ECMWF Talk – March 2019

Running Serverless HPC Workloads on Top of Kubernetes and Jupyter Notebooks

Christopher Woods

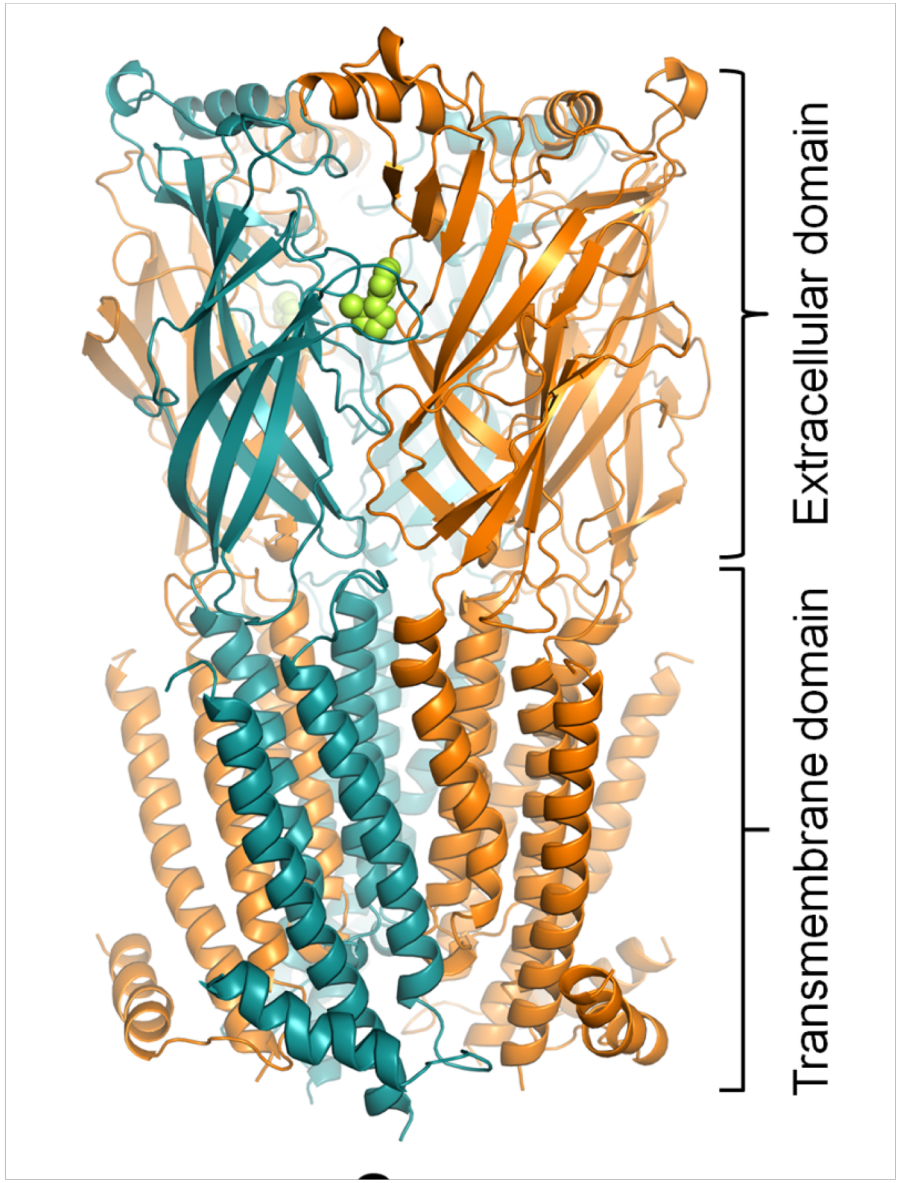
EPSRC Research Software Engineering Fellow

Advanced Computing Research Centre

University of Bristol

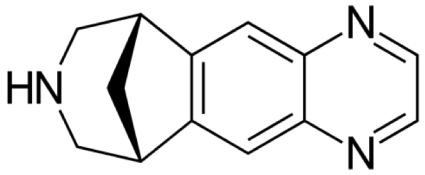
Part 0
Cluster on Premise

https://www.flickr.com/photos/61423903@N06/27720754261

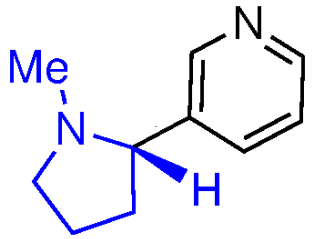


Nicotinic Receptor
(Ligand Gated Ion Channel)

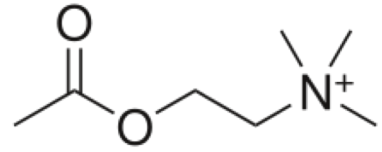
Important drug target for treatment of Alzheimer's, Schizophrenia, Pain etc.



Varenicline



Nicotine



Acetylcholine

To understand this protein, we need to use computer simulations to ask how does the receptor move?

How does the signal move from one side of the receptor to the other?

How would we run these simulation using our on-premise cluster?

1. SSH to login node

2. Upload input (rsync)

3. Submit job to Q

4. Wait in Q

5. Wait for job

6. Analyse results

5. Download output (rsync)

Part 1
Cluster in the Cloud

Cloud Login / management VM

Login (ssh)

https://cluster-in-the-cloud.readthedocs.io

ORACLE
Cloud Infrastructure

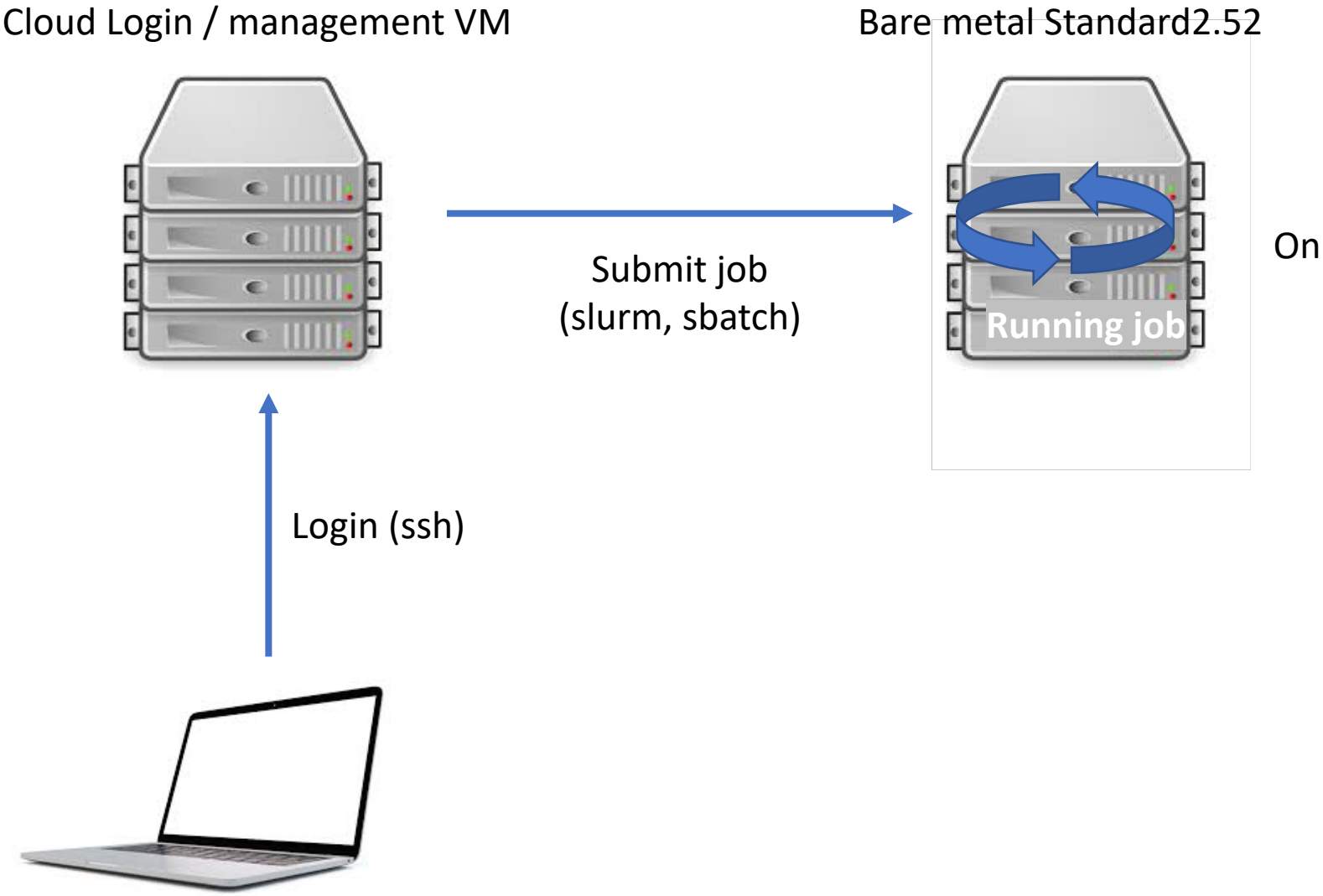
Cloud Login / management VM

Submit job
(slurm, sbatch)

Login (ssh)

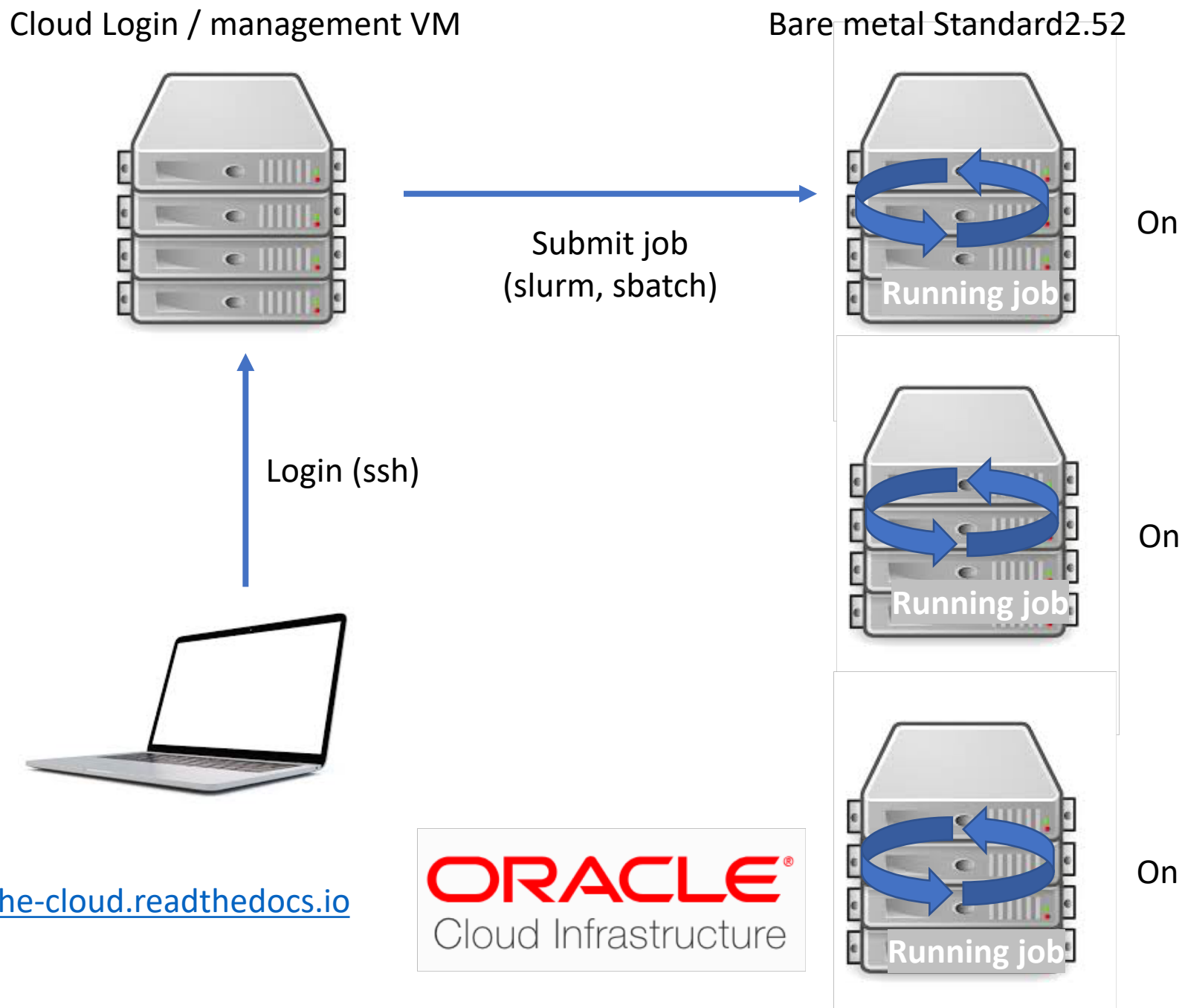
https://cluster-in-the-cloud.readthedocs.io

ORACLE
Cloud Infrastructure



https://cluster-in-the-cloud.readthedocs.io

ORACLE
Cloud Infrastructure


Cloud Login / management VM
Bare metal Standard2.52
Submit job
(slurm, sbatch)
Running job
On
Running job
On
Running job
On
Login (ssh)
https://cluster-in-the-cloud.readthedocs.io
ORACLE
Cloud Infrastructure

Cloud Login / management VM

Bare metal Standard2.52

Submit job
(slurm, sbatch)

On

Login (ssh)

https://cluster-in-the-cloud.readthedocs.io

ORACLE
Cloud Infrastructure

On

Cloud Login / management VM

Login (ssh)

https://cluster-in-the-cloud.readthedocs.io

ORACLE
Cloud Infrastructure

Cloud Login / management VM

Cluster in the Cloud

Scales to the size of the job as needed

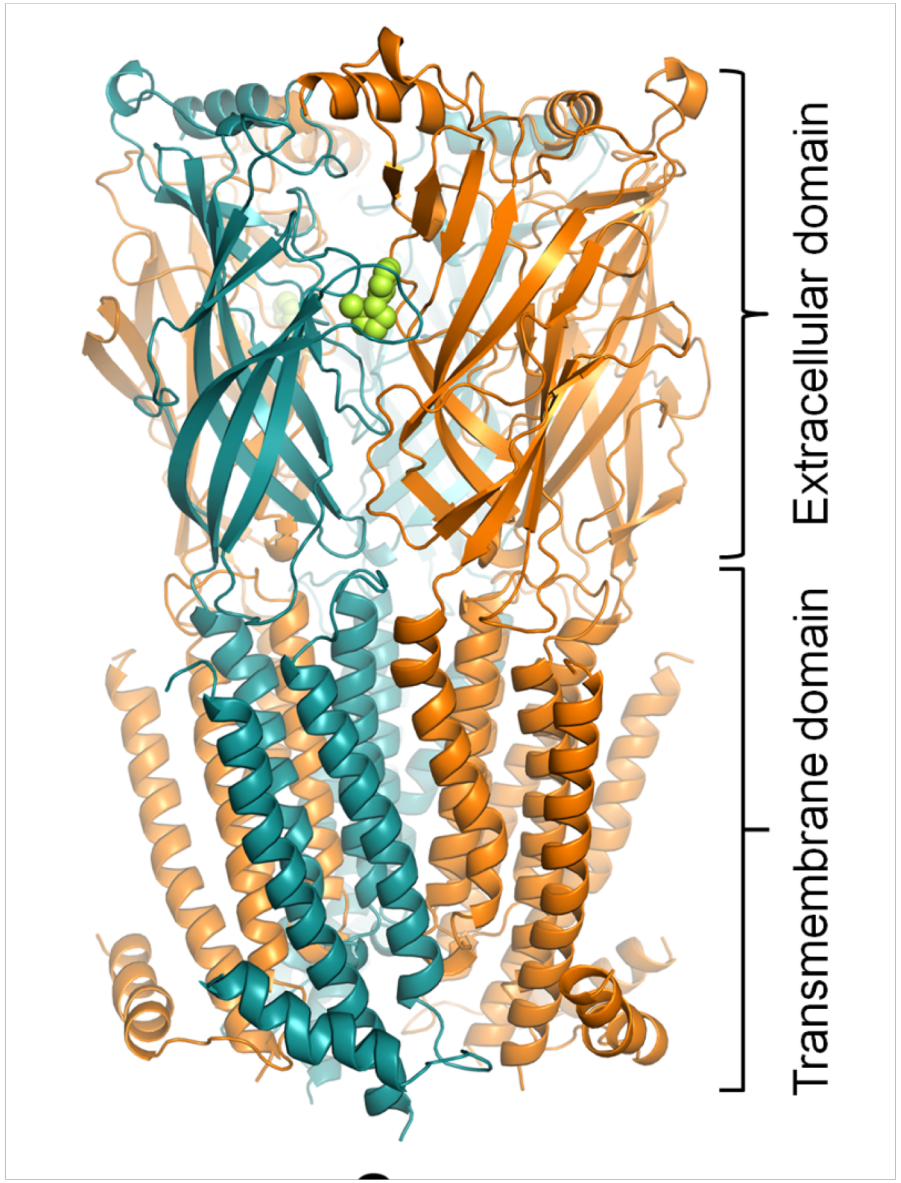
Only pay for what you use

Looks and feels like a “traditional” HPC cluster

Shallow learning curve for researchers

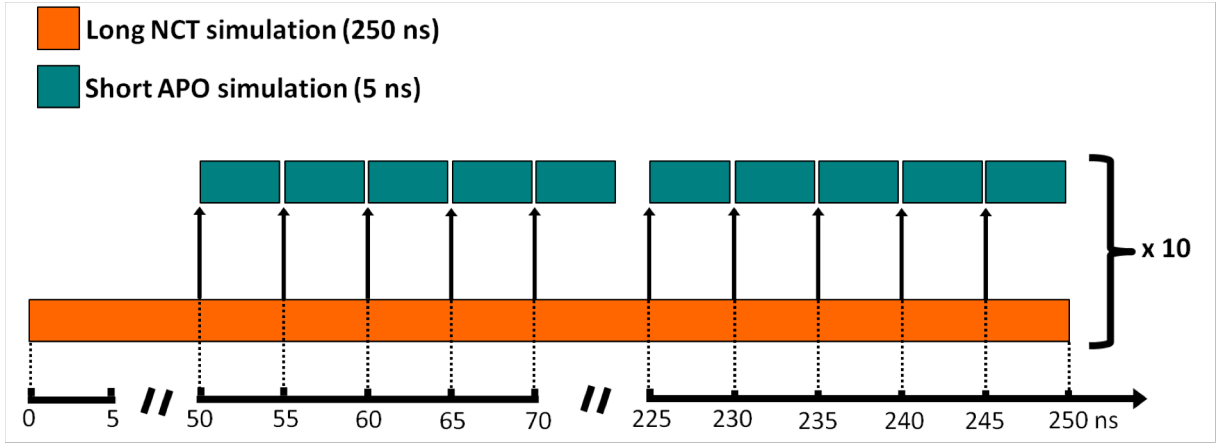
https://cluster-in-the-cloud.readthedocs.io

ORACLE
Cloud Infrastructure



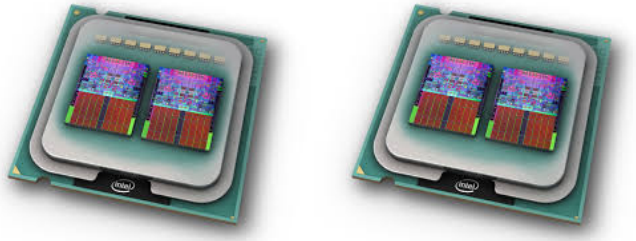
Nicotinic Receptor
(Ligand Gated Ion Channel)

Important drug target for treatment of Alzheimer's, Schizophrenia, Pain etc.



Investigate the signal conduction pathway via 1350 x 5ns molecular dynamics simulations (6.75µs total)

On-premise (BlueCrystal 4)



Installed early 2017
2 x Intel Xeon (Broadwell) 14 core

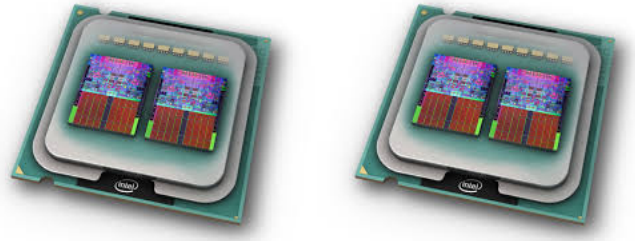
5ns (Gromacs) takes 17 hours

>530 compute nodes shared between ~1000 users

At most ~10 jobs could run at once

Calculation would take ~90 days

Oracle Cloud (BM.Standard2.52)



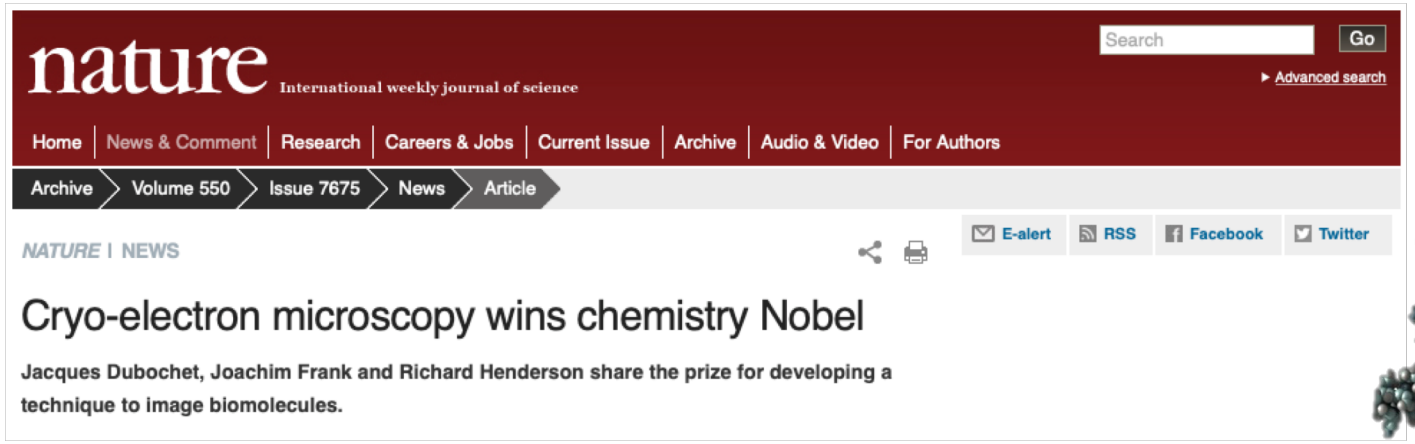
Bare Metal Node using latest Platinum Xeons
2 x Intel Xeon (Skylake) 26 core

5ns (Gromacs) takes 8 hours

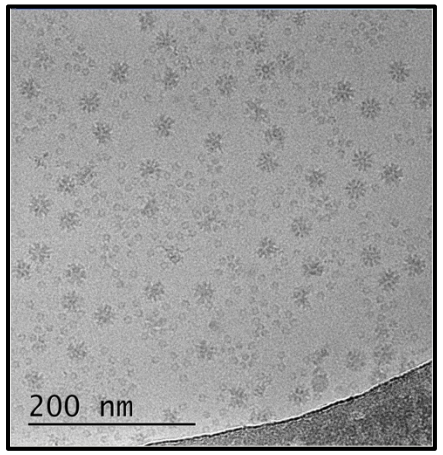
100 nodes available for 1 user

At most 100 jobs could run at once

Calculation took 5 days (Wed-Sun)

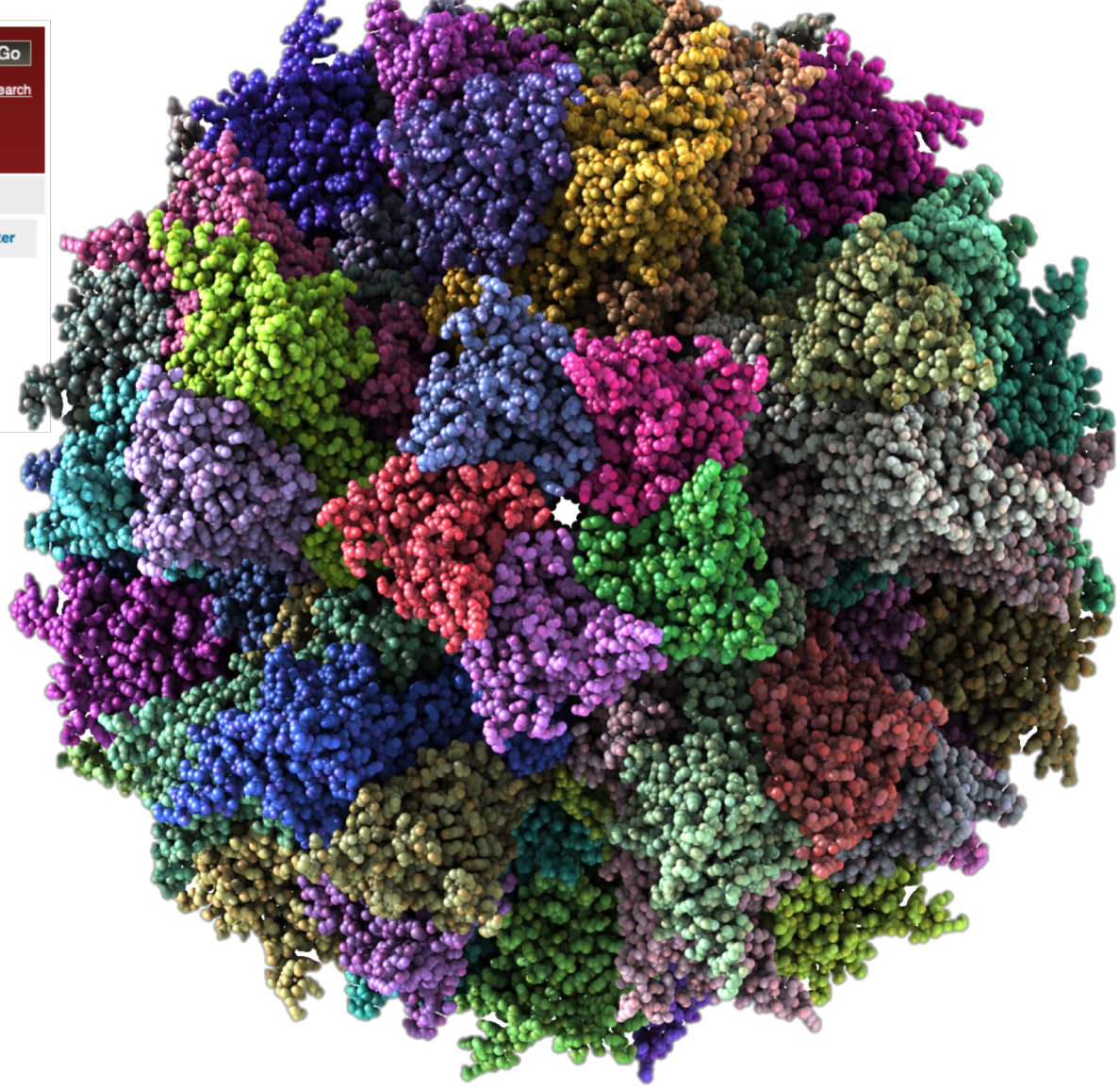


Cryo-electron Microscope

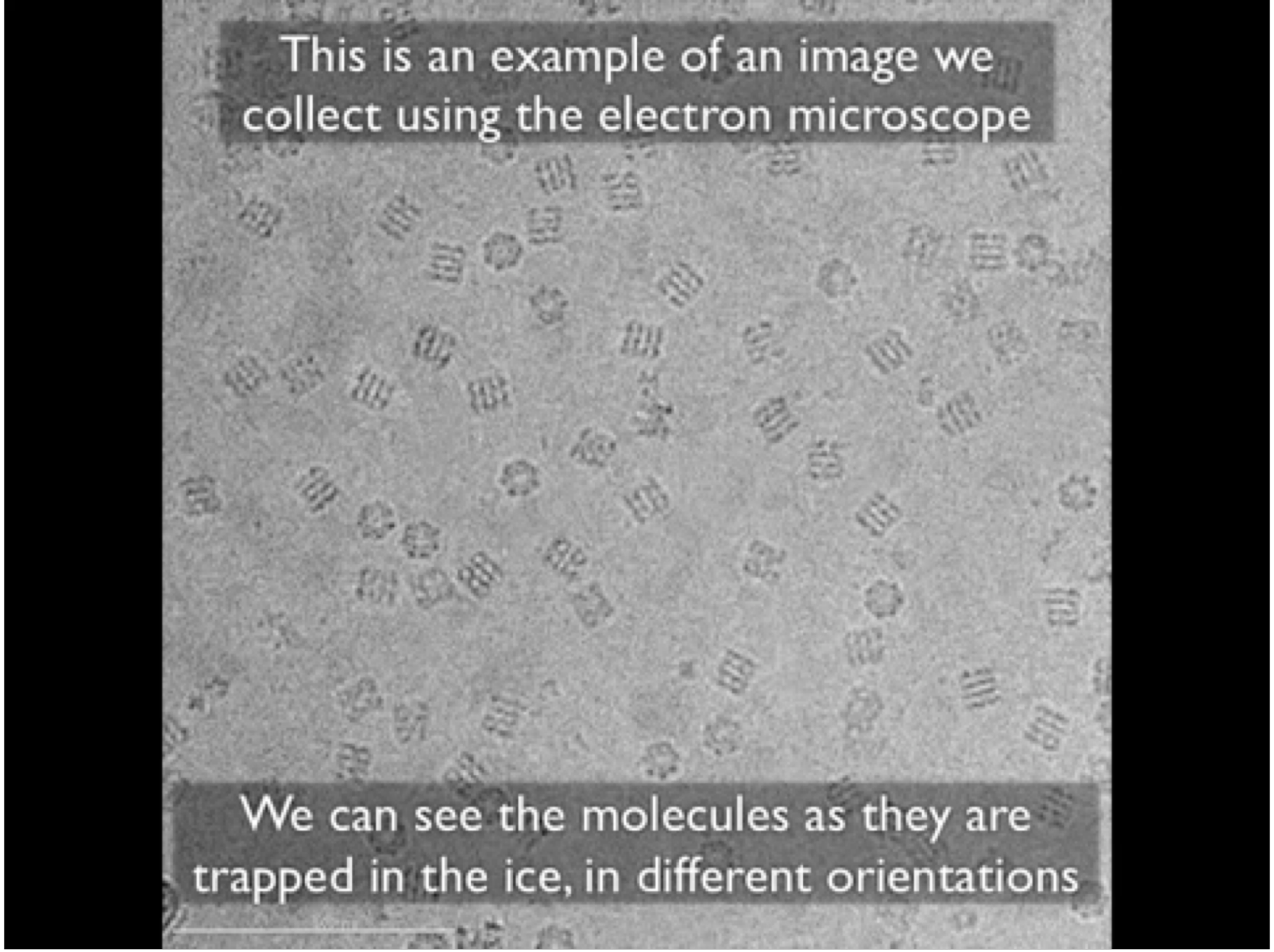


Terabytes of raw data

LOTS of processing



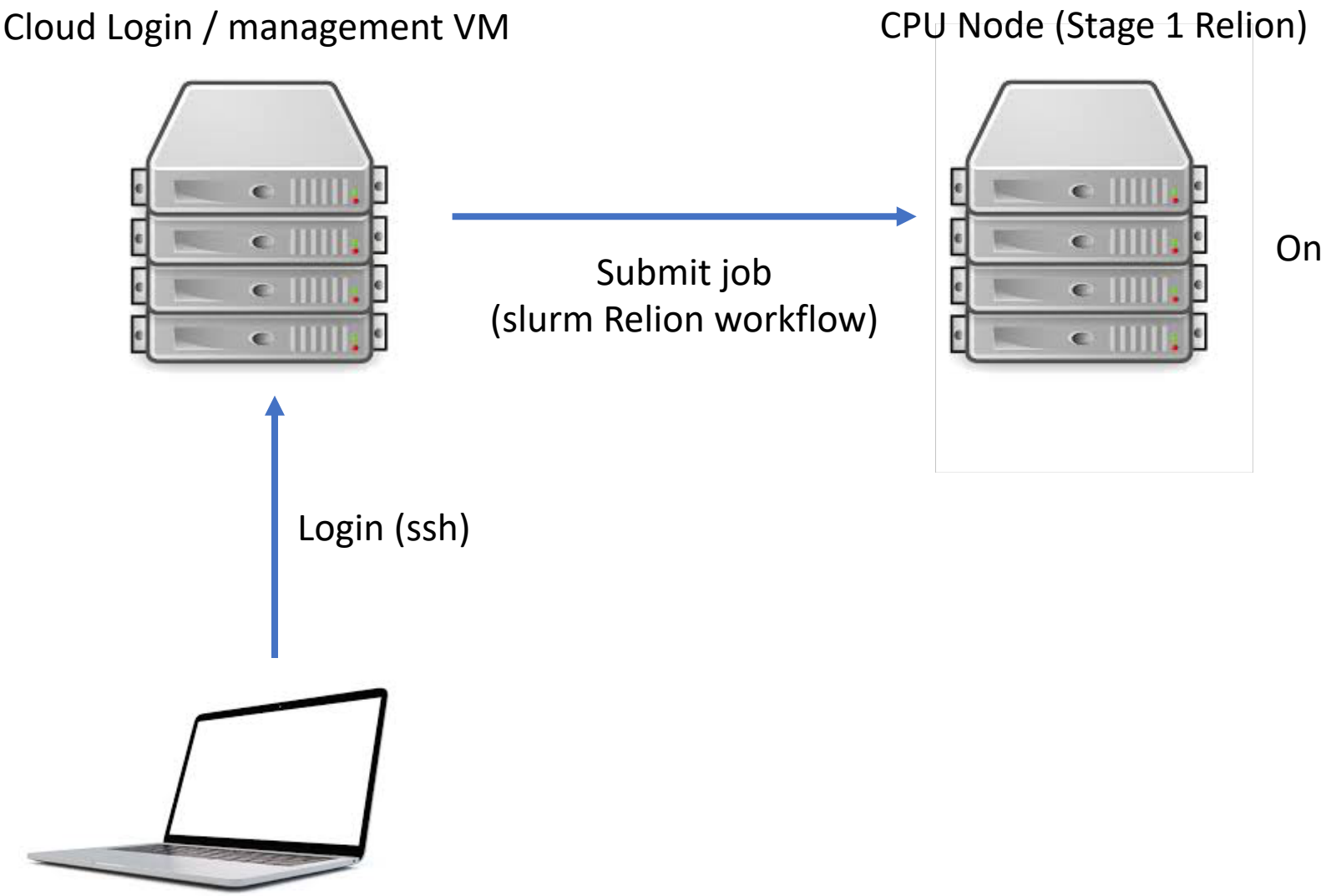
3D atomic structure of the protein



https://www.youtube.com/watch?v=BJKkC0W-6Qk

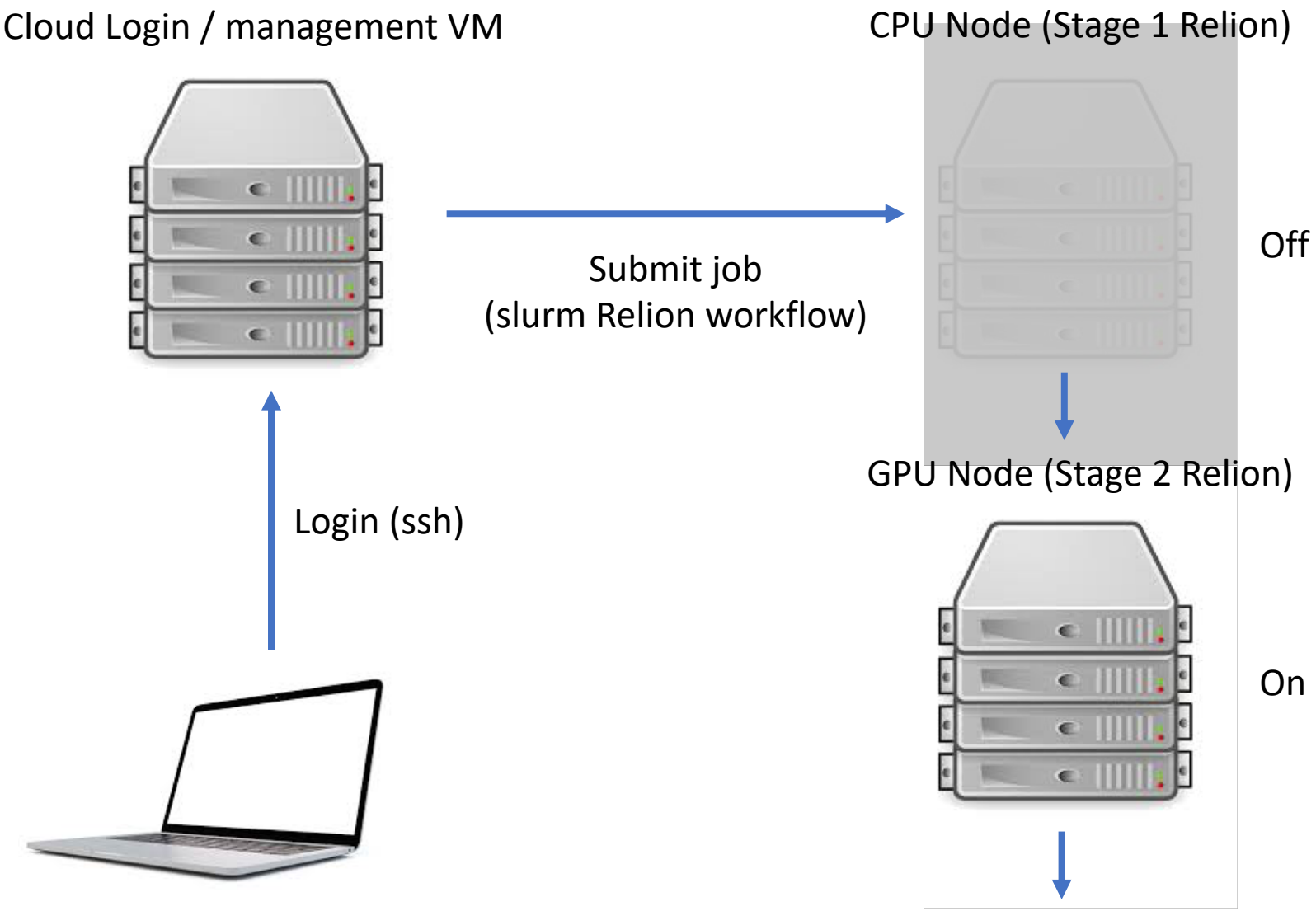
Gabe Lander This is a very brief introduction to how transmission cryo-electron microscopy is used to solve the structures of protein complexes. Download slides as PDF here: http://www.ocf.berkeley.edu/~glander/...

Cryo-EM image analysis (Relion)

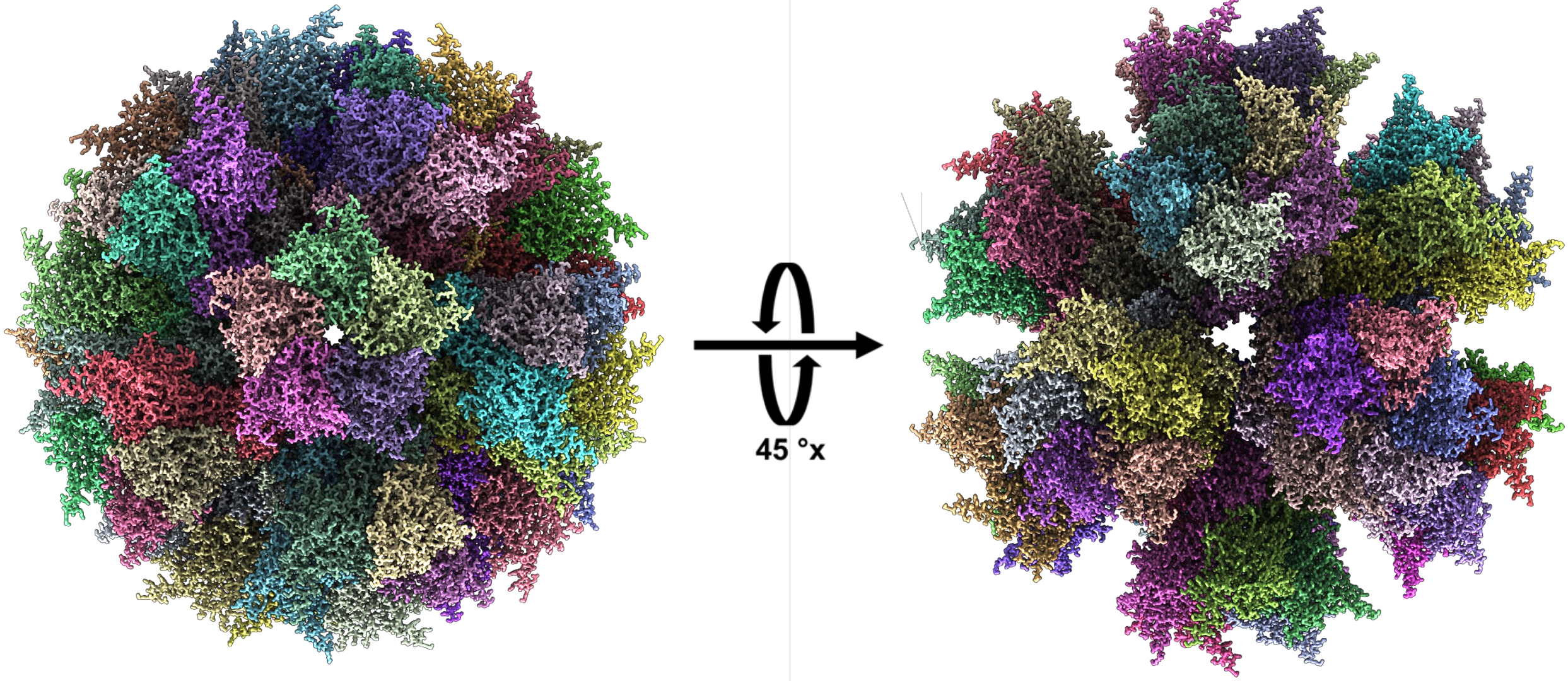


https://cluster-in-the-cloud.readthedocs.io

Cryo-EM image analysis (Relion)



https://cluster-in-the-cloud.readthedocs.io



Processing of Cryo-EM movie frames to reveal ADDomer atomic structure for structure-based drug design

Understanding the atomic structure enables simulation of vaccine behaviour against new viruses

Will provide information enabling rapid response to new virus mutations

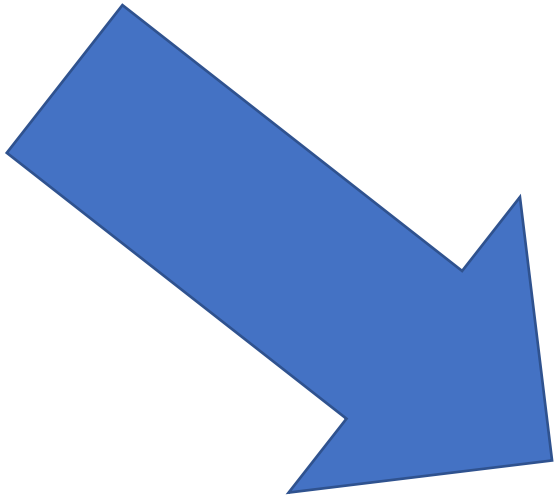
Generating these structures cost just £217 on OCI

Cluster-in-the-cloud projects

- Completed
  - **Molecular dynamics of proteins to investigate cell signaling (5200 cores)**
  - **Cryo-EM image analysis (TB of data)**
  - **GPU-accelerated binding affinity prediction (lots of GPUs)**
- In progress…
  - **Designing silicon devices for detection and control of radiotherapy doses (thousands of quantum physics simulations of x-ray photon trajectories)**
  - **Detection of dark matter signatures (deep physics)**
- **…and many more at other universities ☺**

A good waypoint – but not the final destination…

On-premise HPC



Cluster-in-the-cloud

Cloud is more than just a Personal Cluster…

- **We gave our researchers access to their own personal "cluster in the cloud"**
- **This let them perform their experiments in a familiar environment**
- **But – they were queuing jobs onto their own personal cluster!**
- **Revolutionary potential of the cloud will be realized if we treat it as an on-demand resource, not just as a "batch-cluster in the sky"**

2000

2018

A cellphone is more than just a mobile telephony device

Part 2
Acquire

Interactive molecular dynamics

BioSimSpace is 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.

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

Defining a simulation protocol

BioSimSpace provides functionality for defining various simulation protocols. In this notebook we will construct a typical simulation workflow that uses a sequence of simple protocols, with the output of one forming the input of the next:

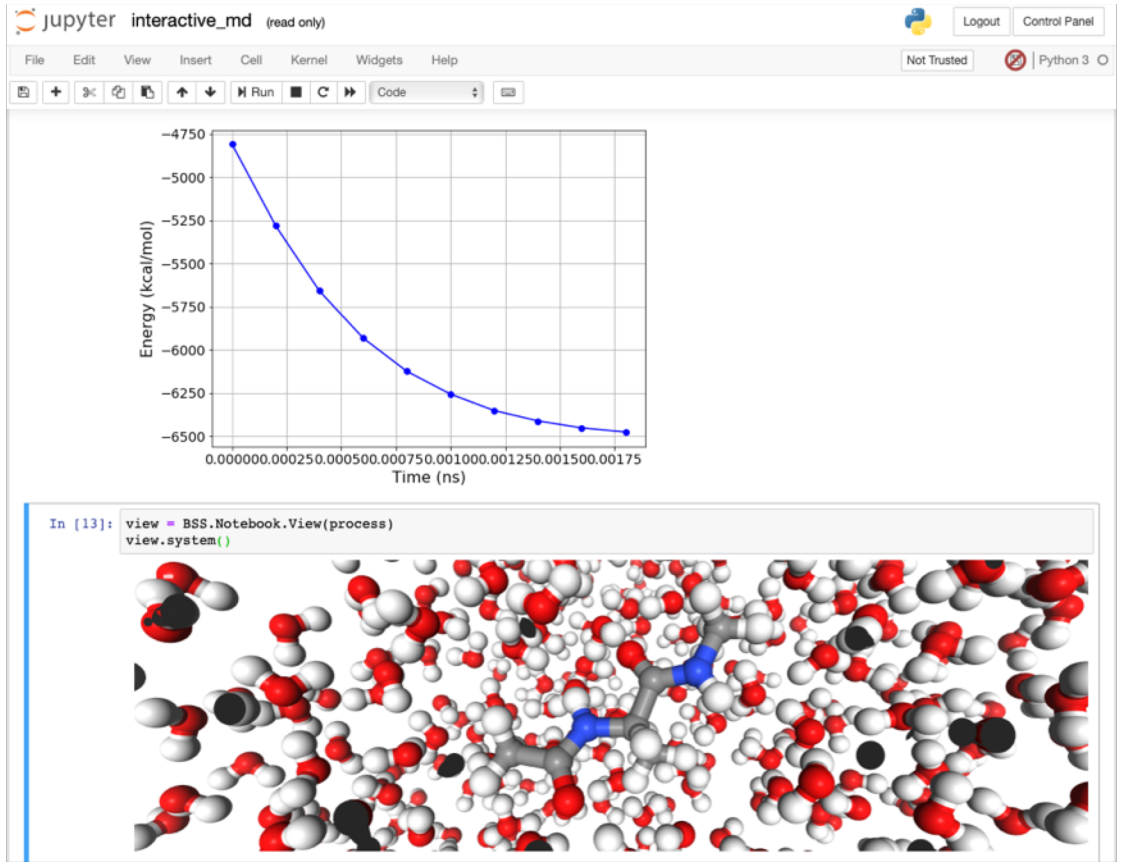
1. *Minimisation:* Energy minimisation the molecular system.
2. *Equilibration:* Equilibration of the system to a target temperature.
3. *Production:* Regular molecular dynamics, run at fixed temperature.
4. *Custom*: A user defined protocol, e.g. a config file for a molecular dynamics package.

When defining a protocol we are configuring the type of simulation that we wish to run, as well as any options for the particular simulation. For example, to create a default equilibration protocol:

```
protocol = BSS.Protcol.Equilibration()
```

This defines a 0.2 nanosecond equilibration protocol at a temperature of 300 Kelvin. For convenience, let's reduce the runtime. We'll also perform a heating protocol and will restrain the position of atoms in the backbone.

Jupyter Notebooks

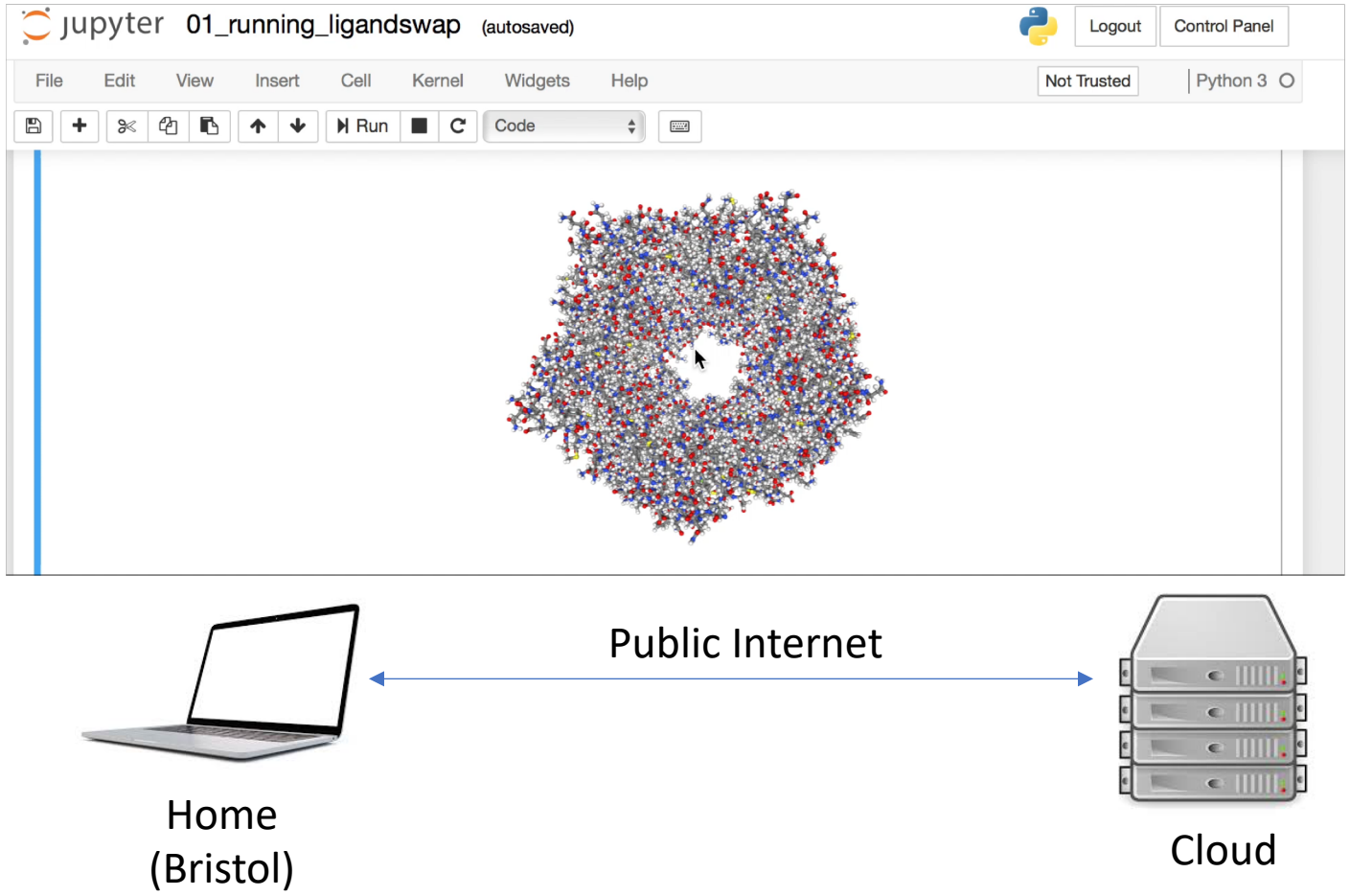


- **Jupyter notebook combines the;**
  - description of the experiment
  - code to run the experiment
  - code to analyse the results
  - graphs and 3D visualisations of the results
  - conclusions of the experiment
- **They contain everything needed to describe and reproduce the experiment**

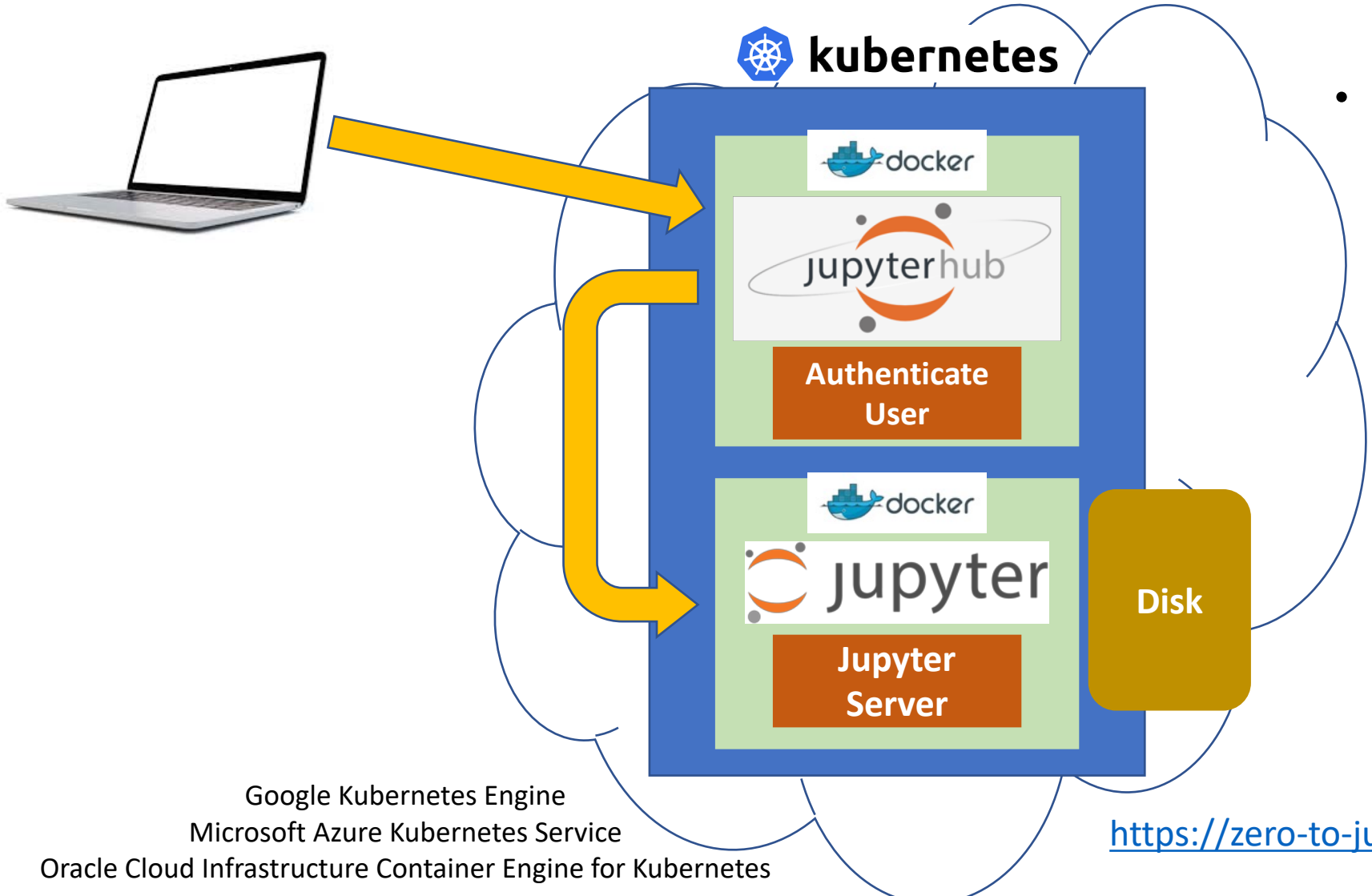
https://jupyter.org – https://notebook.biosimspace.org/hub/tmplogin

Notebooks are interactive papers

- **All compute and data sits in the cloud. Only the information needed to render the data in the notebook is transmitted over the network**
- **Hugely useful for open and reproducible science**
- **Notebooks are, in effect, interactive scientific papers ☺**



Kubernetes and JupyterHub



- **JupyterHub running on k8s**
  - Easy to use helm chart!
  - Great community and instructions
  - Works with lots (all?) cloud kubernetes services, or roll-your-own clusters

Google Kubernetes Engine
Microsoft Azure Kubernetes Service
Oracle Cloud Infrastructure Container Engine for Kubernetes

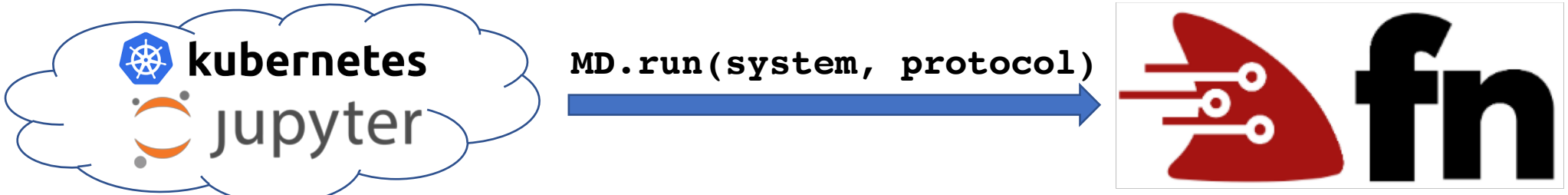
https://zero-to-jupyterhub.readthedocs.io/en/stable/

Simulations as Serverless Functions

We now have everything that is needed to create a process object. To do so, run:

```
In [5]: process = BSS.MD.run(system, protocol)

In [6]: process.isRunning()
Out[6]: True
```

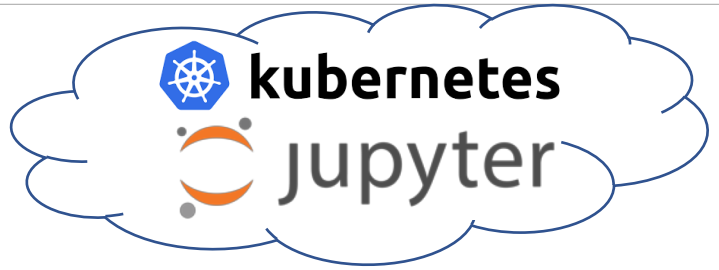
- The above line starts and runs a molecular dynamics simulation
- However, we cannot run this in the k8s pod, as the hardware is too tiny…
  - (…or else the k8s cluster would be too expensive)
- Instead we burst out to HPC hardware using a “serverless” function service



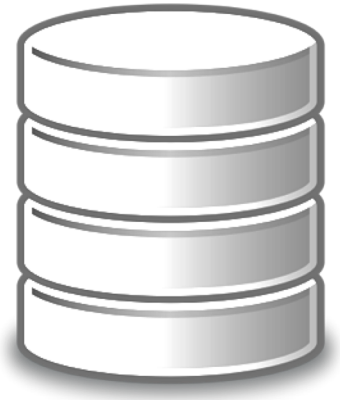
Auto-scaling 1 or 2 core VMs

Simulations triggered by Fn Functions


```
In [ ]: import BioSimSpace as BSS

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

In [ ]: # Initialise a short equilibration protocol.
        protocol = BSS.Protocol.Equilibration(runtime=0.05*BSS.Units.Time.nanosecond,
                                              temperature_start=0*BSS.Units.Temperature.kelvin,
                                              temperature_end=300*BSS.Units.Temperature.kelvin,
                                              restrain_backbone=True)

In [ ]: process = BSS.MD.run(system, protocol)

In [ ]: # Generate a plot of time vs temperature.
        plot1 = BSS.Notebook.plot(process.getTime(time_series=True),
            process.getTemperature(time_series=True))

        # Generate a plot of time vs energy.
        plot2 = BSS.Notebook.plot(process.getTime(time_series=True),
            process.getTotalEnergy(time_series=True))

In [ ]: view = BSS.Notebook.View(process)
        view.system()
```



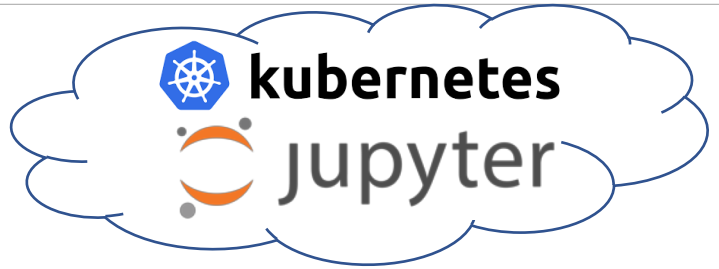
Auto-scaling 1 or 2 core VMs



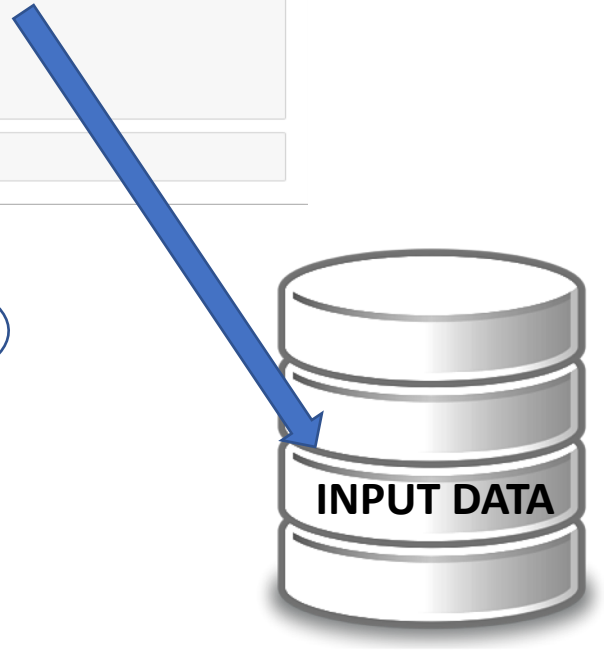
Object Store

Simulations triggered by Fn Functions


```
In [ ]: import BioSimSpace as BSS

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

In [ ]: # Initialise a short equilibration protocol.
        protocol = BSS.Protocol.Equilibration(runtime=0.05*BSS.Units.Time.nanosecond,
                                              temperature_start=0*BSS.Units.Temperature.kelvin,
                                              temperature_end=300*BSS.Units.Temperature.kelvin,
                                              restrain_backbone=True)

In [ ]: process = BSS.MD.run(system, protocol)

In [ ]: # Generate a plot of time vs temperature.
        plot1 = BSS.Notebook.plot(process.getTime(time_series=True),
            process.getTemperature(time_series=True))

        # Generate a plot of time vs energy.
        plot2 = BSS.Notebook.plot(process.getTime(time_series=True),
            process.getTotalEnergy(time_series=True))

In [ ]: view = BSS.Notebook.View(process)
        view.system()
```



Auto-scaling 1 or 2 core VMs

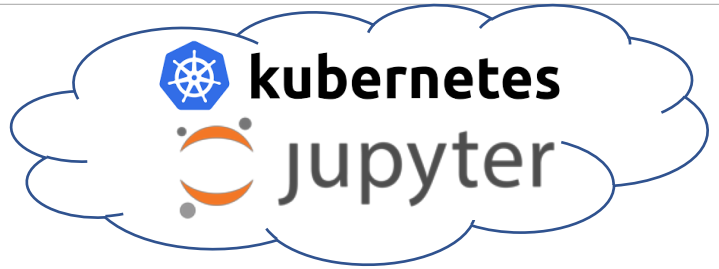


Object Store

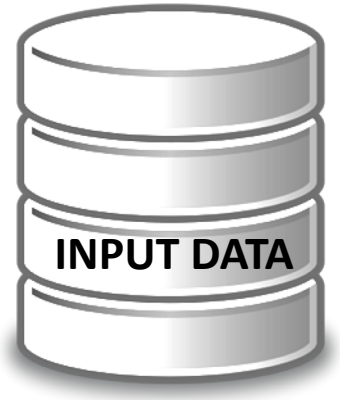
Simulations triggered by Fn Functions

```
In [ ]: import BioSimSpace as BSS

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

In [ ]: # Initialise a short equilibration protocol.
        protocol = BSS.Protocol.Equilibration(runtime=0.05*BSS.Units.Time.nanosecond,
                                              temperature_start=0*BSS.Units.Temperature.kelvin,
                                              temperature_end=300*BSS.Units.Temperature.kelvin,
                                              restrain_backbone=True)

In [ ]: process = BSS.MD.run(system, protocol)

In [ ]: # Generate a plot of time vs temperature.
        plot1 = BSS.Notebook.plot(process.getTime(time_series=True),
            process.getTemperature(time_series=True))

        # Generate a plot of time vs energy.
        plot2 = BSS.Notebook.plot(process.getTime(time_series=True),
            process.getTotalEnergy(time_series=True))

In [ ]: view = BSS.Notebook.View(process)
        view.system()
```

MD.run(bucket)



Auto-scaling 1 or 2 core VMs



Object Store

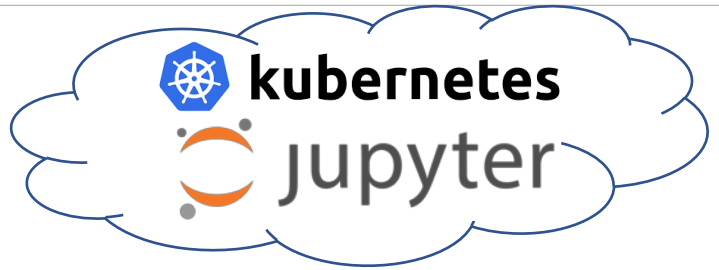
Simulations triggered by Fn Functions

```
In [ ]: import BioSimSpace as BSS

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

In [ ]: # Initialise a short equilibration protocol.
        protocol = BSS.Protocol.Equilibration(runtime=0.05*BSS.Units.Time.nanosecond,
                                              temperature_start=0*BSS.Units.Temperature.kelvin,
                                              temperature_end=300*BSS.Units.Temperature.kelvin,
                                              restrain_backbone=True)

In [ ]: process = BSS.MD.run(system, protocol)

In [ ]: # Generate a plot of time vs temperature.
        plot1 = BSS.Notebook.plot(process.getTime(time_series=True),
            process.getTemperature(time_series=True))

        # Generate a plot of time vs energy.
        plot2 = BSS.Notebook.plot(process.getTime(time_series=True),
            process.getTotalEnergy(time_series=True))

In [ ]: view = BSS.Notebook.View(process)
        view.system()
```

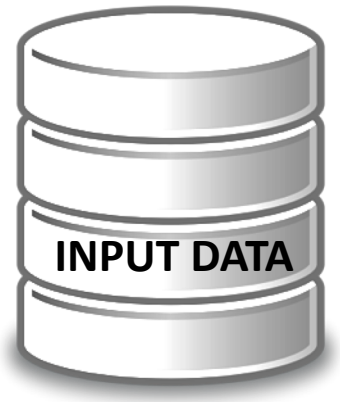
MD.run(bucket)

sbatch job
[bucket]

cluster-in-the-cloud
Login (VM1.1)



Auto-scaling 1 or 2 core VMs



Object Store

Simulations triggered by Fn Functions

```
In [ ]: import BioSimSpace as BSS

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

In [ ]: # Initialise a short equilibration protocol.
        protocol = BSS.Protocol.Equilibration(runtime=0.05*BSS.Units.Time.nanosecond,
                                              temperature_start=0*BSS.Units.Temperature.kelvin,
                                              temperature_end=300*BSS.Units.Temperature.kelvin,
                                              restrain_backbone=True)

In [ ]: process = BSS.MD.run(system, protocol)

In [ ]: # Generate a plot of time vs temperature.
        plot1 = BSS.Notebook.plot(process.getTime(time_series=True),
            process.getTemperature(time_series=True))

        # Generate a plot of time vs energy.
        plot2 = BSS.Notebook.plot(process.getTime(time_series=True),
            process.getTotalEnergy(time_series=True))

In [ ]: view = BSS.Notebook.View(process)
        view.system()
```

MD.run(bucket)

sbatch job
[bucket]

cluster-in-the-cloud
Login (VM1.1)

kubernetes

jupyter

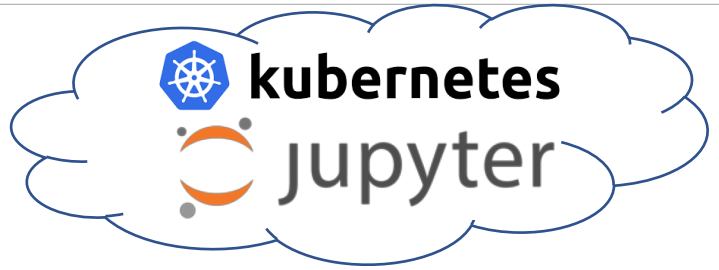
Auto-scaling 1 or 2 core VMs

INPUT DATA

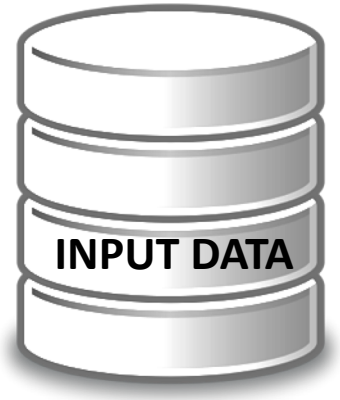
Object Store

On-demand Cluster-in-the-Cloud

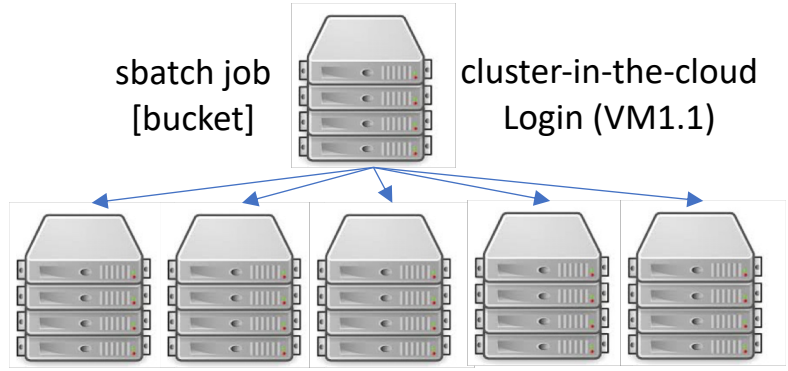
```
In [ ]: import BioSimSpace as BSS

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

In [ ]: # Initialise a short equilibration protocol.
        protocol = BSS.Protocol.Equilibration(runtime=0.05*BSS.Units.Time.nanosecond,
                                              temperature_start=0*BSS.Units.Temperature.kelvin,
                                              temperature_end=300*BSS.Units.Temperature.kelvin,
                                              restrain_backbone=True)

In [ ]: process = BSS.MD.run(system, protocol)

In [ ]: # Generate a plot of time vs temperature.
        plot1 = BSS.Notebook.plot(process.getTime(time_series=True),
            process.getTemperature(time_series=True))

        # Generate a plot of time vs energy.
        plot2 = BSS.Notebook.plot(process.getTime(time_series=True),
            process.getTotalEnergy(time_series=True))

In [ ]: view = BSS.Notebook.View(process)
        view.system()
```



Auto-scaling 1 or 2 core VMs



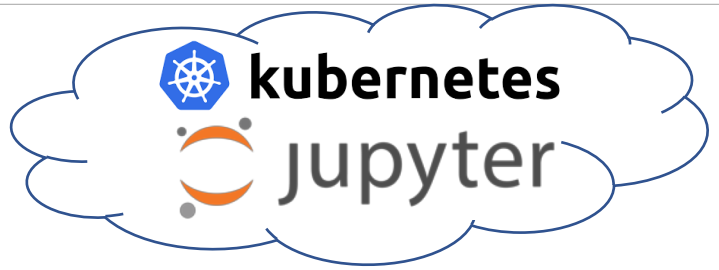
Object Store



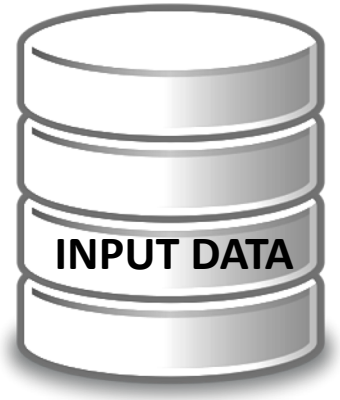
Compute nodes - any shape(s)

On-demand Cluster-in-the-Cloud

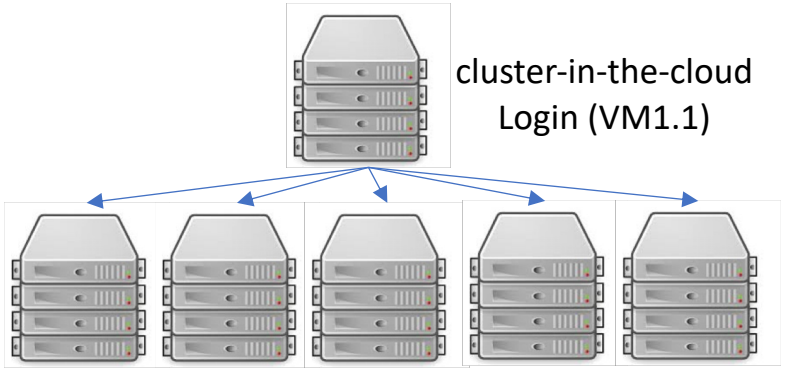
```
In [ ]: import BioSimSpace as BSS

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

In [ ]: # Initialise a short equilibration protocol.
        protocol = BSS.Protocol.Equilibration(runtime=0.05*BSS.Units.Time.nanosecond,
                                              temperature_start=0*BSS.Units.Temperature.kelvin,
                                              temperature_end=300*BSS.Units.Temperature.kelvin,
                                              restrain_backbone=True)

In [ ]: process = BSS.MD.run(system, protocol)

In [ ]: # Generate a plot of time vs temperature.
        plot1 = BSS.Notebook.plot(process.getTime(time_series=True),
            process.getTemperature(time_series=True))

        # Generate a plot of time vs energy.
        plot2 = BSS.Notebook.plot(process.getTime(time_series=True),
            process.getTotalEnergy(time_series=True))

In [ ]: view = BSS.Notebook.View(process)
        view.system()
```



Auto-scaling 1 or 2 core VMs

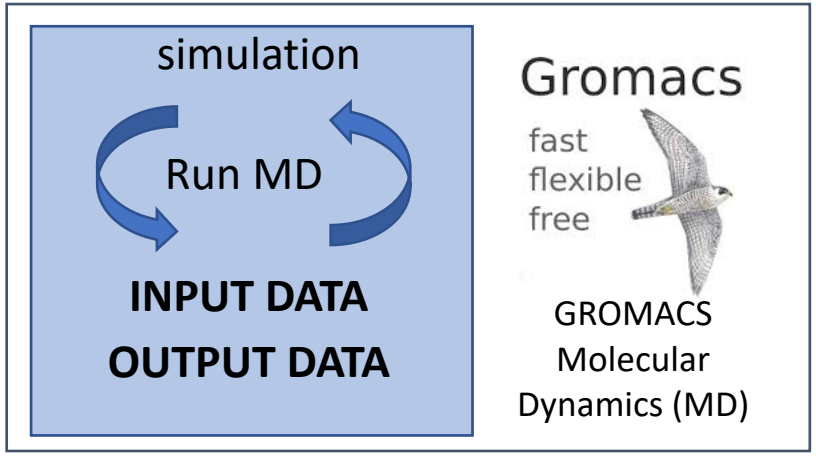


Object Store

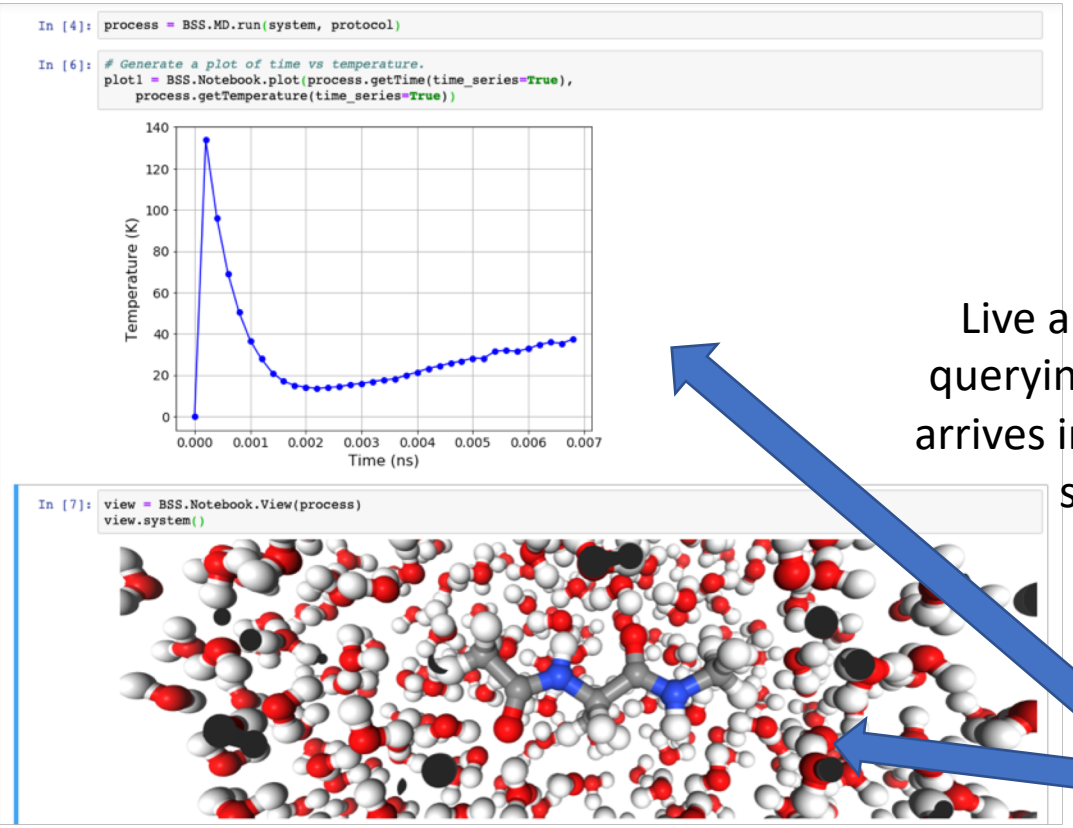


cluster-in-the-cloud
Login (VM1.1)

Compute nodes - any shape(s)



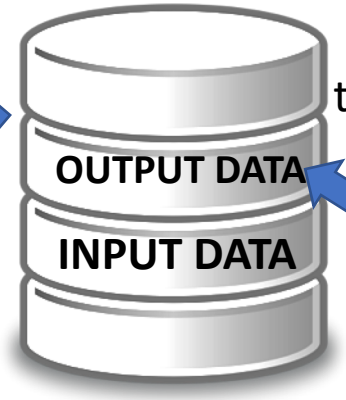
Serverless + Object Store ☺



kubernetes jupyter

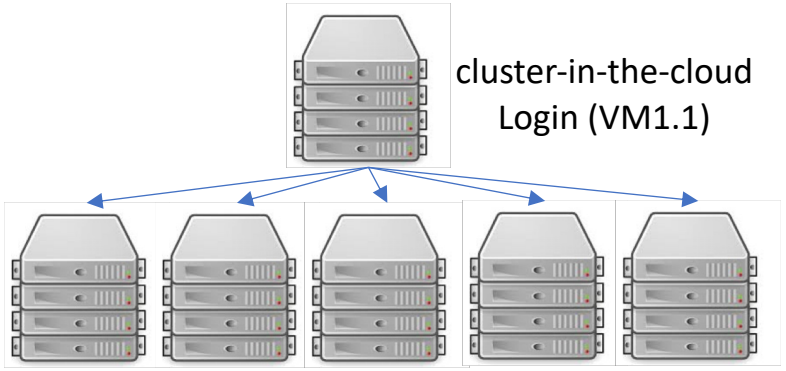
Auto-scaling 1 or 2 core VMs

Live analysis by querying data as it arrives in the object store



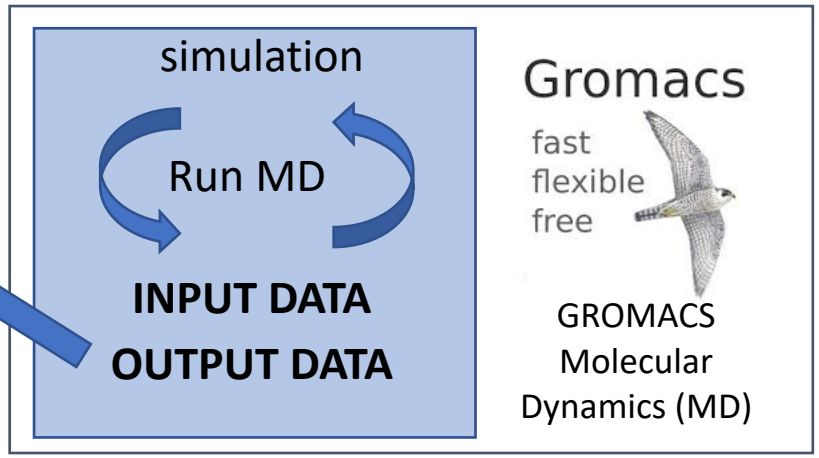
Object Store

Stream output to object store

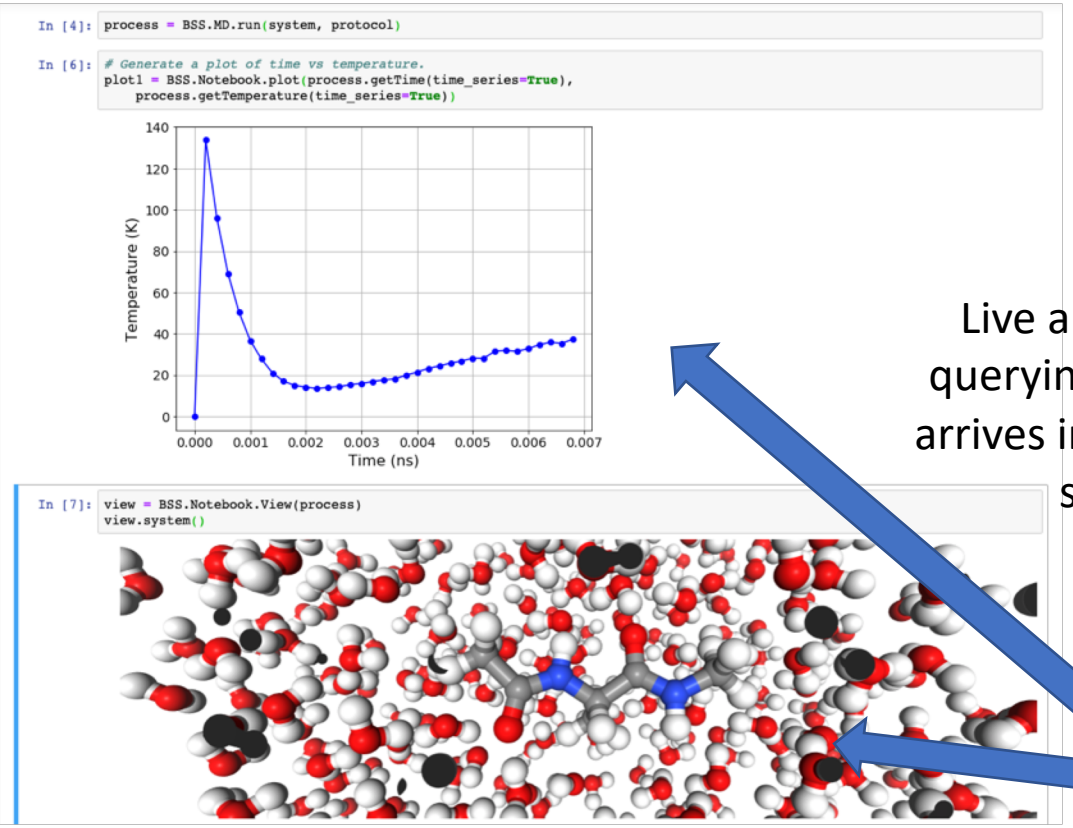


cluster-in-the-cloud
Login (VM1.1)

Compute nodes - any shape(s)

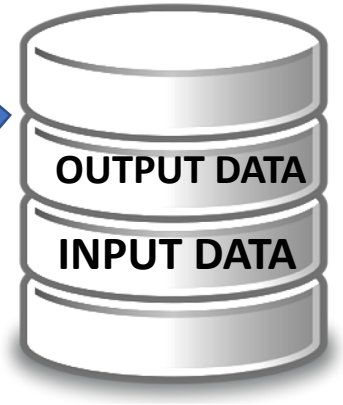


Serverless + Object Store ☺



Live analysis by querying data as it arrives in the object store

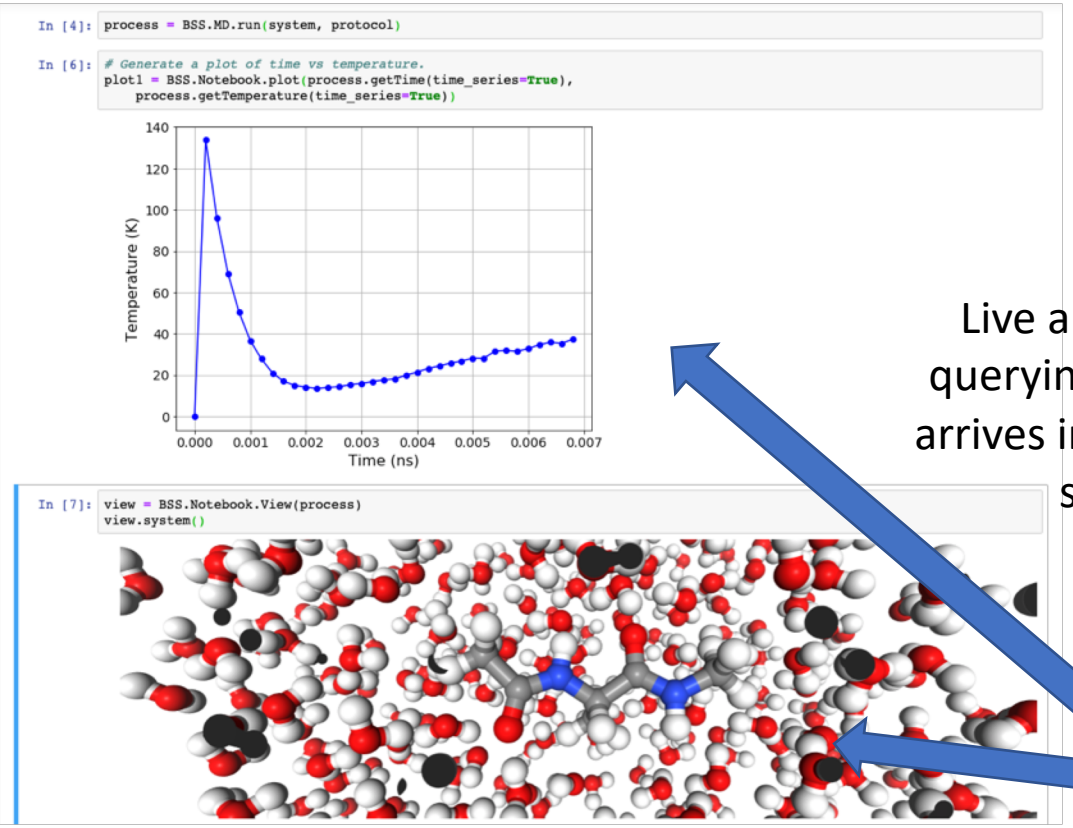
cluster-in-the-cloud
Login (VM1.1)



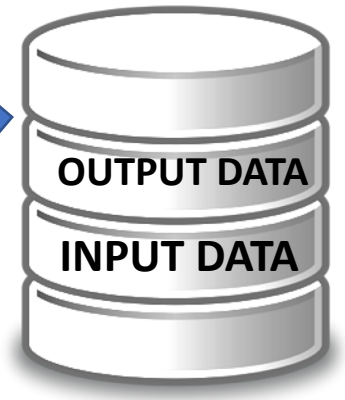
Object Store

Auto-scaling 1 or 2 core VMs

Serverless + Object Store ☺



Live analysis by querying data as it arrives in the object store



Object Store

kubernetes
jupyter

Auto-scaling 1 or 2 core VMs

Anyone can run simulations…!

- **Simple framework that allows ANYONE to run HPC simulations by calling the Fn function via a public URL!**

Anyone can run simulations…!

- **Simple framework that allows ANYONE to run HPC simulations by calling the Fn function via a public URL!**

That could get expensive…!

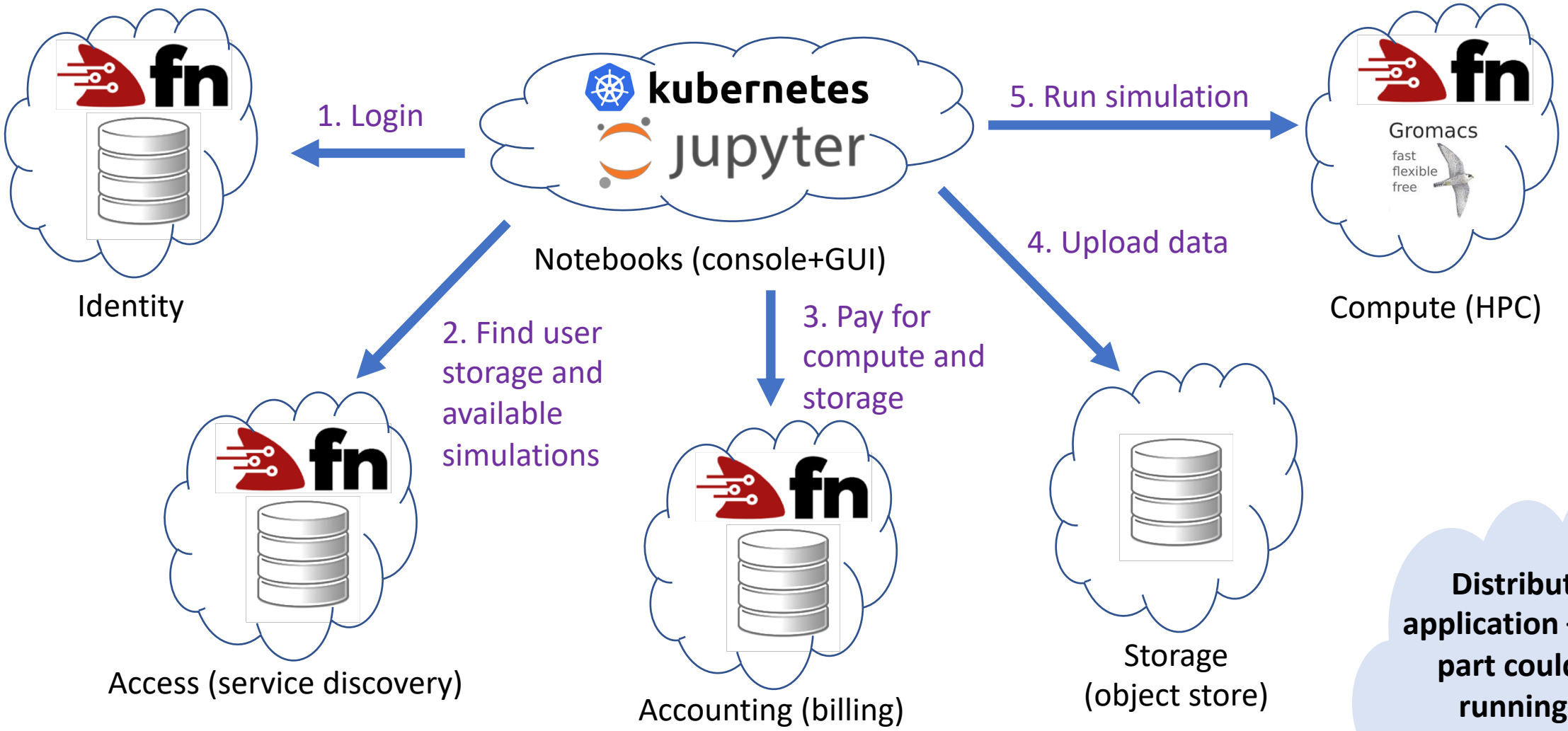
Looks like we need some user authentication, access control and accounting…

Identity (authorisation)

```
In [ ]: from Acquire.Client import User

In [ ]: user = User("chryswoods")

In [ ]: (url,qrcode) = user.request_login()

In [ ]: qrcode

In [ ]: from IPython.core.display import HTML
        print(url)
        HTML("<a href='%s'>Login here</a>" % url)

In [ ]: user.wait_for_login()

In [ ]: user.is_logged_in()

In [ ]: user.logout()

In [ ]:
```

- Built identity (authorisation) service on top of Fn serverless and object store for state
- "request_login" call from the notebook calls "request_login" serverless function. This looks up user details from object store and returns a unique login URL
- Login page also served as html from an Fn function
- Notebook can wait for the login to complete, and uses security tokens to authenticate with simulation function service
- https://github.com/chryswoods/acquire

Identity, Access, Accounting, Storage, Compute



Distributed application – each part could be running in different places

Acquire = Cluster-in-the-Cloud + Accounting + Data Management

- 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.
- **Acquire will 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**

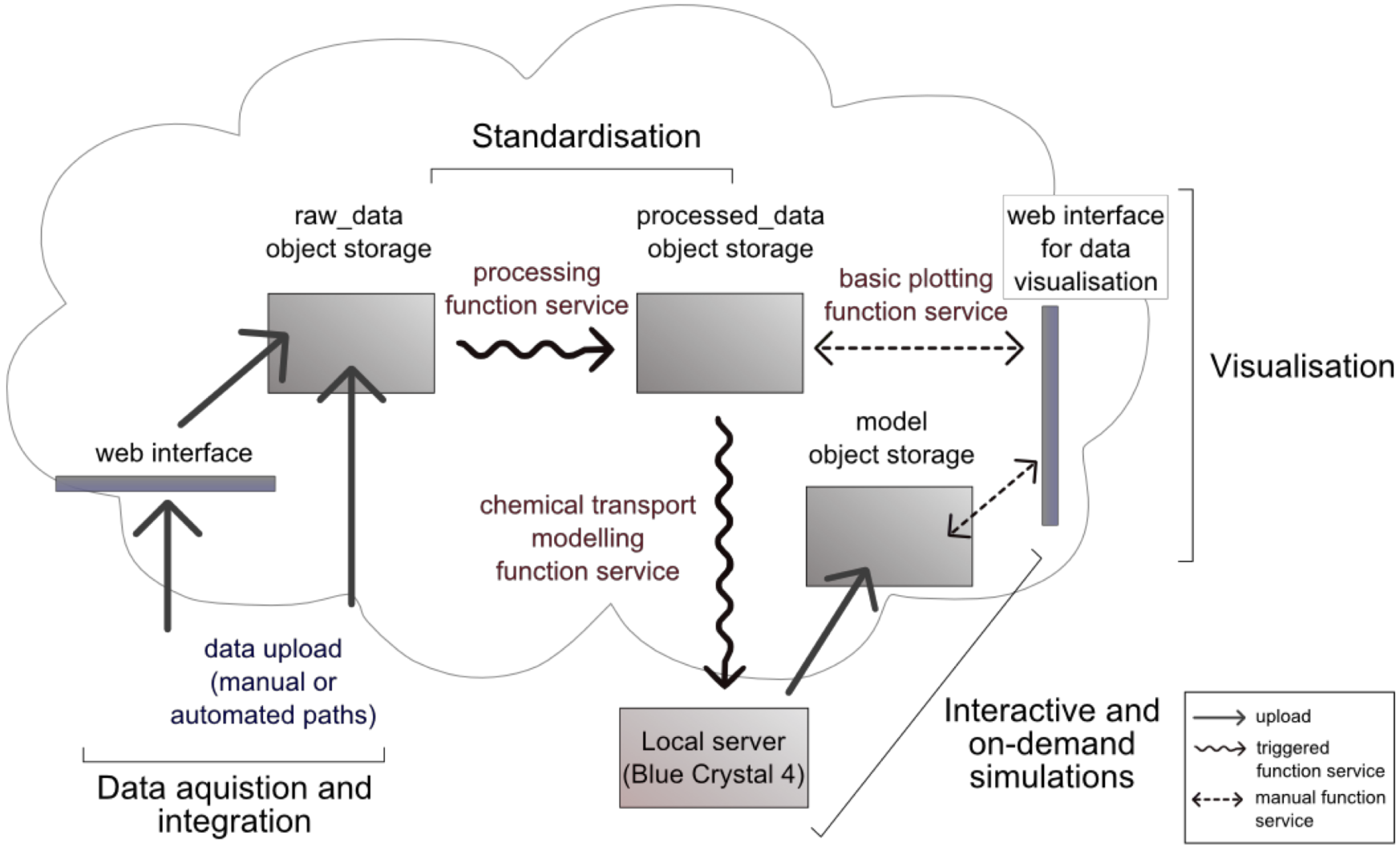
Acquire = Cluster-in-the-Cloud + Accounting + Data Management

- Outputs will be **read-only** and up-front **cost 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**
- Planning a web console to 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)

Acquire = Cluster-in-the-Cloud + Accounting + Data Management

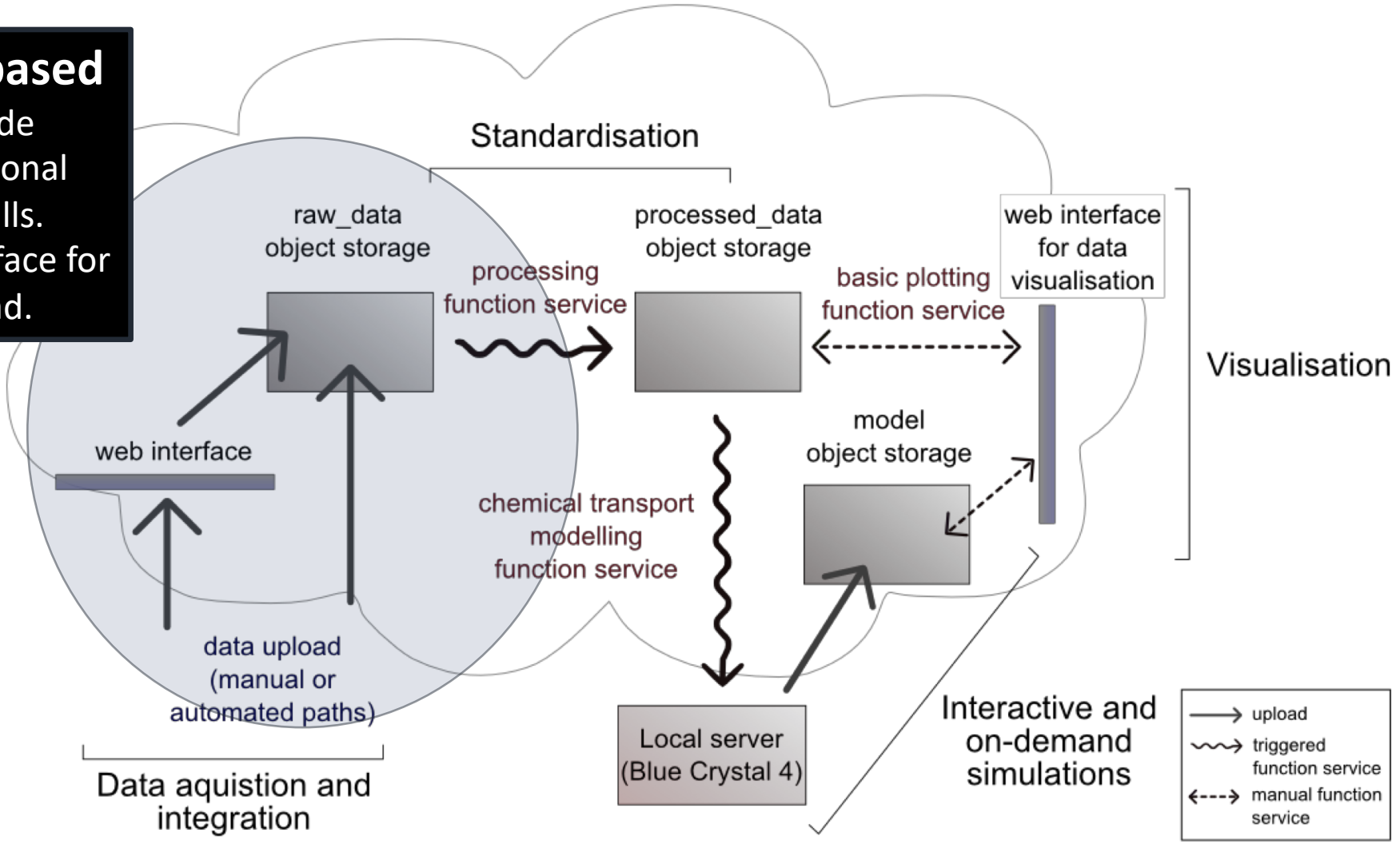
- System will **support Jupyter-notebook type users, and also "traditional" users** running normal HPC workloads via a cluster-in-the-cloud interface.
- It could **scale from a single researcher** on one project using notebooks…
- …**through universities managing 1000's users** on "infinitely" scaling "infinitely" heterogeneous HPC clusters in the cloud…
- …**to researchers across many universities and companies** sharing and collaborating on data running calculations, analyses, publishing interactive papers and **performing modern digital science**

HUGS (Hub for UK Greenhouse Gas Science



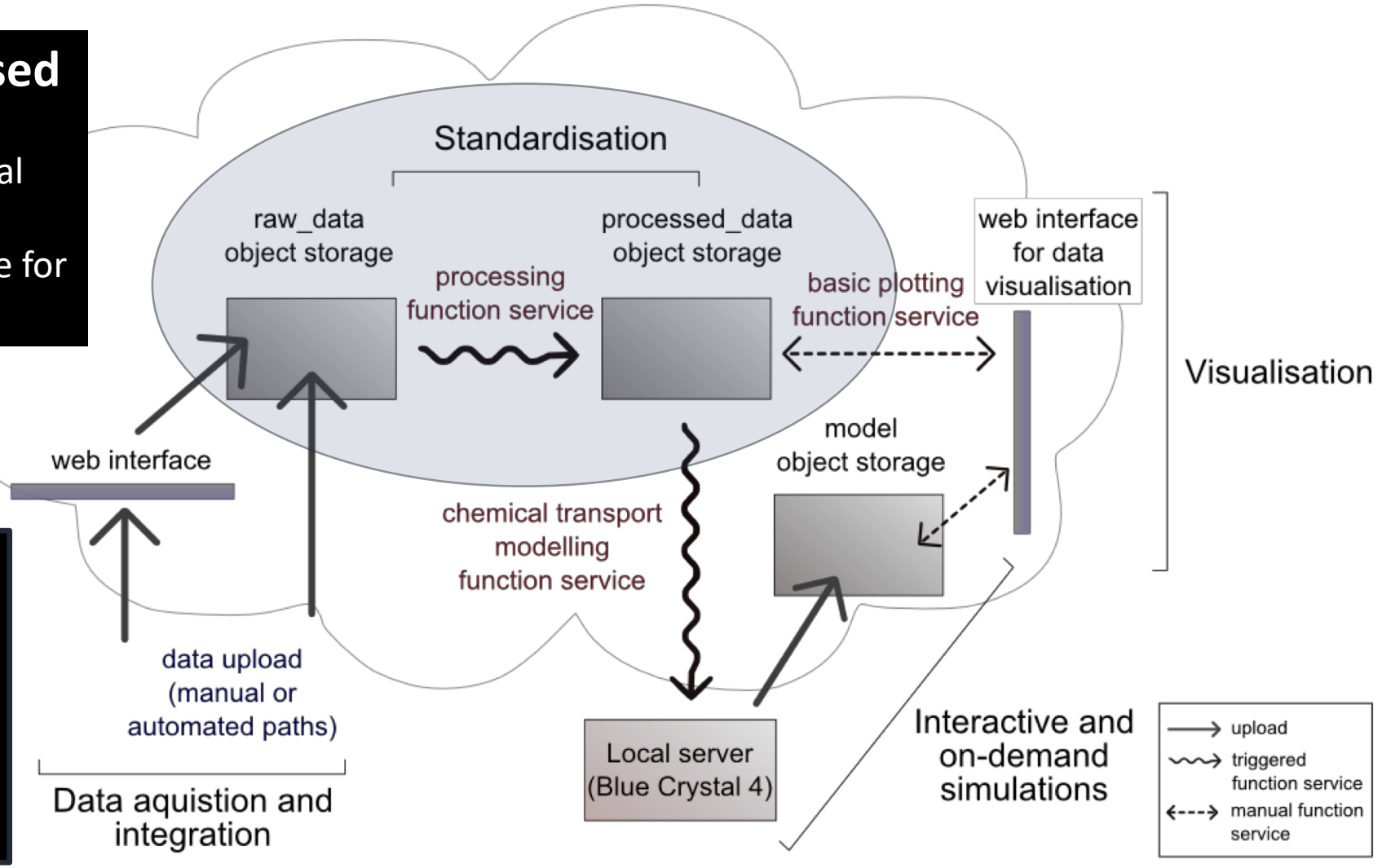
HUGS (Hub for UK Greenhouse Gas Science

Cloud based
Outside institutional firewalls.
Web interface for upload.



HUGS (Hub for UK Greenhouse Gas Science

Cloud based
Outside institutional firewalls.
Web interface for upload.

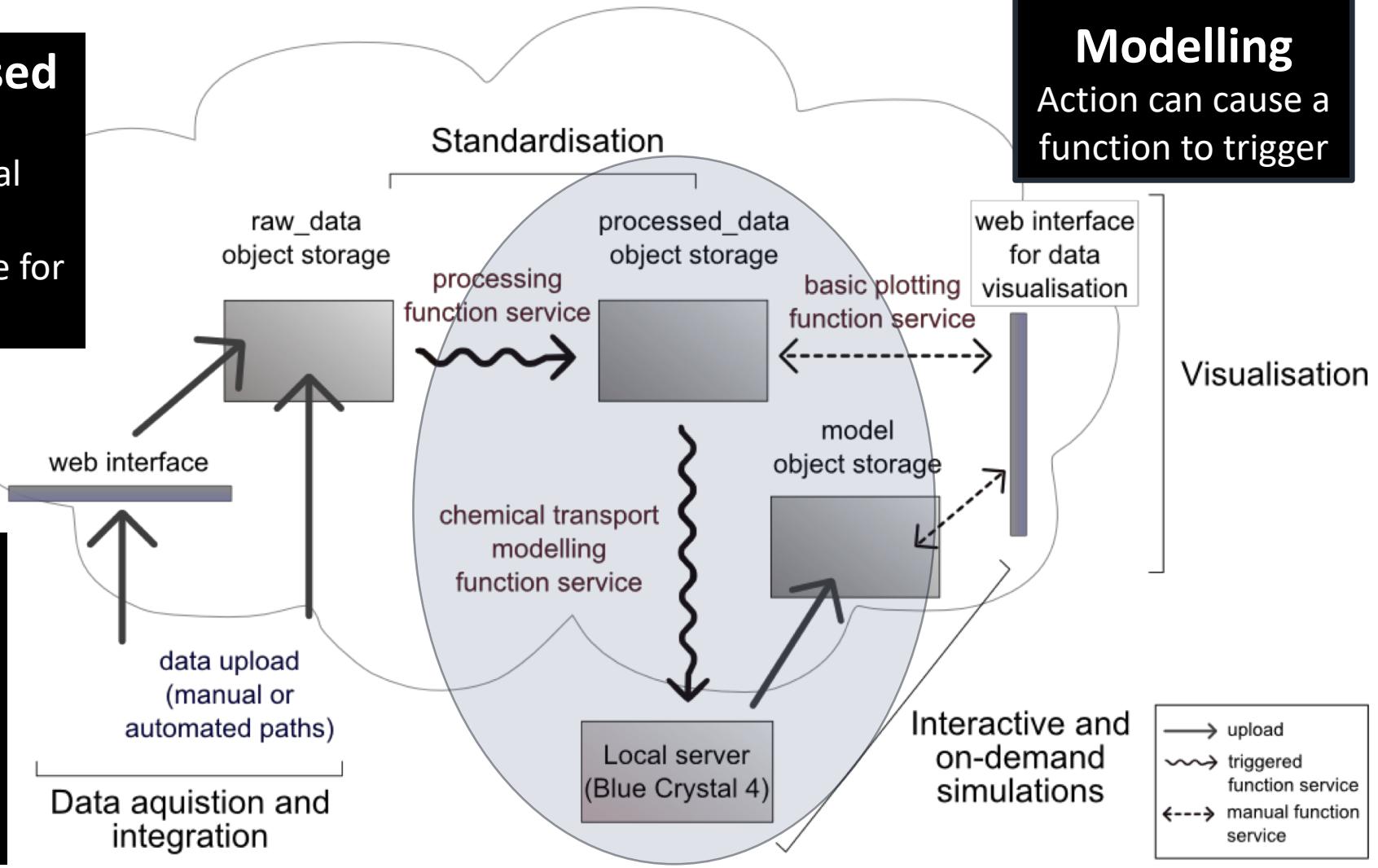


Accepts many formats
Communicate with data providers to determine standardisation.

HUGS (Hub for UK Greenhouse Gas Science

Cloud based
Outside institutional firewalls.
Web interface for upload.

Automated Modelling
Action can cause a function to trigger

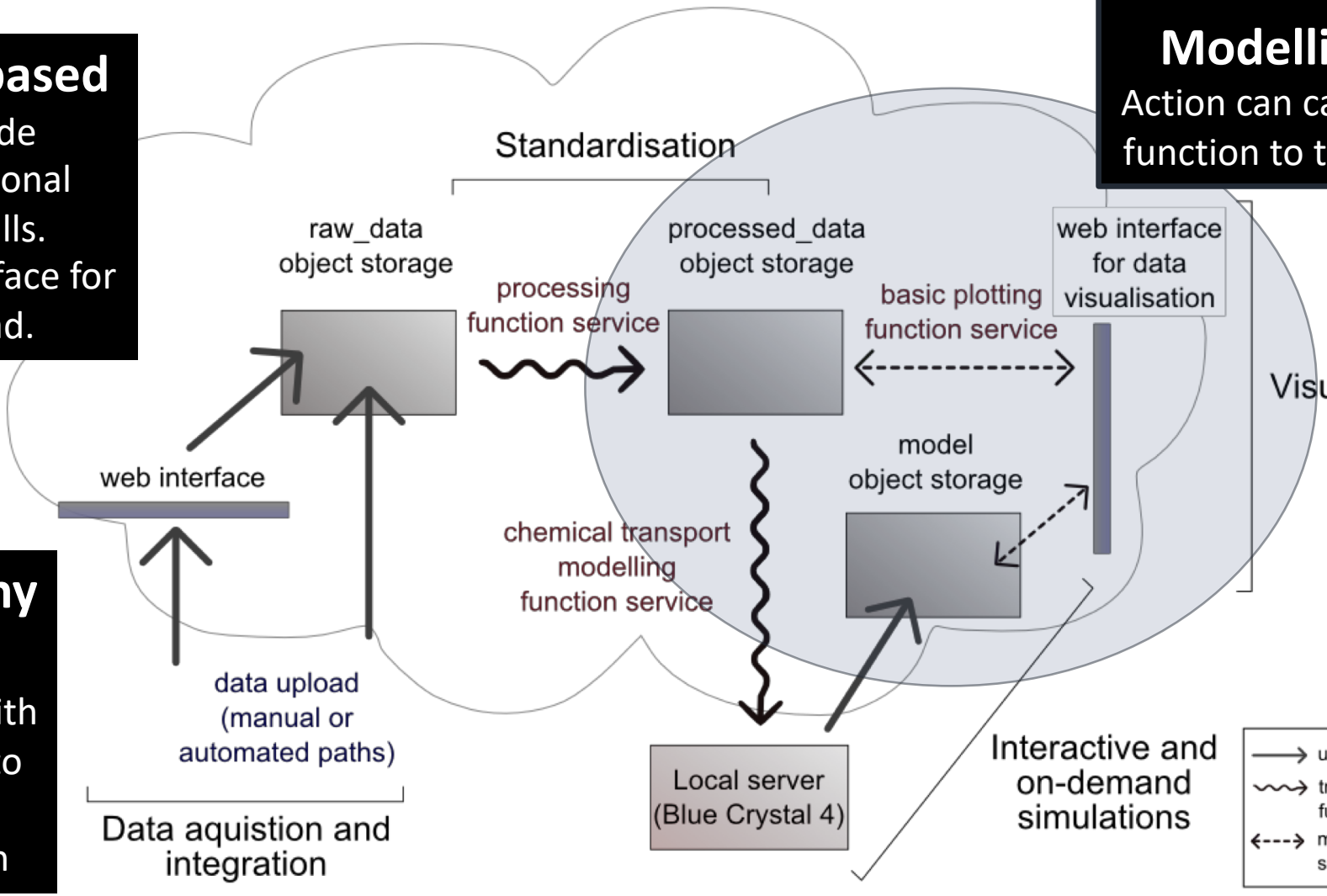


Accepts many formats
Communicate with data providers to determine standardisation

HUGS (Hub for UK Greenhouse Gas Science

Cloud based
Outside institutional firewalls.
Web interface for upload.

Automated Modelling
Action can cause a function to trigger



Analysis front-end
Interactive web-based service

Accepts many formats
Communicate with data providers to determine standardisation

Applications Places System rt17603 Tue Mar 5, 5:38 PM

Plot data and footprint - Mozilla Firefox

Examples | Plot data and footprint

localhost:8888/notebooks/Plot data and footprint.ipynb 110% Search

Most Visited Scientific Linux Distros

jupyter Plot data and footprint Last Checkpoint: Last Thursday at 6:03 PM (unsaved changes) Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python [conda env:anaconda]

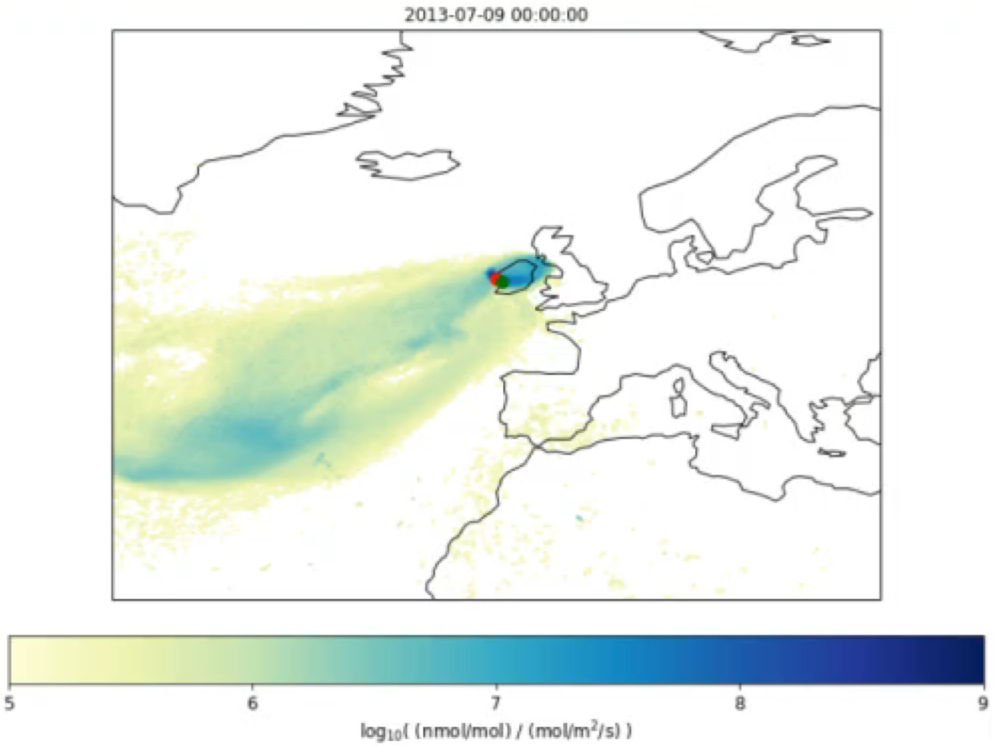
Code

In [7]:
```
data["MHD"]["mf"].idxmax()
```

Out[7]:
```
Timestamp('2013-07-09 05:44:22.500000')
```

In [8]:
```
plot_fp(footprints,date="2013-07-09",lat_range=[20,80],lon_range=[-50,30],figsize=(12,9))
```

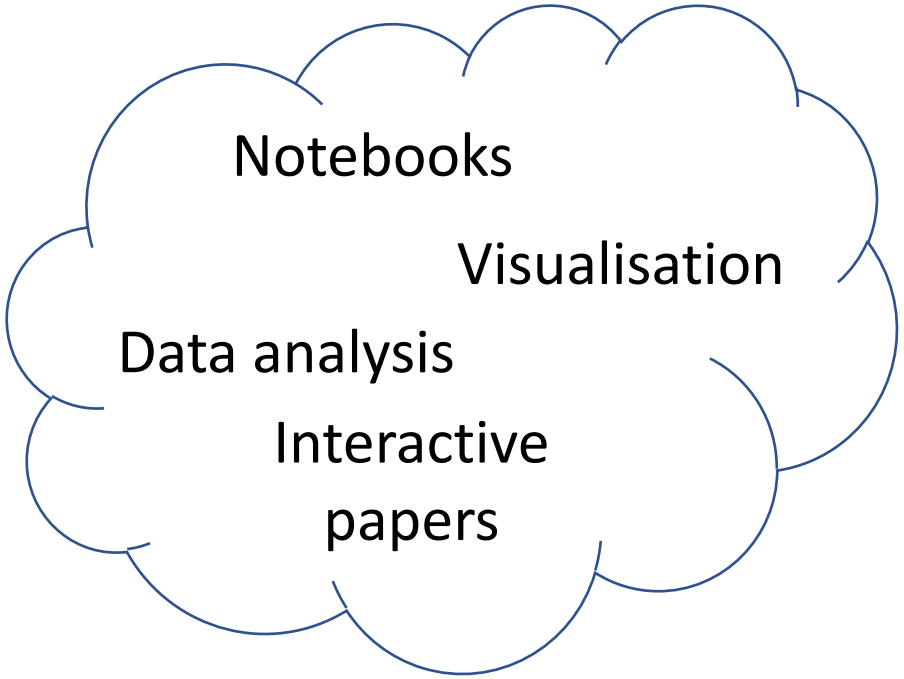
```
/home/rt17603/Documents/ACRG_code/acrg/acrg_name/name.py:1825: RuntimeWarning: divide by zero encountered in log10
  data = np.log10(data)
```



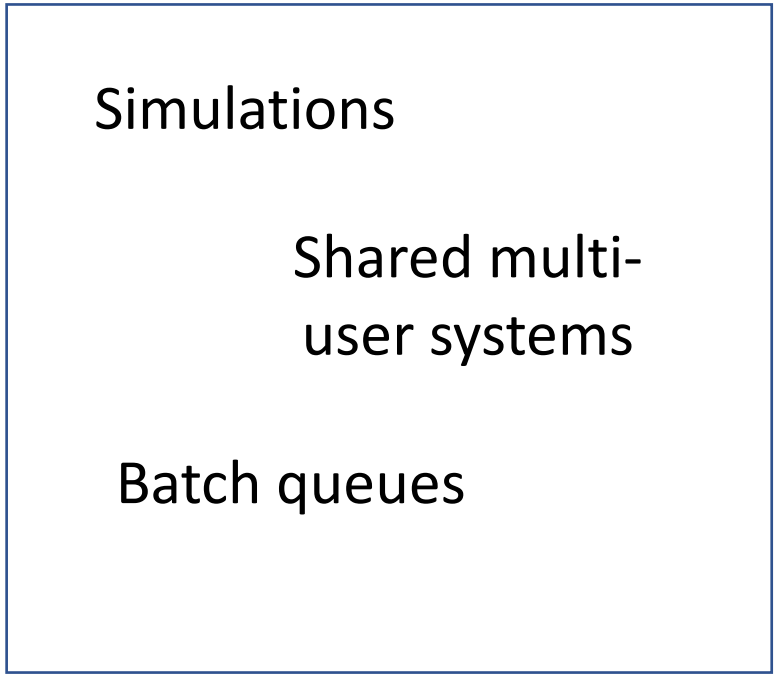
In [ ]:

[VNC config] [rt17603@snowy:~/d... rt17603@snowy:~ rt17603@snowy:~/da... rt17603@snowy:~/Do... Spyder (Python 2.7) Plot data and footprin...

Demand versus Batch Computing

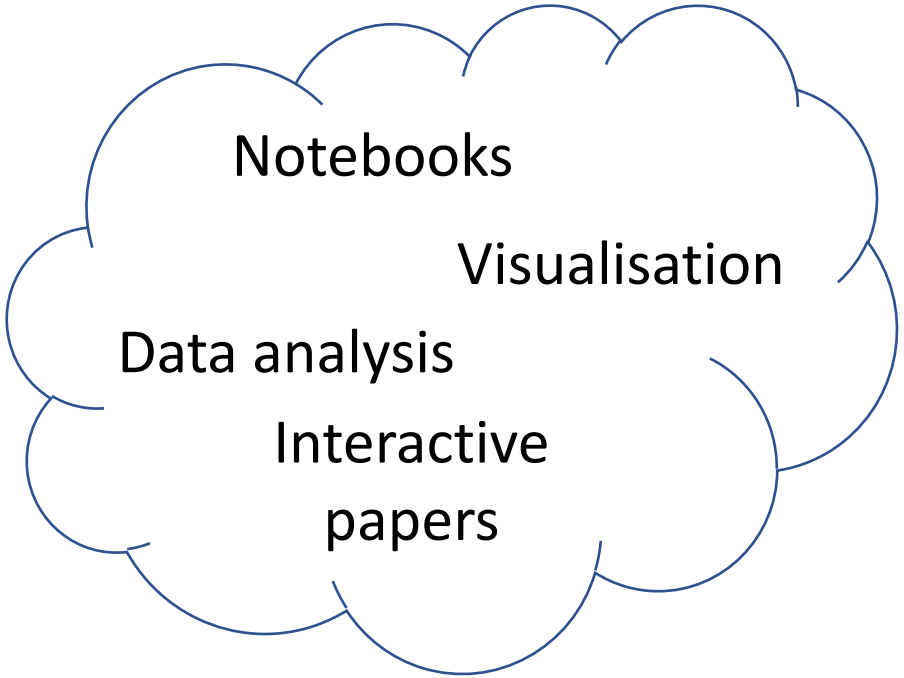


Demand Computing



Batch Computing

Demand versus Batch Computing



Demand Computing

Simulations

Shared multi-
user systems

Batch queues

Batch Computing

NETFLIX

of simulation

The Planetary Supercomputer

Functions == Processes

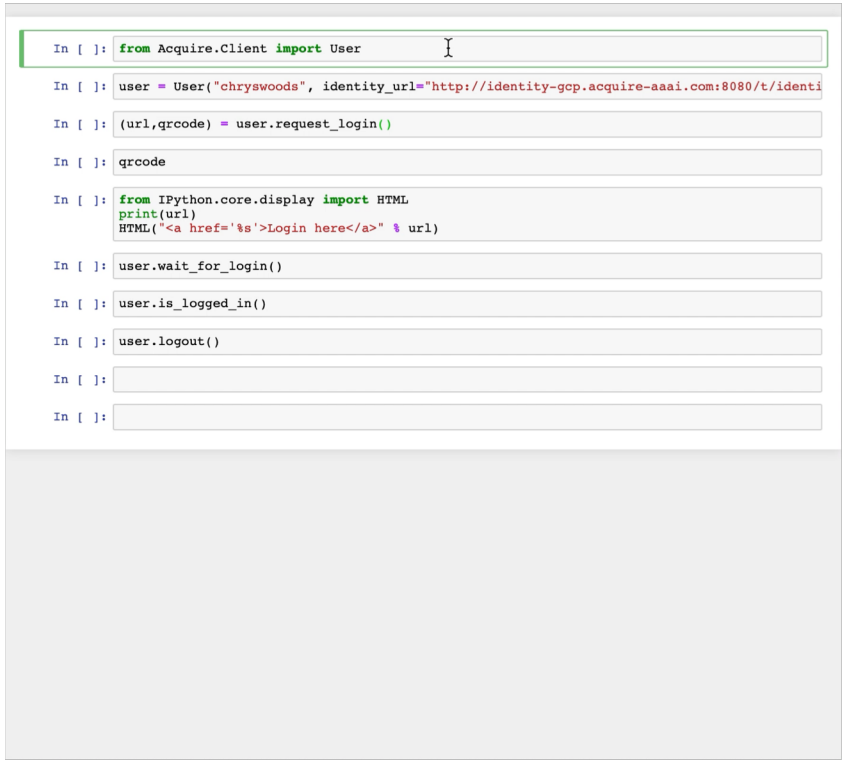
Notebooks == Console/GUI

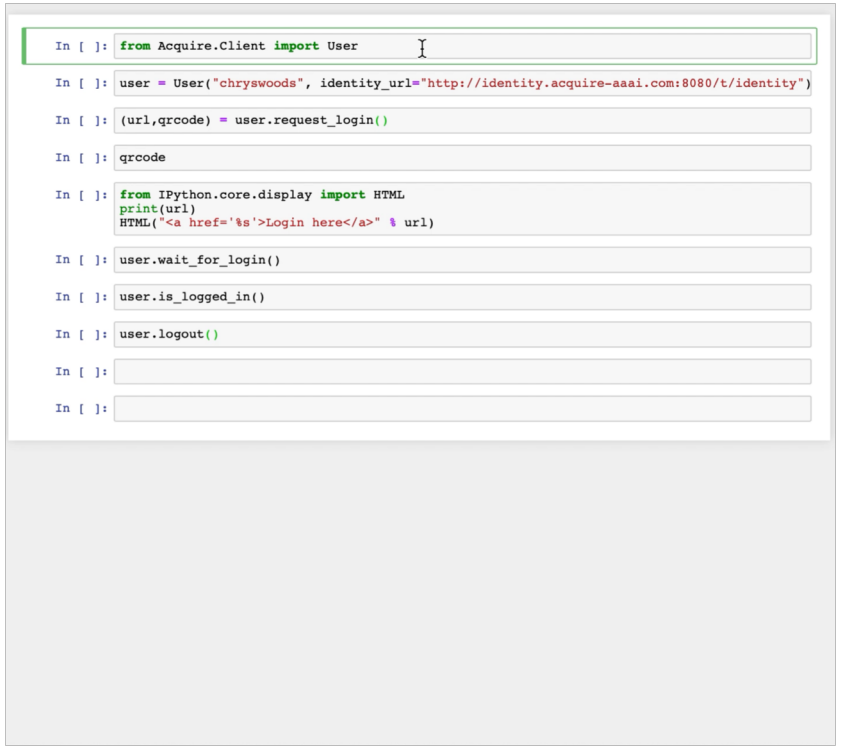
Object Store == Disk/Memory Storage

AAAI == User accounts and resource scheduler

- **Building a service that allows on-demand running of HPC workloads from within interactive Jupyter notebooks with a full user Authentication, Access control and financial Accounting Infrastructure (AAAI)**
- **Fn is an excellent function / serverless platform. Open source ☺**
- **Fully portable – works across clouds!**
- **Notebooks + Serverless + Object Store equals programming the planetary supercomputer**
- **Or, as my students call it, building the Netflix of Simulation**

The Planetary Supercomputer





Fn running
on GCP in Japan

Notebooks running in Seattle

Fn running on Azure
in the Netherlands

OCI Object Store in Germany

Fn running on OCI
in Germany

Acknowledgements

My Research Software Engineering (RSE) Group
Andrew Williams, Chris Edsall, Lester Hedges, Matt Williams

BioSimSpace Research Team
Julien Michel, Antonia Mey, Adrian Mulholland, Charlie Laughton, Francesco Gervasio

HUGS Team
Matt Rigby, Simon O'Doherty, Rachel Tunnicliffe

EPSRC and NERC for funding (EP/N018591/1, EP/P022138/1, NE/S016155/1)

Oracle for providing a lot of compute time and extremely valuable discussions with cloud engineers and the Fn development team. Special thanks to Jenny Tsai-Smith, Phil Bates, Gerardo Viedma and Chad Arimura for all the help and useful technical discussions

Microsoft Azure (particularly Kenji Takeda and Mike Kiernan), and **Google** (particularly Ilias Katsardis) for compute time and engineer discussions to help develop and port to Azure and GCP

Amazon Web Services (particularly Francis Dauncey and Lucy Antysz) and **IBM** (particularly Michael Behrendt and Petrena Prince) for committing to providing compute time and engineer discussions to port to AWS and the IBM cloud

Identity, Access, Accounting, Storage, Compute

- **Identity service supports secure login from any computer, notebook or cloud device. New key/cert generated for each login session and public key/cert uploaded to identity service. Accepted after out-of-band 2FA login. Used to sign-encrypt authorization tokens (json) for resources, actions and identifying users to Fn functions of other services**
- **Accounting service provides fully audited accounting and budget management. Users can create accounts with different daily spend limits, create and destroy accounts easily, create delegated accounts for members of their group etc. All transactions tracked and mandated via authorization tokens verified via trusted identity services**
- **Storage service provides fine-grained control (read/write) to objects or buckets to users via authorization tokens verified via trusted identity services. Bucket/object access is via a Fn-based API with encrypted OCI PARs used on the client and paid for using transaction credit notes and receipts processed via trusted accounting services**
- **Compute service (will) provision a dynamic cluster-in-the-cloud to allow any workload to run on any hardware, jobs described using json, data transferred via OCI PARs generated using the storage service, and authorized and paid for using transaction credit notes and receipts processed via trusted accounting services**
- **Access service (will) collect price lists from different trusted services and chooses which storage and compute services to delegate different work, and mediates payment for those services from users via authorization tokens validated by trusted identity services, and debit+credit notes from trusted accounting services, plus receipts on completion of work from trusted storage and compute services.**

All built as distribute services built on top of Fn ☺