High Performance Analytics Tools/Frameworks in Python

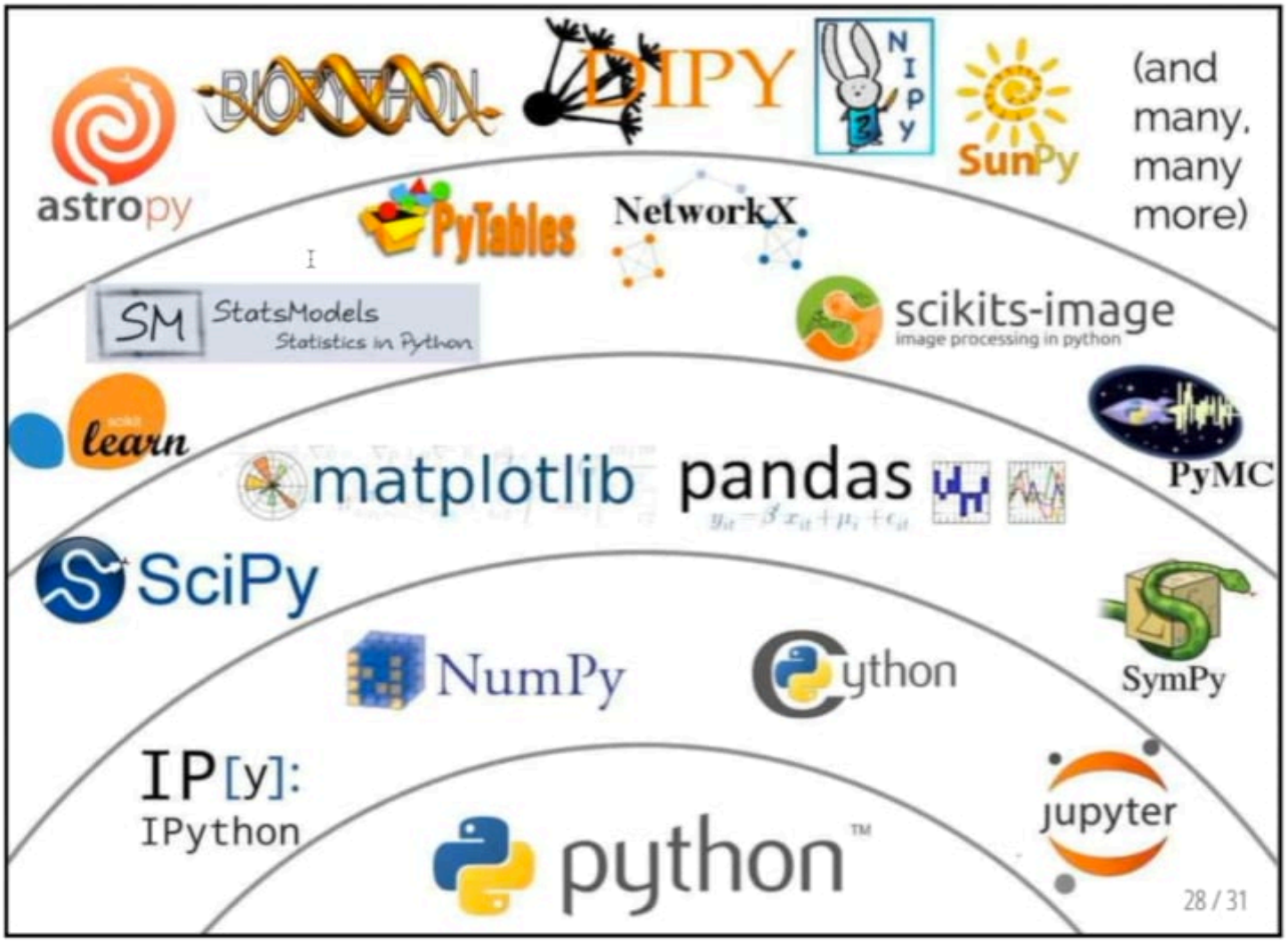
Gopinath Jaganmohan

Co-Founder and CTO

ConverSight.ai

Hire Athena your Analytics-driven, Voice-directed Business Companion

Python Toolbox

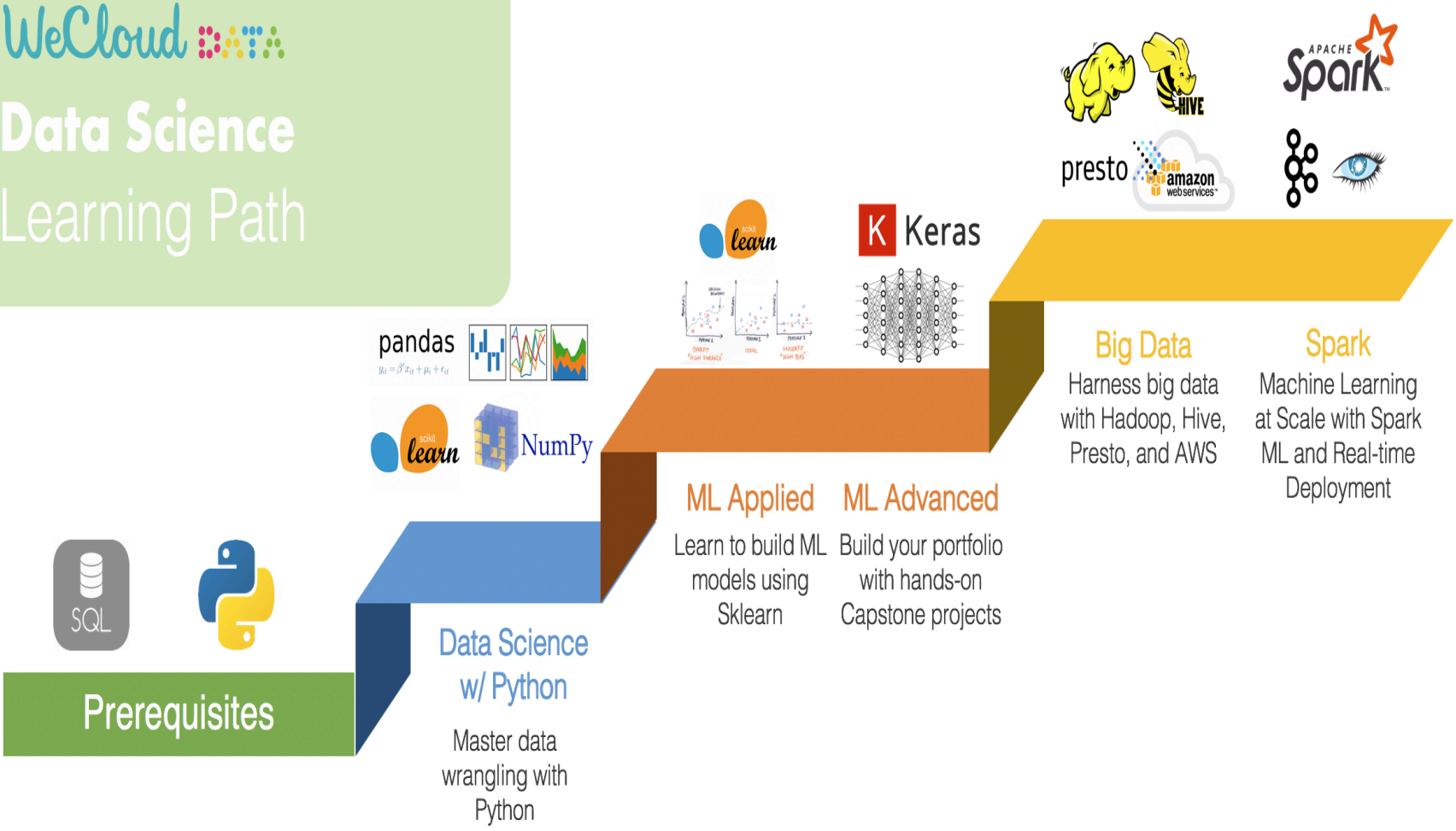


1

1 Scientific Python by Eueung Mulyana

WeCloud DATA

Data Science Learning Path



High Performance Analytics - Data

Data Storage & sharing

Analysis @Source

CPU cache Friendly for Numeric Data

Zero Copy and Memory Efficient

High Performance Analytics - Compute

Efficient use of multi Core/GPU

Efficient use of CPU vector processing

Easy Write algo's

But… Efficiently complied machine code

High Performance Analytics – Distributed Deployment

Data Partitioning

Node and GPU distribution

Scale Horizontally

Language Interoperability

High Performance Analytics - Data

Apache Arrow

Apache Arrow

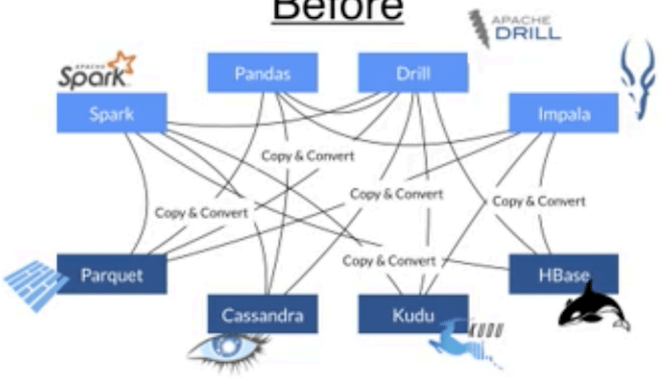
- OSS Community initiative conceived in 2015
- Intersection of big data, database systems, and data science tools
- Key idea: Language agnostic open standards to accelerate in-memory computing
- **https://github.com/apache/arrow**

Apache Arrow At Dataengconf Barcelona 2018

Apache Arrow

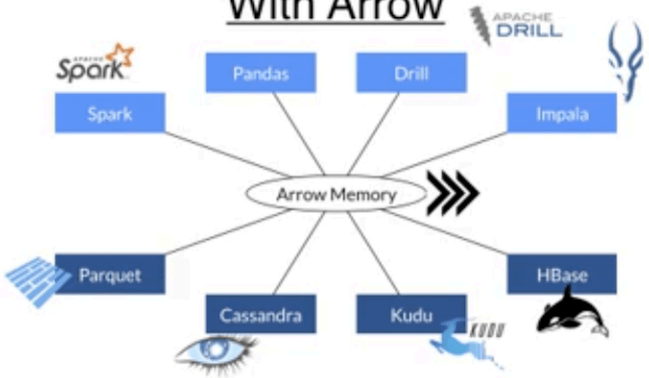
High Performance Sharing & Interchange

Before



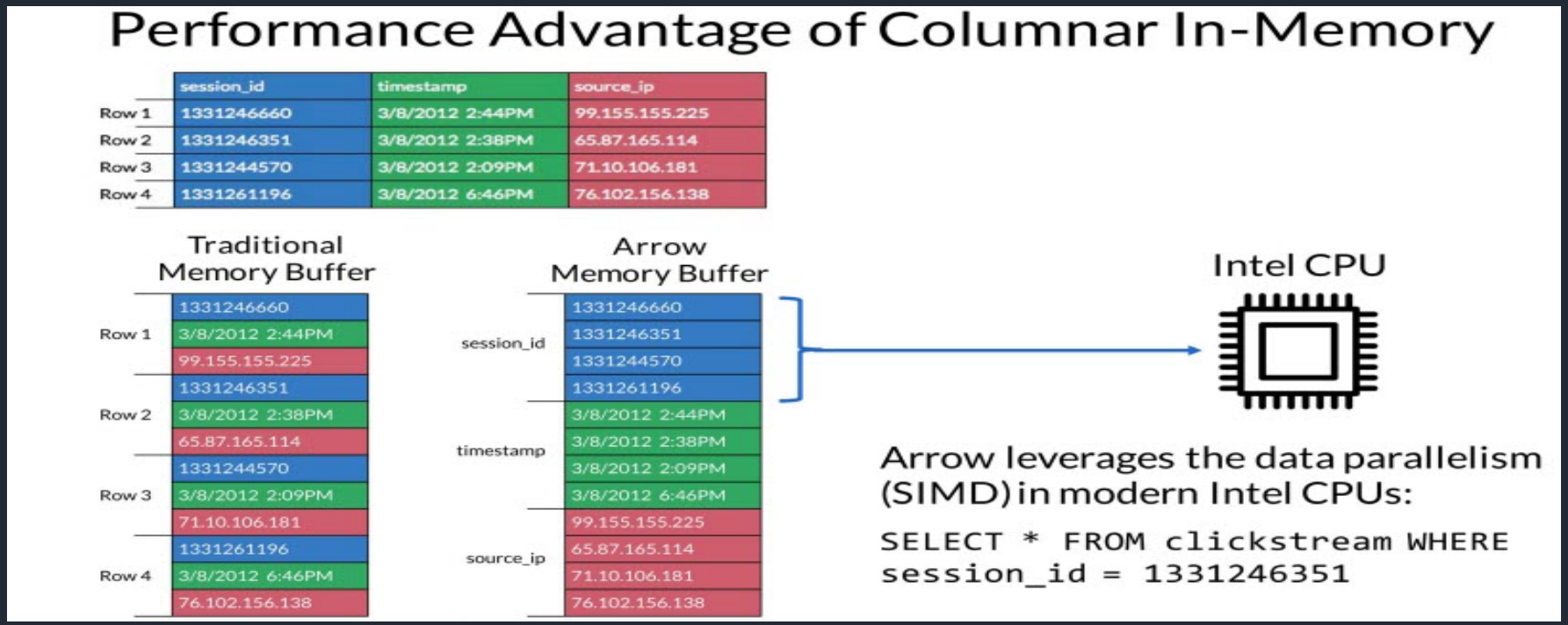
- Each system has its own internal memory format
- 70-80% CPU wasted on serialization and deserialization
- Functionality duplication and unnecessary conversions

With Arrow



- All systems utilize the same memory format
- No overhead for cross-system communication
- Projects can share functionality (eg: Parquet-to-Arrow reader)

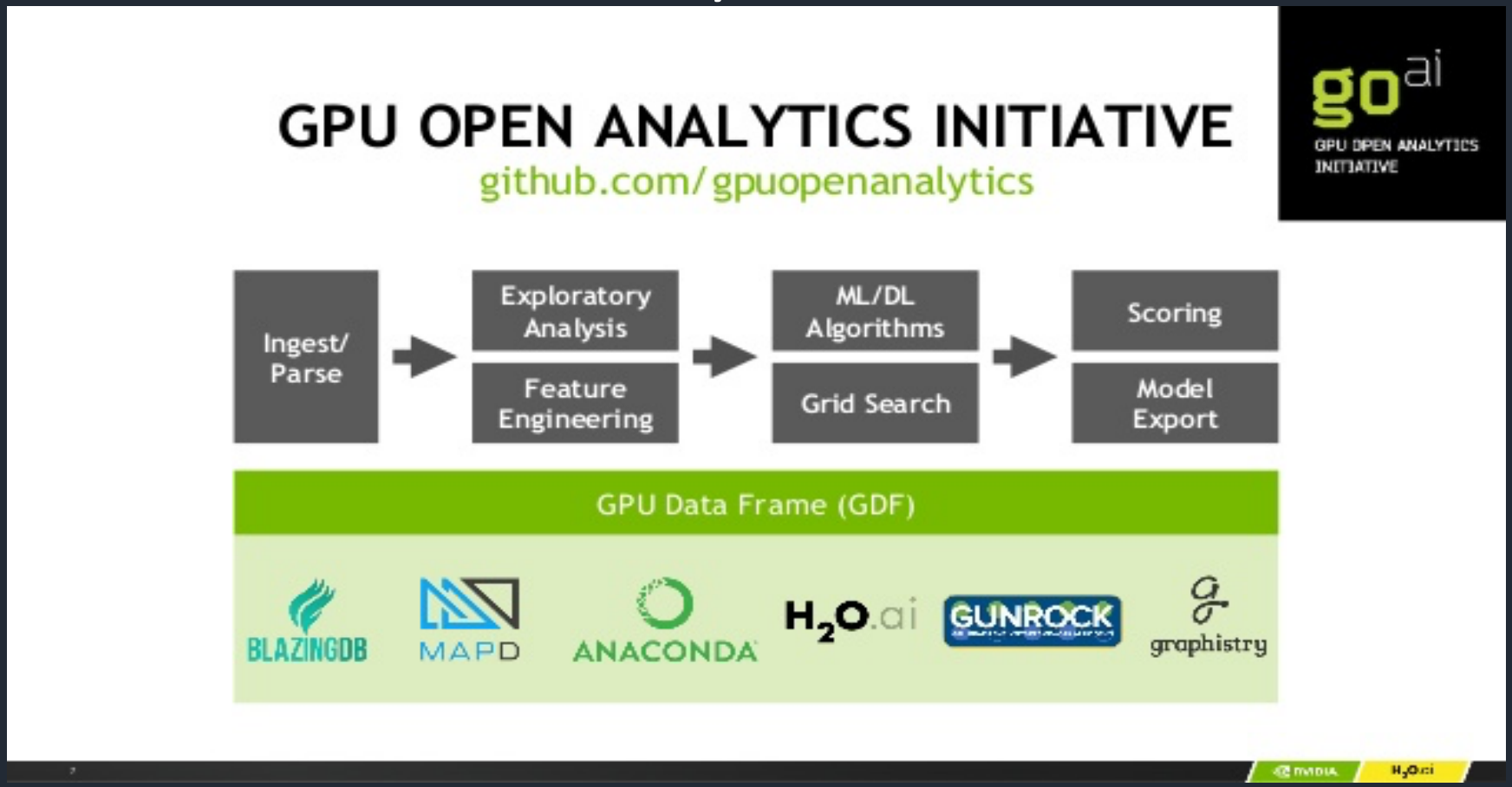
Apache Arrow



Sub Projects

- Gandiva LLVM based Efficient expression evaluation
- Arrow Flight gRPC
- DataFusion

Arrow in the GPU – Nvidia Rapids End to End Analytics in GPU



High Performance Analytics - Computing

Numba makes Python code fast

Numba is an open source JIT compiler that translates a subset of Python and NumPy code into fast machine code.

Learn More | Try Numba »

Accelerate Python Functions

Numba translates Python functions to optimized machine code at runtime using the industry-standard LLVM compiler library. Numba-compiled numerical algorithms in Python can approach the speeds of C or FORTRAN.

You don't need to replace the Python interpreter, run a separate compilation step, or even have a C/C++ compiler installed. Just apply one of the Numba decorators to your Python function, and Numba does the rest.

Learn More » | Try Now »

```
from numba import jit
import random

@jit(nopython=True)
def monte_carlo_pi(nsamples):
    acc = 0
    for i in range(nsamples):
        x = random.random()
        y = random.random()
        if (x ** 2 + y ** 2) < 1.0:
            acc += 1
    return 4.0 * acc / nsamples
```

```
import numpy, numba

@numba.jit
def run_numba(height, width, maxiterations=20):
    y, x = numpy.ogrid[-1:0:height*1j, -1.5:0:width*1j]
    c = x + y*1j
    fractal = numpy.full(c.shape, maxiterations, dtype=numpy.int32)
    for h in range(height):
        for w in range(width):                  # for each pixel (h, w)...
            z = c[h, w]
            for i in range(maxiterations):      # iterate at most 20 times
                z = z**2 + c[h, w]              # applying z → z² + c
                if abs(z) > 2:                  # if it diverges (|z| > 2)
                    fractal[h, w] = i           # color with iteration number
                    break                       # we're done; go away
    return fractal

fractal = run_numba(6400, 9600)                 # runs 50× faster than plain
```

Numba Features

JIT Compiled

CFunc

CUDA Support

Automatic Parallelization

Cython

Example: Fibonacci Numbers

Python: `fib.py`

```python
def fib(n):
    a = 0
    b = 1
    for i in range(n):
        a, b = a + b, a
    return a
```

Cython: `cyfib.pyx`

```python
def fib(int n):
    cdef int i
    cdef double a = 0.0
    cdef double b = 1.0
    for i in range(n):
        a, b = a + b, a
    return a
```

March 2016 NoVa Python Meetup

7

Speed Comparison

Method	Configuration	Speedup	Cores
Plain Python	for-loopy	1×	1
Numba	for-loopy	50×	1
Numba-parallel	for-loopy	165×	all (12)
Numpy	columnar	15×	1
CuPy	columnar	77×	GPU
Dask	columnar	26×	all (12)
Numba-CUDA	CUDA details	800×	GPU
pybind11 `-O3`	for-loopy C++	34×	1
pybind11 `-ffast-math`	for-loopy C++	90×	1
Cython	dual language	3.7×	1

(Sorted by my ease-of-use judgement.)

High Performance Analytics –Distributed

DASK

Dask natively scales Python

Dask provides advanced parallelism for analytics, enabling performance at scale for the tools you love

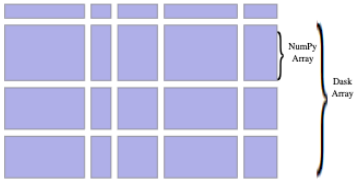
Learn More | Try Now »

Integrates with existing projects

BUILT WITH THE BROADER COMMUNITY

Dask is open source and freely available. It is developed in coordination with other community projects like Numpy, Pandas, and Scikit-Learn.

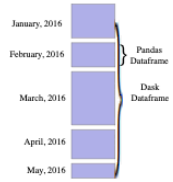
Numpy



Dask arrays scale Numpy workflows, enabling multi-dimensional data analysis in earth science, satellite imagery, genomics, biomedical applications, and machine learning algorithms.

Learn More » | Try Now »

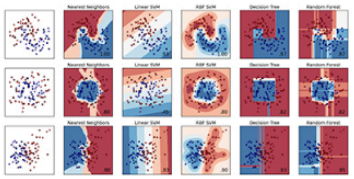
Pandas



Dask dataframes scale Pandas workflows, enabling applications in time series, business intelligence, and general data munging on big data.

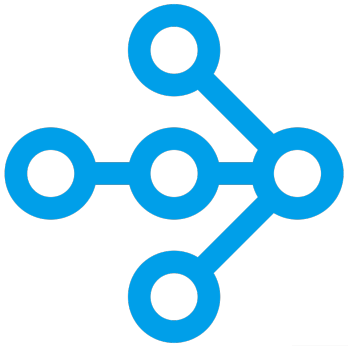
Learn More » | Try Now »

Scikit-Learn



Dask-ML scales machine learning APIs like Scikit-Learn and XGBoost to enable scalable training and prediction on large models and large datasets.

Learn More » | Try Now »



RAY

Example Use

Basic Python

```
# Execute f serially.

def f():
    time.sleep(1)
    return 1


results = [f() for i in range(4)]
```

Distributed with Ray

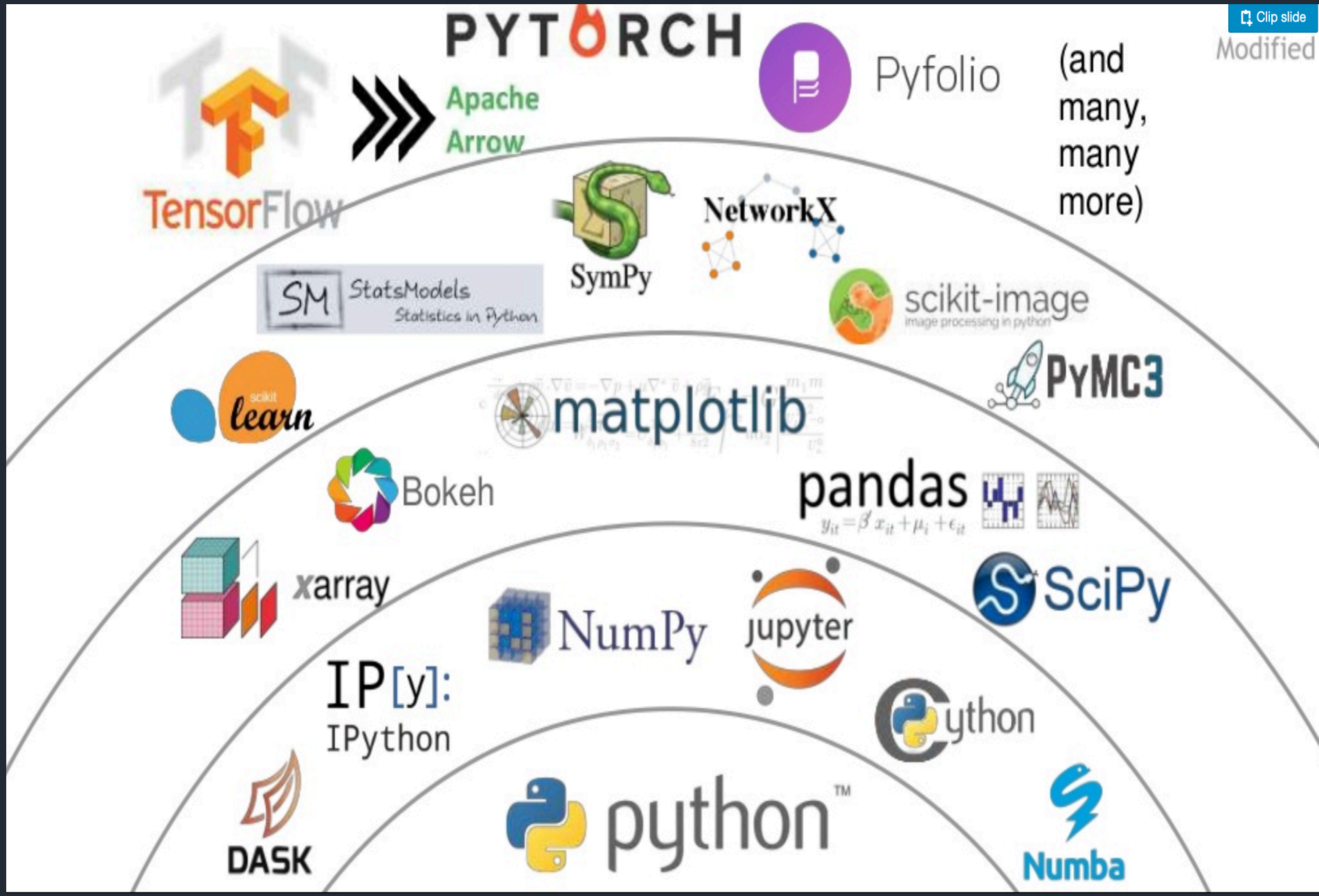
```
# Execute f in parallel.

@ray.remote
def f():
    time.sleep(1)
    return 1


ray.init()
results = ray.get([f.remote() for i in range(4)])
```

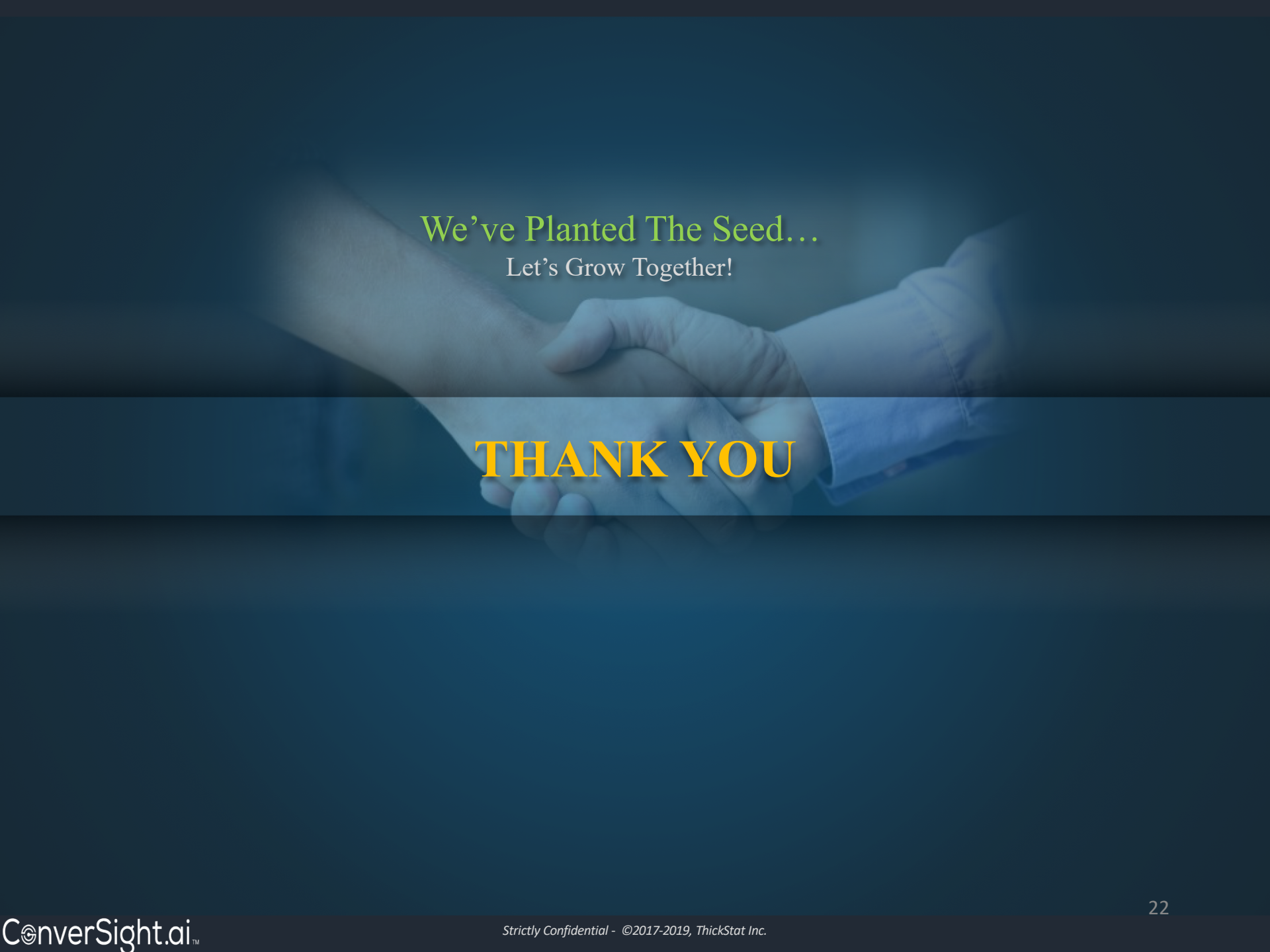
To launch a Ray cluster, either privately, on AWS, or on GCP, follow these instructions.

View the codebase on GitHub.

Ray comes with libraries that accelerate deep learning and reinforcement learning development:

- Tune: Scalable Hyperparameter Search
- RLlib: Scalable Reinforcement Learning
- Distributed Training





ConverSight.ai™