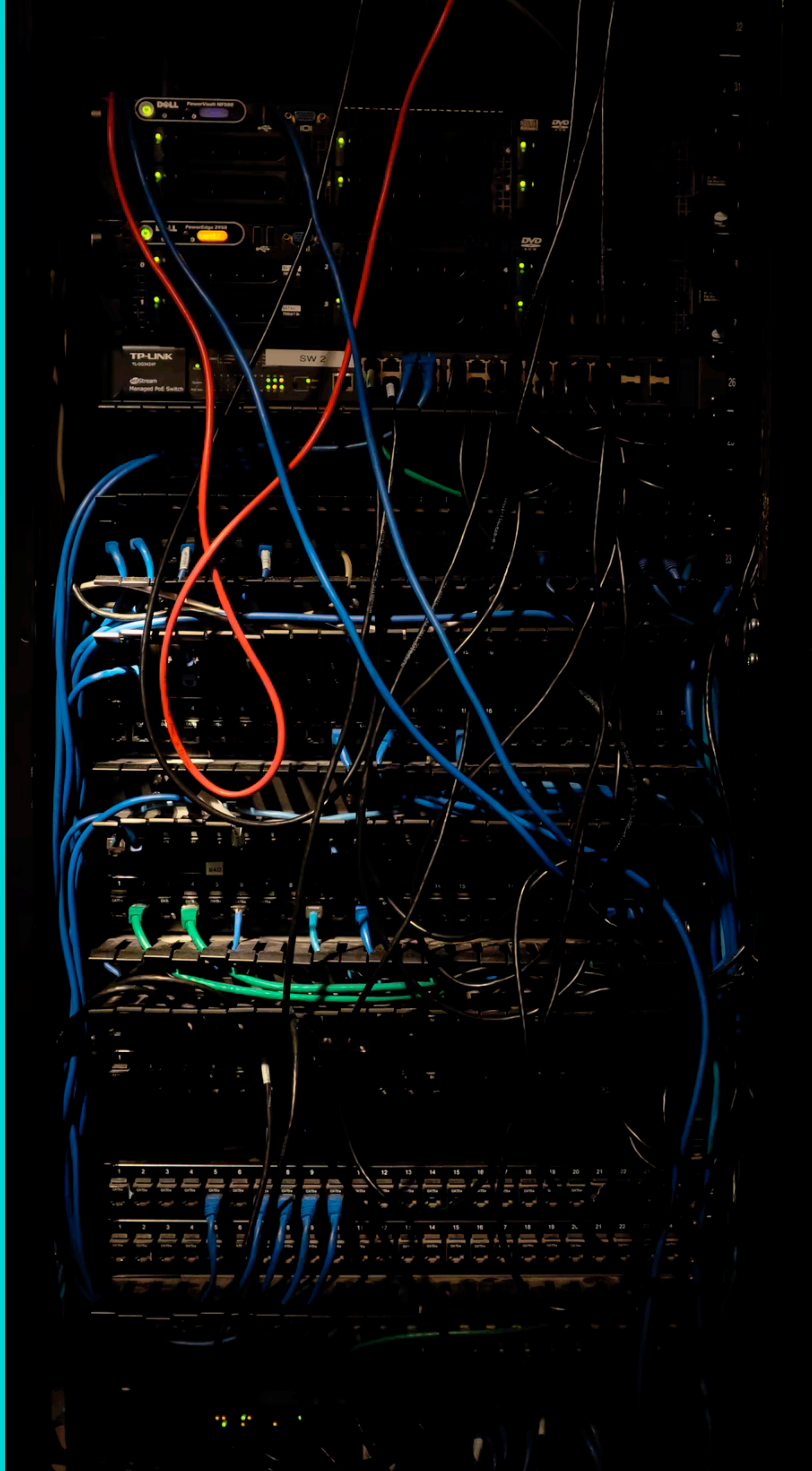
CONTINUOUS INTEGRATION

TECH & TOOLING - CI/CD

CSC491 | UTORONTO



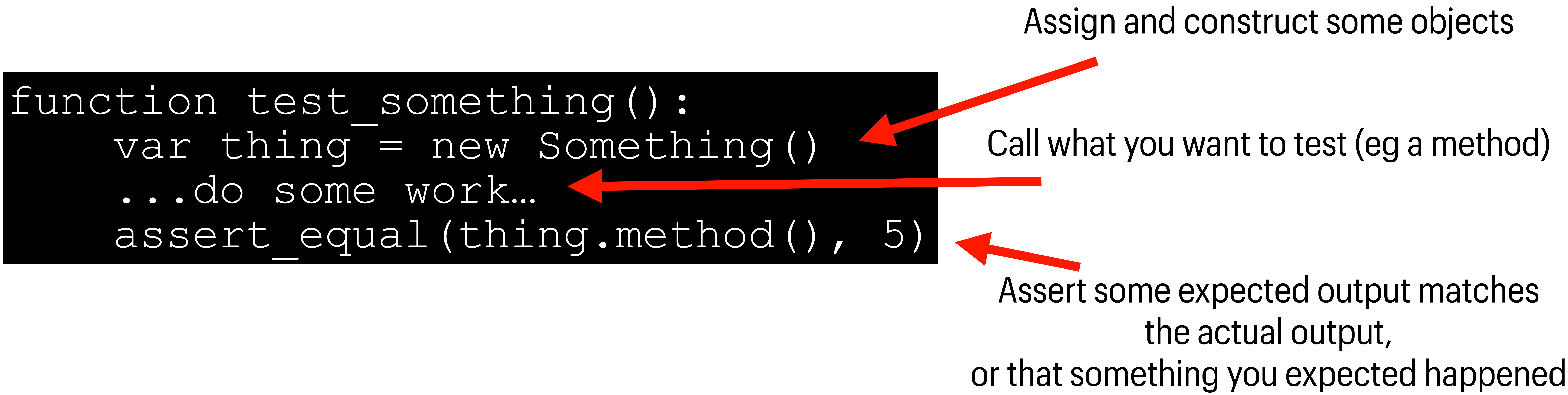
IN THIS LECTURE

1. We will introduce testing concepts and explain common patterns companies run these tests
2. Introduce what Continuous Integration (CI) systems are
3. Explain how CI systems work
4. Demonstrate how to integrate GitHub Actions into your project

TO FIRST UNDERSTAND WHAT CONTINUOUS INTEGRATION IS, WE NEED TO UNDERSTAND HOW SOFTWARE IS TESTED

TESTING YOUR APPLICATION

The exact specifics in testing your application depends on the language and frameworks chosen, however the fundamental concept remains the same



```
function test_something():
    var thing = new Something()
    ...do some work…
    assert_equal(thing.method(), 5)
```

Assign and construct some objects

Call what you want to test (eg a method)

Assert some expected output matches
the actual output,
or that something you expected happened

TEST CASE

```
function test_something():
    var thing = new Something()
    ...do some work…
    assert_equal(thing.method(), 5)
```

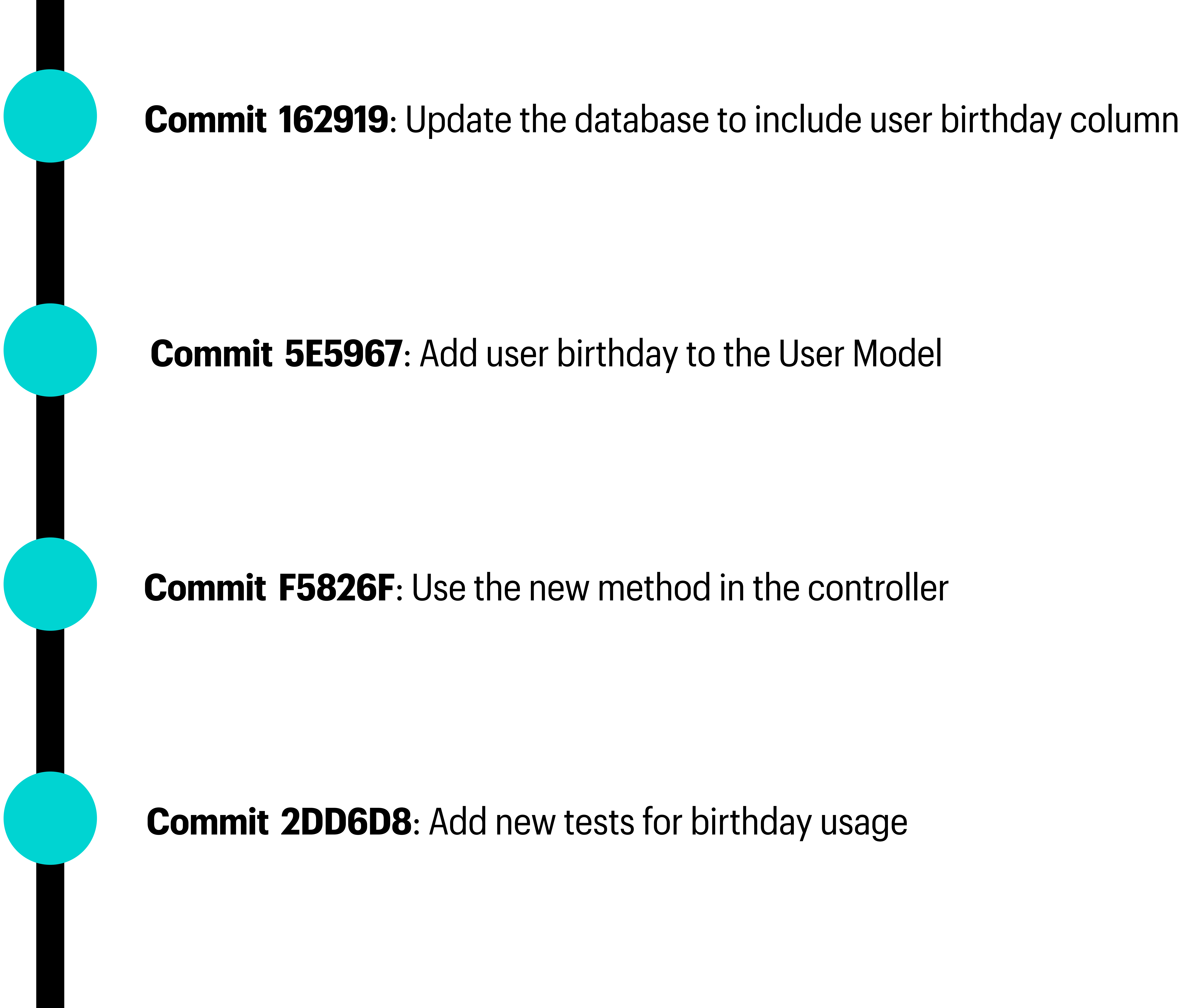
TEST SUITE

```
function test_something():
    var thing = new Something()
    ...do some work…
    assert_equal(thing.method(), 5)

function test_other_something():
    var thing = new OtherThing()
    ...do some work…
    assert_equal(thing.method(), 5)

function test_something_else():
    var thing = new SomethingElse()
    ...do some work…
    assert_equal(thing.method(), 5)
```

HOW ARE TESTS RUN?

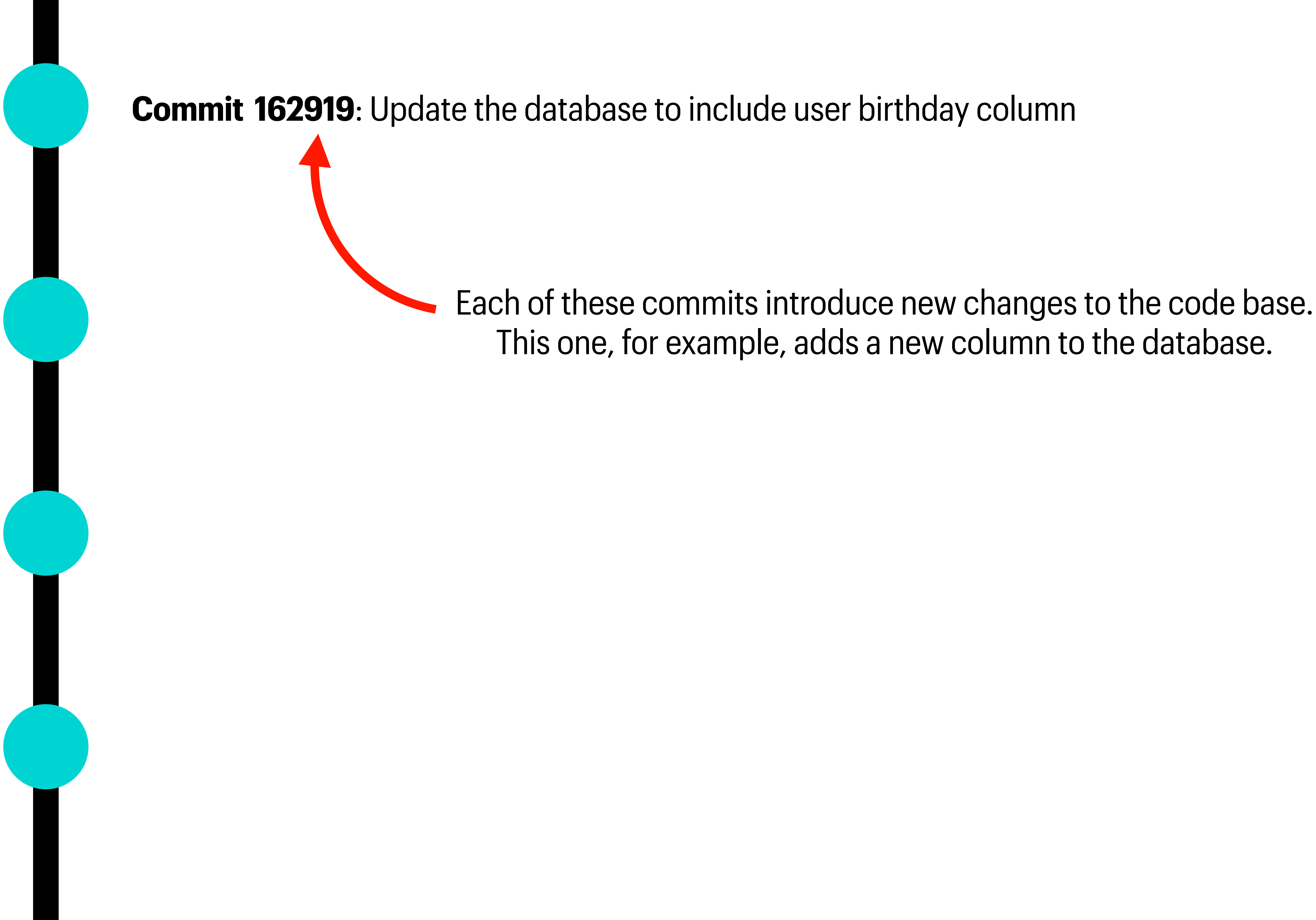


Commit 162919: Update the database to include user birthday column

Commit 5E5967: Add user birthday to the User Model

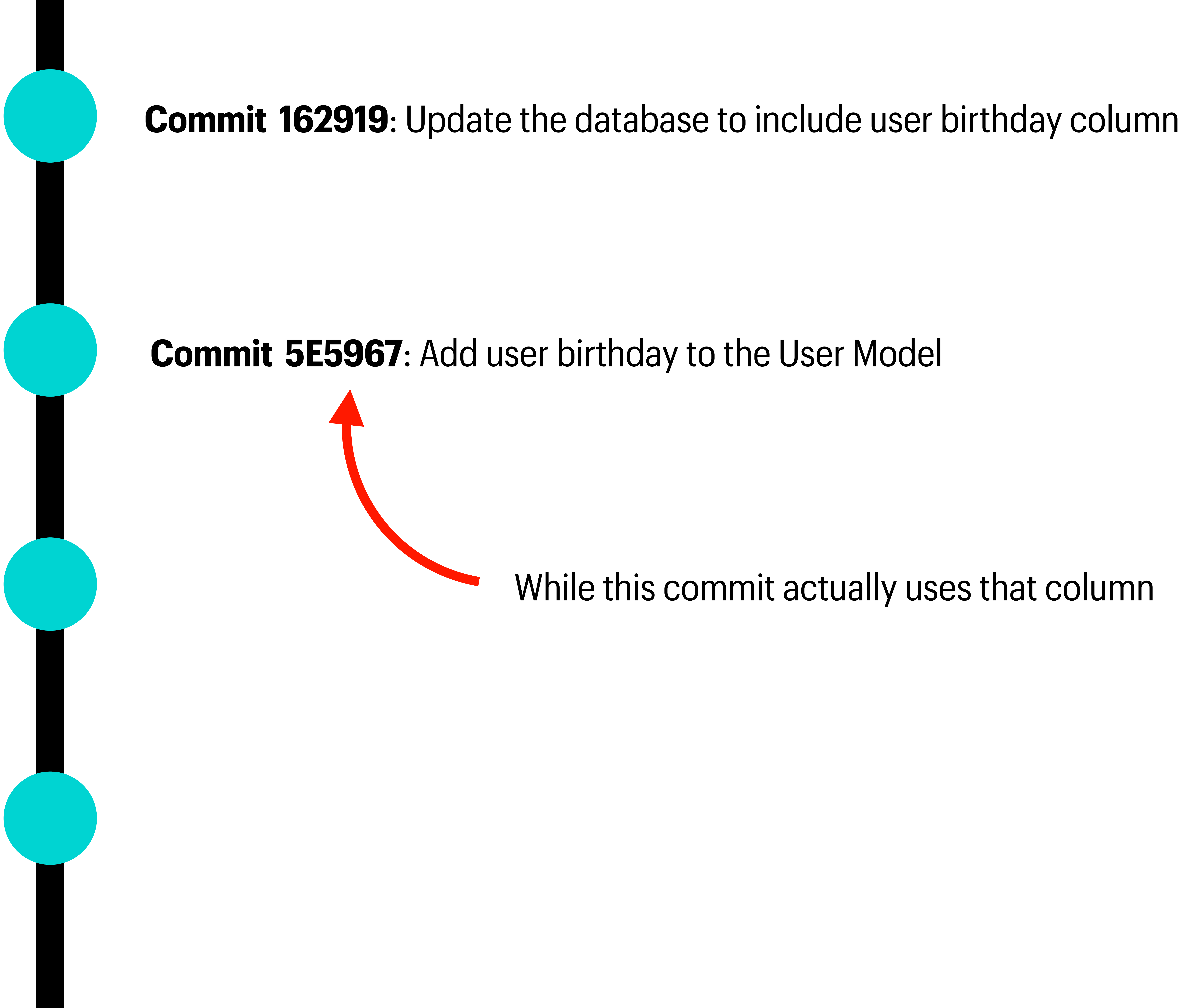
Commit F5826F: Use the new method in the controller

Commit 2DD6D8: Add new tests for birthday usage



Commit 162919: Update the database to include user birthday column

Each of these commits introduce new changes to the code base.
This one, for example, adds a new column to the database.



Commit 162919: Update the database to include user birthday column

Commit 5E5967: Add user birthday to the User Model

While this commit actually uses that column

Along each of these commits, code and configuration has changed.

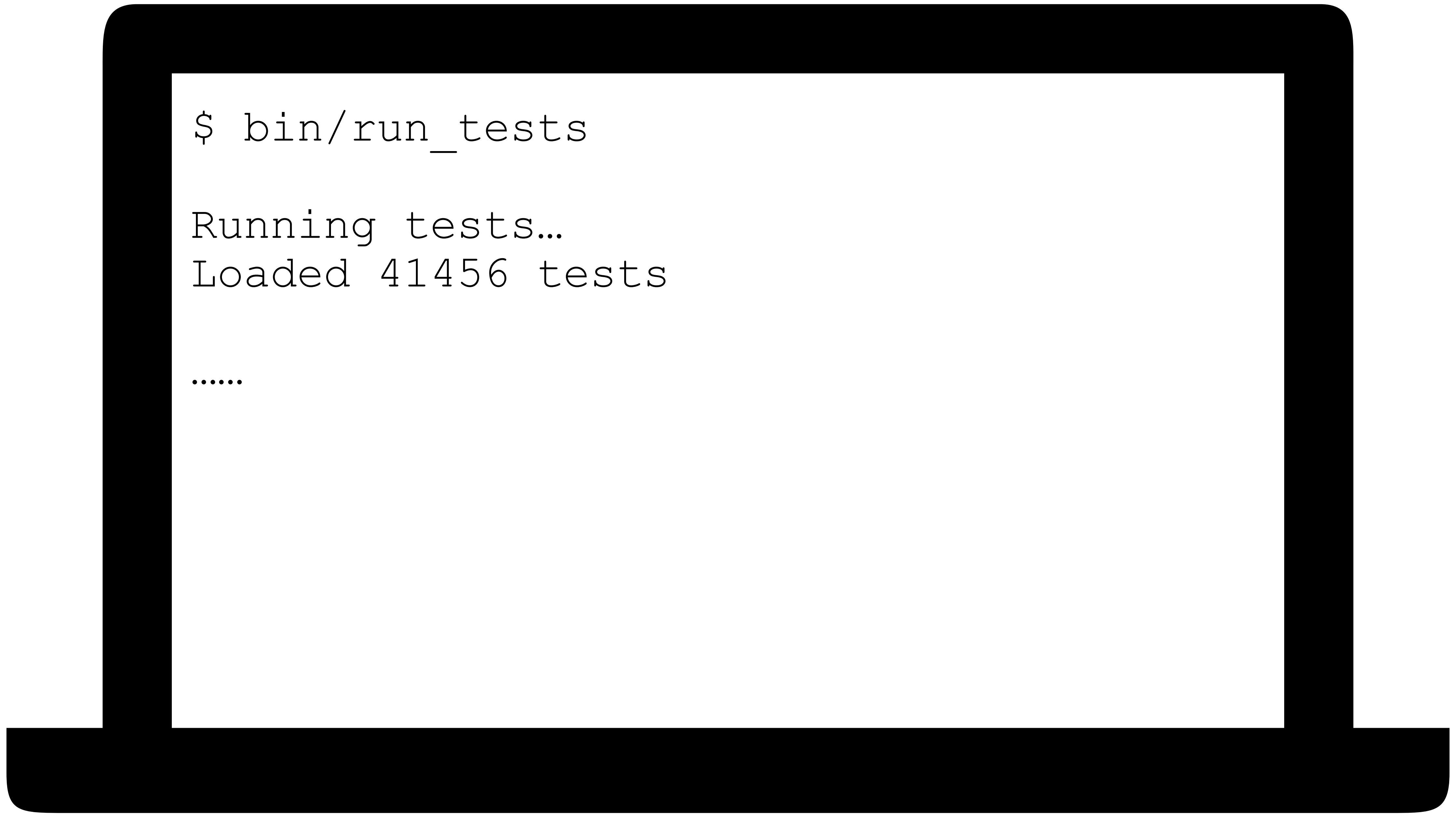
When the code changes, there is a chance something will break that you are not aware of.

Luckily, if you have good test coverage, you can run your test suites to be more confident that nothing has broken.

That is why it is recommended to run your tests on *every push and commit into your repository*. That way if something fails, you know when it failed and can fix it more easily (and prevent it from being deployed).

HOW DO YOU RUN THESE TESTS ON EVERY CHANGE?

YOU COULD (AND SHOULD) RUN WHAT YOU CAN BEFORE PUSHING

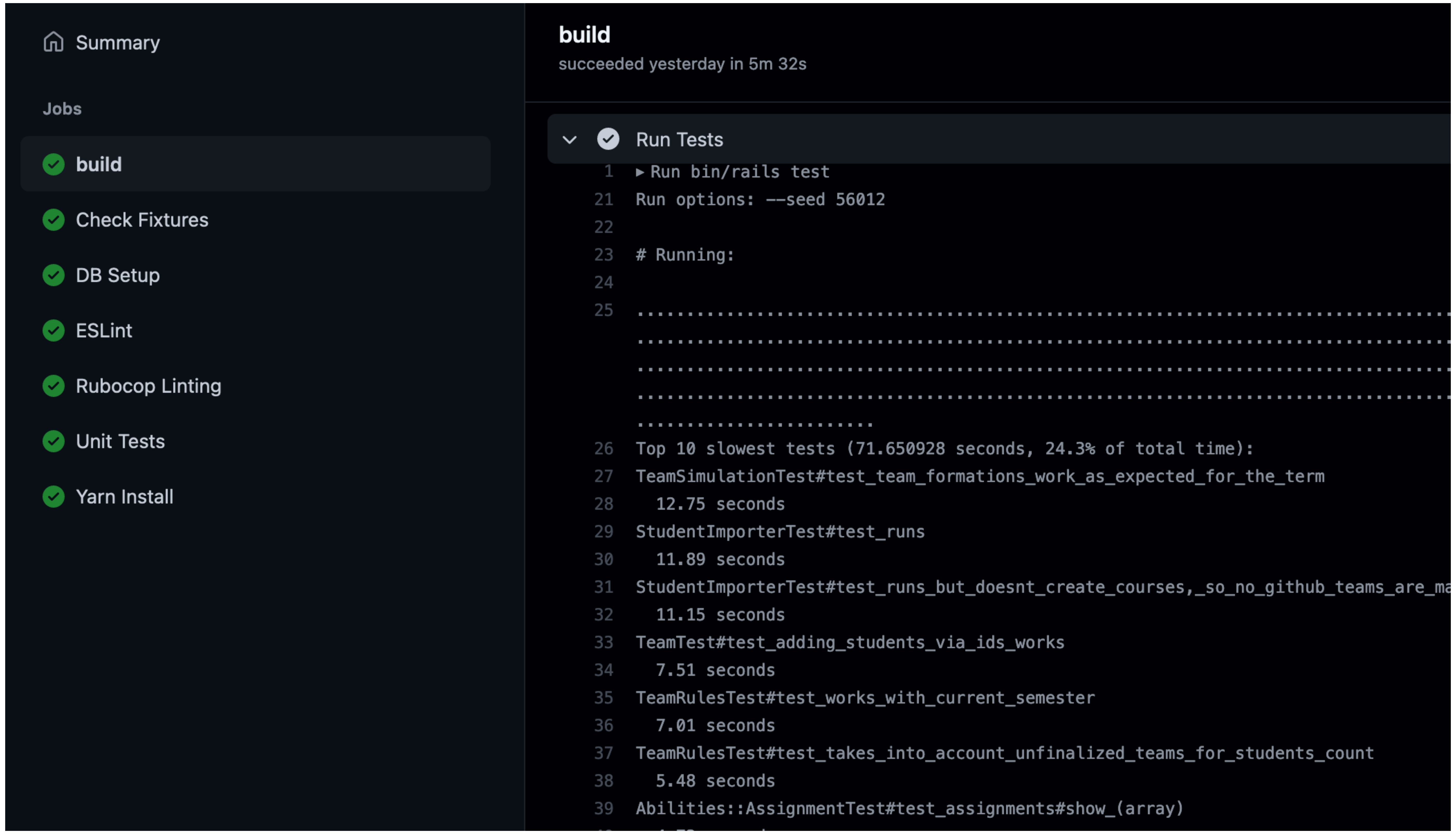


```
$ bin/run_tests

Running tests…
Loaded 41456 tests

……
```

BUT YOU SHOULD AUTOMATICALLY RUN THE TESTS ON EVERY CHANGE
AND BLOCK FAILING COMMITS FROM MERGING TO MAIN



build

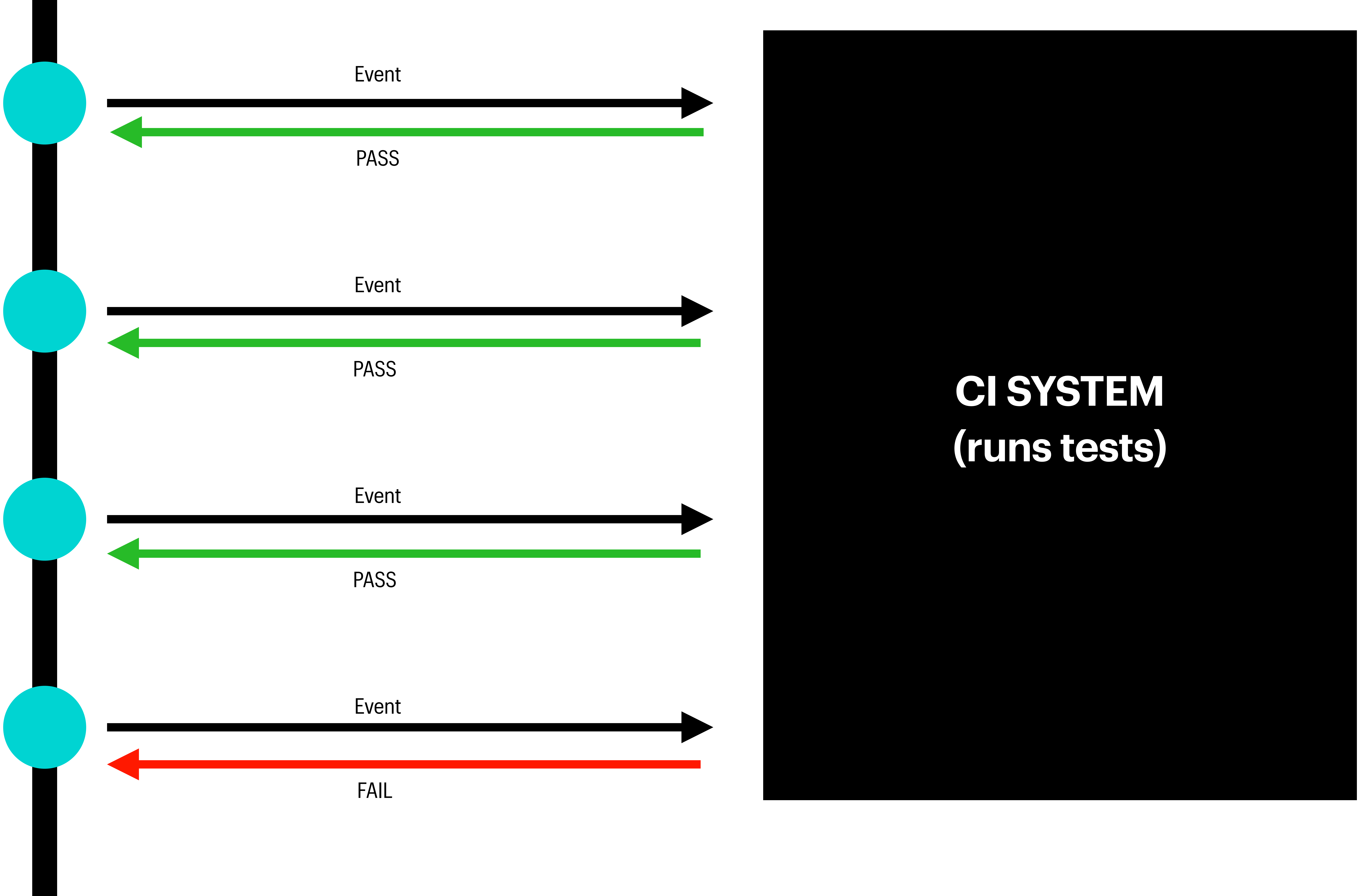
succeeded yesterday in 5m 32s

```
▶ Run bin/rails test
Run options: --seed 56012

# Running:

...............................................................................
...............................................................................
...............................................................................
...............................................................................
........................
Top 10 slowest tests (71.650928 seconds, 24.3% of total time):
TeamSimulationTest#test_team_formations_work_as_expected_for_the_term
  12.75 seconds
StudentImporterTest#test_runs
  11.89 seconds
StudentImporterTest#test_runs_but_doesnt_create_courses,_so_no_github_teams_are_ma
  11.15 seconds
TeamTest#test_adding_students_via_ids_works
  7.51 seconds
TeamRulesTest#test_works_with_current_semester
  7.01 seconds
TeamRulesTest#test_takes_into_account_unfinalized_teams_for_students_count
  5.48 seconds
Abilities::AssignmentTest#test_assignments#show_(array)
```

HOW DO CI SYSTEMS WORK?


Event
PASS
Event
PASS
Event
PASS
Event
FAIL
CI SYSTEM
(runs tests)

REPORTER
WORKER 1
WORKER 2
WORKER 3
Incoming Event
EVENT RECEIVER
SCHEDULER

REPORTER
WORKER 1
WORKER 2
WORKER 3
Incoming Event
EVENT RECEIVER
Schedules a Build
SCHEDULER

REPORTER
WORKER 1
WORKER 2
WORKER 3
Incoming Event
EVENT RECEIVER
Schedules a Build
SCHEDULER

QUEUE

Build 1 (Linux)
Build 2 (Mac)
Build 3 (Linux)
Build 4 (Android)

REPORTER

WORKER 1

WORKER 2

WORKER 3

Sends work to the worker pool
based on criteria of the build

Incoming Event

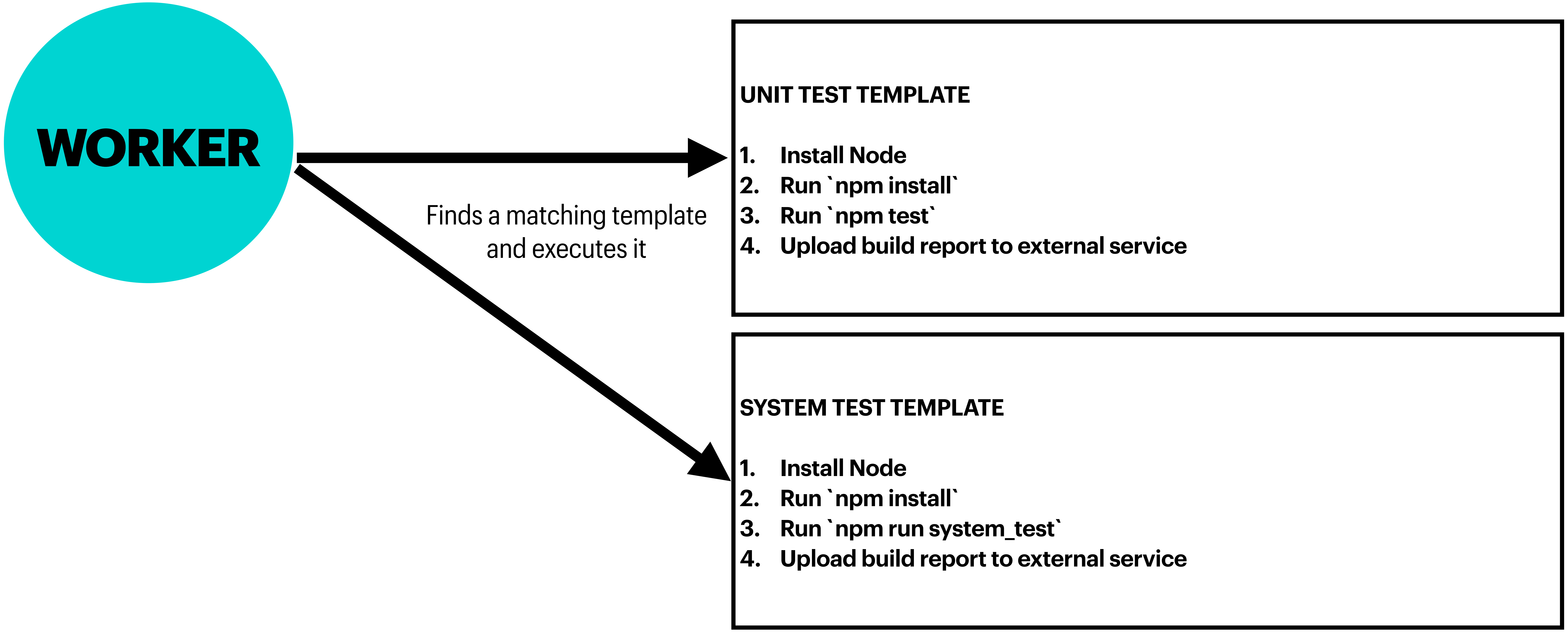
EVENT RECEIVER

Schedules a Build

SCHEDULER

QUEUE

Build 1 (Linux)
Build 2 (Mac)
Build 3 (Linux)
Build 4 (Android)



UNIT TEST TEMPLATE

1. Install Node
2. Run `npm install`
3. Run `npm test`
4. Upload build report to external service

SYSTEM TEST TEMPLATE

1. Install Node
2. Run `npm install`
3. Run `npm run system_test`
4. Upload build report to external service

Sends conclusion and
details to host (eg GitHub)

REPORTER

Report Build
Conclusion

WORKER
1

WORKER
2

WORKER
3

Sends work to the worker pool
based on criteria of the build

Incoming Event

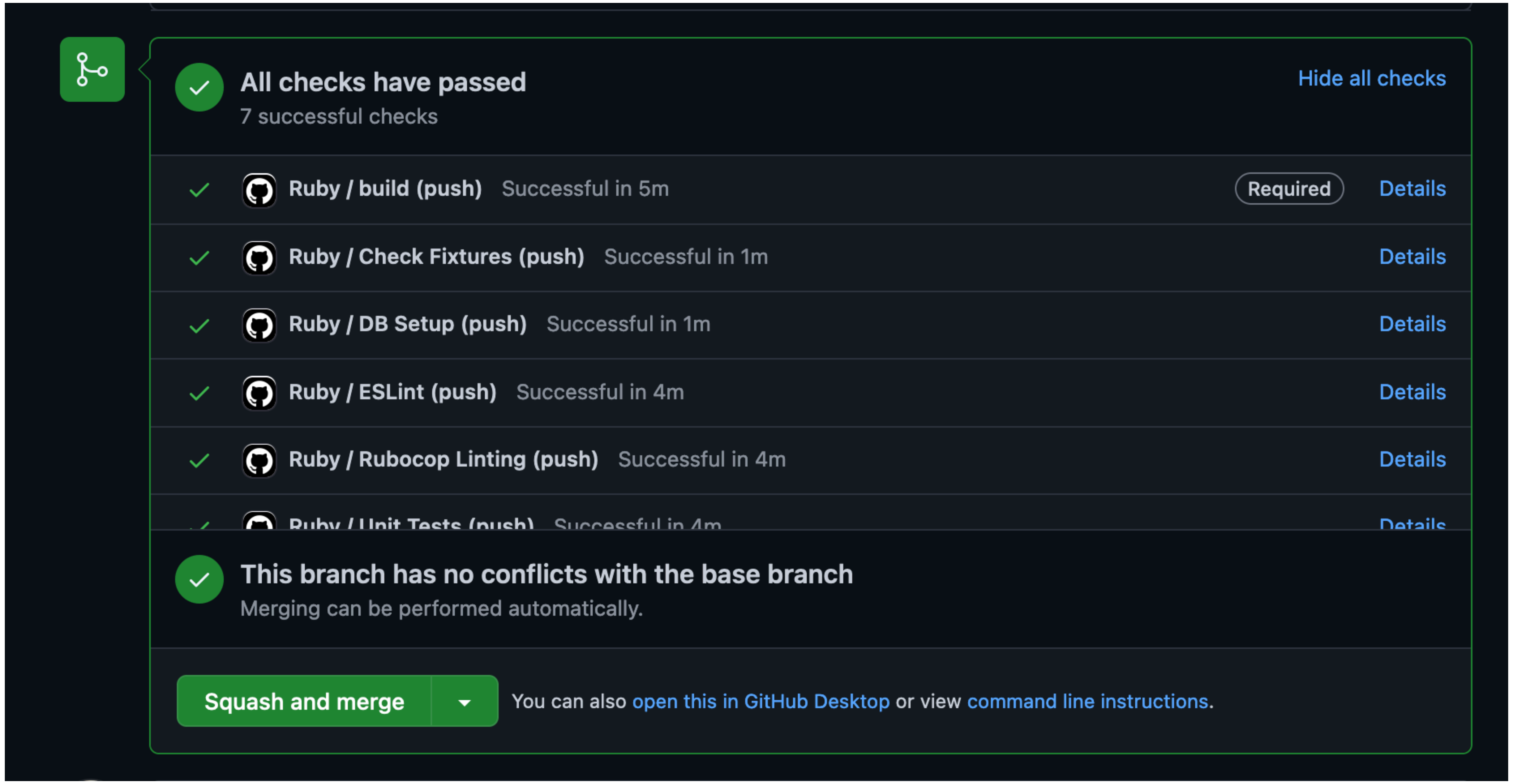
EVENT
RECEIVER

Schedules a Build

SCHEDULER

QUEUE

Build 1 (Linux)
Build 2 (Mac)
Build 3 (Linux)
Build 4 (Android)



All checks have passed

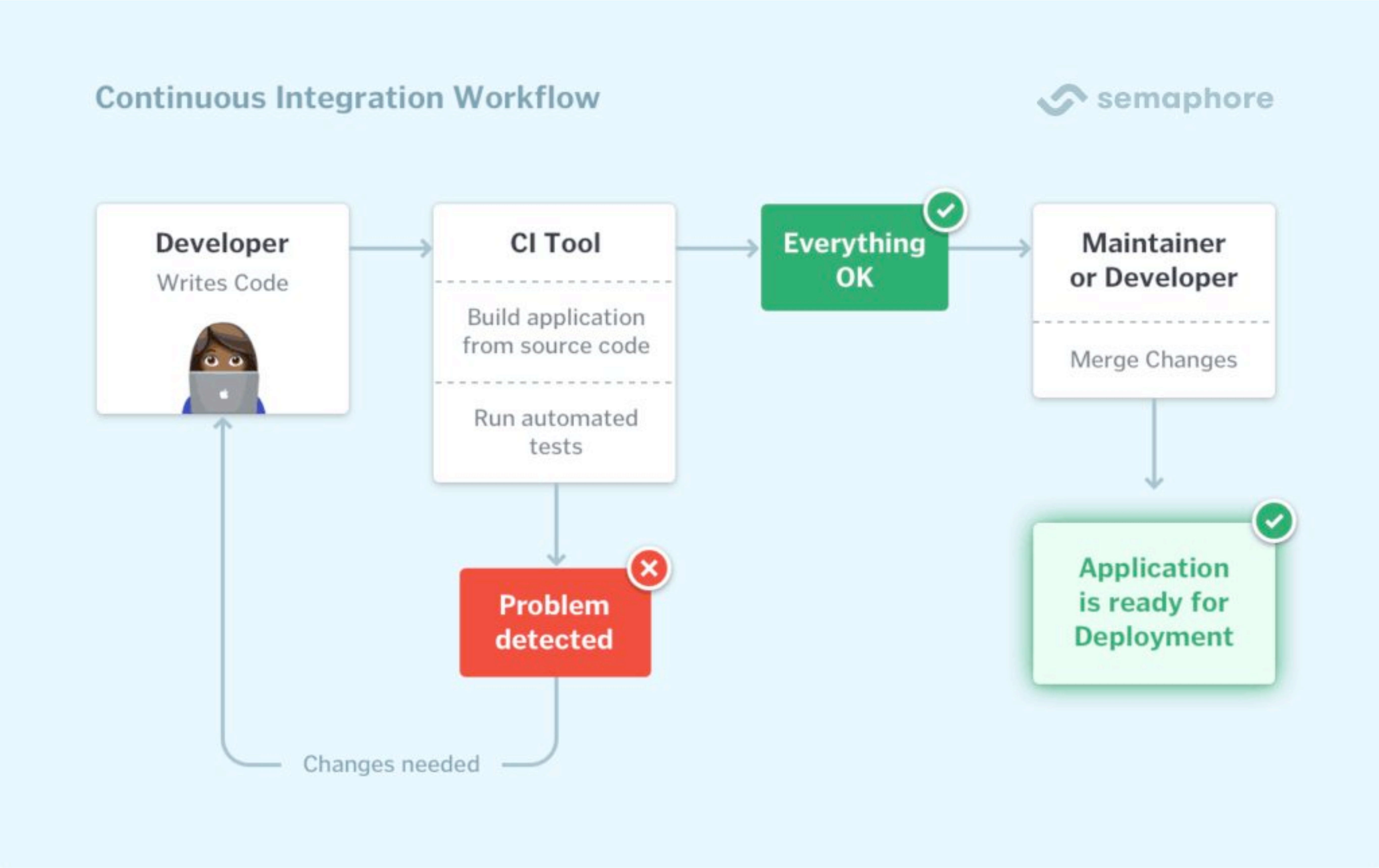
7 successful checks

✓	Ruby / build (push) Successful in 5m	Required	Details
✓	Ruby / Check Fixtures (push) Successful in 1m		Details
✓	Ruby / DB Setup (push) Successful in 1m		Details
✓	Ruby / ESLint (push) Successful in 4m		Details
✓	Ruby / Rubocop Linting (push) Successful in 4m		Details
✓	Ruby / Unit Tests (push) Successful in 4m		Details

This branch has no conflicts with the base branch

Merging can be performed automatically.

You can also open this in GitHub Desktop or view command line instructions.



https://semaphoreci.com/continuous-integration

SOUNDS EASY?

IT IS TO START, BUT NOT TO SCALE

SOME OF THE HARDER PROBLEMS

1. **Scheduling systems are advanced and complicated, handling FIFO queues with multiple criteria. It's easy to overwhelm the system and crash it**

2. **Worker pools are hard to scale. Cloud providers can't handle them all with ease. One company can use 75000+ CPUs for their CI system.**

3. **Workers need to be *isolated*. If they are not isolated, then they can impact one another. The worker needs to be reset to "factory conditions" <u>every single build.</u>**

4. **Scale is not consistent. Some times of the day are busier than others (e.g. right after lunch), so you need thousands more workers. However keeping those workers up all the time is *expensive* (millions of dollars a month).**

SOME OF THE HARDER PROBLEMS (CONT)

5. Tests can be non-deterministic (e.g. pass sometimes and fail sometimes) without warning. This means that test runs could fail when it’s not the developer’s fault. This is known as a Flaky Test. It is a big data problem (billions of data points a week for 1 test suite).

6. 5 to 10 minutes is the key time to meet based on Google research, however setting up the system can take *longer than that*. Which means you need to build aggressive caching mechanisms to keep feedback cycles short.

7. Expensive. Finance doesn’t like expensive.

8. Hard to scale because it increases scale rapidly (number of test cases run x number of commits per day, both of which increase as time goes on so it’s a hyperbolic increase)