

RAPIDS

The Platform Inside and Out Release 0.16

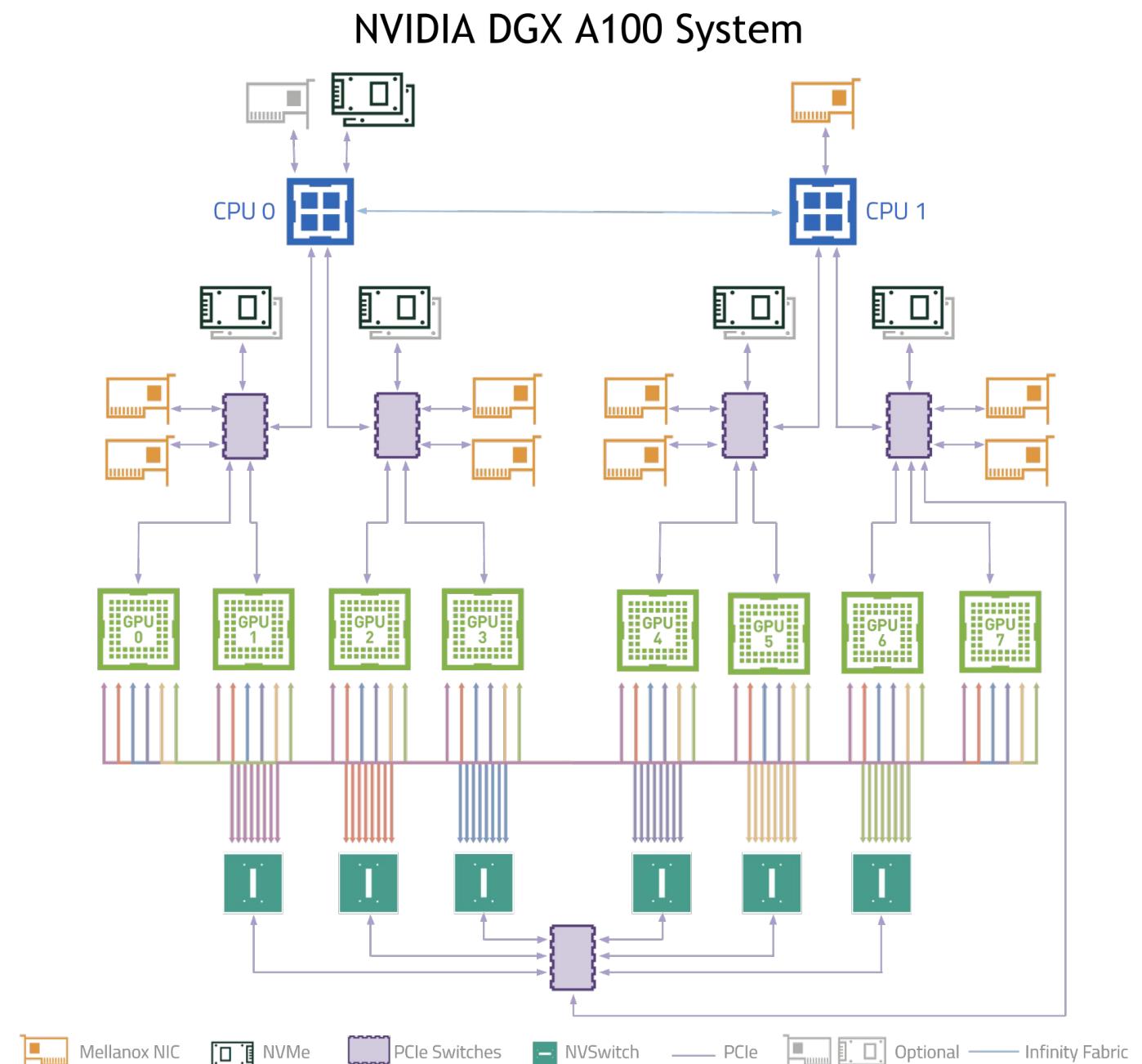
Joshua Patterson - Senior Director, RAPIDS Engineering

Why GPUs?

Numerous hardware advantages

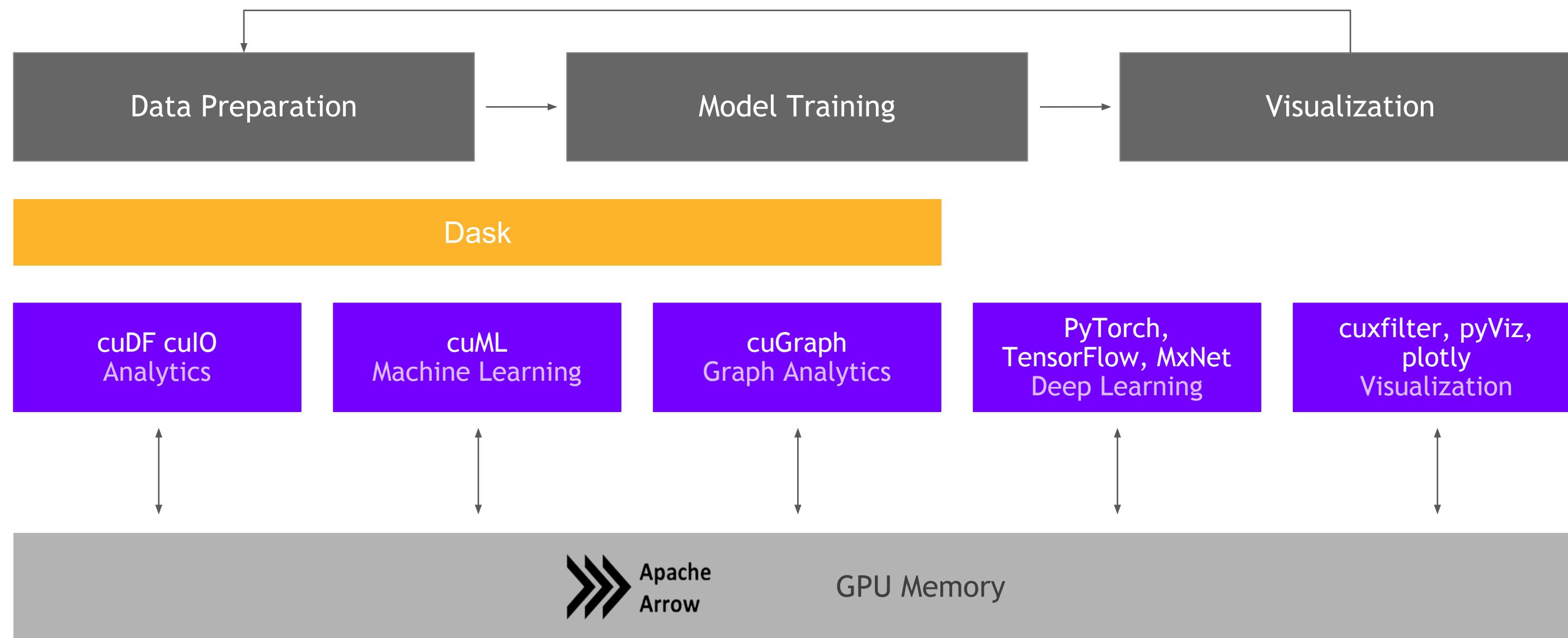
- ▶ Thousands of cores with up to ~20 TeraFlops of general purpose compute performance
- ▶ Up to 1.5 TB/s of memory bandwidth
- ▶ Hardware interconnects for up to 600 GB/s bidirectional GPU <---> GPU bandwidth
- ▶ Can scale up to 16x GPUs in a single node

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

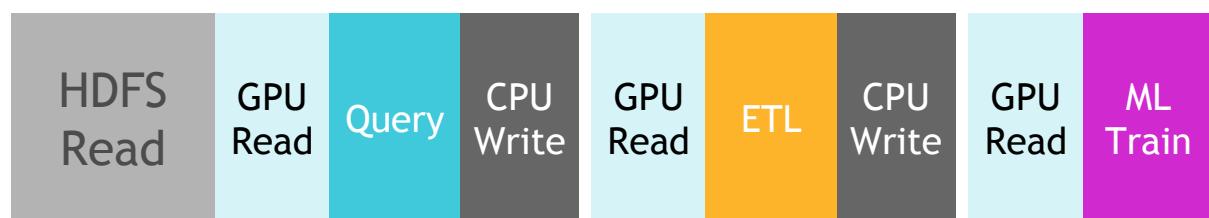
Hadoop Processing, Reading from Disk



Spark In-Memory Processing



Traditional GPU Processing

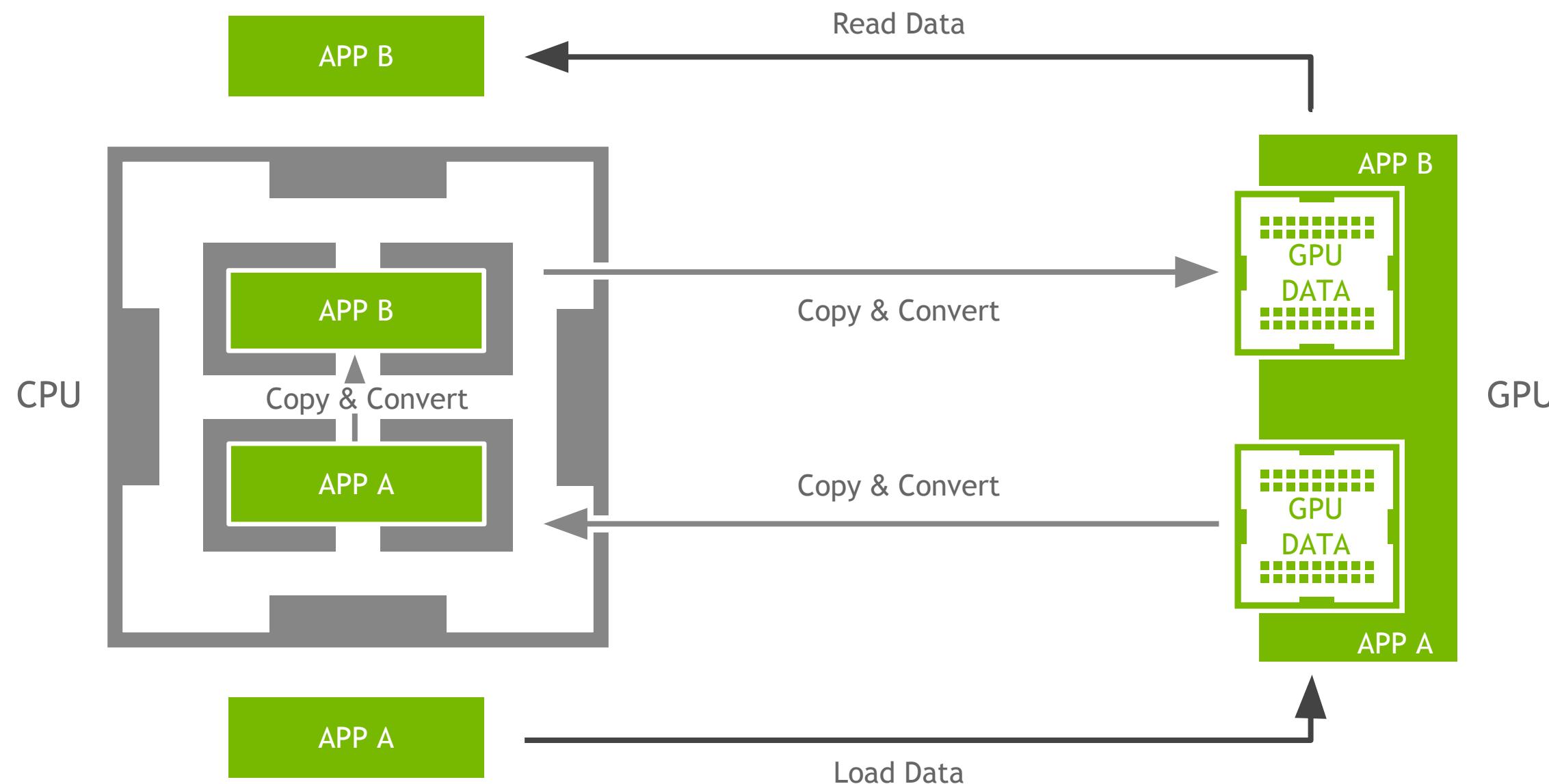


25-100x Improvement
Less Code
Language Flexible
Primarily In-Memory

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

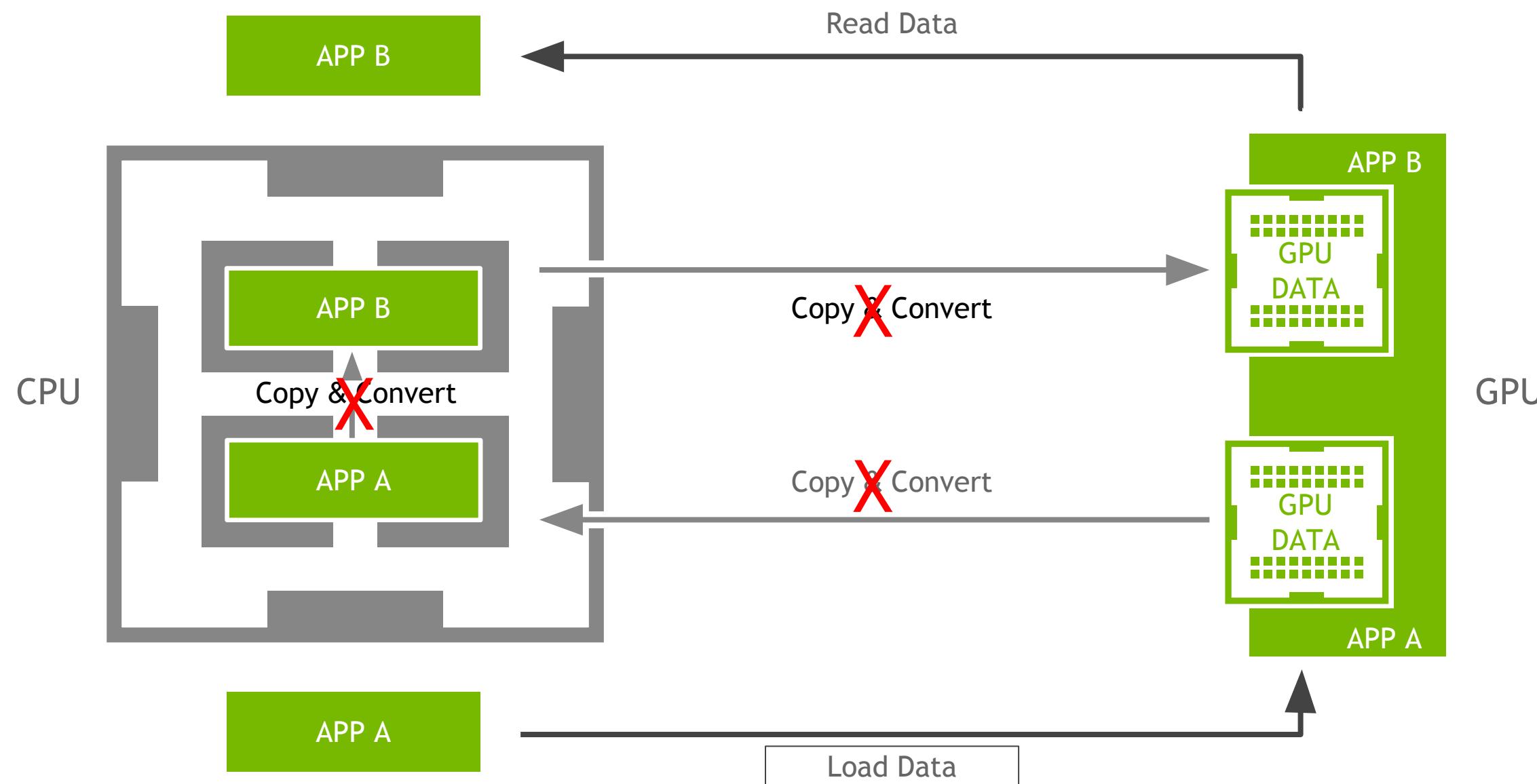
Data Movement and Transformation

The Bane of Productivity and Performance

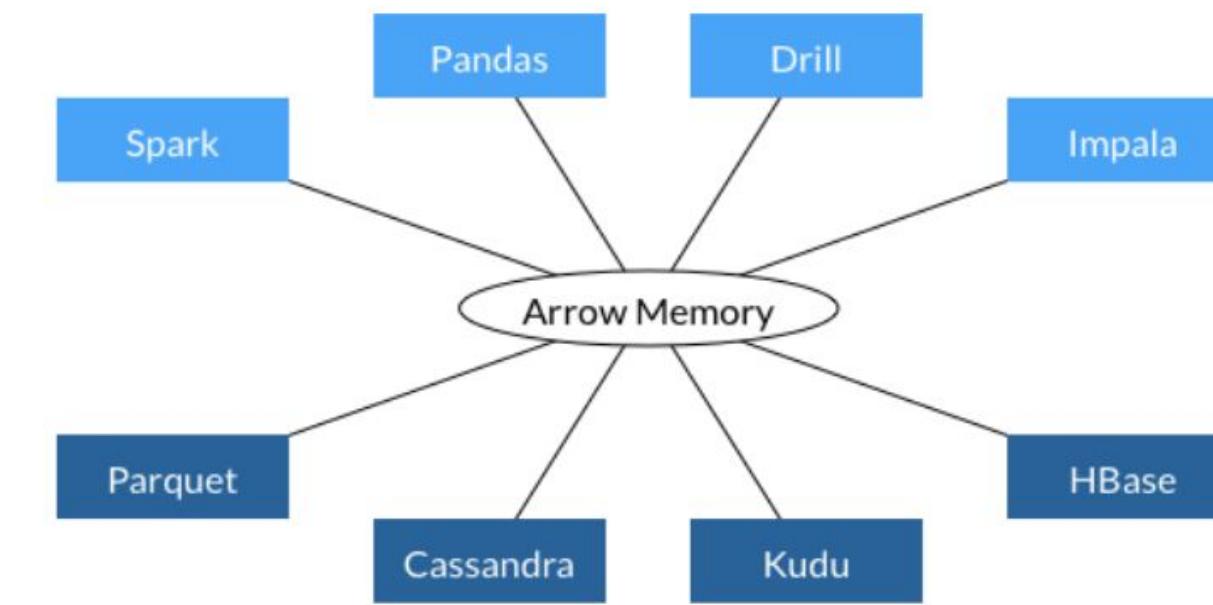
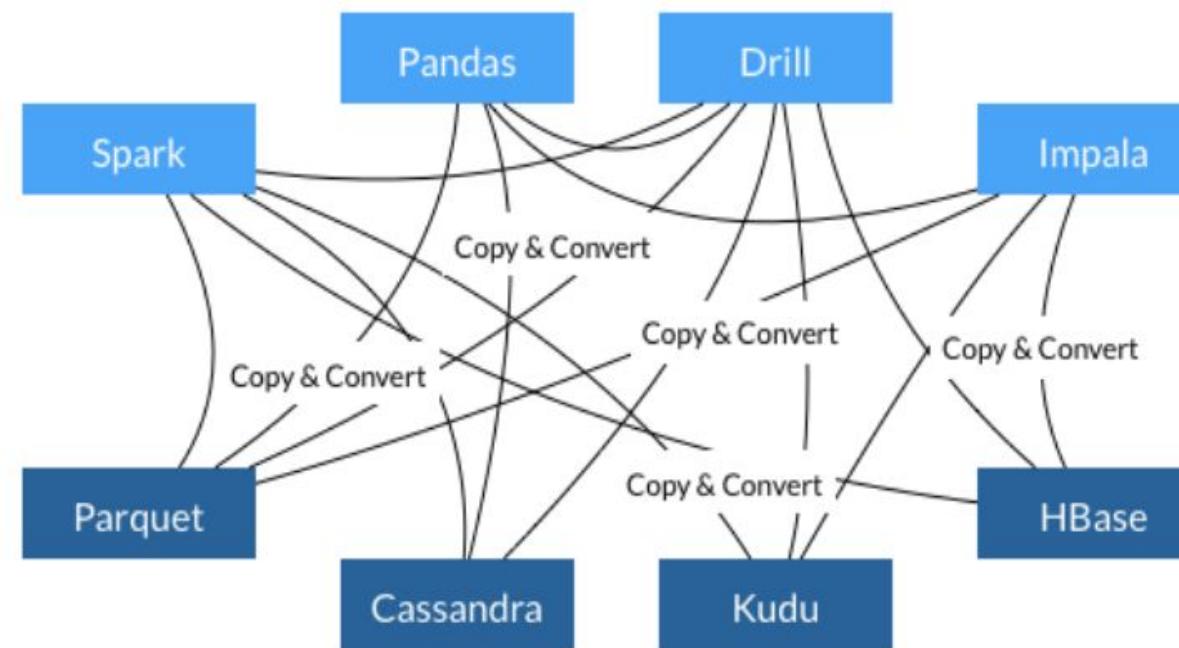


Data Movement and Transformation

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)

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

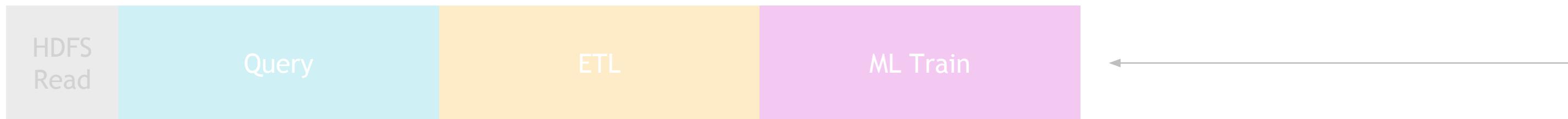
Data Processing Evolution

Faster Data Access, Less Data Movement

Hadoop Processing, Reading from Disk



Spark In-Memory Processing



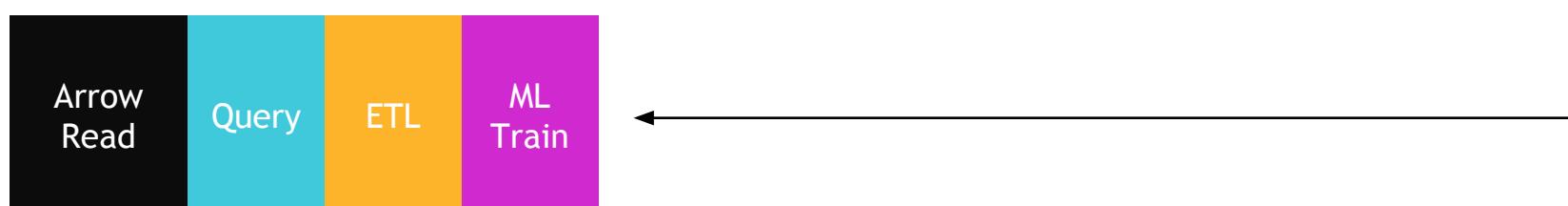
25-100x Improvement
Less Code
Language Flexible
Primarily In-Memory

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

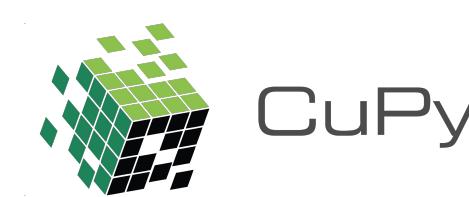
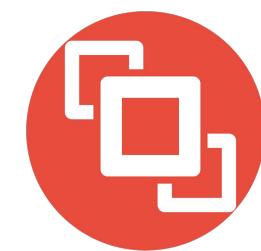
Up to 350x faster queries; Hours to Seconds!

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”.

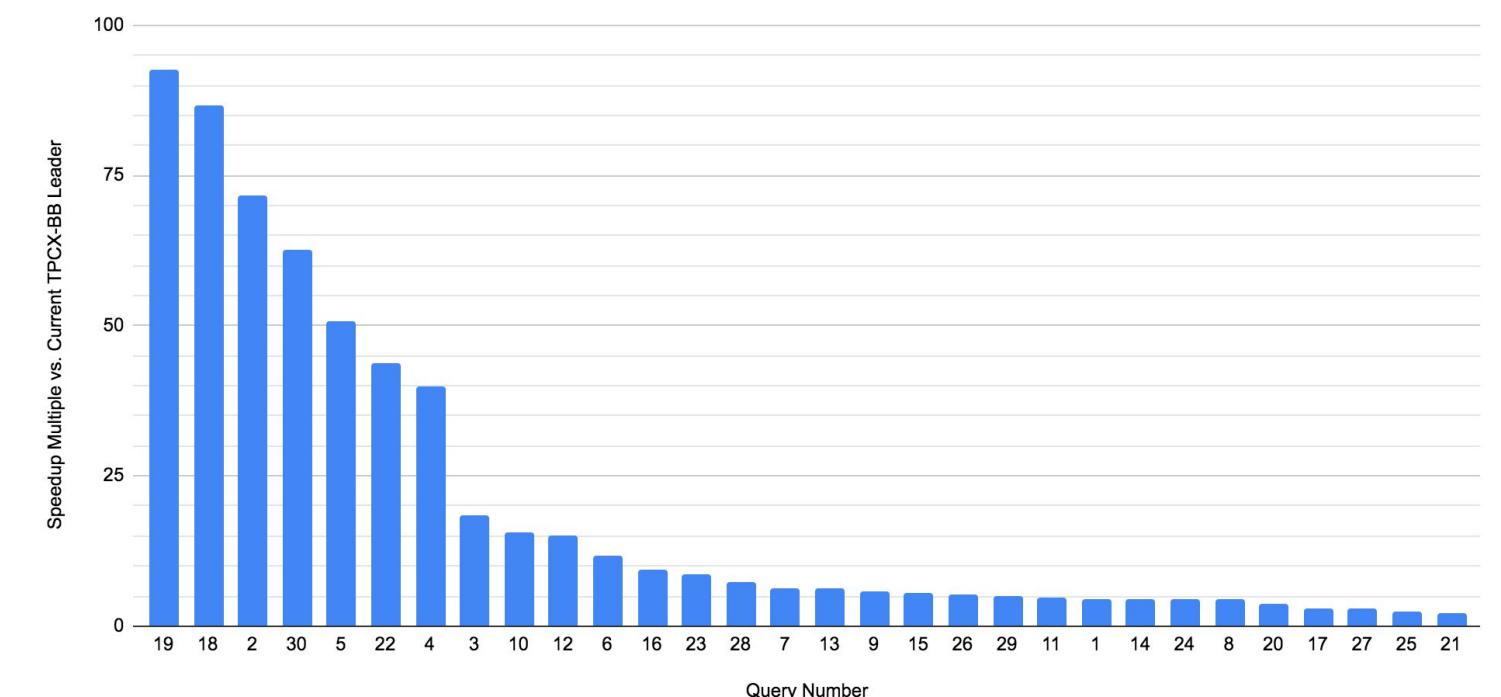
- ▶ SF1 - 1GB
- ▶ SF1K - 1 TB
- ▶ SF10K - 10 TB

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

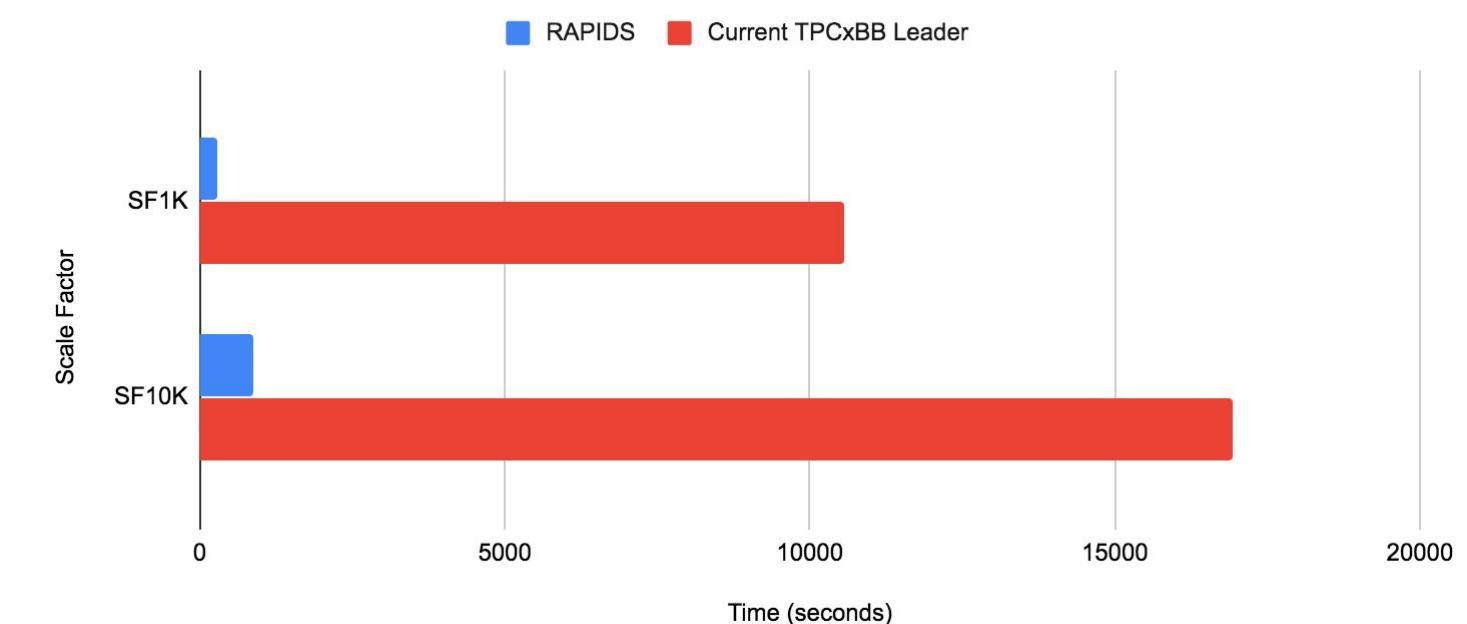
- ▶ SF1K 37.1x average speed-up
- ▶ SF10K 19.5x average speed-up (7x Normalized for Cost)



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.

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
Sci-kit Learn	~80
Keras	~65
LightGBM	~55
XGBoost	~55
PyTorch	~30
TensorFlow (non-Keras)	~25
Caffe	~10
MXNet	~10
Fastai	~10
Caffe2	~10
CatBoost	~10
R Random Forest	~10

0 20 40 60 80
Deep Classic

kaggle

RAPIDS 0.16 Release Summary

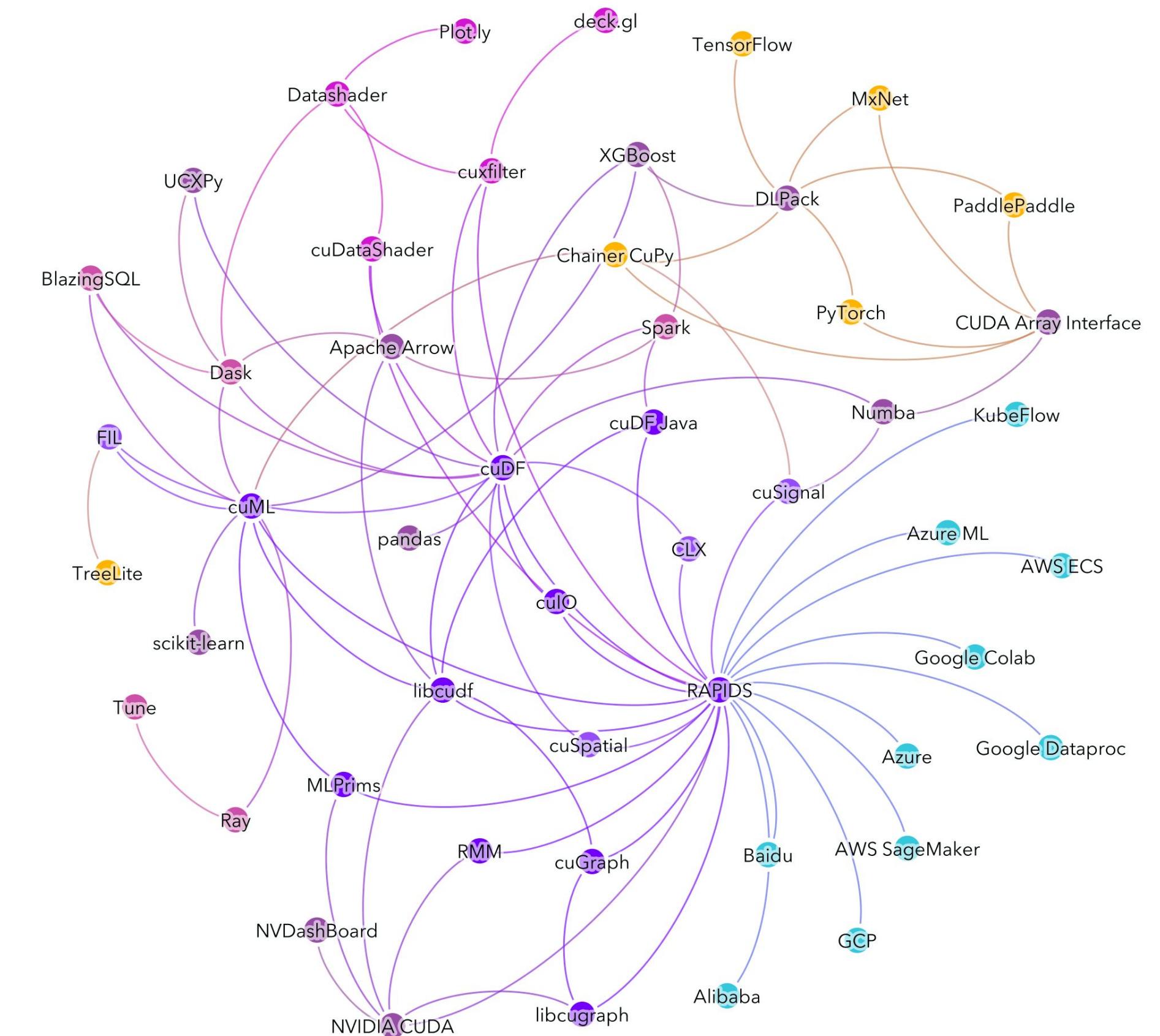
What's New in Release 0.16?

- ▶ cuDF adds initial Struct column support, Dataframe.pivot() and DataFrame.unstack() functions, Groupby.collect() aggregation support, dayofweek datetime function, and custom dataframe accessors.
- ▶ cuML machine learning library adds a large suite of experimental, scikit-learn compatible preprocessors (such as SimpleImputer, Normalizer, and more), Dask support for tf-IDF and label encoding, and Porter stemming
- ▶ cuGraph adds multi-node multi-GPU versions of PageRank, BFS, SSSP, and Louvain. This removed the old 2 billion vertex limits. Also adds support for NetworkX Graphs as input
- ▶ UCX-Py adds more Cython optimizations including strongly typed objects/arrays, new interfaces for Fence/Flush, and documentation clean-up/updates, including a new debugging page.
- ▶ BlazingSQL now supports out-of-core query execution, which enables queries to operate on datasets dramatically larger than available GPU memory.
- ▶ cuSpatial adds Java bindings and Jar.
- ▶ cuXfilter adds large scale graph visualization with datashader, lasso select with cuSpatial, and instructions on how to run dashboards as stand-alone applications.
- ▶ RMM debug logging, New arena memory resource and limiting resource, CMake improvements.

RAPIDS Everywhere

The Next Phase of RAPIDS

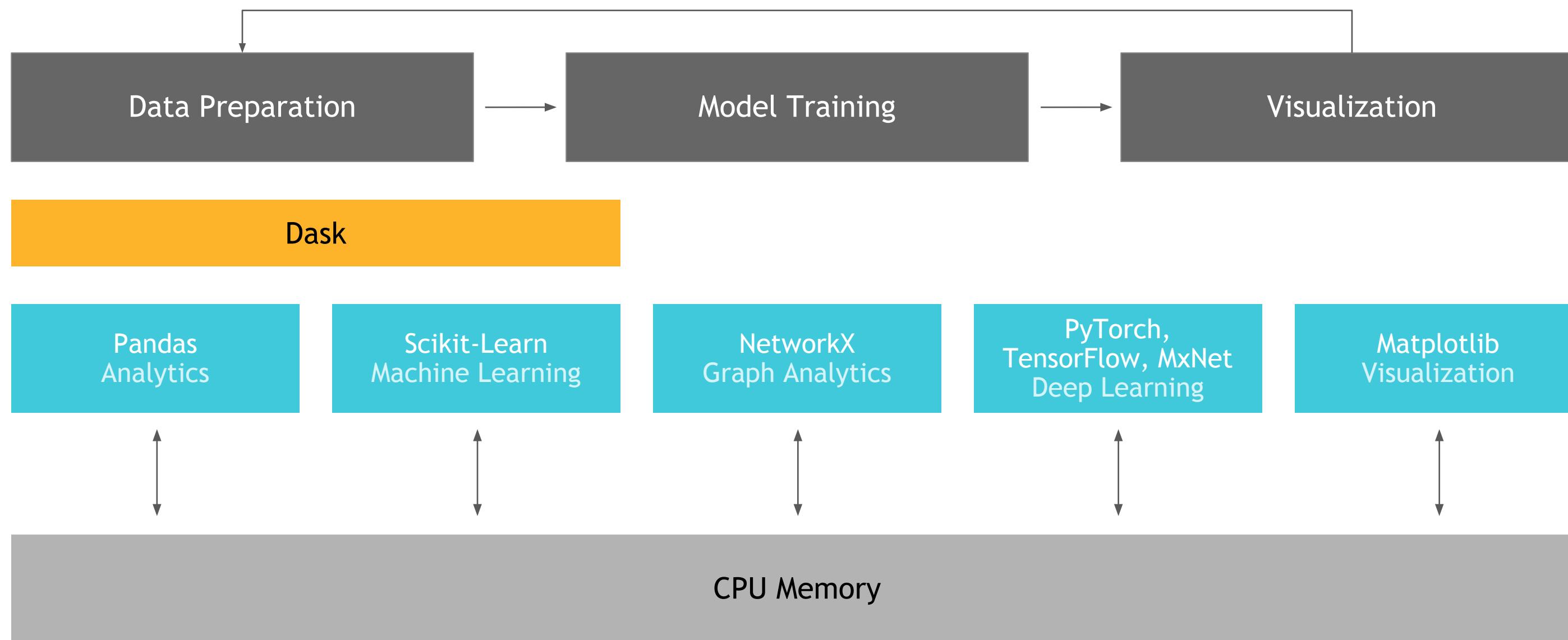
Exactly as it sounds—our goal is to make RAPIDS as usable and performant as possible wherever data science is done. We will continue to work with more open source projects to further democratize acceleration and efficiency in data science.



RAPIDS Core

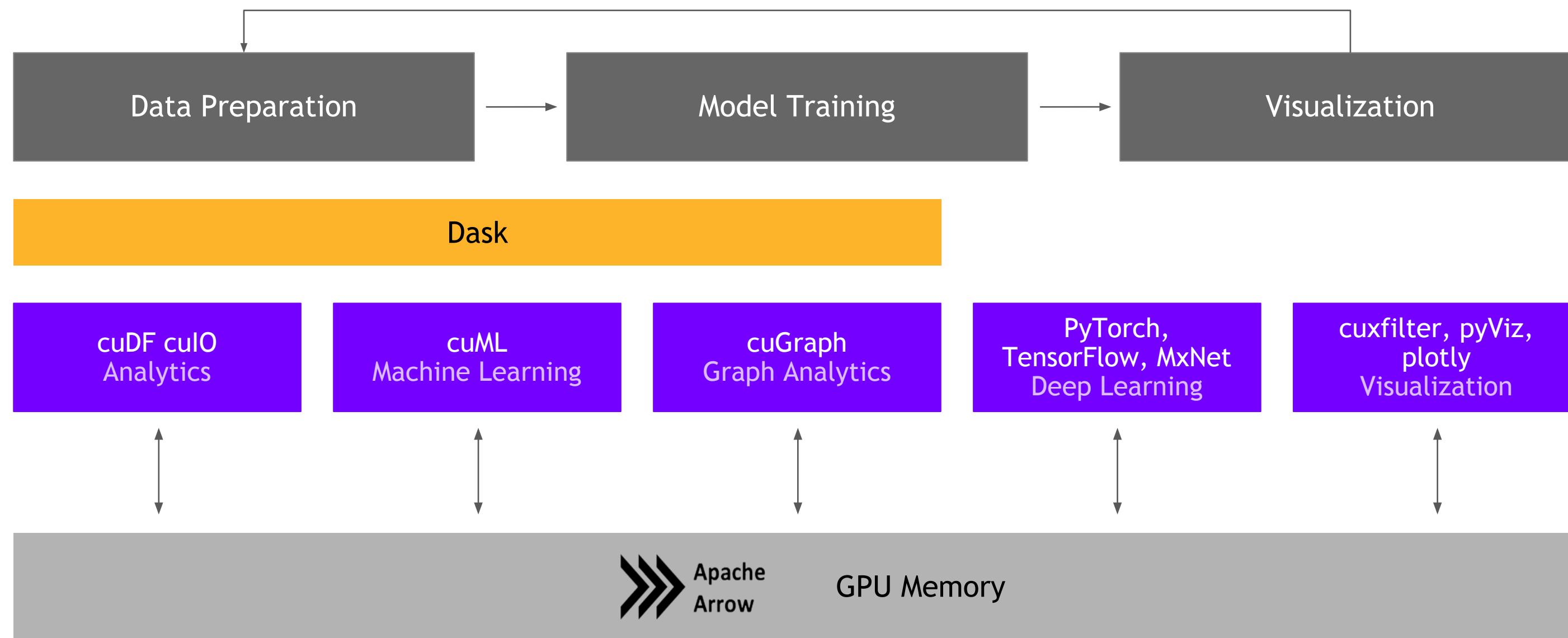
Open Source Data Science Ecosystem

Familiar Python APIs



RAPIDS

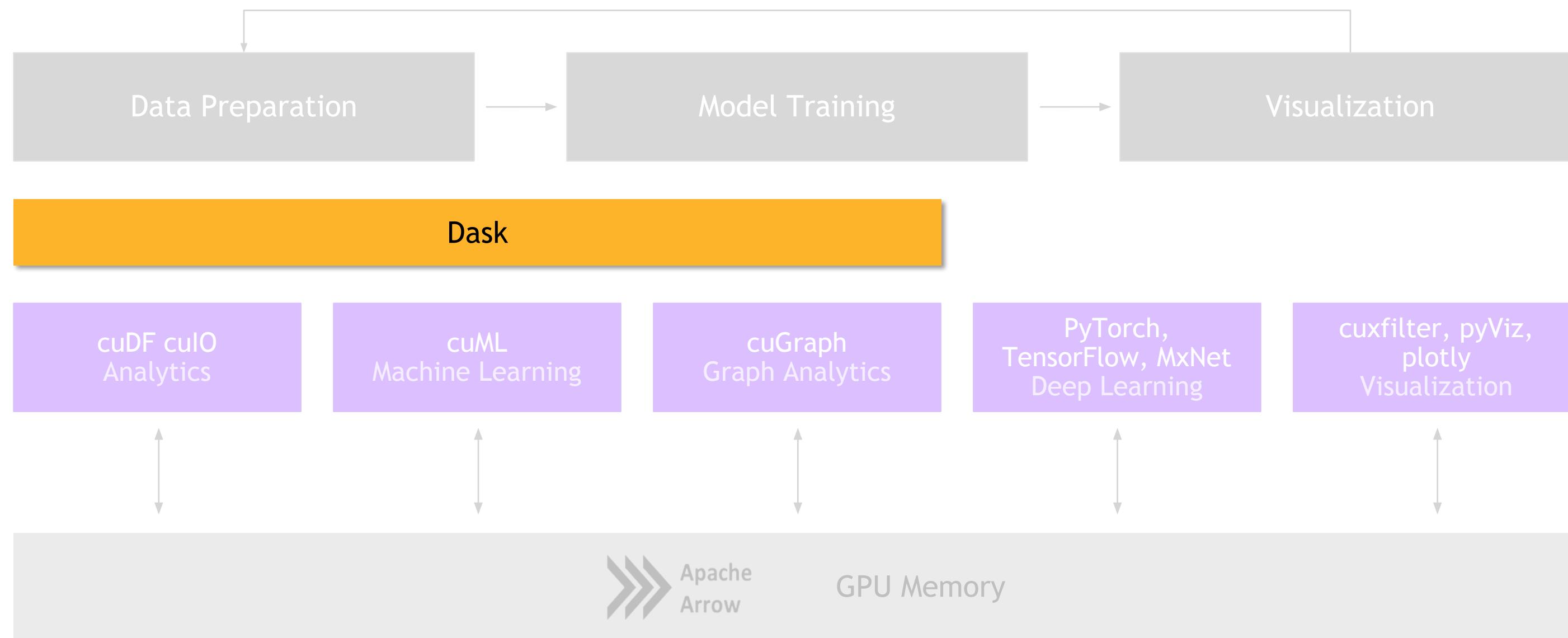
End-to-End Accelerated GPU Data Science



Dask

RAPIDS

Scaling RAPIDS with Dask



Why Dask?

DEPLOYABLE

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

PYDATA NATIVE

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

EASY SCALABILITY

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

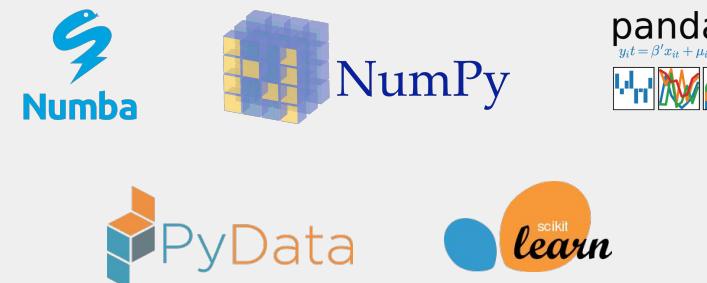
POPULAR

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

PYDATA

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

Single CPU core
In-memory data



DASK

Multi-core and distributed PyData

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



Scale Out / Parallelize

Why OpenUCX?

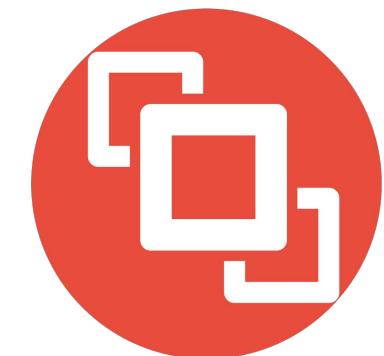
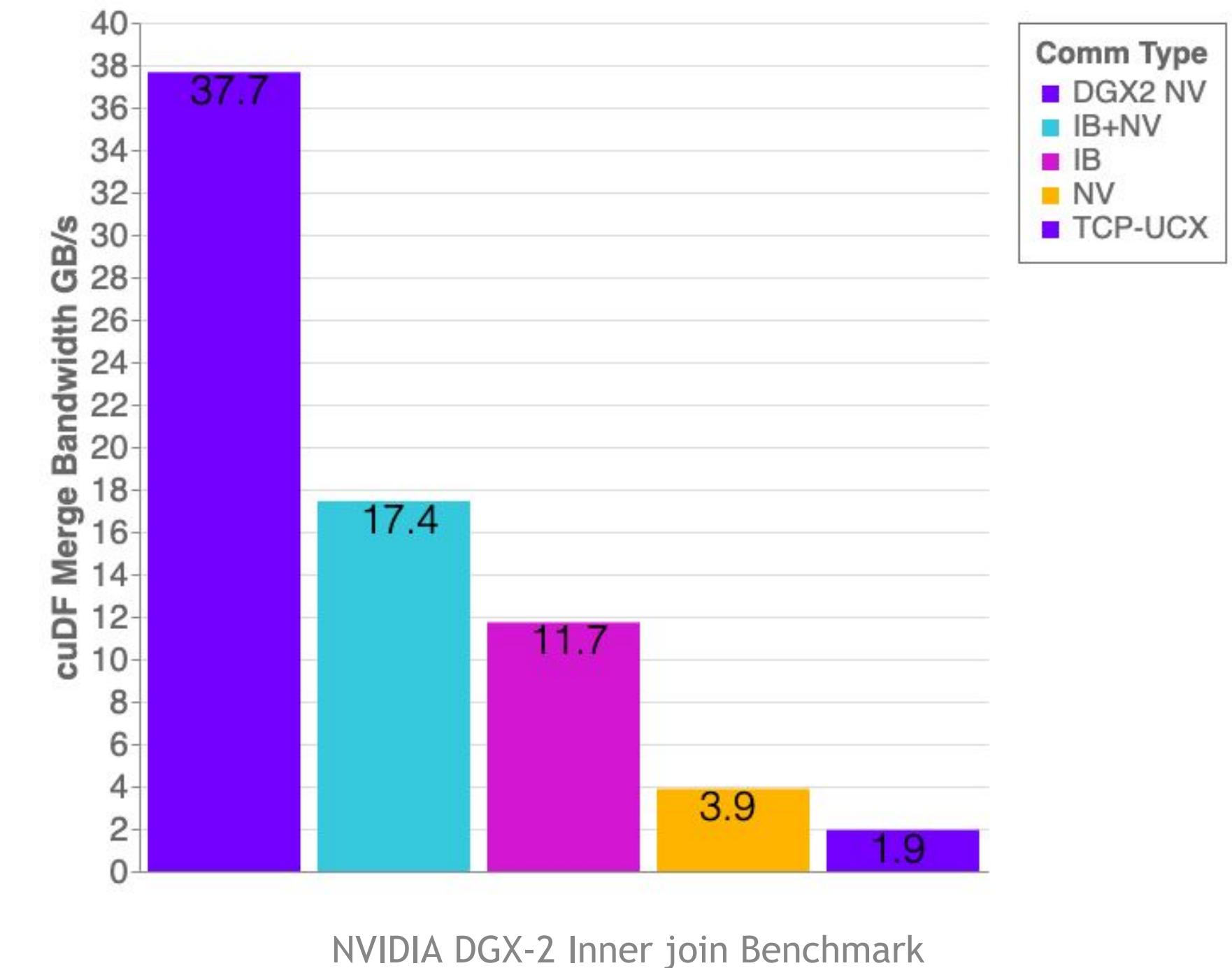
Bringing Hardware Accelerated Communications to Dask

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

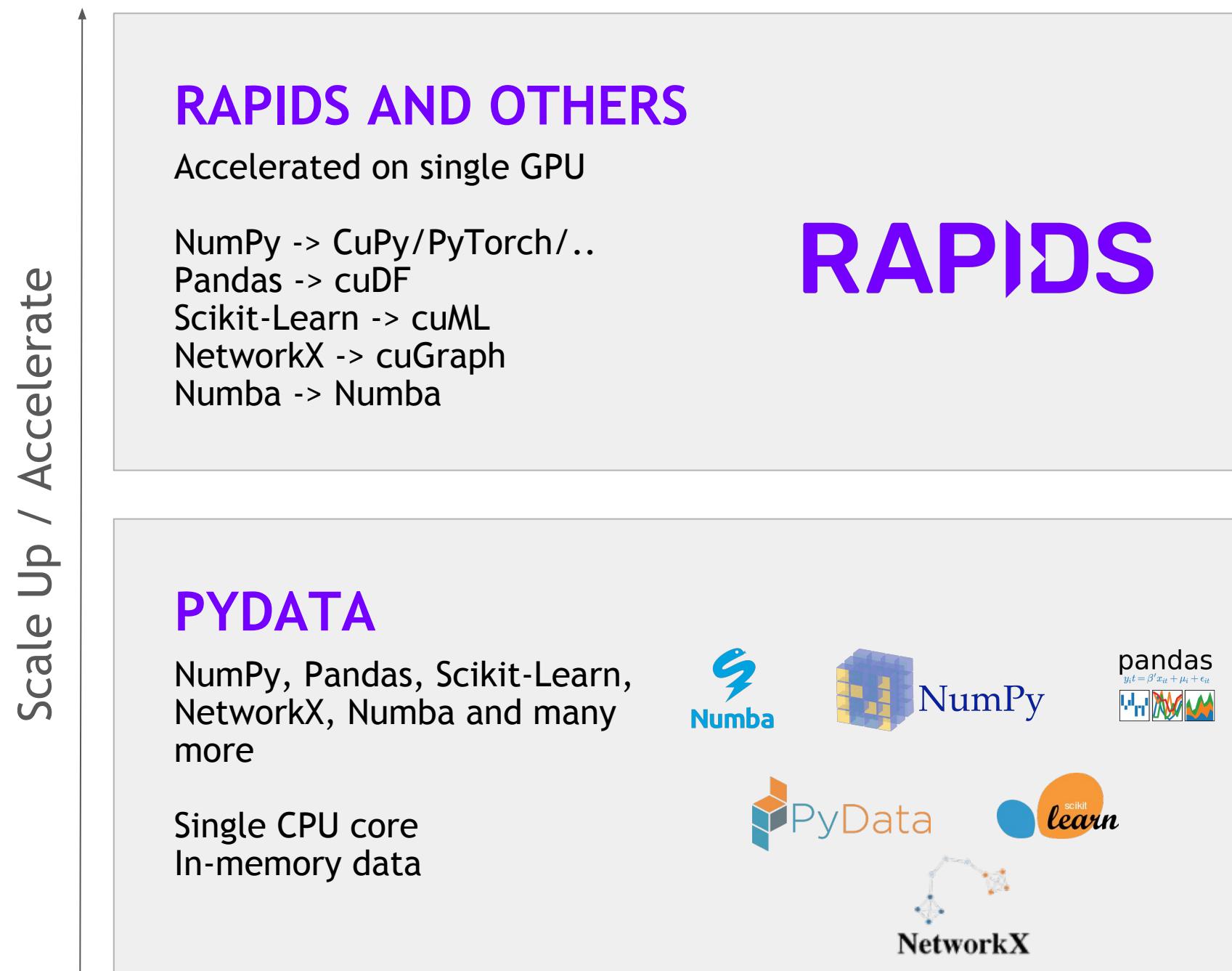
```
conda install -c conda-forge -c rapidsai \
    cudatoolkit=<CUDA version> ucx-proc=*gpu ucx ucx-py

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

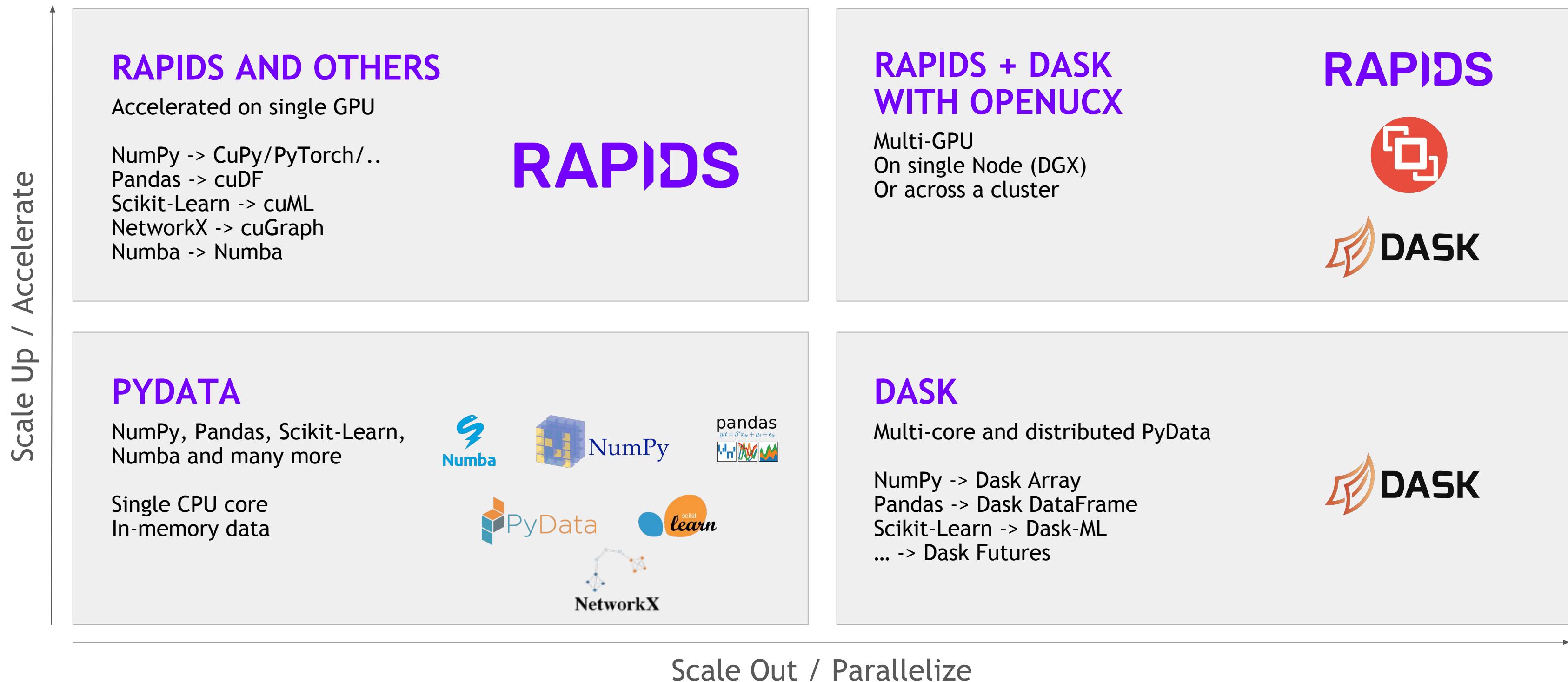
client = Client(cluster)
```



Scale Up with RAPIDS



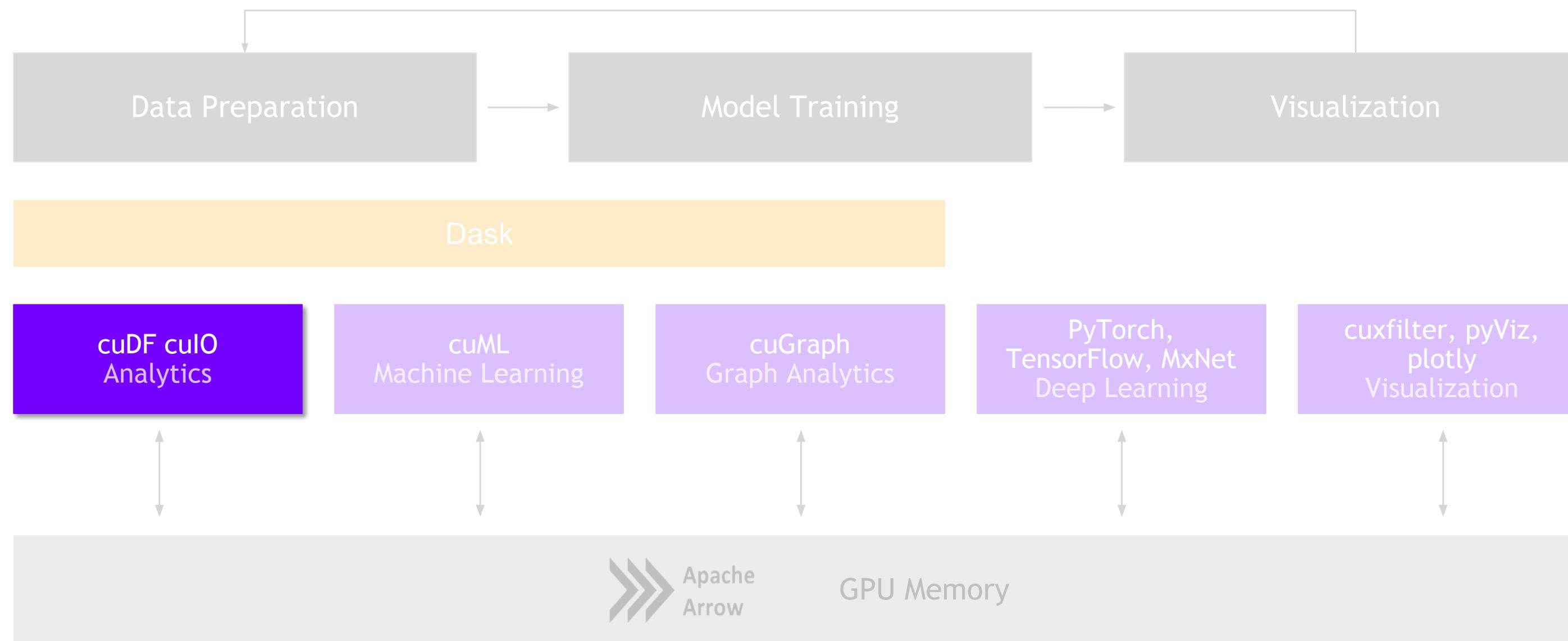
Scale Out with RAPIDS + Dask with OpenUCX



cuDF

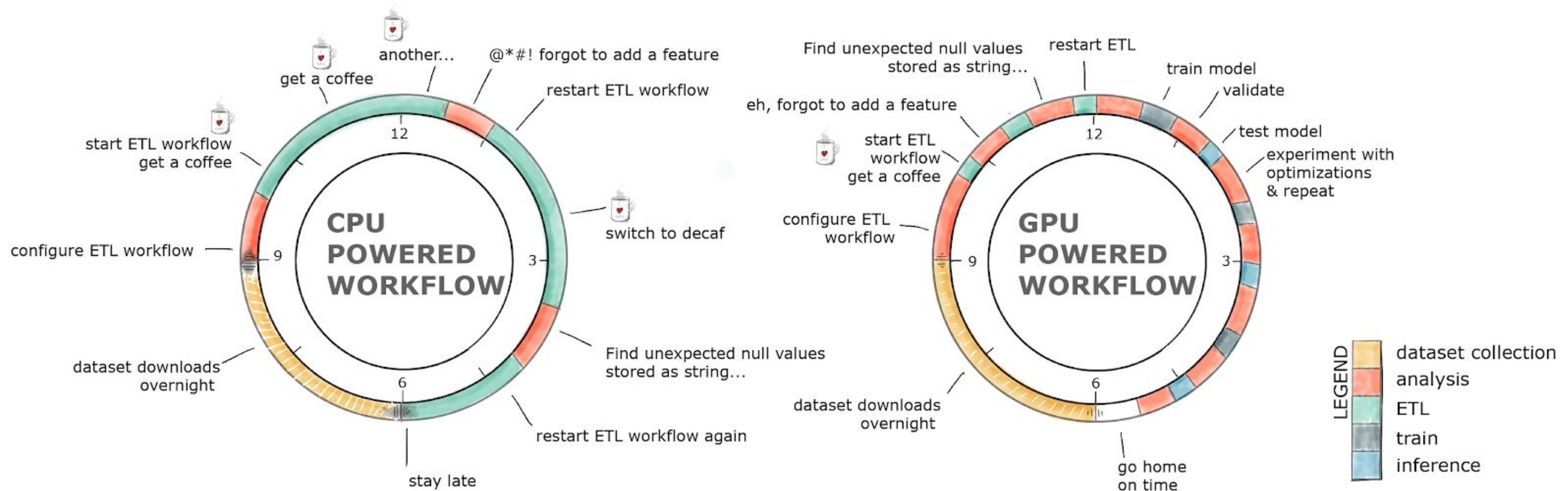
RAPIDS

GPU Accelerated Data Wrangling and Feature Engineering

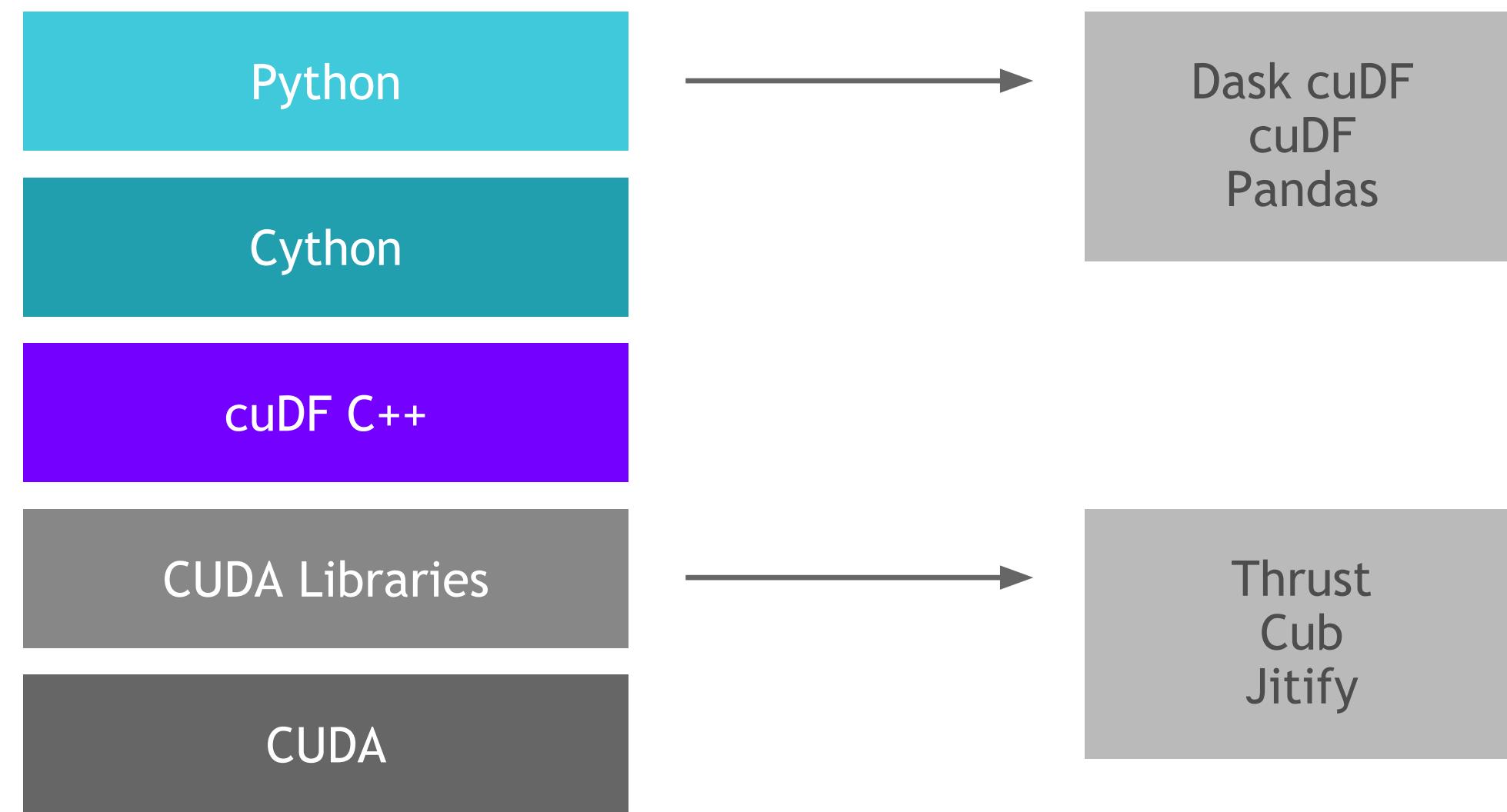


GPU-Accelerated ETL

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



ETL Technology Stack



ETL - the Backbone of Data Science

libcuDF is...

CUDA C++ LIBRARY

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

```
std::unique_ptr




```



ETL - the Backbone of Data Science

cuDF is...

PYTHON LIBRARY

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

```
In [2]: #Read in the data. Notice how it decompresses as it reads the data into memory.  
gdf = cudf.read_csv('/rapids/Data/black-friday.zip')  
  
In [3]: #Taking a look at the data. We use "to_pandas()" to get the pretty printing.  
gdf.head().to_pandas()  
  
Out[3]:
```

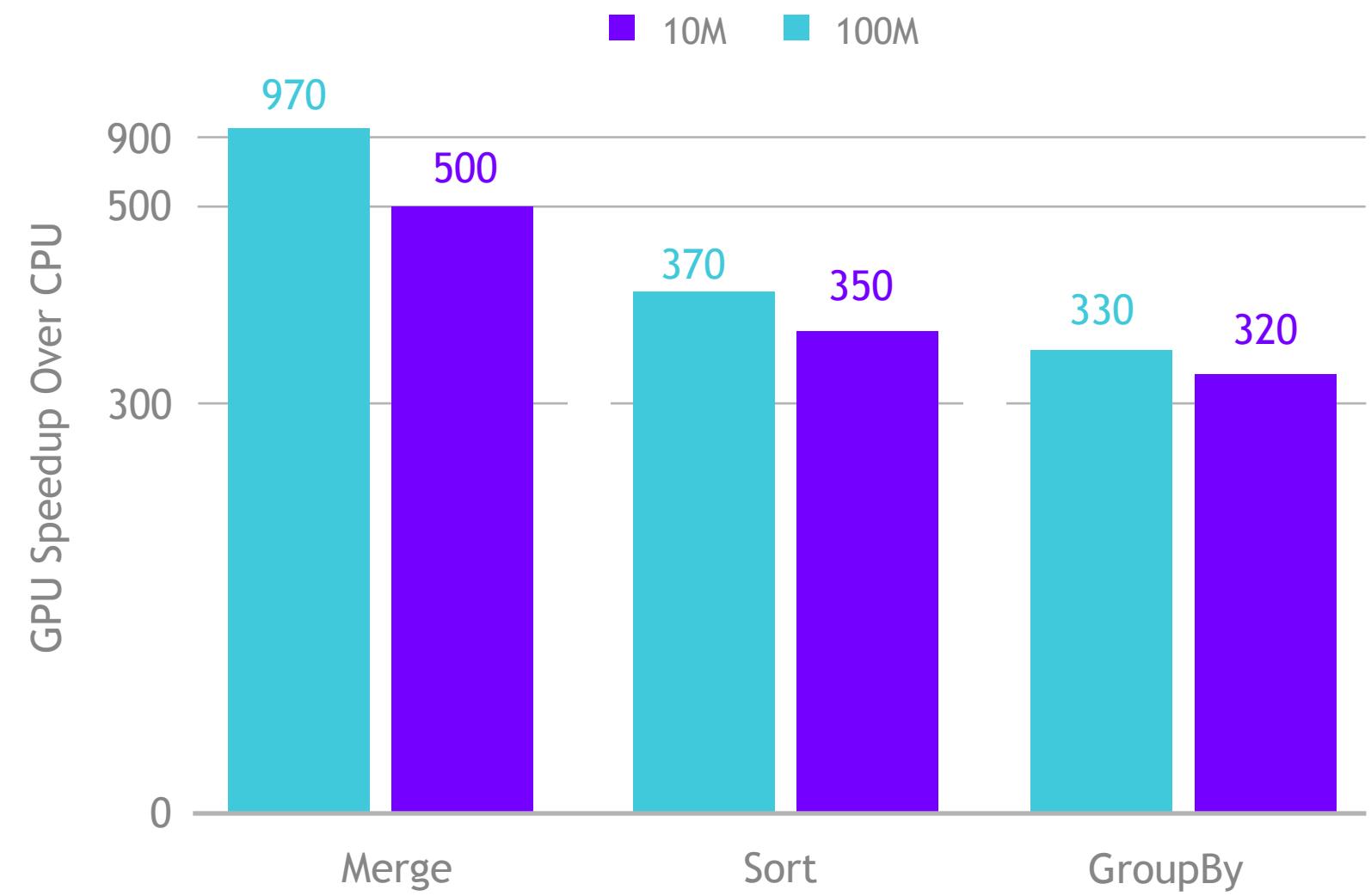
	User_ID	Product_ID	Gender	Age	Occupation	City_Category	Stay_In_Current_City_Years	Marital_Status	Product_Ca
0	1000001	P00069042	F	0-17	10	A	2	0	3
1	1000001	P00248942	F	0-17	10	A	2	0	1
2	1000001	P00087842	F	0-17	10	A	2	0	12
3	1000001	P00085442	F	0-17	10	A	2	0	12
4	1000002	P00285442	M	55+	16	C	4+	0	8


```
In [6]: #grabbing the first character of the years in city string to get rid of plus sign, and converting  
#to int  
gdf['city_years'] = gdf.Stay_In_Current_City_Years.str.get(0).stoi()  
  
In [7]: #Here we can see how we can control what the value of our dummies with the replace method and turn  
#strings to ints  
gdf['City_Category'] = gdf.City_Category.str.replace('A', '1')  
gdf['City_Category'] = gdf.City_Category.str.replace('B', '2')  
gdf['City_Category'] = gdf.City_Category.str.replace('C', '3')  
gdf['City_Category'] = gdf['City_Category'].str.stoi()
```

Benchmarks: Single-GPU Speedup vs. Pandas

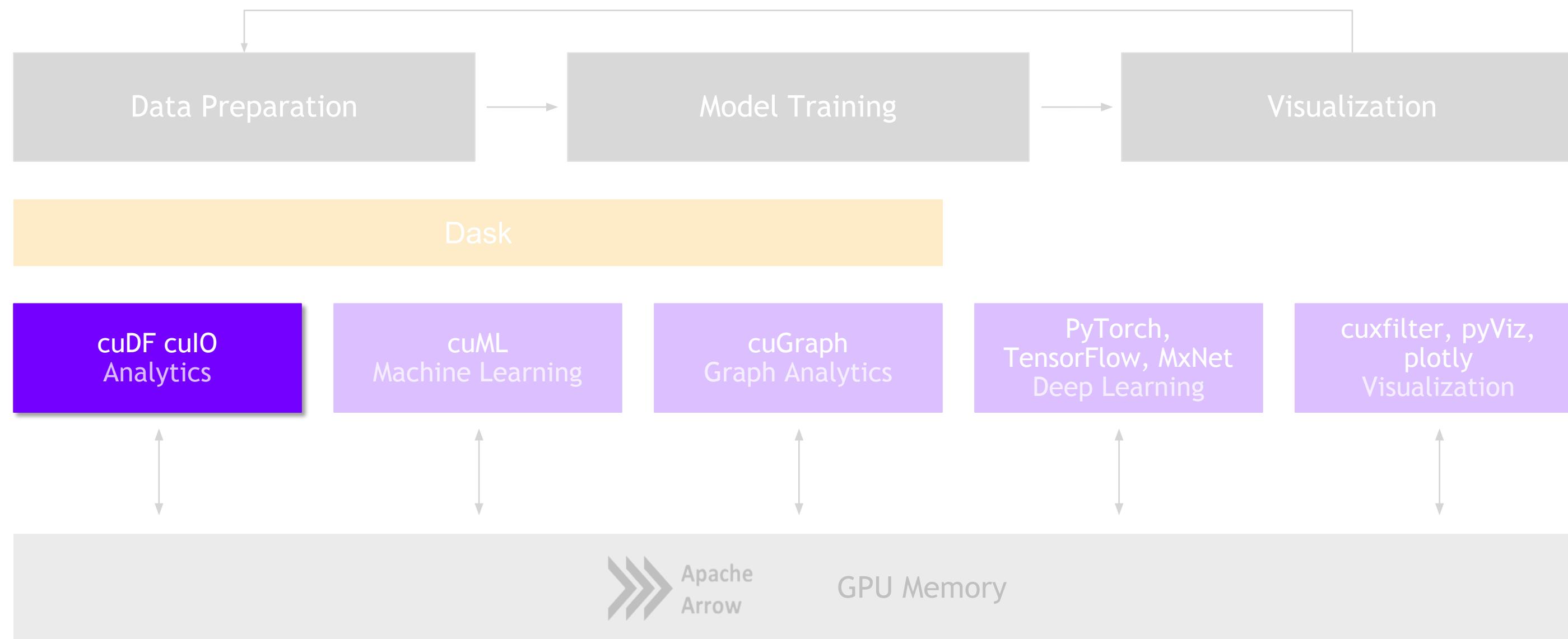
cuDF v0.13, Pandas 0.25.3

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



ETL - the Backbone of Data Science

cuDF is Not the End of the Story



ETL - the Backbone of Data Science

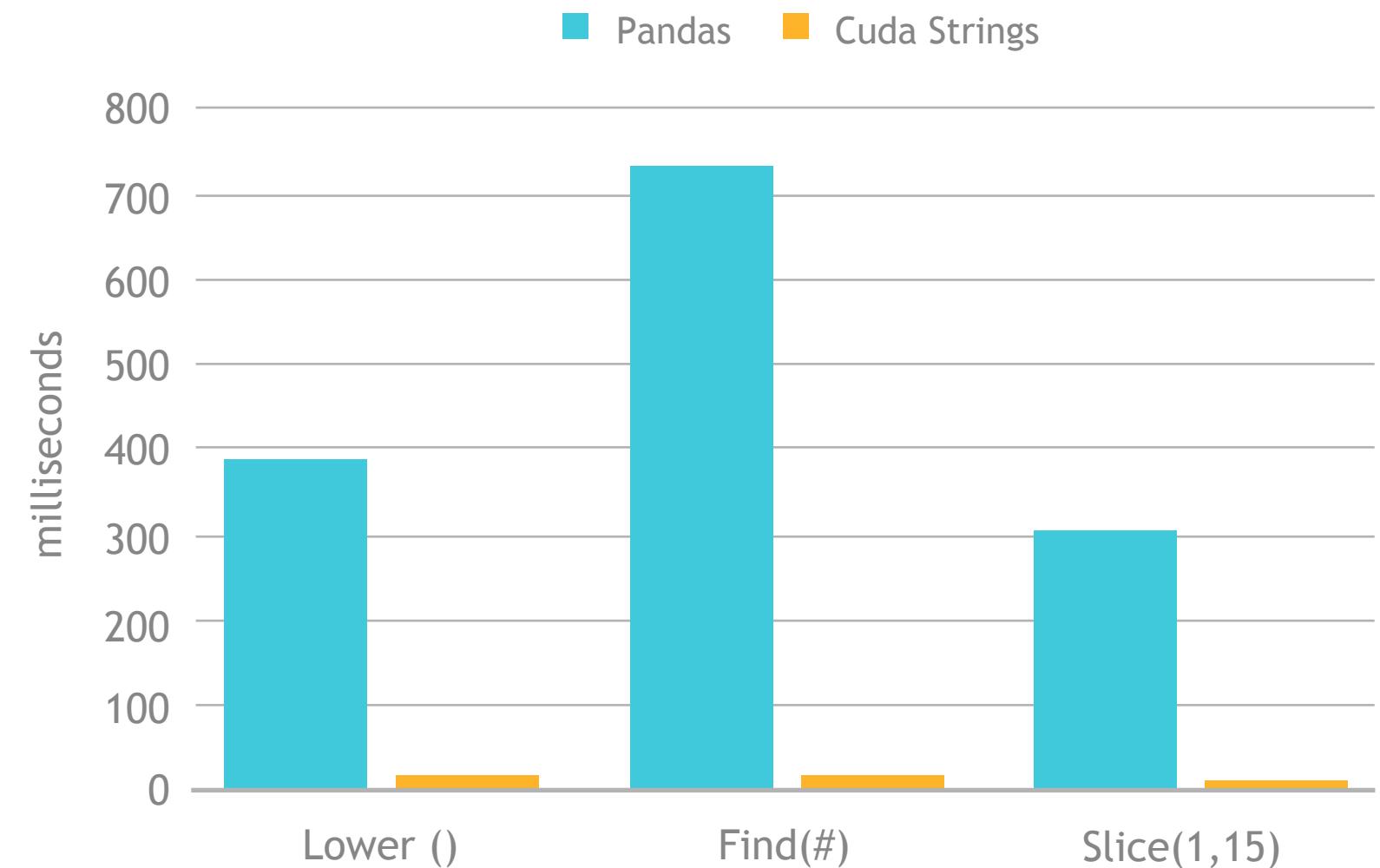
String Support

CURRENT V0.16 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 V0.17+ 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

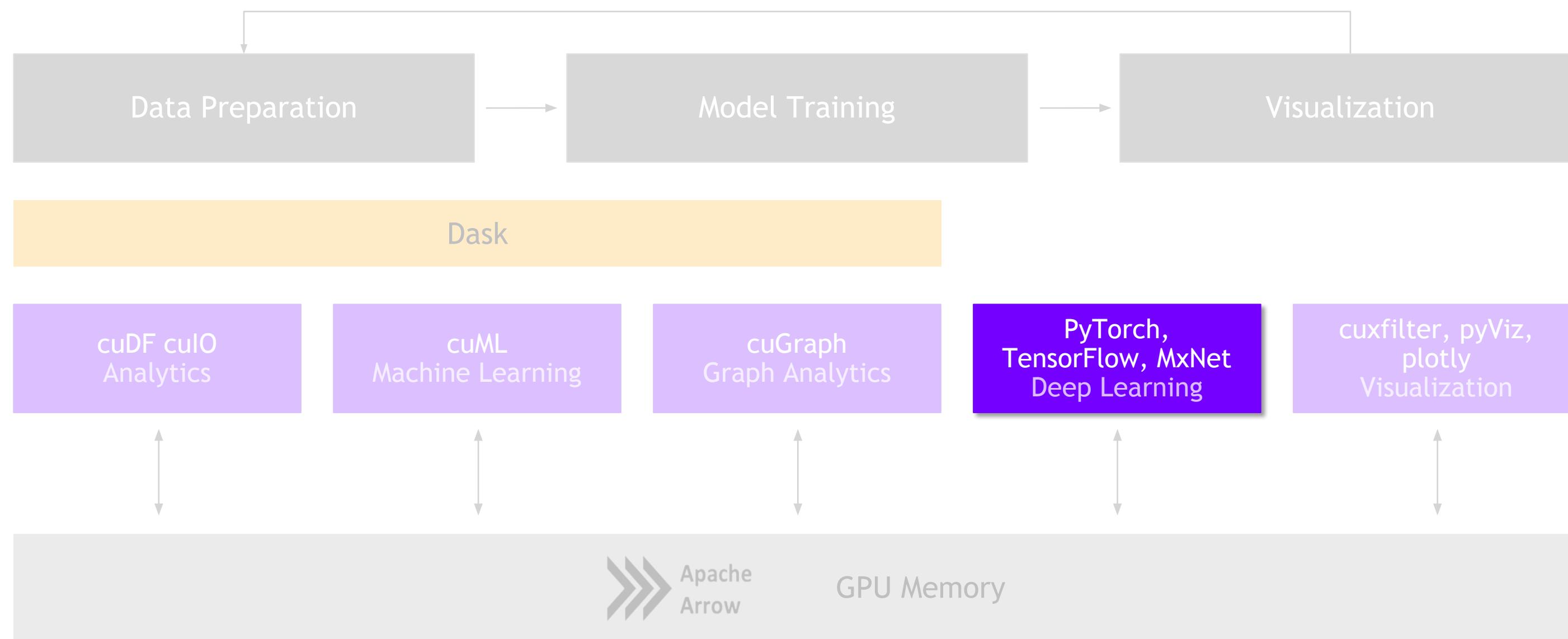
```
1]: import pandas, cudf
2]: %time len(pandas.read_csv('data/nyc/yellow_tripdata_2015-01.csv'))
CPU times: user 25.9 s, sys: 3.26 s, total: 29.2 s
Wall time: 29.2 s
2]: 12748986
3]: %time len(cudf.read_csv('data/nyc/yellow_tripdata_2015-01.csv'))
CPU times: user 1.59 s, sys: 372 ms, total: 1.96 s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

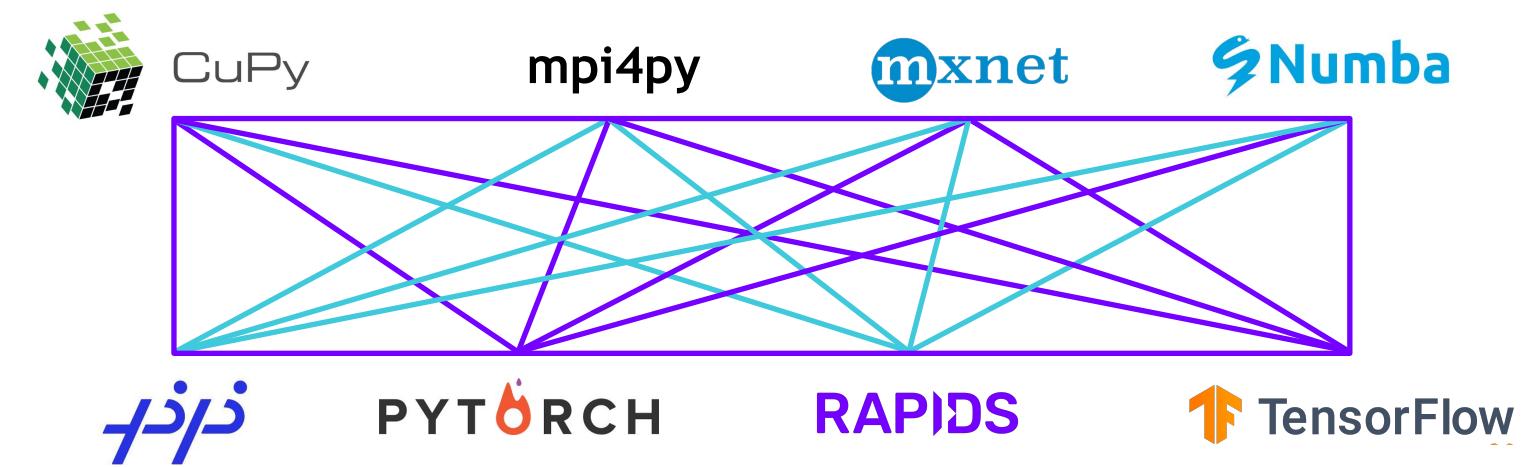
RAPIDS

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



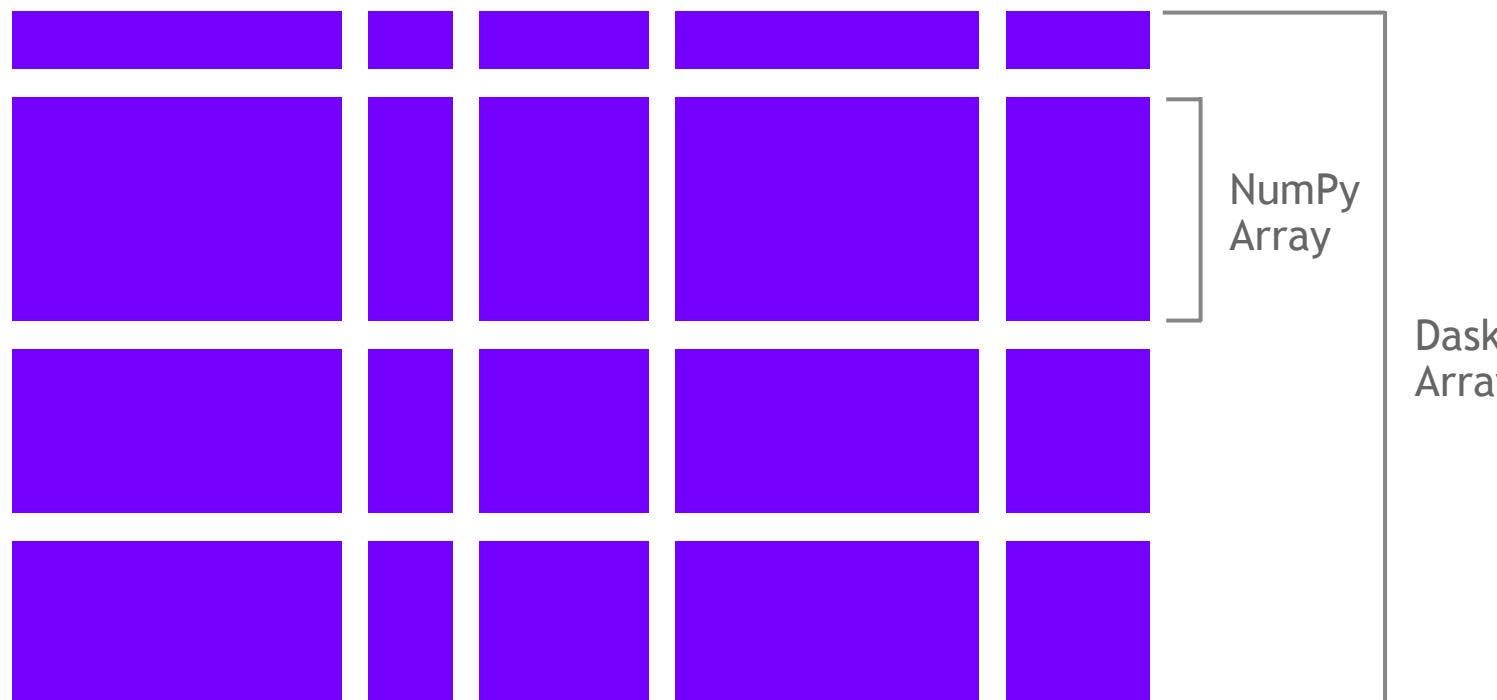
ETL - Arrays and DataFrames

Dask and CUDA Python Arrays



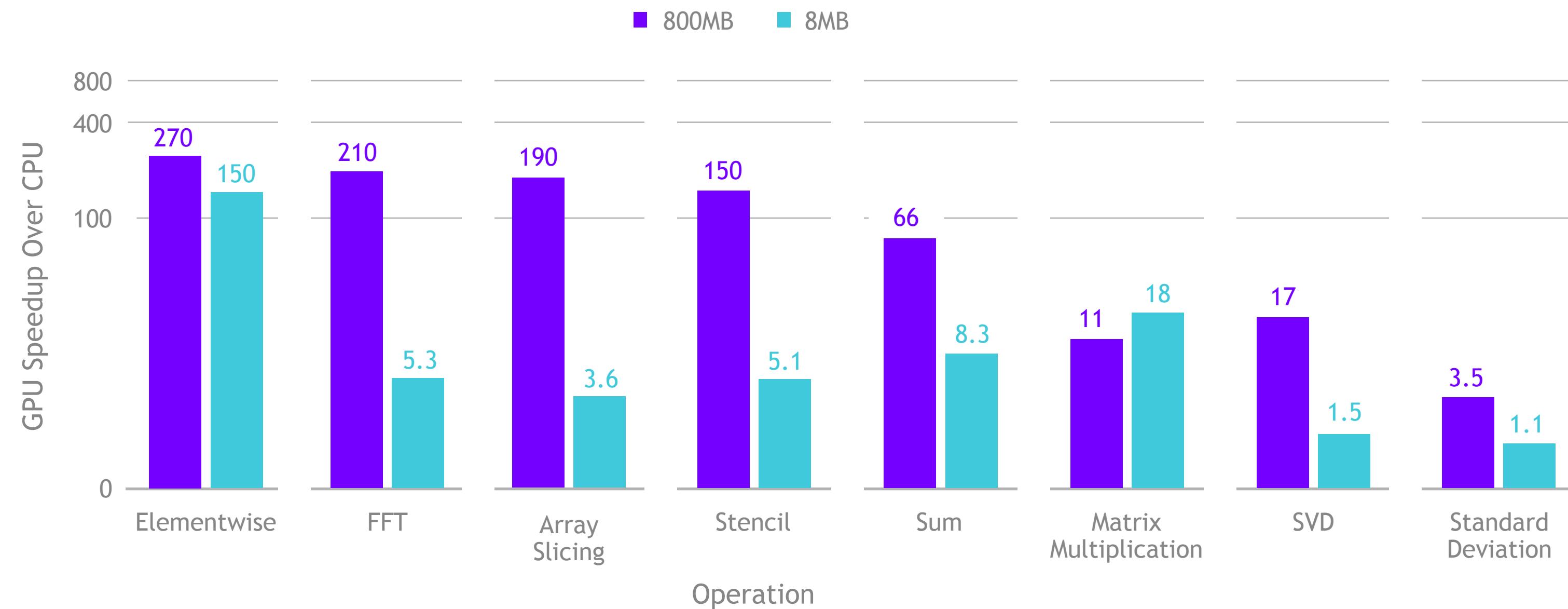
CuPy

 Numba



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

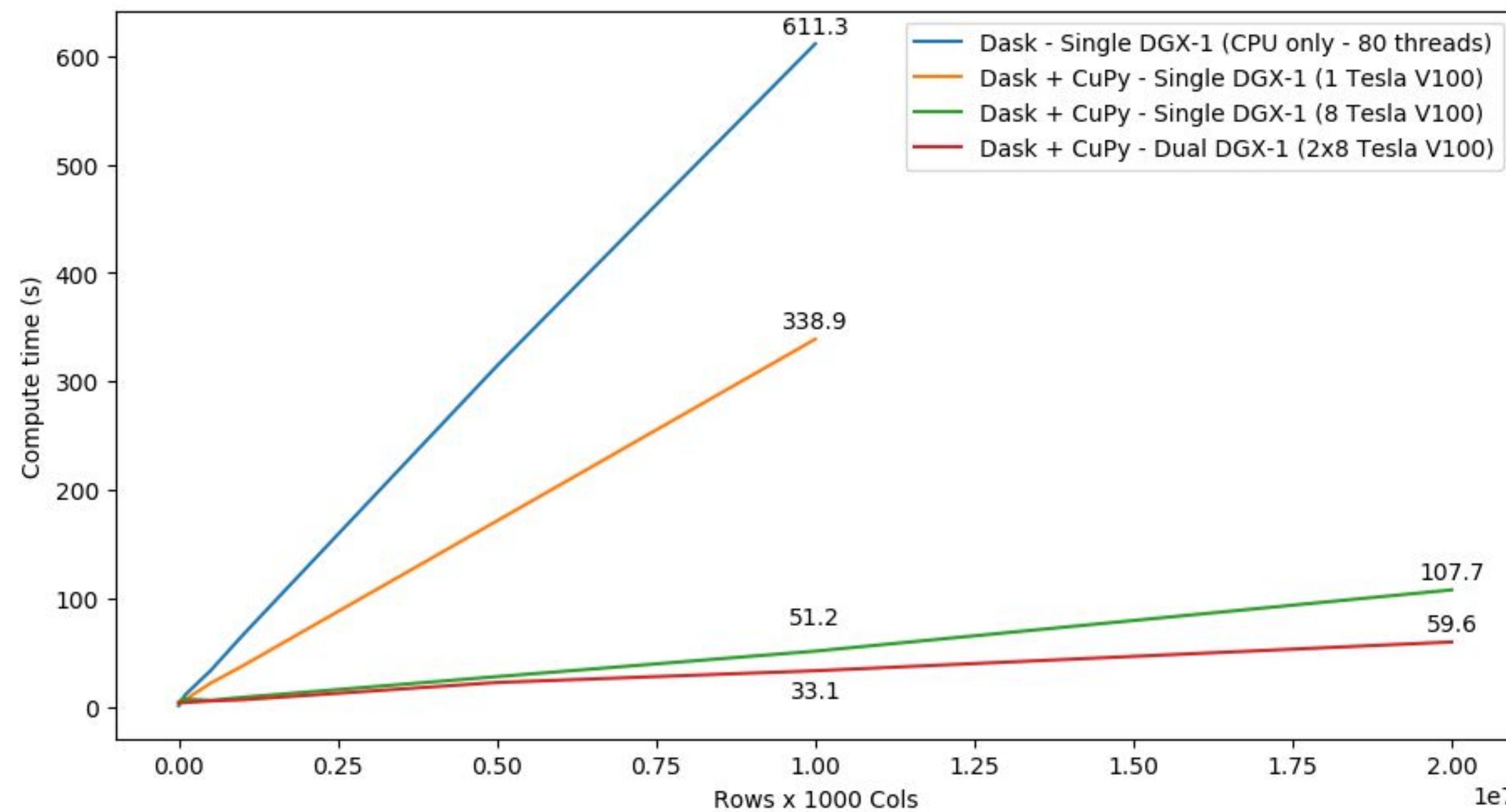
Benchmark: Single-GPU CuPy vs NumPy



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

SVD Benchmark

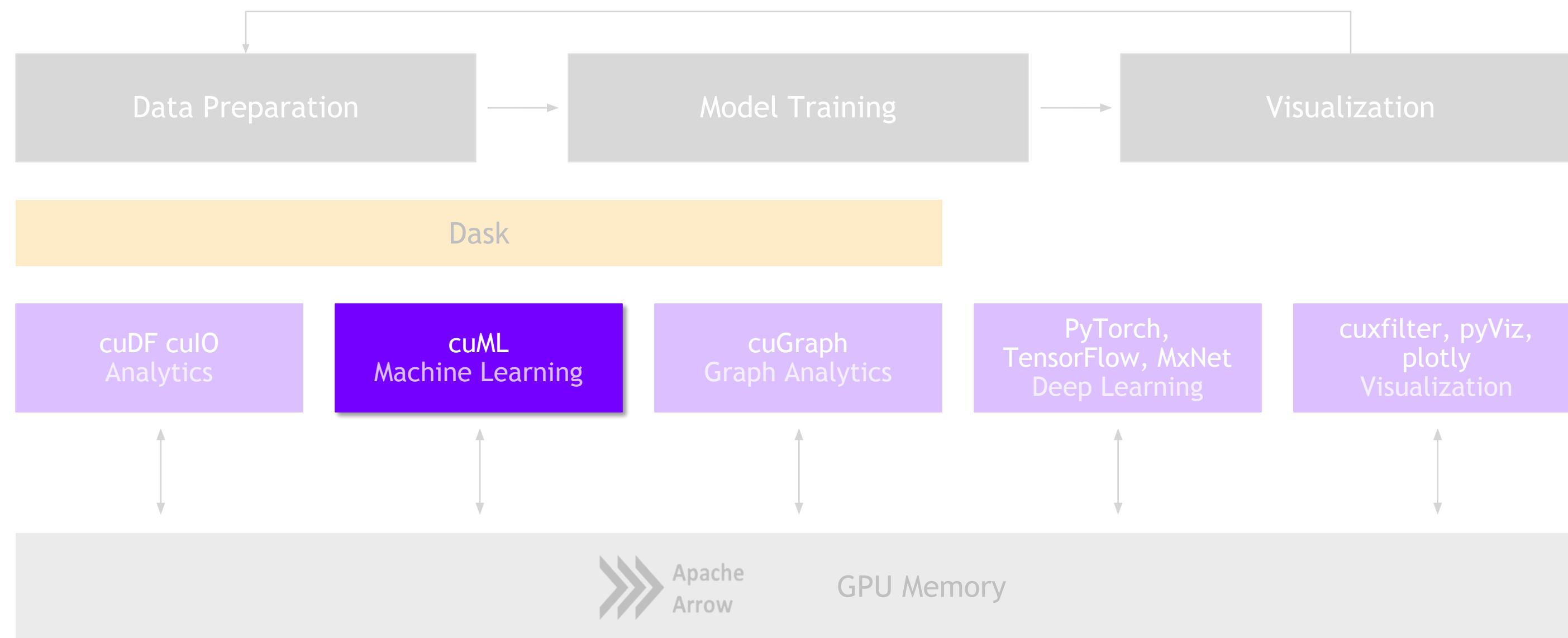
Dask and CuPy Doing Complex Workflows



cuML

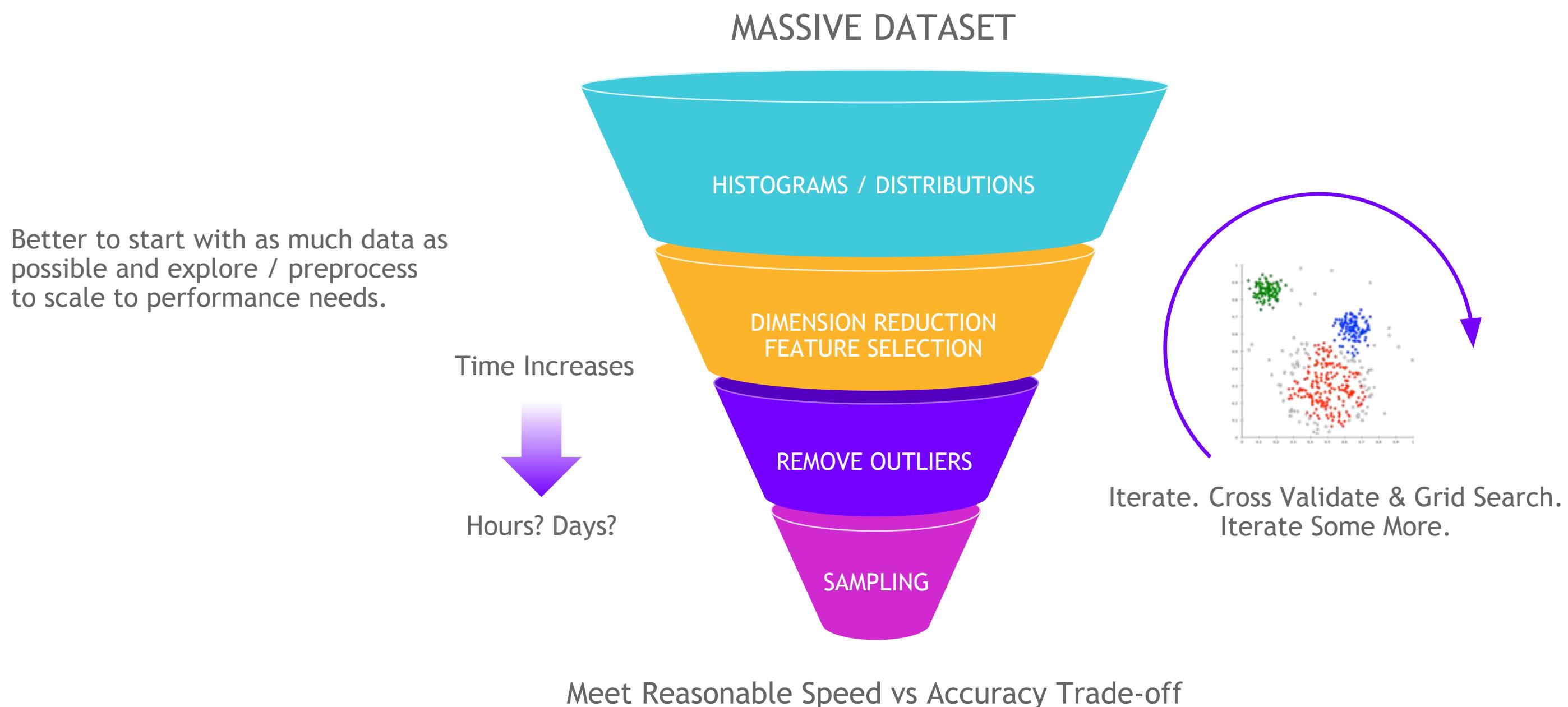
Machine Learning

More Models More Problems

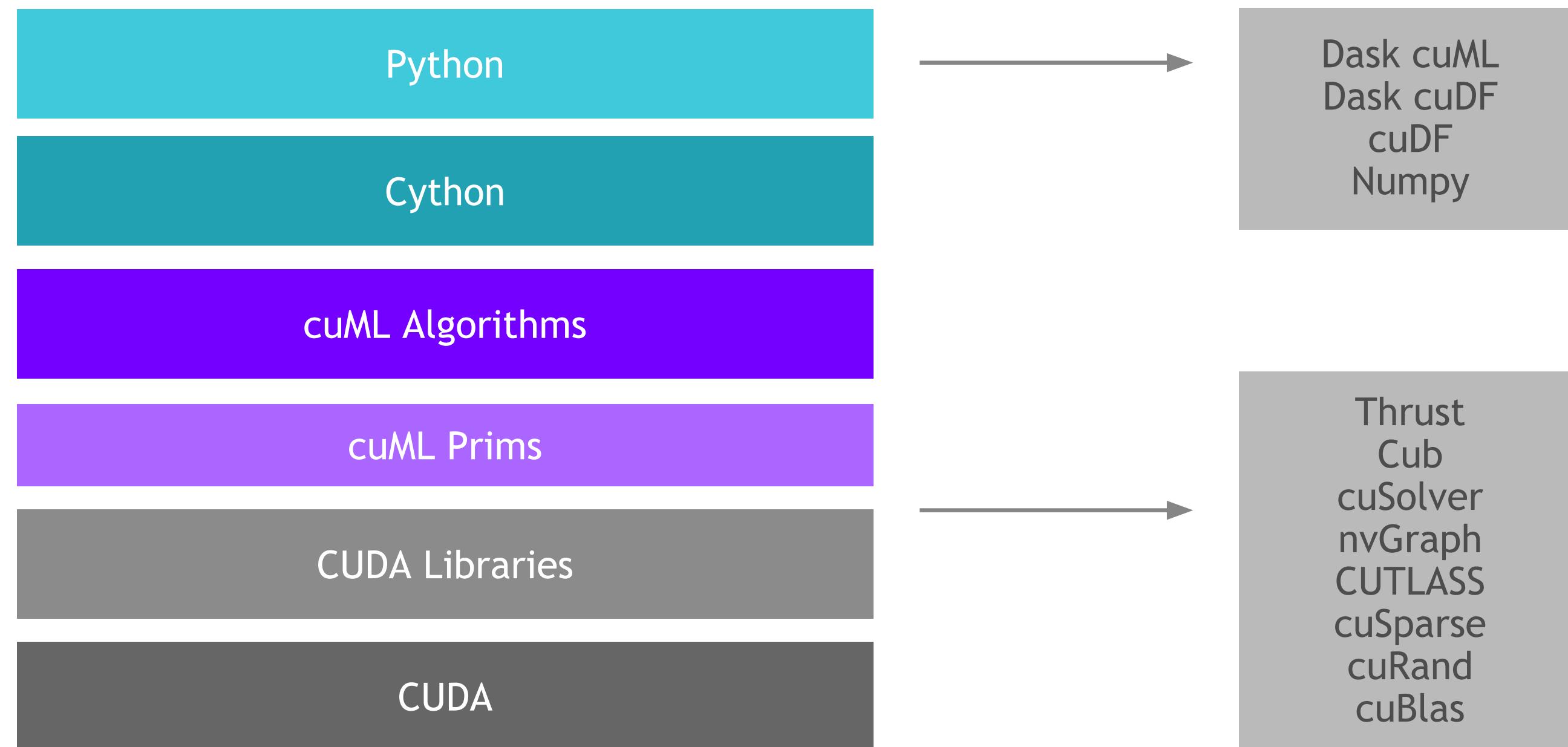


Problem

Data Sizes Continue to Grow



ML Technology Stack



RAPIDS Matches Common Python APIs

CPU-based Clustering

```
from sklearn.datasets import make_moons
import pandas

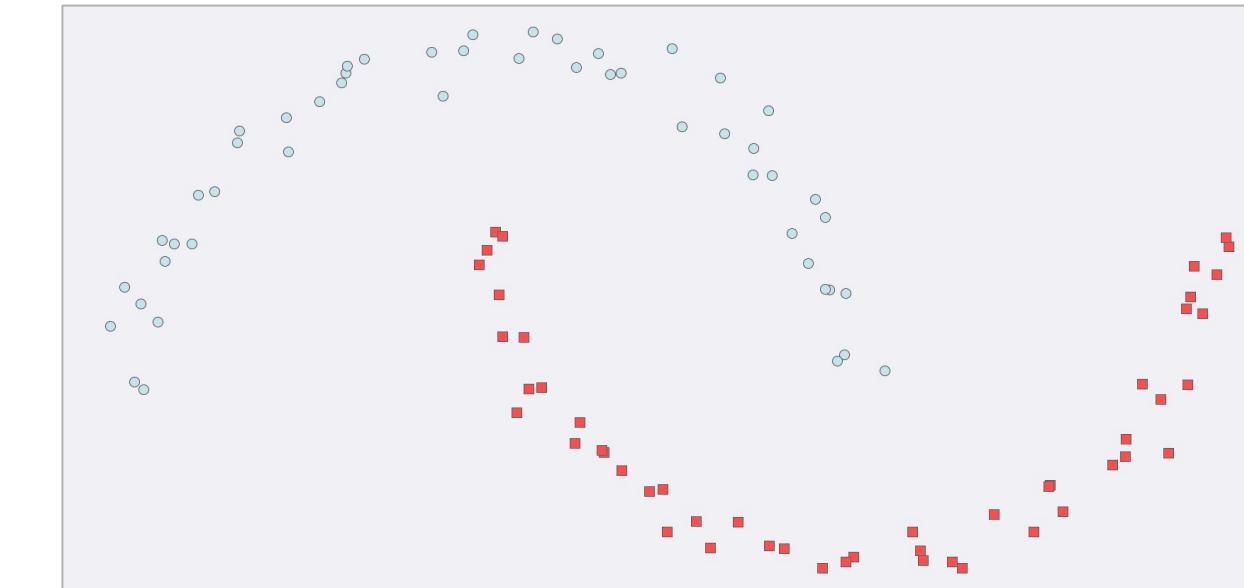
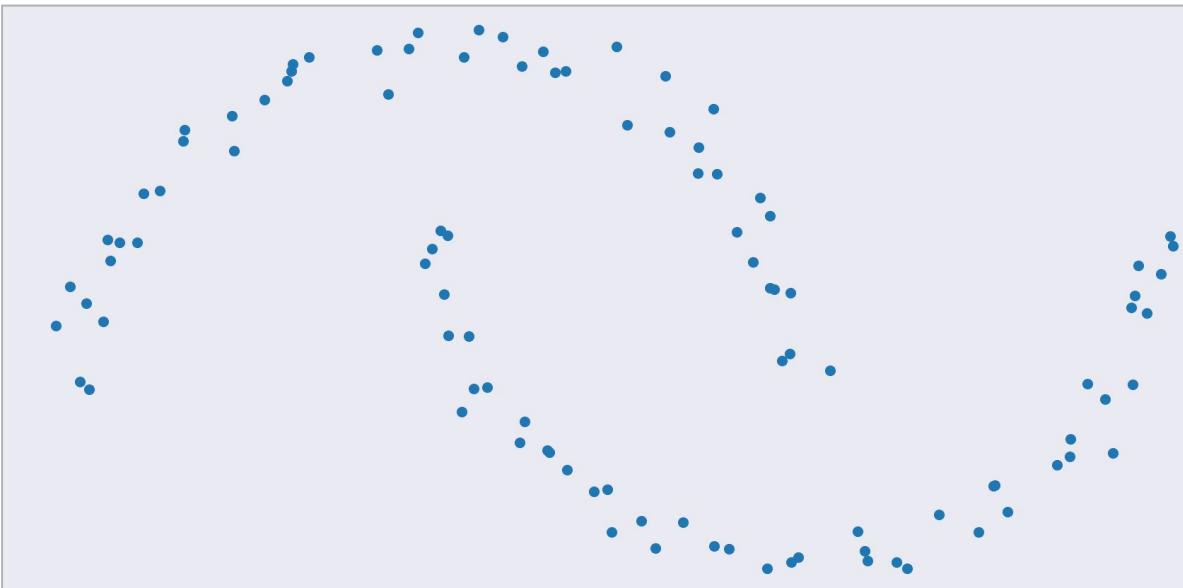
X, y = make_moons(n_samples=int(1e2),
                   noise=0.05, random_state=0)

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

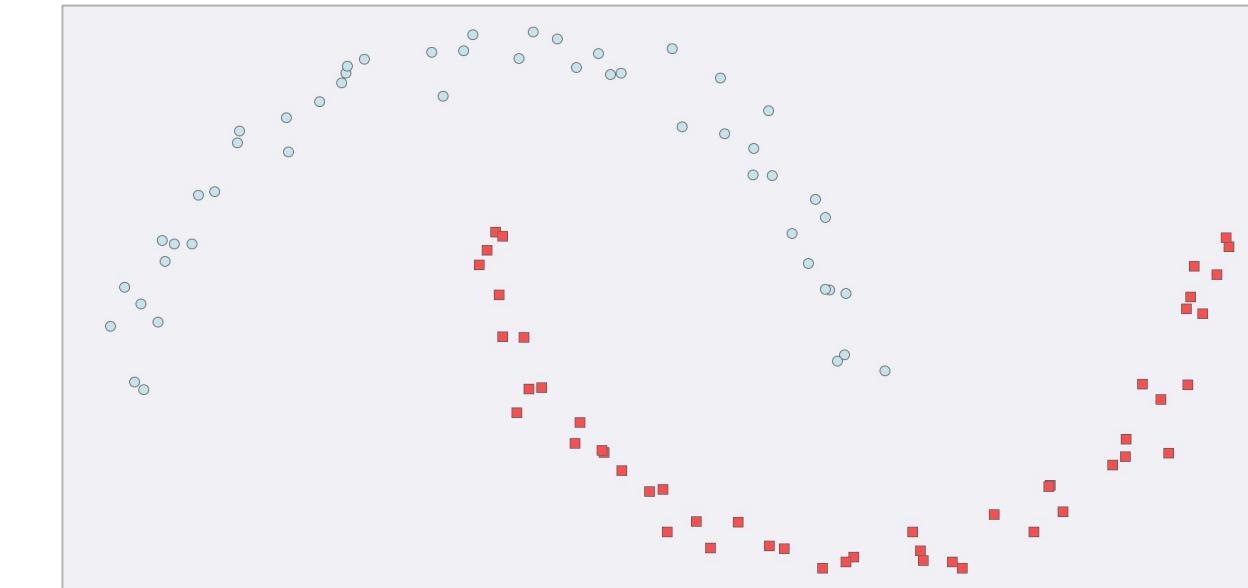
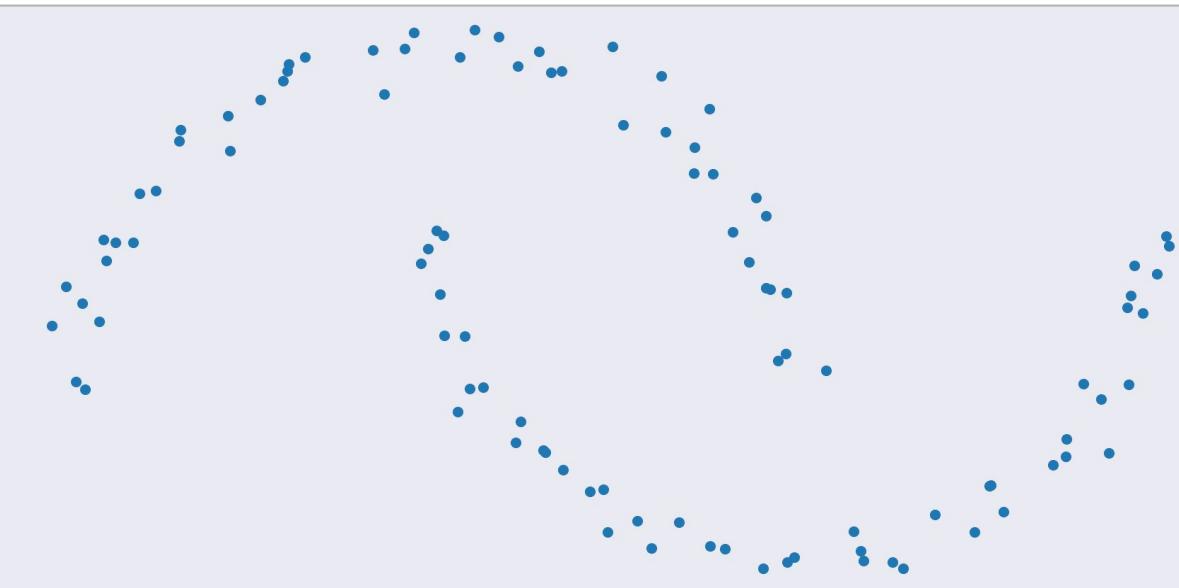
X, y = make_moons(n_samples=int(1e2),
                   noise=0.05, random_state=0)

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)

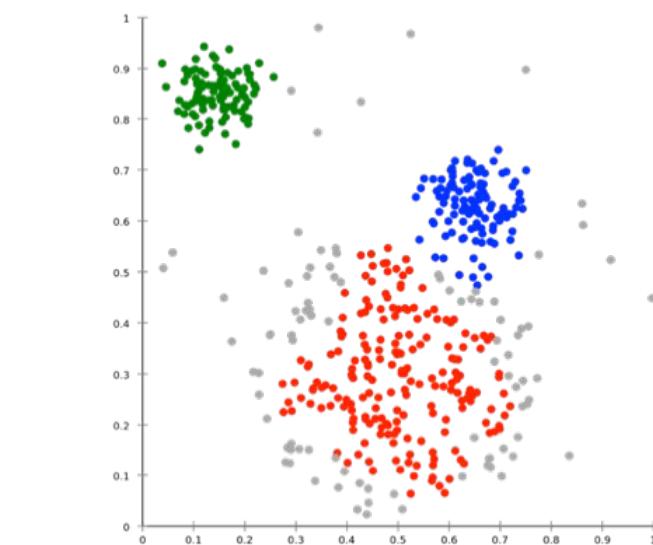
dbscan.fit(X)

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

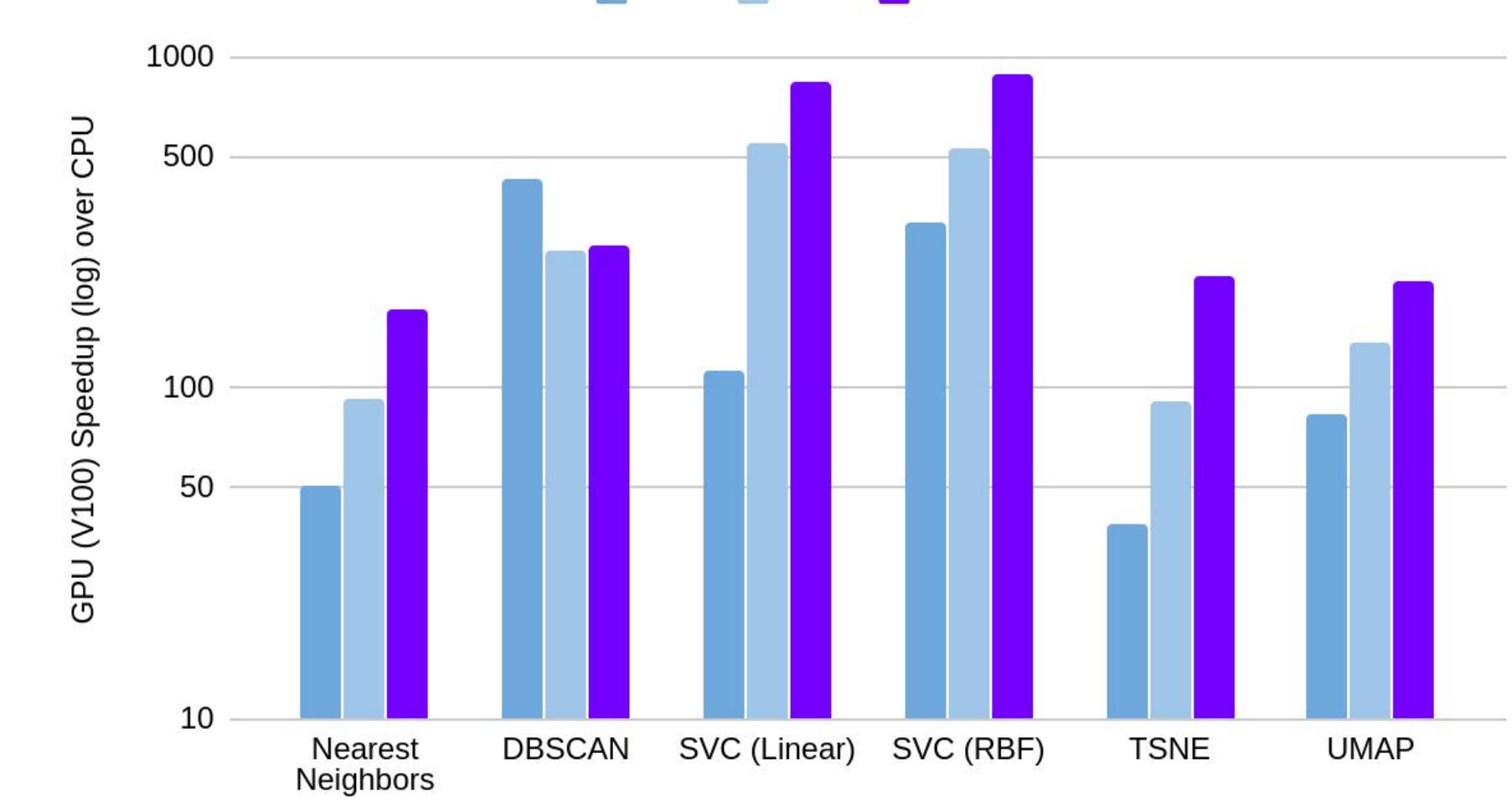
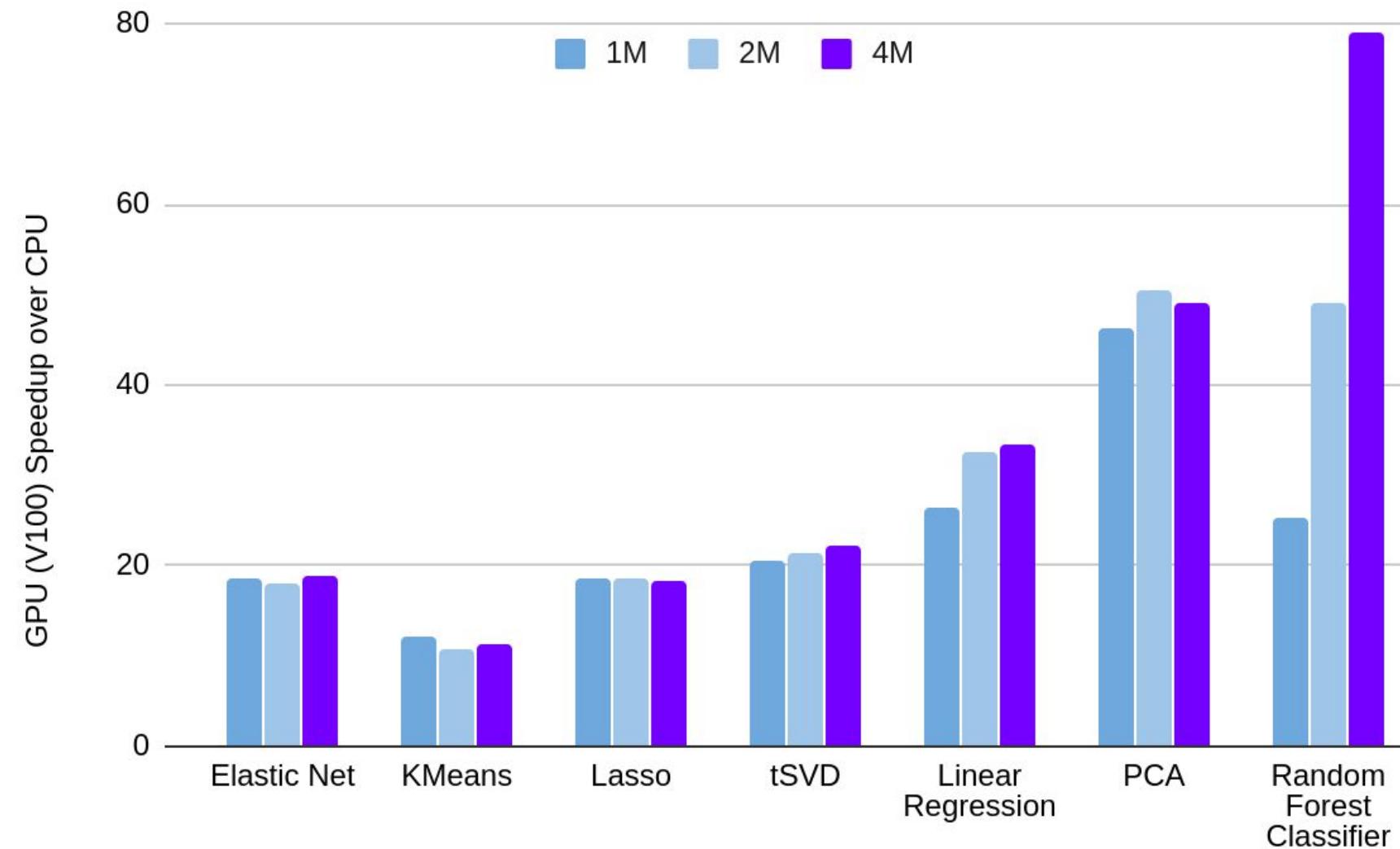
Decision Trees / Random Forests
Linear/Lasso/Ridge/ElasticNet Regression
Logistic Regression
K-Nearest Neighbors
Support Vector Machine Classification and
Regression
Naive Bayes
Random Forest / GBDT Inference (FIL)

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

Benchmarks: Single-GPU cuML vs Scikit-learn



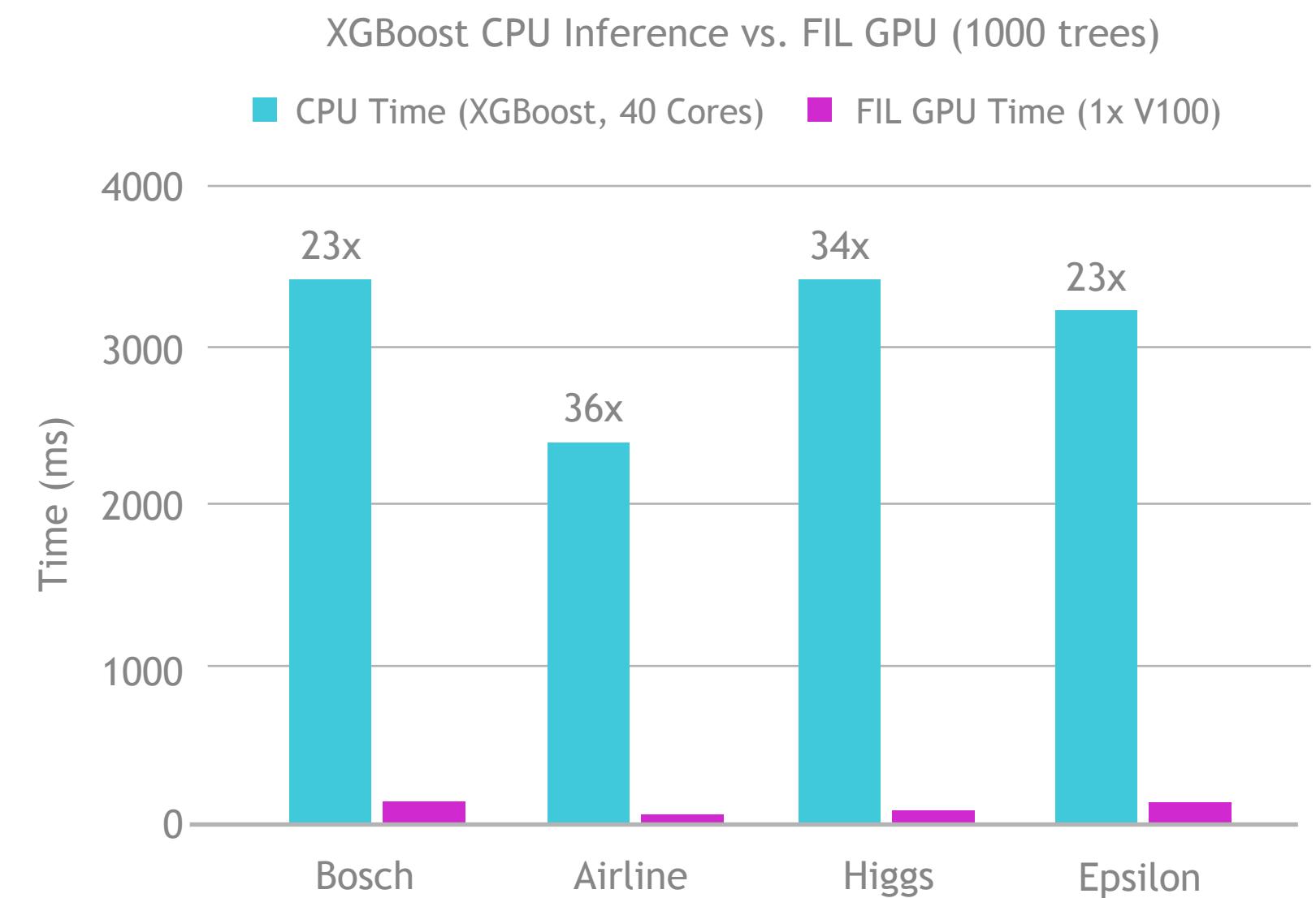
1x V100 vs. 2x 20 Core CPUs (DGX-1, RAPIDS 0.15)

Forest Inference

Taking Models From Training to Production

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

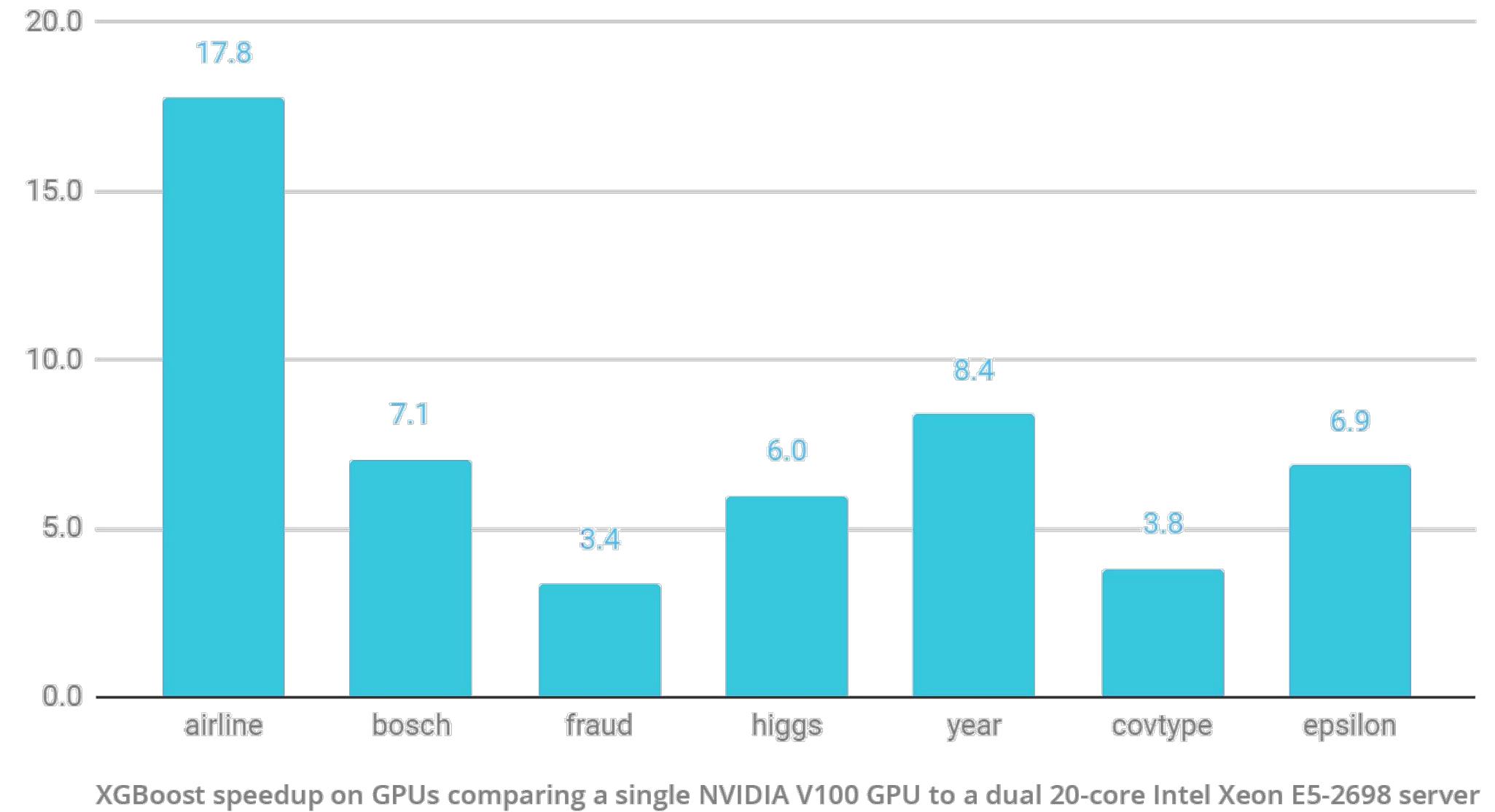
- Works with existing saved models (XGBoost, LightGBM, scikit-learn RF cuML RF soon)
- Lightweight Python API
- Single V100 GPU can infer up to 34x faster than XGBoost dual-CPU node
- Over 100 million forest inferences



XGBoost + RAPIDS: Better Together

- ▶ RAPIDS comes paired with a snapshot of XGBoost 1.3 (as of 0.16)
- ▶ 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



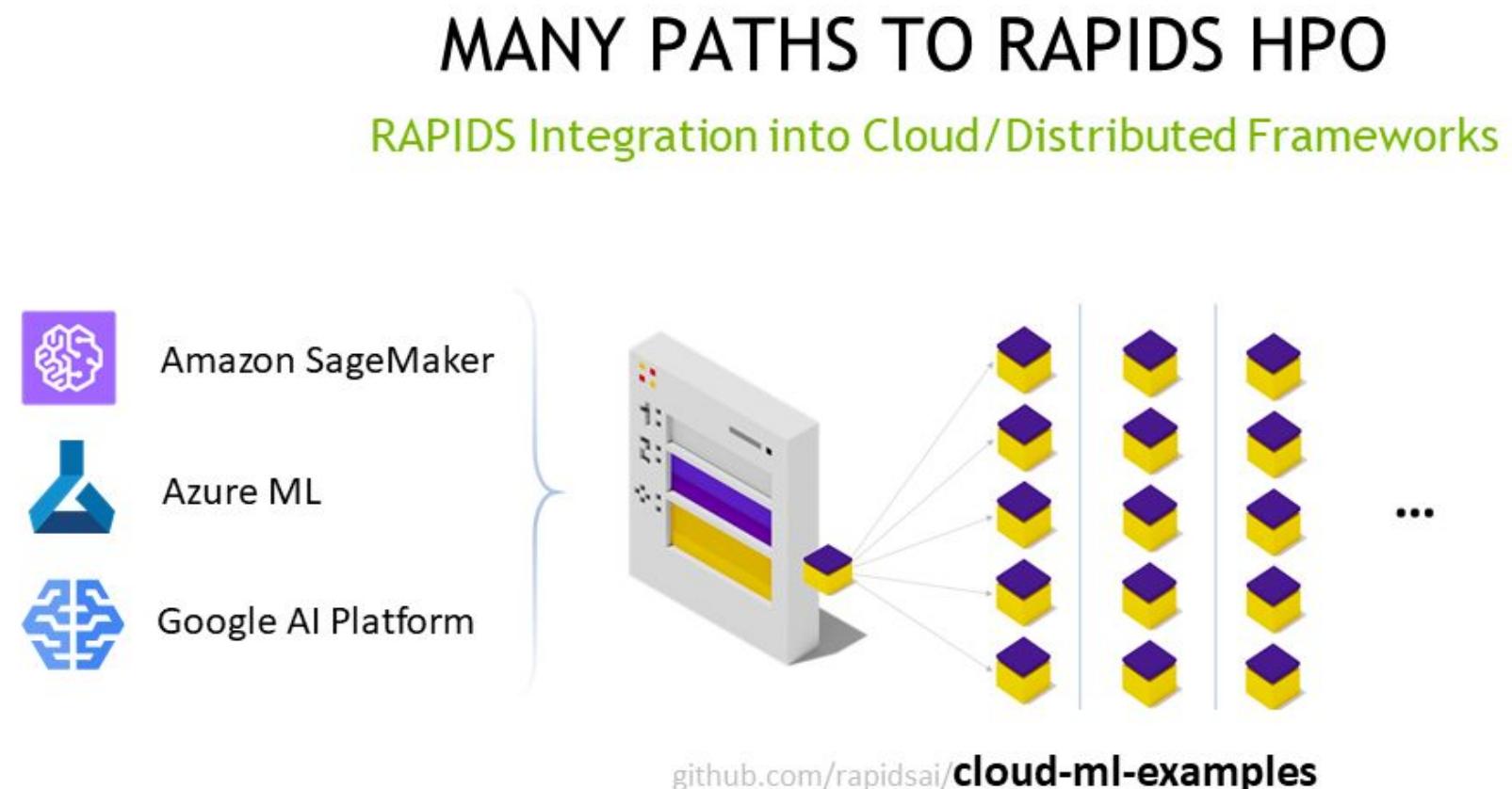
RAPIDS Integrated into Cloud ML Frameworks

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.

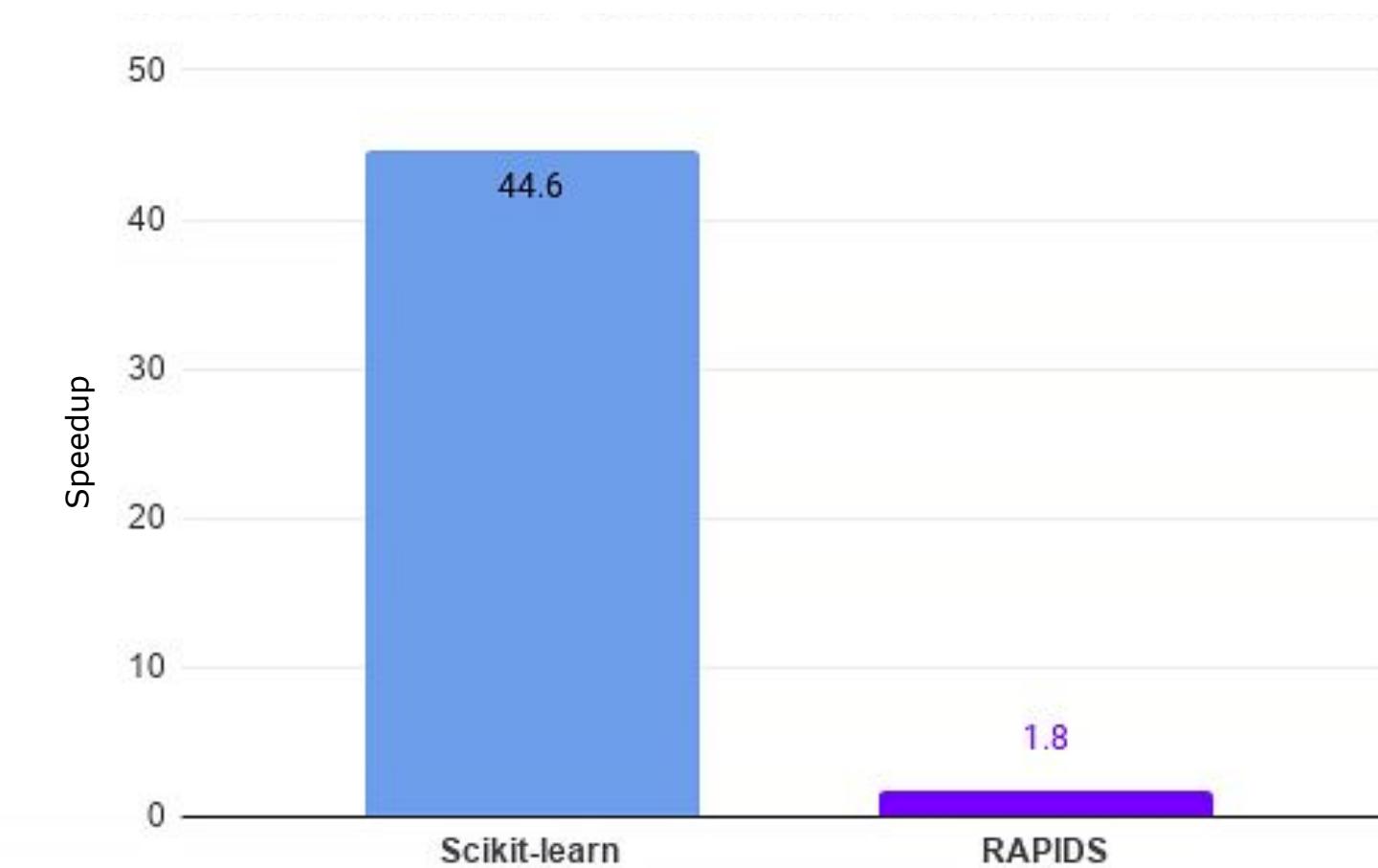
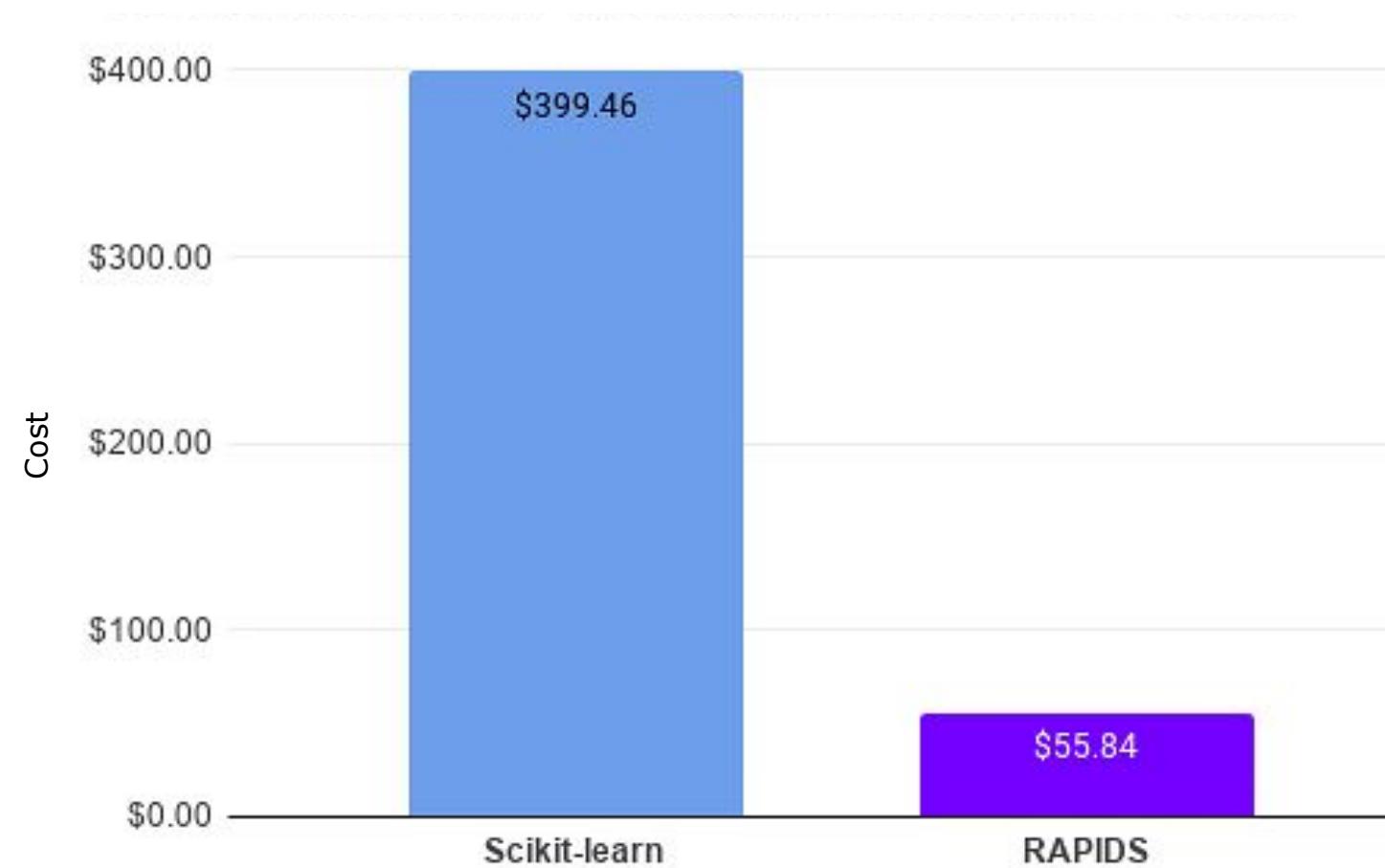
The RAPIDS team works closely with major cloud providers and OSS solution providers to provide code samples to get started with HPO in minutes

<https://rapids.ai/hpo>



HPO Use Case: 100-Job Random Forest Airline Model

Huge speedups translate into >7x TCO reduction



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

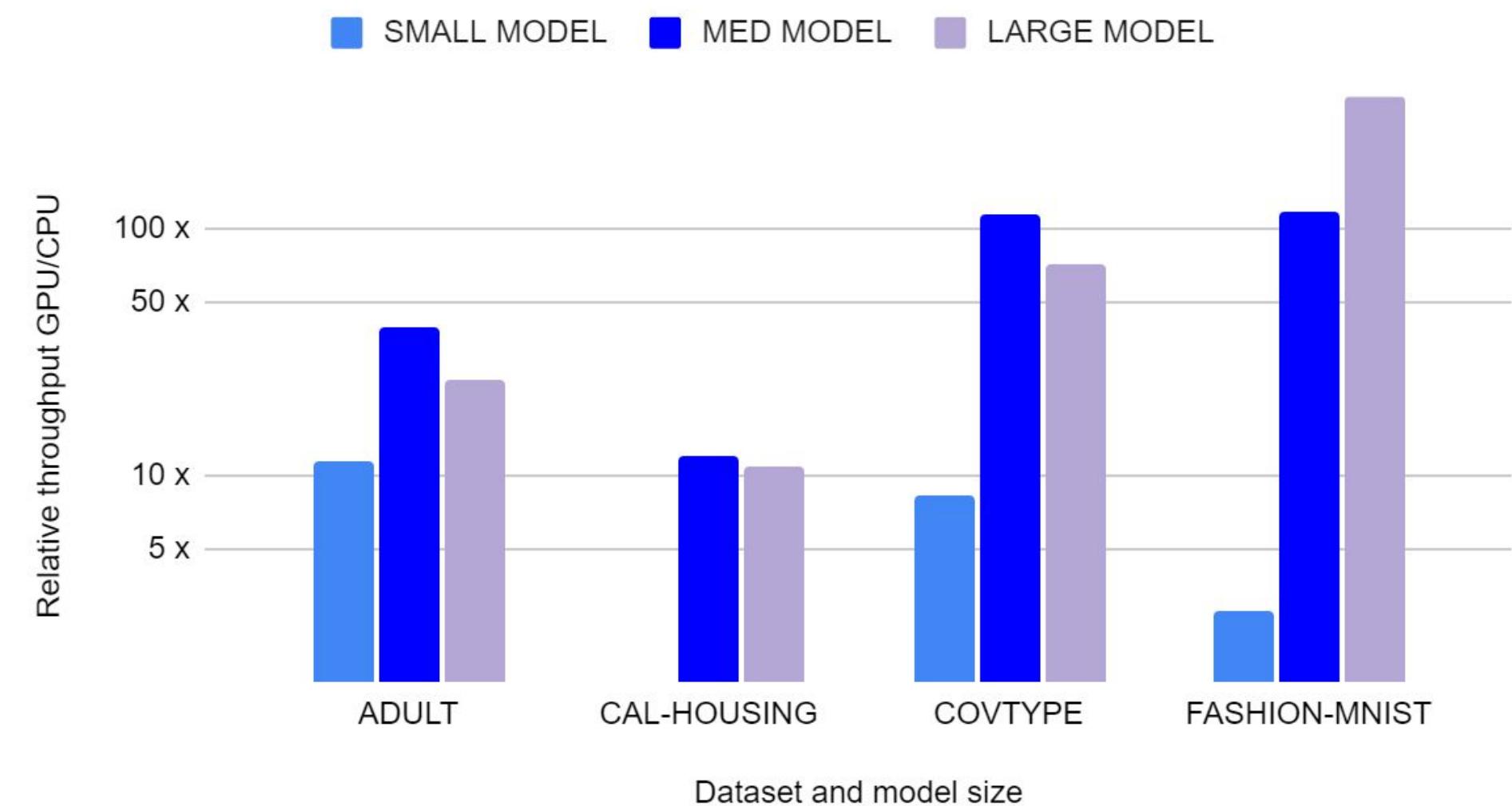
GPU nodes: 10x Standard_NC6s_v3, 1 V100 16G, vCPU 6 memory 112G, Xeon E5-2690 v4 (Broadwell) - \$3.366/hour
CPU nodes: 10x Standard_DS5_v2, vCPU 16 memory 56G, Xeon E5-2673 v3 (Haswell) or v4 (Broadwell) - \$1.017/hour"

SHAP Explainability

GPUTreeSHAP for XGBoost

- ▶ SHAP provides a principled way to explain the impact of input features on each prediction or on the model overall - critical for interpretability
- ▶ SHAP has often been too computationally-expensive to deploy for large-scale production
- ▶ RAPIDS ships with GPU-accelerated SHAP for XGBoost with speedups of 20x or more ([demo code available in the XGBoost repo](#))
- ▶ RAPIDS 0.17 will include Kernel and Permutation explainers for black box models and cuML

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



Road to 1.0 - cuML

June 2020 - RAPIDS 0.16

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

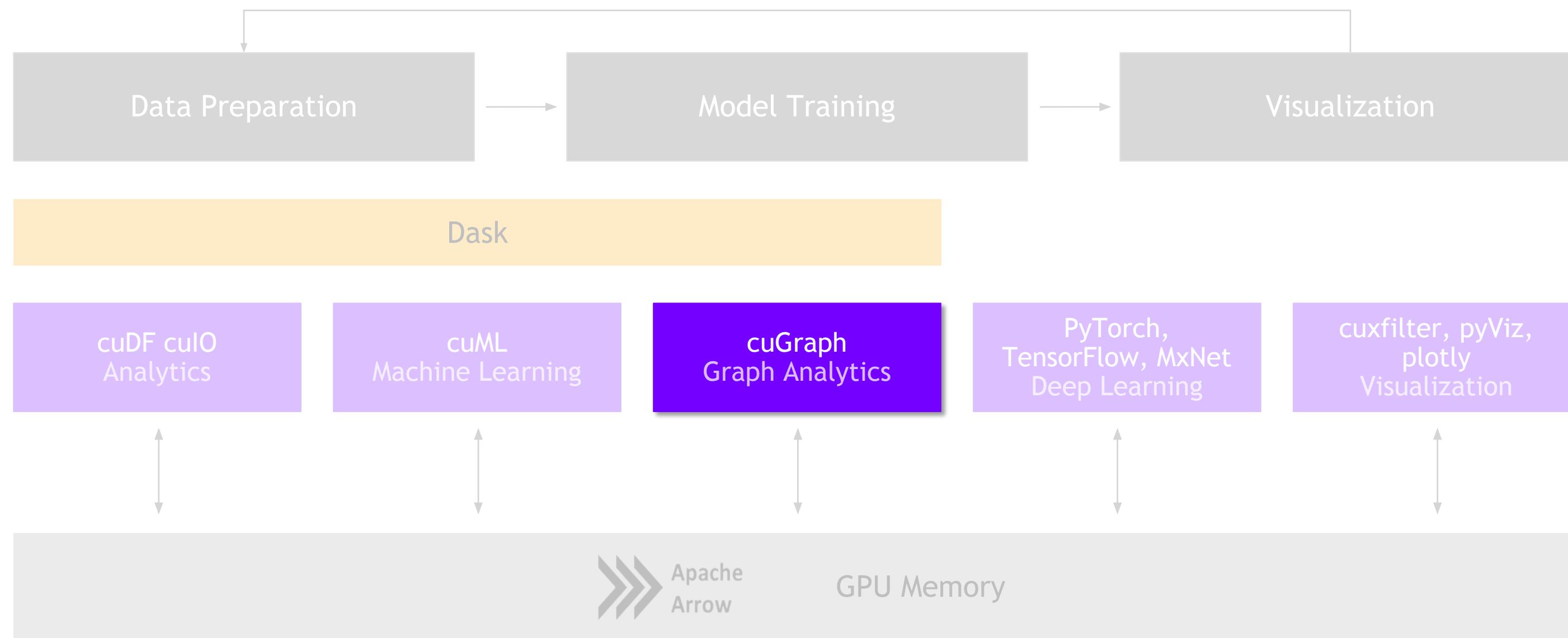
2020 EOY Plan

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

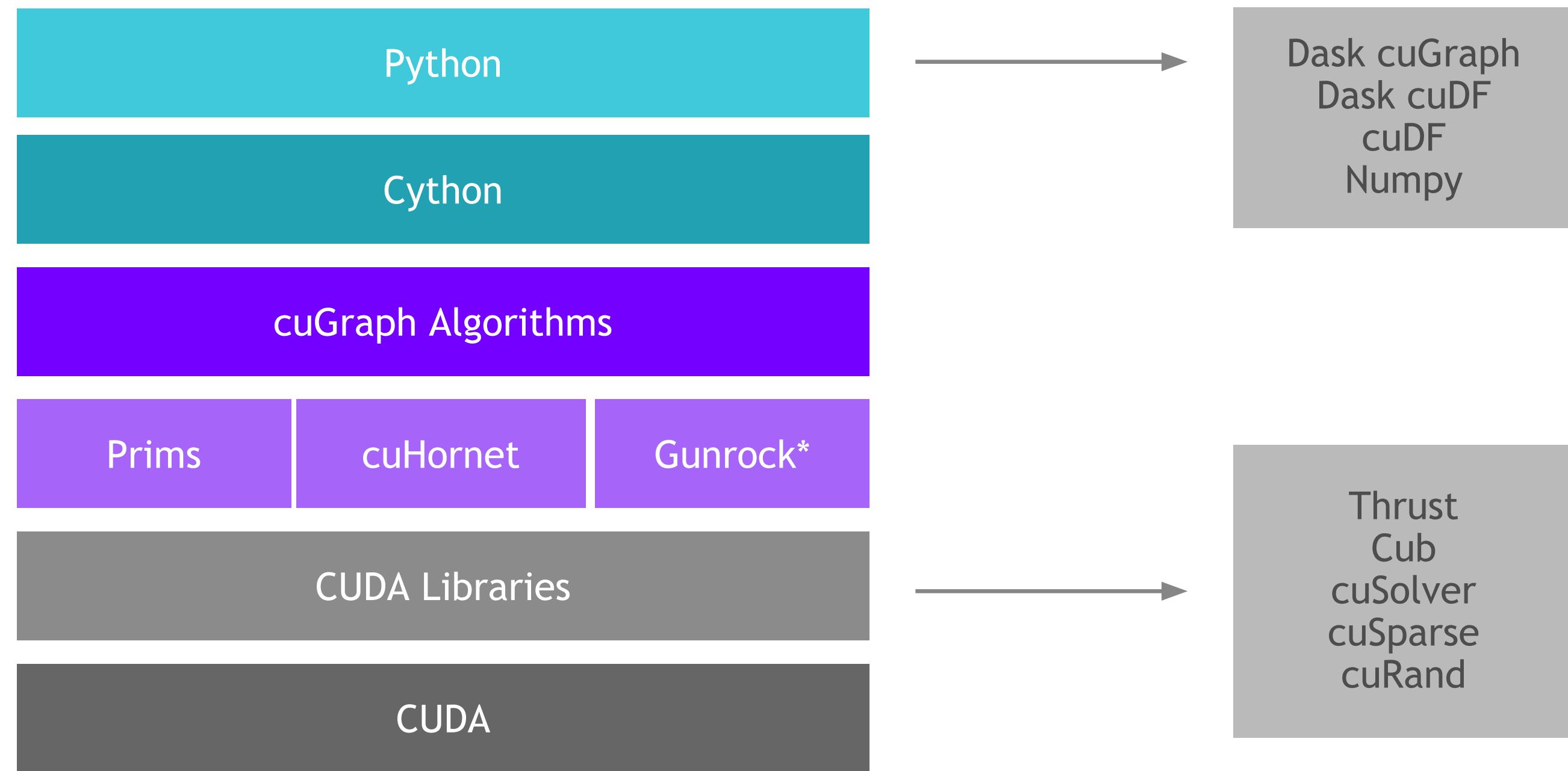
MULTIPLE APIs

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

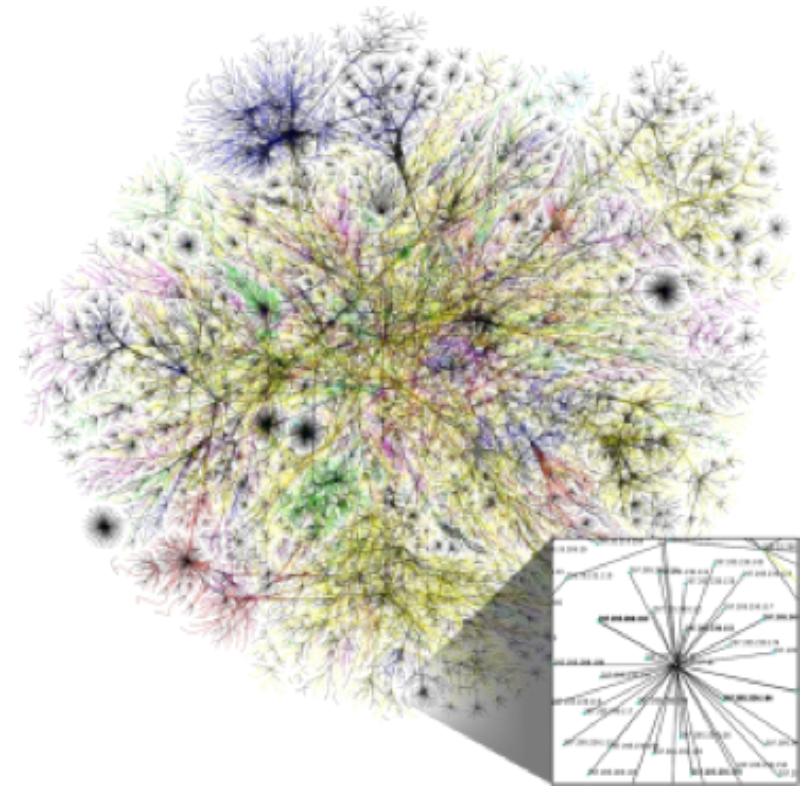
GROWING FUNCTIONALITY

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

Graph Technology Stack



* Gunrock is from UC Davis



Algorithms

GPU-accelerated NetworkX

Graph Classes
Subgraph Extraction

Renumbering
Auto-Renumbering
Force Atlas 2
NetworkX converters

Structure

Utilities

Community

Components

Link Analysis

Link Prediction

Traversal

Centrality

Spectral Clustering

- Balanced Cut and Modularity Maximization

Louvain (**Multi-GPU**) and Leiden

Ensemble Clustering for Graphs

KCore and KCore Number

Triangle Counting

K-Truss

Weakly Connected Components

Strongly Connected Components

Page Rank (**Multi-GPU**)

Personal Page Rank

HITS

Jaccard

Weighted Jaccard

Overlap Coefficient

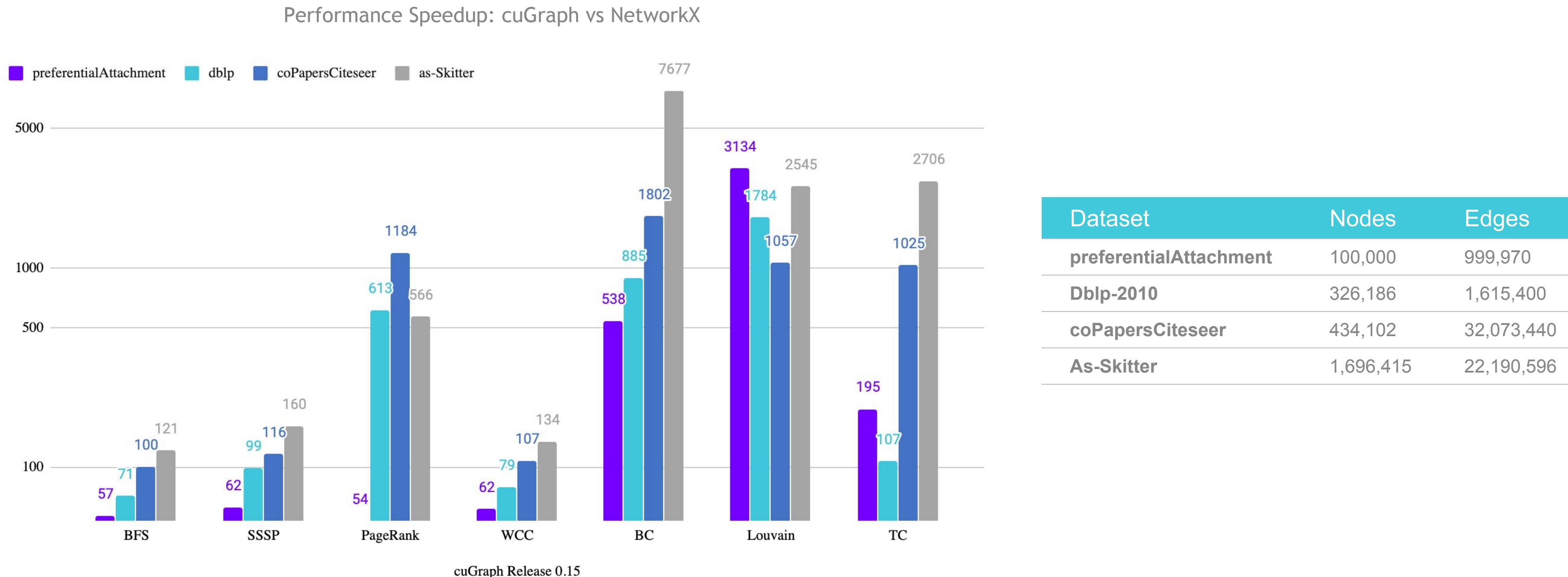
Single Source Shortest Path (SSSP) (**Multi-GPU**)

Breadth First Search (BFS) (**Multi-GPU**)

Katz

Betweenness Centrality (Vertex and Edge)

Benchmarks: Single-GPU cuGraph vs NetworkX



NetworkX Compatibility

Use your NetworkX.Graph Objects directly within cuGraph

```
import networkx as nx
import time
import operator

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

... do some NetworkX stuff ..

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

print(t2)
```

NetworkX

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

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

... do some NetworkX stuff ..

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

print(t2)
```

NetworkX + RAPIDS cuGraph

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

Multi-GPU PageRank Performance

PageRank Portion of the HiBench Benchmark Suite

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

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

Road to 1.0

December 2019 - RAPIDS 0.16

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		

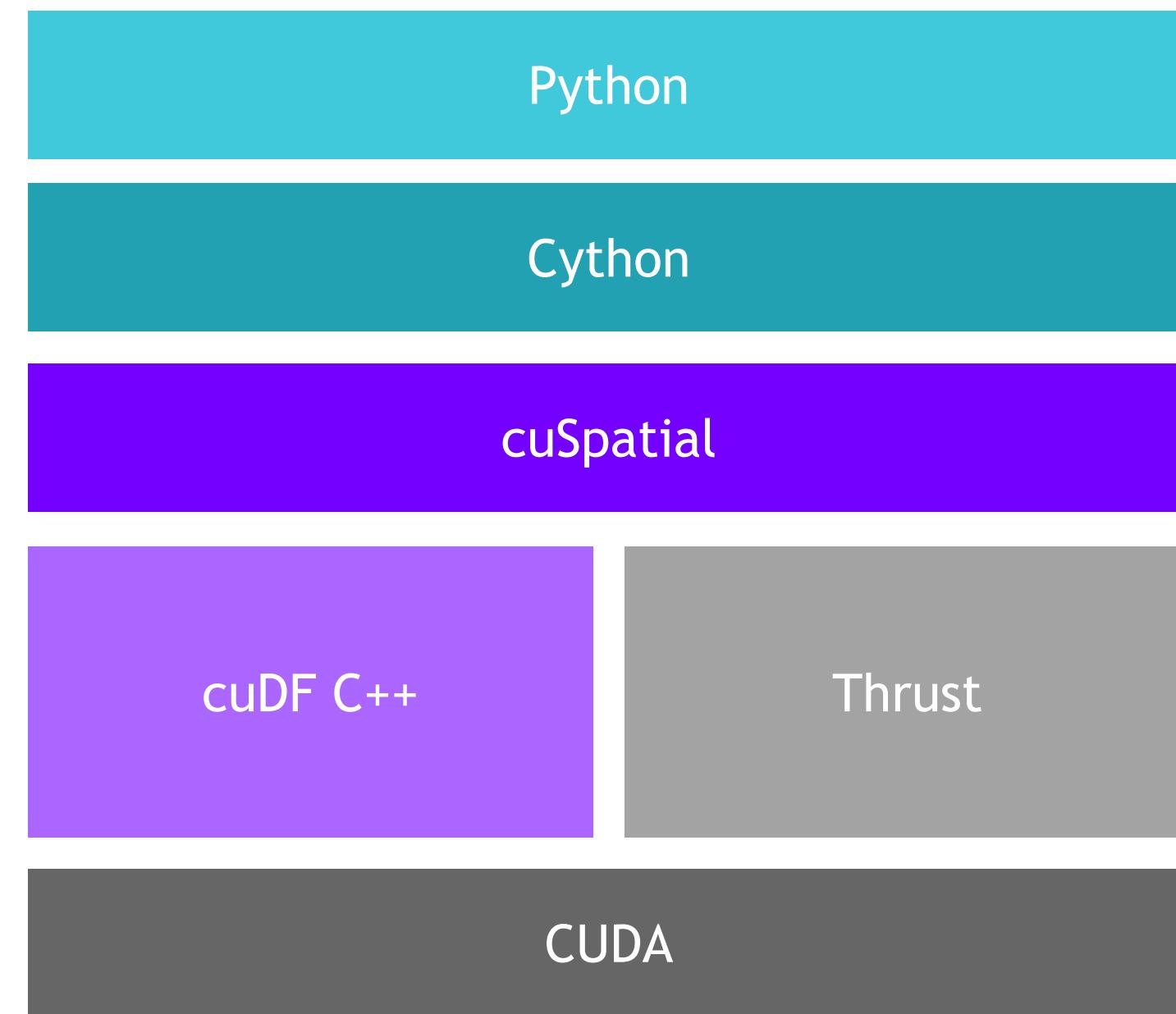
Road to 1.0

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

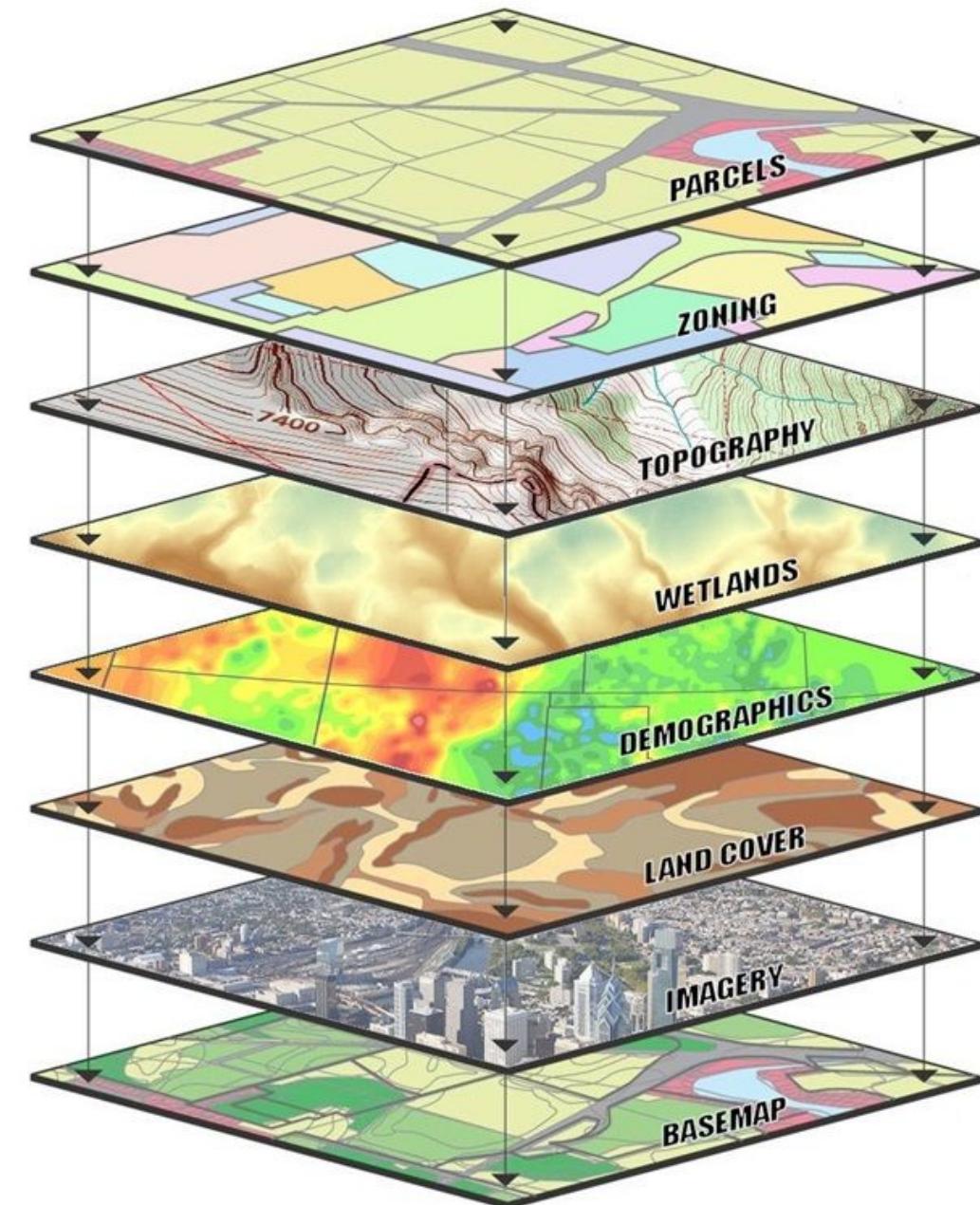
- Up to 1000x faster than CPU spatial libraries
- Python and C++ APIs for maximum usability and integration

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

Current and planned functionality

Layer	0.16 Functionality	Functionality Roadmap
High-level Analytics	C++ Library w. Python bindings enabling distance, speed, trajectory similarity, trajectory clustering	Symmetric segment path distance
Graph Layer	cuGraph	cuGraph
Query Layer	Spatial Window, Nearest polyline	Nearest Neighbor, Spatiotemporal range search and joins
Index Layer	Quadtree	
Geo-operations	Point in polygon (PIP), Haversine distance, Hausdorff distance, lat-lon to xy transformation	ST_distance and ST_contains
Geo-Representation	Shape primitives, points, polylines, polygons	Fiona/Geopandas I/O support and object representations

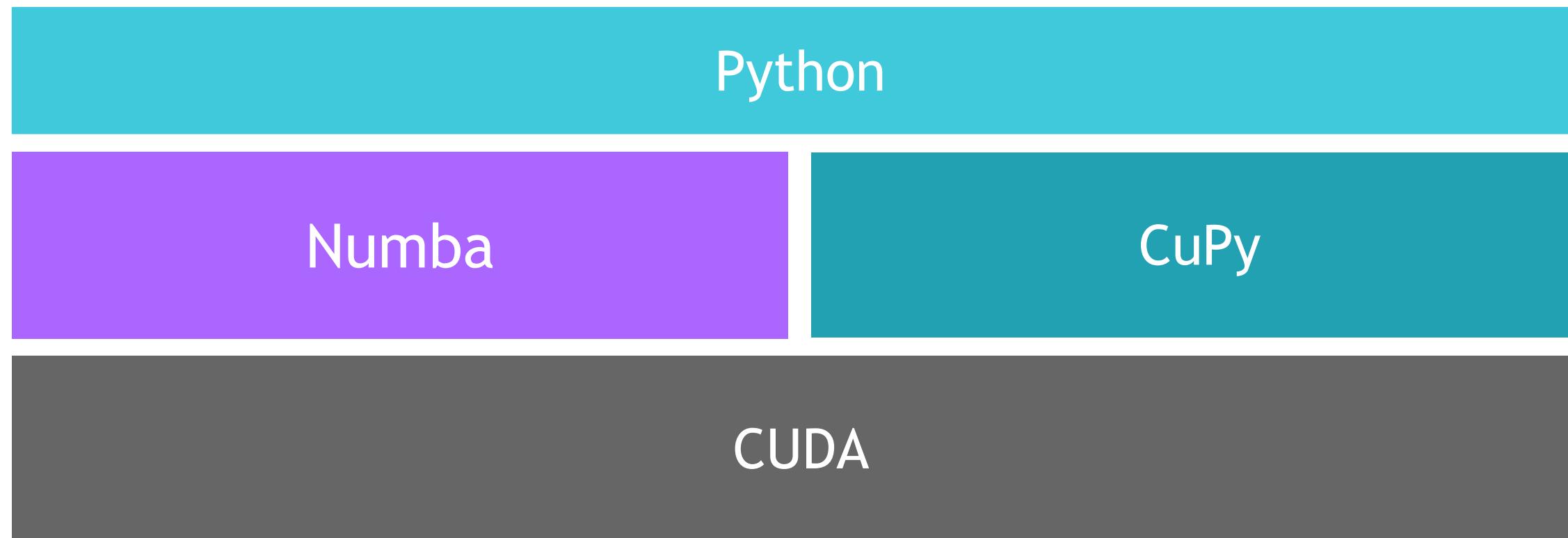
cuSpatial

Performance at a Glance

cuSpatial Operation	Input Data	cuSpatial Runtime	Reference Runtime	Speedup
Point-in-Polygon Test	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)
Haversine Distance Computation	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)
Hausdorff Distance Computation (for clustering)	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()

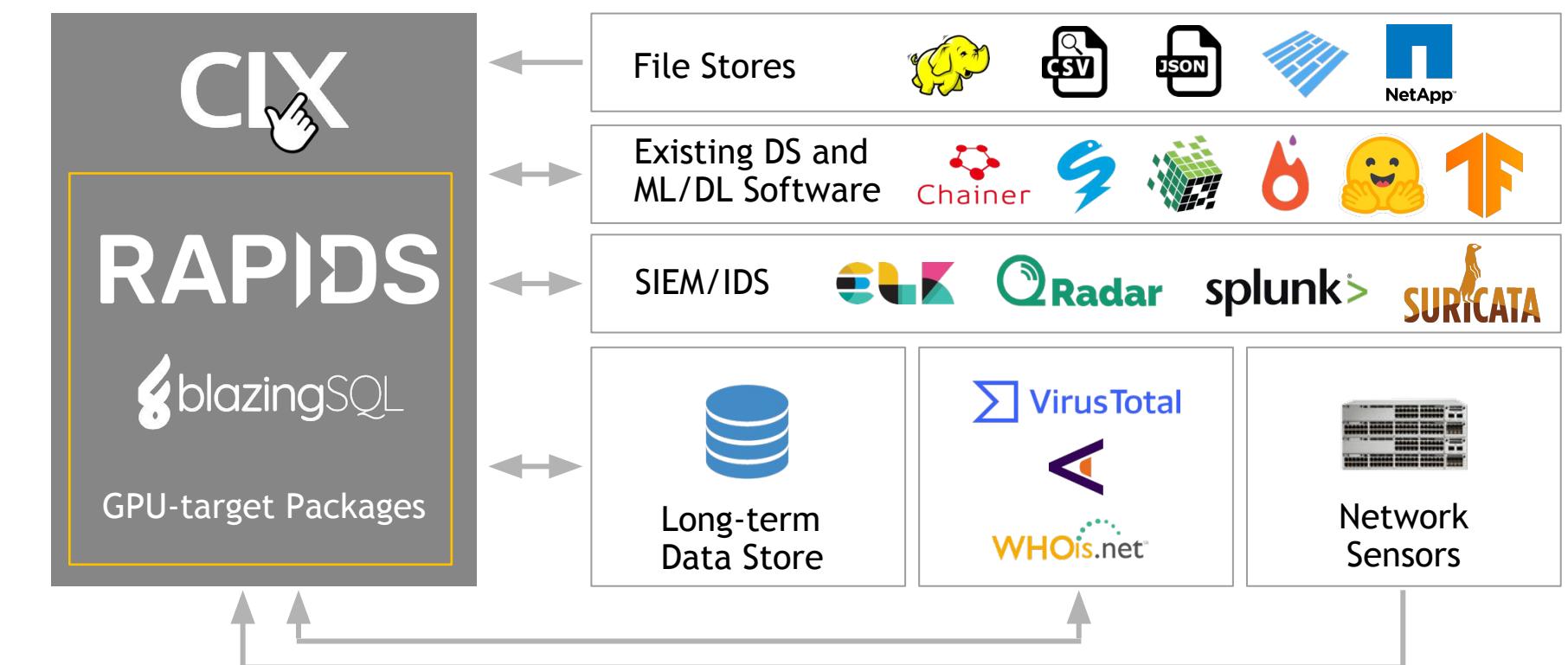
866 µs ± 38 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)
```

Cyber Log Accelerators

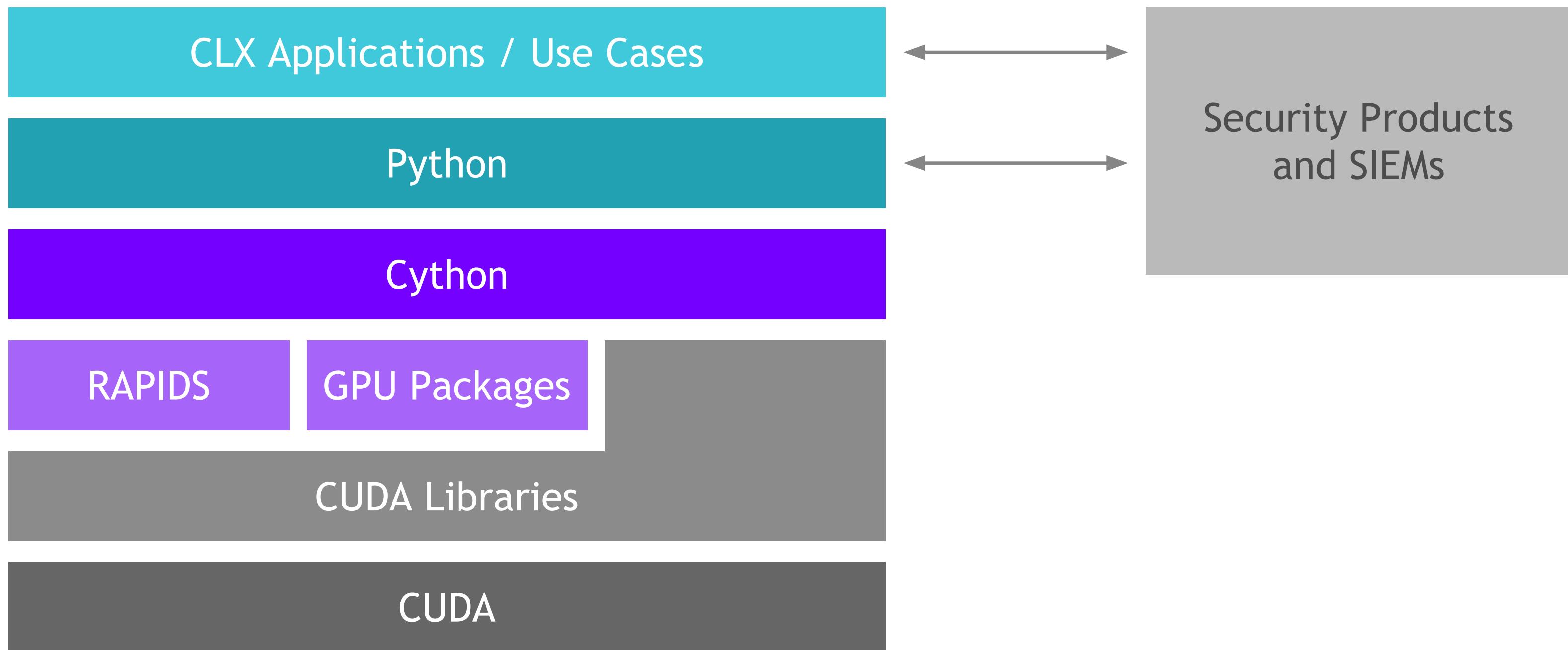
CLX

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 Subword Tokenizer	Pre-Processing		Streaming ready
Accelerated IPv4	Primitive		Now in cuDF
Accelerated DNS	Primitive		

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

- Network Mapping with RAPIDS and CLX**: Shows code for generating heatmaps using cuXFilter.
- Domain Generation Algorithm (DGA) Detection**: Shows code for DGA detection using cuDF.
- cyBERT: a flexible log parser based on the BERT language model**: Shows code for log parsing using cuDF.

The code includes imports for pandas, numpy, cuDF, cuXFilter, and various utility functions. It demonstrates how to convert logs to a DataFrame, filter data by hour or day, and generate heatmaps for analysis.

```

hours_heatmap_df = cudf.DataFrame()
hours_heatmap_df['rule'] = hours_rule_gdf['rule'].unique()
hours_heatmap_df['hour'] = hours_rule_gdf['hour'].min(), hours_rule_gdf['hour'].max(), 3600
days_heatmap_df = cudf.DataFrame()
days_heatmap_df['rule'] = days_rule_gdf['rule'].unique()
days_heatmap_df['day'] = days_rule_gdf['day'].min(), days_rule_gdf['day'].max(), 86400
def normalize_get(input_gdf, time_gdf, output_gdf):
    for rule in input_gdf['rule'].unique():
        temp_gdf = input_gdf[input_gdf['rule'] == rule]
        temp_gdf['count'] = temp_gdf['count'].fillna(0)
        temp_gdf['rule_mean'] = num
        temp_gdf['rule_std'] = std
        if num == 0:
            temp_gdf['rule_mean'] = 0
            temp_gdf['rule_std'] = 1
        else:
            temp_gdf['rule_mean'] = temp_gdf['count'].mean() / temp_gdf['count'].std()
            temp_gdf['rule_std'] = temp_gdf['count'].std()
    heatmap_gdf = cudf.concat([heatmap_gdf, temp_gdf])
    heatmap_gdf['count'] = heatmap_gdf['count'].astype('float')
    heatmap_gdf['rule'] = heatmap_gdf['rule'].astype('float')
    heatmap_gdf['hour'] = heatmap_gdf['hour'].astype('float')
    heatmap_gdf['day'] = heatmap_gdf['day'].astype('float')

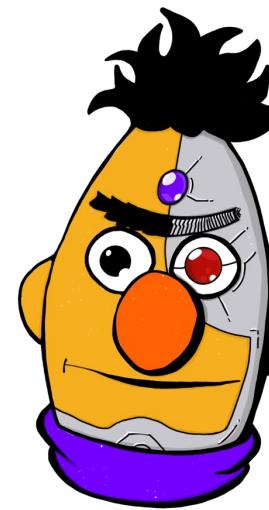
# create cuXfilter heatmap
cuX_df = DataFrame.from_cudf(heatmap_gdf)
chart = charts.cudafashader.heatmap('hour', y='rule_num', aggregate_col='normalized', point_size=30, title='Alerts Per Hour')
d = cuX_df.dashboard(chart1, layout=layouts.single_feature)
chart1.refesh()

```

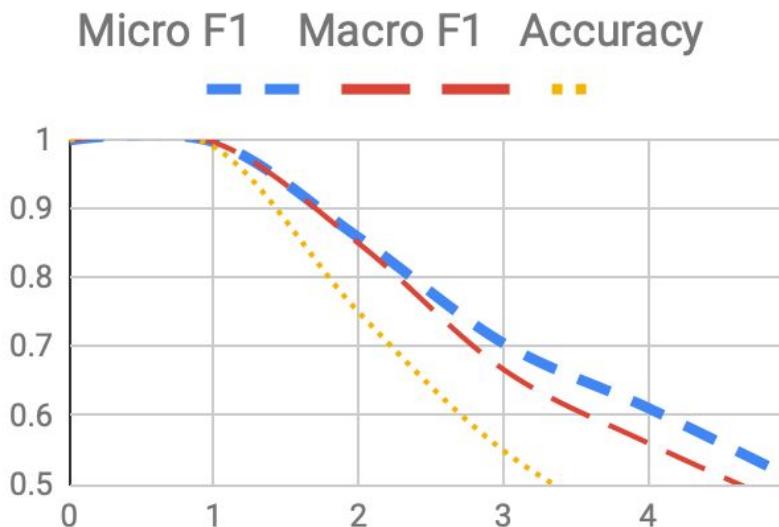
cyBERT

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

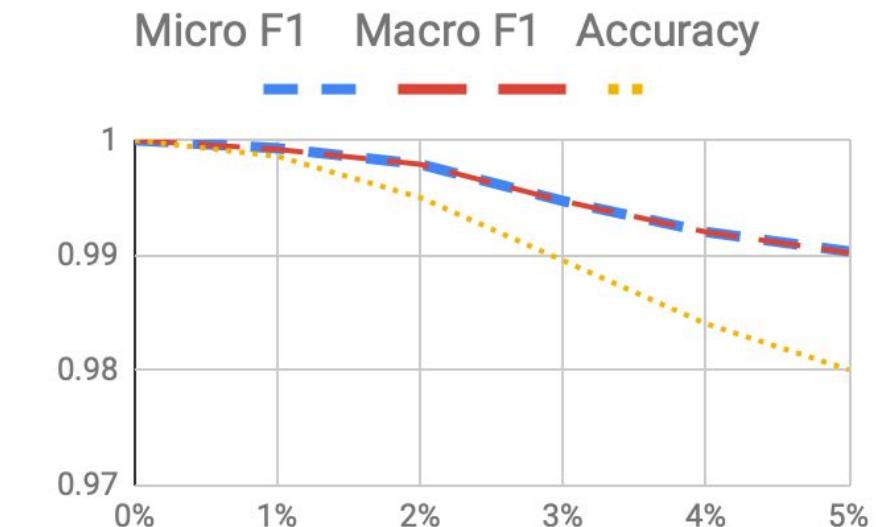
- ▶ Provide a flexible method that does not use heuristics/regex to parse cybersecurity logs
- ▶ Parsing with **micro-F1** and **macro-F1 > 0.999** across heterogeneous log types with a **validation loss of < 0.0048**
- ▶ Second version ~160x (min) faster than first version due to creation of the first all-GPU subword tokenizer that supports non-truncation of logs/sentences
- ▶ Streaming-ready version now available on the CLX GitHub repo (uses cuStreamz and accelerated Kafka reading)



Log Parsing with Inserted Values



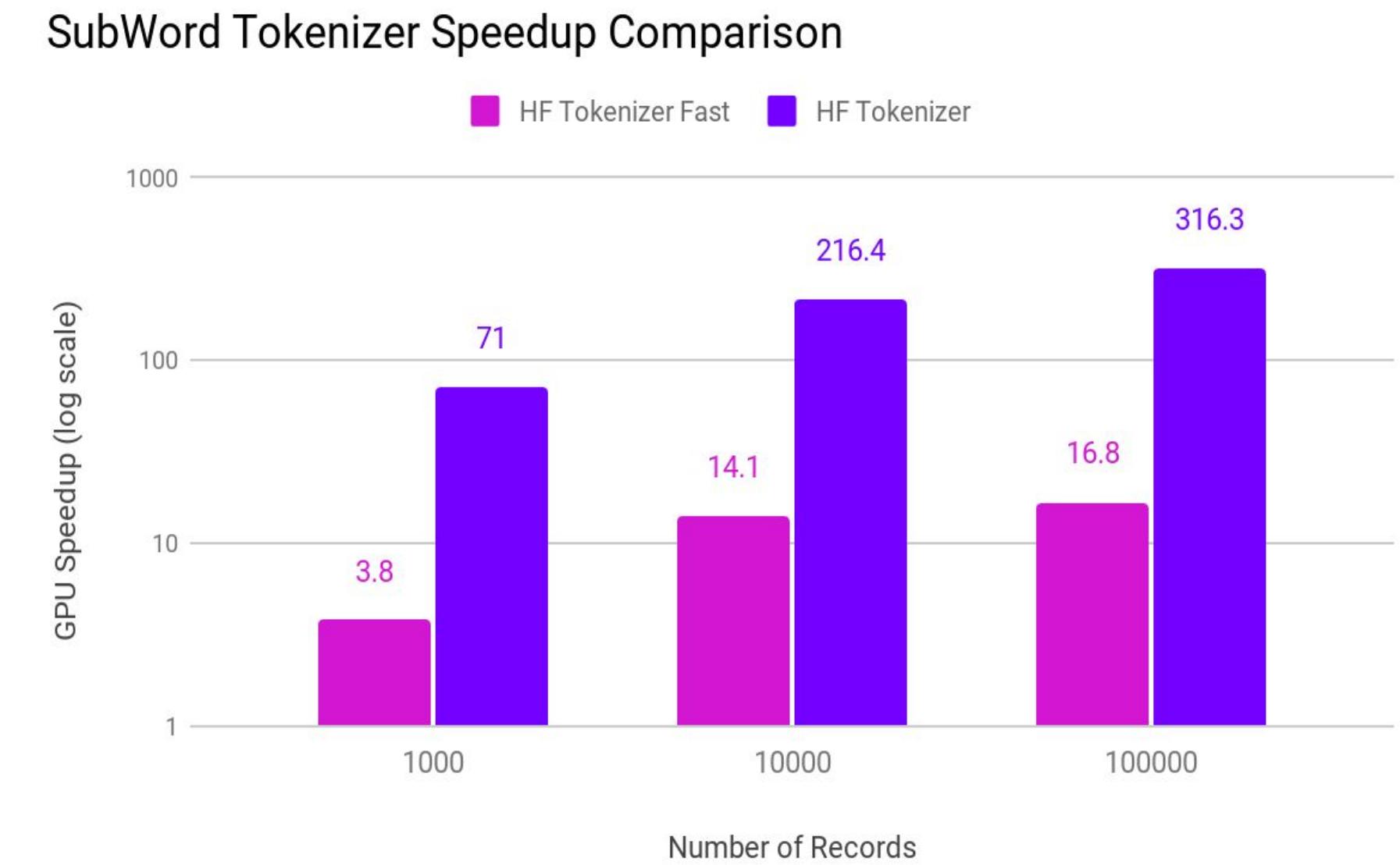
Log Parsing with Missing Values



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
- ▶ Over 316x faster than Python-based Hugging Face tokenizer
- ▶ Nearly 17x faster than new Rust-based Hugging Face tokenizer



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

CLX Query

Query Long-Term Data Store Directly from Splunk

The screenshot shows the Splunk interface with the following details:

- Top Bar:** splunk > App: Clx Search > DEVTEST
- Search Bar:** | clx query="SELECT domain, count(*) as cnt FROM main.dga GROUP BY domain ORDER BY cnt DESC LIMIT 10000" | table domain, cnt
- Results Summary:** 10,000 results (10/29/19 1:00:00.000 PM to 10/30/19 1:31:21.000 PM) No Event Sampling
- Event List:** A list of domains, each with a count value. The first few entries are:
 - pdcqueurgsjsj.pw
 - wcobelvusbujs.cc
 - lketwyitpggxsj.su
 - rfpusradccctv.pw
 - xrqtvsekvdwxw.tw
 - hnbdxgbcffdv.cc
 - oyofyqihjonvfr.cc
 - gflrxihxkpegps.in
 - kucpakwrcilrn.me
 - ewgħtlpjntivpy.cc
 - hvfqeoippqpqs.me
 - nfjuwiesyupbso.in
 - pwgxuuwaitjkb.com
 - lapkgfnbfheveb.me
 - ordqsnkdvwxehp.tw
 - nkrxjgtbteivmi.su
 - eskygbpjotbuc.su
- Bottom Window:** A modal window titled "New Search" containing the same search query and results summary.

RAPIDS

blazingSQL

CLX

Visualization

RAPIDS cuXfilter

GPU Accelerated Cross-Filtering

STREAMLINED FOR NOTEBOOKS

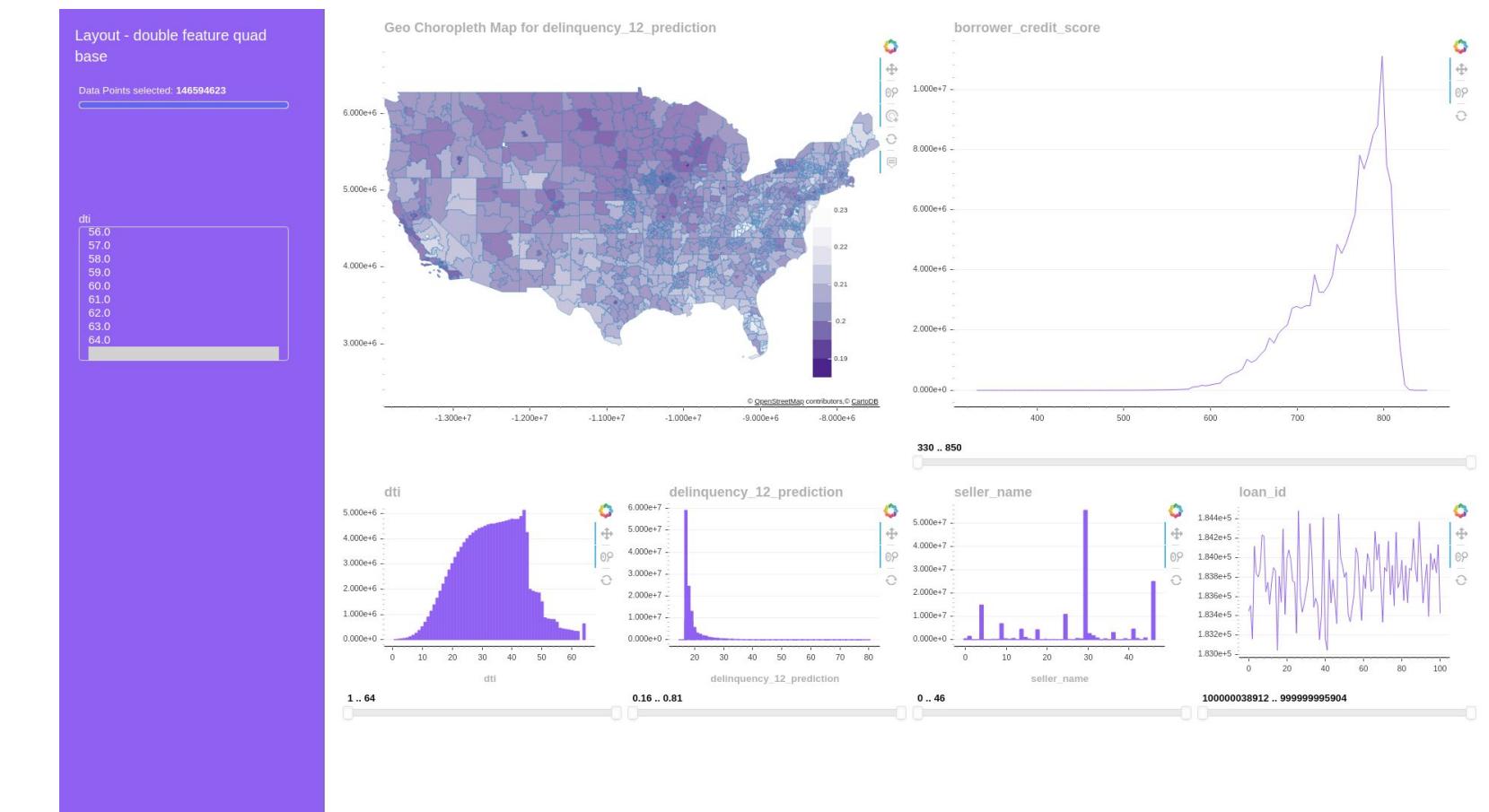
Cuxfilter allows you to visually explore your cuDF dataframes through fully cross-filtered dashboards in less than 10 lines of notebook code.

MINIMAL COMPLEXITY & FAST RESULTS

Select from vetted chart libraries, 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>

Plotly Dash

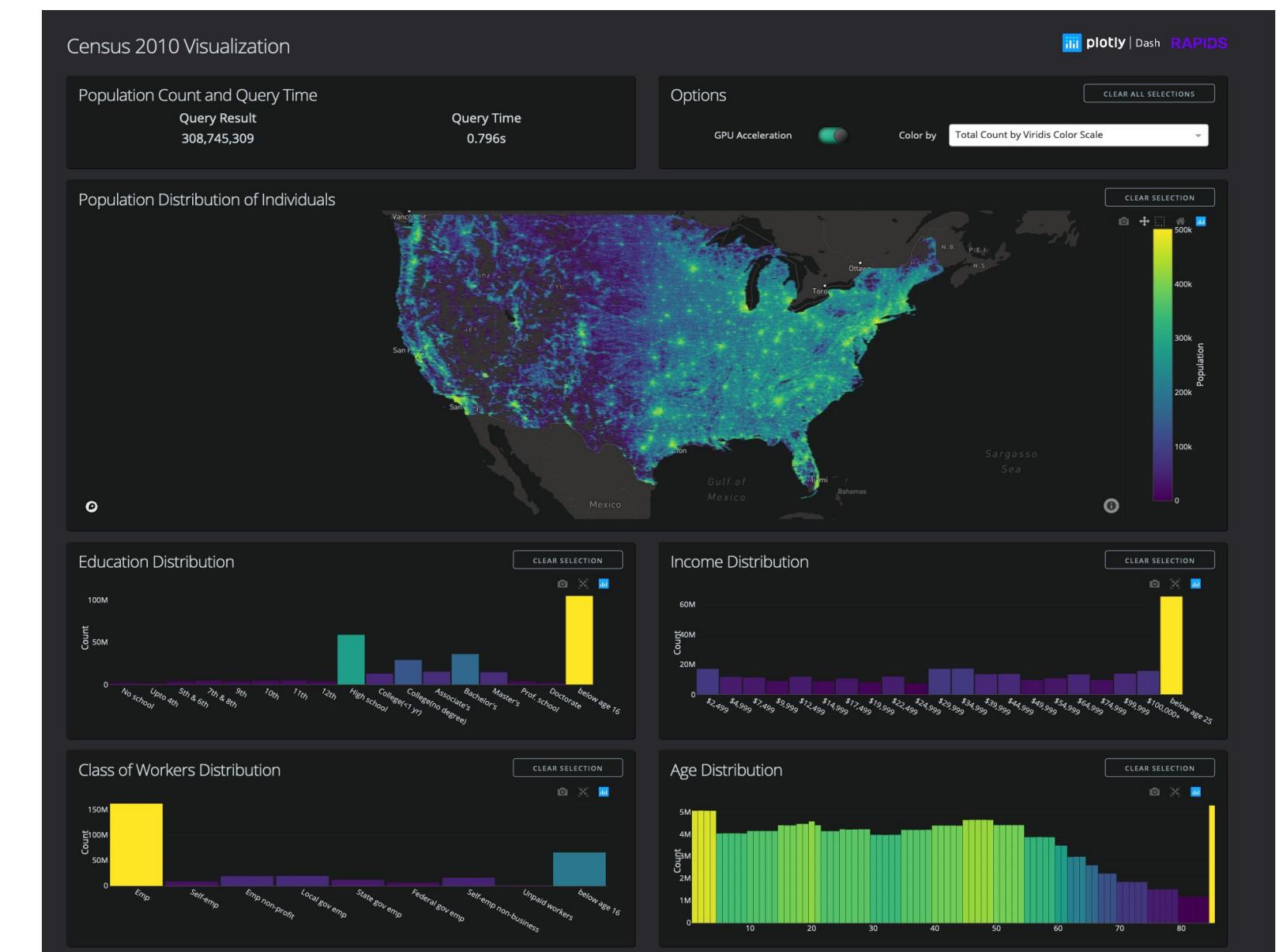
GPU Accelerated Visualization Apps with Python



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

300 MILLION DATAPOINT CENSUS EXAMPLE

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



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

pyViz Community

GPU Accelerated Python Visualizations



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

holoviews.org

Datashader

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

datashader.org



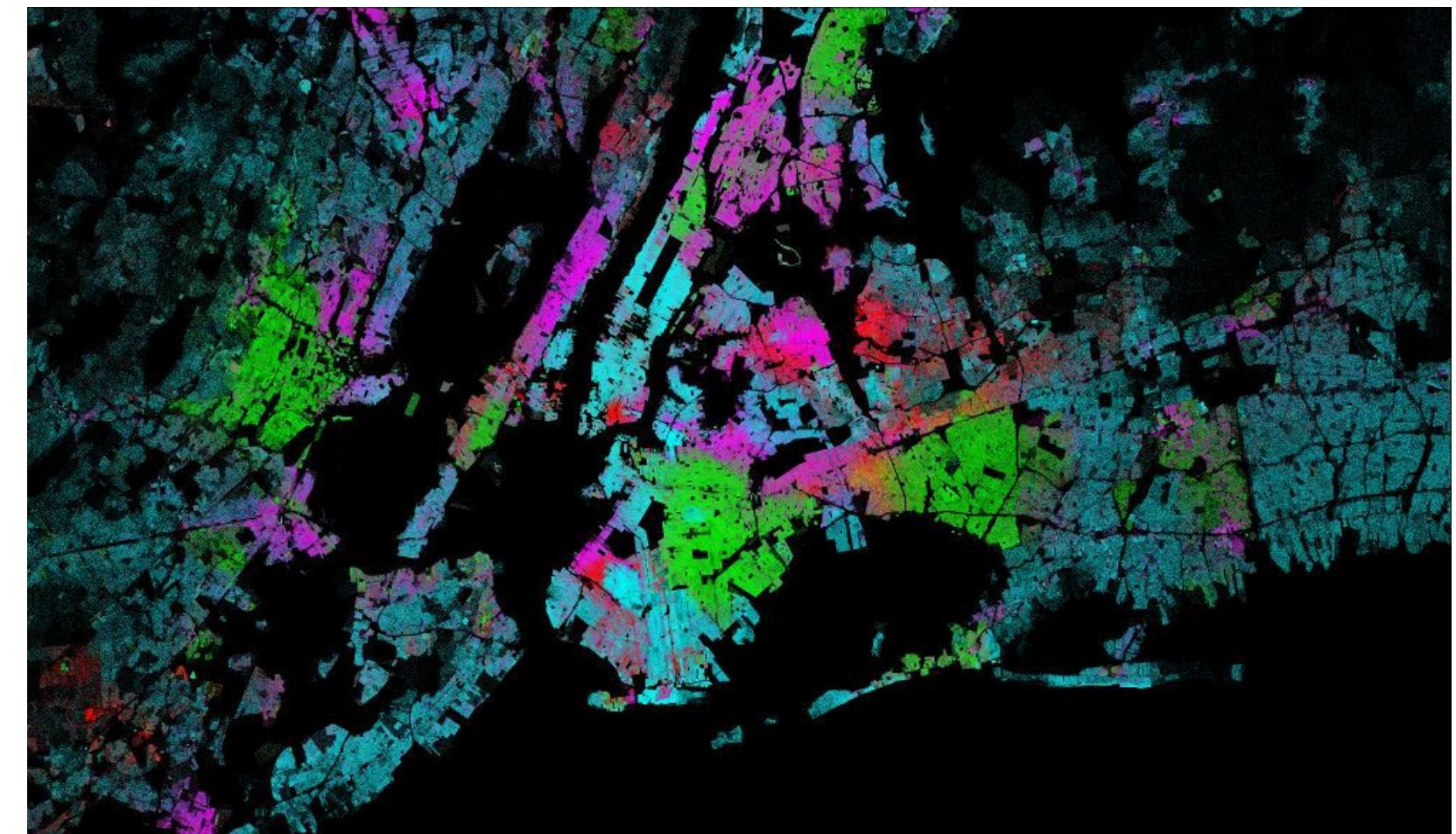
A higher-level plotting API built on HoloViews, work is ongoing to ensure its features can utilize the RAPIDS integration with HoloViews.

hvplot.holoviz.org

bokeh

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

bokeh.org



<https://datashader.org/>

Community

Ecosystem Partners

CONTRIBUTORS



ADOPTERS

Booz | Allen | Hamilton



Uber



Walmart Global Tech



OPEN SOURCE



nuclo



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

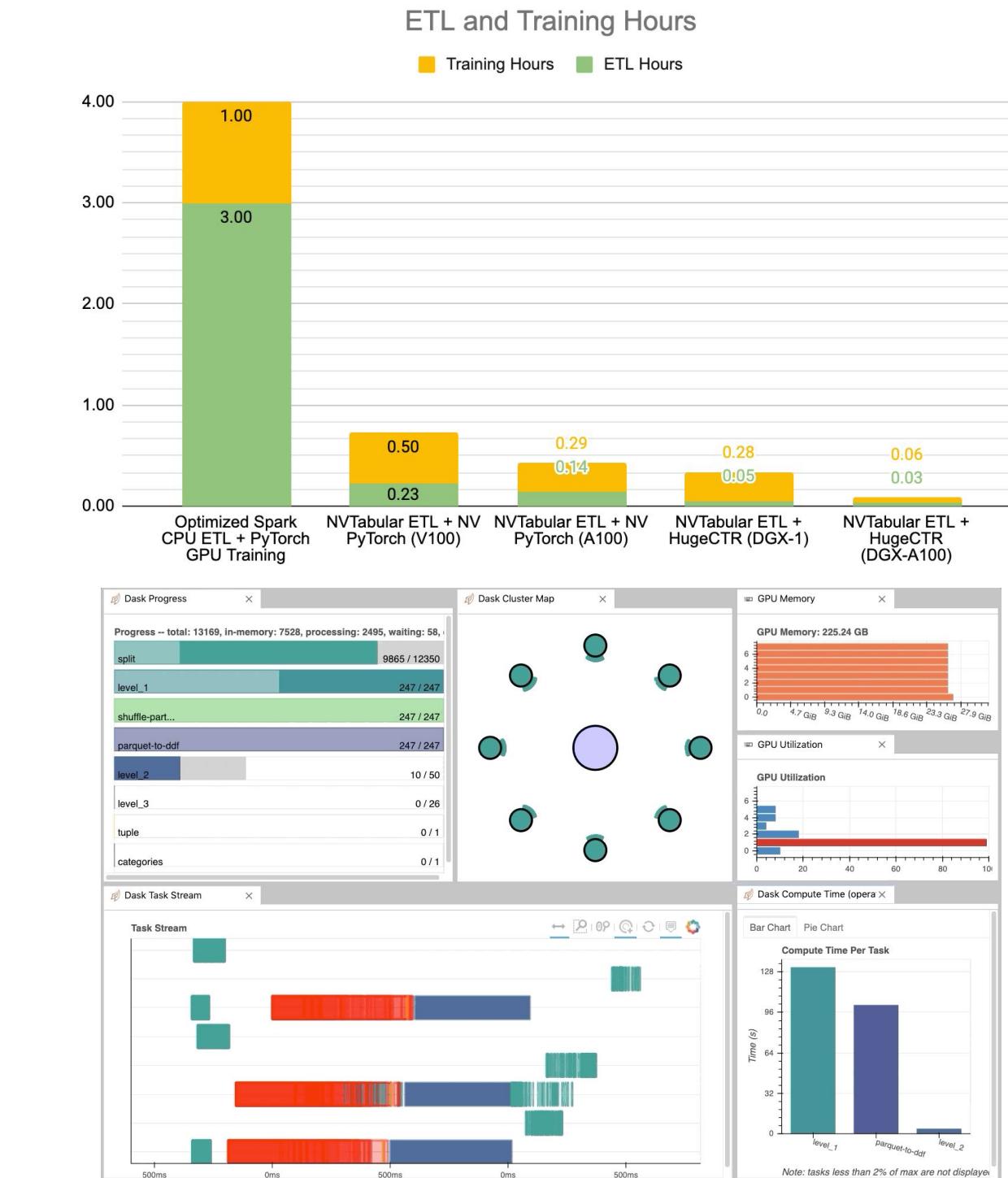
ETL library for recommender systems building off of RAPIDS and Dask

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.



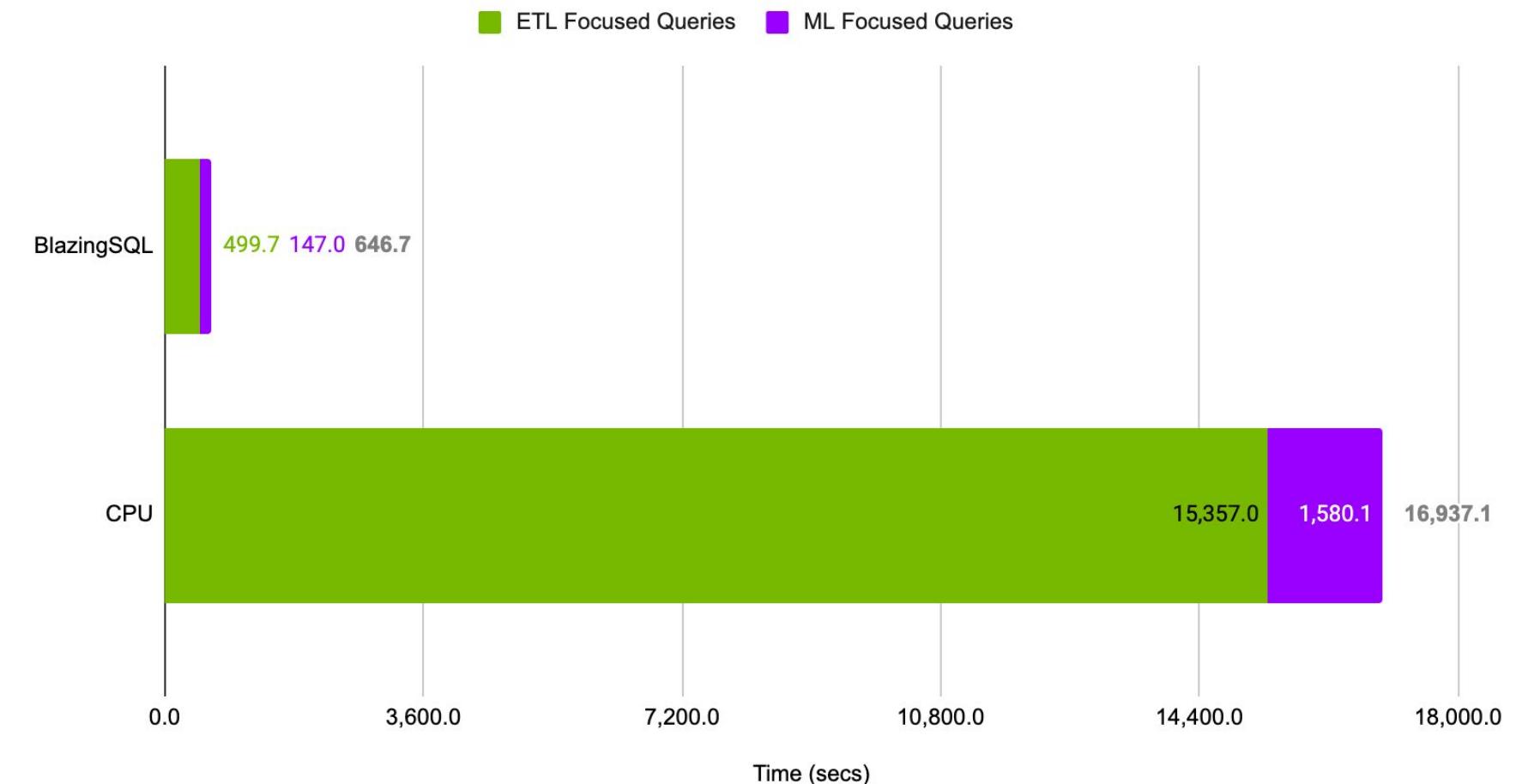
BlazingSQL

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.

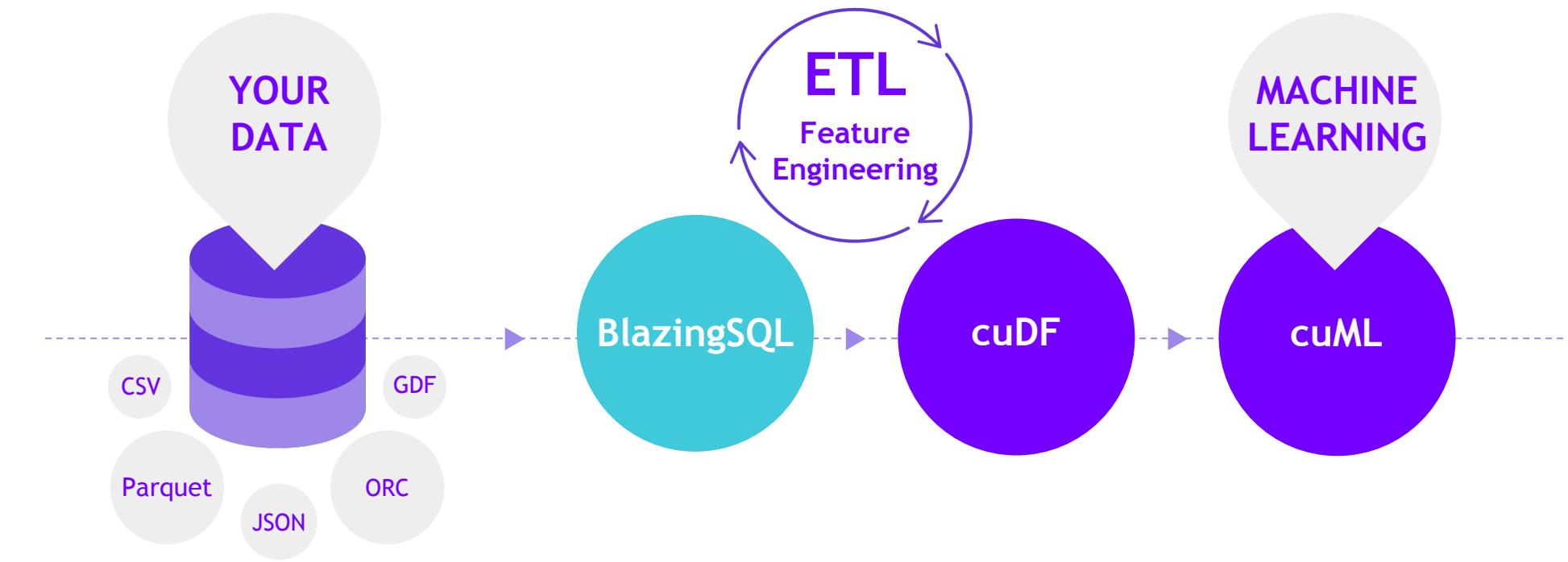
TPCx-BB Results (SF 10k) : BlazingSQL vs CPU Cluster (secs)



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

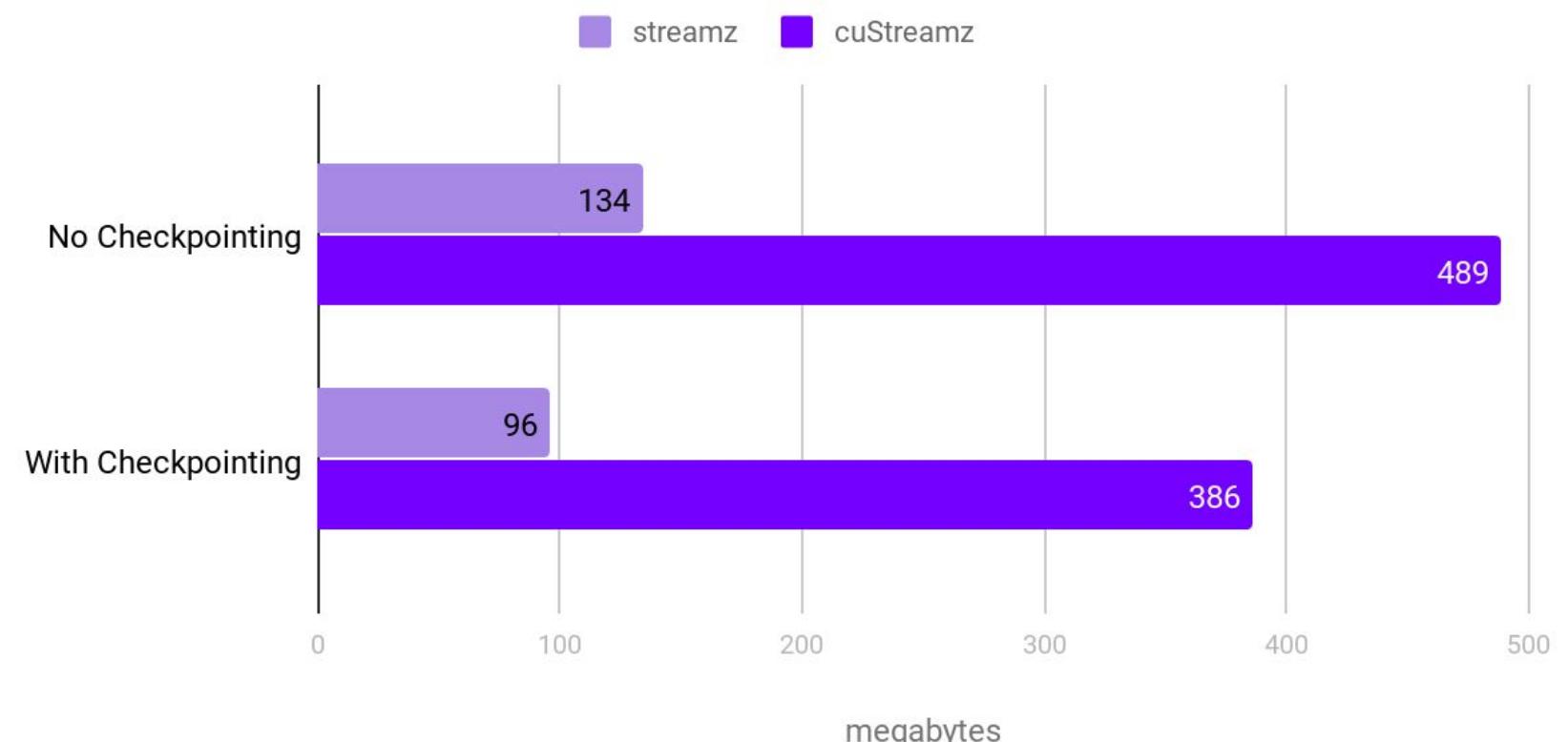
Ingesting Kafka messages to cudf is increased by roughly 3.5 - 4X over standard cpu ingestion. Streaming TCO is lowered by using cheaper VM instance types.

CHECKPOINTING

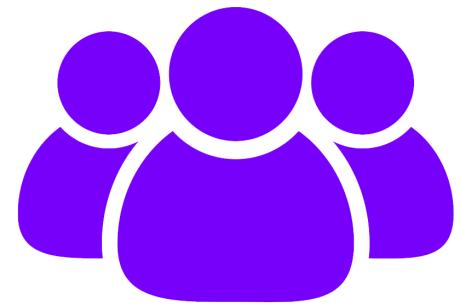
Streaming job version of “where was I?” Streams can gracefully handle errors by continuing to process a stream at the exact point they were before the error occurred.

Kafka throughput

Single Node

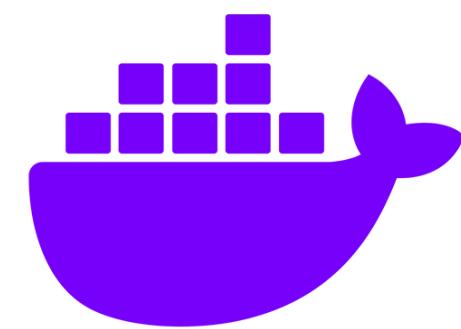


Join the Conversation



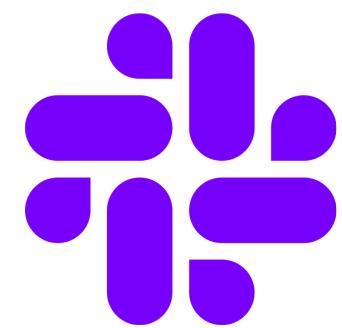
GOOGLE GROUPS

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



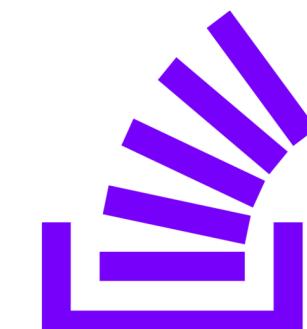
DOCKER HUB

<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 COMPLETED	'dask-xgboost' is deprecated in v0.15 & removed v0.16	Library Deprecation	v0.15	26 August 2020
RGN 1 IN PROGRESS	Stable/Release Branch Renaming to 'main' in v0.15	Git Repo Change	v0.15	26 August 2020
RGN 2 COMPLETED	v0.15 No CUDA 11 Release for 'clx'	Release Change	v0.15	26 August 2020
RGN 4 IN PROGRESS	v0.15 Release Delay for 'cuxfilter'	Release Change	v0.15	26 August 2020
RSN 2 COMPLETED	EOL Python 3.6 & CUDA 10.0 in v0.14	Platform Support Change	v0.14 & v0.15	13 July 2020
RSN 3 COMPLETED	Support for Python 3.8 in v0.15	Platform Support Change	v0.15	17 July 2020
RSN 4 COMPLETED	Support for CUDA 11.0 in v0.15	Platform Support Change	v0.15	26 August 2020

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

Getting Started

5 Steps to Getting Started with RAPIDS

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

Easy Installation

Interactive Installation Guide

RAPIDS RELEASE SELECTOR

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

NOTICES

- ⚠️ RAPIDS 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 a purple header "cuDF" and a list of libraries: clx, cudf (selected), cugraph, cuml, cusignal, cuspatial, cufilter, libcudf, libcuml, libnvlstrings, nvstrings, rmm, and ~~wrapping~~ cuDF. The main content includes sections on "What are these Libraries?", "When to use cuDF and Dask-cuDF", and "Object Creation". It features code snippets in a Jupyter-style cell.

Right Screenshot: The page title is "Welcome to cuDF's documentation!". The left sidebar has a purple header "cuDF" and a dropdown "stable (0.13)". The main content includes a "CONTENTS:" section with a tree view of topics like API Reference, 10 Minutes to cuDF and Dask-cuDF, Multi-GPU with Dask-cuDF, etc., and a "Developer Documentation" section.

<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 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 code for creating a cuDF Series object. At the bottom right of the page, there is a 'RAPIDS' logo.

Docs >> 10 Minutes to cuDF and Dask-cuDF

View page source

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>

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 AI
Open GPU Data Science

RAPIDS RELEASES DATAFRAMES MACHINE LEARNING DATA VISUALIZATION GRAPH ANALYTICS | LEARN MORE [Follow](#)

Reading Larger than Memory CSVs with RAPIDS and Dask
RAPIDS and Dask make it easy to load larger than memory datasets. Learn how with examples.
Nick Becker Oct 22 · 2 min read

RAPIDS CuGraph: NetworkX Compatibility
RAPIDS cuGraph adds NetworkX Graph and DiGraph objects as valid input data types for graph algorithms
Brad Rees Oct 2 · 4 min read

Tutorial: Hyperparameter Optimization (HPO) with RAPIDS on AWS SageMaker
12x speedup in wall clock time and 4.5x reduction in cost when comparing GPU to CPU running HPO jobs in SageMaker.
Miro Enev Sep 28 · 2 min read

RAPIDS Anywhere with Tailscale—My Mobile Device has an RTX 3090
It's never been easier to get started with GPUs and RAPIDS, and with Tailscale, you can kick off RAPIDS workflows from anywhere.
Josh Patterson Sep 24 · 5 min read

<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 tagline "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 Community Notebooks

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

Intermediate Notebooks:

Folder	Notebook Title	Description
examples	DBSCAN_Demo_FULL	This notebook shows how to use DBSCAN algorithm and its GPU accelerated implementation present in RAPIDS.
examples	Dask_with_cuDF_and_XGBoost	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" and it has 47 subscribers. The "HOME" tab is selected. Below it, the "Uploads" section shows four video thumbnails with titles and durations:

- "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" is partially visible on the right.

Below each video thumbnail, there is a timestamp indicating when the video was uploaded (e.g., 1 month ago) and the number of views.

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

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

RAPIDS

How Do I Get the Software?



GITHUB

<https://github.com/rapidsai>



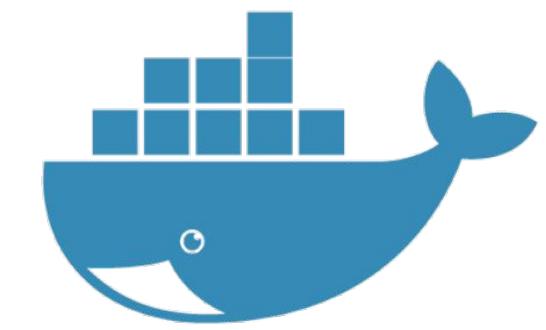
ANACONDA

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



NGC

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



DOCKER

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

Deploy RAPIDS Everywhere

Focused on Robust Functionality, Deployment, and User Experience



Amazon SageMaker



Azure



Azure Machine
Learning



Google Cloud



Alibaba Cloud



Cloud
Dataproc



SaturnCloud



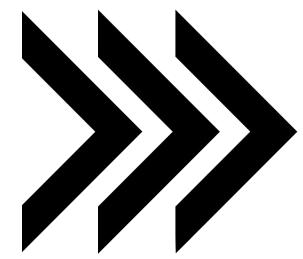
Kubeflow

COILED

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

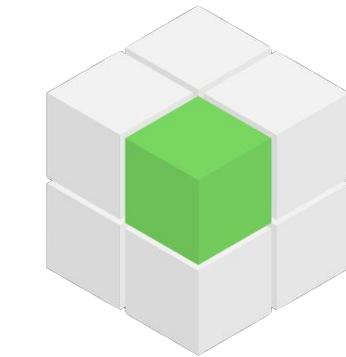
RAPIDS

<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