RAPIDS

GPU-Accelerated Data Science

USENIX OpML 2020

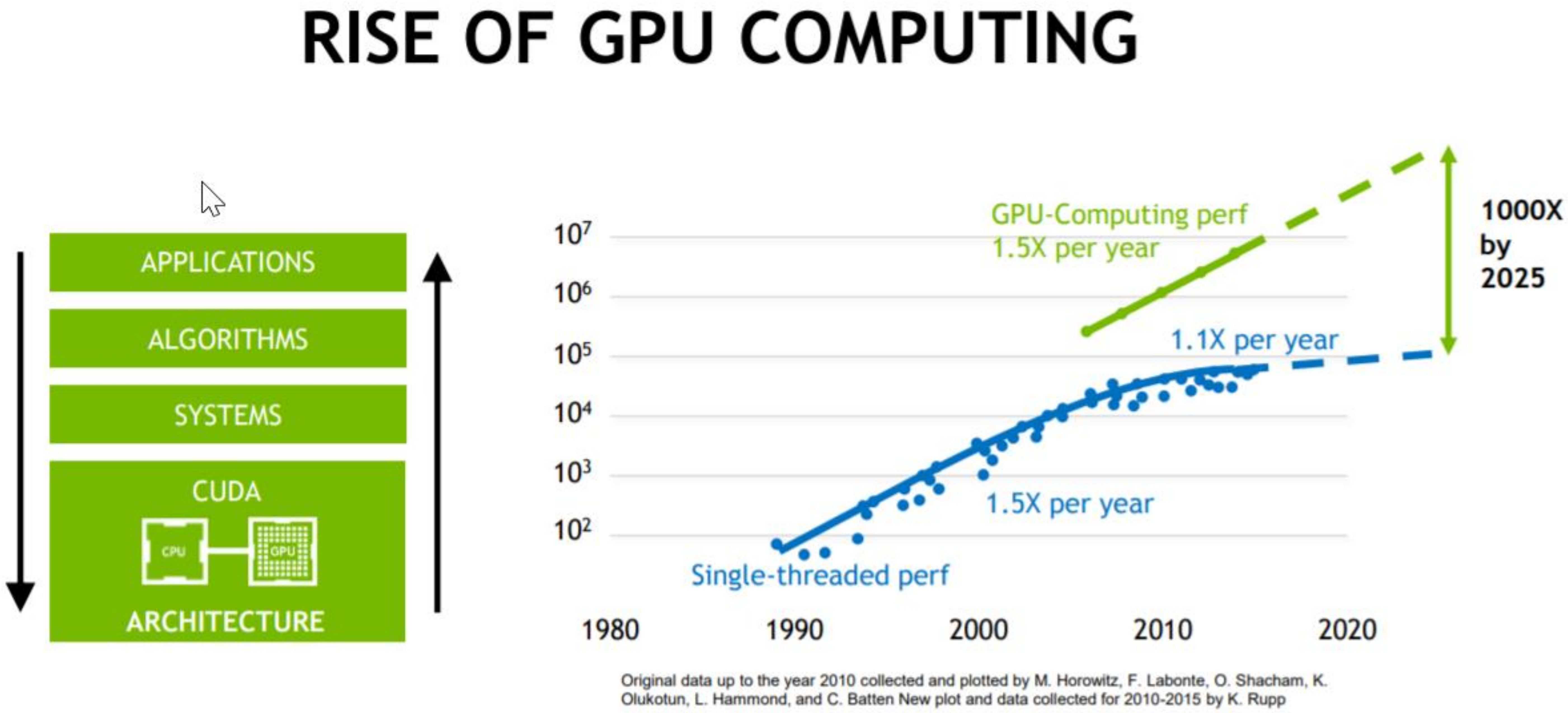
John Zedlewski (jzedlewski@nvidia.com)

Roadmap

- Why GPU acceleration for data science?
- What is the RAPIDS stack?
- Scaling with Dask, UCX, and Infiniband
- Benchmarking
- How to get started?

Why GPU-accelerated data science?

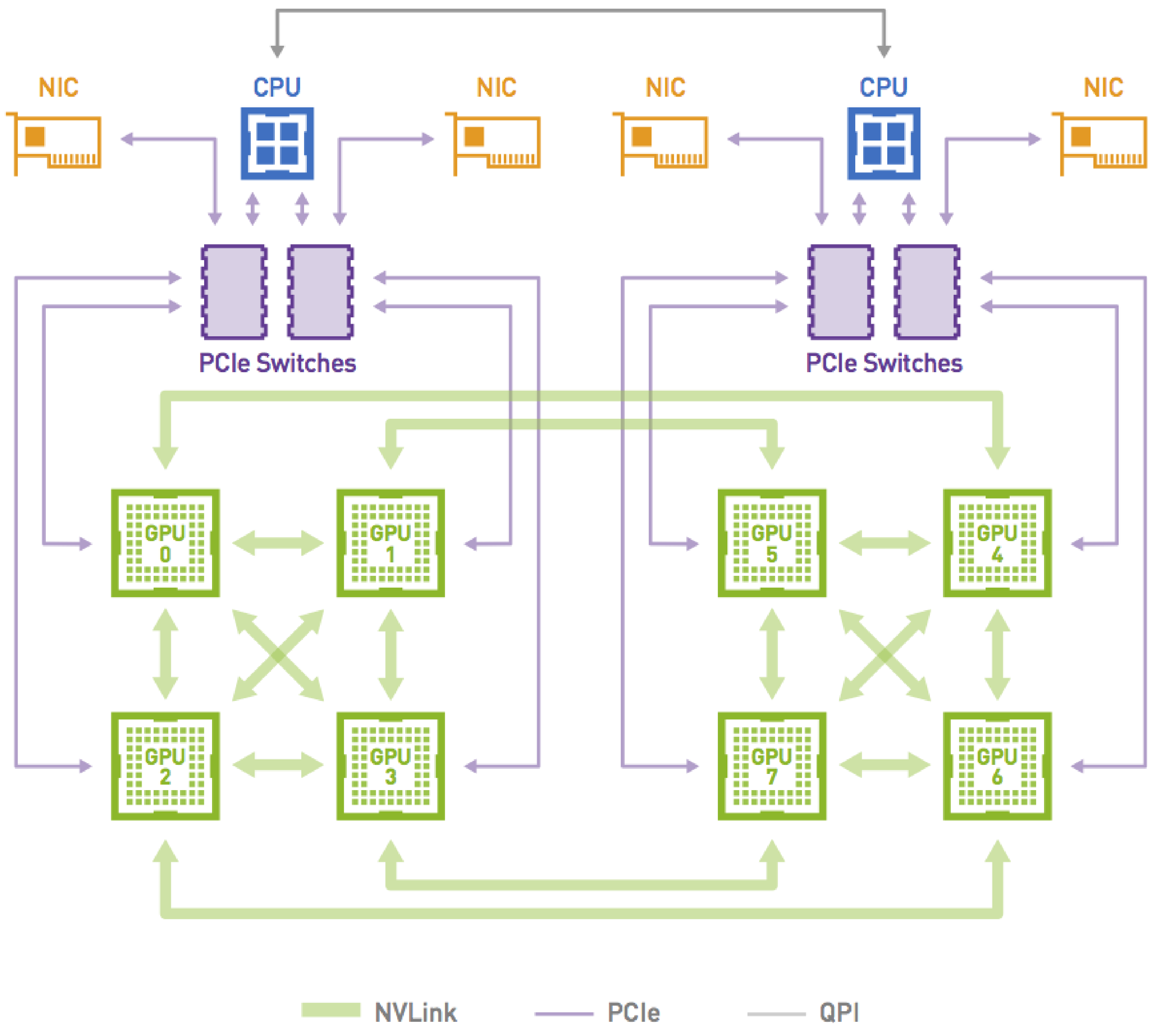
Performance gap between GPU and CPU is growing



Why GPUs?

Numerous hardware advantages

- Thousands of cores with up to ~20 TeraFlops of general purpose compute performance
- Up to 1.6 TB/s of memory bandwidth
- Hardware interconnects for up to 600 GB/s bidirectional GPU <--> GPU bandwidth
- Can scale up to 16x GPUs in a single node



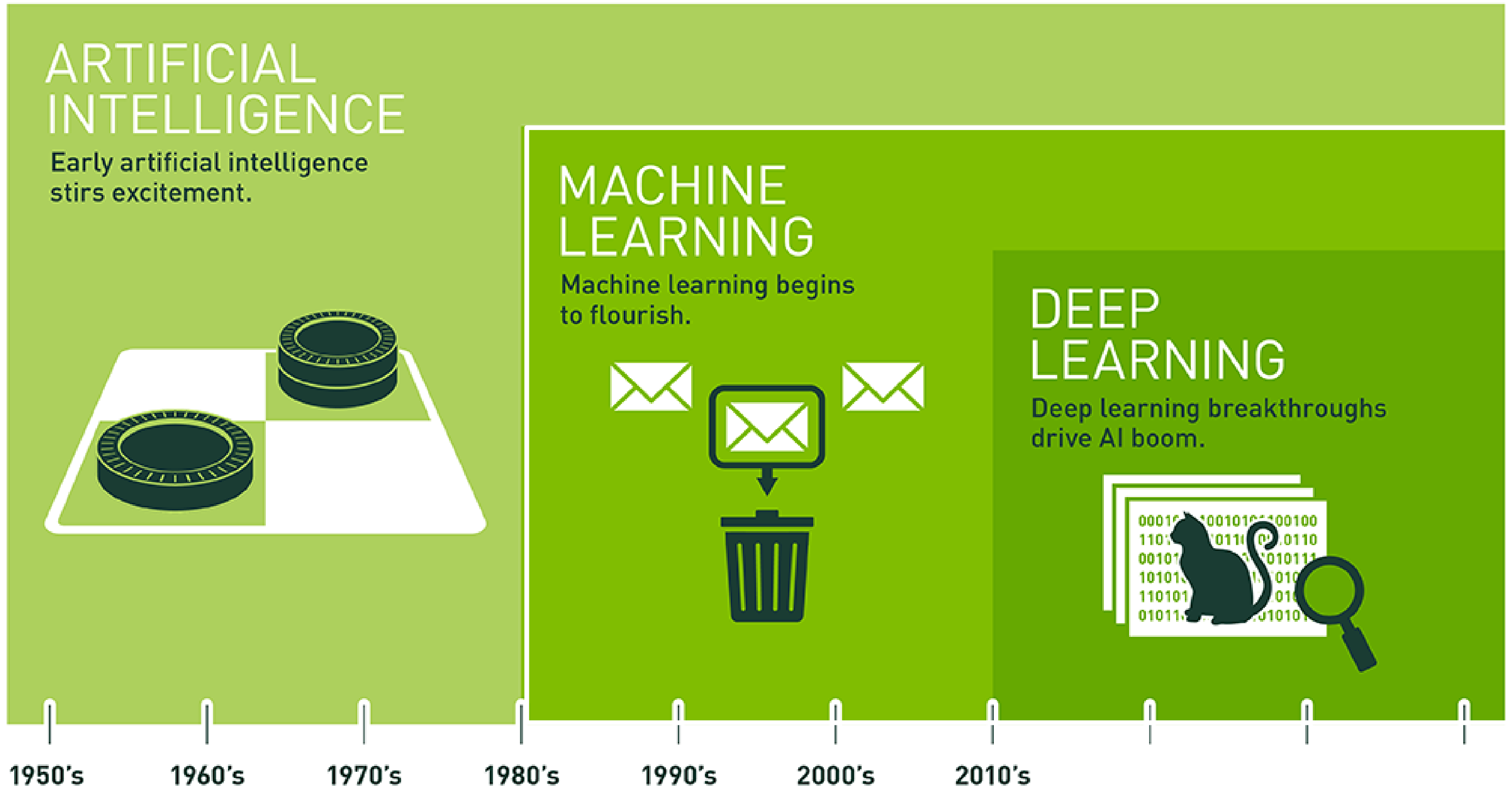
Almost never run out of compute relative to memory bandwidth!

But PCIe bandwidth has not scaled at the same rate

Machine Learning and AI Go Beyond Deep Learning

From Counting to Actions @ Scale

- GPUs are ubiquitous in Deep Learning
- The same matrix operations allow GPUs to be very performant at Machine Learning also
- Regressions, Clustering, Decision Trees, Dimensionality Reduction, etc…
- >90% of Enterprise are primarily using Machine Learning
- Those using Deep Learning often combine with traditional Machine Learning – preprocessing, filtering, clustering, etc.
- ETL is a bottleneck for traditional ML and DL alike



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.

Python and Spark

Speeding up data science requires working *with* the huge PyData community, not trying to replace it

PyData

pandas

spaCy

Numba

scikit learn

DASK

PyTorch

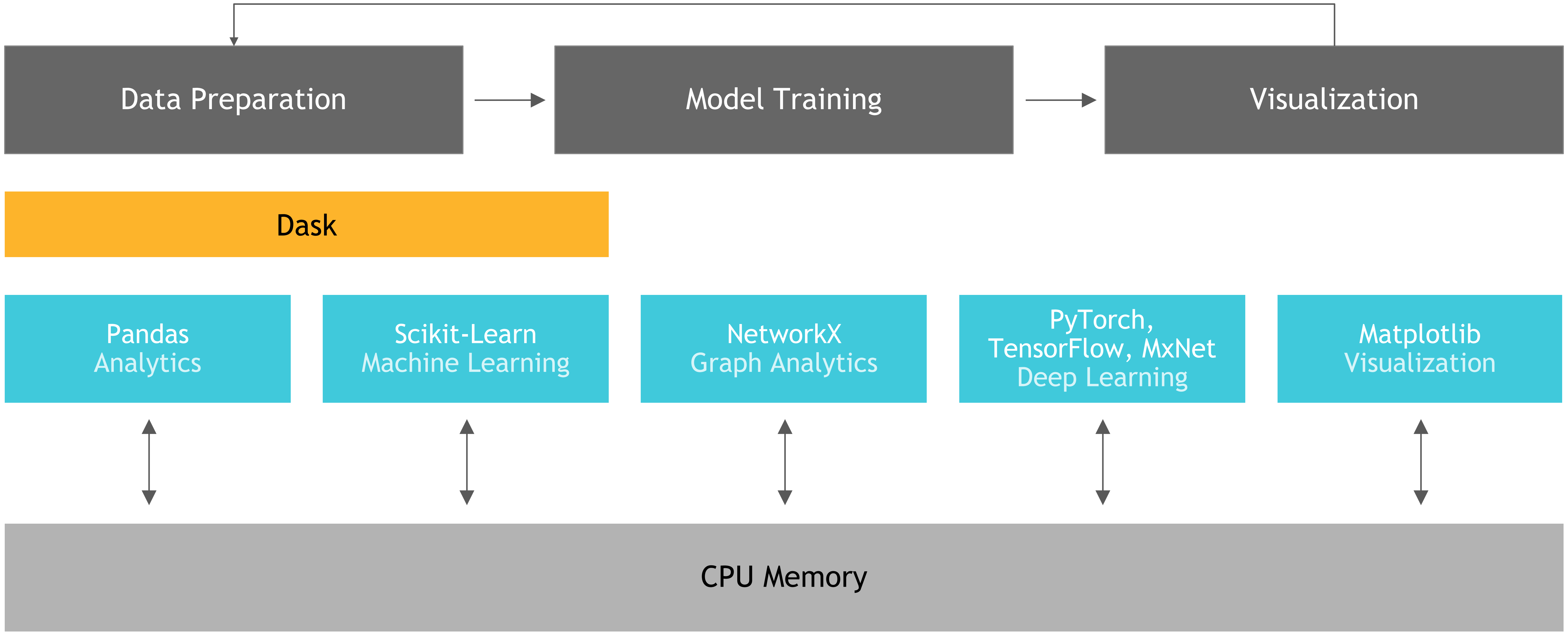
NetworkX

NumPy

What is RAPIDS?

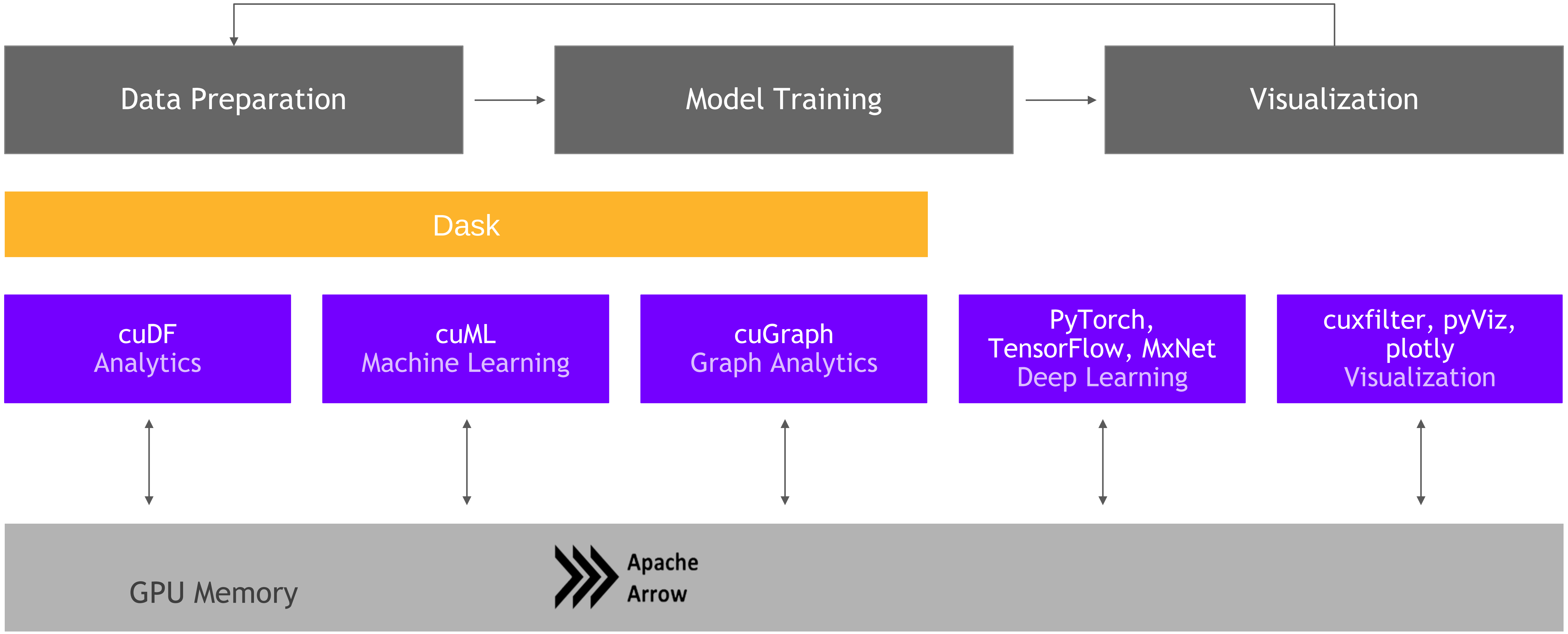
Open Source PyData Ecosystem

Familiar Python APIs



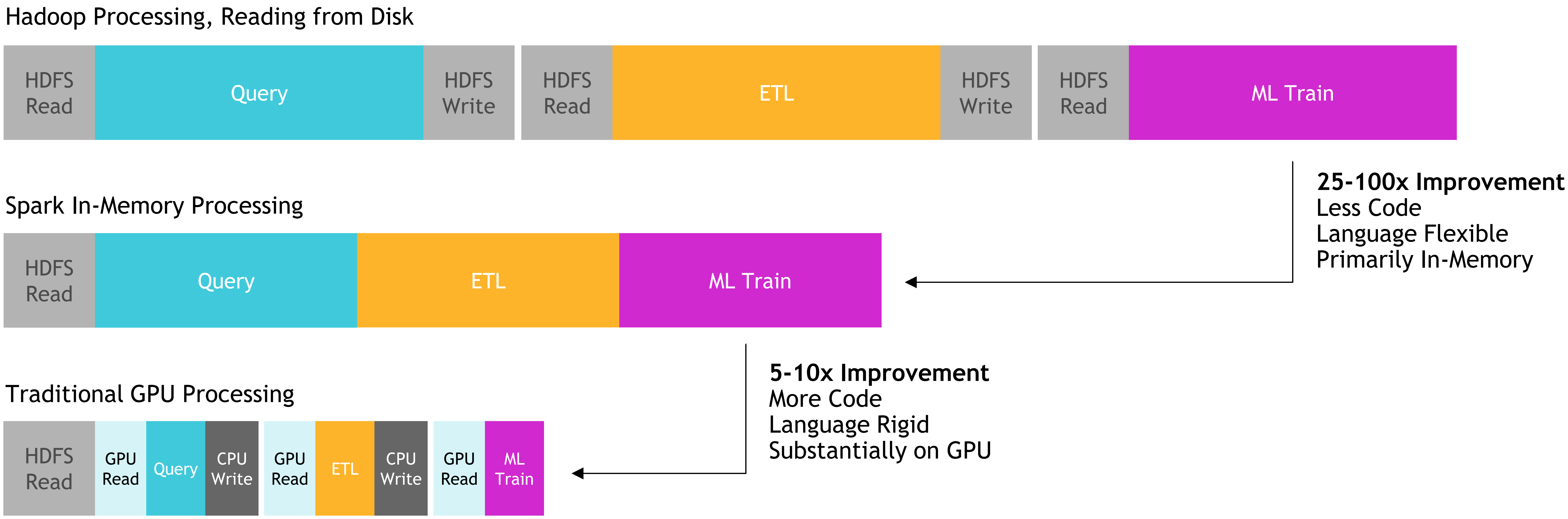
RAPIDS

End-to-End, Open Source Accelerated GPU Data Science



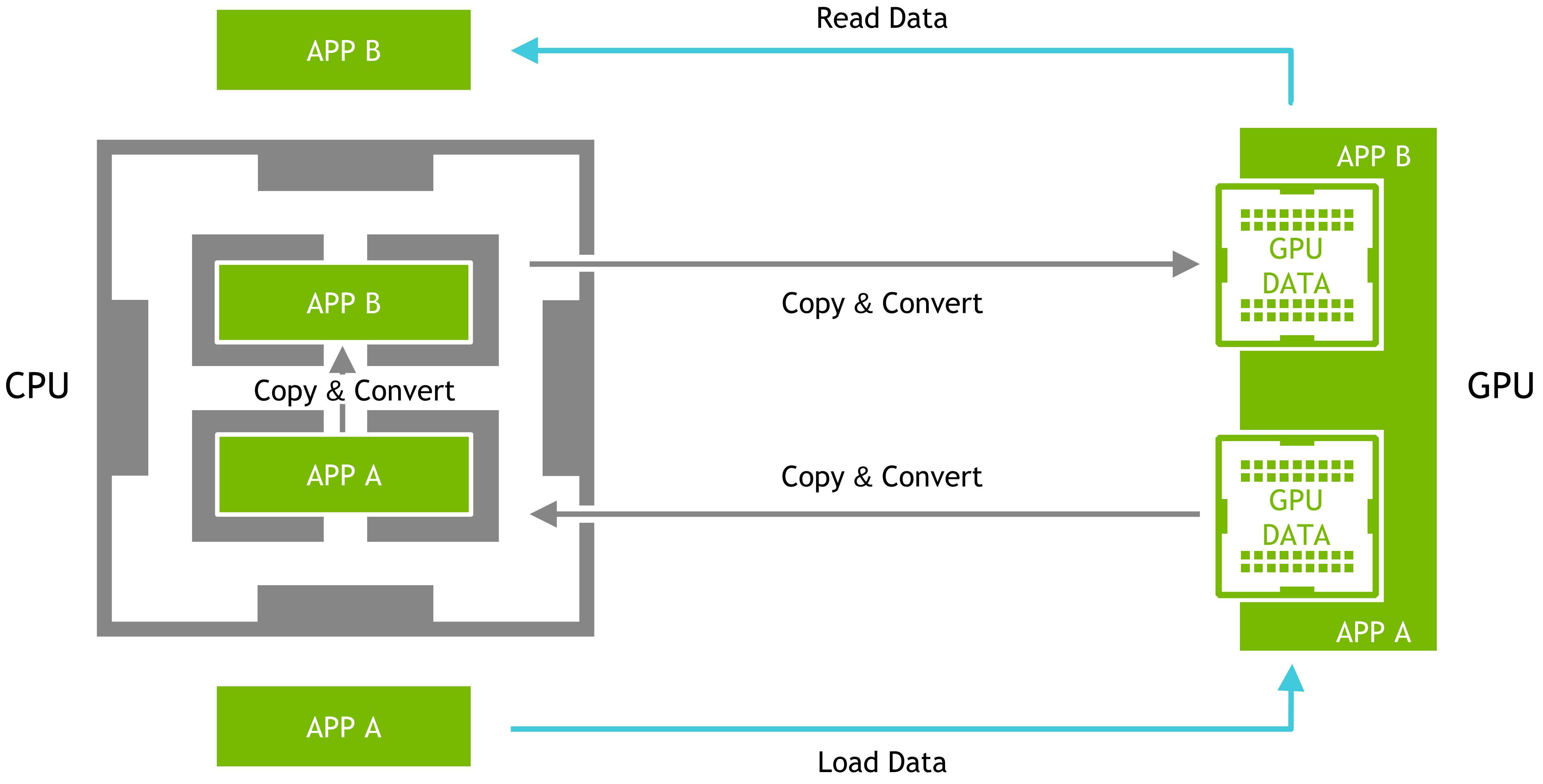
Data Processing Evolution

Faster Data Access, Less Data Movement



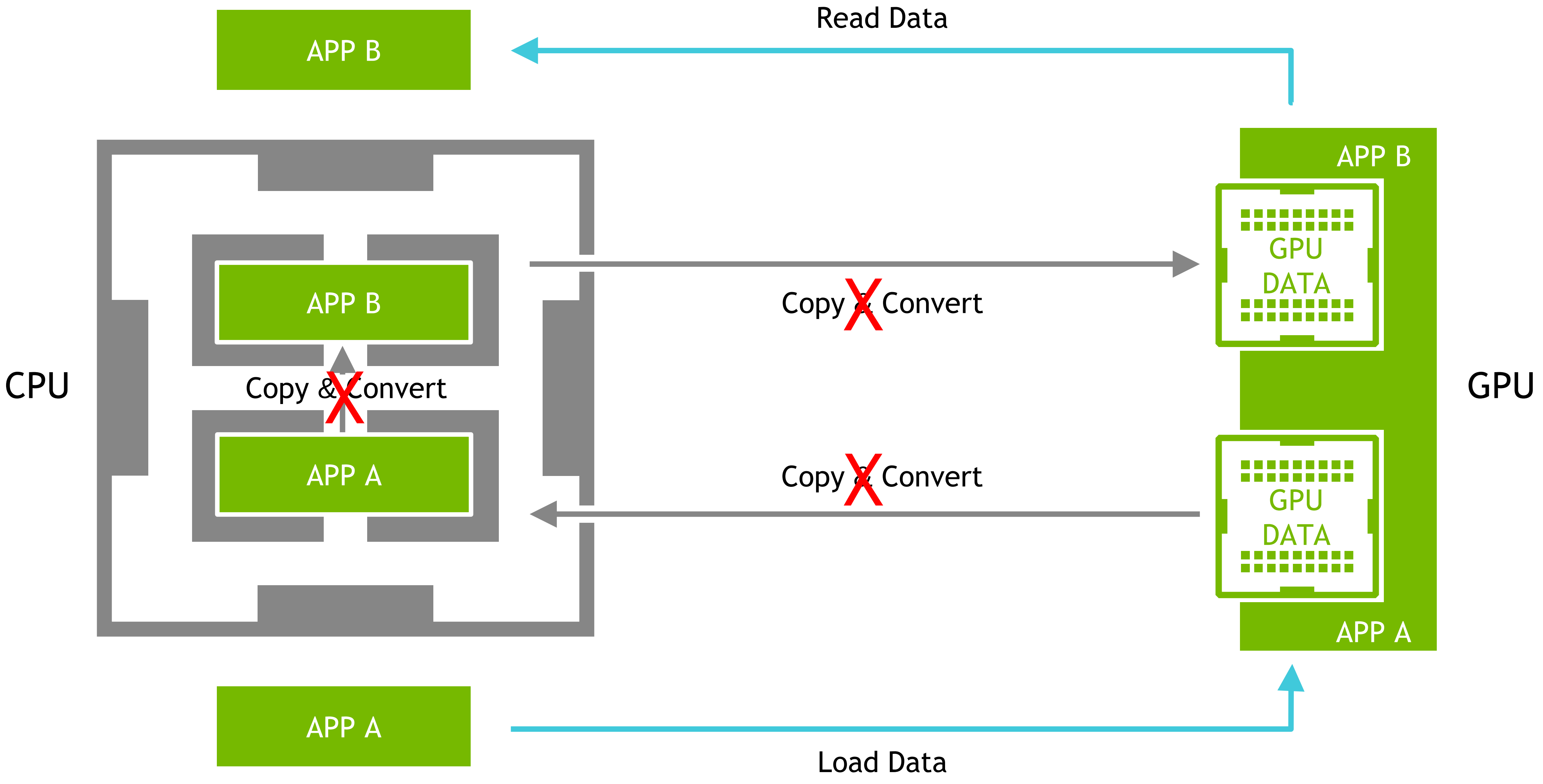
Data Movement and Transformation

The Bane of Productivity and Performance

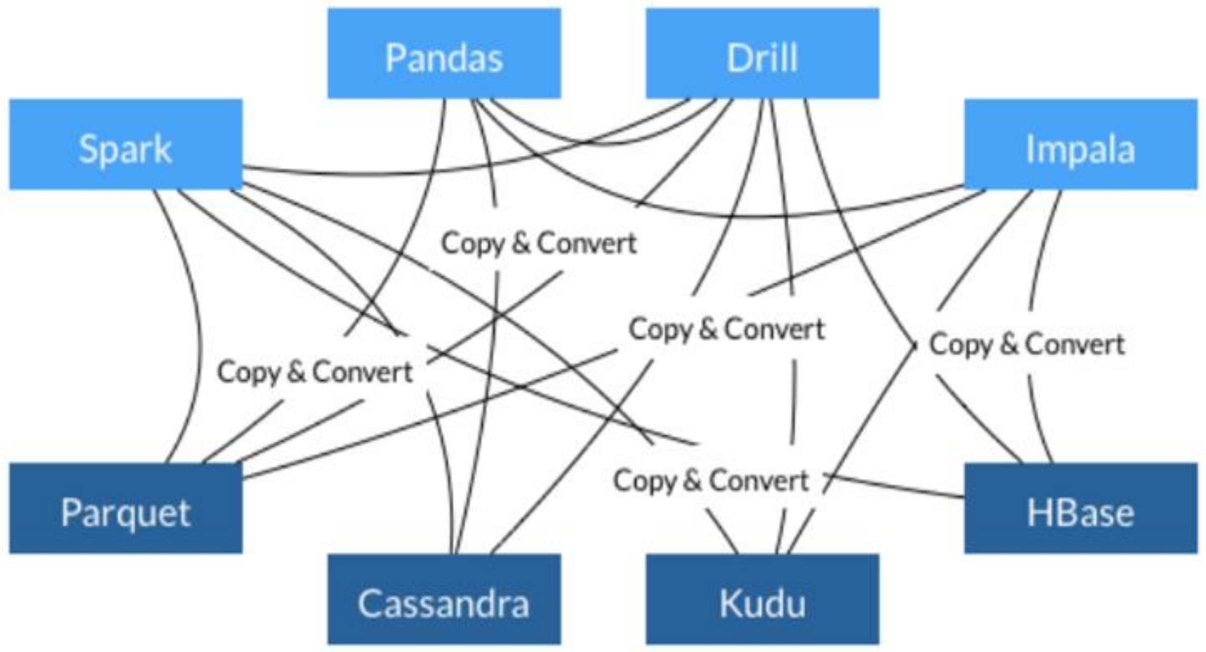


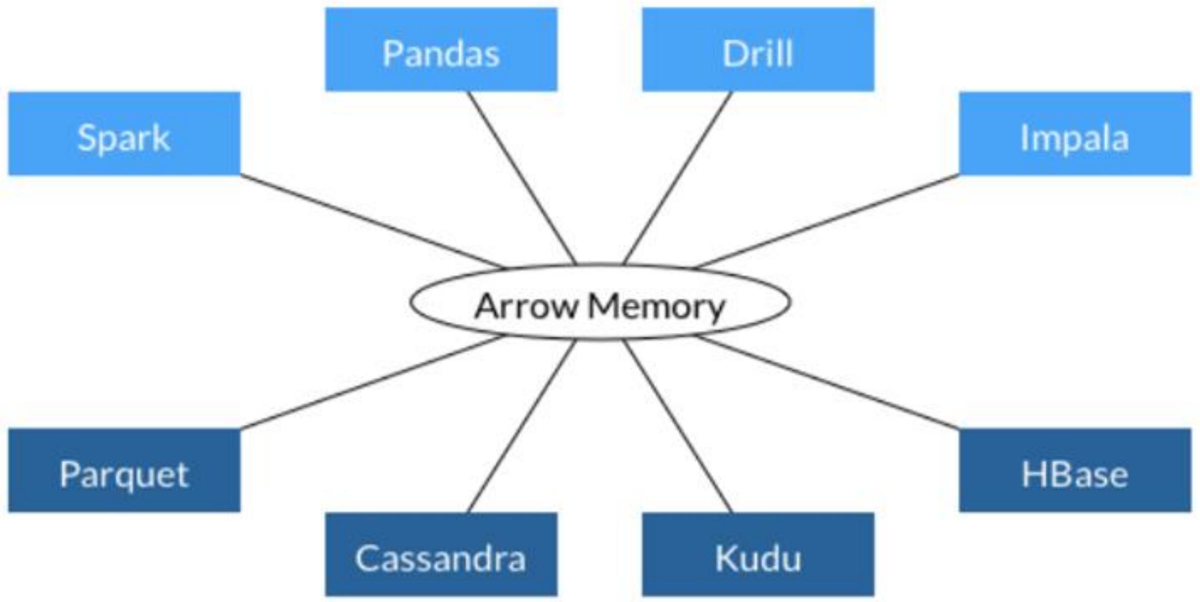
Data Movement and Transformation

What if We Could Keep Data on the GPU?



Learning from Apache Arrow





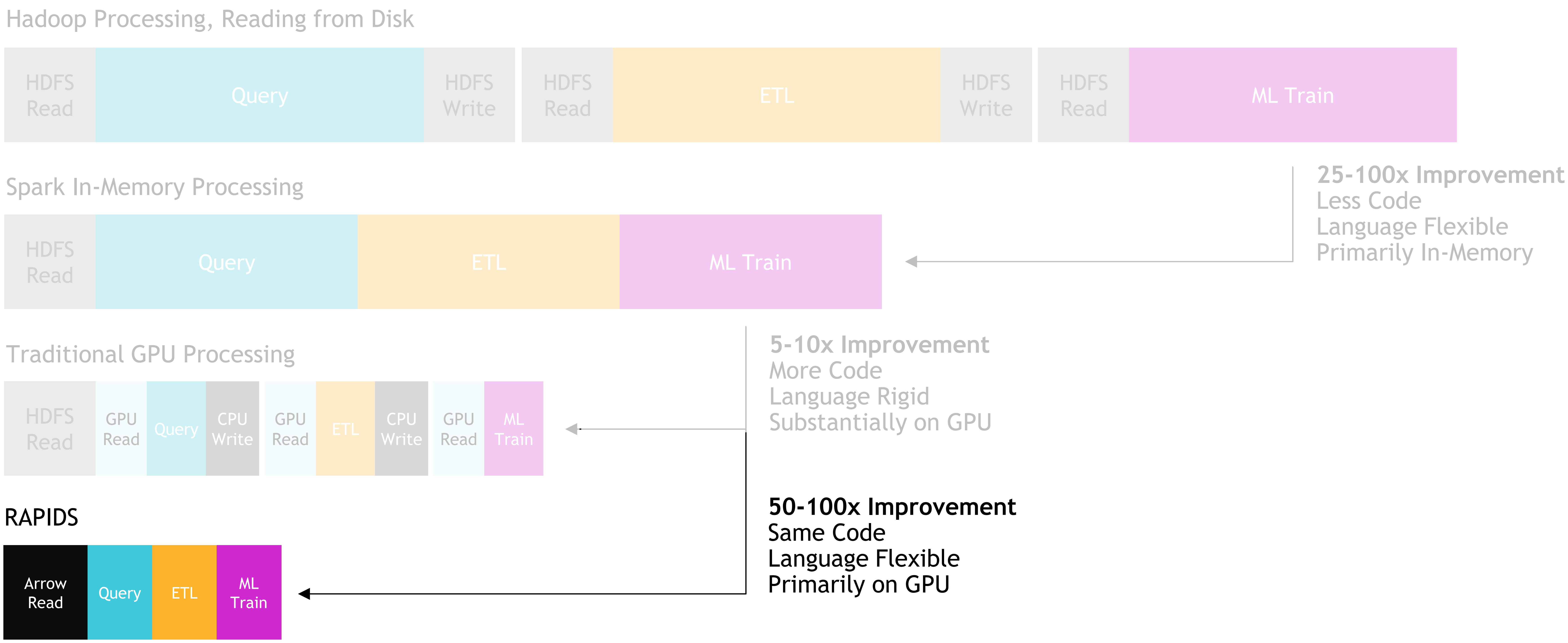
- Each system has its own internal memory format
- 70-80% computation wasted on serialization and deserialization
- Similar functionality implemented in multiple projects

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

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

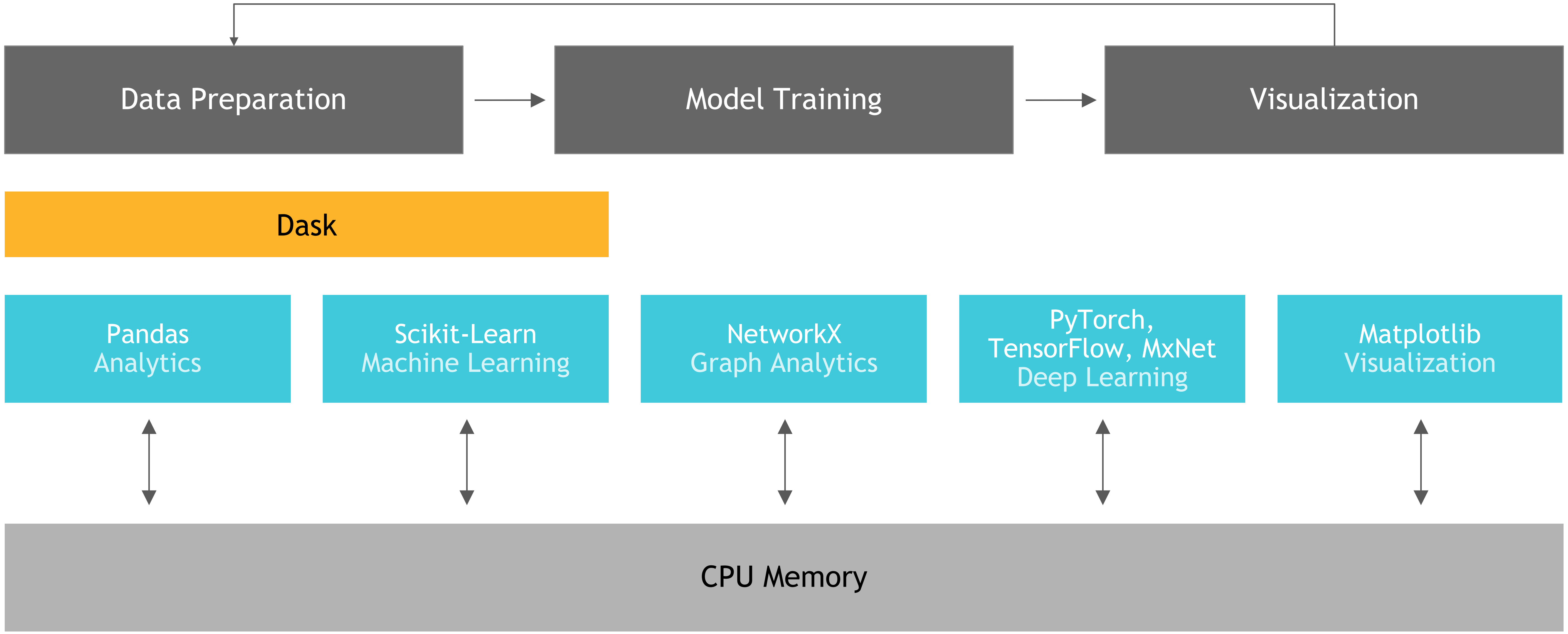
Data Processing Evolution

Faster Data Access, Less Data Movement



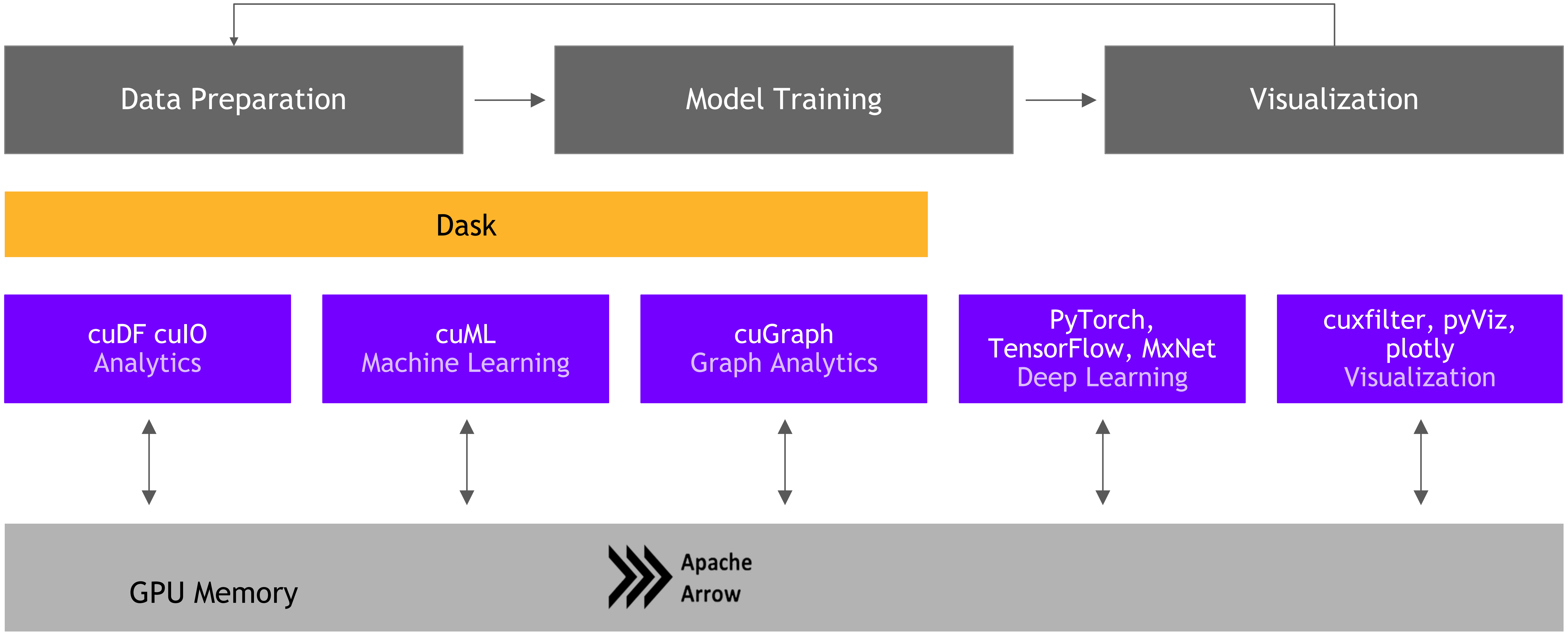
Open Source Data Science Ecosystem

Familiar Python APIs



RAPIDS

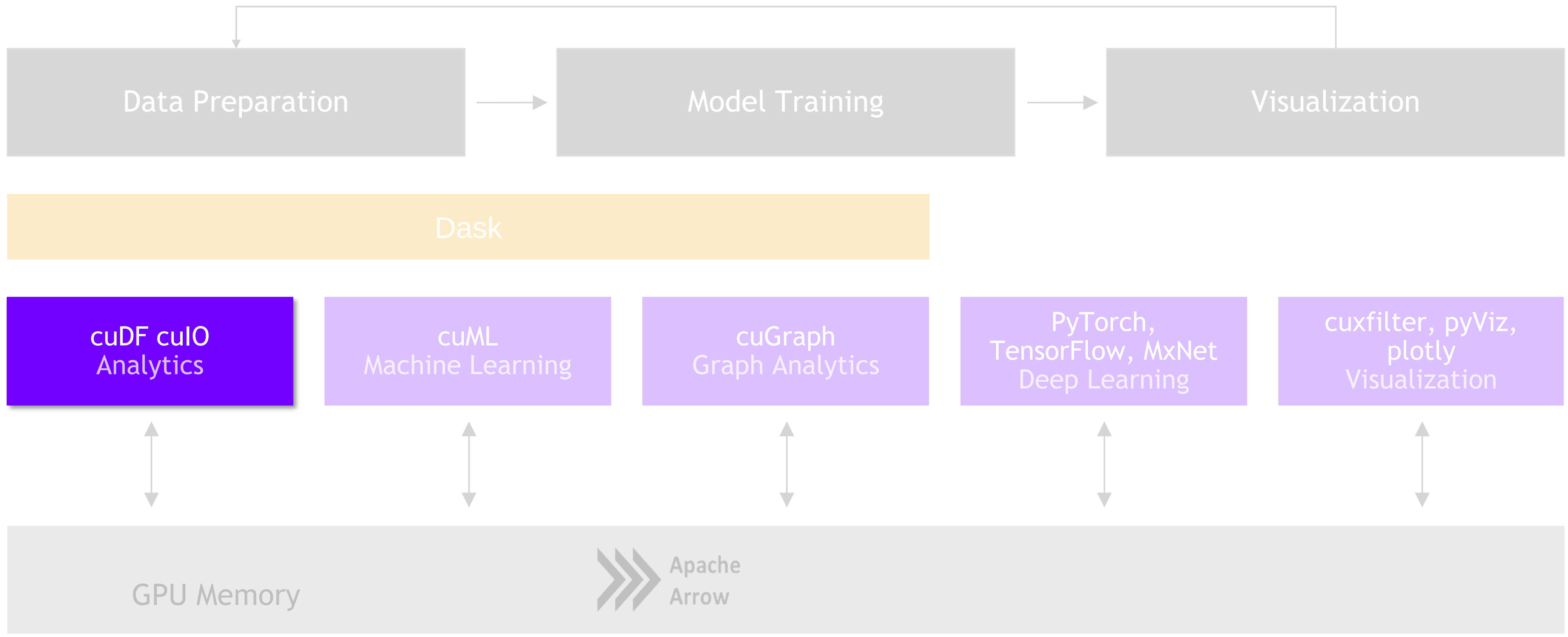
End-to-End Accelerated GPU Data Science



cuDF

RAPIDS

GPU Accelerated Data Wrangling and Feature Engineering



ETL - the Backbone of Data Science

cuDF is…

PYTHON LIBRARY

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

In [2]:
```
#Read in the data. Notice how it decompresses as it reads the data into memory.
gdf = cudf.read_csv('/rapids/Data/black-friday.zip')
```

In [3]:
```
#Taking a look at the data. We use "to_pandas()" to get the pretty printing.
gdf.head().to_pandas()
```

Out[3]:

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

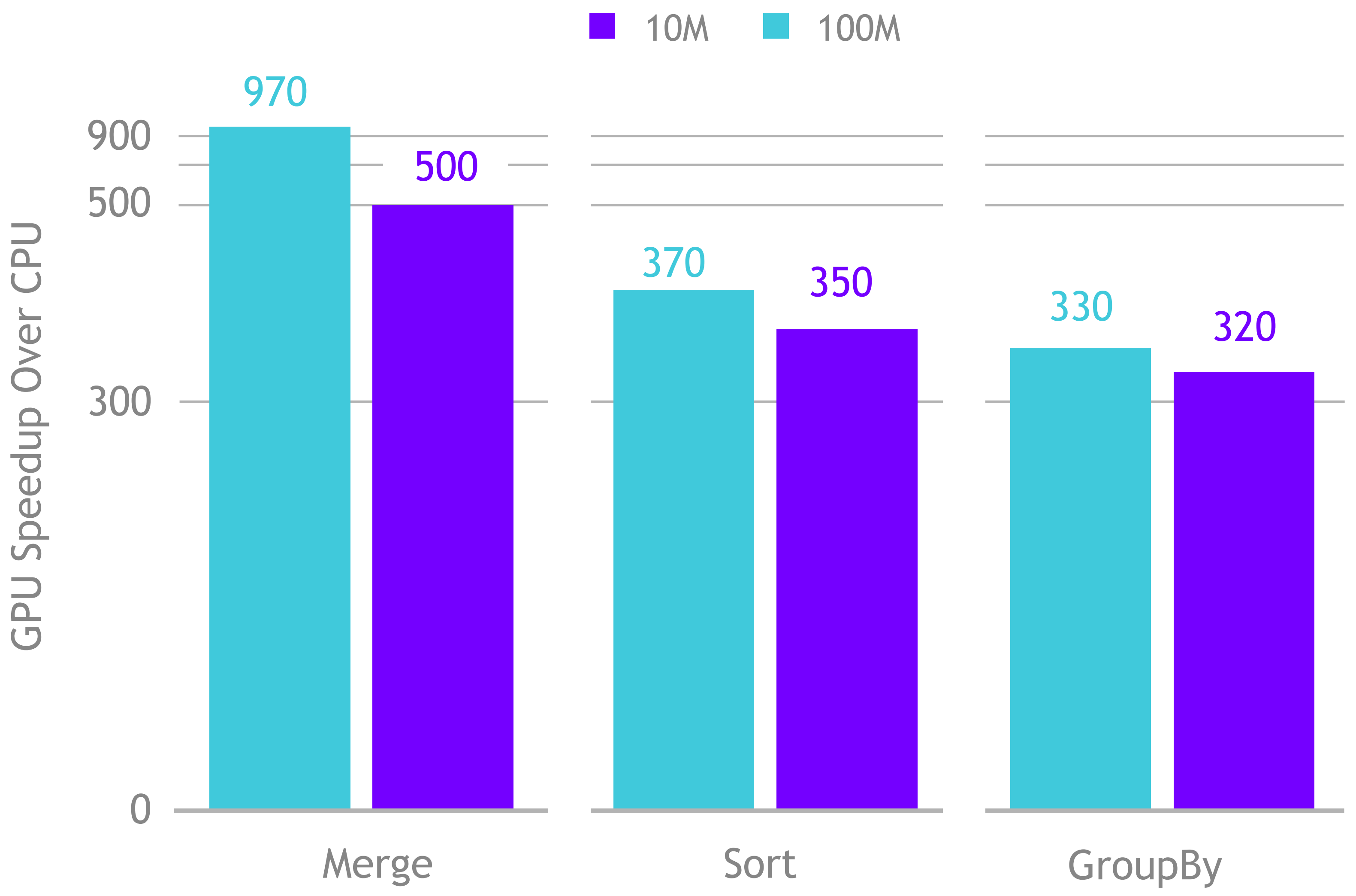
In [6]:
```
#grabbing the first character of the years in city string to get rid of plus sign, and converting
 to int
gdf['city_years'] = gdf.Stay_In_Current_City_Years.str.get(0).stoi()
```

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

Benchmarks: Single-GPU Speedup vs. Pandas

cuDF v0.13, Pandas 0.25.3

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



Extraction is the Cornerstone

cuIO for Faster Data Loading

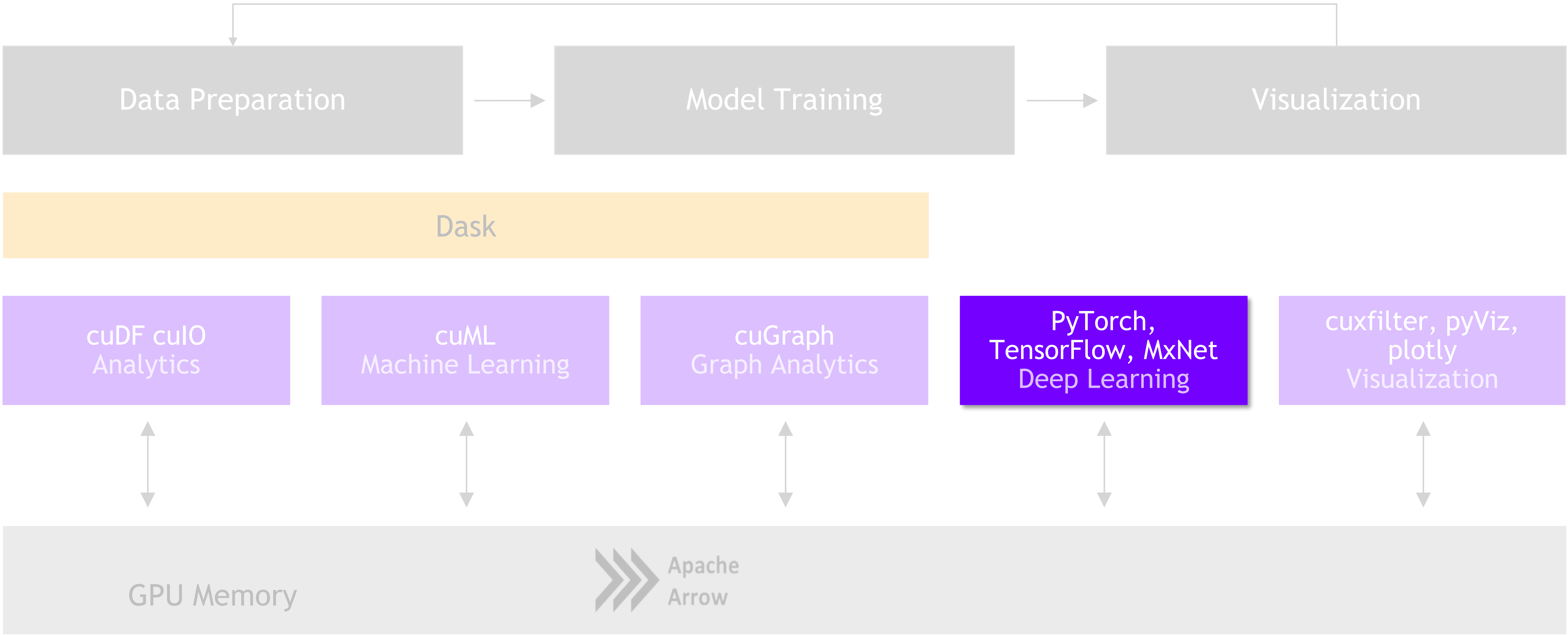
- Follow Pandas APIs and provide >10x speedup
- CSV Reader - v0.2, CSV Writer v0.8
- Parquet Reader – v0.7, Parquet Writer v0.12
- ORC Reader – v0.7, ORC Writer v0.10
- JSON Reader - v0.8
- Avro Reader - v0.9
- GPU Direct Storage integration in progress for bypassing PCIe bottlenecks!
- Key is GPU-accelerating both parsing and decompression

```
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

RAPIDS

Building Bridges into the Array Ecosystem



Interoperability for the Win

DLPack and __cuda_array_interface__

Chainer

CuPy

mpi4py

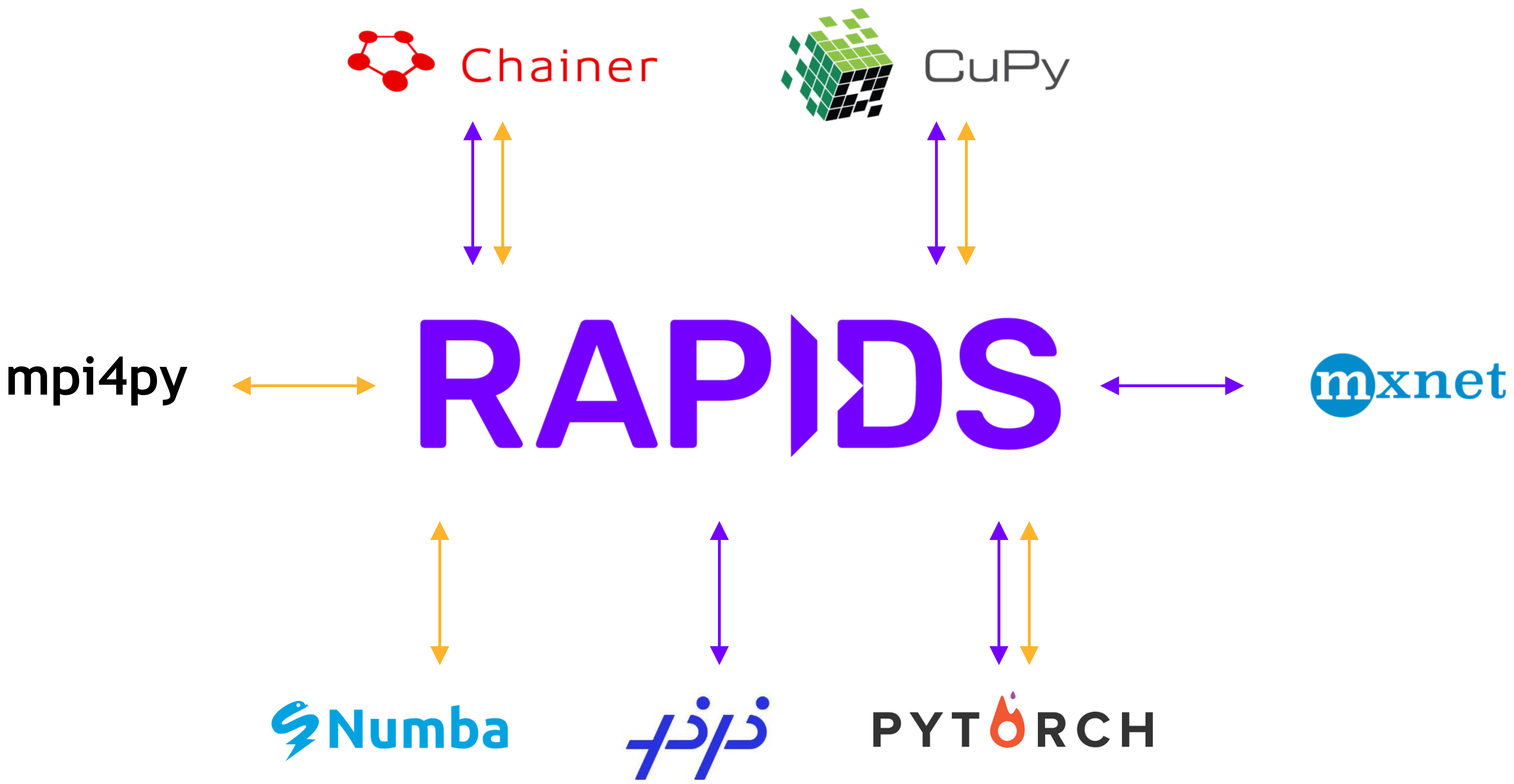
mxnet

Numba

PYTORCH

Interoperability for the Win

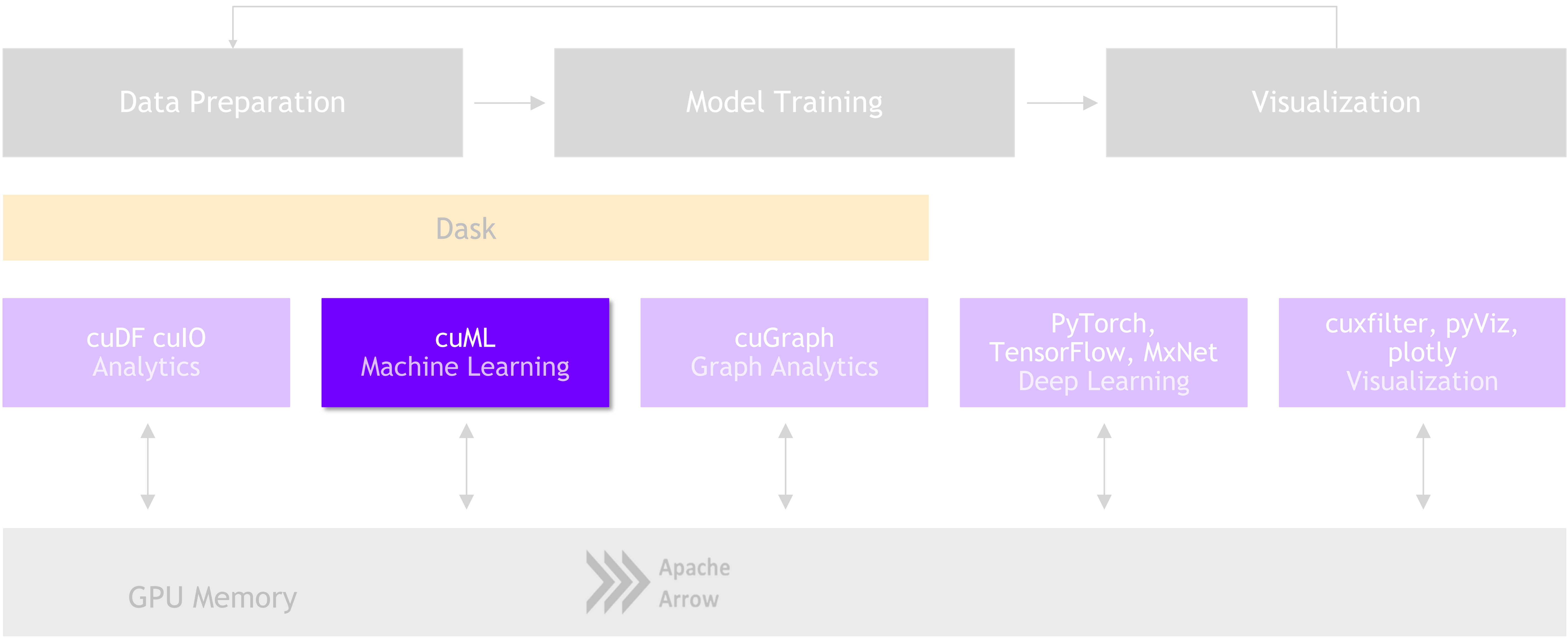
DLPack and __cuda_array_interface__



cuML

Machine Learning

More Models More Problems



ML Technology Stack

- Python
- Cython
- cuML Algorithms
- cuML Prims & RAFT
- CUDA Libraries
- CUDA

Dask cuML
Dask cuDF
cuDF
Numpy

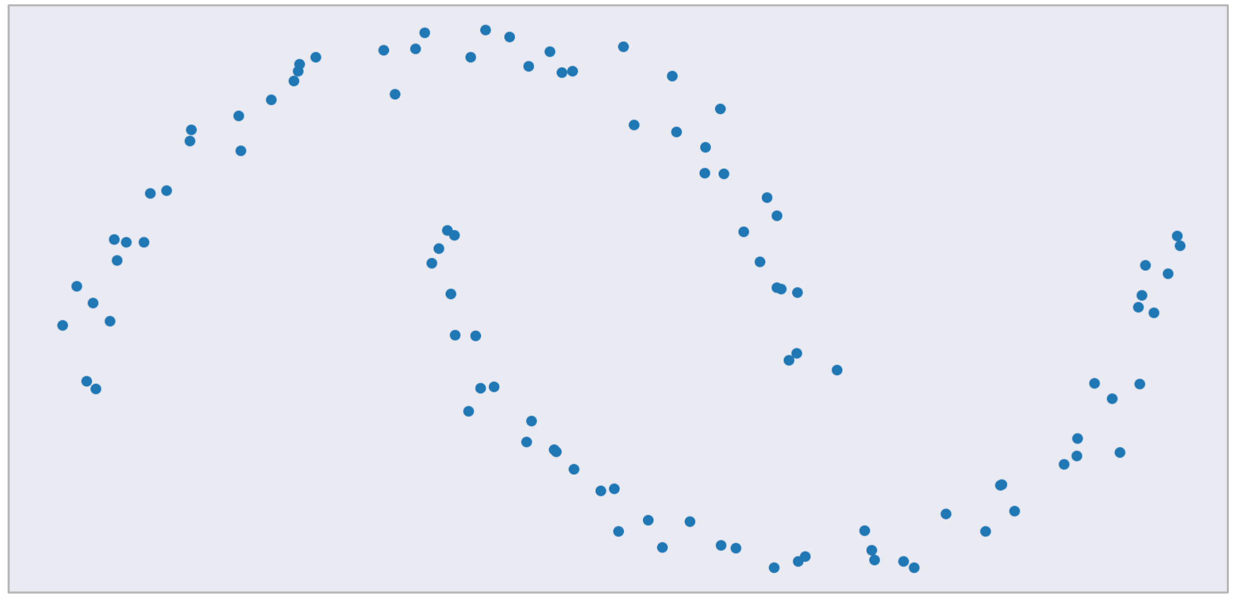
Thrust
Cub
cuSolver
nvGraph
CUTLASS
cuSparse
cuRand
cuBlas

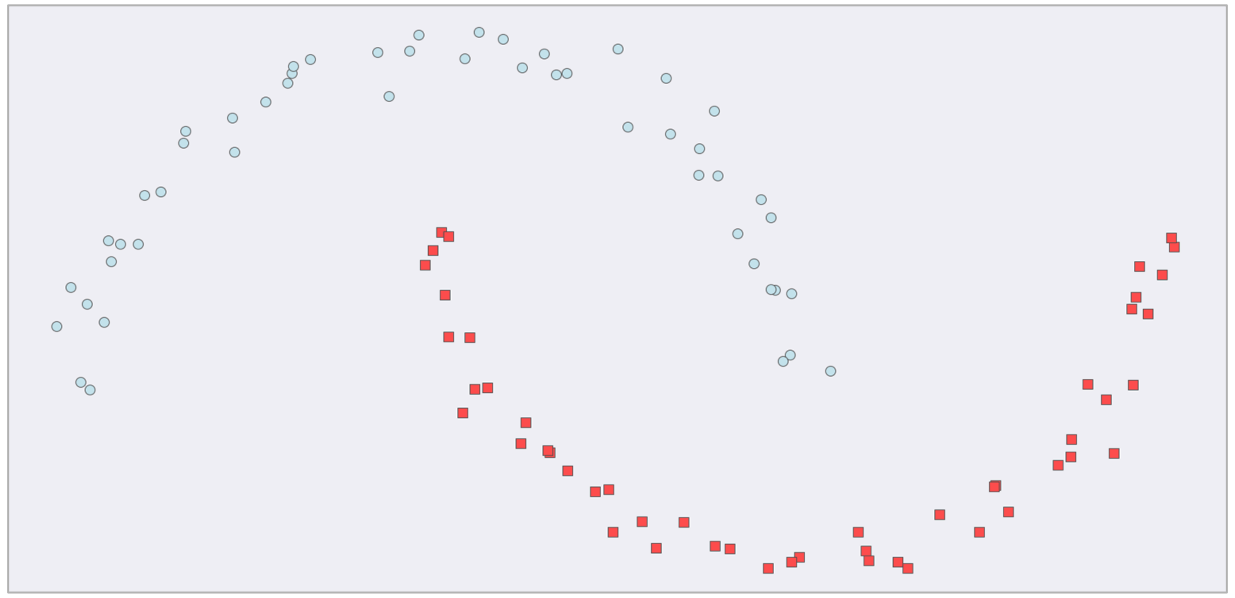
RAPIDS Matches Common Python APIs

CPU-based Clustering

```
from sklearn.datasets import make_moons
import pandas


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

X = pandas.DataFrame({'fea%d'%i: X[:, i]
              for i in range(X.shape[1])})
```

```
from sklearn.cluster import DBSCAN
dbscan = DBSCAN(eps = 0.3, min_samples = 5)

dbscan.fit(X)

y_hat = dbscan.predict(X)
```



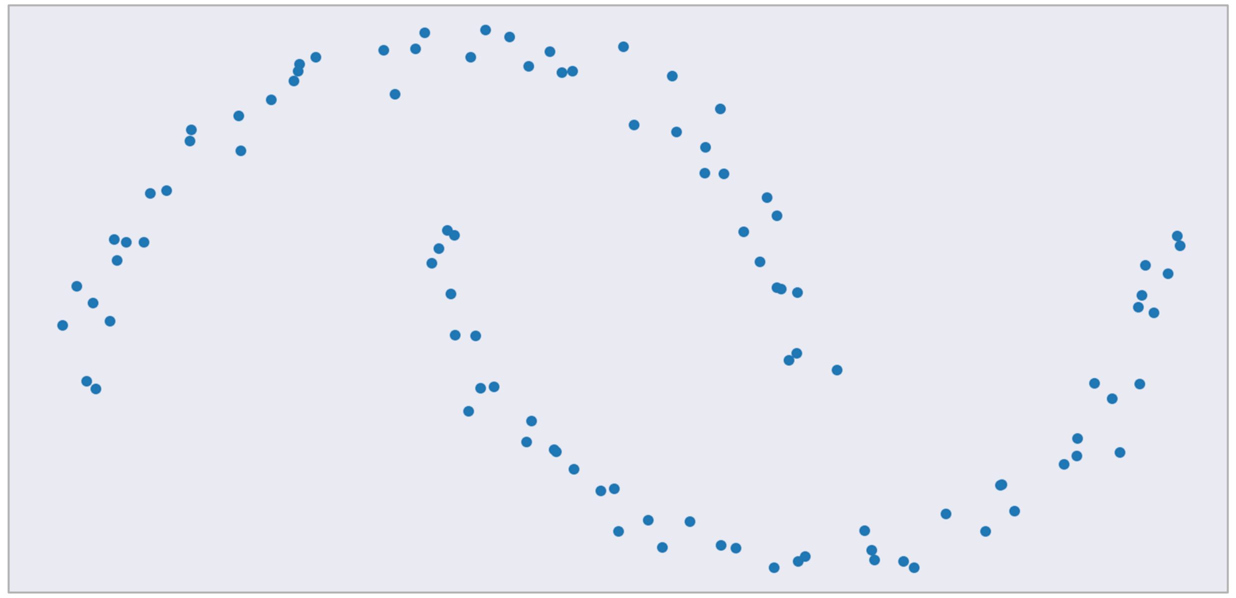


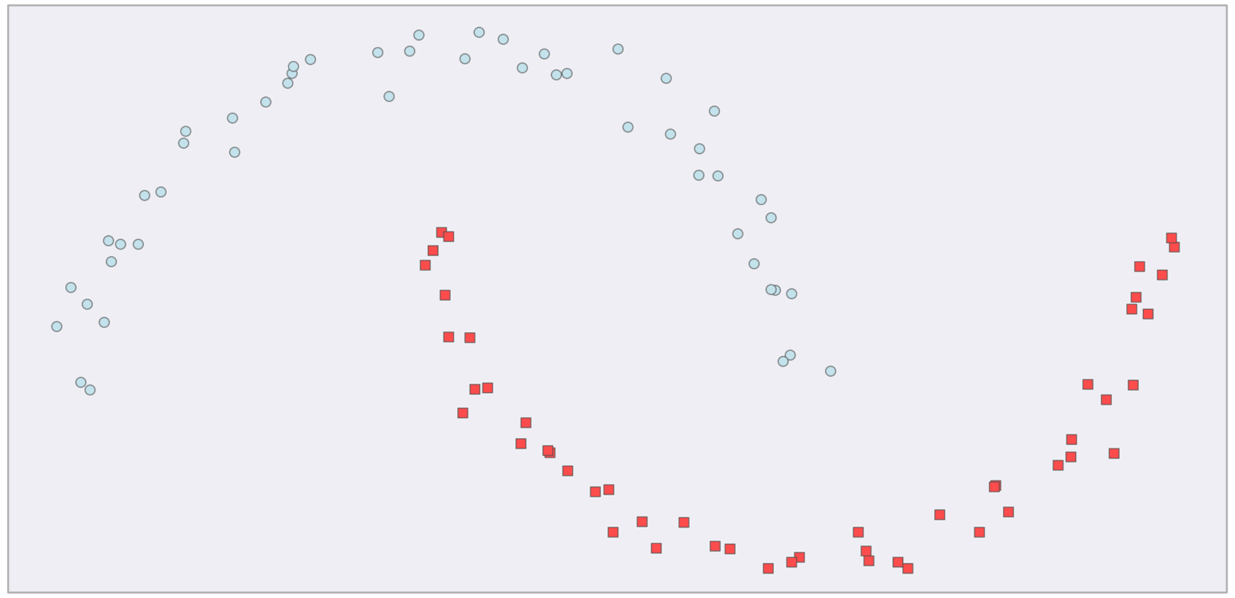
RAPIDS Matches Common Python APIs

GPU-accelerated Clustering

```
from sklearn.datasets import make_moons
import cudf


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

X = cudf.DataFrame({'fea%d'%i: X[:, i]
                  for i in range(X.shape[1])})
```

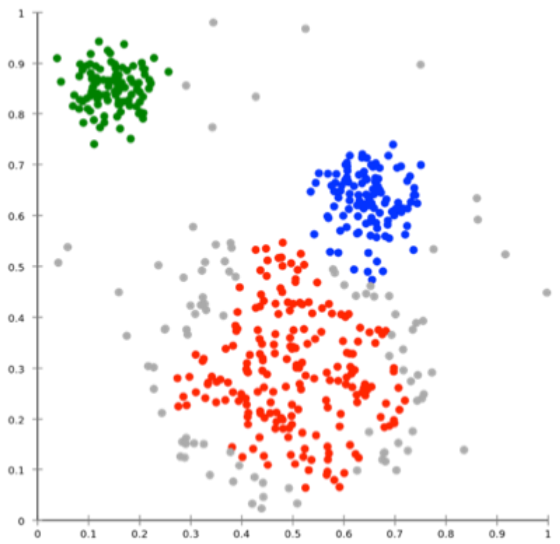
```
from cuml import DBSCAN
dbscan = DBSCAN(eps = 0.3, min_samples = 5)

dbscan.fit(X)

y_hat = dbscan.predict(X)
```





Algorithms

GPU-accelerated Scikit-Learn



Classification \ Regression	Decision Trees / Random Forests **Linear/Lasso/Ridge 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 **Seasonal ARIMA**

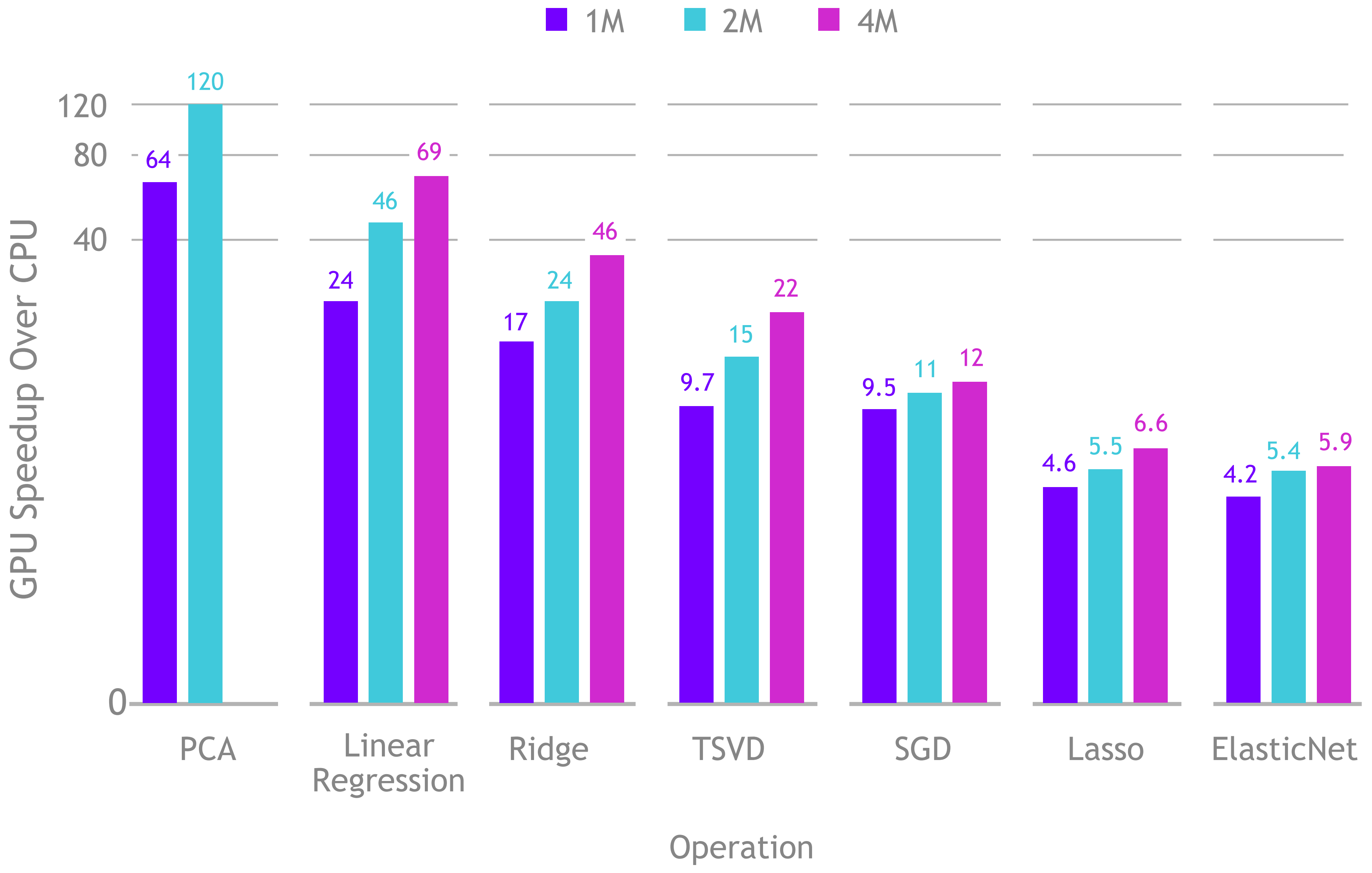
Cross Validation

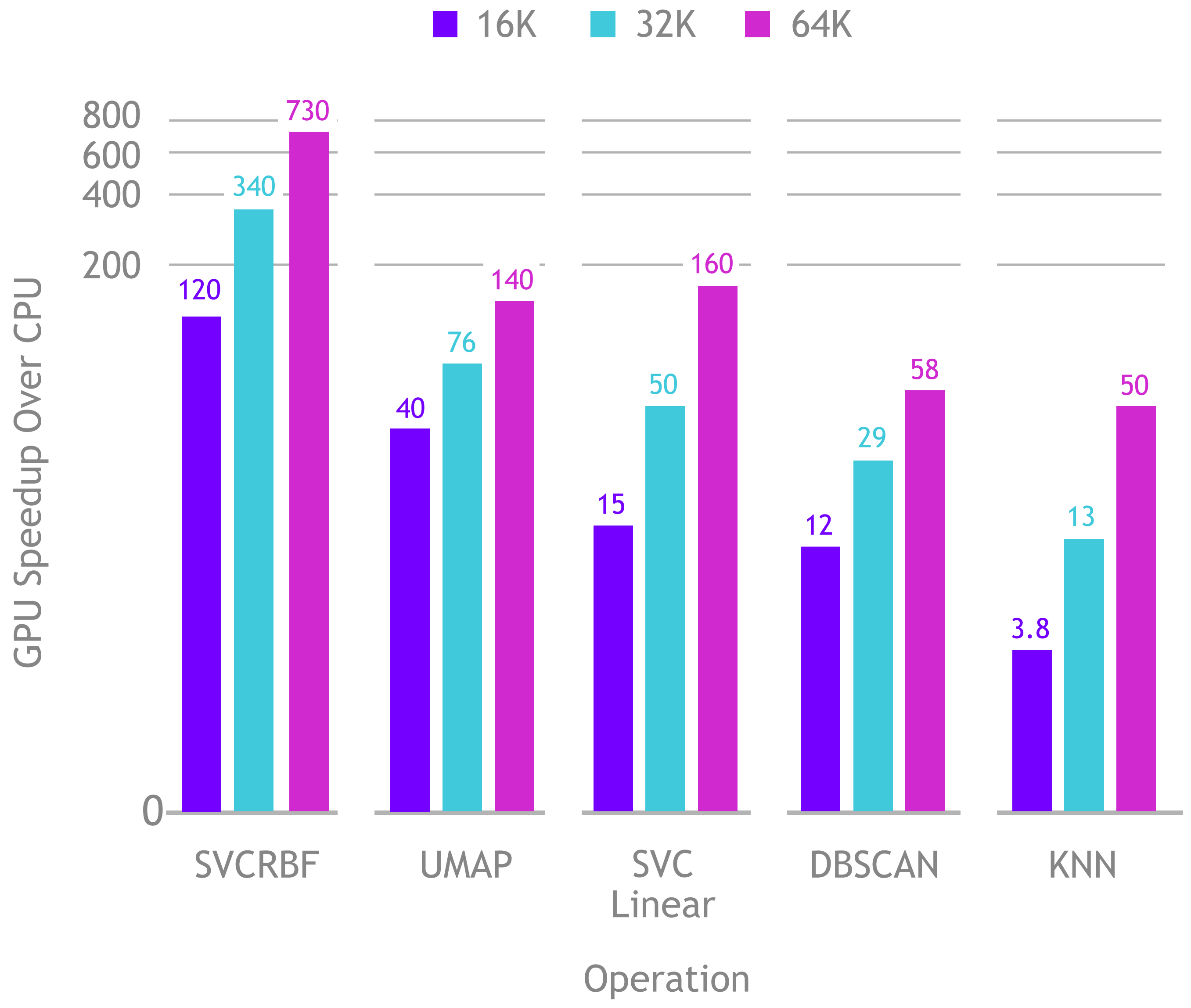
Hyper-parameter Tuning

More to come!

Key:
Preexisting | ***NEW or enhanced for 0.14***

Benchmarks: Single-GPU cuML vs Scikit-learn





1x V100 vs. 2x 20 Core CPU

XGBoost + RAPIDS: Better Together

RAPIDS 0.14 comes paired with XGBoost 1.1

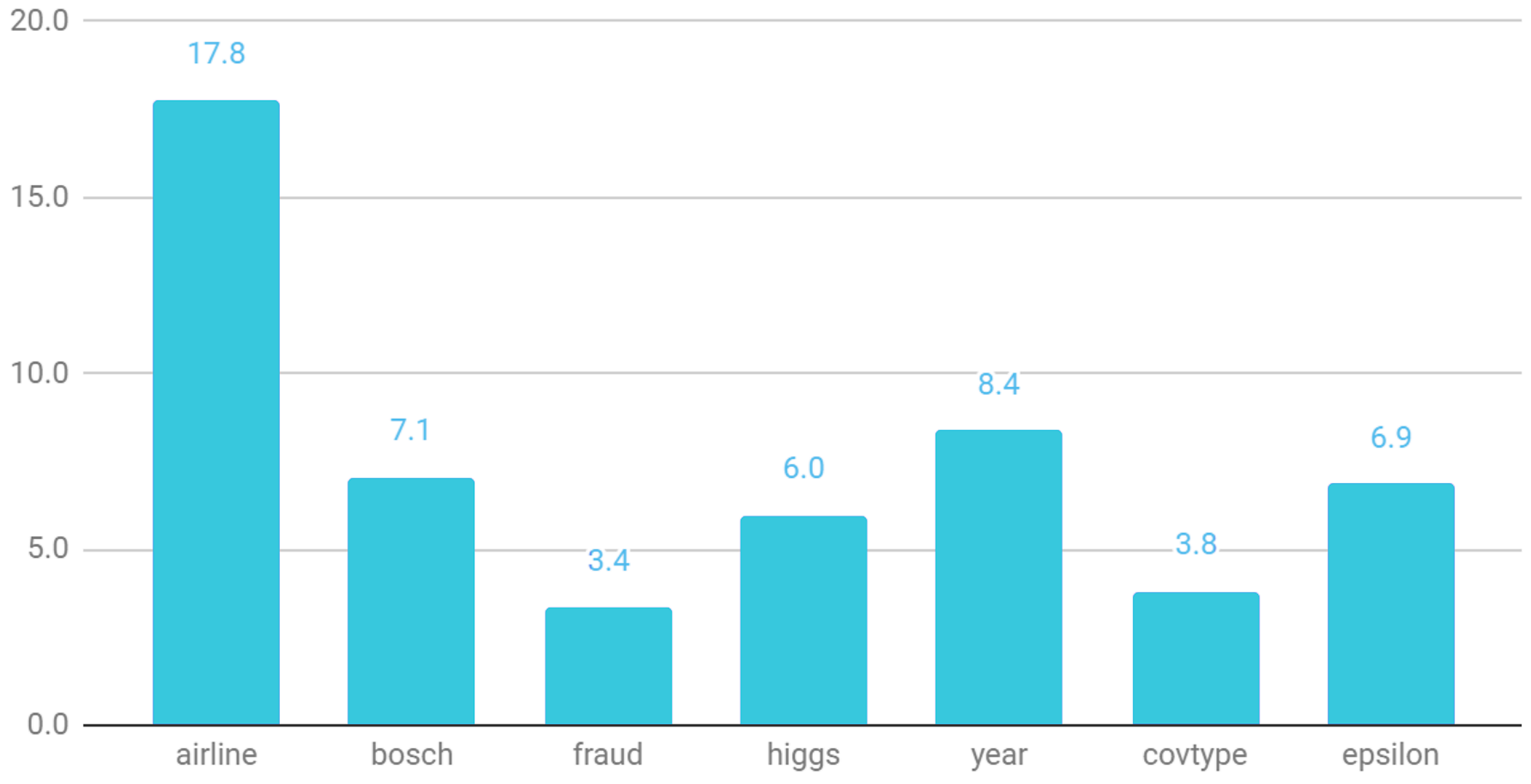
XGBoost now builds on the GPU array interface standards to provide zero-copy data import from cuDF, cuPY, Numba, PyTorch and more

Official Dask API makes it easy to scale to multiple nodes or multiple GPUs

Memory usage when importing GPU data decreased by 2/3 or more

New objectives support Learning to Rank on GPU

All RAPIDS changes are integrated upstream and provided to all XGBoost users – via pypi or RAPIDS conda



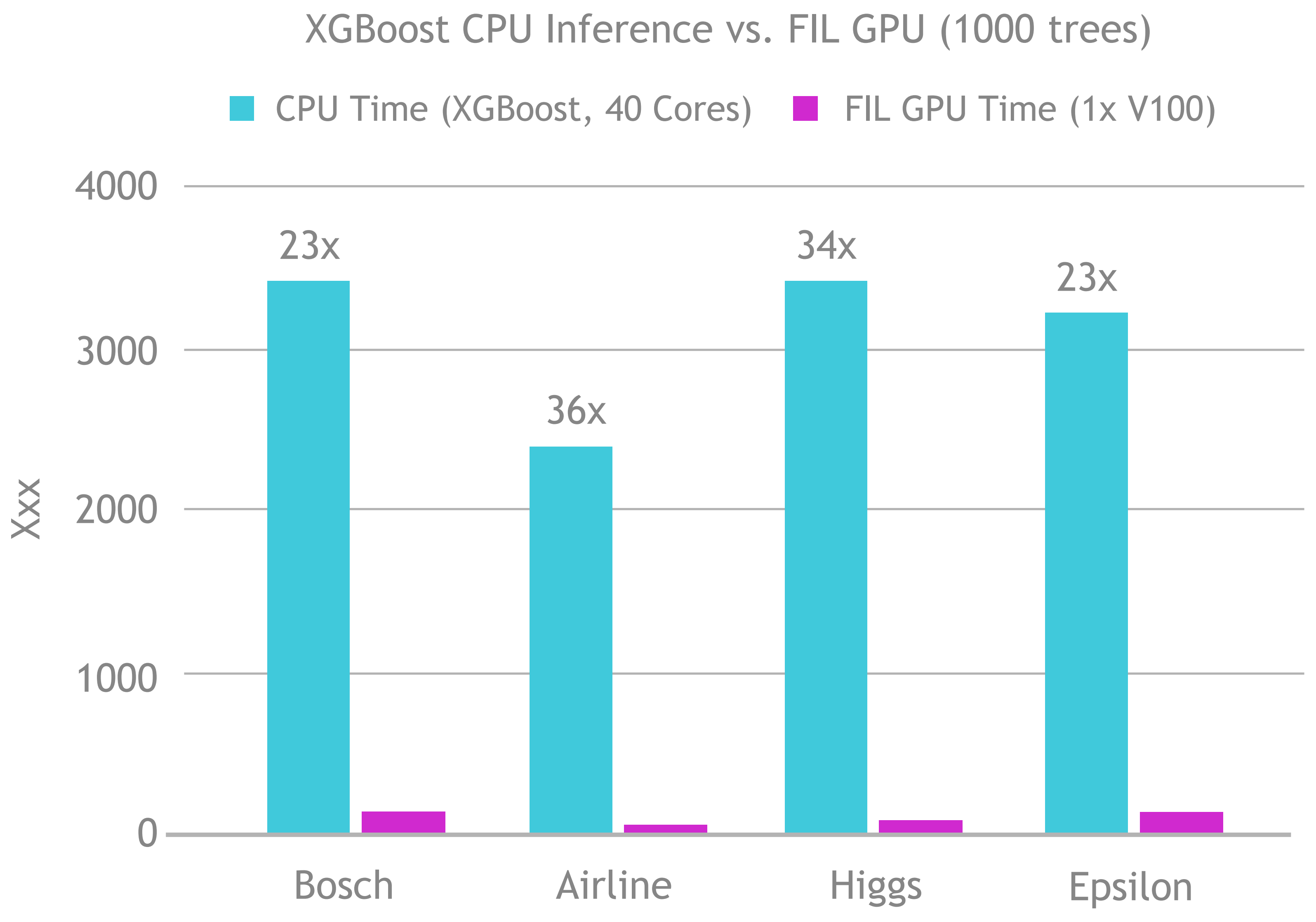
XGBoost speedup on GPUs comparing a single NVIDIA V100 GPU to a dual 20-core Intel Xeon E5-2698 server

Forest Inference

Taking Models From Training to Production

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

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



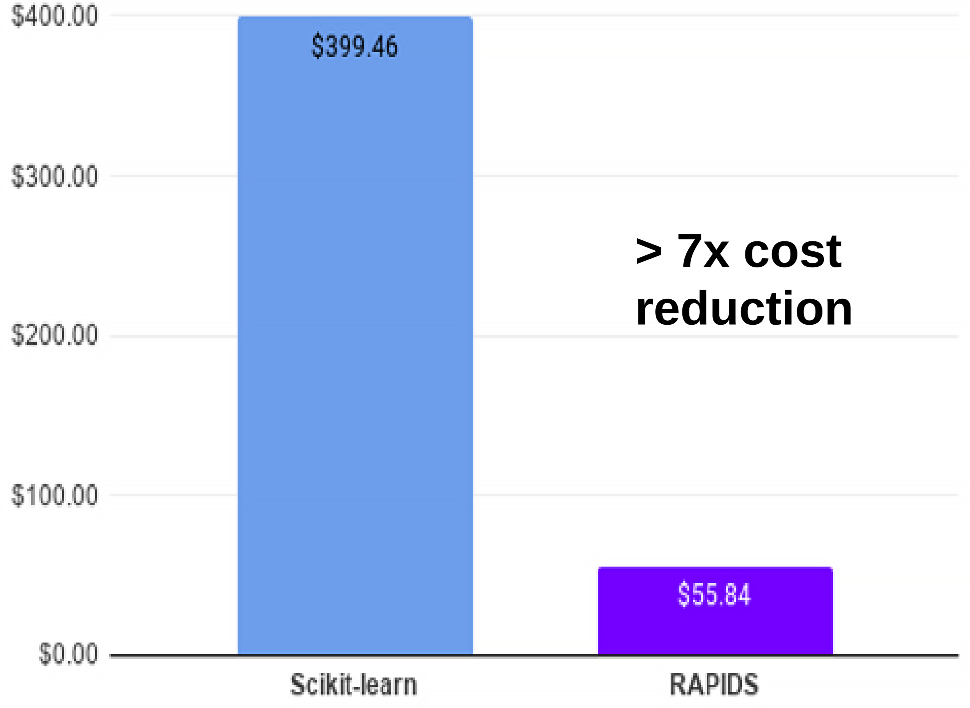
RAPIDS Integrated into Cloud ML Frameworks

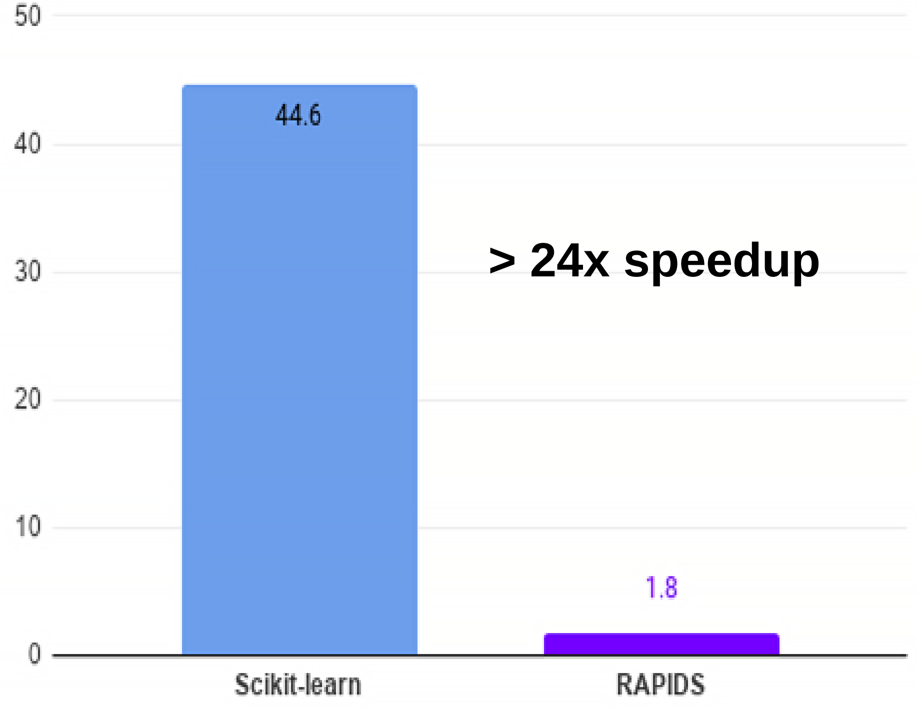
Accelerated machine learning models in RAPIDS give you the flexibility to use hyperparameter optimization (HPO) experiments to explore all variants to find the most accurate possible model for your problem.

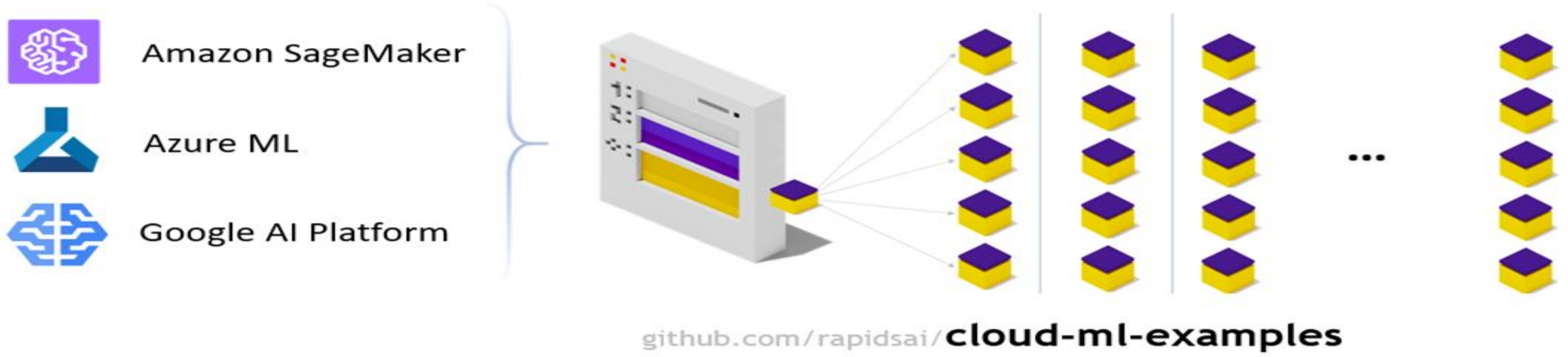
With GPU acceleration, RAPIDS models can train 40x faster than CPU equivalents, enabling more experimentation in less time.

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

https://rapids.ai/hpo







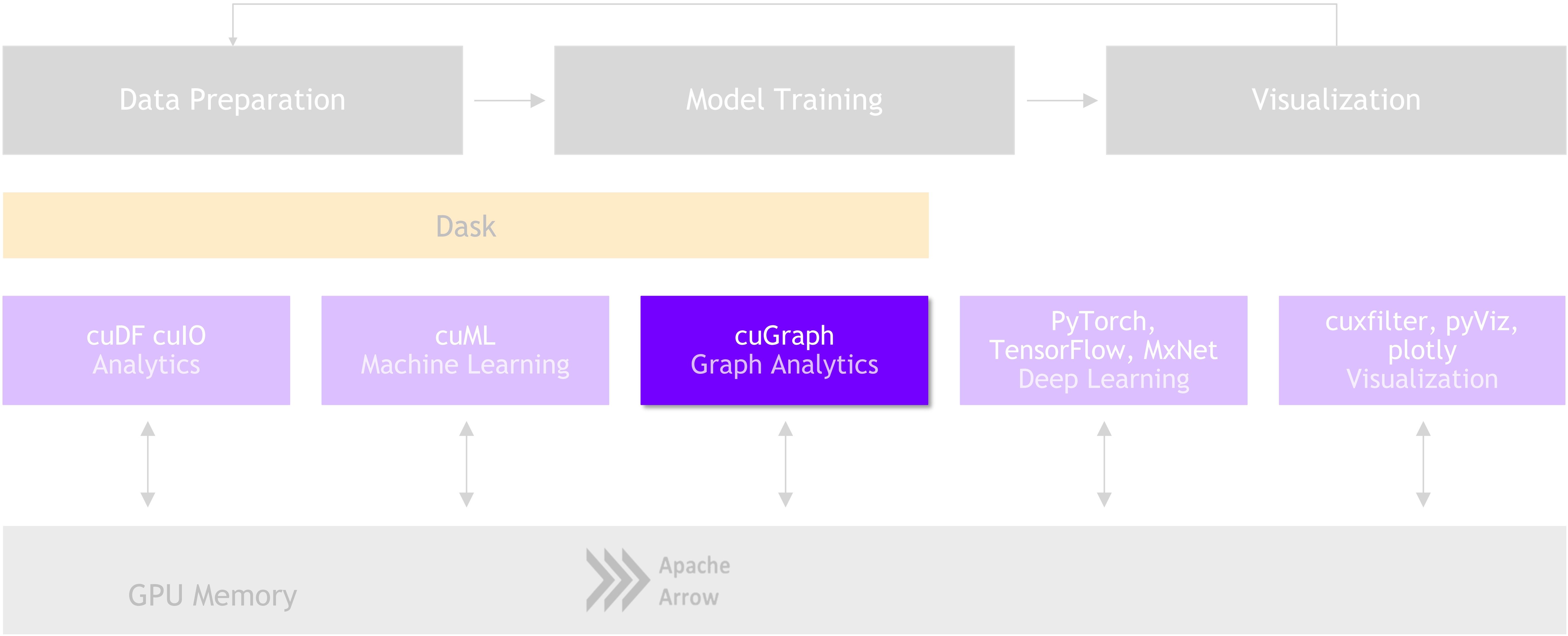
Ray/Tune

Dask ML

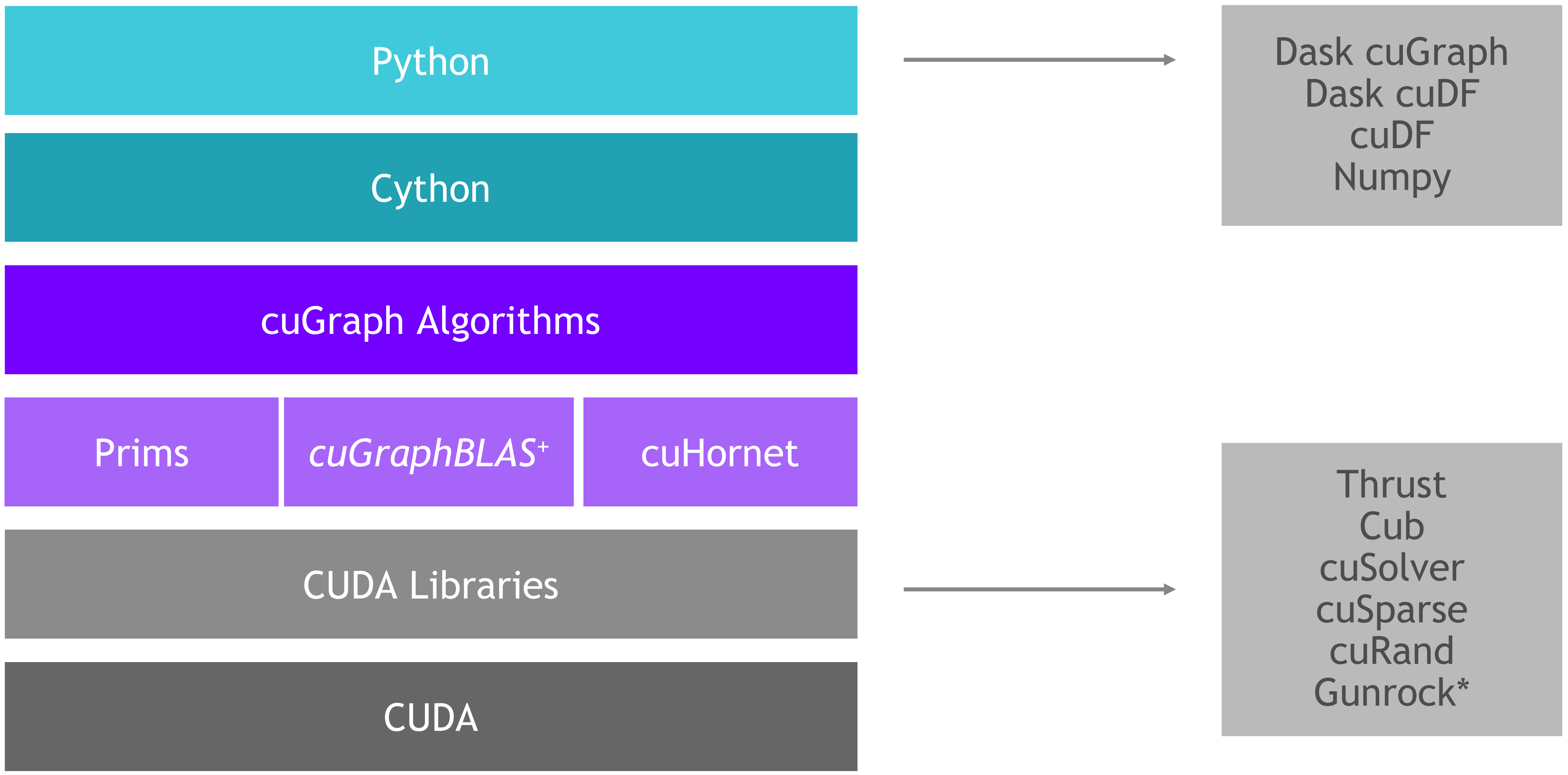
cuGraph

Graph Analytics

More Connections, More Insights



Graph Technology Stack

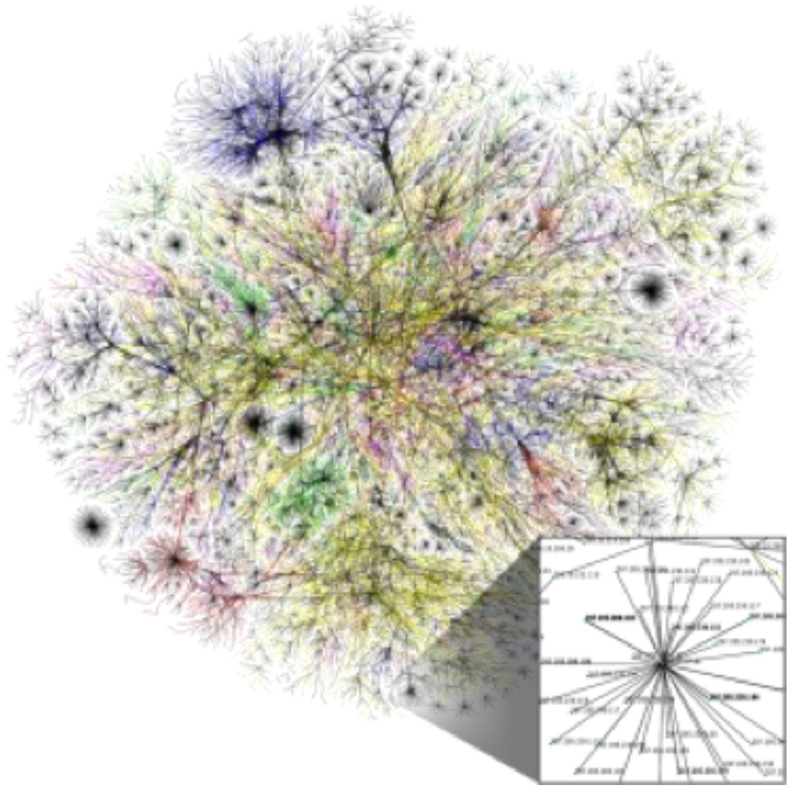


+cuGraphBLAS is still in development and will be ready late 2020

** Gunrock is from UC Davis*

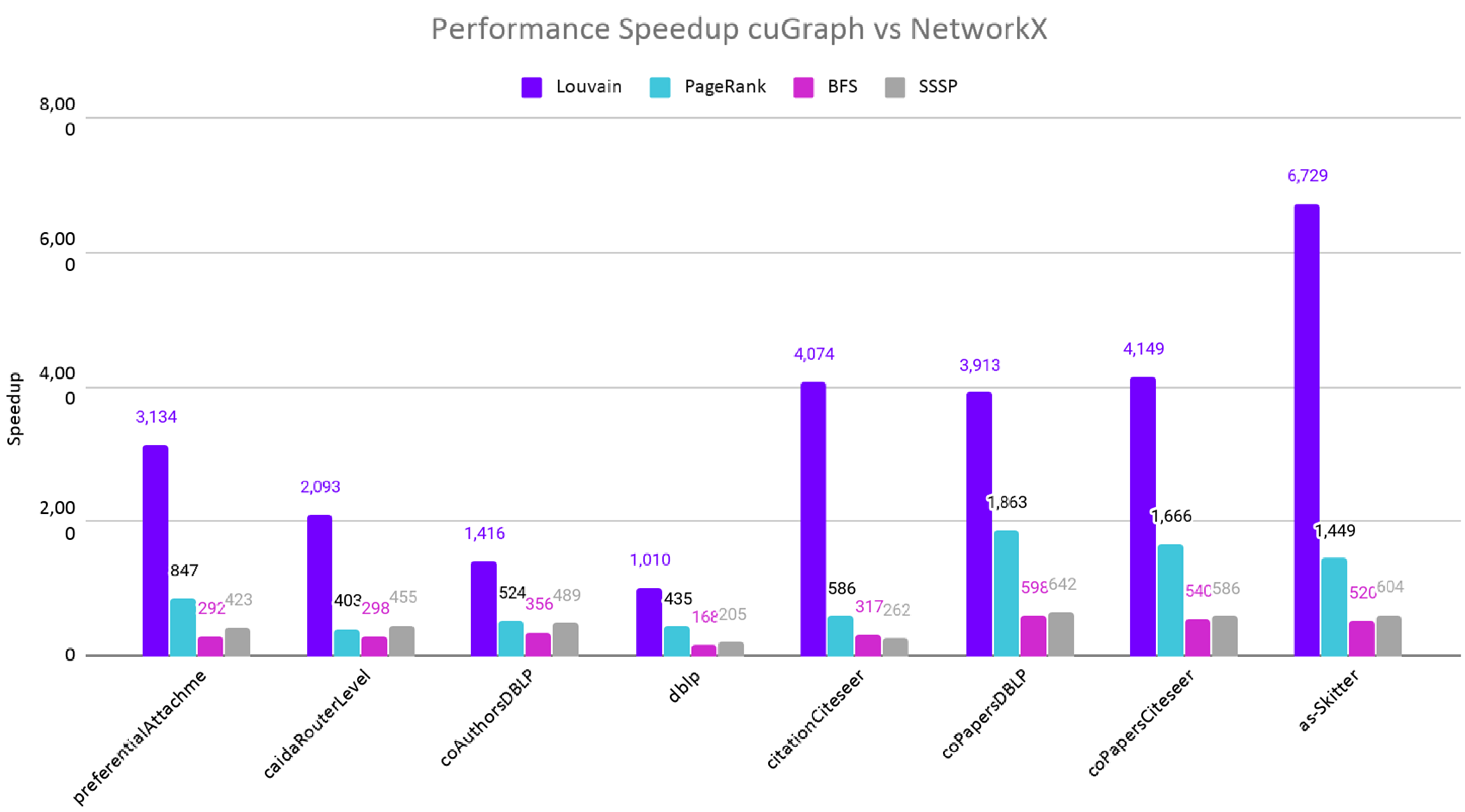
Algorithms

GPU-accelerated NetworkX



Community	Spectral Clustering - Balanced Cu and Modularity Maxim **Louvain (redone for 0.14)** Ensemble Clustering for Graphs KCore and KCore Number Triangle Counting K-Truss
Components	Weakly Connected Components Strongly Connected Components
Link Analysis	Page Rank (Multi-GPU) Personal Page Rank
Link Prediction	Jaccard Weighted Jaccard Overlap Coefficient
Traversal	Single Source Shortest Path (SSSP) Breadth First Search (BFS)
Centrality	Katz **Betweenness Centrality (redone in 0.14)**
Structure	Graph Classes Subgraph Extraction
Utilities	Renumbering Auto-Renumbering **Force Atlas 2**

Benchmarks: Single-GPU cuGraph vs NetworkX



Dataset	Nodes	Edges
preferentialAttachment	100,000	999,970
caidaRouterLevel	192,244	1,218,132
coAuthorsDBLP	299,067	299,067
Dblp-2010	326,186	1,615,400
citationCiteseer	268,495	2,313,294
coPapersDBLP	540,486	20,491,458
coPapersCiteseer	434,102	32,073,440
As-Skitter	1,696,415	22,190,596

Many more!

See also

Many more RAPIDS-related projects

NVIDIA-sponsored projects

- cuSpatial – Spatial Analytics
- cuSignal – Accelerated signal processing
- CLX – RAPIDS and Deep Learning for Cybersecurity and Log Analytics
- cuStreamz – GPU-accelerated streaming data (matching Python streamz API)
- NVTabular – Deep Learning for tabular datam with loaders accelerated by RAPIDS

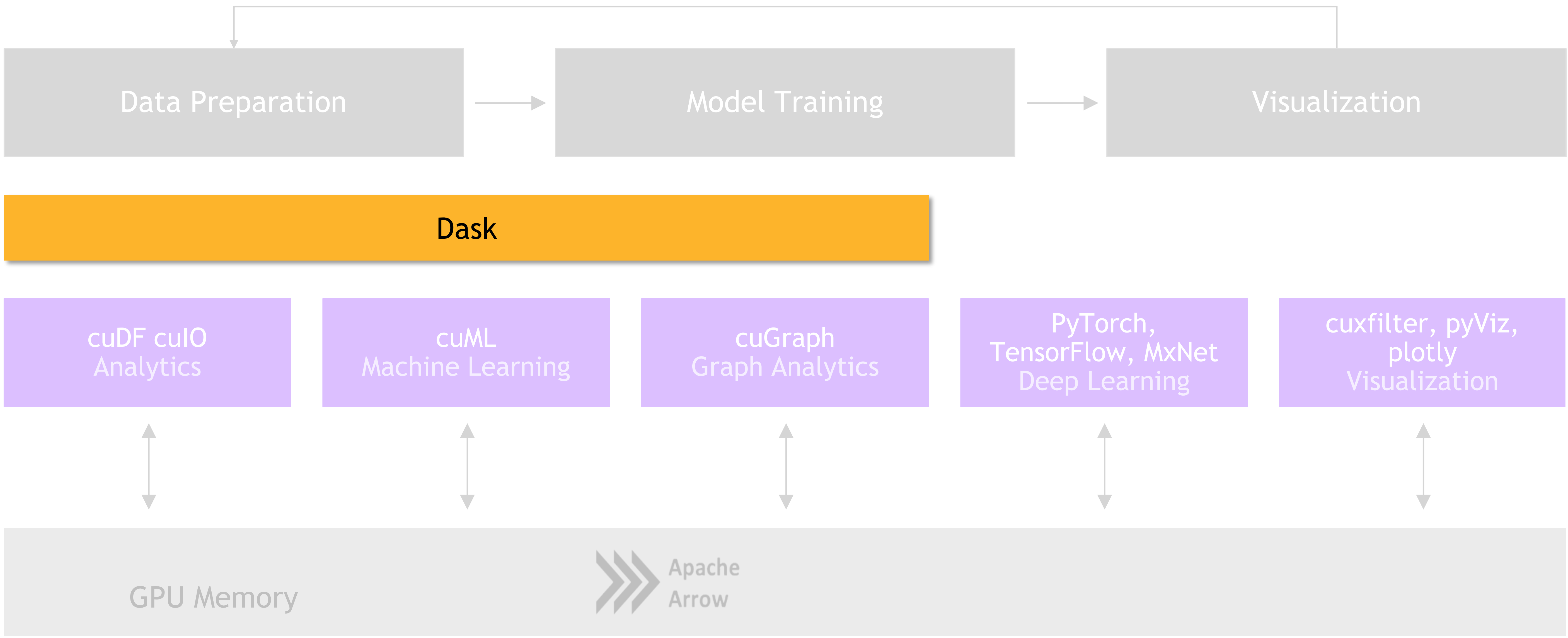
Others:

- BlazingSQL – GPU-accelerated SQL engine
- Plot.ly – Python charting with GPU accelerated backends
- Graphistry – Interactive visualization for graphs and complex data

Dask and
RAPIDS Distributed Compute

RAPIDS

Scaling RAPIDS with Dask



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

Numba NumPy pandas PyData scikit learn

Scaling Up and Out with RAPIDS, Dask, 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

Numba NumPy pandas PyData scikit learn

DASK

Multi-core and distributed PyData

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

DASK

Scale Out / Parallelize

Why Dask?

DEPLOYABLE

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

PYDATA NATIVE

- **Easy Migration:** Built on top of NumPy, Pandas Scikit-Learn, etc
- **Easy Training:** With the same APIs
- **Trusted:** With the same developer community

EASY SCALABILITY

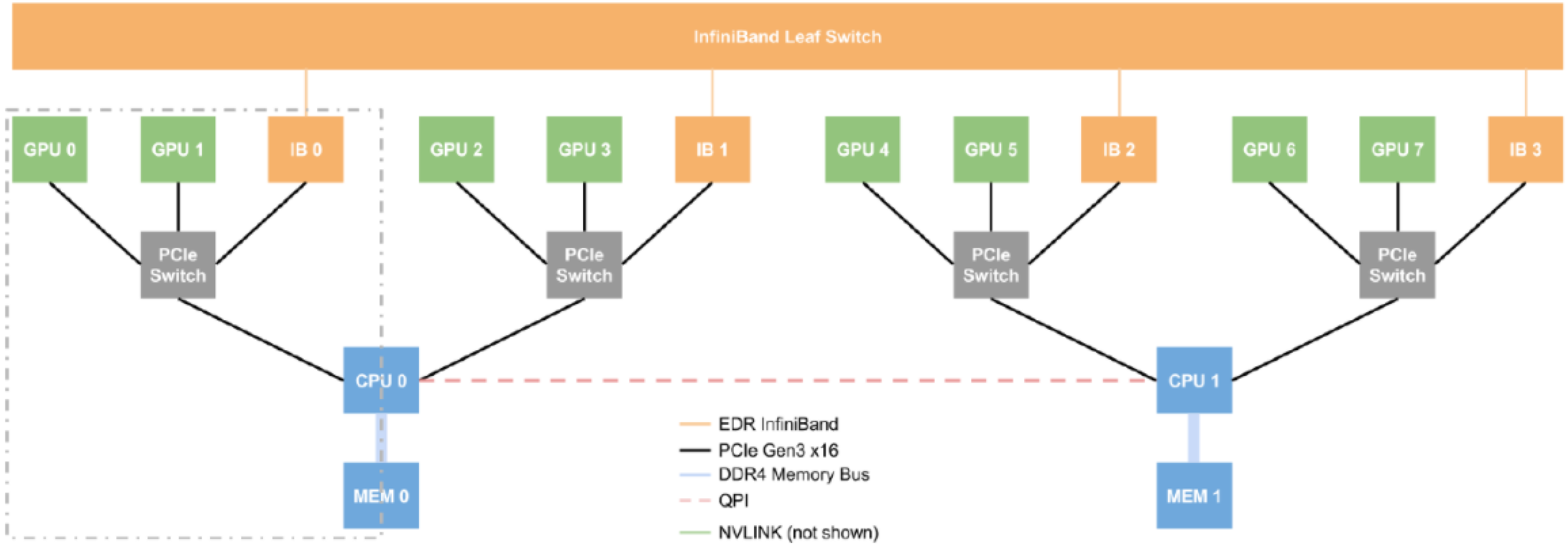
- Easy to install and use on a laptop
- Scales out to thousand node clusters

POPULAR

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

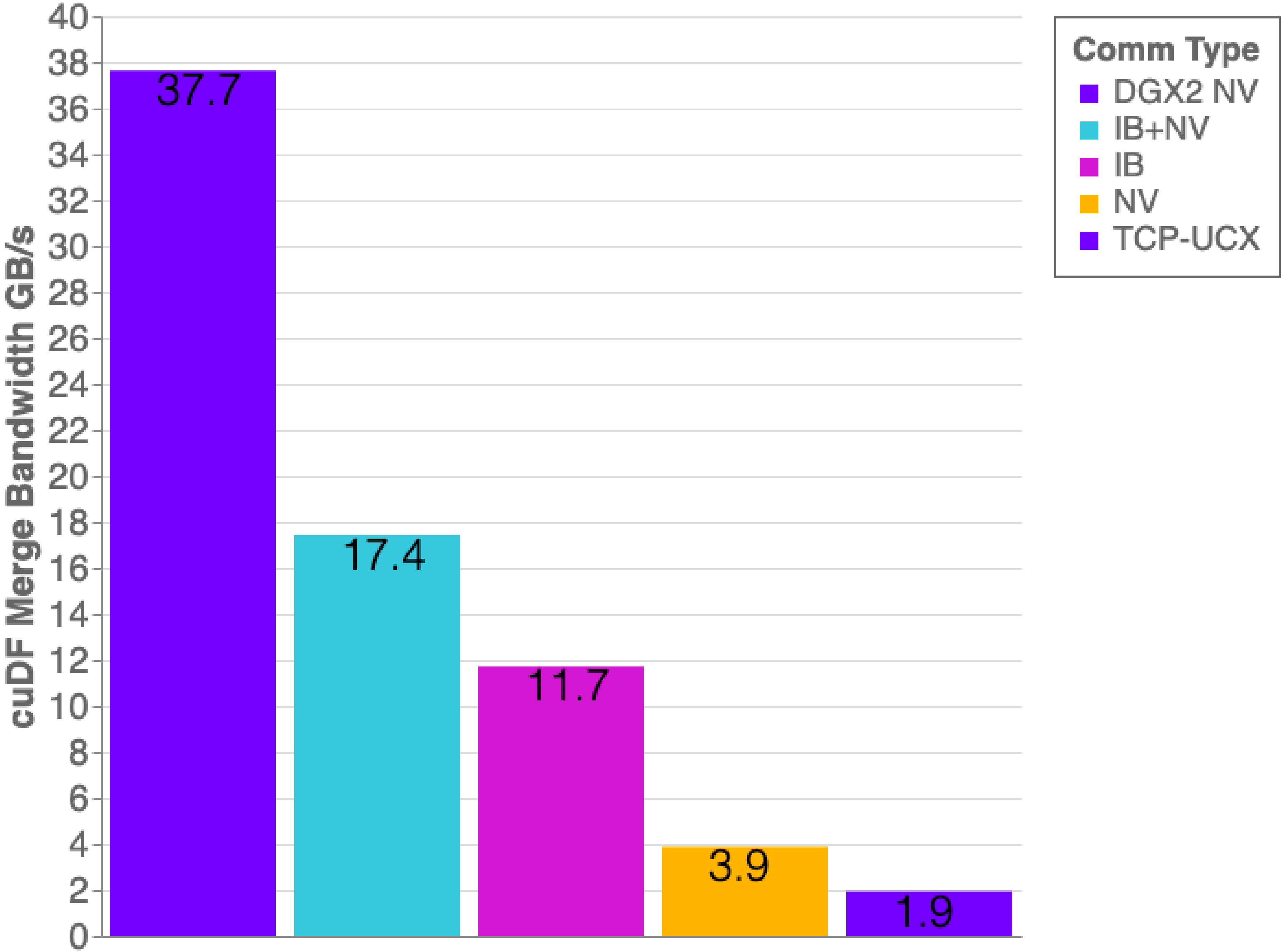
Why OpenUCX?
Bringing Hardware Accelerated Communications to Dask

- TCP sockets are slow!
- Topologies are complex!
- UCX provides uniform access to transports (TCP, InfiniBand, shared memory, NVLink, ethernet)
- Open source Python bindings for UCX (ucx-py) now available in beta
- Will provide best communication performance, with topology-aware routing, to Dask and cuML communications




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

Benchmarks: Distributed cuDF Random Merge



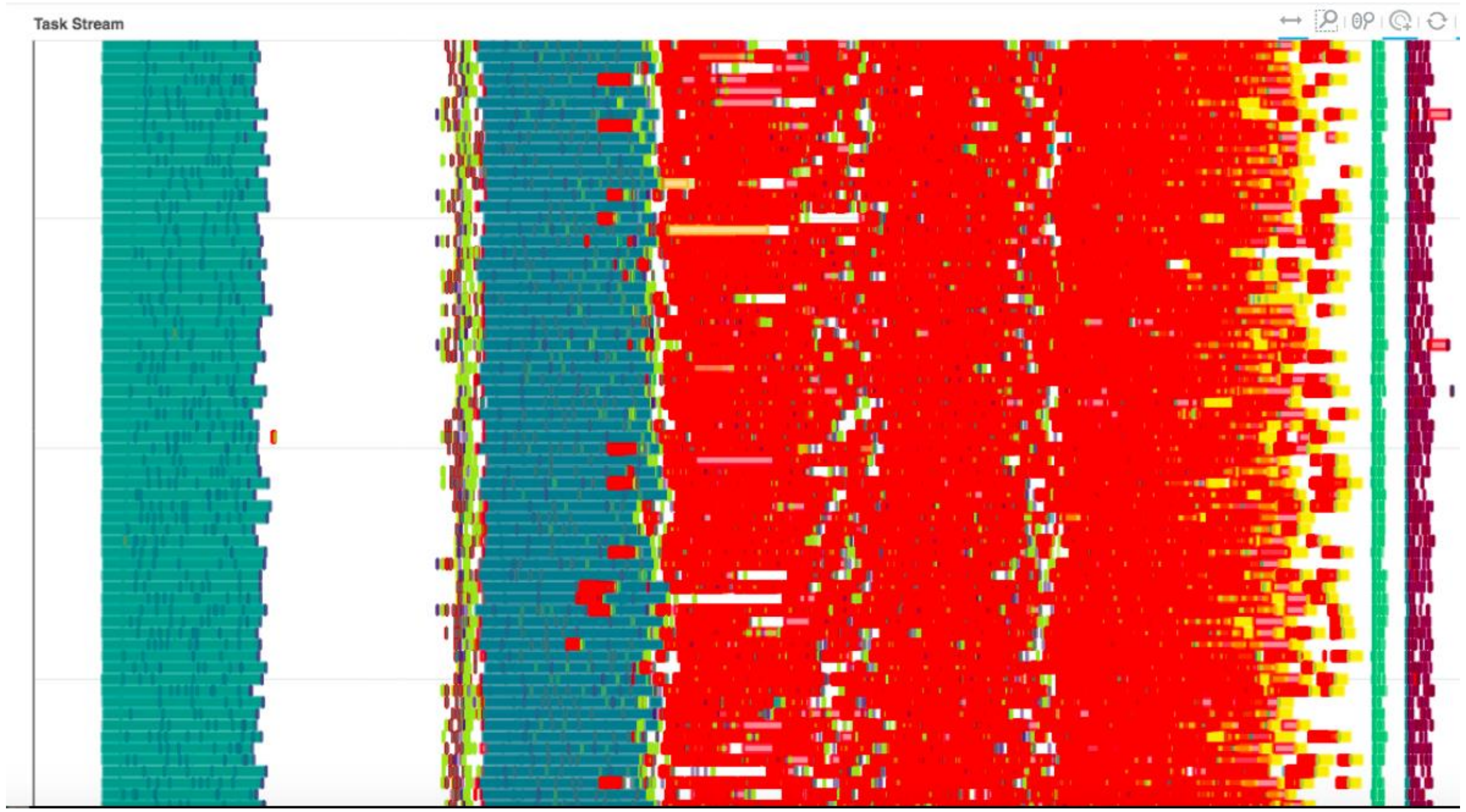
cuDF v0.14, UCX-PY 0.14

- Running on NVIDIA DGX-2:
  - GPU: NVIDIA Tesla V100 32GB
  - CPU: Intel(R) Xeon(R) CPU 8168 @ 2.70GHz
- Benchmark Setup:
  - DataFrames: Left/Right 1x int64 column key column, 1x int64 value columns
  - Merge: Inner
  - 30% of matching data balanced across each partition

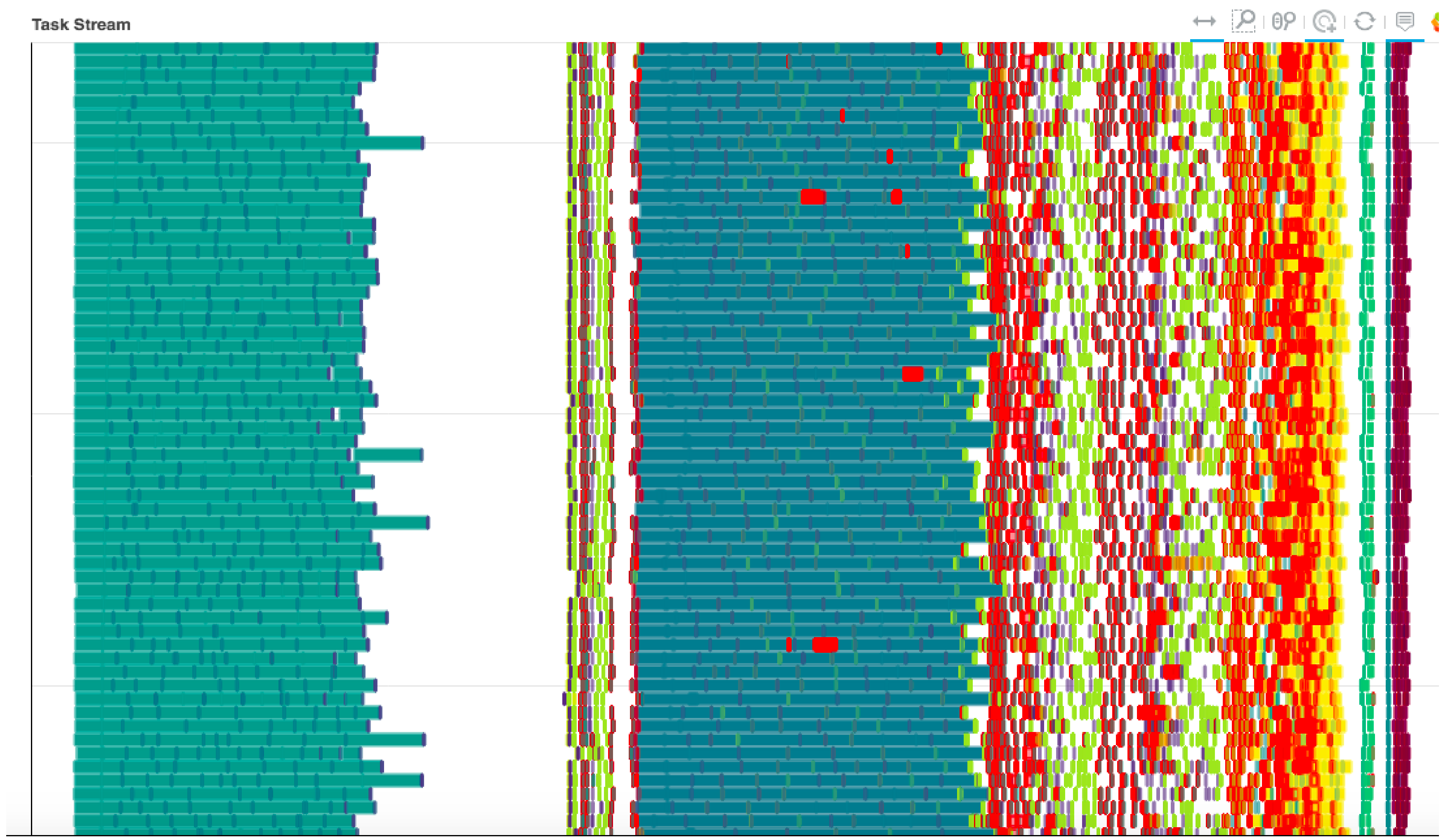
Real-world performance in Dask

UCX impact (with IB+NVLink) on TPCx-BB Query 3

UCX off (red = waiting on comms)



UCX on (red = waiting on comms)



Large-scale Benchmarking with Distributed RAPIDS

WHAT IS TPCX-BB®?

Comparing Big Data Platforms since the Cambrian Explosion of Big Data

TPC is the leader in benchmarking Data Analytics and Data Science Systems

TPCx-BB benchmark measures the performance of both hardware and software components by executing 30 frequently performed analytical queries in the context of retailers with physical and online store presence

Is the only TPC benchmark that starts from disk, does ETL (structured, semi-structured, and unstructured), and machine learning

TPCX-BB

CPU Performance

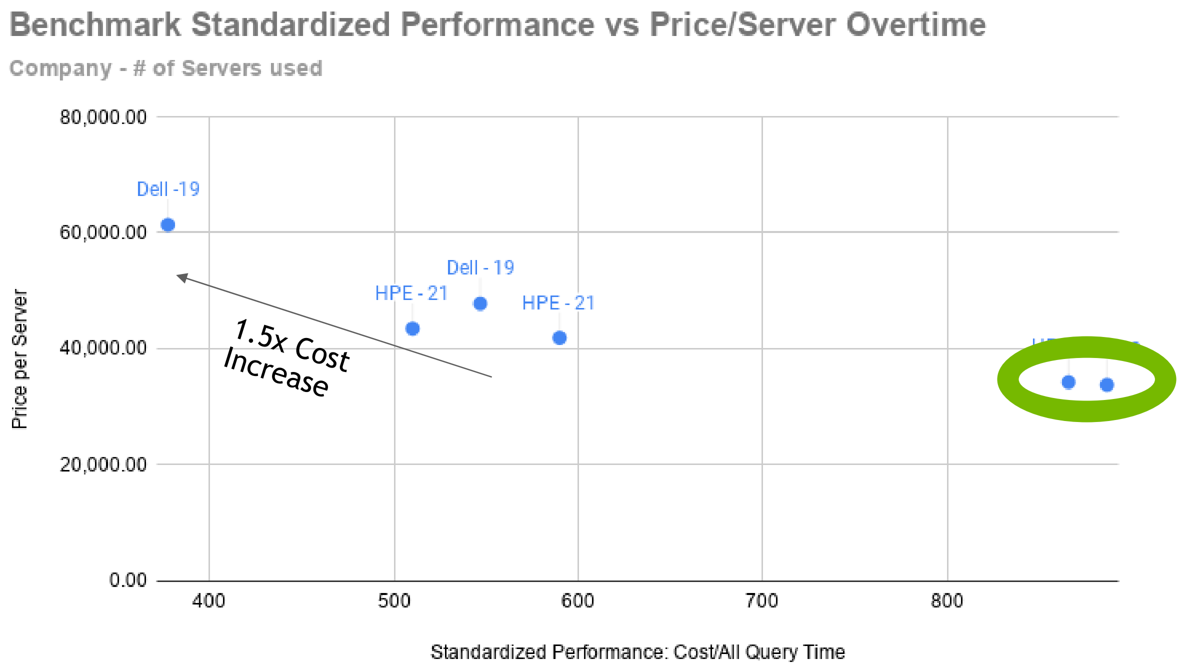
Hadoop Processing, Reading from Disk

HDFS Read	Query	HDFS Write	HDFS Read	ETL	HDFS Write	HDFS Read	ML Train

Spark In-Memory Processing

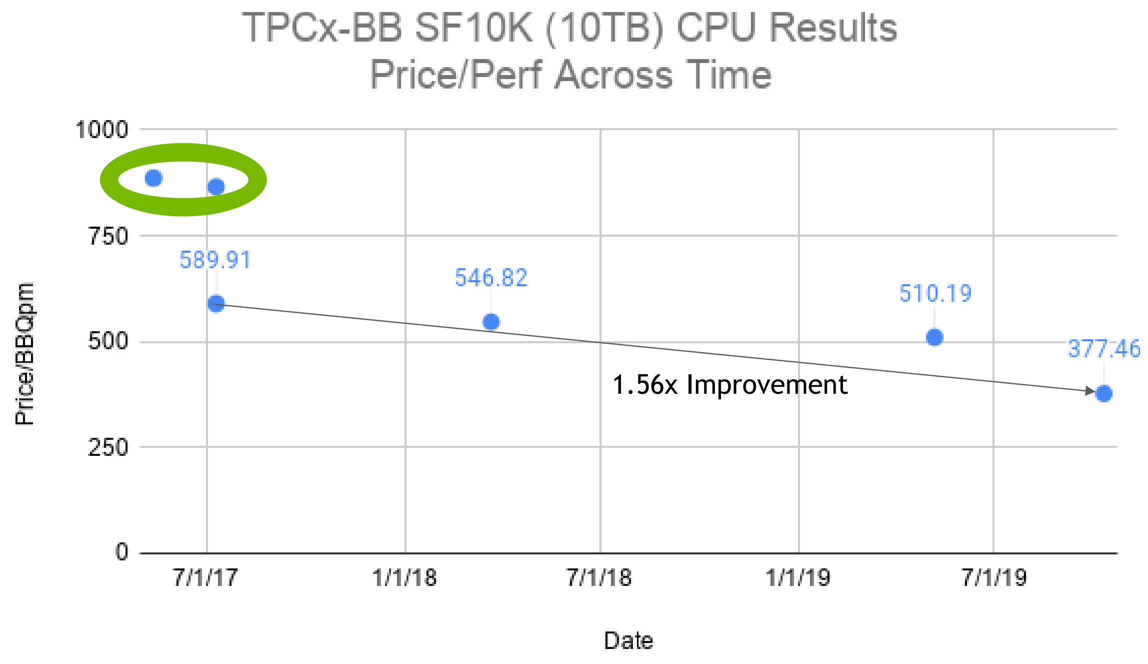
HDFS Read	Query	ETL	ML Train

25-100x Improvement
Less code
Language flexible
Primarily In-Memory



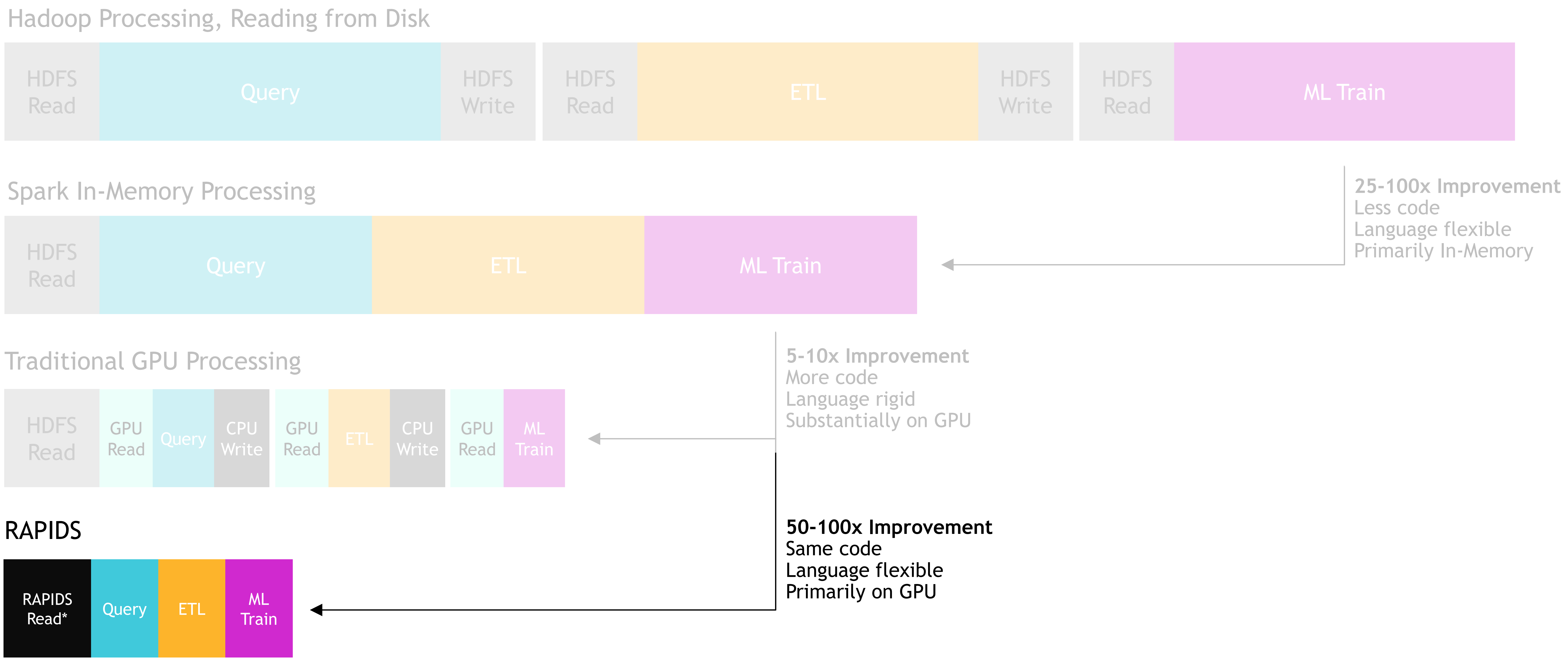
Current Leader, Dell: 19 servers @ $61K/server

Only ~1.5x speedup in last 2 years, driven primarily by scale up as opposed to scale out



TPCX-BB

GPU Performance



RAPIDS RUNNING TPCX-BB AT 1 TB AND 10 TB SFS

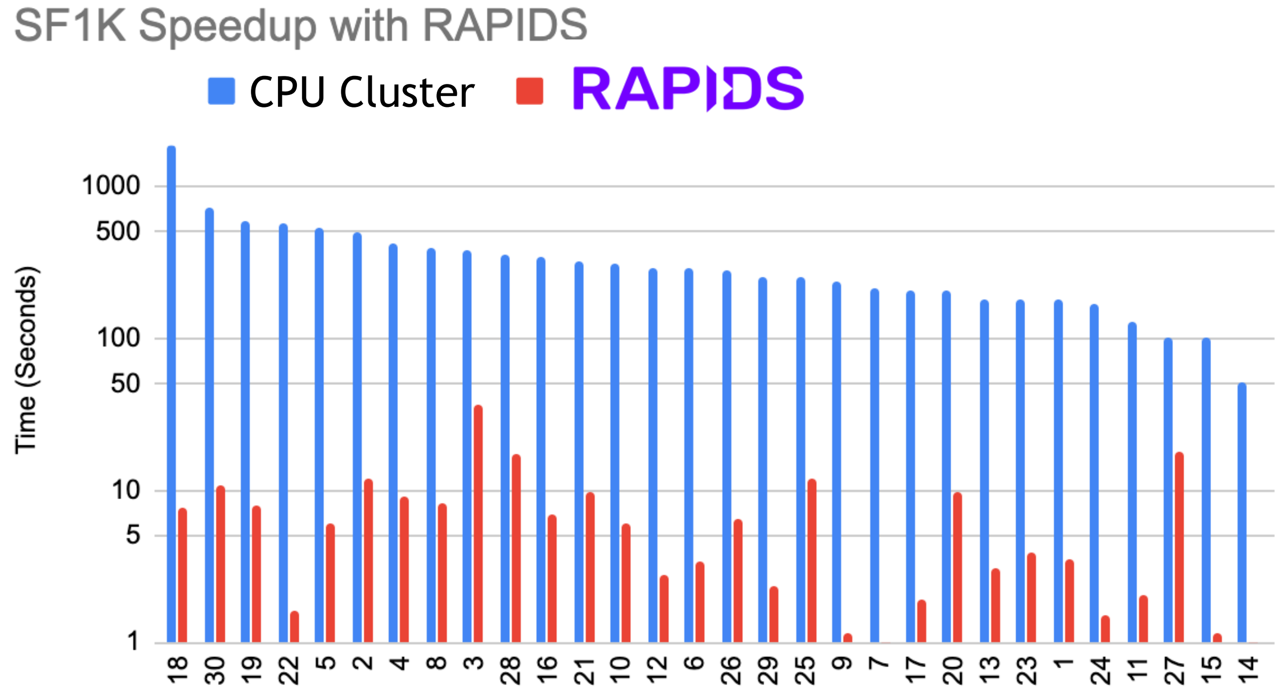
Up to 350x faster queries; Hours to Seconds!

Like other TPC benchmarks, TPCx-BB can be run at multiple “Scale Factors”:

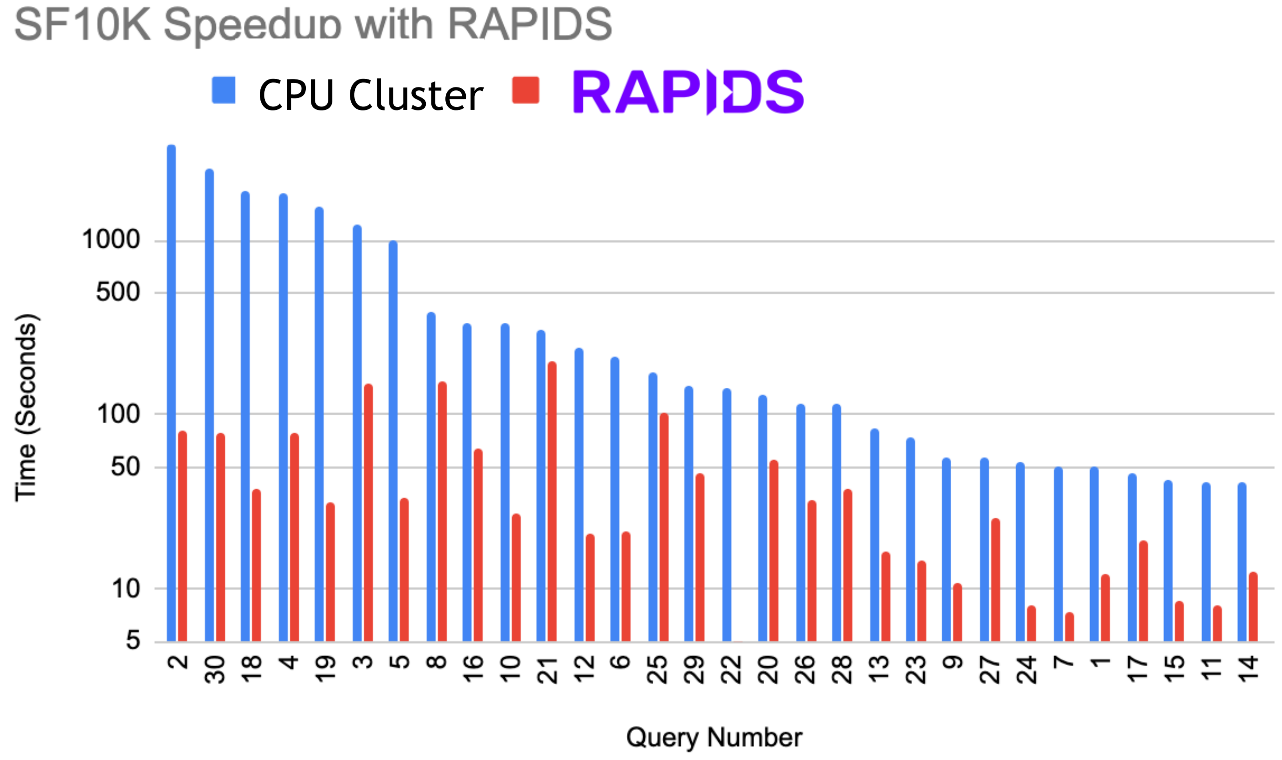
- SF1 - 1GB
- SF1K - 1 TB
- SF10K - 10 TB

We’ve been benchmarking RAPIDS implementations of the TPCx-BB queries at the SF1K (Single DGX-2) & SF10K (17x DGX-1) scales

Our results indicate that GPUs provide dramatic cost and time-savings for small scale *and* large-scale data analytics problems. (Unofficial results currently)



Avg: 51x speed-up
>40x Normalized for Cost



Avg: >10x speed-up
>5x Normalized for Cost

blazingSQL DASK Numba

QUERY SPOTLIGHT - UDFS AT SCALE ON GPUS

Query 3: What is viewed before a purchase?

Repartition web-clickstream table on user key
- Ensure all web activity records available within a single “chunk” (partition) of records that fit within the memory space of a single worker

Compute aggregate metrics on user’s sessions

Sort on event timestamp within user sessions

Run a user-defined-function with custom processing logic for classifying session behavior

- DataFrame APIs are great, but real business logic is complex, needing to support custom code
- RAPIDS uses Numba to compile simple Python expressions into GPU accelerated logic
- Run Python on GPUs!

```python
@cuda.jit
def find_items_viewed_before_purchase_kernel(
    relevant_idx_col, user_col, timestamp_col, item_col, out_col, N
):
    """
    Find the past N items viewed after a relevant purchase was made,
    as defined by the configuration of this query.
    """
    i = cuda.grid(1)
    relevant_item = q03_purchased_item_IN

    if i < (relevant_idx_col.size):  # boundary guard
        # every relevant row gets N rows in the output, so we need to map the indexes
        # back into their position in the original array
        orig_idx = relevant_idx_col[i]
        current_user = user_col[orig_idx]

        # look at the previous N clicks (assume sorted descending)
        rows_to_check = N
        remaining_rows = user_col.size - orig_idx

        if remaining_rows <= rows_to_check:
            rows_to_check = remaining_rows - 1

        for k in range(1, rows_to_check + 1):
            if current_user != user_col[orig_idx + k]:
                out_col[i * N + k - 1] = 0

            # only checking relevant purchases via the relevant_idx_col
            elif (timestamp_col[orig_idx + k] <= timestamp_col[orig_idx]) & (
                timestamp_col[orig_idx + k]
                >= (timestamp_col[orig_idx] - q03_days_in_sec_before_purchase)
            ):
                out_col[i * N + k - 1] = item_col[orig_idx + k]
            else:
```

QUERY SPOTLIGHT - NATURAL LANGUAGE PROCESSING

Query 18 - are bad reviews correlated with bad sales?

Subset the data to a set of four months
- After joining tables containing store, store sales, data, and customer review data, split by row groups for better parallelism

For each store, regress date on the sum of net sales and retain the beta coefficient and select those stores with a negative slope
- Repartition this table to be one partition (it is small: only 192 rows at SF1000)

Make a list of all the unique store names
- RAPIDS has an extensive set of string functions, bringing string manipulation to the GPU

Find reviews that include any of the store names

For reviews that contain a store's name, return sentences containing a negative word and the negative word itself
- Break reviews into sentences
- Search sentences for words contained in a text file of negative words
- Return the store name, date of the review, sentence, and word for sentences where negative words appeared.
- NLP on GPU!

```
no_nulls["pr_review_content"] = no_nulls.pr_review_content.str.replace_multi(
    [". ", "? ", "! "], EOL_CHAR, regex=False
)
sentences = no_nulls.map_partitions(create_sentences_from_reviews)

# need the global position in the sentence tokenized df
sentences["x"] = 1
sentences["sentence_tokenized_global_pos"] = sentences.x.cumsum()
del sentences["x"]

# This file comes from the official TPCx-BB kit
# We extracted it from bigbenchqueriesmr.jar
with open("negativeSentiment.txt") as fh:
    negativeSentiment = list(map(str.strip, fh.readlines()))
    # dedupe for one extra record in the source file
    negativeSentiment = list(set(negativeSentiment))

word_df = sentences.map_partitions(
    create_words_from_sentences,
    global_position_column="sentence_tokenized_global_pos",
)
sent_df = cudf.DataFrame({"word": negativeSentiment})
sent_df["sentiment"] = "NEG"
sent_df = dask_cudf.from_cudf(sent_df, npartitions=1)

word_sentence_sentiment = word_df.merge(sent_df, how="inner", on="word")

word_sentence_sentiment["sentence_idx_global_pos"] = word_sentence_sentiment[
    "sentence_idx_global_pos"
].astype("int64")
sentences["sentence_tokenized_global_pos"] = sentences[
    "sentence_tokenized_global_pos"
].astype("int64")
```

Getting Started

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 example workflows.
3. Build your own data science workflows.
4. Join our community conversations on Slack, Google, and Twitter.
5. Contribute back. Don't forget to ask and answer questions on Stack Overflow.

Easy Installation

Interactive Installation Guide

RAPIDS RELEASE SELECTOR

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

	Preferred		Advanced	
METHOD	Conda	Docker + Examples	Docker + Dev Env	Source
RELEASE	Stable (0.14)		Nightly (0.15a)	

PACKAGES	All Packages	cuDF	cuML	cuGraph	cuSignal	cuSpatial	cuxfilter
LINUX	Ubuntu 16.04	Ubuntu 18.04	CentOS 7	RHEL 7			
PYTHON	Python 3.6	Python 3.7					
CUDA	CUDA 10.0	CUDA 10.1.2	CUDA 10.2				

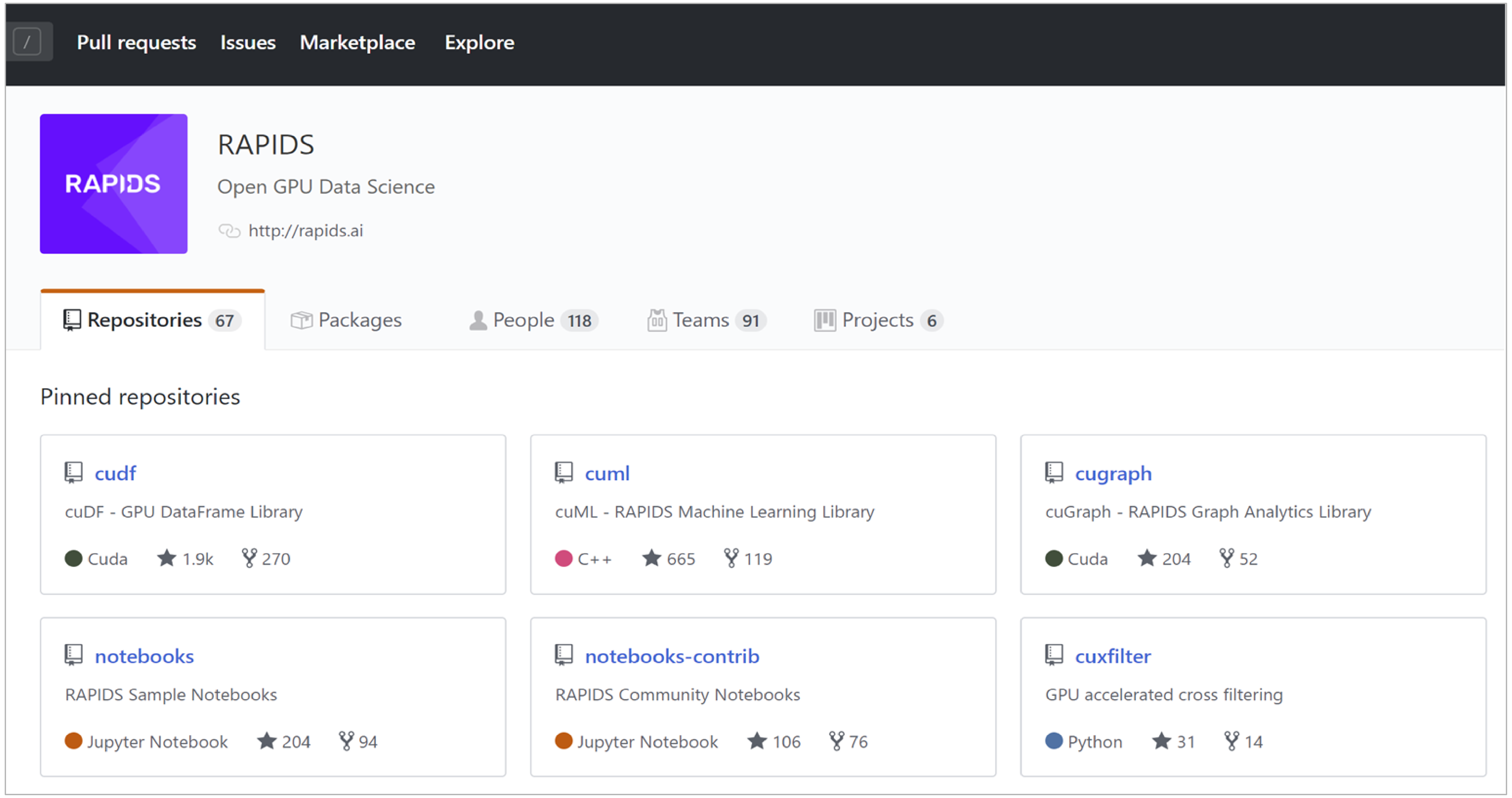
NOTE: Ubuntu 16.04/18.04 & CentOS 7 use the same `conda install` commands.

COMMAND

```
conda install -c rapidsai -c nvidia -c conda-forge \
    -c defaults rapids=0.14 python=3.6
```

https://rapids.ai/start.html

Explore: RAPIDS Github



https://github.com/rapidsai

THANK YOU

John Zedlewski
jzedlewski@nvidia.com

@Zstats

@RAPIDSai

RAPIDS