Large Scale Data Files, Object Stores and Deep Learning –

Lessons Learned While Looking for Extra-terrestrial Life

Gil Vernik – IBM Corporation

Graham Mackintosh – IBM Corporation

SPARK SUMMIT 2017

In this session…

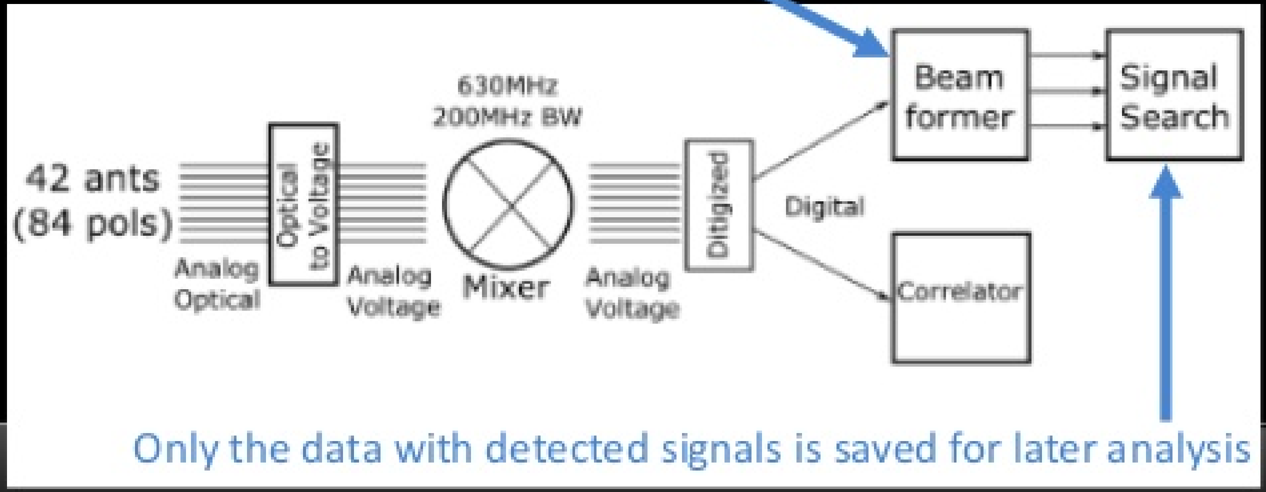
- IBM Spark@SETI
- Deep Learning meets SETI Science
- Beyond Classification – Novel Observations and Analytics
- The SETI Project – By The Numbers
- Back to Earth… Spark and Object Store
- Conclusions and Take-Aways

IBM Spark@SETI

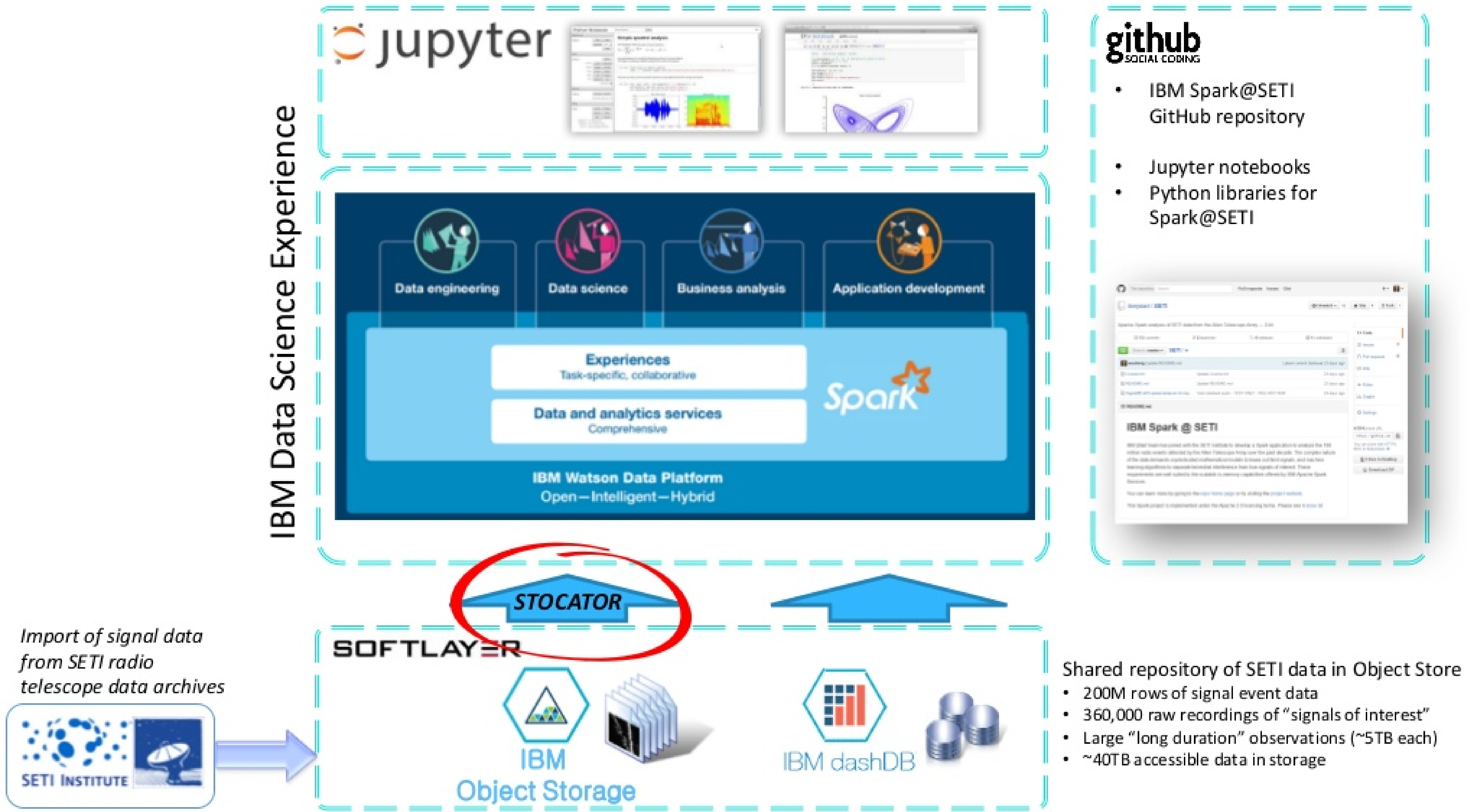
SETI Institute Backgrounder

- Headquartered in Mountain View, CA. Founded 1984. 150 Scientists, researchers and staff.
- The mission of the SETI Institute is to explore the potential for extra-terrestrial life…. search for narrow band radio signals in the frequency range of 1GHz to 10GHz which could be evidence of intelligence outside our solar system.
- Allen Telescope Array (ATA) – Phased Array Synthetic Dish – 3 Beams

4.5TB data every hour



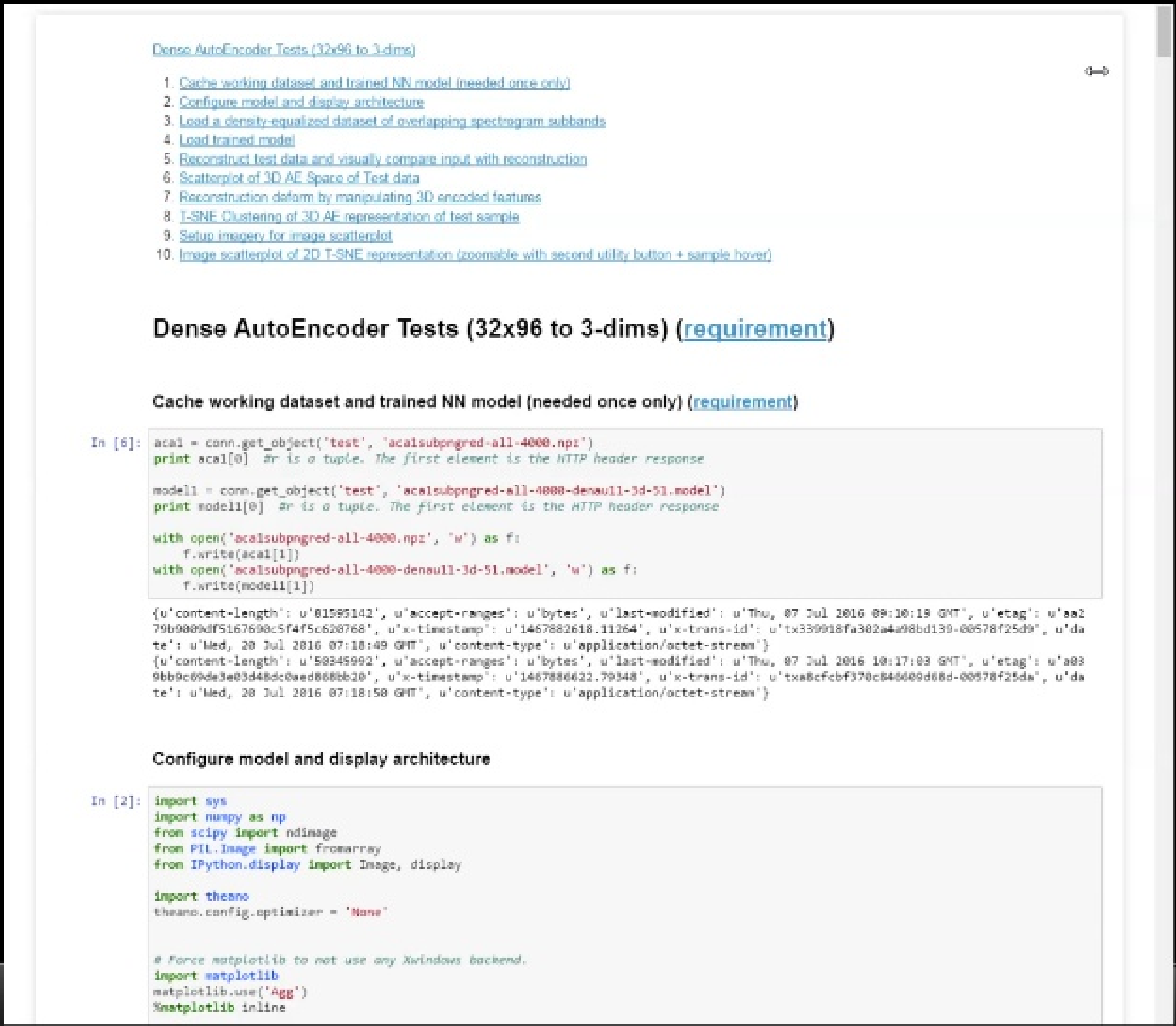
Spark@SETI – Deep analytic capabilities for professional astronomers



Spark@SETI Deep Learning Signal Classifier

Jupyter notebook showing complex radio signals being classified based on morphology (shape) and other features.

Neural net model was trained on the IBM Cognitive Compute Cluster (750 NVIDIA K80 GPUs) and imported into IBM Data Science Experience



Dense AutoEncoder Tests (32x96 to 3-dims) (requirement)

Cache working dataset and trained NN model (needed once only) (requirement)

In [6]:

```
aca1 = conn.get_object('test', 'aca1subpngred-all-4000.npz')
print aca1[0]  #r is a tuple. The first element is the HTTP header response

model1 = conn.get_object('test', 'aca1subpngred-all-4000-denau11-3d-51.model')
print model1[0]  #r is a tuple. The first element is the HTTP header response

with open('aca1subpngred-all-4000.npz', 'w') as f:
    f.write(aca1[1])
with open('aca1subpngred-all-4000-denau11-3d-51.model', 'w') as f:
    f.write(model1[1])
```

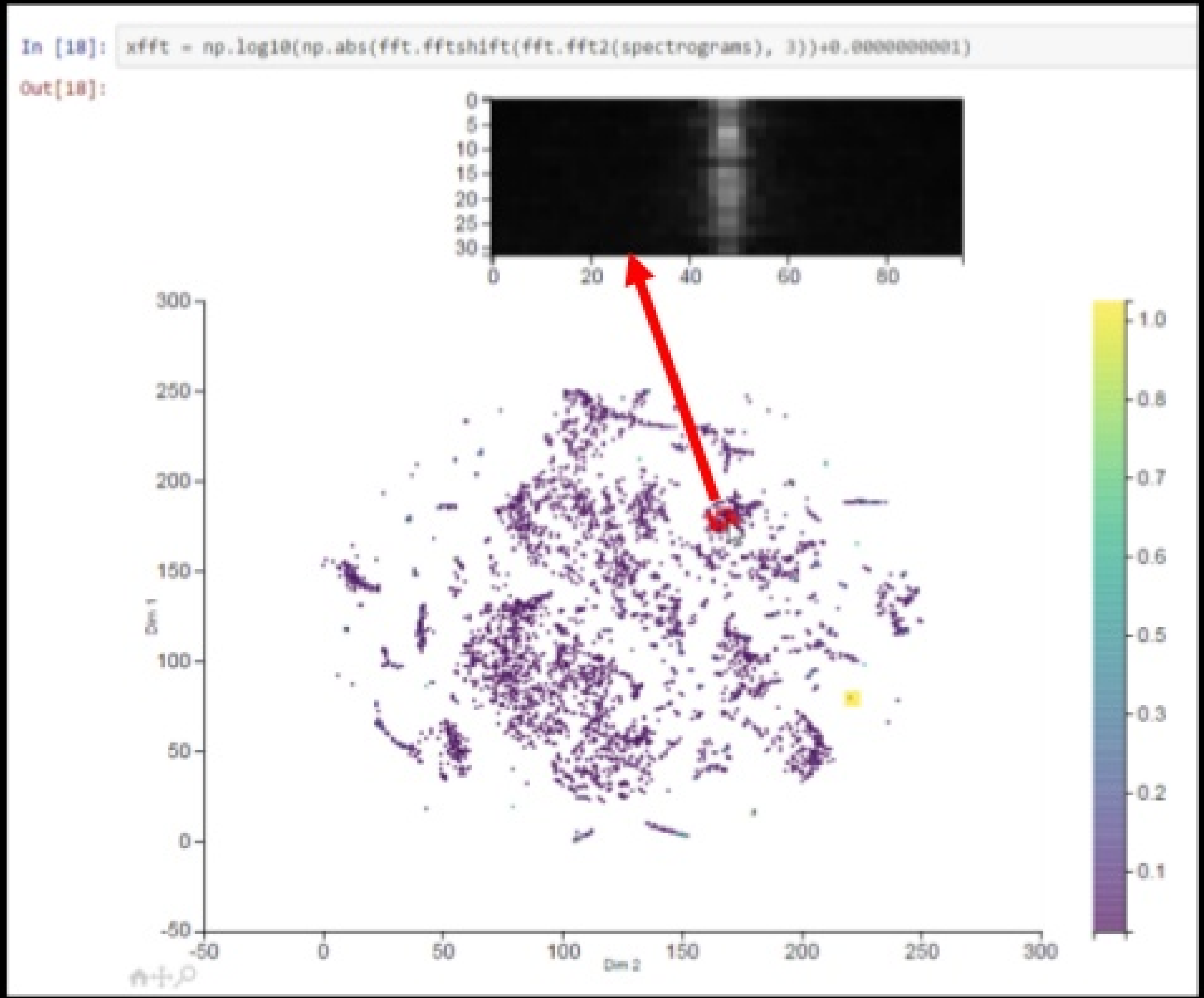

{u'content-length': u'81595142', u'accept-ranges': u'bytes', u'last-modified': u'Thu, 07 Jul 2016 09:10:19 GMT', u'etag': u'aa279b9009df5167690c5f4f5c620768', u'x-timestamp': u'1467882618.11264', u'x-trans-id': u'tx339918fa302a4a08bd139-00578f25d9', u'date': u'Wed, 20 Jul 2016 07:18:49 GMT', u'content-type': u'application/octet-stream'}
{u'content-length': u'58345992', u'accept-ranges': u'bytes', u'last-modified': u'Thu, 07 Jul 2016 10:17:03 GMT', u'etag': u'a039bb9c09de3e03d48dc0aed868bb20', u'x-timestamp': u'1467886622.79348', u'x-trans-id': u'txa8cfcbf370c846609d68d-00578f25da', u'date': u'Wed, 20 Jul 2016 07:18:50 GMT', u'content-type': u'application/octet-stream'}

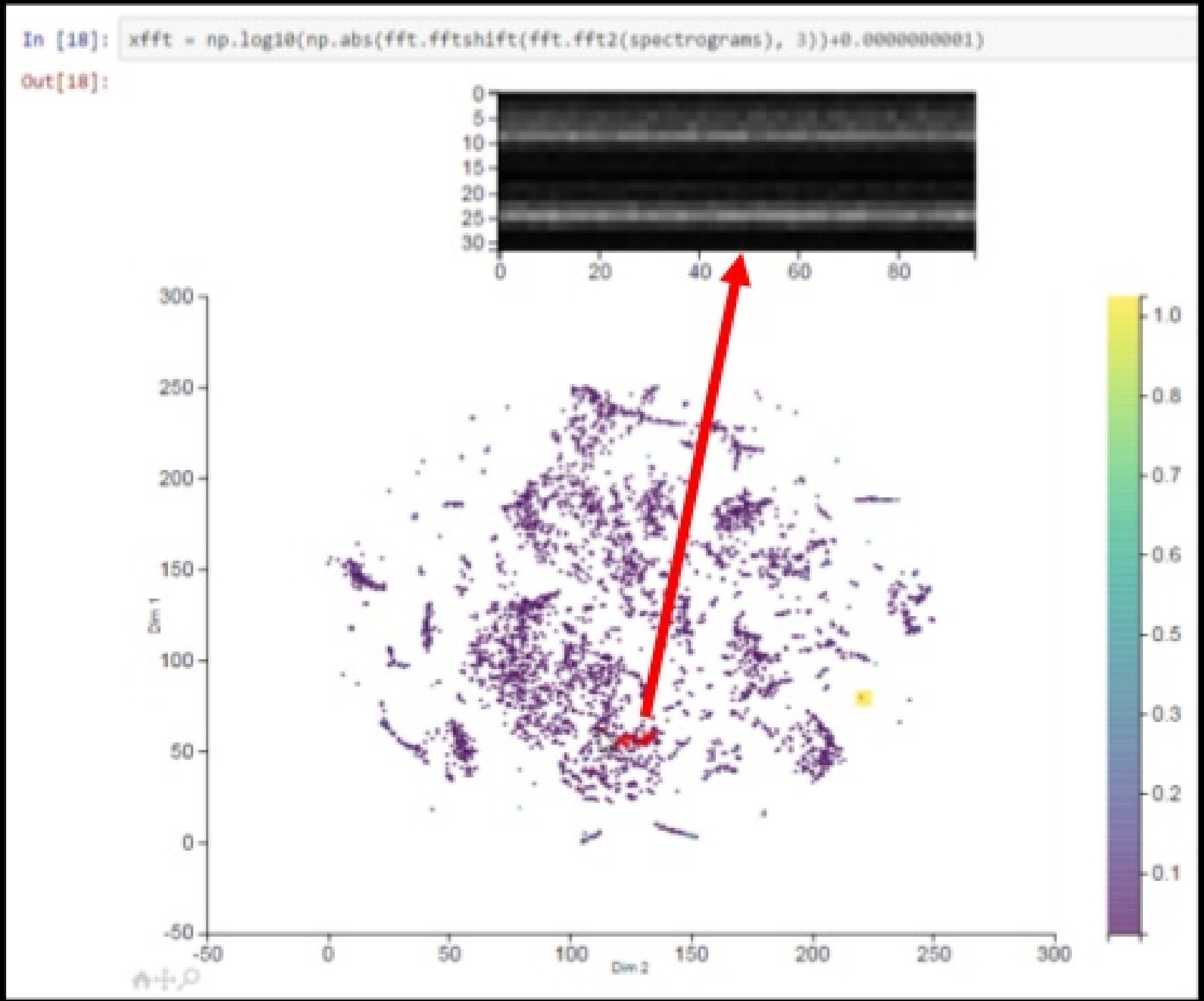
Configure model and display architecture

In [2]:

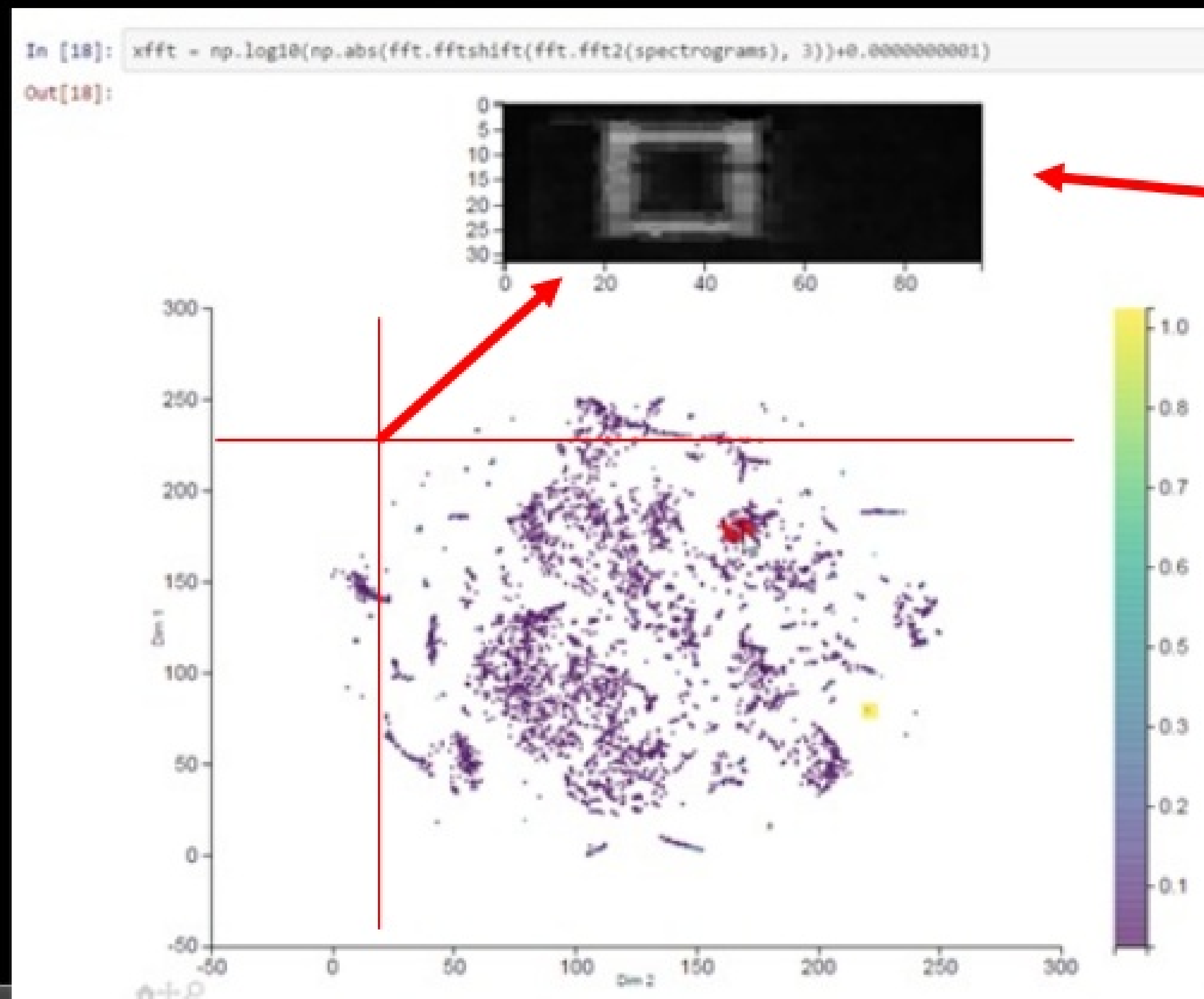
```
import sys
import numpy as np
from scipy import ndimage
from PIL.Image import fromarray
from IPython.display import Image, display

import theano
theano.config.optimizer = 'None'


# Force matplotlib to not use any Xwindows backend.
import matplotlib
matplotlib.use('Agg')
%matplotlib inline
```

From raw antennae voltage data streams to classified signals, all with unsupervised machine learning…





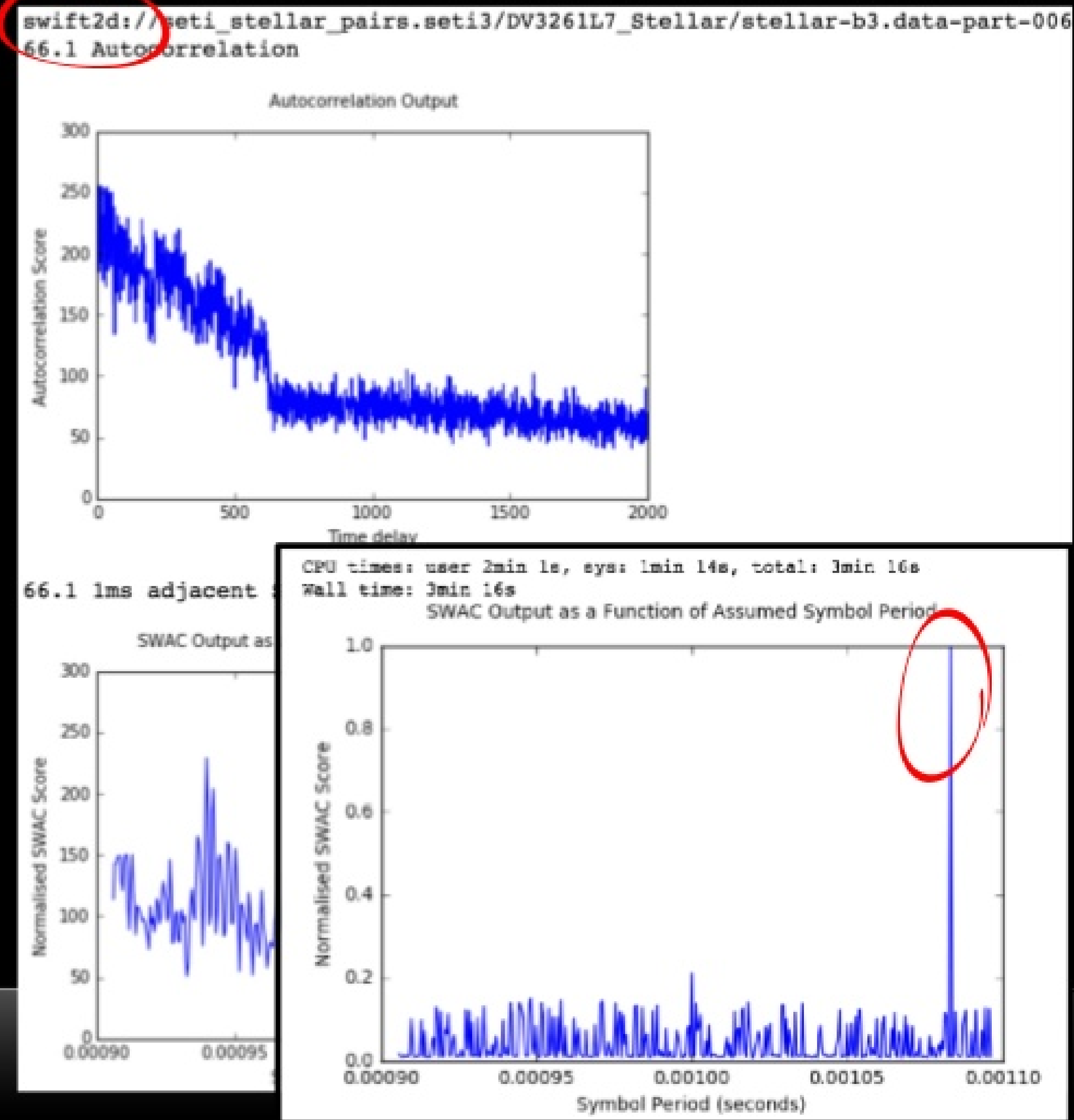
From raw antennae voltage data streams to classified signals, all with unsupervised machine learning…



(NOT REAL - ILLUSTRATIVE ONLY!)

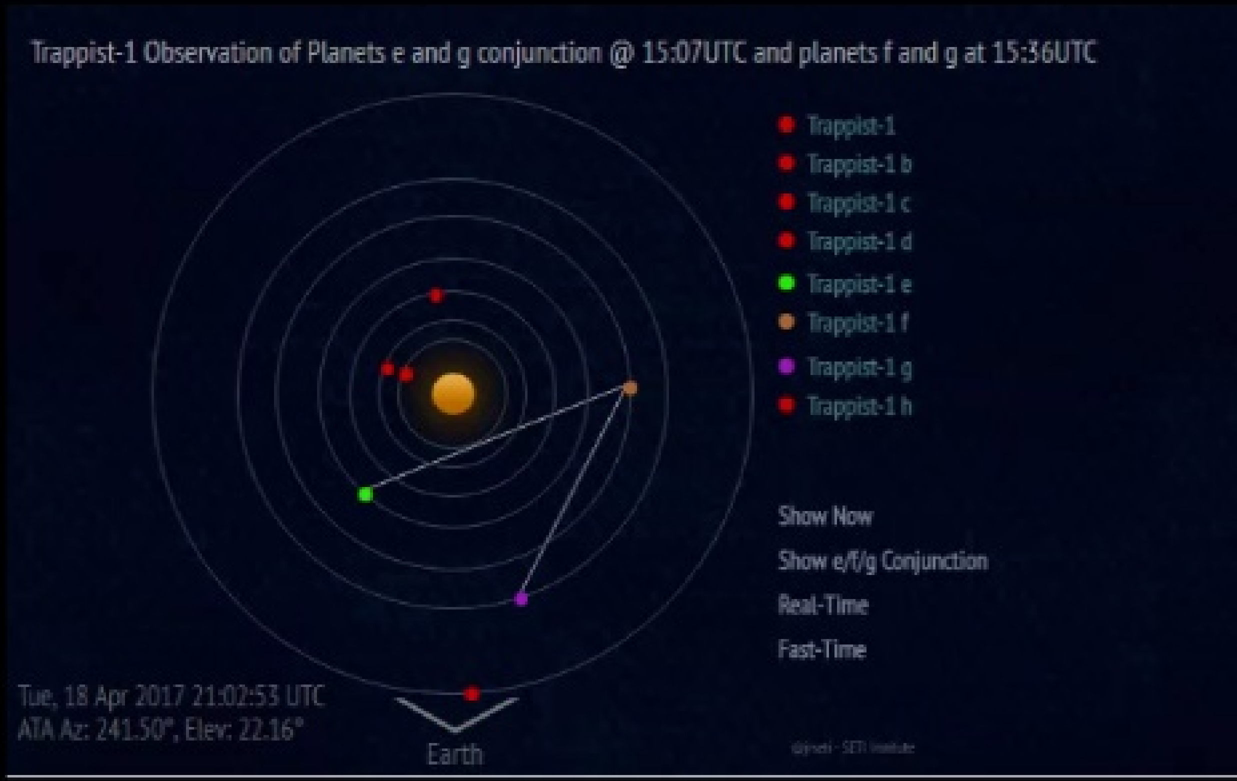
Beyond Classification…

- Classification is great for intentional “beacon” signals (and for getting rid of RFI - radio frequency interference)
- Searching for beacons assumes we are “important enough” to do that … we have to look for signals which were not intended for us
- Leakage and Eavesdropping – much harder to detect, data and compute heavy



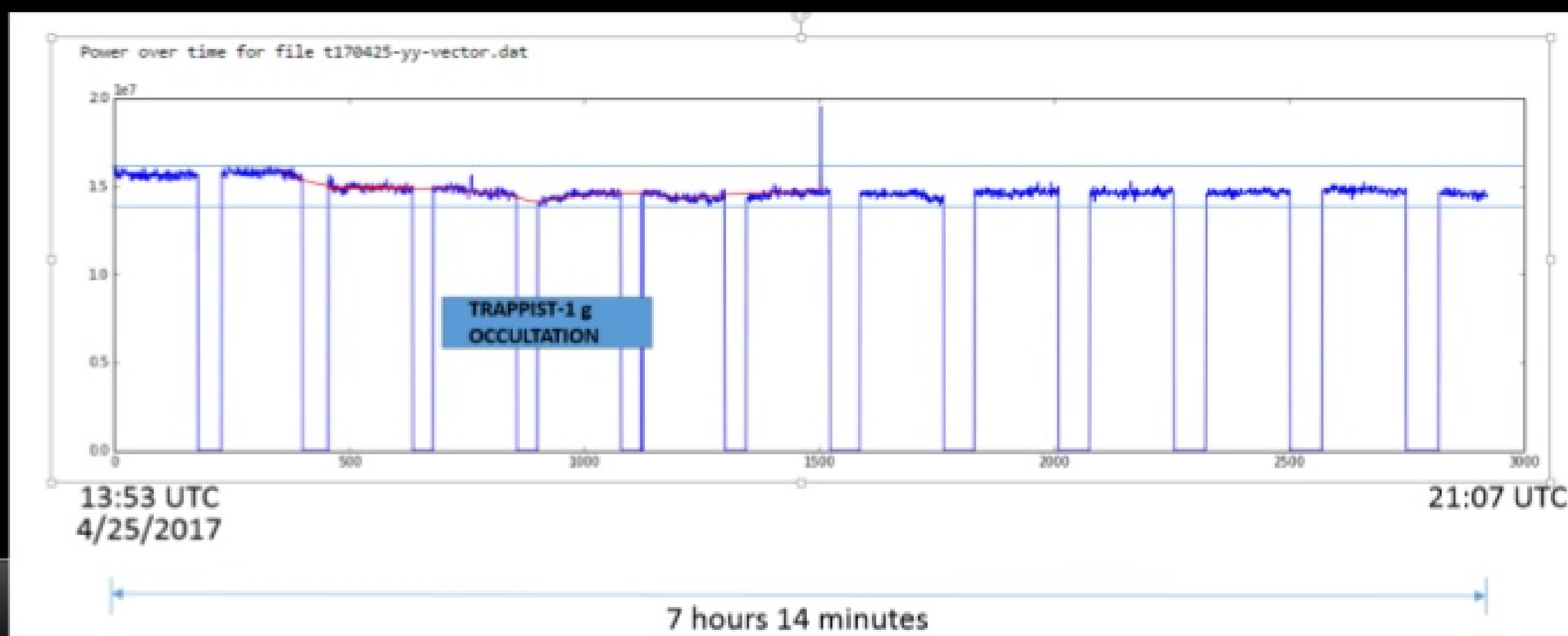
TRAPPIST-1 – One stop shopping for ET

- Newly discovered system with 7 rocky planets, three in the habitable zone (mathematically, liquid water could be on the surface)
- Insanely good opportunity for eavesdropping and leakage detection
- Detected by transit (Kepler-K2)… orbital plane is within 0.3 degrees of perfect alignment with our line of sight = occultations and conjunctions
- Close… only 40 LY
- Recently found… propagated error for orbital predictions is around 2 seconds
- Tight orbits (2-7 days) means LOTS of opportunities to see conjunctions and occultations

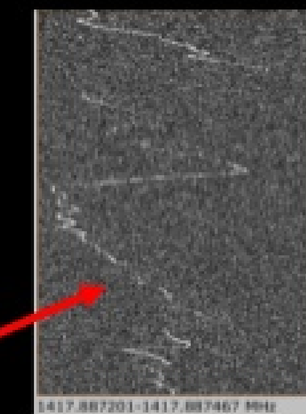


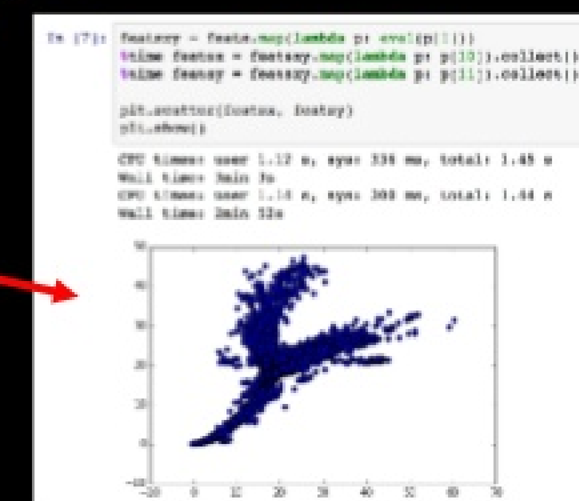
TRAPPIST-1 … so here’s the latest news:

- System model implemented in Python and run on IBM DSX / Spark… a “*conjunction and occultation finder*” accurate to < 45 seconds
- Six observations completed and data queued for trickle upload using Stocator
- Occultation ground-to-cloud completed for power curve analysis… don’t expect anything definitive until we have data folded several observations together.



The Spark@SETI Project – By the Numbers





- 200 million signal events
  - Doppler shift corrected in 22 minutes wall time on 30 executor Spark cluster
- 14 million complex amplitude files in Object Store
  - Each binary file contains 90 second ‘snapshot’ of raw antennae voltages
  - 14M files = 1TB of raw signal data... FFT into 14M spectrograms followed by feature extraction for clustering ~12 hours
- Long duration observations = 2 beams @ 2.5TB each
  - Auto-correlation and SWAC analysis looking for wideband… best case is order $n \bullet \log(n)$, Symbol-Wise Autocorrelation (SWAC) is order nD, where D is the number of delay values calculated (which is generally less than n)
  - Wide-band analysis…. 5TB processed for wideband detection in approximately 13.5 hours wall time.

How did we make all this work?

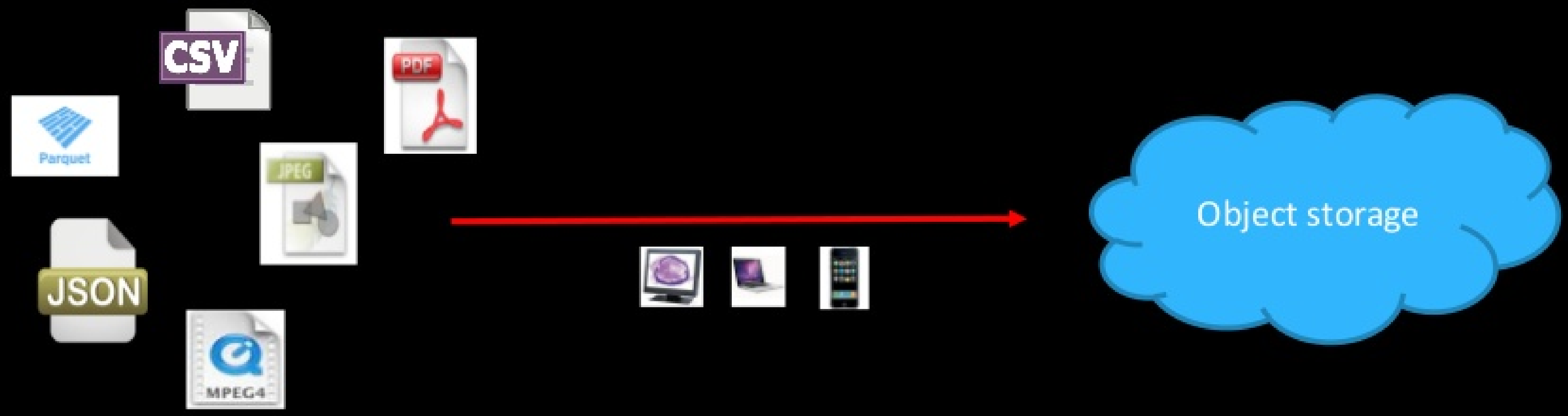
- High performance access to large amounts of data in Object Store was the game changer
- Over to Gil…

Back to the Earth

- What technologies SETI@Spark depends on?
  - Apache Spark
  - Data objects in the variety of formats
  - IBM Cloud Object Storage

What is an object store?

- Object store is a perfect place to store files (we call them objects)
- Each object consists of rich metadata and the data itself



Organize data in the object storage

- Data objects are organized inside the buckets or containers
- Each data object may contain a name with delimiters, usually "/"
- This allows to group objects inside buckets (pseudo directories), an analogy to the directories in file systems but without the overhead or scalability limits of lots of directories

data object

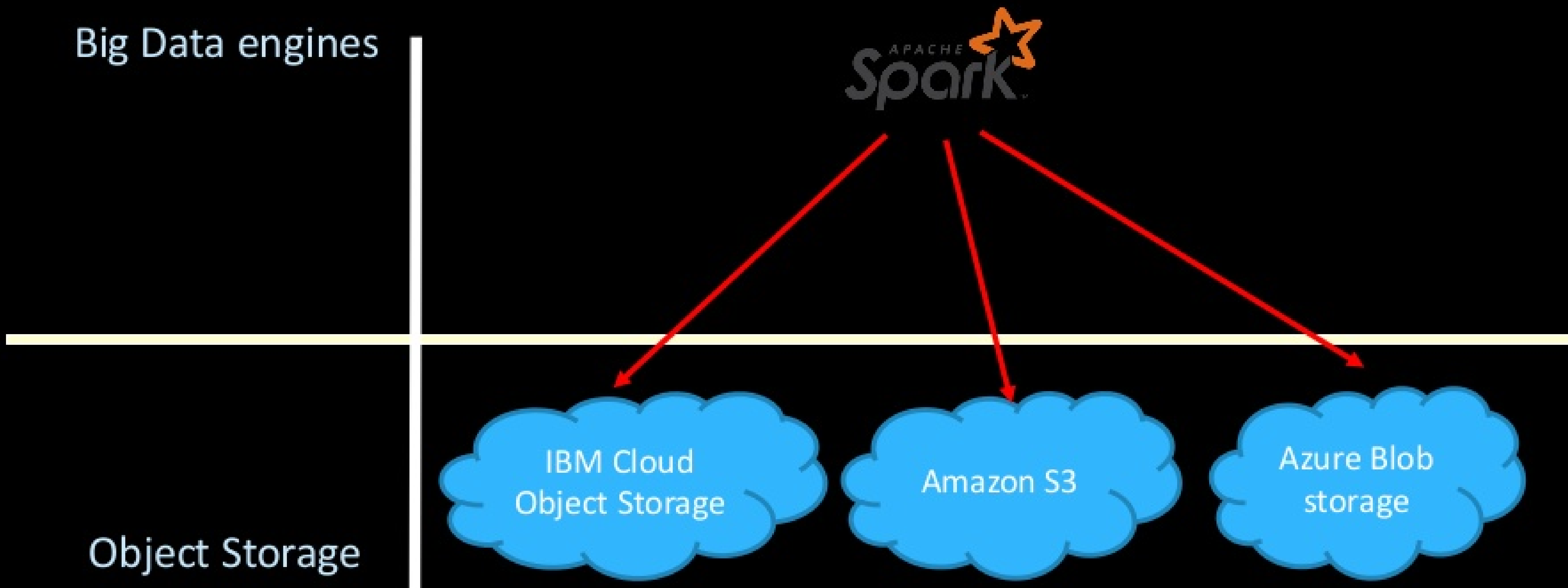
bucket

mytalks/year=2016/month=5/day=24/data-palooza.pdf
mytalks/year=2017/month=5/day=24/hadoop-strata.pdf
mytalks/year=2017/month=6/day=07/spark-summit.pdf

Object storage in more details

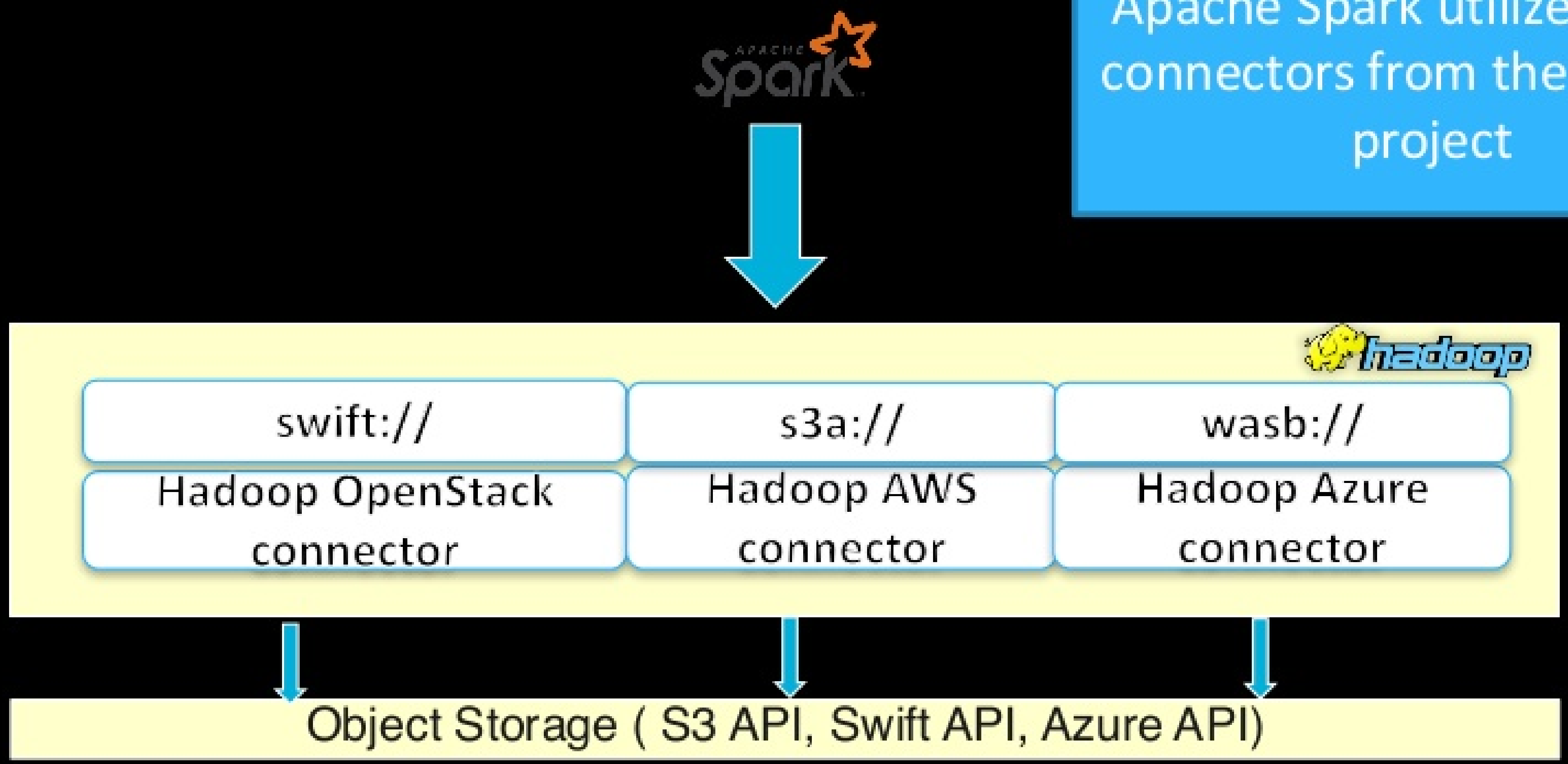
- Capable to store huge amounts of unstructured data in any format
- Resilient store – storage data will not be lost
- Object store designed to operate during failures
- Various security models – storage data is safe protected
- Object stores can be easily accessed for write or read flows
- On premise, cloud based, hybrid, etc.
- Analytic job results can be persisted in the object storage
- Object stores allows easily to share data subsets with others

Choose your object storage



How Spark access object stores?

Apache Spark utilize various connectors from the Hadoop project



Example: persist collection as an object

```
val data = Array(1, 2, 3, 4, 5, 6, 7, 8, 9)
val myData = sc.parallelize(data, 9)
myData.saveAsTextFile(”s3a://mybucket/mydata.txt")
```

	API	GET	HEAD	PUT	DELETE
Hadoop (s3a)	S3	158	361	26	16

561 total requests to the object storage

Behind the numbers

- We observed that the usage of Hadoop connectors with object store is highly inefficient
- Hadoop connectors adapted for file systems and not object stores
- What is wrong? - two major reasons
  - The existing algorithms used by Hadoop to achieve fault tolerance for persisting distributed data sets
  - Cost of supporting FS shell operations and treating object store as a file system

Fault tolerance algorithms in the write flows

- Output committers responsible for persisting data sets generated by MapReduce jobs. Output committers uses temp files and folders for every write operation and then renames them.
- File systems has atomic rename, which perfectly fits into this paradigm.
- Object stores do not support rename natively; use copy and delete instead. This leads to dozens of expensive requests targeted to the object store.

..result/_temporary/0/_temporary/attempt_201702221313_0000_m_000000_0/part-0000

..result/_temporary/0/task_201702221313_0000_m_000000/part-00000

..result/part-00001

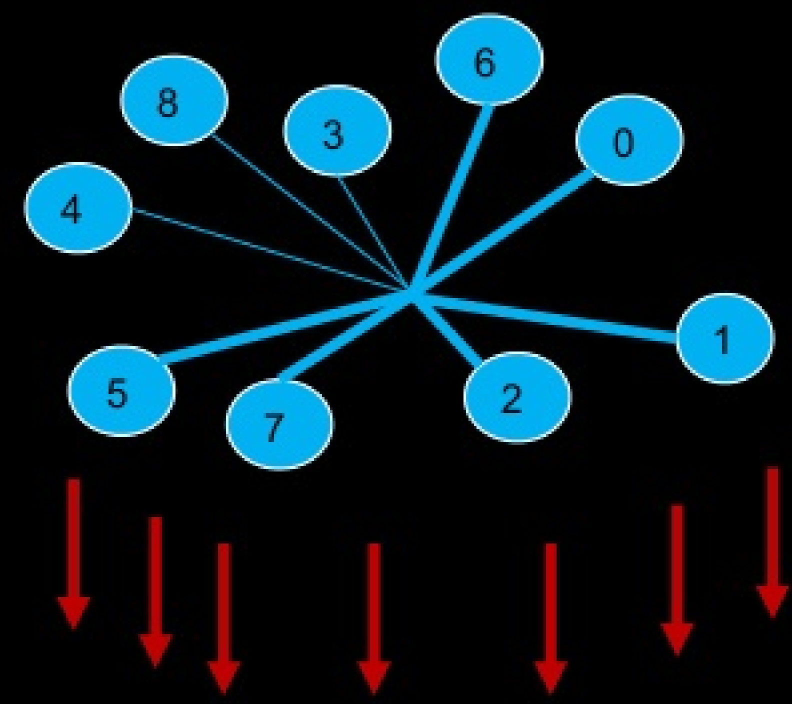
Hadoop FS shell operations

- Hadoop connectors to be 100% compliant with the Hadoop ecosystem must support FS shell operations on files/directories

```
./bin/hadoop fs –mkdirs hdfs://myhdfs/a/b/c/
./bin/hadoop fs –put mydata.txt hdfs://myhdfs/a/b/c/data.txt
```

- Object store vendors provide CLI tools that are preferable over Hadoop shell fs commands
- Analytic flows do not need FS shell operations
- The code to enable FS shell indirectly hurts entire analytic flows in the Hadoop connectors by performing operations that are not inherent to the analytic flows

Why does supporting FS shell affect analytic flows?



Persist distributed data set as an object

/data.txt/_temporary/0/_temporary/attempt_201702221313_0000_m_000000_0/part-0000
/data.txt/_temporary/0/_temporary/attempt_201702221313_0000_m_000001_1/part-0001

……

/dat[illegible]/_temporary/0/_temporary/attempt_201702221313_0000_m_000008_8/p[illegible]0008

An opinionated object store connector for Spark can provide significant gains

Stocator – the next-gen connector

- Advanced object store connector designed for object stores. Doesn't create temp files and folders for write operations and still provides complete fault tolerance coverage including speculative mode.
- Doesn't use Hadoop modules and interacts with object store directly. This makes Stocator superior faster for write flow and generate many less REST calls
- Released under Apache License 2.0
- Implements Hadoop FileSystem interface.
- No need to modify Spark or Hadoop
- No need HDFS, comparing to other optimizations to the Hadoop connectors

https://github.com/SparkTC/stocator

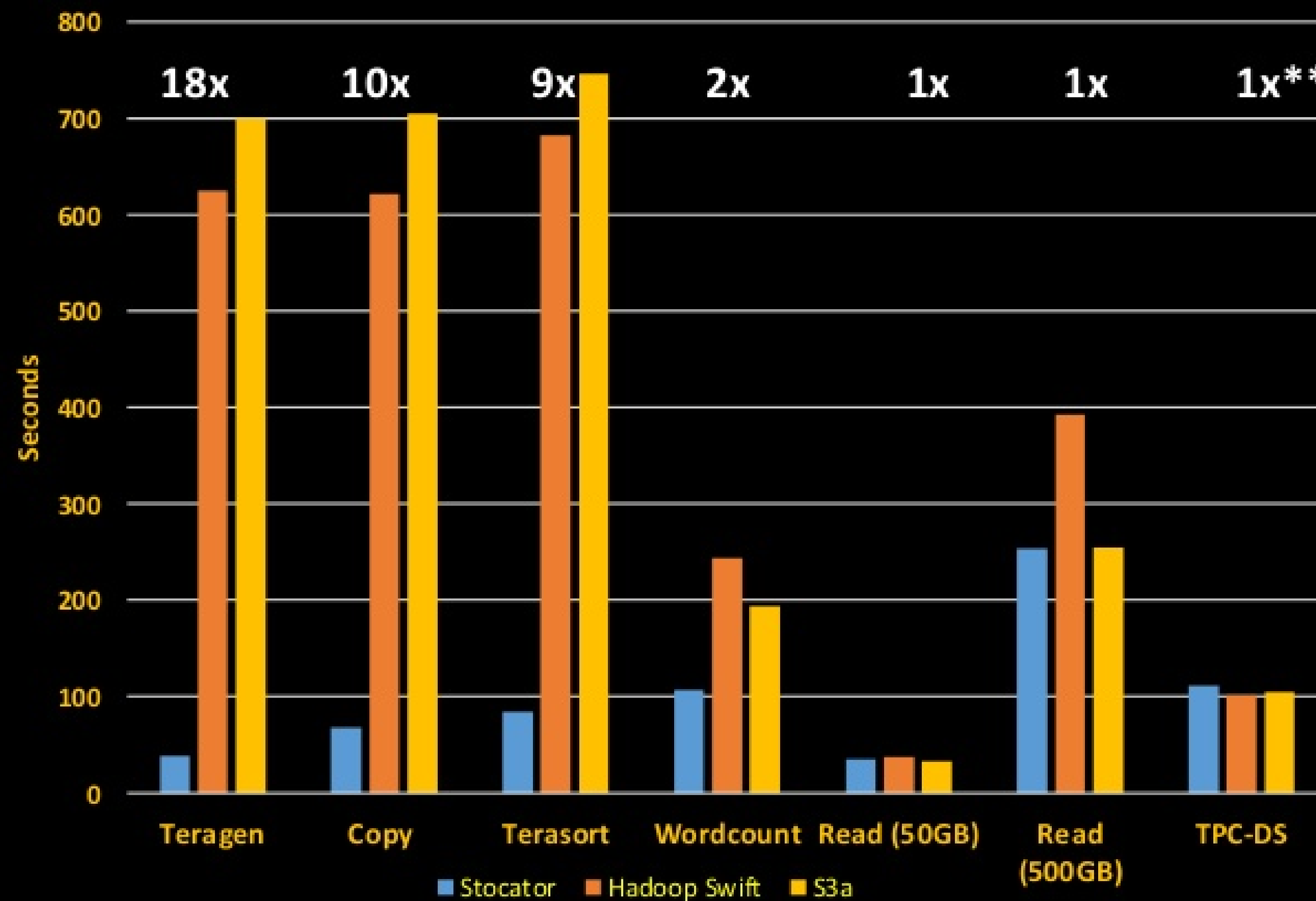
Stocator – the next-gen connector

Stocator adapted for analytic flows

	API	GET	HEAD	PUT	DELETE
Stocator	**S3**	**1**	**2**	**11**	**0**
Hadoop (s3a)	S3	158	361	26	16

Compare performance of Stocator

- Stocator is much faster for write-intensive workloads
- Stocator as good for read-intensive workloads

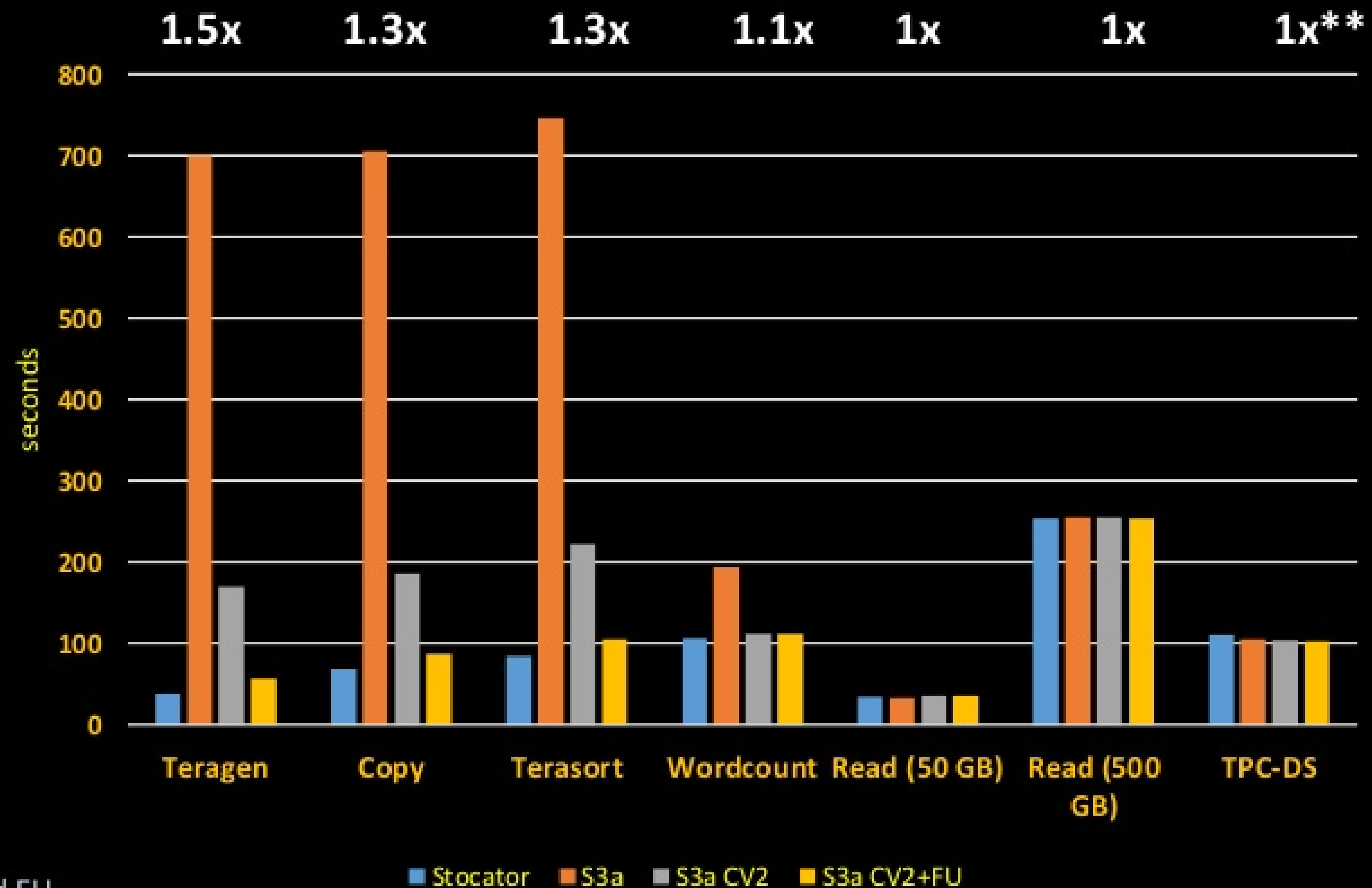


* 40Gbps in accesser tier

** Comparing Stocator to S3a

s3a connector is improving*

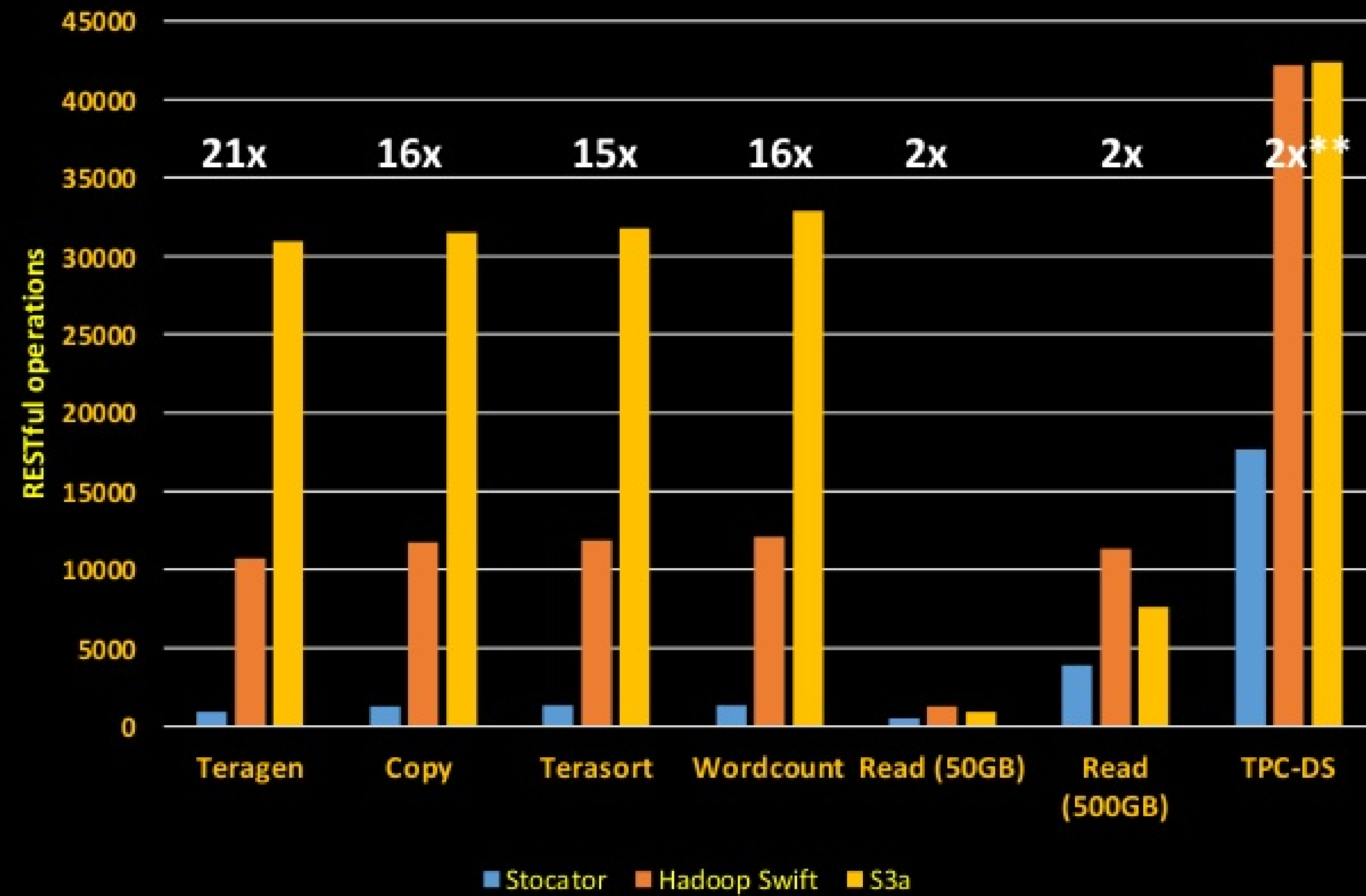
- File Output Committer Algorithm 2 halves number of renames (CV2)
- Fast Upload introduces streaming on output (FU)
- Stocator still faster for write-intensive workloads and as good for read-intensive



** Comparing Stocator to S3a with CV2 and FU

Compare number of REST operations*

- Stocator does many less REST operations
- Less operations means
  - Lower overhead
  - Lower cost



* 40Gbps in accesser tier

** Comparing Stocator to S3a with CV2 and FU

Conclusions and Take-aways

- Spark@SETI is a real-world use case with demanding data and compute workload requirements
- Stocator is a critical component of the solution – enabling high performance data access from Object Store up to the Spark Cluster on IBM Cloud
- Stocator has been contributed to open source under Apache License 2.0

Thank You.

graham.mackintosh@us.ibm.com
gilv@il.ibm.com