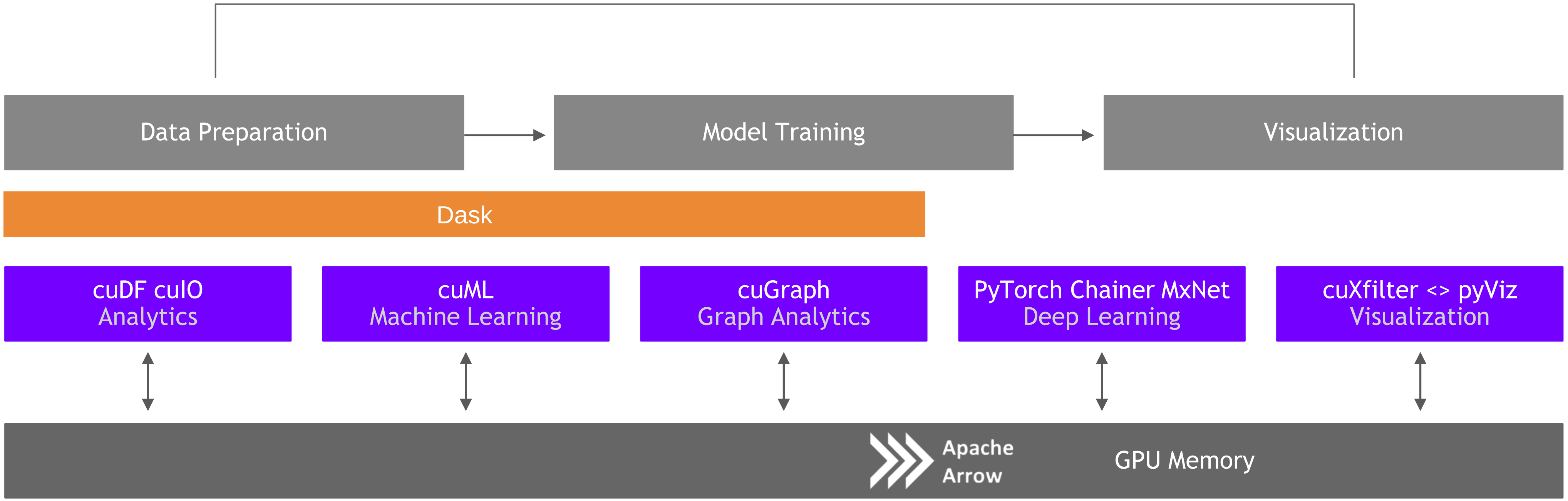
RAPIDS:OPEN SOURCE PYTHON DATA SCIENCE WITH GPU ACCELERATION AND DASK

Joe Eaton, Sept 24, 2019

Principal Sys Engineer for Graph and Data Analytics, NVIDIA

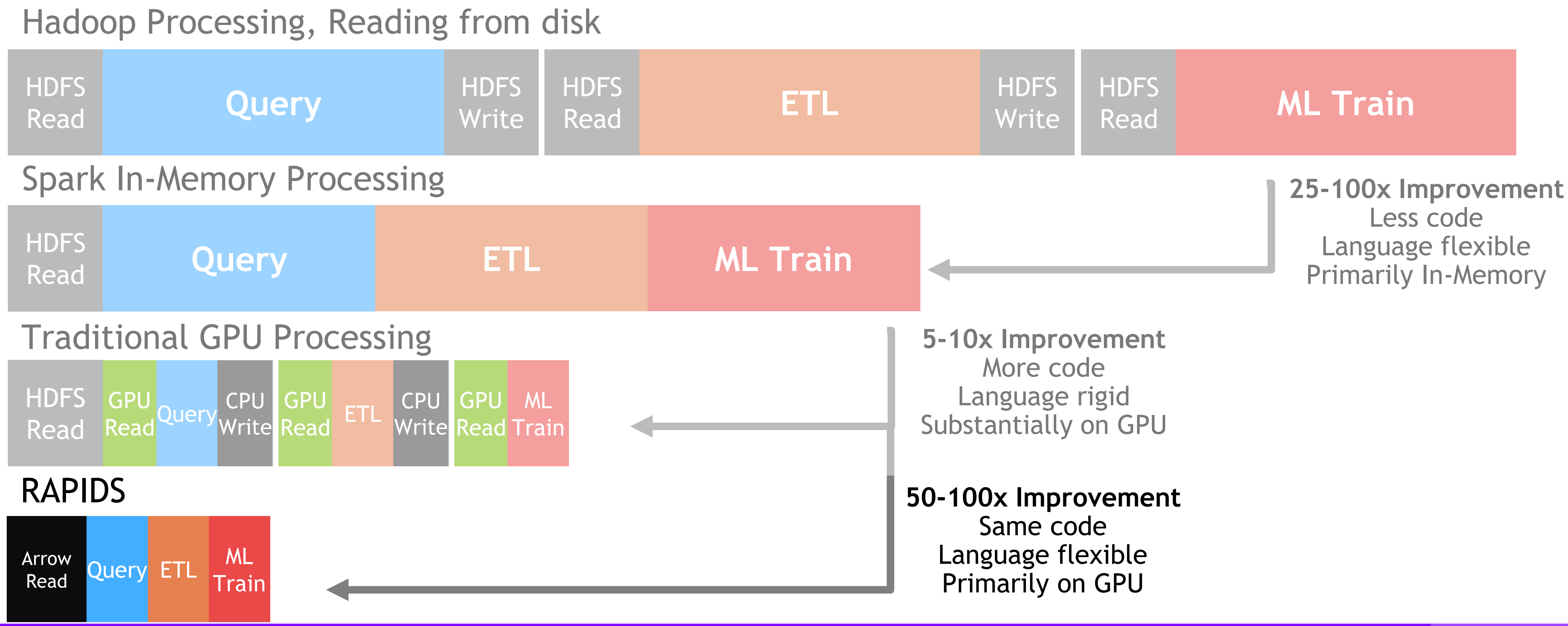
RAPIDS

End-to-End Accelerated GPU Data Science

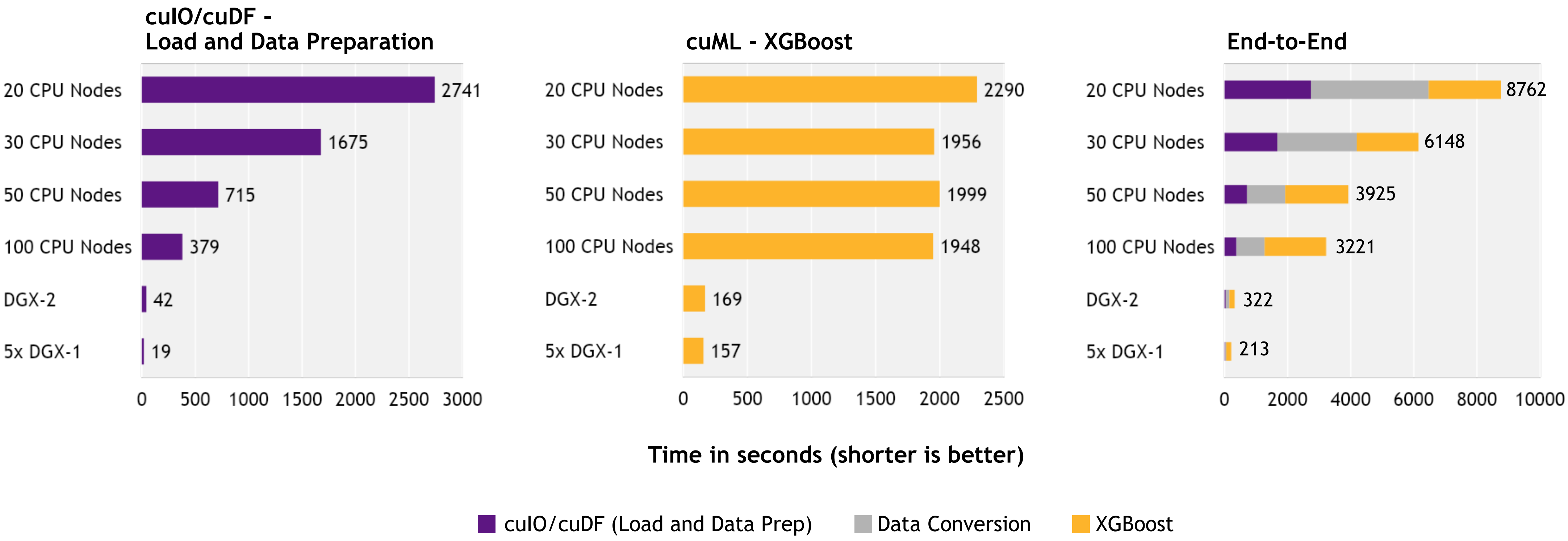


Data Processing Evolution

Faster data access, less data movement



Faster Speeds, Real-World Benefits



Benchmark

200GB CSV dataset; Data prep includes joins, variable transformations

CPU Cluster Configuration

CPU nodes (61 GiB memory, 8 vCPUs, 64-bit platform), Apache Spark v2.3, XGBoost 0.9

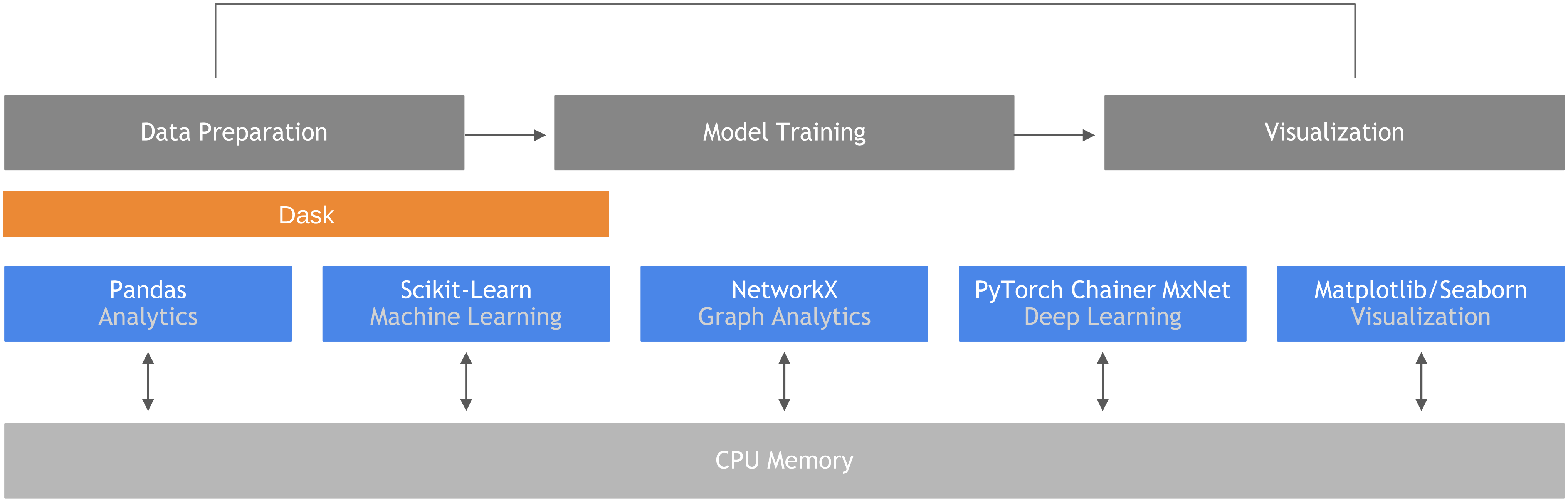
DGX Cluster Configuration

5x DGX-1 on InfiniBand network,Ubuntu 16.04, CUDA 10, Driver 410.48, NCCL 2.4.7

RAPIDS Core

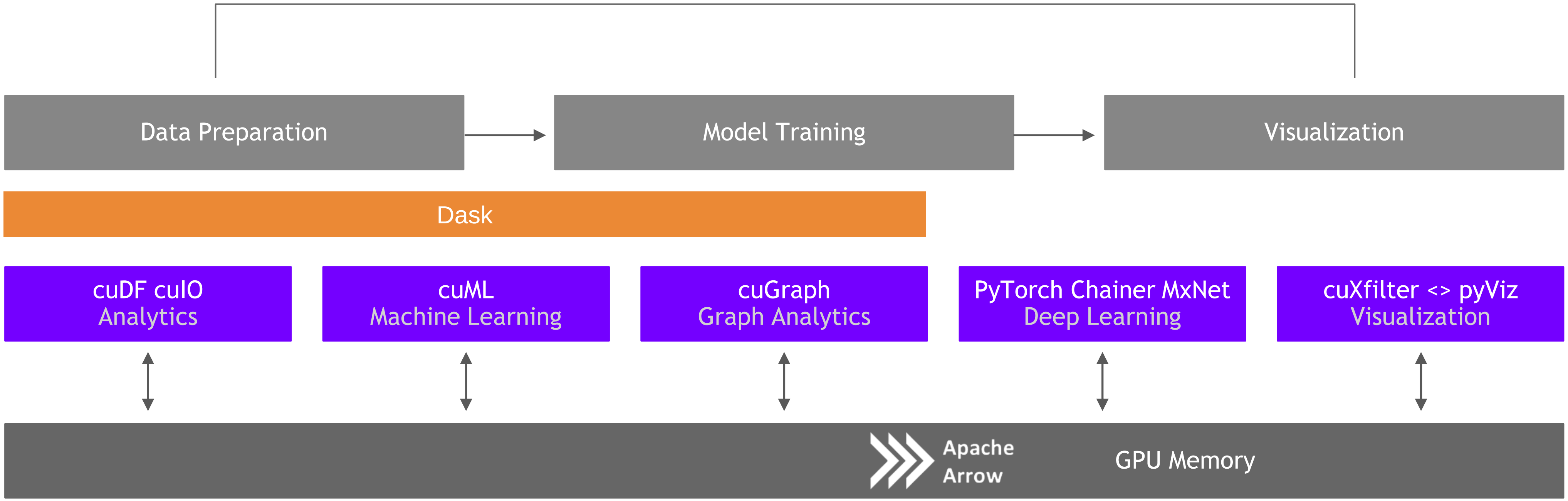
Open Source Data Science Ecosystem

Familiar Python APIs



RAPIDS

End-to-End Accelerated GPU Data Science



Dask

RAPIDS

Scaling RAPIDS with Dask

Data Preparation → Model Training → Visualization

Dask

cuDF cuIO Analytics	cuML Machine Learning	cuGraph Graph Analytics	PyTorch Chainer MxNet Deep Learning	cuXfilter <> pyViz Visualization

Apache Arrow GPU Memory

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

DASK

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

K8s Native API

Quickstart

```python
from dask_kubernetes import KubeCluster

cluster = KubeCluster.from_yaml('worker-spec.yml')
cluster.scale_up(10)  # specify number of nodes explicitly

cluster.adapt(minimum=1, maximum=100)  # or dynamically scale based on current workload
```

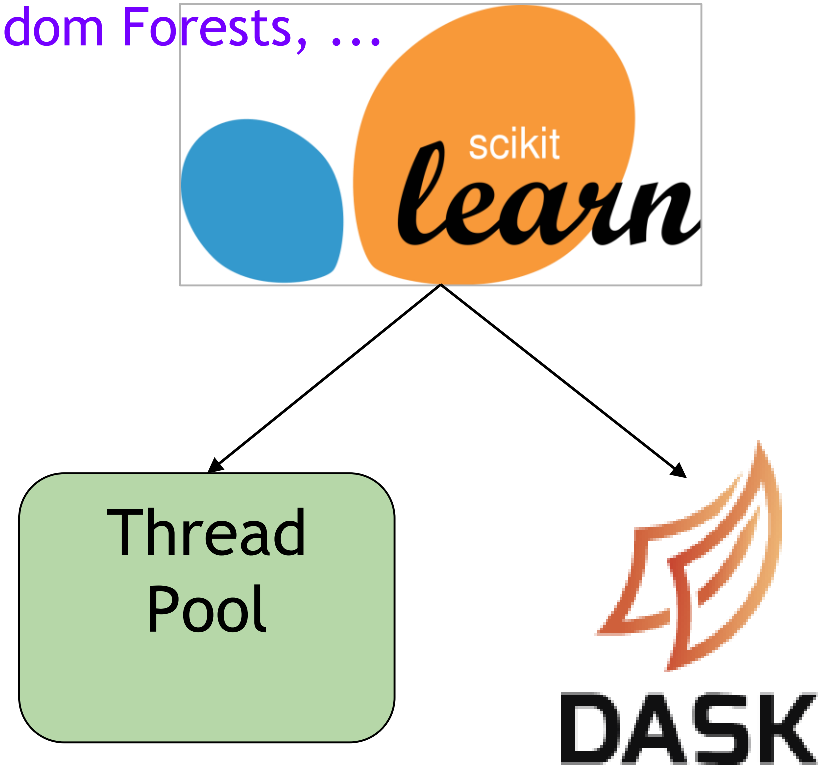
Parallel Scikit-Learn

For Hyper-Parameter Optimization, Random Forests, ...

- Same API

```
from scikit_learn.externals import joblib
with joblib.parallel_backend('dask'):
    estimator = RandomForest()
    estimator.fit(data, labels)
```

- Same exact code, just wrap with a decorator
- Replaces default threaded execution with Dask
  Allowing scaling onto clusters
- Available in most Scikit-Learn algorithms where joblib is used



Parallel Python

For custom systems, ML algorithms, workflow engines

- Parallelize existing codebases

```
results = {}

for x in X:
  for y in Y:
    if x < y:
      result = f(x, y)
    else:
      result = g(x, y)
    results.append(result)
```

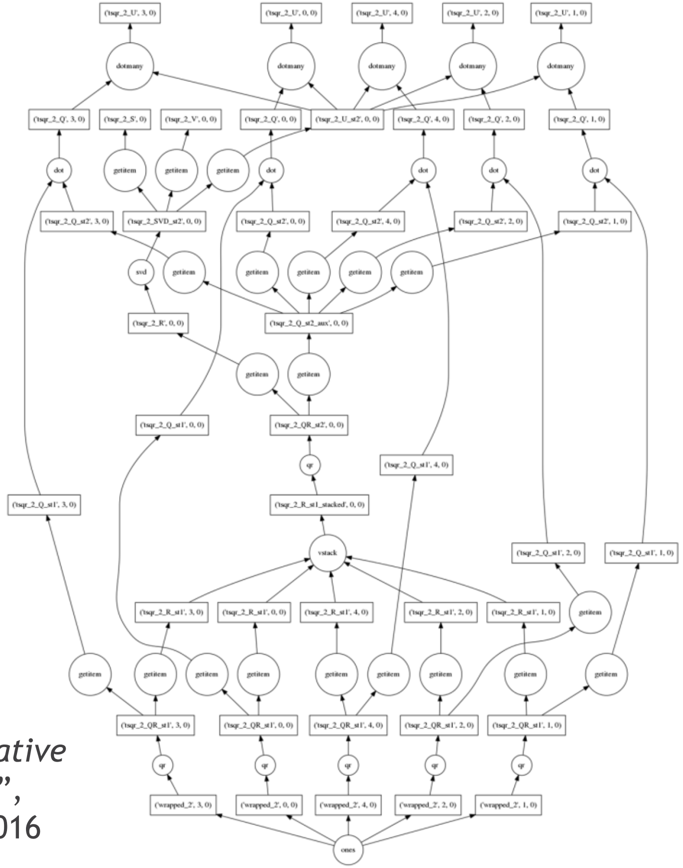
Parallel Python

For custom systems, ML algorithms, workflow engines

- Parallelize existing codebases

```
f = dask.delayed(f)
g = dask.delayed(g)

results = {}

for x in X:
  for y in Y:
    if x < y:
      result = f(x, y)
    else:
      result = g(x, y)
    results.append(result)

result = dask.compute(results)
```

M Tepper, G Sapiro “*Compressed nonnegative matrix factorization is fast and accurate*”, IEEE Transactions on Signal Processing, 2016



Dask Connects Python users to Hardware

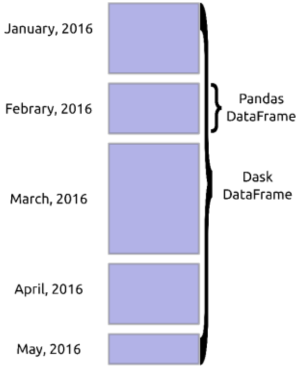
High Productivity Even on Large Scale Problems

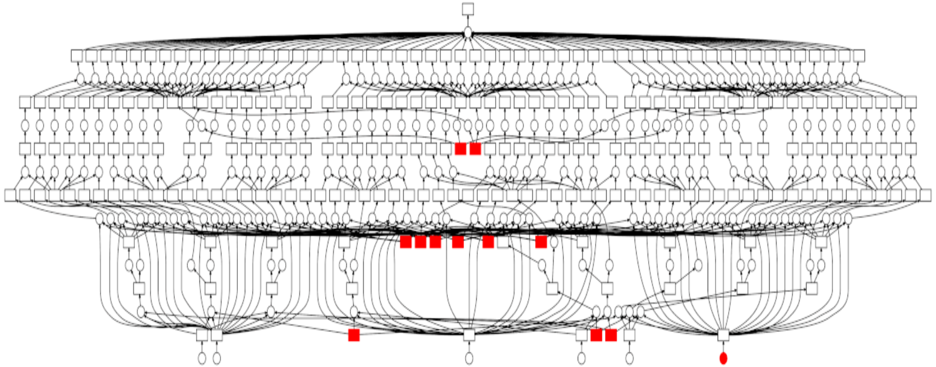
User

Execute on distributed hardware

Dask Connects Python users to Hardware

High Productivity Even on Large Scale Problems





User

Writes high level code (NumPy/Pandas/Scikit-Learn)

Turns into a task graph

Executes on distributed hardware

Why OpenUCX?

Bringing hardware accelerated communications to Dask

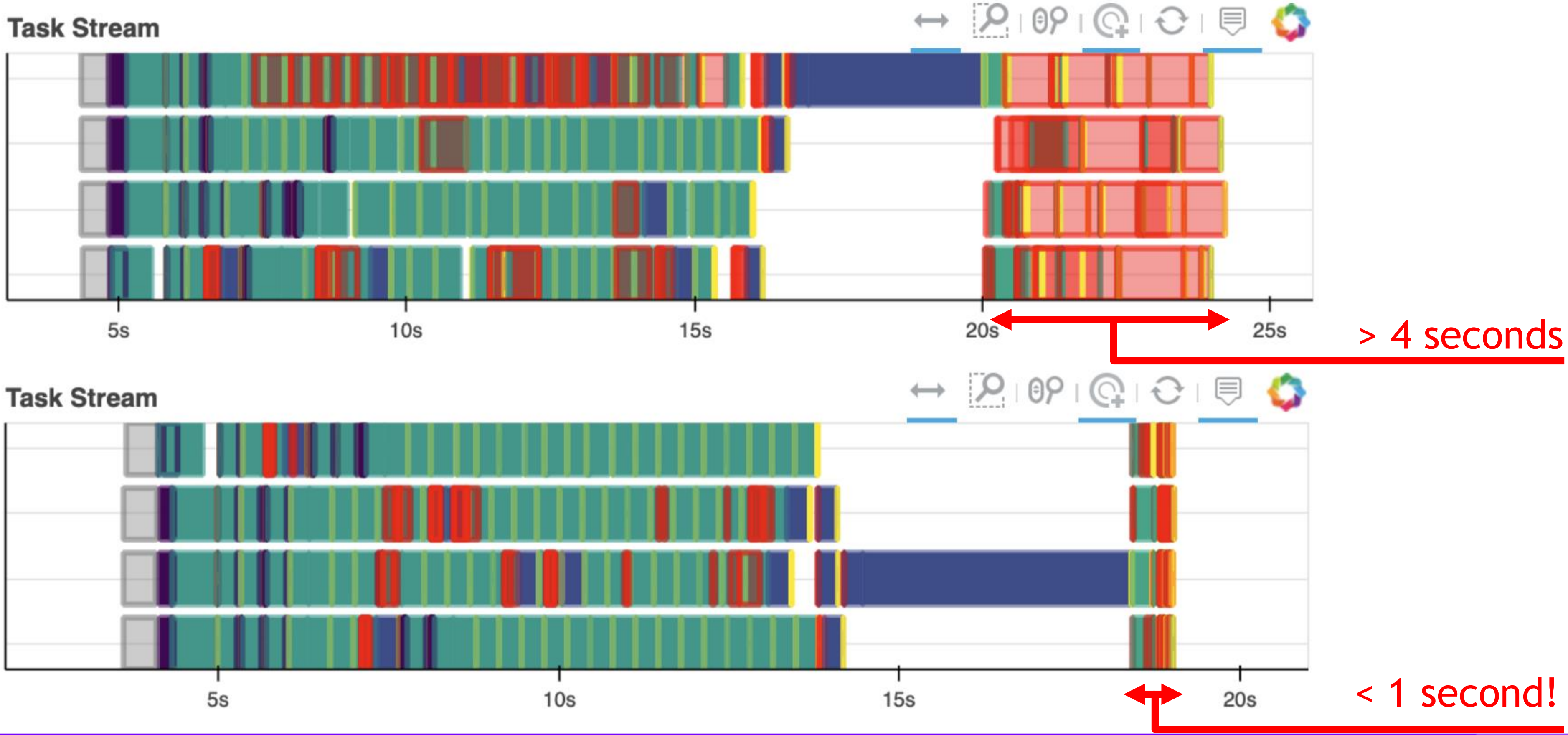
- TCP sockets are slow!

- 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

Challenges: Communication

OpenUCX Performance – Before and After



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

Data Preparation → Model Training → Visualization

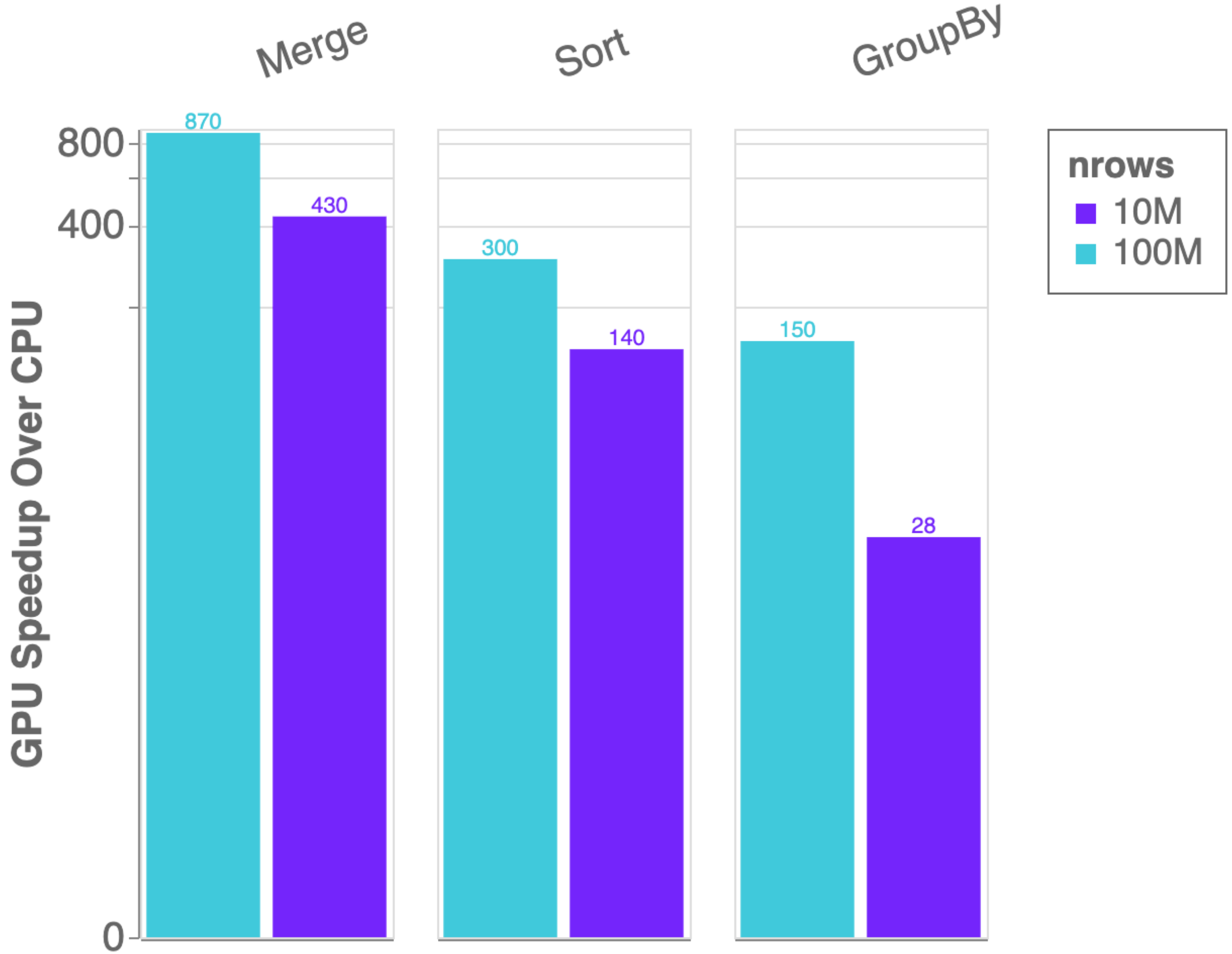
Dask

cuDF cuIO	cuML	cuGraph	PyTorch Chainer MxNet	cuXfilter <> pyViz
Analytics	Machine Learning	Graph Analytics	Deep Learning	Visualization

Apache Arrow

GPU Memory

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:

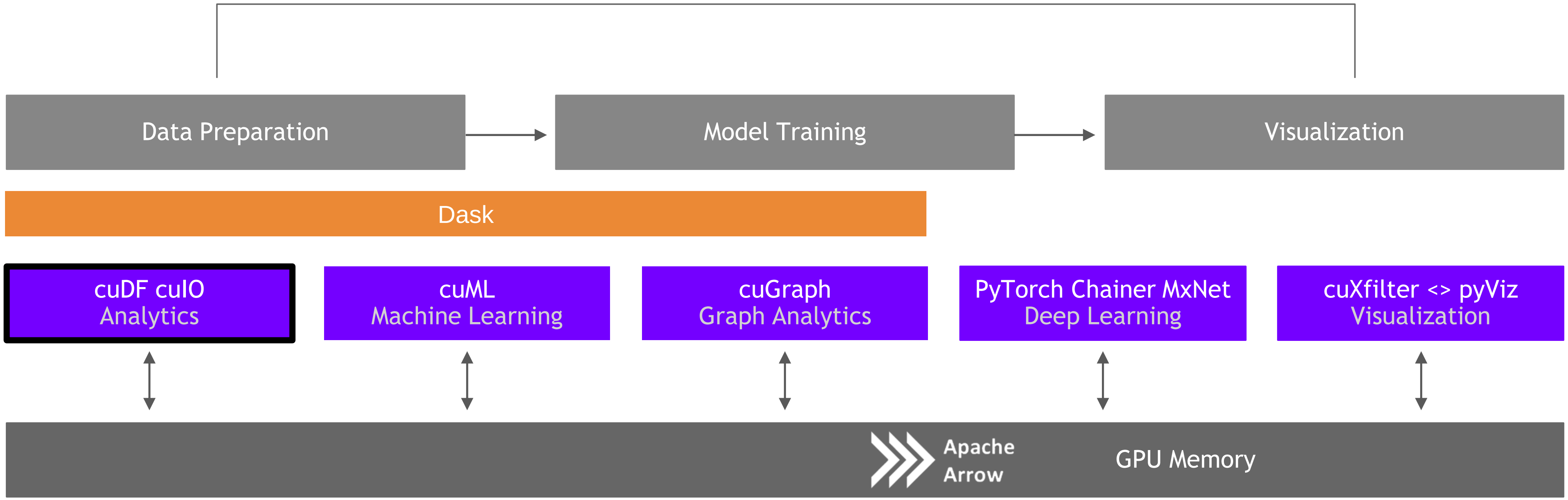
DataFrames: 2x int32 columns key columns, 3x int32 value columns

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

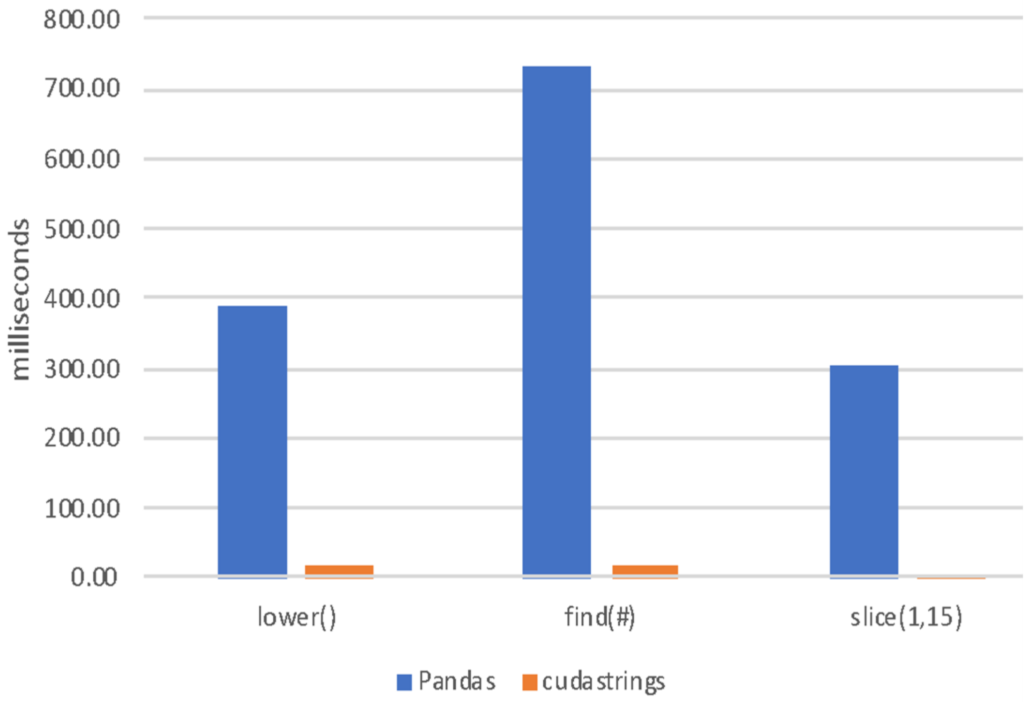
String Support

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

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 v0.10
- 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

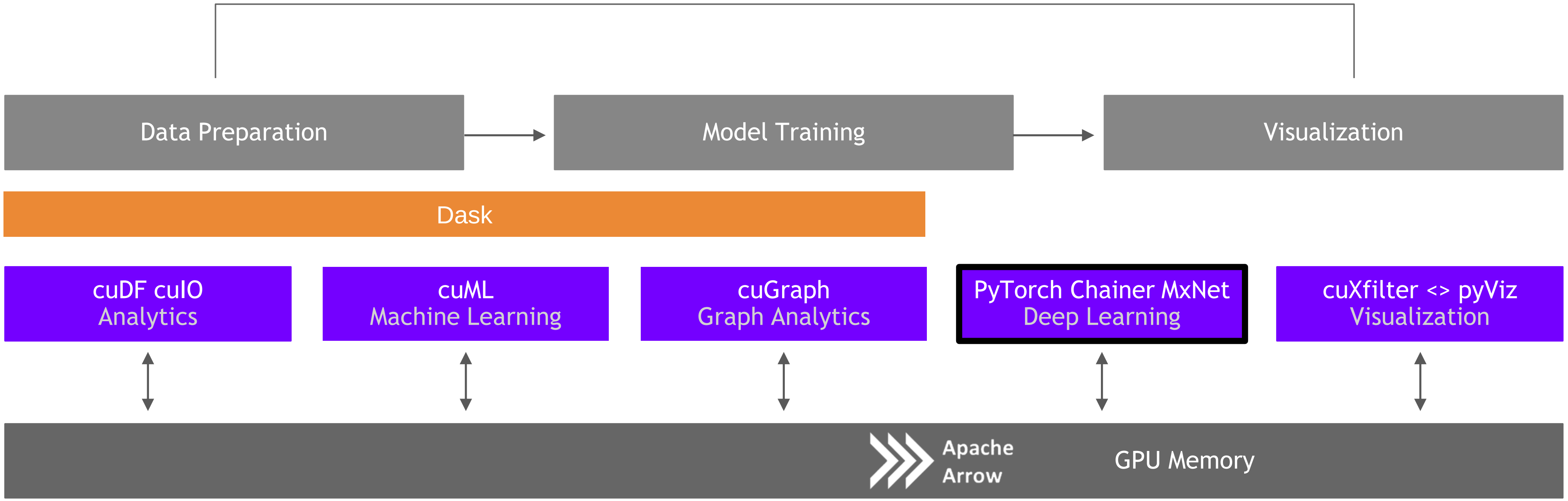
```
1]: import pandas, cudf

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!

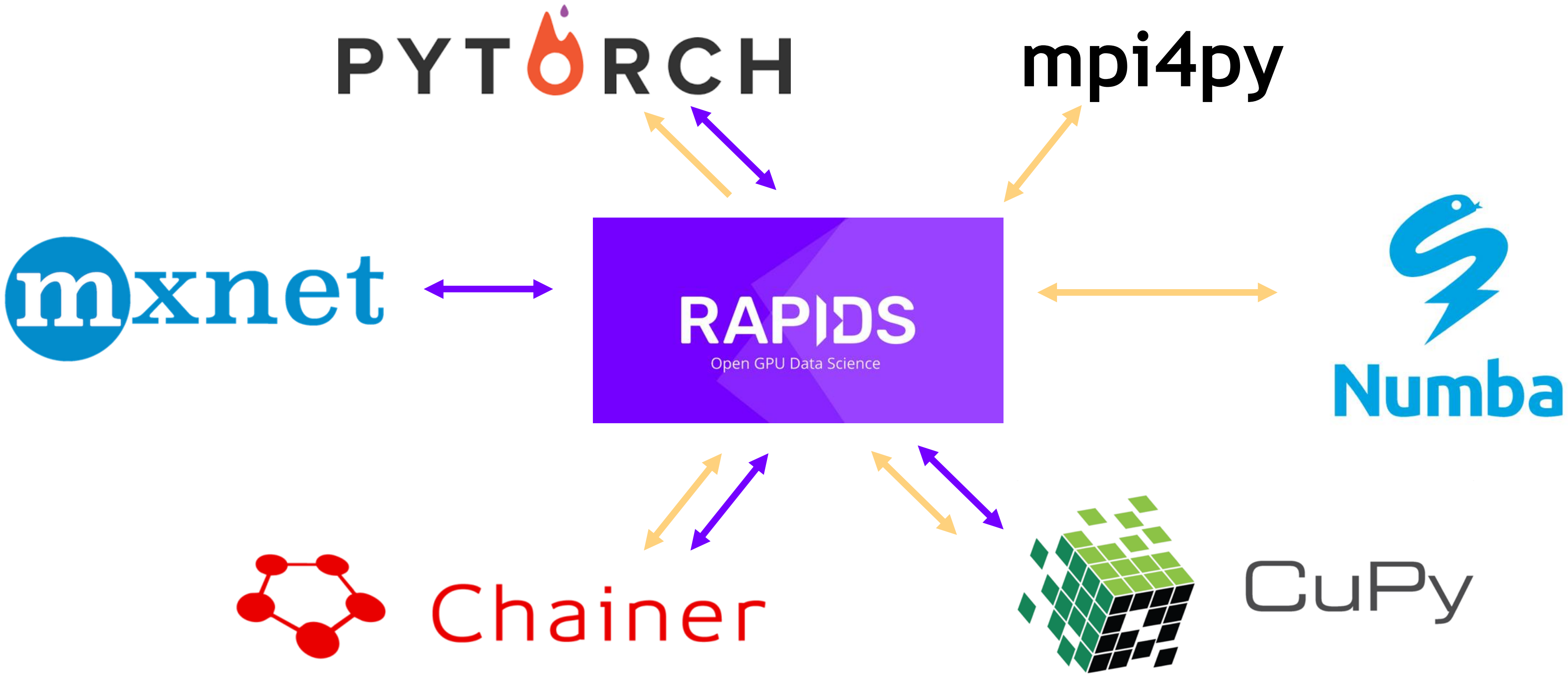
RAPIDS

Building bridges into the array ecosystem



Interoperability With Common Frameworks

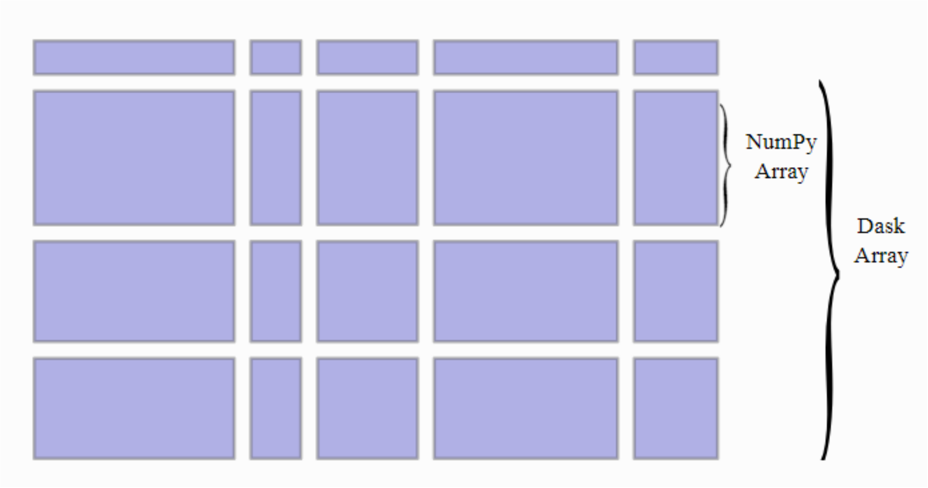
DLPack and __cuda_array_interface__



ETL – Arrays and DataFrames

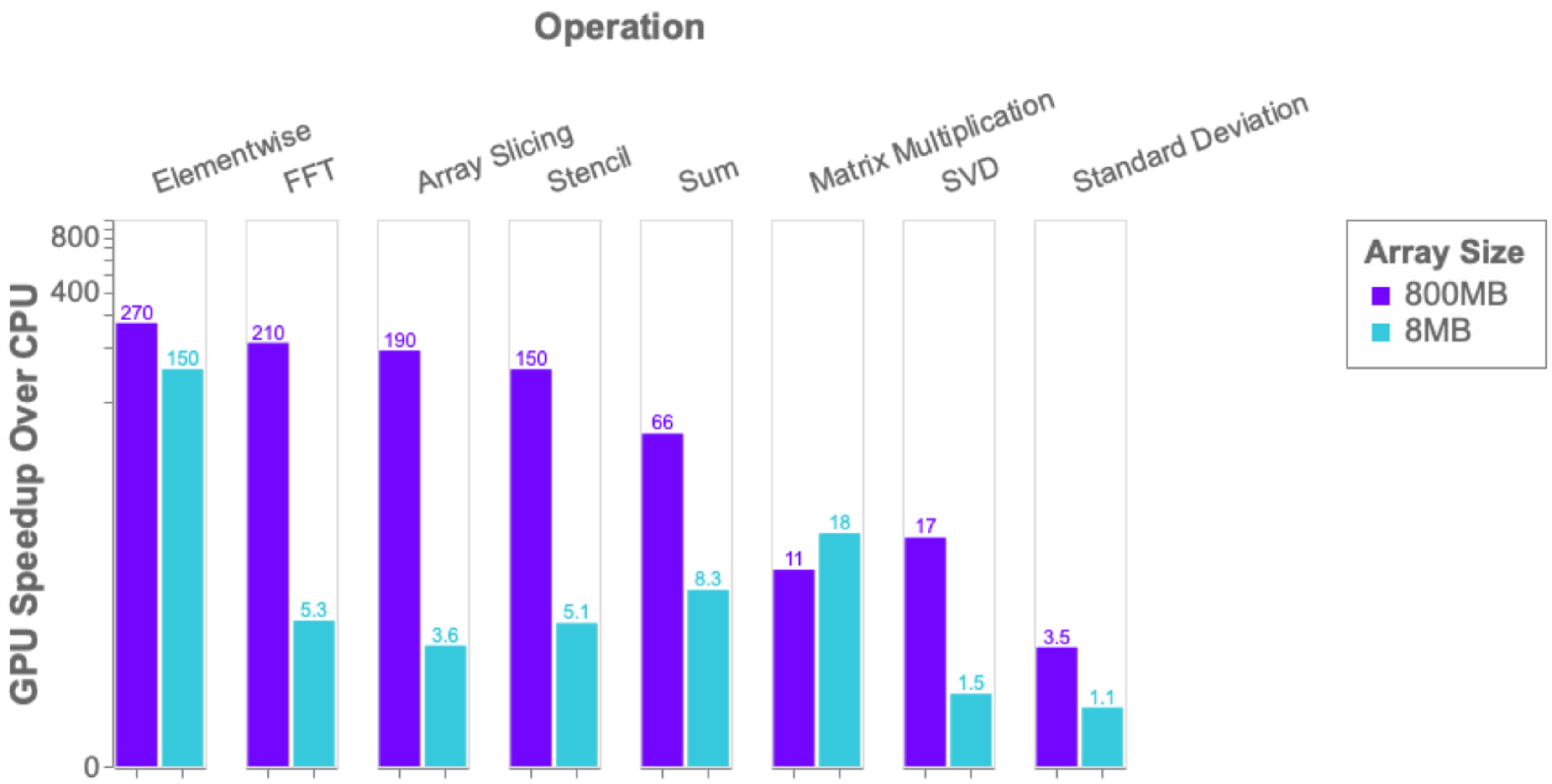
Dask and CUDA Python arrays

Chainer CuPy Numba



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

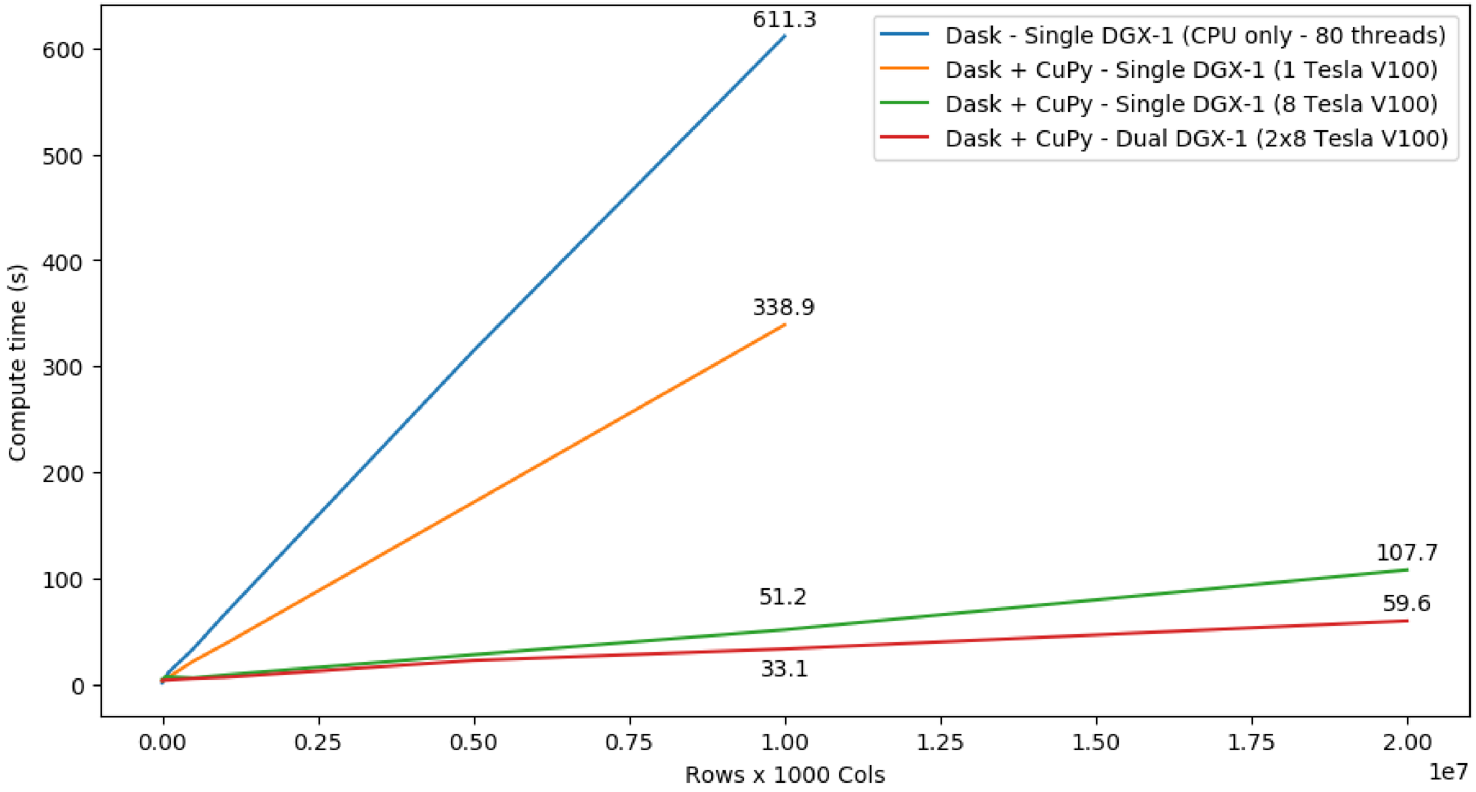
Benchmark: single-GPU CuPy vs NumPy



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

SVD Benchmark

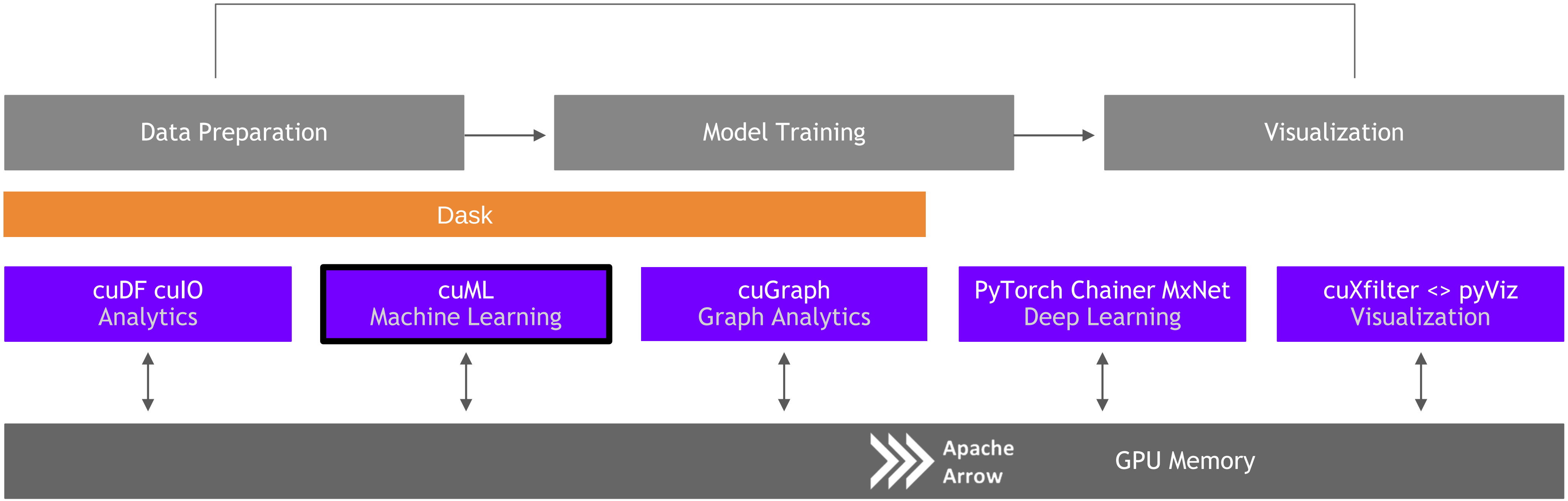
Dask and CuPy Doing Complex Workflows



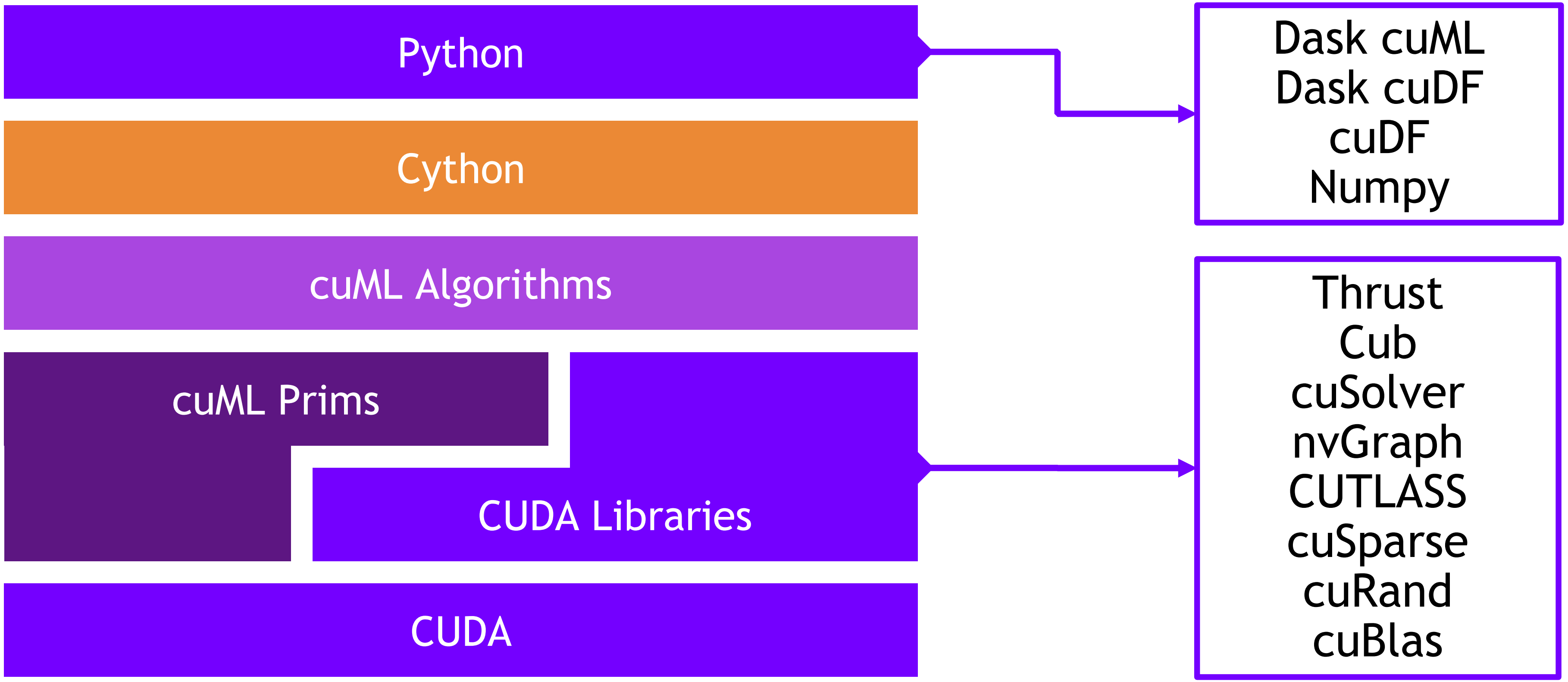
cuML

Machine Learning

More models more problems

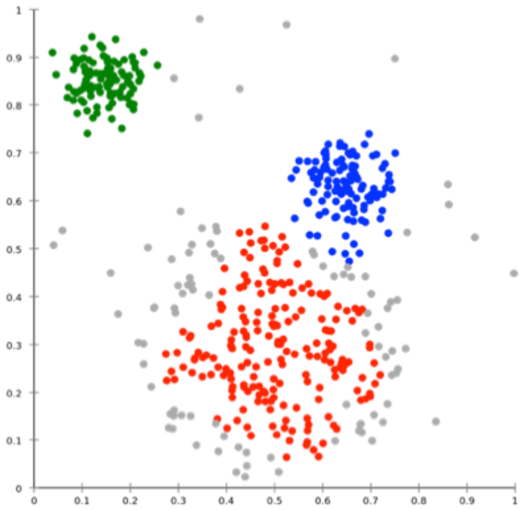


ML Technology Stack



Algorithms

GPU-accelerated Scikit-Learn



Classification / Regression	Decision Trees / Random Forests Linear Regression Logistic Regression K-Nearest Neighbors
Inference	**Random forest / GBDT inference**
Clustering	K-Means DBSCAN Spectral Clustering
Decomposition & Dimensionality Reduction	Principal Components Singular Value Decomposition UMAP Spectral Embedding
Time Series	**Holt-Winters** Kalman Filtering

Cross Validation

Hyper-parameter Tuning

More to come!

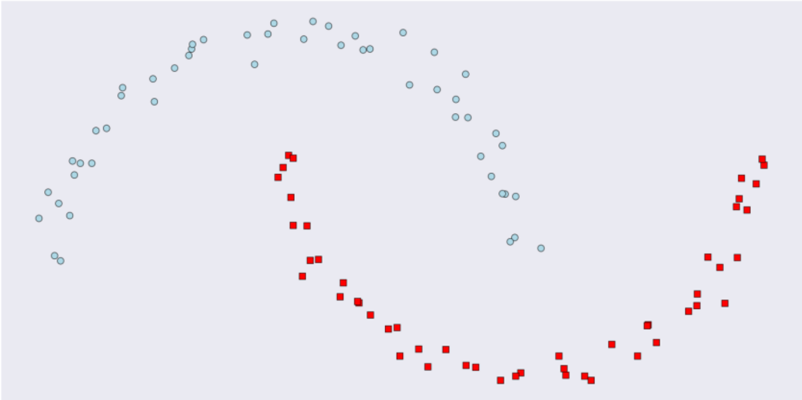
Key:
- Preexisting
- **NEW for 0.9**

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)
```

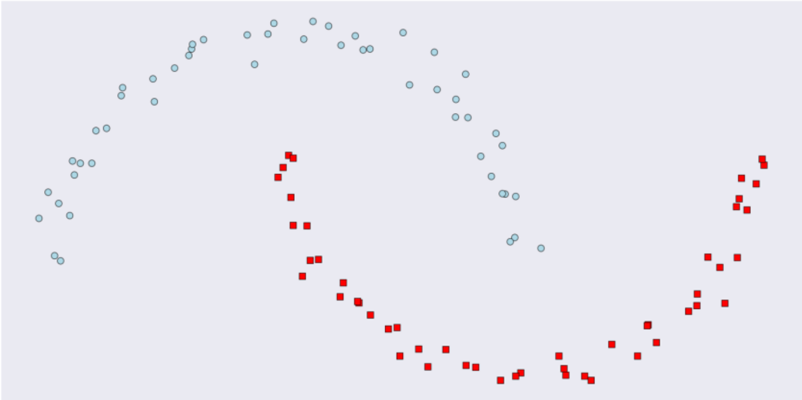


RAPIDS matches common Python APIs

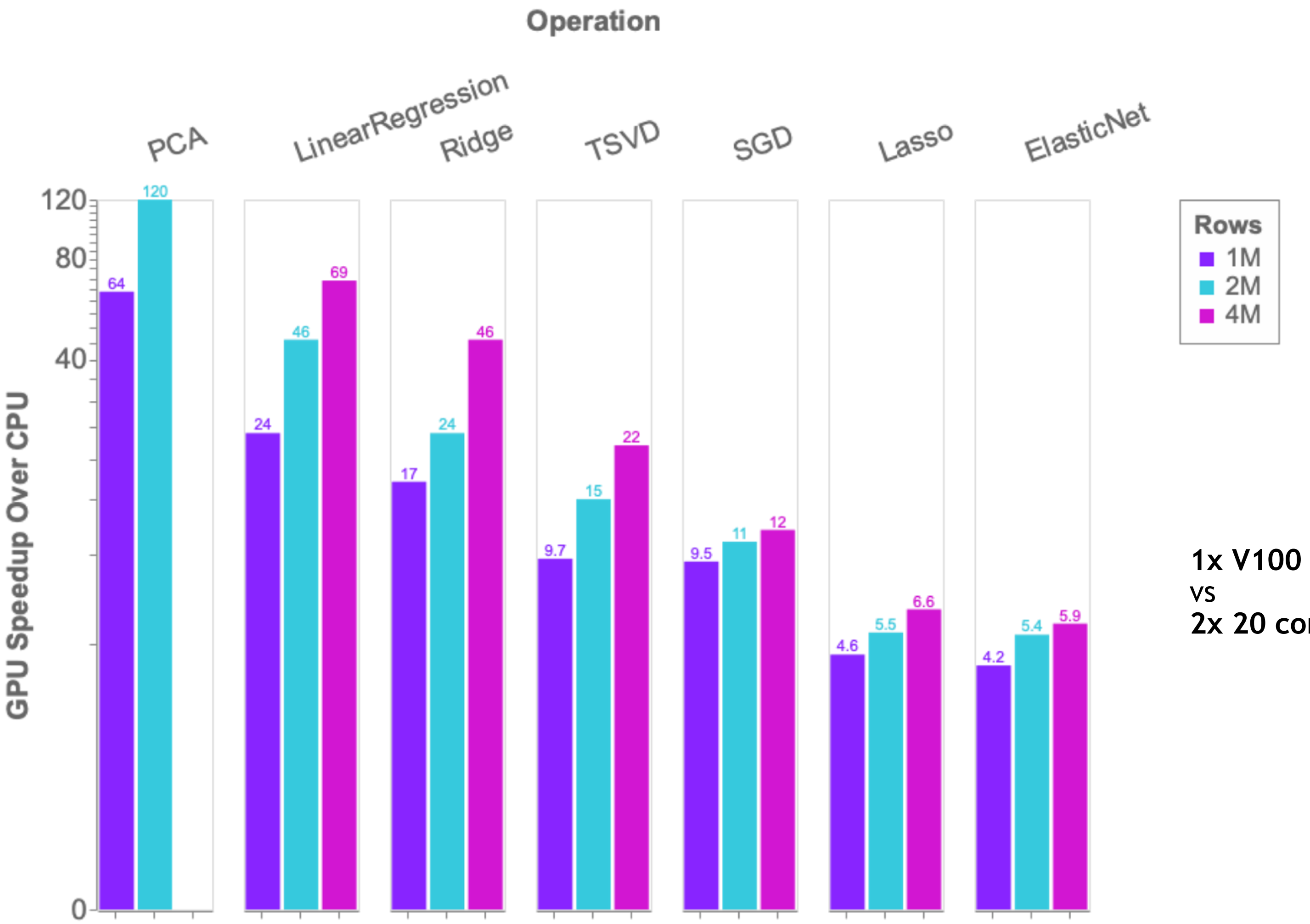
GPU-Accelerated Clustering

```
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])})
```

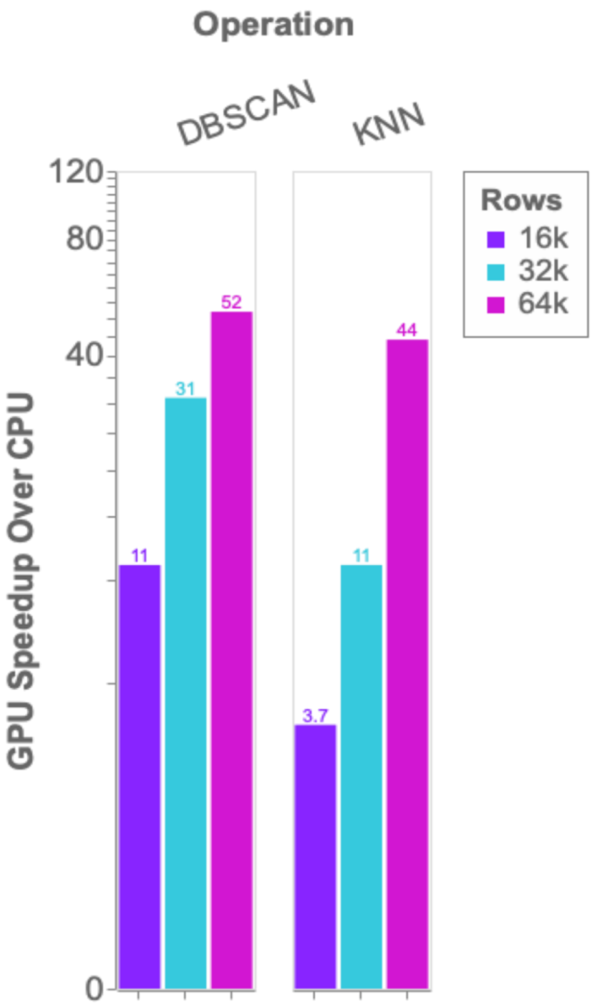
```
from cuml               import DBSCAN
dbscan = DBSCAN(eps = 0.3, min_samples = 5)

dbscan.fit(X)

y_hat = dbscan.predict(X)
```



Benchmarks: single-GPU cuML vs scikit-learn



1x V100
vs
2x 20 core CPU



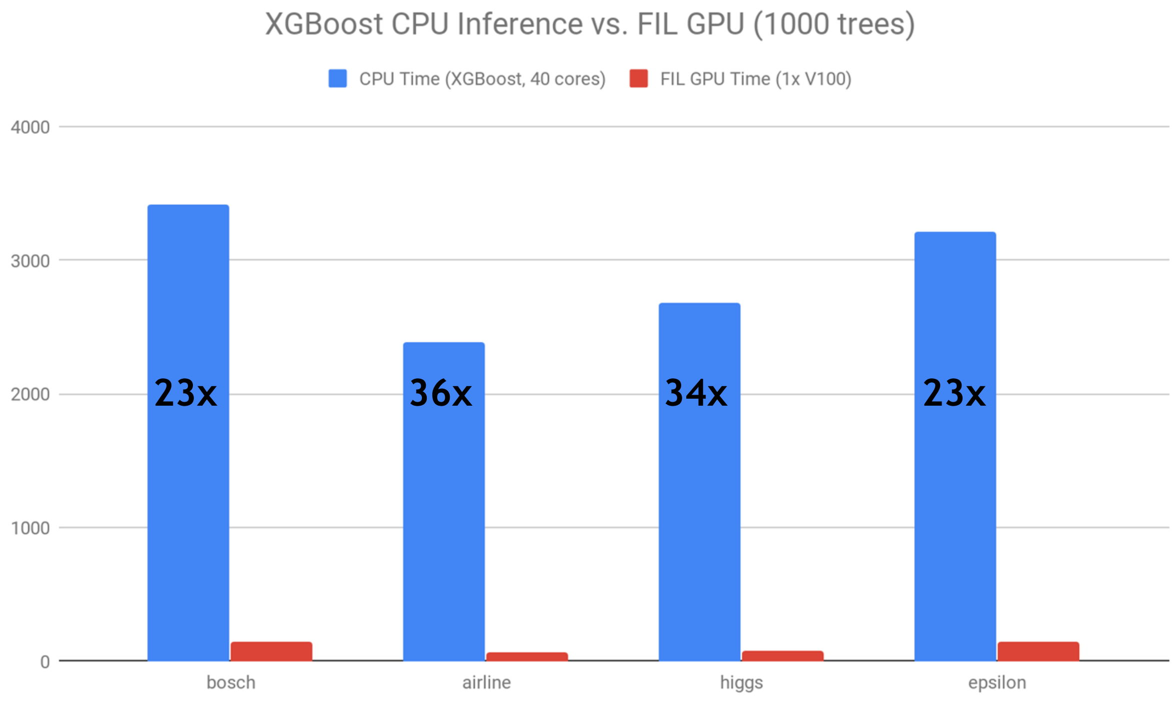
Forest Inference at 100M inferences/sec

Taking models from training to production

cuML's **Forest Inference Library**

Works with existing models
from XGBoost and LightGBM today

- 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

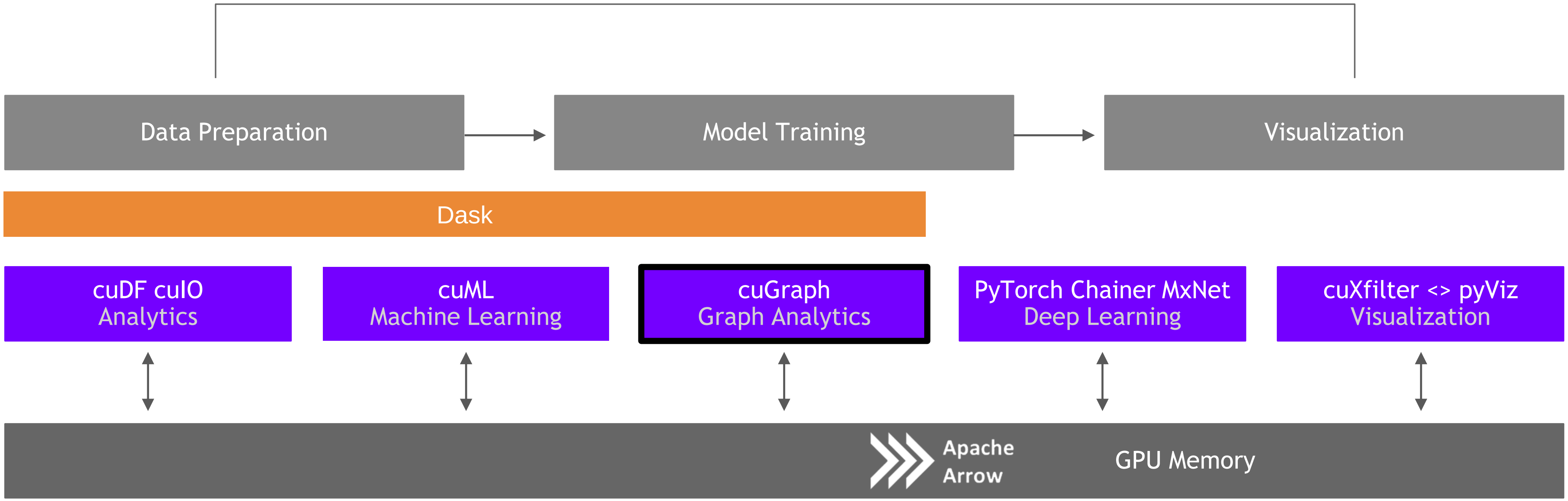
March 2020 - RAPIDS 0.14

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

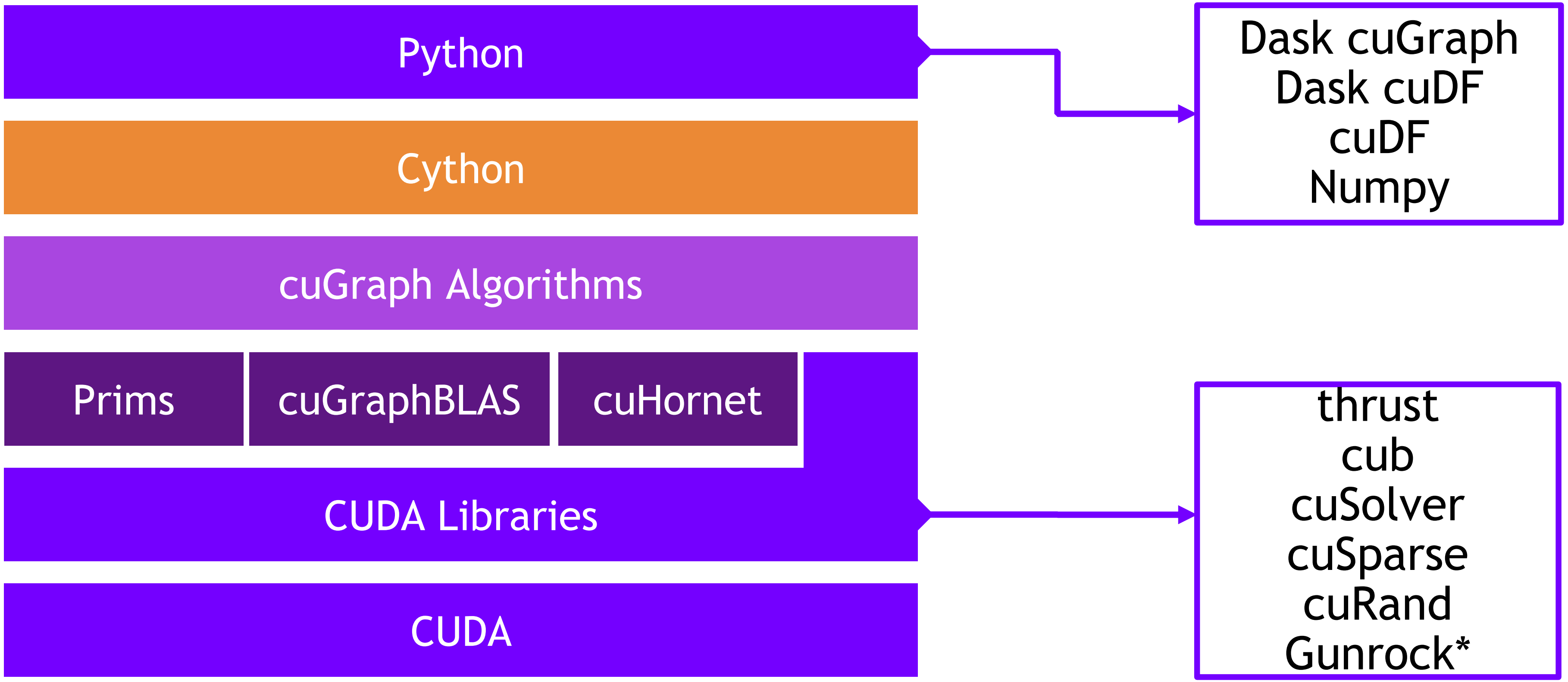
Seamless Integration with cuDF and cuML

- Property Graph support via DataFrames

Growing Functionality

- Extensive collection of algorithm, primitive, and utility functions

Graph Technology Stack

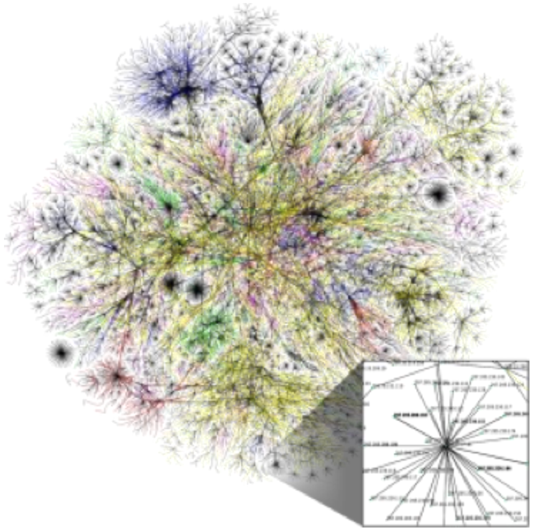


nvGRAPH has been Opened Sourced and integrated into cuGraph. A legacy version is available in a RAPIDS GitHub repo

* Gunrock is from UC Davis

Algorithms

GPU-accelerated NetworkX



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

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)

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

Query Language

Multi-GPU

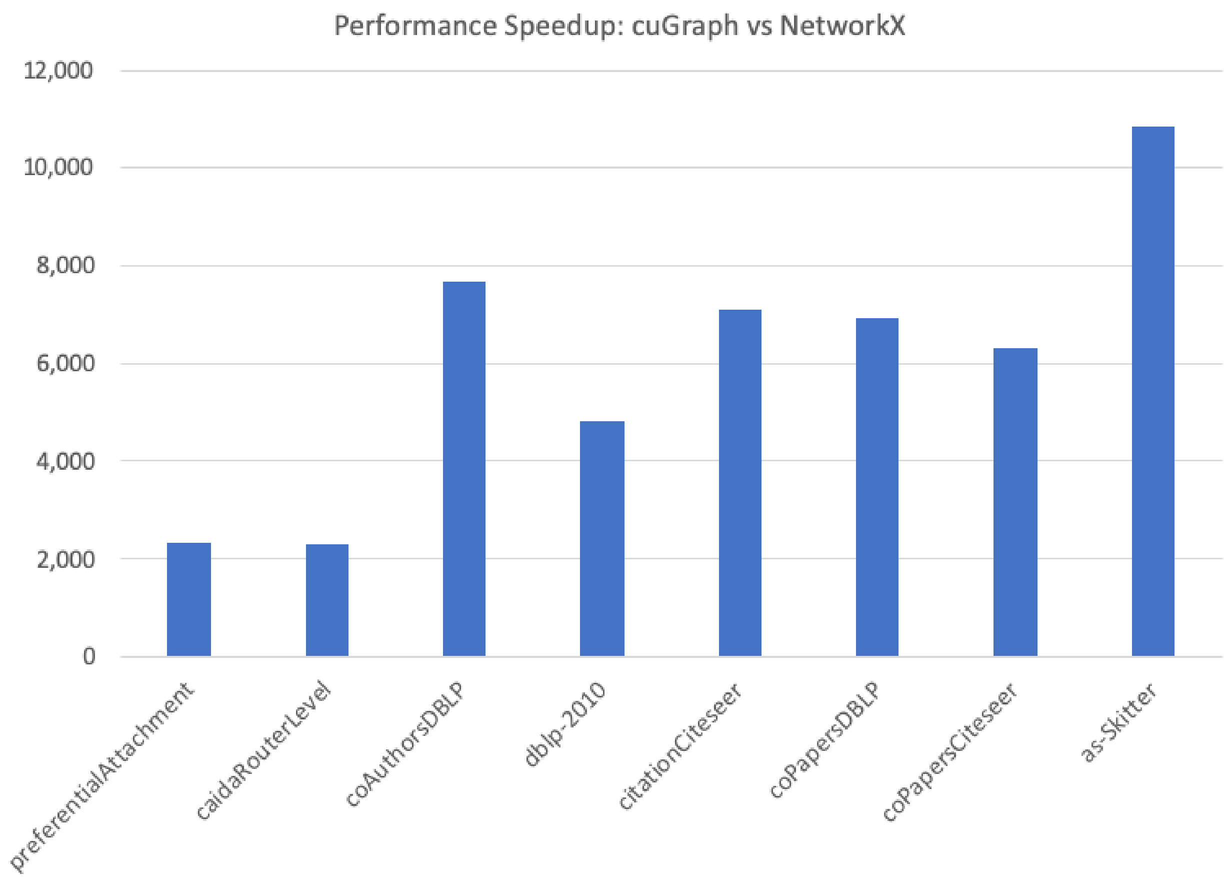
Utilities

More to come!

Louvain Single Run

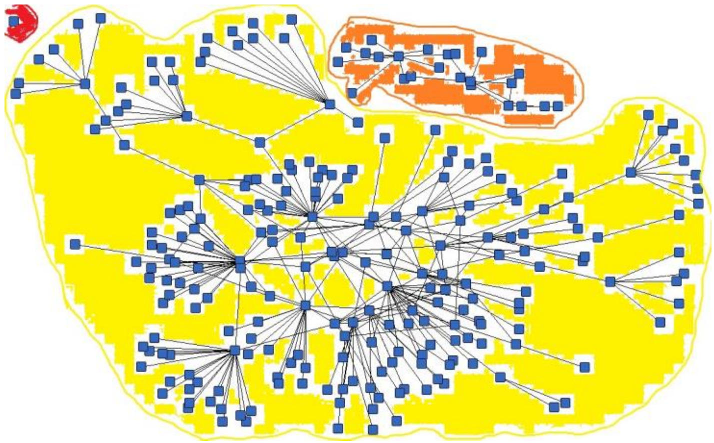
```
G = cugraph.Graph()
G.add_edge_list(
  gdf["src_0"],gdf["dst_0"],
  gdf["data"])

df, mod = cugraph.nvLouvain(G)
```

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

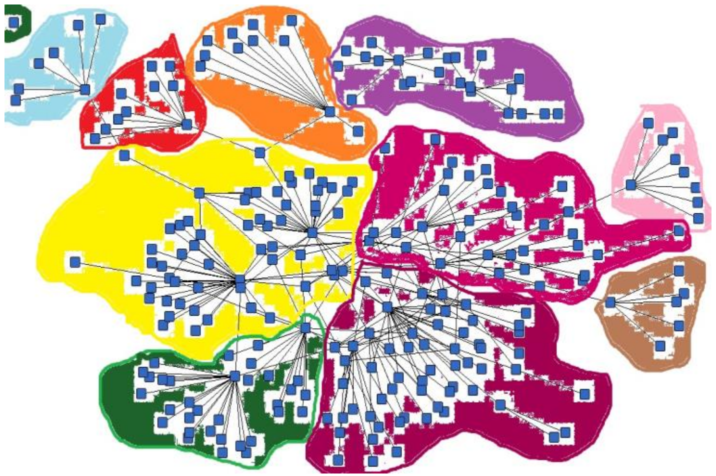


Hierarchical Louvain clusters

See More of the Whole Picture



Check the size of each cluster
If size> threshold : recluster



Dominant Community

Sub-Communities

Dict = {'0' : initial clusters ,
'1' : reclustering on data from '0' ,
'2' : reclustering on data from '1' …… }

Multi-GPU PageRank Performance

PageRank portion of the HiBench benchmark suite, DGX-2 Hardware

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			

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/

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