

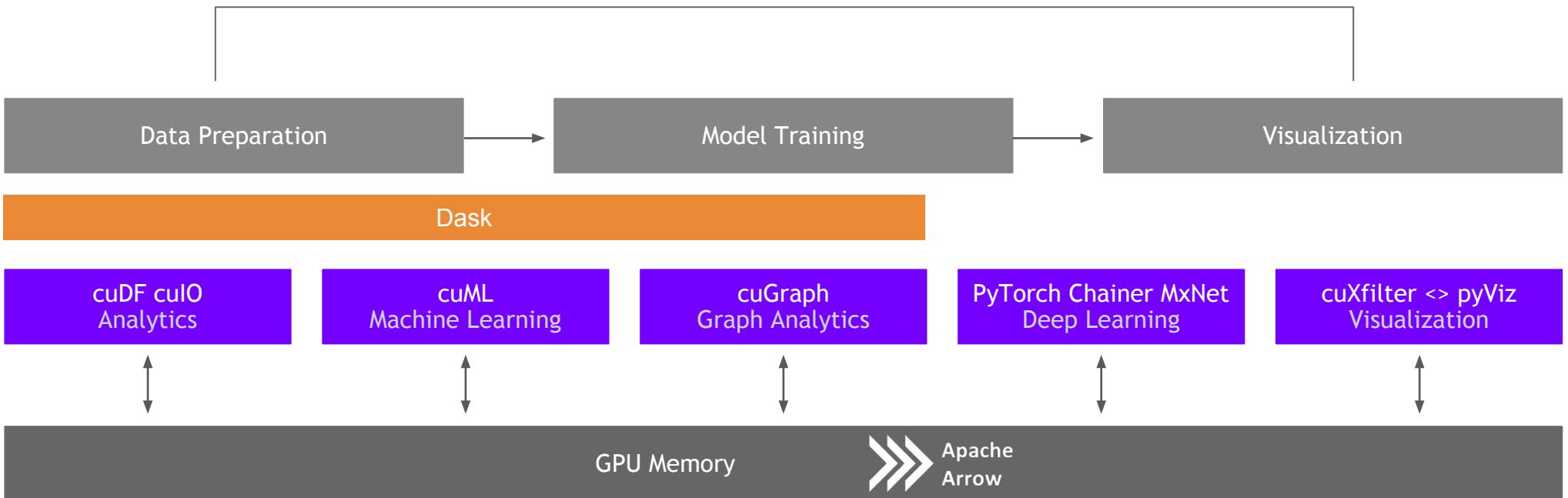
RAPIDS

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

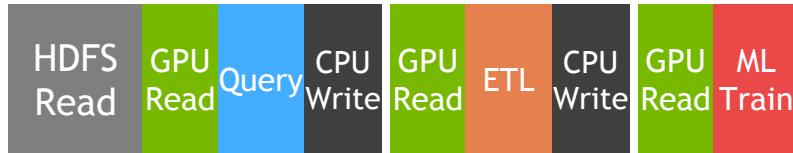
Hadoop Processing, Reading from disk



Spark In-Memory Processing



Traditional GPU Processing

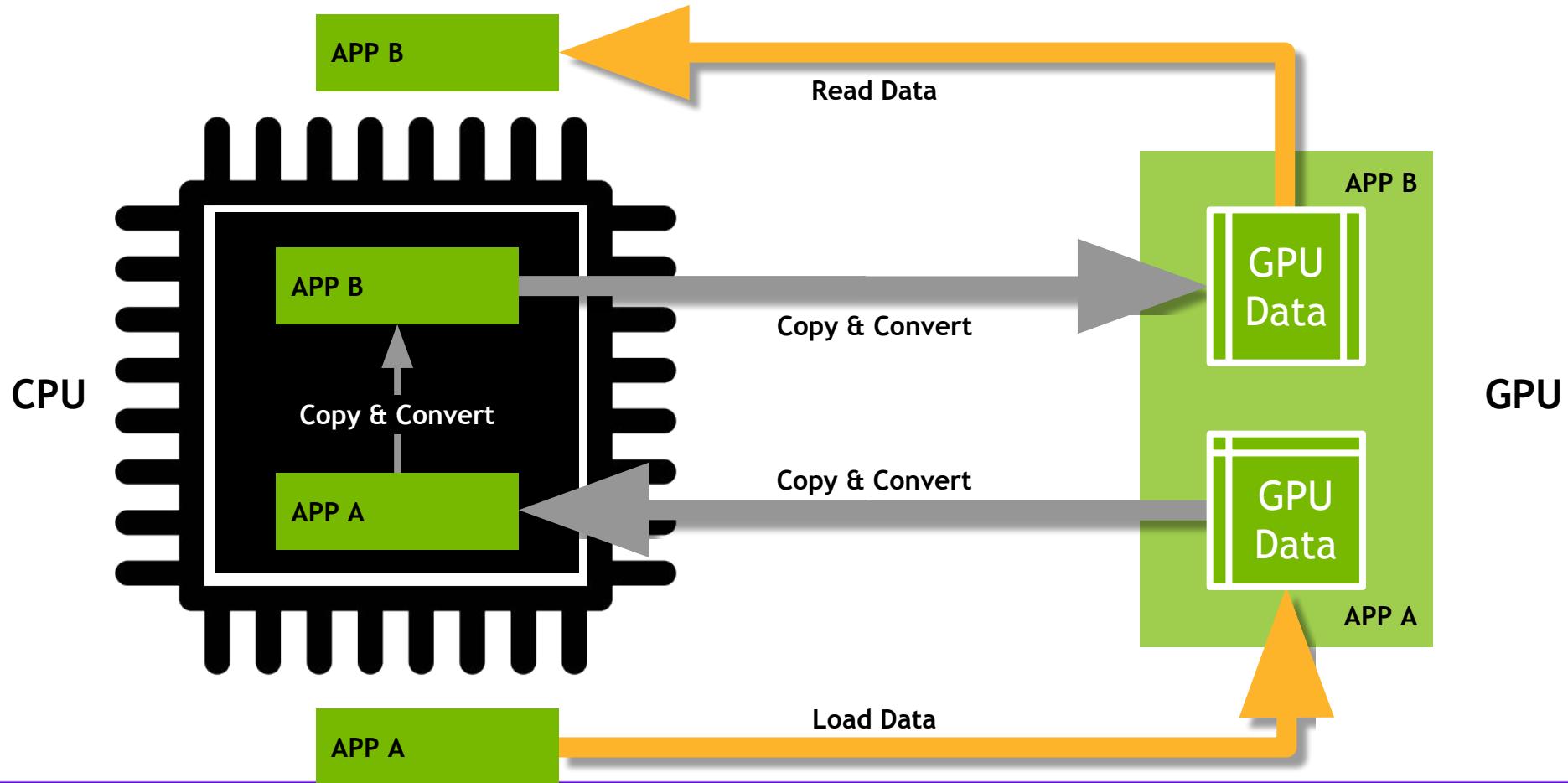


25-100x Improvement
Less code
Language flexible
Primarily In-Memory

5-10x Improvement
More code
Language rigid
Substantially on GPU

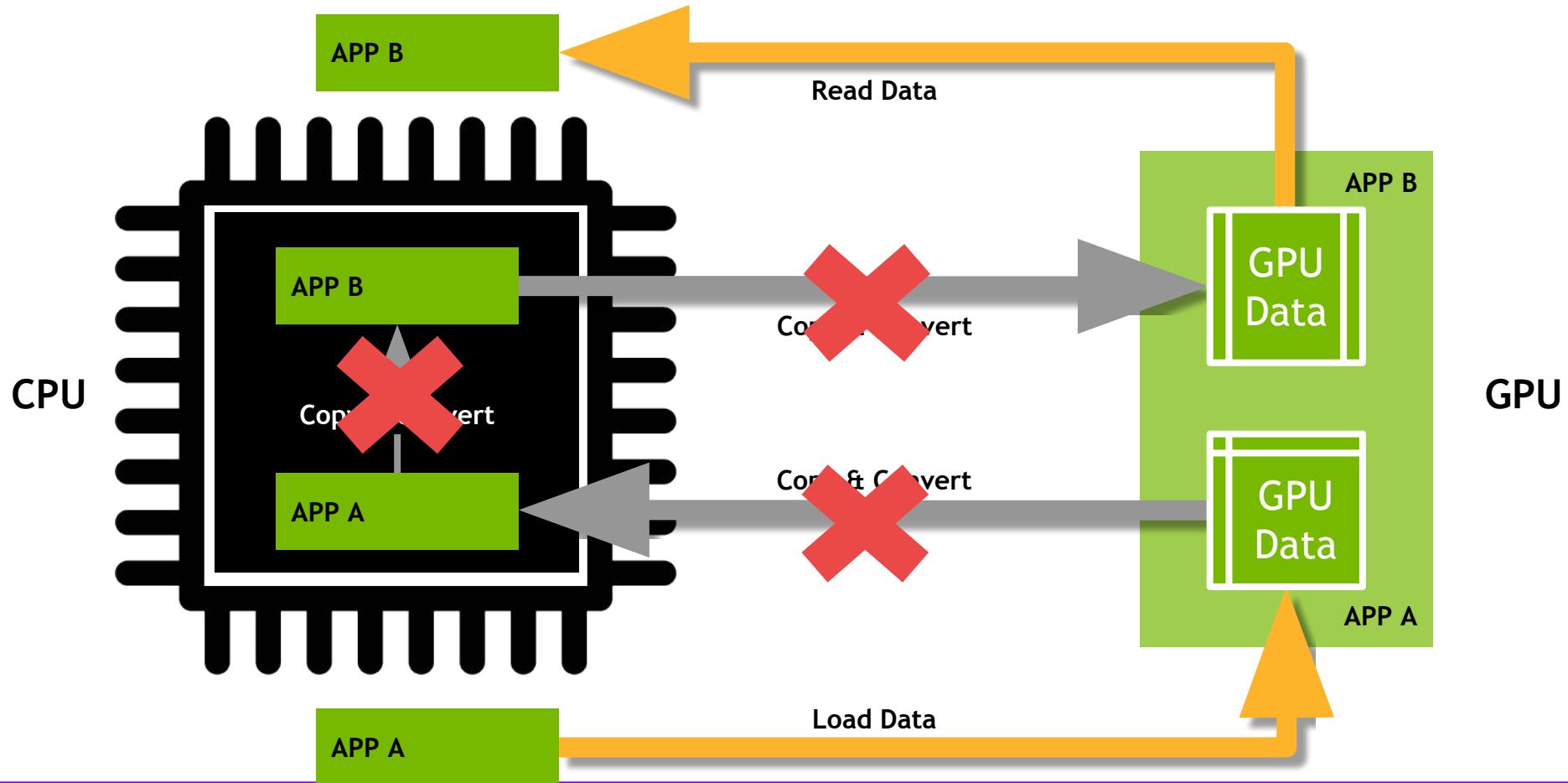
Data Movement and Transformation

The bane of productivity and performance

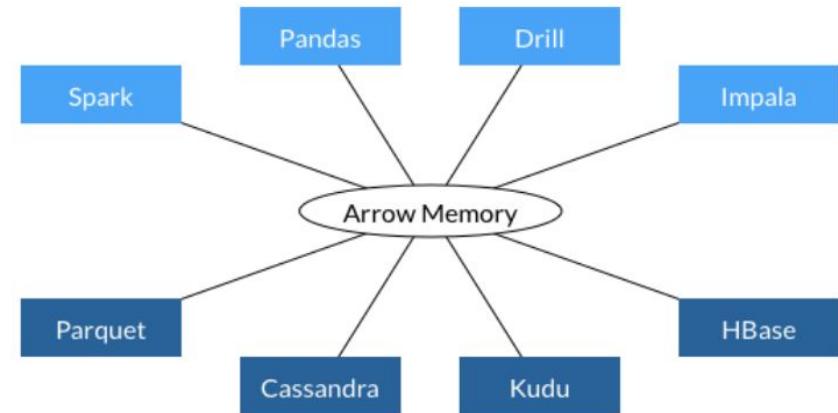
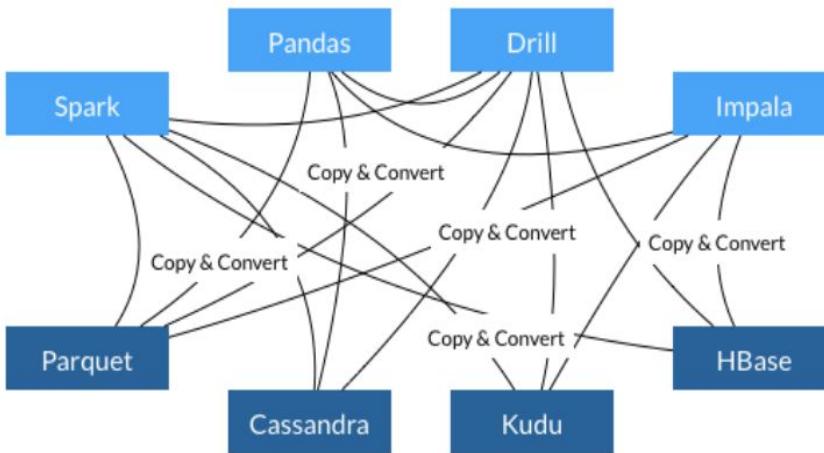


Data Movement and Transformation

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

Hadoop Processing, Reading from disk

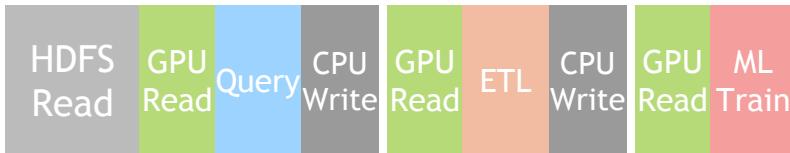


Spark In-Memory Processing



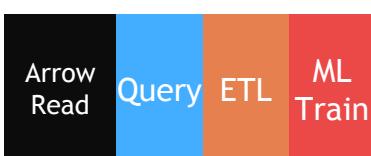
25-100x Improvement
Less code
Language flexible
Primarily In-Memory

Traditional GPU Processing



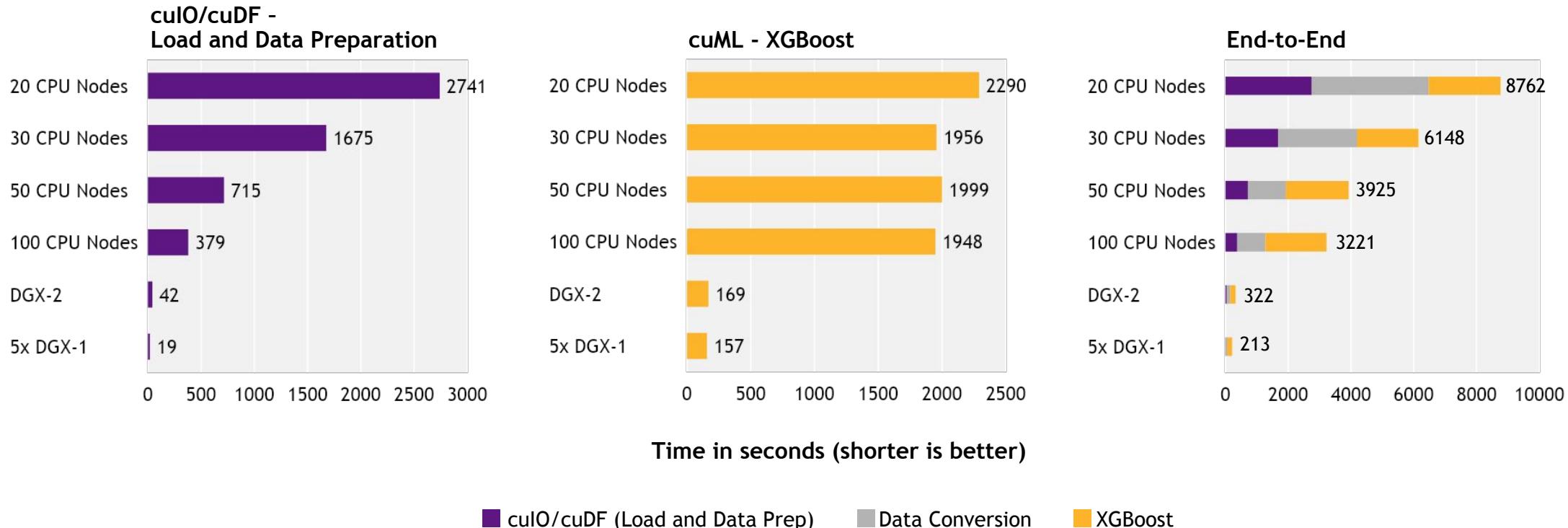
5-10x Improvement
More code
Language rigid
Substantially on GPU

RAPIDS



50-100x Improvement
Same code
Language flexible
Primarily on GPU

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 where Keras gives you an edge.

12:31 PM - 3 April 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.

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.

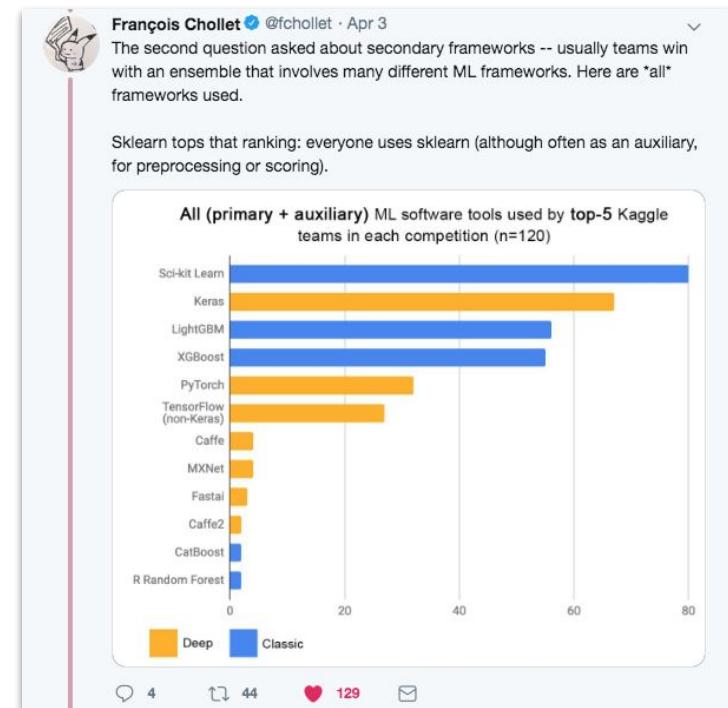
8 8 74

Joshua Patterson @datametrician · Apr 3

Replying to @fchollet

This is the fundamental belief 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

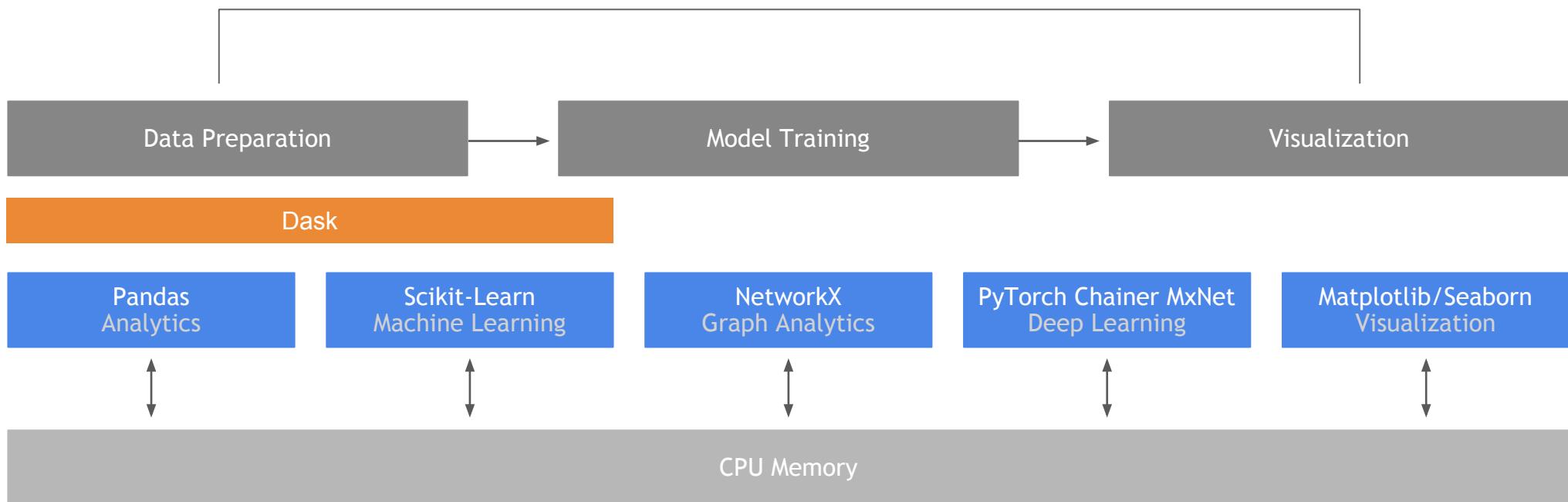


kaggle

RAPIDS Core

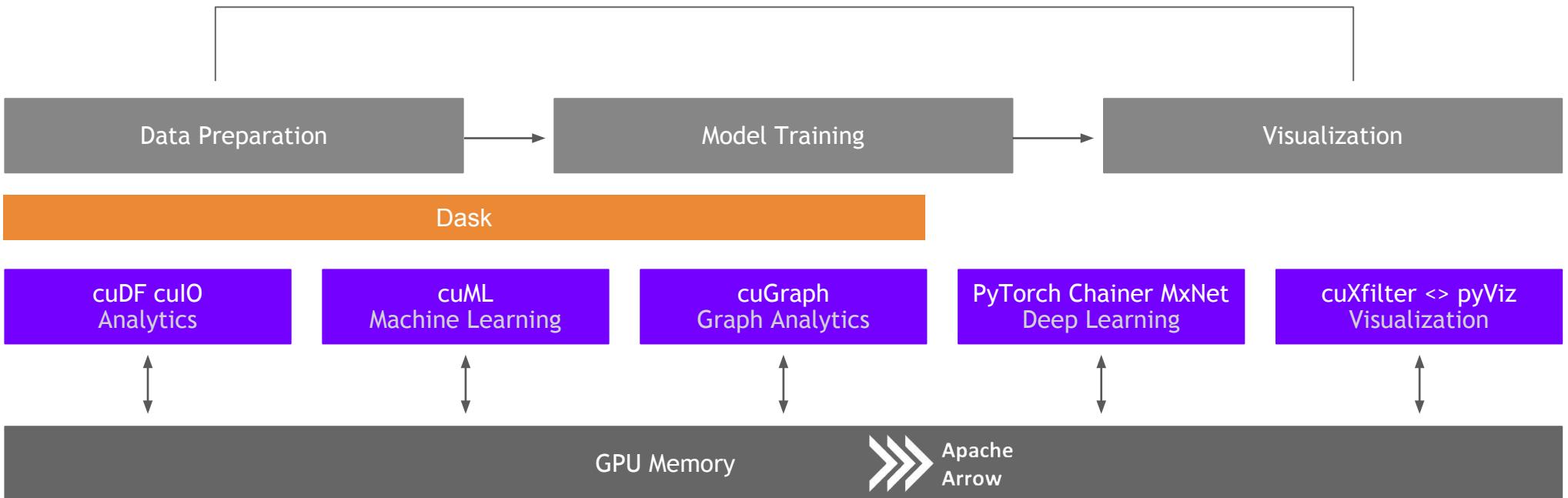
Open Source Data Science Ecosystem

Familiar Python APIs



RAPIDS

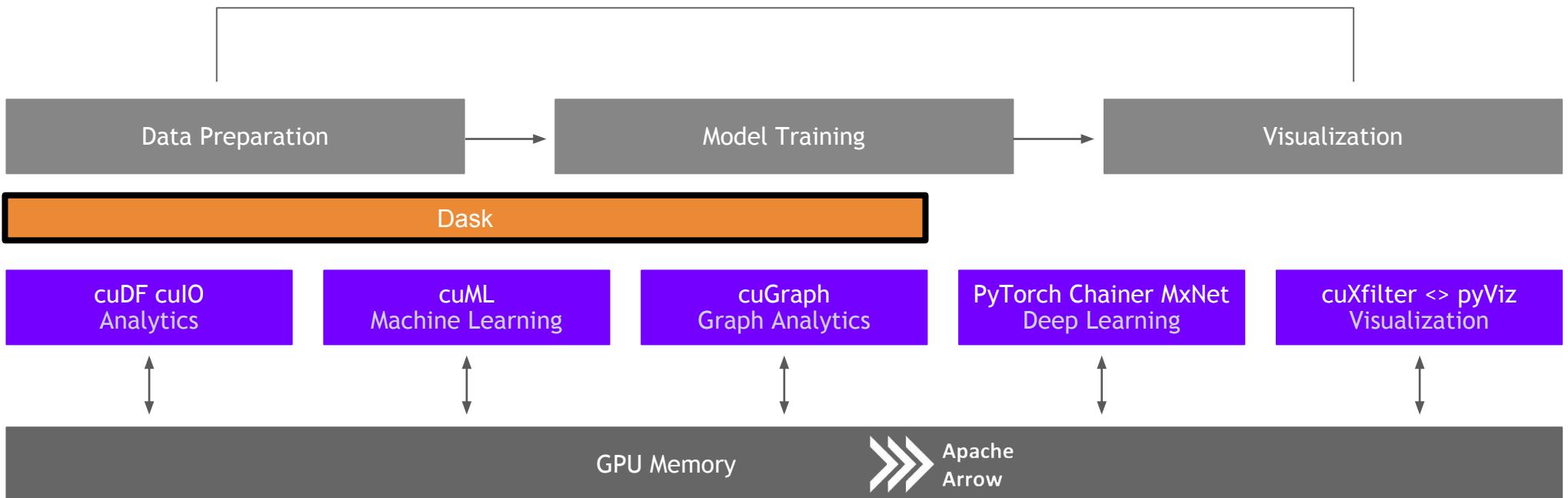
End-to-End Accelerated GPU Data Science



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

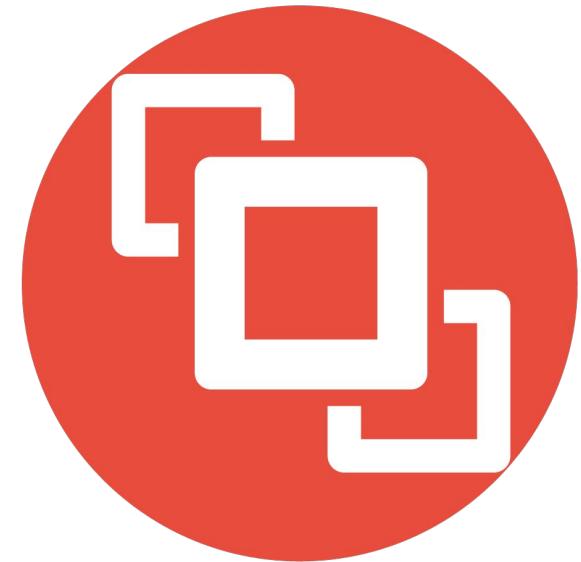
Popular

- 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)
- 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

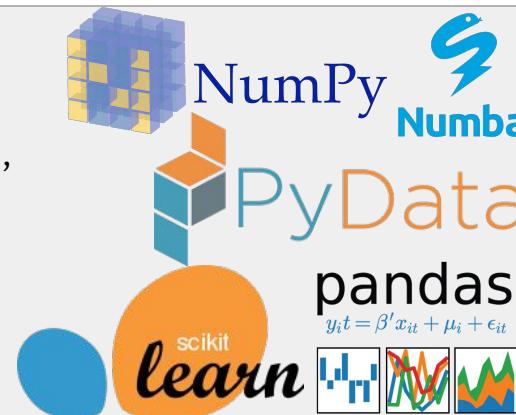
NumPy -> CuPy/PyTorch/..
Pandas -> cuDF
Scikit-Learn -> cuML
Numba -> Numba



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 + Dask with OpenUCX

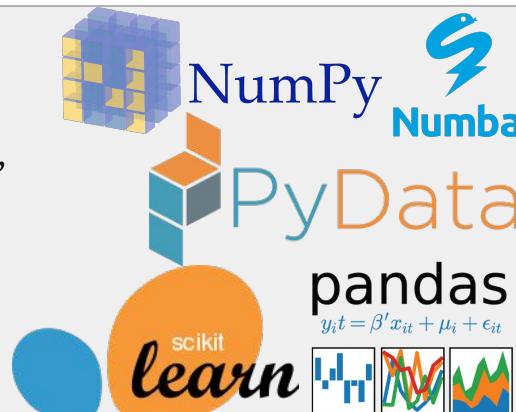
Multi-GPU
On single Node (DGX)
Or across a cluster



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

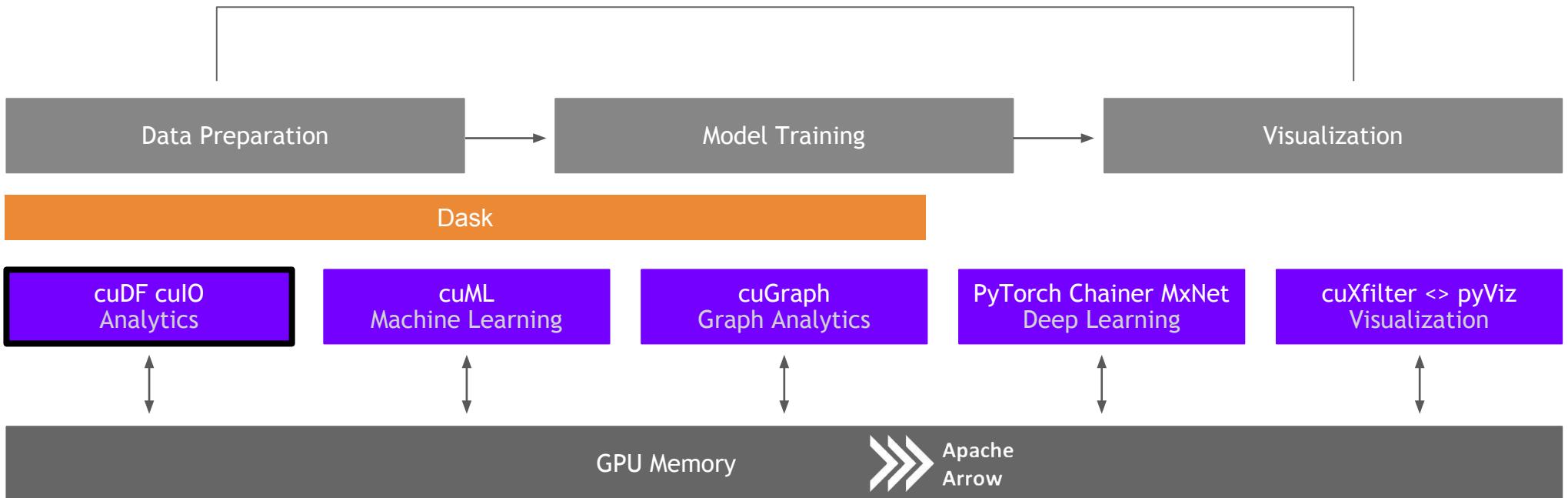


Scale out / Parallelize

cuDF

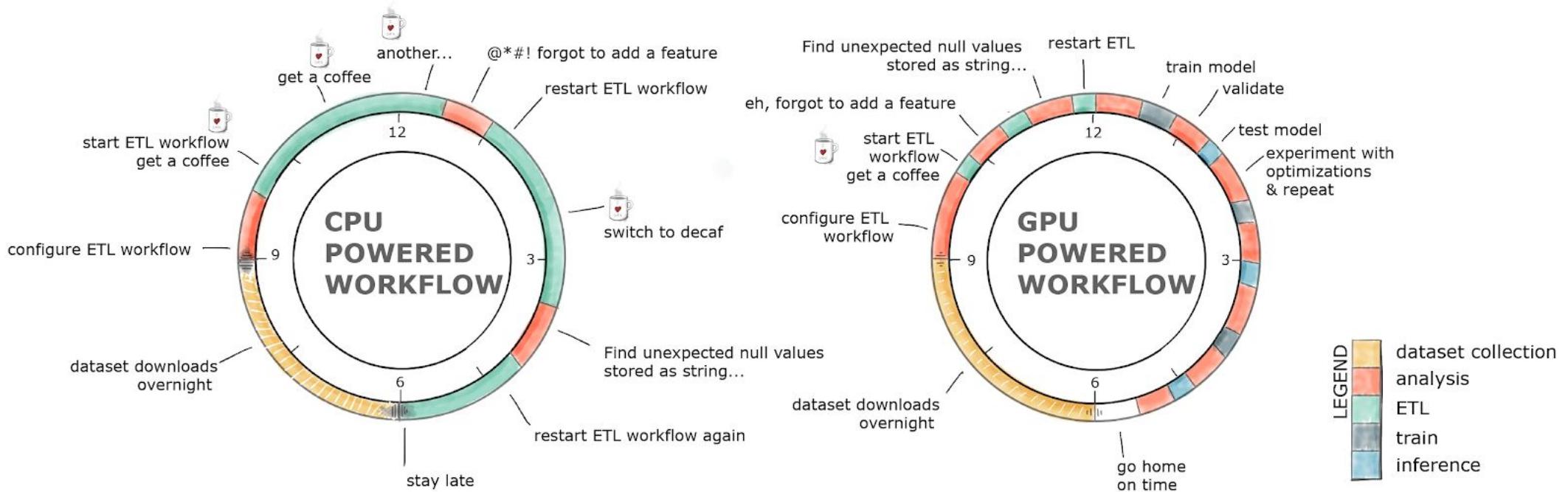
RAPIDS

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



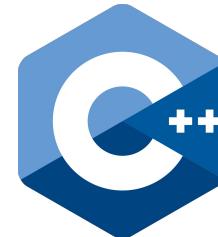
ETL - the Backbone of Data Science

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

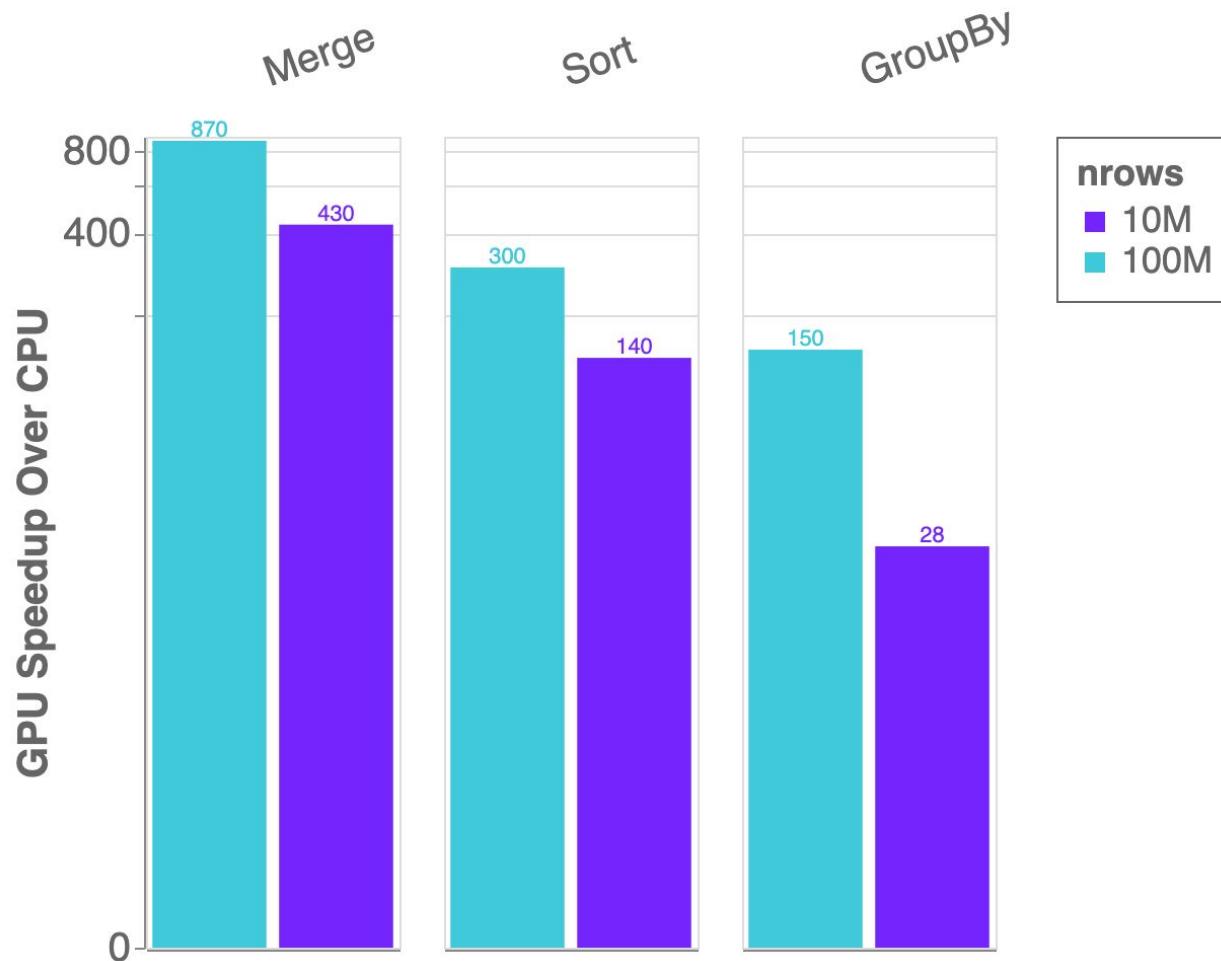
```
In [6]: #grabbing the first character of the years in city string to get rid of plus sign, and converting  
#to int  
gdf['city_years'] = gdf.Stay_In_Current_City_Years.str.get(0).stoi()
```

```
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

Benchmarks: single-GPU Speedup vs. Pandas



cuDF v0.9, 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:

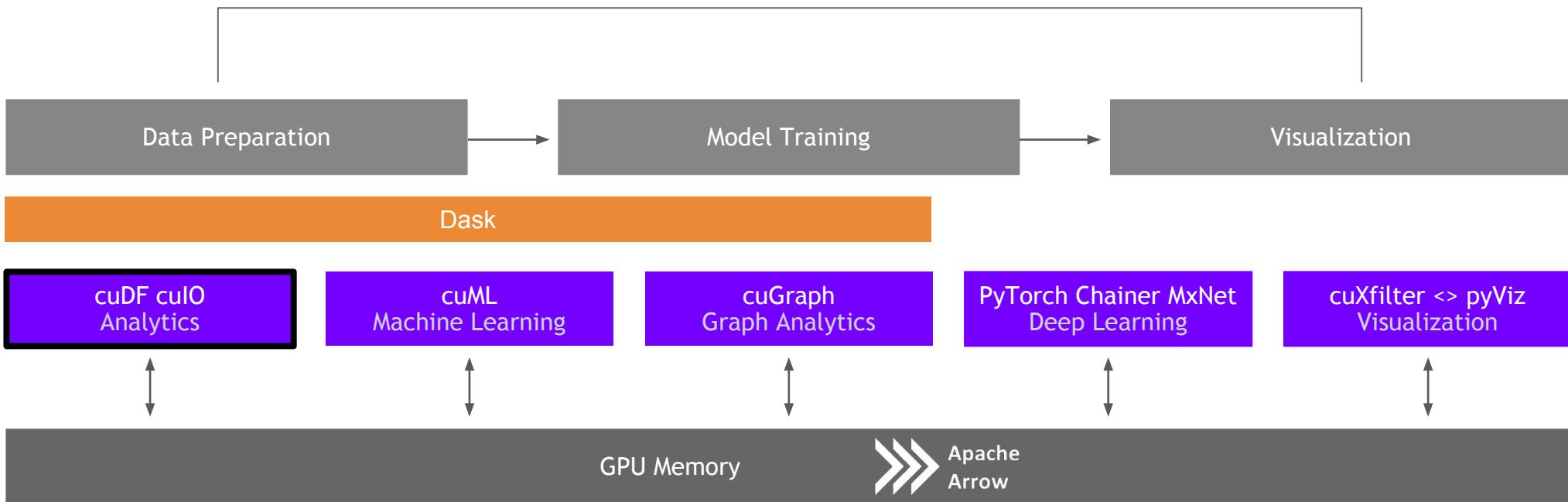
DataFrames: 2x int32 columns key columns,
3x int32 value columns

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



ETL - the Backbone of Data Science

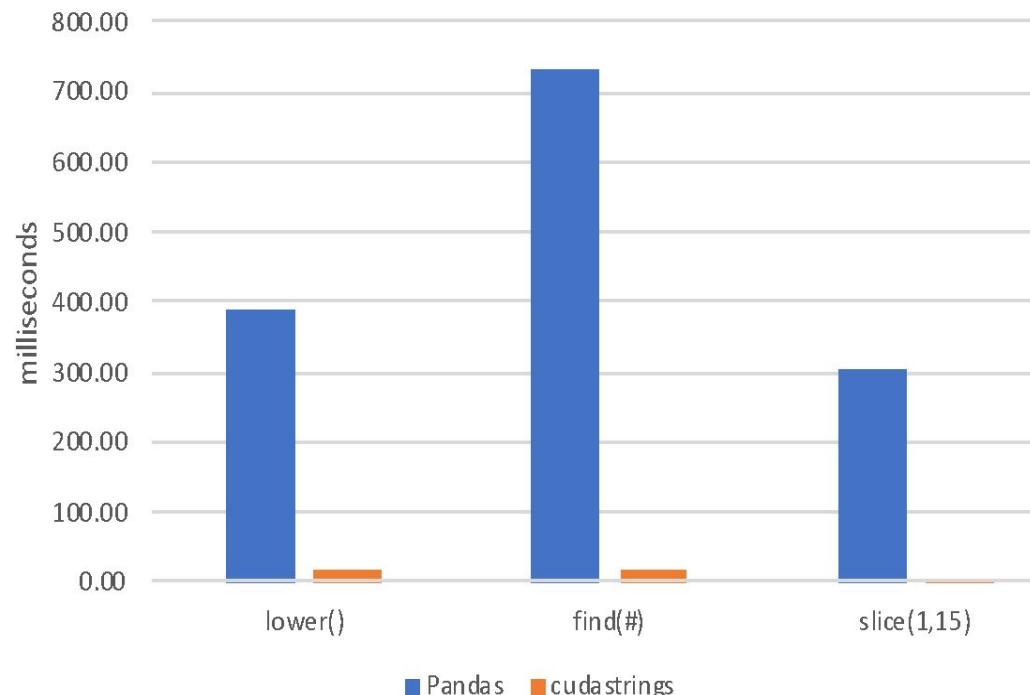
String Support

Current v0.9 String Support

- Regular Expressions
- Element-wise operations
 - Split, Find, Extract, Cat, Typecasting, etc...
- String GroupBys, Joins
- Categorical columns fully on GPU

Future v0.10+ String Support

- Combining cuStrings into libcudf
- Extensive performance optimization
- More Pandas String API compatibility
- JIT-compiled String UDFs



Extraction is the Cornerstone

cudf 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 v0.10
- 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

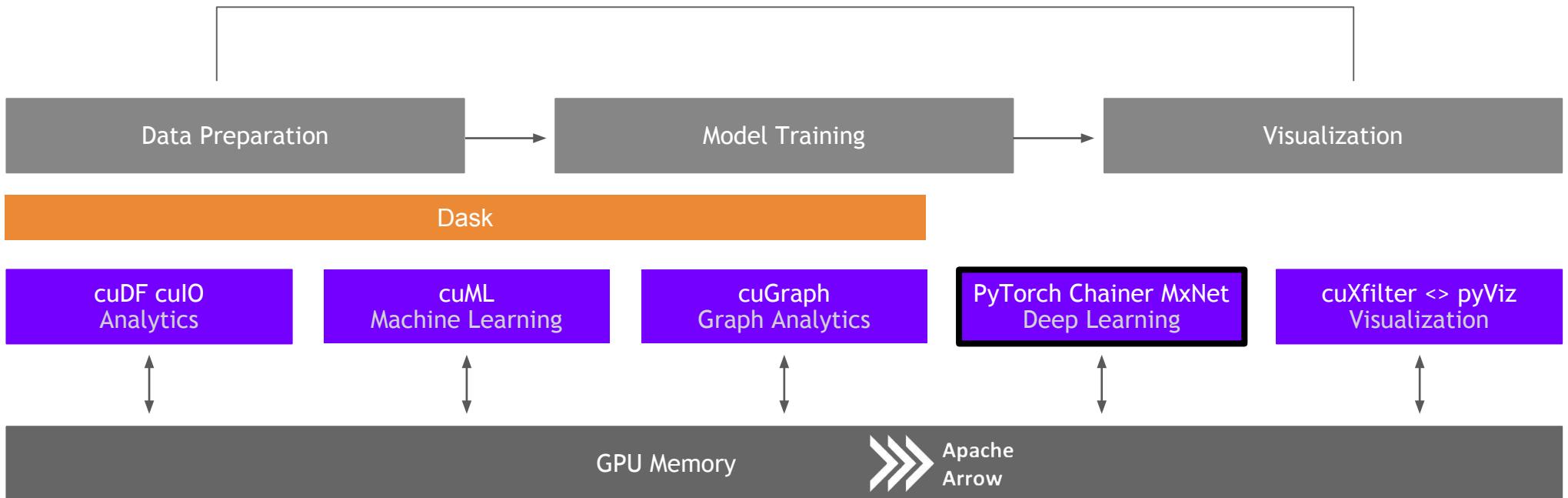
```
1]: import pandas, cudf
2]: %time len(pandas.read_csv('data/nyc/yellow_tripdata_2015-01.csv'))
CPU times: user 25.9 s, sys: 3.26 s, total: 29.2 s
Wall time: 29.2 s
2]: 12748986
3]: %time len(cudf.read_csv('data/nyc/yellow_tripdata_2015-01.csv'))
CPU times: user 1.59 s, sys: 372 ms, total: 1.96 s
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

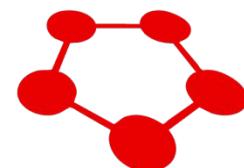
DLPack and __cuda_array_interface__



mpi4py



mxnet



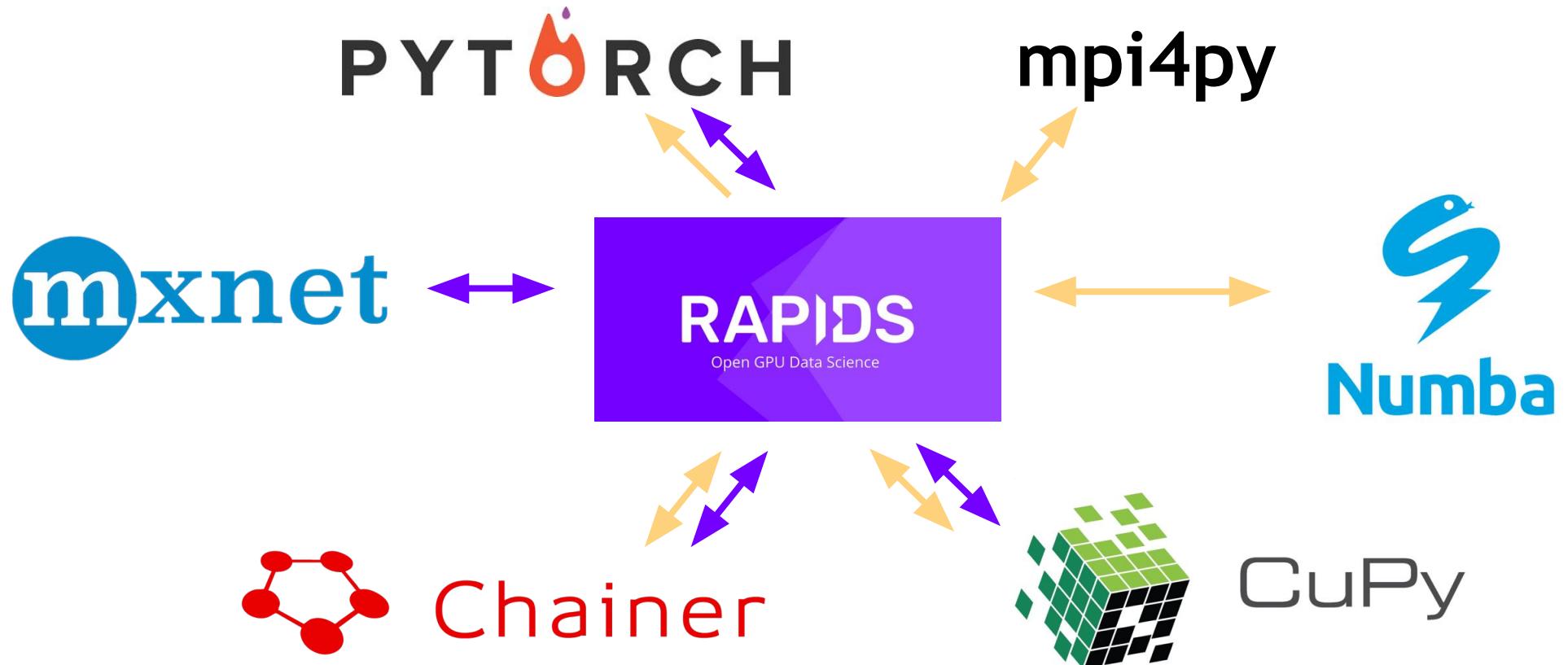
Chainer



CuPy

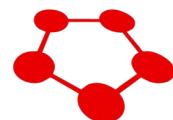
Interoperability for the Win

DLPack and `__cuda_array_interface__`



ETL - Arrays and DataFrames

Dask and CUDA Python arrays



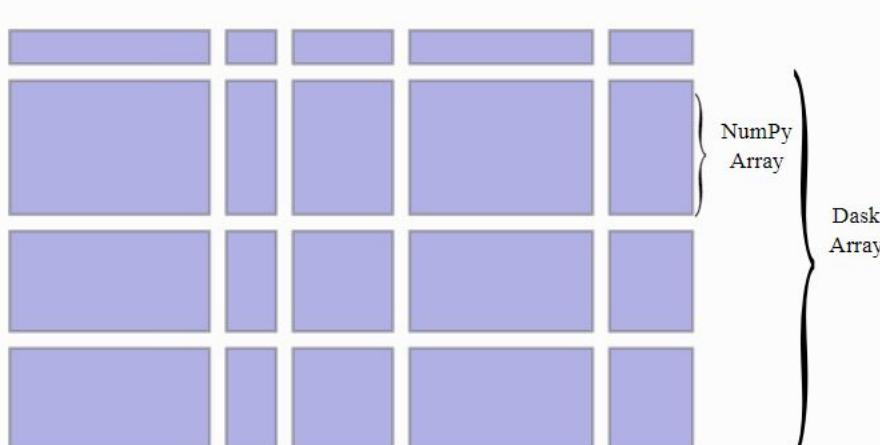
Chainer



CuPy

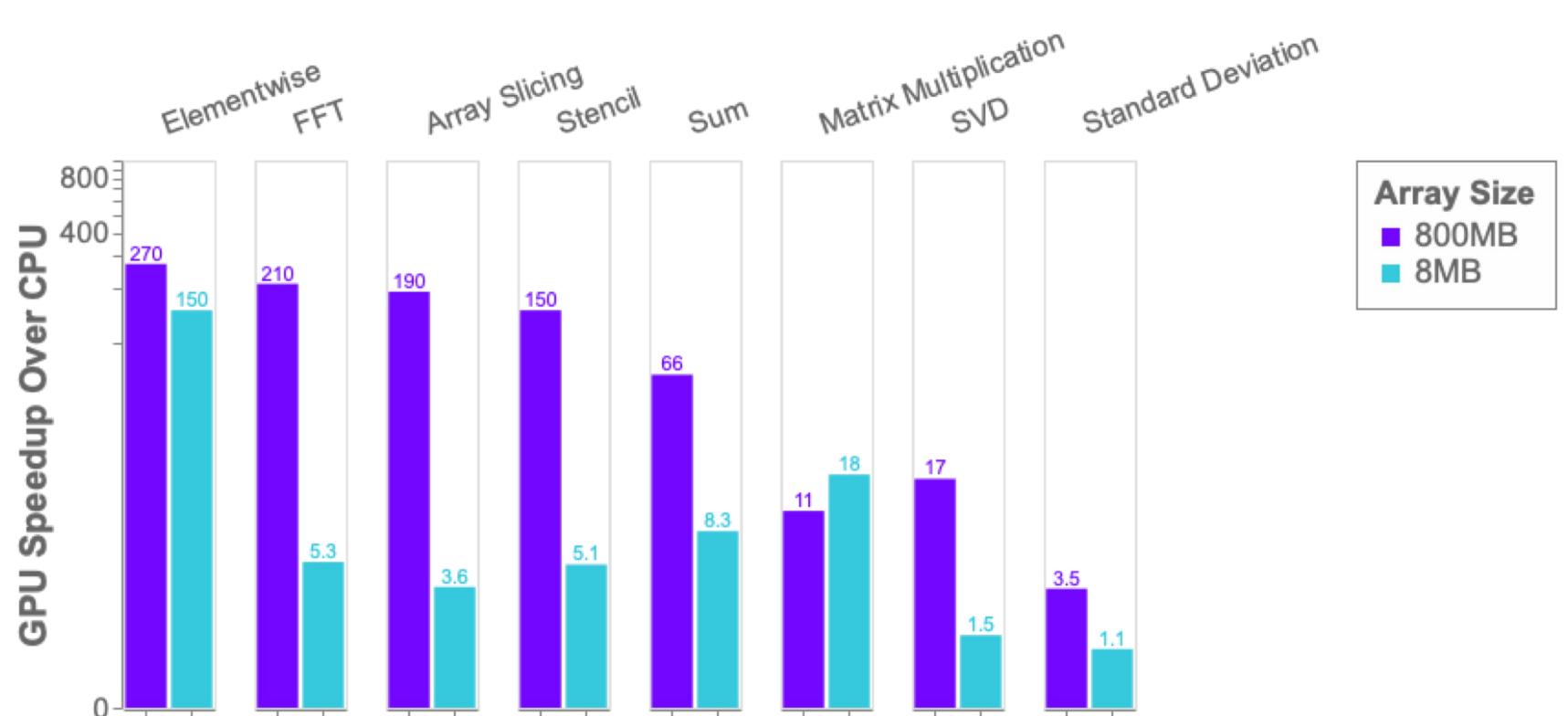


Numba



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

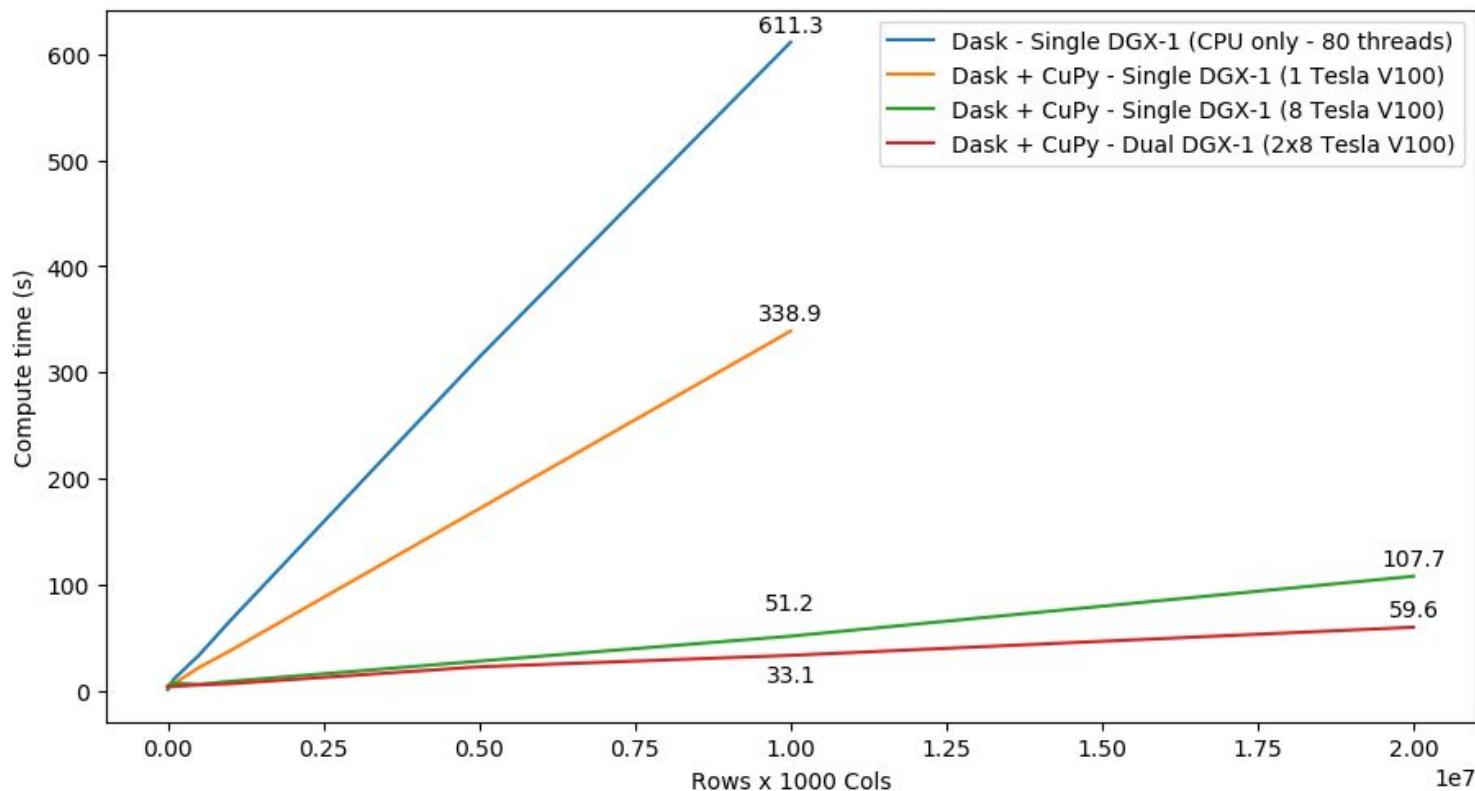
Benchmark: single-GPU CuPy vs NumPy



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

RAPIDS-Dask and cuDF NYCTaxi Screencast

The screenshot shows a Jupyter Notebook interface with several code cells and visualizations. The notebook title is "RAPIDS-Dask and cuDF NYCTaxi Screencast". The first cell contains code for reading a CSV file and performing a groupby operation. The second cell shows a "Time full-pass computation" section with a note about reading data from disk. The third cell shows a "How well do people tip?" section with code for filtering and plotting tip data by hour. The fourth cell shows a "Task Stream" visualization with a timeline from 2s to 12s, showing multiple parallel tasks. The fifth cell shows a "Dask Progress" bar with various stages of processing. The bottom of the screen has a red progress bar and a "3:13 / 4:10" timestamp.

```
3.092469787597656, 4, 3.096337099625, 2, 14, 0.5, 0.5, 0, 0, 0, 15, 3  
1, 2015-01-10 20:33:39, 2015-01-10 20:42:28, 28, 3, .88, -.74, 0.026262586914086, 40, 734142383466797, 1, N, -  
73, 0.95818375976563, 48, 726325988769531, 1, 7, 0.5, 0.5, 1.06, 0, 0, 3, 9, .96  
1, 2015-01-10 20:33:39, 2015-01-10 21:11:13, 3, 18, 28, -.73, 783842967114844, 40, 644355773925781, 2,  
N, -.73, 0.0734644492107, 48, 739357432392578, 2, 24, 0, 0, 3, 0, 1, -3, 9, 2, 38, 13  
(1): import dask_cudf  
(2): # df = dask_cudf.read_csv('gs://anaconda-public-data/nyc-taxi/csv/2015/yellow_*.csv')  
df = dask_cudf.read_csv('data/nyc/yellow_tripdata_2015-*.csv')  
df = df.persist()  
  
Time full-pass computation  
  
Most of the time here is spent reading data from disk and parsing it.  
  
(4): %time df.passenger_count.sum().compute()  
CPU times: user 100 ms, sys: 28 ms, total: 128 ms  
Wall time: 137 ms  
(4): 245566747  
  
(5): %time df.groupby('passenger_count').trip_distance.mean().compute()  
  
(6): df.to_pandas()  
  
How well do people tip?  
  
(7): df2 = df[['trip_pickup_datetime', 'trip_distance', 'tip_amount', 'fare_amount']]  
df2['tip_amount'] = df2['tip_amount'] > 0  
df2['hour'] = df2['trip_pickup_datetime'].dt.hour.astype('int32')  
df2['tip_fraction'] = df2['tip_amount'] / df2['fare_amount']  
hour = df2.groupby('hour').tip_fraction.mean().to_pandas()  
  
(8): %matplotlib inline  
hour.plot(figsize=(10, 6), title="Tip Fraction by Hour")  
  
Task Stream  
  
getitem-read-csv-query: 488.45 ms  
  
Dask Progress  
  
Progress - total: 1032, in-memory: 93, processing: 1, errred: 0  
assign 182 / 182 series-group... 91 / 91  
getitem 182 / 182 series-group... 14 / 14  
read-csv 91 / 91 series-group... 14 / 14  
getitem-read... 91 / 91 series-group... 1 / 1  
query 91 / 91 series-group... 1 / 1  
astype-dt-ho... 91 / 91 truediv 0 / 1  
truediv-session 91 / 91 series-group... 91 / 91  
  
3:13 / 4:10
```

RAPIDS-Dask and CuPy SVD Screencast

The screenshot shows a Jupyter Notebook interface with several code cells and visualizations. The notebook title is "RAPIDS-Dask and CuPy SVD Screencast". The first cell shows code for creating a random dataset. The second cell shows a "Singular Value Decomposition" section with a note about communication overhead. The third cell shows a "Inspect output" section with code for printing memory usage. The fourth cell shows a "Task Stream" visualization with a timeline from 10s to 20s, showing tasks for getitem, dot, and transfer-dot. The bottom of the screen has a red progress bar and a "3:12 / 3:42" timestamp.

```
import dask  
import dask.array  
import numpy  
import cupy  
  
rs = dask.array.random(1000000, 1000, chunks=(1000, 1000))  
x = rs.random(1000000, 1000, chunks=(1000, 1000))  
x = x.persist()  
  
Create Random Dataset  
  
(2): import dask.array.linalg  
u, s, v = dask.array.linalg.svd(x)  
  
(3): dask.visualize(u, s, v)  
  
(4): u, s, v = dask.persist(u, s, v)  
  
Singular Value Decomposition  
  
This computes SVD on GPU and has some communication heavy steps.  
  
(5): import dask.array.linalg  
u, s, v = dask.array.linalg.svd(x)  
  
(6): dask.visualize(u, s, v)  
  
(7): u, s, v = dask.persist(u, s, v)  
  
Inspect output  
  
(8): print(u[10, :10].compute())  
print(s[10].compute())  
print(v[10, :10].compute())  
  
(9): from distributed.utils import format_bytes  
  
(10): print(format_bytes(u.nbytes))  
print(format_bytes(s.nbytes))  
print(format_bytes(v.nbytes))  
  
Progress - total: 756, in-memory: 205, processing: 12, errred: 0  
getitem 424 / 434  
dot 100 / 100  
transfer-dot... 100 / 100  
qr 11 / 11  
stack-di... 11 / 11  
getitem-in... 1 / 1  
  
3:12 / 3:42
```

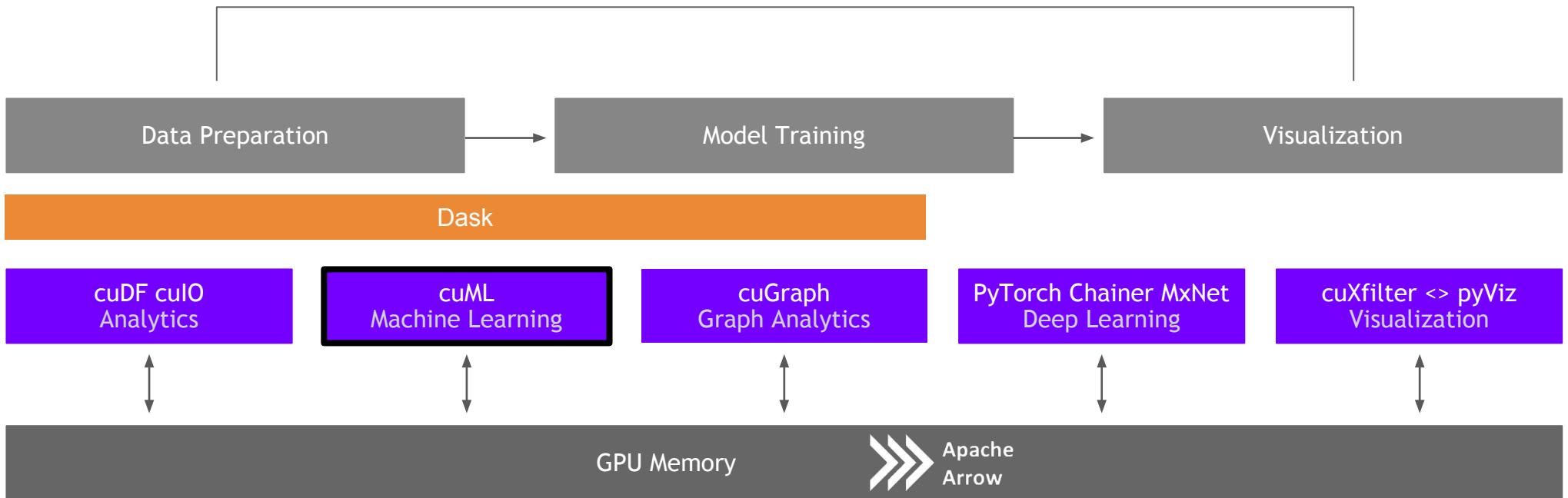
<https://youtu.be/gV0cykgsTPM>

https://youtu.be/R5CiXti_MWo

cuML

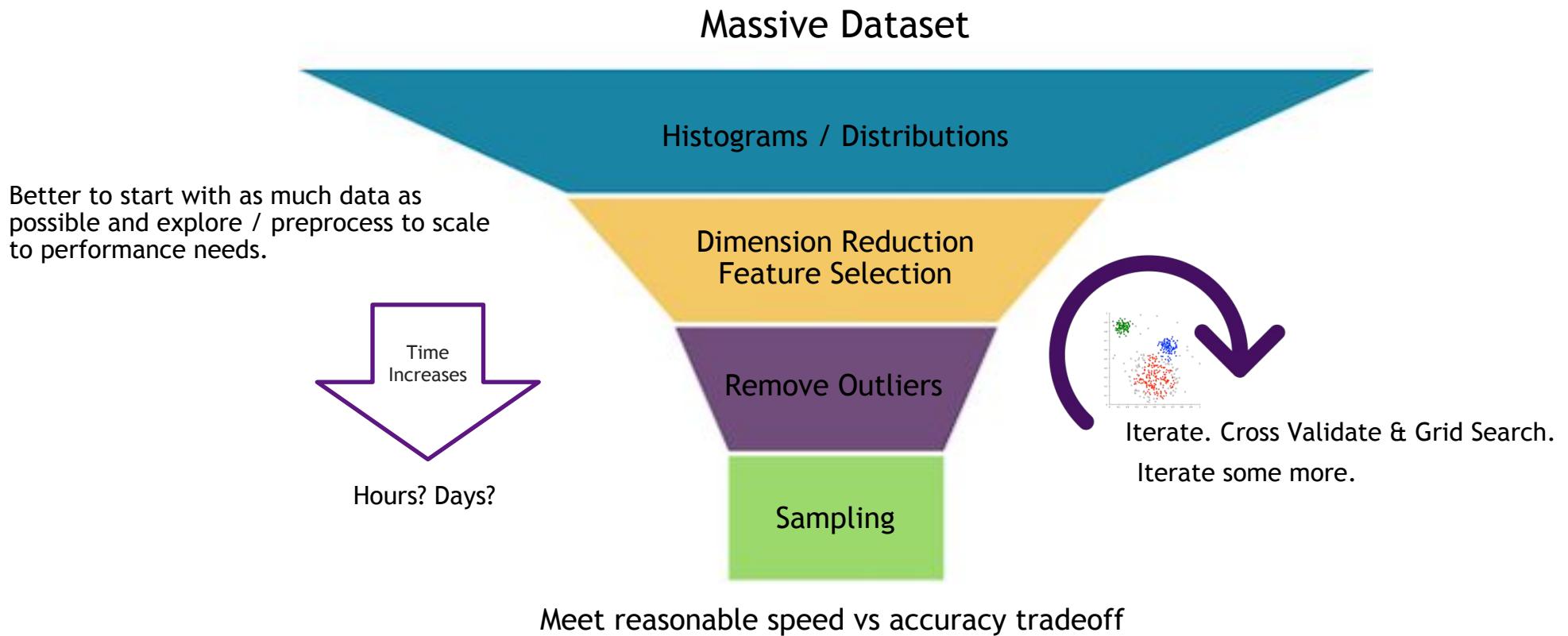
Machine Learning

More models more problems

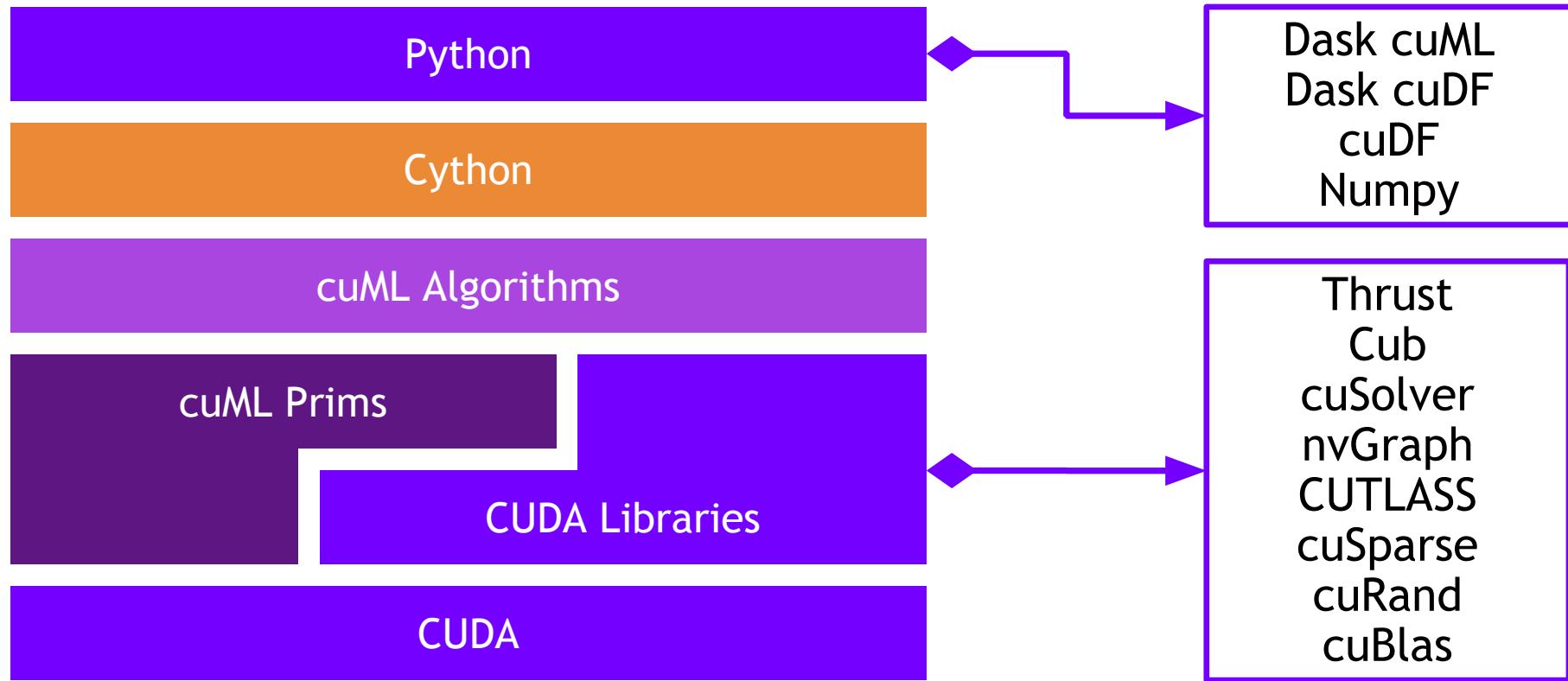


Problem

Data sizes continue to grow

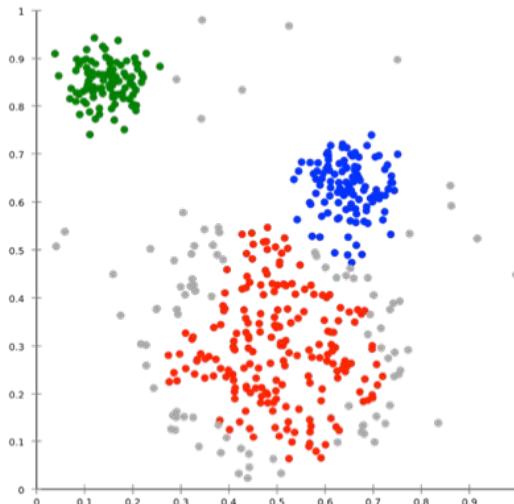


ML Technology Stack



Algorithms

GPU-accelerated Scikit-Learn



Cross Validation

Hyper-parameter Tuning

More to come!

Classification / Regression

Inference

Clustering

Decomposition & Dimensionality Reduction

Time Series

Decision Trees / Random Forests
Linear Regression
Logistic Regression
K-Nearest Neighbors

Random forest / GBDT inference

K-Means
DBSCAN
Spectral Clustering

Principal Components
Singular Value Decomposition
UMAP
Spectral Embedding

Holt-Winters
Kalman Filtering

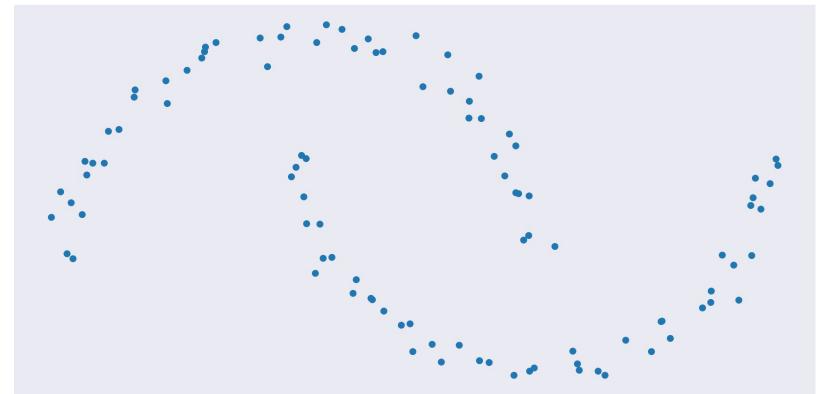
Key:

- Preexisting
- NEW for 0.9

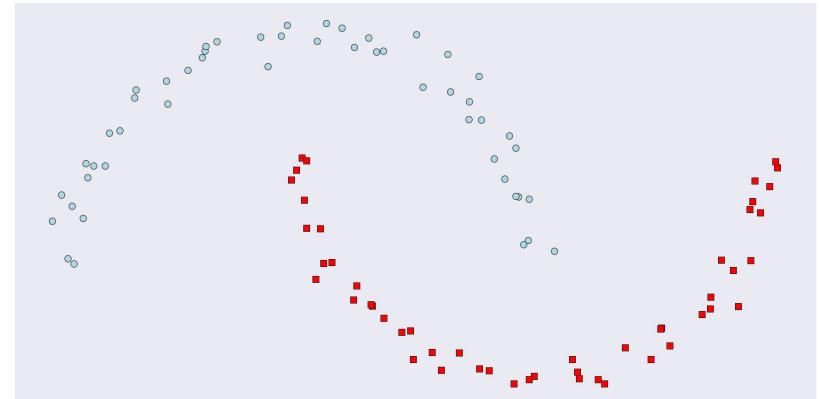
RAPIDS matches common Python APIs

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)
```



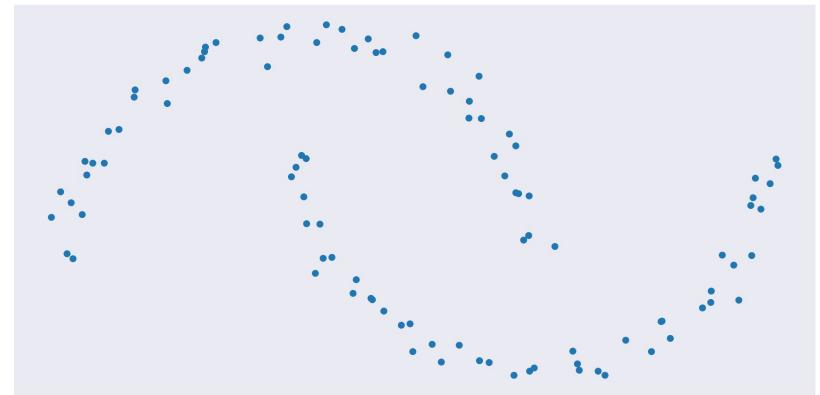
RAPIDS matches common Python APIs

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)

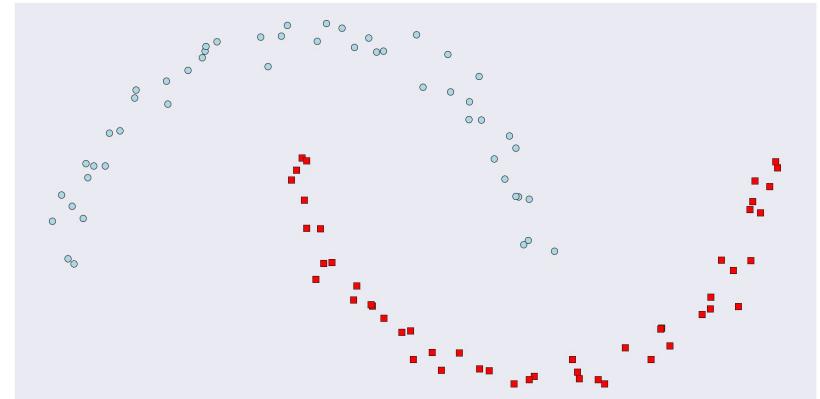
X = cudf.DataFrame({'fea%d'%i: X[:, i]
                    for i in range(X.shape[1])})
```



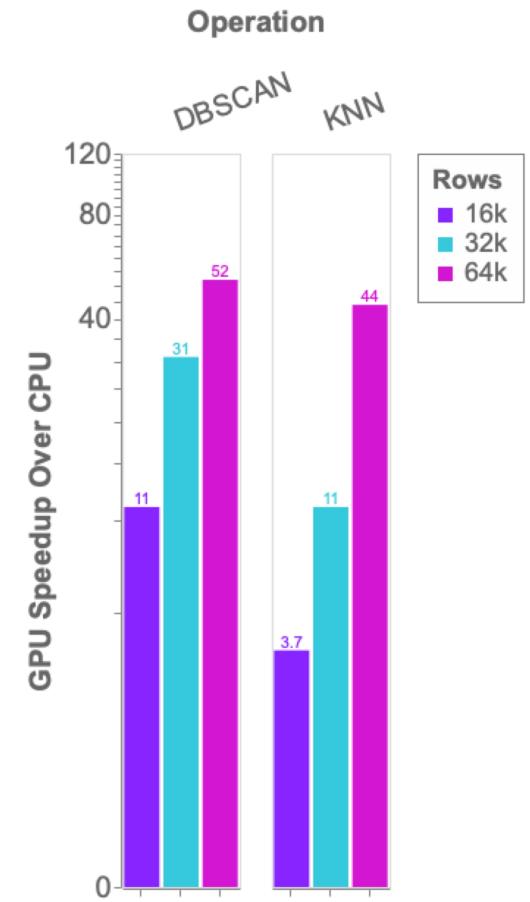
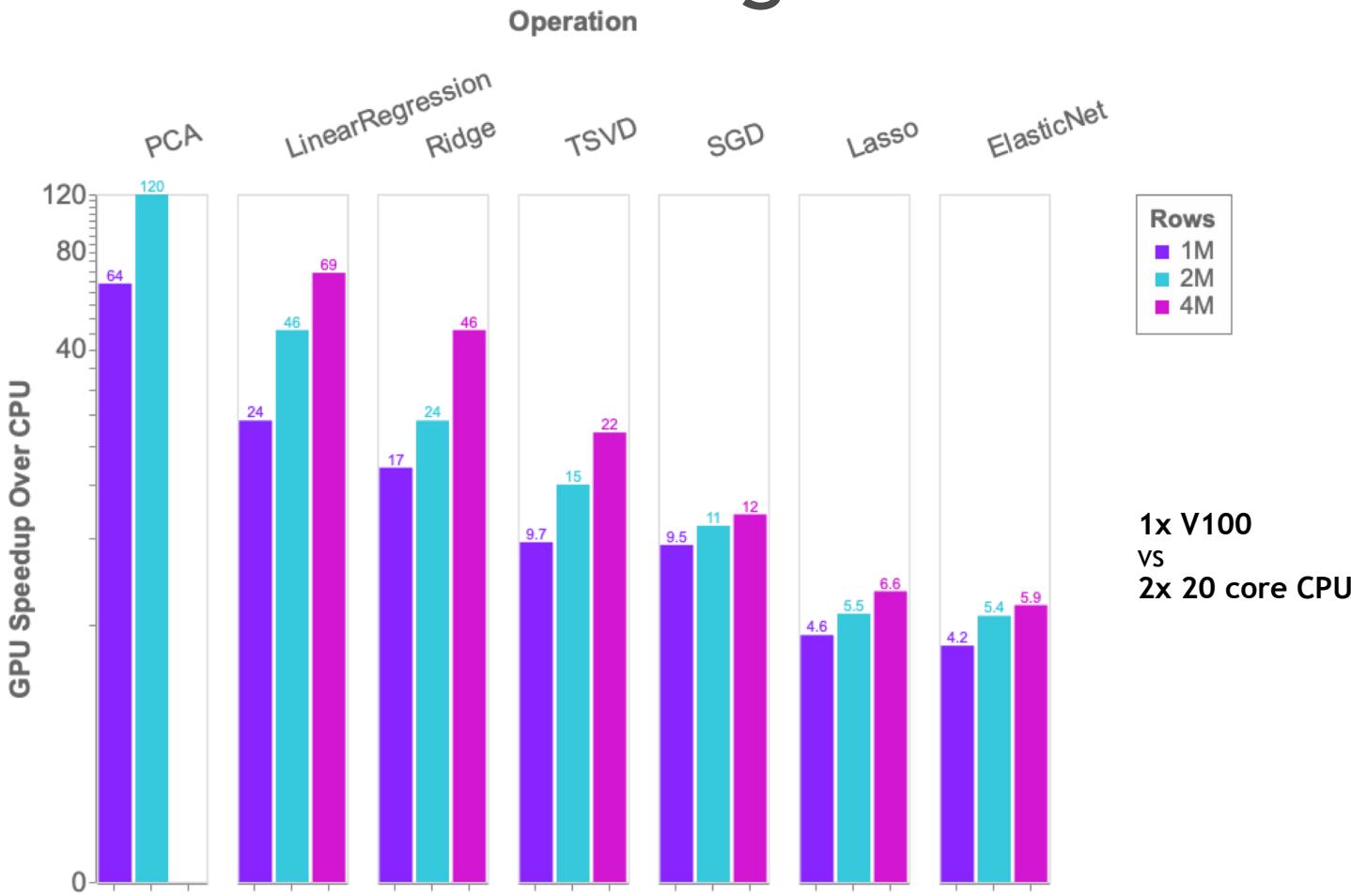
```
from cuml import DBSCAN
dbscan = DBSCAN(eps = 0.3, min_samples = 5)

dbscan.fit(X)

y_hat = dbscan.predict(X)
```



Benchmarks: single-GPU cuML vs scikit-learn

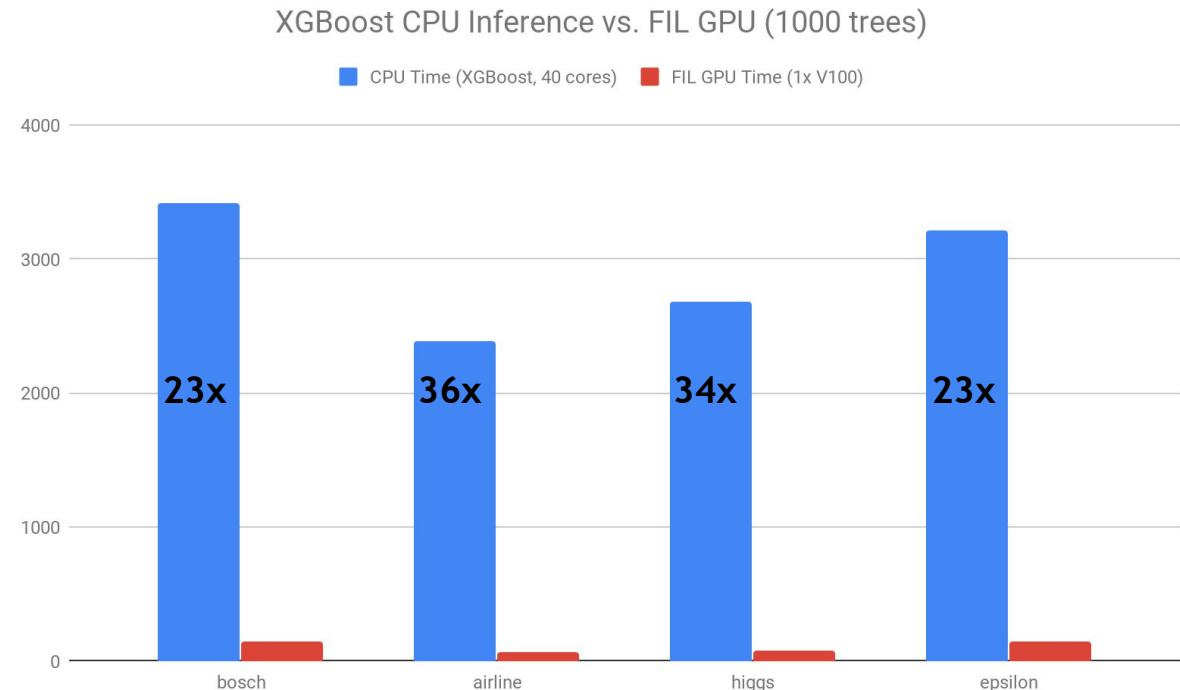


Forest Inference

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

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			
Holt-Winters			
Kalman Filter			
t-SNE			
Principal Components			
Singular Value Decomposition			

Road to 1.0

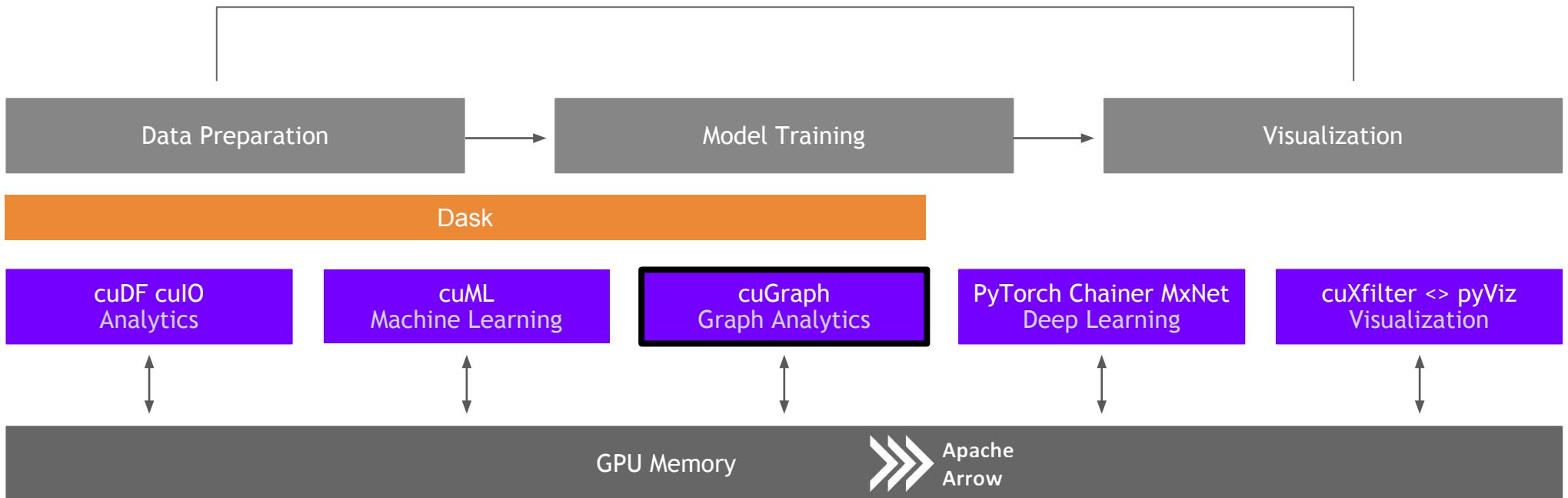
March 2020 - RAPIDS 0.14

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 & Holt-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 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

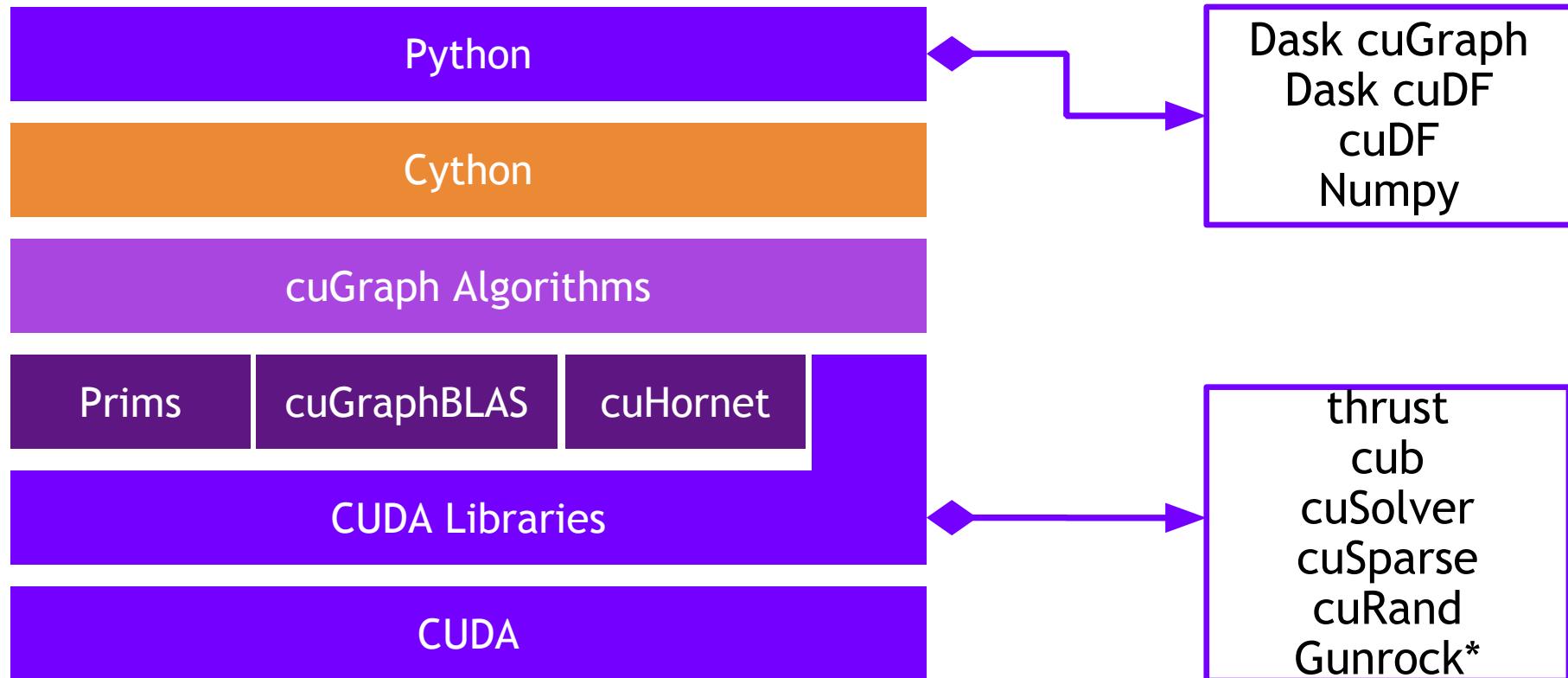
Seamless Integration with cuDF and cuML

- Property Graph support via DataFrames

Growing Functionality

- Extensive collection of algorithm, primitive, and utility functions

Graph Technology Stack

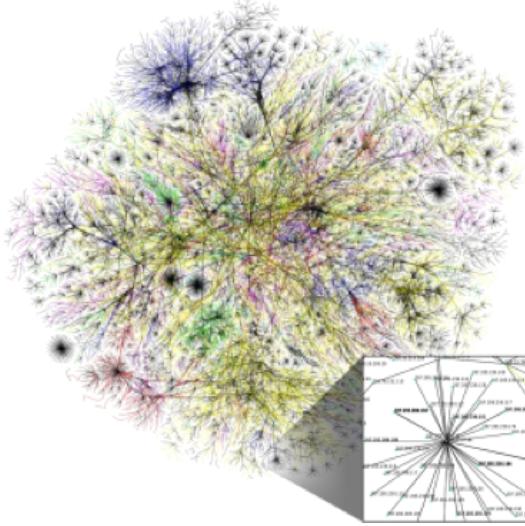


nvGRAPH has been Opened Sourced and integrated into cuGraph. A legacy version is available in a RAPIDS GitHub repo

* Gunrock is from UC Davis

Algorithms

GPU-accelerated NetworkX



Query Language

Multi-GPU

Utilities

More to come!

Community

Components

Link Analysis

Link Prediction

Traversal

Structure

Spectral Clustering
Balanced-Cut
Modularity Maximization
Louvain
Subgraph Extraction
Triangle Counting

Weakly Connected Components
Strongly Connected Components

Page Rank (Multi-GPU)
Personal Page Rank

Jaccard
Weighted Jaccard
Overlap Coefficient

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

COO-to-CSR (Multi-GPU)
Transpose
Renumbering

Louvain Single Run

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

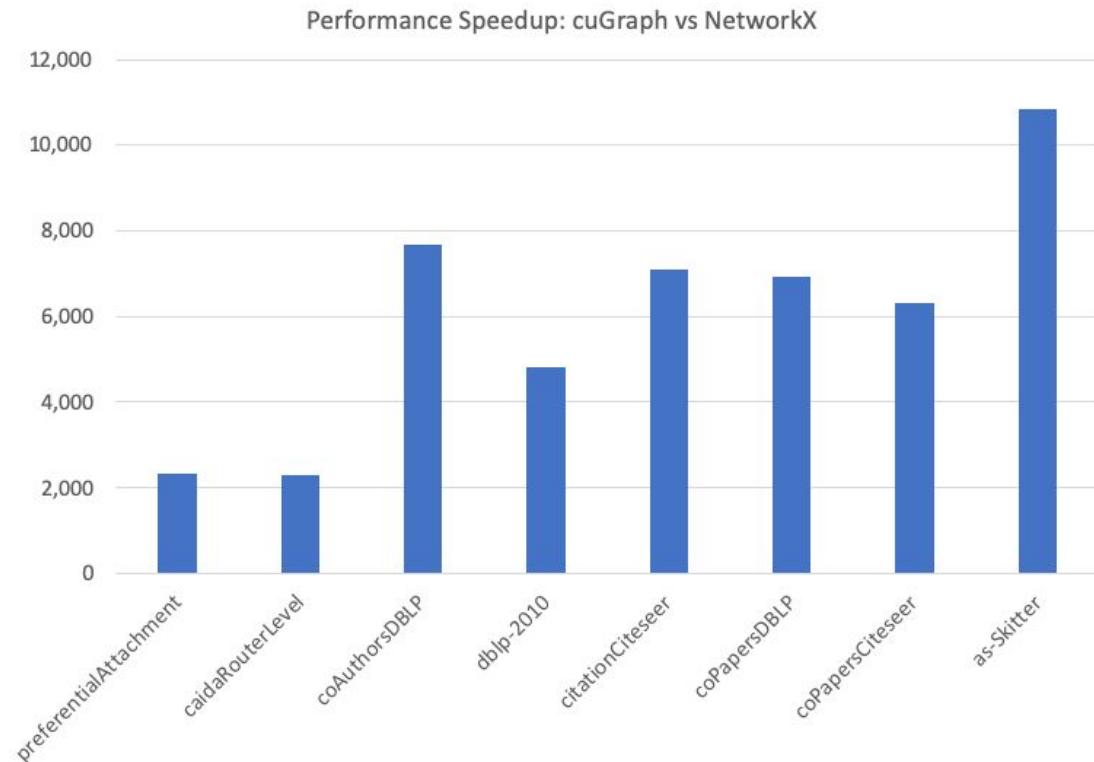
Louvain returns:

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



Multi-GPU PageRank Performance

PageRank portion of the HiBench benchmark suite

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

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

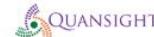
March 2020 - RAPIDS 0.14

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			

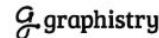
Community

Ecosystem Partners

CONTRIBUTORS



ADOPTERS

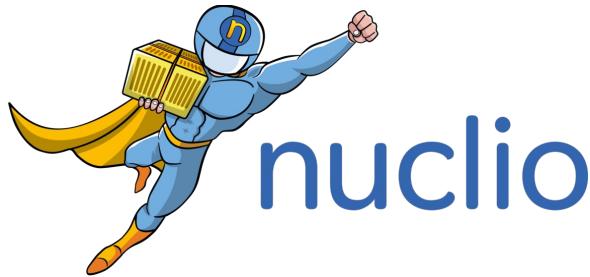


OPEN SOURCE



Building on top of RAPIDS

A bigger, better, stronger ecosystem for all



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



GPU accelerated SQL
engine built on top of
RAPIDS

Streamz

Distributed stream
processing using
RAPIDS and Dask

Deploy RAPIDS Everywhere

Focused on robust functionality, deployment, and user experience



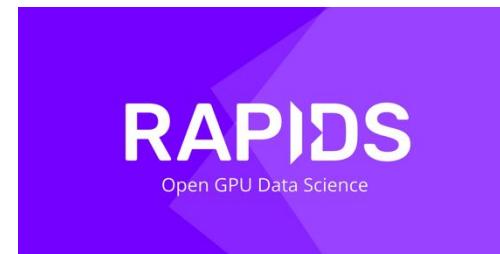
Google Cloud



Cloud Dataproc



Kubeflow



RAPIDS

Open GPU Data Science



Azure Machine Learning



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. Join our community conversations on [Slack](#), [Google](#), and [Twitter](#)
3. Explore our [walk through videos](#), [blog content](#), our [github](#), the [tutorial notebooks](#), and our [examples workflows](#),
4. Build your own data science workflows.
5. Contribute back. Don't forget to ask and answer questions on [Stack Overflow](#)

Easy Installation

Interactive Installation Guide

RAPIDS

HOME ABOUT GET STARTED COMMUNITY BLOG DOCS GITHUB

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 blow](#).

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	<pre>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</pre>			
COPY COMMAND 				

Join the Conversation



[Google Groups](#)



[Docker Hub](#)



[Slack Channel](#)



[Stack Overflow](#)

Explore: RAPIDS Github

<https://github.com/rapidsai>

The screenshot shows the GitHub homepage for the RAPIDS repository. The header includes links for Pull requests, Issues, Marketplace, and Explore. Below the header is the repository's profile card with the name "RAPIDS", a purple gradient background, and the tagline "Open GPU Data Science". A link to the website "http://rapids.ai" is also present. The navigation bar below the profile card includes tabs for Repositories (67), Packages, People (118), Teams (91), and Projects (6). The main content area is titled "Pinned repositories" and displays six repository cards:

- cudf**: cuDF - GPU DataFrame Library. Cuda, 1.9k stars, 270 forks.
- cuml**: cuML - RAPIDS Machine Learning Library. C++ (indicated by a pink dot), 665 stars, 119 forks.
- cugraph**: cuGraph - RAPIDS Graph Analytics Library. Cuda, 204 stars, 52 forks.
- notebooks**: RAPIDS Sample Notebooks. Jupyter Notebook, 204 stars, 94 forks.
- notebooks-contrib**: RAPIDS Community Notebooks. Jupyter Notebook, 106 stars, 76 forks.
- cuxfilter**: GPU accelerated cross filtering. Python, 31 stars, 14 forks.

Explore: RAPIDS Docs

Improved and easier to use!

The screenshot shows the cuDF Stable Docs API Reference page for DataFrame. The navigation bar includes links for cuDF, cuML, cuGraph, nvStrings, LIBS, libcuDF, RMM, NIGHTLY, LEGACY, and APIS. The main content area has a sidebar titled 'CONTENTS:' with sections like API Reference, DataFrame, Series, Groupby, IO, GpuArrowReader, 10 Minutes to cuDF, Multi-GPU with Dask-cuDF, and Developer Documentation. The main content area displays code snippets and descriptions for DataFrame creation and manipulation.

```
>>> import cudf
>>> df = cudf.DataFrame()
>>> df['key'] = [0, 1, 2, 3, 4]
>>> df['val1'] = [float(i + 10) for i in range(5)] # insert column
>>> print(df)
   key    val
0    0  10.0
1    1  11.0
2    2  12.0
3    3  13.0
4    4  14.0
```

```
>>> import cudf
>>> import numpy as np
>>> from datetime import datetime, timedelta
>>> ids = np.arange(5)
```

The screenshot shows the cuDF Stable Docs 10 Minutes to cuDF page. The navigation bar is identical to the first screenshot. The main content area features a sidebar with various topics like Object Creation, Viewing Data, Selection, Getting, Selection by Label, Selection by Position, Boolean Indexing, Setting, Missing Data, Operations, Stats, Applymap, Histogramming, String Methods, Merge, Concat, Join, Append, Grouping, Reshaping, Time Series, Categoricals, Plotting, and Converting Data Representation. The main content area contains code snippets and explanatory text for creating Series and DataFrame.

```
[1]: import os
import numpy as np
import pandas as pd
import cudf
np.random.seed(12)

### Portions of this were borrowed from the
### cuDF cheatsheet, existing cuDF documentation,
### and 10 Minutes to Pandas.
### Created November, 2018.
```

```
[2]: s = cudf.Series([1,2,3,None,4])
print(s)
```

```
0    1
1    2
2    3
3    4
```

<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
```



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 ★



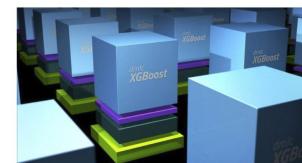
Financial data modeling with RAPIDS.

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



Jiwei Liu

Jul 3 · 5 min read

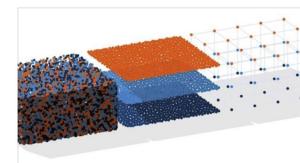


NVIDIA GPUs and Apache Spark, One Step Closer

RAPIDS XGBoost4j-Spark Package Now Available

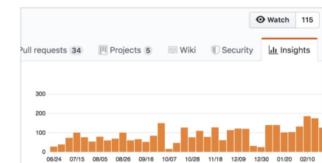


Karthikeyan Rajendran



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://github.com/rapidsai>

<https://medium.com/rapids-ai>

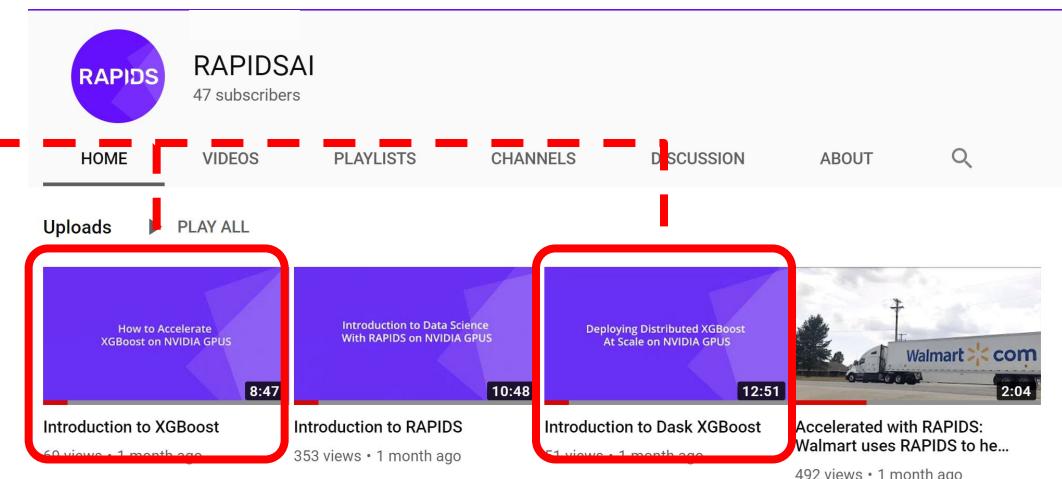
Explore: Notebooks Contrib

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.

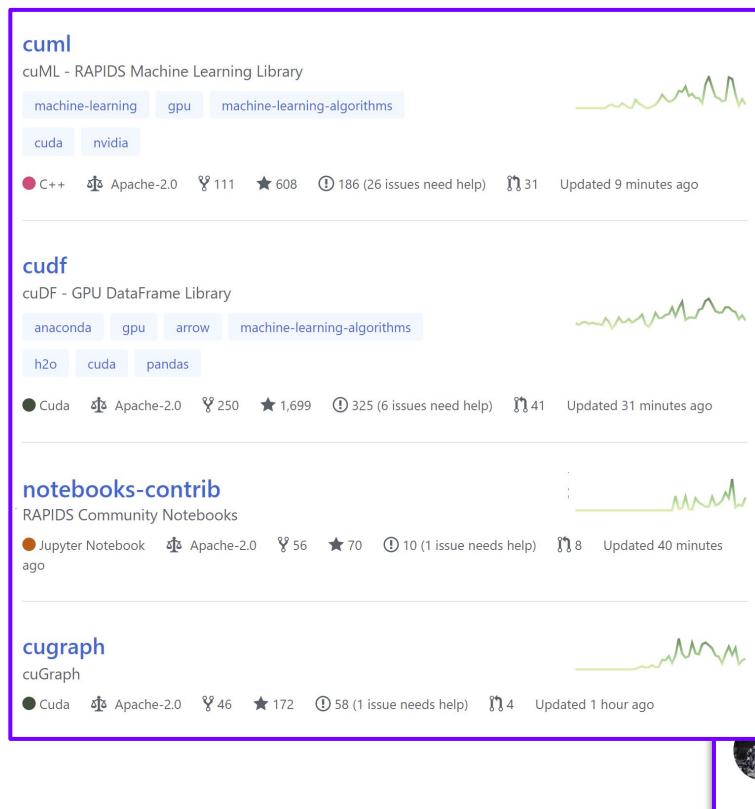
Intermediate Notebooks:

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.



Contribute Back

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

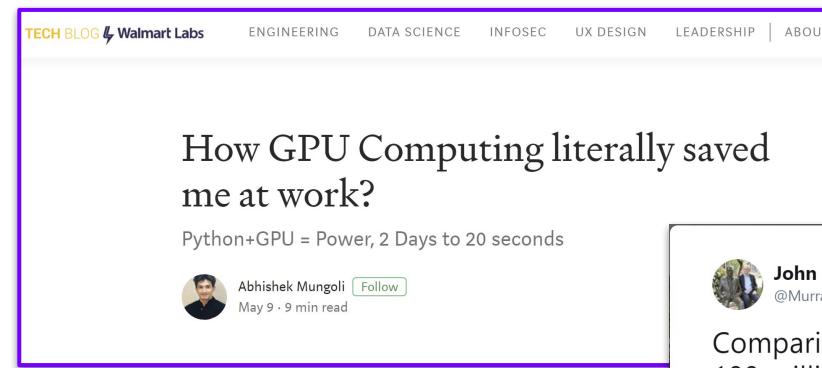


cuml
cuML - RAPIDS Machine Learning Library
machine-learning gpu machine-learning-algorithms
cuda nvidia
C++ Apache-2.0 111 608 186 (26 issues need help) 31 Updated 9 minutes ago

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

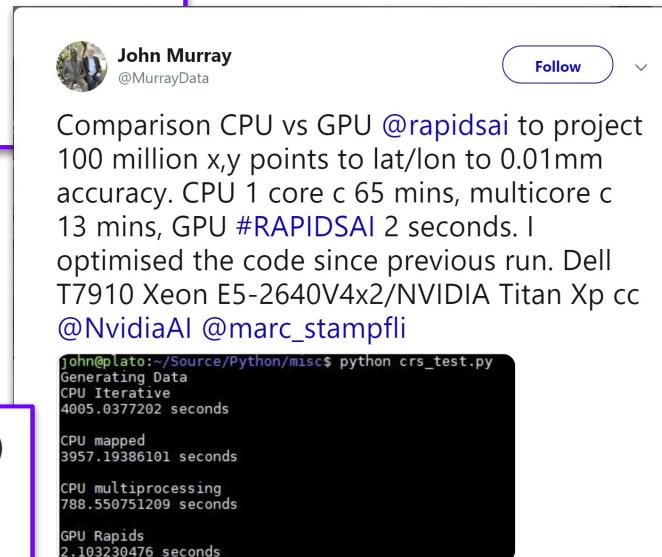


TECH BLOG Walmart Labs ENGINEERING DATA SCIENCE INFOSEC UX DESIGN LEADERSHIP ABOUT

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 May 9 · 9 min read

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 #RAPIDS 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
```

Getting Started with cuDF (RAPIDS)

Darren Ramsook Follow Jun 9 · 3 min read

Getting Started

RAPIDS Docs

New, improved, and easier to use

The screenshot shows a web browser window displaying the RAPIDS Docs API Reference for cuDF DataFrame. The URL is <https://docs.rapids.ai/api/cudf/stable/>. The page has a dark-themed header with tabs for 'STABLE' (selected), 'NIGHTLY', and 'LEGACY'. The left sidebar contains a navigation tree for 'cuDF' version 0.6, with sections for CONTENTS, API Reference (DataFrame, Series, Groupby, IO, GpuArrowReader), and developer documentation. The main content area shows the 'API Reference' for 'DataFrame'. It includes a class definition for `cudf.dataframe.DataFrame`, a description as a GPU Dataframe object, and examples for building dataframes using `__setitem__` and initializers. The code examples are highlighted in green.

API Reference

DataFrame

`class cudf.dataframe.DataFrame(name_series=None, index=None)`

A GPU Dataframe object.

Examples

Build dataframe with `__setitem__`:

```
>>> import cudf
>>> df = cudf.DataFrame()
>>> df['key'] = [0, 1, 2, 3, 4]
>>> df['val'] = [float(i + 10) for i in range(5)] # insert column
>>> print(df)
   key    val
0    0  10.0
1    1  11.0
2    2  12.0
3    3  13.0
4    4  14.0
```

Build dataframe with initializer:

```
>>> import cudf
>>> import numpy as np
>>> from datetime import datetime, timedelta
>>> ids = np.arange(5)
```

<https://docs.rapids.ai>

RAPIDS Docs

Easier than ever to get started with cuDF

The screenshot shows a web browser window titled "cuDF - Stable Docs - RAPIDS". The URL is <https://docs.rapids.ai/api/cudf/stable/>. The page content is the "10 Minutes to cuDF" guide. The sidebar on the left lists various topics under "10 Minutes to cuDF", including Object Creation, Viewing Data, Selection, Getting, Selection by Label, Selection by Position, Boolean Indexing, Setting, Missing Data, Operations, Stats, Applymap, Histogramming, String Methods, Merge, Concat, Join, Append, Grouping, Reshaping, Time Series, Categoricals, Plotting, and Converting Data Representation. The main content area starts with a header "10 Minutes to cuDF" and a note: "Modeled after 10 Minutes to Pandas, this is a short introduction to cuDF, geared mainly for new users." It includes a code block:

```
[1]: import os
import numpy as np
import pandas as pd
import cudf
np.random.seed(12)

#### Portions of this were borrowed from the
#### cuDF cheatsheet, existing cuDF documentation,
#### and 10 Minutes to Pandas.
#### Created November, 2018.
```

Below this, there is a section titled "Object Creation" with a note: "Creating a `Series`." It shows another code block:

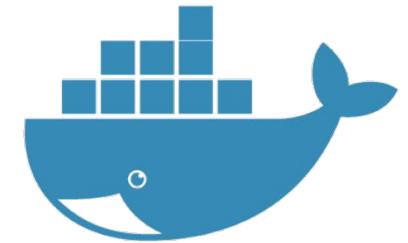
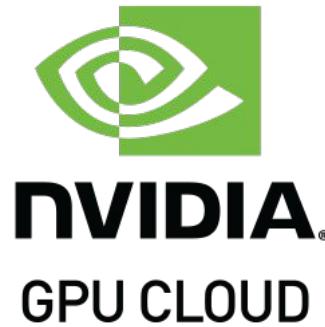
```
[2]: s = cudf.Series([1,2,3,None,4])
print(s)
```

0	1
1	2
2	3
3	
4	4

At the bottom, there is a note: "Creating a `DataFrame` by specifying values for each column."

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

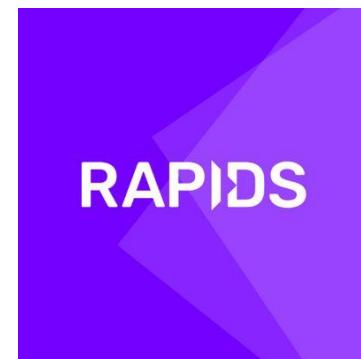
Everyone can help!



APACHE ARROW

<https://arrow.apache.org/>

@ApacheArrow



RAPIDS

<https://rapids.ai>

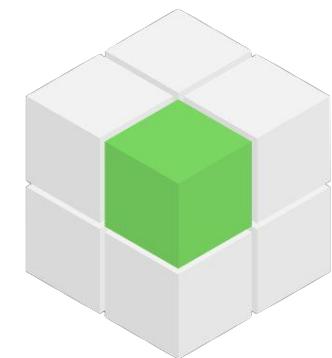
@RAPIDSAI



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

joshuap@nvidia.com

@datametrician



RAPIDS