

# RAPIDS

## The Platform Inside and Out Release 0.15

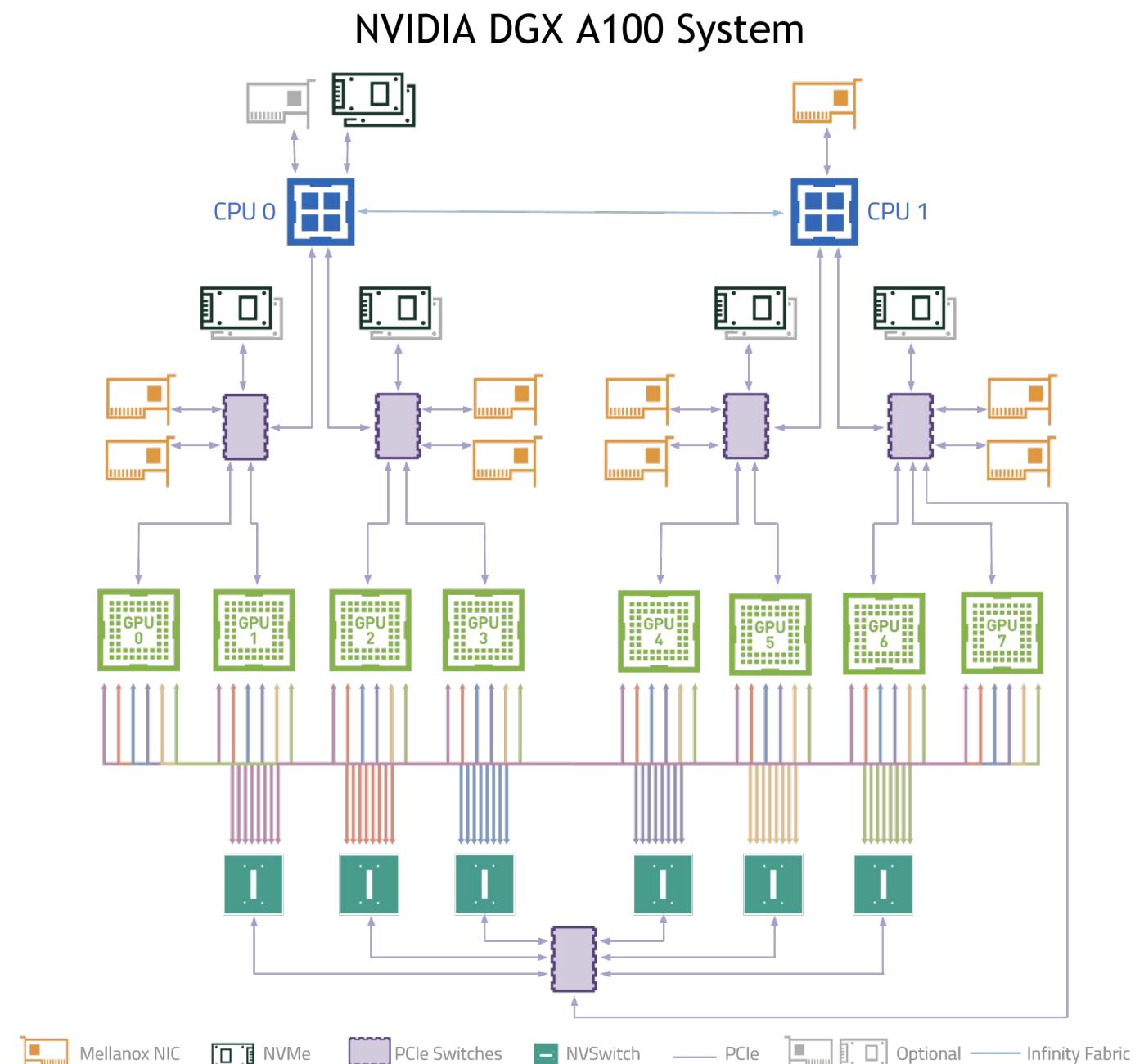
Joshua Patterson - Senior Director, RAPIDS Engineering

# Why GPUs?

## Numerous hardware advantages

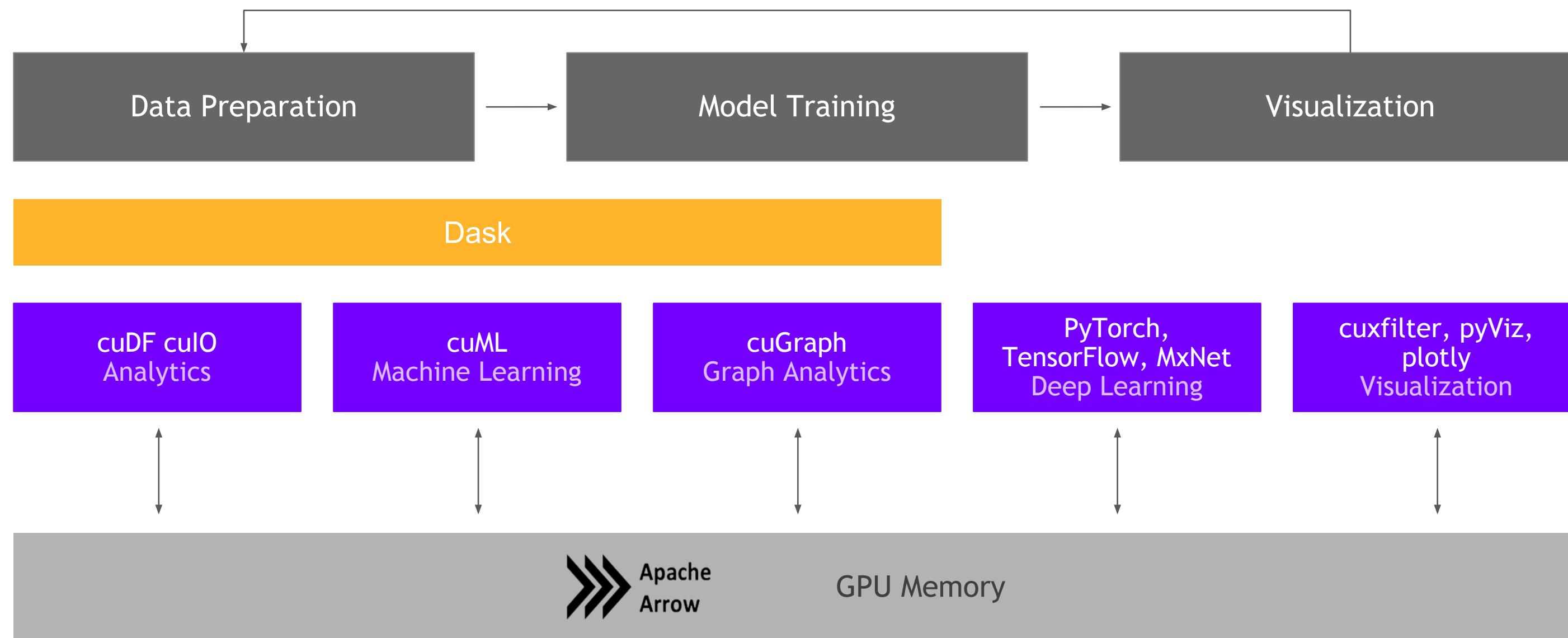
- ▶ Thousands of cores with up to ~20 TeraFlops of general purpose compute performance
- ▶ Up to 1.5 TB/s of memory bandwidth
- ▶ Hardware interconnects for up to 600 GB/s bidirectional GPU <---> GPU bandwidth
- ▶ Can scale up to 16x GPUs in a single node

**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

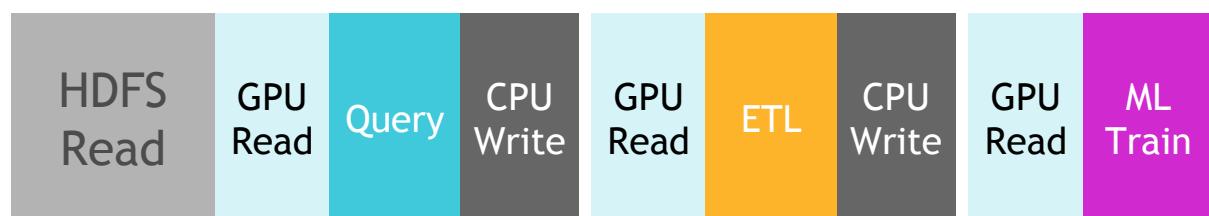
Hadoop Processing, Reading from Disk



Spark In-Memory Processing



Traditional GPU Processing

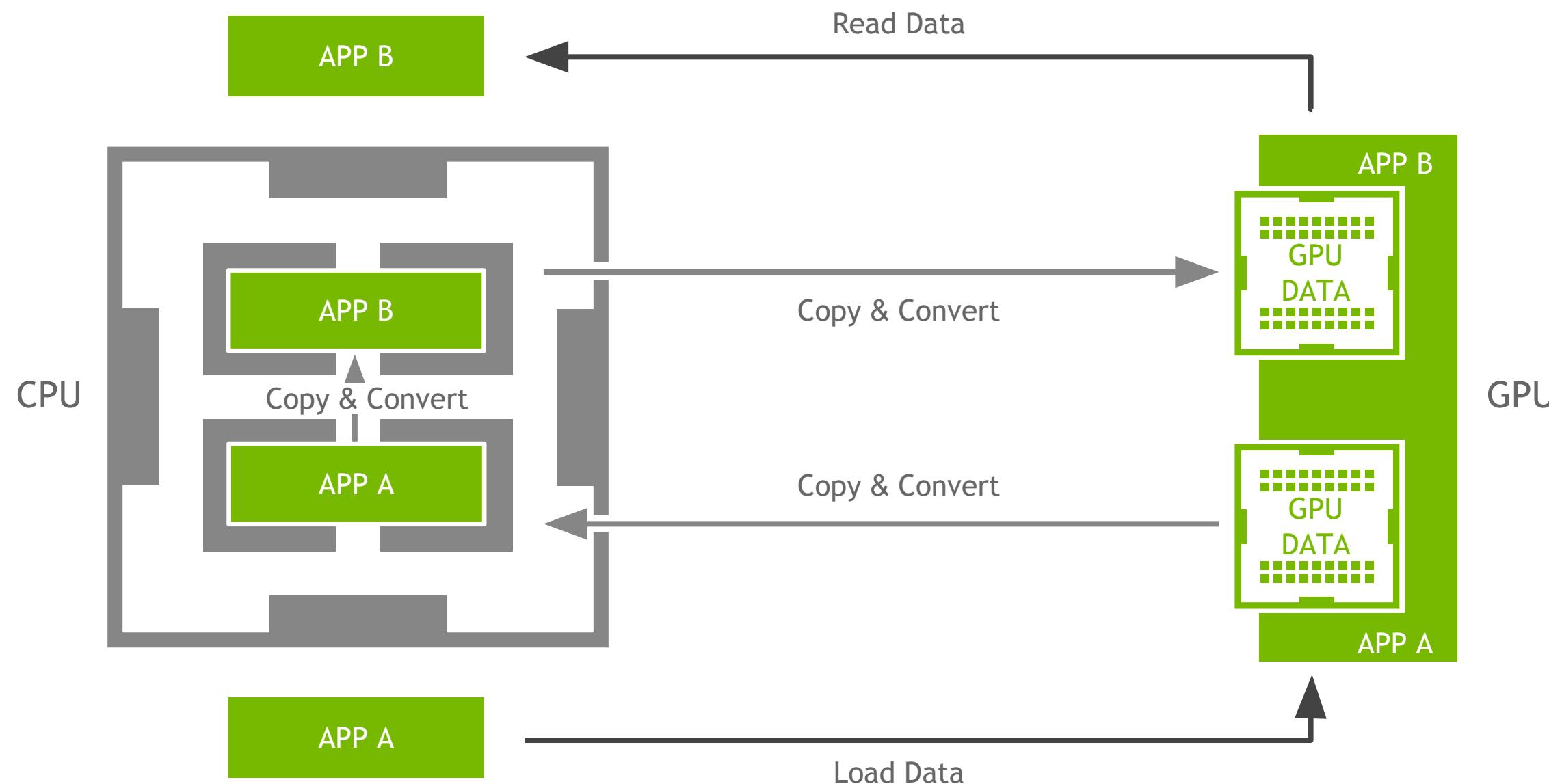


25-100x Improvement  
Less Code  
Language Flexible  
Primarily In-Memory

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

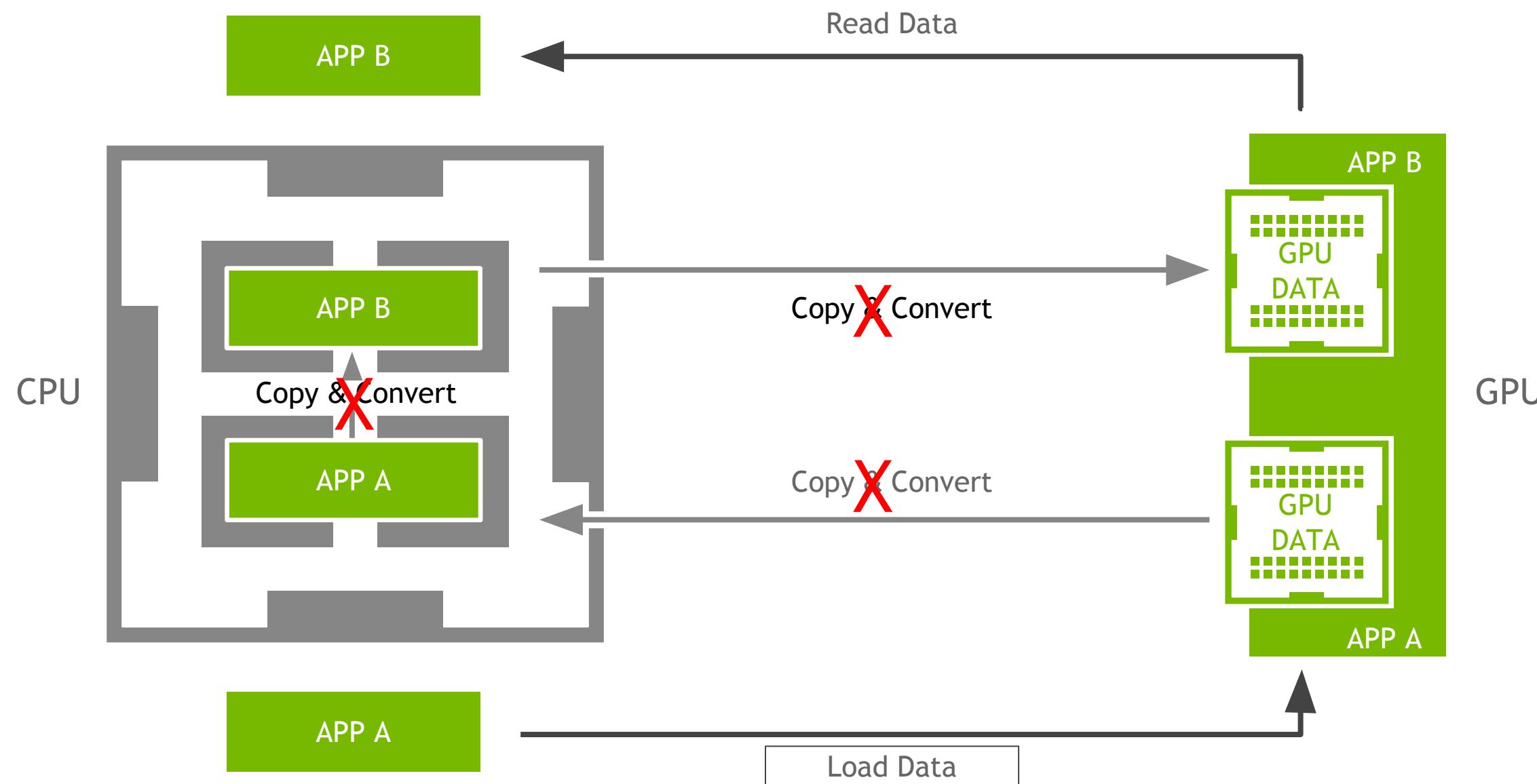
# Data Movement and Transformation

The Bane of Productivity and Performance

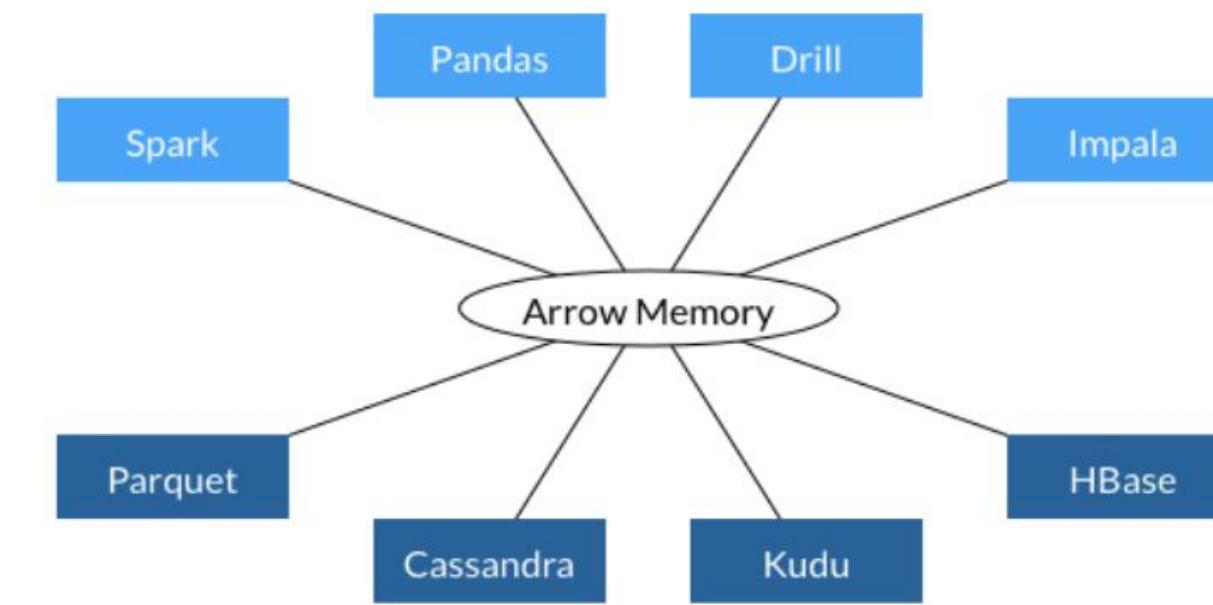
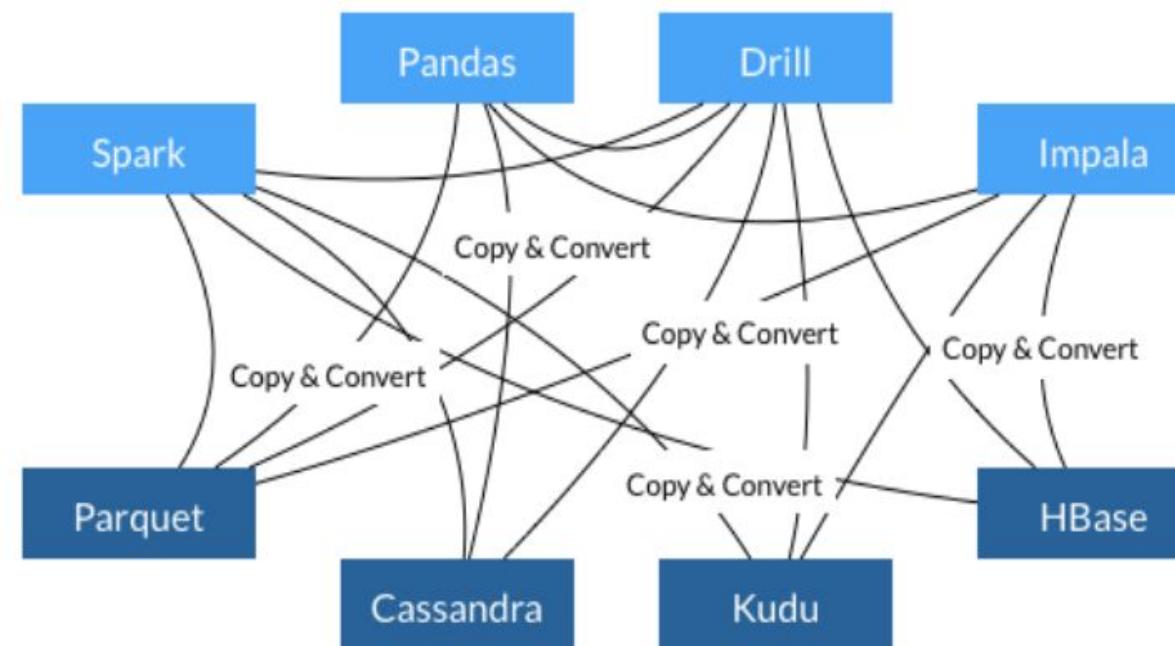


# Data Movement and Transformation

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)

▶ Similar functionality implemented in multiple projects

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

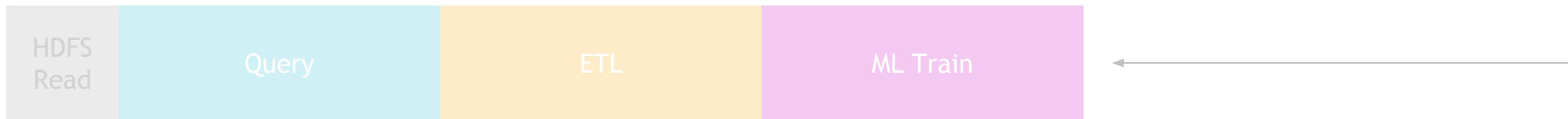
# Data Processing Evolution

## Faster Data Access, Less Data Movement

Hadoop Processing, Reading from Disk



Spark In-Memory Processing



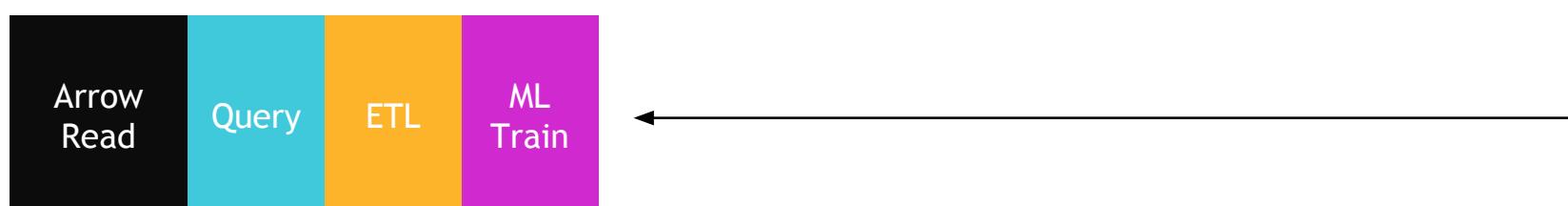
25-100x Improvement  
Less Code  
Language Flexible  
Primarily In-Memory

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

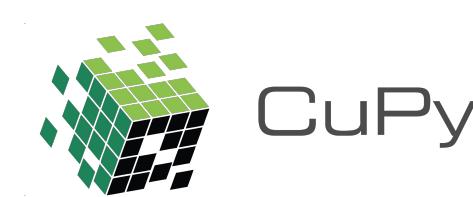
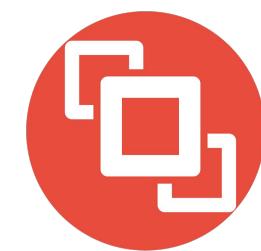
Up to 350x faster queries; Hours to Seconds!

TPCx-BB 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. It can be run at multiple “Scale Factors”.

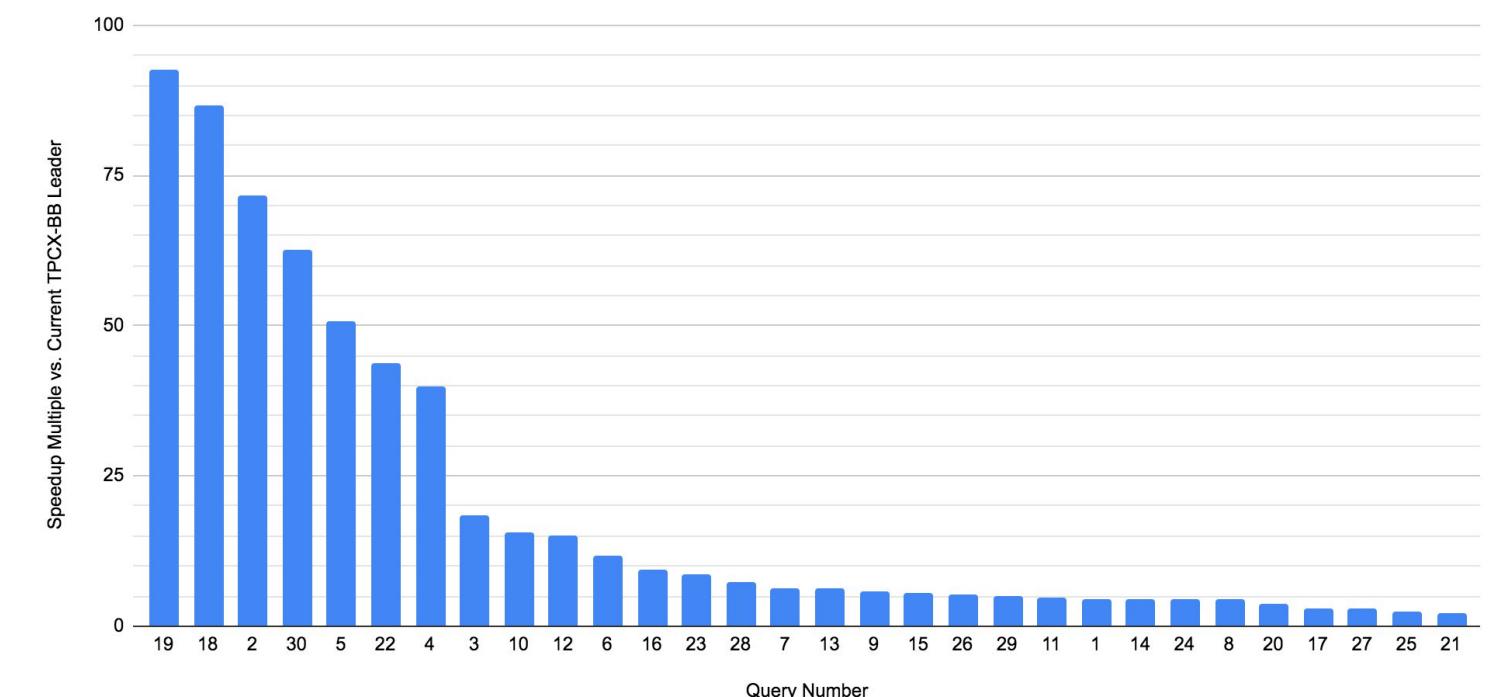
- ▶ SF1 - 1GB
- ▶ SF1K - 1 TB
- ▶ SF10K - 10 TB

RAPIDS results at SF1K (2 DGX A100s) and SF10K (16 DGX A100s) show GPUs provide dramatic cost and time-savings for small scale *and* large-scale data analytics problems

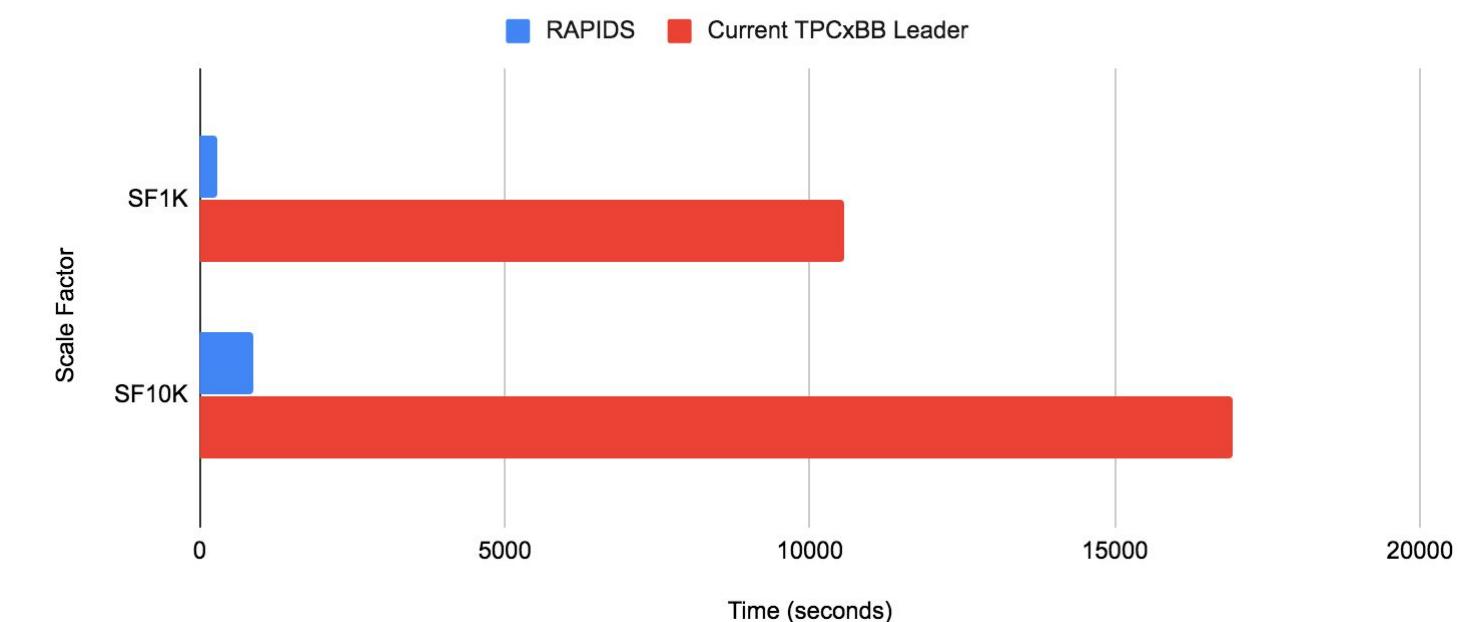
- ▶ SF1K 37.1x average speed-up
- ▶ SF10K 19.5x average speed-up (7x Normalized for Cost)



TPCx-BB SF10K Results - RAPIDS Running on 16 NVIDIA DGX A100s



TPCx-BB Total Time RAPIDS vs. Current Leaders



# Speed, UX, and Iteration

## The Way to Win at Data Science

**François Chollet** @fchollet · Apr 3  
Winners are those who went through \*more iterations\* of the "loop of progress" -- going from an idea, to its implementation, to actionable results. So the winning teams are simply those able to run through this loop \*faster\*. And this is where Keras gives you an edge.

The diagram illustrates the iterative process of data science. It shows a central figure running, with arrows pointing from 'Idea' (lightbulb icon) to 'Experiment' (lab flask icon), then to 'Results' (bar chart icon), and finally back to 'Idea'. Labels include 'Visualization & understanding' and 'Software tools' along the cycle, and 'Infrastructure' at the bottom.

12:31 PM - 3 Apr 2019  
50 Retweets 158 Likes

**François Chollet** @fchollet · Apr 3  
We often talk about how following UX best practices for API design makes Keras more accessible and easier to use, and how this helps beginners. But those who stand to benefit most from good UX \*aren't\* the beginners. It's actually the very best practitioners in the world.

**François Chollet** @fchollet · Apr 3  
Because good UX reduces the overhead (development overhead & cognitive overhead) to setting up new experiments. It means you will be able to iterate faster. You will be able to try more ideas.

**François Chollet** @fchollet · Apr 3  
And ultimately, that's how you win competitions or get papers published.

**François Chollet** @fchollet · Apr 3  
So I don't think it's mere personal preference if Kaggle champions are overwhelmingly using Keras.

**Joshua Patterson** @datametrician · Apr 3  
Replying to @fchollet  
This is the fundamental belief that drives @RAPIDSai. @nvidia #GPU infrastructure is fast, people need to iterate quickly, people want a known #python interface. Combine them and you're off to the races!

**François Chollet** @fchollet · Apr 3  
The second question asked about secondary frameworks -- usually teams win with an ensemble that involves many different ML frameworks. Here are \*all\* frameworks used.

Sklearn tops that ranking: everyone uses sklearn (although often as an auxiliary, for preprocessing or scoring).

A horizontal bar chart titled 'All (primary + auxiliary) ML software tools used by top-5 Kaggle teams in each competition (n=120)'. The x-axis represents the count of teams, ranging from 0 to 80. The y-axis lists frameworks: Sci-kit Learn, Keras, LightGBM, XGBoost, PyTorch, TensorFlow (non-Keras), Caffe, MXNet, Fastai, Caffe2, CatBoost, and R Random Forest. A legend indicates that orange bars represent 'Deep' frameworks and blue bars represent 'Classic' frameworks. The chart shows that Sci-kit Learn and Keras are the most widely used frameworks.

Framework	Count (approx.)	Type
Sci-kit Learn	78	Classic
Keras	65	Deep
LightGBM	55	Classic
XGBoost	55	Classic
PyTorch	30	Deep
TensorFlow (non-Keras)	25	Deep
Caffe	5	Deep
MXNet	5	Deep
Fastai	5	Deep
Caffe2	5	Deep
CatBoost	5	Classic
R Random Forest	5	Classic

kaggle

# RAPIDS 0.15 Release Summary

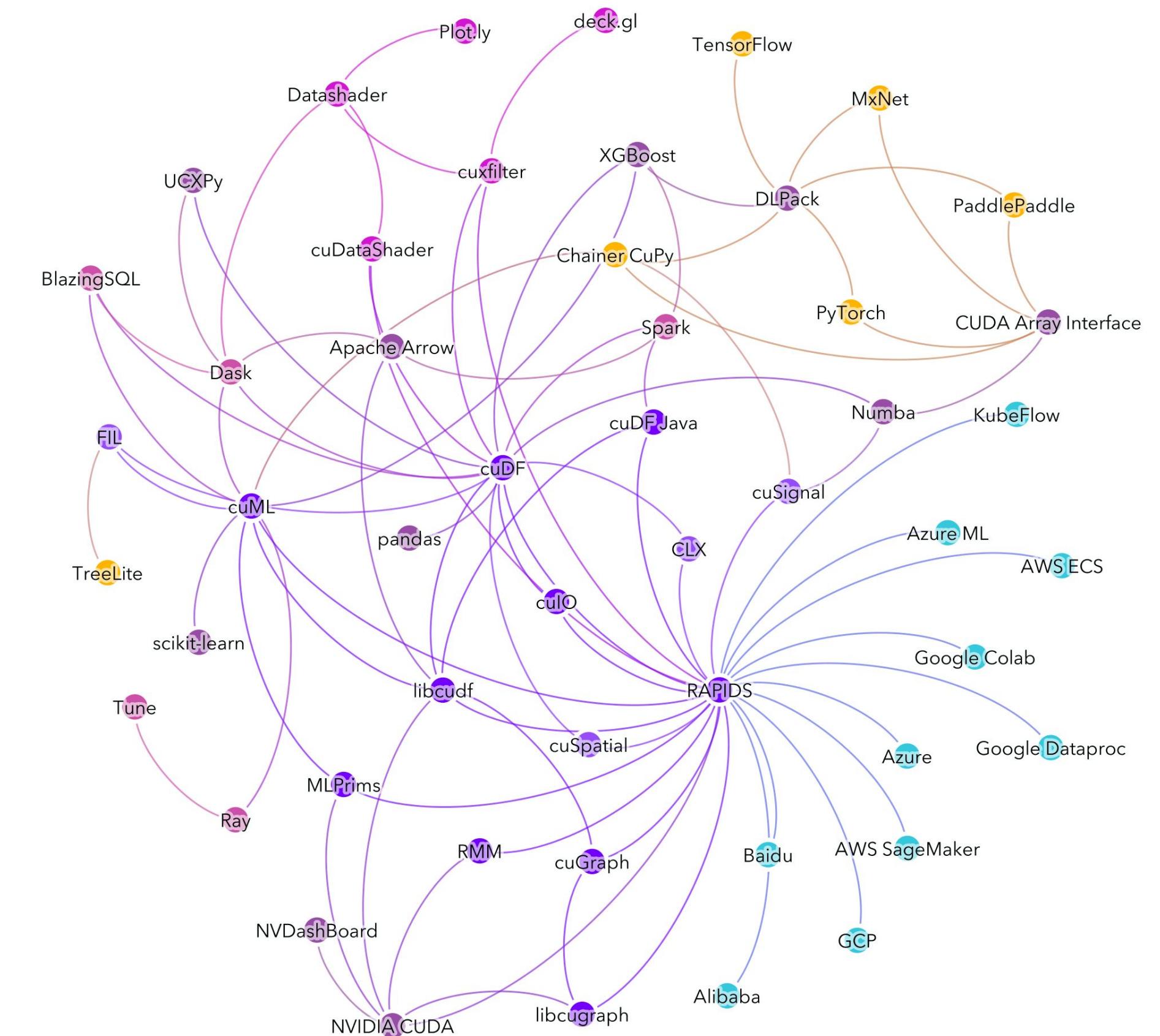
# What's New in Release 0.15?

- ▶ CUDA 11 and Ampere support added across all libraries.
- ▶ cuDF adds unsigned integer and timedelta (duration) dtypes support as well as experimental support for a List dtype, Kafka as a supported data source, Groupby.rolling() support, and additional NLP preprocessing functions.
- ▶ cuML machine learning library adds scikit-learn compatible NLP preprocessors, initial AutoARIMA support, MNMG kNN classification and regression wrappers, accelerated multi-class forest inference, SVM sample weights and probability prediction, and an experimental Incremental PCA model for out-of-core fitting.
- ▶ cuGraph initial version of the Leiden algorithm, added Edge Betweenness Centrality, improved Louvain and the ECG analytics, and added the HITS algorithm.
- ▶ UCX-Py adds reusable endpoints, Cython optimizations, code clean up, and bug fixes.
- ▶ BlazingSQL now supports out-of-core query execution, which enables queries to operate on datasets dramatically larger than available GPU memory.
- ▶ cuSignal adds multi-point kalman filter, improved embedded GPU support, and performance optimizations.
- ▶ cuSpatial adds point-to-nearest polyline queries and accelerated point-in-polygon via quadtree.
- ▶ cuxfilter will have a delayed release, but now has large graph support. See [Notice RGN4](#) in our docs for details.
- ▶ RMM deprecates the CNMeM, adds per-thread default stream support, adds `binning\_memory\_resource.

# RAPIDS Everywhere

## The Next Phase of RAPIDS

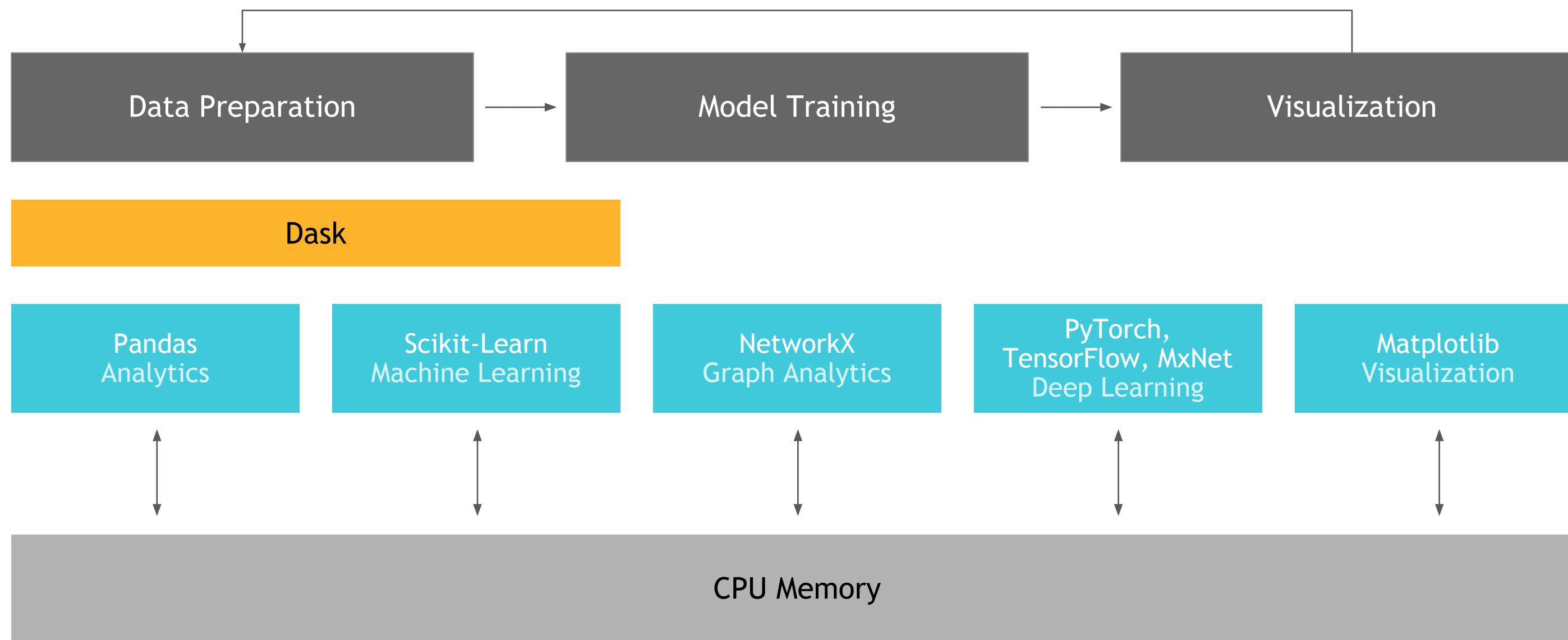
Exactly as it sounds—our goal is to make RAPIDS as usable and performant as possible wherever data science is done. We will continue to work with more open source projects to further democratize acceleration and efficiency in data science.



# RAPIDS Core

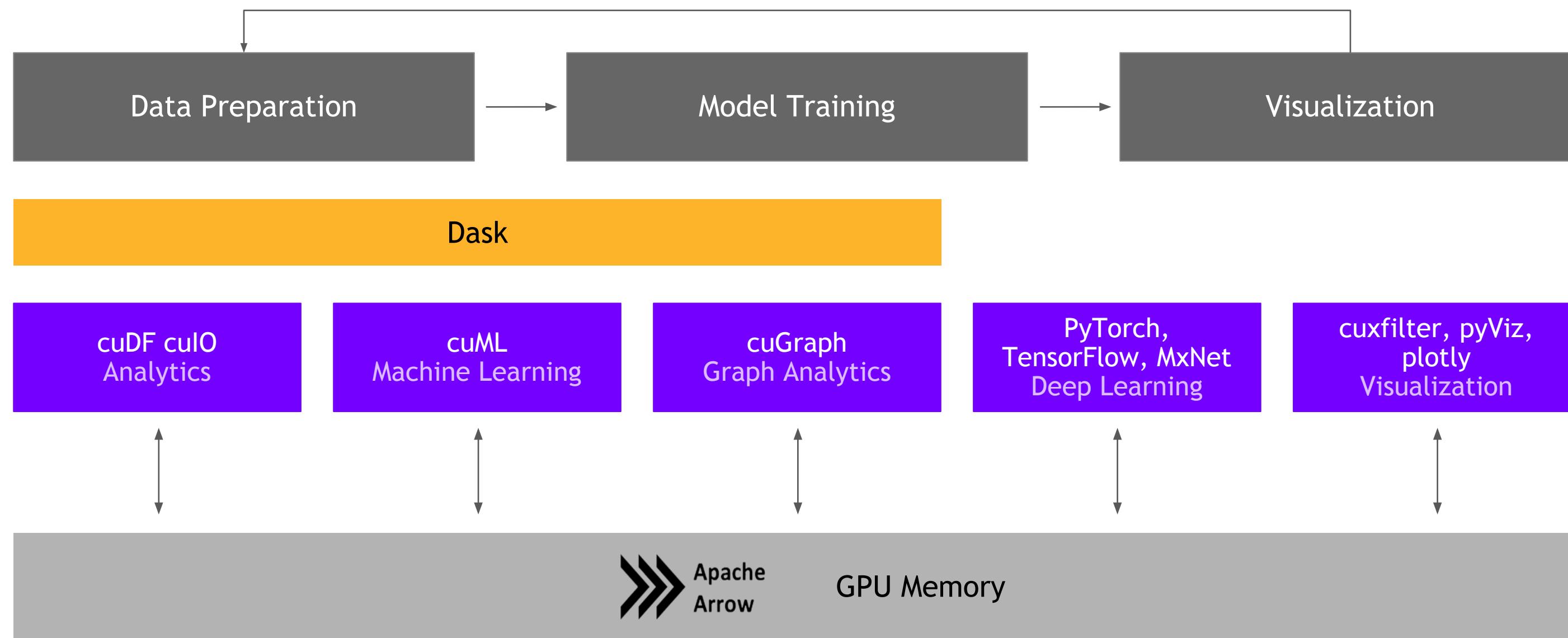
# Open Source Data Science Ecosystem

Familiar Python APIs



# RAPIDS

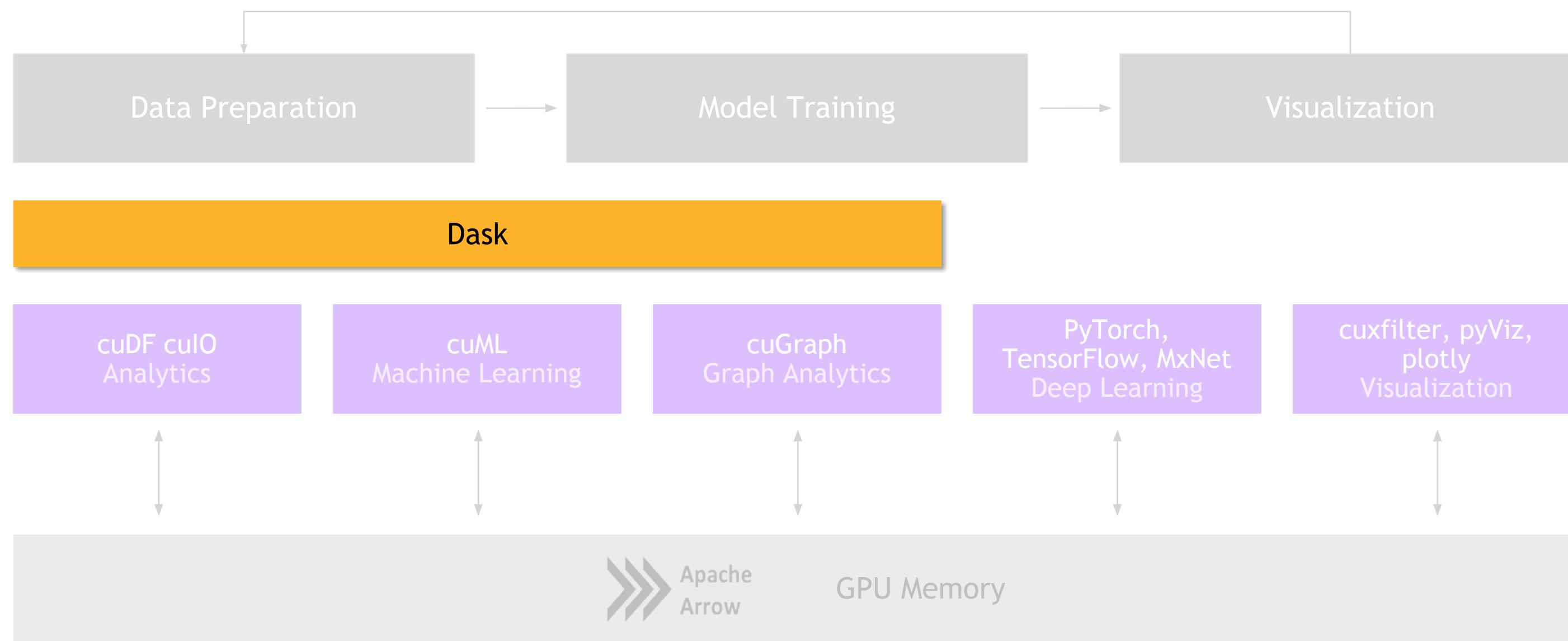
## End-to-End Accelerated GPU Data Science



# Dask

# RAPIDS

## Scaling RAPIDS with Dask



# Why Dask?

## DEPLOYABLE

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

## PYDATA NATIVE

- Easy Migration: Built on top of NumPy, Pandas Scikit-Learn, etc
- Easy Training: With the same APIs
- Trusted: With the same developer community

## EASY SCALABILITY

- Easy to install and use on a laptop
- Scales out to thousand node clusters

## POPULAR

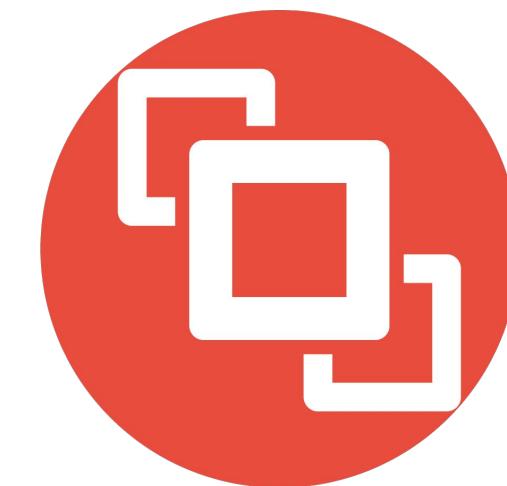
- Most Common parallelism framework today in the PyData and SciPy community



# Why OpenUCX?

## Bringing Hardware Accelerated Communications to Dask

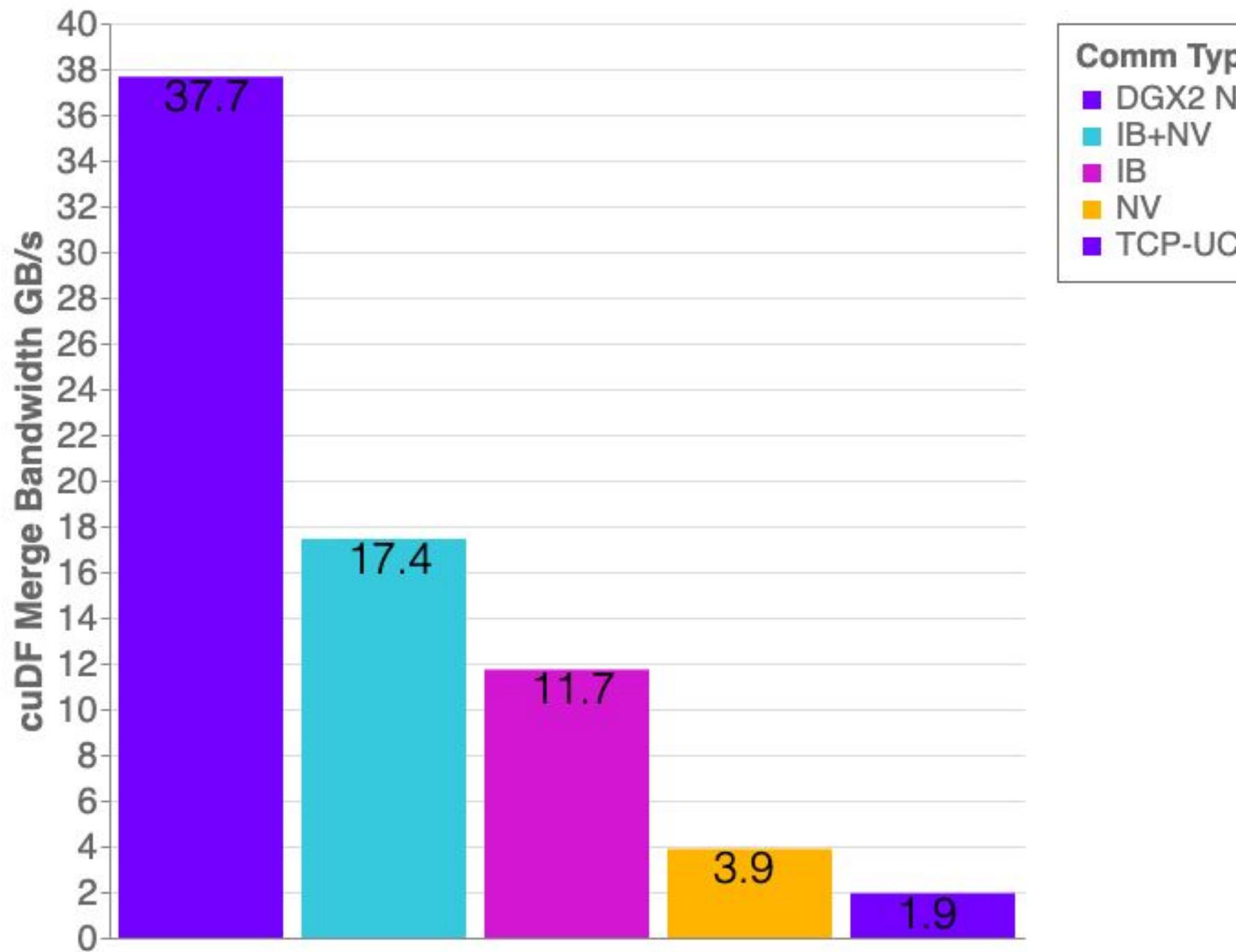
- ▶ TCP sockets are slow!
- ▶ UCX provides uniform access to transports (TCP, InfiniBand, shared memory, NVLink)
- ▶ Alpha Python bindings for UCX (ucx-py)
- ▶ Will provide best communication performance, to Dask based on available hardware on nodes/cluster



---

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

# Benchmarks: Distributed cuDF Random Merge



cuDF v0.14, UCX-PY 0.14

- ▶ Running on NVIDIA DGX-2:

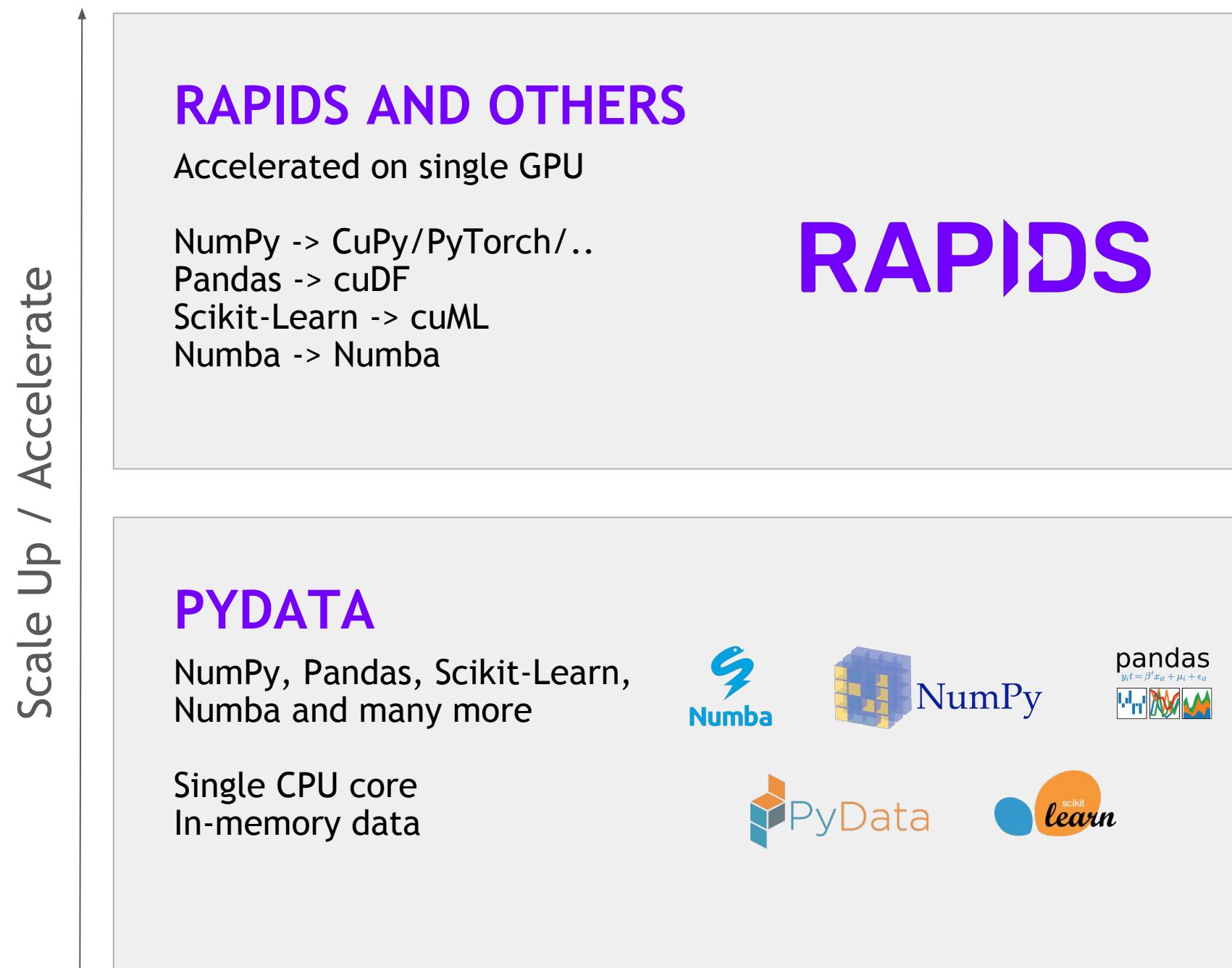
- ▶ GPU: NVIDIA Tesla V100 32GB

- ▶ CPU: Intel(R) Xeon(R) CPU 8168 @ 2.70GHz

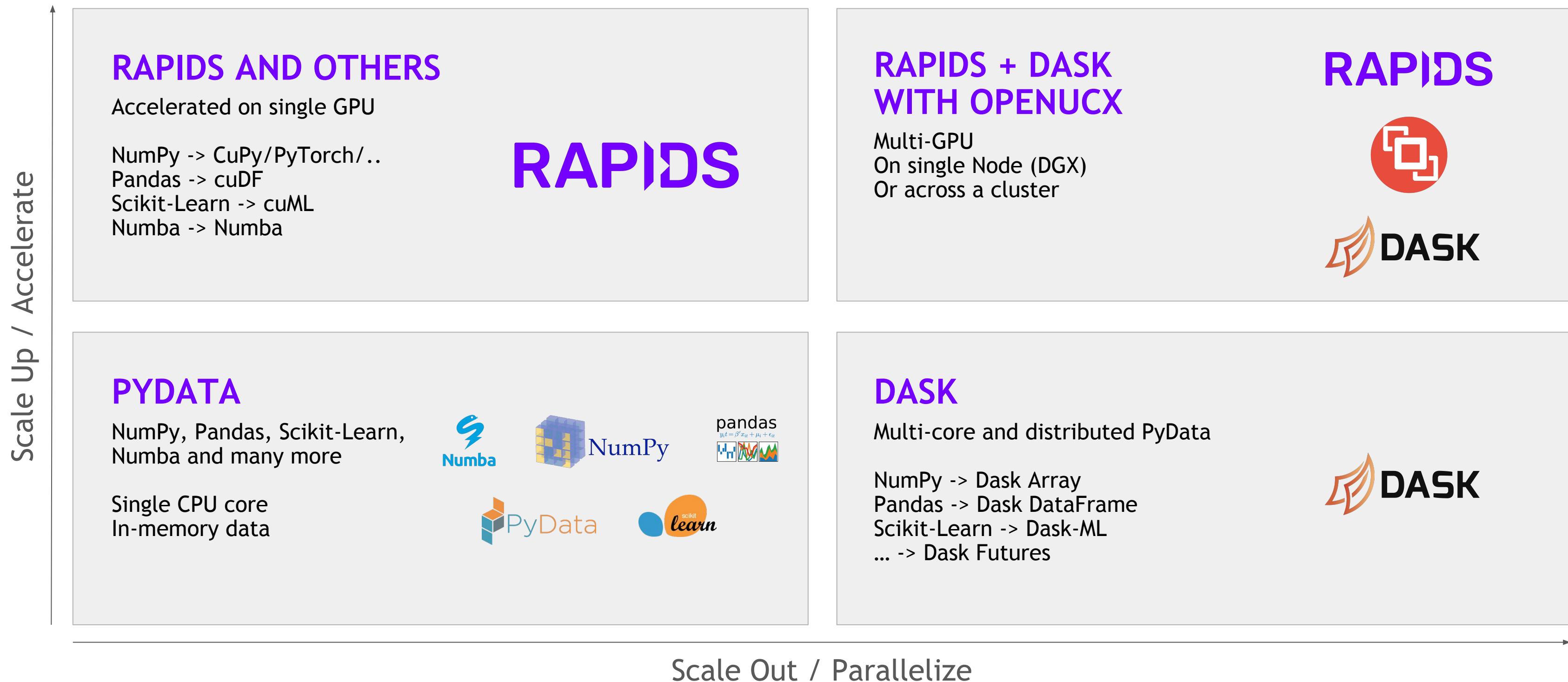
## □ Benchmark Setup:

- ▶ DataFrames: Left/Right 1x int64 column key column, 1x int64 value columns
- ▶ Merge: Inner
- ▶ 30% of matching data balanced across each partition

# Scale Up with RAPIDS



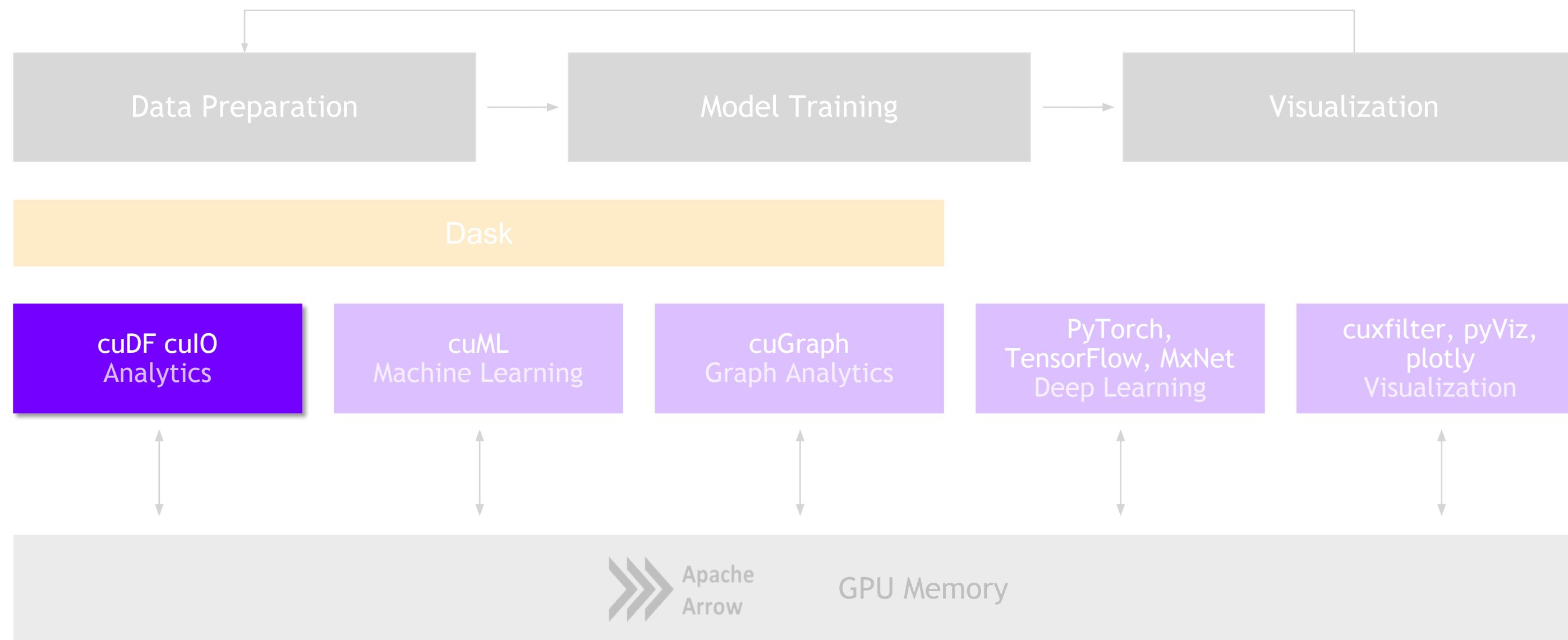
# Scale Out with RAPIDS + Dask with OpenUCX



cuDF

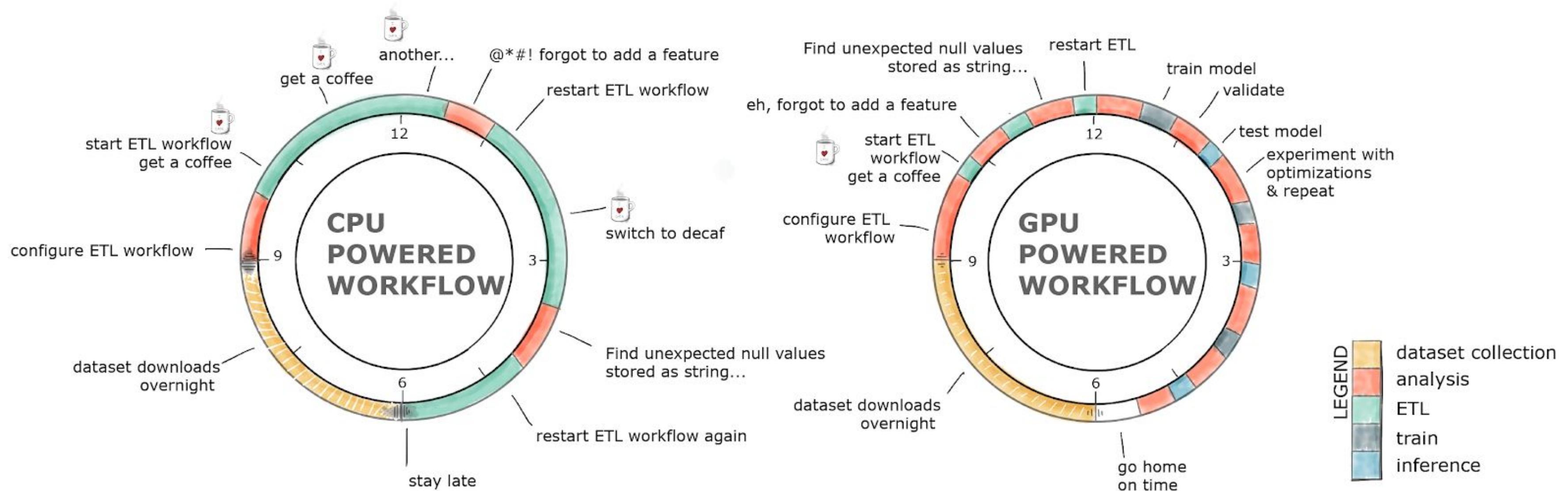
# RAPIDS

## GPU Accelerated Data Wrangling and Feature Engineering

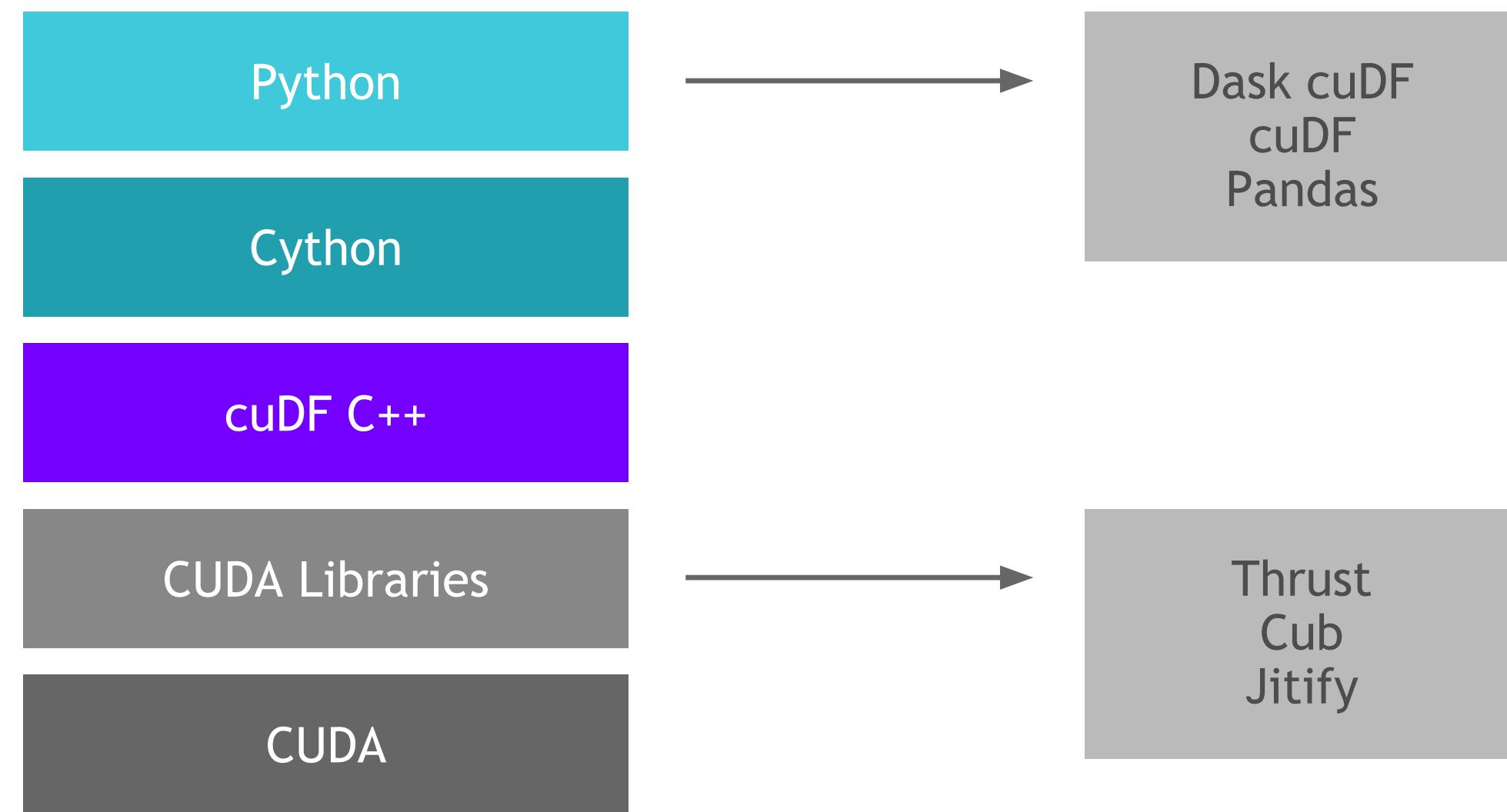


# GPU-Accelerated ETL

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

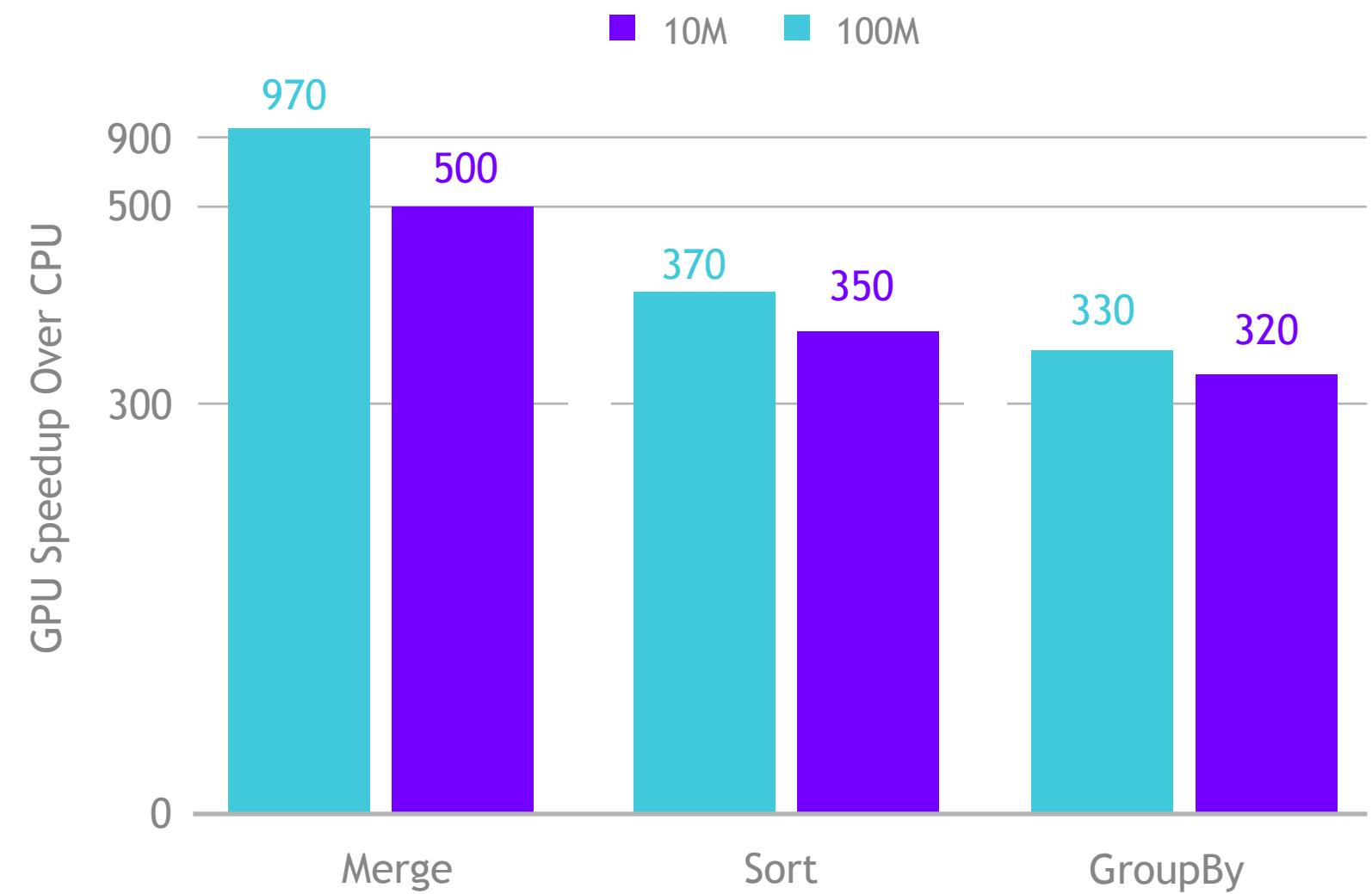
  

```
In [6]: #grabbing the first character of the years in city string to get rid of plus sign, and converting  
#to int  
gdf['city_years'] = gdf.Stay_In_Current_City_Years.str.get(0).stoi()  
  
In [7]: #Here we can see how we can control what the value of our dummies with the replace method and turn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

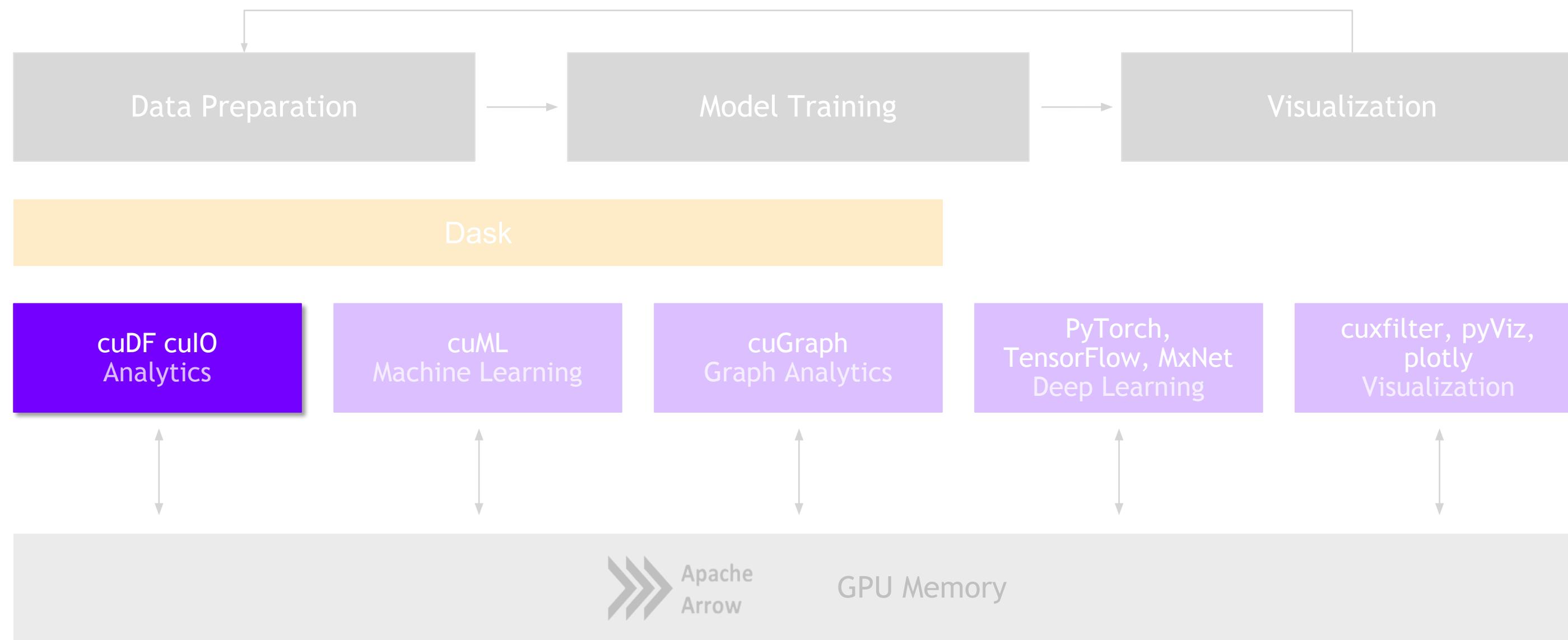
cuDF v0.13, Pandas 0.25.3

- ▶ 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

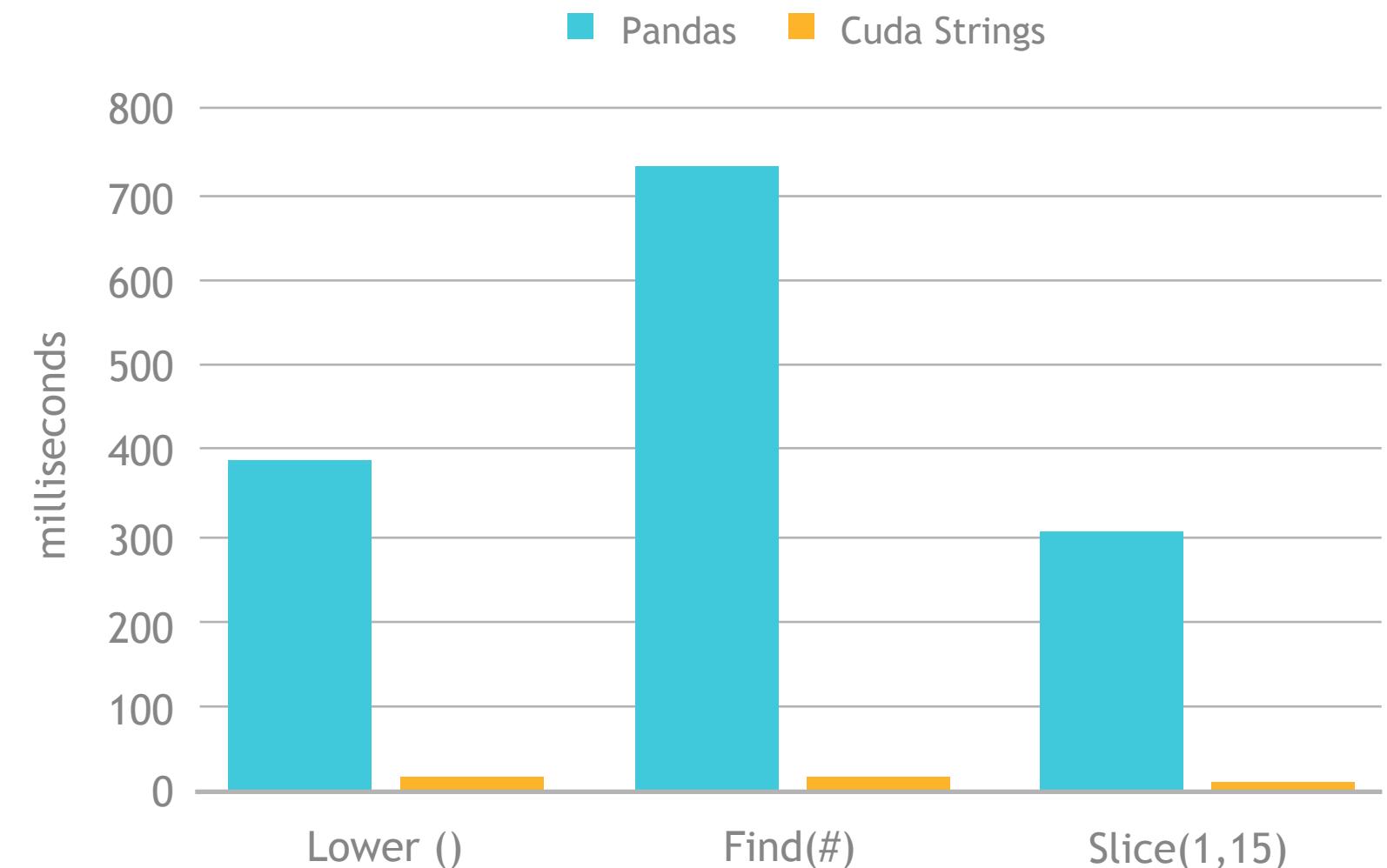
## String Support

### CURRENT V0.15 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 V0.16+ 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
- ▶ 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

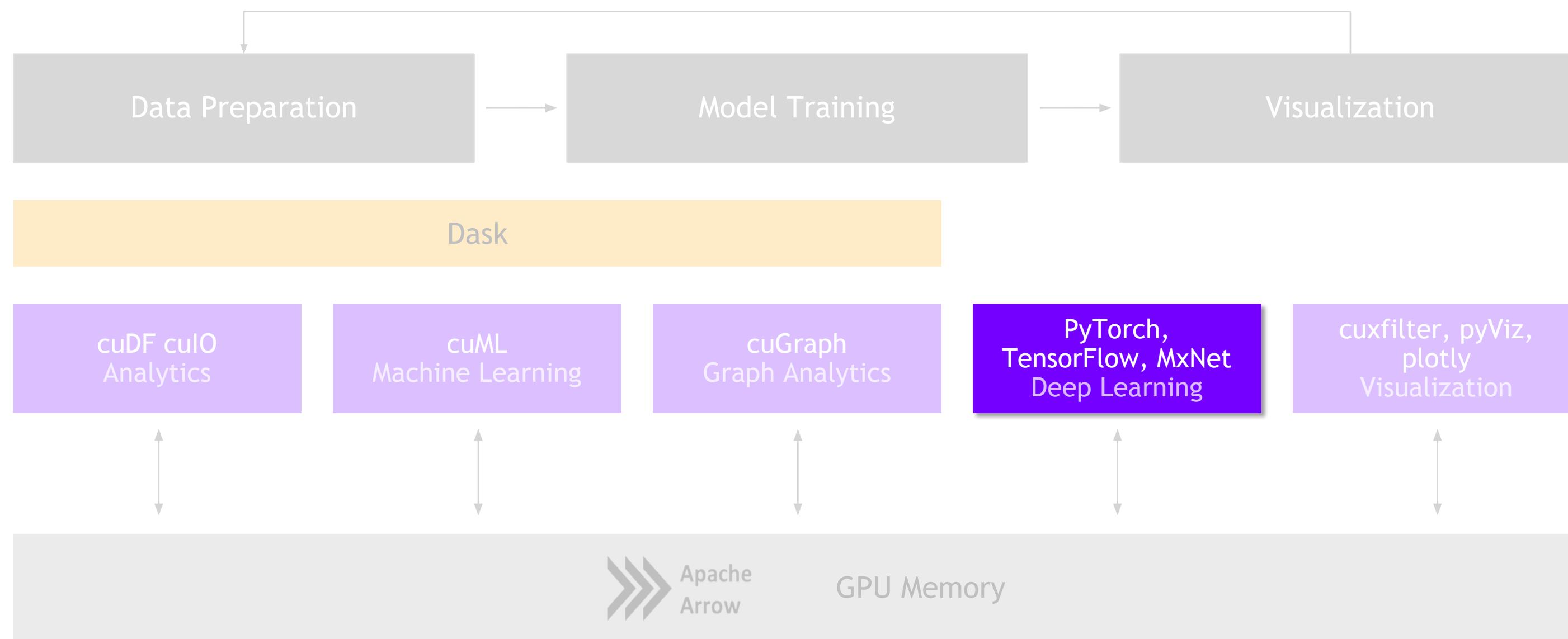
```
1]: import pandas, cudf
2]: %time len(pandas.read_csv('data/nyc/yellow_tripdata_2015-01.csv'))
CPU times: user 25.9 s, sys: 3.26 s, total: 29.2 s
Wall time: 29.2 s
2]: 12748986
3]: %time len(cudf.read_csv('data/nyc/yellow_tripdata_2015-01.csv'))
CPU times: user 1.59 s, sys: 372 ms, total: 1.96 s
Wall time: 2.12 s
3]: 12748986
4]: !du -hs data/nyc/yellow_tripdata_2015-01.csv
1.9G  data/nyc/yellow_tripdata_2015-01.csv
```

Source: Apache Crail blog: [SQL Performance: Part 1 - Input File Formats](#)

# ETL is Not Just DataFrames!

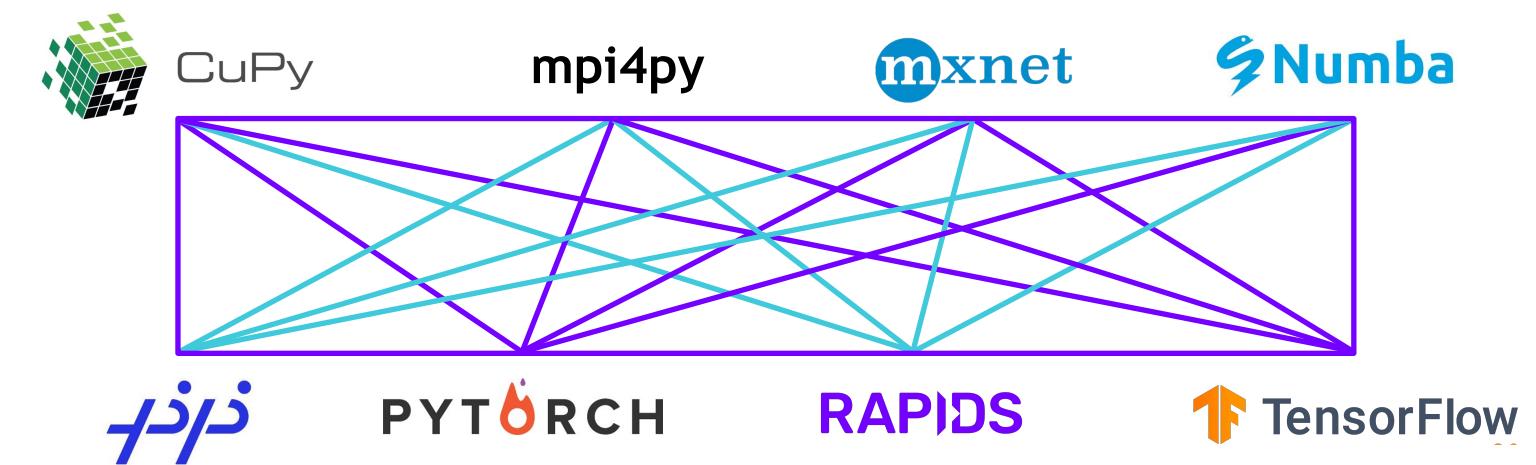
# RAPIDS

## 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



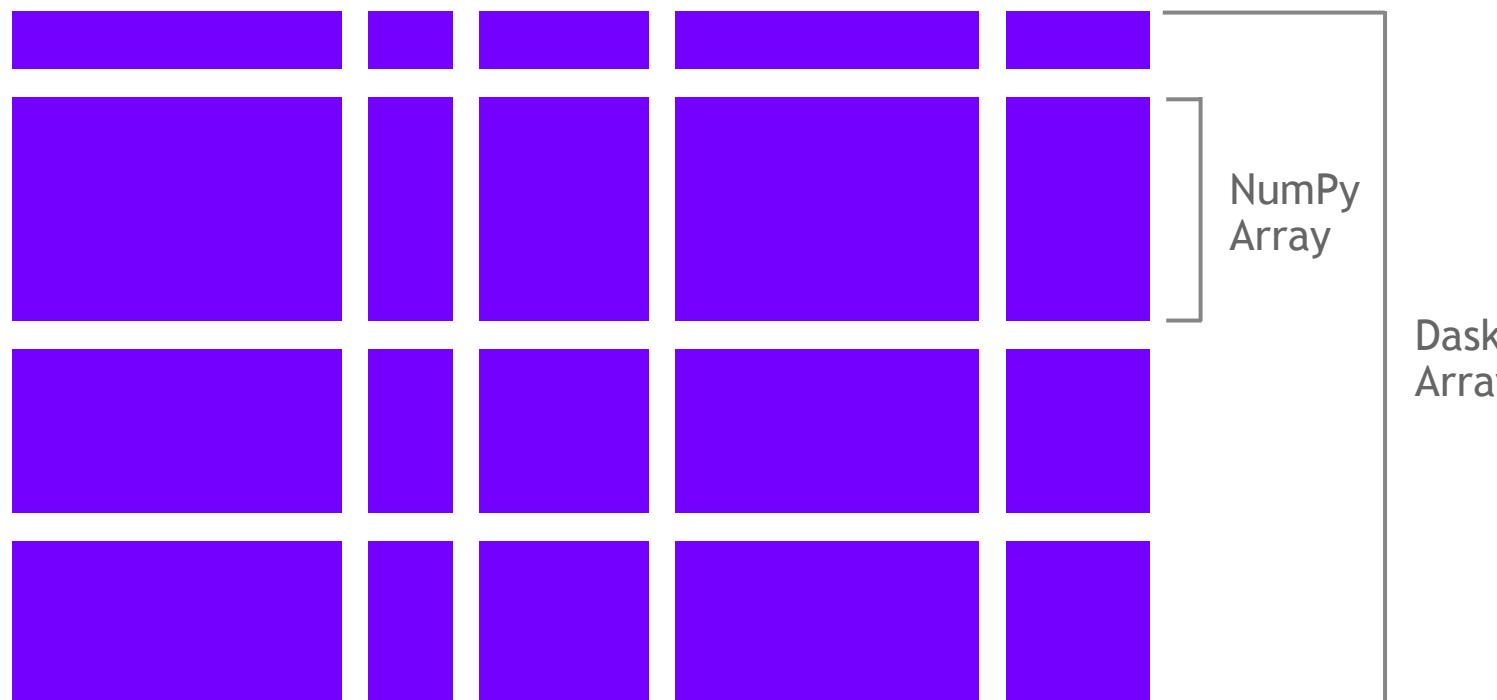
# ETL - Arrays and DataFrames

## Dask and CUDA Python Arrays



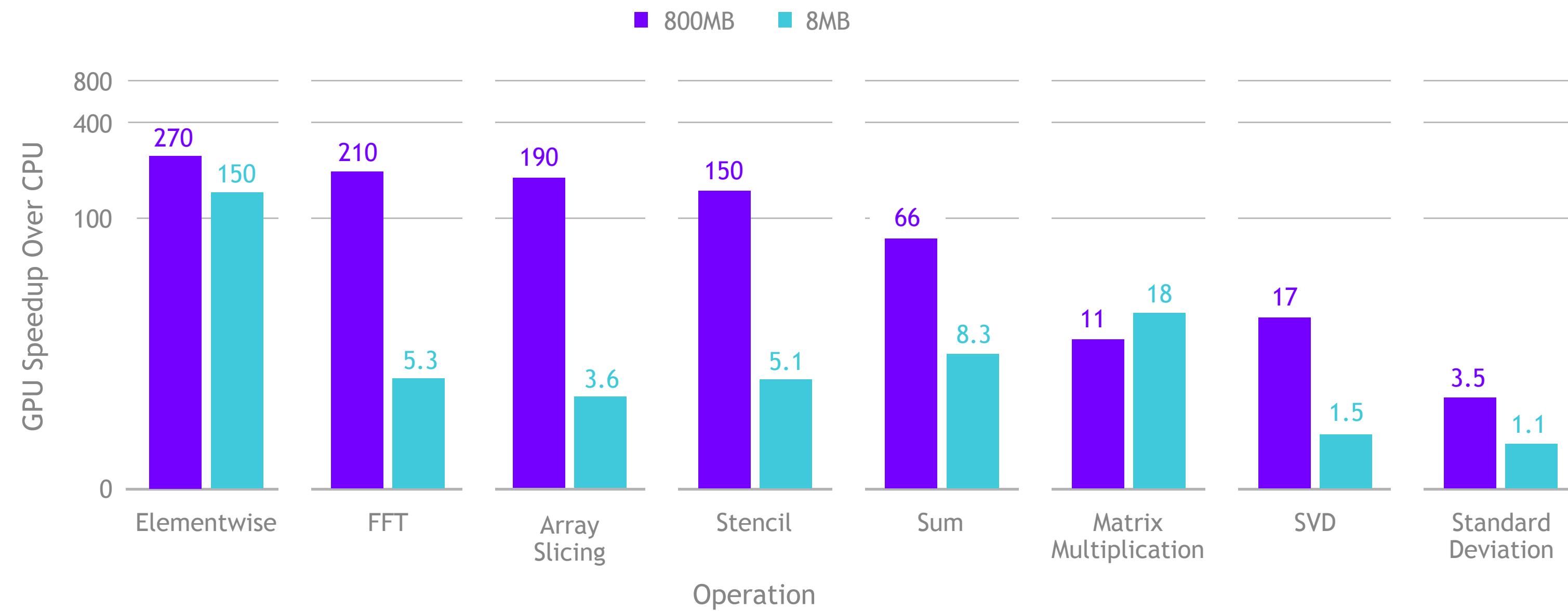
CuPy

 Numba



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

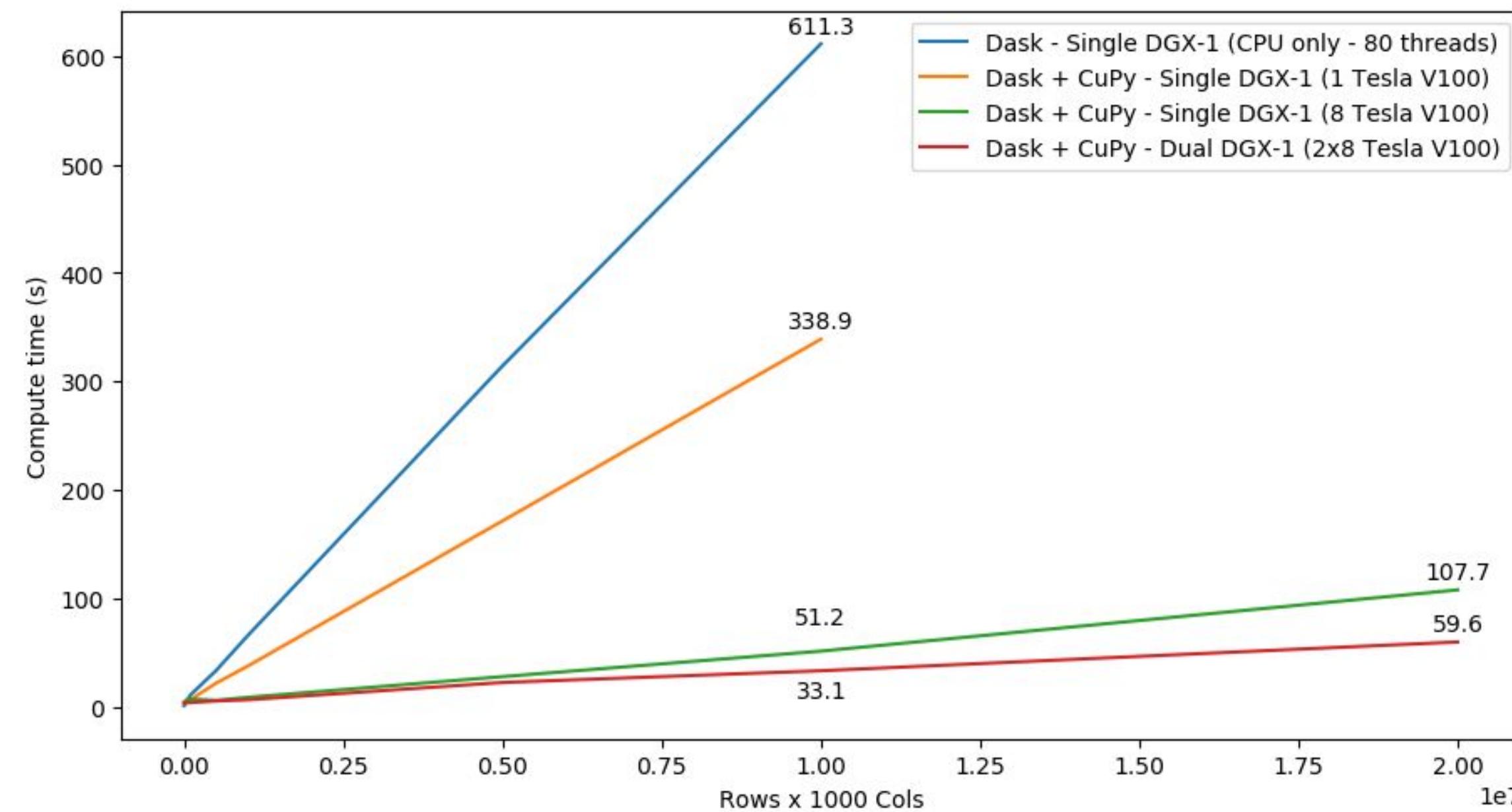
# Benchmark: Single-GPU CuPy vs NumPy



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

# SVD Benchmark

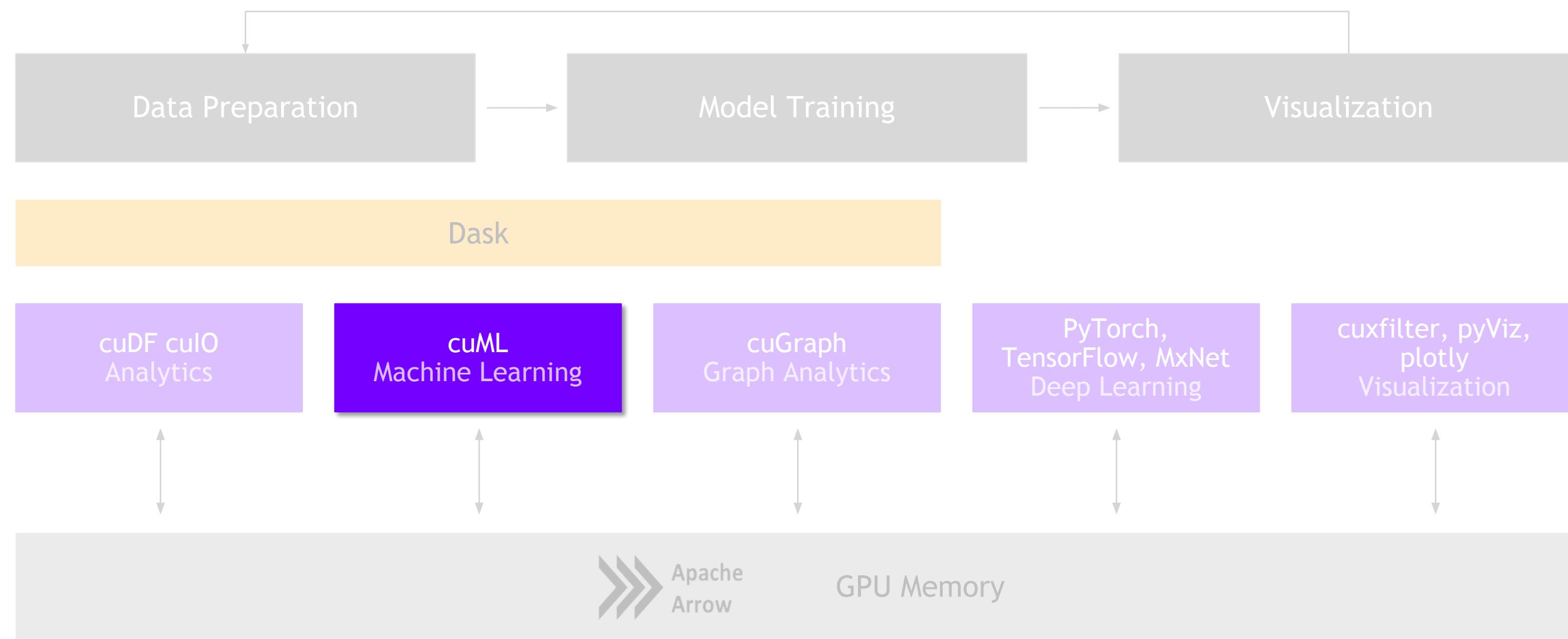
## Dask and CuPy Doing Complex Workflows



cuML

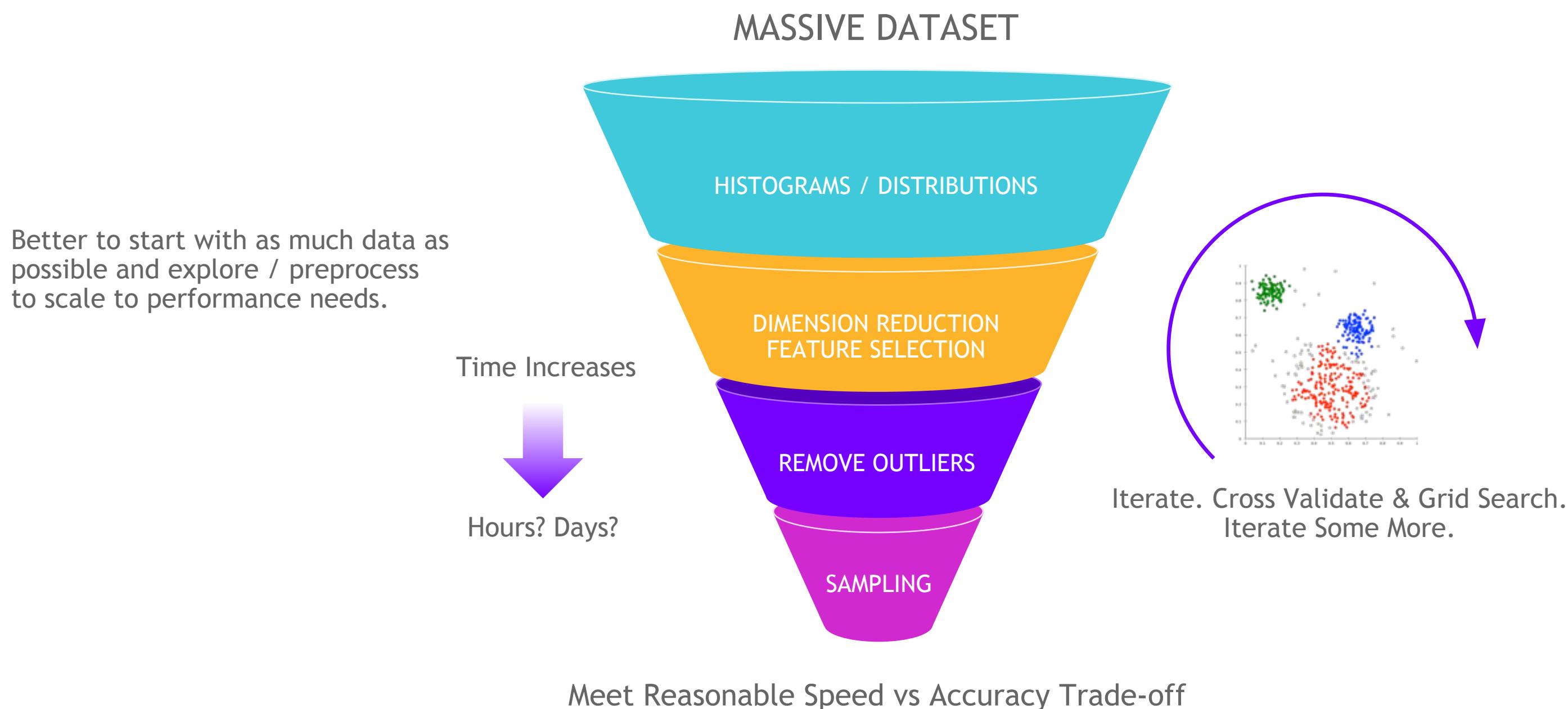
# Machine Learning

## More Models More Problems

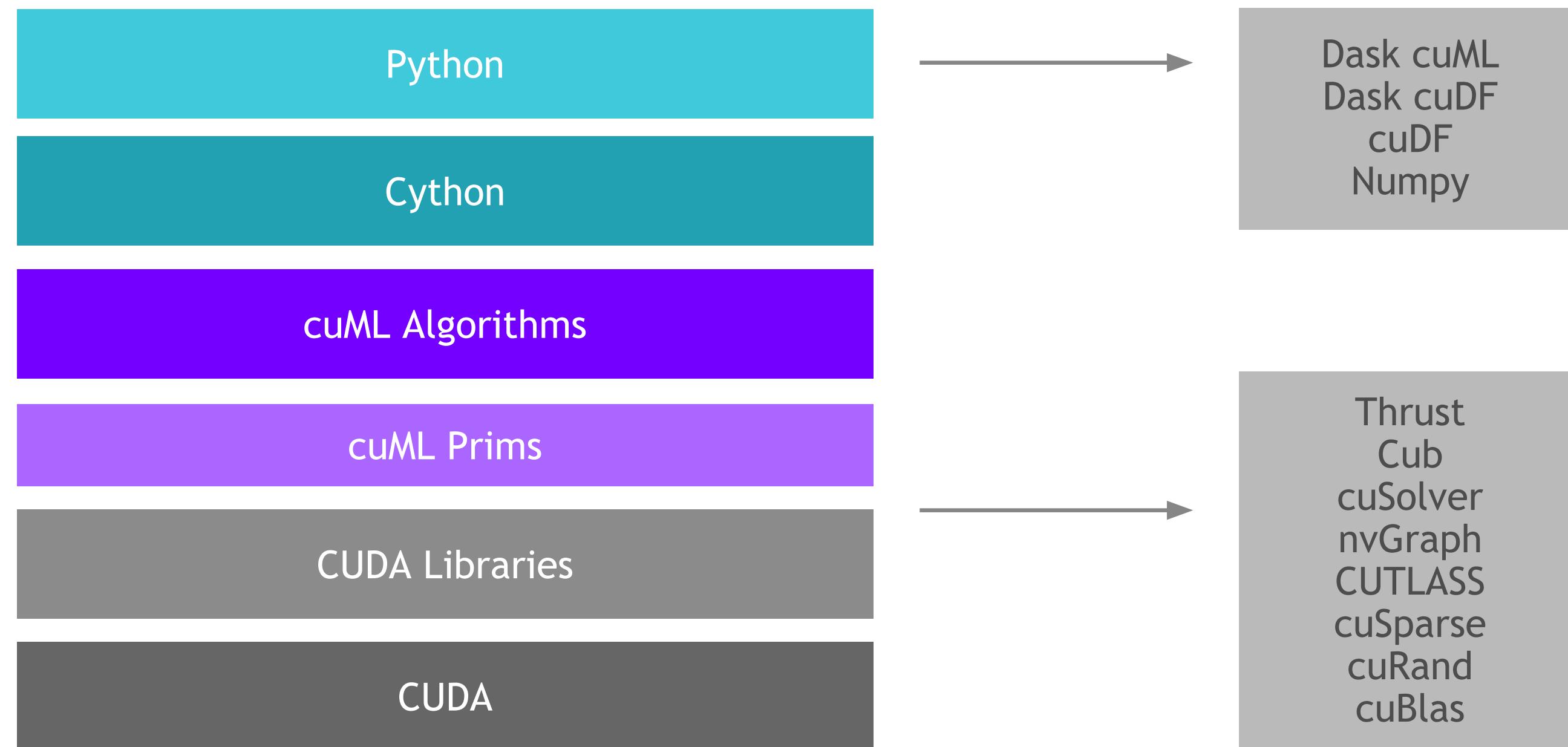


# Problem

## Data Sizes Continue to Grow



# ML Technology Stack



# RAPIDS Matches Common Python APIs

## CPU-based Clustering

```
from sklearn.datasets import make_moons
import pandas

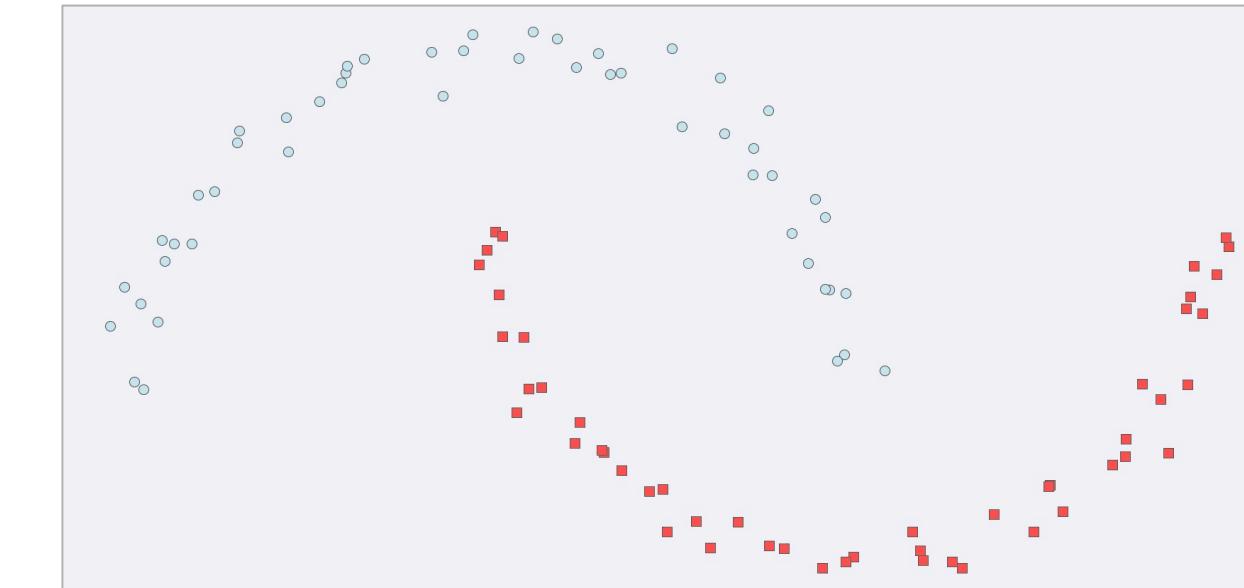
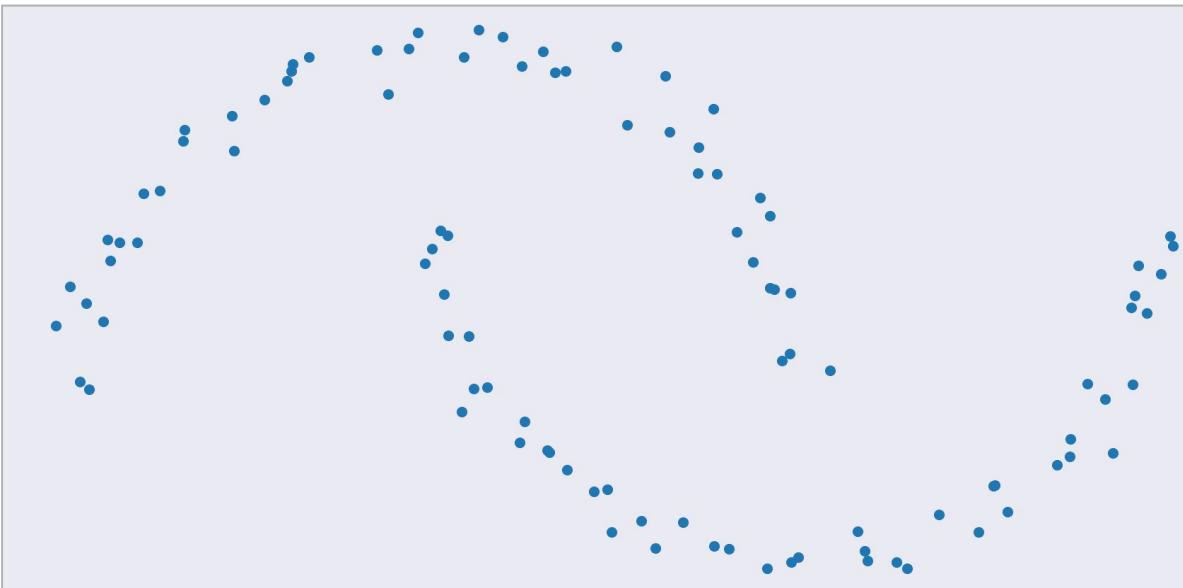
X, y = make_moons(n_samples=int(1e2),
                   noise=0.05, random_state=0)

X = pandas.DataFrame({'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)
```



# RAPIDS Matches Common Python APIs

## GPU-accelerated Clustering

```
from sklearn.datasets import make_moons
import cudf

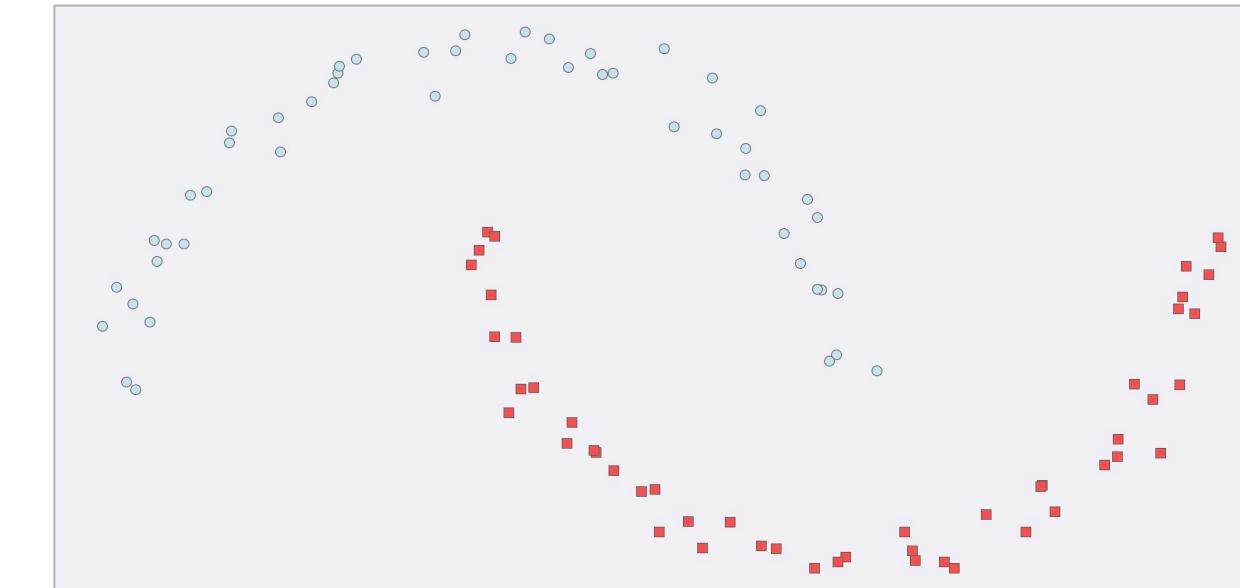
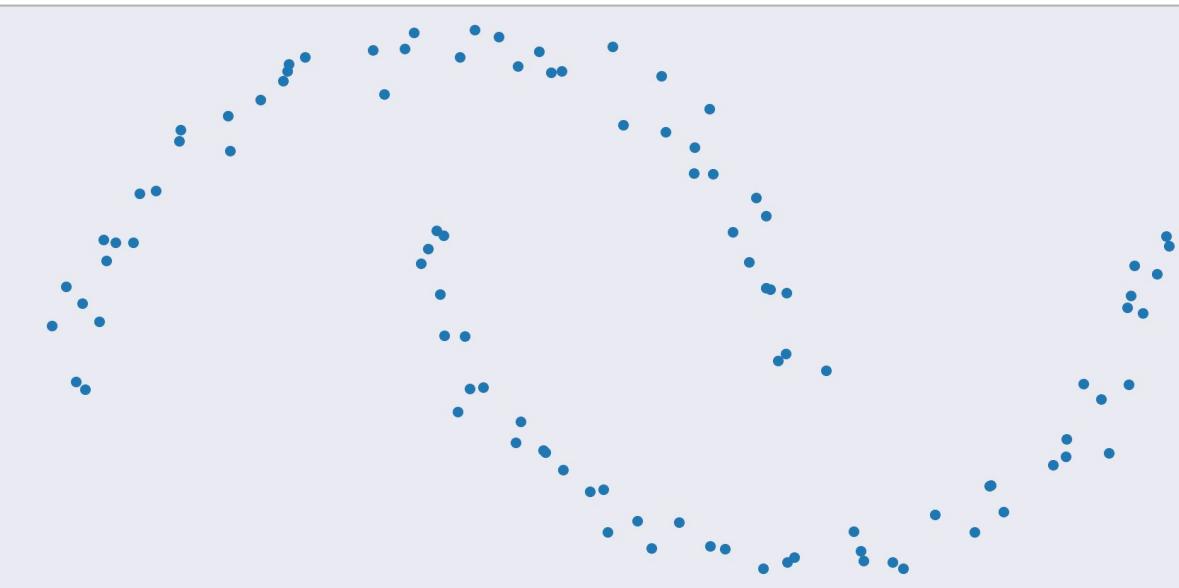
X, y = make_moons(n_samples=int(1e2),
                   noise=0.05, random_state=0)

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)

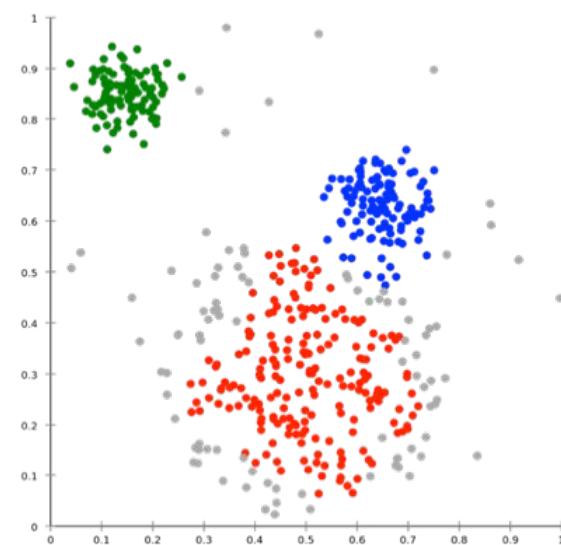
dbscan.fit(X)

y_hat = dbscan.predict(X)
```



# Algorithms

## GPU-accelerated Scikit-Learn



Cross Validation

Hyper-parameter Tuning

More to come!

Classification / Regression

Inference

Preprocessing

Clustering  
Decomposition &  
Dimensionality Reduction

Time Series

Decision Trees / Random Forests  
Linear/Lasso/Ridge/ElasticNet Regression  
Logistic Regression  
K-Nearest Neighbors  
Support Vector Machine Classification and  
Regression  
Naive Bayes

Random Forest / GBDT Inference (FIL)

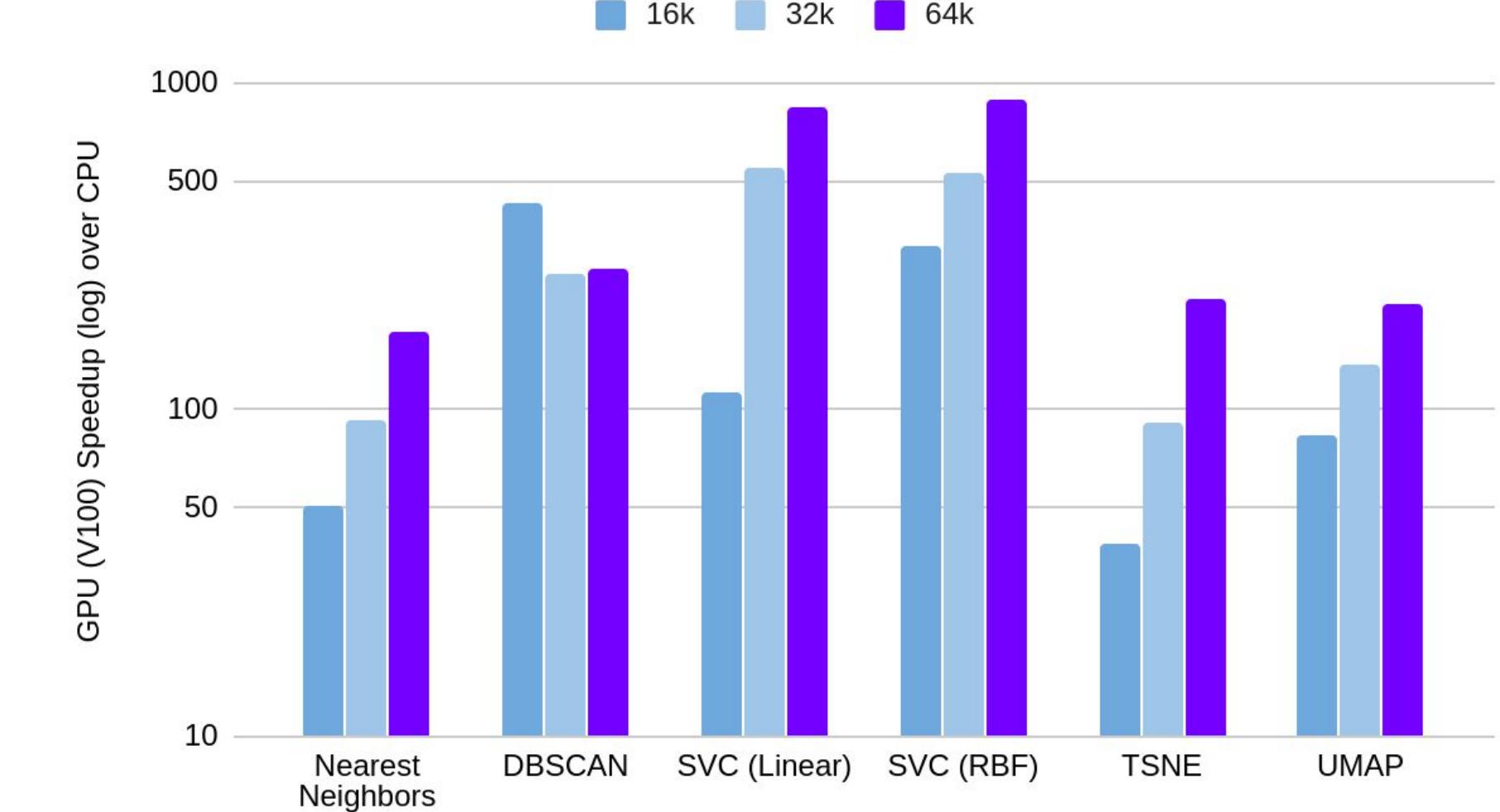
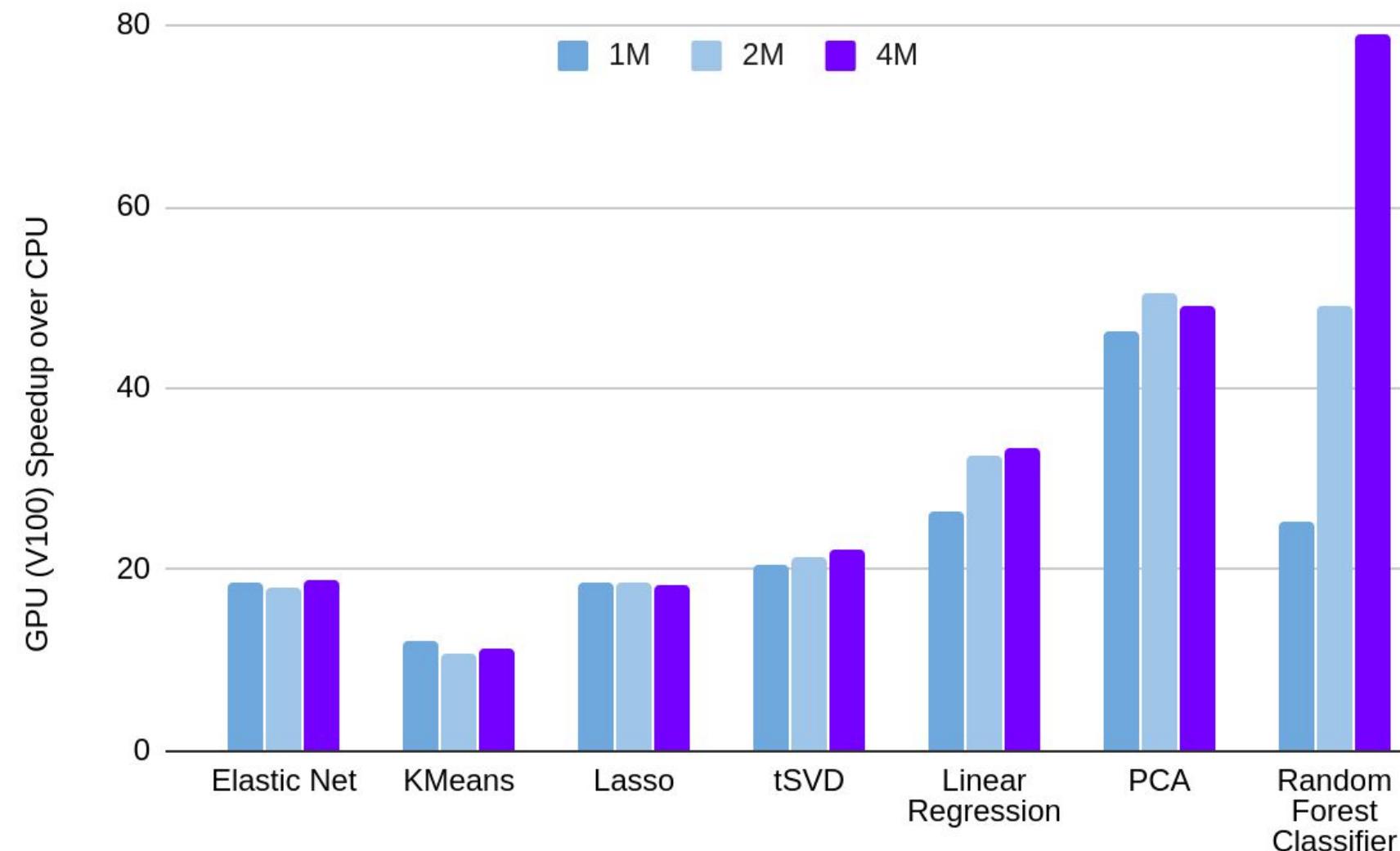
Text vectorization (TF-IDF / Count)  
Target Encoding  
Cross-validation / splitting

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

Holt-Winters  
Seasonal ARIMA / Auto ARIMA

*Key:*  
Preexisting | NEW or enhanced for 0.15

# Benchmarks: Single-GPU cuML vs Scikit-learn (WIP)



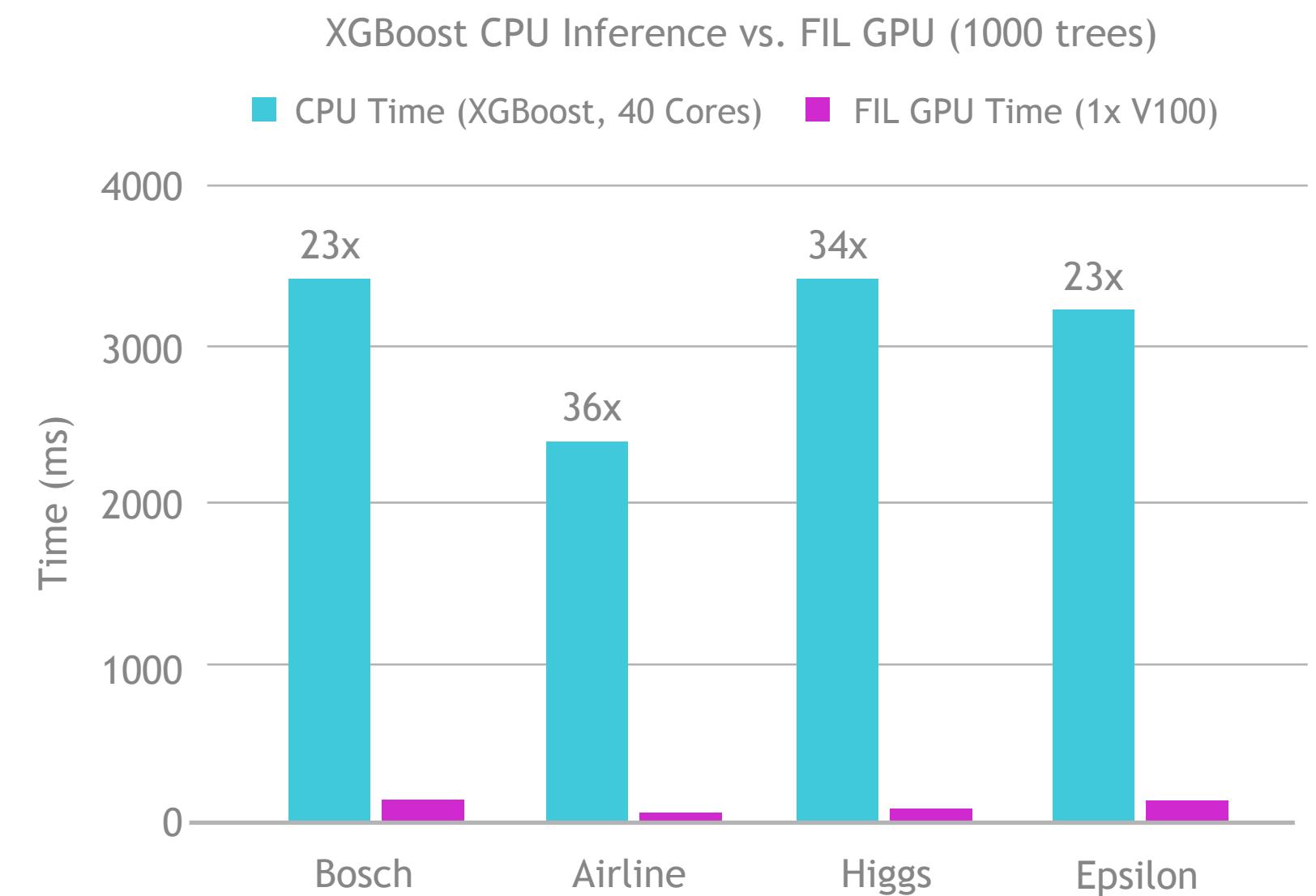
1x V100 vs. 2x 20 Core CPUs (DGX-1, RAPIDS 0.15)

# Forest Inference

## Taking Models From Training to Production

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

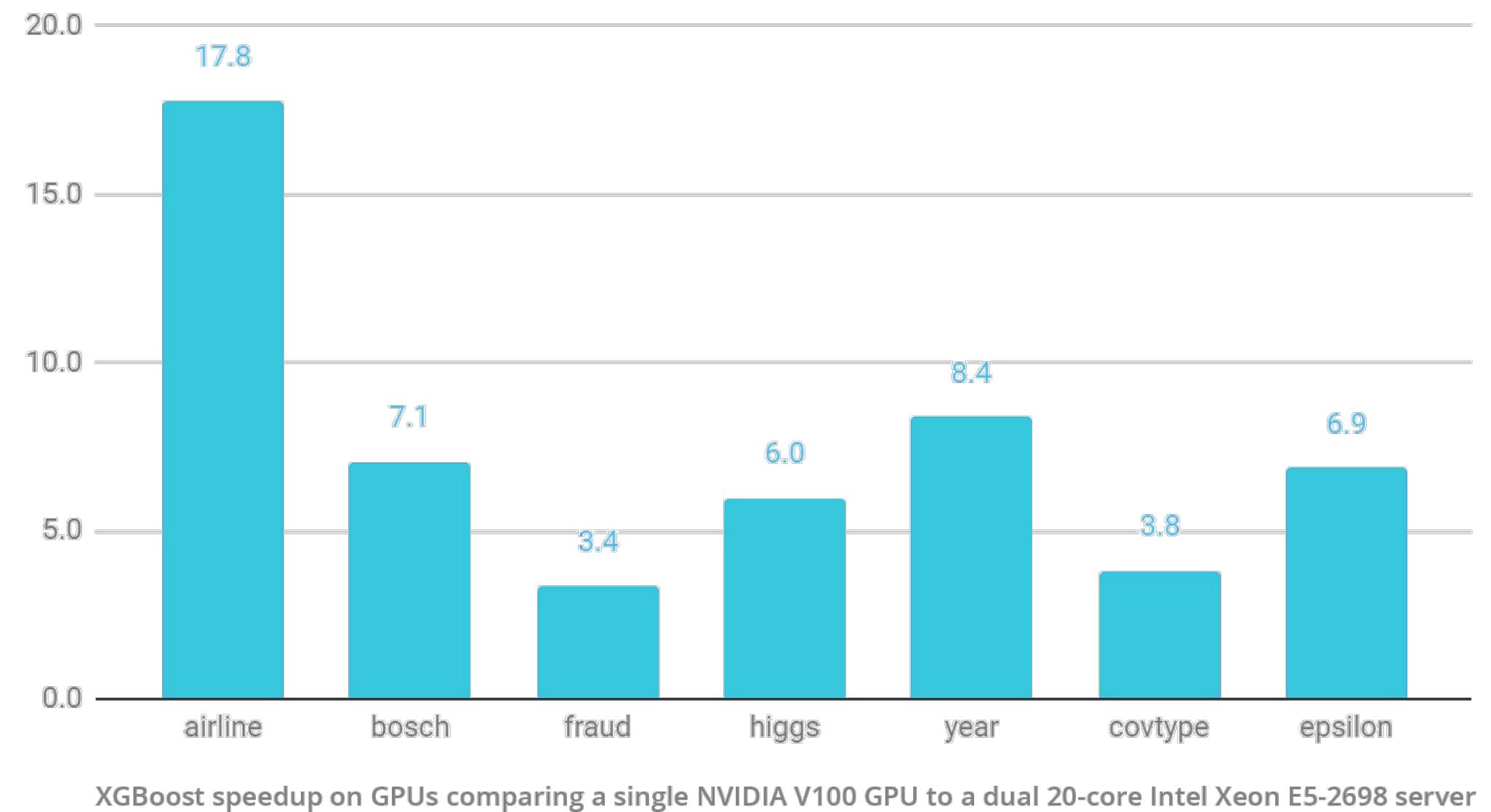
- Works with existing saved models (XGBoost, LightGBM, scikit-learn RF cuML RF soon)
- Lightweight Python API
- Single V100 GPU can infer up to 34x faster than XGBoost dual-CPU node
- Over 100 million forest inferences



# XGBoost + RAPIDS: Better Together

- ▶ RAPIDS comes paired with XGBoost 1.1 (as of 0.14)
- ▶ 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
- ▶ New objectives support Learning to Rank on GPU

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



# 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.

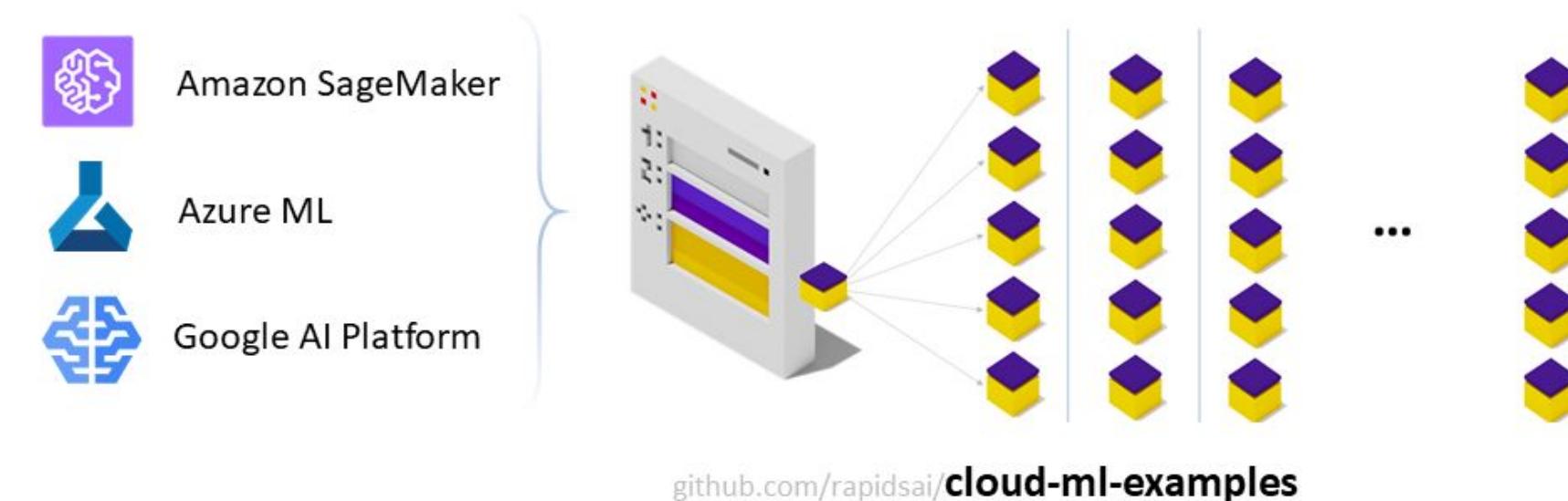
With GPU acceleration, RAPIDS models can train 40x faster than CPU equivalents, enabling more experimentation in less time.

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>

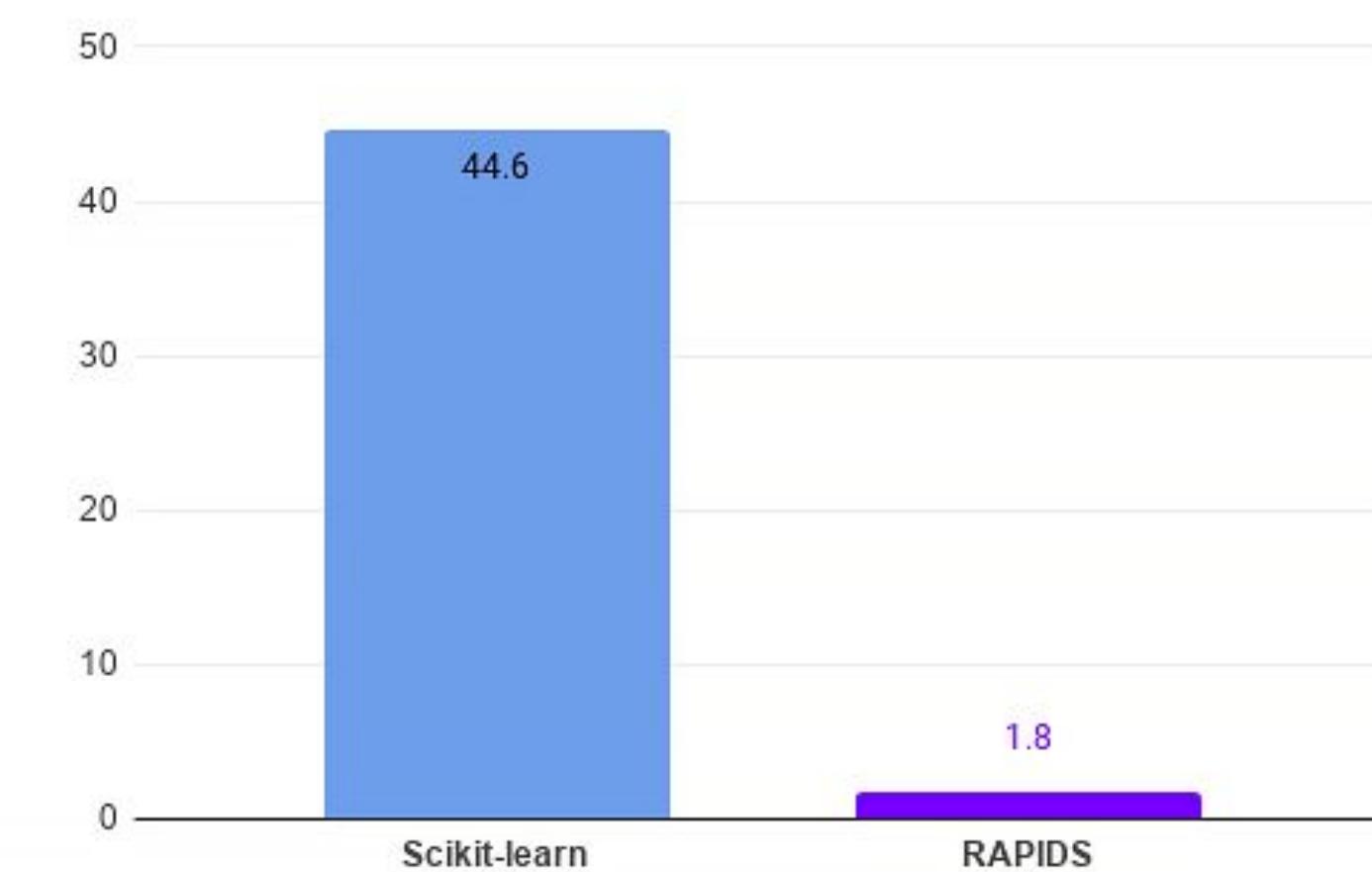
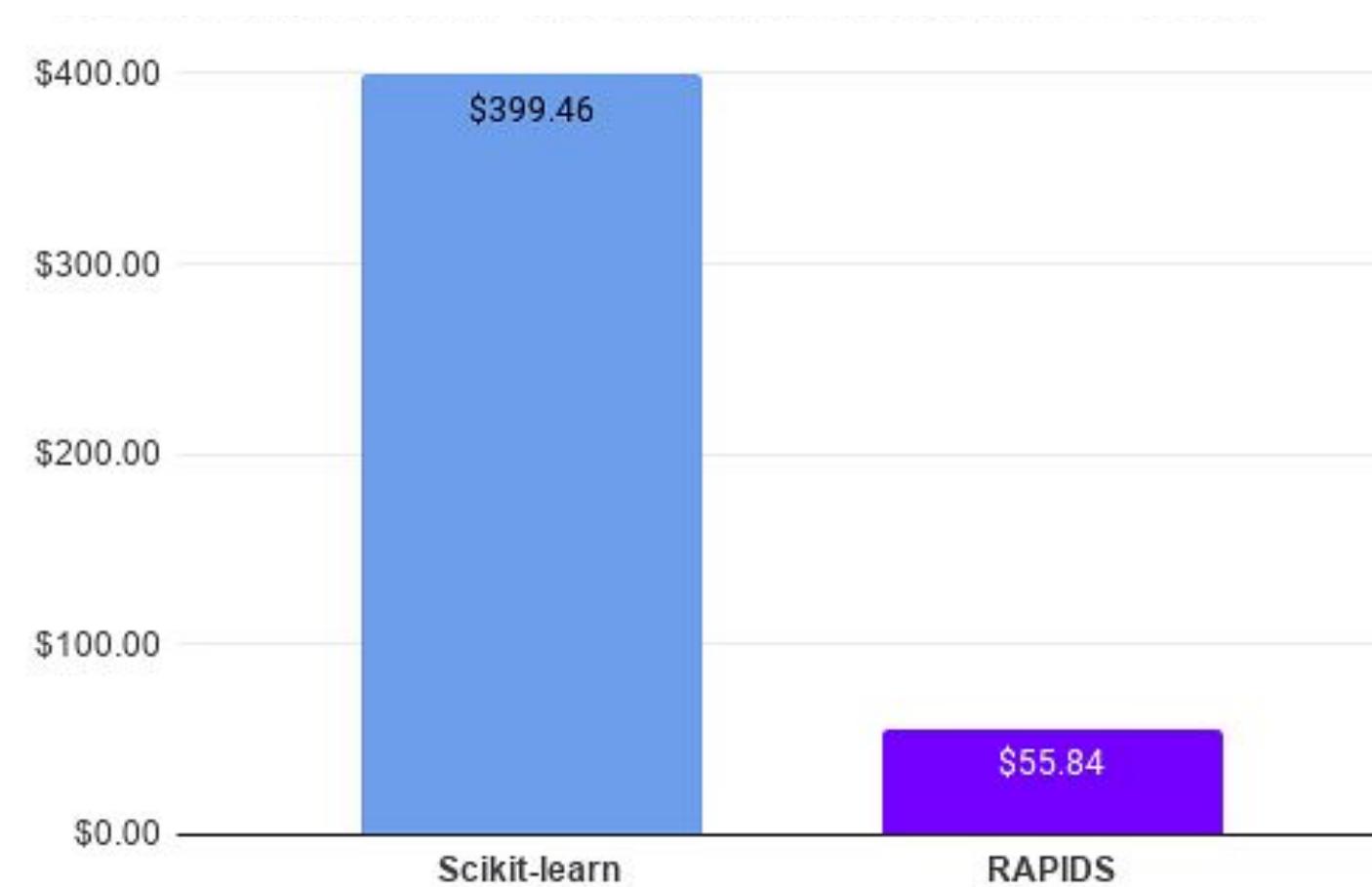
## MANY PATHS TO RAPIDS HPO

RAPIDS Integration into Cloud/Distributed Frameworks



# 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"

# Road to 1.0 - cuML

June 2020 - RAPIDS 0.15

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		

# Road to 1.0 - cuML

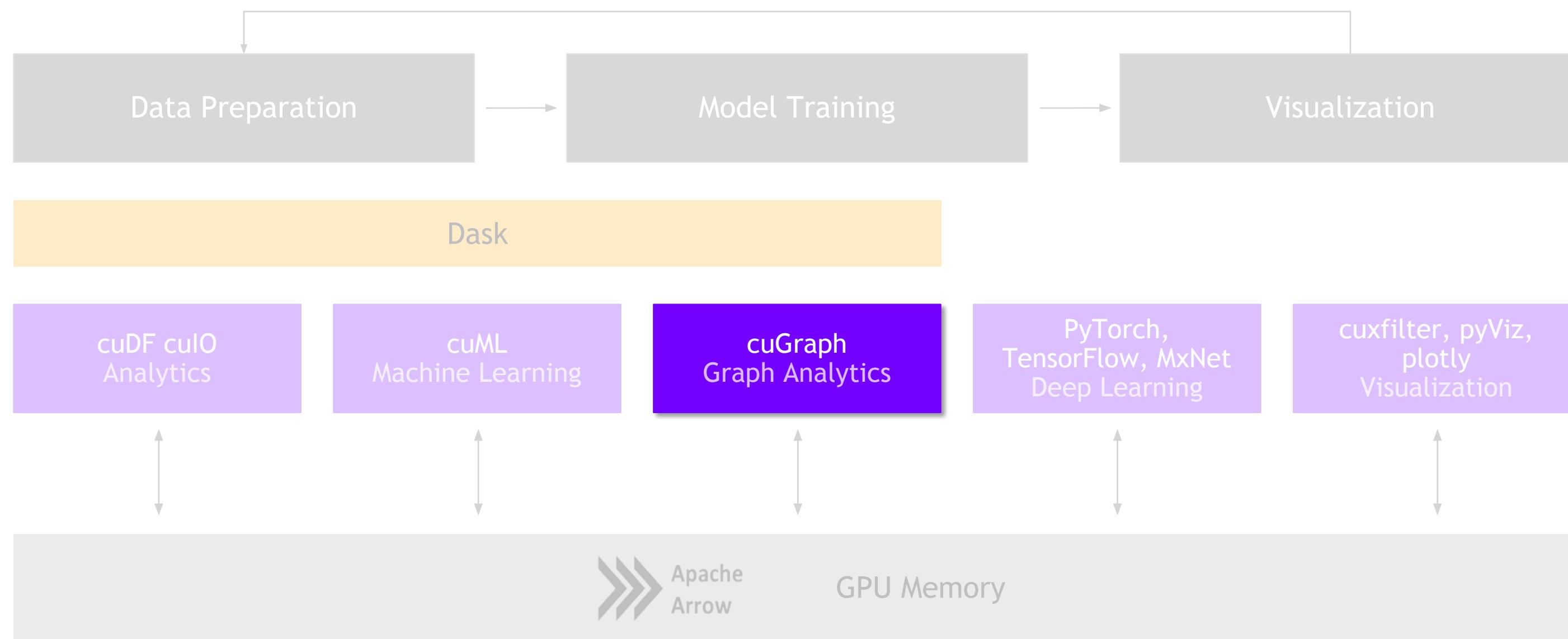
## 2020 EOY Plan

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

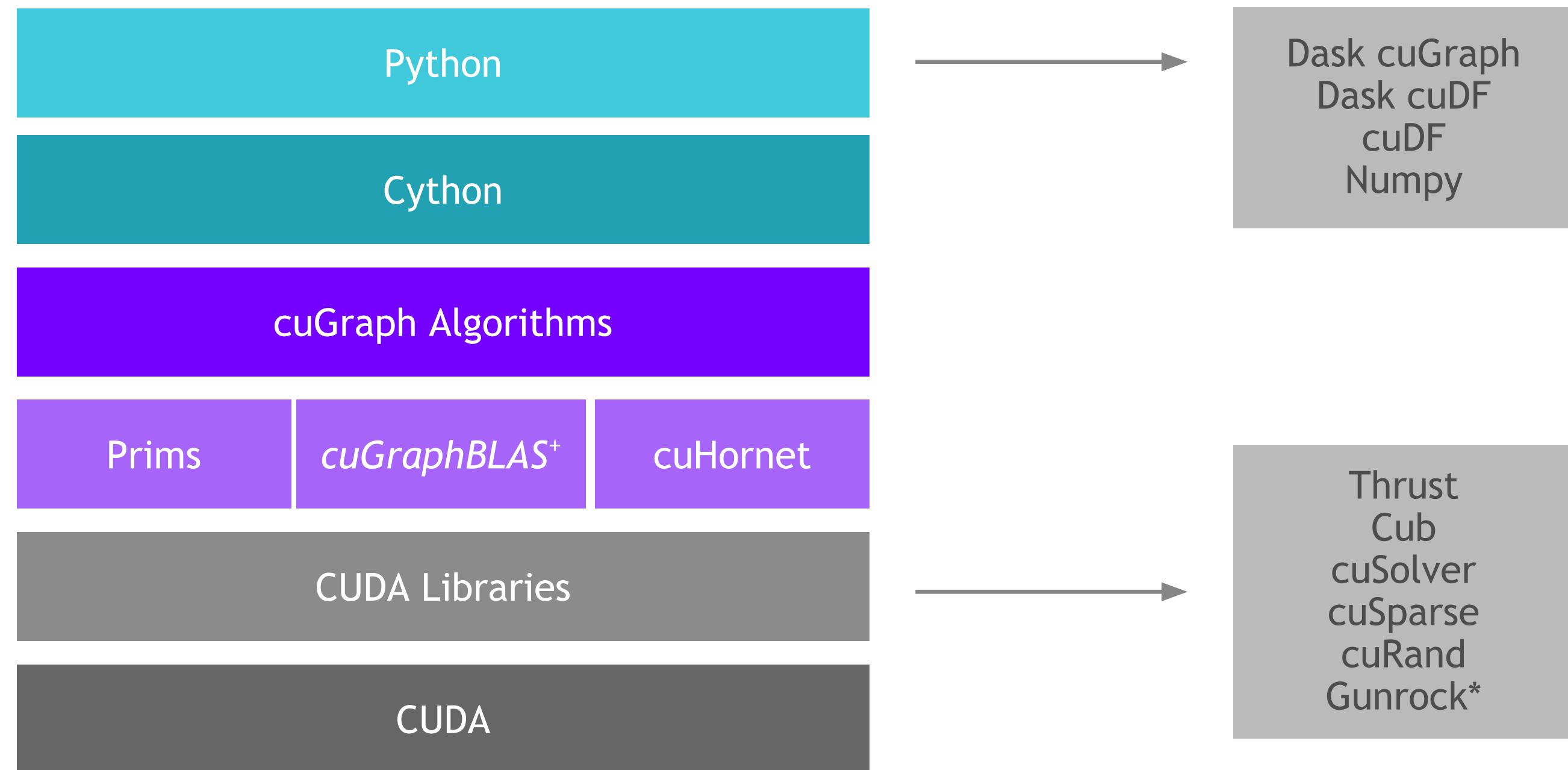
### MULTIPLE APIs

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

### GROWING FUNCTIONALITY

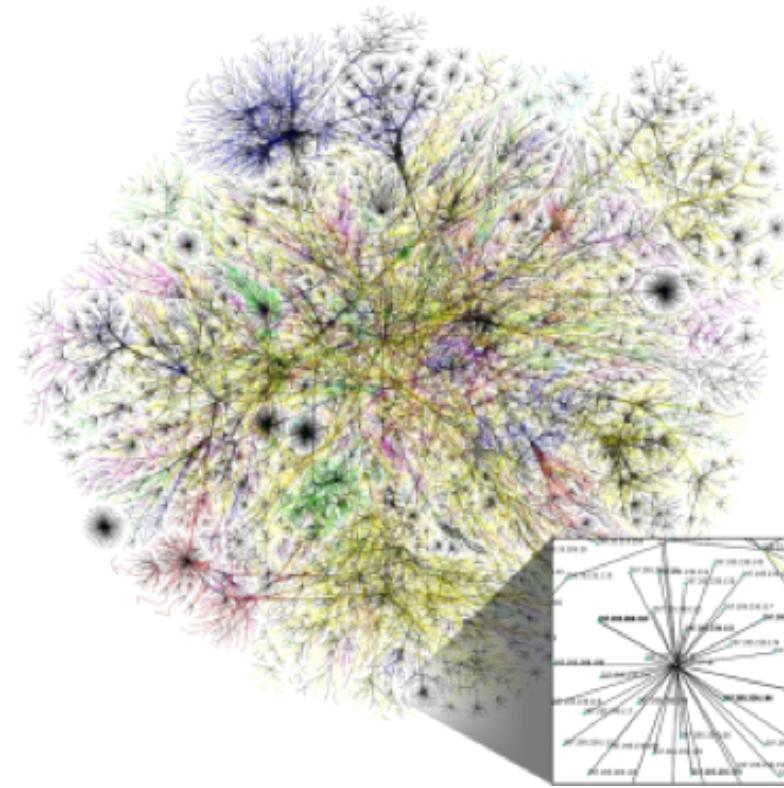
- ▶ Extensive collection of algorithm, primitive, and utility functions

# Graph Technology Stack



<sup>+</sup>*cuGraphBLAS* is still in development and will be ready late 2020

\* Gunrock is from UC Davis



# Algorithms

## GPU-accelerated NetworkX

Graph Classes  
Subgraph Extraction

Structure

Renumbering  
Auto-Renumbering  
Force Atlas 2

Utilities

Community

Spectral Clustering - Balanced Cu and Modularity Maxim  
Louvain 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  
*HITS*

Link Prediction

Jaccard  
Weighted Jaccard  
Overlap Coefficient

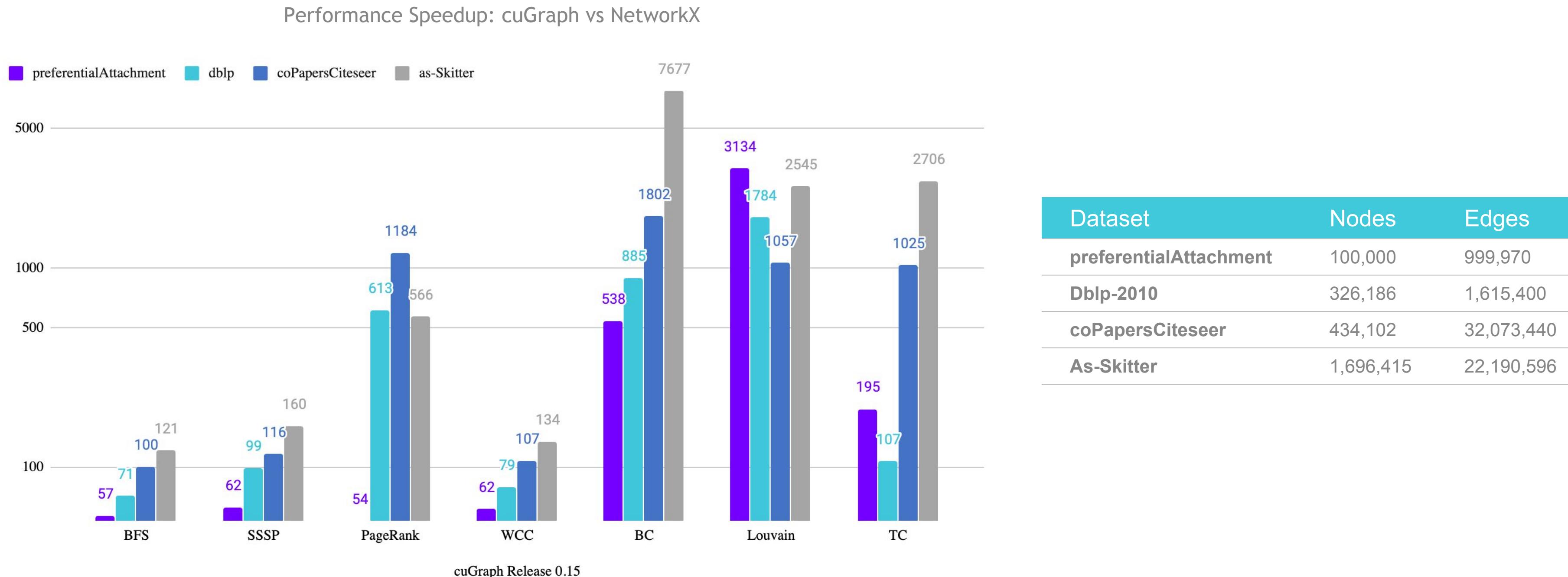
Traversal

Single Source Shortest Path (SSSP)  
Breadth First Search (BFS)

Centrality

Katz  
*Betweenness Centrality (Vertex and Edge)*

# Benchmarks: Single-GPU cuGraph vs NetworkX



# 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!

# Road to 1.0

## December 2019 - RAPIDS 0.15

cuGRAPH	Single-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	
Leiden	

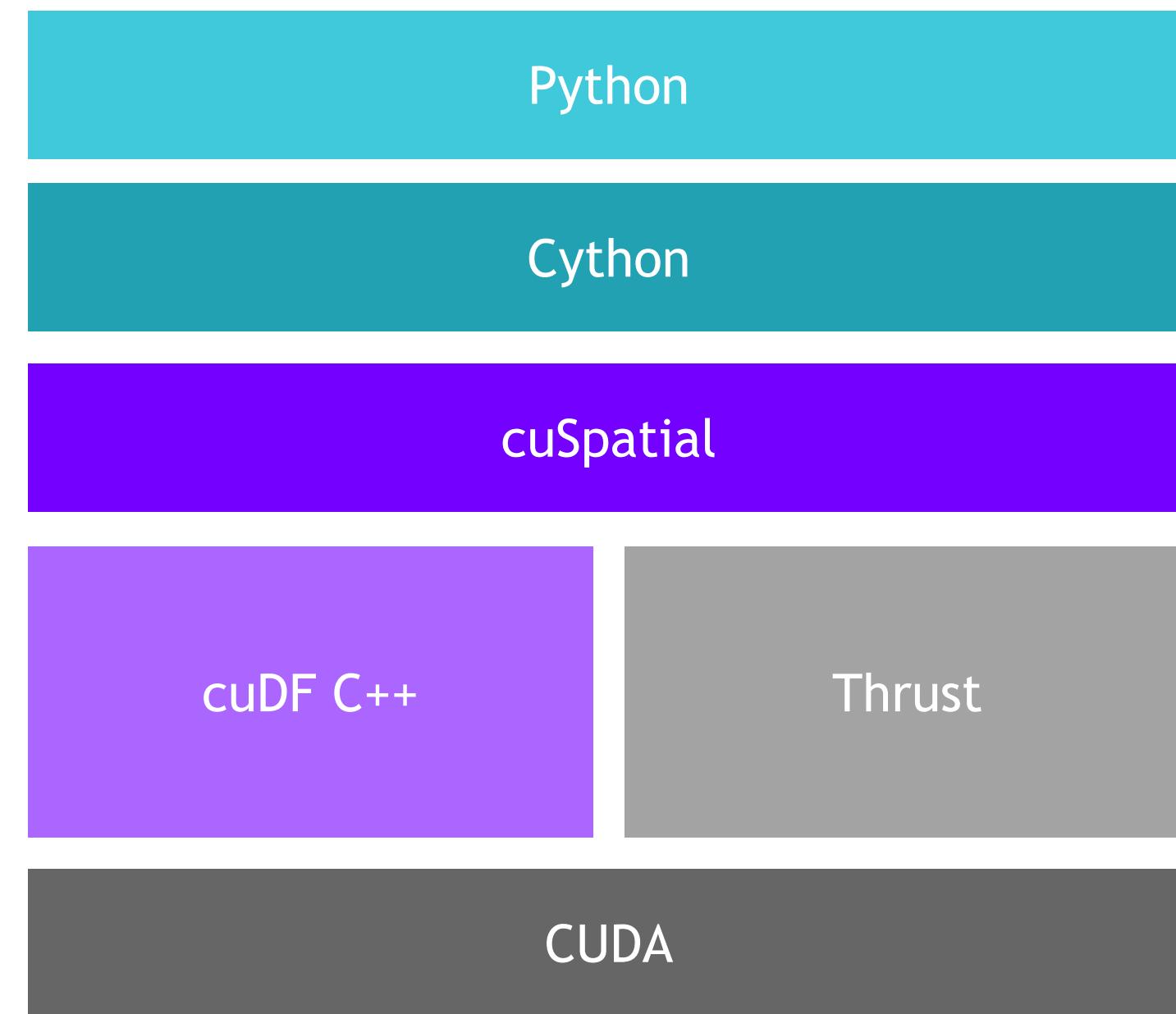
# Road to 1.0

## 2020 EOY Plan

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

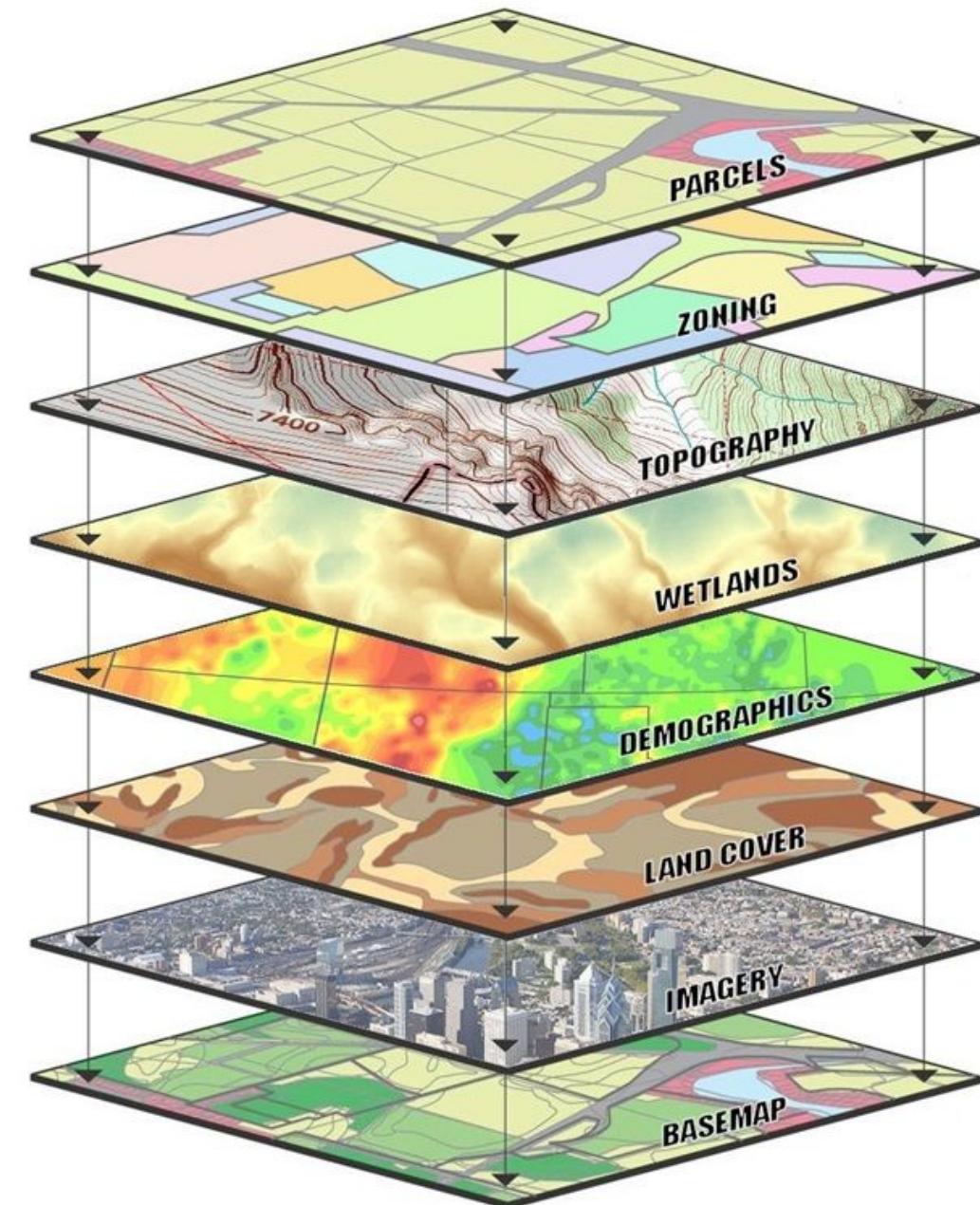
- Up to 1000x faster than CPU spatial libraries
- Python and C++ APIs for maximum usability and integration

## 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

## 0.15 and Beyond

Layer	0.15 Functionality	Functionality Roadmap (0.16)
<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

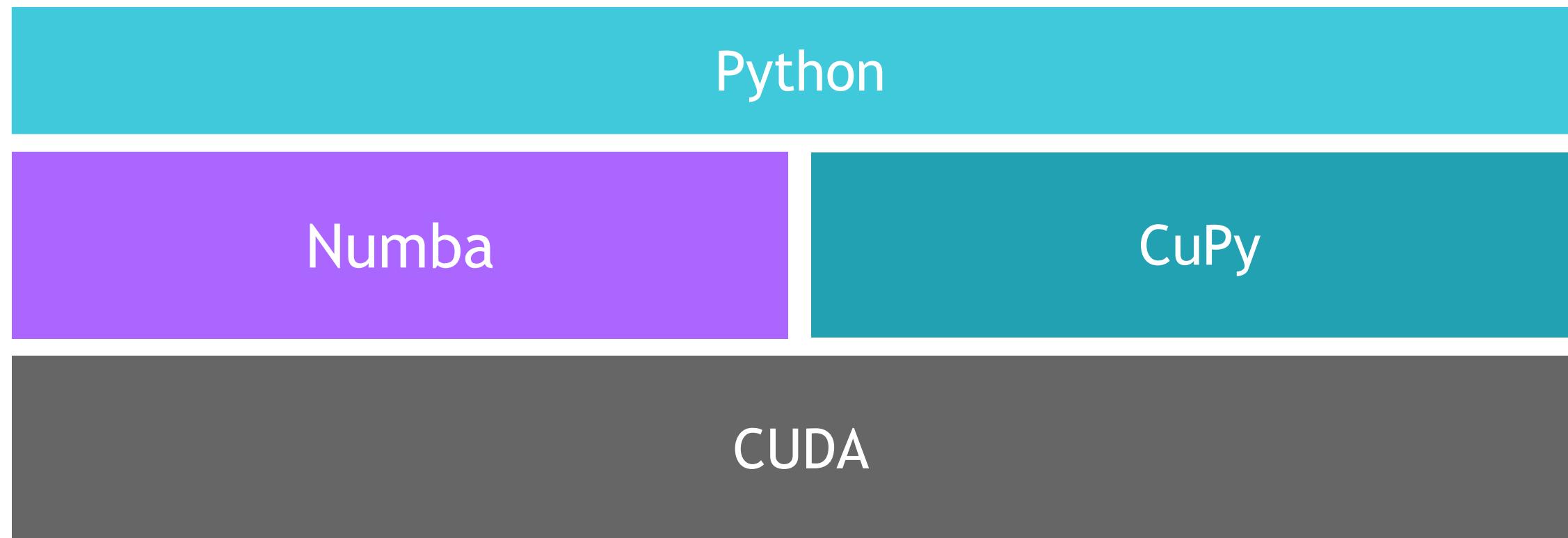
# cuSpatial

## 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



Convolution

Convolve/Correlate  
FFT Convolve  
Convolve/Correlate 2D

Filtering and Filter Design

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

Waveform Generation

Chirp  
Square  
Gaussian Pulse

Wavelets

Kaiser  
Blackman  
Hamming  
Hanning

Window Functions

Peak Finding

Periodogram  
Welch  
Spectrogram

Spectral Analysis

More to come!

# 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()

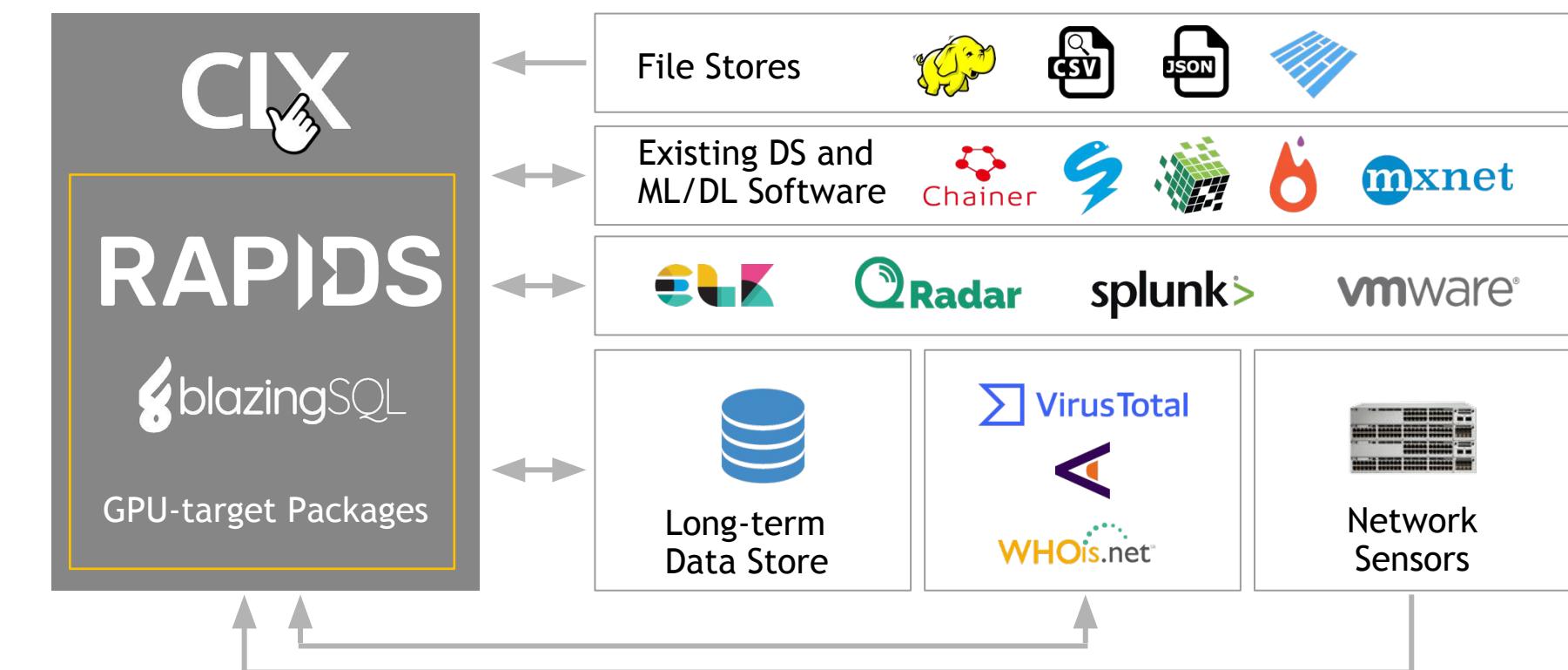
866 µs ± 38 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)
```

# Cyber Log Accelerators

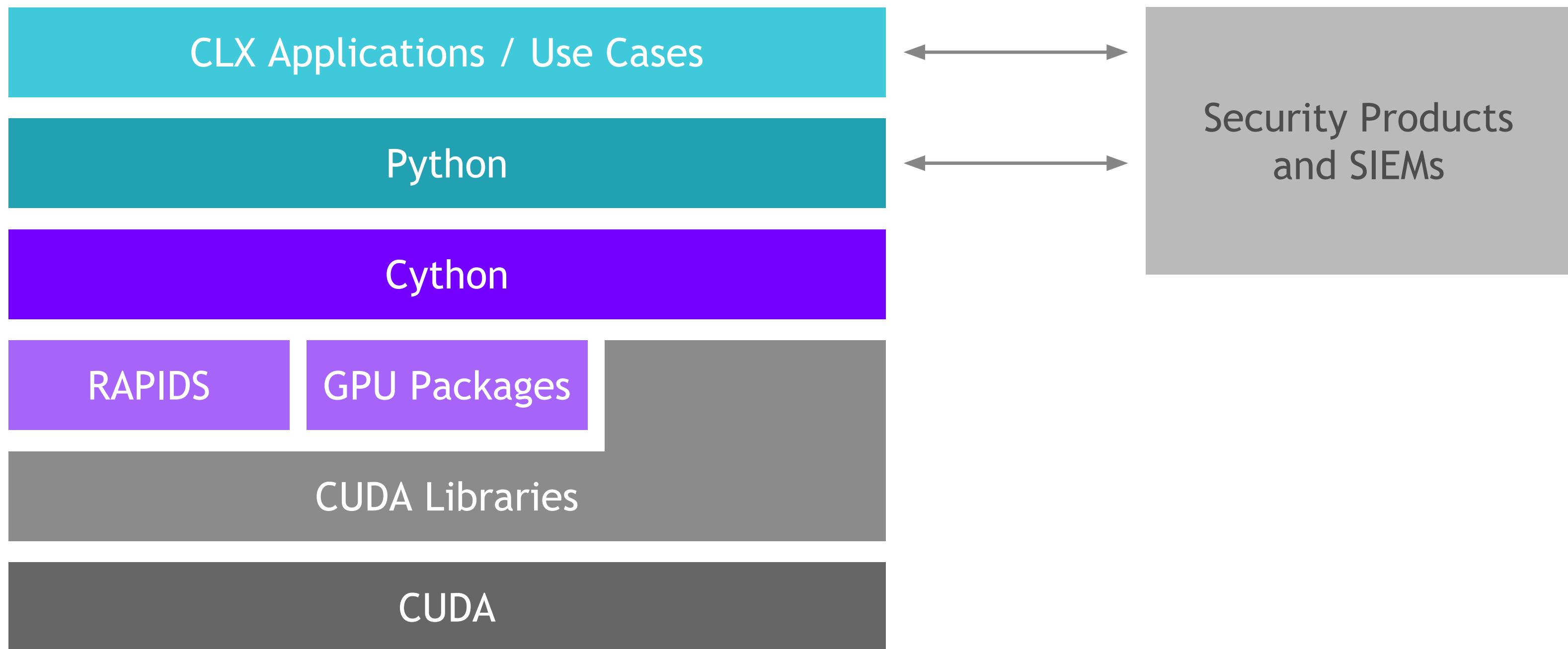
# CLX

## 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		
Splunk Integration	Integration		
CLX Query	Integration		
cyBERT	Log Parsing		
GPU Wordpiece Tokenizer	Pre-Processing		
Accelerated IPv4	Primitive		
Accelerated DNS	Primitive		

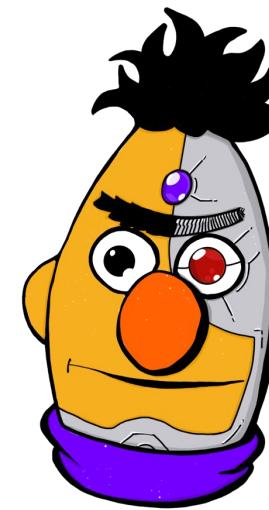
The screenshot shows a Jupyter Notebook window titled "jupyter 10min-clx". It contains three tabs:

- Network Mapping with RAPIDS and CLX**: This tab displays a heatmap visualization titled "Heatmap Visualization with cuXfilter" showing alerts per hour. The code in the cell uses cuDF to filter and aggregate data from a CSV file.
- Domain Generation Algorithm (DGA) Detection**: This tab shows a snippet of code for DGA detection using cuDF, including imports for pandas, numpy, and cuDF, along with data loading and filtering logic.
- cyBERT: a flexible log parser based on the BERT language model**: This tab contains a brief introduction to the cyBERT project and its GitHub repository.

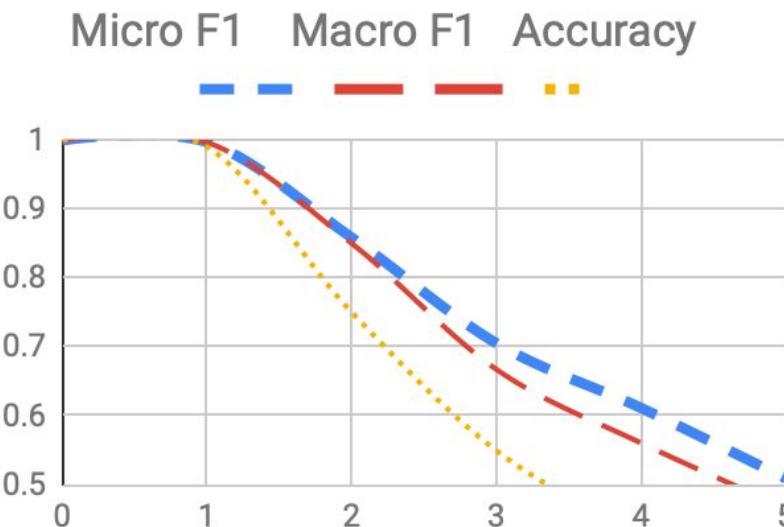
# cyBERT

## AI Log Parsing for Known, Unknown, and Degraded Cyber Logs

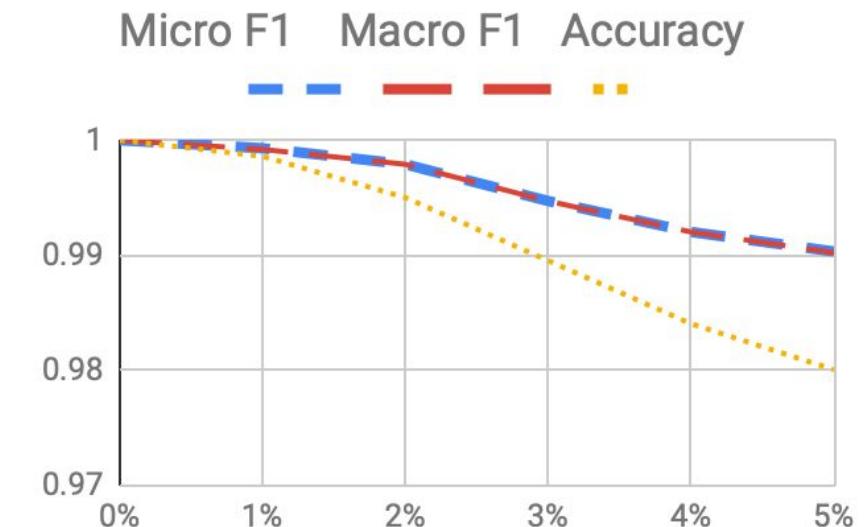
- ▶ Provide a flexible method that does not use heuristics/regex to parse cybersecurity logs
- ▶ Built using RAPIDS and PyTorch, tested with a variety of language models (including BERT)
- ▶ Parsing with **micro-F1** and **macro-F1 > 0.999** across heterogeneous log types with a **validation loss of < 0.0048**
- ▶ Second version ~160x (min) faster than first version due to creation of the first all-GPU wordpiece tokenizer that supports non-truncation of logs/sentences



### Log Parsing with Inserted Values



### Log Parsing with Missing Values

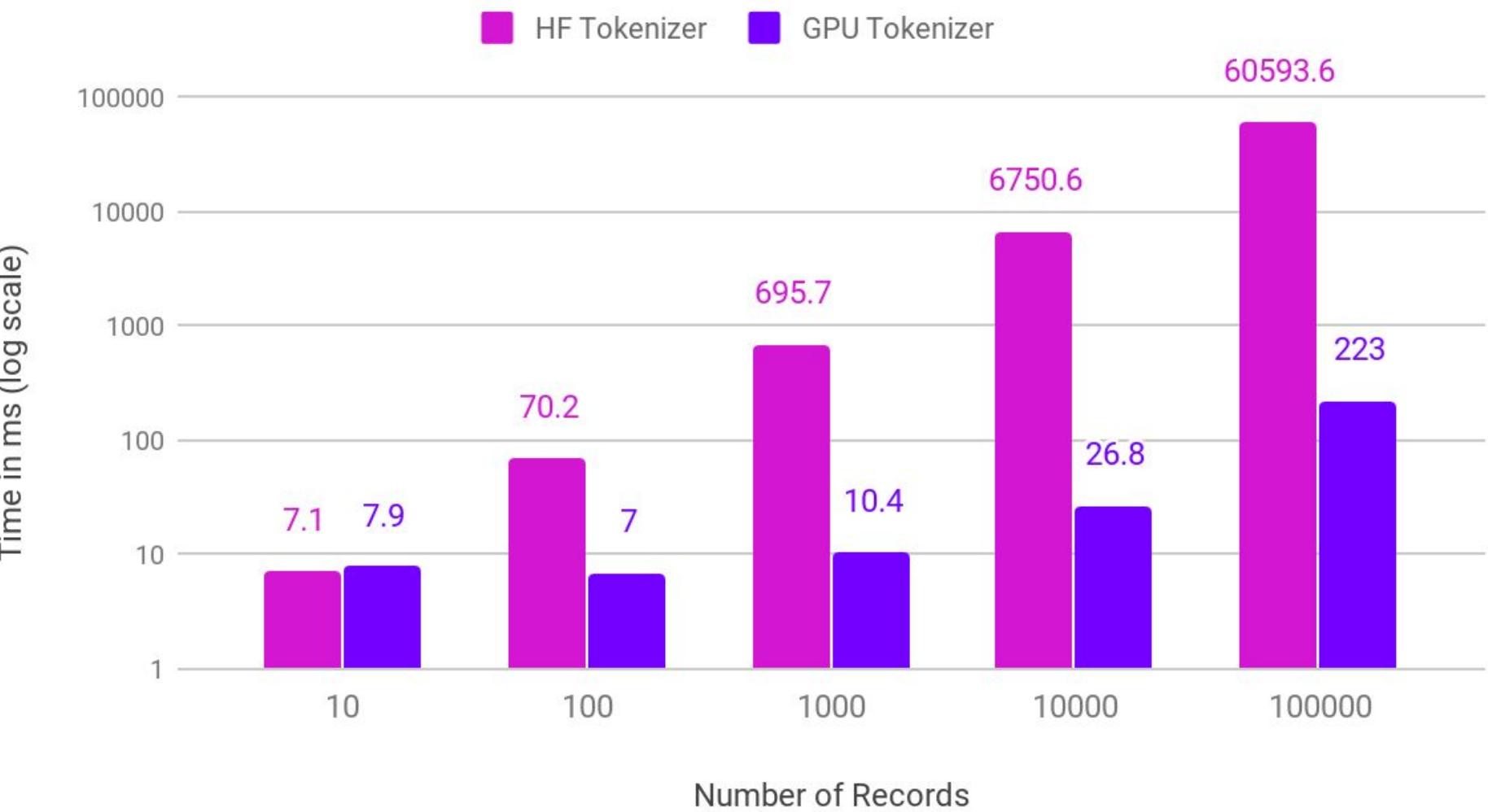


# GPU SubWord Tokenizer

## 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
- ▶ Up to 270x faster than production-ready Hugging Face wordpiece tokenizer (Python version)

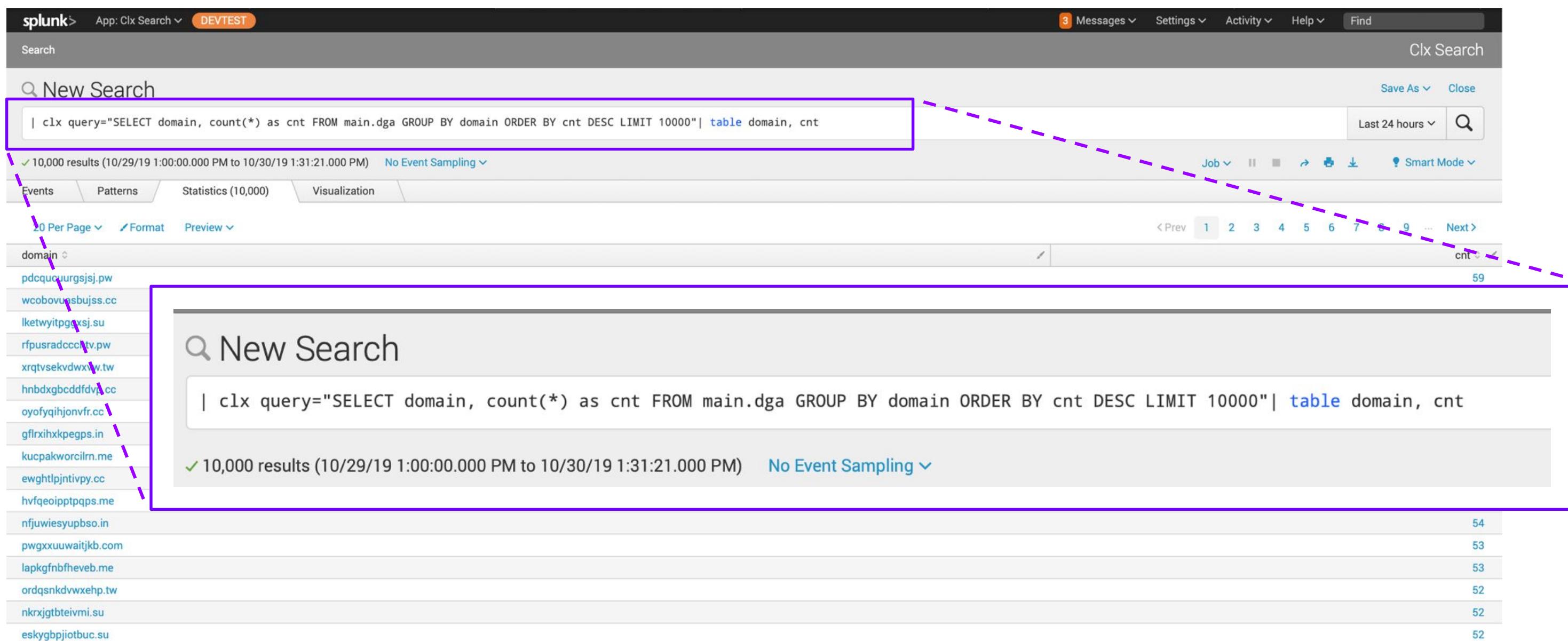
SubWord Tokenizer Speedup Comparison



CPU numbers ran on 2x Intel Xeon E5 v4 @ 2.2 GHz. GPU numbers ran on 1x NVIDIA Tesla V100

# CLX Query

## Query Long-Term Data Store Directly from Splunk



# RAPIDS



# Visualization

# cuXfilter

## GPU Accelerated Cross-Filtering

### STREAMLINED FOR NOTEBOOKS

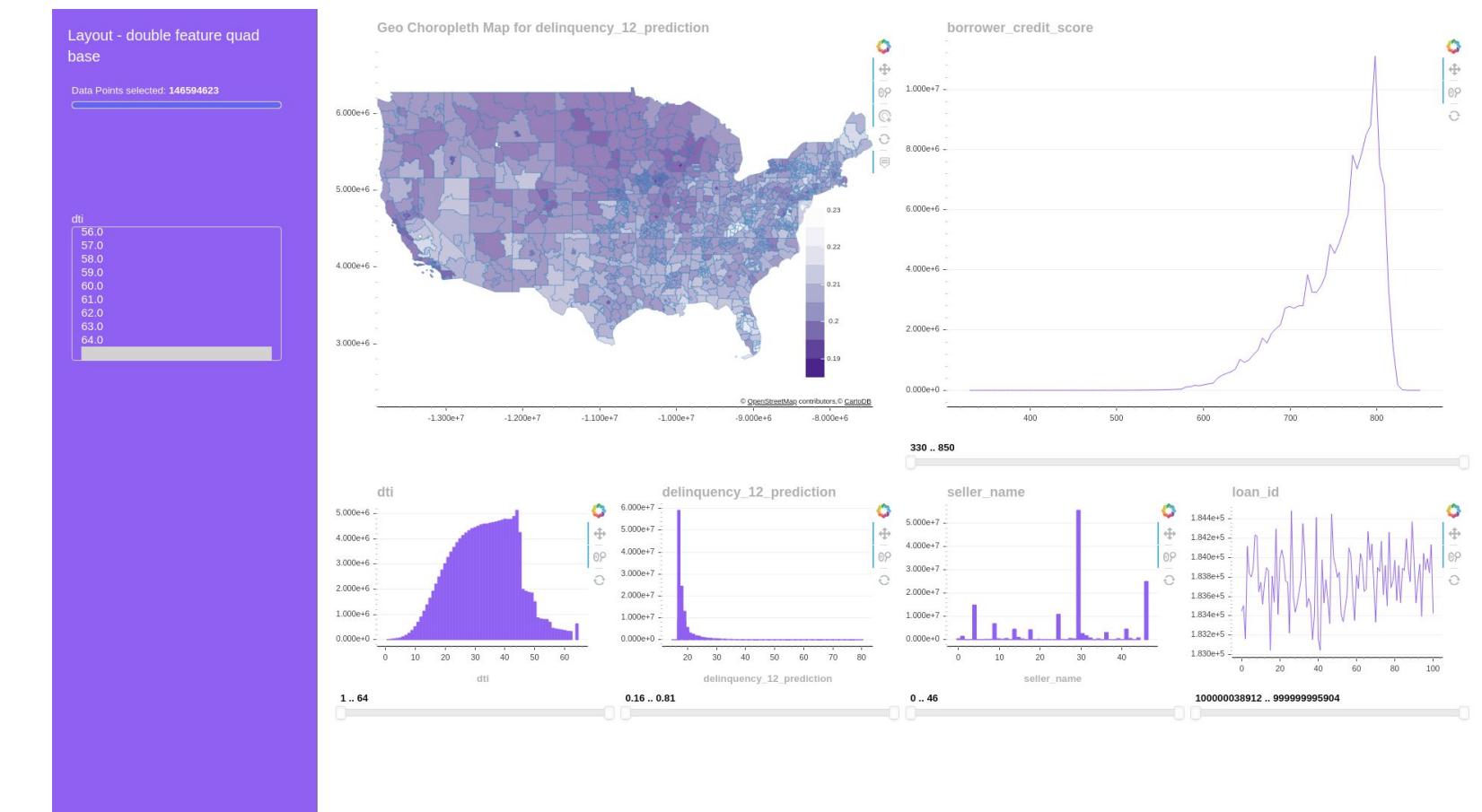
Cuxfilter allows you to visually explore your cuDF dataframes through fully cross-filtered dashboards in less than 10 lines of notebook code.

### MINIMAL COMPLEXITY & FAST RESULTS

Select from vetted chart libraries, pre-designed 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 to be easy to [install](#) and use with RAPIDS. Learn more about our approach [here](#).



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

# pyViz Integrations

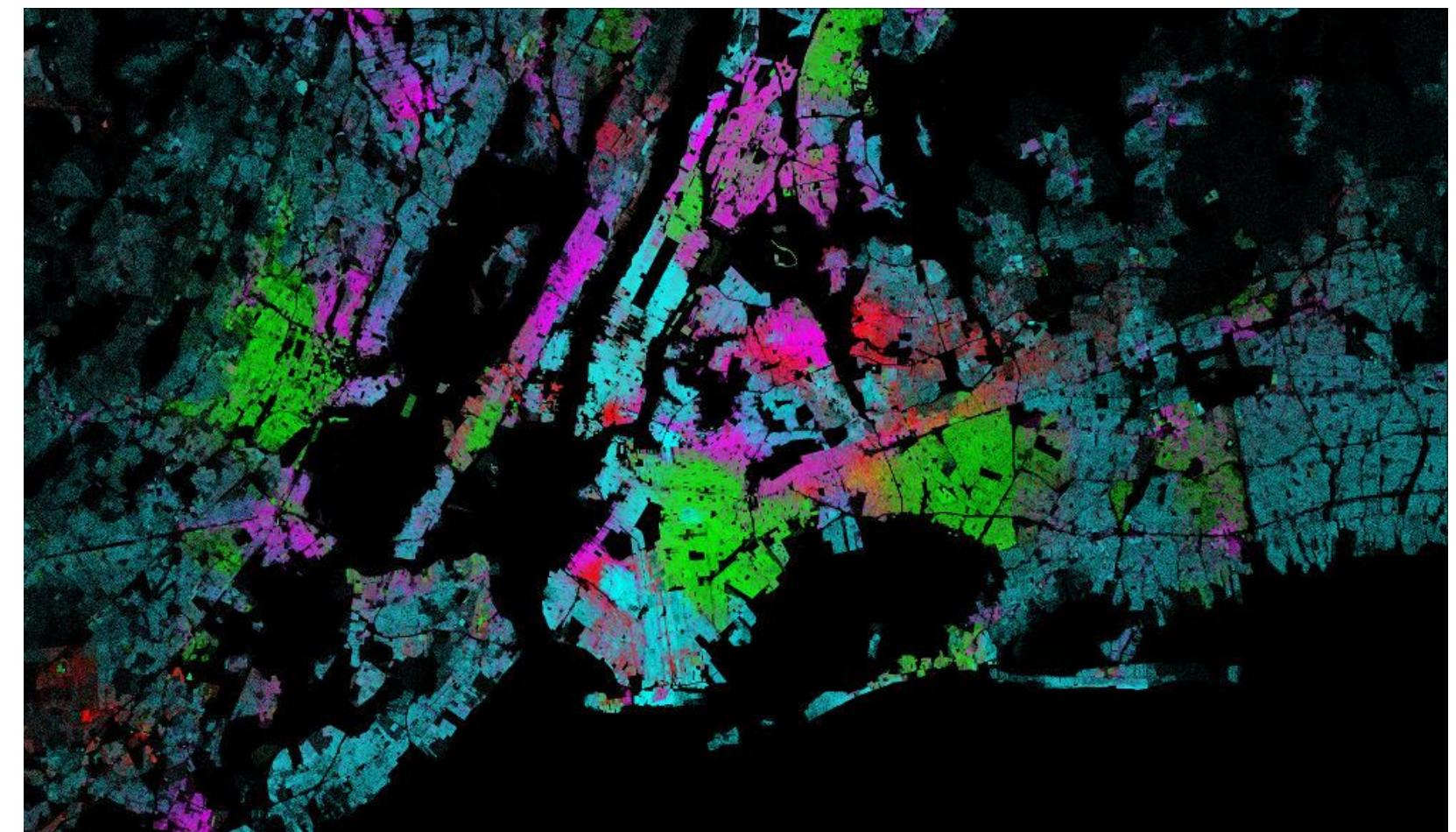
## GPU Accelerated Python Visualizations

### ACCELERATED PYTHON VIZ

The RAPIDS focus on python makes it straightforward to integrate GPU acceleration for most python visualization libraries. We are currently collaborating with bokeh, plotly, and datashader, among others.

### HIGHLIGHT: DATASHADER INTEGRATION

Datashader's ability to visualize billions of points makes it well suited for GPU acceleration. It now includes cuDF integration as well as other enhanced capabilities that offer between a 10-120x speed up. More details [here](#).



<https://datashader.org/>

# Plotly Dash

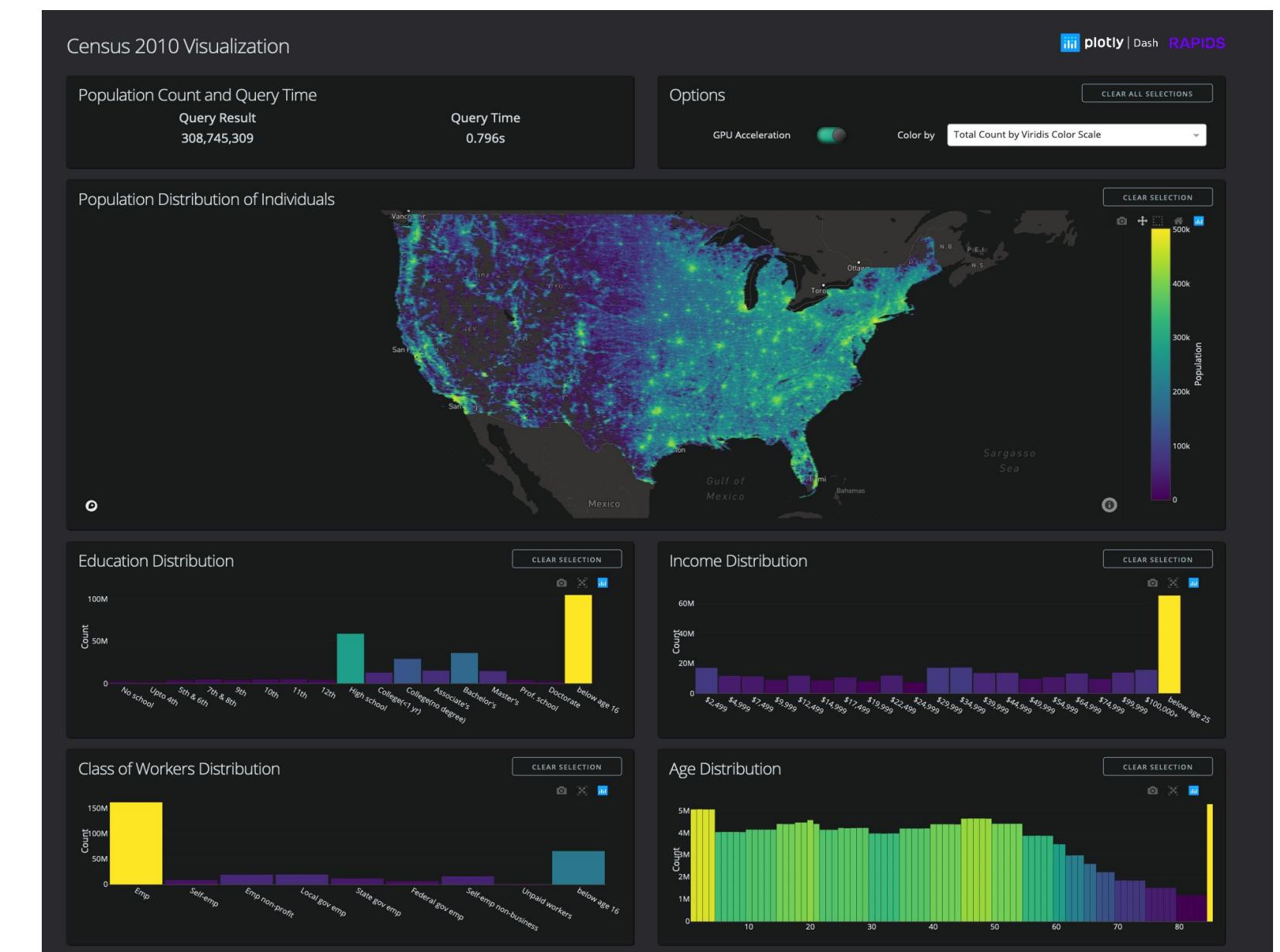
## GPU Accelerated Visualization Apps with Python

### PLOTLY & RAPIDS PARTNERSHIP

Plotly Dash now has RAPIDS acceleration, enabling the ease-of-use development and deployment of Dash apps, with the speed of RAPIDS. Find out more on RAPIDS.ai and Plotly's announcement.

### 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>

# Community

# Ecosystem Partners

## CONTRIBUTORS



## ADOPTERS

Booz | Allen | Hamilton



Uber



## OPEN SOURCE



nucleo



# 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

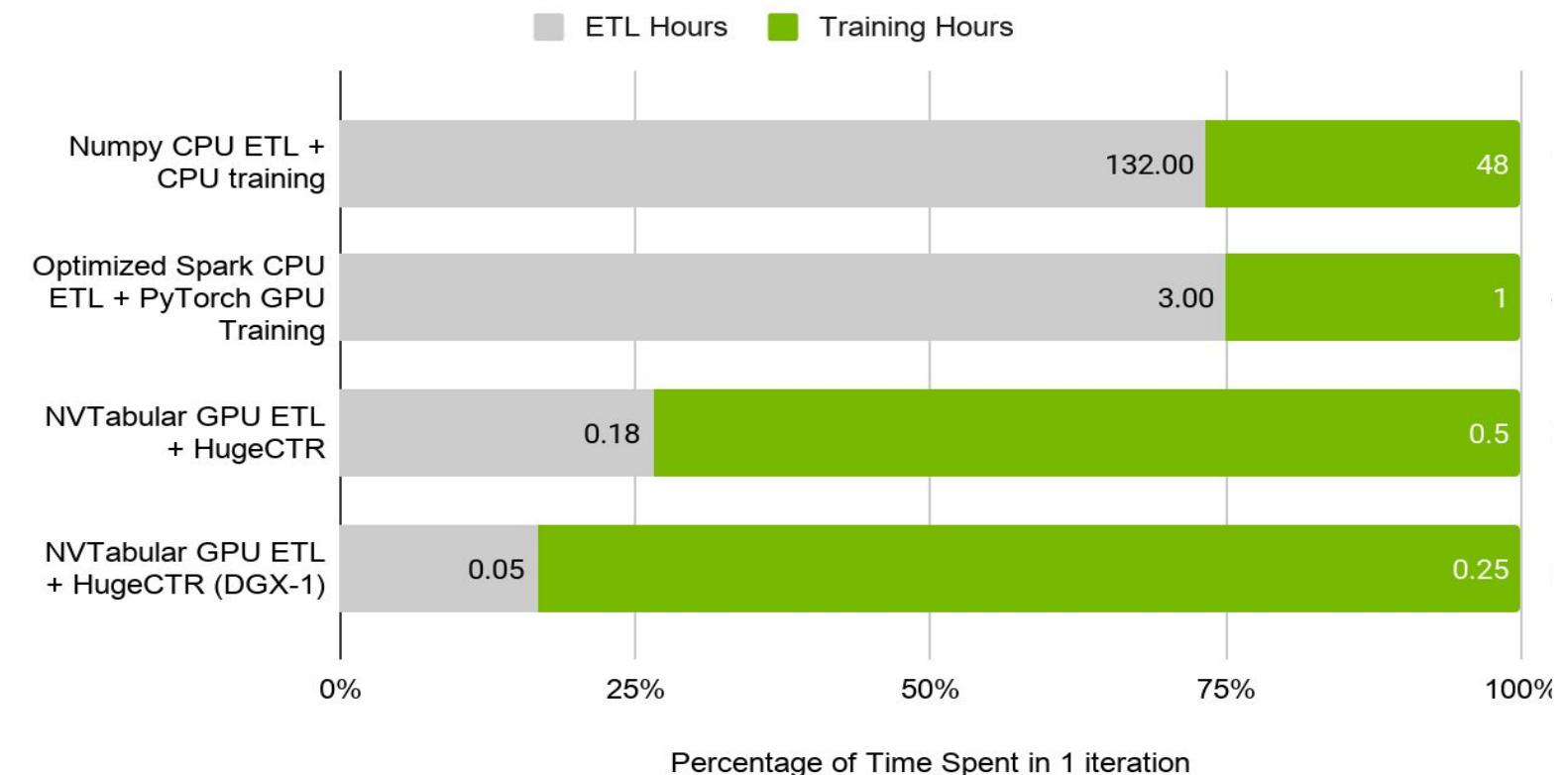
## ETL library for recommender systems building off of RAPIDS and Dask

### 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

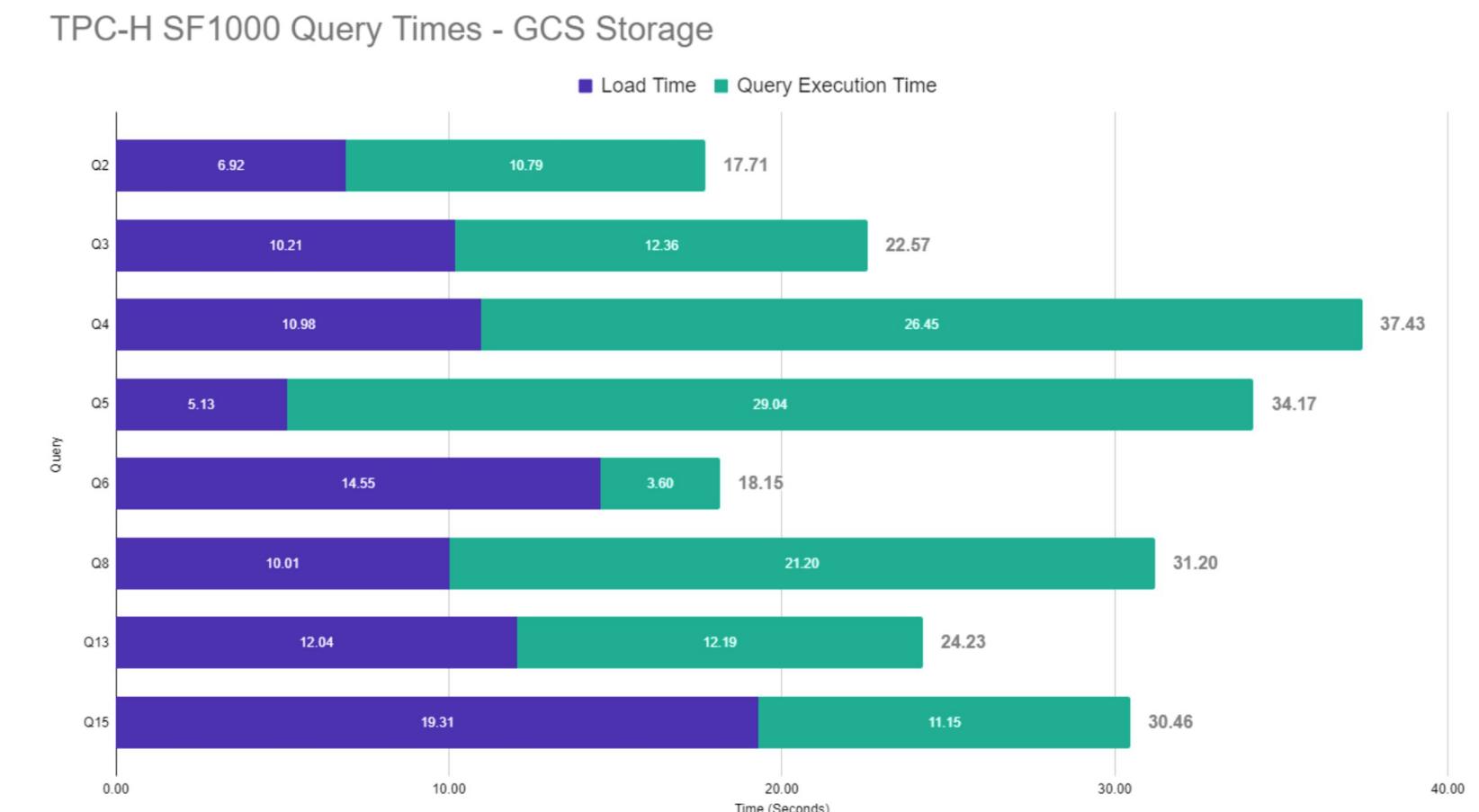
## GPU-accelerated SQL engine built with RAPIDS

### BLAZING FAST SQL ON RAPIDS

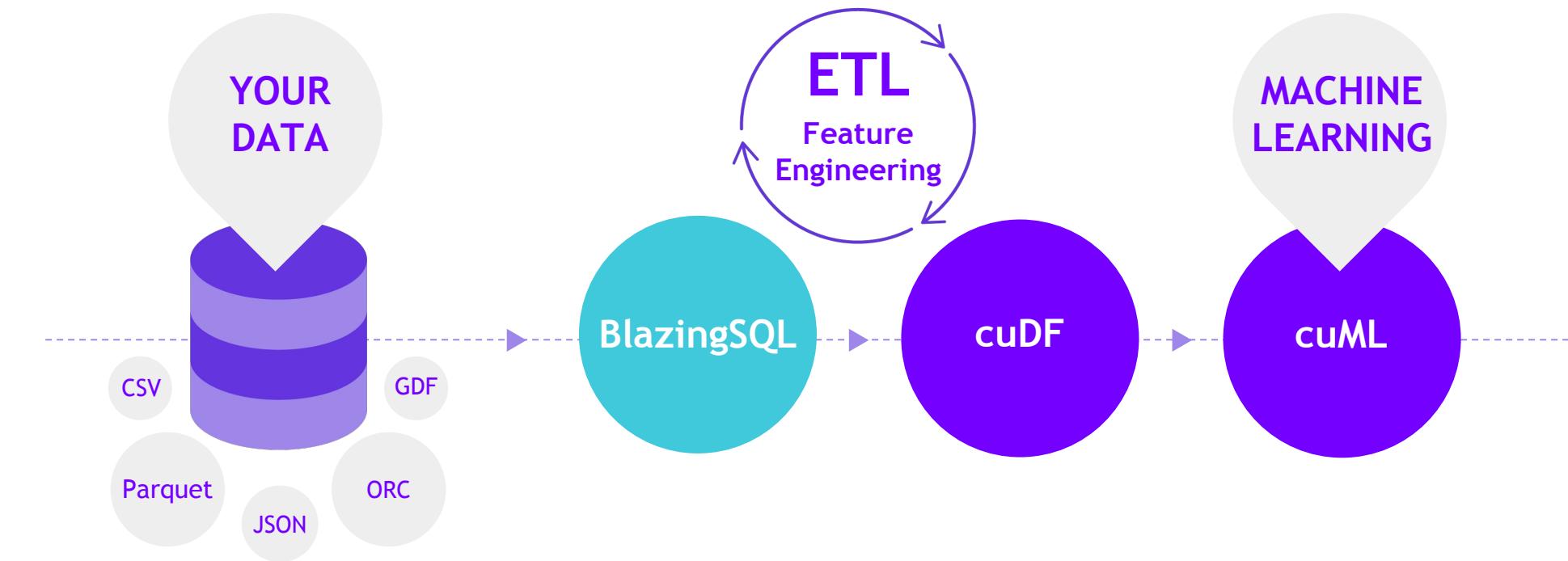
- ▶ Incredibly fast distributed SQL engine on GPUs--natively compatible with RAPIDS!
- ▶ Allows data scientists to easily connect large-scale data lakes to GPU-accelerated analytics
- ▶ Directly query raw file formats such as CSV and Apache Parquet inside Data Lakes like HDFS and AWS S3, and directly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



# 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

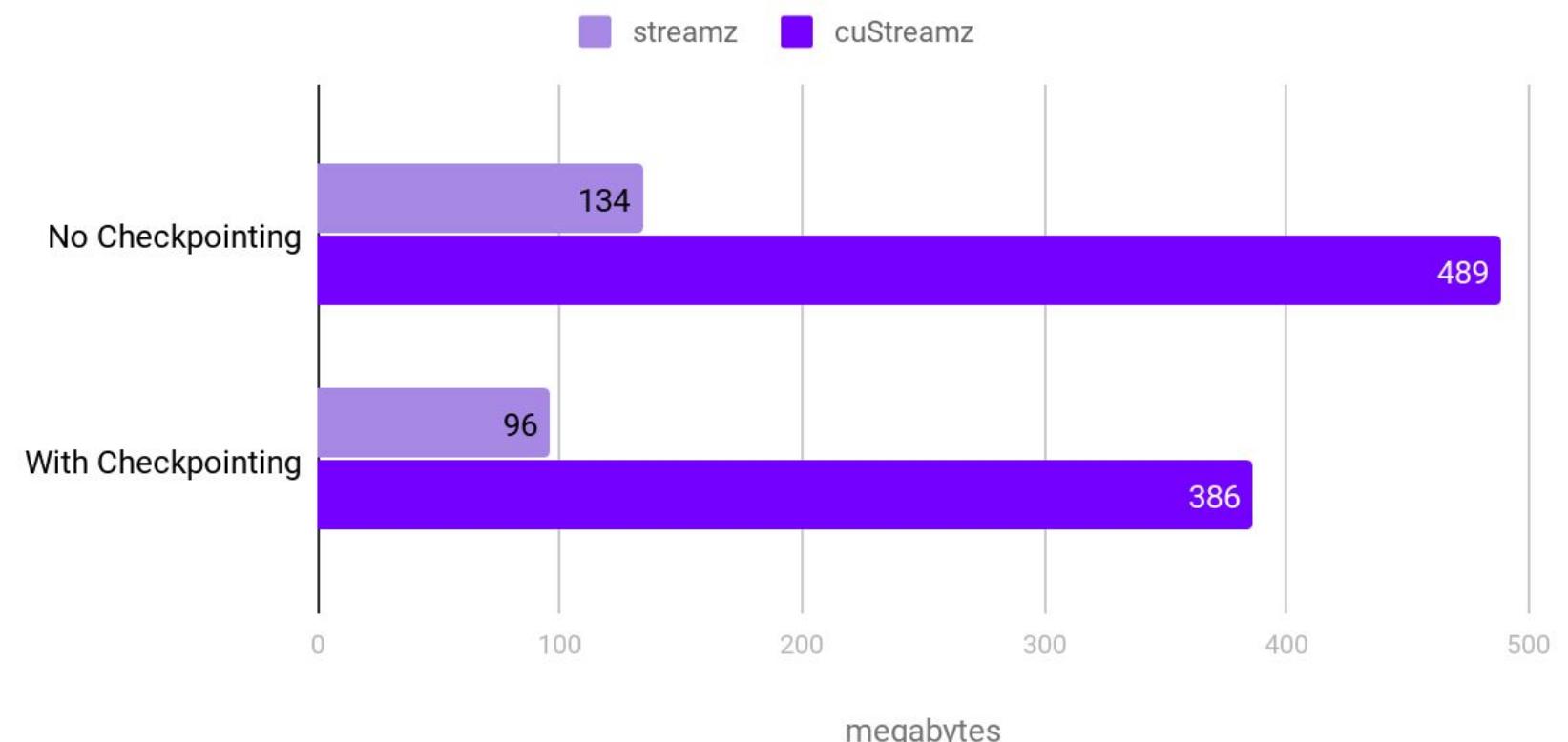
Ingesting Kafka messages to cudf is increased by roughly 3.5 - 4X over standard cpu ingestion. Streaming TCO is lowered by using cheaper VM instance types.

### CHECKPOINTING

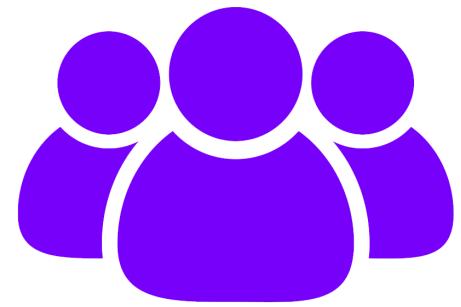
Streaming job version of “where was I?” Streams can gracefully handle errors by continuing to process a stream at the exact point they were before the error occurred.

### Kafka throughput

Single Node

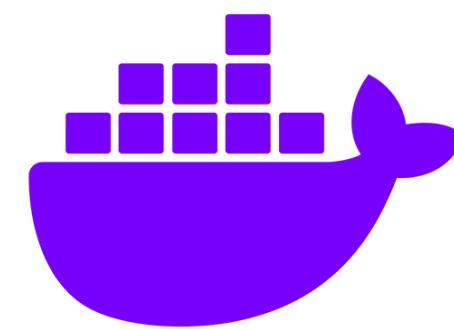


# Join the Conversation



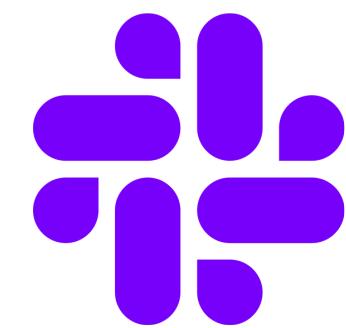
## GOOGLE GROUPS

<https://groups.google.com/forum/#!forum/rapidsai>



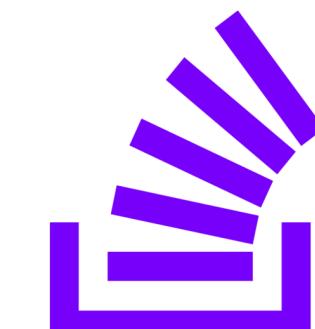
## DOCKER HUB

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



## SLACK CHANNEL

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



## STACK OVERFLOW

<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 Mungoli [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 using [Docker](#), [Conda](#), or [Colab](#).
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 repos will **rename** stable/release branches in v0.15
- ⚠️ Python 3.6 & CUDA 10.0 EOL in v0.14
- ⚠️ Release changes to **clx** and **cuxfilter** in v0.15
- ⚠️ RAPIDS **dask-xgboost** library is deprecated in v0.15

**METHOD** Conda 📥 Docker + Examples 🛠 Docker + Dev Env 🛠 Source ⚙

**RELEASE** Legacy (0.14) Stable (0.15) Nightly (0.16a)

**PACKAGES** All Packages cuDF cuML cuGraph cuSignal cuSpatial **cuxfilter**

**LINUX** Ubuntu 16.04 🐧 Ubuntu 18.04 🐧 CentOS 7 🐧 RHEL 7 🐧

**PYTHON** Python 3.6 (0.14 only) Python 3.7 Python 3.8 (0.15/0.16 only)

**CUDA** CUDA 10.0 (0.14 only) CUDA 10.1.2 CUDA 10.2 CUDA 11.0 (0.15/0.16 only)

**COMMAND** `conda install -c rapidsai -c nvidia -c conda-forge \ -c defaults rapids=0.15 python=3.7`

**NOTE:** Ubuntu 16.04/18.04 & CentOS 7 use the same `conda install` commands.

<https://rapids.ai/start.html>

# RAPIDS Docs

## Up to Date and Easy to Use

The image shows two screenshots of the cuDF documentation website.

**Left Screenshot:** The page title is "10 Minutes to cuDF and Dask-cuDF". The left sidebar has "cudf" selected. The main content includes a brief introduction, sections on "What are these Libraries?", "When to use cuDF and Dask-cuDF", and code snippets for object creation:

```
[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.
```

**Right Screenshot:** The page title is "Welcome to cuDF's documentation!". The left sidebar has "stable (0.13)" selected. The main content includes a "Contents" section with a hierarchical list of topics:

- 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>

# RAPIDS Docs

## Easier than Ever to Get Started with cuDF

The screenshot shows a web browser displaying the RAPIDS Docs website. The left sidebar contains a navigation menu with items like Home, cudf, clx, cugraph, cuml, cusignal, cuspatial, cufilter, libcudf, libcuml, libnvstrings, nvstrings, and rmm. The 'cudf' item is currently selected and highlighted in purple. The main content area has a header '10 Minutes to cuDF and Dask-cuDF'. Below the header, there's a brief introduction: 'Modeled after 10 Minutes to Pandas, this is a short introduction to cuDF and Dask-cuDF, geared mainly for new users.' It then defines 'cuDF' as a Python GPU DataFrame library and 'Dask' as a flexible library for parallel computing. 'Dask-cuDF' is described as extending Dask to support cuDF DataFrame partitions. A section titled 'When to use cuDF and Dask-cuDF' explains that cuDF is suitable for workflows on a single GPU or when data fits in memory. Below this, a code block shows Python imports for os, numpy, pandas, cuDF, and dask\_cudf, followed by a note about portions borrowed from cuDF cheatsheets. Another code block shows the creation of a cuDF Series object.

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

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

```
[2]: 0      1
```

<https://docs.rapids.ai>

# Explore: RAPIDS Code and Blogs

## Check out our Code and How We Use It

README.md

### RAPIDS cuDF - GPU DataFrames

build running

NOTE: For the latest stable README.md ensure you are on the master branch.

Built based on the Apache Arrow columnar memory format, cuDF is a GPU DataFrame library for loading, joining, aggregating, filtering, and otherwise manipulating data.

cuDF provides a pandas-like API that will be familiar to data engineers & data scientists, so they can use it to easily accelerate their workflows without going into the details of CUDA programming.

For example, the following snippet downloads a CSV, then uses the GPU to parse it into rows and columns and run calculations:

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

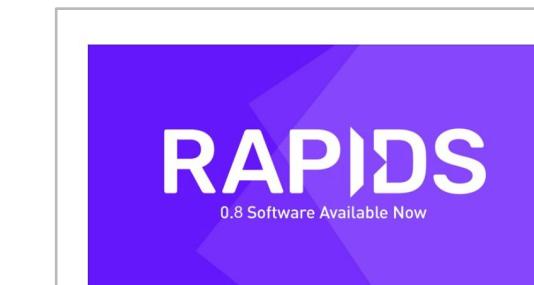
url="https://github.com/plotly/datasets/raw/master/tips.csv"
content = requests.get(url).content.decode('utf-8')

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>



**RAPIDS**  
0.8 Software Available Now



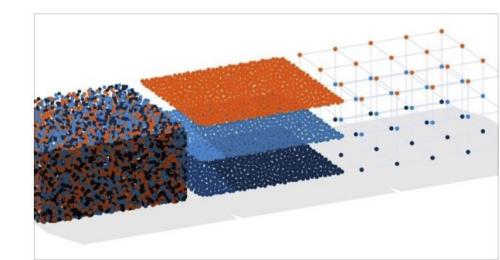
**RAPIDS Release 0.8: Same Community New Freedoms**  
Making more friends and building more bridges to more ecosystems. It's now easier than ever to get started with RAPIDS.

 Josh Patterson  
Jul 19 · 7 min read



**gQuant—GPU Accelerated examples for Quantitative Analyst Tasks**  
A simple trading strategy backtest for 5000 stocks using GPUs and getting 20X speedup

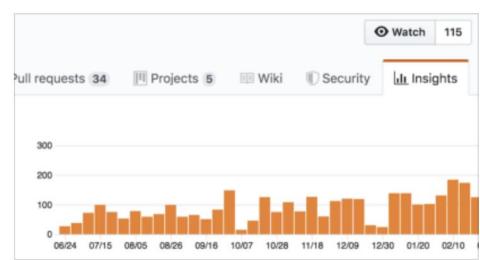
 Yi Dong  
Jul 16 · 6 min read ★



**NVIDIA GPUs and Apache**

**When Less is More: A brief**

**Nightly News: CI produces**



Pull requests 34 Projects 5 Wiki Security Insights  
Watch 115

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

# Explore: RAPIDS Github

The screenshot shows the GitHub profile for the RAPIDS organization. The top navigation bar includes links for Pull requests, Issues, Marketplace, and Explore. The main header features the RAPIDS logo and the text "Open GPU Data Science". Below the header, there are links for Packages, People (118), Teams (91), and Projects (6). The "Repositories" tab is selected, showing 67 repositories. A section titled "Pinned repositories" displays six repositories:

- cudf**: cuDF - GPU DataFrame Library. Cuda, 1.9k stars, 270 forks.
- cuml**: cuML - RAPIDS Machine Learning Library. C++ 665 stars, 119 forks.
- cugraph**: cuGraph - RAPIDS Graph Analytics Library. Cuda, 204 stars, 52 forks.
- notebooks**: RAPIDS Sample Notebooks. Jupyter Notebook, 204 stars, 94 forks.
- notebooks-contrib**: RAPIDS Community Notebooks. Jupyter Notebook, 106 stars, 76 forks.
- cuxfilter**: GPU accelerated cross filtering. Python, 31 stars, 14 forks.

<https://github.com/rapidsai>

# Explore: Notebooks Contrib

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. The channel has 47 subscribers. The navigation bar includes HOME (which is selected), VIDEOS, PLAYLISTS, CHANNELS, DISCUSSION, and ABOUT. Below the navigation, there is a section titled 'Uploads' with a 'PLAY ALL' button. Four video thumbnails are listed:

- How to Accelerate XGBoost on NVIDIA GPUS (8:47)
- Introduction to Data Science With RAPIDS on NVIDIA GPUS (10:48)
- Deploying Distributed XGBoost At Scale on NVIDIA GPUS (12:51)
- A thumbnail for a video titled 'Accelerated with RAF Walmart uses RAPIDS' showing a Walmart delivery truck.

Below each video thumbnail, the title, duration, and view count are displayed. The first three videos were uploaded 1 month ago, while the fourth one was uploaded 4 months ago.

Notebooks Contrib Repo has tutorials and examples, and various E2E demos.  
RAPIDS Youtube channel has explanations, code walkthroughs and use cases.

# RAPIDS

## How Do I Get the Software?



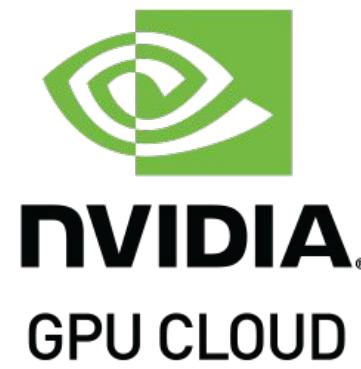
GITHUB

<https://github.com/rapidsai>



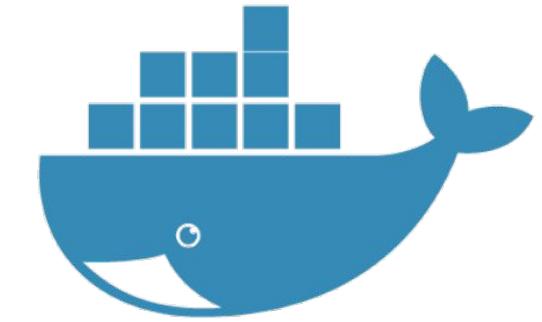
ANACONDA

<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 Machine  
Learning



Cloud  
Dataproc



Google Cloud

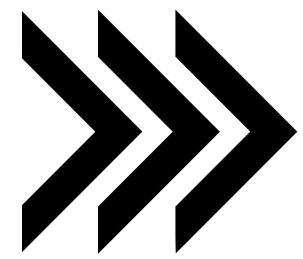


Kubeflow

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

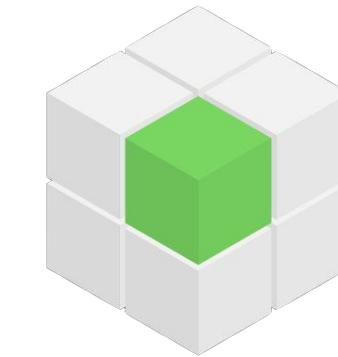
RAPIDS

<https://rapids.ai>  
@RAPIDSAl



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

Joshua Patterson  
joshuap@nvidia.com

 @datametrician

# RAPIDS