RAPIDS

The Platform Inside and Out
Release 0.18

@RAPIDSai

https://github.com/rapidsai

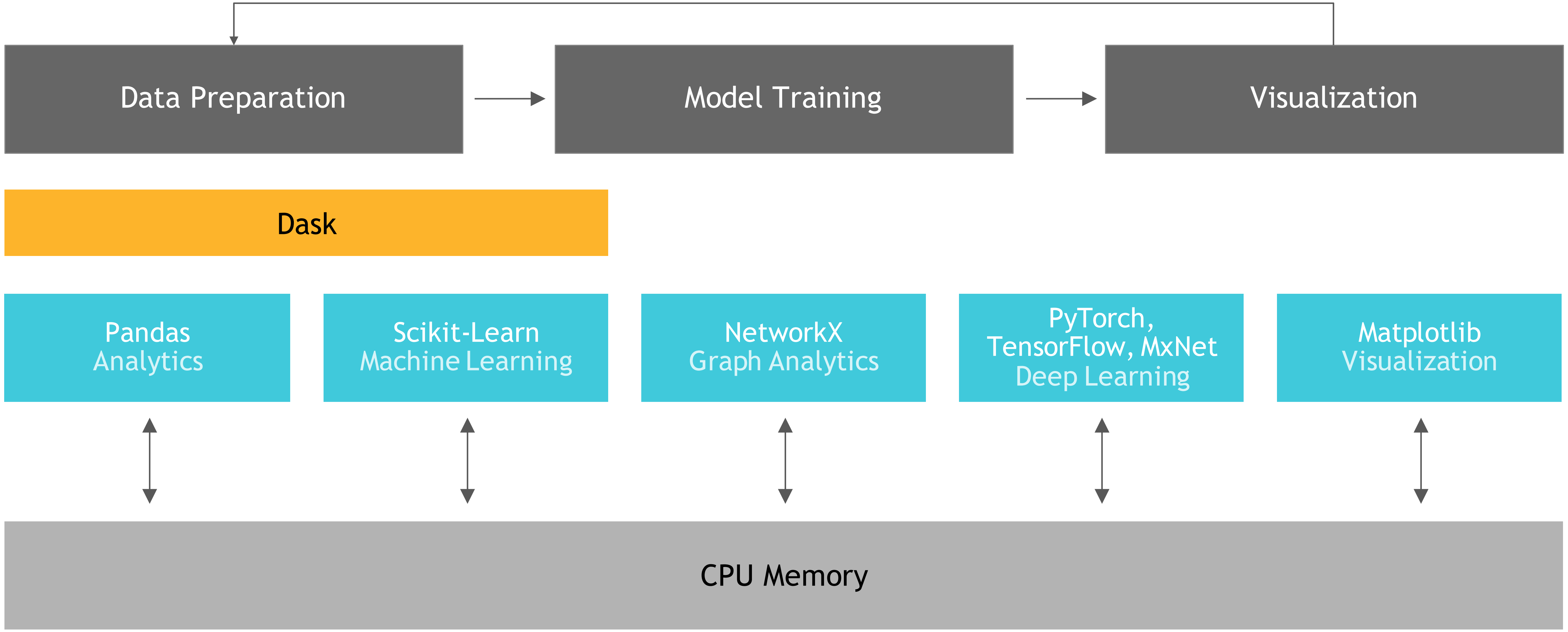
https://rapids-goai.slack.com/join

RAPIDS

https://rapids.ai

Open Source Data Science Ecosystem

Familiar Python APIs



Data Preparation

Model Training

Visualization

Dask

Pandas
Analytics

Scikit-Learn
Machine Learning

NetworkX
Graph Analytics

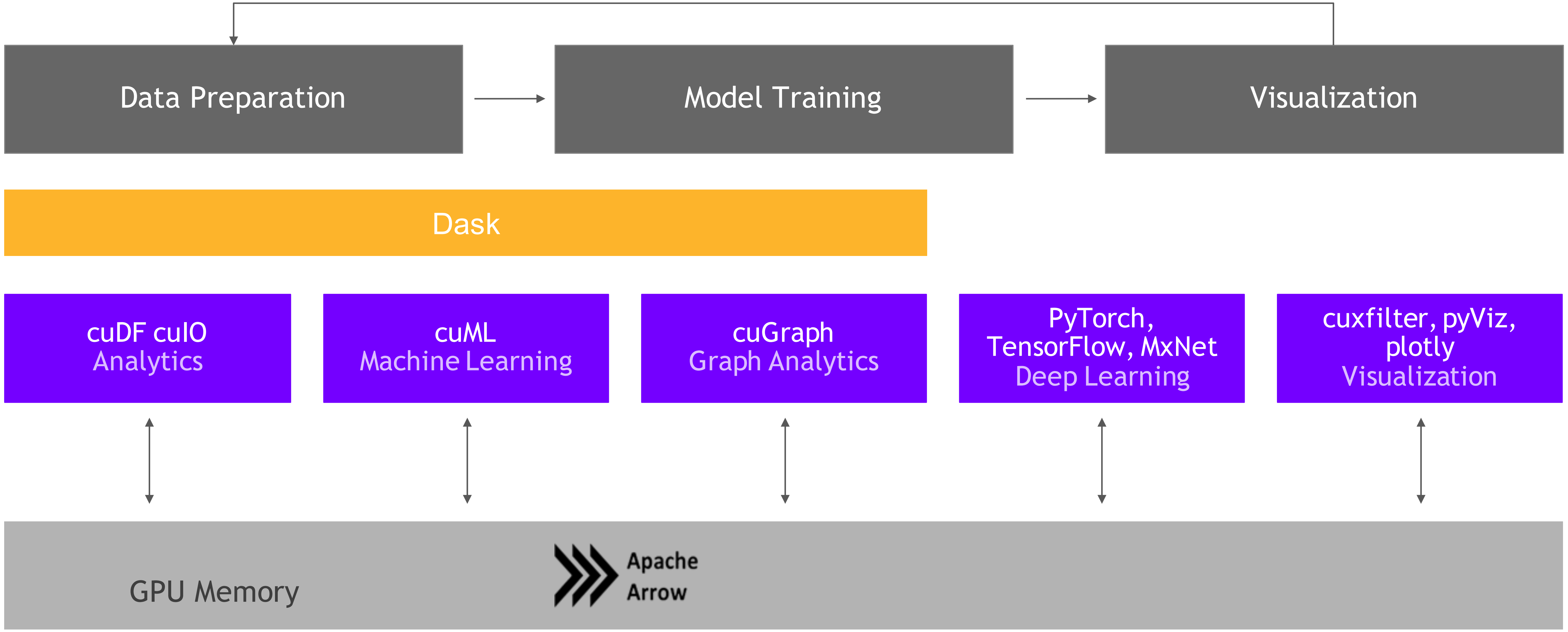
PyTorch,
TensorFlow, MxNet
Deep Learning

Matplotlib
Visualization

CPU Memory

RAPIDS

End-to-End GPU Accelerated Data Science



Data Preparation

Model Training

Visualization

Dask

cuDF cuIO
Analytics

cuML
Machine Learning

cuGraph
Graph Analytics

PyTorch,
TensorFlow, MxNet
Deep Learning

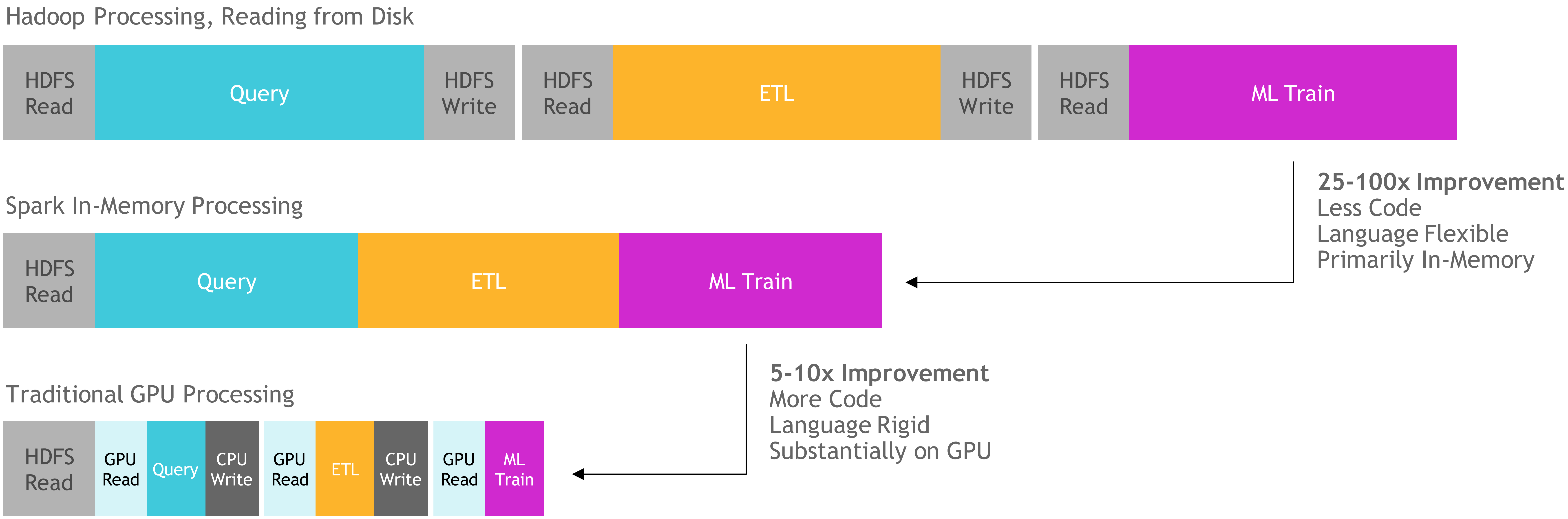
cuxfilter, pyViz,
plotly
Visualization

GPU Memory

Apache
Arrow

Data Processing Evolution

Faster Data Access, Less Data Movement

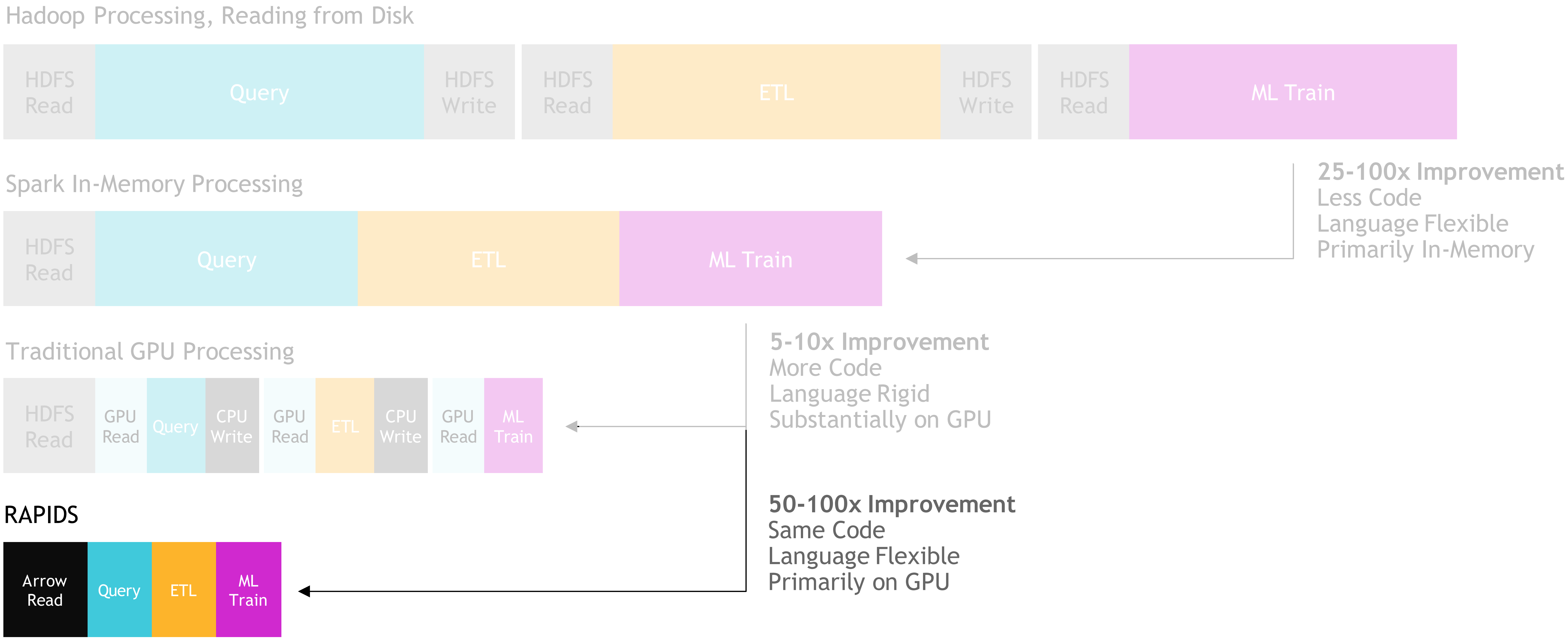


25-100x Improvement
Less Code
Language Flexible
Primarily In-Memory

5-10x Improvement
More Code
Language Rigid
Substantially on GPU

Data Processing Evolution

Faster Data Access, Less Data Movement



Hadoop Processing, Reading from Disk

HDFS Read | Query | HDFS Write | HDFS Read | ETL | HDFS Write | HDFS Read | ML Train

Spark In-Memory Processing

HDFS Read | Query | ETL | ML Train

25-100x Improvement
Less Code
Language Flexible
Primarily In-Memory

Traditional GPU Processing

HDFS Read | GPU Read | Query | CPU Write | GPU Read | ETL | CPU Write | GPU Read | ML Train

5-10x Improvement
More Code
Language Rigid
Substantially on GPU

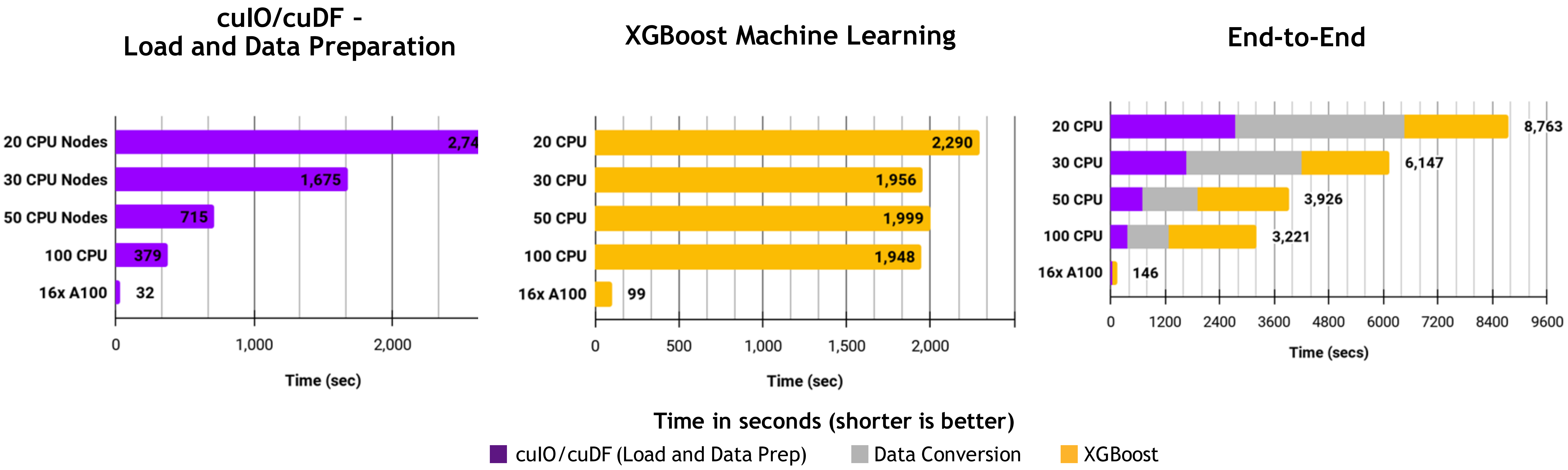
RAPIDS

Arrow Read | Query | ETL | ML Train

50-100x Improvement
Same Code
Language Flexible
Primarily on GPU

Faster Speeds, Real World Benefits

Faster Data Access, Less Data Movement



cuIO/cuDF – Load and Data Preparation

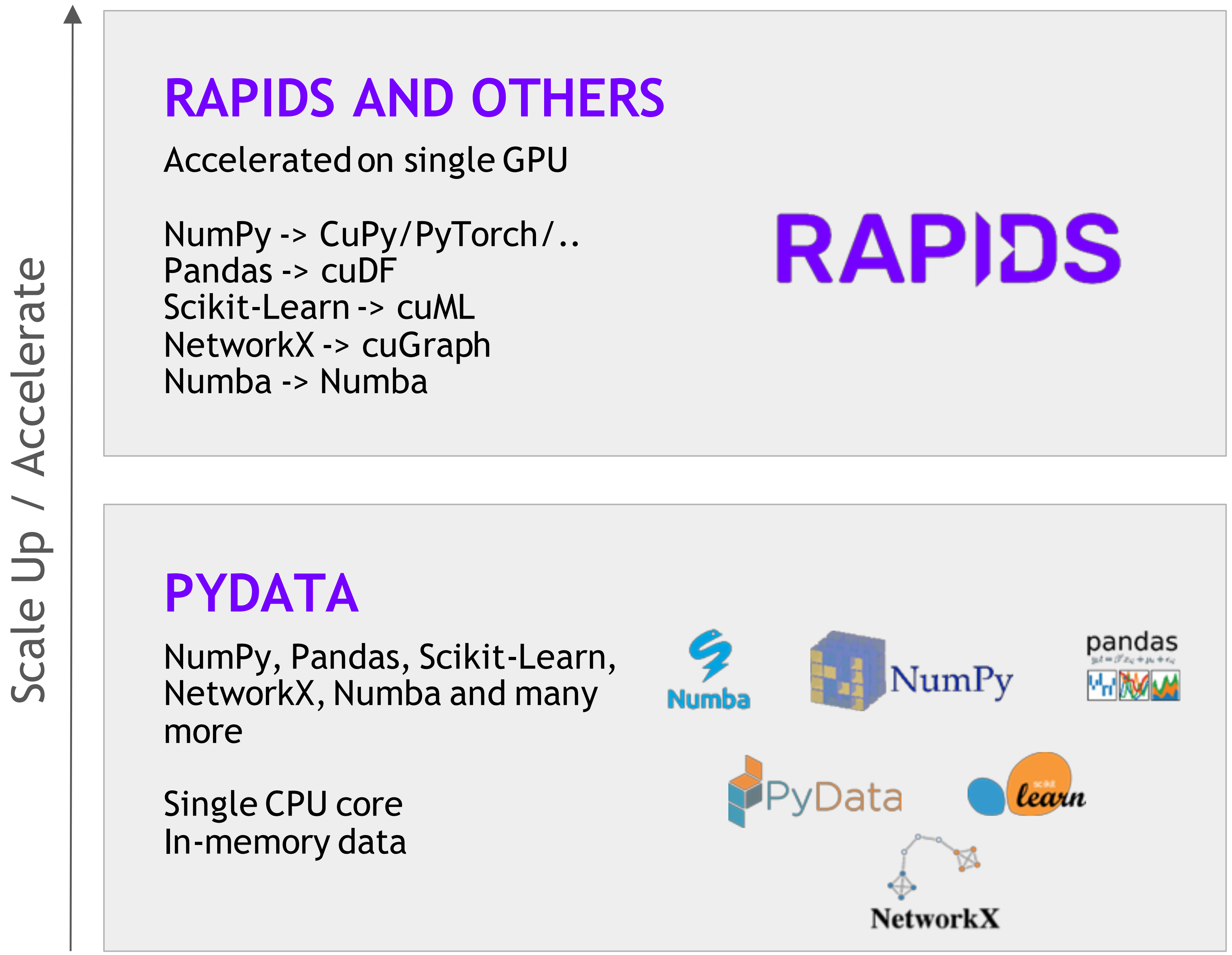
XGBoost Machine Learning

End-to-End

Time in seconds (shorter is better)

Benchmark	CPU Cluster Configuration	A100 Cluster Configuration	RAPIDS Version
200GB CSV dataset; Data prep includes joins, variable transformations	CPU nodes (61 GiB memory, 8 vCPUs, 64-bit platform), Apache Spark	16 A100 GPUs (40GB each)	RAPIDS 0.17

Scale Up with RAPIDS



RAPIDS AND OTHERS

Accelerated on single GPU

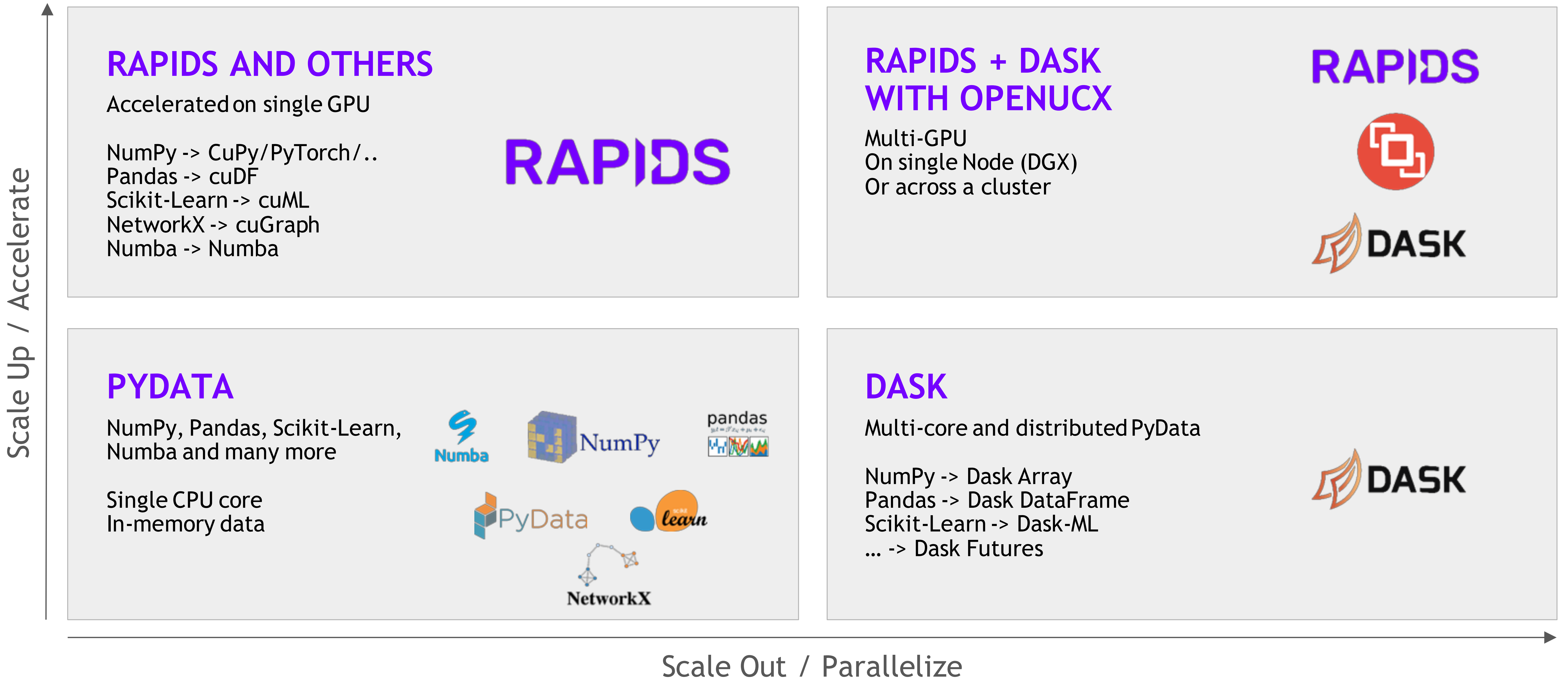
NumPy -> CuPy/PyTorch/..
Pandas -> cuDF
Scikit-Learn -> cuML
NetworkX -> cuGraph
Numba -> Numba

PYDATA

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

Single CPU core
In-memory data

Scale Out with RAPIDS + Dask with OpenUCX



RAPIDS AND OTHERS

Accelerated on single GPU

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

RAPIDS + DASK WITH OPENUCX

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

PYDATA

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

Single CPU core
In-memory data

DASK

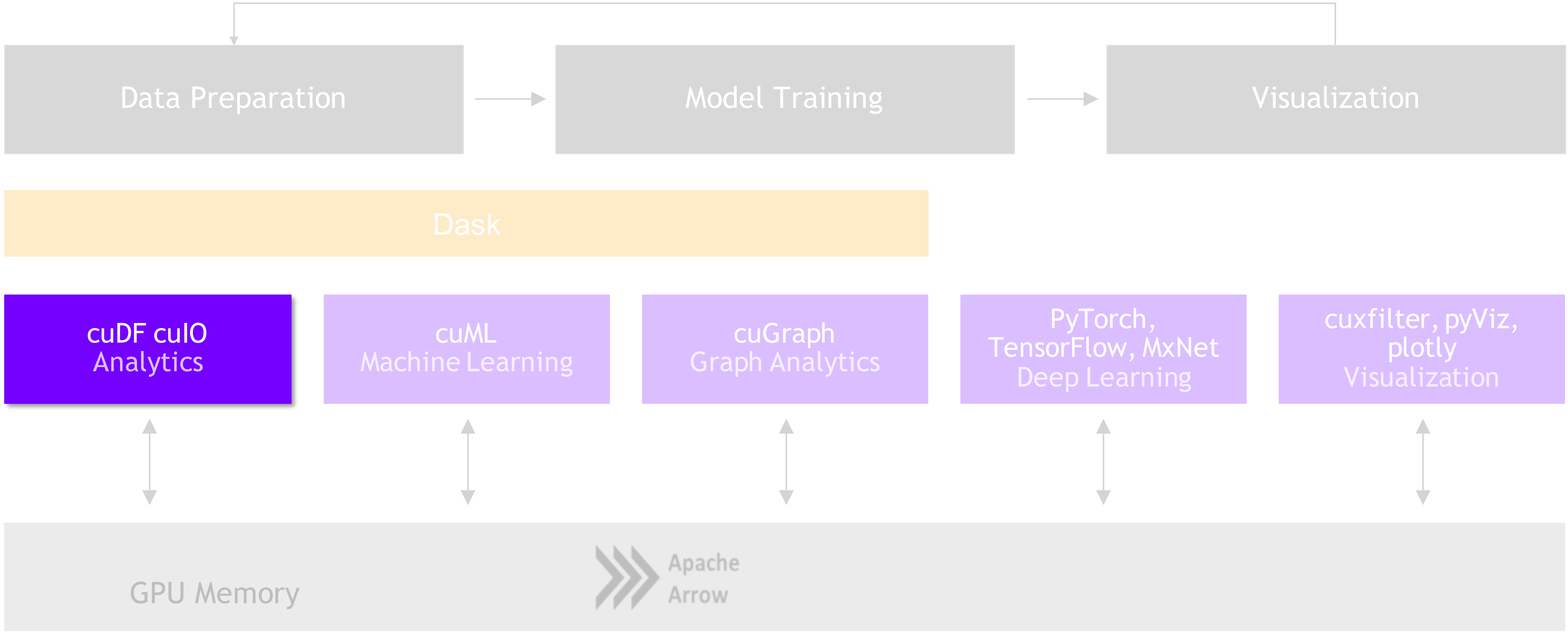
Multi-core and distributed PyData

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

cuDF

RAPIDS

GPU Accelerated Data Wrangling and Feature Engineering

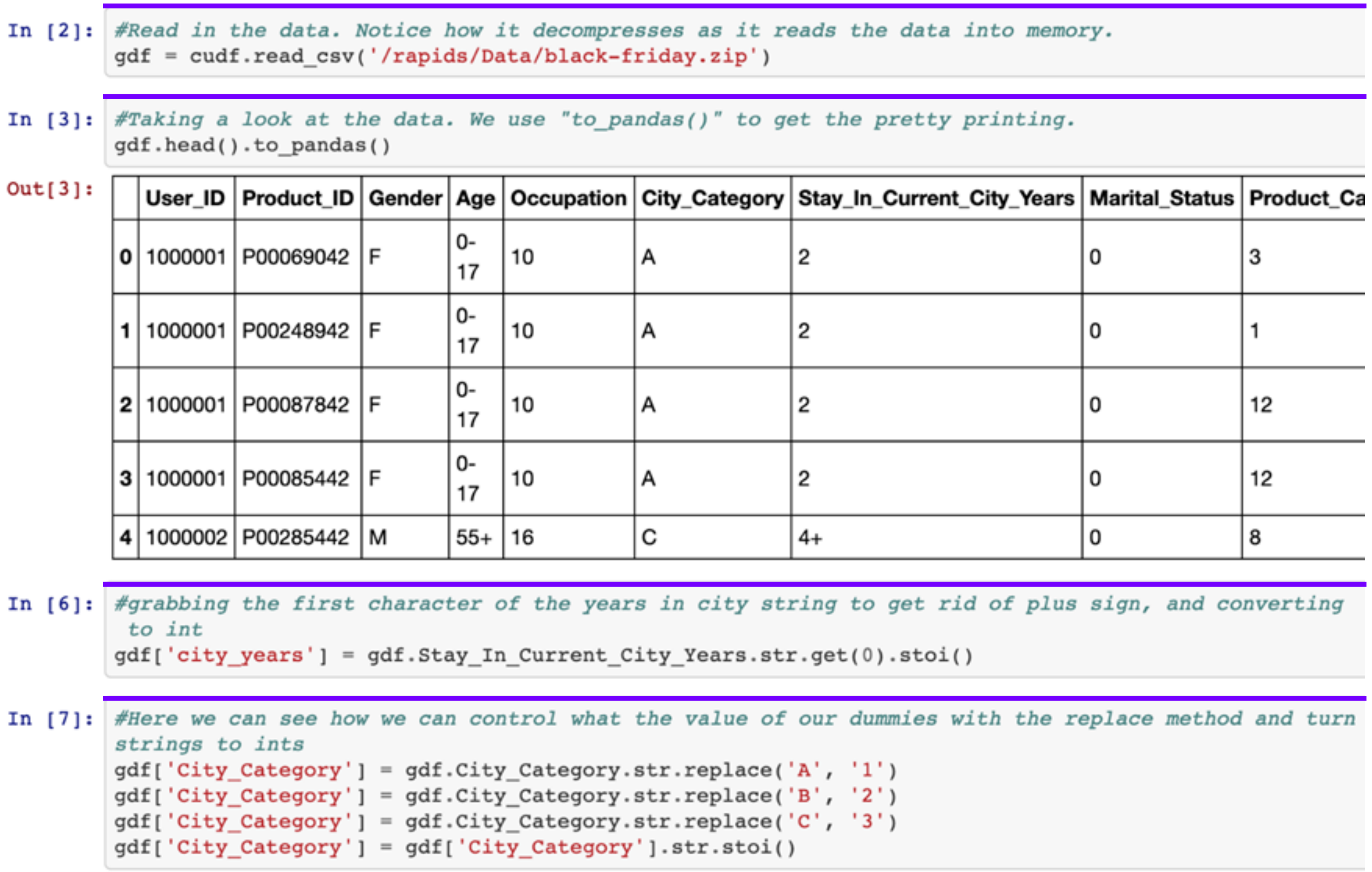


ETL - the Backbone of Data Science

cuDF is…

PYTHON LIBRARY

- A Python library for manipulating GPU DataFrames following the Pandas API
- Python interface to CUDA C++ library with additional functionality
- Creating GPU DataFrames from Numpy arrays, Pandas DataFrames, and PyArrow Tables
- JIT compilation of User-Defined Functions (UDFs) using Numba



```
In [2]: #Read in the data. Notice how it decompresses as it reads the data into memory.
        gdf = cudf.read_csv('/rapids/Data/black-friday.zip')
```

```
In [3]: #Taking a look at the data. We use "to_pandas()" to get the pretty printing.
        gdf.head().to_pandas()
```

Out[3]:

	User_ID	Product_ID	Gender	Age	Occupation	City_Category	Stay_In_Current_City_Years	Marital_Status	Product_Ca
0	1000001	P00069042	F	0-17	10	A	2	0	3
1	1000001	P00248942	F	0-17	10	A	2	0	1
2	1000001	P00087842	F	0-17	10	A	2	0	12
3	1000001	P00085442	F	0-17	10	A	2	0	12
4	1000002	P00285442	M	55+	16	C	4+	0	8

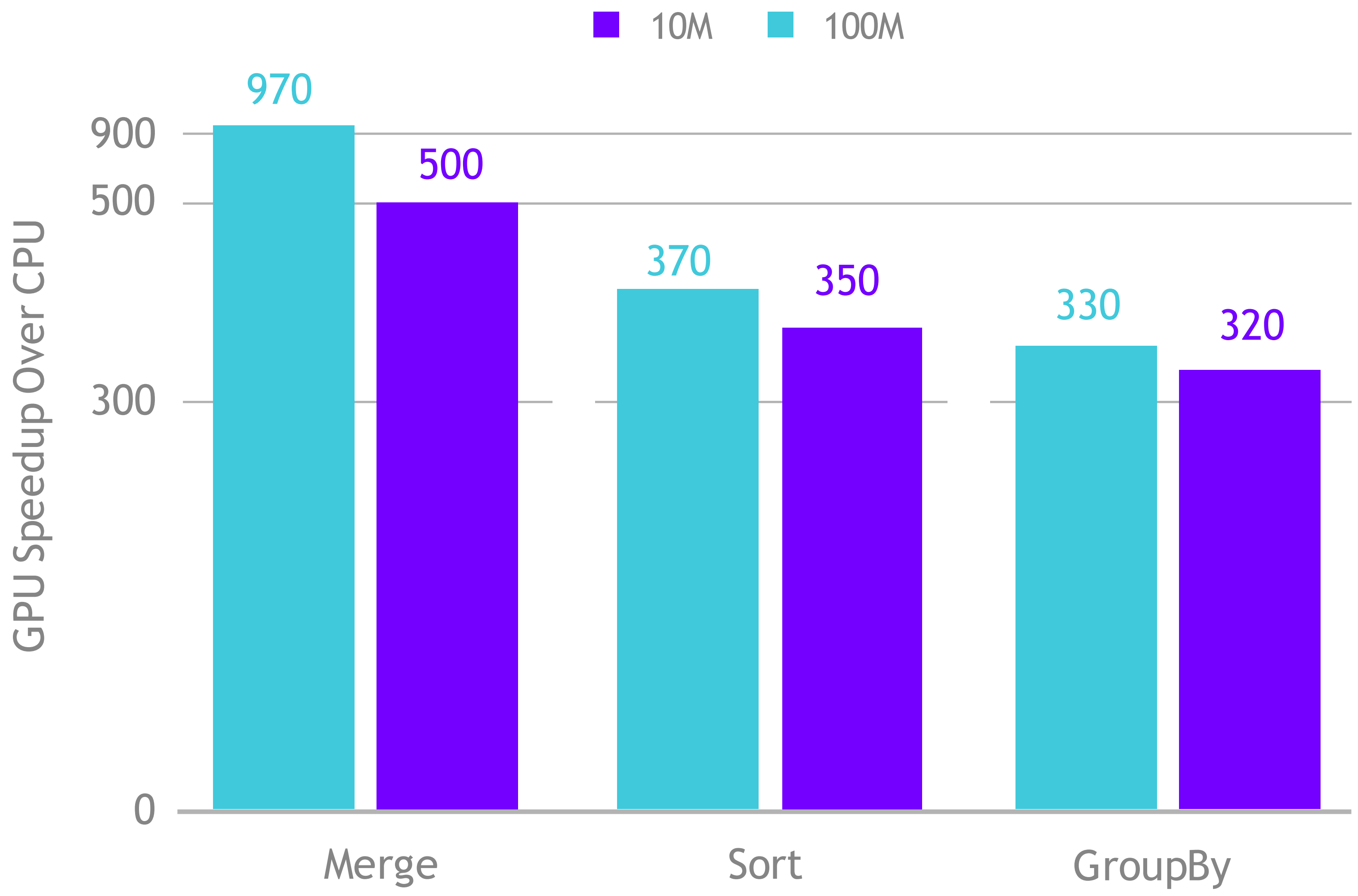
```
In [6]: #grabbing the first character of the years in city string to get rid of plus sign, and converting
         to int
        gdf['city_years'] = gdf.Stay_In_Current_City_Years.str.get(0).stoi()
```

```
In [7]: #Here we can see how we can control what the value of our dummies with the replace method and turn
        strings to ints
        gdf['City_Category'] = gdf.City_Category.str.replace('A', '1')
        gdf['City_Category'] = gdf.City_Category.str.replace('B', '2')
        gdf['City_Category'] = gdf.City_Category.str.replace('C', '3')
        gdf['City_Category'] = gdf['City_Category'].str.stoi()
```

Benchmarks: Single-GPU Speedup vs. Pandas

cuDF v0.13, Pandas 0.25.3

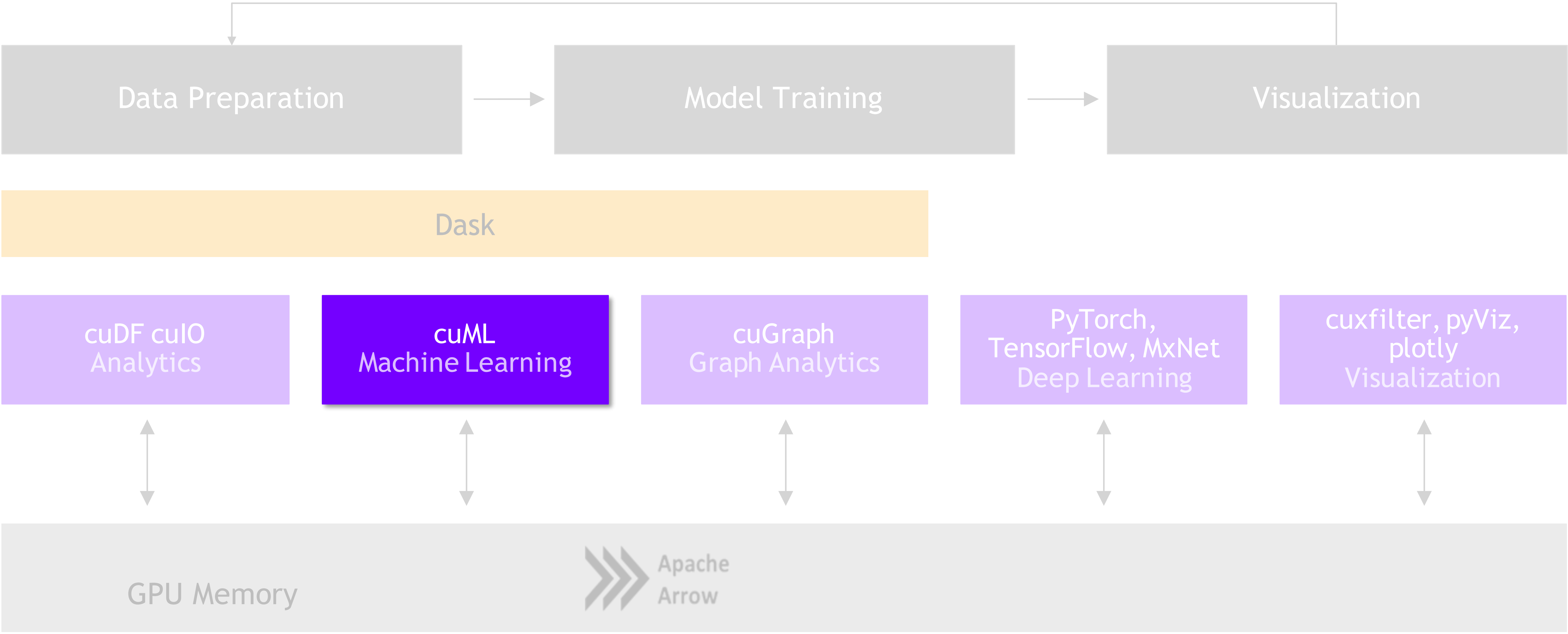
- Running on NVIDIA DGX-1:
  - GPU: NVIDIA Tesla V100 32GB
  - CPU: Intel(R) Xeon(R) CPU E5-2698 v4 @ 2.20GHz
- Benchmark Setup:
  - RMM Pool Allocator Enabled
  - DataFrames: 2x int32 columns key columns, 3x int32 value columns
  - Merge: inner; GroupBy: count, sum, min, max calculated for each value column



cuML

Machine Learning

More Models More Problems



Data Preparation

Model Training

Visualization

Dask

cuDF cuIO
Analytics

cuML
Machine Learning

cuGraph
Graph Analytics

PyTorch, TensorFlow, MxNet
Deep Learning

cuxfilter, pyViz, plotly
Visualization

GPU Memory

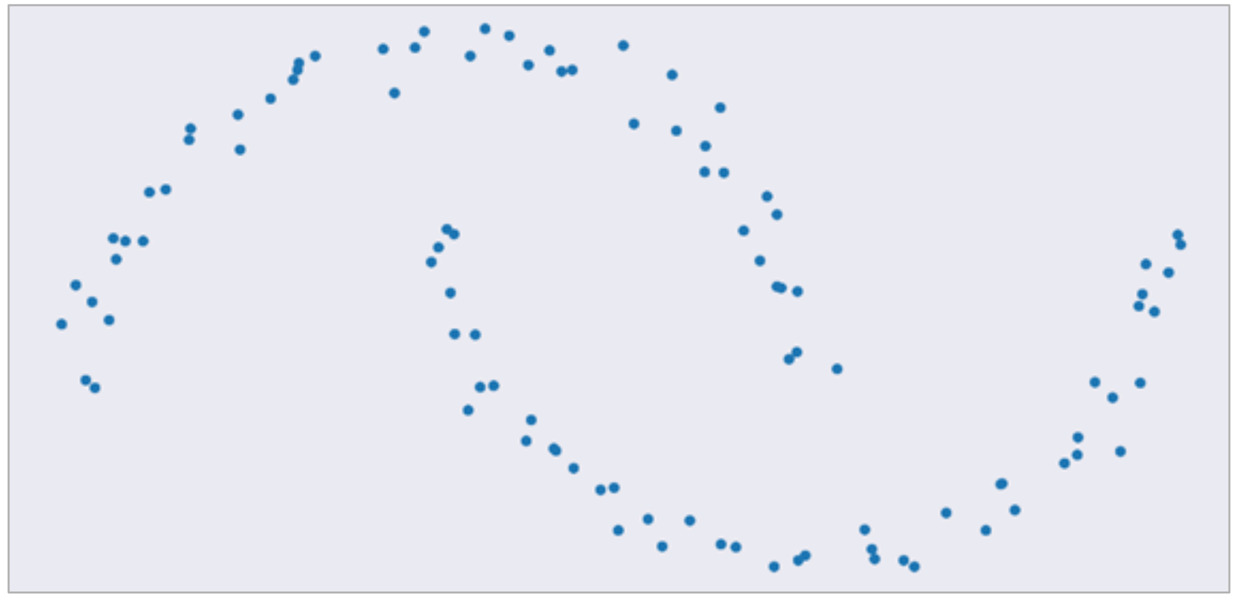
Apache Arrow

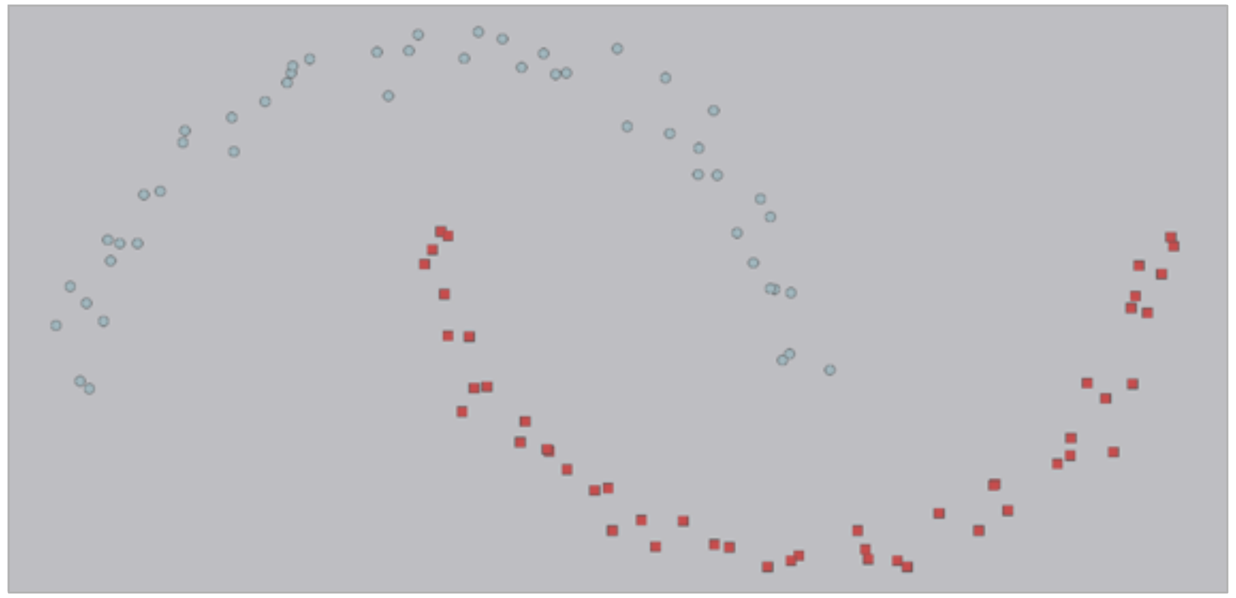
RAPIDS Matches Common Python APIs

CPU-based Clustering

```
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)


y_hat = dbscan.fit_predict(X)
```



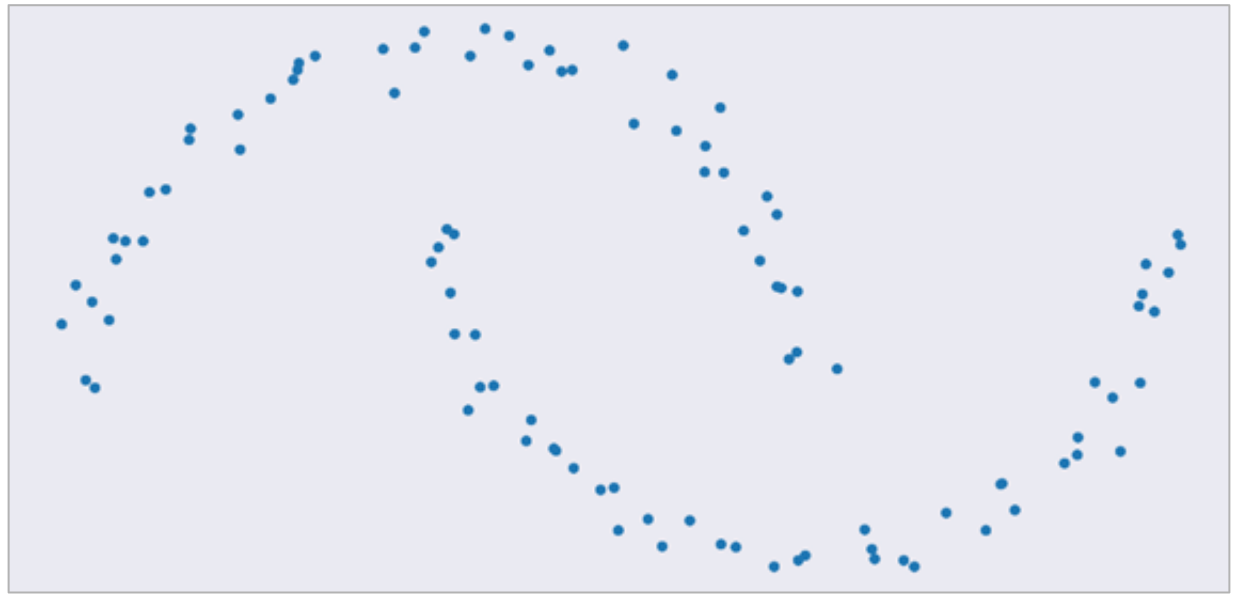


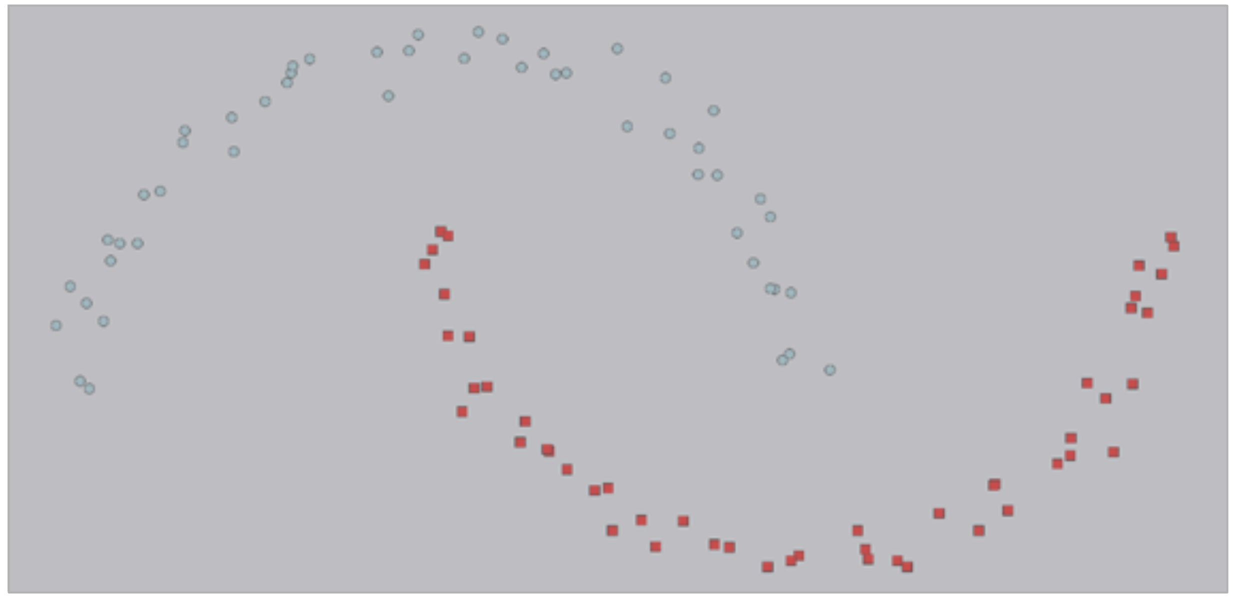
RAPIDS Matches Common Python APIs

GPU-accelerated Clustering

```
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)


y_hat = dbscan.fit_predict(X)
```





Algorithms

GPU-accelerated Scikit-Learn

Classification / Regression

Decision Trees / **Random Forests**
Linear/Lasso/Ridge/**LARS**/ElasticNet Regression
Logistic Regression
K-Nearest Neighbors (**exact or approximate**)
Support Vector Machine Classification and Regression
Naive Bayes

Inference

Random Forest / GBDT Inference (FIL)

Preprocessing

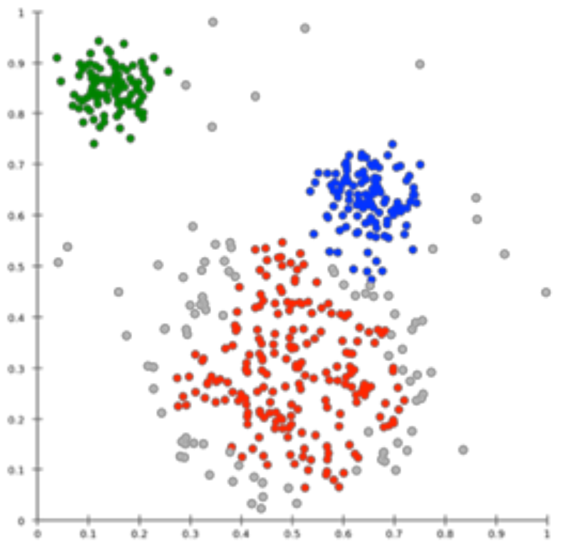
Text vectorization (TF-IDF / Count)
Target Encoding
Cross-validation / splitting

Clustering Decomposition & Dimensionality Reduction

K-Means
DBSCAN
Spectral Clustering
Principal Components (including iPCA)
Singular Value Decomposition
UMAP
Spectral Embedding
T-SNE

Time Series

Holt-Winters
Seasonal ARIMA / Auto ARIMA

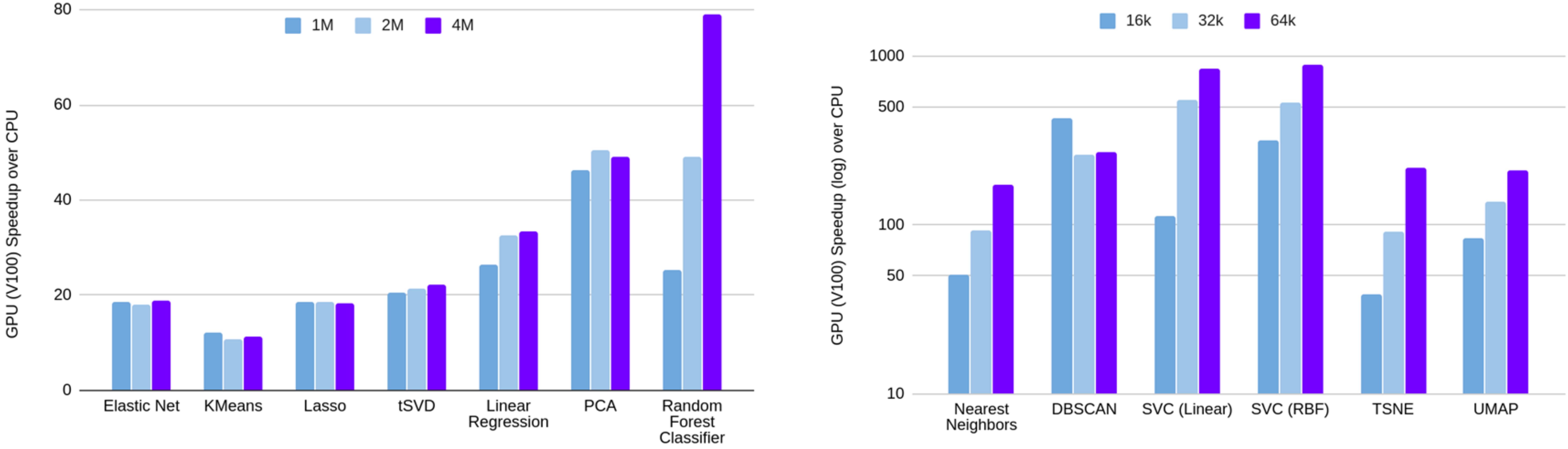


Cross Validation

Hyper-parameter Tuning

More to come!

Benchmarks: Single-GPU cuML vs Scikit-learn

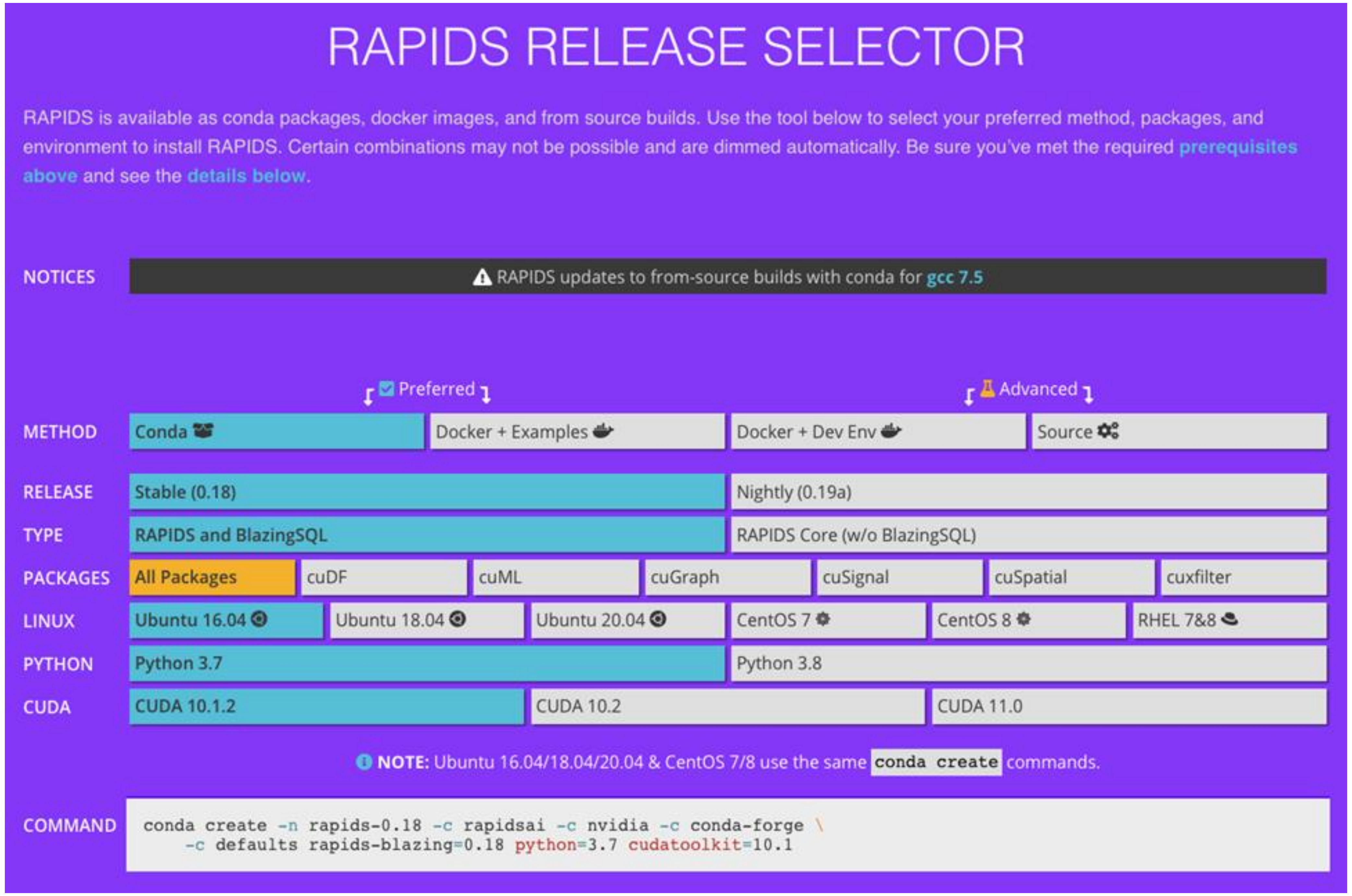


1x V100 vs. 2x 20 Core CPUs (DGX-1, RAPIDS 0.15)

Getting Started

Easy Installation

Interactive Installation Guide



RAPIDS RELEASE SELECTOR

RAPIDS is available as conda packages, docker images, and from source builds. Use the tool below to select your preferred method, packages, and environment to install RAPIDS. Certain combinations may not be possible and are dimmed automatically. Be sure you've met the required prerequisites above and see the details below.

NOTICES ⚠ RAPIDS updates to from-source builds with conda for gcc 7.5

	Preferred		Advanced	
METHOD	[✓ Conda]	Docker + Examples	Docker + Dev Env	Source

RELEASE [✓ Stable (0.18)] | Nightly (0.19a)

TYPE [✓ RAPIDS and BlazingSQL] | RAPIDS Core (w/o BlazingSQL)

PACKAGES [✓ All Packages] | cuDF | cuML | cuGraph | cuSignal | cuSpatial | cuxfilter

LINUX [✓ Ubuntu 16.04] | Ubuntu 18.04 | Ubuntu 20.04 | CentOS 7 | CentOS 8 | RHEL 7&8

PYTHON [✓ Python 3.7] | Python 3.8

CUDA [✓ CUDA 10.1.2] | CUDA 10.2 | CUDA 11.0

NOTE: Ubuntu 16.04/18.04/20.04 & CentOS 7/8 use the same `conda create` commands.

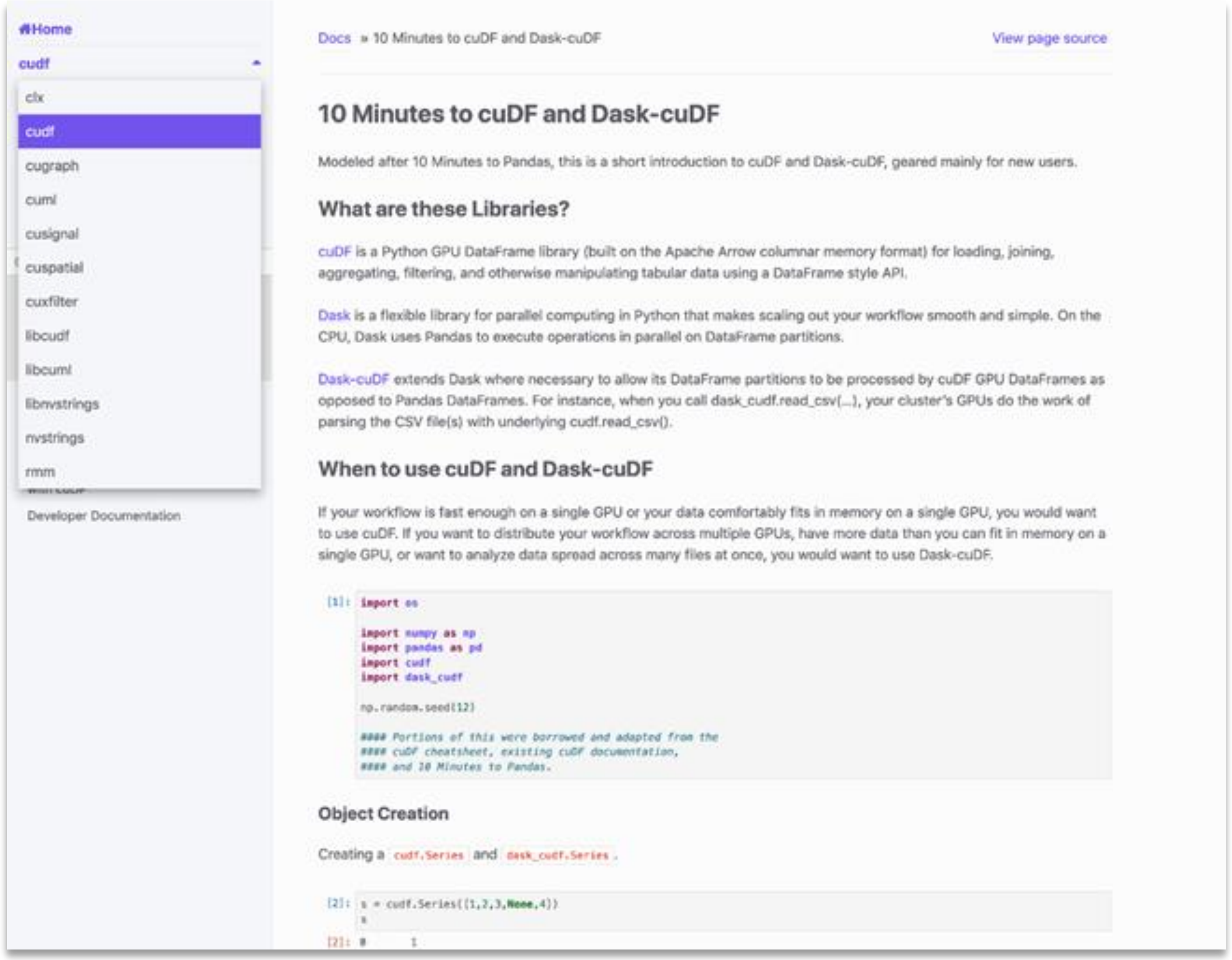
COMMAND

```
conda create -n rapids-0.18 -c rapidsai -c nvidia -c conda-forge \
    -c defaults rapids-blazing=0.18 python=3.7 cudatoolkit=10.1
```

https://rapids.ai/start.html

RAPIDS Docs

Up to Date and Easy to Use



10 Minutes to cuDF and Dask-cuDF

Modeled after 10 Minutes to Pandas, this is a short introduction to cuDF and Dask-cuDF, geared mainly for new users.

What are these Libraries?

cuDF is a Python GPU DataFrame library (built on the Apache Arrow columnar memory format) for loading, joining, aggregating, filtering, and otherwise manipulating tabular data using a DataFrame style API.

Dask is a flexible library for parallel computing in Python that makes scaling out your workflow smooth and simple. On the CPU, Dask uses Pandas to execute operations in parallel on DataFrame partitions.

Dask-cuDF extends Dask where necessary to allow its DataFrame partitions to be processed by cuDF GPU DataFrames as opposed to Pandas DataFrames. For instance, when you call dask_cudf.read_csv(...), your cluster's GPUs do the work of parsing the CSV file(s) with underlying cudf.read_csv().

When to use cuDF and Dask-cuDF

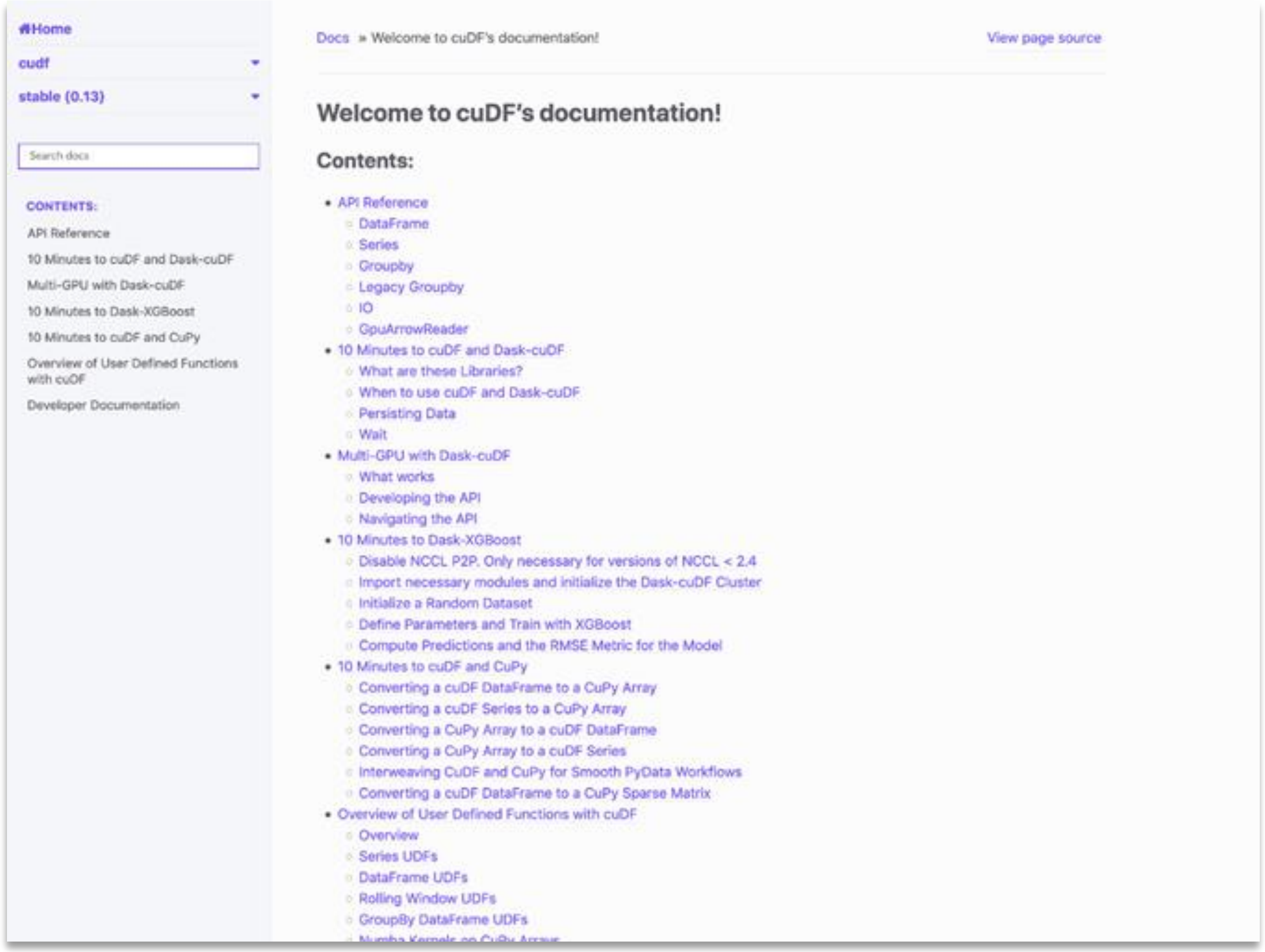
If your workflow is fast enough on a single GPU or your data comfortably fits in memory on a single GPU, you would want to use cuDF. If you want to distribute your workflow across multiple GPUs, have more data than you can fit in memory on a single GPU, or want to analyze data spread across many files at once, you would want to use Dask-cuDF.

```
[1]: import os

     import numpy as np
     import pandas as pd
     import cudf
     import dask_cudf

     np.random.seed(12)

     #### Portions of this were borrowed and adapted from the
     #### cuDF cheatsheet, existing cuDF documentation,
     #### and 10 Minutes to Pandas.
```

Object Creation

Creating a `cudf.Series` and `dask_cudf.Series`.

```
[2]: s = cudf.Series([1,2,3,None,4])
     s
[2]: 0      1
```


Home
cudf
stable (0.13)
Search docs
CONTENTS:
API Reference
10 Minutes to cuDF and Dask-cuDF
Multi-GPU with Dask-cuDF
10 Minutes to Dask-XGBoost
10 Minutes to cuDF and CuPy
Overview of User Defined Functions with cuDF
Developer Documentation
Docs » Welcome to cuDF's documentation!
View page source

Welcome to cuDF's documentation!

Contents:

- API Reference
  - DataFrame
  - Series
  - Groupby
  - Legacy Groupby
  - IO
  - GpuArrowReader
- 10 Minutes to cuDF and Dask-cuDF
  - What are these Libraries?
  - When to use cuDF and Dask-cuDF
  - Persisting Data
  - Wait
- Multi-GPU with Dask-cuDF
  - What works
  - Developing the API
  - Navigating the API
- 10 Minutes to Dask-XGBoost
  - Disable NCCL P2P. Only necessary for versions of NCCL < 2.4
  - Import necessary modules and initialize the Dask-cuDF Cluster
  - Initialize a Random Dataset
  - Define Parameters and Train with XGBoost
  - Compute Predictions and the RMSE Metric for the Model
- 10 Minutes to cuDF and CuPy
  - Converting a cuDF DataFrame to a CuPy Array
  - Converting a cuDF Series to a CuPy Array
  - Converting a CuPy Array to a cuDF DataFrame
  - Converting a CuPy Array to a cuDF Series
  - Interweaving CuDF and CuPy for Smooth PyData Workflows
  - Converting a cuDF DataFrame to a CuPy Sparse Matrix
- Overview of User Defined Functions with cuDF
  - Overview
  - Series UDFs
  - DataFrame UDFs
  - Rolling Window UDFs
  - GroupBy DataFrame UDFs
  - Numba Kernels on CuPy Arrays

https://docs.rapids.ai