BioSimSpace – filling the gaps between molecular simulation codes

BioExcel Webinar Series

Presenter: Christopher Woods, *University of Bristol*
Host: Rossen Apostolov

27th June, 2018

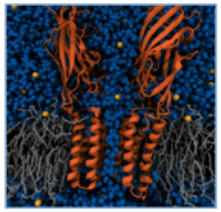
Partners

Funding
Horizon 2020
European Union Funding
for Research & Innovation

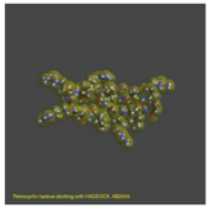
This webinar is being recorded

BioExcel Overview

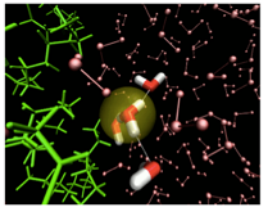
- **Excellence in Biomolecular Software**
  - Improve the performance, efficiency and scalability of key codes



MD simulations
/GROMACS/

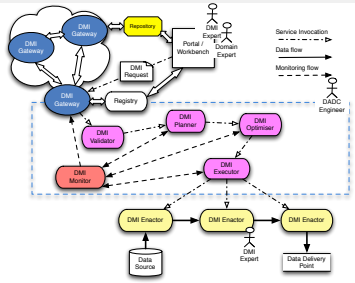


Docking
/HADDOCK/



QM/MM
/CPMD/

- **Excellence in Usability**
  - Devise efficient workflow environments with associated data integration



Key Workflows and Platforms

Galaxy PROJECT | Open PHACTS | COMPSs | Open for Innovation KNIME | Apache Taverna

- **Excellence in Consultancy and Training**
  - Promote best practices and train end users

Interest Groups

- Integrative Modeling IG
- Free Energy Calculations IG
- Hybrid methods for biomolecular systems IG
- Biomolecular simulations entry level users IG
- Practical applications for industry IG
- Training IG
- Workflows IG

Support platforms

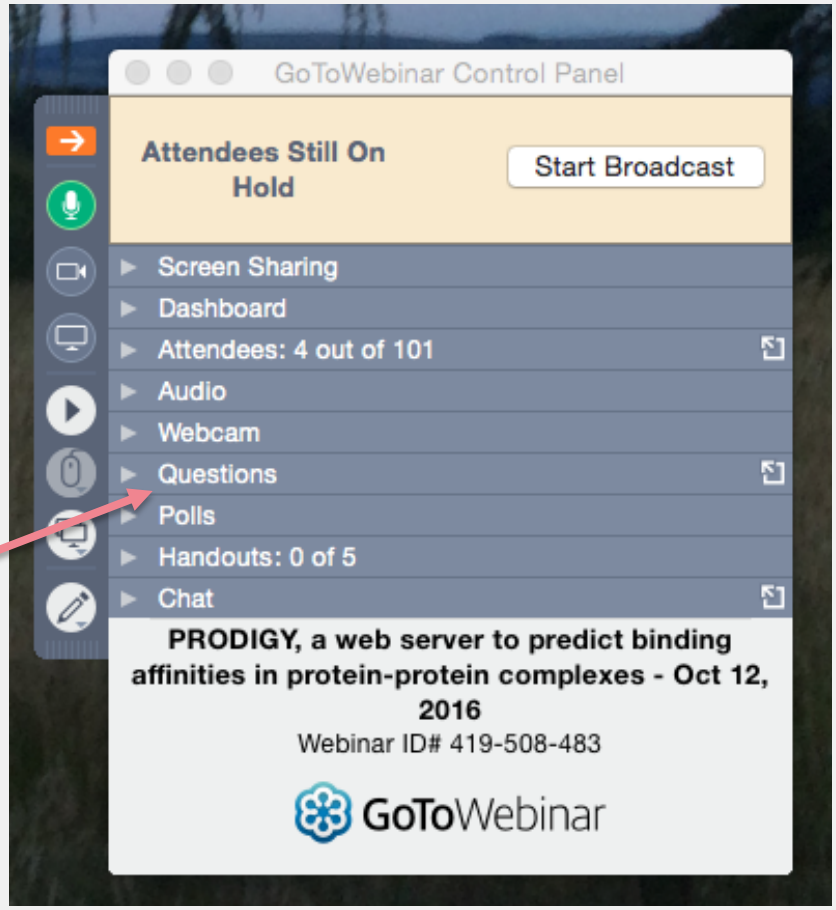
http://bioexcel.eu/contact

Forums **Code Repositories** **Chat Channel** **Video Channel**

Audience Q&A session

Please use the **Questions** function in GoToWebinar application

Any other questions or points to discuss after the live webinar? Join the discussion the discussion at http://ask.bioexcel.eu.



Today's Presenter

Christopher Woods, University of Bristol

christopher.woods@bristol.ac.uk

Christopher manages the Research Software Engineering (RSE) Group at the University of Bristol. He is an EPSRC RSE Fellow and Joint-Chair of the UK RSE Association. He obtained his undergraduate and postgraduate degrees in Chemistry from the University of Southampton, working with Prof Jonathan Essex, before moving to the University of Bristol as a chemist developing novel software and algorithms for modelling biological molecules. In 2016 he started the University of Bristol's RSE Group within the Advanced Computing Research Centre.

Website: https://chryswoods.com

ORCID: https://orcid.org/0000-0001-6563-9903

Twitter: @chryswoods, @biosimspace / #biosimspace

BioSimSpace

Filling the gaps between molecular simulation codes

Christopher Woods

EPSRC Research Software Engineering Fellow

Advanced Computing Research Centre

University of Bristol

EPSRC
Engineering and Physical Sciences Research Council

University of BRISTOL

Download these slides using the link at
https://chryswoods.com/talks

biosimspace_workshop/ | 03_interactive_md | Christopher

Secure https://workshop.biosimspace.org/user/a25045c3-b93a-4e6b-a660-66fbc91d3b27/notebo...

jupyter 03_interactive_md (unsaved changes)

Logout | Control Panel

File | Edit | View | Insert | Cell | Kernel | Widgets | Help | Trusted | Python 3

Run | Markdown

Interactive molecular dynamics

So far you have seen how to use BioSimSpace to write workflow components and run them in a Jupyter notebook, or from the command-line. BioSimSpace is also a great tool for playing around with molecular simulations directly and interacting with them in real-time. In this notebook you'll learn how to use BioSimSpace to set up and run an equilibration protocol, then query the running process for information, plot graphs of the latest data, visualise molecular configurations, and analyse trajectory data.

Before we get started, let's import BioSimSpace so that it's available inside of our notebook.

In [ ]:
```
import BioSimSpace as BSS
```

Creating a molecular system

First of all we need to load a molecular system. Once again, we'll use the examples from the `input` directory.

In [ ]:
```
system = BSS.readMolecules(["input/ala.crd", "input/ala.top"])
```

We have now created a molecular system. The system consists of an alanine dipeptide molecule in

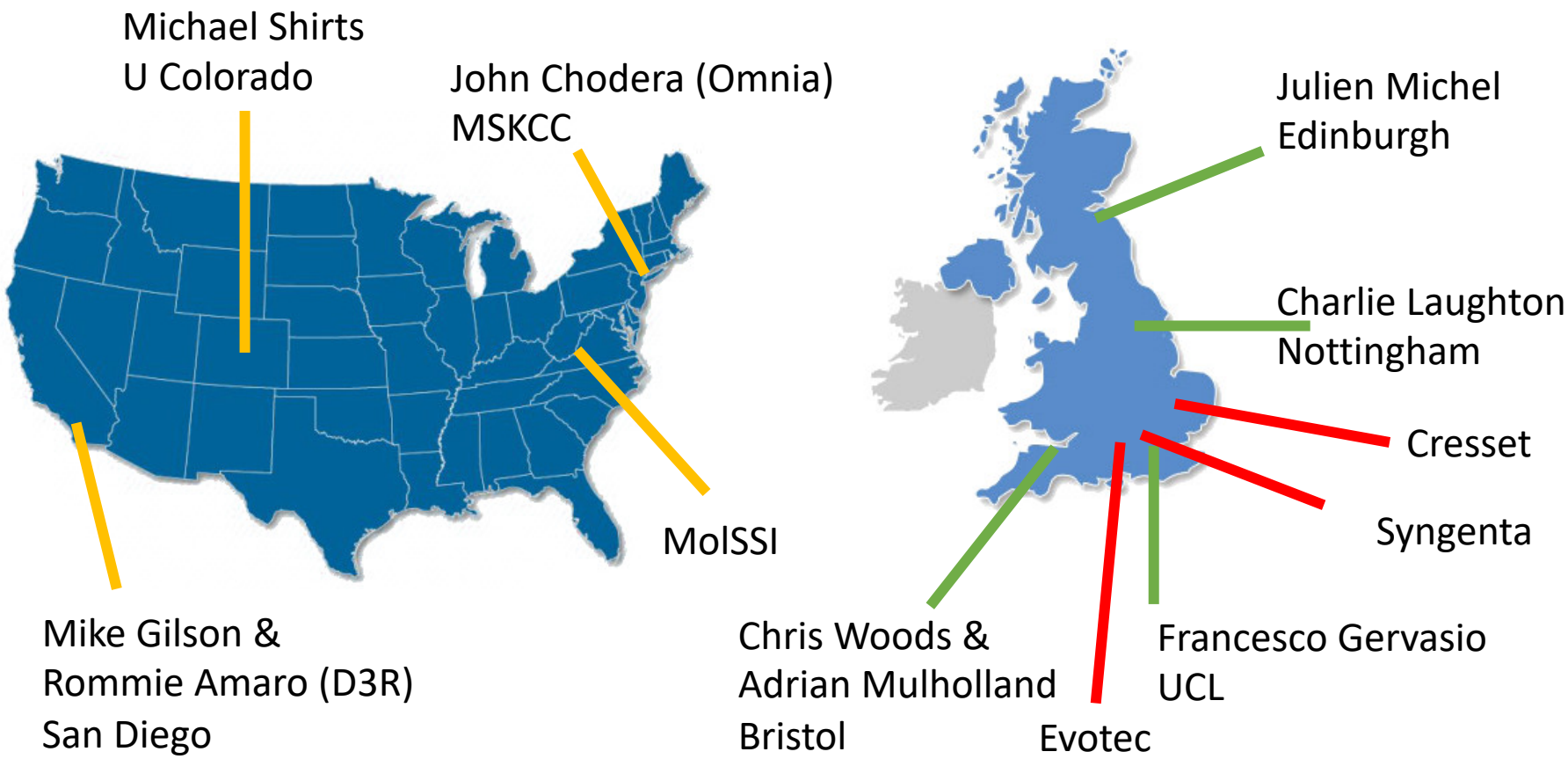
Why BioSimSpace?

How do I ….?

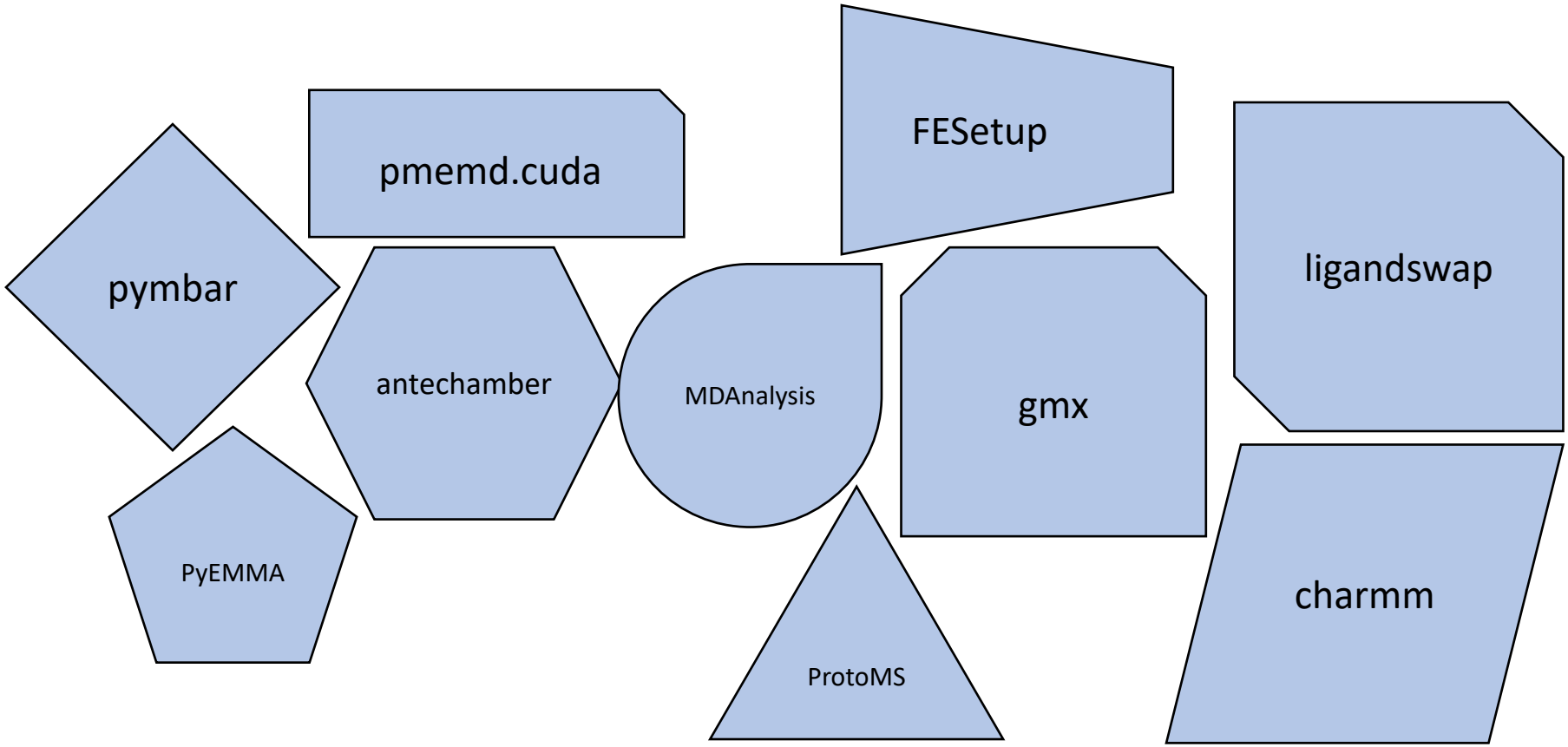
- **Researcher needs to do X**
  - Searches the web for how to do it
  - Finds a tutorial, blog post or online script
  - Copies and pastes onto their computer
    - Either something doesn't work, instructions fail
    - Or worse, it "works" but does the wrong thing
    - Or, if lucky, it works
  - **Researcher keeps going down the list of search results until they find a satisfactory solution (or give up)**
- **As a field, we are not good at sharing best practice or making it easy for newcomers to learn how to perform basic molecular simulation tasks**

Who are we?

- CCP-BioSim and HEC-BioSim led a successful bid at the last EPSRC Software Flagship Funding Call



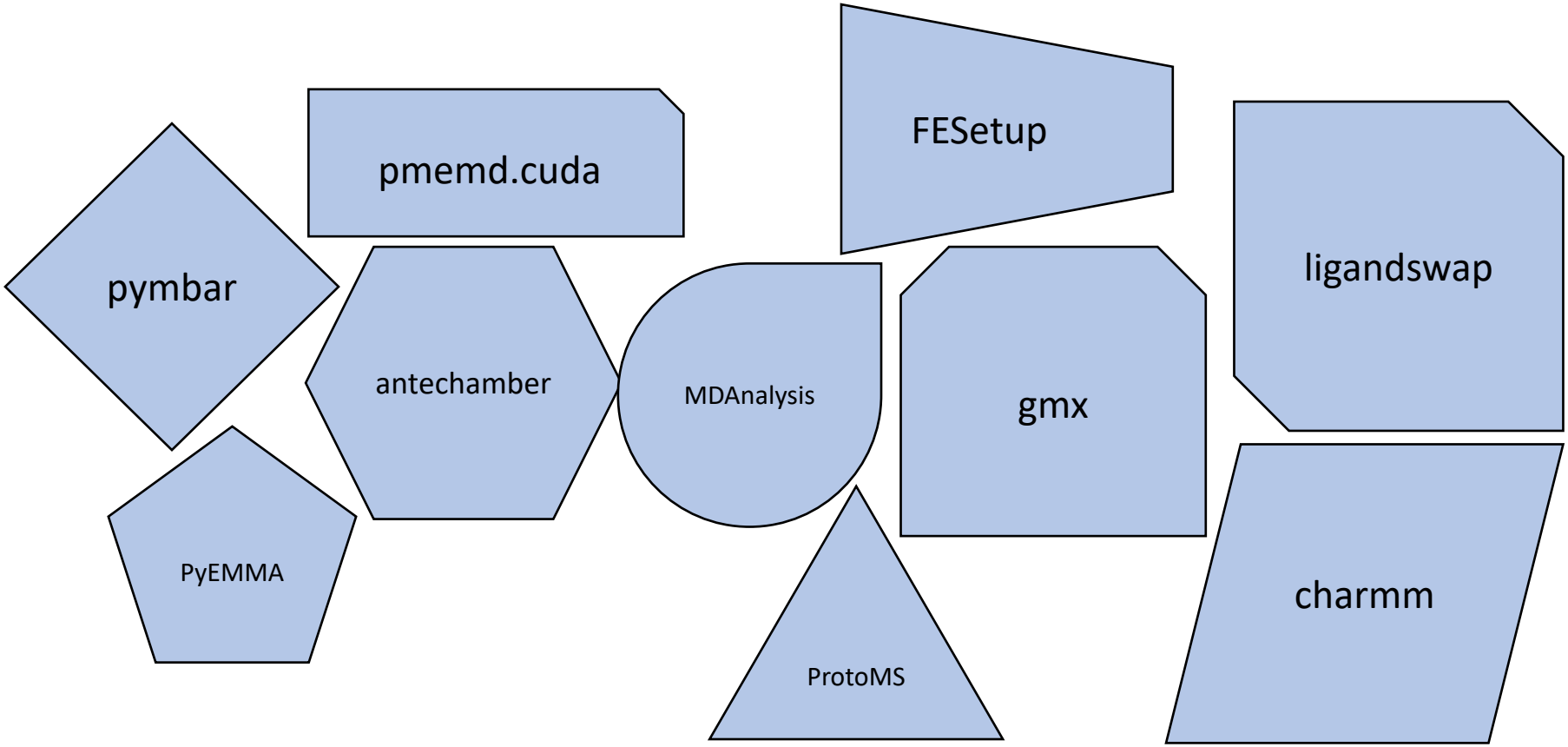
What is our software problem?



We have a lot of software in the community…

…but it doesn't fit together very well!

What is our software problem?

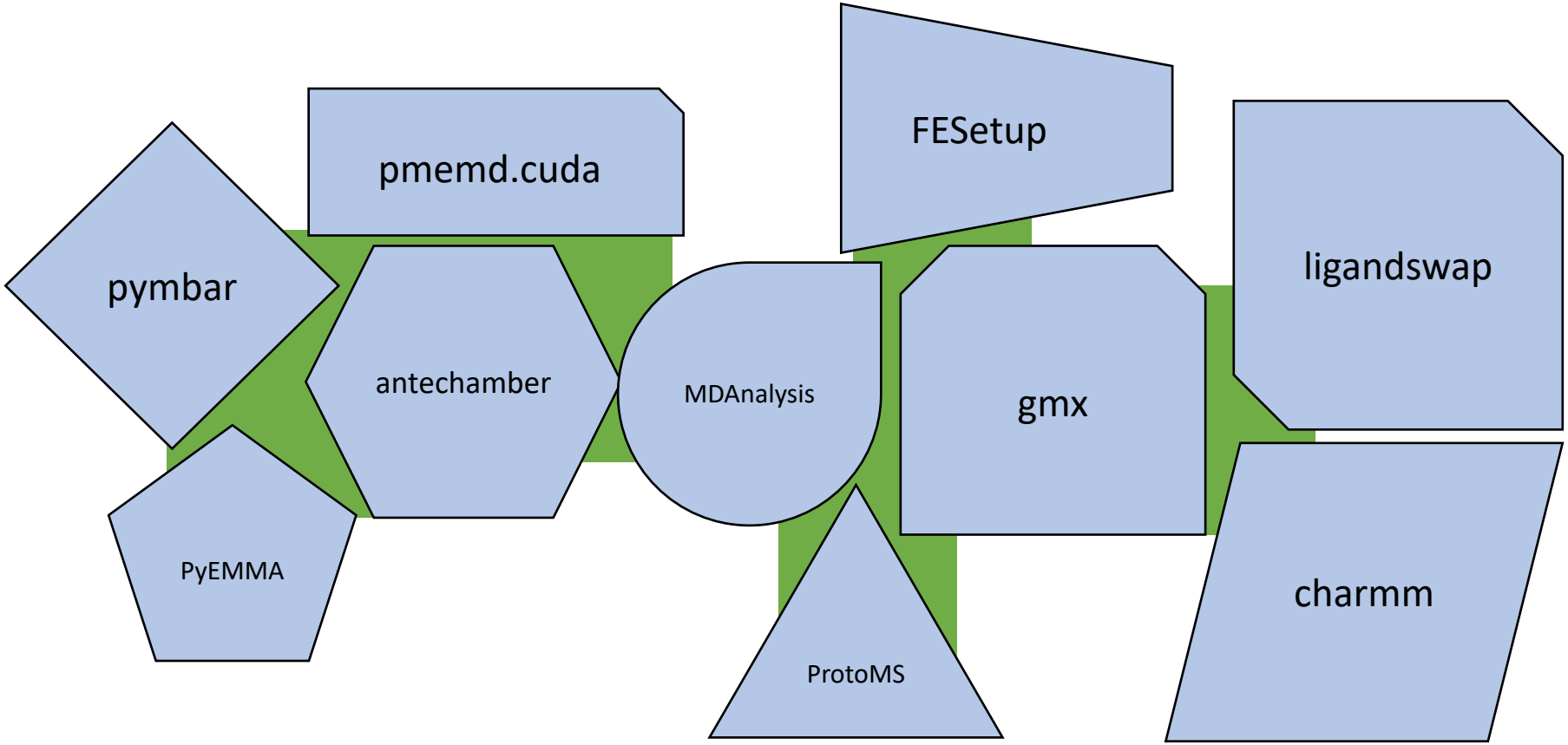


Difficult to know what exists, how to use it, what it does, or how to plug existing tools into a workflow – lack of **compatibility and interoperability**

Result - Lots of bespoke and brittle blogs / scripts / workflows

What is the **wrong** solution?

- Top-down
  - Collect the GREAT AND THE GOOD to create and mandate a NEW STANDARD FORMAT for Biomolecular Simulation
  - Make everyone read and write to this NEW STANDARD
  - Create a single STANDARD SIMULATION PACKAGE that everyone must use and develop tools within
  - Replace all existing tools with this **STANDARD**
- A top-down approach would fail.
  - Wouldn't unify anything. Would just create another "standard"
  - We do not need more standards and more software!



- Work with the existing formats and software we have
- Make it easier for this software to plug together
- Make it easier to translate one format into another

Make it easier to write the "shims"

What is BioSimSpace?

What is our solution?

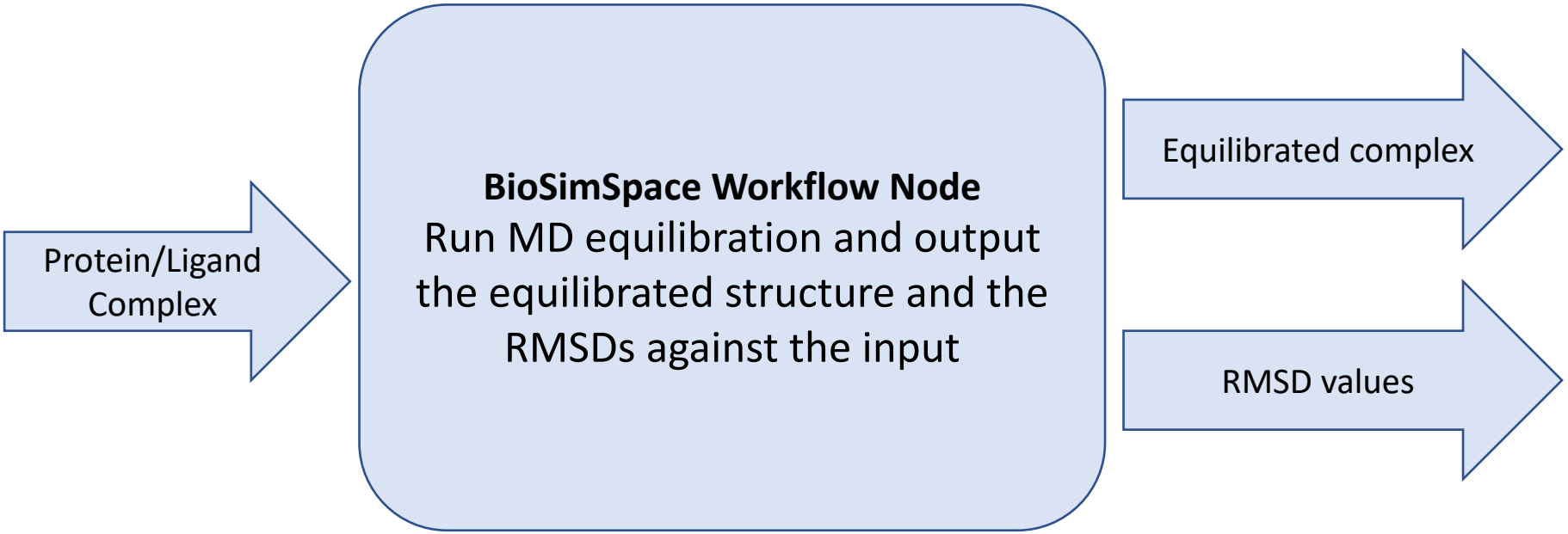
- **Bottom-up**
  - BioSimSpace is a collection of shims that make it easy for us to plug the community's existing software together
  - Exposes these tools within an easy-to-use **Python** environment
  - Ensure all tools can be used with a common, simple API, i.e. **same interface to run dynamics in all dynamics packages**, same interface to do alignment, trajectory analysis etc. etc.
  - BioSimSpace Python scripts can act as **workflow nodes** that **plug into existing workflow engines**, e.g. Knime, Pipeline-Pilot, ExTASY, command-line etc.
  - **Scripts can be used from the command line, from within a workflow engine, or interactively in a notebook**

Example workflow node

- Load a protein-ligand complex
- Run equilibration for a certain number of steps
- Calculate the RMSD with respect to the starting structure
- Output the equilibrated structure with a plot of the RMSD

Example workflow node

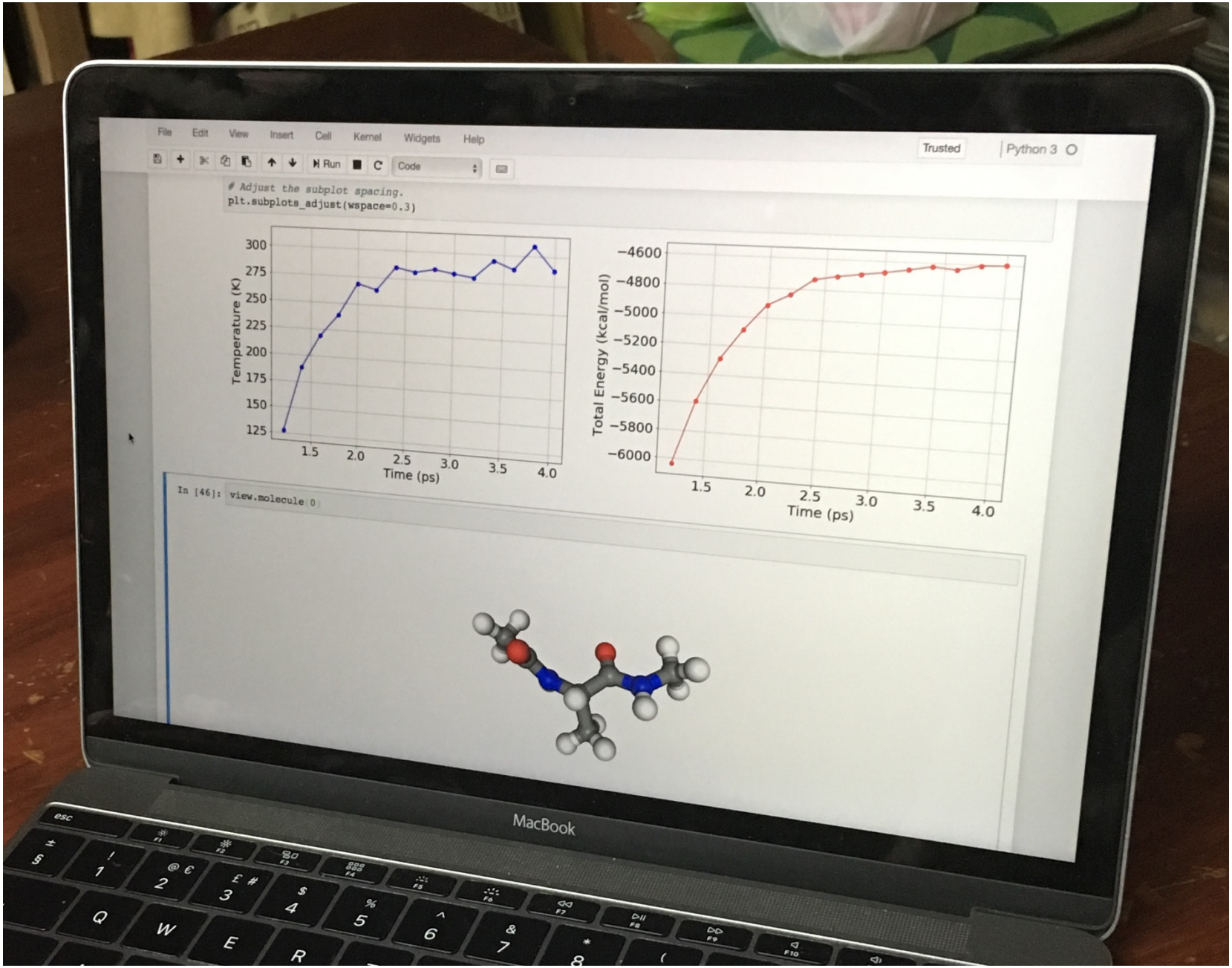
- Load a protein-ligand complex
  - …in any file format
- Run equilibration for a certain number of steps
  - …using any available MD package
- Calculate the RMSD with respect to the starting structure
  - …using any available trajectory analysis tool
- Output the equilibrated structure with a plot of the RMSD
  - …using the same file format as was loaded



Plug into Knime, Pipeline Pilot or ExTASY
Run in a Jupyter notebook
Or run from the command line using;

```
python component.py --complex files
```

```python
import BioSimSpace as BSS

node = BSS.Gateway.Node("A node to perform MD equilibration.")
node.addAuthor(name="Lester Hedges", email="lester.hedges@bristol.ac.uk",
                    affiliation="University of Bristol")
node.setLicense("GPLv3")

node.addInput("complex", BSS.Gateway.FileSet(help="a set of molecular input files"))
node.addOutput("equilibrated", BSS.Gateway.FileSet(
                                    help="the equilibrated molecular system"))
node.addOutput("rmsd", BSS.Gateway.File(help="the rmsd"))

system = BSS.readMolecules( node.getInput("files") )

protocol = BSS.Protocol.Equilibration(runtime=0.05, temperature_start=0,
                                      temperature_end=300, restrain_backbone=True)

process = BSS.MD.run(system, protocol)

rmsd = process.getTrajectory().RMSD(frame=0,
                                     molecule=system[BSS.MolWithResName("ALA")])
with open("rmsd.txt", "w") as file:
    for index, value in enumerate(rmsd):
        file.write("%d %5.4f\n" % (index, value))

node.setOutput(”equilibrated", BSS.saveMolecules("equilibrated”,
                                     process.getSystem(), system.fileFormat())

node.setOutput(”rmsd”, ”rmsd.txt”)
node.validate()
```


File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code
# Adjust the subplot spacing.
plt.subplots_adjust(wspace=0.3)
Temperature (K)
300
275
250
225
200
175
150
125
1.5 2.0 2.5 3.0 3.5 4.0
Time (ps)
Total Energy (kcal/mol)
−4600
−4800
−5000
−5200
−5400
−5600
−5800
−6000
1.5 2.0 2.5 3.0 3.5 4.0
Time (ps)
In [46]: view.molecule 0
MacBook

How much of BioSimSpace exists?

Current progress…

- Written many file conversion parsers
  - *Amber, Gromacs, Charmm, PDB, Mol2*
- Written drivers for MD programs
  - *Amber, Gromacs*
- Written interfaces for molecular analysis
  - *MDAnalysis, MDTraj*
- Written node interface
  - Command line, Jupyter, Knime (coming soon)
- Written molecular search parser
  - `molecules with (resname /ala/i or within 5 angstrom of ligand)`

Working on now…

- Setting up molecules
  - *tleap, antechamber, parmchk, sqm, pdb2gmx*
- Solvating molecules
  - *tleap, solvate*
- Mapping molecules for single-topology free energy calculations
- Drivers for single- and dual-topology free energy calculations
  - *Amber, gromacs, somd*
- **Project runs until the end of 2019**

```python
import BioSimSpace as BSS
from BSS.Units import angstrom

# Read a protein-ligand complex that has been parameterised with charmm
s = BSS.IO.readMolecules( ["NA16.gro", "NA16.grotop"] )

# Extract the protein and ligand molecules
protein = s.search("molecule with resname /ala/i")
ligand = s.search("molecule with resname /zan/i")

# Parameterise the protein using Amber FF14SB
protein = BSS.Parameters.ff14SB(protein).getMolecule()

# Parameterise the ligand using Amber GAFF2
ligand = BSS.Parameters.gaff2(ligand).getMolecule()

# Now re-solvate the complex using TIP3P water with a 10 A buffer
system = BSS.Solvate.tip3p( protein + ligand, buffer=10*angstrom ).getSystem()

# Minimise and then equilibrate the new system
process = BSS.MD.run(system, BSS.Protocol.Minimisation())
process = BSS.MD.run(process.getSystem(), BSS.Protocol.Equilibration())

# Save the new molecules using the same file format as input
BSS.IO.saveMolecules(process.getSystem(), "NA16_updated", s.fileFormat())
```

What Science will we be Investigating?

- Software should be used while it is developed!
  - Ensures it is useful, and finds and fixes bugs
- Two “grand challenge” applications
  1. Automatic setup and running of binding free energy calculations on a large number of protein-ligand systems, comparing results against D3R datasets, to improve quality and predictivity of results (run Q4 2018)
  2. Automatic run and investigation of protein-ligand binding kinetics and binding free energies from advanced metadynamics and post-metadynamics simulations (run H1 2019)

BioSimSpace on Jupyter

HTML5 Web Browser
(with fully-working WebSockets)

jupyter

Client

HTTP / HTTPS

Server

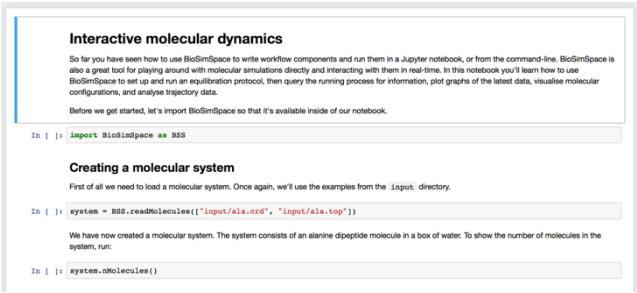
HTML5 Web Browser
(with fully-working WebSockets)

jupyter

Client

HTTP / HTTPS

Server



Interactive molecular dynamics

So far you have seen how to use BioSimSpace to write workflow components and run them in a Jupyter notebook, or from the command-line. BioSimSpace is also a great tool for playing around with molecular simulations directly and interacting with them in real-time. In this notebook you'll learn how to use BioSimSpace to set up and run an equilibration protocol, then query the running process for information, plot graphs of the latest data, visualise molecular configurations, and analyse trajectory data.

Before we get started, let's import BioSimSpace so that it's available inside of our notebook.

In [ ]:
```
import BioSimSpace as BSS
```

Creating a molecular system

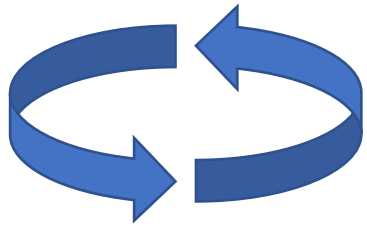
First of all we need to load a molecular system. Once again, we'll use the examples from the `input` directory.

In [ ]:
```
system = BSS.readMolecules(["input/ala.crd", "input/ala.top"])
```

We have now created a molecular system. The system consists of an alanine dipeptide molecule in a box of water. To show the number of molecules in the system, run:

In [ ]:
```
system.nMolecules()
```

Jupyter Notebook



Python Kernel

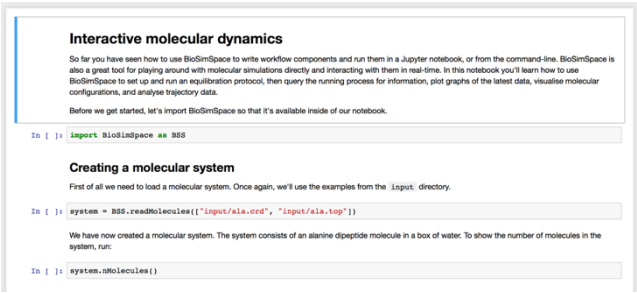
HTML5 Web Browser
(with fully-working WebSockets)

jupyter

Client

HTTP / HTTPS

Server



Interactive molecular dynamics

So far you have seen how to use BioSimSpace to write workflow components and run them in a Jupyter notebook, or from the command-line. BioSimSpace is also a great tool for playing around with molecular simulations directly and interacting with them in real-time. In this notebook you'll learn how to use BioSimSpace to set up and run an equilibration protocol, then query the running process for information, plot graphs of the latest data, visualise molecular configurations, and analyse trajectory data.

Before we get started, let's import BioSimSpace so that it's available inside of our notebook.

In [ ]:
```
import BioSimSpace as BSS
```

Creating a molecular system

First of all we need to load a molecular system. Once again, we'll use the examples from the `input` directory.

In [ ]:
```
system = BSS.readMolecules(["input/ala.crd", "input/ala.top"])
```

We have now created a molecular system. The system consists of an alanine dipeptide molecule in a box of water. To show the number of molecules in the system, run:

In [ ]:
```
system.nMolecules()
```

Jupyter Notebook

Python Snippet

Python Kernel

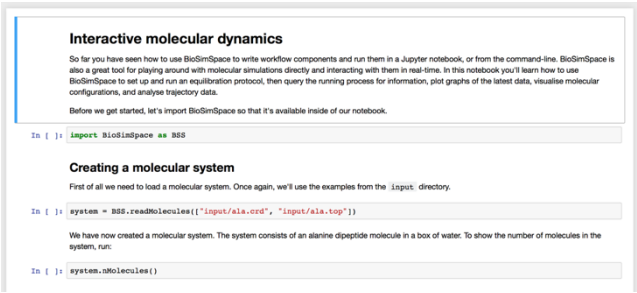
HTML5 Web Browser
(with fully-working WebSockets)

jupyter

Client

HTTP / HTTPS

Server



Interactive molecular dynamics

So far you have seen how to use BioSimSpace to write workflow components and run them in a Jupyter notebook, or from the command-line. BioSimSpace is also a great tool for playing around with molecular simulations directly and interacting with them in real-time. In this notebook you'll learn how to use BioSimSpace to set up and run an equilibration protocol, then query the running process for information, plot graphs of the latest data, visualise molecular configurations, and analyse trajectory data.

Before we get started, let's import BioSimSpace so that it's available inside of our notebook.

```
In [ ]: import BioSimSpace as BSS
```

Creating a molecular system

First of all we need to load a molecular system. Once again, we'll use the examples from the `input` directory.

```
In [ ]: system = BSS.readMolecules(["input/ala.crd", "input/ala.top"])
```

We have now created a molecular system. The system consists of an alanine dipeptide molecule in a box of water. To show the number of molecules in the system, run:

```
In [ ]: system.nMolecules()
```

Jupyter Notebook

Python Snippet

Python Kernel

Execute on server

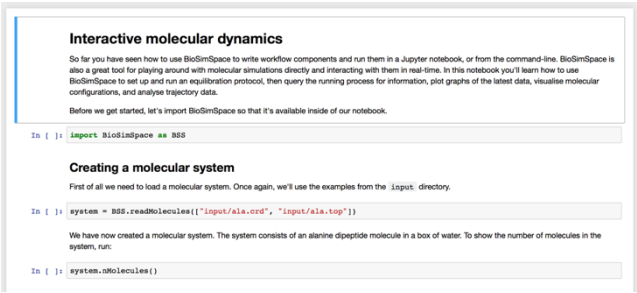
HTML5 Web Browser
(with fully-working WebSockets)

jupyter

Client

HTTP / HTTPS

Server



Interactive molecular dynamics

So far you have seen how to use BioSimSpace to write workflow components and run them in a Jupyter notebook, or from the command-line. BioSimSpace is also a great tool for playing around with molecular simulations directly and interacting with them in real-time. In this notebook you'll learn how to use BioSimSpace to set up and run an equilibration protocol, then query the running process for information, plot graphs of the latest data, visualise molecular configurations, and analyse trajectory data.

Before we get started, let's import BioSimSpace so that it's available inside of our notebook.

```
In [ ]: import BioSimSpace as BSS
```

Creating a molecular system

First of all we need to load a molecular system. Once again, we'll use the examples from the `input` directory.

```
In [ ]: system = BSS.readMolecules(["input/ala.crd", "input/ala.top"])
```

We have now created a molecular system. The system consists of an alanine dipeptide molecule in a box of water. To show the number of molecules in the system, run:

```
In [ ]: system.nMolecules()
```

Jupyter Notebook

Python Snippet

Python Kernel

Return result

Execute on server

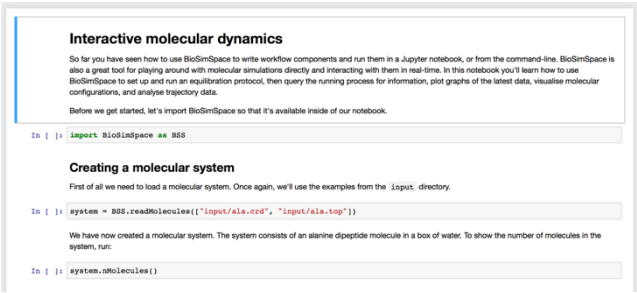
HTML5 Web Browser
(with fully-working WebSockets)

jupyter

Client

HTTP / HTTPS

Server



Interactive molecular dynamics

So far you have seen how to use BioSimSpace to write workflow components and run them in a Jupyter notebook, or from the command-line. BioSimSpace is also a great tool for playing around with molecular simulations directly and interacting with them in real-time. In this notebook you'll learn how to use BioSimSpace to set up and run an equilibration protocol, then query the running process for information, plot graphs of the latest data, visualise molecular configurations, and analyse trajectory data.

Before we get started, let's import BioSimSpace so that it's available inside of our notebook.

In [ ]:
```
import BioSimSpace as BSS
```

Creating a molecular system

First of all we need to load a molecular system. Once again, we'll use the examples from the `input` directory.

In [ ]:
```
system = BSS.readMolecules(["input/ala.crd", "input/ala.top"])
```

We have now created a molecular system. The system consists of an alanine dipeptide molecule in a box of water. To show the number of molecules in the system, run:

In [ ]:
```
system.nMolecules()
```

Jupyter Notebook

Python Snippet

Return renderable

Python Kernel

Return result

Execute on server

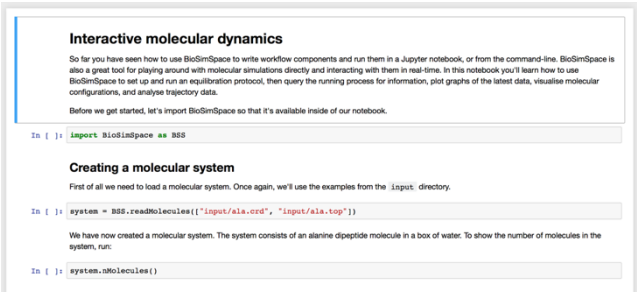
HTML5 Web Browser
(with fully-working WebSockets)

jupyter

Client

HTTP / HTTPS

Server



Interactive molecular dynamics

So far you have seen how to use BioSimSpace to write workflow components and run them in a Jupyter notebook, or from the command-line. BioSimSpace is also a great tool for playing around with molecular simulations directly and interacting with them in real-time. In this notebook you'll learn how to use BioSimSpace to set up and run an equilibration protocol, then query the running process for information, plot graphs of the latest data, visualise molecular configurations, and analyse trajectory data.

Before we get started, let's import BioSimSpace so that it's available inside of our notebook.

In [ ]:
```
import BioSimSpace as BSS
```

Creating a molecular system

First of all we need to load a molecular system. Once again, we'll use the examples from the `input` directory.

In [ ]:
```
system = BSS.readMolecules(["input/ala.crd", "input/ala.top"])
```

We have now created a molecular system. The system consists of an alanine dipeptide molecule in a box of water. To show the number of molecules in the system, run:

In [ ]:
```
system.nMolecules()
```

Jupyter Notebook

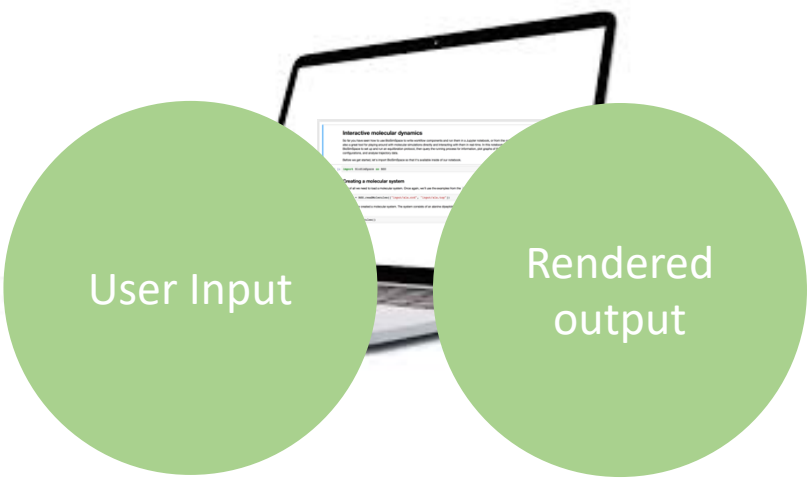
Python Snippet

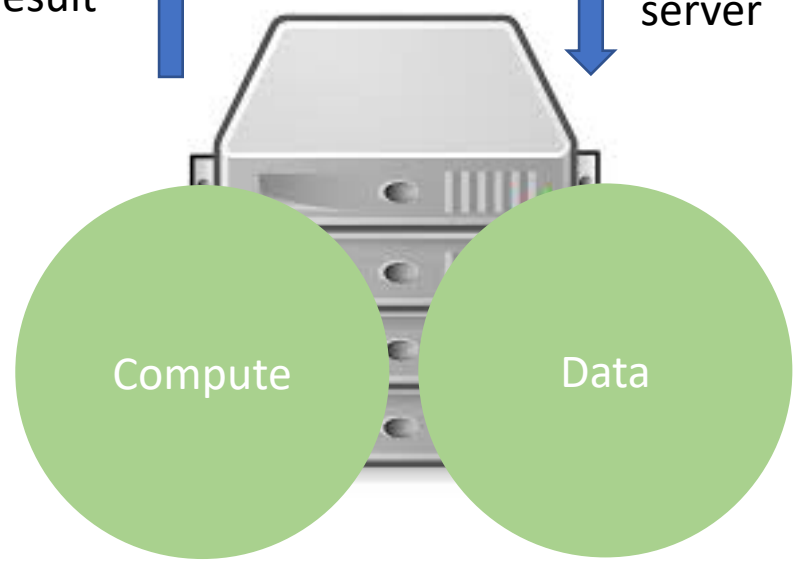
Return renderable

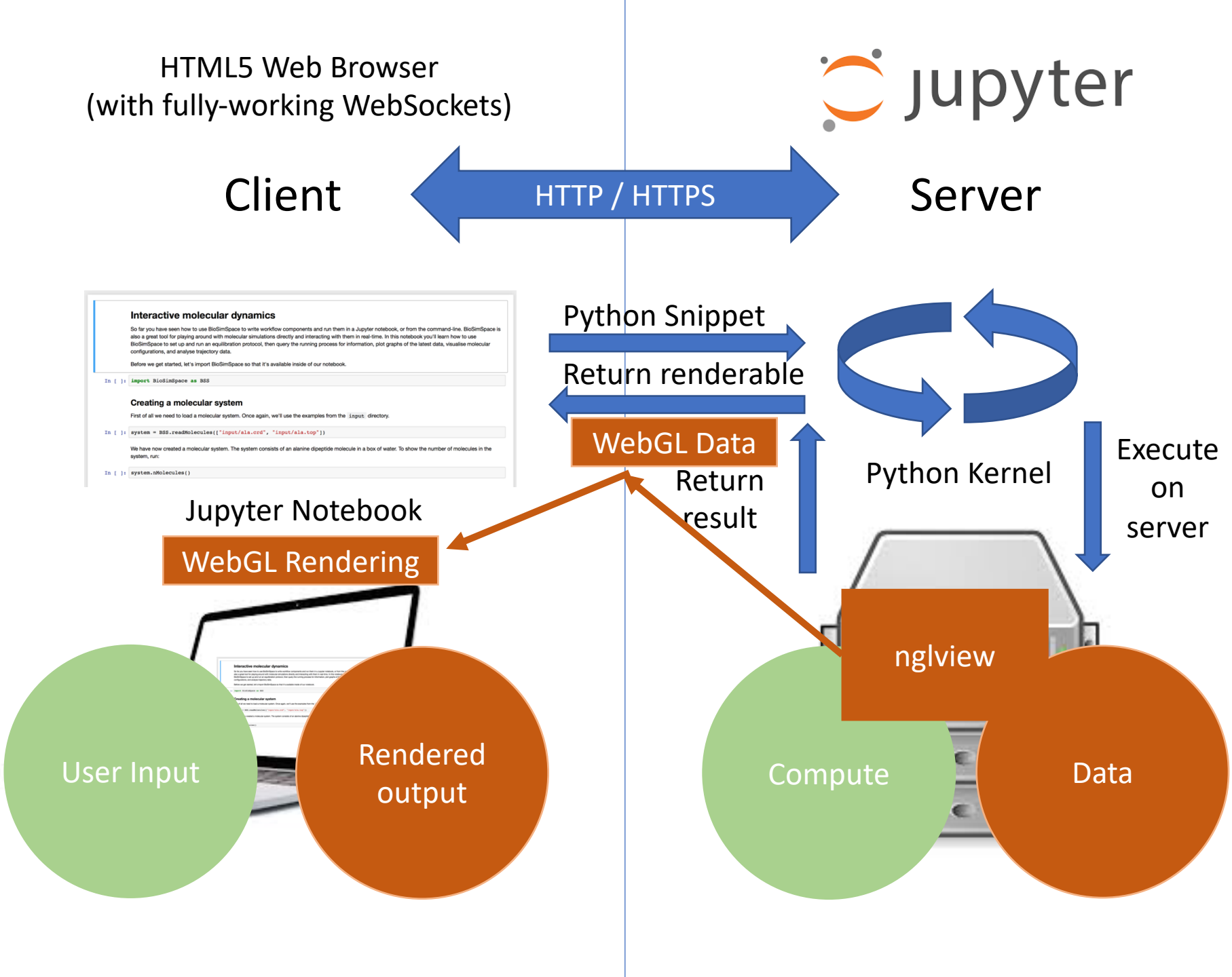
Return result

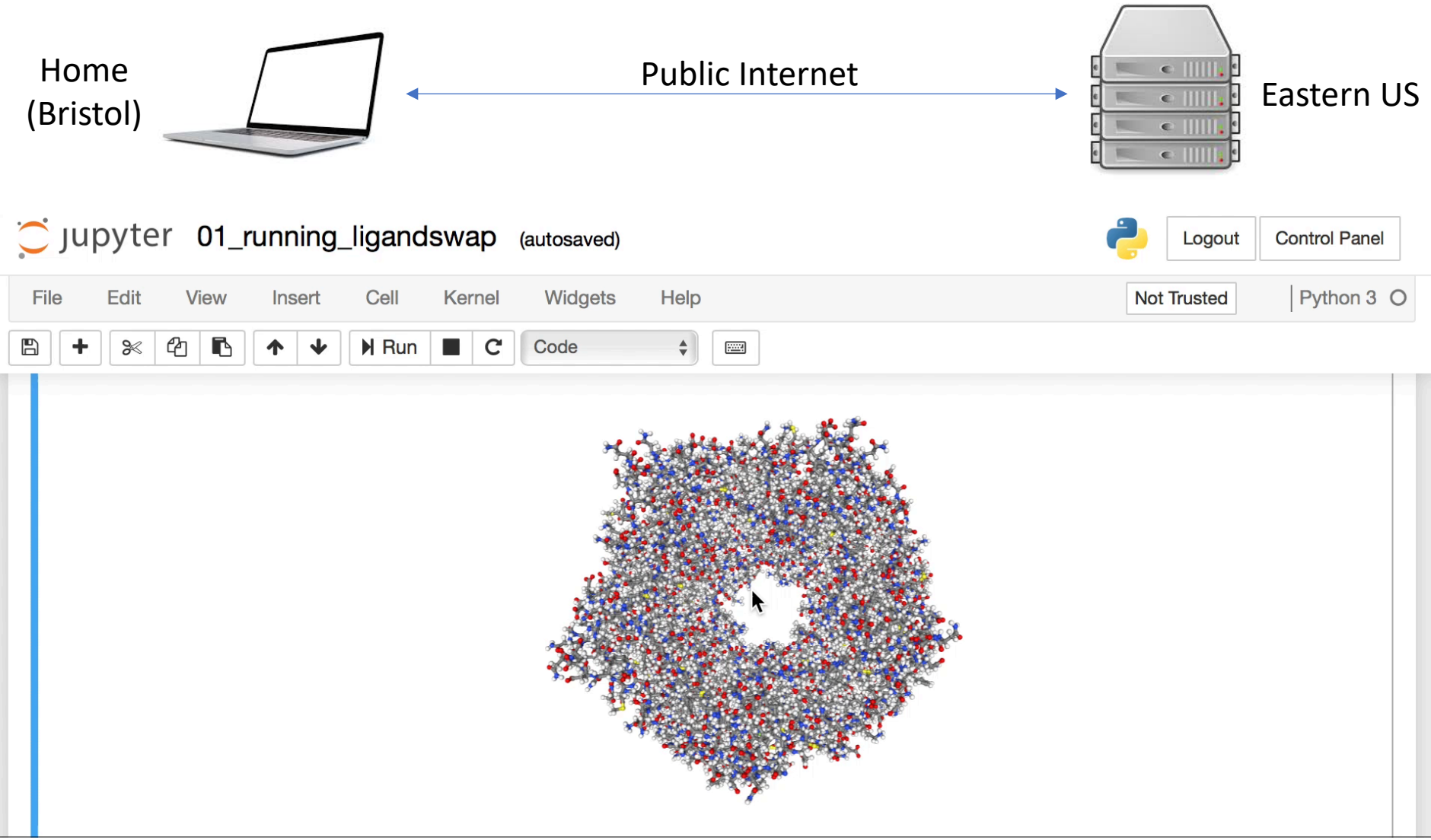
Python Kernel

Execute on server






HTML5 Web Browser
(with fully-working WebSockets)
jupyter
Client
HTTP / HTTPS
Server
Interactive molecular dynamics
Creating a molecular system
Python Snippet
Return renderable
WebGL Data
Return result
Python Kernel
Execute on server
Jupyter Notebook
WebGL Rendering
nglview
User Input
Rendered output
Compute
Data

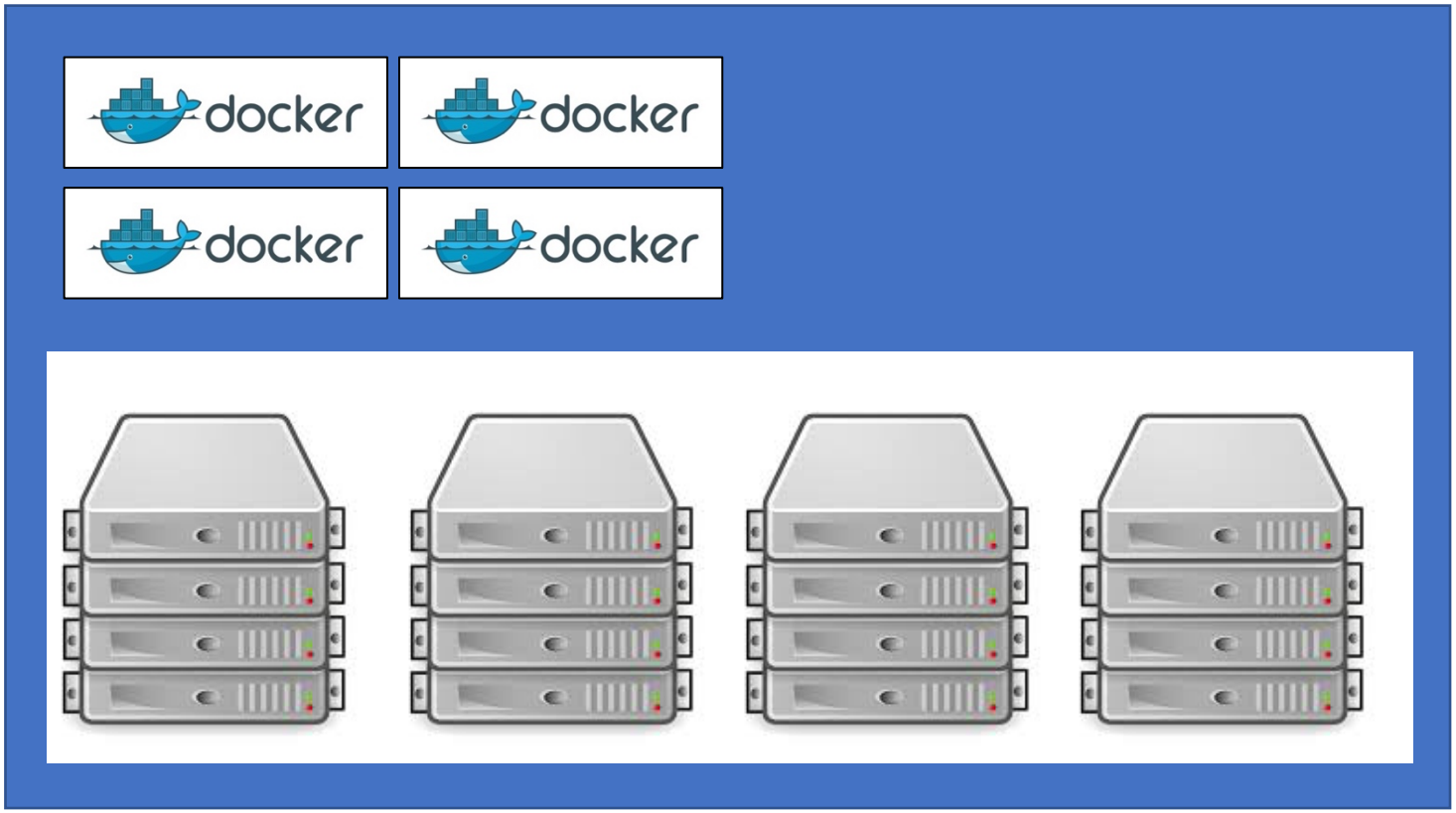


Data transfer limited to what is needed to actually render the visualization (e.g. WebGL data)

All interaction and rendering performed in the browser, making it smooth and lag-free

BioSimSpace on Kubernetes

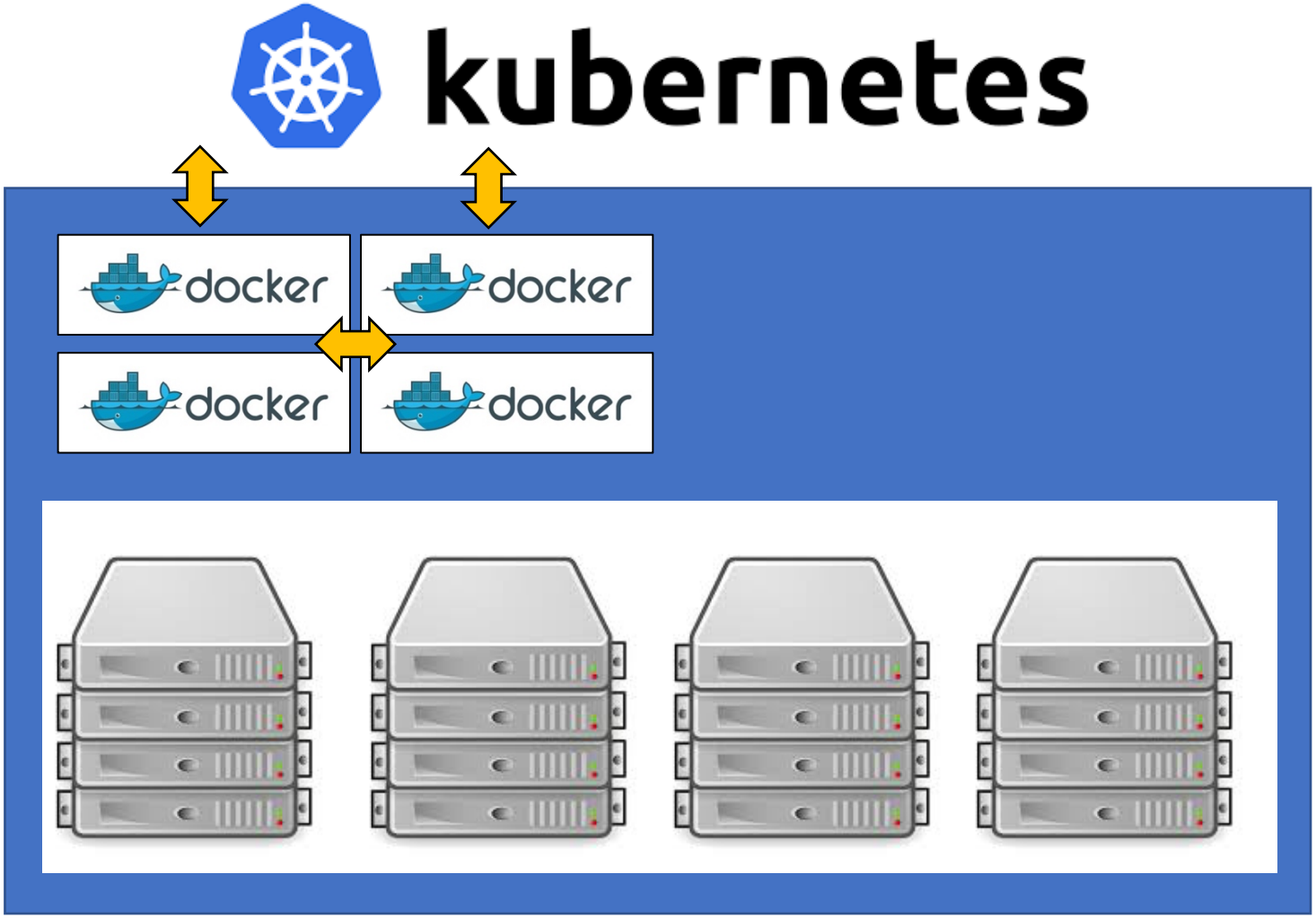
kubernetes



Kubernetes is a container orchestrator

Dynamically allocates containers to servers

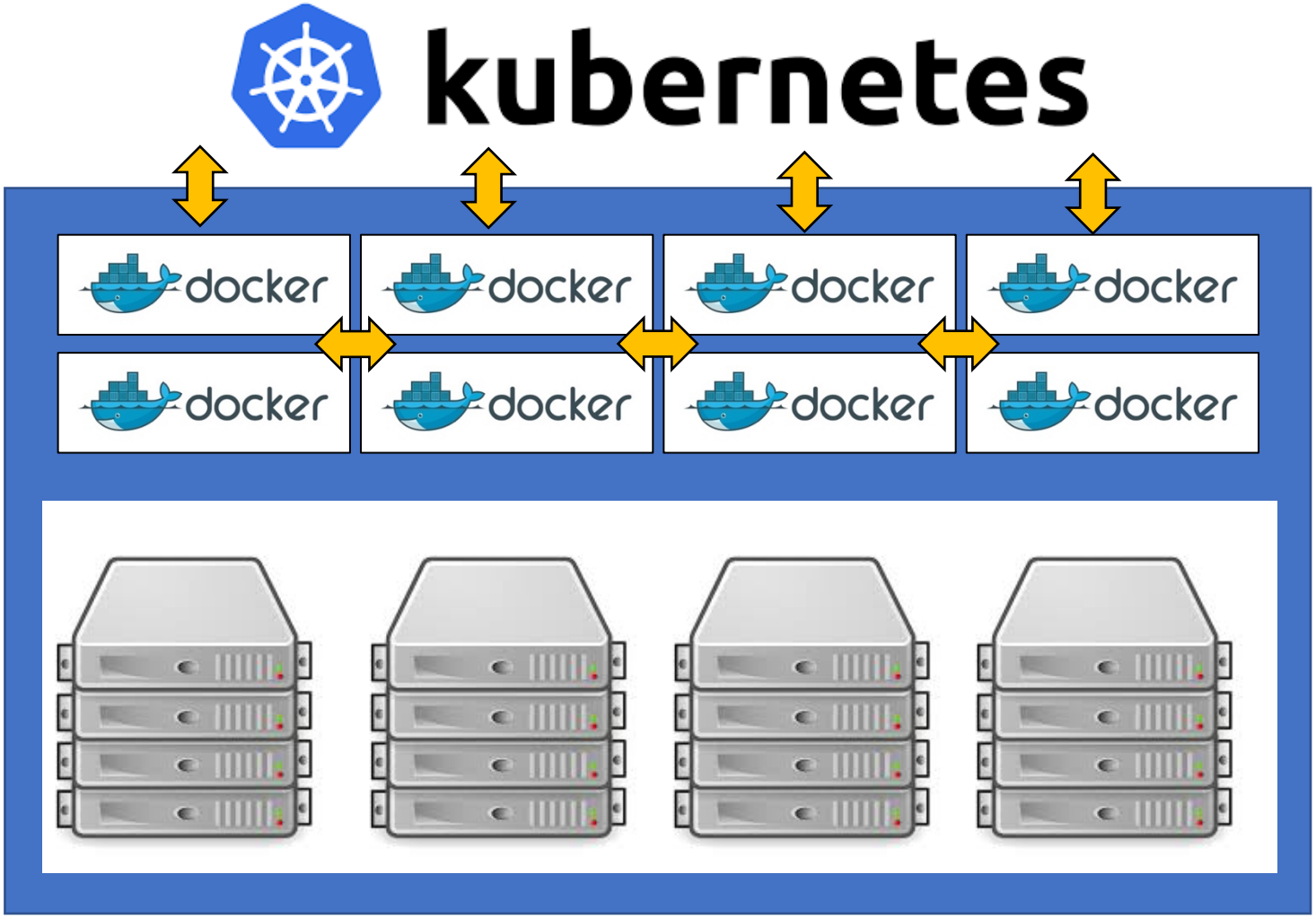
A container running on a server is a called a pod



Kubernetes is a container orchestrator

Pods are networked together using named services

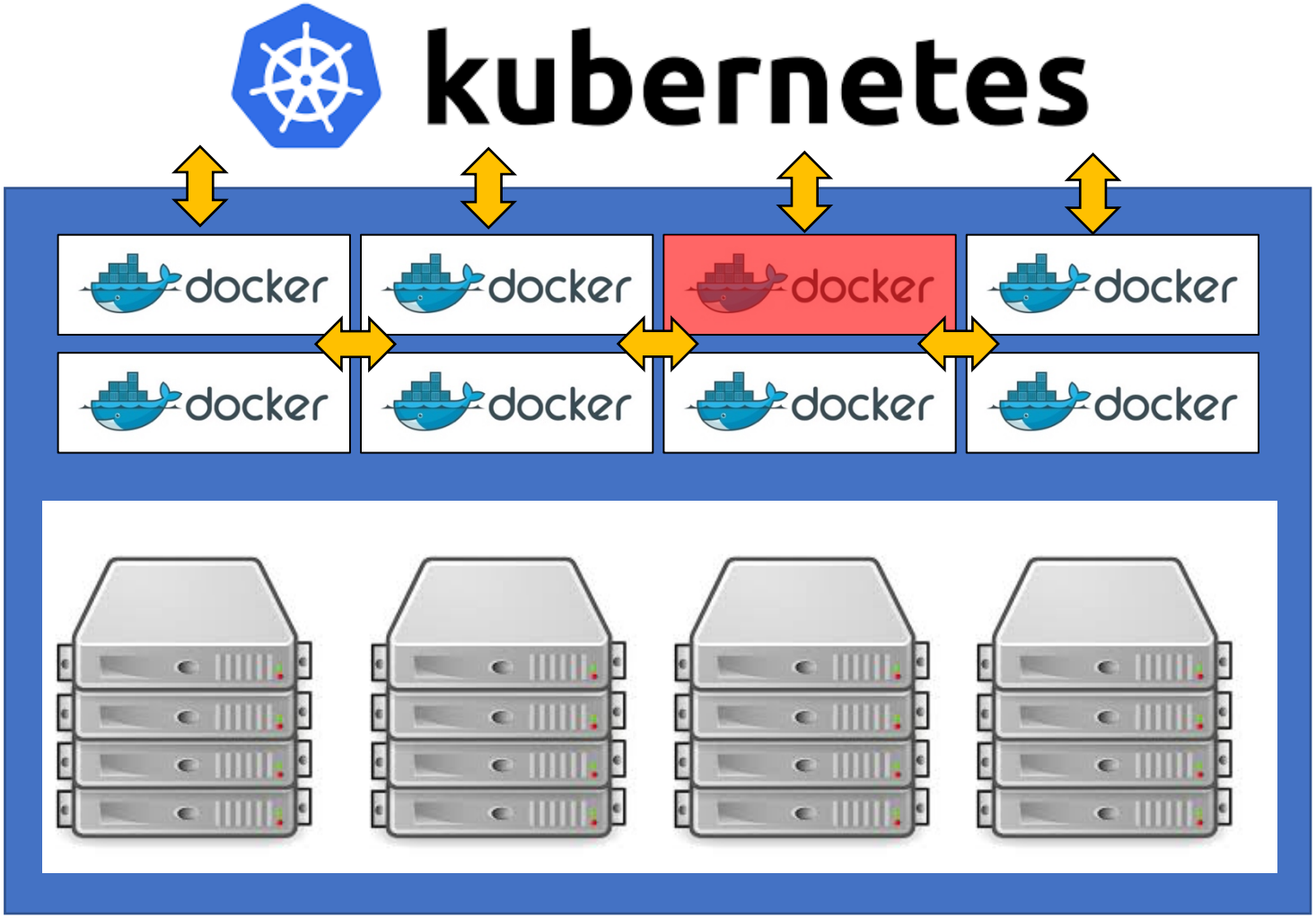
Services can be made visible outside the cluster using a LoadBalancer



Kubernetes is a container orchestrator

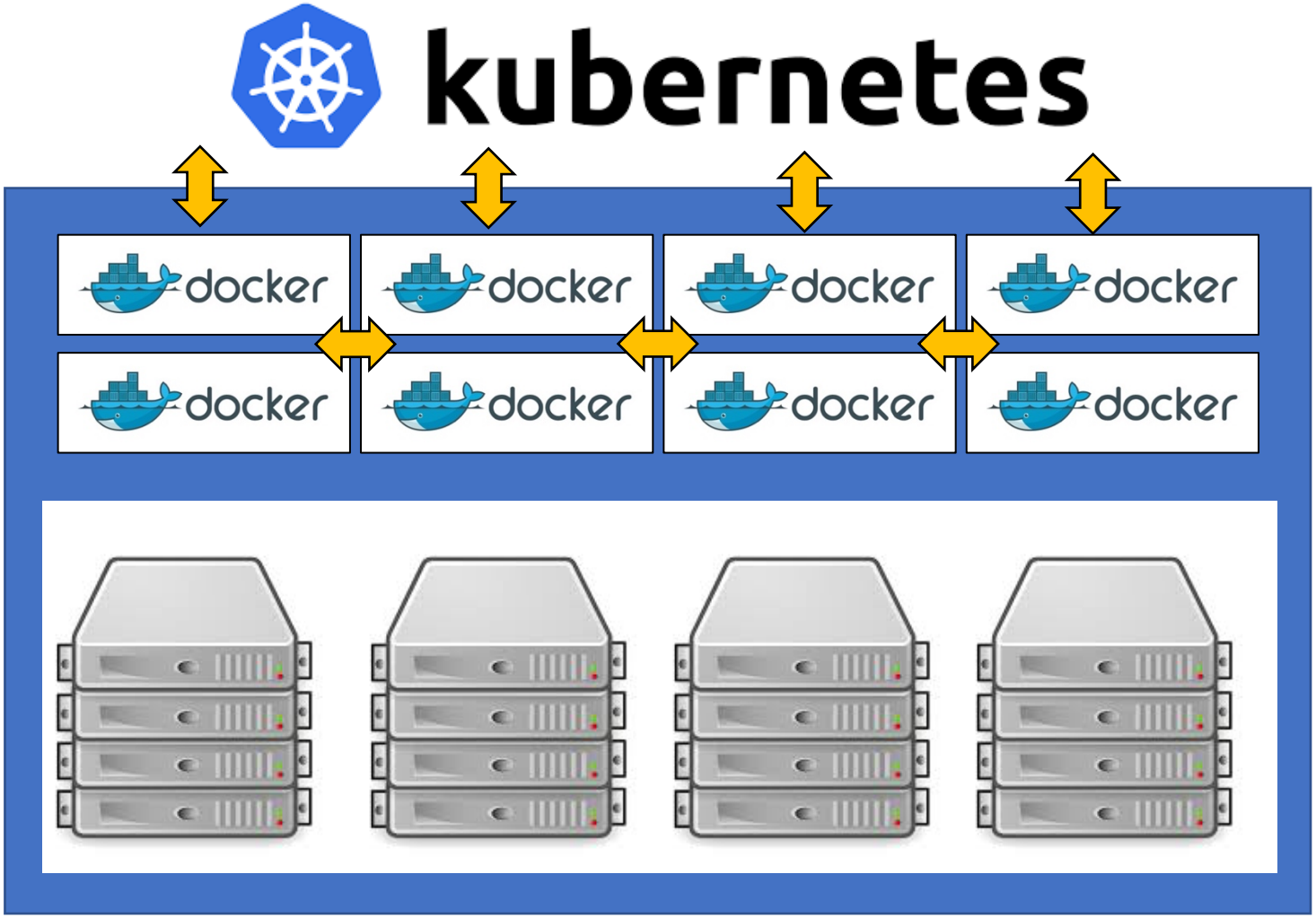
If demand for services increases, then more pods are spawned

If demand for services decreases, then pods are destroyed



Kubernetes is a container orchestrator

If one of the pods fails, or goes silent, then it is automatically killed and restarted. Can also do in-place upgrades and A/B testing



Kubernetes is a container orchestrator

Essentially, Kubernetes is an on-demand “scheduler for containers”

Client(s)

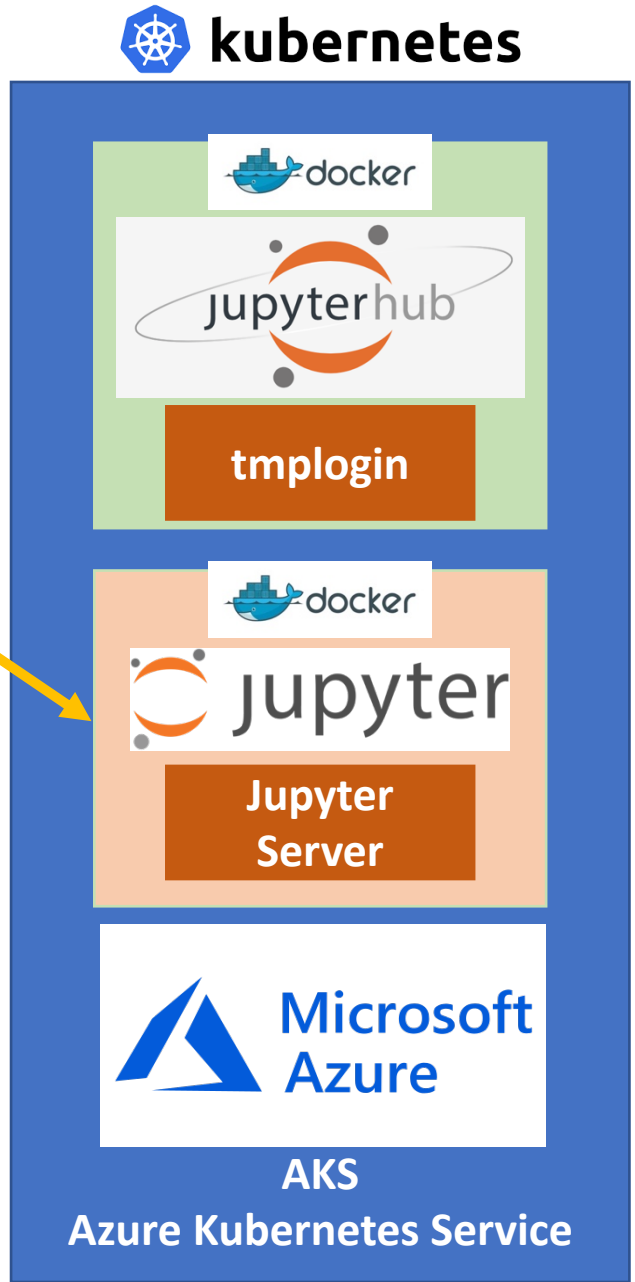
workshop.biosimspace.org

https

Run JupyterHub/Jupyter on the Azure Kubernetes Container Service in the Cloud.

Use “tmplogin” authenticator so anyone can connect without a password. Spawns single-use containers with custom docker image

Cost to support up to 60 simultaneous users is £11 per day (£4152 per year)



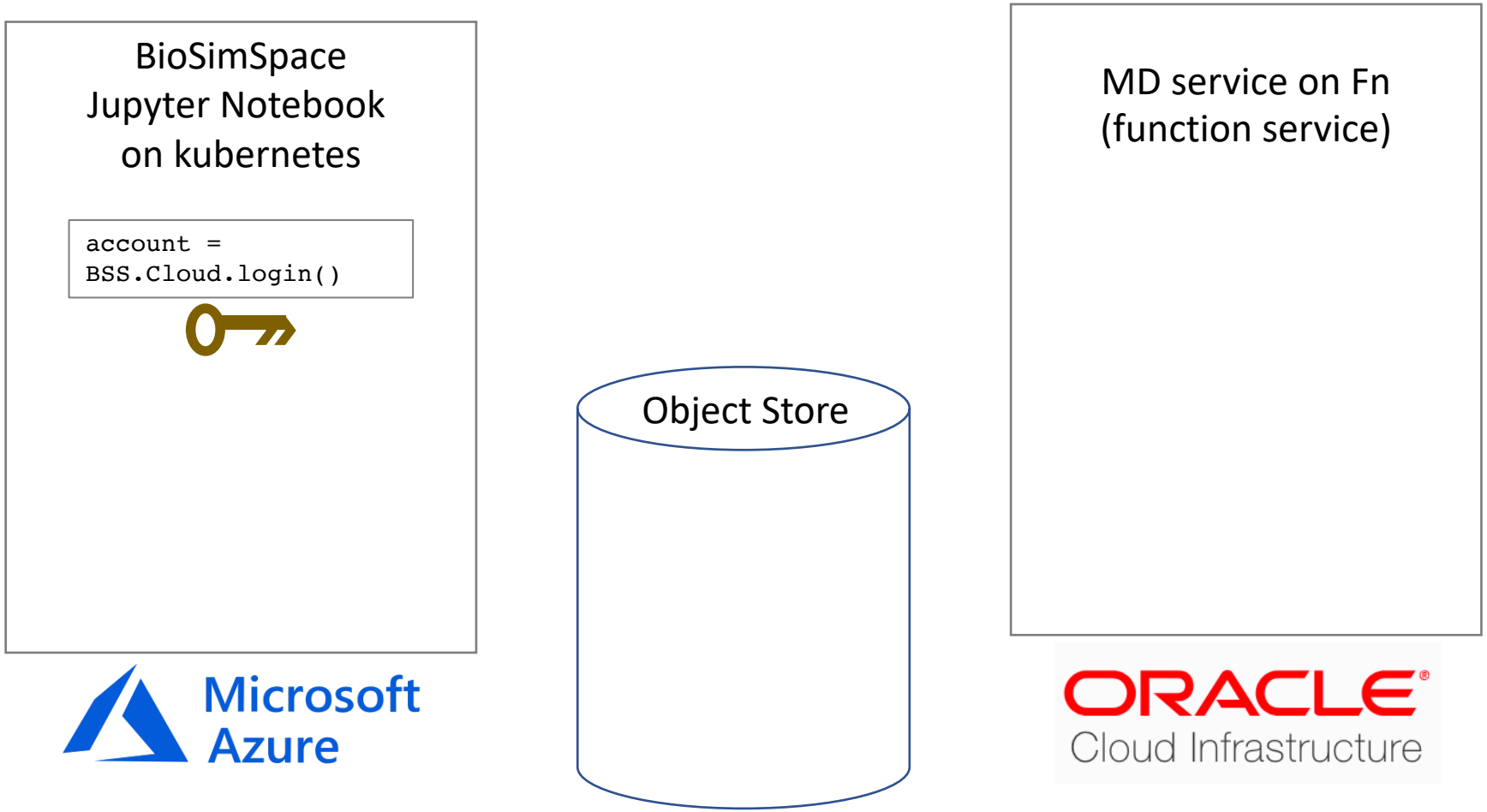
BioSimSpace on the Cloud

(*who pays?)

BioSimSpace Cloud

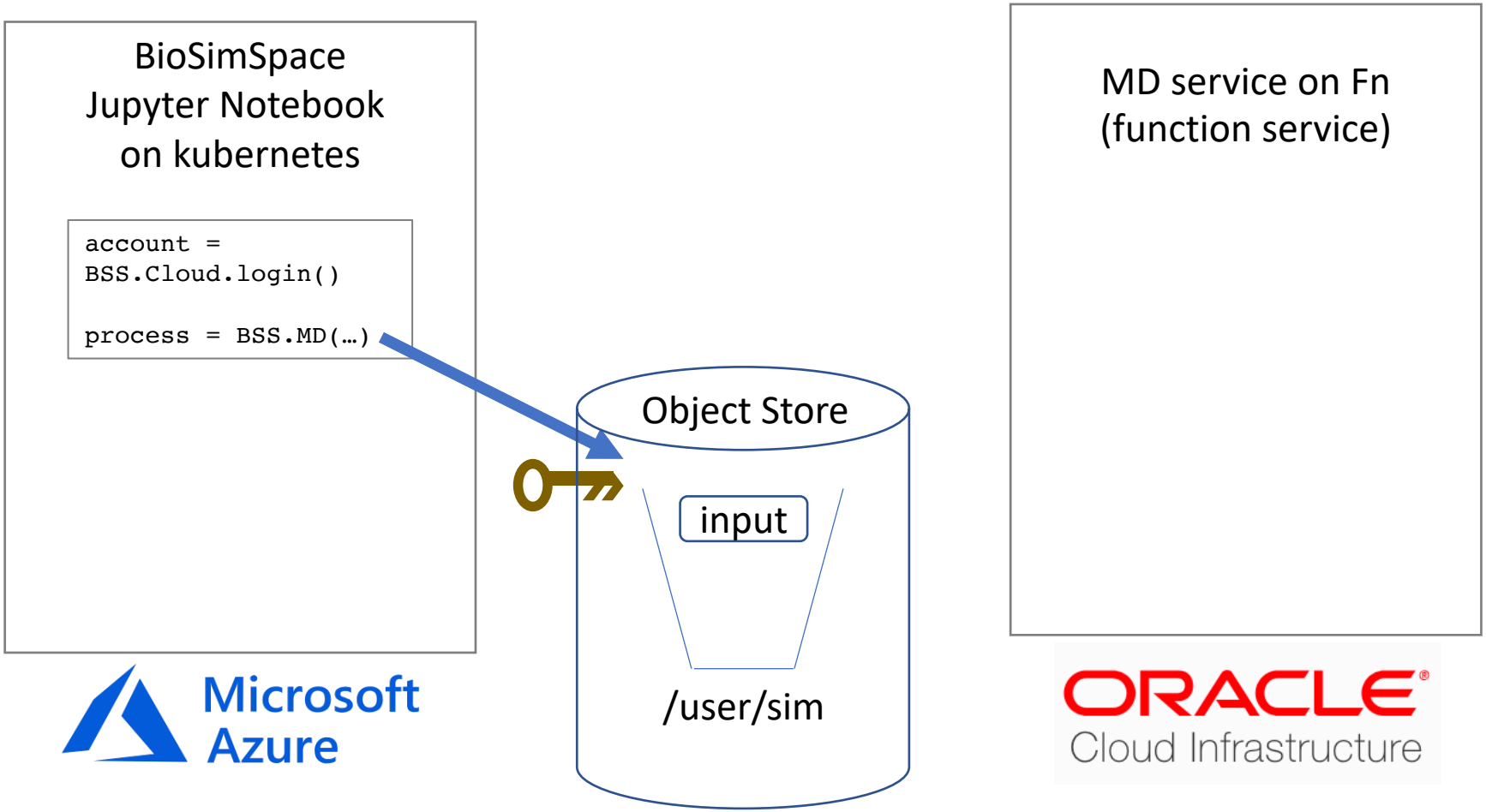
- Partnership with **Microsoft Azure** and **Oracle Cloud Infrastructure**
- Aim to allow **upfront charging** of cloud compute and storage costs on a **"per simulation"** basis
- Replaces current low-level charging based on VMs, network, disk etc.
- **BioSimSpace can estimate compute and storage requirement of a simulation and will present an up-front guaranteed cost to run the calculation**
- Users can set **daily caps** and **maximum runtimes**
- **Best resource that fits the caps will be automatically chosen**

```python
import BioSimSpace as BSS

account = BSS.Cloud.login()
account.setMaxDailySpend("£10")
account.setMaxRunTime("1 week")

process = None
system = # load the system
protocol = # define the simulation protocol

try:
    process = BSS.MD.run( system, protocol )

except BSS.Cloud.CostBreakConstraintsError as e:
    permission = account.emailRequestForPermission(e)

    if permission.granted():
        process = BSS.MD.run( system, protocol, account.grant(permission) )

except BSS.Cloud.InsufficientFundsError as e:
    account.emailRequestForFunds(e)

    if account.waitForFunds(e.requiredCost(), "48h"):
        process = BSS.MD.run( system, protocol )

if not process:
    print("Cannot run the simulation as insufficient cloud funds!")
```

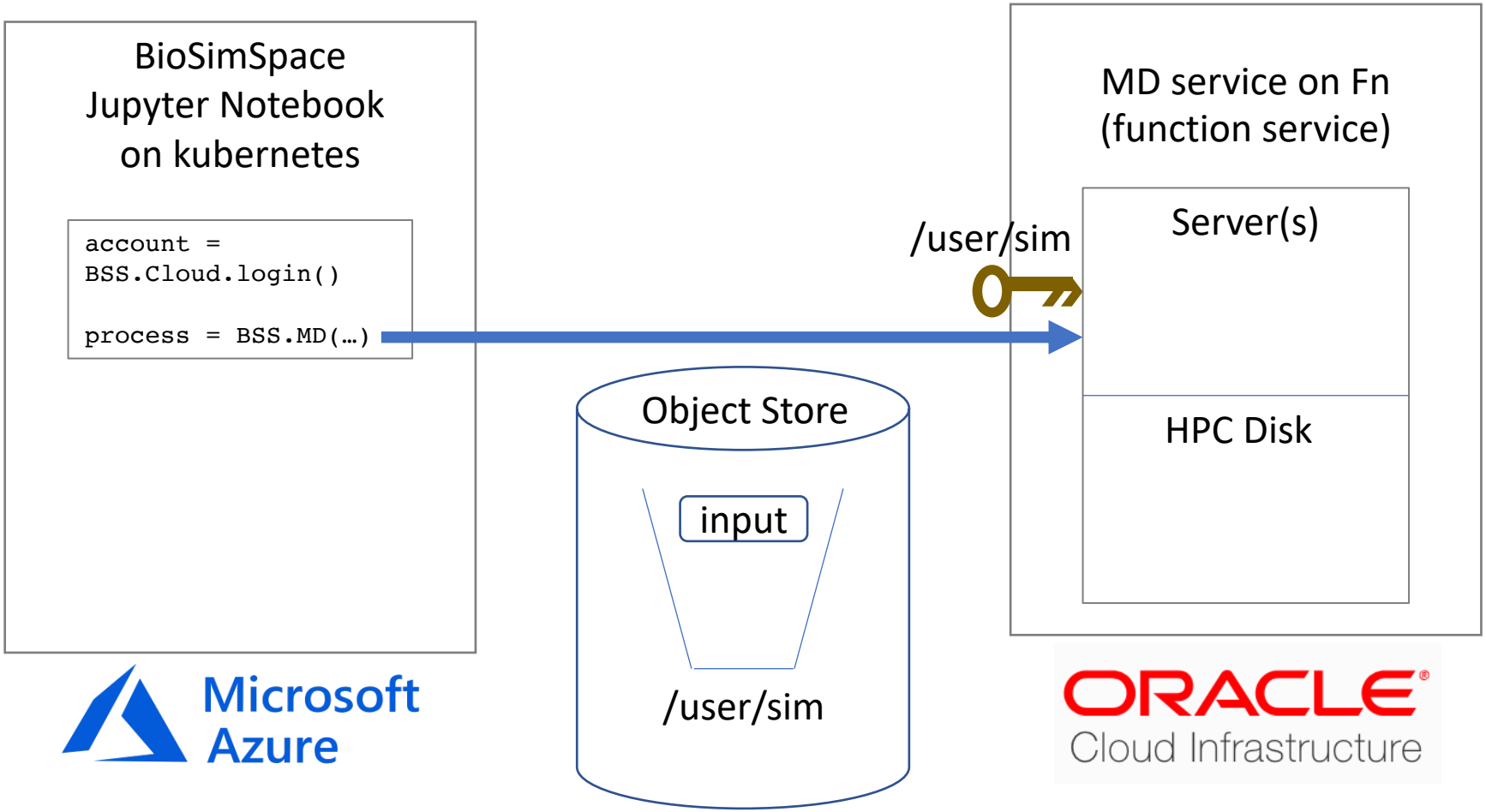


User logs into their cloud account from within the BioSimSpace Jupyter notebook.

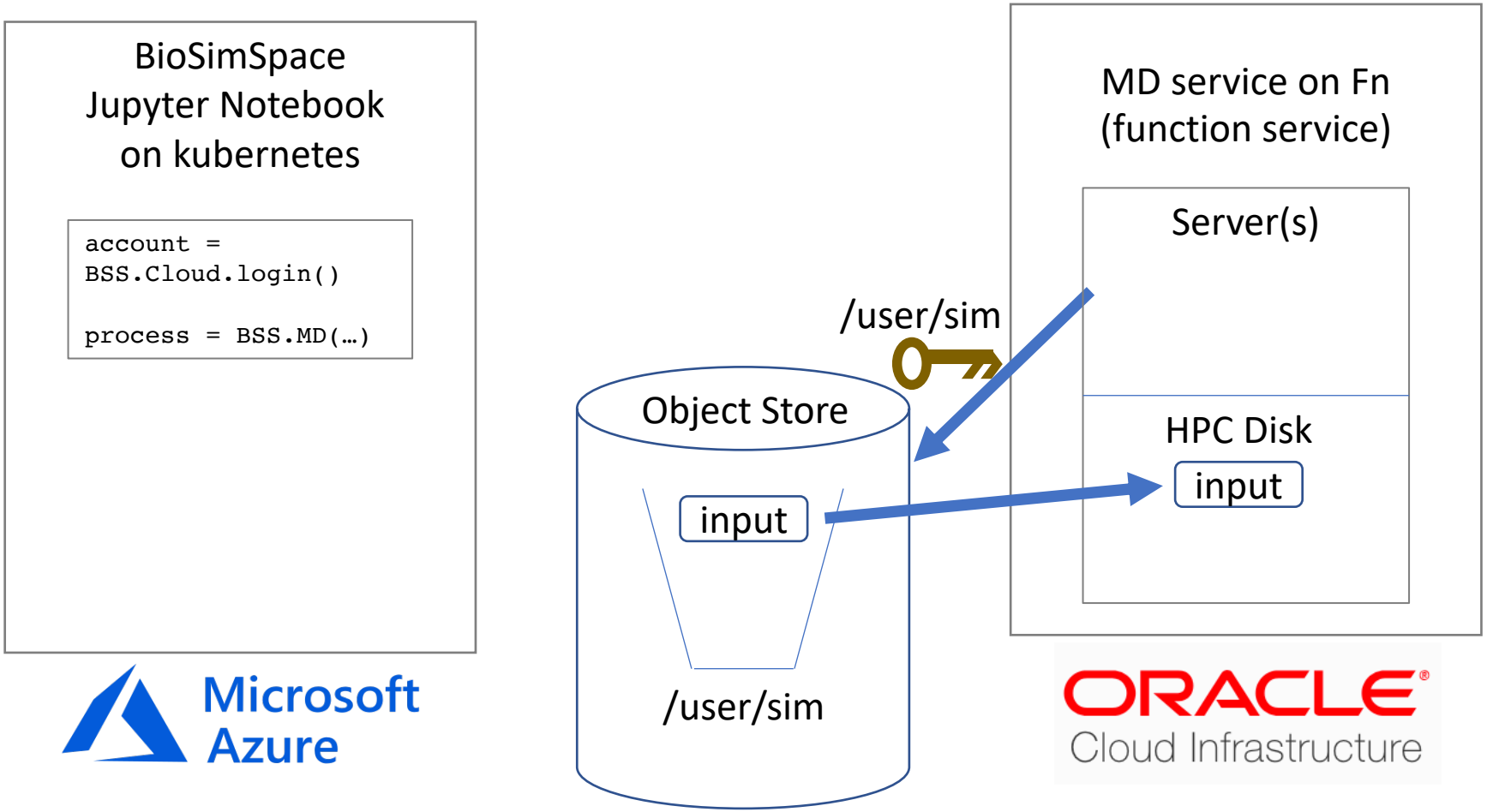
This returns a key that can be used to authenticate the user with other services.



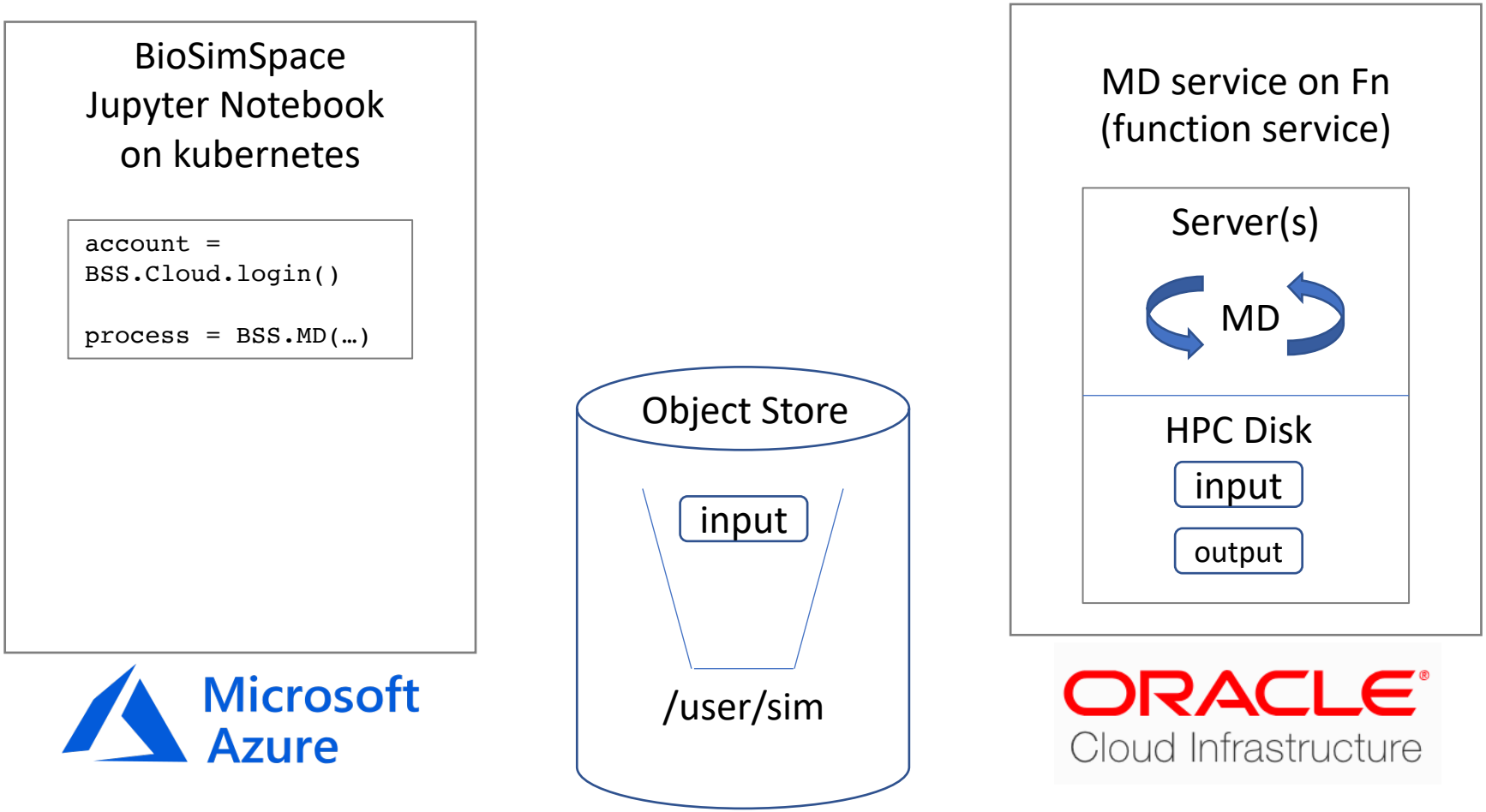
To run the simulation, BioSimSpace will use the key to authenticate with an object store. The input simulation data will be copied into a bucket for the simulation within this object store, e.g. /user/sim/input



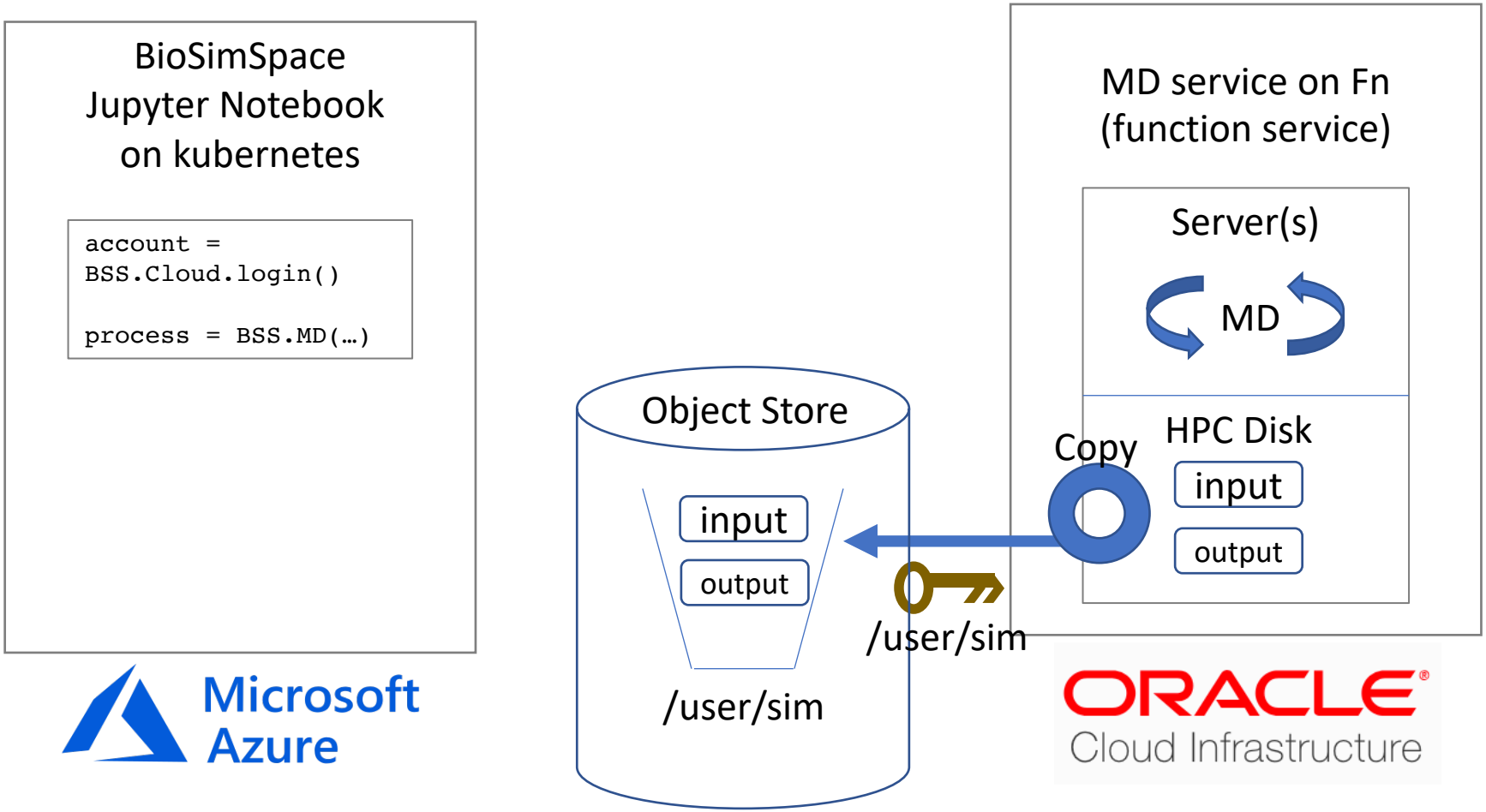
Next, BioSimSpace will use the key to authenticate with a serverless function service (e.g. Fn running on Oracle). This will automatically provision a fast server connected to a fast disk that will be used to run the simulation. BioSimSpace supplies the server with the authentication key needed to access the object store.



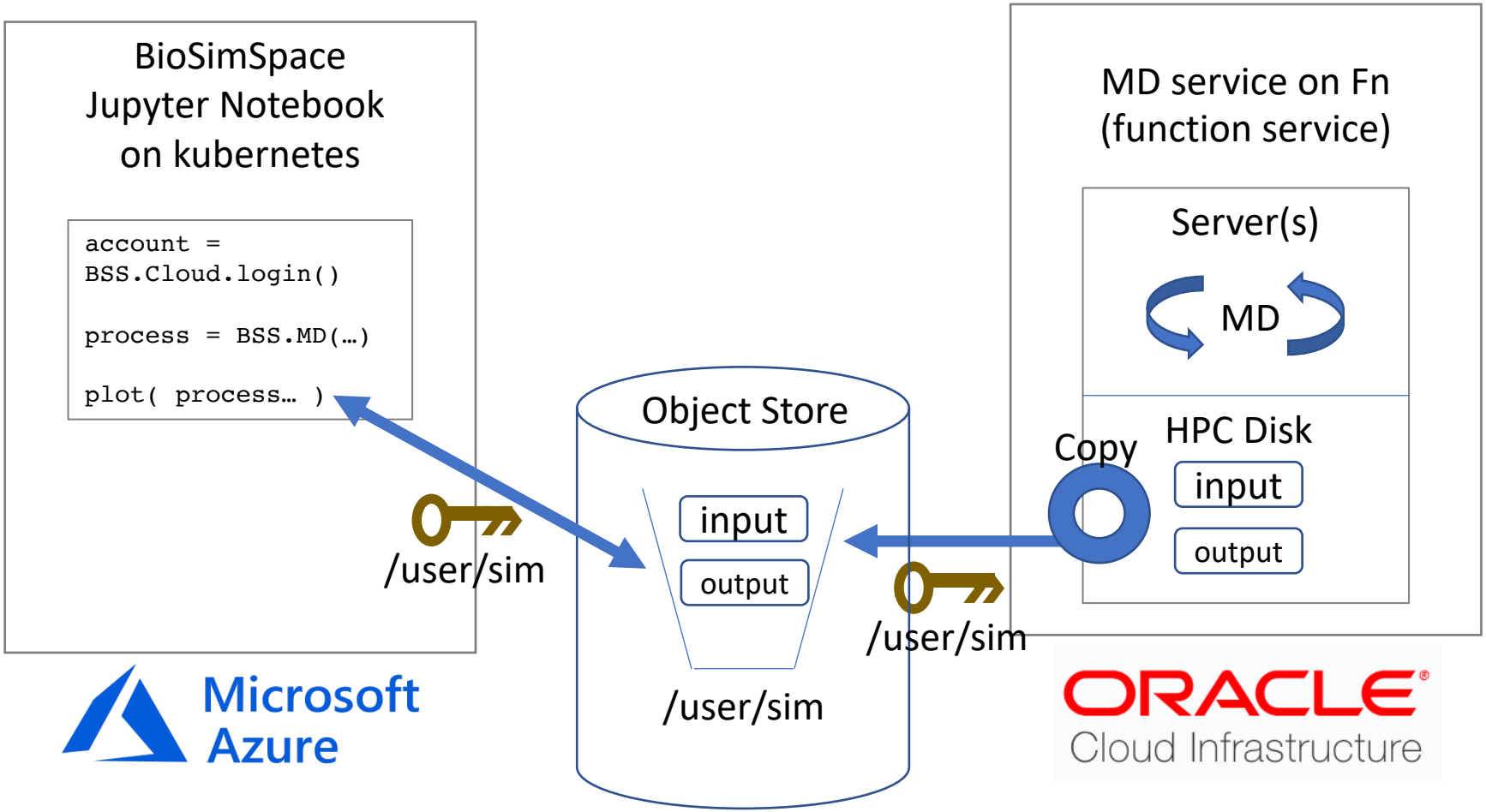
The function server uses the authentication key to copy data from the /user/sim bucket in the object store to the fast (posix) disk which will be used for the simulation.



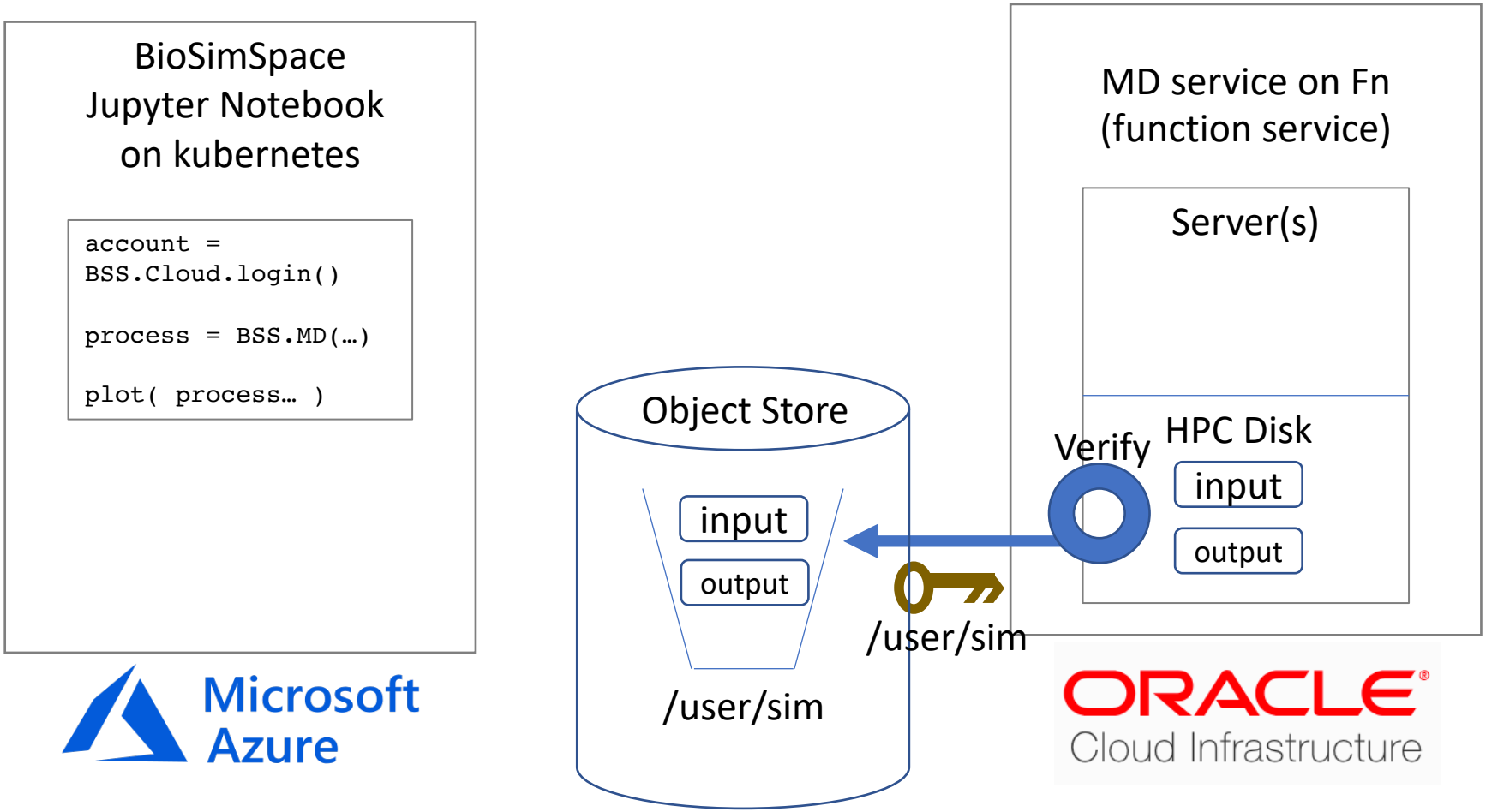
The function server runs the MD simulation, writing output to the fast (posix) disk.



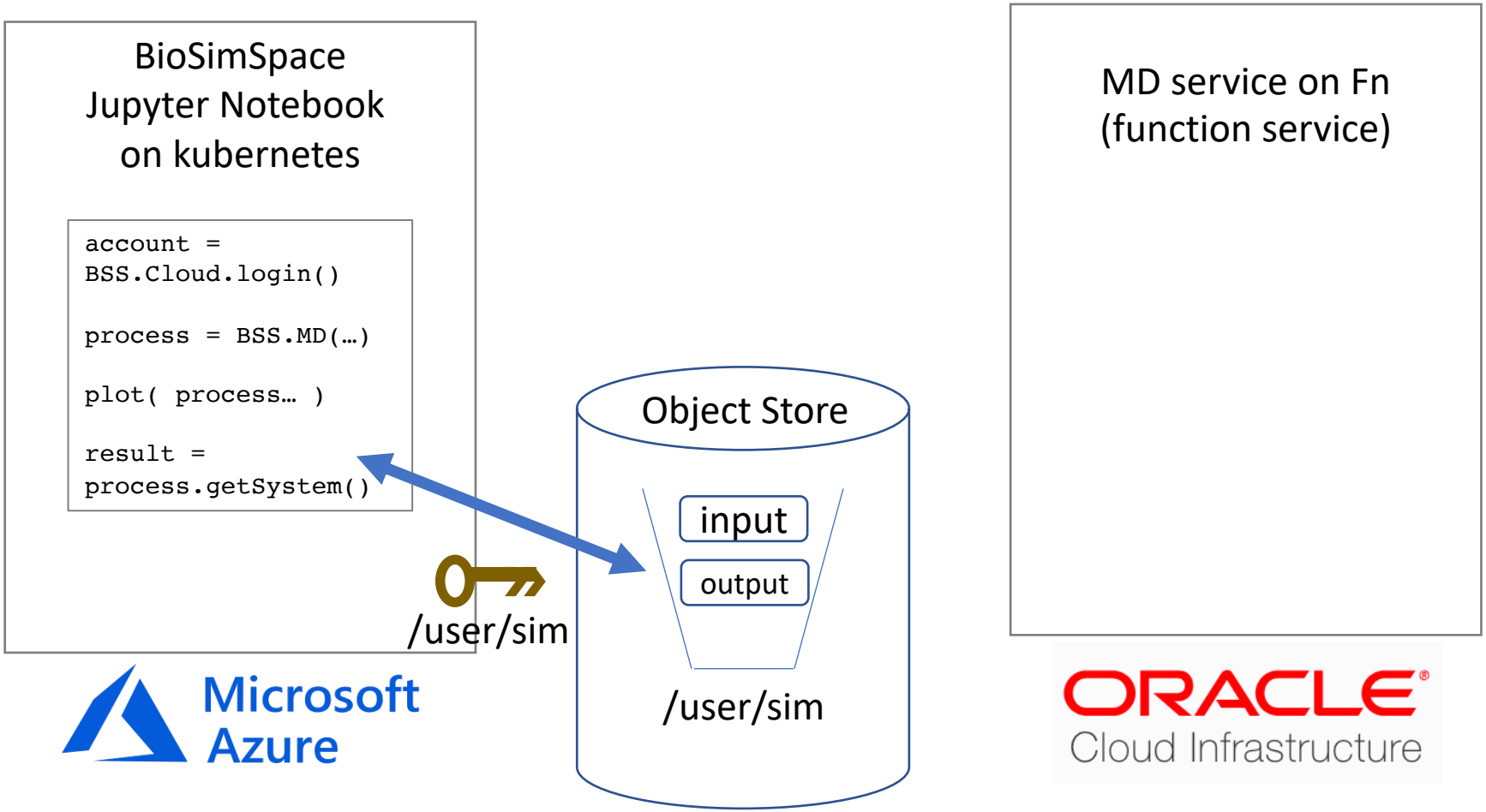
As the simulation is running a “copy service” copies files output from the simulation back to an “output” key in the /user/sim bucket on the object store. This authenticates using the key originally supplied by the BioSimSpace from the notebook



At any time while the simulation is running BioSimSpace in the notebook can use the object store authentication key to query the output, and thus plot graphs, extract energies or plot 3D views of the molecules as they are being simulated



Once the simulation has finished on the function server, the copy service verifies that all output data has been successfully copied to the object store (e.g. by comparing checksums and object sizes). Once all output data has been copied and verified, the function server shuts down, with all data removed from the HPC disk

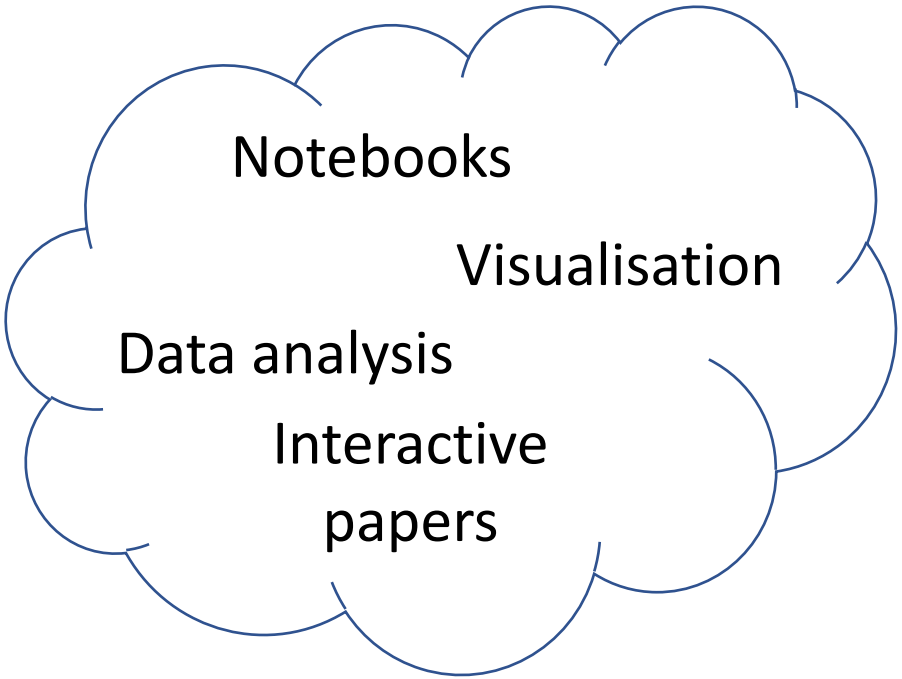


The user interacting with BioSimSpace running in the Jupyter server can query and analyse the results using the authentication key and path to the output data in the object store (e.g. /user/sim/output). The data in the object store is write-protected, so that it can be safely re-used by other scripts without fear of modification or deletion

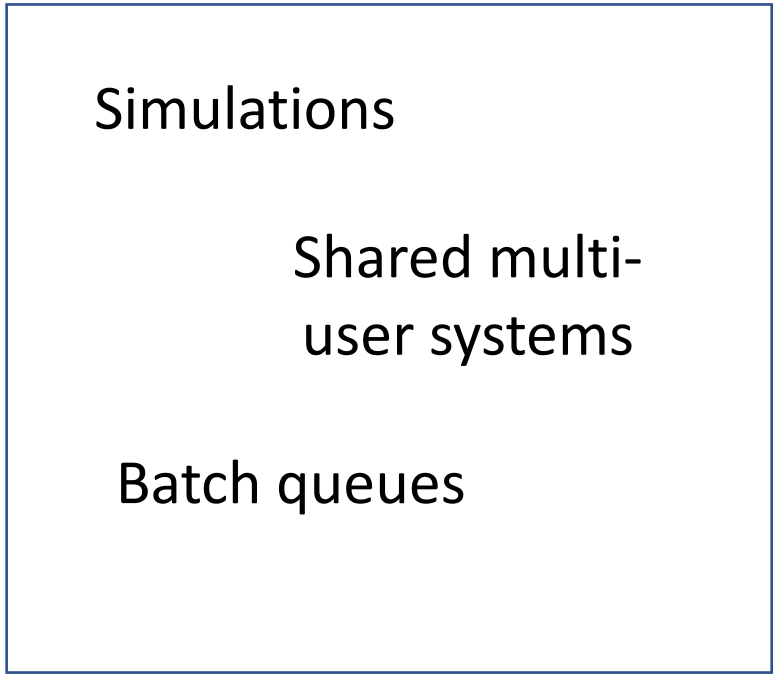
Simulation Output

- Outputs will be **read-only** and up-front **costs covers one year's storage** in the object store
- Object store key can become a **DOI** that allows them to be **accessed from other scripts**, or **published and accessed by others**
- Web console will allow researchers to manage outputs, e.g. **control access permissions**, **delete the output** (receive a pro-rata storage refund), pay of extra years storage, or pay a **one-off charge for the output to be archived** (15 years)

Demand versus Batch Computing



Demand Computing



Batch Computing

Demand versus Batch Computing

Notebooks

Visualisation

Data analysis

Interactive papers

Demand Computing

Simulations

Shared multi-user systems

Batch queues

Batch Computing

NETFLIX

of simulation

Demand versus Batch Computing

Notebooks

Visualisation

Data analysis

Interactive papers

Demand Computing

Simulations

Shared multi-user systems

Batch queues

Batch Computing

NETFLIX

of simulation

How to handle user accounts on multiple systems; movement of data; custom docker images; usage/cost accounting

Acknowledgements

- BioSimSpace Team
  - **Lester Hedges, Antonia Mey**, Julien Michel, Adrian Mulholland, Charlie Laughton, Francesco Gervasio
- EPSRC for funding (EP/P022138/1)
- CCP-BioSim and HEC-BioSim for support
- Microsoft – in particular Kenji Takeda
- Oracle – in particular Phil Bates and Gerardo Viedma
- BioExcel – for inviting me and hosting this webinar

EPSRC
Engineering and Physical Sciences Research Council

https://biosimspace.org
https://chryswoods.com/talks

University of BRISTOL

Audience Q&A session

Please use the **Questions** function in GoToWebinar application

Any other questions or points to discuss after the live webinar? Join the discussion the discussion at http://ask.bioexcel.eu.