Continuous Integration and Delivery

Karl Kosack

CEA Paris-Saclay Astrophysics Department

Université de Paris (-Saclay, -Cité)

CTAO

S3 School 2026, Annecy

S3 School

Sustainable Scientific Software School

Overview

Previously...

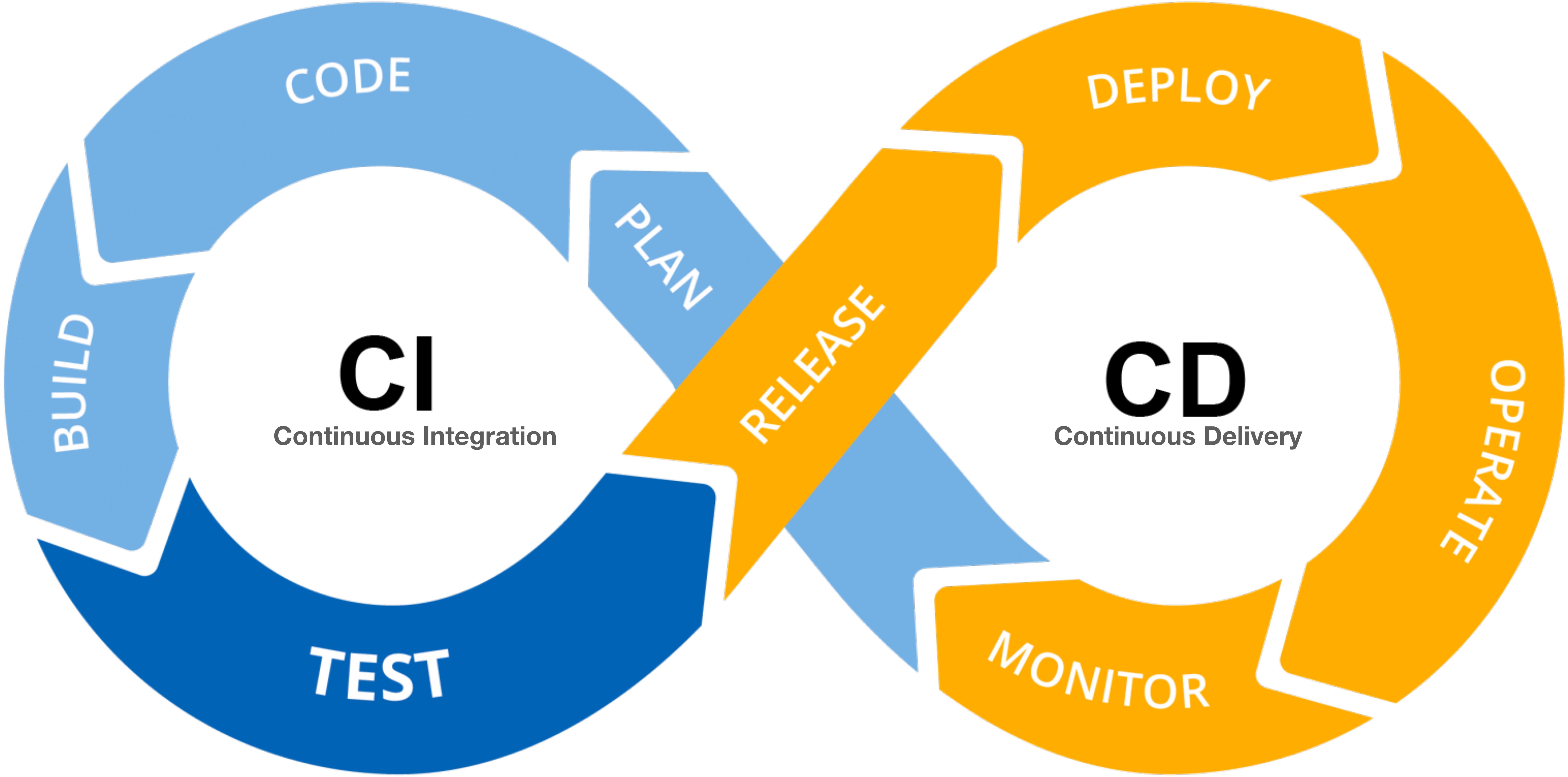
- You know how to **debug** code
- You know how to write **unit tests** and run them locally

In this lecture:

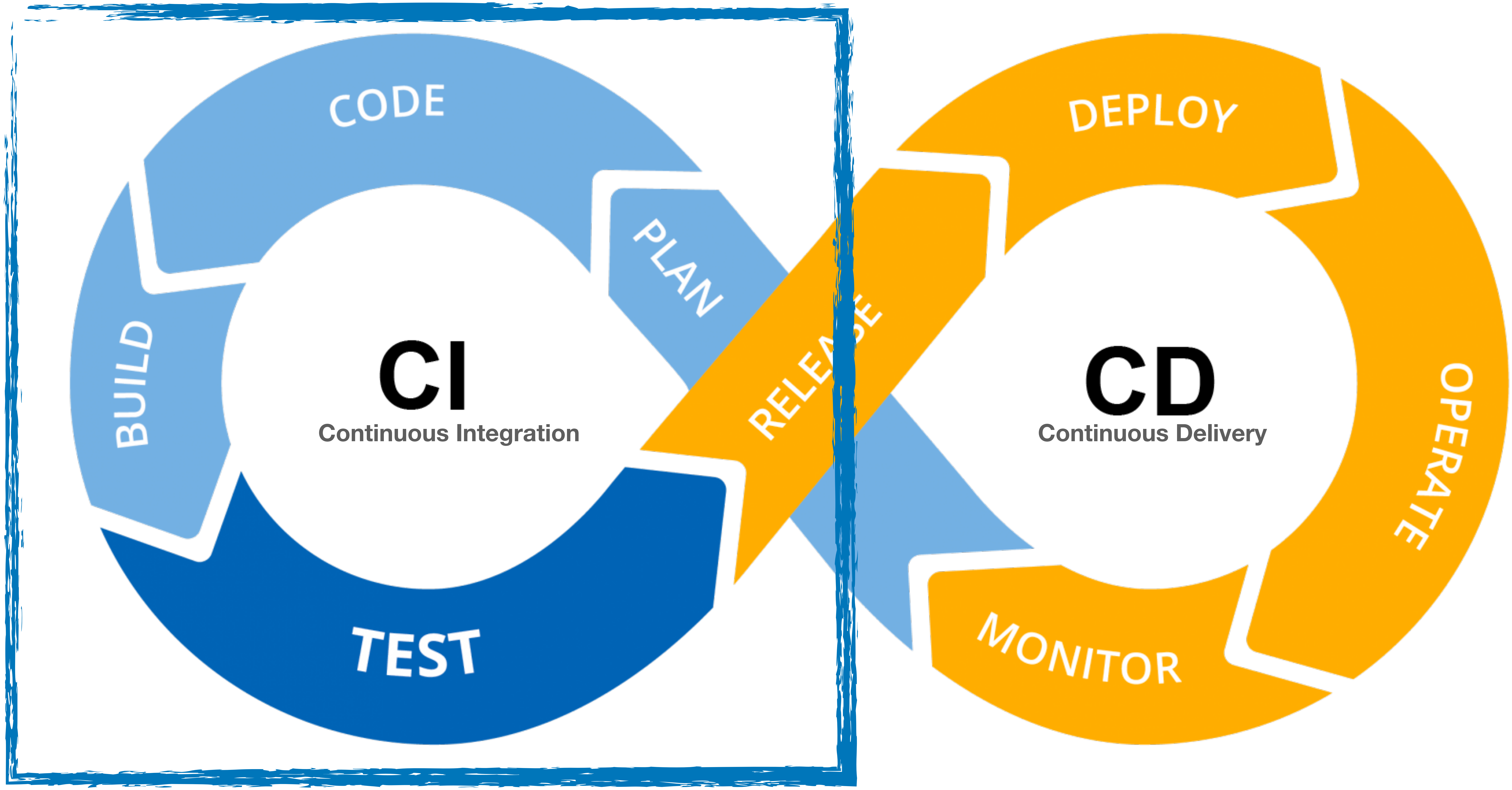
- How to *automate* linting* and unit testing on GitHub?
- How to ensure you code works in different environments or with newer dependencies?
- How to automatically package your code or results and share them?

* **linting** = **code quality checking**, literally finding and removing bits of lint from your clothes to make them more presentable

Technology + Development Philosophy...



Technology + Development Philosophy...



Continuous Integration:

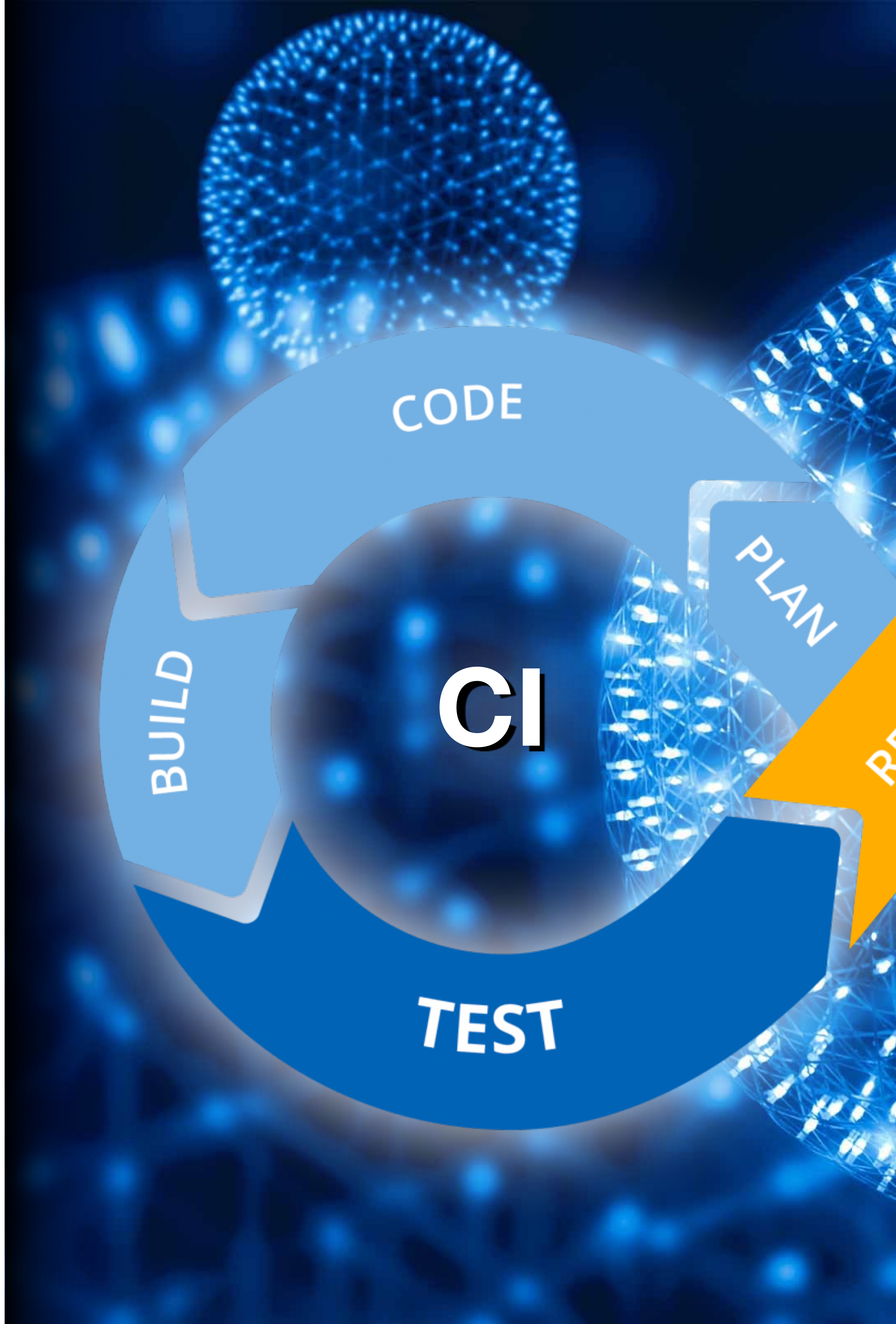
How to automate linting and unit testing on GitHub?

What is Continuous Integration (CI)?

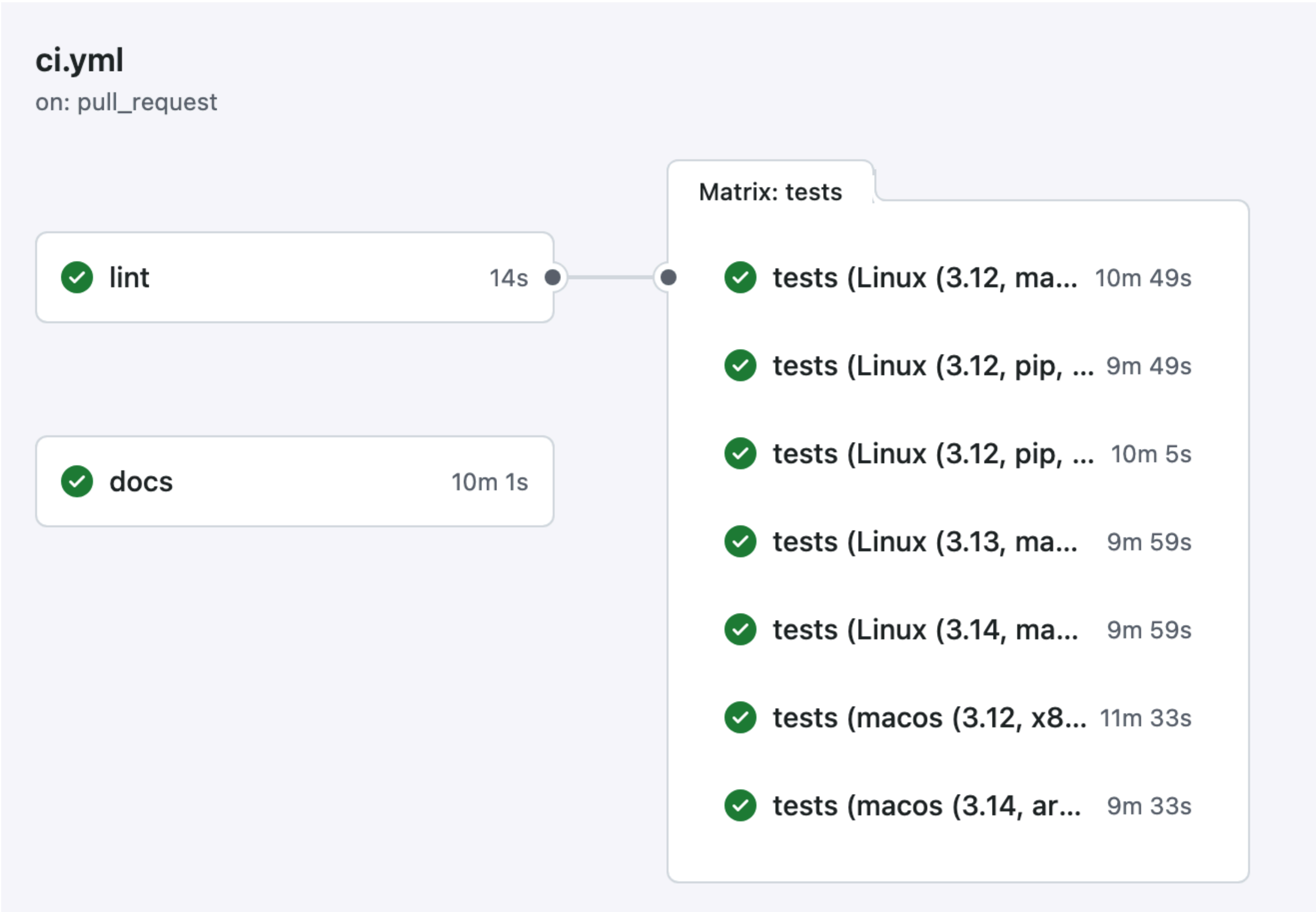
- the practice of frequently integrating code changes into a shared main branch.
- Each integration triggers an automated build and test process to verify that the code is in a healthy, working state

Why is it useful?

- Catch bugs early!
- Reduce human error and burden
- Makes working with multiple developers much easier



What does CI mean in practice?



ci.yml
on: pull_request

✓ lint 14s

✓ docs 10m 1s

Matrix: tests

✓ tests (Linux (3.12, ma... 10m 49s

✓ tests (Linux (3.12, pip, ... 9m 49s

✓ tests (Linux (3.12, pip, ... 10m 5s

✓ tests (Linux (3.13, ma... 9m 59s

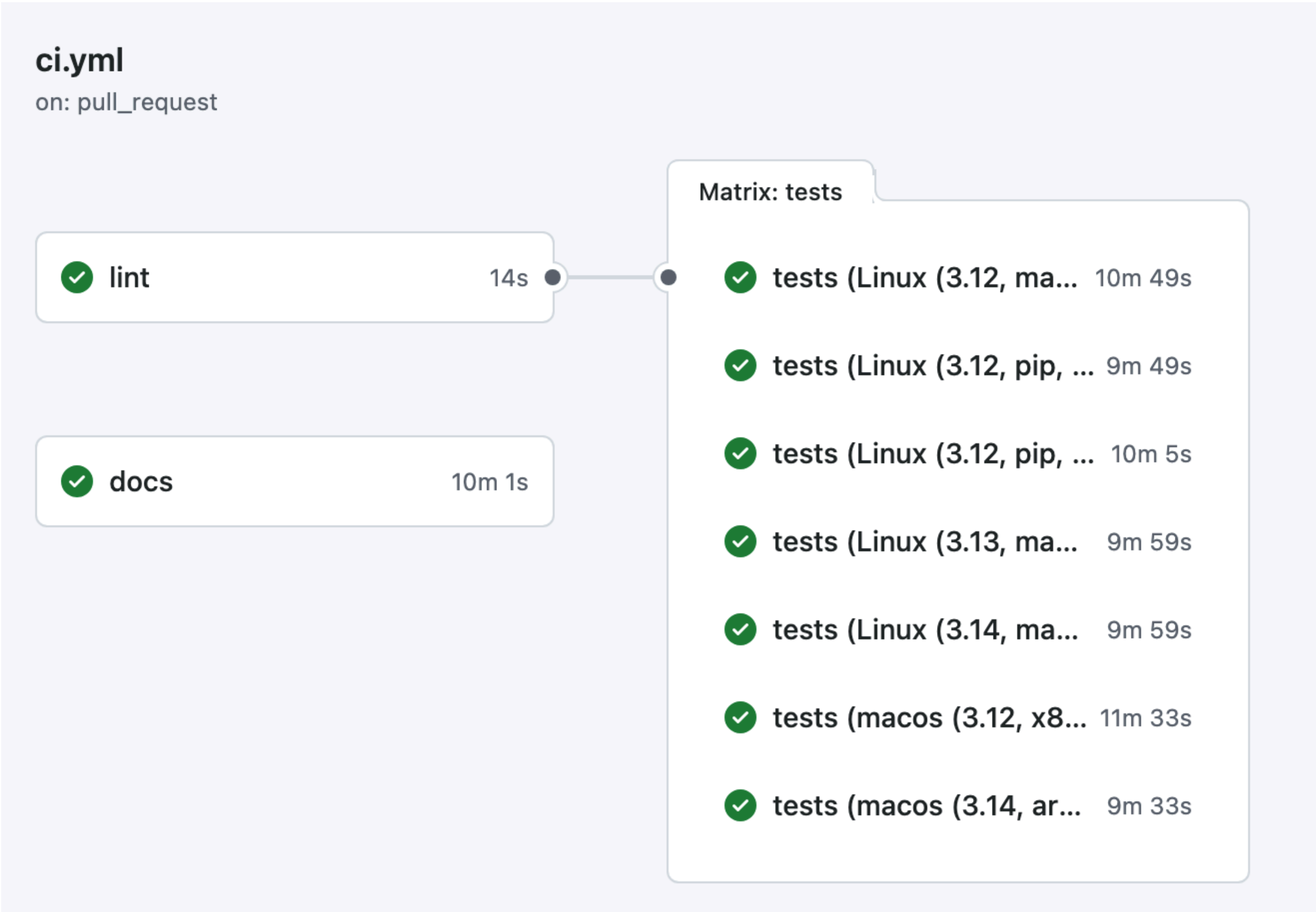
✓ tests (Linux (3.14, ma... 9m 59s

✓ tests (macos (3.12, x8... 11m 33s

✓ tests (macos (3.14, ar... 9m 33s

What does CI mean in practice?

An external service is used to:



ci.yml

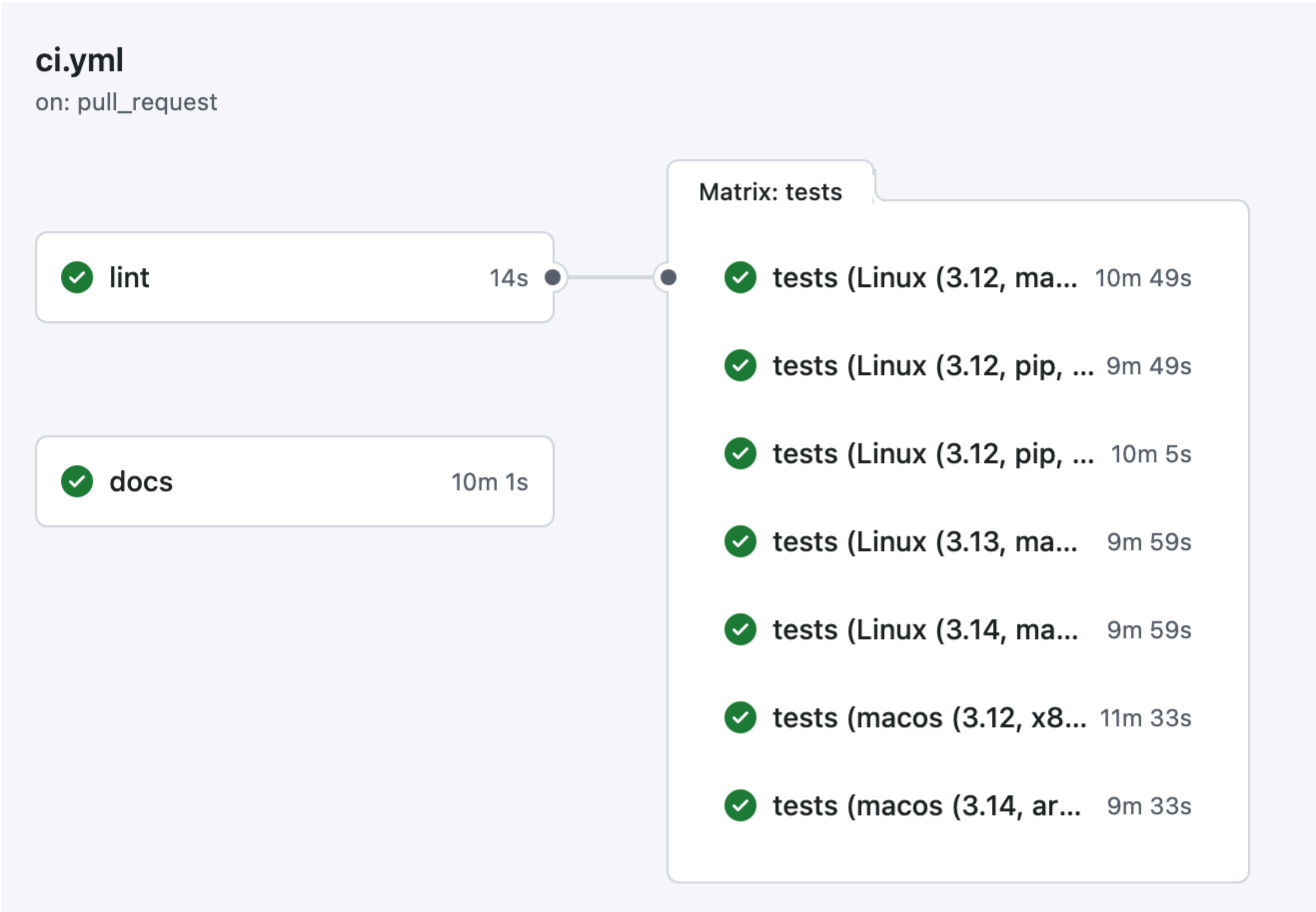
on: pull_request

- ✓ lint 14s
- ✓ docs 10m 1s

Matrix: tests

- ✓ tests (Linux (3.12, ma... 10m 49s
- ✓ tests (Linux (3.12, pip, ... 9m 49s
- ✓ tests (Linux (3.12, pip, ... 10m 5s
- ✓ tests (Linux (3.13, ma... 9m 59s
- ✓ tests (Linux (3.14, ma... 9m 59s
- ✓ tests (macos (3.12, x8... 11m 33s
- ✓ tests (macos (3.14, ar... 9m 33s

What does CI mean in practice?



ci.yml
on: pull_request

✓ lint 14s

✓ docs 10m 1s

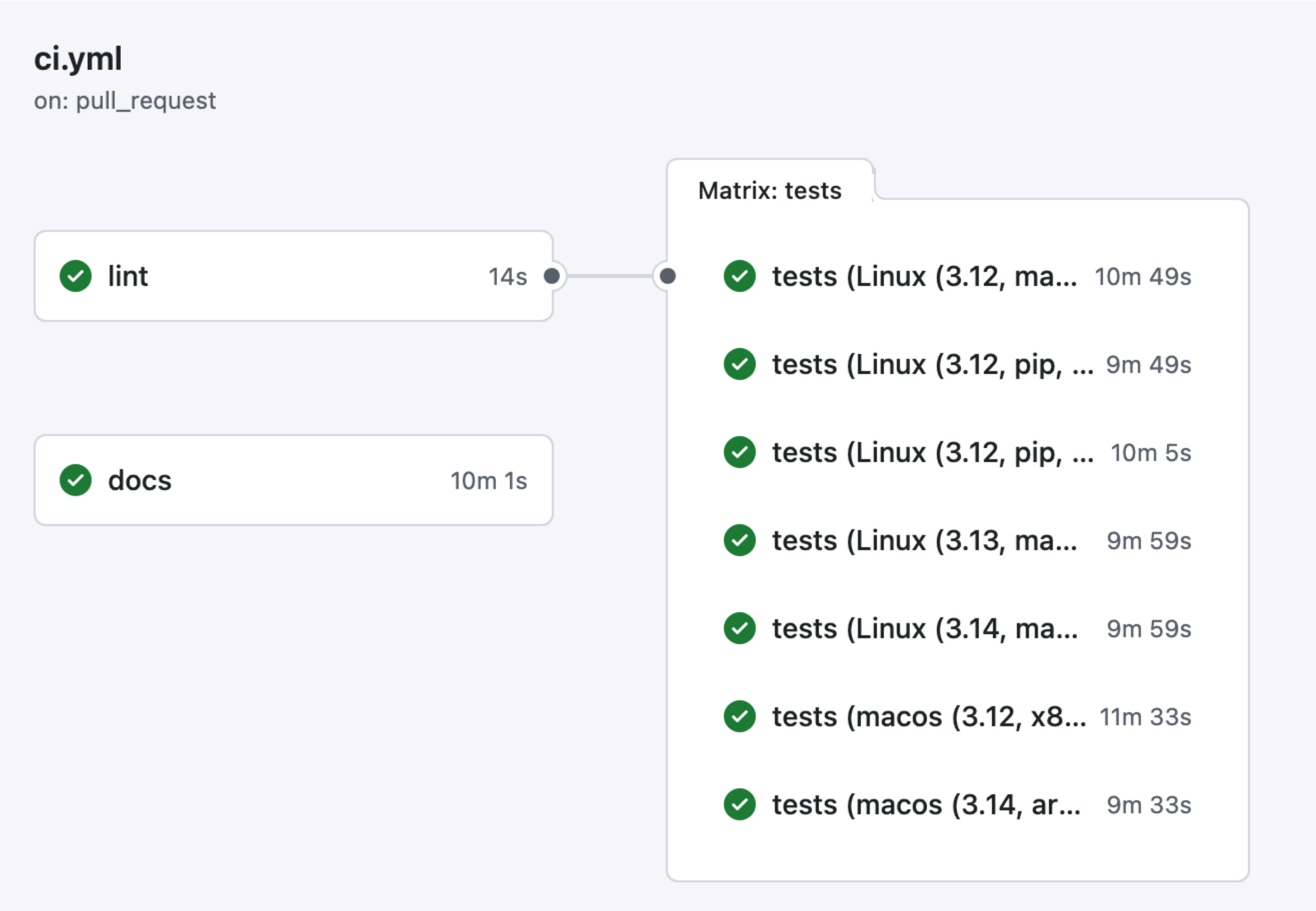
Matrix: tests

✓ tests (Linux (3.12, ma...	10m 49s
✓ tests (Linux (3.12, pip, ...	9m 49s
✓ tests (Linux (3.12, pip, ...	10m 5s
✓ tests (Linux (3.13, ma...	9m 59s
✓ tests (Linux (3.14, ma...	9m 59s
✓ tests (macos (3.12, x8...	11m 33s
✓ tests (macos (3.14, ar...	9m 33s

An external service is used to:

- Run code "**linting**" utilities to ensure good coding practices are followed

What does CI mean in practice?



ci.yml
on: pull_request

✓ lint 14s

✓ docs 10m 1s

Matrix: tests

✓ tests (Linux (3.12, ma... 10m 49s

✓ tests (Linux (3.12, pip, ... 9m 49s

✓ tests (Linux (3.12, pip, ... 10m 5s

✓ tests (Linux (3.13, ma... 9m 59s

✓ tests (Linux (3.14, ma... 9m 59s

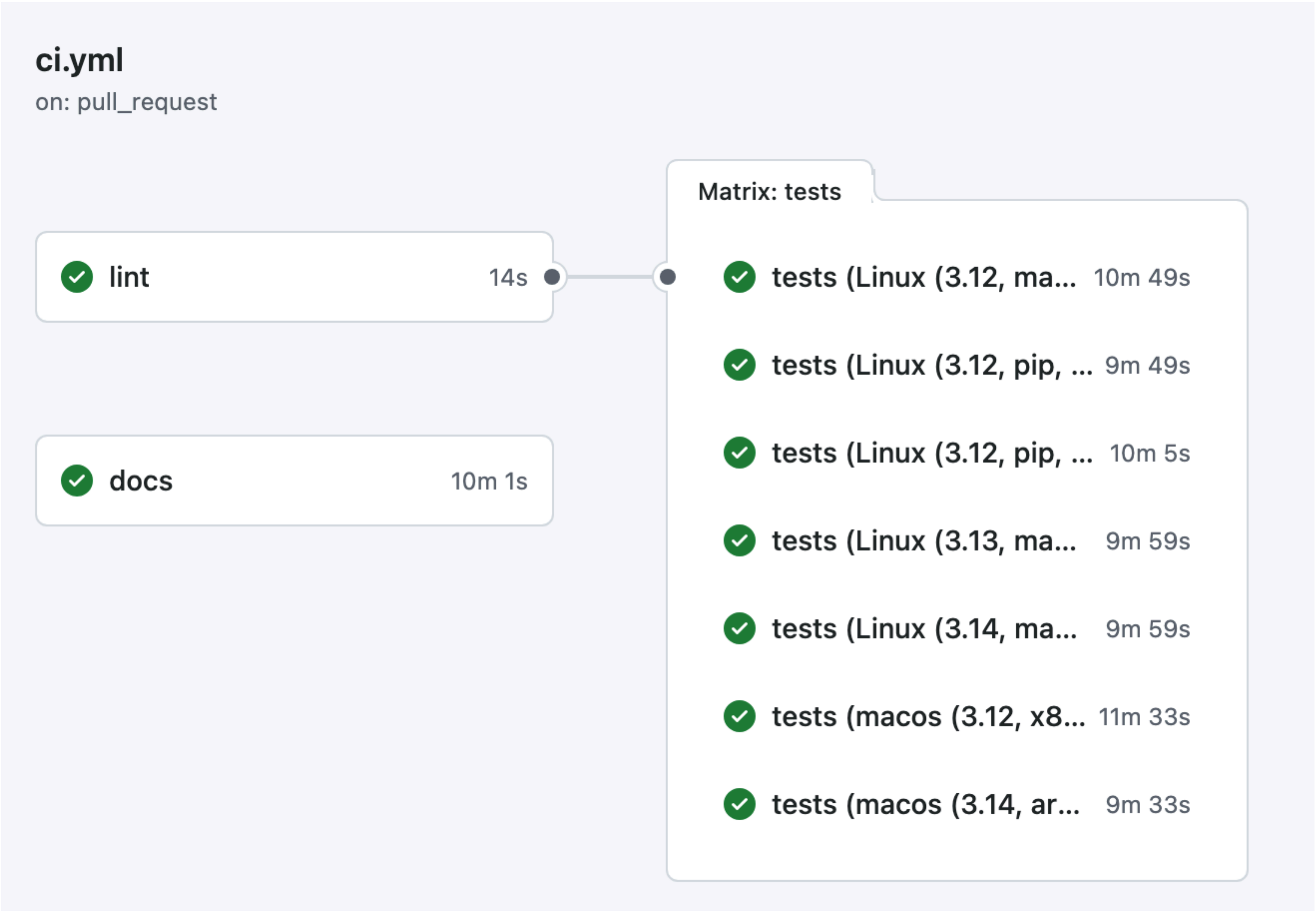
✓ tests (macos (3.12, x8... 11m 33s

✓ tests (macos (3.14, ar... 9m 33s

An external service is used to:

- Run code "**linting**" utilities to ensure good coding practices are followed
- Download, build, and install your code in a **clean, repeatable environment**

What does CI mean in practice?



ci.yml
on: pull_request

Job	Time
✓ lint	14s
✓ docs	10m 1s

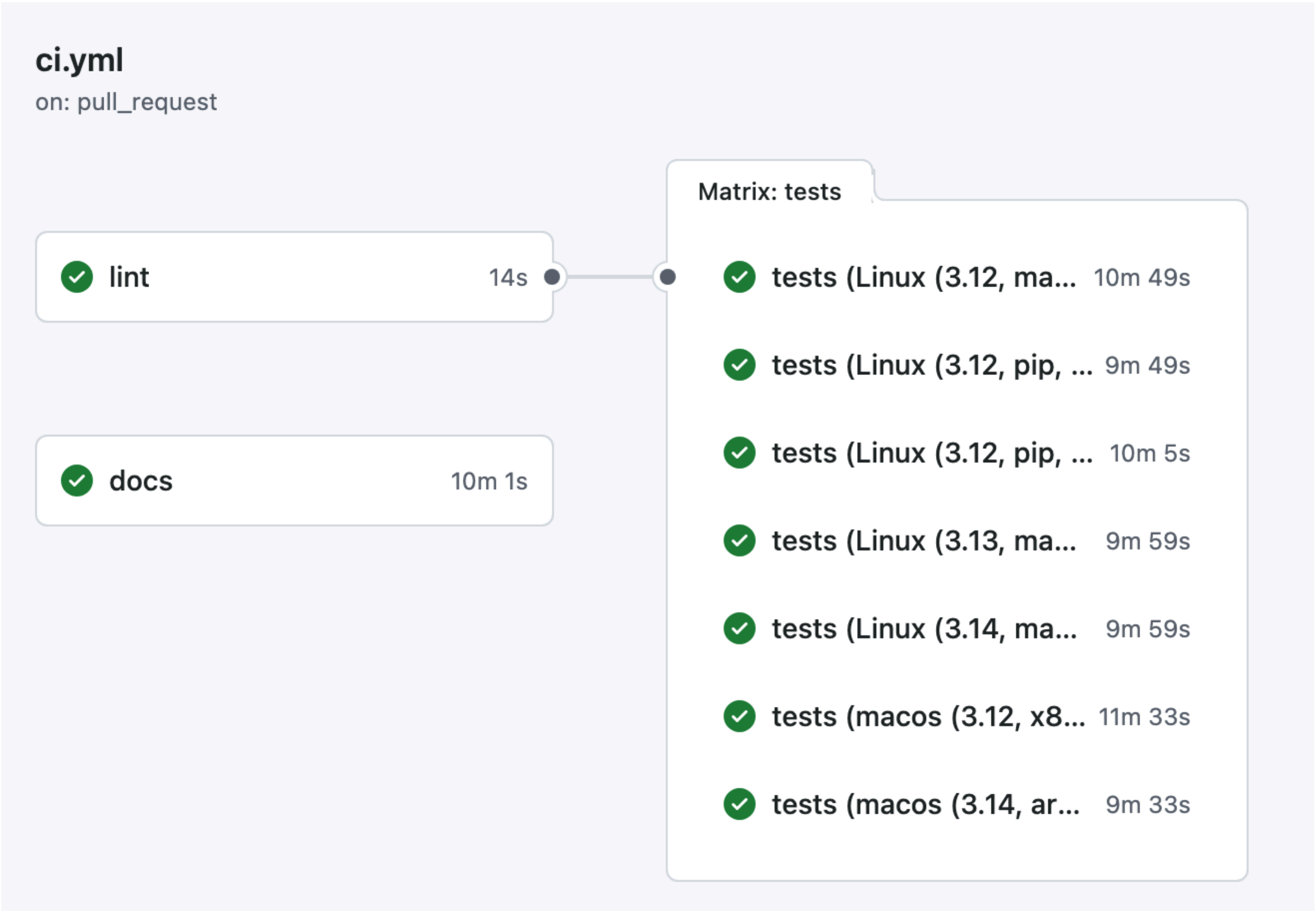
Matrix: tests

Job	Time
✓ tests (Linux (3.12, ma...	10m 49s
✓ tests (Linux (3.12, pip, ...	9m 49s
✓ tests (Linux (3.12, pip, ...	10m 5s
✓ tests (Linux (3.13, ma...	9m 59s
✓ tests (Linux (3.14, ma...	9m 59s
✓ tests (macos (3.12, x8...	11m 33s
✓ tests (macos (3.14, ar...	9m 33s

An external service is used to:

- Run code "**linting**" utilities to ensure good coding practices are followed
- Download, build, and install your code in a **clean, repeatable environment**
- Run all **unit tests** to ensure the code "works"

What does CI mean in practice?



ci.yml
on: pull_request

- ✓ lint 14s
- ✓ docs 10m 1s

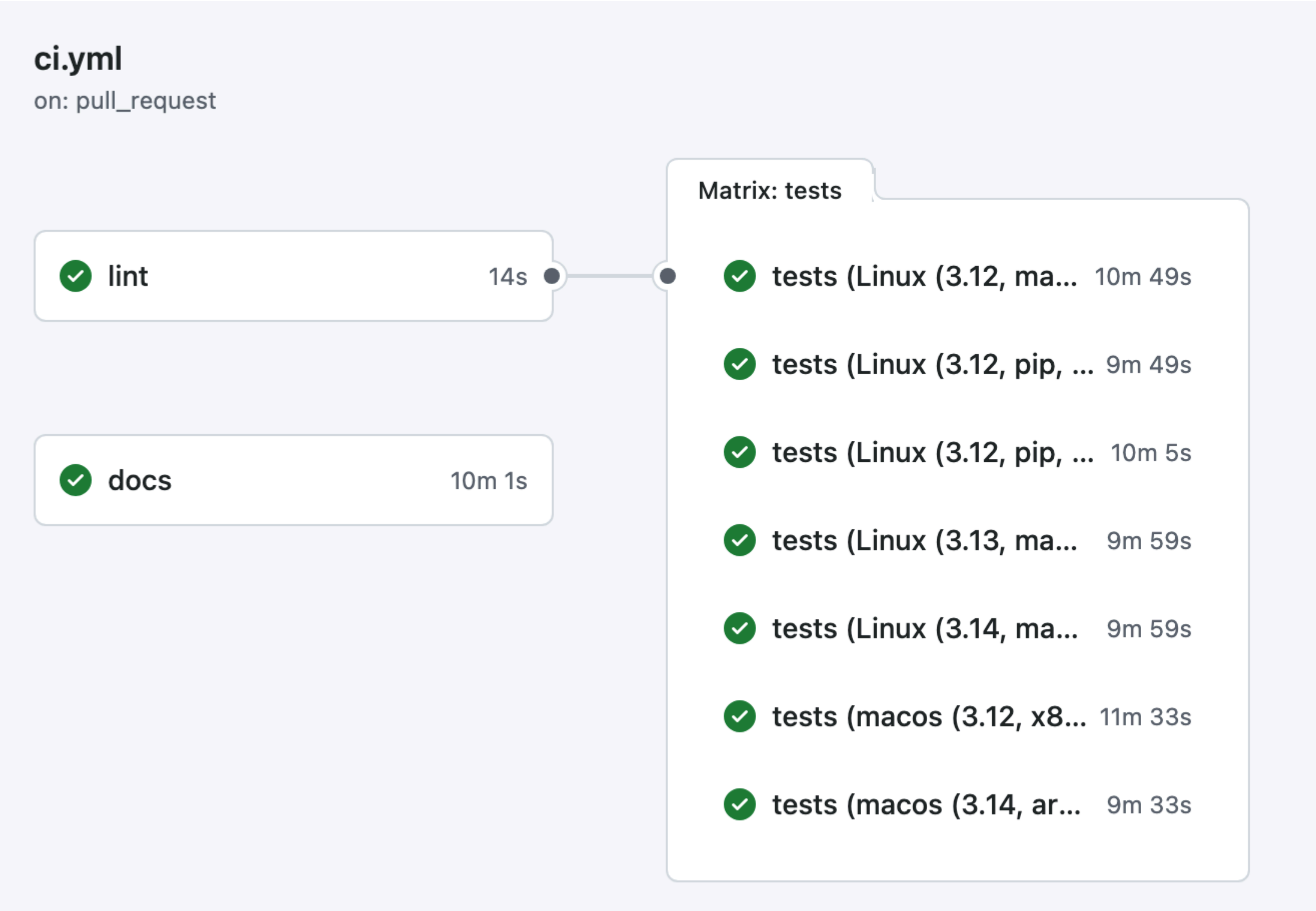
Matrix: tests

- ✓ tests (Linux (3.12, ma... 10m 49s
- ✓ tests (Linux (3.12, pip, ... 9m 49s
- ✓ tests (Linux (3.12, pip, ... 10m 5s
- ✓ tests (Linux (3.13, ma... 9m 59s
- ✓ tests (Linux (3.14, ma... 9m 59s
- ✓ tests (macos (3.12, x8... 11m 33s
- ✓ tests (macos (3.14, ar... 9m 33s

An external service is used to:

- Run code "**linting**" utilities to ensure good coding practices are followed
- Download, build, and install your code in a **clean, repeatable environment**
- Run all **unit tests** to ensure the code "works"
- Run larger ***integration tests*** that connect things together

What does CI mean in practice?



ci.yml
on: pull_request

✓ lint	14s
✓ docs	10m 1s

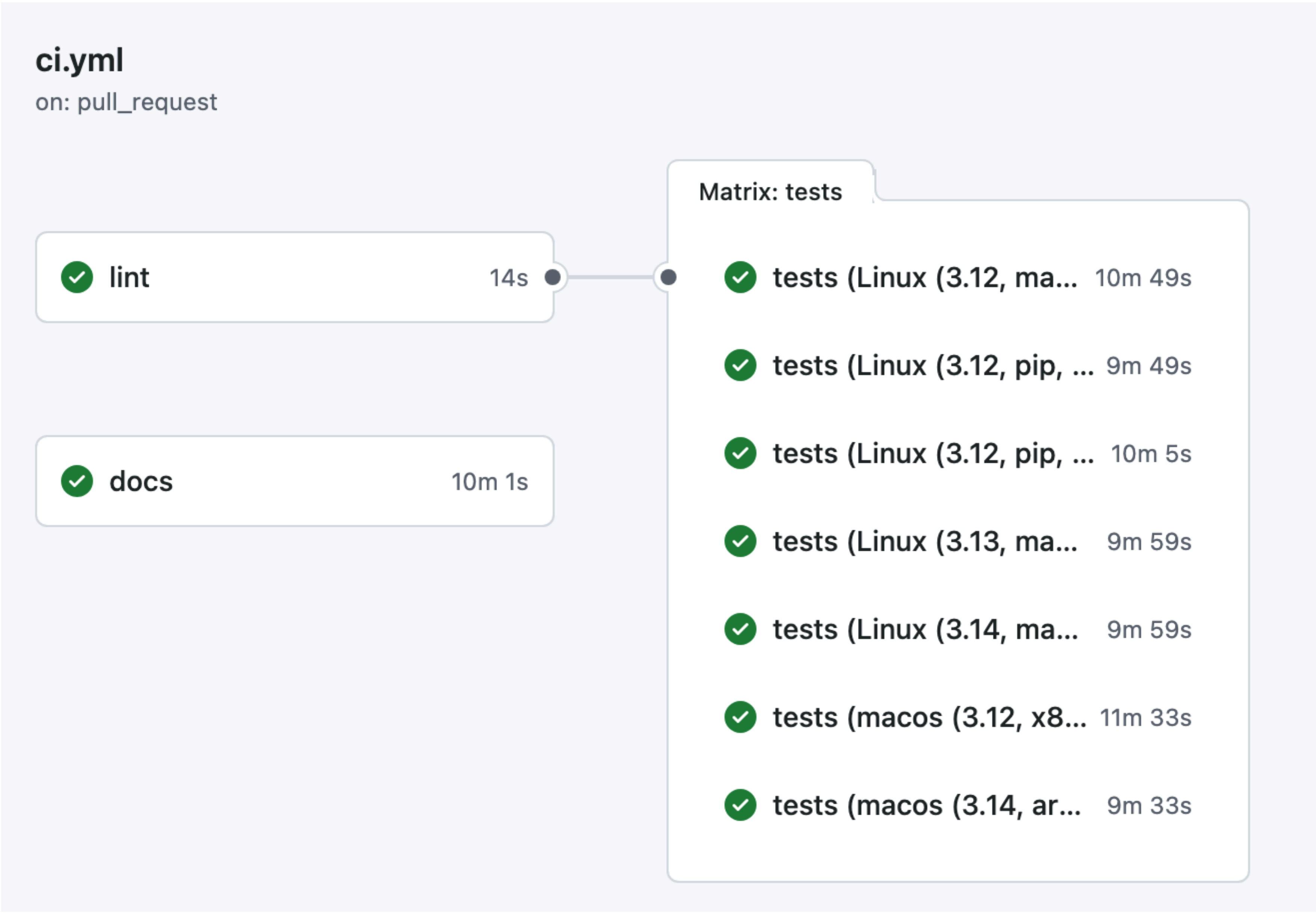
Matrix: tests

✓ tests (Linux (3.12, ma...	10m 49s
✓ tests (Linux (3.12, pip, ...	9m 49s
✓ tests (Linux (3.12, pip, ...	10m 5s
✓ tests (Linux (3.13, ma...	9m 59s
✓ tests (Linux (3.14, ma...	9m 59s
✓ tests (macos (3.12, x8...	11m 33s
✓ tests (macos (3.14, ar...	9m 33s

An external service is used to:

- Run code "**linting**" utilities to ensure good coding practices are followed
- Download, build, and install your code in a **clean, repeatable environment**
- Run all **unit tests** to ensure the code "works"
- Run larger ***integration tests*** that connect things together
- Optionally keep track of **changes to test coverage**, and warn if new code is missing tests

What does CI mean in practice?



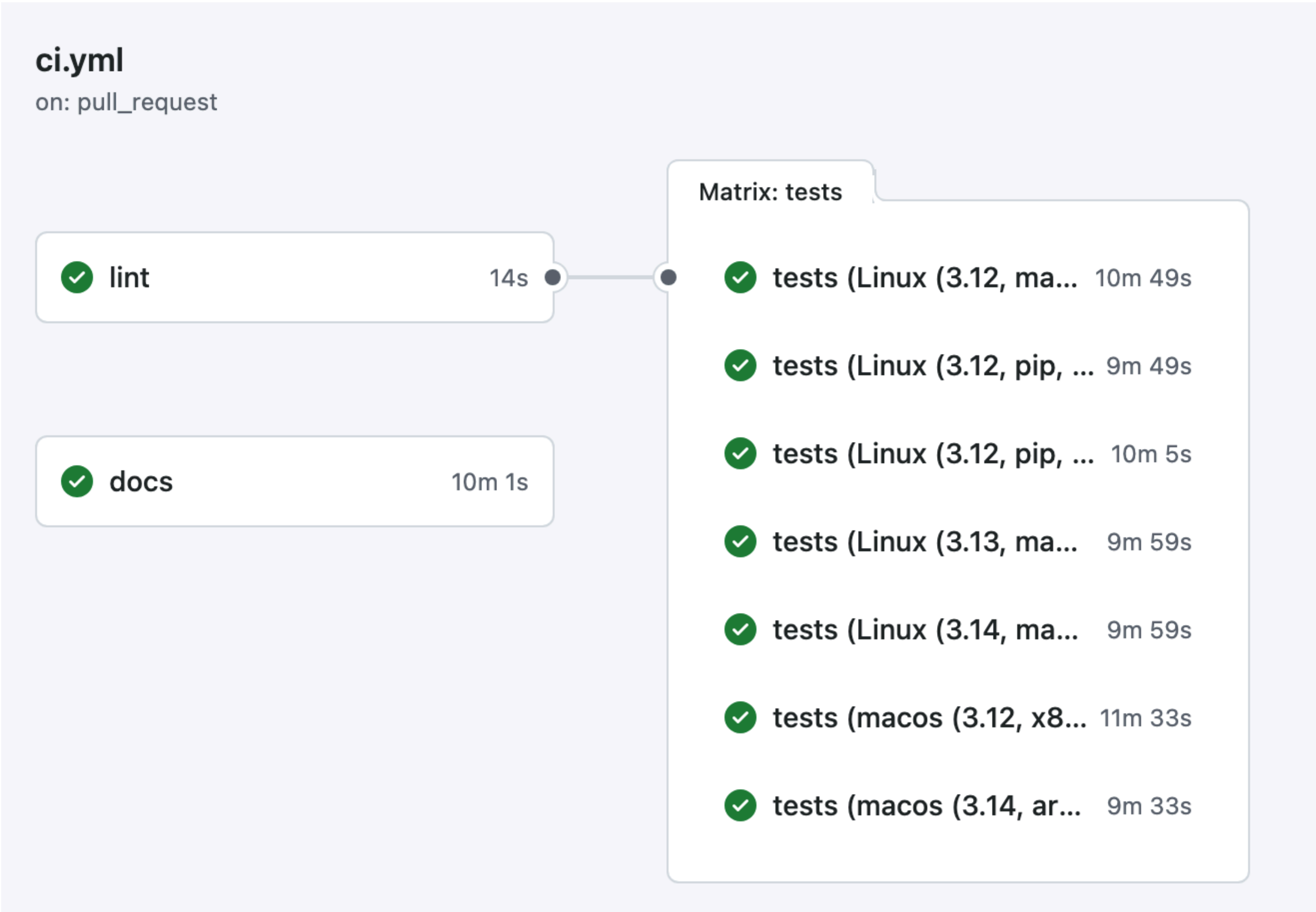
ci.yml
on: pull_request

✓ lint 14s
✓ docs 10m 1s
Matrix: tests
✓ tests (Linux (3.12, ma... 10m 49s
✓ tests (Linux (3.12, pip, ... 9m 49s
✓ tests (Linux (3.12, pip, ... 10m 5s
✓ tests (Linux (3.13, ma... 9m 59s
✓ tests (Linux (3.14, ma... 9m 59s
✓ tests (macos (3.12, x8... 11m 33s
✓ tests (macos (3.14, ar... 9m 33s

An external service is used to:

- Run code "**linting**" utilities to ensure good coding practices are followed
- Download, build, and install your code in a **clean, repeatable environment**
- Run all **unit tests** to ensure the code "works"
- Run larger ***integration tests*** that connect things together
- Optionally keep track of **changes to test coverage**, and warn if new code is missing tests
- Report on any **failures or problems**

What does CI mean in practice?



ci.yml
on: pull_request

✓ lint	14s
✓ docs	10m 1s

Matrix: tests

✓ tests (Linux (3.12, ma...	10m 49s
✓ tests (Linux (3.12, pip, ...	9m 49s
✓ tests (Linux (3.12, pip, ...	10m 5s
✓ tests (Linux (3.13, ma...	9m 59s
✓ tests (Linux (3.14, ma...	9m 59s
✓ tests (macos (3.12, x8...	11m 33s
✓ tests (macos (3.14, ar...	9m 33s

An external service is used to:

- Run code "**linting**" utilities to ensure good coding practices are followed
- Download, build, and install your code in a **clean, repeatable environment**
- Run all **unit tests** to ensure the code "works"
- Run larger ***integration tests*** that connect things together
- Optionally keep track of **changes to test coverage**, and warn if new code is missing tests
- Report on any **failures or problems**
- Generate **Artifacts** like packages and documentation (see later → "continuous deployment)

But can't I just do that myself?

But can't I just do that myself?

Yes, and you should... (pixi run test, pixi run check, ...)

But can't I just do that myself?

Yes, and you should... (pixi run test, pixi run check, ...)

But!

But can't I just do that myself?

Yes, and you should... (pixi run test, pixi run check, ...)

But!

- Do you *really* remember to run all tests, lints, checks every time you commit a change? When test time gets large (minutes+) do you still do it?

But can't I just do that myself?

Yes, and you should... (pixi run test, pixi run check, ...)

But!

- Do you *really* remember to run all tests, lints, checks every time you commit a change? When test time gets large (minutes+) do you still do it?
- Do you sometimes forget and merge something that has a bug or breaks the code?

But can't I just do that myself?

Yes, and you should... (pixi run test, pixi run check, ...)

But!

- Do you *really* remember to run all tests, lints, checks every time you commit a change? When test time gets large (minutes+) do you still do it?
- Do you sometimes forget and merge something that has a bug or breaks the code?
- Do you regularly check that new versions of dependencies don't break things?

But can't I just do that myself?

Yes, and you should... (pixi run test, pixi run check, ...)

But!

- Do you *really* remember to run all tests, lints, checks every time you commit a change? When test time gets large (minutes+) do you still do it?
- Do you sometimes forget and merge something that has a bug or breaks the code?
- Do you regularly check that new versions of dependencies don't break things?
- Do you check on different machines and OSes*?

***Differ on different machines:**
Filename conventions, temporary directory location, which packages are available, even CPU byte order!

But can't I just do that myself?

Yes, and you should... (pixi run test, pixi run check, ...)

But!

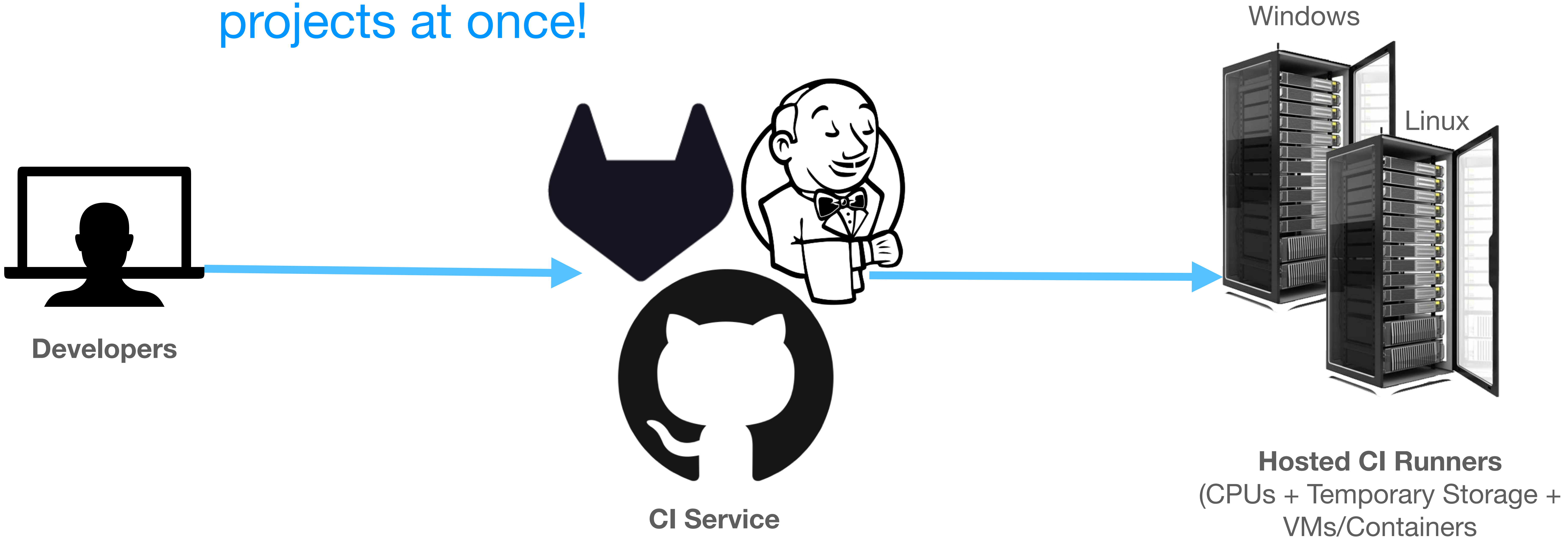
- Do you *really* remember to run all tests, lints, checks every time you commit a change? When test time gets large (minutes+) do you still do it?
- Do you sometimes forget and merge something that has a bug or breaks the code?
- Do you regularly check that new versions of dependencies don't break things?
- Do you check on different machines and OSes*?
- Do you work with other developers who might not be as strict as you?

***Differ on different machines:**
Filename conventions, temporary directory location, which packages are available, even CPU byte order!

Continuous Integration Services

What is required to do this?

- Usually a dedicated machine (or cluster)
- Large storage to cache code, dependencies, test data
- Sufficient CPUs to run unit test frequently, and often for many projects at once!



Some common options

	GitHub Actions	Gitlab CI	Jenkins
Integrates with	GitHub	GitLab or local GitLab	Any
Type of Host	Cloud	Cloud or Local	Local
Type of Runners?	Cloud	Cloud or Local	Local
OSes	Linux, Windows, macOS	Linux, Windows†, macOS† + user-hosted	Any provided
Architectures	x86, ARM, GPUs†	x86, ARM†, GPUs†	user-provided
Technology host/runner	closed/open	open/open	open/open
Who provides it?	Microsoft	GitLab	You, some companies provide paid hosting

† not for free, though!

Others: CircleCI, TravisCI, Azure Pipelines, Atlassian Bamboo, BitBucket pipelines...

Some common options

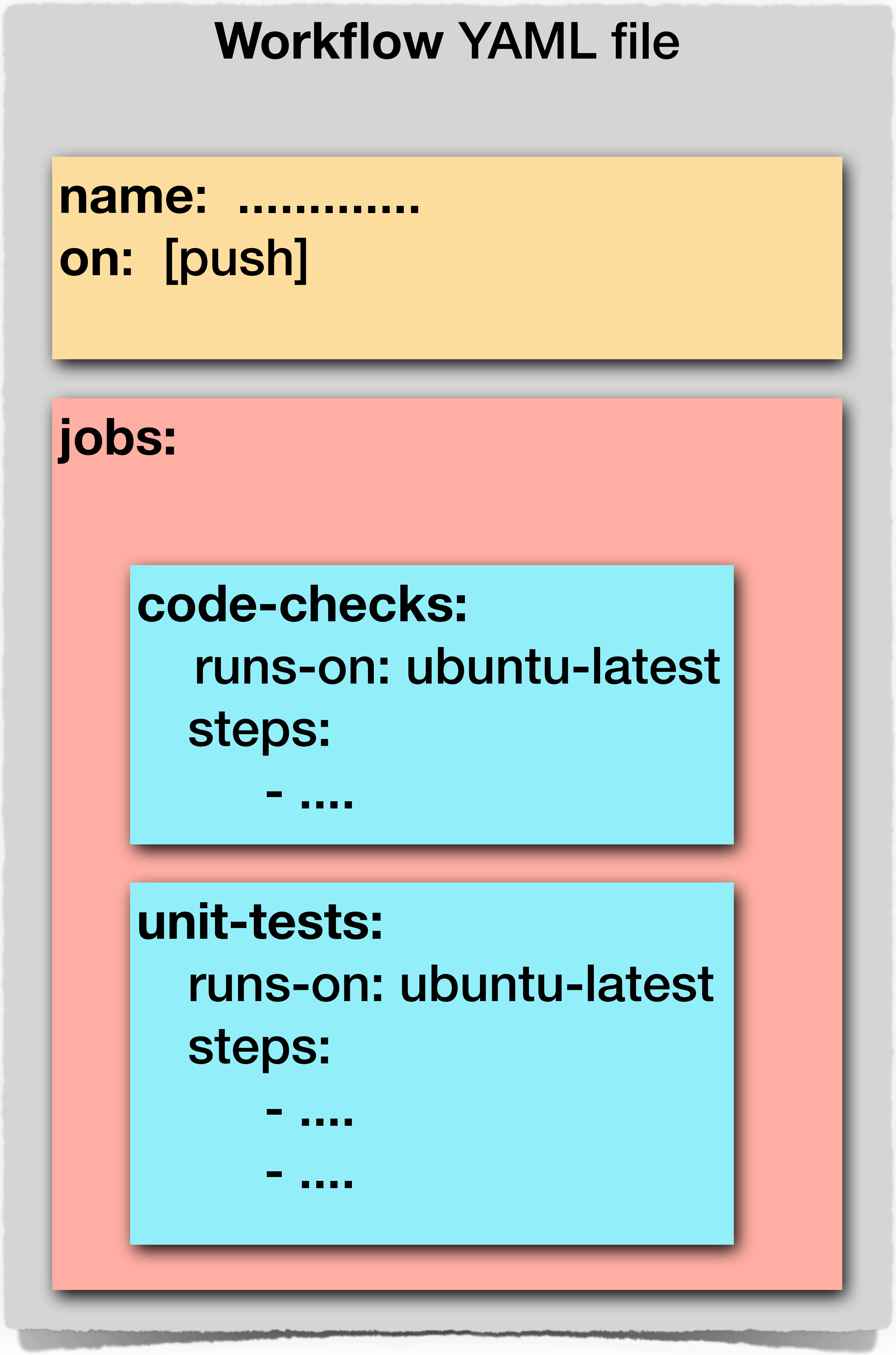
Will use in this lecture

	GitHub Actions	Gitlab CI	Jenkins
Integrates with	GitHub	GitLab or local GitLab	Any
Type of Host	Cloud	Cloud or Local	Local
Type of Runners?	Cloud	Cloud or Local	Local
OSes	Linux, Windows, macOS	Linux, Windows†, macOS† + user-hosted	Any provided
Architectures	x86, ARM, GPUs†	x86, ARM†, GPUs†	user-provided
Technology host/runner	closed/open	open/open	open/open
Who provides it?	Microsoft	GitLab	You, some companies provide paid hosting

† not for free, though!

Others: CircleCI, TravisCI, Azure Pipelines, Atlassian Bamboo, BitBucket pipelines...

Creating a CI workflow in your repository

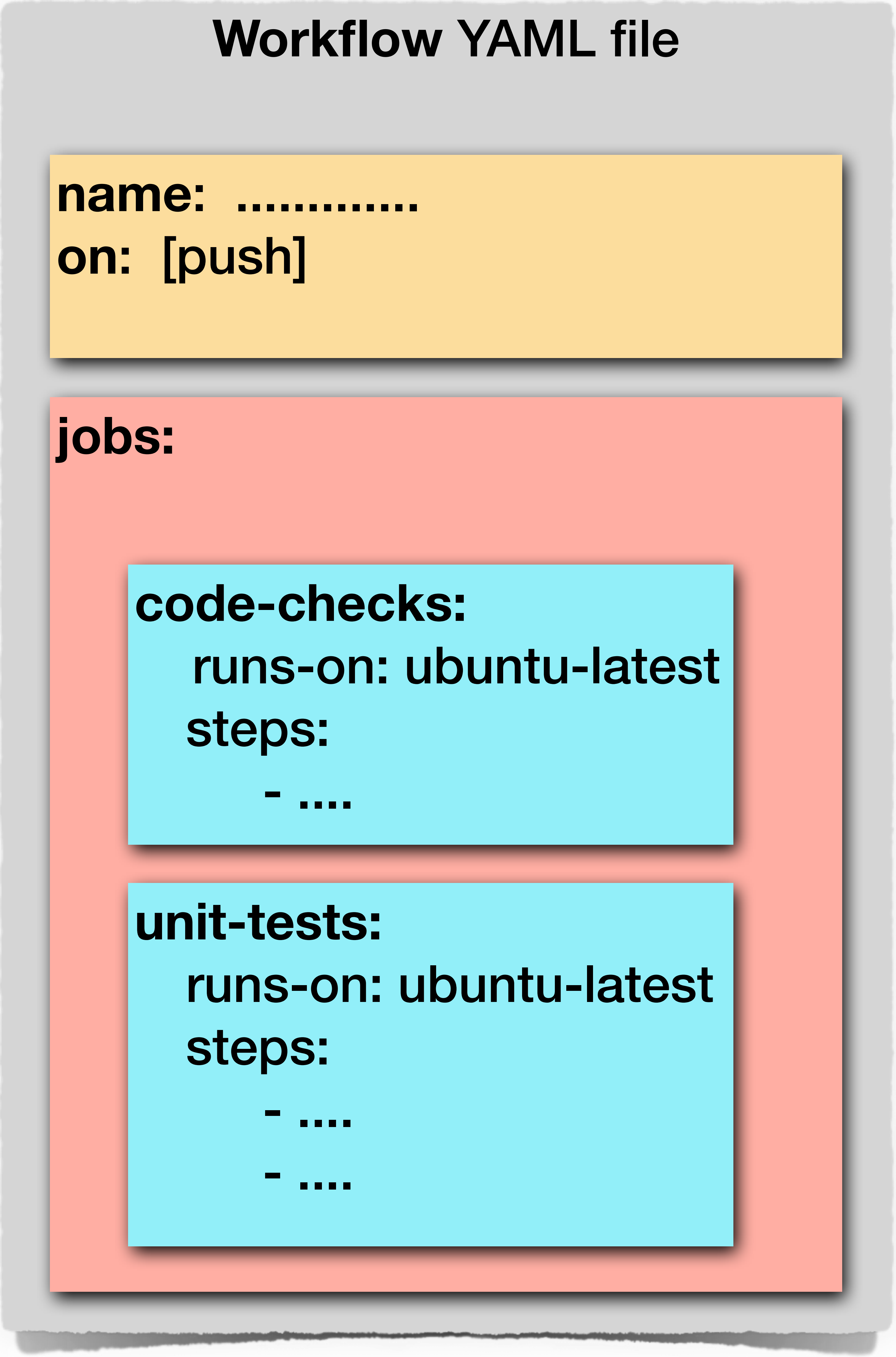


Workflow YAML file

```
name:  ............
on:  [push]

jobs:

    code-checks:
        runs-on: ubuntu-latest
        steps:
            - ....

    unit-tests:
        runs-on: ubuntu-latest
        steps:
            - ....
            - ....
```

Creating a CI workflow in your repository

In GitHub, this means adding a YAML file to
.github/workflows/ → it will be used automatically



Workflow YAML file

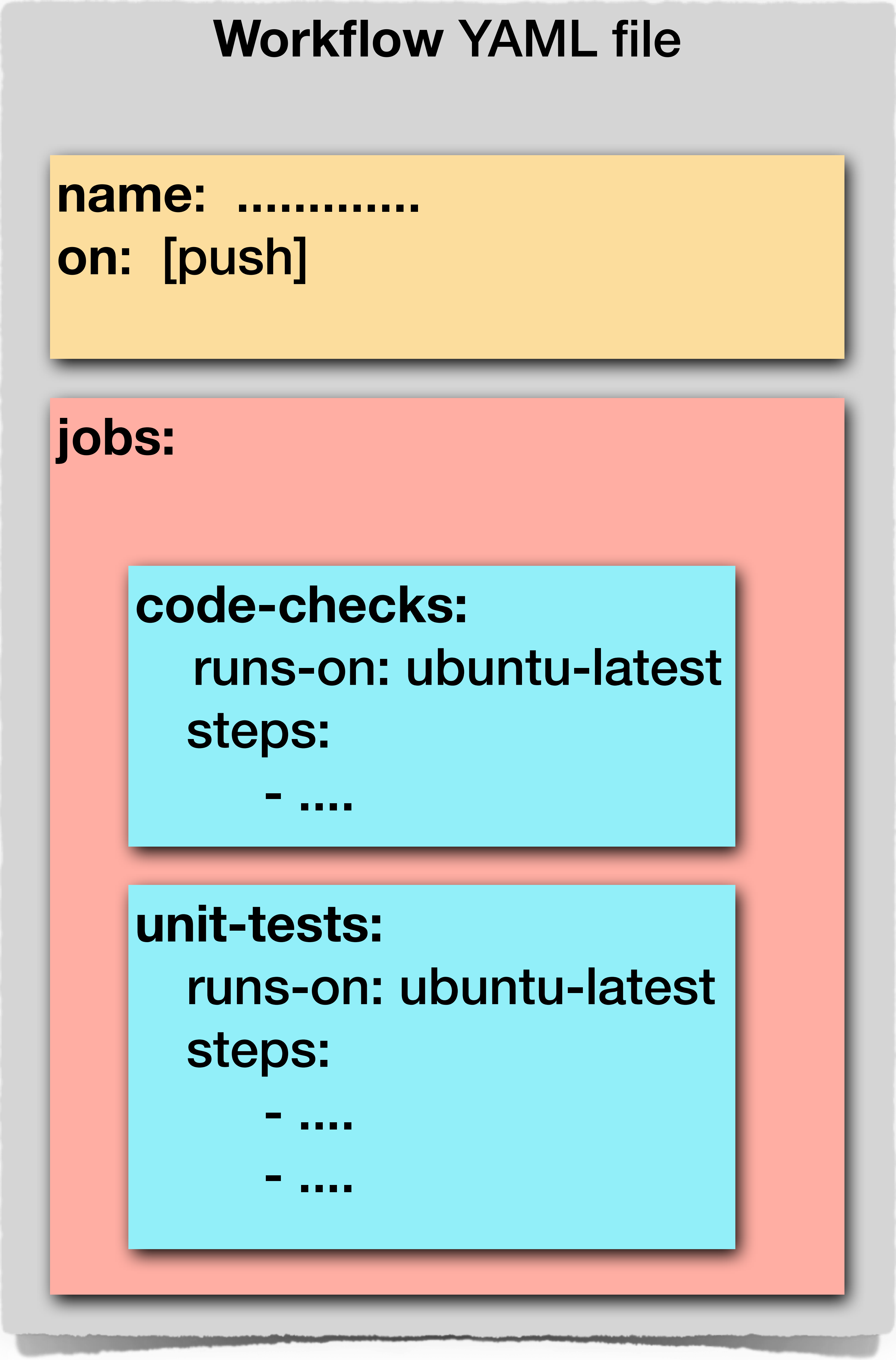
```
name:  .............
on:  [push]

jobs:

   code-checks:
      runs-on: ubuntu-latest
      steps:
         - ....

   unit-tests:
      runs-on: ubuntu-latest
      steps:
         - ....
         - ....
```

Creating a CI workflow in your repository

In GitHub, this means adding a YAML file to
.github/workflows/ → it will be used automatically

Workflows (yaml files)

- **on:** (when to run)
  - ➤ e.g on each push, pull-request, only specific branches, …
- **jobs:** (what to run)
  - ➤ run in parallel (unless you give dependencies)



Workflow YAML file

```
name:  ............
on:  [push]

jobs:

    code-checks:
        runs-on: ubuntu-latest
       steps:
           - ....

    unit-tests:
       runs-on: ubuntu-latest
       steps:
           - ....
           - ....
```

Creating a CI workflow in your repository

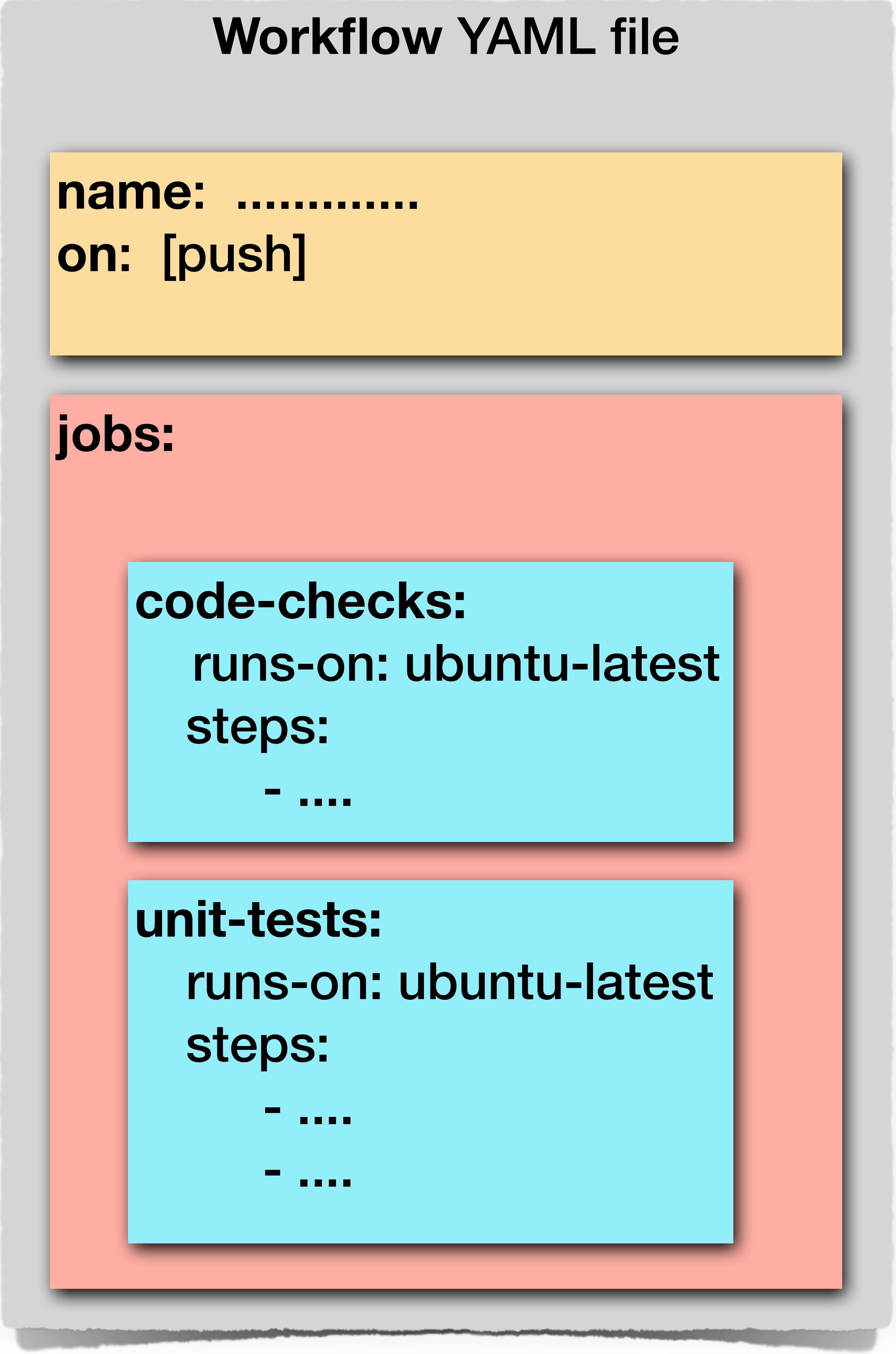
In GitHub, this means adding a YAML file to
.github/workflows/ → it will be used automatically

Workflows (yaml files)

- **on:** (when to run)
  - ➤ e.g on each push, pull-request, only specific branches, …
- **jobs:** (what to run)
  - ➤ run in parallel (unless you give dependencies)

Job:

- **runs-on**: what machine to use? (docker container name)
  - ➤ A new clean environment is started for each job!
- **steps**: list of what specifically to do



Workflow YAML file

```
name:  ............
on:  [push]

jobs:

    code-checks:
        runs-on: ubuntu-latest
        steps:
            - ....

    unit-tests:
        runs-on: ubuntu-latest
        steps:
            - ....
            - ....
```

Creating a CI workflow in your repository

In GitHub, this means adding a YAML file to
.github/workflows/ → it will be used automatically

Workflows (yaml files)

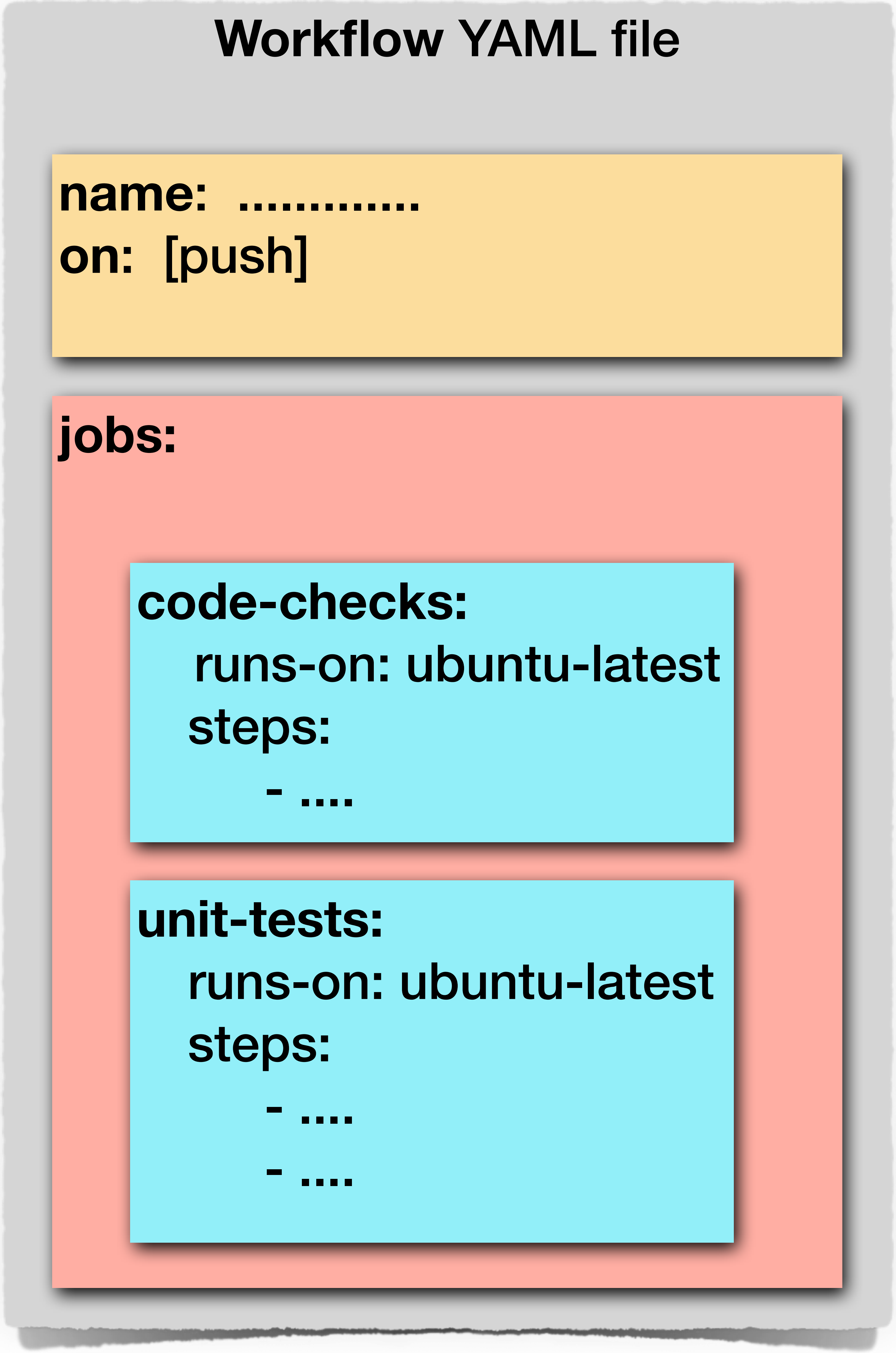
- **on:** (when to run)
  - ➤ e.g on each push, pull-request, only specific branches, ...
- **jobs:** (what to run)
  - ➤ run in parallel (unless you give dependencies)

Job:

- **runs-on**: what machine to use? (docker container name)
  - ➤ A new clean environment is started for each job!
- **steps**: list of what specifically to do

Step: can be one of

- **run:** a command to run in the given OS's shell
- **uses**: a pre-written block ← Prefer these, others have done the work!



Workflow YAML file

```
name:  ............
on:  [push]

jobs:

    code-checks:
        runs-on: ubuntu-latest
        steps:
            - ....

    unit-tests:
        runs-on: ubuntu-latest
        steps:
            - ....
            - ....
```

A very simple starting point

A very simple starting point

pkoffee/.github/workflows/test.yml

```
name: A simple test workflow
on: [push]   # when to run this workflow, on every push
jobs:
    unit-tests:  # this can be any name: you choose it
        runs-on: ubuntu-latest   # which OS to start up
        steps:   # what steps to run in this job
            - run: "echo 'Running tests on: ${{ runner.os }}'"
            - run: "echo 'Repo: ${{ github.repository }}'"
            - run: "echo 'Branch: ${{ github.ref }}'"
            - run: apt-get update && apt-get install -y plantuml  # add packages to machine (not needed for pixi!)
```

A very simple starting point

pkoffee/.github/workflows/test.yml

```yaml
name: A simple test workflow
on: [push]   # when to run this workflow, on every push
jobs:
    unit-tests:  # this can be any name: you choose it
        runs-on: ubuntu-latest   # which OS to start up
        steps:   # what steps to run in this job
            - run: "echo 'Running tests on: ${{ runner.os }}'"
            - run: "echo 'Repo: ${{ github.repository }}'"
            - run: "echo 'Branch: ${{ github.ref }}'"
            - run: apt-get update && apt-get install -y plantuml  # add packages to machine (not needed for pixi!)
```

Where to find some standard pre-defined blocks:

- https://github.com/actions
- https://github.com/marketplace
    - **setup-pixi**: https://github.com/marketplace/actions/setup-pixi
    - **deploy-pages**: https://github.com/actions/deploy-pages

A very simple starting point

pkoffee/.github/workflows/test.yml

```
name: A simple test workflow
on: [push]   # when to run this workflow, on every push
jobs:
    unit-tests:  # this can be any name: you choose it
        runs-on: ubuntu-latest   # which OS to start up
        steps:   # what steps to run in this job
            - run: "echo 'Running tests on: ${{ runner.os }}'"
            - run: "echo 'Repo: ${{ github.repository }}'"
            - run: "echo 'Branch: ${{ github.ref }}'"
            - run: apt-get update && apt-get install -y plantuml  # add packages to machine (not needed for pixi!)
```

Where to find some standard pre-defined blocks:

- https://github.com/actions
- https://github.com/marketplace
  - ➤ **setup-pixi**: https://github.com/marketplace/actions/setup-pixi
  - ➤ **deploy-pages**: https://github.com/actions/deploy-pages

Now let's add a pre-defined actions: checkout our code and install pixi

```
            - uses: actions/checkout@v5
            - uses: prefix-dev/setup-pixi@v0.9.3
              with:
                  pixi-version: v0.63.0
```

Demo Time!

SETUP

- You will work in your forked version of pkoffee, so that you have full access to the repository settings
- You should create a branch to work in (not required, but best practice!)
  - ➤ from your working or main branch if you have implemented tests

```
$ git switch -c add_ci
```

  - ➤ If you don't yet have tests, you can use the *upstream day_1_solution* branch, but you need to add some basic testing...

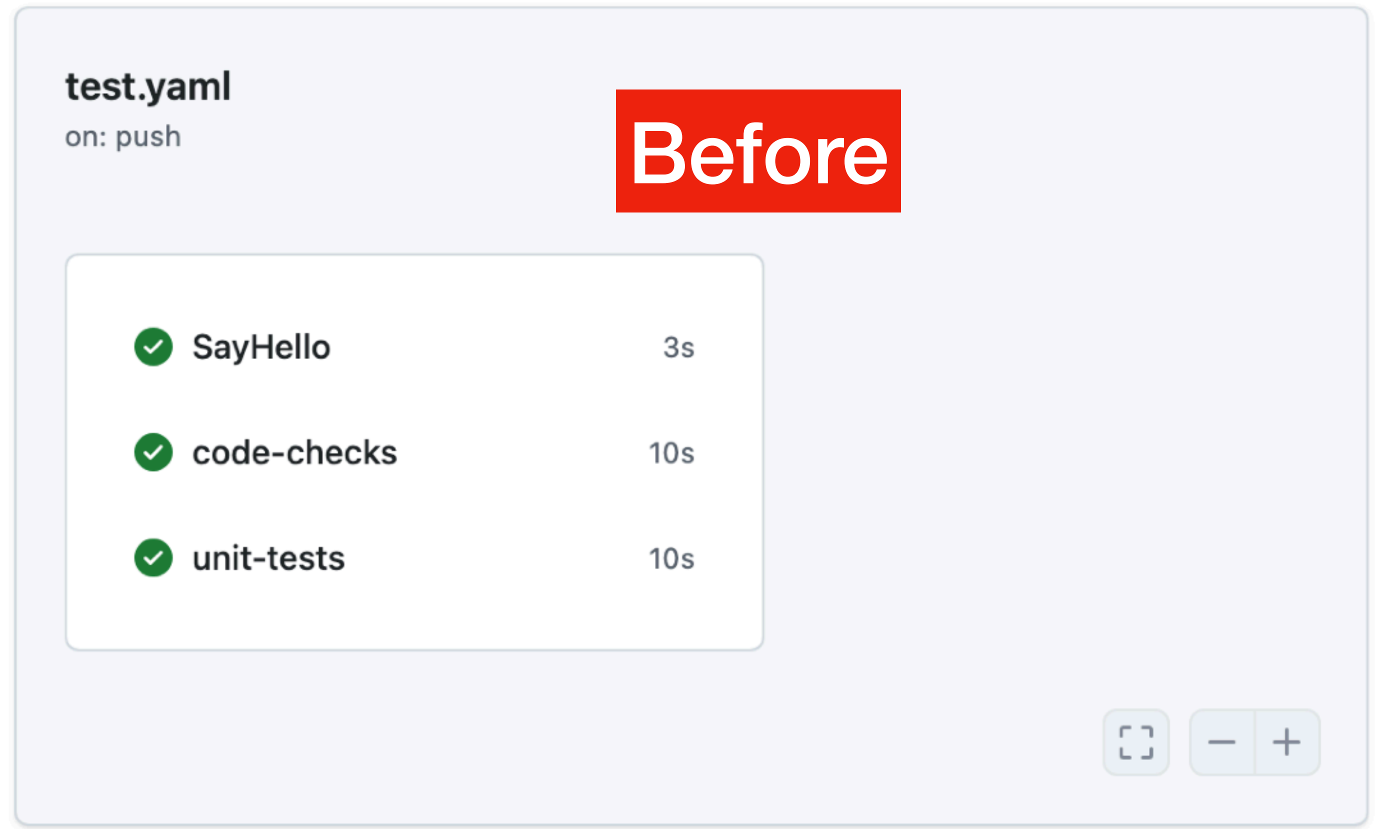
```
$ git switch -c add_ci upstream/day_1_solution
```

```
$ pixi add pytest
```

and add at least one test file to **tests/**

Job dependencies

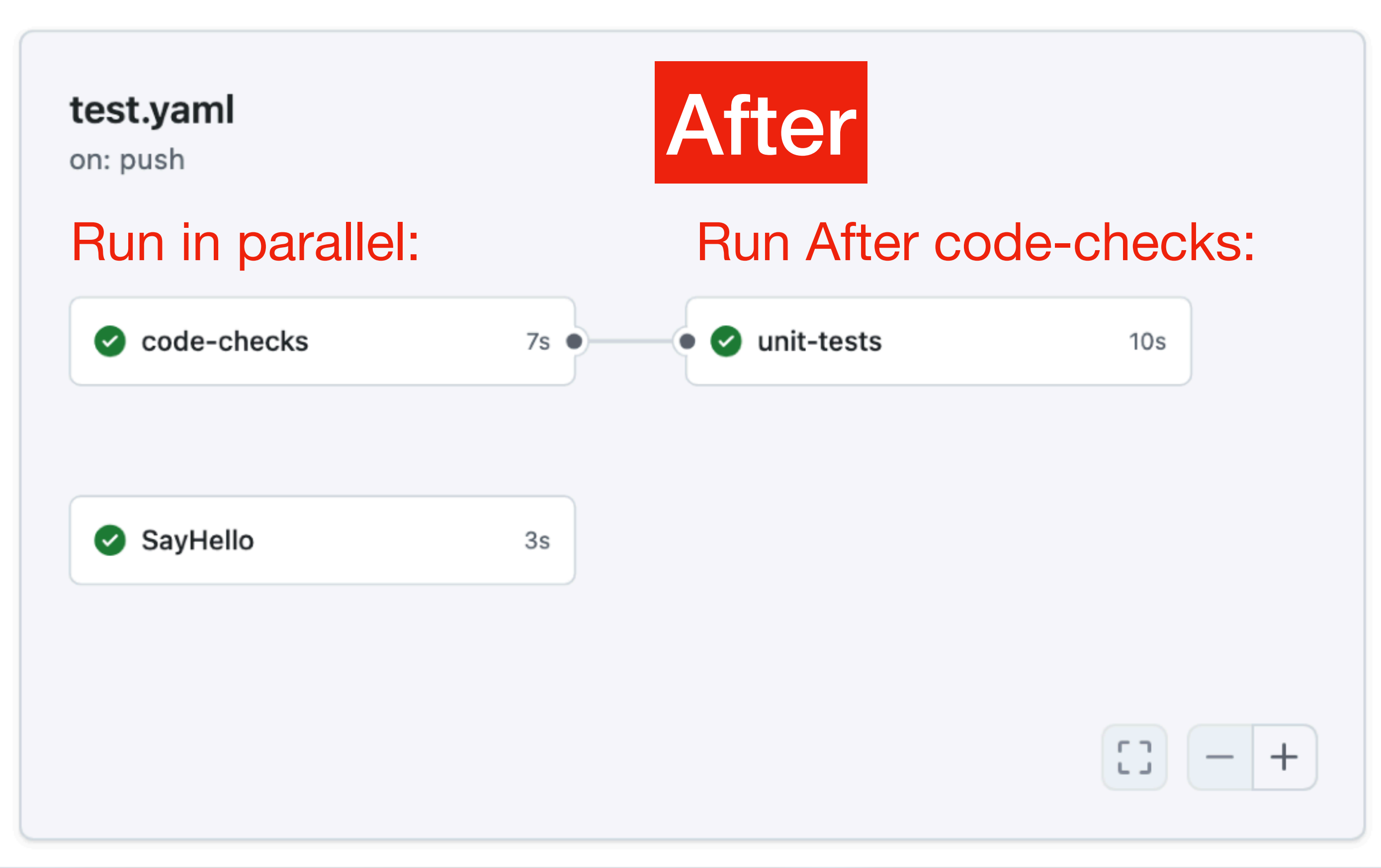
```yaml
name: Test
on: [push]

jobs:
    SayHello:
        runs-on: ubuntu-latest

        steps:
        - run: "echo 'Running tests for ${{ github.repository }}'"
        - run: "echo 'Repo: ${{ github.repository }}'"
        - run: "echo 'Branch: ${{ github.ref }}'"

    code-checks:
        runs-on: ubuntu-latest

        steps:
        - name: Checkout the code
          uses: actions/checkout@v6

        - name: Install pixi
          uses: prefix-dev/setup-pixi@v0.9.3
          with:
              pixi-version: v0.62.2  # good to pin this since pixi is not 1.0 yet!

        - name: look for syntax errors
          run: pixi run ruff check

    unit-tests:
        runs-on: ubuntu-latest
        needs: code-checks  # don't bother running tests if code-checks fails.

        steps:
        - name: Checkout the code
          uses: actions/checkout@v6

        - name: Install pixi
          uses: prefix-dev/setup-pixi@v0.9.3
          with:
              pixi-version: v0.62.2  # good to pin this since pixi is not 1.0 yet!

        - run: pixi run pytest
```



Before

test.yaml
on: push

✓ SayHello 3s
✓ code-checks 10s
✓ unit-tests 10s



After

test.yaml
on: push

Run in parallel:

Run After code-checks:

✓ code-checks 7s → ✓ unit-tests 10s

✓ SayHello 3s

Adding a Status Badge!

https://docs.github.com/en/actions/how-tos/monitor-workflows/add-a-status-badge

Show off your test status, which is an SVG image at this URL:

```
https://github.com/OWNER/REPOSITORY/actions/workflows/WORKFLOW-FILE/badge.svg
```

- Example: In README.md, I can add this for my repo:

```
![[pkoffee status](https://github.com/kosack/pkoffee/actions/workflows/test.yaml/badge.svg)
```

Test
passing

Advanced Integration Testing:

How to ensure you code works in different environments?

What changes where code works?

My code works fine on my linux laptop, does it also work on different:

- **OS**? (e.g. Linux, Windows, MacOS?)
- **OS distribution**? (ubuntu, alama, etc)
- **OS Version**? (<os>-latest changes in time!)
- **CPU Architecture**? (x86, ARM)
- **Word size** (32-bit, 64 bit?)

You have to decide what is important: **what is your user base expecting?**

Does it work on newer/older versions of?

- **Python**: (python-3.13, 3.12, 3.11...)
- Other core dependencies like **matplotlib, numpy, scipy**?

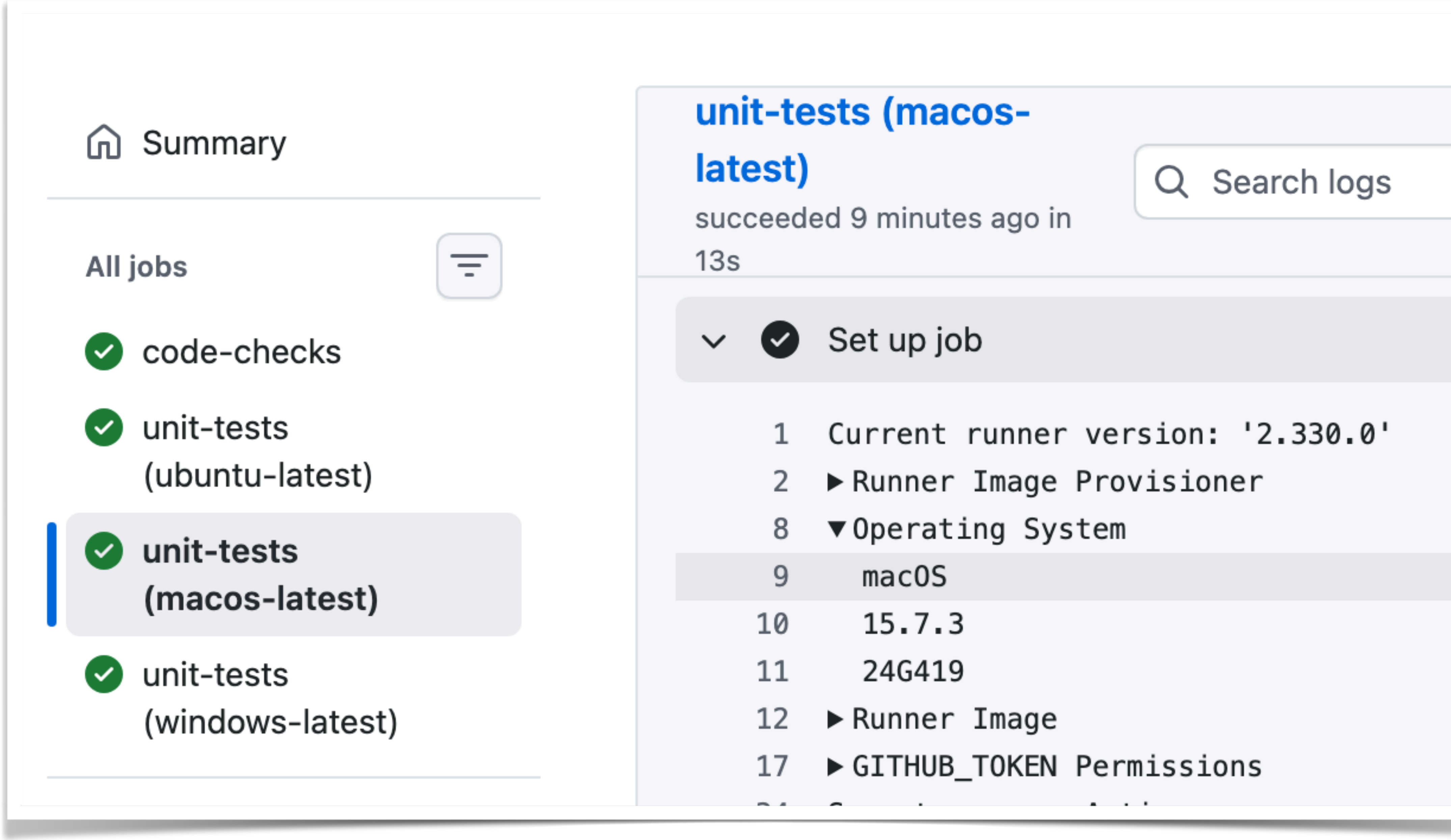
Using ***pixi***, mostly solves this

But sometimes it's nice to keep up with the latest developments and see incompatibilities automatically!

Matrix jobs to the rescue!

Matrix jobs let you run the same job over a N-dimensional "grid" of options

```yaml
unit-tests:
  runs-on: ${{ matrix.os }}
  needs: code-checks # don't bother running tests if code-checks fails.

  strategy:
    fail-fast: false # stop all jobs if one fails
    matrix:
      os:  # first dimension (and only in this case)
        - ubuntu-latest
        - macos-latest
        - windows-latest

  steps:
    - name: Checkout the code
      uses: actions/checkout@v6

    - name: Install pixi
      uses: prefix-dev/setup-pixi@v0.9.3
      with:
        pixi-version: v0.62.2 # good to pin this since pixi is not 1.0 yet!

    - run: pixi run pytest --verbose
```


Summary
All jobs
code-checks
unit-tests (ubuntu-latest)
unit-tests (macos-latest)
unit-tests (windows-latest)
Search logs
Set up job

unit-tests (macos-latest)

succeeded 9 minutes ago in 13s

```
Current runner version: '2.330.0'
▶Runner Image Provisioner
▼Operating System
  macOS
  15.7.3
  24G419
▶Runner Image
▶GITHUB_TOKEN Permissions
```

What about multiple Python versions?

Pixi's base environment has a fixed version of python...

- we can fix that by adding "features" that support different versions of python, and then defining "environments" that use those features for testing.

➤ Set up **features** in pixi.toml:

```
[feature.py313.dependencies]
python = "~=3.13.0"
[feature.py312.dependencies]
python = "~=3.12.0"
```

➤ Define **environments** that use those features in pixi.toml

environment-name = ["list", "of , "features"] ← note that "default" is implicit.

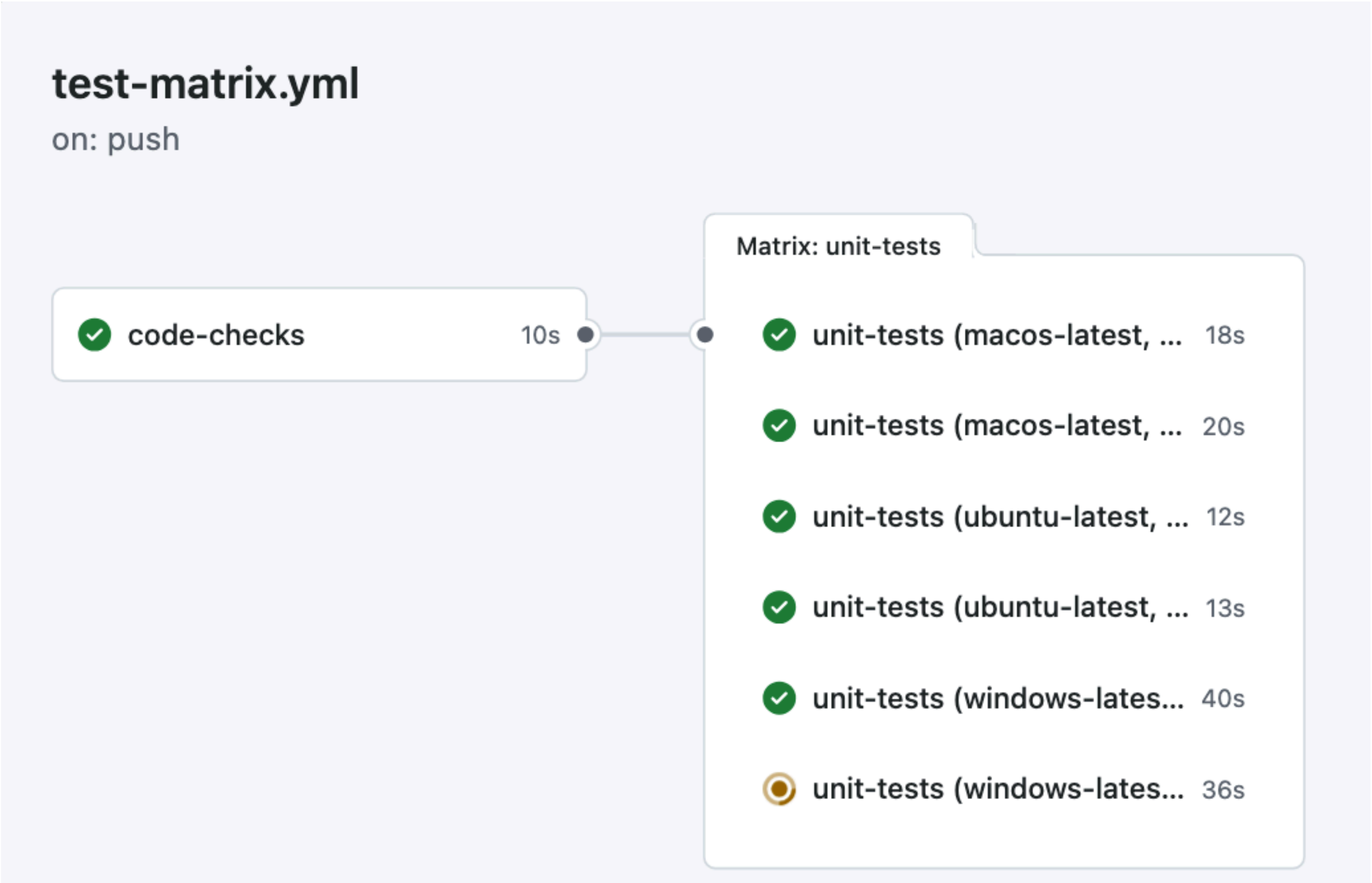
```
[environments]
env-py313 = ["py313"]
env-py312 = ["py312"]
```

Locally you can run: `pixi run -e env-py313 pytest`

Matrix 2D: Also test python version...

Matrix jobs let you run the same job over a N-dimensional "grid" of options

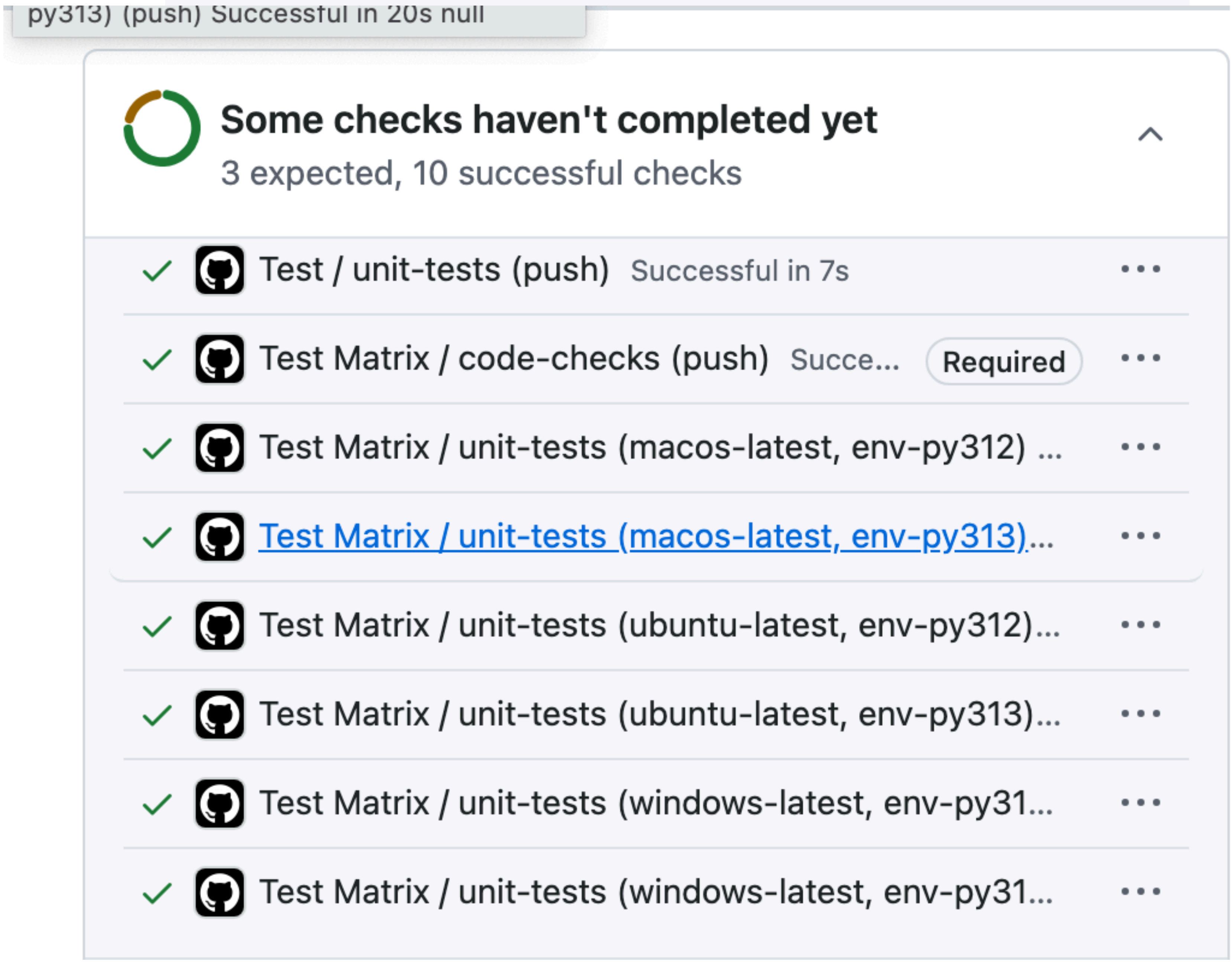
```yaml
unit-tests:
  runs-on: ${{ matrix.os }}
  needs: code-checks # don't bother running tests if code-checks fails.

  strategy:
    fail-fast: false # stop all jobs if one fails
    matrix:
      os:
        - ubuntu-latest
        - macos-latest
        - windows-latest
      environment:
        - env-py313
        - env-py312

  steps:
    - name: Checkout the code
      uses: actions/checkout@v6

    - name: Install pixi
      uses: prefix-dev/setup-pixi@v0.9.3
      with:
        pixi-version: v0.62.2 # good to pin this since pixi is not 1.0 yet!
        environments: ${{ matrix.environment }}

    - run: pixi run -e ${{ matrix.environment }} pytest
```



test-matrix.yml
on: push

code-checks 10s

Matrix: unit-tests
- unit-tests (macos-latest, ... 18s
- unit-tests (macos-latest, ... 20s
- unit-tests (ubuntu-latest, ... 12s
- unit-tests (ubuntu-latest, ... 13s
- unit-tests (windows-lates... 40s
- unit-tests (windows-lates... 36s



py313) (push) Successful in 20s null

Some checks haven't completed yet
3 expected, 10 successful checks

- ✓ Test / unit-tests (push) Successful in 7s
- ✓ Test Matrix / code-checks (push) Succe... Required
- ✓ Test Matrix / unit-tests (macos-latest, env-py312) ...
- ✓ Test Matrix / unit-tests (macos-latest, env-py313)...
- ✓ Test Matrix / unit-tests (ubuntu-latest, env-py312)...
- ✓ Test Matrix / unit-tests (ubuntu-latest, env-py313)...
- ✓ Test Matrix / unit-tests (windows-latest, env-py31...
- ✓ Test Matrix / unit-tests (windows-latest, env-py31...

Final version: test.yaml

```yaml
name: Test Matrix
on: [push]

jobs:

  code-checks:
    runs-on: ubuntu-latest

    steps:
      - name: Checkout the code
        uses: actions/checkout@v6

      - name: Install pixi
        uses: prefix-dev/setup-pixi@v0.9.3
        with:
          pixi-version: v0.62.2

      - name: look for syntax errors
        run: pixi run ruff check

  unit-tests:
    runs-on: ${{ matrix.os }}
    needs: code-checks

    strategy:
      fail-fast: false # stop all jobs if one fails
      matrix:
        os:
          - ubuntu-latest
          - macos-latest
          - windows-latest
        environment:
          - default
          - env-py313
          - env-py312

    steps:
      - name: Checkout the code
        uses: actions/checkout@v6

      - name: Install pixi
        uses: prefix-dev/setup-pixi@v0.9.3
        with:
          pixi-version: v0.62.2
          environments: ${{ matrix.environment }}

      - run: pixi run -e ${{ matrix.environment }} pytest
```

Continuous Delivery

How to automatically package your code and share it?

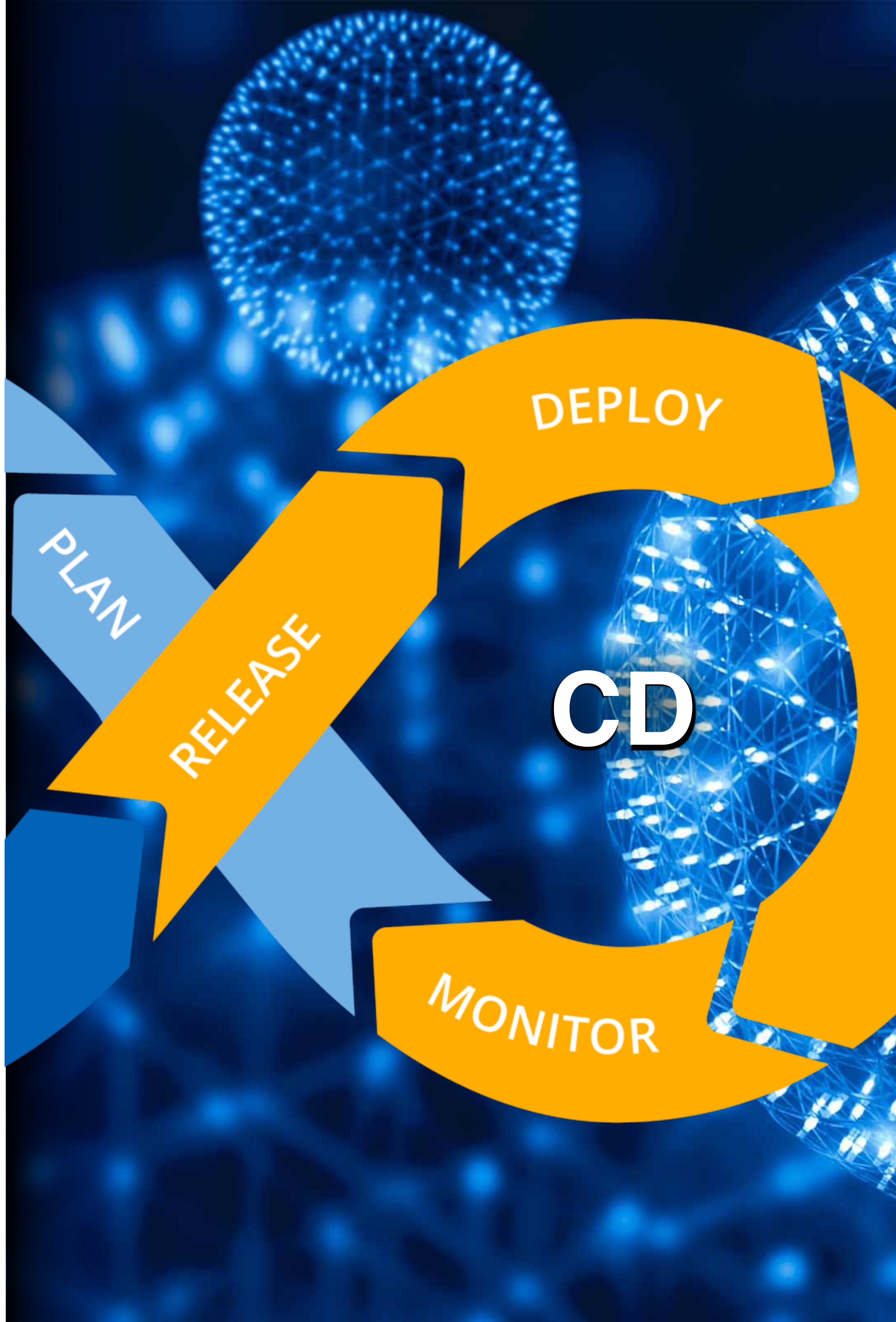
What is Continuous Delivery?

- Automation of software releasing [on: release]
  - ➤ create packages
  - ➤ upload to package archives
- Automatic generation of other artifacts:
  - ➤ package **documentation** + upload to a host like *GitHub Pages*
  - ➤ even generate **analysis results***

*__caveat:__ only use this for small/simple analyses, as GitHub doesn't give you much storage and can limit computing

Why is it useful?

- Make your code always easily sharable
- Keep documentation up to date
- Reduce human burden!
  - ➤ "I didn't have time to update the docs..."



How does CD work?

It's the same as CI!

Add workflow to GitHub Actions (or equivalent)

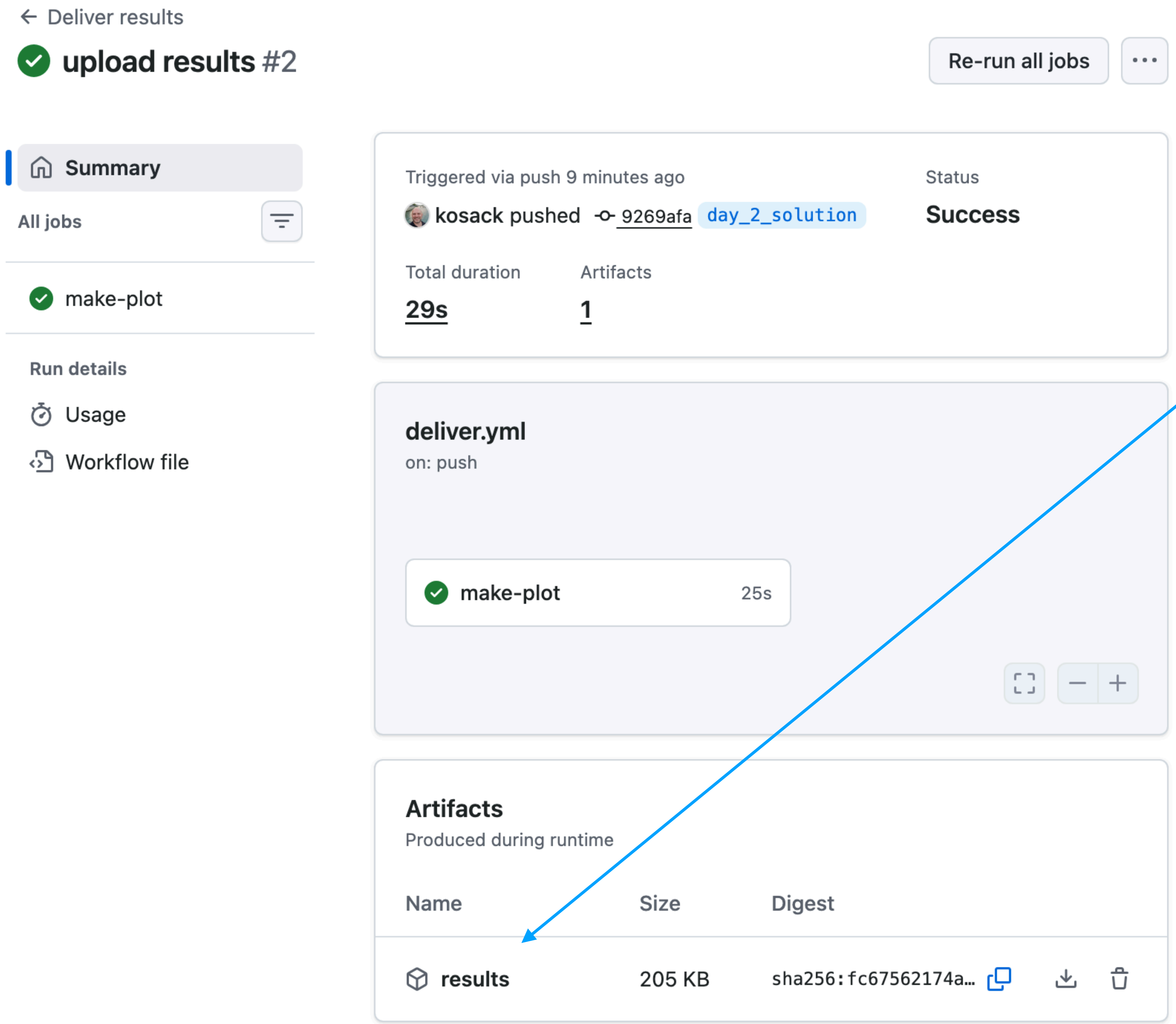
- **good practice**: create new workflow(s) for CD tasks, rather than adding new jobs to our *test* workflow.
  - ➤ Select different *trigger (runs-on:)*, e.g. only run on "**release**", or "**merge**"

Let's make an example:

- Run our analysis (on push for now)
- Expose the final plot artifact

DEMO

Where did the artifact go?



← Deliver results

✓ upload results #2

Re-run all jobs

Summary

All jobs

✓ make-plot

Run details

Usage

Workflow file

Triggered via push 9 minutes ago

kosack pushed 9269afa day_2_solution

Status

Success

Total duration

29s

Artifacts

1

deliver.yml

on: push

✓ make-plot 25s

Artifacts

Produced during runtime

Name	Size	Digest
results	205 KB	sha256:fc67562174a…

The upload-artifact block uploads to GitHub's free (but not long-term) artifact storage

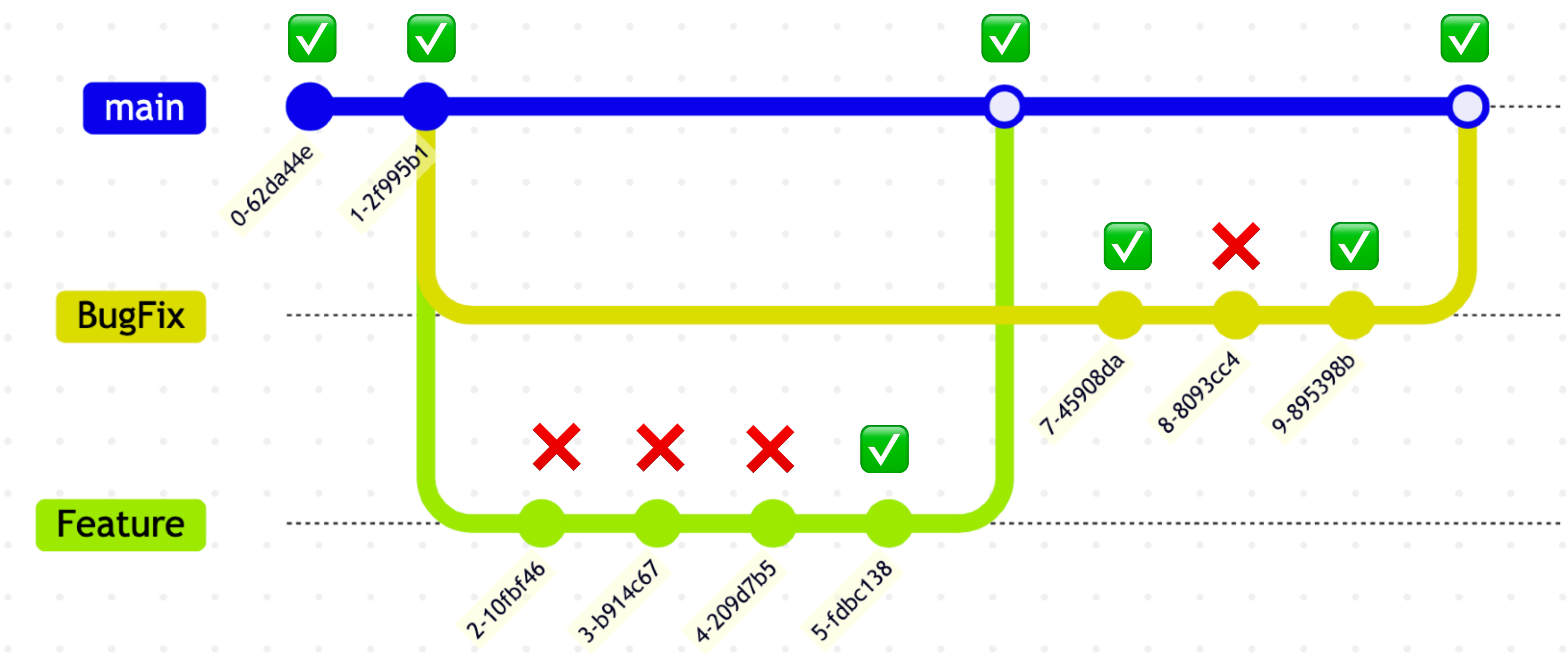
- You can see it in the workflow status view, and download them as a .zip file.

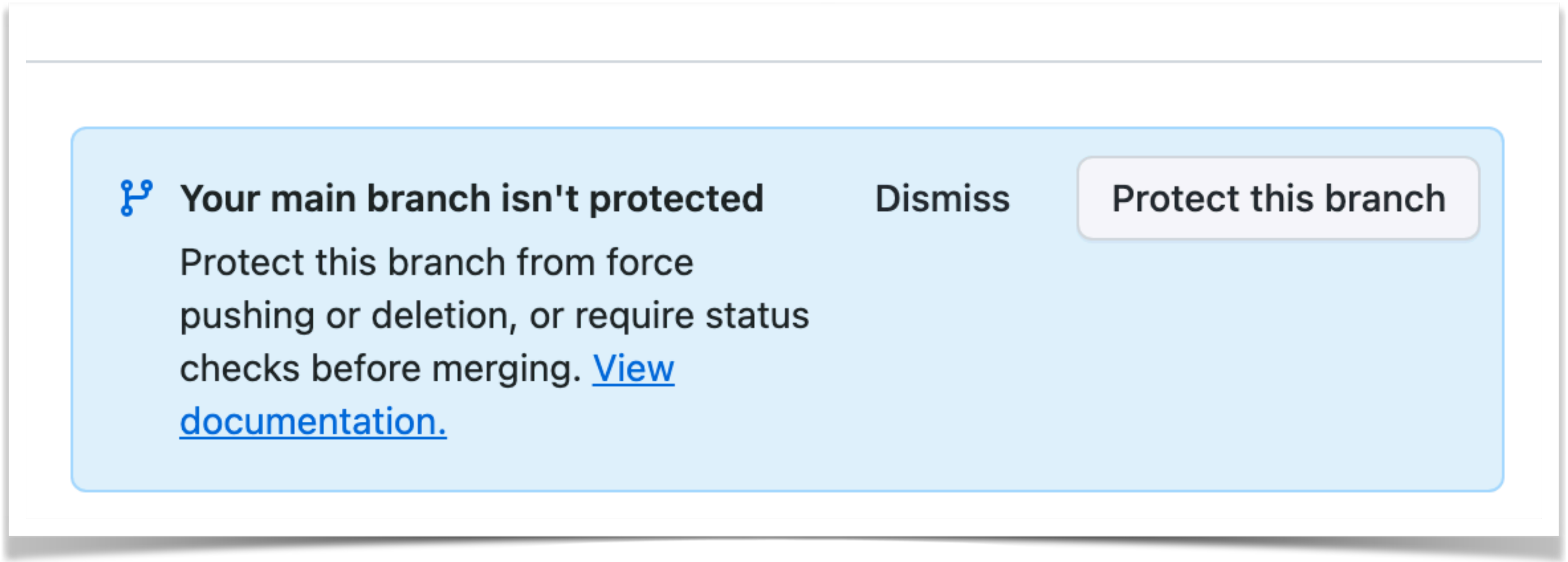
Another way you could auto-generate results:

- Add them to the documentation or a Jupyter Notebook and build them in the CI!
  - ➤ See lecture next week on adding documentation!

Advanced Topics

CI: Enforcement of Development Workflow



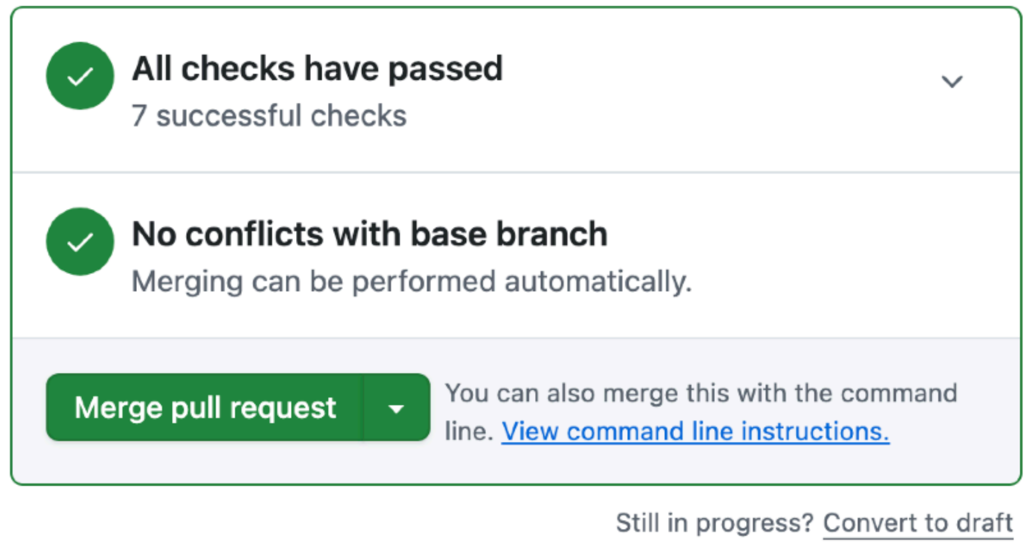


Your main branch isn't protected

Dismiss

Protect this branch

Protect this branch from force pushing or deletion, or require status checks before merging. View documentation.



All checks have passed
7 successful checks

No conflicts with base branch
Merging can be performed automatically.

Merge pull request ▾

You can also merge this with the command line. View command line instructions.

Still in progress? Convert to draft

Recommended: the "Github Flow" model

- The *main* branch of repo is always **green and ready for delivery**.
- **Forbid** direct pushes to *main*
- Code changes require **branches + pull requests**
- Pull-requests only merged **if the CI tests pass!**

GitHub can enforce this

Is this required?

- No, you can still benefit from CI even if you are one dev who just pushes to main with no branches
- **However!** it will make your life simpler if you always have a working version to compare to!

Configuring merge restrictions

General

Access

Collaborators

Moderation options

Code and automation

Branches

Tags

Rules

Rulesets

Actions

Models

Webhooks

Copilot

Environments

Codespaces

Pages

Security

Advanced Security

Deploy keys

Secrets and variables

Rulesets / New branch ruleset

Protect your most important branches

Rulesets define whether collaborators can delete or force push and set requirements for any pushes, such as passing status checks or a linear commit history.

Ruleset Name *

enforce workflow

Enforcement status

Active

Bypass list — + Add bypass

Exempt roles, teams, and apps from this ruleset by adding them to the bypass list.

Bypass list is empty

Target branches

Which branches should be matched?

Branch targeting criteria — Add target

Default ← target the main branch

Branch rules

☐ Restrict creations
Only allow users with bypass permission to create matching re

☐ Restrict updates
Only allow users with bypass permission to update matching r

☑ Restrict deletions
Only allow users with bypass permissions to delete matching r

☐ Require linear history
Prevent merge commits from being pushed to matching refs.

☐ Require deployments to succeed
Choose which environments must be successfully deployed tc pushed into a ref that matches this rule.

☐ Require signed commits
Commits pushed to matching refs must have verified signature

☑ Require a pull request before merging
Require all commits be made to a non-target branch and subm before they can be merged.

Hide additional settings ^

Required approvals

0 ← If working in a team, recommend set to ≥1!

The number of approving reviews that are required before a pull request can be merged.

☐ Dismiss stale pull request approvals when new commits are pushed
New, reviewable commits pushed will dismiss previous pull request review approvals.

☐ Require review from specific teams Preview
A collection of reviewers and associated file patterns. Each reviewer has a list of file patterns which determine the files that reviewer is required to review.

☐ Require review from Code Owners
Require an approving review in pull requests that modify files that have a

☑ Require status checks to pass
Choose which status checks must pass before the ref is updated. When enabled, commits must first be pushed to another ref where the checks pass.

Hide additional settings ^

☐ Require branches to be up to date before merging
Whether pull requests targeting a matching branch must be tested with the latest code. This setting will not take effect unless at least one status check is enabled.

☐ Do not require status checks on creation
Allow repositories and branches to be created if a check would otherwise prohibit it.

Status checks that are required	+ Add checks
unit-tests (macos-latest)	GitHub Actions
unit-tests (ubuntu-latest)	GitHub Actions
unit-tests (windows-latest)	GitHub Actions
code-checks	GitHub Actions

Add all CI checks you think MUST pass before a PR can be merged (doesn't have to be all)

Topics that require next week's lectures...

Automatic packaging:

- **build** python wheel, conda package, Docker container
- **deploy** package to repository like pypi, github's container repo

Automatic documentation:

- **build** documentation (sphinx, htmldoc)
- **deploy** rendered documentation to github pages

Tip:

deploy workflows for packages should only be on: release

Tip:

deploy workflows for documentation can be set up to work *on: push* (preview) as well as *on: release*

Activity: Implement CI and CD for your project

1. Implement a *Test* workflow

- check code quality (ruff)
- run unit tests
- generate test coverage report
- ensure the documentation can build

Optional Advanced tasks:

- Add a status badge to your README.md
- use a matrix to run unit tests on several environments
- add a test that updates your dependencies to the latest version and runs the unit tests. It should be marked as

2. Implement a simple *Delivery* workflow(s):

- run your code to produce a plot and expose it as an artifact
    - ➤ Next week, after you add documentation and packaging, you can try automating that as well!