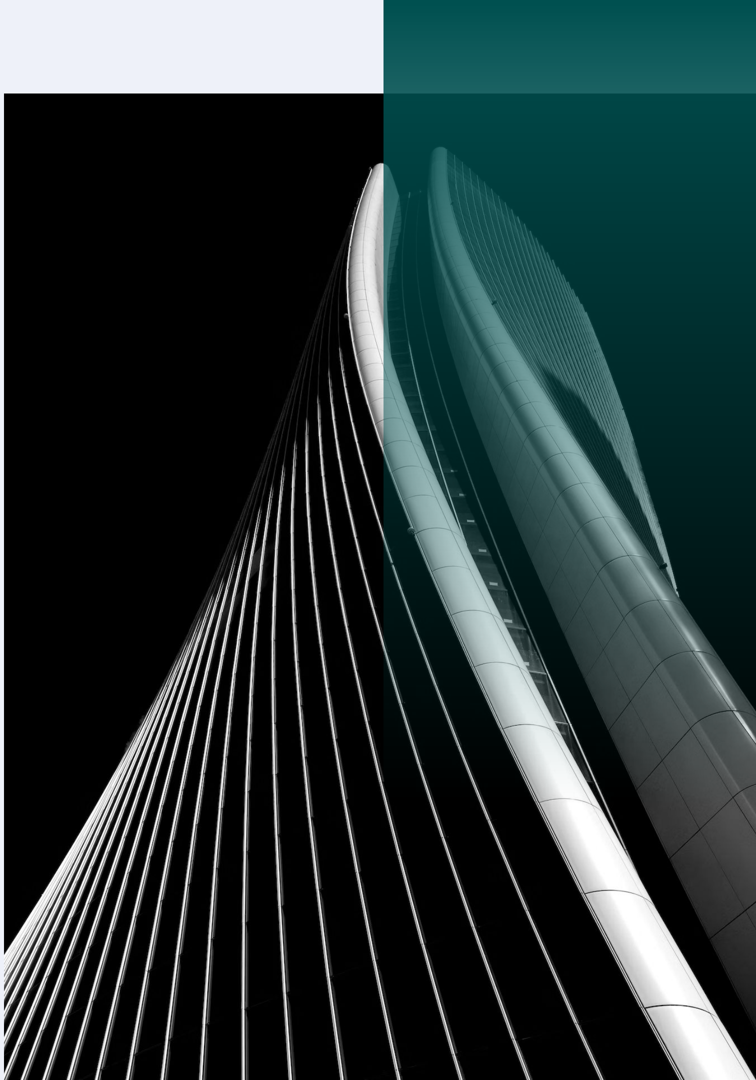
VOLTRON DATA

VOLTRON DATA

May 23rd, 2024

Scaling data processing from CPU to distributed GPUs

By William Malpica,
Director of Engineering and
Co-Founder at Voltron Data



CPUs vs GPUs

CPUs are great at everything!

- … well good enough at most things
- Generic, easy to use, found everywhere and less expensive than GPUs
- Fast and low latency
- Efficient at doing lots of different things
- They are like USPS

Generic Serial Processing

GPUs are even better

- … at specific things….
- GPUs excel at doing the same thing over and over in parallel
- Prefer data to be very structured in vectors and arrays
- High latency, high throughput
- They are like shipping trucking company

Massively Parallel Processing

Data processing... what kind of data processing?

OLTP (Online Transactional Processing)

- Row databases are commonly used for OLTP, where a single “transaction,” such as inserting, removing or updating, can be performed quickly using small amounts of data.
- Each row can hold very different datatypes and each row can be different sizes in memory
- Classic Row databases include MySQL and PosgreSQL
- Classic Row-wise file formats include: CSV, TFRecord file
- Not GPU friendly

A Row of packages?

OLAP (Online Analytical Processing)

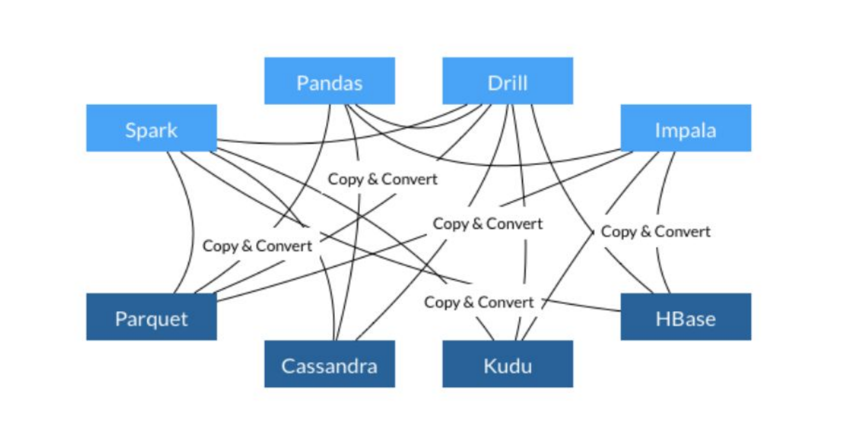
- Column based databases are commonly used for OLAP, where each column is held in contiguous memory
- Column based databases make it awkward to insert or remove data, but it makes it much easier to do analytics.
- Classic Column databases include Snowflake and AWS Redshift.
- Classic Column-wise file formats include: Apache Parquet and Apache ORC
- SIMD and GPU friendly

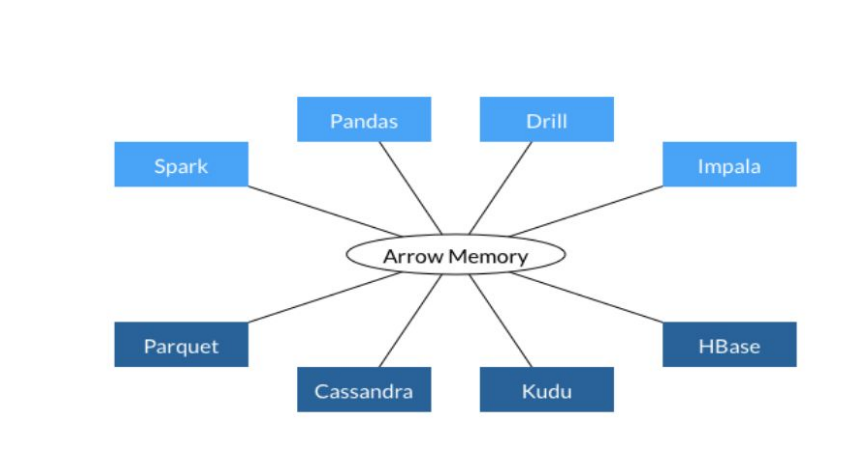
Columns of packages?

Meet Apache Arrow

- It's a columnar memory format
- Its language-independent
- Designed for efficient analytic operations on modern CPUs and GPUs

- Supports zero-copy data transfers
- Its the de-facto standard of memory representation in modern data analytics
- Over 100M downloads a month



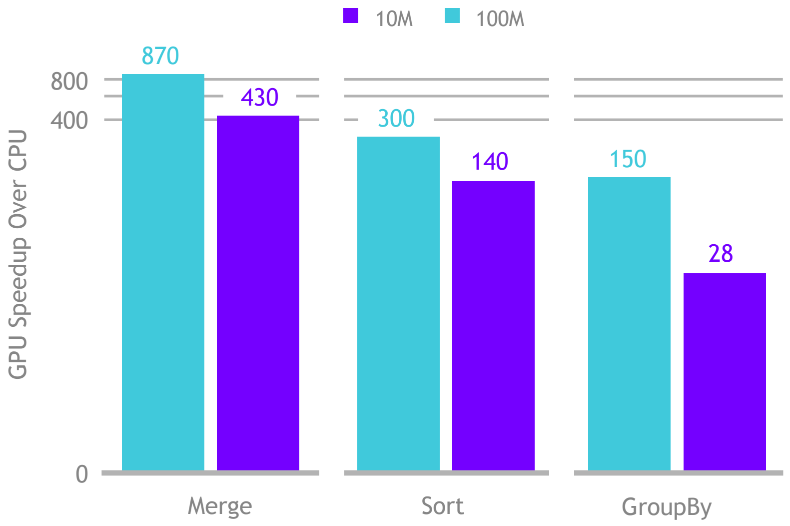


GPUs can accelerate many types of Data Processing

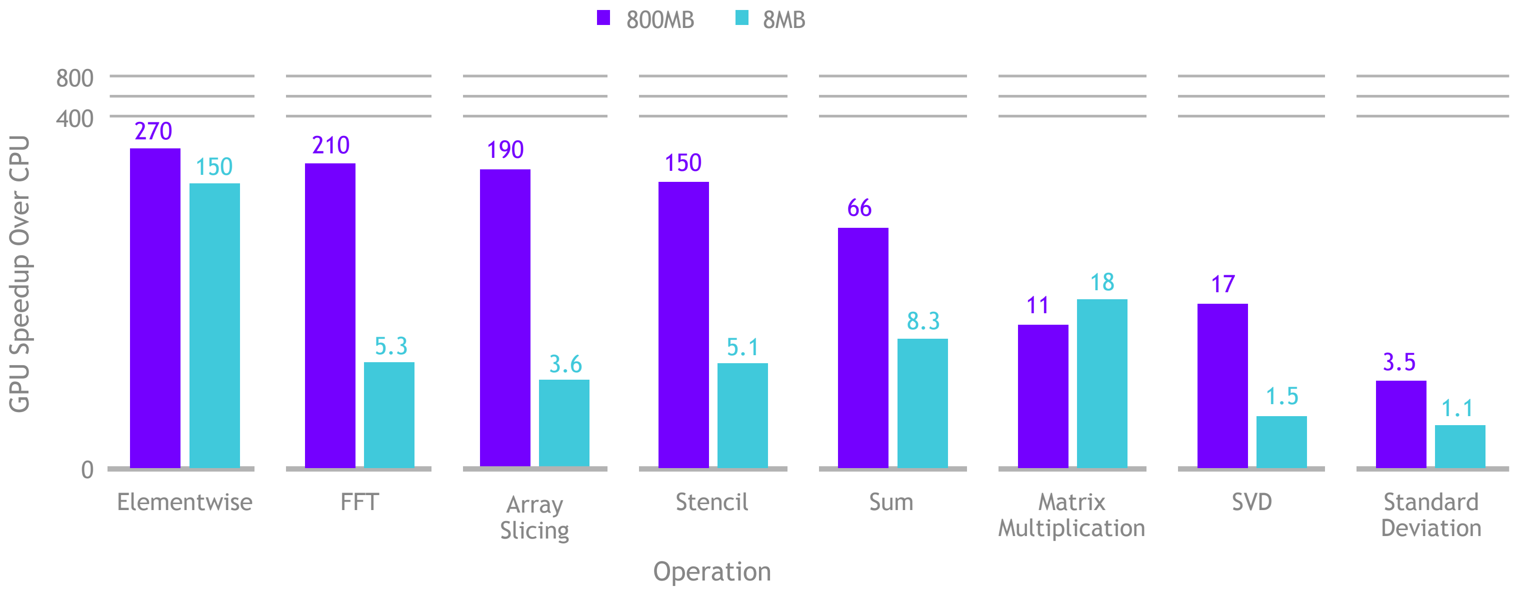
Single GPU DataFrame operations: cuDF vs. Pandas

cuDF v0.10, 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

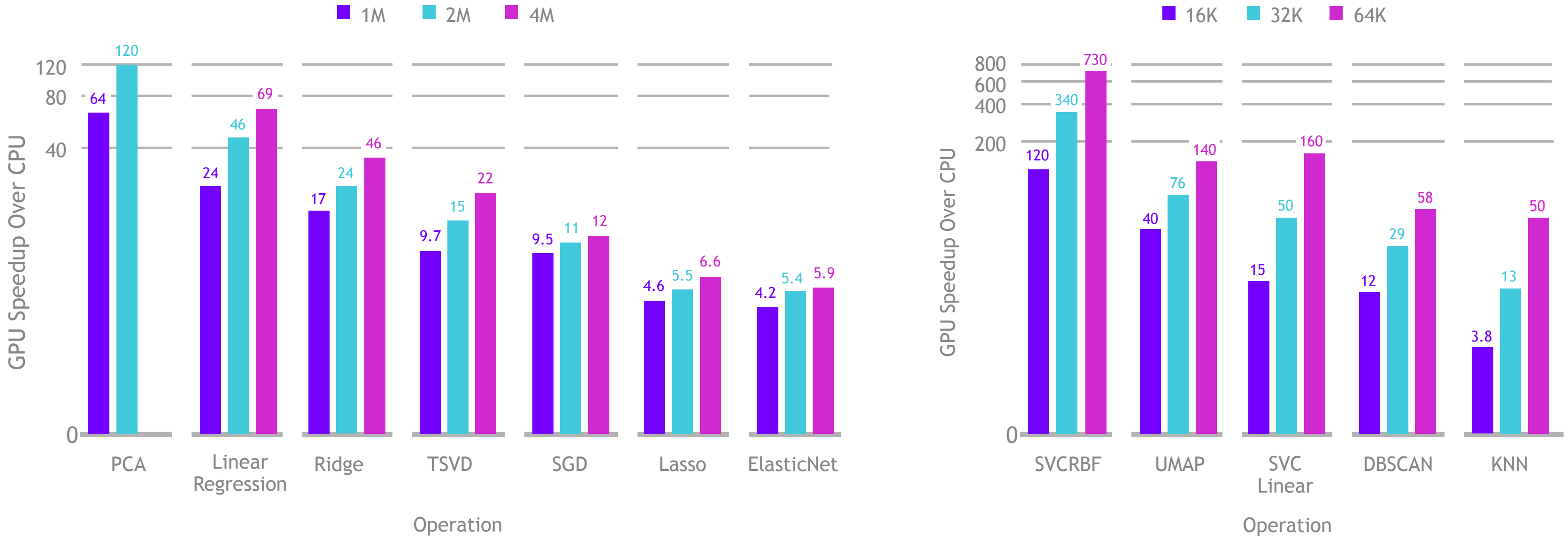


Single GPU Numeric Array operations: CuPy vs NumPy



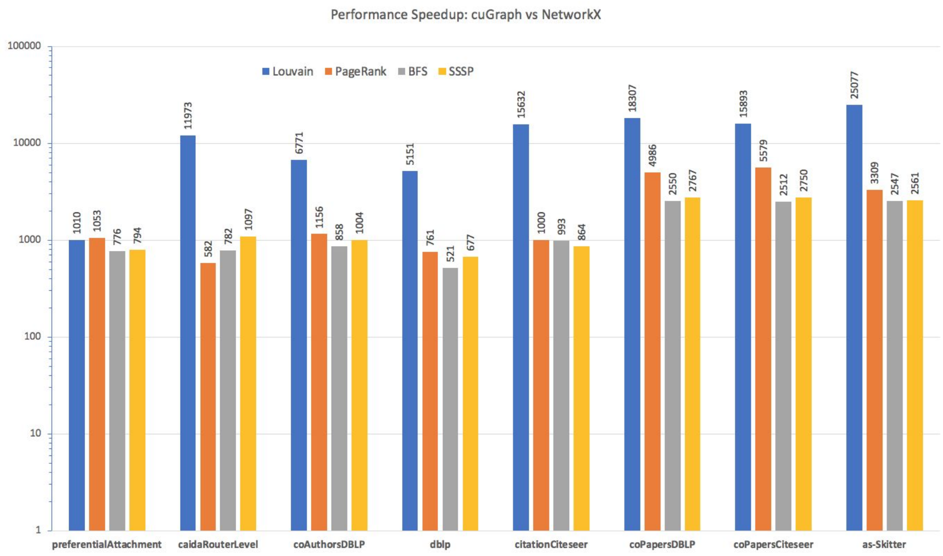
More details: https://blog.dask.org/2019/06/27/single-gpu-cupy-benchmarks

Single GPU Machine Learning: cuML vs Scikit-learn



1x V100 vs. 2x 20 Core CPU

Single GPU Graph Analytics: cuGraph vs NetworkX



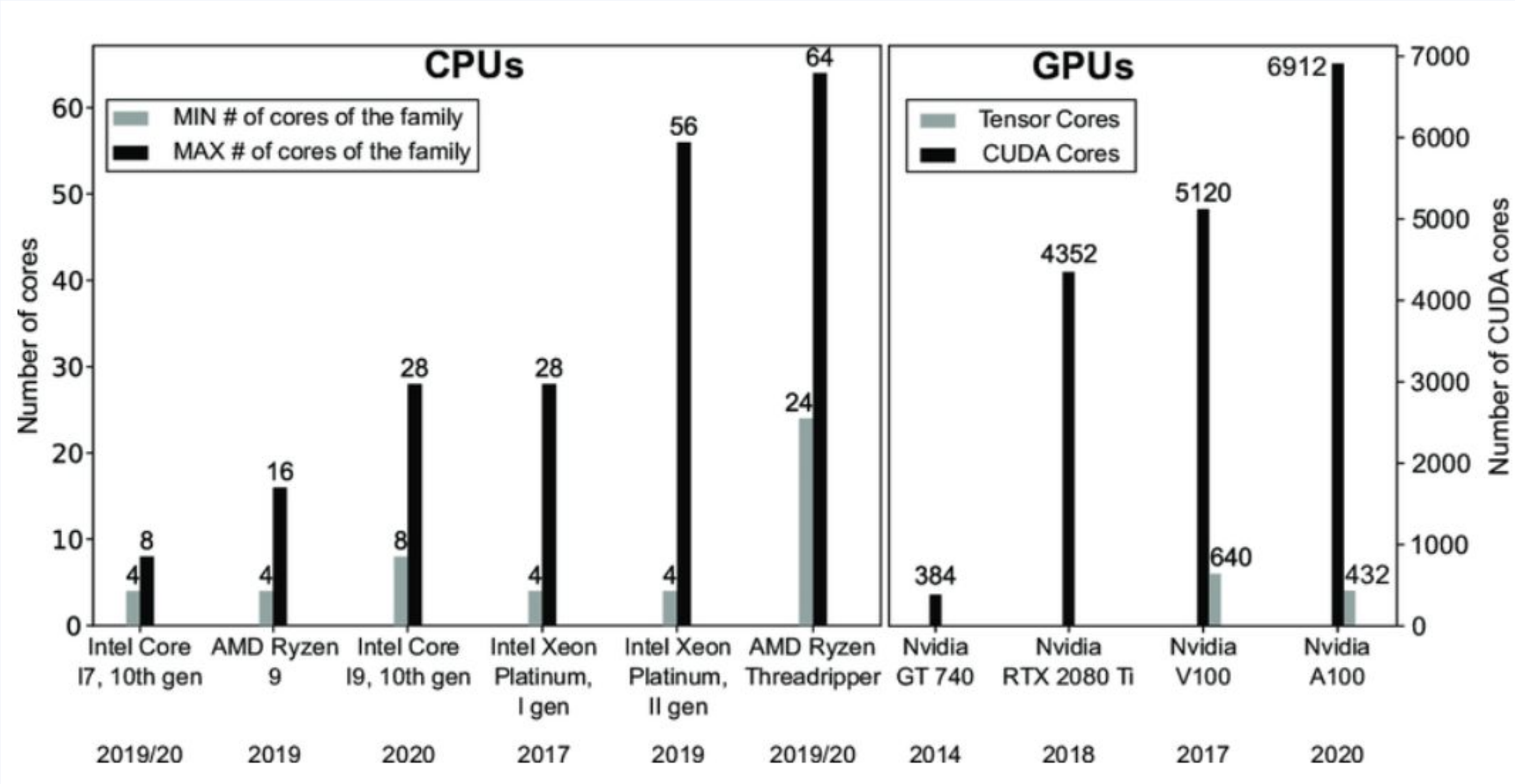
Dataset	Nodes	Edges
preferentialAttachment	100,000	999,970
caidaRouterLevel	192,244	1,218,132
coAuthorsDBLP	299,067	299,067
Dblp-2010	326,186	1,615,400
citationCiteseer	268,495	2,313,294
coPapersDBLP	540,486	20,491,458
coPapersCiteseer	434,102	32,073,440
As-Skitter	1,696,415	22,190,596

Other GPU accelerated libraries aside from RAPIDS:

- **pyTorch**
  - Deep learning
- **ArrayFire**
  - General parallel computing
- **TensorFlow**
  - Machine learning platform
- **Numba**
  - Is a just-in-time compiler for accelerating python code
- **Bend**
  - A massively parallel, high-level programming language.
- **XGBoost**
  - ML library which uses Distributed gradient boosting
- **Voltron Data Theseus SQL Engine**
  - SQL data analytics

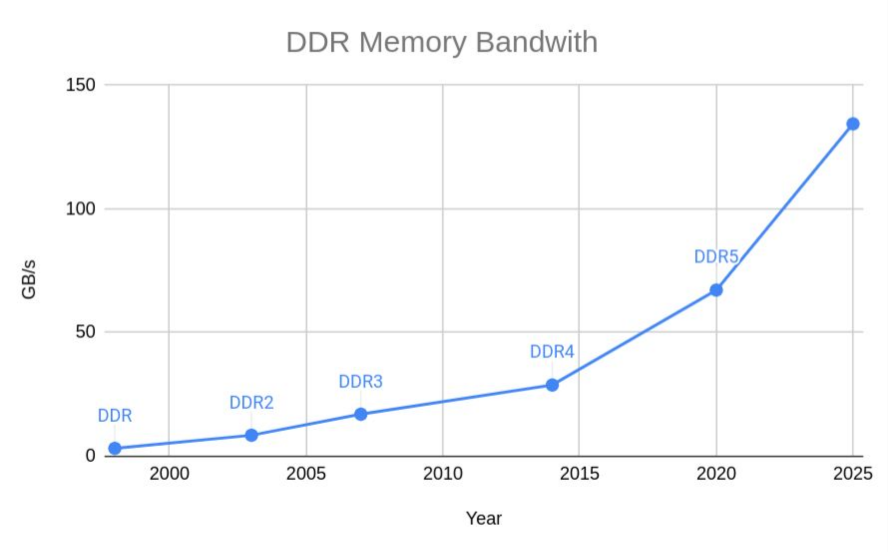
How are GPUs so fast?

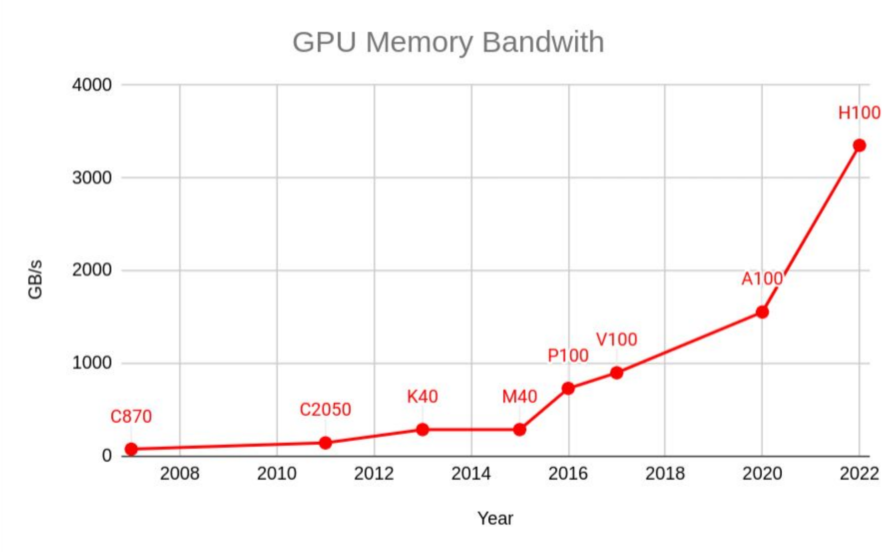
Number of Cores



Graphic from https://www.researchgate.net/publication/342756865_An_Updated_Survey_of_Efficient_Hardware_Architectures_for_Accelerating_Deep_Convolutional_Neural_Networks

Memory Bandwidth





Ok, so GPUs are fast.

Should I always use them?

When GPUs are a good idea?

CPUs might be a better idea:

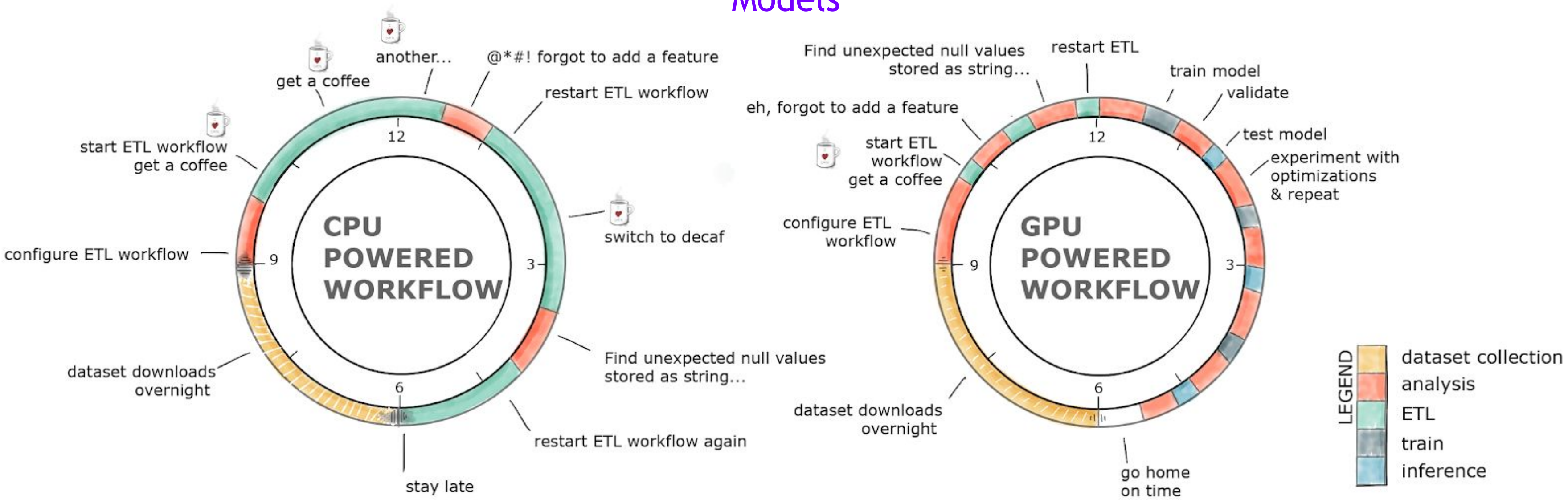
- If the data processing is latency bound
- If the data processing is I/O bound
- If the costs using GPUs outweigh the benefits
- If the amount of data is not too much

GPUs might be a better idea if:

- If you your data processing is compute bound
  - NOTE: When using GPUs you may become I/O bound. It can be hard to feed the beast!
- If compute density is important to you (more on this later)
- If throughput speed enables you to do more
- If you are using GPUs anyways

GPU Powered workflow can reduce iteration time

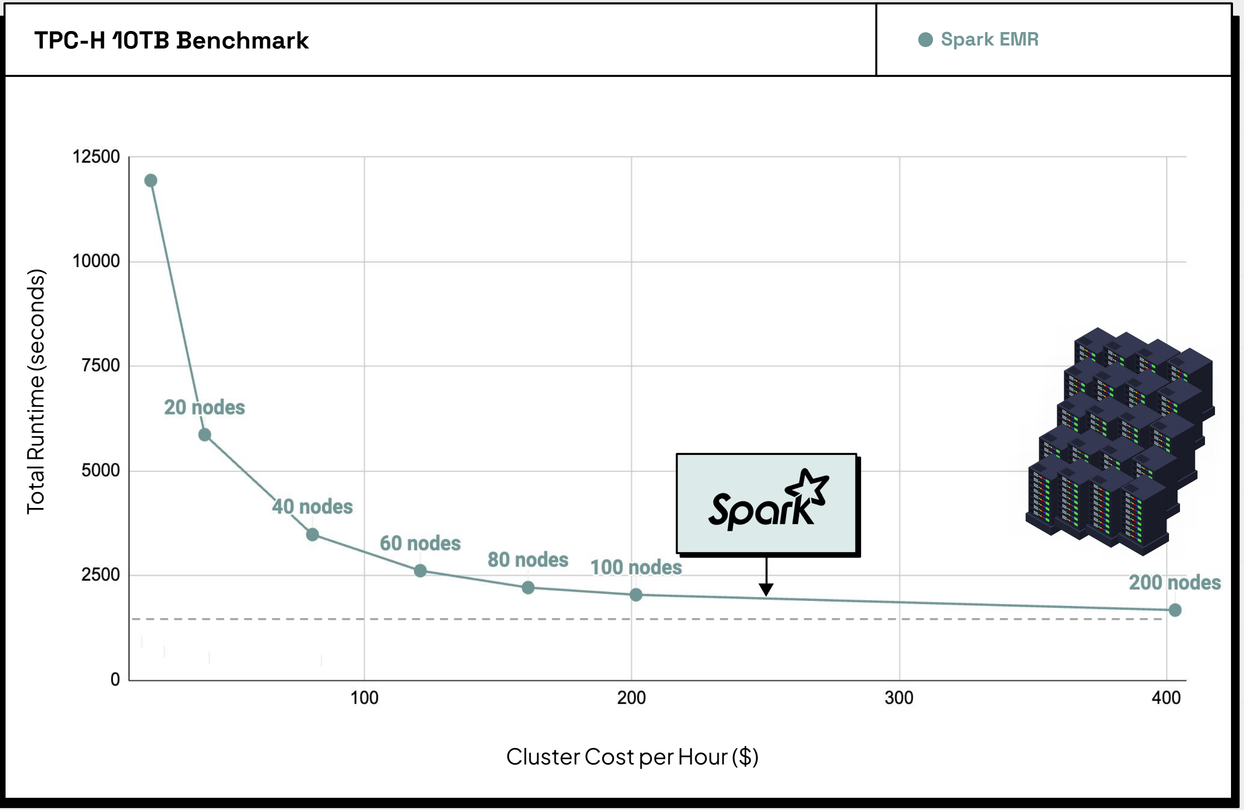
The Average Data Scientist Spends 90+% of Their Time in ETL as Opposed to Training Models



Large data problems usually require distributed solutions

The Wall

CPU performance is capped. No amount of money will jump over this wall.



Note: Theseus: 1 Node 8 x A100 80 GB, Spark: 1 Node r5.8xlarge (AWS) 32 VCPU 32 GB

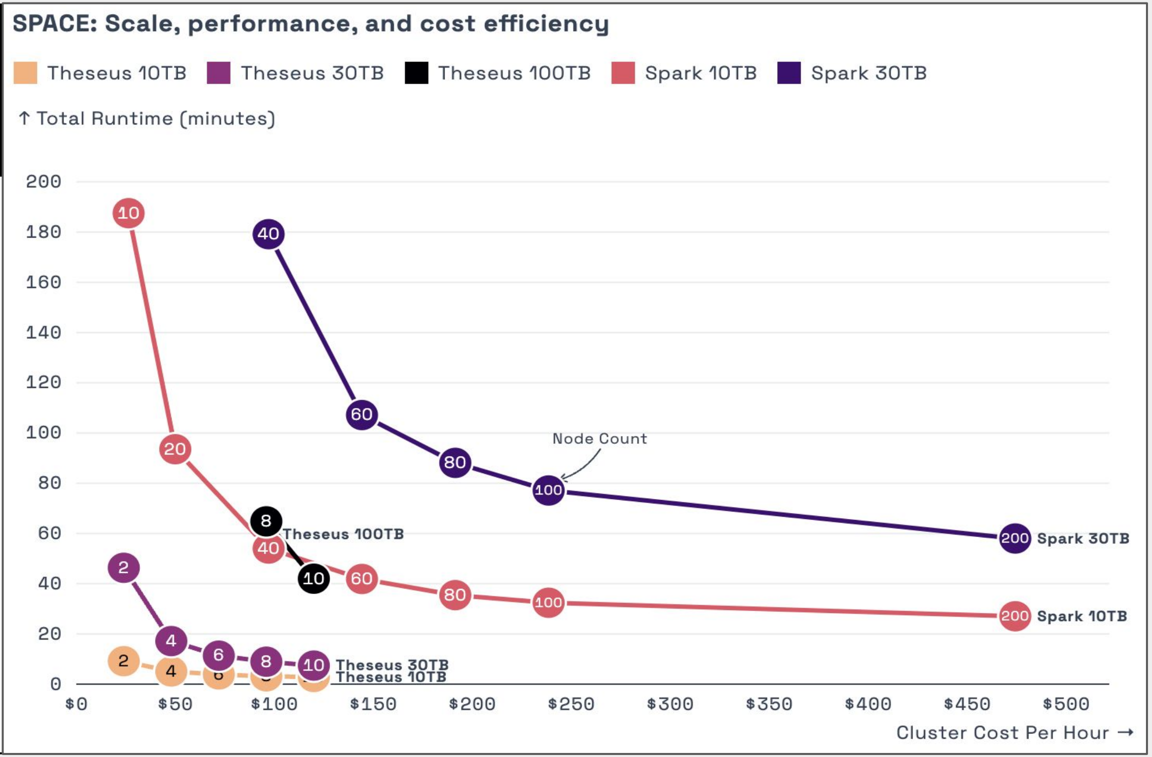
GPUs can help jump over the wall

TPC-H
(10TB, 30TB, 100TB)

- ✓ **Up to 10 DGX Servers**
- ✓ **Parquet Files**
- ✓ **Remote File System**
- ✓ **Lots of Spilling**

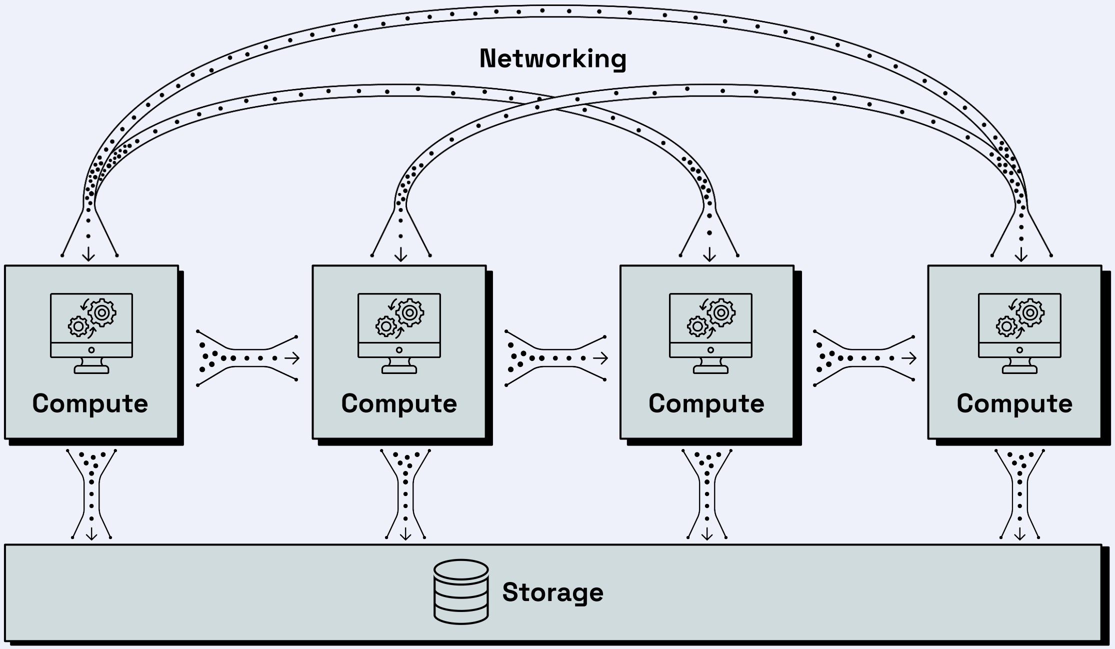
- ✘ **No** Sorting
- ✘ **No** Indexing
- ✘ **No** Caching
- ✘ **No** Warm Up (Cold Queries)

Note: Theseus: 1 Node 8 x A100 80 GB, Spark: 1 Node r5.8xlarge (AWS) 32 VCPU 32 GB



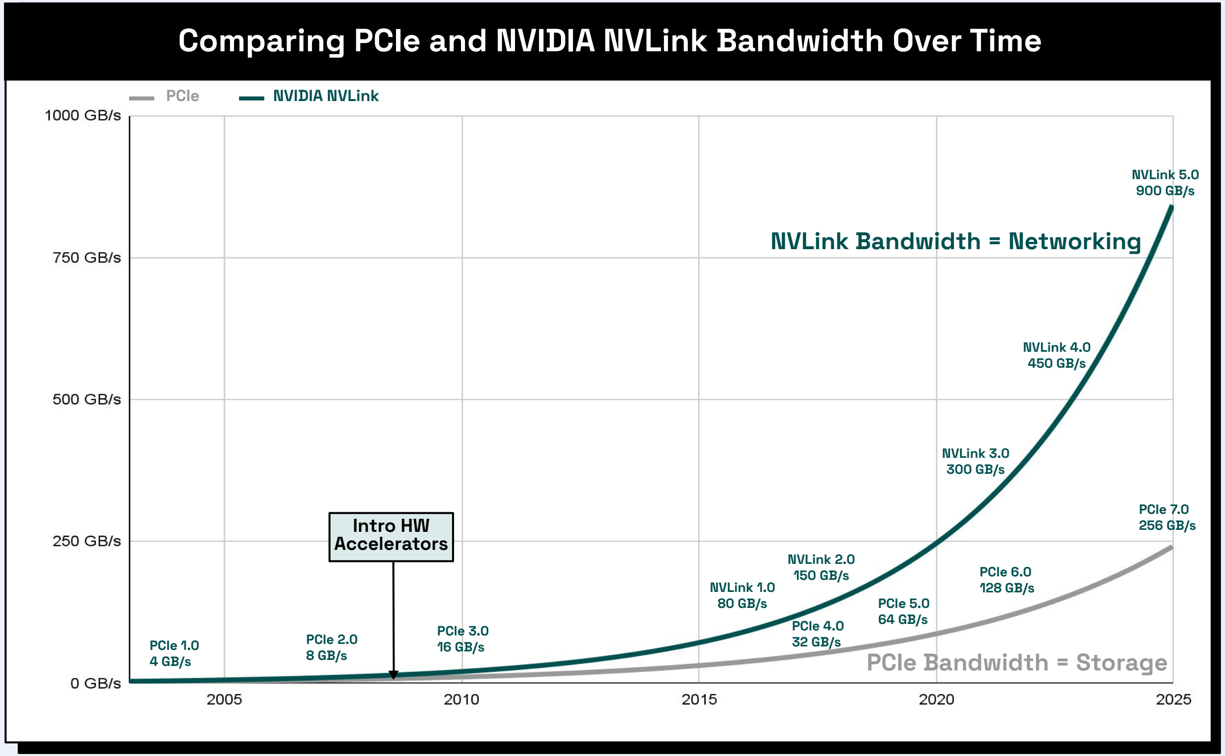
New Bottleneck

Speeding up the compute just moves the bottleneck elsewhere, Networking and Storage



Networking and Storage

Need to be aware of the hardware when architecting the software



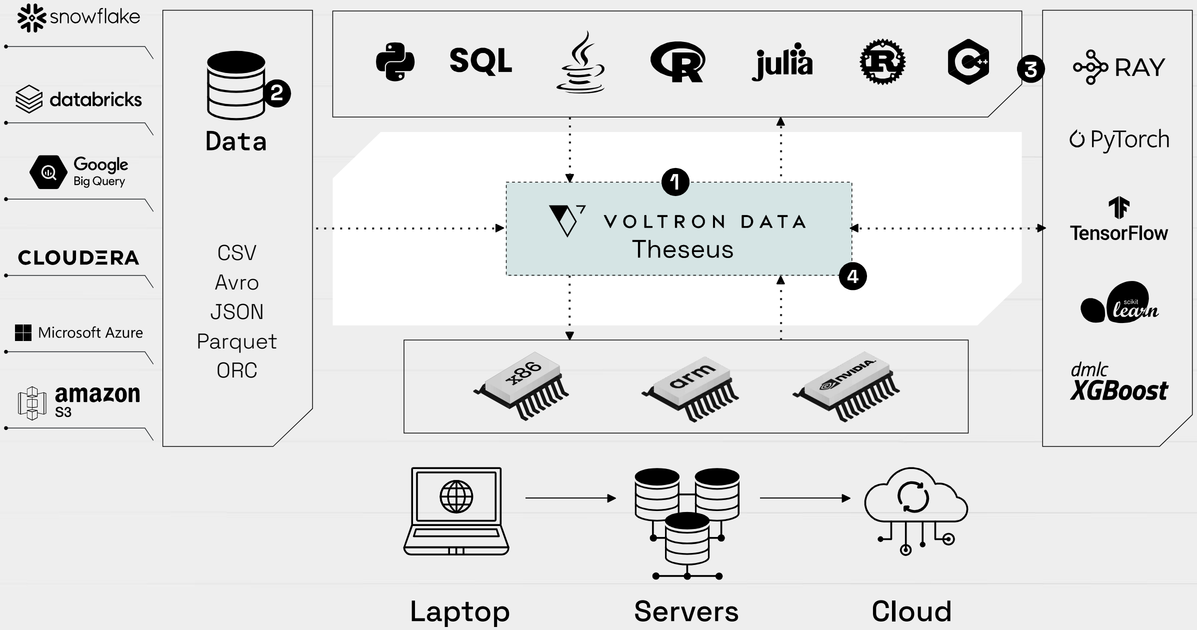
How to get the most performance on your distributed GPU data processing system?

- Use GPUDirect RDMA to communicate with Infiniband or RoCE
- Use GPUDirect Storage to write to local or remote storage, such as NVMe or NVMe over Fabric (NVMe-oF)
- It avoids extra copies through a bounce buffer in the CPU’s memory, enabling a direct memory access (DMA) to move data on a direct path into or out of GPU memory — all without burdening the CPU.

- Need to use distributed systems which can support these types of technologies:
  - Dask: can be configured for GPUDirect RDMA with OpenUCX
  - Spark Rapids: can use GPUDirect RDMS and GPUDirect Storage
  - MPI: can be configured for GPUDirect RDMA
  - Theseus: will use GPUDirect RDMS and GPUDirect Storage when available

Voltron Data Theseus

A Compute Mesh unifying hardware, languages, and applications



1. **Accelerator-Native:** Distributed query engine built from the ground up to take advantage of full system hardware acceleration.
2. **Petabyte Scale:** Focusing on problems too big and time sensitive for Spark
3. **Composable:** Built on open source standards that enables interoperability from storage to application
4. **Evolutionary:** A composable engine that seamlessly adapts to new hardware and languages

Thank You!

Questions?