RAPIDS

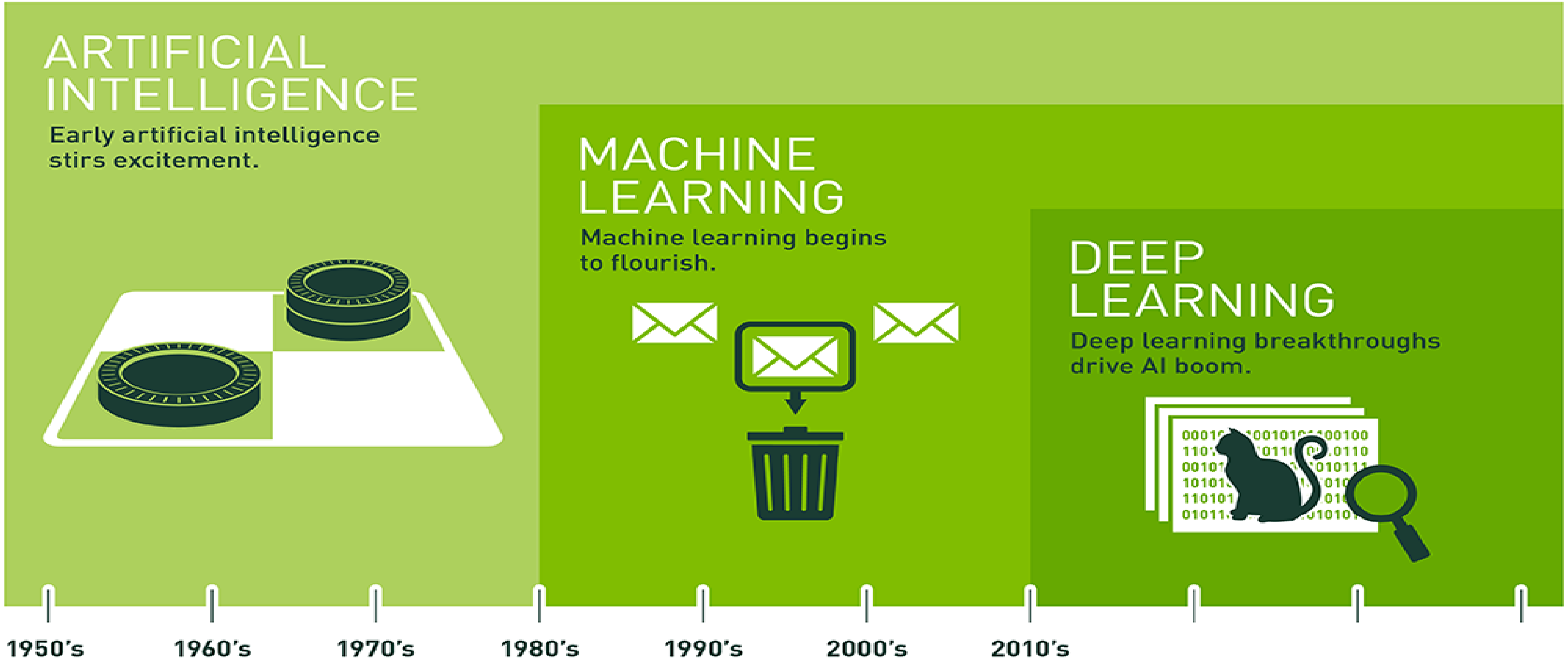
ACCELERATING END-TO-END DATA SCIENCE

Mitra Bhanu Rath – Sr. Solution Architect

AGENDA

- Introduction to RAPIDS & how it is built
- RAPIDS Overview: cuDF, cuML, cuGraph, DL and Visualization
- Benchmark with XGBoost Example
- Code comparision
- How to get started

CAPABILITY OF MACHINE TO IMITATE INTELLIGENT BEHAVIOR

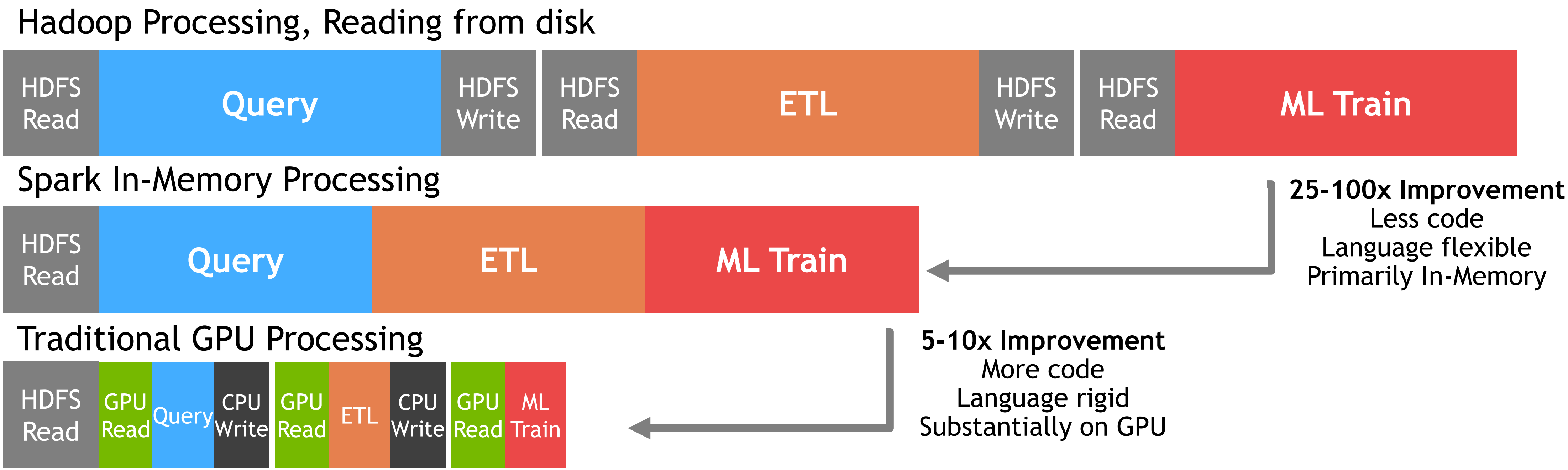


Since an early flush of optimism in the 1950s, smaller subsets of artificial intelligence – first machine learning, then deep learning, a subset of machine learning – have created ever larger disruptions.

NVIDIA

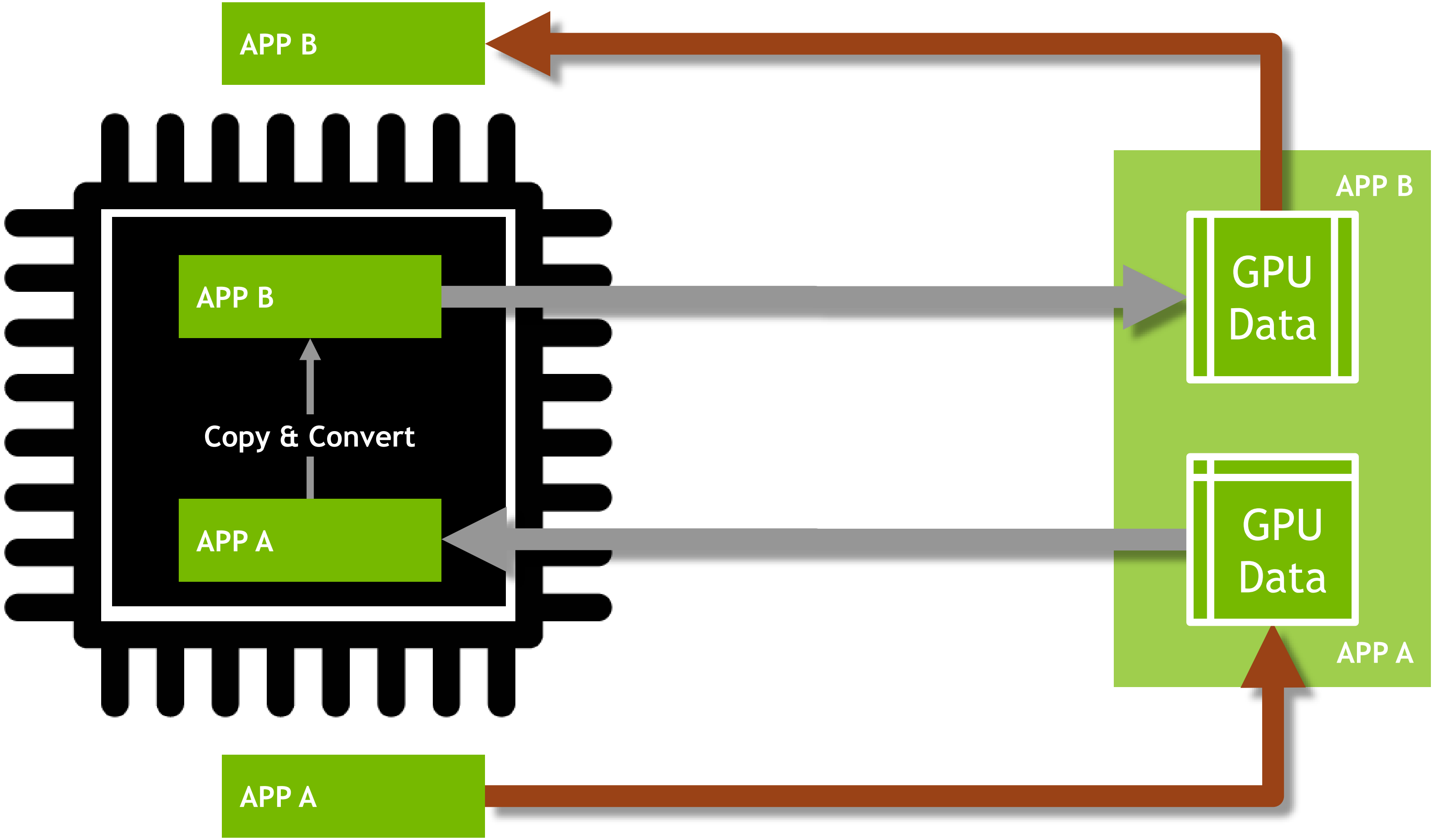
DATA PROCESSING EVOLUTION

Faster data access, less data movement



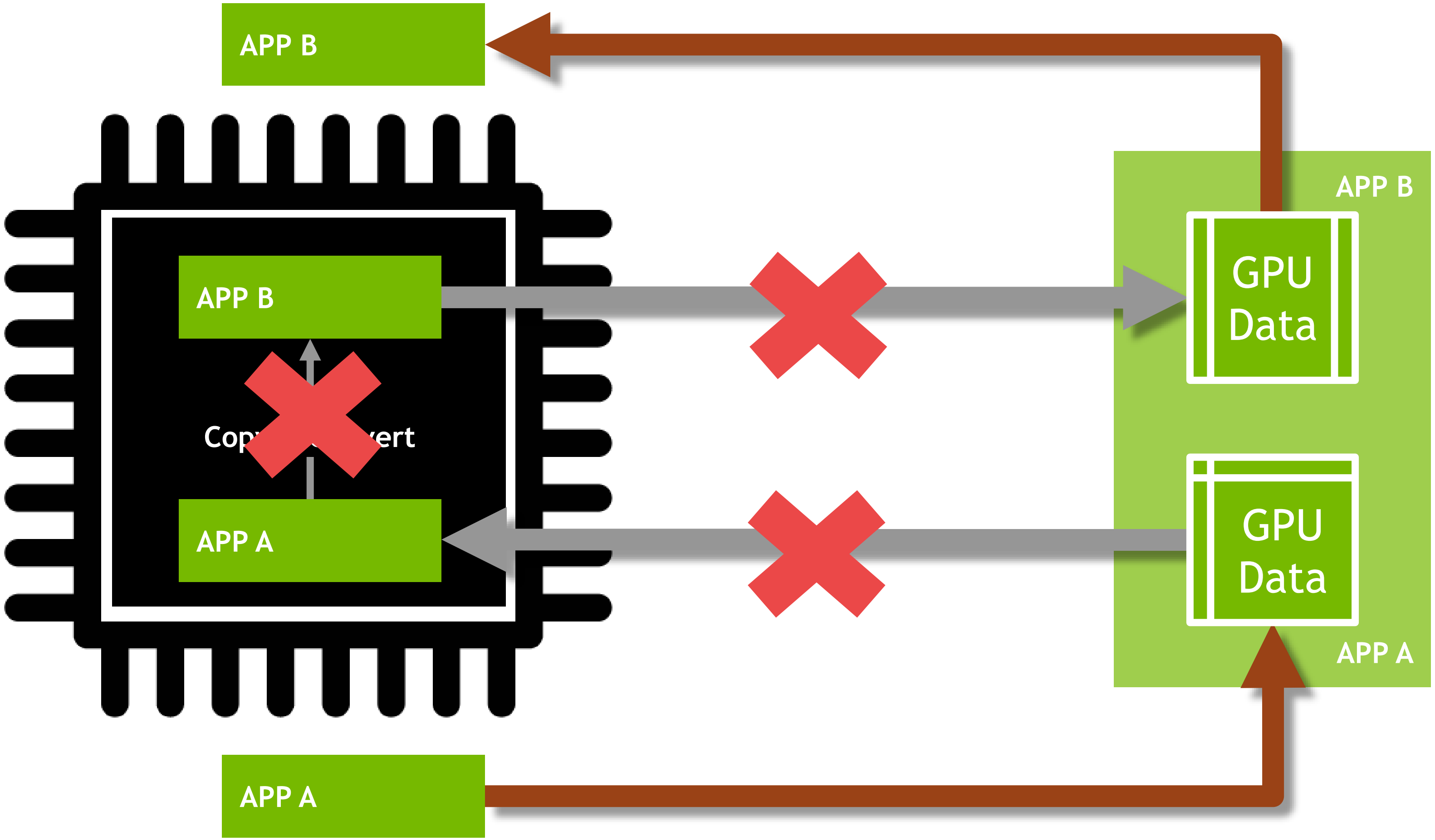
DATA MOVEMENT AND TRANSFORMATION

The bane of productivity and performance

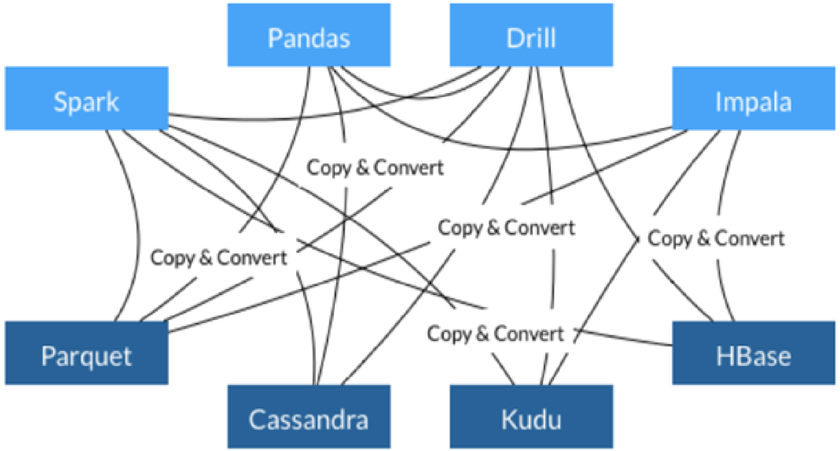


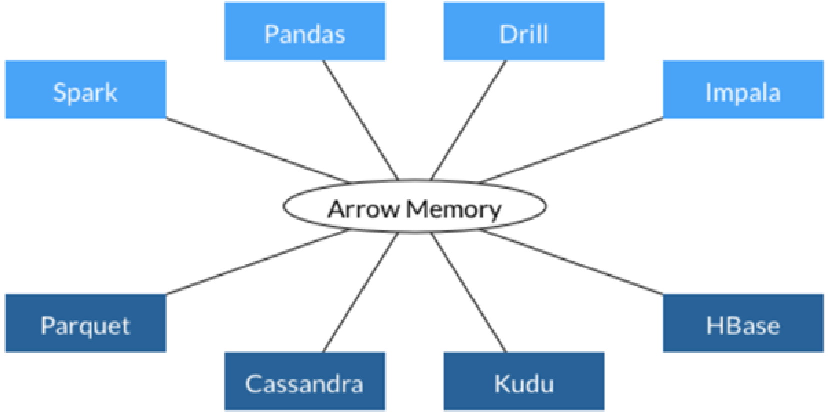
DATA MOVEMENT AND TRANSFORMATION

What if we could keep data on the GPU?



LEARNING FROM APACHE ARROW





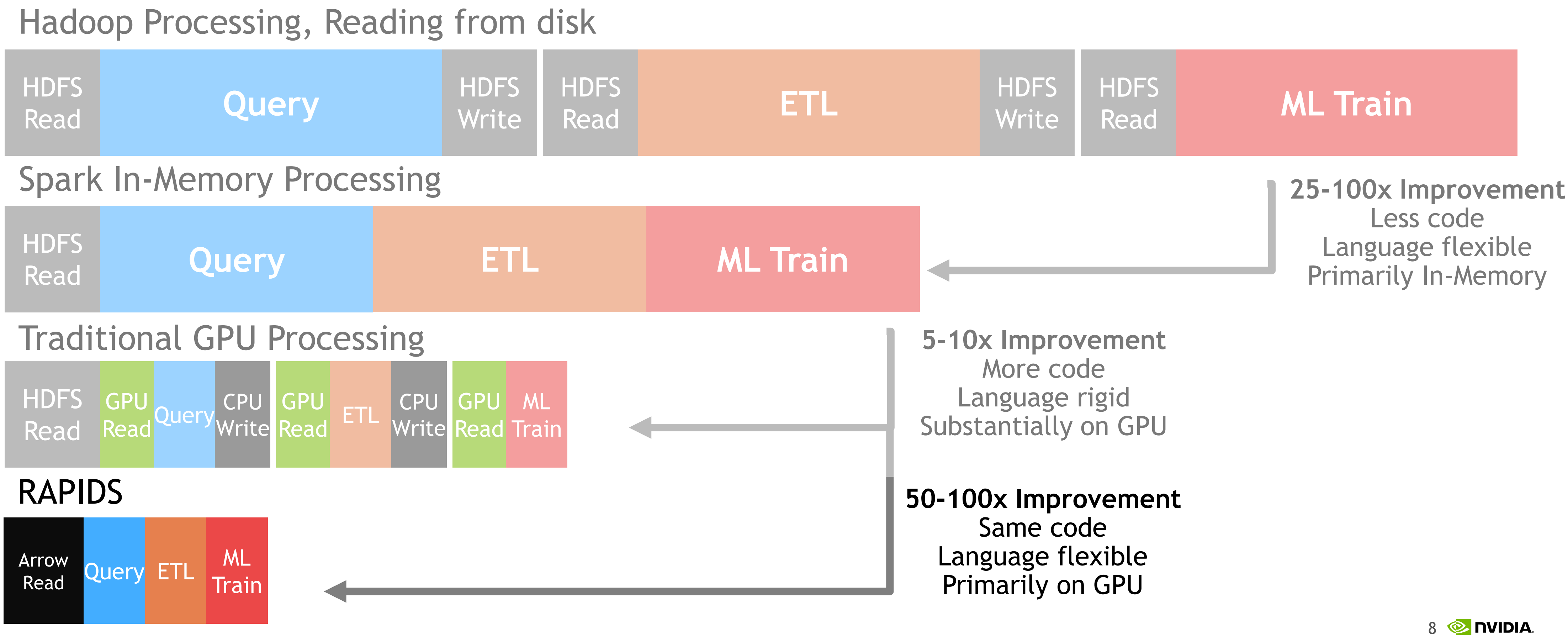
- Each system has its own internal memory format
- 70-80% computation wasted on serialization and deserialization
- Similar functionality implemented in multiple projects

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

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

DATA PROCESSING EVOLUTION

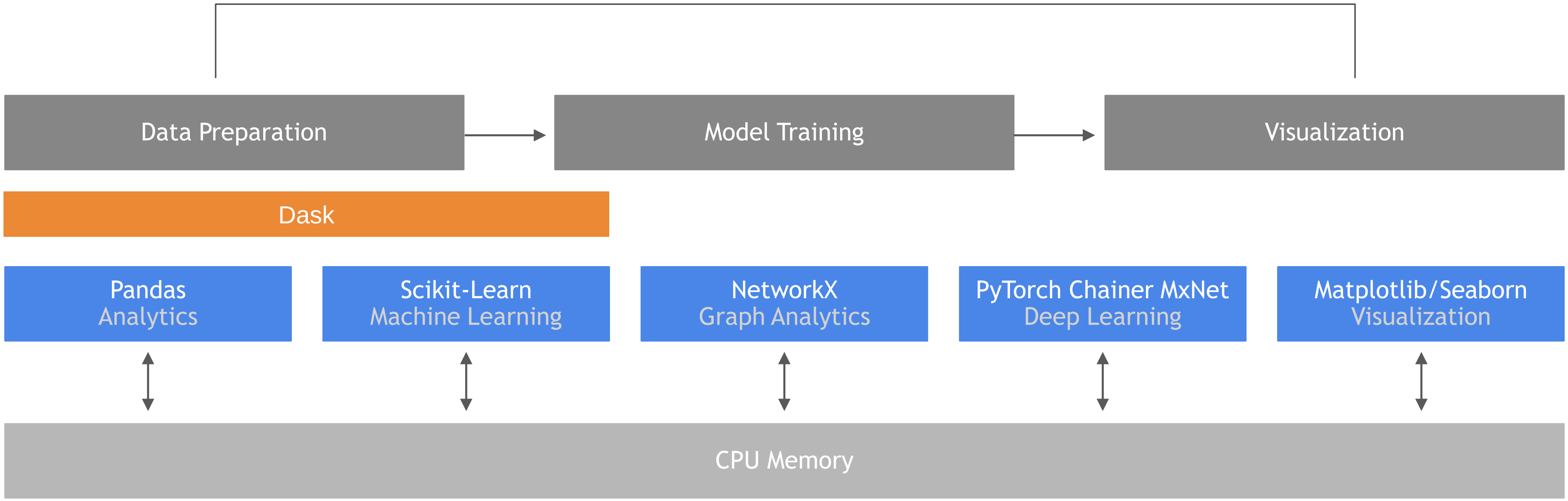
Faster data access, less data movement



RAPIDS CORE

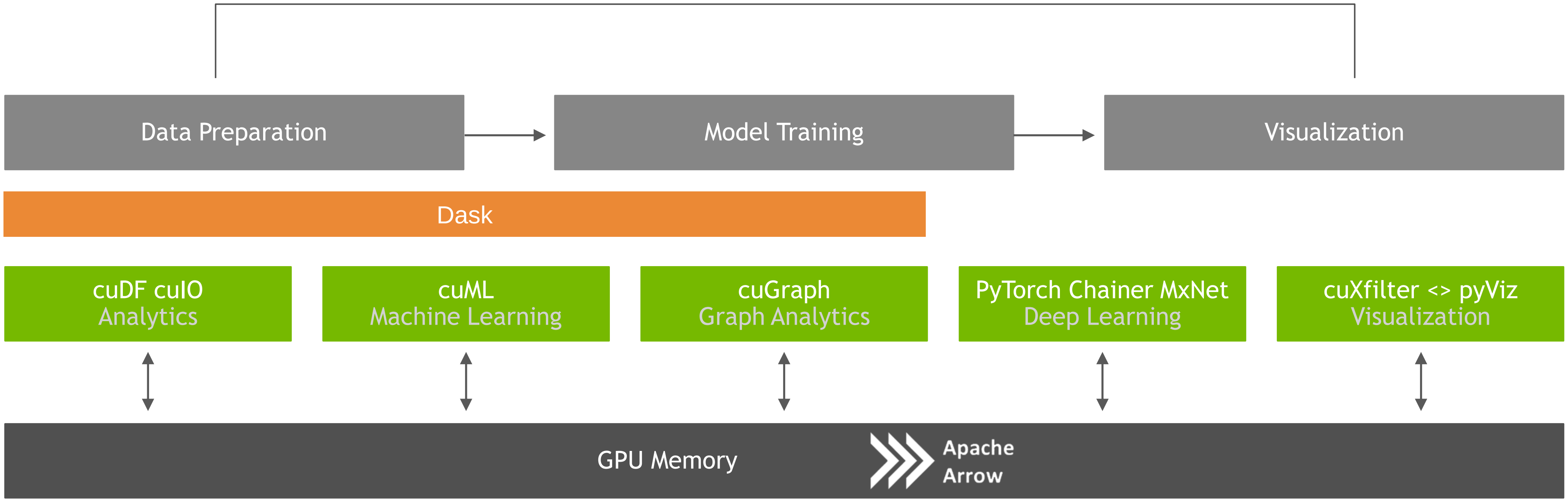
OPEN SOURCE DATA SCIENCE ECOSYSTEM

Familiar Python APIs



RAPIDS

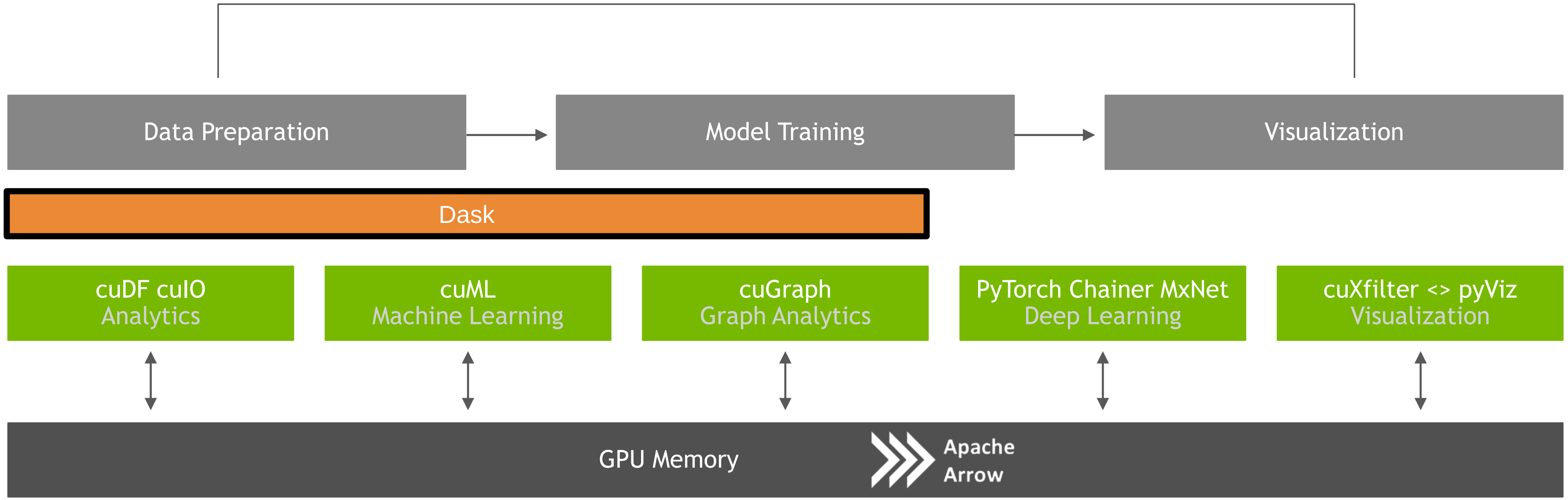
End-to-End Accelerated GPU Data Science



DASK

RAPIDS

Scaling RAPIDS with Dask



WHY DASK?

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
  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

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 scikit learn

Dask

Multi-core and Distributed PyData

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

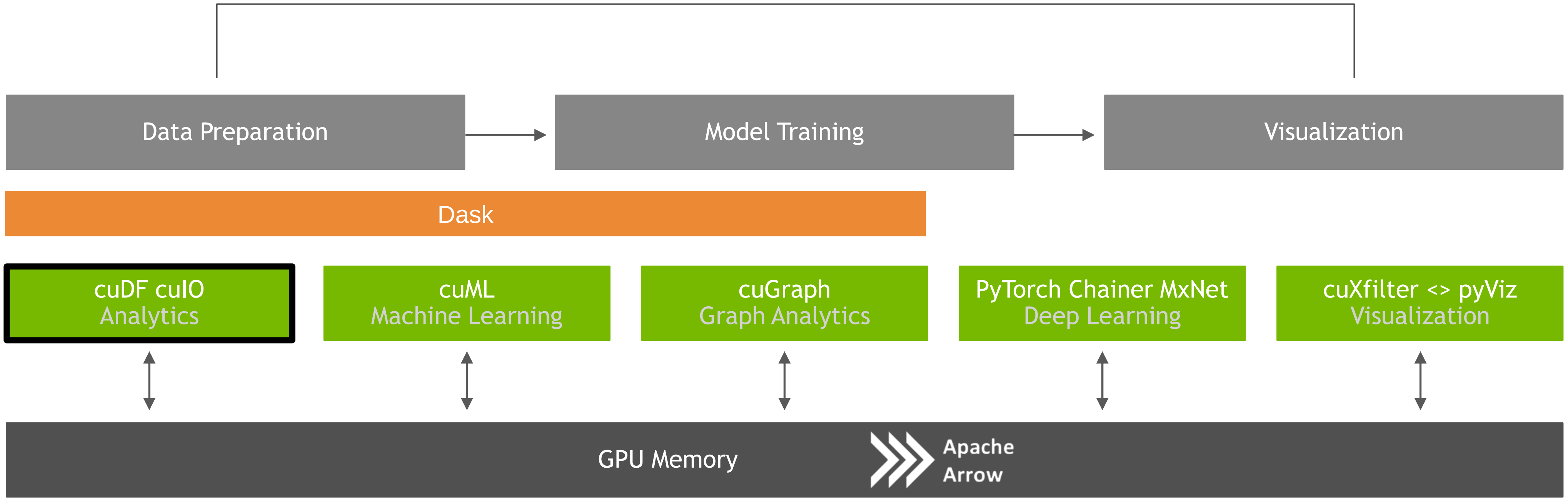
DASK

Scale out / Parallelize

CUDF

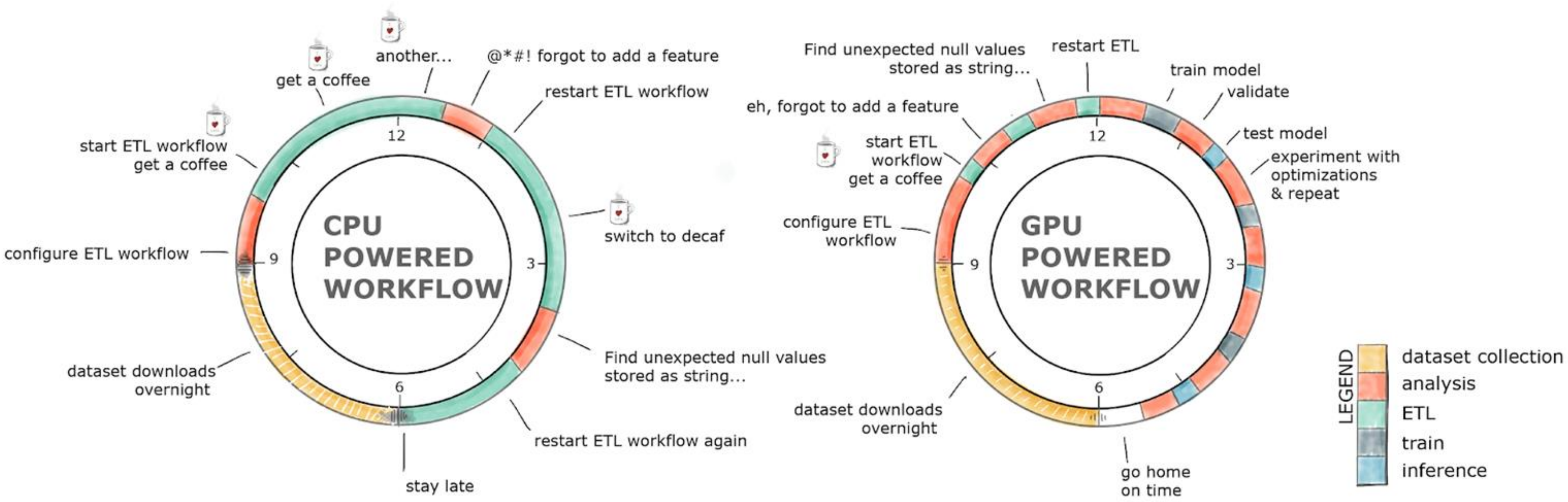
RAPIDS

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()
```

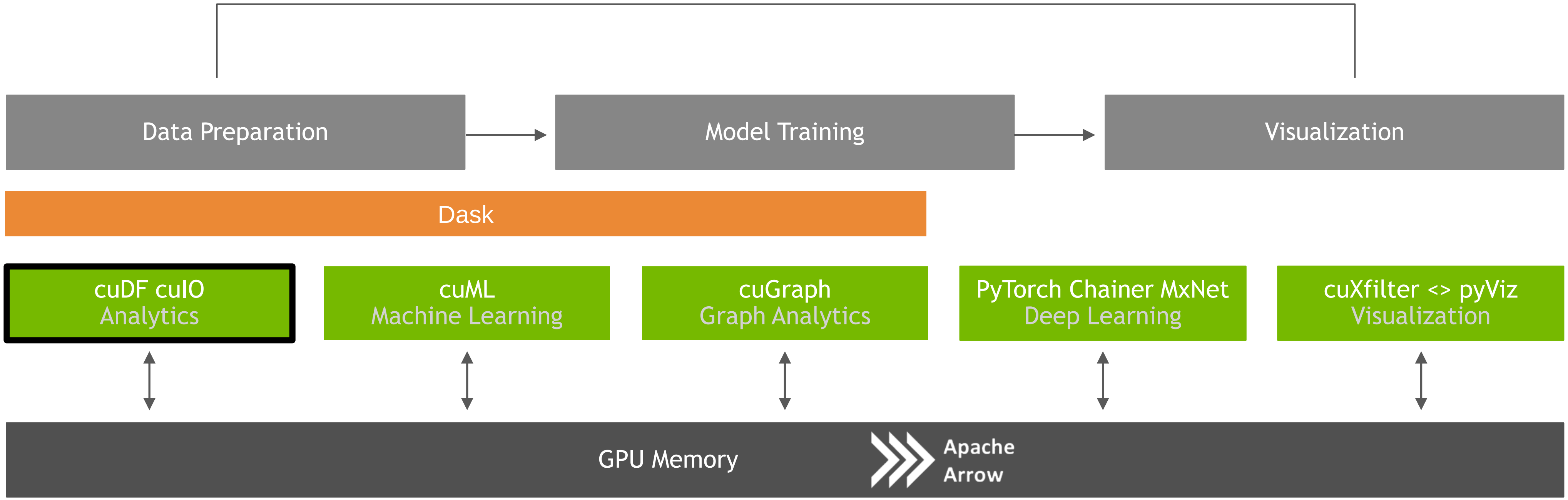
```
In [7]: #Here we can see how we can control what the value of our dummies with the replace method and turn
        strings to ints
        gdf['City_Category'] = gdf.City_Category.str.replace('A', '1')
        gdf['City_Category'] = gdf.City_Category.str.replace('B', '2')
        gdf['City_Category'] = gdf.City_Category.str.replace('C', '3')
        gdf['City_Category'] = gdf['City_Category'].str.stoi()
```

Python

- 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



EXTRACTION IS THE CORNERSTONE

cuIO is born

- Follow Pandas APIs and provide >10x speedup

- CSV Reader - v0.2, CSV Writer v0.8
- Parquet Reader – v0.7, Parquet Writer v0.10
- 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

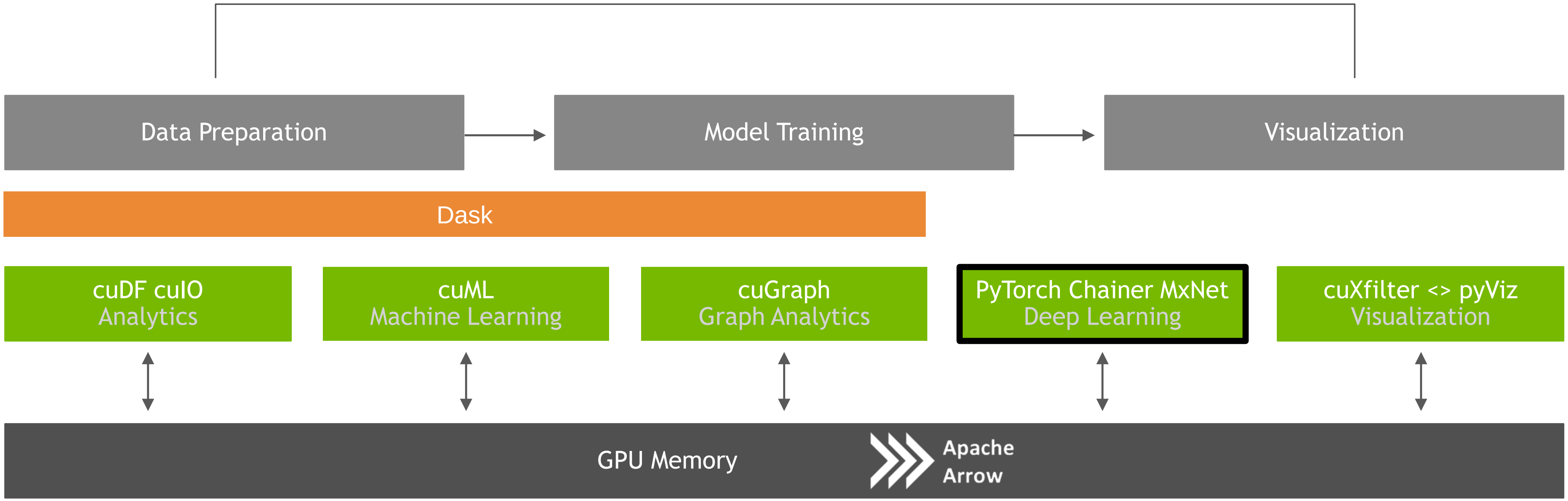
```
1]: import pandas, cudf

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!

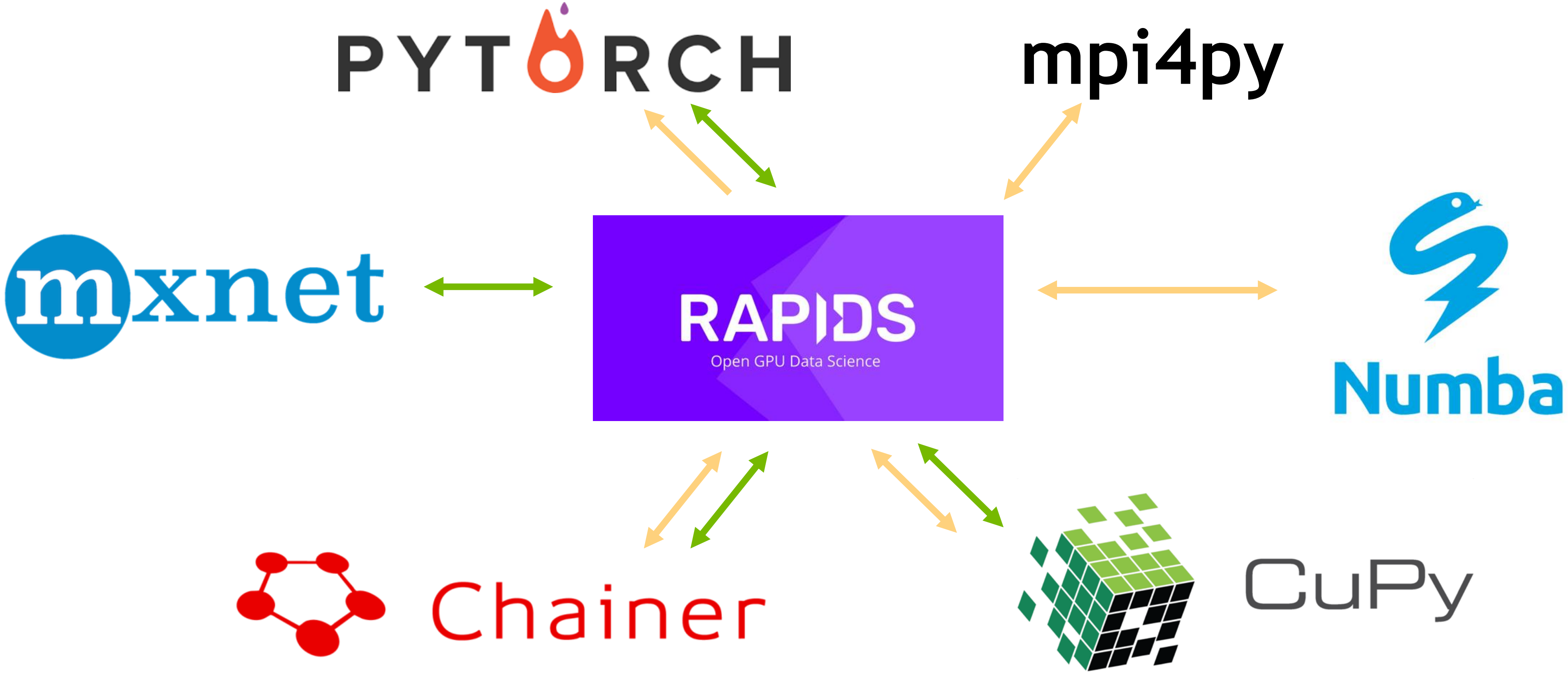
RAPIDS

Building bridges into the array ecosystem



INTEROPERABILITY FOR THE WIN

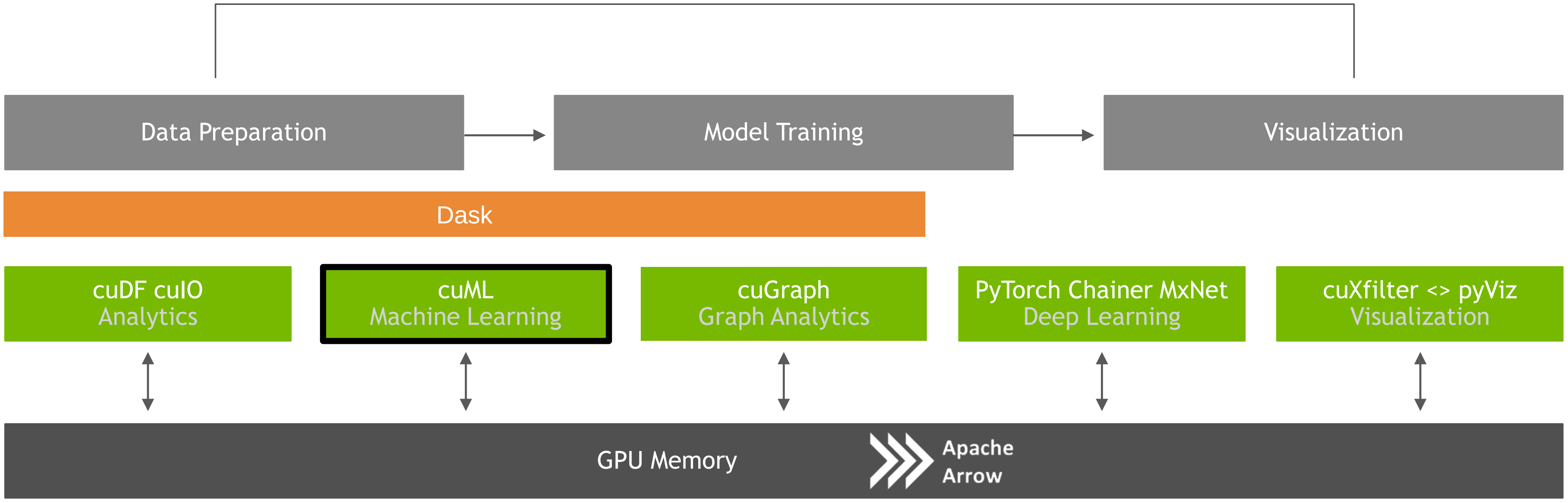
DLPack and __cuda_array_interface__



CUML

MACHINE LEARNING

More models more problems

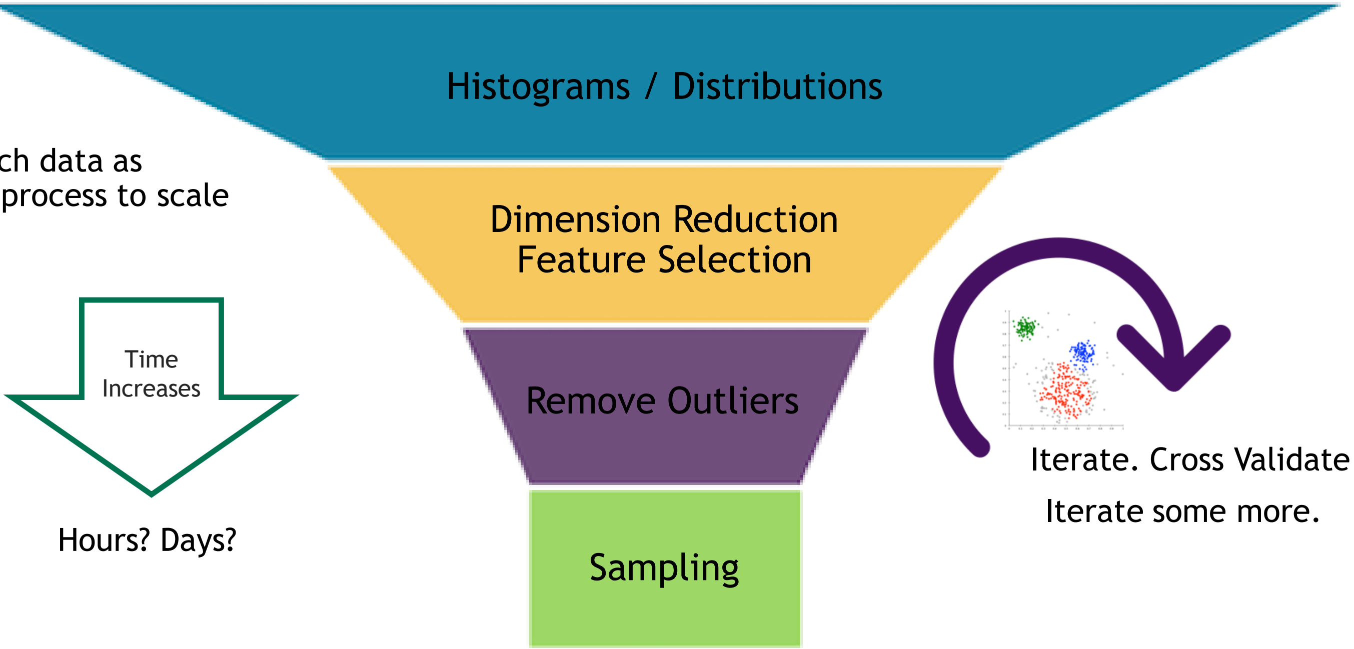


PROBLEM

Data sizes continue to grow

Massive Dataset

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



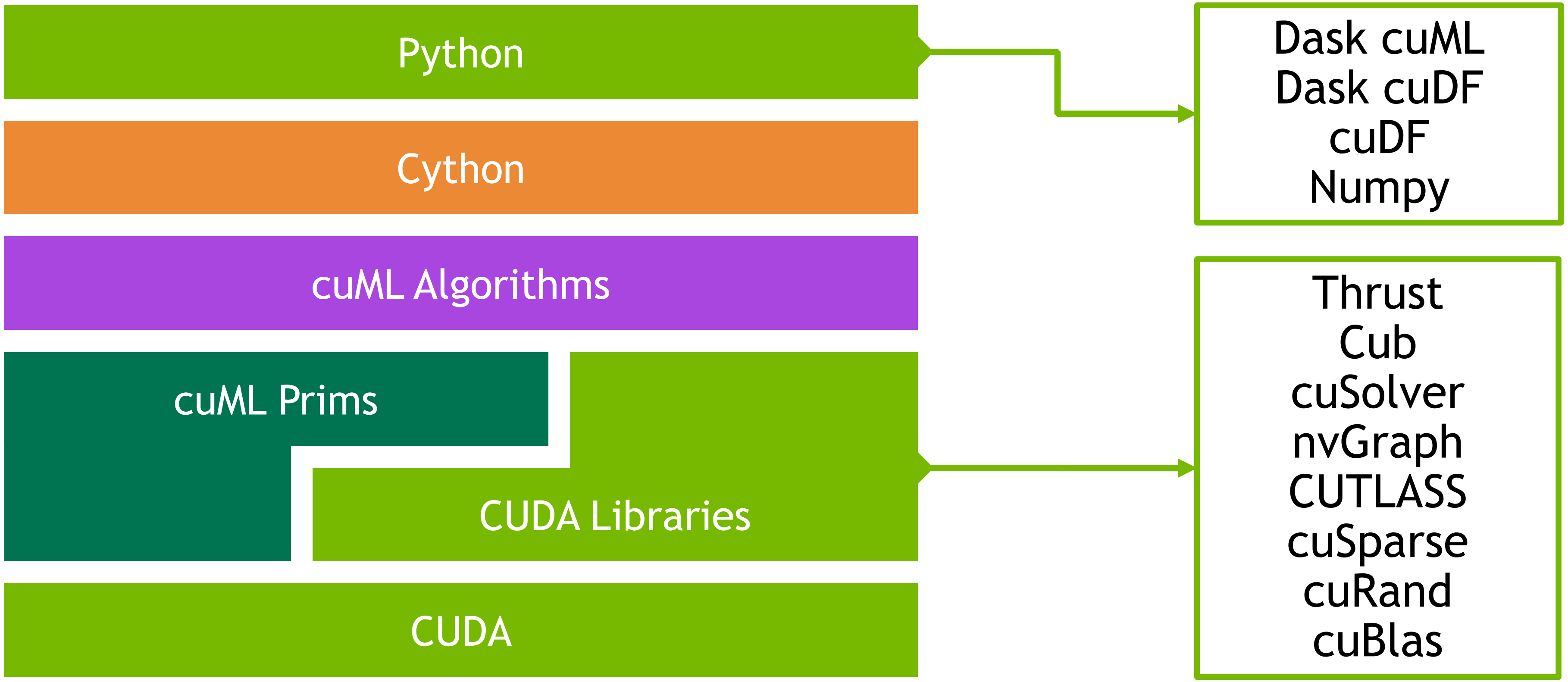
Hours? Days?

Iterate. Cross Validate & Grid Search.

Iterate some more.

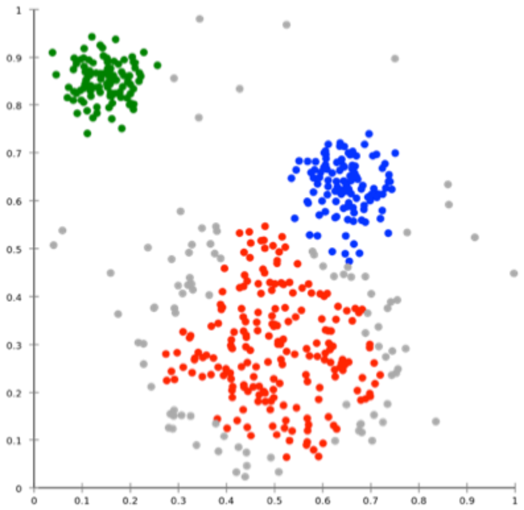
Meet reasonable speed vs accuracy tradeoff

ML TECHNOLOGY STACK



ALGORITHMS

GPU-accelerated Scikit-Learn



Cross Validation

Hyper-parameter Tuning

More to come!

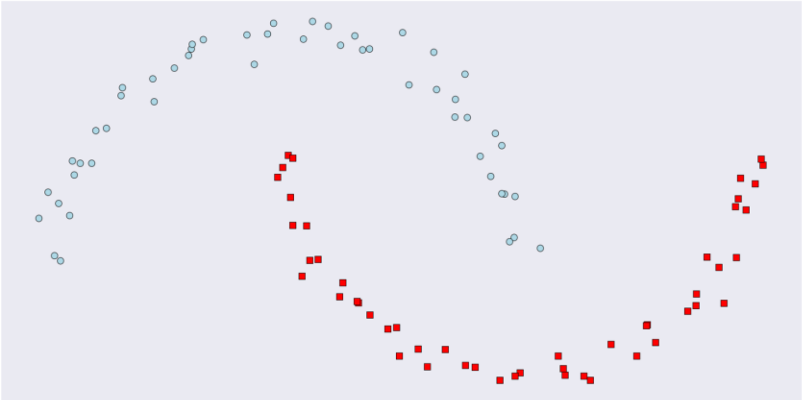
Classification / Regression	Decision Trees / Random Forests Linear Regression Logistic Regression K-Nearest Neighbors
Statistical Inference	Kalman Filtering Bayesian Inference Gaussian Mixture Models Hidden Markov Models
Clustering	K-Means DBSCAN Spectral Clustering
Decomposition & Dimensionality Reduction	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)
```

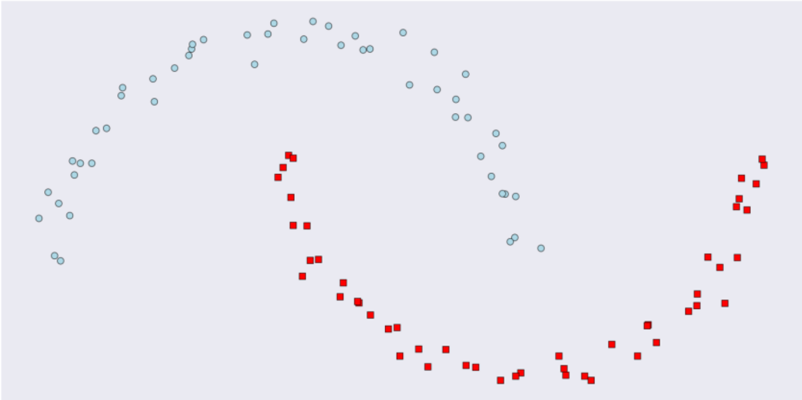


GPU-ACCELERATED CLUSTERING

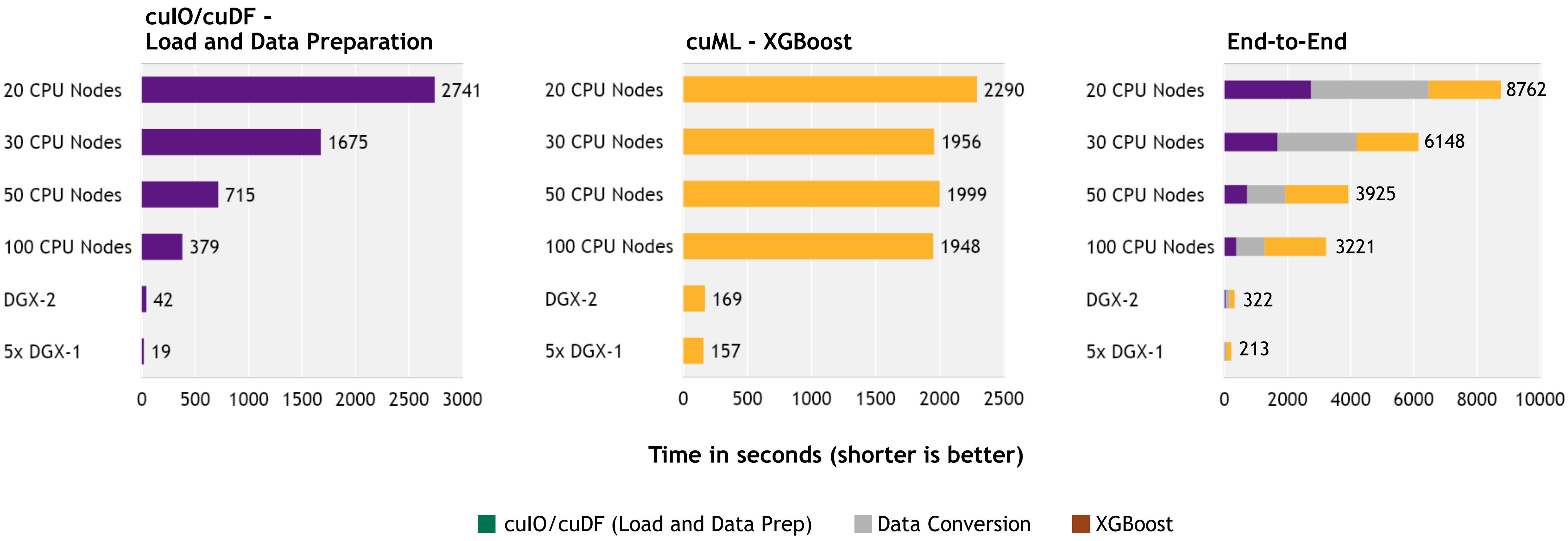
Code Example

```
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)
```



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

CPU vs GPU

KNN

Training results:

CPU: ~9 minutes
GPU: 1.12 seconds

System: AWS p3.8xlarge
CPUs: Intel(R) Xeon(R) CPU E5-2686 v4 @ 2.30GHz, 32 vCPU cores, 244 GB RAM
GPU: Tesla V100 SXM2 16GB
Dataset: https://github.com/rapidsai/cuml/tree/master/python/notebooks/data

k-Nearest Neighbors (KNN)

Before...

Specific: Import CPU algorithm

[1]:
```
from sklearn.neighbors import KDTree as KNN
```

Common: Helper functions

[2]:
```
# Timer, load_data...
from helper import *
```

Common: Data loading and algo params

[3]:
```
# Data loading
nrows = 2**17
ncols = 40

X = load_data(nrows, ncols)
print('data', X.shape)

# Algorithm parameters
n_neighbors = 10
```

```
use mortgage data
data (131072, 40)
```

Specific: Training

[4]:
```
%%time
knn = KNN(X)
_ = knn.query(X, n_neighbors)
```

```
CPU times: user 9min 2s, sys: 272 ms, total: 9min 2s
Wall time: 8min 59s
```

...Now!

Specific: Import GPU algorithm

[1]:
```
from cuml import KNN
```

Common: Helper functions

[2]:
```
# Timer, load_data...
from helper import *
```

Common: Data loading and algo params

[3]:
```
# Data loading
nrows = 2**17
ncols = 40

X = load_data(nrows, ncols)
print('data', X.shape)

# Algorithm parameters
n_neighbors = 10
```

```
use mortgage data
data (131072, 40)
```

Specific: DataFrame from Pandas to cuDF

[4]:
```
%%time
import cudf
X = cudf.DataFrame.from_pandas(X)
```

```
CPU times: user 3 s, sys: 552 ms, total: 3.56 s
Wall time: 839 ms
```

Specific: Training

[5]:
```
%%time
knn = KNN(n_gpus=1)
knn.fit(X)
_ = knn.query(X, n_neighbors)
```

```
CPU times: user 692 ms, sys: 428 ms, total: 1.12 s
Wall time: 1.12 s
```

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

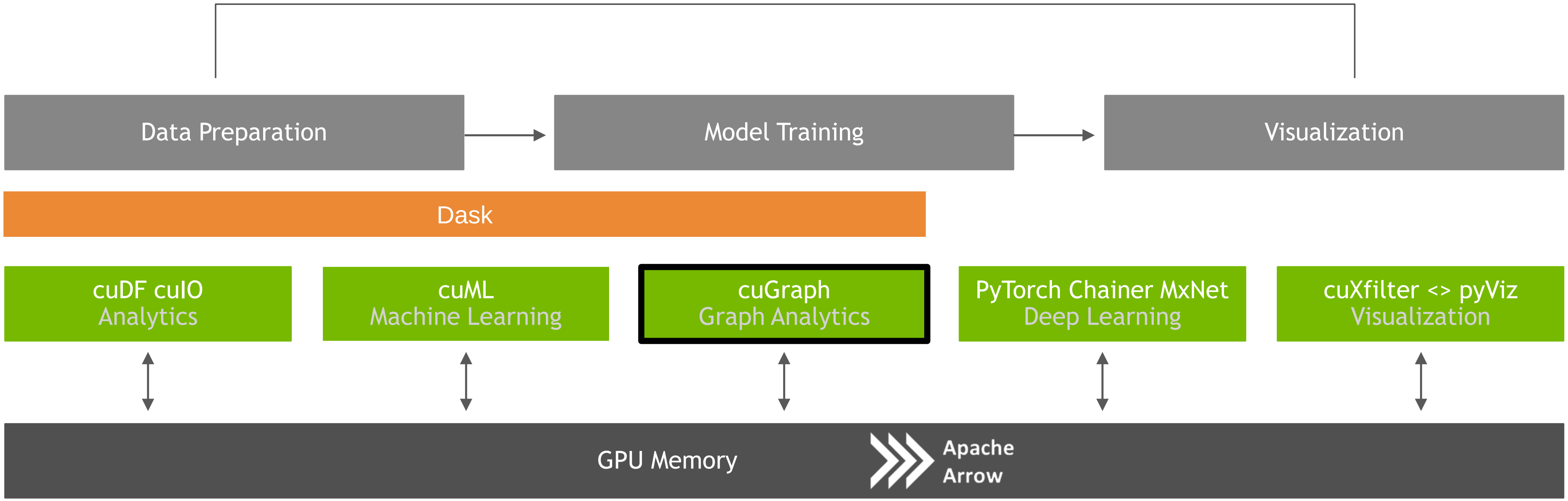
January 2020 - RAPIDS 0.12?

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

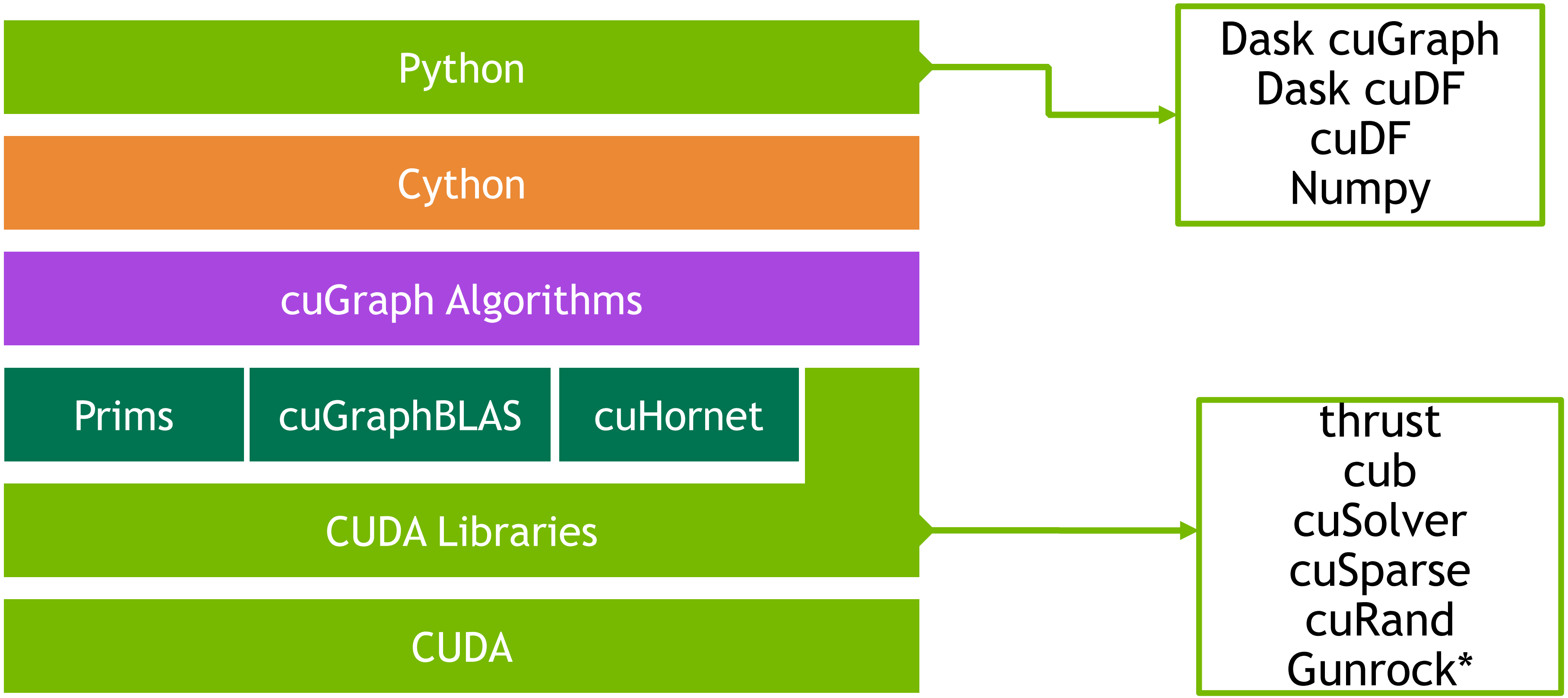


GOALS AND BENEFITS OF CUGRAPH

Focus on Features and User Experience

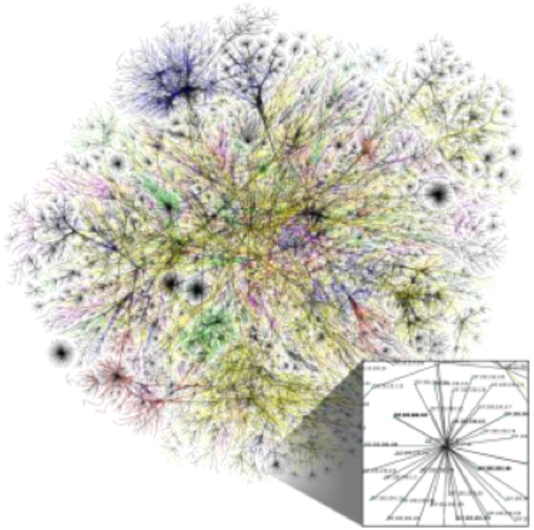
- Breakthrough Performance
  - Up tp 500 million edges on a single 32GB GPU
  - Multi-GPU support for scaling into the billions of edges
- Seamless integration with cuDF and cuML
  - Property Graph support via DataFrames
- Extensive collection of algorithm, primitive, and utility functions
- Multiple APIs:
  - Python: NetworkX-like
  - C/C++: lower-level granular control for application developers

GRAPH TECHNOLOGY STACK



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

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 (Weak and Strong)			
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 (Weak and Strong)			
Louvain			
Spectral Clustering			
InfoMap			
K-Cores			

Getting Started

RAPIDS PREREQUISITES

See more at rapids.ai

- **GPU:** NVIDIA Pascal™ or better with **compute capability** 6.0+
- **Supported OS:** Ubuntu 16.04/18.04 or CentOS 7 with gcc 5.4 & 7.3
- **Docker Prereqs:** Docker CE v18+ and **NVIDIA-docker v2+**
- **CUDA: 9.2 with driver v396.37+** or **10.0 with driver v410.48+**
  - CUDA 10.1 is *not supported in v0.9*, support will be added soon

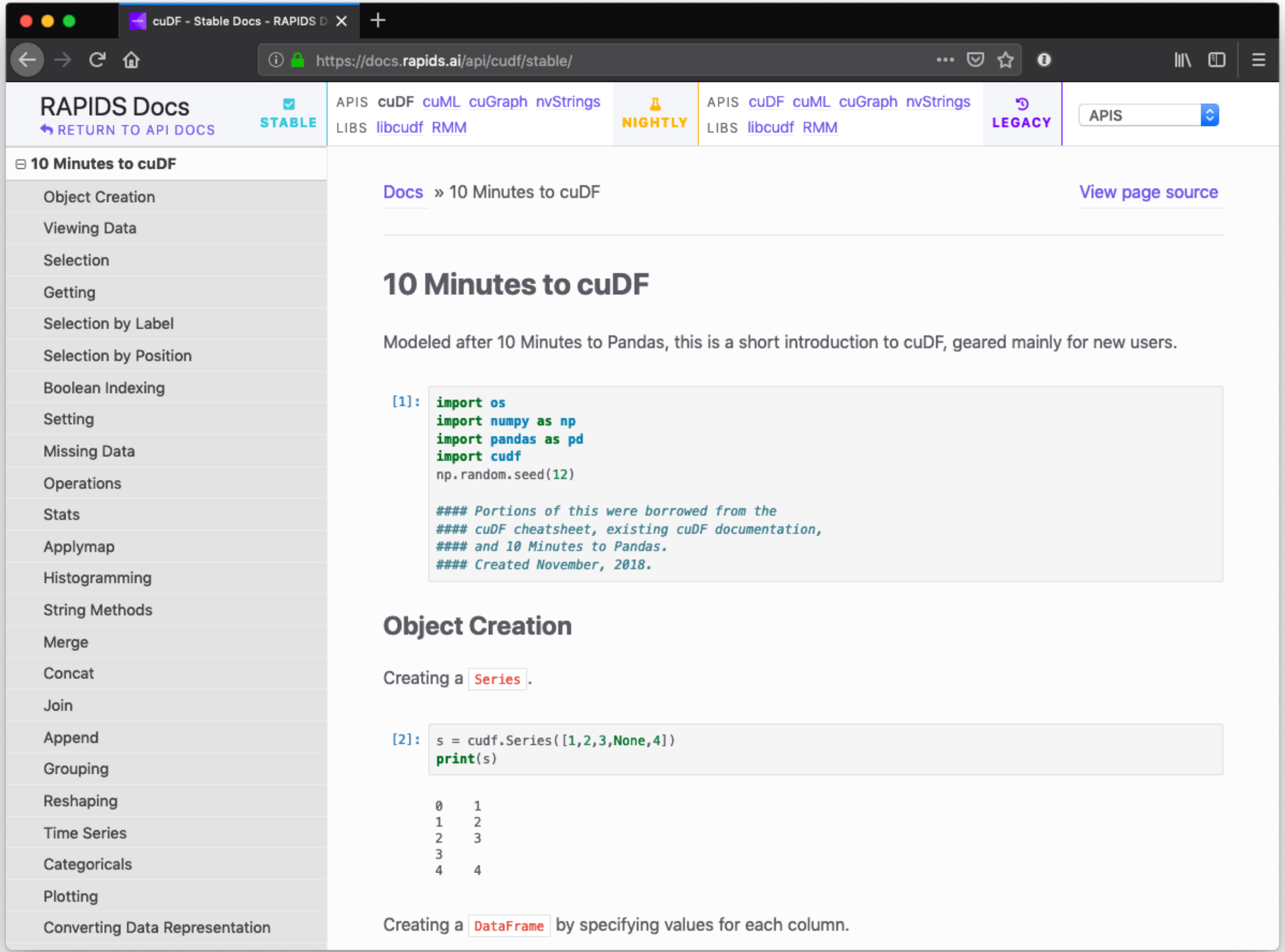
	Preferred		Advanced	
METHOD	Conda	Docker + Examples	Docker + Dev Env	Source
RELEASE	Stable (0.9)		Nightly (0.10a)	
PACKAGES	cuDF	cuML	cuGraph	All Packages
LINUX	Ubuntu 16.04	Ubuntu 18.04	CentOS 7	
PYTHON	Python 3.6		Python 3.7	
CUDA	CUDA 9.2		CUDA 10.0	

COMMAND

```
conda install -c rapidsai -c nvidia -c numba -c conda-forge -c anaconda \
    cudf=0.9 cuml=0.9 cugraph=0.9 python=3.6 anaconda::cudatoolkit=9.2
```

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/

NVIDIA GPU CLOUD

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

COMMUNITY

ECOSYSTEM PARTNERS

CONTRIBUTORS

ANACONDA · BLAZINGDB · UNROCK · NVIDIA · QUANSIGHT · Walmart

ADOPTERS

Chainer · databricks · graphistry · H2O.ai · IBM · iguazio · Inria

MAPR · omni sci · PyTorch · Uber · URSA LABS

OPEN SOURCE

Apache Arrow · DASK · GOAi · Numba · scikit learn · dmlc XGBoost

BUILDING ON TOP OF RAPIDS

A bigger, better, stronger ecosystem for all

nuclio

blazingSQL

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

Google Cloud

Kubeflow

Azure

Cloud Dataproc

Azure Machine Learning

RAPIDS

Open GPU Data Science

Baidu 百度

Alibaba Cloud

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

USEFUL LINKS

Developer Zone

- **https://developer.nvidia.com/**
- https://devtalk.nvidia.com/

RAPIDS Blogs Page

https://medium.com/rapids-ai

RAPIDS GITHUB Page

https://medium.com/rapids-ai

RAPIDS Docs

https://docs.rapids.ai/

Contact Presenter@

Mail ID : mitrar@nvidia.com
LinkedIn : https://www.linkedin.com/in/mitrabhanurath

NVIDIA®