CONTINUOUS INTEGRATION IN GITHUB

QUALITY AND PRODUCTIVITY OUTCOMES

Bogdan Vasilescu

Yue Yu

Prem Devanbu

Vladimir Filkov

UCDAVIS
UNIVERSITY OF CALIFORNIA

国防科学技术大学
National University of Defense Technology

SOFTWARE PROCESSES ARE SOFTWARE TOO

Leon Osterweil

University of Colorado Boulder, Colorado USA

1. The Nature of Process.

The major theme of this meeting is the exploration of the importance of .ul process as a vehicle for improving both the quality of software products and the the way in which we develop and evolve them. In beginning this exploration it seems important to spend at least a short time examining the nature of process and convincing ourselves that this is indeed a promising vehicle.

We shall take as our elementary notion of a process that it is a systematic approach to the creation of a product or the accomplishment of some task. We observe that this characterization describes the notion of process commonly used in operating systems-- namely that a process is a computational task executing on a single computing device. Our characterization is much broader, however, describing any mechanism used to carry out work or achieve a goal in an orderly way. Our processes need not even be executable on a computer.

It is important for us to recognize that the notion of process is a pervasive one in the realm of human activities and that humans seem particularly adept at creating and carrying out processes. Knuth [Knuth 69] has observed that following recipes for food preparation is an example of carrying out what we now characterize as a process. Similarly it is not difficult to see that following assembly instructions in building toys or modular furniture is carrying out a process. Following office procedures or pursuing the steps of a manufacturing activity are more widely understood to be the pursuit of orderly process.

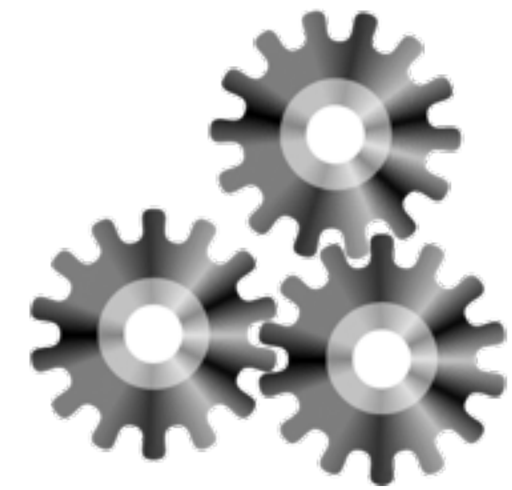
The latter examples serve to illustrate an important point-- namely that there is a key difference between a process and a process description. While a process is a vehicle for doing a job, a process description is a specification of how the job is to be done. Thus cookbook recipes are process descriptions while the carrying out of the recipes are processes. Office procedure manuals are process descriptions, while getting a specific office task done is a process. Similarly instructions for how to drive from one location to another are process descriptions, while doing the actual navigation and piloting is a process. From the point of view of a computer scientist the difference can be seen to be the difference between a type or class and an instance of that type or class. The process description defines a class or set of objects related to each other by virtue of the fact that they are all activities which follow the dictated behavior. We shall have reason to return to this point later in this presentation.

For now we should return to our consideration of the intuitive notion of process and study the important ramifications of the observations that 1) this notion is widespread and 2) exploitation of it is done very effectively by humans. Processes are used to effect generalized, indirect problem solving. The essence of the process exploitation paradigm seems to be that humans solve problems by creating process descriptions and then instantiating processes to solve individual problems. Rather than repetitively and directly solving individual instances of problems, humans prefer to create generalized solution specifications and make them available for instantiation (often by others) to solve individual problems directly.

One significant danger in this approach is that the process itself is a dynamic entity and the process description is a static entity. Further, the static process description is often constructed so as to specify a very wide and diverse collection of dynamic processes. This leaves open the distinct possibility that the process description may allow for process instances which do not perform "correctly." Dijkstra makes this observation in his famous letter on the GOTO statement, [Dijkstra 69] observing that computer programs are static entities and are thus easier for human minds to comprehend, while program executions are dynamic and far harder to comprehend and reason about effectively. Dijkstra's point was important then and no less significant now. Processes are hard to comprehend and reason about, while process descriptions, as static objects, are far easier to comprehend. Finally it is important to also endorse Dijkstra's conclusion that our reasoning about process descriptions is increasingly useful in understanding processes as the descriptions are increasingly transparent descriptions of all processes which might be instantiated.

In view of all of these dangers and difficulties it is surprising that humans embark upon the indirect process description/instantiation/execution approach to problem solving so frequently. It is even more startling to observe that this approach is successful and effective so often. This suggests that humans have an innate facility for indirect problem solving through process specification. It is precisely this innate ability which should be able to propel us to become far more systematic and effective in the development and evolution of computer software. What currently stands most directly in our way is our failure--to date--to understand our most central and difficult problems in terms of the process description/instantiation/execution paradigm.

ICSE, 1987



SOFTWARE PROCESSES ARE SOFTWARE TOO

Leon Osterweil

University of Colorado Boulder, Colorado USA

1. The Nature of Process.

The major theme of this meeting is the exploration of the importance of .ul process as a vehicle for improving both the quality of software products and the the way in which we develop and evolve them. In beginning this exploration it seems important to spend at least a short time examining the nature of process and convincing ourselves that this is indeed a promising vehicle.

We shall take as our elementary notion of a process that it is a systematic approach to the creation of a product or the accomplishment of some task. We observe that this characterization describes the notion of process commonly used in operating systems-- namely that a process is a computational task executing on a single computing device. Our characterization is much broader, however, describing any mechanism used to carry out work or achieve a goal in an orderly way. Our processes need not even be executable on a computer.

It is important for us to recognize that the notion of process is a pervasive one in the realm of human activities and that humans seem particularly adept at creating and carrying out processes. Knuth [Knuth 69] has observed that following recipes for food preparation is an example of carrying out what we now characterize as a process. Similarly it is not difficult to see that following assembly instructions in building toys or modular furniture is carrying out a process. Following office procedures or pursuing the steps of a manufacturing activity are more widely understood to be the pursuit of orderly process.

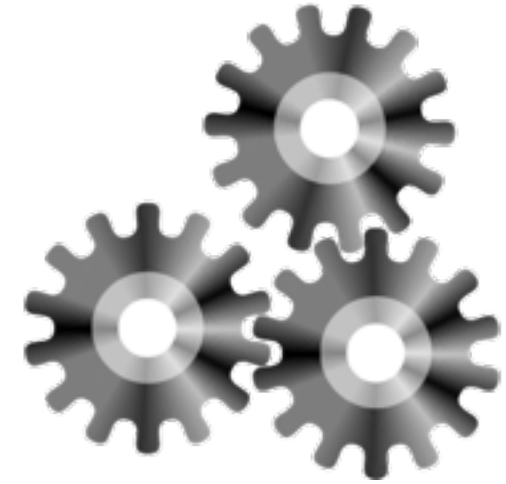
The latter examples serve to illustrate an important point-- namely that there is a key difference between a process and a process description. While a process is a vehicle for doing a job, a process description is a specification of how the job is to be done. Thus cookbook recipes are process descriptions while the carrying out of the recipes are processes. Office procedure manuals are process descriptions, while getting a specific office task done is a process. Similarly instructions for how to drive from one location to another are process descriptions, while doing the actual navigation and piloting is a process. From the point of view of a computer scientist the difference can be seen to be the difference between a type or class and an instance of that type or class. The process description defines a class or set of objects related to each other by virtue of the fact that they are all activities which follow the dictated behavior. We shall have reason to return to this point later in this presentation.

For now we should return to our consideration of the intuitive notion of process and study the important ramifications of the observations that 1) this notion is widespread and 2) exploitation of it is done very effectively by humans. Processes are used to effect generalized, indirect problem solving. The essence of the process exploitation paradigm seems to be that humans solve problems by creating process descriptions and then instantiating processes to solve individual problems. Rather than repetitively and directly solving individual instances of problems, humans prefer to create generalized solution specifications and make them available for instantiation (often by others) to solve individual problems directly.

One significant danger in this approach is that the process itself is a dynamic entity and the process description is a static entity. Further, the static process description is often constructed so as to specify a very wide and diverse collection of dynamic processes. This leaves open the distinct possibility that the process description may allow for process instances which do not perform "correctly." Dijkstra makes this observation in his famous letter on the GOTO statement, [Dijkstra 69] observing that computer programs are static entities and are thus easier for human minds to comprehend, while program executions are dynamic and far harder to comprehend and reason about effectively. Dijkstra's point was important then and no less significant now. Processes are hard to comprehend and reason about, while process descriptions, as static objects, are far easier to comprehend. Finally it is important to also endorse Dijkstra's conclusion that our reasoning about process descriptions is increasingly useful in understanding processes as the descriptions are increasingly transparent descriptions of all processes which might be instantiated.

In view of all of these dangers and difficulties it is surprising that humans embark upon the indirect process description/instantiation/execution approach to problem solving so frequently. It is even more startling to observe that this approach is successful and effective so often. This suggests that humans have an innate facility for indirect problem solving through process specification. It is precisely this innate ability which should be able to propel us to become far more systematic and effective in the development and evolution of computer software. What currently stands most directly in our way is our failure--to date--to understand our most central and difficult problems in terms of the process description/instantiation/execution paradigm.

ICSE, 1987



SOFTWARE PROCESSES ARE SOFTWARE TOO

Leon Osterweil

University of Colorado Boulder, Colorado USA

1. The Nature of Process.

The major theme of this meeting is the exploration of the importance of .ul process as a vehicle for improving both the quality of software products and the the way in which we develop and evolve them. In beginning this exploration it seems important to spend at least a short time examining the nature of process and convincing ourselves that this is indeed a promising vehicle.

We shall take as our elementary notion of a process that it is a systematic approach to the creation of a product or the accomplishment of some task. We observe that this characterization describes the notion of process commonly used in operating systems-- namely that a process is a computational task executing on a single computing device. Our characterization is much broader, however, describing any mechanism used to carry out work or achieve a goal in an orderly way. Our processes need not even be executable on a computer.

It is important for us to recognize that the notion of process is a pervasive one in the realm of human activities and that humans seem particularly adept at creating and carrying out processes. Knuth [Knuth 69] has observed that following recipes for food preparation is an example of carrying out what we now characterize as a process. Similarly it is not difficult to see that following assembly instructions in building toys or modular furniture is carrying out a process. Following office procedures or pursuing the steps of a manufacturing activity are more widely understood to be the pursuit of orderly process.

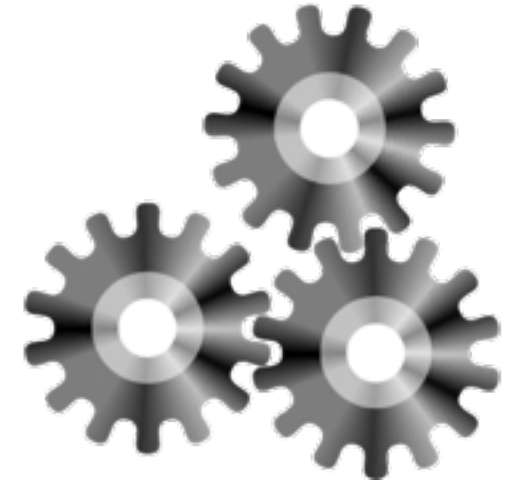
The latter examples serve to illustrate an important point-- namely that there is a key difference between a process and a process description. While a process is a vehicle for doing a job, a process description is a specification of how the job is to be done. Thus cookbook recipes are process descriptions while the carrying out of the recipes are processes. Office procedure manuals are process descriptions, while getting a specific office task done is a process. Similarly instructions for how to drive from one location to another are process descriptions, while doing the actual navigation and piloting is a process. From the point of view of a computer scientist the difference can be seen to be the difference between a type or class and an instance of that type or class. The process description defines a class or set of objects related to each other by virtue of the fact that they are all activities which follow the dictated behavior. We shall have reason to return to this point later in this presentation.

For now we should return to our consideration of the intuitive notion of process and study the important ramifications of the observations that 1) this notion is widespread and 2) exploitation of it is done very effectively by humans. Processes are used to effect generalized, indirect problem solving. The essence of the process exploitation paradigm seems to be that humans solve problems by creating process descriptions and then instantiating processes to solve individual problems. Rather than repetitively and directly solving individual instances of problems, humans prefer to create generalized solution specifications and make them available for instantiation (often by others) to solve individual problems directly.

One significant danger in this approach is that the process itself is a dynamic entity and the process description is a static entity. Further, the static process description is often constructed so as to specify a very wide and diverse collection of dynamic processes. This leaves open the distinct possibility that the process description may allow for process instances which do not perform "correctly." Dijkstra makes this observation in his famous letter on the GOTO statement, [Dijkstra 69] observing that computer programs are static entities and are thus easier for human minds to comprehend, while program executions are dynamic and far harder to comprehend and reason about effectively. Dijkstra's point was important then and no less significant now. Processes are hard to comprehend and reason about, while process descriptions, as static objects, are far easier to comprehend. Finally it is important to also endorse Dijkstra's conclusion that our reasoning about process descriptions is increasingly useful in understanding processes as the descriptions are increasingly transparent descriptions of all processes which might be instantiated.

In view of all of these dangers and difficulties it is surprising that humans embark upon the indirect process description/instantiation/execution approach to problem solving so frequently. It is even more startling to observe that this approach is successful and effective so often. This suggests that humans have an innate facility for indirect problem solving through process specification. It is precisely this innate ability which should be able to propel us to become far more systematic and effective in the development and evolution of computer software. What currently stands most directly in our way is our failure--to date--to understand our most central and difficult problems in terms of the process description/instantiation/execution paradigm.

ICSE, 1987



Productivity?

Quality?

The pull-based model

... traditionally

Octocats from https://octodex.github.com/

The pull-based model

... traditionally

Octocats from https://octodex.github.com/

The pull-based model

... traditionally

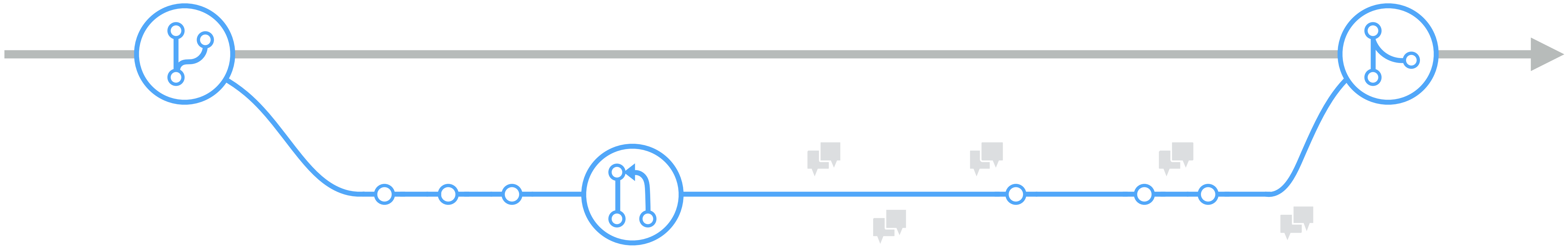
Octocats from https://octodex.github.com/

The pull-based model

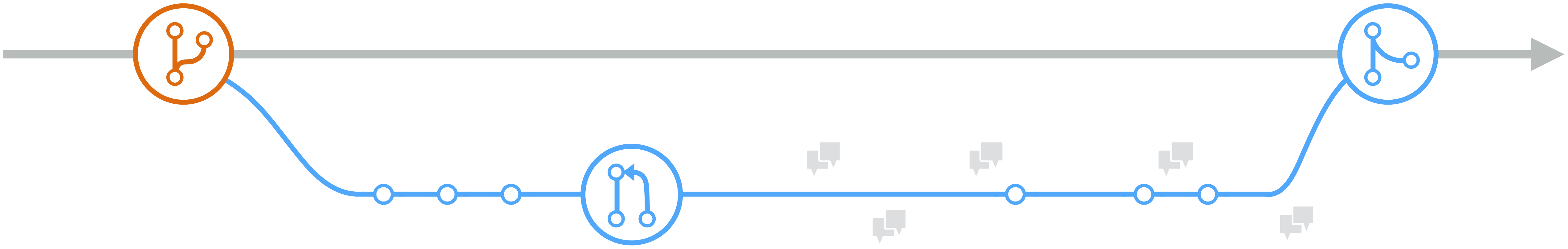
… traditionally

Octocats from https://octodex.github.com/

The Pull Request process

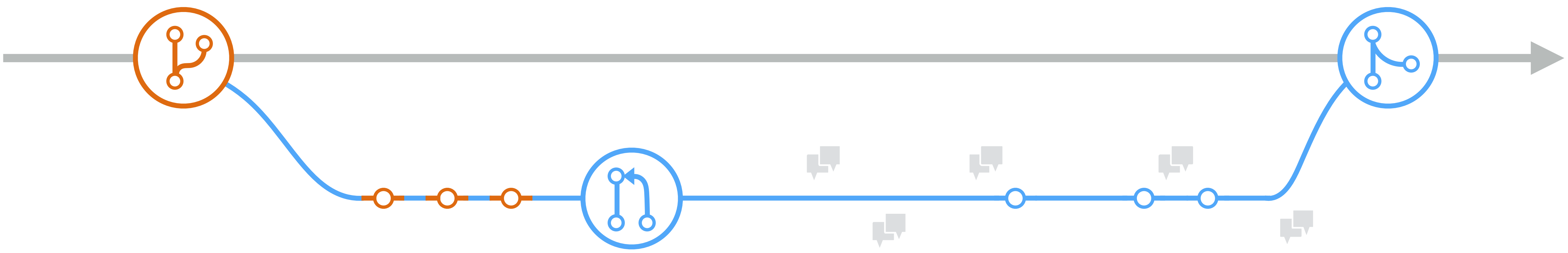


The Pull Request process



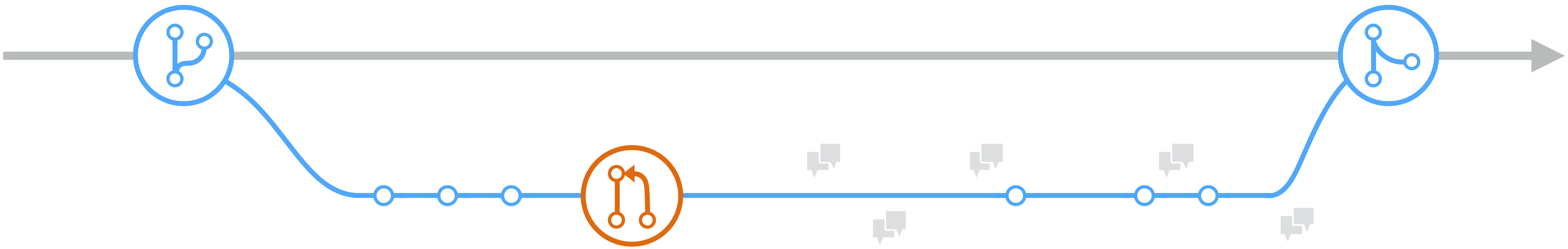
Create
a branch

The Pull Request process



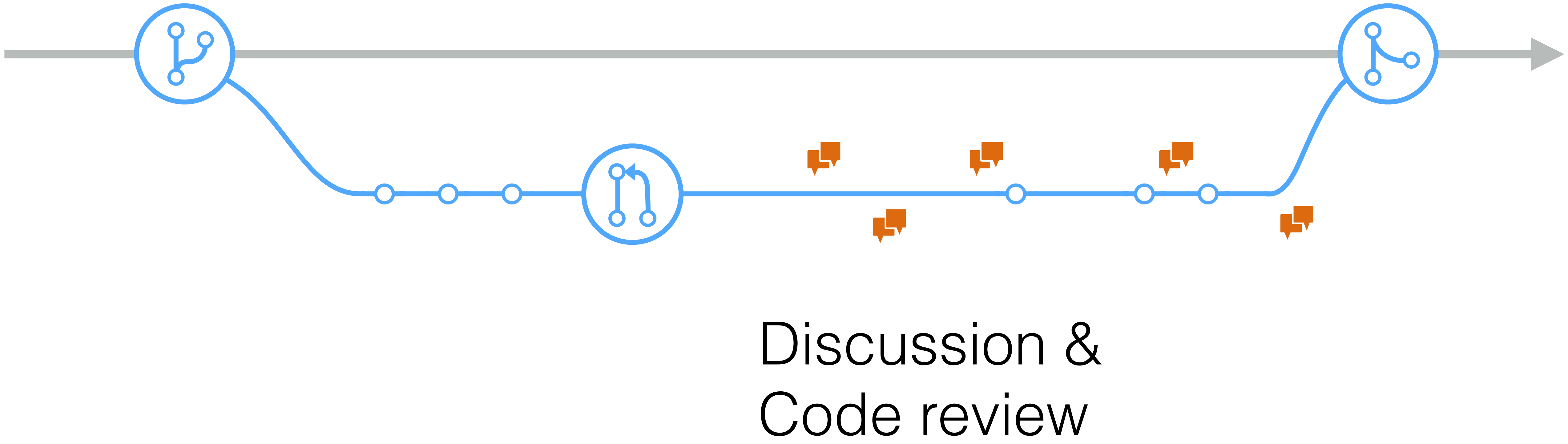
Add
commits

The Pull Request process

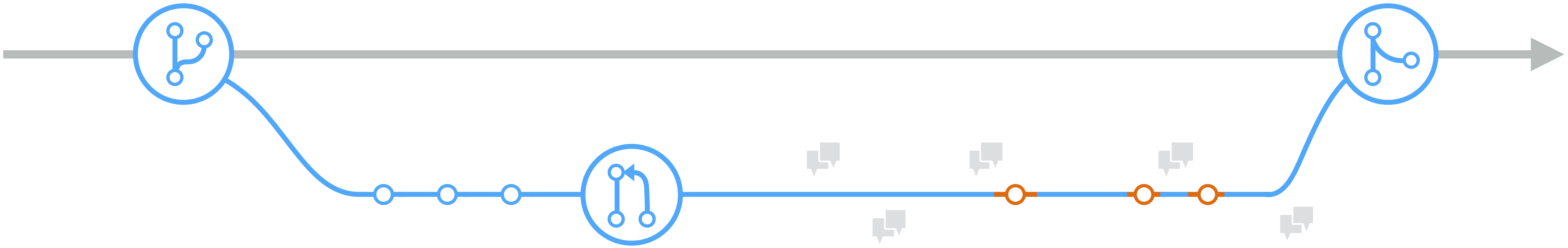


Open a
Pull Request

The Pull Request process

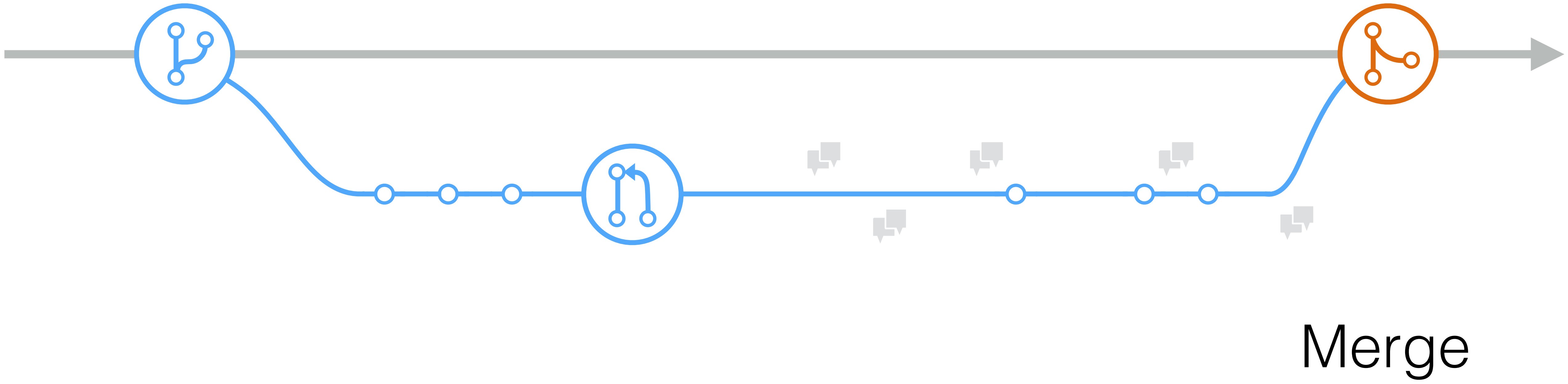


The Pull Request process



Pull Request
updates

The Pull Request process



The pull-based model

... modernly

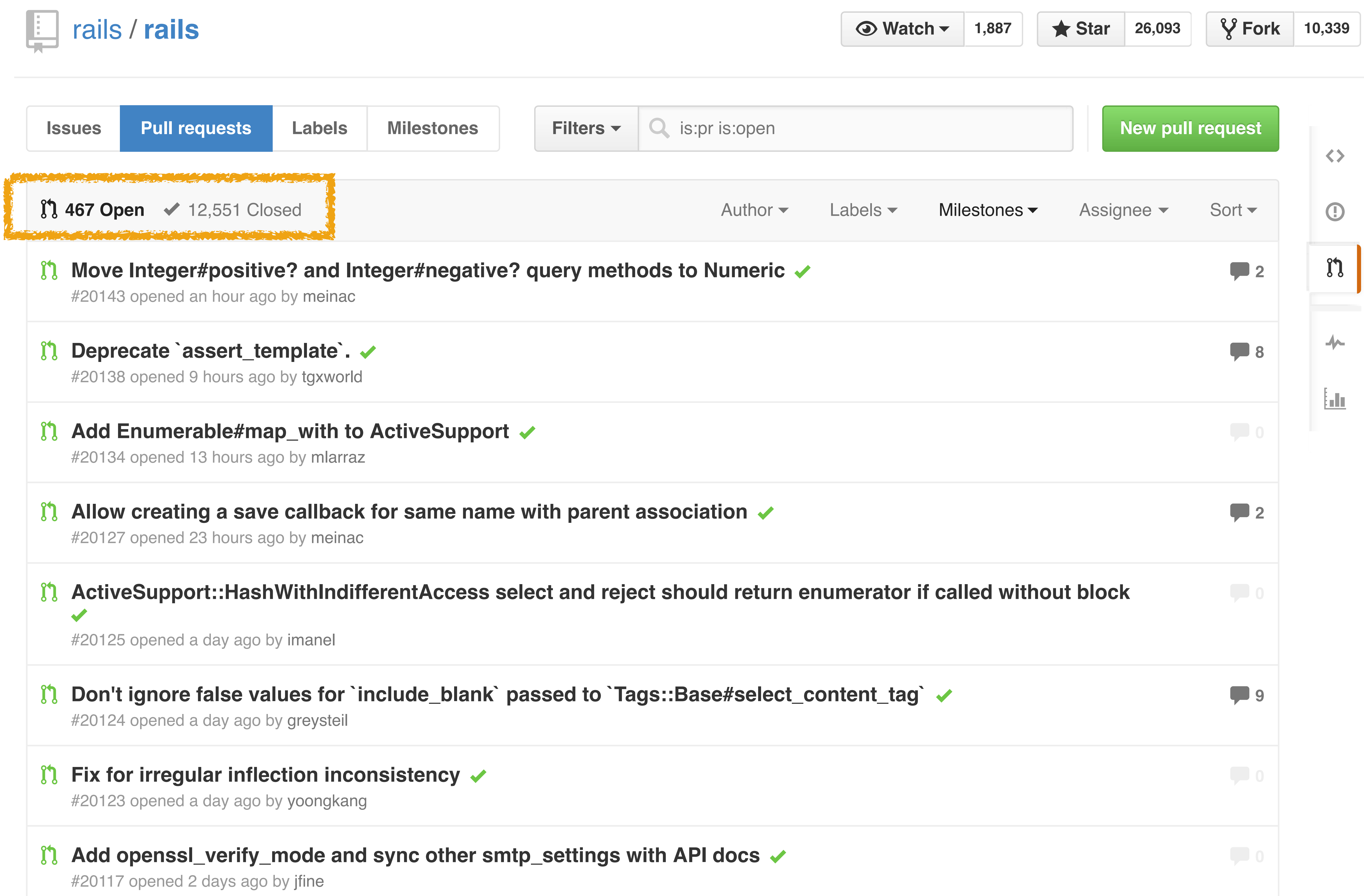
The pull-based model

... modernly

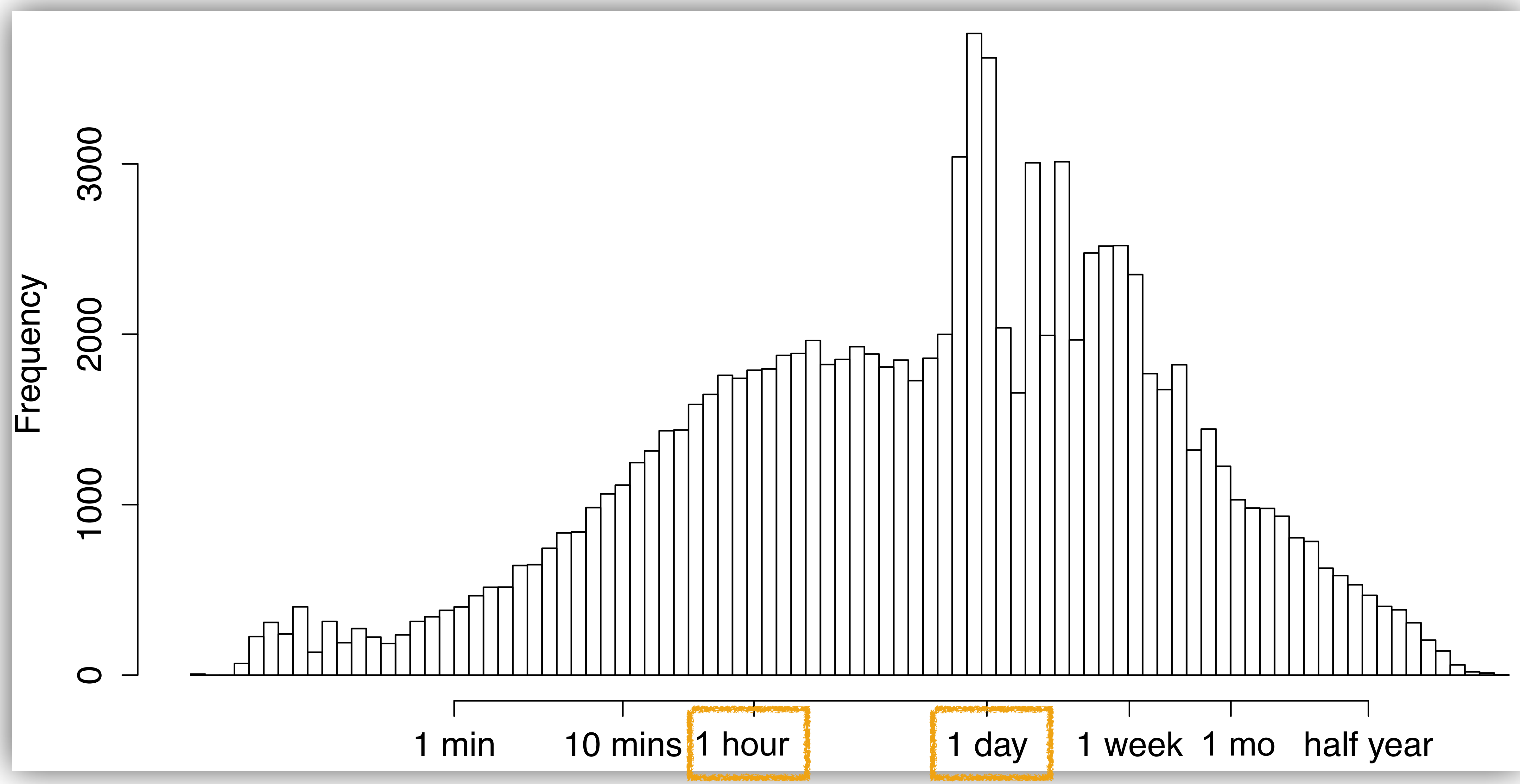
... because code review

- Open source-style collaborative development practices in commercial projects using GitHub
  E Kalliamvakou, D Damian, K Blincoe, L Singer, DM German. *ICSE 2015*
- Work practices and challenges in pull-based development: the integrator's perspective
  G Gousios, A Zaidman, MA Storey, A Van Deursen. *ICSE 2015*

Considerable review load



Considerable review load

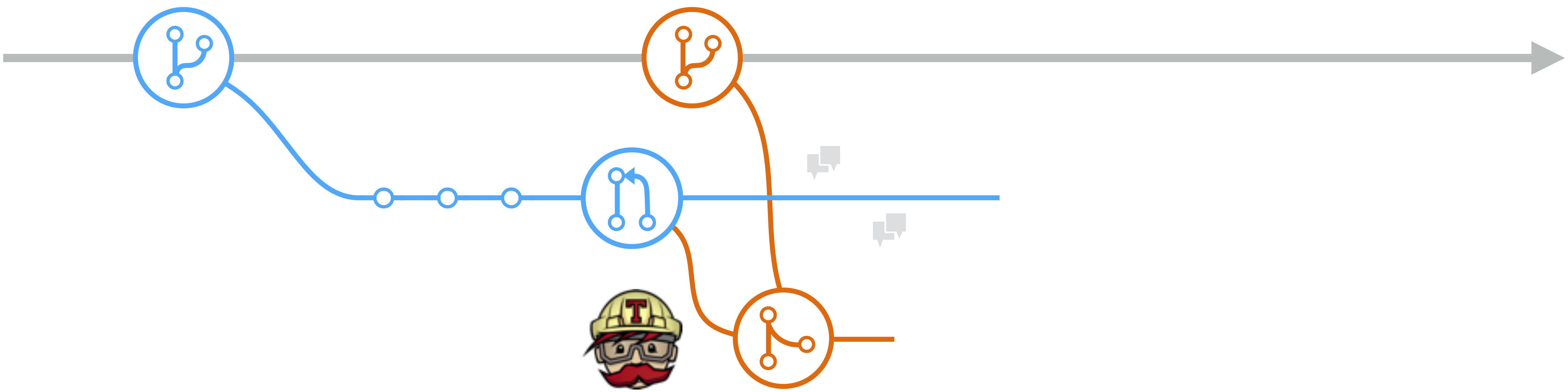


- Wait for it: Determinants of pull request evaluation latency on GitHub
  Y Yu, H Wang, V Filkov, P Devanbu, B Vasilescu. *MSR 2015*

The Pull Request process

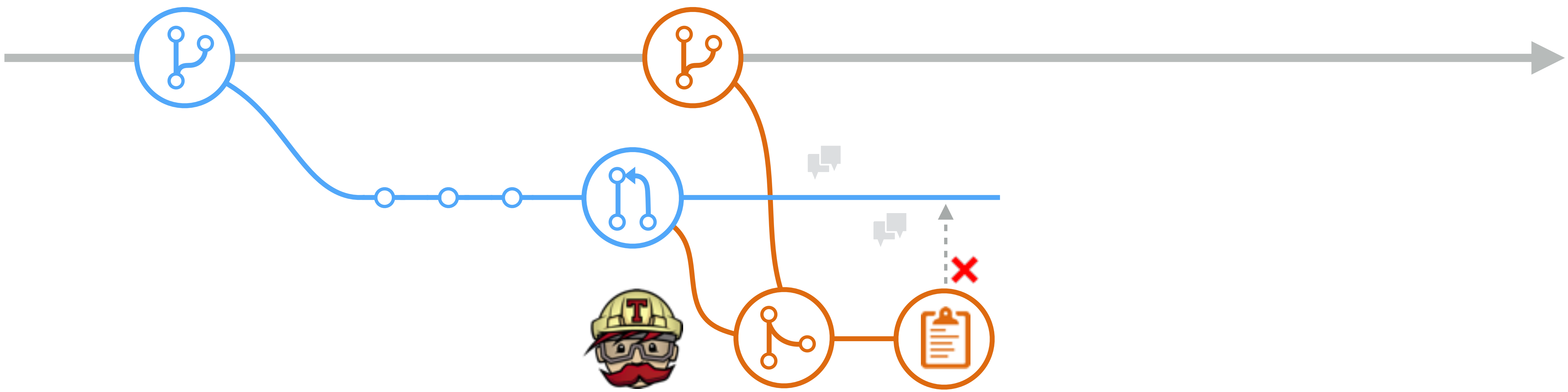
… with Travis-CI

The Pull Request process
... with Travis-CI



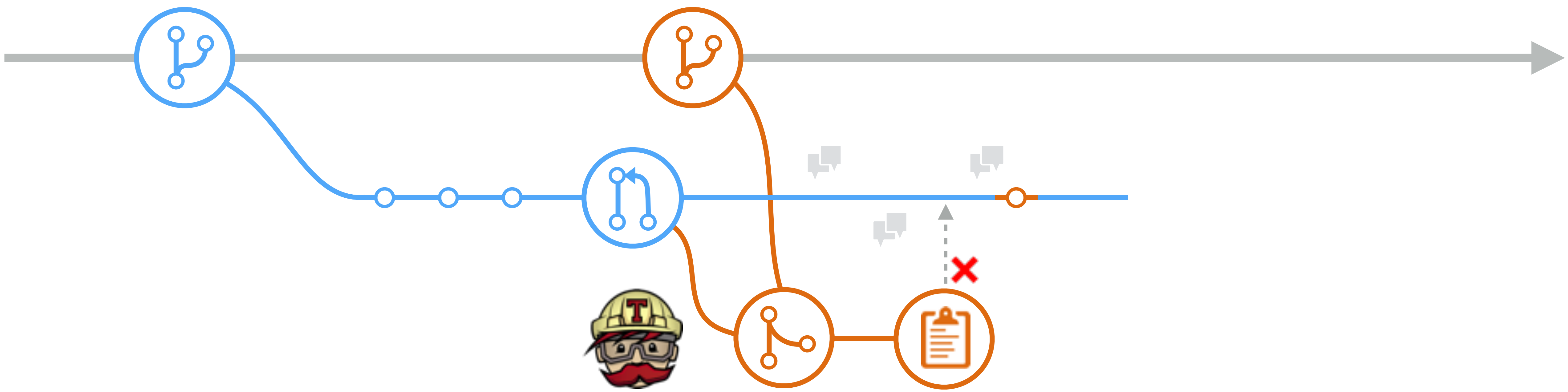
Pull Request is
automatically
merged into
testing branch

The Pull Request process
… with Travis-CI



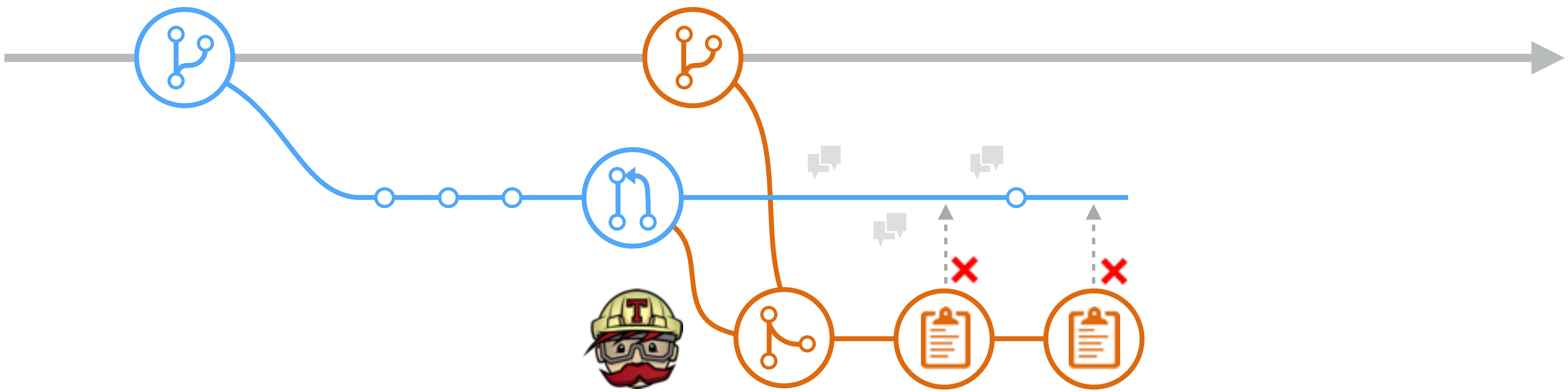
Test suite runs automatically

The Pull Request process
… with Travis-CI



Pull Request
is updated in
response to
test failures

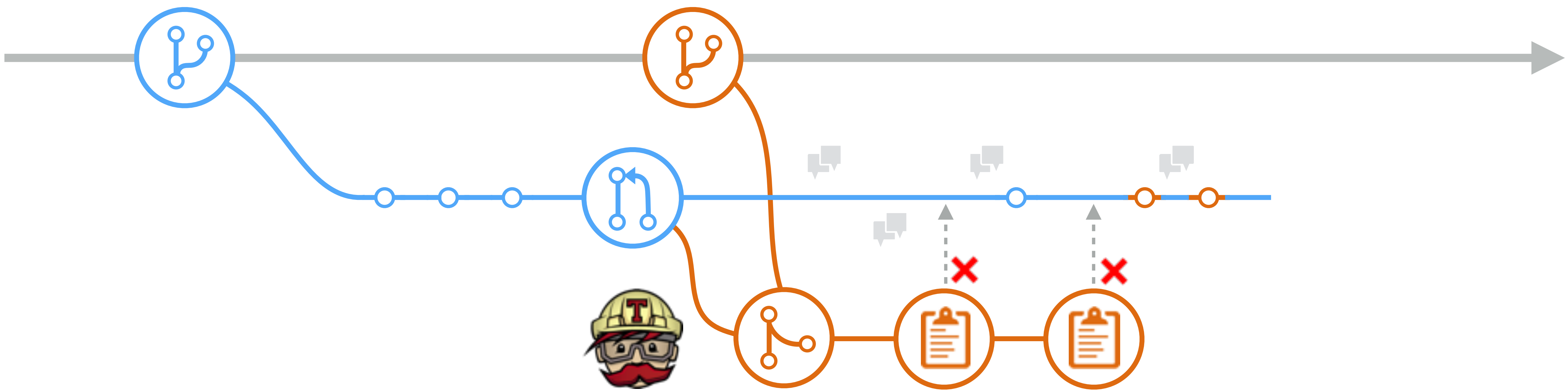
The Pull Request process
… with Travis-CI



Tests rerun
after update

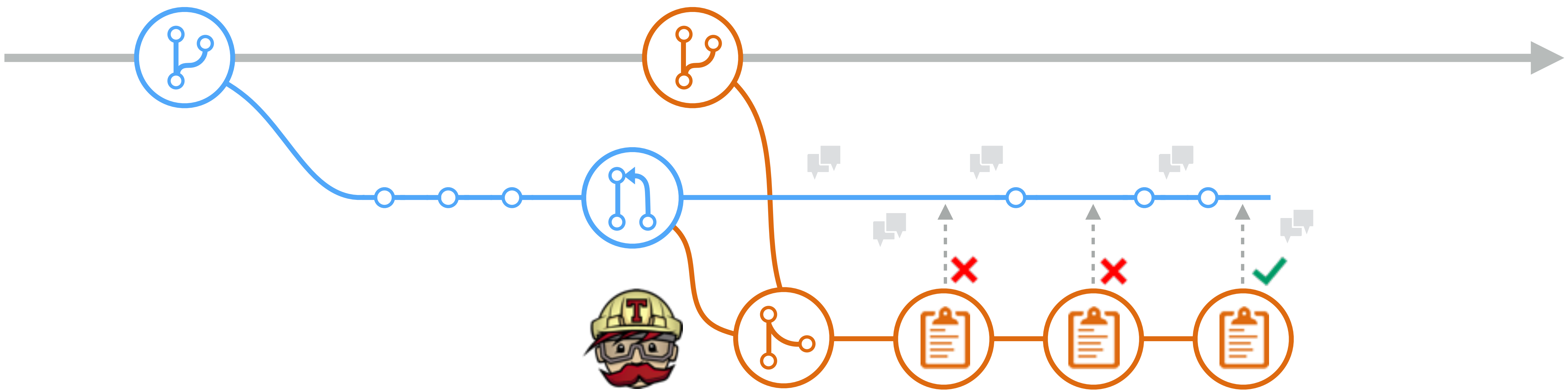
The Pull Request process

... with Travis-CI



More
updates

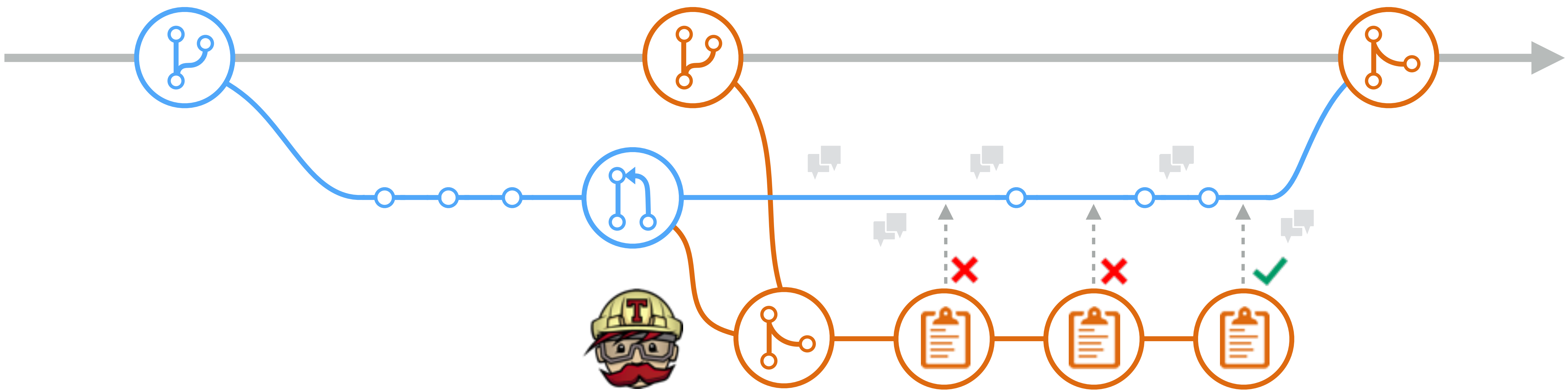
The Pull Request process
… with Travis-CI



Tests
finally
pass

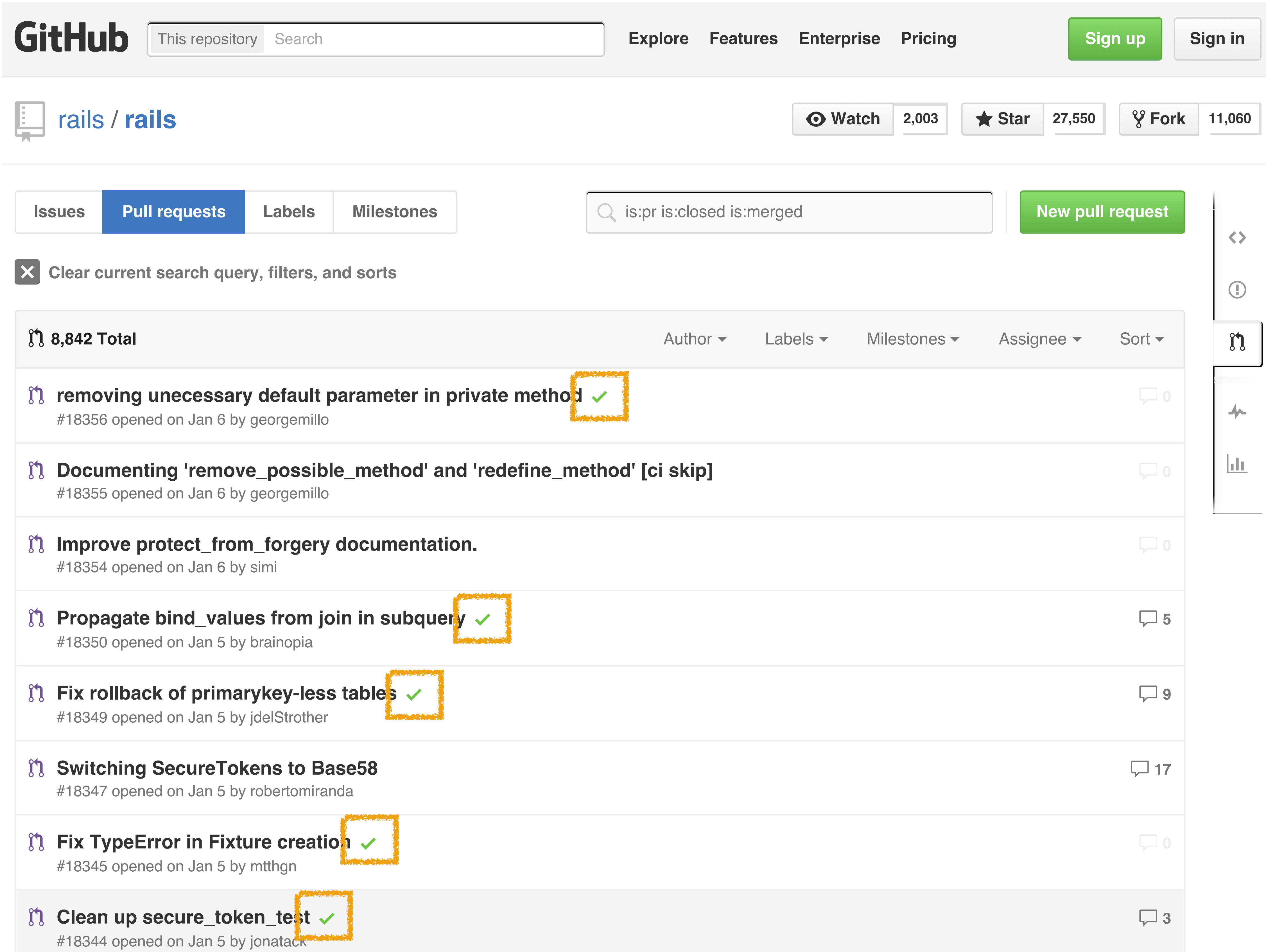
The Pull Request process

… with Travis-CI

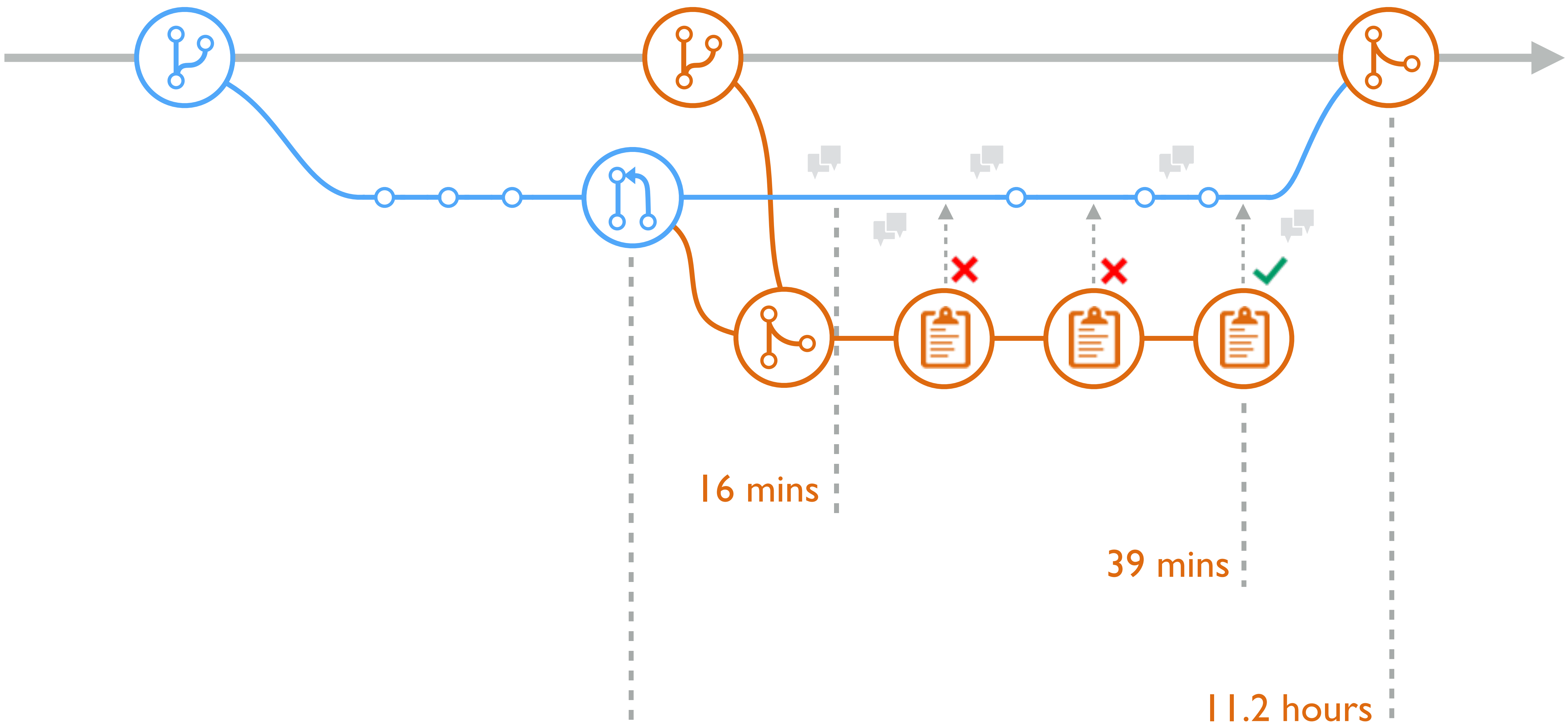


Merge

Merge after CI tests pass

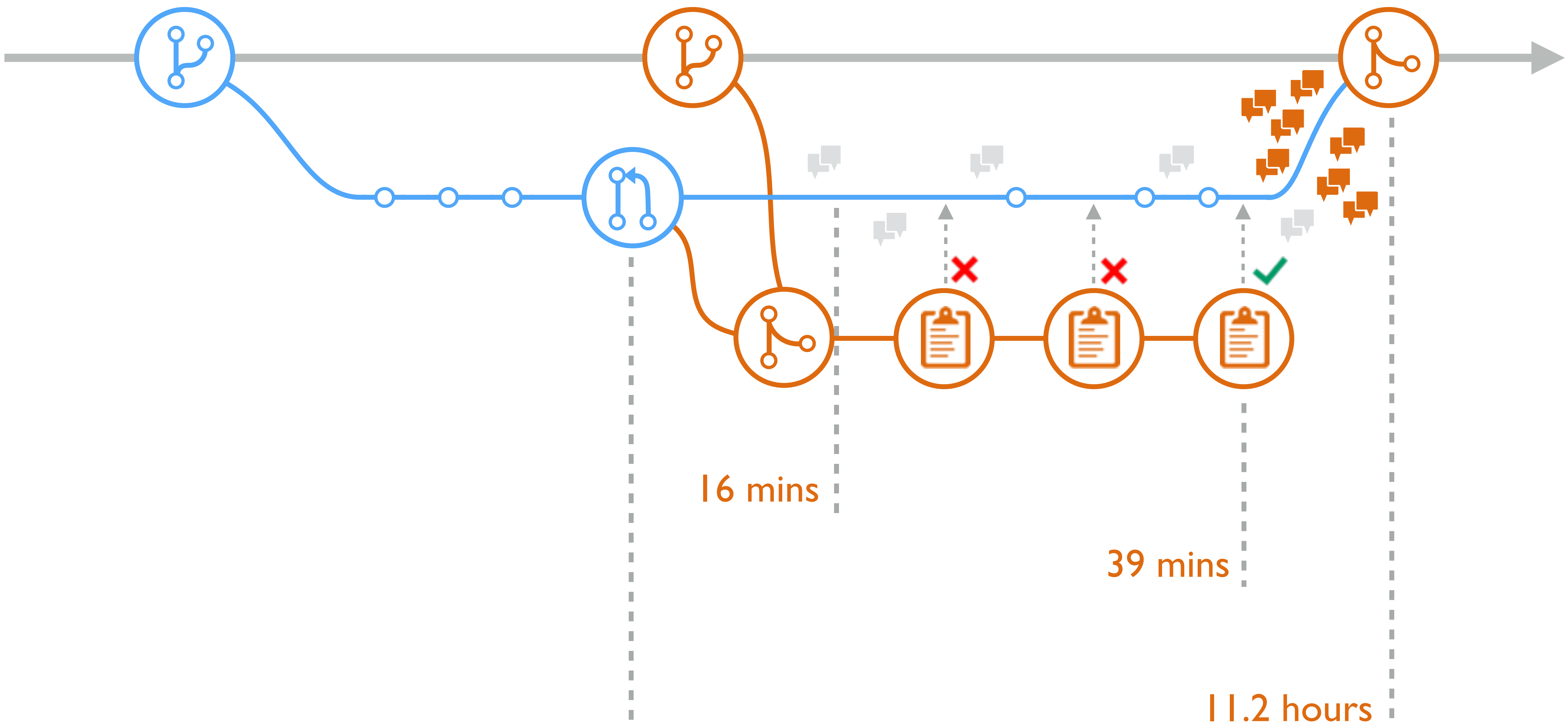


~~Merge~~ after CI tests pass
Code review

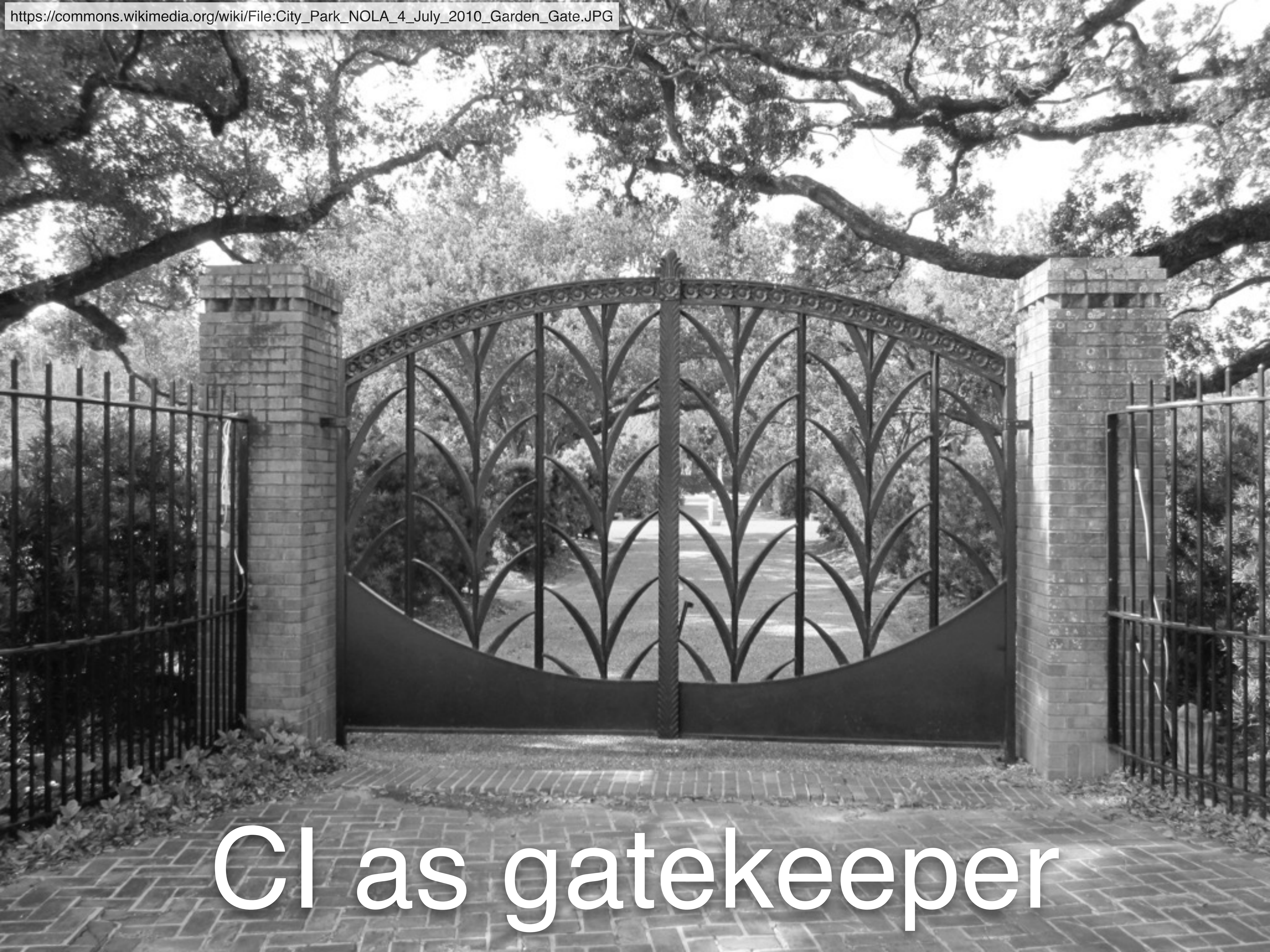


- Wait for it: Determinants of pull request evaluation latency on GitHub
  Y Yu, H Wang, V Filkov, P Devanbu, B Vasilescu. *MSR 2015*

~~Merge~~ after CI tests pass
Code review



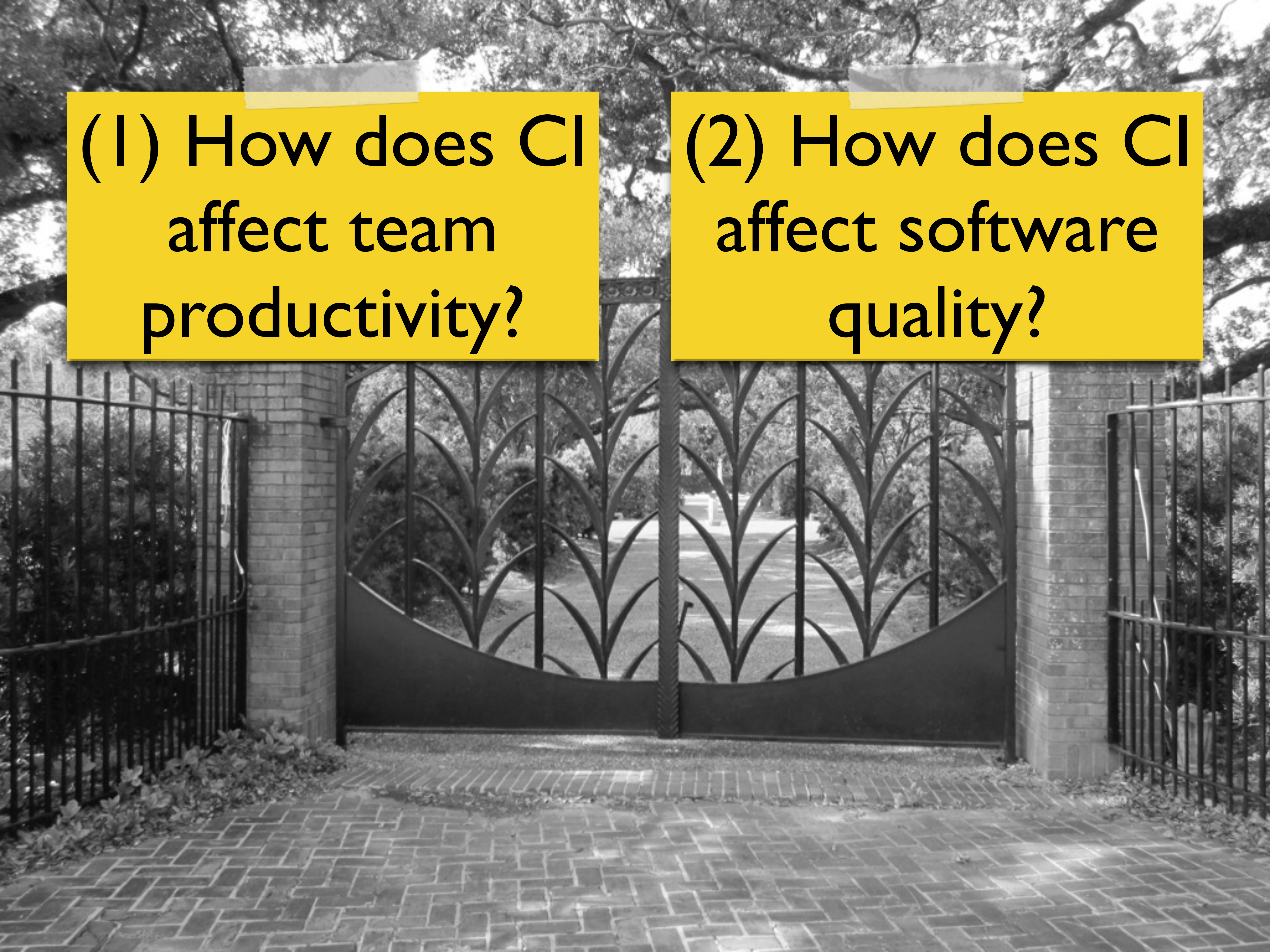
- Wait for it: Determinants of pull request evaluation latency on GitHub
  Y Yu, H Wang, V Filkov, P Devanbu, B Vasilescu. *MSR 2015*


https://commons.wikimedia.org/wiki/File:City_Park_NOLA_4_July_2010_Garden_Gate.JPG
CI as gatekeeper

"[CI] enables us to automate more of our process which frees us up to focus on the important things — like implementing and shipping features! [...]

[The integration of Travis-CI in GitHub] enables the team to rapidly find integration errors or regression failures in the test suite. This tightens the feedback loop and not only enables more defect free code, but greatly speeds up our process."

- Automate All The Things: Continuous Integration and Deployment at RevUnit
  Addam Hardy. http://addamhardy.com/blog/2013/09/28/automate-all-the-things-continuous-integration-and-continuous-deployment-at-revunit/


(1) How does CI affect team productivity?
(2) How does CI affect software quality?

(1) How does CI affect team productivity?

20% more pull requests merged &
40% fewer rejected

(2) How does CI affect software quality?

50% more bugs reported monthly by <u>core dev's</u>

No impact on bugs reported by <u>externals</u>

(1) How does CI affect team productivity?

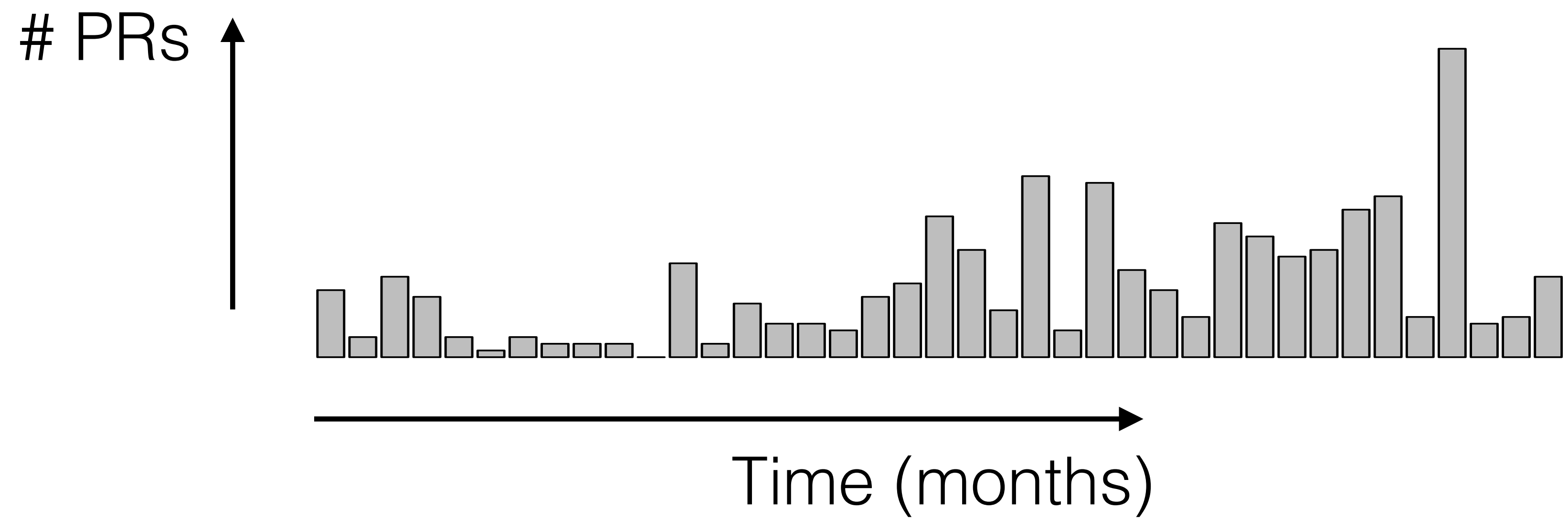
Data Set

246 GitHub projects

- Not forks
- Ruby, Python, JavaScript, PHP, Java, Scala, C, C++
- 200+ pull requests
- Travis-CI

(1) How does CI affect team productivity?

Data Set

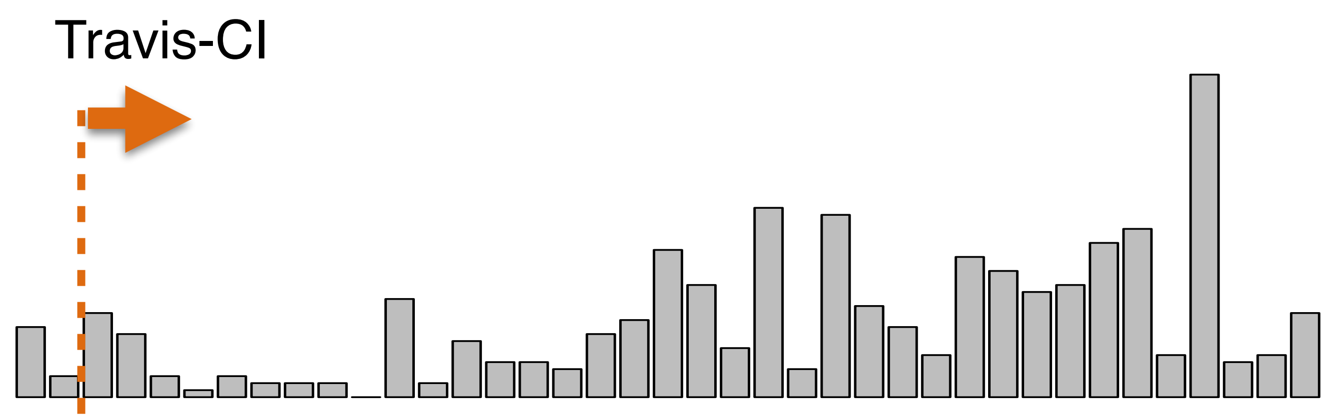


246 GitHub projects

- Not forks
- Ruby, Python, JavaScript, PHP, Java, Scala, C, C++
- 200+ pull requests
- Travis-CI

(1) How does CI affect team productivity?

Data Set



246 GitHub projects

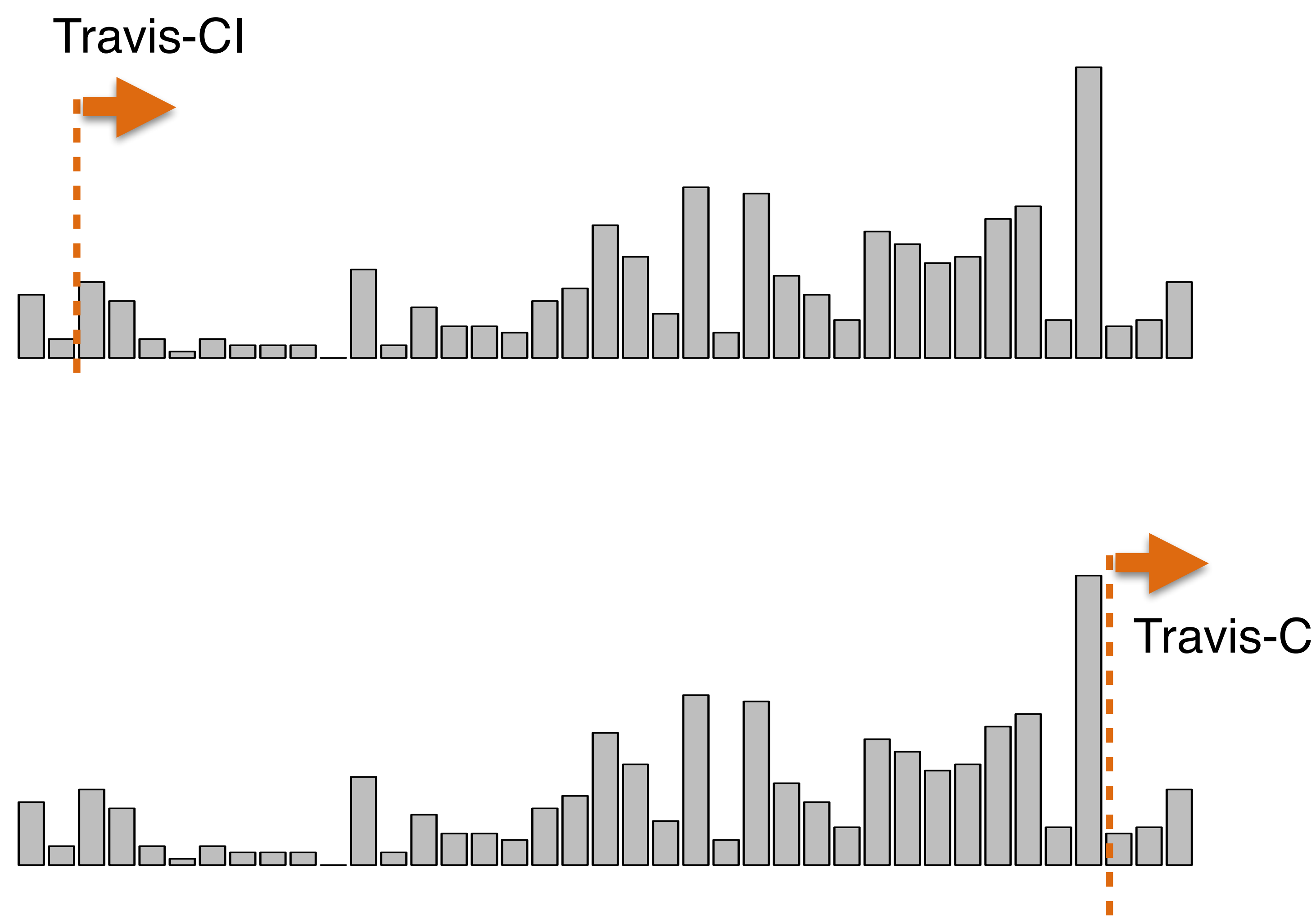
- Not forks
- Ruby, Python, JavaScript, PHP, Java, Scala, C, C++
- 200+ pull requests
- Travis-CI

(1) How does CI affect team productivity?

Data Set

246 GitHub projects

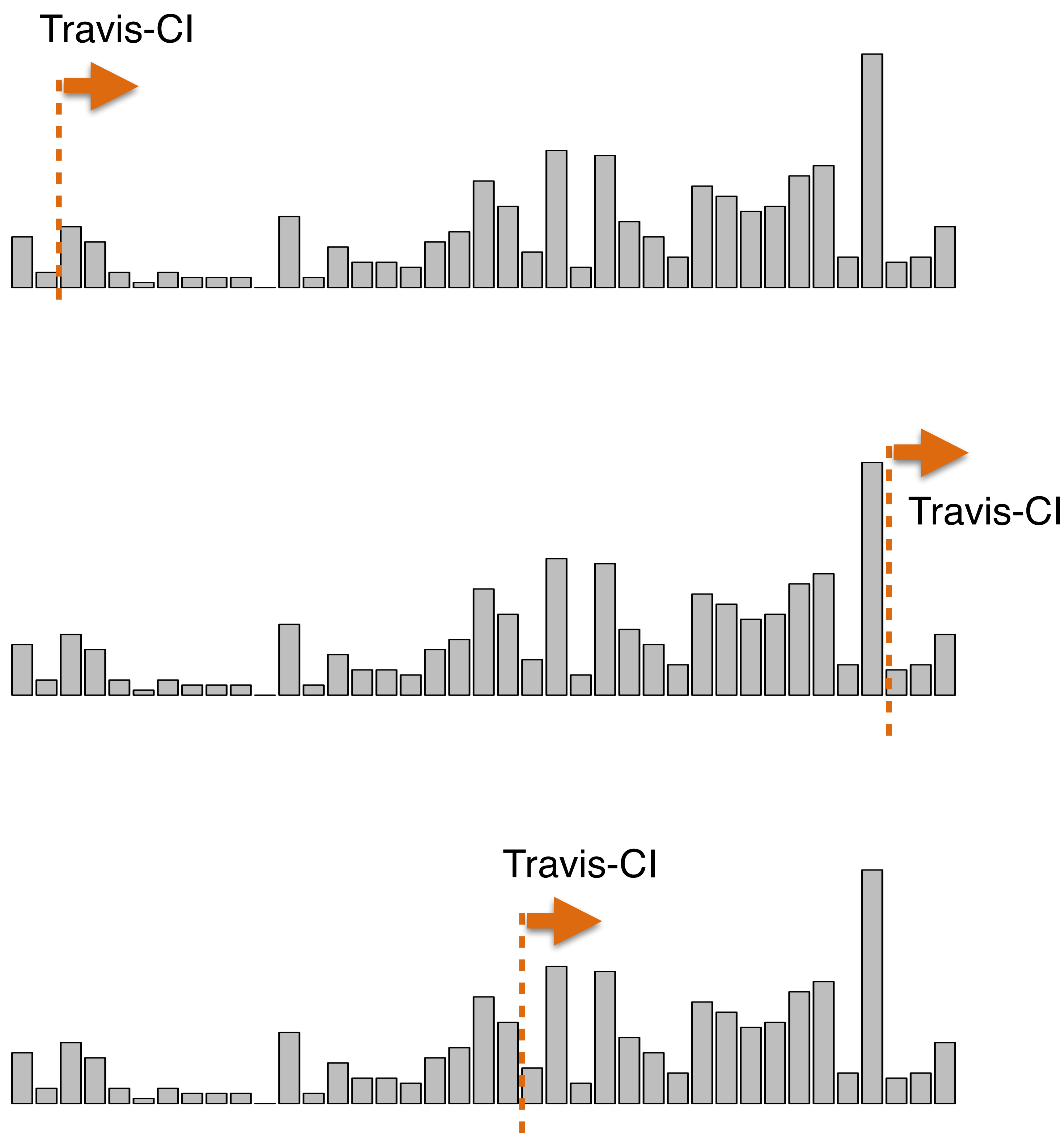
- Not forks
- Ruby, Python, JavaScript, PHP, Java, Scala, C, C++
- 200+ pull requests
- Travis-CI



Data Set

246 GitHub projects

- Not forks
- Ruby, Python, JavaScript, PHP, Java, Scala, C, C++
- 200+ pull requests
- Travis-CI

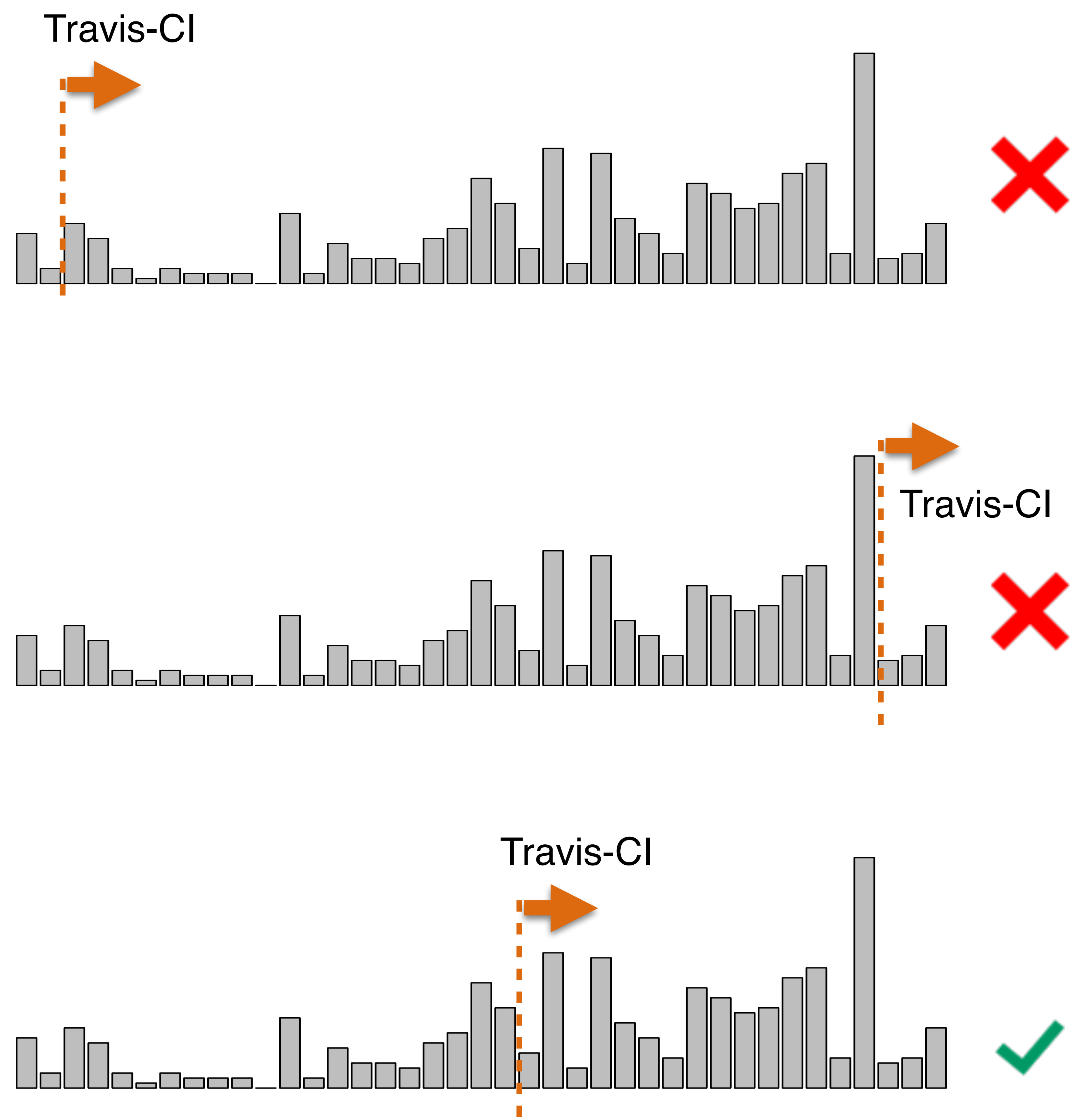


(1) How does CI affect team productivity?

Data Set

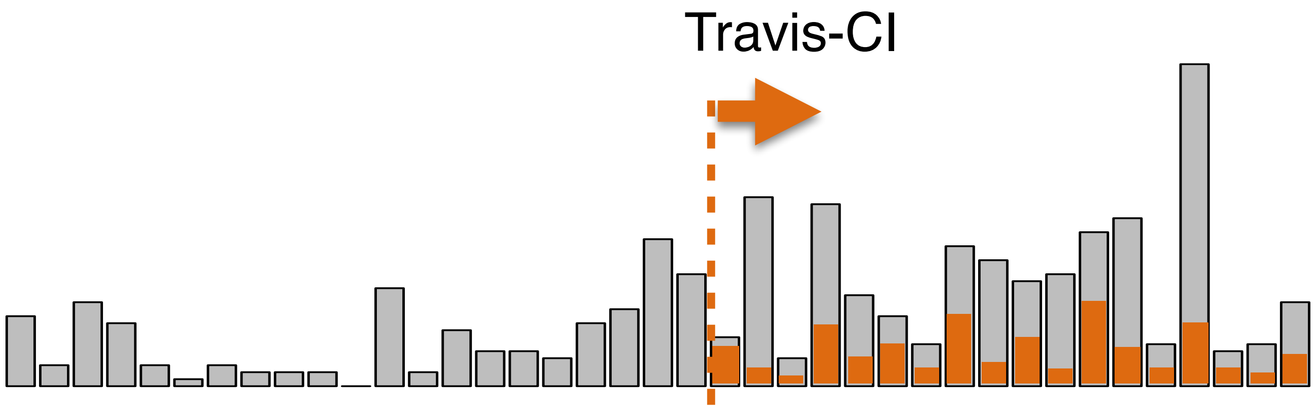
246 GitHub projects

- Not forks
- Ruby, Python, JavaScript, PHP, Java, Scala, C, C++
- 200+ pull requests
- Travis-CI



(1) How does CI affect team productivity?

Data Set



246 GitHub projects

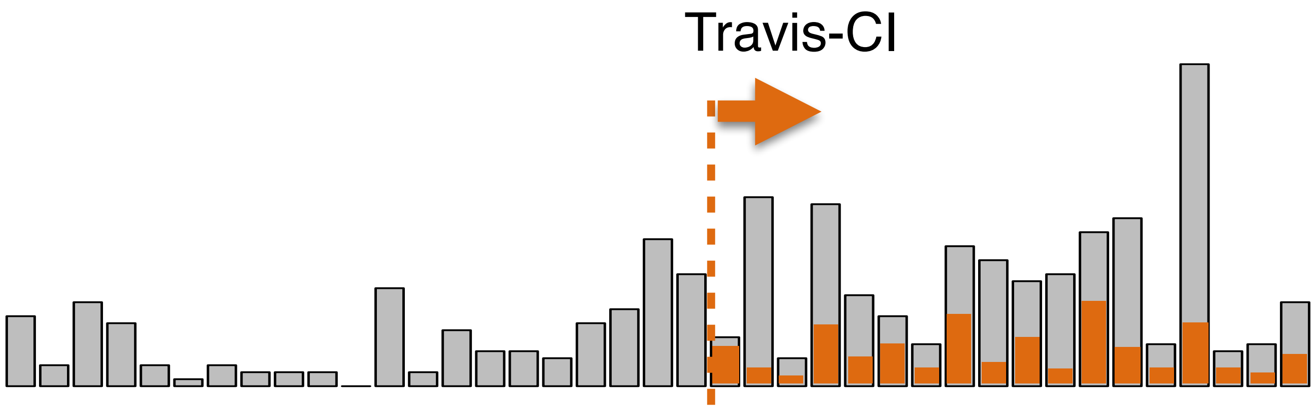
- Not forks
- Ruby, Python, JavaScript, PHP, Java, Scala, C, C++
- 200+ pull requests
- Travis-CI

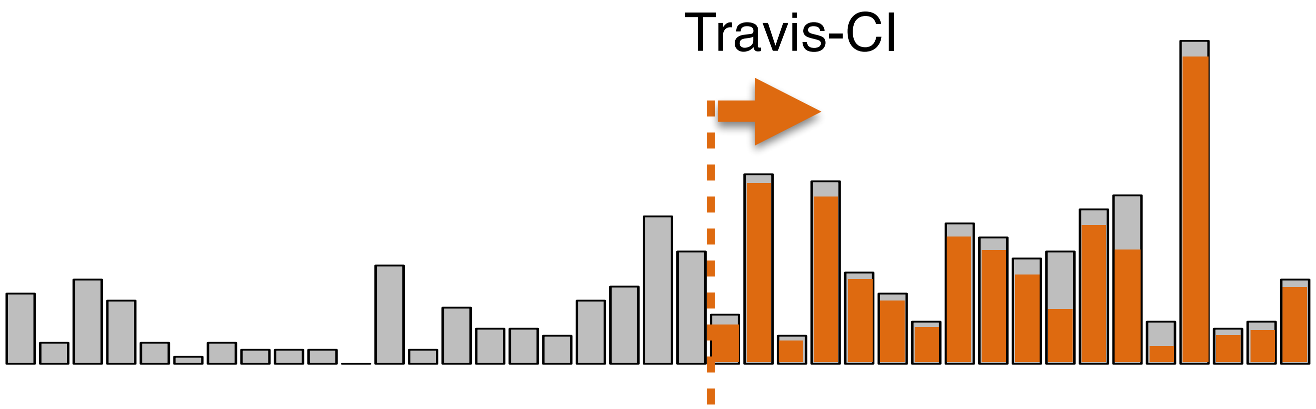
(1) How does CI affect team productivity?

Data Set

246 GitHub projects

- Not forks
- Ruby, Python, JavaScript, PHP, Java, Scala, C, C++
- 200+ pull requests
- Travis-CI

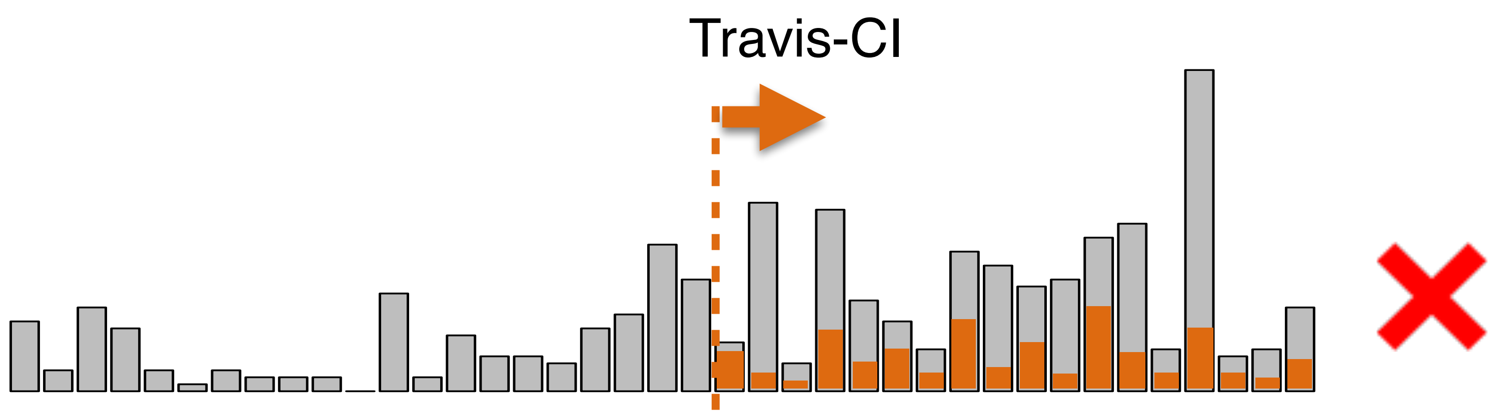




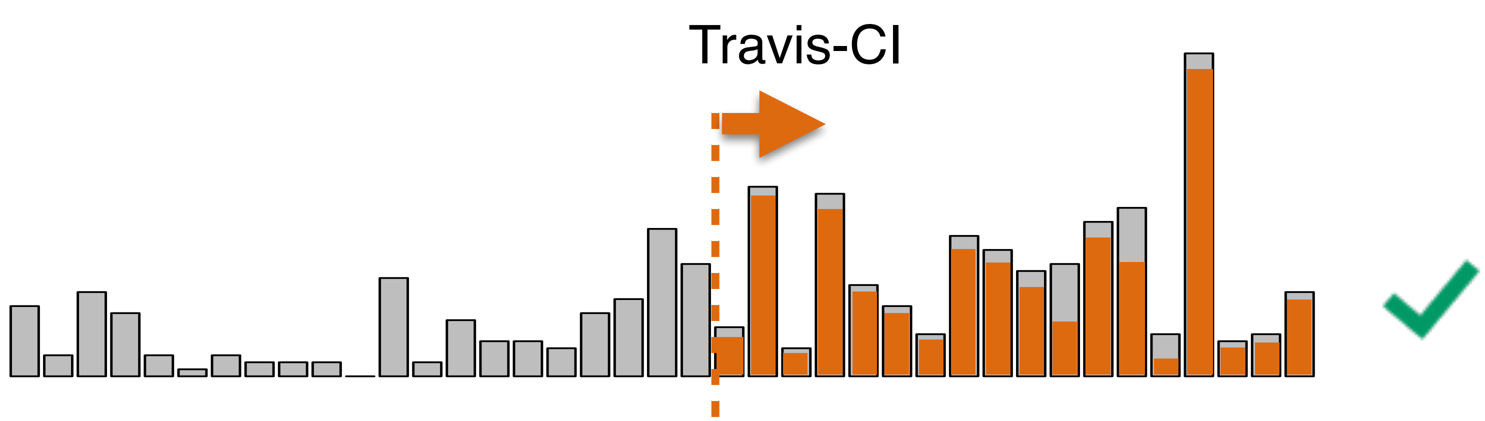
(1) How does CI affect team productivity?

Data Set

246 GitHub projects



- Not forks
- Ruby, Python, JavaScript, PHP, Java, Scala, C, C++
- 200+ pull requests
- Travis-CI

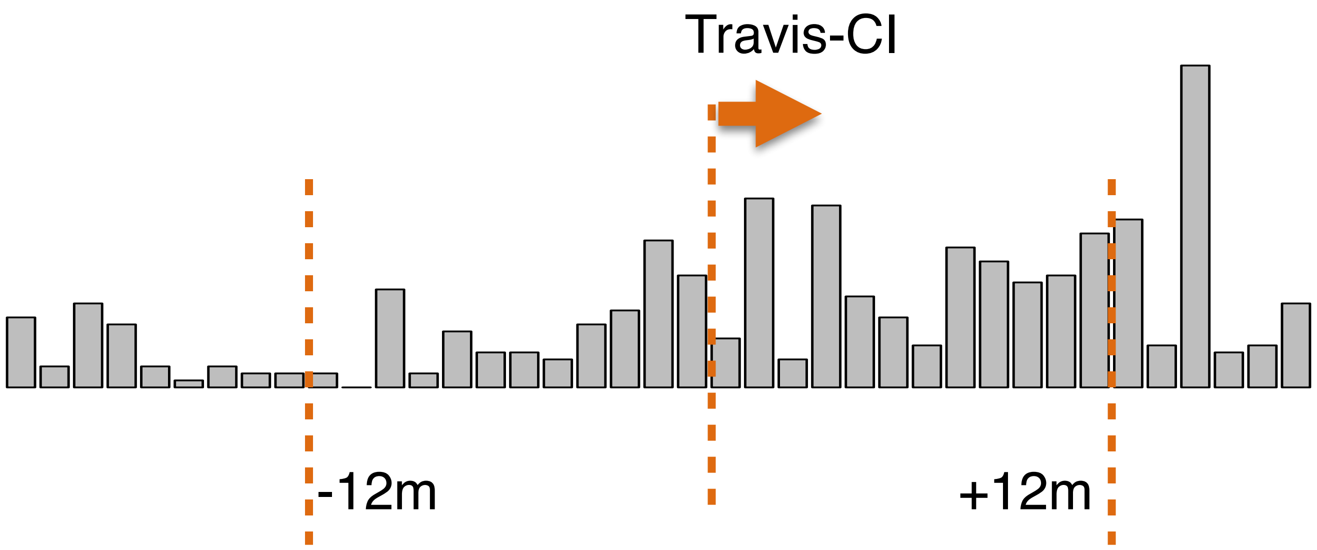


(1) How does CI affect team productivity?

Data Set

246 GitHub projects

- Not forks
- Ruby, Python, JavaScript, PHP, Java, Scala, C, C++
- 200+ pull requests
- Travis-CI

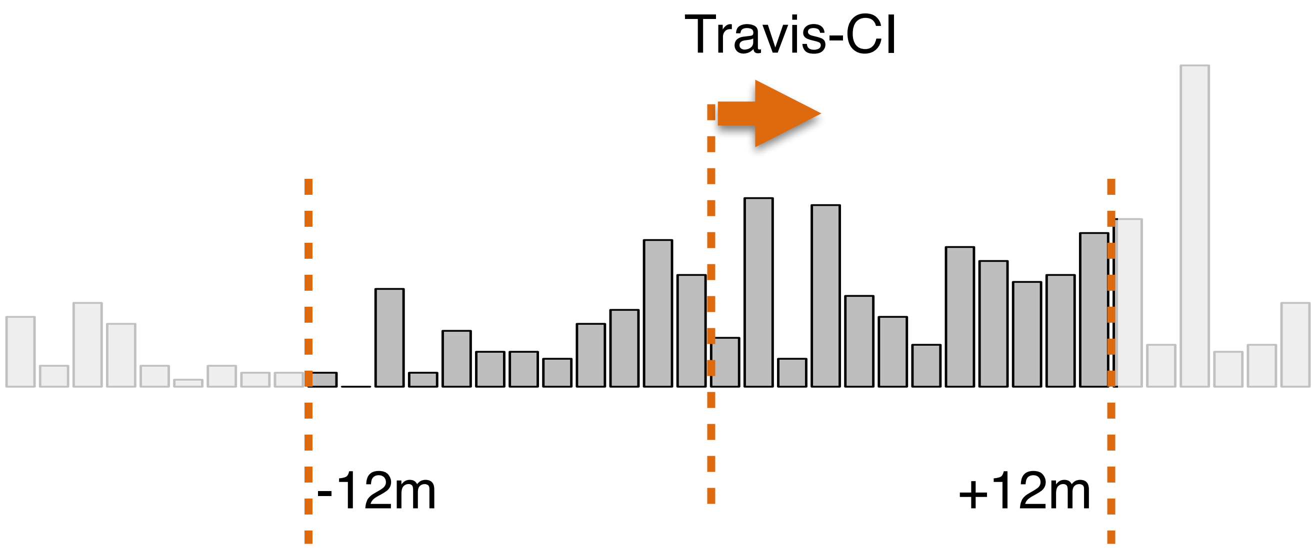


(1) How does CI affect team productivity?

Data Set

246 GitHub projects

- Not forks
- Ruby, Python, JavaScript, PHP, Java, Scala, C, C++
- 200+ pull requests
- Travis-CI

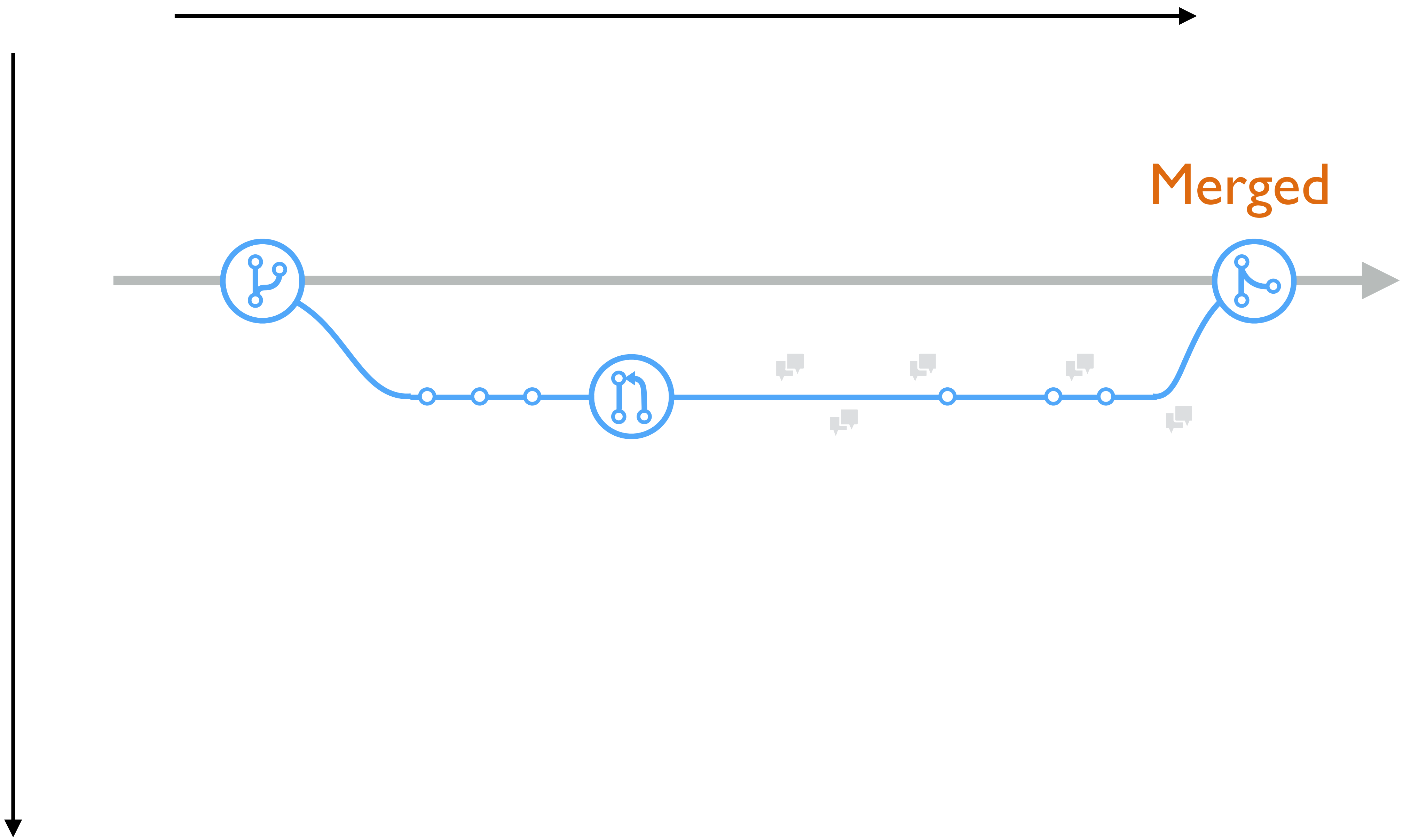


(1) How does CI affect team productivity?

Pull Requests

(1) How does CI affect team productivity?

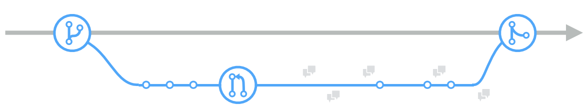
Pull Requests



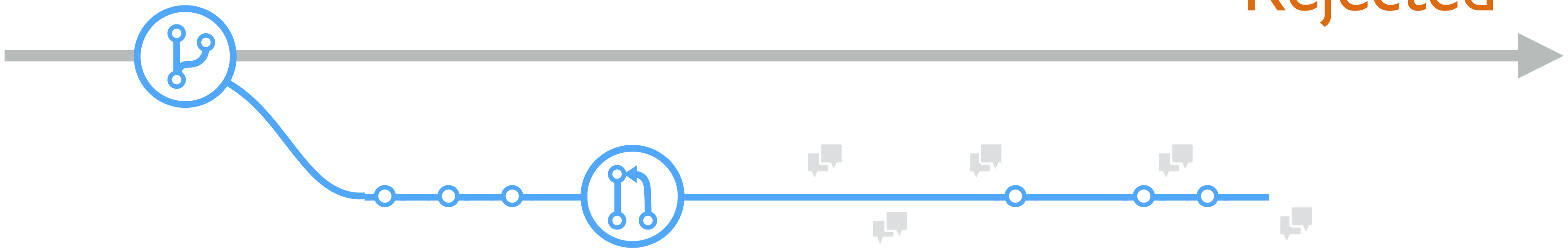
(1) How does CI affect team productivity?

Pull Requests

Merged



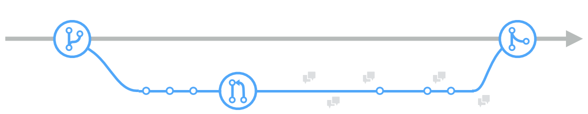
Rejected



(1) How does CI affect team productivity?

Pull Requests

Merged



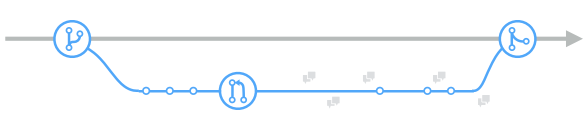
Rejected

(1) How does CI affect team productivity?

From insiders

Pull Requests

Merged



Rejected

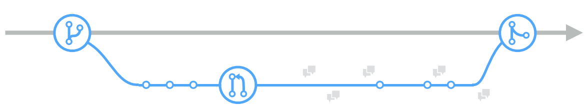
- Influence of social and technical factors for evaluating contribution in GitHub
  J Tsay, L Dabbish, J Herbsleb. *ICSE 2014*

(1) How does CI affect team productivity?

Pull Requests

From insiders | From outsiders

Merged



Rejected

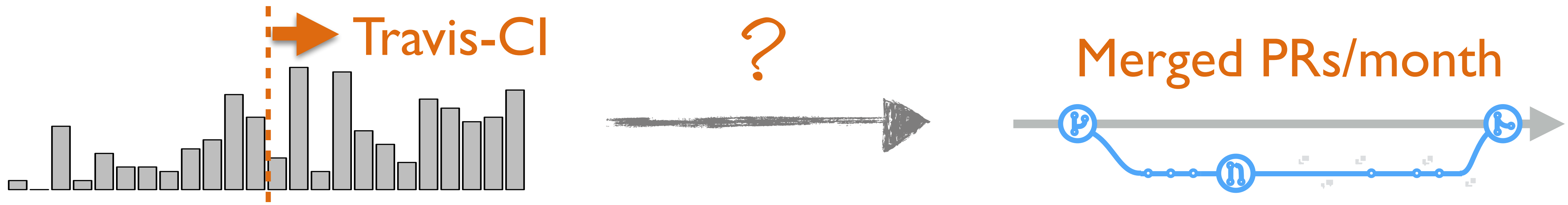
- Influence of social and technical factors for evaluating contribution in GitHub
  J Tsay, L Dabbish, J Herbsleb. *ICSE 2014*

(1) How does CI affect team productivity?

Pull Requests	From insiders	From outsiders
Merged	Regression model	Regression model
Rejected	Regression model	Regression model

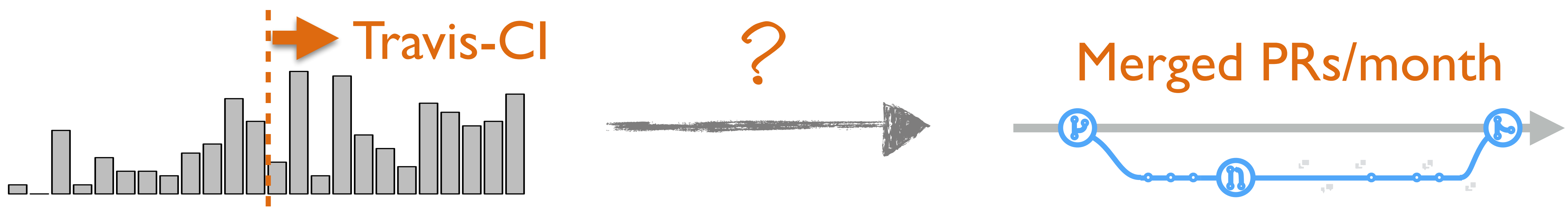
(1) How does CI affect team productivity?

Models



(1) How does CI affect team productivity?

Models



Project size

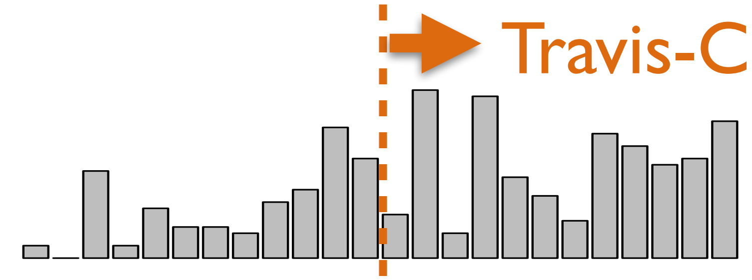
Project test suite size

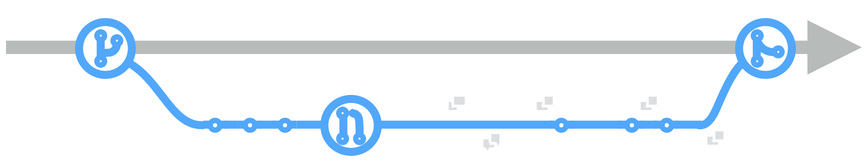
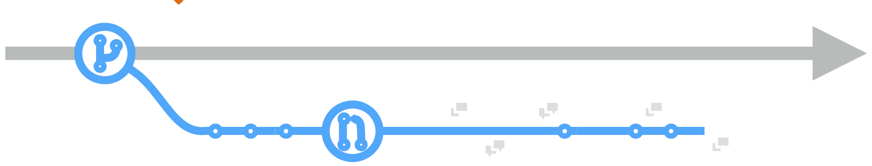
Team size

Project popularity

(1) How does CI affect team productivity?

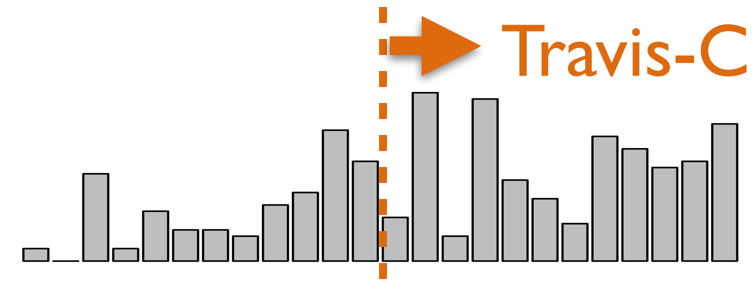
Results



	From insiders	From outsiders
Merged PRs/month 	+20.5%	-
Rejected PRs/month 	-42.3%	-26%

(1) How does CI affect team productivity?

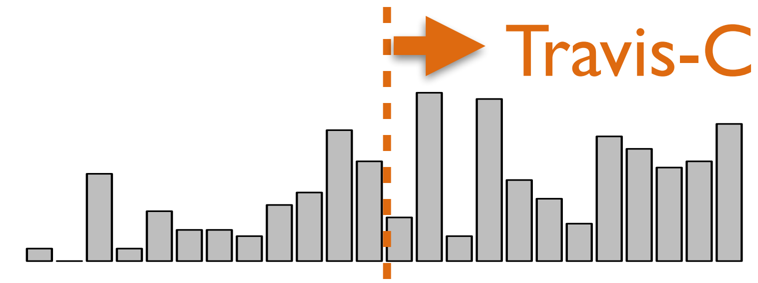
Results



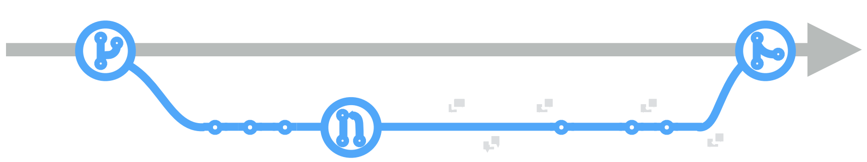
	From insiders	From outsiders
Merged PRs/month	+20.5%	-
Rejected PRs/month	-42.3%	-26%

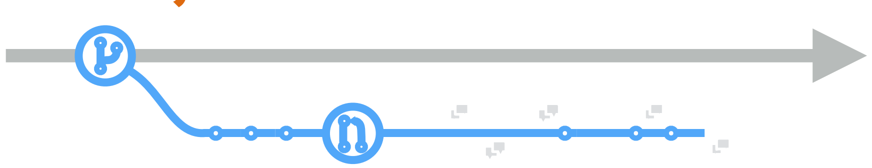
(1) How does CI affect team productivity?

Results



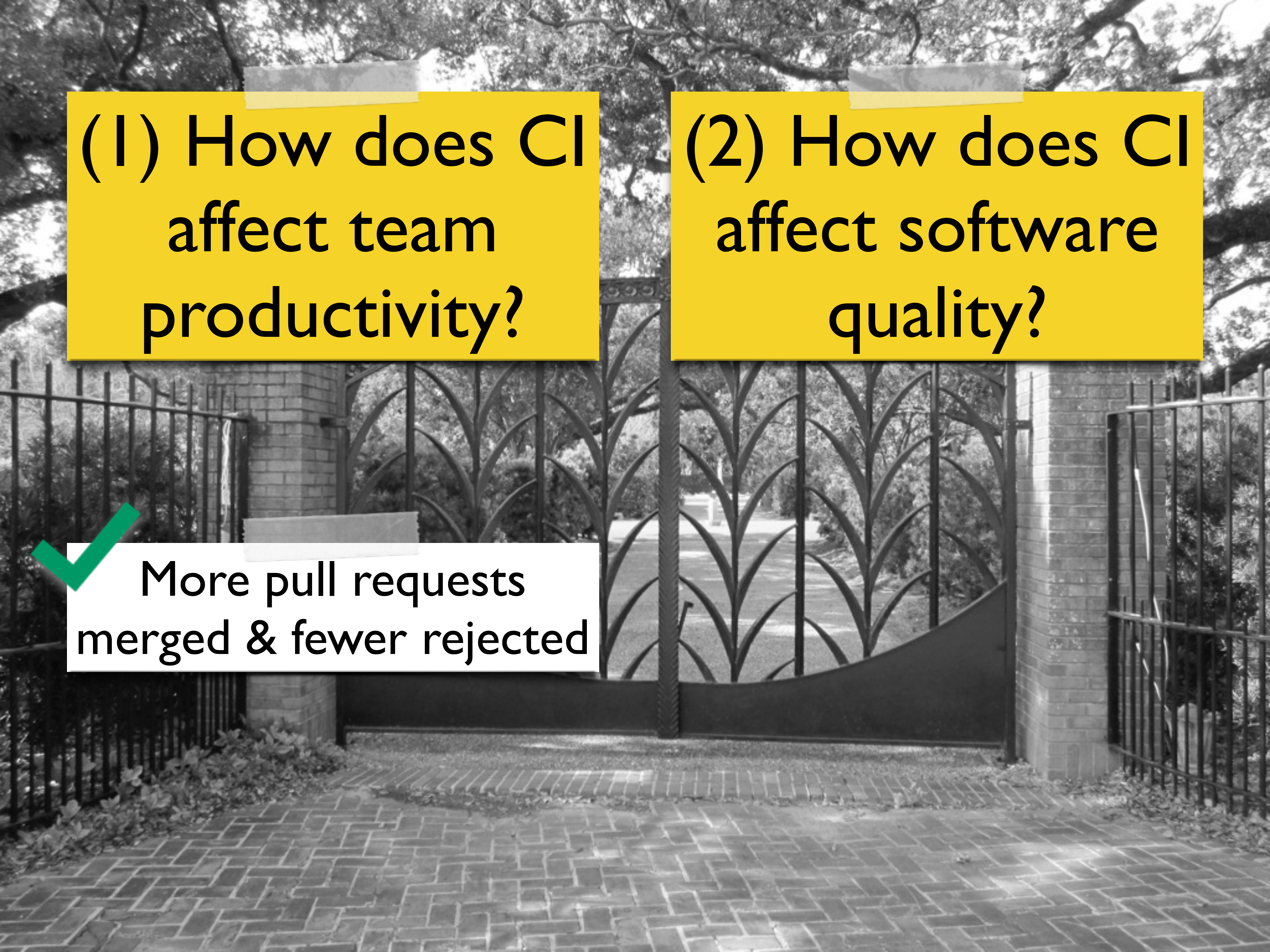
	From insiders	From outsiders
Merged PRs/month	+20.5%	-
Rejected PRs/month	-42.3%	-26%





(1) How does CI affect team productivity?

More pull requests merged & fewer rejected

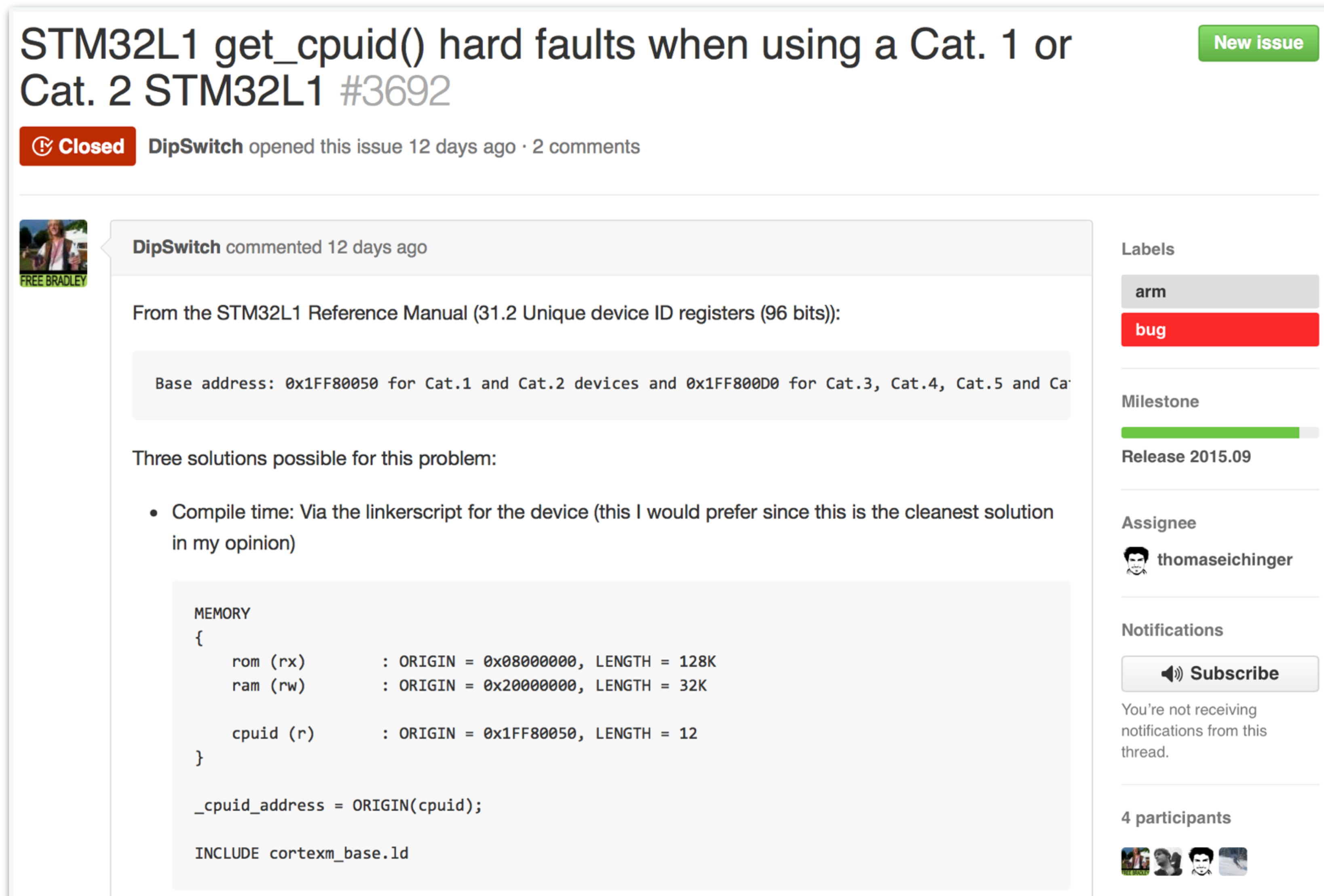

(1) How does CI affect team productivity?
(2) How does CI affect software quality?
More pull requests merged & fewer rejected

(2) How does CI affect software quality?

Data Set

42 GitHub projects

- Not forks
- Ruby, Python, JavaScript, PHP, Java, Scala, C, C++
- 200+ pull requests
- Travis-CI
- 100+ issues reported (75%+ tagged)

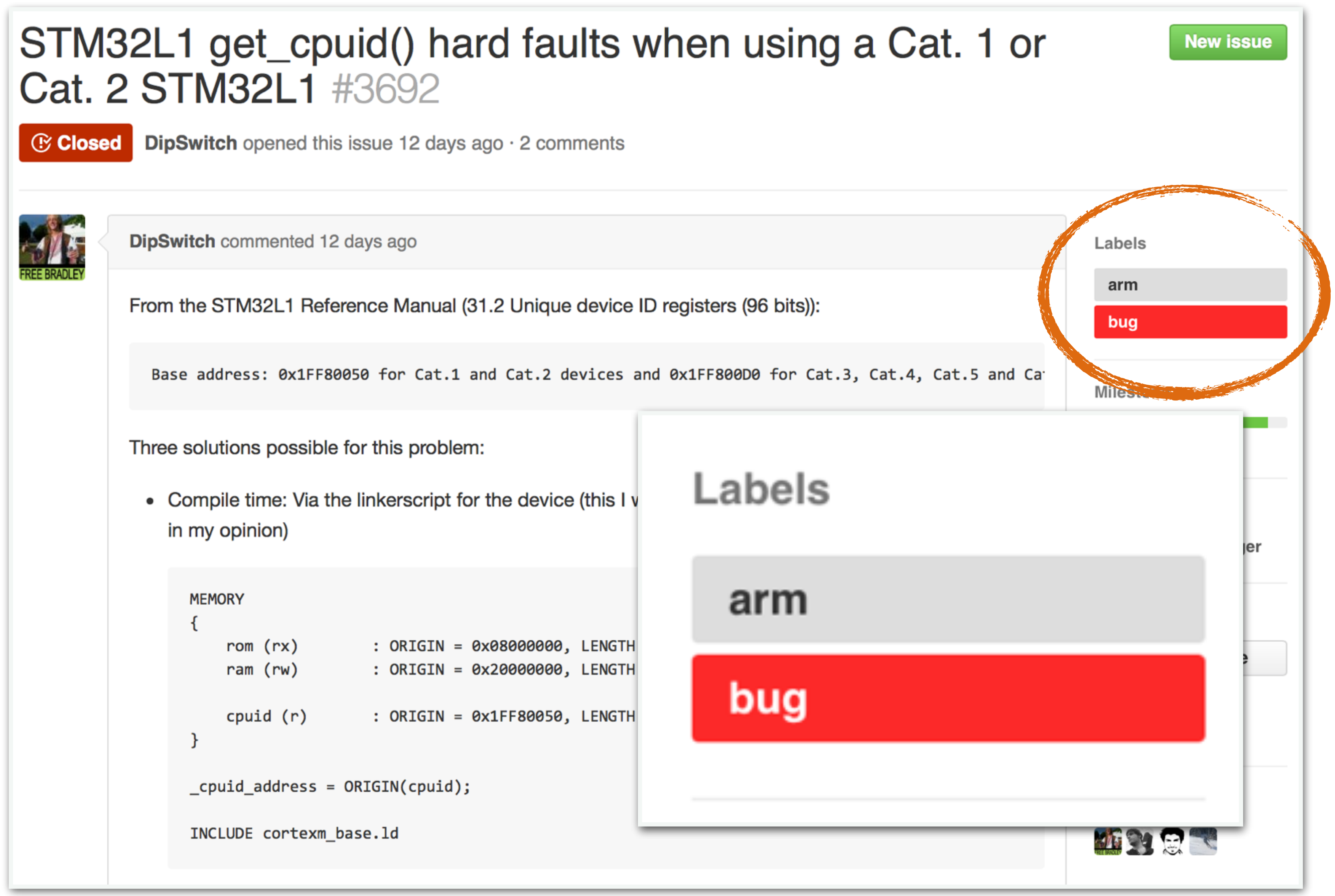


(2) How does CI affect software quality?

Data Set

42 GitHub projects

- Not forks
- Ruby, Python, JavaScript, PHP, Java, Scala, C, C++
- 200+ pull requests
- Travis-CI
- 100+ issues reported (75%+ tagged)



(2) How does CI affect software quality?

"Bug" reports

STM32L1 get_cpuid() hard faults when using a Cat. 1 or Cat. 2 STM32L1 #3692

New issue

Closed **DipSwitch** opened this issue 12 days ago · 2 comments

DipSwitch commented 12 days ago

From the STM32L1 Reference Manual (31.2 Unique device ID registers (96 bits)):

```
Base address: 0x1FF80050 for Cat.1 and Cat.2 devices and 0x1FF800D0 for Cat.3, Cat.4, Cat.5 and Ca
```

Three solutions possible for this problem:

- Compile time: Via the linkerscript for the device (this I would prefer since this is the cleanest solution in my opinion)

```
MEMORY
{
    rom (rx)        : ORIGIN = 0x08000000, LENGTH = 128K
    ram (rw)        : ORIGIN = 0x20000000, LENGTH = 32K

    cpuid (r)       : ORIGIN = 0x1FF80050, LENGTH = 12
}

_cpuid_address = ORIGIN(cpuid);

INCLUDE cortexm_base.ld
```

Labels

arm

bug

Milestone

Release 2015.09

Assignee

thomaseichinger

Notifications

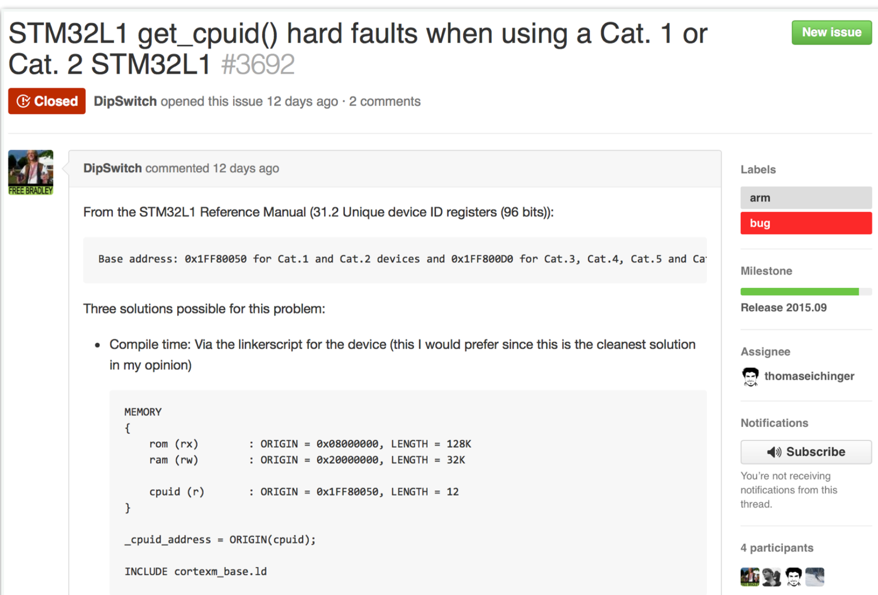
Subscribe

You're not receiving notifications from this thread.

4 participants

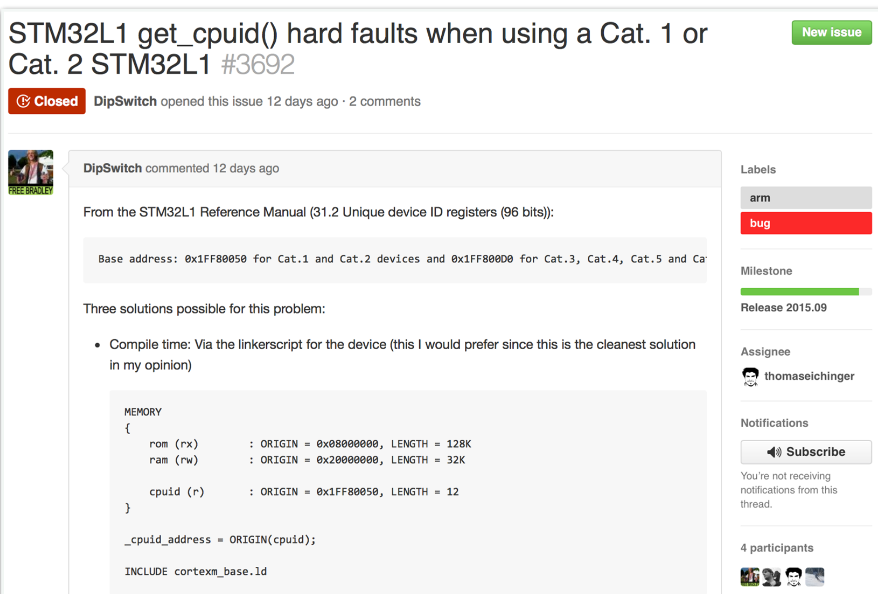
(2) How does CI affect software quality?

“Bug” reports



(2) How does CI affect software quality?

"Bug" reports



STM32L1 get_cpuid() hard faults when using a Cat. 1 or Cat. 2 STM32L1 #3692

New issue

Closed DipSwitch opened this issue 12 days ago · 2 comments

DipSwitch commented 12 days ago

From the STM32L1 Reference Manual (31.2 Unique device ID registers (96 bits)):

```
Base address: 0x1FF80050 for Cat.1 and Cat.2 devices and 0x1FF800D0 for Cat.3, Cat.4, Cat.5 and Ca
```

Three solutions possible for this problem:

- Compile time: Via the linkerscript for the device (this I would prefer since this is the cleanest solution in my opinion)

```
MEMORY
{
    rom (rx)        : ORIGIN = 0x08000000, LENGTH = 128K
    ram (rw)        : ORIGIN = 0x20000000, LENGTH = 32K

    cpuid (r)       : ORIGIN = 0x1FF80050, LENGTH = 12
}

_cpuid_address = ORIGIN(cpuid);

INCLUDE cortexm_base.ld
```

Labels
arm
bug

Milestone
Release 2015.09

Assignee
thomaseichinger

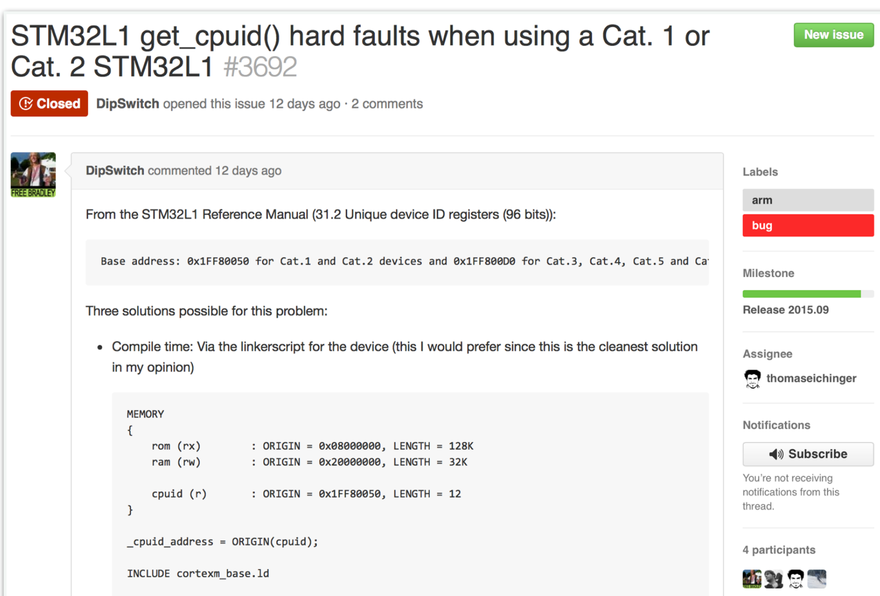
Notifications
Subscribe
You're not receiving notifications from this thread.

4 participants

(2) How does CI affect software quality?

From insiders

“Bug” reports


STM32L1 get_cpuid() hard faults when using a Cat. 1 or Cat. 2 STM32L1 #3692

New issue

Closed DipSwitch opened this issue 12 days ago · 2 comments

DipSwitch commented 12 days ago

From the STM32L1 Reference Manual (31.2 Unique device ID registers (96 bits)):

```
Base address: 0x1FF80050 for Cat.1 and Cat.2 devices and 0x1FF800D0 for Cat.3, Cat.4, Cat.5 and Ca
```

Three solutions possible for this problem:

- Compile time: Via the linkerscript for the device (this I would prefer since this is the cleanest solution in my opinion)

```
MEMORY
{
    rom (rx)        : ORIGIN = 0x08000000, LENGTH = 128K
    ram (rw)        : ORIGIN = 0x20000000, LENGTH = 32K

    cpuid (r)       : ORIGIN = 0x1FF80050, LENGTH = 12
}

_cpuid_address = ORIGIN(cpuid);

INCLUDE cortexm_base.ld
```

Labels

arm

bug

Milestone

Release 2015.09

Assignee

thomaseichinger

Notifications

Subscribe

You're not receiving notifications from this thread.

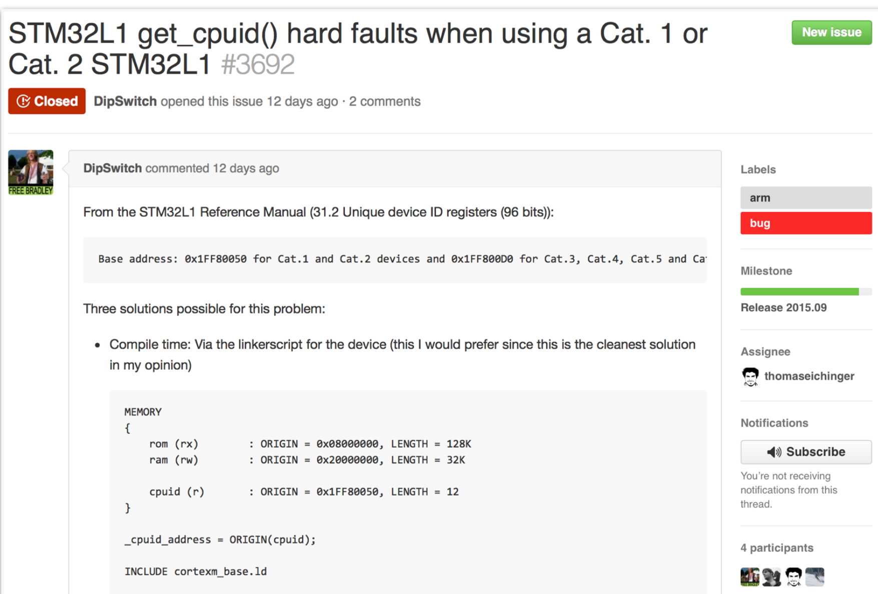
4 participants

(2) How does CI affect software quality?

From insiders

From outsiders

“Bug” reports


STM32L1 get_cpuid() hard faults when using a Cat. 1 or Cat. 2 STM32L1 #3692

New issue

Closed DipSwitch opened this issue 12 days ago · 2 comments

DipSwitch commented 12 days ago

From the STM32L1 Reference Manual (31.2 Unique device ID registers (96 bits)):

```
Base address: 0x1FF80050 for Cat.1 and Cat.2 devices and 0x1FF800D0 for Cat.3, Cat.4, Cat.5 and Ca
```

Three solutions possible for this problem:

- Compile time: Via the linkerscript for the device (this I would prefer since this is the cleanest solution in my opinion)

```
MEMORY
{
    rom (rx)        : ORIGIN = 0x08000000, LENGTH = 128K
    ram (rw)        : ORIGIN = 0x20000000, LENGTH = 32K

    cpuid (r)       : ORIGIN = 0x1FF80050, LENGTH = 12
}

_cpuid_address = ORIGIN(cpuid);

INCLUDE cortexm_base.ld
```

Labels

arm

bug

Milestone

Release 2015.09

Assignee

thomaseichinger

Notifications

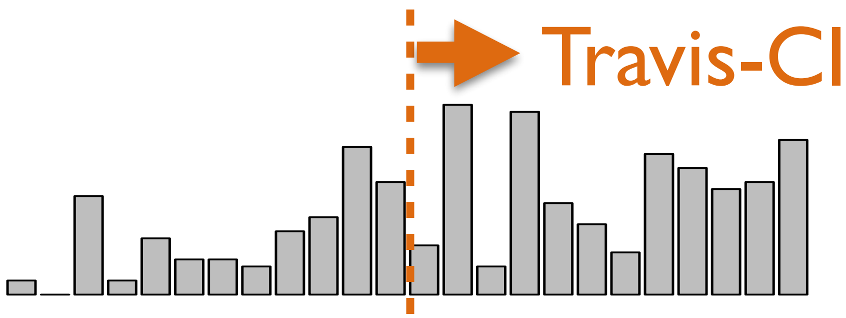
Subscribe

You're not receiving notifications from this thread.

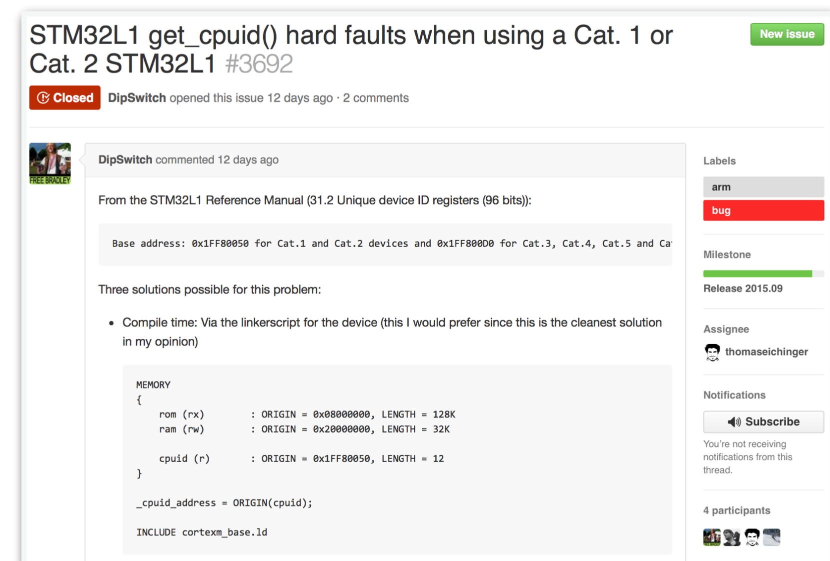
4 participants

(2) How does CI affect software quality?

Models



“Bug” reports/month


STM32L1 get_cpuid() hard faults when using a Cat. 1 or Cat. 2 STM32L1 #3692

New issue

Closed DipSwitch opened this issue 12 days ago · 2 comments

DipSwitch commented 12 days ago

From the STM32L1 Reference Manual (31.2 Unique device ID registers (96 bits)):

```
Base address: 0x1FF80050 for Cat.1 and Cat.2 devices and 0x1FF800D0 for Cat.3, Cat.4, Cat.5 and Ca
```

Three solutions possible for this problem:

- Compile time: Via the linkerscript for the device (this I would prefer since this is the cleanest solution in my opinion)

```
MEMORY
{
    rom (rx)        : ORIGIN = 0x08000000, LENGTH = 128K
    ram (rw)        : ORIGIN = 0x20000000, LENGTH = 32K

    cpuid (r)       : ORIGIN = 0x1FF80050, LENGTH = 12
}

_cpuid_address = ORIGIN(cpuid);

INCLUDE cortexm_base.ld
```

Labels

arm

bug

Milestone

Release 2015.09

Assignee

thomaseichinger

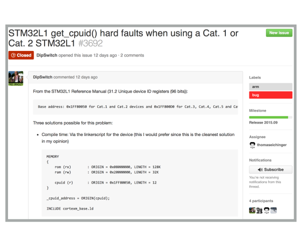
Notifications

Subscribe

You're not receiving notifications from this thread.

4 participants

Non-bug issues



Project size

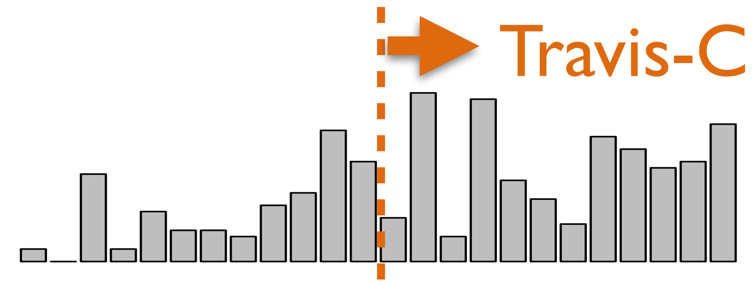
Project test suite size

Team size

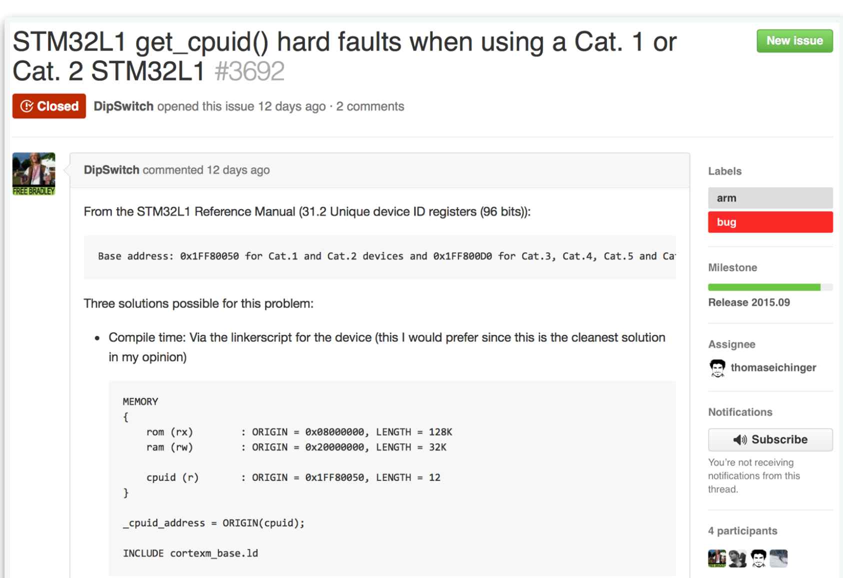
Project popularity

(2) How does CI affect software quality?

Results



	From insiders	From outsiders
“Bug” reports/month	+48%	-

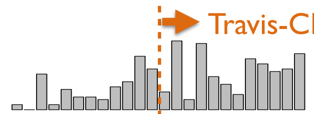


CONTINUOUS INTEGRATION IN GITHUB

QUALITY AND PRODUCTIVITY OUTCOMES

(1) How does CI affect team productivity?

Results

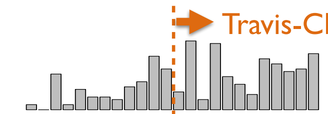


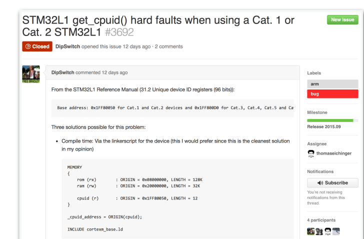
	From insiders	From outsiders
Merged PRs/month 	+20.5%	-
Rejected PRs/month 	-42.3%	-26%

More pull requests merged & fewer rejected

(2) How does CI affect software quality?

Results



	From insiders	From outsiders
"Bug" reports/month 	+48%	-

More bugs reported monthly by <u>core dev's</u>

No impact on bugs reported by <u>externals</u>


Bogdan Vasilescu


Yue Yu


Prem Devanbu


Vladimir Filkov

UCDAVIS UNIVERSITY OF CALIFORNIA

国防科学技术大学 National University of Defense Technology