

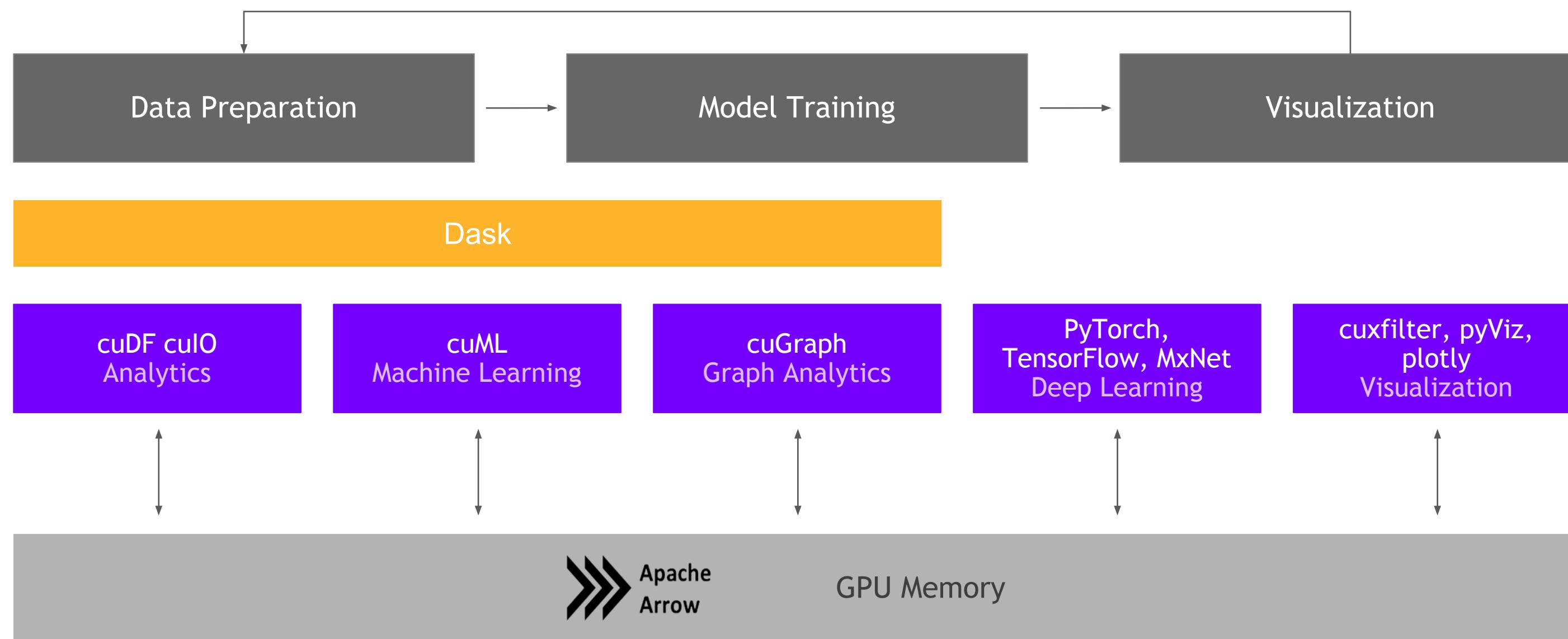
# RAPIDS

## The Platform Inside and Out Release 0.14

Joshua Patterson - Senior Director, RAPIDS Engineering

# RAPIDS

## End-to-End Accelerated GPU Data Science



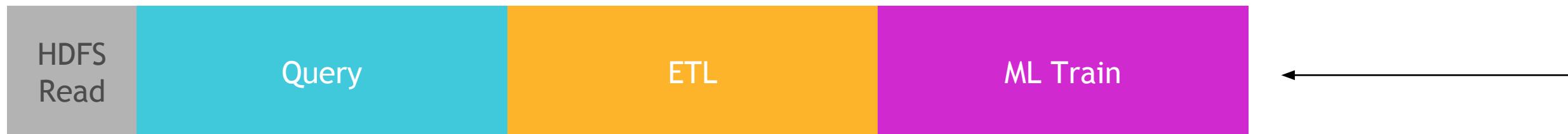
# Data Processing Evolution

## Faster Data Access, Less Data Movement

Hadoop Processing, Reading from Disk



Spark In-Memory Processing



Traditional GPU Processing

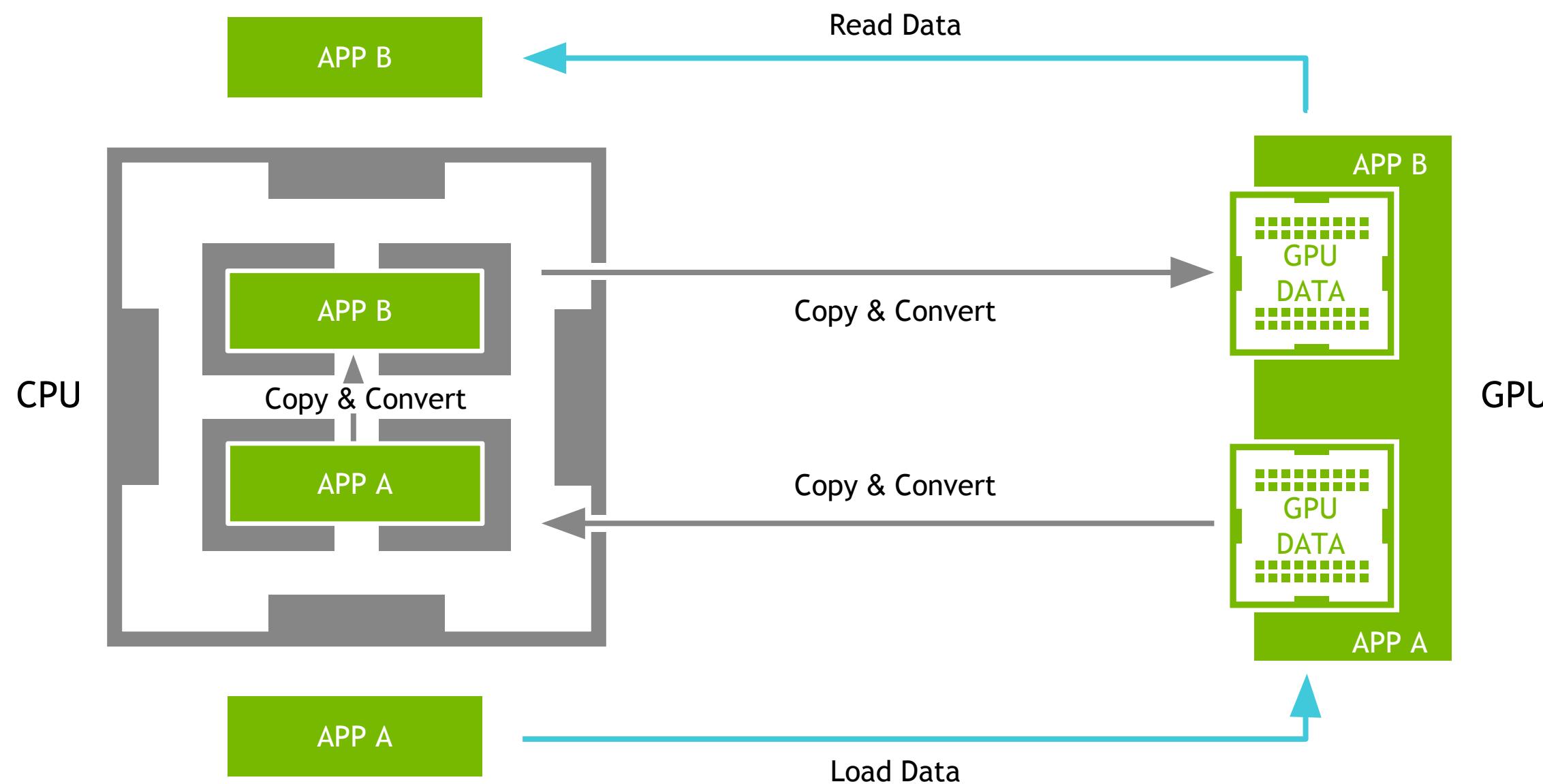


**25-100x Improvement**  
Less Code  
Language Flexible  
Primarily In-Memory

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

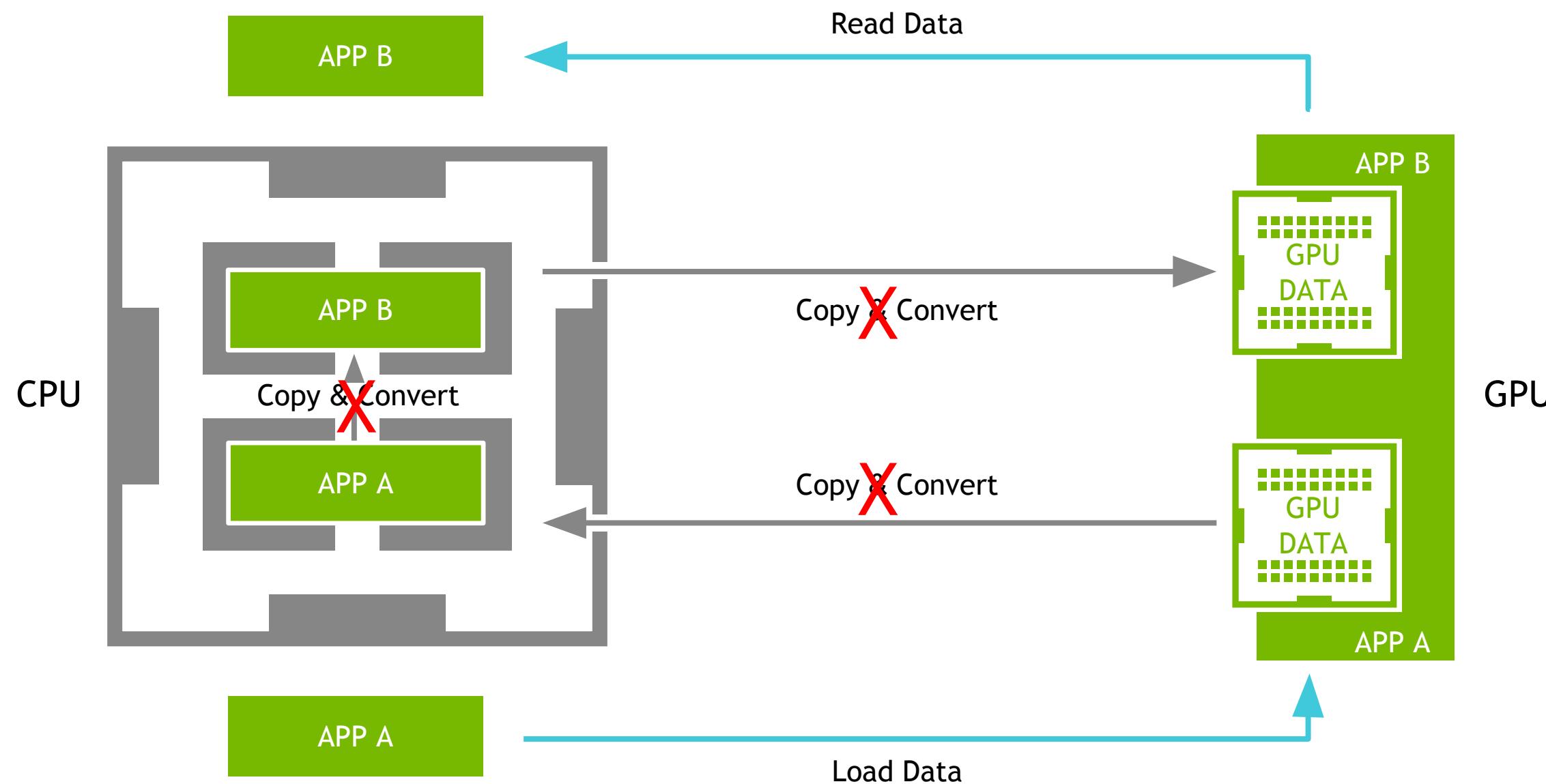
# Data Movement and Transformation

The Bane of Productivity and Performance

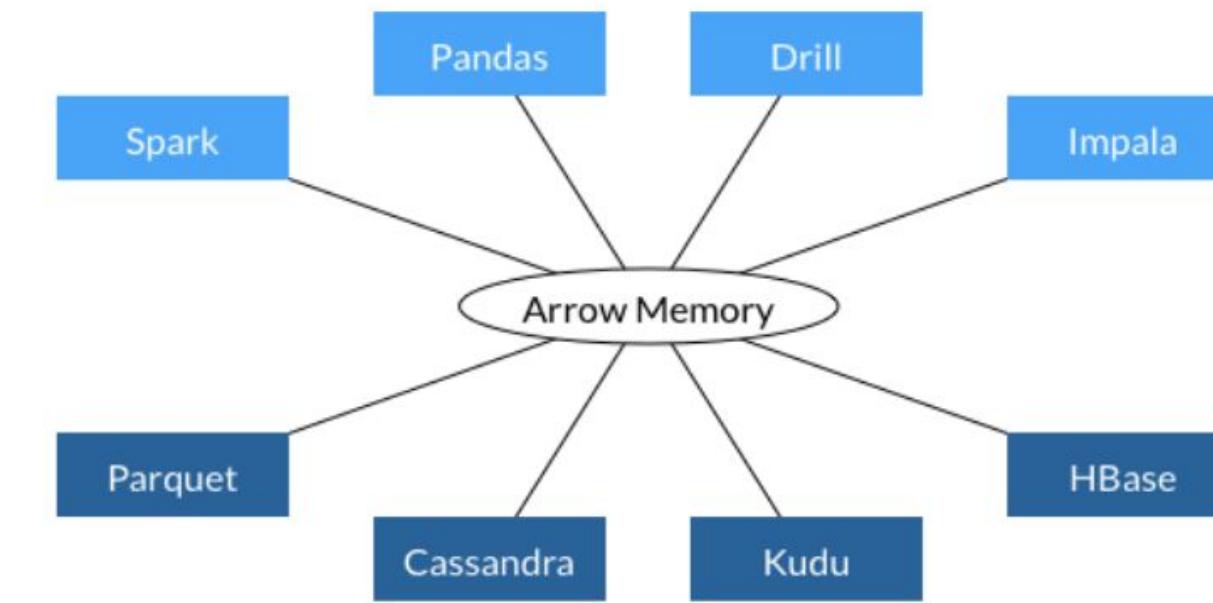
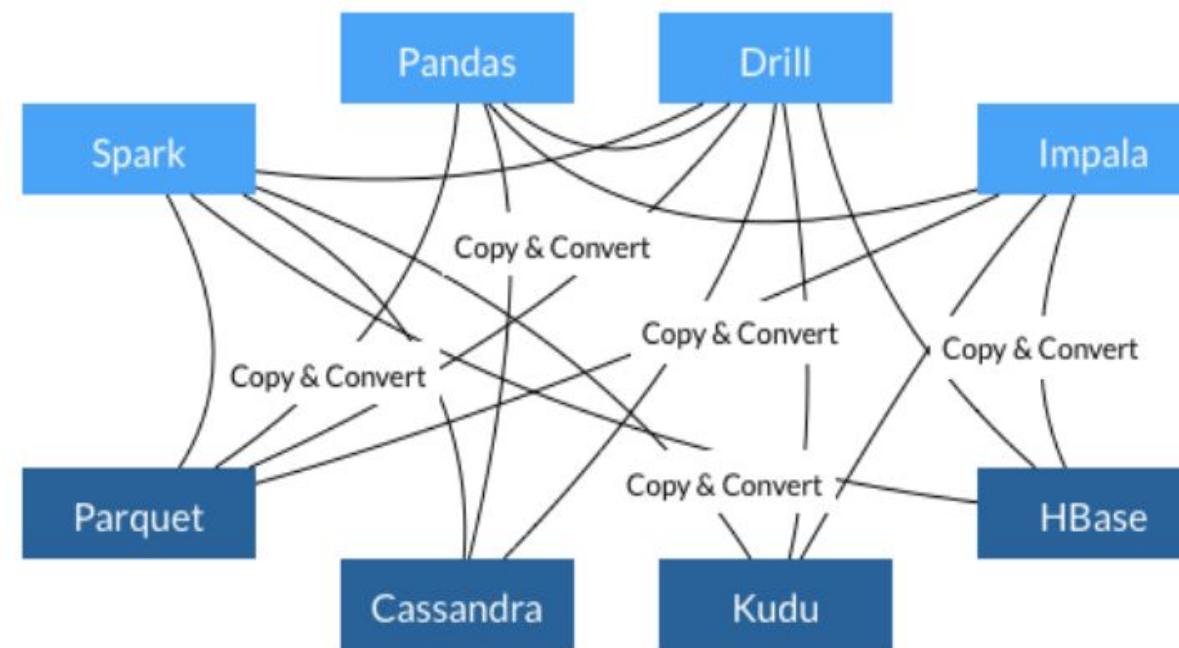


# Data Movement and Transformation

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/>

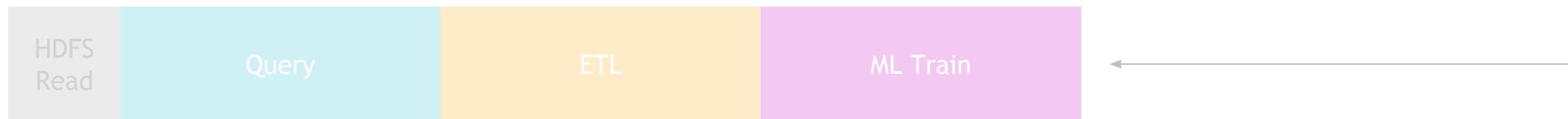
# Data Processing Evolution

## Faster Data Access, Less Data Movement

Hadoop Processing, Reading from Disk



Spark In-Memory Processing



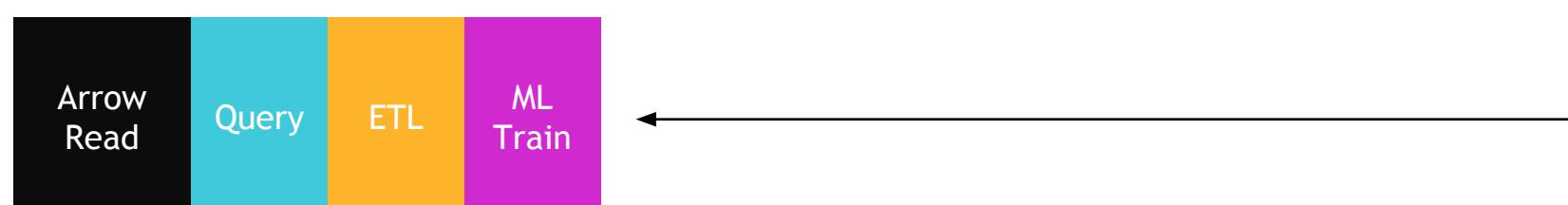
25-100x Improvement  
Less Code  
Language Flexible  
Primarily In-Memory

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

Up to 350x faster queries; Hours to Seconds!

TPCx-BB is a data science benchmark consisting of 30 analytics queries representing real-world ETL and Machine Learning workflows. It can be run at multiple “Scale Factors”.

SF1 - 1GB

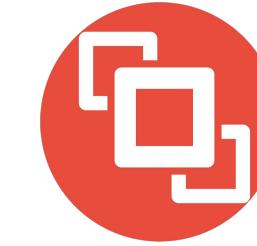
SF1K - 1 TB

SF10K - 10 TB

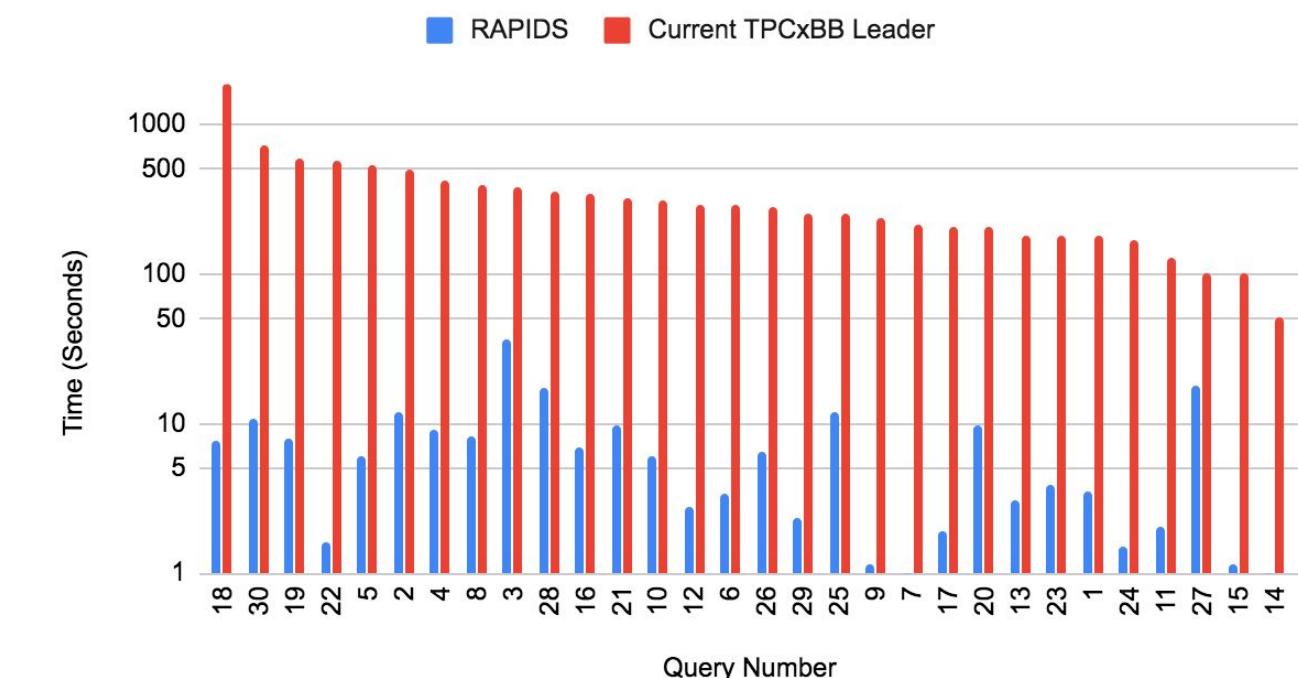
RAPIDS preliminary results at SF1K (Single DGX-2) and SF10K (17x DGX-1) show GPUs provide dramatic cost and time-savings for small scale *and* large-scale data analytics problems

SF1K Avg: 51x speed-up (>40x Normalized for Cost)

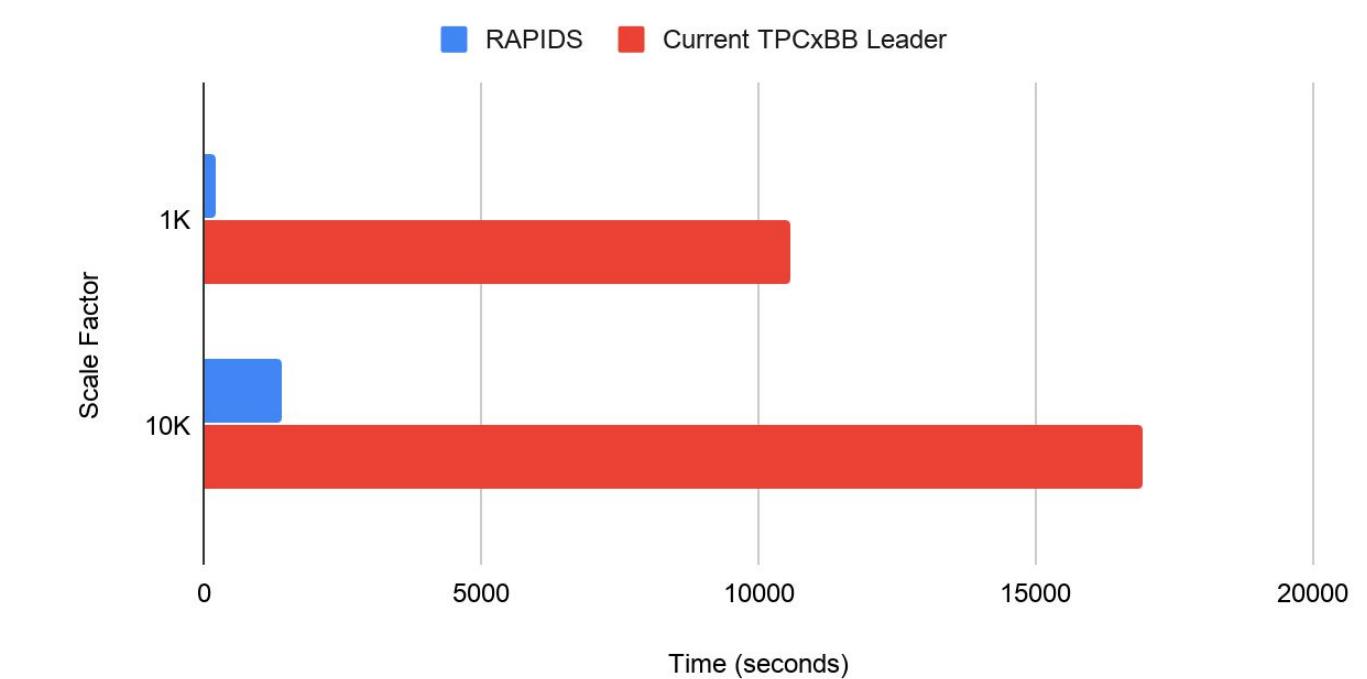
SF10K Avg: 12x speed-up (>6x Normalized for Cost)



SF1K Speedup with RAPIDS



TPCxBB 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.

Idea  
Visualization & understanding  
Results  
Experiment  
Software tools  
Infrastructure

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. 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. Using Keras means you're more likely to win, and inversely, those who practice the sort of fast experimentation strategy that sets them up to win are more likely to prefer 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).

Framework	Count (approx.)
Sci-kit Learn	80
Keras	65
LightGBM	55
XGBoost	55
PyTorch	30
TensorFlow (non-Keras)	25
Caffe	5
MXNet	5
Fastai	5
Caffe2	5
CatBoost	5
R Random Forest	5

kaggle

# RAPIDS 0.14 Release Summary

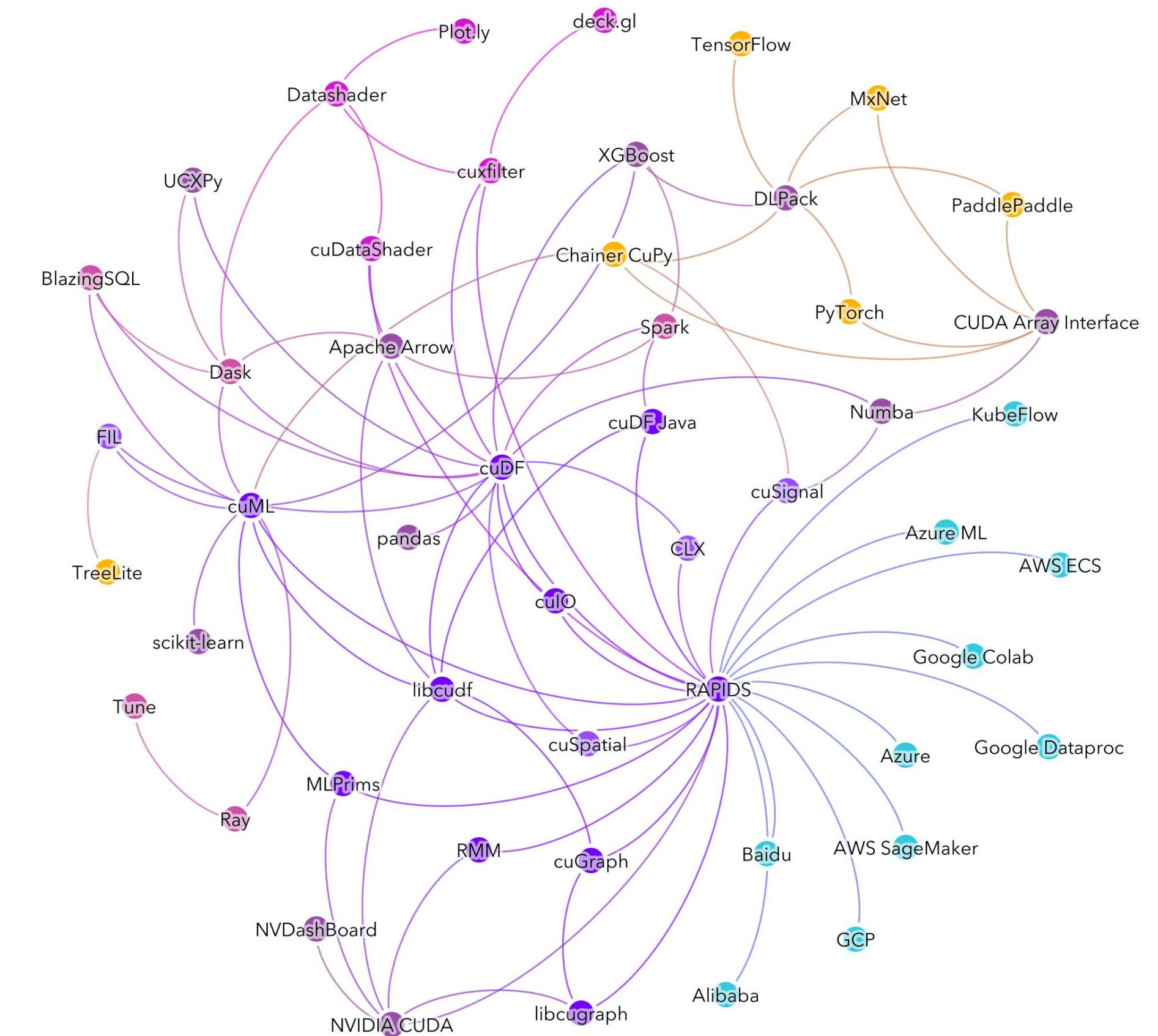
# What's New in Release 0.14?

- cuDF drops libcudf legacy code, greatly improves API documentation, and adds bug fixes as well as performance and memory usage optimizations
- cuML machine learning library adds support for flexible data formats (inputs from Pandas, NumPy, cuDF or GPU) for all models, weighted K-means, and MNMG coordinate descent
- cuGraph major improvement to the Louvain algorithm, new Betweenness Centrality, added Force Atlas 2, finished the big refactoring effort
- UCX-Py add InfiniBand support as well as Multi-Node Multi-GPU NVLink/InfiniBand tests
- BlazingSQL now supports out-of-core query execution, which enables queries to operate on datasets dramatically larger than available GPU memory
- cuSignal major refactoring and improvements to documentation and sample notebooks
- cuSpatial adds fast quadtree building and major documentation and API improvements
- cuXfilter adds dask cuDF support, groupby optimizations, chart autoscaling and other general improvements
- RMM adds fast logging of allocations/deallocations and an uninitialized vector container

# RAPIDS Everywhere

## The Next Phase of RAPIDS

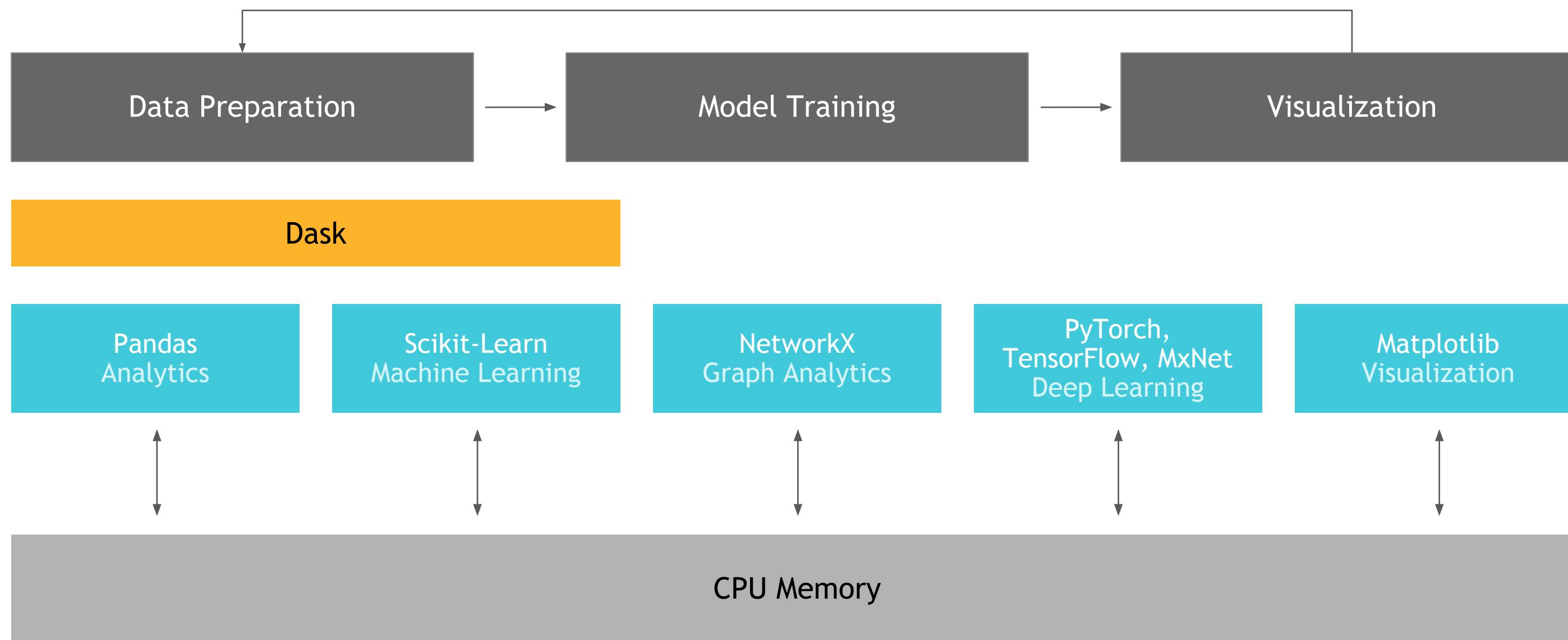
Exactly as it sounds—our goal is to make RAPIDS as usable and performant as possible wherever data science is done. We will continue to work with more open source projects to further democratize acceleration and efficiency in data science.



# RAPIDS Core

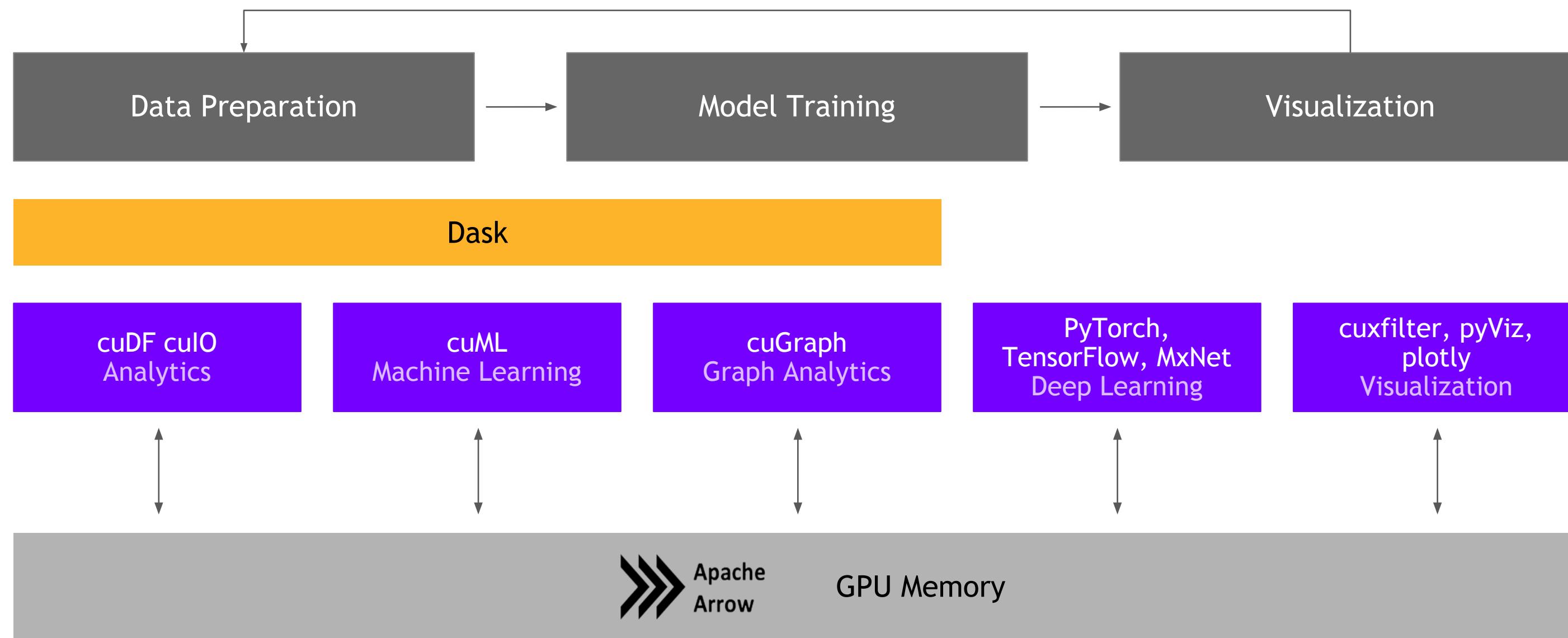
# Open Source Data Science Ecosystem

Familiar Python APIs



# RAPIDS

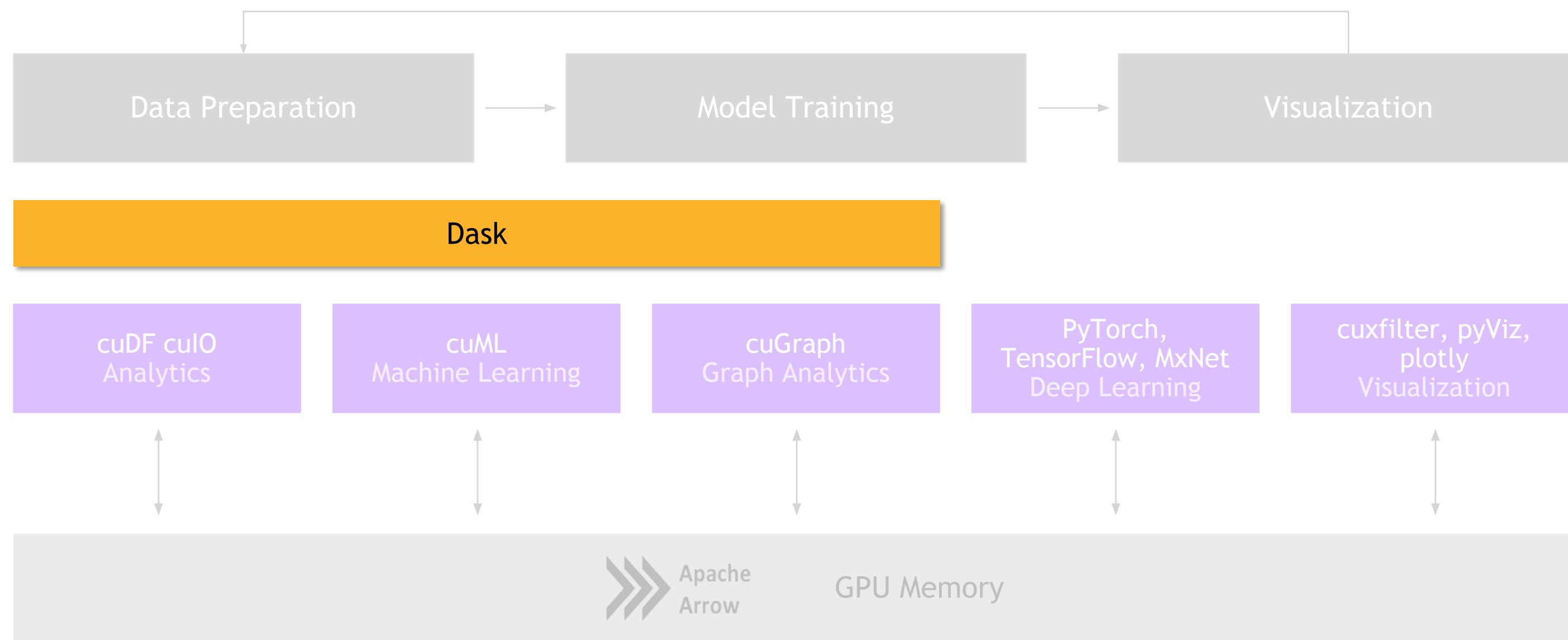
## End-to-End Accelerated GPU Data Science



# 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



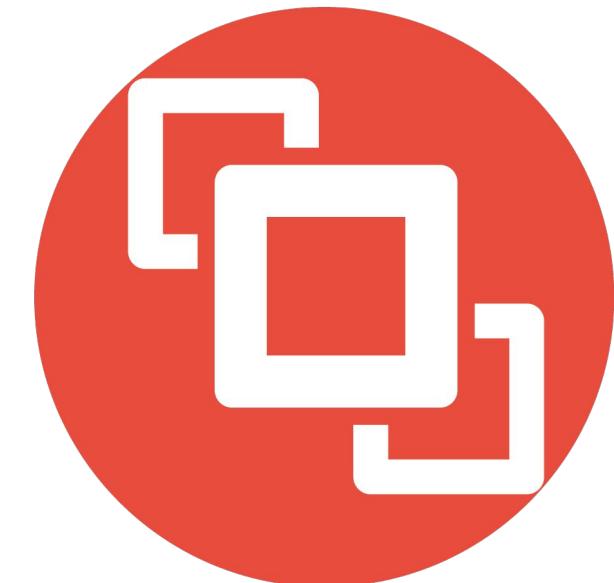
# Why OpenUCX?

## Bringing Hardware Accelerated Communications to Dask

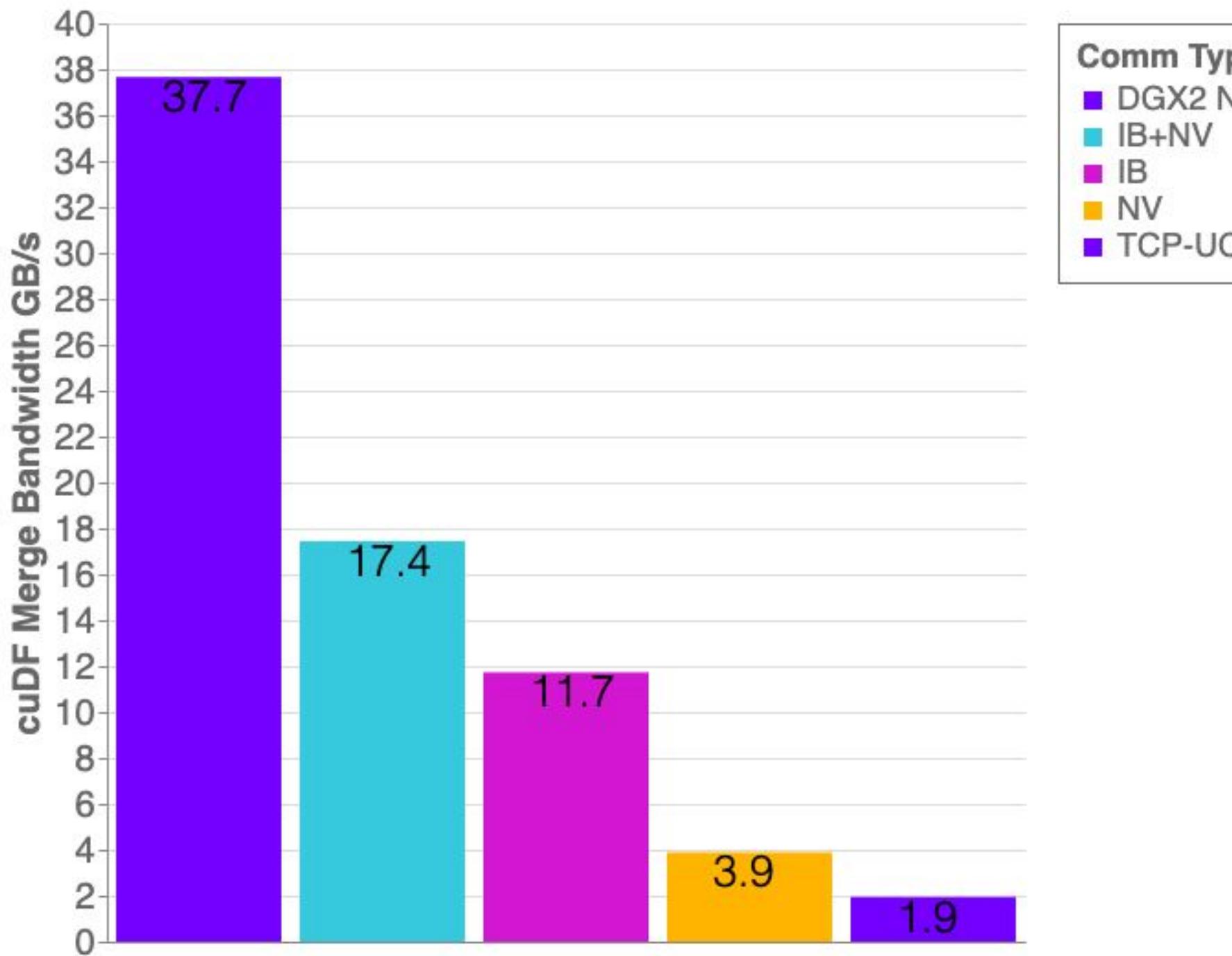
- TCP sockets are slow!
- UCX provides uniform access to transports (TCP, InfiniBand, shared memory, NVLink)
- Alpha Python bindings for UCX (ucx-py)
- Will provide best communication performance, to Dask based on available hardware on nodes/cluster

---

```
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

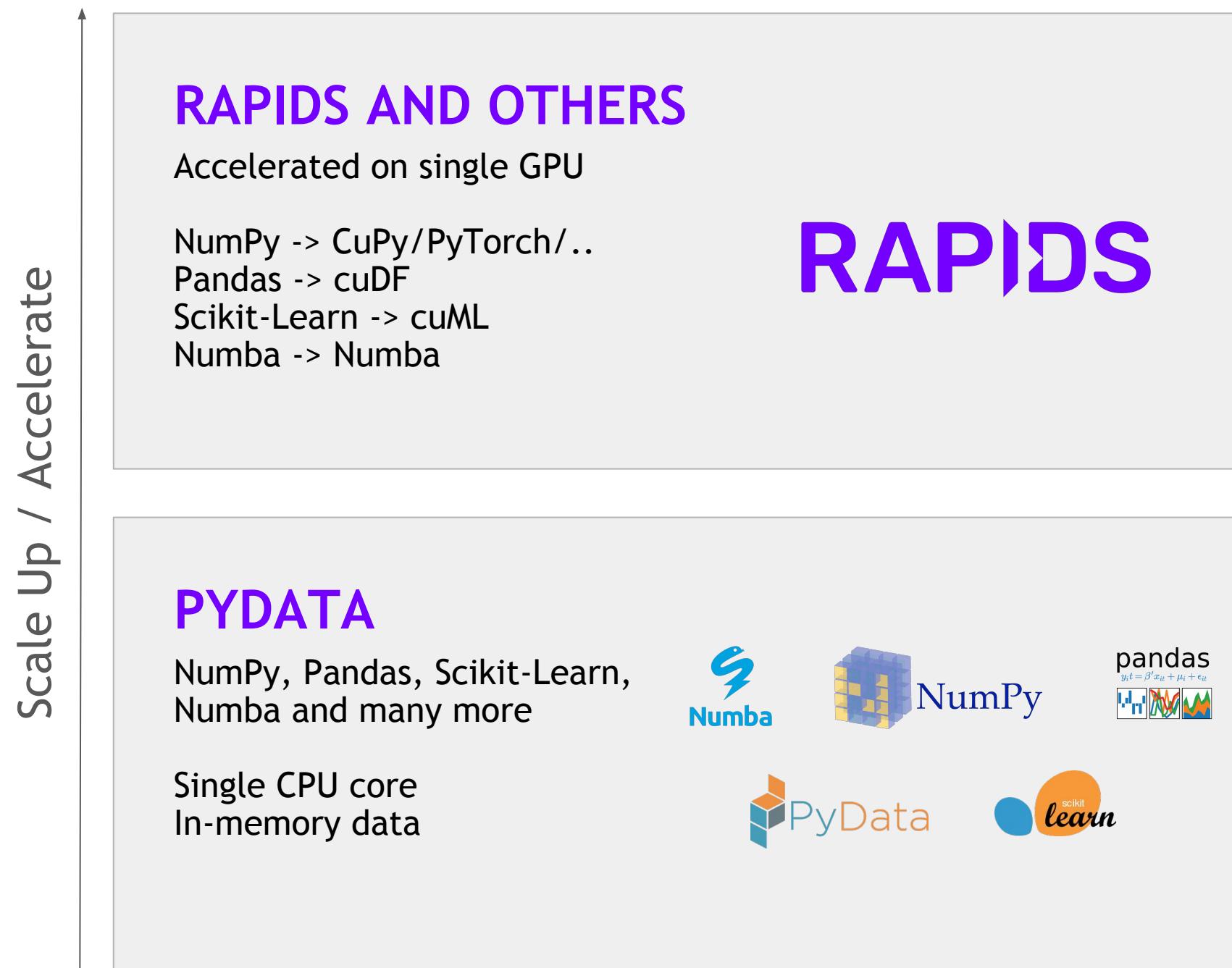
□ Benchmark Setup:

- DataFrames: Left/Right 1x int64 column key column, 1x int64 value columns

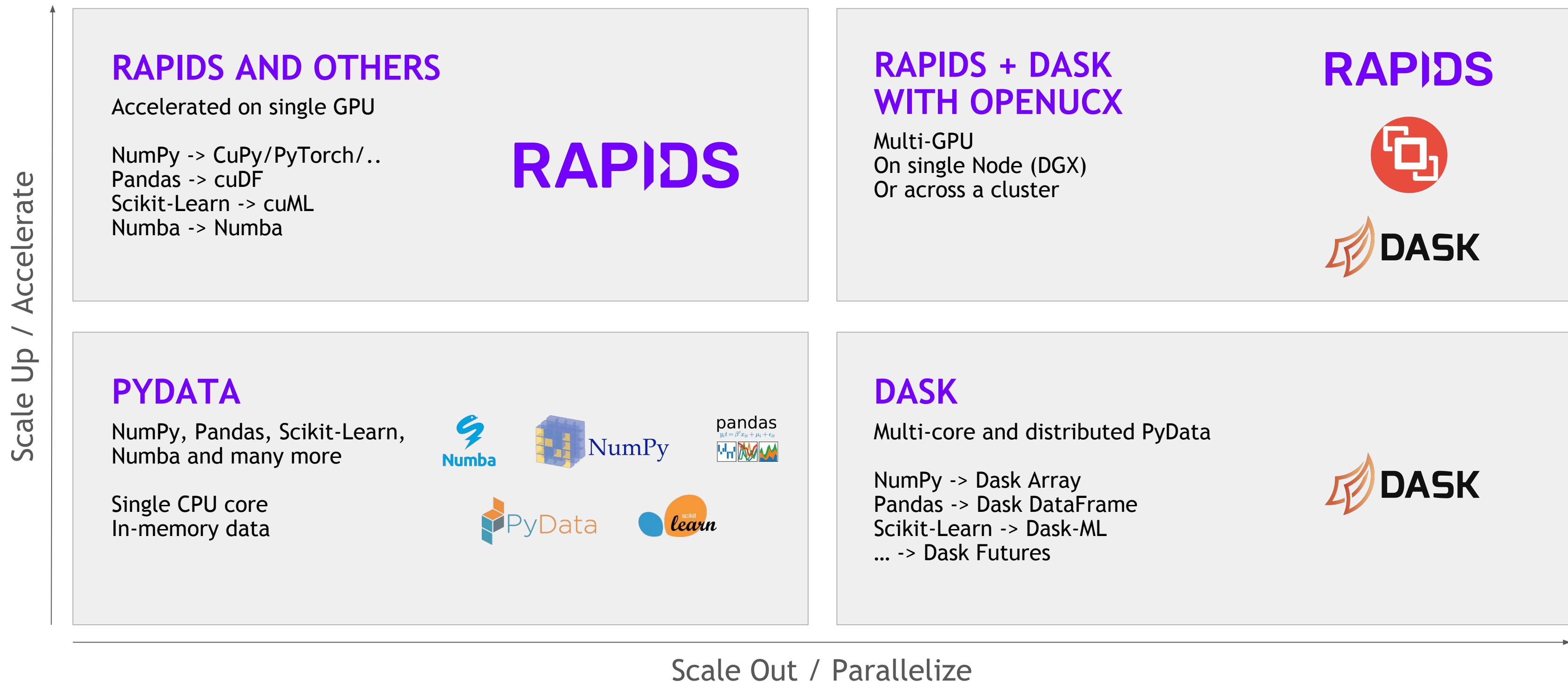
- Merge: Inner

- 30% of matching data balanced across each partition

# Scale Up with RAPIDS



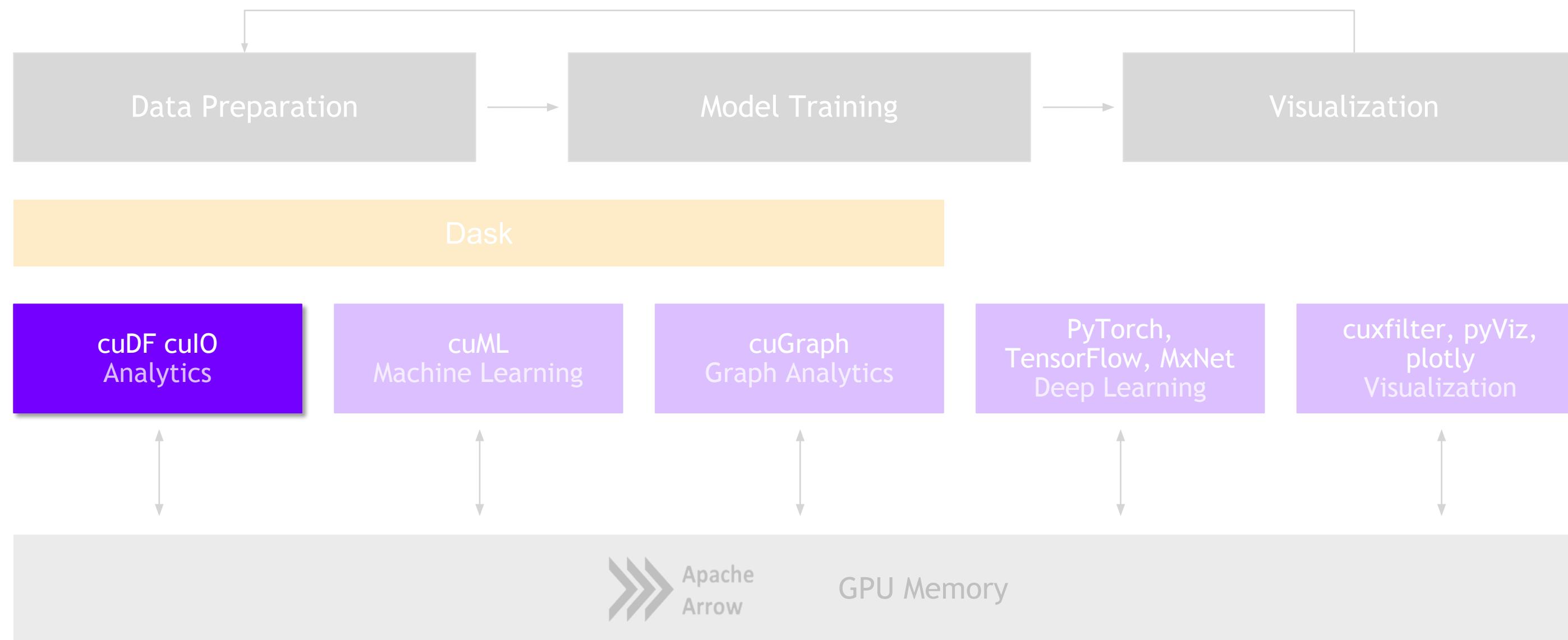
# Scale Out with RAPIDS + Dask with OpenUCX



cuDF

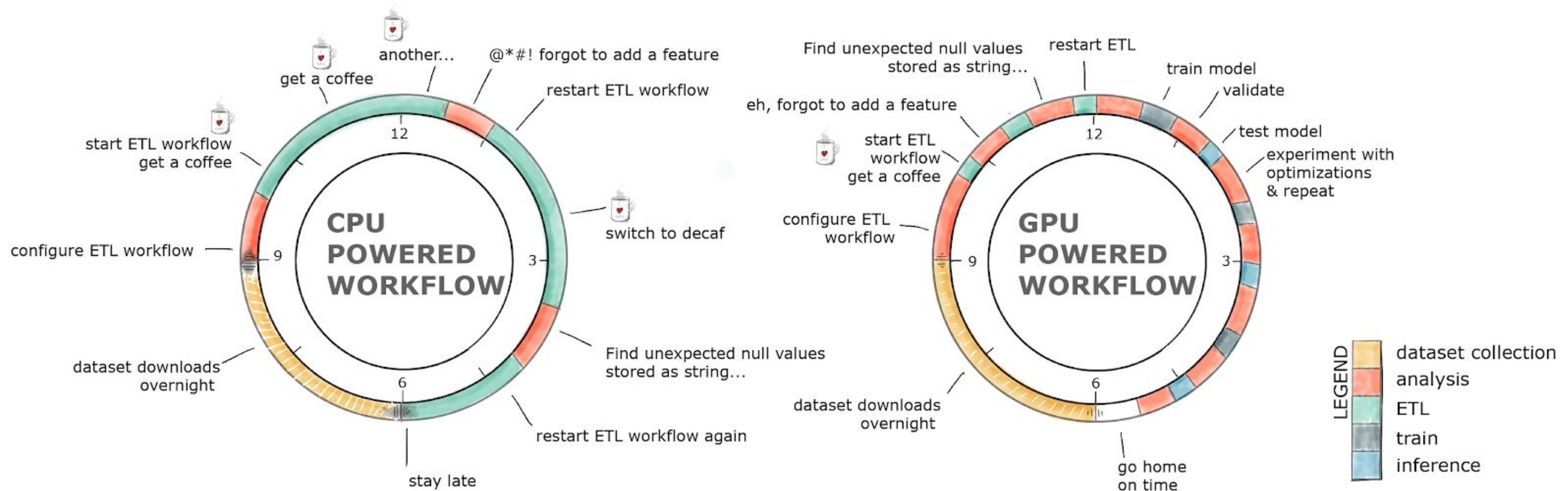
# RAPIDS

## GPU Accelerated Data Wrangling and Feature Engineering

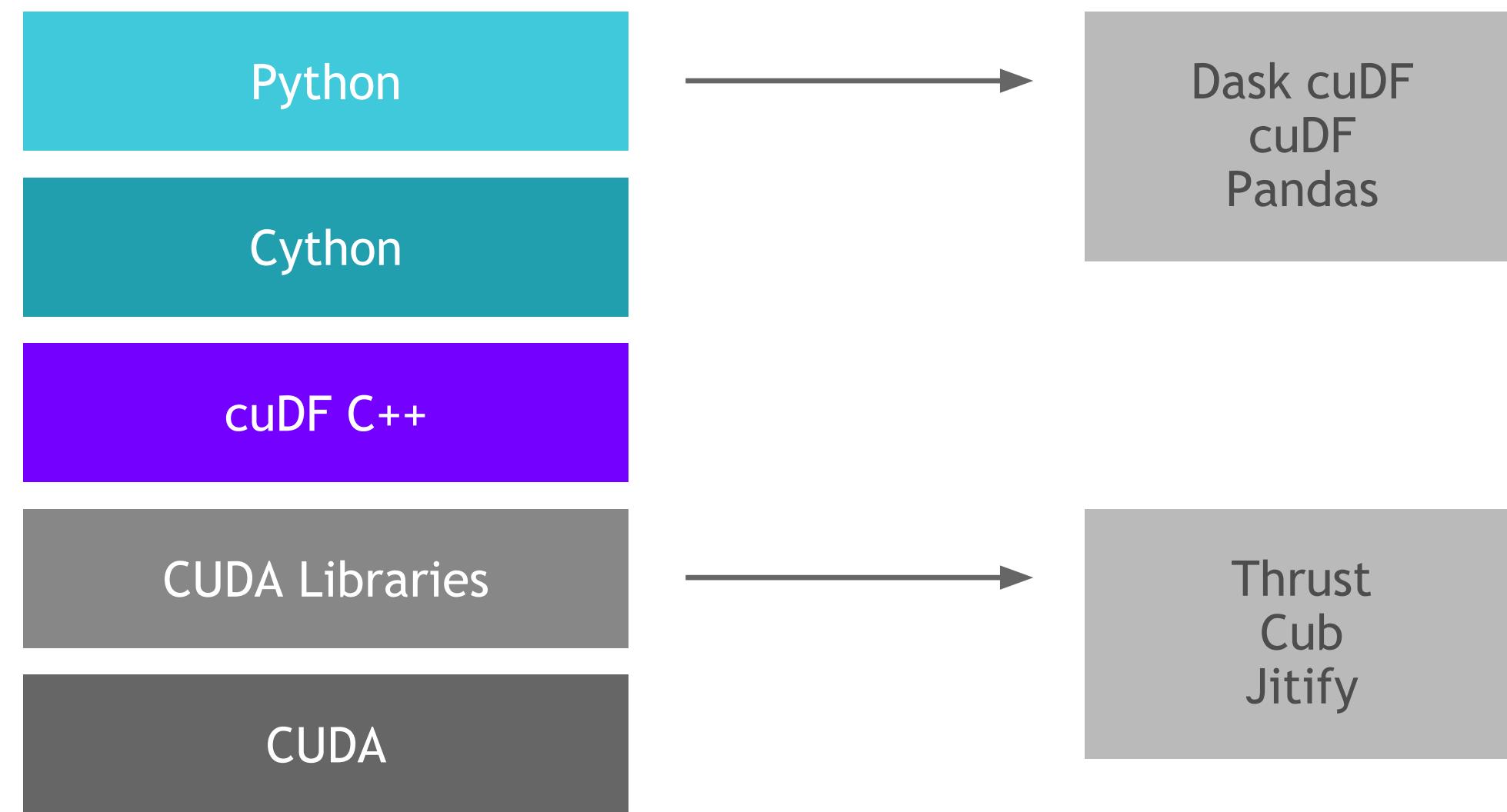


# GPU-Accelerated ETL

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]:
```

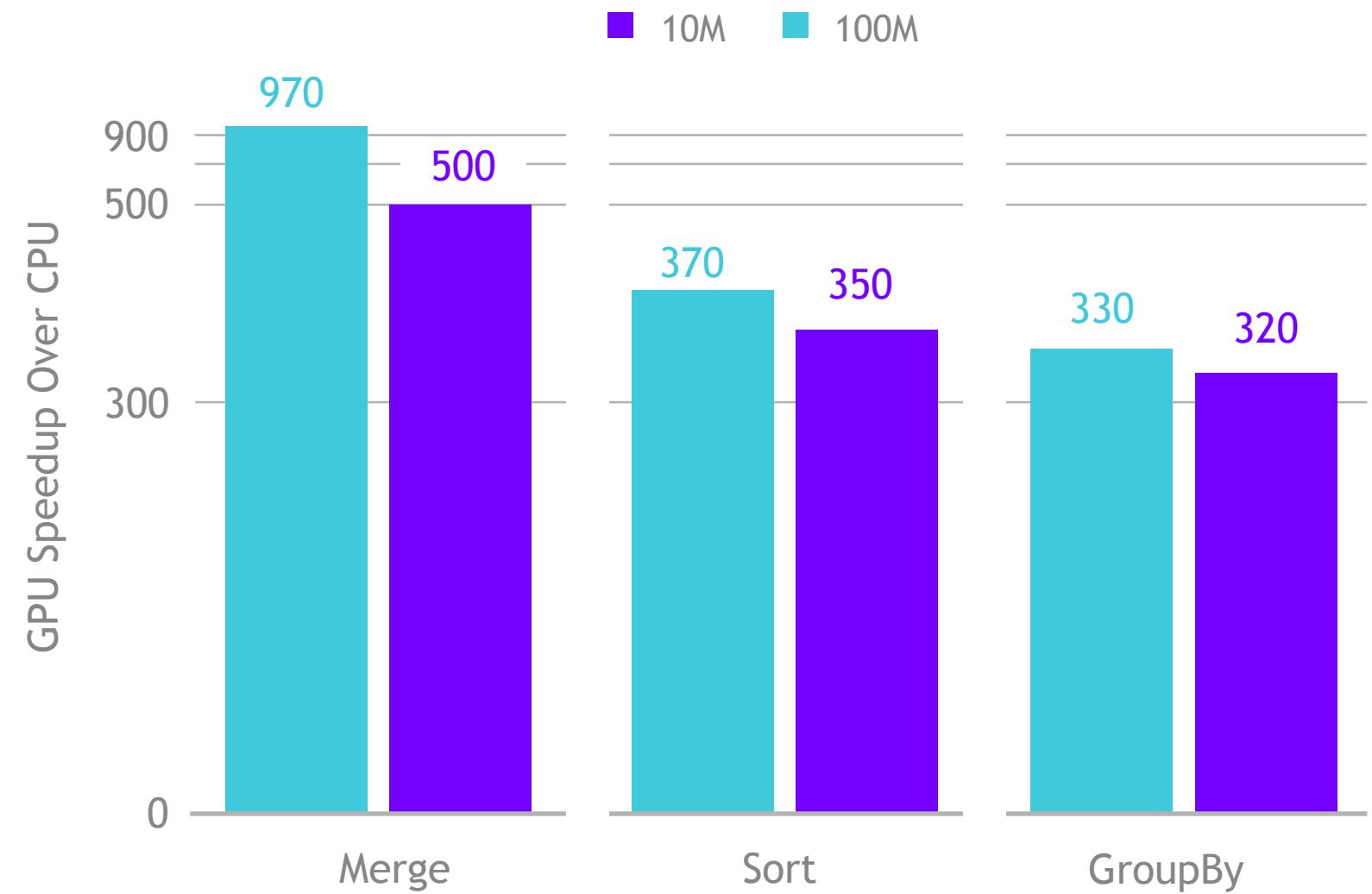
	User_ID	Product_ID	Gender	Age	Occupation	City_Category	Stay_In_Current_City_Years	Marital_Status	Product_Ca
0	1000001	P00069042	F	0-17	10	A	2	0	3
1	1000001	P00248942	F	0-17	10	A	2	0	1
2	1000001	P00087842	F	0-17	10	A	2	0	12
3	1000001	P00085442	F	0-17	10	A	2	0	12
4	1000002	P00285442	M	55+	16	C	4+	0	8

```
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

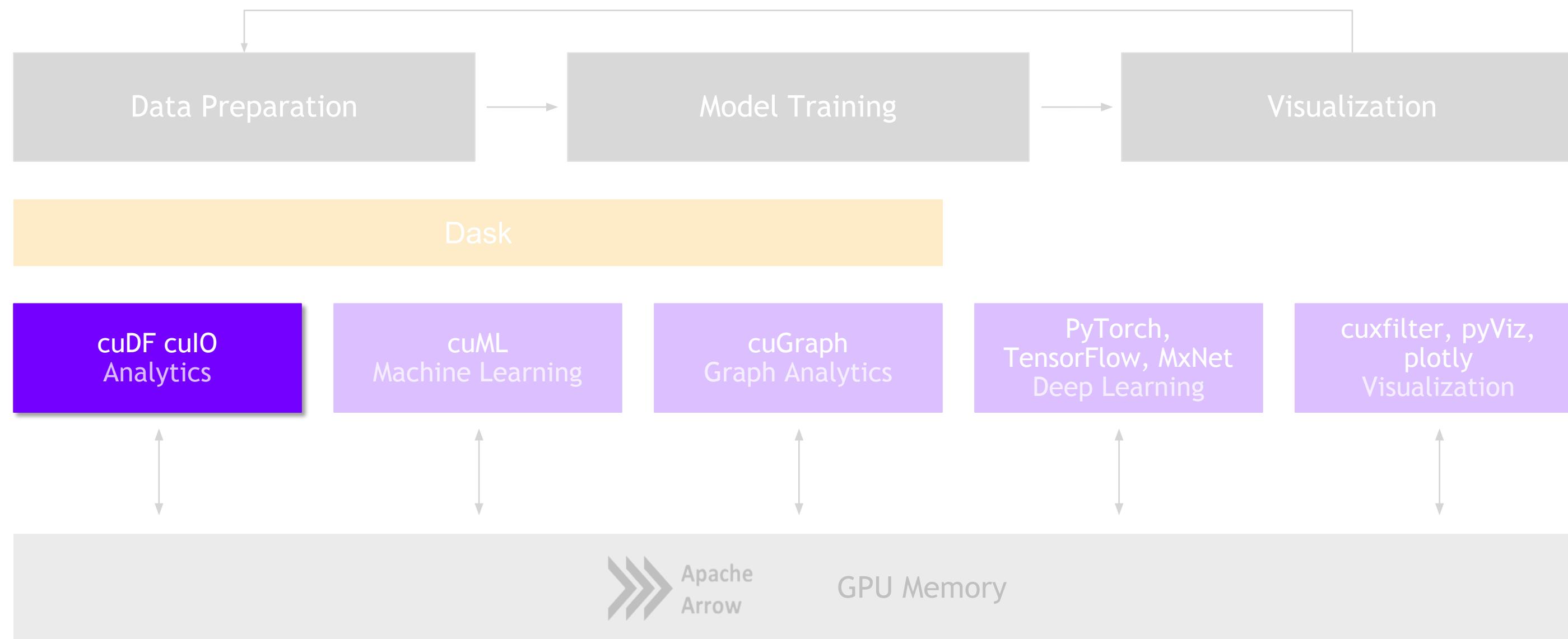
cuDF v0.13, Pandas 0.25.3

- 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

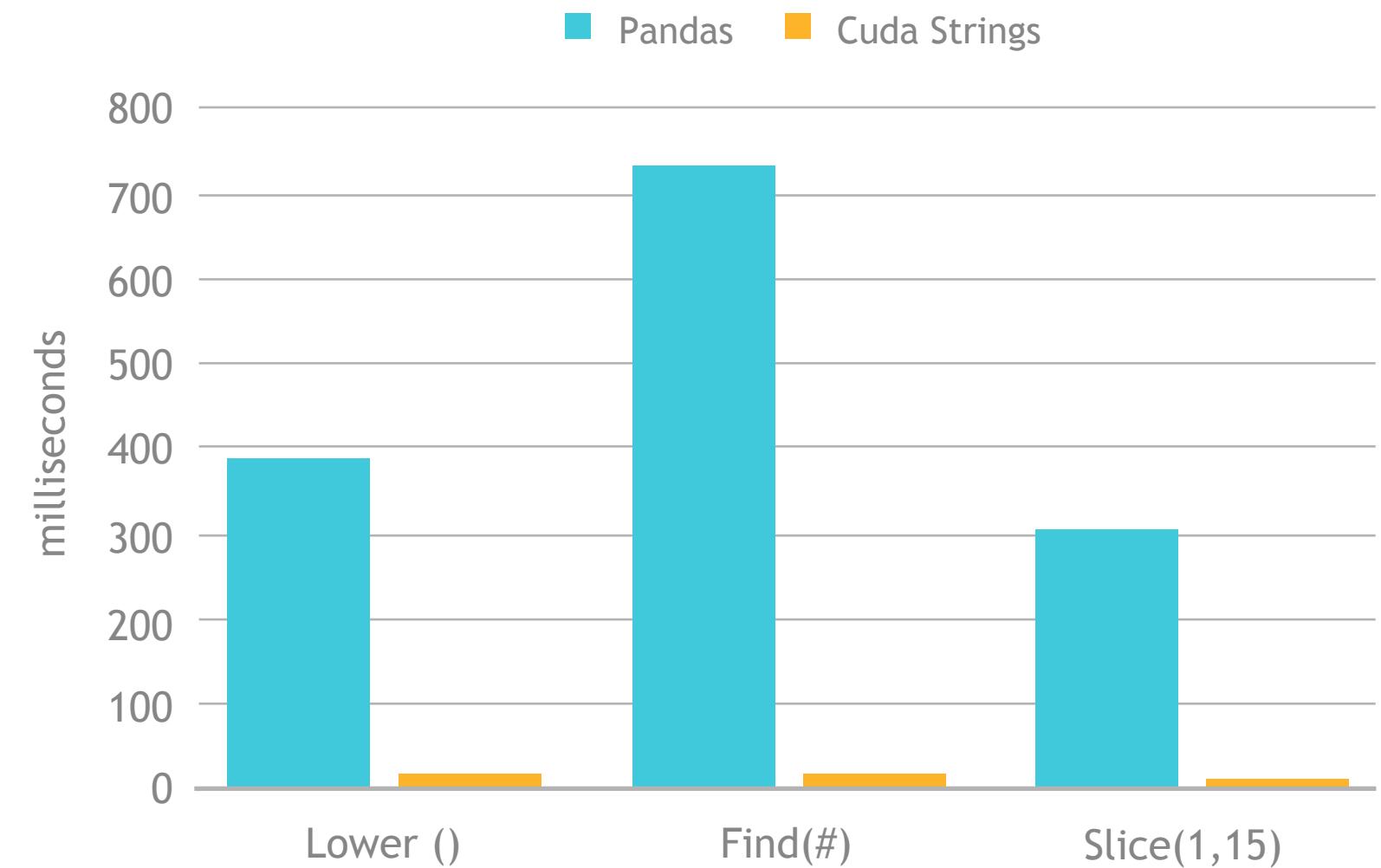
## String Support

### CURRENT V0.14 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++

### FUTURE V0.15+ 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 - v0.2, CSV Writer v0.8
- Parquet Reader - v0.7, Parquet Writer v0.12
- ORC Reader - v0.7, ORC Writer v0.10
- JSON Reader - v0.8
- Avro Reader - v0.9
- GPU Direct Storage integration in progress for bypassing PCIe bottlenecks!
- Key is GPU-accelerating both parsing and decompression

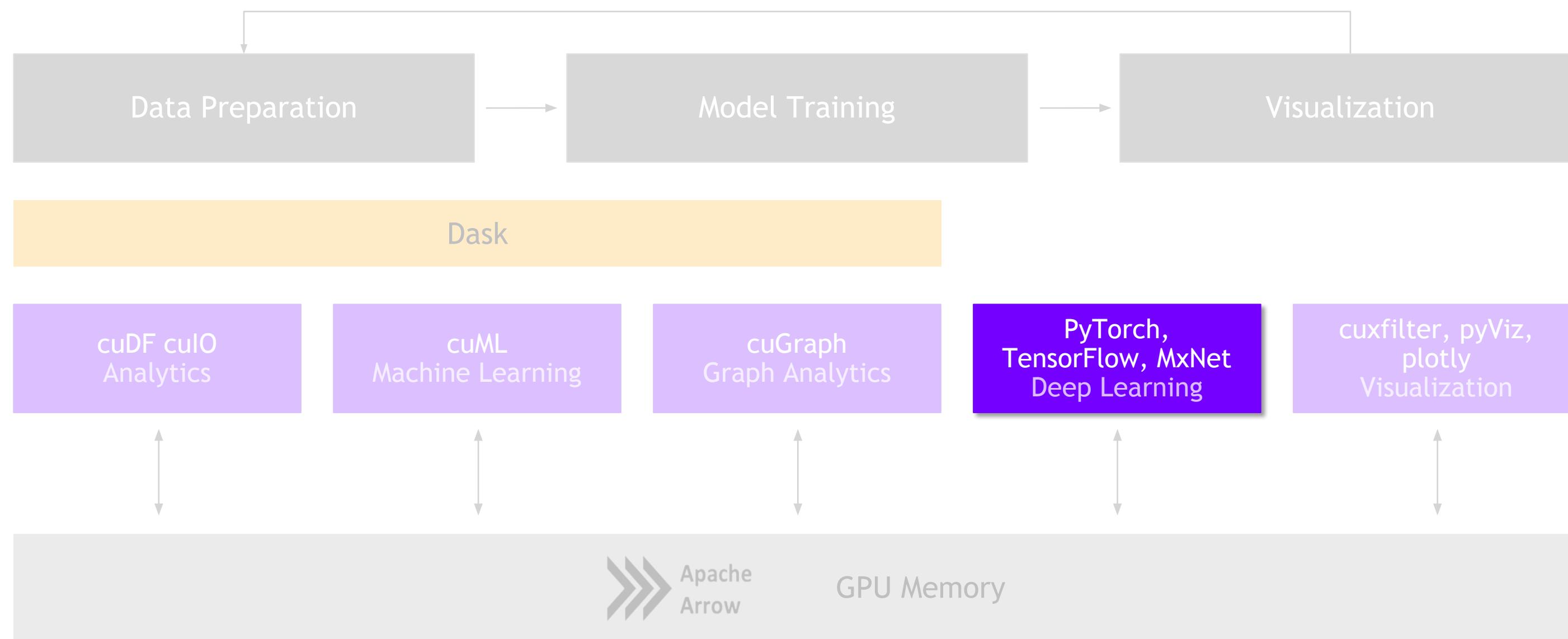
```
1]: import pandas, cudf
2]: %time len(pandas.read_csv('data/nyc/yellow_tripdata_2015-01.csv'))
CPU times: user 25.9 s, sys: 3.26 s, total: 29.2 s
Wall time: 29.2 s
2]: 12748986
3]: %time len(cudf.read_csv('data/nyc/yellow_tripdata_2015-01.csv'))
CPU times: user 1.59 s, sys: 372 ms, total: 1.96 s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

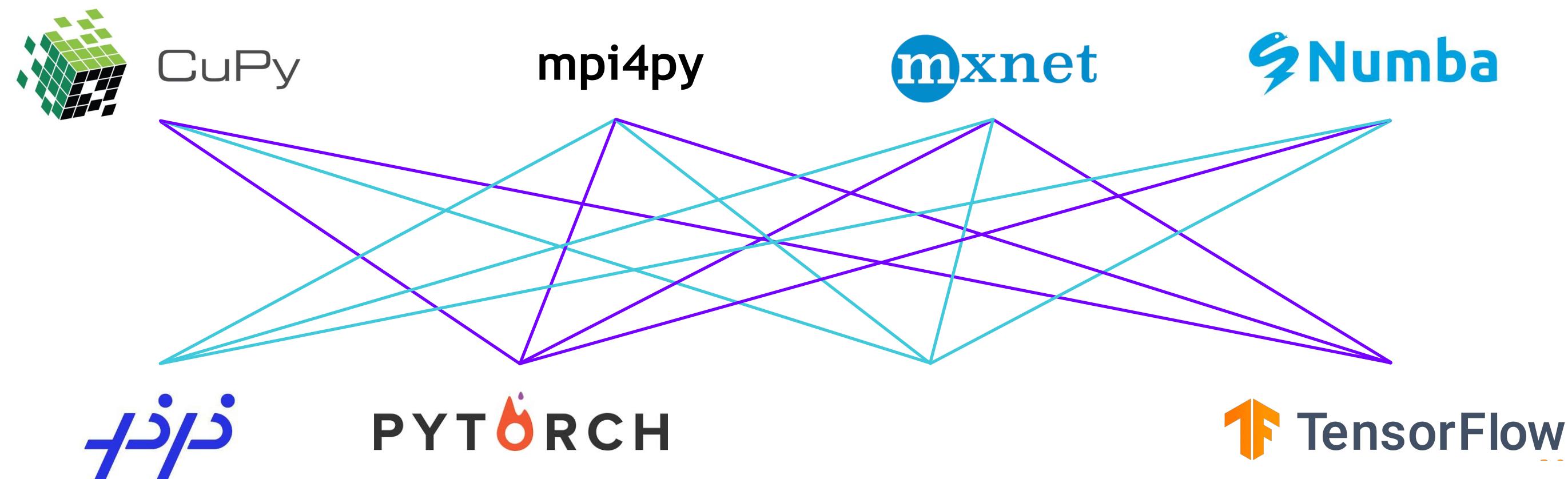
# RAPIDS

## Building Bridges into the Array Ecosystem



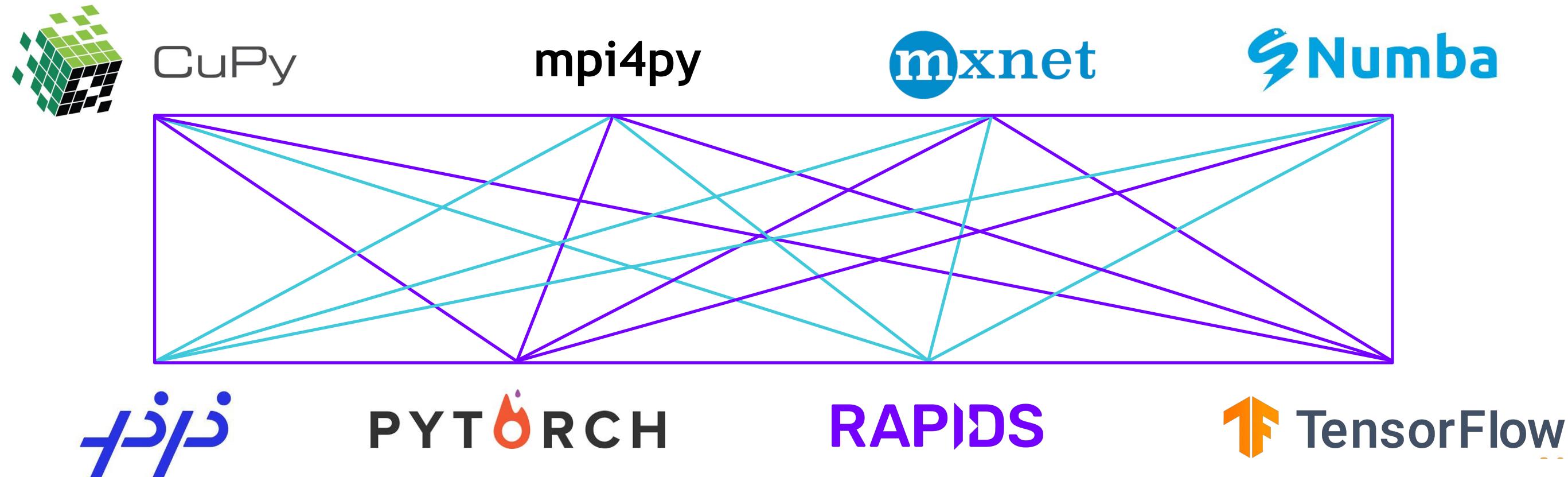
# Interoperability for the Win

DLPack and `__cuda_array_interface__`



# Interoperability for the Win

DLPack and `__cuda_array_interface__`



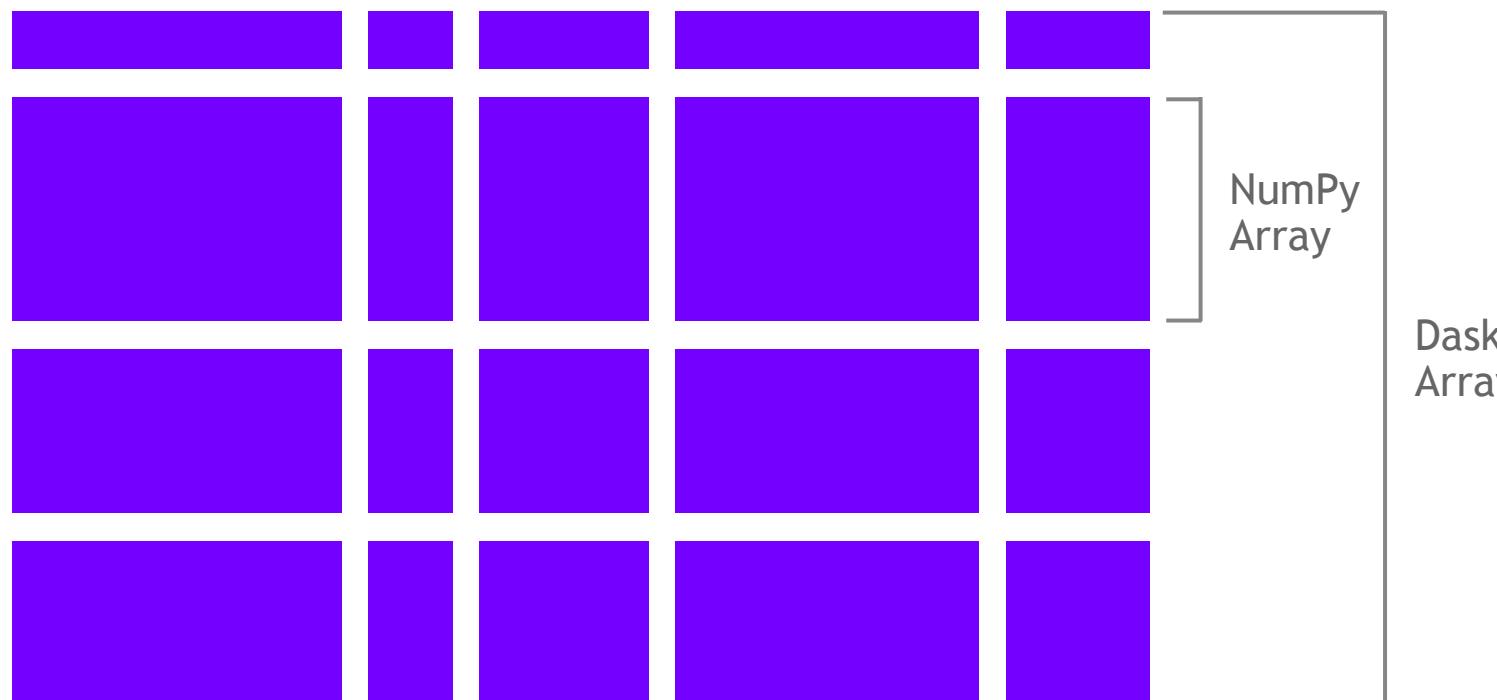
# ETL - Arrays and DataFrames

## Dask and CUDA Python Arrays



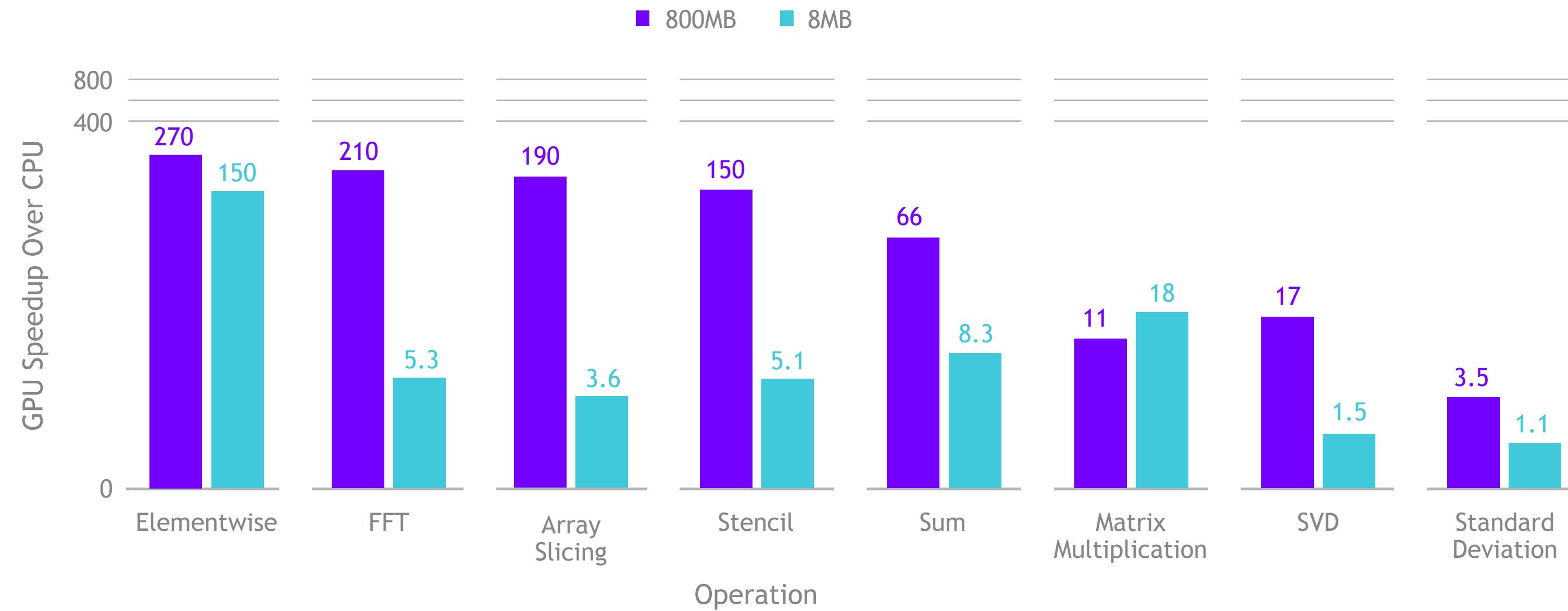
CuPy

 Numba



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

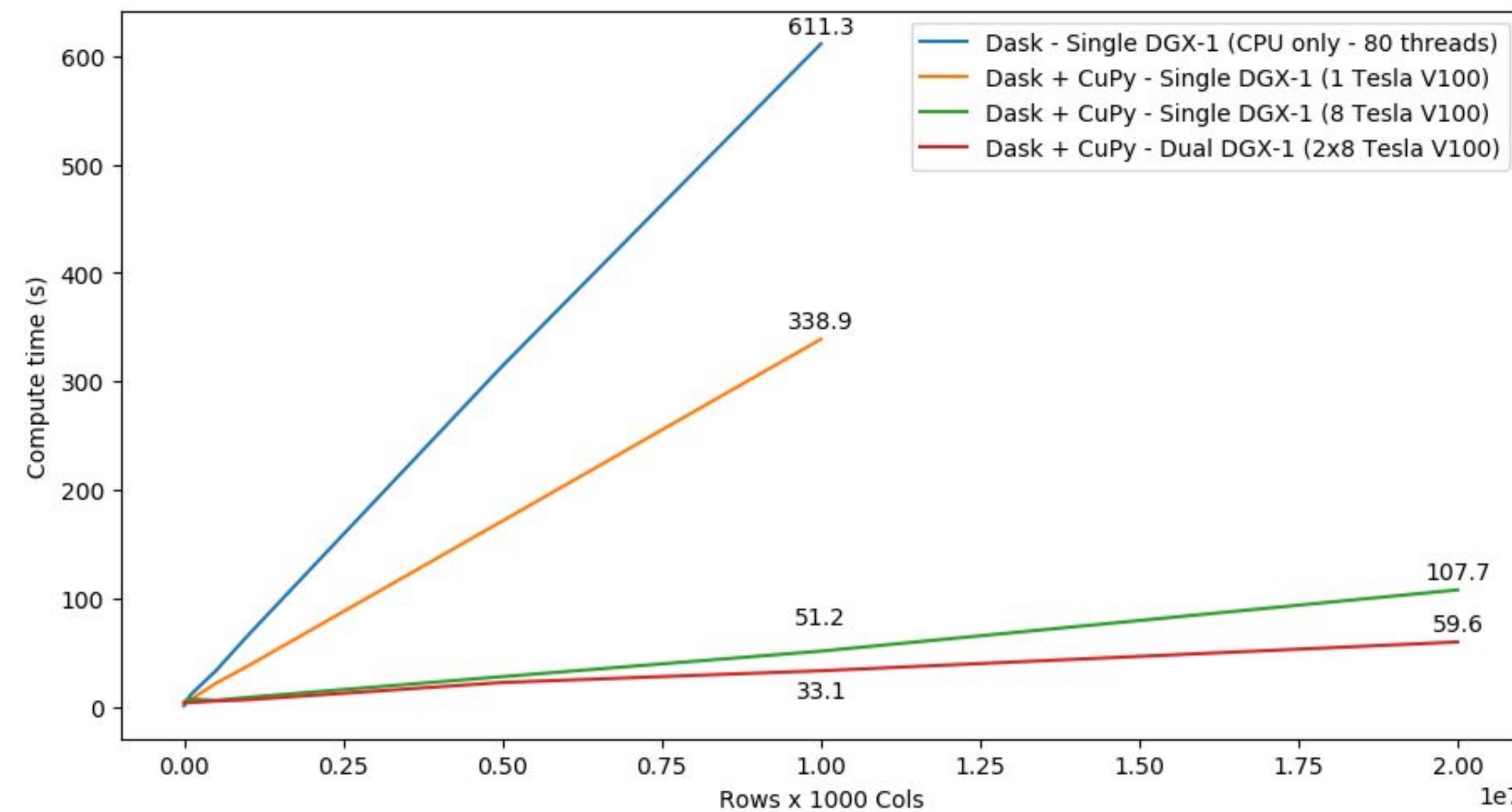
# Benchmark: Single-GPU CuPy vs NumPy



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

# SVD Benchmark

## Dask and CuPy Doing Complex Workflows



# Also...Achievement Unlocked

## Petabyte Scale Data Analytics with Dask and cuPy

Architecture	Time
Single CPU Core	2hr 39min
Forty CPU Cores	11min 30s
One GPU	1min 37s
Eight GPUs	19s

<https://blog.dask.org/2019/01/03/dask-array-gpus-first-steps>



### 3.2 Petabytes in Less Than 1 Hour

Distributed GPU Array | Parallel Reduction | Using 76x GPUs

Array Size	Wall Time (data creation + compute)
3.2 PB (20M x 20M doubles)	54min 51s

#### Cluster Configuration

20x GCP instances, each instance has:

#### CPU

1 VM socket (Intel Xeon CPU @ 2.30GHz), 2-core, 2 threads/core, 132GB mem, GbE ethernet, 950 GB disk

#### GPU

4x NVIDIA Tesla P100-16GB-PCIe (total GPU DRAM across nodes 1.22 TB)

#### Software

Ubuntu 18.04, RAPIDS 0.5.1, Dask=1.1.1, Dask-Distributed=1.1.1, CuPY=5.2.0, CUDA 10.0.130

# ETL - Arrays and DataFrames

## More Dask Awesomeness from RAPIDS

`my-taxi.ipynb`

```

3.992469787597656,40.7496337899625,2,14,0.5,0.5,0,0,0,3,15,3
1,2015-01-10 20:33:39,2015-01-10 20:42:20,3,.88,-74.002662658691406,40.734142303466797,1,N,-
73.995818375976563,40.726325988769531,1,7,0.5,0.5,1.66,0,0,3,9.96
1,2015-01-10 20:33:39,2015-01-10 21:11:35,3,18.20,-73.783842987714844,40.64435773925781,2,
N,-73.987594684492187,40.759357452392578,2,52,0,0.5,0.5,33,0,3,58.13

[1]: import dask_cudf
# df = dask_cudf.read_csv('gcs://anaconda-public-data/nyc-taxi/csv/2015/yellow_*.csv')
df = dask_cudf.read_csv('data/nyc/yellow_tripdata_2015-*.csv')
df = df.persist()

Time a full-pass computation

Most of the time here is spent reading data from disk and parsing it.

[4]: %time df.passenger_count.sum().compute()
CPU times: user 100 ms, sys: 28 ms, total: 128 ms
Wall time: 137 ms
245566747

[5]: %time df.groupby('passenger_count').trip_distance.mean().compute()
[6]: _ .to_pandas()

How well do people tip?

[7]: df2 = df[['tpep_pickup_datetime', 'trip_distance', 'tip_amount', 'fare_amount']]
df2 = df2.query('tip_amount > 0 and fare_amount > 0')
df2['hour'] = df2.tpep_pickup_datetime.dt.hour.astype('int32')
df2['tip_fraction'] = df2.tip_amount / df2.fare_amount
hour = df2.groupby('hour').tip_fraction.mean().compute().to_pandas()
hour

[8]: %matplotlib inline
hour.plot(figsize=(10, 6), title='Tip Fraction by Hour')


```

3:12 / 4:10

`copy-svd.ipynb`

Memory: 540.96 GB

### Create Random Dataset

```

import dask
import dask.array
import numpy
import cupy
rs = dask.array.random.RandomState(RandomState=Cupy.random.RandomState)
x = rs.random((1000000, 1000), chunks=(10000, 1000))
x = x.persist()


```

### Singular Value Decomposition

This computes SVD on GPU and has some communication heavy steps.

```

import dask.array.linalg
u, s, v = dask.array.linalg.svd(x)
dask.visualize(u, s, v)
u, s, v = dask.persist(u, s, v)


```

### Inspect output

```

print(u[:10].compute())
print(s[:10].compute())
print(v[:10].compute())

from distributed.utils import format_bytes
print(format_bytes(u.nbytes))
print(format_bytes(s.nbytes))
print(format_bytes(v.nbytes))


```

3:25 / 3:42

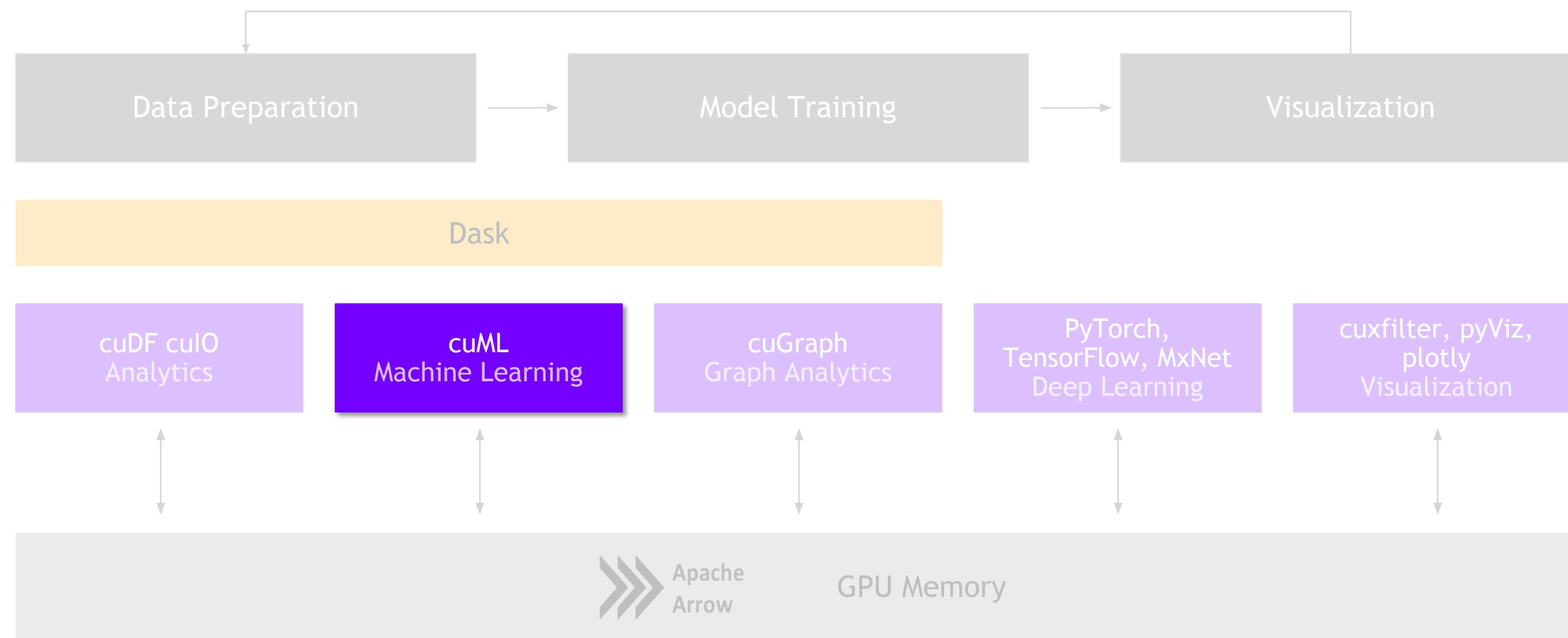
<https://youtu.be/gV0cykgsTPM>

[https://youtu.be/R5CiXti\\_MWo](https://youtu.be/R5CiXti_MWo)

cuML

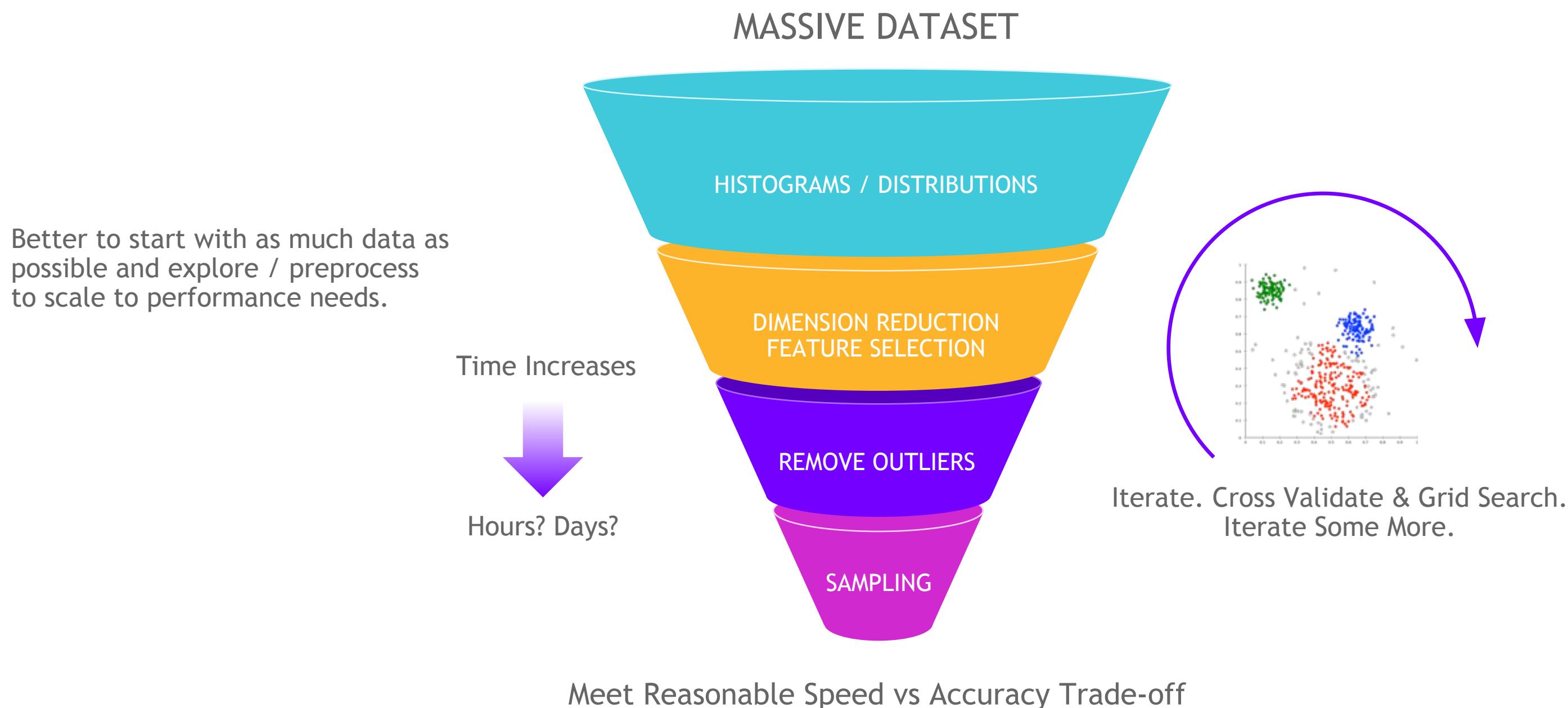
# Machine Learning

## More Models More Problems

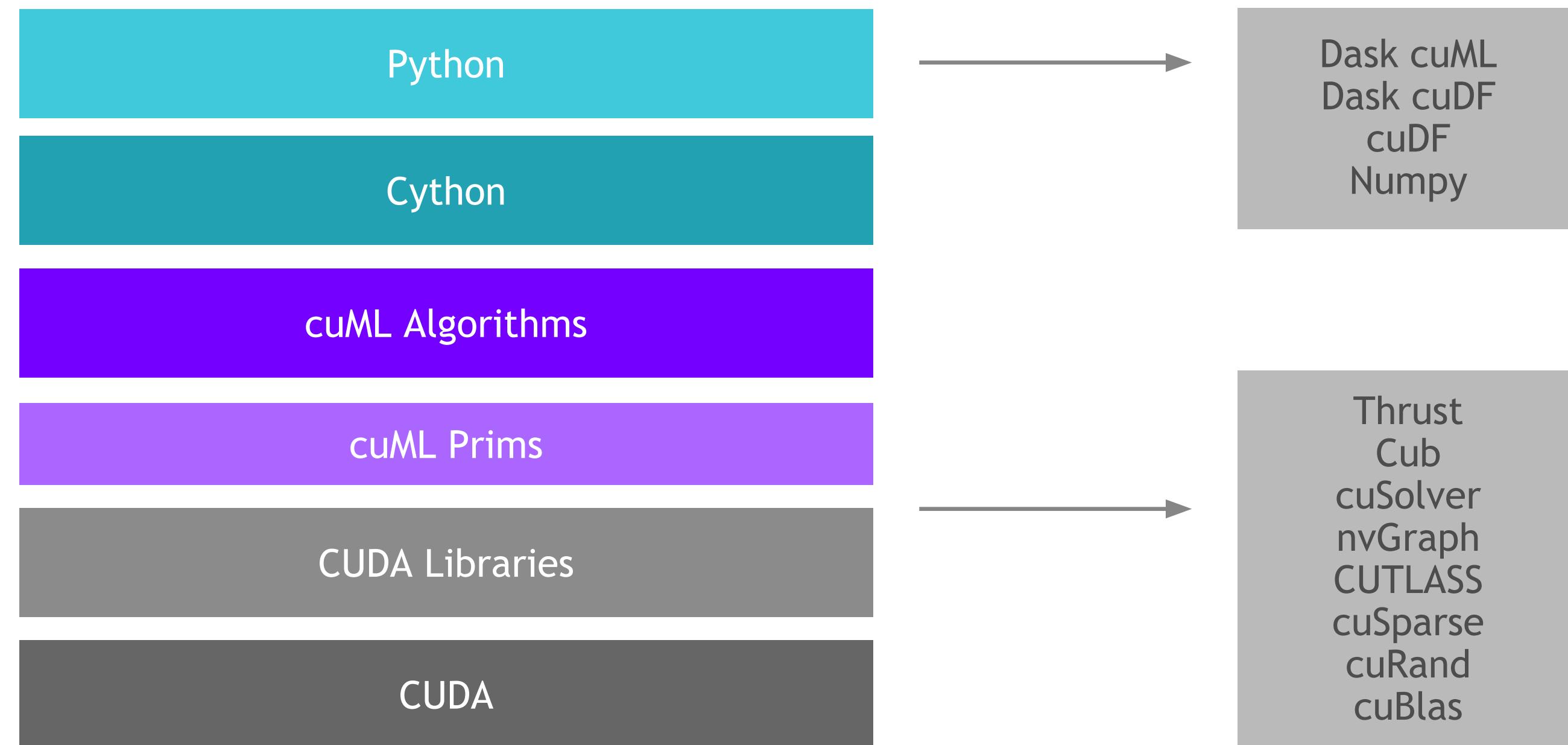


# Problem

## Data Sizes Continue to Grow



# ML Technology Stack



# RAPIDS Matches Common Python APIs

## CPU-based Clustering

```
from sklearn.datasets import make_moons
import pandas

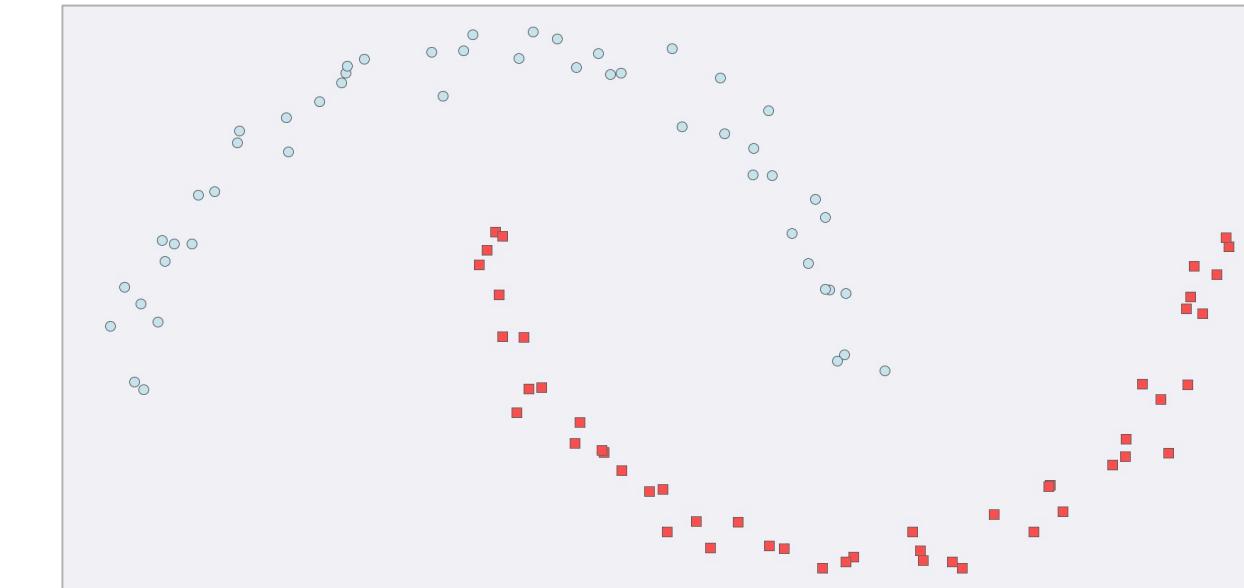
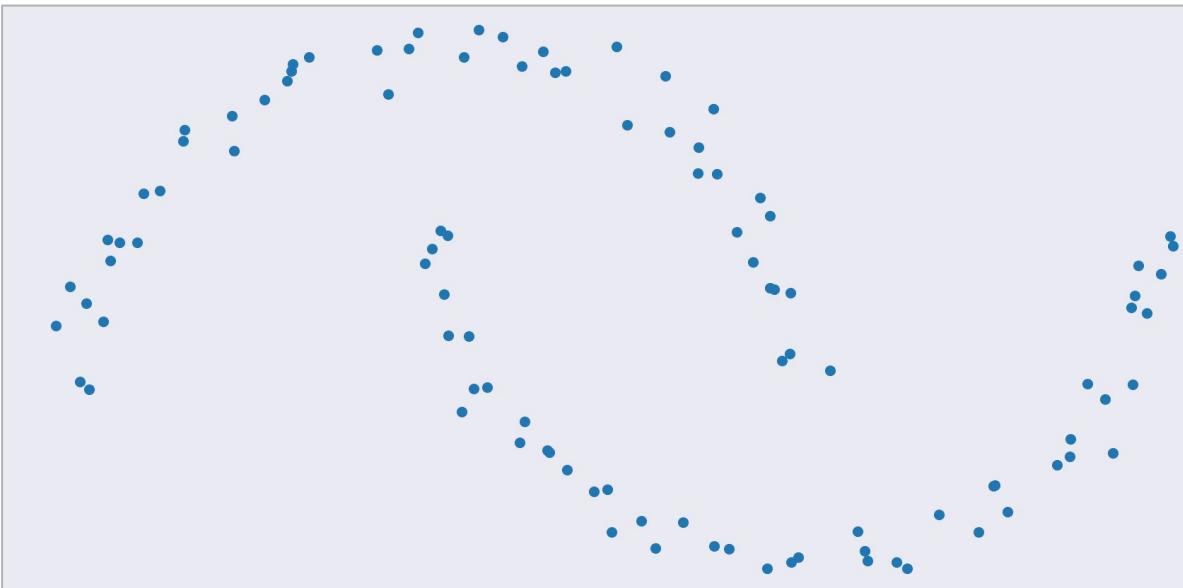
X, y = make_moons(n_samples=int(1e2),
                   noise=0.05, random_state=0)

X = pandas.DataFrame({'feat%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

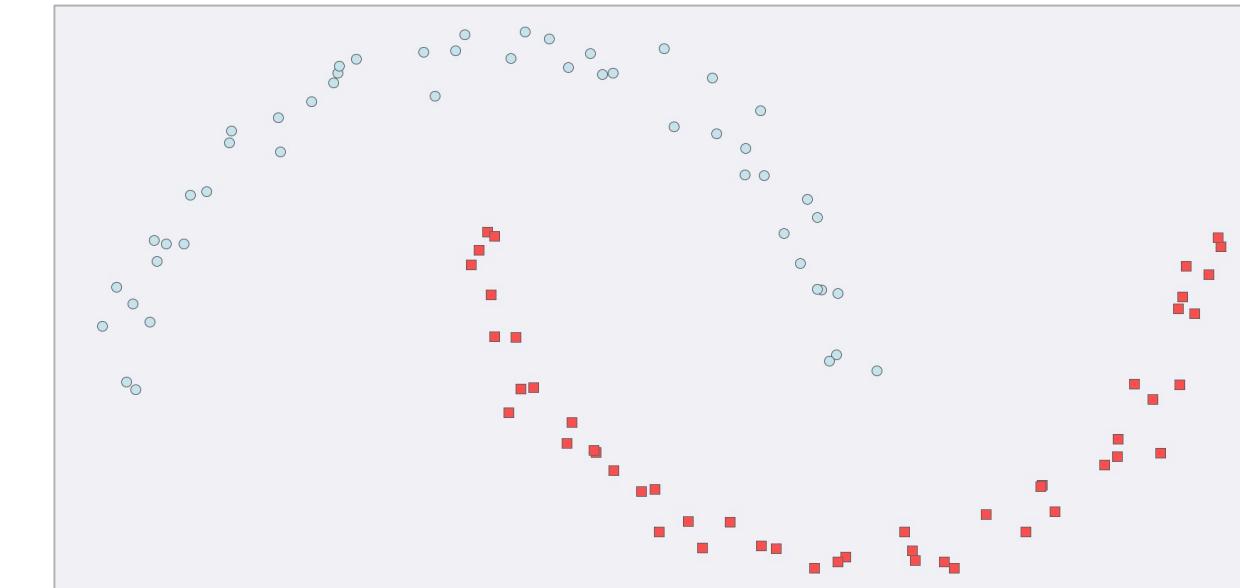
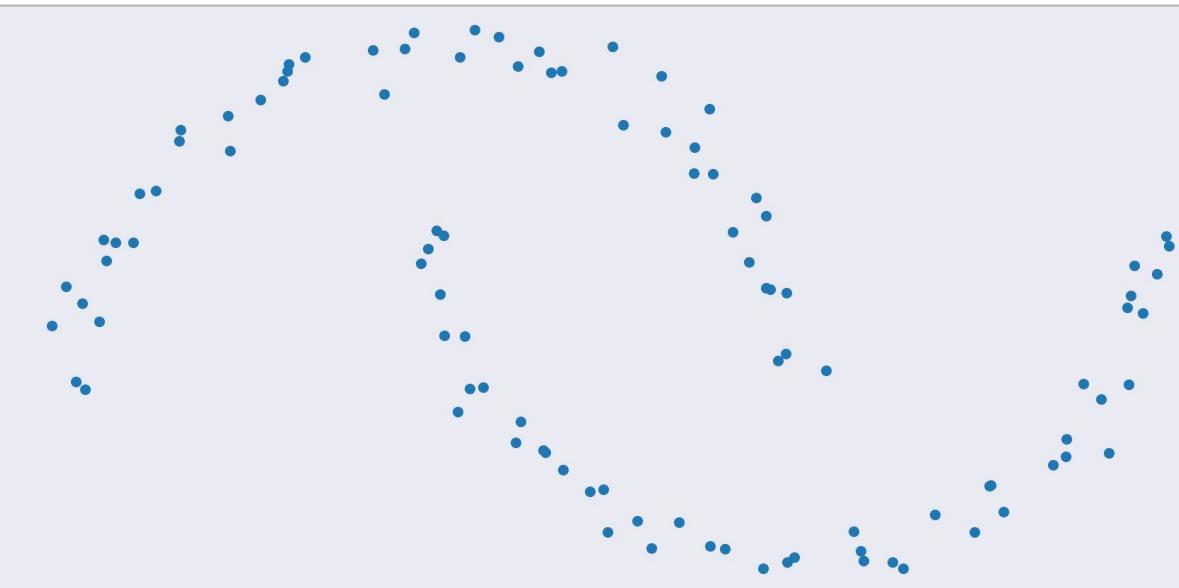
X, y = make_moons(n_samples=int(1e2),
                   noise=0.05, random_state=0)

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)

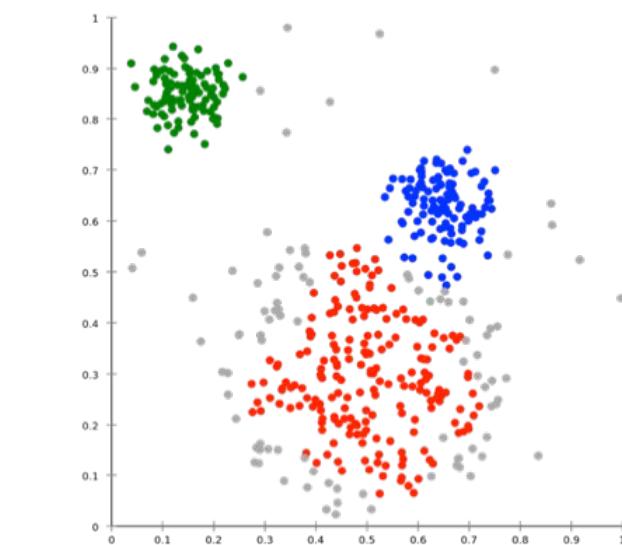
dbscan.fit(X)

y_hat = dbscan.predict(X)
```



# Algorithms

## GPU-accelerated Scikit-Learn



Cross Validation

Hyper-parameter Tuning

More to come!

Classification \ Regression

Inference

Clustering

Decomposition &  
Dimensionality Reduction

Time Series

Decision Trees / Random Forests  
**Linear/Lasso/Ridge Regression**  
Logistic Regression  
K-Nearest Neighbors  
Support Vector Machine Classification

Random Forest / GBDT Inference

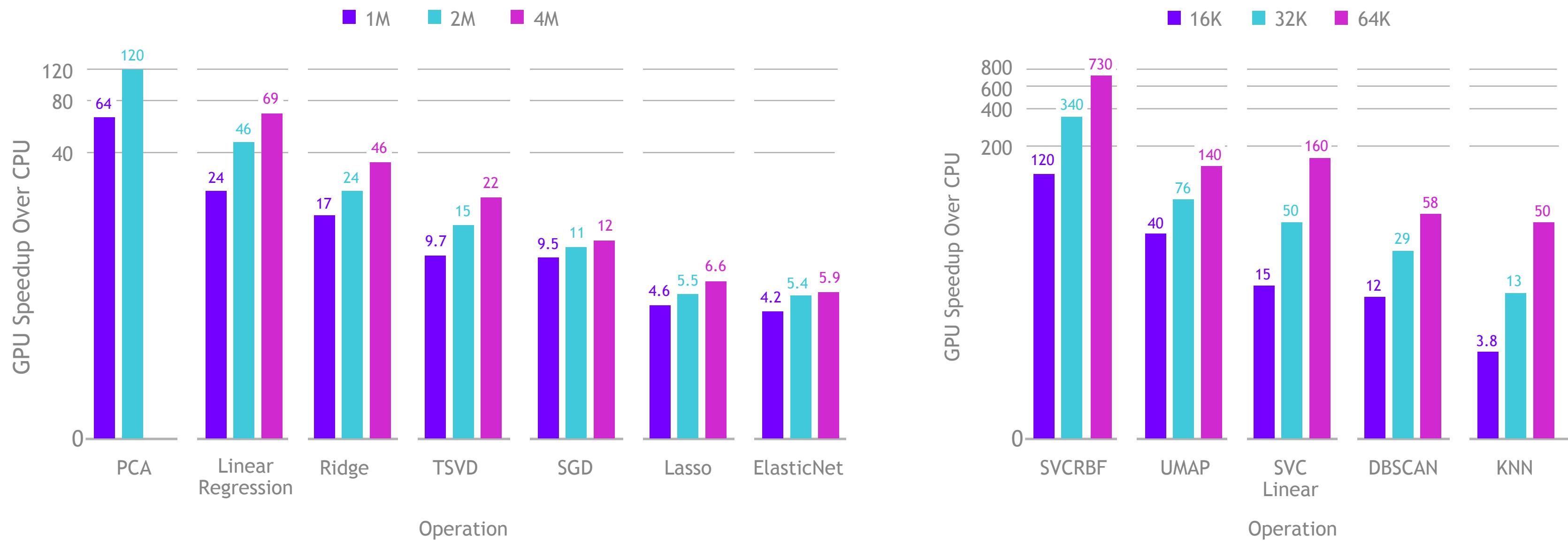
K-Means  
DBSCAN  
Spectral Clustering

Principal Components  
Singular Value Decomposition  
UMAP  
Spectral Embedding  
T-SNE

Holt-Winters  
Seasonal ARIMA

Key:  
Preexisting | NEW or enhanced for 0.14

# Benchmarks: Single-GPU cuML vs Scikit-learn



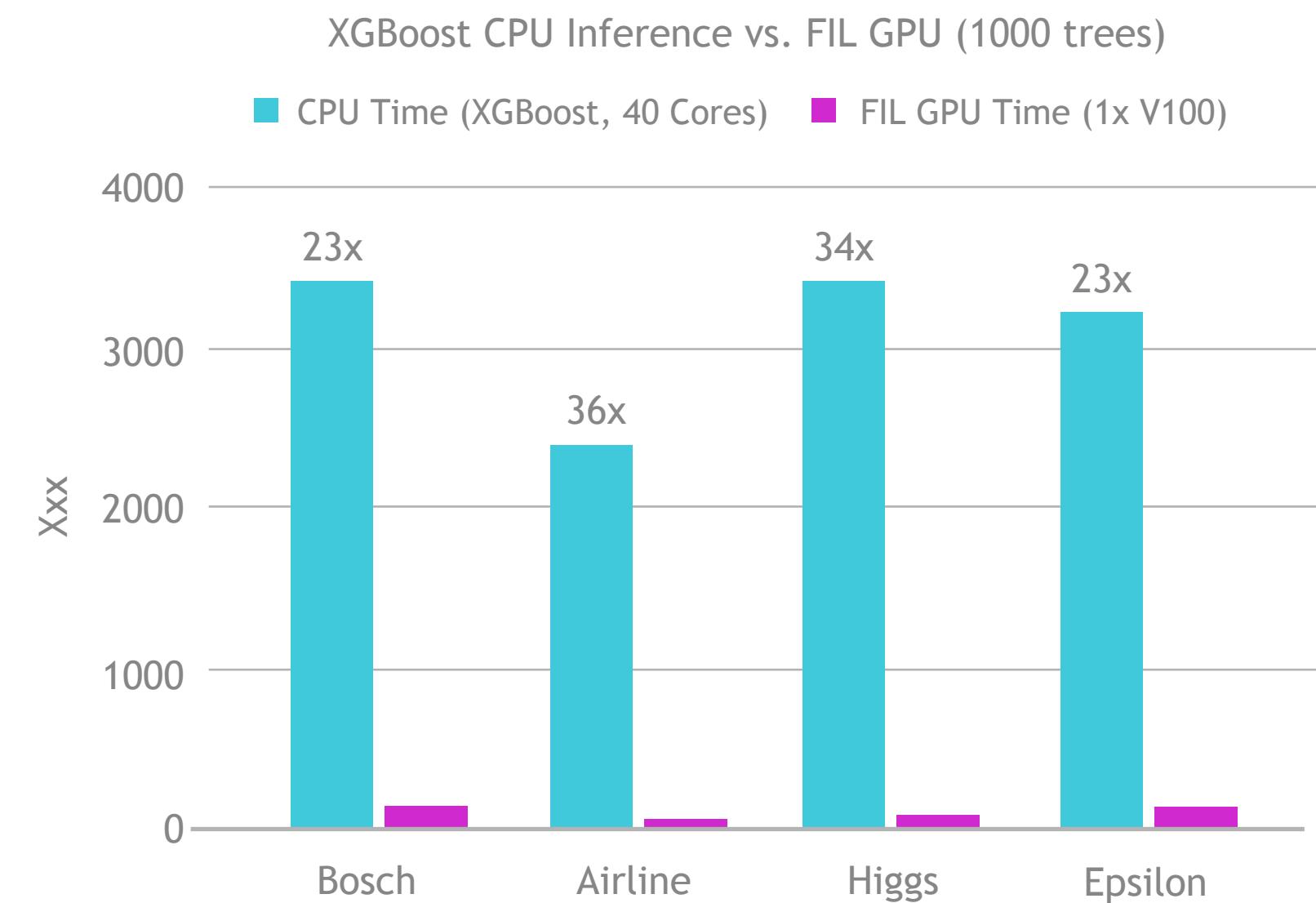
1x V100 vs. 2x 20 Core CPU

# 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 0.14 comes paired with XGBoost 1.1

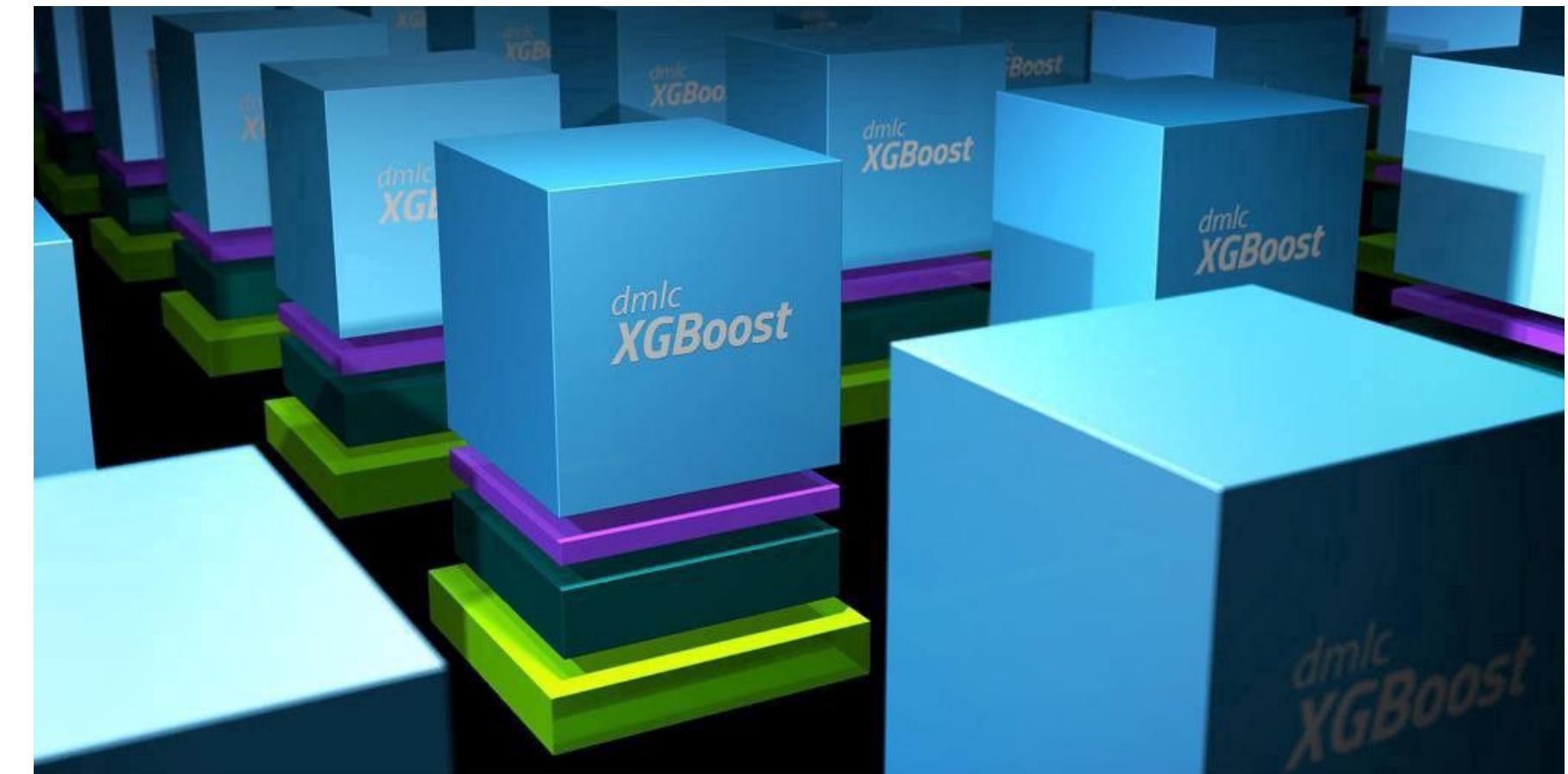
XGBoost now builds on the GoAI interface standards to provide zero-copy data import from cuDF, cuPY, Numba, PyTorch and more

Official Dask API makes it easy to scale to multiple nodes or multiple GPUs

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



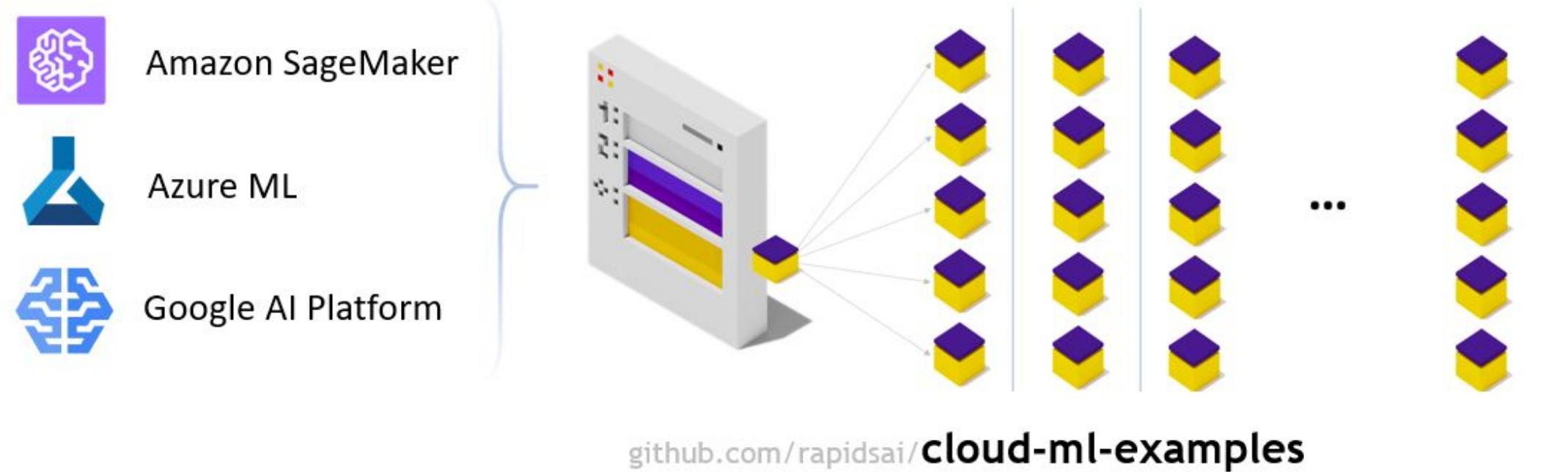
# RAPIDS Integrated into Cloud ML Frameworks

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.

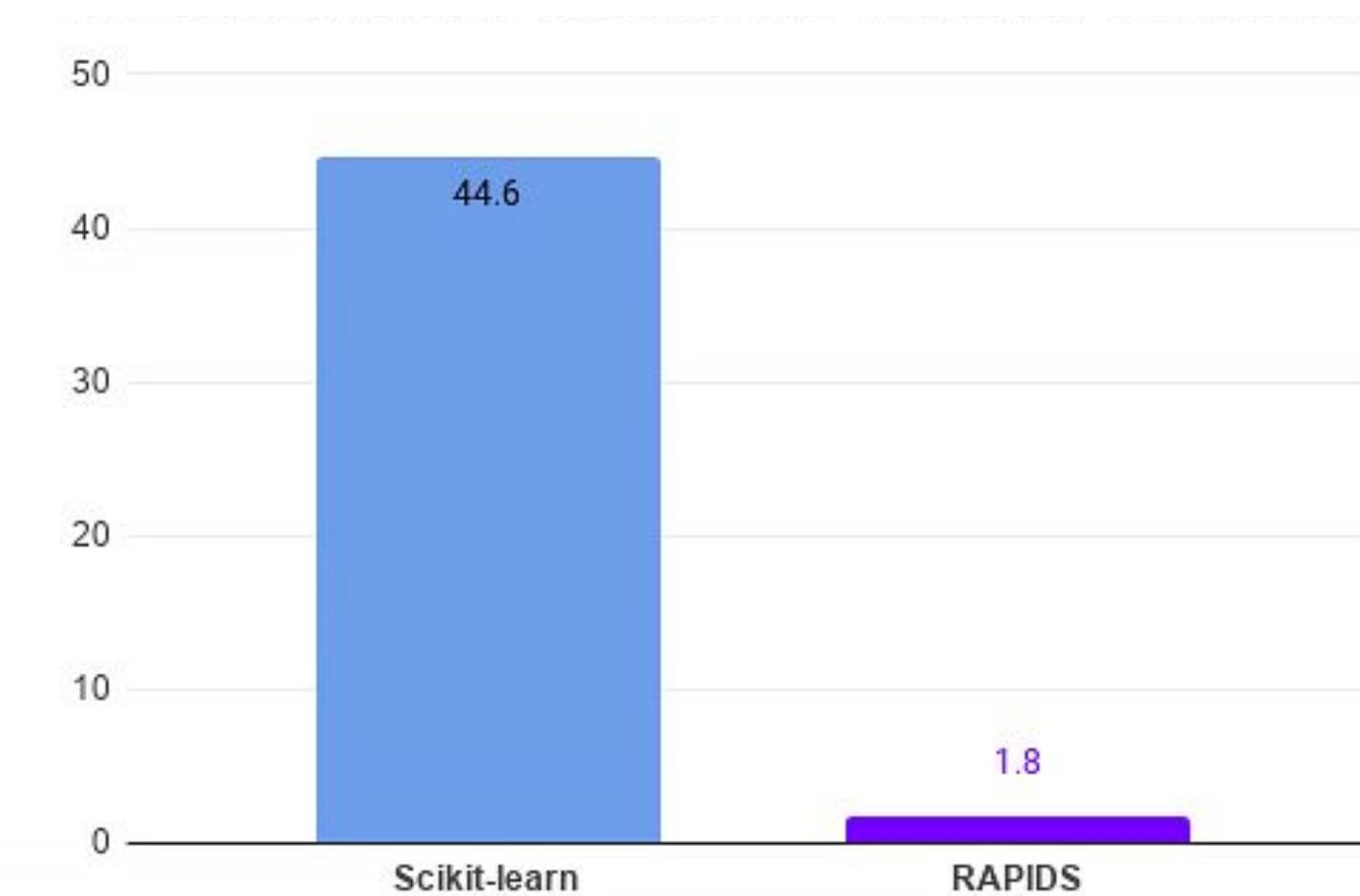
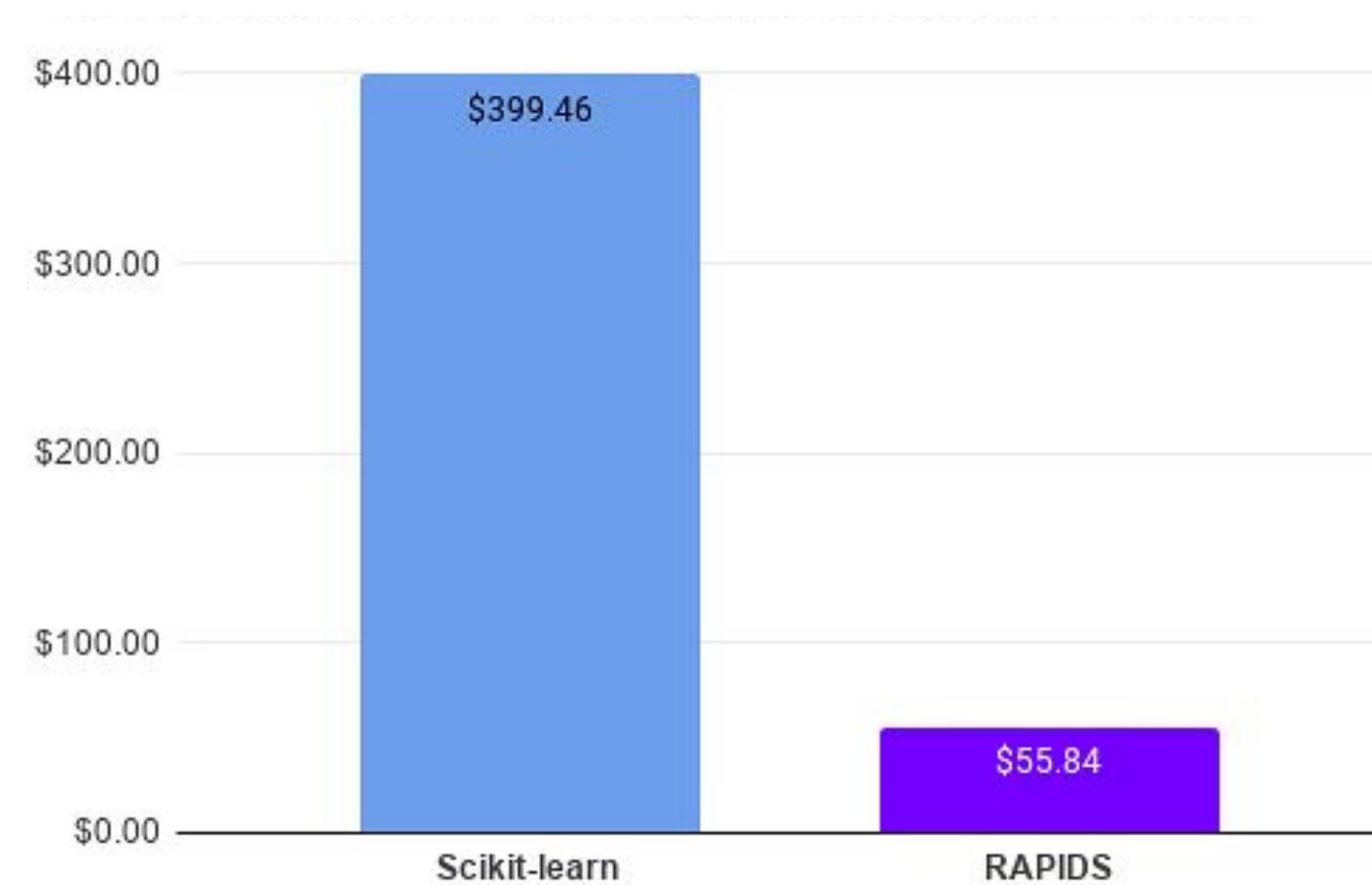
The RAPIDS team works closely with major cloud providers and OSS solution providers to provide code samples to get started with HPO in minutes

<https://rapids.ai/hpo>



# 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 0.14

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

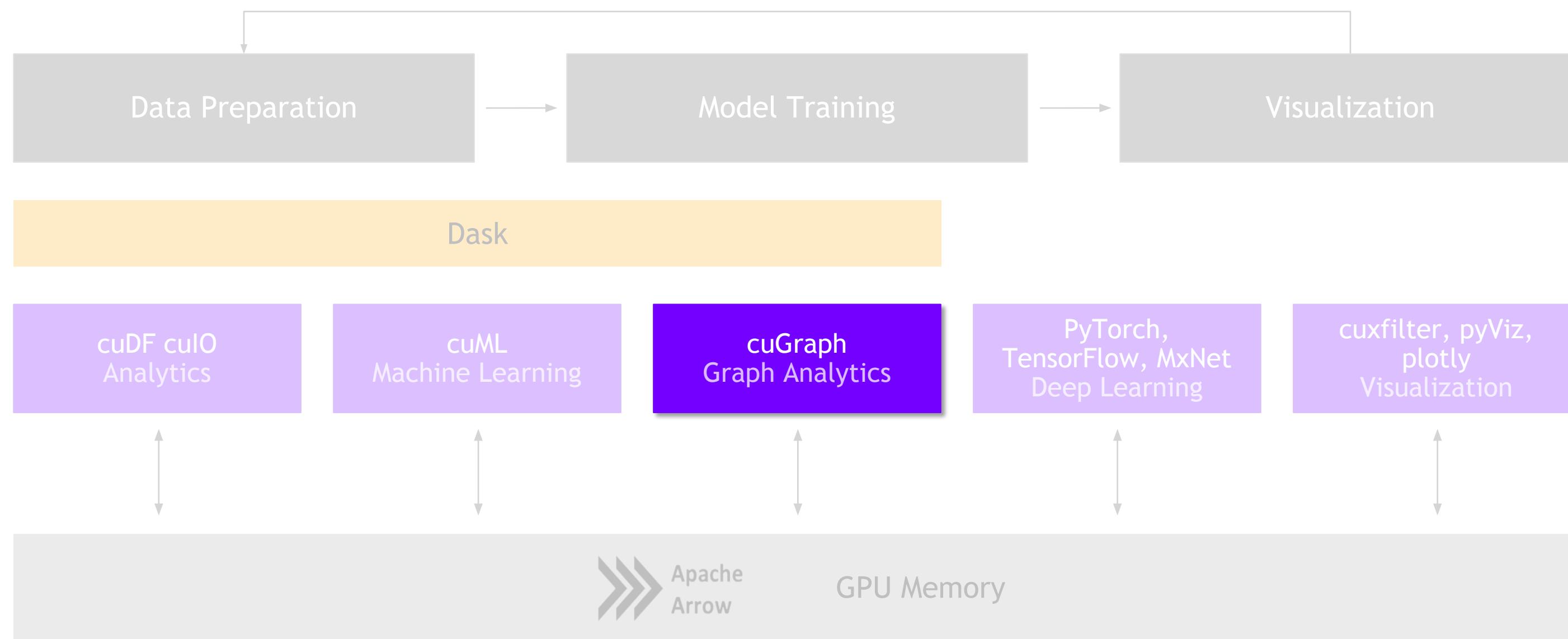
Later in 2020 - RAPIDS 1.0

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

Need a Picture here

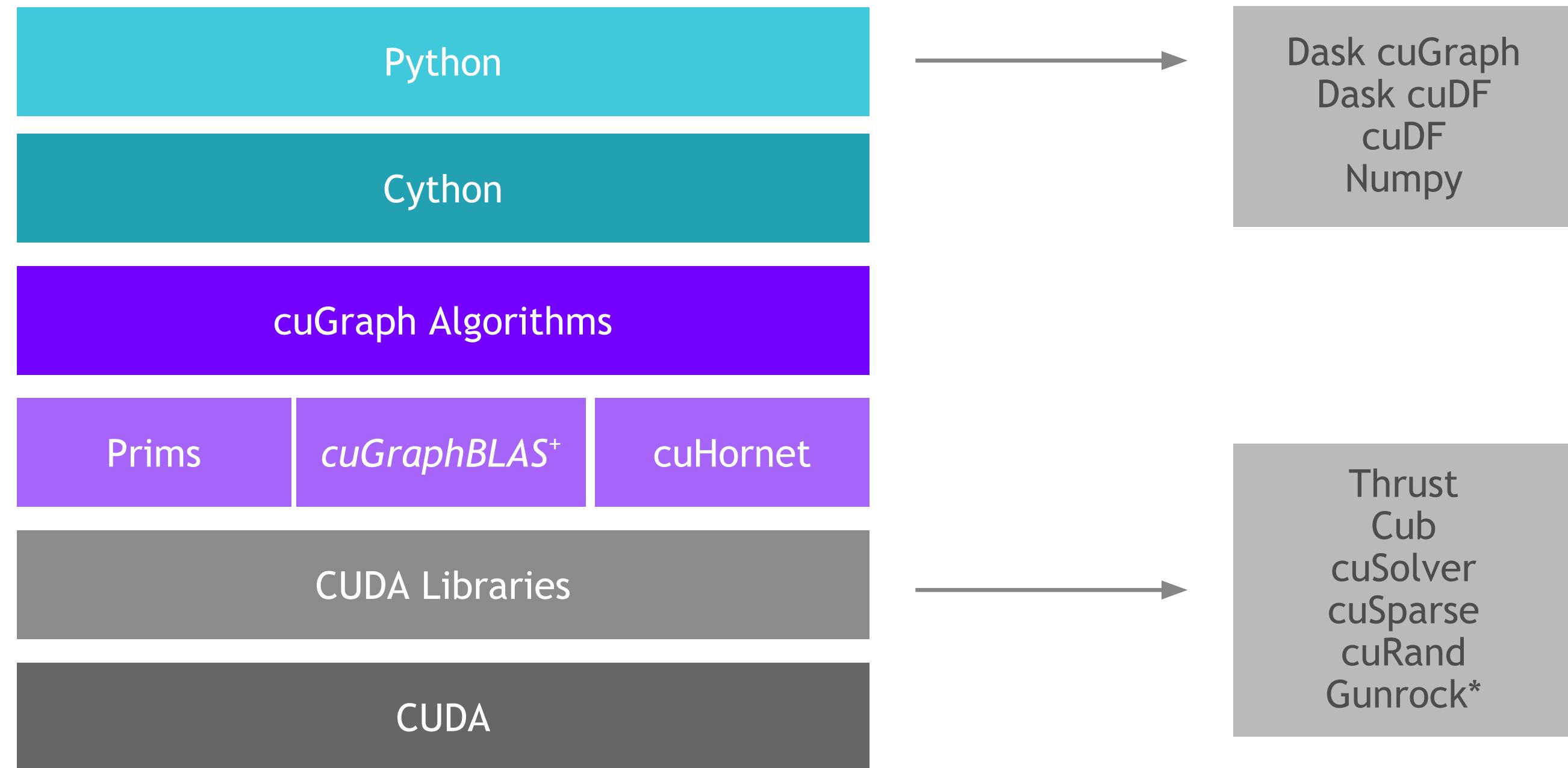
### MULTIPLE APIs

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

### GROWING FUNCTIONALITY

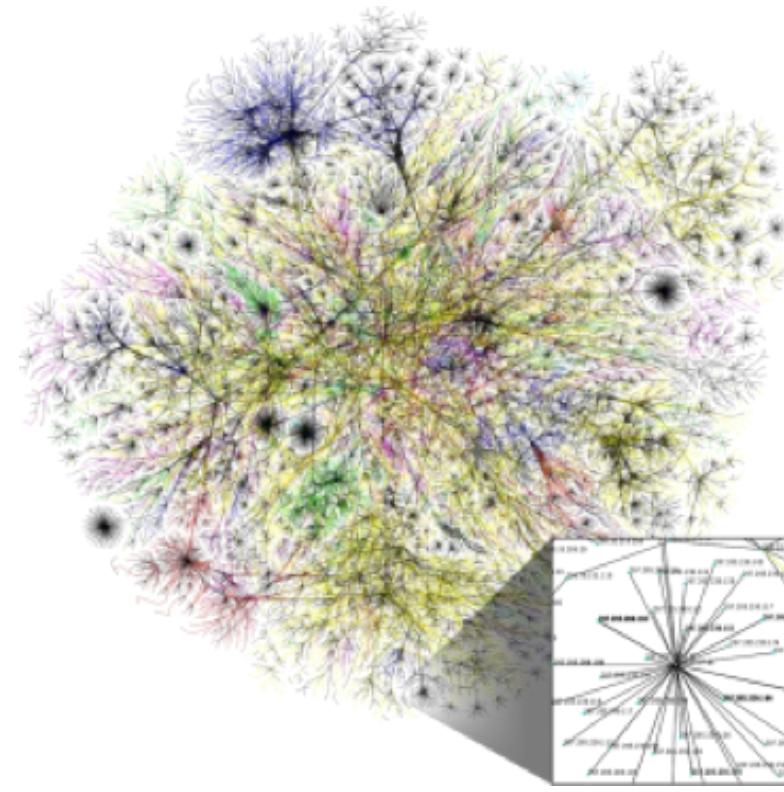
- Extensive collection of algorithm, primitive, and utility functions

# Graph Technology Stack



+cuGraphBLAS is still in development and will be ready late 2020

\* Gynrock 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** (redone for 0.14)  
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

Link Prediction

Jaccard  
Weighted Jaccard  
Overlap Coefficient

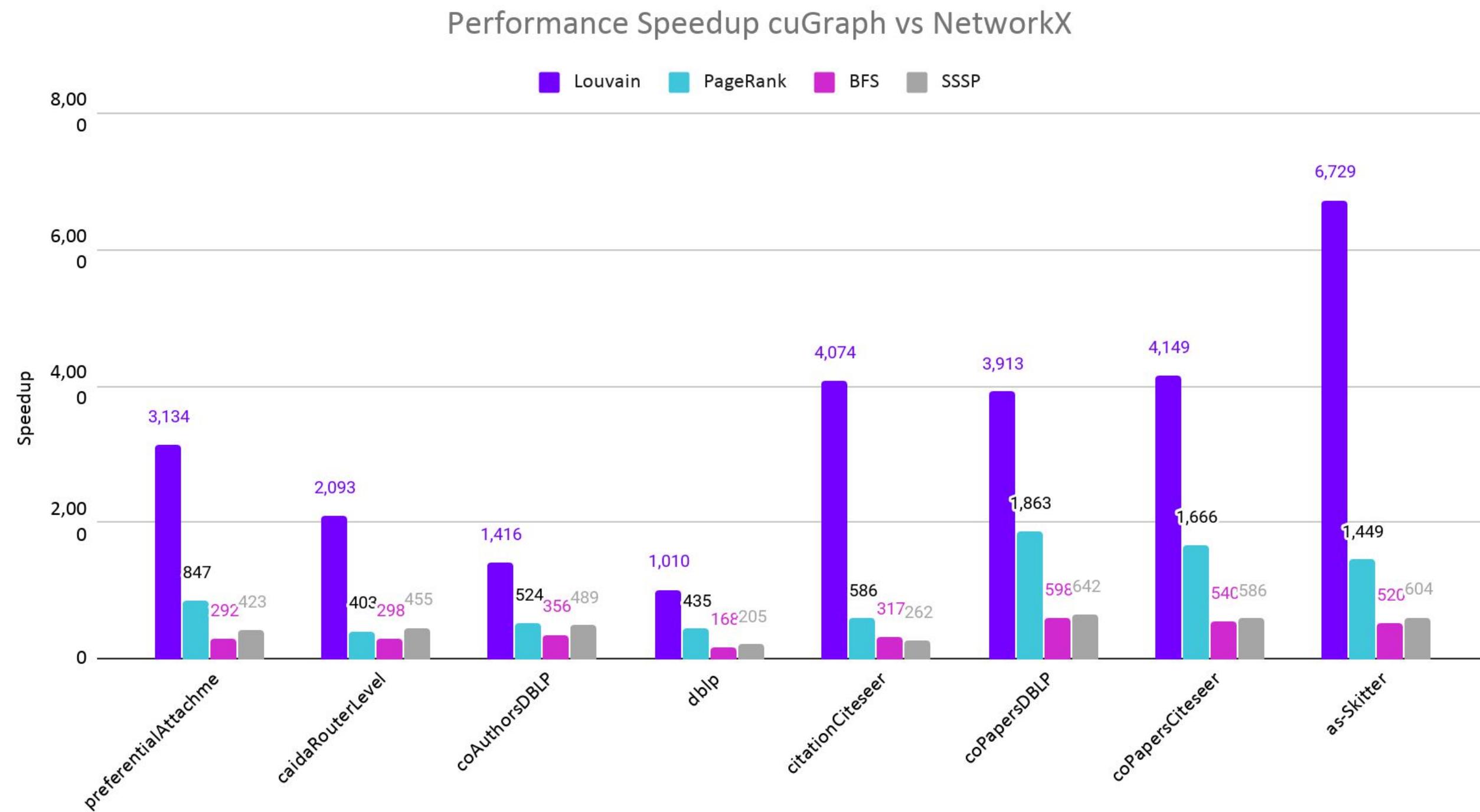
Traversal

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

Centrality

Katz  
**Betweenness Centrality** (redone in 0.14)

# Benchmarks: Single-GPU cuGraph vs NetworkX



Dataset	Nodes	Edges
preferentialAttachment	100,000	999,970
caidaRouterLevel	192,244	1,218,132
coAuthorsDBLP	299,067	299,067
Dblp-2010	326,186	1,615,400
citationCiteseer	268,495	2,313,294
coPapersDBLP	540,486	20,491,458
coPapersCiteseer	434,102	32,073,440
As-Skitter	1,696,415	22,190,596

# 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 0.14

cuGRAPH	Single-GPU	Multi-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			

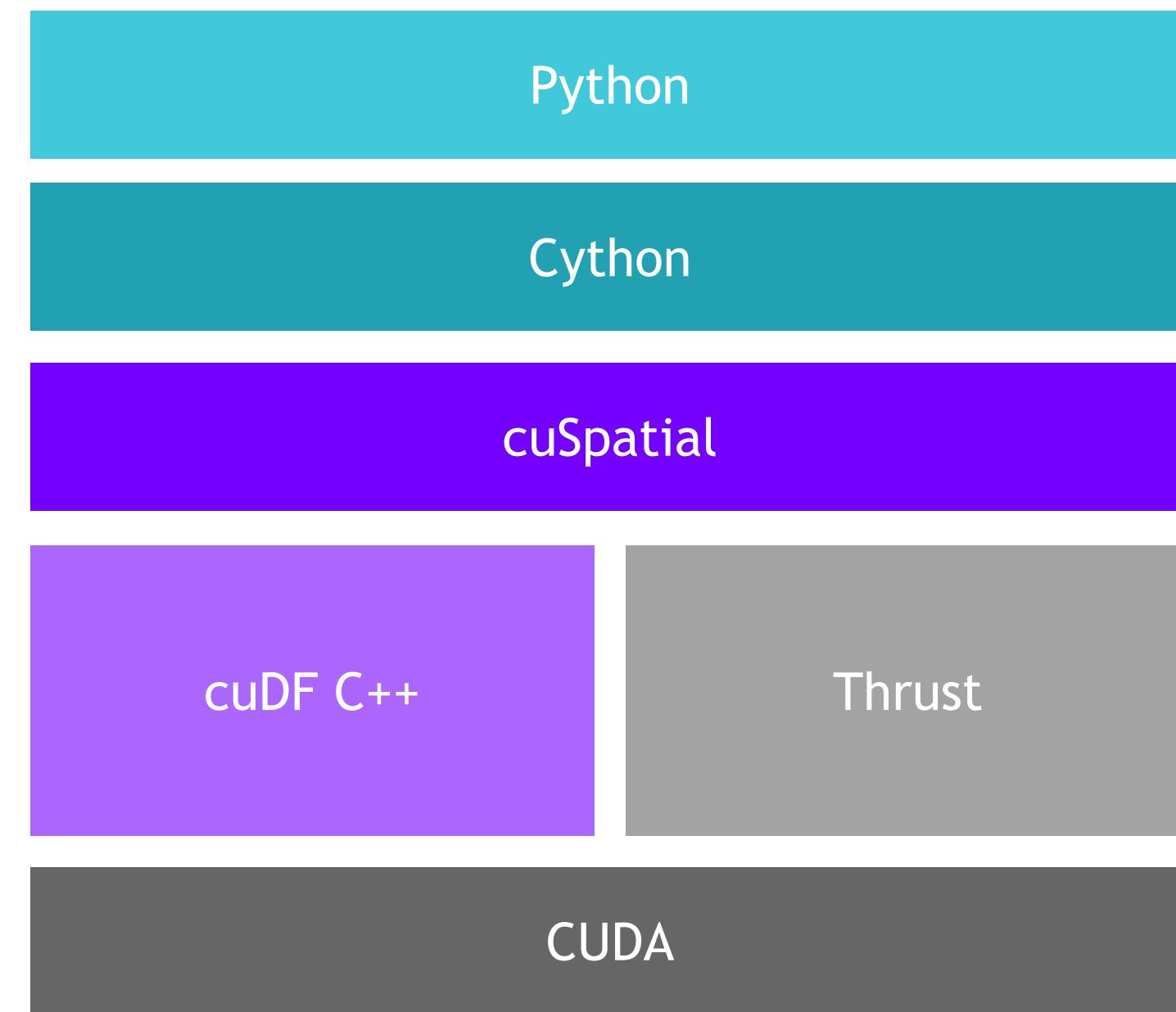
# Road to 1.0

## June 2020 - RAPIDS 1.0

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

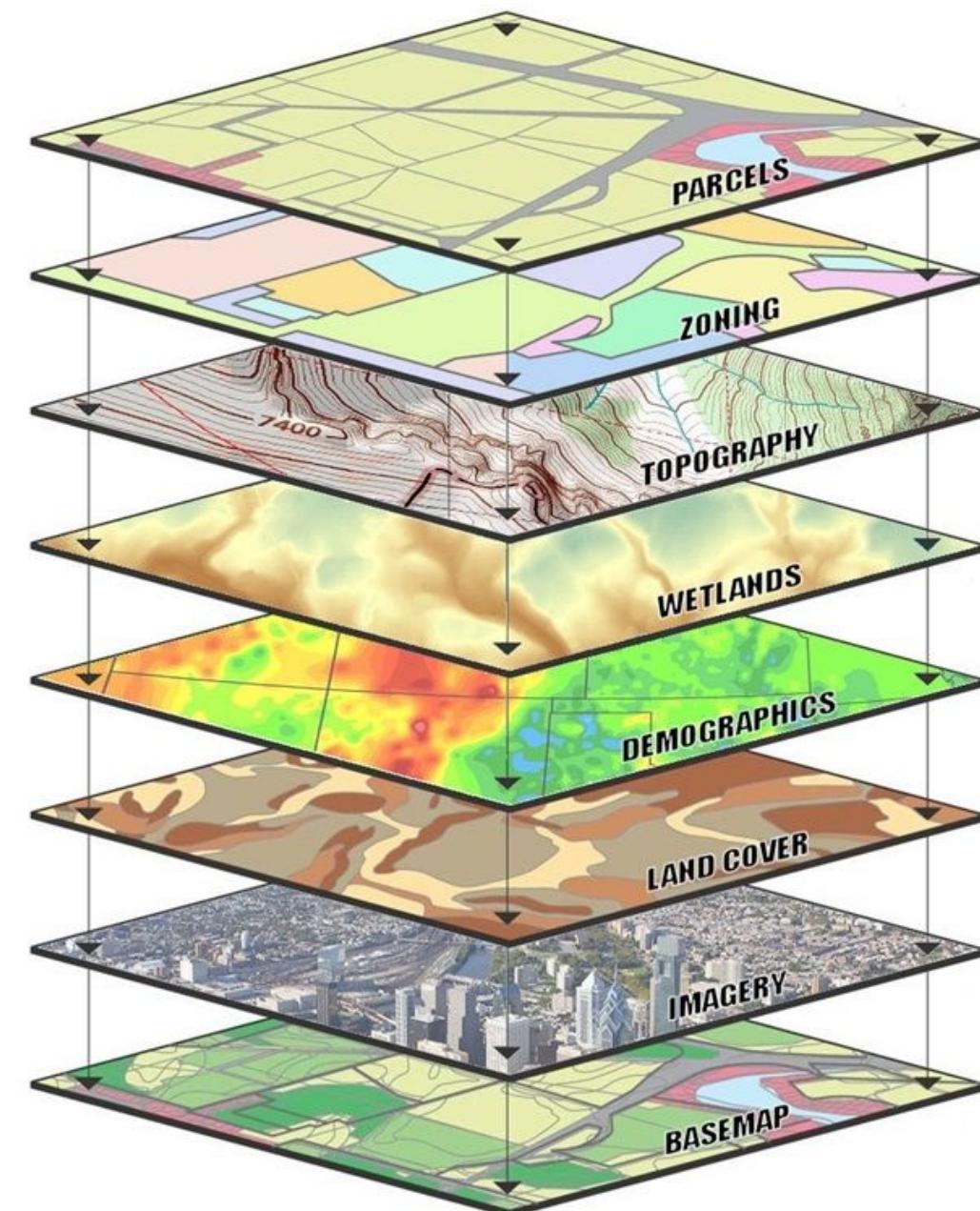
- Up to 1000x faster CPU spatial libraries
- Python and C++ APIs for maximum usability and integration

## 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

## 0.14 and Beyond

Layer	0.14 Functionality	Functionality Roadmap (2020)
<b>High-level Analytics</b>	C++ Library w. Python bindings enabling distance, speed, trajectory similarity, trajectory clustering	C++ Library w. Python bindings for additional spatio-temporal trajectory clustering, acceleration, dwell-time, salient locations, trajectory anomaly detection, origin destination, etc.
<b>Graph Layer</b>	cuGraph	Map matching, Djikstra algorithm, Routing
<b>Query Layer</b>	Spatial Window	Nearest Neighbor, KNN, Spatiotemporal range search and joins
<b>Index Layer</b>		Grid, Quad Tree, R-Tree, Geohash, Voronoi Tessellation
<b>Geo-operations</b>	Point in polygon (PIP), Haversine distance, Hausdorff distance, lat-lon to xy transformation	Line intersecting polygon, Other distance functions, Polygon intersection, union
<b>Geo-Representation</b>	Shape primitives, points, polylines, polygons	Additional shape primitives

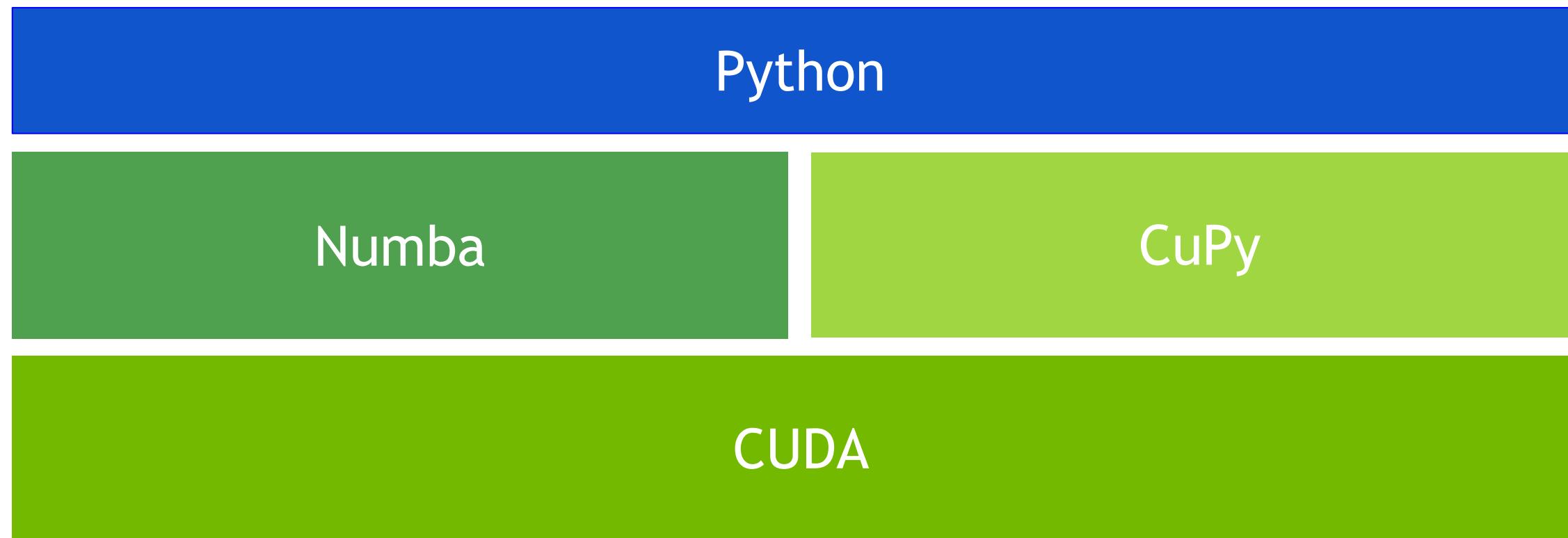
# cuSpatial

## 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

Periodogram  
Welch  
Spectrogram

### Peak Finding

More to come!

### Spectral Analysis

# 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)
```

# Community

# Ecosystem Partners

## CONTRIBUTORS



## ADOPTERS

Booz | Allen | Hamilton



kinētica



o m n i · s c i



Uber



## OPEN SOURCE



nucleo



# Building on Top of RAPIDS

A Bigger, Better, Stronger Ecosystem for All

**nuclio**

High-Performance Serverless event  
and data processing that utilizes  
RAPIDS for GPU Acceleration

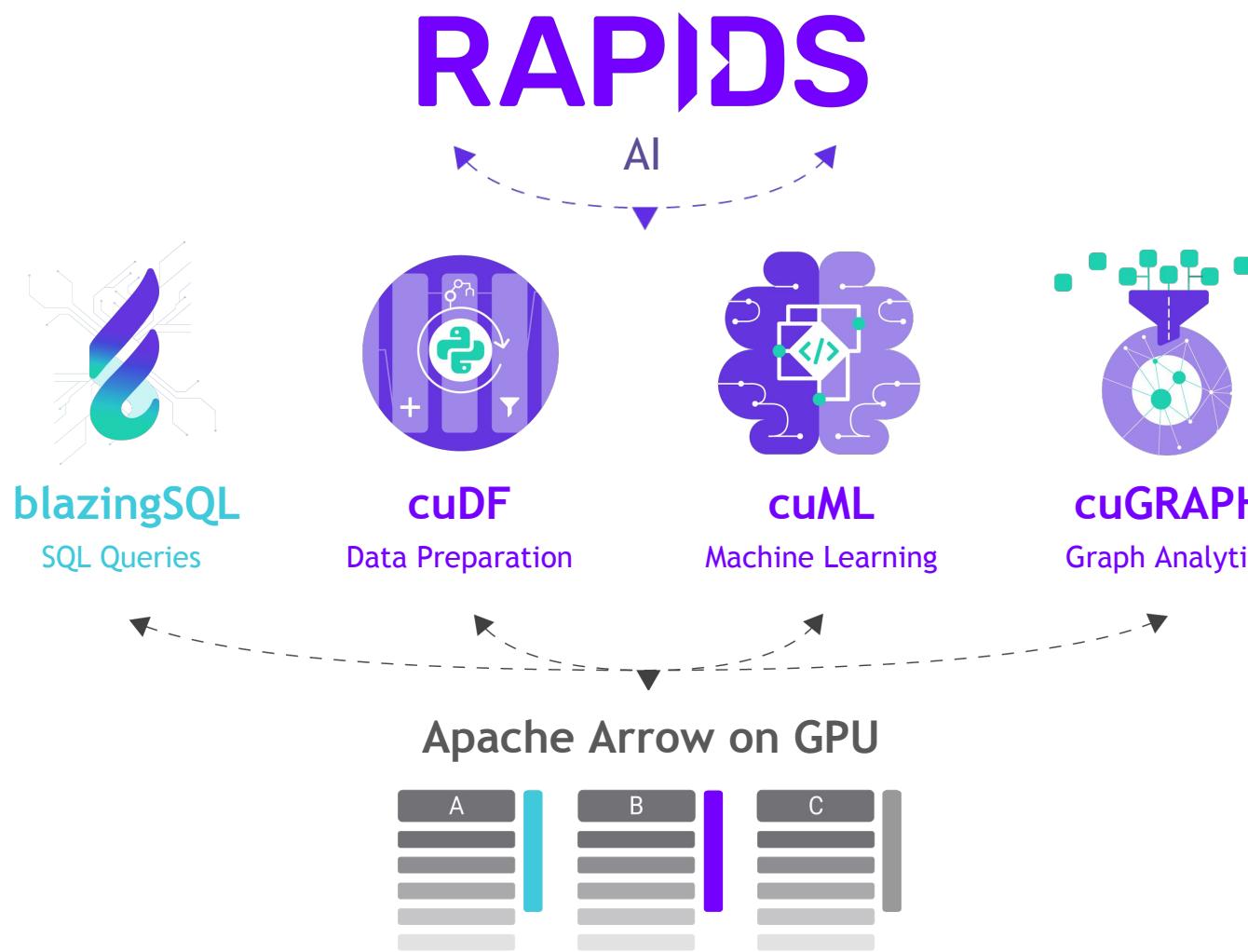


GPU accelerated SQL engine  
built on top of RAPIDS

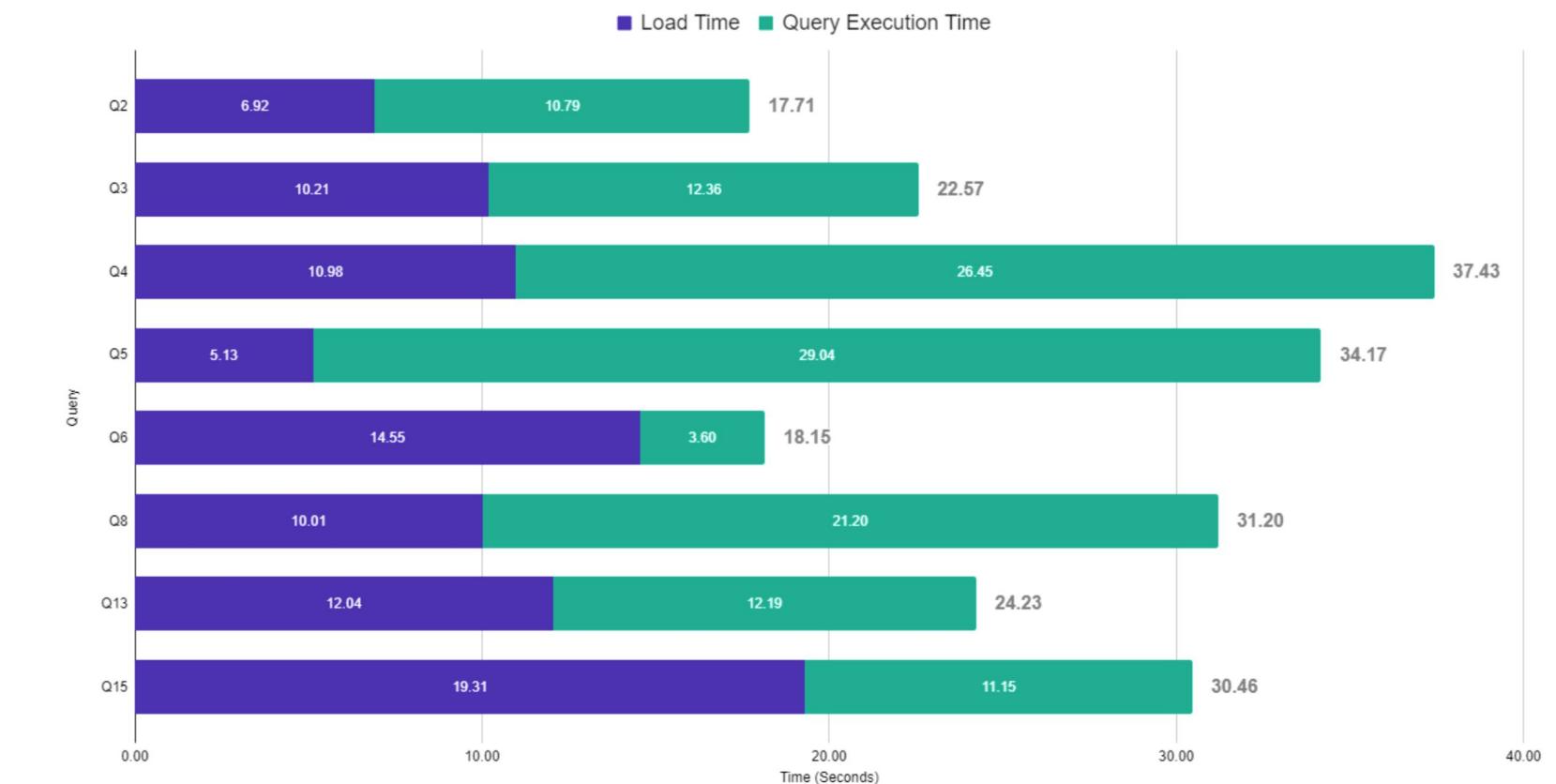
**Streamz**

Distributed stream processing  
using RAPIDS and Dask

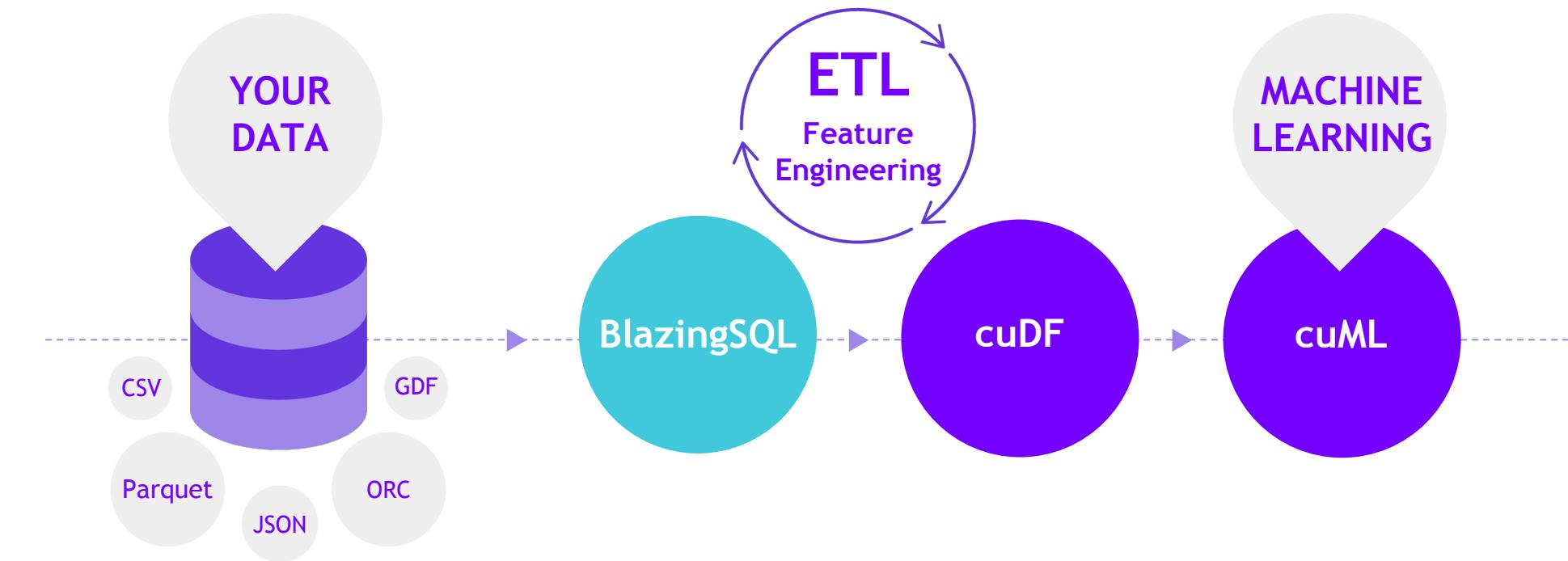
# BlazingSQL



TPC-H SF1000 Query Times - GCS Storage



# BlazingSQL



```
from blazingsql import BlazingContext
import cudf

bc = BlazingContext()

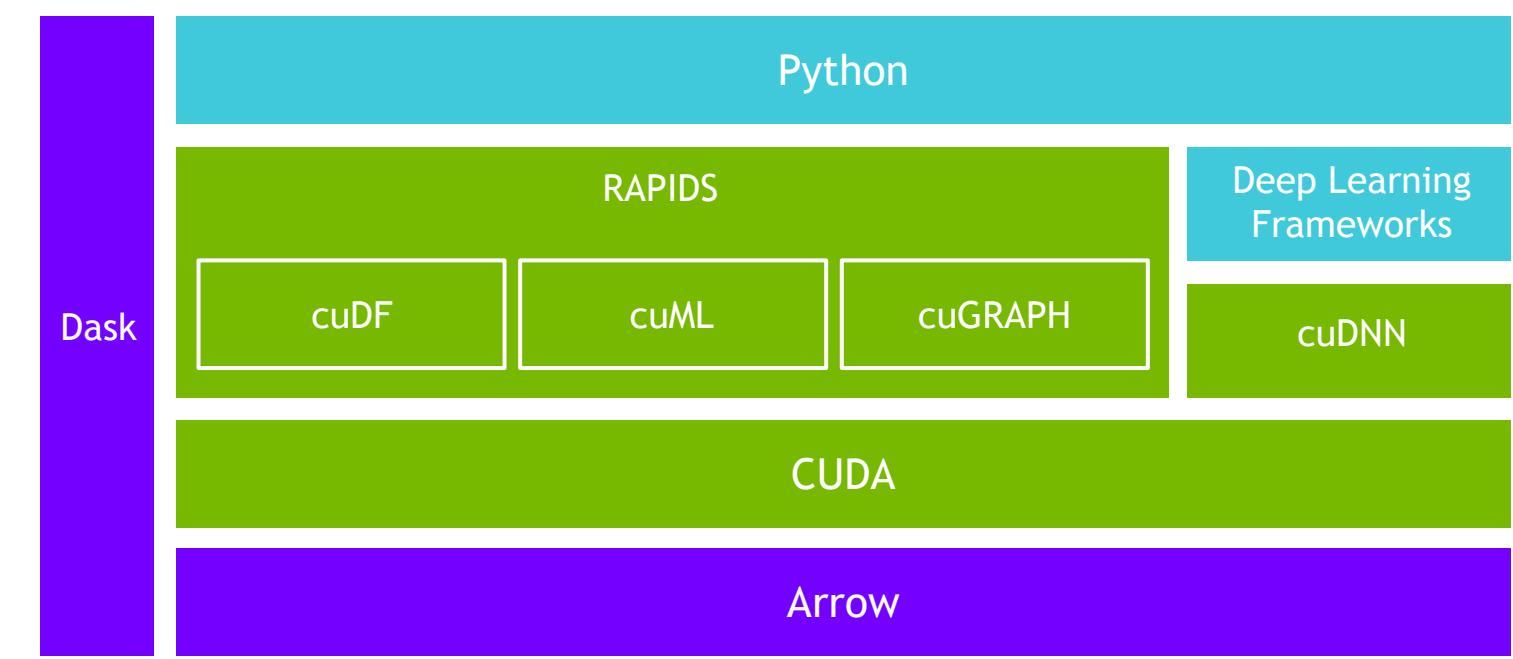
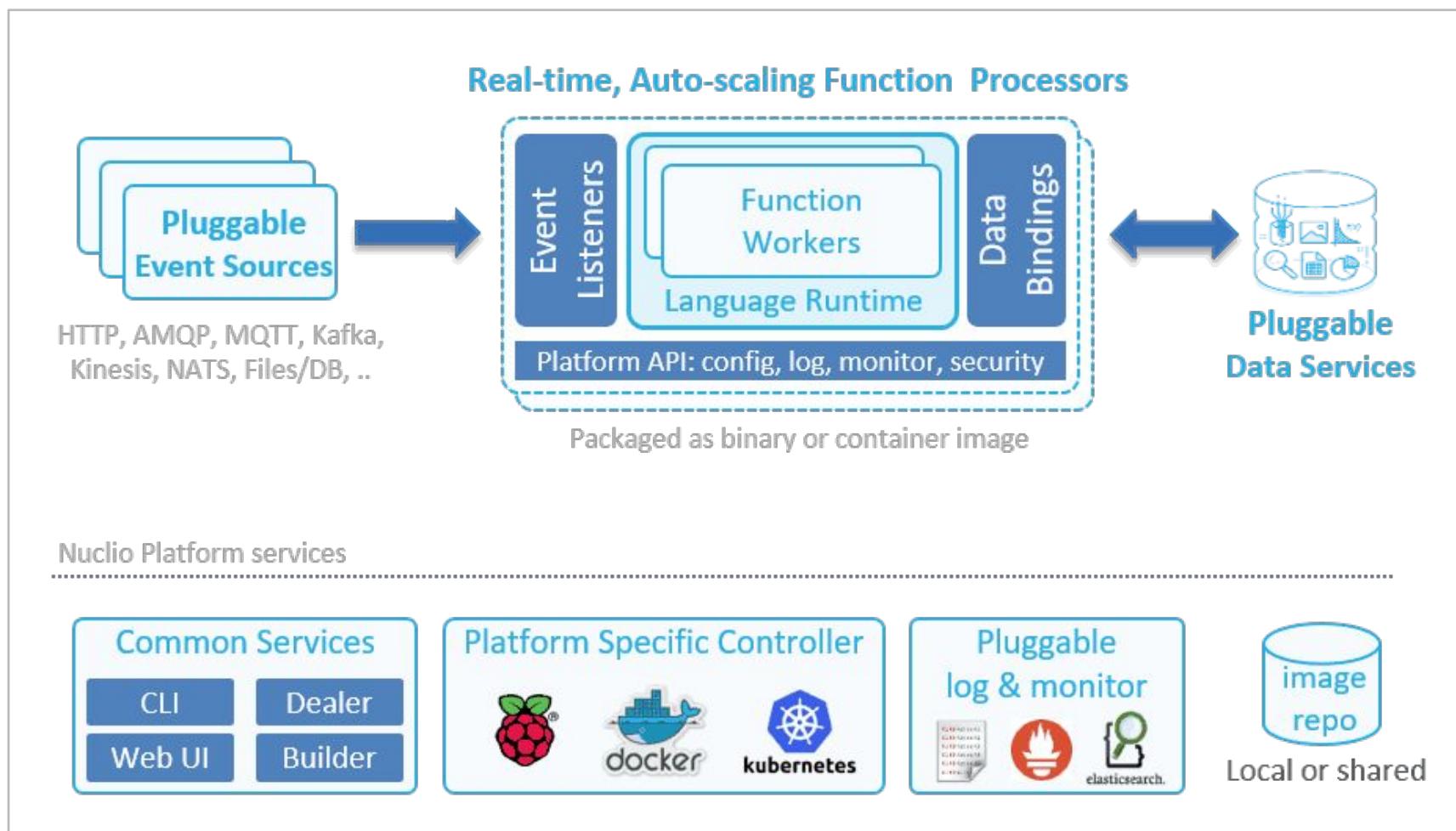
bc.s('bsql', bucket_name='bsql', access_key_id='<access_key>', secret_key='<secret_key>')

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

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

# RAPIDS + Nuclio

## Serverless Meets GPUs



<https://towardsdatascience.com/python-pandas-at-extreme-performance-912912b1047c>

# 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](#).

Preferred ↓ Advanced ↓

METHOD	Conda 📦	Docker + Examples 🎨	Docker + Dev Env 🛠️	Source ⚙️			
RELEASE	Stable (0.14)		Nightly (0.15a)				
PACKAGES	All Packages	cuDF	cuML	cuGraph	cuSignal	cuSpatial	cuxfilter
LINUX	Ubuntu 16.04 🐧	Ubuntu 18.04 🐧		CentOS 7 🐧		RHEL 7 🐧	
PYTHON	Python 3.6			Python 3.7			
CUDA	CUDA 10.0		CUDA 10.1.2		CUDA 10.2		

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

COMMAND

```
conda install -c rapidsai -c nvidia -c conda-forge \
    -c defaults rapids=0.14 python=3.6
```

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

# 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: RAPIDS Docs

## Improved and Easier to Use!

The screenshot shows the cuDF documentation page for "10 Minutes to cuDF and Dask-cuDF". The page includes a sidebar with navigation links for various RAPIDS libraries like clx, cudf, cugraph, cuml, cusignal, cuspatial, cufilter, libcudf, libcuml, libnvstrings, nvstrings, and rmm. The main content area features a heading "10 Minutes to cuDF and Dask-cuDF" and a sub-section "What are these Libraries?". It also contains code snippets and sections on "When to use cuDF and Dask-cuDF" and "Object Creation". A "View page source" link is at the top right.

<https://docs.rapids.ai>

The screenshot shows the cuDF documentation homepage. The sidebar indicates the version is "stable (0.13)". The main content area features a heading "Welcome to cuDF's documentation!" and a "Contents" section with a hierarchical list of topics such as API Reference, 10 Minutes to cuDF and Dask-cuDF, Multi-GPU with Dask-cuDF, 10 Minutes to Dask-XGBoost, 10 Minutes to cuDF and CuPy, Overview of User Defined Functions with cuDF, Developer Documentation, and more. A "Search docs" input field is at the top left, and a "View page source" link is at the top right.

<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

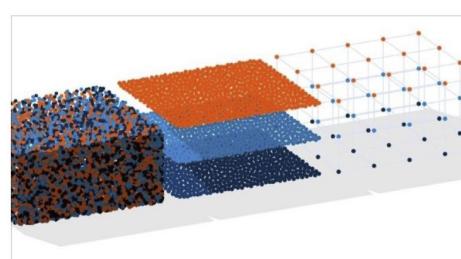
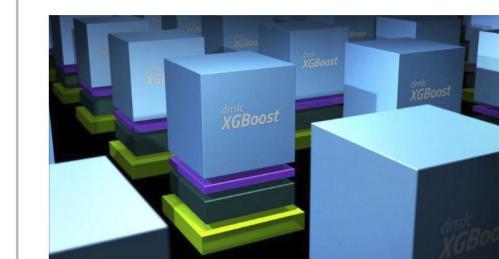


**gQuant—GPU Accelerated examples for Quantitative Analyst Tasks**

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

Yi Dong Jul 16 · 6 min read ★

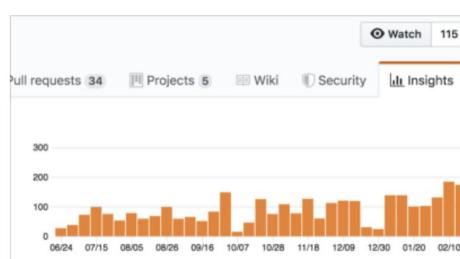


**NVIDIA GPUs and Apache**

**When Less is More: A brief**

**Nightly News: CI produces**

Jiwei Liu Jul 3 · 5 min read



Watch 115

Pull requests 34 Projects 5 Wiki Security Insights

0 100 200 300

06/24 07/15 08/05 08/26 09/16 10/07 10/28 11/18 12/09 01/20 02/10

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

# 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.

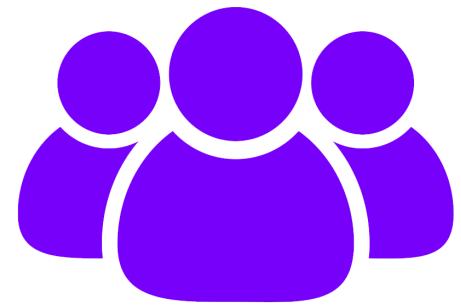
**Intermediate Notebooks:**

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 name is "RAPIDS" with 47 subscribers. The "HOME" tab is selected. Below it, there's a section titled "Uploads" with a "PLAY ALL" button. Four video thumbnails are listed, each with a red dashed box highlighting the thumbnail area. The videos are: "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), and a thumbnail for "Walmart uses RAPIDS" which is partially cut off.

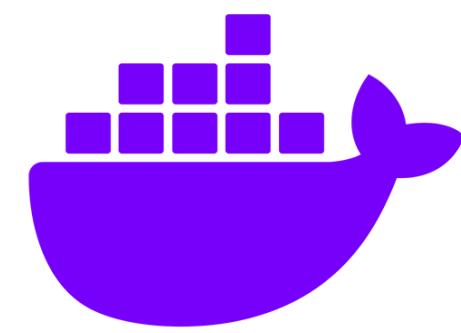
Notebooks Contrib Repo has tutorials and examples, and various E2E demos.  
RAPIDS Youtube channel has explanations, code walkthroughs and use cases.

# Join the Conversation



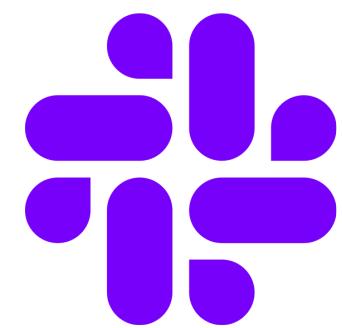
## GOOGLE GROUPS

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



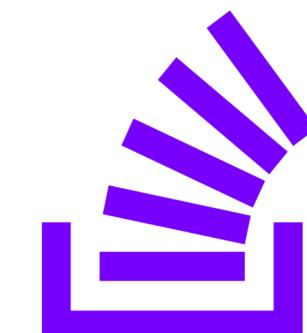
## DOCKER HUB

<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

# Getting Started

# RAPIDS Docs

## New, Improved, and Easier to Use

The image shows two screenshots of the RAPIDS Docs website. The left screenshot displays the '10 Minutes to cuDF and Dask-cuDF' page. The right screenshot displays the 'Welcome to cuDF's documentation!' page.

**Left Screenshot: 10 Minutes to cuDF and Dask-cuDF**

- Header:** Docs > 10 Minutes to cuDF and Dask-cuDF | View page source
- Sidebar:** Home, cudf (selected), clx, cugraph, cuml, cusignal, cuspatial, cufilter, libcudf, libcuml, libnvstrings, nvstrings, rmm, with cuDF.
- Content:**
  - 10 Minutes to cuDF and Dask-cuDF**

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

**What are these Libraries?**

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

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

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

**When to use cuDF and Dask-cuDF**

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

```
[1]: import os  
import numpy as np  
import pandas as pd  
import cudf  
import dask_cudf  
  
np.random.seed(12)  
  
### Portions of this were borrowed and adapted from the  
### cuDF cheatsheet, existing cuDF documentation,  
### and 10 Minutes to Pandas.
```

**Object Creation**

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

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

<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, cufilter, 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 includes sections on 'What are these Libraries?', 'When to use cuDF and Dask-cuDF', and 'Object Creation'. A code block shows Python imports for os, numpy, pandas, cuDF, and dask\_cudf, followed by a portion of a cuDF cheatsheet. Below that, there's a code block for creating a cuDF Series.

```
[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
```

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

<https://docs.rapids.ai>

# RAPIDS

## How Do I Get the Software?



GITHUB

<https://github.com/rapidsai>



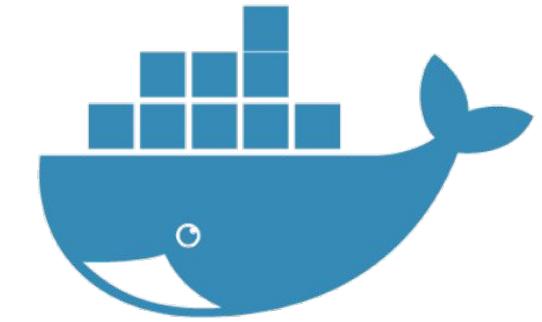
ANACONDA

<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

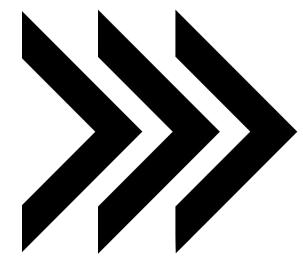


Kubeflow

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

RAPIDS

<https://rapids.ai>

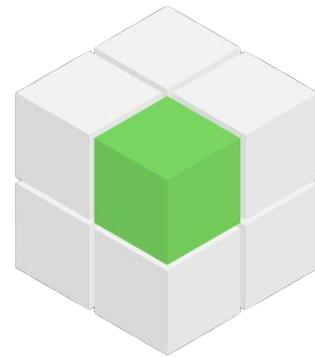
@RAPIDSAI



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

Joshua Patterson  
[joshuap@nvidia.com](mailto:joshuap@nvidia.com)

 @datametrician

# RAPIDS