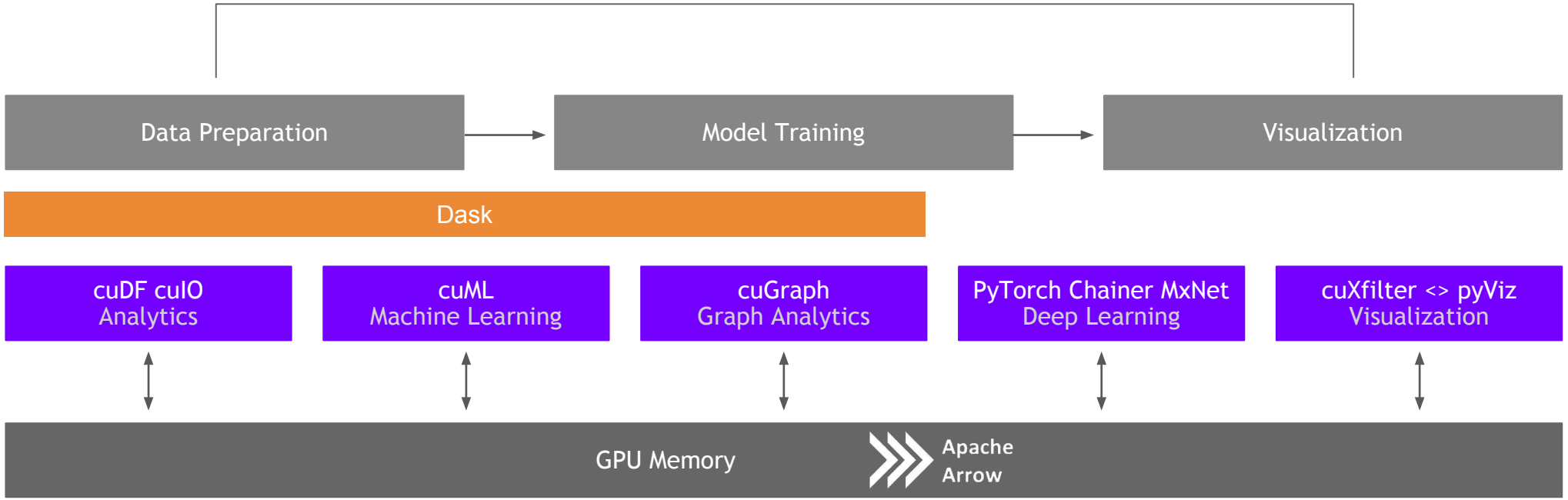
RAPIDS

The Platform Inside and Out

Joshua Patterson – GM, Data Science

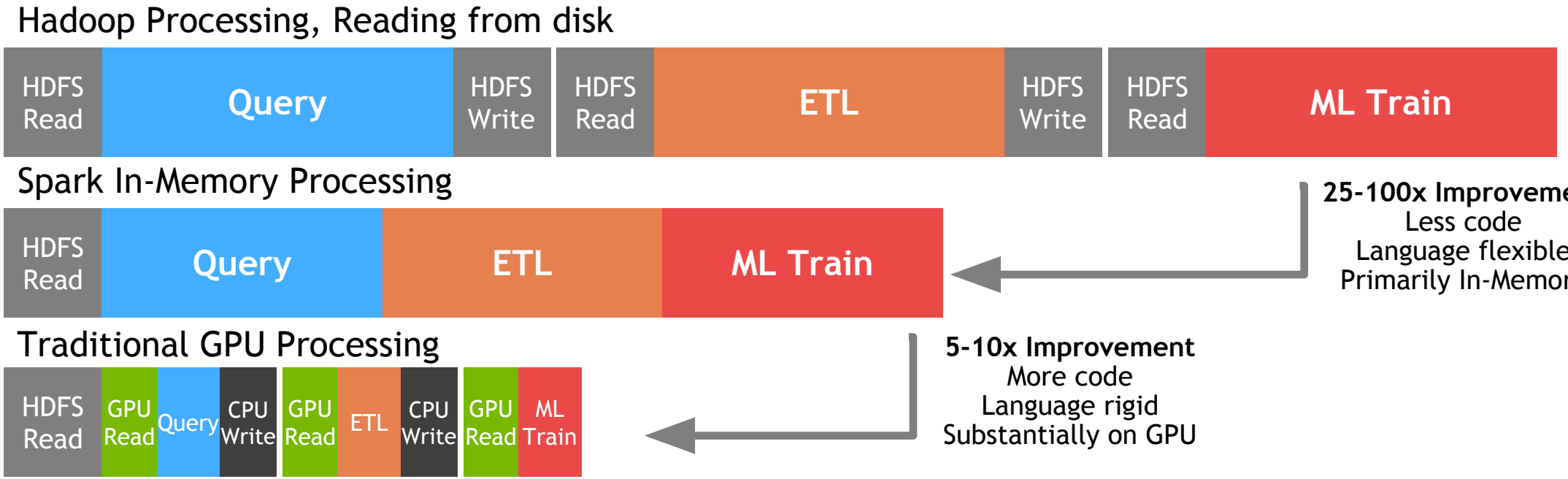
RAPIDS

End-to-End Accelerated GPU Data Science



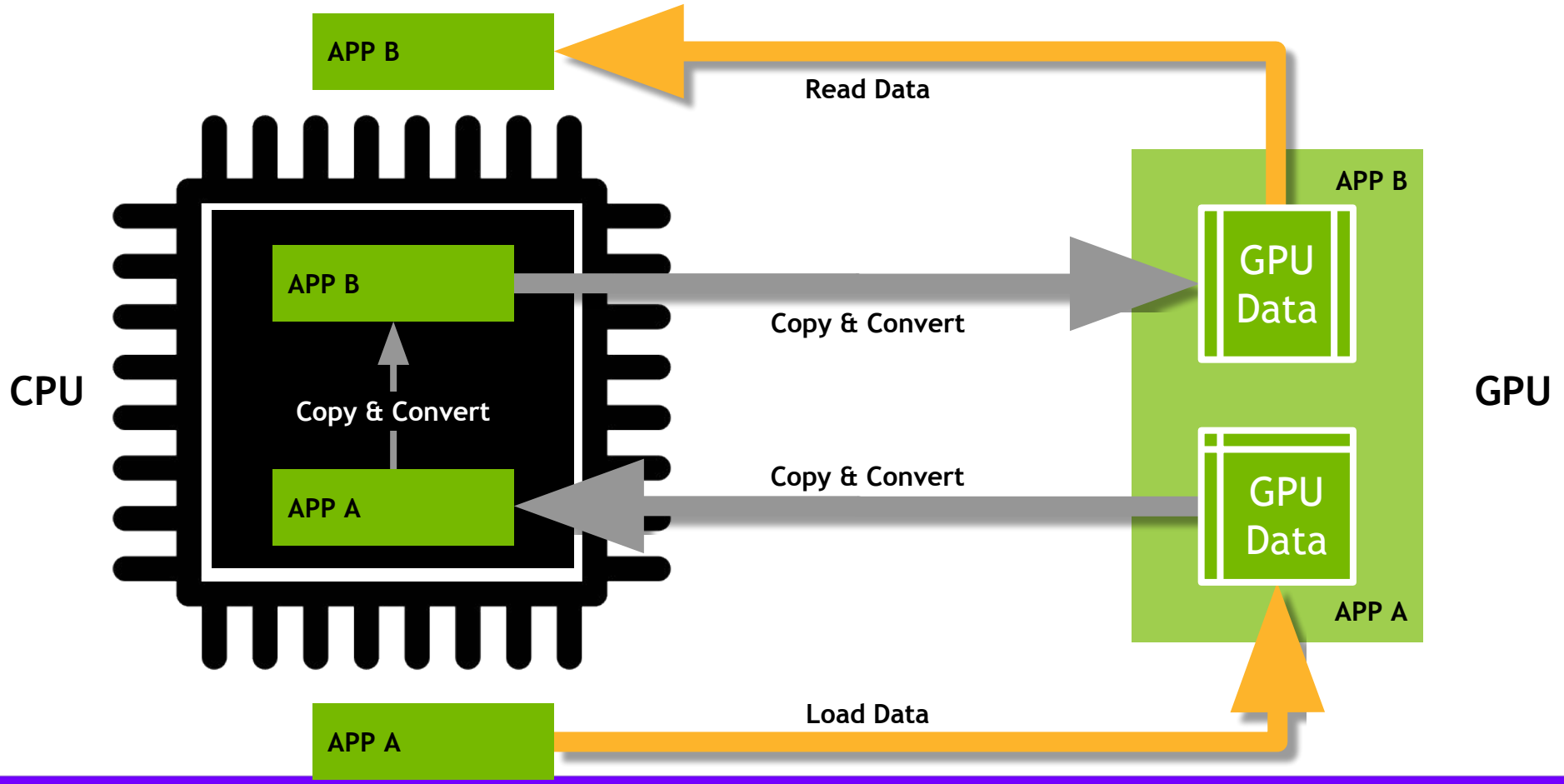
Data Processing Evolution

Faster data access, less data movement



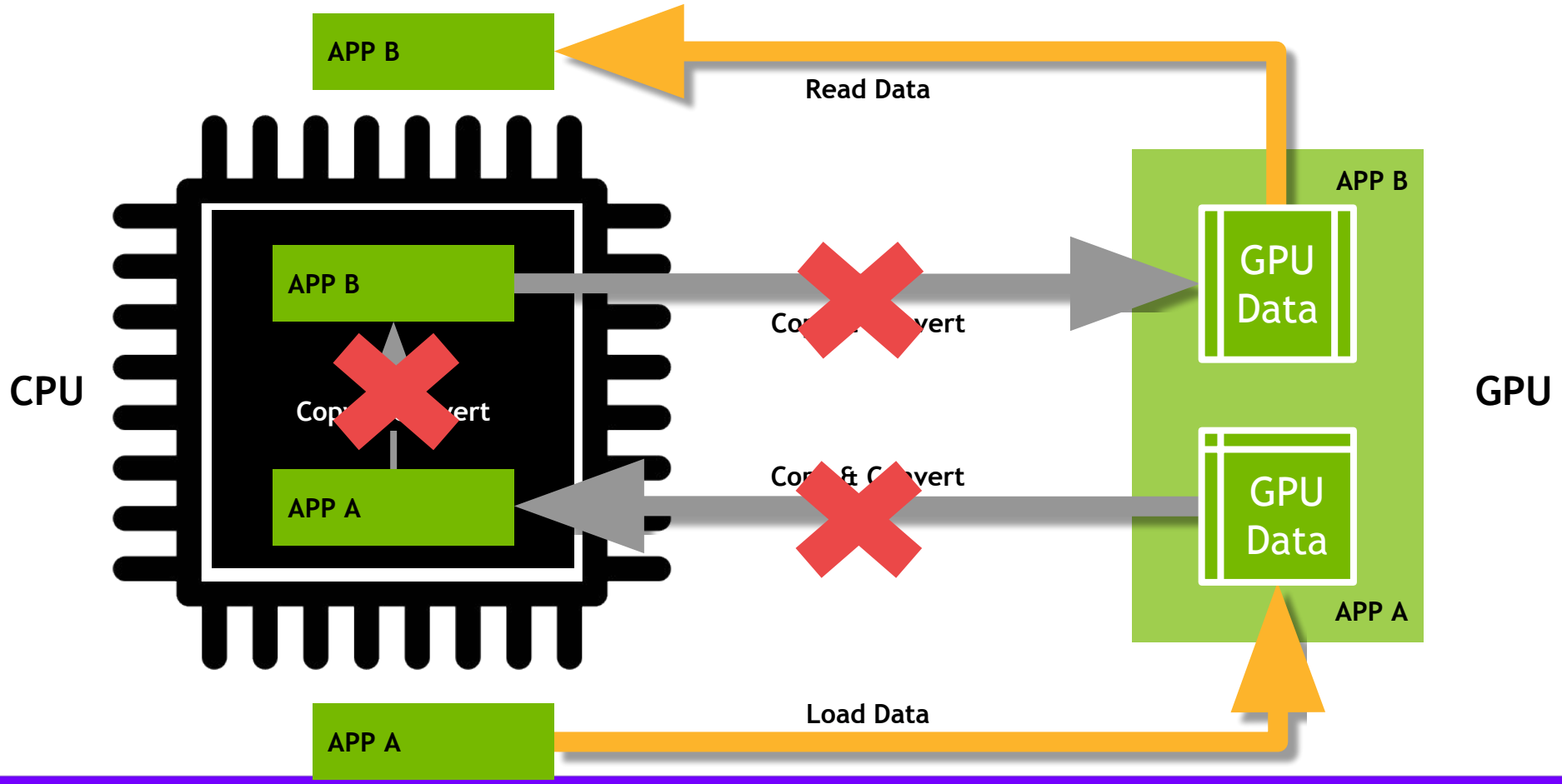
Data Movement and Transformation

The bane of productivity and performance

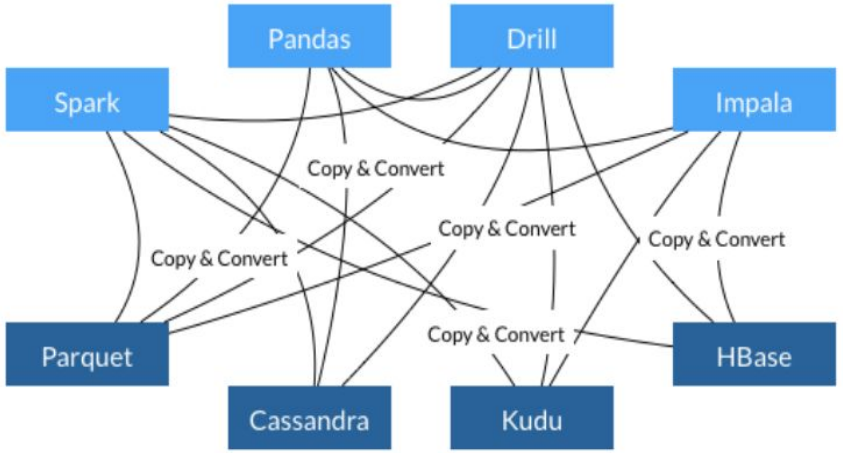


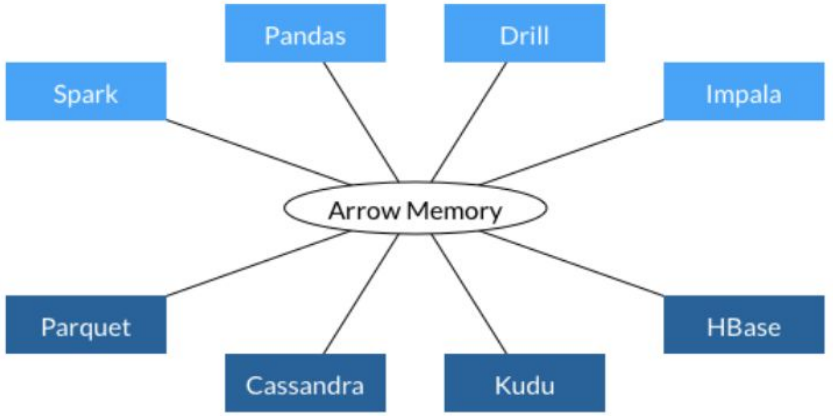
Data Movement and Transformation

What if we could keep data on the GPU?



Learning from Apache Arrow





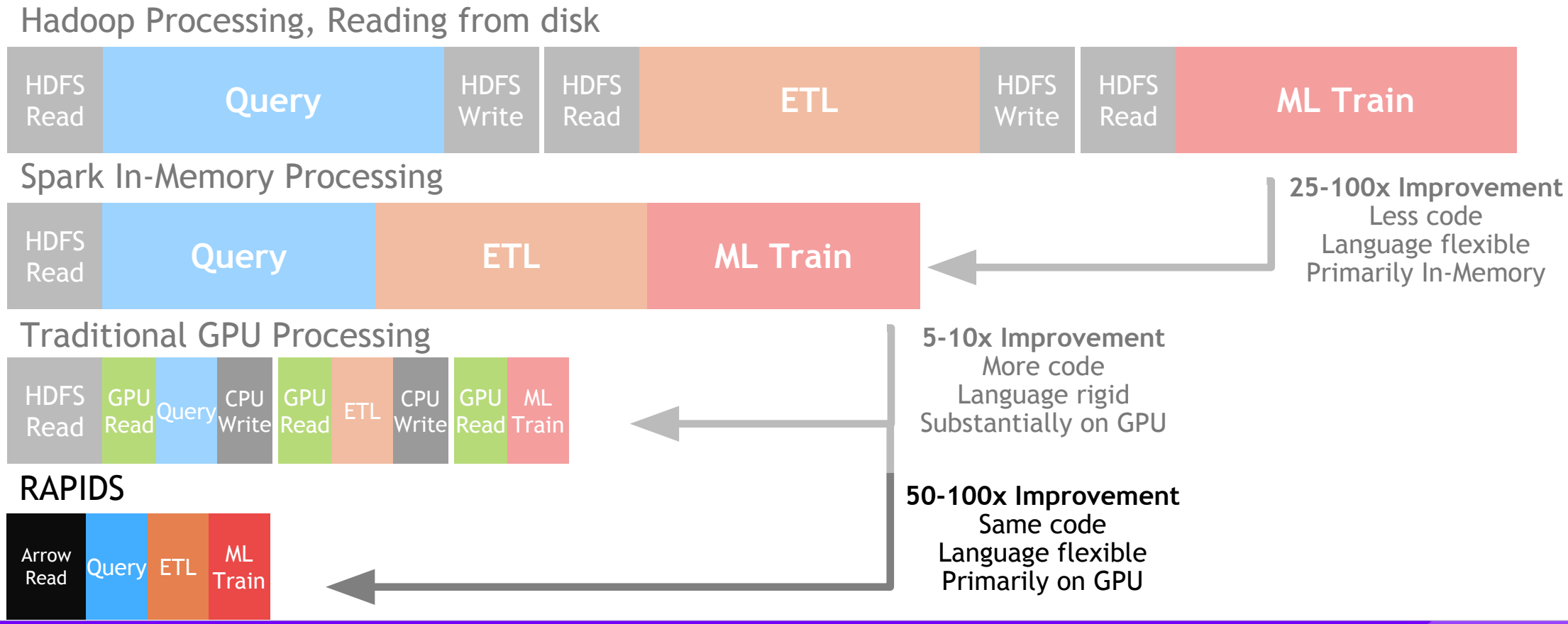
- Each system has its own internal memory format
- 70-80% computation wasted on serialization and deserialization
- Similar functionality implemented in multiple projects

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

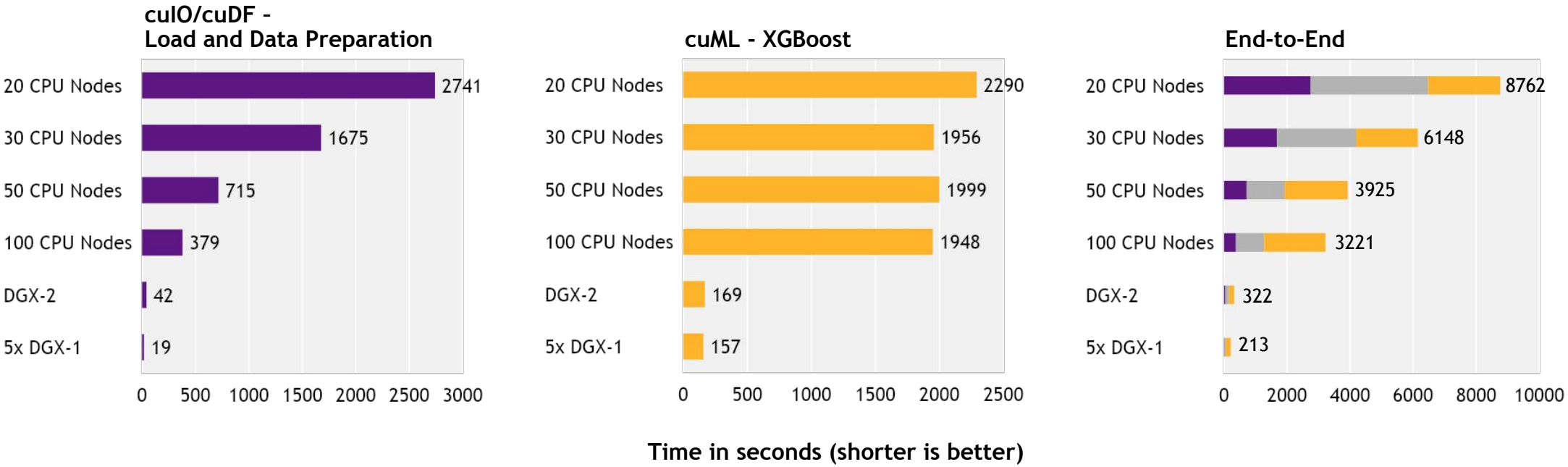
From Apache Arrow Home Page - https://arrow.apache.org/

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

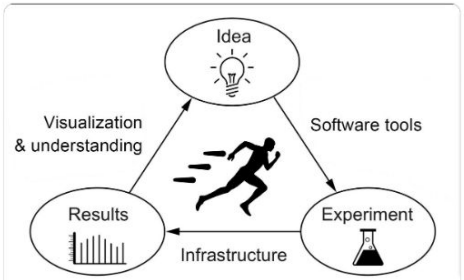
Speed, UX, and Iteration

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

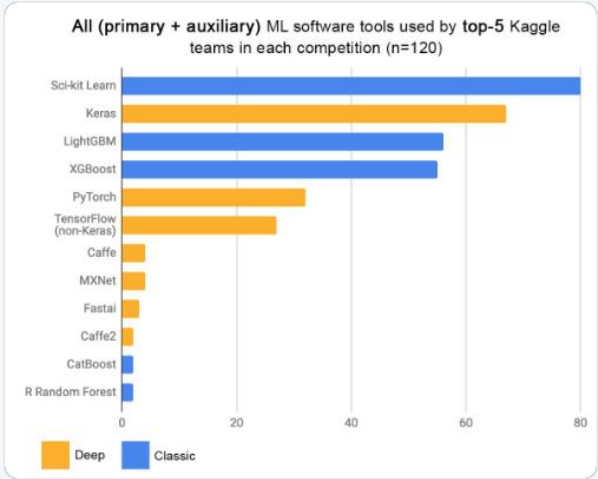
This is the fundamental believe that drives @RAPIDSai. @nvidia #GPU infrastructure is fast, people need to iterate quickly, people want a known #python interface. Combine them and you're off to the races!

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).



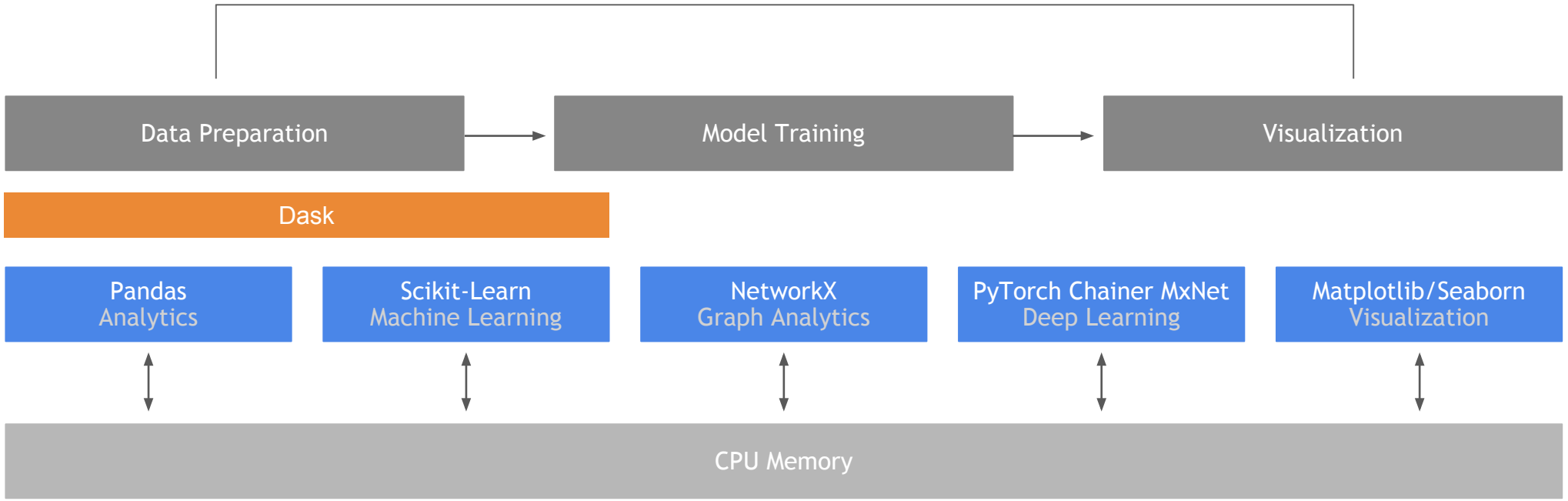
4 44 129

kaggle

RAPIDS Core

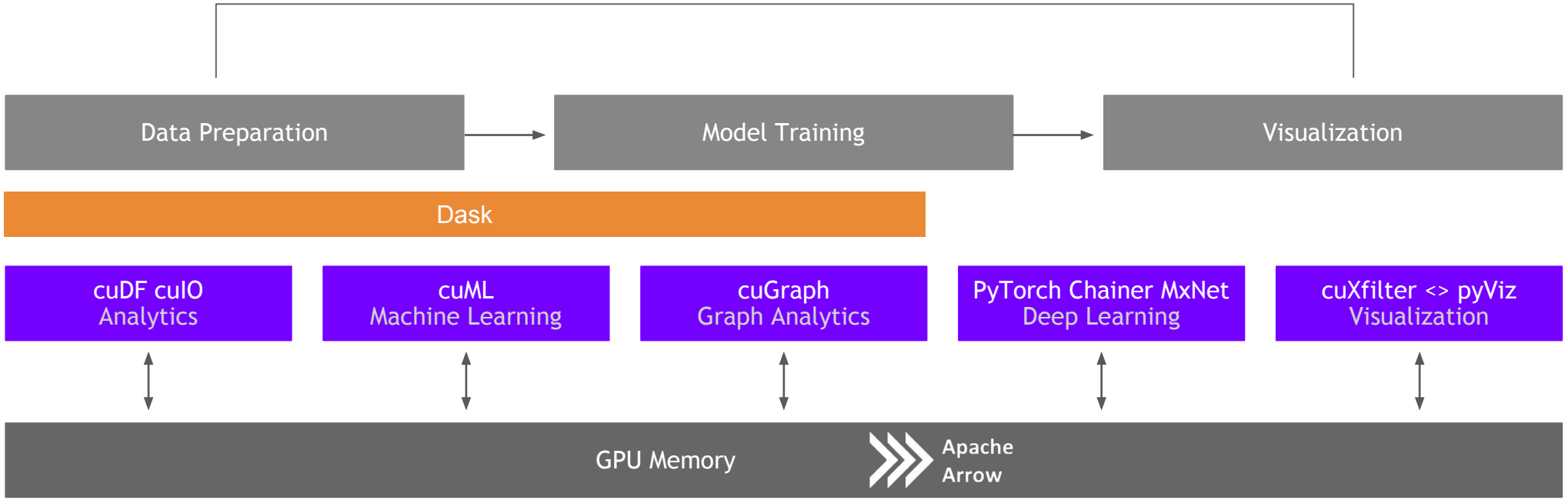
Open Source Data Science Ecosystem

Familiar Python APIs



RAPIDS

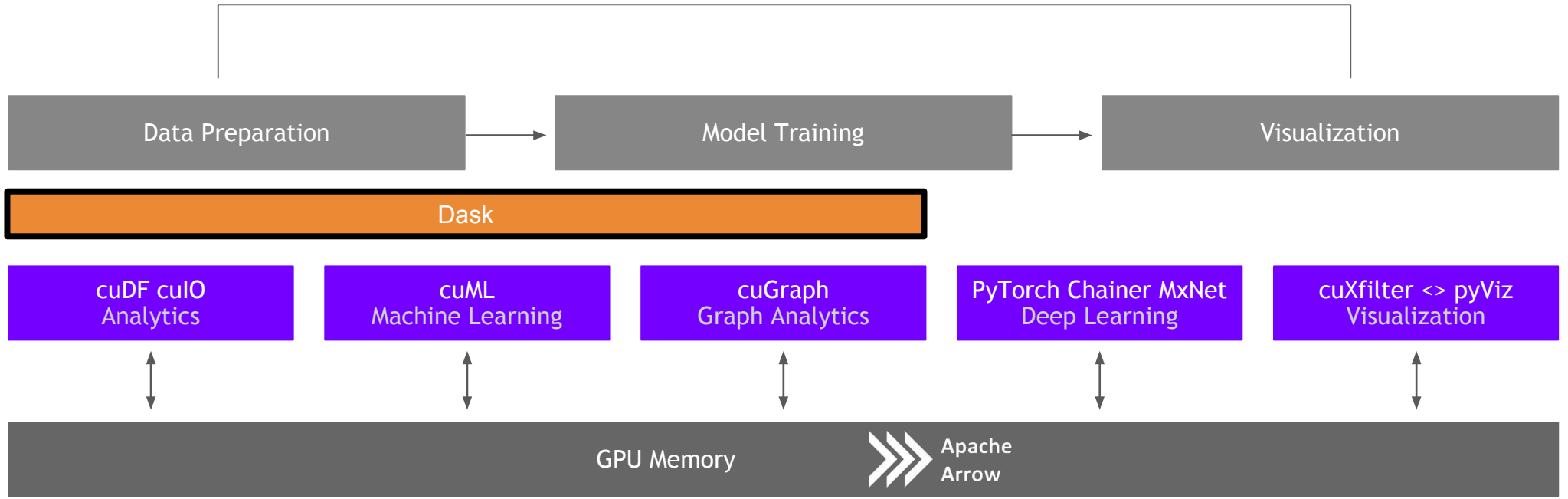
End-to-End Accelerated GPU Data Science



Dask

RAPIDS

Scaling RAPIDS with Dask



Why Dask?

- **PyData Native**
  - Built on top of NumPy, Pandas Scikit-Learn, etc. (easy to migrate)
  - With the same APIs (easy to train)
  - With the same developer community (well trusted)
- **Scales**
  - Easy to install and use on a laptop
  - Scales out to thousand-node clusters
- **Popular**
  - Most common parallelism framework today at PyData and SciPy conferences
- **Deployable**
  - HPC: SLURM, PBS, LSF, SGE
  - Cloud: Kubernetes
  - Hadoop/Spark: Yarn

Why OpenUCX?

Bringing hardware accelerated communications to Dask

- TCP sockets are slow!
- UCX provides uniform access to transports (TCP, InfiniBand, shared memory, NVLink)
- Python bindings for UCX (ucx-py) in the works
- 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

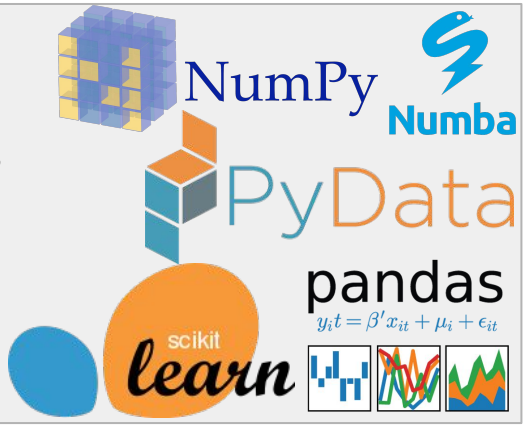
NumPy -> CuPy/PyTorch/..
Pandas -> cuDF
Scikit-Learn -> cuML
Numba -> Numba

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

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

$y_i t = \beta' x_{it} + \mu_i + \epsilon_{it}$

scikit learn

Dask

Multi-core and Distributed PyData

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

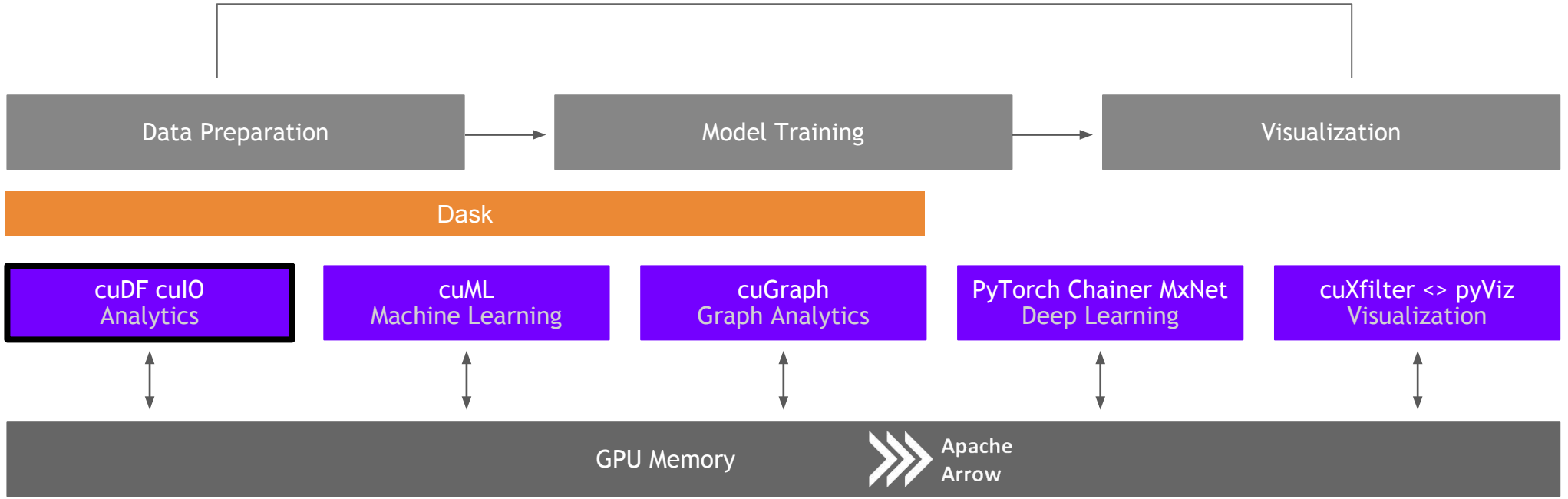
DASK

Scale out / Parallelize

cuDF

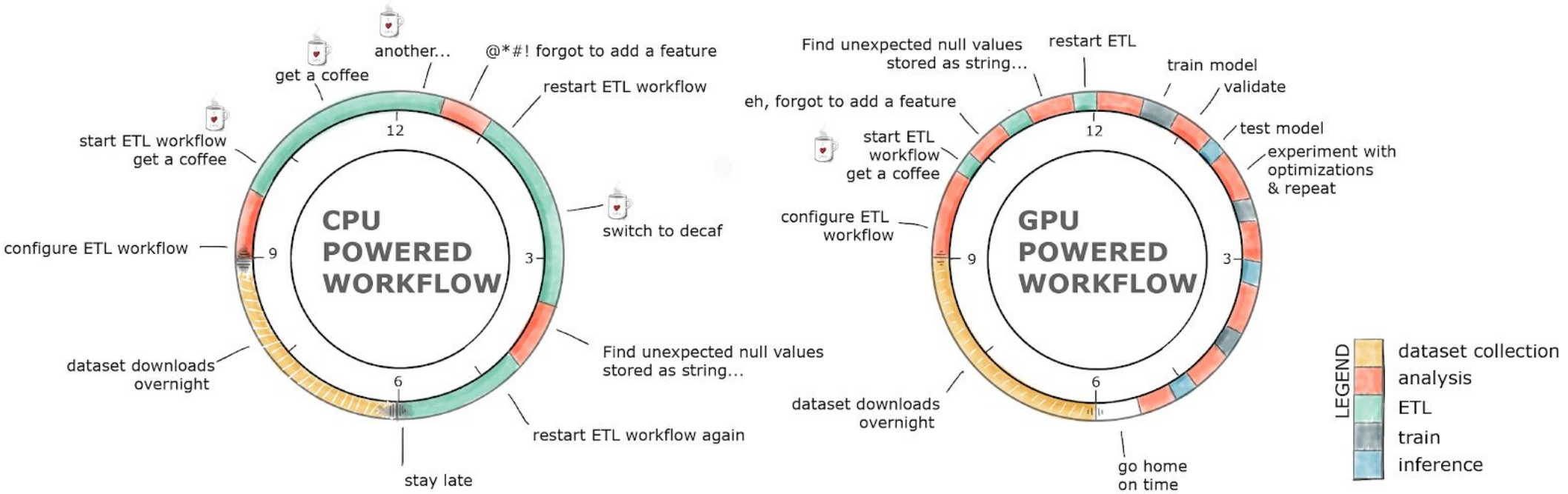
RAPIDS

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



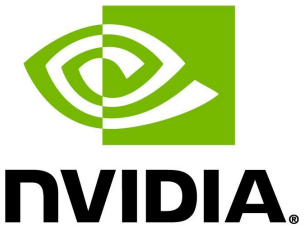
ETL - the Backbone of Data Science

libcuDF is…

CUDA C++ Library

- Low level library containing function implementations and C/C++ API
- Importing/exporting Apache Arrow in GPU memory using CUDA IPC
- CUDA kernels to perform element-wise math operations on GPU DataFrame columns
- CUDA sort, join, groupby, reduction, etc. operations on GPU DataFrames

```
void some_function( cudf::column const* input,
                    cudf::column * output,
                    args...)
{
    // Do something with input
    // Produce output
}
```



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

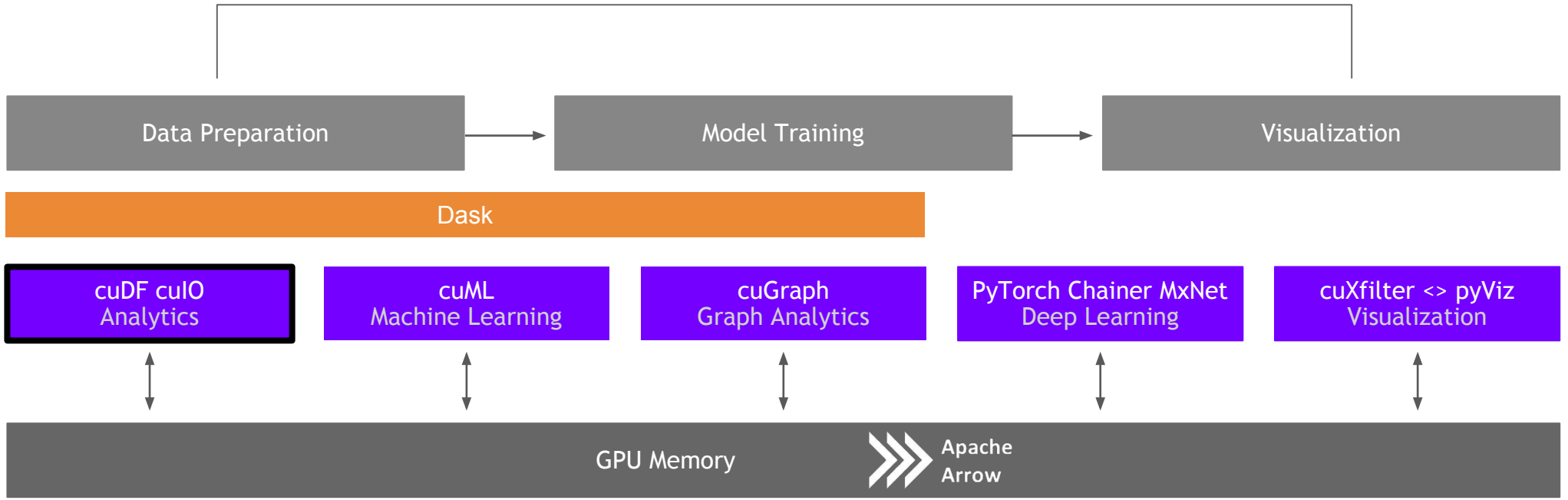
```
In [6]: #grabbing the first character of the years in city string to get rid of plus sign, and converting
          to int
        gdf['city_years'] = gdf.Stay_In_Current_City_Years.str.get(0).stoi()

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

ETL - the Backbone of Data Science

cuDF is not the end of the story



ETL - the Backbone of Data Science

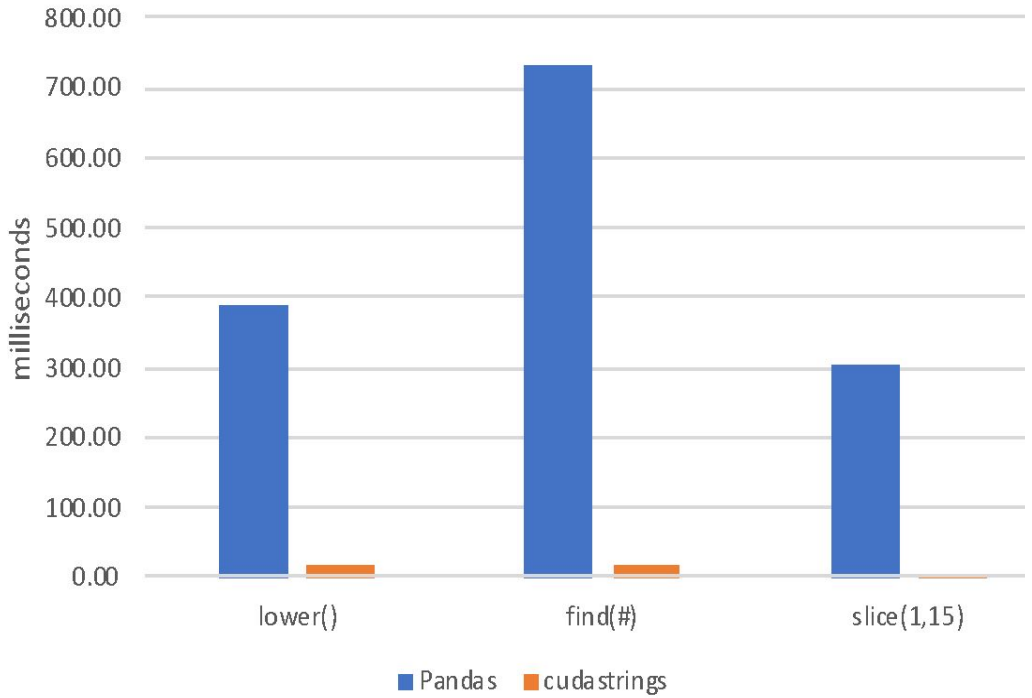
String Support

Now v0.8 String Support:

- Regular Expressions
- Element-wise operations
  - Split, Find, Extract, Cat, Typecasting, etc…
- String GroupBys, Joins

Future v0.9+ String Support:

- Combining cuStrings into libcudf
- Extensive performance optimization
- More Pandas String API compatibility
- Improved Categorical column support



Extraction is the Cornerstone of ETL

cuIO is born

- Follows the APIs of Pandas and provide >10x speedup

- CSV Reader - v0.2, CSV Writer v0.8
- Parquet Reader – v0.7
- ORC Reader – v0.7
- JSON Reader - v0.8
- Avro Reader - v0.9
- HDF5 Reader - v0.10

- Key is GPU-accelerating both parsing and decompression wherever possible

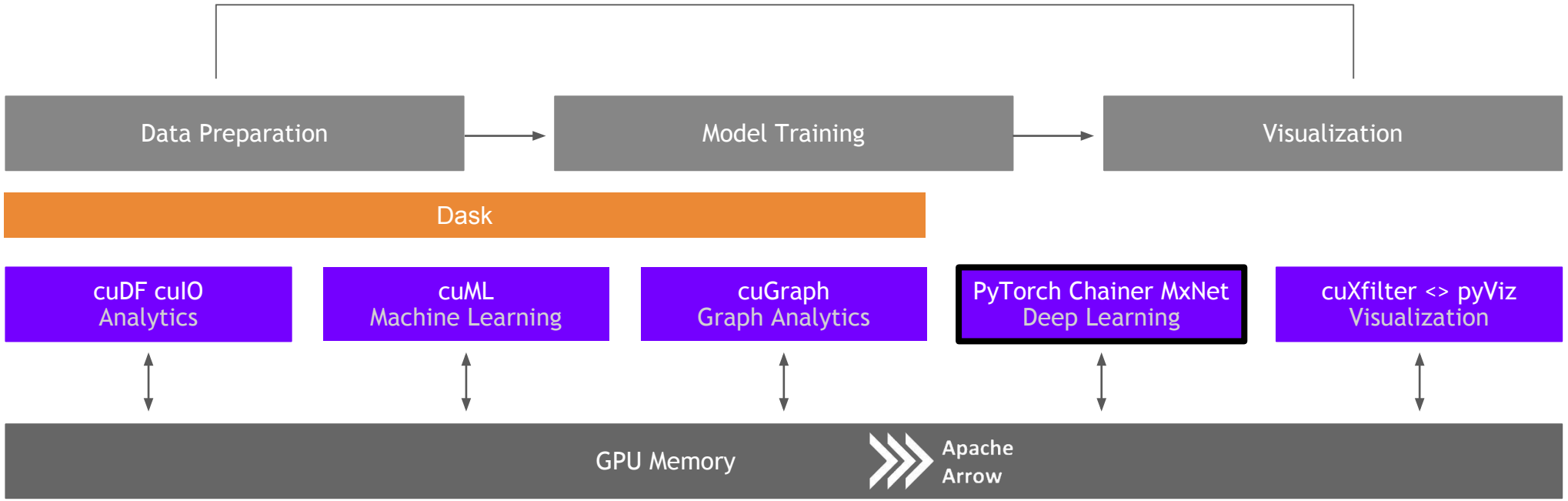
```
1]: import pandas, cudf

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

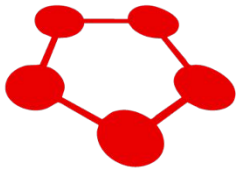
Building bridges into the array ecosystem



Interoperability for the Win

DLPack and __cuda_array_interface__

mpi4py

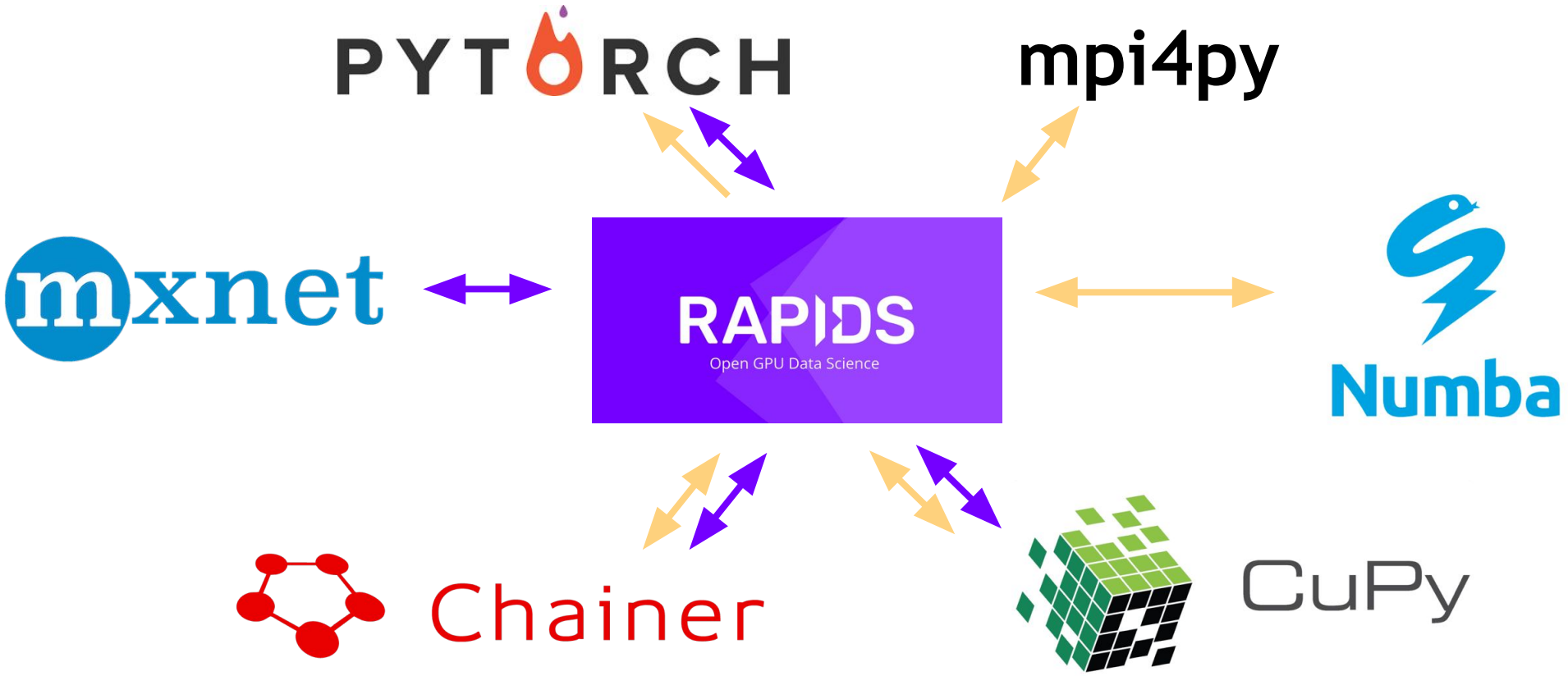


Chainer

CuPy

Interoperability for the Win

DLPack and __cuda_array_interface__



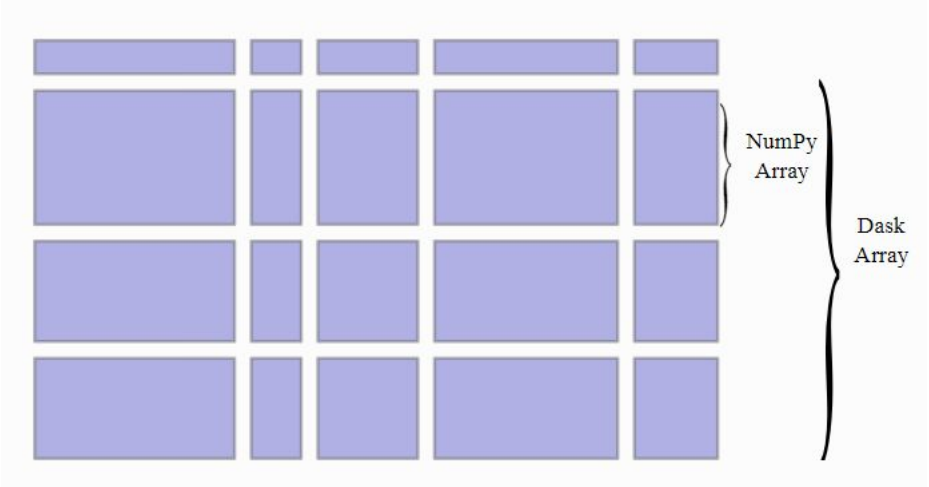
ETL – Arrays and DataFrames

Dask and CUDA Python arrays

Chainer

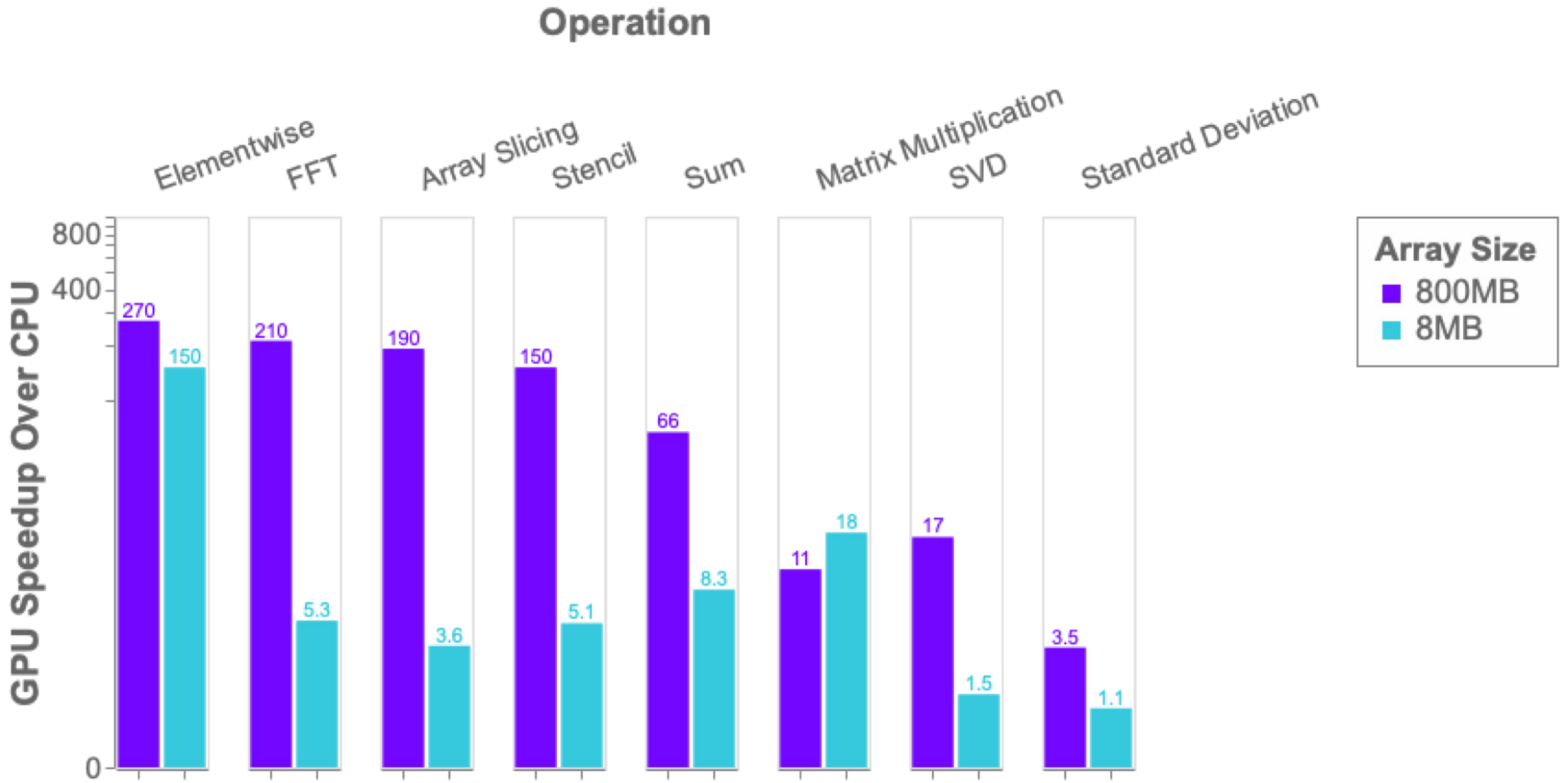
CuPy

Numba



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

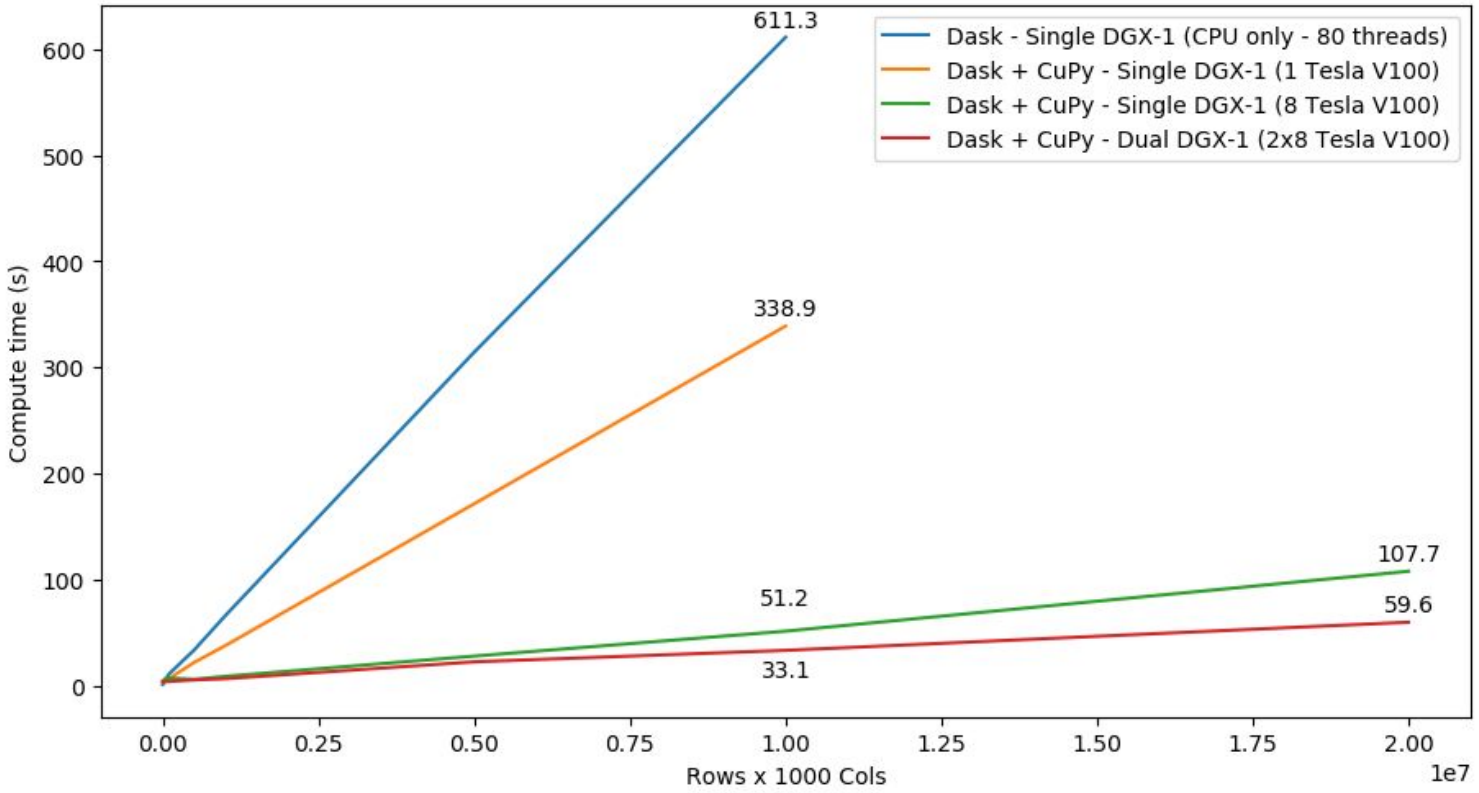
Benchmark: single-GPU CuPy vs NumPy



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

SVD Benchmark

Dask and CuPy Doing Complex Workflows



Also…Achievement Unlocked:

Petabyte Scale Data Analytics with Dask and CuPy

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

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

3.2 PETABYTES IN LESS THAN 1 HOUR

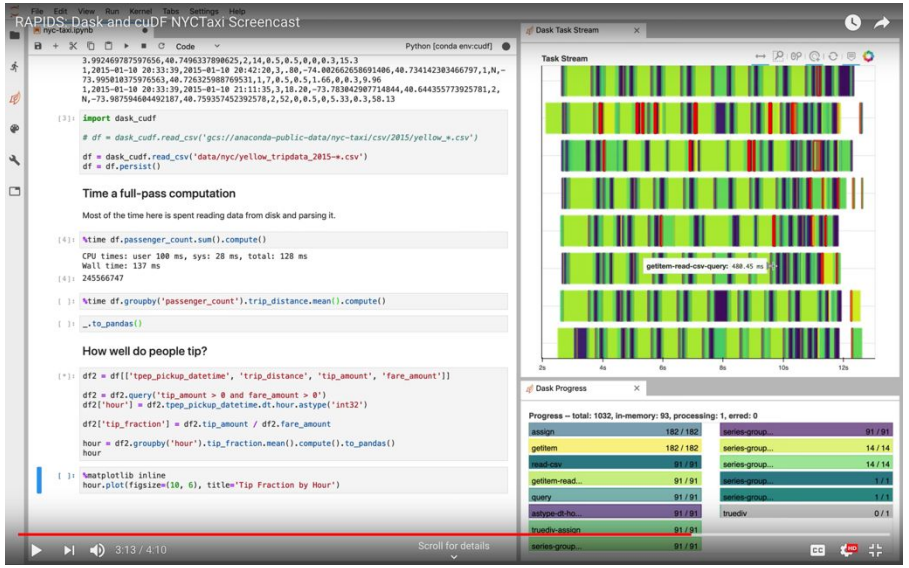
Distributed GPU array | parallel reduction | using 76x GPUs

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

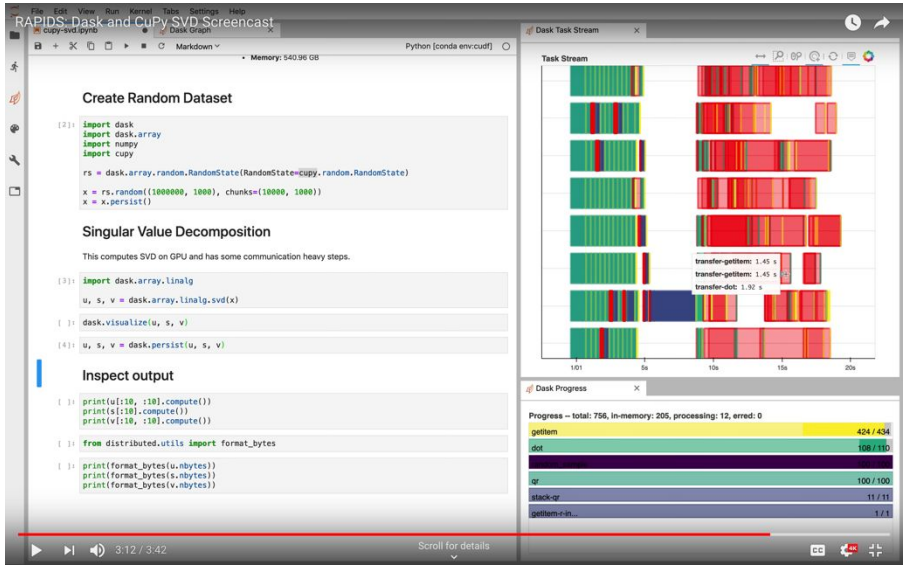
Cluster configuration: 20x GCP instances, each instance has:
CPU: 1 VM socket (Intel Xeon CPU @ 2.30GHz), 2-core, 2 threads/core, 132GB mem, GbE ethernet, 950 GB disk
GPU: 4x NVIDIA Tesla P100-16GB-PCIe (total GPU DRAM across nodes 1.22 TB)
Software: Ubuntu 18.04, RAPIDS 0.5.1, Dask=1.1.1, Dask-Distributed=1.1.1,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

ETL – Arrays and DataFrames

More Dask Awesomeness from RAPIDS



https://youtu.be/gV0cykgsTPM

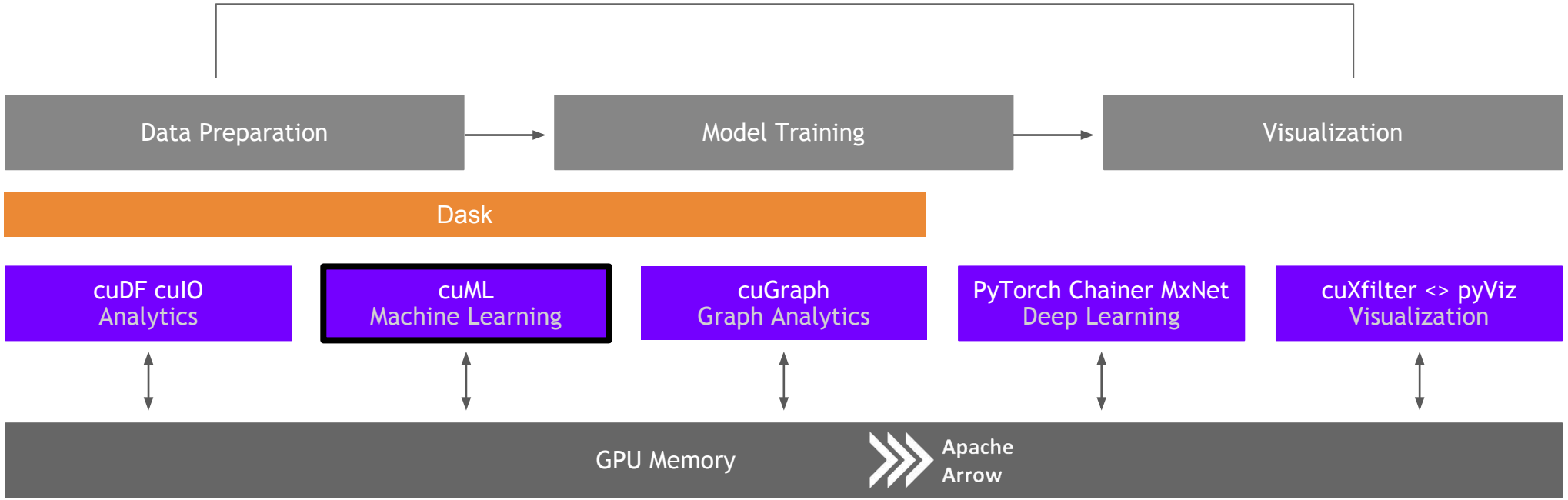


https://youtu.be/R5CiXti_MWo

cuML

Machine Learning

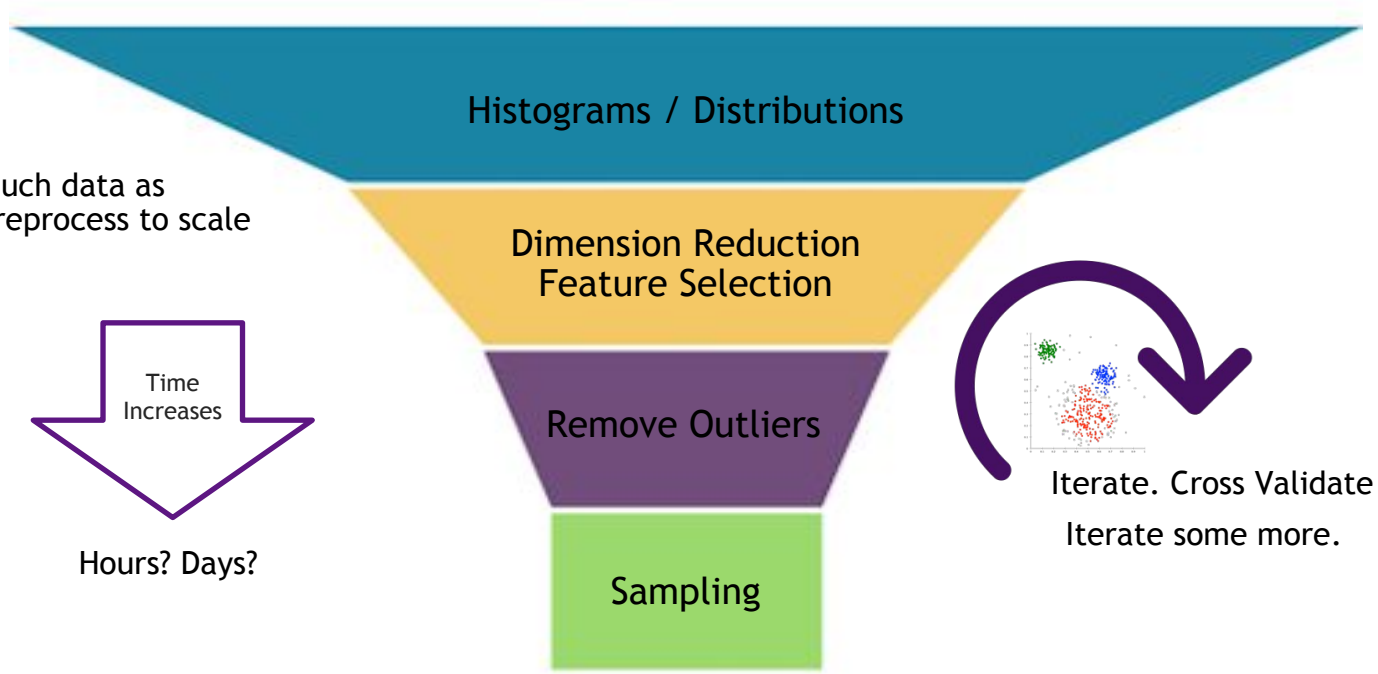
More models more problems



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

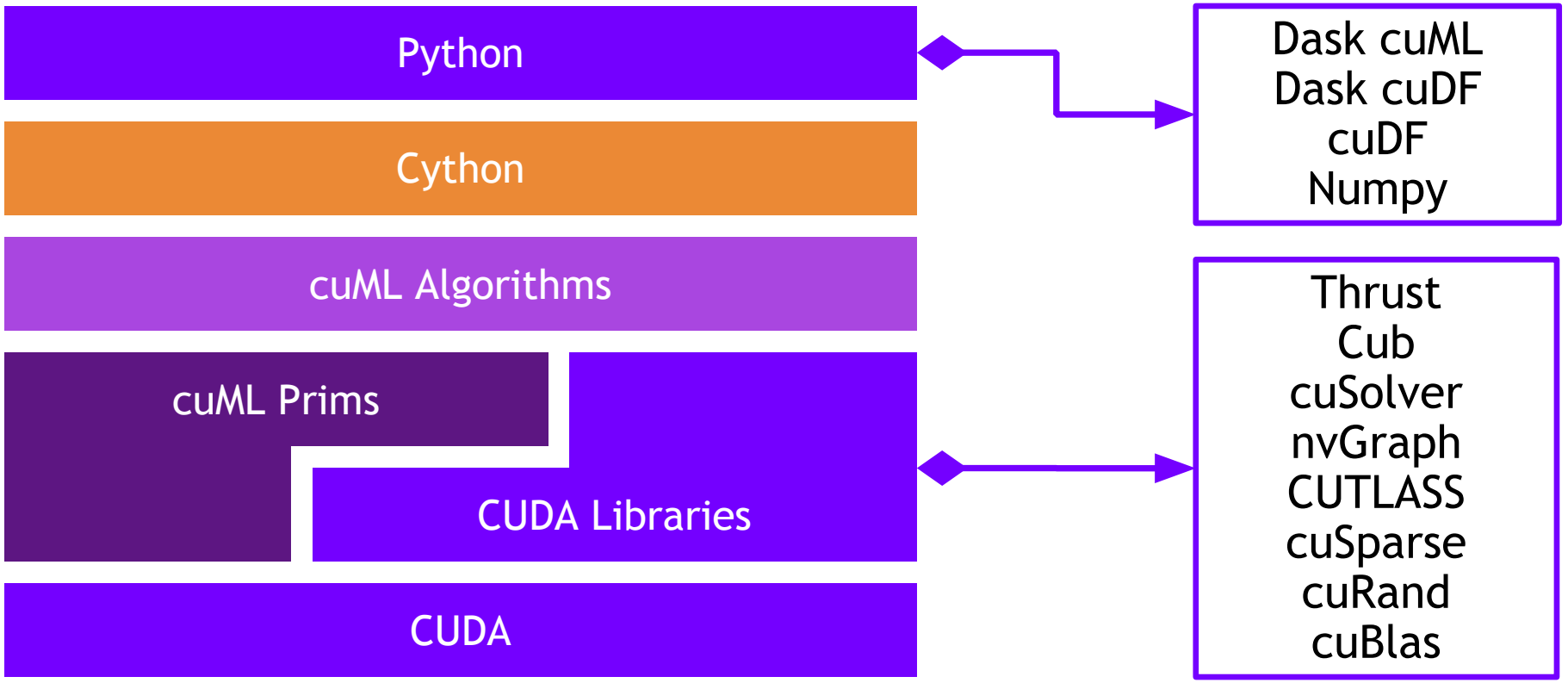
Hours? Days?

Iterate. Cross Validate & Grid Search.

Iterate some more.

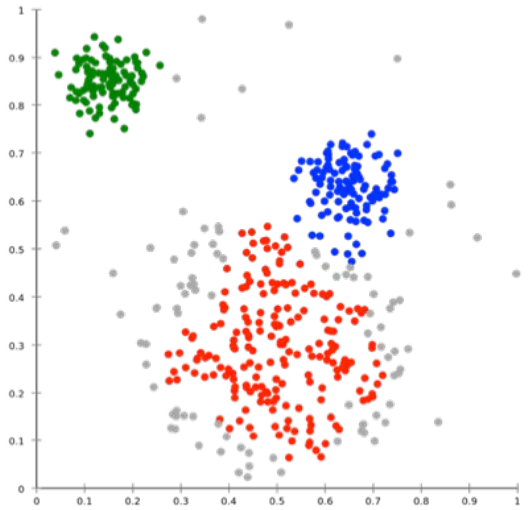
Meet reasonable speed vs accuracy tradeoff

ML Technology Stack



Algorithms

GPU-accelerated Scikit-Learn



Cross Validation

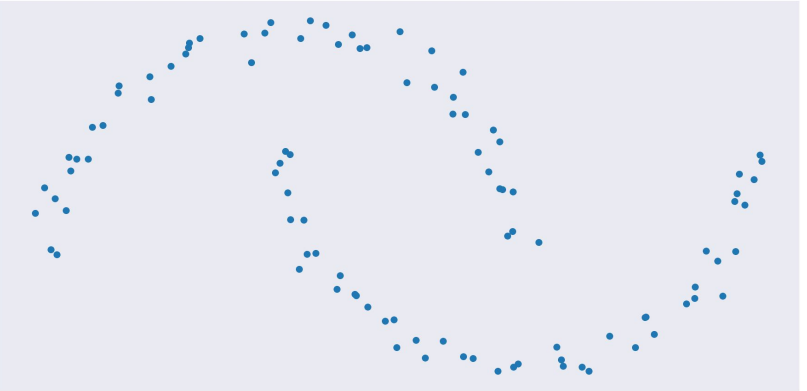
Hyper-parameter Tuning

More to come!

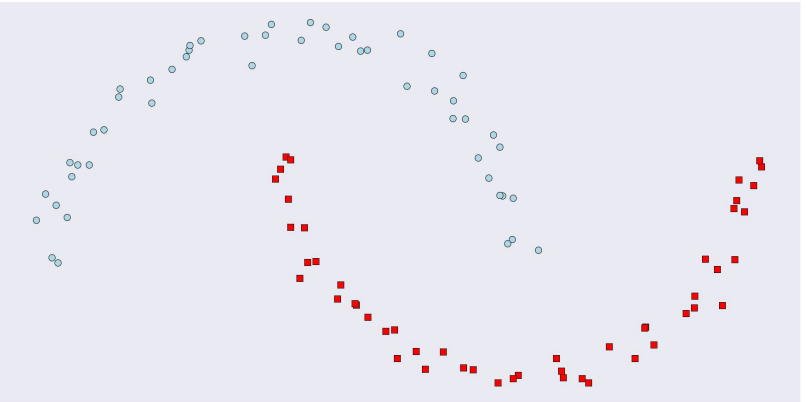
Classification / Regression	Decision Trees / Random Forests Linear Regression Logistic Regression K-Nearest Neighbors
Statistical Inference	Kalman Filtering Bayesian Inference Gaussian Mixture Models Hidden Markov Models
Clustering	K-Means DBSCAN Spectral Clustering
Decomposition & Dimensionality Reduction	Principal Components Singular Value Decomposition UMAP Spectral Embedding
Time Series Forecasting	ARIMA Holt-Winters
Recommendations	Implicit Matrix Factorization

Clustering
Code Example

```
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)
```

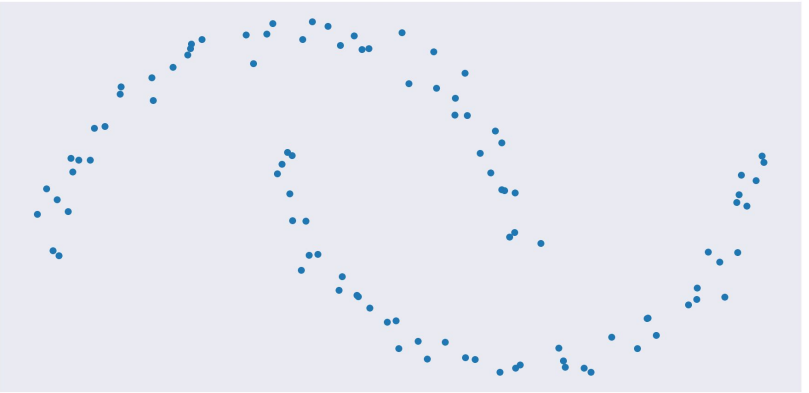


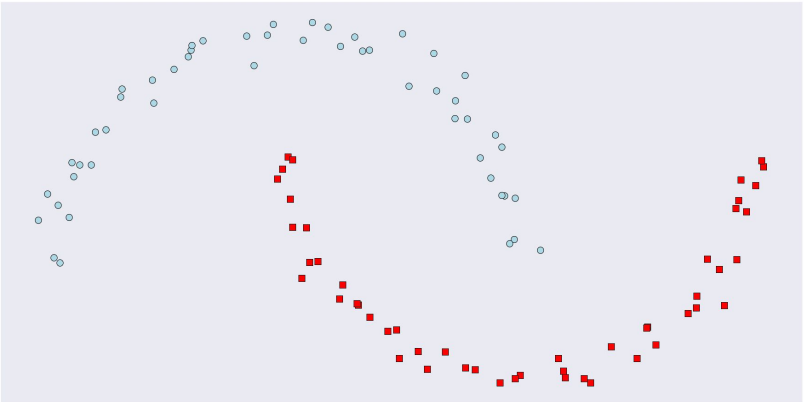
GPU-Accelerated Clustering

Code Example

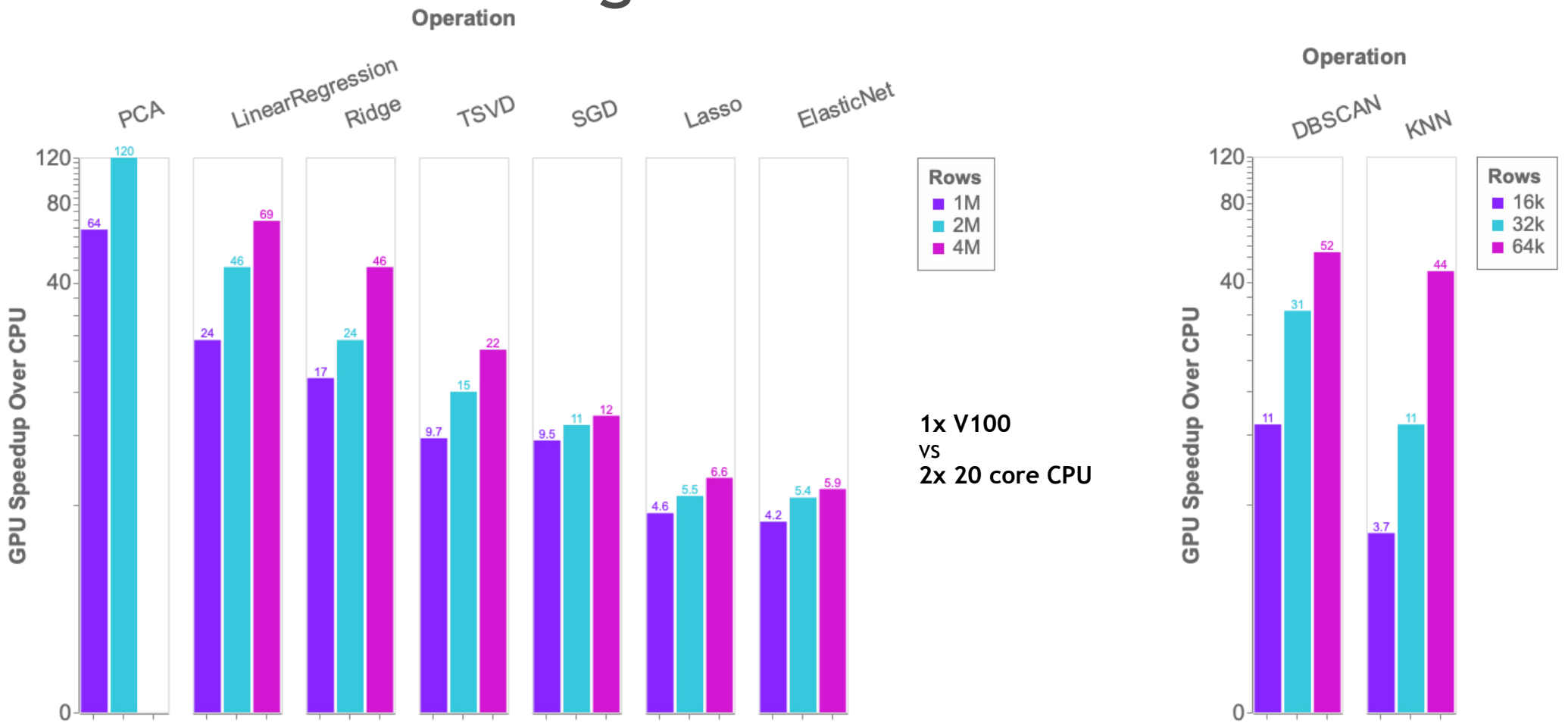
```
from sklearn.datasets import make_moons
import cudf

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



Road to 1.0

October 2018 - RAPIDS 0.1

cuML	Single-GPU	Multi-GPU	Multi-Node-Multi-GPU
Gradient Boosted Decision Trees (GBDT)			
GLM			
Logistic Regression			
Random Forest			
K-Means			
K-NN			
DBSCAN			
UMAP			
ARIMA & Holts-Winters			
Kalman Filter			
t-SNE			
Principal Components			
Singular Value Decomposition			

Road to 1.0

June 2019 - RAPIDS 0.8

cuML	Single-GPU	Multi-GPU	Multi-Node-Multi-GPU
Gradient Boosted Decision Trees (GBDT)	■	■	■
GLM	■	■	
Logistic Regression	■		
Random Forest	■		
K-Means	■		
K-NN	■	■	
DBSCAN	■		
UMAP	■		
ARIMA & Holts-Winters			
Kalman Filter	■		
t-SNE			
Principal Components	■		
Singular Value Decomposition	■	■	

Road to 1.0

August 2019 - RAPIDS 0.9

cuML	Single-GPU	Multi-GPU	Multi-Node-Multi-GPU
Gradient Boosted Decision Trees (GBDT)	■	■	■
GLM	■	■	
Logistic Regression	■		
Random Forest	■	■	■
K-Means	■	■	■
K-NN	■	■	
DBSCAN	■		
UMAP	■		
ARIMA & Holts-Winters	■		
Kalman Filter	■		
t-SNE	■		
Principal Components	■		
Singular Value Decomposition	■	■	

Road to 1.0

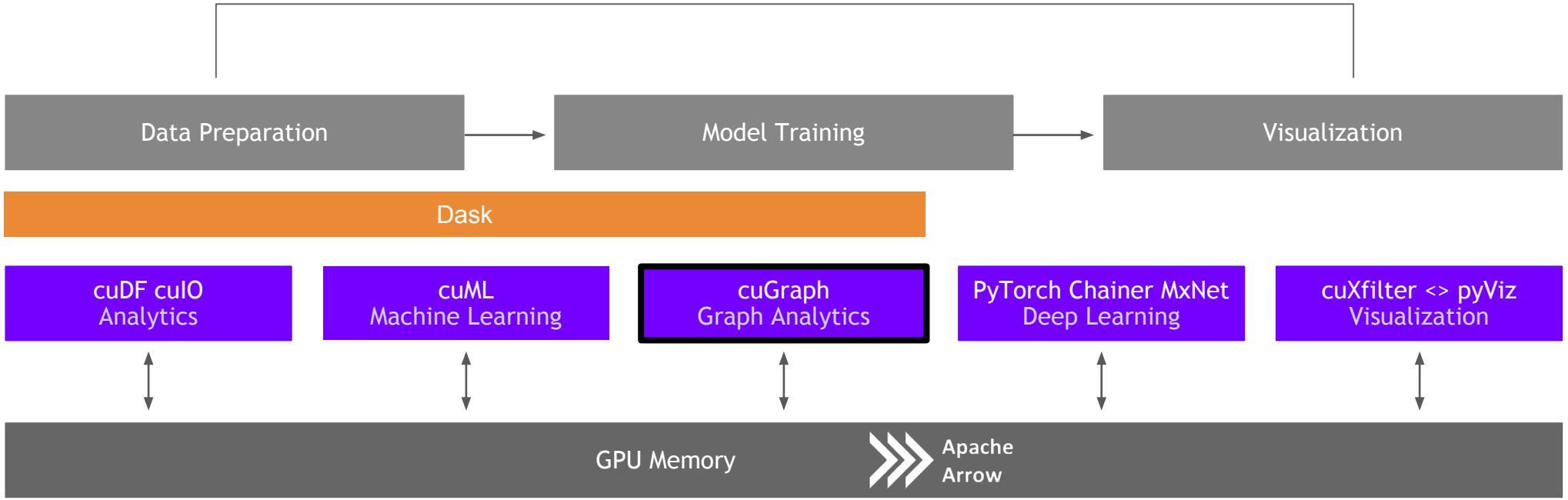
January 2020 - RAPIDS 0.12?

cuML	Single-GPU	Multi-GPU	Multi-Node-Multi-GPU
Gradient Boosted Decision Trees (GBDT)			
GLM			
Logistic Regression			
Random Forest			
K-Means			
K-NN			
DBSCAN			
UMAP			
ARIMA & Holts-Winters			
Kalman Filter			
t-SNE			
Principal Components			
Singular Value Decomposition			

cuGraph

Graph Analytics

More connections more insights



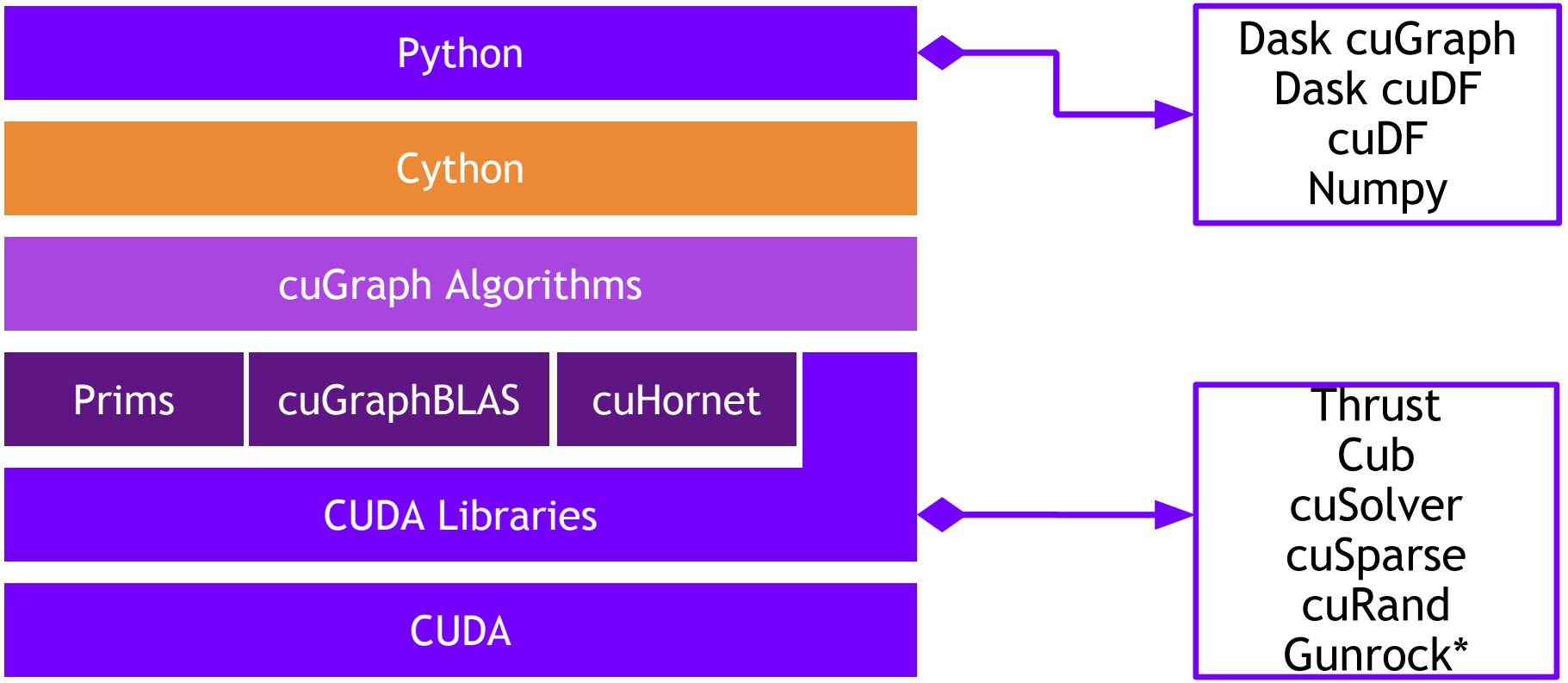
GOALS AND BENEFITS OF CUGRAPH

Focus on Features an Easy-of-Use

- Seamless integration with cuDF and cuML
  - Python APIs accepts and returns cuDF DataFrames
- Features
  - Extensive collection of algorithm, primitive, and utility functions**
  - Python API:
    - Multiple APIs: NetworkX, Pregel**, Frontier**
    - Graph Query Language**
  - C/C++
    - Full featured C++ API that gives both an easy to use experience, as well as exposing lower-level granular control to developers
- Breakthrough Performance

** On Roadmap

Graph Technology Stack

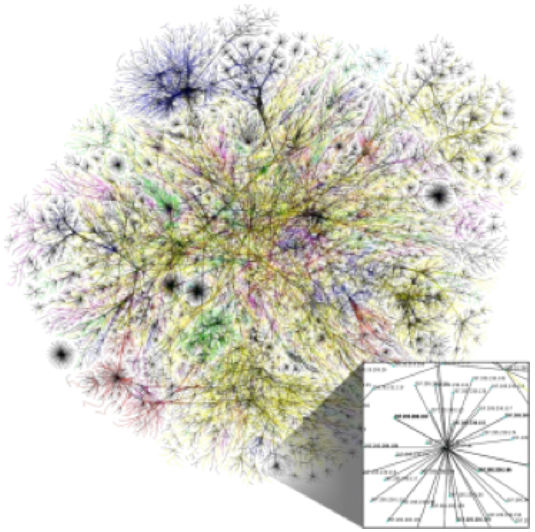


nvGRAPH has been Opened Sourced and integrated into cuGraph. A legacy version is available in a RAPIDS GitHub repo

* Gunrock is from UC Davis

Algorithms

GPU-accelerated NetworkX



Query Language

Multi-GPU

Utilities

More to come!

Community	Spectral Clustering Balanced-Cut Modularity Maximization Louvain Subgraph Extraction Triangle Counting
Components	Weakly Connected Components Strongly Connected Components
Link Analysis	Page Rank Personal Page Rank
Link Prediction	Jaccard Weighted Jaccard Overlap Coefficient
Traversal	Single Source Shortest Path (SSSP) Breadth First Search (BFS)
Structure	COO-to-CSR Transpose Renumbering

WHAT'S IN CUGRAPH

Current Single GPU Algorithms – as of v0.8

- COO-to/from-CSR Conversion
- Renumbering
- PageRank
- Personal PageRank
- Jaccard (1 and 2-hop)
- Weighted Jaccard
- Overlap Coefficient
- Single Source Shortest Path (SSSP)
- Breadth First Search (BFS)
- Triangle Counting (TC)
- Subgraph Extraction
- Spectral Clustering
  - Balanced-Cut
  - Modularity Maximization
- Louvain
- Graph functions
  - Size, Order, Degree
  - Get two hop neighbors

SG – Single GPU MG – Multi-GPU

Louvain Single Run

```
G = cugraph.Graph()
G.add_edge_list(gdf["src_0"], gdf["dst_0"], gdf["data"])
df, mod = cugraph.nvLouvain(G)
```

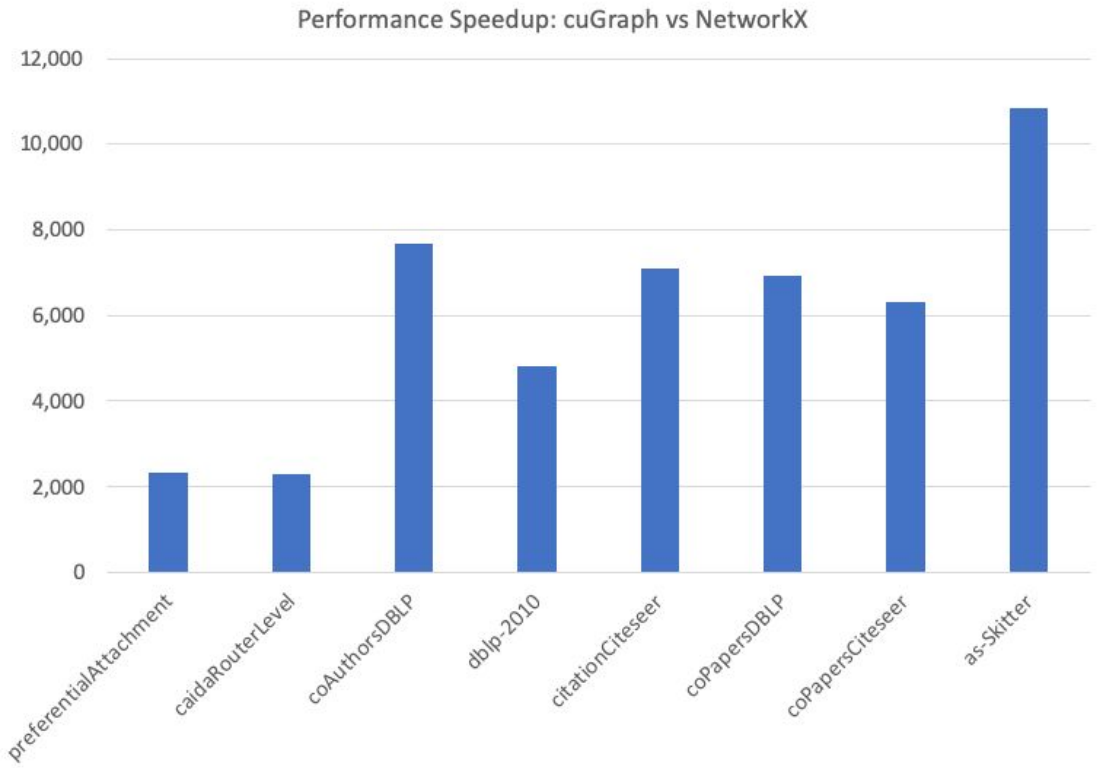
Louvain returns:

cudf.DataFrame with two names columns:

louvain["vertex"]: The vertex id.

louvain["partition"]: The assigned partition.

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



PageRank Performance

Performance Using PageRank Pipeline Benchmark**

Single and Dual GPU on Commodity Workstation (PCIe)

Vertices	Edges	Single GPU	Dual GPU
1,048,576	16,777,216	0.019	0.20
2,097,152	3,354,432	0.047	0.035
4,194,304	134,217,728	0.302	0.162
16,777,216	268,435,456	0.771	0.353
33,554,432	536,870,912	1.747	0.821
67,108,864	1,073,741,824		1.880

Single DGX-2 (16 NVLinked GPUs)

Vertices	Edges	Runtime
33,554,432	536,870,912	1.405
67,108,864	1,073,741,824	1.389
134,217,728	2,122,307,214	1.389
268,435,456	4,294,967,296	1.410
536,870,912	8,589,934,592	1.468

Running on V100 32G

** Benchmarking code. Final produce performance might differ

Road to 1.0

June 2019 - RAPIDS 0.8

cuGraph	Single-GPU	Multi-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			
Louvain			
Spectral Clustering			
InfoMap			
K-Cores			

Road to 1.0

August 2019 - RAPIDS 0.9

cuGraph	Single-GPU	Multi-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	■		
Louvain	■		
Spectral Clustering	■		
InfoMap			
K-Cores			

Road to 1.0

January 2020 - RAPIDS 0.12?

cuGraph	Single-GPU	Multi-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			
Louvain			
Spectral Clustering			
InfoMap			
K-Cores			

UPCOMING

(MG == Multi-GPU, SG == Single-GPU)

- **Up Next: 0.9**
  - MG PageRank
  - MG COO-to-CSR
  - MG Degree Computation
  - Strongly Connected Component
  - Initial Hornet integration (cuHornet)
  - Draft Pregel/BSP versions
    - bicliques
  - Utility functions
- **Next Steps: 0.10**
  - Initial Gunrock Integration
    - Subgraph Matching
  - Initial graphBLAS Integration
  - cuHornet Analytics
    - Katz Centrality
    - K-Cores
  - Utilities
    - Symmetrization
    - Transpose
- **The Future: 0.10 +**
  - Graph query language: OpenCypher
  - MG BSF
  - Expanded community detection
  - Dynamic Graph support
  - Betweenness Centrality
  - More Multi-GPU analytics

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

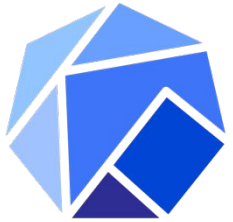
GPU accelerated SQL engine built on top of RAPIDS

Distributed stream processing using RAPIDS and Dask

Deploy RAPIDS Everywhere

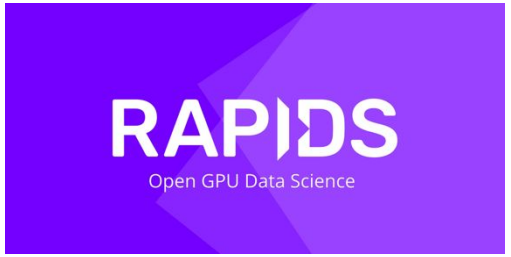
Focused on robust functionality, deployment, and user experience

Cloud Dataproc



Kubeflow

Azure Machine Learning

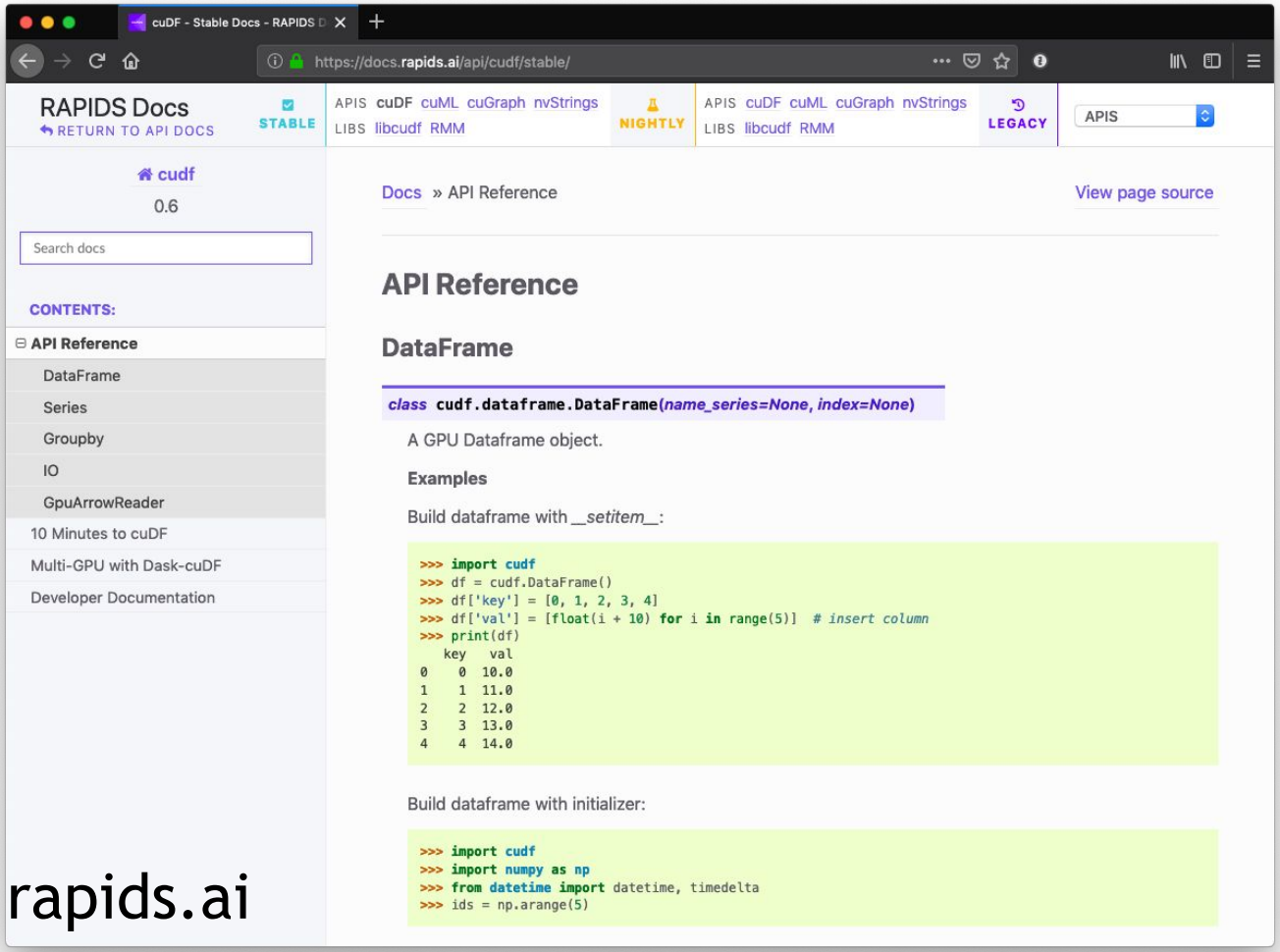


Integration with major cloud providers
Both containers and cloud specific machine instances
Support for Enterprise and HPC Orchestration Layers

Getting Started

RAPIDS Docs

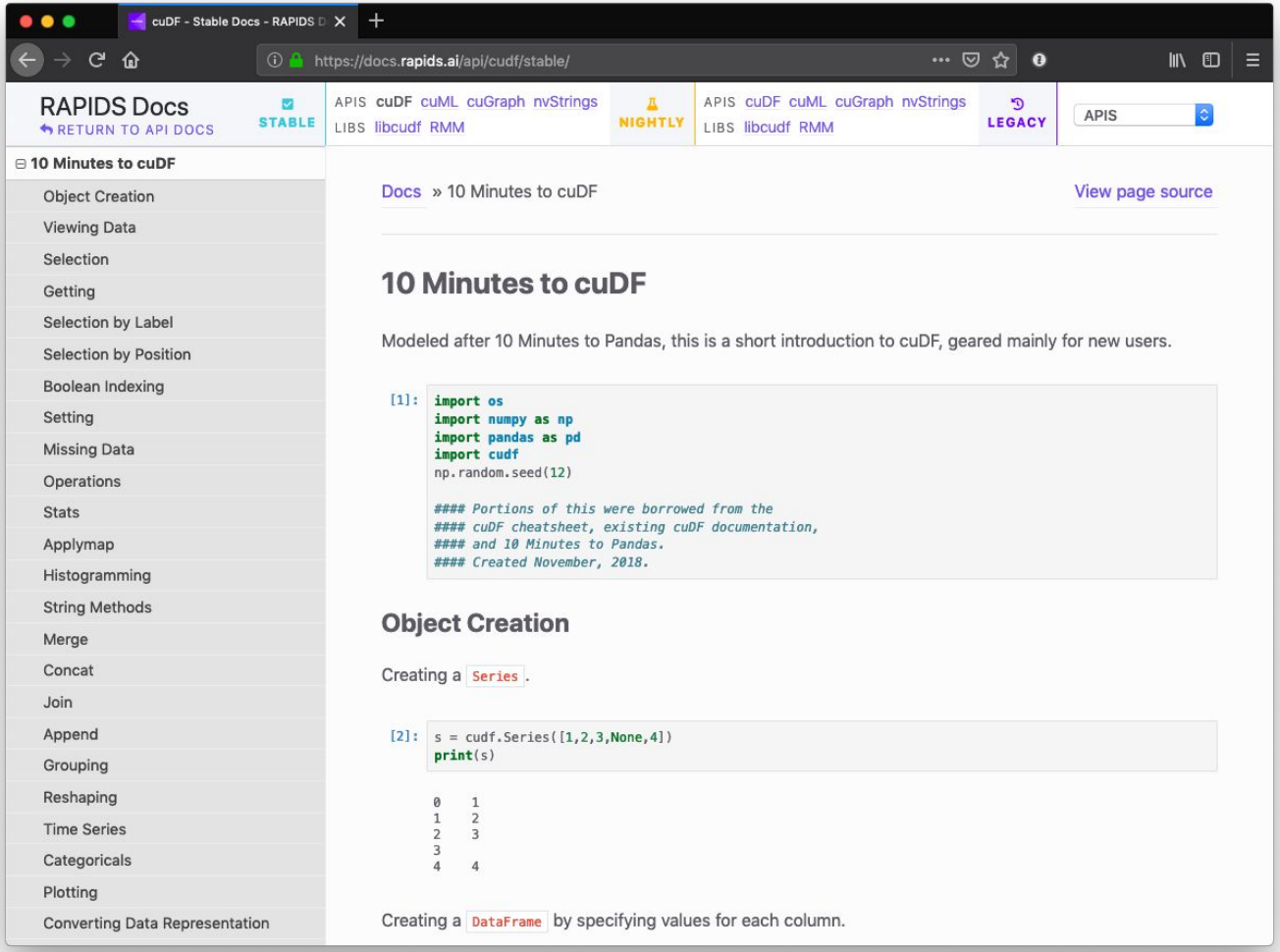
New, improved, and easier to use



https://docs.rapids.ai

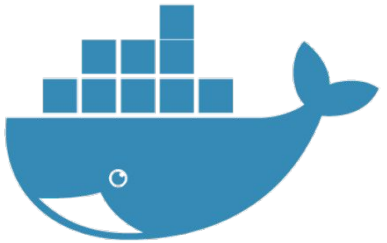
RAPIDS Docs

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

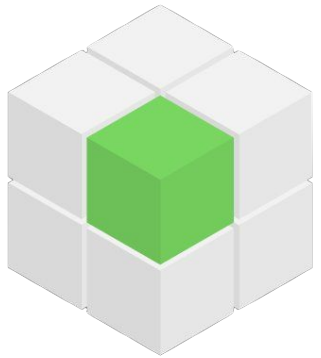
https://rapids.ai

@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