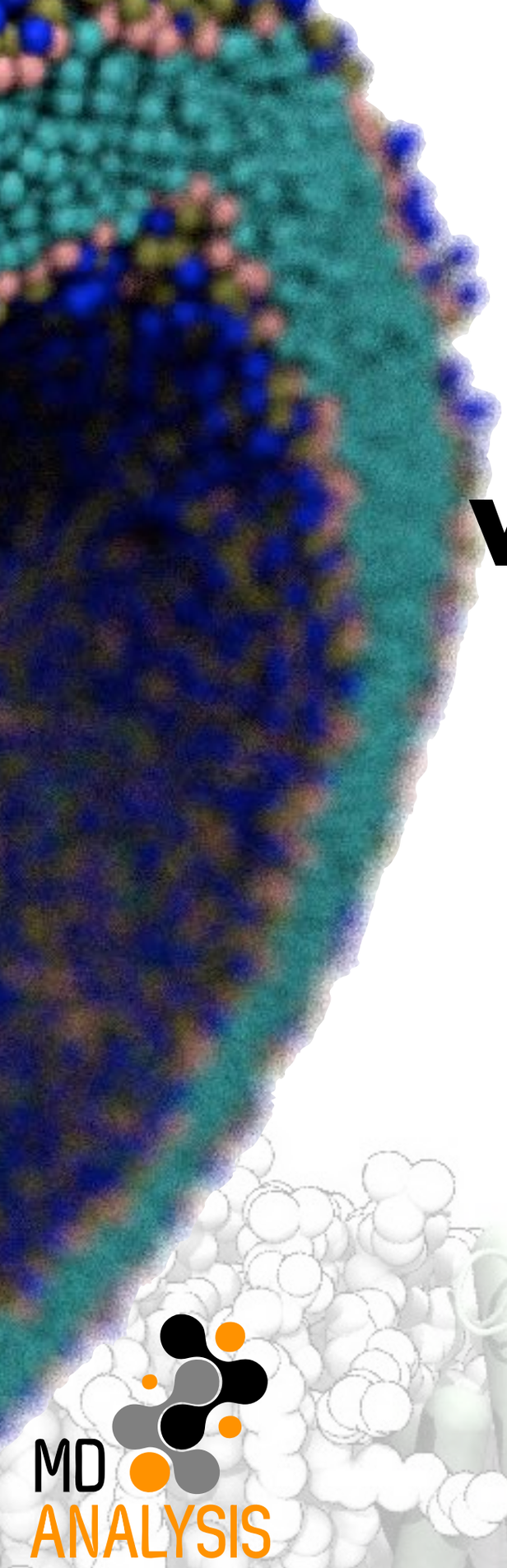


Integration via Python: building blocks for simulation and analysis workflows for molecular dynamics

Sustainable Software for Computational Molecular Science Symposium
ACS Spring Meeting
March 31, 2019

Oliver Beckstein
Arizona State University

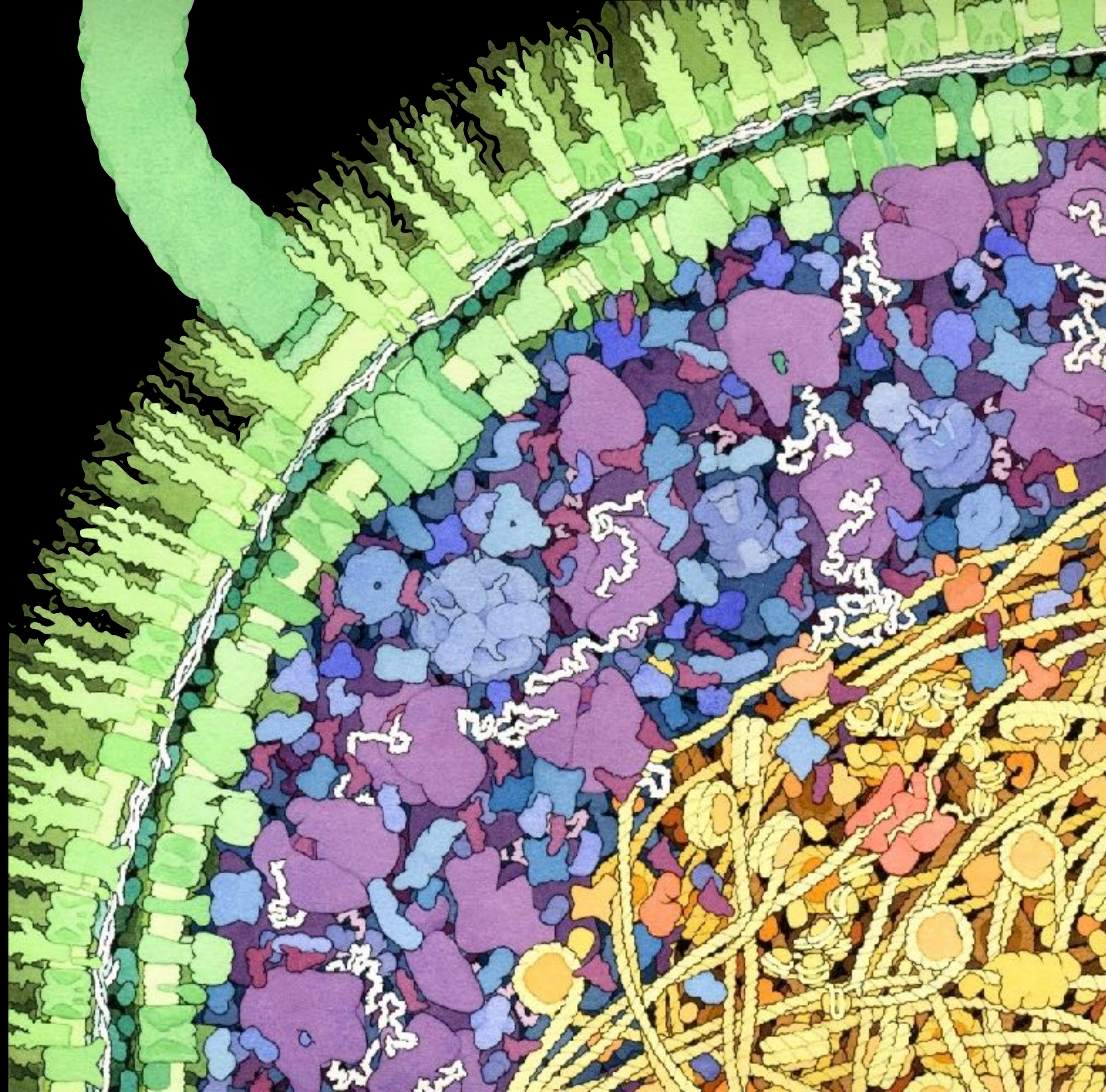
MD ANALYSIS

LIFE

cells

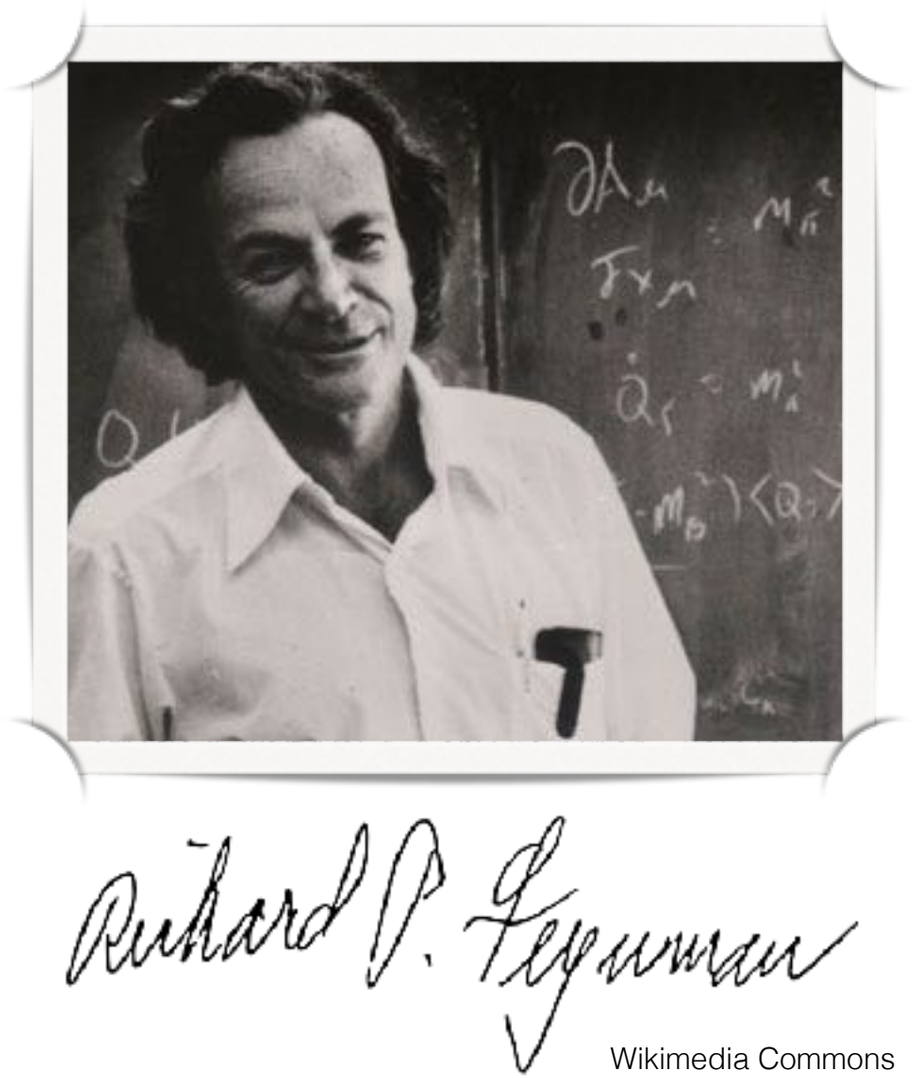
molecules

atoms



Escherichia coli (© 1999 David S. Goodsell, the Scripps Research Institute)

"Everything that living things do can be understood in terms of the jiggling and wiggling of atoms." —
Richard Feynman*



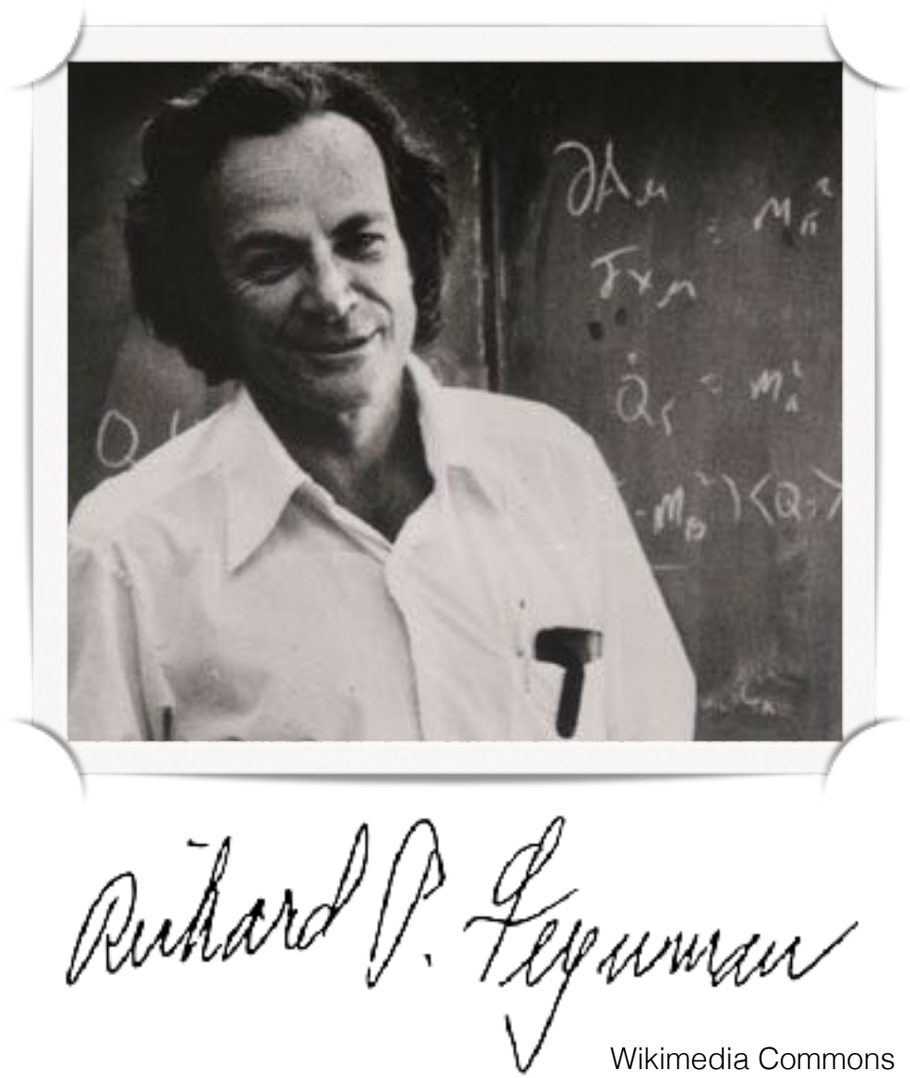
Wikimedia Commons

* R.P. Feynman. *The Feynman Lectures on Physics*. 1963

"Everything that living things do can be understood in terms of the jiggling and wiggling of atoms." —
Richard Feynman*

$$(\mathbf{r}_1(t), \ldots, \mathbf{r}_N(t))$$

positions of all *N* atoms over time

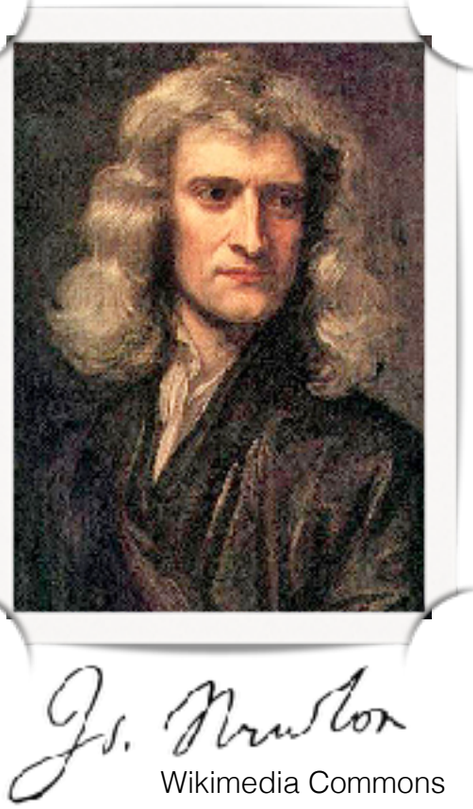


Wikimedia Commons

* R.P. Feynman. *The Feynman Lectures on Physics*. 1963

$$\mathbf{F}_i = m_i \mathbf{a}_i$$

$$= m_i \frac{d^2 \mathbf{r}_i}{dt^2}$$



Wikimedia Commons

trajectory

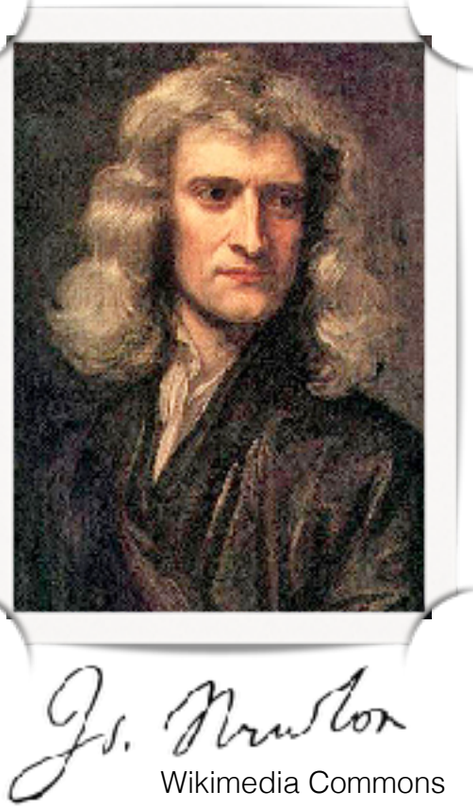
$$\left(\mathbf{r}_1(t), \ldots, \mathbf{r}_N(t)\right)$$

$$0 \le t \le \tau$$

MD simulation of intestinal fatty acid binding protein. Rendered with VMD.

$$\mathbf{F}_i = m_i \mathbf{a}_i$$

$$= m_i \frac{d^2 \mathbf{r}_i}{dt^2}$$



Wikimedia Commons

trajectory

$$\left(\mathbf{r}_1(t), \ldots, \mathbf{r}_N(t)\right)$$

$$0 \le t \le \tau$$

MD simulation of intestinal fatty acid binding protein. Rendered with VMD.

Computing observables from trajectories

- average over *estimator* function to get **observable** (phase space/"ensemble" **average**)

$$Z = \int d\mathbf{x}_1 \dots d\mathbf{x}_N e^{-U(\mathbf{x}_1,\dots,\mathbf{x}_N)/kT}$$

$$p(\mathbf{x}_1,\dots,\mathbf{x}_N) = e^{-U(\mathbf{x}_1,\dots,\mathbf{x}_N)/kT} / Z$$

$$A = \langle \mathcal{A} \rangle = \int d\mathbf{x}_1 \dots d\mathbf{x}_N p(\mathbf{x}_1,\dots,\mathbf{x}_N)\mathcal{A}(\mathbf{x}_1,\dots,\mathbf{x}_N)$$

- ergodic hypothesis: time average = ensemble average

$$\overline{\mathcal{A}(x_t)} = \lim_{\tau\to\infty} \frac{1}{\tau}\int_0^\tau dt \mathcal{A}(x_t) \approx \frac{1}{N}\sum_{t=0}^{N} \mathcal{A}(x_t)$$

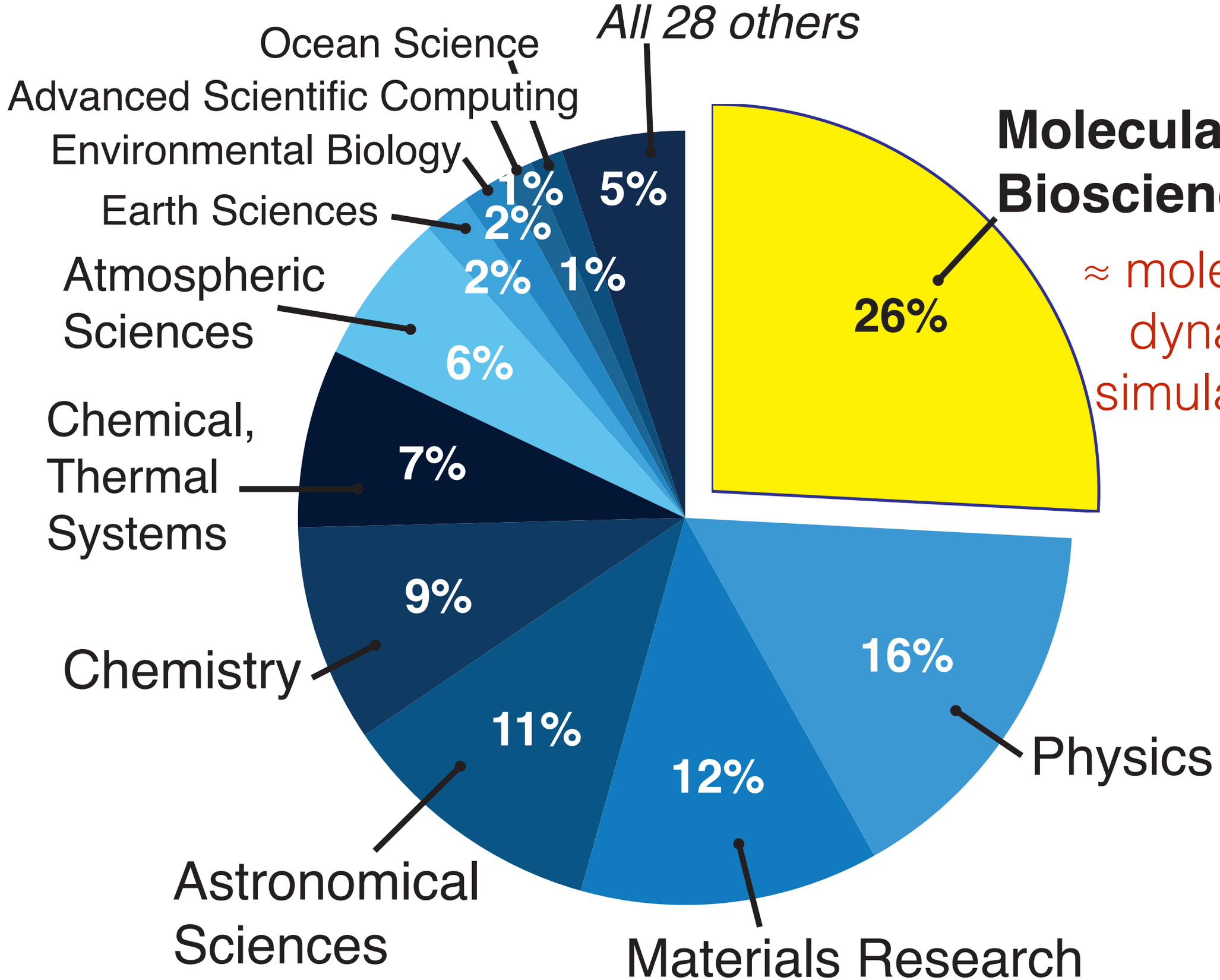
$$\langle \mathcal{A}(x) \rangle = \overline{\mathcal{A}(x_t)}$$

- measure *probabilities* to obtain **free energy differences**

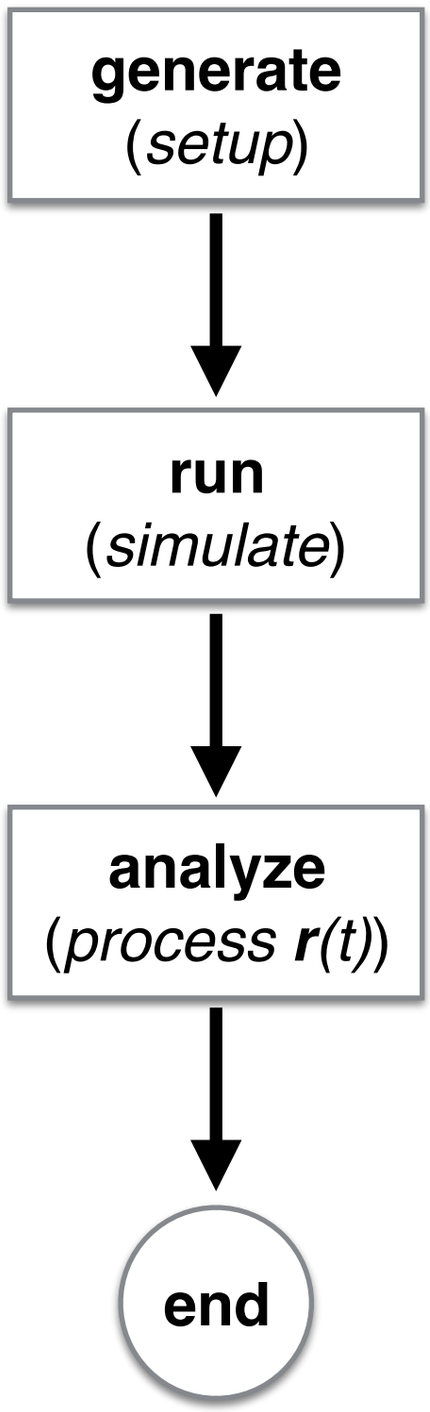
$$F = -kT \ln Z$$

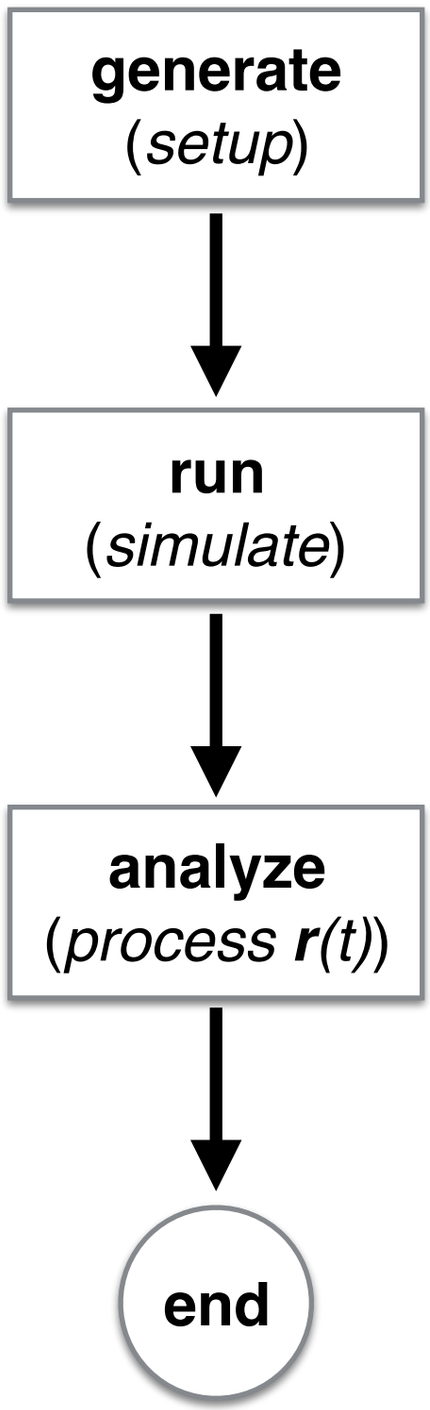
$$\Delta F = F_2 - F_1 = -kT \ln \frac{Z_1/Z}{Z_2/Z} = -kT \ln \frac{\mathcal{P}_1}{\mathcal{P}_2}$$

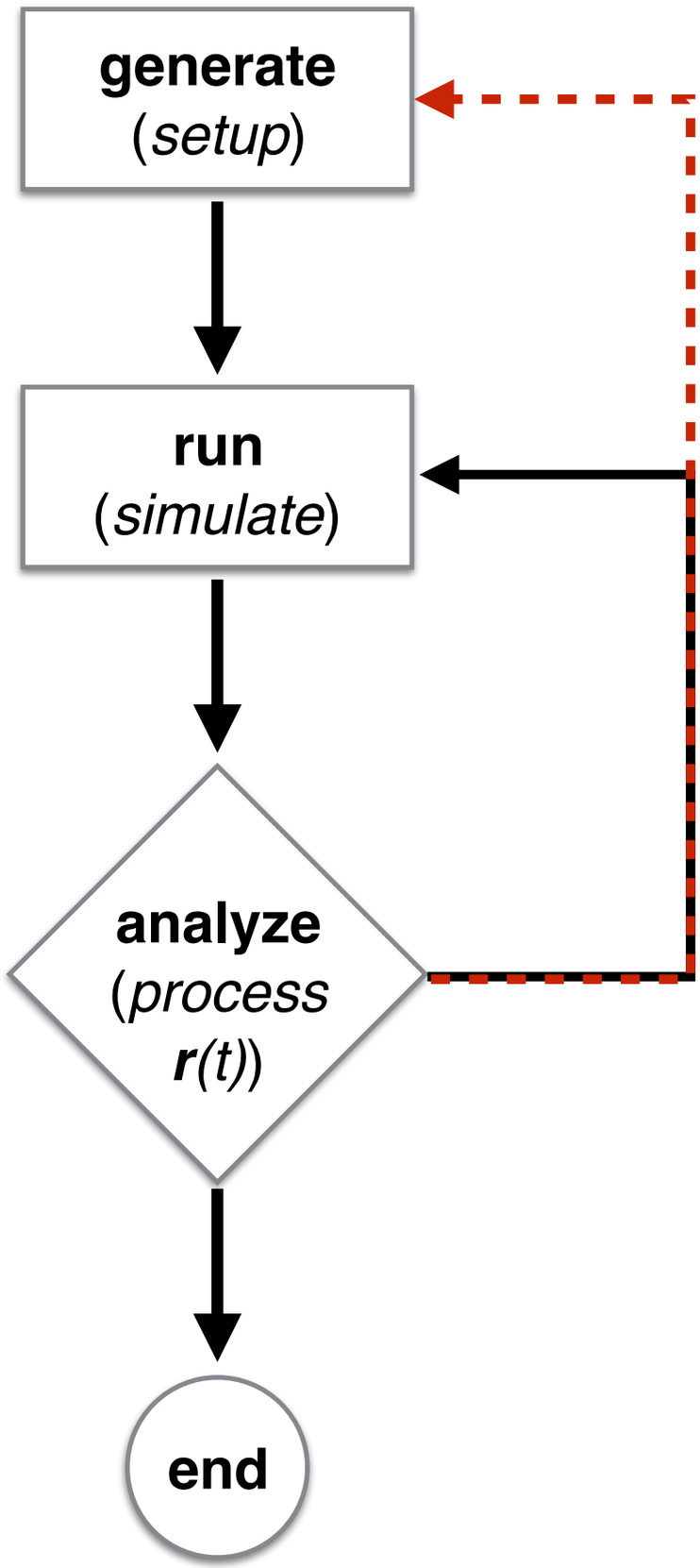
XSEDE SUs 2011–2017



source: xsede.org


generate
(*setup*)
run
(*simulate*)
analyze
(*process **r**(t)*)
end


generate
(setup)
run
(simulate)
analyze
(process r(t))
end


generate
(setup)
run
(simulate)
analyze
(process
r(t))
end

workflows

building blocks

workflows

MDAnalysis

scipy | mmtf | networkx

numpy

building blocks

Python

Python

workflows

MDAnalysis

scipy | mmtf | networkx

numpy

alchemlyb

pymbar | pandas

numpy

building blocks

Python

Python

workflows

MDAnalysis

scipy | mmtf | networkx

numpy

GromacsWrapper

Gromacs

alchemlyb

pymbar | pandas

numpy

building blocks

Python

Python

“not Python”

workflows

MDPOW

Gromacs Wrapper

MDAnalysis

alchemlyb

building blocks

MDAnalysis

scipy

mmtf

networkx

numpy

GromacsWrapper

Gromacs

alchemlyb

pymbar

pandas

numpy

Python

Python

“not Python”

workflows

MDPOW

Gromacs Wrapper | **MDAnalysis** | **alchemlyb**

mdworks

Gromacs Wrapper | FireWorks | **alchemlyb**

MDAnalysis

scipy | mmtf | networkx

numpy

GromacsWrapper

Gromacs

alchemlyb

pymbar | pandas

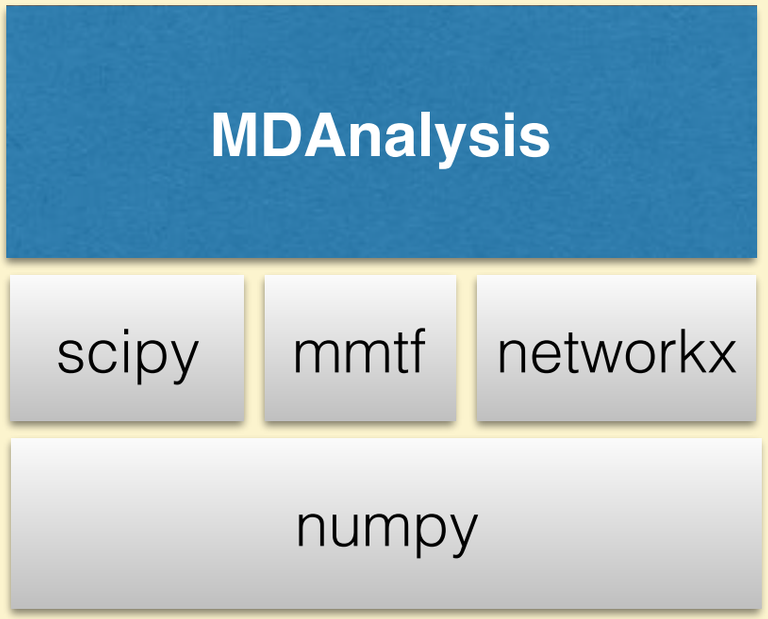
numpy

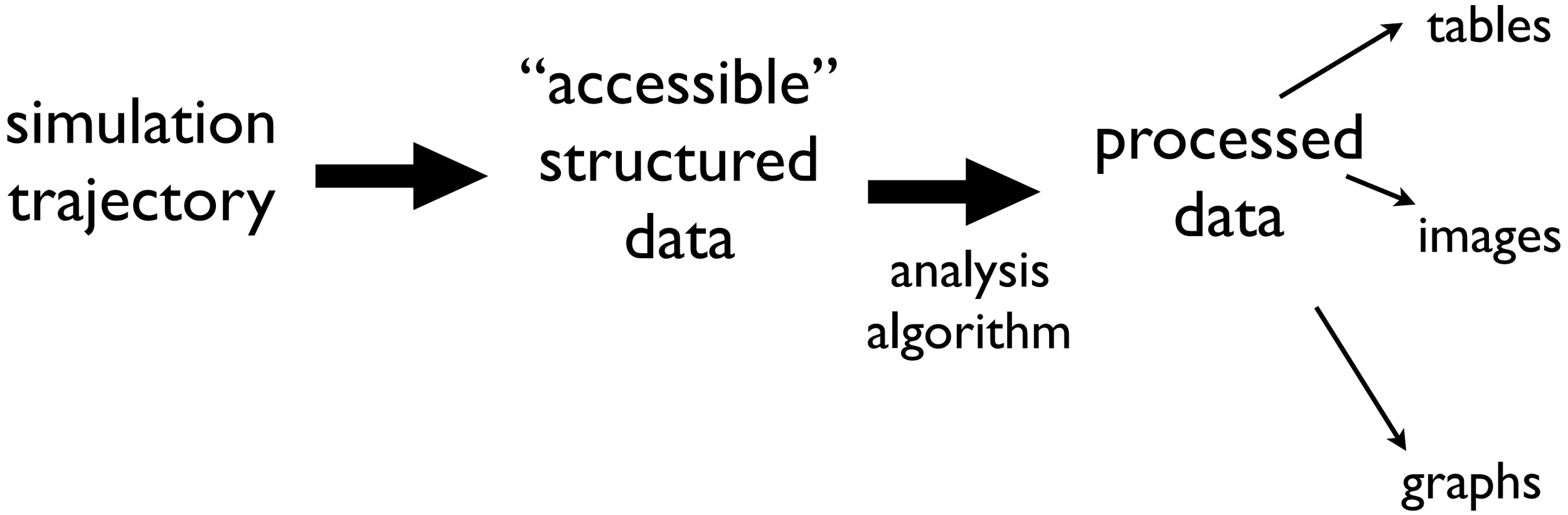
building blocks

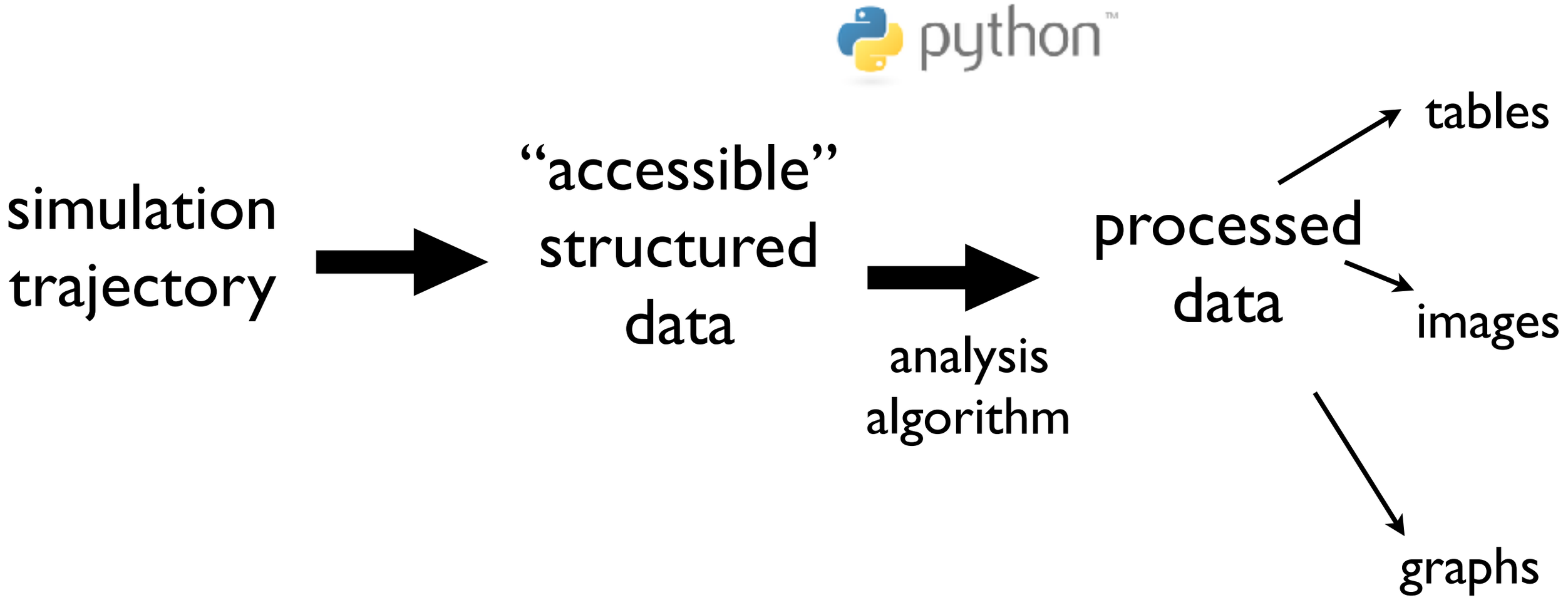
Python

Python

"not Python"


MDAnalysis
scipy
mmtf
networkx
numpy


simulation
trajectory
"accessible"
structured
data
analysis
algorithm
processed
data
tables
images
graphs


python
simulation trajectory
"accessible" structured data
analysis algorithm
processed data
tables
images
graphs

NumPy python™

simulation trajectory

"accessible" structured data

processed data

analysis algorithm

tables

images

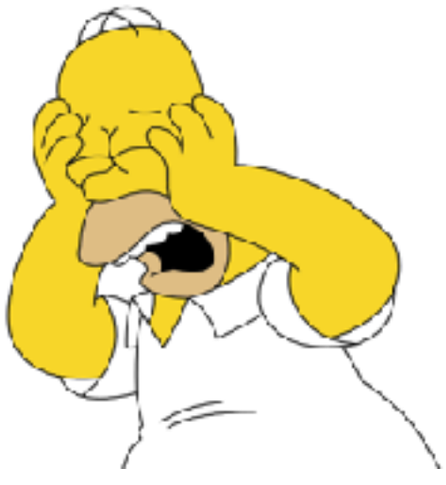
graphs

NumPy python™

simulation trajectory

dcd, xtc, trr, ncdf, trj, pdb, pqr, gro, crd, dms, trz, mol2, xyz, config, history, gms, ...

psf, tpr, prmtop, dms, mol2, hoomd xml, ...



Oh nooooo!

"accessible" structured data

analysis algorithm

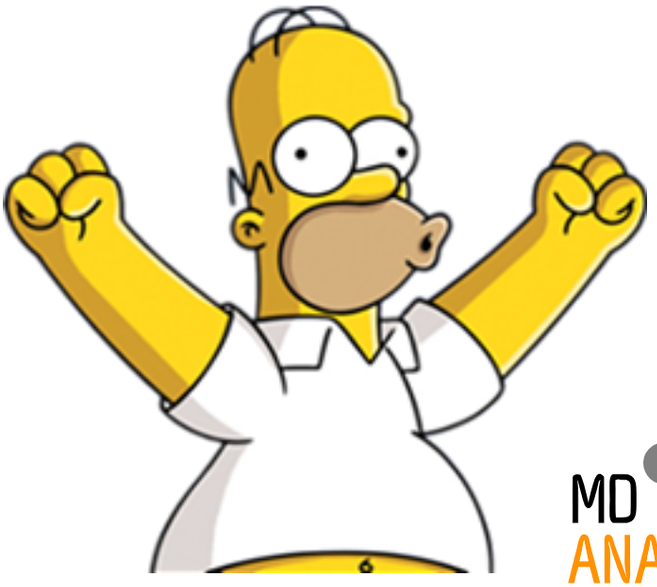
processed data

tables

images

graphs

Woo-hoo!



MD ANALYSIS

NumPy

python™

simulation trajectory

dcd, xtc, trr, ncdf, trj, pdb, pqr, gro, crd, dms, trz, mol2, xyz, config, history, gms, ...

psf, tpr, prmtop, dms, mol2, hoomd xml, ...

"accessible" structured data

analysis algorithm

processed data

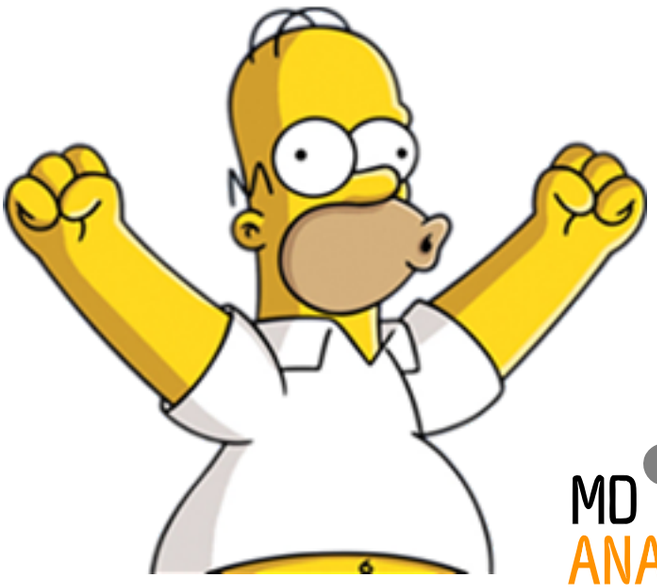
tables

images

graphs

Oh nooooo!

Woo-hoo!



MD ANALYSIS

NumPy

python™

simulation trajectory

dcd, xtc, trr, ncdf, trj, pdb, pqr, gro, crd, dms, trz, mol2, xyz, config, history, gms, ...

psf, tpr, prmtop, dms, mol2, hoomd xml, ...

"accessible" structured data

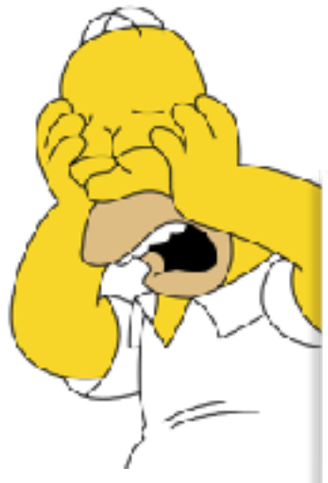
analysis algorithm

processed data

tables

images

graphs



Oh nooo

25 coordinate formats

20 topology formats

(v0.19.2)

MDTools (JC Phillips, 1996)

MMTK (K Hinsen, 2000)

simulation trajectory

dcd, xtc, trr, ncdf, trj, pdb, pqr, gro, crd, dms, trz, mol2, xyz, config, history, gms, ...

psf, tpr, prmtop, dms, mol2, hoomd xml, ...

"accessible" structured data

LOOS (2009)

MDAnalysis (2011)

pyLOOS (2014)

mdtraj (2015)

pytraj

…

analysis algorithm

VMD
Gromacs
cpptraj
CHARMM
Chimera

…

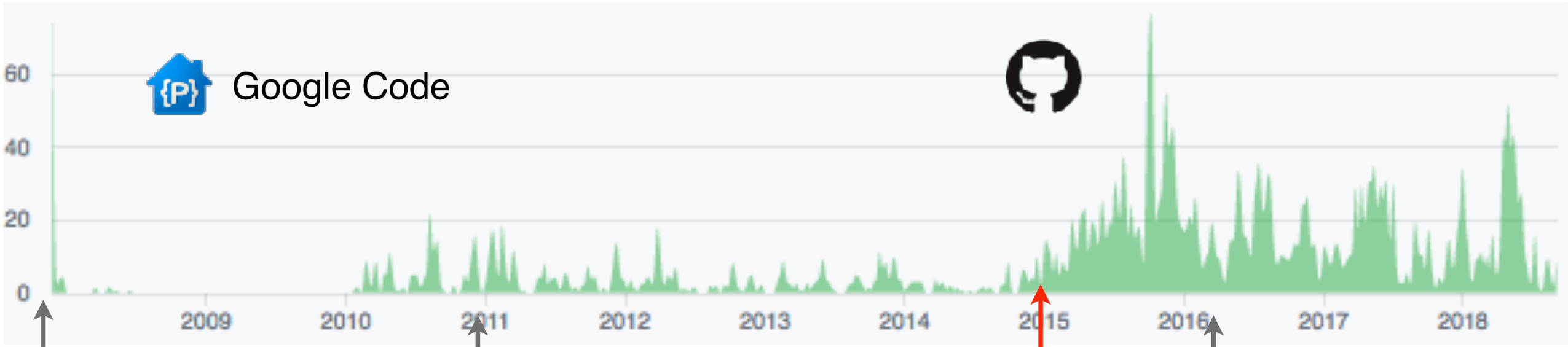
processed data

tables

images

graphs

MDAnalysis repository commit history




```
Author: Naveen Michaud-Agrawal
Date:   Thu Jan 31 11:42:33 2008 +0000

initial import
```

Software News and Updates
MDAnalysis: A Toolkit for the Analysis of Molecular Dynamics Simulations

NAVEEN MICHAUD-AGRAWAL,[1] ELIZABETH J. DENNING,[1,2] THOMAS B. WOOLF,[1,3] OLIVER BECKSTEIN[3,4]

Received 23 October 2010; Revised 6 February 2011; Accepted 12 February 2011
DOI 10.1002/jcc.21787
Published online 15 April 2011 in Wiley Online Library (wileyonlinelibrary.com).

J Comput Chem 32: 2319–2327, 2011

MDAnalysis: A Python Package for the Rapid Analysis of Molecular Dynamics Simulations

Richard J. Gowers[‖**†], Max Linke[‡‡†], Jonathan Barnoud[§†], Tyler J. E. Reddy[‡], Manuel N. Melo[§], Sean L. Seyler[¶], Jan Domański[‡], David L. Dotson[¶], Sébastien Buchoux[††], Ian M. Kenney[¶], Oliver Beckstein[¶*]

In S. Benthall and S. Rostrup, editors, Proceedings of the 15th Python in Science Conference, pages 98–105, Austin, TX, 2016. SciPy

(>600 citations on GoogleScholar (Feb 2019))

Naveen Michaud-Agrawal, Elizabeth J. Denning, Christian Beckstein (logo), Joshua L. Adelman, Shobhit Agarwal, Irfan Alibay, Balasubramanian, Utkarsh Bansal, Jonathan Barnoud, Tone Bengtsen, Alejandro Bernardin, Mateusz Bieniek, Wouter Boomsma, Jose Borreguero, Bart Bruininks, Sébastien Buchoux, Sören von Bülow, David Caplan, Matthieu Chavent, Kathleen Clark, Ruggero Cortini, Davide Cruz, Robert Delgado, John Detlefs, Xavier Deupi, Jan Domanski, David L. Dotson, Shujie Fan, Lennard van der Feltz, Philip Fowler, Joseph Goose, Richard J. Gowers, Lukas Grossar, Abhinav Gupta, Akshay Gupta, Benjamin Hall, Eugen Hruska, Kyle J. Huston, Joe Jordan, Jon Kapla, Navya Khare, Andrew William King, Max Linke, Philip Loche, Jinju Lu, Micaela Matta, Andrew R. McCluskey, Robert McGibbon, Manuel Nuno Melo, Dominik 'Rathann' Mierzejewski, Henry Mull, Fiona B. Naughton, Alex Nesterenko, Hai Nguyen, Sang Young Noh, Nabarun Pal, Mattia F. Palermo, Danny Parton, Joshua L. Phillips, Kashish Punjani, Vedant Rathore, Tyler Reddy, Pedro Reis, Paul Rigor, Carlos Yanez S., Utkarsh Saxena, Sean L. Seyler, Paul Smith, Andy Somogyi, Caio S. Souza, Shantanu Srivastava, Lukas Stelzl, Gorman Stock, Ayush Suhane, Xiki Tempula, Matteo Tiberti, Isaac Virshup, Nestor Wendt, Zhiyi Wu, Zhuyi Xue, Juan Eiros Zamora, Johannes Zeman, and Oliver Beckstein.

9 core developers
5 GSoC students
4 REU students
83 contributors

Join us at

mdanalysis.org
github.com/MDAnalysis

NUMFOCUS
OPEN CODE = BETTER SCIENCE

ASU

NSF
ACI-1443054

python SOFTWARE FOUNDATION

Google
GSoC 2016–2019

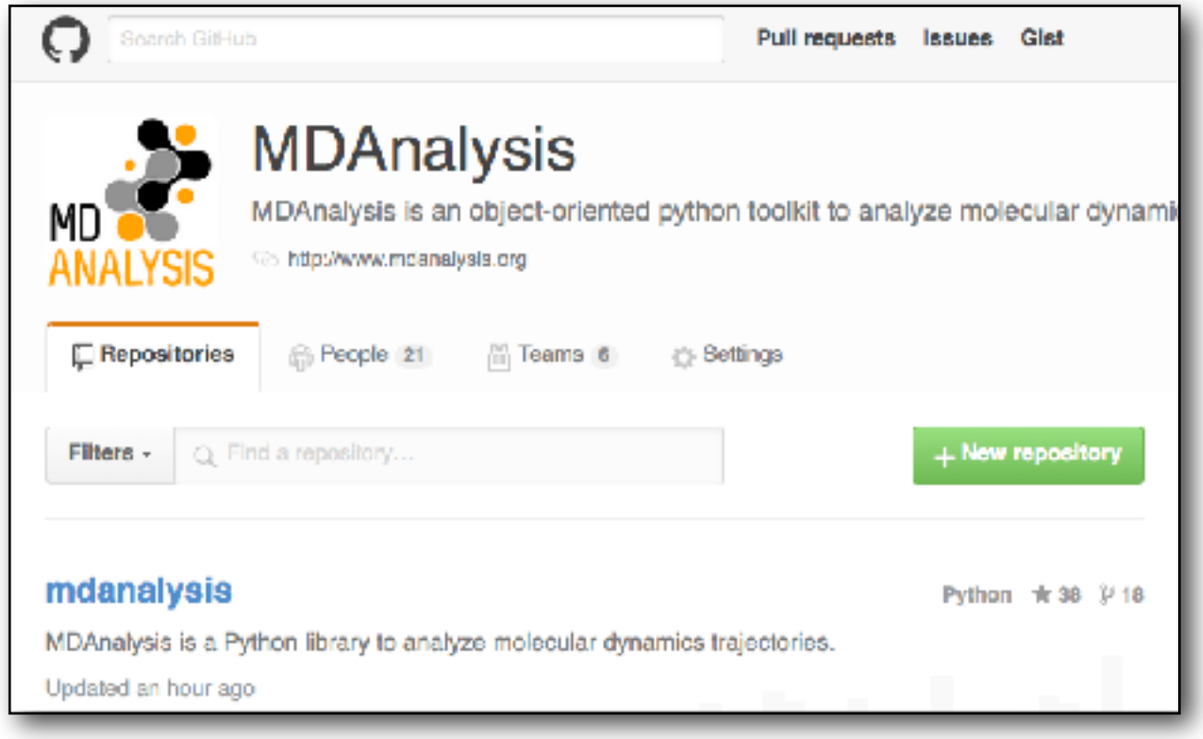
Open source

- GPL v2
- github.com/MDAnalysis

Runs on

- Linux
- macOS
- Windows

GitHub



Installs with

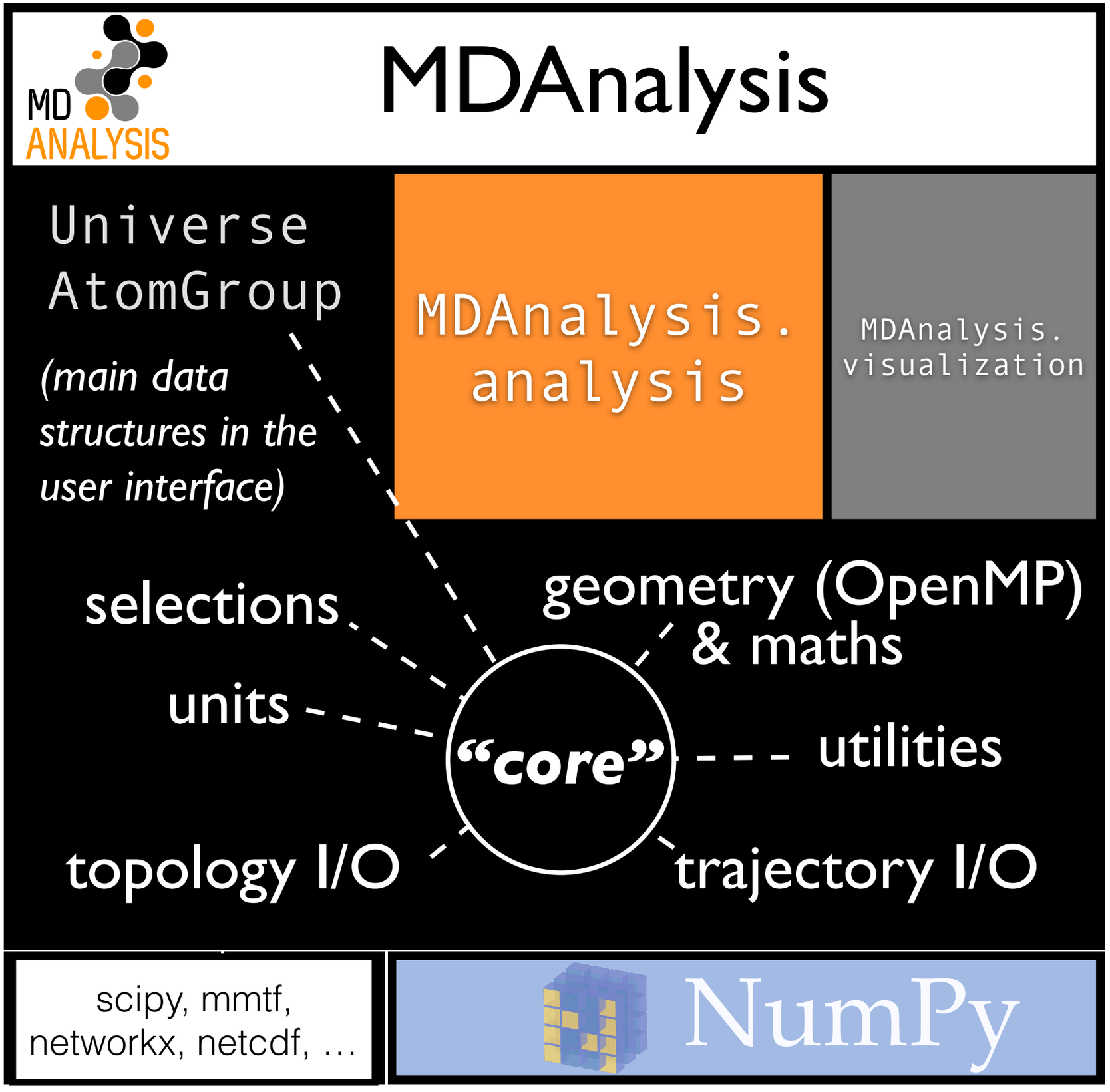
- `conda install -c conda-forge mdanalysis`
- `pip install mdanalysis`

Development process

- pull request / review / merge
- continuous integration with > 6,500 unit tests

Travis CI

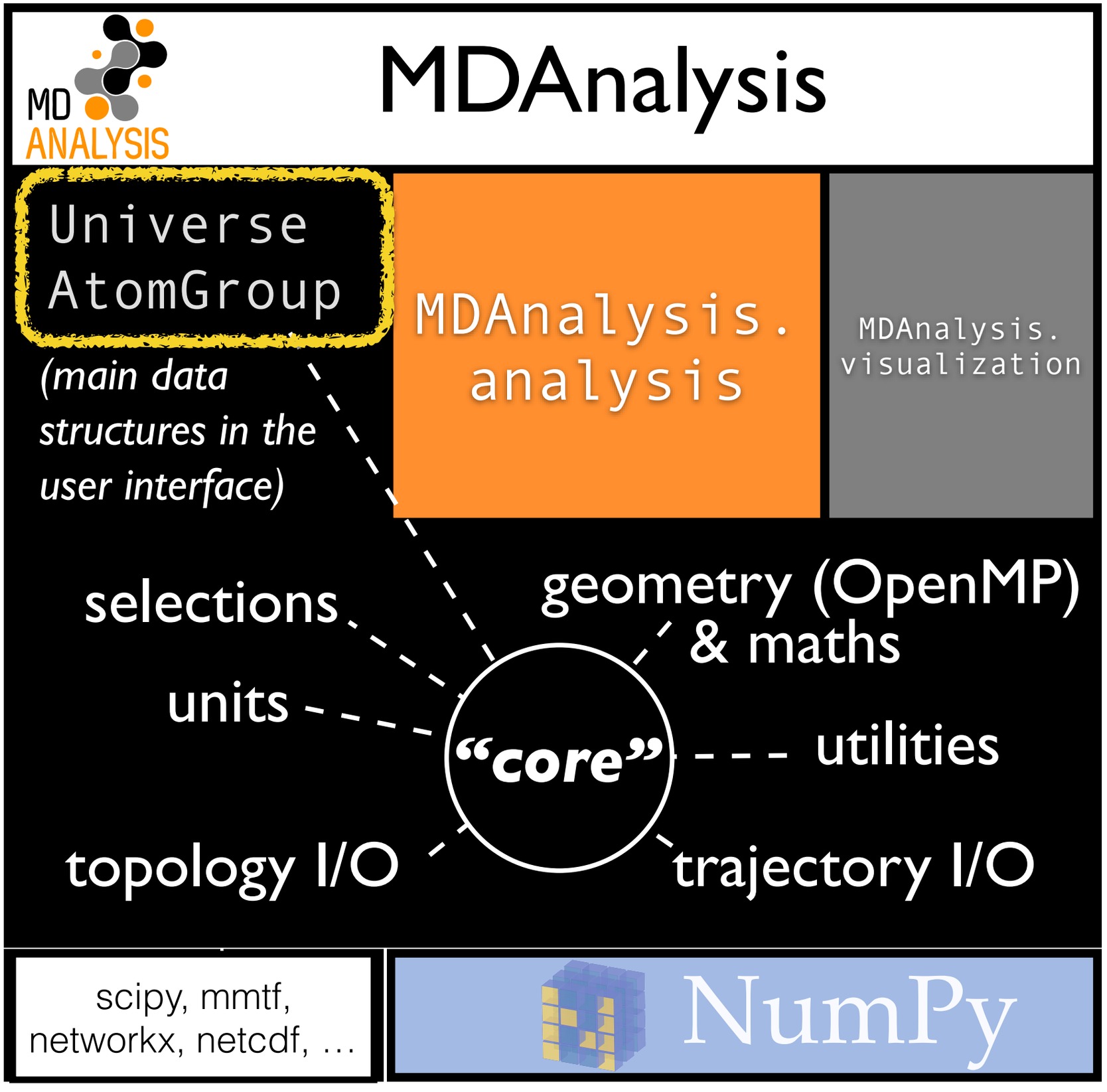
AppVeyor



https://mdanalysis.org

Code base:

- python 3 & 2.7
- cython
- C
- ~63k LOC
- ~39k lines comments



https://mdanalysis.org

Code base:

- python 3 & 2.7
- cython
- C
- ~63k LOC
- ~39k lines comments

Example analysis: Per-residue RMSF

- root mean square fluctuation ρ_i (RMSF) measures local flexibility of amino acid *i*
- standard quantity to compute for protein simulations

$$\rho_i = \sqrt{\left\langle (\mathbf{x}_i(t) - \langle \mathbf{x}_i \rangle)^2 \right\rangle}$$

- use the "C-alpha" atom in each residue to characterise the motion of the whole residue: $\boldsymbol{x}_i$ (position of $C_{\alpha,i}$)

C_α RMSF

$$\rho_i = \sqrt{\left\langle (\mathbf{x}_i(t) - \langle \mathbf{x}_i \rangle)^2 \right\rangle}$$


```
import numpy as np
import MDAnalysis as mda

u = mda.Universe("topol.tpr", "trj.xtc")
ca = u.select_atoms("name CA")
means = np.zeros((len(ca), 3))
sumsq = np.zeros_like(means)
for k, ts in enumerate(u.trajectory):
    sumsq += (k/(k+1.0)) * (ca.positions - means)**2
    means[:] = (k*means + ca.positions)/(k+1.0)
rmsf = np.sqrt(sumsq.sum(axis=1)/(k+1.0))

matplotlib.pyplot.plot(ca.residues.resids, rmsf)
```

C_α RMSF

$$\rho_i = \sqrt{\left\langle (\mathbf{x}_i(t) - \langle \mathbf{x}_i \rangle)^2 \right\rangle}$$


```
import numpy as np
import MDAnalysis as mda

u = mda.Universe("topol.tpr", "trj.xtc")
ca = u.select_atoms("name CA")
means = np.zeros((len(ca), 3))
sumsq = np.zeros_like(means)
for k, ts in enumerate(u.trajectory):
    sumsq += (k/(k+1.0)) * (ca.positions - means)**2
    means[:] = (k*means + ca.positions)/(k+1.0)
rmsf = np.sqrt(sumsq.sum(axis=1)/(k+1.0))

matplotlib.pyplot.plot(ca.residues.resids, rmsf)
```

C_α RMSF

$$\rho_i = \sqrt{\left\langle (\mathbf{x}_i(t) - \langle \mathbf{x}_i \rangle)^2 \right\rangle}$$


```
import numpy as np
import MDAnalysis as mda

u = mda.Universe("topol.tpr", "trj.xtc")
ca = u.select_atoms("name CA")
means = np.zeros((len(ca), 3))
sumsq = np.zeros_like(means)
for k, ts in enumerate(u.trajectory):
    sumsq += (k/(k+1.0)) * (ca.positions - means)**2
    means[:] = (k*means + ca.positions)/(k+1.0)
rmsf = np.sqrt(sumsq.sum(axis=1)/(k+1.0))

matplotlib.pyplot.plot(ca.residues.resids, rmsf)
```

C_α RMSF

$$\rho_i = \sqrt{\left\langle (\mathbf{x}_i(t) - \langle \mathbf{x}_i \rangle)^2 \right\rangle}$$


```
import numpy as np
import MDAnalysis as mda

u = mda.Universe("topol.tpr", "trj.xtc")
ca = u.select_atoms("name CA")
means = np.zeros((len(ca), 3))
sumsq = np.zeros_like(means)
for k, ts in enumerate(u.trajectory):
    sumsq += (k/(k+1.0)) * (ca.positions - means)**2
    means[:] = (k*means + ca.positions)/(k+1.0)
rmsf = np.sqrt(sumsq.sum(axis=1)/(k+1.0))

matplotlib.pyplot.plot(ca.residues.resids, rmsf)
```

C_α RMSF

$$\rho_i = \sqrt{\left\langle (\mathbf{x}_i(t) - \langle \mathbf{x}_i \rangle)^2 \right\rangle}$$


```
import numpy as np
import MDAnalysis as mda

u = mda.Universe("topol.tpr", "trj.xtc")
ca = u.select_atoms("name CA")
means = np.zeros((len(ca), 3))
sumsq = np.zeros_like(means)
for k, ts in enumerate(u.trajectory):
    sumsq += (k/(k+1.0)) * (ca.positions - means)**2
    means[:] = (k*means + ca.positions)/(k+1.0)
rmsf = np.sqrt(sumsq.sum(axis=1)/(k+1.0))

matplotlib.pyplot.plot(ca.residues.resids, rmsf)
```

C_α RMSF

$$\rho_i = \sqrt{\left\langle (\mathbf{x}_i(t) - \langle \mathbf{x}_i \rangle)^2 \right\rangle}$$


```
import numpy as np
import MDAnalysis as mda

u = mda.Universe("topol.tpr", "trj.xtc")
ca = u.select_atoms("name CA")
means = np.zeros((len(ca), 3))
sumsq = np.zeros_like(means)
for k, ts in enumerate(u.trajectory):
    sumsq += (k/(k+1.0)) * (ca.positions - means)**2
    means[:] = (k*means + ca.positions)/(k+1.0)
rmsf = np.sqrt(sumsq.sum(axis=1)/(k+1.0))

matplotlib.pyplot.plot(ca.residues.resids, rmsf)
```

C_α RMSF

$$\rho_i = \sqrt{\left\langle (\mathbf{x}_i(t) - \langle \mathbf{x}_i \rangle)^2 \right\rangle}$$


```
import numpy as np
import MDAnalysis as mda

u = mda.Universe("topol.tpr", "trj.xtc")
ca = u.select_atoms("name CA")
means = np.zeros((len(ca), 3))
sumsq = np.zeros_like(means)
for k, ts in enumerate(u.trajectory):
    sumsq += (k/(k+1.0)) * (ca.positions - means)**2
    means[:] = (k*means + ca.positions)/(k+1.0)
rmsf = np.sqrt(sumsq.sum(axis=1)/(k+1.0))

matplotlib.pyplot.plot(ca.residues.resids, rmsf)
```

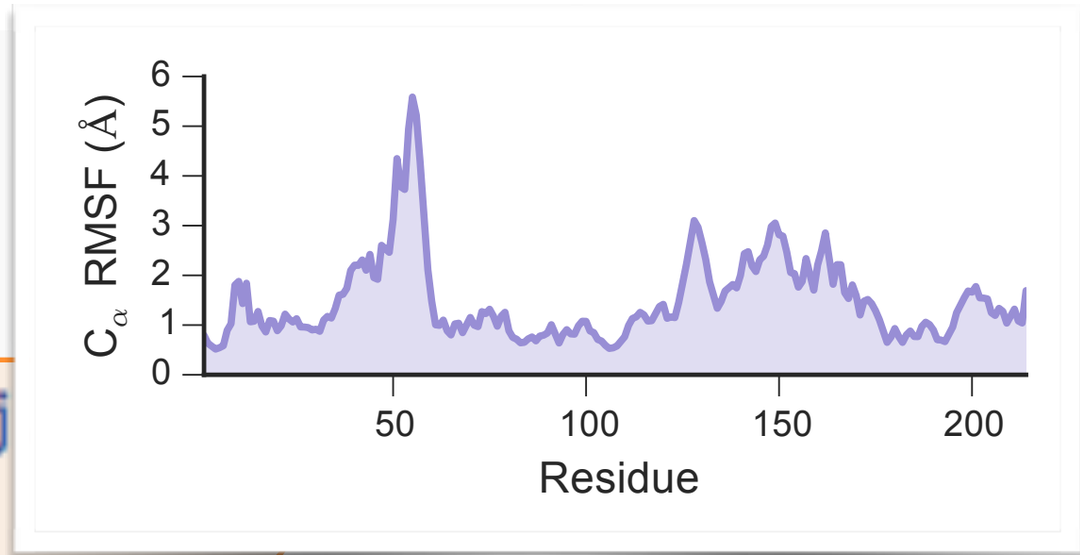
C_α RMSF

$$\rho_i = \sqrt{\left\langle (\mathbf{x}_i(t) - \langle \mathbf{x}_i \rangle)^2 \right\rangle}$$

MD ANALYSIS

NumPy

```
import numpy as np
import MDAnalysis as mda

u = mda.Universe("topol.tpr", "trj
ca = u.select_atoms("name CA")
means = np.zeros((len(ca), 3))
sumsq = np.zeros_like(means)
for k, ts in enumerate(u.trajectory):
    sumsq += (k/(k+1.0)) * (ca.positions - means)**2
    means[:] = (k*means + ca.positions)/(k+1.0)
rmsf = np.sqrt(sumsq.sum(axis=1)/(k+1.0))

matplotlib.pyplot.plot(ca.residues.resids, rmsf)
```



GromacsWrapper

Gromacs

GromacsWrapper

https://github.com/becksteinlab/gromacswrapper

- terrible hack to pretend that there's a Python API for the MD package ***Gromacs*** (http://www.gromacs.org)

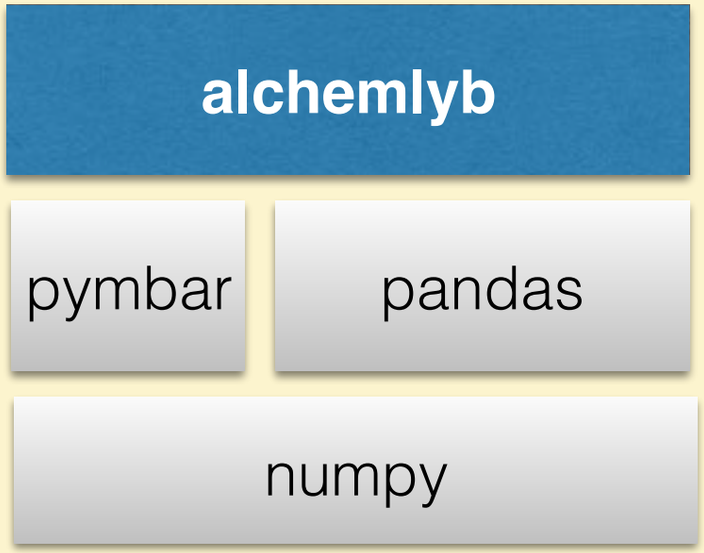
```
gmx grompp -f md.mdp -s md.tpr -c system.pdb
gmx mdrun -s md.tpr -v
```

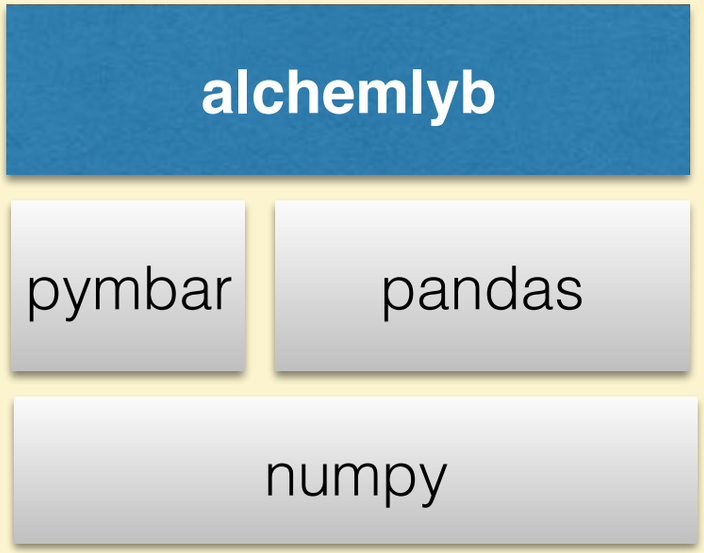
shell

```
import gromacs as gmx
gmx.grompp(f="md.mdp", s="md.tpr", c="system.pdb")
gmx.mdrun(s="md.tpr", v=True)
```

Python

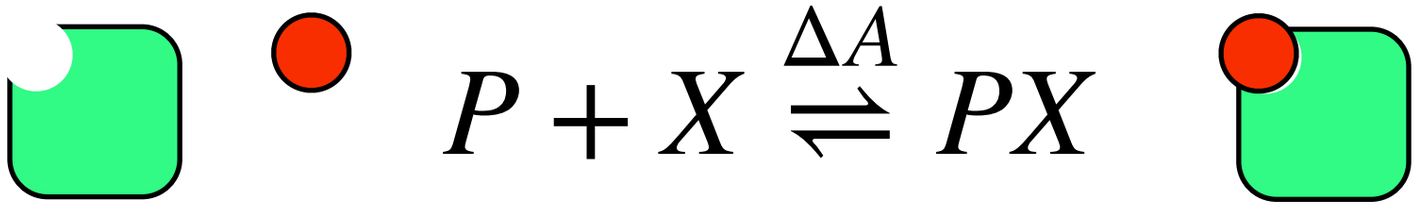
- **(now: see also *gmxapi* https://github.com/kassonlab/gmxapi)**


alchemlyb
pymbar
pandas
numpy

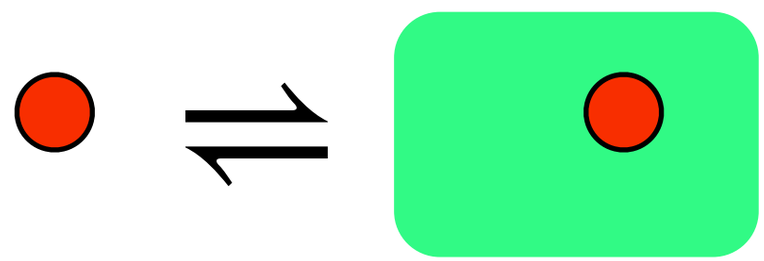


- O Beckstein (ASU), D Mobley (UC Irvine), M Shirts (U Colorado, Boulder)
- **David Dotson** (ASU), Dominik Wille (Freie Univ. Berlin)
- Silicon Therapeutics (STX) (Bryce Allen, Shuai Liu)

Binding free energy



Solvation free energy



- **free** energy (i.e., averaged over all other degrees of freedom (such as solvent, protein motions, …))

$$\exp[-A(T,V,N)/kT] = \int dp^{3N} dx^{3N} e^{-H(p,x)/kT} \qquad H(p,x) = \sum_{i=1}^{N} \frac{\mathbf{p}_i^2}{2m_i} + U(x)$$

- **free energy difference**

$$\Delta A = A_{\textbf{bound}} - A_{\textbf{unbound}}$$

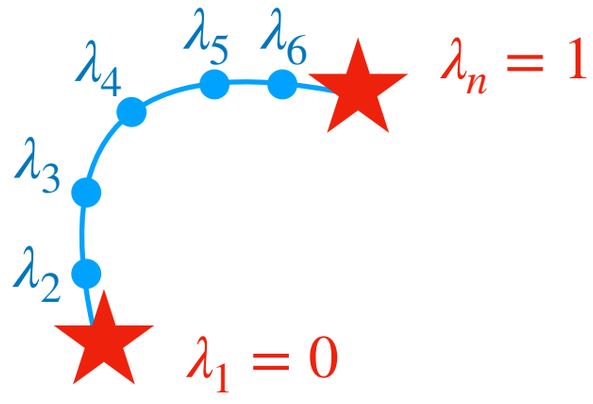
$$\Delta A = -\ln \frac{\int dx^{3N} \exp[-U(x)/kT]\, \chi_{\textbf{bound}}(x)}{\int dx^{3N} \exp[-U(x)/kT]\, \chi_{\textbf{unbound}}(x)}$$

Alchemical free energy calculations

- force fields for *H*, molecular dynamics (MD) simulations for sampling
- free energy is a **state function:** generate non-physical paths between physical end states (bound/unbound)
- use **stratification** (“windows”) of path with parameter λ

$$H(\lambda) = (1-\lambda)H_{\textbf{bound}} + \lambda H_{\textbf{unbound}}, \quad 0 \le \lambda \le 1$$

$$U_{\textbf{Coulomb}}(\mathbf{x}_1, \mathbf{x}_2; \lambda) = \frac{1}{4\pi\epsilon_0}\frac{(1-\lambda)q_1 q_2}{|\mathbf{x}_1 - \mathbf{x}_2|}$$



Methods

- "Free Energy Perturbation" (**FEP**): Zwanzig FEP, **BAR**, **MBAR** (overlaps of distributions)

$$\Delta U(x) = U_{\lambda_{n+1}}(x) - U_{\lambda_n}(x)$$

$$\Delta A = -kT \ln\langle \exp[-\Delta U(x)/kT] \rangle_1, \quad \textbf{with } \Delta U(x) = U_1(x) - U_0(x) \qquad \textbf{FEP}$$

$$\exp(-\Delta A/kT) = \frac{\langle 1 + \exp[(\Delta U - C)/kT]^{-1} \rangle_0}{\langle 1 + \exp[(\Delta U - C)/kT]^{-1} \rangle_1} \exp(-C/kT) \qquad \textbf{BAR}$$

$$\Delta A/kT = C/kT - \ln \frac{n_1}{n_2}$$

(MBAR is more complicated: uses overlaps between all windows)

- Thermodynamic Integration (**TI**): TI

$$\Delta A = \int_0^1 d\lambda \left\langle \frac{\partial H(\lambda)}{\partial \lambda} \right\rangle_\lambda \qquad \textbf{TI}$$

C. Chipot and A. Pohorille, editors. Free energy calculations. Number 86 in Springer Series in Chemical Physics. Springer, Berlin, 2007.

Simulation output

Per timestep

- FEP: ΔU
- TI: $\partial U/\partial \lambda$

$\Delta U_{\lambda_i,\lambda_j} \forall j$ for window *i*

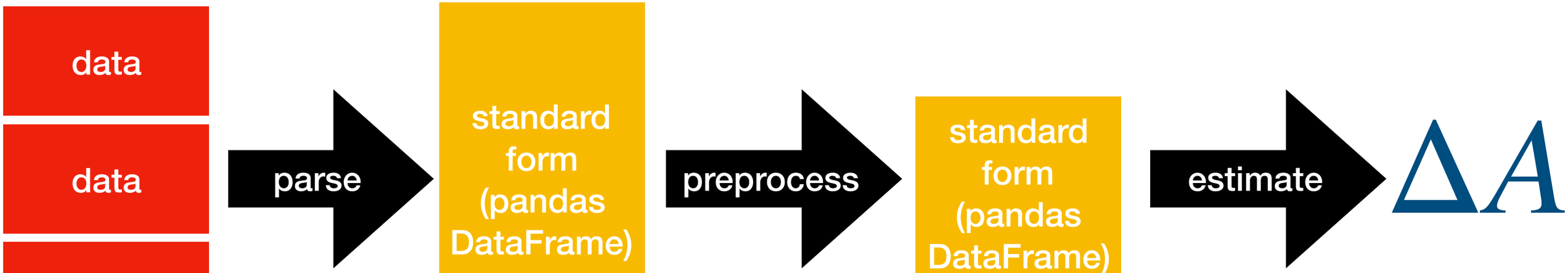
$$\left.\frac{\partial U(x;\lambda)}{\partial \lambda}\right|_{\lambda_i}$$

But: **different file formats for different codes** (Gromacs, Amber, NAMD, …)

solution: common interface via ***alchemlyb***

alchemlyb

https://github.com/alchemistry/alchemlyb



$$\Delta U_{\lambda_i,\lambda_j} \forall j$$

$$\left.\frac{\partial U(x;\lambda)}{\partial \lambda}\right|_{\lambda_i}$$

preprocessors

- **slicing**
- **statistical inefficiency**
- **equilibrium detection**
- **…**

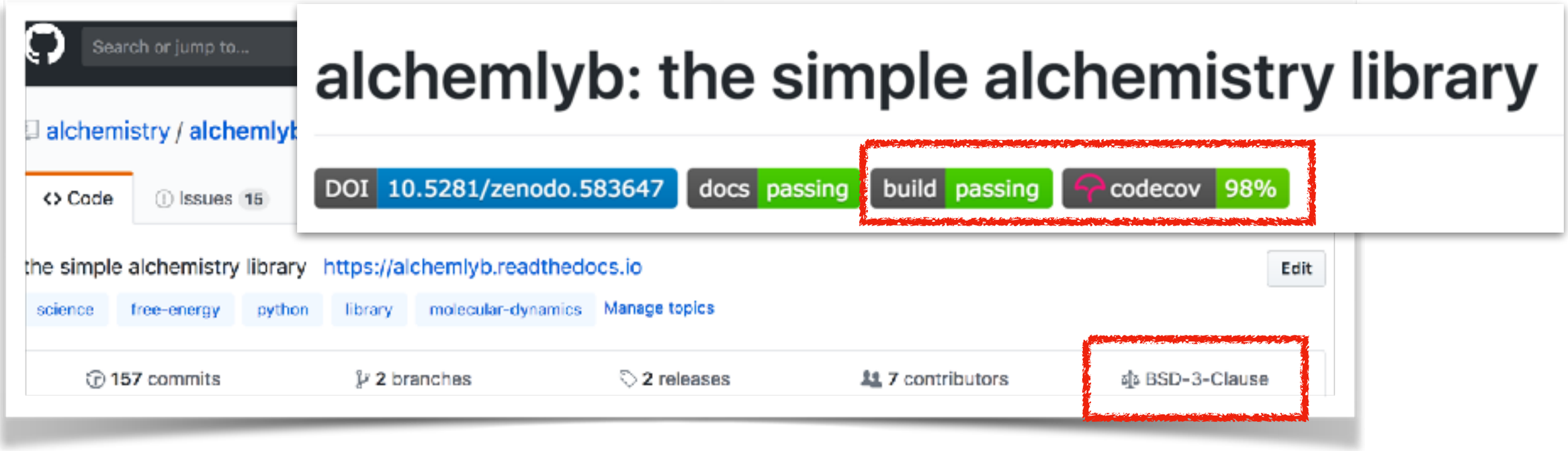
estimators

- **TI**
- **BAR**
- **MBAR**
- **…**

(interface modelled after *scikit-learn*)

alchemlyb

https://github.com/alchemistry/alchemlyb



https://github.com/becksteinlab/mdpow

SAMPL3 –5

ΔG_{hyd}

$\log P_{cw}$

SAMPL6 $\log P_{ow}$

[1] O. Beckstein, A. Fourrier, and B. I. Iorga. J Comput Aided Mol Des, 28(3): 265–276, 2014.
[2] O. Beckstein and B. I. Iorga. J Comput Aided Mol Des, 26(5):635–645, 2012.
[3] I. M. Kenney, O. Beckstein, and B. I. Iorga. J Comput Aided Mol Des, 30(11):1045–1058, 2016.

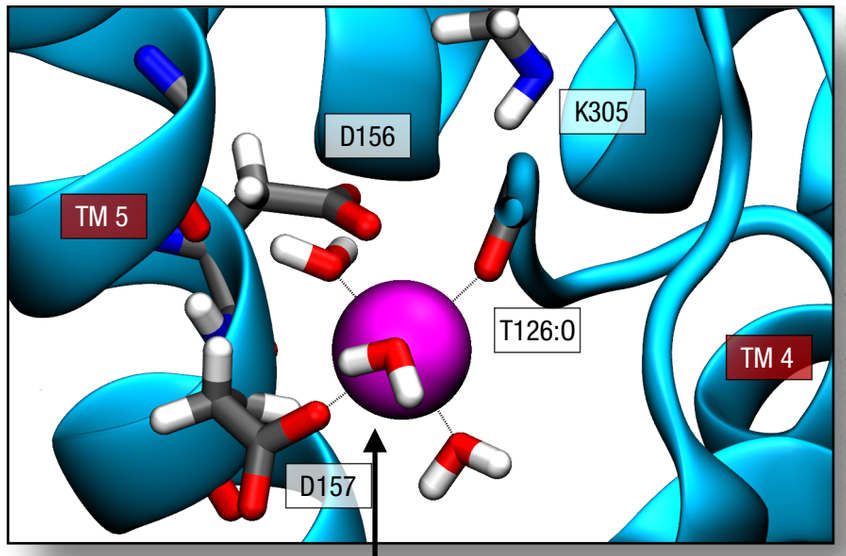
mdworks

Gromacs Wrapper

FireWorks

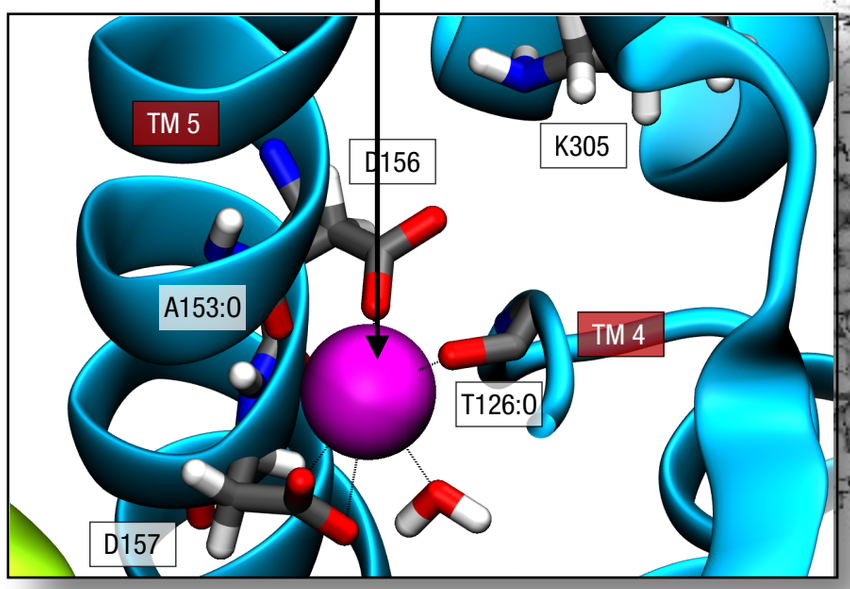
alchemlyb

NapA elevator mechanism



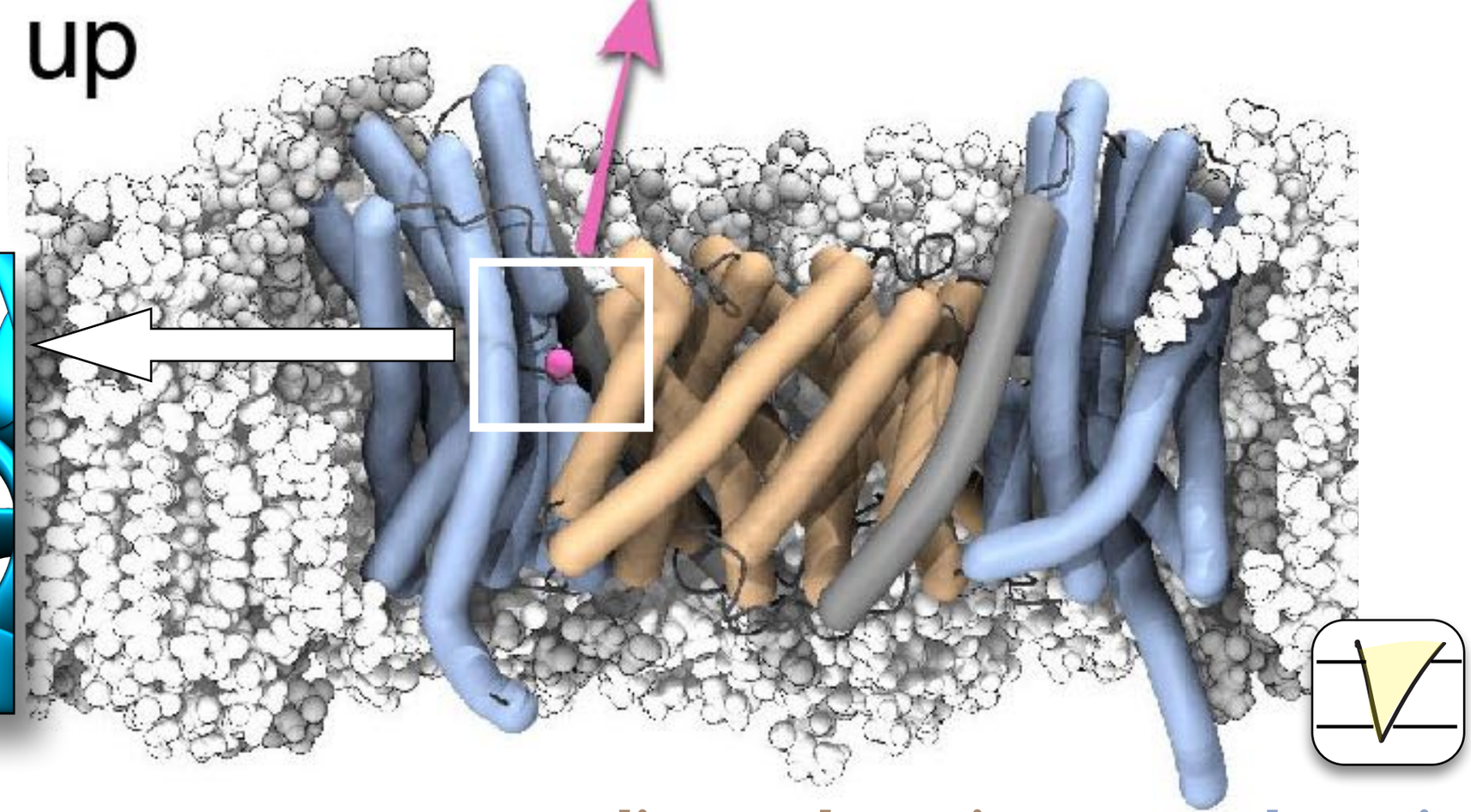
Nature Struct. Mol. Biol., 23 (2016):248–255

ion binding in different states?



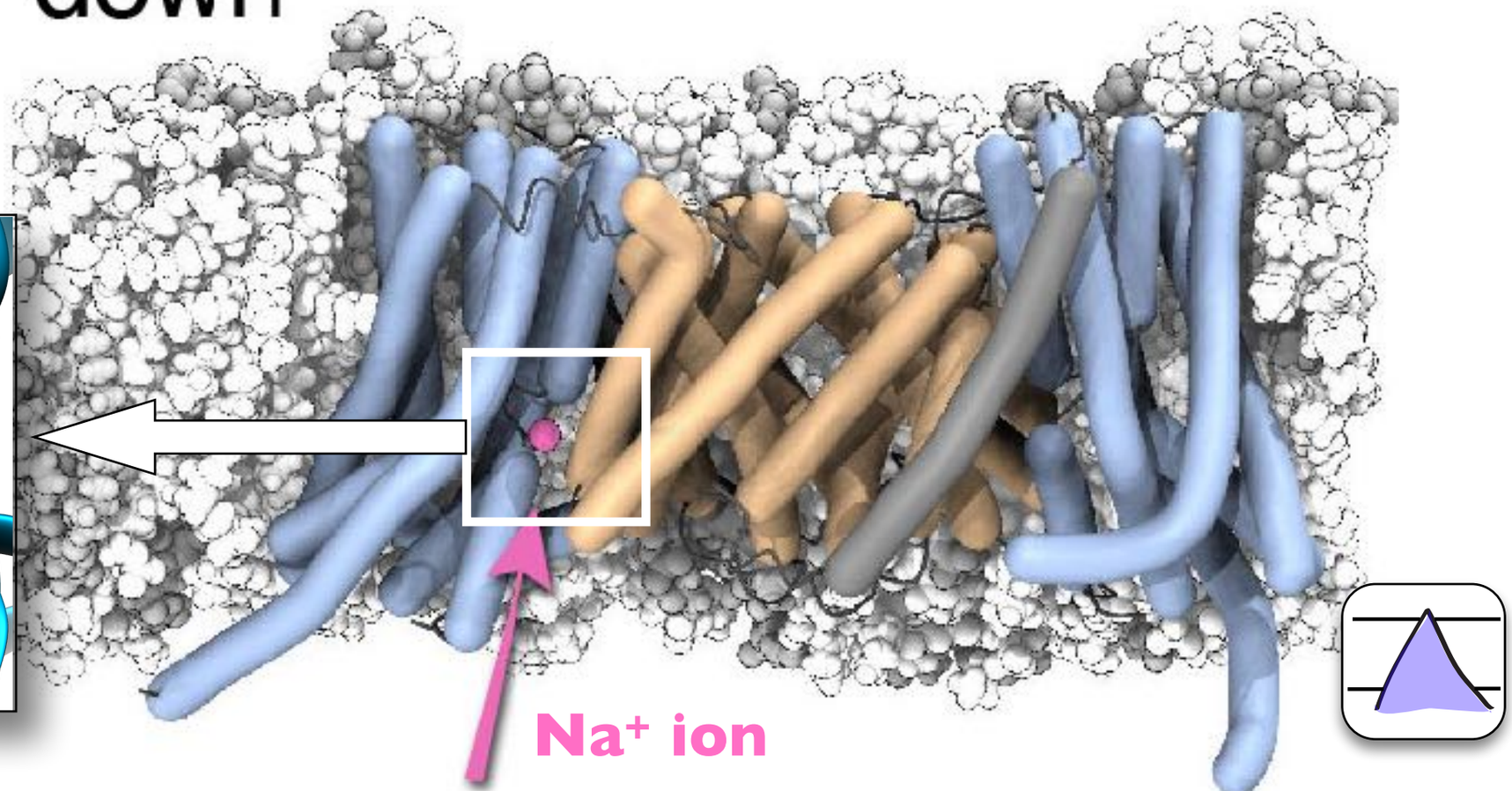
Nature, 501 (2013):573–577.

up



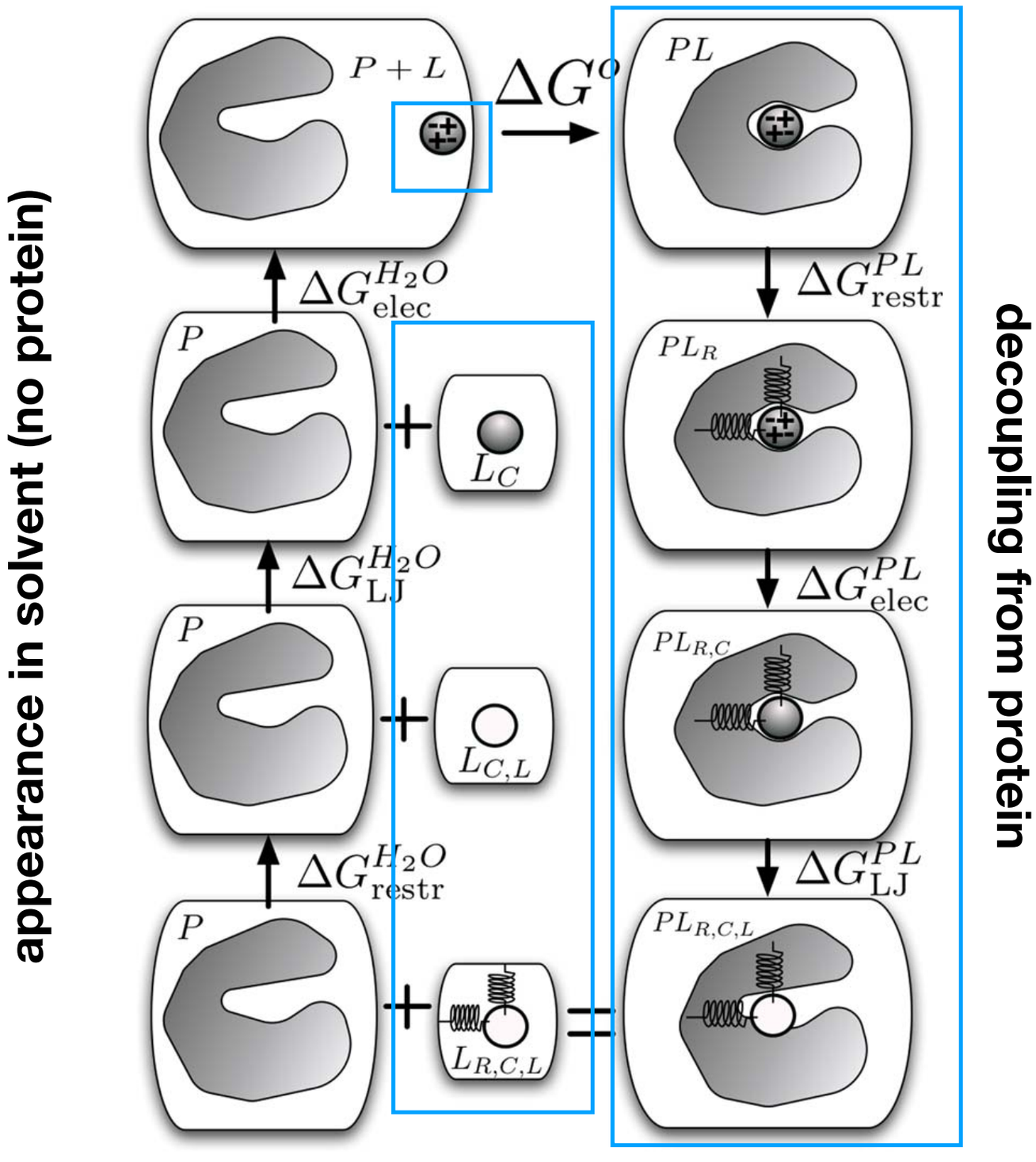
dimer domain core domain

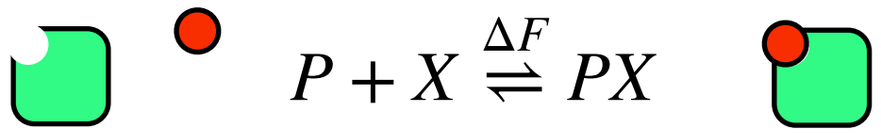
down



Na^+ ion

Thermodynamic cycle for absolute binding free energy calculations



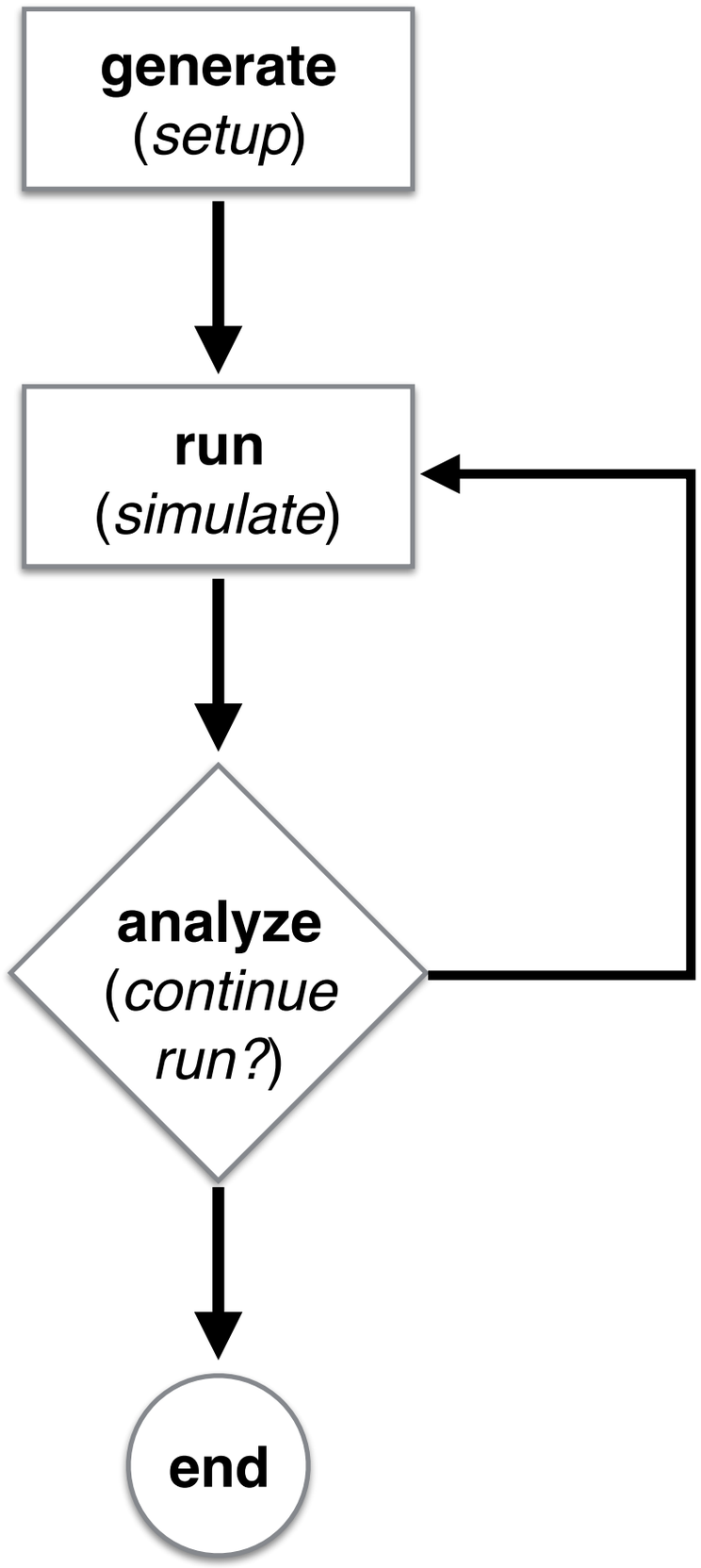


MD simulations with multiple λ

- 5 – 50 λ-windows per free energy component
- 10 ns – 250 ns per window

D. L. Mobley, J. D. Chodera, and K. A. Dill. On the use of orientational restraints and symmetry corrections in alchemical free energy calculations. 125:084902, 2006.

for each λ window:



GromacsWrapper

Gromacs

GromacsWrapper

MDAnalysis

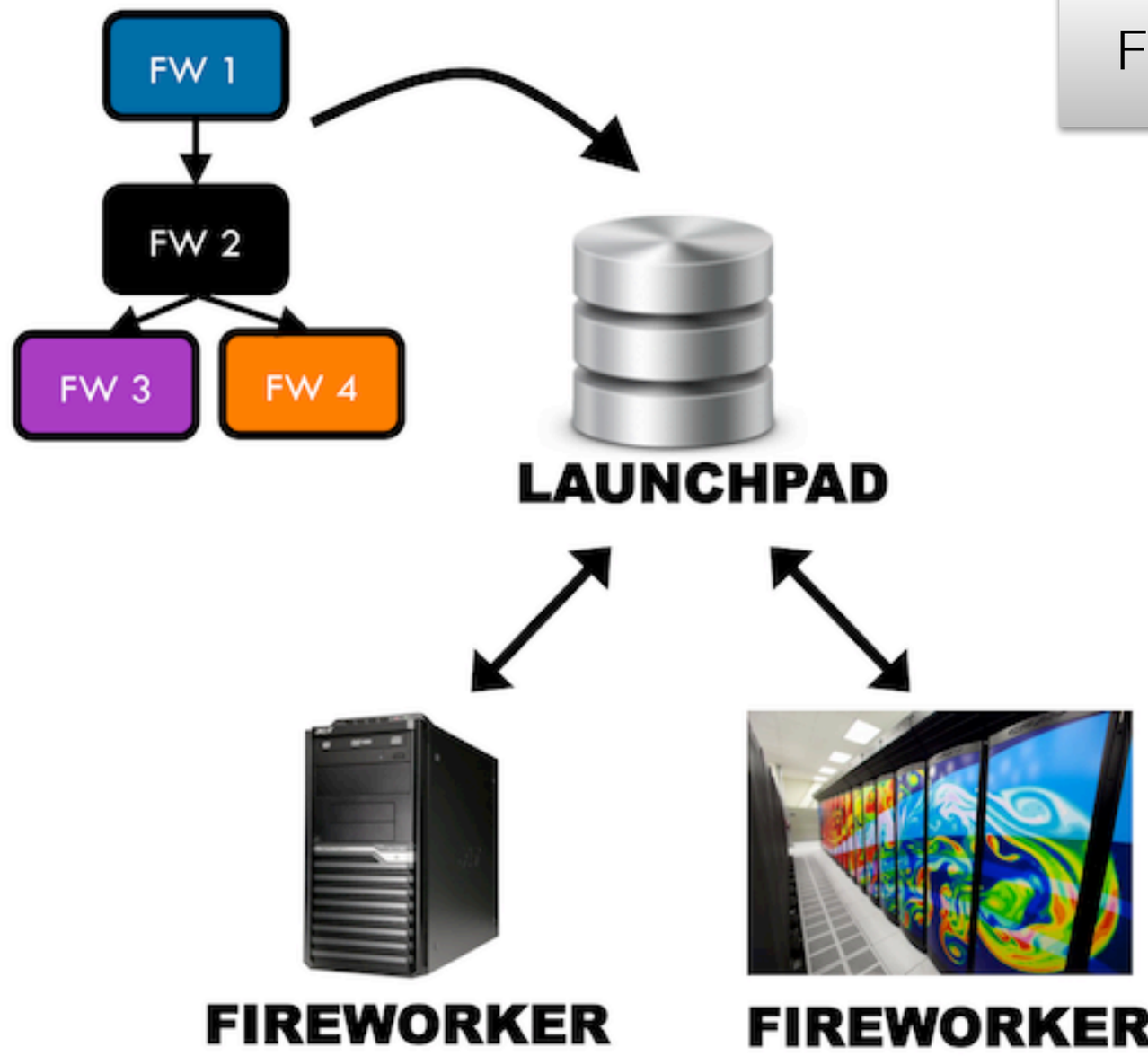
alchemlyb

Requirements for the FEP workflow

- utilize heterogenous resources (XSEDE, local clusters, workstations)
- fault tolerant and re-runnable
- runs automatically until prescribed run time has been simulated

FireWorks

work units = directed acyclic graph (DAG)



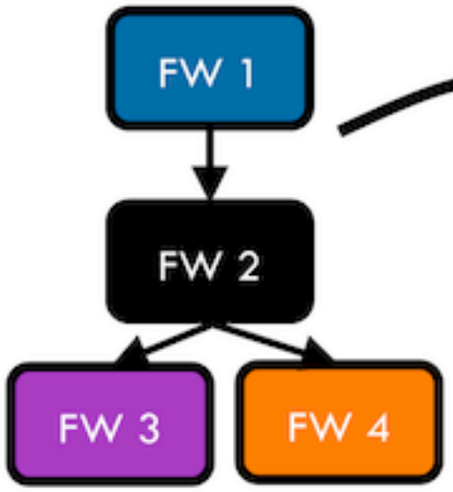
https://materialsproject.github.io/fireworks/

FireWorks

https://materialsproject.github.io/fireworks/

FireWorks

work units = directed acyclic graph (DAG)

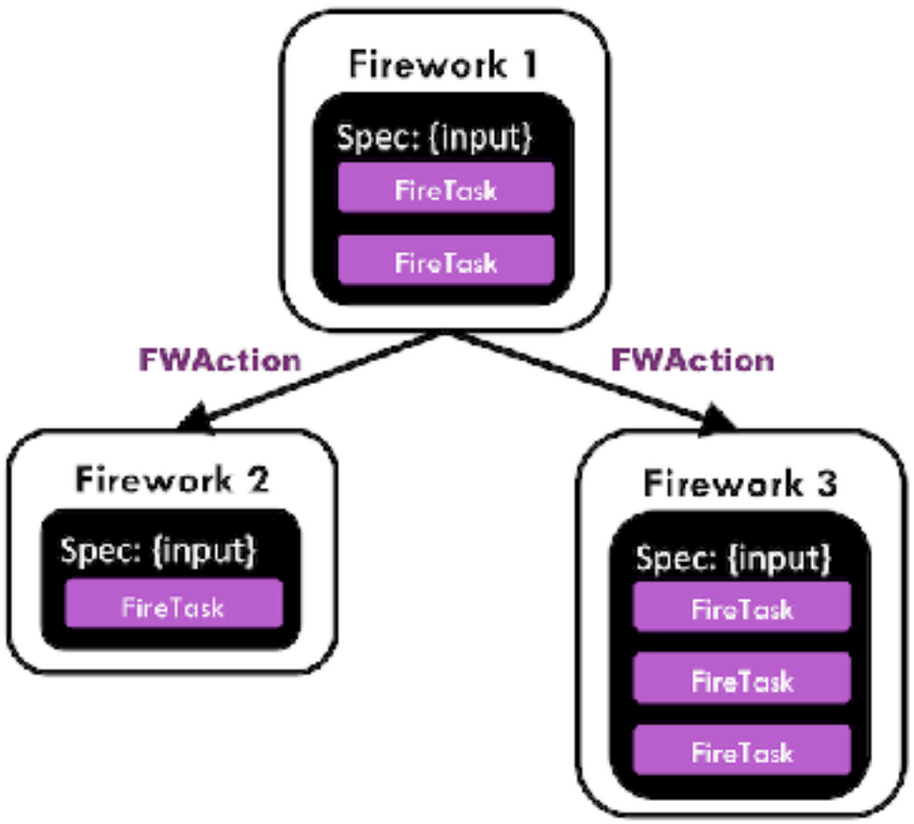


LAUNCHPAD

FIREWORKER

FIREWORKER

https://materialsproject.github.io/fireworks/



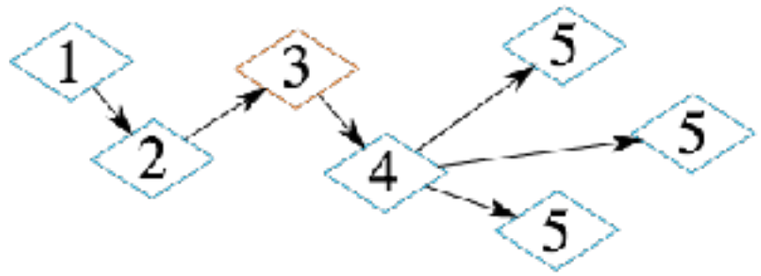
https://materialsproject.github.io/fireworks/

FireWorks

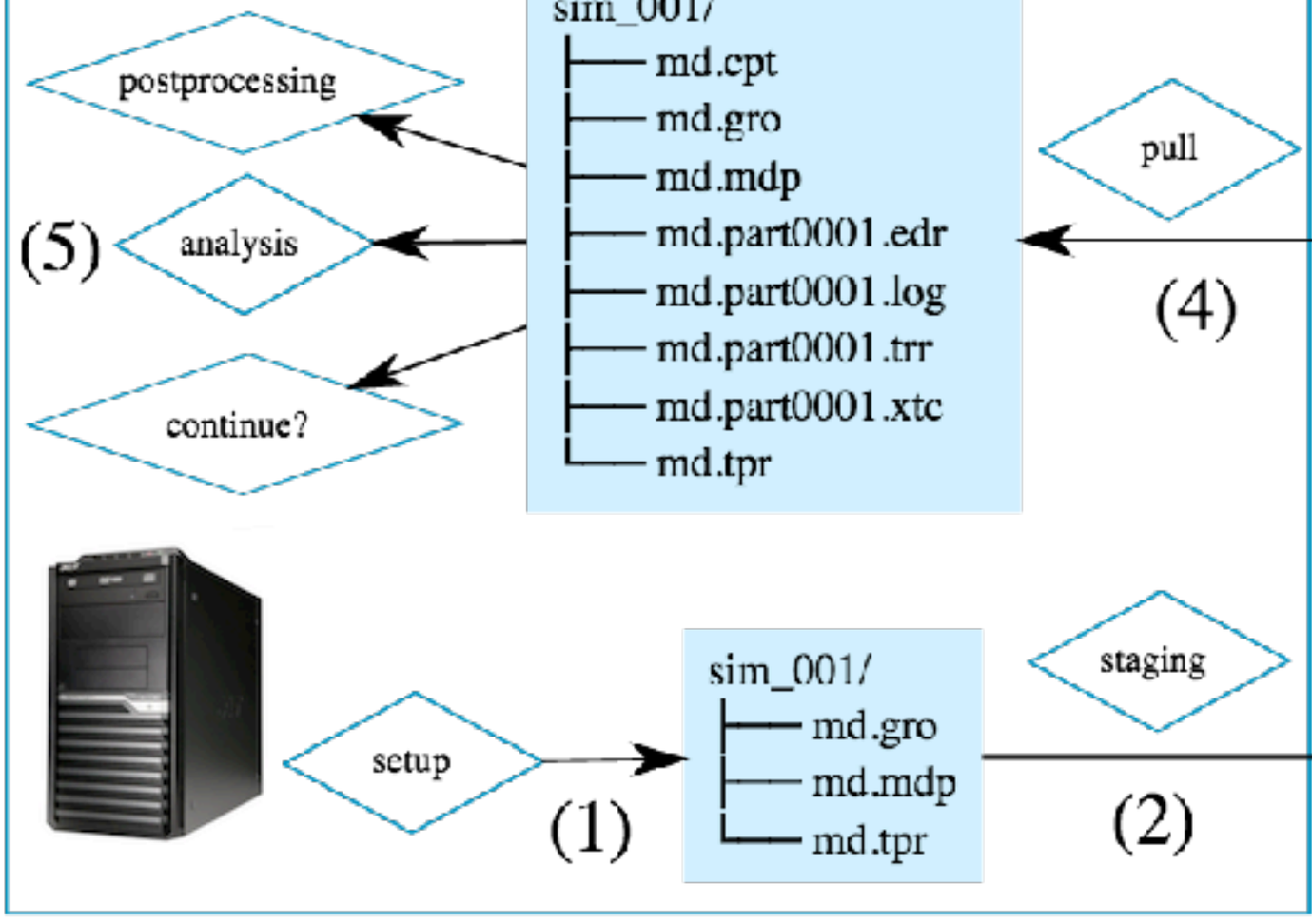
https://materialsproject.github.io/fireworks/

Molecular dynamics/FEP workflow

Webserver

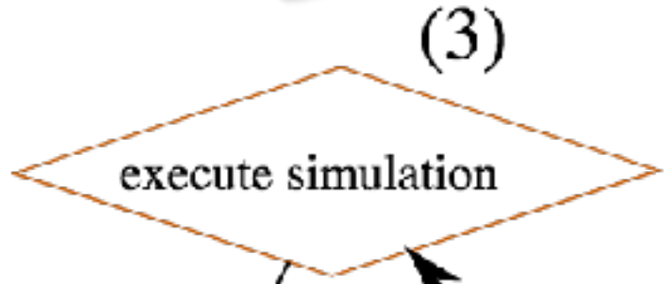


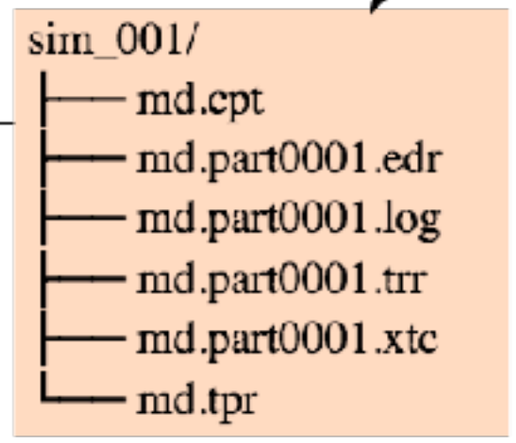
Fileserver

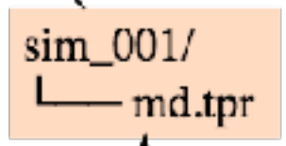


Stampede

Compute Node







Filesystem : Lustre

Login Node

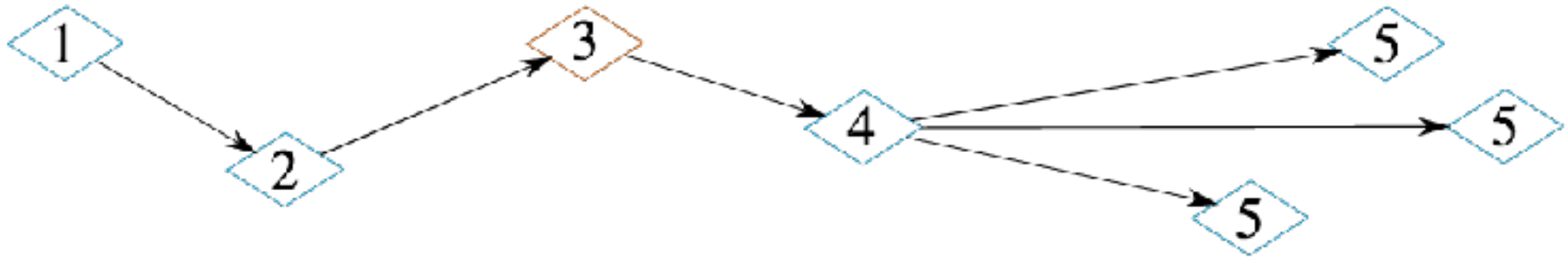
(2)

Webserver

Database : MongoDB

Launchpad

Workflow



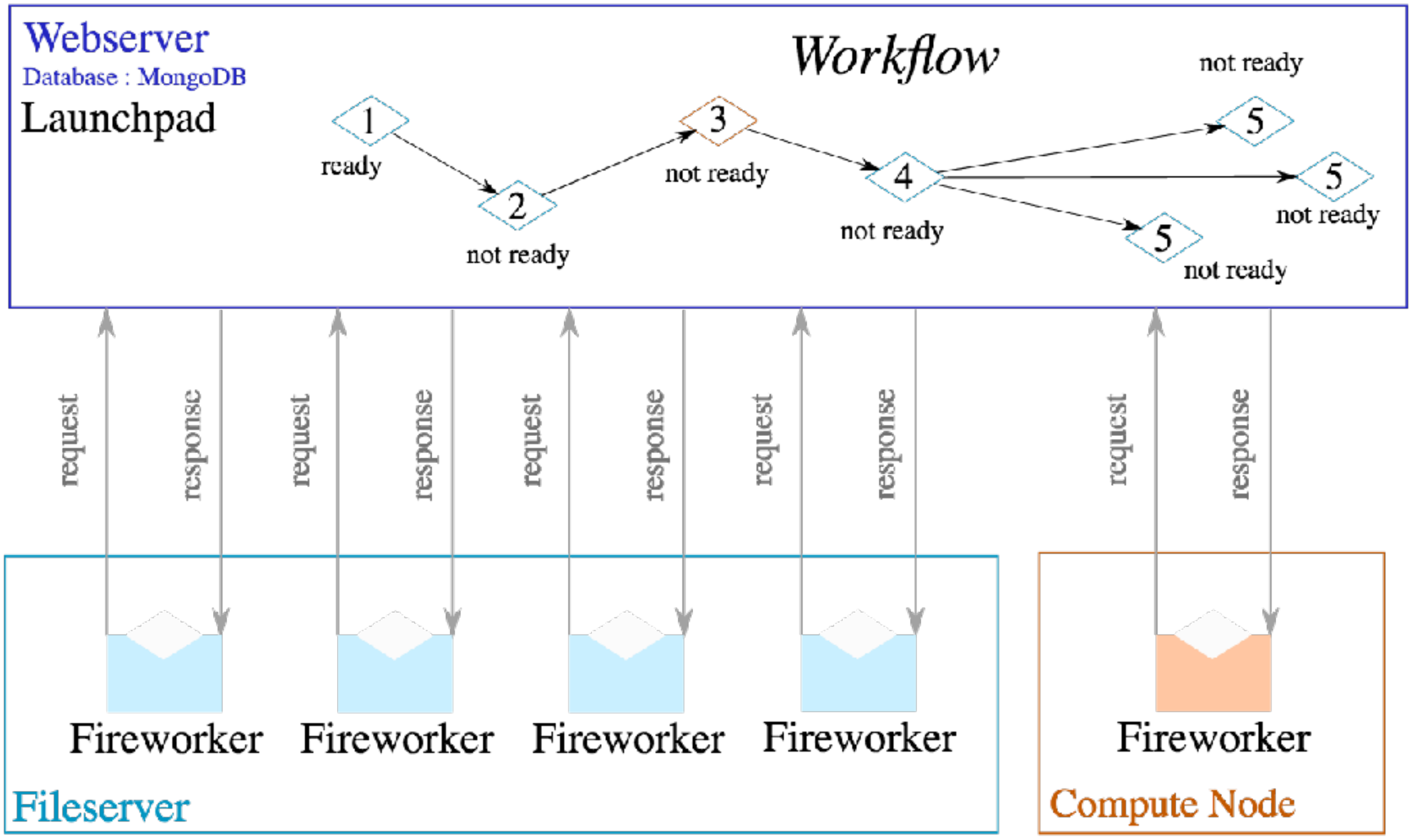
request
response
request
response
request
response
request
response
request
response

Fireworker
Fireworker
Fireworker
Fireworker

Fileserver

Fireworker

Compute Node

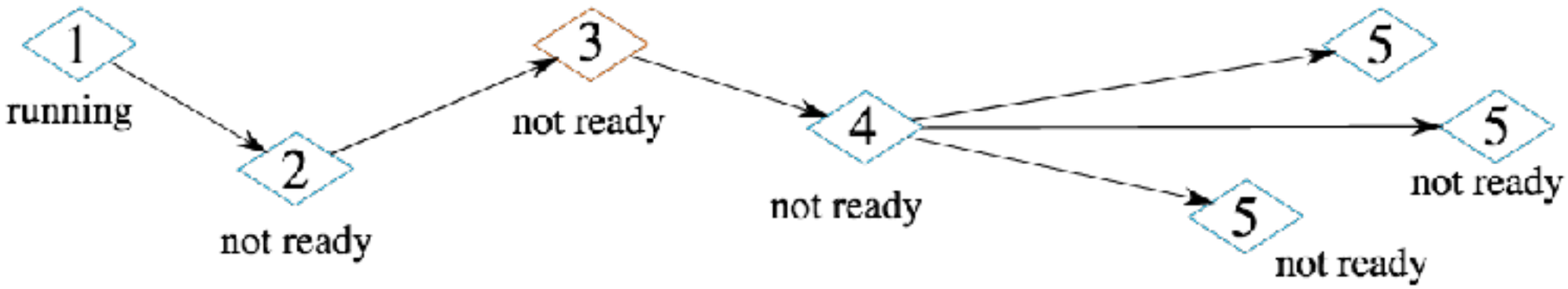

Webserver
Database : MongoDB
Launchpad
Workflow
1
ready
2
not ready
3
not ready
4
not ready
5
not ready
5
not ready
5
not ready
request
response
request
response
request
response
request
response
request
response
Fireworker
Fireworker
Fireworker
Fireworker
Fileserver
Fireworker
Compute Node

Webserver

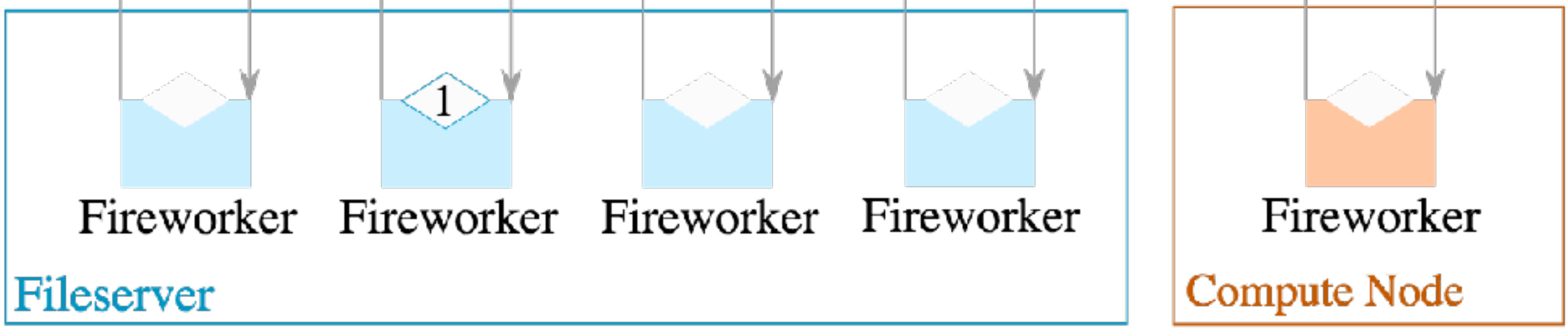
Database : MongoDB

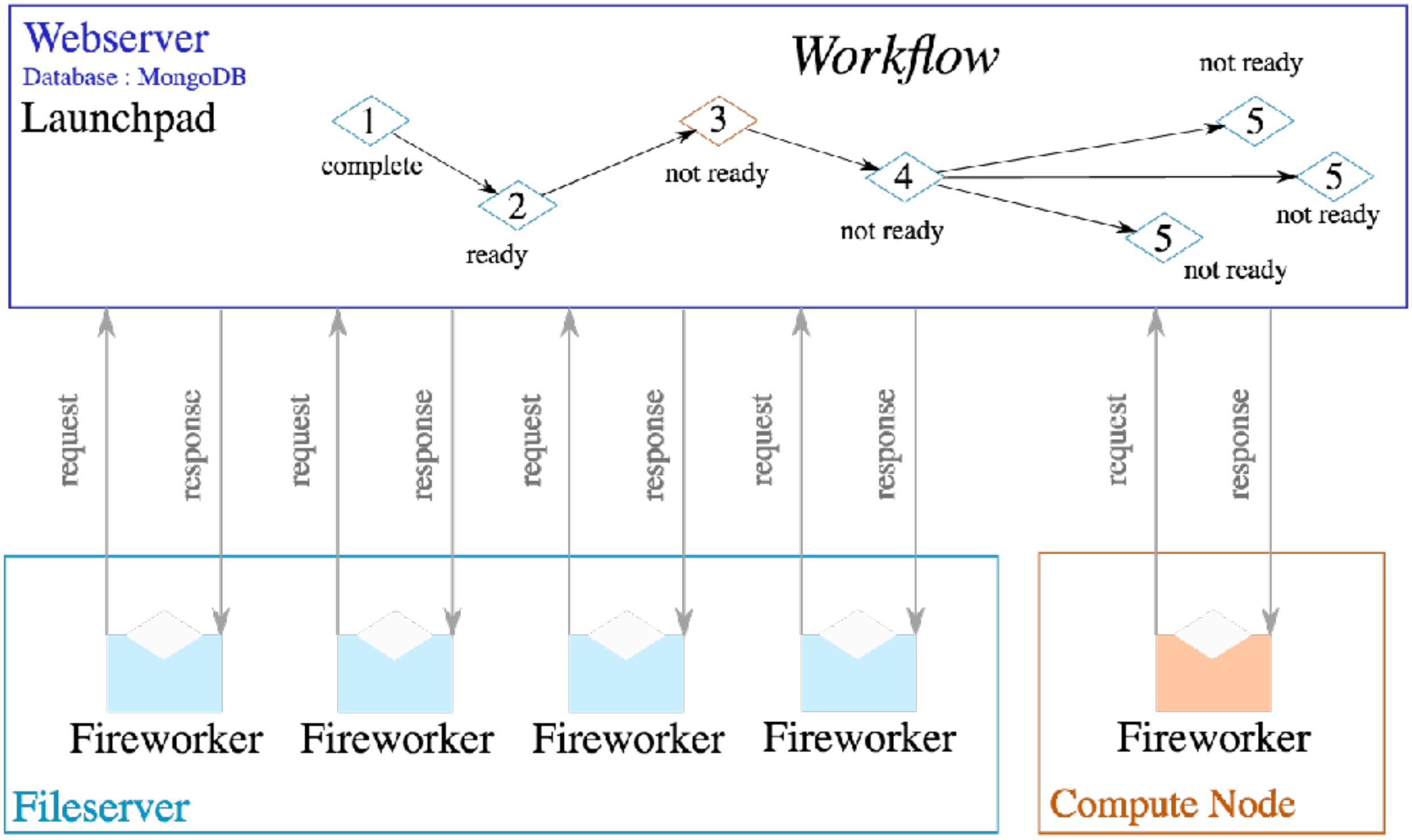
Launchpad

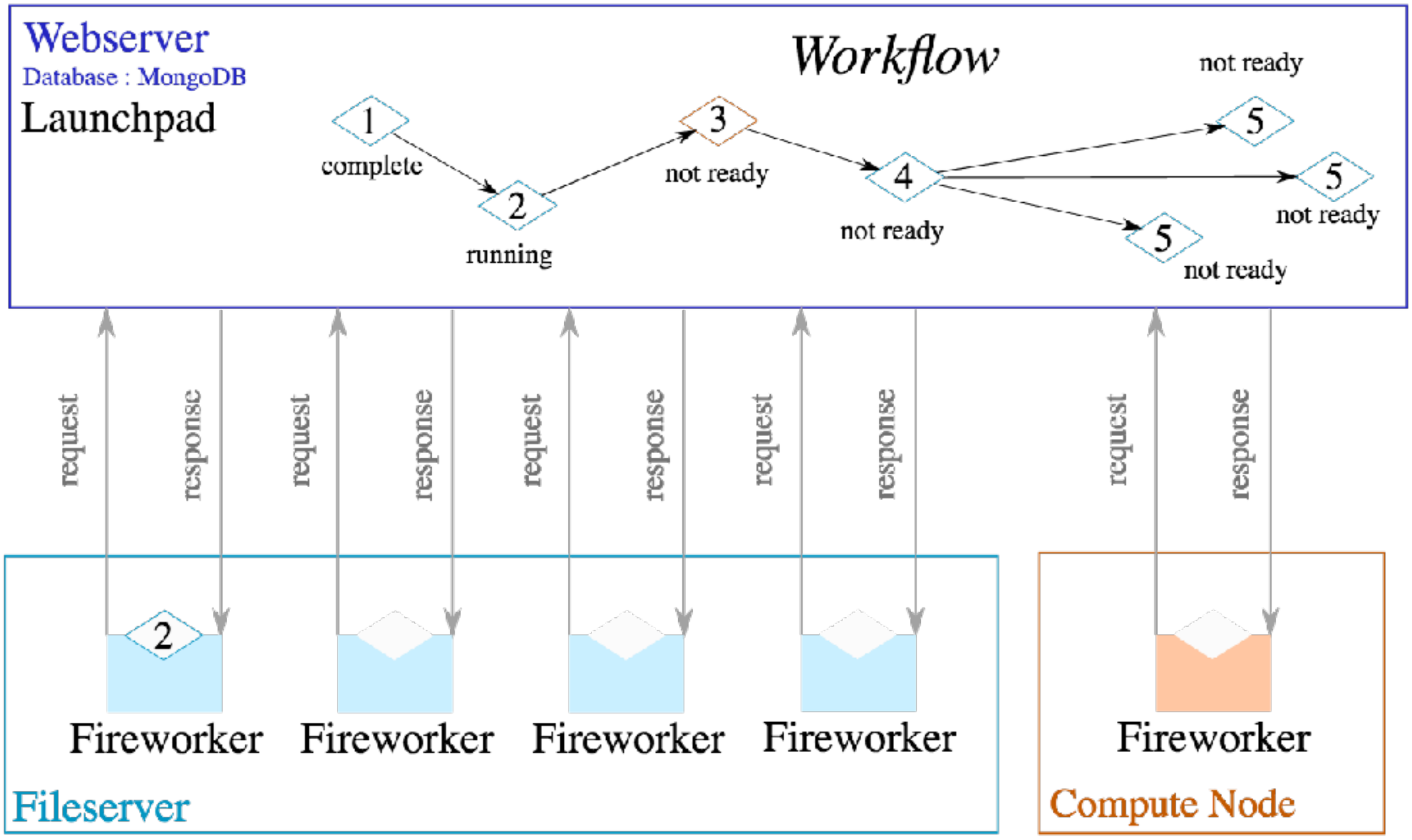
Workflow

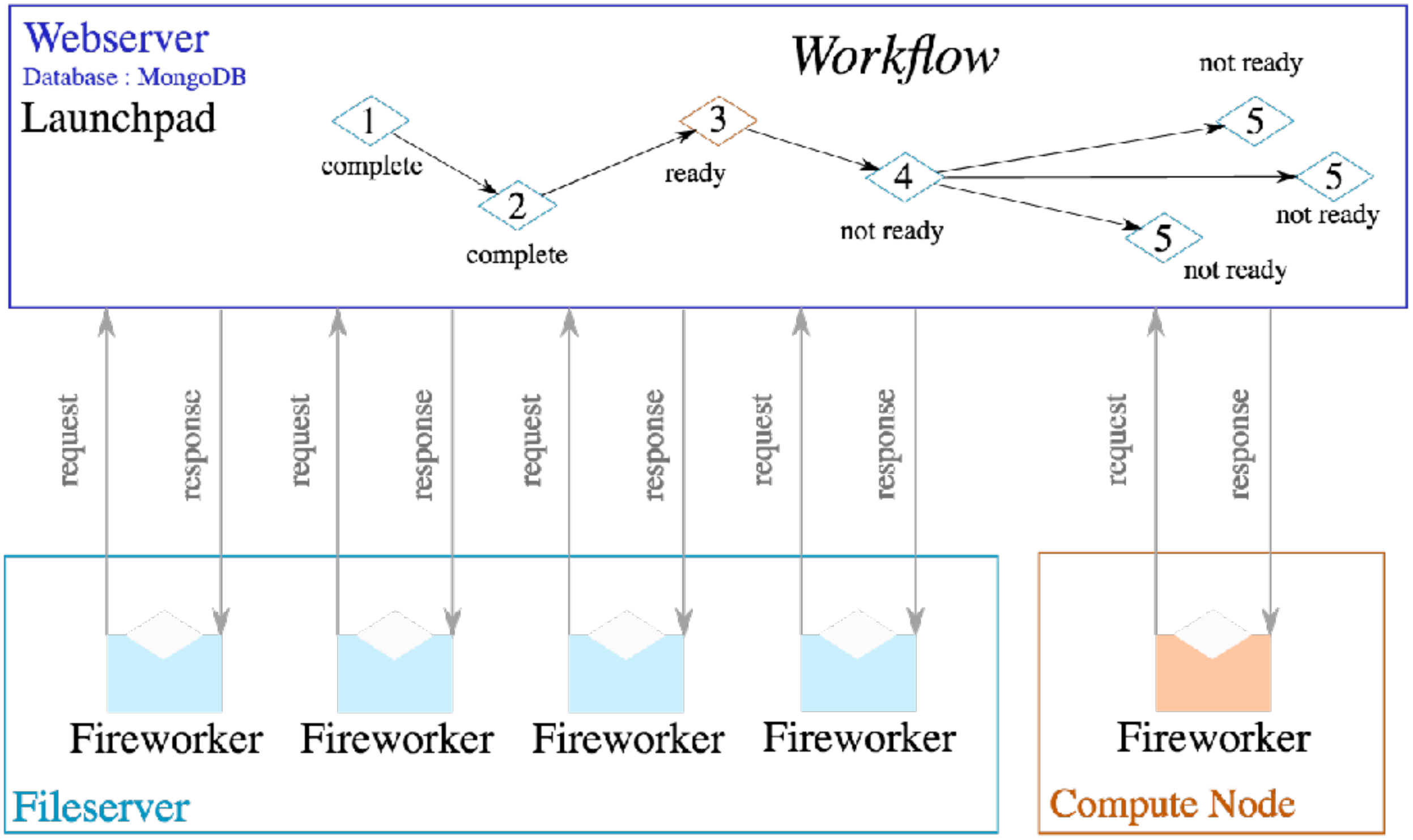


request response request response request response request response request response

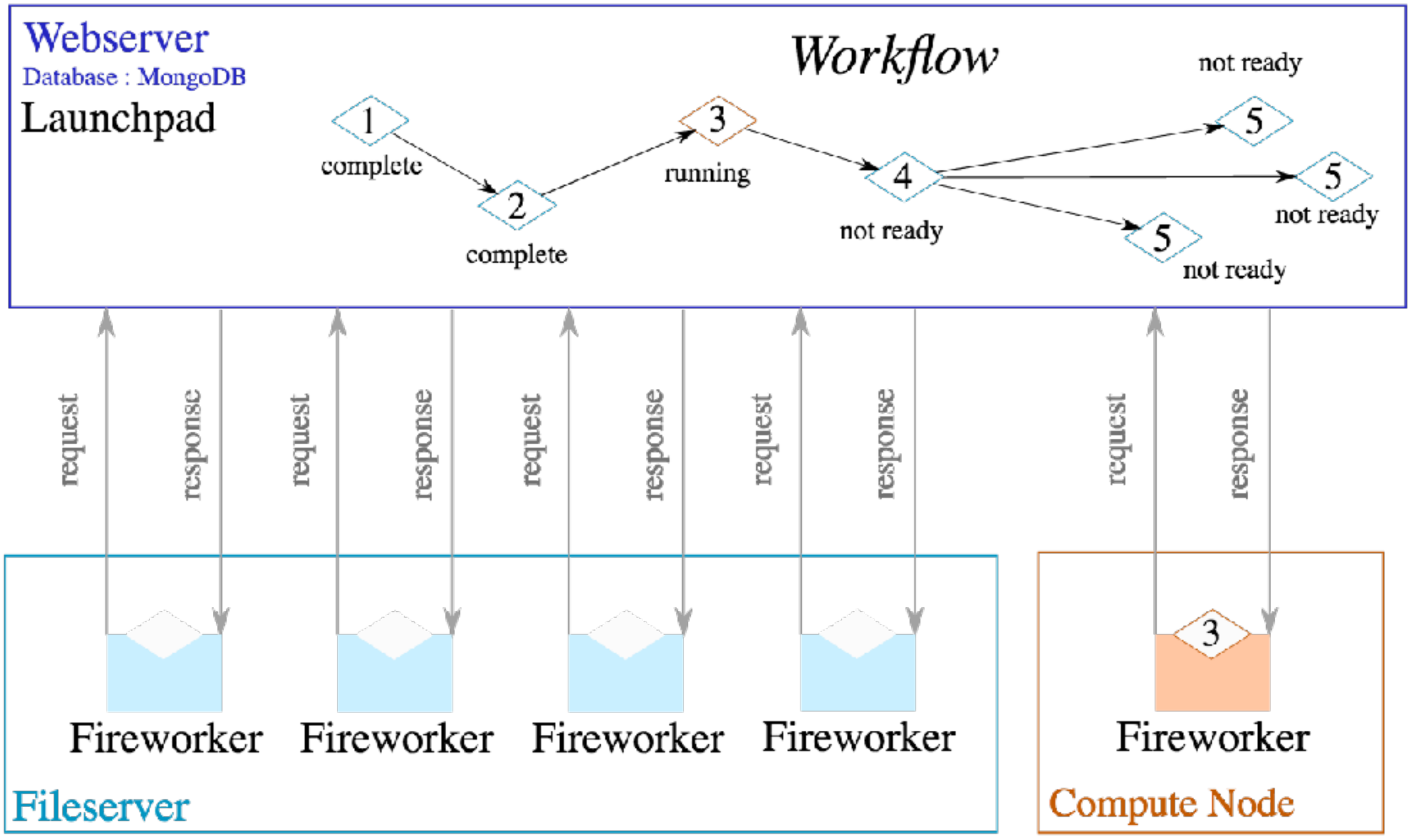



Webserver
Database : MongoDB
Launchpad
Workflow
1
complete
2
ready
3
not ready
4
not ready
5
not ready
5
not ready
5
not ready
request
response
request
response
request
response
request
response
request
response
Fireworker
Fireworker
Fireworker
Fireworker
Fileserver
Fireworker
Compute Node

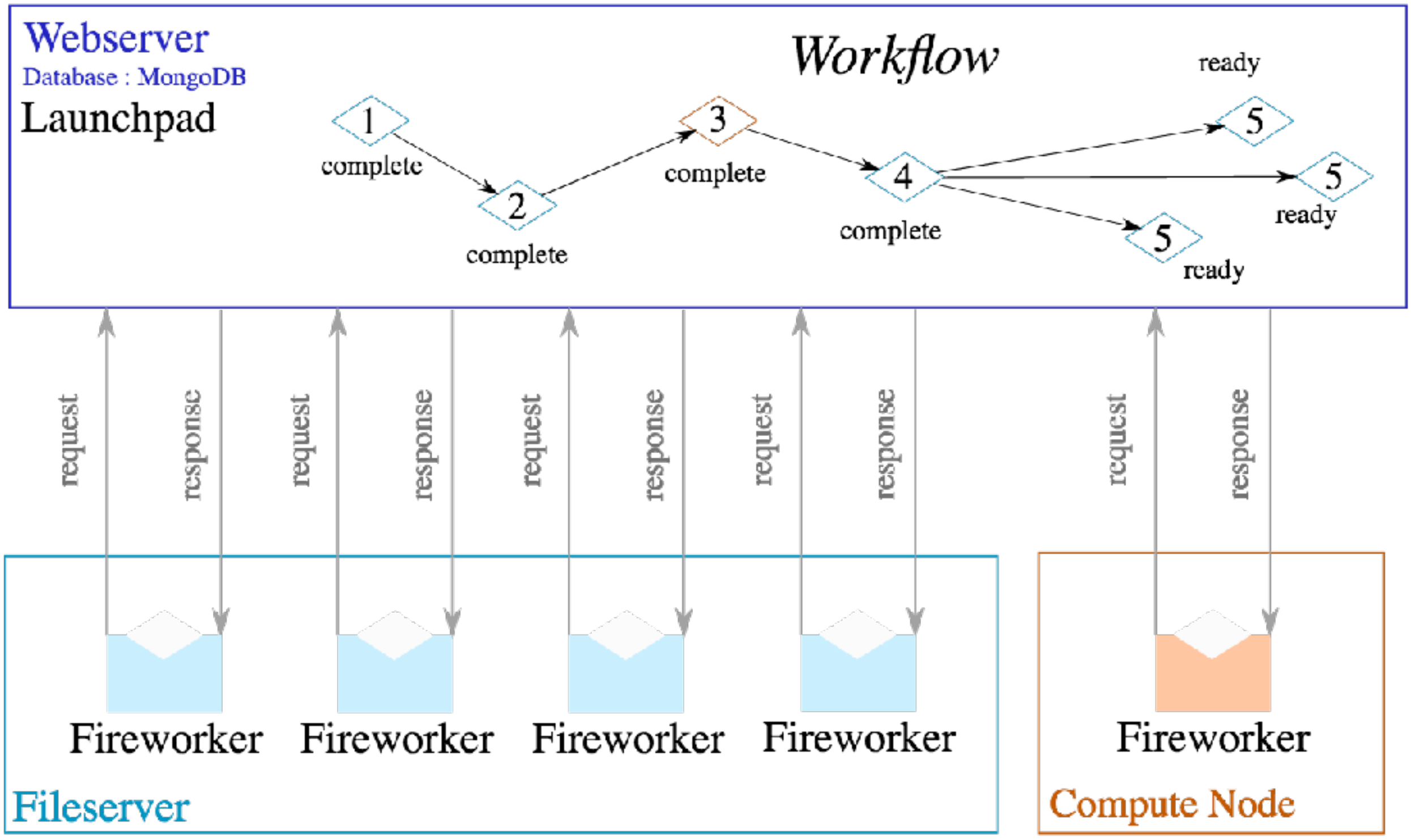

Webserver
Database : MongoDB
Launchpad
Workflow
1
complete
2
running
3
not ready
4
not ready
5
not ready
5
not ready
5
not ready
request
response
request
response
request
response
request
response
request
response
2
Fireworker
Fireworker
Fireworker
Fireworker
Fileserver
Fireworker
Compute Node

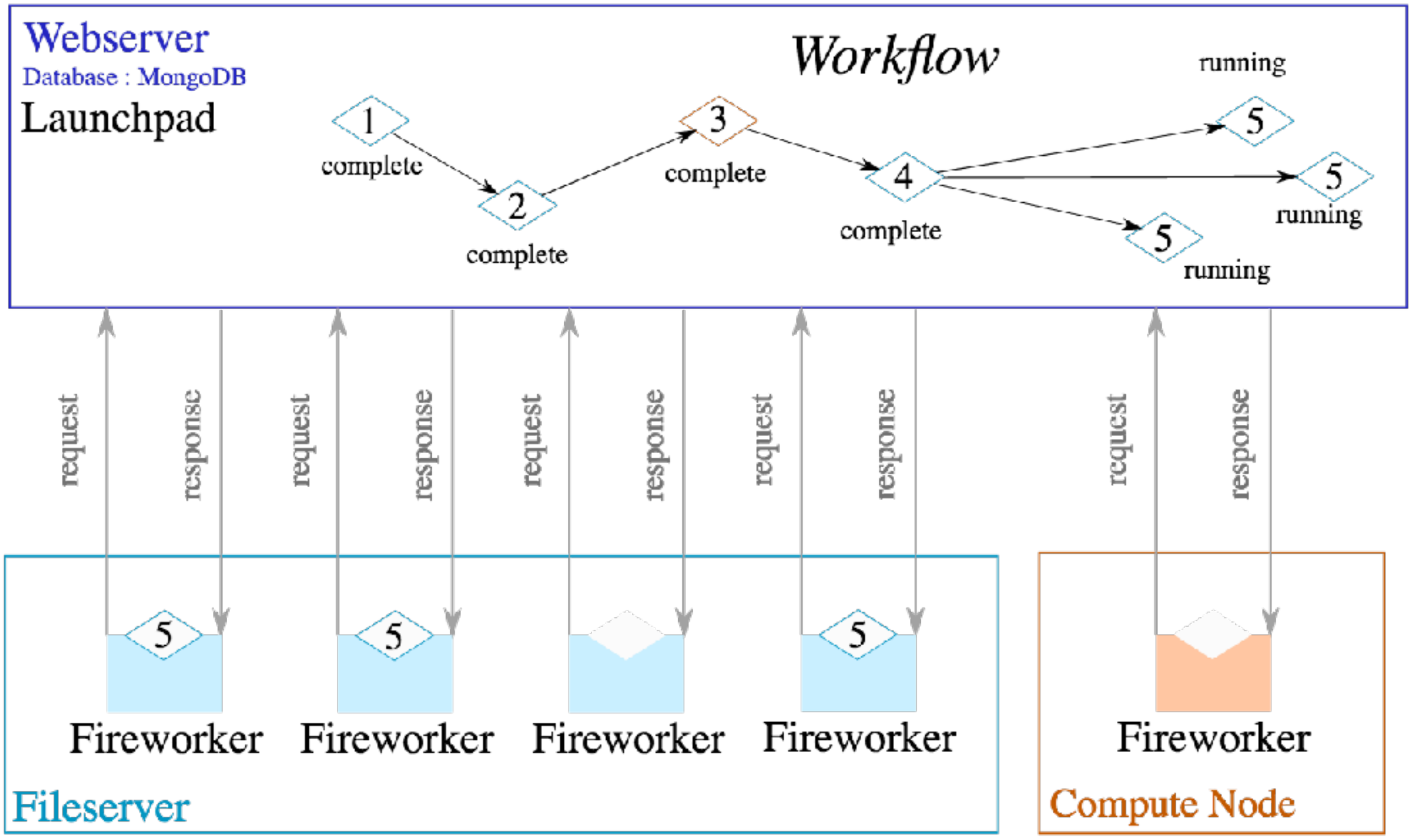

Webserver
Database : MongoDB
Launchpad
Workflow
1
complete
2
complete
3
ready
4
not ready
5
not ready
5
not ready
5
not ready
request
response
request
response
request
response
request
response
request
response
Fireworker
Fireworker
Fireworker
Fireworker
Fileserver
Fireworker
Compute Node

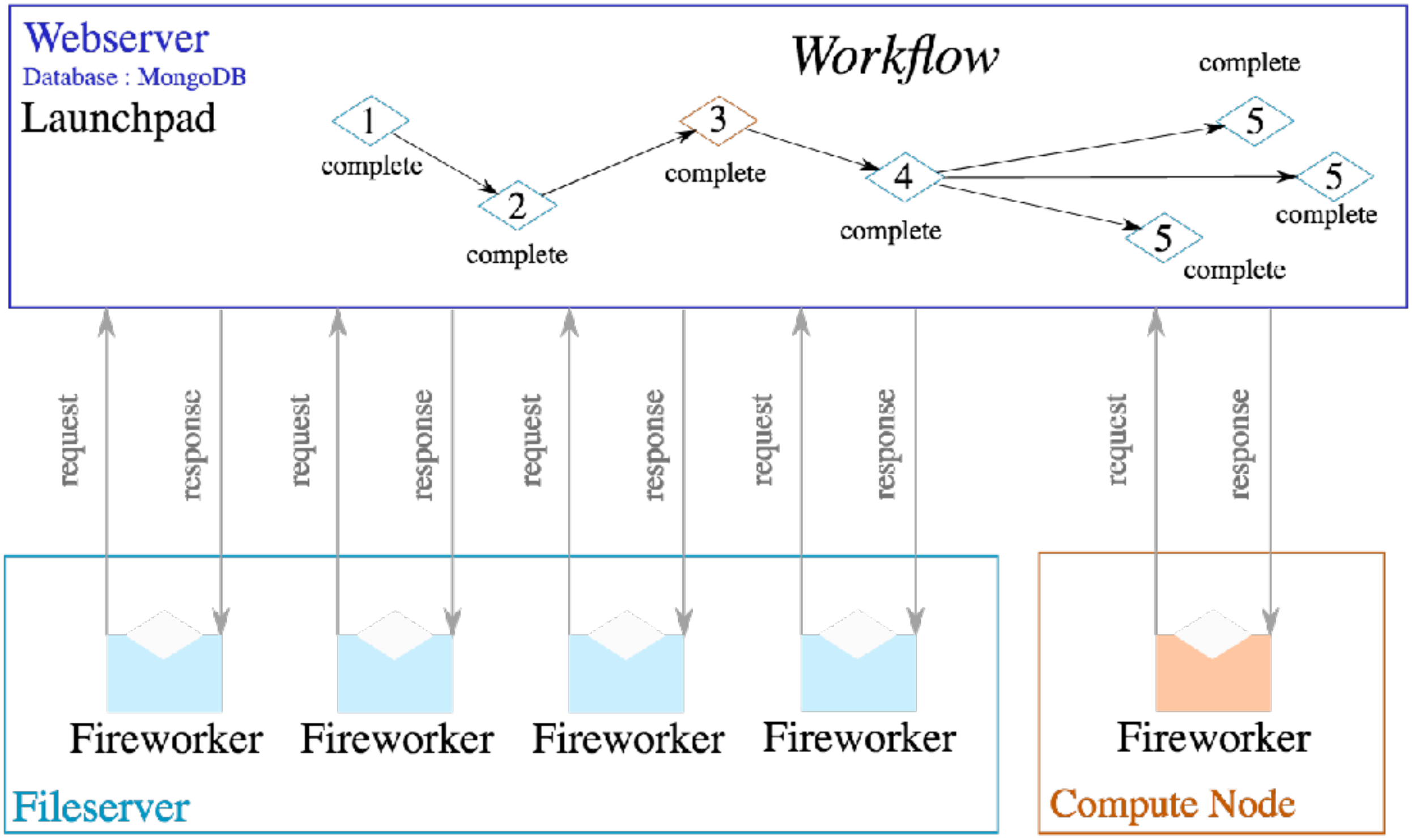
execution categories



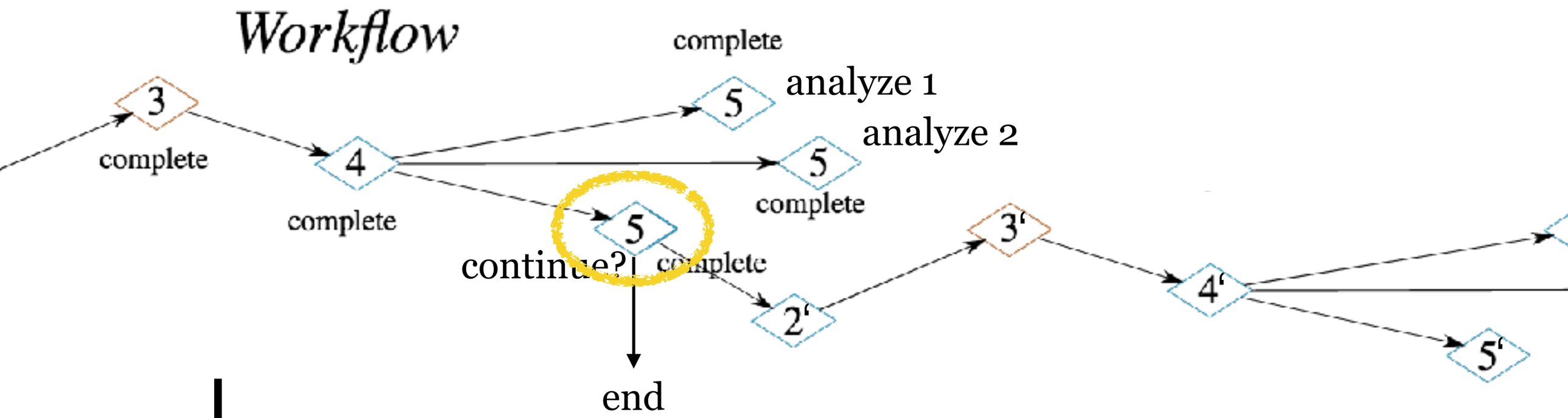
parallelism

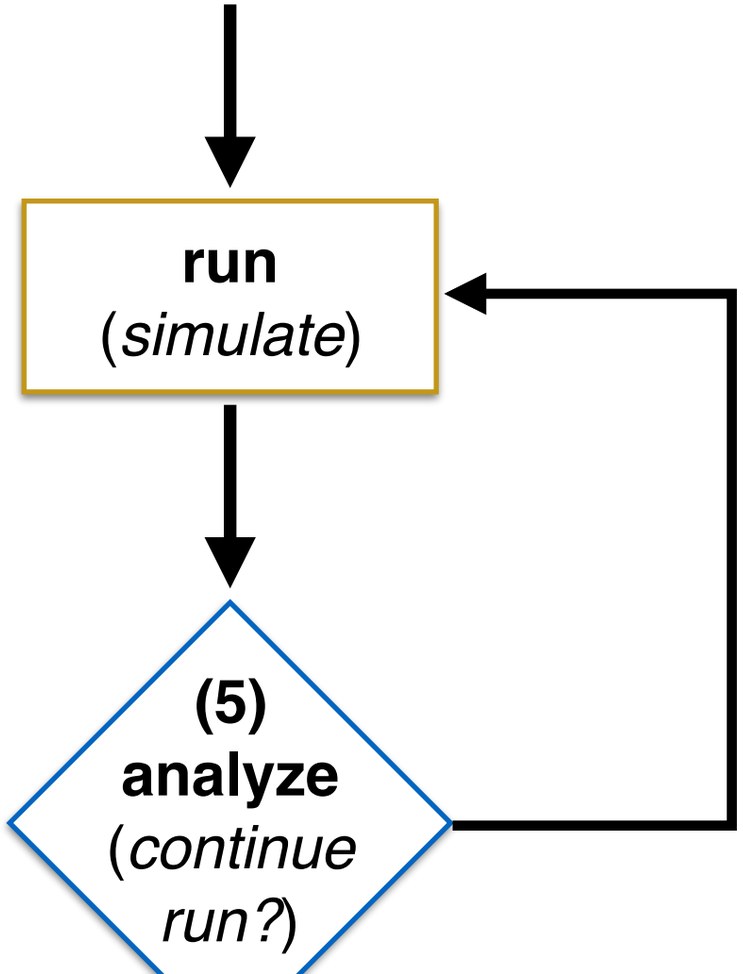



Webserver
Database : MongoDB
Launchpad
Workflow
1
complete
2
complete
3
complete
4
complete
5
running
5
running
5
running
request
response
request
response
request
response
request
response
request
response
5
5
5
Fireworker
Fireworker
Fireworker
Fireworker
Fileserver
Fireworker
Compute Node

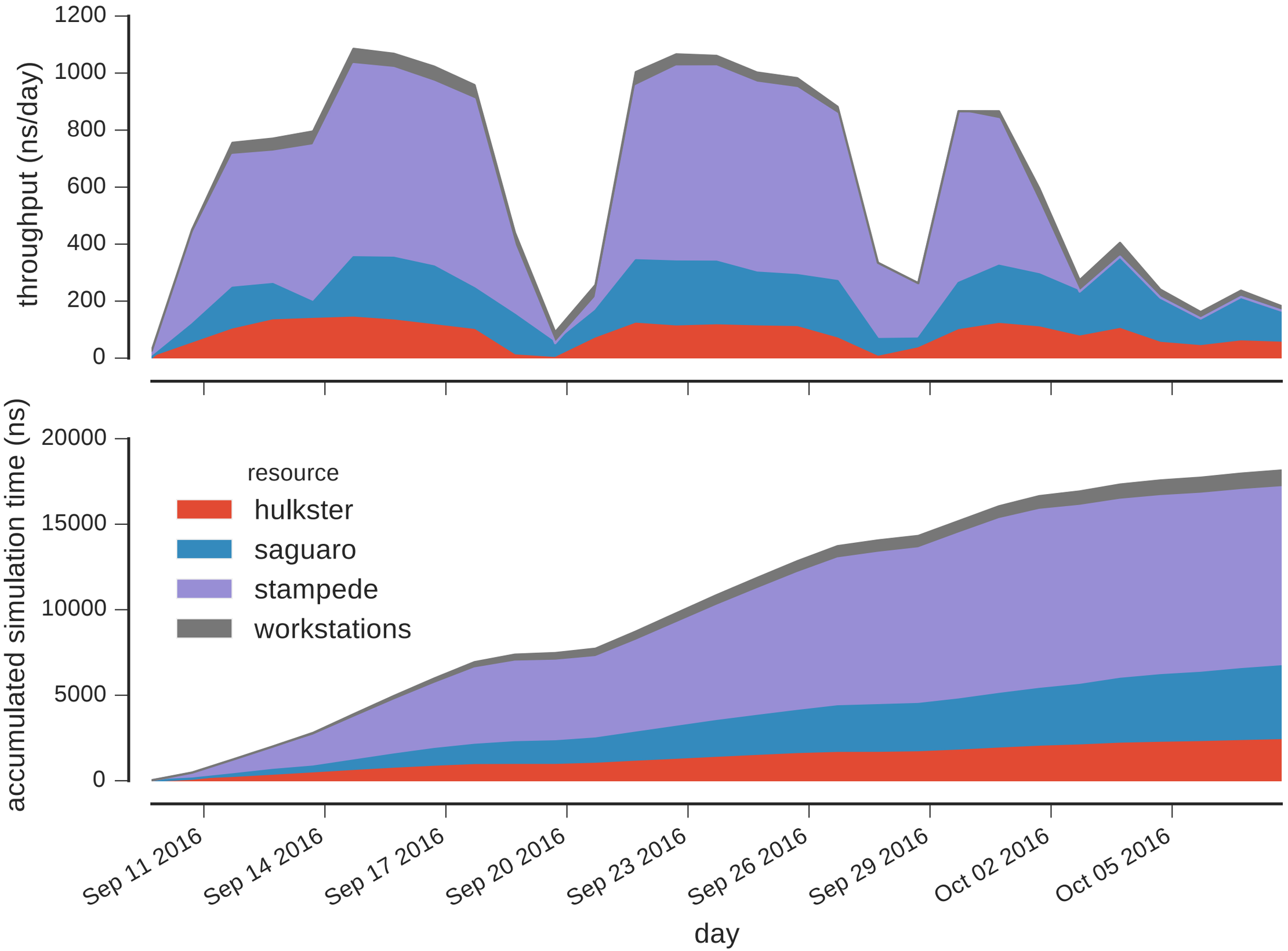

Webserver
Database : MongoDB
Launchpad
Workflow
1
complete
2
complete
3
complete
4
complete
5
complete
5
complete
5
complete
request
response
request
response
request
response
request
response
request
response
Fireworker
Fireworker
Fireworker
Fireworker
Fileserver
Fireworker
Compute Node

self-modifying tasks: (5) can add new nodes to the DAG



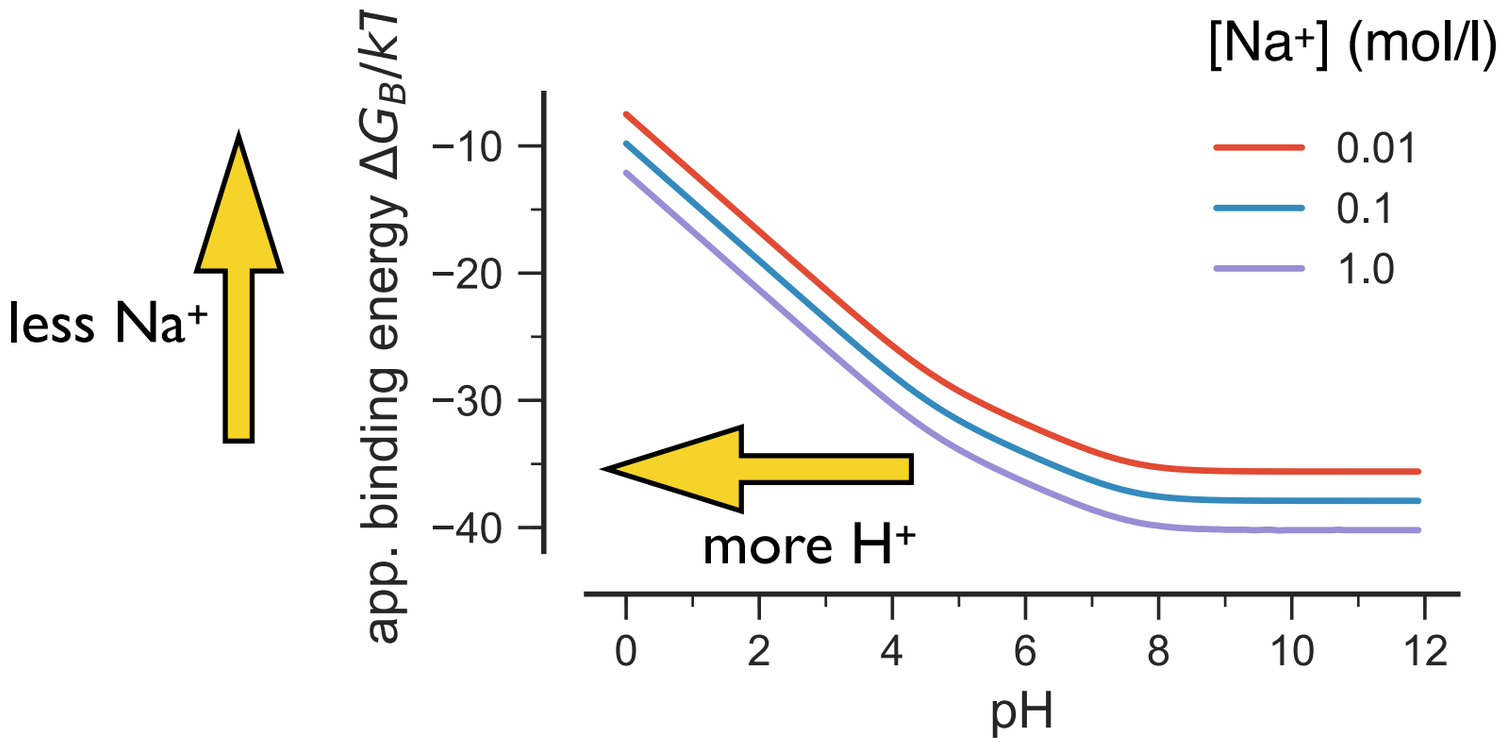



```
class GromacsContinueTask(FireTaskBase):
    def run_task(self, fw_spec):
        ...
        if step < nsteps:
            wf = make_md_workflow(...)
        else:
            wf = post_wf  # finalize
        return FWAction(additions=[wf])
```


throughput (ns/day)
1200
1000
800
600
400
200
0
accumulated simulation time (ns)
20000
15000
10000
5000
0
resource
hulkster
saguaro
stampede
workstations
Sep 11 2016
Sep 14 2016
Sep 17 2016
Sep 20 2016
Sep 23 2016
Sep 26 2016
Sep 29 2016
Oct 02 2016
Oct 05 2016
day

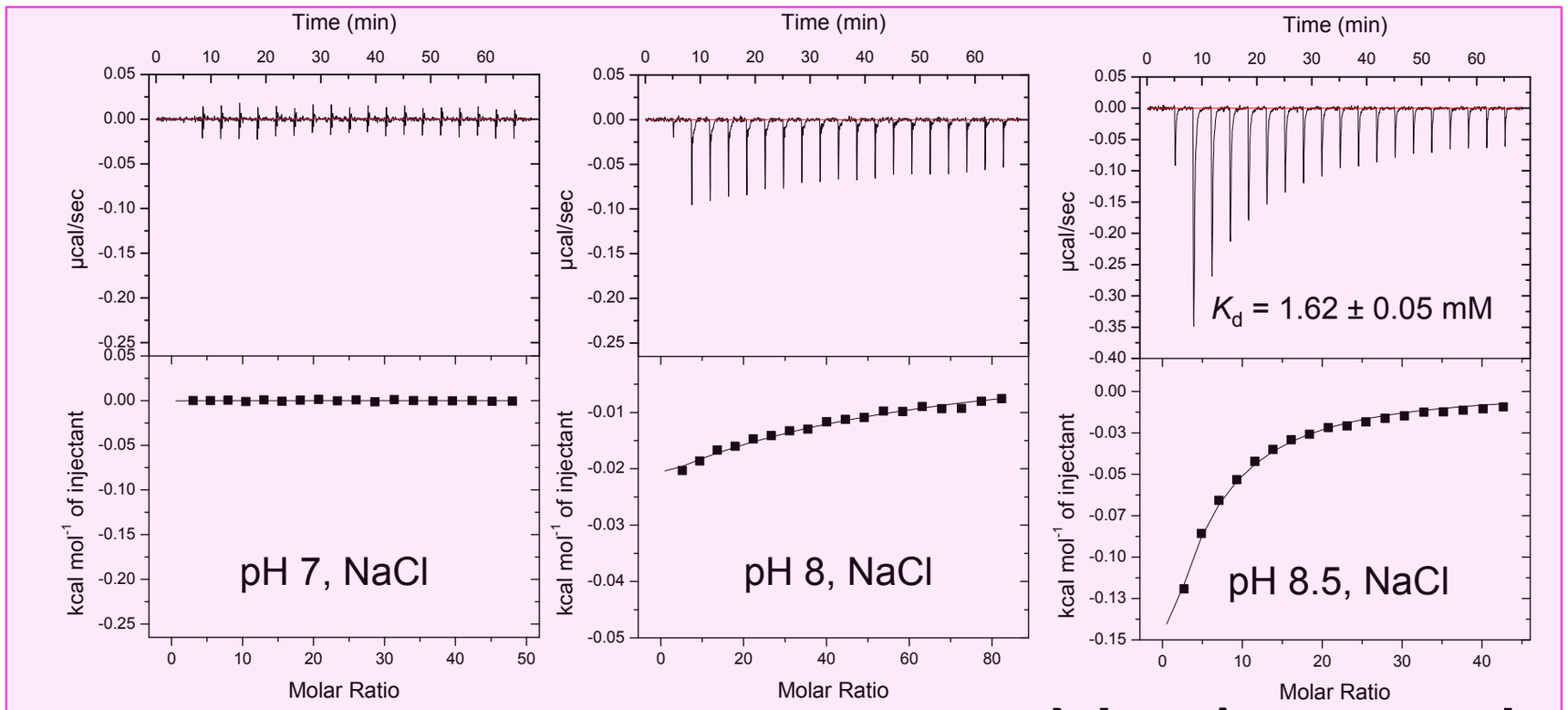
NapA: Competitive binding (qualitatively)

(preliminary and unpublished)



- ~20 µs
- ~0.4 M CPU-h
- 1 binding free energy

- so far: 4 binding free energies, 2 repeats
- ~160 µs
- ~3.2 M CPU-h

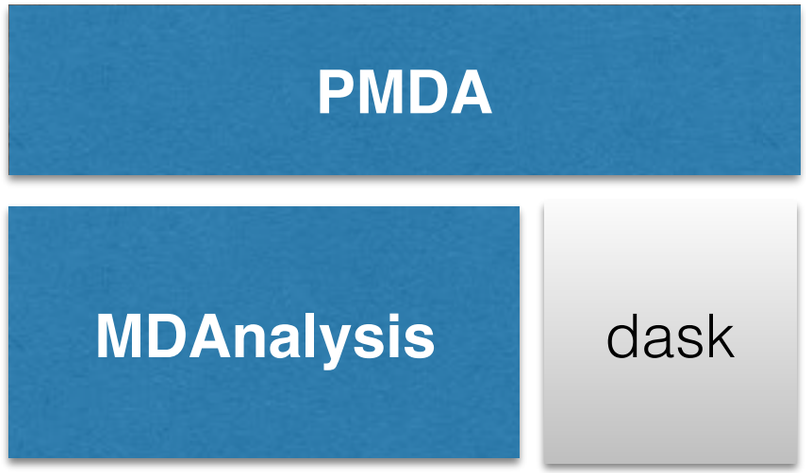


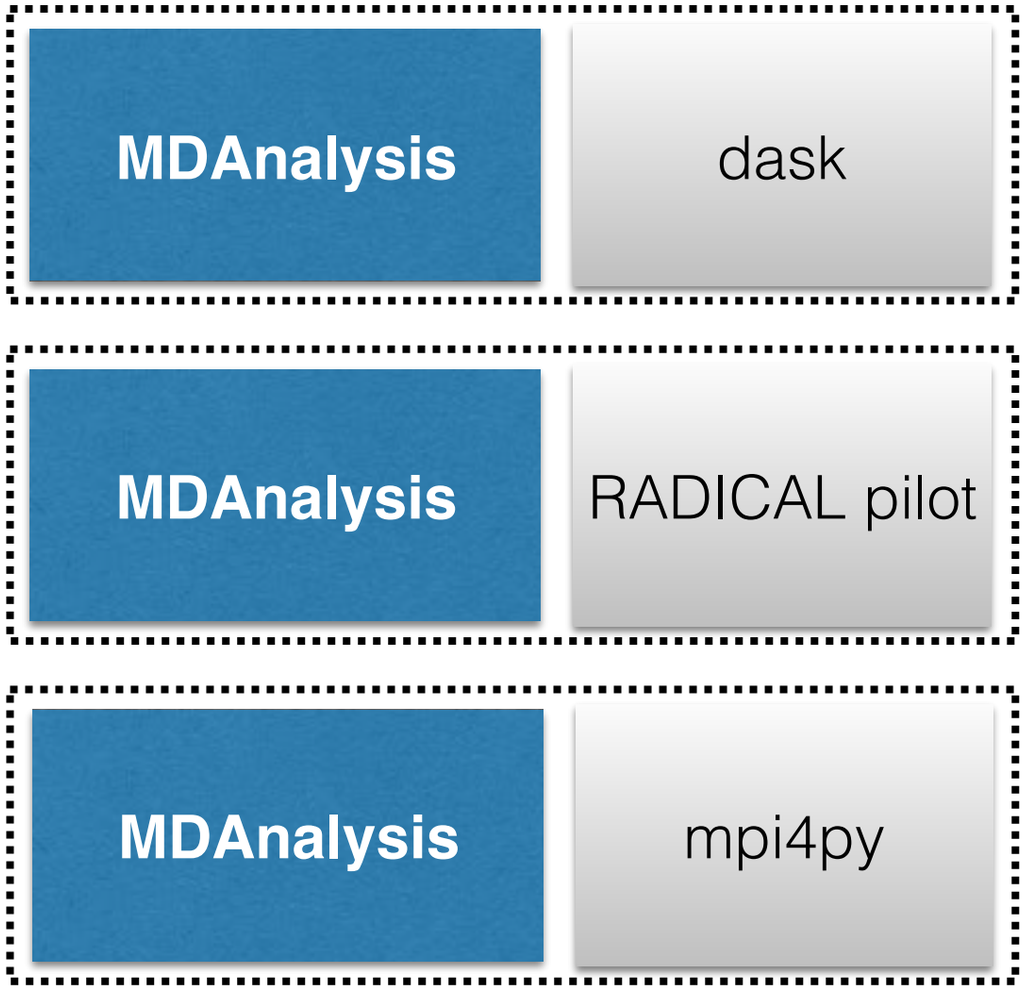
Nat Struct Mol Biol 23 (2016), 248

binding measured with ITC

NapA inward

Parallelizing trajectory analysis



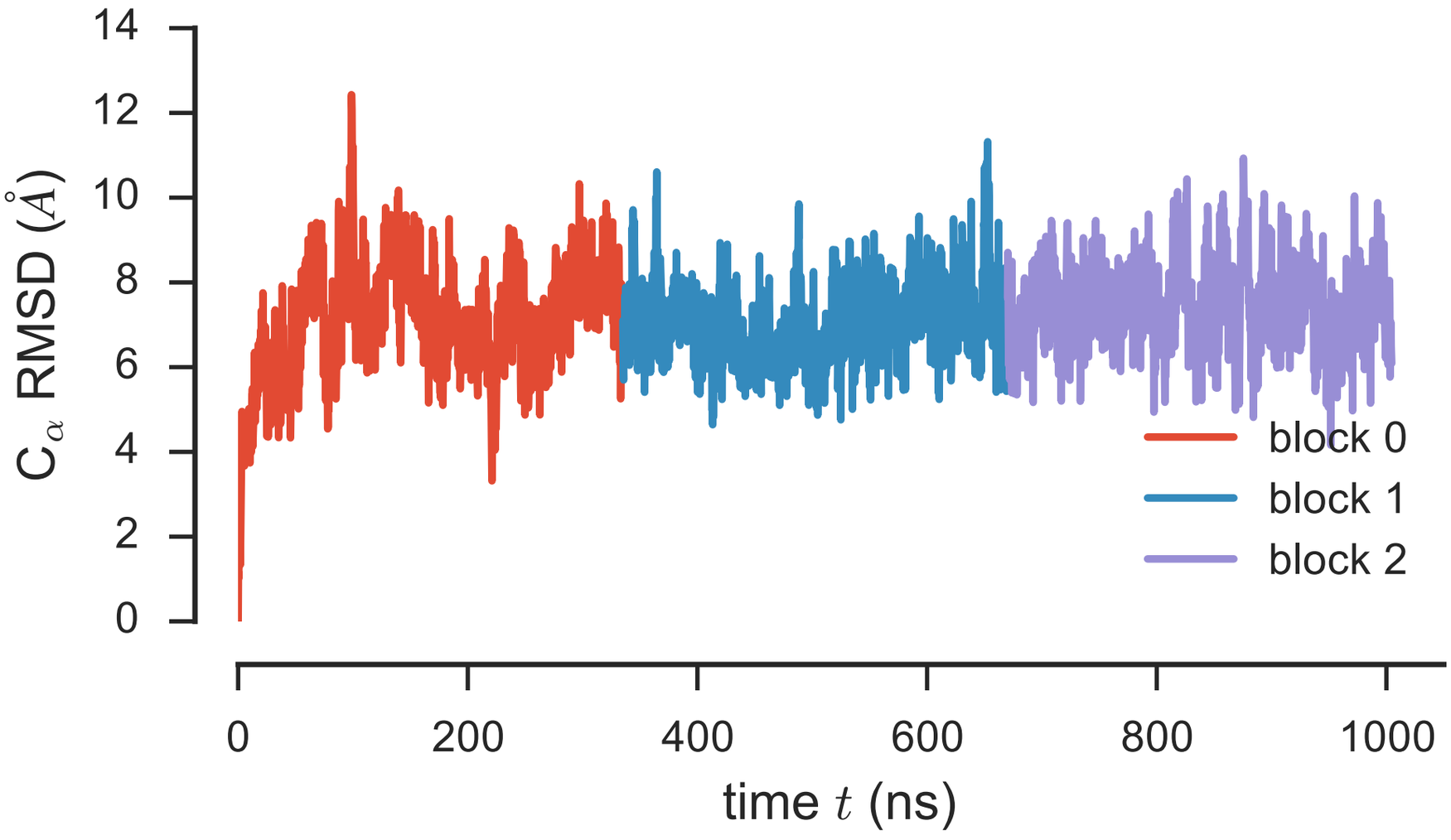


https://github.com/mnmelo/mdreader/

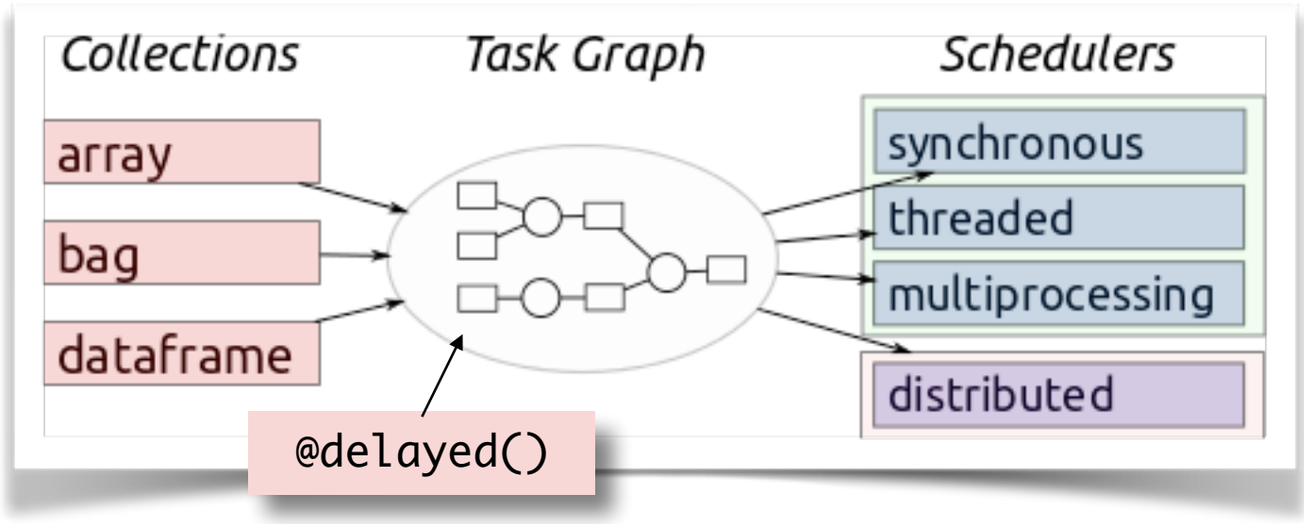
I. Paraskevakos et al. *Task-parallel analysis of molecular dynamics trajectories.* In ICPP 2018: 47th International Conference on Parallel Processing, August 13–16, 2018, Eugene, OR, USA, Article No. 49, New York, NY, USA, August 13–16 2018. Association for Computing Machinery, ACM.

M. Khoshlessan, et al. *Parallel analysis in MDAnalysis using the Dask parallel computing library*. In Katy Huff, David Lippa, Dillon Niederhut, and M. Pacer, editors, Proceedings of the 16th Python in Science Conference, pages 64–72, Austin, TX, 2017. SciPy.

Parallel analysis: Trajectory slicing

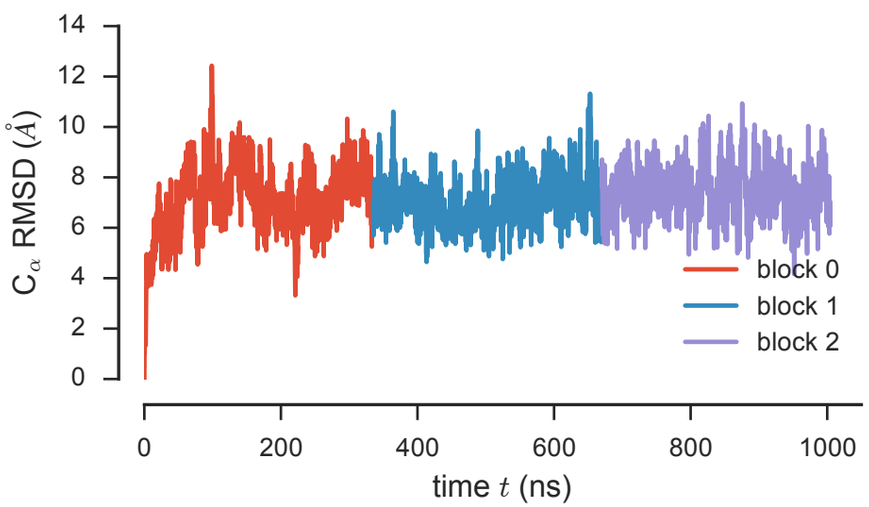


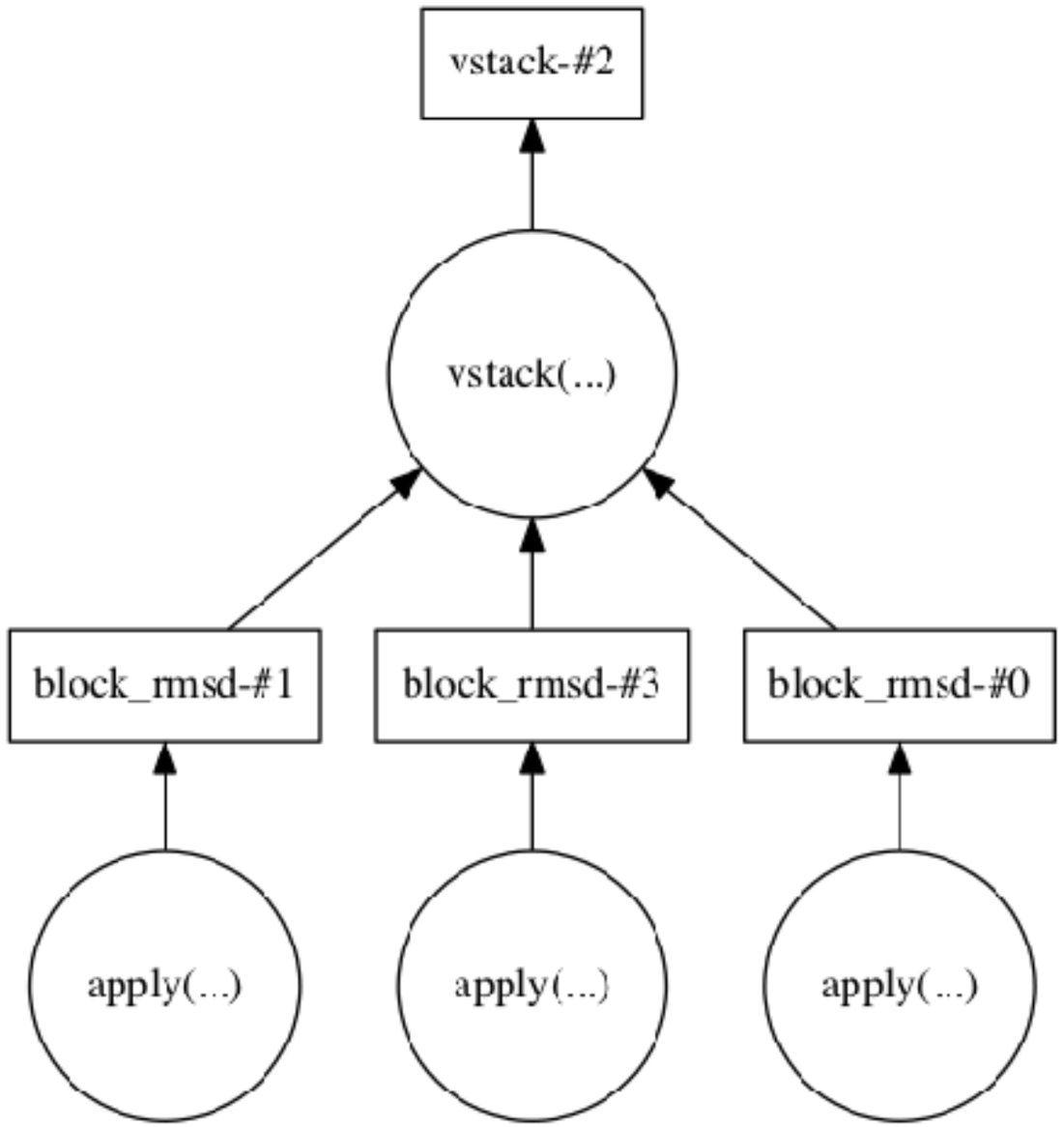
- MDAnalysis
- dask http://**dask**.org/



PMDA

https://www.mdanalysis.org/pmda/





PMDA uses dask/distributed to spread work

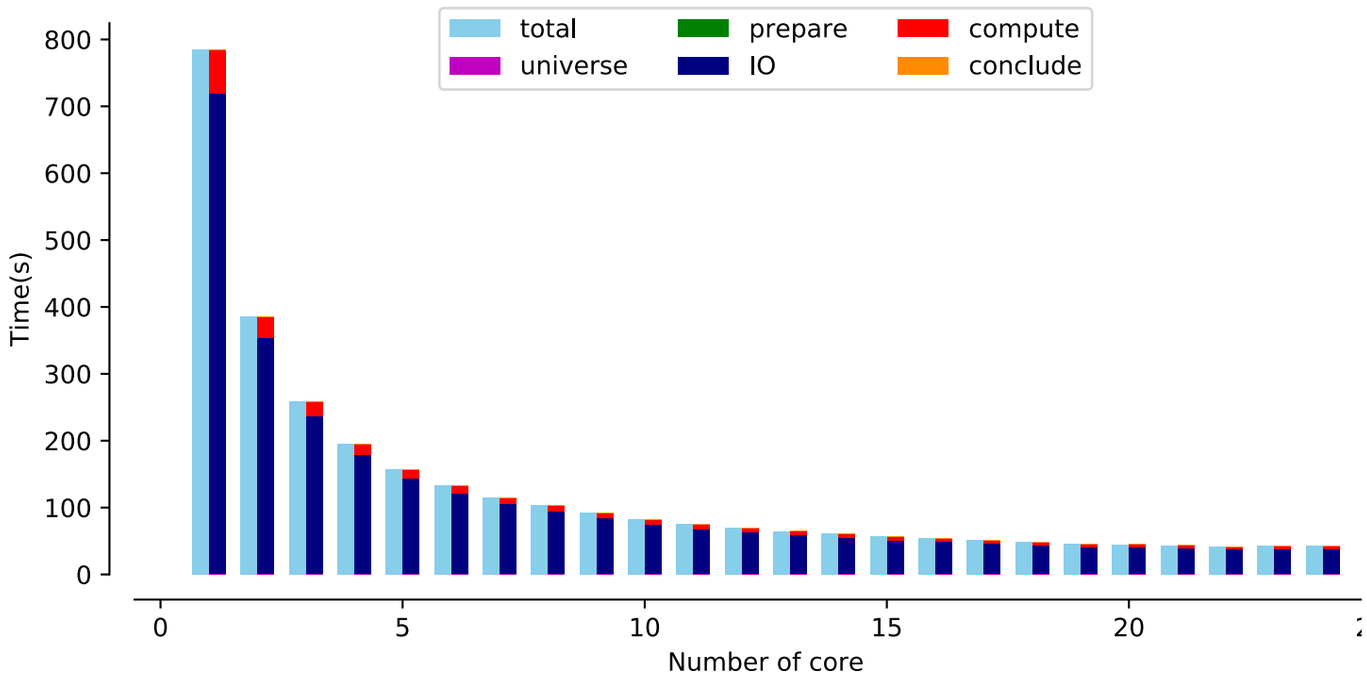
- usage is almost identical to MDAnalysis

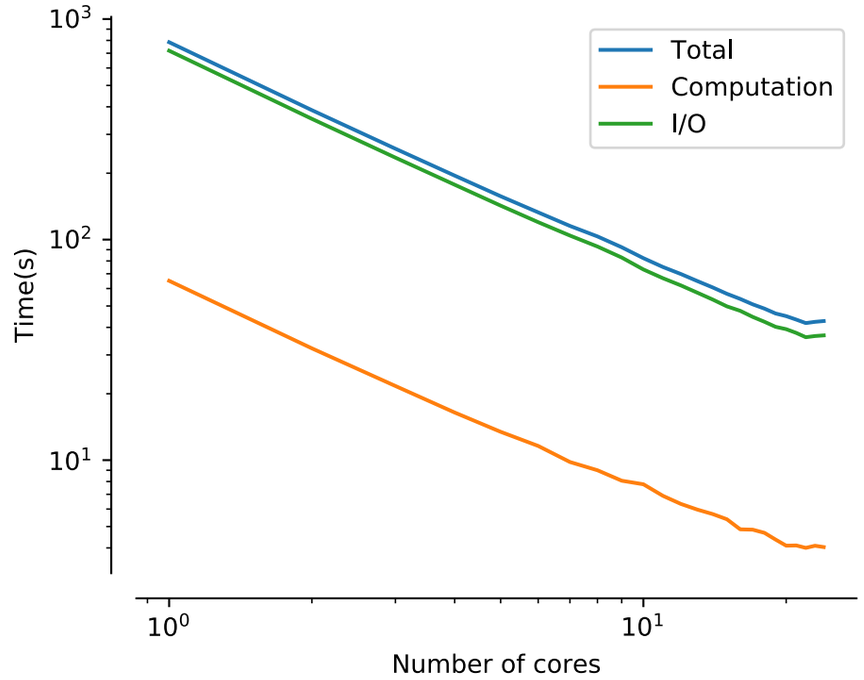
```
import MDAnalysis as mda
from pmda import rms

u = mda.Universe(top, traj)
ref = mda.Universe(top, traj)

import distributed
client = distributed.Client('192.168.0.1:8786')

rmsd_ana = rms.RMSD(u.atoms, ref.atoms).run()
```

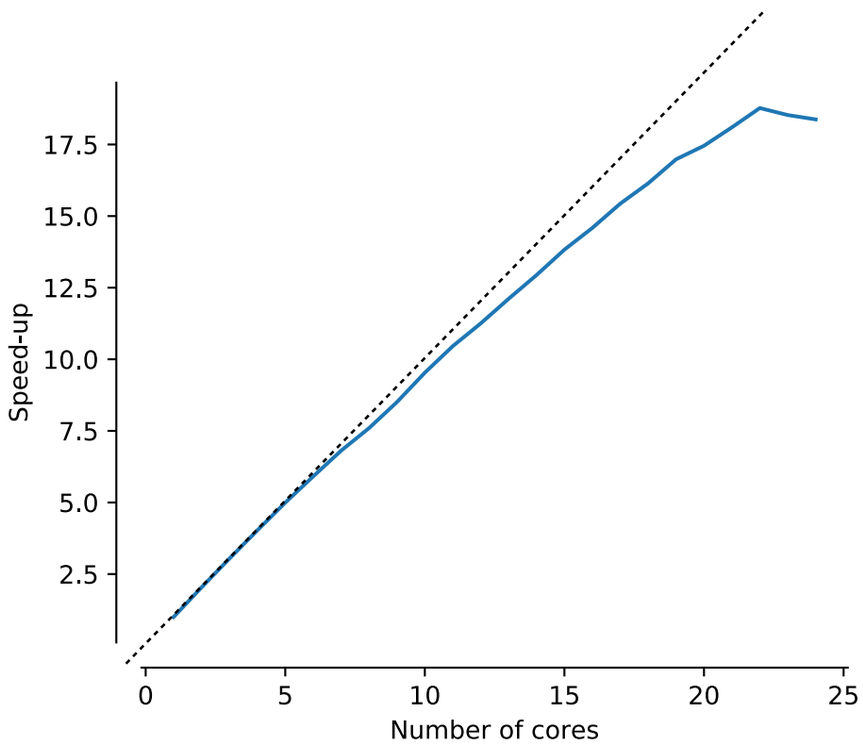
- easy to derive new "parallel" analysis classes from *pmda.parallel.ParallelAnalysisBase*

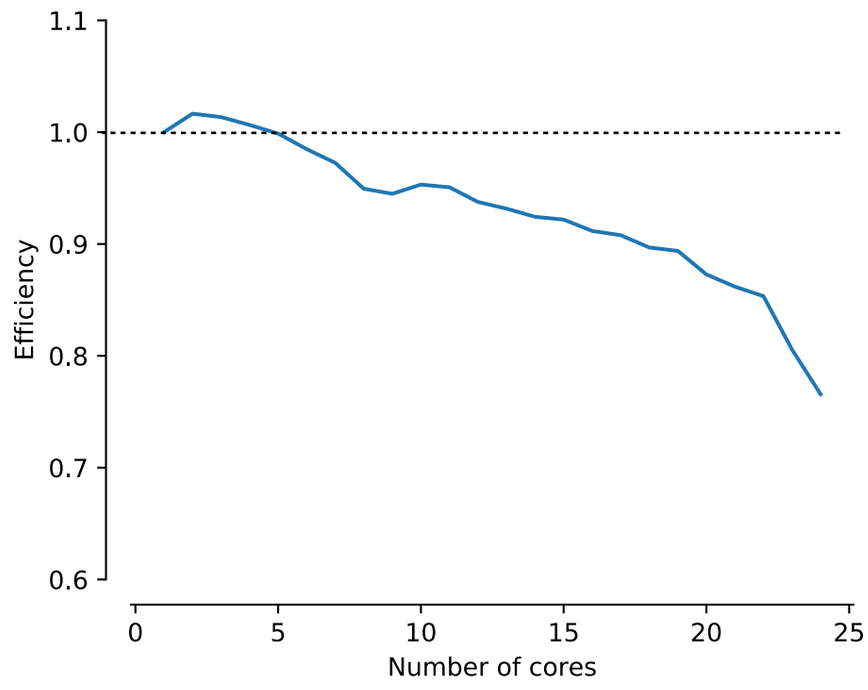
Performance on SDSC Comet

(RMSD calculation, up to 24 cores)









Conclusions

- A *lingua franca* (Python) makes it easy to build workflows for biomolecular simulations.
  - Pythonic "building blocks" (e.g., *MDAnalysis*, *alchemlyb*) are preferred.
    - If Python-bindings are missing, workarounds will be used… (*GromacsWrapper*…)
- FireWorks: workflows with decisions (e.g., free energy calculations)
- Next steps: analysis should influence sampling (steering)

(Software) Challenges

- debugging and testing workflows is hard
- creating re-usable software is hard
- getting other people to use one's software is hard
- maintaining software is hard

Acknowledgements

ASU College of Liberal Arts & Sciences
ARIZONA STATE UNIVERSITY

David Dotson
Shujie Fan
Mahzad Khoshlessan
Ian Kenney
Max Linke (visitor from MPI Frankfurt)

Beckstein Lab

RUTGERS
THE STATE UNIVERSITY OF NEW JERSEY

Shantenu Jha
Ioannis Paraskevakos
Hyungro Lee
Andre Merzky
Andre Luckow

ICSN Institut de Chimie des Substances Naturelles
CNRS (France)
Bogdan Iorga

UCI University of California, Irvine
David Mobley

University of Colorado Boulder
Michael Shirts

INDIANA UNIVERSITY
Geoffrey Fox

XSEDE
Extreme Science and Engineering Discovery Environment

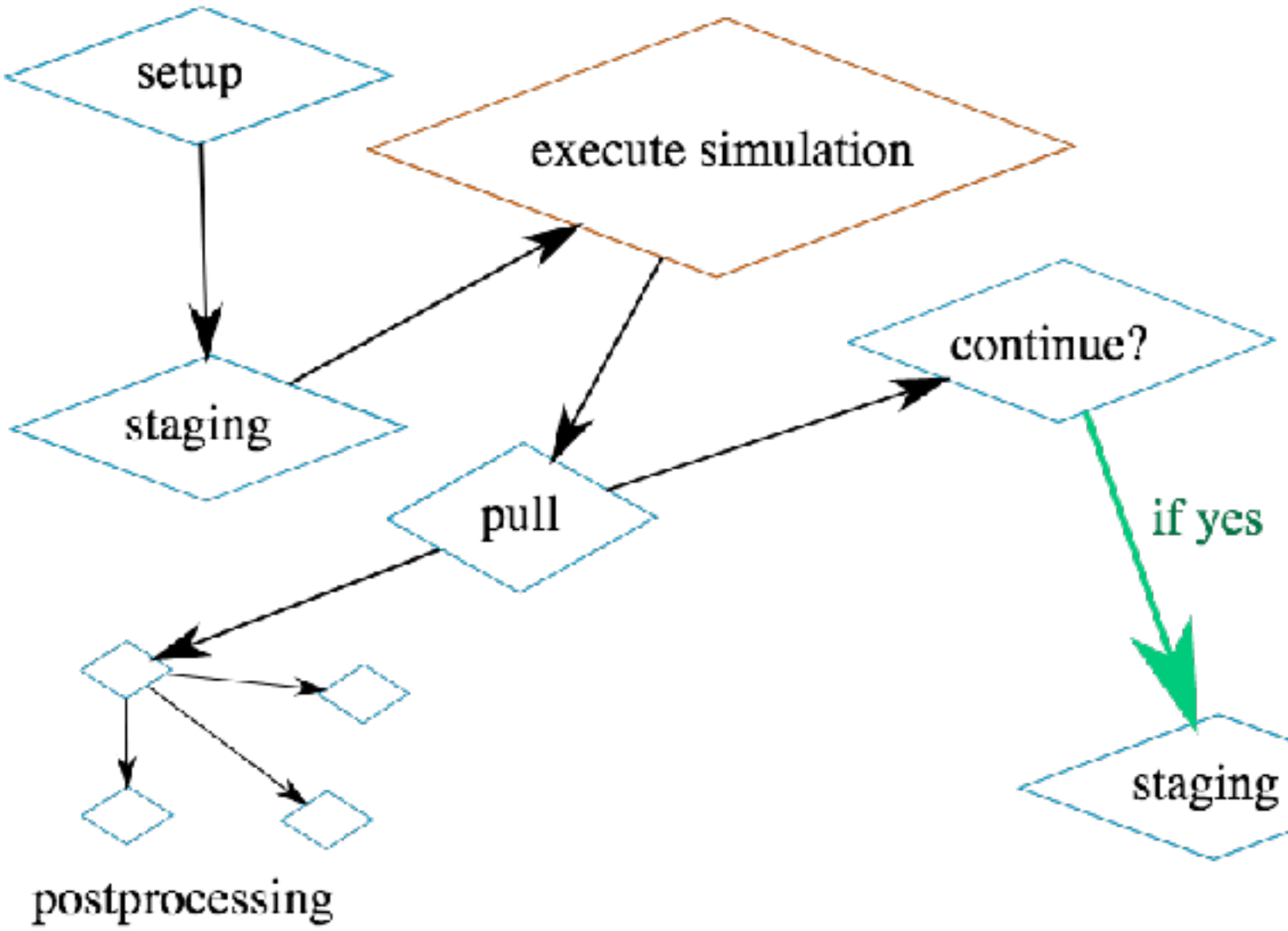
National Institutes of Health
NIGMS

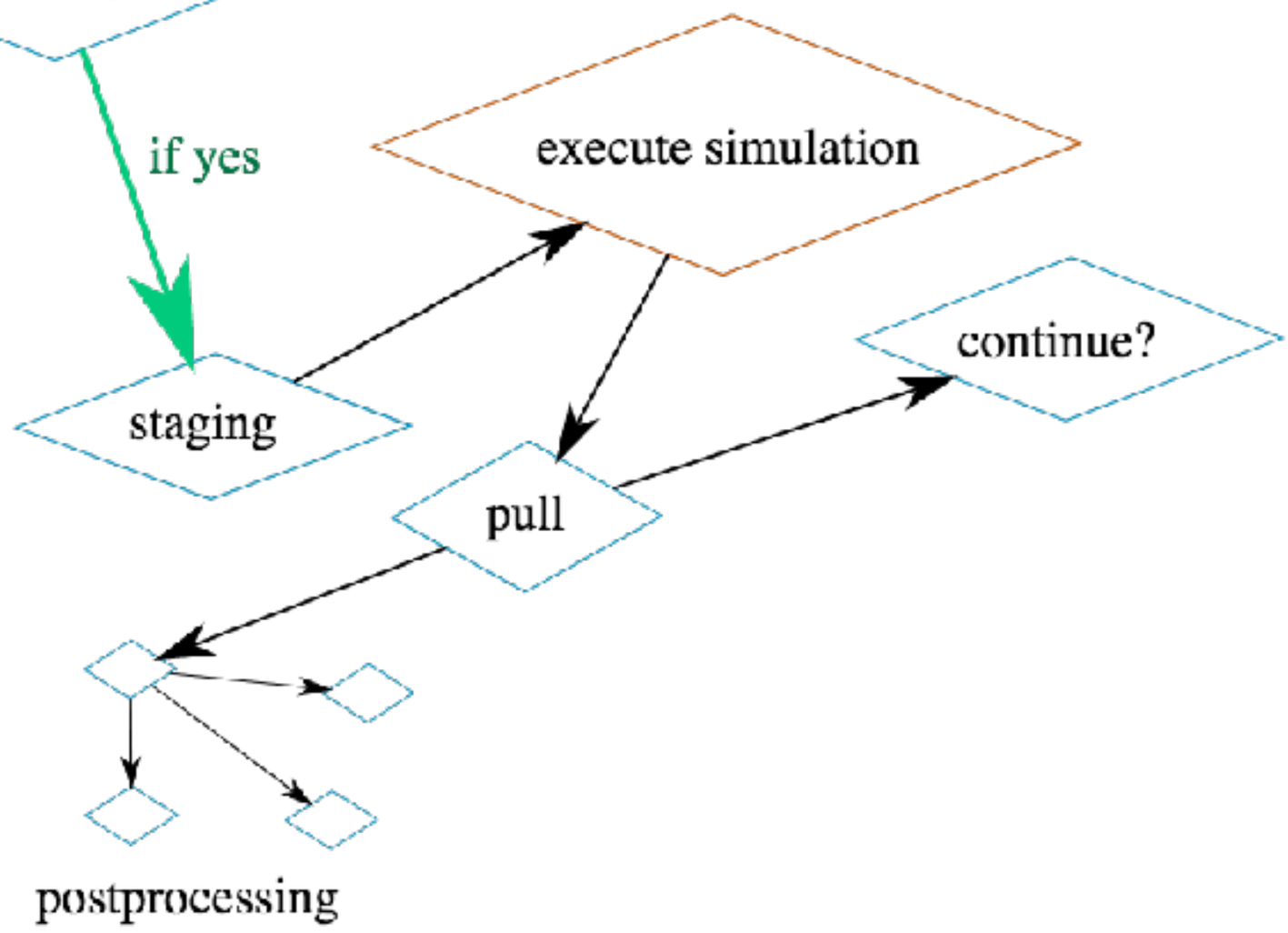
NSF

Google

NUMFOCUS
OPEN CODE = BETTER SCIENCE

Appendix


setup
execute simulation
staging
continue?
pull
if yes
postprocessing


execute simulation
continue?
staging
pull
postprocessing

Task: RMSD fitting

- optimally superimpose protein structure (selected CA atoms, ~110 atoms) on initial frame by minimising RMSD (Theobald QCProt[1,2] algorithm)

- RMSD as a function of time

- simple *map-reduce*

```
for iframe, ts in enumerate(u.trajectory[start:stop]):
        block[iframe, :] = ts.time, rmsd(CA, xref)
```

1. split trajectory in *N* blocks

2. compute RMSD for all frames in each block

3. join arrays of RMSDs in correct order

```
rmsds = numpy.vstack(blocks)
```

[1] P. Liu, D. K. Agrafiotis, and D. L. Theobald. Fast determination of the optimal rotational matrix for macromolecular superpositions. *J Comput Chem*, 31(7):1561–3, May 2010. doi: 10.1002/jcc.21439.
[2] D. L. Theobald. Rapid calculation of RMSDs using a quaternion-based characteristic polynomial. *Acta Crystallogr A*, 61(Pt 4):478–80, Jul 2005. doi: 10.1107/S0108767305015266.

How to use multicore workstations and/or HPC to speed-up analysis?

- ensembles (parallelize over trajectories): MPI
  - cpptraj[1]
  - radical.pilot[2]
- slice trajectories
  - HiMach[3]
  - pytraj
- accelerate functions (OpenMP)… everyone

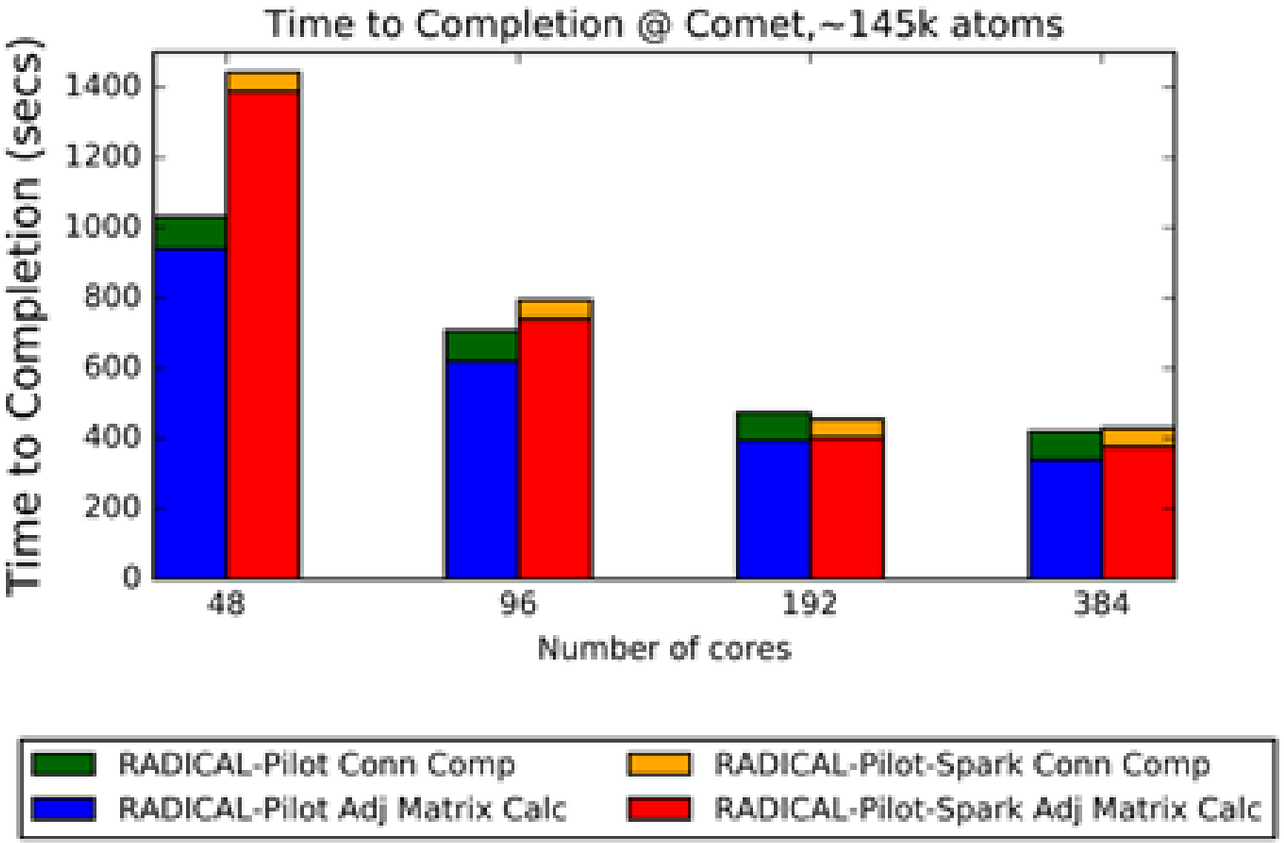
1) D. R. Roe and T. E. Cheatham. PTRAJ and CPPTRAJ: Software for processing and analysis of molecular dynamics trajectory data. *Journal of Chemical Theory and Computation*, 9(7):3084–3095, 2013. doi: 10.1021/ct400341p.

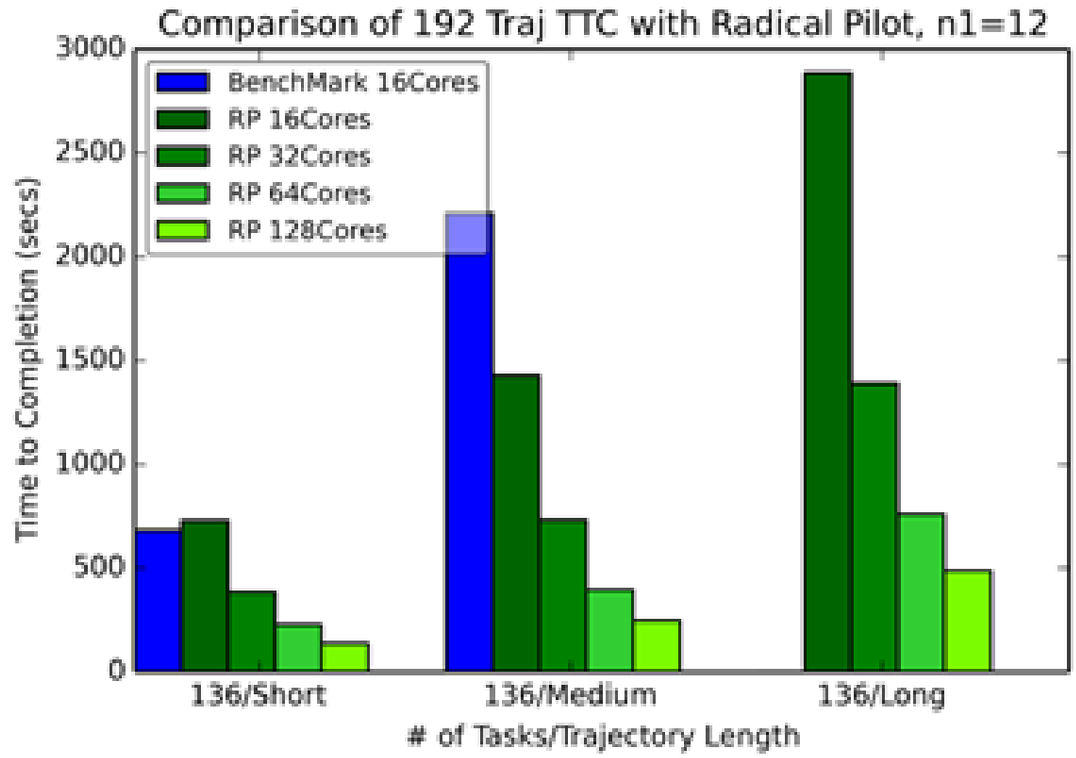
2) S Jha and co-workers, see http://radical-cybertools.github.io/radical-pilot/

3) T. Tu, C. Rendleman, D. Borhani, R. Dror, J. Gullingsrud, M. Jensen, J. L. Klepeis, P. Maragakis, P. Miller, K. Stafford, and D. E. Shaw. A scalable parallel framework for analyzing terascale molecular dynamics sim- ulation trajectories. In *International Conference for High Performance Computing, Networking, Storage and Analysis, 2008. SC 2008.*, pages 1–12, Austin, TX, 2008. IEEE. doi: 10.1109/SC.2008.5214715.

Radical.pilot

- **Shantenu Jha**, Ioannis Paraskevakos, Andre Merzky, Andre Luckow





LeafletFinder
(mixed problem)

Path Similarity Analysis
(Hausdorff distance matrix,
all-pairs problem)

$$T_1 \times T_2 \times n^2$$

As a usage example, we'll use `TI` to calculate the free energy of solvation of benzene in water. We'll use the benzene-in-water dataset from `alchemtest.gmx`:

```
>>> from alchemtest.gmx import load_benzene
>>> bz = load_benzene().data
```

and parse the datafiles separately for each alchemical leg using `alchemlyb.parsing.gmx.extract_dHdl()` to obtain dHdl gradients:

```
>>> from alchemlyb.parsing.gmx import extract_dHdl
>>> import pandas as pd
```

```
>>> dHdl_coul = pd.concat([extract_dHdl(xvg, T=300) for xvg in bz['Coulomb']])
>>> dHdl_vdw = pd.concat([extract_dHdl(xvg, T=300) for xvg in bz['VDW']])
```

We can now use the `TI` estimator to obtain the free energy differences between each λ window sampled. The `fit()` method is used to perform the free energy estimate, given the gradient data:

```
>>> from alchemlyb.estimators import TI

>>> ti_coul = TI()
>>> ti_coul.fit(dHdl_coul)
TI(verbose=False)

# we could also just call the `fit` method
# directly, since it returns the `TI` object
>>> ti_vdw = TI().fit(dHdl_vdw)
```

https://alchemlyb.readthedocs.io/en/latest/estimators-ti.html

The sum of the endpoint free energy differences will be the free energy of solvation for benzene in water. The free energy differences (in units of $k_B T$) between each λ window can be accessed via the `delta_f_` attribute:

```
>>> ti_coul.delta_f_
          0.00      0.25      0.50      0.75      1.00
0.00  0.000000  1.620328  2.573337  3.022170  3.089027
0.25 -1.620328  0.000000  0.953009  1.401842  1.468699
0.50 -2.573337 -0.953009  0.000000  0.448832  0.515690
0.75 -3.022170 -1.401842 -0.448832  0.000000  0.066857
1.00 -3.089027 -1.468699 -0.515690 -0.066857  0.000000
```

So we can get the endpoint differences (free energy difference between $\lambda = 0$ and $\lambda = 1$) of each with:

```
>>> ti_coul.delta_f_.loc[0.00, 1.00]
3.08902702186768 96

>>> ti_vdw.delta_f_.loc[0.00, 1.00]
-3.0558175199846058
```

giving us a solvation free energy in units of $k_B T$ for benzene of:

```
>>> ti_coul.delta_f_.loc[0.00, 1.00] + ti_vdw.delta_f_.loc[0.00, 1.00]
0.033209501883083803
```

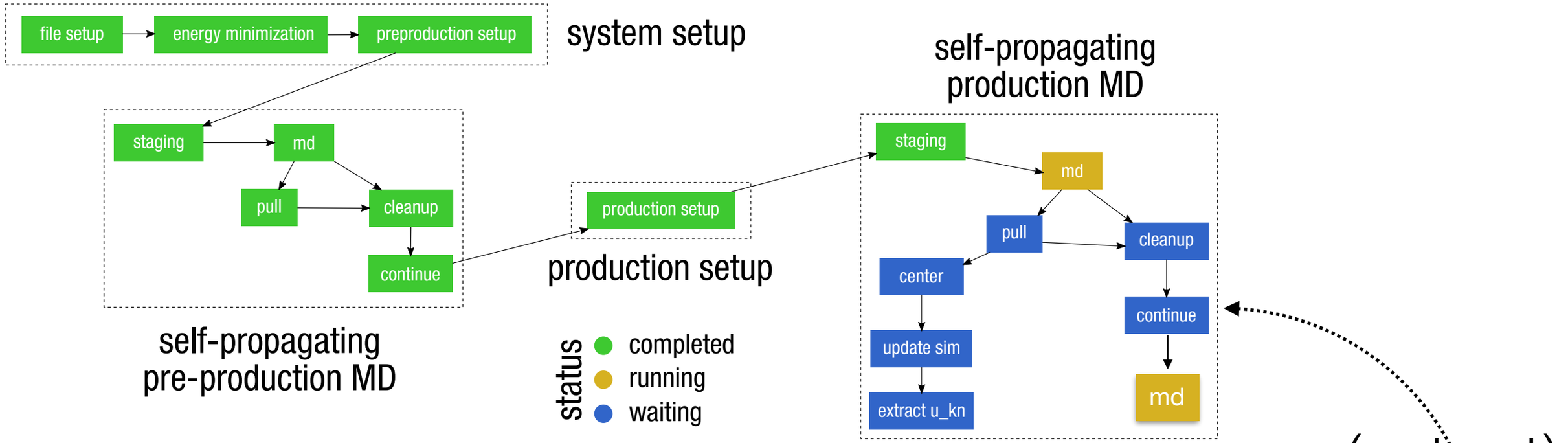
https://alchemlyb.readthedocs.io/en/latest/estimators-ti.html

In addition to the free energy differences, we also have access to the errors on these differences via the `d_delta_f_` attribute:

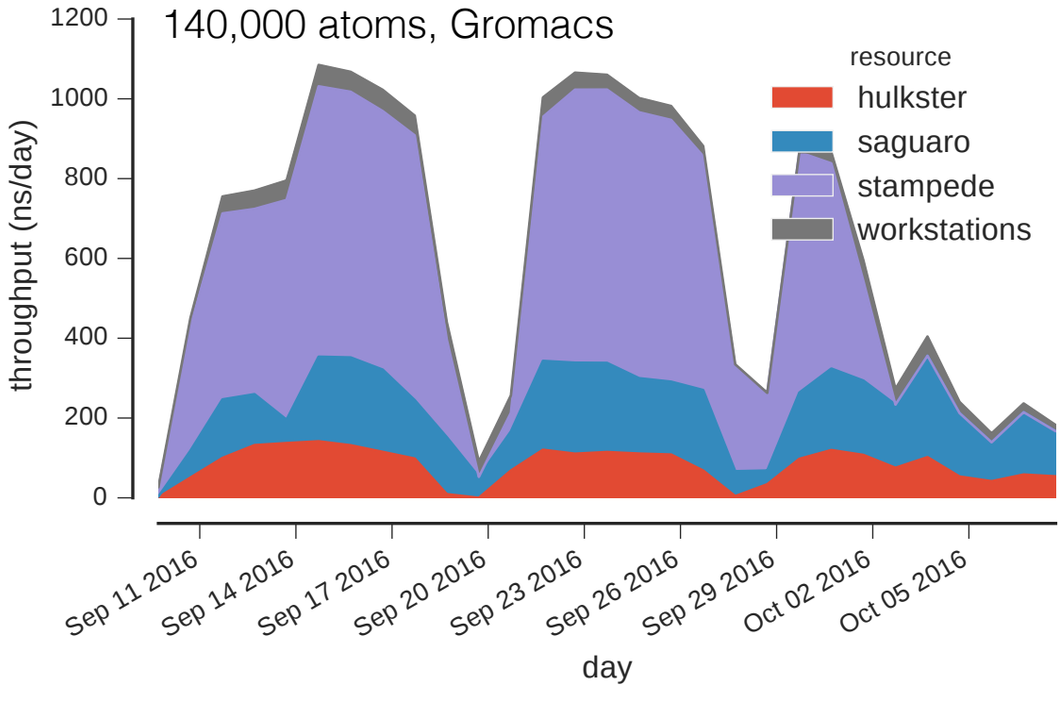
```
>>> ti_coul.d_delta_f_
          0.00      0.25      0.50      0.75      1.00
0.00  0.000000  0.009706  0.013058  0.015038  0.016362
0.25  0.009706  0.000000  0.008736  0.011486  0.013172
0.50  0.013058  0.008736  0.000000  0.007458  0.009858
0.75  0.015038  0.011486  0.007458  0.000000  0.006447
1.00  0.016362  0.013172  0.009858  0.006447  0.000000
```

https://alchemlyb.readthedocs.io/en/latest/estimators-ti.html

FEP workflow + analysis



mdworks: automated molecular dynamics workflow



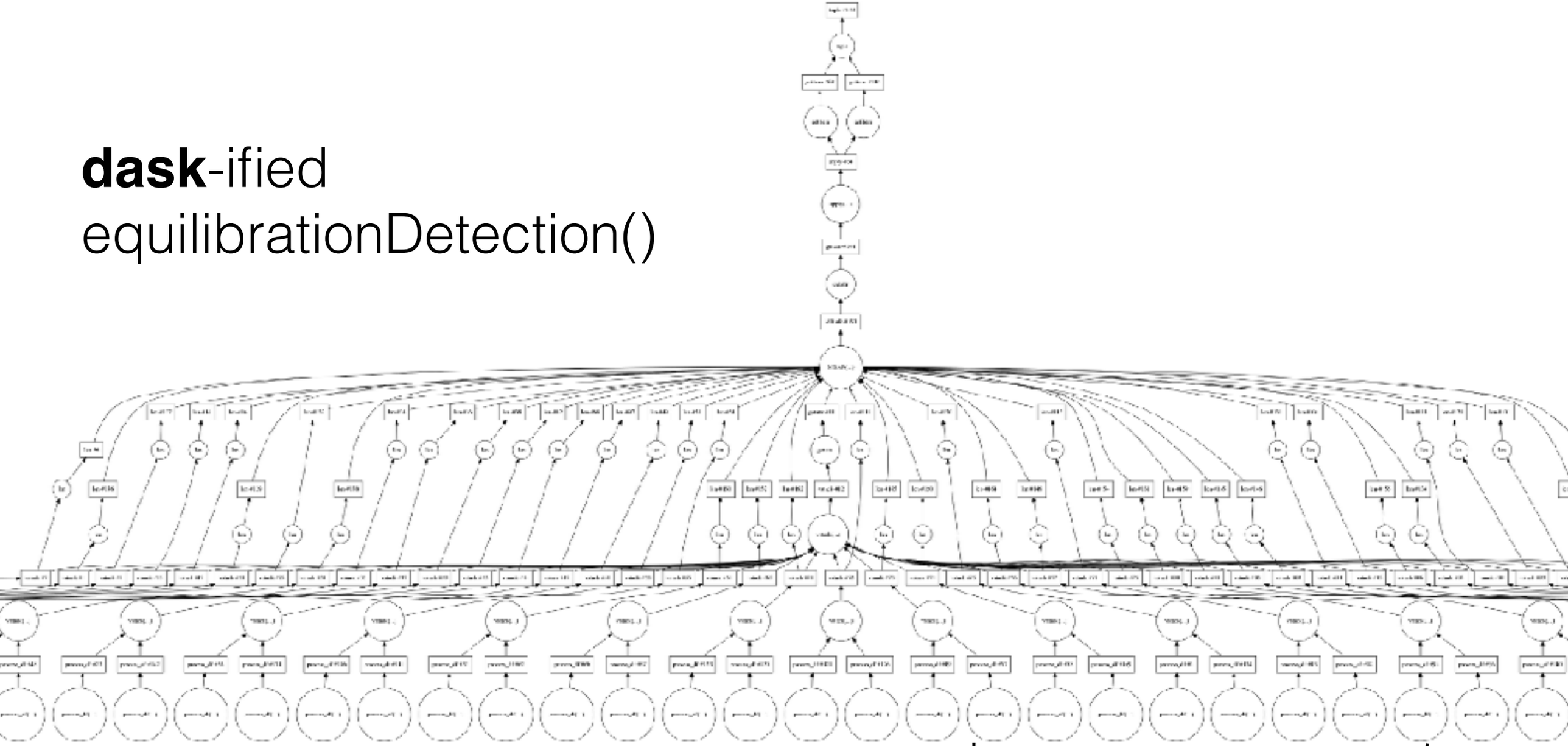
equilibrationDetection()

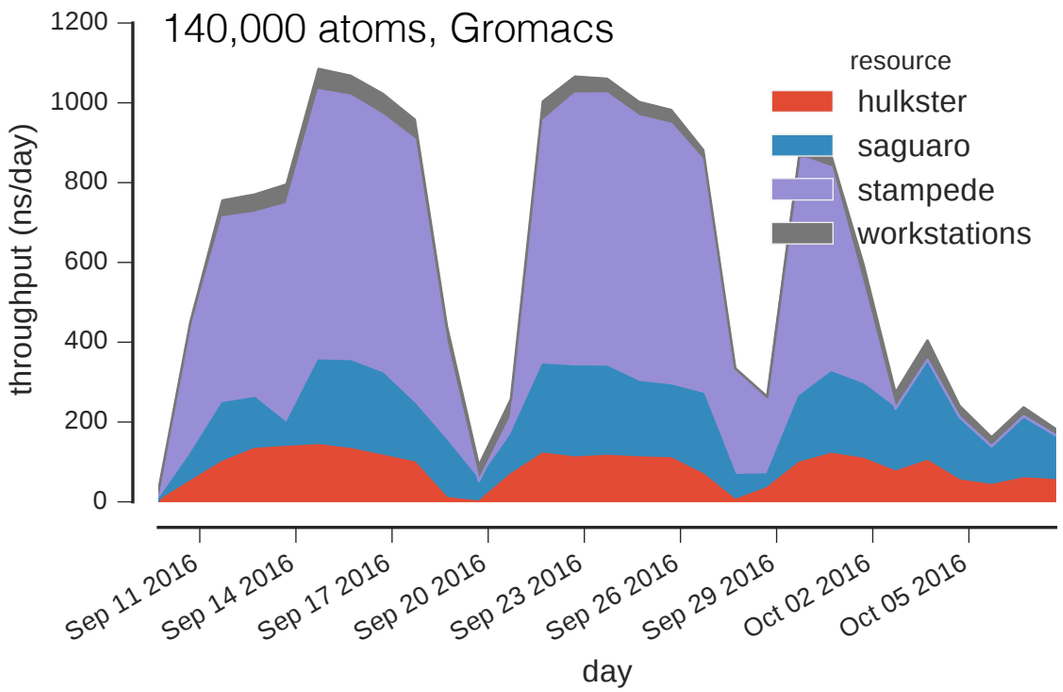
J. D. Chodera. *JCTC*, 12(4):1799–1805, 2016.

MBAR

M. R. Shirts and J. D. Chodera. *J Chem Phys*, 129(12):124105, 2008.

dask-ified
equilibrationDetection()





equilibrationDetection()

J. D. Chodera. *JCTC*, 12(4):1799–1805, 2016.

MBAR

M. R. Shirts and J. D. Chodera. *J Chem Phys*, 129(12):124105, 2008.