

# RAPIDS

The Platform Inside and Out  
Release 0.18



@RAPIDSai



<https://github.com/rapidsai>



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

**RAPIDS**

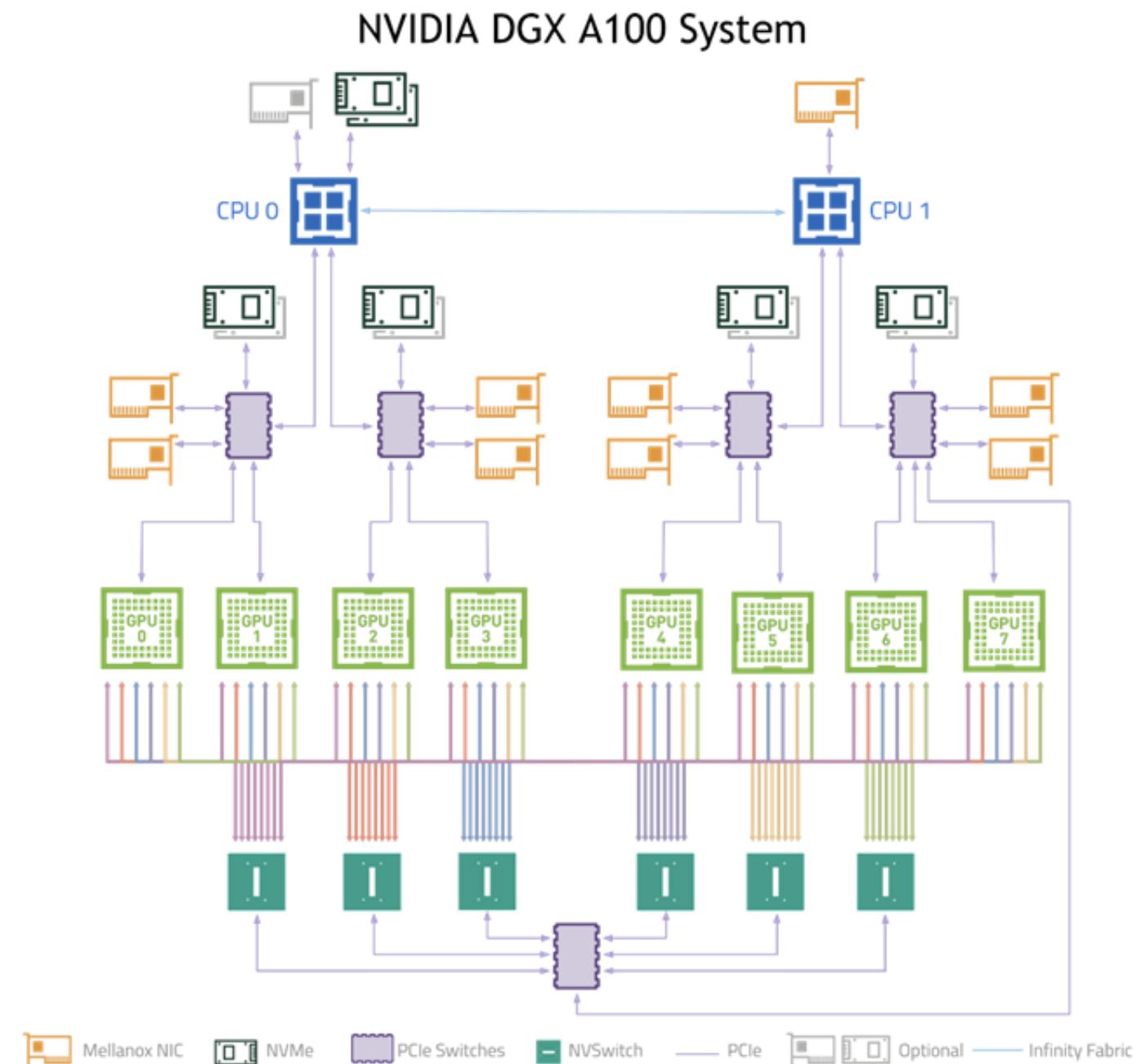
<https://rapids.ai>

# Why GPUs?

## Numerous hardware advantages

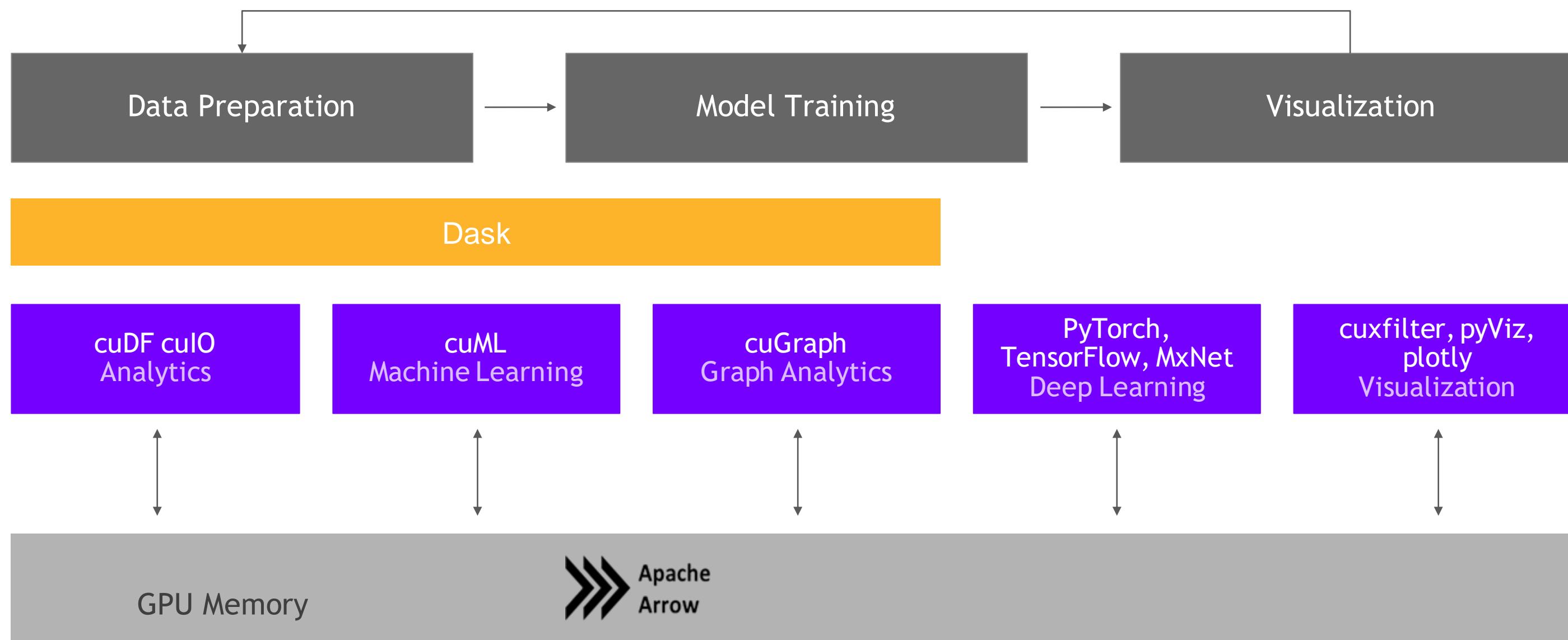
- ▶ Thousands of cores with up to ~20 TeraFlops of general purpose compute performance
- ▶ Up to 1.5 TB/s of memory bandwidth
- ▶ Hardware interconnects for up to 600 GB/s bidirectional GPU <--> GPU bandwidth
- ▶ Can scale up to 16x GPUs in a single node

**Almost never run out of compute relative to memory bandwidth!**



# RAPIDS

## End-to-End GPU Accelerated Data Science



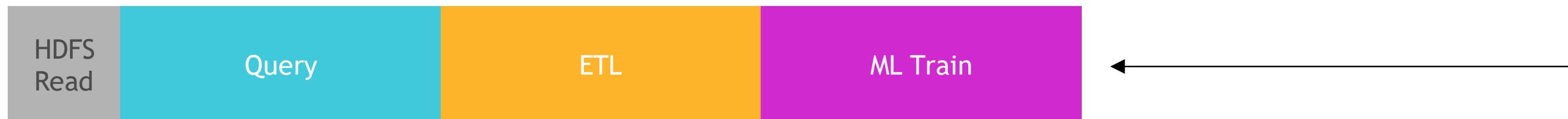
# Data Processing Evolution

## Faster Data Access, Less Data Movement

Hadoop Processing, Reading from Disk



Spark In-Memory Processing



Traditional GPU Processing

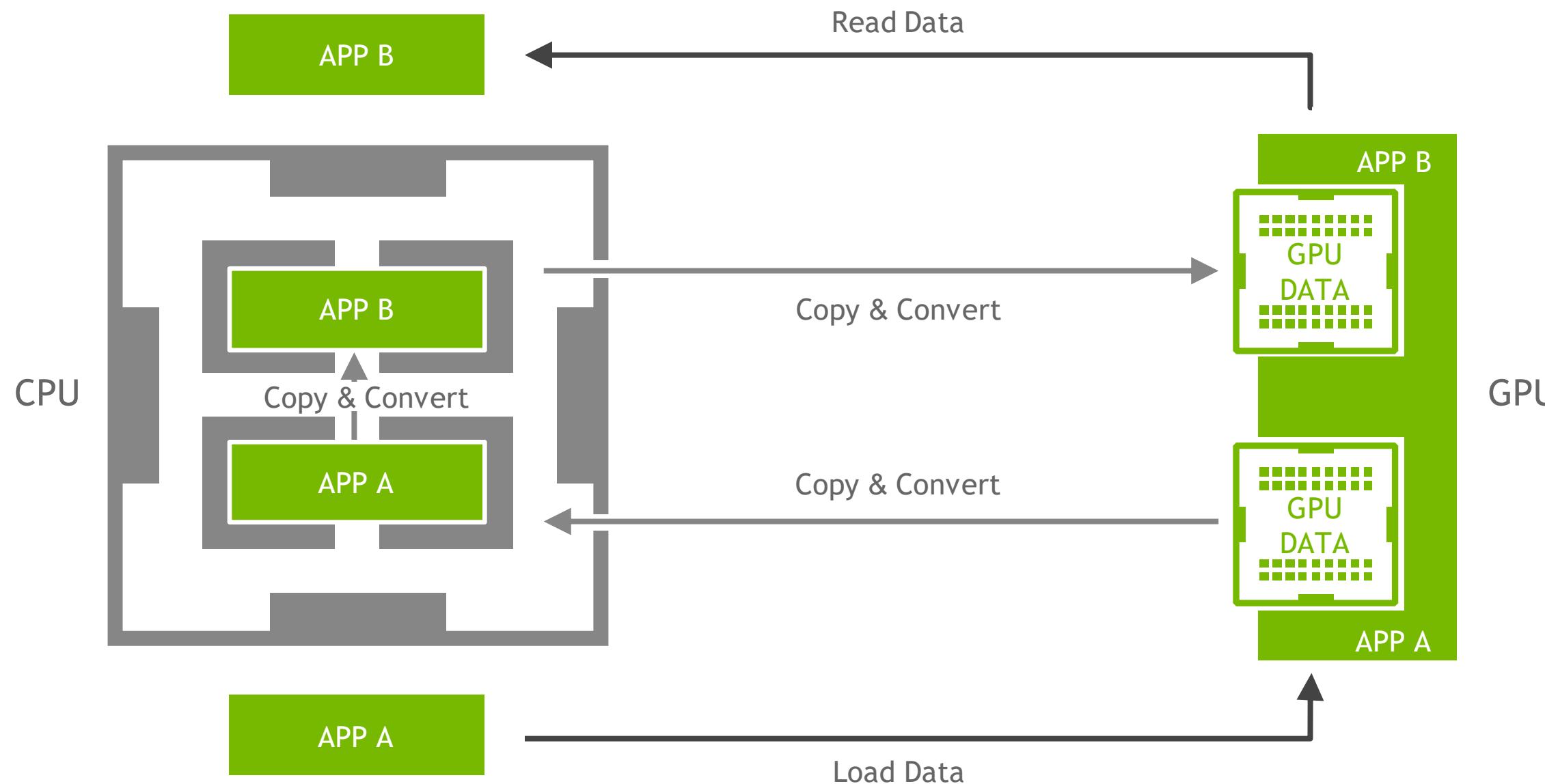


25-100x Improvement  
Less Code  
Language Flexible  
Primarily In-Memory

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

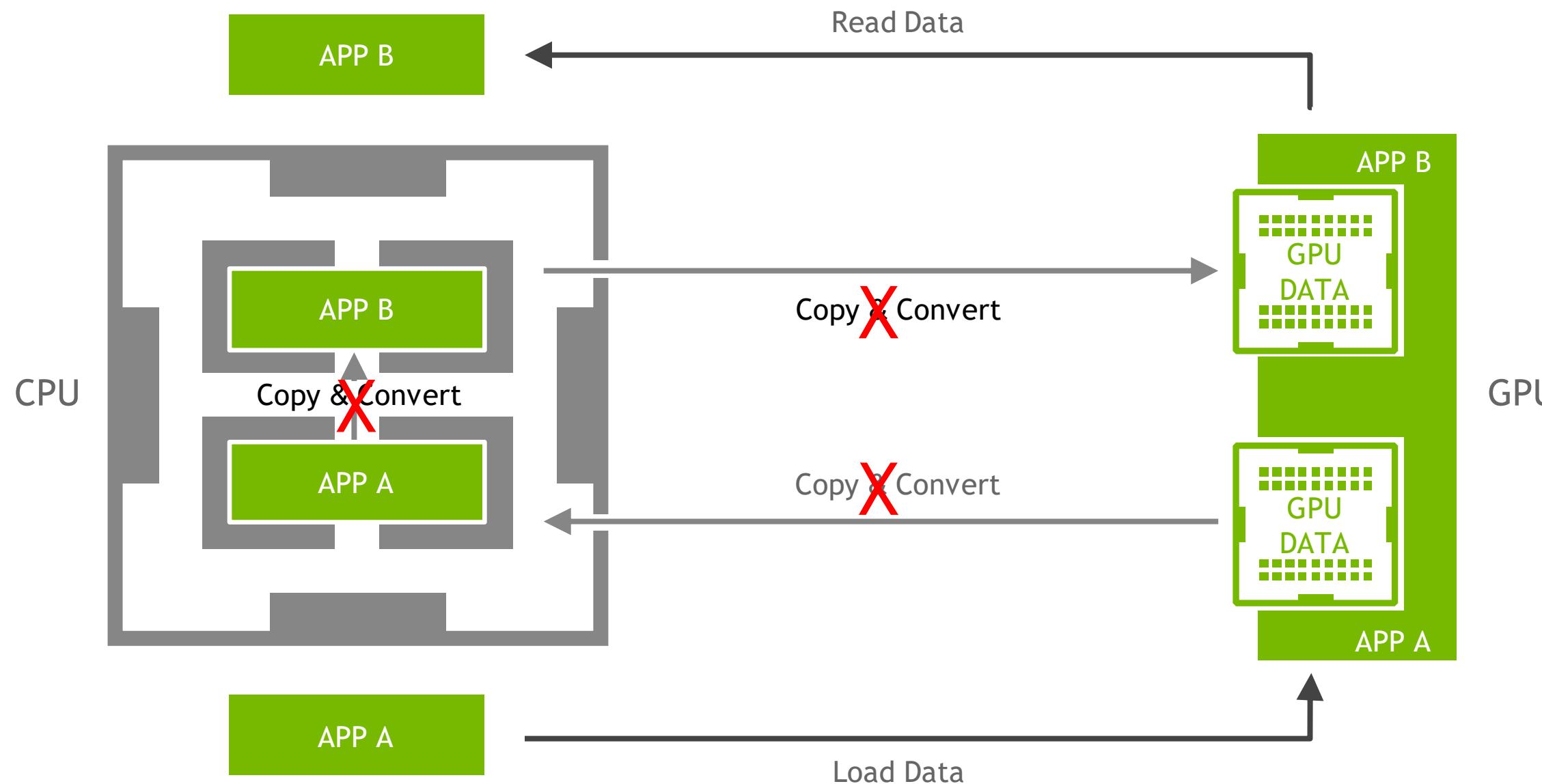
# Data Movement and Transformation

The Bane of Productivity and Performance

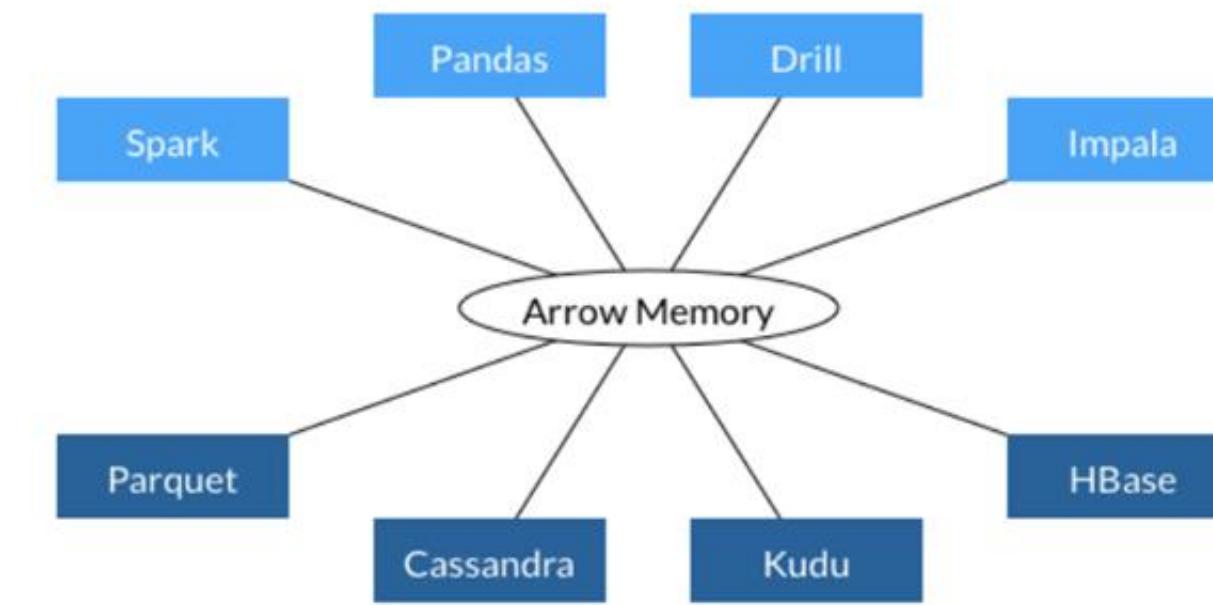
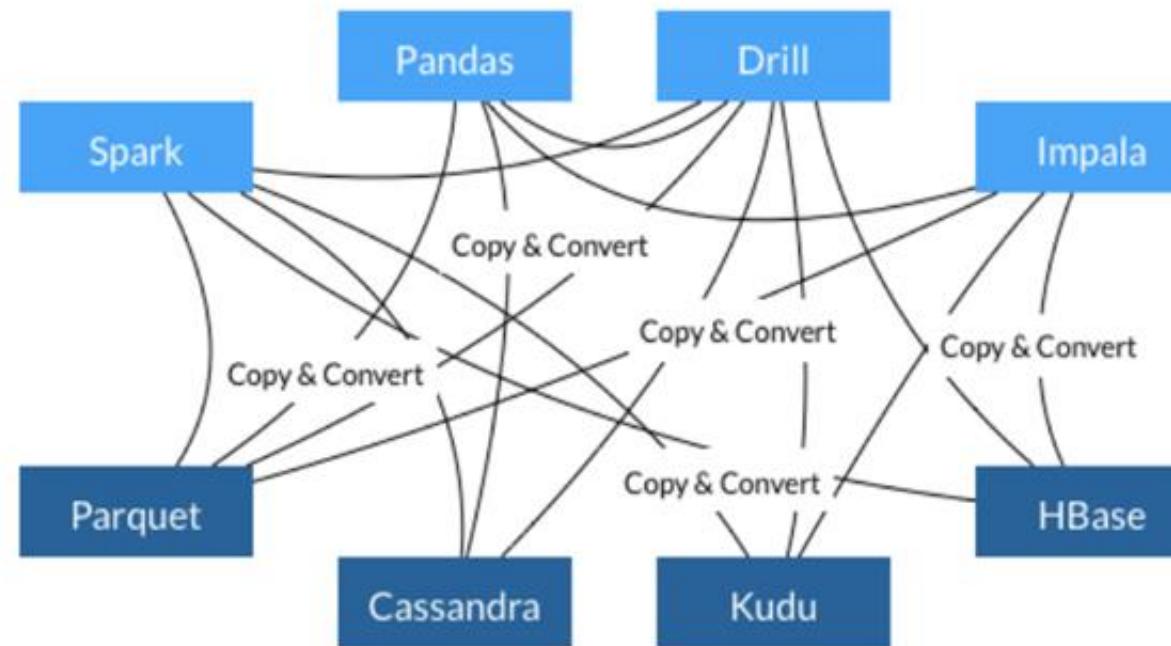


# Data Movement and Transformation

## What if We Could Keep Data on the GPU?



# Learning from Apache Arrow ➤



- ▶ Each system has its own internal memory format
- ▶ 70-80% computation wasted on serialization and deserialization
- ▶ Similar functionality implemented in multiple projects

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

Source: From Apache Arrow Home Page - <https://arrow.apache.org/>

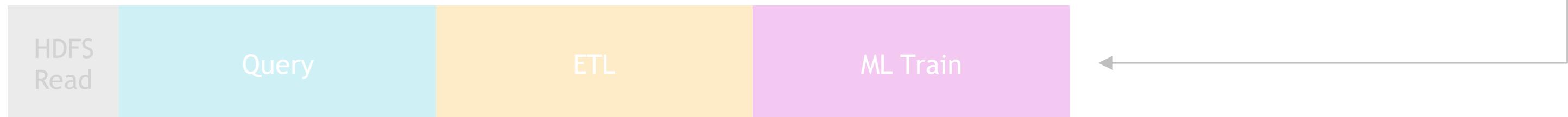
# Data Processing Evolution

## Faster Data Access, Less Data Movement

Hadoop Processing, Reading from Disk



Spark In-Memory Processing



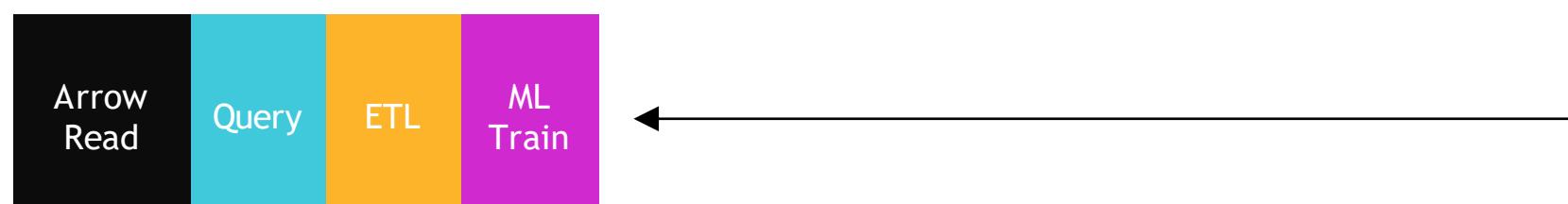
25-100x Improvement  
Less Code  
Language Flexible  
Primarily In-Memory

Traditional GPU Processing



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

RAPIDS



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

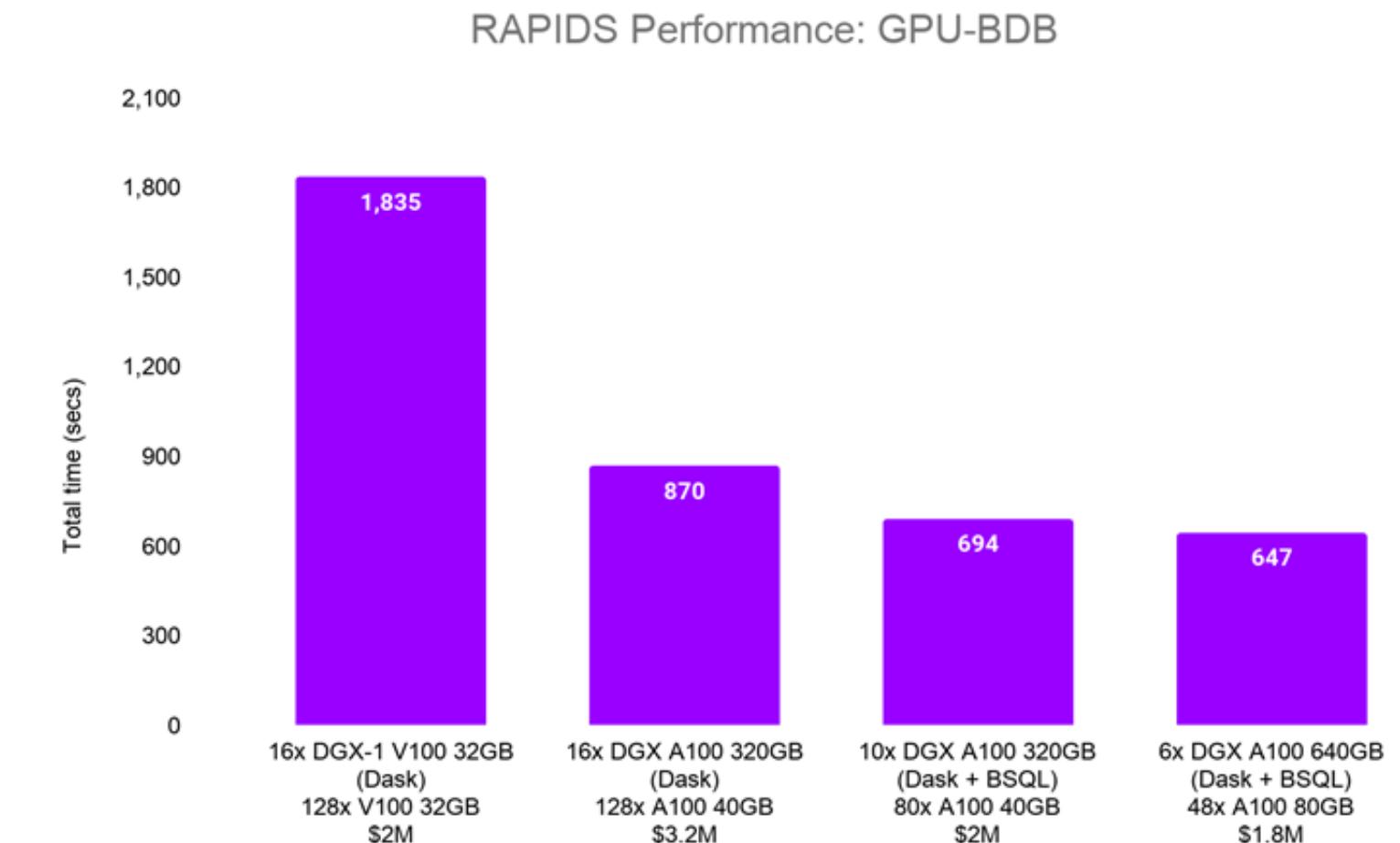
# Lightning-fast performance on real-world use cases

GPU Big Data Benchmark (GPU-BDB) is a data science benchmark derived from TPCx-BB<sup>1</sup>, consisting of 30 end-to-end queries representing real-world ETL and Machine Learning workflows. It involves both structured and unstructured data. The benchmark starts with reading data from disk, performs common analytical and ML techniques (including NLP), then writes results back to disk to simulate a real world workflow.

Results at 10TB scale show RAPIDS' performance increasing over time, while TCO continues to go down. The recently announced DGX-A100 640GB is perfectly suited to data science workloads, and lets us do more work in almost half as many nodes as the DGX-A100 320GB (6 nodes vs 10) for even better TCO.



<sup>1</sup>: GPU-BDB is derived from the TPCx-BB benchmark and is used for internal performance testing. Results from GPU-BDB are not comparable to TPCx-BB.



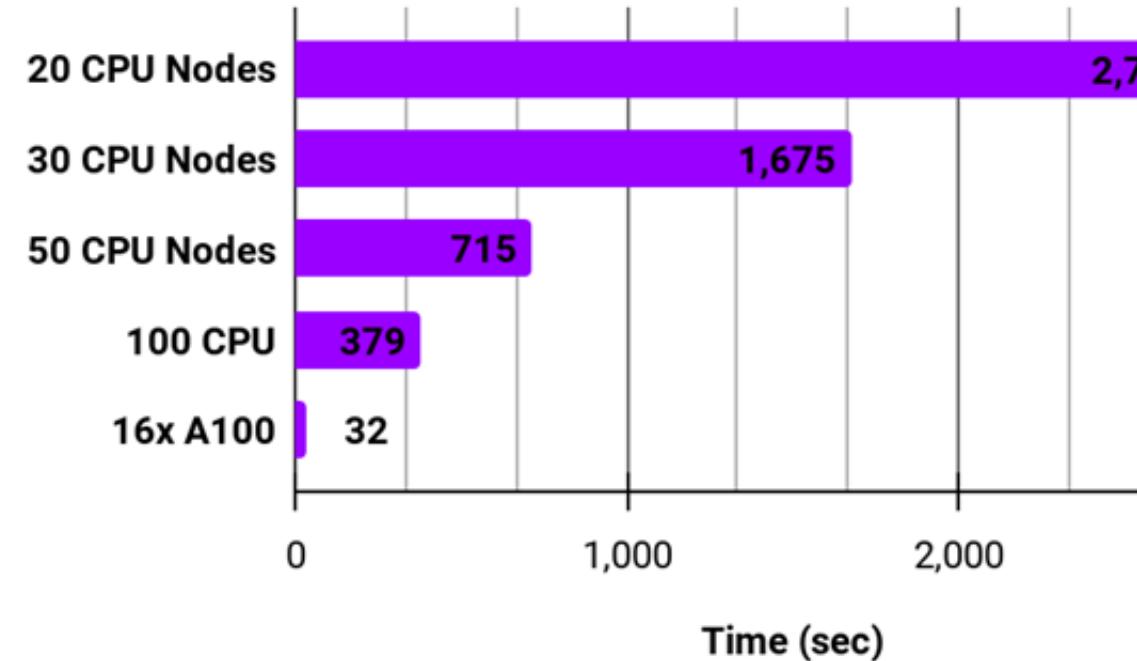
## Continuous Improvement

- 2.8x performance, almost a third the nodes, and cheaper to boot—in <1 year
- BlazingSQL at 10TB showing 25% improvement compared to Dask over TCP
- Q27 faster and more accurate with Hugging Face

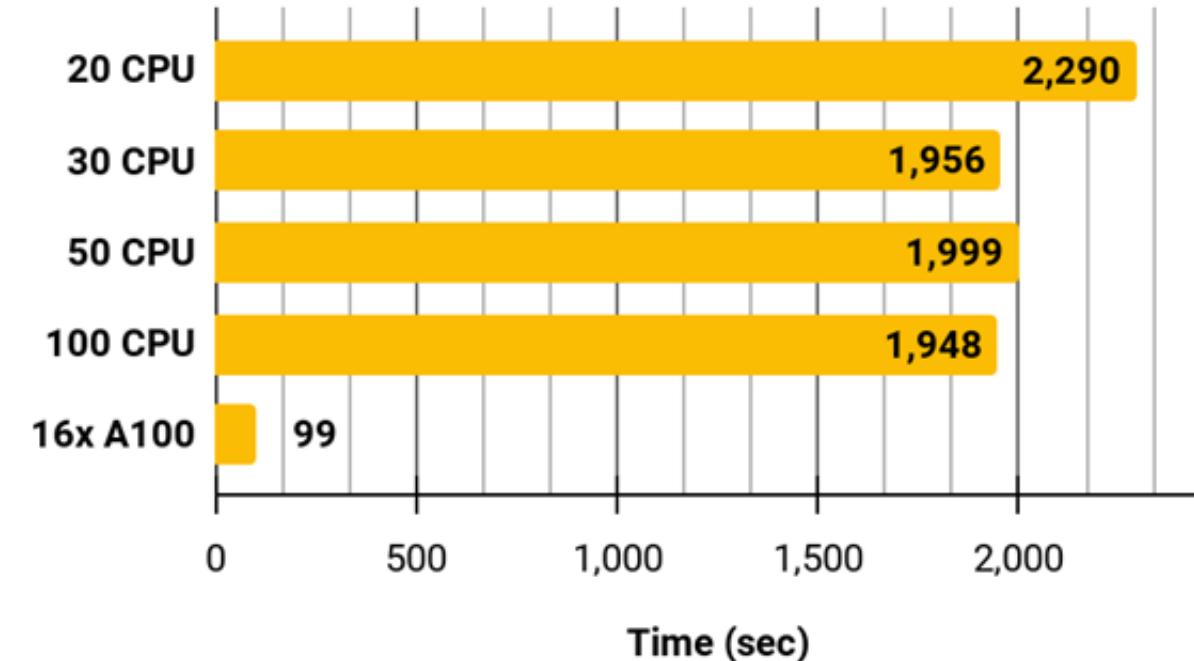
# Faster Speeds, Real World Benefits

## Faster Data Access, Less Data Movement

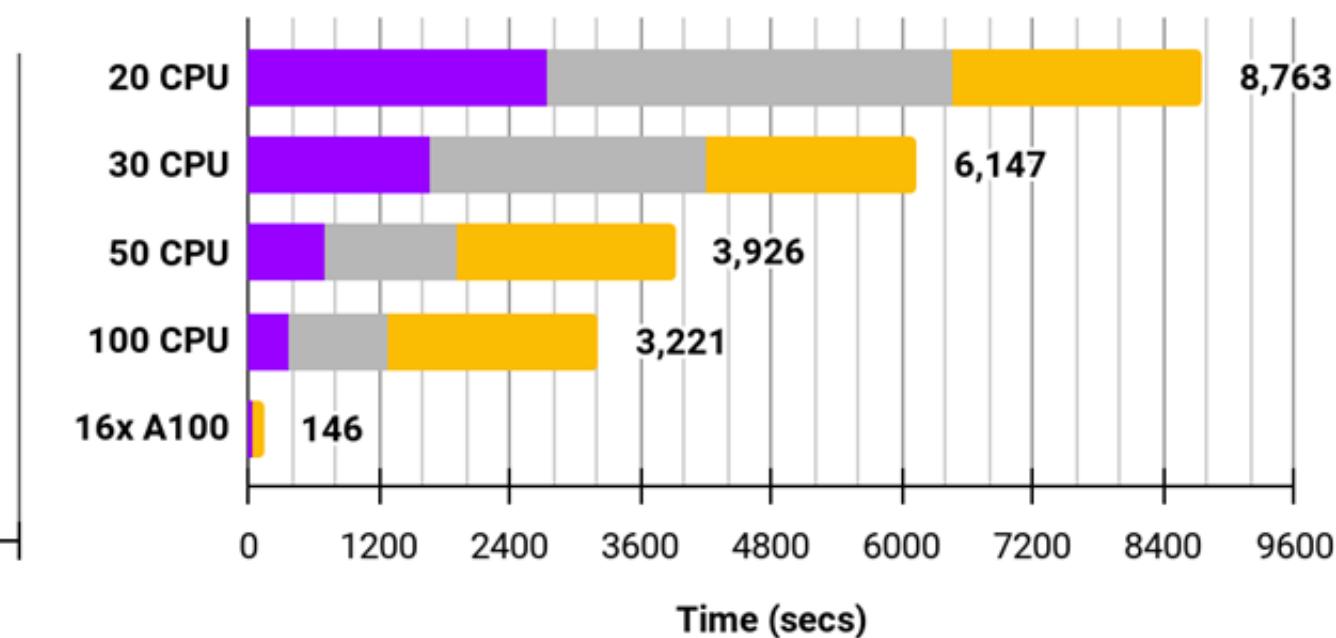
**cuIO/cuDF -  
Load and Data Preparation**



**XGBoost Machine Learning**



**End-to-End**



Time in seconds (shorter is better)

■ cuIO/cuDF (Load and Data Prep) ■ Data Conversion ■ XGBoost

**Benchmark**

200GB CSV dataset; Data prep includes joins, variable transformations

**CPU Cluster Configuration**

CPU nodes (61 GiB memory, 8 vCPUs, 64-bit platform), Apache Spark

**A100 Cluster Configuration**

16 A100 GPUs (40GB each)

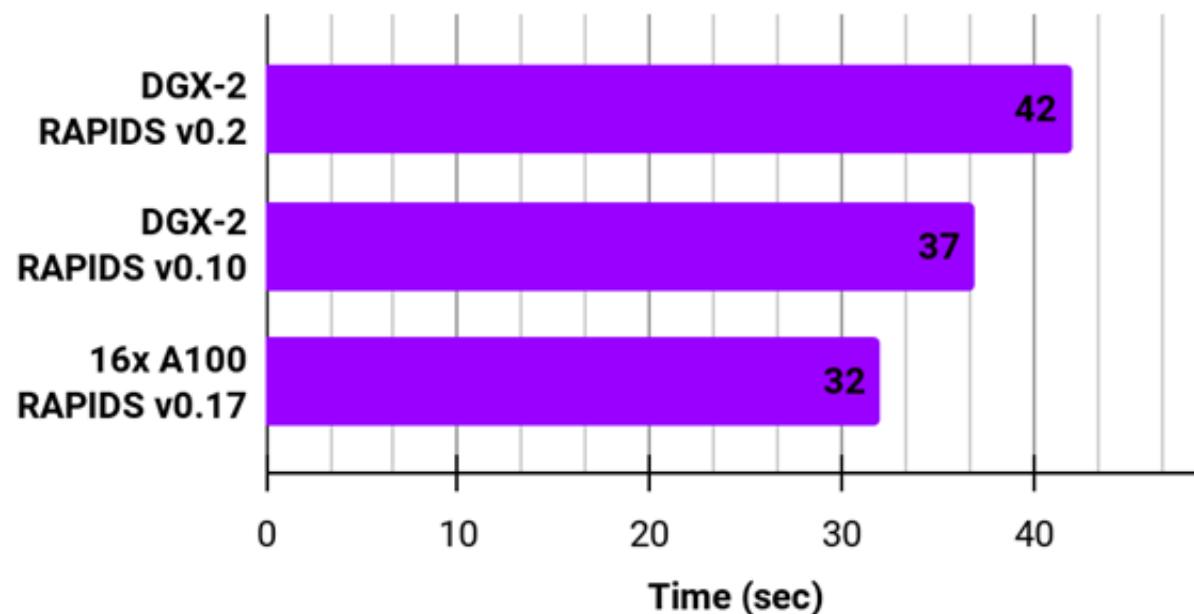
**RAPIDS Version**

RAPIDS 0.17

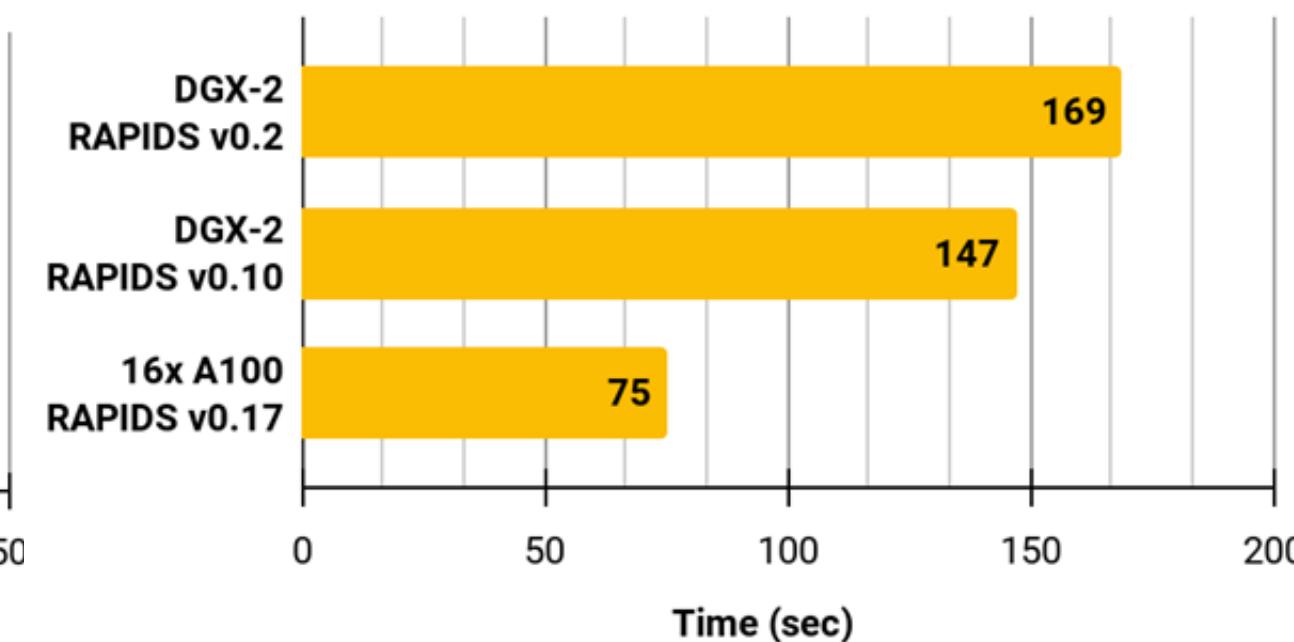
# Faster Speeds, Real World Benefits

Getting faster over time--and even better with A100

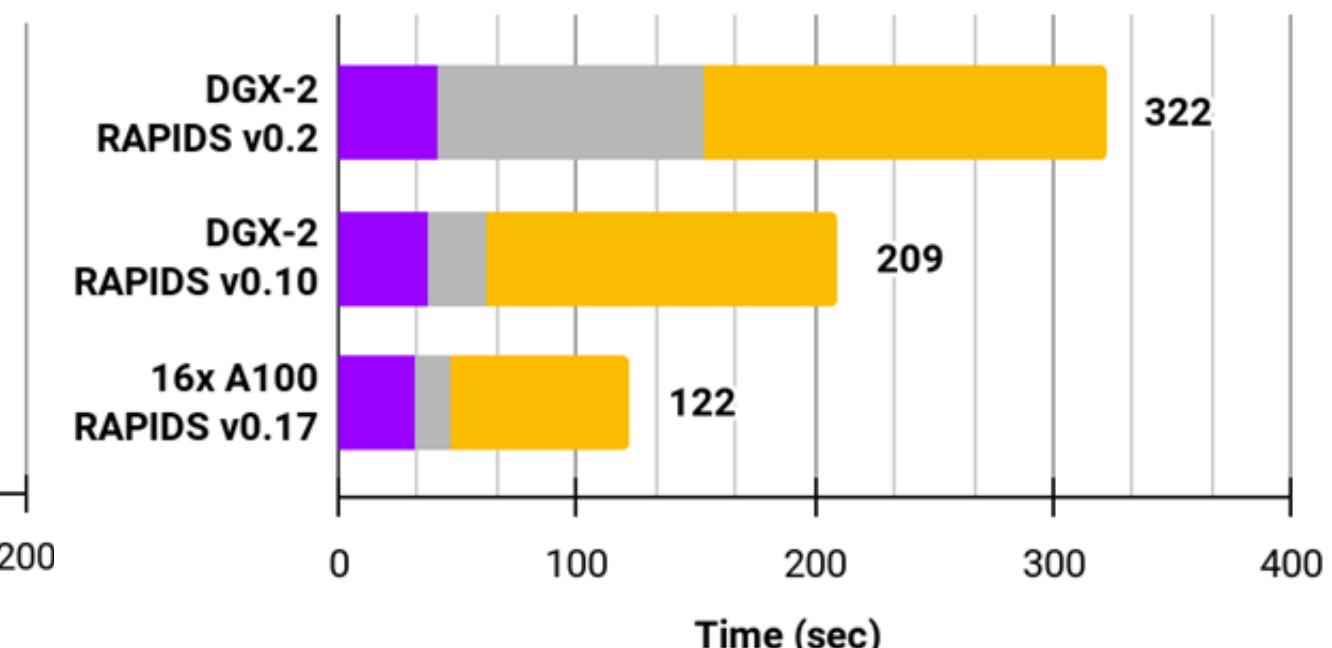
**cuIO/cuDF -  
Load and Data Preparation**



**XGBoost Machine Learning**



**End-to-End**



Time in seconds (shorter is better)

■ cuIO/cuDF (Load and Data Prep) ■ Data Conversion ■ XGBoost

## Benchmark

200GB CSV dataset; Data prep includes joins, variable transformations

## CPU Cluster Configuration

CPU nodes (61 GiB memory, 8 vCPUs, 64- 16 A100 GPUs (40GB each bit platform), Apache Spark

## A100 Cluster Configuration

# RAPIDS 0.18 Release Summary

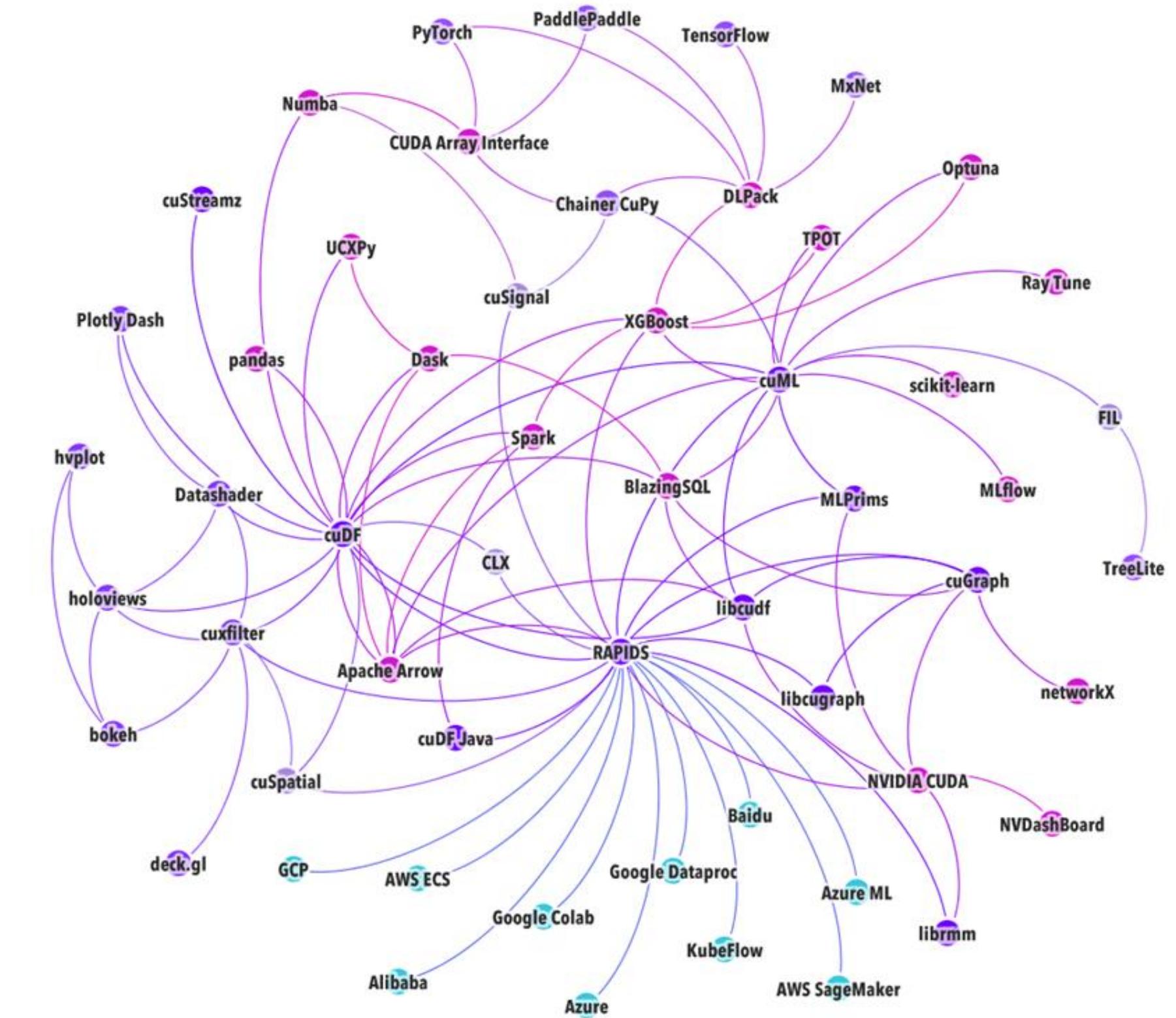
# What's New in Release 0.18?

- ▶ **cuDF** Broader support for fixed-point decimal types in C++, initial support in Python; nested types support continues to improve; more groupby and rolling window aggregations
- ▶ **cuML** Expanded SHAP explainability, much faster Random Forest - FIL conversion, multi-GPU/multi-node DBSCAN, IncrementalPCA, sparse TSNE, approximate nearest neighbors, silhouette score, and many smaller fixes
- ▶ **XGBoost** 1.3.3 ships with 0.18, including GPU-accelerated TreeSHAP explainability, and several bugfixes
- ▶ **cuGraph** new Traveling Salesman (TSP) and Egonet, enhanced BFS to have a depth limit, and continued improving graph primitives
- ▶ **UCX-Py** Focused on code cleanup and documentation improvements
- ▶ **BlazingSQL** New string functions (REPLACE, TRIM, UPPER, and others). Also a new communication layer that improves distributed performance and supports UCX.
- ▶ **cuxfilter** Newly updated and fully customizable responsive layouts, themes, and panel widgets.
- ▶ **RMM** New C++ `cuda_async_memory_resource` built on `cudaMallocAsync`; New C++ stream pool; Python stream wrapper class; Thrust interop improvements

# RAPIDS Everywhere

## The Next Phase of RAPIDS

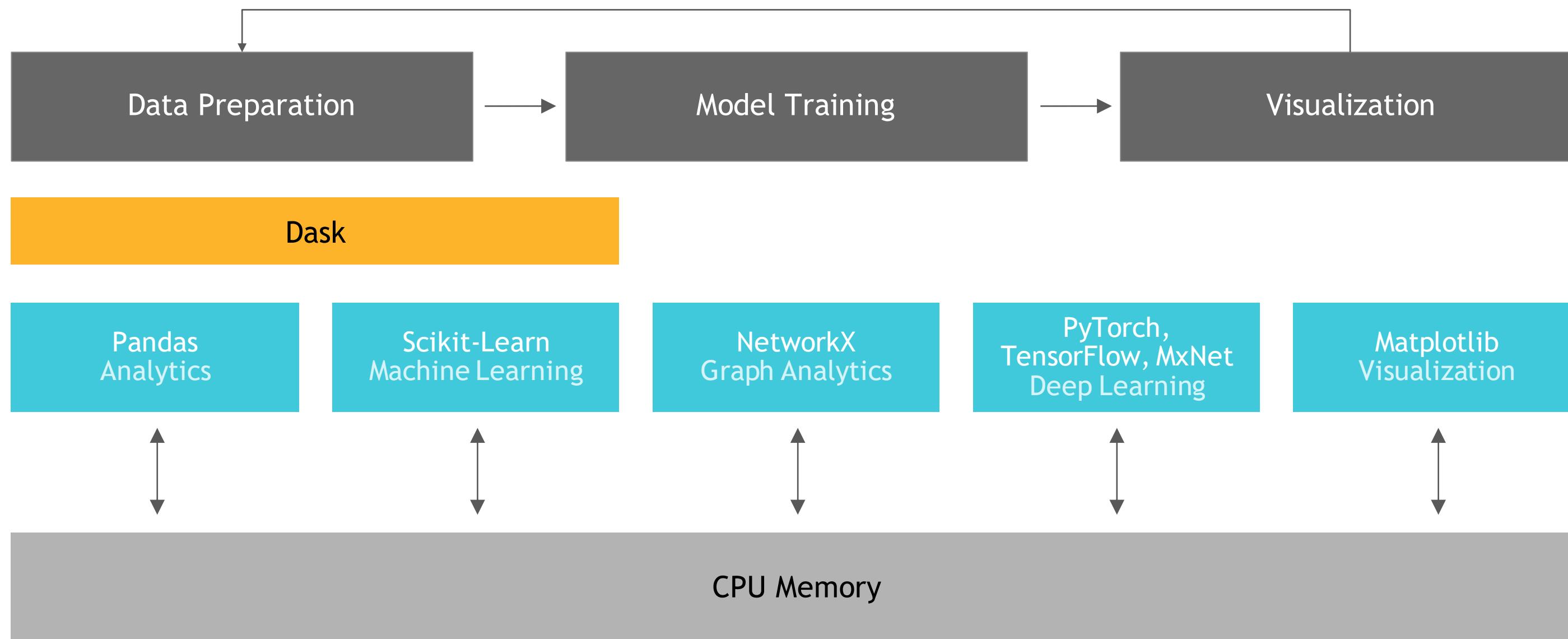
Exactly as it sounds—our goal is to make RAPIDS as usable and performant as possible wherever data science is done. We will continue to work with more open source projects to further democratize acceleration and efficiency in data science.



# RAPIDS Core

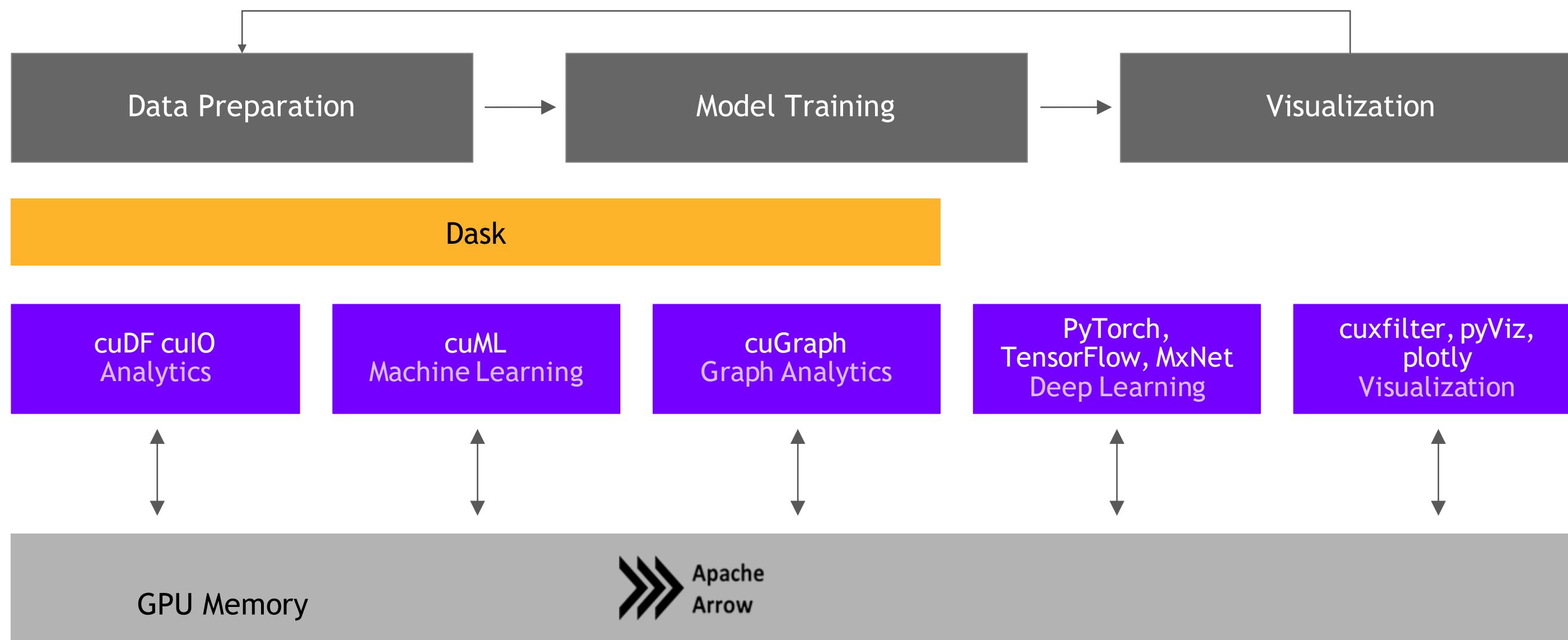
# Open Source Data Science Ecosystem

## Familiar Python APIs



# RAPIDS

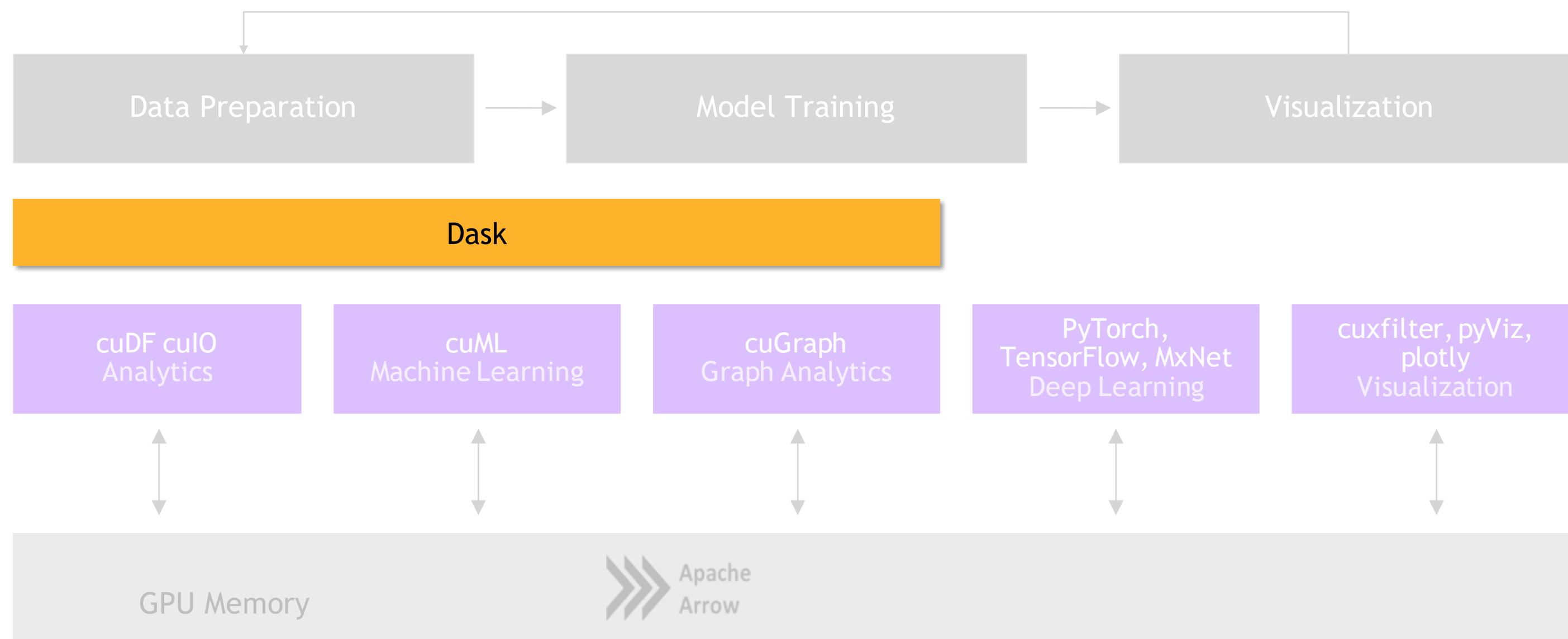
## End-to-End Accelerated GPU Data Science



# Dask

# RAPIDS

## Scaling RAPIDS with Dask



# Why Dask?

## DEPLOYABLE

- ▶ HPC: SLURM, PBS, LSF, SGE
- ▶ Cloud: Kubernetes
- ▶ Hadoop/Spark: Yarn

## EASY SCALABILITY

- ▶ Easy to install and use on a laptop
- ▶ Scales out to thousand node clusters
- ▶ Modularly built for acceleration

## PYDATA NATIVE

- ▶ Easy Migration: Built on top of NumPy, Pandas Scikit-Learn, etc
- ▶ Easy Training: With the same API

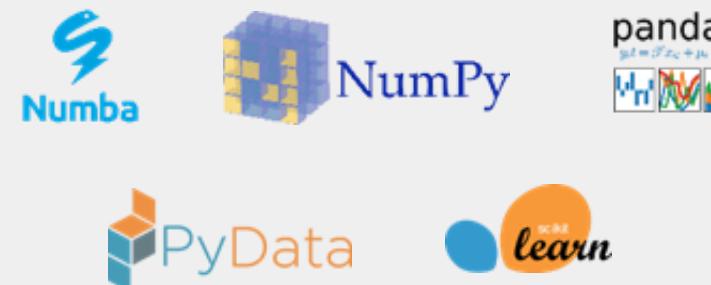
## POPULAR

- ▶ Most Common parallelism framework today in the PyData and SciPy community
- ▶ Millions of monthly Downloads and Dozens of Integrations

## 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



Scale Out / Parallelize

# Why OpenUCX?

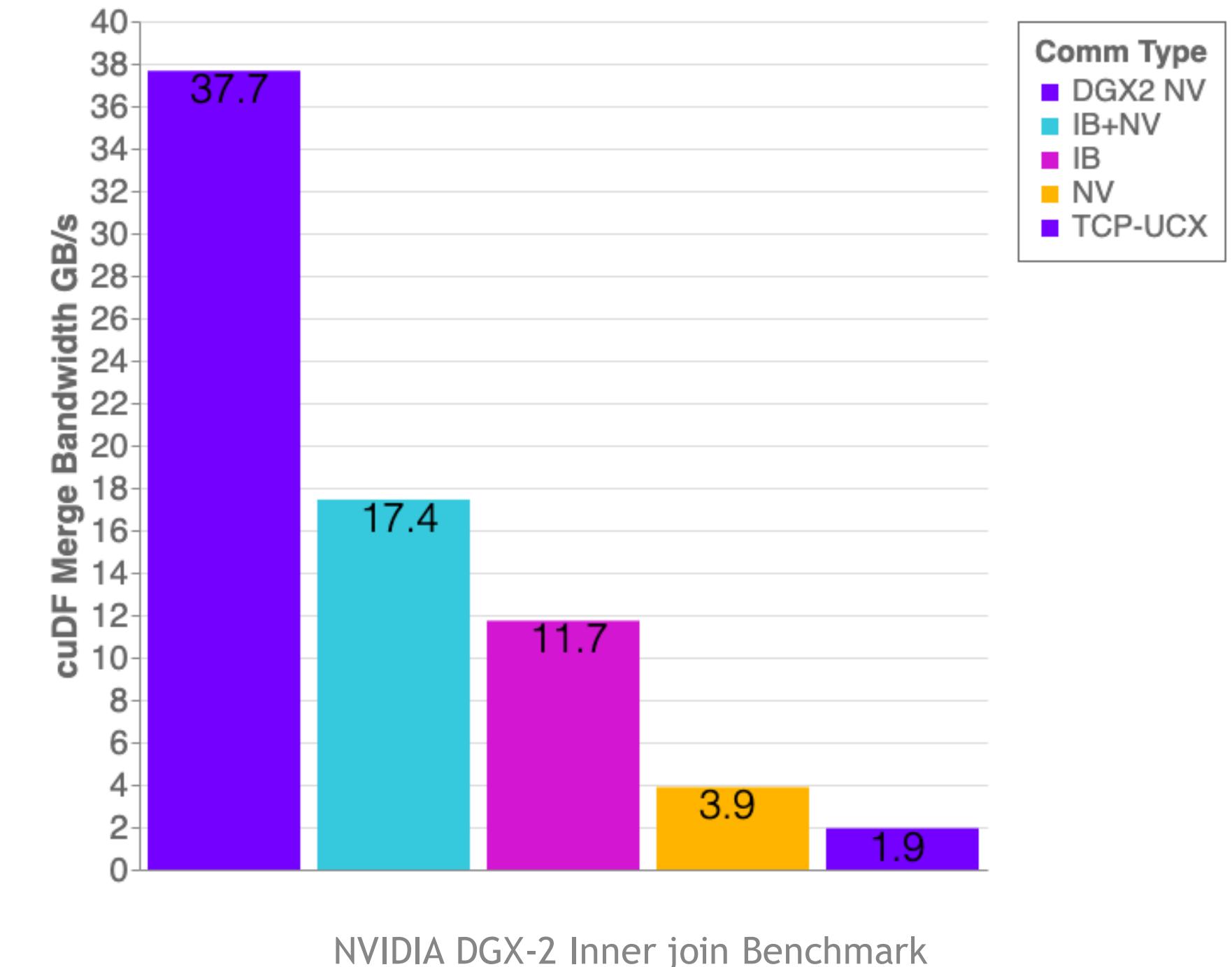
## Bringing Hardware Accelerated Communications to Dask

- ▶ TCP sockets are slow!
- ▶ UCX provides uniform access to transports (TCP, InfiniBand, shared memory, NVLink)
- ▶ Python bindings for UCX (ucx-py)
- ▶ Will provide best communication performance, to Dask based on available hardware on nodes/cluster
- ▶ Easy to use!

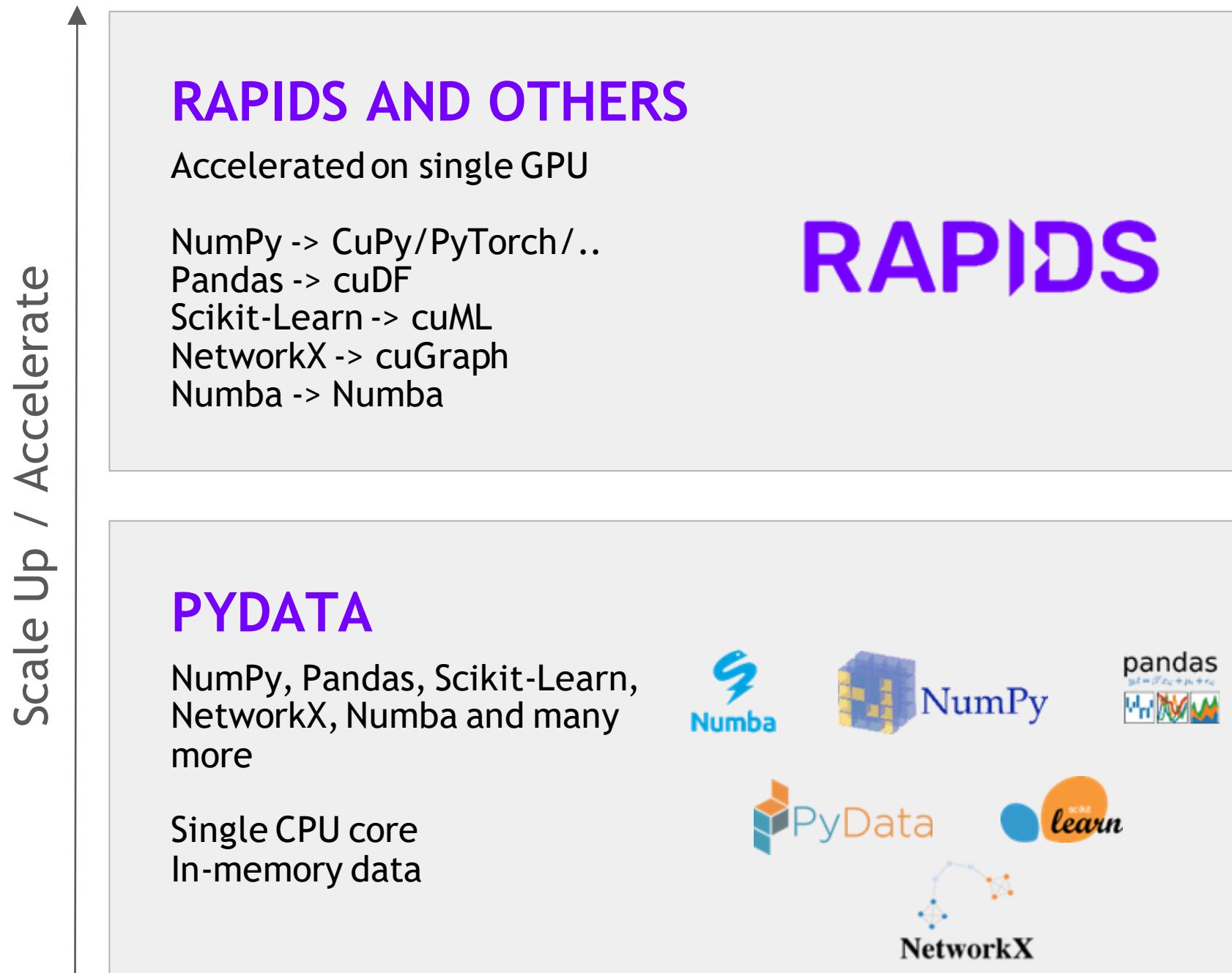
```
conda install -c conda-forge -c rapidsai \
  cudatoolkit=<CUDA version> ucx-proc=*gpu ucx ucx-py

cluster = LocalCUDACluster(protocol='ucx',
                           enable_infiniband=True,
                           enable_nvlink=True)

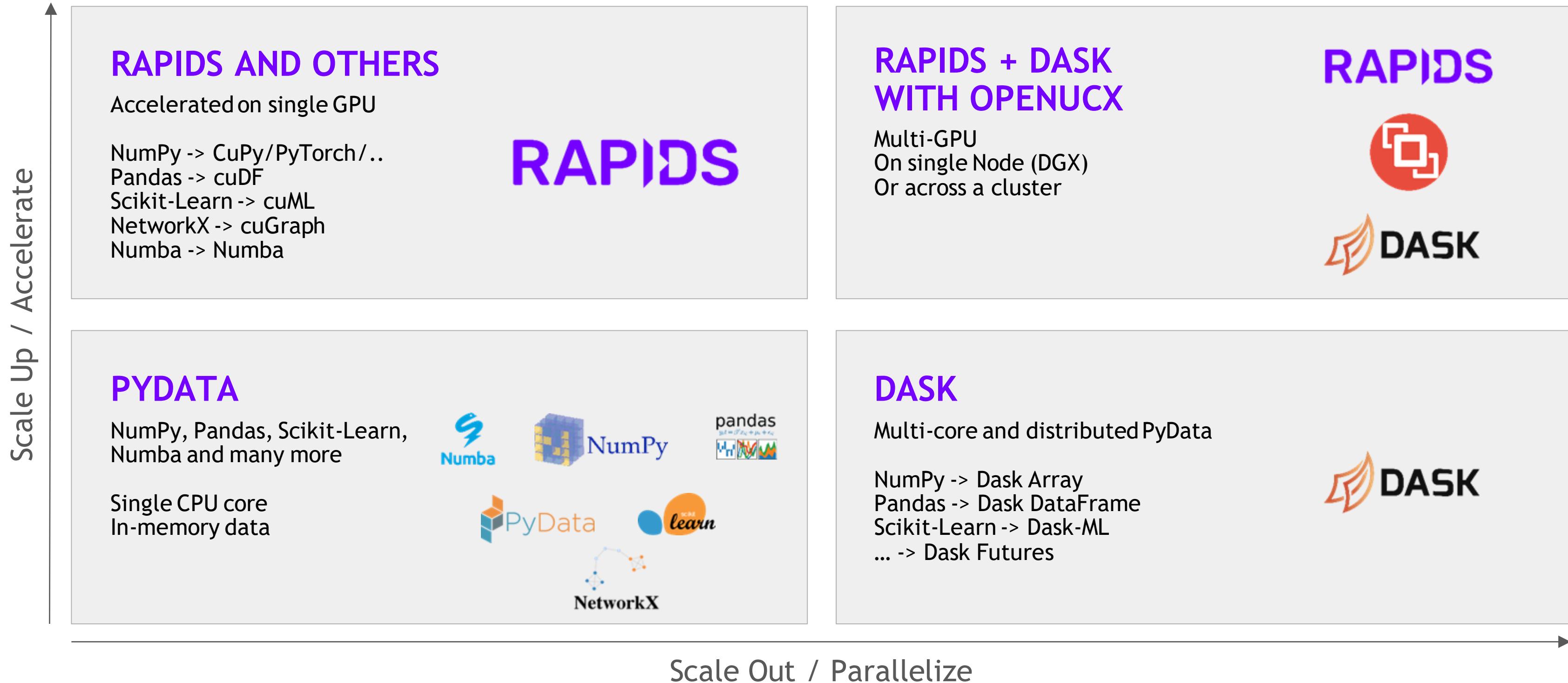
client = Client(cluster)
```



# Scale Up with RAPIDS



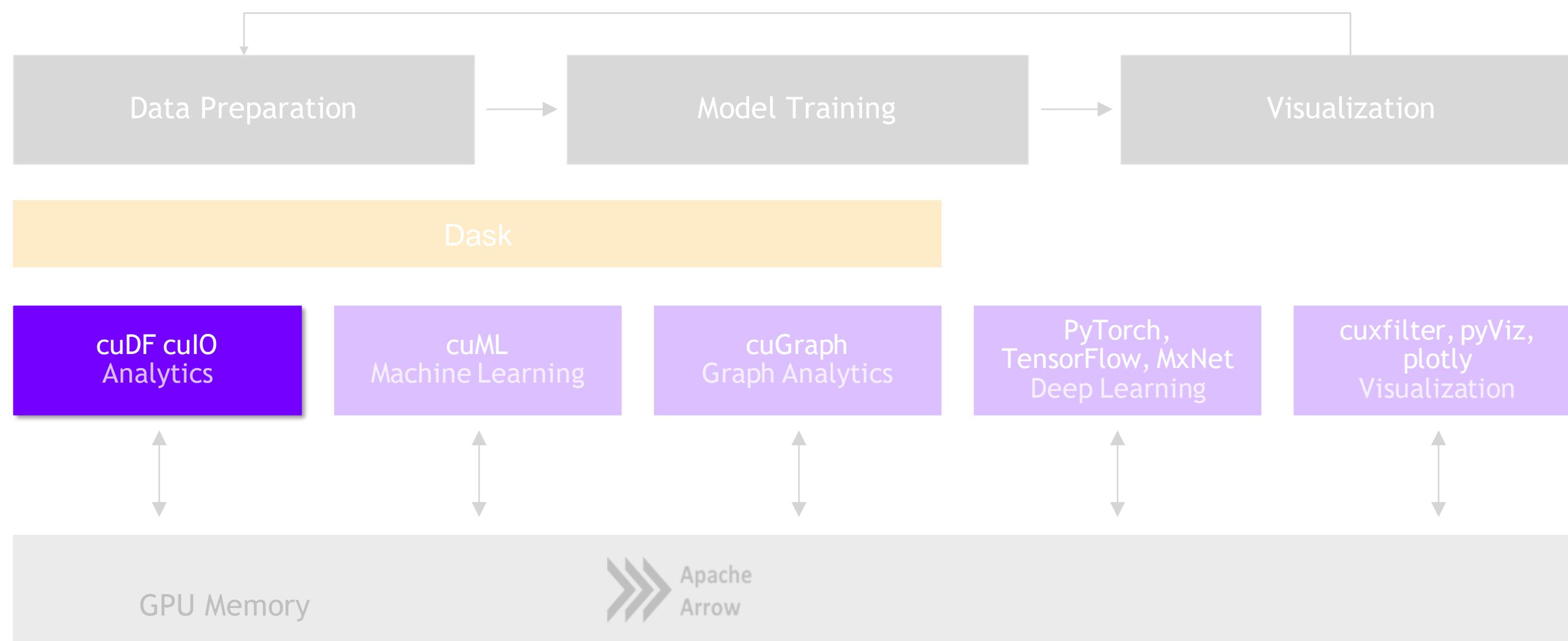
# Scale Out with RAPIDS + Dask with OpenUCX



cuDF

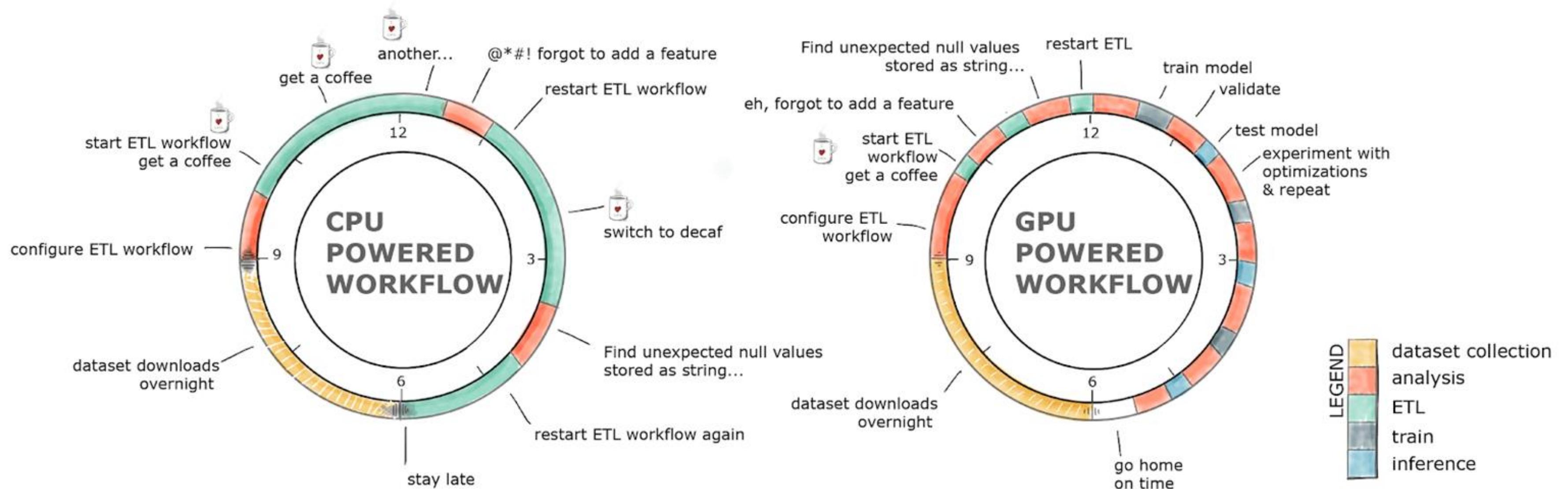
# RAPIDS

## GPU Accelerated Data Wrangling and Feature Engineering

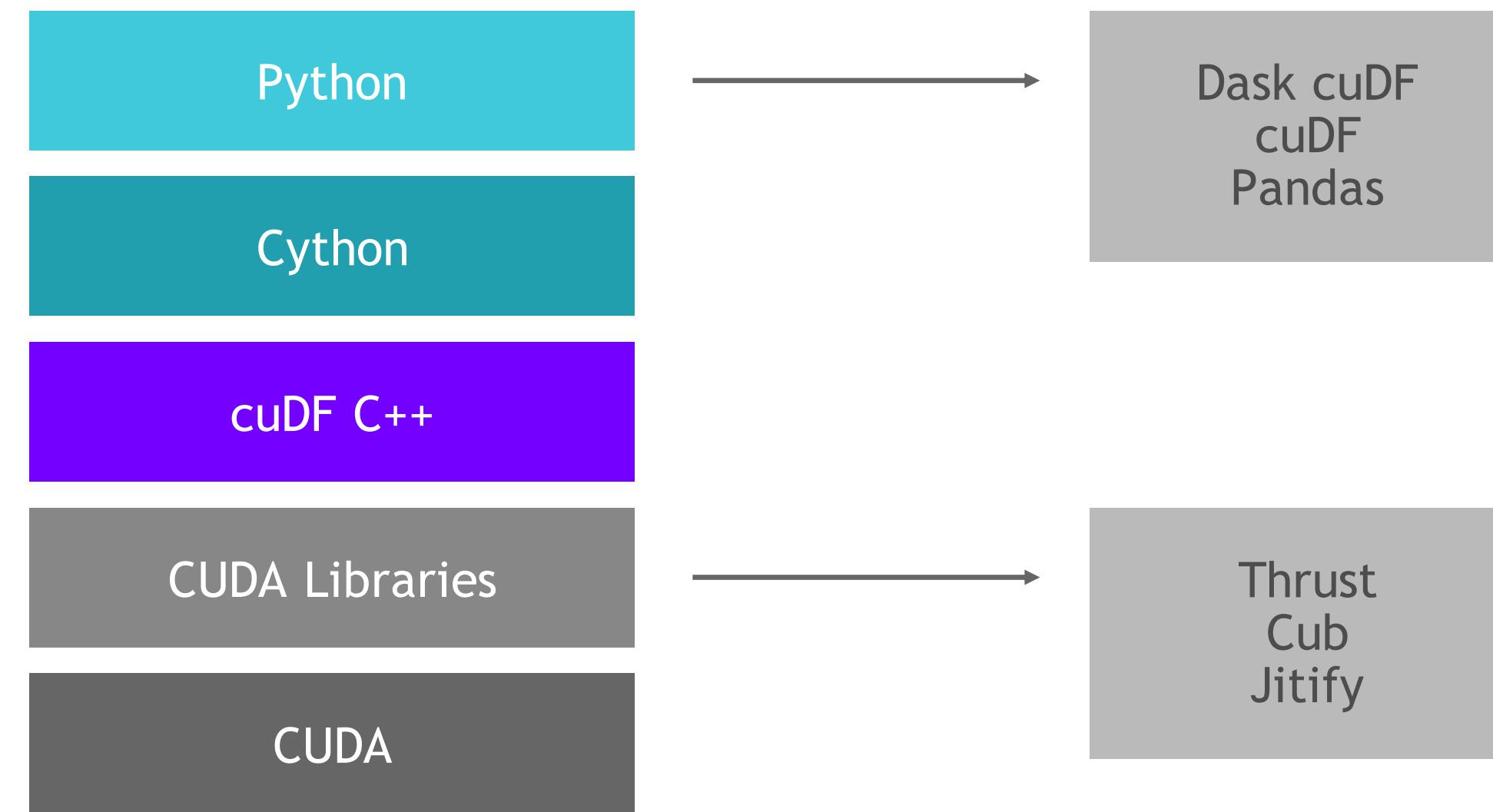


# GPU-Accelerated ETL

The Average Data Scientist Spends 90% of Their Time in ETL as Opposed to Training Models



# ETL Technology Stack



# ETL - the Backbone of Data Science

libcuDF is...

## CUDA C++ LIBRARY

- ▶ Table (dataframe) and column types and algorithms
- ▶ CUDA kernels for sorting, join, groupby, reductions, partitioning, elementwise operations, etc.
- ▶ Optimized GPU implementations for strings, timestamps, numeric types (more coming)
- ▶ Primitives for scalable distributed ETL

```
std::unique_ptr




```



# 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

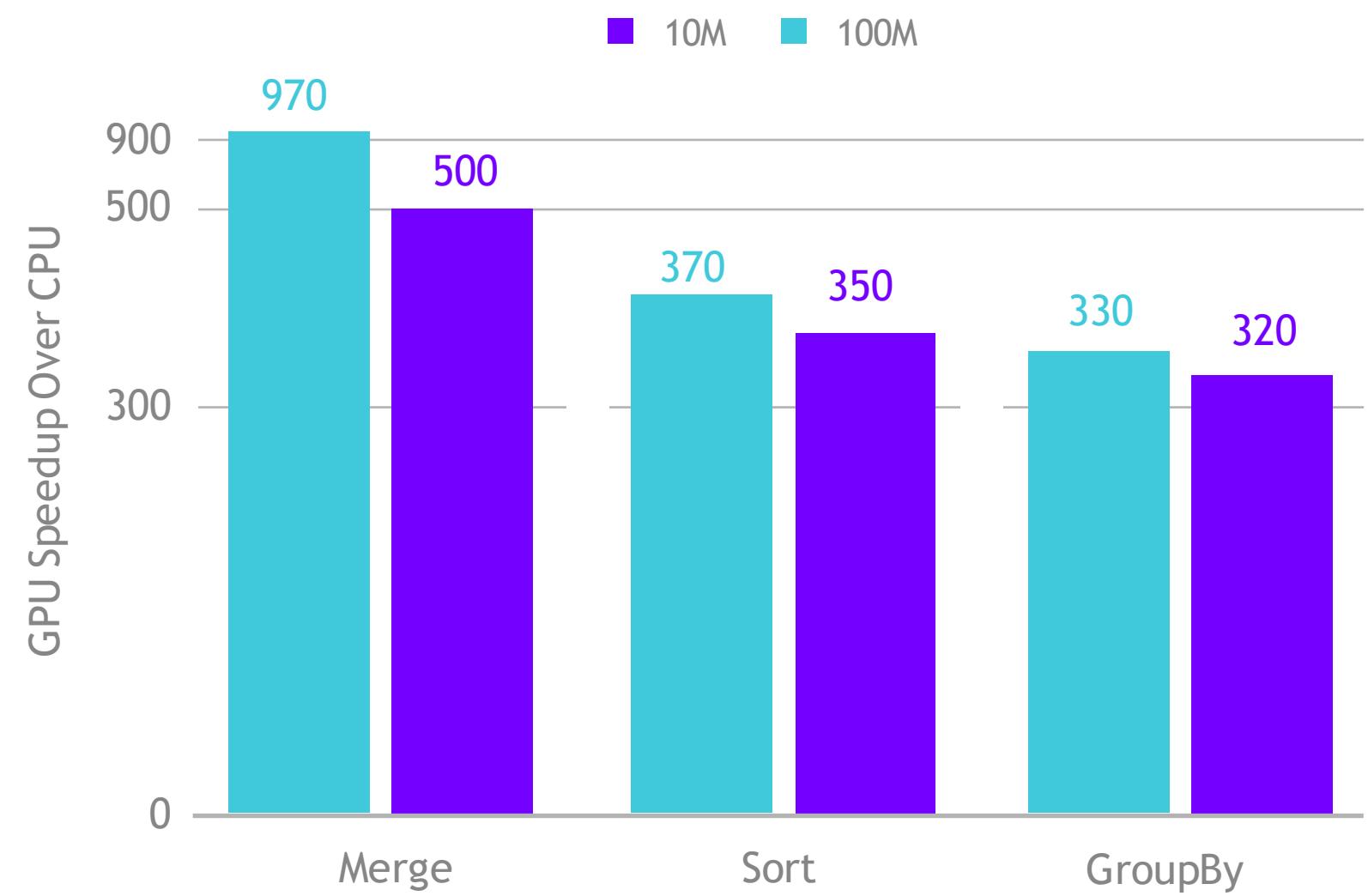
  

```
In [6]: #grabbing the first character of the years in city string to get rid of plus sign, and converting  
#to int  
gdf['city_years'] = gdf.Stay_In_Current_City_Years.str.get(0).stoi()  
  
In [7]: #Here we can see how we can control what the value of our dummies with the replace method and turn  
#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

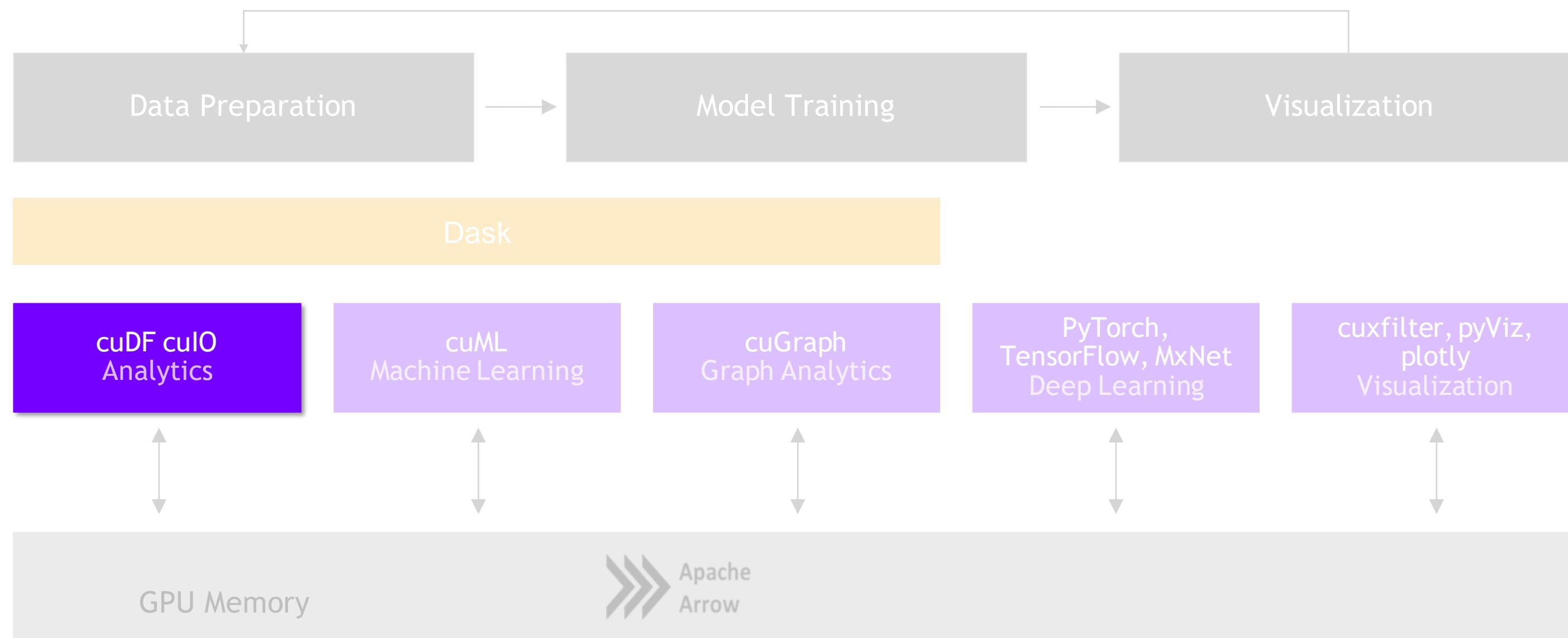
cuDF v0.13, Pandas 0.25.3

- ▶ 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



# ETL - the Backbone of Data Science

cuDF is Not the End of the Story



# ETL - the Backbone of Data Science

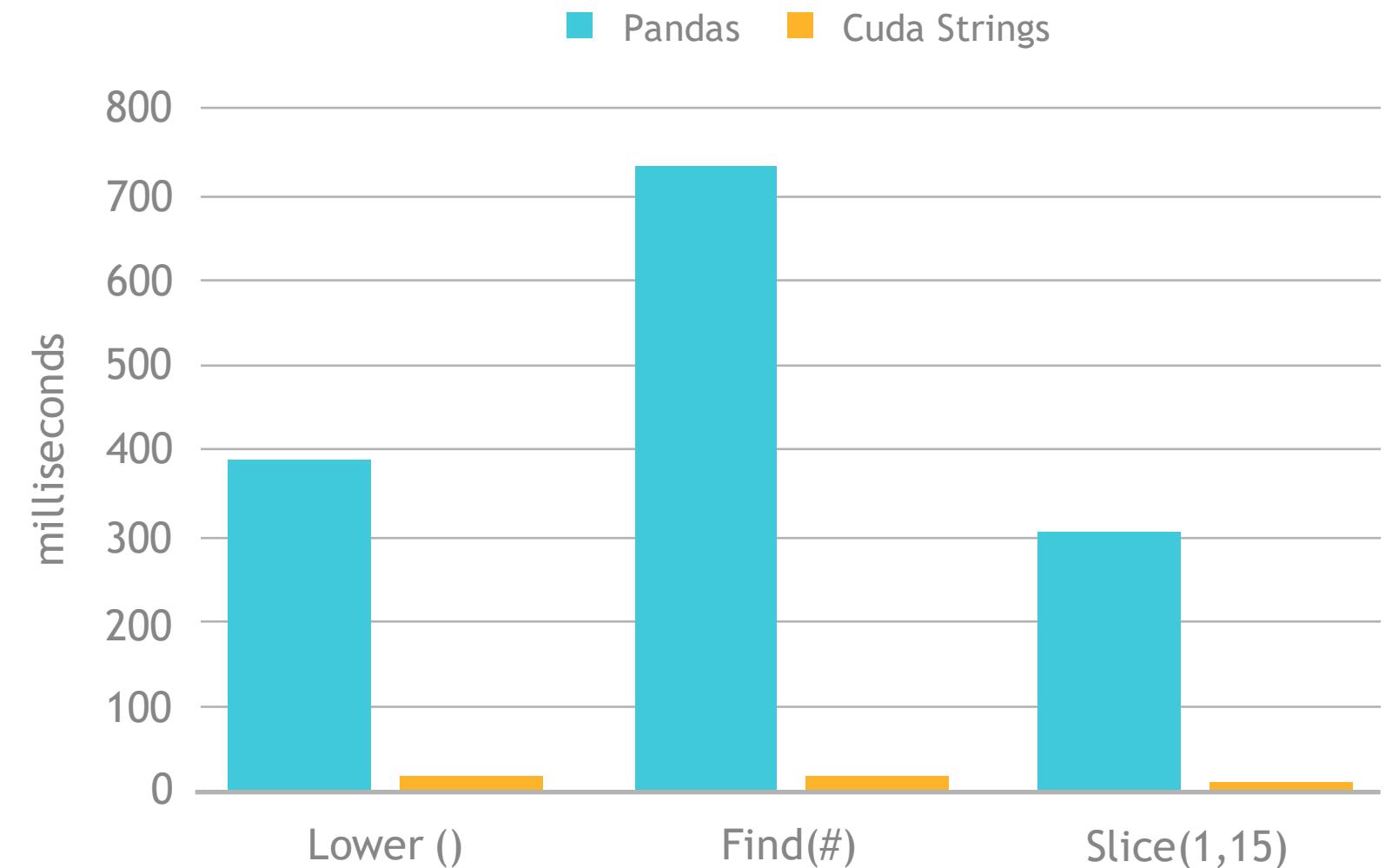
## String Support

### CURRENT V0.18 STRING SUPPORT

- ▶ Regular Expressions
- ▶ Element-wise operations
  - ▶ Split, Find, Extract, Cat, Typecasting, etc...
- ▶ String GroupBys, Joins, Sorting, etc.
- ▶ Categorical columns fully on GPU
- ▶ Native String type in libcudf C++
- ▶ NLP Preprocessors
  - ▶ Tokenizers, Normalizers, Edit Distance, Porter Stemmer, etc.

### FUTURE V0.19+ STRING SUPPORT

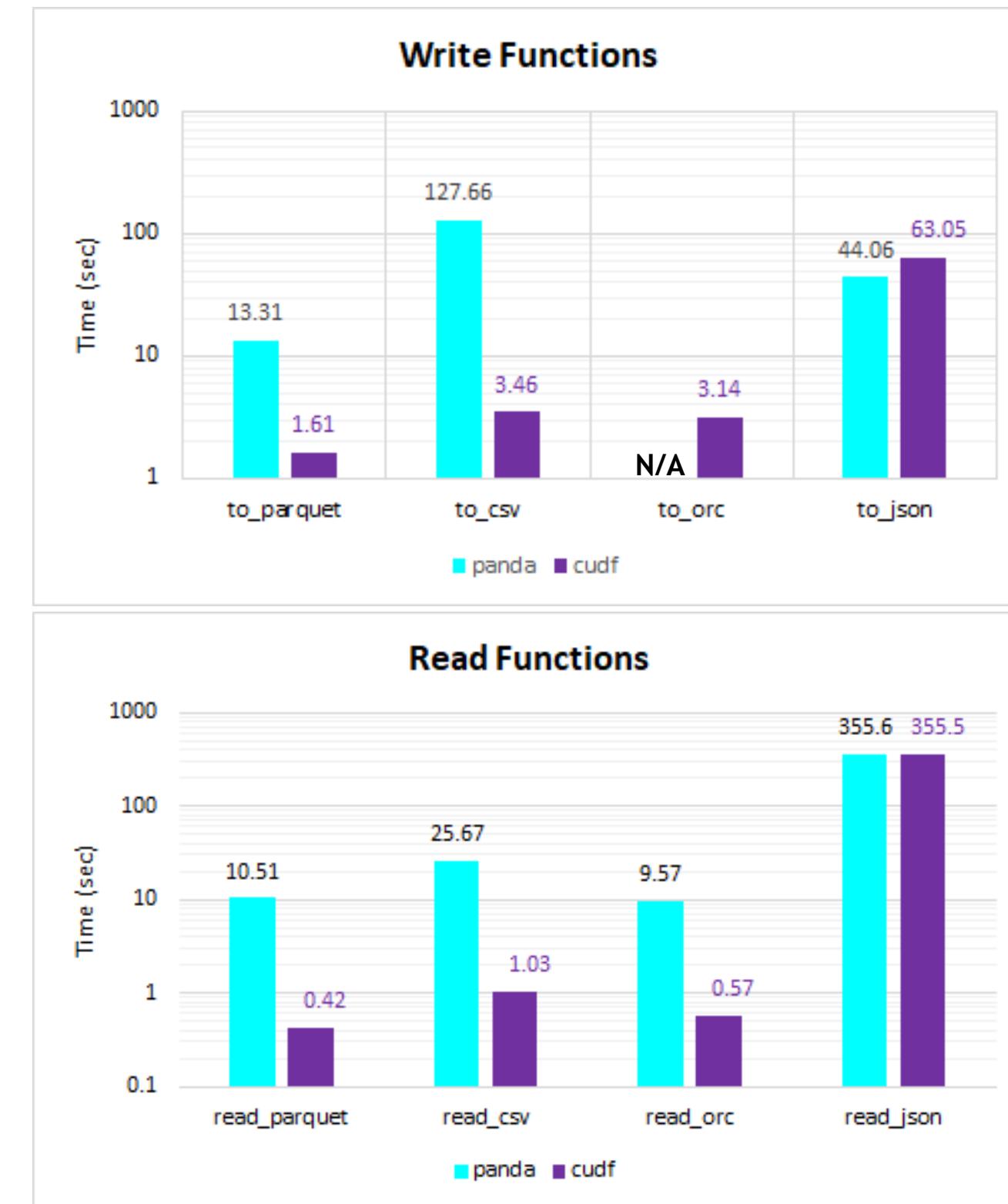
- ▶ Further performance optimization
- ▶ JIT-compiled String UDFs



# Extraction is the Cornerstone

## cuIO for Faster Data Loading

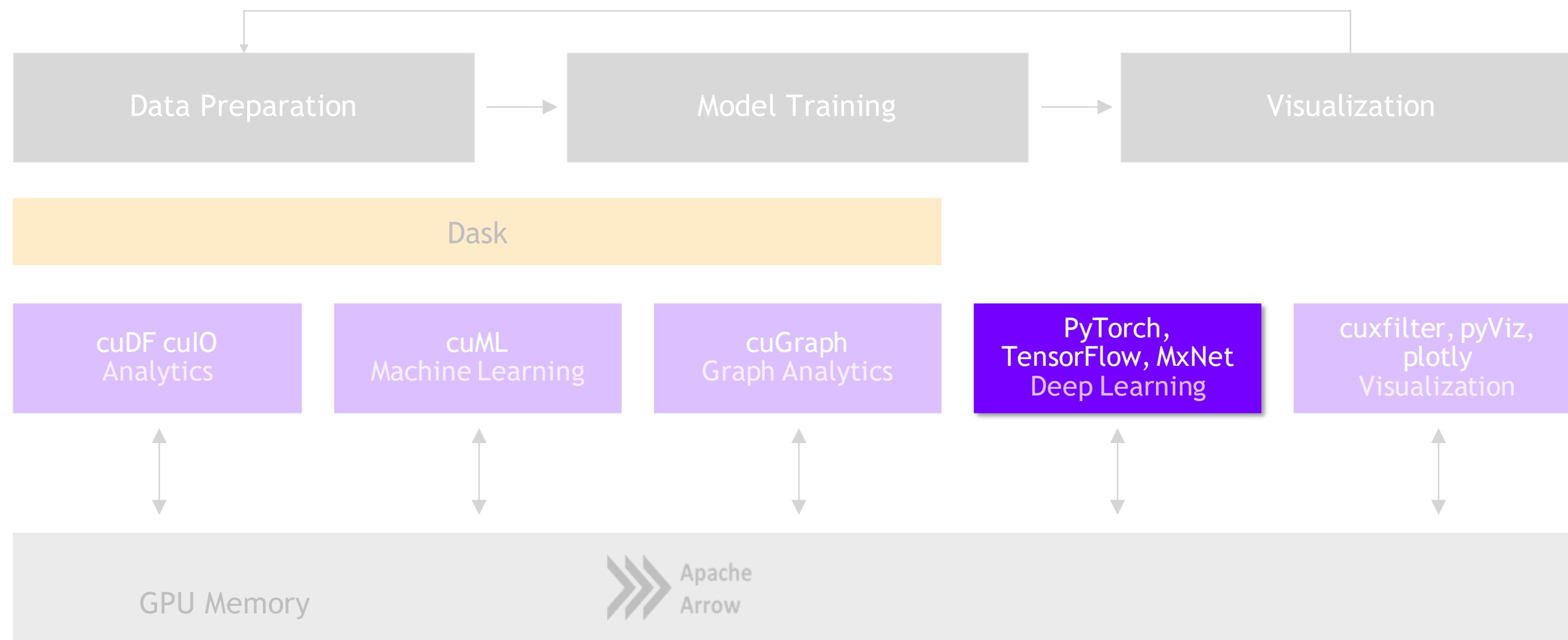
- ▶ Follow Pandas APIs and provide >10x speedup
- ▶ Multiple supported formats, including:
  - ▶ CSV Reader, CSV Writer
  - ▶ Parquet Reader, Parquet Writer
  - ▶ ORC Reader, ORC Writer
  - ▶ JSON Reader
  - ▶ Avro Reader
- ▶ GPU Direct Storage integration in progress for bypassing PCIe bottlenecks!
- ▶ Key is GPU-accelerating both parsing and decompression
- ▶ Benchmark:
  - ▶ Dataset: NY Taxi dataset (Jan 2015)
  - ▶ GPU: Single 32GB V100
  - ▶ RAPIDS Version: 0.17



# ETL is Not Just DataFrames!

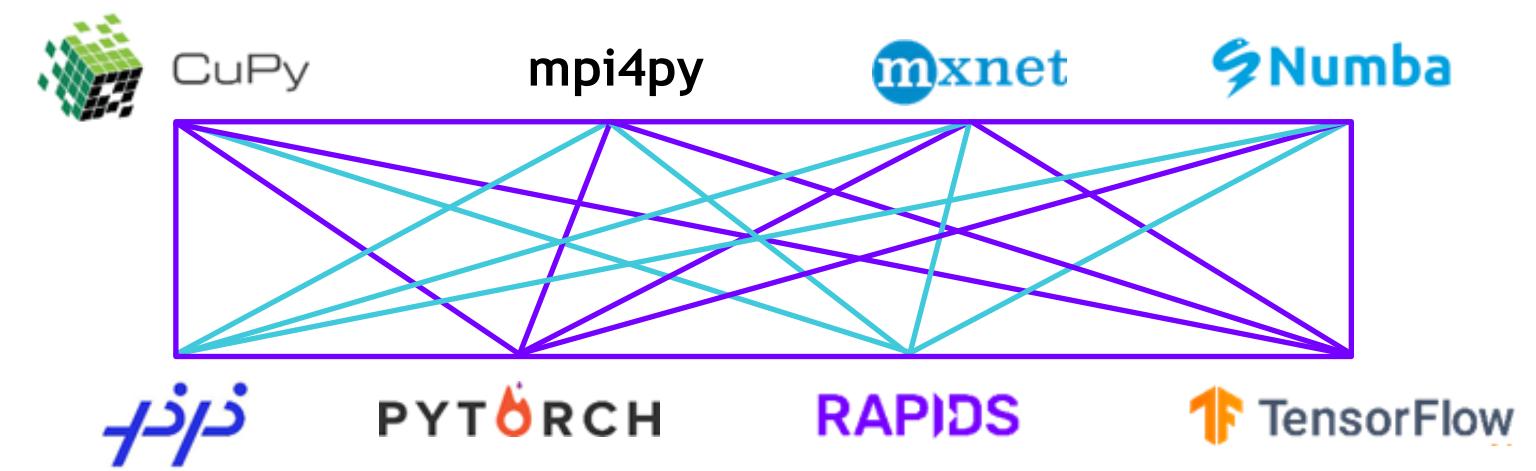
# RAPIDS

## Building Bridges into the Array Ecosystem



# Interoperability for the Win

- ▶ Real-world workflows often need to share data between libraries
- ▶ RAPIDS supports device memory sharing between many popular data science and deep learning libraries
- ▶ Keeps data on the GPU--avoids costly copying back and forth to host memory
- ▶ Any library that supports DLPack or `__cuda_array_interface__` will allow for sharing of memory buffers between RAPIDS and supported libraries



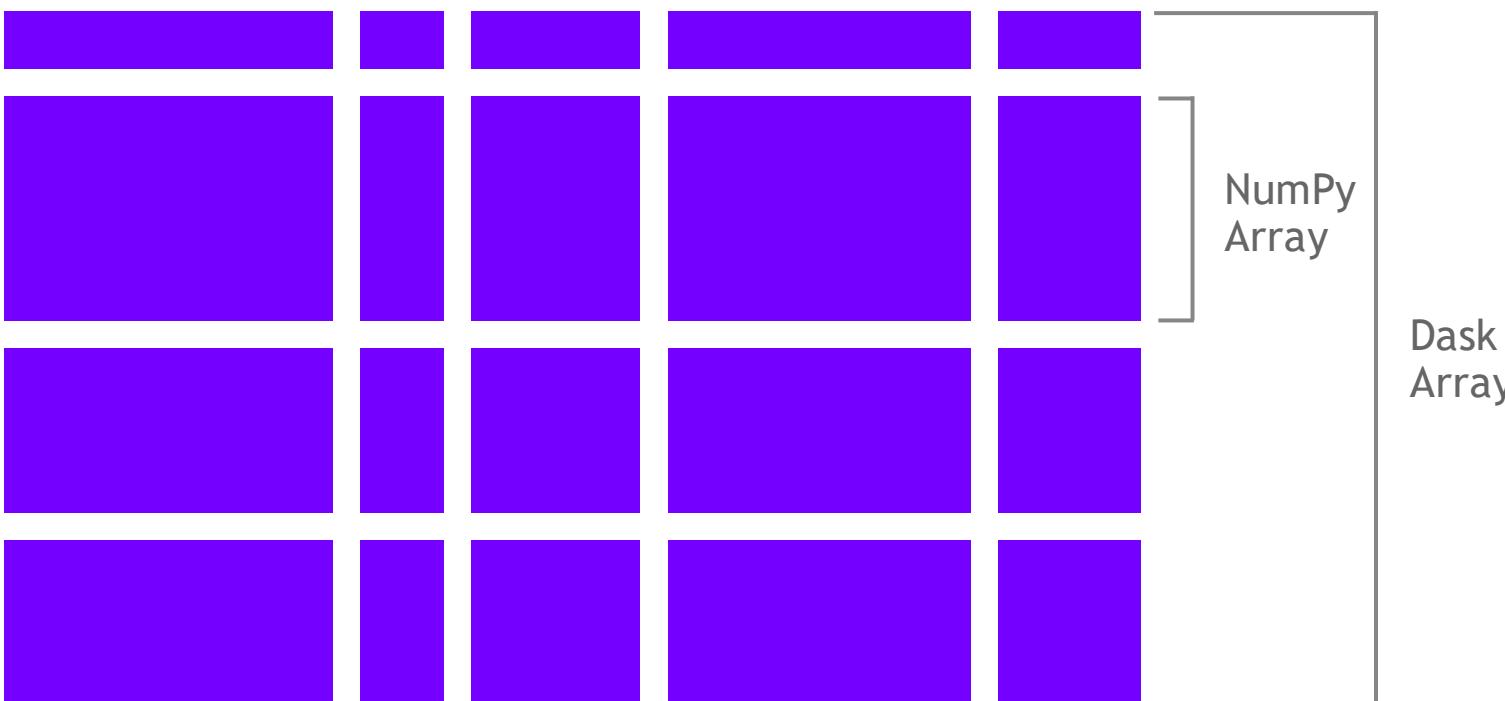
# ETL - Arrays and DataFrames

## Dask and CUDA Python Arrays



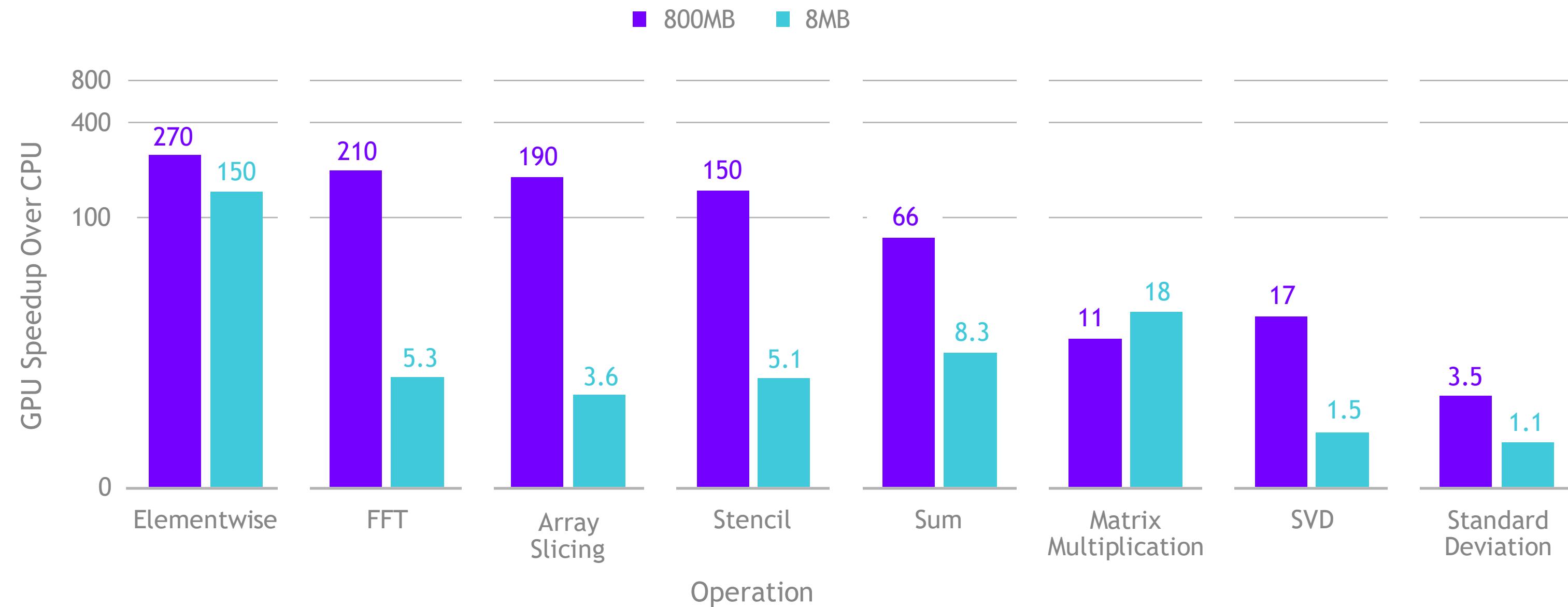
CuPy

The Numba logo features a blue lightning bolt icon followed by the word "Numba" in a bold, blue, sans-serif font.



- ▶ Scales NumPy to distributed clusters
- ▶ Used in climate science, imaging, HPC analysis up to 100TB size
- ▶ Now seamlessly accelerated with GPUs

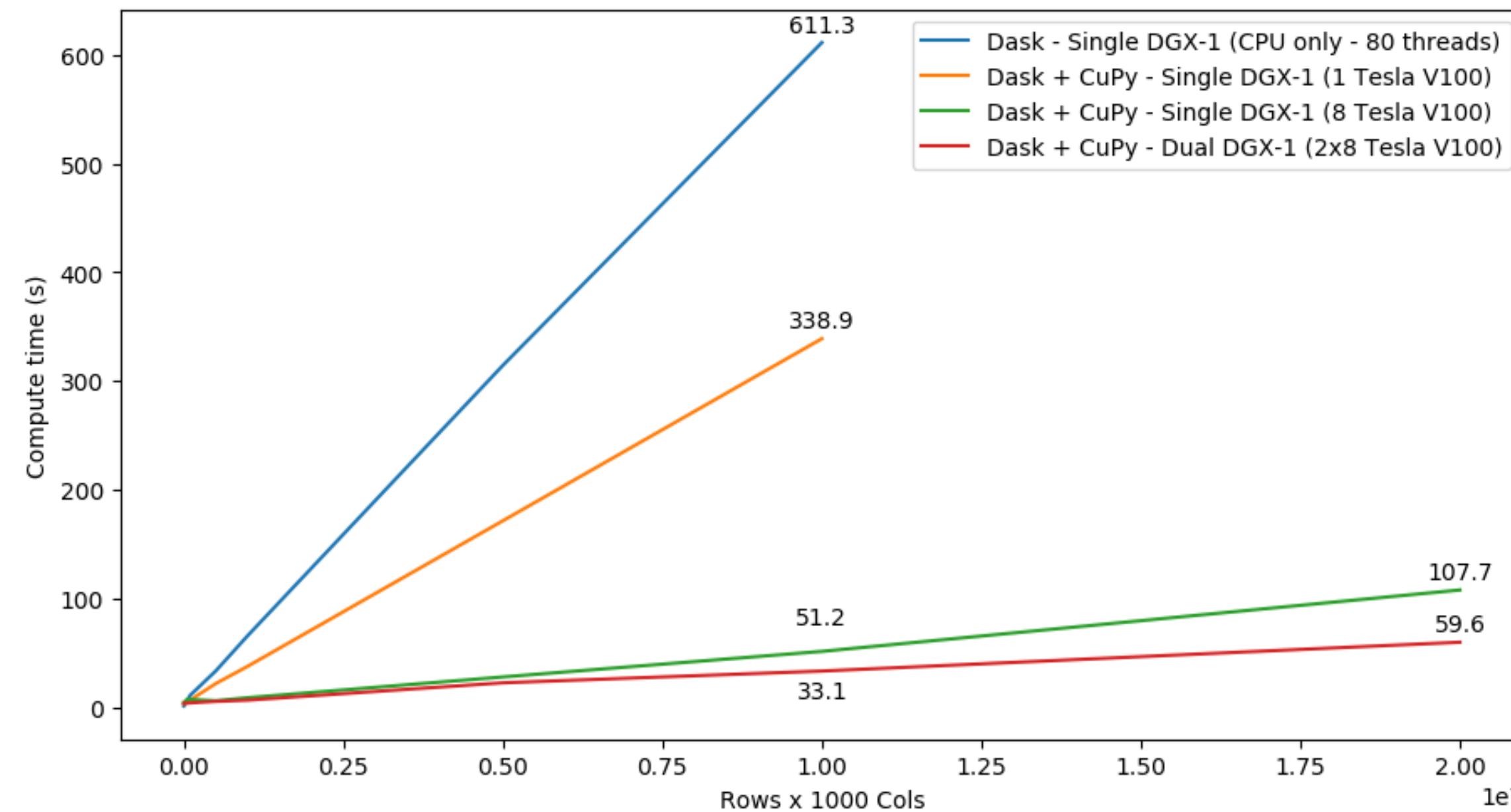
# Benchmark: Single-GPU CuPy vs NumPy



More details: <https://blog.dask.org/2019/06/27/single-gpu-cupy-benchmarks>

# SVD Benchmark

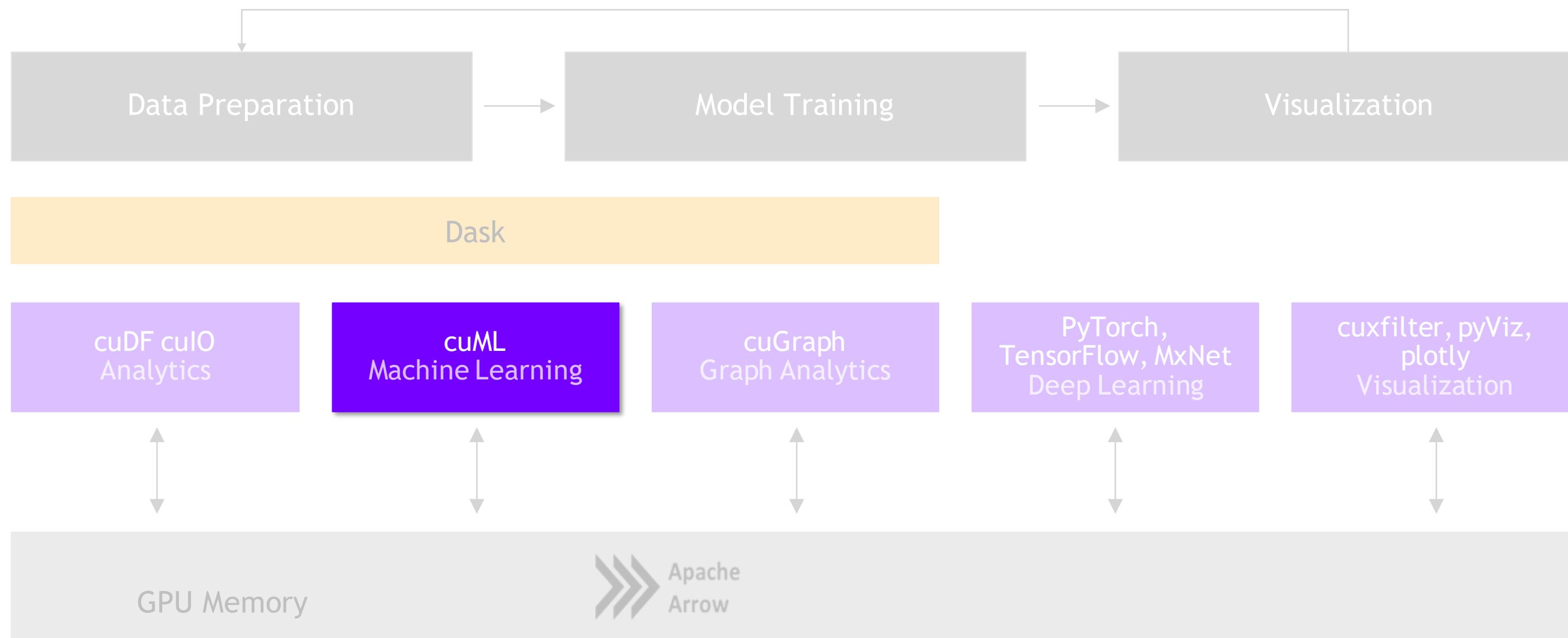
## Dask and CuPy Doing Complex Workflows



cuML

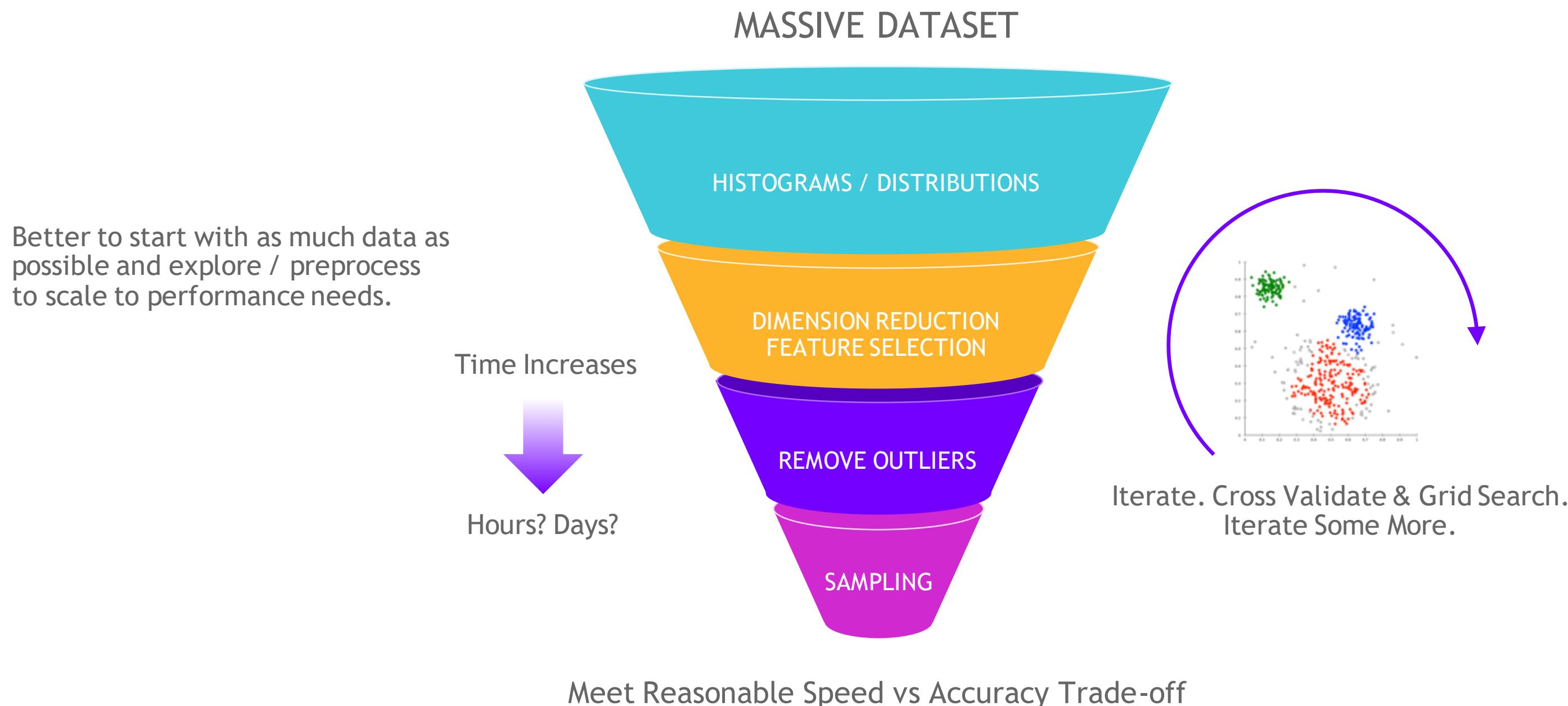
# Machine Learning

## More Models More Problems

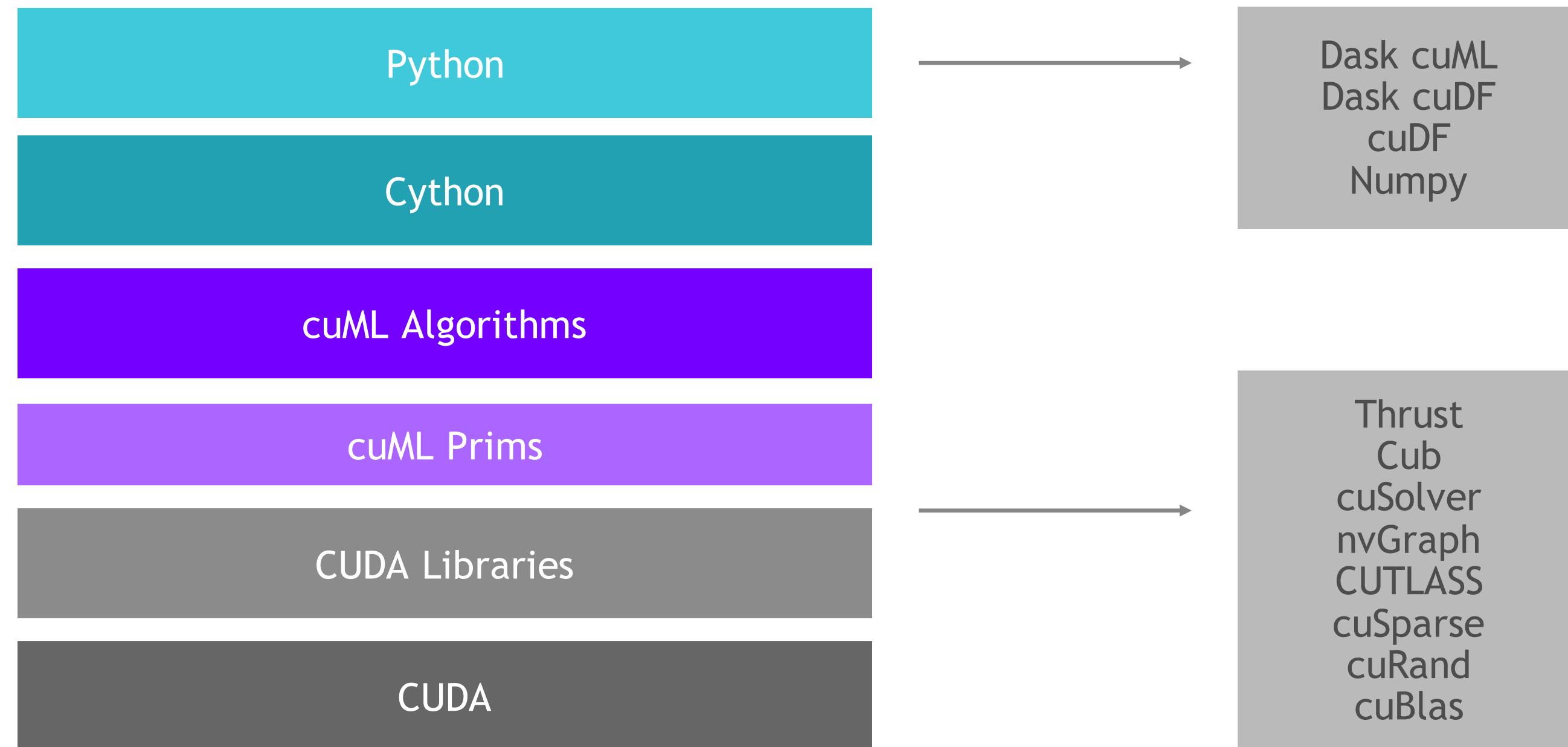


# Problem

## Data Sizes Continue to Grow



# ML Technology Stack



# RAPIDS Matches Common Python APIs

## CPU-based Clustering

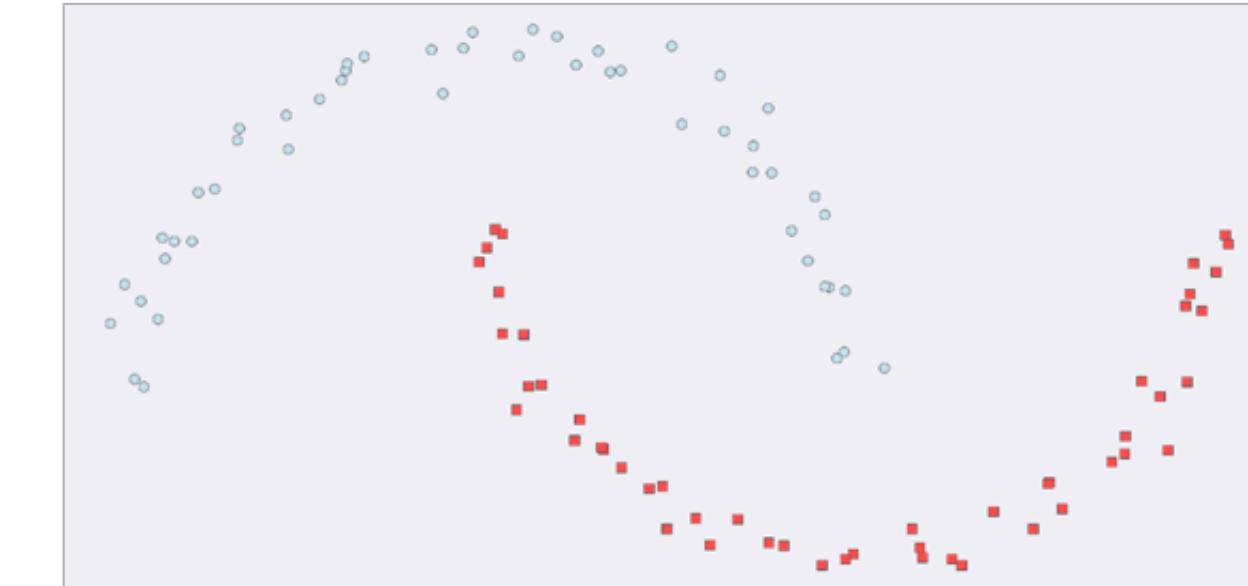
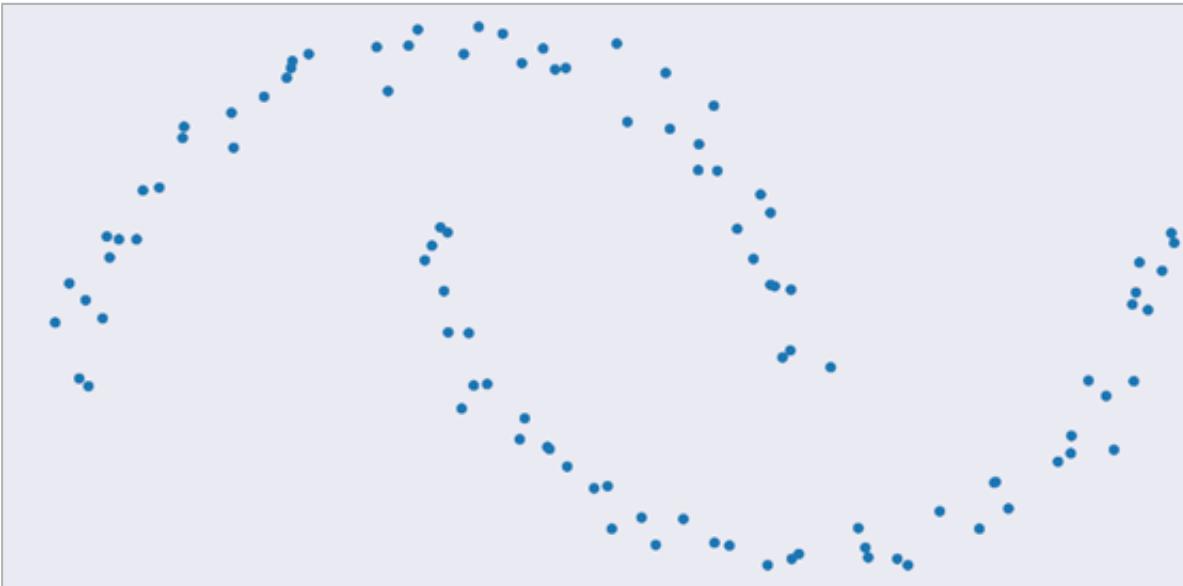
```
from sklearn.datasets import make_moons
import pandas

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

X = pandas.DataFrame({'feature' + str(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

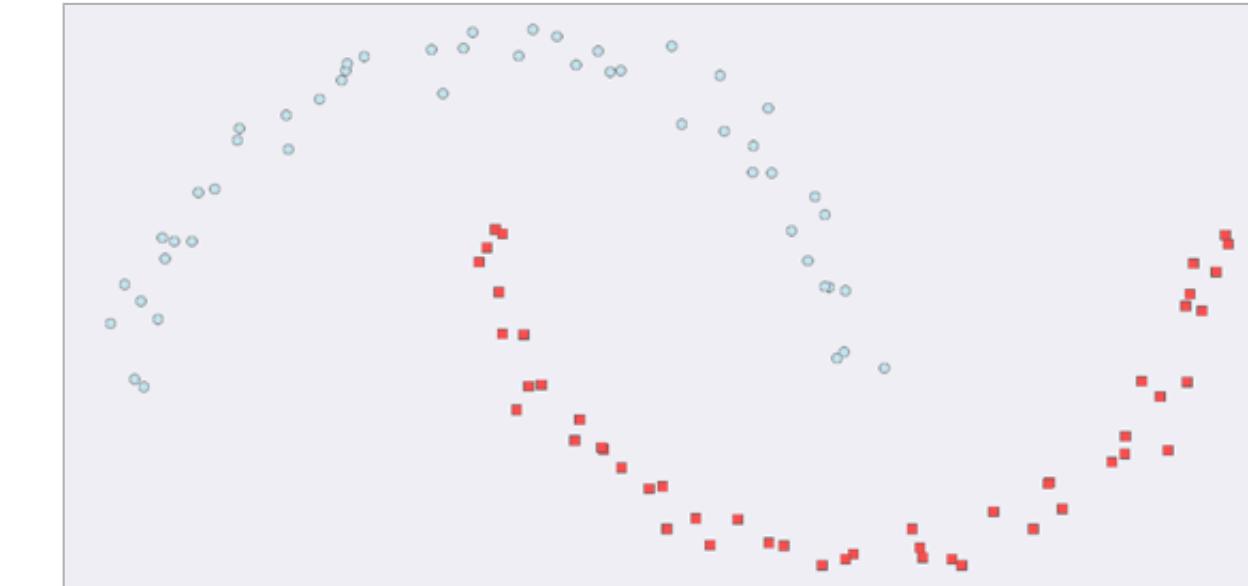
```
from sklearn.datasets import make_moons
import cudf

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

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

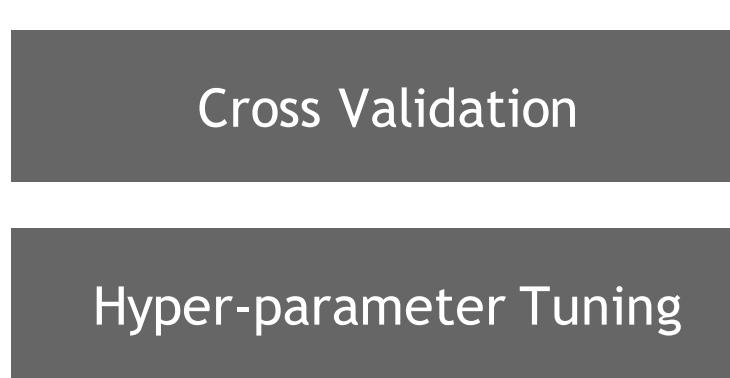
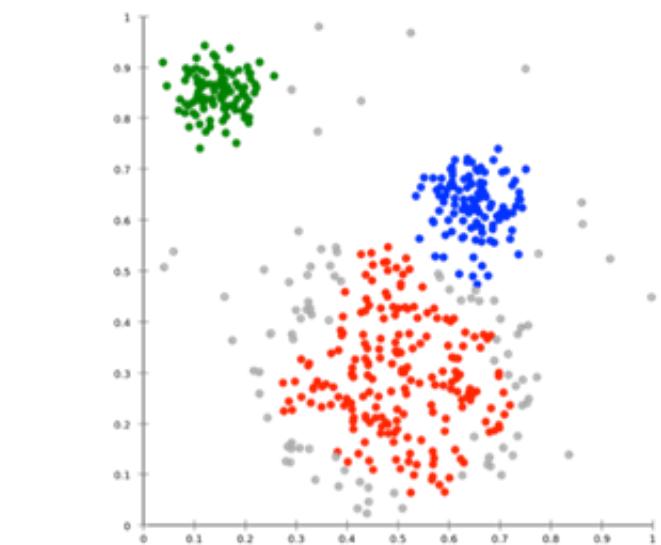
```
from cuml import DBSCAN
dbscan = DBSCAN(eps = 0.3, min_samples = 5)

y_hat = dbscan.fit_predict(X)
```



# Algorithms

## GPU-accelerated Scikit-Learn



More to come!

Classification / Regression

Inference

Preprocessing

Clustering  
Decomposition &  
Dimensionality Reduction

Time Series

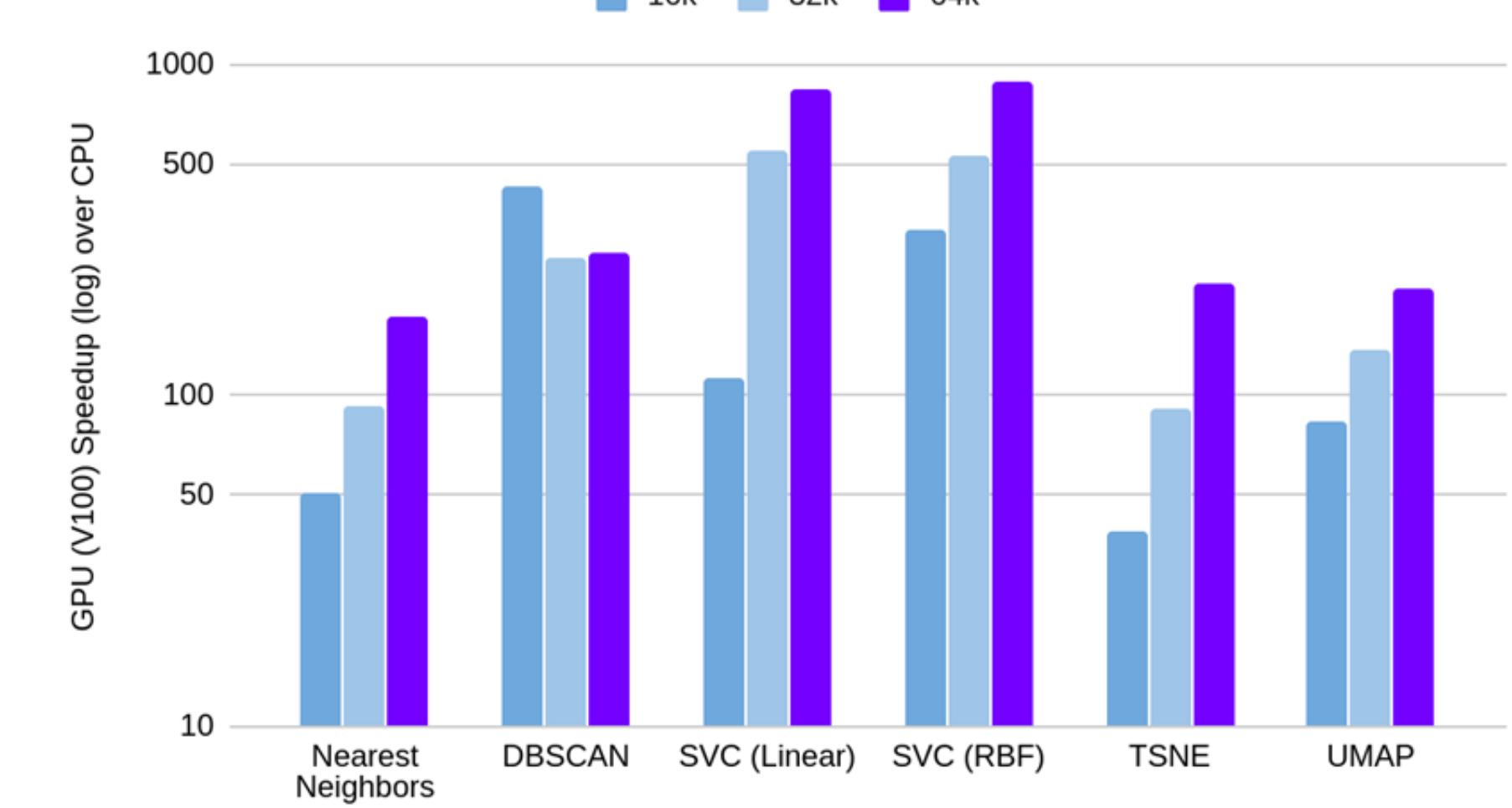
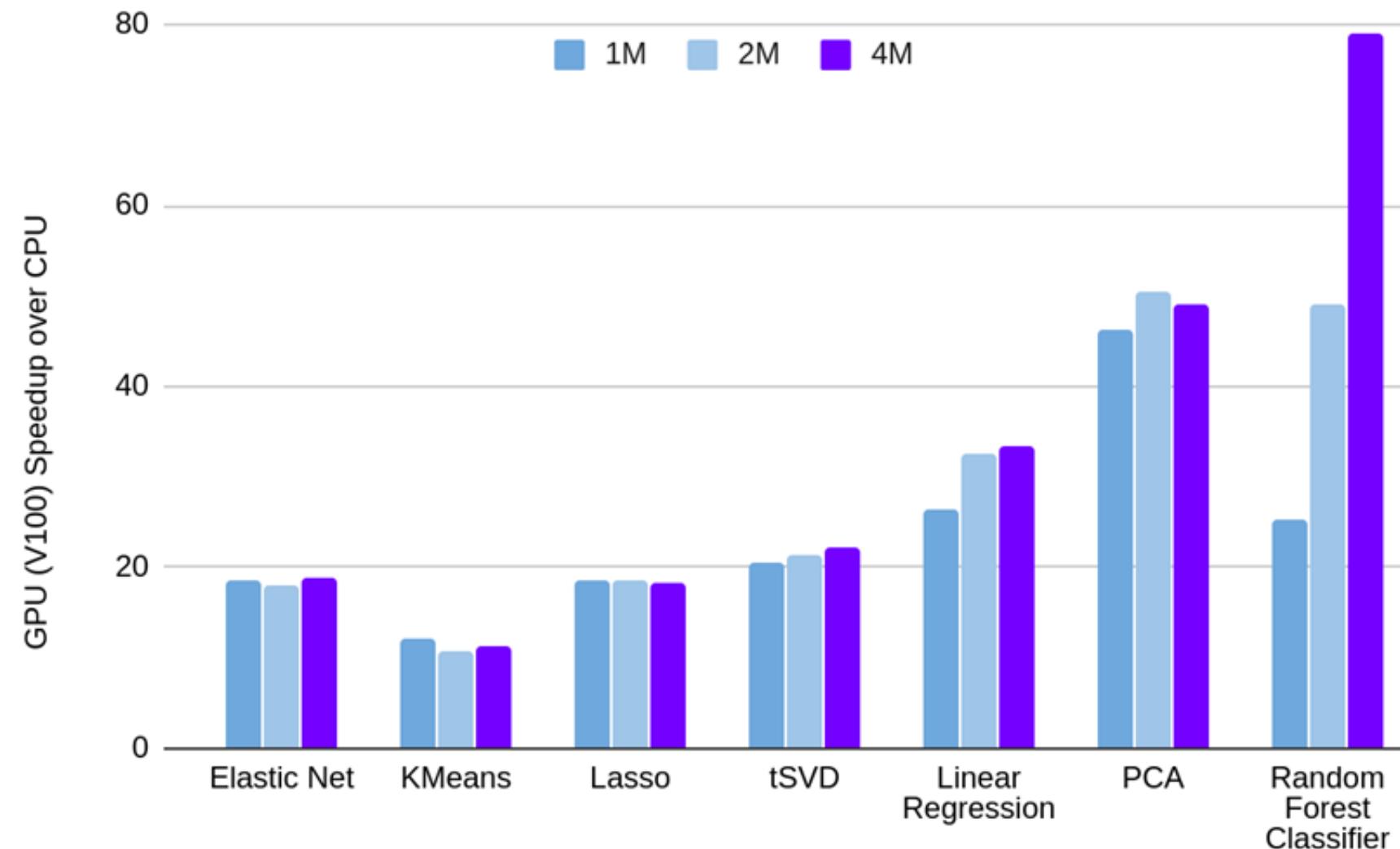
Decision Trees / Random Forests  
Linear/Lasso/Ridge/LARS/ElasticNet Regression  
Logistic Regression  
K-Nearest Neighbors (**exact or approximate**)  
Support Vector Machine Classification and  
Regression  
Naive Bayes  
Random Forest / GBDT Inference (FIL)

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

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

Holt-Winters  
Seasonal ARIMA / Auto ARIMA

# Benchmarks: Single-GPU cuML vs Scikit-learn



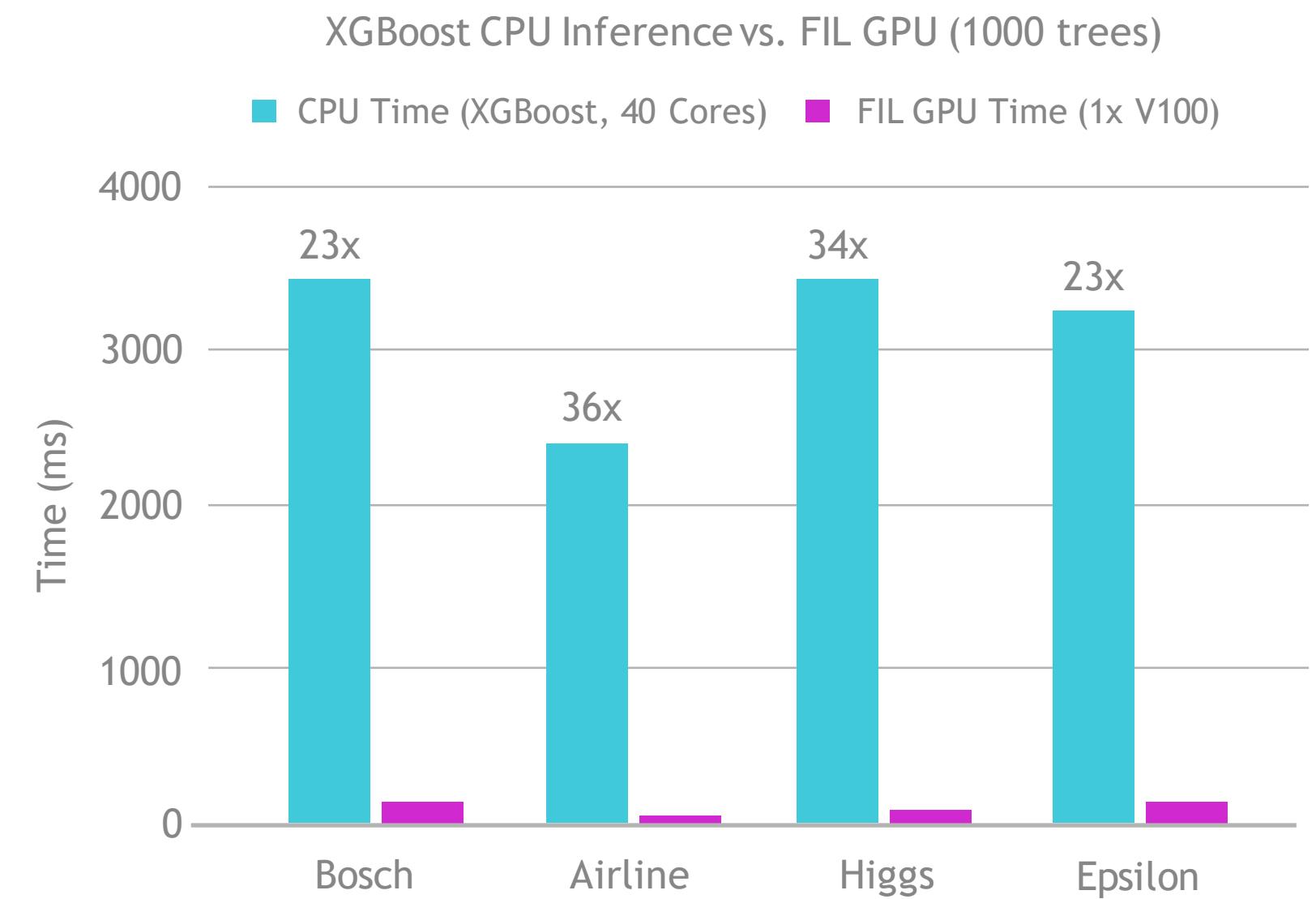
1x V100 vs. 2x 20 Core CPUs (DGX-1, RAPIDS 0.15)

# Forest Inference

## Taking Models From Training to Production

cuML's Forest Inference Library accelerates prediction (inference) for random forests and boosted decision trees:

- Works with existing saved models (XGBoost, LightGBM, scikit-learn RF cuML RF soon)
- Lightweight Python API
- Single V100 GPU can infer up to 34x faster than XGBoost dual-CPU node
- Over 100 million forest inferences



# XGBoost + RAPIDS: Better Together

All RAPIDS changes are integrated upstream and provided to all XGBoost users – via pypi or RAPIDS conda

- ▶ RAPIDS comes paired with XGBoost 1.3.0
- ▶ XGBoost now builds on the GoAI interface standards to provide zero-copy data import from cuDF, cuPY, Numba, PyTorch and more
- ▶ Official Dask API makes it easy to scale to multiple nodes or multiple GPUs
- ▶ gpu\_hist tree builder delivers huge perf gains  
Memory usage when importing GPU data decreased by 2/3 or more
- ▶ Now supports Learning to Rank on GPU



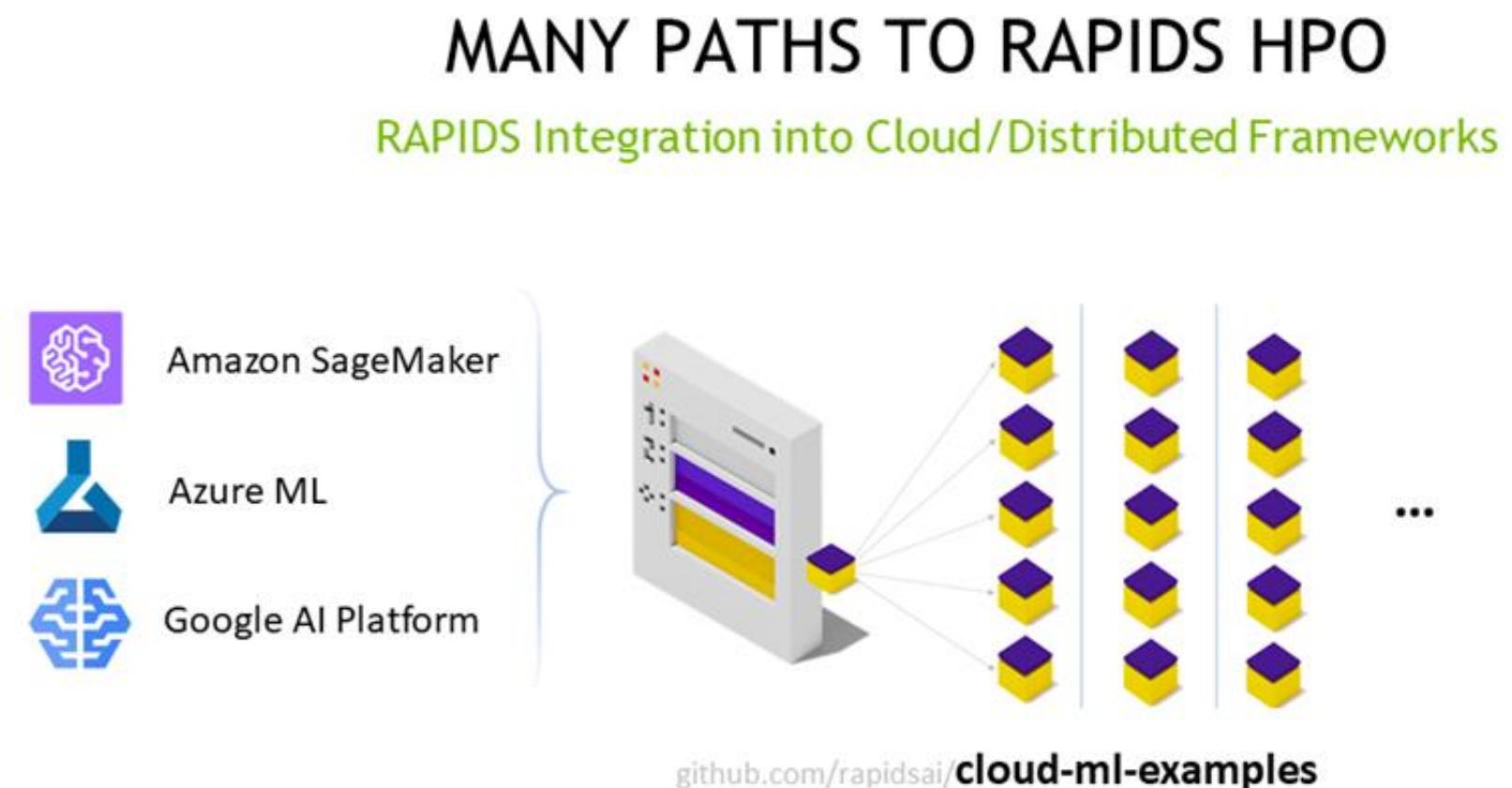
# RAPIDS Integrated into Cloud ML Frameworks

Accelerated machine learning models in RAPIDS give you the flexibility to use hyperparameter optimization (HPO) experiments to explore all variants to find the most accurate possible model for your problem.

With GPU acceleration, RAPIDS models can train 25x faster than CPU equivalents, enabling more experimentation in less time.

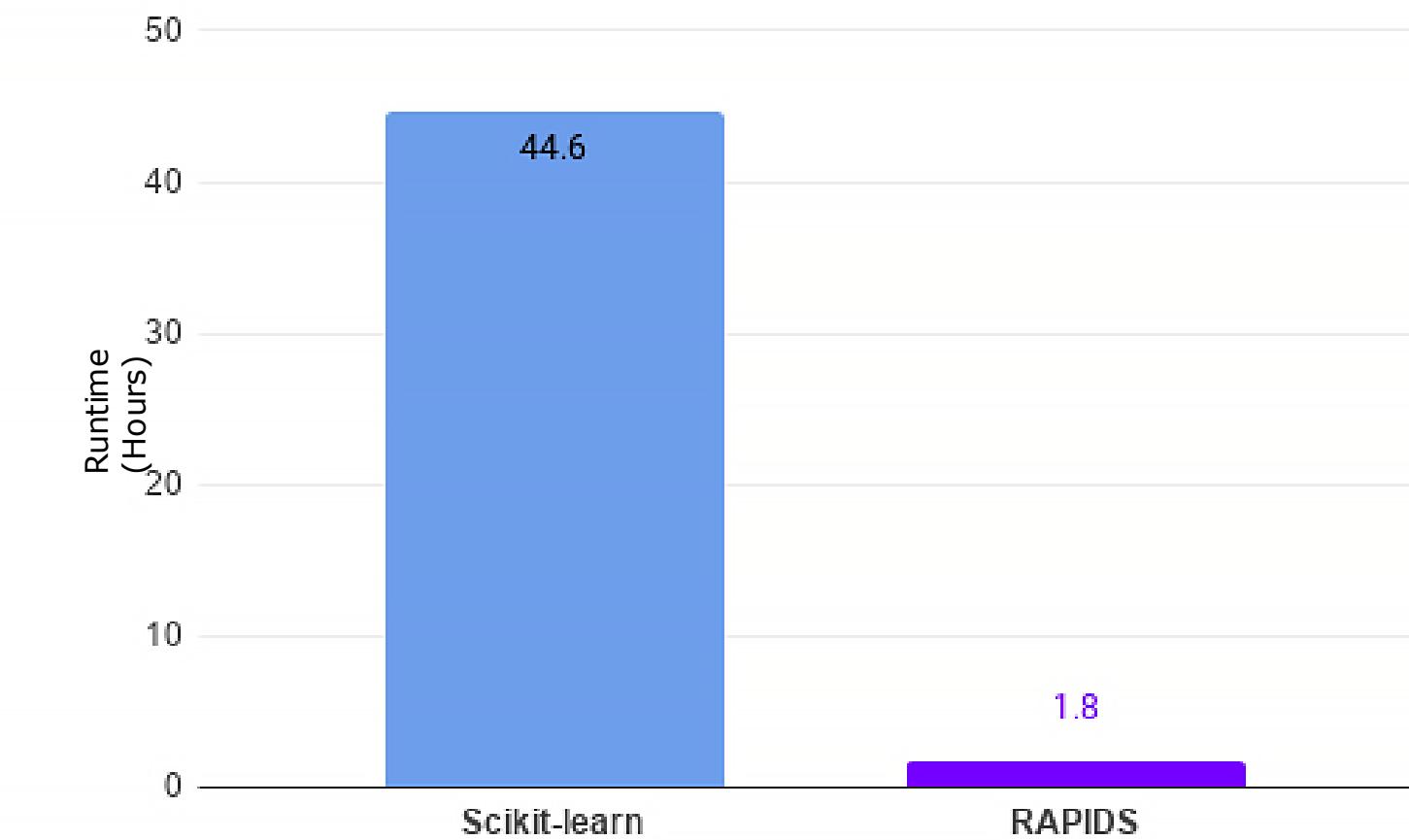
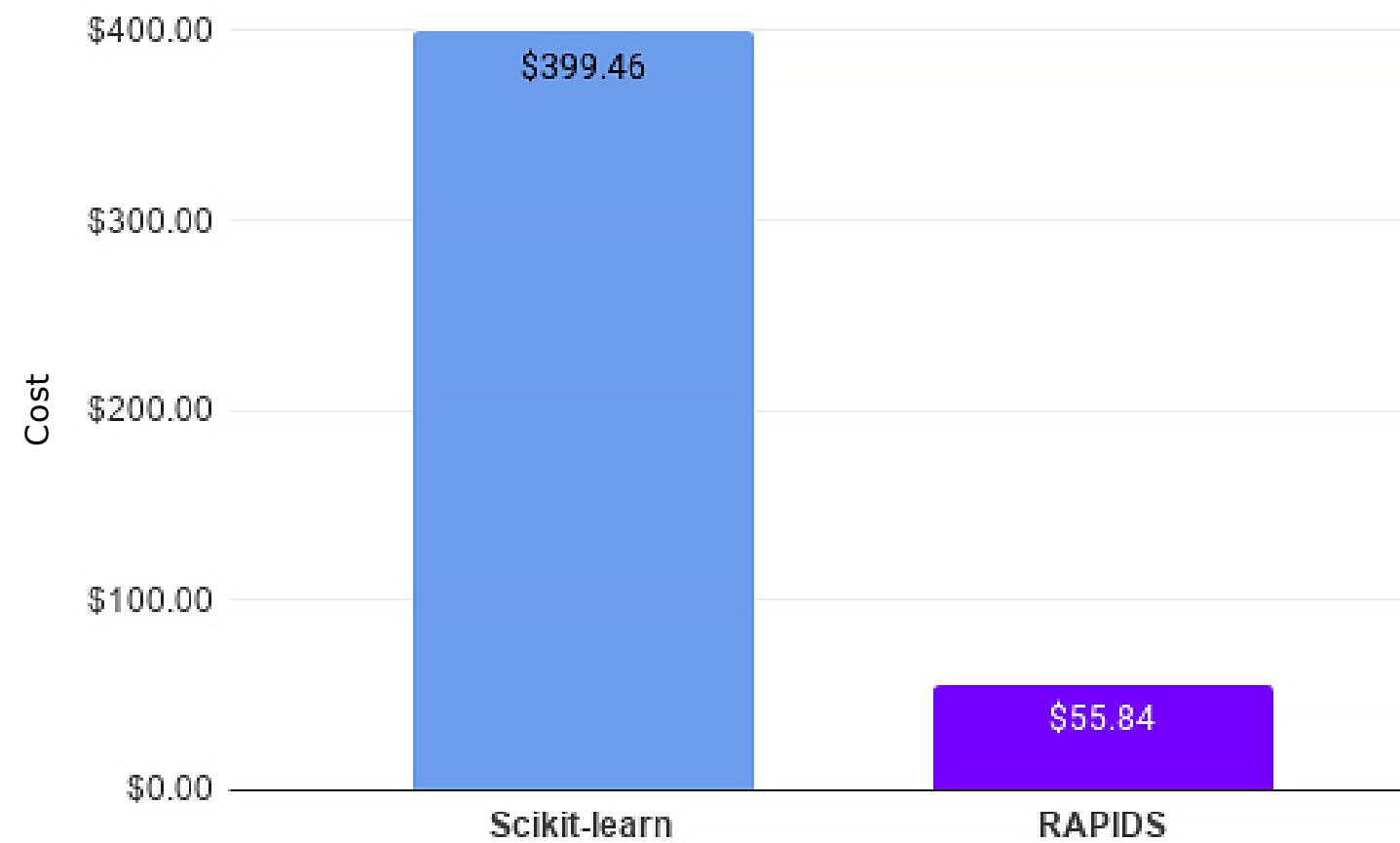
The RAPIDS team works closely with major cloud providers and OSS solution providers to provide code samples to get started with HPO in minutes.

<https://rapids.ai/hpo>



# HPO Use Case: 100-Job Random Forest Airline Model

Huge speedups translate into >7x TCO reduction



Based on sample Random Forest training code from cloud-ml-examples repository, running on Azure ML. 10 concurrent workers with 100 total runs, 100M rows, 5-fold cross-validation per run.

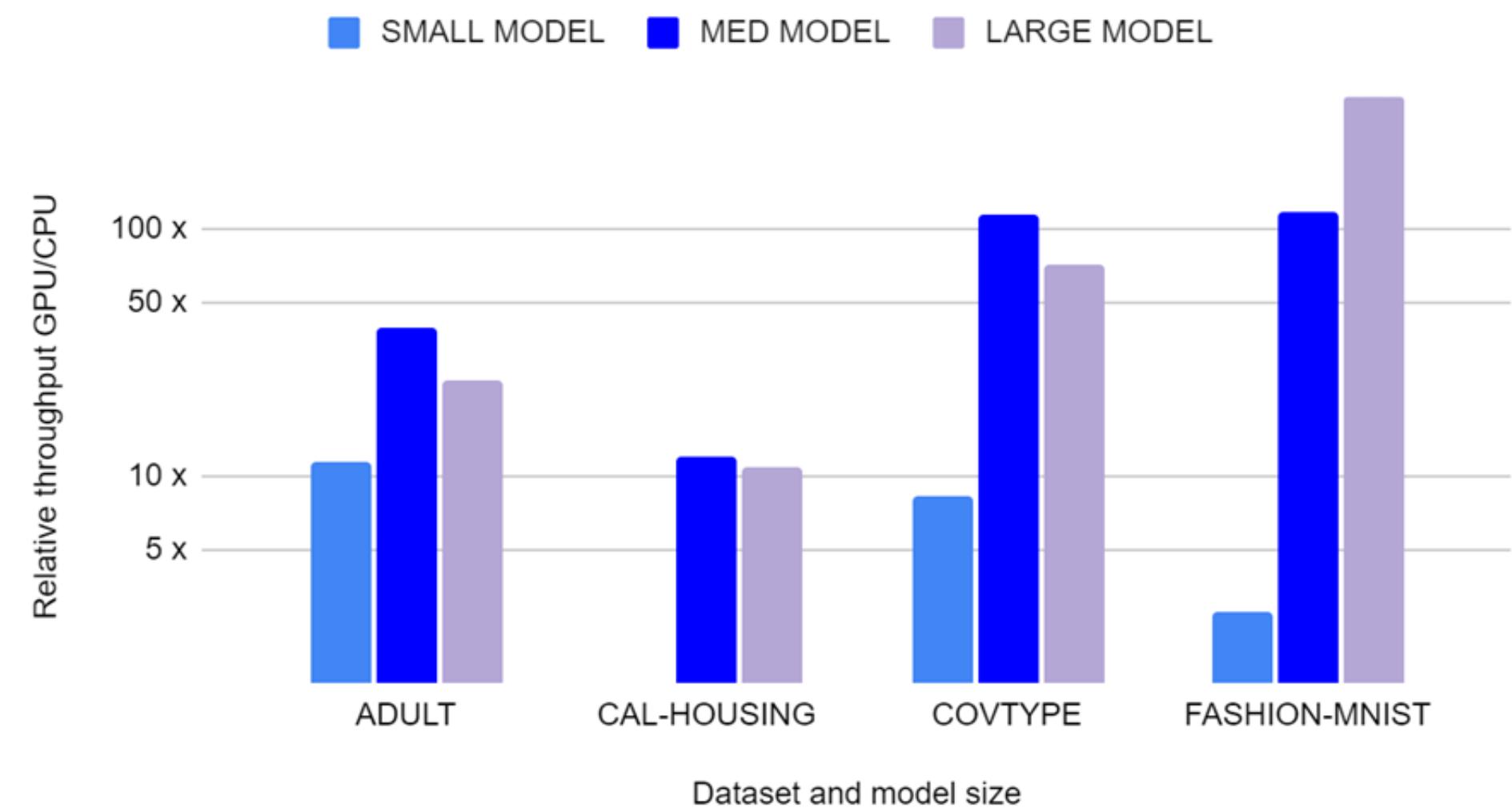
GPU nodes: 10x Standard\_NC6s\_v3, 1 V100 16G, vCPU 6 memory 112G, Xeon E5-2690 v4 (Broadwell) - \$3.366/hour  
CPU nodes: 10x Standard\_DS5\_v2, vCPU 16 memory 56G, Xeon E5-2673 v3 (Haswell) or v4 (Broadwell) - \$1.017/hour"

# SHAP Explainability

## GPUTreeSHAP for XGBoost

- ▶ SHAP provides a principled way to explain the impact of input features on each prediction or on the model overall - critical for interpretability
- ▶ SHAP has often been too computationally-expensive to deploy for large-scale production
- ▶ RAPIDS ships with GPU-accelerated SHAP for XGBoost with speedups of 20x or more (demo code available in the XGBoost repo)
- ▶ RAPIDS 0.18 includes experimental Kernel and Permutation explainers for black box models

GPUTreeSHAP Speedups (1x V100 vs. 2x 20 E5-2698)



# Overview of cuML Algorithms

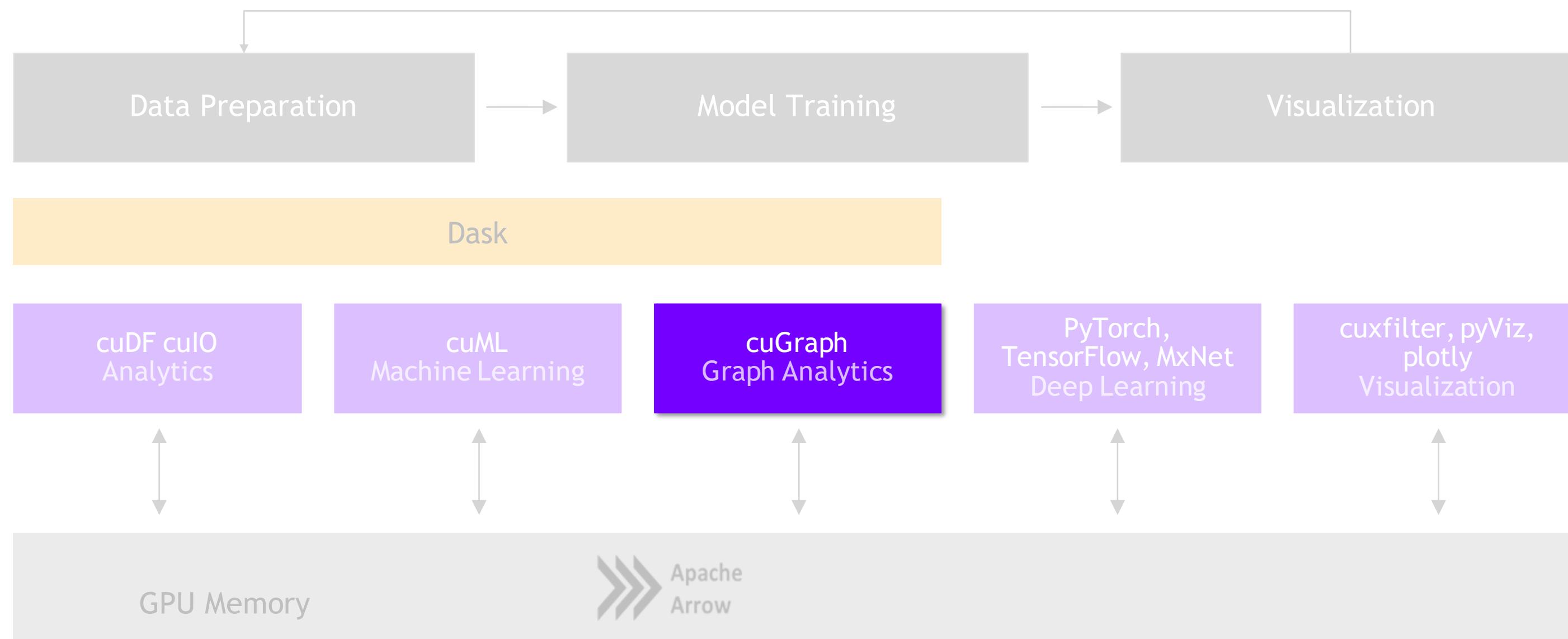
RAPIDS 0.18 - February 2021

cuML	Single-GPU	Multi-Node-Multi-GPU
Gradient Boosted Decision Trees (GBDT)		
Linear Regression		
Logistic Regression		
Random Forest		
K-Means		
K-NN		
DBSCAN		
UMAP		
Holt-Winters		
ARIMA		
T-SNE		
Principal Components		
Singular Value Decomposition		
SVM		

# cuGraph

# Graph Analytics

## More Connections, More Insights



# Goals and Benefits of cuGraph

## Focus on Features and User Experience

### BREAKTHROUGH PERFORMANCE

- ▶ Up to 500 million edges on a single 32GB GPU
- ▶ Multi-GPU support for scaling into the billions of edges

### SEAMLESS INTEGRATION WITH cuDF AND cuML

- ▶ Property Graph support via DataFrames

### Scaling without changing API

- ▶ Scale without having to rewrite workflow

### MULTIPLE APIs

- ▶ **Python:** Familiar NetworkX-like API
- ▶ **C/C++:** lower-level granular control for application developers

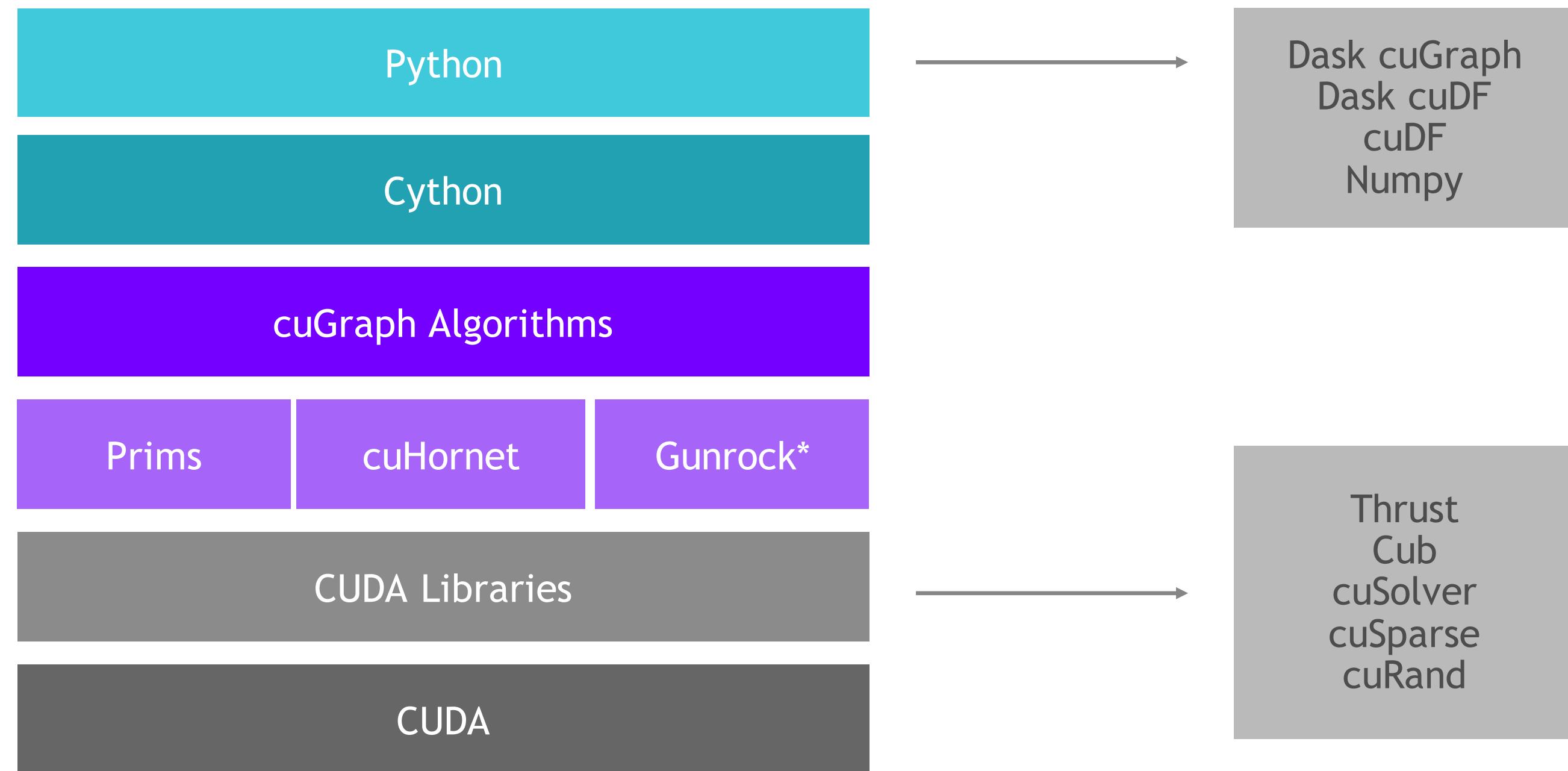
### Compatibility

- ▶ Accepts NetworkX graph
- ▶ Accepts SciPy and CuPy Sparse Matrix

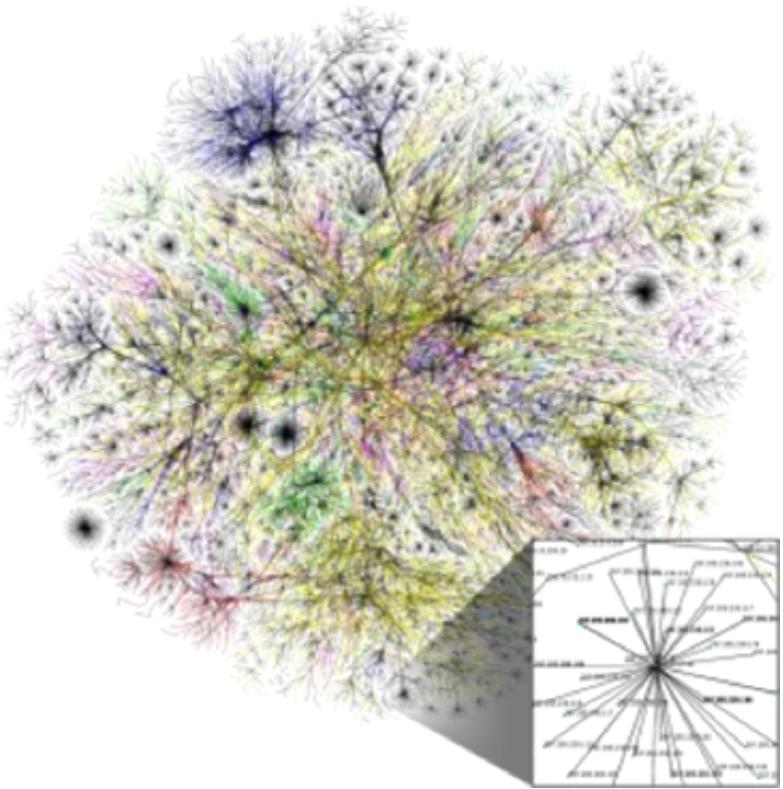
### GROWING FUNCTIONALITY

- ▶ Extensive collection of algorithm, primitive, and utility functions

# Graph Technology Stack



\* Gunrock is from UC Davis



# Algorithms

## GPU-accelerated NetworkX

Traveling Salesman

Routing

Minimum Spanning Tree  
Maximum Spanning Tree

Tree

Graph Classes  
Subgraph Extraction  
**Egonet**

Structure

Force Atlas 2  
Hungarian Algorithm

Other

Renumbering  
Auto-Renumbering  
NetworkX converters

Utilities

Community

Spectral Clustering

- Balanced Cut and Modularity Maximization

Louvain (Multi-GPU) and Leiden

Ensemble Clustering for Graphs

KCore and KCore Number

Triangle Counting

K-Truss

Components

Weakly Connected Components

Strongly Connected Components

Link Analysis

Page Rank (Multi-GPU)

Personal Page Rank (Multi-GPU)

HITS

Link Prediction

Jaccard

Weighted Jaccard

Overlap Coefficient

Traversal

Single Source Shortest Path (SSSP) (Multi-GPU)

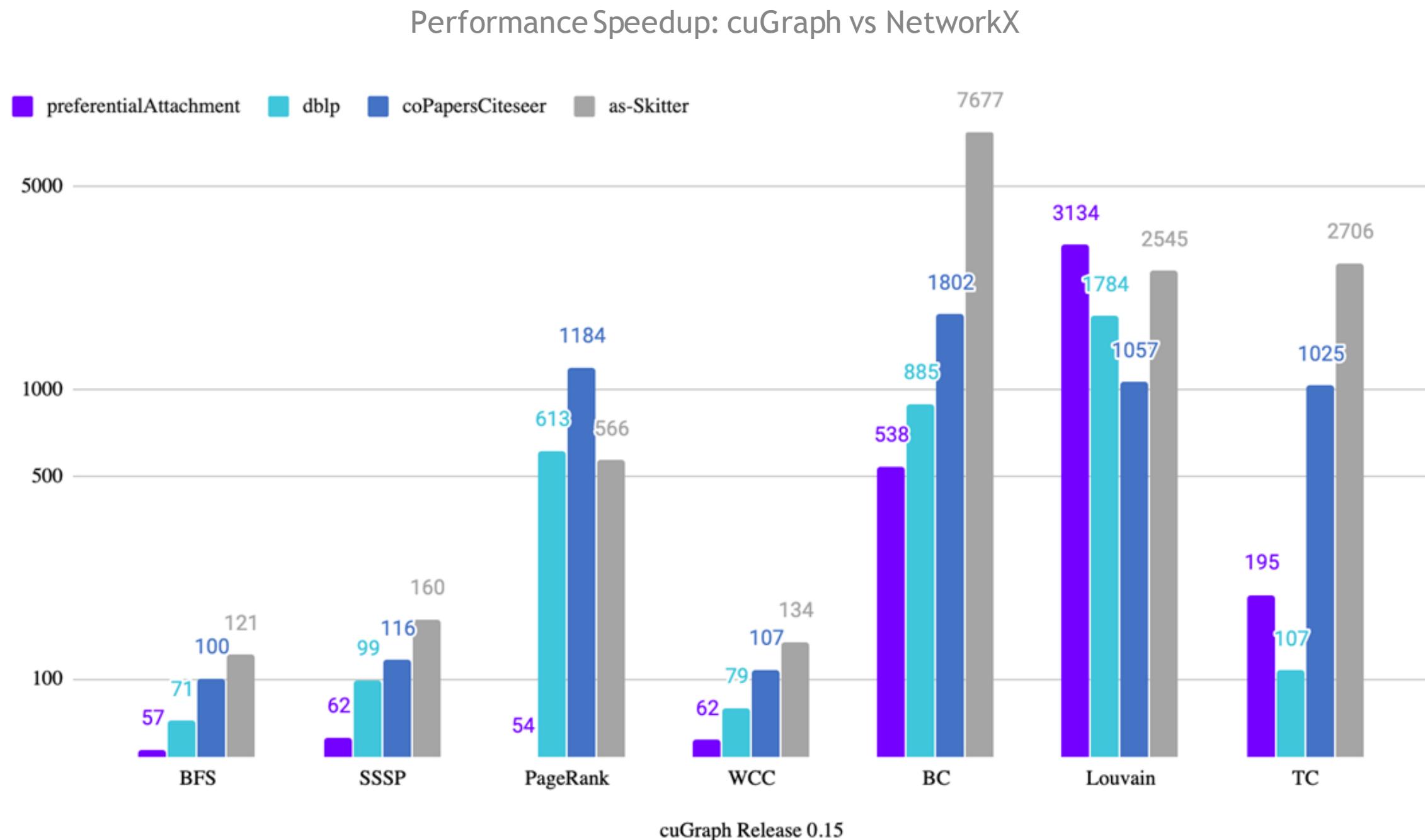
Breadth First Search (BFS) (Multi-GPU)

Centrality

Katz (Multi-GPU)

Betweenness Centrality (Vertex and Edge)

# Benchmarks: Single-GPU cuGraph vs NetworkX



Dataset	Nodes	Edges
preferentialAttachment	100,000	999,970
Dblp-2010	326,186	1,615,400
coPapersCiteseer	434,102	32,073,440
As-Skitter	1,696,415	22,190,596

# NetworkX Compatibility

Use your NetworkX.Graph Objects directly within cuGraph

```
import networkx as nx
import time
import operator

# create a random graph
G = nx.barabasi_albert_graph(N, M)

... do some NetworkX stuff ..

t1 = time.time()
bc = nx.betweenness_centrality(G)
t2 = time.time() - t1

print(t2)
```

NetworkX

```
import networkx as nx
import time
import operator
import cugraph as cnx

# create a random graph
G = nx.barabasi_albert_graph(N, M)

... do some NetworkX stuff ..

t1 = time.time()
bc = cnx.betweenness_centrality(G)
t2 = time.time() - t1

print(t2)
```

NetworkX + RAPIDS cuGraph

Node	Edges	Speedup
100	1,344	0.45
200	2,944	1.14
400	6,144	2.64
800	12,544	5.26
1,600	25,344	12.99
3,200	50,944	26.5
6,400	102,144	48.62
12,800	204,544	89.81
25,600	409,344	180.42
51,200	818,944	328.05

# Multi-GPU PageRank Performance

PageRank Portion of the HiBench Benchmark Suite

HiBench Scale	Vertices	Edges	CSV File (GB)	# of GPUs	# of CPU Threads	Pagerank for 3 Iterations (secs)
Huge	5,000,000	198,000,000	3	1		1.1
BigData	50,000,000	1,980,000,000	34	3		5.1
BigData x2	100,000,000	4,000,000,000	69	6		9.0
BigData x4	200,000,000	8,000,000,000	146	12		18.2
BigData x8	400,000,000	16,000,000,000	300	16		31.8
BigData x8	400,000,000	16,000,000,000	300		800*	5760*

\*BigData x8, 100x 8-vCPU nodes, Apache Spark GraphX ⇒ 96 mins!

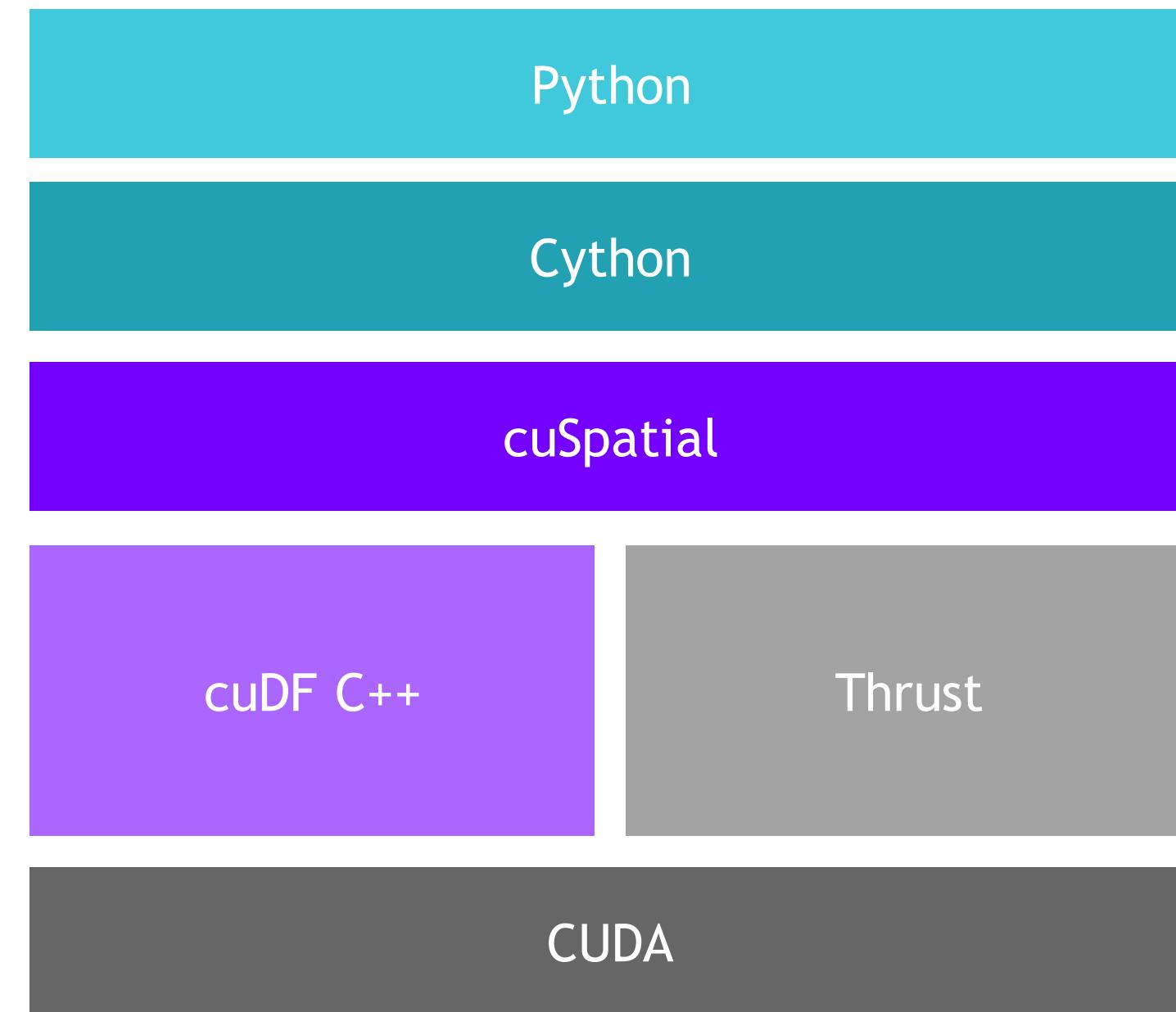
# Overview of cuGraph Algorithms

RAPIDS 0.18 - February 2021

cuGRAPH	Single-GPU	Multi-Node-Multi-GPU
Page Rank		
Personal Page Rank		
Katz		
Betweenness Centrality		
Spectral Clustering		
Louvain		
Ensemble Clustering for Graphs		
K-Truss & K-Core		
Connected Components (Weak & Strong)		
Triangle Counting		
Single Source Shortest Path (SSSP)		
Breadth-First Search (BFS)		
Jaccard & Overlap Coefficient		
Force Atlas 2		
Hungarian Algorithm		
Leiden		

cuSpatial

# cuSpatial Technology Stack



# cuSpatial

## BREAKTHROUGH PERFORMANCE & EASE OF USE

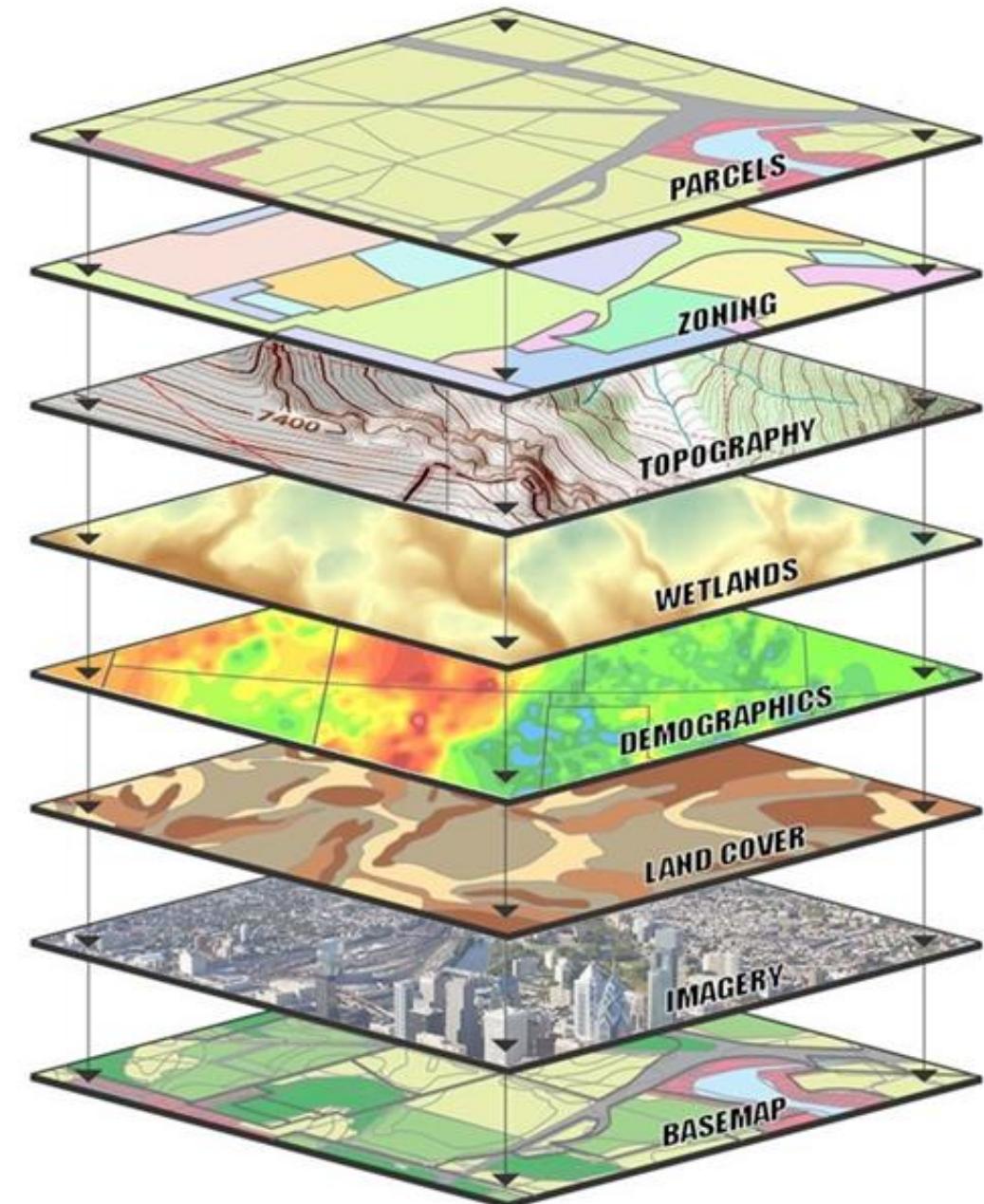
- ▶ Up to 1000x faster than CPU spatial libraries
- ▶ Python and C++ APIs for maximum usability and integration

## GROWING FUNCTIONALITY

- ▶ Extensive collection of algorithm, primitive, and utility functions for spatial analytics

## SEAMLESS INTEGRATION INTO RAPIDS

- ▶ cuDF for data loading, cuGraph for routing optimization, and cuML for clustering are just a few examples



# cuSpatial

## Current and planned functionality

Layer	Current Functionality	Functionality Roadmap
<b>High-level Analytics</b>	C++ Library w. Python bindings enabling distance, speed, trajectory similarity, trajectory clustering	Symmetric segment path distance
<b>Graph Layer</b>	cuGraph	cuGraph
<b>Query Layer</b>	Spatial Window, Nearest polyline	Nearest Neighbor, Spatiotemporal range search and joins
<b>Index Layer</b>	Quadtree	
<b>Geo-operations</b>	Point in polygon (PIP), Haversine distance, Hausdorff distance, lat-lon to xy transformation	ST_distance and ST_contains
<b>Geo-Representation</b>	Shape primitives, points, polylines, polygons	Fiona/Geopandas I/O support and object representations

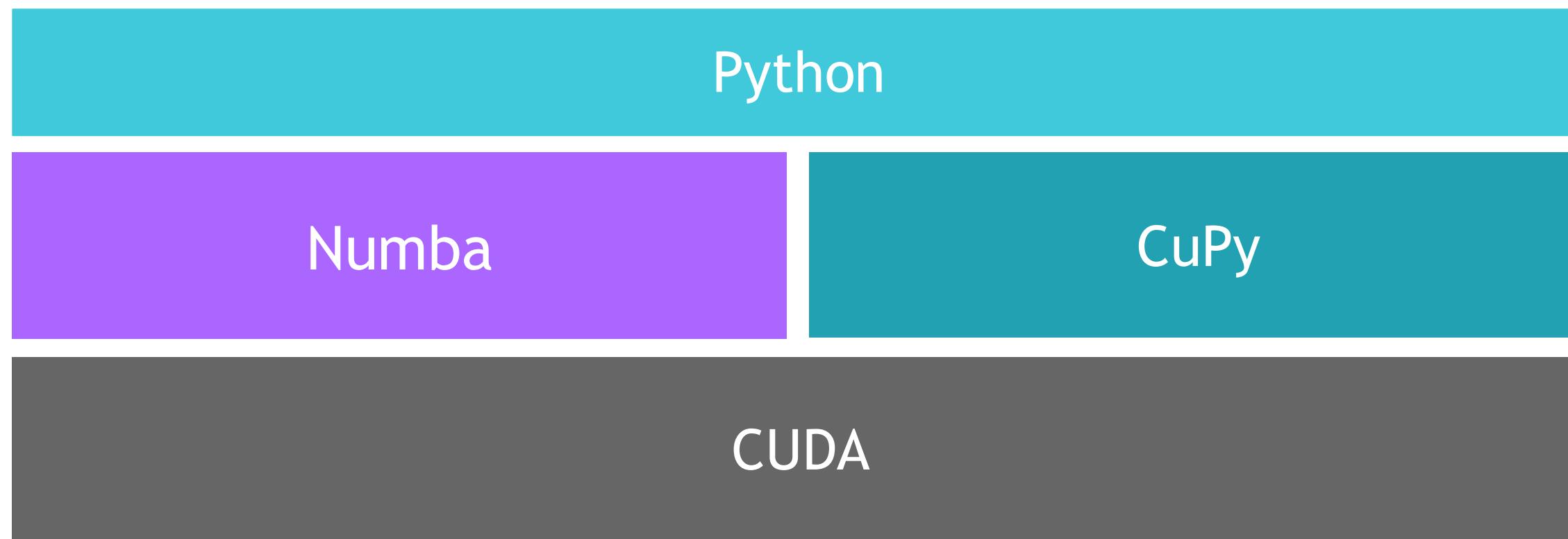
# cuSpatial

## Performance at a Glance

cuSpatial Operation	Input Data	cuSpatial Runtime	Reference Runtime	Speedup
<b>Point-in-Polygon Test</b>	1.3+ million vehicle point locations and 27 Region of Interests	1.11 ms (C++) 1.50 ms (Python) [Nvidia Titan V]	334 ms (C++, optimized serial) 130468.2 ms (python Shapely API, serial) [Intel i7-7800X]	301X (C++) 86,978X (Python)
<b>Haversine Distance Computation</b>	13+ million Monthly NYC taxi trip pickup and drop-off locations	7.61 ms (Python) [Nvidia T4]	416.9 ms (Numba) [Nvidia T4]	54.7X (Python)
<b>Hausdorff Distance Computation (for clustering)</b>	52,800 trajectories with 1.3+ million points	13.5s [Quadro V100]	19227.5s (Python SciPy API, serial) [Intel i7-6700K]	1,400X (Python)

cuSignal

# cuSignal Technology Stack



Unlike other RAPIDS libraries, cuSignal is purely developed in Python with custom CUDA Kernels written with Numba and CuPy (notice no Cython layer).

# cuSignal - Selected Algorithms

## GPU-accelerated SciPy Signal



Wavelets

Peak Finding

More to come!

Convolution

Filtering and Filter Design

Waveform Generation

Window Functions

Spectral Analysis

Convolve/Correlate  
FFT Convolve  
Convolve/Correlate 2D

Resampling - Polyphase, Upfirdn, Resample  
Hilbert/Hilbert 2D  
Wiener  
Firwin

Chirp  
Square  
Gaussian Pulse

Kaiser  
Blackman  
Hamming  
Hanning

Periodogram  
Welch  
Spectrogram

# Speed of Light Performance - V100

*timeit* (7 runs) rather than *time*. Benchmarked with ~1e8 sample signals on a DGX Station

Method	Scipy Signal (ms)	cuSignal (ms)	Speedup (xN)
fftconvolve	27300	85.1	320.8
correlate	4020	47.4	84.8
resample	14700	45.9	320.2
resample_poly	2360	8.29	284.6
welch	4870	45.5	107.0
spectrogram	2520	23.3	108.1
convolve2d	8410	9.92	847.7

Learn more about cuSignal functionality and performance by browsing the [notebooks](#)

# Efficient Memory Handling

## Seamless Data Handoff from cuSignal to PyTorch >=1.4

Leveraging the `__cuda_array_interface__` for Speed of Light End-to-End Performance

```
from numba import cuda
import cupy as cp
import torch
from cusignal import resample_poly

# Create CuPy Array on GPU
gpu_arr = cp.random.randn(100_000_000, dtype=cp.float32)

# Polyphase Resample
resamp_arr = resample_poly(gpu_arr, up=2, down=3, window=('kaiser', 0.5))

# Zero Copy to PyTorch
torch_arr = torch.as_tensor(resamp_arr, device='cuda')

# Confirm Pointers
print('Resample Array: ', resamp_arr.__cuda_array_interface__['data'])
print('Torch Array: ', torch_arr.__cuda_array_interface__['data'])

Resample Array: (140516096213080, False)
Torch Array: (140516096213080, False)
```

## Enabling Online Signal Processing with Zero-Copy Memory

CPU <-> GPU Direct Memory Access with Numba's Mapped Array

```
import numpy as np
import cupy as cp
from numba import cuda
import cusignal

# Create CPU/GPU Shared Memory, similar to numpy.zeros()
N = 2**18
shared_arr = cusignal.get_shared_mem(N, dtype=np.complex64)
print('CPU Pointer: ', shared_arr.__array_interface__['data'])
print('GPU Pointer: ', shared_arr.__cuda_array_interface__['data'])

CPU Pointer: (139895179837440, False)
GPU Pointer: (139895179837440, False)

# Load Shared Array with Numpy Array
shared_arr = np.random.randn(N) + 1j*np.random.randn(N)

%%timeit
# Perform CPU FFT
cpu_fft = np.fft.fft(shared_arr)
8 ms ± 60.2 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)

%%timeit
# Perform GPU FFT
gpu_fft = cp.fft.fft(cp.asarray(shared_arr))
cp.cuda.Device(0).synchronize()

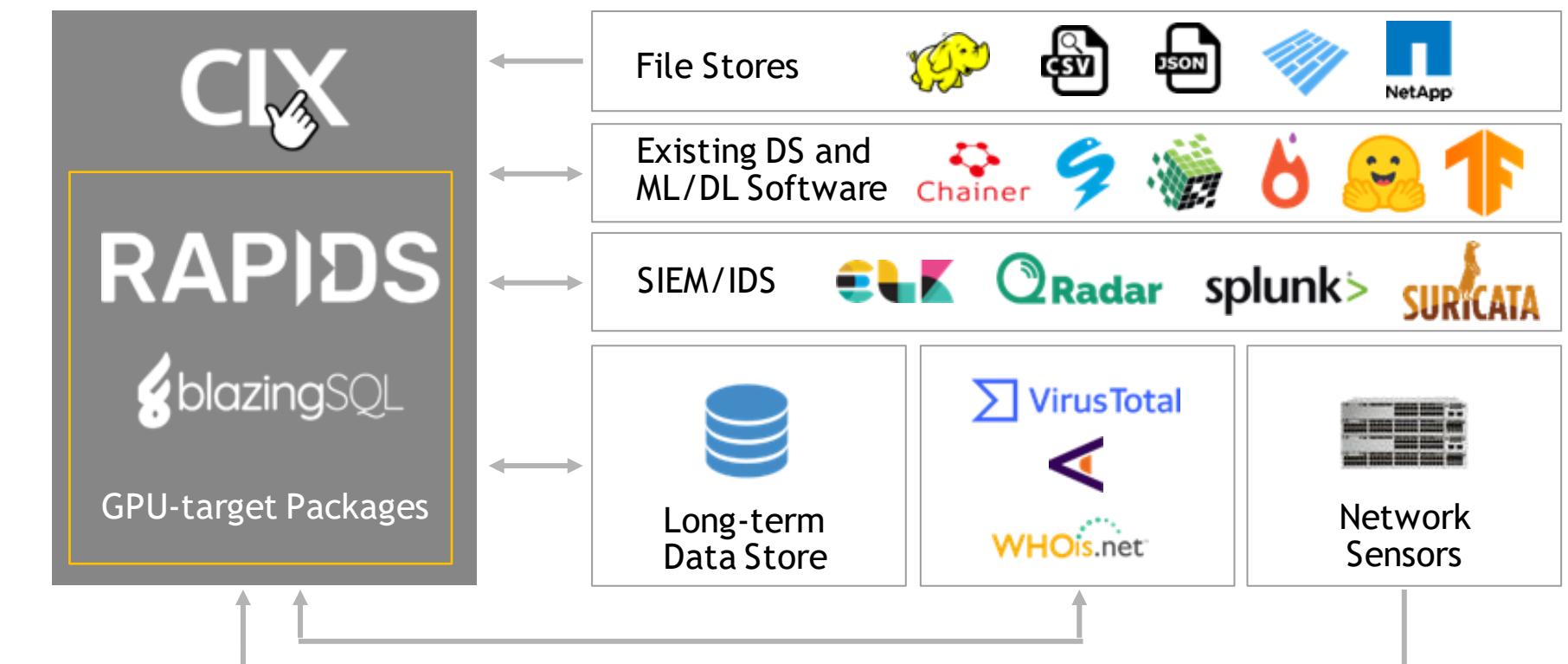
866 µs ± 38 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)
```

# Cyber Log Accelerators

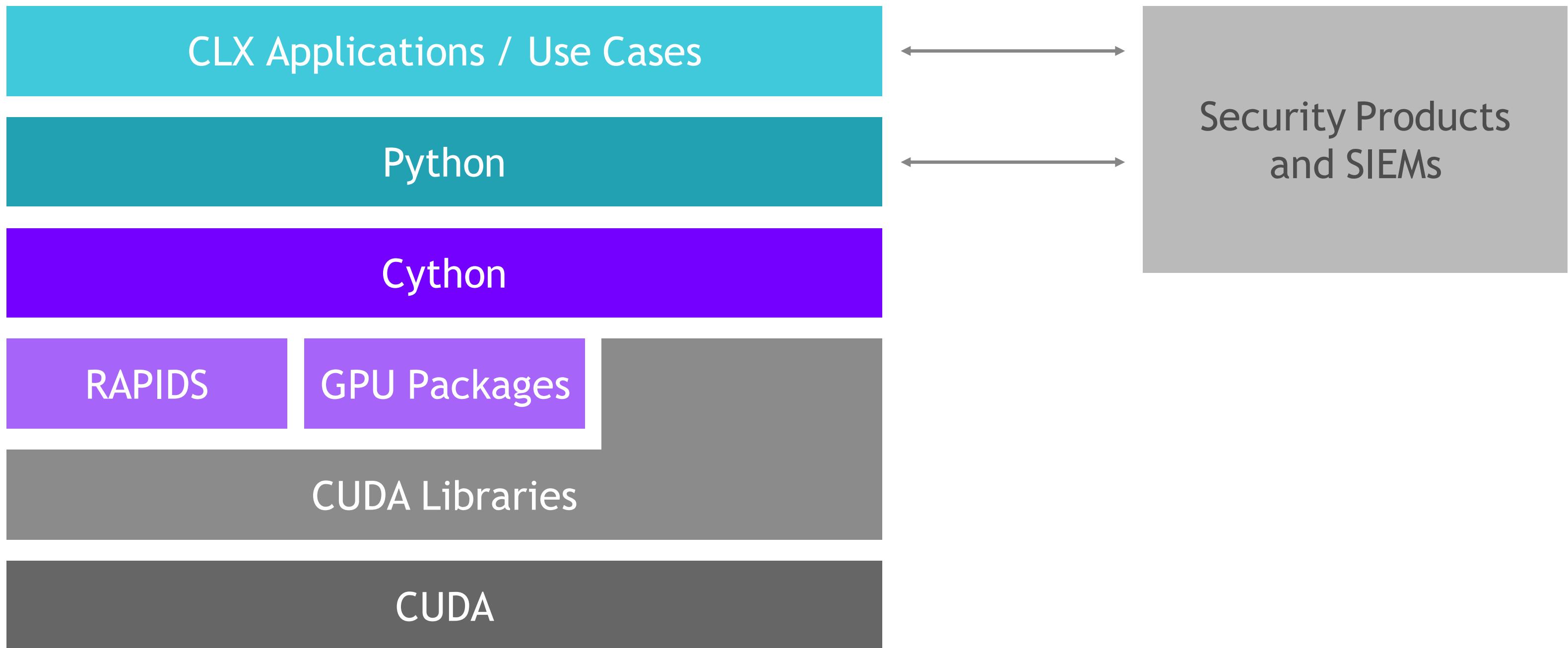
# CLX

## Cyber Log Accelerators

- ▶ Built using RAPIDS and GPU-accelerated platforms
- ▶ Targeted towards Senior SOC (Security Operations Center) Analysts, InfoSec Data Scientists, Threat Hunters, and Forensic Investigators
- ▶ Notebooks geared towards info sec and cybersecurity data scientists and data engineer
- ▶ SIEM integrations that enable easy data import/export and data access
- ▶ Workflow and I/O components that enable users to instantiate new use cases while
- ▶ Cyber-specific primitives that provide accelerated functions across cyber data types



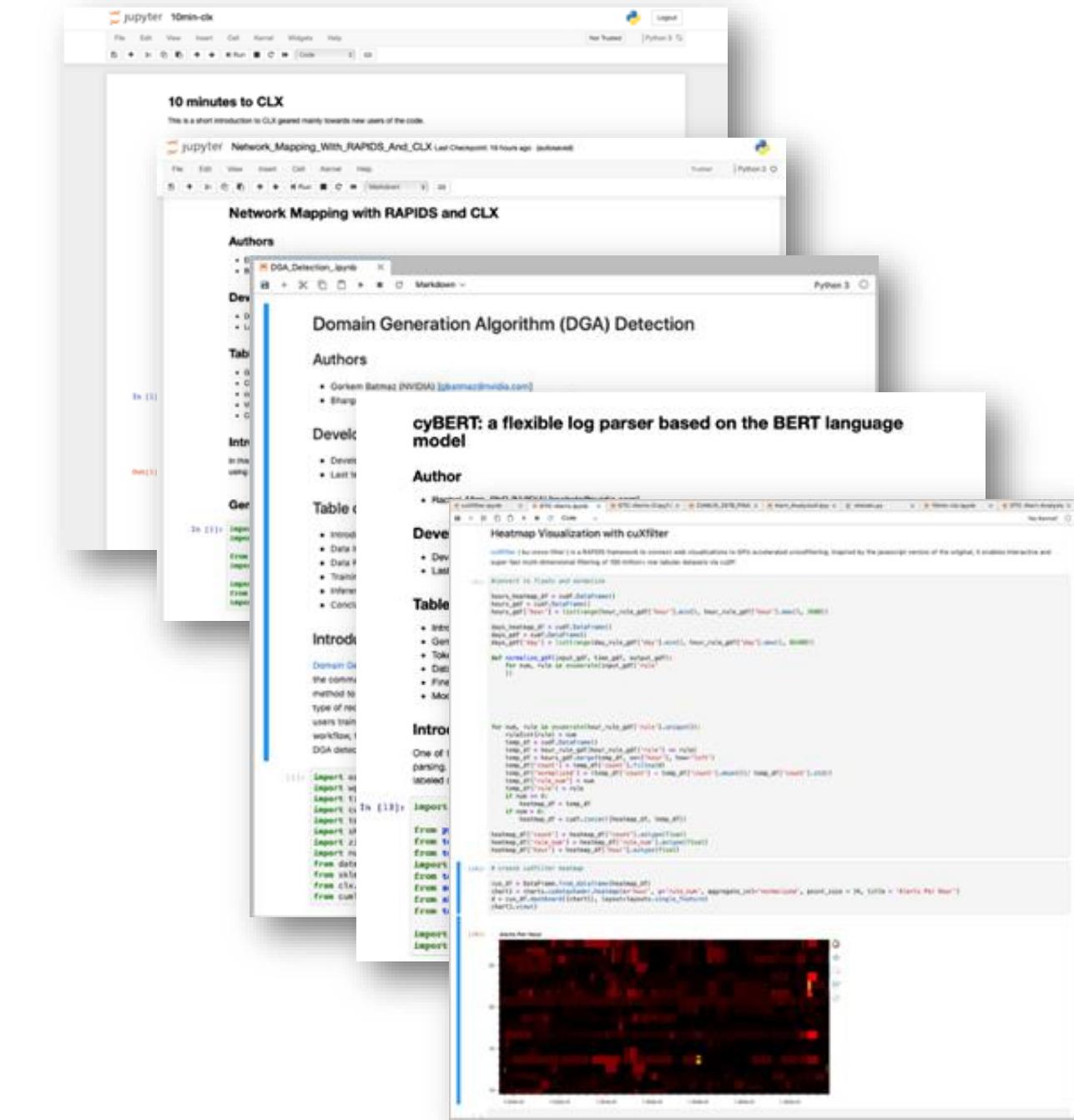
# CLX Technology Stack



# CLX Contains Various Use Cases and Connectors

## Example Notebooks Demonstrate RAPIDS for Cybersecurity Applications

CLX	Type	Proof-of-Concept	Stable
DGA Detection	Use Case		
Network Mapping	Use Case		
Asset Classification	Use Case		
Phishing Detection	Use Case		
Security Alert Analysis	Use Case		
EDA for Cyber	Use Case	NEW	NEW
Predictive Maintenance	Use Case	NEW	
LODA	Method	NEW	NEW
Sequence Classifier	Method	NEW	NEW
Splunk Integration	Integration		
CLX Query	Integration		
cyBERT	Log Parsing		
GPU SubwordTokenizer	Pre-Processing		
Accelerated IPv4	Primitive		
Accelerated DNS	Primitive		
Cyber-specific Docker	Docker Image	NEW	NEW
3rd Party APIs	API	Added SlashNext	

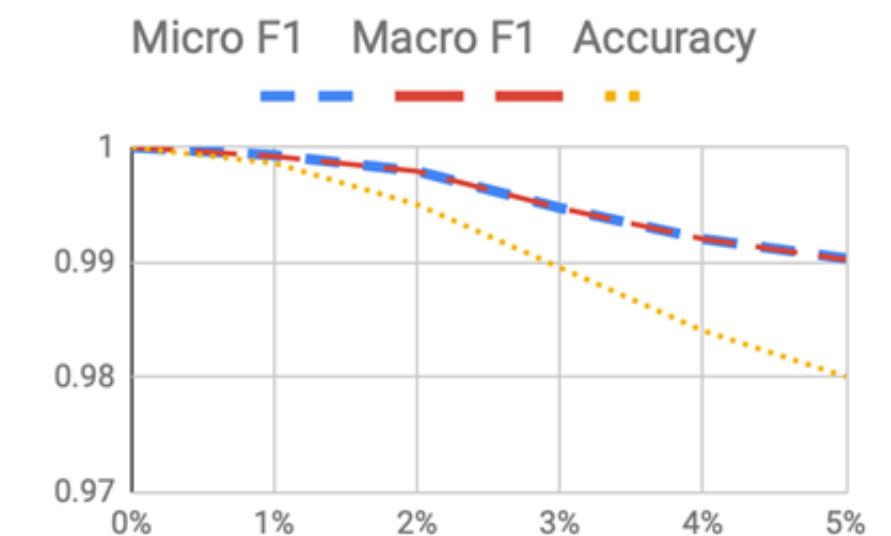


# cyBERT

Now for log parsing, PII detection, predictive maintenance, and more

- ▶ Utilizes NLP techniques and transformer architectures to accomplish cybersecurity tasks
- ▶ cyBERT for log parsing can extract fields of value (without regex) with a **micro-F1** and **macro-F1 > 0.999** across heterogeneous log types with a **validation loss of < 0.0048**, and is **160x faster** than previous versions
- ▶ PII (Personally Identifying Information) discovery and scrubbing can be accomplished with a fine-tuned BERT model. **F1 score > 0.9643** to identify PII fields compared to 0.281 using the IT spaCy NER implementation
- ▶ NLP models and sequence classification are useful in predictive maintenance tasks, where a model was able to identify **over 100 additional lines** in unstructured logs that are possibility sources of errors. These lines would have been missed by static regex rules.

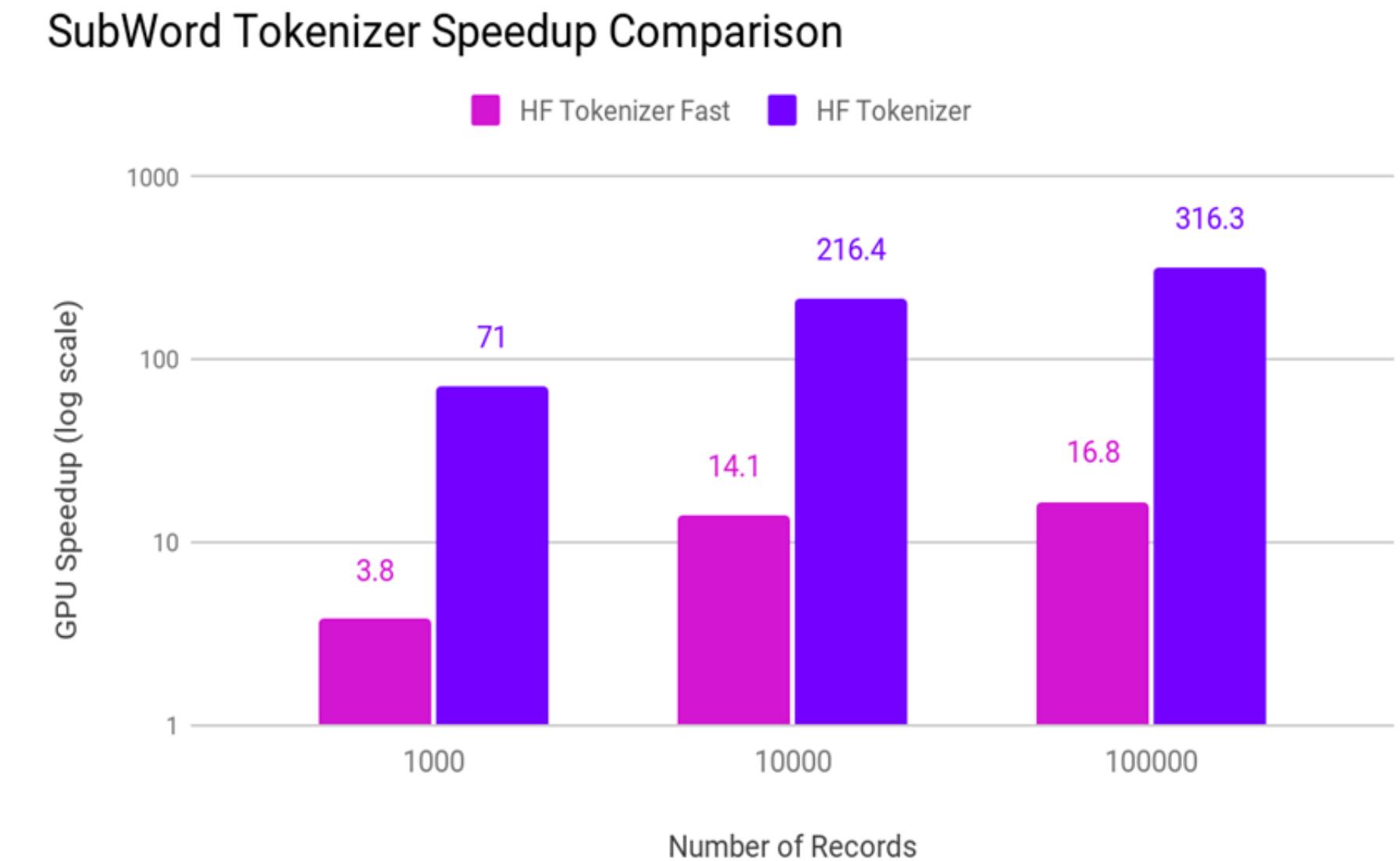
## Log Parsing with Missing Values



# GPU SubWord Tokenizer

## Fully On-GPU Pre-Processing for BERT Training/Inference

- ▶ Only wordpiece tokenizer that supports non-truncation of logs/sentences
- ▶ Returns encoded tensor, attention mask, and metadata to reform broken logs
- ▶ Supports stride/overlap
- ▶ Ready for immediate pipelining into PyTorch for inference
- ▶ Over 316x faster than Python-based Hugging Face tokenizer
- ▶ Nearly 17x faster than new Rust-based Hugging Face tokenizer



CPU numbers ran on 2x Intel Xeon E5 v4 @ 2.2 GHz. GPU numbers ran on 1x NVIDIA Tesla V100 using CLX v0.17 nightly

# CLX Query

## Query Long-Term Data Store Directly from Splunk

The screenshot shows the Splunk interface with the following details:

- Top Bar:** splunk > App: Clx Search > DEVTEST
- Search Bar:** | clx query="SELECT domain, count(\*) as cnt FROM main.dga GROUP BY domain ORDER BY cnt DESC LIMIT 10000" | table domain, cnt
- Results Summary:** 10,000 results (10/29/19 1:00:00.000 PM to 10/30/19 1:31:21.000 PM) No Event Sampling
- Event List:** A list of domains with their counts, such as pdcqcuurgsj.sj.pw (59), wcobovuasbujs.cc (53), lketwyitpggxsj.sj (53), rfpusradccctv.pw (53), xrqtvsekvdwxw.tw (53), hnbdxgbcffdv.cc (52), oyofyqihjonvfr.cc (52), gflxihxkpegps.in (52), kucpakwrcilrn.me (52), ewgħtlpjntivpy.cc (52), hvfgeoippqpqs.me (52), nfjuwiesyupbso.in (54), pwgxuuwaitjkb.com (53), lapkgfnbfheveb.me (53), ordqsnkdvwxehp.tw (52), nkrxjgtbeivmi.su (52), and eskygbpjotbuc.su (52).

RAPIDS



CLX

# Visualization

# RAPIDS cuxfilter

## GPU Accelerated Cross-Filtering

### STREAMLINED FOR NOTEBOOKS

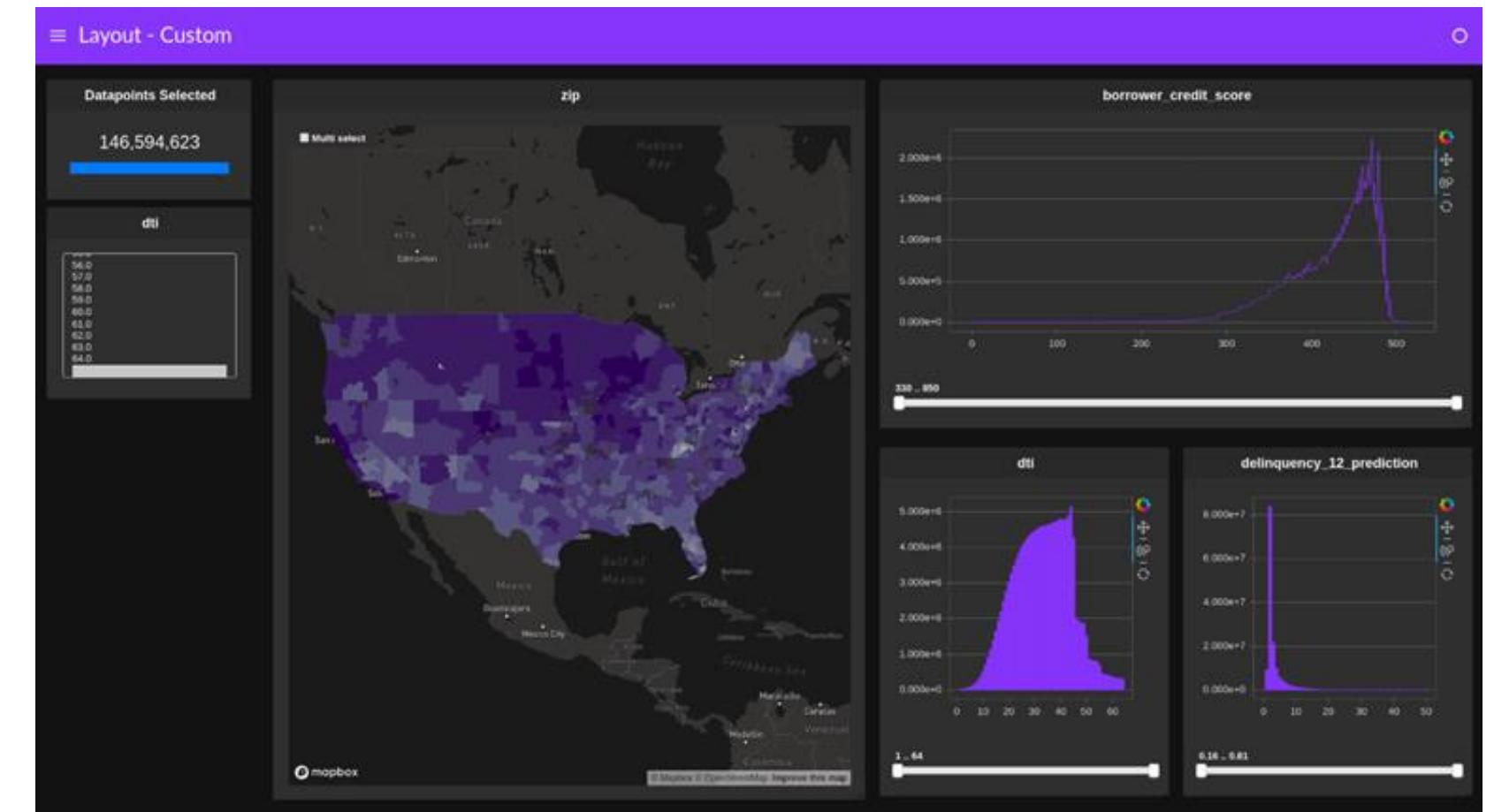
Cuxfilter allows you to visually explore your cuDF dataframes through fully cross-filtered dashboards in less than 10 lines of notebook code.

### MINIMAL COMPLEXITY & FAST RESULTS

Select from vetted chart libraries, customizable layouts, and multiple themes to find the best fit for a use case, at speeds typically 20x faster per chart than Pandas.

### SEAMLESS INTEGRATION WITH RAPIDS

Cuxfilter is designed for easy [install](#) and use with RAPIDS. Learn more about our approach [here](#).



<https://docs.rapids.ai/api/cuxfilter/stable>

# Plotly Dash

## GPU Accelerated Visualization Apps with Python



Plotly Dash is RAPIDS accelerated, combining the ease-of-use development and deployment of Dash apps with the compute speed of RAPIDS. Work continues to optimize Plotly's API for even deeper RAPIDS integration.

### 300 MILLION DATAPOINT CENSUS EXAMPLE

Interact with data points of every individual in the United States, in real time, with the 2010 Census visualization. Get the code on [GitHub](#) and read about details [here](#).



<https://github.com/rapidsai/plotly-dash-rapids-census-demo>

# pyViz Community

## GPU Accelerated Python Visualizations



Uses cuDF to easily annotate and interactively explore data with minimal syntax. Work continues to optimize its API for deeper RAPIDS integration.

[holoviews.org](https://holoviews.org)



Uses cuDF / dask cuDF for accelerated server-side rendering of extremely high density visualizations. Work continues to optimize more of its features for GPUs.

[datashader.org](https://datashader.org)



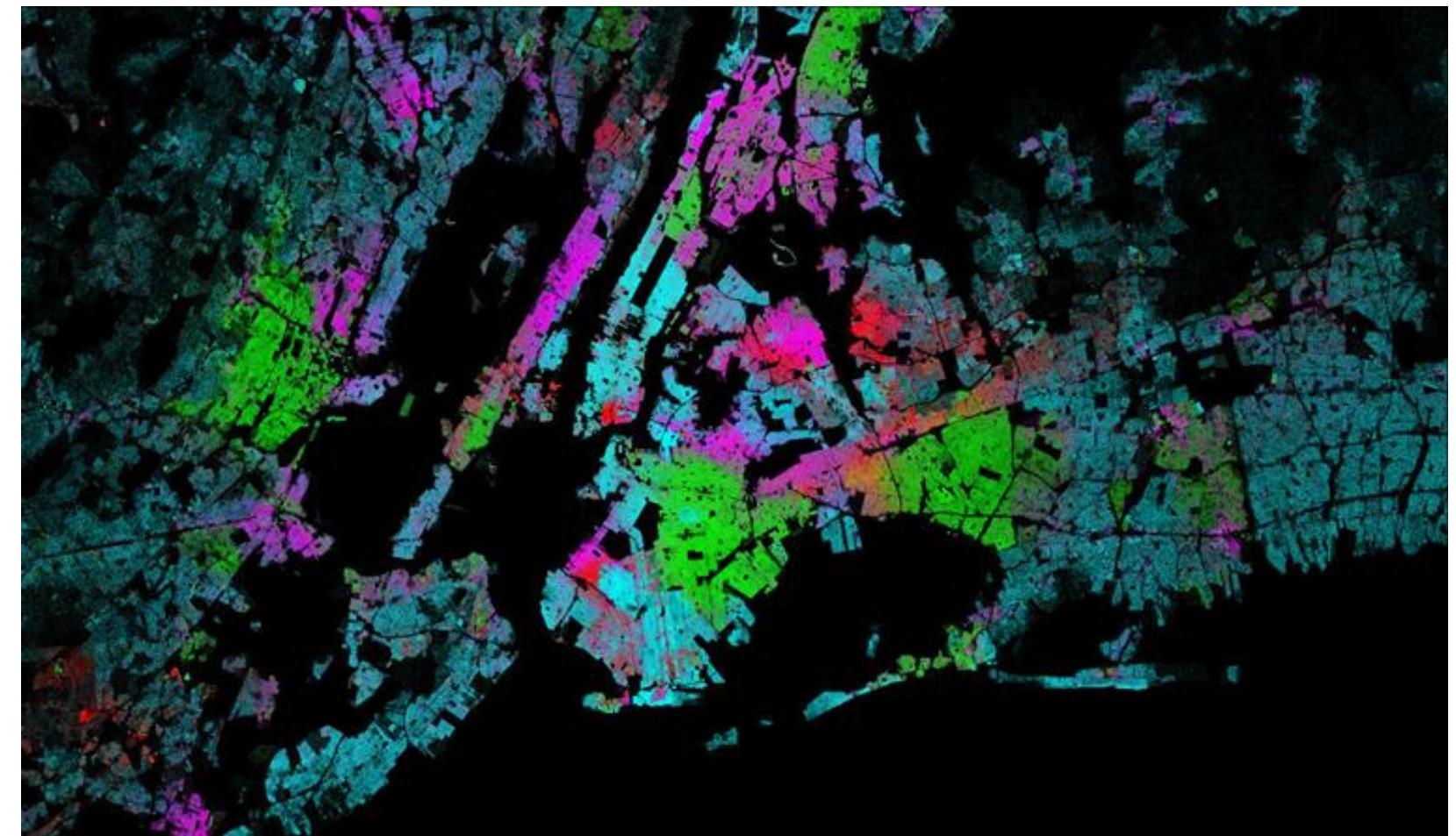
A higher-level plotting API built on HoloViews, work is ongoing to ensure its features can utilize the RAPIDS integration with HoloViews.

[hvplot.holoviz.org](https://hvplot.holoviz.org)



A backbone chart library used throughout the pyViz community, RAPIDS is actively supporting integration development to further its use in GPU accelerated libraries and enhance rendering performance.

[bokeh.org](https://bokeh.org)



<https://datashader.org/>

# Community

# Ecosystem Partners

## CONTRIBUTORS

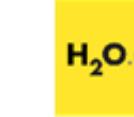


## ADOPTERS

Booz | Allen | Hamilton



Walmart Global Tech



## OPEN SOURCE



# Building on Top of RAPIDS

A Bigger, Better, Stronger Ecosystem for All

NVTabular

ETL library for recommender systems  
building off of RAPIDS and Dask



GPU accelerated SQL engine  
built on top of RAPIDS

Streamz

Distributed stream processing  
using RAPIDS and Dask

# NVTabular

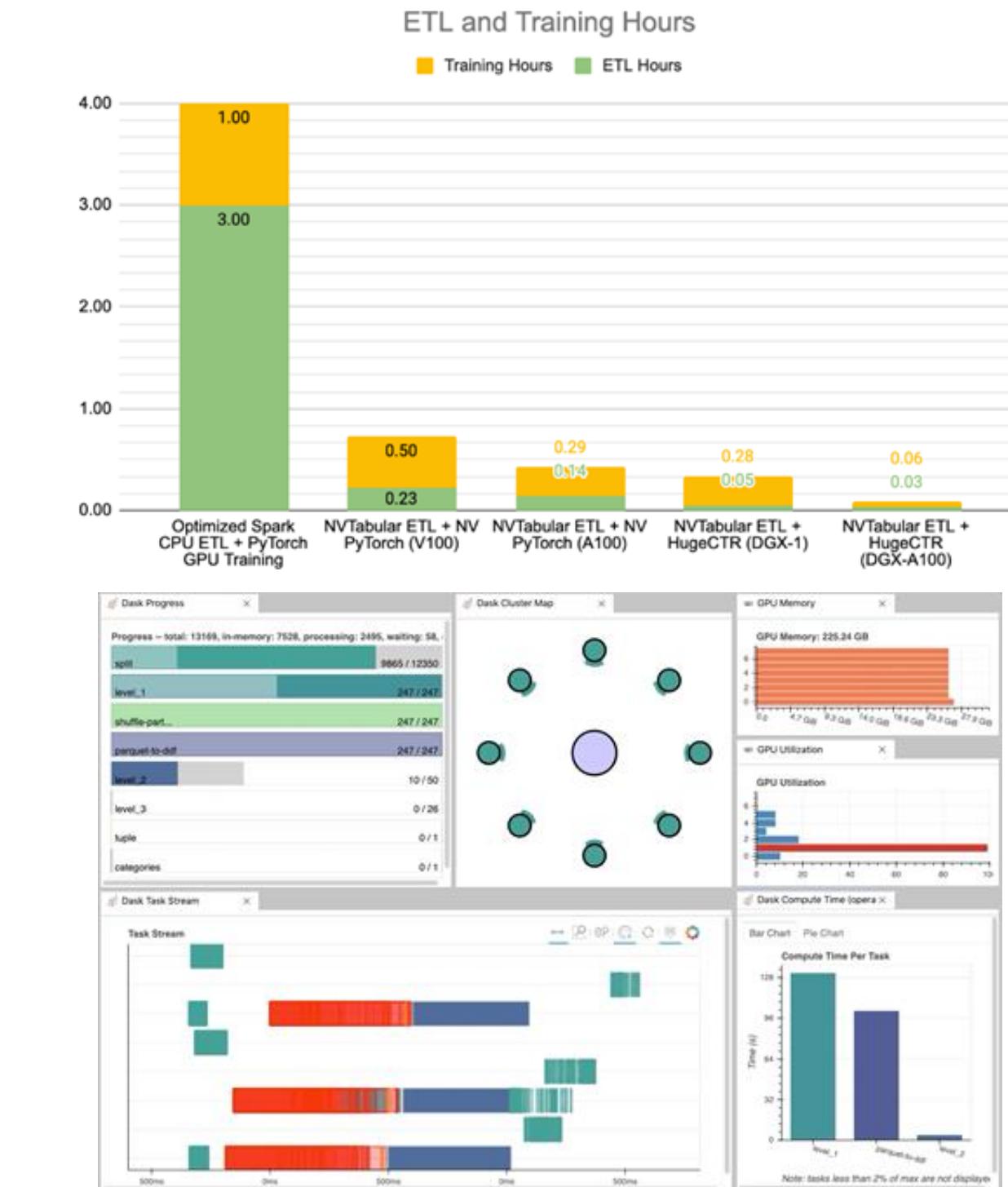
## ETL library for recommender systems building off of RAPIDS and Dask

### A HIGH LEVEL API BUILDING UPON DASK-CUDF

NVTabular's high level API allows users to think about what they want to do with the data, not how they have to do it or how to scale it, for operations common within recommendation workflows.

### ACCELERATED GPU DATALOADERS

Using cuIO primitives and cuDF, NVTabular accelerates dataloading for PyTorch & Tensorflow, removing I/O issues common in deep learning based recommender system models.



# BlazingSQL

## GPU-accelerated SQL engine built with RAPIDS

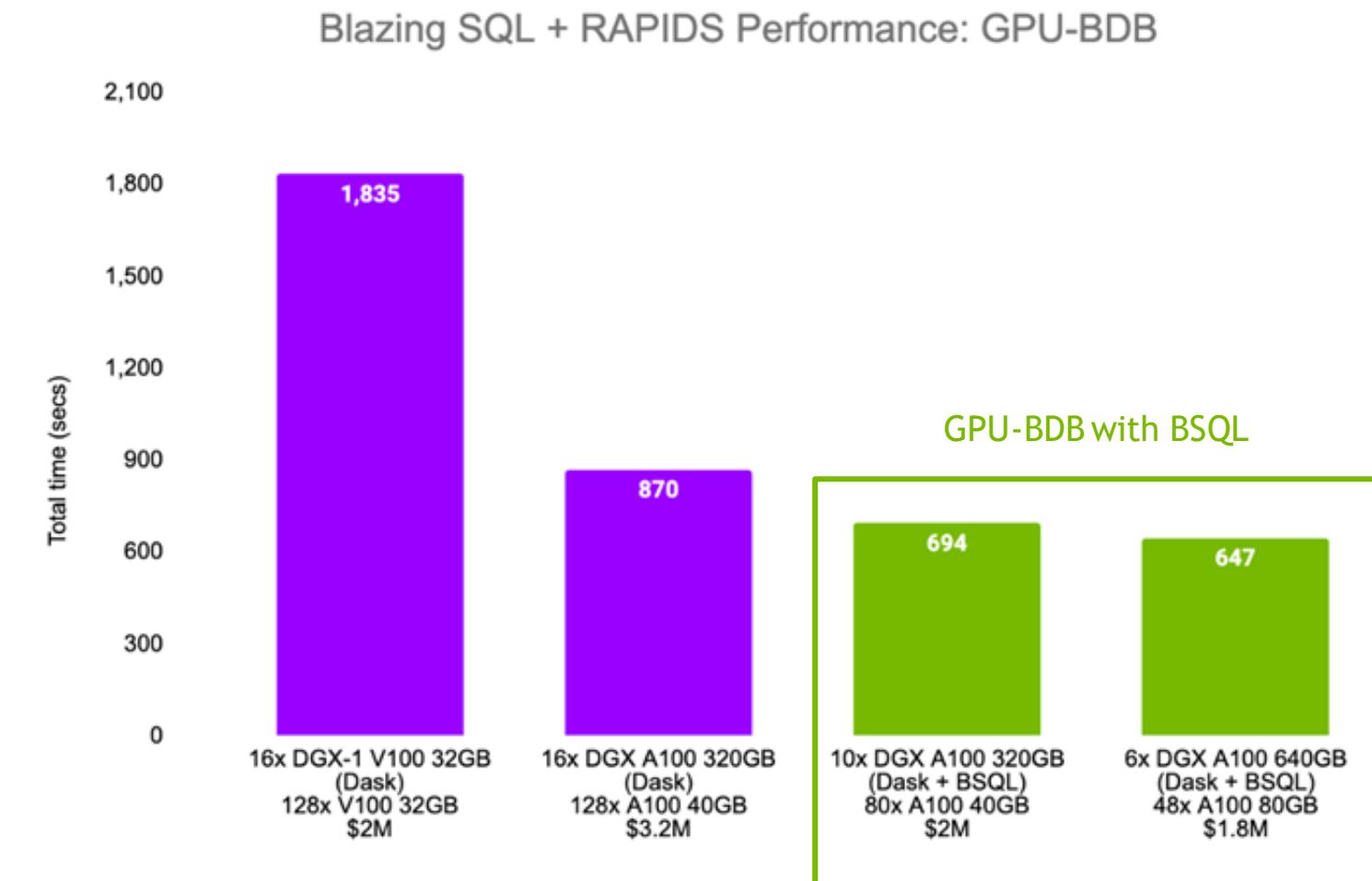
### BLAZING FAST SQL ON RAPIDS

- ▶ Incredibly fast distributed SQL engine on GPUs--natively compatible with RAPIDS!
- ▶ Allows data scientists to easily connect large-scale data lakes to GPU-accelerated analytics
- ▶ Directly query raw file formats such as CSV and Apache Parquet inside Data Lakes like HDFS and AWS S3, and directly pipe the results into GPU memory.

### NOW WITH OUT-OF-CORE EXECUTION

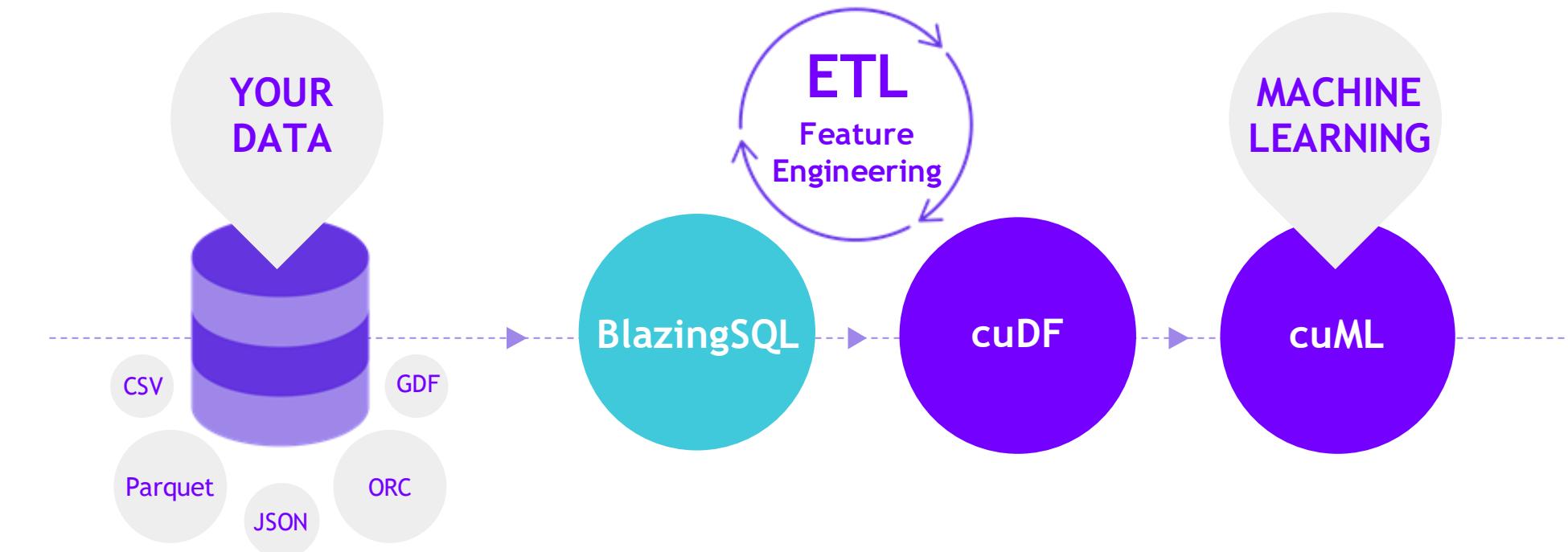
- ▶ Users no longer limited by available GPU memory
- ▶ 10TB workloads on a single Tesla V100 (32GB)!

1: GPU-BDB is derived from the TPCx-BB benchmark and is used for internal performance testing. Results from GPU-BDB are not comparable to TPCx-BB.



- ▶ GPU Big Data Benchmark (GPU-BDB<sup>1</sup>) is a data science benchmark consisting of 30 end-to-end queries representing real-world ETL and Machine Learning workflows, involving both structured and unstructured data.
- ▶ Using BlazingSQL, we saw a **20% increase in performance** - on **6 fewer nodes**!

# BlazingSQL



```
from blazingsql import BlazingContext
import cudf

bc = BlazingContext()

bc.s('bsql', bucket_name='bsql', access_key_id='<access_key>', secret_key='<secret_key>')

bc.create_table('orders', s3://bsql/orders/')

gdf = bc.sql('select * from orders').get()
```

# cuStreamz

## Stream processing powered by RAPIDS

### ACCELERATED KAFKA CONSUMER

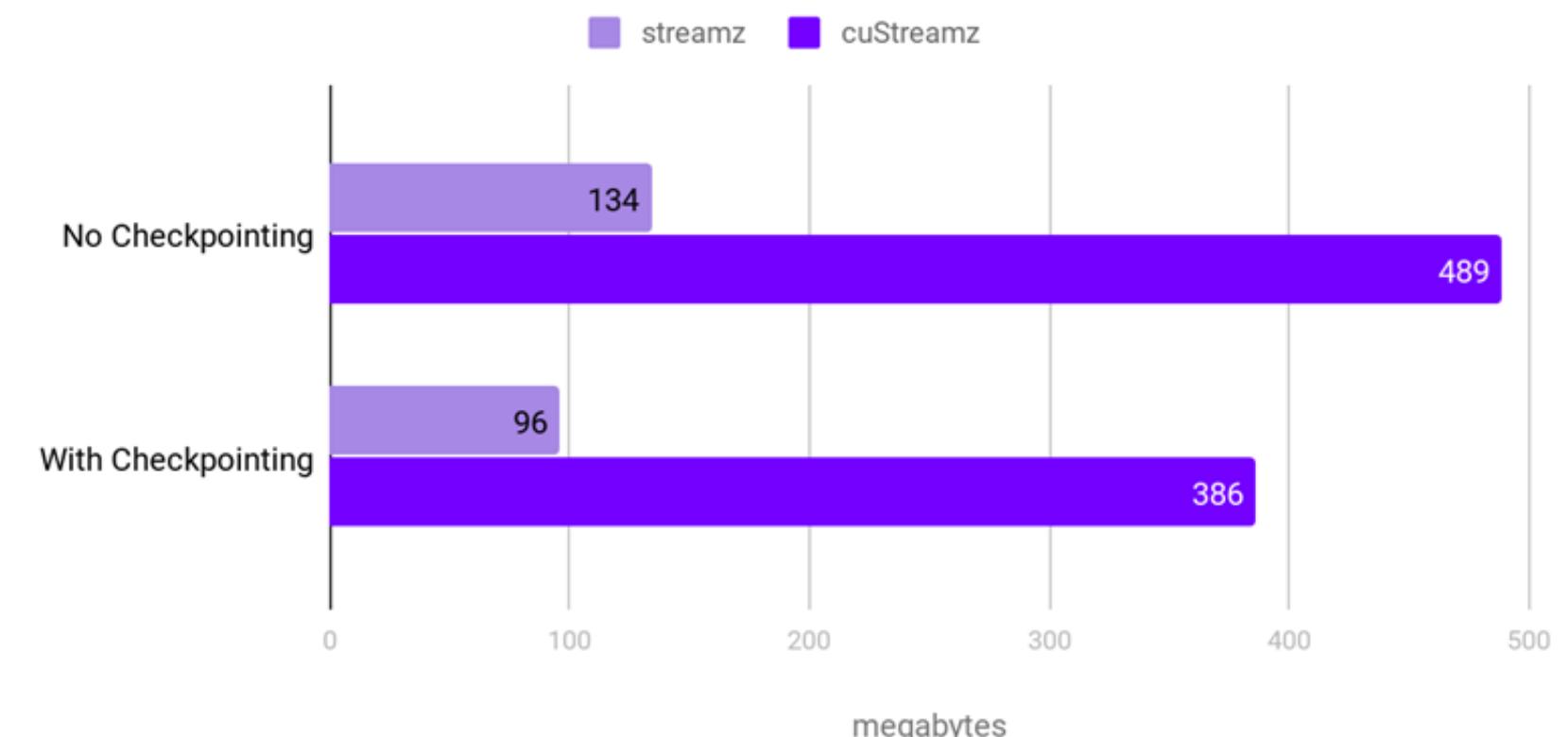
Ingesting Kafka messages to cudf is increased by roughly 3.5 - 4X over standard cpu ingestion. Streaming TCO is lowered by using cheaper VM instance types.

### CHECKPOINTING

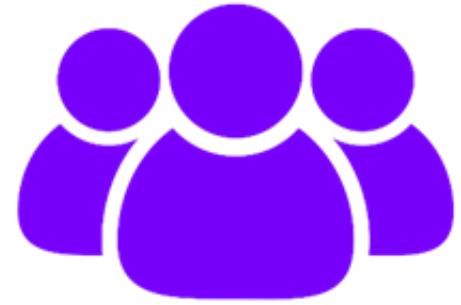
Streaming job version of “where was I?” Streams can gracefully handle errors by continuing to process a stream at the exact point they were before the error occurred.

Kafka throughput

Single Node

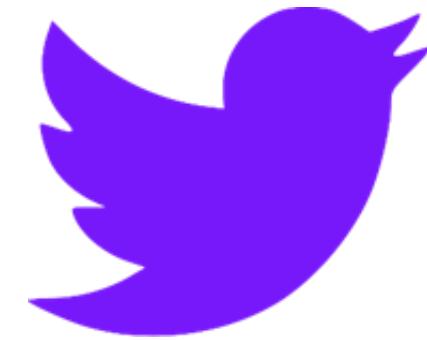


# Join the Conversation



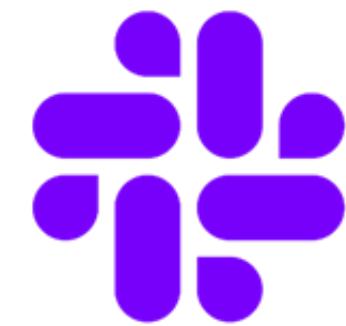
## GOOGLE GROUPS

[https://groups.google.com/  
forum/#!forum/rapidsai](https://groups.google.com/forum/#!forum/rapidsai)



## TWITTER

<https://twitter.com/RAPIDSai>



## SLACK CHANNEL

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



## STACK OVERFLOW

[https://stackoverflow.com/  
tags/rapids](https://stackoverflow.com/tags/rapids)

# Contribute Back

## Issues, Feature Requests, PRs, Blogs, Tutorials, Videos, QA...Bring Your Best!

**cuml**  
cuML - RAPIDS Machine Learning Library

machine-learning gpu machine-learning-algorithms

cuda nvidia

● C++ Apache-2.0 111 ★ 608 ⓘ 186 (26 issues need help) 31 Updated 9 minutes ago



---

**cudf**  
cuDF - GPU DataFrame Library

anaconda gpu arrow machine-learning-algorithms

h2o cuda pandas

● Cuda Apache-2.0 250 ★ 1,699 ⓘ 325 (6 issues need help) 41 Updated 31 minutes ago



---

**notebooks-contrib**  
RAPIDS Community Notebooks

Jupyter Notebook Apache-2.0 56 ★ 70 ⓘ 10 (1 issue needs help) 8 Updated 40 minutes ago



John Murray [Follow](#)

TECH BLOG Walmart Labs ENGINEERING DATA SCIENCE INFOSEC UX DESIGN LEADER

Comparison CPU vs GPU @rapidsai to project 100 million x,y points to lat/lon to 0.01mm accuracy. CPU 1 core c 65 mins, multicore c 13 mins, GPU #RAPIDS 2 seconds. I optimised the code since previous run. Dell T7910 Xeon E5-2640V4x2/NVIDIA Titan Xp cc @NvidiaAI @marc\_stampfli

```
john@plato:~/Source/Python/misc$ python crs_test.py
Generating Data
CPU Iterative
4005.0377202 seconds
CPU mapped
3957.19386101 seconds
CPU multiprocessing
788.550751209 seconds
GPU Rapids
2.103230476 seconds
```

How GPU Computing literally saved me at work?  
Python+GPU = Power, 2 Days to 20 seconds

Abhishek Mungali [Follow](#)  
May 9, 2019 · 9 min read



## Getting Started with cuDF (RAPIDS)

Darren Ramsook [Follow](#)  
Jun 9 · 3 min read

# RAPIDS Notices

Communicate and document changes to RAPIDS for contributors, core developers, users, and the community

Notice	Title	Topic	RAPIDS Version	Updated
RDN 1 <b>COMPLETED</b>	<a href="#">'dask-xgboost' is deprecated in v0.15 &amp; removed v0.16</a>	Library Deprecation	v0.15	26 August 2020
RGN 1 <b>IN PROGRESS</b>	<a href="#">Stable/Release Branch Renaming to 'main' in v0.15</a>	Git Repo Change	v0.15	26 August 2020
RGN 2 <b>COMPLETED</b>	<a href="#">v0.15 No CUDA 11 Release for 'clx'</a>	Release Change	v0.15	26 August 2020
RGN 4 <b>IN PROGRESS</b>	<a href="#">v0.15 Release Delay for 'cuxfilter'</a>	Release Change	v0.15	26 August 2020
RSN 2 <b>COMPLETED</b>	<a href="#">EOL Python 3.6 &amp; CUDA 10.0 in v0.14</a>	Platform Support Change	v0.14 & v0.15	13 July 2020
RSN 3 <b>COMPLETED</b>	<a href="#">Support for Python 3.8 in v0.15</a>	Platform Support Change	v0.15	17 July 2020
RSN 4 <b>COMPLETED</b>	<a href="#">Support for CUDA 11.0 in v0.15</a>	Platform Support Change	v0.15	26 August 2020

<https://docs.rapids.ai/notices>

# Getting Started

# 5 Steps to Getting Started with RAPIDS

1. Install RAPIDS on [Docker](#), [Conda](#), [deploy in your favorite cloud instance](#), or quick start with [app.blazingsql.com](#).
2. Explore our [walk through videos](#), [blog content](#), our [github](#), the [tutorial notebooks](#), and our [example workflows](#).
3. Build your own data science workflows.
4. Join our community conversations on [Slack](#), [Google](#), and [Twitter](#).
5. Contribute back. Don't forget to ask and answer questions on [Stack Overflow](#).

# 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

The image displays two screenshots of the cuDF documentation website, illustrating its structure and content.

**Left Screenshot:** This is the '10 Minutes to cuDF and Dask-cuDF' page. The navigation bar on the left includes links for Home, cuDF, clx, cudf, cuGraph, cuml, cusignal, cuspatial, cuXfilter, libcudf, libcuml, libnvstrings, nvstrings, and rmm. The main content area has a heading '10 Minutes to cuDF and Dask-cuDF' and a sub-section 'What are these Libraries?'. It contains text about cuDF and Dask, and a code block showing Python imports for numpy, pandas, and cuDF, followed by a note about borrowing from cuDF's cheatsheet. Below this is a section titled 'Object Creation' with a code snippet for creating a cuDF.Series object.

**Right Screenshot:** This is the 'Welcome to cuDF's documentation!' page. The navigation bar on the left includes links for Home, cuDF, and stable (0.13). The main content area features a heading 'Welcome to cuDF's documentation!', a 'Contents:' sidebar with sections like API Reference, 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, and a large list of topics under '10 Minutes to cuDF and Dask-cuDF' such as 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, and so on.

<https://docs.rapids.ai>

# RAPIDS Docs

## Easier than Ever to Get Started with cuDF

The screenshot shows a web browser displaying the RAPIDS Docs website. The left sidebar has a navigation menu with items like Home, cuDF, clx, cugraph, cuml, cusignal, cuspatial, cuxfilter, libcudf, libcuml, libnvstrings, nvstrings, and rmm. The 'cuDF' item is currently selected and highlighted in purple. The main content area is titled '10 Minutes to cuDF and Dask-cuDF'. It contains sections on what cuDF is, how to use it, and code snippets for creating cuDF Series objects.

**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(123)

##### 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,NaN,4])
s
```

```
[2]: #      1
```

<https://docs.rapids.ai>

# Explore: RAPIDS Code and Blogs

## Check out our Code and How We Use It

The screenshot shows the README.md file for the cuDF repository on GitHub. The title is "RAPIDS cuDF - GPU DataFrames". Below the title, there are two buttons: "build" and "running". A note at the top says: "NOTE: For the latest stable README.md ensure you are on the master branch." The main text describes cuDF as a GPU DataFrame library for loading, joining, aggregating, filtering, and otherwise manipulating data. It provides a pandas-like API for data engineers and scientists. An example code snippet is provided to demonstrate reading a CSV file and performing calculations:

```
import cudf, io, requests
from io import StringIO

url="https://github.com/plotly/datasets/raw/master/tips.csv"
content = requests.get(url).content.decode('utf-8')

tips_df = cudf.read_csv(StringIO(content))
tips_df['tip_percentage'] = tips_df['tip']/tips_df['total_bill']*100

# display average tip by dining party size
print(tips_df.groupby('size').tip_percentage.mean())
```

Output:

<https://github.com/rapidsai>

The screenshot shows the RAPIDS AI Medium blog page. The header features the RAPIDS logo and the text "RAPIDS AI Open GPU Data Science". The navigation bar includes links for RAPIDS RELEASES, DATAFRAMES, MACHINE LEARNING, DATA VISUALIZATION, GRAPH ANALYTICS, LEARN MORE, and FOLLOW. The main content area displays several blog posts with their titles, authors, and brief descriptions:

- Reading Larger than Memory CSVs with RAPIDS and Dask** by Nick Becker (Oct 22 - 2 min read) - Describes how RAPIDS and Dask make it easy to load larger than memory datasets.
- RAPIDS CuGraph: NetworkX Compatibility** by Brad Bees (Oct 7 - 4 min read) - Discusses adding NetworkX Graph and DiGraph objects as valid input data types for graph algorithms.
- Tutorial: Hyperparameter Optimization (HPO) with RAPIDS on AWS SageMaker** by Mira Enav (Sep 28 - 2 min read) - Details a 12x speedup in wall clock time and 4.5x reduction in cost when comparing GPU to CPU running HPO jobs in SageMaker.
- RAPIDS Anywhere with TailScale—My Mobile Device has an RTX 3090** by Josh Patterson (Sep 26 - 5 min read) - Shows how it's easier to get started with GPUs and RAPIDS, and with TailScale, you can kick off RAPIDS workflows from anywhere.

<https://medium.com/rapids-ai>

# Explore: RAPIDS Github

The screenshot shows the GitHub profile for the RAPIDS organization. The profile features a purple logo with the word "RAPIDS" in white. Below the logo, the text "Open GPU Data Science" is displayed, along with links to the website (<http://rapids.ai>) and Twitter (@RAPIDSai). The navigation bar at the top includes links for Pull requests, Issues, Marketplace, and Explore.

Below the navigation bar, there are tabs for Repositories (123), Packages, People (186), Teams (199), and Projects (5). The "Repositories" tab is selected, indicated by a red underline.

The main content area is titled "Pinned repositories" and lists six repositories:

- cudf**: cuDF - GPU DataFrame Library. Language: C++. Stars: 3.6k. Forks: 483.
- gpu-bdb**: RAPIDS GPU-BDB. Language: Python. Stars: 63. Forks: 27.
- cuml**: cuML - RAPIDS Machine Learning Library. Language: Cuda. Stars: 1.9k. Forks: 305.
- cugraph**: cuGraph - RAPIDS Graph Analytics Library. Language: Jupyter Notebook. Stars: 638. Forks: 138.
- cusignal**: cuSignal - RAPIDS Signal Processing Library. Language: Python. Stars: 440. Forks: 60.
- cuspatial**: CUDA-accelerated GIS and spatiotemporal algorithms. Language: Cuda. Stars: 218. Forks: 86.

<https://github.com/rapidsai>

# Explore: RAPIDS Community Notebooks

intro_tutorials	<a href="#">05_Introduction_to_Dask_cuDF</a>	This notebook shows how to work with cuDF DataFrames distributed across multiple GPUs using Dask.
intro_tutorials	<a href="#">06_Introduction_to_Supervised_Learning</a>	This notebook shows how to do GPU accelerated Supervised Learning in RAPIDS.
intro_tutorials	<a href="#">07_Introduction_to_XGBoost</a>	This notebook shows how to work with GPU accelerated XGBoost in RAPIDS.
intro_tutorials	<a href="#">08_Introduction_to_Dask_XGBoost</a>	This notebook shows how to work with Dask XGBoost in RAPIDS.
intro_tutorials	<a href="#">09_Introduction_to_Dimensionality_Reduction</a>	This notebook shows how to do GPU accelerated Dimensionality Reduction in RAPIDS.
intro_tutorials	<a href="#">10_Introduction_to_Clustering</a>	This notebook shows how to do GPU accelerated Clustering in RAPIDS.

**Intermediate Notebooks:**

Folder	Notebook Title	Description
examples	<a href="#">DBSCAN_Demo_FULL</a>	This notebook shows how to use DBSCAN algorithm and its GPU accelerated implementation present in RAPIDS.
examples	<a href="#">Dask_with_cuDF_and_XGBoost</a>	In this notebook we show how to quickly setup Dask and train an XGBoost model using cuDF.

The screenshot shows the RAPIDS YouTube channel page. At the top, there's a purple header bar with the channel name "RAPIDS" and a circular profile picture. Below the header, the channel has 333 subscribers. The main navigation menu includes HOME, VIDEOS, PLAYLISTS, CHANNELS, DISCUSSION, and ABOUT. A search icon is also present. The "VIDEOS" tab is selected, showing a list of uploaded videos. The first video in the list is titled "Introduction to Data Science With RAPIDS on NVIDIA GPUS" and has a thumbnail showing a purple geometric background with the text "Introduction to Data Science With RAPIDS on NVIDIA GPUS". It has 2K views and was posted 1 year ago. Below this, there are three more video thumbnails, each with a cyan dashed box around it, indicating they are part of a specific group or series. The first video in this group is titled "GPU Accelerated Single Cell RNA with RAPIDS and Clara..." and has 344 views. The second is "Hyperparameter Optimization (HPO) with RAPIDS on AWS..." and has 321 views. The third is "Hyperparameter Optimization (HPO) with RAPIDS on..." and has 353 views. To the right of these videos, there's a sidebar with a "RAPIDS: GPU ACCELERATED DATA SCIENCE" section featuring a timeline diagram with various colored boxes representing different tools and services like PyTorch, TensorFlow, Dask, XGBoost, etc. At the bottom of the sidebar, there's a video titled "ISBM 2020 - GPU Accelerated Interactive..." with 193 views.

Community supported notebooks have tutorials, examples, and various E2E demos.  
RAPIDS Youtube channel has explanations, code walkthroughs and use cases.

<https://github.com/rapidsai-community/notebooks-contrib>

# RAPIDS

## How Do I Get the Software?



GITHUB

<https://github.com/rapidsai>



ANACONDA

<https://anaconda.org/rapidsai/>



NGC

<https://ngc.nvidia.com/registry/nvidia-rapidsai-rapidsai>



DOCKER

<https://hub.docker.com/r/rapidsai/rapidsai/>

# Deploy RAPIDS Everywhere

Focused on Robust Functionality, Deployment, and User Experience



Amazon SageMaker



Azure



Azure Machine  
Learning



Google Cloud



Cloud  
Dataproc



SaturnCloud



Kubeflow



Integration with major cloud providers | Both containers and cloud specific machine instances  
Support for Enterprise and HPC Orchestration Layers

# Join the Movement

Everyone Can Help!



APACHE ARROW

<https://arrow.apache.org/>

@ApacheArrow

RAPIDS

RAPIDS

<https://rapids.ai>

@RAPIDSai



DASK

<https://dask.org>

@Dask\_dev



GPU OPEN ANALYTICS  
INITIATIVE

<http://gpuopenanalytics.com/>

@GPUOAI

Integrations, feedback, documentation support, pull requests, new issues, or code donations welcomed!

# THANK YOU

 @RAPIDSai

# RAPIDS