

HDF Server

John Readey
jreadey@hdfgroup.org



My Background

Sr. Architect at The HDF Group

Started in 2014

Focused on HDF for the Cloud

Previously: Dev Manager at Amazon/AWS

More previously: Used HDF5 while a developer at Intel



Agenda

- Motivation for HDF as a service
- HDF Server Architecture
- Client Libraries and Tools
- Case Study
- Kita Lab
- Demo

HDF as a service

- Ideas
 - Web based - provide a RESTful API that is feature compatible with HDF5 Lib API
 - Utilize object storage – cost effective, scalable throughput, redundant
 - Elastic compute – scale throughput by autoscaling compute clusters
 - Compatibility - Provide client SDKs so existing HDF applications can just work
 - Enable web-based applications
- Benefits of SOA (https://en.wikipedia.org/wiki/Service-oriented_architecture) vs library:
 - Language independent (just need JSON and ability to send http requests)
 - Server hardware can scale independently of the clients
 - Server updates don't require clients to change
 - Anywhere reference-able data – ie via URI
 - Parallelization can happen on the server side (less complexity for clients to manage)

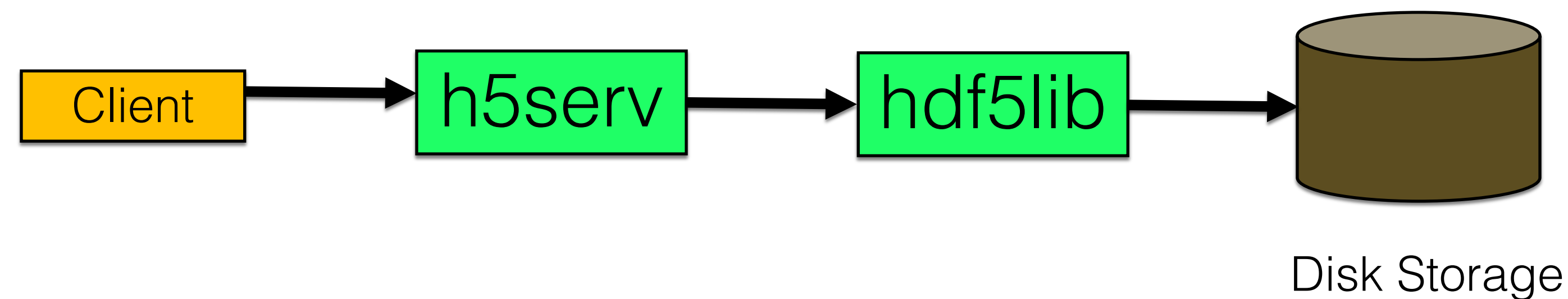
History

- 2014: First definition of the HDF REST API
 - Based on Gerd Heber's white paper:
https://support.hdfgroup.org/pubs/papers/RESTful_HDF5.pdf
- 2015-2016: HDF Server prototype: h5serv
- 2017-2018: Highly Scalable Data Server: HSDS
- 2019: Kita – commercial version of HSDS for AWS Marketplace

Characteristics of REST API

- Client/Server – separate concerns of client and server
 - Server – maintains resources and their state
 - Client – learns about resource state by requesting a representation (binary or json)
- Stateless – keep no client-state on server
- Cacheable – clients can cache responses (mapped to requests)
- Uniform:
 - 4 operations: GET, PUT, POST, DELETE
 - Acting on resources (groups, datasets, datatypes, etc.)
 - Well understood protocol
 - E.g. GET operations are idempotent

- The h5serv project was started in 2015 to demonstrate accessing HDF5 via a RESTful API
- Simple implementation
 - requests translated to HDF5 library calls on server
 - Files stored on local disk
 - Non-scalable (single server, requests handled sequentially, etc)



H5serv to HSDS

- Address scaling challenges with h5serv
 - Handle any request load
 - Any number of clients
 - Any size file, number of files
 - MWMR – Multiple reader/multiple writer support
 - But keep compatibility with REST API as used in h5serv
- These goals led us to a design based on:
 - Object Storage rather than POSIX
 - Container-based microservices approach
 - New implementation of HDF5 rather than using HDF5 library
- New service was called HSDS – Highly Scalable Data Service

Object Storage

- Removes much of the complexity (and functionality!) of POSIX to provide a distributed key-value store
- Compared to POSIX-based storage it's generally cheaper, more scalable, more durable, easier to maintain, etc.
- Popular with cloud providers: AWS S3, Azure Blob Storage, Google Cloud Storage
- Also on prem storage systems: CEPH, OpenStack Swift, OpenIO

Object Storage Challenges for HDF

- Not POSIX! – not compatible with existing HDF5 Lib
- High latency compared with disk I/O
- Not necessarily write/read consistent
- Entire object needs be updated on writes
- High throughput needs some tricks (use many async requests)

For HDF5, using the HDF5 library directly on an object storage system is a non-starter. Will need an alternative solution...

HDF Cloud Schema

How to store HDF5 content in an object storage system?

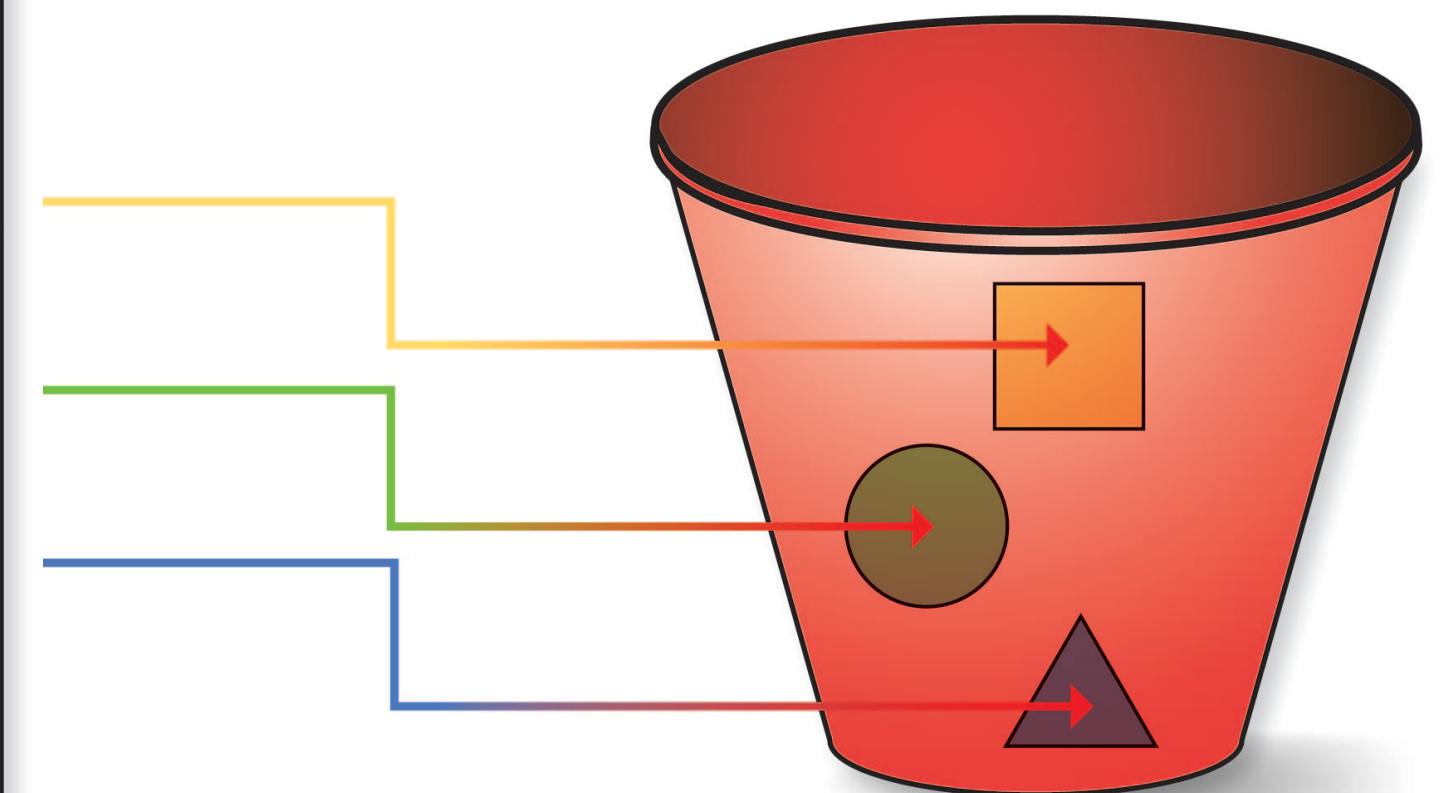
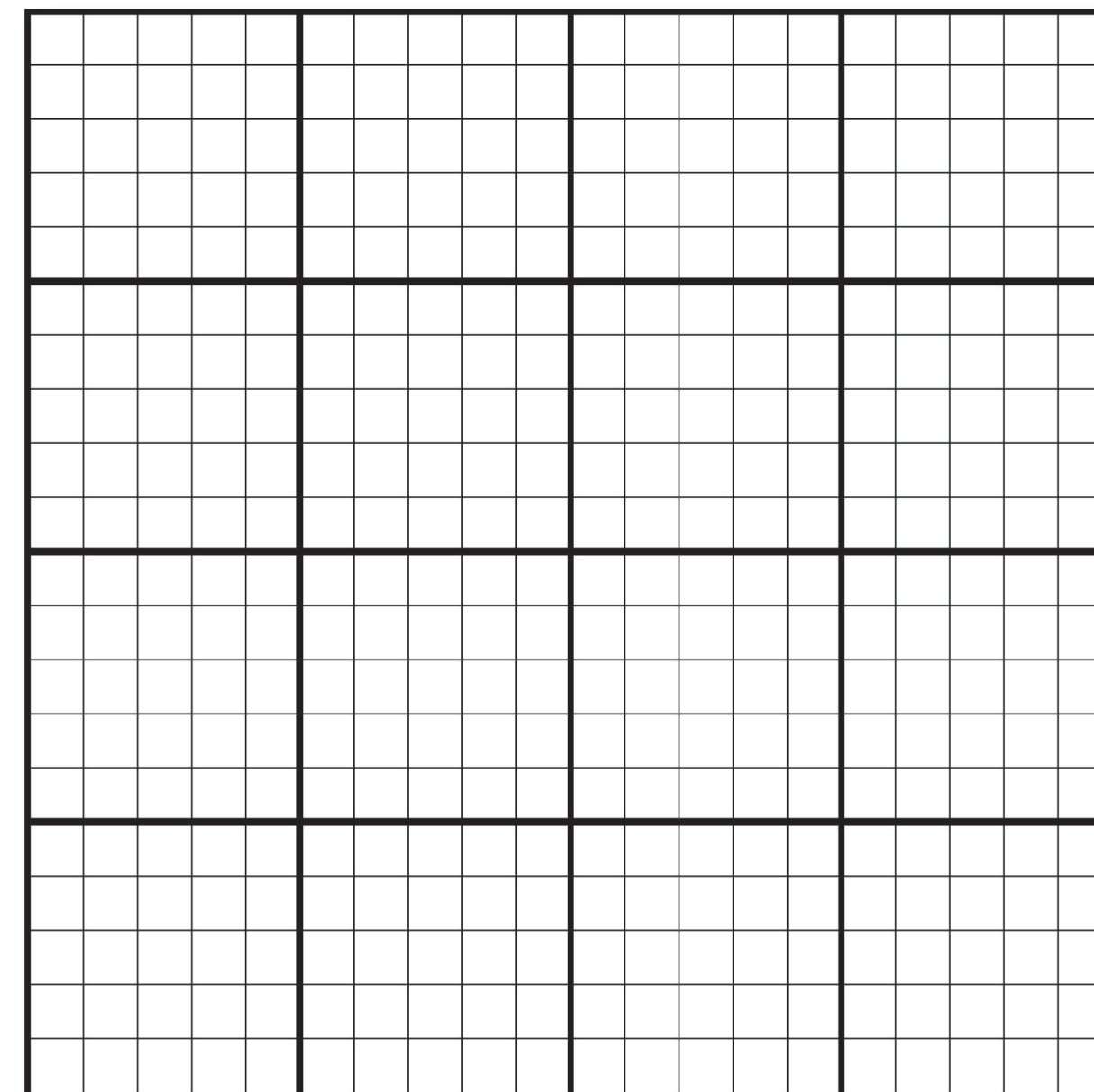
- Limit maximum storage object size
- Support parallelism for read/write
- Only data that is modified needs to be updated
- Multiple clients can be reading/updating the same “file”

Big Idea: Map individual HDF5 objects (datasets, groups, chunks) as Object Storage Objects

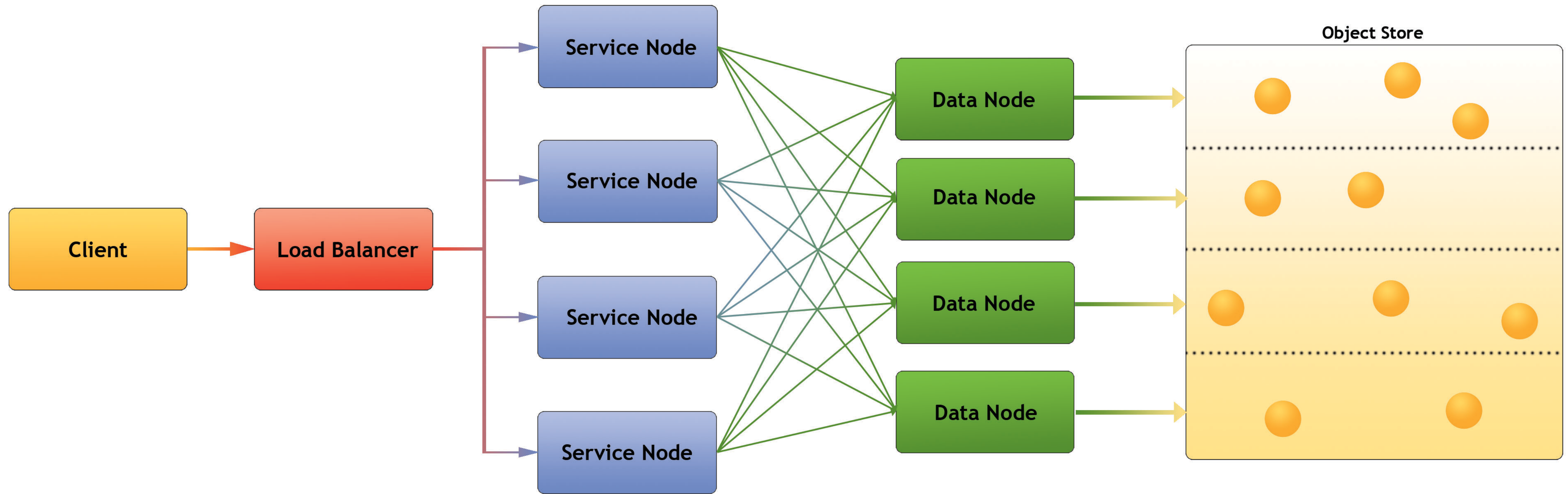
Each chunk (heavy outlines) get persisted as a separate object

Legend:

- *Dataset is partitioned into chunks*
- *Each chunk stored as a storage object*
- *Dataset meta data (type, shape, attributes, etc.) stored in a separate object (as JSON text)*



Architecture



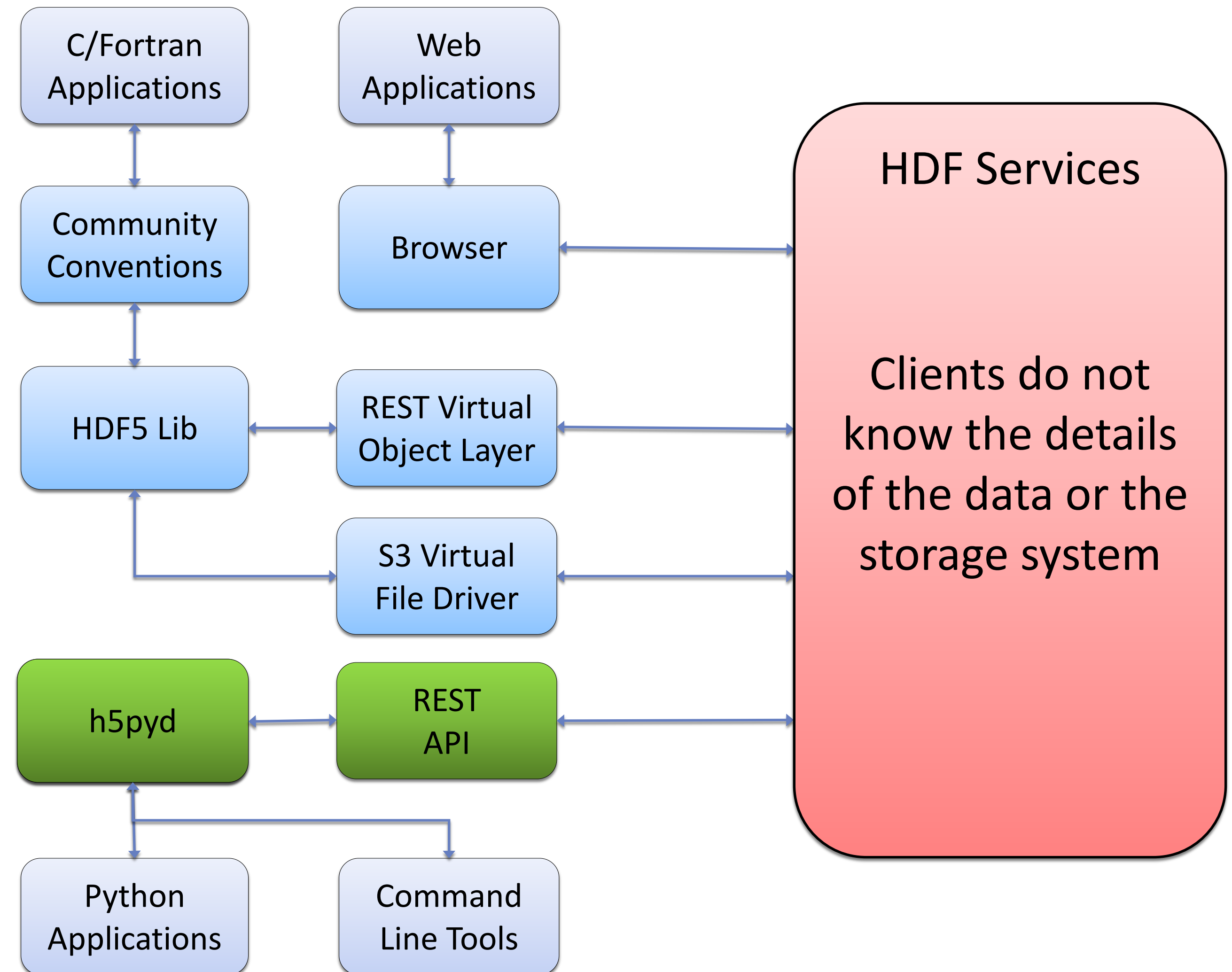
- Load balancer – distributes requests to Service nodes
- Service Nodes – processes requests from clients (with help from Data Nodes)
- Data Nodes – responsible for partition of Object Store
- Object Store: Base storage service (e.g. Ceph)

Architecture

Client SDKs for Python and C are drop-in replacements for libraries used with local files.

No significant code change to access local and cloud based data.

Data Access Options



HSDS to Kita

- **Kita is the open source HSDS plus some special hooks for turnkey solutions**
 - Customers can one—click install using AWS Marketplace
 - Server auto-configures based on type of instance
 - Connects with AWS CloudWatch for monitoring
 - Support provided by The HDF Group

H5pyd – Python client

- H5py is a popular Python package that provide a Pythonic interface to the HDF5 library
- H5pyd (for h5py distributed) provides a h5py compatible h5py for accessing the server
- Pure Python – uses requests package to make http calls to server
- Include several extensions to h5py:
 - List content in folders
 - Get/Set ACLs (access control list)
 - Pytables-like table class and query interface

REST VOL

- The HDF5 VOL architecture is a plugin layer for HDF5
- Public API stays the same, but different back ends can be implemented
- REST VOL substitutes REST API requests for file i/o actions
- C/Fortran applications should be able to run as is
 - (A couple of calls needed to enable the VOL)

Command Line Interface (CLI)

- Accessing HDF via a service means one can't utilize usual shell commands: ls, rm, chmod, etc.
- Command line tools are a set of simple apps to use instead:
 - hinfo: display server version, connect info
 - hls: list content of folder or file
 - hsmv: move file
 - hstouch: create folder or file
 - hsdel: delete a file
 - hsload: upload an HDF5 file
 - hsget: download content from server to an HDF5 file
 - hsacl: create/list/update ACLs (Access Control Lists)
- Implemented in Python & uses h5pyd

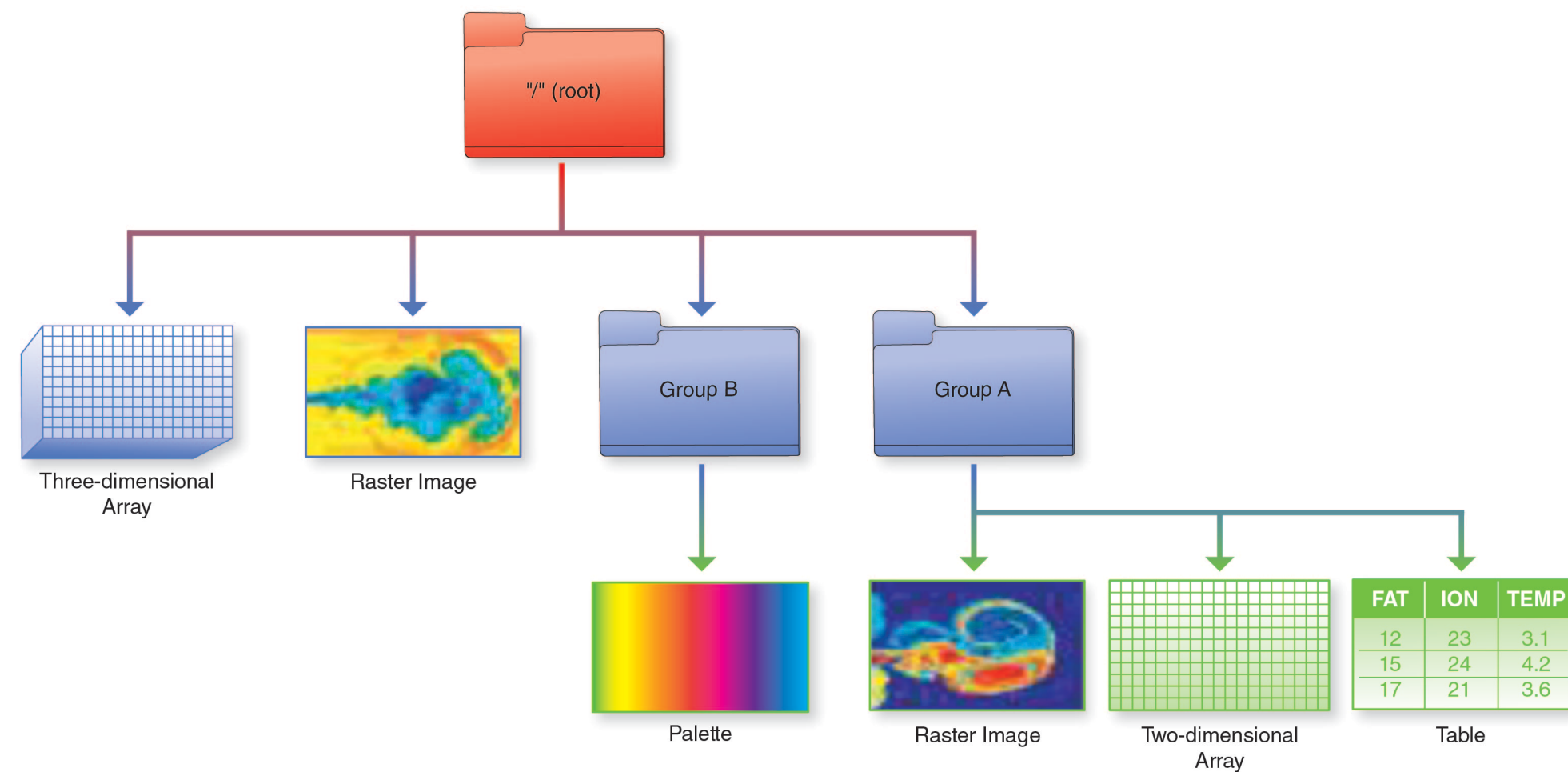
New Features: Supporting traditional HDF5 files

- Downside of the HDF S3 Schema is that data needs be transmogrified
- Since the bulk of the data is usually the chunk data it makes sense to leave this in place:
 - Convert just the metadata of the source HDF5 file to the S3 Schema
 - Store the source file as a S3 object
 - For data reads, metadata provides offset and length into the HDF5 file
 - S3 Range GET returns needed data
- This approach can be used either directly or with HDF Server
- Compared with with reading traditional HDF5 files directly from object storage, you reduce the number of S3 requests needed

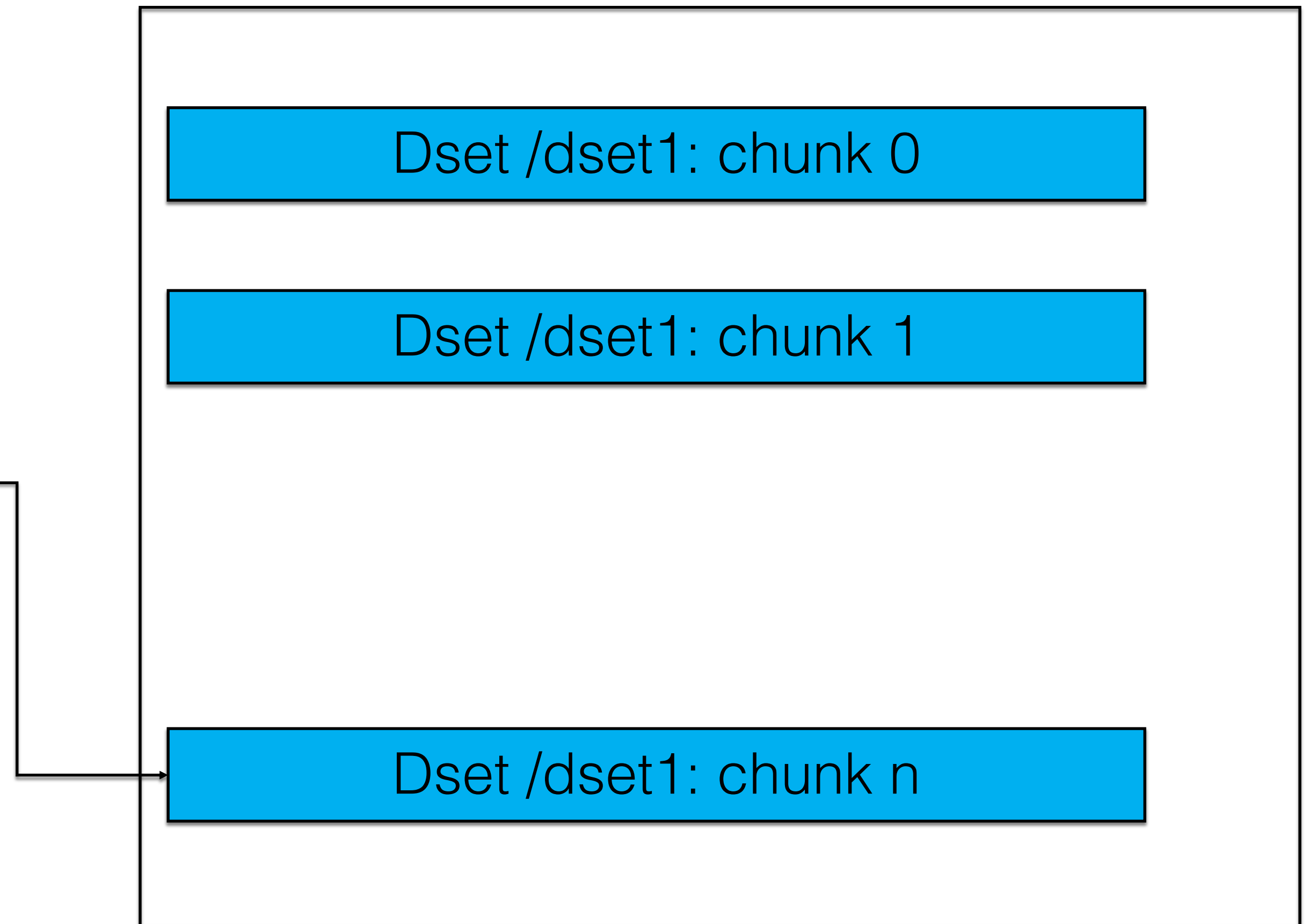
Hybrid Approach: Metadata + HDF5 Files

Imported Metadata (JSON)

HDF5 File stored as S3 object



S3 Range GET(
• S3 Key
• Offset
• Num Bytes)



S3://BIG_REPO/.../AN_HDF5_FILE.h5

New Features: OpenIO Integration



- OpenIO SDS is an object storage solution
- HSDS is now integrated alongside with existing OpenIO services
- Enables users to manage one cluster that provides both storage on HDF access

Jean-Francoise @ OpenIO will be exploring this in more depth in the next talk

New Features: Support for Kubernetes



- Docker is a popular environment for running containers on a single workstation
- Kubernetes provides an environment for running containers across a cluster of computers
- Advantages:
 - Can provide more CPU/Memory/Bandwidth than a single server can provide
 - Services can continue running in advent a host goes down (Kubernetes will reschedule containers that were running on the failed host)
 - The number of container instances (“Pods” in Kubernetes speak) can be dynamically scaled (e.g. based on service load)
 - Additional hardware can be spun up without interrupting currently running services
- Kubernetes is supported by all major cloud providers and can also be used for on-prem installations
- Provides common platform to avoid vendor lock in

Case Study: NREL

- Problem: NREL (National Renewable Energy Laboratory) had uploaded 50TBs of simulation data to AWS, but there was no practical way for others to get the data
- NREL contracted with us to load the data into HSDS and validate that the service could be used by external users to extract data as needed:
 - Access a geographic region at a given time
 - Access a time series for a given geographic point



Case Study NREL – moving the data to the Cloud

- NREL used HPC run to create datasets that represents wind, temperature, precipitation over the continental US from 2007-13
- Known as the WTK dataset – output of simulation was 500 TB, then down sampled to 50 TB
- NREL moved the data to AWS S3 using the Snowball appliance

FedEx beats Fiber, data was shipped to Amazon via Snowball appliance



NREL Case Study - Data Import

- Source data was aggregated to one virtual file managed by HDFCloud using a custom script
- Input:
 - 84 HDF5 files
 - ~500GB each
 - 40 datasets
 - one month of data per file
- Output:
 - 1 HSDS domain (file), ~27MM S3 objects
 - ~40TB
 - 40 datasets
 - Seven years of data

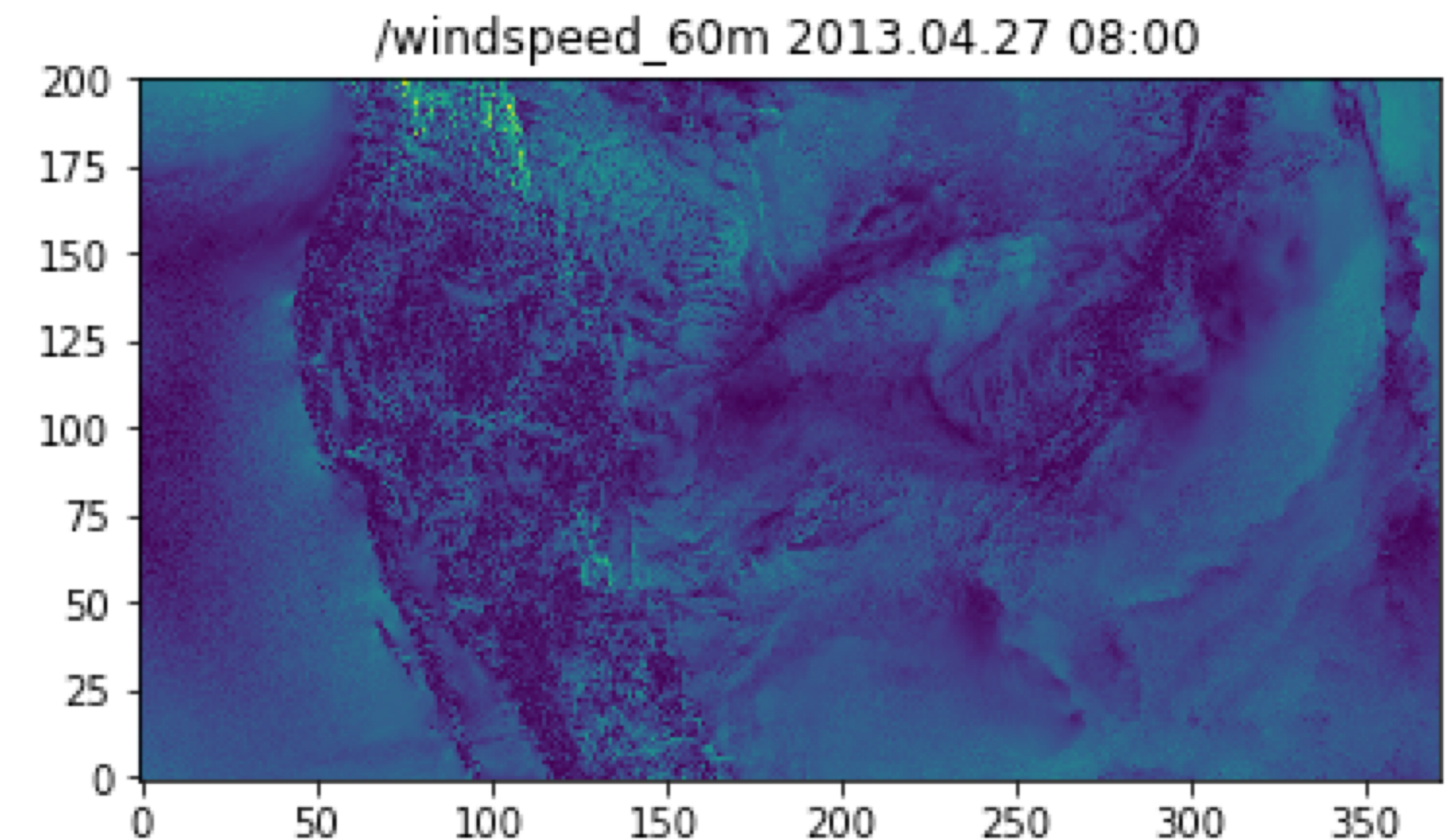
Exposing the data as a single virtual file simplifies application programming

NREL Case Study - Client Access

- Clients use Python h5pyd package or REST API to access data
- H5pyd provides h5py compatible API
- Accessing a slice:

```
data = dset[timestep, :, :]
```

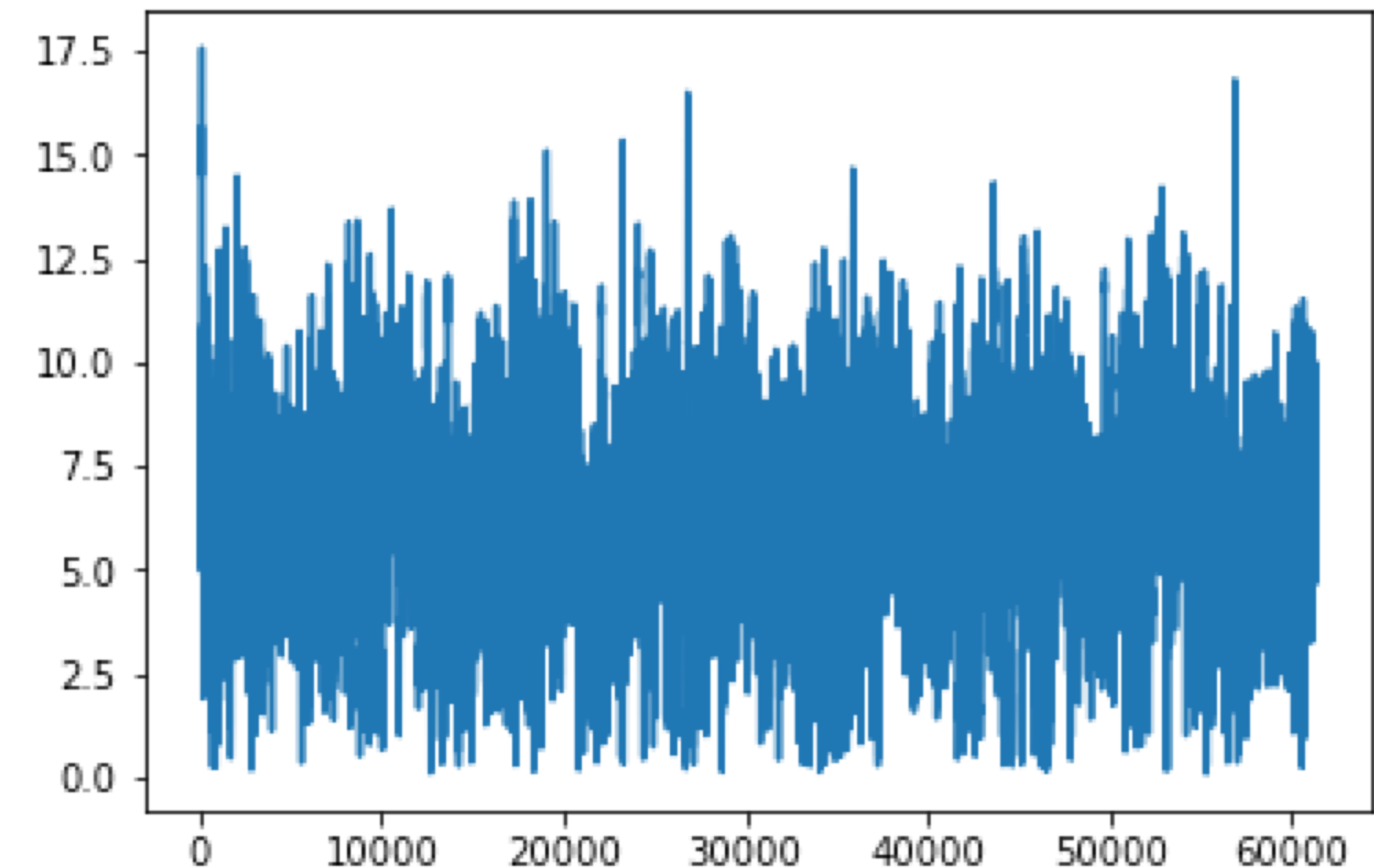
- Dataset shape is (61368, 1602, 2976)
- Chunk shape is (25, 89, 186)
- So this request would access $1602/89 * 2976/186 = 18 * 16 = 288$ Chunks
- 403 MB S3 Data read, 18 MB returned



NREL Case Study - Time Series Extraction

- Getting all the values for a particular geographic point:

```
data = dset[:,lat_index,lon_index]
```



- As before, we have:
 - dataset shape (61368, 1602, 2976)
 - Chunk shape (25,89,186)
- So this request would access $61368/25 = 2456$ Chunks
- 3.4 GB S3 Data read, 240 KB returned

NREL Case Study - Conclusion

- NREL has been running HSDS with the WTK data for more than a year
- Is available for public use
- 100's of users, millions of requests handled
- Cost has been moderate (1 8-core server, 50TB S3 storage)
- NREL is now expanding their use of HSDS to include other datasets

New Features: Speeding up meta data access

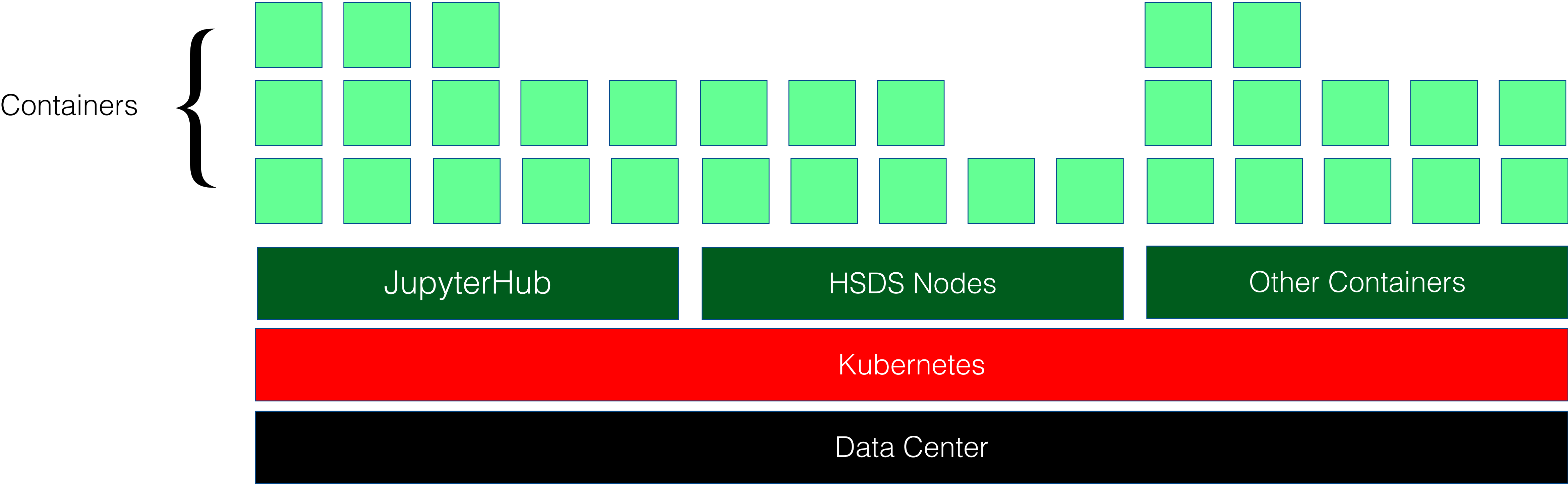
- Use case: Many HDF Schemas (e.g. NEXUS!) have deep hierarchies with lots of attributes and small datasets
- Challenge: Back and forth with server can slow down access as smaller objects are loaded into client memory
- Idea: Pull in all metadata for a file in one request
- Implementation: The REST API now supports a request parameter to return all metadata for a file in one request
- H5pyd utilizes this by default with files open as read-only
- Result: Access to attributes, links, dataset information (everything except dataset reads) is super fast.

Collaboration with HDF Server and Jupyter

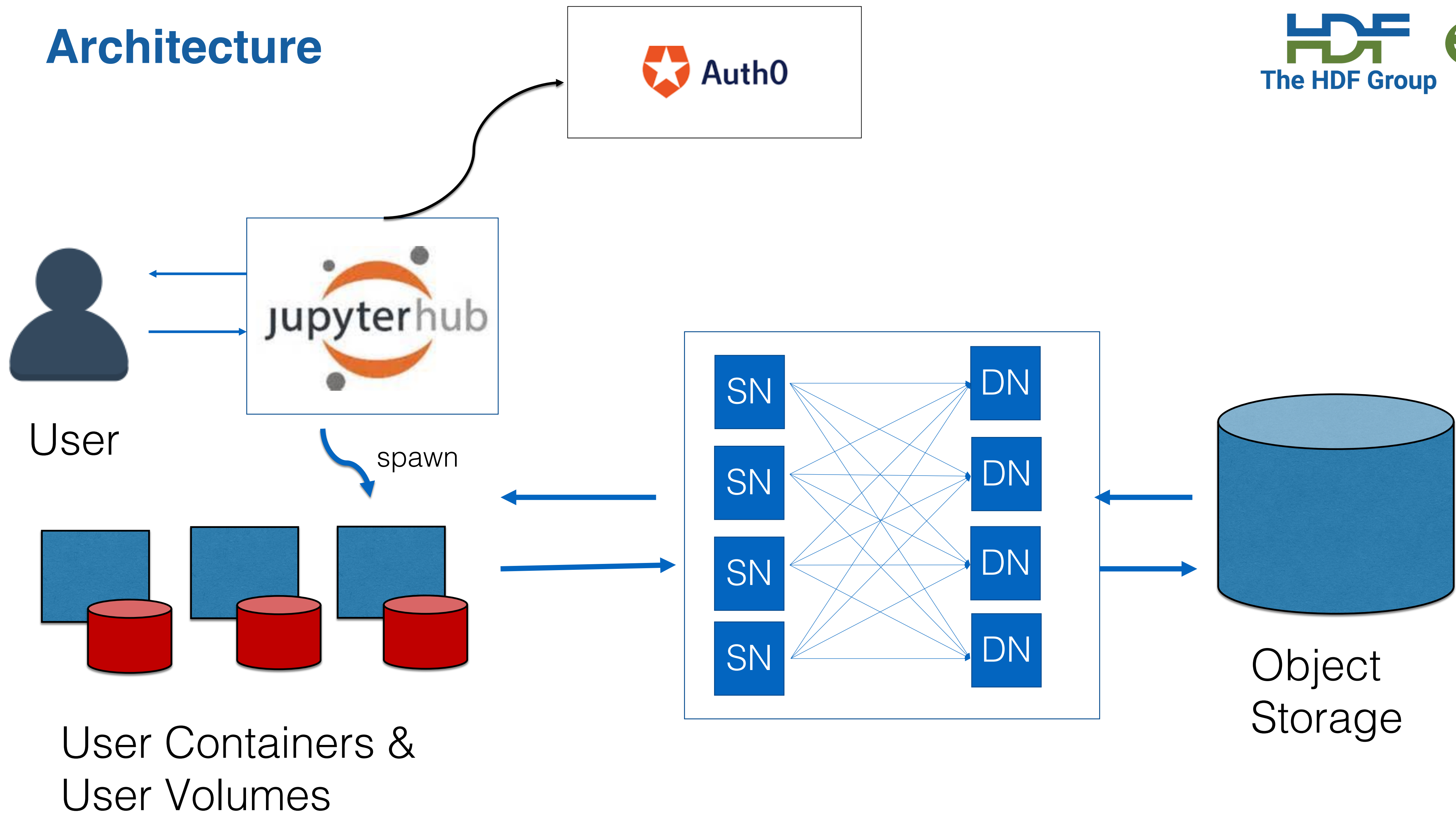
- HSDS/Kita enables data to be shared between widely distributed users
- Jupyter notebooks enable interactive computing via applications running on a remote computer
- Together these provide an effective tool for remotely working with large datasets
 - Don't move the data when you don't have to
- Shared/editable notebooks (e.g. like with Google Docs) is on the roadmap for Project Jupyter but not here yet

JupyterHub, JupyterLab, and Kubernetes

- JupyterHub enables Python notebooks to be run remotely
 - Nothing to configure/install on users desktop
- JupyterLab is the next generation Notebook interface
 - Revamped UI
 - Provides a terminal window (yay!)
- Kubernetes is a container management platform that enables containers to run across a cluster of computers
 - Any number of containers can be run assuming that the cluster is large enough
 - Each Jupyter users environment runs as a container with an attached disk volume
 - Can run other services/jobs in addition to Jupyter. E.g.: HSDS, Dask



Architecture



Collaboration with Jupyter

- Users can share notebooks via Github, Dropbox, etc.
- Data files not so easily shared!
 - Each user gets his own volume that can't be accessed by other users
 - Volume size is typically small, may not be sufficient for scientific data
- HSDS/Kita solves this conundrum (at least for HDF data)
 - Any size data collection can be managed by the server (using object storage)
 - Data can be shared with anyone
 - Users can control who has access to which data

HDF Kita Lab

- The HDF Group is providing a hosted JupyterLab environment at <https://hdflab.hdfgroup.org>
- Open to anyone (you just need to register with The HDF Group)
- Provides access to HDF Kita
- Comes with sample notebooks, tutorials, datasets
- There is a small subscription fee (\$10/month) after 2 month trial period



HDF Kita Lab

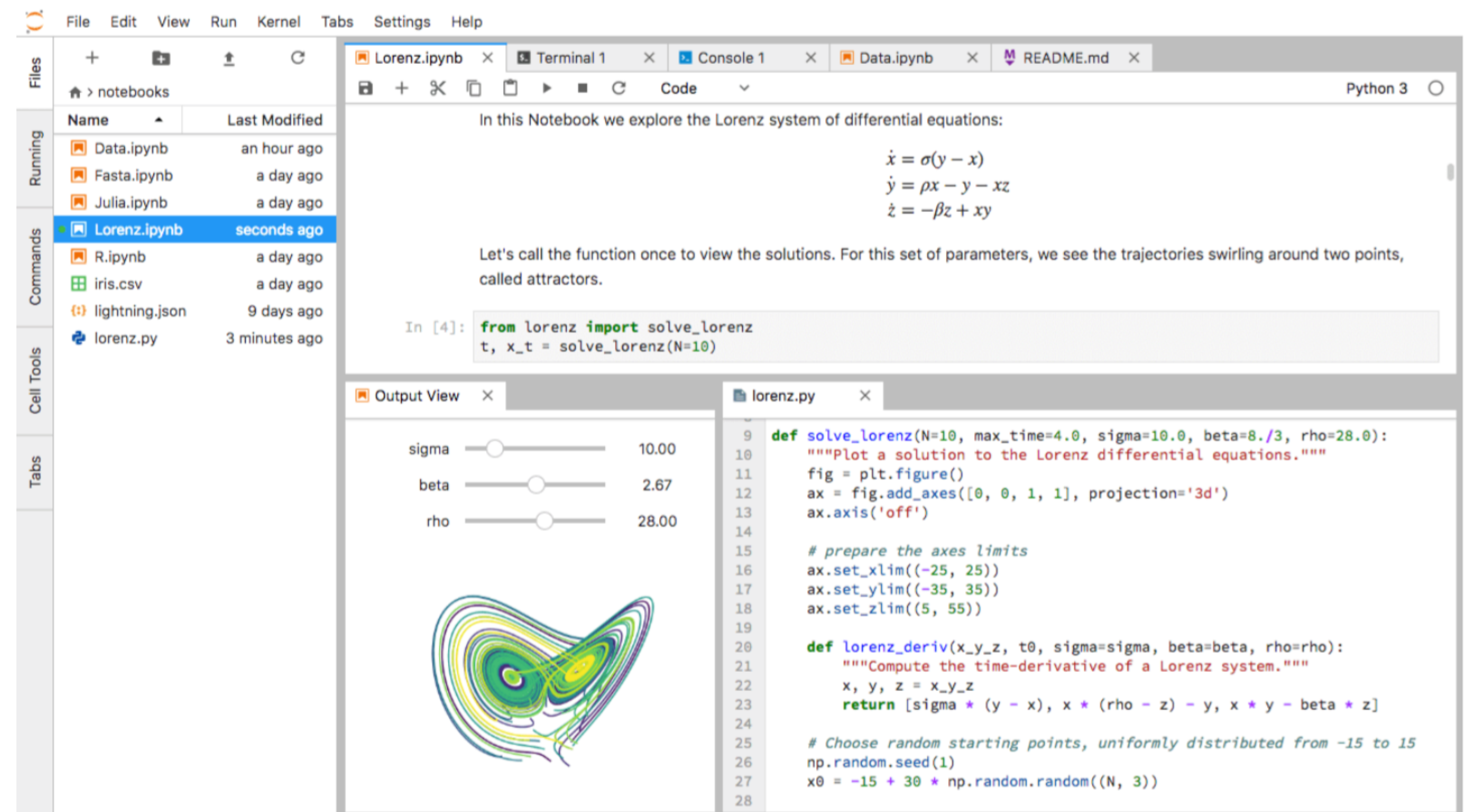


Kubernetes



JupyterLab

- HDF Kita Lab is based on JupyterLab
- JupyterLab is the next-generation web-based interface for running Python notebooks
- Extends classic Ipython environment with:
 - Content browser for documents
 - Upload/downloading of files
 - Terminal App
- HDF Kita Lab Extends JupyterLab:
 - Auto configures Kita Server
 - FAQ Page on launcher
 - HDF branding



The screenshot displays the JupyterLab web interface. On the left, a sidebar contains a 'Files' panel showing a list of notebooks and files, with 'Lorenz.ipynb' selected. Below it are 'Running' and 'Commands' panels. The main area is divided into three panes: a top pane for the notebook, a bottom-left pane for the 'Output View', and a bottom-right pane for the 'lorenz.py' code file. The notebook pane shows the Lorenz system of differential equations and a code cell that has been executed. The 'Output View' pane displays a 3D plot of the Lorenz attractor, with sliders for parameters sigma, beta, and rho. The 'lorenz.py' pane shows the Python code used to generate the plot.

File Edit View Run Kernel Tabs Settings Help

Files

notebooks

Name	Last Modified
Data.ipynb	an hour ago
Fasta.ipynb	a day ago
Julia.ipynb	a day ago
Lorenz.ipynb	seconds ago
R.ipynb	a day ago
iris.csv	a day ago
lightning.json	9 days ago
lorenz.py	3 minutes ago

Running

Commands

Cell Tools

Tabs

Lorenz.ipynb x Terminal 1 x Console 1 x Data.ipynb x README.md x

Code Python 3

In this Notebook we explore the Lorenz system of differential equations:

$$\begin{aligned}\dot{x} &= \sigma(y - x) \\ \dot{y} &= \rho x - y - xz \\ \dot{z} &= -\beta z + xy\end{aligned}$$

Let's call the function once to view the solutions. For this set of parameters, we see the trajectories swirling around two points, called attractors.

In [4]: `from lorenz import solve_lorenz
t, x_t = solve_lorenz(N=10)`

Output View x

lorenz.py x

sigma 10.00
beta 2.67
rho 28.00

def solve_lorenz(N=10, max_time=4.0, sigma=10.0, beta=8./3, rho=28.0):
 """Plot a solution to the Lorenz differential equations."""
 fig = plt.figure()
 ax = fig.add_axes([0, 0, 1, 1], projection='3d')
 ax.axis('off')

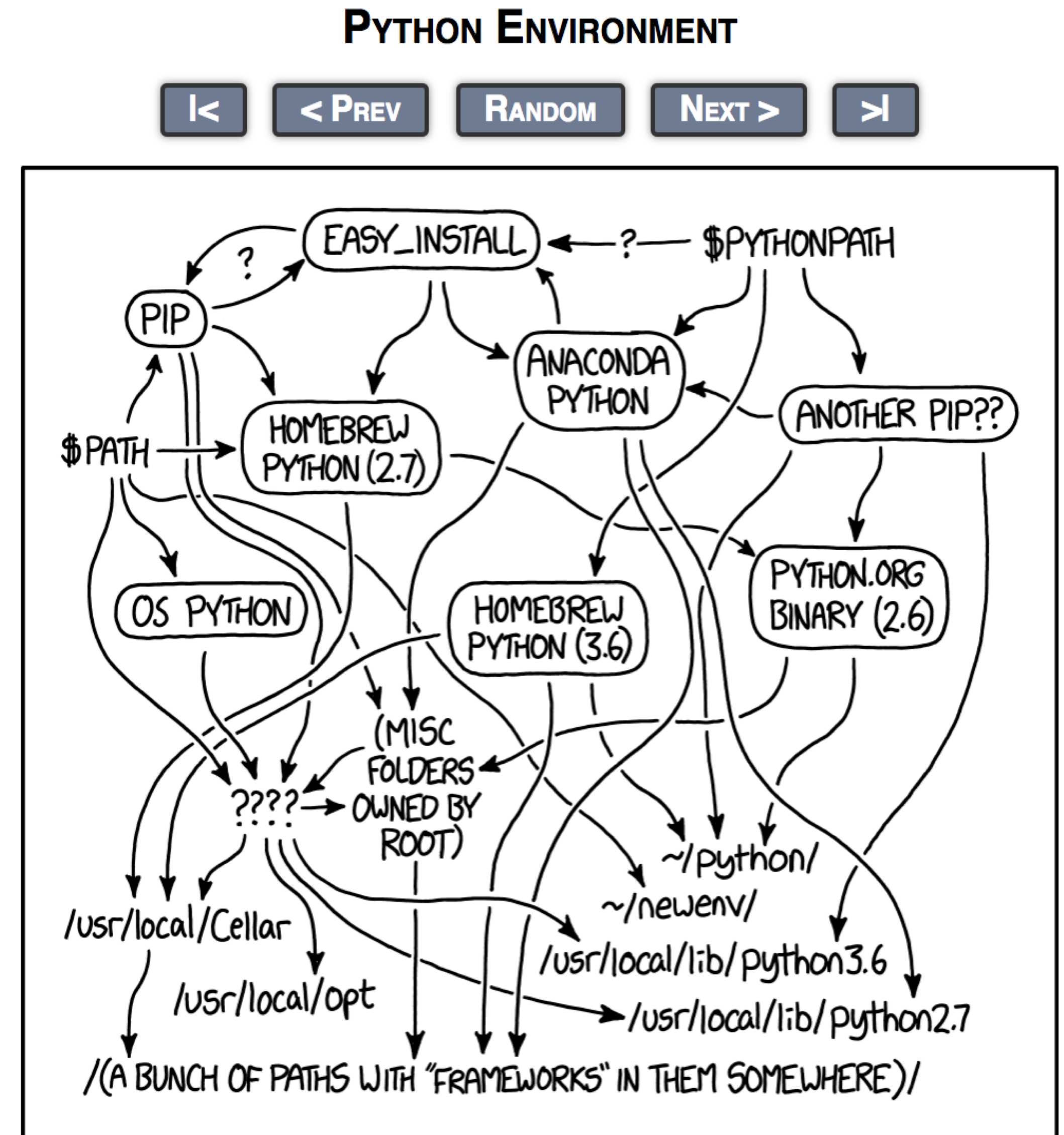
 # prepare the axes limits
 ax.set_xlim((-25, 25))
 ax.set_ylim((-35, 35))
 ax.set_zlim((5, 55))

 def lorenz_deriv(x_y_z, t0, sigma=sigma, beta=beta, rho=rho):
 """Compute the time-derivative of a Lorenz system."""
 x, y, z = x_y_z
 return [sigma * (y - x), x * (rho - z) - y, x * y - beta * z]

 # Choose random starting points, uniformly distributed from -15 to 15
 np.random.seed(1)
 x0 = -15 + 30 * np.random.random((N, 3))
 return x0

Simplify your life...

- No messing with Python, package installs, AWS, etc.
- Data ready for you
- Simple means to harness compute cluster
- JupyterHub admin can setup user image with commonly used packages



MY PYTHON ENVIRONMENT HAS BECOME SO DEGRADED THAT MY LAPTOP HAS BEEN DECLARED A SUPERFUND SITE.

Kita Lab Features

- HDF Kita Lab runs on AWS in a Kubernetes cluster
 - Cluster can scale to handle different number of users
 - Each user gets:
 - 1 CPU Core (2.5GHz Xeon)
 - 8 GB RAM
 - 10 GB Disk
 - 100 GB S3 Storage
- Access to HDF Kita Server
 - Ability to read/write HDF data stored on S3
- User environment configured for commonly used Python Packages for HDF users:
 - H5py(d), pandas, h5netcdf, xarray, bokeh, dask
- HDF Kita Command Line tools:
 - Hsinfo, hsls, hsget, hsload, etc.

References

- RESTFul HDF white paper:
https://support.hdfgroup.org/pubs/papers/RESTful_HDF5.pdf
- HDF Schema:
https://github.com/HDFGroup/hsds/blob/master/docs/design/obj_store_schema/obj_store_schema_v2.md
- HDF Server Security Blog article: <https://www.hdfgroup.org/2015/12/serve-protect-web-security-hdf5/>
- SciPy2017 talk:
https://s3.amazonaws.com/hdfgroup/docs/hdf_data_services_scipy2017.pdf
- AWS Big Data Blog article: <https://aws.amazon.com/blogs/big-data/power-from-wind-open-data-on-aws/>
- Project Jupyter for Scientist: <https://www.nature.com/articles/d41586-018-07196-1>

Github Repos

- H5serv: <https://github.com/HDFGroup/h5serv>
- HSDS: <https://github.com/HDFGroup/hsds>
- H5pyd: <https://github.com/HDFGroup/h5pyd>
- HDF Lab Notebook examples:
https://github.com/HDFGroup/hdflab_examples
- REST API spec: <https://github.com/HDFGroup/hdf-rest-api>

Issue requests are welcome!

Pull requests even more welcomed!