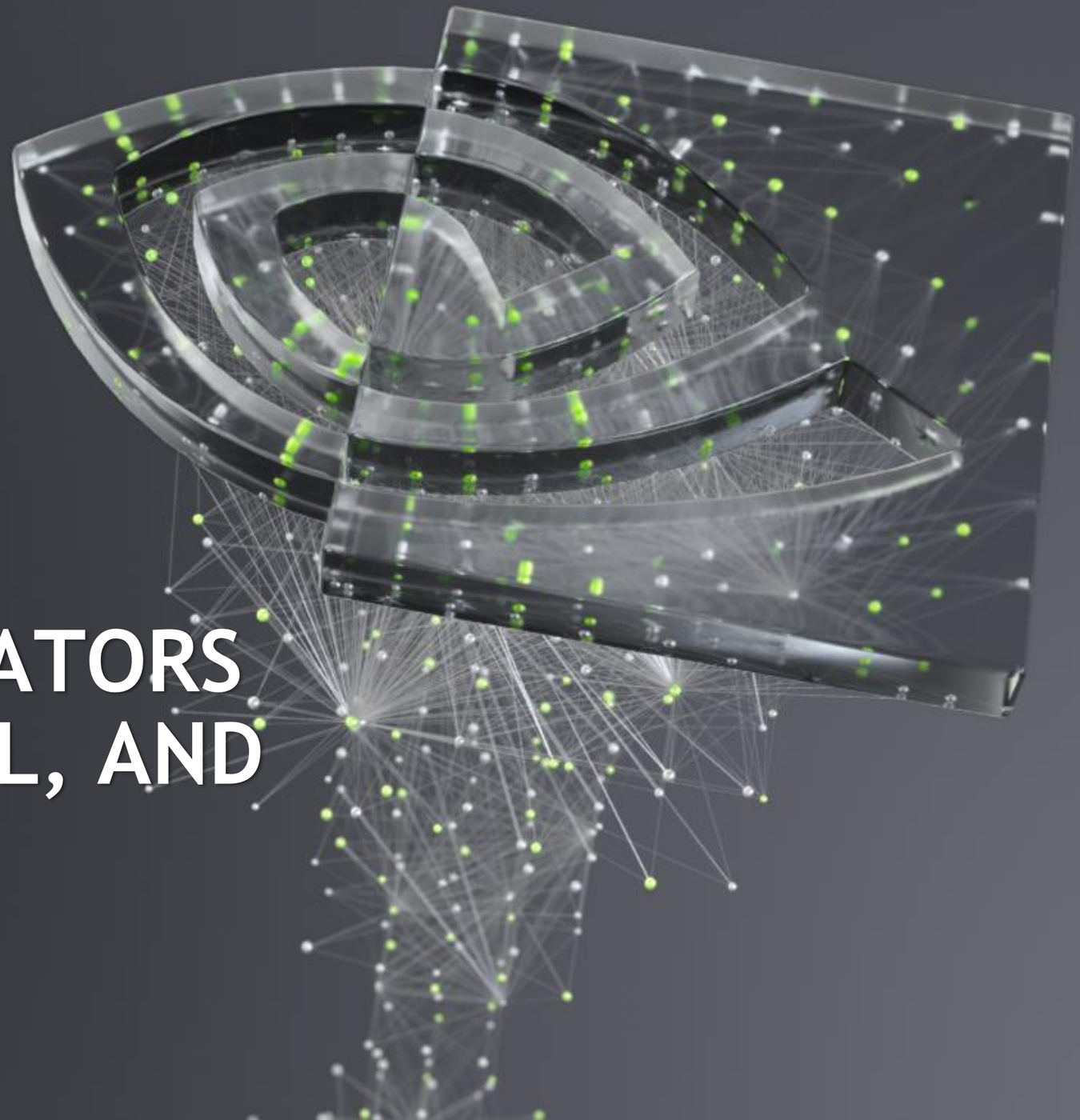


UTILIZING ACCELERATORS TO SPEEDUP ETL, ML, AND DL APPLICATIONS

Jason Lowe and Robert Evans, 05/19/2020

AGENDA

Accelerated ETL

Accelerated SQL/Dataframe

Accelerated Shuffle

What's Next

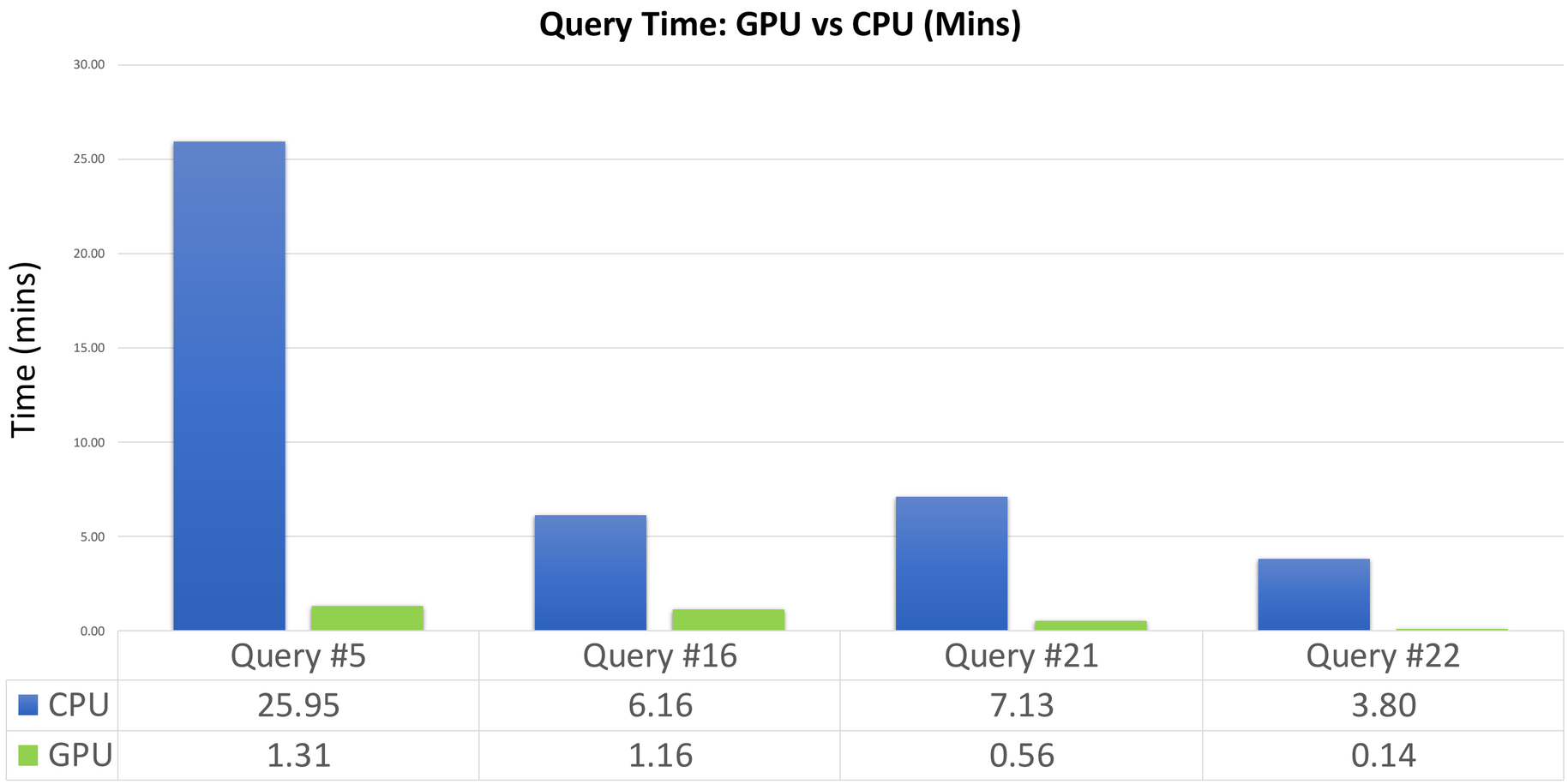
ACCELERATED ETL?

Can a GPU make an elephant fast?

https://www.piqsels.com/en/public-domain-photo-zrkia

YES

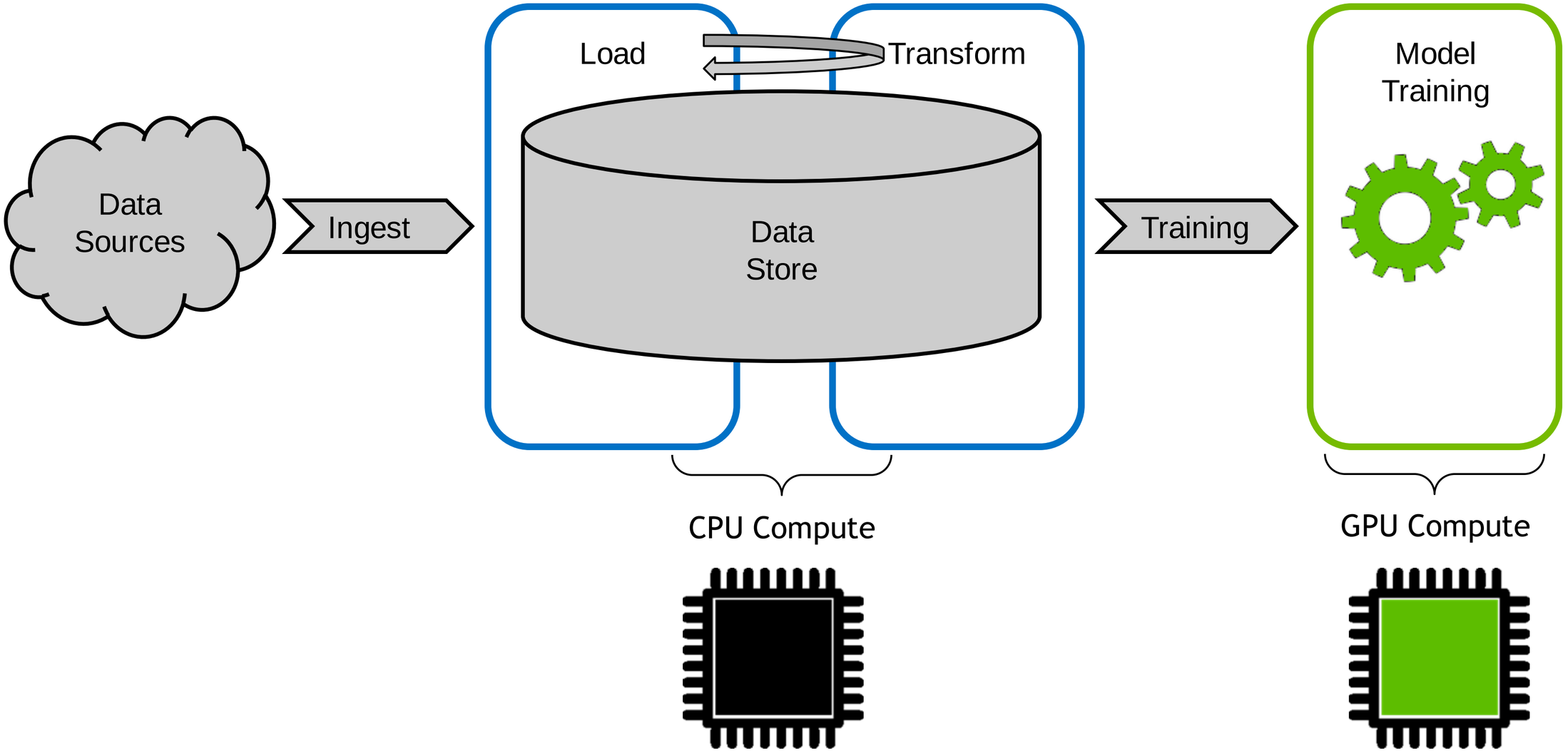
TPCx—BB Like Benchmark Results (10TB Dataset, Two Nodes DGX-2 Cluster)*



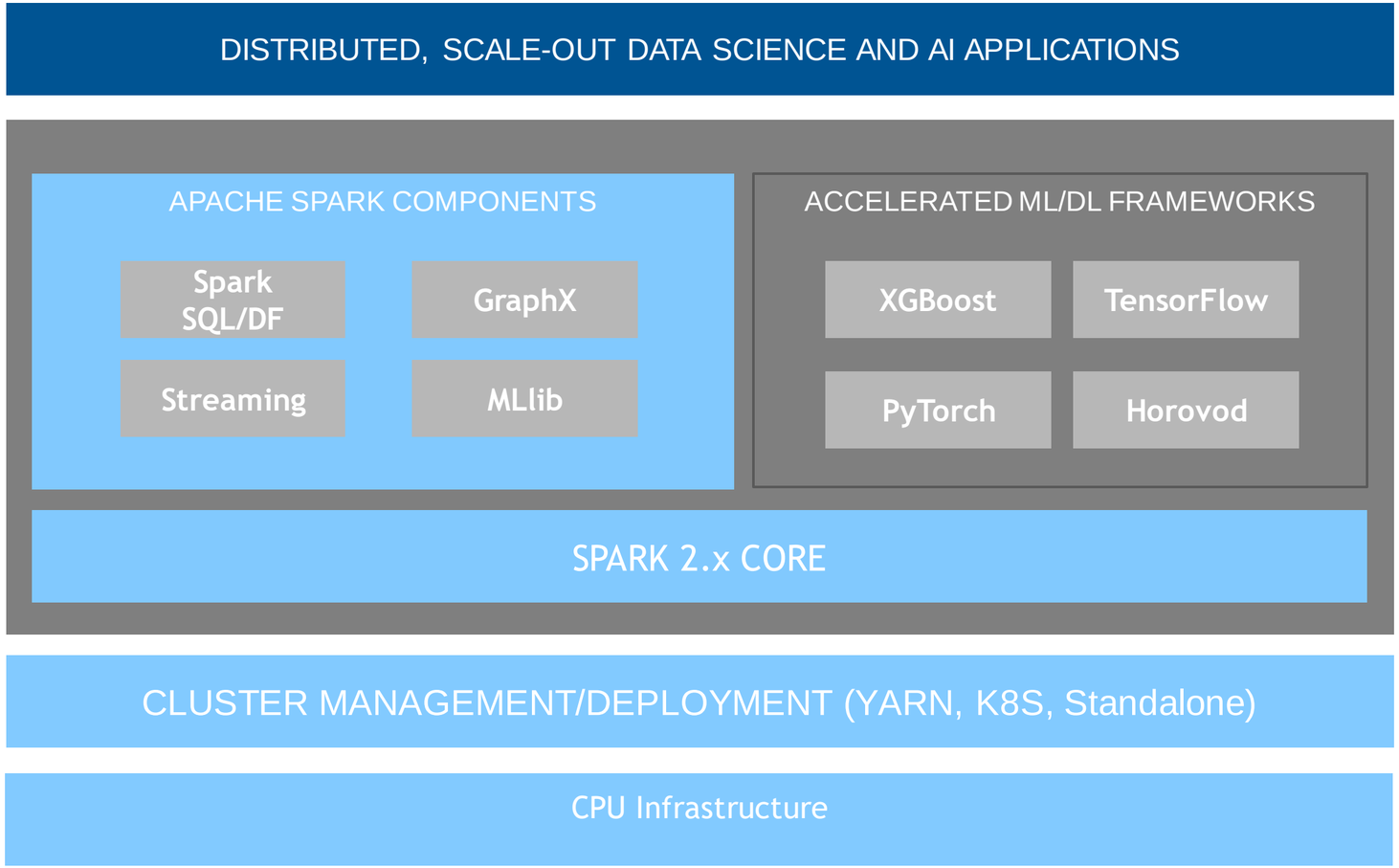
	Query #5	Query #16	Query #21	Query #22
CPU	25.95	6.16	7.13	3.80
GPU	1.31	1.16	0.56	0.14

Environment: Two DGX-2 (96 CPU Cores, 1.5TB Host memory, 16 V100 GPUs, 512 GB GPU Memory)
* Not official or complete TPCx-BB runs (ETL power only).

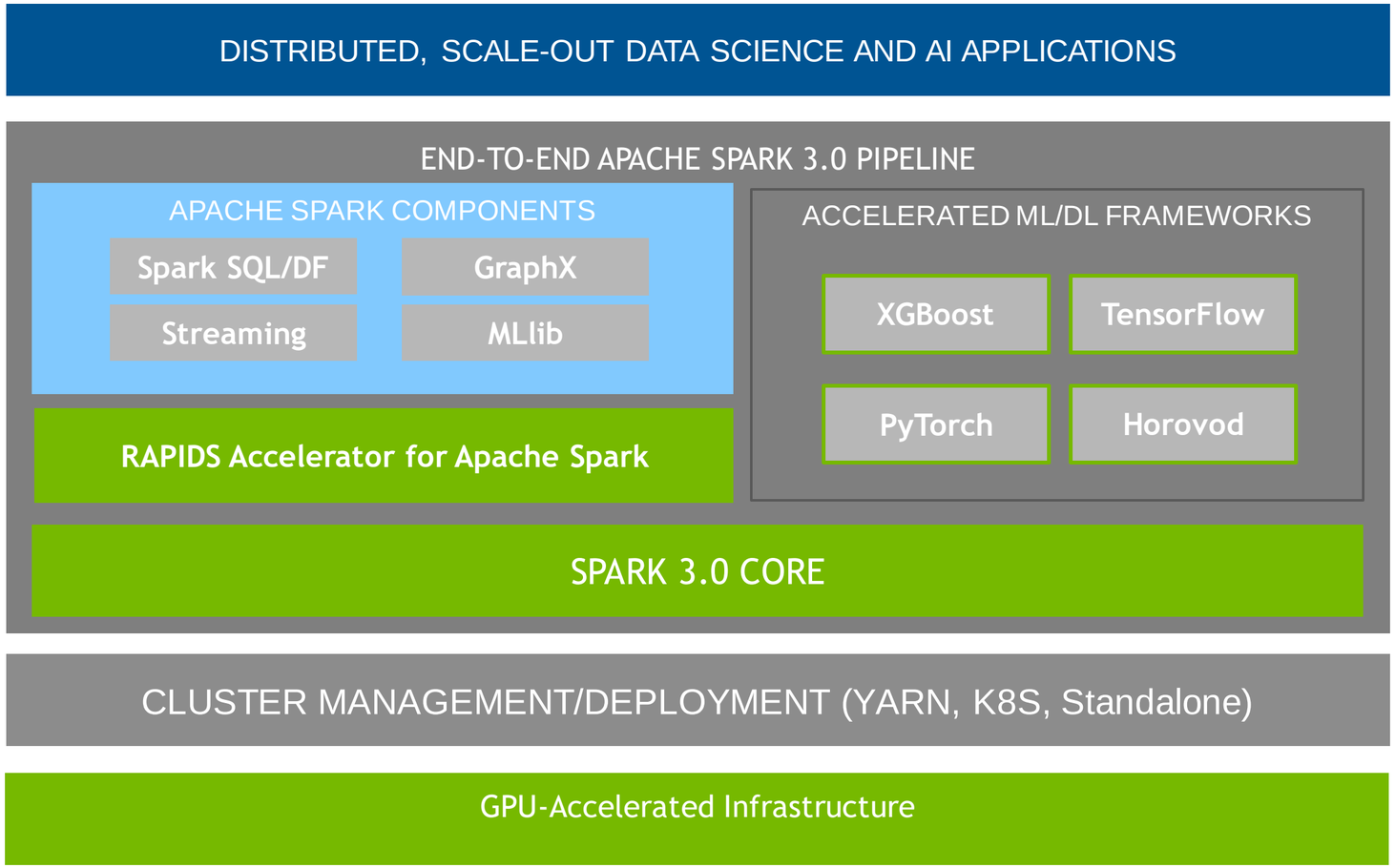
MODERN ML/DL WORKFLOW



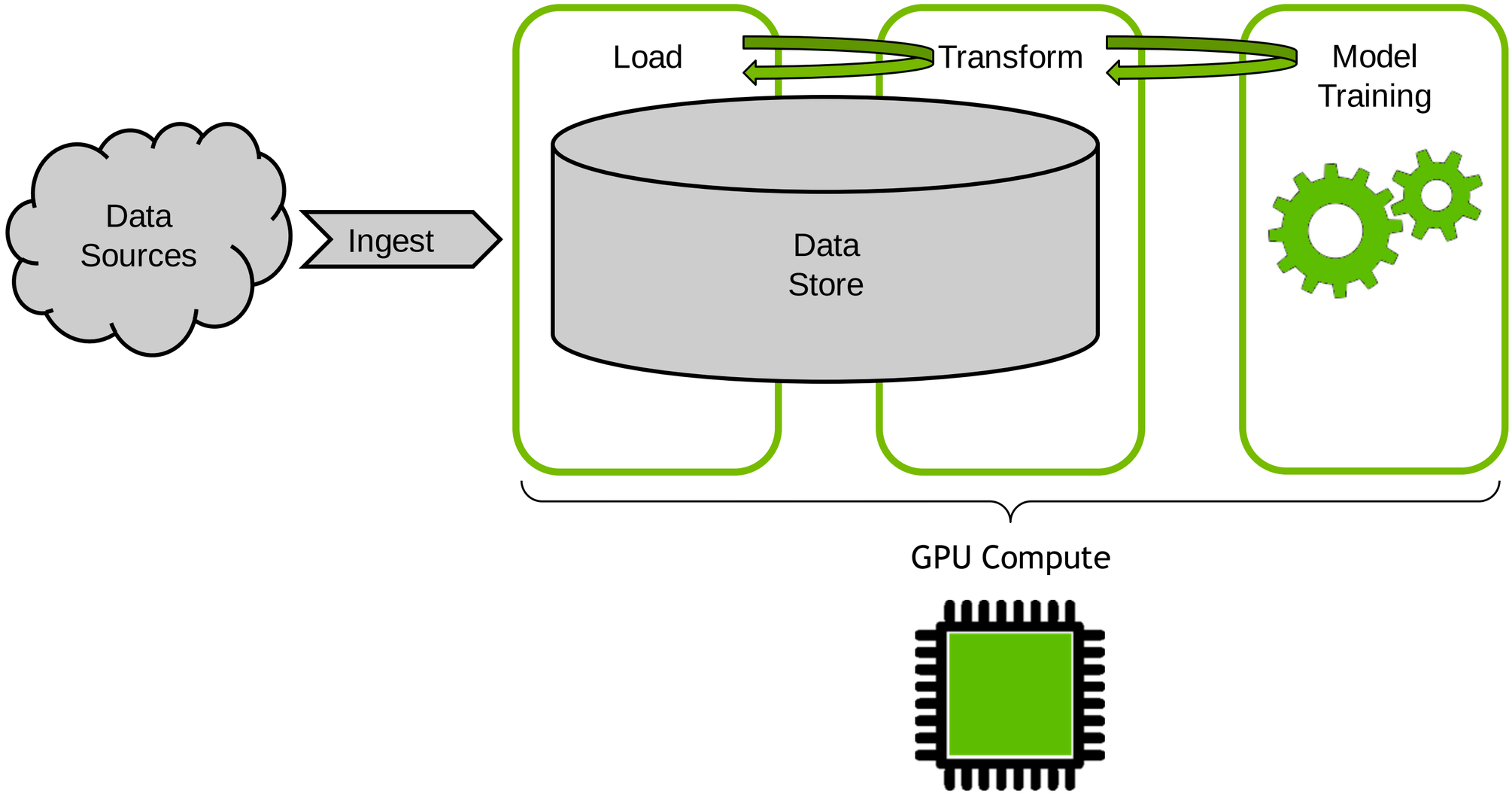
APACHE SPARK 2.X



SPARK 3.X IS A UNIFIED AI PLATFORM



ETL + ML/DL WORKFLOW



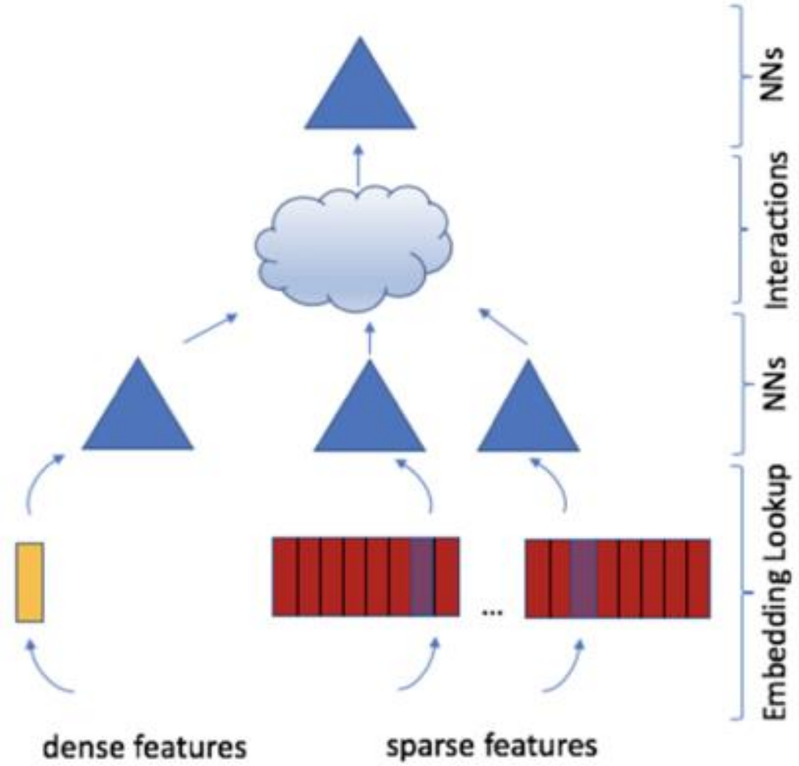
DEEP LEARNING RECOMMENDATION MACHINES

Example use case: Criteo dataset

Anonymized 7-day clickstream dataset (1 TB)

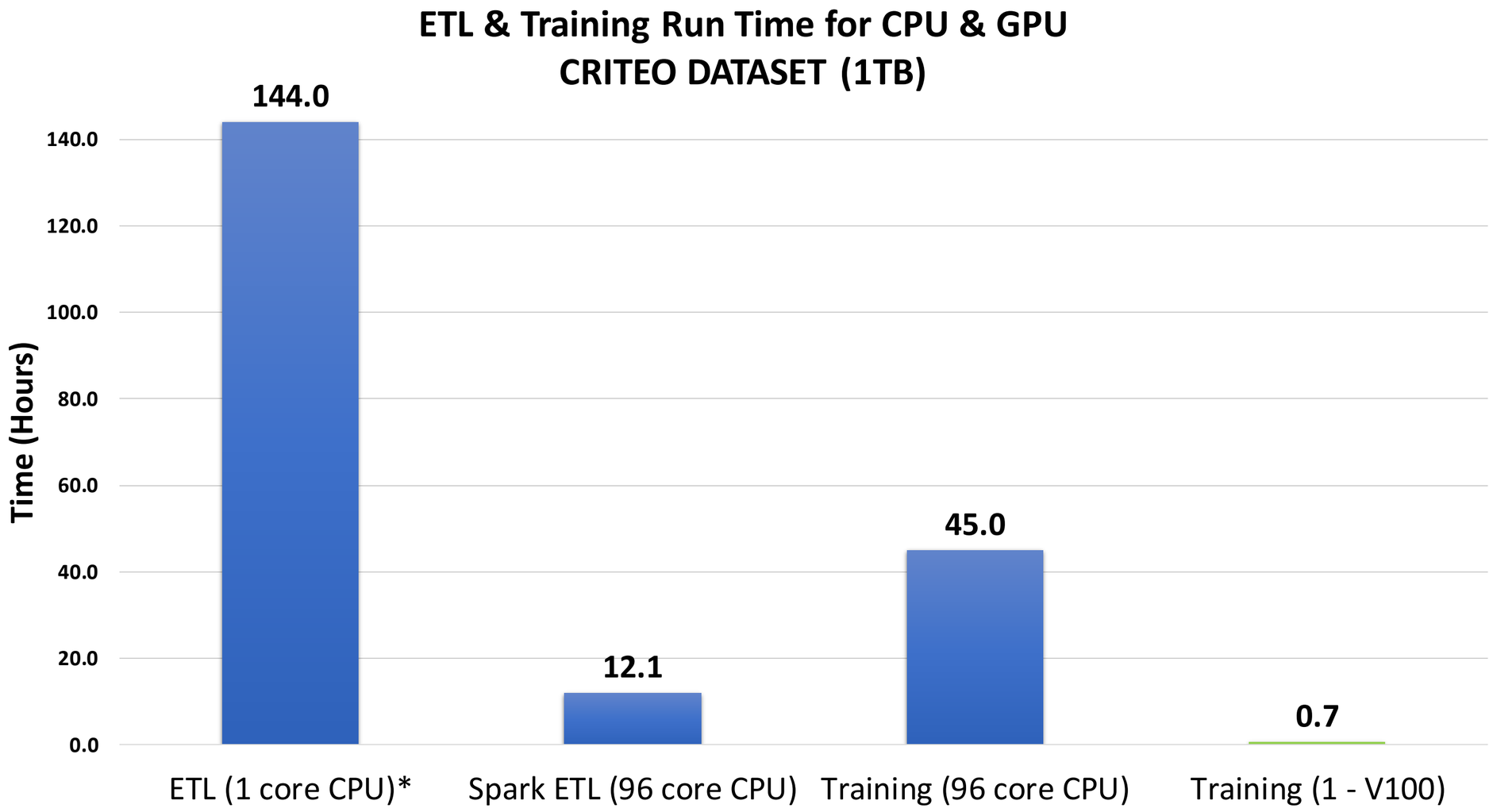
Convert high cardinality string categorical data to contiguous integer ids

DLRM github repo has scripts for this out of the box



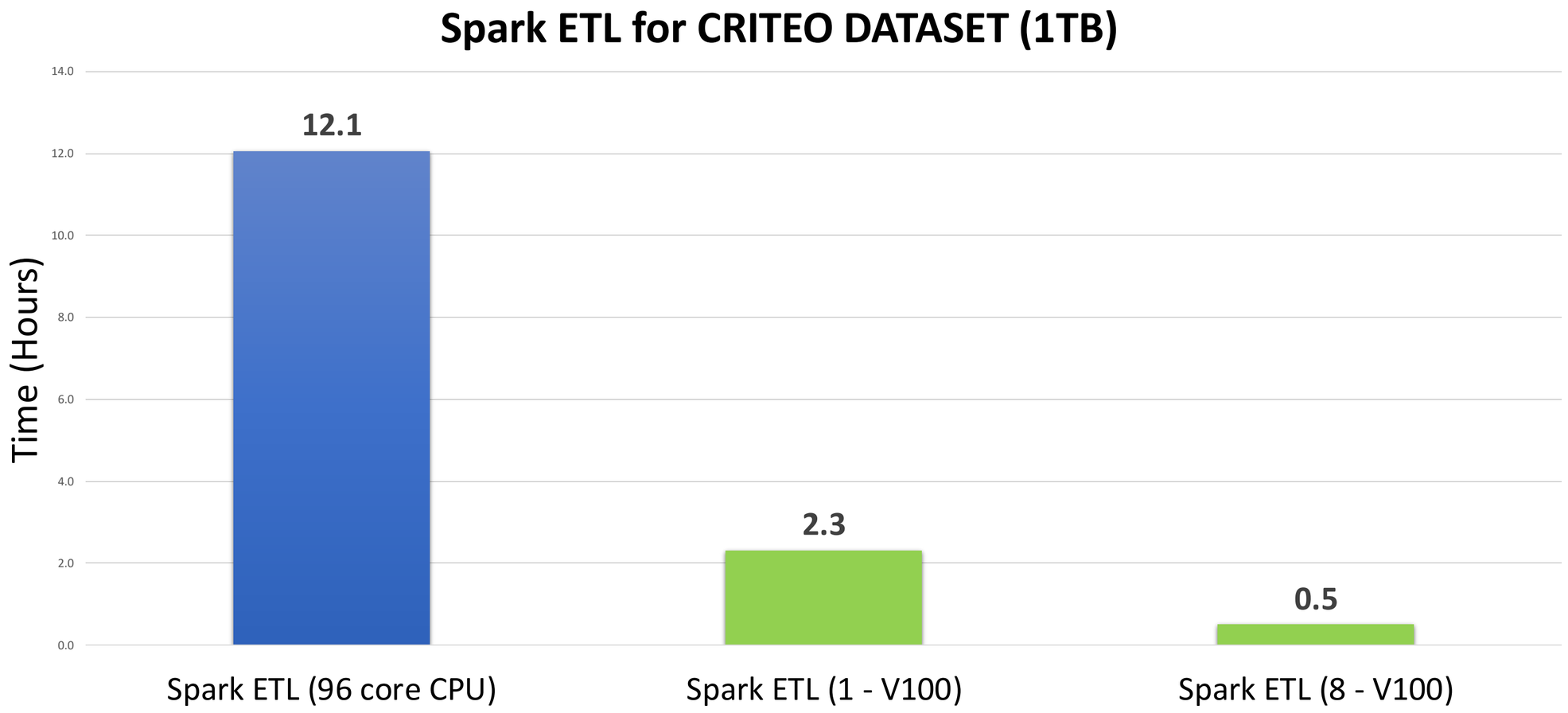
https://medium.com/analytics-vidhya/deep-learning-recommendation-machines-dlrm-4fec2a5e7ef8

DLRM ON CRITEO DATASET (PAST)

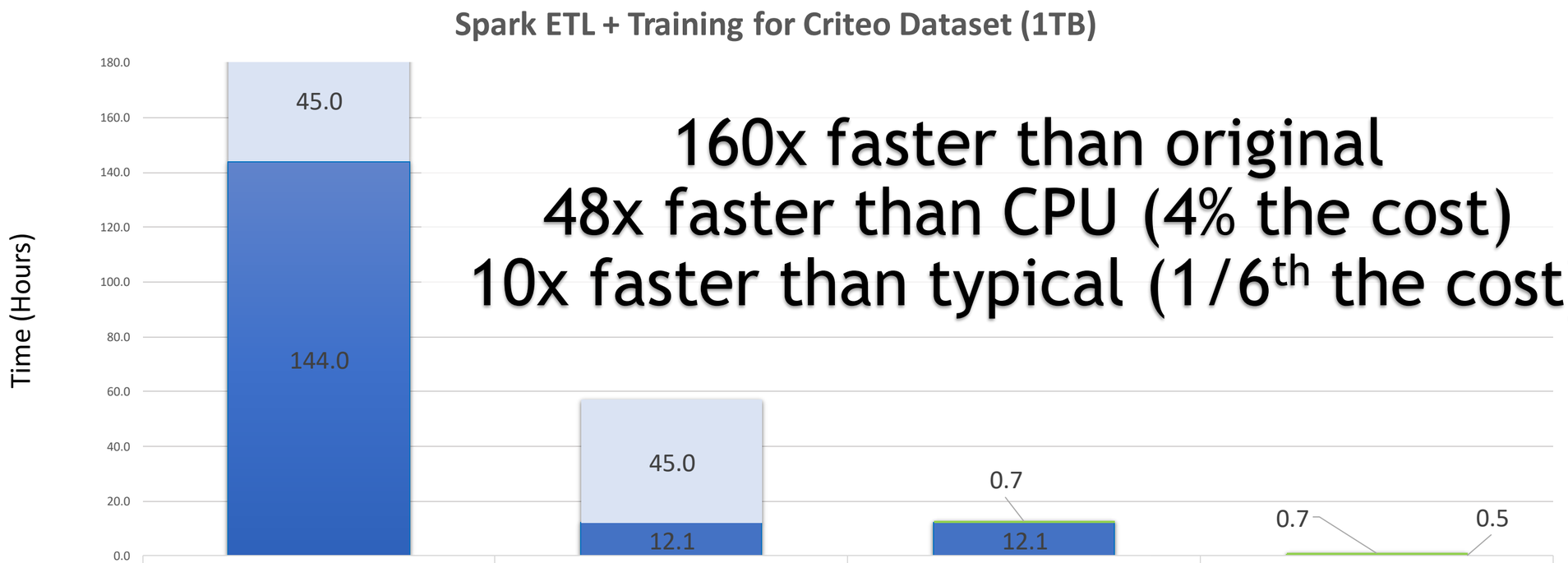


* Extrapolated couldn't convince anyone to wait that long

DLRM ETL ON CRITEO DATASET (PRESENT)



DLRM END-TO-END ON CRITEO DATASET (PRESENT)

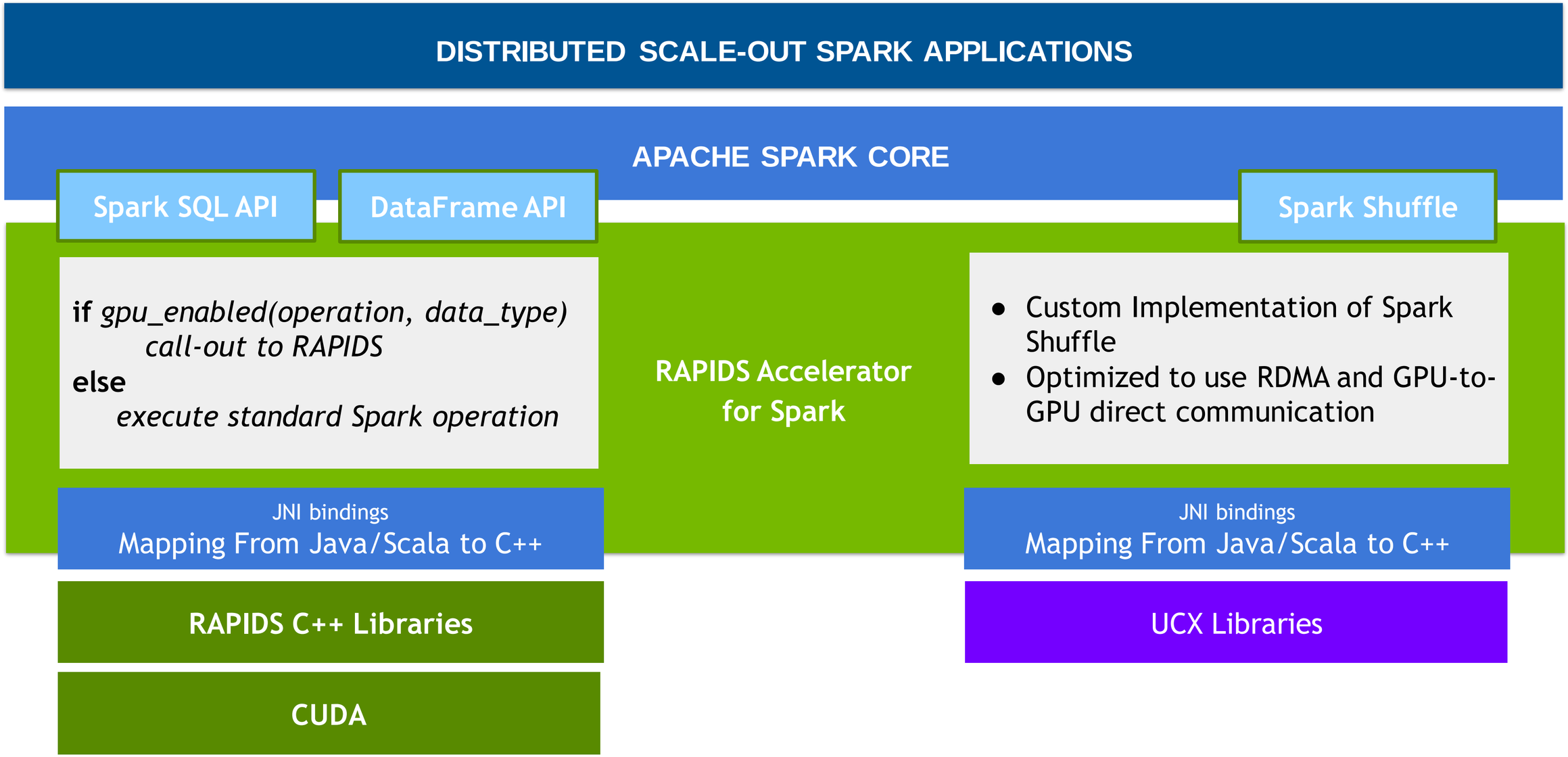


	Original CPU (1 Core for ETL, 96 Core CPU for Training)	Spark CPU (96 Core for ETL & Training)	Spark CPU (96 Core for ETL) & Spark GPU (1-V100 Training)	Spark GPU (8-V100 for ETL & 1-V100 Training)
Training	45.0	45.0	0.7	0.7
ETL	144.0	12.1	12.1	0.5

“The more you buy, the more you save.”

— Jensen Huang, GTC 2018

RAPIDS ACCELERATOR FOR APACHE SPARK (PLUGIN)



SQL/DATAFRAME PLUGIN

No Code Changes

(none)

Same SQL and Dataframe code.

```
spark.conf.set("spark.rapids.sql.enabled", "true")
```

```
start = time.time()
spark.sql("""
select
        o_orderpriority,
        count(*) as order_count
from
        orders
where
        o_orderdate >= date '1993-07-01'
        and o_orderdate < date '1993-07-01' + interval '3' month
        and exists (
                select
                        *
                from
                        lineitem
                where
                        l_orderkey = o_orderkey
                        and l_commitdate < l_receiptdate
        )
group by
        o_orderpriority
order by
        o_orderpriority""").show()
time.time() - start
```

WHAT WE SUPPORT

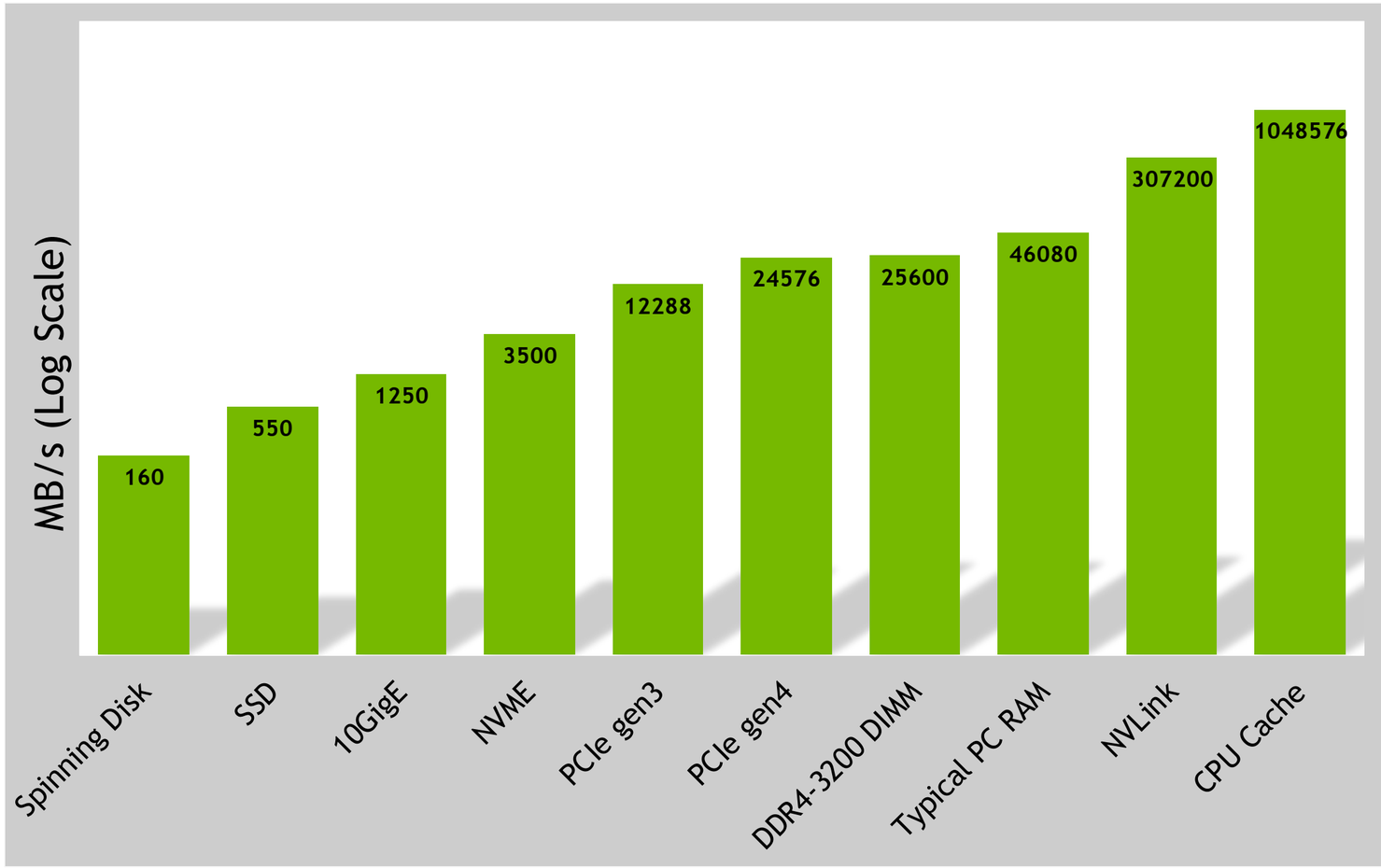
and growing…

- !
- %
- &
- *
- +
- -
- /
- <
- <=
- <=>
- =
- ==
- >
- >=
- ^
- abs
- acos
- and
- asin
- atan
- avg
- bigint
- boolean
- cast
- cbrt
- ceil
- ceiling
- coalesce
- concat
- cos
- cosh
- cot
- count
- cube
- current_date
- current_timestamp
- date
- datediff
- day
- dayofmonth
- degrees
- double
- e
- exp
- expm1
- first
- first_value
- float
- floor
- from_unixtime
- hour
- if
- ifnull
- in
- initcap
- input_file_block_length
- input_file_block_start
- input_file_name
- int
- isnan
- isnotnull
- isnull
- last
- last_value
- lcase
- like
- ln
- locate
- log
- log10
- log1p
- log2
- lower
- max
- mean
- min
- minute
- mod
- monotonically_increasing_id
- month
- nanvl
- negative
- not
- now
- nullif
- nvl
- nvl2
- or
- pi
- posexplode*
- position
- pow
- power
- radians
- rand*
- regexp_replace*
- replace
- rint
- rollup
- row_number
- second
- shiftleft
- shiftright
- shiftrightunsigned
- sign
- signum
- sin
- sinh
- smallint
- spark_partition_id
- sqrt
- string
- substr
- substring
- sum
- tan
- tanh
- timestamp
- tinyint
- trim
- ucase
- upper
- when
- window
- year
- |
- ~
- CSV Reading*
- Orc Reading
- Orc Writing
- Parquet Reading
- Parquet Writing
- ANSI casts
- TimeSub for time ranges
- startswith
- endswith
- contains
- limit
- order by
- group by
- filter
- union
- repartition
- equi-joins
- select

IS THIS A SILVER BULLET?

NO

- Small amounts of data
  - Few hundred MB per partition for GPU
- Highly cache coherent processing
- Data Movement
  - Slow I/O (networking, disks, etc.)
  - Going back and forth to the CPU (UDFs)
  - Shuffle
- Limited GPU Memory



BUT IT CAN BE AMAZING

What the SQL plugin excels at

High cardinality joins

High cardinality aggregates

High cardinality sort

Window operations (especially on large windows)

Complicated processing

Transcoding (Writing Parquet and ORC is hard, reading CSV is hard)

HOW DOES IT WORK

SPARK SQL & DATAFRAME COMPILATION FLOW

QUERY

```
SELECT product_id, ds,
max(price) - min(price) AS
range FROM bar GROUP BY
product_id, ds
```

```
bar.groupBy(
    col("product_id"),
    col("ds"))
.agg(
    max(col("price")) -

min(col("price")).alias("range"))
```

DataFrame

Logical Plan

Physical Plan

RDD[InternalRow]

CPU PHYSICAL PLAN

SPARK SQL & DATAFRAME COMPILATION FLOW

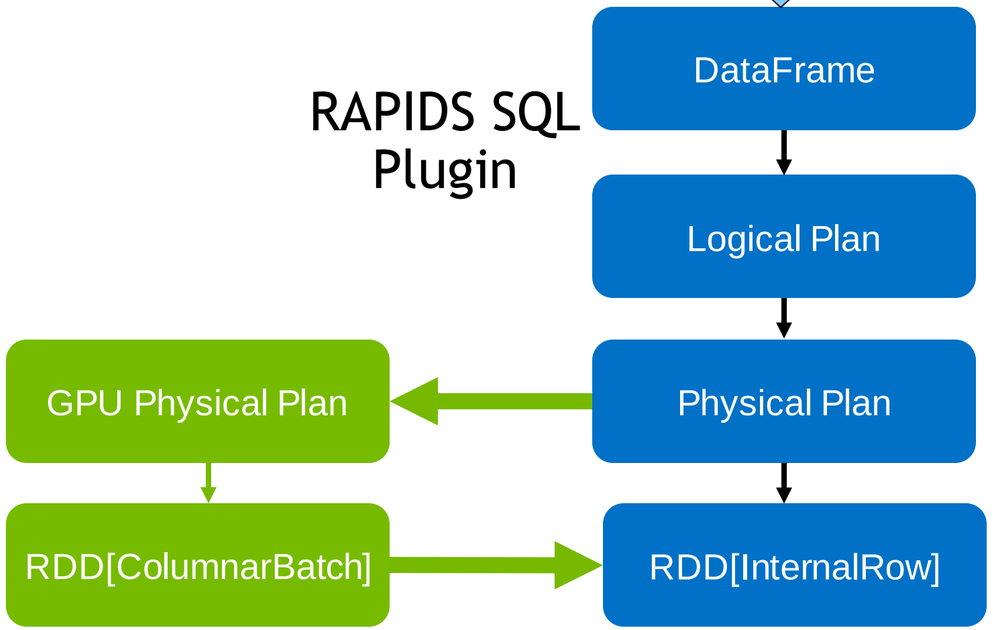
QUERY

```
SELECT product_id, ds,
max(price) - min(price) AS
range FROM bar GROUP BY
product_id, ds
```

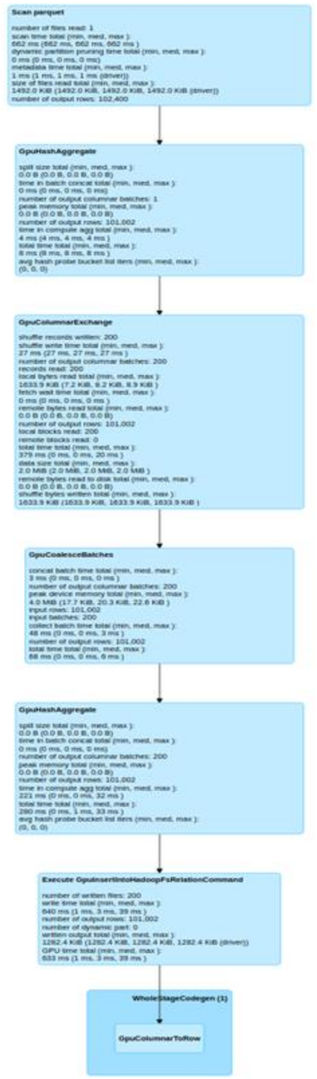
```
bar.groupBy(
    col("product_id"),
    col("ds"))
.agg(
    max(col("price")) -

min(col("price")).alias("range"))
```

RAPIDS SQL Plugin

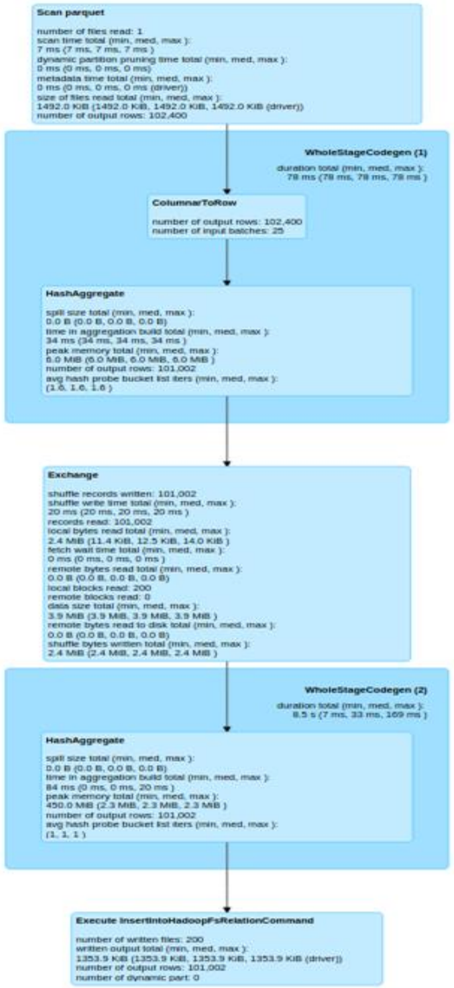


GPU PHYSICAL PLAN



SPARK SQL & DATAFRAME COMPILATION FLOW

CPU PHYSICAL PLAN



Read Parquet File

Convert to Row Format

First Stage Aggregate

Shuffle Exchange

Second Stage Aggregate

Write Parquet File

Read Parquet File

First Stage Aggregate

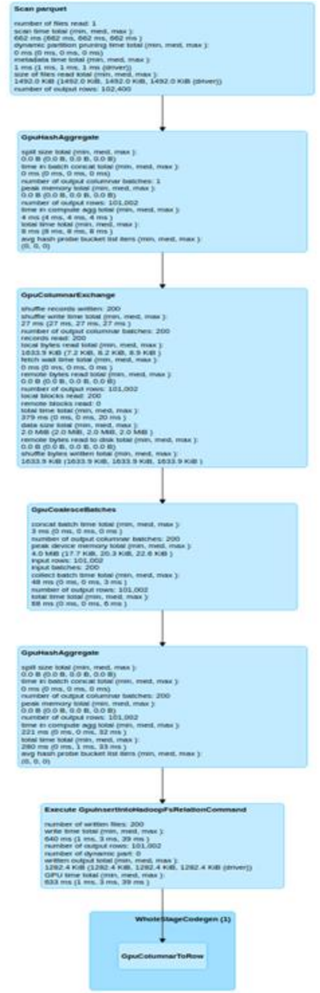
Shuffle Exchange

Combine Shuffle Data

Second Stage Aggregate

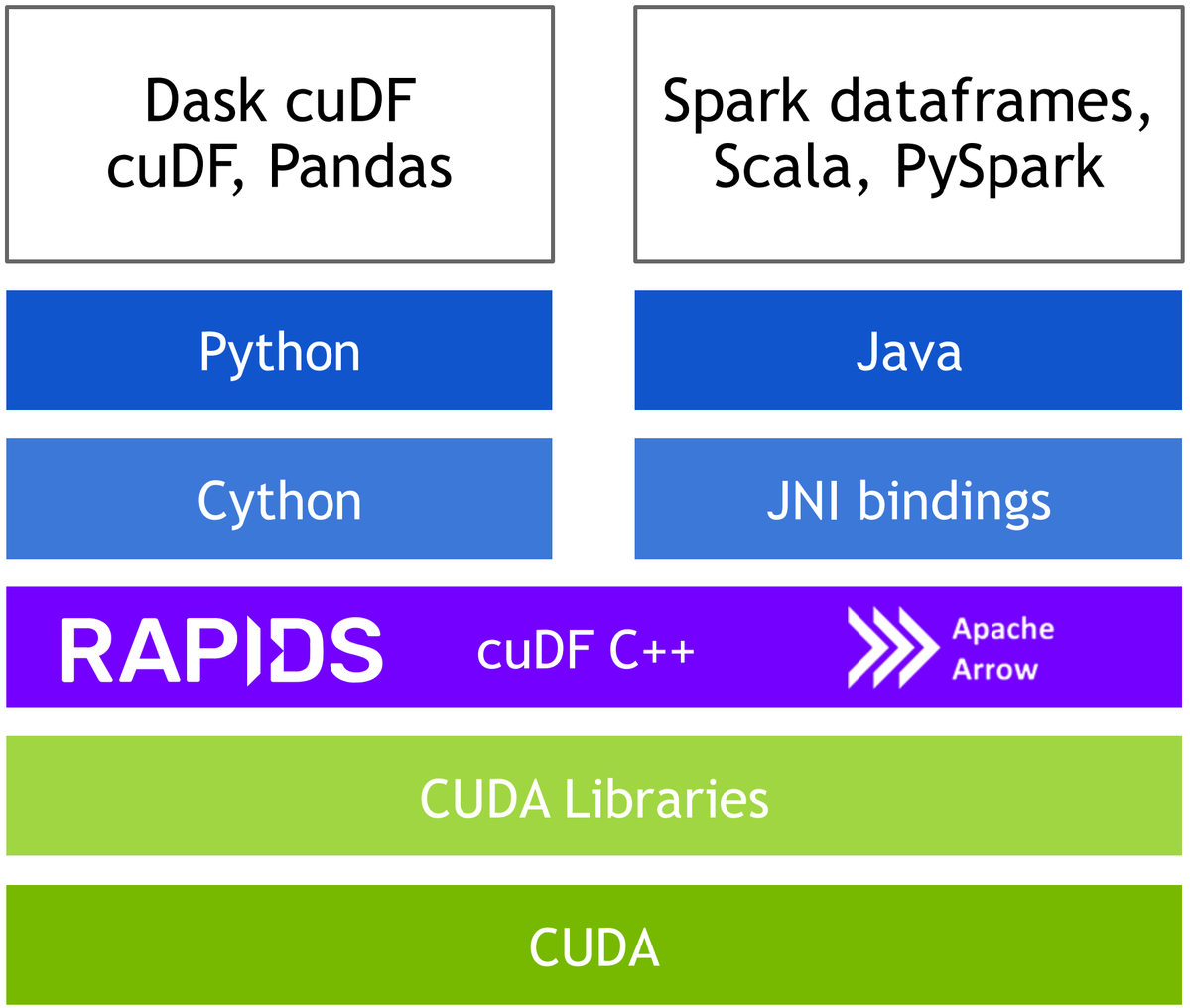
Write Parquet File

Convert to Row Format



GPU PHYSICAL PLAN

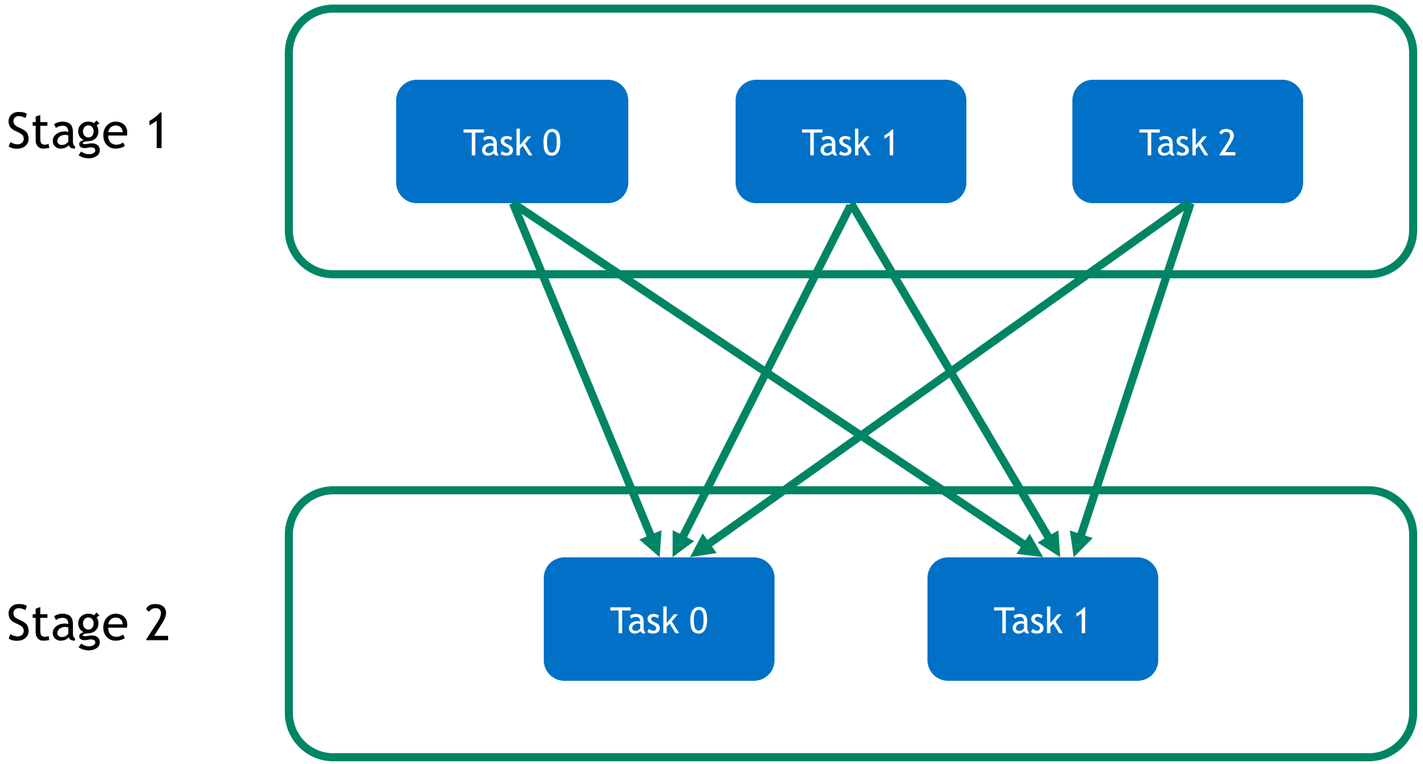
ETL TECHNOLOGY STACK



ACCELERATED SHUFFLE

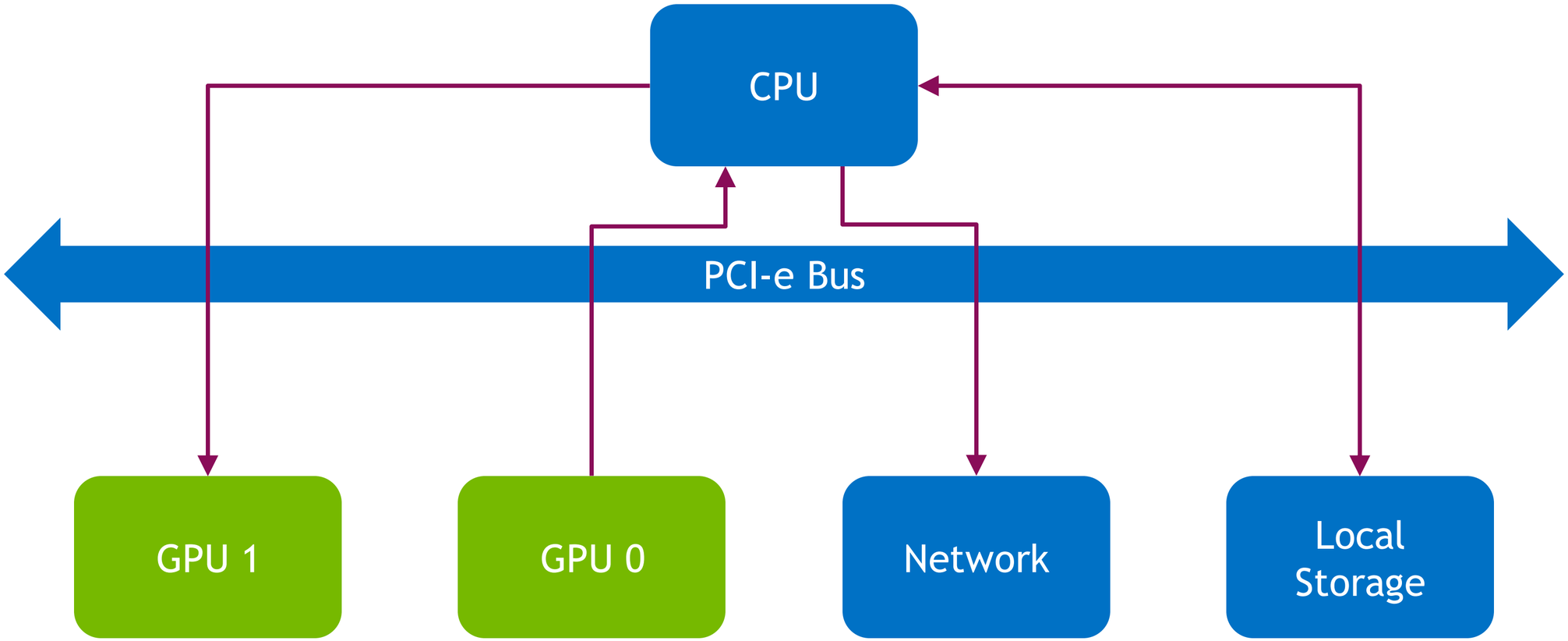
SPARK SHUFFLE

Data Exchange Between Stages



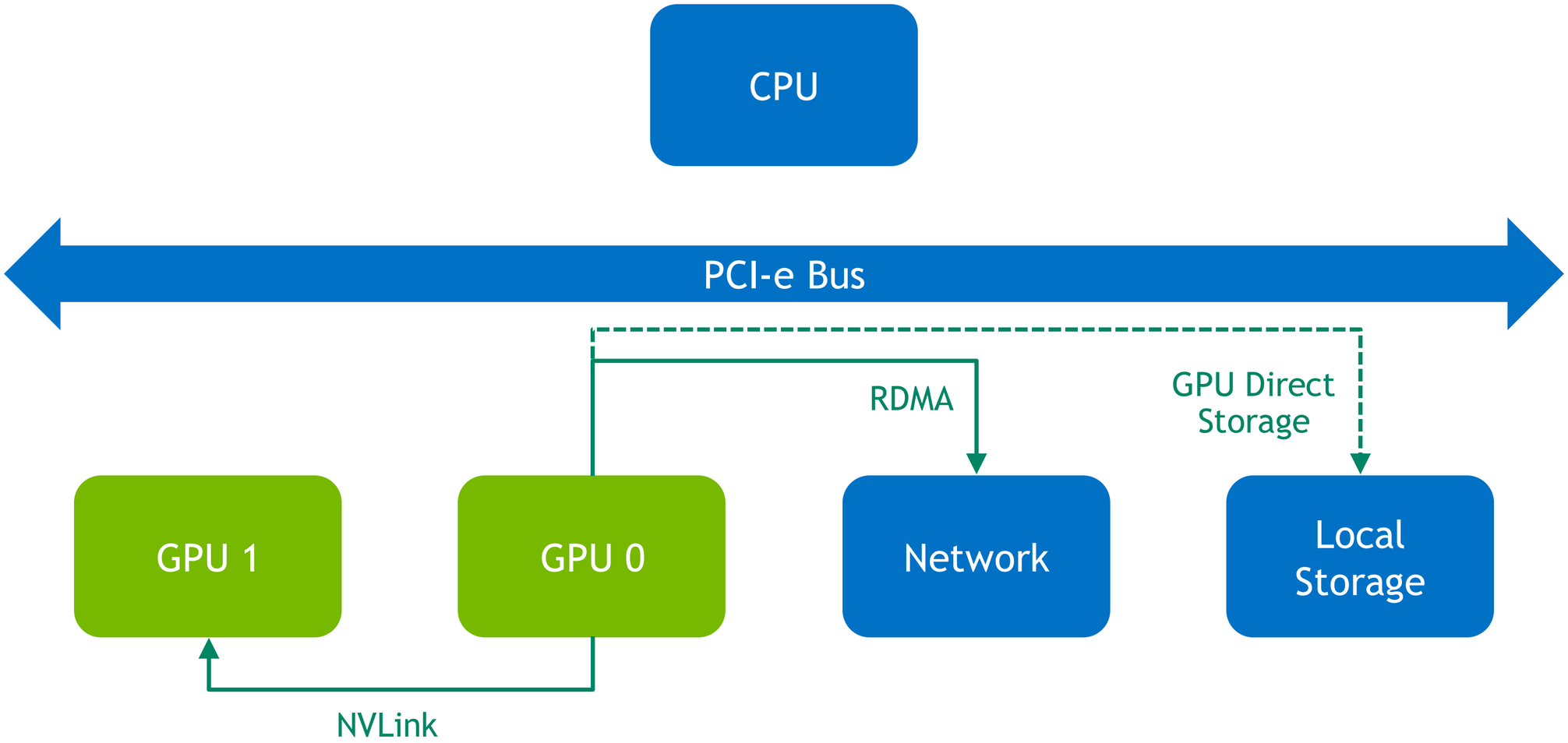
SPARK SHUFFLE

CPU-Centric Data Movement



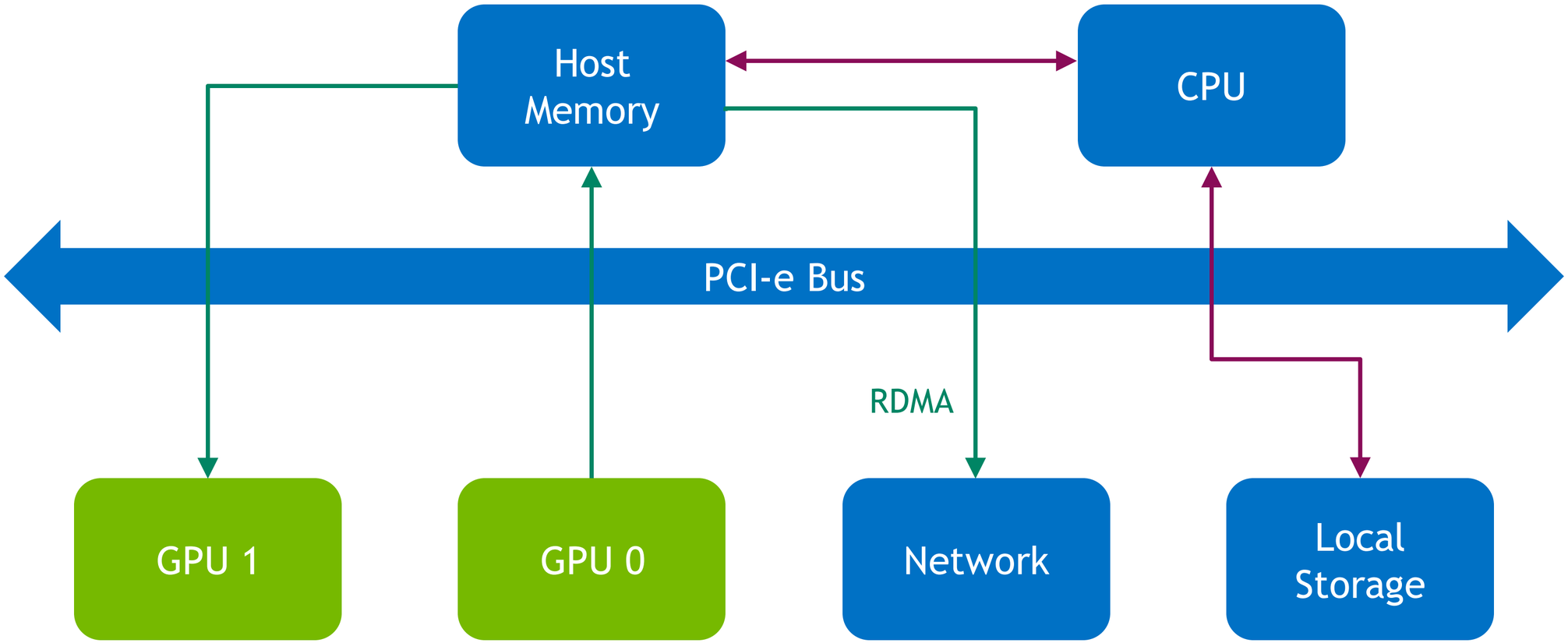
ACCELERATED SPARK SHUFFLE

GPU-Centric Data Movement



ACCELERATED SPARK SHUFFLE

Shuffling Spilled Data



UCX LIBRARY

Unified Communication X

Abstracts communication transports

Selects best available route(s) between endpoints

TCP, RDMA, Shared Memory, GPU

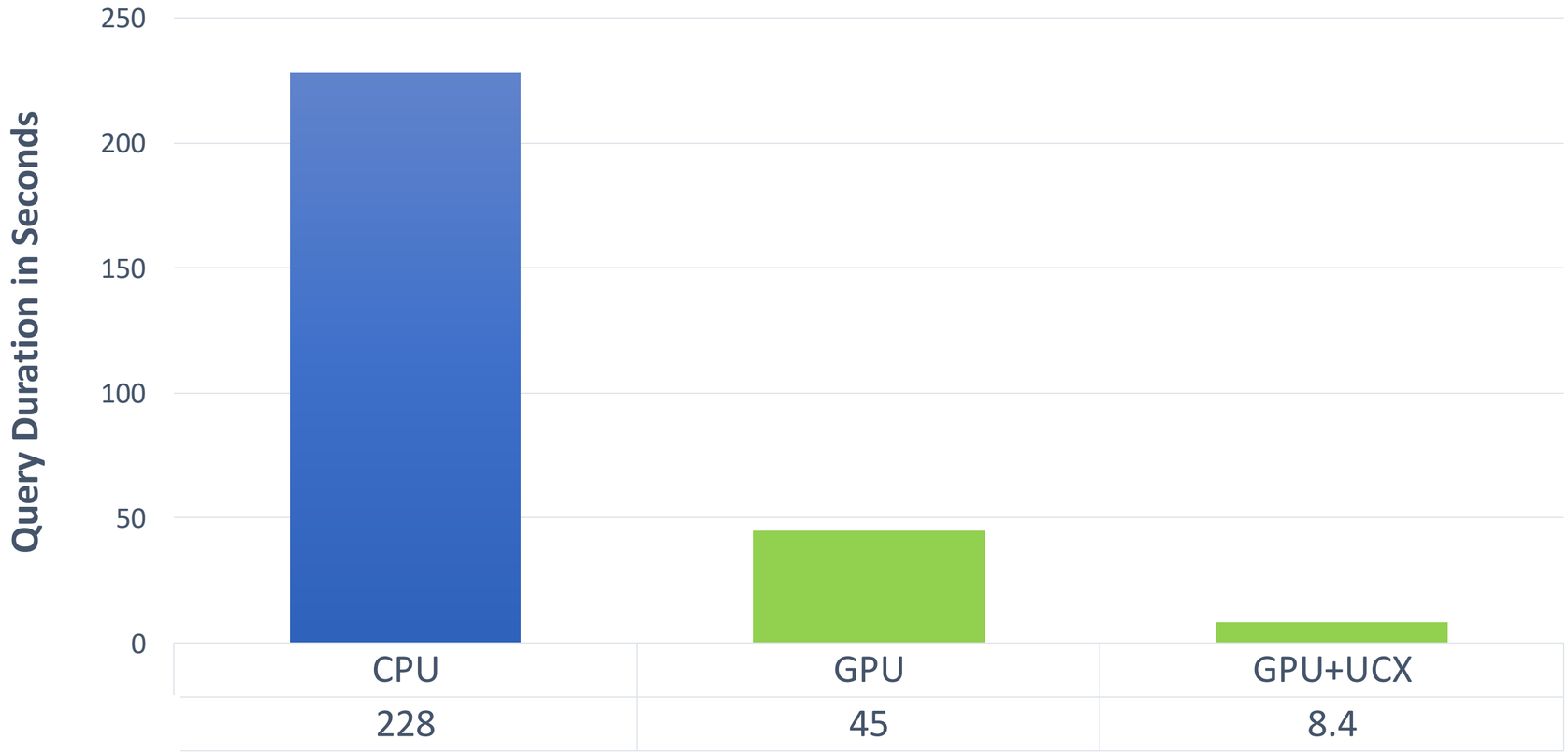
Zero-copy GPU memory transfers over RDMA

RDMA requires network support (IB or RoCE)

http://openucx.org

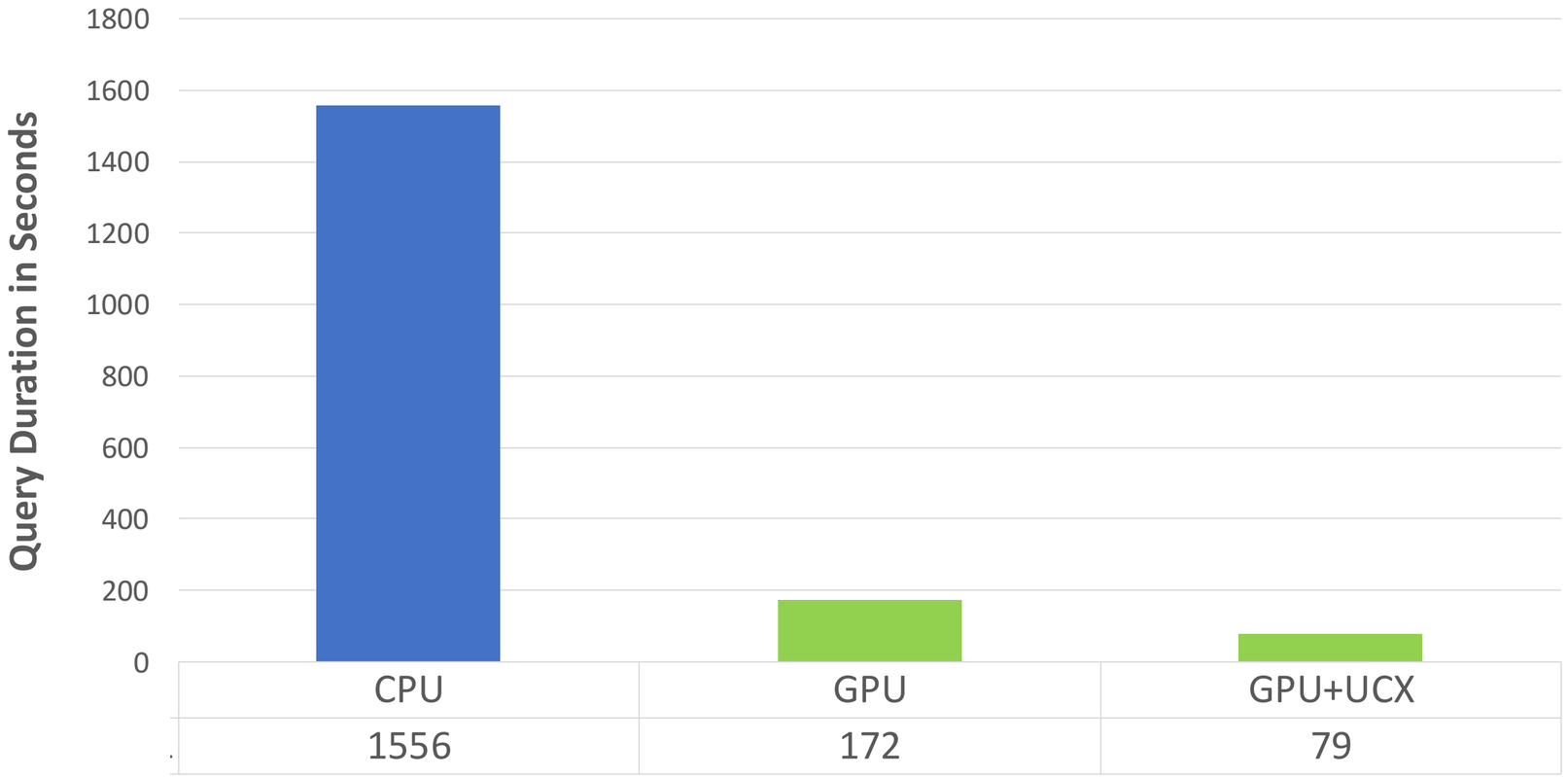
ACCELERATED SHUFFLE RESULTS

Inventory Pricing Query



ACCELERATED SHUFFLE RESULTS

ETL for Logistical Regression Model



WHAT’S NEXT?

WHAT'S NEXT

Open Source/Spark 3.0 Release

Nested types Arrays, Structs, and Maps

Decimal type

More operators

COMING SOON

FURTHER OUT

GPU Direct Storage

Time zone support for timestamps (only UTC for now)

Higher order functions

UDFs

WHERE TO GET MORE INFO

Learn more about the RAPIDS Accelerator for Apache Spark

Visit: NVIDIA.com/Spark

Please use the “contact us” to get in touch with NVIDIA’s Spark team

Listen to how Adobe Email Marketing Intelligent Services leverages the RAPIDS Accelerator & Spark 3.0 on Databricks

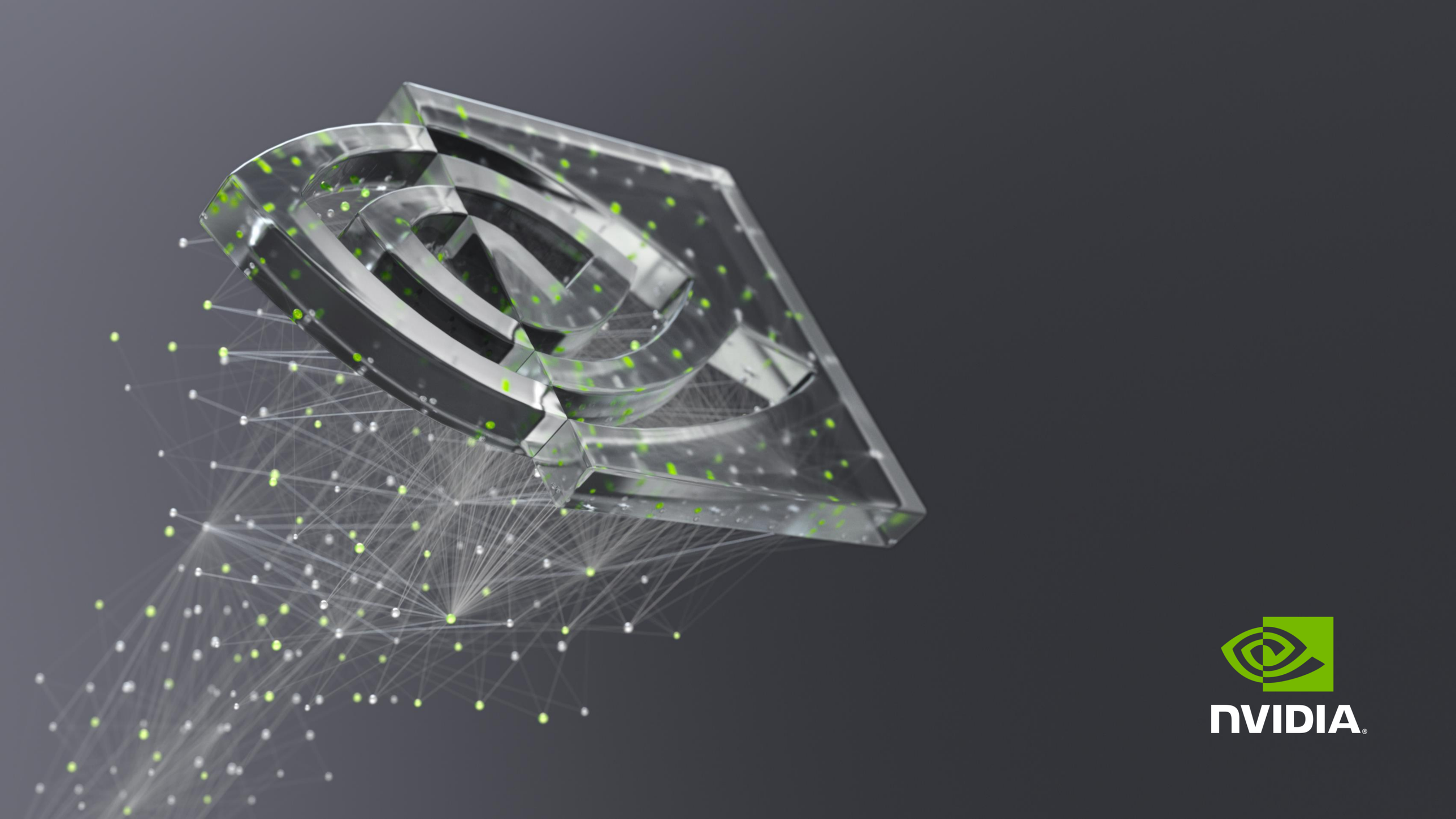
Upcoming Spark+AI Summit Sessions on GPU support for Apache Spark 3.0:

Deep Dive into GPU Support in Apache Spark 3.x

Scalable Acceleration of XGBoost Training on Apache Spark GPU Clusters

Preview of Spark 3.0 GPU Features: NVIDIA.com/Spark-Book

QUESTIONS


NVIDIA

BACKUP SLIDES

FAQS

Q: What are the minimum requirements?

A: The RAPIDS accelerator requires:

Apache Spark 3.0

RAPIDS cudf 0.14

CUDA 10.1 or later

NVIDIA GPU with Pascal architecture or later

Ubuntu 16.04+ or CentOS 7+

FAQS

Q: Do all cluster nodes require GPUs?

A: All Spark executors running with the RAPIDS accelerator require their own GPU.

The Spark driver process does not require a node with a GPU.

Q: Can I run more than one executor per GPU?

A: No, there must be a one-to-one mapping between Spark executors and GPUs.

You can run more than one concurrent task per executor.

FAQS

Q: Will the RAPIDS accelerator work in the cloud?

A: Yes, if the VM environment meets the minimum requirements.

Q: Will the RAPIDS accelerator be available for Apache Spark 2.x?

A: No. The columnar processing APIs added in Apache Spark 3.0 are required.

Q: How can I tell if an operation is being accelerated?

A: Accelerated operations appear in the query explanation and SQL UI.

RAPIDS ACCELERATOR CONFIGURATION

`spark.rapids.sql.enabled` is the master enable

`spark.rapids.sql.explain` enables logging of operations not accelerated

`spark.rapids.sql.concurrentGpuTasks` controls concurrent task count per GPU

SPARK ACCELERATOR-AWARE SCHEDULING

Tracking JIRA: SPARK-24615

Request executor and driver resources (GPU, FPGA, etc.)

Resource discovery

Specify task resources

API to determine assigned resources

YARN, Kubernetes, and Standalone

SPARK ACCELERATOR-AWARE SCHEDULING

Sample Command-Line

```
./bin/spark-shell  --master yarn --executor-cores 2 \
  --conf spark.driver.resource.gpu.amount=1 \
  --conf spark.driver.resource.gpu.discoveryScript=/opt/spark/getGpuResources.sh \
  --conf spark.executor.resource.gpu.amount=2 \
  --conf spark.executor.resource.gpu.discoveryScript=./getGpuResources.sh \
  --conf spark.task.resource.gpu.amount=1 \
  --files examples/src/main/scripts/getGpusResources.sh
```

SPARK STAGE LEVEL SCHEDULING

Tracking JIRA: SPARK-27495

Specify task resource requirements per RDD operation

Dynamically allocates containers to meet resource requirements

Schedules tasks on appropriate containers