# Certains logiciels : big data et analyse des données

Sergey Kirgizov

ESIREM

# Plan

Big data Frameworks ?

Hadoop et Spark

scikit-learn

Mathplotlib

# Big data frameworks ?

# well...


rapidminer
Apache Solr
Dato
elastic
IBM Watson
clarifai
Lucene
Spark
ORACLE NOSQL DATABASE
cassandra
TEZ
CouchDB
APACHE HBASE
NumPy
MADlib
APACHE STORM
HIVE
Map Reduce
mongoDB
SciPy
DATASTAX
influxdata
SciDB
MATLAB
MESOS
hadoop
TACHYON
sas
OOZIE
APACHE DRILL
Apache Ambari
nifi
Couchbase
SPSS
DataTorrent
redislabs
amazon DynamoDB
mllib
plotly
Scala
accumulo
kognitio
mahout
data iku
kibana
HPCC SYSTEMS
MariaDB
TensorFlow
splunk>
sumologic
NUODB
VERTICA
TERADATA
WEKA The University of Waikato
sense
OrientDB
INFINITEGRAPH
Clustrix
Pivotal
dremio
memsql
NLTK
CONTINUUM
DataRobot
neo4j
APACHE GIRAPH
GraphLab
Greenplum
NETEZZA
SAP HANA
Data Flair

# Histoire

- **2003** Google File System (Sanjay Ghemawat, Howard Gobioff, Shun-Tak Leung)

# Histoire

- **2003** Google File System (Sanjay Ghemawat, Howard Gobioff, Shun-Tak Leung)
- **2004** Map Reduce @ Google (Jeffrey Deanand et Sanjay Ghemawat)

# Histoire

- **2003** Google File System (Sanjay Ghemawat, Howard Gobioff, Shun-Tak Leung)
- **2004** Map Reduce @ Google (Jeffrey Deanand et Sanjay Ghemawat)
- **2006** Hadoop @ Yahoo, puis Apache (Doug Cutting)

# Histoire

- **2003** Google File System (Sanjay Ghemawat, Howard Gobioff, Shun-Tak Leung)
- **2004** Map Reduce @ Google (Jeffrey Deanand et Sanjay Ghemawat)
- **2006** Hadoop @ Yahoo, puis Apache (Doug Cutting)
- **2006** Hadoop trie 1.8 TB en 188 nœuds en 47.9 hours

# Histoire

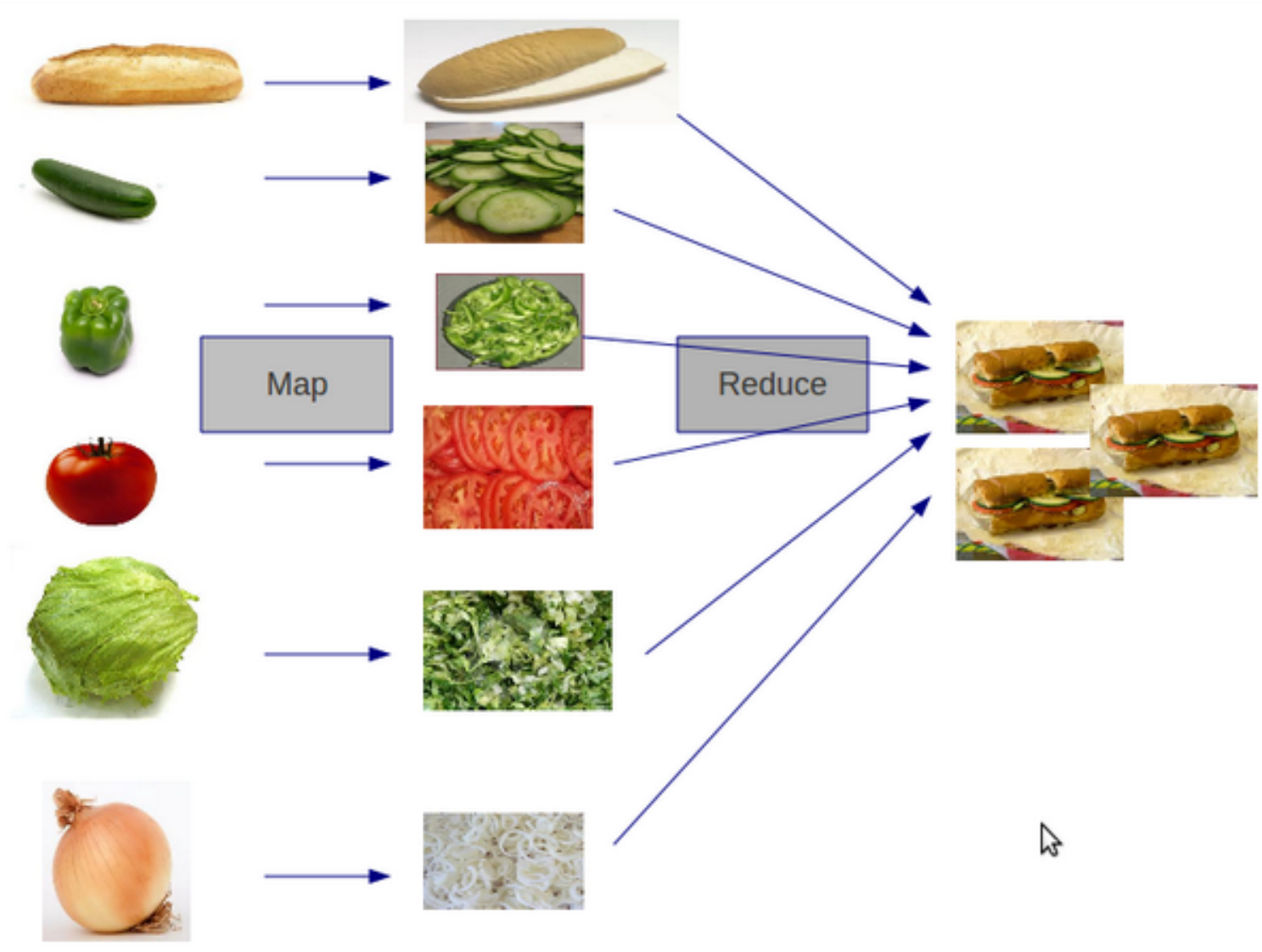
- **2003** Google File System (Sanjay Ghemawat, Howard Gobioff, Shun-Tak Leung)
- **2004** Map Reduce @ Google (Jeffrey Deanand et Sanjay Ghemawat)
- **2006** Hadoop @ Yahoo, puis Apache (Doug Cutting)
- **2006** Hadoop trie 1.8 TB en 188 nœuds en 47.9 hours
- **2008** Hadoop remporte un concours de tri !
  - 910 nœuds
  - 1 téraoctet
  - 209 seconds
  - **tri rapide + tri fusion**

**Inspiration.**

Map Reduce chez Google $\longrightarrow$ Hadoop Map Reduce

GFS $\longrightarrow$ Hadoop Distributed File System (HDFS)

# Map reduce



# Histoire

- **2009** Hadoop trie un pétaoctet !

# Histoire

- **2009** Hadoop trie un pétaoctet !
- **2010** Yahoo 4 000 nœuds, 70 petaoctets

# Histoire

- **2009** Hadoop trie un pétaoctet !
- **2010** Yahoo 4 000 nœuds, 70 petaoctets
- **2010** Facebook 2 300 nœuds, 40 petaoctets

# Histoire

- **2009** Hadoop trie un pétaoctet !
- **2010** Yahoo 4 000 nœuds, 70 petaoctets
- **2010** Facebook 2 300 nœuds, 40 petaoctets
- **2010** Spark paper (Matei Zaharia *et al.*)

# Spark "3X faster, 10X fewer machines!"

| | Hadoop MR Record | Spark Record | Spark 1 PB |
|---|---|---|---|
| Data Size | 102.5 TB | 100 TB | 1000 TB |
| Elapsed Time | 72 mins | 23 mins | 234 mins |
| # Nodes | 2100 | 206 | 190 |
| # Cores | 50400 physical | 6592 virtualized | 6080 virtualized |
| Cluster disk throughput | 3150 GB/s (est.) | 618 GB/s | 570 GB/s |
| Sort Benchmark Daytona Rules | Yes | Yes | No |
| Network | dedicated data center, 10Gbps | virtualized (EC2) 10Gbps network | virtualized (EC2) 10Gbps network |
| **Sort rate** | **1.42 TB/min** | **4.27 TB/min** | **4.27 TB/min** |
| **Sort rate/node** | **0.67 GB/min** | **20.7 GB/min** | **22.5 GB/min** |

`https://databricks.com/blog/2014/11/05/spark-officially-sets-a-new-record-in-large-scale-sorting.html`

# Le concours de tri continue

# Sort Benchmark Home Page

***New:*** The 2019 deadline, September 1, 2019, has passed. All entrants should have already received an email acknowledgement of their entry.

## Background

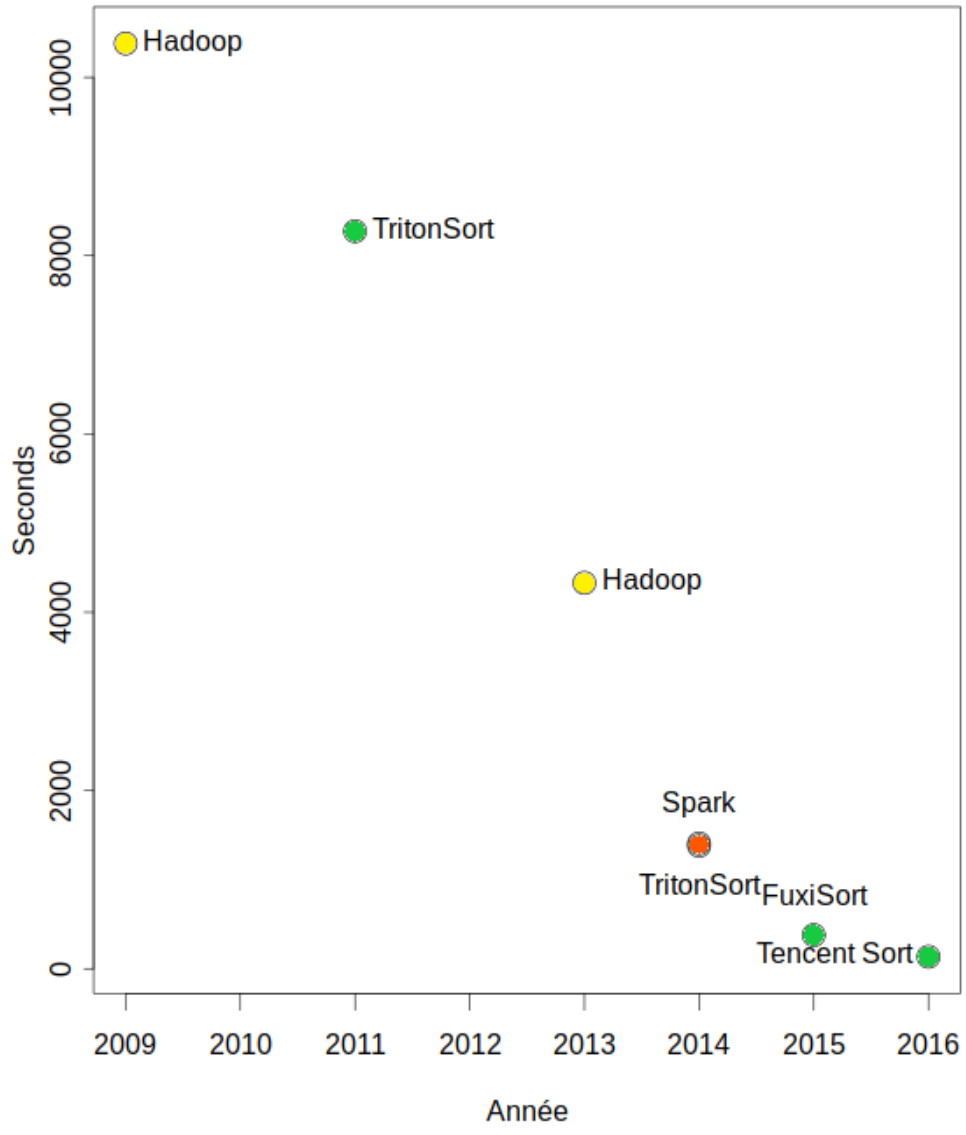
Until 2007, the sort benchmarks were primary defined, sponsored and administered by Jim Gray. Following Jim's disappearance at sea in January 2007, the sort benchmarks have been continued by a committee of past colleagues and sort benchmark winners. The Sort Benchmark committee members include:

- Chris Nyberg of Ordinal Technology Corp
- Mehul Shah of Amazon Web Services
- Naga Govindaraju of Microsoft

## Top Results

| | Daytona | Indy |
|---|---|---|
| Gray | 2016, 44.8 TB/min<br>**Tencent Sort**<br>100 TB in 134 Seconds<br>512 nodes x (2 OpenPOWER 10-core POWER8 2.926 GHz,<br>512 GB memory, 4x Huawei ES3600P V3 1.2TB NVMe SSD,<br>100Gb Mellanox ConnectX4-EN)<br>Jie Jiang, Lixiong Zheng, Junfeng Pu, Xiong Cheng, Chongqing Zhao<br>Tencent Corporation<br>Mark R. Nutter, Jeremy D. Schaub | 2016, 60.7 TB/min<br>**Tencent Sort**<br>100 TB in 98.8 Seconds<br>512 nodes x (2 OpenPOWER 10-core POWER8 2.926 GHz,<br>512 GB memory, 4x Huawei ES3600P V3 1.2TB NVMe SSD,<br>100Gb Mellanox ConnectX4-EN)<br>Jie Jiang, Lixiong Zheng, Junfeng Pu, Xiong Cheng, Chongqing Zhao<br>Tencent Corporation<br>Mark R. Nutter, Jeremy D. Schaub |

https://sortbenchmark.org/




Pour trier 100Tb.

2004

# MapReduce: Simplified Data Processing on Large Clusters

Jeffrey Dean and Sanjay Ghemawat

jeff@google.com, sanjay@google.com

*Google, Inc.*

## Abstract

MapReduce is a programming model and an associated implementation for processing and generating large data sets. Users specify a *map* function that processes a key/value pair to generate a set of intermediate key/value pairs, and a *reduce* function that merges all intermediate values associated with the same intermediate key. Many real world tasks are expressible in this model, as shown in the paper.

Programs written in this functional style are automatically parallelized and executed on a large cluster of commodity machines. The run-time system takes care of the details of partitioning the input data, scheduling the program's execution across a set of machines, handling machine failures, and managing the required inter-machine communication. This allows programmers without any experience with parallel and distributed systems to easily utilize the resources of a large distributed system.

Our implementation of MapReduce runs on a large cluster of commodity machines and is highly scalable: a typical MapReduce computation processes many terabytes of data on thousands of machines. Programmers find the system easy to use: hundreds of MapReduce pro-

given day, etc. Most such computations are conceptually straightforward. However, the input data is usually large and the computations have to be distributed across hundreds or thousands of machines in order to finish in a reasonable amount of time. The issues of how to parallelize the computation, distribute the data, and handle failures conspire to obscure the original simple computation with large amounts of complex code to deal with these issues.

As a reaction to this complexity, we designed a new abstraction that allows us to express the simple computations we were trying to perform but hides the messy details of parallelization, fault-tolerance, data distribution and load balancing in a library. Our abstraction is inspired by the *map* and *reduce* primitives present in Lisp and many other functional languages. We realized that most of our computations involved applying a *map* operation to each logical "record" in our input in order to compute a set of intermediate key/value pairs, and then applying a *reduce* operation to all the values that shared the same key, in order to combine the derived data appropriately. Our use of a functional model with user-specified map and reduce operations allows us to parallelize large computations easily and to use re-execution

2010

# Spark: Cluster Computing with Working Sets

Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, Ion Stoica
*University of California, Berkeley*

## Abstract

MapReduce and its variants have been highly successful in implementing large-scale data-intensive applications on commodity clusters. However, most of these systems are built around an acyclic data flow model that is not suitable for other popular applications. This paper focuses on one such class of applications: those that reuse a working set of data across multiple parallel operations. This includes many iterative machine learning algorithms, as well as interactive data analysis tools. We propose a new framework called Spark that supports these applications while retaining the scalability and fault tolerance of MapReduce. To achieve these goals, Spark introduces an abstraction called resilient distributed datasets (RDDs). An RDD is a read-only collection of objects partitioned across a set of machines that can be rebuilt if a partition is lost. Spark can outperform Hadoop by 10x in iterative machine learning jobs, and can be used to interactively query a 39 GB dataset with sub-second response time.

## 1 Introduction

A new model of cluster computing has become widely popular, in which data-parallel computations are executed on clusters of unreliable machines by systems that automatically provide locality-aware scheduling, fault toler-

MapReduce/Dryad job, each job must reload the data from disk, incurring a significant performance penalty.

- **Interactive analytics**: Hadoop is often used to run ad-hoc exploratory queries on large datasets, through SQL interfaces such as Pig [21] and Hive [1]. Ideally, a user would be able to load a dataset of interest into memory across a number of machines and query it repeatedly. However, with Hadoop, each query incurs significant latency (tens of seconds) because it runs as a separate MapReduce job and reads data from disk.

This paper presents a new cluster computing framework called Spark, which supports applications with working sets while providing similar scalability and fault tolerance properties to MapReduce.

The main abstraction in Spark is that of a *resilient distributed dataset* (RDD), which represents a read-only collection of objects partitioned across a set of machines that can be rebuilt if a partition is lost. Users can explicitly cache an RDD in memory across machines and reuse it in multiple MapReduce-like *parallel operations*. RDDs achieve fault tolerance through a notion of *lineage*: if a partition of an RDD is lost, the RDD has enough information about how it was derived from other RDDs to be able to rebuild just that partition. Although RDDs are not a general shared memory abstraction, they represent

# Hadoop

Le framework Hadoop de base se compose des modules suivants :

- **Hadoop Common** : les bibliothèques et les utilitaires nécessaires pour les autres modules
- **Hadoop Distributed File System (HDFS)** : le système de fichiers
- **Hadoop YARN (Yet Another Resource Negotiator)** : un gestionnaire de cluster
- **Hadoop MapReduce** : une implémentation du modèle de programmation MapReduce pour le traitement de données à grande échelle

# Écosystème d'Hadoop

Apache Pig, Apache Hive, Apache HBase, Apache Phoenix, **Apache Spark**, Apache ZooKeeper, Apache Impala, Apache Flume, Apache Sqoop, Apache Oozie, et Apache Storm...

# Apache Spark


- Développé à l'université de Californie à Berkeley par AMPLab.
- Son langage natif est Scala, sinon on peut faire du Java, python, R

*Spark exécute la totalité des opérations d'analyse de données en mémoire et en temps réel. Il s'appuie sur des disques seulement lorsque sa mémoire n'est plus suffisante. À l'inverse, avec Hadoop classique les données sont écrites sur le disque après chacune des opérations. Ce travail en mémoire permet de réduire les temps de latence entre les traitements, ce qui explique une telle rapidité.*

– `https://fr.wikipedia.org/wiki/Apache_Spark`

*En cas de panne ou de défaillance du système : les objets de données sont stockés dans ce que l'on appelle des ensembles de données distribués résilients (RDD : resilient distributed datasets) répartis sur le cluster de données permettant la récupération complète de données.*

– `https://fr.wikipedia.org/wiki/Apache_Spark`

*Spark ne dispose pas de système de gestion de fichier qui lui est propre. Il est nécessaire de lui en fournir un, par exemple Hadoop Distributed File System, Informix, Cassandra, OpenStack Swift ou Amazon S3.*

– `https://fr.wikipedia.org/wiki/Apache_Spark`

# Matei Zaharia

"Matei Zaharia este un informatician româno-canadian specializat în big data, sisteme distribuite și cloud computing. El este co-fondator și CTO al Databricks și profesor asistent de informatică la Universitatea Stanford."

— https://ro.wikipedia.org/wiki/Matei_Zaharia

- Créateur d'Apache Spark.
- Sa thèse de doctorat sur Spark : https://www2.eecs.berkeley.edu/Pubs/TechRpts/2014/EECS-2014-12.pdf
- Son site : https://cs.stanford.edu/~matei/
- **Une partie de ces cours est dispo :** https://cs.stanford.edu/~matei/#teaching

# Matei Zaharia


Matei Zaharia, creator of Apache Spark | True Technologist Ep 2

`https://www.youtube.com/watch?v=2lypTlbjqHE`

# Analyse et Visualisation

# scikit-learn

# scikit-learn

Une bibliothèque libre Python d'apprentissage statistique.


Elle est créée en 2007 par David Cournapeau et développée par de nombreux contributeurs.

`https://scikit-learn.org/`

Code source :
`https://github.com/scikit-learn/scikit-learn`

Licence BSD 3-clauses

# scikit-learn

*This project was started in 2007 as a Google Summer of Code project by David Cournapeau. Later that year, Matthieu Brucher started work on this project as part of his thesis.*

*In 2010 Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort and Vincent Michel of INRIA took leadership of the project and made the first public release, February the 1st 2010. Since then, several releases have appeared following a ~3-month cycle, and a thriving international community has been leading the development.*

– `https://scikit-learn.org/stable/about.html`

David Cournapeau

`https://github.com/cournape`

2004, Master de Telecom Paristech, Paris

2009, PhD en Informatique, Kyoto University, 2009

...

Depuis 2022, Engineering director, Mercari, Inc.

# David Cournapeau


How to meaningfully contribute to Python without being very good at programming - David Cournapeau

`https://www.youtube.com/watch?v=ajW0JaxLK44`

# scikit-learn

- Classification (identifying which category an object belongs to)
- Regression (predicting a continuous-valued attribute)
- Clustering (automatic grouping of similar objects into sets)
- Dimensionality reduction (reducing the number of variables to consider)
- Model selection (comparing, validating and choosing parameters and models, ...)
- Preprocessing (feature extraction, normalization, ...)

# Read the paper !

Fabian Pedregosa *et al.*
Journal of Machine Learning Research 12 (2011)

"Scikit-learn : Machine Learning in Python"

`https://www.jmlr.org/papers/volume12/pedregosa11a/pedregosa11a.pdf`

# Big data and scikit-learn

https://scikit-learn.org/stable/computing/scaling_strategies.html?highlight=bigger+data

https://speakerdeck.com/datasciencelondon/parallel-and-large-scale-machine-learning-with-scikit-learn

Scikit-Learn + Spark

https://datascience-enthusiast.com/Python/sklearn_spark.html

https://pypi.org/project/spark-sklearn/

# Mathplotlib

Une bibliothèque de visualisation pour Python.


`https://matplotlib.org/`
Créateur : John D. Hunter (1968 - 2012)
Il était un neurobiologiste américain.
Mathplotlib est développé par Michael Droettboom *et al.*

Code source : `https://github.com/matplotlib/matplotlib`

Une licence de style BSD (Mathplotlib Licence)

# John D. Hunter

Master de Princeton University

2004, doctorat en neurobiologie d'University of Chicago

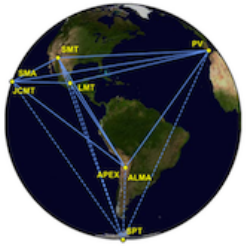
# Mathplotlib

*Matplotlib was originally conceived to visualize electrocorticography (ECoG) data of epilepsy patients during post-doctoral research in neurobiology.*

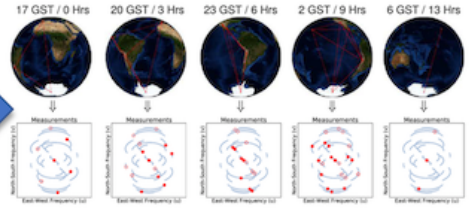
*[...]*

*Matplotlib was used for data visualization during landing of the Phoenix spacecraft in 2008 as well as for the creation of the first image of a black hole.*

– `https://en.wikipedia.org/wiki/John_D._Hunter`




Event Horizon Telescope




Petabytes of data (measurements)

Data correlation, calibration, reduction pipelines

Scientific, Astrophysics & Computing Algorithms

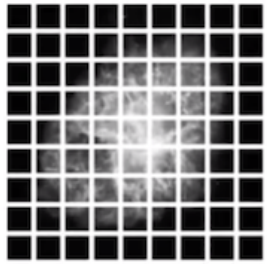


Image analysis

https://github.com/achael/eht-imaging/

Imaging, analysis and simulation software for radio interferometry

M87 – the first image of a black hole

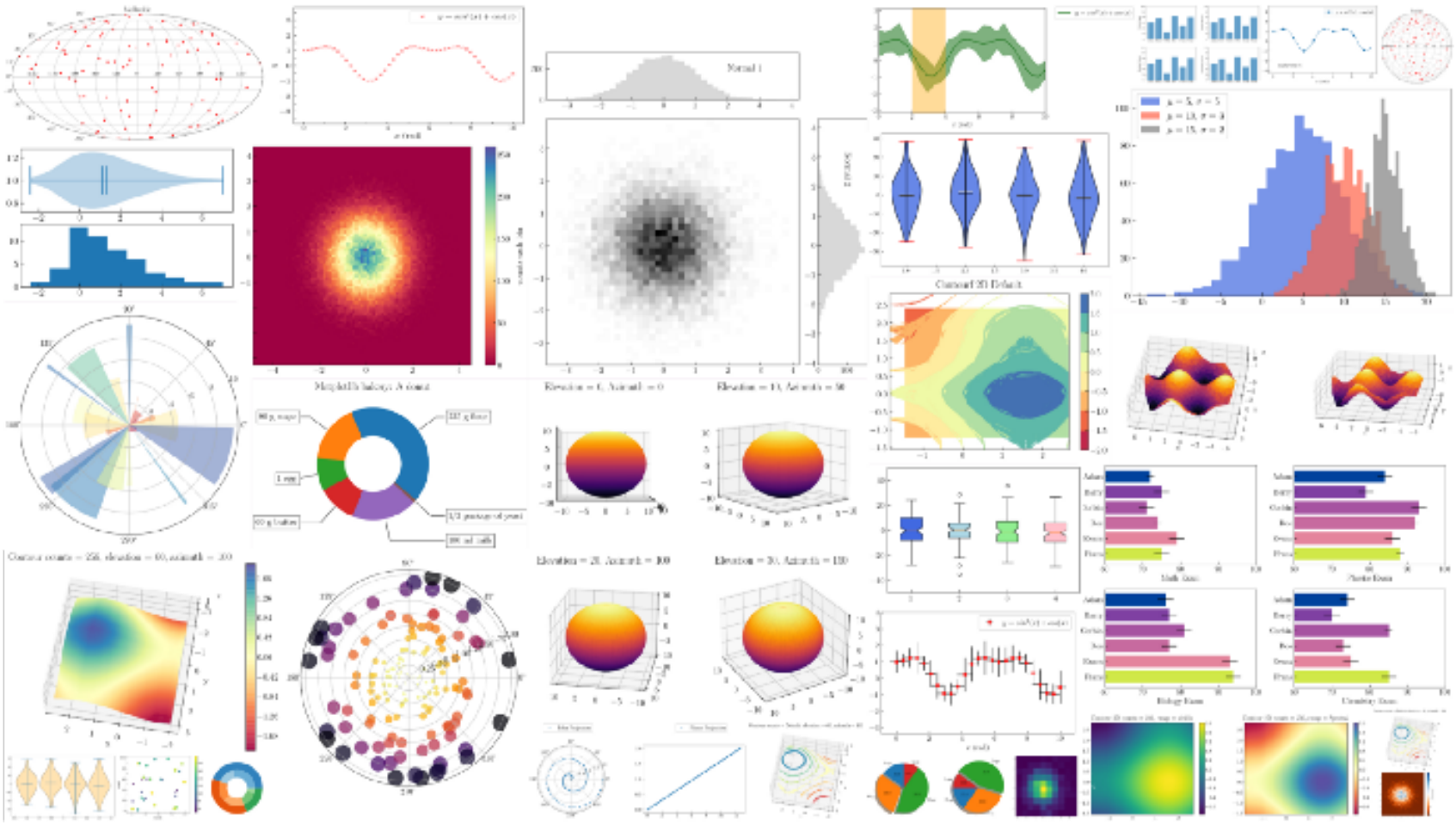
*Software:* DiFX (Deller et al. 2011), CALC, PolConvert (Martí-Vidal et al. 2016), HOPS (Whitney et al. 2004), CASA (McMullin et al. 2007), AIPS (Greisen 2003), ParselTongue (Kettenis et al. 2006), GNU Parallel (Tange 2011), GILDAS, eht-imaging (Chael et al. 2016, 2018), Numpy (van der Walt et al. 2011), Scipy (Jones et al. 2001), Pandas (McKinney 2010), Astropy (The Astropy Collaboration et al. 2013, 2018), Jupyter (Kluyver et al. 2016), Matplotlib (Hunter 2007).

**NumPy was one of the software used!**

"First M87 Event Horizon Telescope Results. III. Data Processing and Calibration"
The Astrophysical Journal Letters
The Event Horizon Telescope Collaboration
`https://iopscience.iop.org/article/10.3847/2041-8213/ab0c57`
`https://numpy.org/case-studies/blackhole-image/`

# John D. Hunter


`https://www.youtube.com/watch?v=Z4hIrCoCGgo`



https://towardsdatascience.com/visualizations-with-matplotlib-part-1-c9651008b6b8

# Questions ?