HIGH-PERFORMANCE DATA SCIENCE WITH RAPIDS

Zahra Ronaghi
AI Infrastructure Manager

END-TO-END ACCELERATED GPU DATA SCIENCE

Data Processing Evolution

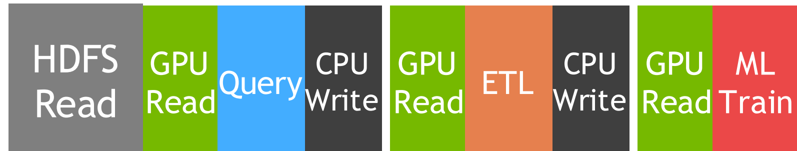
Faster data access, less data movement

Hadoop Processing, Reading from disk

Spark In-Memory Processing

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

Traditional GPU Processing

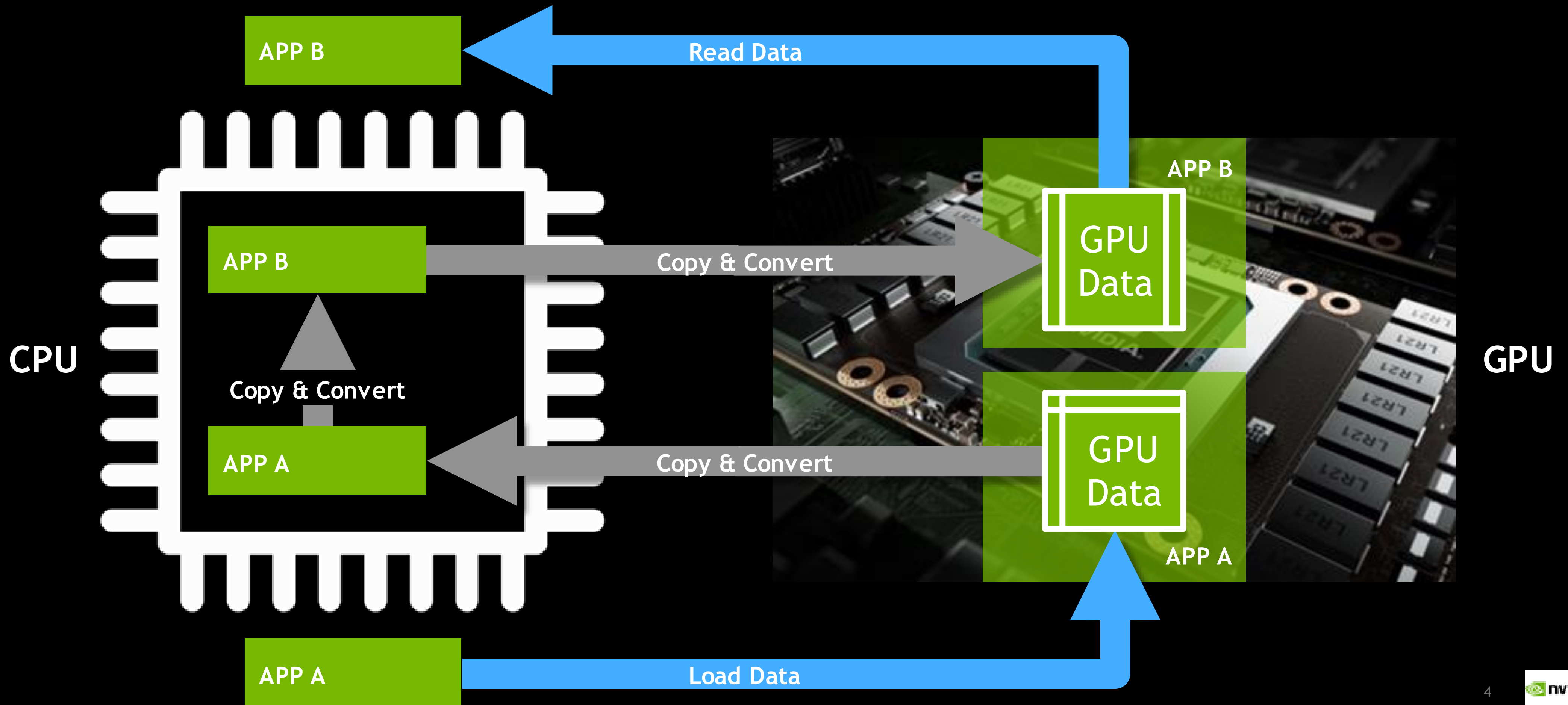


5-10x Improvement
More code
Language rigid
Substantially on GPU

NVIDIA

Data Movement and Transformation

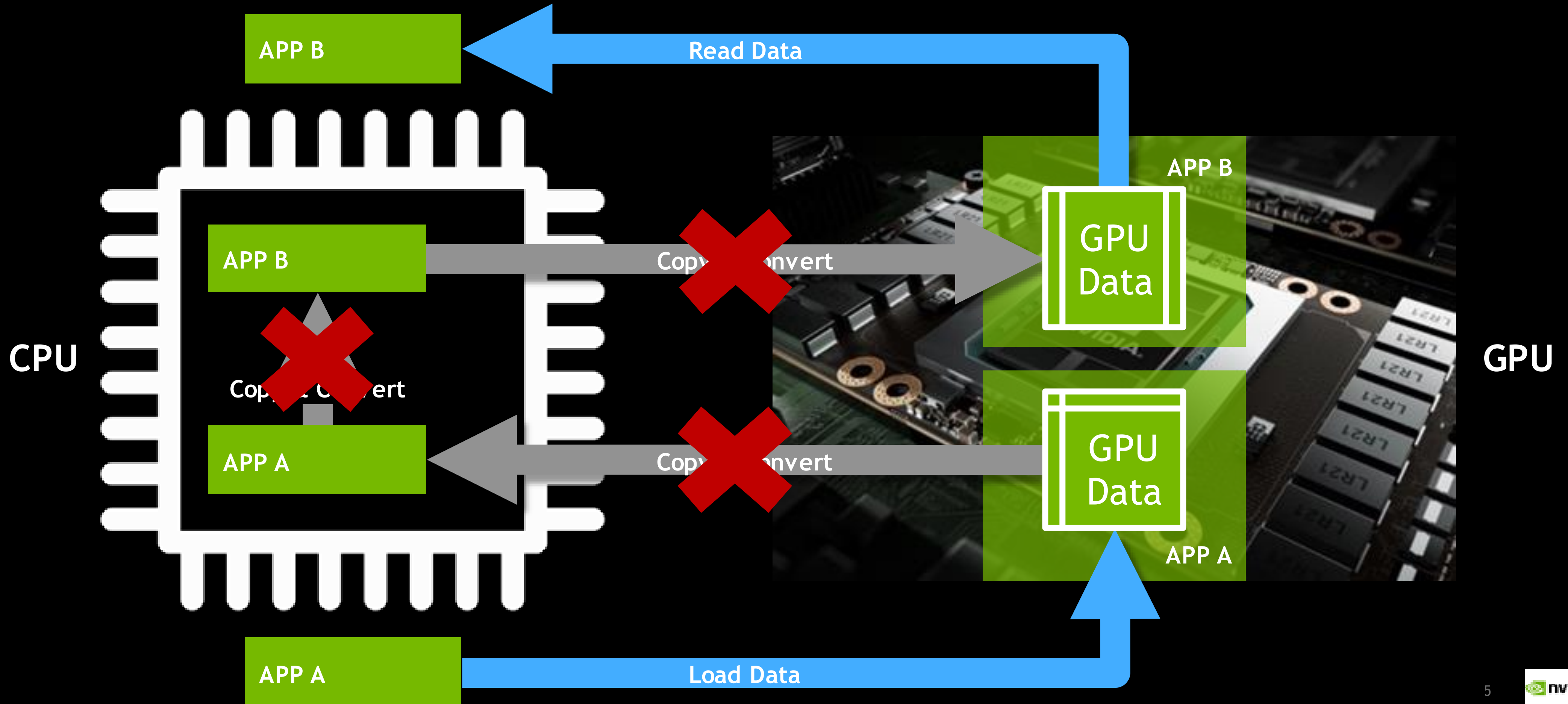
The bane of productivity and performance



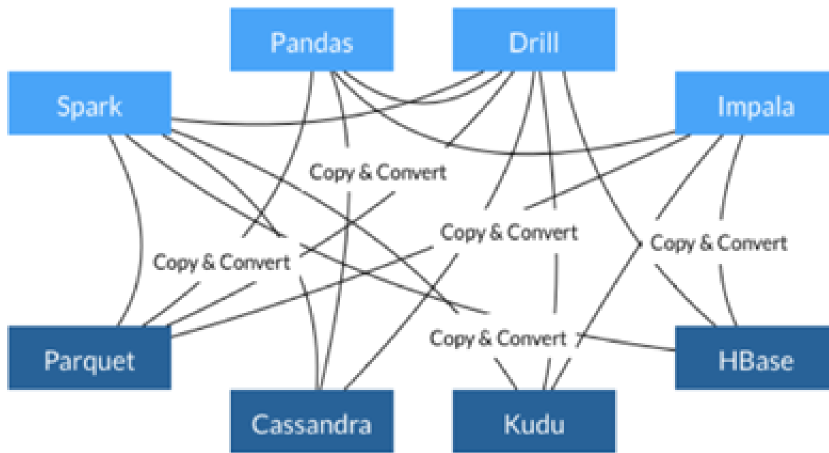
NVIDIA

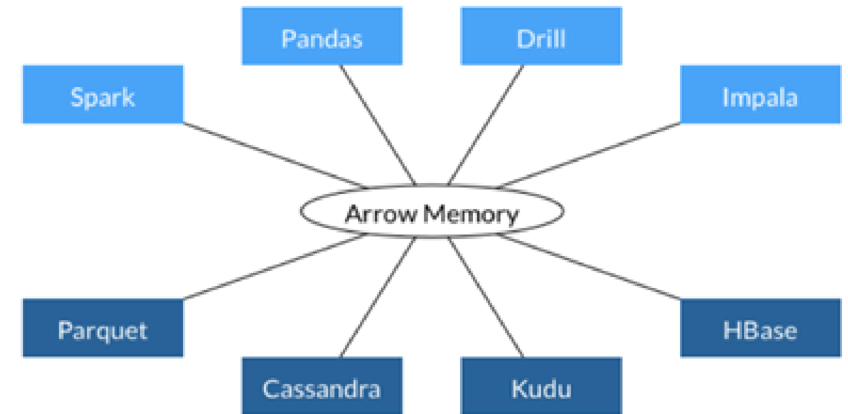
Data Movement and Transformation

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/

NVIDIA

Data Processing Evolution

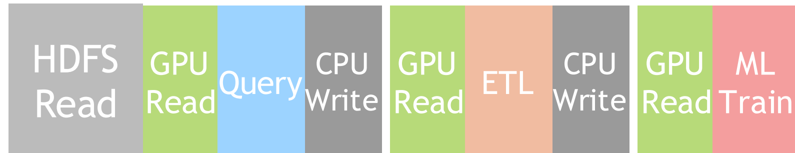
Faster data access, less data movement

Hadoop Processing, Reading from disk

Spark In-Memory Processing

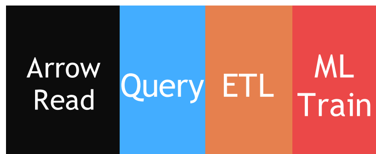
25-100x Improvement
Less code
Language flexible
Primarily In-Memory

Traditional GPU Processing



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

RAPIDS

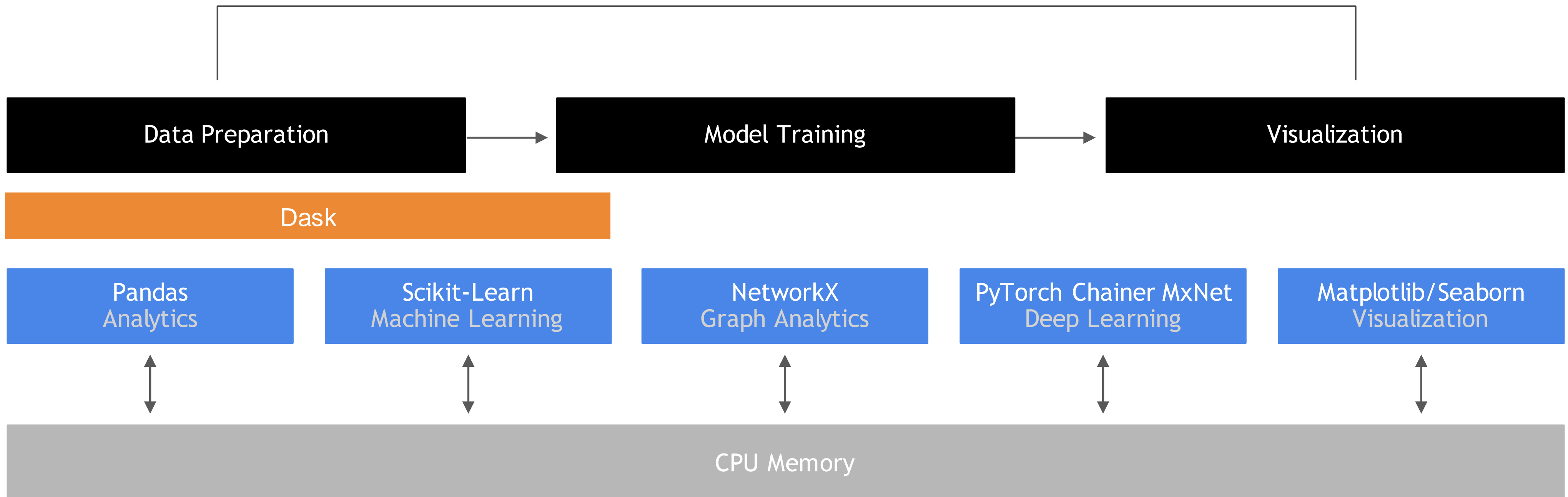


50-100x Improvement
Same code
Language flexible
Primarily on GPU

NVIDIA

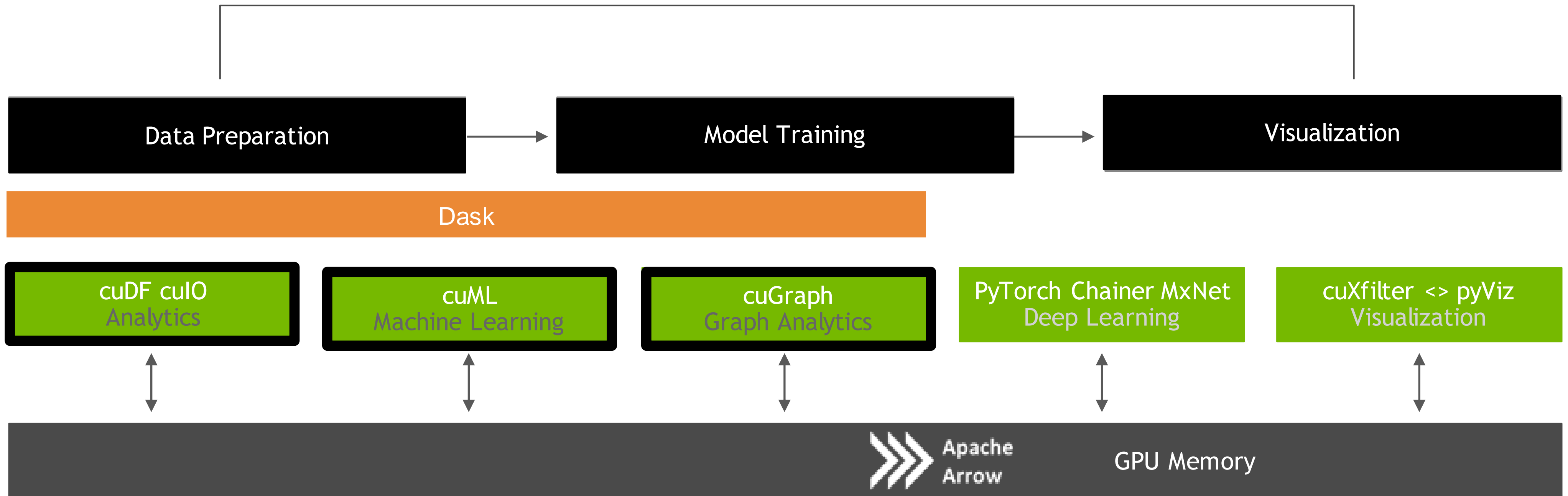
Open Source Data Science Ecosystem

Familiar Python APIs

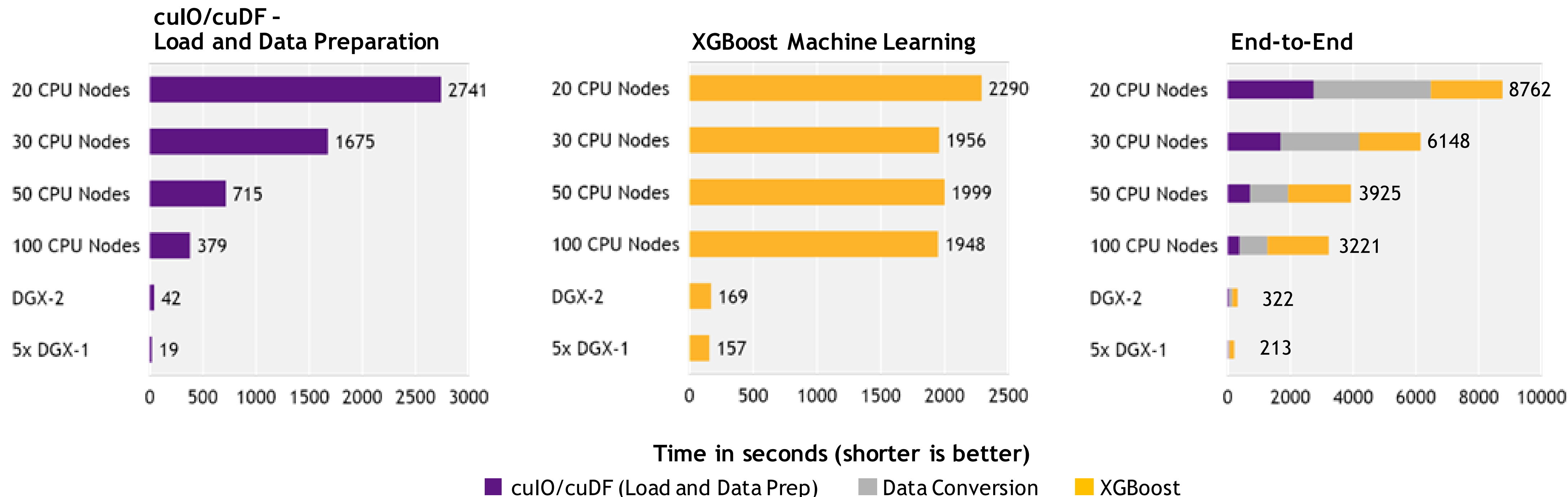


RAPIDS

End-to-End Accelerated GPU Data Science



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

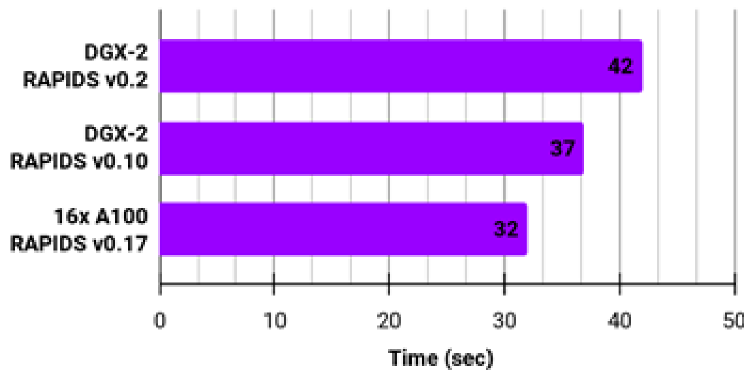
5x DGX-1 on InfiniBand network

NVIDIA

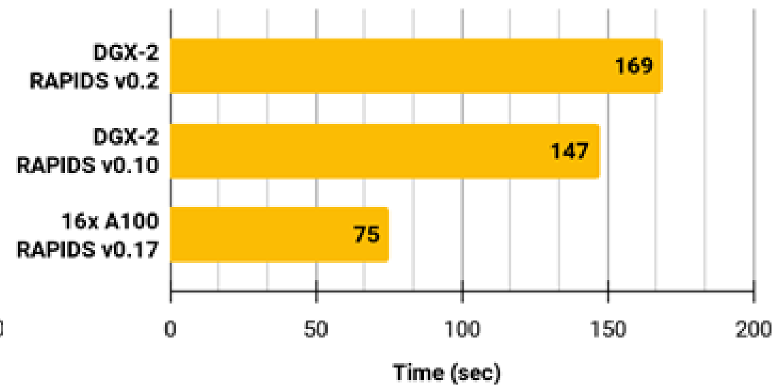
Faster Speeds, Real-World Benefits

Even Better with A100 and RAPIDS 0.17

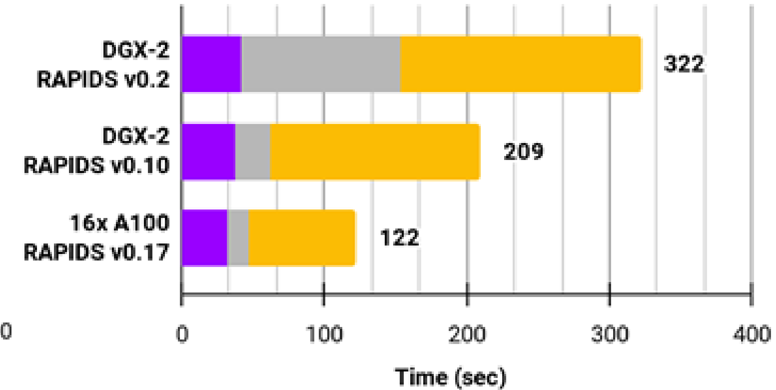
cuIO/cuDF - Load and Data Preparation



XGBoost Machine Learning



End-to-End



Time in seconds (shorter is better)

cuIO/cuDF (Load and Data Prep) Data Conversion XGBoost

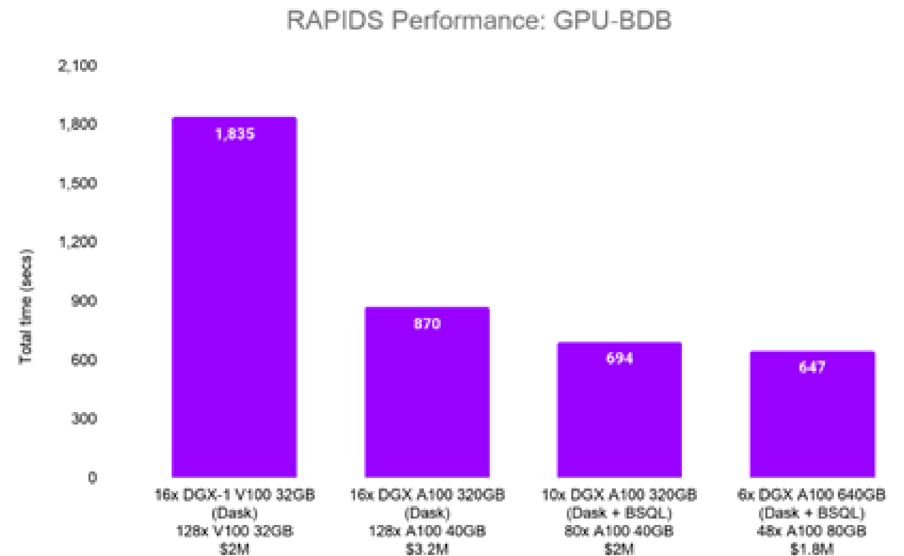
Benchmark	CPU Cluster Configuration	A100 Cluster Configuration
200GB CSV dataset; Data prep includes joins, variable transformations	CPU nodes (61 GiB memory, 8 vCPUs, 64-bit platform), Apache Spark	16 A100 GPUs (40GB each)

NVIDIA

Lightning-fast performance on real-world use cases

GPU Big Data Benchmark (GPU-BDB) is a data science benchmark derived from TPCx-BB[1], consisting of 30 end-to-end queries representing real-world ETL and Machine Learning workflows. It involves both structured and unstructured data. The benchmark starts with reading data from disk, performs common analytical and ML techniques (including NLP), then writes results back to disk to simulate a real world workflow.

Results at 10TB scale show RAPIDS' performance increasing over time, while TCO continues to go down. The recently announced DGX-A100 640GB is perfectly suited to data science workloads, and lets us do more work in almost half as many nodes as the DGX-A100 320GB (6 nodes vs 10) for even better TCO.



Continuous Improvement

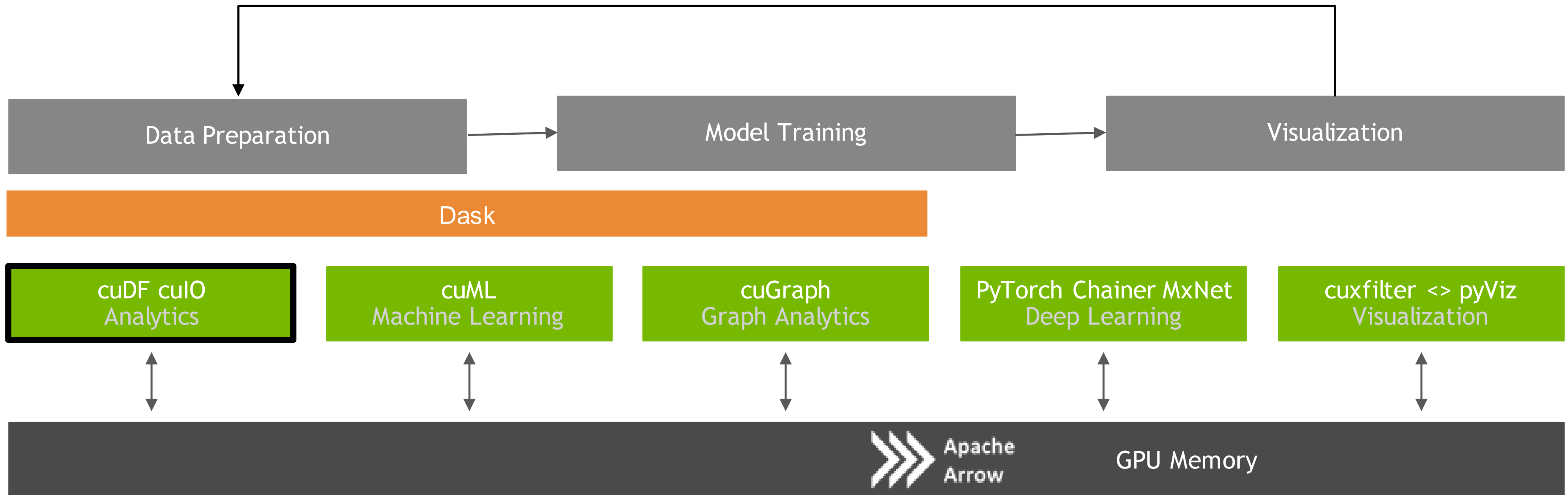
- 2.8x performance, almost a third the nodes, and cheaper to boot—in <1 year
- BlazingSQL at 10TB showing 25% improvement compared to Dask over TCP
- Q27 faster and more accurate with hugging Face

1: GPU-BDB is derived from the TPCx-BB benchmark and is used for internal performance testing. Results from GPU-BDB are not comparable to TPCx-BB.

cuDF

RAPIDS

GPU Accelerated data wrangling and feature engineering



NVIDIA

cuDF — ANALYTICS

GPU DataFrame Library Built on Apache Arrow

Libraries

Dask-cuDF: Distributed Computing using Dask; Support for multi-GPU, multi-node

cuDF: Python bindings for libcudf (Pandas like API for DataFrame manipulation)

libcudf: CUDA C++ Apache Arrow GPU DataFrame and operators (Join/Merges, GroupBys, Sort, Filters, etc.)

Dask-cuDF
Distributed Computing

cuDF
Python Bindings

libcudf
CUDA C++ Implementation

cuIO — FILE I/O Direct File Loading to cuDF

Availability	Supported File Formats
Now	CSV, Parquet, ORC, JSON

Extraction is the Cornerstone

cuDF I/O 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

- GPUDirect Storage integration in progress for bypassing PCIe bottlenecks!

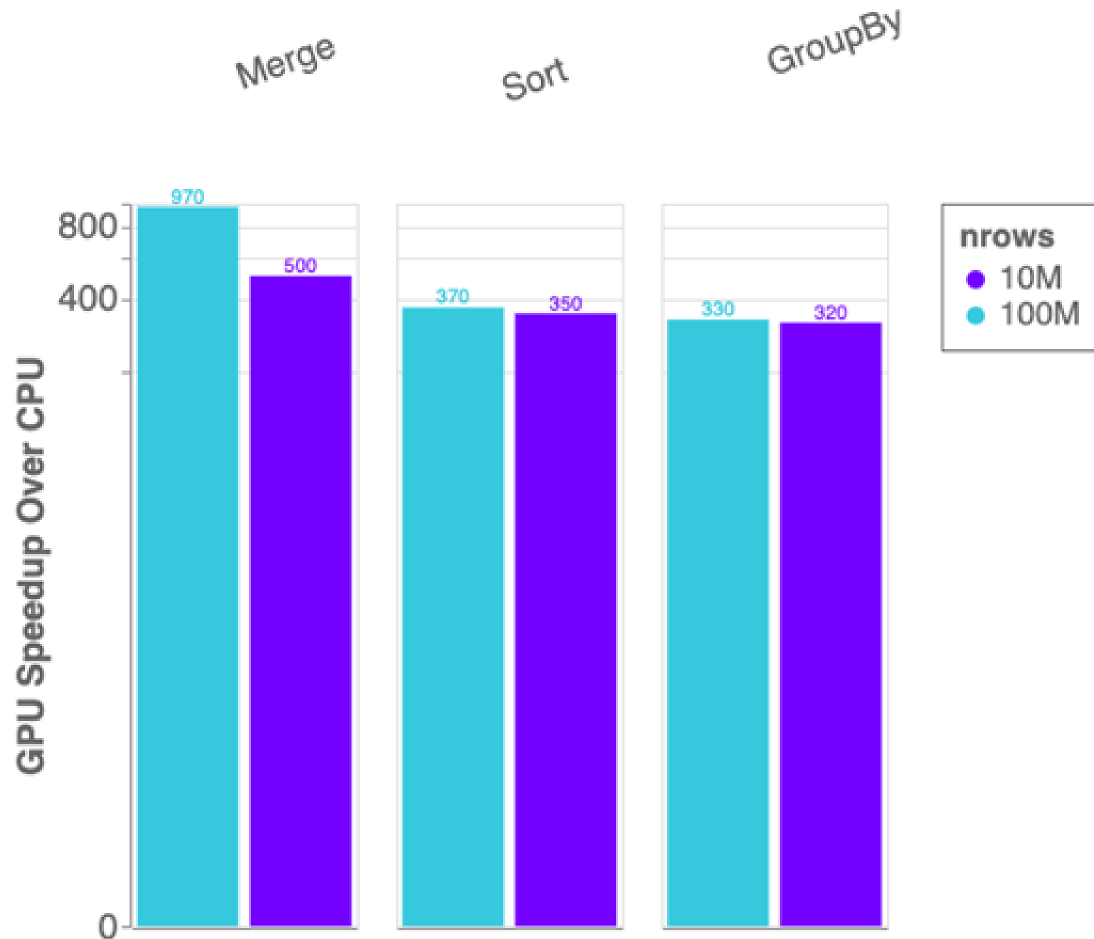
- Key is GPU-accelerating both parsing and decompression wherever possible

```
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

NVIDIA

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

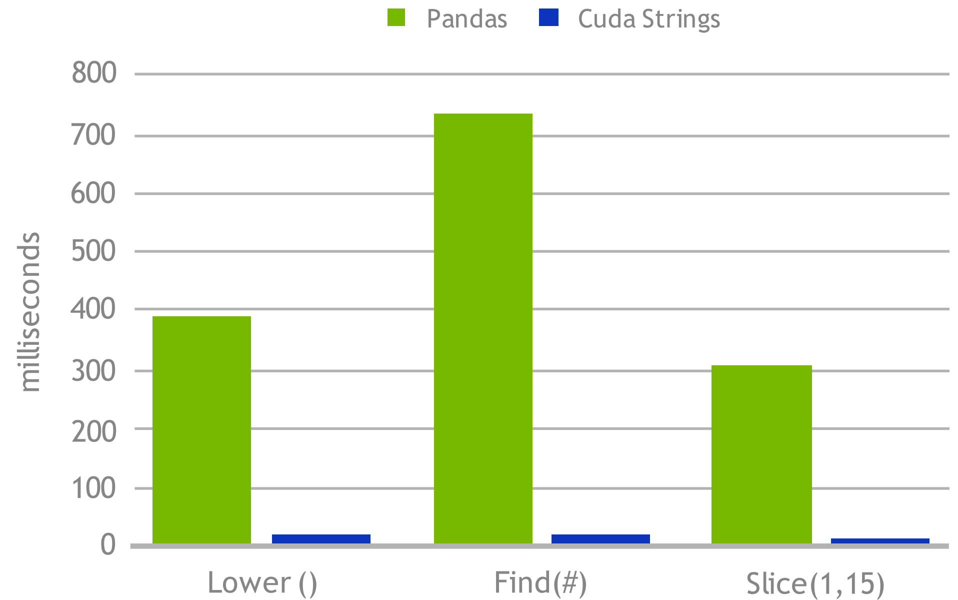
DataFrames: 2x int32 columns key columns, 3x int32 value columns

Merge: inner; GroupBy: count, sum, min, max calculated for each value column

cuDF String Support

Current v0.16 String Support

- Regular Expressions
- Element-wise operations
  - Split, Find, Extract, Cat, Typecasting, etc…
- String GroupBys, Joins, Sorting, etc.
- Categorical columns fully on GPU
- Native String type in libcudf C++
- NLP Preprocessors
  - Tokenizers, Normalizers, Edit Distance, Porter Stemmer, etc.
- Further performance optimization
- JIT-compiled String UDFs



NVIDIA

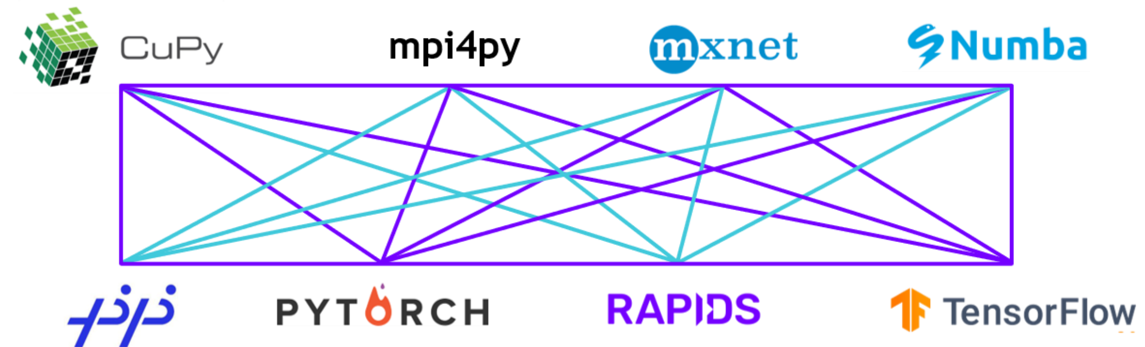
Interoperability for the Win

Real-world workflows often need to share data between libraries

RAPIDS supports device memory sharing between many popular data science and deep learning libraries

Keeps data on the GPU--avoids costly copying back and forth to host memory

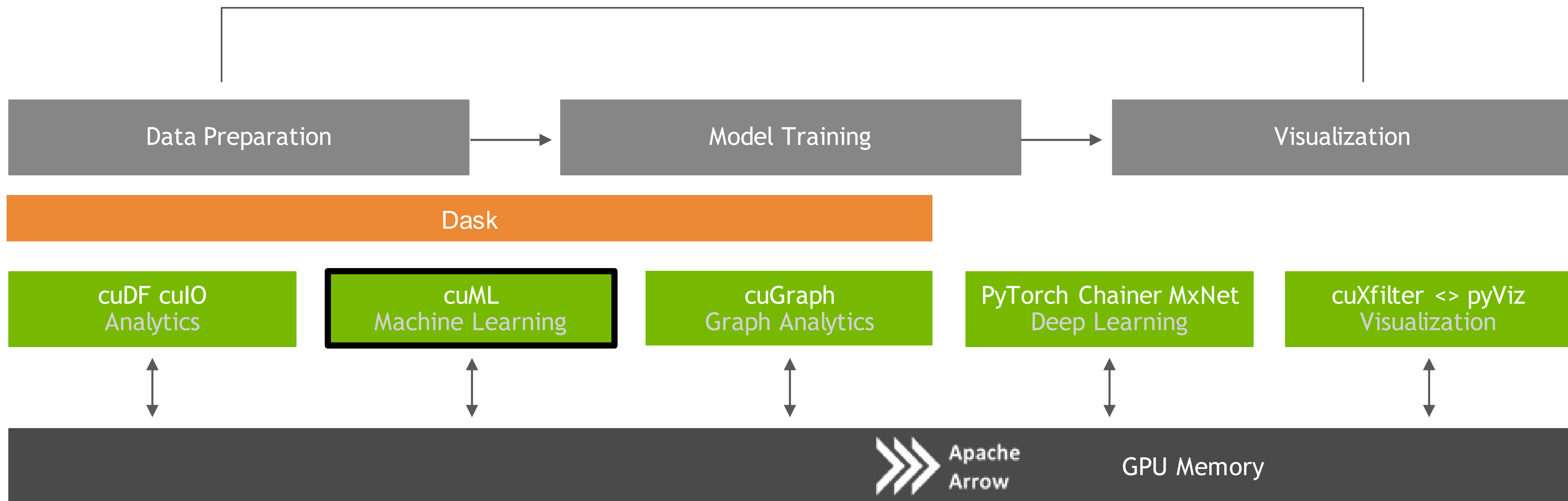
Any library that supports DLPack or __cuda_array_interface__ will allow for sharing of memory buffers between RAPIDS and supported libraries



cuML

Machine Learning

More models more problems



cuML — MACHINE LEARNING

GPU Accelerated Scikit-learn + XGBoost Libraries

Dask

Distributed Training: Used for distributed cuML model training

Python API

Language Bindings: Python bindings to C++/CUDA based cuML

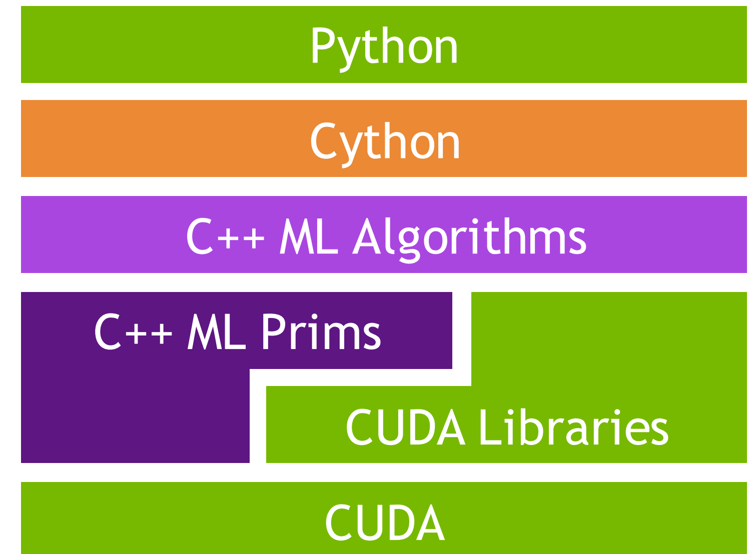
Uses cuDF DataFrames as input

cuML

C++/CUDA ML Algorithms: C++/CUDA machine learning algorithms

ml-prims

CUDA ML Primitives: Low level machine learning primitives used in cuML | Linear algebra, statistics, matrix operations, distance functions, random number generation

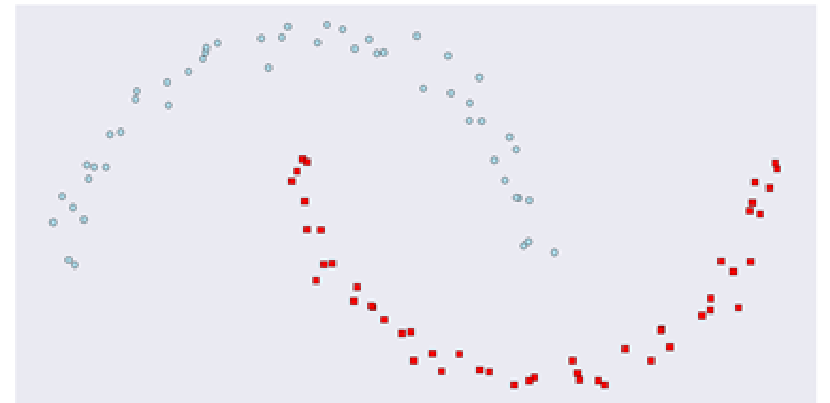


RAPIDS matches common Python APIs

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)
```



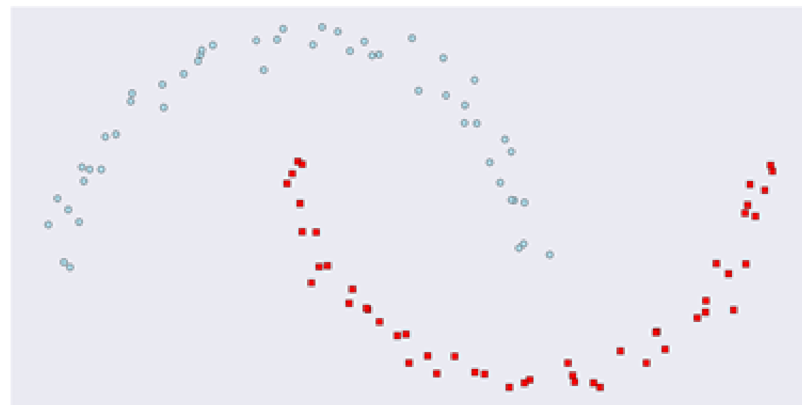
NVIDIA

RAPIDS matches common Python APIs

GPU-Accelerated Clustering

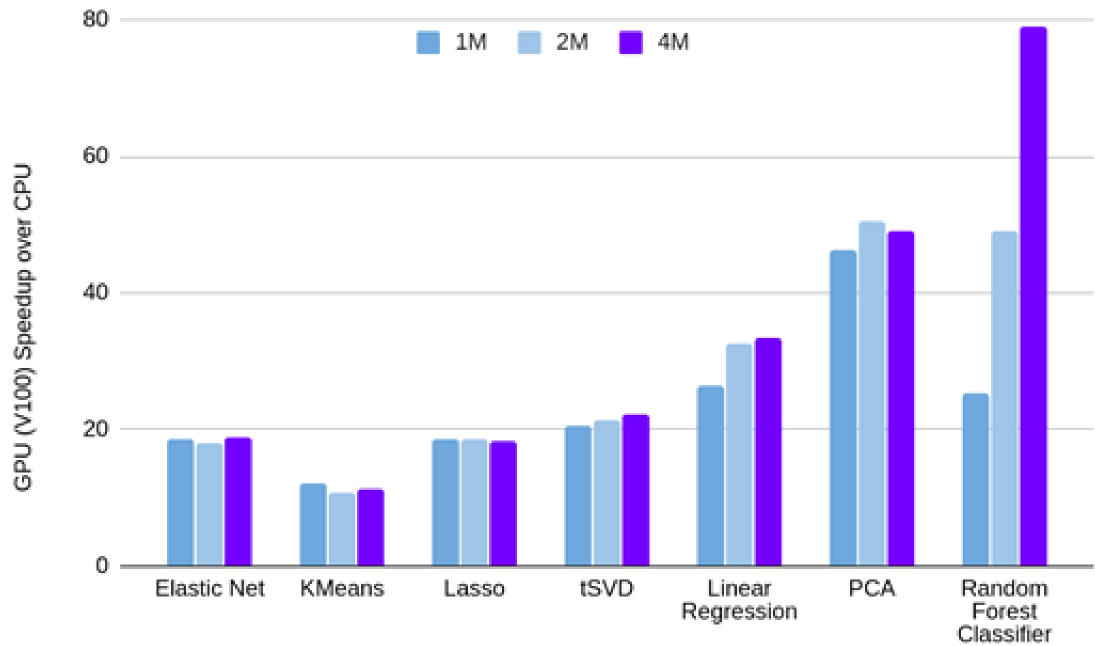
```
from sklearn.datasets import make_moons
import cudf

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])})
```

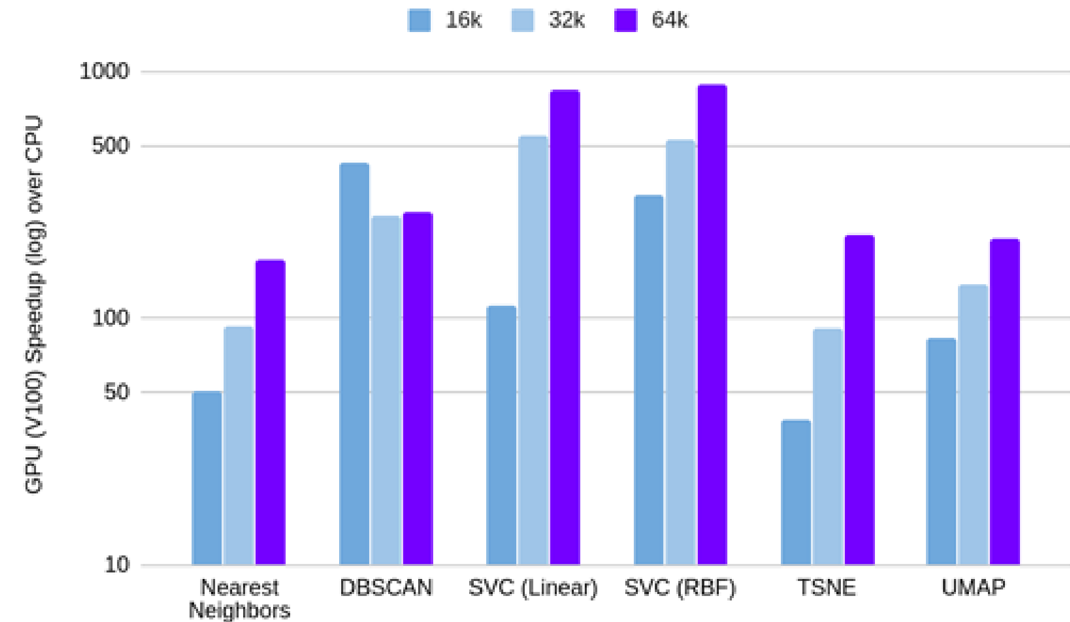
```
from cuml          import DBSCAN
dbscan = DBSCAN(eps = 0.3, min_samples  = 5)

dbscan.fit(X)

y_hat = dbscan.predict(X)
```



NVIDIA

Benchmarks: Single-GPU cuML vs Scikit-learn





1x V100 vs. 2x 20 Core CPUs (DGX-1, RAPIDS 0.15)

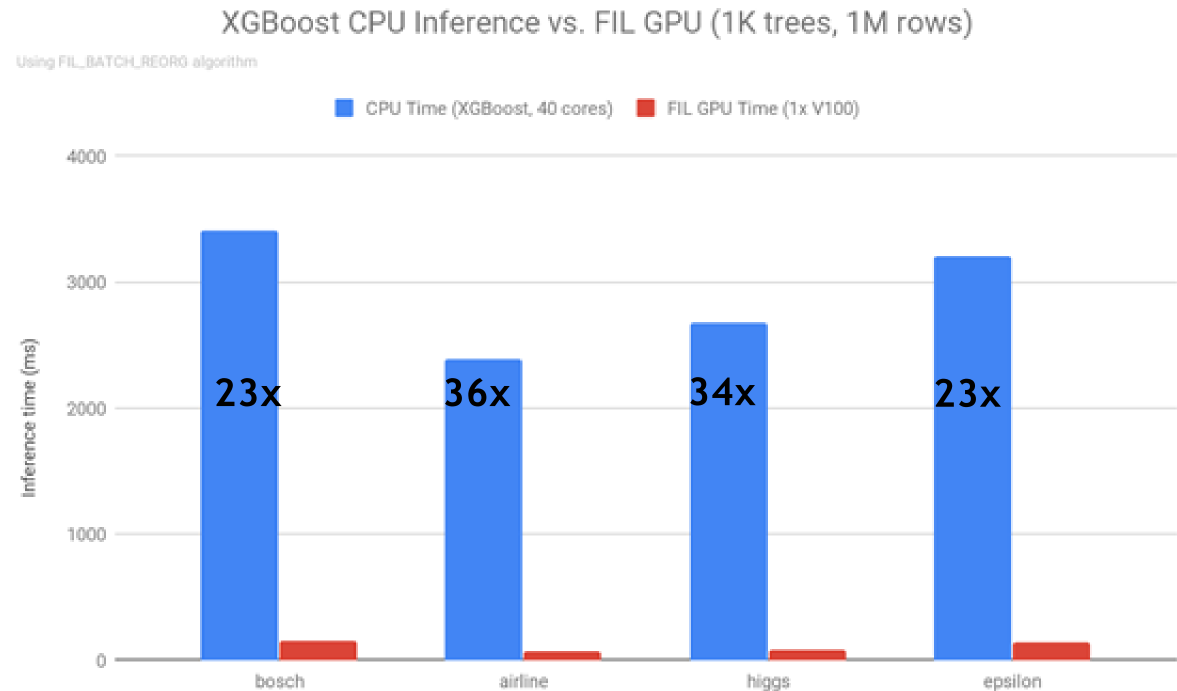
NVIDIA

Forest Inference

Taking models from training to production

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

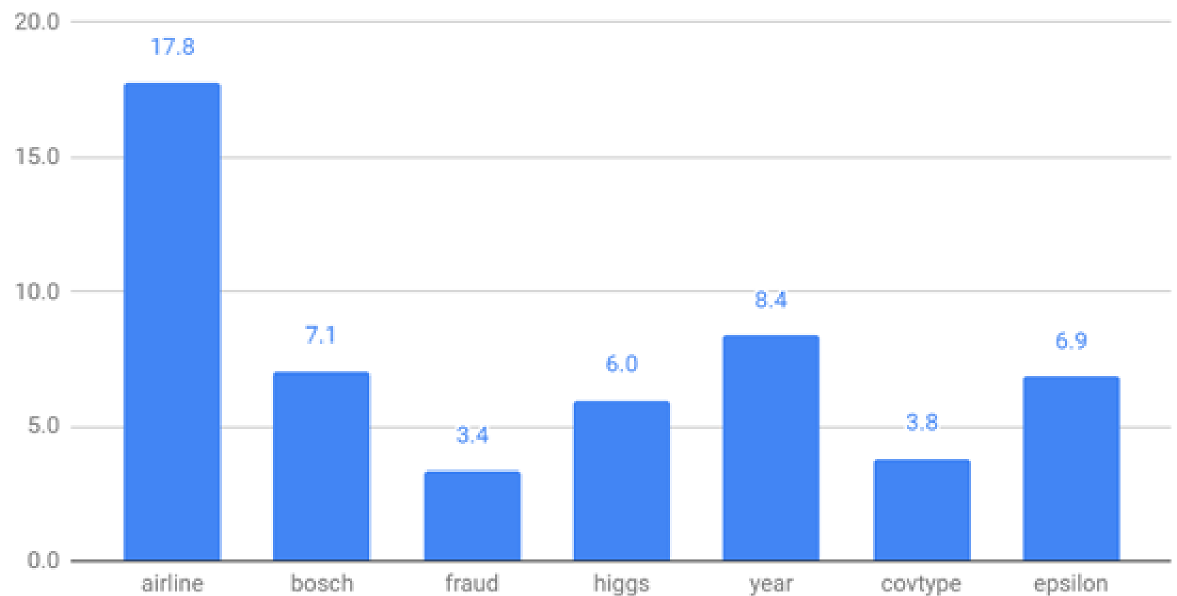
- Works with existing saved models (XGBoost, LightGBM, scikit-learn RF cuML RF soon)
- Lightweight Python API
- Single V100 GPU can infer up to 34x faster than XGBoost dual-CPU node
- Over 100 million forest inferences per sec (with 1000 trees) on a DGX-1 for large (sparse) or dense models



NVIDIA

+

- RAPIDS works closely with the XGBoost community to accelerate GBDTs on GPU
- XGBoost can seamlessly load data from cuDF dataframes and cuPy arrays
- Dask allows XGBoost to scale to arbitrary numbers of GPUs
- With the *gpu_hist* tree method, a single GPU can outpace 10s to 100s of CPUs
- RAPIDS comes paired with XGBoost 1.1 (as of 0.14)



NVIDIA

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.

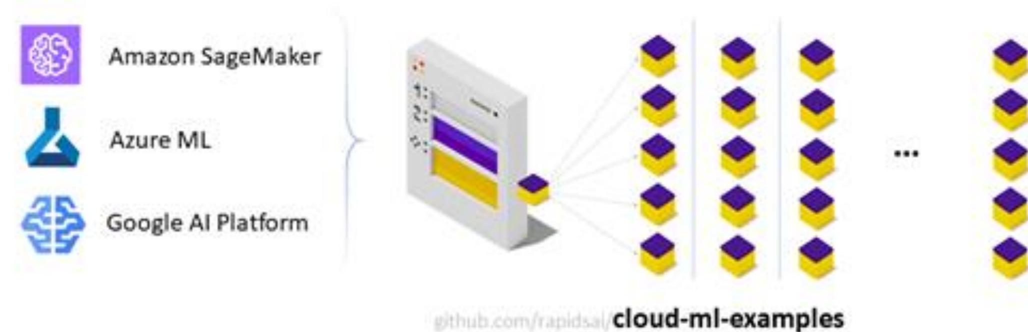
With GPU acceleration, RAPIDS models can train 25x faster than CPU equivalents, enabling more experimentation in less time.

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

MANY PATHS TO RAPIDS HPO

RAPIDS Integration into Cloud/Distributed Frameworks

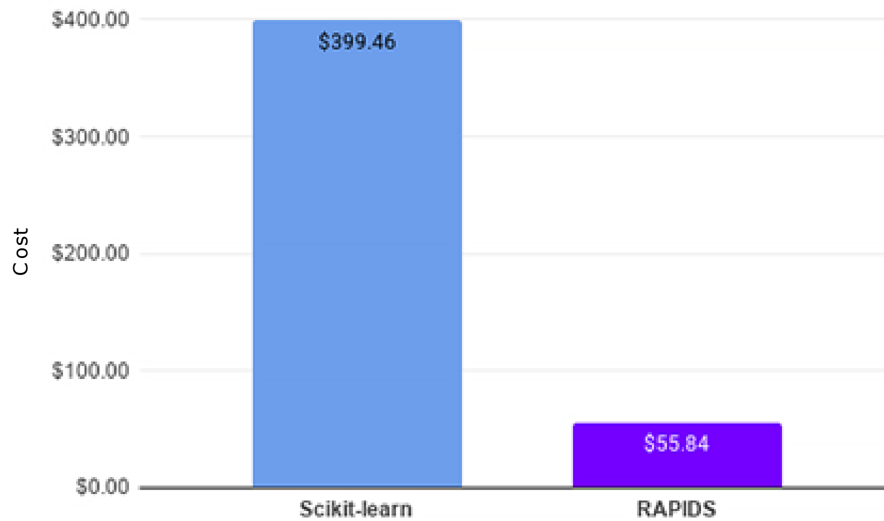


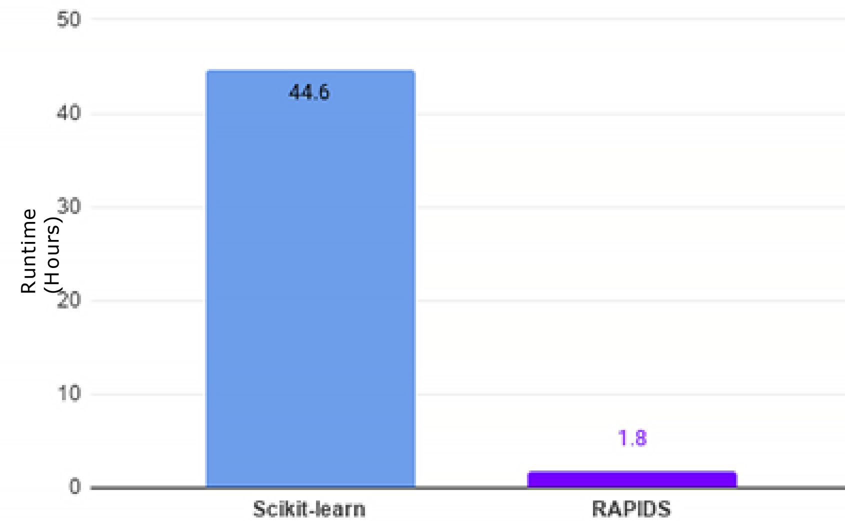
Ray/Tune

Dask ML

HPO Use Case: 100-Job Random Forest Airline Model

Huge speedups translate into >7x TCO reduction





Based on sample Random Forest training code from cloud-ml-examples repository, running on Azure ML. 10 concurrent workers with 100 total runs, 100M rows, 5-fold cross-validation per run.

GPU nodes: 10x Standard_NC6s_v3, 1 V100 16G, vCPU 6 memory 112G, Xeon E5-2690 v4 (Broadwell) - $3.366/hour
CPU nodes: 10x Standard_DS5_v2, vCPU 16 memory 56G, Xeon E5-2673 v3 (Haswell) or v4 (Broadwell) - $1.017/hour"

Road to 1.0 - cuML

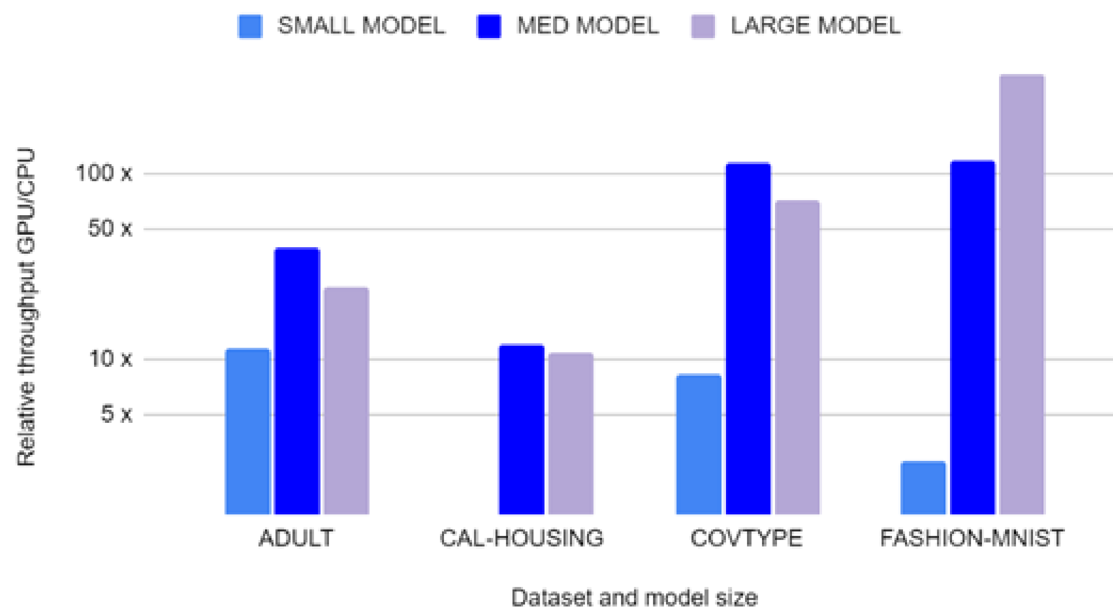
RAPIDS 0.17 - December 2020

cuML	Single-GPU	Multi-Node-Multi-GPU
Gradient Boosted Decision Trees (GBDT)		
Linear Regression		
Logistic Regression		
Random Forest		
K-Means		
K-NN		
DBSCAN		
UMAP		
Holt-Winters		
ARIMA		
T-SNE		
Principal Components		
Singular Value Decomposition		
SVM		

SHAP Explainability

GPUTreeSHAP for XGBoost

- SHAP provides a principled way to explain the impact of input features on each prediction or on the model overall - critical for interpretability
- SHAP has often been too computationally-expensive to deploy for large-scale production
- RAPIDS ships with GPU-accelerated SHAP for XGBoost with speedups of 20x or more (demo code available in the XGBoost repo)
- RAPIDS 0.17 includes experimental Kernel and Permutation explainers for black box models

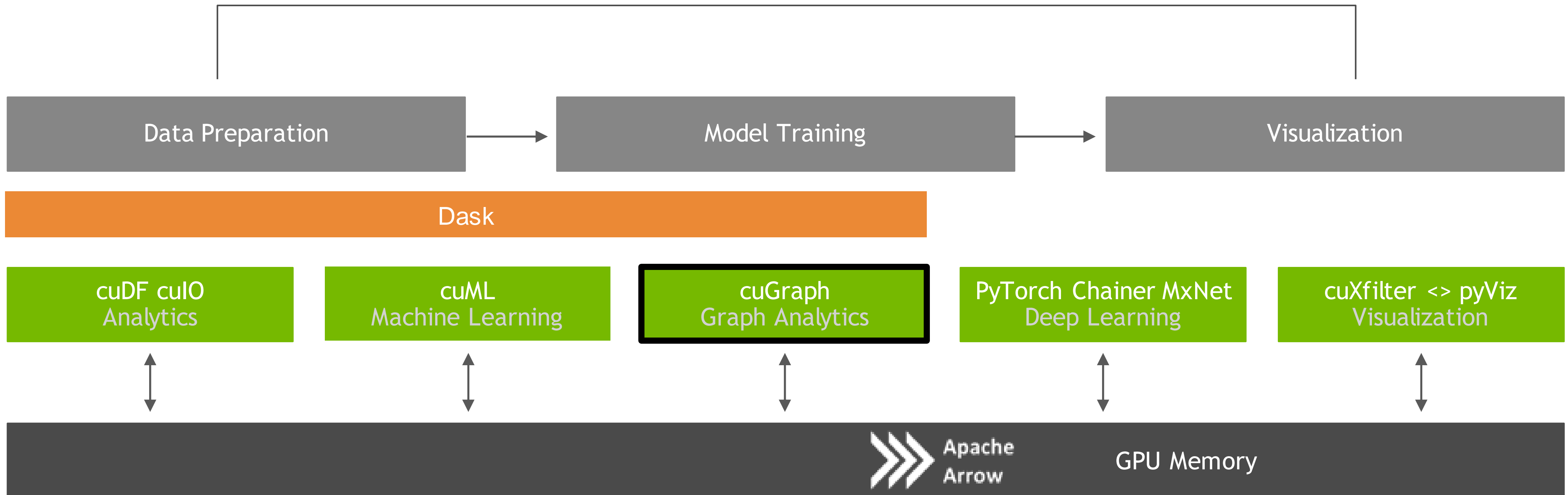


NVIDIA

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

SEAMLESS INTEGRATION WITH cuDF AND cuML

- Property Graph support via DataFrames

MULTIPLE APIs

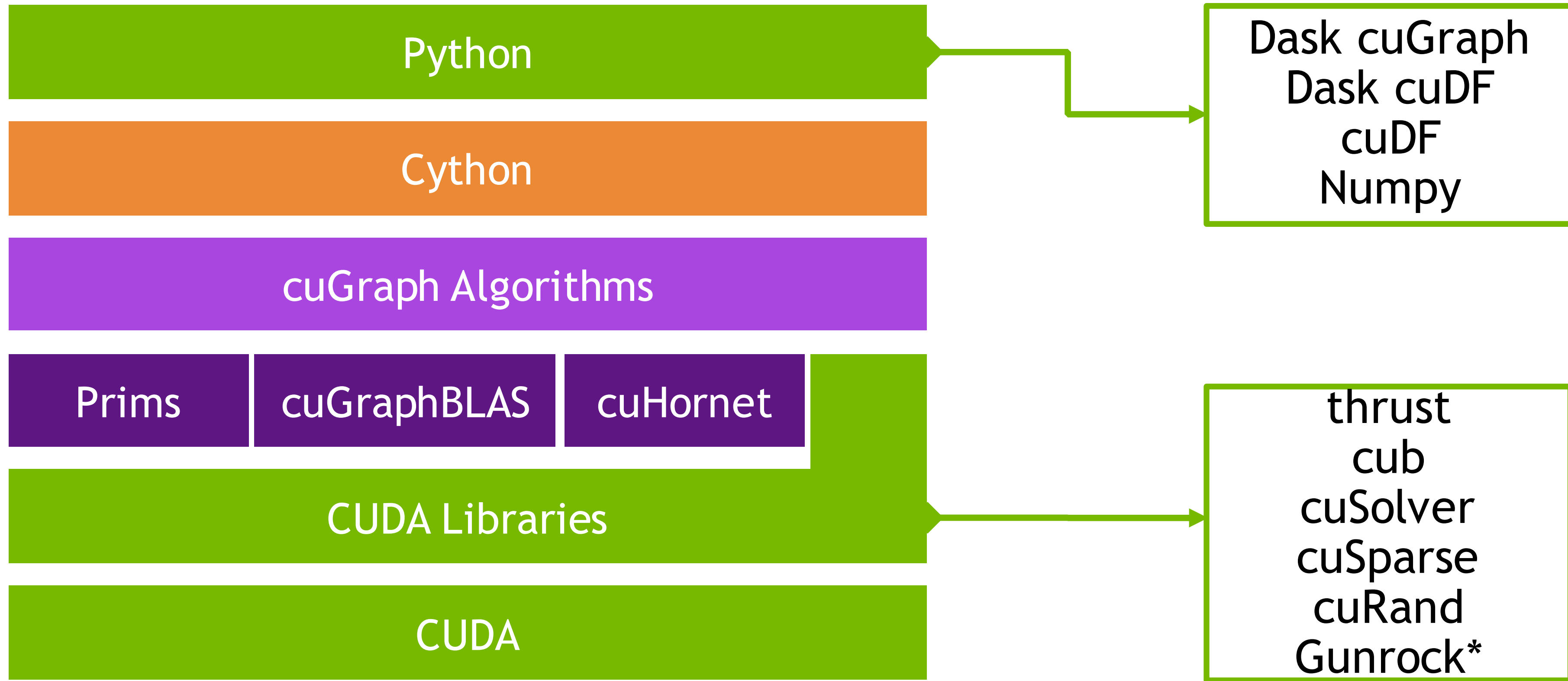
- **Python:** Familiar NetworkX-like API
- **C/C++:** lower-level granular control for application developers

GROWING FUNCTIONALITY

- Extensive collection of algorithm, primitive, and utility functions

NVIDIA

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

NVIDIA

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

*BigData x8, 100x 8-vCPU nodes, Apache Spark GraphX ⇒ 96 mins!

Road to 1.0 - cuGraph

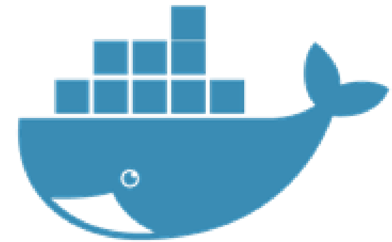
RAPIDS 0.17 - December 2020

cuGRAPH	Single-GPU	Multi-Node-Multi-GPU
Page Rank		
Personal Page Rank		
Katz		
Betweenness Centrality		
Spectral Clustering		
Louvain		
Ensemble Clustering for Graphs		
K-Truss & K-Core		
Connected Components (Weak & Strong)		
Triangle Counting		
Single Source Shortest Path (SSSP)		
Breadth-First Search (BFS)		
Jaccard & Overlap Coefficient		
Force Atlas 2		
Hungarian Algorithm		
Leiden		

HOW TO GET STARTED AND EXAMPLES

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/

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.

NOTICES	RAPIDS repos will rename stable/release branches in v0.15			Python 3.6 & CUDA 10.0 EOL in v0.14			
	Release changes to clx and cuxfilter in v0.15			RAPIDS dask-xgboost library is deprecated in v0.15			
	Preferred			Advanced			
METHOD	Conda	Docker + Examples		Docker + Dev Env	Source		
RELEASE	Legacy (0.14)	Stable (0.15)		Nightly (0.16a)			
PACKAGES	All Packages	cuDF	cuML	cuGraph	cuSignal	cuSpatial	cuxfilter
LINUX	Ubuntu 16.04	Ubuntu 18.04		CentOS 7	RHEL 7		
PYTHON	Python 3.6 (0.14 only)	Python 3.7		Python 3.8 (0.15/0.16 only)			
CUDA	CUDA 10.0 (0.14 only)	CUDA 10.1.2		CUDA 10.2	CUDA 11.0 (0.15/0.16 only)		

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

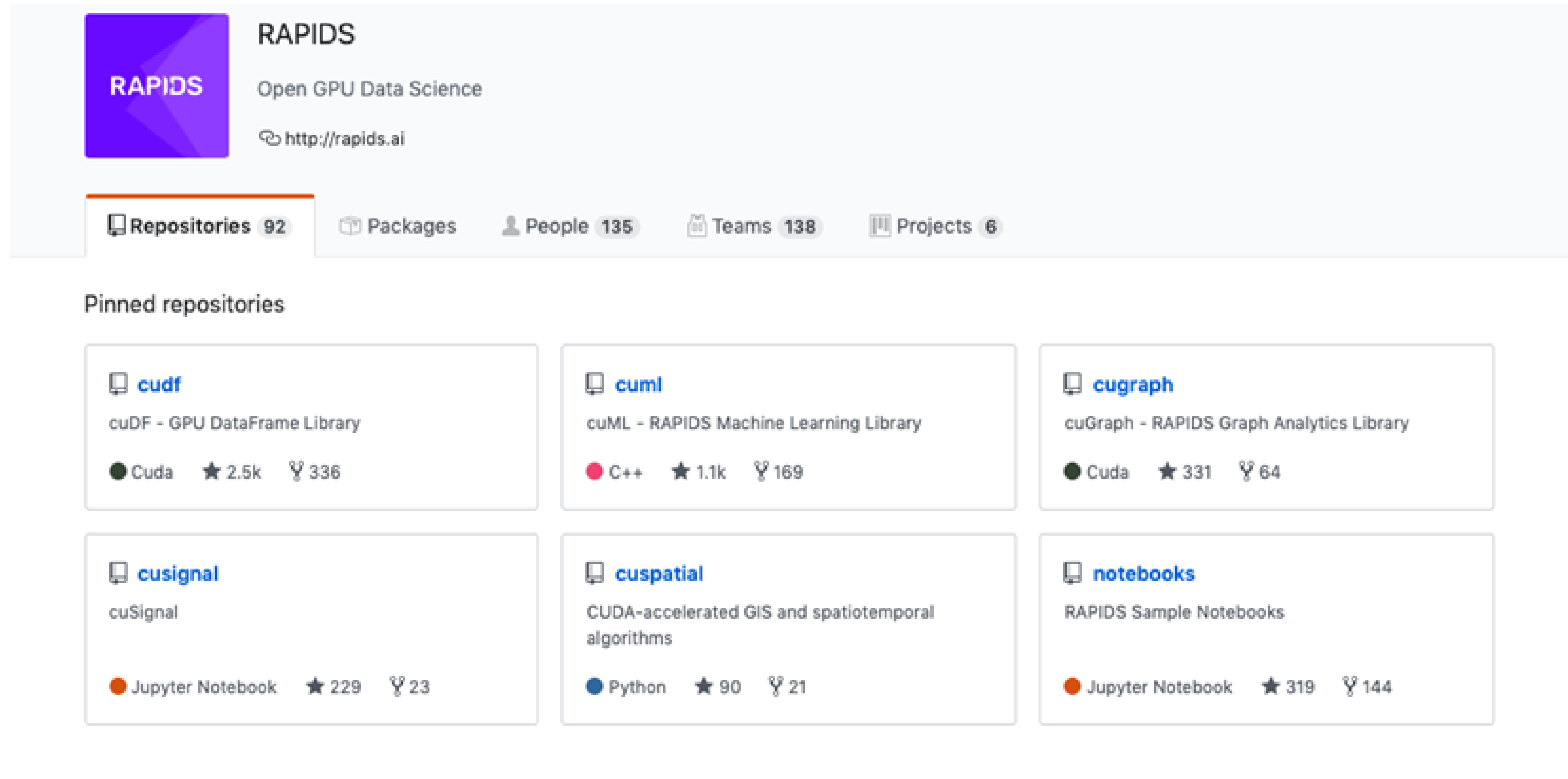
COMMAND

```
conda install -c rapidsai -c nvidia -c conda-forge \
    -c defaults rapids=0.15 python=3.7
```


NVIDIA

Explore: RAPIDS GitHub

https://github.com/rapidsai



NVIDIA

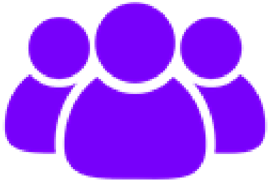
Notebook Examples

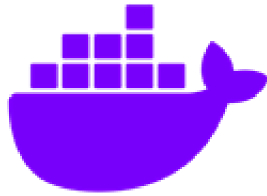
https://github.com/rapidsai/notebooks

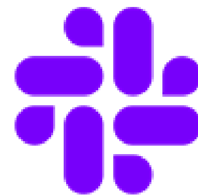
RAPIDS PREREQUISITES

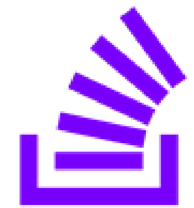
NVIDIA Pascal GPU architecture or better
CUDA 10.1, 10.2, 11.0 & compatible NVIDIA driver
Ubuntu 16.04/18.04, CentOs 7
Docker CE v19.03+
nvidia-docker v2+

Join the Conversation









GOOGLE GROUPS

https://groups.google.com/forum/#!forum/rapidsai

DOCKER HUB

https://hub.docker.com/r/rapidsai/rapidsai

SLACK CHANNEL

https://rapids-goai.slack.com/join

STACK OVERFLOW

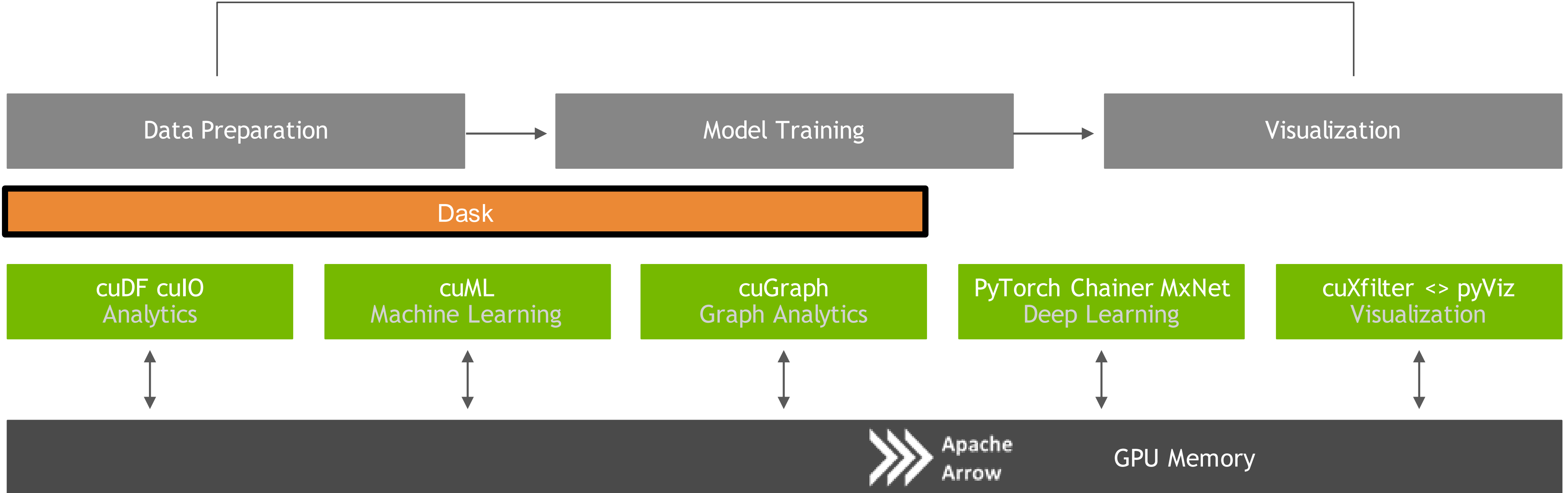
https://stackoverflow.com/tags/rapids

Contribute back: Issues, Feature Requests, PRs, Blogs, Tutorials, Videos, QA

Dask + GPUs

RAPIDS

Scaling RAPIDS with Dask



NVIDIA

Dask

What is Dask?

- Distributed compute scheduler built to scale Python
- Scales workloads from laptops to supercomputer clusters
- Extremely modular: disjoint scheduling, compute, data transfer and out-of-core handling
- Multiple workers per node allow easier one-worker-per-GPU model

NVIDIA

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

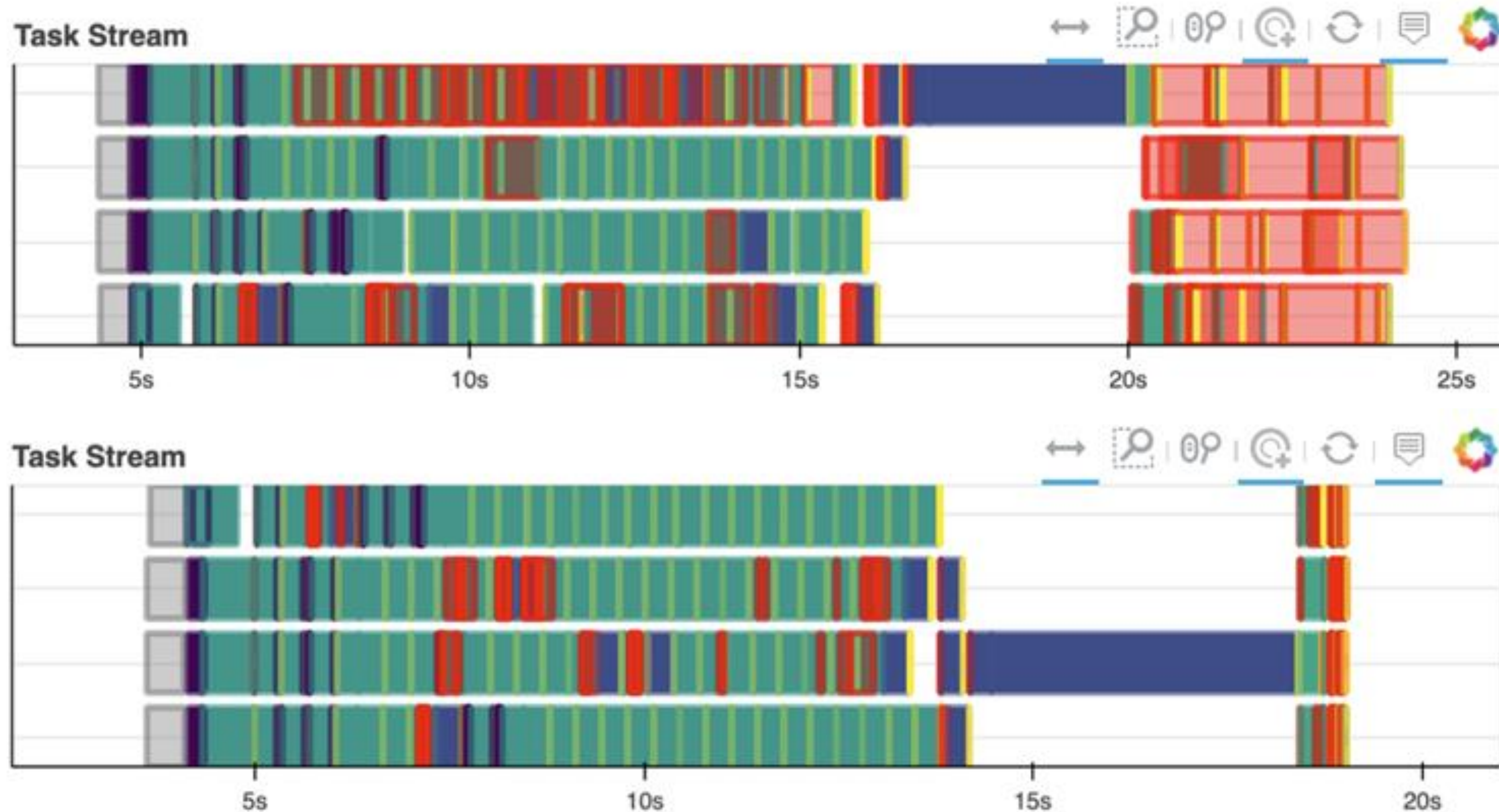
Why OpenUCX?

Bringing hardware accelerated communications to Dask

- TCP sockets are slow!
- UCX provides uniform access to transports (TCP, InfiniBand, shared memory, NVLink)
- Alpha Python bindings for UCX (ucx-py) https://github.com/rapidsai/ucx-py
- Will provide best communication performance, to Dask based on available hardware on nodes/cluster

OpenUCX

Dask Array SVD + CuPy Experiment with and without UCX



NVIDIA

Scale up with RAPIDS

Scale Up / Accelerate

RAPIDS and Others

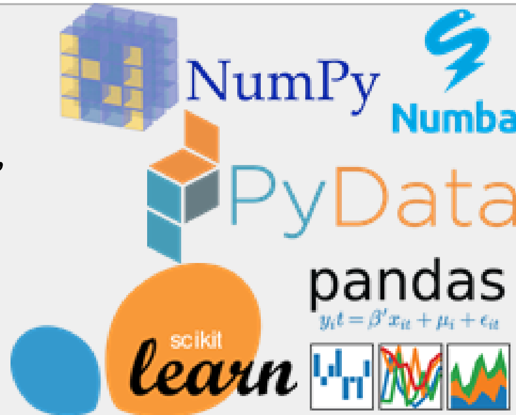
Accelerated on single GPU

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



NVIDIA

Scale out with RAPIDS + Dask with OpenUCX

RAPIDS and Others

Accelerated on single GPU

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

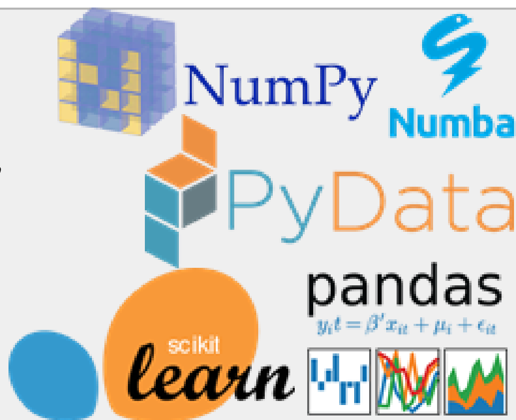
RAPIDS + Dask with OpenUCX

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 Up / Accelerate

Scale out / Parallelize

Development Environment

```
import distributed
from dask_cuda import LocalCUDACluster
import dask_cudf

cluster = LocalCUDACluster()
client = distributed.Client(cluster)
```

```
gdf = dask_cudf.read_csv('/datasets/nyc_taxi/**/*')
```

```
gdf = gdf.persist()
```

```
len(gdf)
```


Jupyter Lab - Dask Extension - NVDashboard Extension

THANK YOU!