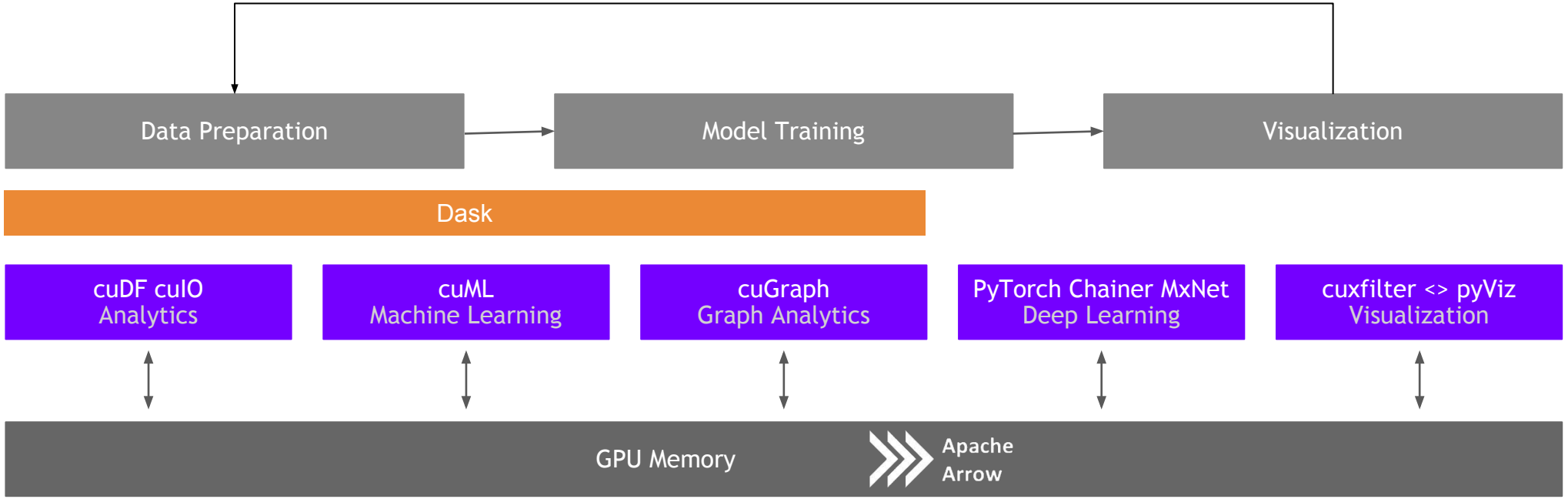
RAPIDS

The Platform Inside and Out
Release 0.13

Joshua Patterson – Director, RAPIDS Engineering

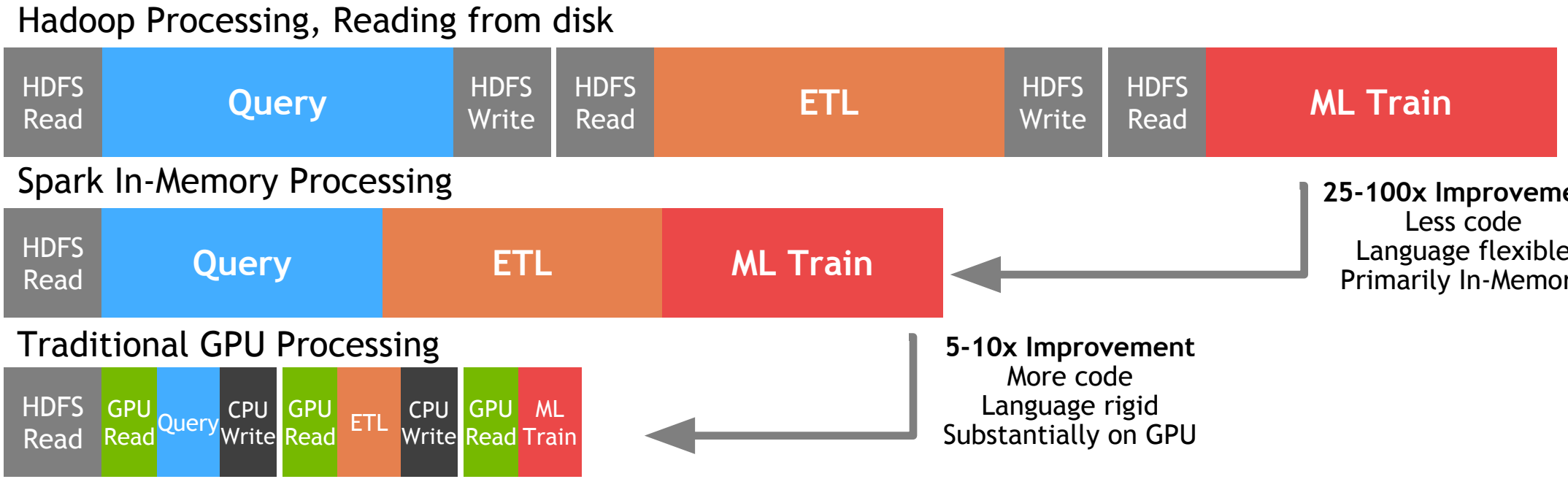
RAPIDS

End-to-End Accelerated GPU Data Science



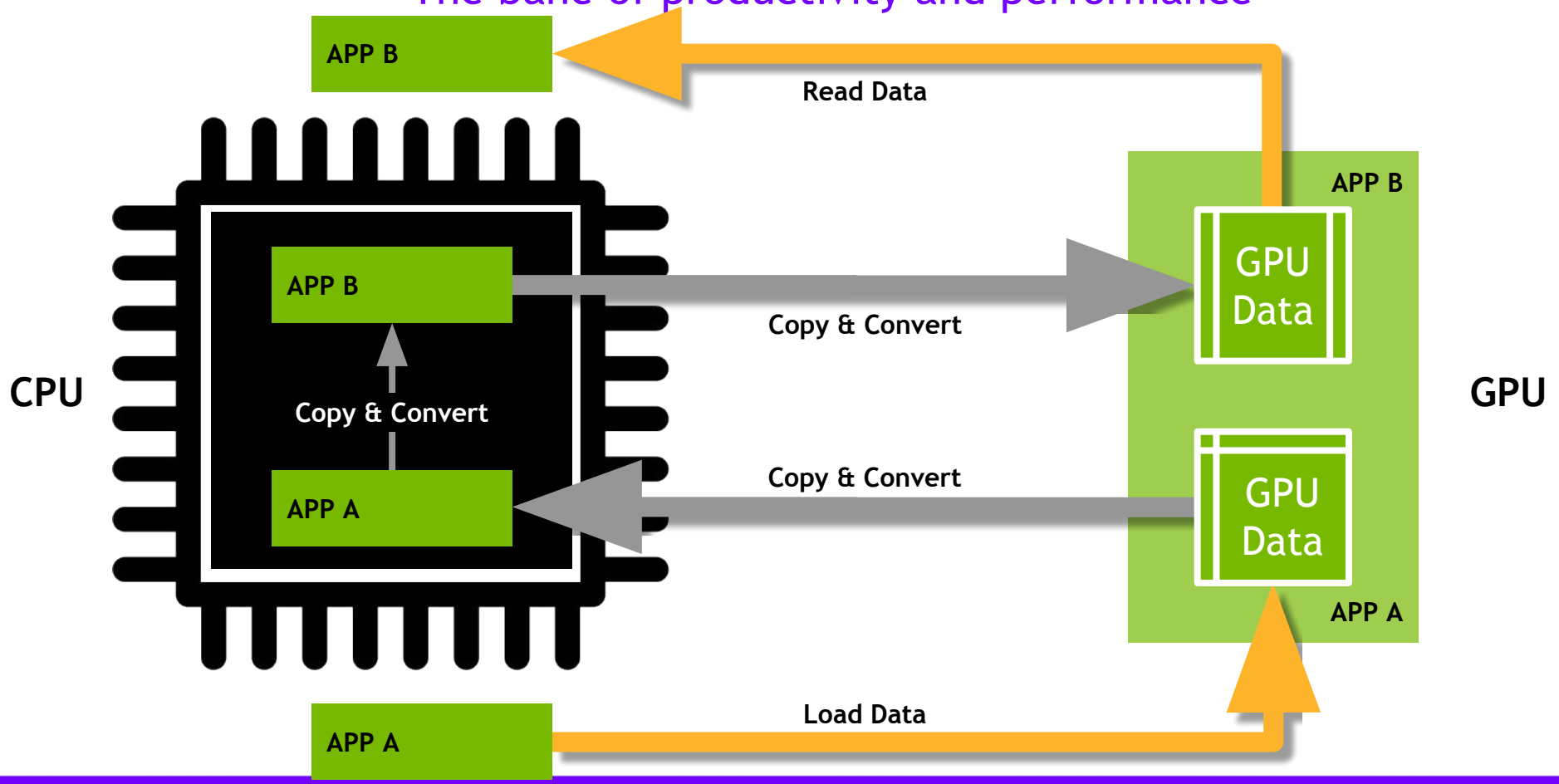
Data Processing Evolution

Faster data access, less data movement



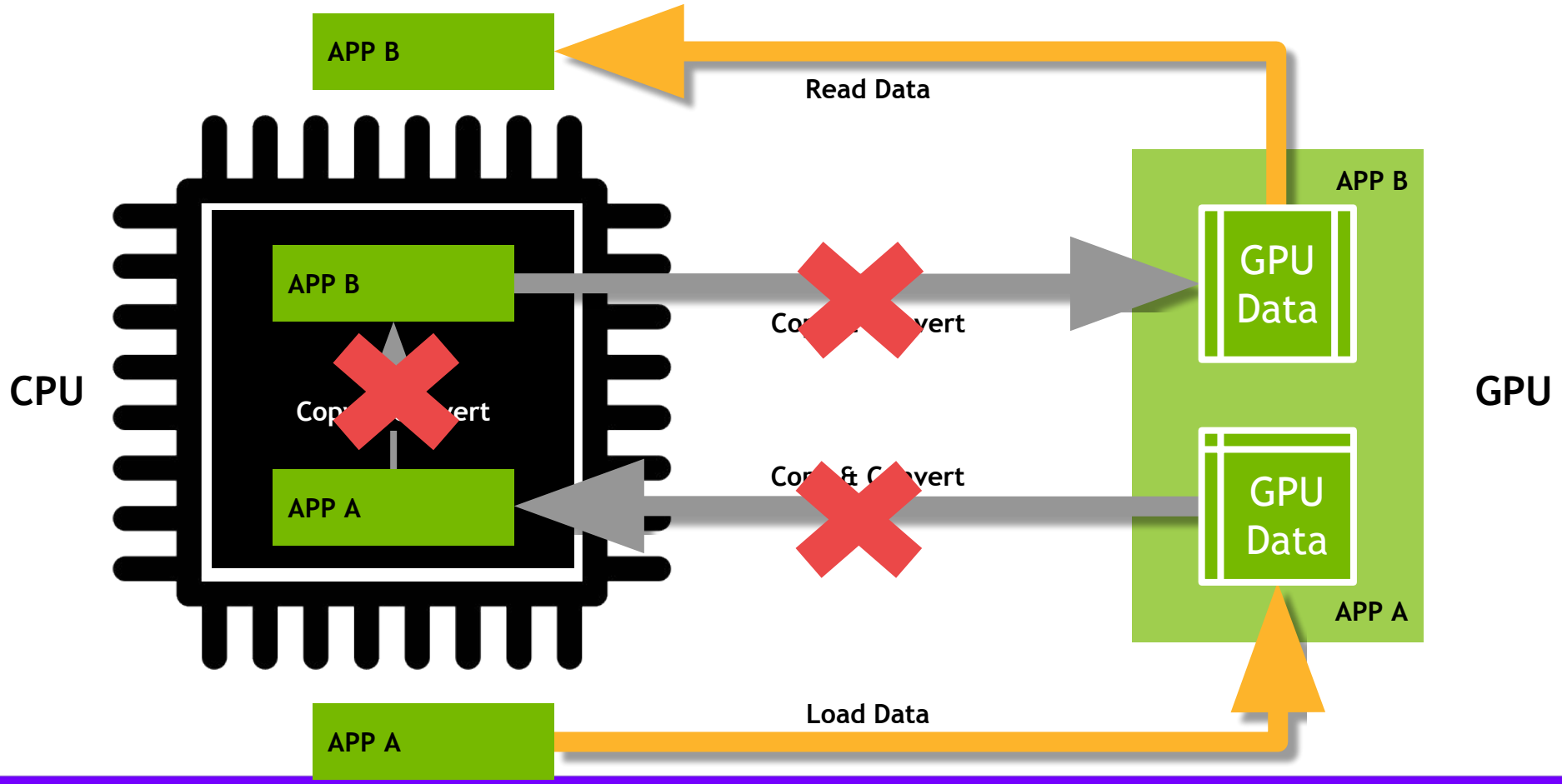
Data Movement and Transformation

The bane of productivity and performance

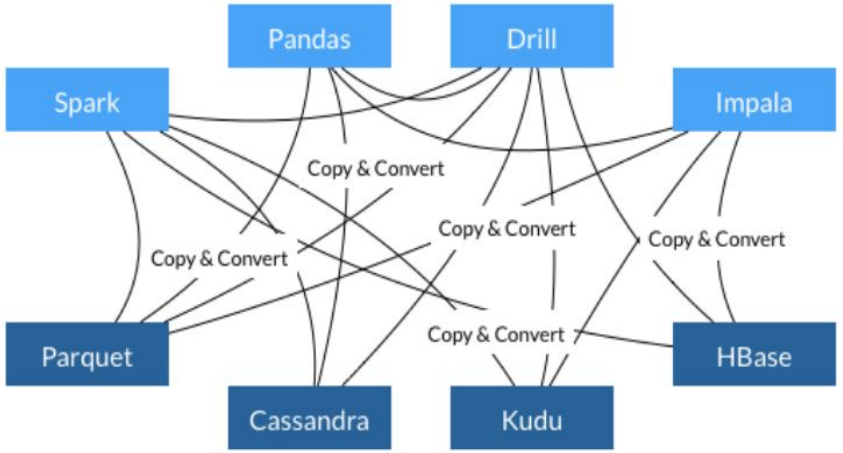


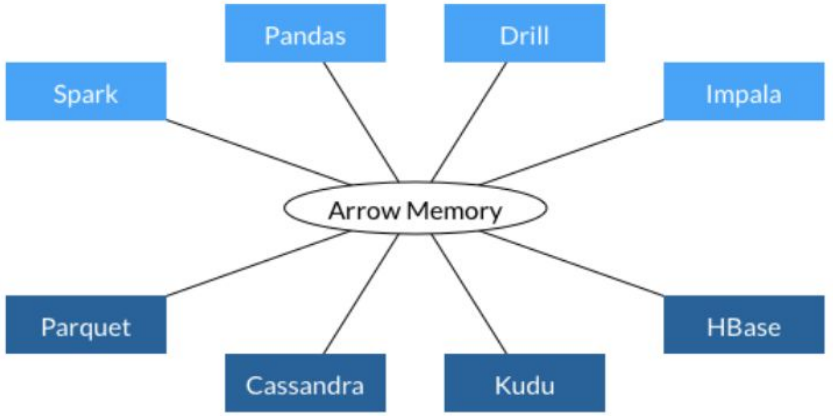
Data Movement and Transformation

What if we could keep data on the GPU?



Learning from Apache Arrow





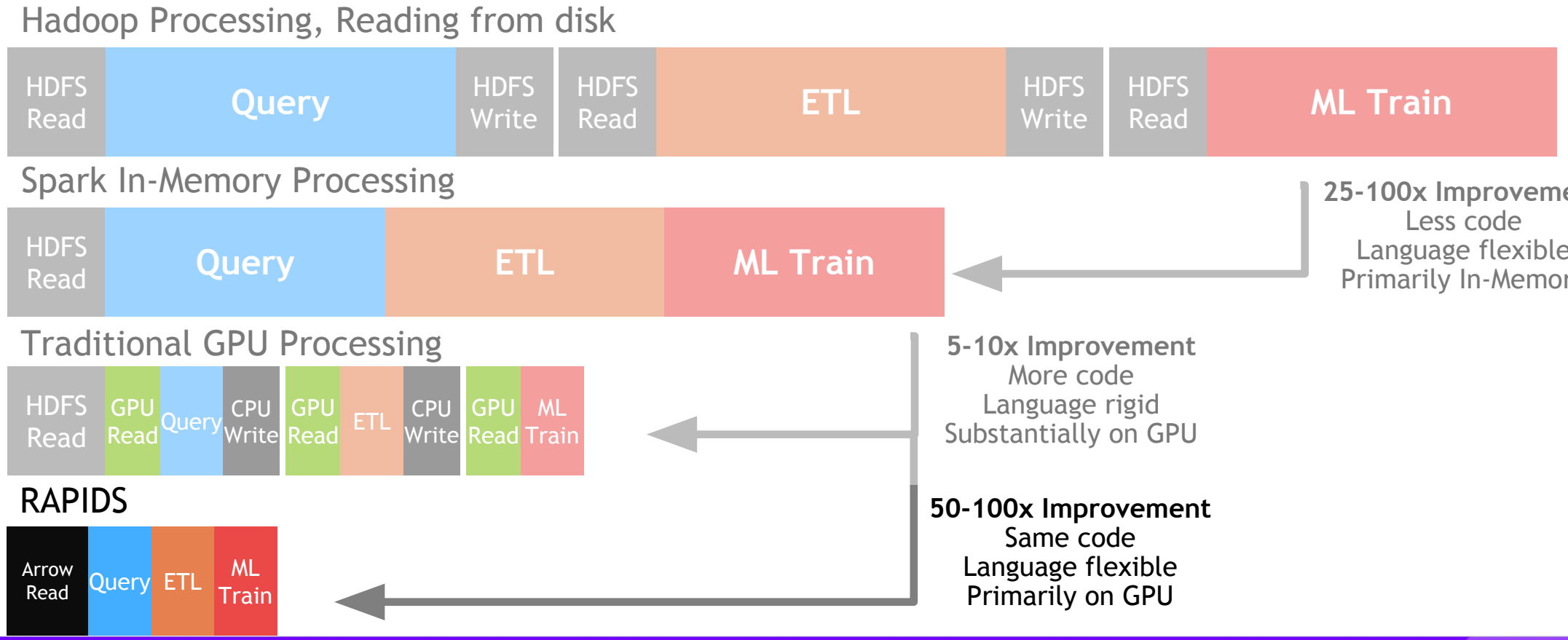
- Each system has its own internal memory format
- 70-80% computation wasted on serialization and deserialization
- Similar functionality implemented in multiple projects

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

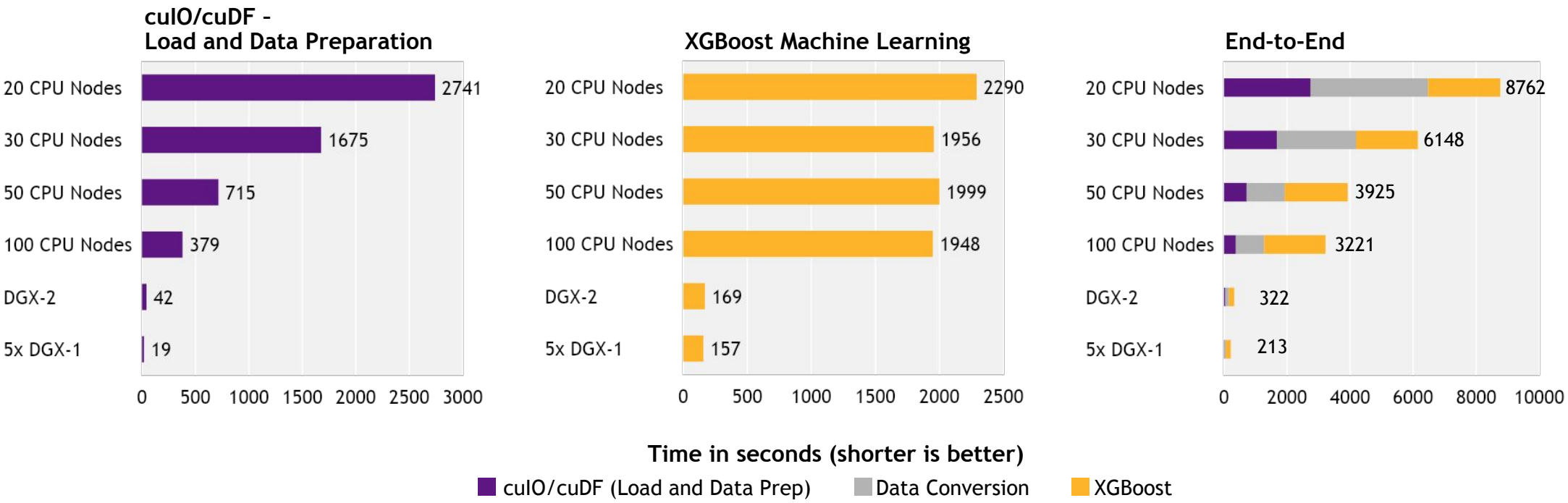
From Apache Arrow Home Page - https://arrow.apache.org/

Data Processing Evolution

Faster data access, less data movement



Faster Speeds, Real-World Benefits



Benchmark

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

CPU Cluster Configuration

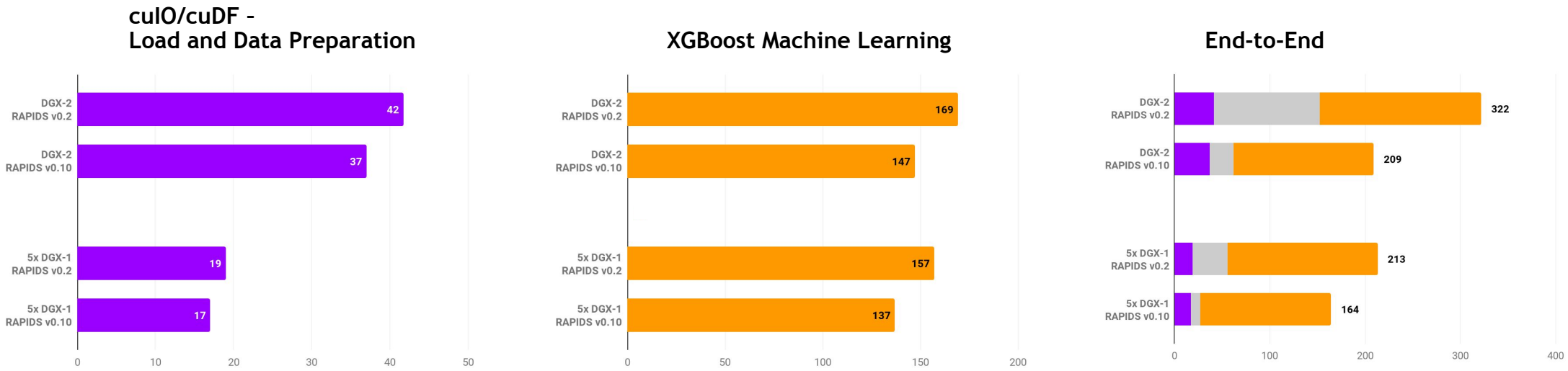
CPU nodes (61 GiB memory, 8 vCPUs, 64-bit platform), Apache Spark

DGX Cluster Configuration

5x DGX-1 on InfiniBand network

Faster Speeds, Real-World Benefits

Improving Over Time



Benchmark

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

CPU Cluster Configuration

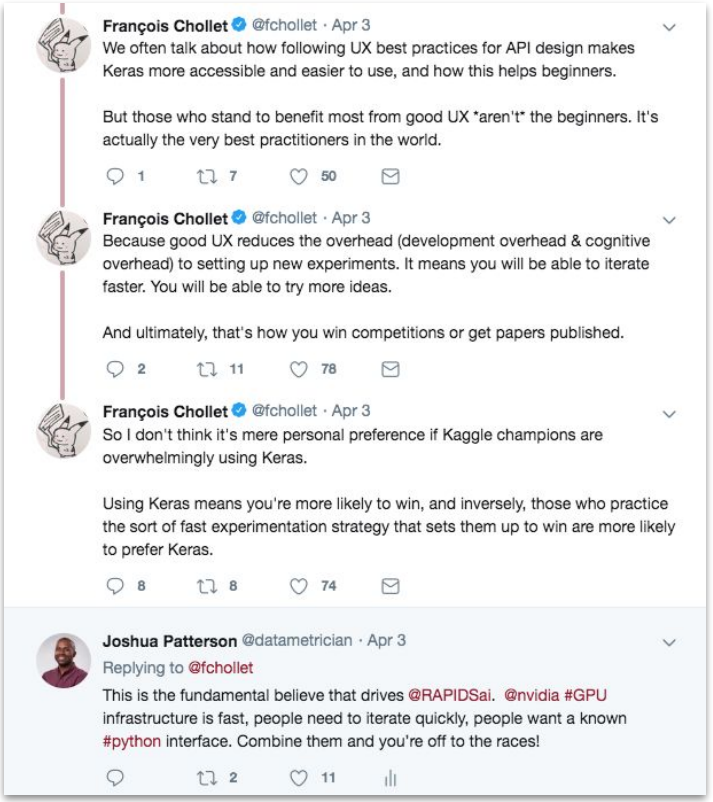
CPU nodes (61 GiB memory, 8 vCPUs, 64-bit platform), Apache Spark

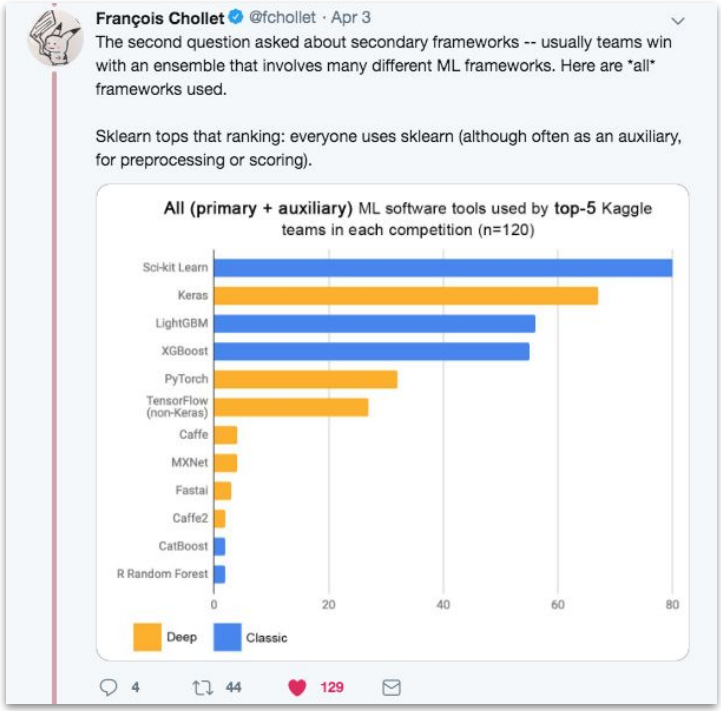
DGX Cluster Configuration

5x DGX-1 on InfiniBand network

Speed, Ease of Use, and Iteration

The Way to Win at Data Science





kaggle

RAPIDS 0.13 Release Summary

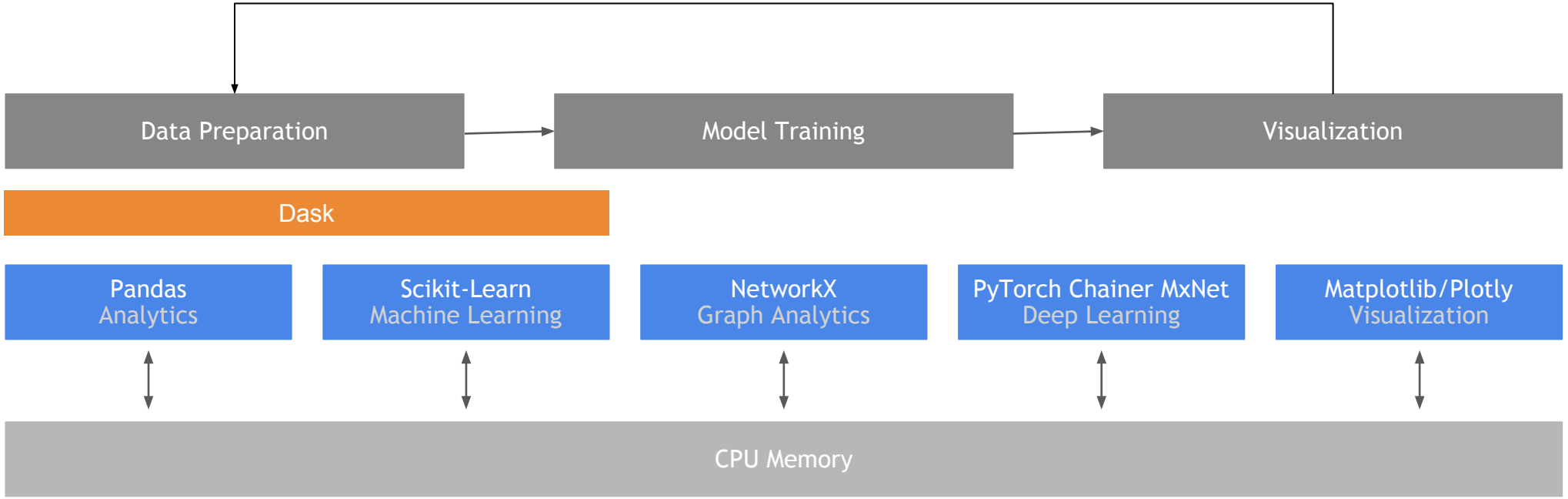
What's New in Release 0.13?

- **cuDF** Dataframes library python code base now ported to use the libcudf++ API, adds expanded groupby aggregations, join methods, concatenate optimizations, distributed multi column sorting and multi column hash repartitioning
- **cuML** machine learning library adds new Dask API to XGBoost 1.0, new configurable types for estimators, and multi-node, multi-GPU support for linear models (PCA, tSVD, OLS, Ridge)
- **cuGraph** graph analytics library adds betweenness centrality, K-Truss community detection, and code refactoring
- **UCX-Py** focused on resolving Multi-Node Multi-GPU InfiniBand tests
- **BlazingSQL** adds ROUND(), CASE with Strings, and for distributed queries AVG() support
- **cuSignal** adds conda install support, polyphase resampler, and new acoustics module
- **cuSpatial** adds batched cubic spline interpolation for trajectories and major documentation improvements

RAPIDS Core

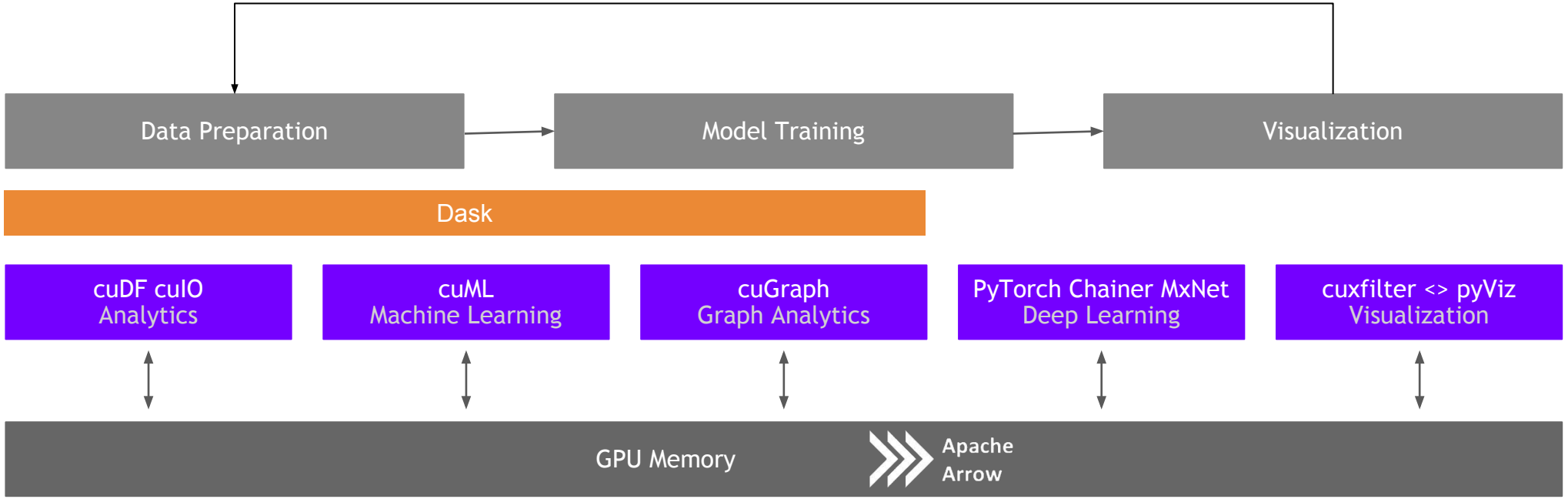
Open Source Data Science Ecosystem

Familiar Python APIs



RAPIDS

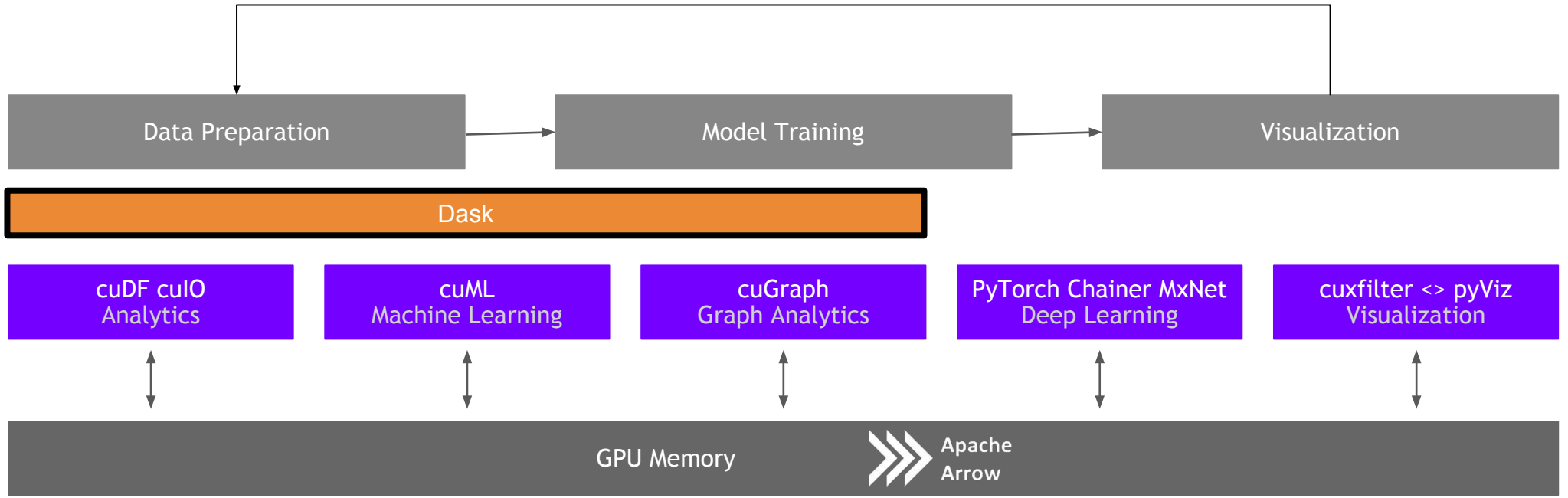
End-to-End Accelerated GPU Data Science



Dask

RAPIDS

Scaling RAPIDS with Dask



Why Dask?

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

Deployable

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

Easy Scalability

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

Popular

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

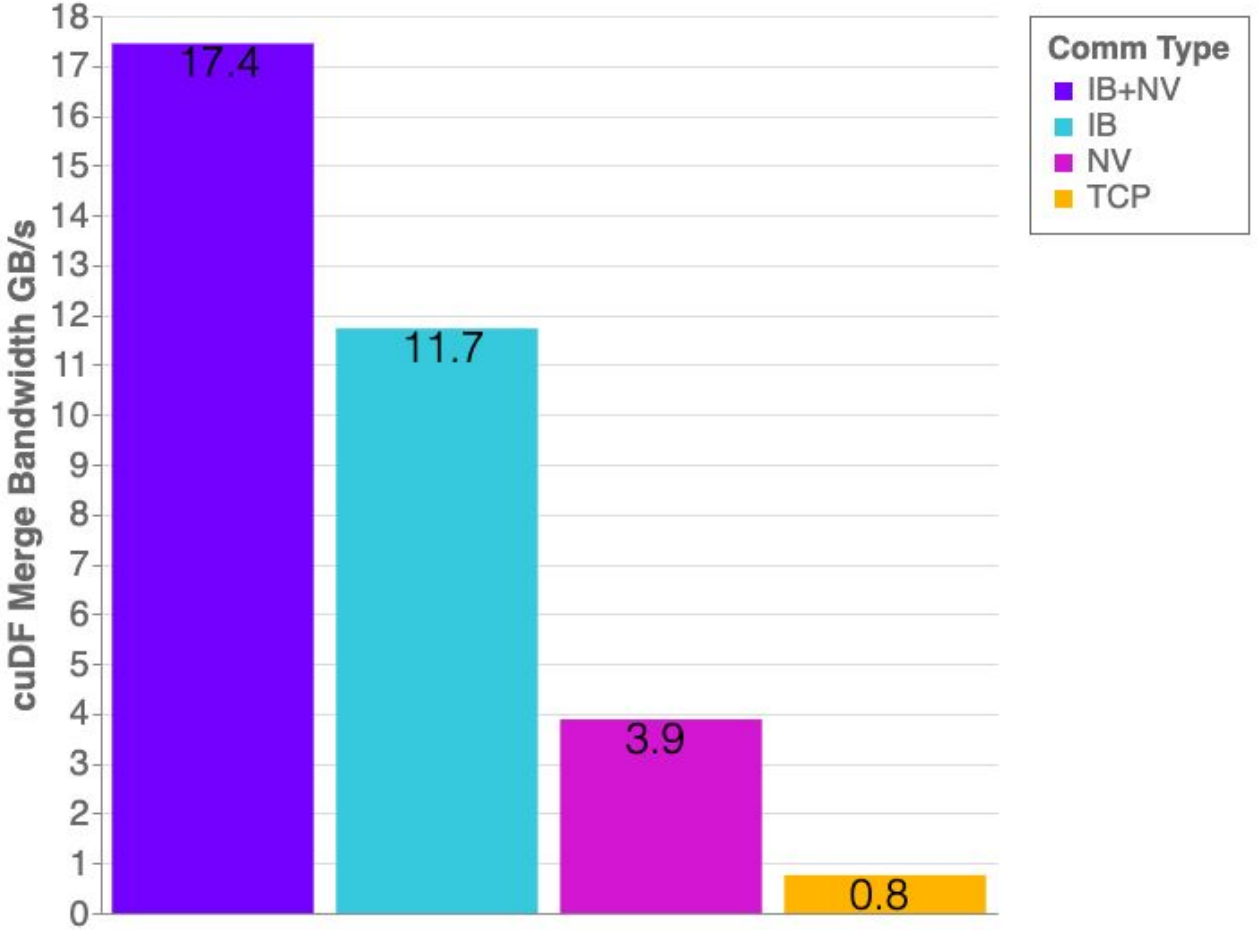
Why OpenUCX?

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 v0.13, UCX-PY 0.13

Running on NVIDIA DGX-1 (8GPUs):

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 Up / Accelerate

RAPIDS and Others

Accelerated on single GPU

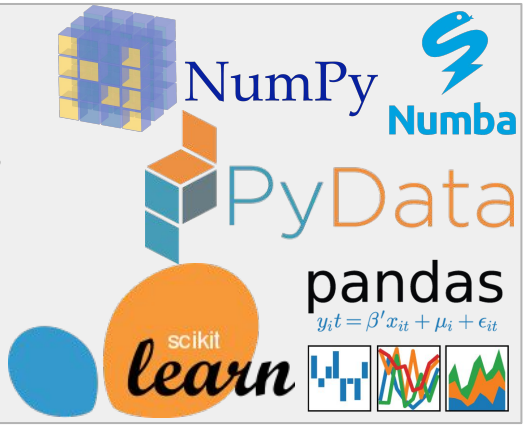
NumPy -> CuPy/PyTorch/..
Pandas -> cuDF
Scikit-Learn -> cuML
Numba -> Numba

RAPIDS

PyData

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

Single CPU core
In-memory data



Scale out with RAPIDS + Dask with OpenUCX

Scale Up / Accelerate

Scale out / Parallelize

RAPIDS and Others

Accelerated on single GPU

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

RAPIDS

RAPIDS + Dask with OpenUCX

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

RAPIDS

DASK

PyData

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

Single CPU core
In-memory data

NumPy Numba PyData pandas scikit learn

Dask

Multi-core and Distributed PyData

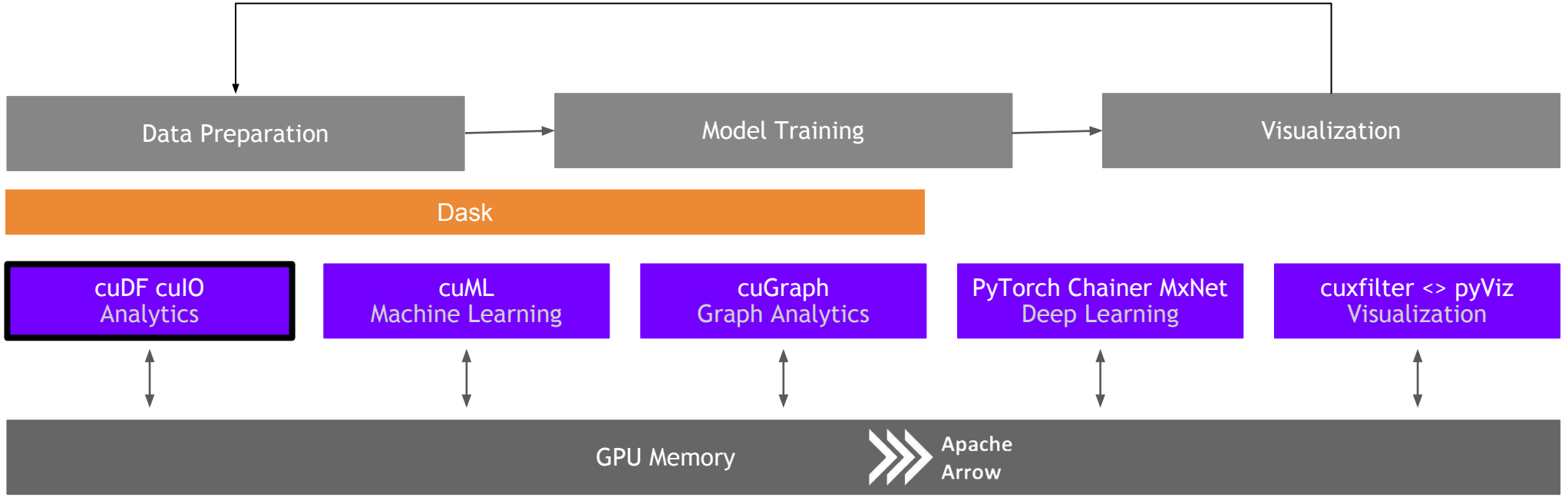
NumPy -> Dask Array
Pandas -> Dask DataFrame
Scikit-Learn -> Dask-ML
… -> Dask Futures

DASK

cuDF

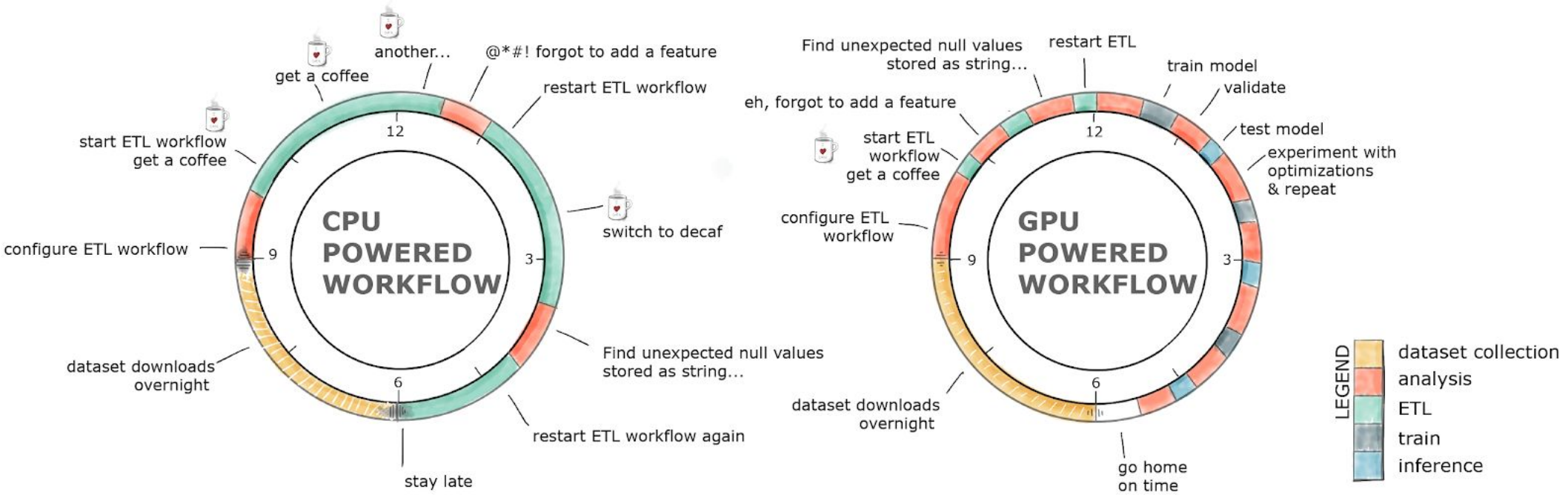
RAPIDS

GPU Accelerated data wrangling and feature engineering

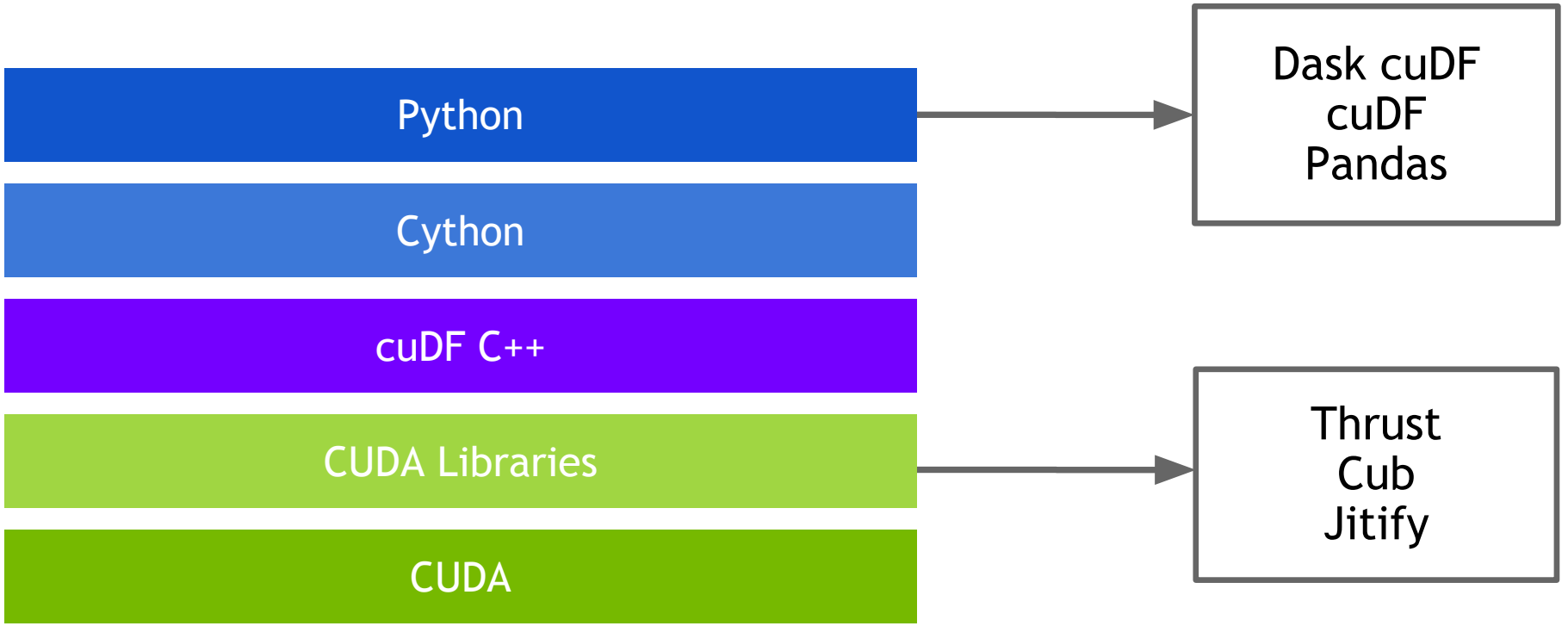


GPU-Accelerated ETL

The average data scientist spends 90+% of their time in ETL as opposed to training models



ETL Technology Stack



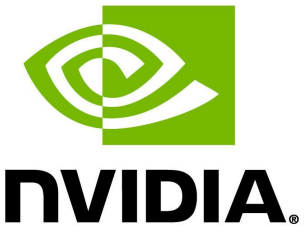
ETL: the Backbone of Data Science

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<table>
gather(table_view const& input,
       column_view const& gather_map, …)
{
    // return a new table containing
    // rows from input indexed by
    // gather_map
}
```



ETL: the Backbone of Data Science

cuDF is…

```
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]:

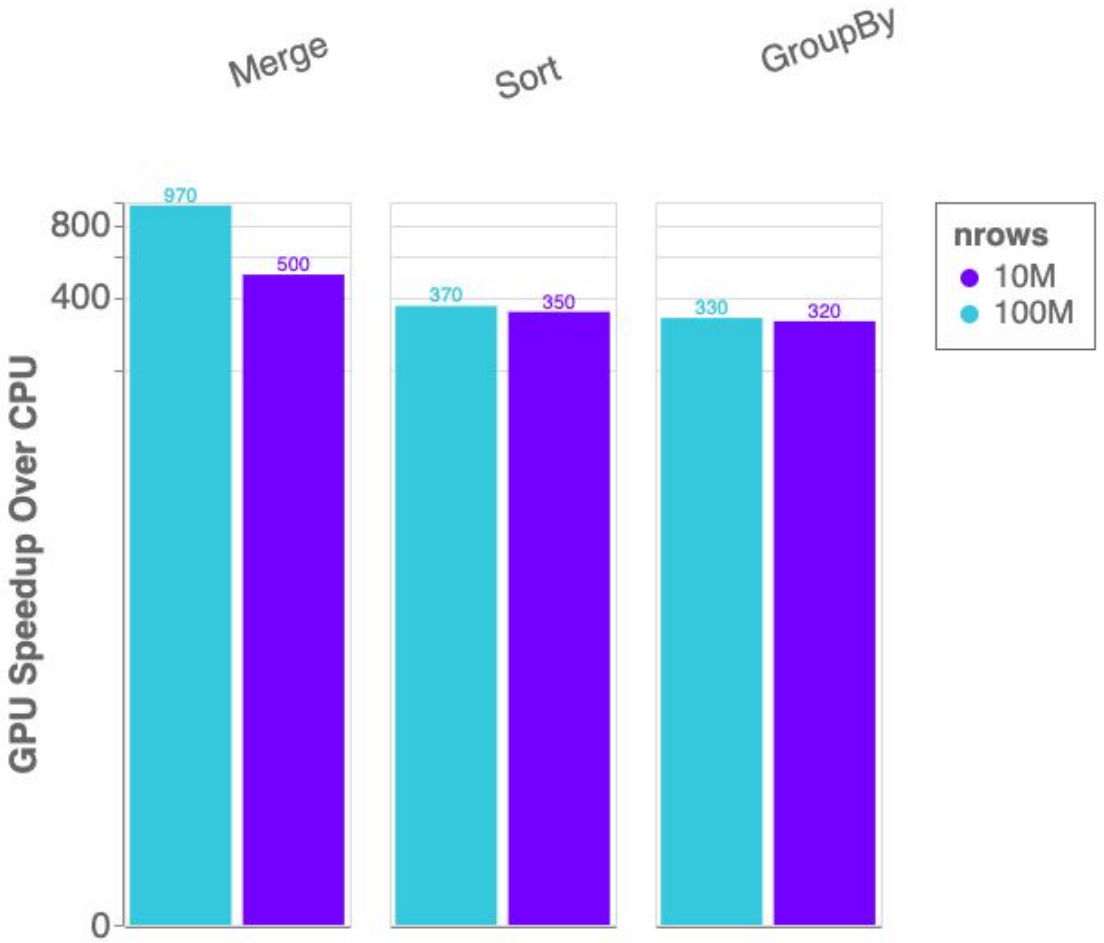
	User_ID	Product_ID	Gender	Age	Occupation	City_Category	Stay_In_Current_City_Years	Marital_Status	Product_Cat
0	1000001	P00069042	F	0-17	10	A	2	0	3
1	1000001	P00248942	F	0-17	10	A	2	0	1
2	1000001	P00087842	F	0-17	10	A	2	0	12
3	1000001	P00085442	F	0-17	10	A	2	0	12
4	1000002	P00285442	M	55+	16	C	4+	0	8

```
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()
```

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

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

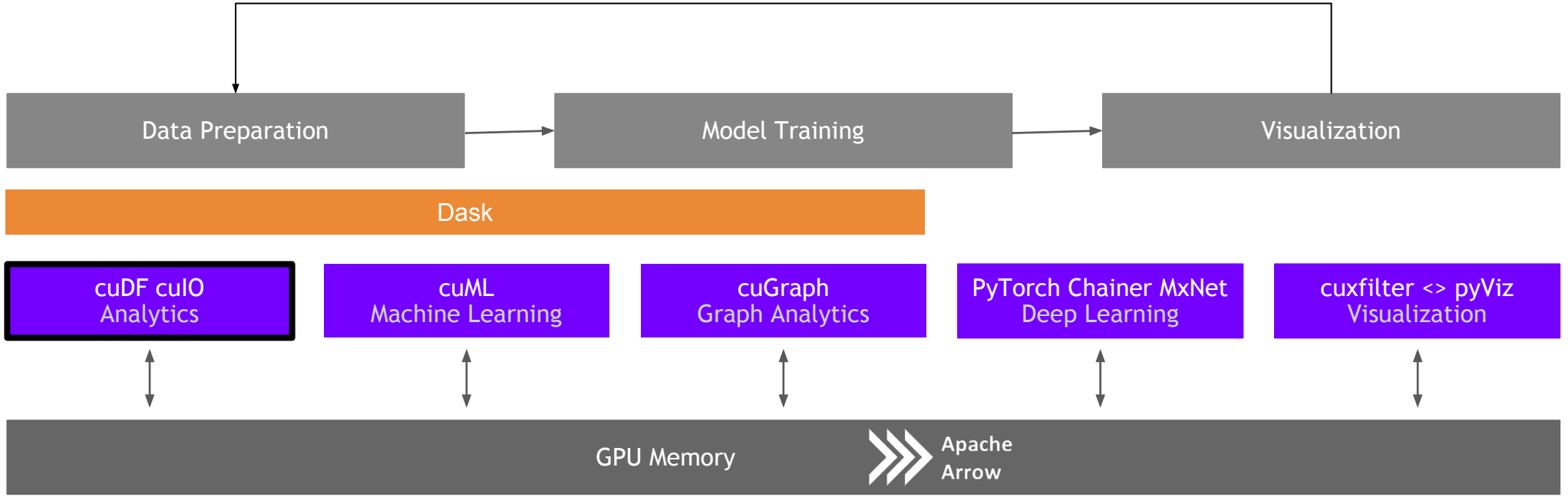
DataFrames: 2x int32 columns key columns, 3x int32 value columns

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

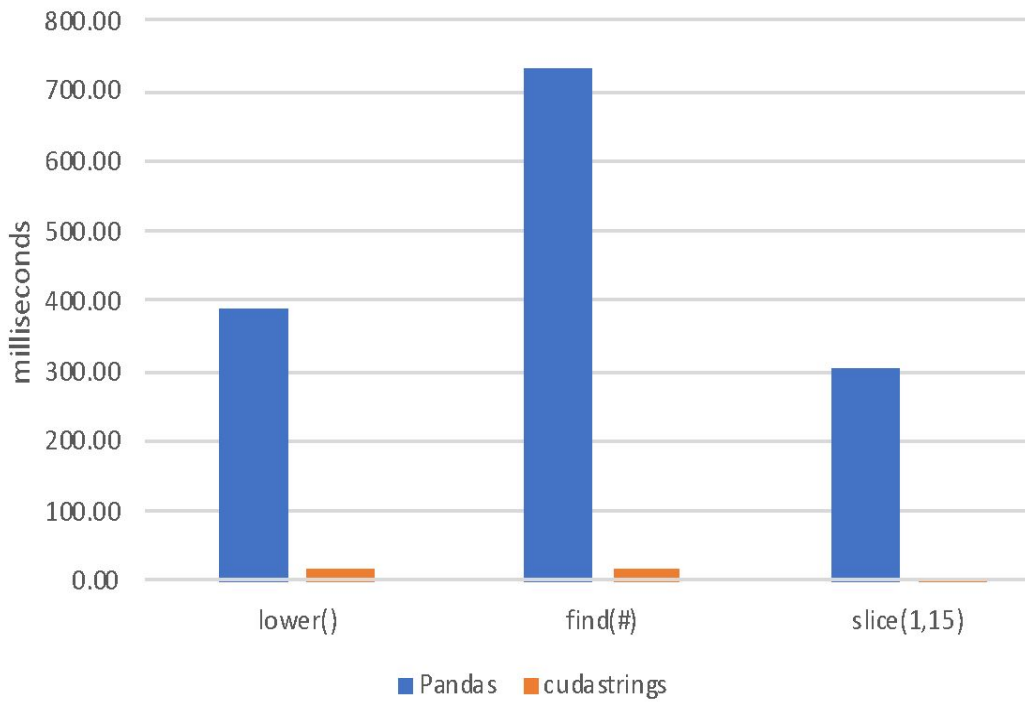
String Support

Current v0.13 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 v0.14+ String Support

- Further performance optimization
- JIT-compiled String UDFs



Extraction is the Cornerstone

cuDF I/O 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 wherever possible

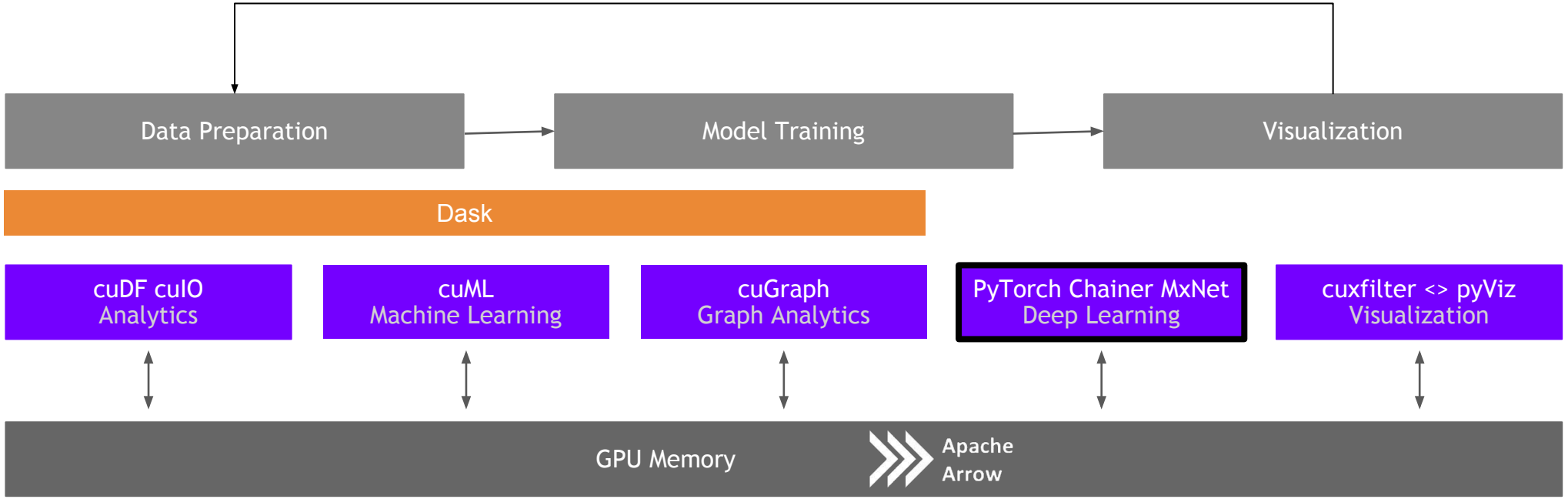
```
1]: import pandas, cudf

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!

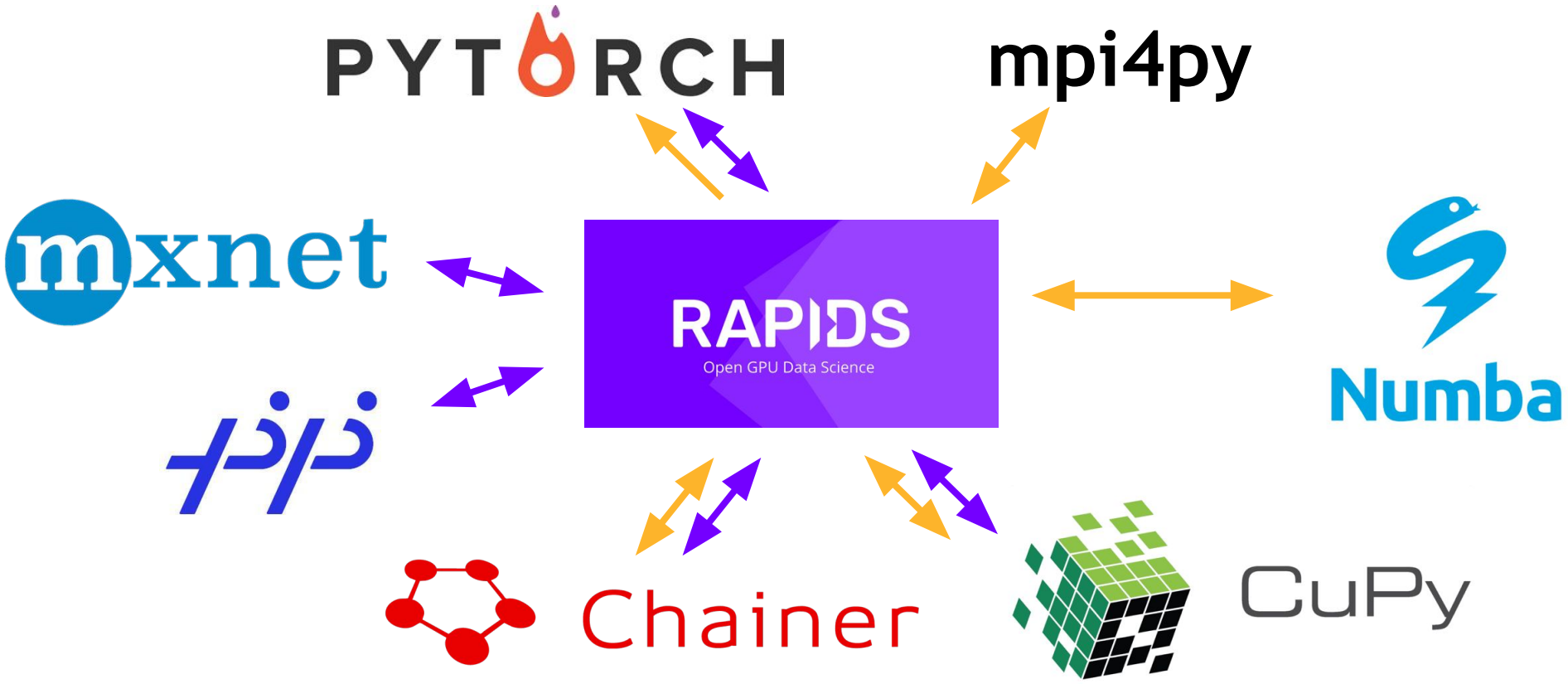
RAPIDS

Building bridges into the array ecosystem



Interoperability for the Win

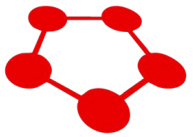
DLPack and __cuda_array_interface__



Interoperability for the Win

DLPack and __cuda_array_interface__

mpi4py



Chainer

CuPy

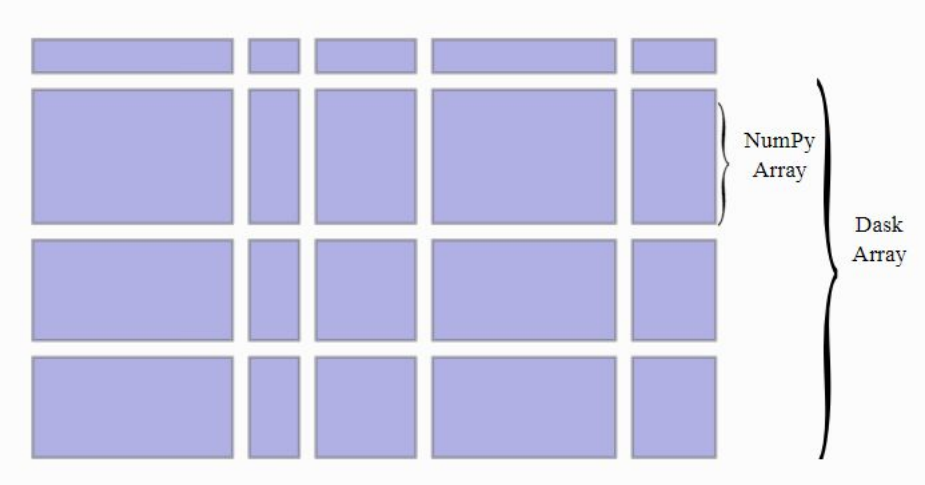
ETL: Arrays and DataFrames

Dask and CUDA Python arrays

Chainer

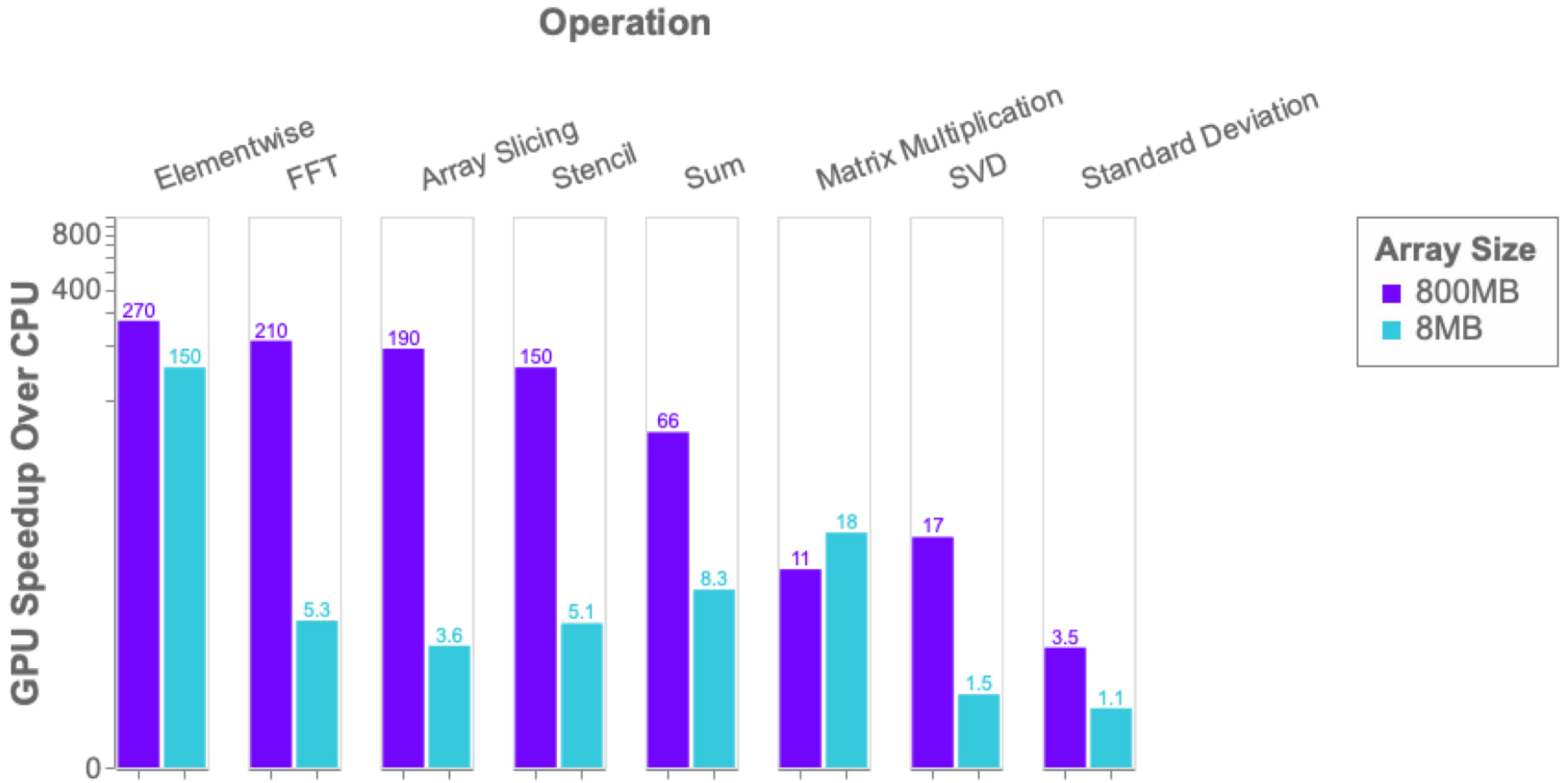
CuPy

Numba



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

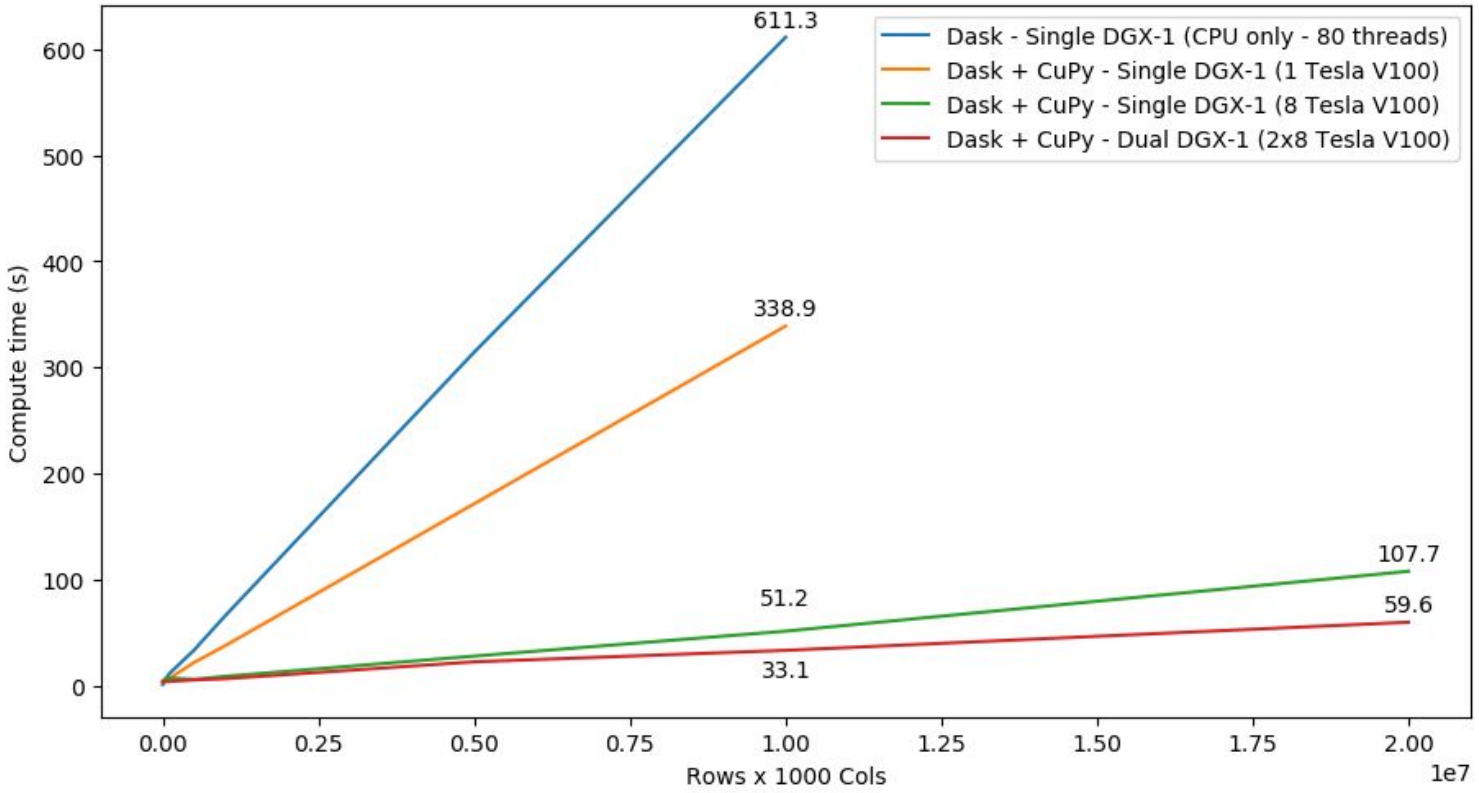
Benchmark: single-GPU CuPy vs NumPy



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

SVD Benchmark

Dask and CuPy Doing Complex Workflows



Petabyte Scale 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

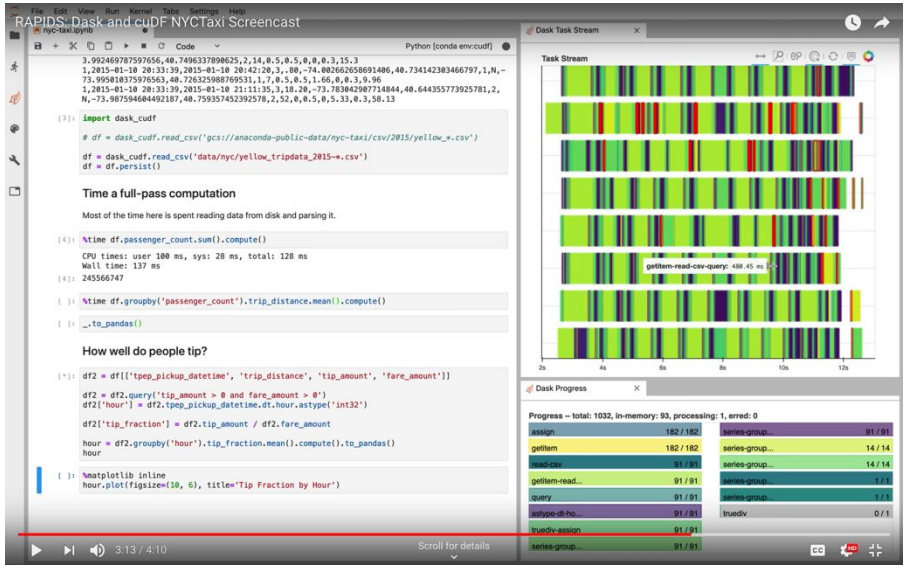
Distributed GPU array | parallel reduction | using 76x GPUs

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

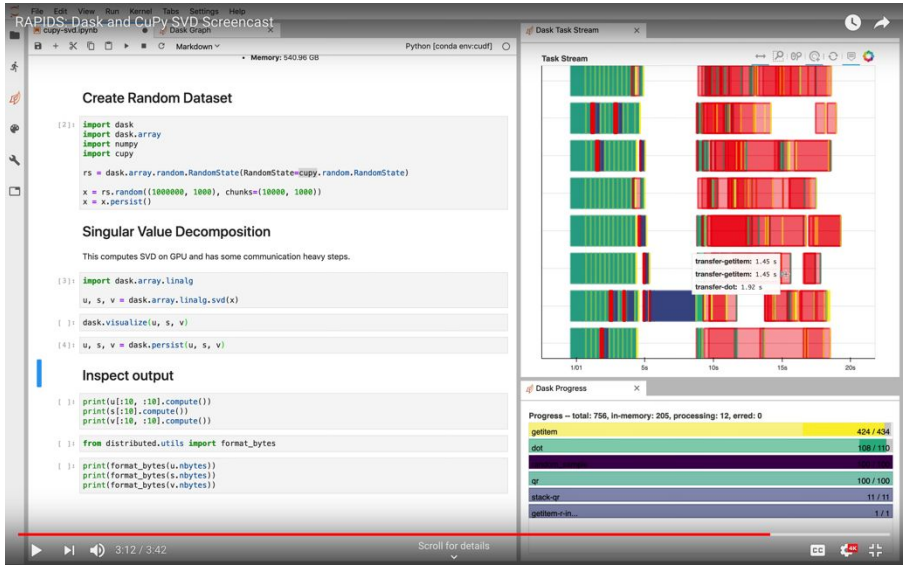
Cluster configuration: 20x GCP instances, each instance has:
CPU: 1 VM socket (Intel Xeon CPU @ 2.30GHz), 2-core, 2 threads/core, 132GB mem, GbE ethernet, 950 GB disk
GPU: 4x NVIDIA Tesla P100-16GB-PCIe (total GPU DRAM across nodes 1.22 TB)
Software: Ubuntu 18.04, RAPIDS 0.5.1, Dask=1.1.1, Dask-Distributed=1.1.1, CuPY=5.2.0, CUDA 10.0.130

ETL: Arrays and DataFrames

More Dask Awesomeness from RAPIDS



https://youtu.be/gV0cykgsTPM

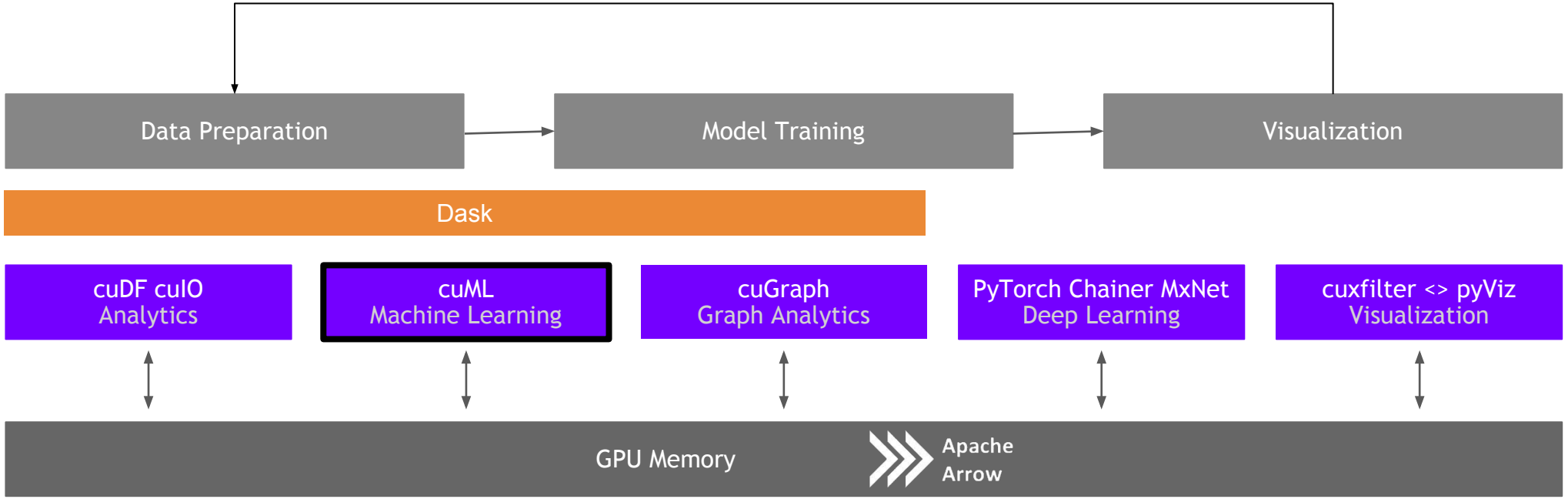


https://youtu.be/R5CiXti_MWo

cuML

Machine Learning

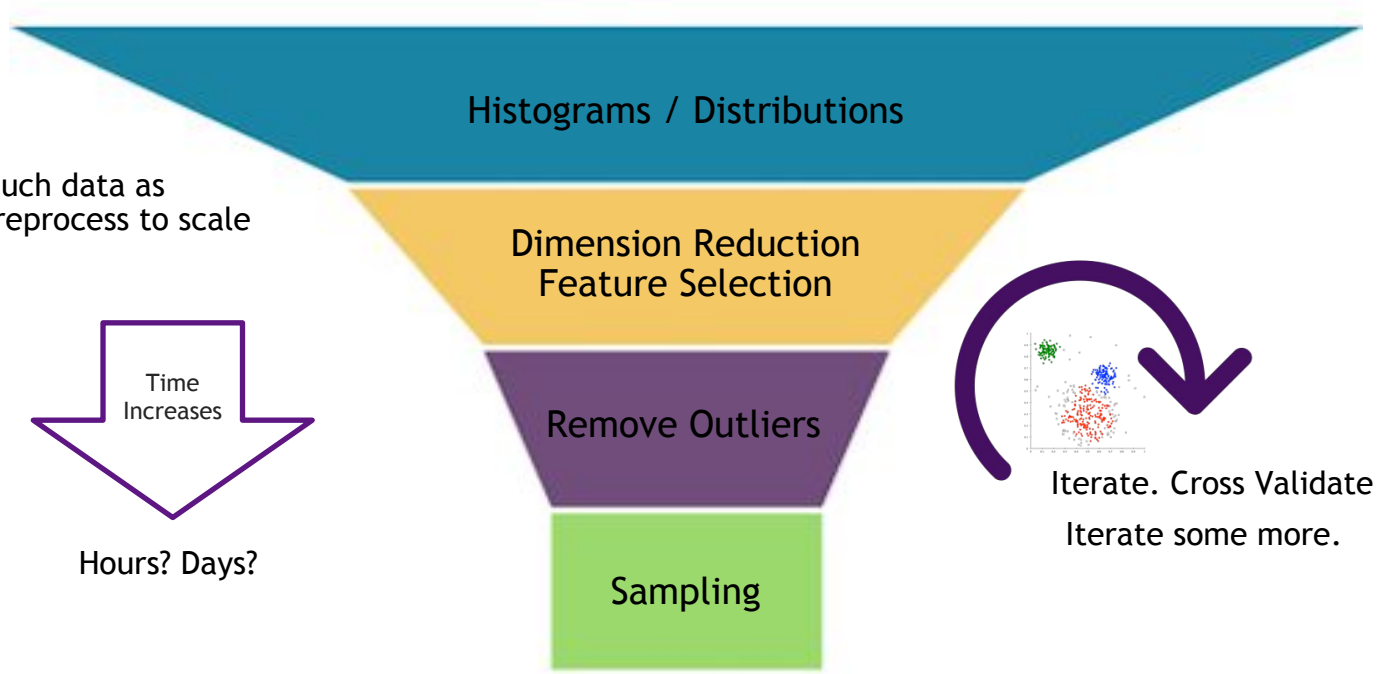
More models more problems



Problem

Data sizes continue to grow

Massive Dataset



Better to start with as much data as possible and explore / preprocess to scale to performance needs.

Time Increases

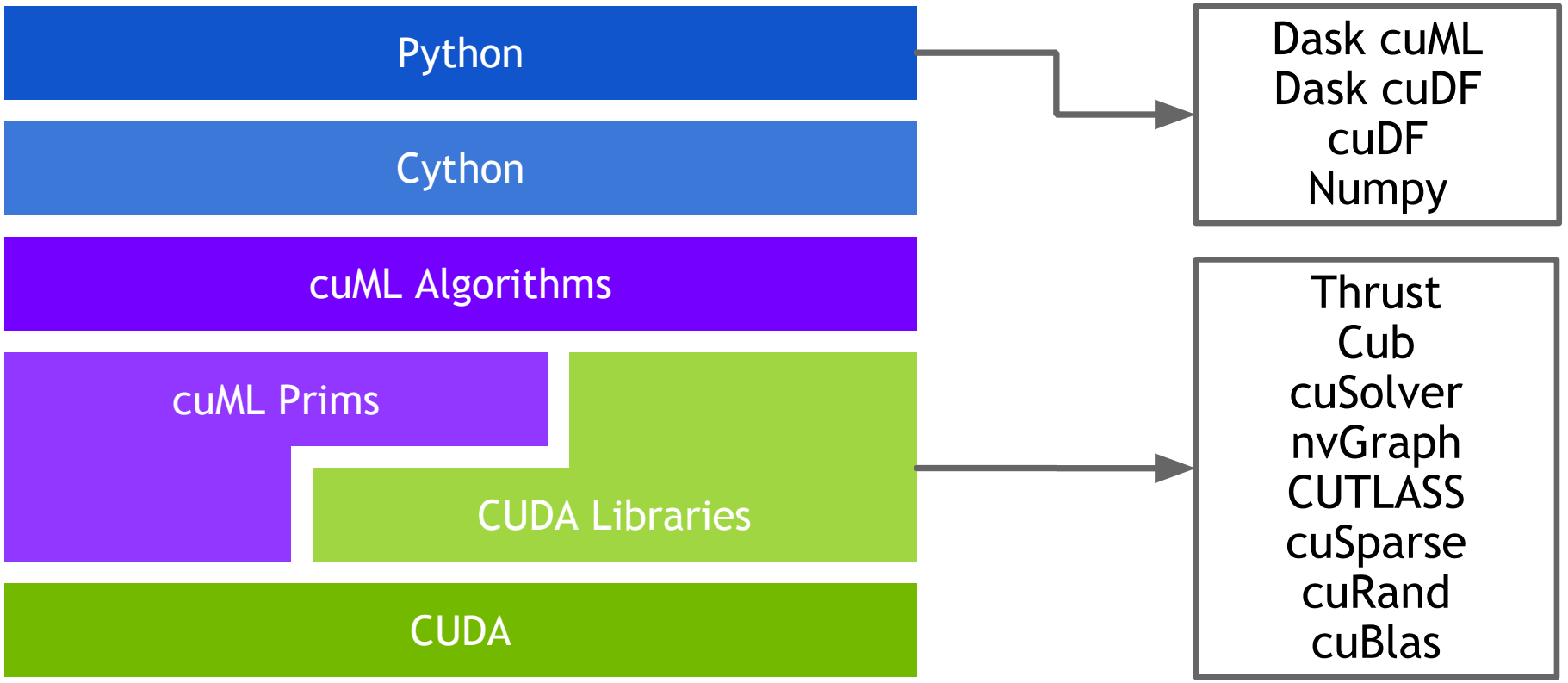
Hours? Days?

Iterate. Cross Validate & Grid Search.

Iterate some more.

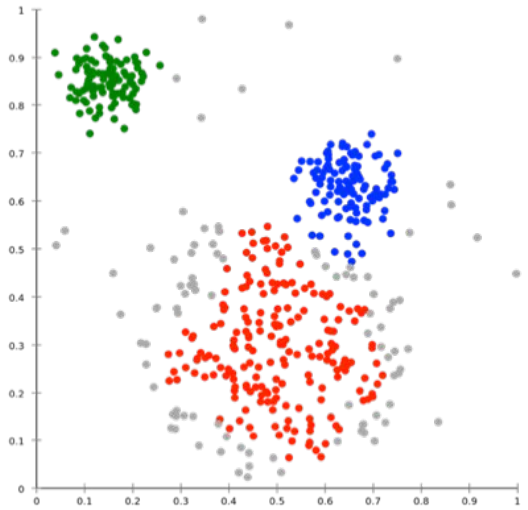
Meet reasonable speed vs accuracy tradeoff

ML Technology Stack



Algorithms

GPU-accelerated Scikit-Learn



Cross Validation

Hyper-parameter Tuning

More to come!

Classification / Regression	Decision Trees / **Random Forests** **Linear Regression** Logistic Regression K-Nearest Neighbors Support Vector Machines
Inference	Random forest / GBDT inference
Clustering	K-Means DBSCAN Spectral Clustering
Decomposition & Dimensionality Reduction	Principal Components Singular Value Decomposition UMAP Spectral Embedding T-SNE
Time Series	Holt-Winters **Seasonal ARIMA**

Key:

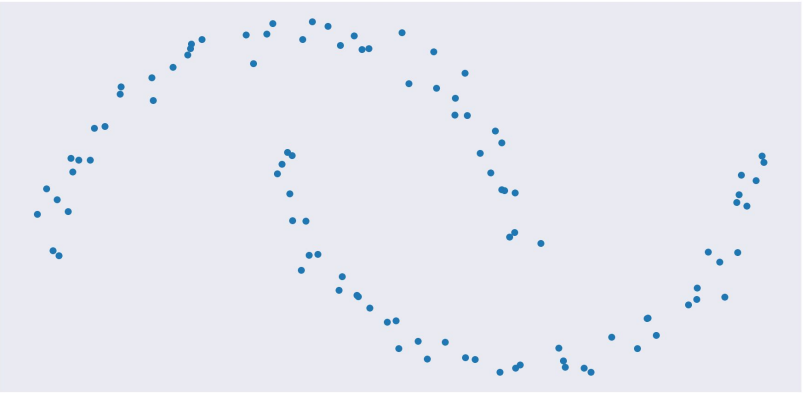
- Preexisting
- **NEW or enhanced for 0.13**

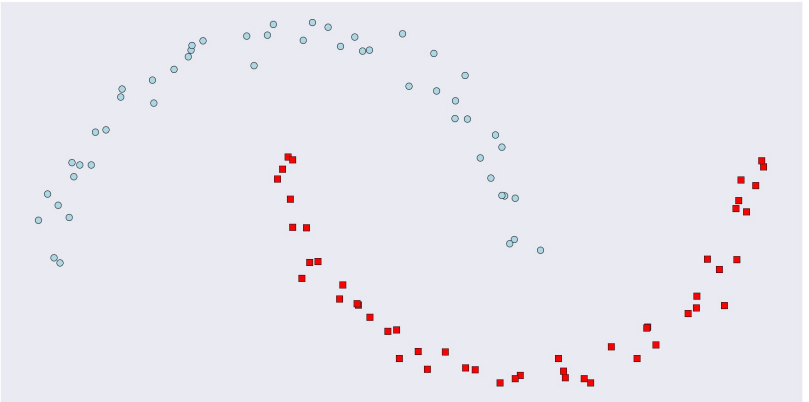
RAPIDS matches common Python APIs

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)
```



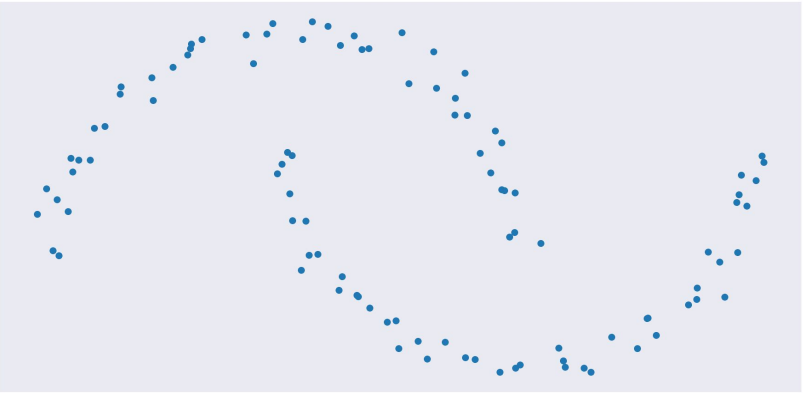


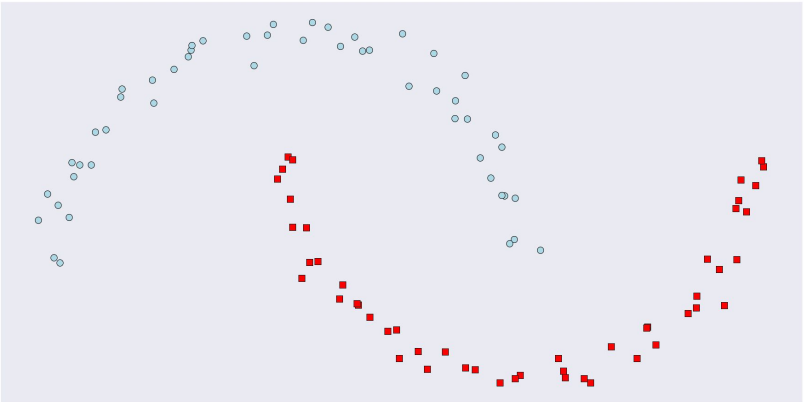
RAPIDS matches common Python APIs

GPU-Accelerated Clustering

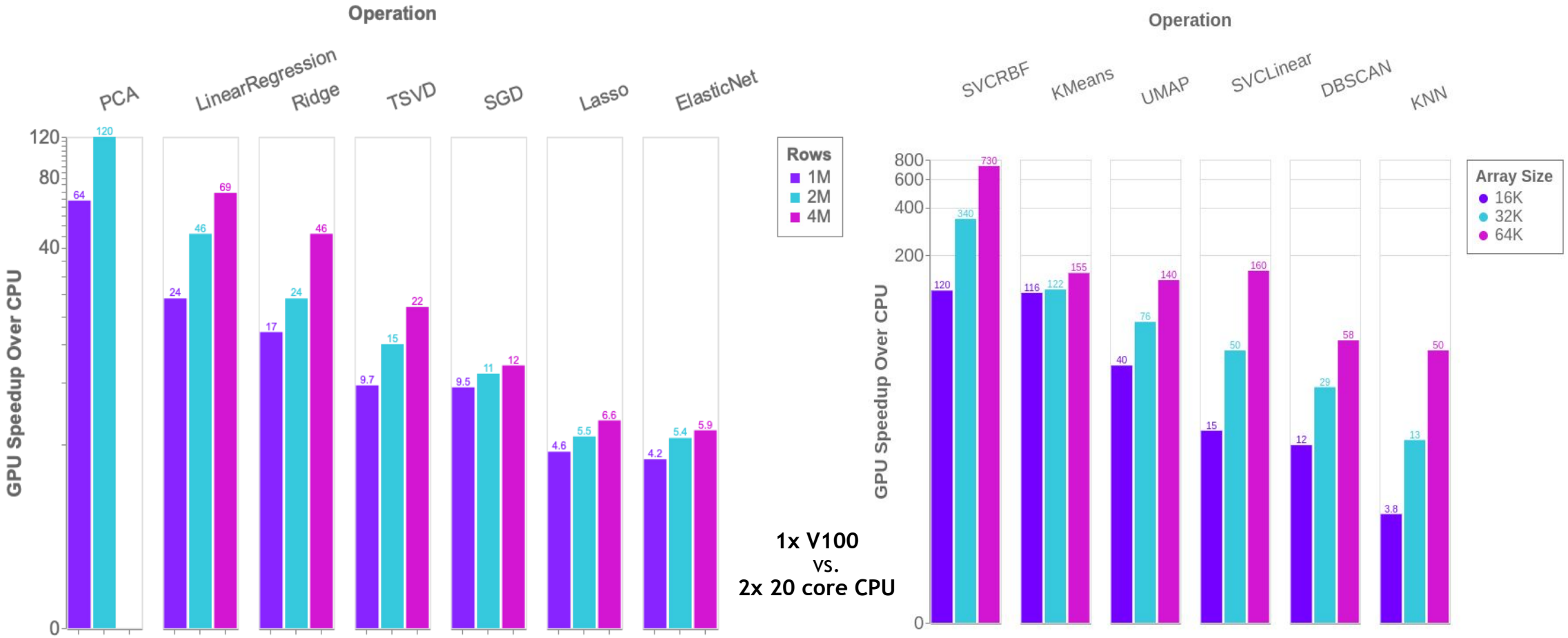
```
from sklearn.datasets import make_moons
import cudf

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)
```





Benchmarks: single-GPU cuML vs scikit-learn

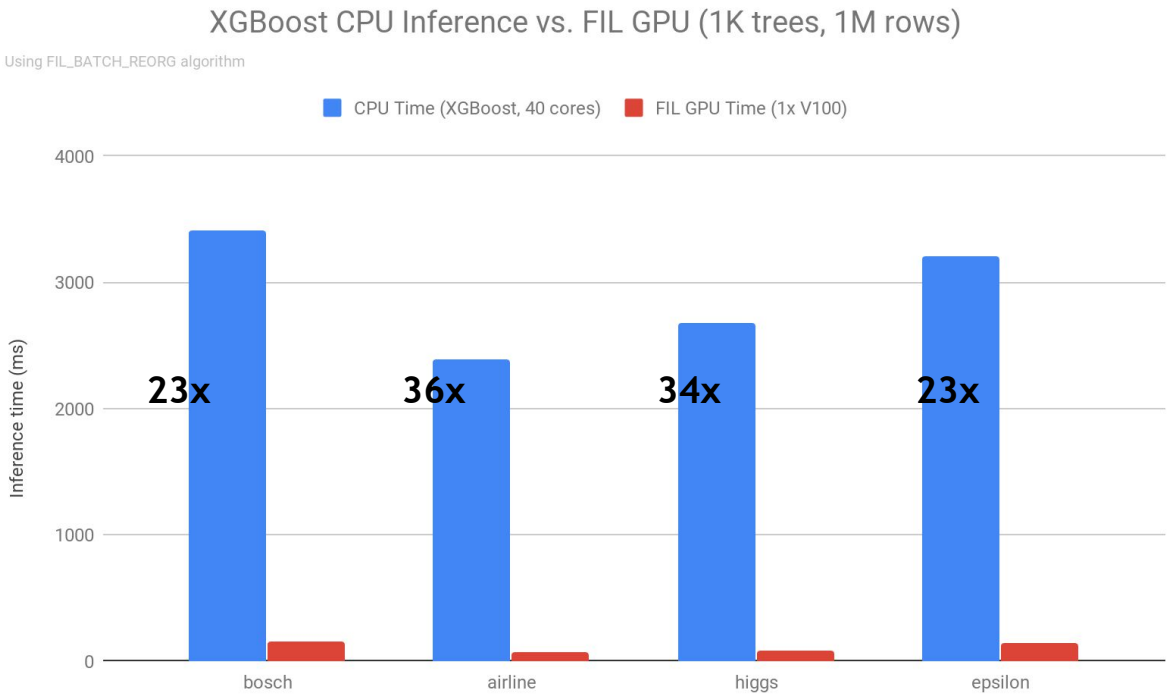


Forest Inference

Taking models from training to production

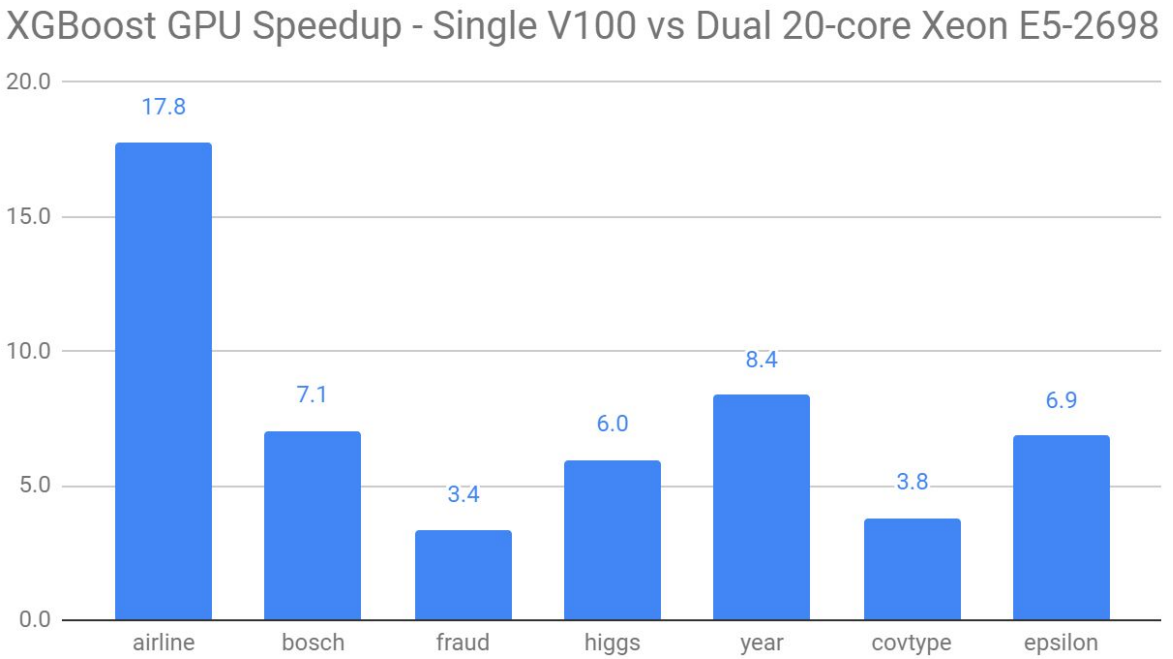
cuML's **Forest Inference Library** accelerates prediction (inference) for random forests and boosted decision trees:

- 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 per sec (with 1000 trees) on a DGX-1 for large (sparse) or dense models



RAPIDS + dmlc XGBoost

- RAPIDS works closely with the XGBoost community to accelerate GBDTs on GPU
- The default rapids conda metapackage includes XGBoost
- XGBoost can seamlessly load data from cuDF dataframes and cuPy arrays
- Dask allows XGBoost to scale to arbitrary numbers of GPUs
- With the *gpu_hist* tree method, a single GPU can outpace 10s to 100s of CPUs
- Version 1.0 of XGBoost launched with the RAPIDS 0.13 stack.



Road to 1.0

March 2020 - RAPIDS 0.13

cuML	Single-GPU	Multi-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

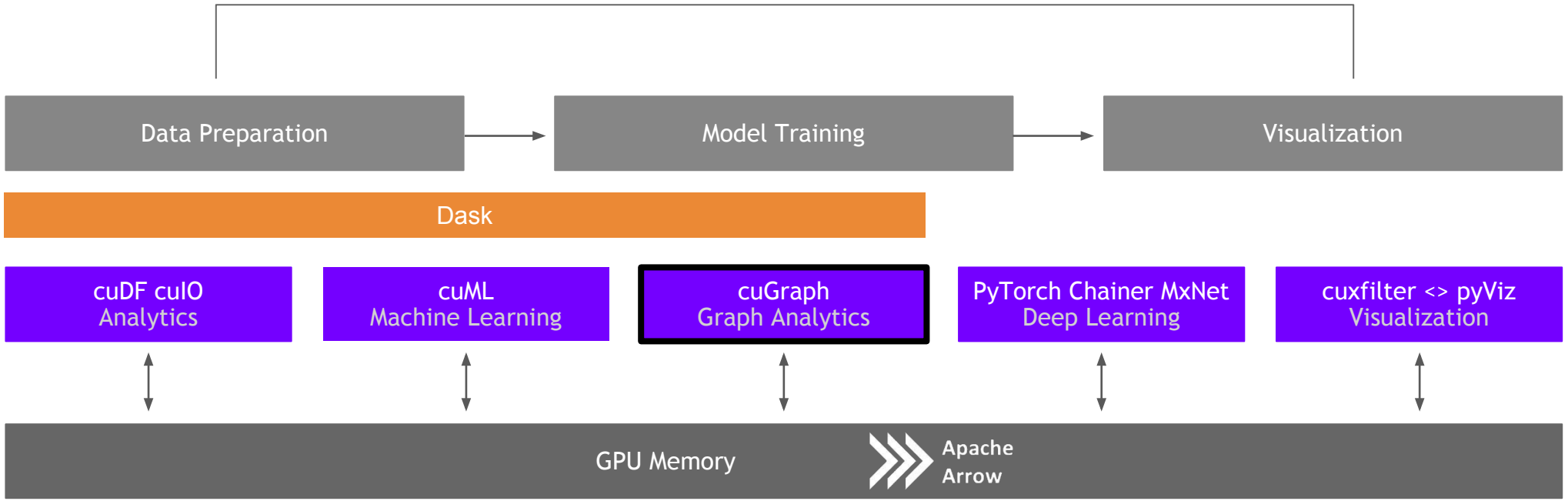
2020 - RAPIDS 1.0

cuML	Single-GPU	Multi-Node-Multi-GPU
Gradient Boosted Decision Trees (GBDT)		
Linear Regression (regularized)		
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

Multiple APIs

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

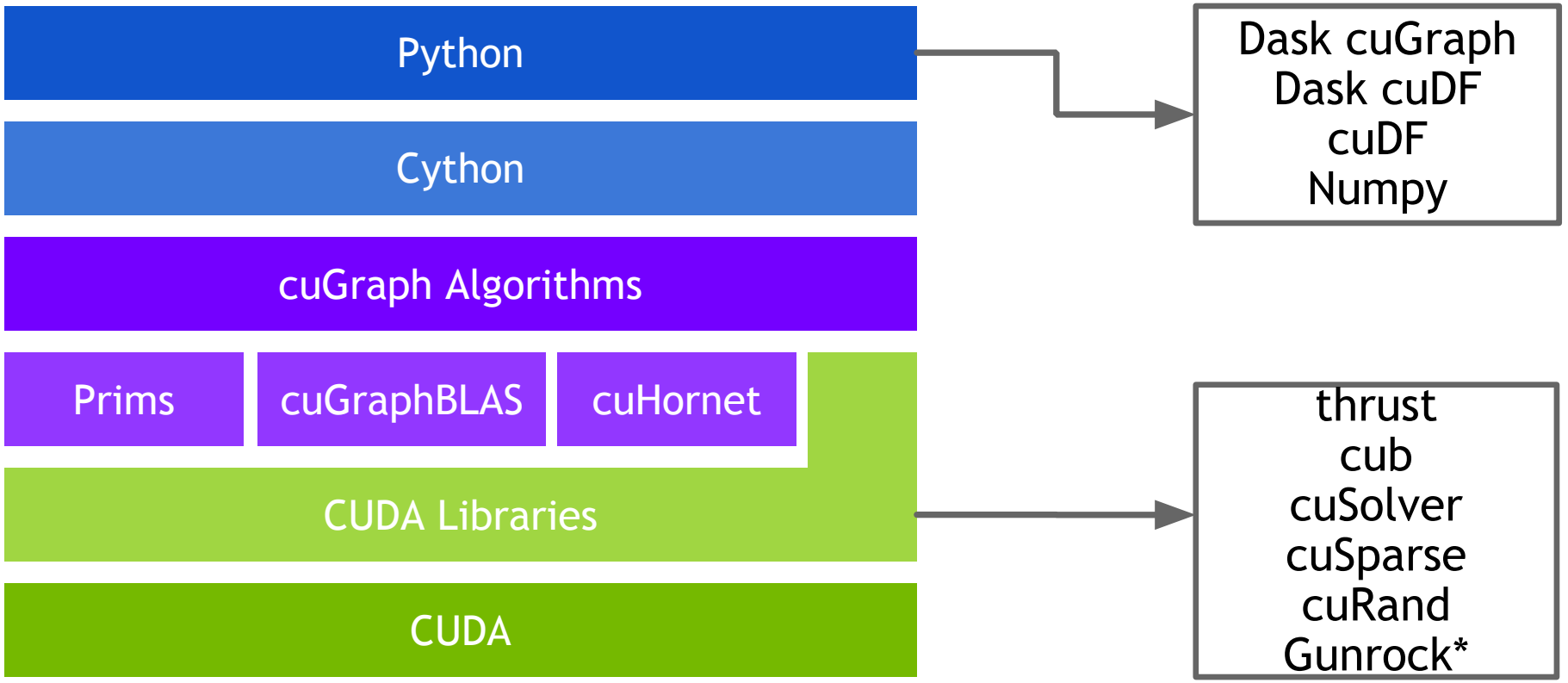
Seamless Integration with cuDF and cuML

- Property Graph support via DataFrames

Growing Functionality

- Extensive collection of algorithm, primitive, and utility functions

Graph Technology Stack

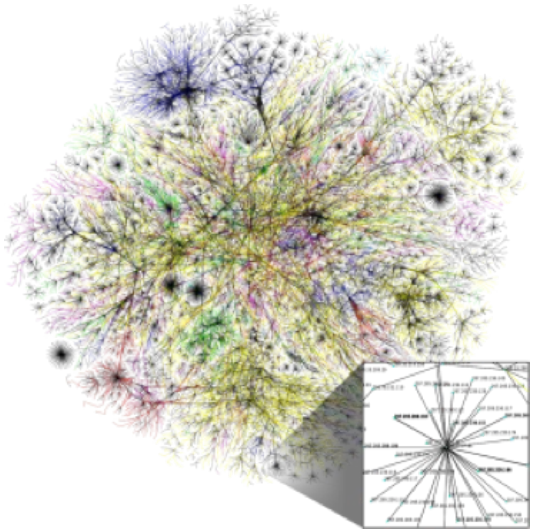


nvGRAPH has been Opened Sourced and integrated into cuGraph. A legacy version is available in a RAPIDS GitHub repo

* Gunrock is from UC Davis

Algorithms

GPU-accelerated NetworkX



Graph Classes

Structure

Multi-GPU

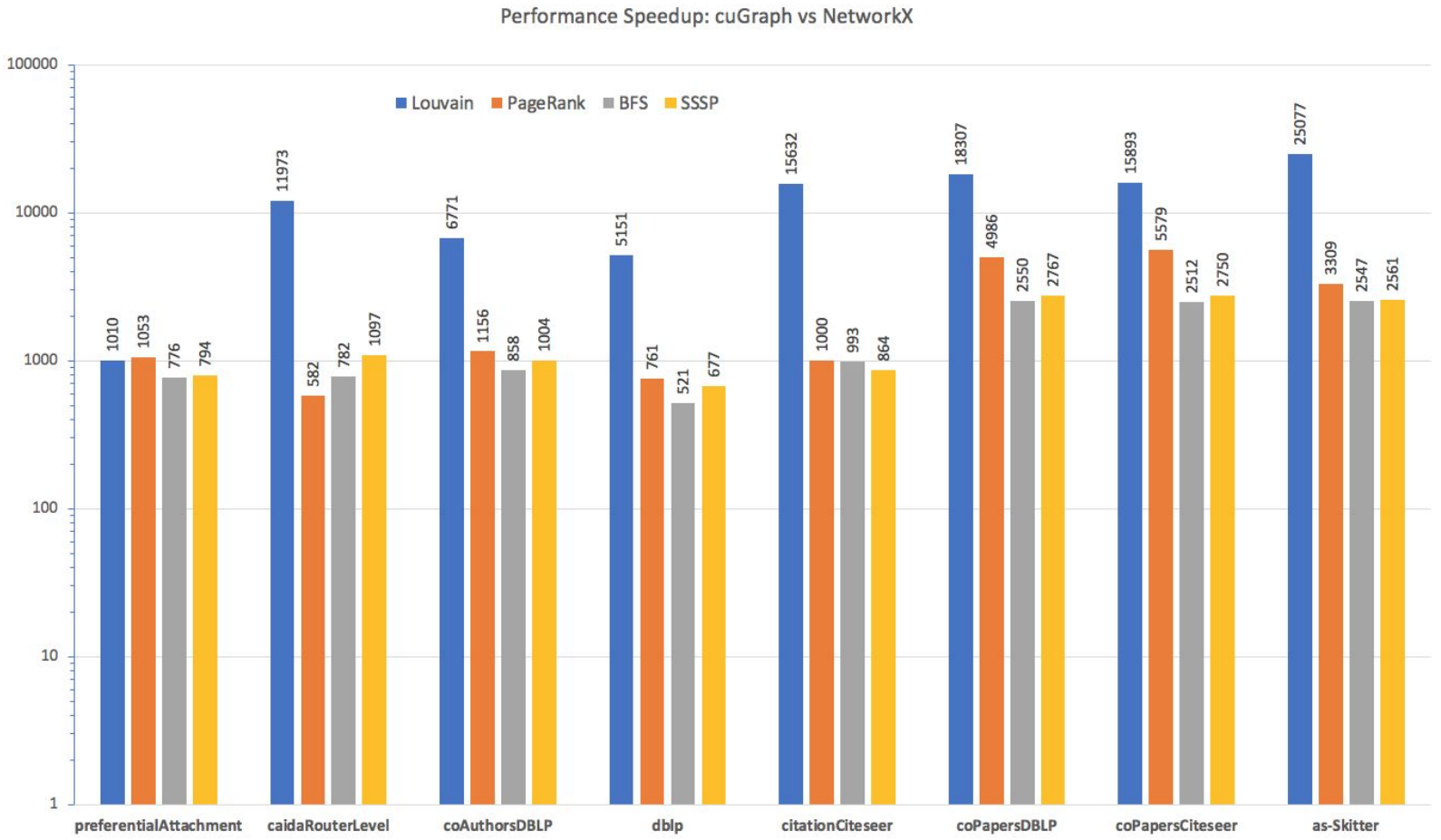
Renumbering
Auto-renumbering

Utilities

More to come!

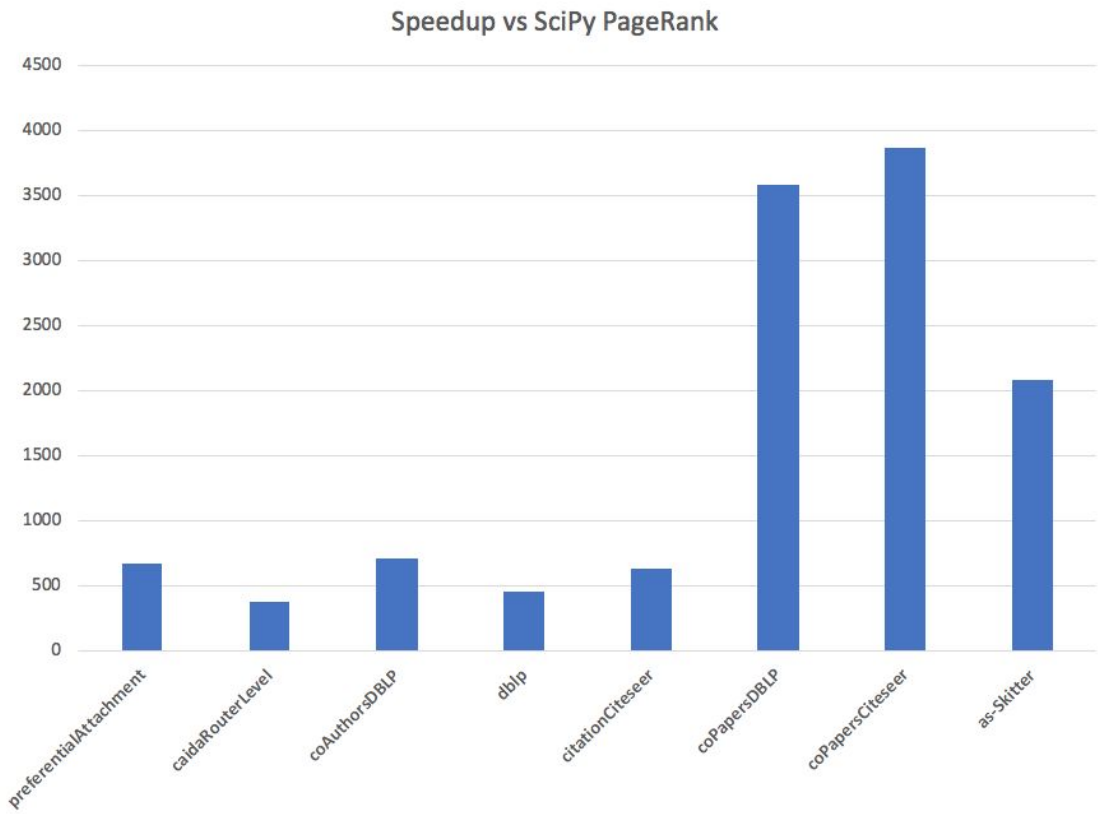
Community	Spectral Clustering Balanced-Cut Modularity Maximization Louvain Ensemble Clustering for Graphs Subgraph Extraction KCore and KCore Number Triangle Counting ***K-Truss***
Components	Weakly Connected Components Strongly Connected Components
Link Analysis	Page Rank (Multi-GPU) Personal Page Rank
Link Prediction	Jaccard Weighted Jaccard Overlap Coefficient
Traversal	Single Source Shortest Path (SSSP) Breadth First Search (BFS)
Centrality	Katz ***Betweenness Centrality***

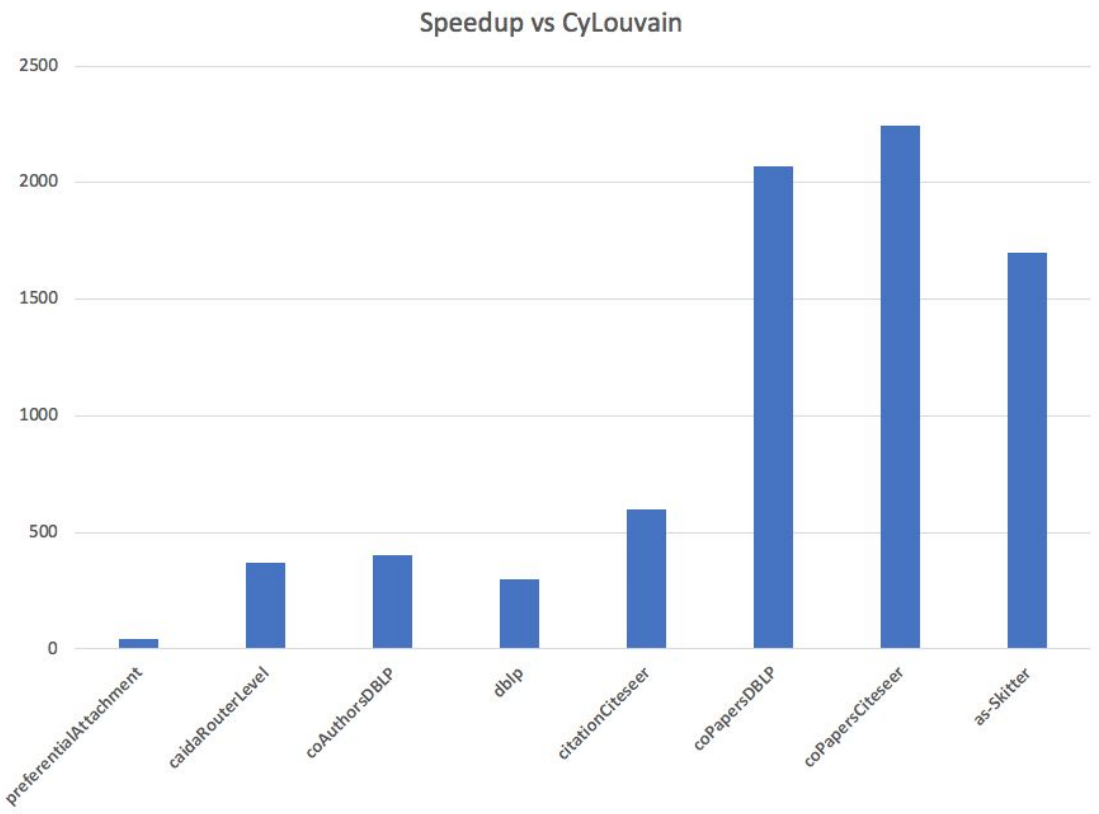
Benchmarks: single-GPU cuGraph vs NetworkX



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	30,491,458
coPapersCiteseer	434,102	32,073,440
as-Skitter	1,696,415	22,190,596

Benchmarks: cyLouvain and SciPy PageRank





Multi-GPU PageRank Performance

PageRank portion of the HiBench benchmark suite

HiBench Scale	Vertices	Edges	CSV File (GB)	# of V100 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

March 2020 - RAPIDS 0.13

cuGraph	Single-GPU	Multi-GPU	Multi-Node-Multi-GPU
PageRank			
Personal Page Rank			
Katz			
Betweenness Centrality			
Spectral Clustering			
Louvain			
Ensemble Clustering for Graphs			
K-Core			
K-Truss			
Triangle Counting			
Connected Components (Weak and Strong)			
Jaccard			
Overlap Coefficent			
Single Source Shortest Path (SSSP)			
Breadth First Search (BFS)			

Road to 1.0

March 2020 - RAPIDS 1.0

cuGraph	Single-GPU	Multi-Node-Multi-GPU
PageRank		
Personal Page Rank		
Katz		
Betweenness Centrality		
Spectral Clustering		
Louvain		
Ensemble Clustering for Graphs		
K-Core		
K-Truss		
Triangle Counting		
Connected Components (Weak and Strong)		
Jaccard		
Overlap Coefficent		
Single Source Shortest Path (SSSP)		
Breadth First Search (BFS)		

cuSpatial

cuSpatial

Breakthrough Performance & Ease of Use

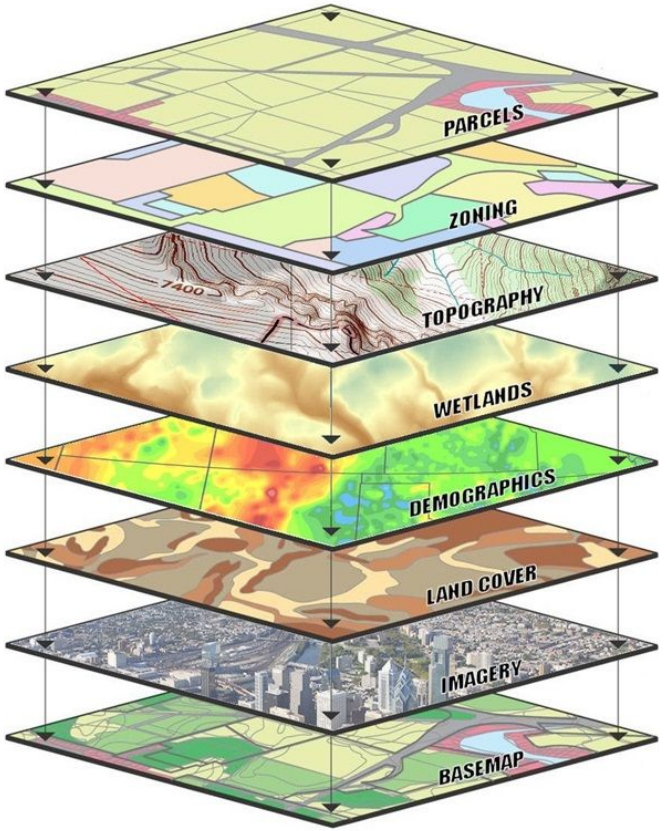
- Up to 1000x faster than CPU spatial libraries
- Python and C++ APIs for maximum usability and integration

Growing Functionality

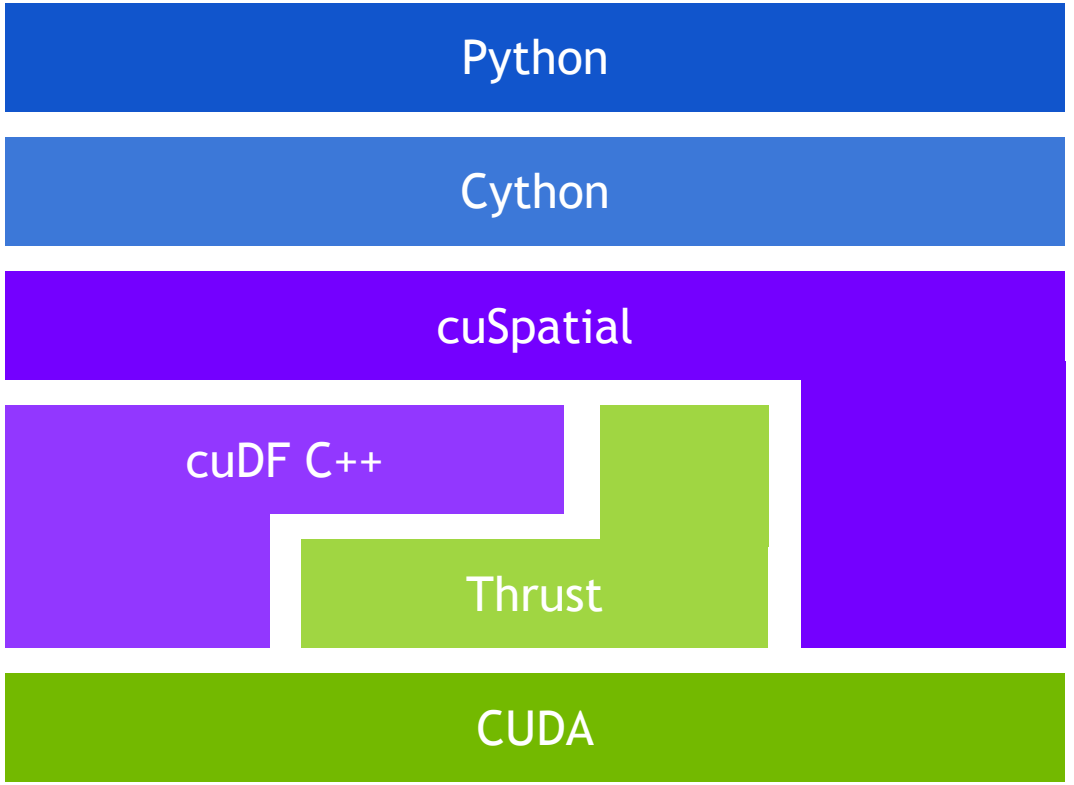
- Extensive collection of algorithm, primitive, and utility functions for spatial analytics

Seamless Integration into RAPIDS

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



cuSpatial Technology Stack



cuSpatial

0.13 and Beyond

Layer	0.13 Functionality	Functionality Roadmap (2020)
High-level Analytics	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.
Graph layer	cuGraph	Map matching, Djikstra algorithm, Routing
Query layer	Spatial Window	Nearest Neighbor,KNN, Spatiotemporal range search and joins
Index layer		Grid, Quad Tree, R-Tree, Geohash, Voronoi Tessellation
Geo-operations	Point in polygon (PIP), Haversine distance, Hausdorff distance, lat-lon to xy transformation, spline interpolation	Line intersecting polygon, Other distance functions, Polygon intersection, union
Geo-representation	Shape primitives, points, polylines, polygons	Additional shape primitives

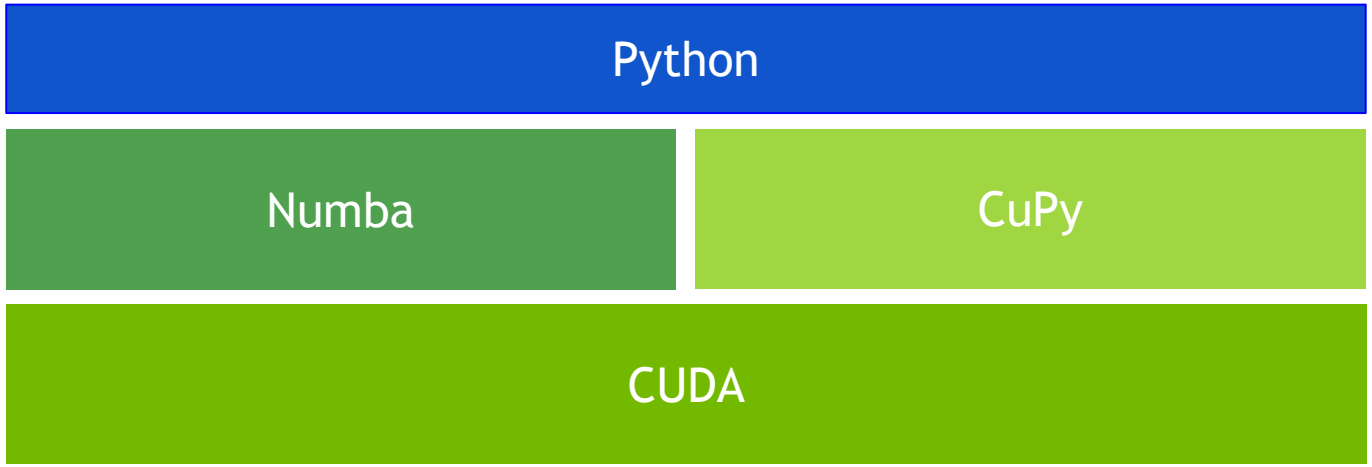
cuSpatial

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
Window Functions	Kaiser Blackman Hamming Hanning
Spectral Analysis	Periodogram Welch Spectrogram

Wavelets

Peak Finding

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	28400	92.2	308.0
correlate	16800	48.4	347.1
resample	14700	51.1	287.7
resample_poly	3110	13.7	227.0
welch	4620	53.7	86.0
spectrogram	2520	28	90.0
cwt	46700	277	168.6

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()
```

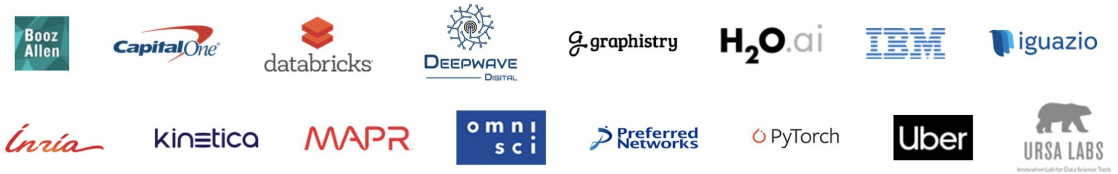
```
866 µs ± 38 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)
```

Community

Ecosystem Partners

CONTRIBUTORS

ADOPTERS



OPEN SOURCE

Building on top of RAPIDS

A bigger, better, stronger ecosystem for all

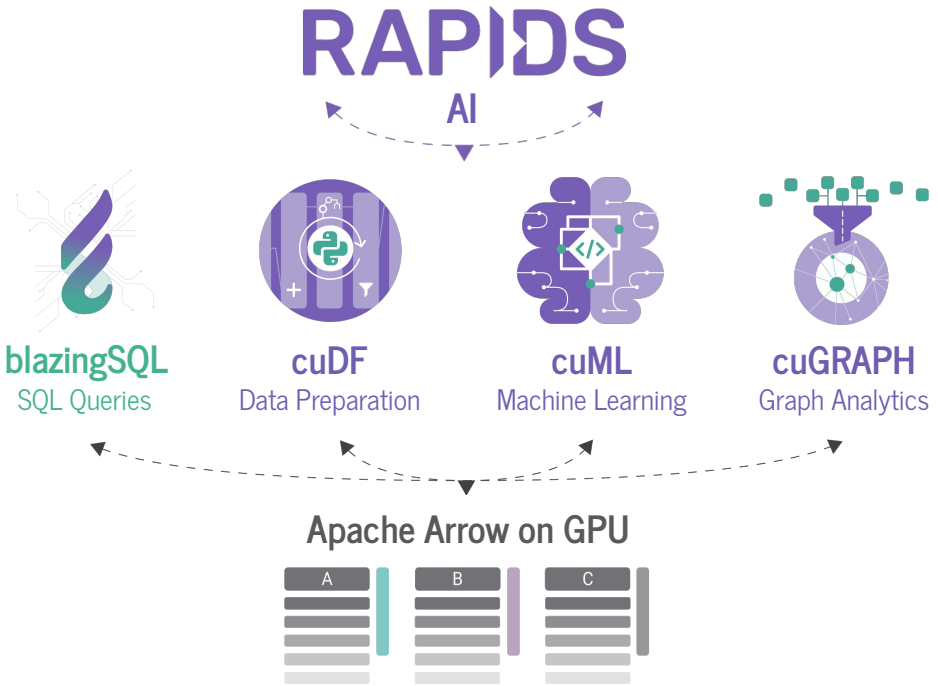
Streamz

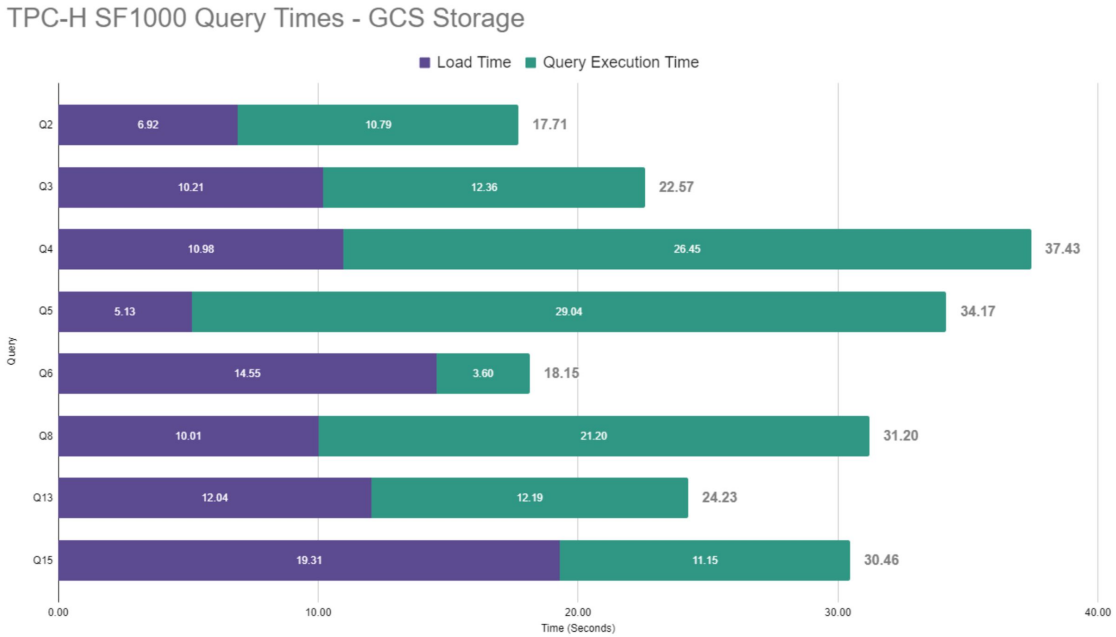
High-Performance Serverless event and data processing that utilizes RAPIDS for GPU Acceleration

GPU accelerated SQL engine built on top of RAPIDS

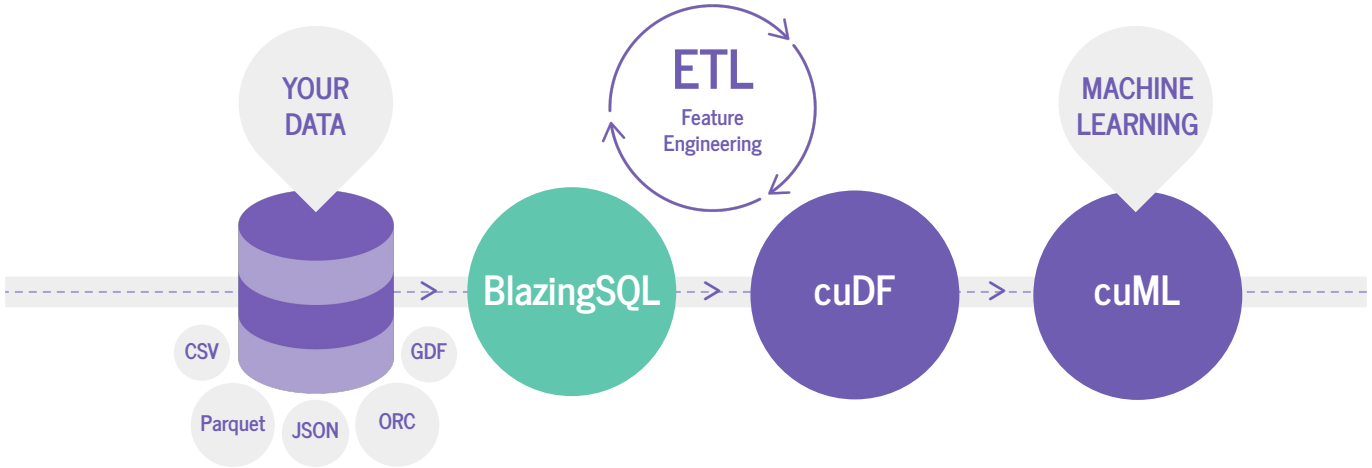
Distributed stream processing using RAPIDS and Dask

BlazingSQL


RAPIDS
AI
blazingSQL
SQL Queries
cuDF
Data Preparation
cuML
Machine Learning
cuGRAPH
Graph Analytics
Apache Arrow on GPU
A
B
C


TPC-H SF1000 Query Times - GCS Storage
Load Time
Query Execution Time
Query
Q2
6.92
10.79
17.71
Q3
10.21
12.36
22.57
Q4
10.98
26.45
37.43
Q5
5.13
29.04
34.17
Q6
14.55
3.60
18.15
Q8
10.01
21.20
31.20
Q13
12.04
12.19
24.23
Q15
19.31
11.15
30.46
0.00
10.00
20.00
30.00
40.00
Time (Seconds)

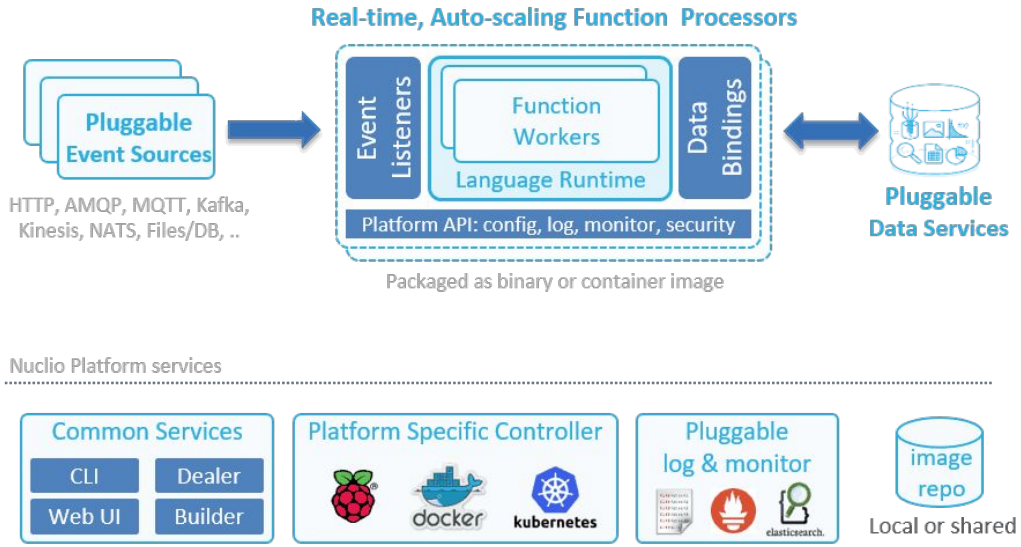
BlazingSQL




```
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

Deploy RAPIDS Everywhere

Focused on robust functionality, deployment, and user experience

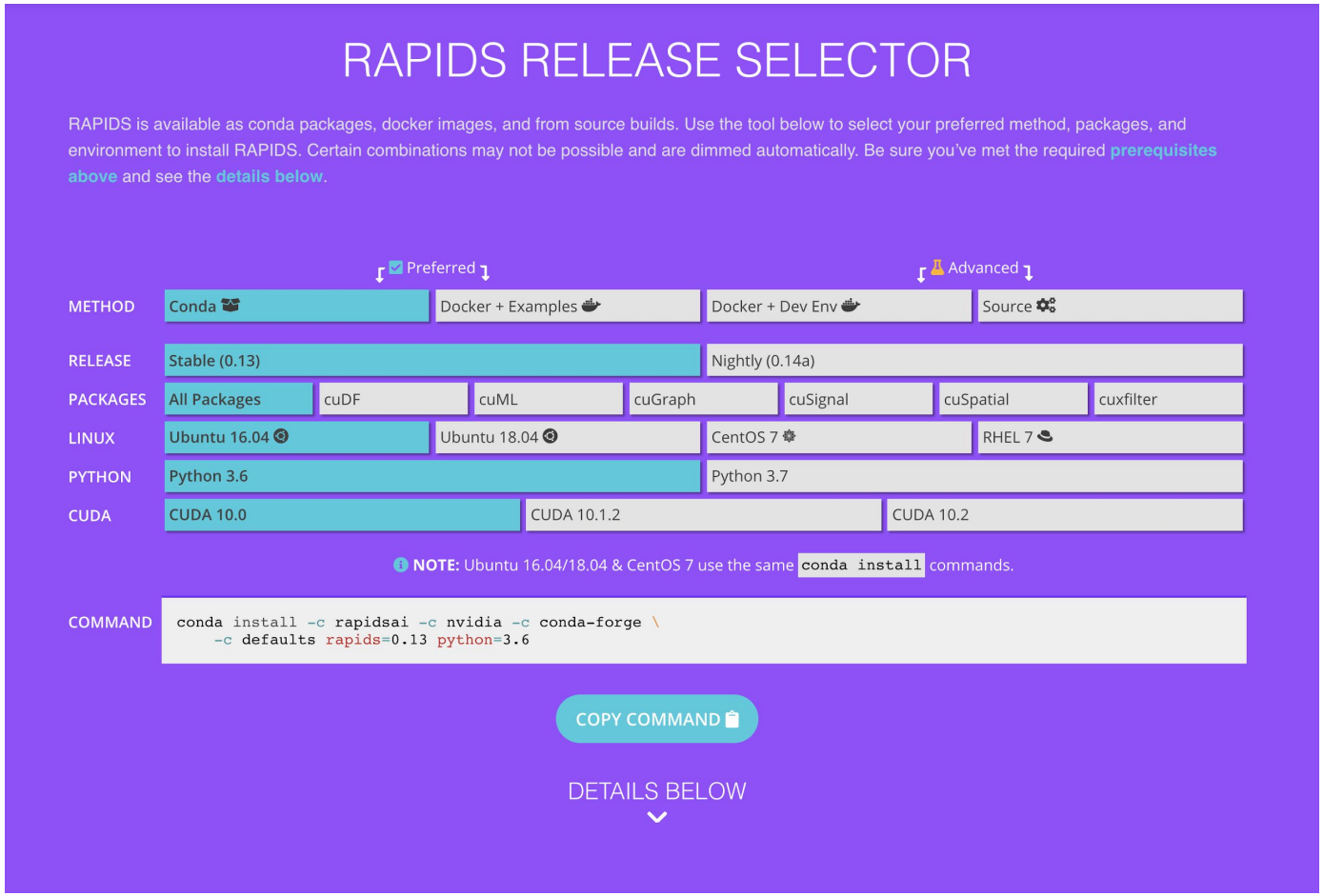
Integration with major cloud providers
Both containers and cloud specific machine instances
Support for Enterprise and HPC Orchestration Layers

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 examples 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

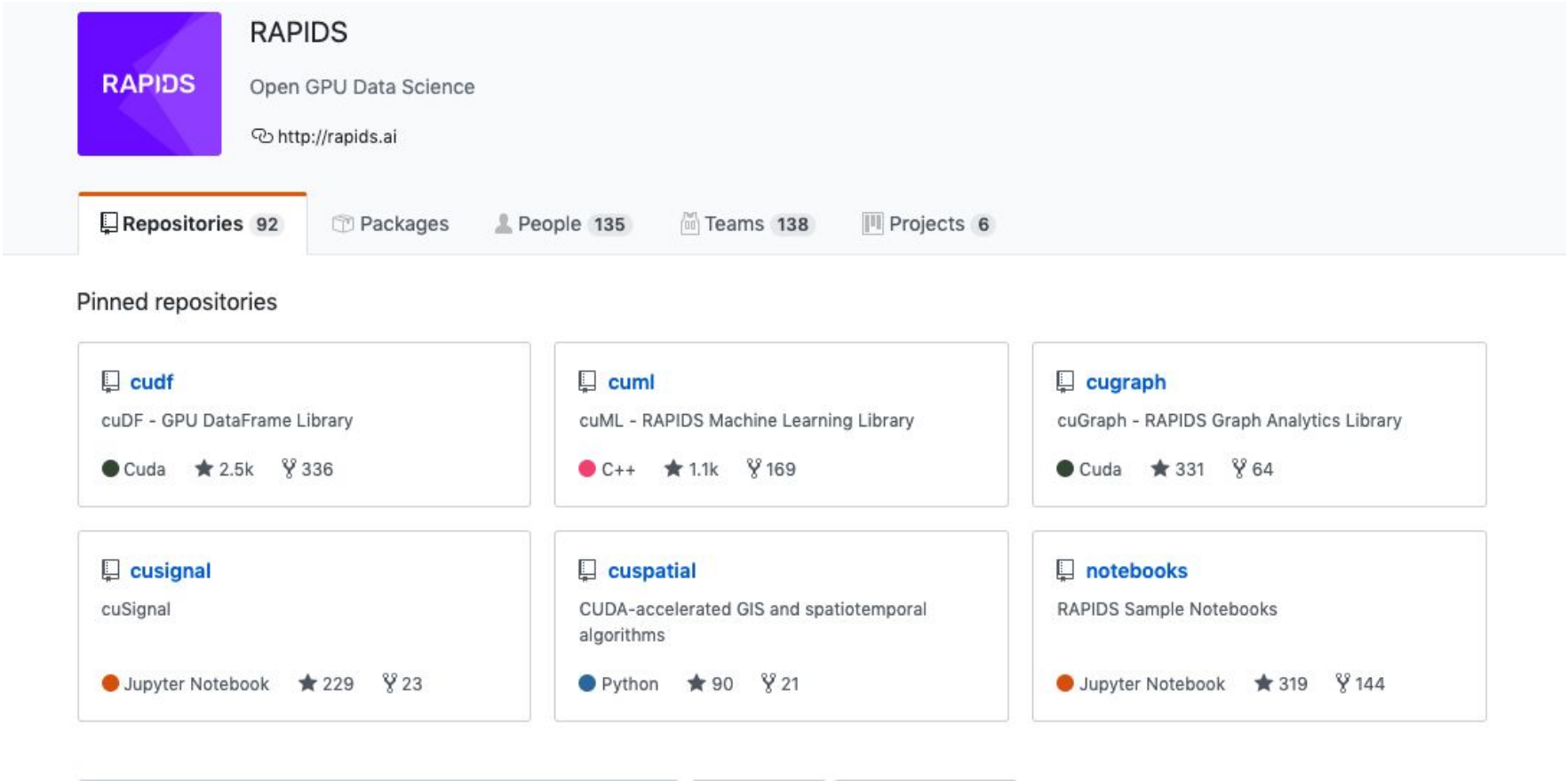
Easy Installation

Interactive Installation Guide



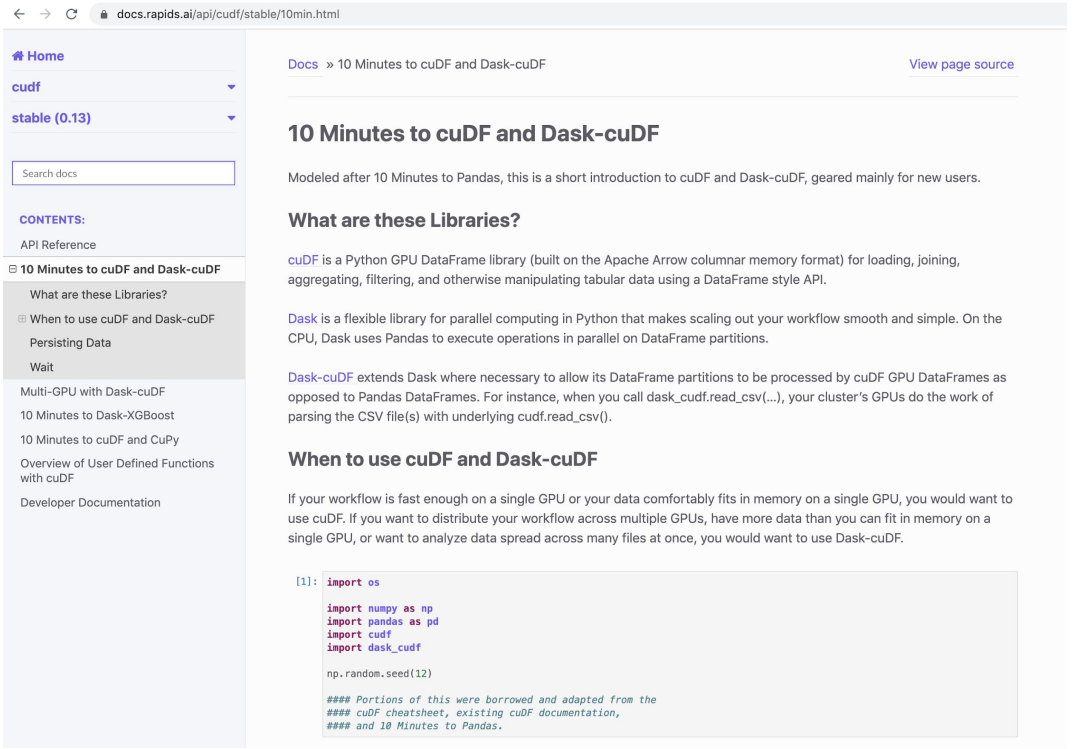
Explore: RAPIDS Github

https://github.com/rapidsai



Explore: RAPIDS Docs

Improved and easier to use!



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:

```
size
```

https://github.com/rapidsai

RAPIDS Release 0.8: Same Community New Freedoms

Making more friends and building more bridges to more ecosystems. It's now easier than ever to get started with RAPIDS.

Josh Patterson
Jul 19 · 7 min read

gQuant—GPU Accelerated examples for Quantitative Analyst Tasks

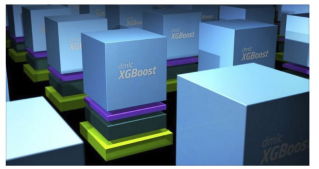
A simple trading strategy backtest for 5000 stocks using GPUs and getting 20X speedup

Yi Dong
Jul 16 · 6 min read ★

Financial data modeling with RAPIDS.

See how RAPIDS was used to place 17th in the Banco Santander Kaggle Competition

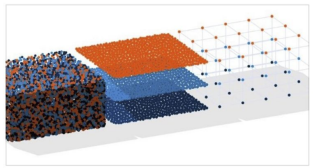
Jiwei Liu
Jul 3 · 5 min read



NVIDIA GPUs and Apache Spark, One Step Closer

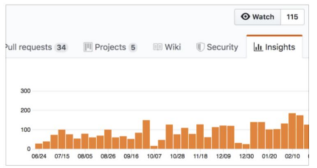
RAPIDS XGBoost4J-Spark Package Now Available

Karthikeyan Rajendran



When Less is More: A brief story about XGBoost feature engineering

A glimpse into how a Data Scientist makes decisions about featuring engineering an XGBoost machine



Nightly News: CI produces latest packages

Release code early and often. Stay current on latest features with our nightly conda and container releases.

https://medium.com/rapids-ai

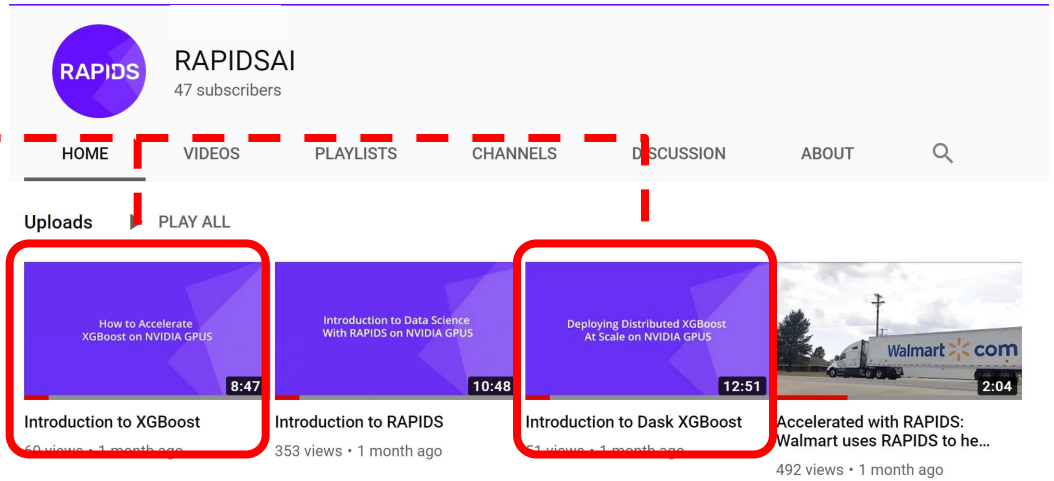
Explore: Notebooks Contrib

Tutorials, examples, and various E2E demos available, with Youtube explanations, code walkthroughs and use cases

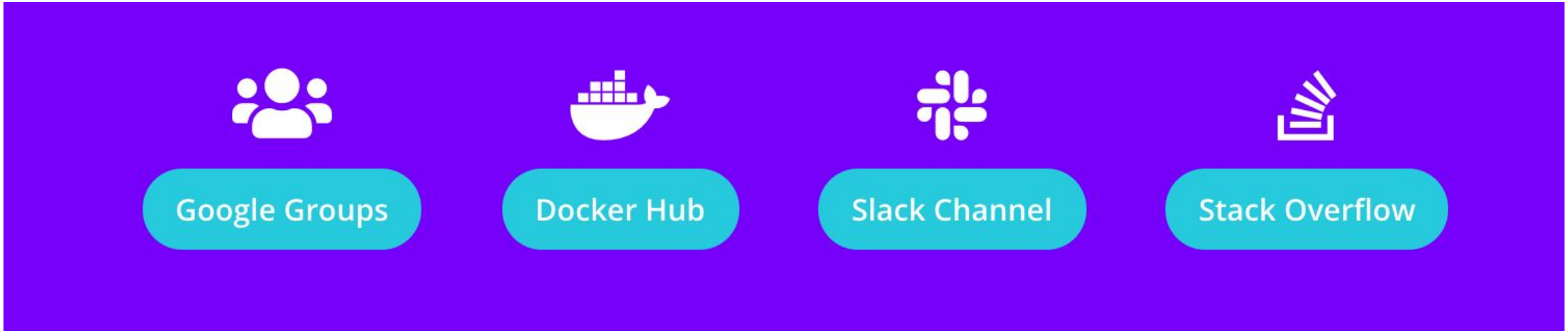
intro_tutorials	05_Introduction_to_Dask_cuDF	This notebook shows how to work with cuDF DataFrames distributed across multiple GPUs using Dask.
intro_tutorials	06_Introduction_to_Supervised_Learning	This notebook shows how to do GPU accelerated Supervised Learning in RAPIDS.
intro_tutorials	07_Introduction_to_XGBoost	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.

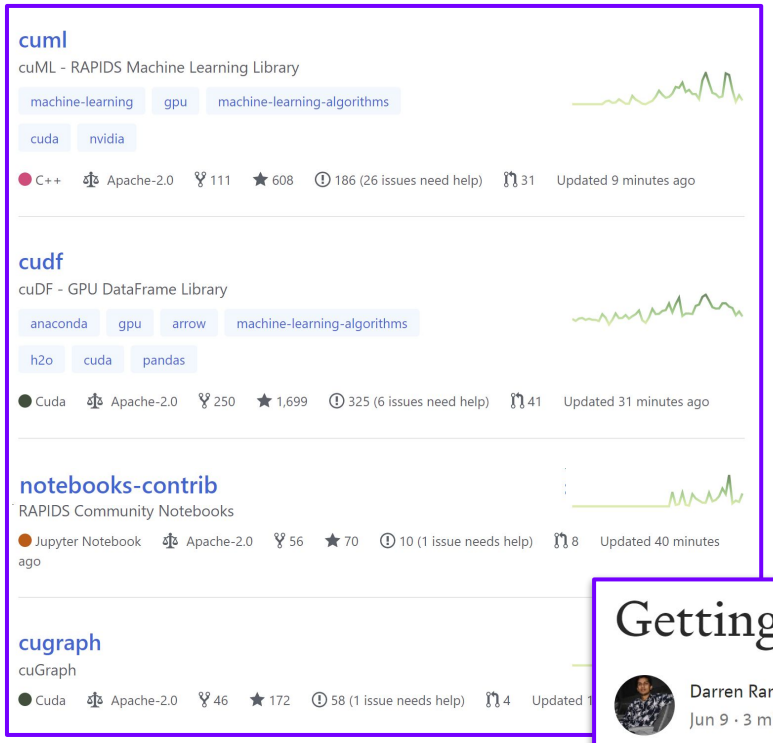


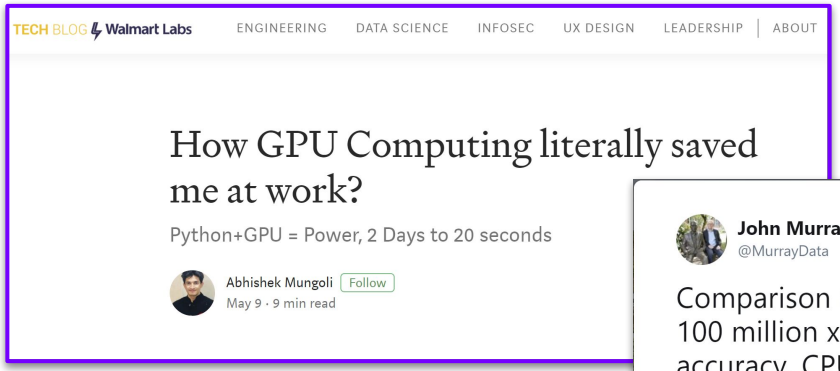
Join the Conversation

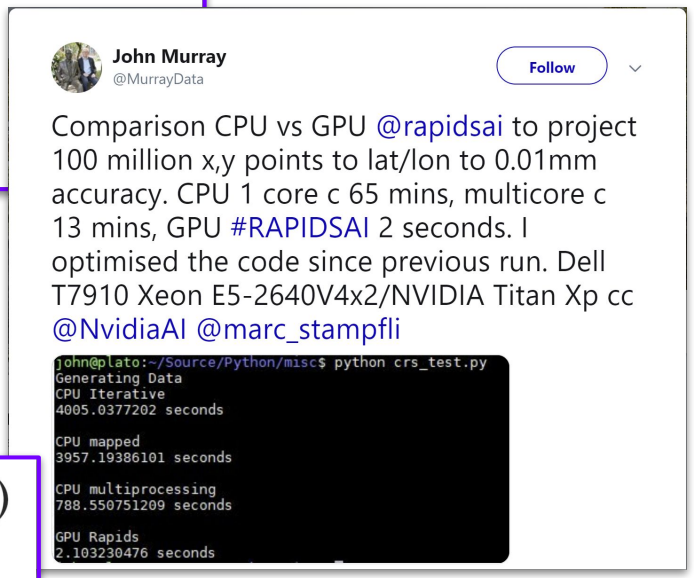


Contribute Back

Issues, feature requests, PRs, Blogs, Tutorials, Videos, QA…bring your best!



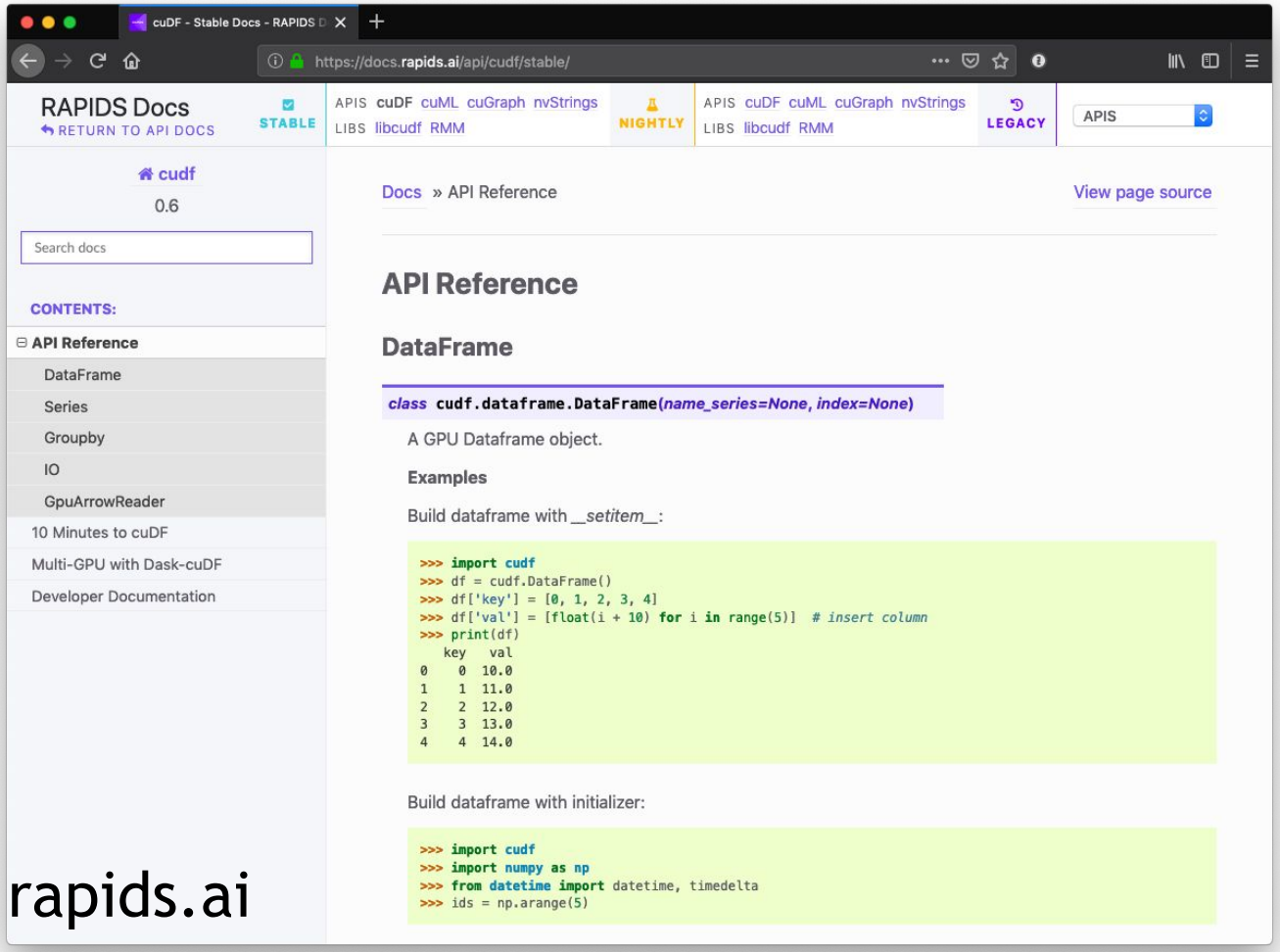




Getting Started

RAPIDS Docs

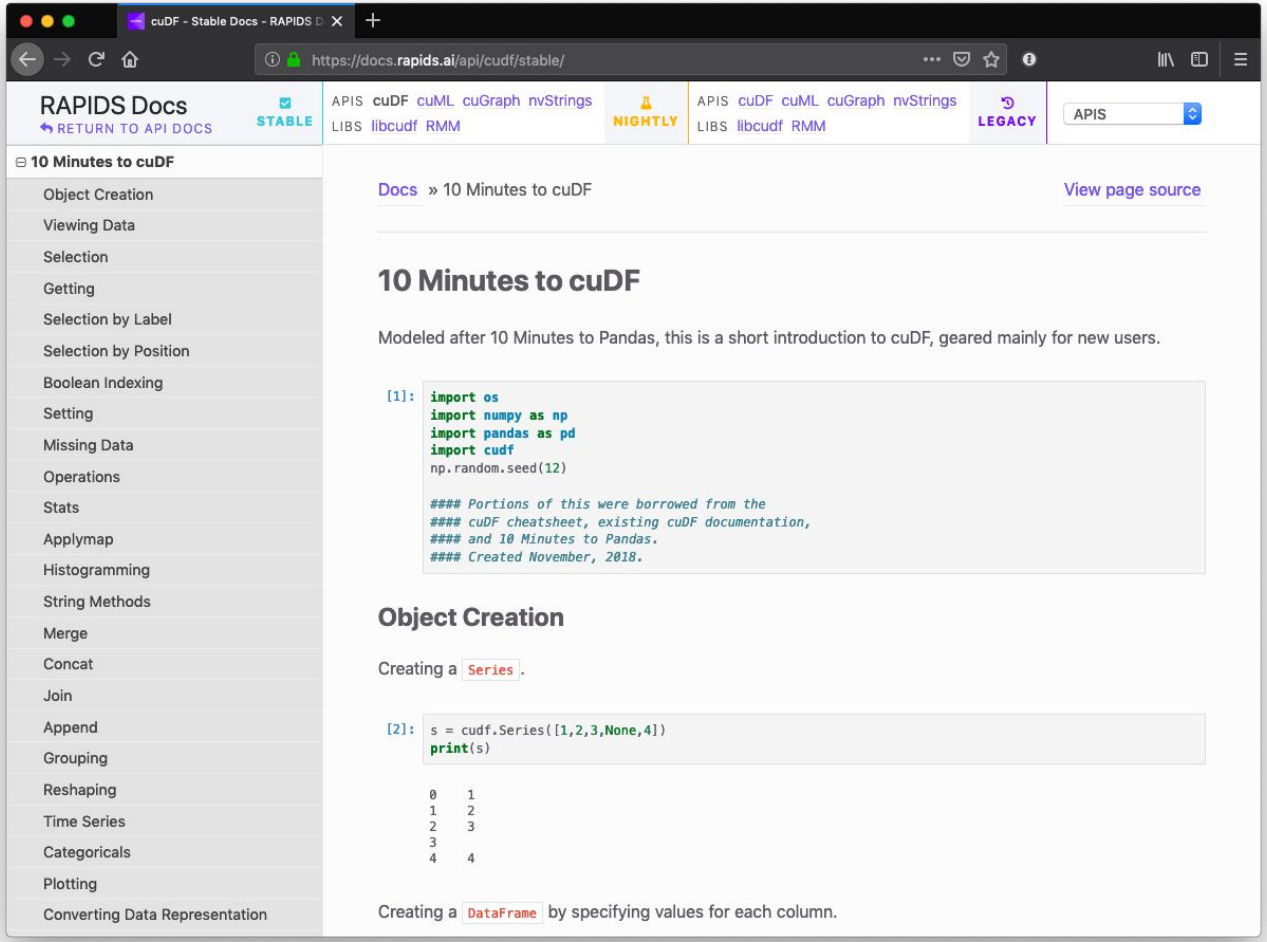
New, improved, and easier to use



https://docs.rapids.ai

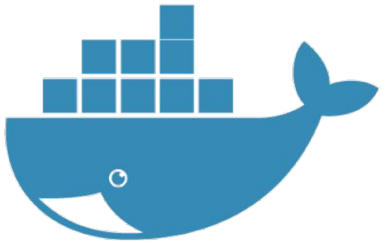
RAPIDS Docs

Easier than ever to get started with cuDF



RAPIDS

How do I get the software?



- https://github.com/rapidsai
- https://anaconda.org/rapidsai/

- https://ngc.nvidia.com/registry/nvidia-rapidsai-rapidsai
- https://hub.docker.com/r/rapidsai/rapidsai/

Join the Movement

Everyone can help!

APACHE ARROW

https://arrow.apache.org/

@ApacheArrow

RAPIDS

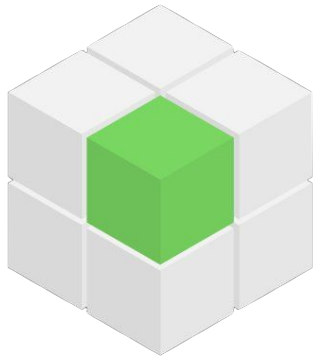
https://rapids.ai

@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 @datametrician
joshuap@nvidia.com

RAPIDS