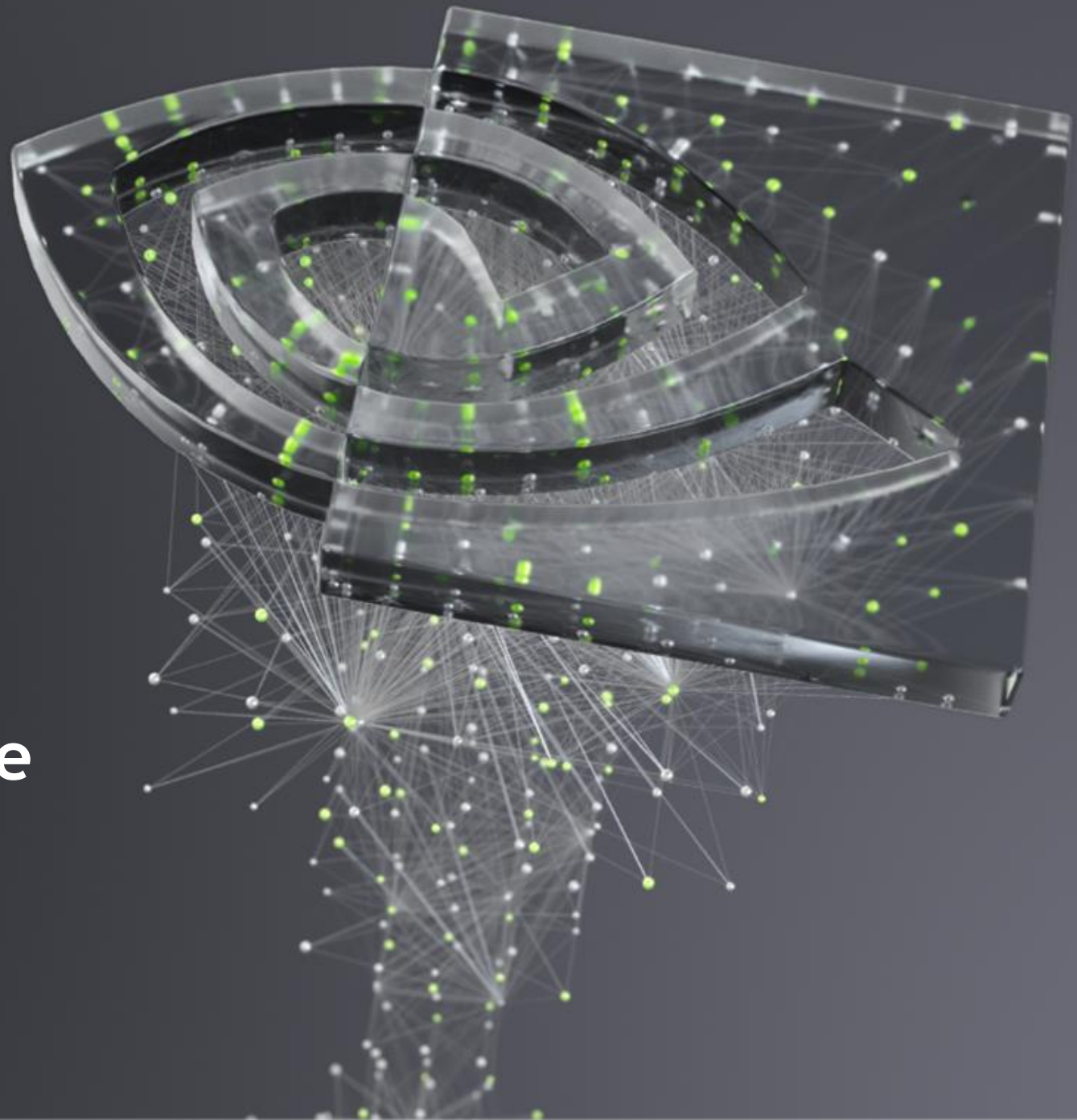


Accelerating Data Science Workloads with GPUs

October 2021

AGENDA

Intro to Accelerated Data Science & RAPIDS ecosystem

Accelerated Data Preparation and Machine learning

Scalable ETL & Machine learning

Inferencing and HPO at scale

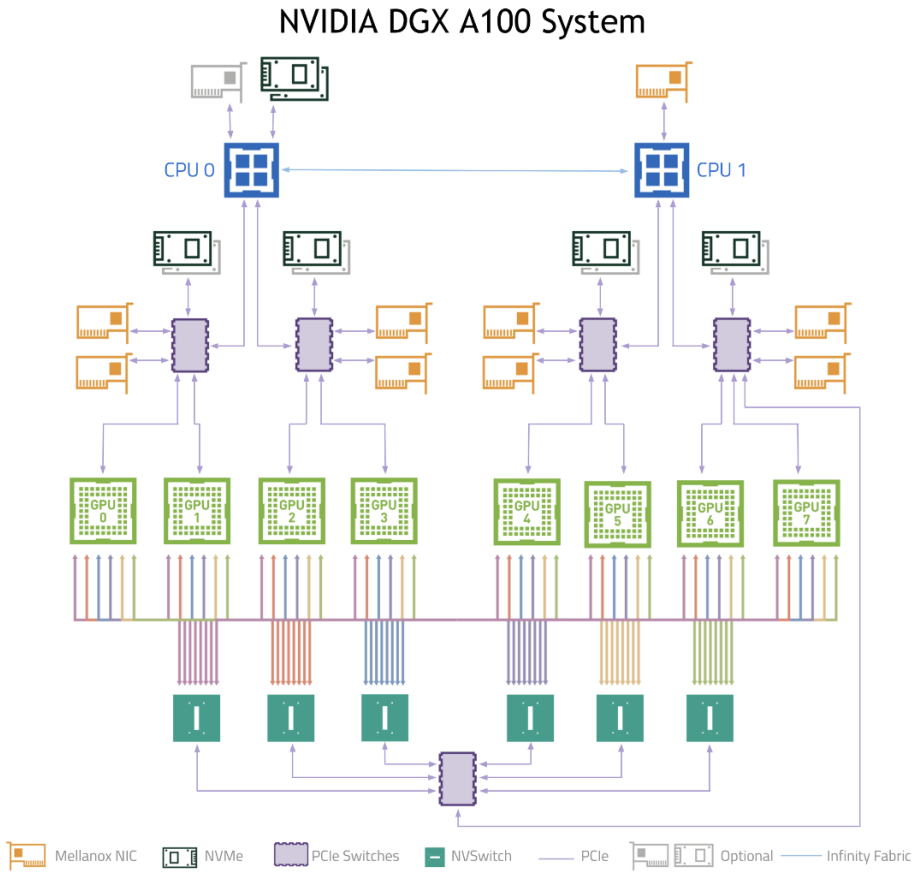
Model explain ability, visualization

WHY GPUS?

Numerous hardware advantages

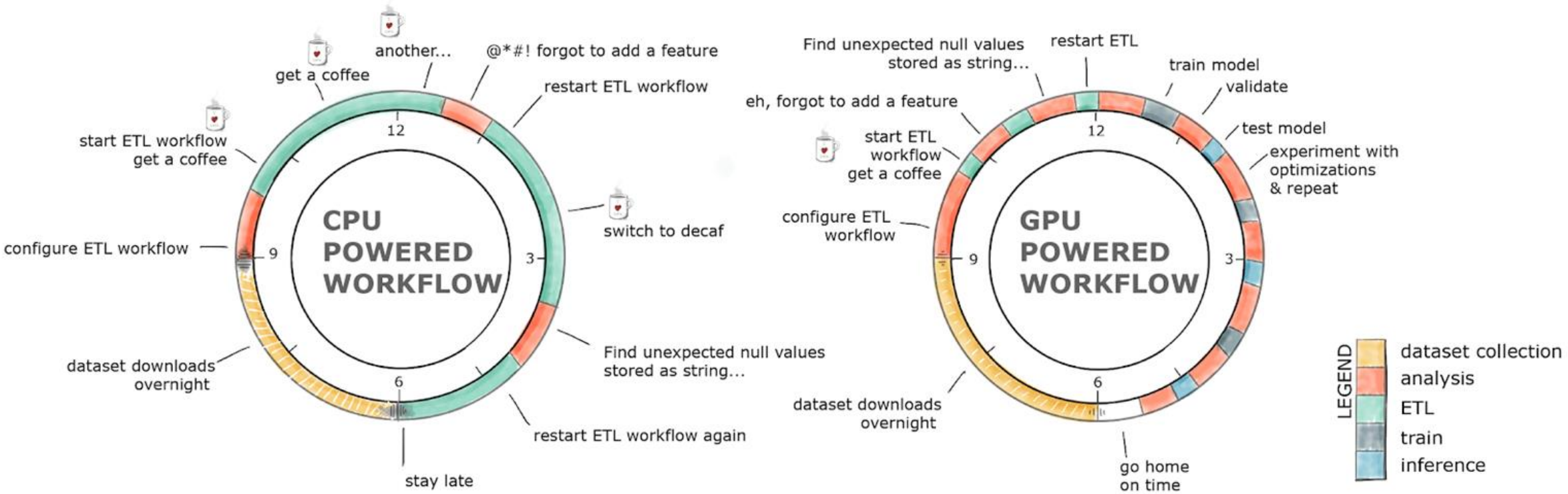
- Thousands of cores with up to ~20 TeraFlops of general purpose compute performance
- Up to 1.5 TB/s of memory bandwidth
- Hardware interconnects for up to 600 GB/s bidirectional GPU <--> GPU bandwidth
- Can scale up to 16x GPUs in a single node

Almost never run out of compute relative to memory bandwidth!



END-TO-END ML WORKFLOW

The Average Data Scientist Spends 80+% of Their Time in ETL as Opposed to Training Models



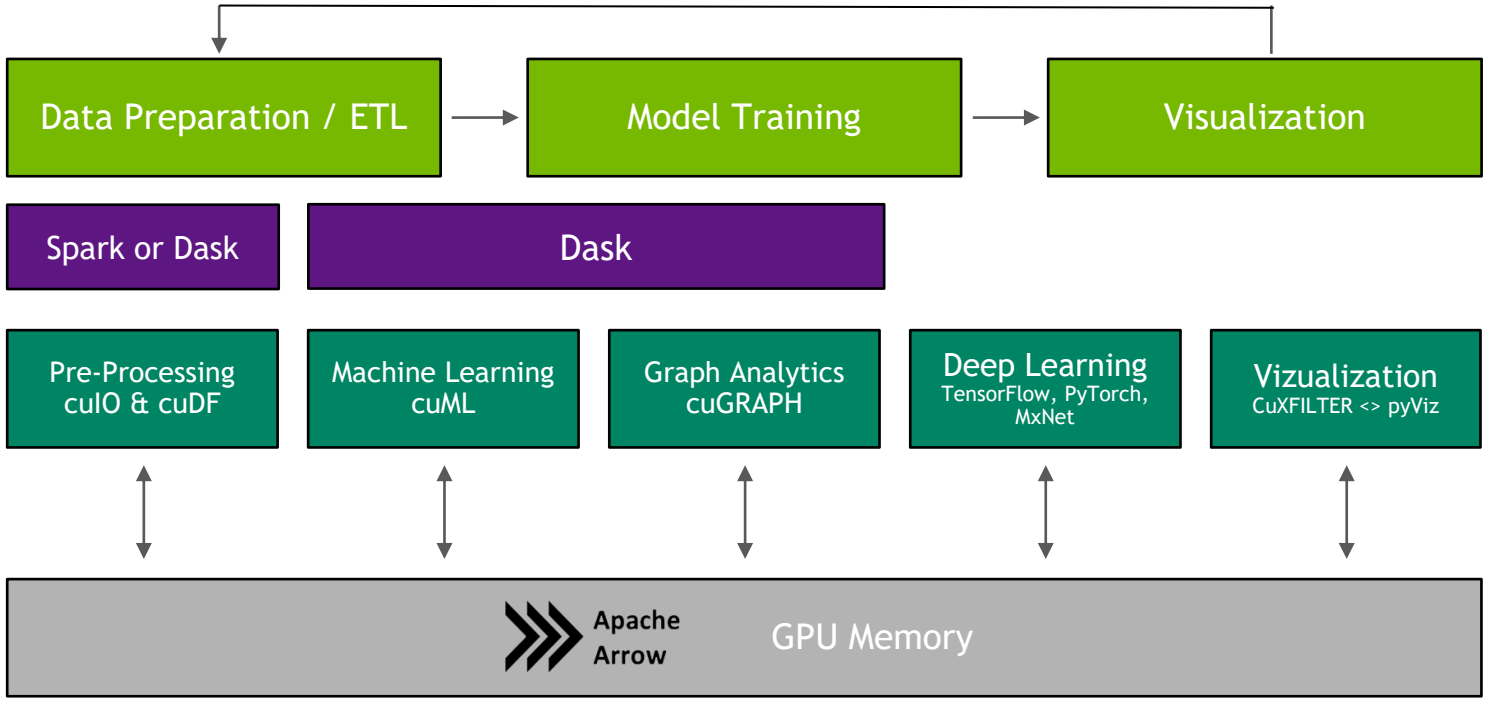
RAPIDS ACCELERATES POPULAR DATA SCIENCE TOOLS

DELIVERING ENTERPRISE-GRADE DATA SCIENCE SOLUTIONS

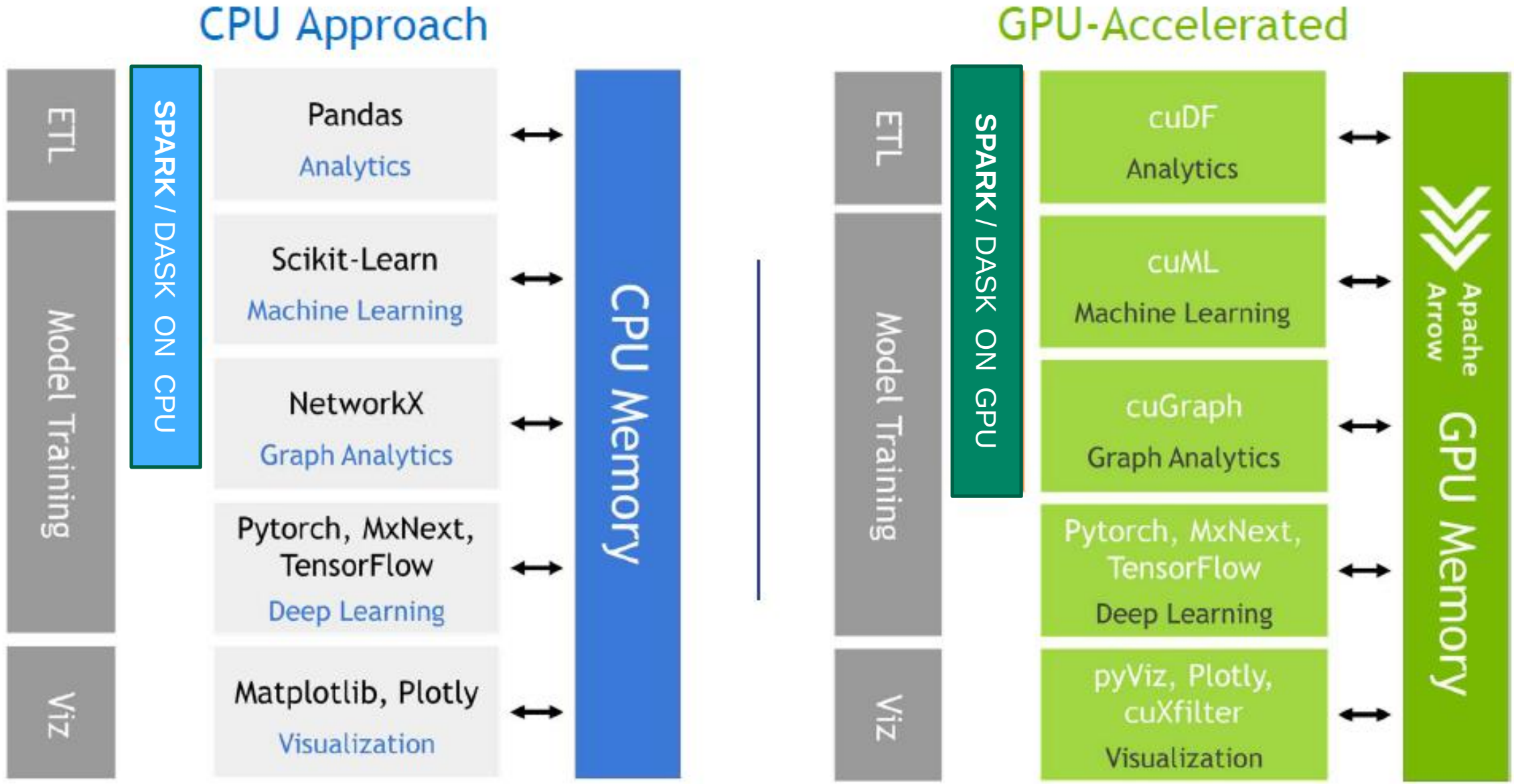
The RAPIDS suite of open source software libraries gives you the freedom to execute end-to-end data science and analytics pipelines entirely on GPUs.

RAPIDS utilizes **NVIDIA CUDA** primitives for low-level compute optimization and exposes GPU parallelism and high-bandwidth memory speed through user-friendly interfaces like pandas, scikit-learn, Apache Spark or Dask.

With Spark or Dask, RAPIDS can scale out to multi-node, multi-GPU cluster to power through big data processes.

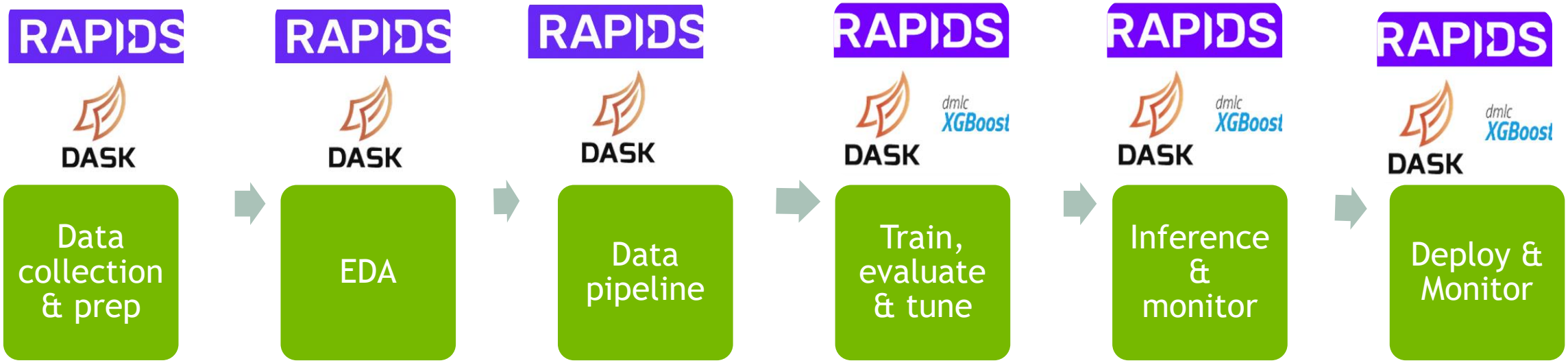


RAPIDS accelerates enterprise Data Science with NVIDIA GPUs


CPU Approach
ETL
Model Training
Viz
SPARK / DASK ON CPU
Pandas
Analytics
Scikit-Learn
Machine Learning
NetworkX
Graph Analytics
Pytorch, MxNext, TensorFlow
Deep Learning
Matplotlib, Plotly
Visualization
CPU Memory
GPU-Accelerated
ETL
Model Training
Viz
SPARK / DASK ON GPU
cuDF
Analytics
cuML
Machine Learning
cuGraph
Graph Analytics
Pytorch, MxNext, TensorFlow
Deep Learning
pyViz, Plotly, cuXfilter
Visualization
Apache Arrow
GPU Memory

ACCELERATING END-TO-END WORKFLOW

Accelerating every step of ML model lifecycle with RAPIDS and Dask

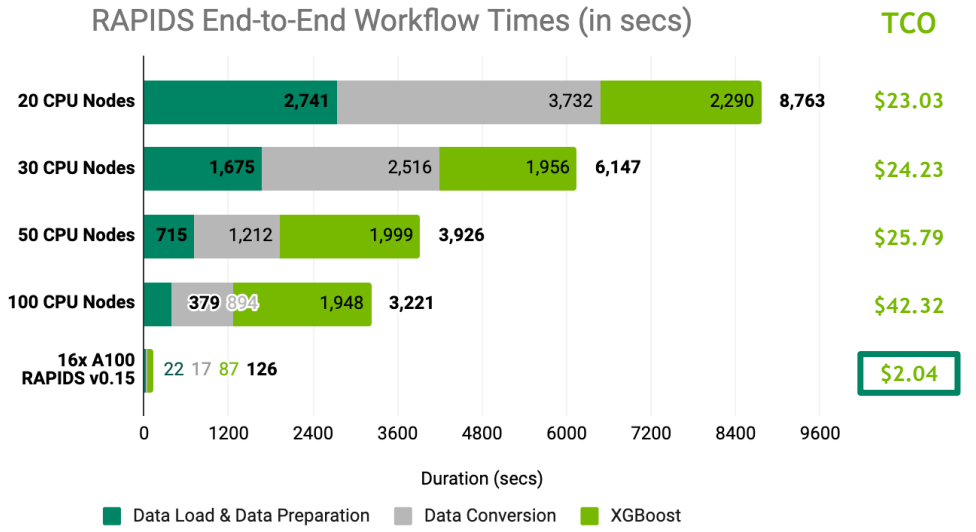


LIGHTNING-FAST END-TO-END PERFORMANCE

REDUCING DATA SCIENCE PROCESSES FROM HOURS TO SECONDS

RAPIDS delivers massive speed-ups across the end-to-end data science lifecycle. Conducting benchmarks in a commercial cloud environment, we're able to get incredible performance running a common ML model training pipeline.

Between loading and cleansing data, engineering features, and training a classifier using a 200GB CSV dataset, a RAPIDS-based pipeline completed these operations in ***just over two minutes***. The same process takes two and half hours on a similar CPU-configuration.



16

A100s Provide More Power than 100 CPU Nodes*

70x

Faster Performance than Similar CPU Configuration

20x

More Cost-Effective than Similar CPU Configuration

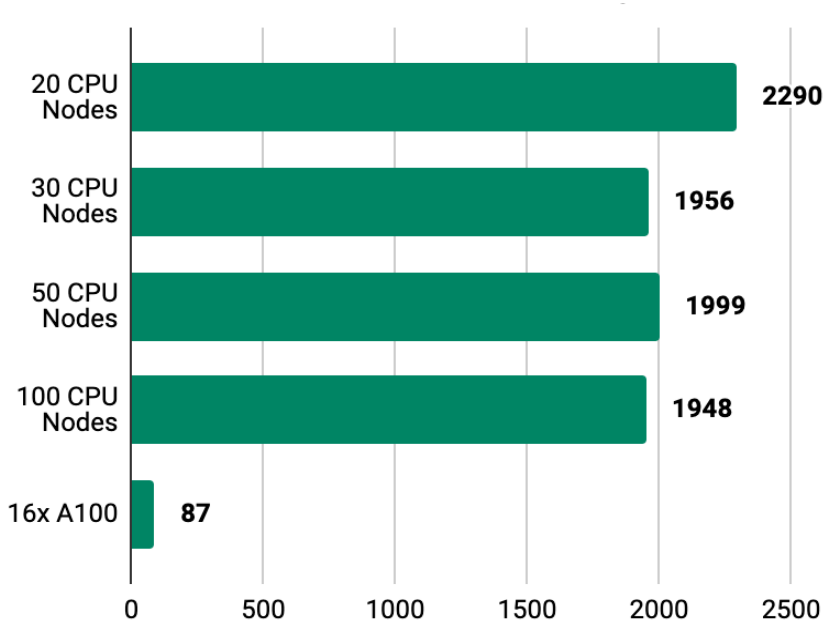
*CPU approximate to n1-highmem-8 (8 vCPUs, 52GB memory) on Google Cloud Platform.

ACCELERATED MACHINE LEARNING

GPU-POWER WITH THE FEEL OF SCIKIT-LEARN

cuML is a suite of libraries that implement machine learning algorithms and mathematical primitives functions that share compatible APIs with other RAPIDS projects.

cuML enables data scientists, researchers, and software engineers to run traditional tabular ML tasks on GPUs without going into the details of CUDA programming. In most cases, cuML's Python API matches the API from scikit-learn.

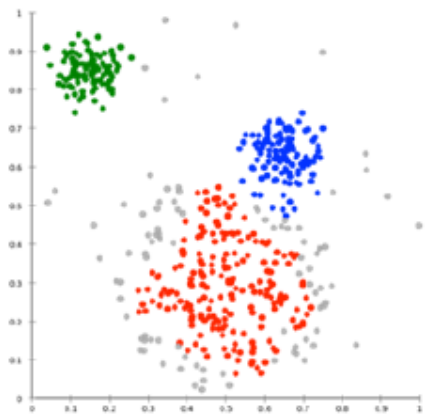


XGBoost Training Improvements

GPU: NVIDIA Ampere DGX A100s
CPU: 61GB memory, 8 vCPUs, 64-bit platform

Algorithms

GPU-accelerated Scikit-Learn



Cross Validation

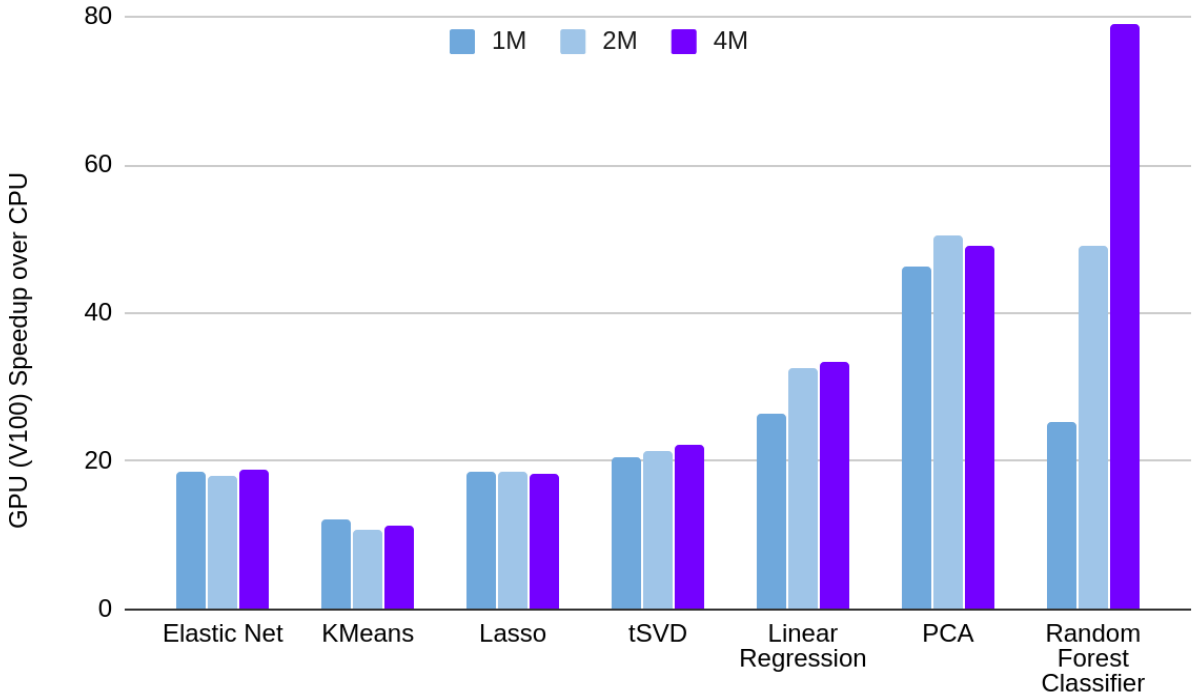
Hyper-parameter Tuning

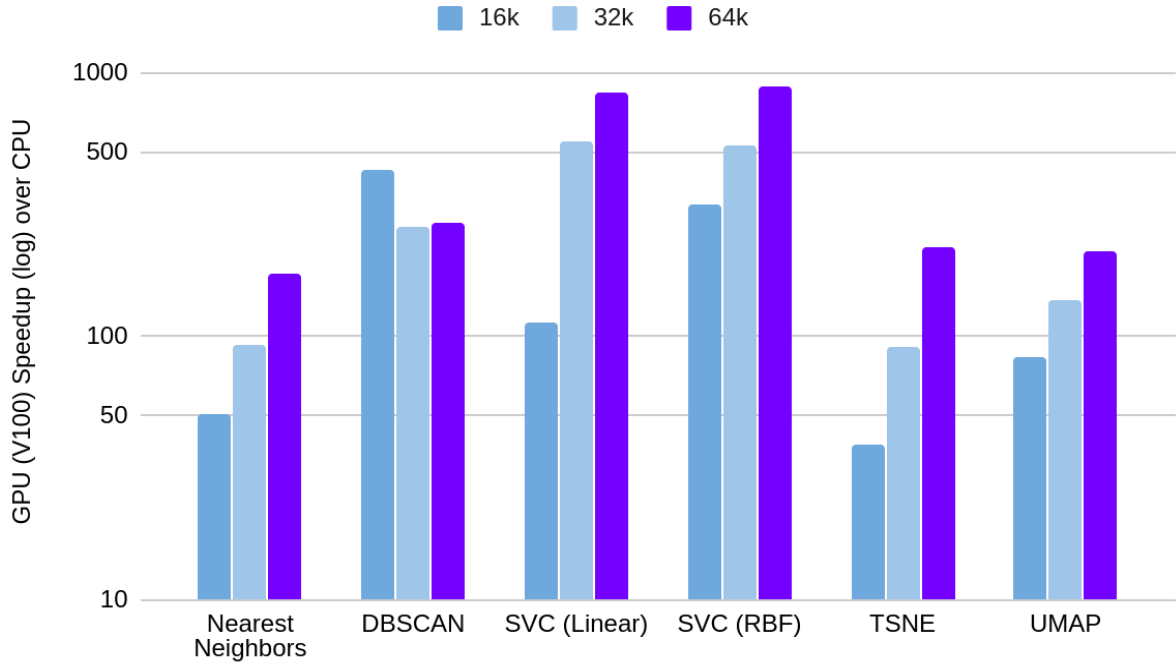
More to come!

Classification / Regression	Decision Trees / **Random Forests** Linear / Lasso / Ridge / LARS / ElasticNet Regression Logistic Regression K-Nearest Neighbors (**exact or approximate**) Support Vector Machine Classification and Regression Naive Bayes
Inference	Random Forest / GBDT Inference (FIL)
Preprocessing	Text vectorization (TF-IDF / Count) Target Encoding Cross-validation / splitting
Clustering Decomposition & Dimensionality Reduction	K-Means **DBSCAN** Spectral Clustering Principal Components (including iPCA) Singular Value Decomposition UMAP Spectral Embedding T-SNE
Time Series	Holt-Winters Seasonal ARIMA / Auto ARIMA

BENCHMARKS: SINGLE-GPU CUML VS SCIKIT-LEARN

20-1000x Faster than Scikit-Learn





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

USE WITH MINIMAL CODE CHANGES

GPU-Acceleration with the same XGBoost Usage

BEFORE

```
import xgboost as xgb

params = {'max_depth': 3,
          'learning_rate': 0.1}


dtrain = xgb.DMatrix(X, y)
bst = xgb.train(params, dtrain)
```

AFTER

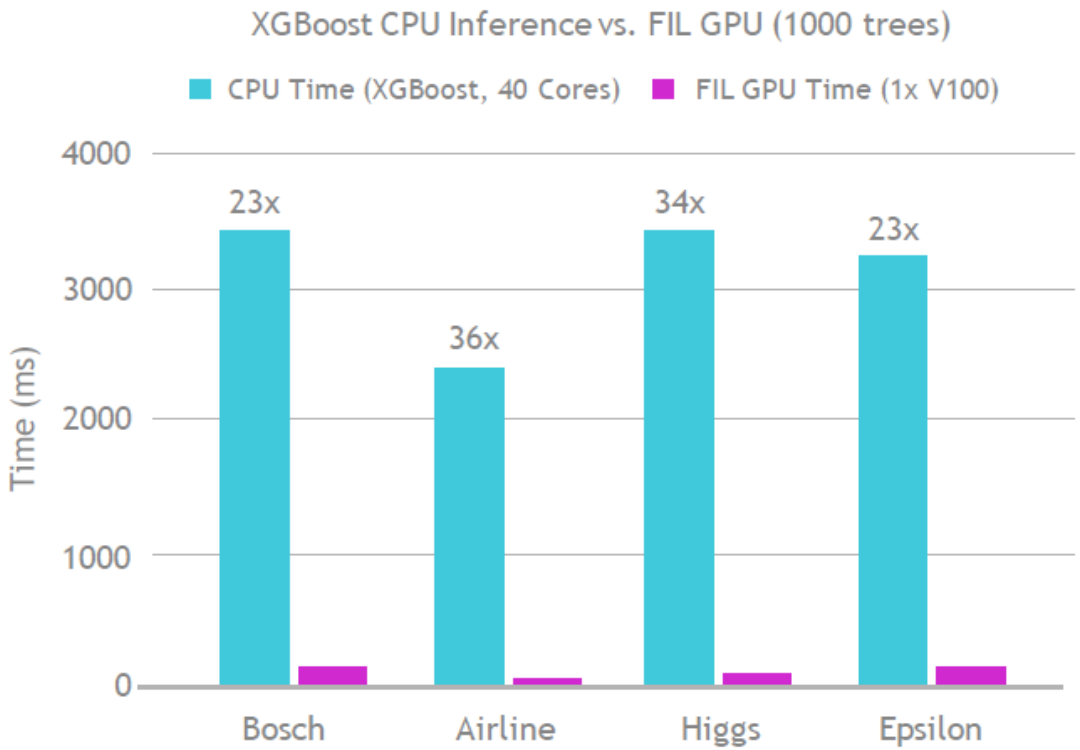
```
import xgboost as xgb

params = {'tree_method': 'gpu_hist',
          'max_depth': 3,
          'learning_rate': 0.1}

dtrain = xgb.DMatrix(X, y)
bst = xgb.train(params, dtrain)
```

Forest Inference

Taking Models From Training to Production

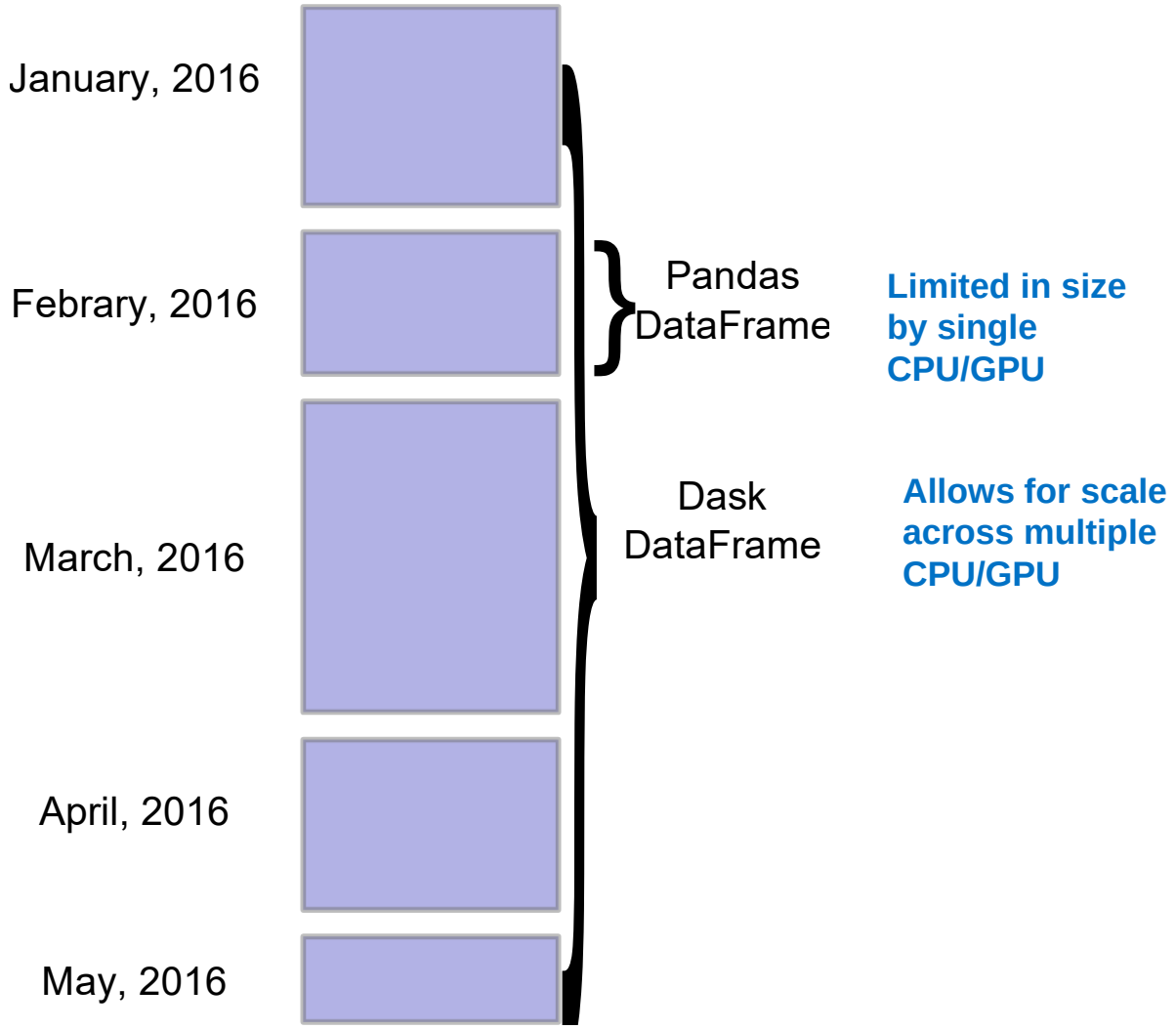
cuML's Forest Inference Library accelerates prediction (inference) for random forests and boosted decision trees:

- 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



WHAT IS DASK?

- A distributed DataFrame for CPU/GPU allowing scale-out memory
  - Bigger dataframe (rows, terabytes)
  - Keeps Data Scientist in Python, favored tool
- A task-based library for parallel scheduling and execution
- A decompose large DataFrames/ Series (pandas/ cuDF) into a collection of DataFrames/ Series
- Schedules and executes the optimized task graph on one or more processes/ threads

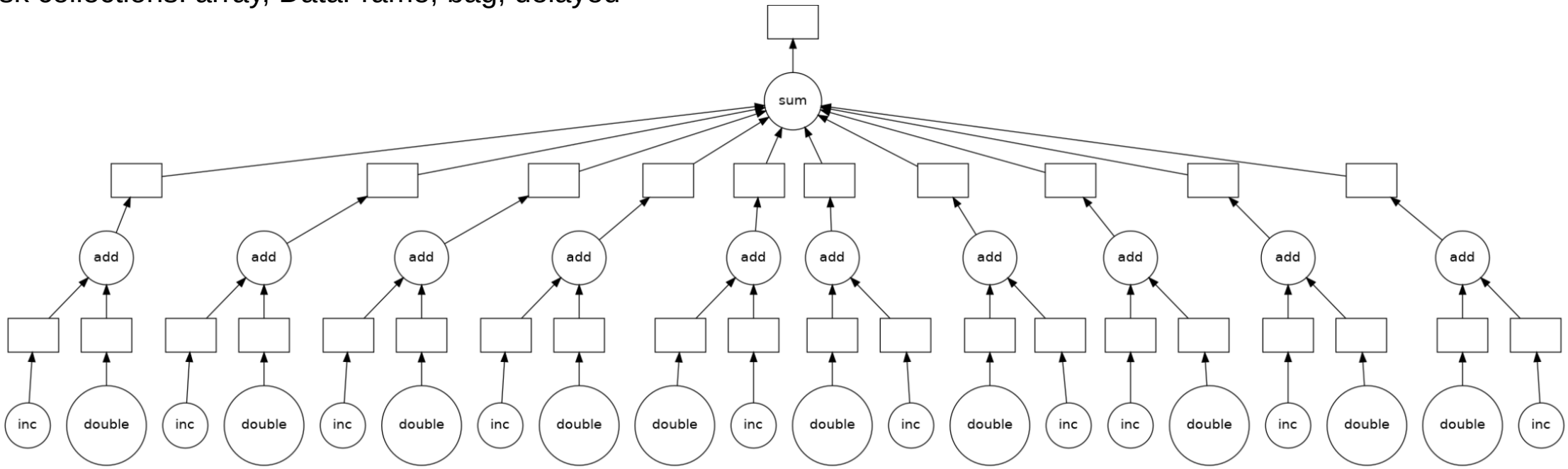


https://github.com/rjzamora/notebooks/tree/master/nvtabular_dask_demo

DASK TASK GRAPH

Operations on Dask collections build task graphs and execution is differed/lazy

Dask collections: array, DataFrame, bag, delayed



Execution graph of a simple python function calls

Key:

- Circles are function calls
- Rectangles represent data/results

ACCELERATING ETL WITH DASK

End-end-end ETL pipeline using GPUs

```
def ml_data_pipeline(X_train, X_test, y_train, y_test, gpu=True):
    """Process data for ML model training and evaluation.

    Parameters:
        Dataframes (Pandas)
    Returns:
        GPU and CPU Dask DataFrame

    """
    if gpu:
        print('Processing data for distributed GPUs')

        train_test_dfs = [X_train, X_test, y_train, y_test]
        X_train_gdf, X_test_gdf, y_train_gdf, y_test_gdf = [cudf.from_pandas(df) for df in train_test_dfs]


        X_train_gdf, X_test_gdf = impute_missing_data(X_train_gdf, X_test_gdf, gpu)


        X_train_gdf, X_test_gdf = normalize_feats(X_train_gdf, X_test_gdf, gpu)


        X_train_gdf, X_test_gdf = one_hot_encd(X_train_gdf, X_test_gdf, gpu)


        X_train_gdf, X_test_gdf = target_encod_high_card(X_train_gdf, X_test_gdf, y_train_gdf, gpu)


        X_train_gdf, X_test_gdf = normalize_feats(X_train_gdf, X_test_gdf, gpu)


        return X_train_gdf, X_test_gdf, y_train_gdf, y_test_gdf
```

Data preprocessing and feature engineering pipeline using GPUs. Follow the steps to perform ETL at scale.

- Convert Pandas DF to RAPIDS cuDF
- Impute missing values/Nans
- Normalize numerical features
- One-hot encode categorical features
- Target encode high cardinality features
- Normalize high cardinality features

ACCELERATING ETL WITH DASK

Distributing data across GPUs

```
def distribute(X, y, device, persist=False):
    """Partition and distribute data across workers"""

    if device=='gpu':
        n_partitions = n_workers

        X_dask = dask_cudf.from_cudf(X, npartitions=n_partitions)

        y_dask = dask_cudf.from_cudf(y, npartitions=n_partitions)

        if True:
            X_dask, y_dask = dask_utils.persist_across_workers(
                client, [X_dask, y_dask], workers=workers)

    if device=='cpu':
        n_partitions = n_wrks_cpu

        X_dask = dd.from_pandas(X, npartitions=n_partitions)

        y_dask = dd.from_pandas(y, npartitions=n_partitions)

        if True:
            X_dask, y_dask = dask_utils.persist_across_workers(
                client_cpu, [X_dask, y_dask], workers=workers_cpu)

    return X_dask, y_dask
```

Performing parallel across GPUs and CPUs. Using Dask and RAPIDS. Below are steps

- Select GPU for your analysis and call `dask_cudf`
- Partition data across GPUs
- Persist data to disc, when possible
- For CPUs, convert data to Pandas DF
- Partition data as per available # CPUs and persist to disc
- Return Dask dataframe to your ML pipeline

DISTRIBUTED MODEL TRAINING AND EVALUATION

Accelerate model training using RAPIDS and GPU Dask with ease

```
#---
# SKLearn multi-CPU training: Using n_jobs=-1
#---

from sklearn.ensemble import RandomForestClassifier as sklRF
max_depth = 12
n_bins = 16
n_trees = 1000

# Reshape data
X, y = X_train_cpu, y_train_cpu.values.ravel()

skl_model = sklRF(
                max_depth=max_depth,
                n_estimators=n_trees,
                n_jobs=-1)

skl_model.fit(X, y)


#---
# SKLearn multi-CPU training: Using joblib.parallel_backend
#---
clsf_dask = sklRF(max_depth=max_depth, n_estimators=n_trees, n_jobs=-1)

# Use joblib to use cluster to train a model
with joblib.parallel_backend('dask', scatter=[X, y]):
    clsf_dask.fit(X, y)
```

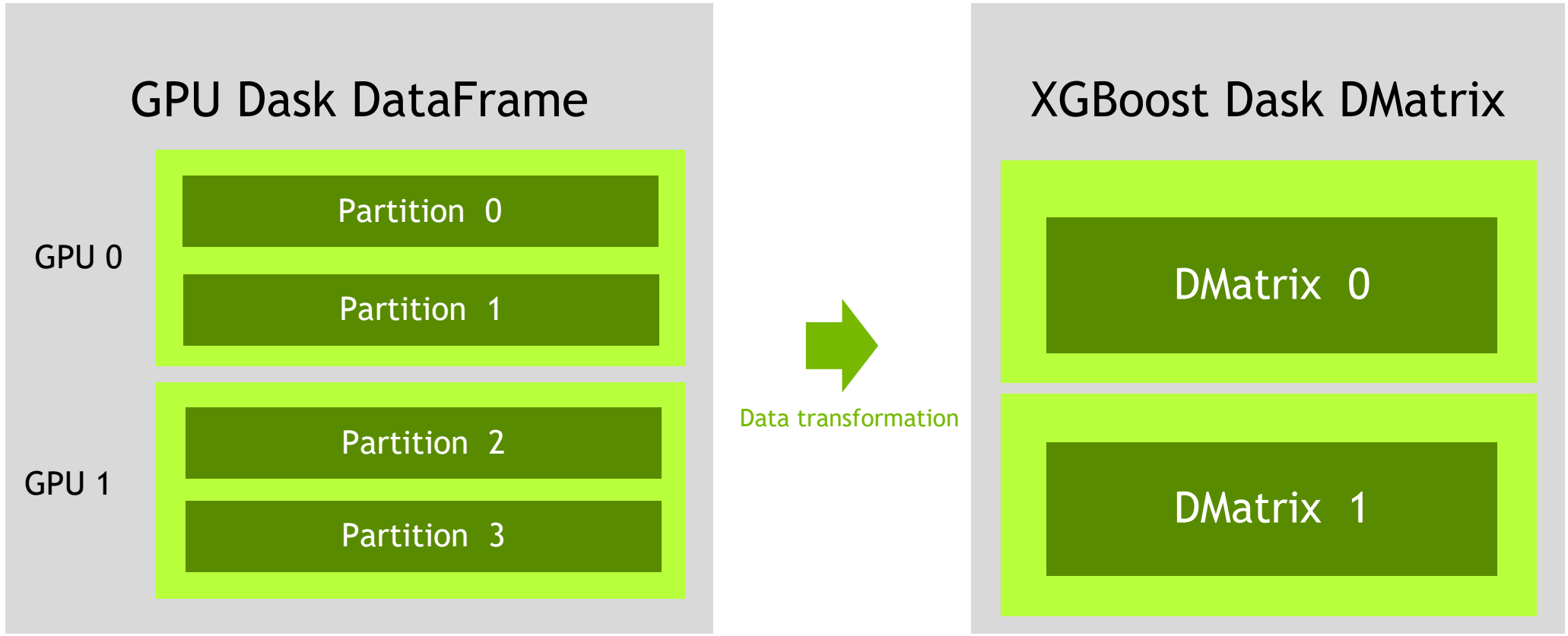
```
#----
# Distributed model training with Dask cuDF
#----
from cuml.dask.ensemble import RandomForestClassifier as cumlDaskRF

max_depth = 12
n_bins = 16
n_trees = 1000

n_streams = 8

X_dask_cudf = X_train_dask_cudf.astype(np.float32)
y_dask_cudf = y_train_dask_cudf.astype('int32')

cuml_model = cumlDaskRF(max_depth=max_depth,
                        n_estimators=n_trees,
                        n_bins=n_bins,
                        n_streams=n_streams)

cuml_model.fit(X_dask_cudf,
               y_train_dask_cudf,
               convert_dtype=False)

wait(cuml_model.rfs)

cuml_y_pred = cuml_model.predict(X_test_dask_cudf)


cuml_y_pred = cupy.asarray(cuml_y_pred.compute())
```

Dask data is already distributed

RandomForestClassifier with Multi-GPU and -CPU

DISTRIBUTED XGBOOST WITH DASK

Dask and XGBoost can work together to train gradient boosted trees in parallel



Dask and XGBoost can work together to train gradient boosted trees in parallel

```
import xgboost as xgb
import dask

dmatrix_train = xgb.dask.DaskDMatrix(client, X_train_dask_cudf, y_train_dask_cudf)
dmatrix_test = xgb.dask.DaskDMatrix(client, X_test_dask_cudf, y_test_dask_cudf)

params = {
      'objective': 'binary:logistic',
      'eval_metric': 'auc',
      'n_estimators': '500',
      'max_depth': '4',
      'n_jobs': '4',
      'tree_method': 'gpu_hist',    # Use GPU training algo
      'scale_pos_weight' : 10,
}
xgb_clasf = xgb.dask.train(client,
                           params,
                           dmatrix_train,
                           num_boost_round=100,
                           evals=[(dmatrix_train, 'train')],
                          )

model_xgb = xgb_clasf['booster']

history = xgb_clasf['history']

model_xgb.set_param({'predictor': 'gpu_predictor'})

y_predict = xgb.dask.predict(client, xgb_clasf['booster'], dmatrix_test)

y_test_c = y_test_dask_cudf.astype('int32')

y_test_c, y_predict = dask.compute(y_test_c, y_predict)

print("RAPIDS cuML XGB model AUC: {:.2%}".format(cuml.metrics.roc_auc_score(y_test_c, y_predict)))
```

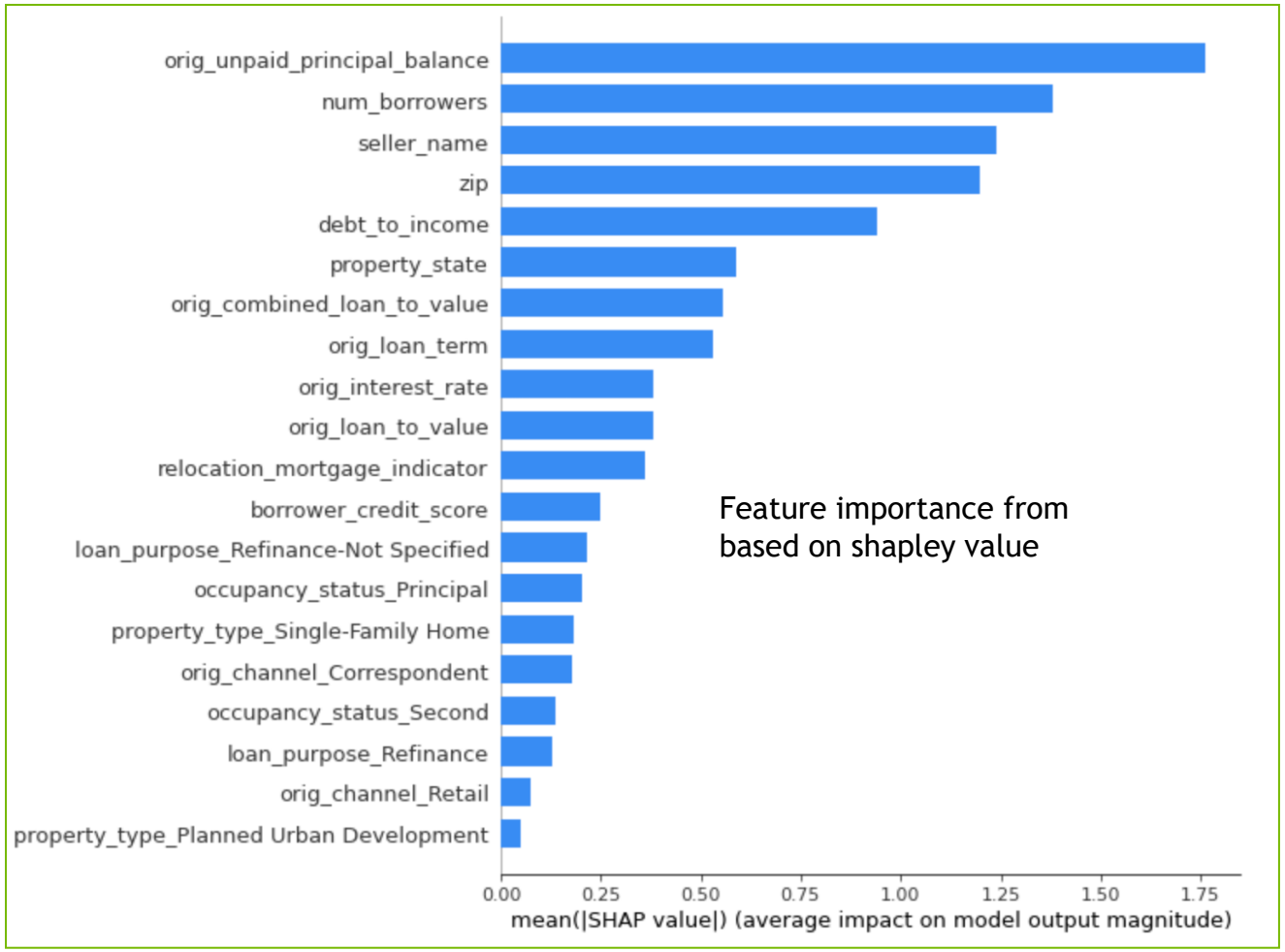
Steps to scale XGBoost with Dask

- Import xgb and dask
- Convert dask_cudf to DaskDMatrix
- Set xgb params
- Set GPU for training algorithm
- Pass client to xgb for distributed algorithm
- Get booster is the trained model, history is the eval metric collected during training
- Set gpu_predictor to use GPU for inference
- Call xgb.dask. predict to make prediction
- Call compute to get prediction result

MODEL EXPLAINABLITY

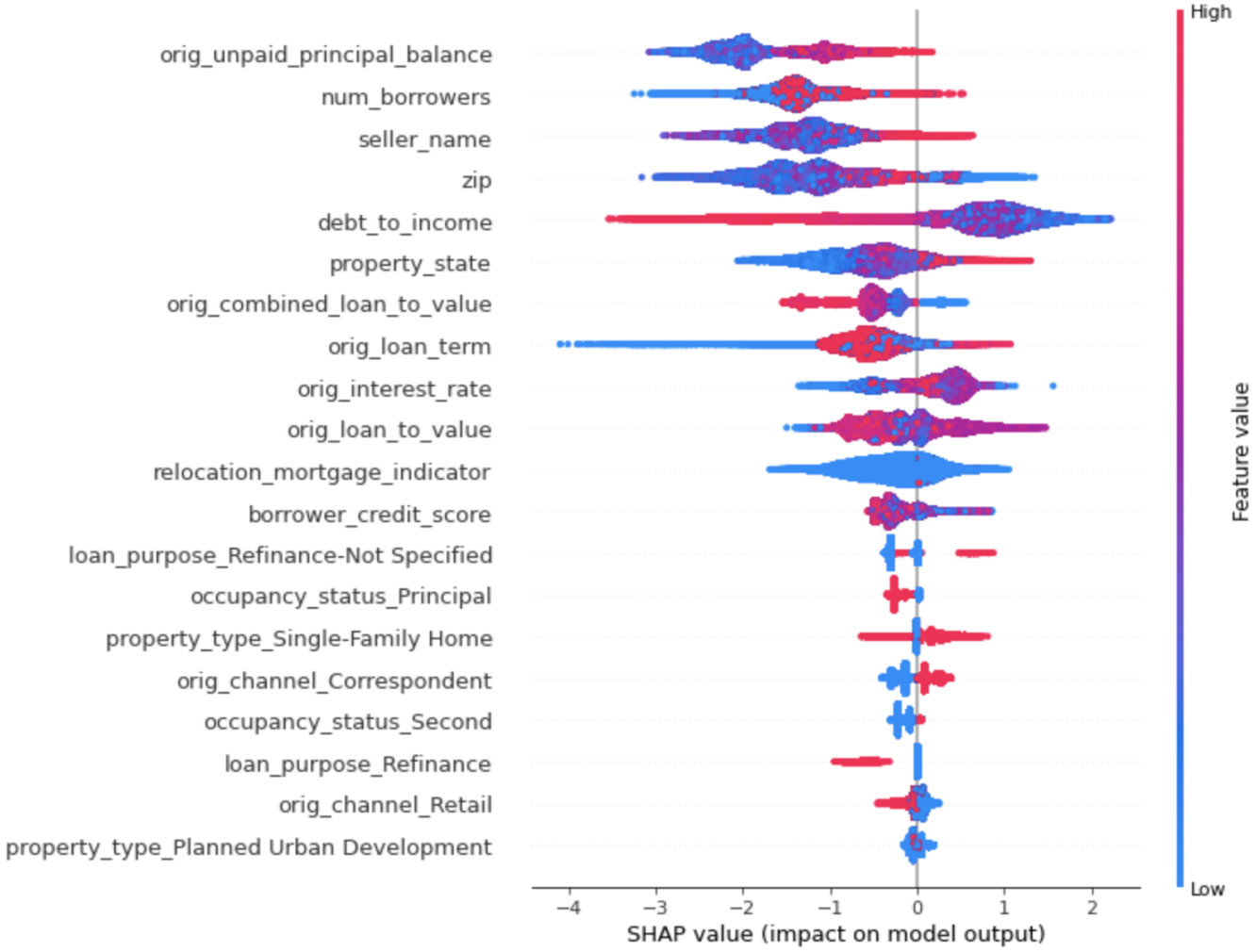
GPUTreeSHAP for XGBoost

- 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)



MODEL EXPLAINABLITY

GPU-Accelerated SHAP values



INFERENCE AT SCALE

RAPIDS Forest Inference Library (FIL)

- Forest Inference Library is a GPU accelerated inference library for forest-based models.
    - It is based on C++ and CUDA core library.
- Accelerates forest-based models: Random Forest and Gradient Boosted models (XGBoost, LightGBM)
- FIL avoids common bottlenecks in streaming (low-latency) and batch (high-throughput) predication jobs
- Using FIL, a single V100 GPU can deliver up to **35x** more inference throughput than a CPU-only node with 40 cores

INFERENCE AT SCALE

Best practices

Performance factors and optimizing FIL

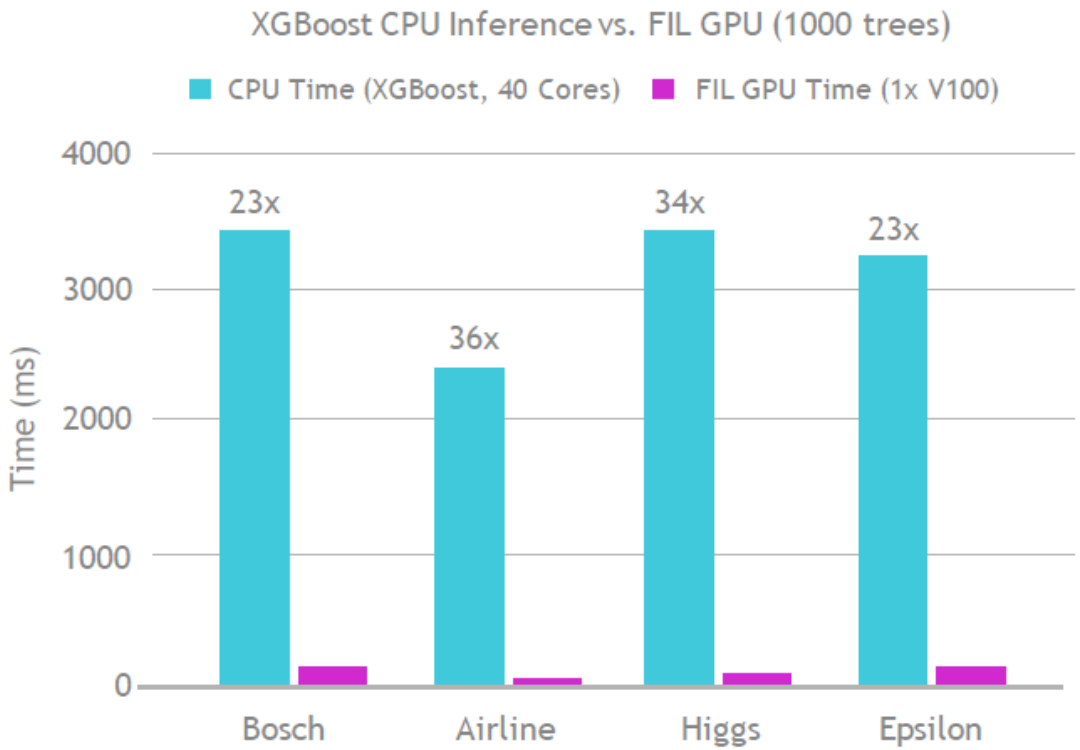
- Number of trees: runtime is roughly linear in number of trees
- Storage format of input data
    - row-major GPU arrays are fastest,
    - column-major arrays will need potentially-expensive conversion
- Batch size:
    - larger batches are slower but cost less on a per-row basis, as startup costs are amortized
- Tree depth: the deeper the tree the slower the prediction pipeline

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

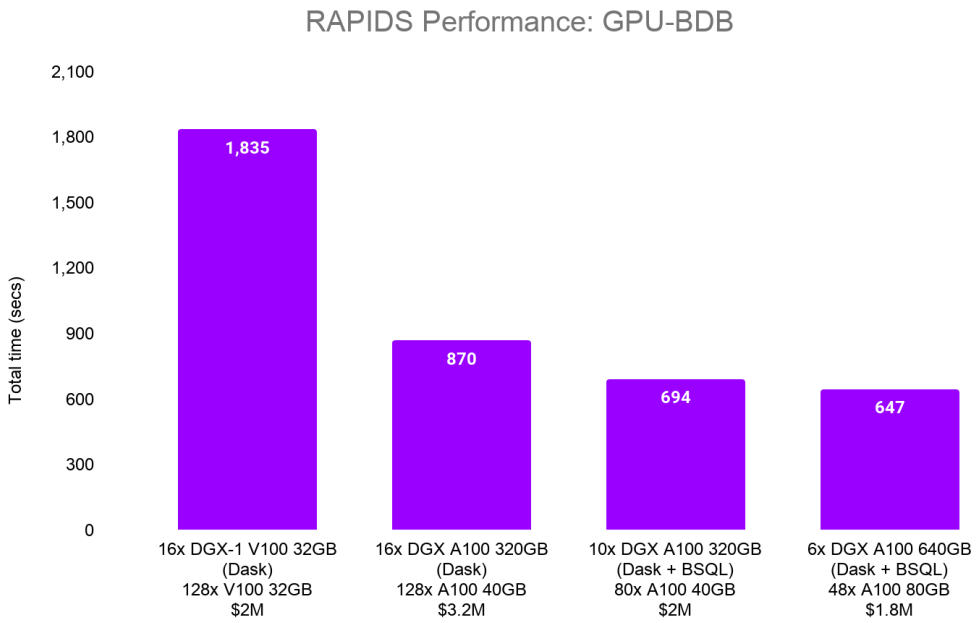


LIGHTNING-FAST PERFORMANCE ON REAL-WORLD USE CASES

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.

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.



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

CUXFILTER

(coo-cross-filter)

- Efficient GPU-backed in-browser visualization engine
- Built on a host of pyViz ecosystem tools
- Bokeh and Datashader charts
- Multiple charts with cross-filtering widgets

Step by Step Learning Path

Master Accelerated Data Science with RAPIDS

Step 1 – Go through RAPIDS.AI website for details on RAPIDS suite of libraries

Step 2 – Hands on with Google Colab OR Download NGC Catalog Rapids Container Image OR use Conda to install Rapids

Step 3 – Run RAPIDS Notebooks and Blogs example after installing RAPIDS

Step 4 – Get help from RAPIDS Documentation for any syntax / issue help

Step 5 – Go to RAPIDS cuML GitHub Repo for any bugs /issue

Bonus - Try out RAPIDS Spark

The rapids advantage

How RAPIDS delivers Data Science value





Maximized Productivity	Top Model Accuracy	Lowest TCO	Low Learning Curve	Targeted Acceleration
Massive speedups using RAPIDS with XGBoost	Huge savings and low error rates	Best value for money, low infrastructure costs	Easy integration with familiar APIs	Acceleration of most commonly-used programing languages
Example: 215X speedups at Oak Ridge National Labs	Example: $1B potential savings and 4% error rate reduction at Global Retail Giant	Example: $1.5B infrastructure cost savings at Streaming Media Company	Example: Easily replaces popularly-used libraries such as like Pandas and scikit-learn	Example: RAPIDS accelerates Python and SQL workflows which have increasing market share

DOWNLOAD AND DEPLOY

Source available on Github | Container available on NGC and Dockerhub | PIP available at a later date

GitHub

NGC

docker

CONDA

python Package Index

Source code, libraries, packages

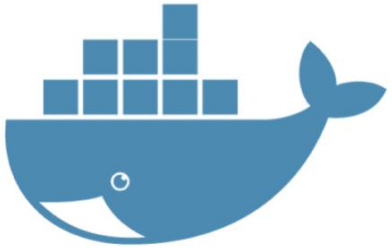
On-premises

Cloud

Rapids

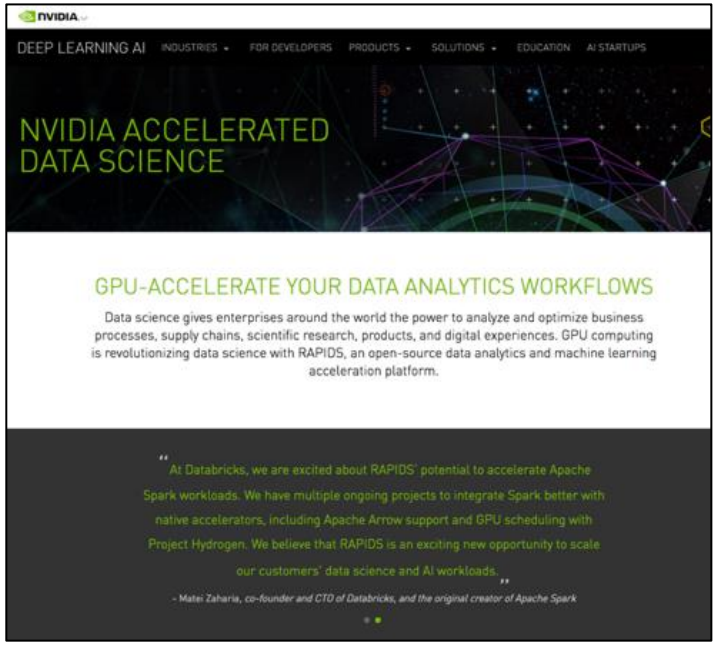
How do I get the software?

- https://github.com/rapidsai
- https://anaconda.org/rapidsai/



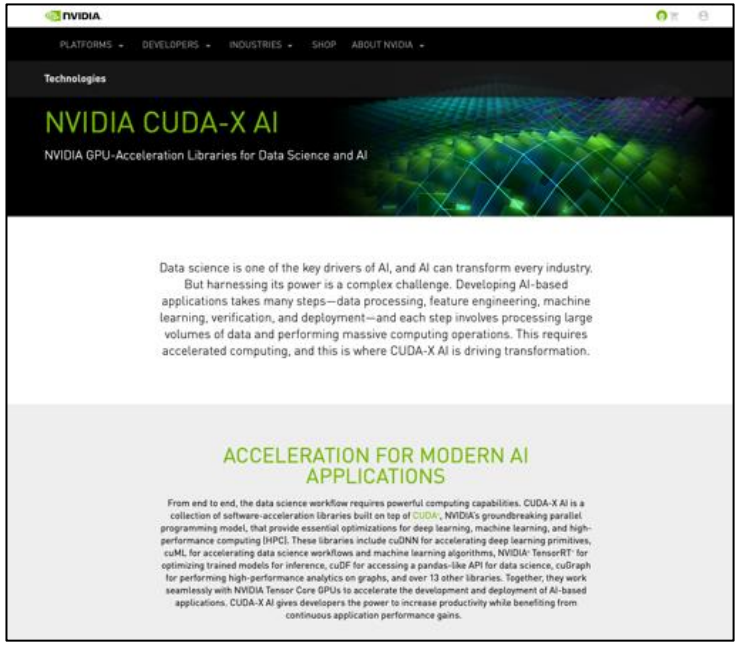
- https://ngc.nvidia.com/registry/nvidia-rapidsai-rapidsai
- https://hub.docker.com/r/rapidsai/rapidsai/

FOR MORE INFORMATION



nvidia.com/datascience

rapids.ai



nvidia.com/en-us/technologies/cuda-x/