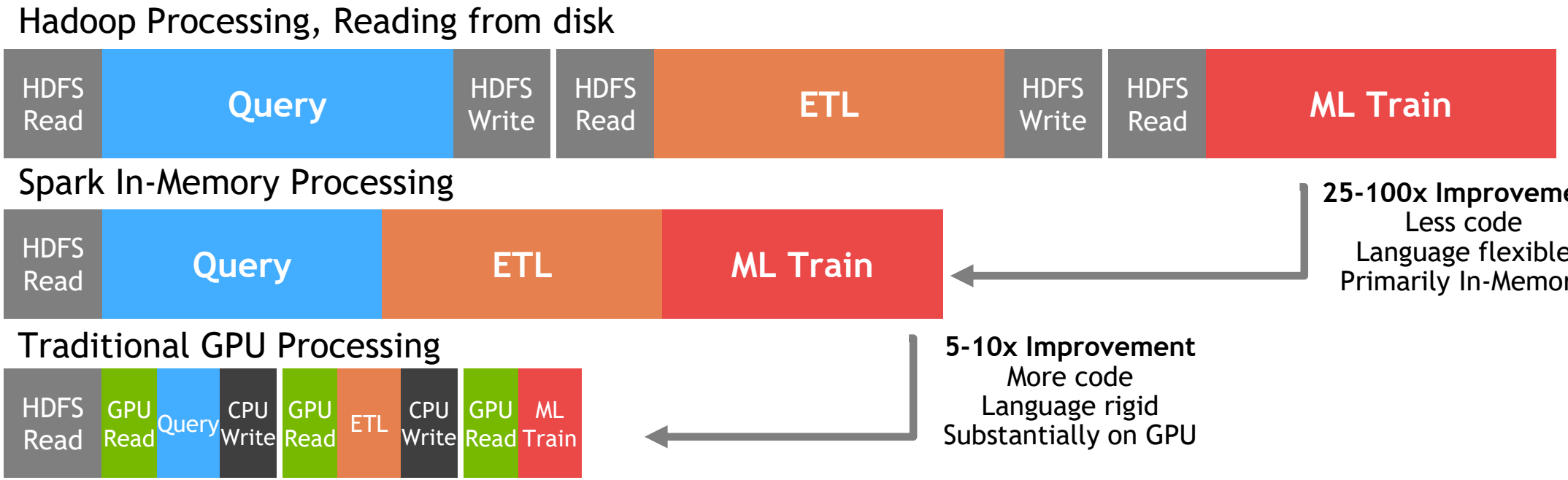
RAPIDS

The Platform Inside and Out

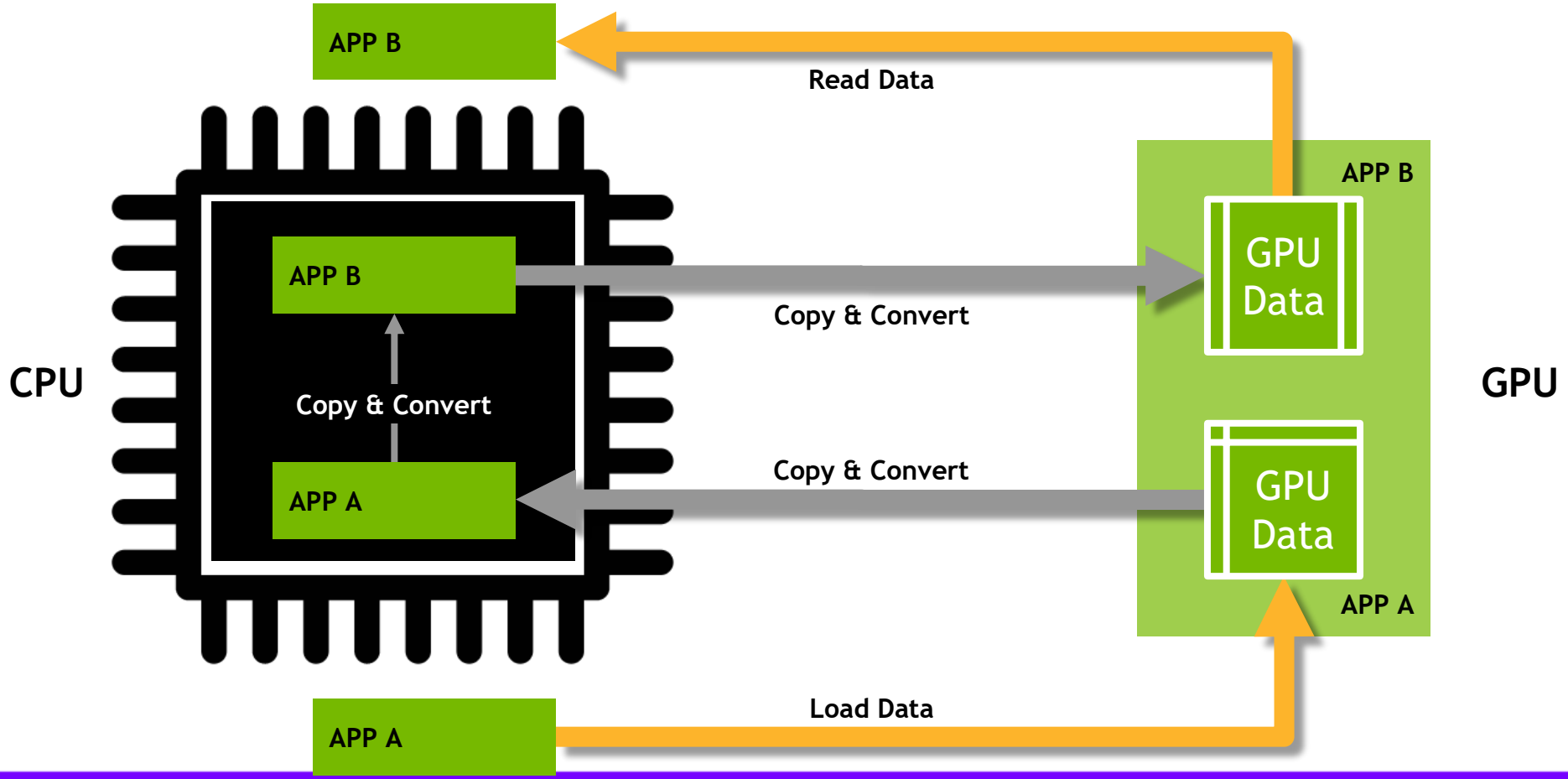
Data Processing Evolution

Faster data access, less data movement



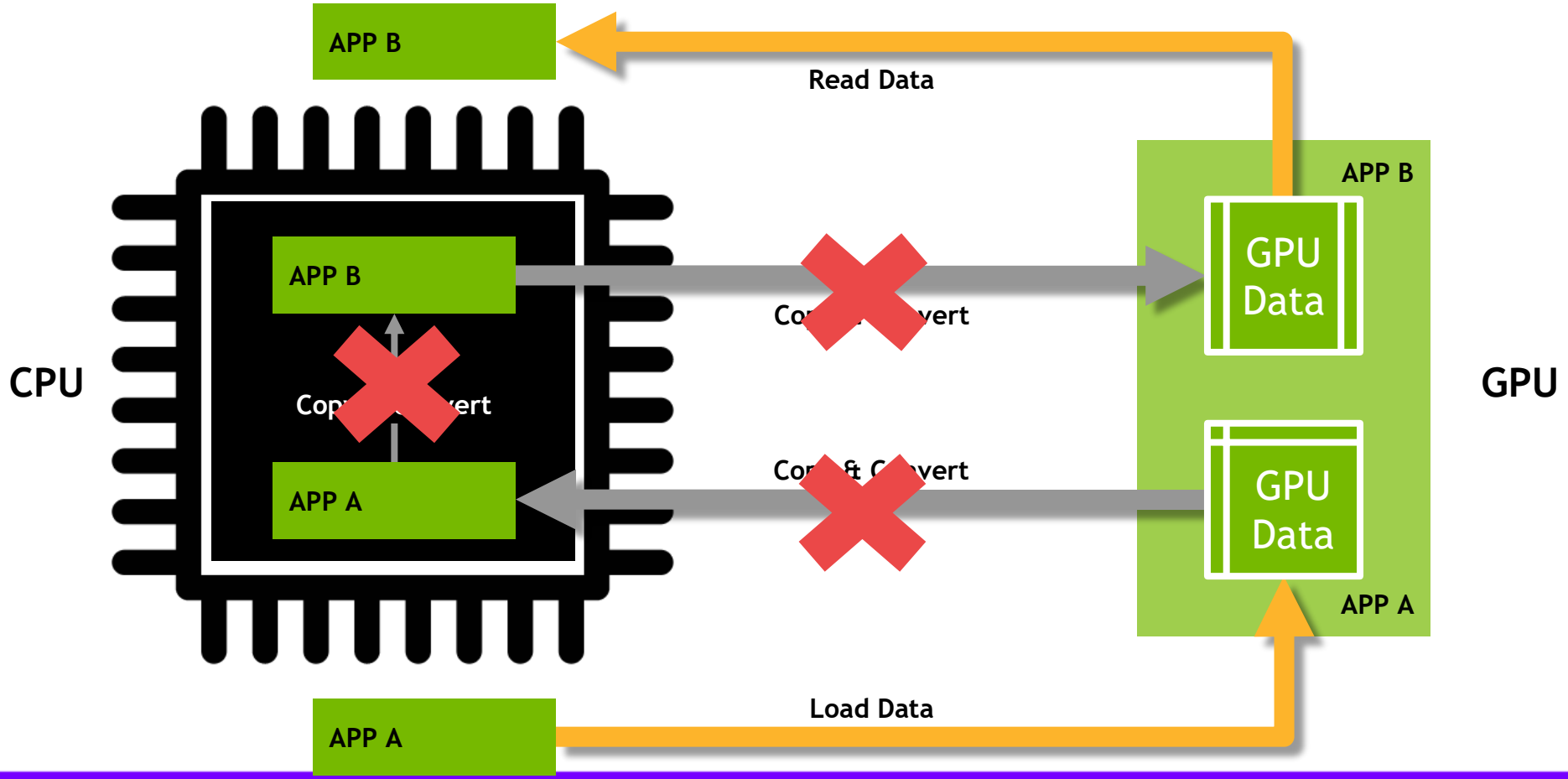
Data Movement and Transformation

The bane of productivity and performance

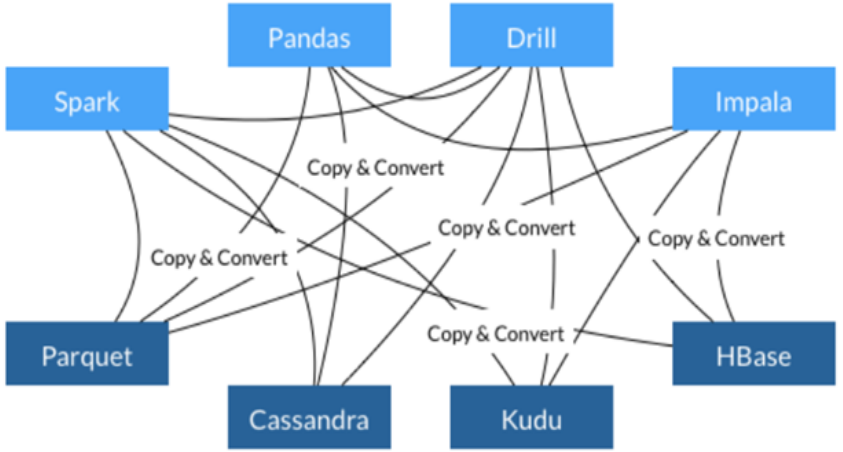


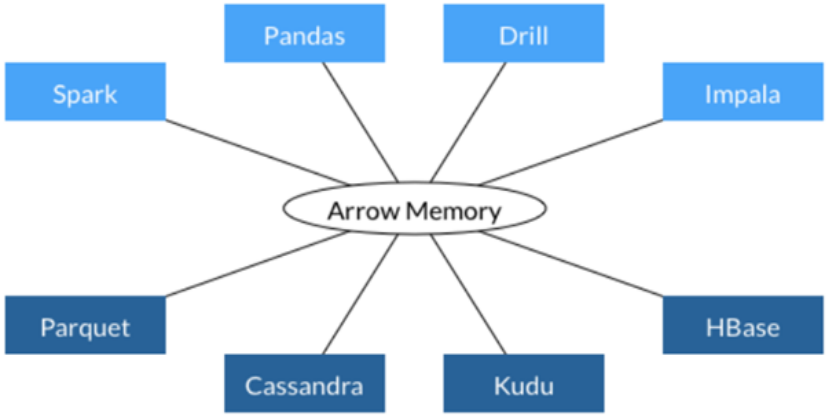
Data Movement and Transformation

What if we could keep data on the GPU?



Learning from Apache Arrow





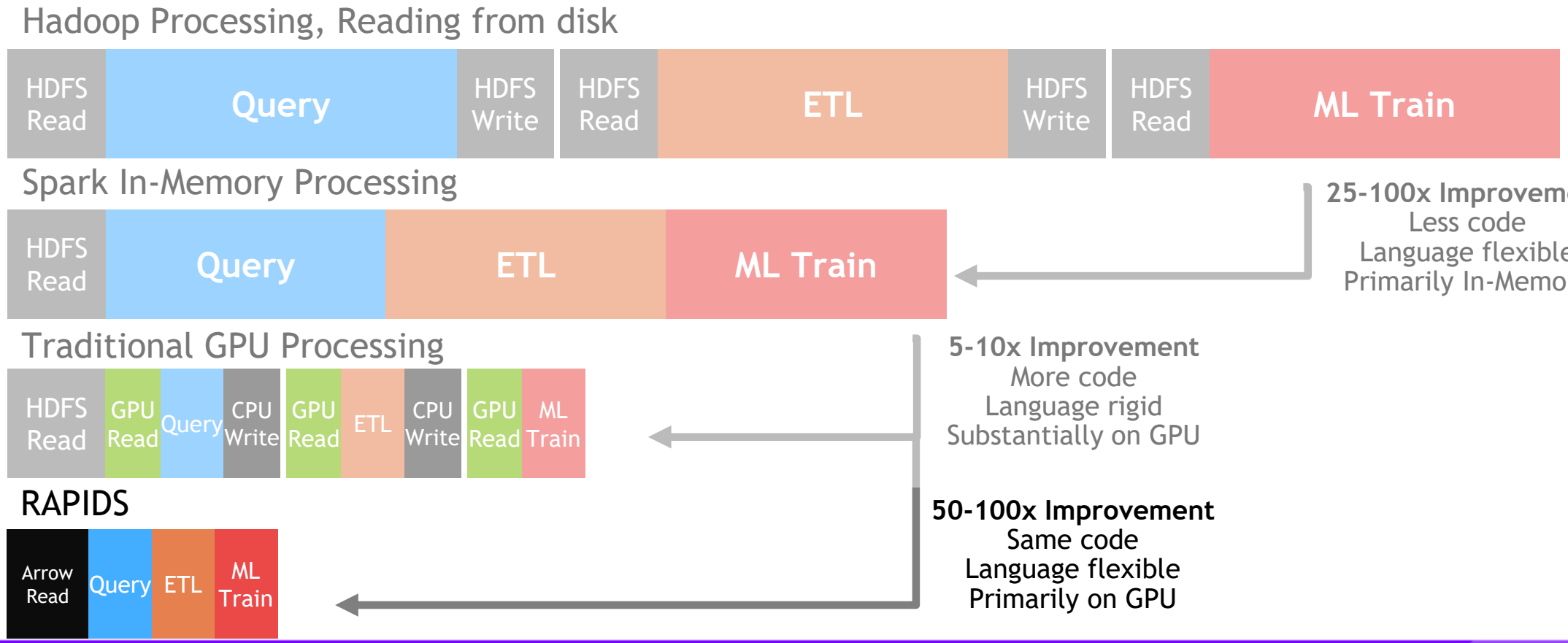
- Each system has its own internal memory format
- 70-80% computation wasted on serialization and deserialization
- Similar functionality implemented in multiple projects

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

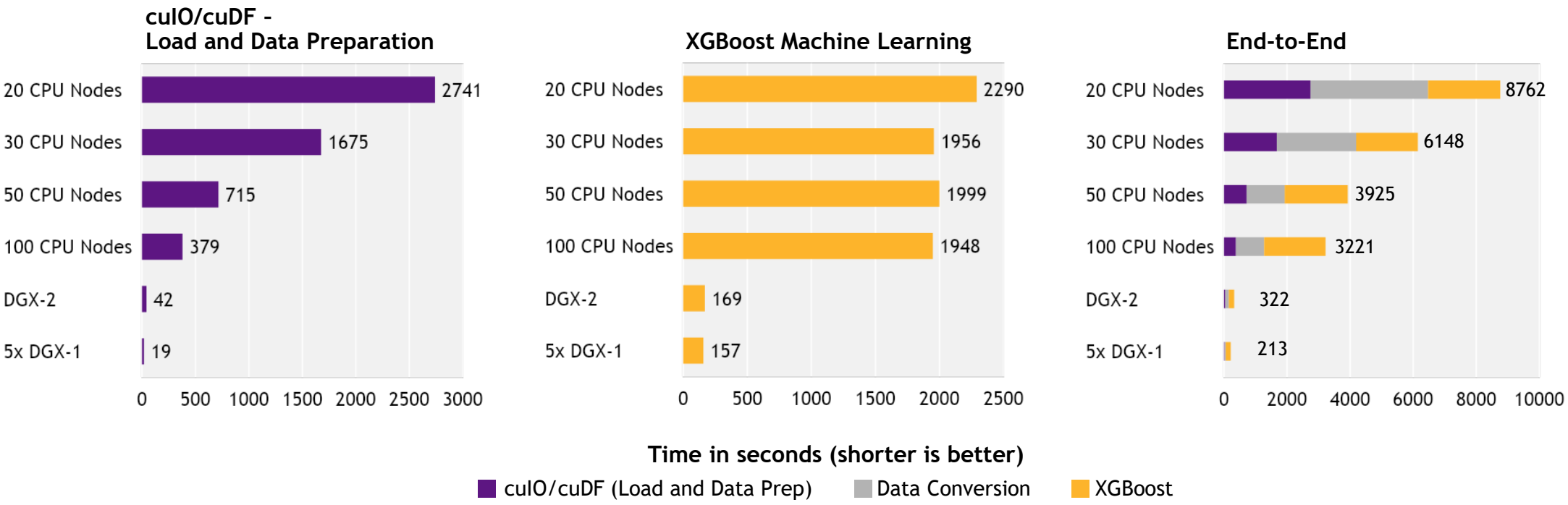
From Apache Arrow Home Page - https://arrow.apache.org/

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

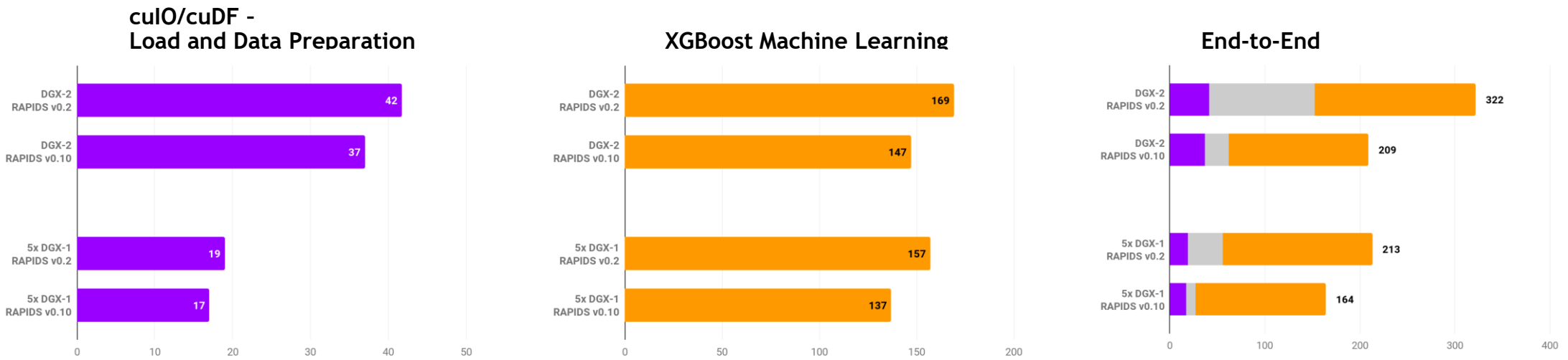
CPU nodes (61 GiB memory, 8 vCPUs, 64-bit platform), Apache Spark

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, UX, and Iteration

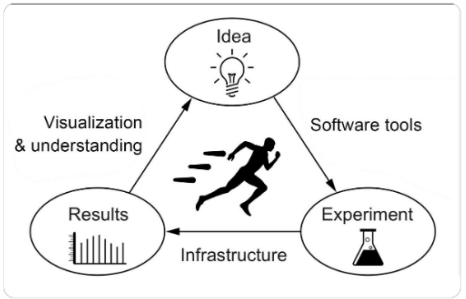
The Way to Win at Data Science

François Chollet ✓
@fchollet

Following

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 were Keras gives you an edge.



12:31 PM - 3 Apr 2019

50 Retweets 158 Likes

5 50 158

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.

1 7 50

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.

2 11 78

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.

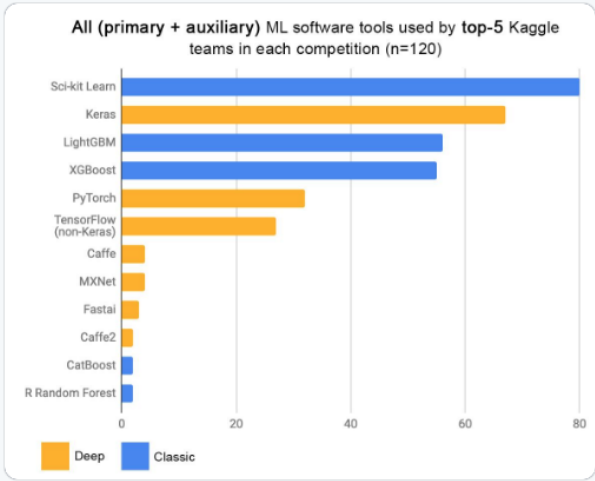
8 8 74

Joshua Patterson @datametrician · Apr 3
Replying to @fchollet
This is the fundamental believe 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!

2 11

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).



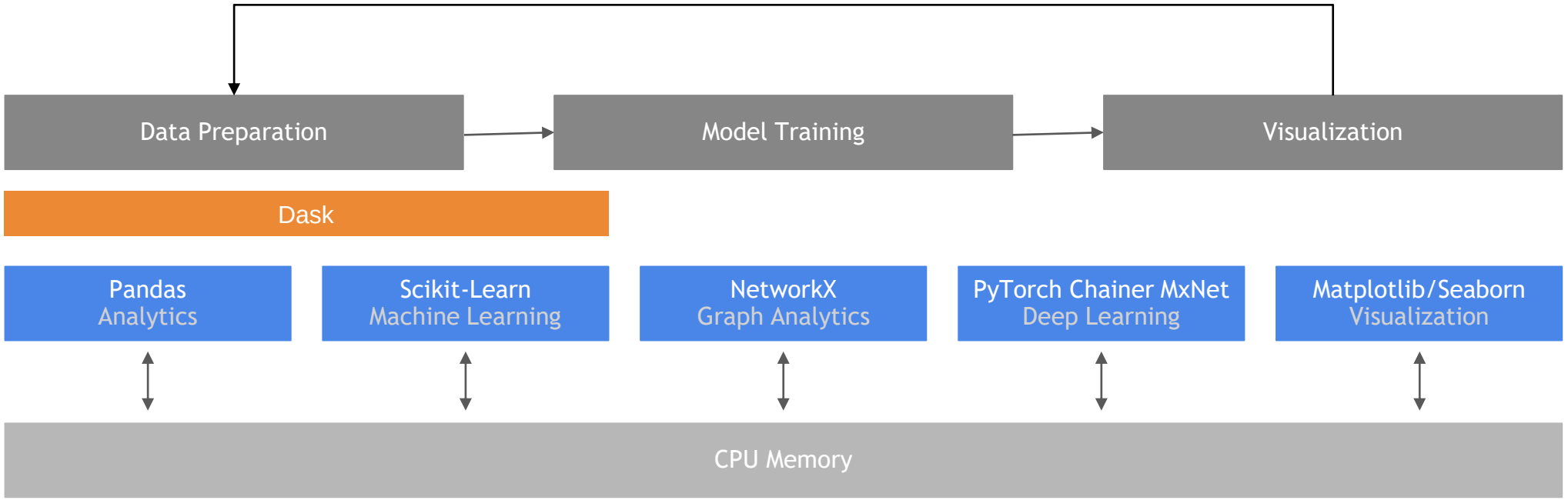
4 44 129

kaggle

RAPIDS Core

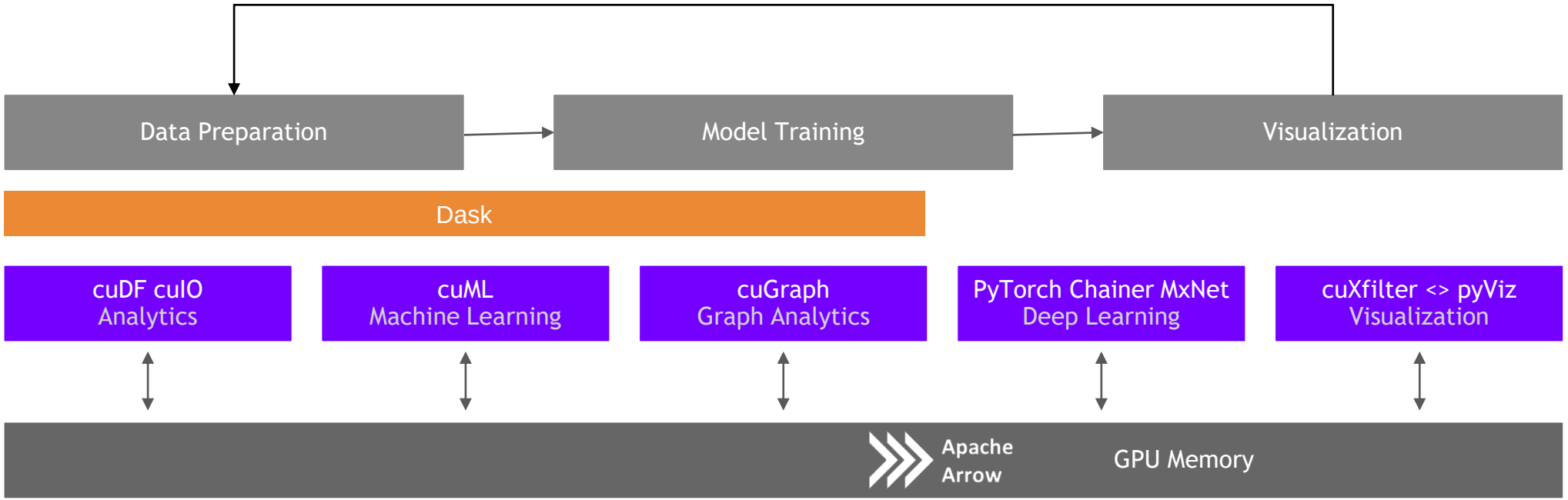
Open Source Data Science Ecosystem

Familiar Python APIs



RAPIDS

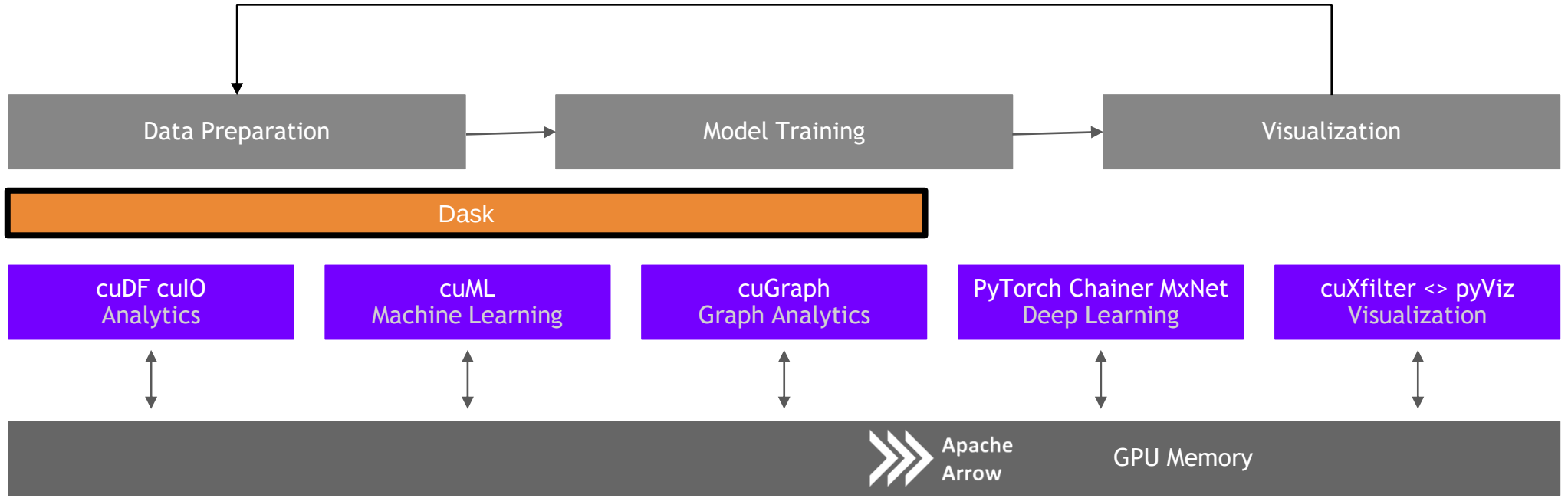
End-to-End Accelerated GPU Data Science



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) https://github.com/rapidsai/ucx-py
- Will provide best communication performance, to Dask based on available hardware on nodes/cluster

Scale up with RAPIDS

Scale Up / Accelerate

RAPIDS and Others

Accelerated on single GPU

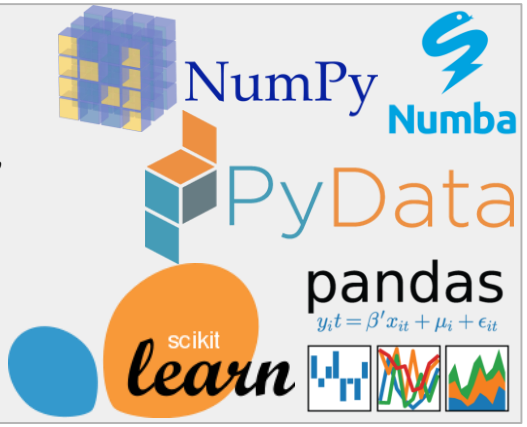
NumPy -> CuPy/PyTorch/..
Pandas -> cuDF
Scikit-Learn -> cuML
Numba -> Numba

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

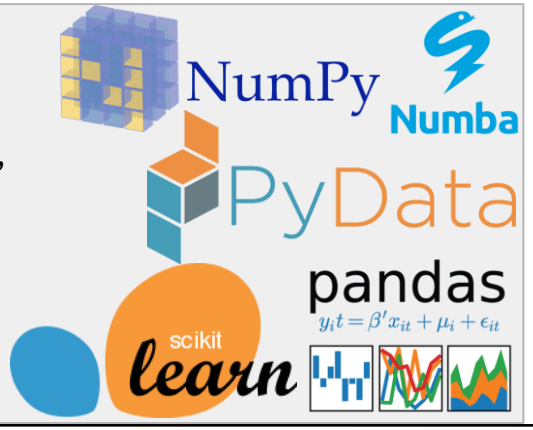
RAPIDS

DASK

PyData

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

Single CPU core
In-memory data



Dask

Multi-core and Distributed PyData

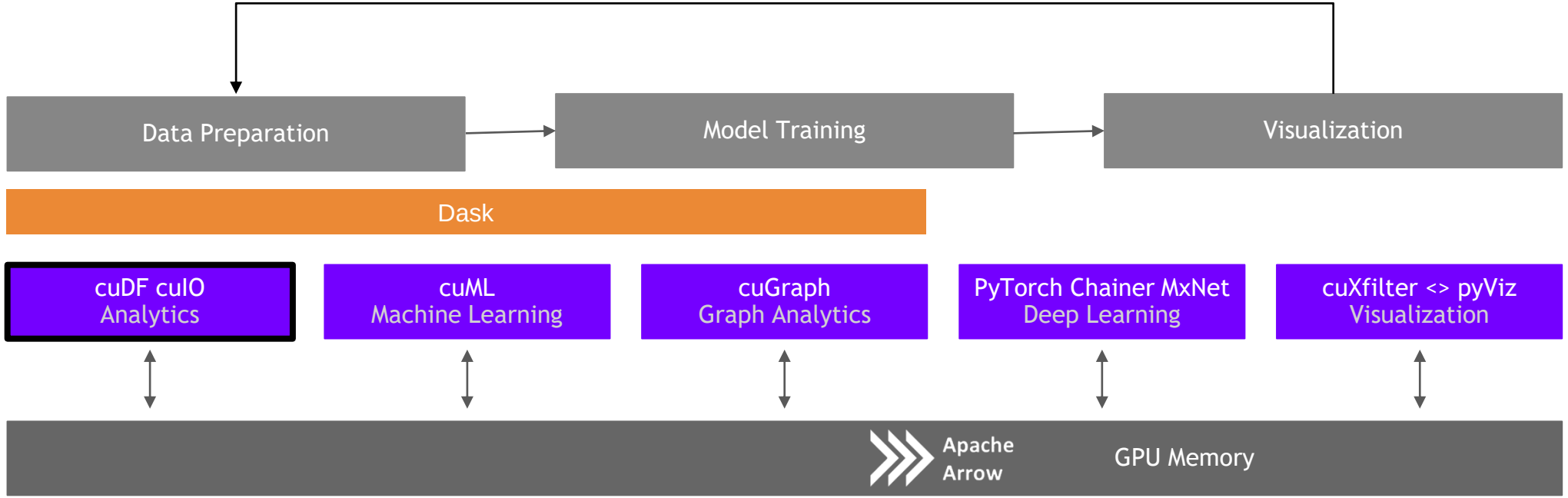
NumPy -> Dask Array
Pandas -> Dask DataFrame
Scikit-Learn -> Dask-ML
… -> Dask Futures

DASK

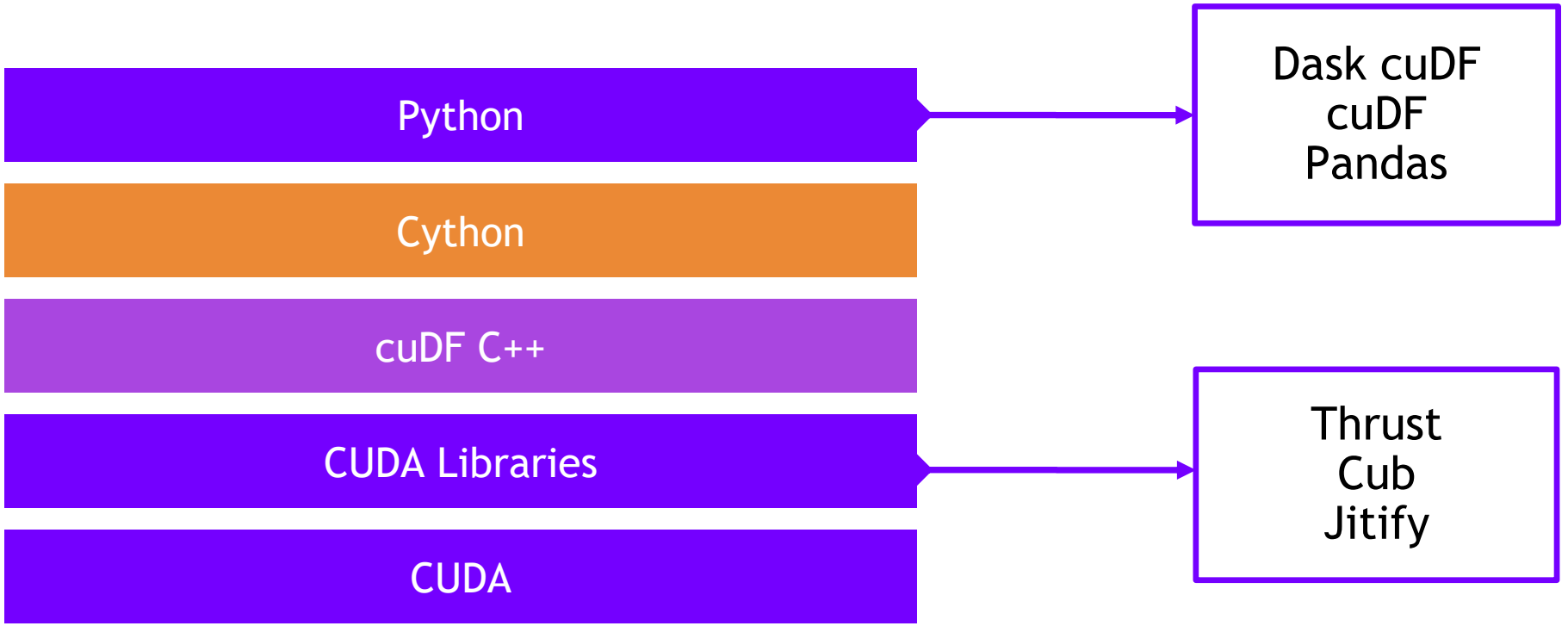
cuDF

RAPIDS

GPU Accelerated data wrangling and feature engineering



ETL Technology Stack



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]:

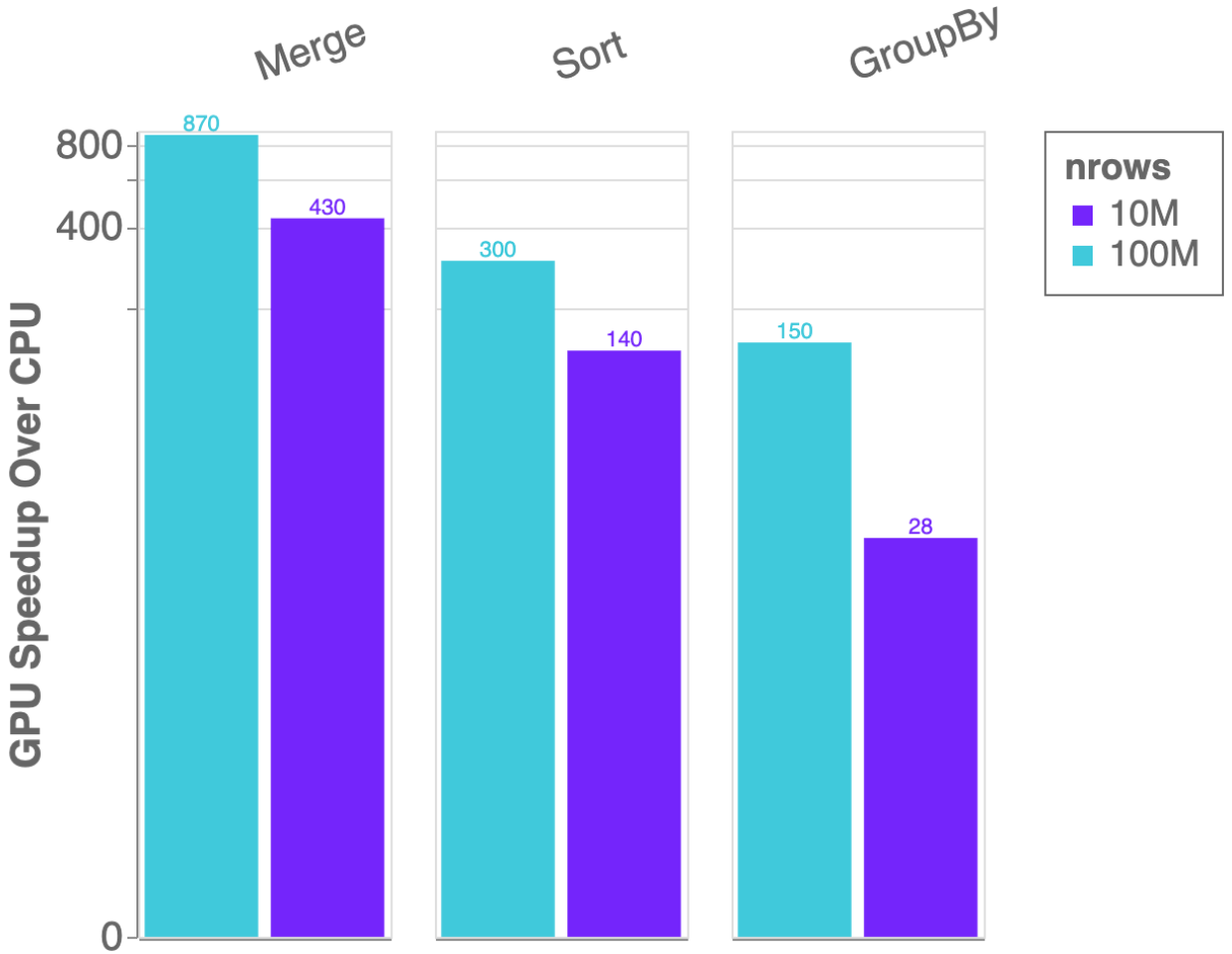
	User_ID	Product_ID	Gender	Age	Occupation	City_Category	Stay_In_Current_City_Years	Marital_Status	Product_Cat
0	1000001	P00069042	F	0-17	10	A	2	0	3
1	1000001	P00248942	F	0-17	10	A	2	0	1
2	1000001	P00087842	F	0-17	10	A	2	0	12
3	1000001	P00085442	F	0-17	10	A	2	0	12
4	1000002	P00285442	M	55+	16	C	4+	0	8

```
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 v0.10, Pandas 0.24.2

Running on NVIDIA DGX-1:

GPU: NVIDIA Tesla V100 32GB
CPU: Intel(R) Xeon(R) CPU E5-2698 v4 @ 2.20GHz

Benchmark Setup:

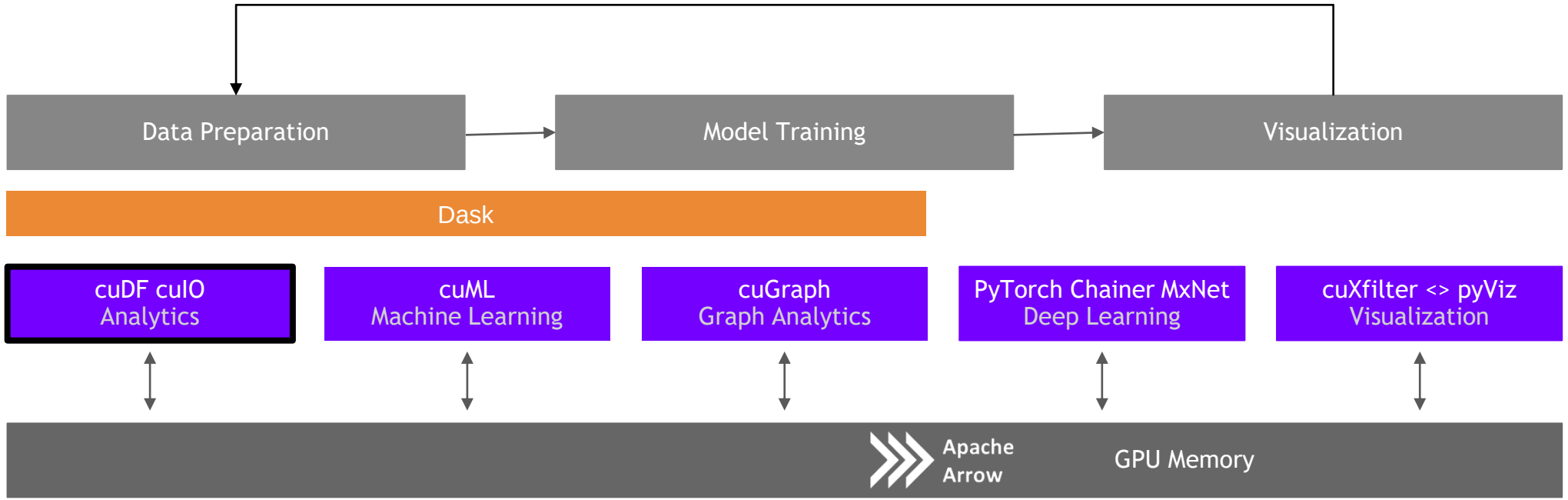
DataFrames: 2x int32 columns key columns, 3x int32 value columns

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



Extraction is the Cornerstone

cuIO for Faster Data Loading

- Follow Pandas APIs and provide >10x speedup

- CSV Reader - v0.2, CSV Writer v0.8
- Parquet Reader – v0.7, Parquet Writer v0.11
- 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

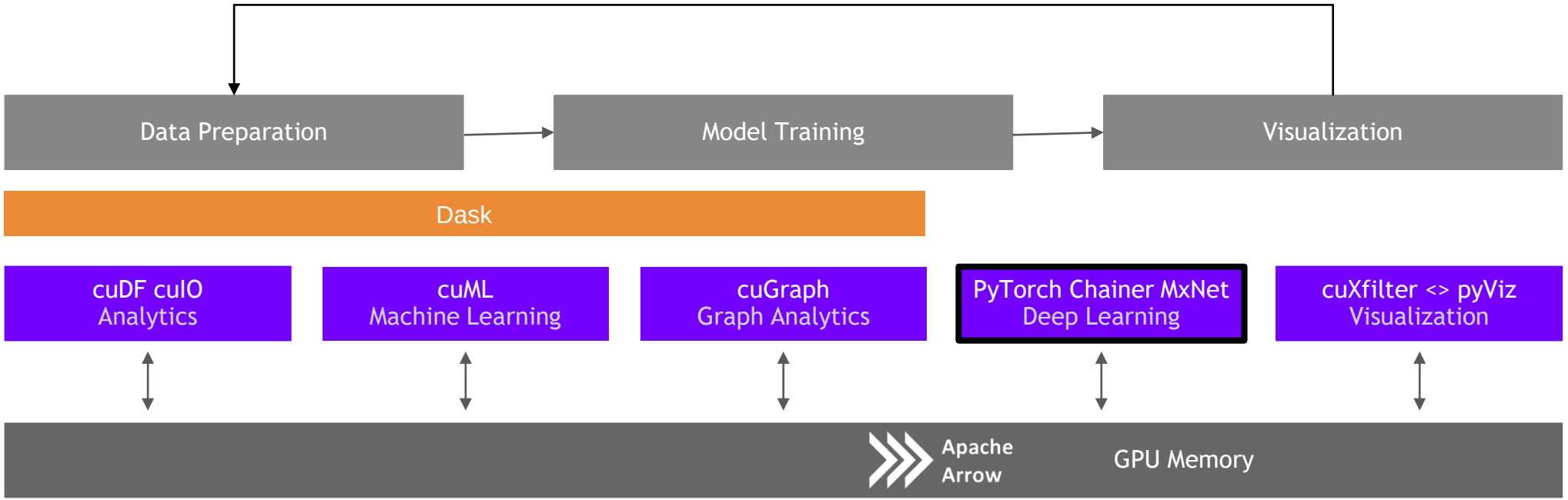
```
1]: import pandas, cudf

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



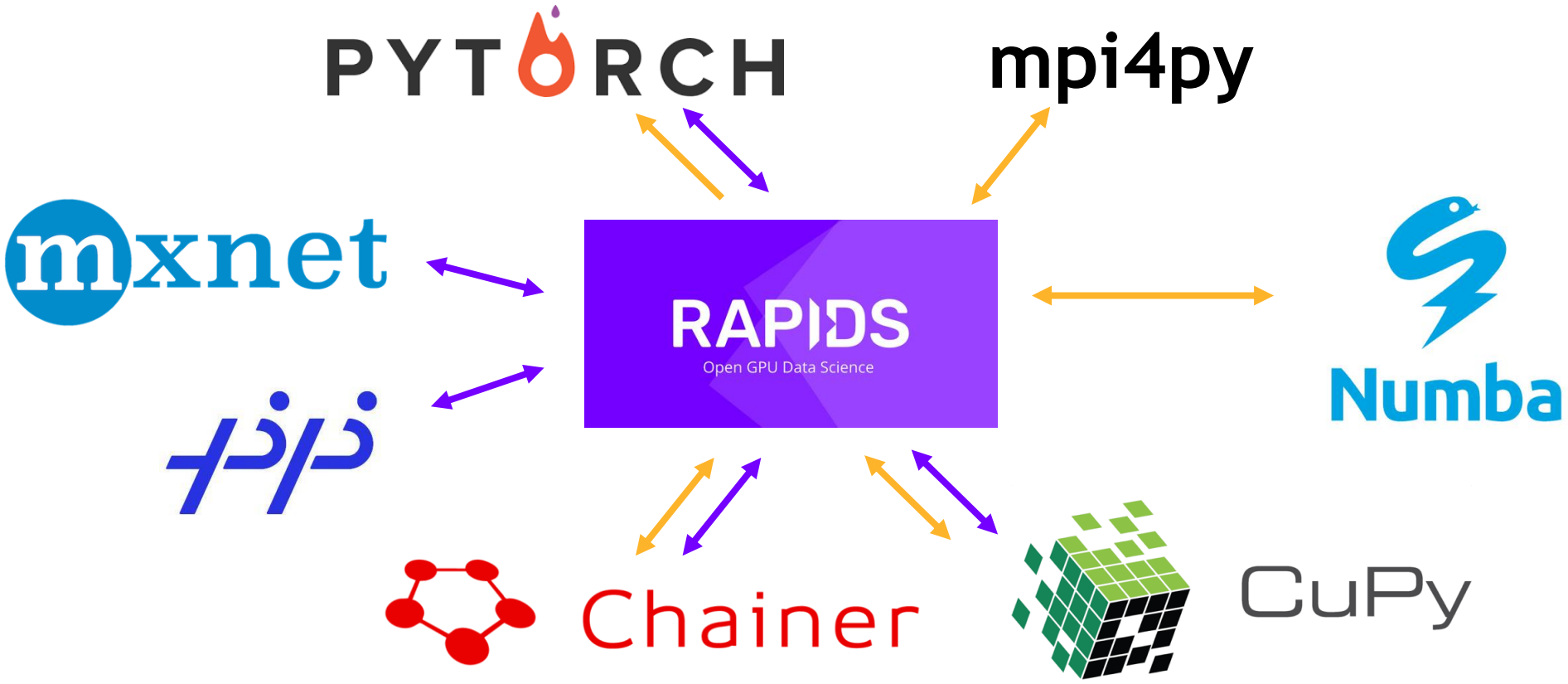
Interoperability for the Win

DLPack and __cuda_array_interface__

mpi4py

Interoperability for the Win

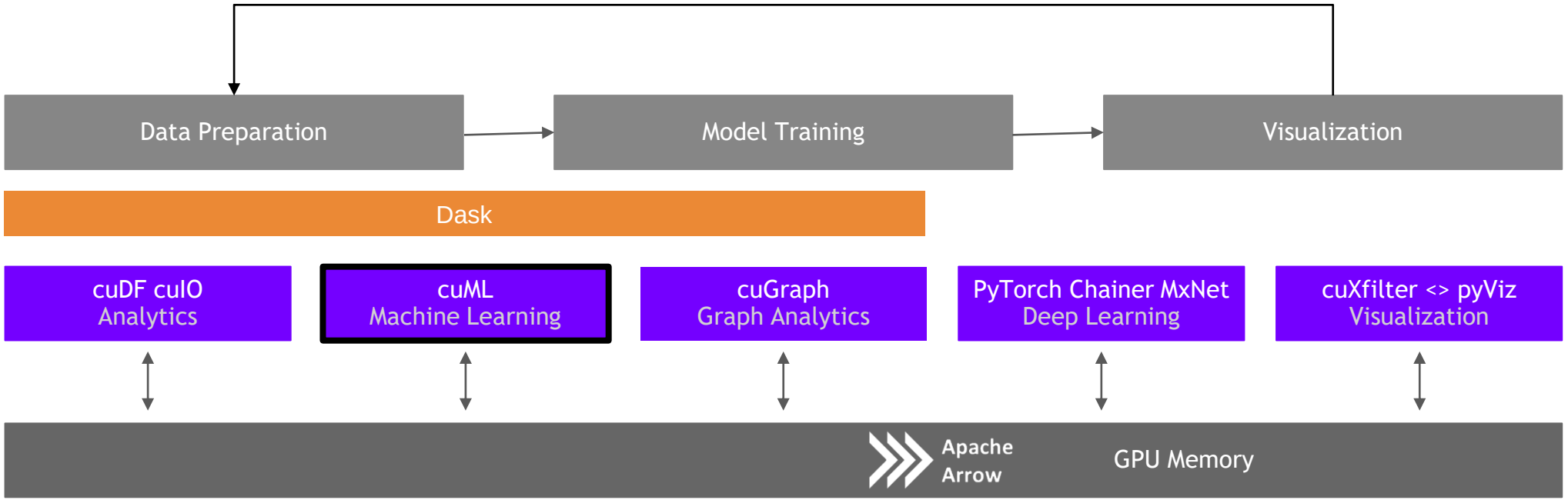
DLPack and __cuda_array_interface__



cuML

Machine Learning

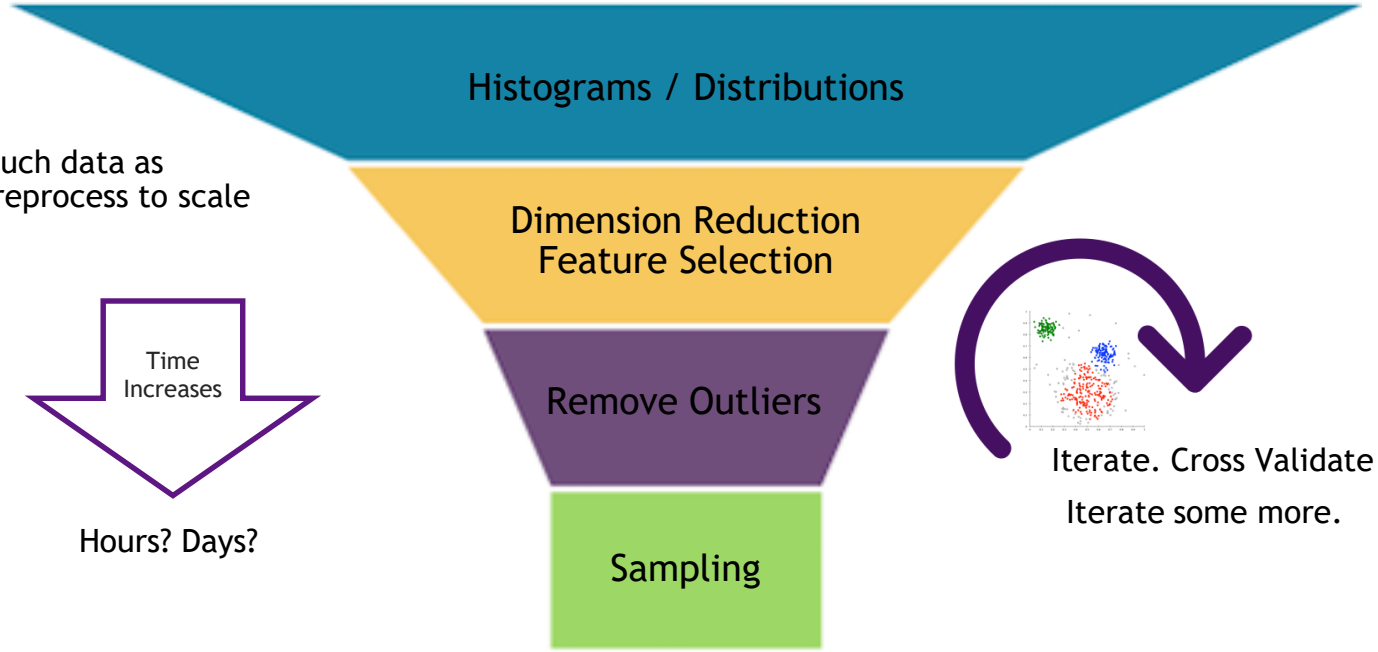
More models more problems



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

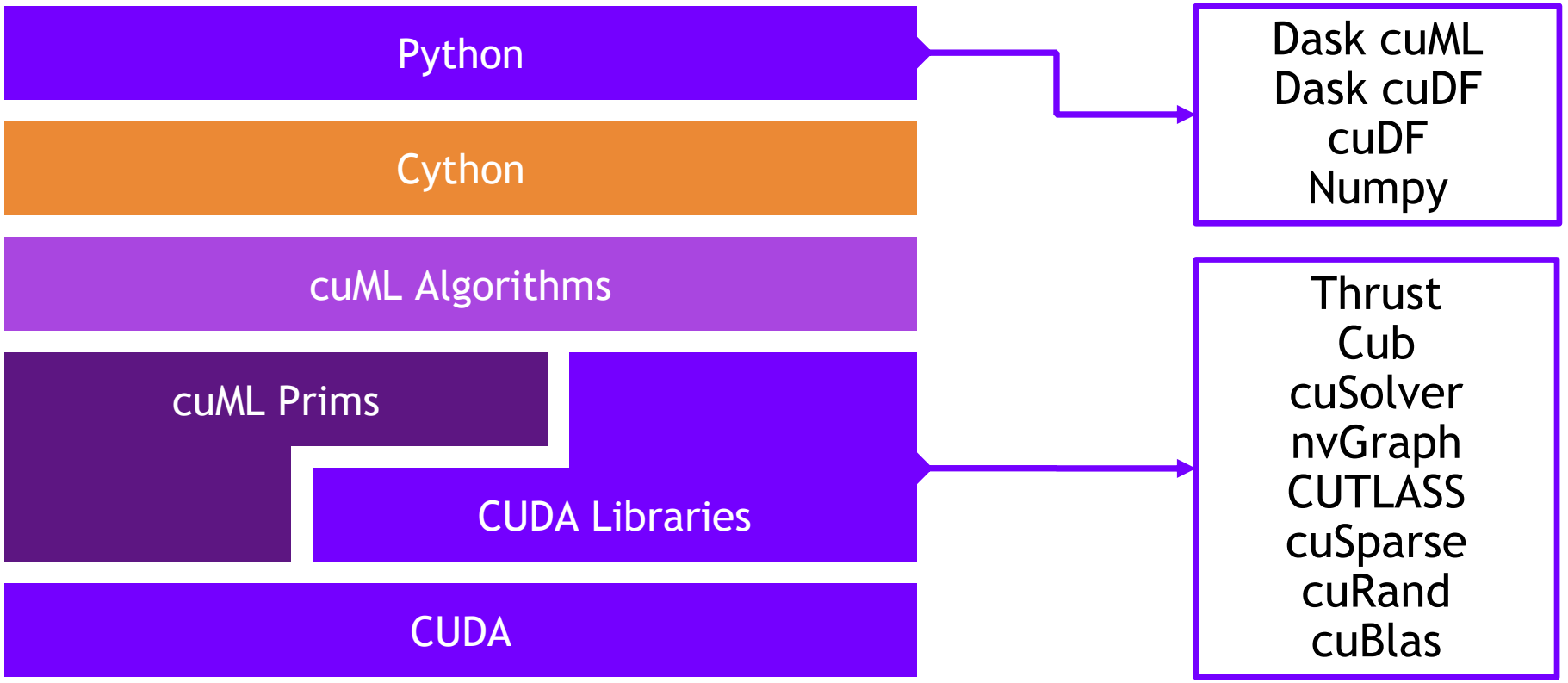
Hours? Days?

Iterate. Cross Validate & Grid Search.

Iterate some more.

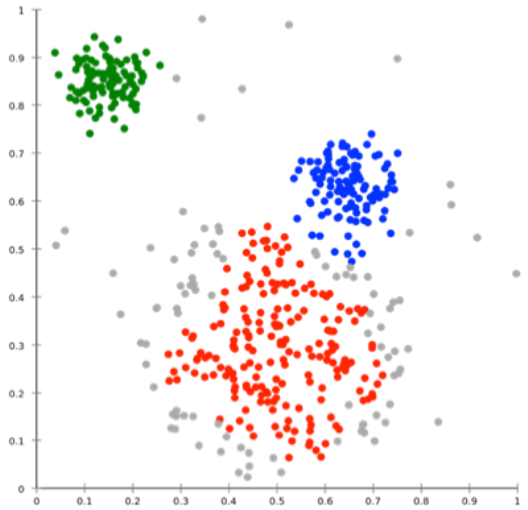
Meet reasonable speed vs accuracy tradeoff

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 Machine Classification**
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 Kalman Filtering

Key:

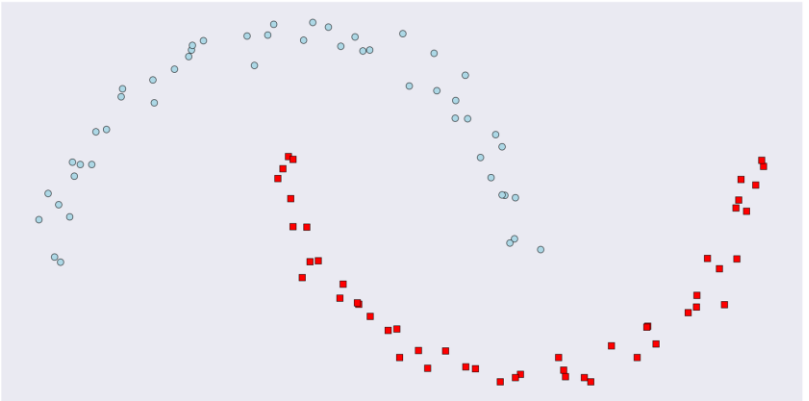
- Preexisting
- **NEW for 0.10**

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)
```

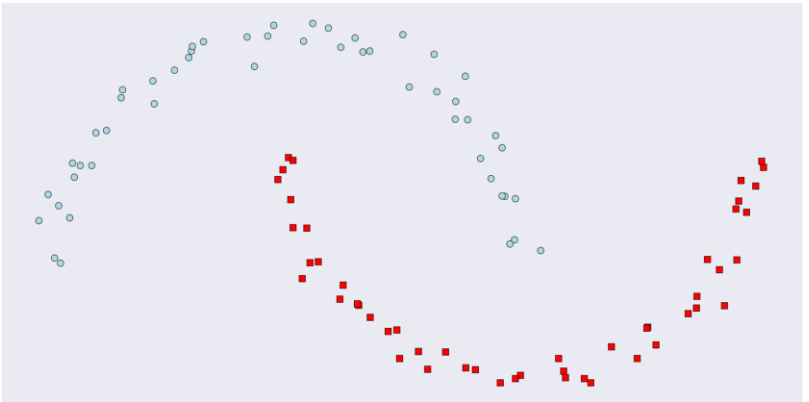


RAPIDS matches common Python APIs

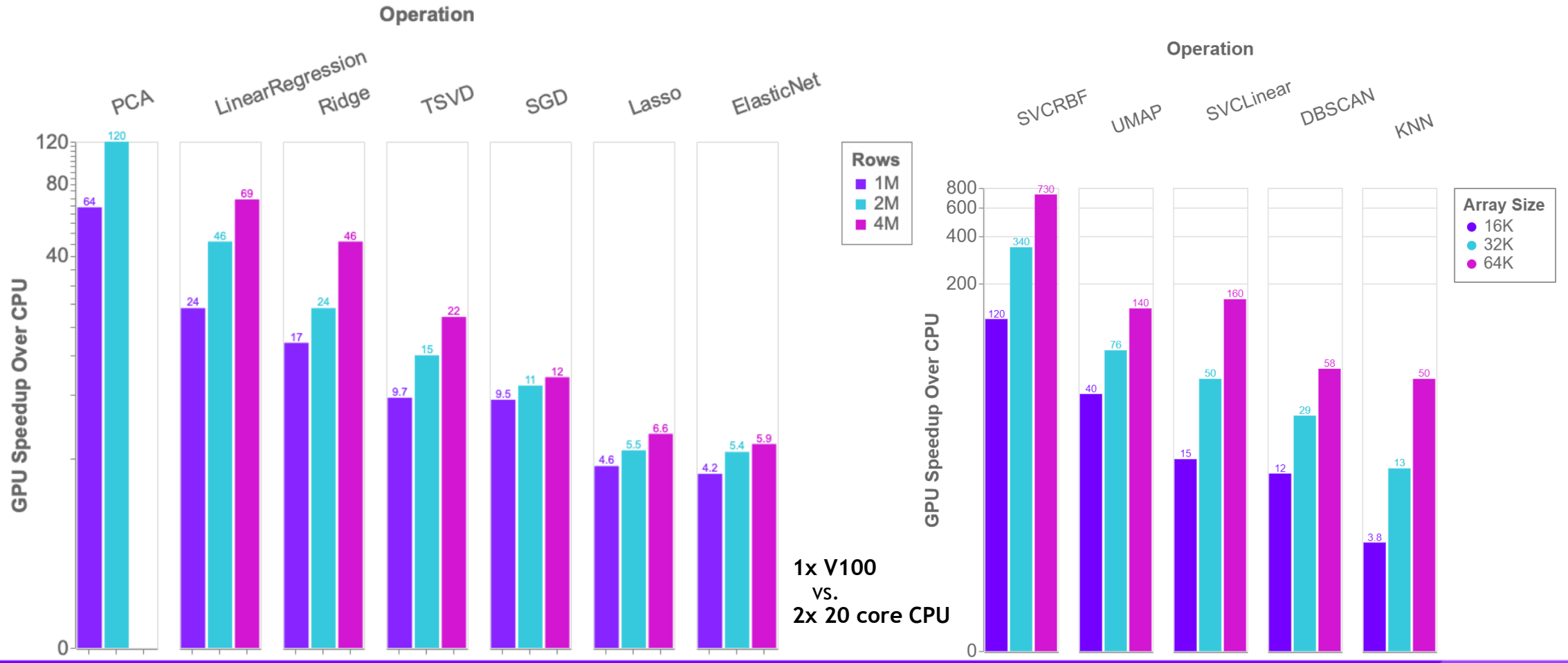
GPU-Accelerated Clustering

```
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

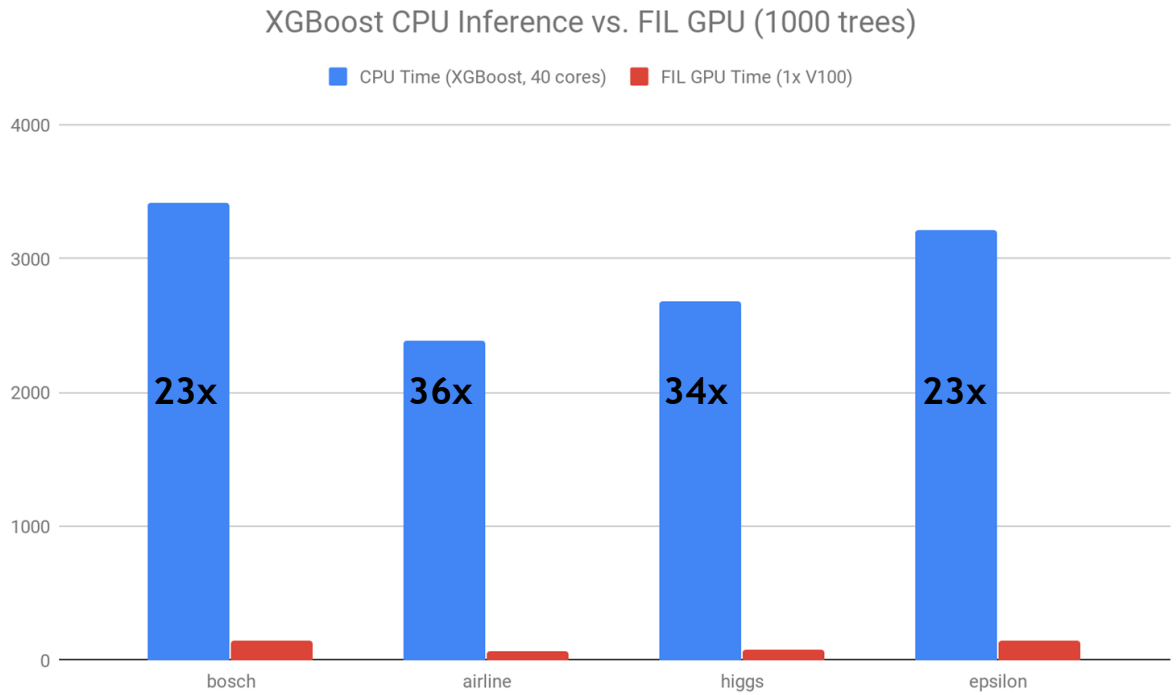


Forest Inference

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 and LightGBM today, scikit-learn RF and 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



Road to 1.0

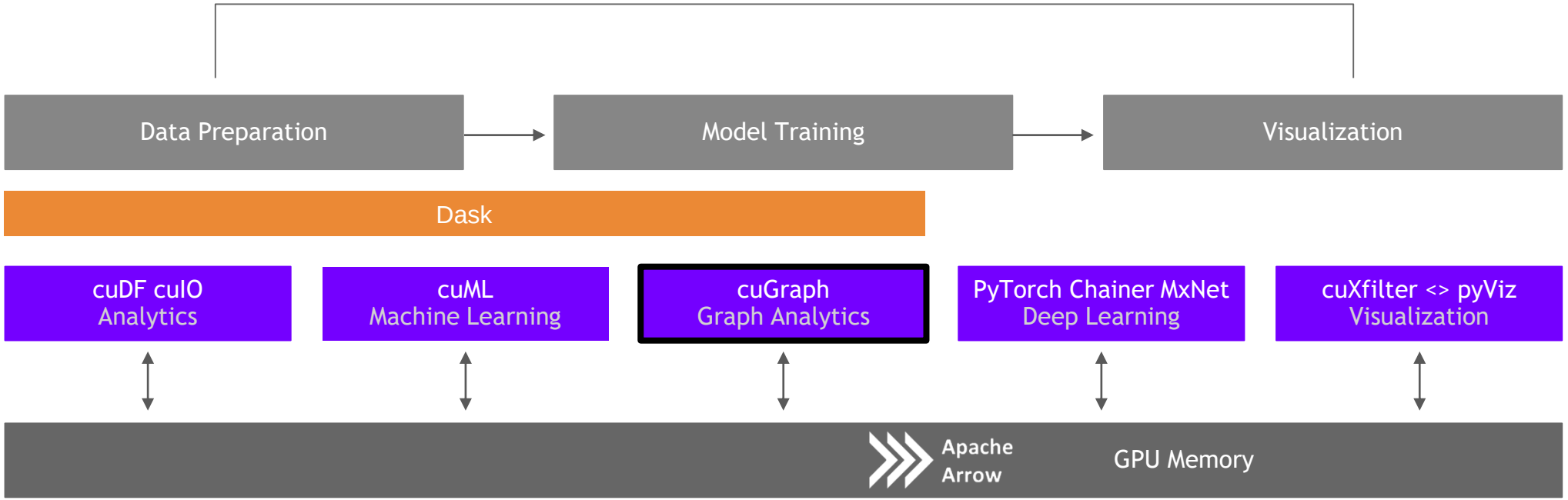
March 2020 - RAPIDS 0.13

cuML	Single-GPU	Multi-Node-Multi-GPU
Gradient Boosted Decision Trees (GBDT)	✓	✓
GLM	✓	✓
Logistic Regression	✓	✓
Random Forest	✓	✓
K-Means	✓	✓
K-NN	✓	✓
DBSCAN	✓	✓
UMAP	✓	✓
ARIMA & Holt-Winters	✓	
Kalman Filter	✓	
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

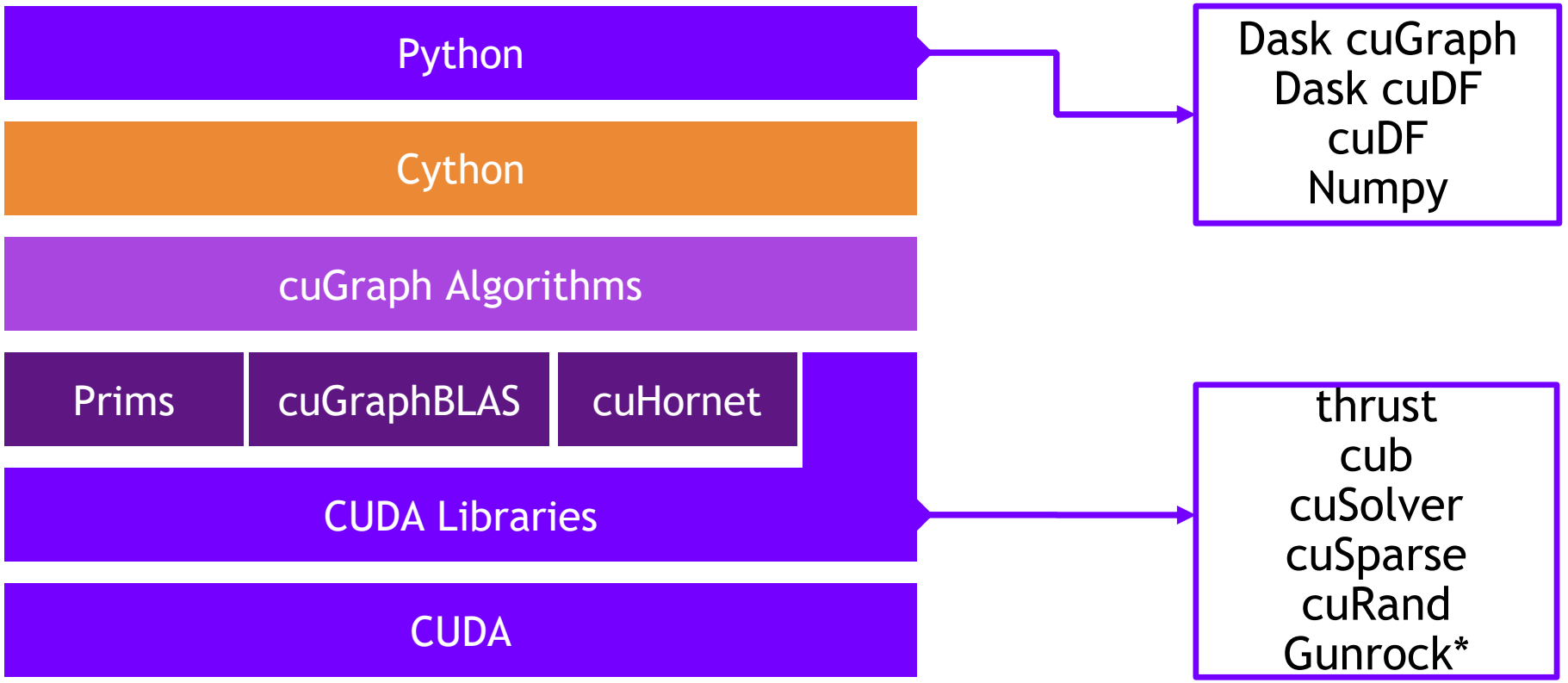
Seamless Integration with cuDF and cuML

- Property Graph support via DataFrames

Growing Functionality

- Extensive collection of algorithm, primitive, and utility functions

Graph Technology Stack

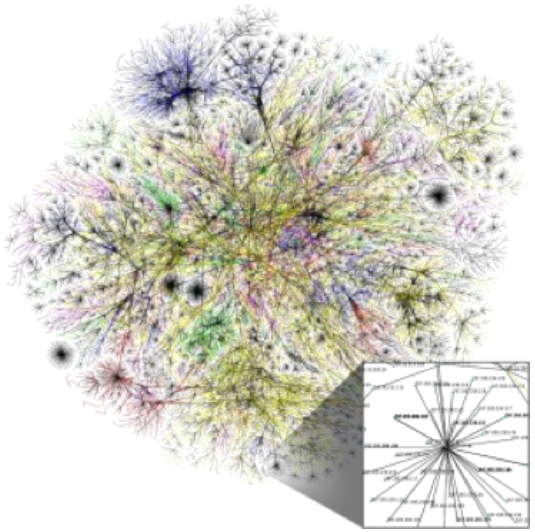


nvGRAPH has been Opened Sourced and integrated into cuGraph. A legacy version is available in a RAPIDS GitHub repo

* Gunrock is from UC Davis

Algorithms

GPU-accelerated NetworkX



Query Language

Multi-GPU

Utilities

Renumbering

More to come!

Community

Spectral Clustering
- Balanced-Cut
- Modularity Maximization

Louvain
Subgraph Extraction
KCore

Components

Weakly Connected Components
Strongly Connected Components

Link Analysis

Page Rank (Multi-GPU)
Personal Page Rank
Katz

Link Prediction

Jaccard
Weighted Jaccard
Overlap Coefficient

Traversal

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

Structure

Triangle Counting
COO-to-CSR (Multi-GPU)
Transpose

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!

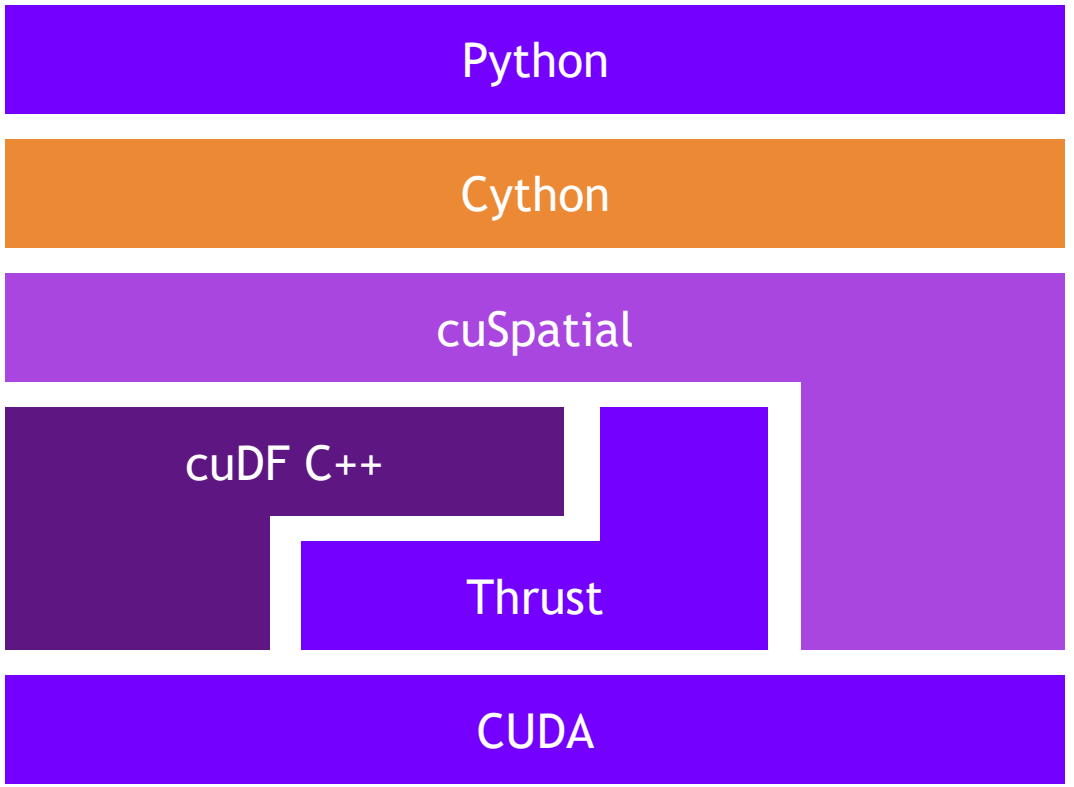
Road to 1.0

March 2020 - RAPIDS 0.13

cuGraph	Single-GPU	Multi-Node-Multi-GPU
Jaccard and Weighted Jaccard		
Page Rank		
Personal Page Rank		
SSSP		
BFS		
Triangle Counting		
Subgraph Extraction		
Katz Centrality		
Betweenness Centrality		
Connected Components (Weak and Strong)		
Louvain		
Spectral Clustering		
K-Cores		

cuSpatial

cuSpatial Technology Stack



cuSpatial 0.10

Breakthrough Performance & Ease of Use

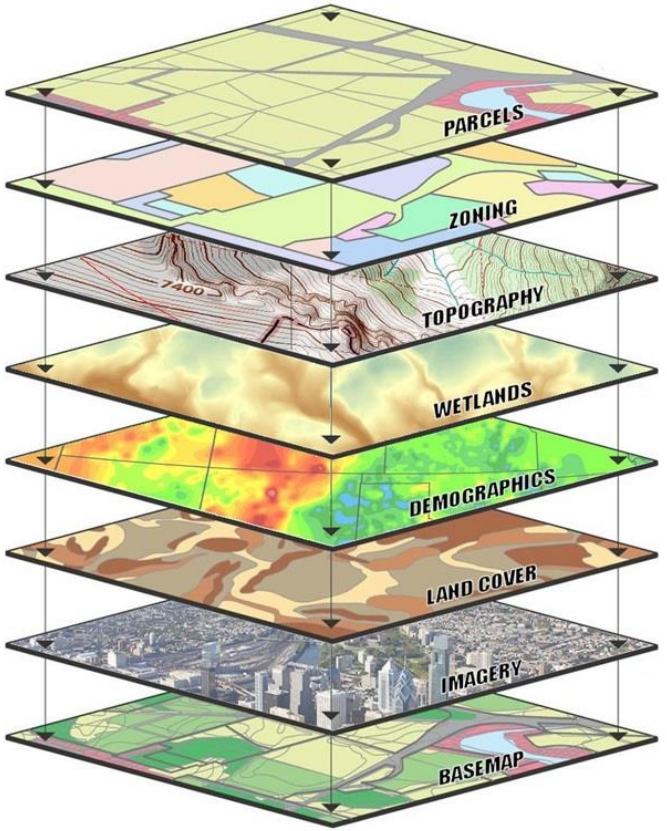
- Up to 1000x faster than CPU spatial libraries
- Python and C++ APIs for maximum usability and integration

Growing Functionality

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

Seamless Integration into RAPIDS

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



cuSpatial

0.10 and Beyond

Layer	0.10/0.11 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	Line intersecting polygon, Other distance functions, Polygon intersection, union
Geo-representation	Shape primitives, points, polylines, polygons	Additional shape primitives

RAPIDS in Enterprise

Apache Spark

Flexible, in-memory data processing for Big Data

Easy Development

- Rich APIs for Scala, Java, and Python
- Interactive shell

Flexible Extensible API

- APIs for different types of workloads:
  - Batch
  - Streaming
  - Machine Learning
  - Graph

Fast Batch & Stream Processing

- In-Memory processing and caching

Easy Development

High Productivity Language Support

Python

```
lines = sc.textFile(...)
lines.filter(lambda s: “ERROR” in s).count()
```

Scala

```
val lines = sc.textFile(...)
lines.filter(s => s.contains(“ERROR”)).count()
```

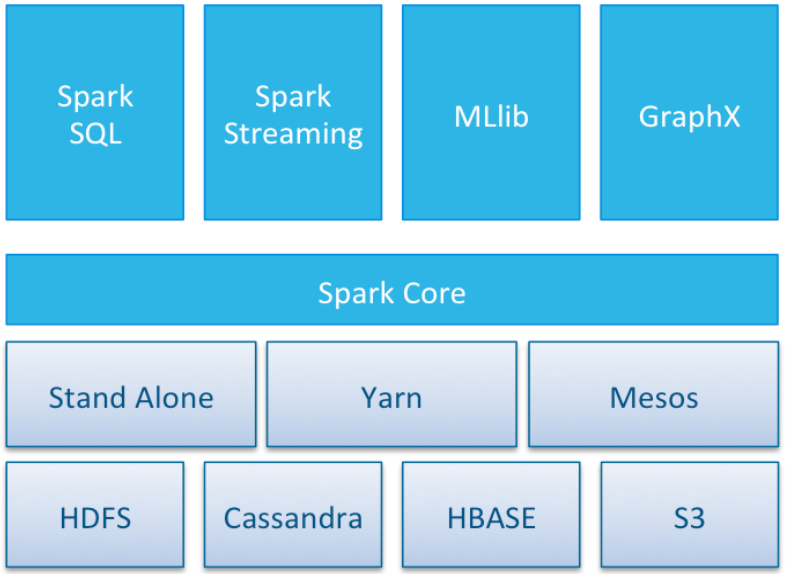
Java

```
JavaRDD<String> lines = sc.textFile(...);
lines.filter(new Function<String, Boolean>() {
  Boolean call(String s) {
    return s.contains(“error”);
  }
}).count();
```

- Native support for multiple languages with identical APIs
  - Scala, Java, Python
- Use of closures, iterations, and other common language constructs to minimize code
  - 2-5x less code

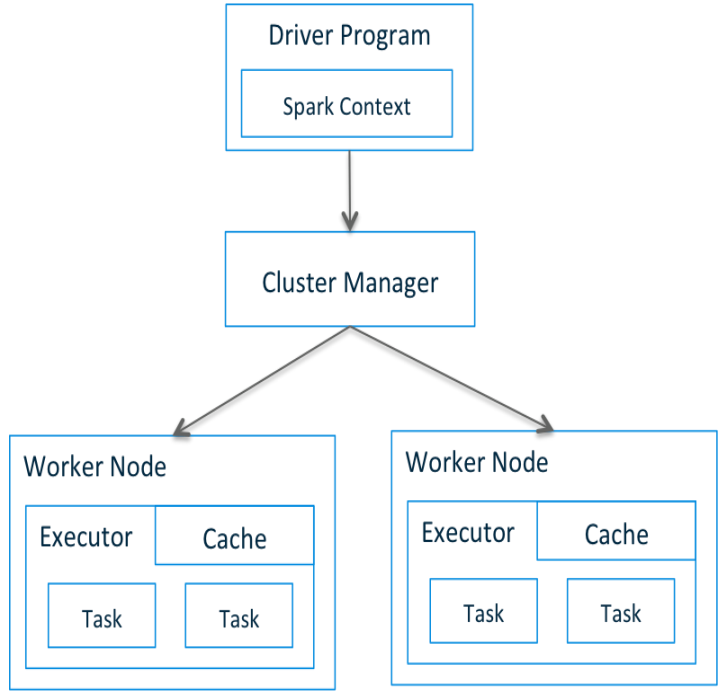
Overview: Spark Components

Spark is a fast, general purpose cluster computing platform.
Spark takes advantage of parallelism, by distributing processing across a cluster of nodes, in order to provide fast processing of data.



Each Spark application gets its own executor processes which stay up for the duration of the application. The executor runs tasks in multiple threads.

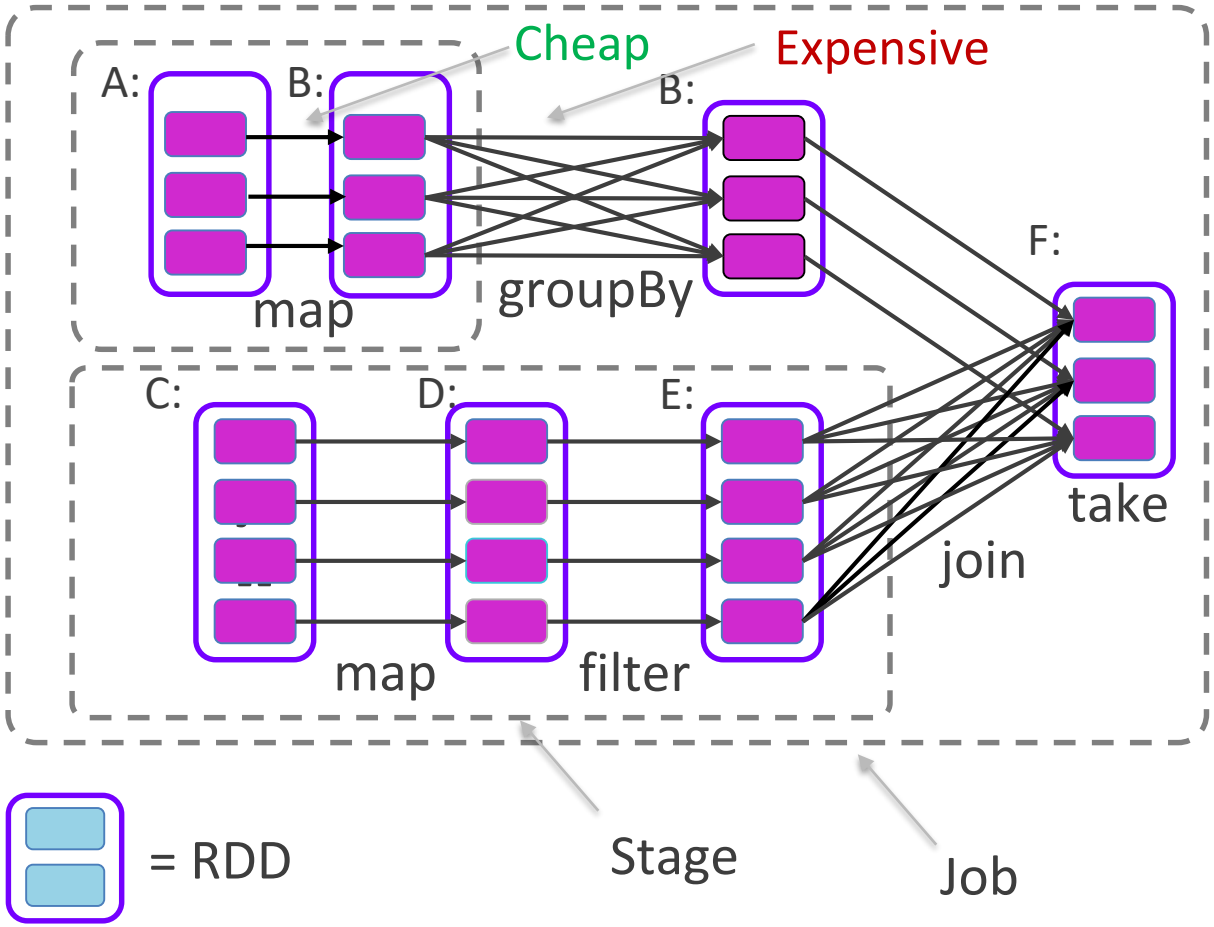
The driver program coordinates tasks and handles resource requests to the cluster manager. The driver distributes application code to each executor. Normally, each task takes 1 core.



Example: When Spark is run interactively via pyspark or spark-shell executors are assigned to the shell (driver) until the shell is exited. Each time the user invokes an action on the data the driver invokes that action on each executor as a task.

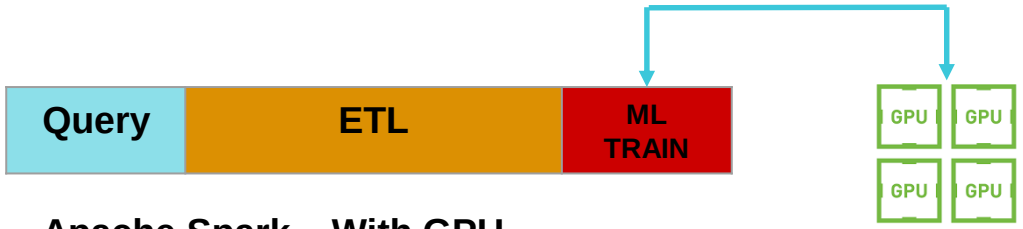
Another Quick Overview: Spark Execution

- Contrary to what some believe, Spark does not do everything in memory.
- Operations like groupBy, join, etc force a break in stages and shuffle data.
- Shuffles write data to disk.
- Shuffles can cause problems and also performance degradations
- Necessary Evil



Spark: Data Preparation Still on CPU's

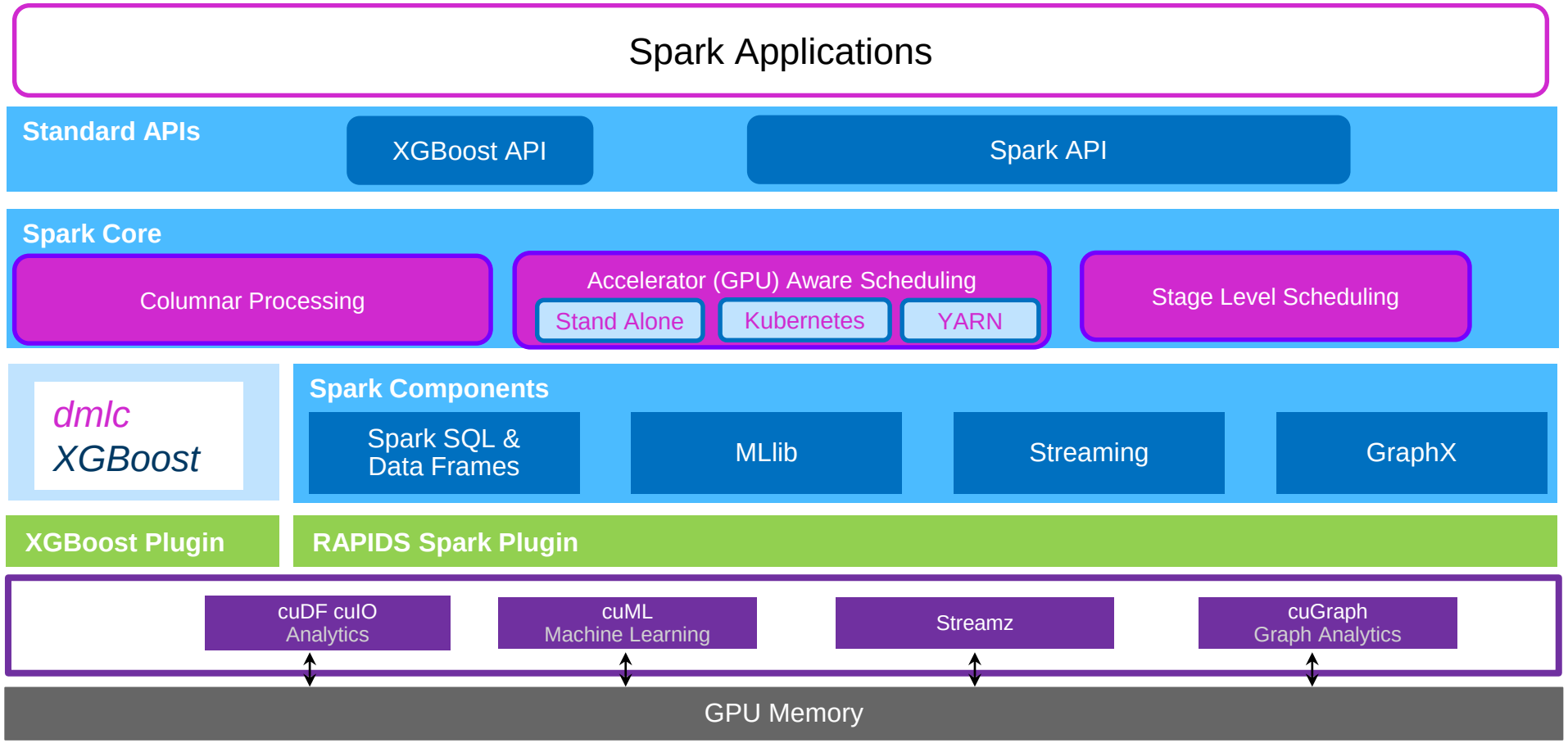
- **Apache Spark – With CPU Only Setup**



- **Apache Spark – With GPU**

Enterprise GPU users find it challenging to "**Feed the Beast**".

Spark+GPU: Accelerated Data Science at Scale

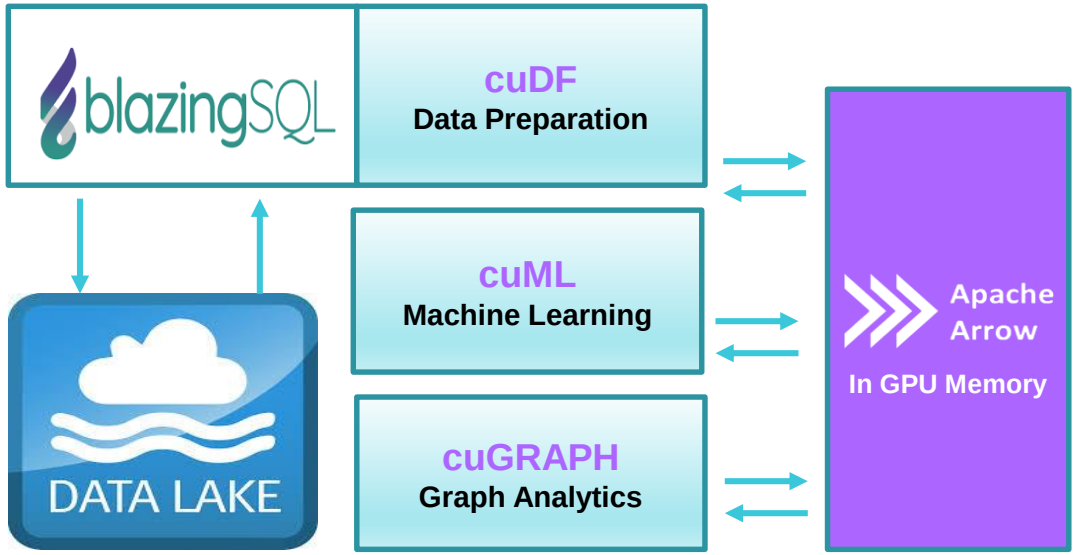


SQL Engines with RAPIDS

BlazingSQL

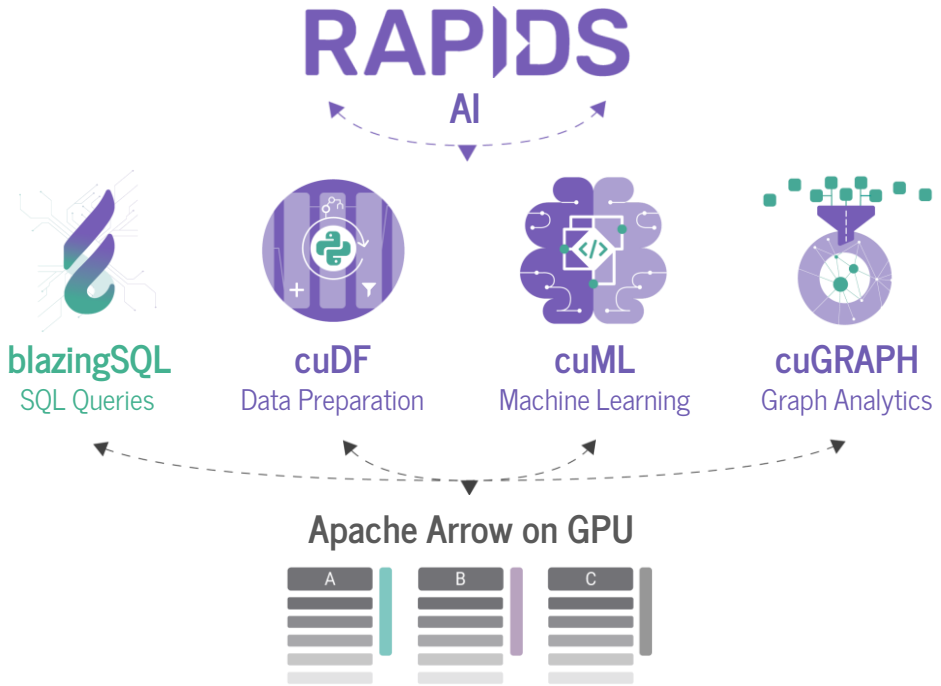
BLAZINGDB

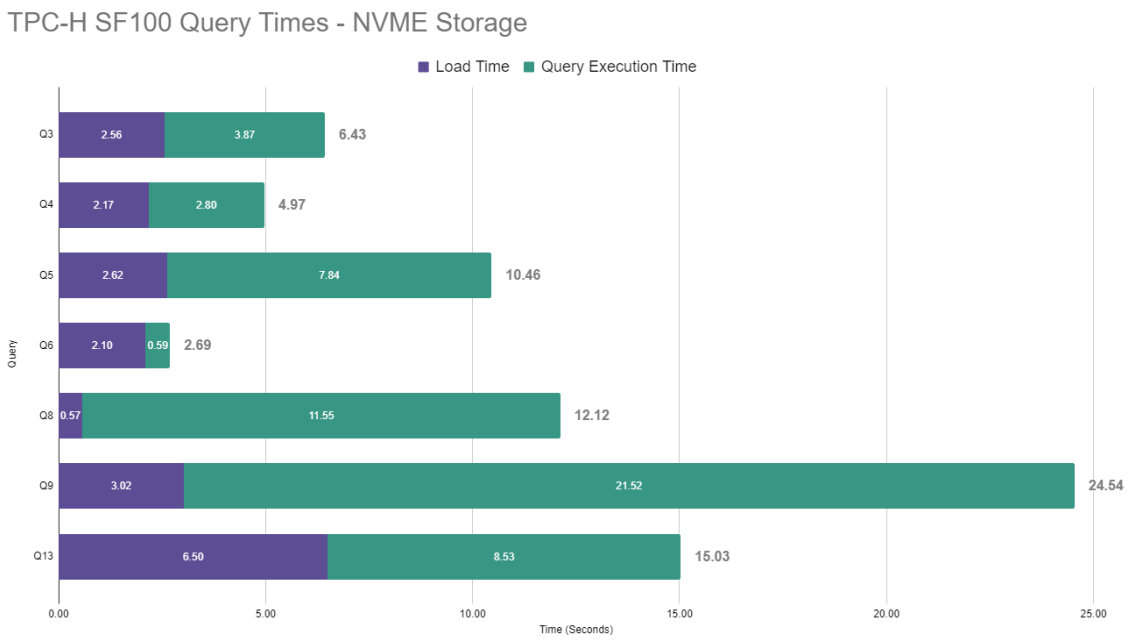
A SQL engine built on RAPIDS. Query enterprise data lakes lightning fast with full interoperability with RAPIDS stack.




```
from blazingsql import BlazingContext

bc = BlazingContext()

#Register Filesystem
bc.hdfs('data', host='129.13.0.12',
port=54310)

# Create Table
bc.create_table('performance',
file_type='parquet',
path='hdfs://data/performance/')

#Execute Query
result_gdf = bc.run_query('SELECT * FROM
performance WHERE
YEAR(maturity_date)>2005')

print(result_gdf)
```

BlazingSQL


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

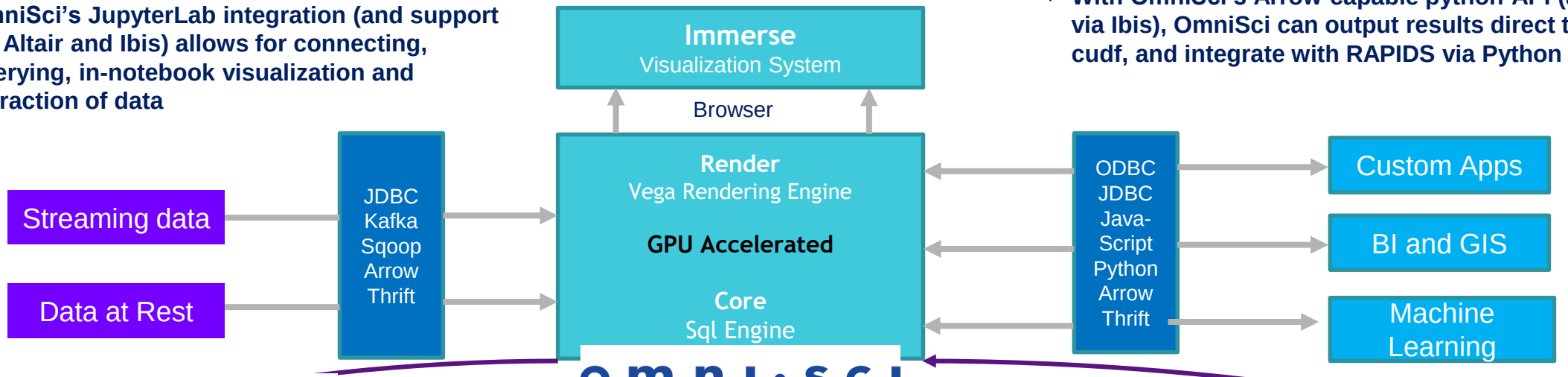

TPC-H SF100 Query Times - NVME Storage
Load Time
Query Execution Time
Query
Q3
2.56
3.87
6.43
Q4
2.17
2.80
4.97
Q5
2.62
7.84
10.46
Q6
2.10
0.59
2.69
Q8
0.57
11.55
12.12
Q9
3.02
21.52
24.54
Q13
6.50
8.53
15.03
0.00
5.00
10.00
15.00
20.00
25.00
Time (Seconds)

SQL Engines with RAPIDS

OmniSci

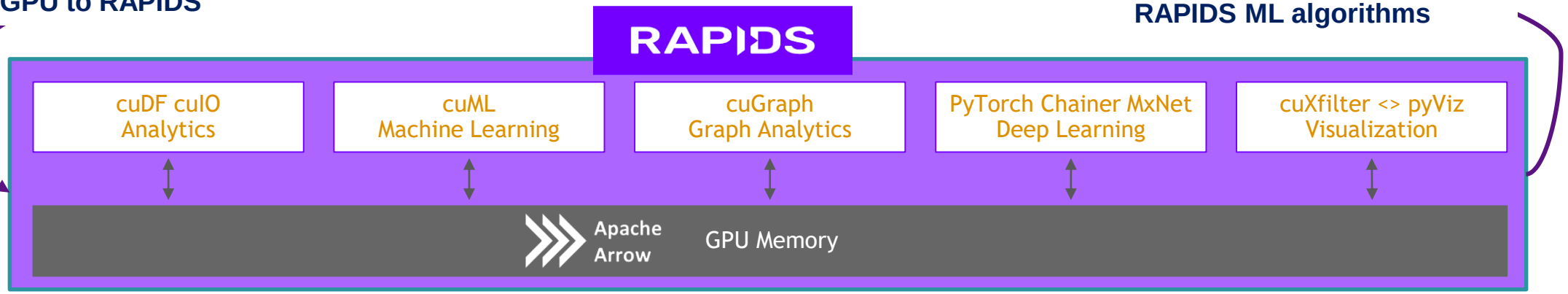
✓ **OmniSci's JupyterLab integration (and support for Altair and Ibis) allows for connecting, querying, in-notebook visualization and extraction of data**

✓ **With OmniSci's Arrow-capable python API (and via Ibis), OmniSci can output results direct to cudf, and integrate with RAPIDS via Python**



OmniSci query result set in-GPU to RAPIDS

GPU-resident outputs from RAPIDS ML algorithms



Community

Ecosystem Partners

CONTRIBUTORS

ADOPTERS

OPEN SOURCE

Building on top of RAPIDS

A bigger, better, stronger ecosystem for all

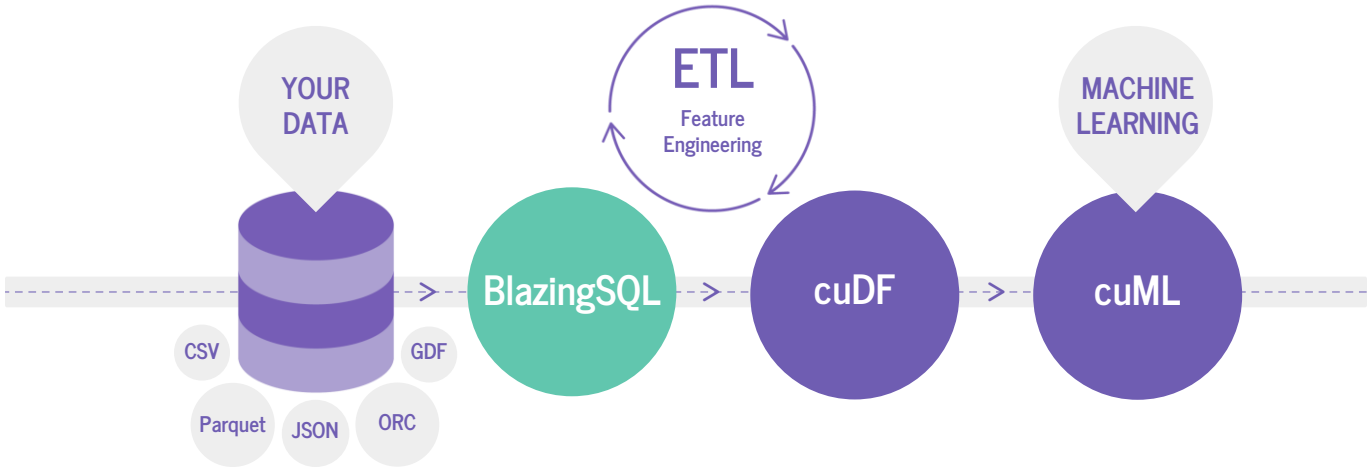
Streamz

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

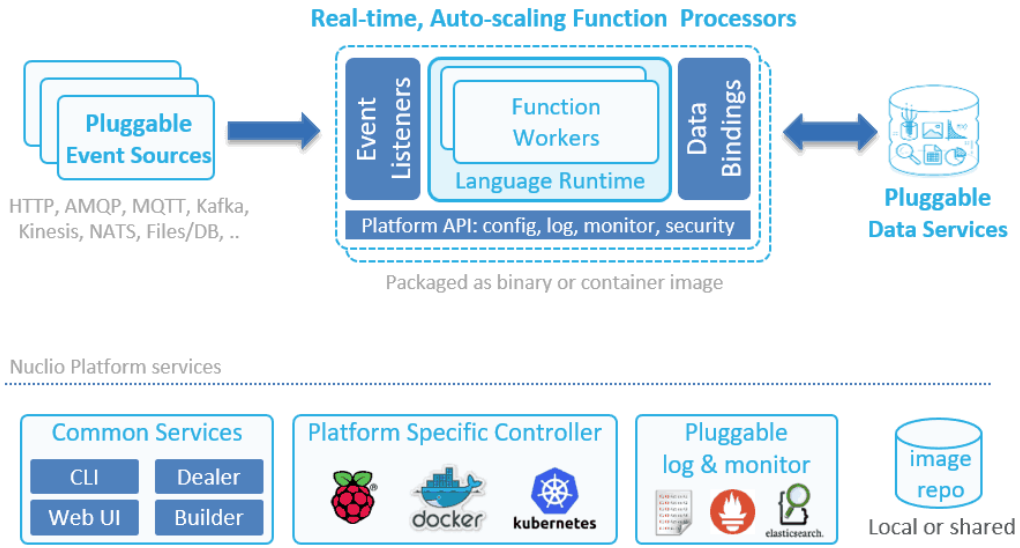
BlazingSQL




```
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

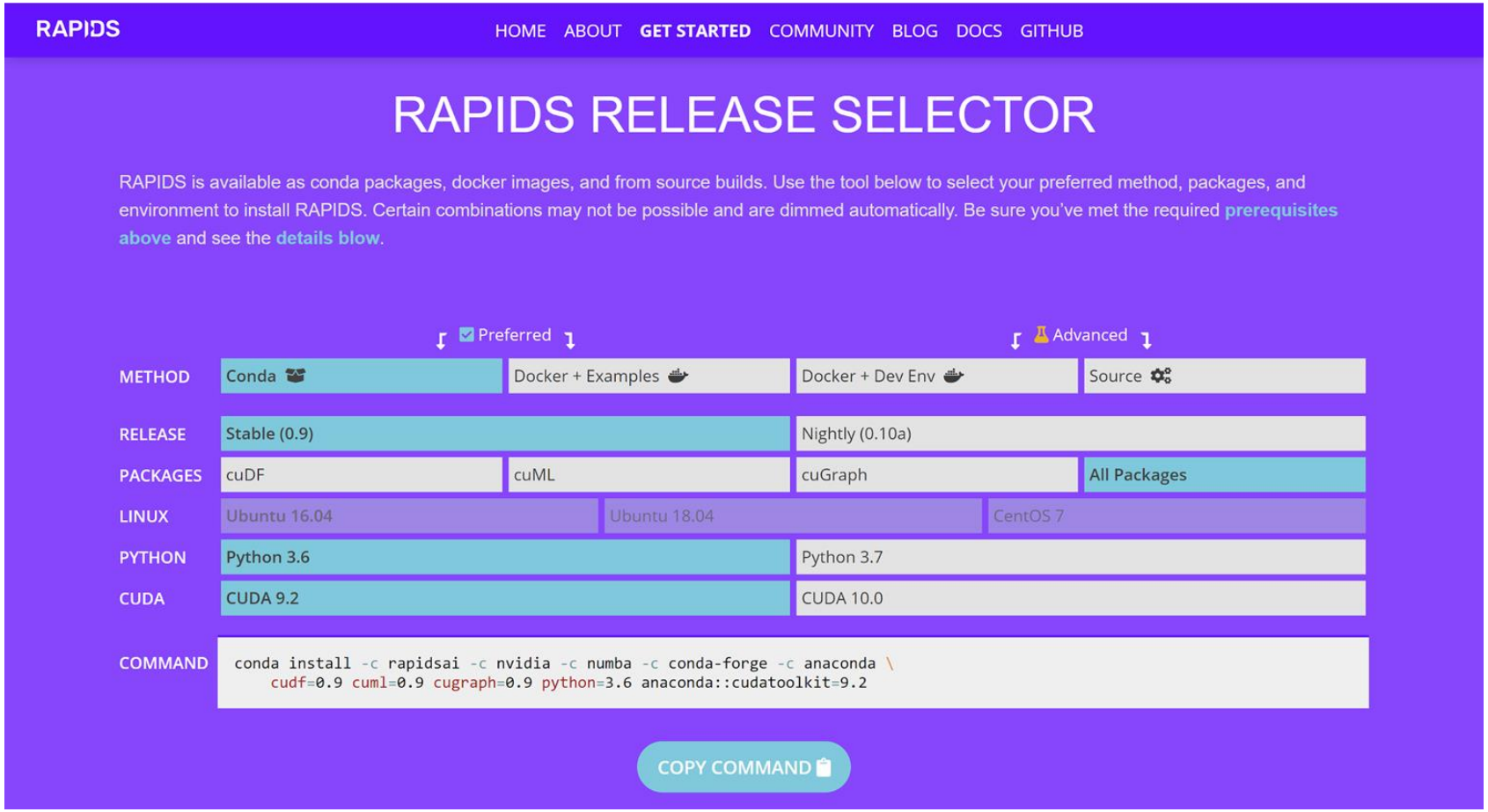
Integration with major cloud providers
Both containers and cloud specific machine instances
Support for Enterprise and HPC Orchestration Layers

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

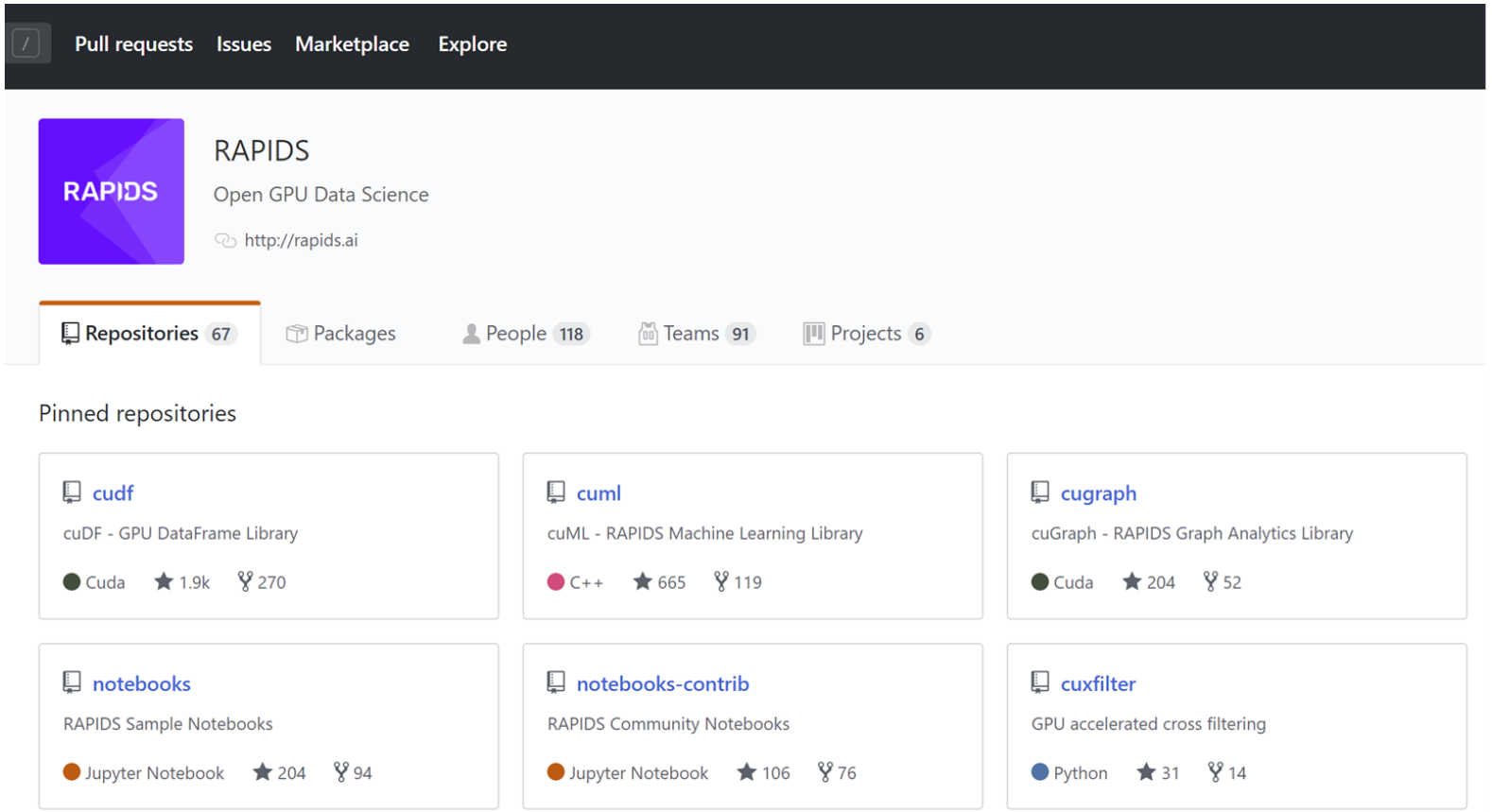
Easy Installation

Interactive Installation Guide



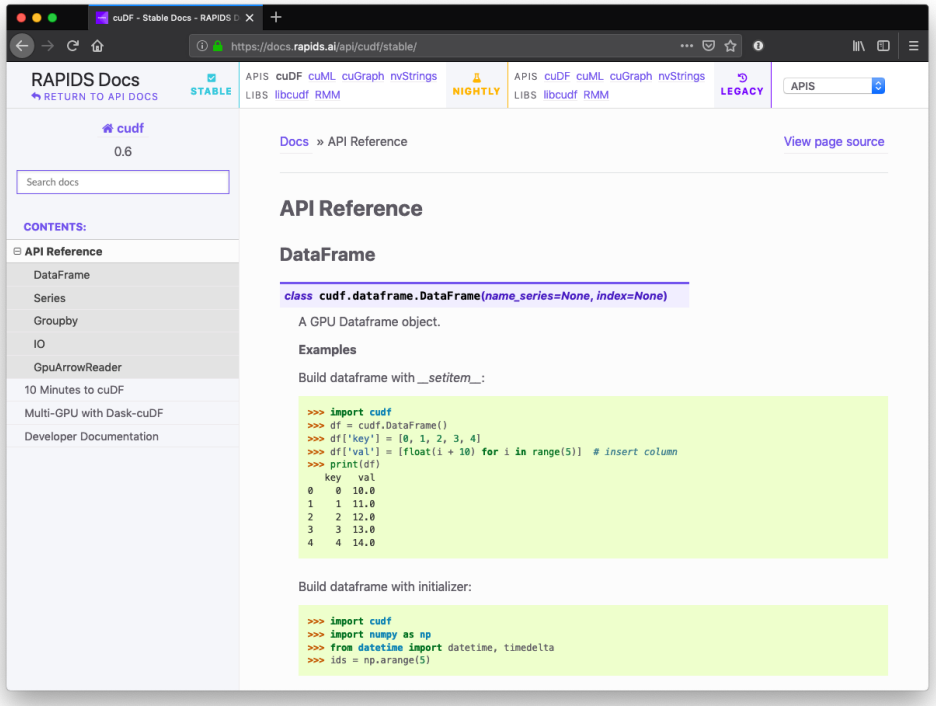
Explore: RAPIDS Github

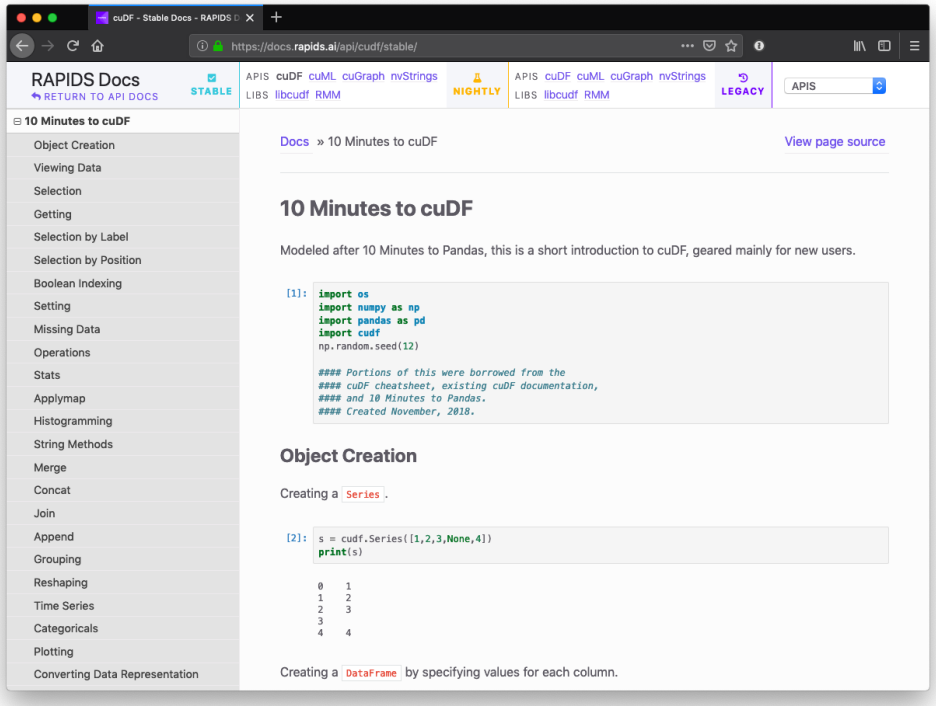
https://github.com/rapidsai



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.

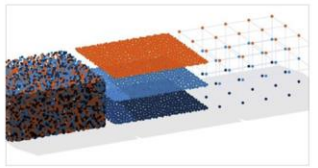
See how RAPIDS was used to place 17th in the Banco Santander Kaggle Competition

Jiwei Liu
Jul 3 · 5 min read

NVIDIA GPUs and Apache Spark, One Step Closer

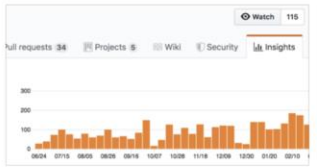
RAPIDS XGBoost4J-Spark Package Now Available

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

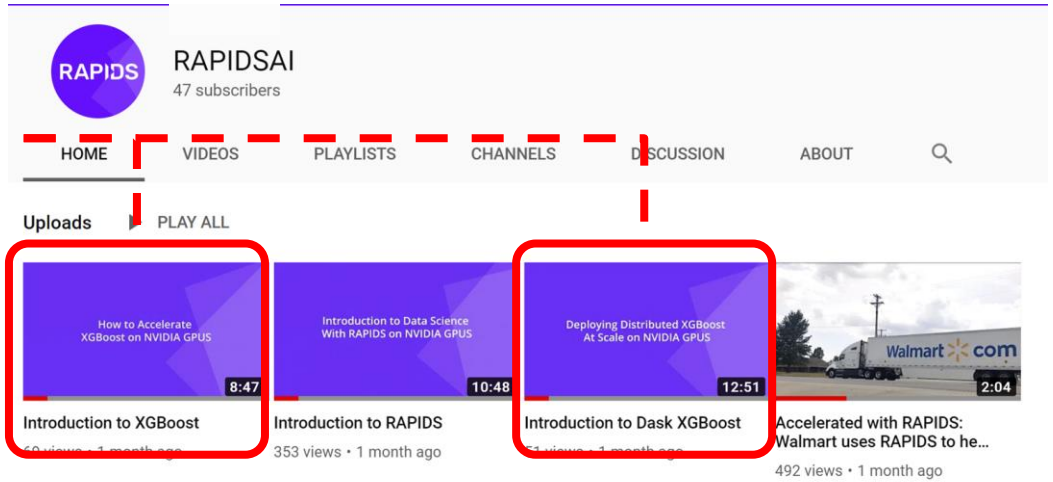
Explore: Notebooks Contrib

Notebooks Contrib Repo has tutorials and examples, and various E2E demos. RAPIDS Youtube channel has 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

Google Groups

Docker Hub

Slack Channel

Stack Overflow

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

cugraph
cuGraph
Cuda · Apache-2.0 · 46 · 172 · 58 (1 issue needs help) · 4 · Updated 1 hour ago

TECH BLOG Walmart Labs ENGINEERING DATA SCIENCE INFOSEC UX DESIGN LEADERSHIP | ABOUT

How GPU Computing literally saved me at work?

Python+GPU = Power, 2 Days to 20 seconds

Abhishek Mungoli Follow
May 9 · 9 min read

John Murray
@MurrayData
Follow

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 #RAPIDSAI 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
```

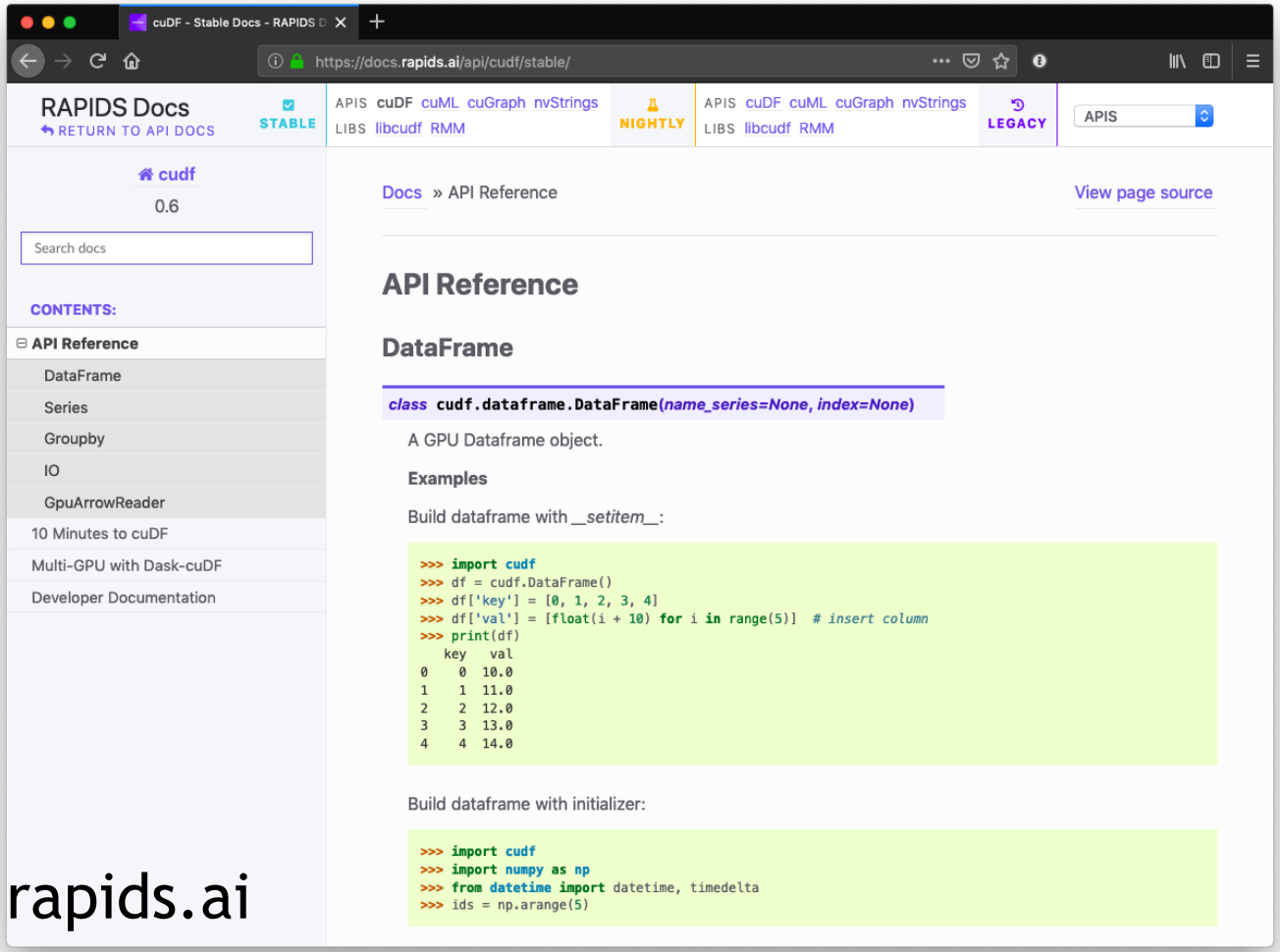
etting Started with cuDF (RAPIDS)

Darren Ramsook Follow
Jun 9 · 3 min read

Getting Started

RAPIDS Docs

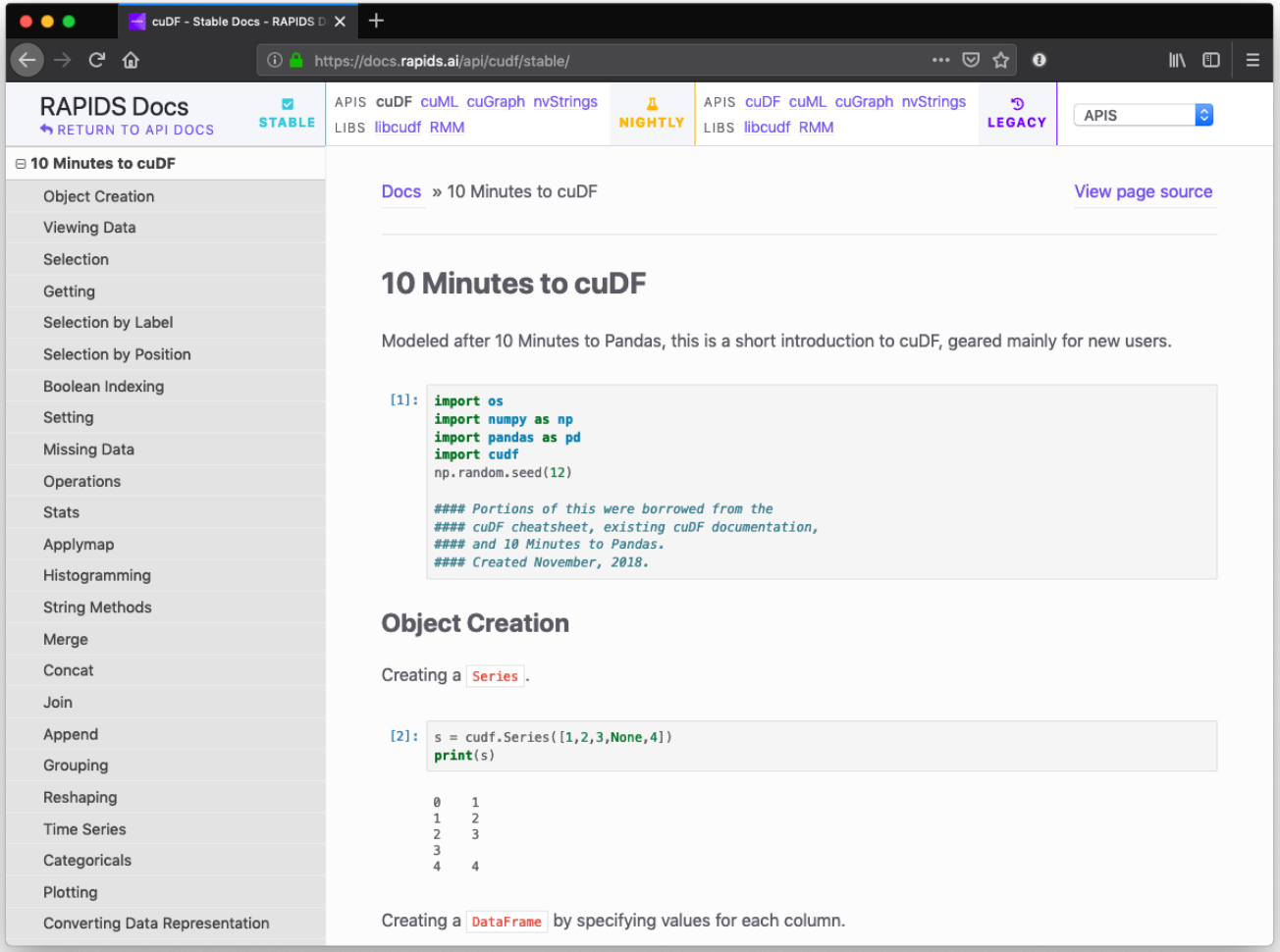
New, improved, and easier to use



https://docs.rapids.ai

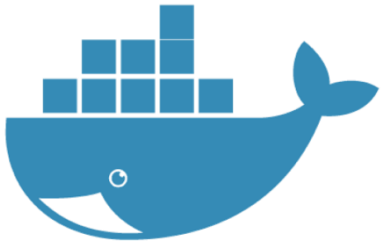
RAPIDS Docs

Easier than ever to get started with cuDF



RAPIDS

How do I get the software?

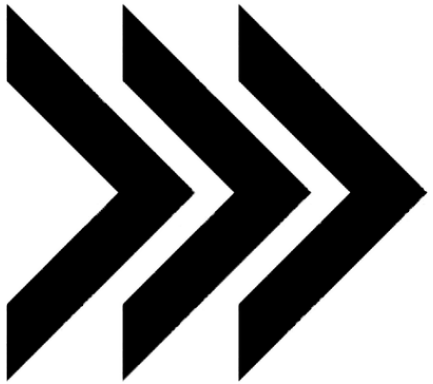


- https://github.com/rapidsai
- https://anaconda.org/rapidsai/

- 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

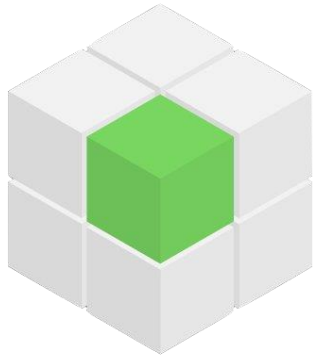
https://rapids.ai

@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

RAPIDS