Freie Universität Berlin

PyEMMA Package Overview and Software Development

Martin K. Scherer
Free University Berlin

February 17, 2019

Outline

Software overview and design patterns

- Python
- Anaconda stack
- Package overview
  - Coordinates package
  - MSM package

PyEMMA Development

- Principles
- Processes
  - GitHub
  - Continous Integration Services
- Collaboration

Outline

- Software overview and design patterns
  - Python
  - Anaconda stack
  - Package overview
    - Coordinates package
    - MSM package
- PyEMMA Development
  - Principles
  - Processes
    - GitHub
    - Continous Integration Services
  - Collaboration

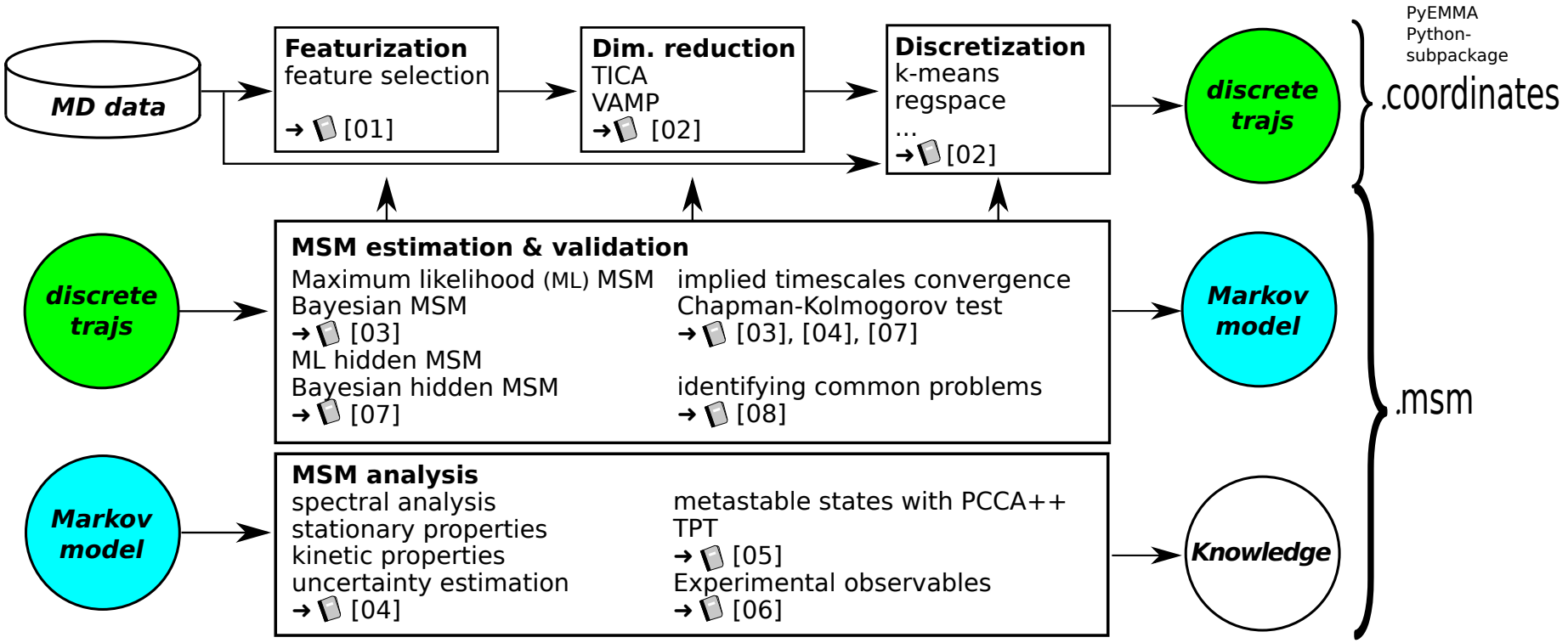
Python in Data Science

- Easy to use core libraries (eg. NumPy, SciPy, Pandas, Jupyter, Matplotlib, . . .)
- Scientific software for MD, data science, biology, chemistry . . .
- Easy to learn general purpose language
- Quick prototyping
- Glue together software written in faster languages (eg. C/C++, Fortran)

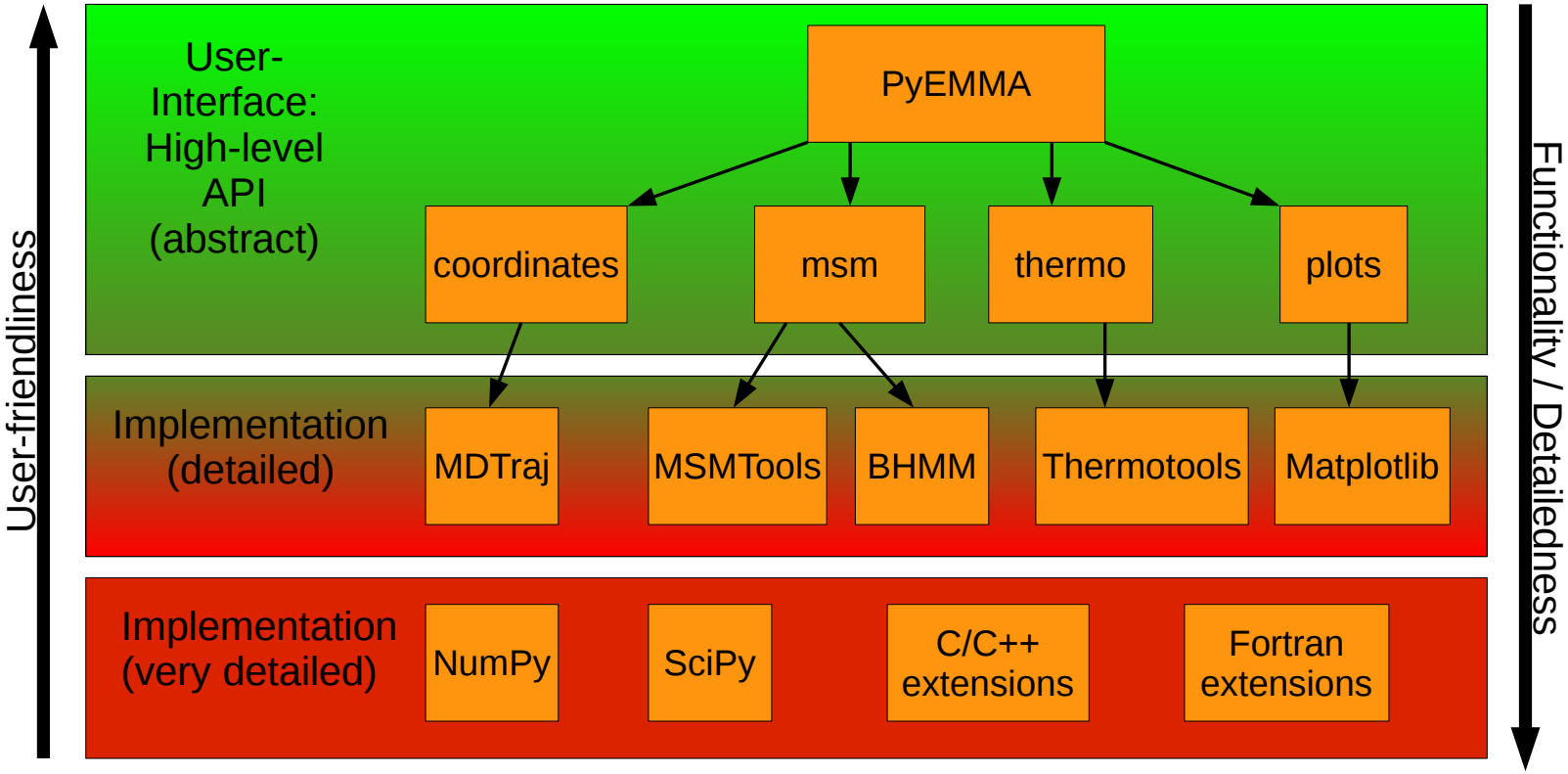
Anaconda Cloud and Conda package manager

- Anaconda is a (Python-based) software stack built for all three major platforms (Linux, OSX, Windows)
- Easy installation and upgrading, no need to compile anything yourself.
- Different software channels for different purposes (eg. Omnia [MD], BioConda [Bioinformatics], . . .)
- Automatic handling of dependencies (conflict checking)
- Possibility to create isolated work environments (separate package versions etc.)

From MD data to Knowledge



Package hierarchy - abstracting detailedness



- Streaming data pattern
- Avoid the need of dumping intermediate results to disk
- Support for multiple data formats
- Random access possible (either simulated or IO efficient)

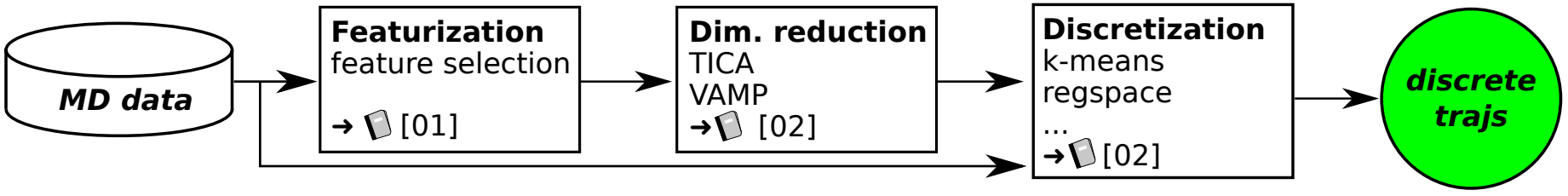


Figure: Workflow: state space discretisation

- All readers are Python-“iterable”, which means you can process data in chunks. The more general concept in PyEMMA is called ‘DataSource‘.

```
1 my_source = pyemma.coordinates.source(['traj001.xtc', ...)
2 for element in my_source:
3     print(element)
```

Supported reader data formats:

- MD-simulation data (XTC, DCD, . . . via MDTraj)
- NumPy (.npy) files
- Tabulated ASCII data (around three times more efficient than Numpy.loadtxt)
- Fragmented trajectories [(’sim_0_part0.xtc’, ’sim_0_part1.xtc’), ’sim_1_part0.xtc’, ’sim_1_part1.xtc’)]

Python package for reading/writing and analyzing molecular trajectories.
Analysis functions:

- distances
- bonds/angles/dihedrals
- hydrogen bonding identification
- secondary structure assignment
- NMR observables
- . . . and many more

Supported formats:

- DCD
- XTC
- TRR
- PDB
- XYZ
- binpos
- NetCDF
- LH5
- HDF5
- . . .

MSM package

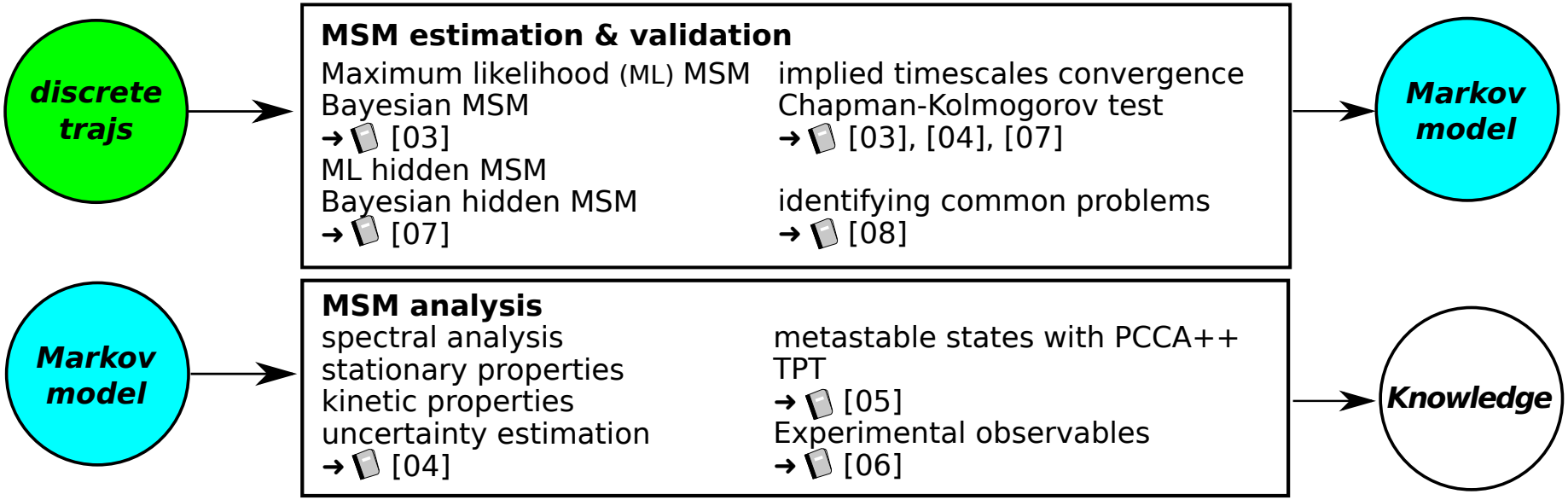


Figure: MSM estimation and analysis workflow.

MSM package User-API examples

Step	Goal	API function (all in pyemma.msm package)
1.a	choose lag time	its = timescales_msm(dtrajs)
1.b	choose lag time (visual inspection)	pyemma.plots. plot_implied_timescales(its)
2	estimate a model	msm_obj = estimate_markov_model(dtrajs, lag)
3.a	validate model	ck_obj = msm_obj.cktest()
3.b	validate model (vis. inspection)	pyemma.plots.plot_cktest(ck_obj)
4.a	Analyze slow processes	msm_obj.timescales() etc.
4.b	Perform coarse graining	coarsed = msm_obj.pcca()
4.c	Transition path analysis	coarsed.tpt()

Outline

- Software overview and design patterns
  - Python
  - Anaconda stack
  - Package overview
    - Coordinates package
    - MSM package
- PyEMMA Development
  - Principles
  - Processes
    - GitHub
    - Continous Integration Services
  - Collaboration

Principles

- Use Python as the glue to faster languages (C/C++, Fortran)
- Stable and easy to use high level user interface
- Open source (GNU Lesser Public license 3+, minimal restrictions on redistribution)
- Open development process on GitHub (everybody can contribute)
- Focus on speed and stability (NumPy, SciPy under the hood)
- Focus on good documentation (see http://emma-project.org)

LGPL Free Software
Free as in Freedom

github
SOCIAL CODING

NumPy
SciPy.org
Sponsored By ENTHOUGHT
ANACONDA CLOUD

Development processes

- GitHub as frontend (collect issues/bugs, discuss proposed changes, plan new features, . . .)
- Continuous integration/deployment (Travis-CI, AppVeyor, custom Jenkins instances)
- Unit-tests for API and implementation
- Integration tests of notebooks
- Release bug fixes regularly
- Release major/minor versions, if API changes.
- Preserve API compatibility (deprecate functions first, to notice users, that in the future their program/scripts will not work the same way as before)

Releasing and deploying

- Before a release we freeze acceptance of new features (their milestone gets postponed to the next release)
- Testing sessions - eliminate all found bugs
- Deploy source archive to PyPI (installable with pip) and binaries to Anaconda.org binary services.
- Version scheme:
  Major.minor.micro

major = major new (and API break features)
minor = new features preserving existing API
micro = patches/bug fixes

GitHub

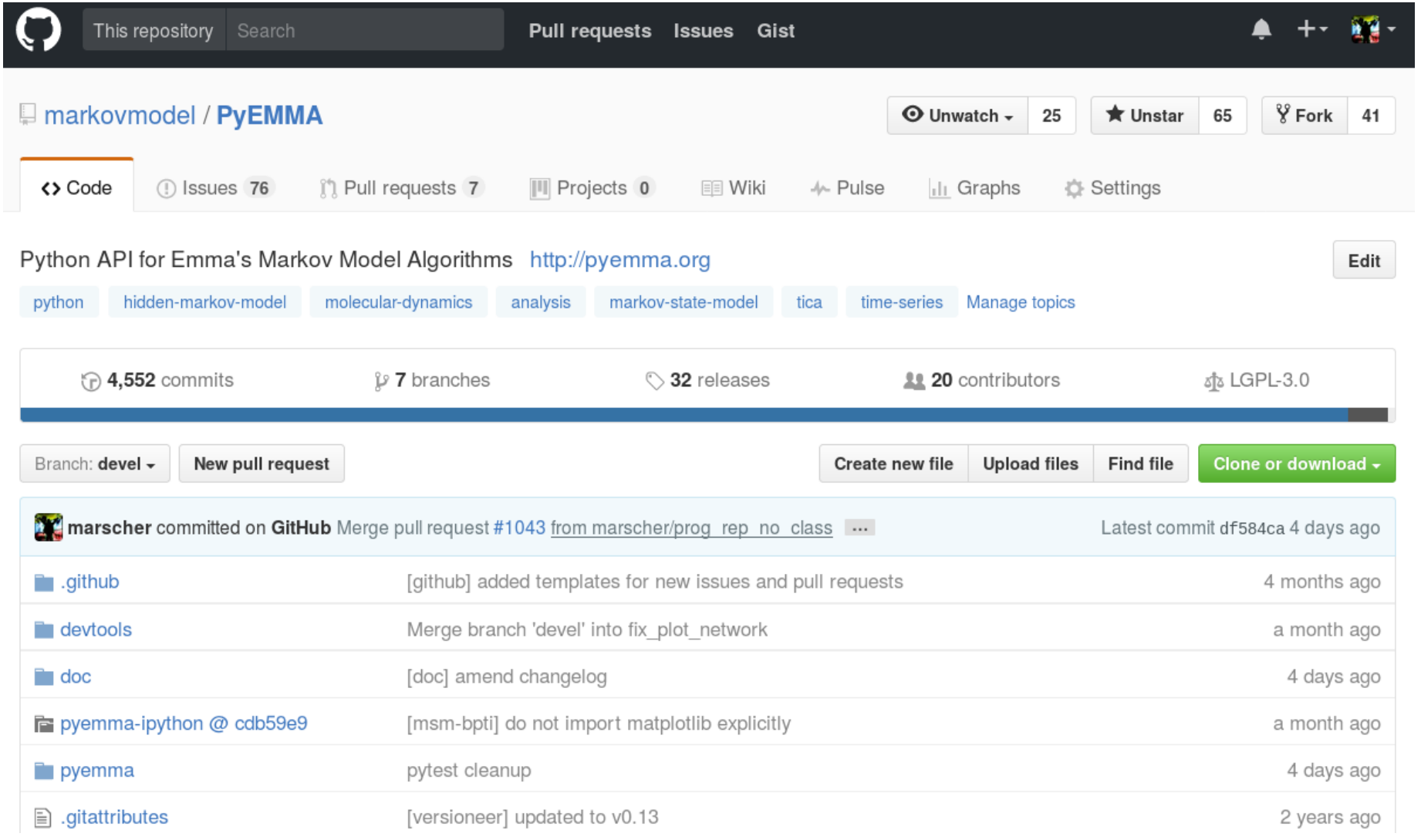
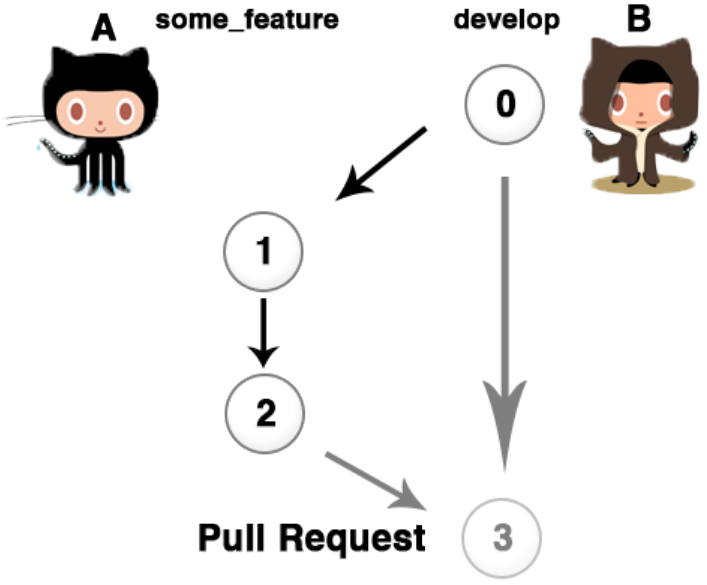


Figure: PyEMMA GitHub page

Collaboration on GitHub

1. Propose a change/feature via an issue
2. Create a local branch in Git to work on
3. Push the (tested) branch to your fork
4. Open a “pull request” (PR) on main repository (markovmodel/PyEMMA)
5. Discuss changes, eventually add more commits
6. Maintainer merges your PR



Propose file change on GitHub

...continued

Participate

- Create a GitHub account to directly post issues (**preferred**).
- Join our channel on Gitter.im
- Send mails to the developers (more overhead for us, might not reach somebody in time).

Thank you for your attention!
Further questions?