DATA SCIENCE LEVERAGING GPU'S

BILL VEENHUIS, PRINCIPAL ARCHITECT

AGENDA

GPU – more than an accelerator

RAPIDS – ETL, operationalized the data

CuPY – for the love of Pandas, NumPy & SciPy

Merlin – Recommenders

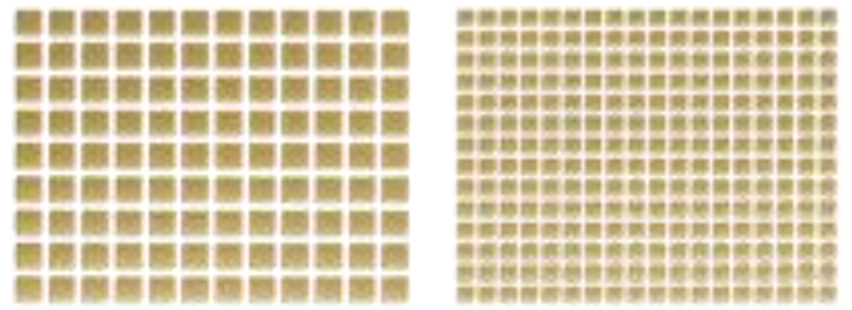
GPUS

SMALL CHANGES, BIG SPEED-UP

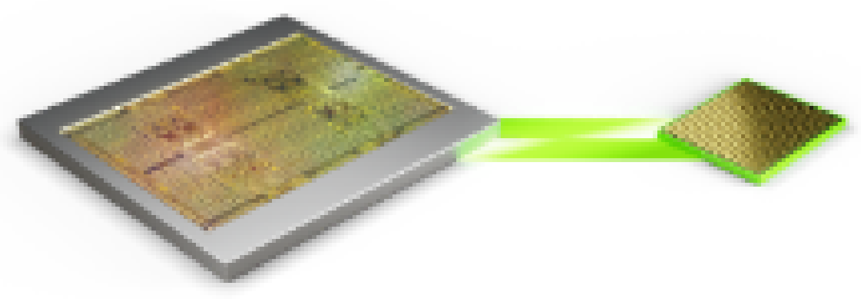
Application Code
Compute-Intensive Functions
5% of Code
GPU
Rest of Sequential
CPU Code
CPU
+

NVIDIA A100 80GB

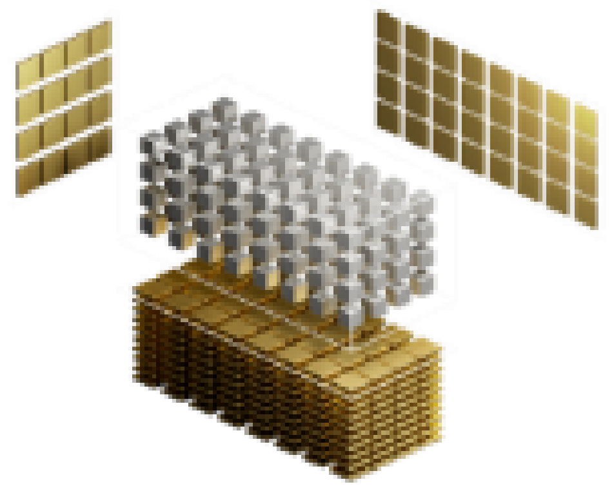
Supercharging The World’s Highest Performing AI Supercomputing GPU



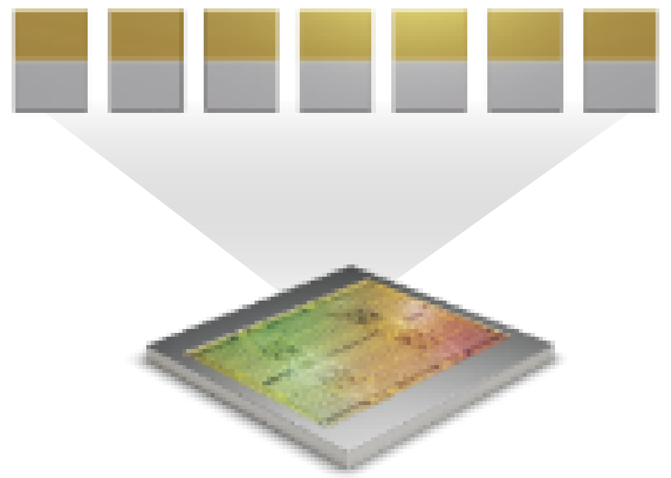
80GB HBM2e
For largest datasets
and models



2TB/s +
World’s highest memory bandwidth
to feed the world’s fastest GPU

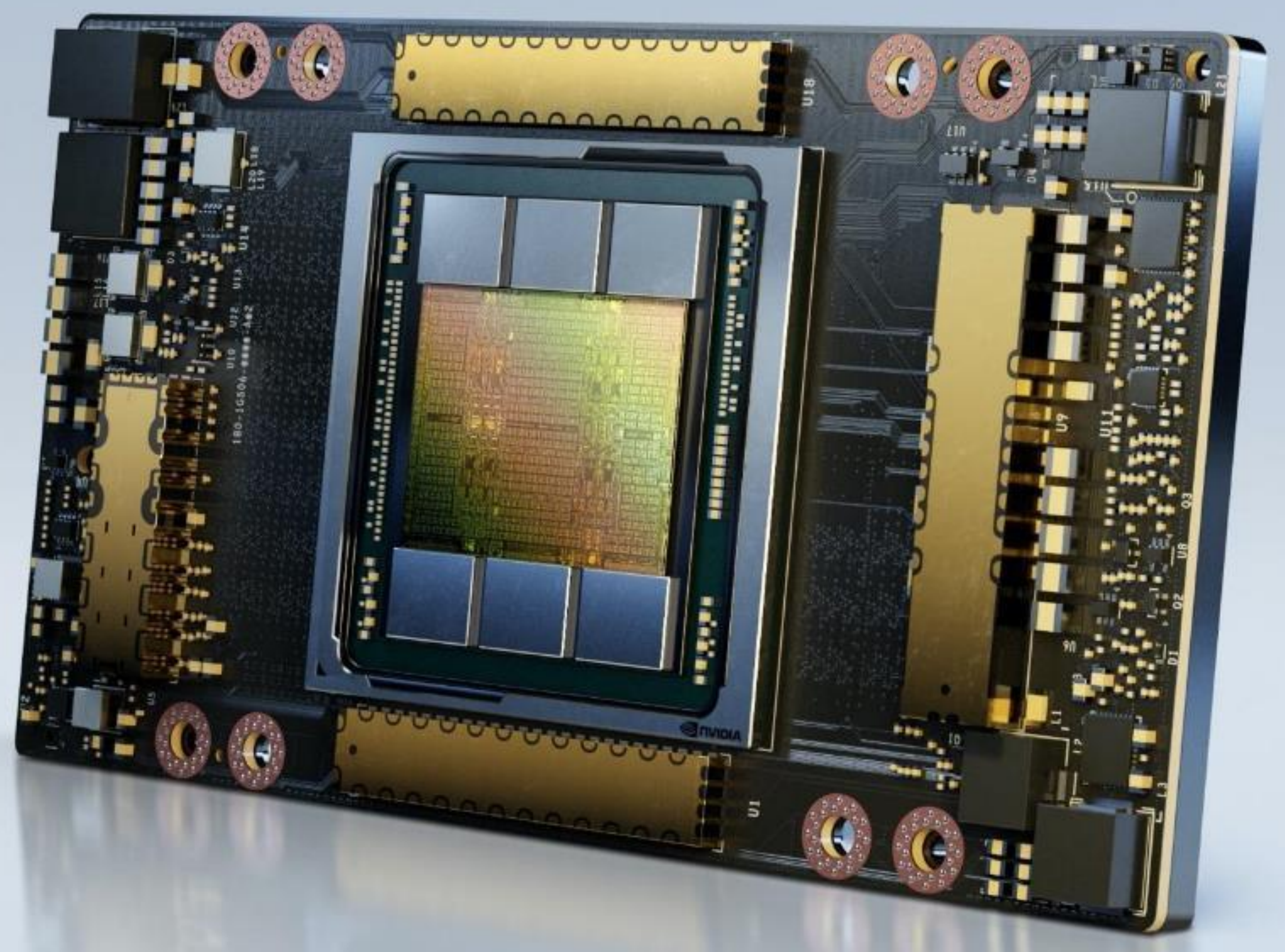


3rd Gen Tensor Core



Multi-Instance GPU

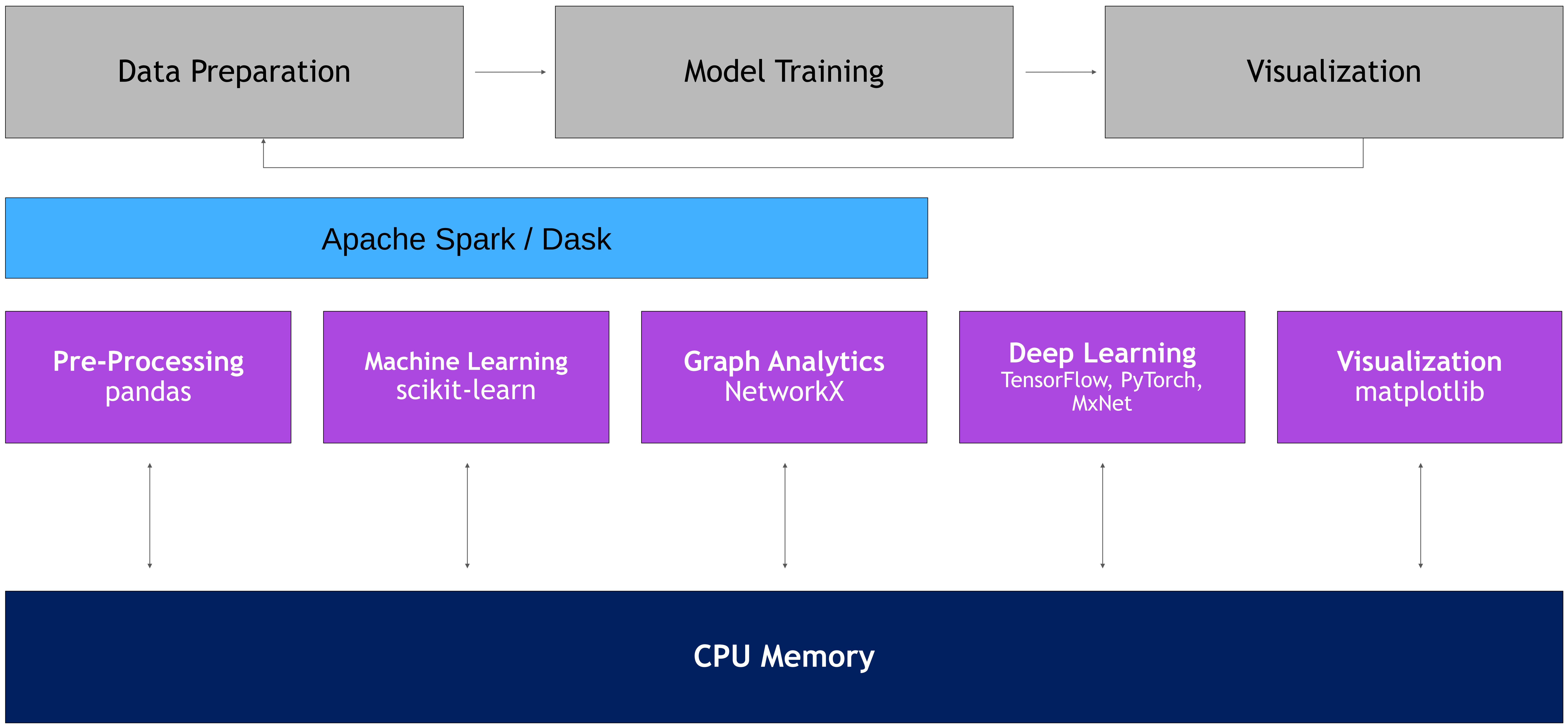
3rd Gen NVLink



RAPIDS

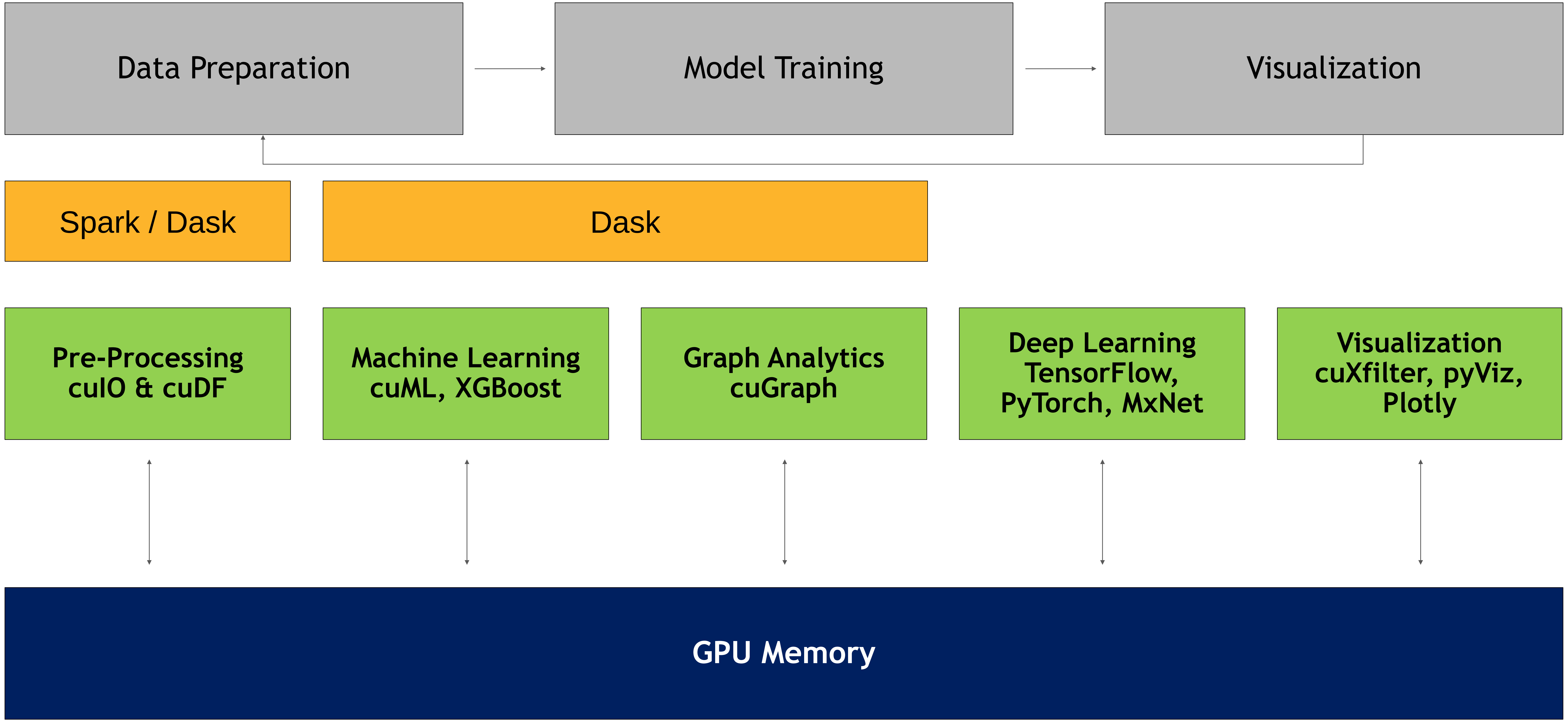
OPEN SOURCE SOFTWARE HAS DEMOCRATIZED DATA SCIENCE

Highly Accessible, Easy to Use Tools Abstract Complexity



ACCELERATED DATA SCIENCE WITH RAPIDS

Powering Popular Data Science Ecosystems with NVIDIA GPUs



Data Preparation
Model Training
Visualization
Spark / Dask
Dask
Pre-Processing cuIO & cuDF
Machine Learning cuML, XGBoost
Graph Analytics cuGraph
Deep Learning TensorFlow, PyTorch, MxNet
Visualization cuXfilter, pyViz, Plotly
GPU Memory

MINOR CODE CHANGES FOR MAJOR BENEFITS

Abstracting Accelerated Compute through Familiar Interfaces

CPU

pandas

```
>>> import pandas as pd
>>> df = pd.read_csv("filepath")
```

GPU

cuDF

```
>>> import cudf
>>> df = cudf.read_csv("filepath")
```

Average Speed-Ups: *150x*

CPU

CPU Spark

```
spark.sql("""
select
   order
   count(*) as order_count
from
   orders""")
```

GPU

GPU Spark

```
spark.conf.set("spark.plugins
","com.nvidia.spark.SQLPlugin")

spark.sql("""
select
   order
   count(*) as order_count
from
   orders""")
```

Average Speed-Ups: *10x*

CPU

scikit-learn

```
>>> from sklearn.ensemble
import
RandomForestClassifier
>>> clf =
RandomForestClassifier()
>>> clf.fit(x, y)
```

GPU

cuML

```
>>> from cuml.ensemble import
RandomForestClassifier
>>> cuclf =
RandomForestClassifier()
>>> cuclf.fit(x, y)
```

Average Speed-Ups: *50x*

CPU

NetworkX

```
>>> import networkx as nx
>>> page_rank =
nx.pagerank(graph)
```

GPU

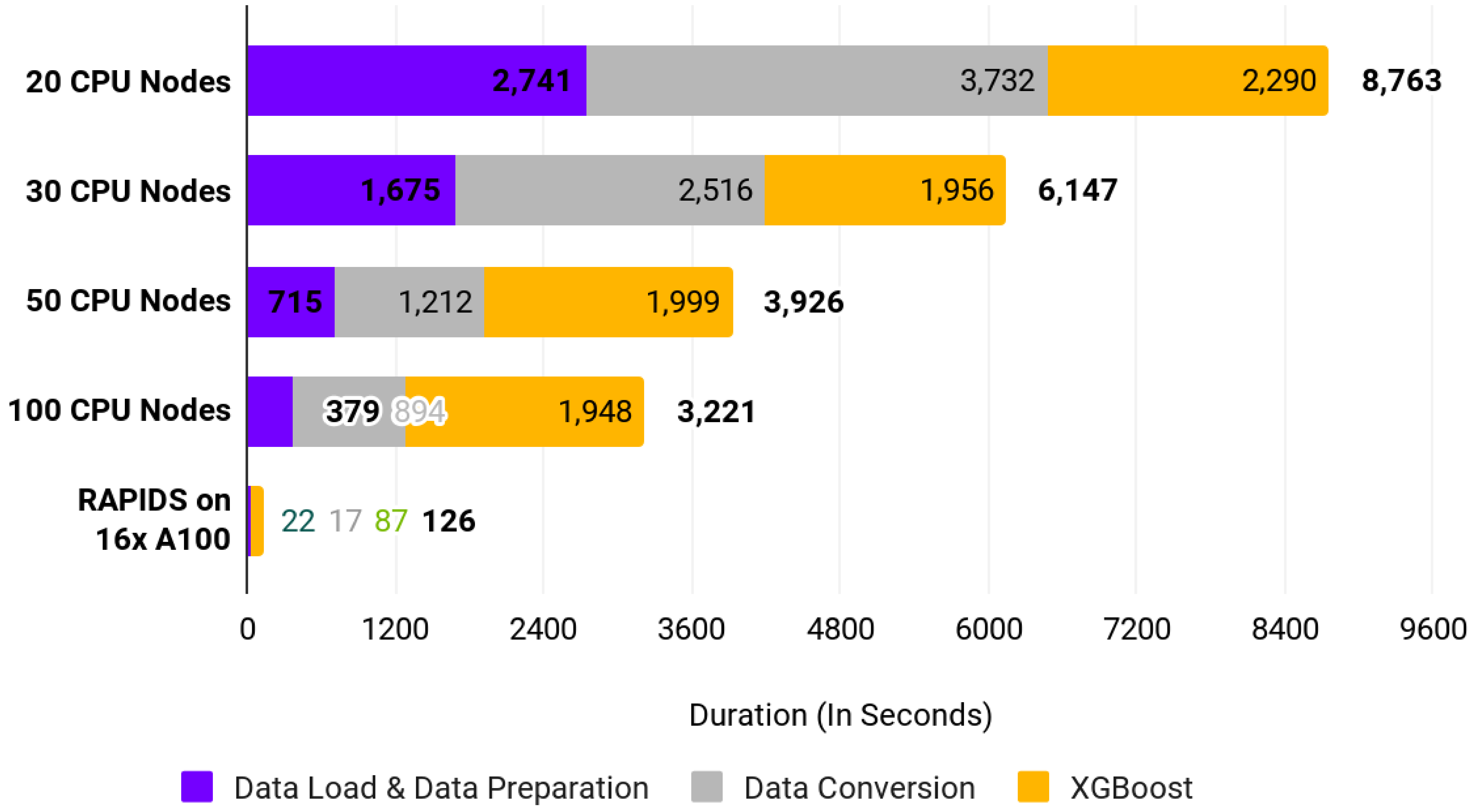
cuGraph

```
>>> import cugraph
>>> page_rank =
cugraph.pagerank(graph)
```

Average Speed-Ups: *250x*

LIGHTNING-FAST END-TO-END PERFORMANCE

Reducing Data Science Processes from Hours to Seconds



RAPIDS End-to-End Workflow Runtimes

CUDF USABILITY IMPROVEMENTS

Making developer life a bit easier

- Expanded IO Readers/Writers and data types (Decimal, List, Struct, Nested, etc.)

- Significant API expansion to empower Pandas and Dask users - 41 new Pandas-compatible APIs added

- **New Pandas-like User Defined Function interface**

- Improved and growing functionality for time-series analysis

```
def custom_add(row):
    if row["a"] > 0:
        return row["a"] + row["b"]
    elif row["a"] is cudf.NA:
        return 99
    else:
        return row["a"]

df["out"] = df.apply(custom_add)
df.head()
```

```
   a             b      out
0  -0.691674315  979    -0.691674
1   0.480099393  1005   1005.480099
2          <NA>  1026   99.000000
3   0.067478787  1026   1026.067479
4  -0.970850075  960    -0.970850
```

TIME SERIES FUNCTIONS

cuDF masters the fourth dimension

- API additions for convenient time-series analysis: date_range() for timestamp generation, interpolate() for fast linear interpolation

- Grouping by a time frequencies with Grouper, as well as Groupby.{corr, std, var, diff} and Groupby.Rolling.{std, var}

- Upsampling and downsampling of time-series data via resample()

- Now calendar-aware! Functions like isocalendar(), quarter(), dayofweek() now available. DateOffsets with non-fixed frequencies like month and year supported.

```
>>> df
                   ts  value
0 2000-01-01 00:00:02      1
1 2000-01-01 00:00:07      2
2 2000-01-01 00:00:02      3
3 2000-01-01 00:00:15      4
4 2000-01-01 00:00:05      5
5 2000-01-01 00:00:09      6

>>> grouper = cudf.Grouper(key="ts", freq="4s")
>>> df.groupby(grouper).mean()


                     value
ts
2000-01-01 00:00:00    2.0
2000-01-01 00:00:04    3.5
2000-01-01 00:00:08    6.0
2000-01-01 00:00:12    4.0
```

CUDA ENHANCED COMPATIBILITY

RAPIDS Ecosystem Integration

- Support for CUDA Minor Version compatibility starting from the 21.12 release
- No longer need to update your CUDA driver or toolkit to use RAPIDS with CUDA 11 and driver >= 450.80.02.
- Enables seamless compatibility with other GPU libraries, like PyTorch and Tensorflow

RAPIDS

PyTorch

TensorFlow

NEW PLATFORMS, NEW CONTAINERS

RAPIDS Going Everywhere

- Windows Subsystem for Linux in 21.10 (experimental)
- ARM SBSA support in 21.10 (experimental)
- CUDA 11.5 support in 21.12
- NVIDIA NGC optimized containers for PyTorch and TensorFlow now include RAPIDS libraries

HOW TO GET STARTED WITH RAPIDS

A Variety of Ways to Get Up & Running

More about RAPIDS

- Learn more at RAPIDS.ai
- Read the API docs
- Check out the RAPIDS blog
- Read the NVIDIA DevBlog

Self-Start Resources

- Get started with RAPIDS
- Deploy on the Cloud today
- Start with Google Colab
- Look at the cheat sheets

Discussion & Support

- Check the RAPIDS GitHub
- Use the NVIDIA Forums
- Reach out on Slack
- Talk to NVIDIA Services

Get Engaged

@RAPIDSai

https://github.com/rapidsai

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

RAPIDS

https://rapids.ai

CUPY

GETTING STARTED

NumPy

- “The fundamental package for scientific computing with Python”
- N-Dimensional array and numerical computing
- API closely matched by several Python projects (*CuPy*, Dask, JAX, and others)

GETTING STARTED

CuPy

- “The fundamental package for *CUDA* scientific computing with Python”
- N-Dimensional array and numerical computing
- Matches the NumPy API
- Extrapolates NumPy API where necessary, e.g., sparse computing

GETTING STARTED

Array Basics

- CuPy Implements NumPy-compatible API

NumPy L2 example:

```
>>> import numpy as np
>>> a_cpu = np.array([1,2,3])
>>> a_cpu
array([1, 2, 3])
>>> type(a_cpu)
<class 'numpy.ndarray'>
>>>
>>> l2_cpu = np.linalg.norm(a_cpu)
>>> l2_cpu
3.7416573867739413
```

CuPy equivalent:

```
>>> import cupy as cp
>>> a_gpu = cp.array([1,2,3])
>>> a_gpu
array([1, 2, 3])
>>> type(a_gpu)
<class 'cupy.core.core.ndarray'>
>>>
>>> l2_gpu = cp.linalg.norm(a_gpu)
>>> l2_gpu
array(3.74165739)
>>>
>>> # Note the output here is a CuPy array
>>> # and not a Python float, intentionally
>>> # avoiding implicit D2H copy
```

GETTING STARTED

Array Basics

- CuPy Implements NumPy-compatible API

NumPy transpose matrix-multiply example:

```
>>> import numpy as np
>>> a_cpu = np.array([1,2,3])
>>> a_cpu
array([1, 2, 3])
>>> a_cpu * a_cpu.T  # 1D array transpose
array([1, 4, 9])
>>>
>>> a_cpu = a_cpu.reshape((1, 3))
>>> a_cpu  # This is now a 2D array
array([[1, 2, 3]])
>>> a_cpu * a_cpu.T
array([[1, 2, 3],
       [2, 4, 6],
       [3, 6, 9]])
```

CuPy equivalent:

```
>>> import cupy as cp
>>> a_gpu = cp.array([1,2,3])
>>> a_gpu
array([1, 2, 3])
>>> a_gpu * a_gpu.T  # 1D array transpose
array([1, 4, 9])
>>>
>>> a_gpu = a_gpu.reshape((1, 3))
>>> a_gpu  # This is now a 2D array
array([[1, 2, 3]])
>>> a_gpu * a_gpu.T
array([[1, 2, 3],
       [2, 4, 6],
       [3, 6, 9]])
```

GETTING STARTED

Choosing Device

- **CuPy is single-GPU**

- Device can be set for the default context or temporarily for local context only

```
# Default is Device 0
a_gpu0 = cp.array([1,2,3])

# Switch to Device 1
cp.cuda.Device(1).use()
a_gpu1 = cp.array([1,2,3])

# Temporarily switch to Device 2
with cp.cuda.Device(2):
    a_gpu2 = cp.array([1,2,3])

# Back to Device 1
b_gpu1 = cp.array([1,2,3])
```

GETTING STARTED

Data Transfer

- Data movement functions are not part of NumPy’s API

```
a_cpu = np.array([1,2,3])

# Copy a_cpu to array a_gpu0 in device 0
with cp.cuda.Device(0):
    a_gpu0 = cp.asarray(a_cpu)

# Copy a_gpu0 to array a_gpu1 in device 1
with cp.cuda.Device(1):
    a_gpu1 = cp.asarray(a_gpu0)

# Copy a_gpu1 back to array b_cpu on host
b_cpu = cp.asnumpy(a_gpu1)
b_cpu = a_gpu1.get()                    # Equivalent to cp.asnumpy(a_gpu1)
```

LOW LEVEL CUDA SUPPORT

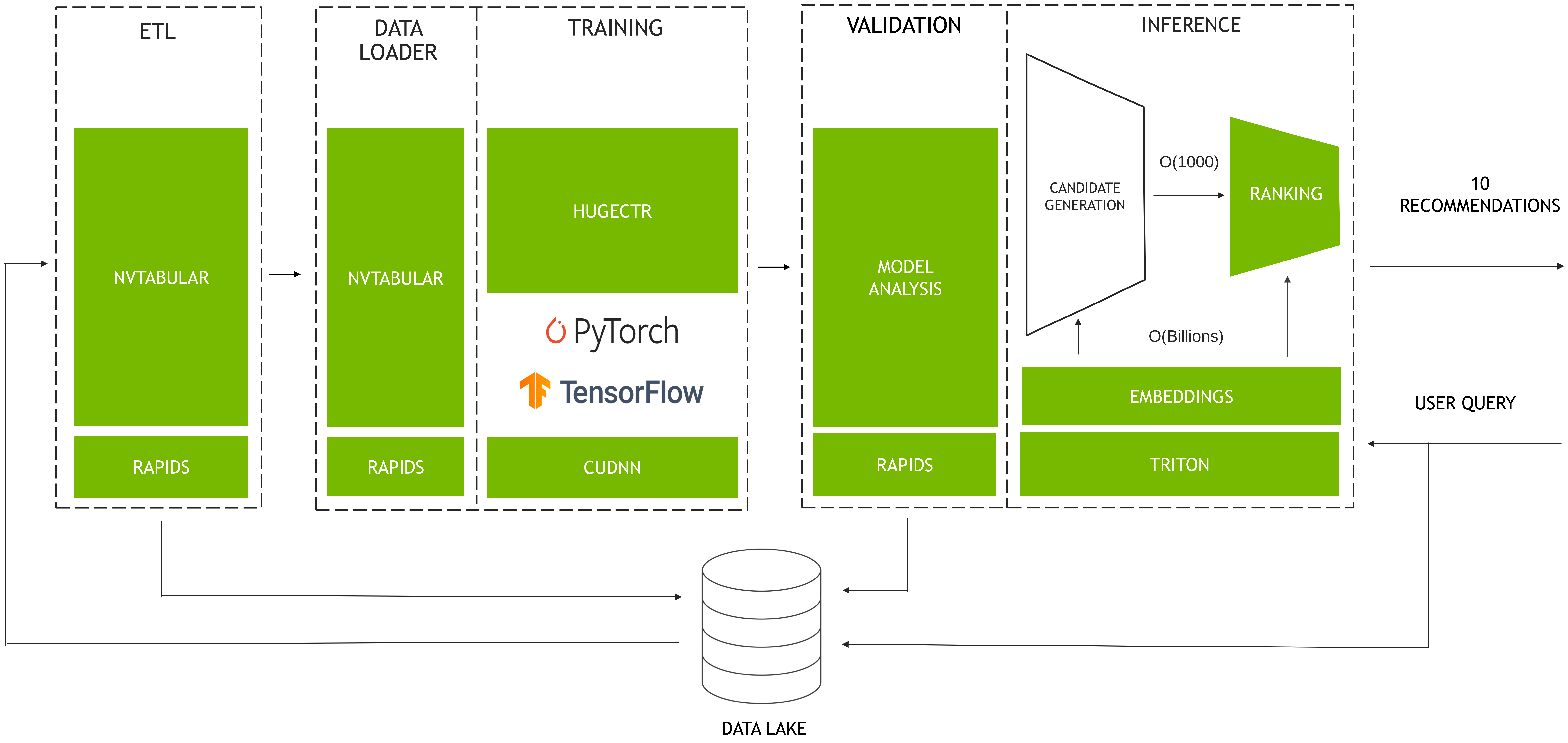
Memory Management

- Default memory pool caches allocated memory blocks for later reuse

```
CUPY_GPU_MEMORY_LIMIT=1073741824 python
>>> import cupy as cp
>>> mempool = cp.get_default_memory_pool()
>>> mempool.get_limit()
1073741824
>>> with cp.cuda.Device(1):
...     mempool.set_limit(2*1024**3)
...     mempool.get_limit()
...
2147483648
>>> with cp.cuda.Device(0):
...     mempool.get_limit()
...
1073741824
```

MERLIN

MERLIN IS AN END-2-END LIBRARY FOR GPU-ACCELERATED RECOMMENDER SYSTEMS



BUILDING RECOMMENDER SYSTEMS END-TO-END IS COMPLEX AND REQUIRES 4 STAGES

NVIDIA MERLIN

Retrieval
Filtering
Scoring
Ordering
Retrieval
Filtering
Scoring
Ordering

Generating ~1000-10000 candidates used for recommendations from all available items.

Scoring all available items is computential too expensive, therefore only a subset should be scored

Removing invalid candidates

For example, products out of stock

More advanced models scores the candidates and determines the best recommendations

Reordering the recommendations by other business logic

⬚ Stage which is often referred to for Recommender Systems

EXAMPLES FOR THE 4 STAGE RECOMMENDER SYSTEMS

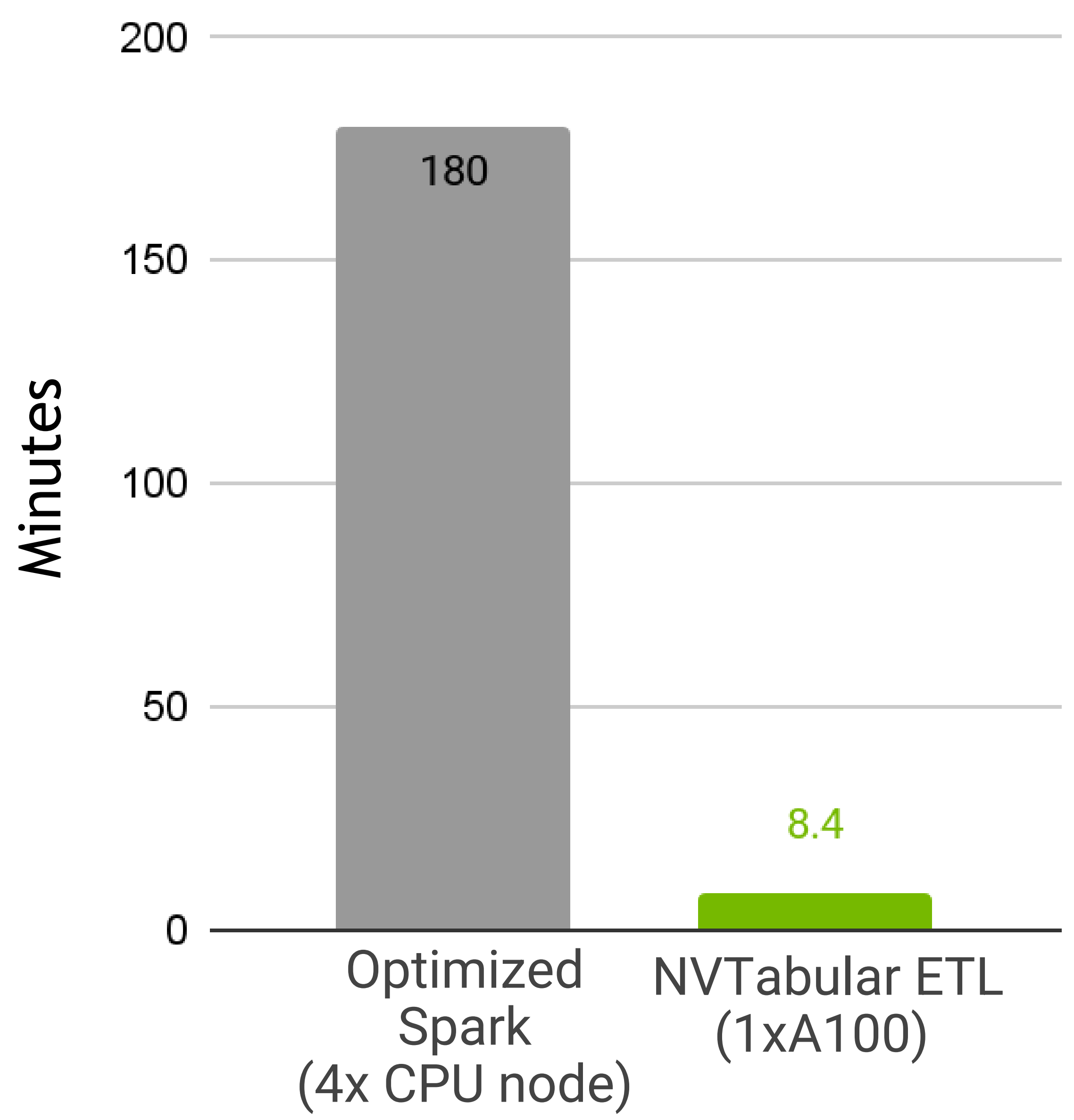
	Retrieval	Filtering	Scoring	Ordering
Music Discovery	Find similar songs based on nearest neighbour search	Remove tracks users listen before	Predict likelihood user will listen to the song	Trade-Off between score, similarity, BPM, etc
Social Media	Find new posts in user’s network	Remove posts from blocked and muted users	Predict likelihood user will interact with it	Change order that adjust posts are from different authors
Online Store	Find items which are usually co-purchased	Remove items which are out of stock	Predict likelihood user will purchase the item	Reorder items based on price points
Streaming Service	Find items based on different rows/shelves/topics	Remove items which are not available for user’s country	Predict user’s stream time per item	Organize recommendations to fit genre distributions

MERLIN SPEEDS UP THE ENTIRE PIPELINE

Speedup

NVTabular

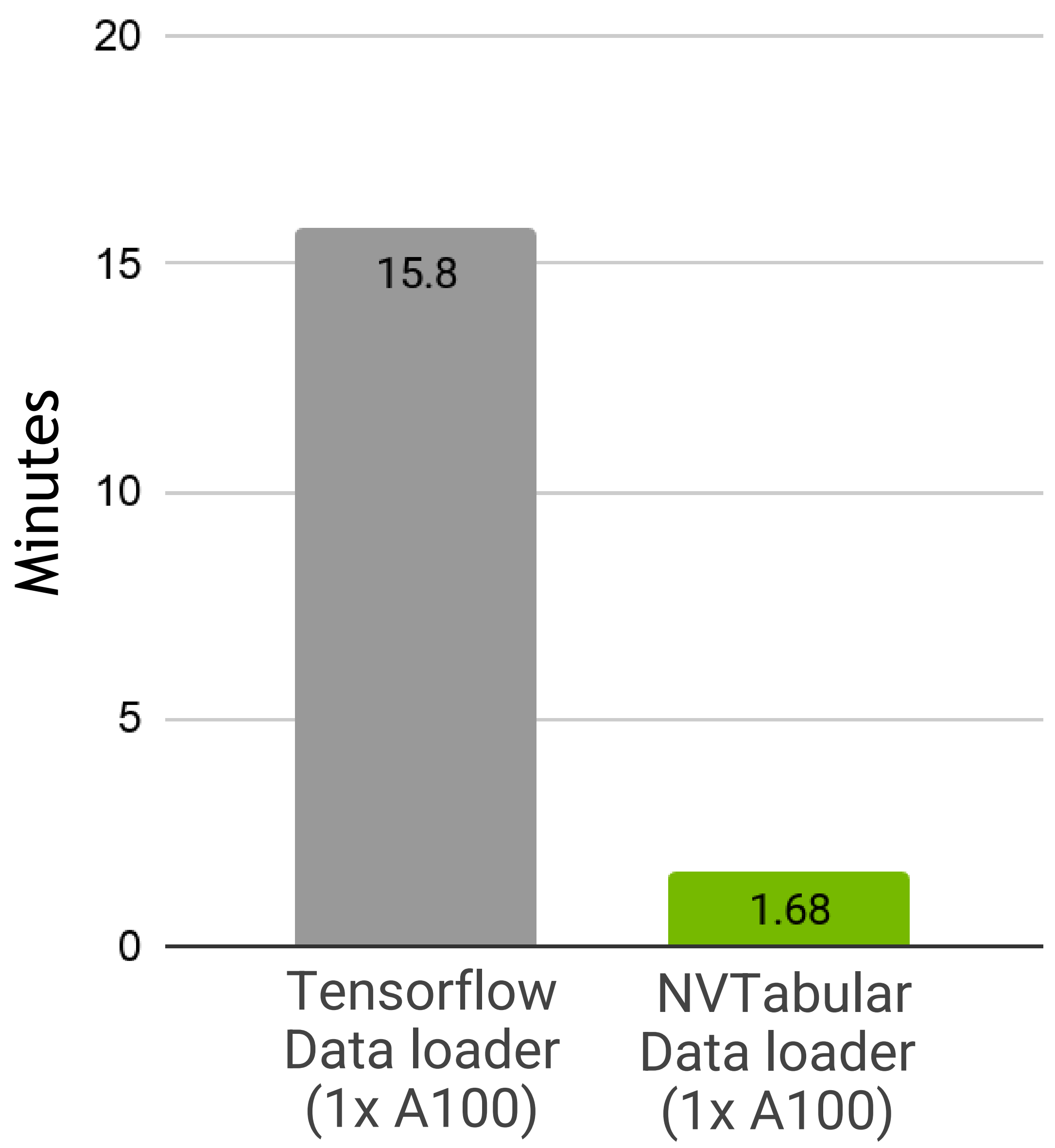
ETL



21x

TF/PyT Plugins

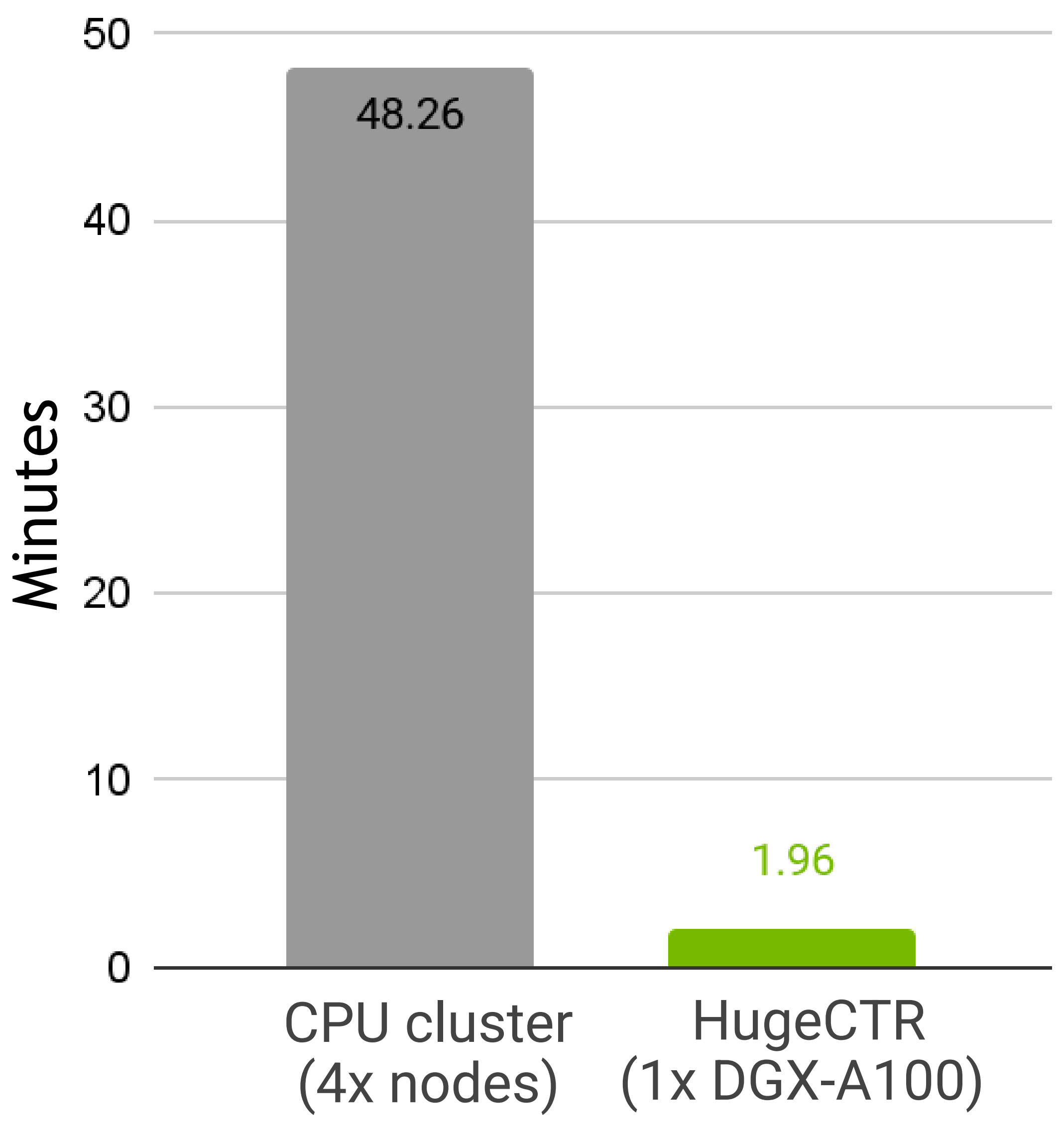
Accelerating Training



9x

HugeCTR

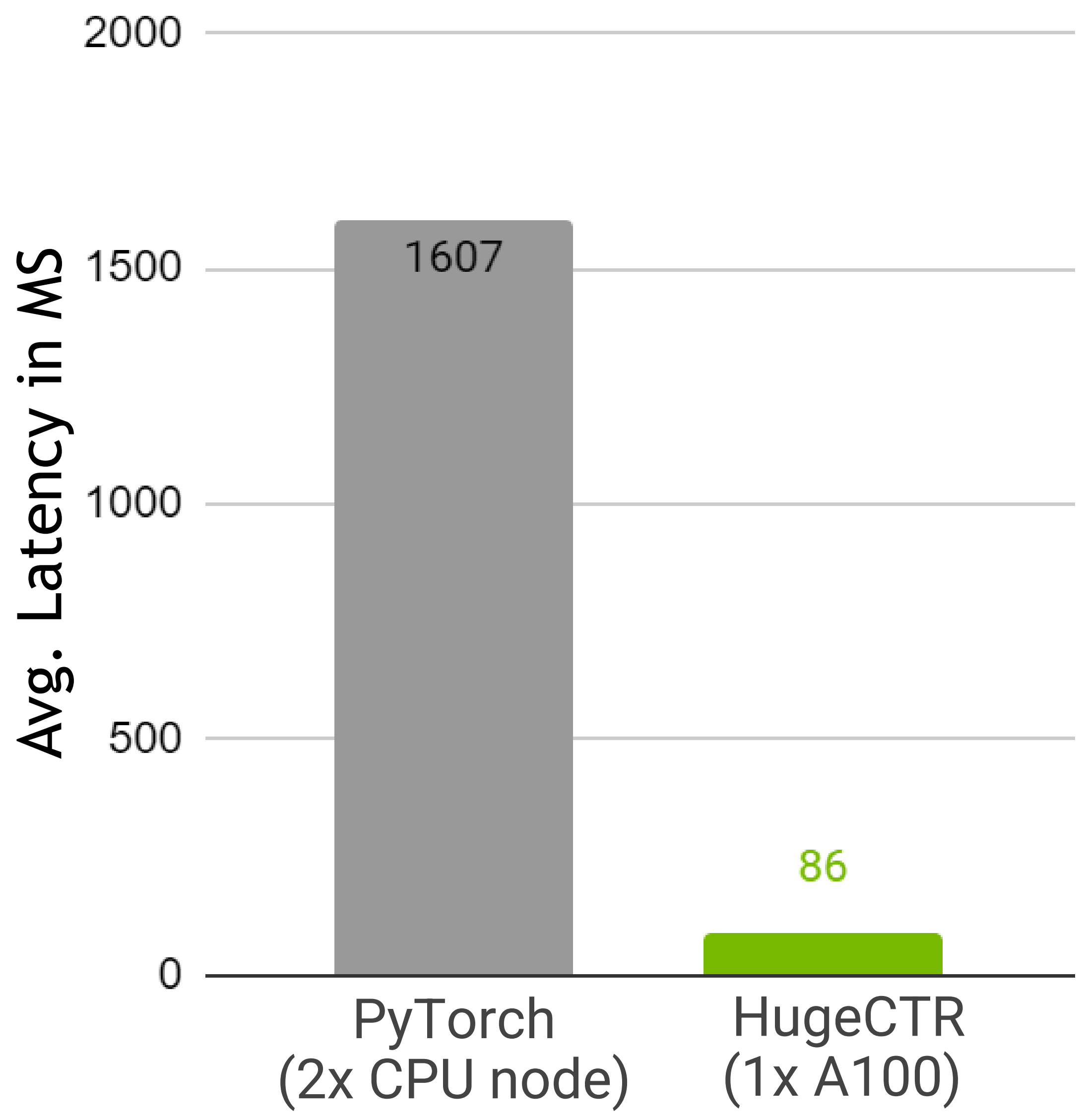
Scaling Accelerated Training



24x

Triton

Inference



18x

NVIDIA Merlin provides 9-24x speed-up in ETL+Training+Inference RecSys models and easily scales to multiple GPUs

THANK YOU