Fast GPU-powered and auto-differentiable forward modeling for cosmological hydrodynamical simulations

Anna Lena Schaible
annalena.schaible@iwr.uni-heidelberg.de

IWR Interdisciplinary Center for Scientific Computing

UNIVERSITÄT HEIDELBERG ZUKUNFT SEIT 1386

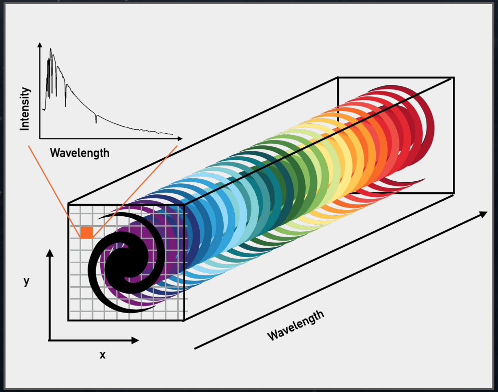
Carl Zeiss Stiftung

SCIENTIFIC SOFTWARE CENTER

IMPRS-HD IMPRS for Astronomy & Cosmic Physics at the University of Heidelberg

https://astro-rubix.web.app/

Observations of galaxies



GECKOS

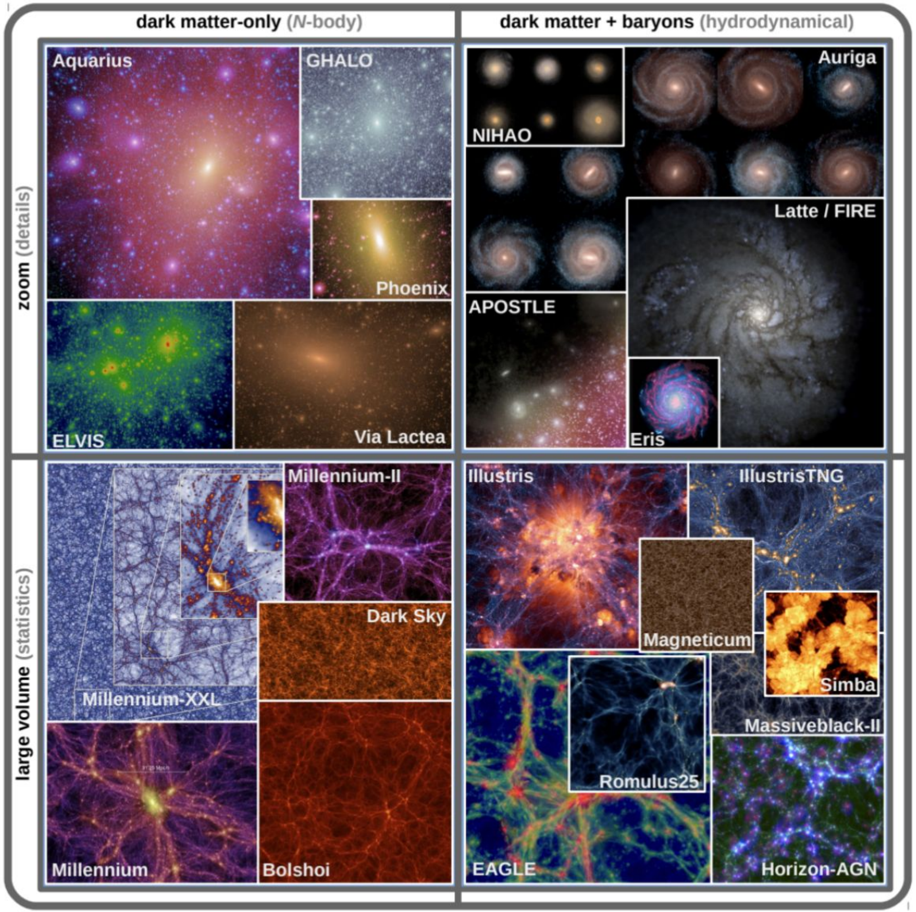
Cosmological Simulations

- Observations produce a lot of data

  → interpretation difficult

- Simulate galaxy formation using computational methods
- Assume cosmology, initial conditions and physical processes, then let it run

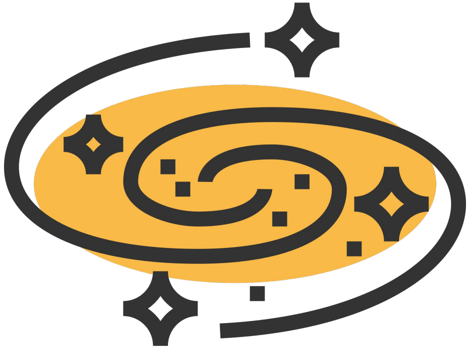
Good overview: Vogelsberger et al. 2019



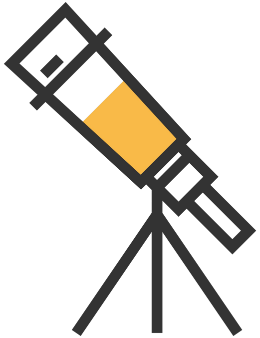
Vogelsberger et al. 2019

Methods of comparing simulations and observations

Cosmological hydrodynamical simulation:
Particles with physical properties

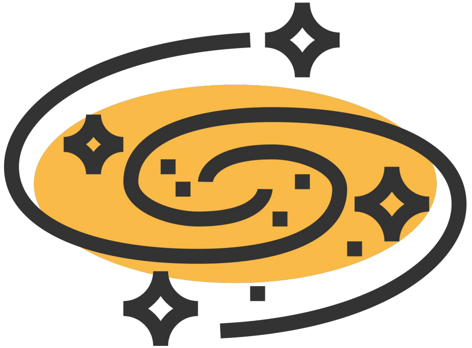


IFU observation:
photon counts/
integrated light



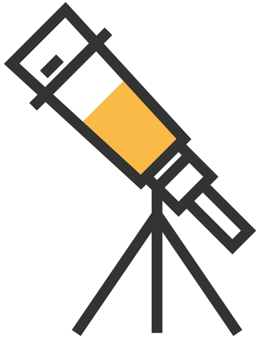
Methods of comparing simulations and observations

Cosmological hydrodynamical simulation: Particles with physical properties

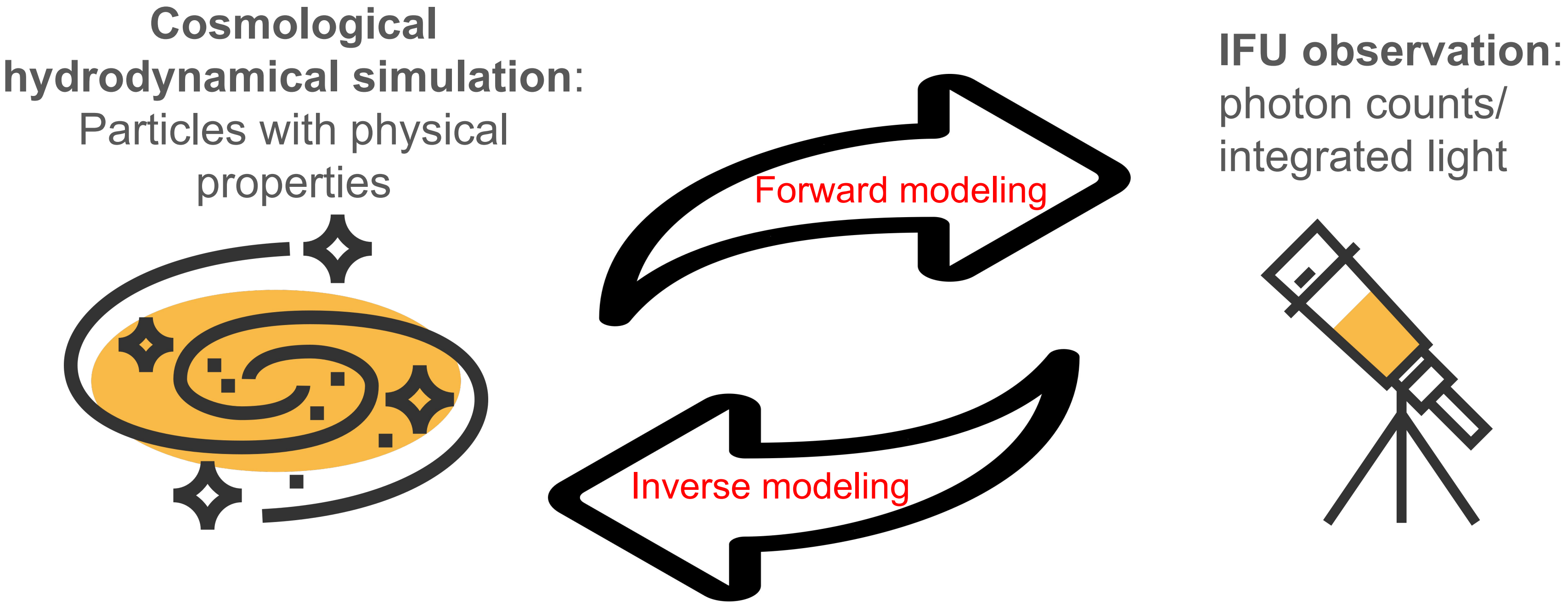


Forward modeling

IFU observation: photon counts/ integrated light



Methods of comparing simulations and observations



Other codes

SimSpin

Harborne et al. (2023)

- Written in R
- Great documentation

MaNGIA

Sarmiento et al. (2022)

- MaNGA mocks

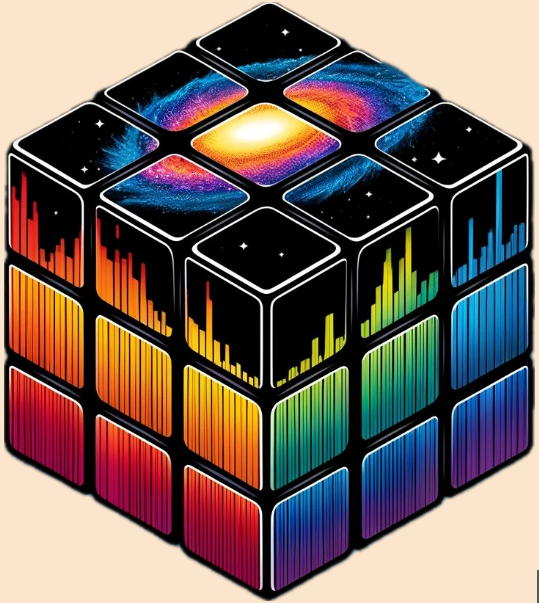
GalCraft (not yet public)

Wang et al. (2023)

- MW type galaxies

- Forward modeling
- CPU only
- Usually very slow

Virtual telescope: RUBIX



https://astro-rubix.web.app/

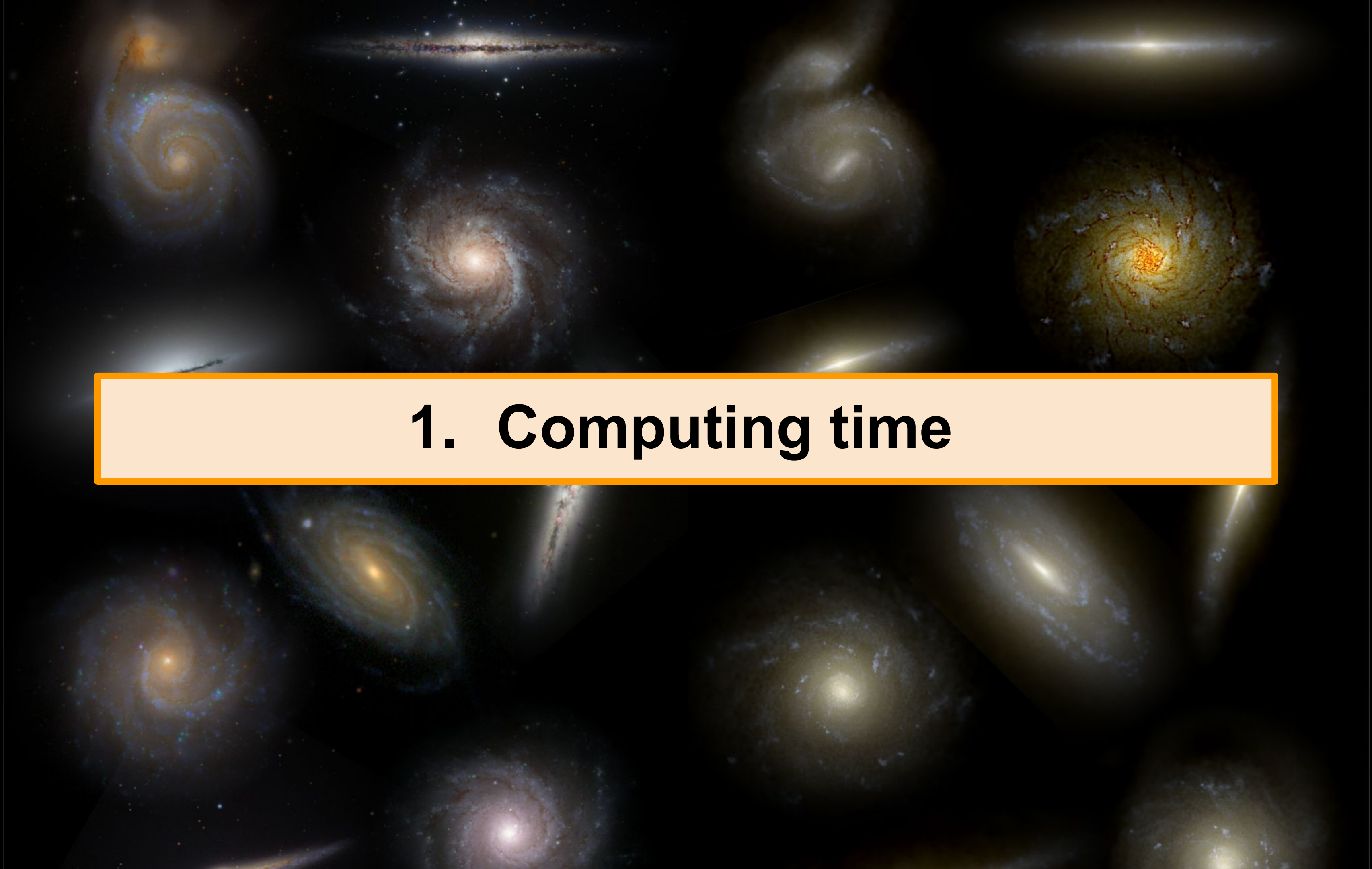
What is different to already existing software

1. Computation time

2. Code structure: research software practice for good open source software

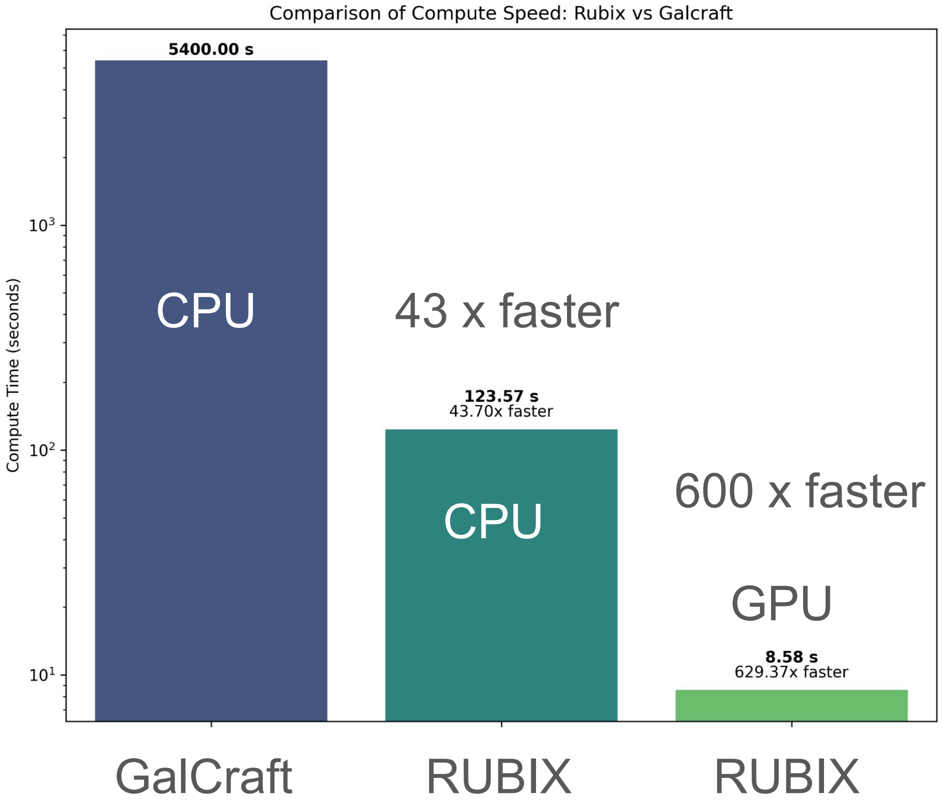
3. Forward modeling and inverse modeling

4. Applications in machine learning



1. Computing time

Speed Comparison



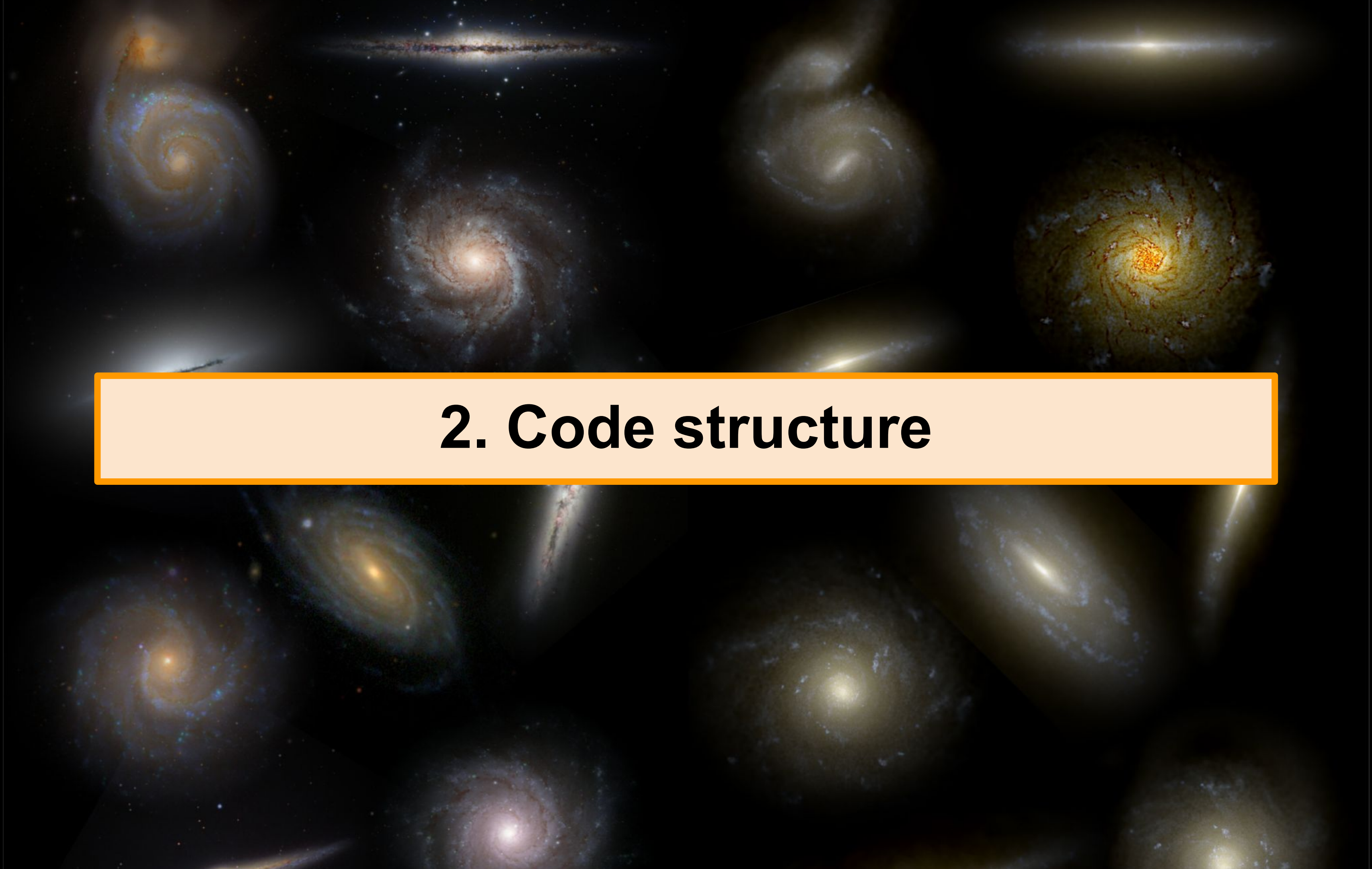
GalCraft (Wang et al 2023):

- 6×10^6 particles approx. 1.4 hours
- 24 Core CPU (2.5Ghz)

Rubix: 5.8×10^6 particles

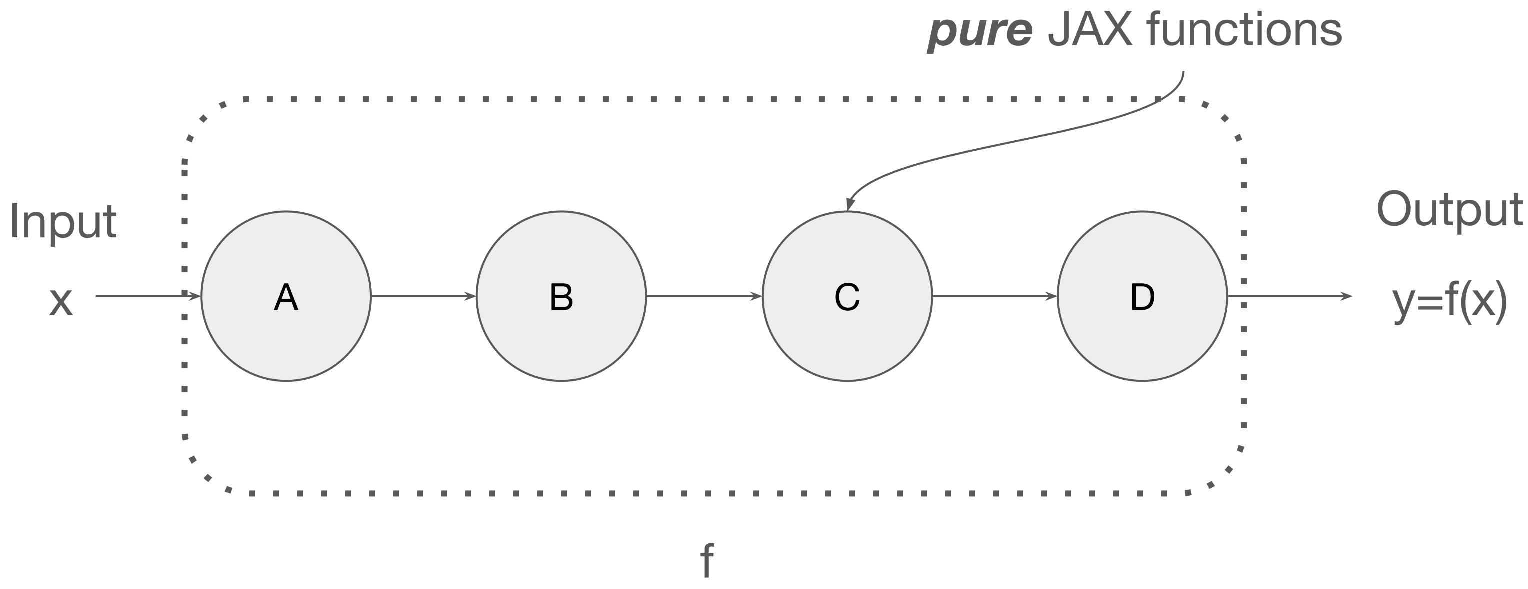
- 24-Core CPU: ~120s (40x)
- NVIDIA A100: ~10s (600x)

Galaxy: TNG50-1, Snapshot 99, Subhalo ID 15

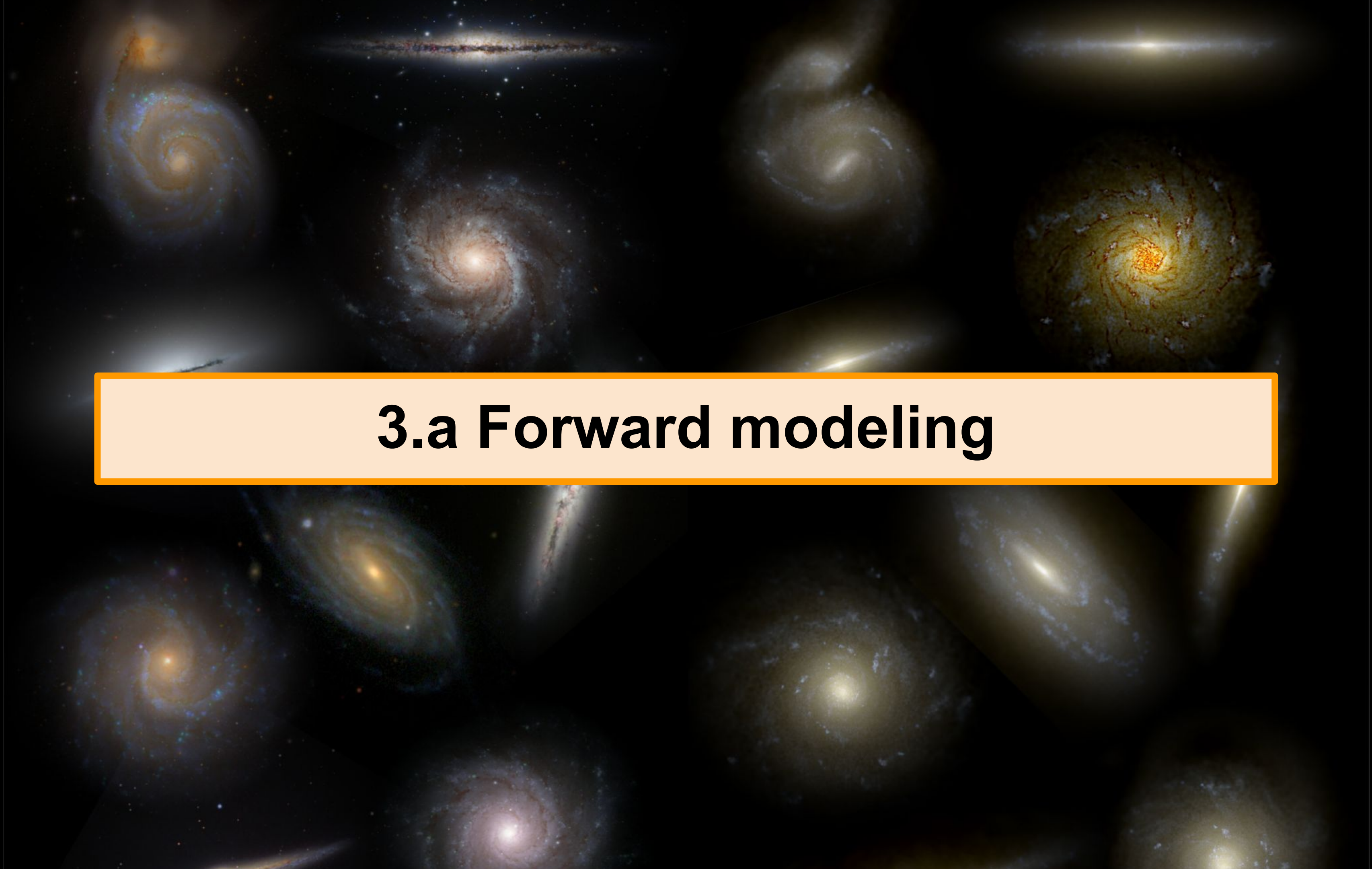


2. Code structure

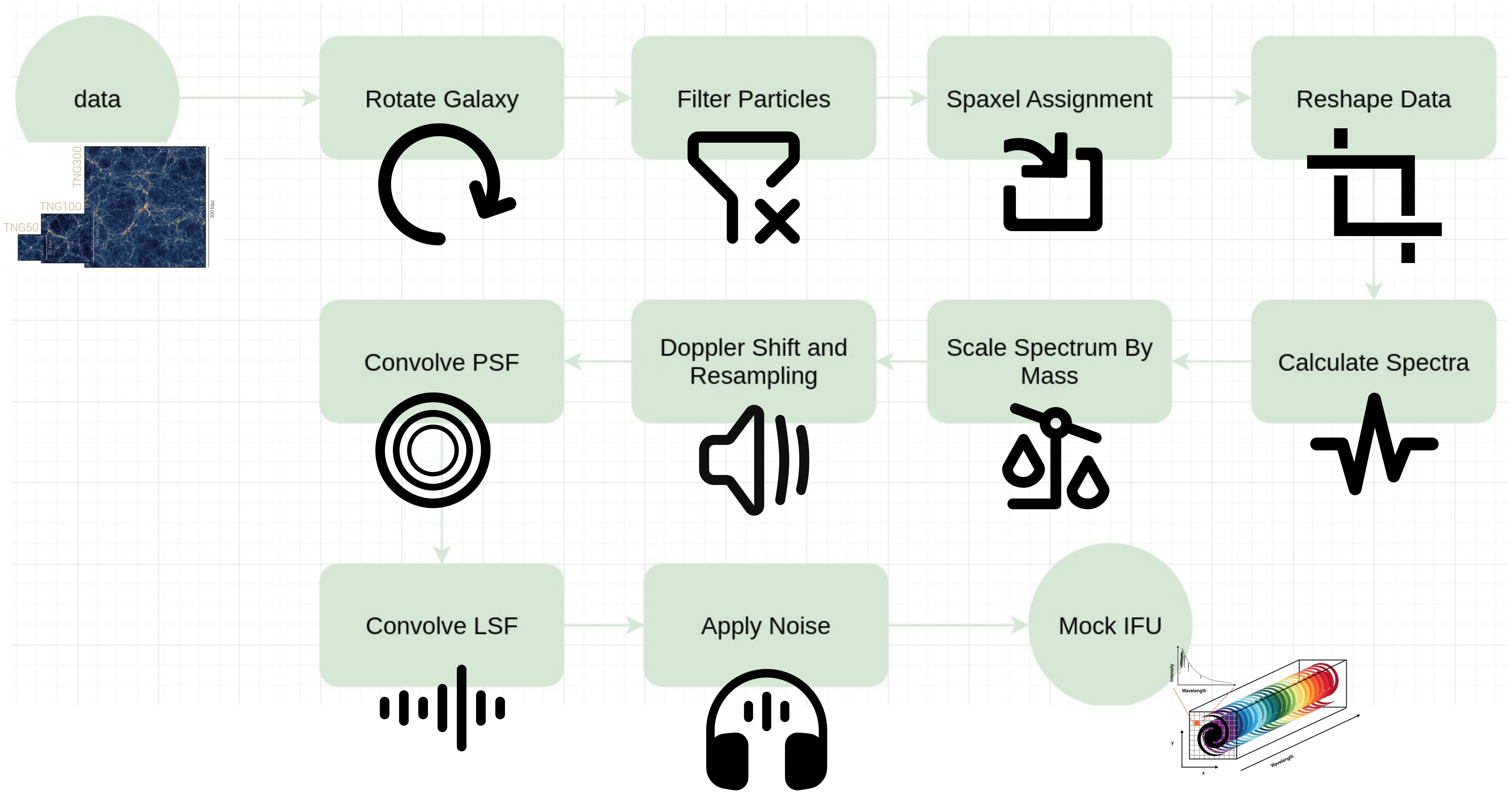
Code structure behind RUBIX



linear pipeline



3.a Forward modeling


data
TNG300
TNG100
TNG50
Rotate Galaxy
Filter Particles
Spaxel Assignment
Reshape Data
Calculate Spectra
Scale Spectrum By Mass
Doppler Shift and Resampling
Convolve PSF
Convolve LSF
Apply Noise
Mock IFU
Intensity
Wavelength
y
x
Wavelength

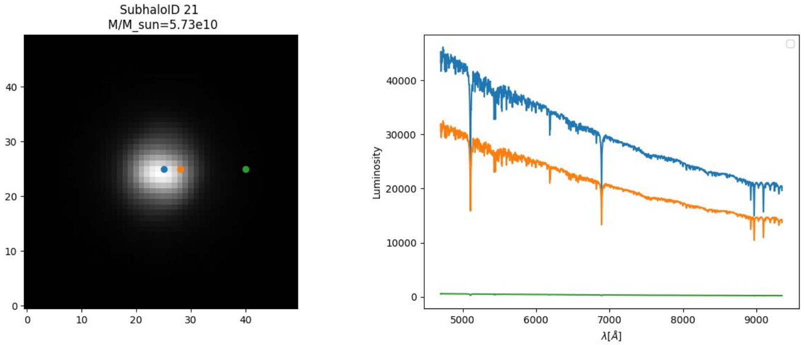
How to run the pipeline?

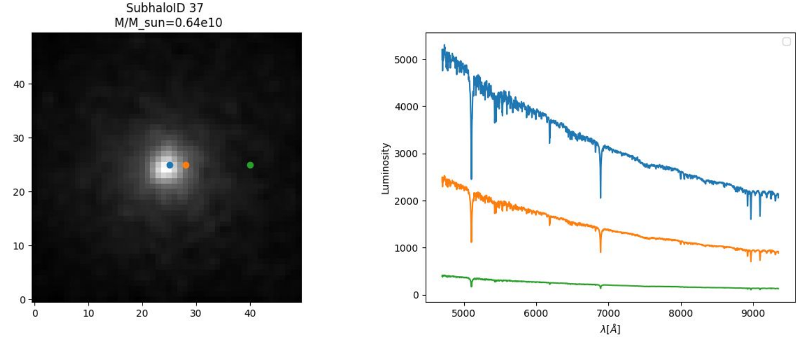
1. User configuration to specify e.g telescope, distance to galaxy, cosmology, SSP template
2. (Pipeline configuration)
3. Then simply run …

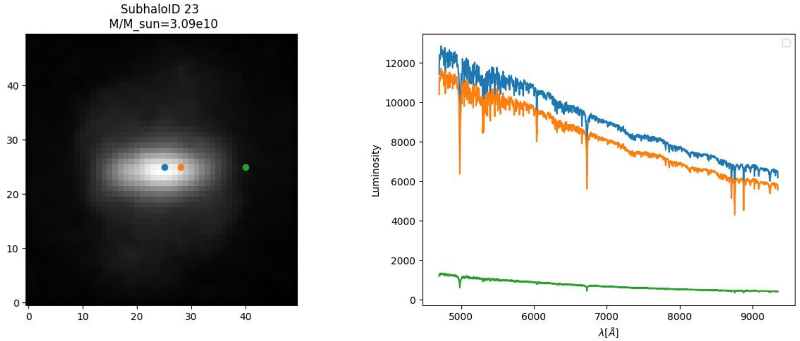
```python
pipe = RubixPipeline(config) # Setup
data= pipe.run() # Run forward model
```

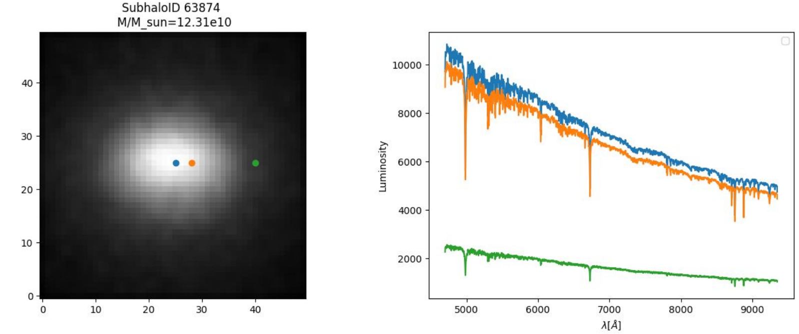
… and analyze the mock-data

```python
from rubix.core.pipeline import RubixPipeline
import os
config = {
    "pipeline":{"name": "calc_ifu"},

    "logger": {
        "log_level": "INFO",
        "log_file_path": None,
        "format": "%(asctime)s - %(name)s - %(levelname)s - %(message)s",
    },
    "data": {
        "output_path": os.path.join(os.getcwd(), "output"),
        "save_name": "tng_14",
        "subset": {
            "use_subset": True,
            "subset_size": 10,
        },
        "simulation": {
            "name": "IllustrisAPI",
            "args": {
                "api_key": os.environ.get("ILLUSTRIS_API_KEY"),
                "particle_type": ["stars"],
                "simulation": "TNG50-1",
                "snapshot": 99,
                "galaxy_id": 14,
                "reuse": True,
                }
                    },

    },
    "telescope":
        {"name": "MUSE",
         "psf": {"name": "gaussian", "size": 5, "sigma": 0.6},
         "lsf": {"sigma": 0.5},
         "noise": {"signal_to_noise": 1,"noise_distribution": "normal"},},
    "cosmology":
        {"name": "PLANCK15"},

    "galaxy":
        {"dist_z": 0.1,
         "rotation": {"type": "edge-on"},
        },

    "ssp": {
        "template": {
            "name": "BruzualCharlot2003"
        },
    },
```

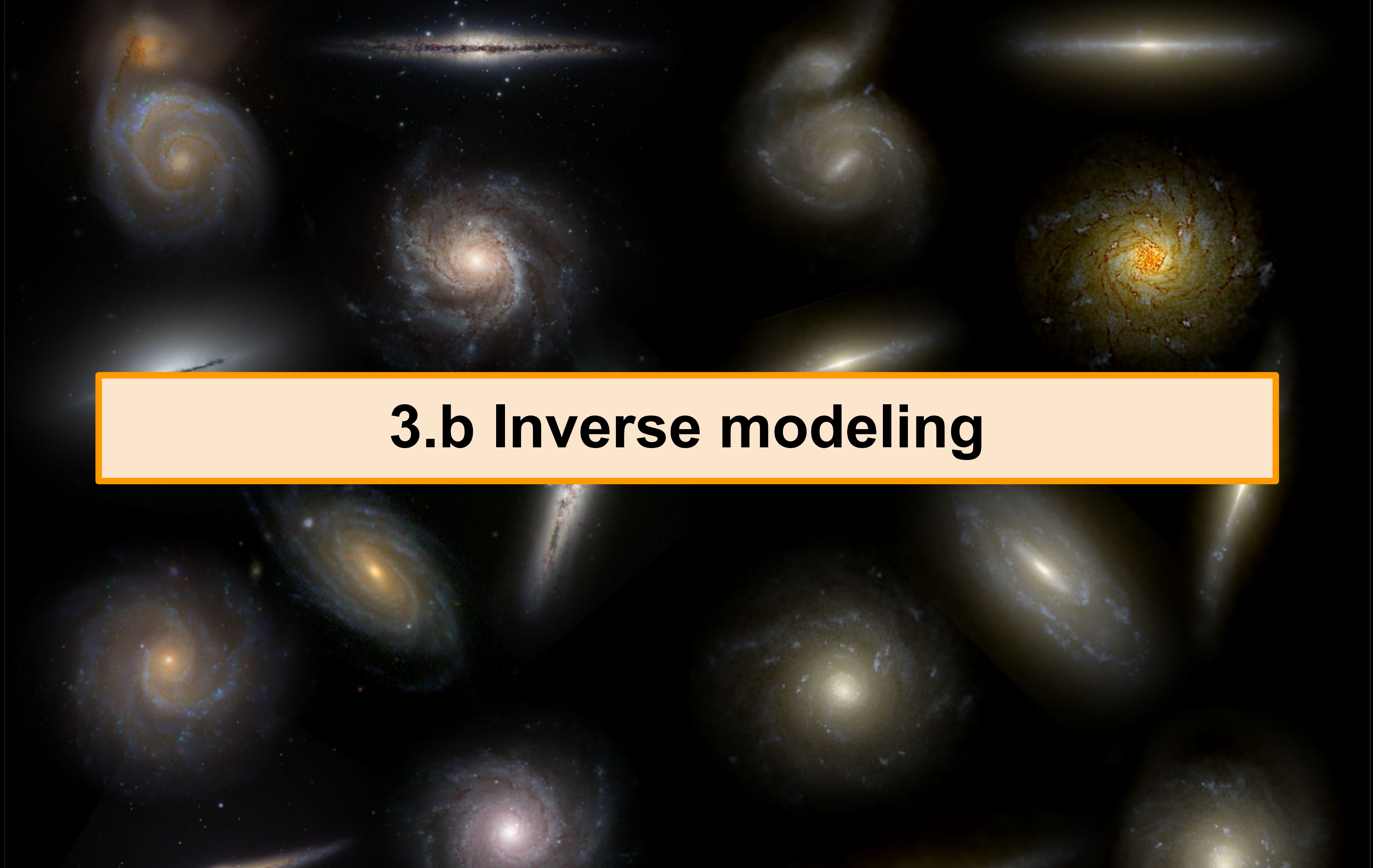








TNG50-1, Snapshot 99, SSP: Mastar_CB19_SLOG_1_5, z = 0.15



3.b Inverse modeling

Inverse modeling

Use JAX auto differentiation

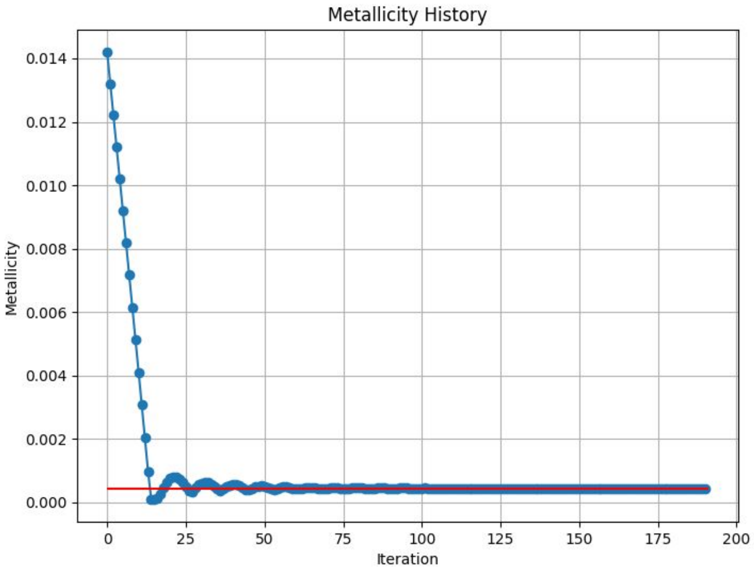
Toy example:

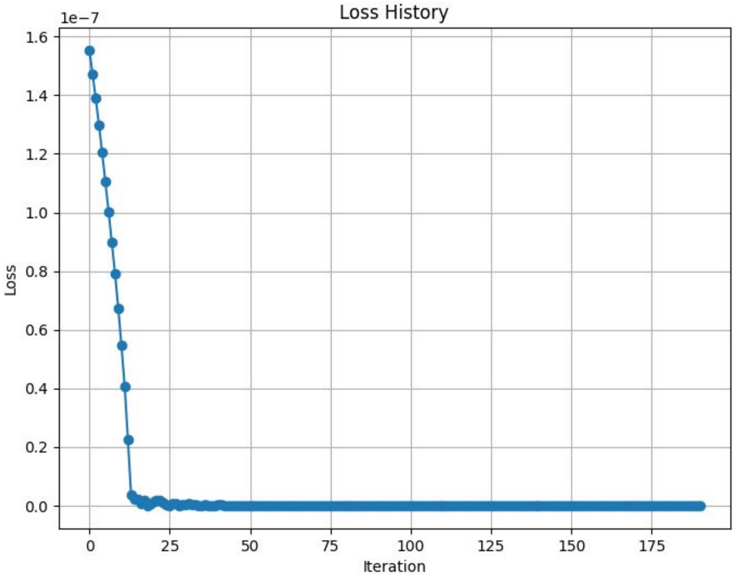
- one pixel
- one particle
- just vary the metallicity

Inverse modelling: Metallicity

At age 10 Gyr (100), start metallicity 1.4e-2 (9), target metallicity 4.5e-4 (3)

Learning rate 0.001, tol 1e-8

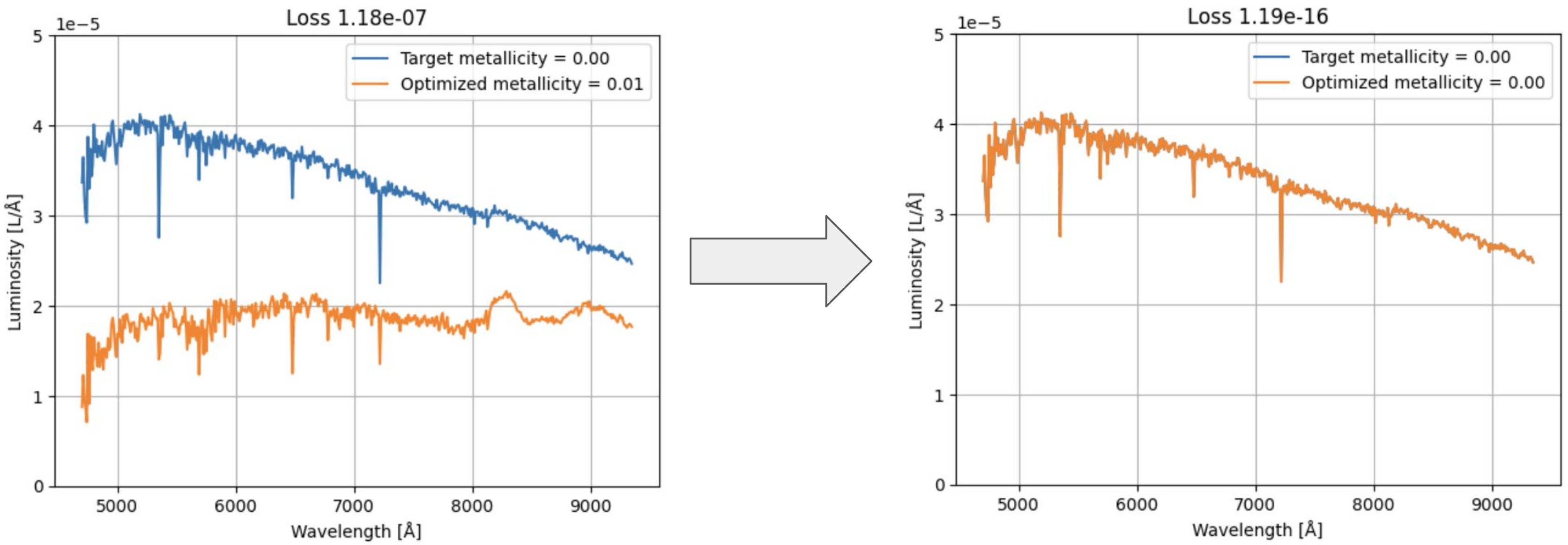


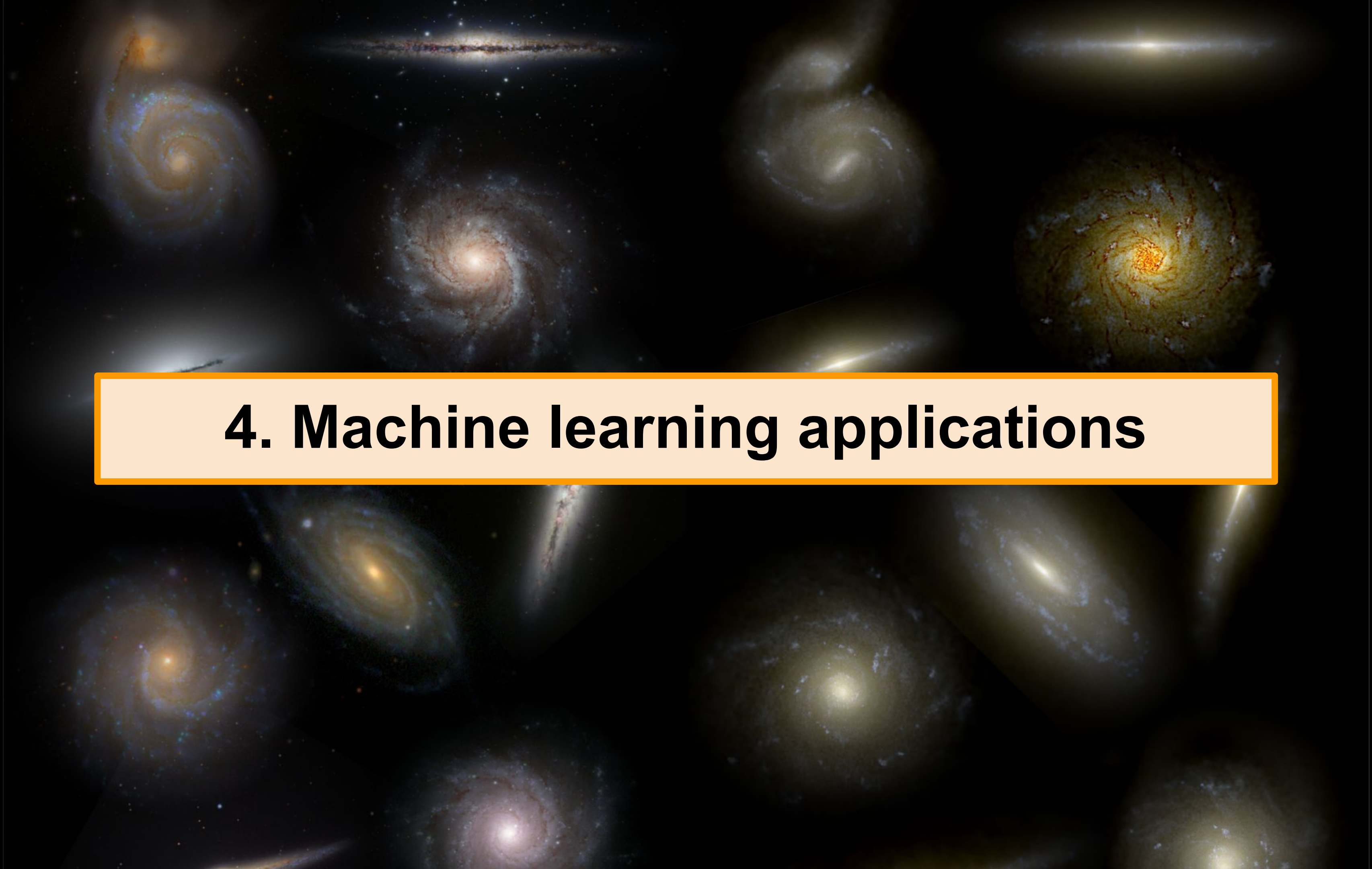


Inverse modelling: Metallicity

At age 10 Gyr (100), start metallicity 1.4e-2 (9), target metallicity 4.5e-4 (3)

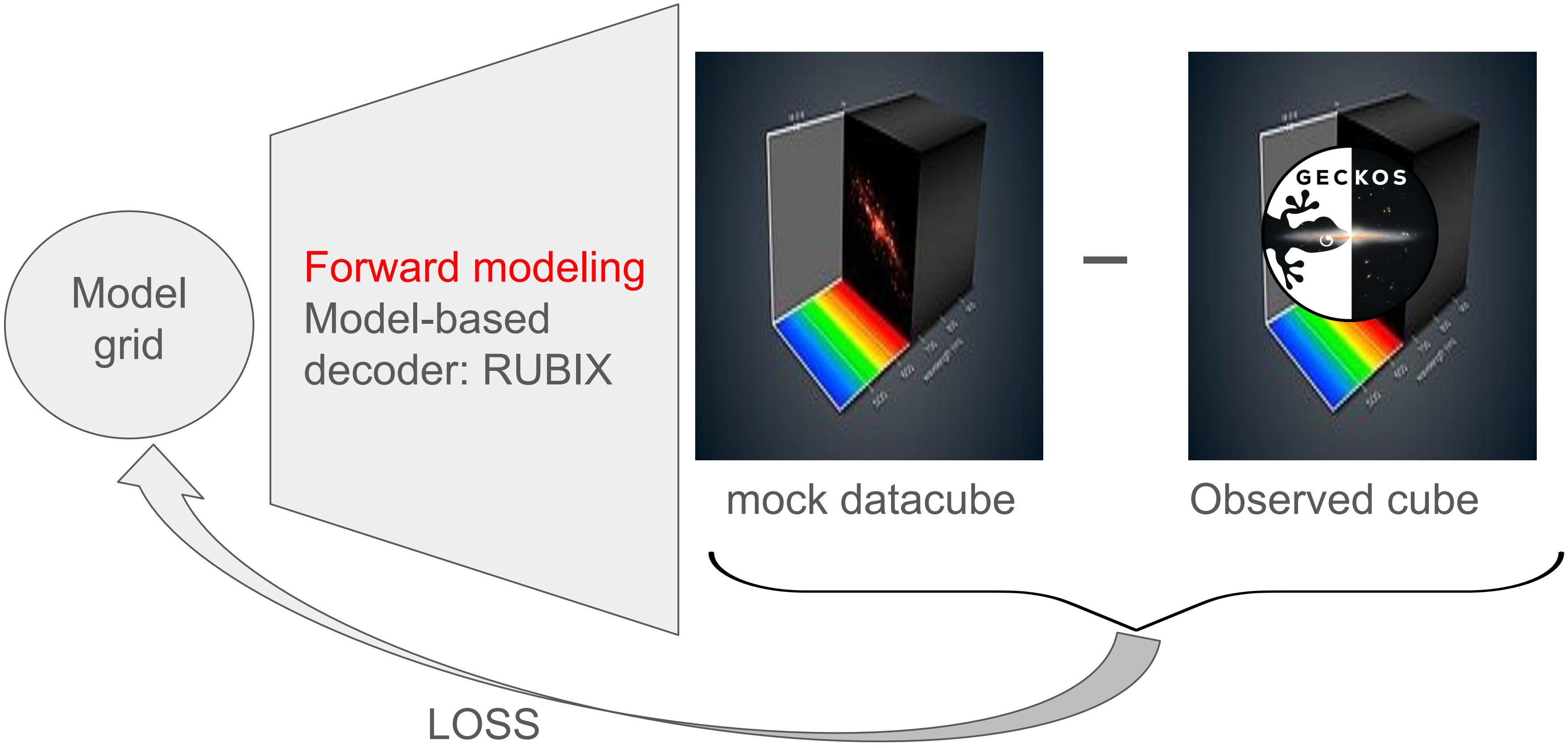
Learning rate 0.001, tol 1e-8



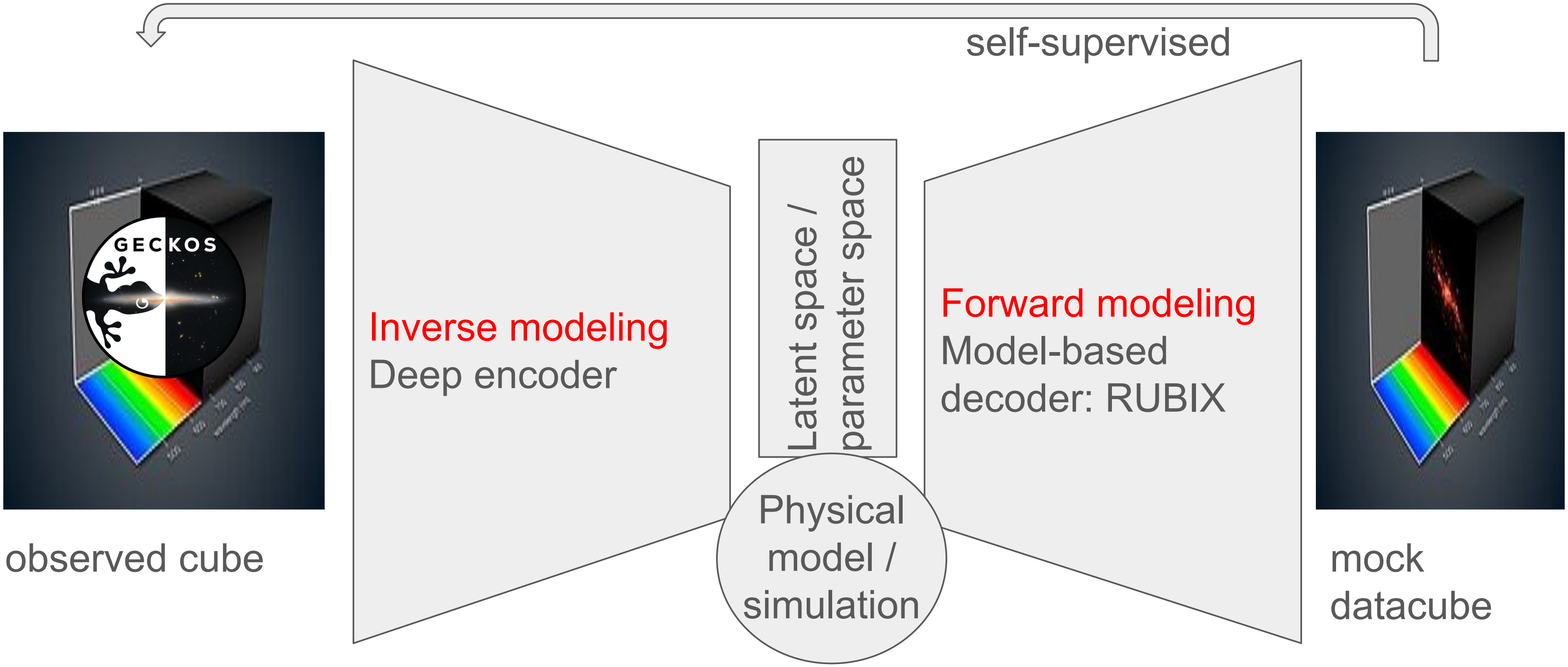


4. Machine learning applications

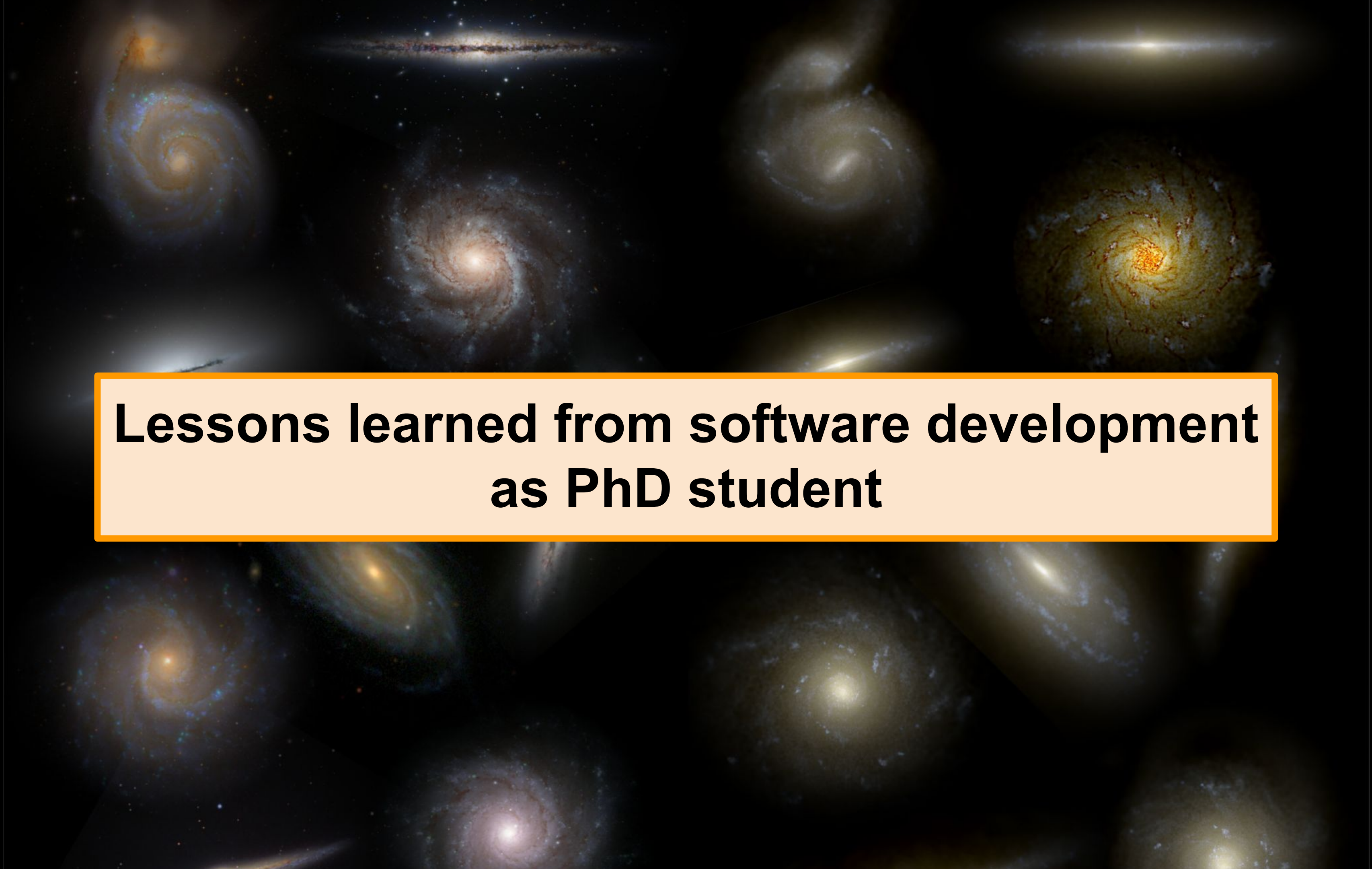
Gradient based optimization



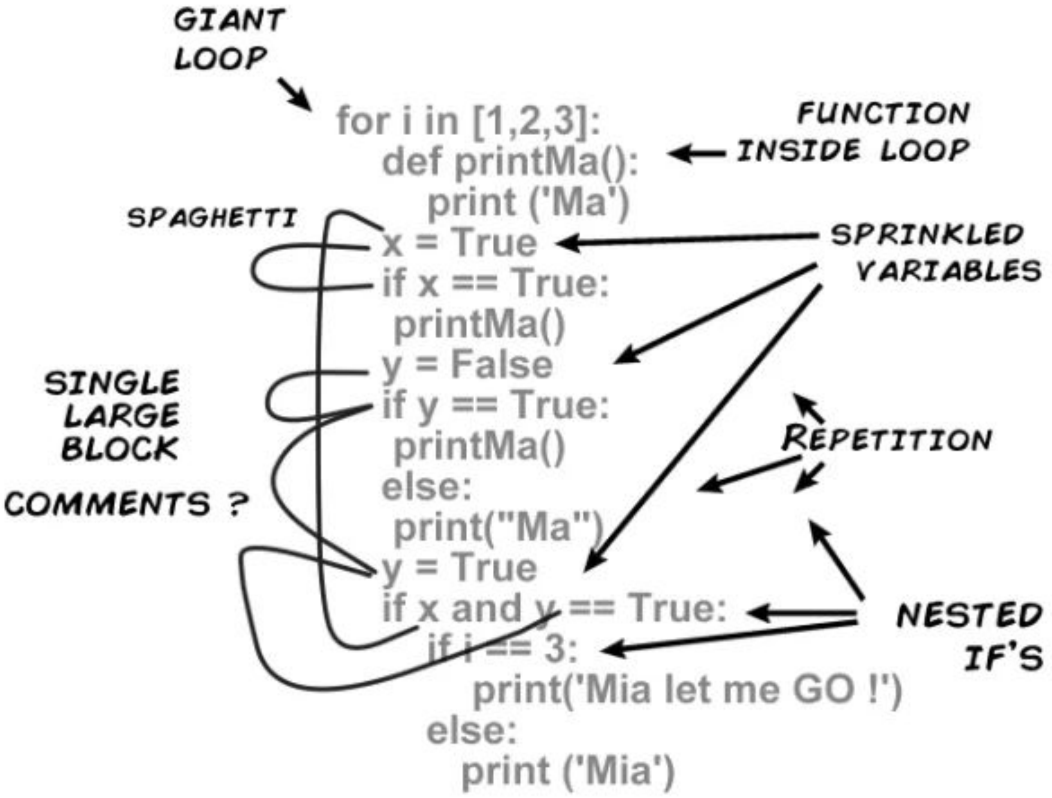
Self-supervised simulation based inference for GECKOS



Lessons learned from software development as PhD student



Spaghetti Code



SCIENTIFIC SOFTWARE CENTER

ssciwr / cookiecutter-python-package

Type / to search

Code | Issues 3 | Pull requests | Actions | Projects | Security | Insights

cookiecutter-python-package Public

Watch 2 | Star 11

main | 5 Branches | 1 Tag | Go to file t | Add file | Code

dokempf Merge pull request #41 from ssciwr/tokenless-codecov · e707747 · last month · 92 Commits

.github	Bump CI Python versions	last month
hooks	Tokenless Codecov authentication	last month
tests	Make the use of setuptools_scm optional	11 months ago

About

A cookiecutter for a Python project with lots of configuration options

python · documentation · continuous-integration · pypi · cookiecutter · hacktoberfest · iwr-hacktoberfest

README | MIT license | License

Welcome to Python Package Cookiecutter!

This repository is a template repository (a cookiecutter) that allows you to quickly set up new Python packages. It is geared towards scientific applications and implements the best practice guidelines of the [Scientific Software Center of Heidelberg University](#).

SSC PROGRAM “SSC FELLOWS”

The Scientific Software Center is establishing a new mentoring program “SSC Fellows”. This competitive program replaces the previous mentoring program “Reproducible Science”.

The program

SSC fellows receive individual training in state-of-the-art software engineering and tools for improving their research software development skills. Each fellow is assigned an RSE mentor from the SSC with whom they will meet on a monthly basis, to track and support their research software engineering project. Fellows will further engage in community events and educational efforts organized by the SSC. In addition, fellows will receive a travel stipend to participate in a national RSE conference, with the opportunity to engage in the research software engineering community, and present their contribution.

Eligibility and Selection Criteria

The SSC fellowship is limited to graduate students and postdoctoral researchers at Heidelberg University. Research software development should play a major role in the applicant’s PhD /postdoctoral project. The SSC fellowships are an equal opportunity program open to all qualified persons without regard to race, gender, religion, age, physical disability, or national origin. SSC fellows will be selected by the SSC Advisory Board based on (1) the quality of the applicant’s research software proposal and its relevance within the department/domain; (2) the applicant’s research productivity, including previous

https://www.ssc.uni-heidelberg.de/en/learning/ssc-program-ssc-fellows

Guidelines we try to follow for RUBIX

Python language

CPU and GPU compatible

Documentation

Profiling

Unit tests

Continuous integration

Design patterns

Type annotations

<u>Good Code Quality</u> (hopefully!)

Takes a lot of time, but may be worth in the end

Welcome to RUBIX's documentation!

RUBIX is a tested and modular Open Source tool developed in JAX, designed to forward model IFU cubes of galaxies from cosmological hydrodynamical simulations. The code automatically parallelizes computations across multiple GPUs, demonstrating performance improvements over state-of-the-art codes. For further details see the publications or the documentation of the individual functions.

Currently the following functionalities are provided:

- Generate mock IFU flux cubes for stars from IllustrisTNG50
- Generate mock photometric images for stars for different filter curves
- Use different stellar population synthesis models
- Use MUSE as telescope instrument (and some other instruments)

Currently the code is under development and is not yet all functionality is available. We are working on adding more features and improving the code, espectially we work on the following features:

- Adding support for more simulations
- Adding support for more telescopes
- Adding gas emission lines and gas continuum
- Adding dust attenuation
- Adding support for gradient calculation

If you are interested in contributing to the code or have ideas for further features, please contact us via a github issue or via email. If you use the code in your research, please cite the following paper: Publications

Contents

Guidelines we try to follow for RUBIX

Python language

CPU and GPU compatible

Documentation

Profiling

Unit tests

Continuous integration

Design patterns

Type annotations

<u>Good Code Quality</u> (hopefully!)

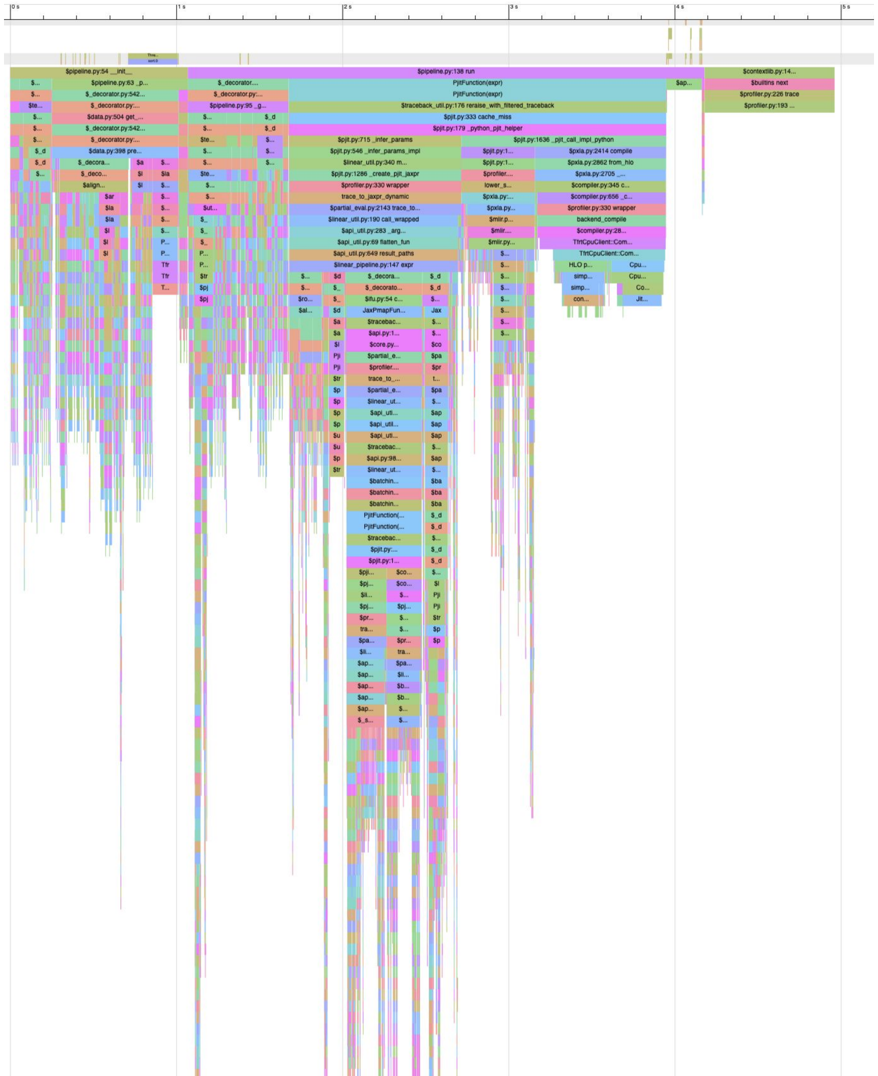
Takes a lot of time, but may be worth in the end

Pipeline run for 1000 stellar particles

Pipeline time 3.09 sec

Data preparation in the beginning 1 sec

Calculate spectra function 463 millisec



Guidelines we try to follow for RUBIX

- Python language
- CPU and GPU compatible
- Documentation
- Profiling
- Unit tests
- Continuous integration
- Design patterns
- Type annotations

Good Code Quality (hopefully!)

Takes a lot of time, but may be worth in the end

Open **61 simple dust attenuation** #134

TobiBu wants to merge 10 commits into `main` from `61-simple-dust-attenuat…`

Review required

At least 2 approving reviews are required by reviewers with write access.

1 pending review

All checks have passed

5 successful checks

CI / Coverage Testing (pull_request)	Successful in 1m	
CI / Testing on macos-latest (pull_request)	Successful in 2m	Required
CI / Testing on macos-latest (pull_request)	Successful in 2m	Required
CI / Testing on ubuntu-latest (pull_request)	Successful in 1m	Required
CI / Testing on ubuntu-latest (pull_request)	Successful in 2m	Required

Merging is blocked

At least 2 approving reviews are required by reviewers with write access.

Merge pull request

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

Preview Switch back to the classic merge experience · Give feedback

Guidelines we try to follow for RUBIX

- Python language
- CPU and GPU compatible
- Documentation
- Profiling
- Unit tests
- Continuous integration
- Design patterns
- Type annotations

<u>Good Code Quality</u> (hopefully!)

Takes a lot of time, but may be worth in the end

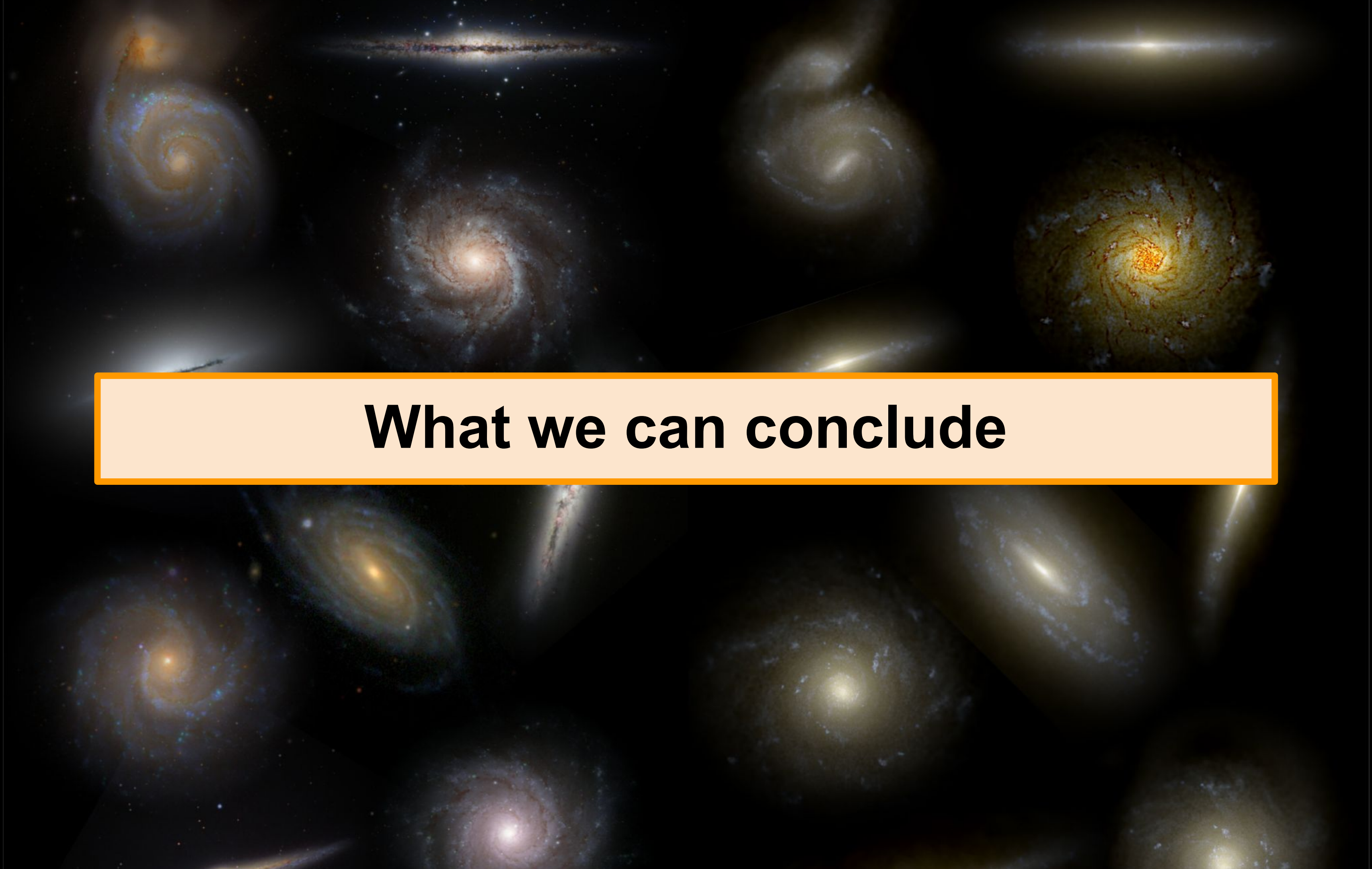
```python
from typing import List, Optional, Union
from jaxtyping import Int, Float, Array, jaxtyped
from beartype import beartype as typechecker
import numpy as np

import equinox as eqx


@jaxtyped(typechecker=typechecker)
class BaseTelescope(eqx.Module):
    """
    Base class for the telescope module.
    This class contains the base parameters for the telescope module.

    Args:
        fov (float): The field of view of the telescope.
        spatial_res (float): The spatial resolution of the telescope.
        wave_range (list): The wavelength range of the telescope.
        wave_res (float): The wavelength resolution of the telescope.
        lsf_fwhm (float): The full width at half maximum of the line spread function.
        signal_to_noise (float): The signal to noise ratio of the telescope.
        sbin (int): The size of the spatial bin in each direction for the aperture mask.
        aperture_region (jnp.ndarray): The aperture region of the telescope.
        pixel_type (str): The type of pixel used in the telescope.
        wave_seq (jnp.ndarray): The wavelength sequence of the telescope.
        wave_edges (jnp.ndarray): The wavelength edges of the telescope.
    """

    fov: Union[float, int]
    spatial_res: Union[float, int]
    wave_range: List[float]  # upper and lower limits
    wave_res: Union[float, int]
    lsf_fwhm: Union[float, int]
    signal_to_noise: Optional[float]
    sbin: np.int64
    aperture_region: Union[Float[Array, "..."], Int[Array, "..."]]
    pixel_type: str
    wave_seq: Float[Array, "..."]
    wave_edges: Float[Array, "..."]
```

Benefits of good code quality

<u>Good Code Quality</u> (hopefully!)

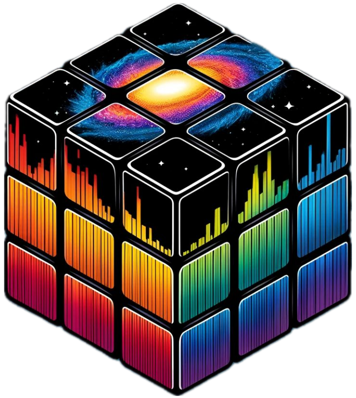
- Readable
- Extensible
- Modular
- Configurable
- Reproducible
- Flexible
- Large user base
- Easy to use
- Community development

What we can conclude



RUBIX

- Forward and inverse modeling of simulated galaxies and IFU data
- With help of the SSC Heidelberg: project developed to a good codebase
- Lots of effort into testing, integration, documentation
- Hopefully become a broadly used community tool



https://astro-rubix.web.app/


Thank you!