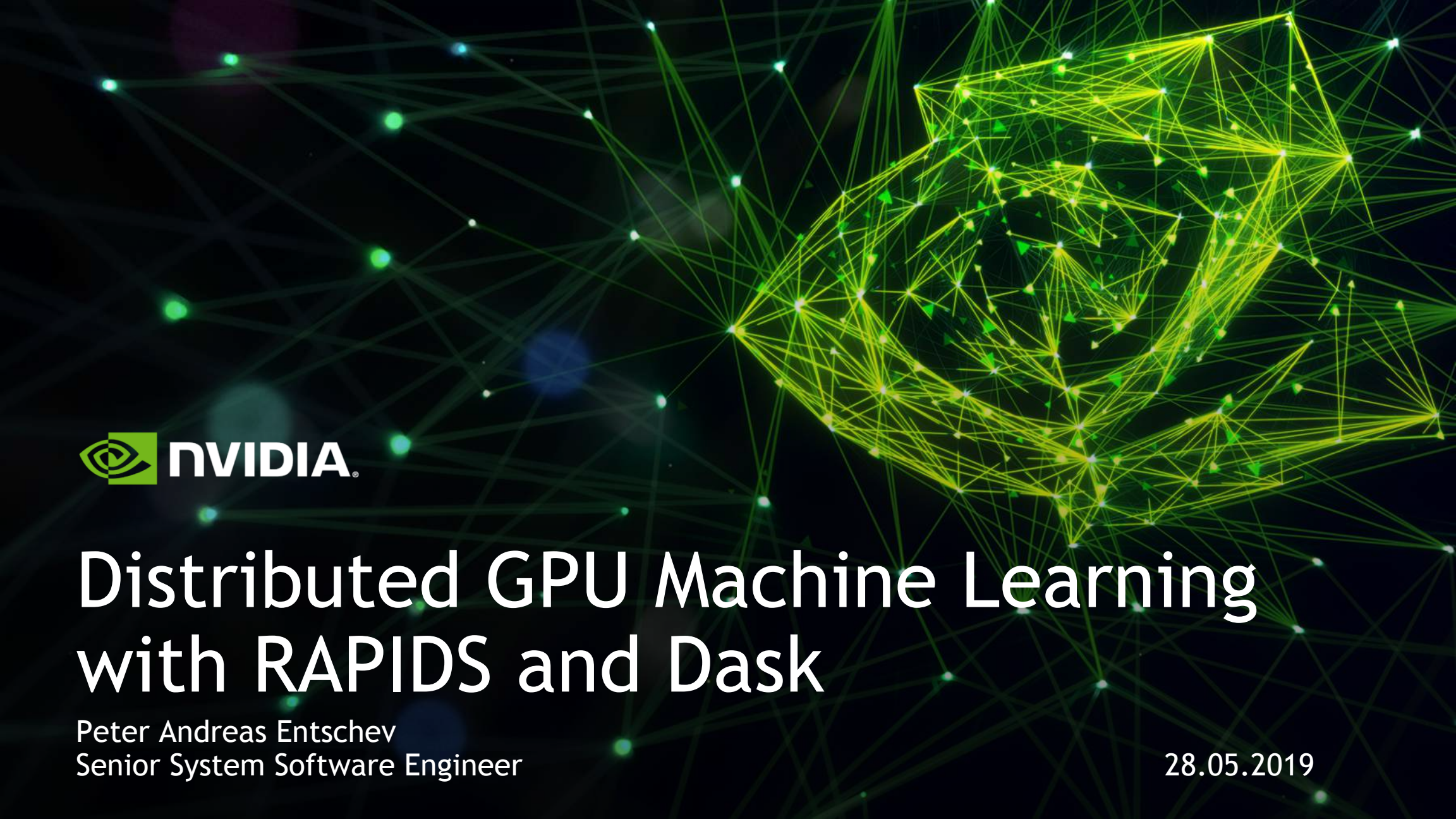


# Distributed GPU Machine Learning with RAPIDS and Dask

Peter Andreas Entschev
Senior System Software Engineer

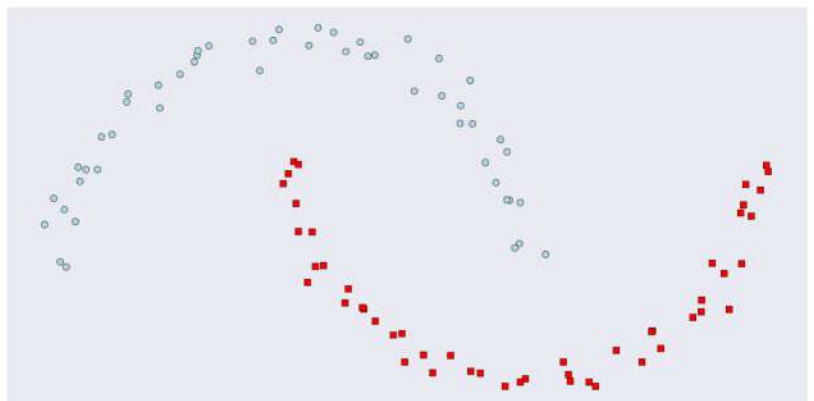
28.05.2019

# Clustering

## Code example

```
from sklearn.datasets import make_moons
import pandas

X, y = make_moons(n_samples=int(1e2),
                  noise=0.05, random_state=0)

X = pandas.DataFrame({'fea%d'%i: X[:, i]
                      for i in range(X.shape[1])})
```

```
from sklearn.cluster import DBSCAN
dbscan = DBSCAN(eps = 0.3, min_samples = 5)

dbscan.fit(X)

y_hat = dbscan.predict(X)
```

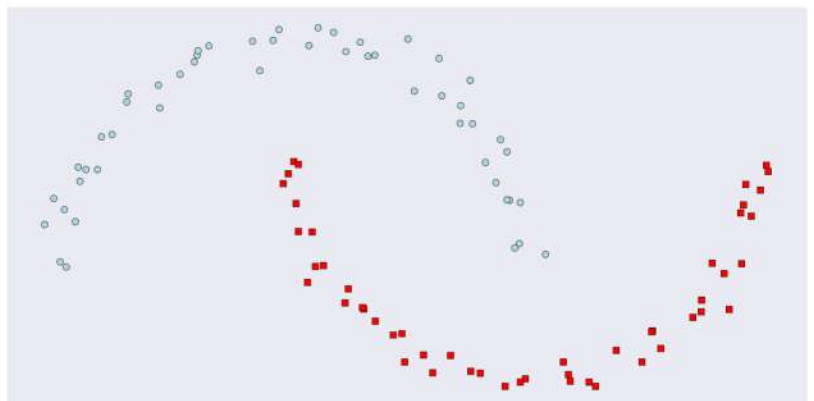


# GPU-Accelerated Clustering

## Code example

```
from sklearn.datasets import make_moons
import cudf

X, y = make_moons(n_samples=int(1e2),
                  noise=0.05, random_state=0)

X =   cudf.DataFrame({'fea%d'%i: X[:, i]
                      for i in range(X.shape[1])})
```

```
from cuml             import DBSCAN
dbscan = DBSCAN(eps = 0.3, min_samples = 5)

dbscan.fit(X)

y_hat = dbscan.predict(X)
```



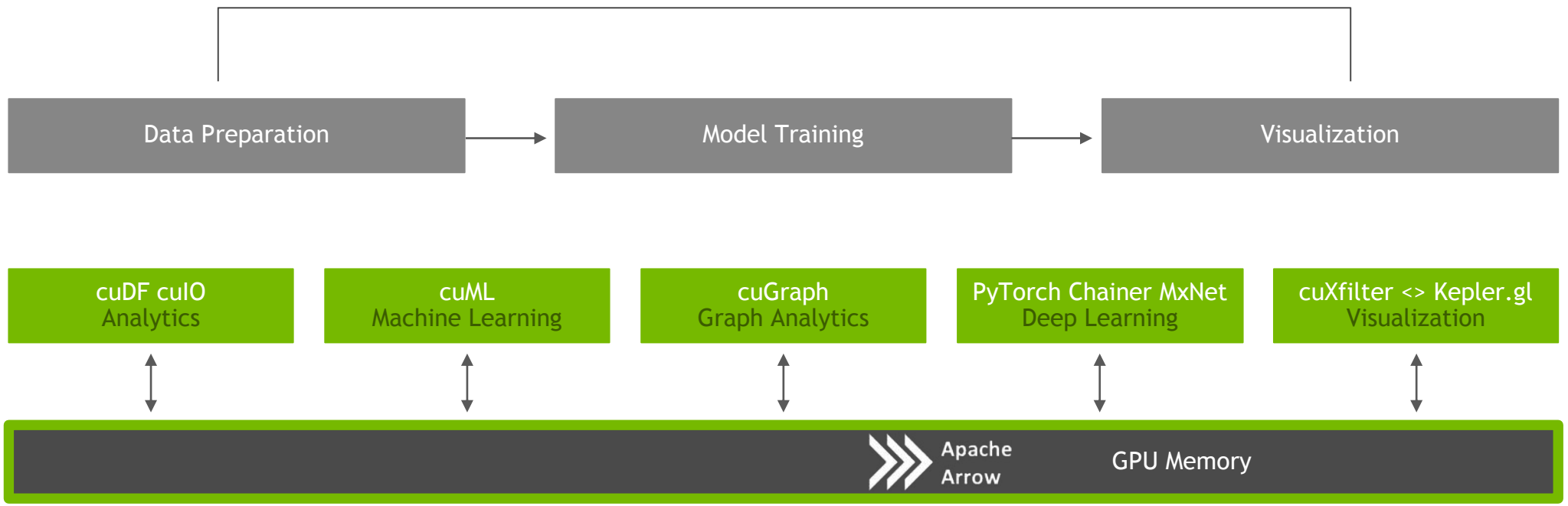
# WHAT IS RAPIDS?

## New GPU-Accelerated Data Science Pipeline

- Suite of open source, end-to-end data science tools
- Built on CUDA
- Unifying framework for GPU data science
- Pandas-like API for data preparation
- Scikit-learn-like API for machine learning

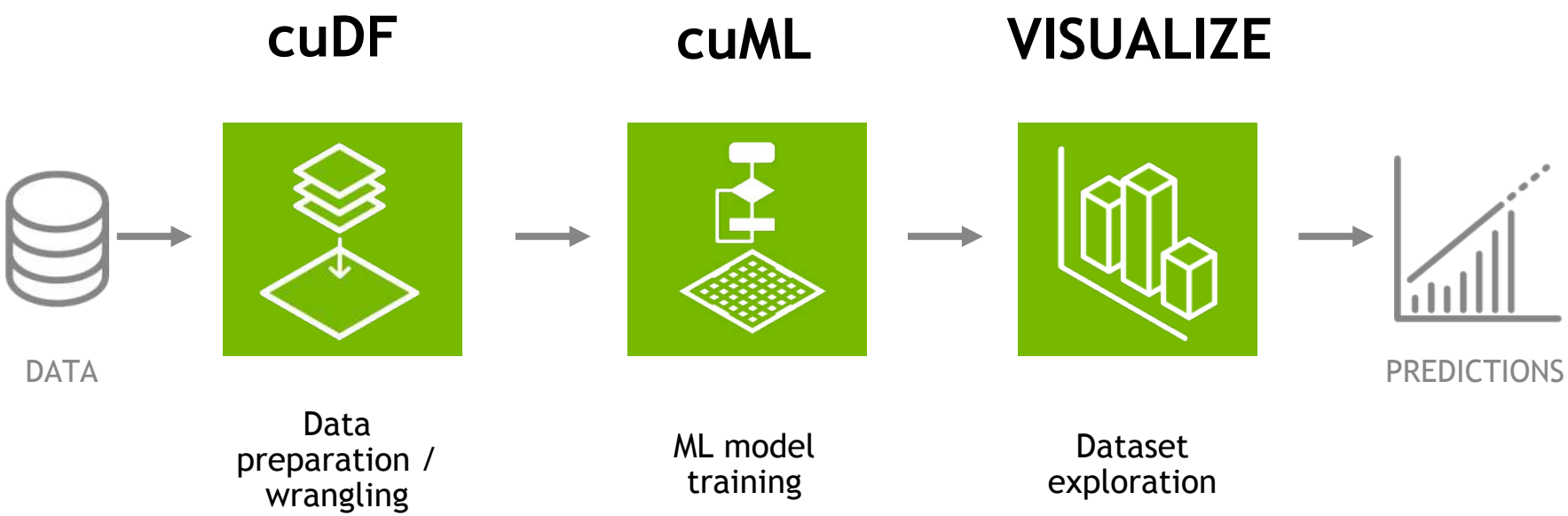
# RAPIDS

## End-to-End GPU-Accelerated Data Science



# Data Science Workflow with RAPIDS

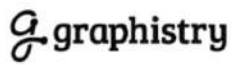
## Open Source, GPU-Accelerated ML Built on CUDA



# Ecosystem Partners

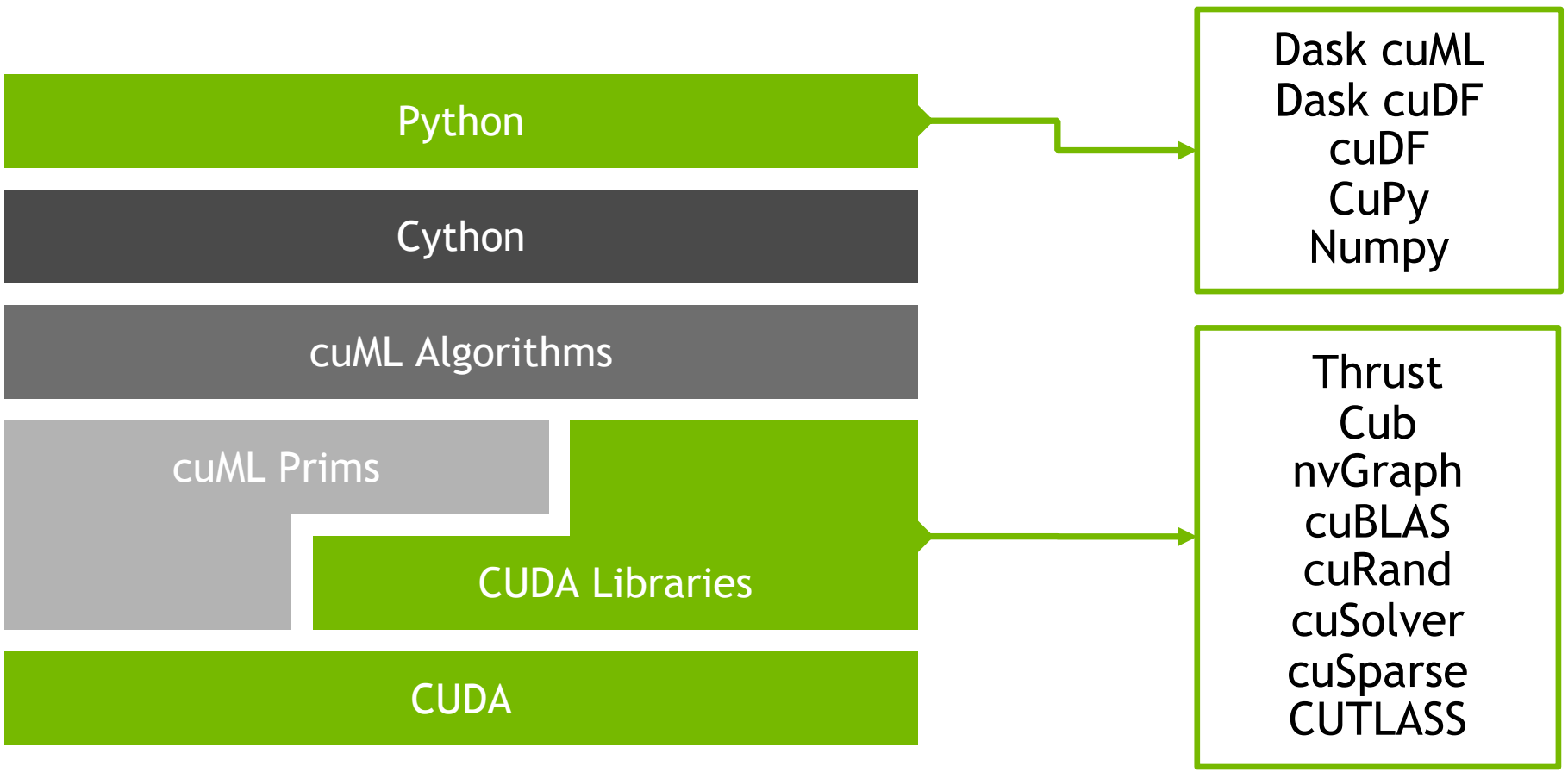
## Community Contributors


## Ecosystem Partners

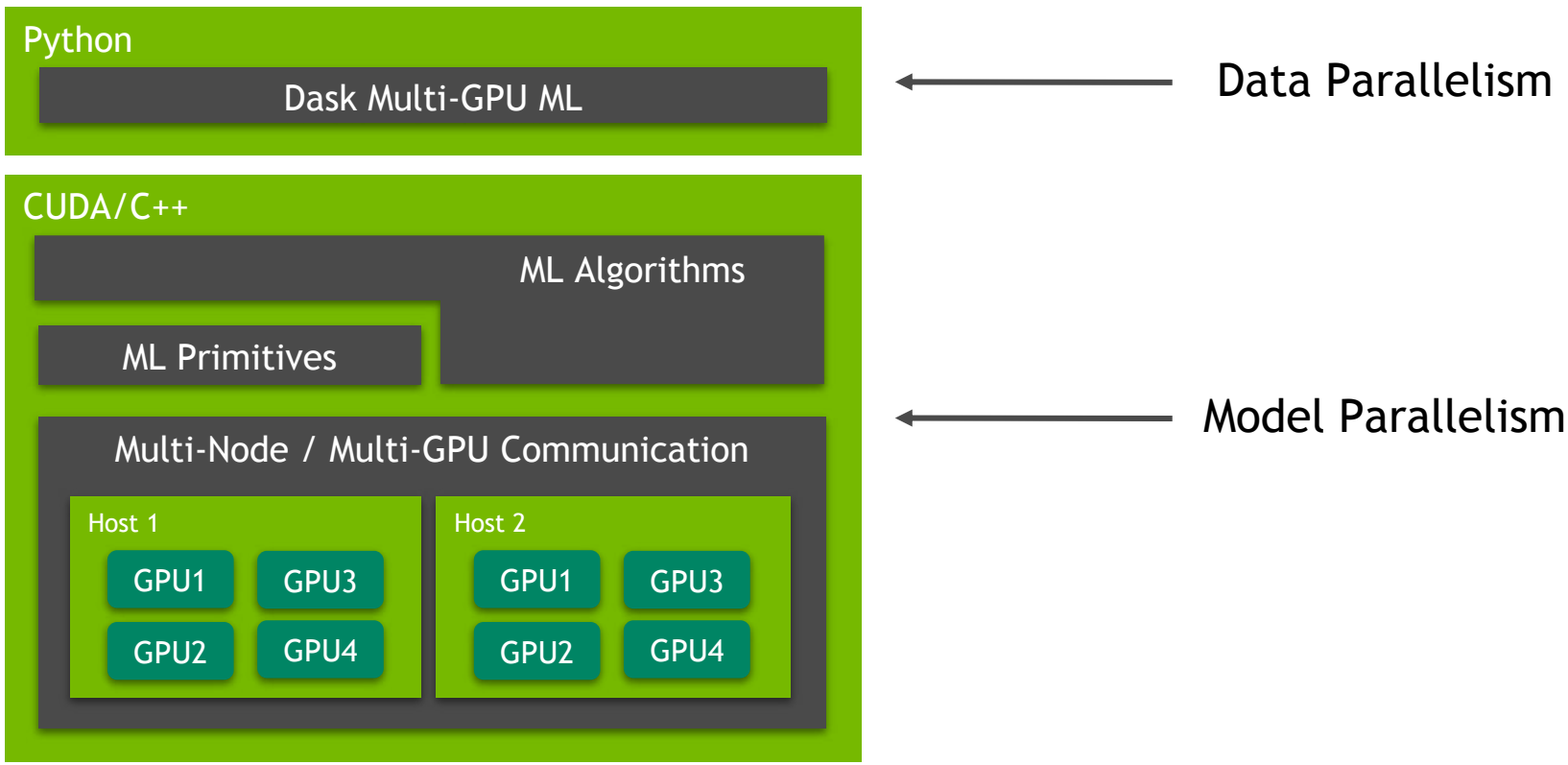



Walmart

# ML Technology Stack

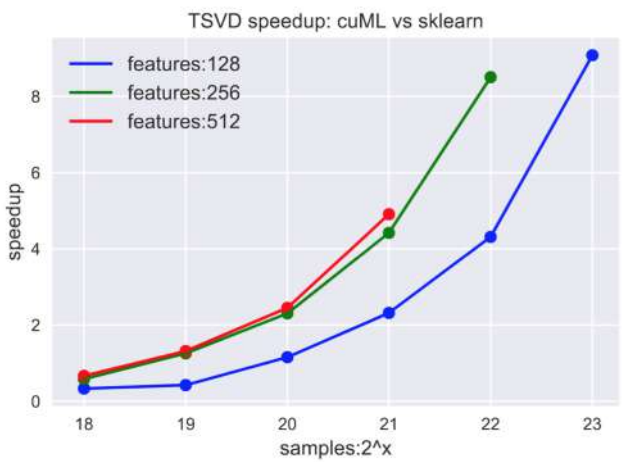


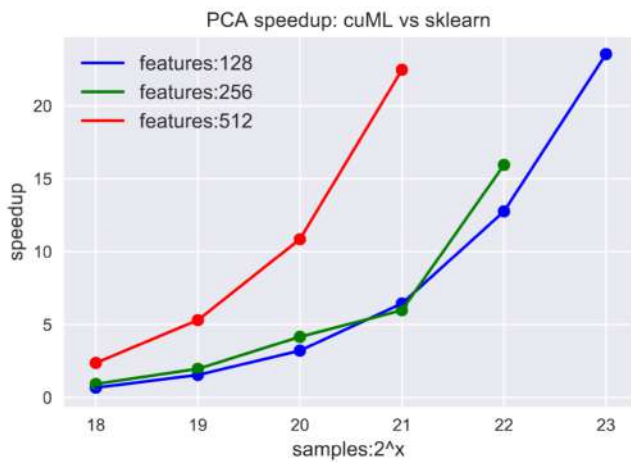
# High-Level APIs

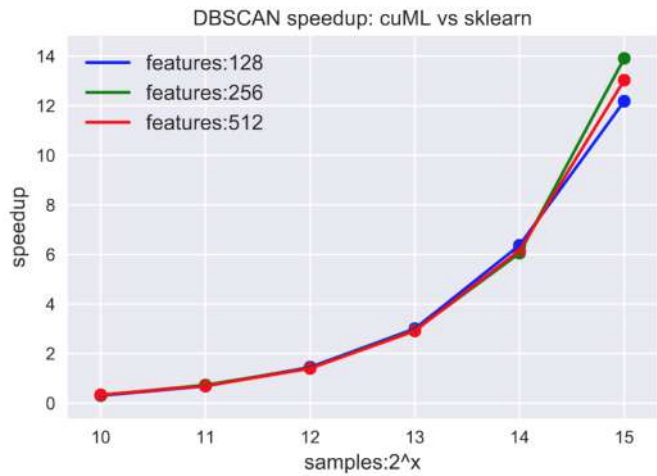


# cuML

## Benchmarks of initial algorithms



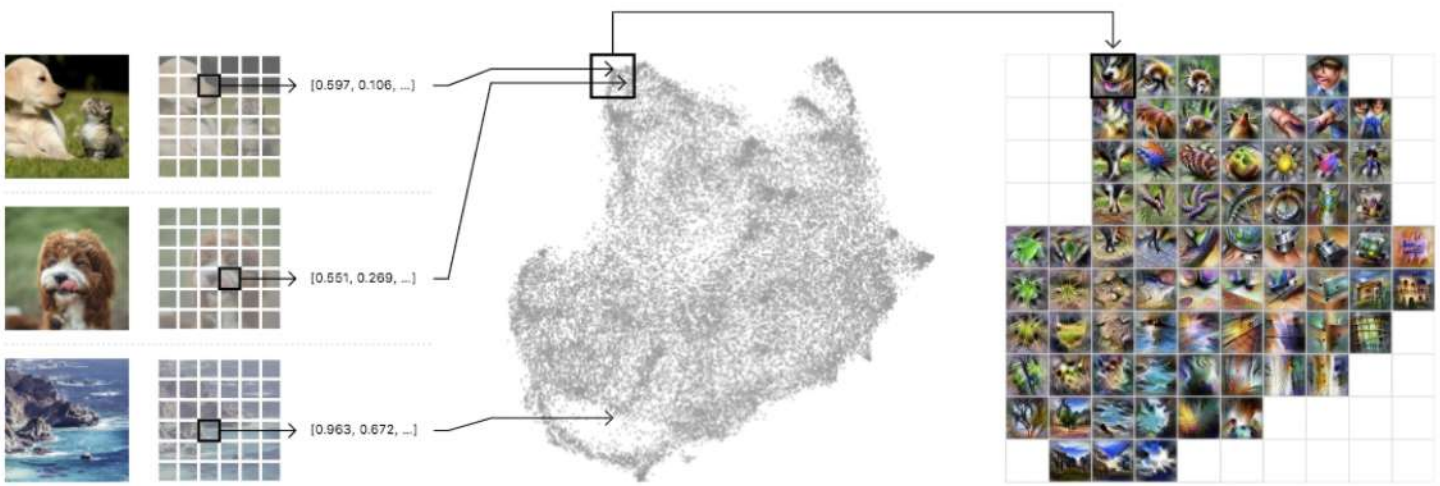



# UMAP

## Dimensionality reduction technique now on GPU

Uniform Manifold Approximation and Projection (UMAP) is a dimension reduction technique that can be used for visualization similarly to t-SNE, but also for general non-linear dimension reduction.




https://ai.googleblog.com/2019/03/exploring-neural-networks.html

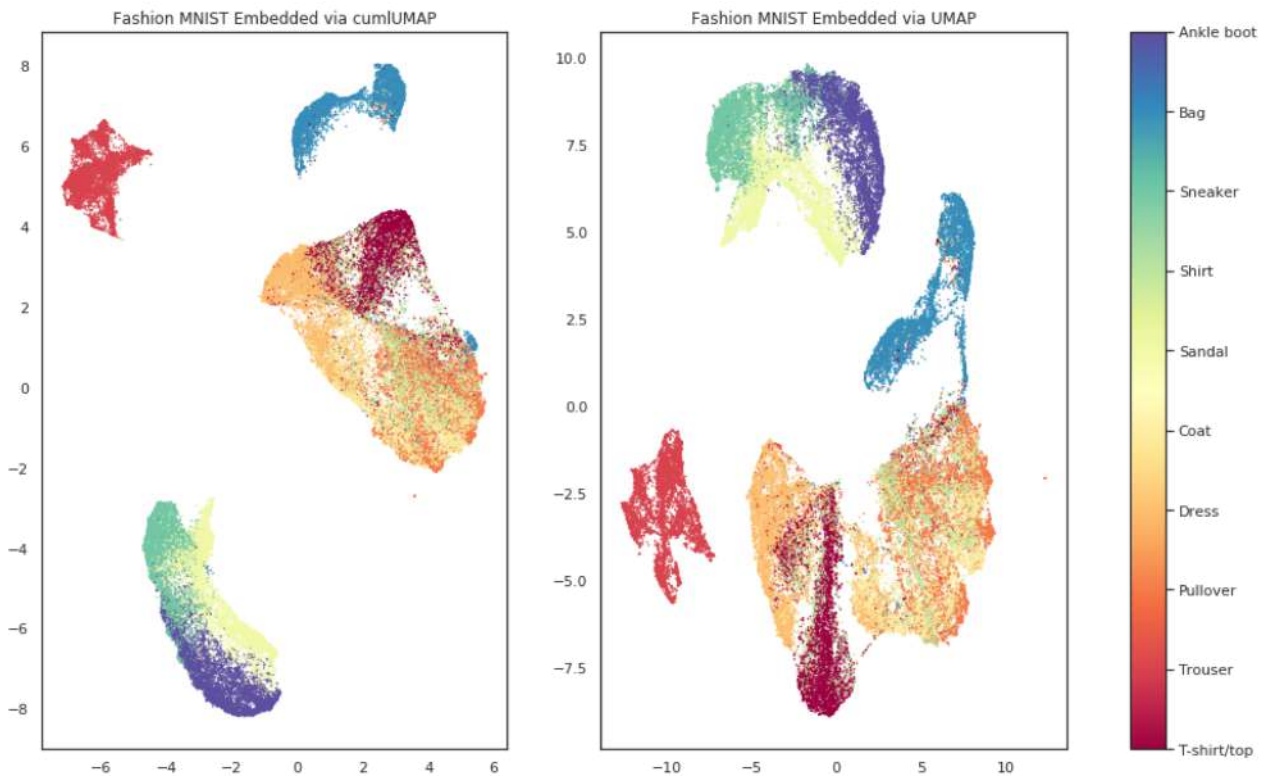
- **Fast**
- **General purpose** dimension reduction
- **Scales beyond** what most t-SNE packages can manage
- Often **preserves global structure** better than t-SNE
- Supports a **wide variety of distance functions**
- Supports **adding new points to an existing embedding** via the standard scikit-learn transform method
- **Supports supervised and semi-supervised** dimension reduction
- Has **solid theoretical foundations** in manifold learning

https://arxiv.org/pdf/1802.03426.pdf

# UMAP

## GPU vs CPU

GPU: 10.5 seconds




CPU: 100 seconds

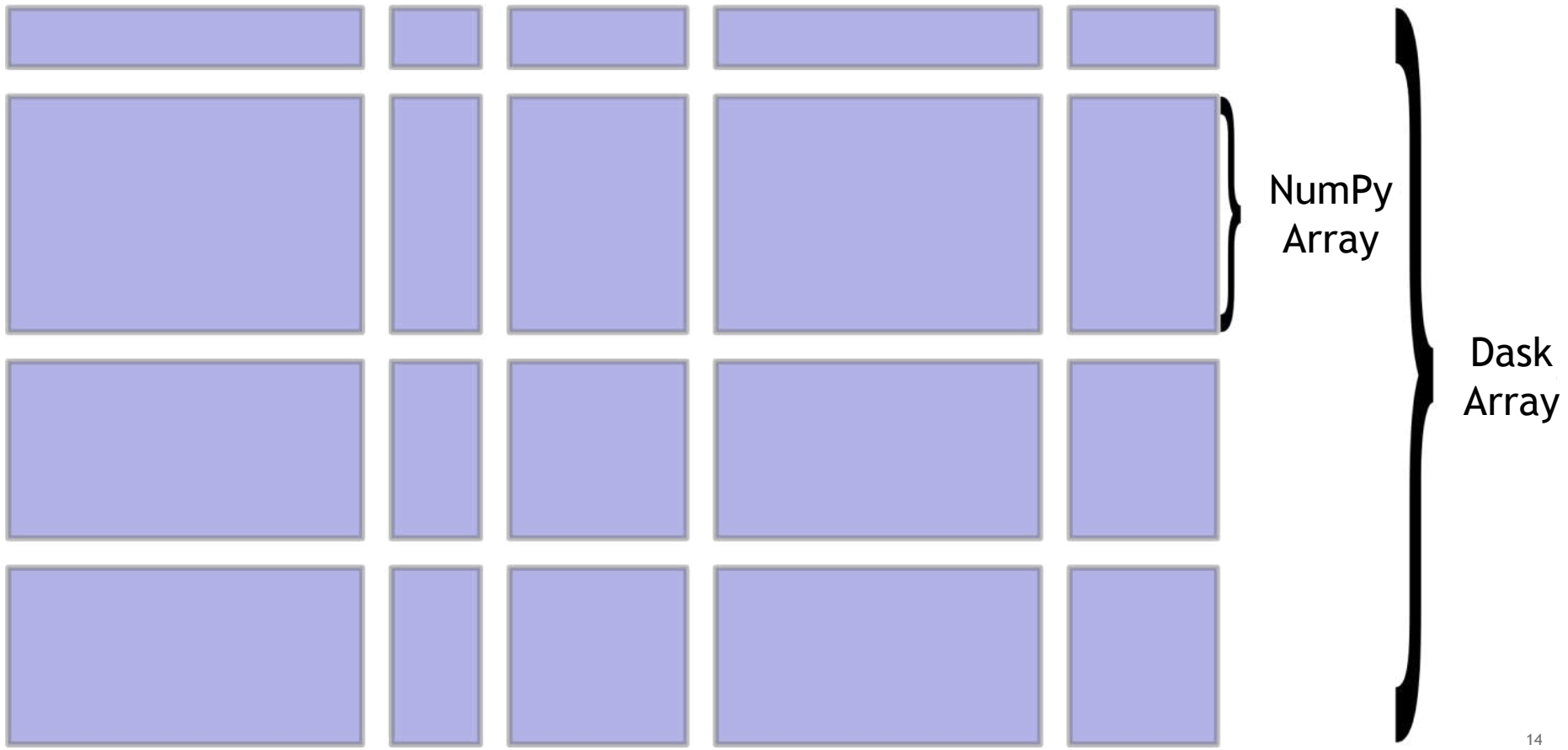
# Dask

## What is Dask and why does RAPIDS use it for scaling out?

- Distributed compute scheduler built to scale Python
- Scales workloads from laptops to supercomputer clusters
- Extremely modular: disjoint scheduling, compute, data transfer and out-of-core handling
- Multiple workers per node allow easier one-worker-per-GPU model

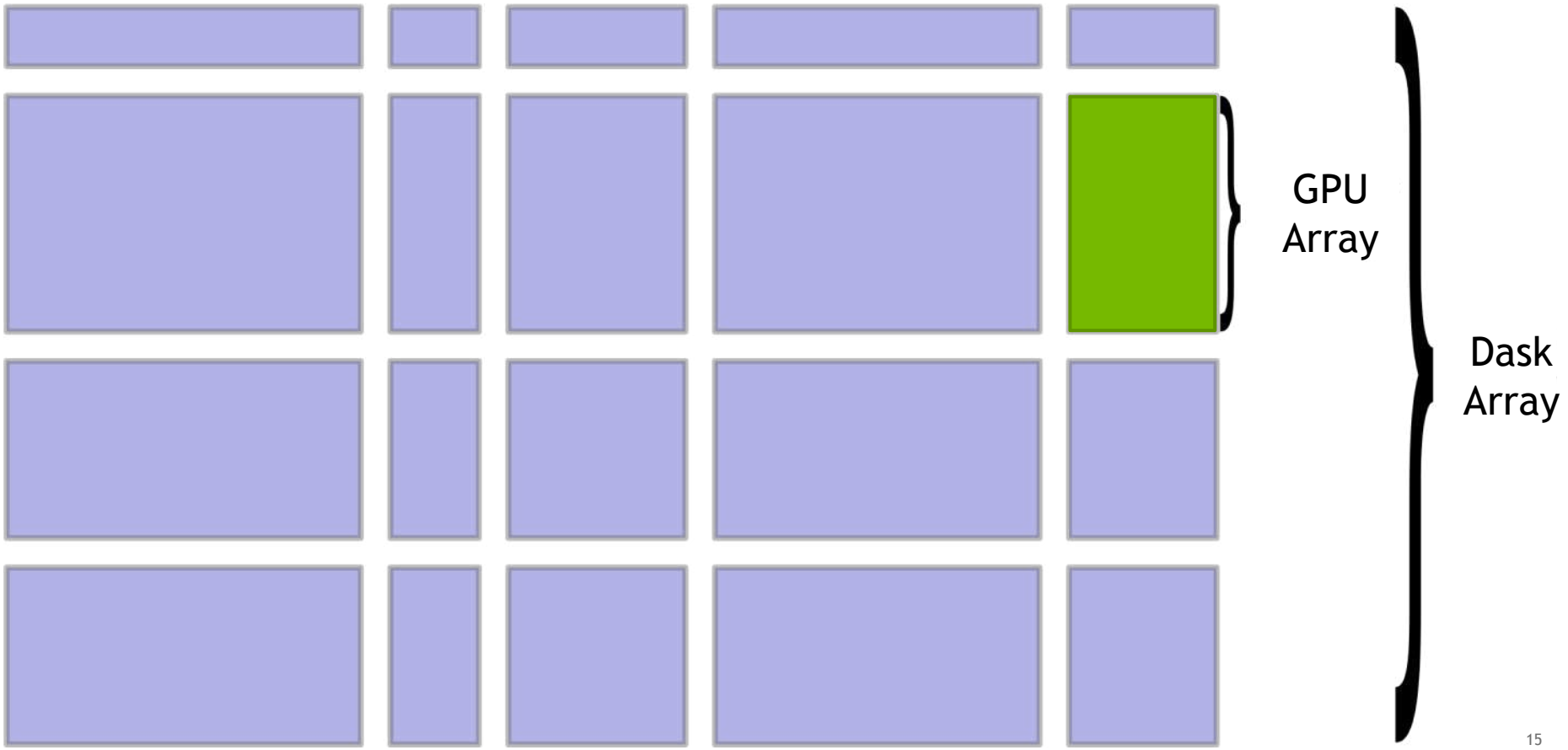
# Distributing Dask

## Distributed array from many arrays



# Combine Dask with CuPy

## Distributed GPU array from many GPU arrays



# NumPy Array Function (NEP-18)

## Interoperability of NumPy-like Libraries

- Function dispatch mechanism
- Allows using NumPy as a high-level API
- NumPy-like arrays need only to implement `__array_function__`



# Python CUDA Array Interface

## Interoperability for Python GPU Array Libraries

- GPU array standard
- Allows sharing GPU array between different libraries
- Native ingest and export of `__cuda_array_interface__` compatible objects via Numba device arrays in cuDF
- Numba, CuPy, and PyTorch are the first libraries to adopt the interface:
  - https://numba.pydata.org/numba-doc/dev/cuda/cuda_array_interface.html
  - https://github.com/cupy/cupy/releases/tag/v5.0.0b4
  - https://github.com/pytorch/pytorch/pull/11984

Numba

CuPy

PYTORCH

# Dask SVD Example

## Interoperability of NumPy-like Libraries

```
In [1]: import dask, dask.array
   ...: import numpy


In [2]: x = numpy.random.random((1000000, 1000))
   ...: dx = dask.array.from_array(x, chunks=(10000, 1000), asarray=False)

In [3]: u, s, v = numpy.linalg.svd(dx)

In [4]: %%time
   ...: u, s, v = dask.compute(u, s, v)
CPU times: user 39min 4s, sys: 47min 31s, total: 1h 26min 35s
Wall time: 1min 21s
```

# NumPy Array Function (NEP-18)

## Interoperability of NumPy-like Libraries

```
In [1]: import dask, dask.array
   ...: import numpy
   ...: import cupy

In [2]: x =  cupy.random.random((1000000, 1000))
   ...: dx = dask.array.from_array(x, chunks=(10000, 1000), asarray=False)

In [3]: u, s, v = numpy.linalg.svd(dx)

In [4]: %%time
   ...: u, s, v = dask.compute(u, s, v)
CPU times: user 34.5 s, sys: 17.6 s, total: 52.1 s
Wall time: 41 s
```

# NumPy Array Function (NEP-18)

## Interoperability of NumPy-like Libraries

```
In [1]: import dask, dask.array
   ...: import numpy
   ...: import cupy

In [2]: x =  cupy.random.random((1000000, 1000))
   ...: dx = dask.array.from_array(x, chunks=(10000, 1000), asarray=False)

In [3]: u, s, v = numpy.linalg.svd(dx)

In [4]: %%time
   ...: u, s, v = dask.compute(u, s, v, scheduler='single-threaded')
CPU times: user 21.2 s, sys: 11.1 s, total: 32.2 s
Wall time: 34.8 s
```

# Scale up with RAPIDS

Scale Up / Accelerate

## RAPIDS and Others

Accelerated on single GPU

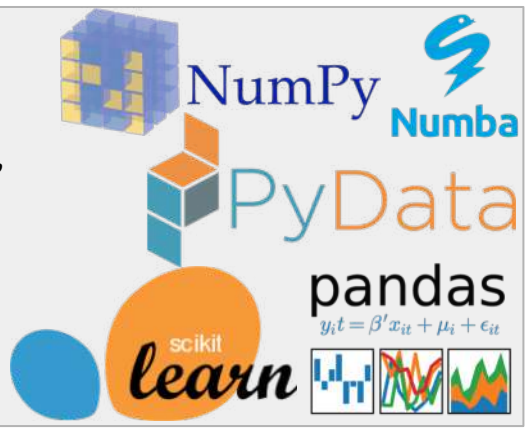
NumPy -> CuPy/PyTorch/..
Pandas -> cuDF
Scikit-Learn -> cuML
Numba -> Numba

RAPIDS

## PyData

NumPy, Pandas, Scikit-Learn,
Numba and many more

Single CPU core
In-memory data



# Scale up and out with RAPIDS and Dask

Scale Up / Accelerate

## RAPIDS and Others

Accelerated on single GPU

NumPy -> CuPy/PyTorch/..
Pandas -> cuDF
Scikit-Learn -> cuML
Numba -> Numba

RAPIDS

## Dask + RAPIDS

Multi-GPU
On single Node (DGX)
Or across a cluster

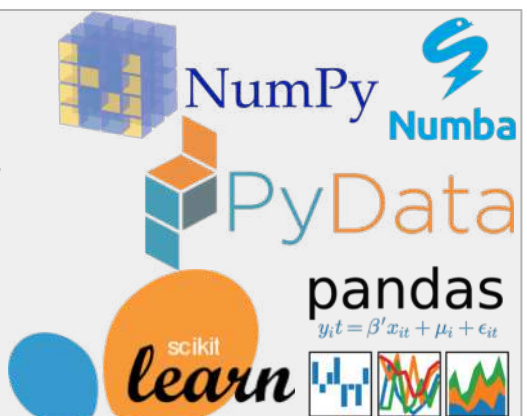
RAPIDS

DASK

## PyData

NumPy, Pandas, Scikit-Learn, Numba and many more

Single CPU core
In-memory data




## Dask

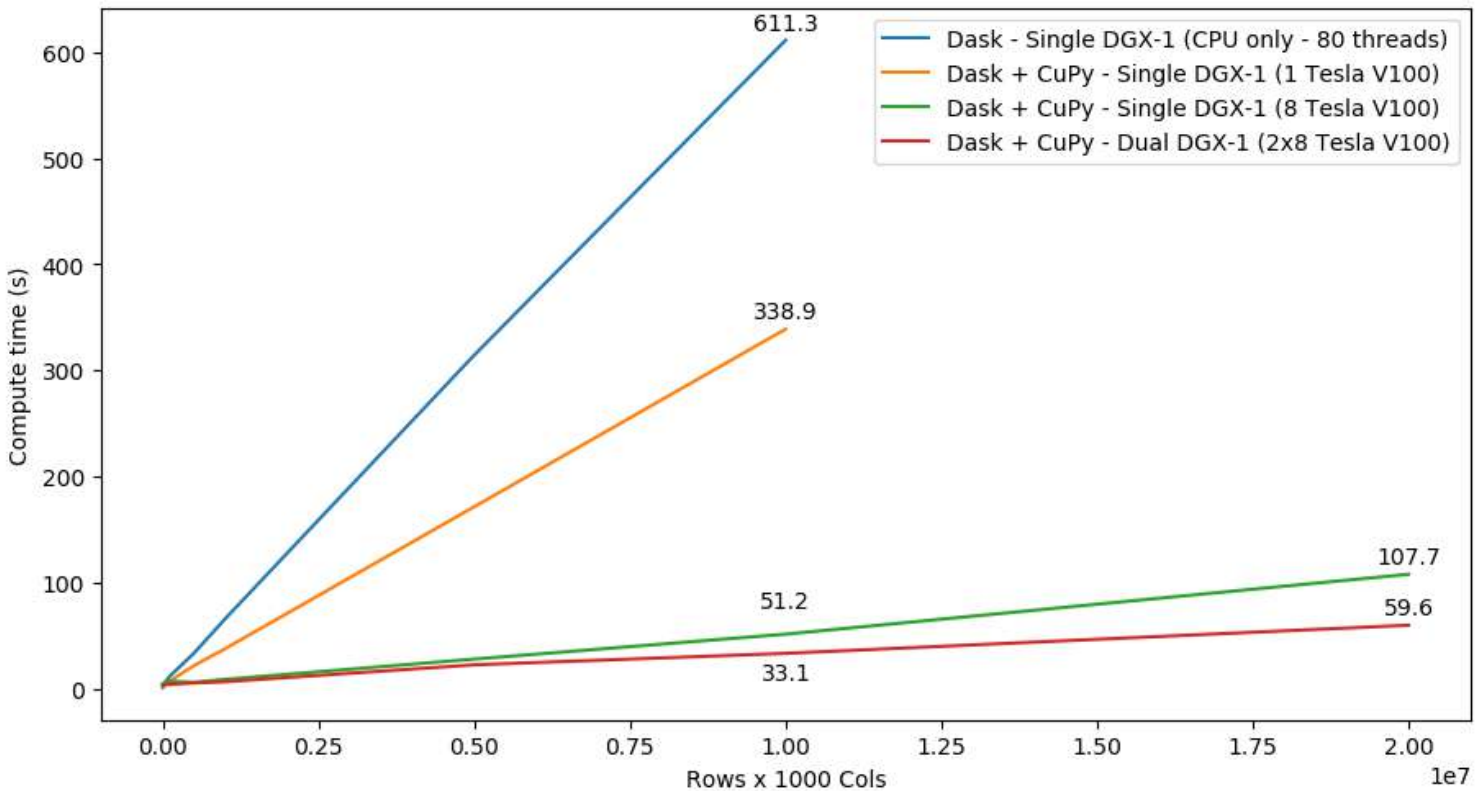
Multi-core and Distributed PyData

NumPy -> Dask Array
Pandas -> Dask DataFrame
Scikit-Learn -> Dask-ML
… -> Dask Futures

DASK

Scale out / Parallelize

# SVD Benchmark



# Road to 1.0

## October 2018 - RAPIDS 0.1

| cuML | Single-GPU | Multi-GPU | Multi-Node-Multi-GPU |
|---|---|---|---|
| Gradient Boosted Decision Trees (GBDT) | ■ | ■ | |
| GLM | | | |
| Logistic Regression | | | |
| Random Forest (regression) | | | |
| K-Means | | | |
| K-NN | | | |
| DBSCAN | ■ | | |
| UMAP | | | |
| ARIMA | | | |
| Kalman Filter | | | |
| Holts-Winters | | | |
| Principal Components | | | |
| Singular Value Decomposition | ■ | | |

# Road to 1.0

## May 2019 - RAPIDS 0.7

| cuML | Single-GPU | Multi-GPU | Multi-Node-Multi-GPU |
|---|---|---|---|
| Gradient Boosted Decision Trees (GBDT) | ■ | ■ | ■ |
| GLM | ■ | ■ | |
| Logistic Regression | ■ | | |
| Random Forest (regression) | ■ | ■ | ■ |
| K-Means | ■ | ■ | |
| K-NN | ■ | ■ | |
| DBSCAN | ■ | | |
| UMAP | ■ | | |
| ARIMA | | | |
| Kalman Filter | ■ | | |
| Holts-Winters | | | |
| Principal Components | ■ | | |
| Singular Value Decomposition | ■ | ■ | |

# Road to 1.0

## Q4 – 2019 - RAPIDS 0.12?

| cuML | Single-GPU | Multi-GPU | Multi-Node-Multi-GPU |
|---|---|---|---|
| Gradient Boosted Decision Trees (GBDT) | | | |
| GLM | | | |
| Logistic Regression | | | |
| Random Forest (regression) | | | |
| K-Means | | | |
| K-NN | | | |
| DBSCAN | | | |
| UMAP | | | |
| ARIMA | | | |
| Kalman Filter | | | |
| Holts-Winters | | | |
| Principal Components | | | |
| Singular Value Decomposition | | | |

# THANK YOU

Peter Andreas Entschev
pentschev@nvidia.com
@PeterEntschev