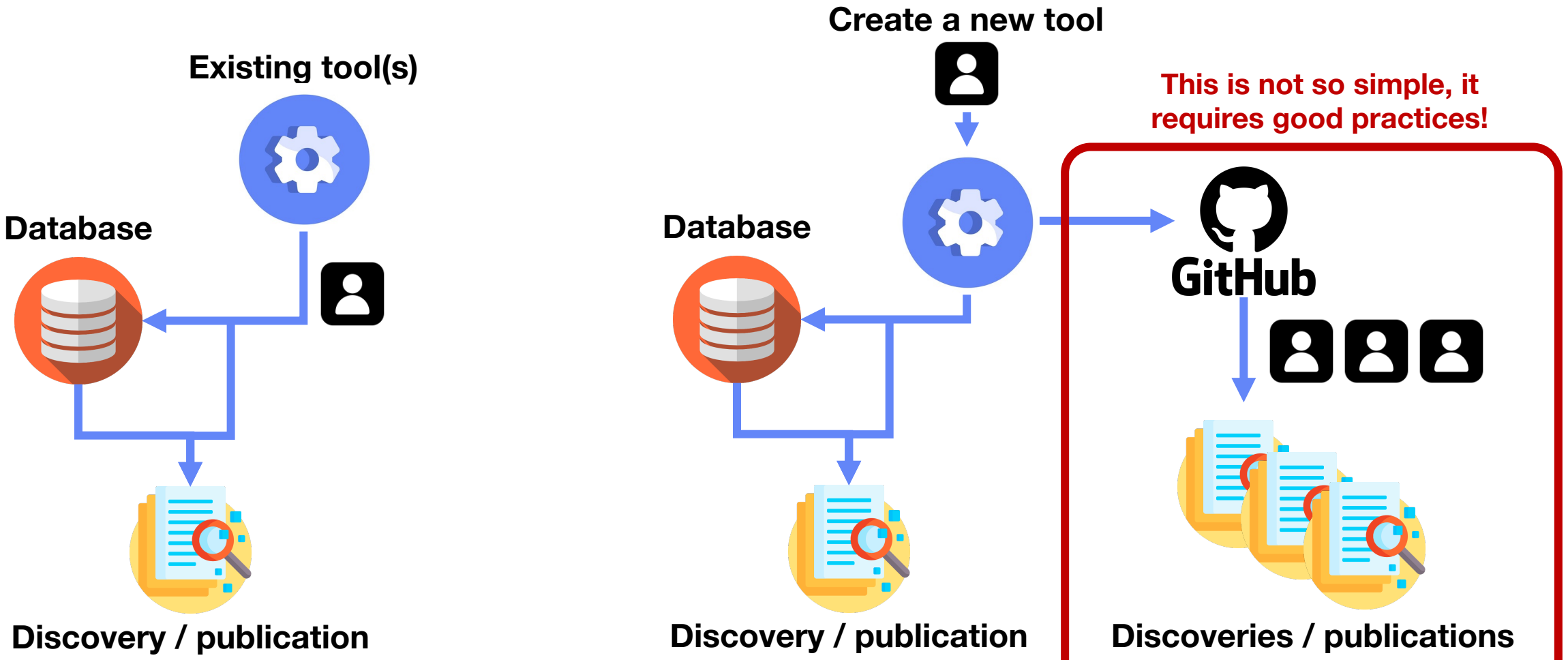
Good practices

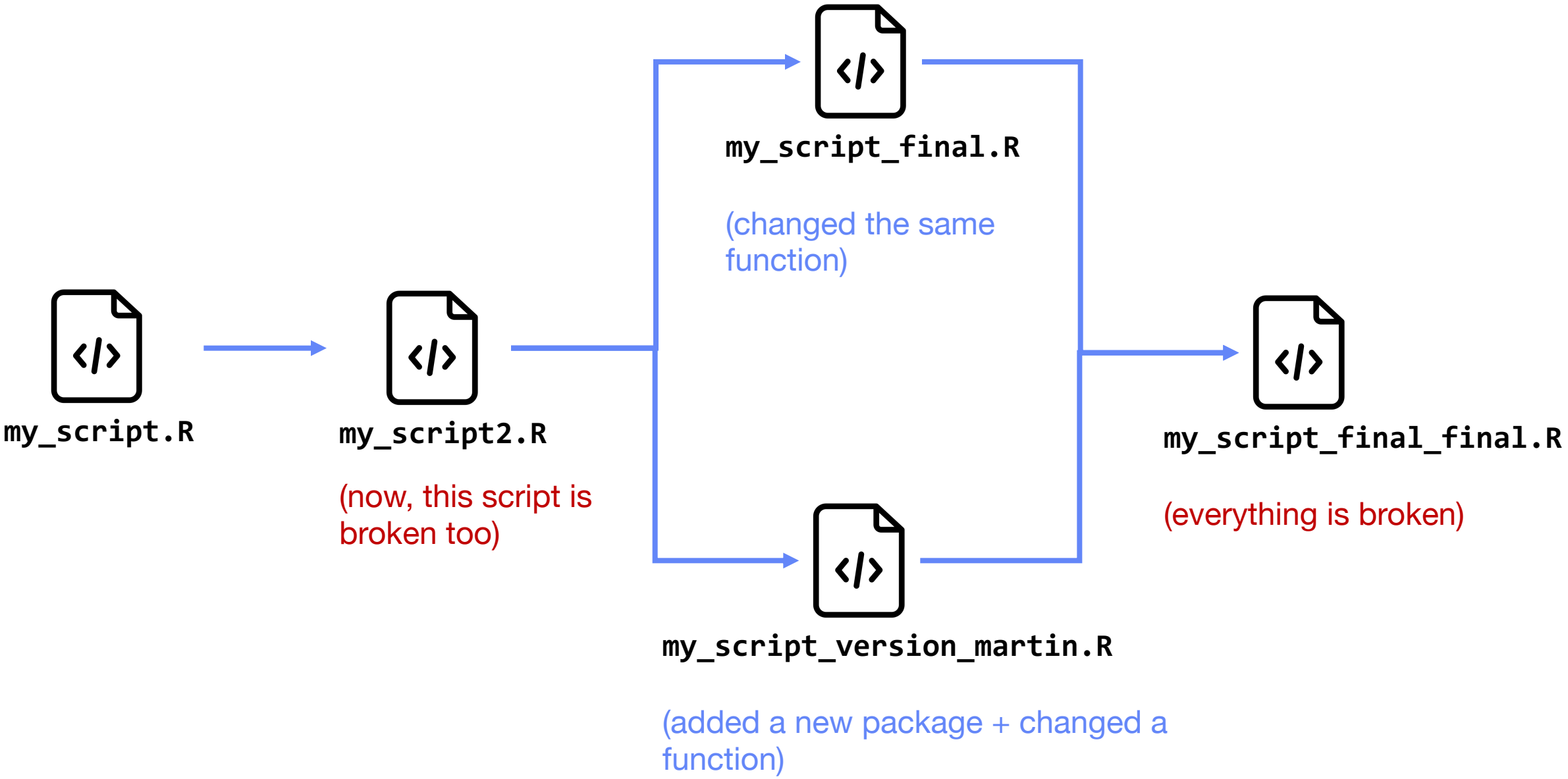
Nathalie Lehmann and Quentin Blampey

Link: menti.com/ale44temte5w

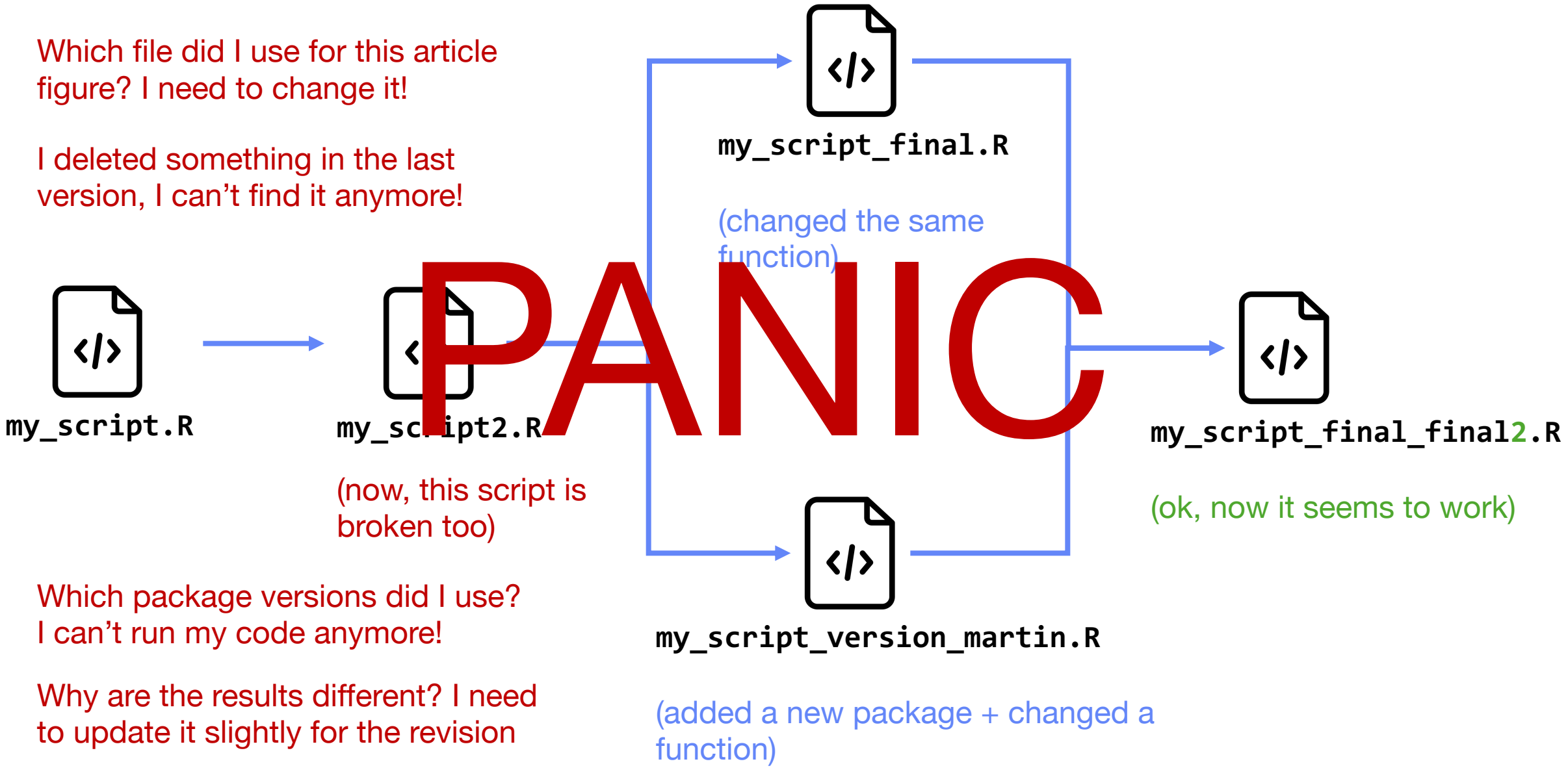
Different types of bioinformatics projects



Why using a versioning tool (e.g., Git)



Why using a versioning tool (e.g., Git)



Why using a versioning tool (e.g., Git)

+

Source code hub

Writing readable and understandable code

What is this function doing? What is "data"? To what the arguments correspond? What are the possible values? What is the result?

```
def run(data, c=7, a=13.2, m="dense"):
    ... # some code
    return results
```

Provide meaningful function AND variable names (self-understandable code)
Document your input types and available values
Describe precisely the output

Writing readable and understandable code

If you can: try to write an online documentation

Sopa | Search | gustaveroussy/sopa v1.1.5 128 15

Sopa
- Home
- Getting Started
- Tutorials
- CLI
- API
- Frequently asked questions
- Cite us

Spatial-omics pipeline and analysis

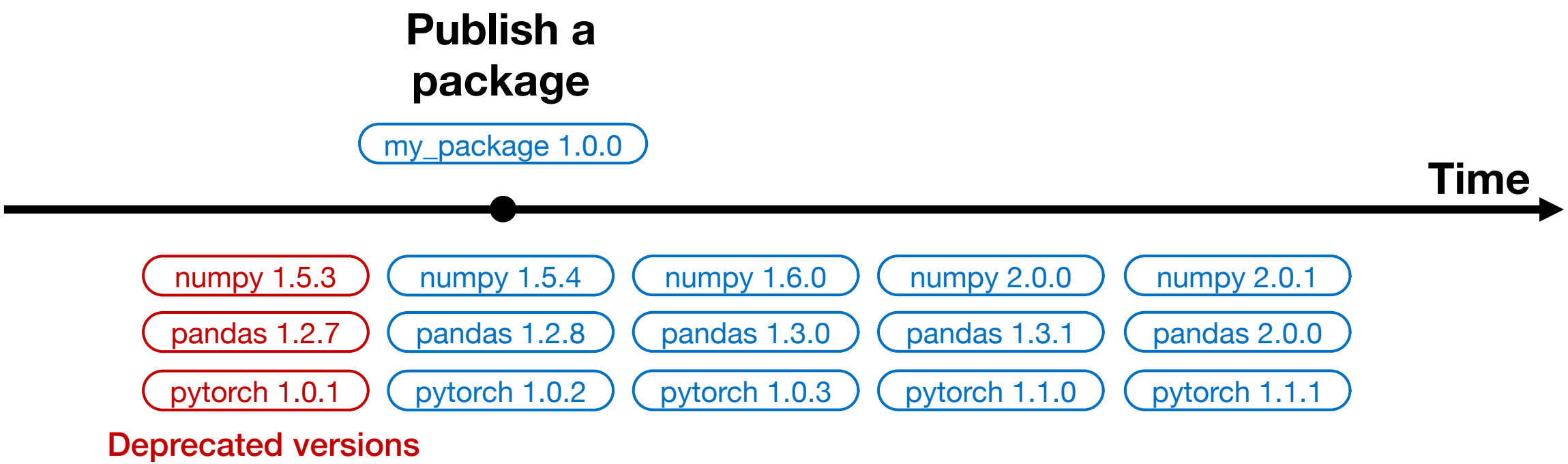
Table of contents
- Overview
- Why use sopa

Built on top of SpatialData, Sopa enables processing and analyses of image-based spatial omics using a standard data structure and output. We currently support the following technologies: Xenium, MERSCOPE, CosMX, PhenoCycler, MACSIMA, Hyperion. Sopa was designed for generability and low memory consumption on large images (scales to `1TB+` images).

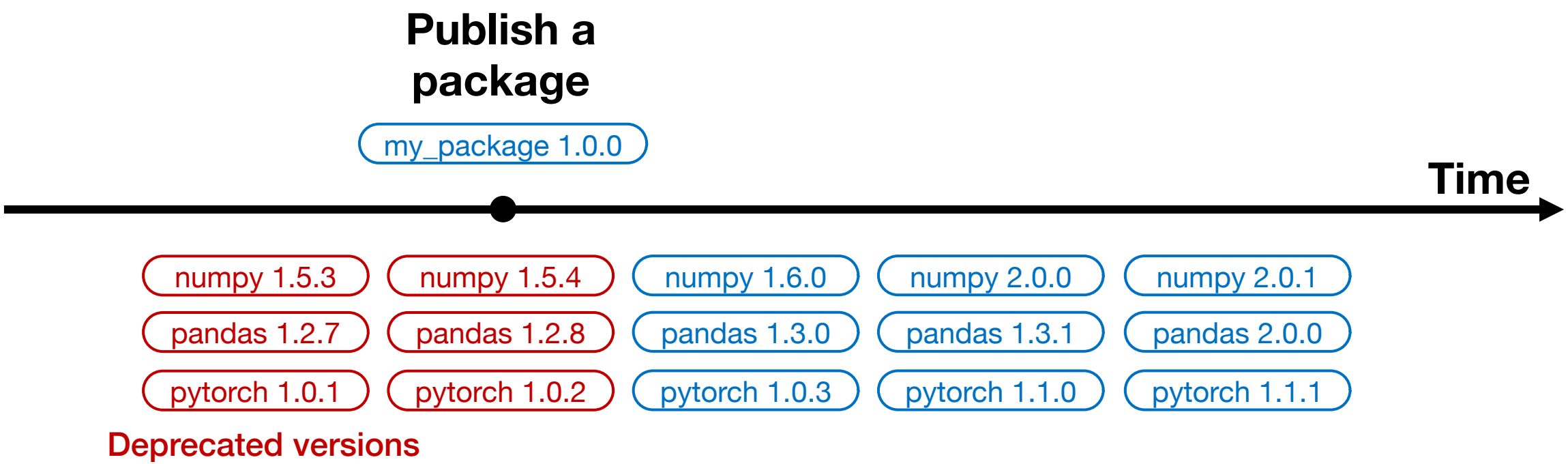
https://gustaveroussy.github.io/sopa/

Maintaining a package over time

Maintaining a package over time

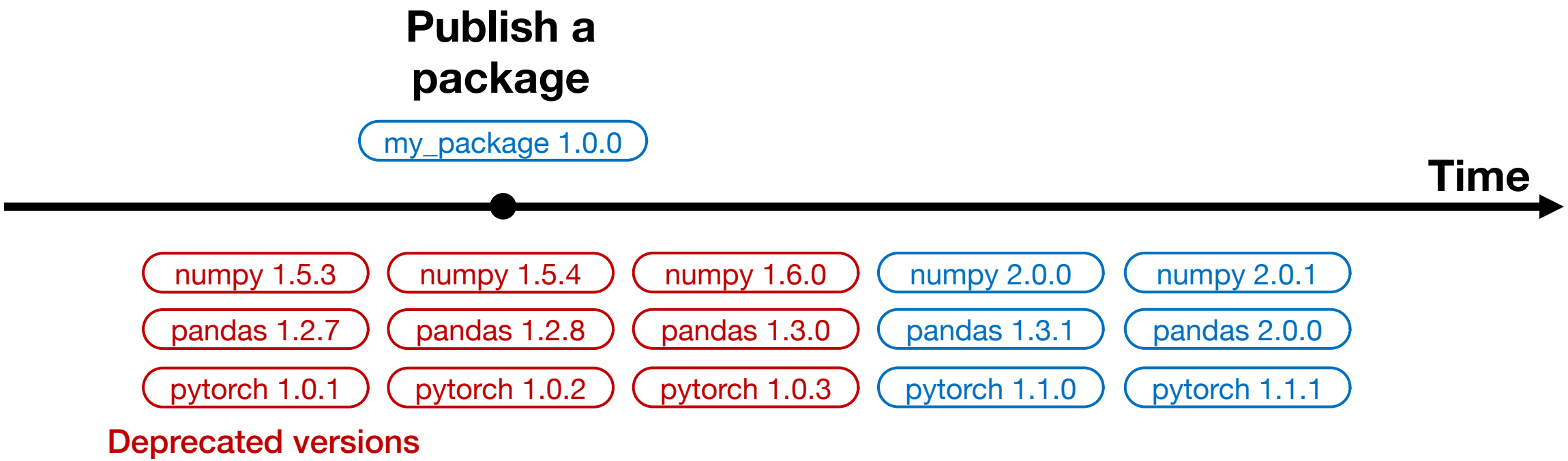


Maintaining a package over time



Maintaining a package over time

If you don't maintain your code, it will eventually become deprecated!



Backward compatibility

my_package 1.0.0

```python
def annotate_cell_types(X: np.ndarray, d: dict):
  ...
```

New version

my_package 1.1.0

```python
def get_cell_types(X: np.ndarray, markers: dict):
  ...
```

It will break everything! E.g. Seurat

Solution 1:

```python
def annotate_cell_types(X: np.ndarray, d: dict):
  warning("This function is deprecated and will be removed in Jan. 2025")

  return get_cell_types(X, markers=d)
```

Backward compatibility

Solution 2:
Release notes

v0.2.2

[0.2.2] - 2024-08-07

Major

- New disk format for shapes using `GeoParquet` (the change is backward compatible) #542

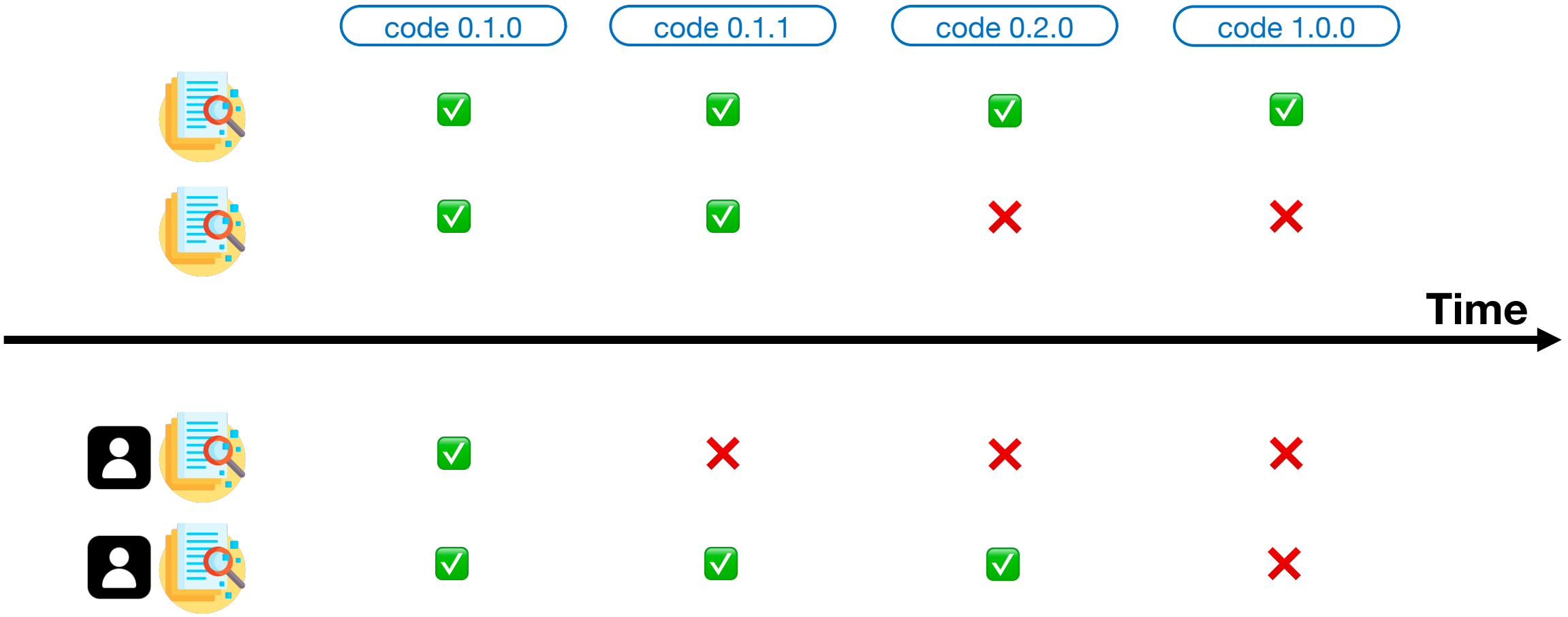
Minor

- Add `return_background` as argument to `get_centroids` and `get_element_instances` #621
- Ability to save data using older disk formats #542

Fixed

- Circles validation now checks for inf or nan radii #653
- Bug with table name in torch dataset #654 **@LLehner**

Testing and continuous integration (CI)



Which change broke what? And when? On which OS? On which use case?

Testing and continuous integration (CI)

Testing = write some code that tests if everything is working

```
============================= test session starts ==============================
platform linux2 -- Python 2.7.3 -- py-1.4.20 -- pytest-2.5.2 -- /usr/bin/python
collected 9 items

test/test_basic_integers.c:14: test_some_integers() PASSED
test/test_basic_integers.c:15: test_some_integers() PASSED
test/test_basic_integers.c:21: test_more_integers() FAILED
test/test_basic_integers.c:22: test_more_integers() FAILED
test/test_basic_strings.c:16: test_some_strings() PASSED
test/test_basic_strings.c:17: test_some_strings() PASSED
test/test_basic_strings.c:26: test_more_strings() FAILED
test/test_basic_strings.c:27: test_more_strings() FAILED
test/test_basic_strings.c:28: test_more_strings() PASSED
```

It tells you when some specific use cases are broken

When everything is “passing”, you’ll be more confident that your code is stable

Testing and continuous integration (CI)

`.yaml` file

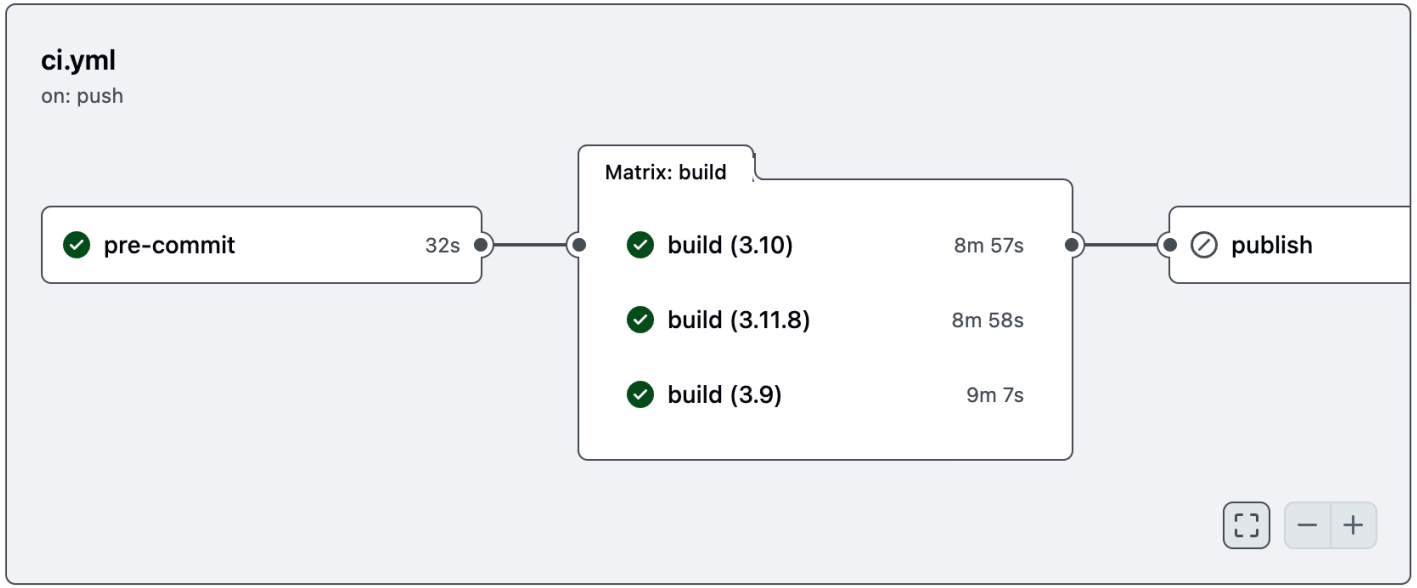
```yaml
on:
  push:
    branches: [master]

jobs:
  build:
    runs-on: ubuntu-latest

    - name: Install dependencies
      run: pip install -e .

    - name: Tests
      run: poetry run pytest

    - name: Deploy doc
      run: mkdocs gh-deploy
```

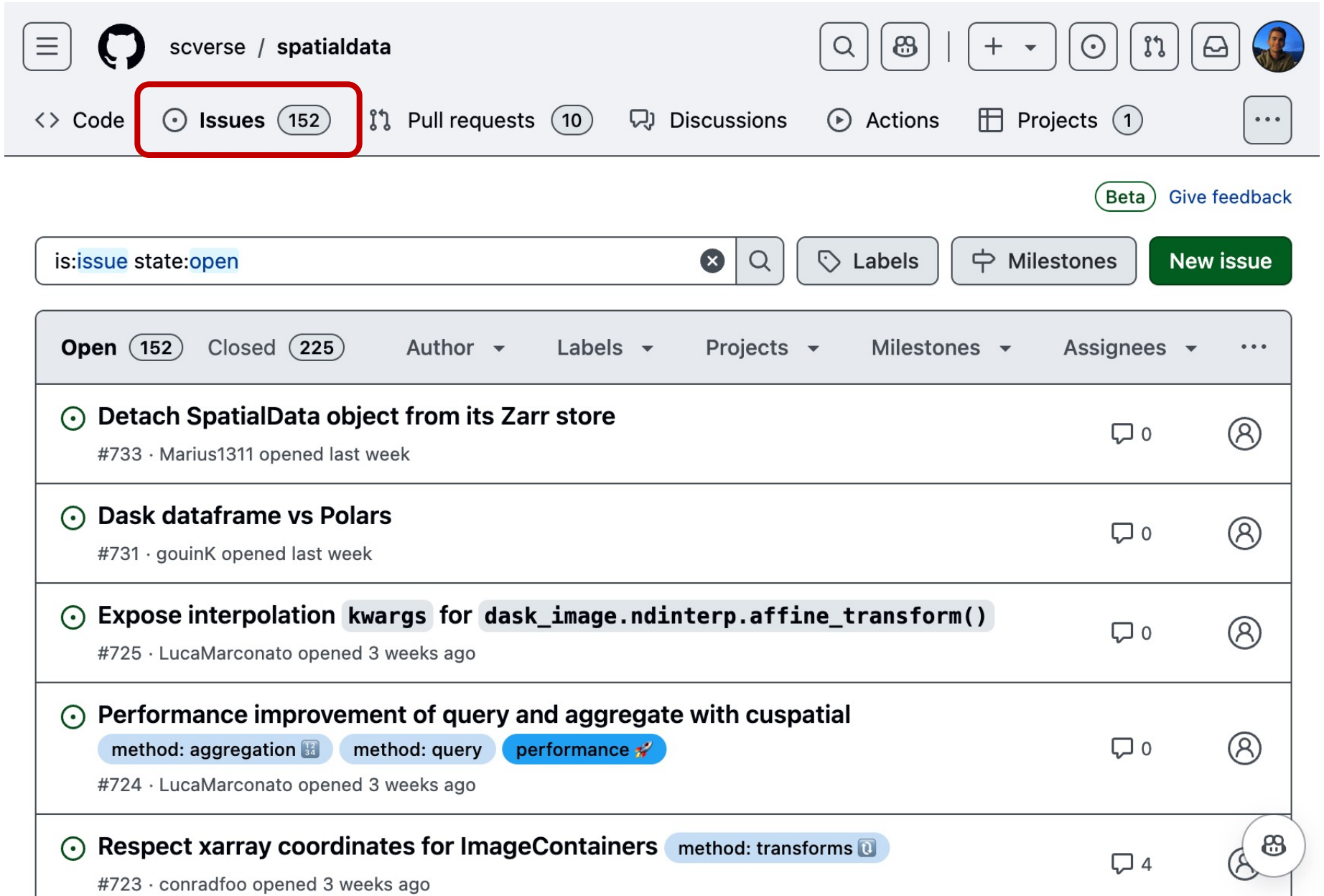


https://github.com/gustaveroussy/sopa/actions/runs/11442161900

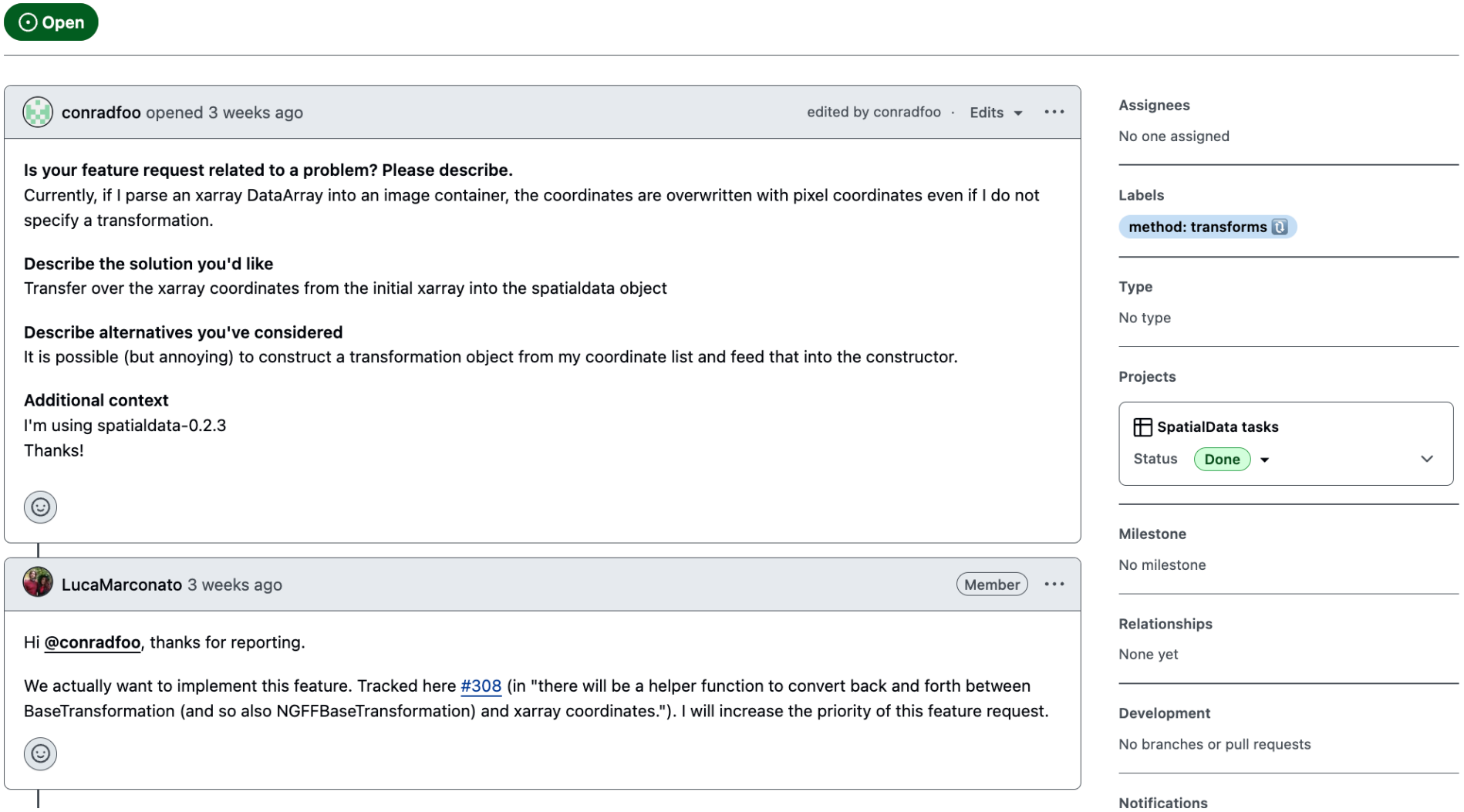
Other keywords to dive deeper

- Precise error handling (e.g., using assert)
- PEP style guide
- Formatting and linting (e.g., **`black`**, **`flake8`**, **`ruff`**, …)
- **`pre-commit`** tool
- Typing

Contributing to open-source



Contributing to open-source

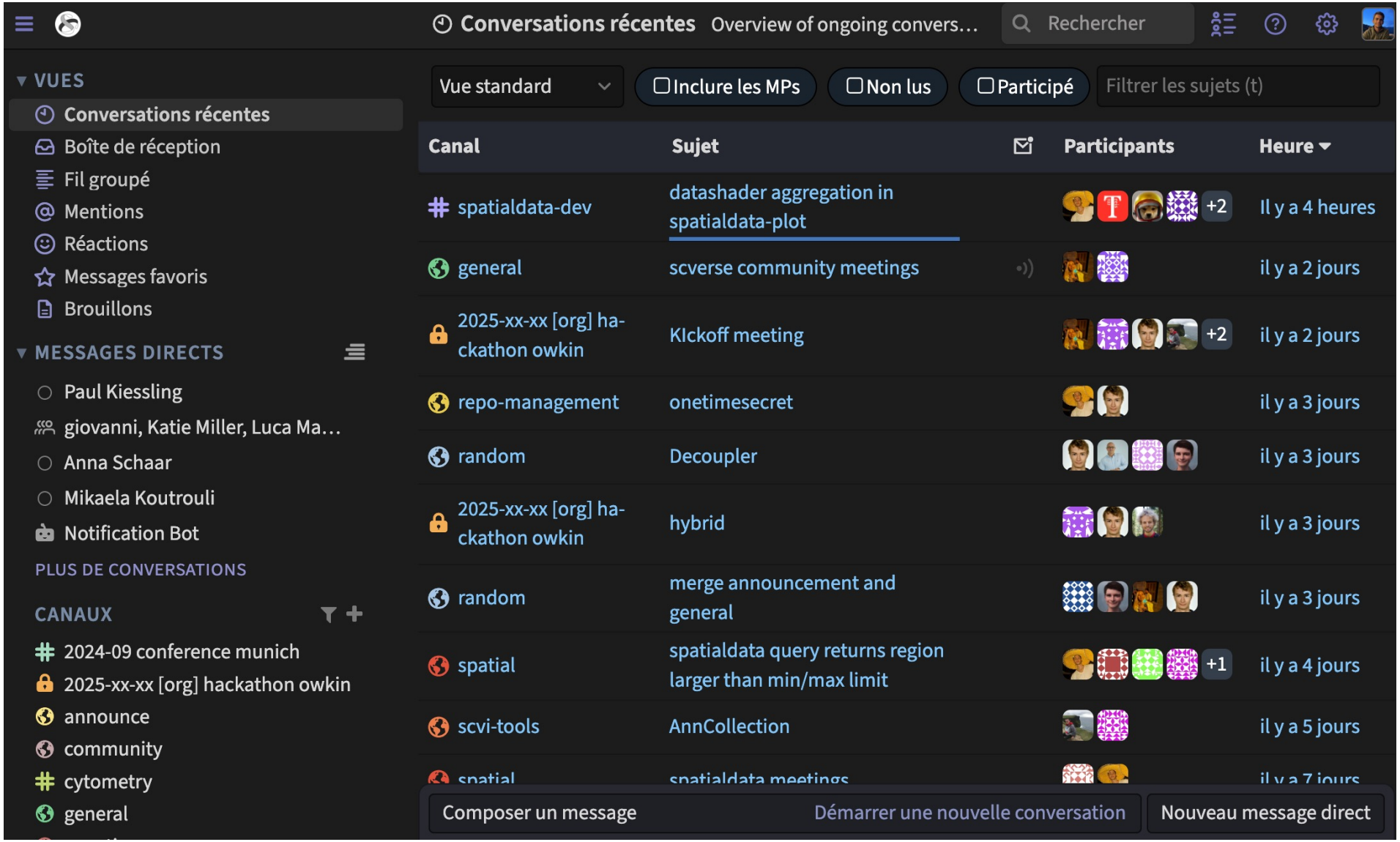


You can answer to other people issues (even if it’s not your own package)

If you fixed your issue, add a comment to explain how you fixed it

Contributing to open-source

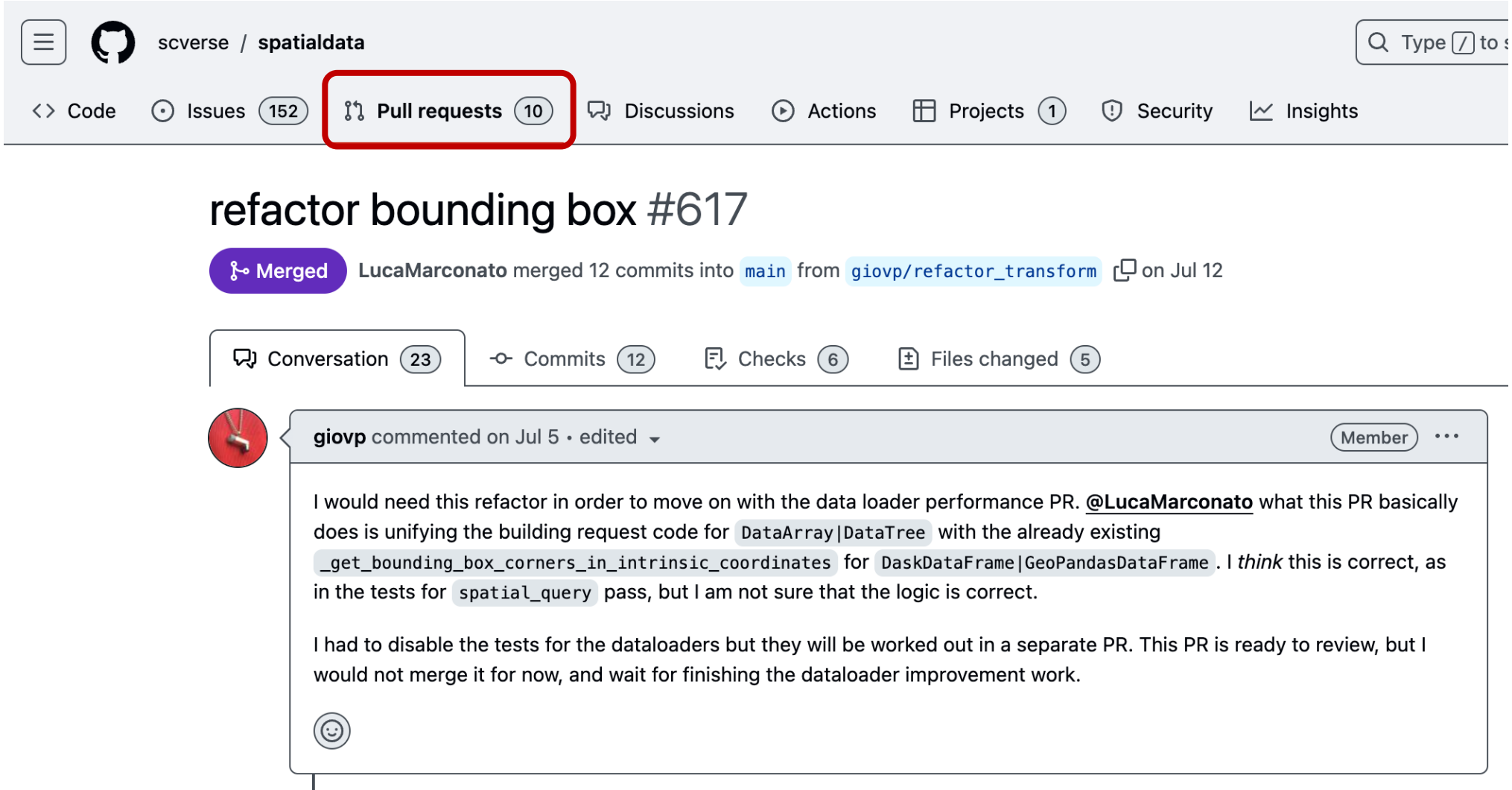
https://scverse.zulipchat.com/



Contributing to open-source (going further)

Pull Request example:

https://github.com/scverse/spatialdata/pull/617



Contributing to open-source (going further)

Create your own library:

- Good dependency manager
- Good testing (all OS, many Python/R versions)
- Good documentation
- Release your tool on PyPI / Bioconductor
- Manage your package over time