:46:57 |
| scala | scala | Scala | shell | 57442 | 2015-10-29 10:46:57 |
| scala | scala | Scala | python | 44027 | 2015-10-29 10:46:57 |
| scala | scala | Scala | css | 37216 | 2015-10-29 10:46:57 |
| scala | scala | Scala | html | 6396 | 2015-10-29 10:46:57 |
| scala | scala | Scala | batchfile | 2970 | 2015-10-29 10:46:57 |
| scala | scala | Scala | ... | ... | 2015-10-29 10:46:57 |
| scala | scala | Scala | ruby | 142 | 2015-10-29 10:46:57 |
| scala | scala | Scala | c | 141 | 2015-10-29 10:46:57 |

# Full language retrieval

```
$ gem install sqlite3 bundler
$ git clone https://github.com/gousiosg/github-mirror
$ cd github-mirror
$ bundle install
$ mv config.yaml.standalone > config.yaml
$ ruby -Ilib bin/ght-retrieve-repo -t token rails rails
```

# Roll your own dataset

# Statistics

Since Feb 2012

**12TB** in MongoDB

**4.5B** rows in MySQL

**2GB** per hour

**120k** API reqs/hour

**46** user donated API keys

**130+** users, **80+** institutions

**80+** papers

**3** data mining challenges

**2** best paper awards

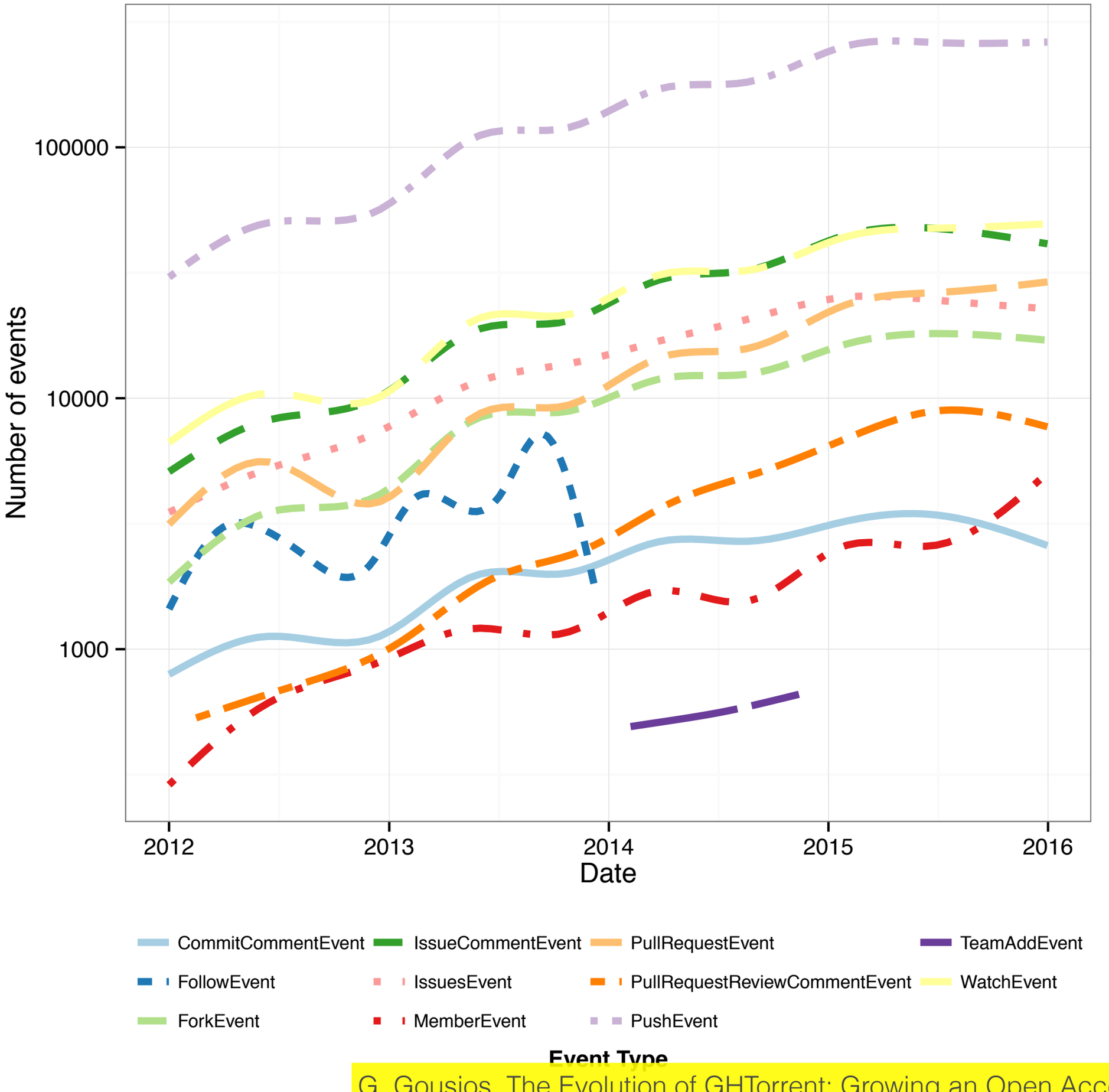
G. Gousios, "The GHTorrent dataset and tool suite," in MSR, 2013: 233-236

G. Gousios and D. Spinellis, " GHTorrent: GitHub's Data from a Firehose," in MSR, 2012, 12–21

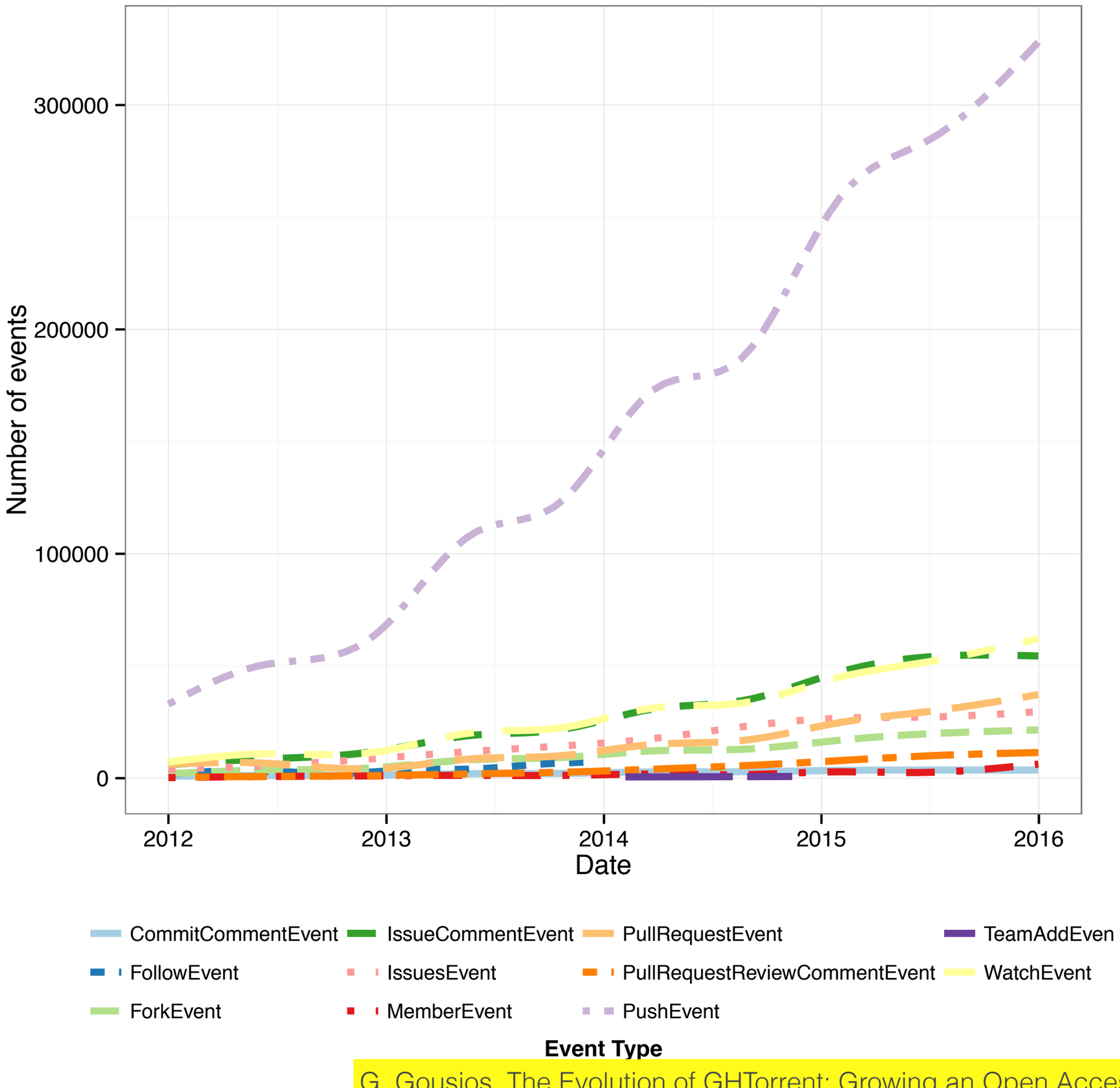
# Growth

| | MongoDB | MySQL | Diff 2016/2013 |
|---|---|---|---|
| Events | 476 | | **11.1x** |
| Users | 6,7 | 9,2 | **8.4x** |
| Repos | 28 | 25,5 | **21.8x** |
| Commits | 367 | 362 | **12.3x** |
| Issues | 24,1 | 25,3 | **10.3x** |
| Pull requests | 11,9 | 11,1 | **9.7x** |
| Issue comments | 42 | 43 | **14.6x** |
| Watchers | 51 | 37 | **6.6x** |

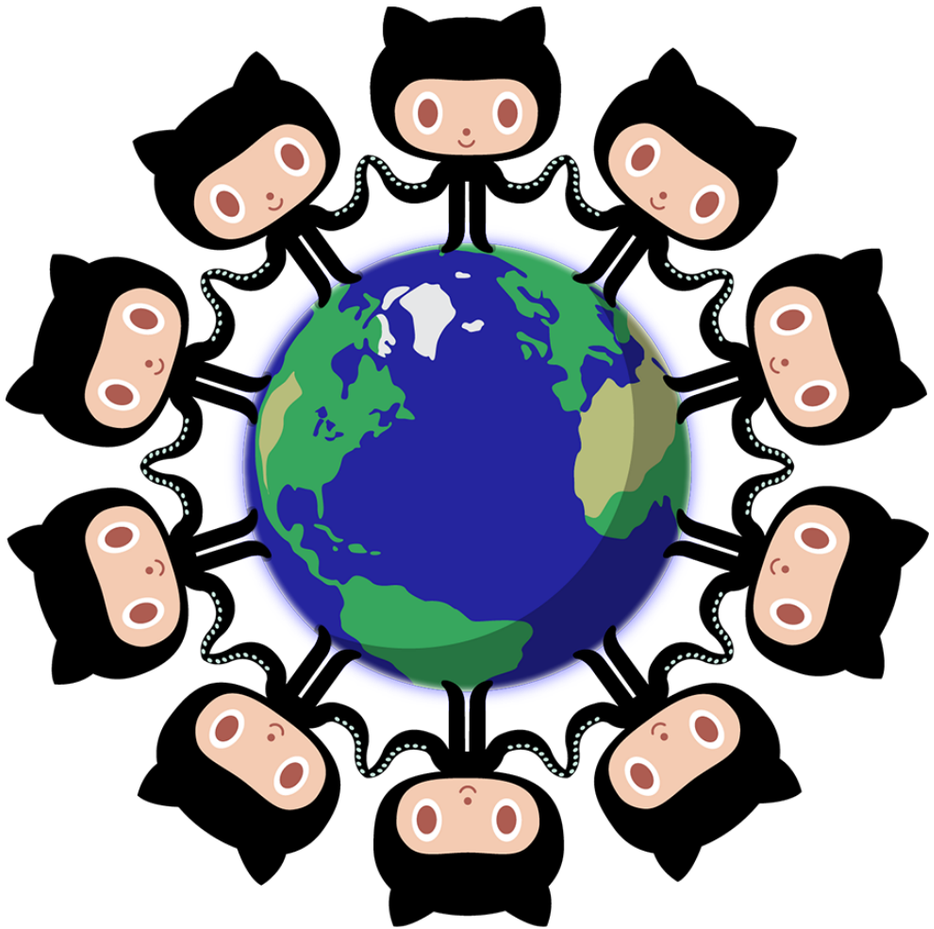
G. Gousios, The Evolution of GHTorrent: Growing an Open Access Dataset 10x



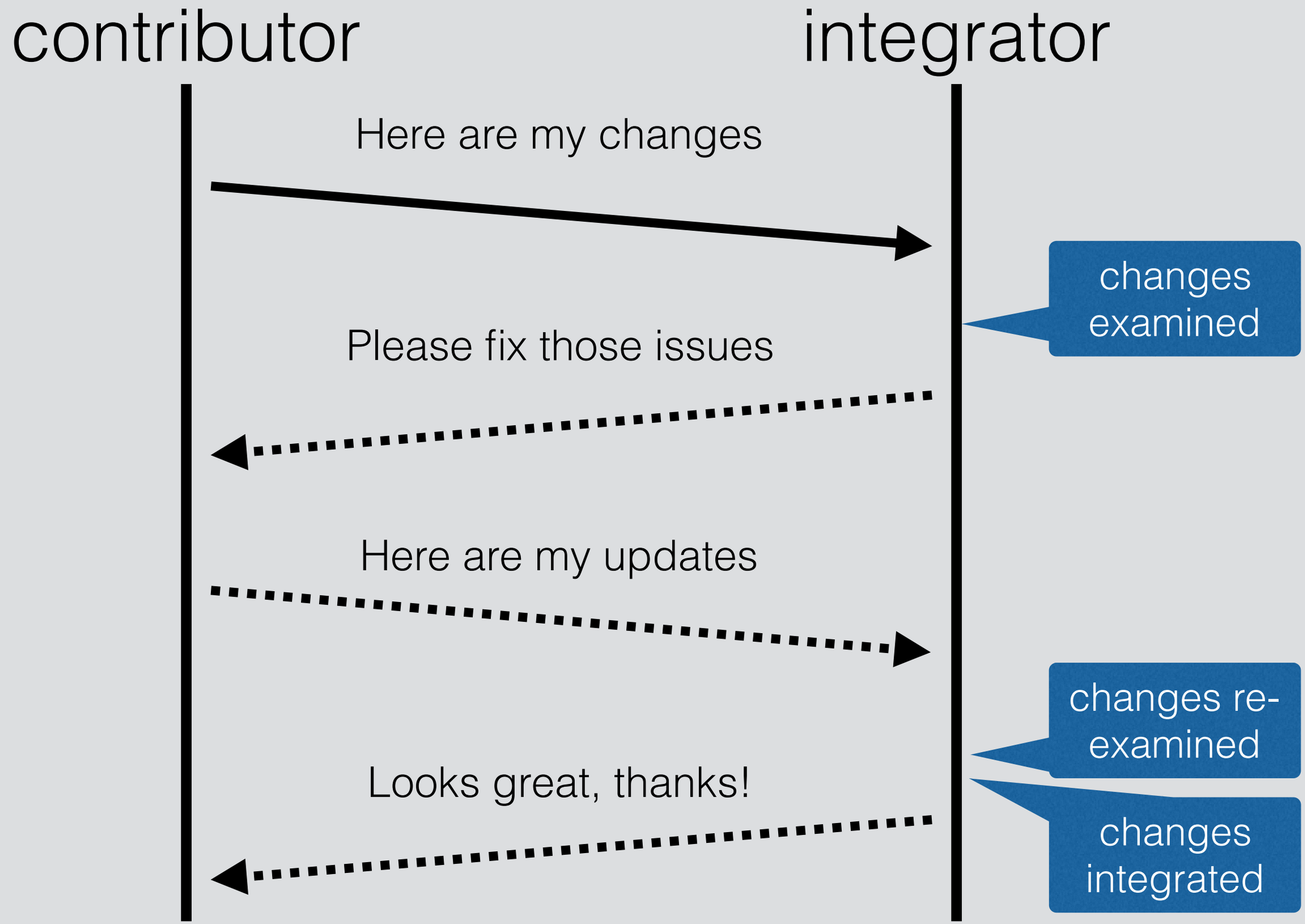

G. Gousios, The Evolution of GHTorrent: Growing an Open Access Dataset 10x




G. Gousios, The Evolution of GHTorrent: Growing an Open Access Dataset 10x



https://octodex.github.com


contributor
integrator
Here are my changes
changes examined
Please fix those issues
Here are my updates
changes re-examined
Looks great, thanks!
changes integrated

# What factors affect PR acceptance?

Across 5k repos/ ~1M pull requests

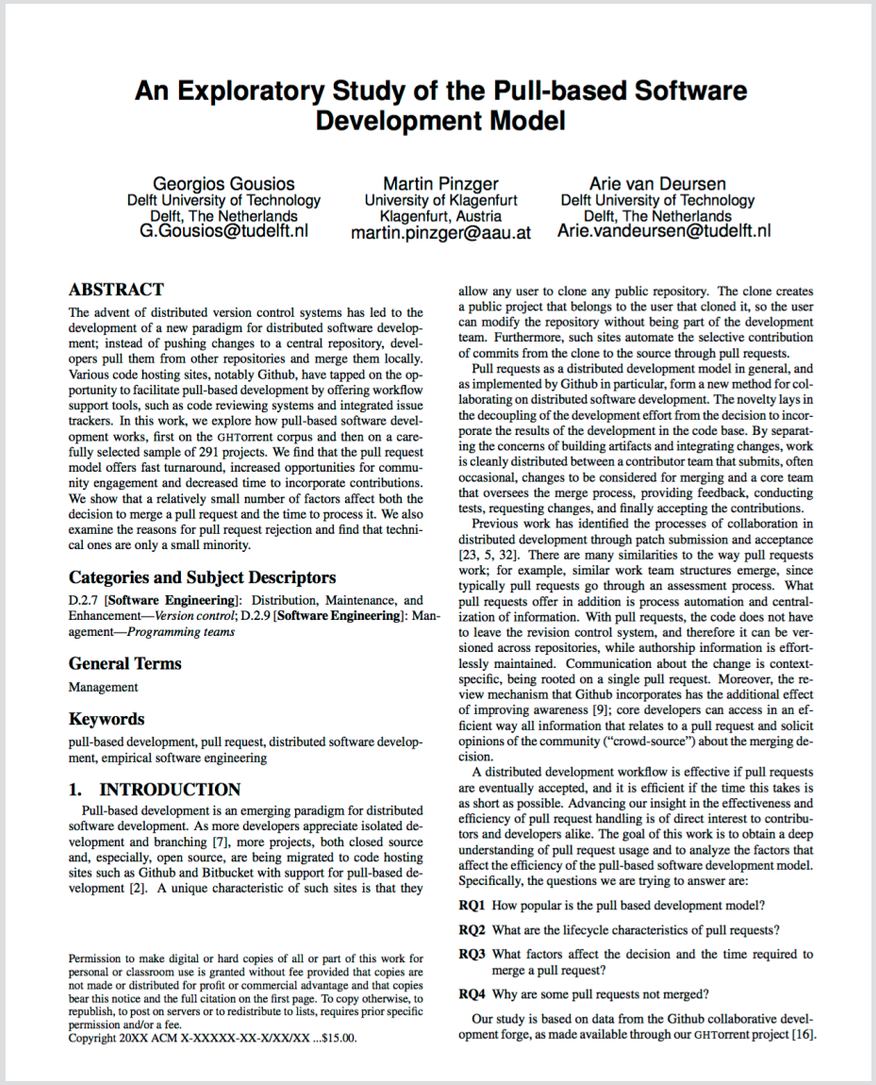
**85%** merged, 70% with merge button

**80%** < 150 lines, < 7 files, 3 commits

**66%** < 1 day to merge

**80%** 4 comments, 3 participants

Mostly rejected due to observability/awareness issues (not technical!)



**An Exploratory Study of the Pull-based Software Development Model**

Georgios Gousios
Delft University of Technology
Delft, The Netherlands
G.Gousios@tudelft.nl

Martin Pinzger
University of Klagenfurt
Klagenfurt, Austria
martin.pinzger@aau.at

Arie van Deursen
Delft University of Technology
Delft, The Netherlands
Arie.vandeursen@tudelft.nl

**ABSTRACT**

The advent of distributed version control systems has led to the development of a new paradigm for distributed software development; instead of pushing changes to a central repository, developers pull them from other repositories and merge them locally. Various code hosting sites, notably Github, have tapped on the opportunity to facilitate pull-based development by offering workflow support tools, such as code reviewing systems and integrated issue trackers. In this work, we explore how pull-based software development works, first on the GHTorrent corpus and then on a carefully selected sample of 291 projects. We find that the pull request model offers fast turnaround, increased opportunities for community engagement and decreased time to incorporate contributions. We show that a relatively small number of factors affect both the decision to merge a pull request and the time to process it. We also examine the reasons for pull request rejection and find that technical ones are only a small minority.

**Categories and Subject Descriptors**

D.2.7 [**Software Engineering**]: Distribution, Maintenance, and Enhancement—*Version control*; D.2.9 [**Software Engineering**]: Management—*Programming teams*

**General Terms**

Management

**Keywords**

pull-based development, pull request, distributed software development, empirical software engineering

**1. INTRODUCTION**

Pull-based development is an emerging paradigm for distributed software development. As more developers appreciate isolated development and branching [7], more projects, both closed source and, especially, open source, are being migrated to code hosting sites such as Github and Bitbucket with support for pull-based development [2]. A unique characteristic of such sites is that they allow any user to clone any public repository. The clone creates a public project that belongs to the user that cloned it, so the user can modify the repository without being part of the development team. Furthermore, such sites automate the selective contribution of commits from the clone to the source through pull requests.

Pull requests as a distributed development model in general, and as implemented by Github in particular, form a new method for collaborating on distributed software development. The novelty lays in the decoupling of the development effort from the decision to incorporate the results of the development in the code base. By separating the concerns of building artifacts and integrating changes, work is cleanly distributed between a contributor team that submits, often occasional, changes to be considered for merging and a core team that oversees the merge process, providing feedback, conducting tests, requesting changes, and finally accepting the contributions.

Previous work has identified the processes of collaboration in distributed development through patch submission and acceptance [23, 5, 32]. There are many similarities to the way pull requests work; for example, similar work team structures emerge, since typically pull requests go through an assessment process. What pull requests offer in addition is process automation and centralization of information. With pull requests, the code does not have to leave the revision control system, and therefore it can be versioned across repositories, while authorship information is effortlessly maintained. Communication about the change is context-specific, being rooted on a single pull request. Moreover, the review mechanism that Github incorporates has the additional effect of improving awareness [9]; core developers can access in an efficient way all information that relates to a pull request and solicit opinions of the community ("crowd-source") about the merging decision.

A distributed development workflow is effective if pull requests are eventually accepted, and it is efficient if the time this takes is as short as possible. Advancing our insight in the effectiveness and efficiency of pull request handling is of direct interest to contributors and developers alike. The goal of this work is to obtain a deep understanding of pull request usage and to analyze the factors that affect the efficiency of the pull-based software development model. Specifically, the questions we are trying to answer are:

**RQ1** How popular is the pull based development model?

**RQ2** What are the lifecycle characteristics of pull requests?

**RQ3** What factors affect the decision and the time required to merge a pull request?

**RQ4** Why are some pull requests not merged?

Our study is based on data from the Github collaborative development forge, as made available through our GHTorrent project [16].



G. Gousios, M. Pinzger, and A. van Deursen, "An Exploratory Study of the Pull-based Software Development Model," ICSE 2014, pp. 345–355

# Which factors affect PR acceptance?

**?**

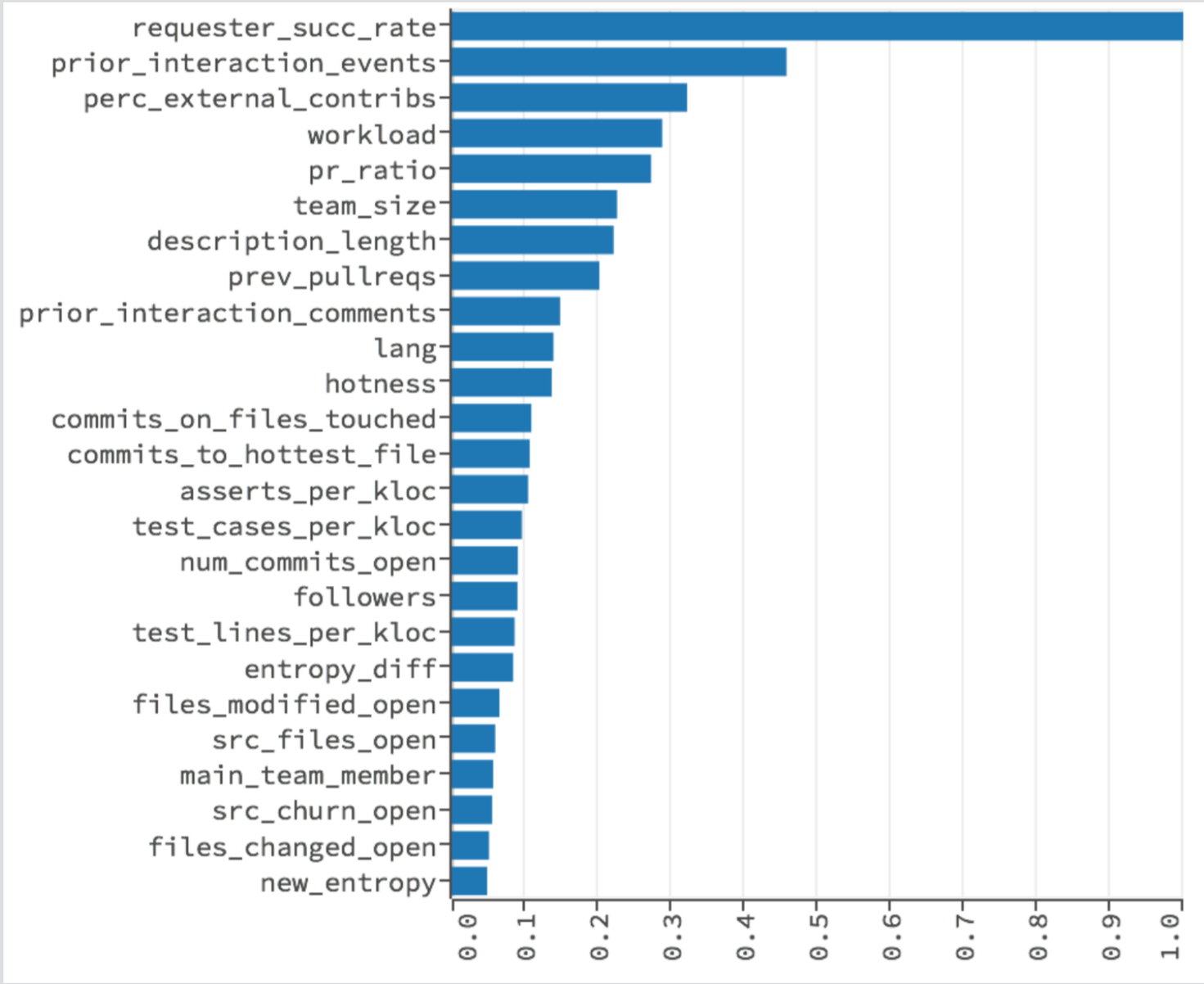
# Which factors affect PR acceptance?

Do we know the submitter?

Can we handle the workload?

What does the PR look like?

How ready is our project for PRs?




G. Gousios et al., “Distributed software development with pull requests”. Unpublished.

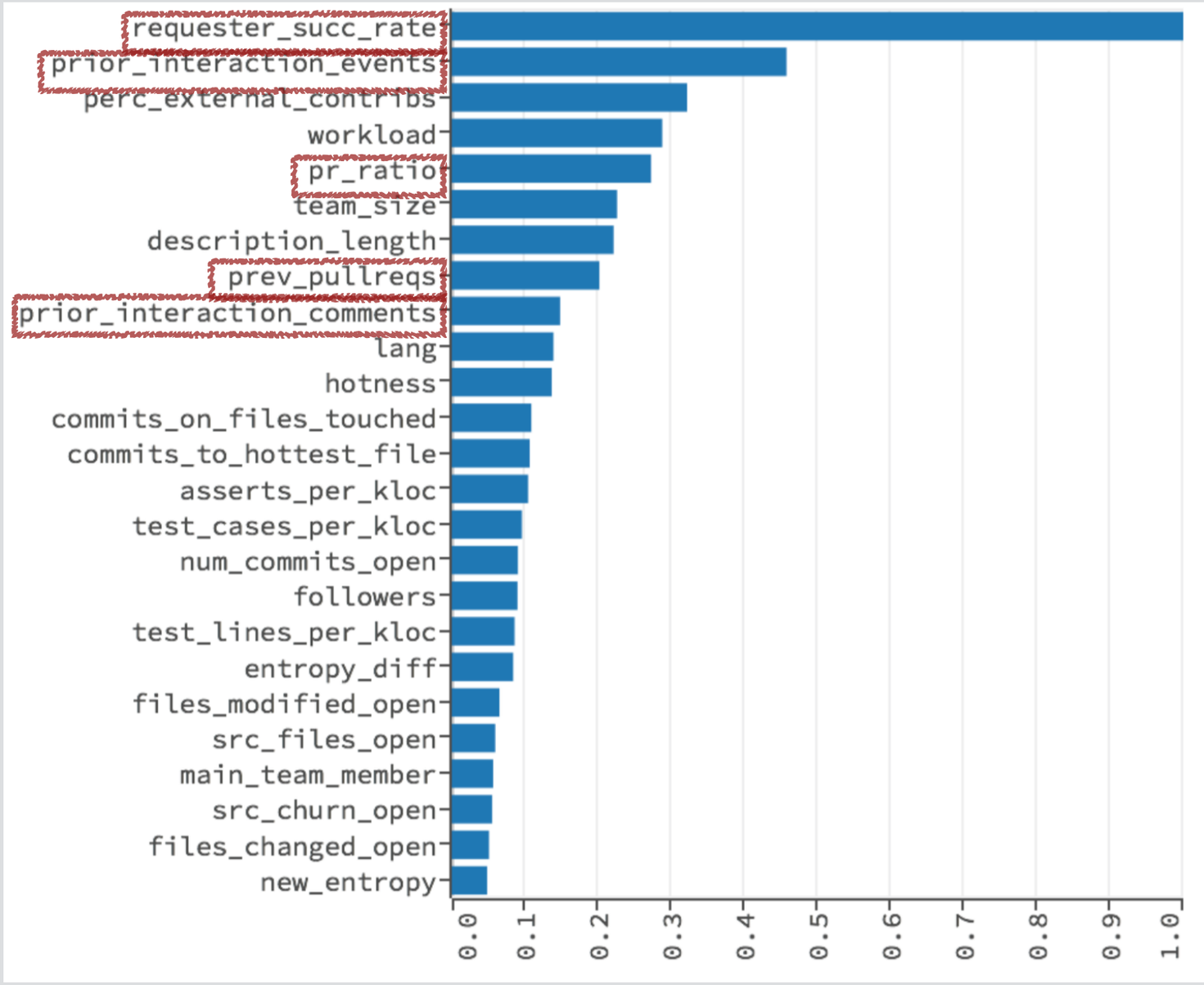
# Which factors affect PR acceptance?

Do we know the submitter?

Can we handle the workload?

What does the PR look like?

How ready is our project for PRs?




G. Gousios et al., “Distributed software development with pull requests”. Unpublished.

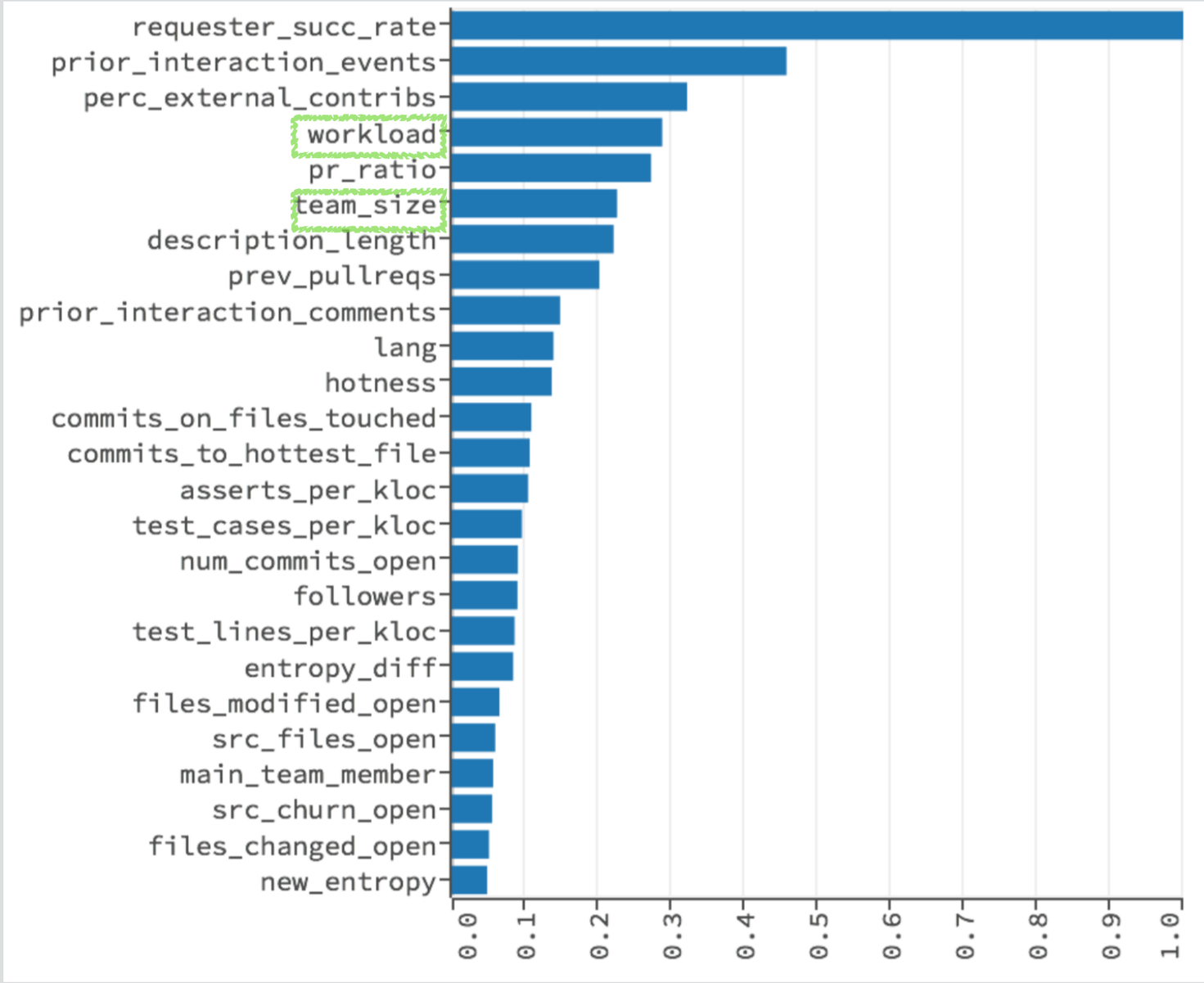
# Which factors affect PR acceptance?

Do we know the submitter?

Can we handle the workload?

What does the PR look like?

How ready is our project for PRs?




G. Gousios et al., “Distributed software development with pull requests”. Unpublished.

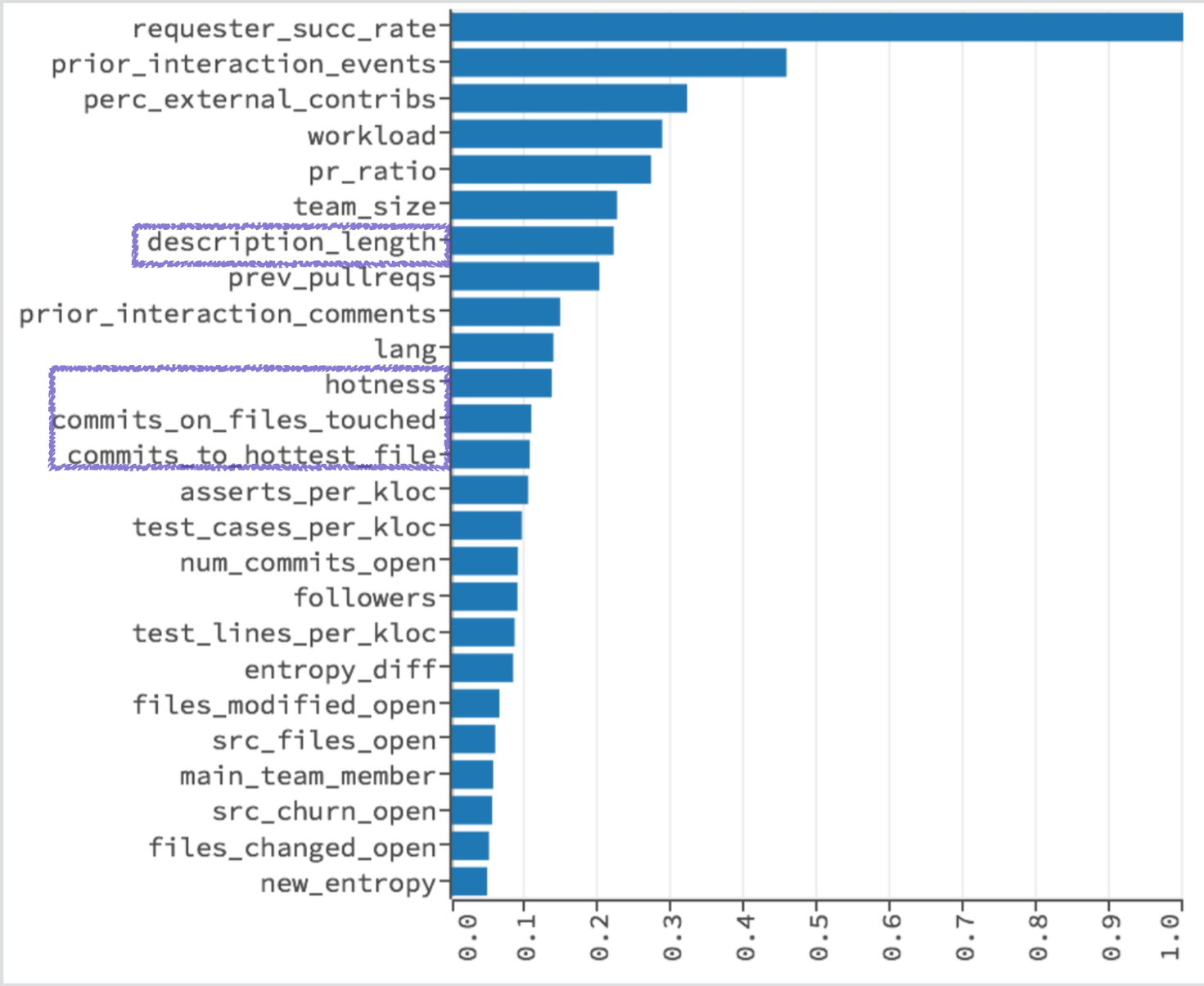
# Which factors affect PR acceptance?

Do we know the submitter?

Can we handle the workload?

What does the PR look like?

How ready is our project for PRs?




G. Gousios et al., “Distributed software development with pull requests”. Unpublished.

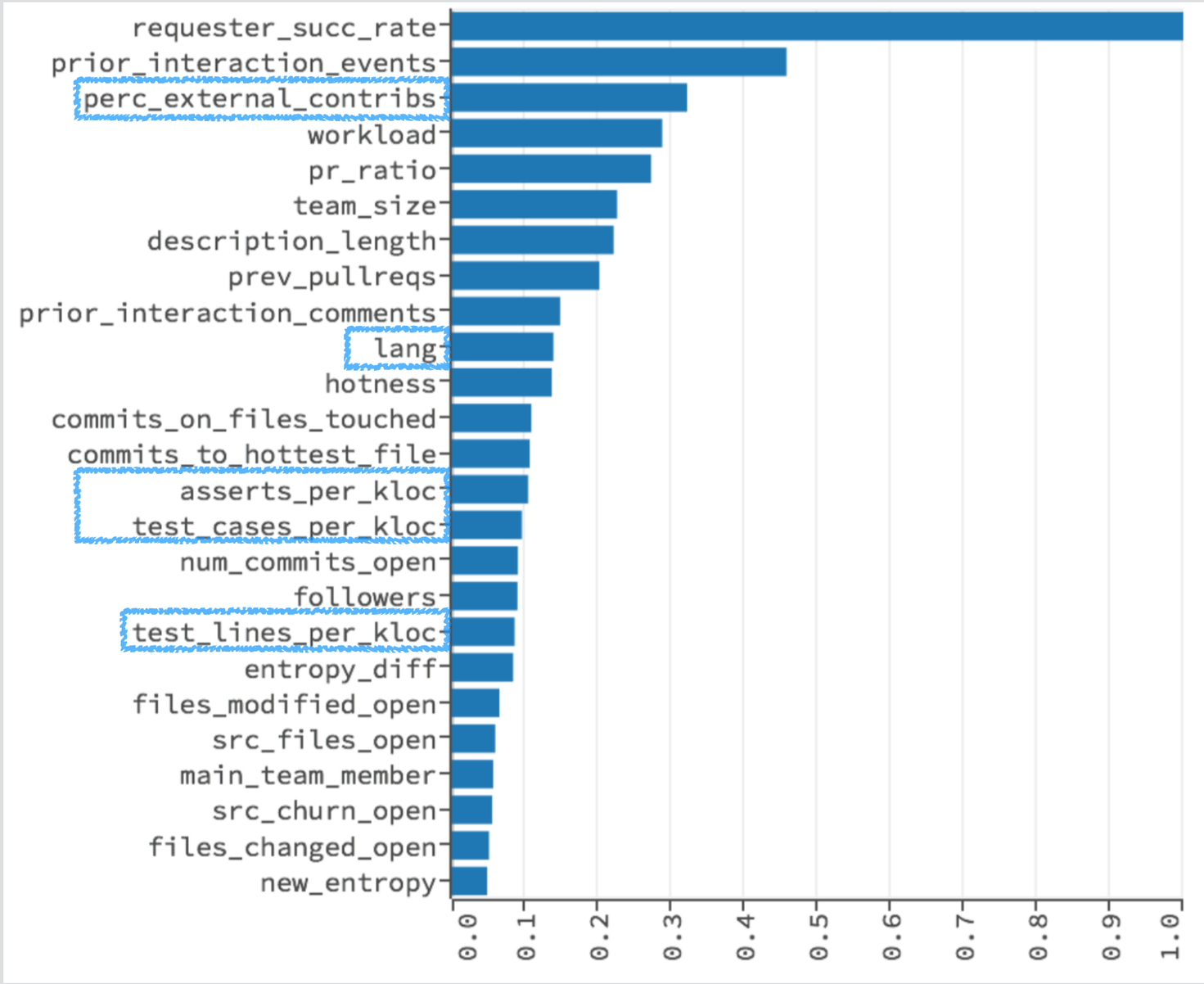
# Which factors affect PR acceptance?

Do we know the submitter?

Can we handle the workload?

What does the PR look like?

How ready is our project for PRs?




G. Gousios et al., “Distributed software development with pull requests”. Unpublished.

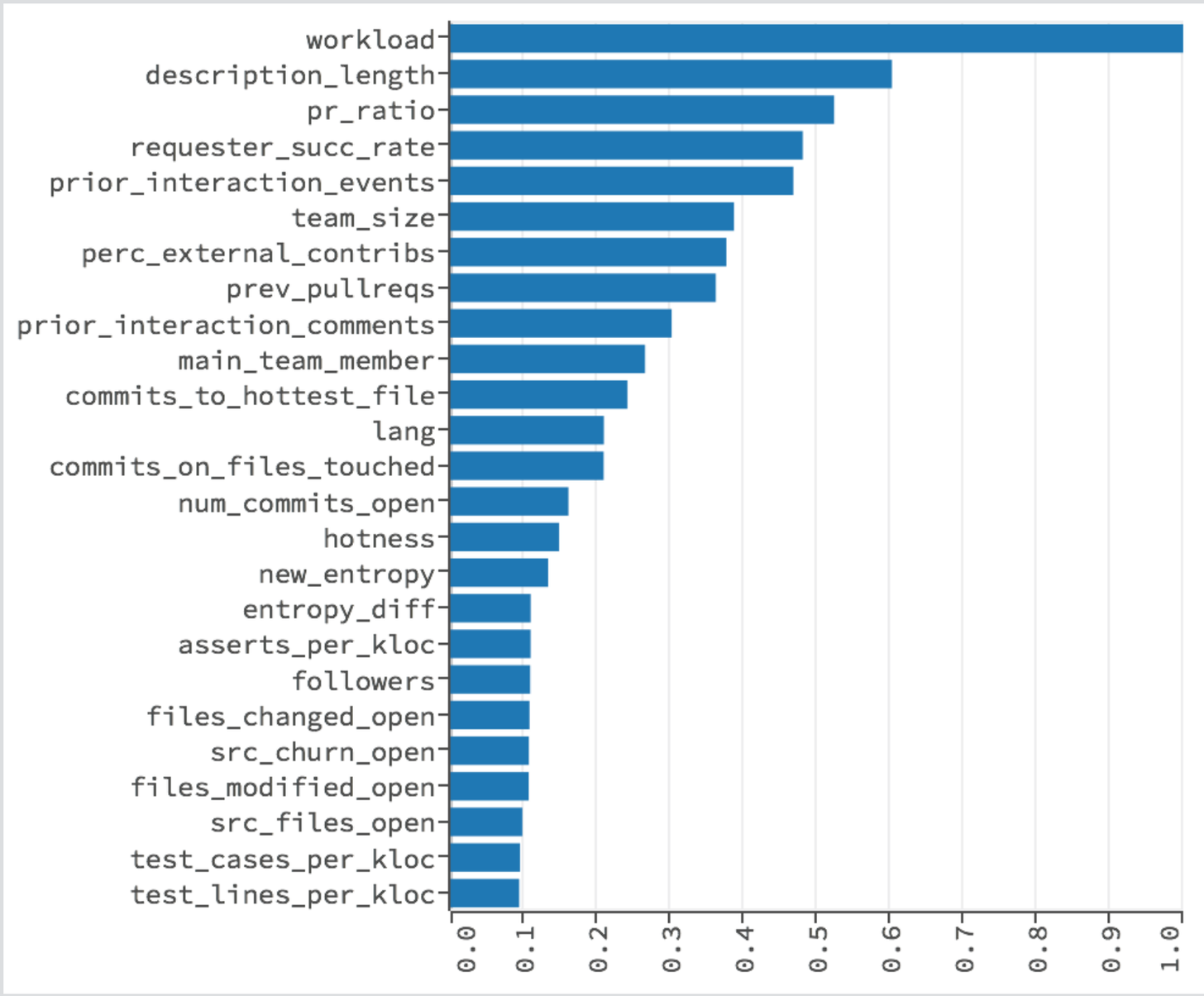
# Which factors affect the time to process PRs?

Can we handle the workload?

Do we know the submitter?

How ready is our project for PRs?

What does the PR look like?




G. Gousios et al., “Distributed software development with pull requests”. Unpublished.

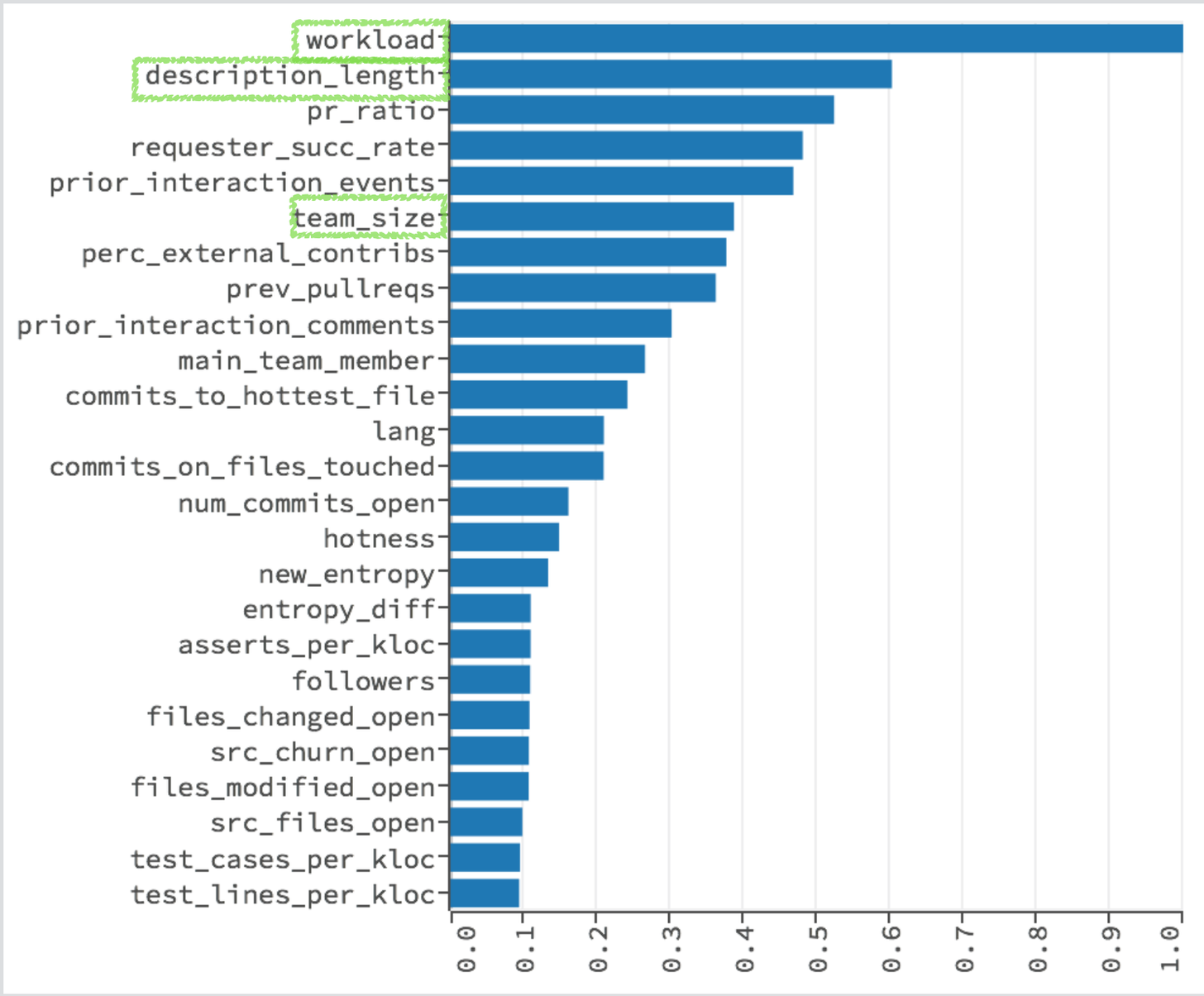
# Which factors affect the time to process PRs?

Can we handle the workload?

Do we know the submitter?

How ready is our project for PRs?

What does the PR look like?




G. Gousios et al., “Distributed software development with pull requests”. Unpublished.

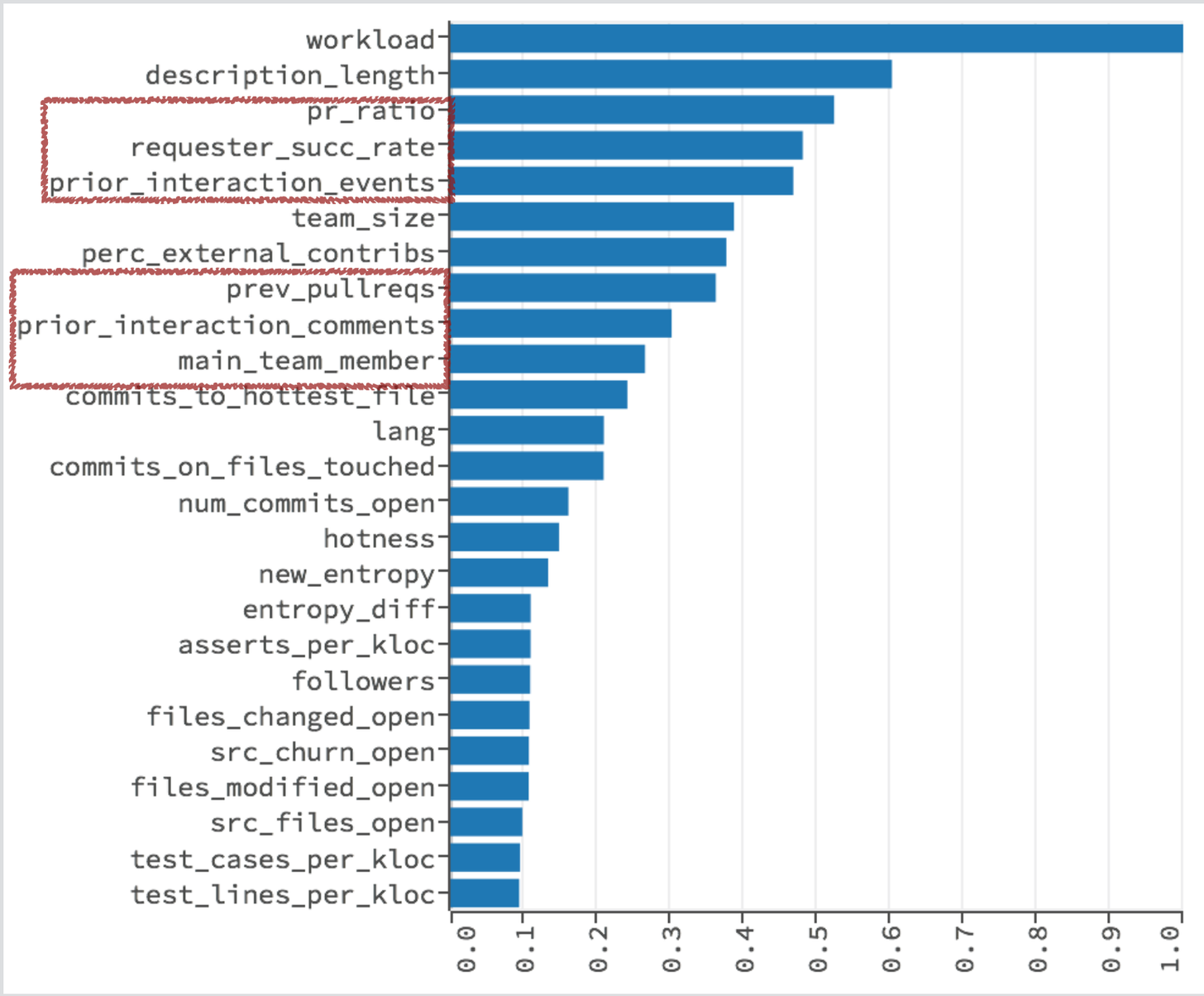
# Which factors affect the time to process PRs?

Can we handle the workload?

Do we know the submitter?

How ready is our project for PRs?

What does the PR look like?




G. Gousios et al., “Distributed software development with pull requests”. Unpublished.

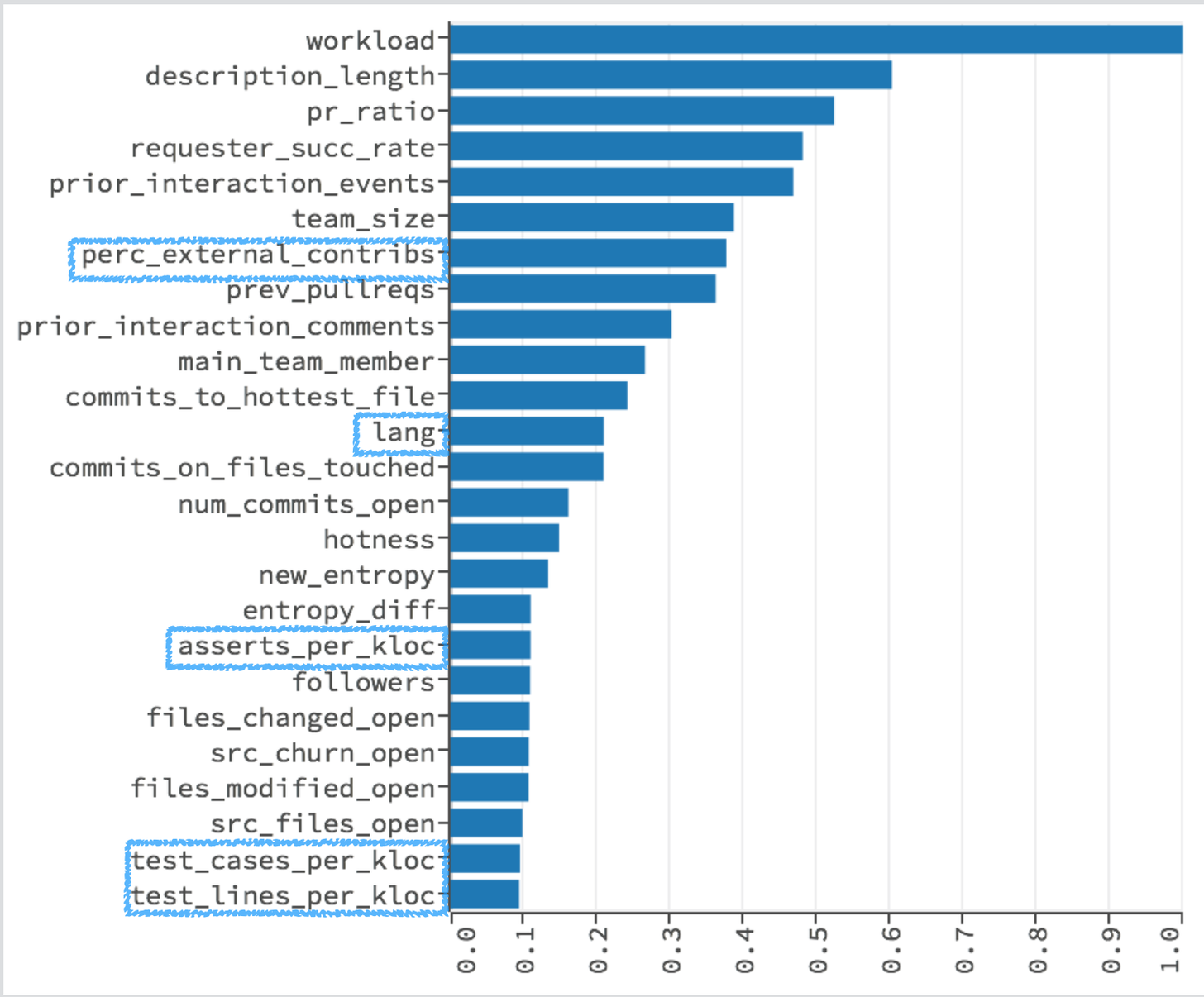
# Which factors affect the time to process PRs?

Can we handle the workload?

Do we know the submitter?

How ready is our project for PRs?

What does the PR look like?




G. Gousios et al., “Distributed software development with pull requests”. Unpublished.

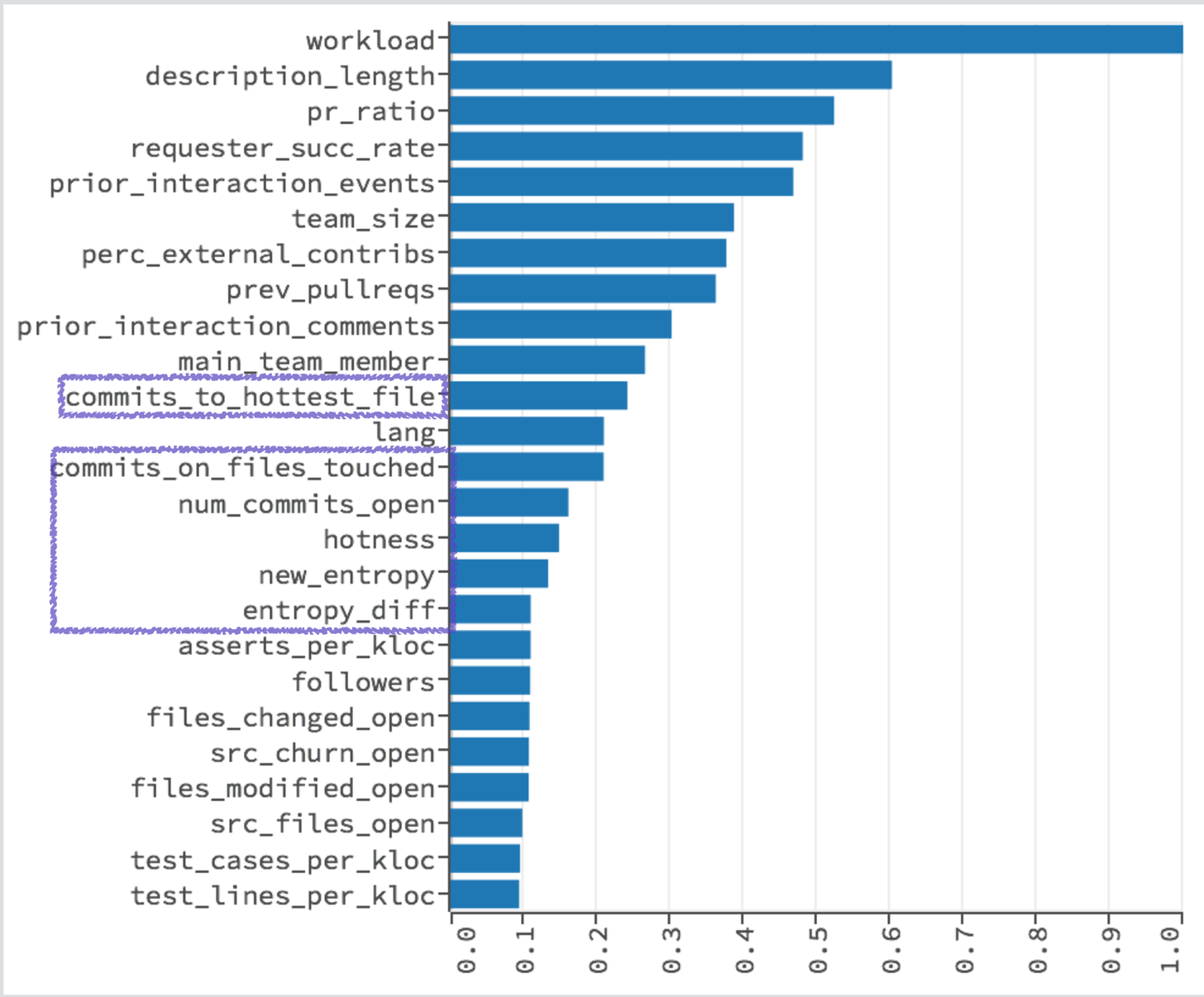
# Which factors affect the time to process PRs?

Can we handle the workload?

Do we know the submitter?

How ready is our project for PRs?

What does the PR look like?




G. Gousios et al., “Distributed software development with pull requests”. Unpublished.

# What do ***integrators*** actually believe?

Survey of **650** integrators

Generally, few complains about the process

Points of pain are mostly social (workload, drive-by PRs, explaining rejection)

Needed tools

Quality analysis

Impact analysis

Work prioritization

Work Practices and Challenges in Pull-Based Development: The Integrator's Perspective

Georgios Gousios*, Andy Zaidman†, Margaret-Anne Storey‡, Arie van Deursen†
* Radboud University Nijmegen, the Netherlands
Email: g.gousios@cs.ru.nl
† Delft University of Technology, the Netherlands
Email: {a.e.zaidman, arie.vandeursen}@tudelft.nl
‡ University of Victoria, BC, Canada
Email: mstorey@uvic.ca

***Abstract*—In the pull-based development model, the integrator has the crucial role of managing and integrating contributions. This work focuses on the role of the integrator and investigates working habits and challenges alike. We set up an exploratory qualitative study involving a large-scale survey of 749 integrators, to which we add quantitative data from the integrator's project. Our results provide insights into the factors they consider in their decision making process to accept or reject a contribution. Our key findings are that integrators struggle to maintain the quality of their projects and have difficulties with prioritizing contributions that are to be merged. Our insights have implications for practitioners who wish to use or improve their pull-based development process, as well as for researchers striving to understand the theoretical implications of the pull-based model in software development.**

## I. Introduction

Pull-based development as a distributed development model is a distinct way of collaborating in software development. In this model, the project's main repository is not shared among potential contributors; instead, contributors fork (clone) the repository and make their changes independent of each other. When a set of changes is ready to be submitted to the main repository, they create a pull request, which specifies a local branch to be merged with a branch in the main repository. A member of the project's core team (from hereon, the *integrator*[1]) is responsible to inspect the changes and integrate them into the project's main development line.

The role of the integrator is crucial. The integrator must act as a guardian for the project's quality while at the same time keeping several (often, more than ten) contributions "in-flight" through communicating modification requirements to the original contributors. Being a part of a development team, the integrator must facilitate consensus-reaching discussions and timely evaluation of the contributions. In Open Source Software (OSS) projects, the integrator is additionally taxed with enforcing an online discussion etiquette and ensuring the project's longevity by on-boarding new contributors.

The pull-based development process is quickly becoming a widely used model for distributed software development [1]. On GitHub alone, it is currently being used exclusively or complementary to the shared repository model in almost half of the collaborative projects. With GitHub hosting more than 1 million collaborative projects and competing services, such as BitBucket and Gitorious, offering similar implementations of the pull-based model, we expect the pull-based development model to become the default model for distributed software development in the years to come.

By better understanding the work practices and the challenges that integrators face while working in pull-based settings, we can inform the design of better tools to support their work and come up with best practices to facilitate efficient collaboration. To do so, we set up an exploratory qualitative investigation and survey integrators on how they use the pull-based development model in their projects. Our field of study is GitHub; using our GHTorrent database [2], we aimed our survey at integrators from high profile and high volume projects. An explicit goal is to learn from many projects rather than study a few projects in depth. We therefore use surveys as our main research instrument, generously sprinkled with open-ended questions. We motivate our survey questions based on a rigorous analysis of the existing literature and our own experience with working with and analysing the pull-based model during the last 2 years. We conducted a two-round (pilot and main) survey with 21 and 749 respondents respectively.

Our main findings reveal that integrators successfully use pull requests to solicit external contributions and we provide insights into the decision making process that integrators go through while evaluating contributions. The two key factors that integrators are concerned with in their day-to-day work are *quality* and *prioritization*. The quality phenomenon manifests itself by the explicit request of integrators that pull requests undergo code review, their concern for quality at the source code level and the presence of tests. Prioritization is also a concern for integrators as they typically need to manage large amounts of contribution requests simultaneously.

## II. Background and Related Work

The goal of distributed software development methods is to allow developers to work on the same software product while being geographically and timezone dispersed [3]. The proliferation of distributed software development techniques

[1]Also referred to as "integration manager": http://git-scm.com/book/en/Distributed-Git-Distributed-Workflows. We use the term integrator for brevity.

G. Gousios, A. Zaidman, M.-A. Storey, and A. van Deursen, "Work Practices and Challenges in Pull-Based Development: The Integrator's Perspective," ICSE 2015, pp. 358–368.

# What do ***contributors*** actually believe?

Survey of **640** contributors.
Similar issues, reversed

Awareness

Asynchrony

Responsiveness

**Work Practices and Challenges in Pull-Based Development: The Contributor's Perspective**

Georgios Gousios
Radboud University Nijmegen
Nijmegen, the Netherlands
g.gousios@cs.ru.nl

Margaret-Anne Storey
University of Victoria
BC, Canada
mstorey@uvic.ca

Alberto Bacchelli
Delft University of Technology
Delft, the Netherlands
a.bacchelli@tudelft.nl

**ABSTRACT**

The pull-based development model is an emerging way of contributing to distributed software projects and it is gaining enormous popularity within the open source software (OSS) world. Previous work examined this model by focusing on projects and their owners, we complement it by examining the work practices of project contributors and the challenges they face.

We conduct a survey with 645 top contributors to active OSS projects using the pull-based model on GitHub, the prevalent social coding site, also analyzing traces extracted from corresponding GitHub repositories. We find that: Contributors have a strong interest in maintaining awareness on the project status to get inspiration and to avoid duplicating work, but they do not actively propagate information; communication within pull-requests is reportedly limited to low-level concerns and contributors often use communication channels external to pull-requests; challenges are mostly social in nature, with integrators' low responsiveness being the most reported; and the increased transparency of this setting is a confirmed motivation to contribute. Based on those findings, we formulate recommendations for practitioners to streamline the contribution process and discuss potential future research directions.

**Categories and Subject Descriptors**

D.2.7 [**Software Engineering**]: Distribution, Maintenance, and Enhancement—*Version control*; D.2.9 [**Software Engineering**]: Management—*Programming teams*

**Keywords**

pull based development, open-source contribution, pull request, distributed software development, GitHub

**1. INTRODUCTION**

Distributed software development projects employ collaboration models and patterns to streamline the process of integrating incoming contributions [37]. The pull-based development model is a recent form of distributed software development [27] that is gaining tremendous traction in the open source software (OSS) world.





Figure 1: Monthly growth of pull request usage on GitHub.

As Figure 1 shows, its popularity is constantly growing; on January 2016, 135,000 repositories received more that 600,000 pull requests. In total, 1,000,000 collaborative projects (*i.e.*, 45% of all collaborative projects) on GitHub (*i.e.*, the prevalent social coding site) used at least one pull request during their lifetime.

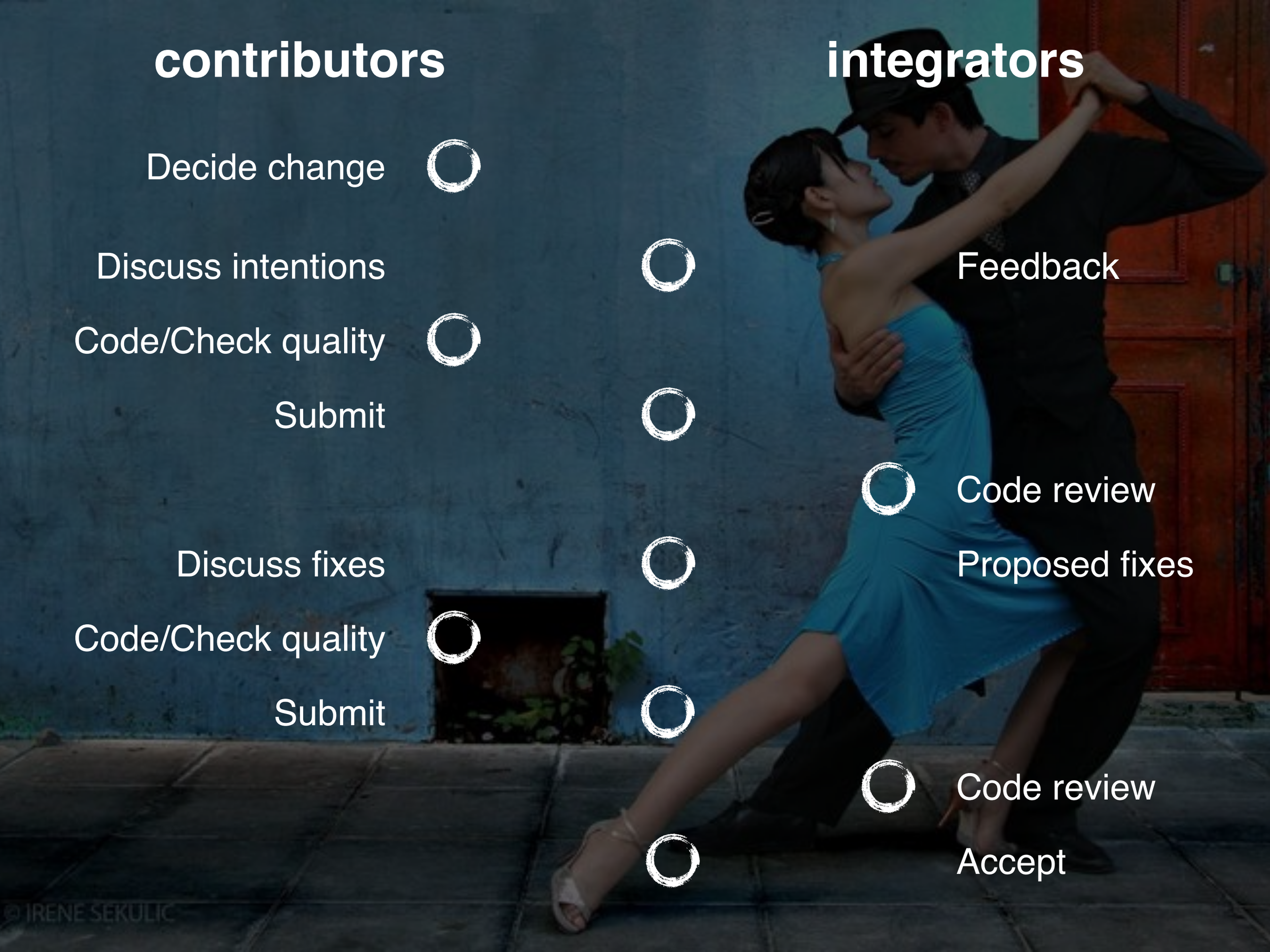
As opposed to more classic form of contributions (*e.g.*, change sets sent to development mailing lists [6] to issue tracking systems [5] or through direct access to the version control system [19]), in the pull-based model, contributors fork (*i.e.*, locally duplicate) the main repository of the project they want to contribute to, make their changes independently, then create a pull request (PR) asking to merge their changes within the main repository. Then the members of the project's core team (the *integrators*) are responsible for evaluating the quality of the contributions, proposing corrections, engaging in discussion with the contributors, and eventually merging or rejecting the changes.

Social coding sites (*e.g.*, GitHub [22], Bitbucket [8], and Gitorious [23]) offer the pull-based development model in conjunction with social media functions, which allow users to subscribe to and/or visualize information about activities of projects and users and offer threaded asynchronous communication within PRs.

To grasp the complexity of the pull-based development model, as offered by social coding sites, it is necessary to examine it from multiple perspectives. Previous research considered the lifetime characteristics of PRs [27], macroscopic factors that lead to contribution acceptance [27, 46], the barriers faced by first time contributors [45], how contributions are evaluated through discussions [47], and the working habits and challenges faced by *integrators* [28]. Here we present *the contributor's perspective*, by investigating contributors' work habits and the challenges they face.

The overall goal with this work is to *understand how contributing to OSS projects works using the pull-based development model* in the context of social coding sites. Understanding the contributor's perspective is needed to reveal weaknesses with the pull-based model and to guide the design of tools and processes to support contributors' work, which is essential part of the workflow. More-

G. Gousios, M.-A. Storey, and A. Bacchelli, "Work Practices and Challenges in Pull-Based Development: The Contributor's Perspective," ICSE, 2016.


contributors
integrators
Decide change
Discuss intentions
Feedback
Code/Check quality
Submit
Code review
Proposed fixes
Discuss fixes
Code/Check quality
Submit
Code review
Accept
© IRENE SEKULIC

## contributors

## integrators

Decide change

Discuss intentions

Code/Check quality

Submit

Feedback

Code review

Proposed fixes

Discuss fixes

Code/Check quality

Submit

Code review

Accept

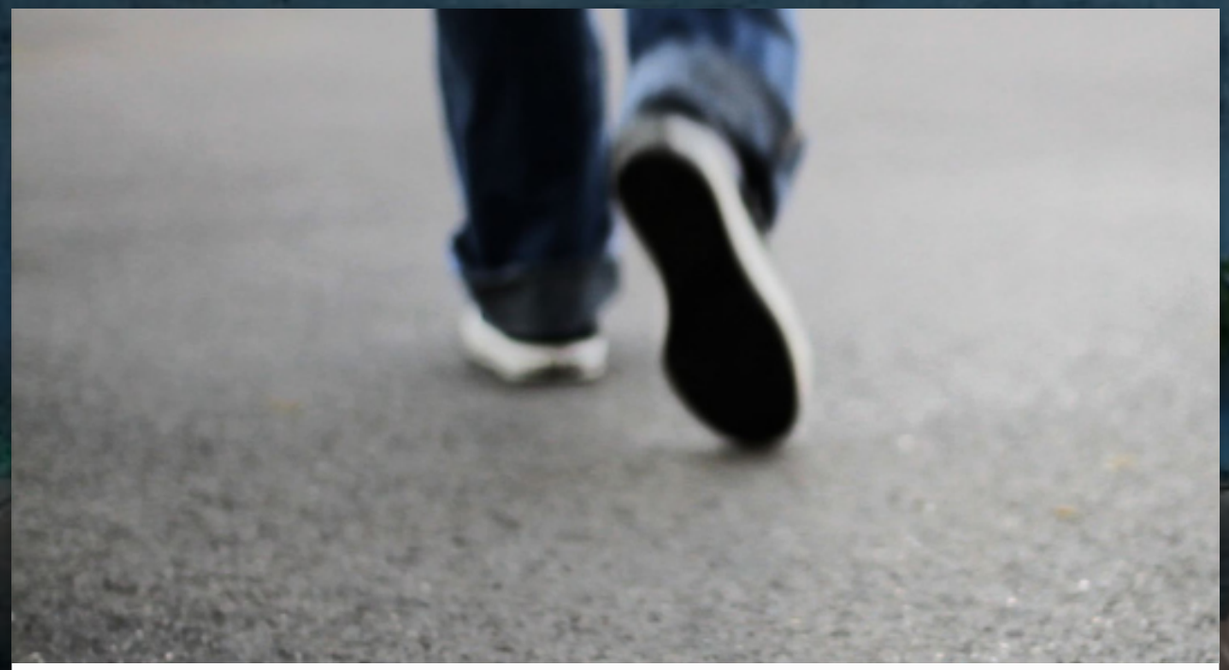
**quality**

**quality**

**lack of process**

quality

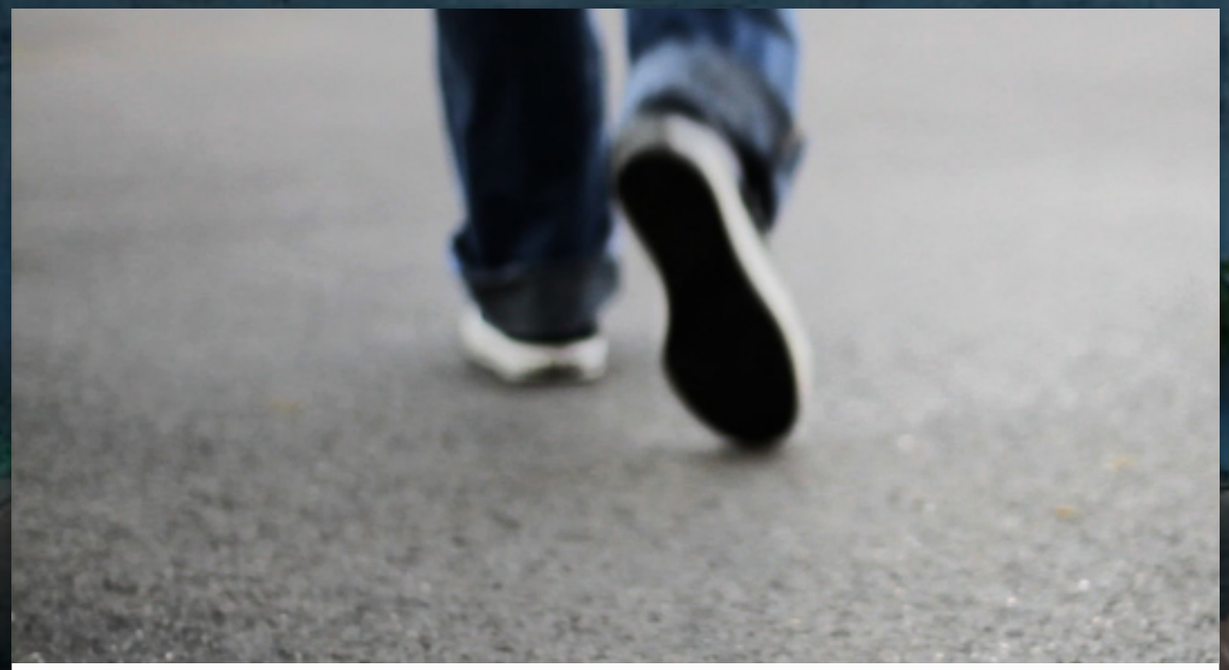
lack of process



workload and responsiveness

quality

lack of process



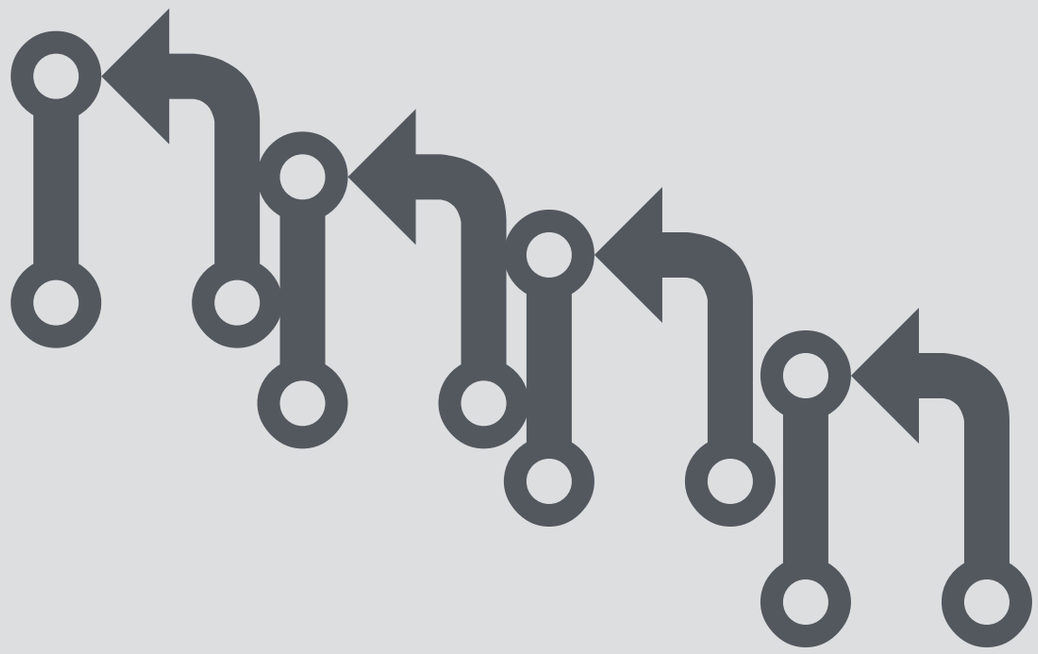
workload and responsiveness

communication

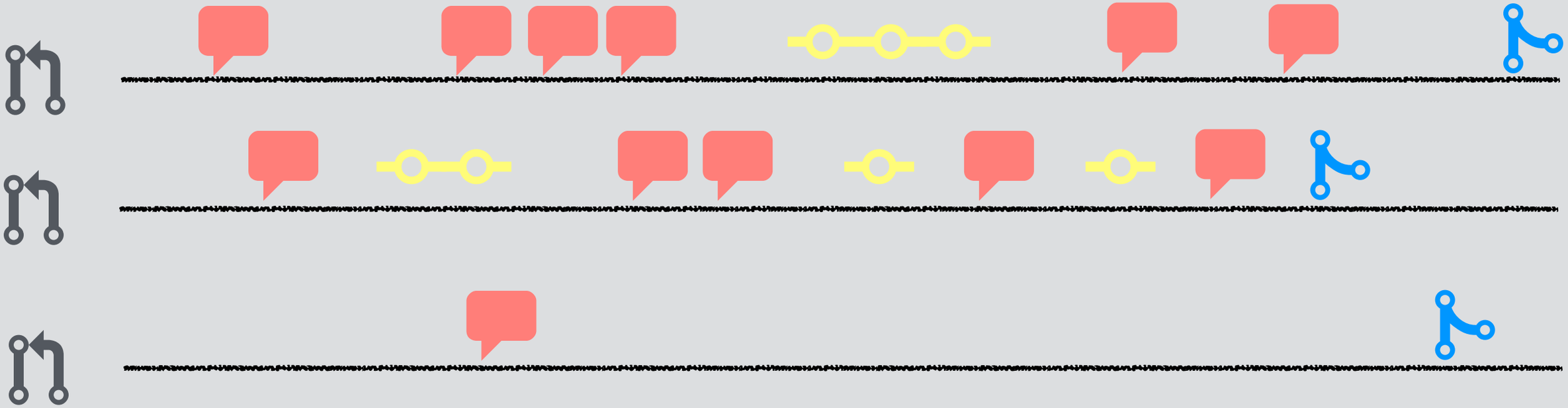
IMPORTANT



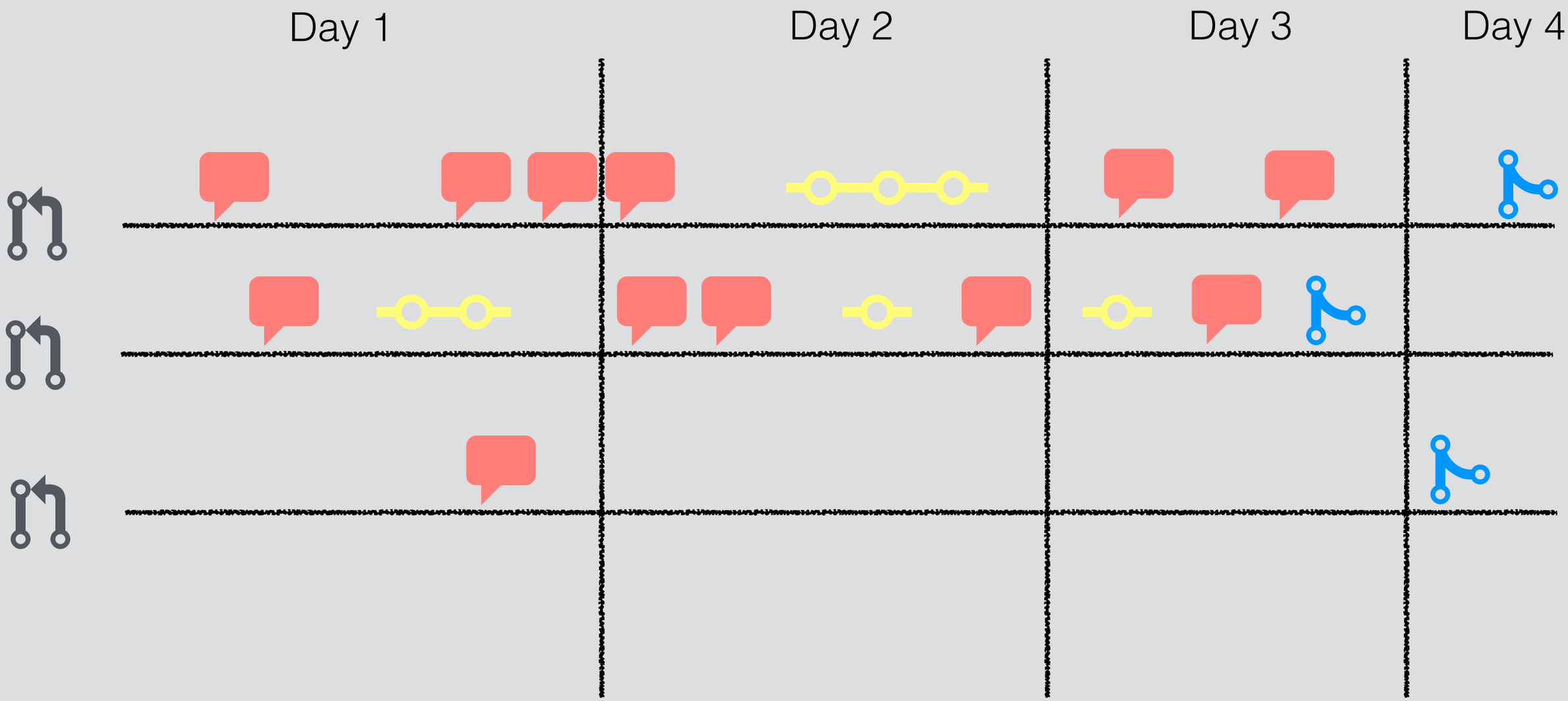
EVERYTHING ELSE



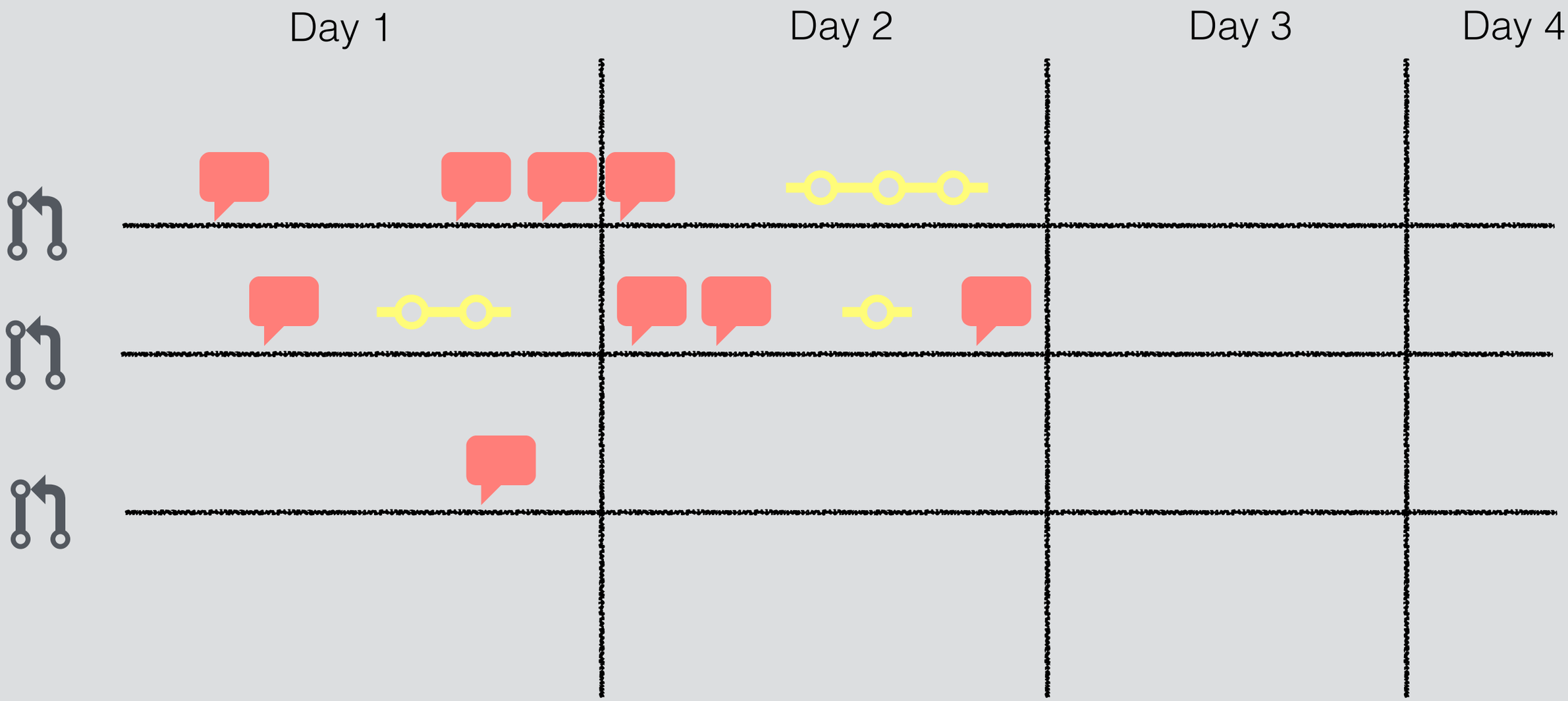
# Which  are IMPORTANT ?



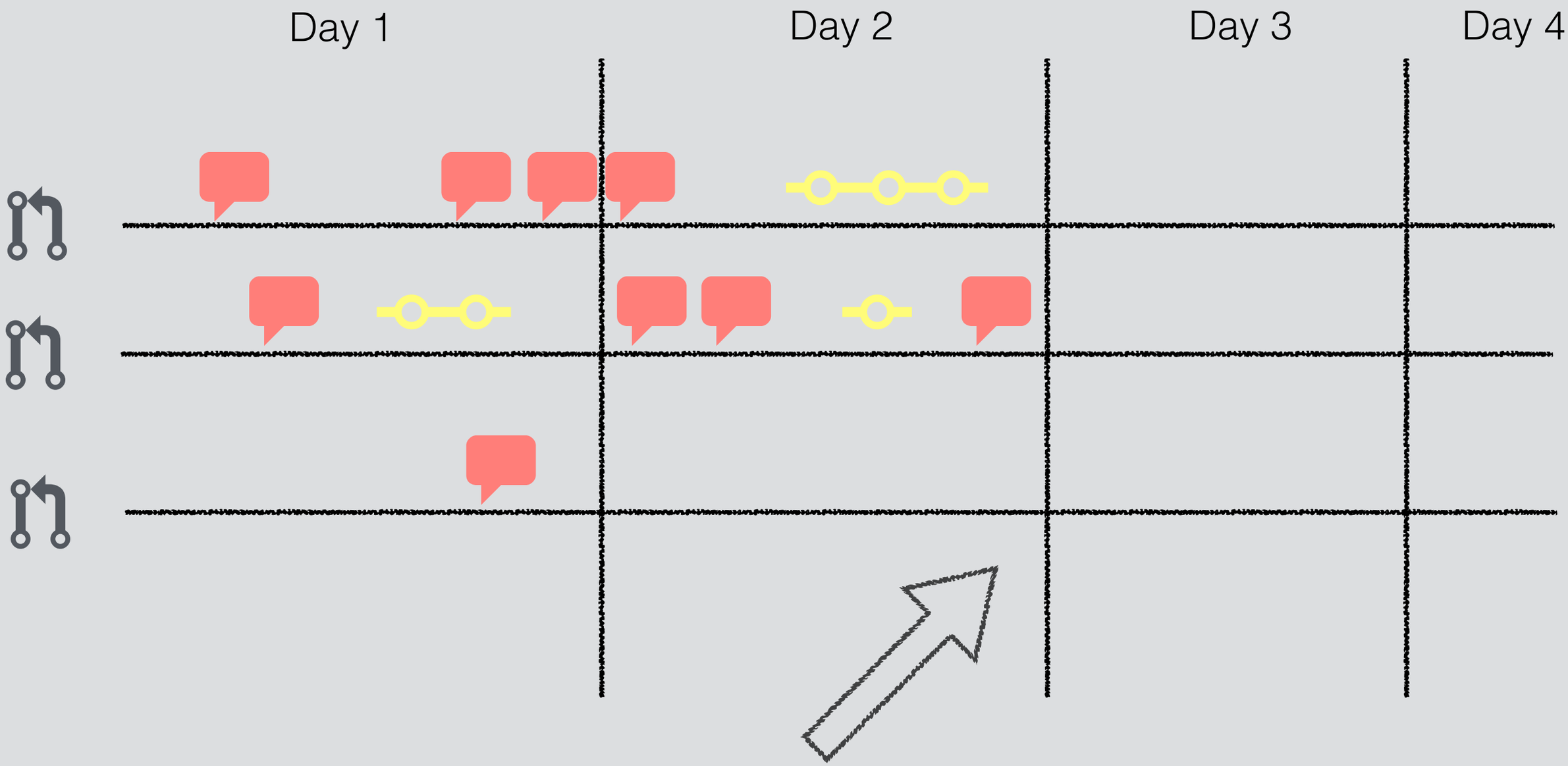
# Which  are IMPORTANT ?



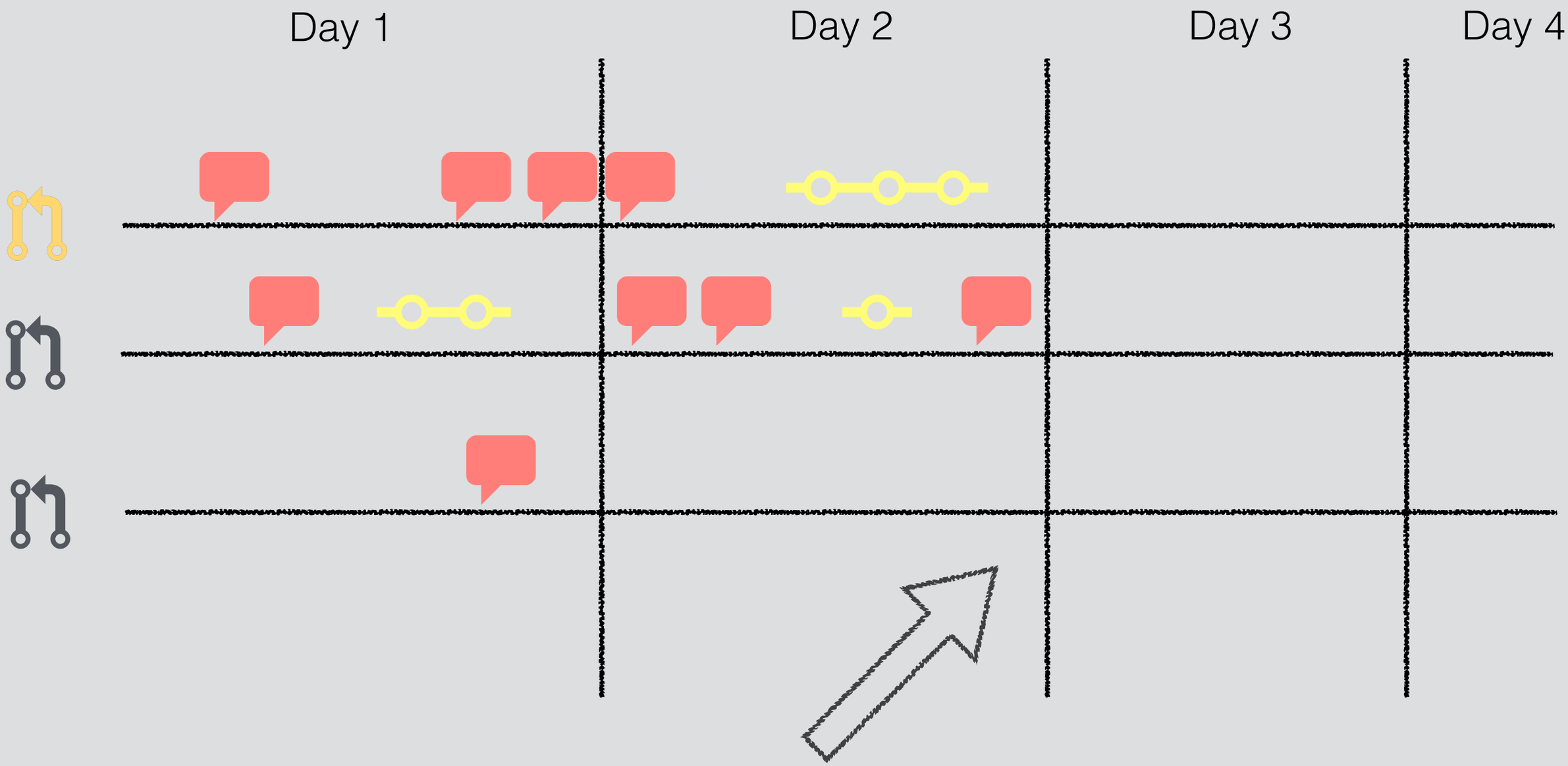
# Which  are IMPORTANT ?



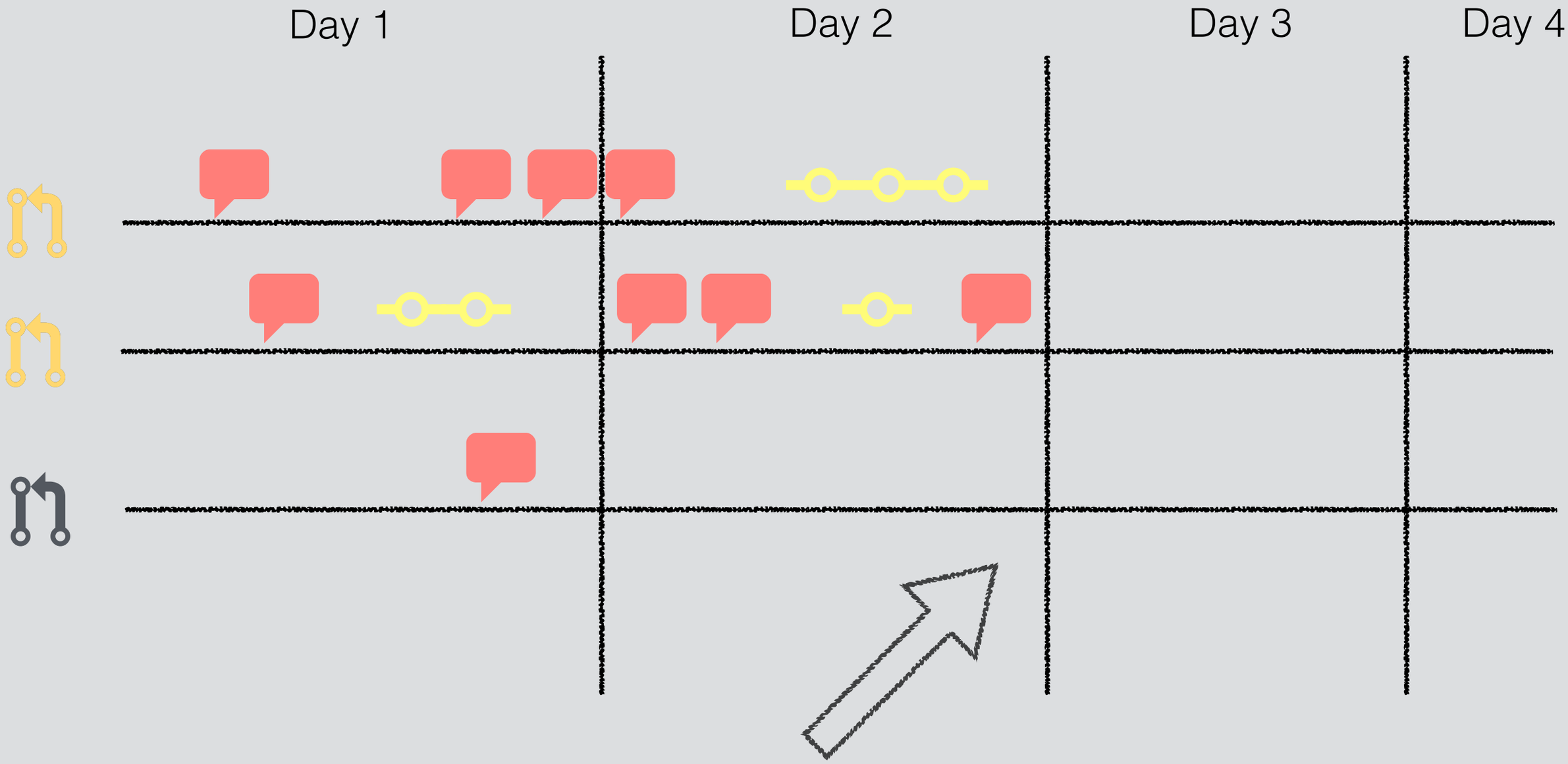
# Which  are IMPORTANT ?



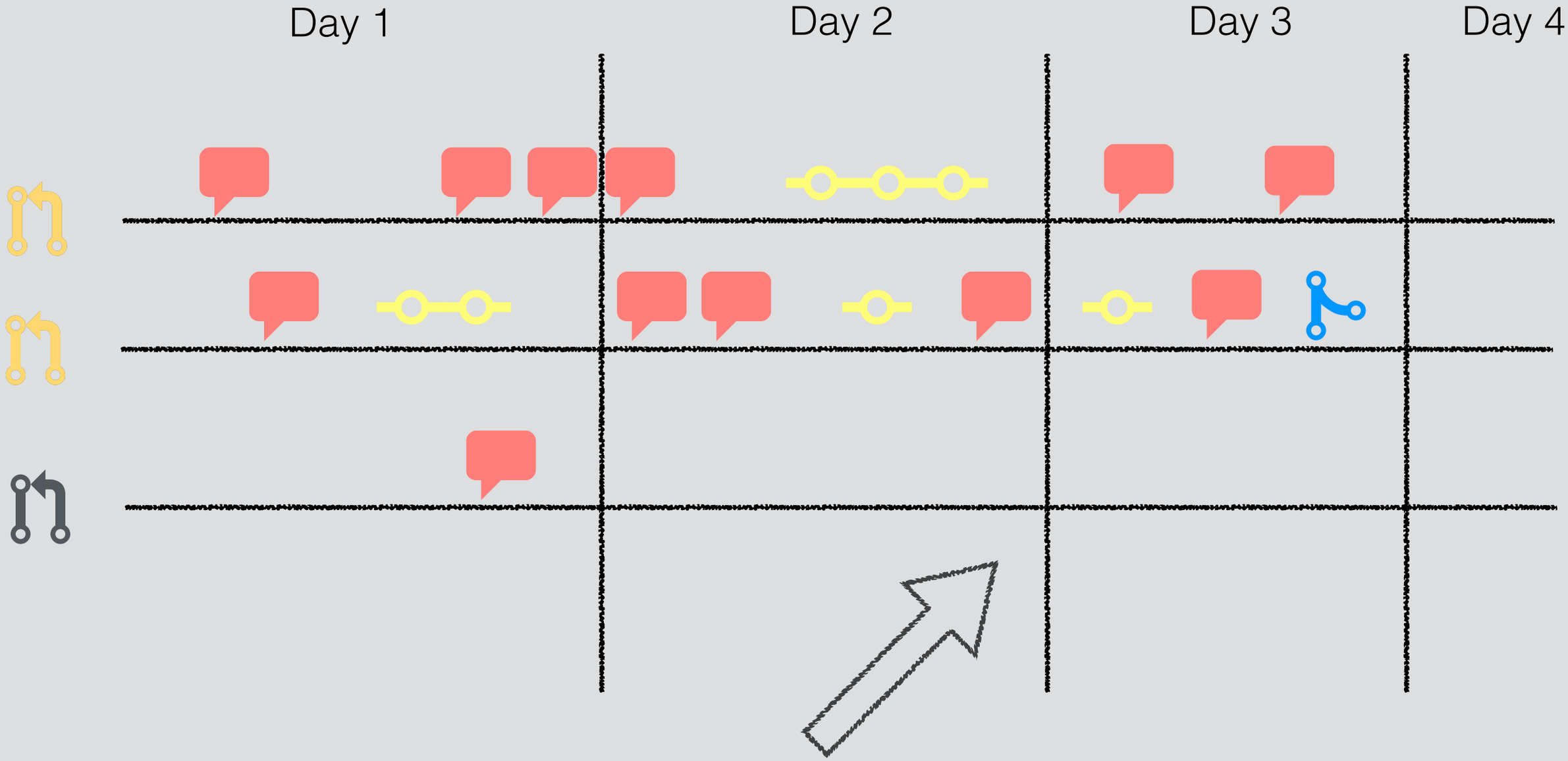
# Which  are IMPORTANT ?



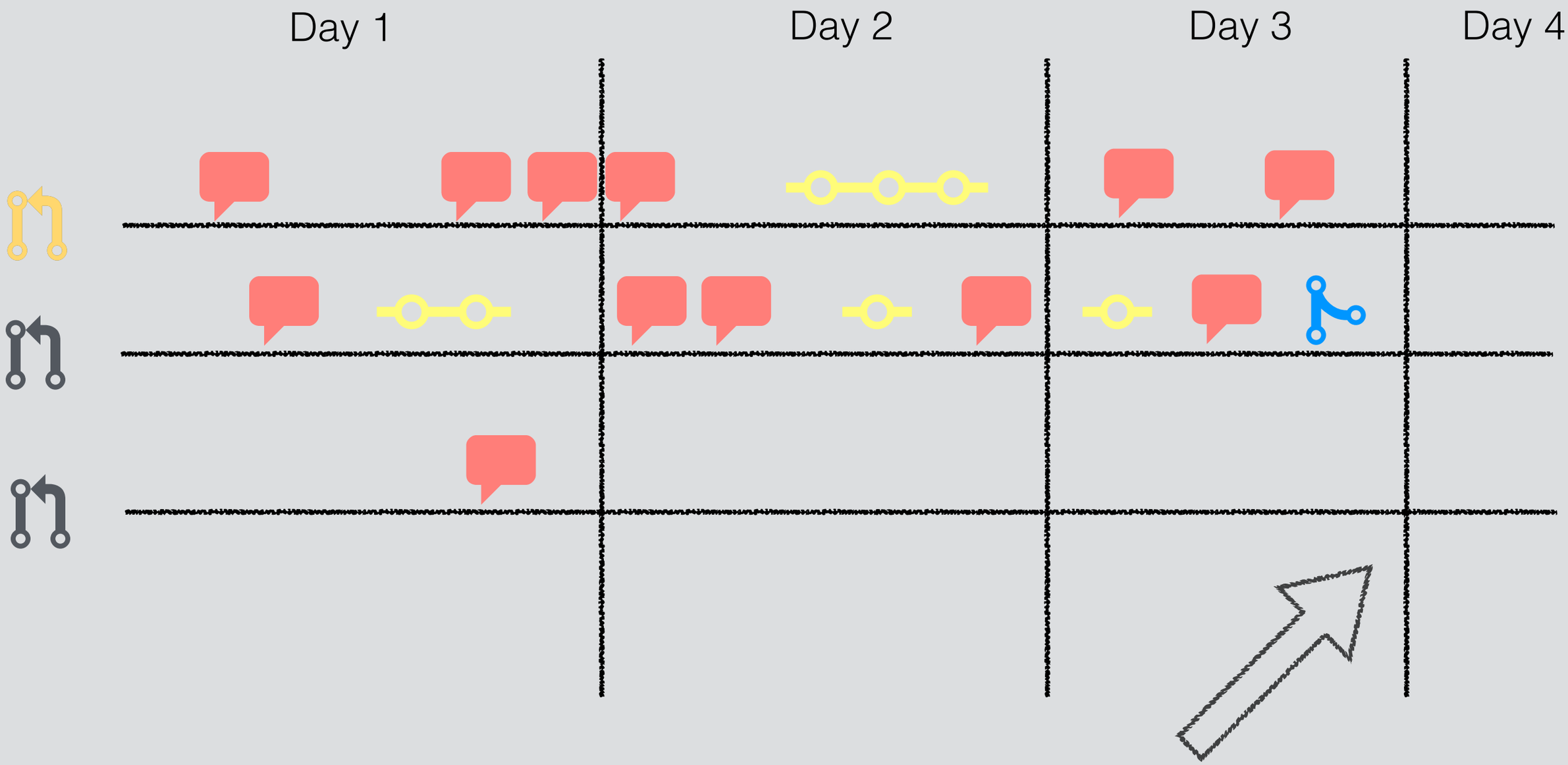
# Which  are IMPORTANT ?



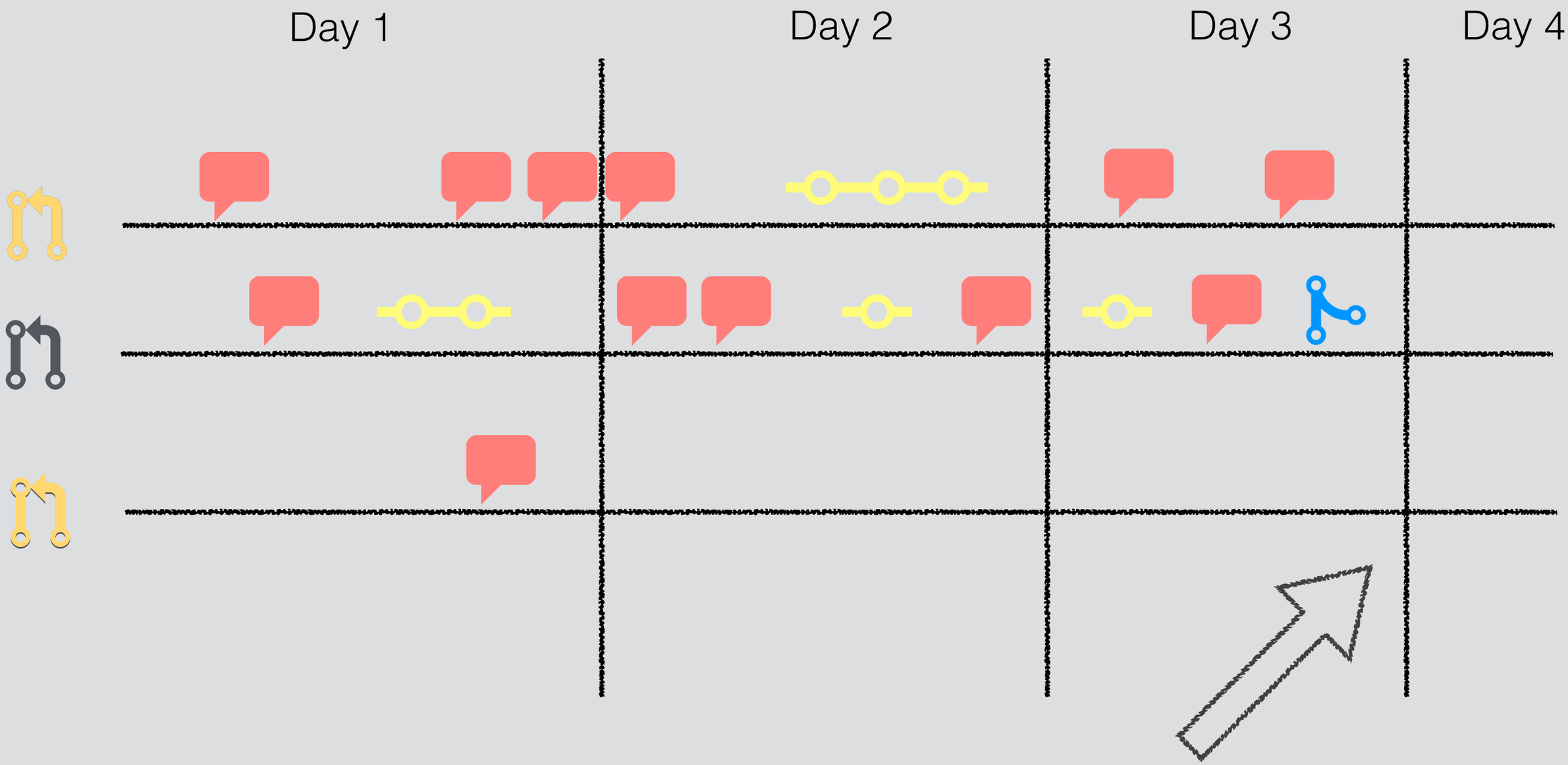
# Which ⎇ are IMPORTANT ?



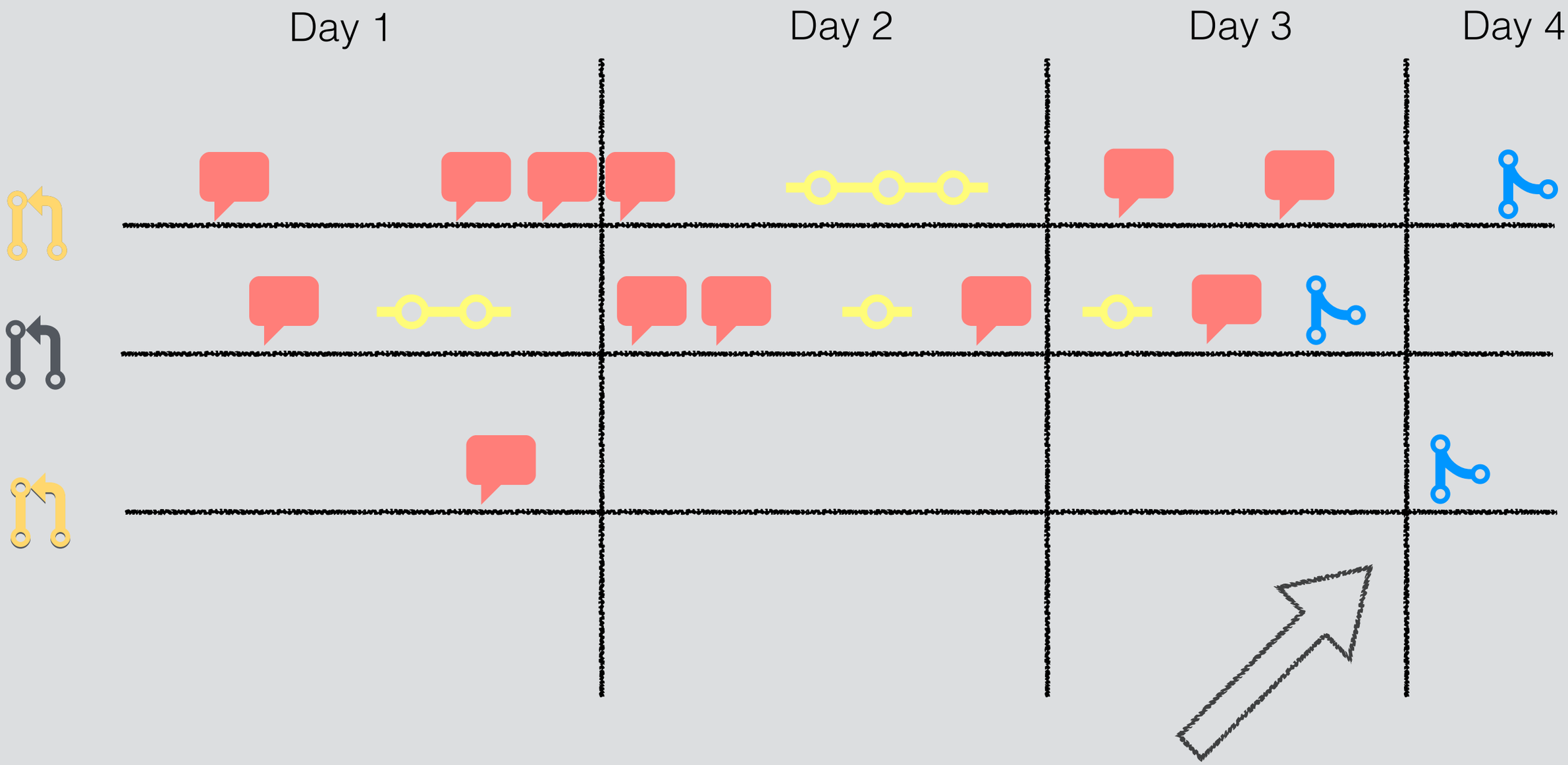
# Which  are IMPORTANT ?



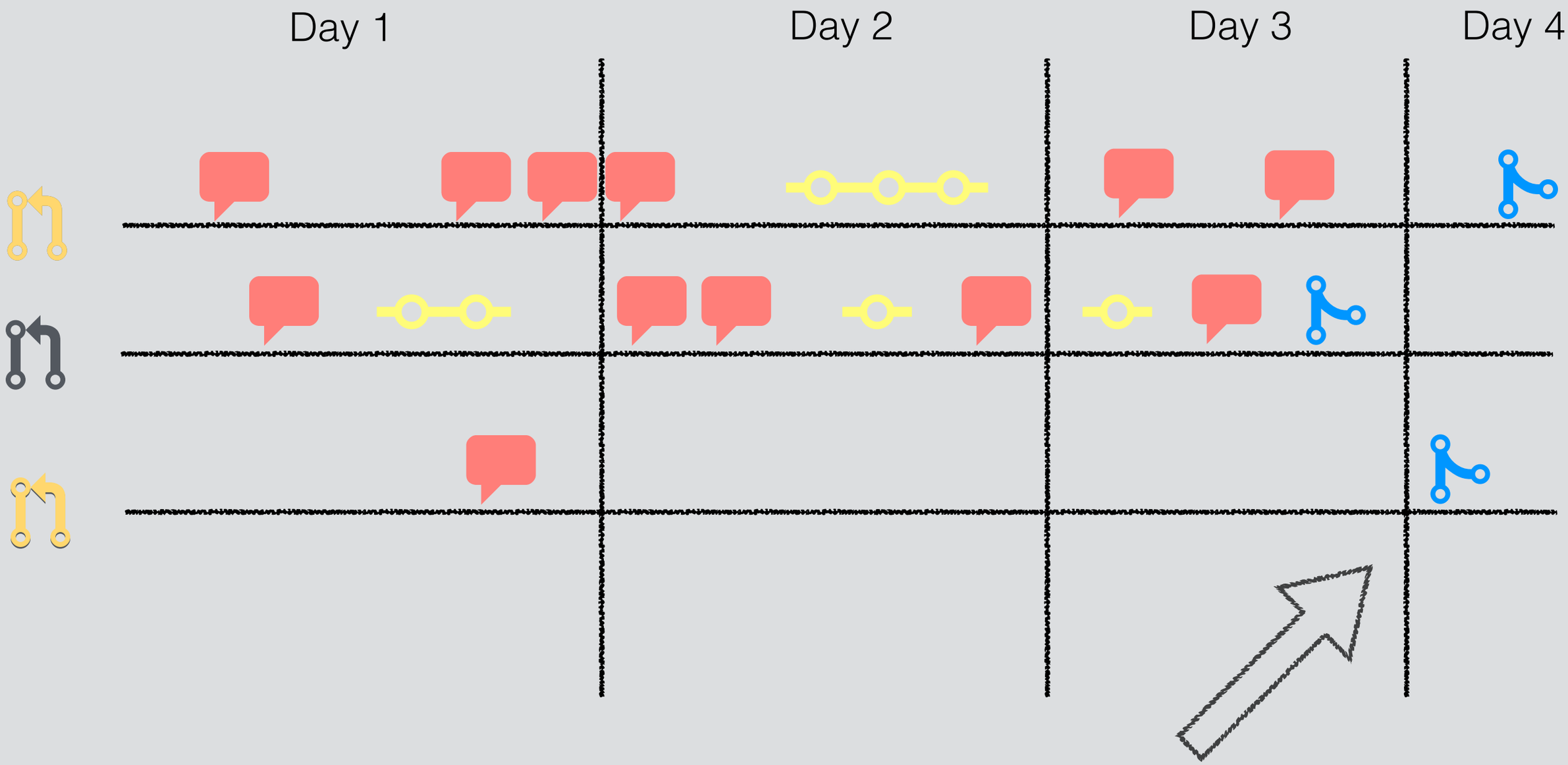
# Which  are IMPORTANT ?



# Which  are IMPORTANT ?



# Which  are IMPORTANT ?




IMPORTANT = about to be active

| | Precision | Recall | AUC | Accuracy |
|---|---|---|---|---|
| **Random Forests** | **0.66** | **0.63** | **0.89** | **0.86** |
| **Naive Bayes** | 0.34 | 0.79 | 0.75 | 0.60 |
| **Logistic regression** | 0.36 | 0.84 | 0.81 | 0.62 |

E. van der Veen, G. Gousios, and A. Zaidman, "Automatically Prioritizing Pull Requests," MSR, 2015, pp. 357–361.

| | Precision | Recall | AUC | Accuracy |
|---|---|---|---|---|
| **Random Forests** | **0.66** | **0.63** | **0.89** | **0.86** |
| **Naive Bayes** | 0.34 | 0.79 | 0.75 | 0.60 |
| **Logistic regression** | 0.36 | 0.84 | 0.81 | 0.62 |

E. van der Veen, G. Gousios, and A. Zaidman, "Automatically Prioritizing Pull Requests," MSR, 2015, pp. 357–361.

Pull Requests · Microsoft/C

https://github.com/Microsoft/ChakraCore/pulls

GitHub | This repository Search | Explore | Features | Enterprise | Pricing | Sign up | Sign in

Microsoft / ChakraCore

Watch 337 | Star 4,826 | Fork 477

Code | Issues 66 | Pull requests 9 | Wiki | Pulse | Graphs

is:pr is:open | Labels | Milestones | New pull request

9 Open | 157 Closed | Author | Labels | Milestones | Assignee | Sort

**Force -win7 on Legacy Daily Builds because the test machinery doesn't automatically detect Windows Server 2008 R2 as Windows 7.** ✕ cla-not-required Continuous Integration — 4
Processing time? SLOW Merge? TRUE
#284 opened 11 hours ago by dilijev

**Don't defer event handlers in debug/rundown mode** cla-not-required — 1
Processing time? SLOW Merge? FALSE
#282 opened 14 hours ago by pleath

**Show String.prototype.search in the debugger call stack** cla-not-required — 3
Processing time? FAST Merge? TRUE
#281 opened 15 hours ago by goyakin

**Object.prototype bug** cla-not-required — 6
Processing time? SLOW Merge? FALSE
#278 opened 4 days ago by LouisLaf

Pourquoi

**linux: build lib/common/common/Jobs.cpp** cla-not-required — 4
Processing time? SLOW Merge? TRUE
#273 opened 4 days ago by jianchun

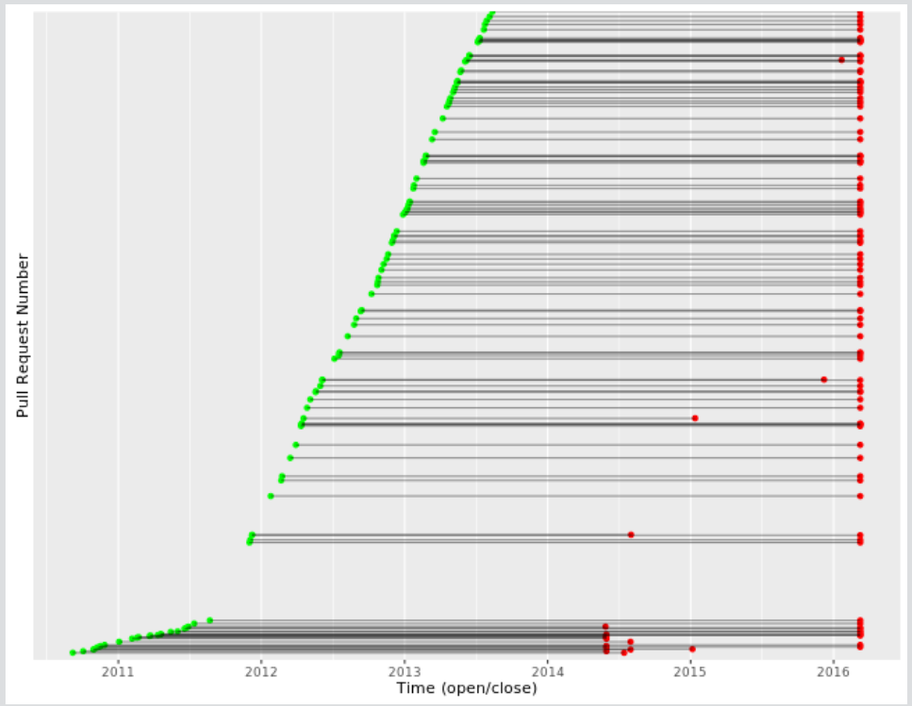
**Move rarely used pointers from FunctionBody to a dynamic structure to reduce FunctionBody size** cla-already-signed — 55

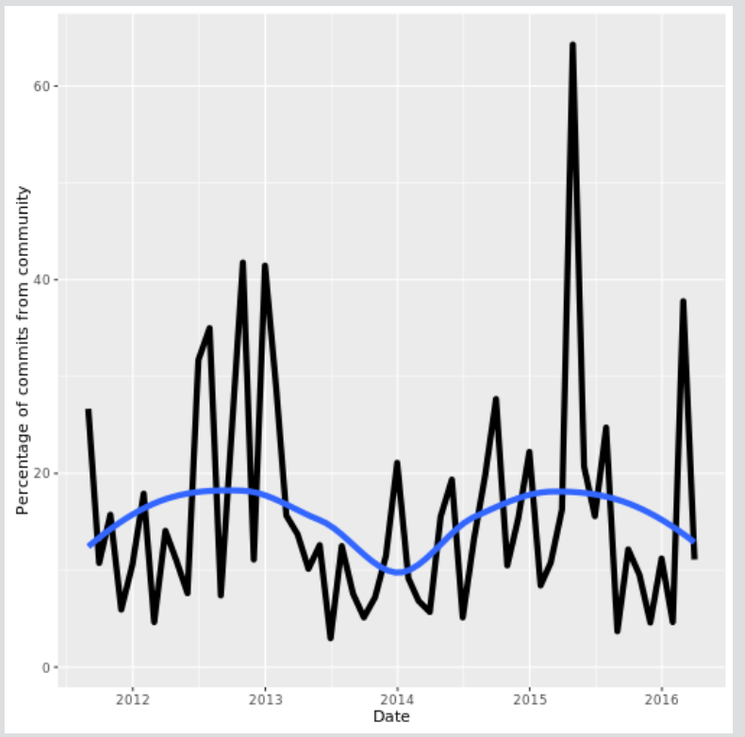
# Openess perf reports

http://ghtorrent.org/pullreq-perf

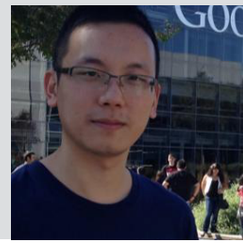
How open is your project to community contributions?

- 5k projects
- every 15 days



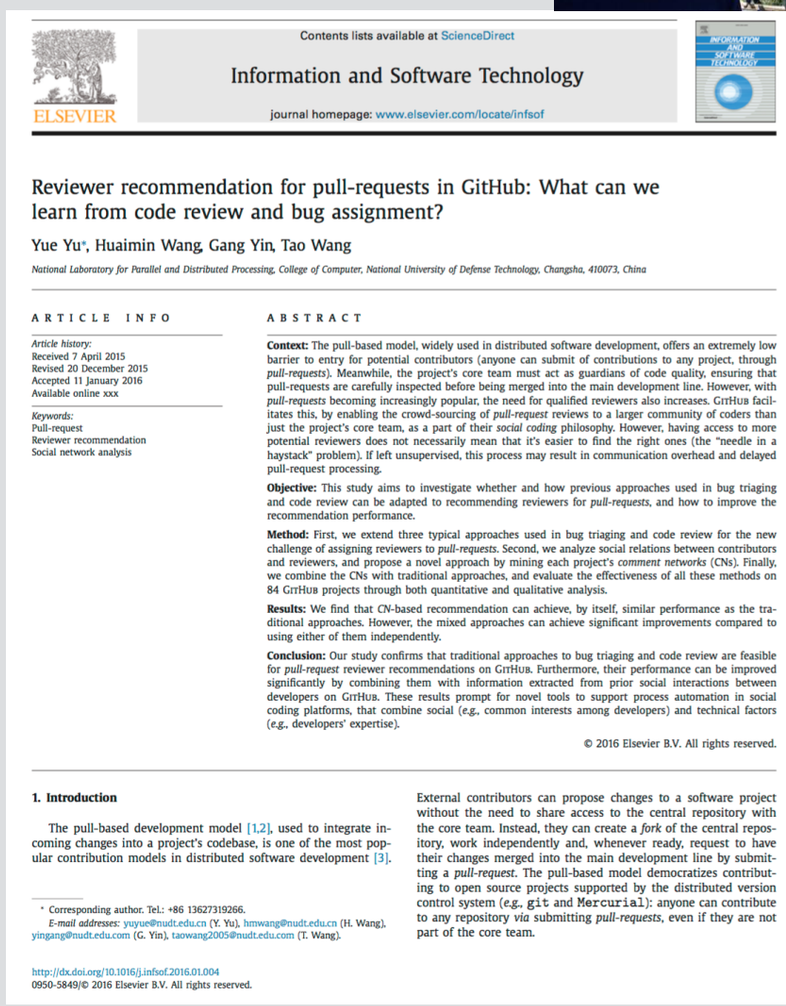




# Reviewer recommendation



Exploit *@mention* networks to propose top-3 reviewers for incoming pull requests.

Accuracy **~60%** on top-3 recommendation



Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof

Reviewer recommendation for pull-requests in GitHub: What can we learn from code review and bug assignment?

Yue Yu*, Huaimin Wang, Gang Yin, Tao Wang

*National Laboratory for Parallel and Distributed Processing, College of Computer, National University of Defense Technology, Changsha, 410073, China*

ARTICLE INFO

*Article history:*
Received 7 April 2015
Revised 20 December 2015
Accepted 11 January 2016
Available online xxx

*Keywords:*
Pull-request
Reviewer recommendation
Social network analysis

ABSTRACT

**Context:** The pull-based model, widely used in distributed software development, offers an extremely low barrier to entry for potential contributors (anyone can submit of contributions to any project, through *pull-requests*). Meanwhile, the project's core team must act as guardians of code quality, ensuring that pull-requests are carefully inspected before being merged into the main development line. However, with *pull-requests* becoming increasingly popular, the need for qualified reviewers also increases. GitHub facilitates this, by enabling the crowd-sourcing of *pull-request* reviews to a larger community of coders than just the project's core team, as a part of their *social coding* philosophy. However, having access to more potential reviewers does not necessarily mean that it's easier to find the right ones (the "needle in a haystack" problem). If left unsupervised, this process may result in communication overhead and delayed pull-request processing.

**Objective:** This study aims to investigate whether and how previous approaches used in bug triaging and code review can be adapted to recommending reviewers for *pull-requests*, and how to improve the recommendation performance.

**Method:** First, we extend three typical approaches used in bug triaging and code review for the new challenge of assigning reviewers to *pull-requests*. Second, we analyze social relations between contributors and reviewers, and propose a novel approach by mining each project's *comment networks* (CNs). Finally, we combine the CNs with traditional approaches, and evaluate the effectiveness of all these methods on 84 GitHub projects through both quantitative and qualitative analysis.

**Results:** We find that *CN*-based recommendation can achieve, by itself, similar performance as the traditional approaches. However, the mixed approaches can achieve significant improvements compared to using either of them independently.

**Conclusion:** Our study confirms that traditional approaches to bug triaging and code review are feasible for *pull-request* reviewer recommendations on GitHub. Furthermore, their performance can be improved significantly by combining them with information extracted from prior social interactions between developers on GitHub. These results prompt for novel tools to support process automation in social coding platforms, that combine social (*e.g.*, common interests among developers) and technical factors (*e.g.*, developers' expertise).



## 1. Introduction

The pull-based development model [1,2], used to integrate incoming changes into a project's codebase, is one of the most popular contribution models in distributed software development [3].

External contributors can propose changes to a software project without the need to share access to the central repository with the core team. Instead, they can create a *fork* of the central repository, work independently and, whenever ready, request to have their changes merged into the main development line by submitting a *pull-request*. The pull-based model democratizes contributing to open source projects supported by the distributed version control system (*e.g.*, `git` and `Mercurial`): anyone can contribute to any repository *via* submitting *pull-requests*, even if they are not part of the core team.

* Corresponding author. Tel.: +86 13627319266.
*E-mail addresses:* yuyue@nudt.edu.cn (Y. Yu), hmwang@nudt.edu.cn (H. Wang), yingang@nudt.edu.com (G. Yin), taowang2005@nudt.edu.com (T. Wang).

http://dx.doi.org/10.1016/j.infsof.2016.01.004
0950-5849/

Yue Yu, Huaimin Wang, Gang Yin, Tao Wang, Reviewer recommendation for pull-requests in GitHub: What can we learn from code review and bug assignment?,IST, 2016

# Automated code review

Examine how “natural” the PR code is WRT the project’s code base.

Accepted PRs are significantly similar to the project

More debated PRs are significantly less similar

## Will they like this? Evaluating Code Contributions With Language Models

Vincent J. Hellendoorn,[1] Premkumar T. Devanbu,[2] Alberto Bacchelli[1]
1: SORCERERS @ Software Engineering Research Group, Delft University of Technology, The Netherlands
2: Department of Computer Science, University of California, Davis, CA. USA

***Abstract*—Popular open-source software projects receive and review contributions from a diverse array of developers, many of whom have little to no prior involvement with the project. A recent survey reported that reviewers consider conformance to the project’s code style to be one of the top priorities when evaluating code contributions on Github. We propose to quantitatively evaluate the existence and effects of this phenomenon. To this aim we use language models, which were shown to accurately capture stylistic aspects of code. We find that rejected changesets do contain code significantly less similar to the project than accepted ones; furthermore, the less similar changesets are more likely to be subject to thorough review. Armed with these results we further investigate whether new contributors learn to conform to the project style and find that experience is positively correlated with conformance to the project’s code style.**

### I. Introduction

Code review is the manual assessment of source code by human reviewers. Most open source software (OSS) projects, which often heavily rely on contributions of disparate developers, consider code review a best practice both to foster a productive development community [13] and to ensure high code quality [6]. Nowadays code reviews are often mediated by tools (*e.g.*, [14], [15], [26], [20]), which record information that can be mined to better understand the factors influencing the code review process and the acceptability of contributions.

Prior research (in both commercial and OSS settings) has investigated meta-properties stored in code review data, such as size, number of commits, and time to review, and properties of the files that were changed (*e.g.*, [16], [22], [5]). Other research investigated review information to explore the influence of social aspects on software development in OSS communities [5], [11], in particular on the actions of a reviewer when faced with a new contribution [21]. Rigby *et al.* used code review recorded data to triangulate an investigation on the influence of personal aspects of the reviewer, such as experience and efficiency on the code review process [29]. This research has led to valuable insights on how the reviewer’s attitude towards the (potentially unknown) contributor affects the process and outcome of code review.

Little is known, yet, of what properties of the *submitted code* influence the review of a changeset and its acceptability. The earliest and most significant insights in this area are those by Gousios *et al.*, who conducted a qualitative study with reviewers of proposed code changes in GitHub [17]. Gousios *et al.* reported that integrators consider style conformance the top factor when evaluating quality of submitted code. Code quality and code style were reported as the top two factors influencing the decision to accept a contribution.

Our goal is to extend these qualitative insights by quantitatively evaluating the influence of stylistic properties of submitted code on both the process and the outcome of code review. To achieve this, we make use of *language models* constructed on source code [18], which were proven to be well-suited to capture stylistic properties of code in the context of a project [1]. Hence, we use this tool to measure how well submitted code *fits in* with the project’s code as a whole and analyze how this influences code review.

We conduct our evaluation on projects that use GitHub, the popular social coding platform at the basis of the study by Gousios *et al.* GitHub offers built-in capability for code review by implementing the pull-based development model [16]. In this model, contributors do not have access to the main repository, rather they fork it, make their changes independently, and create a pull request with their proposed changes to be merged in the main repository. The project’s core team is then responsible for reviewing and (if found acceptable) eventually merging the changes on the main development line. We consider 22 popular and independent projects on GitHub and analyze a total of 1.4M lines of submitted code, across 6,000 pull requests.

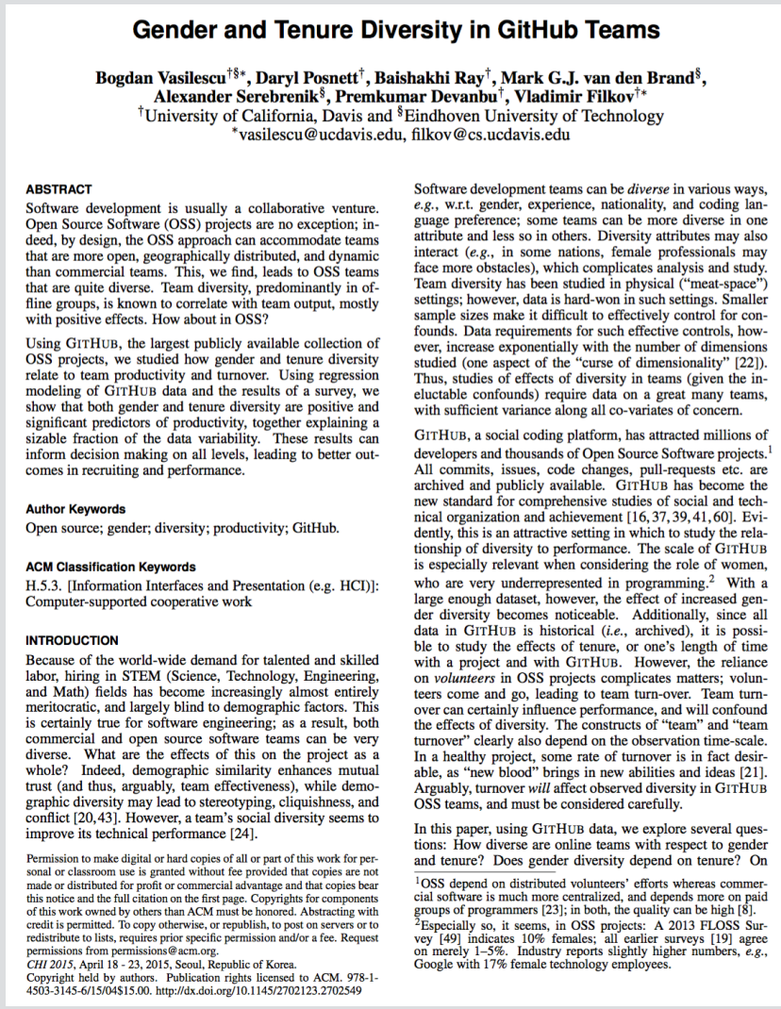
The results of our evaluation show that accepted changesets are significantly more similar to the project at the time of submission than rejected pull requests, supporting that conformance to the code style is a factor that influences code review. We further show that contributions that were subject to more extensive reviews, such as debate regarding a changeset, were substantially less similar to the project’s code style. Further investigation supports our finding that highly dissimilar contributions are isolated by project maintainers and receive substantially different treatment during code review. Finally, we show that contributions by novel contributors show a substantial increase in similarity to the project’s code as the contributor gains experience.

**Structure of the paper.** In Section II, we describe the technical background, particularly in the field of natural language processing (NLP) related to language models applied to source code. In Section III, we introduce our research questions and detail the research method we follow. We present our findings in Section IV, and discuss them in Section V. In Section VI we identify a number of threats to the validity of our study. We conclude in Section VII.

Vincent J. Hellendoorn, Premkumar T. Devanbu, and Alberto Bacchelli. 2015. Will they like this?: evaluating code contributions with language models. MSR ’15, pp157-167

# Gender and Tenure

*"Our study suggests that, overall, when forming or recruiting a software team, increased gender and tenure diversity are associated with greater productivity."*



**Gender and Tenure Diversity in GitHub Teams**

**Bogdan Vasilescu[†§*], Daryl Posnett[†], Baishakhi Ray[†], Mark G.J. van den Brand[§], Alexander Serebrenik[§], Premkumar Devanbu[†], Vladimir Filkov[†*]**
[†]University of California, Davis and [§]Eindhoven University of Technology
[*]vasilescu@ucdavis.edu, filkov@cs.ucdavis.edu

**ABSTRACT**

Software development is usually a collaborative venture. Open Source Software (OSS) projects are no exception; indeed, by design, the OSS approach can accommodate teams that are more open, geographically distributed, and dynamic than commercial teams. This, we find, leads to OSS teams that are quite diverse. Team diversity, predominantly in offline groups, is known to correlate with team output, mostly with positive effects. How about in OSS?

Using GitHub, the largest publicly available collection of OSS projects, we studied how gender and tenure diversity relate to team productivity and turnover. Using regression modeling of GitHub data and the results of a survey, we show that both gender and tenure diversity are positive and significant predictors of productivity, together explaining a sizable fraction of the data variability. These results can inform decision making on all levels, leading to better outcomes in recruiting and performance.

**Author Keywords**

Open source; gender; diversity; productivity; GitHub.

**ACM Classification Keywords**

H.5.3. [Information Interfaces and Presentation (e.g. HCI)]: Computer-supported cooperative work

**INTRODUCTION**

Because of the world-wide demand for talented and skilled labor, hiring in STEM (Science, Technology, Engineering, and Math) fields has become increasingly almost entirely meritocratic, and largely blind to demographic factors. This is certainly true for software engineering; as a result, both commercial and open source software teams can be very diverse. What are the effects of this on the project as a whole? Indeed, demographic similarity enhances mutual trust (and thus, arguably, team effectiveness), while demographic diversity may lead to stereotyping, cliquishness, and conflict [20,43]. However, a team's social diversity seems to improve its technical performance [24].


*CHI 2015*, April 18 - 23, 2015, Seoul, Republic of Korea.
 978-1-4503-3145-6/15/04$15.00. http://dx.doi.org/10.1145/2702123.2702549

Software development teams can be *diverse* in various ways, *e.g.*, w.r.t. gender, experience, nationality, and coding language preference; some teams can be more diverse in one attribute and less so in others. Diversity attributes may also interact (*e.g.*, in some nations, female professionals may face more obstacles), which complicates analysis and study. Team diversity has been studied in physical ("meat-space") settings; however, data is hard-won in such settings. Smaller sample sizes make it difficult to effectively control for confounds. Data requirements for such effective controls, however, increase exponentially with the number of dimensions studied (one aspect of the "curse of dimensionality" [22]). Thus, studies of effects of diversity in teams (given the ineluctable confounds) require data on a great many teams, with sufficient variance along all co-variates of concern.

GitHub, a social coding platform, has attracted millions of developers and thousands of Open Source Software projects.[1] All commits, issues, code changes, pull-requests etc. are archived and publicly available. GitHub has become the new standard for comprehensive studies of social and technical organization and achievement [16, 37, 39, 41, 60]. Evidently, this is an attractive setting in which to study the relationship of diversity to performance. The scale of GitHub is especially relevant when considering the role of women, who are very underrepresented in programming.[2] With a large enough dataset, however, the effect of increased gender diversity becomes noticeable. Additionally, since all data in GitHub is historical (*i.e.*, archived), it is possible to study the effects of tenure, or one's length of time with a project and with GitHub. However, the reliance on *volunteers* in OSS projects complicates matters; volunteers come and go, leading to team turn-over. Team turnover can certainly influence performance, and will confound the effects of diversity. The constructs of "team" and "team turnover" clearly also depend on the observation time-scale. In a healthy project, some rate of turnover is in fact desirable, as "new blood" brings in new abilities and ideas [21]. Arguably, turnover *will* affect observed diversity in GitHub OSS teams, and must be considered carefully.

In this paper, using GitHub data, we explore several questions: How diverse are online teams with respect to gender and tenure? Does gender diversity depend on tenure? On

[1]OSS depend on distributed volunteers' efforts whereas commercial software is much more centralized, and depends more on paid groups of programmers [23]; in both, the quality can be high [8].
[2]Especially so, it seems, in OSS projects: A 2013 FLOSS Survey [49] indicates 10% females; all earlier surveys [19] agree on merely 1–5%. Industry reports slightly higher numbers, *e.g.*, Google with 17% female technology employees.

Vasilescu, Posnett, Ray, van den Brand, Serebrenik, Devanbu, and Filkov. Gender and Tenure Diversity in GitHub Teams. CHI. 2015

# Geographical equality

## Traces of bias on contributions from certain countries

- Contributors perceive it
- Integrator's do not

**All Contributors are Equal; Some Contributors are More Equal than Others**


Ayushi Rastogi
IIIT-Delhi
Delhi, India
ayushir@iiitd.ac.in

Nachiappan Nagappan
Microsoft Research Labs
Redmond, USA
nachin@microsoft.com

Georgios Gousios
Radboud University Nijmegen
The Netherlands
g.gousios@cs.ru.nl



**ABSTRACT**

Open source development has often been considered to be a level playing field for all developers. But there has been little work to investigate if bias plays any role in getting contributions accepted. The work presented in this study tries to understand the influence of geographical location on the evaluation of pull requests in GitHub - one of the primary open source development platforms.

Using a mixed-methods approach that analyzes 70,000+ pull requests and 2,500+ survey responses, we find that geographical location explains statistically significant differences in pull request acceptance decisions. Compared to the United States, submitters from United Kingdom (22%), Canada (25%), Japan (40%), Netherlands (43%), and Switzerland (58%) have higher chances of getting their pull requests accepted. However, submitters from Germany (15%), Brazil (17%), China (24%), and Italy (19%) have lower chances of getting their pull requests accepted compared to the United States. The probability of pull request acceptance decisions increase by 19% when the submitter and integrator are from the same geographical location. Survey responses from submitters indicate the perception of bias is strong in Brazil and Italy matching our results. Also, 8 out of every 10 integrators feel that it is easy to work with submitters from the same geographical location.



**Categories and Subject Descriptors**

D.2.9 [**Software Engineering**]: Management—*Programming teams*

**General Terms**

Management

**Keywords**

Empirical software engineering, software process, empirical studies




**1. INTRODUCTION**

Biases have been seen to deter meritocracy in offline work groups [6][41]. For years, visible characteristics have been used to differentiate people in all spheres of life ranging from sports [31] to health care [23] to job applications [5]. These biases have started making their presence felt in online environments too [14][13].

Open source software evolves through geographically distributed developers collaborate online on a wide range of projects, building on meritocracy []. Evolution of OSS gave rise to terms like 'code is king' [4]. In recent years, social work environments like GitHub [3], Bitbucket [1], etc. have concentrated large crowds of developers. These platforms provided transparency and access to developers' profiles including their contributions and social skills []. Developers started using this information to evaluate other developers based on their social skills in addition to technical skills [36].

Earlier studies on perceived differences in values and norms pointed towards the likelihood of engaging in stereotyping, cliquishness, and conflicts [22]. Our goal with this work is to understand whether geographical location of developers influences the way their contributions are evaluated. We choose to study the influence of geographical location on the evaluation of contributions for the following reasons: 1) observed impact on work-related decisions in offline groups [6][13], and 2) a reasonable degree of visibility of geographical location in social work environments [37]. Through this study, our aim is to investigate the presence of bias and bridge gaps in perceptions, if any.

To measure the influence of geographical location, we leverage GitHub - the largest, most popular online collaborative coding platform. We examine GitHub data for work acceptance related decisions, as observed in pull-based development model – one of the most popular models for collaboration with 45% of collaboratively developed repositories on GitHub. The fundamental research question we try to answer is:

> Does the geographical location of the submitter influence pull request acceptance decisions?

One of the key challenges in conducting this study is to identify the presence of bias when developers themselves might not be aware of it. Even when developers are aware of their biases, it is hard to make developers accept it. So, we use a mixed-methods approach. We combine observations from 70,000+ pull requests and 2,500+ survey responses - one of the largest response on open source projects[19], to

A. Rastogi, N. Nagappan, and G. Gousios, "All contributors are equal; some contributors are more equal than others,", TR, 2016

# Gender & Contributions

When gender is identifiable: women rejected more often

When gender is *not* identifiable: women accepted more often

PeerJ Preprints NOT PEER-REVIEWED

Gender Bias in Open Source: Pull Request Acceptance of Women Versus Men

Josh Terrell[1], Andrew Kofink[2], Justin Middleton[2], Clarissa Rainear[2], Emerson Murphy-Hill[2*], Chris Parnin[2]

[1]Department of Computer Science, Cal Poly, San Luis Obispo, USA
[2]Department of Computer Science, North Carolina State University, USA

*To whom correspondence should be addressed; E-mail: emerson@csc.ncsu.edu

**Biases against women in the workplace have been documented in a variety of studies. This paper presents the largest study to date on gender bias, where we compare acceptance rates of contributions from men versus women in an open source software community. Surprisingly, our results show that women's contributions tend to be accepted *more* often than men's. However, when a woman's gender is identifiable, they are rejected more often. Our results suggest that although women on GitHub may be more competent overall, bias against them exists nonetheless.**

Terrell J, Kofink A, Middleton J, Rainear C, Murphy-Hill E, Parnin C. (2016) Gender bias in open source: Pull request acceptance of women versus men. *PeerJ PrePrints* 4:e1733v1

# The #issue32 incident

# Request for deletion #32

Edit | New issue

**Closed** **steipete** opened this issue on Feb 18 · 80 comments

**steipete** commented on Feb 18

Hi,

I'm getting a ton of emails for surveys/research around software, and it's really annoying. I recognize the scientific effort and initially helped many people here, however after so many requests, I feel that I did my part and would like to opt out. Can you remove all of my data/emails from this giant data blob?

Thank you.

**futuretap** commented on Feb 18

Please remove my data, too.

**slang800** commented on Feb 26

👎 - There are a few reasons why this is a bad idea:

- Removal from the GHTorrent dataset won't get rid of your data. Both of your email addresses are still publicly available on GitHub. **@steipete** has his in his profile (see below), and while **@futuretap** doesn't, it's still in every single commit (here's the first commit I could find, for example).

steipete

PSPDFKit GmbH

Vienna, Austria

steipete@gmail.com

**Labels**

None yet

**Milestone**

No milestone

**Assignee**

No one—assign yourself

**Notifications**

Unsubscribe

You're receiving notifications because you modified the open/close state.

**25 participants**

and others

**Unlock conversation**

# Request for deletion #32

Edit | New issue

**Closed** **steipete** opened this issue on Feb 18 · 80 comments

**steipete** commented on Feb 18

Hi,

I'm getting a ton of emails for surveys/research around software, and it's really annoying. I recognize the scientific effort and initially helped many people here, however after so many requests, I feel that I did my part and would like to opt out. Can you remove all of my data/emails from this giant data blob?

Thank you.

Labels

None yet

Milestone

**steipete** commented on Feb 18

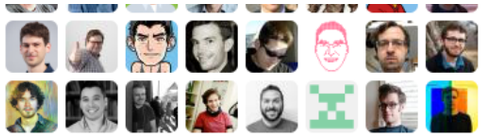
- Removal from the GHTorrent dataset won't get rid of your data. Both of your email addresses are still publicly available on GitHub. **@steipete** has his in his profile (see below), and while **@futuretap** doesn't, it's still in every single commit (here's the first commit I could find, for example).

steipete

PSPDFKit GmbH

Vienna, Austria

steipete@gmail.com



and others

Unlock conversation

# I am not a lawyer!

- Other commenters are no lawyers either
- The law is complicated and open to interpretation

# Two important issues

- *Copyright:* Who owns the data?

- *Privacy:* How does GHTorrent protect users from personal data misuse?

# Copyright — General terms

- For *original content*, the *publisher* maintains full copyright by default
  - Licenses *restrict* the effect of copyright
- Events (e.g. the fact that an issue comment was created) are *not* copyrightable, but their content may be

# Copyright — GitHub's POV

- GitHub: *We claim no intellectual property rights over the material you provide to the Service.* (TOS F.1)

- Structure of API responses is GitHub's IP

- Several fields in API responses may contain copyrighted material

# Copyright situation example


```
{
    "id": "4141500869",
    "type": "IssueCommentEvent",
    "actor": {},
    "repo": {},
    "payload": {
      "action": "created",
      "issue": {
        "id": 158442053,
        "number": 138,
        "title": "Issue in CopyrightedProjectName",
        "user": {},
        "labels": [],
        "state": "closed",
        "body": "Added data holding classes and a
map manager. Will add a system soon"
      },
      "comment": {
        "created_at": "2016-06-14T05:51:16Z",
        "updated_at": "2016-06-14T05:51:16Z",
        "body": "continuing in #141 \r\n"
      }
    }
}
```

# Copyright situation example


```
{
    "id": "4141500869",
    "type": "IssueCommentEvent",
    "actor": {},
    "repo": {},
    "payload": {
      "action": "created",
      "issue": {
        "id": 158442053,
        "number": 138,
        "title": "Issue in CopyrightedProjectName",
        "user": {},
        "labels": [],
        "state": "closed",
        "body": "Added data holding classes and a
map manager. Will add a system soon"
      },
      "comment": {
        "created_at": "2016-06-14T05:51:16Z",
        "updated_at": "2016-06-14T05:51:16Z",
        "body": "continuing in #141 \r\n"
      }
    }
}
```


©Issue initiator

# Copyright situation example


```
{
    "id": "4141500869",
    "type": "IssueCommentEvent",
    "actor": {},
    "repo": {},
    "payload": {
      "action": "created",
      "issue": {
        "id": 158442053,
        "number": 138,
        "title": "Issue in CopyrightedProjectName",
        "user": {},
        "labels": [],
        "state": "closed",
        "body": "Added data holding classes and a
map manager. Will add a system soon"
      },
      "comment": {
        "created_at": "2016-06-14T05:51:16Z",
        "updated_at": "2016-06-14T05:51:16Z",
        "body": "continuing in #141 \r\n"
      }
    }
}
```


©Issue initiator

©Issue commenter

# Copyright situation example


```
{
    "id": "4141500869",
    "type": "IssueCommentEvent",
    "actor": {},
    "repo": {},
    "payload": {
      "action": "created",
      "issue": {
        "id": 158442053,
        "number": 138,
        "title": "Issue in CopyrightedProjectName",
        "user": {},
        "labels": [],
        "state": "closed",
        "body": "Added data holding classes and a
map manager. Will add a system soon"
      },
      "comment": {
        "created_at": "2016-06-14T05:51:16Z",
        "updated_at": "2016-06-14T05:51:16Z",
        "body": "continuing in #141 \r\n"
      }
    }
}
```


©Project Name

©Issue initiator

©Issue commenter

# Copyright situation example


```
{
    "id": "4141500869",
    "type": "IssueCommentEvent",
    "actor": {},
    "repo": {},
    "payload": {
      "action": "created",
      "issue": {
        "id": 158442053,
        "number": 138,
        "title": "Issue in CopyrightedProjectName",
        "user": {},
        "labels": [],
        "state": "closed",
        "body": "Added data holding classes and a
map manager. Will add a system soon"
      },
      "comment": {
        "created_at": "2016-06-14T05:51:16Z",
        "updated_at": "2016-06-14T05:51:16Z",
        "body": "continuing in #141 \r\n"
      }
    }
}
```


©GitHub

©Project Name

©Issue initiator

©Issue commenter

**Privacy** is the ability of an individual or group to seclude themselves, or information about themselves, and thereby express themselves selectively.

# Privacy provisions — EU

- *Personal data* identify a person uniquely
  - *Facts* are not personal data
- GHTorrent processes personal data, therefore is a *controller*
- Controllers must
  - get consent for processing (except in the case of *legitimate interest*)
  - include mechanisms for opting out

# Privacy provisions — USA

- No single law/directive
- Consent only required for specific types of data storage (e.g. social security numbers)
- Offering an opting out mechanism

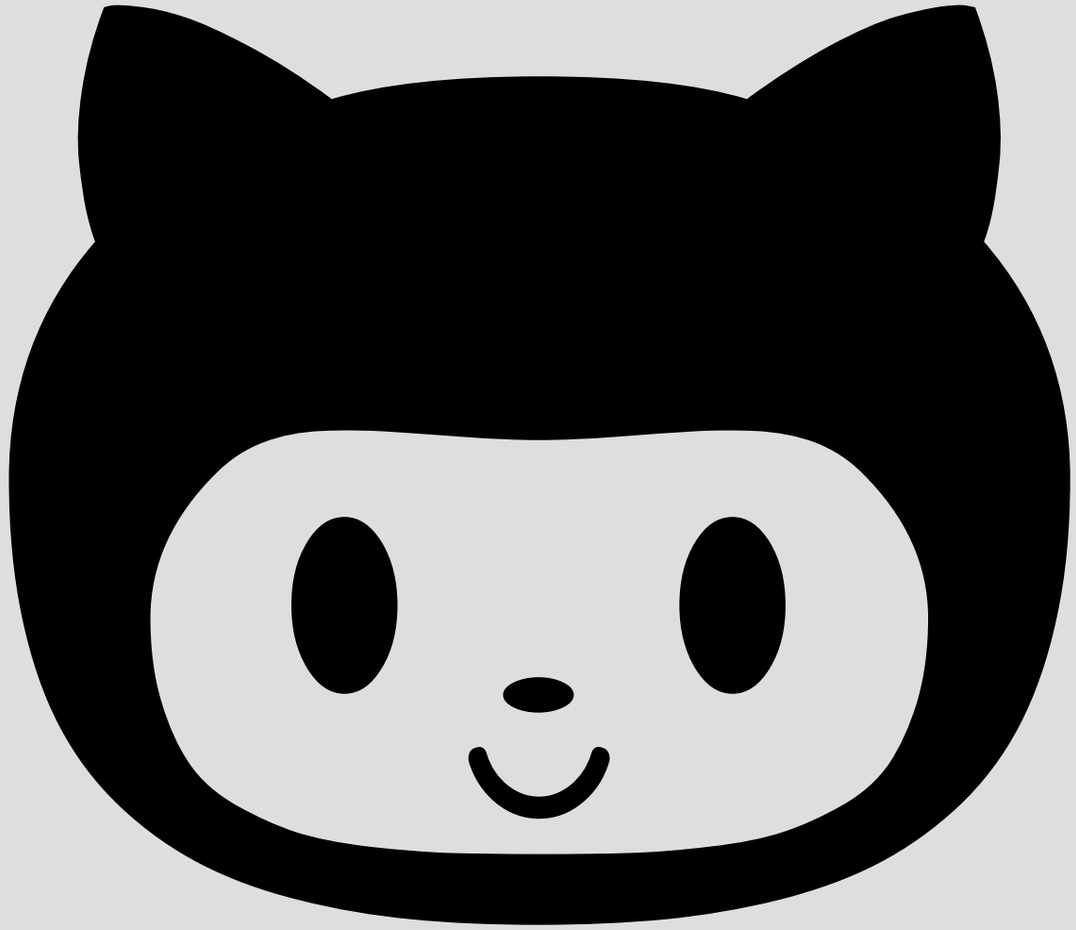
# What did GHTorrent do?

- Stopped distributing user names and emails in MySQL data dumps
  - Researchers can “sign” a form to get access to private data
- Created an opt-out process
- In the process of creating Terms of Fair Use

# A question of research ethics

Can we, in the name of science,

- send emails to developers?
- create developer profiles?
- recommend work to developers?
- rank developers based on contributions?
- compare project characteristics?
- characterise community practices?



http://ghtorrent.org