Fundamentals of Accelerated Data Science

NVIDIA

Workshop Overview

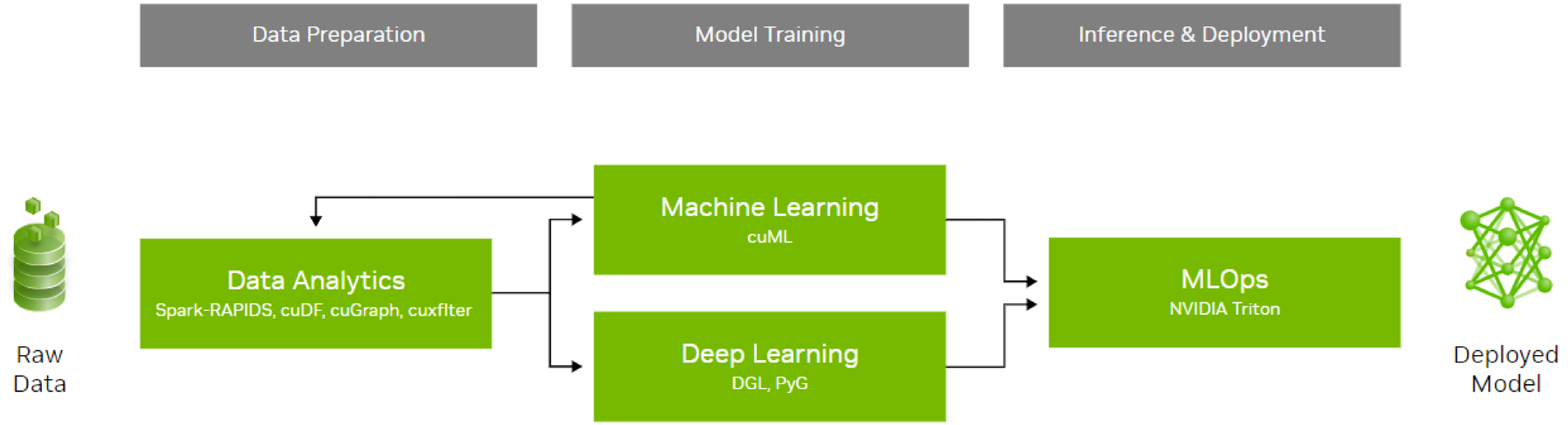
An introduction to data science with a focus on speed and efficiency

Learning objectives:

- Understand the fundamental concepts of data science and parallel computing
- Explore practical examples of accelerated data science pipelines
- Examine the methods used to achieve acceleration in data science and discuss their broader implications

This workshop will not cover statistical analysis, neural networks, and distributed computing

	Workshop Outline
Task 1	Data science overview Data manipulation
Task 2	Graph analytics
Task 3	Machine Learning
Coding Assessment	Biodefense



Section 1 Agenda

- Data Science Fundamentals
- GPU Computing & RAPIDS
- Accelerating and Scaling Data Manipulation
- Data Visualization
- Interoperability
- Hands-On Lab

Data Science Is the Key to Unlocking the Future of Modern Business

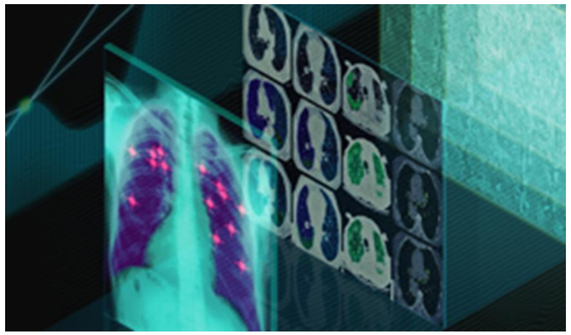
Enables data-driven decision making

Retail

Supply Chain & Inventory Management
Price Management / Markdown Optimization
Promotion Prioritization And Ad Targeting

Telecom

Detect Network/Security Anomalies
Forecasting Network Performance
Network Resource Optimization (SON)



Healthcare

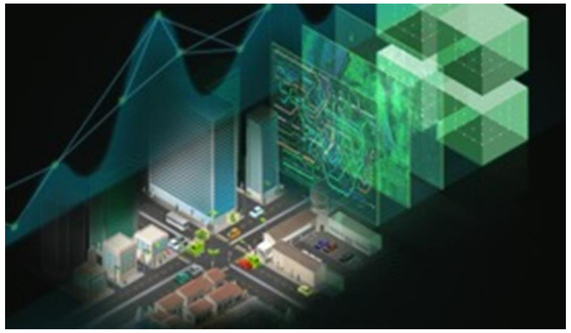
Improve Clinical Care
Drive Operational Efficiency
Speed Up Drug Discovery

Manufacturing

Remaining Useful Life Estimation
Failure Prediction
Demand Forecasting

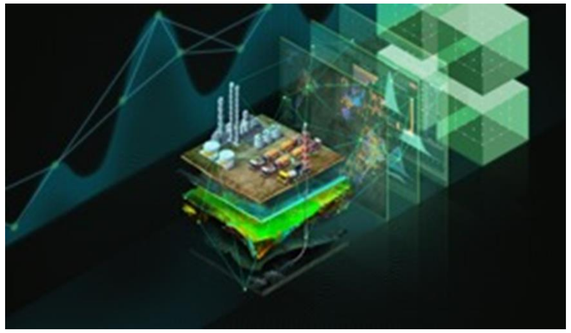
Financial Services

Claim Fraud
Customer Service Chatbots/Routing
Risk Evaluation



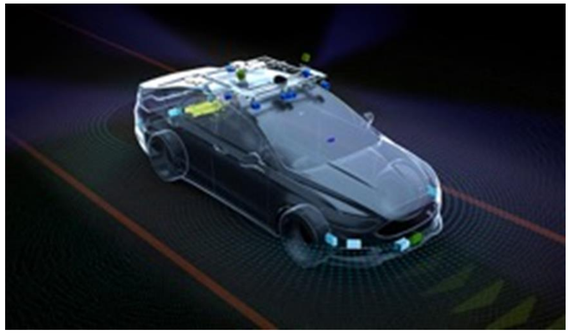
Consumer Internet

Ad Personalization
Click Through Rate Optimization
Churn Reduction



Oil & Gas

Sensor Data Tag Mapping
Anomaly Detection
Robust Fault Prediction



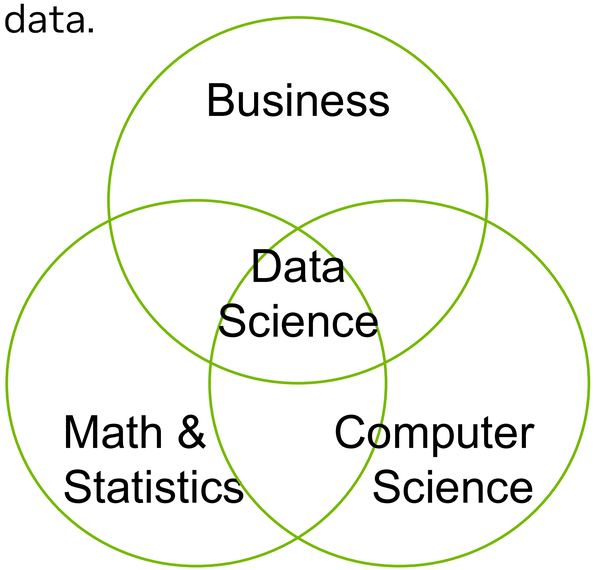
Automotive

Personalization & Intelligent Customer Interactions
Connected Vehicle Predictive Maintenance
Forecasting, Demand, & Capacity Planning

Data Science Overview

The study of data to extract meaningful insights

- Data science is a multidisciplinary field that combines principles and practices from the fields of mathematics, statistics, artificial intelligence, and computer engineering to analyze large amounts of data.



- Helps to ask and answer questions like:
  - Descriptive Analysis (what happened)
  - Diagnostic Analysis (why it happened)
  - Predictive Analysis (what will happen)
  - Prescriptive Analysis (what can be done with the results)

- More than just quantitative analysis - it combines knowledge of tools, methods, and technologies to generate meaning from data

Data Science Is More Than Machine Learning and Analytics

Non-analytical tasks related to data science are equally important

Analytical tasks include:

- Machine learning model development
- Measuring key metrics
- Statistical analysis
- Data visualizations

Non-analytical tasks (**upstream**) include:

- **Extract, transform, and load (ETL)**
  - Data collection
  - Data cleaning and preprocessing
  - Data modeling
- **Data engineering**

Data Science Is More Than Machine Learning and Analytics

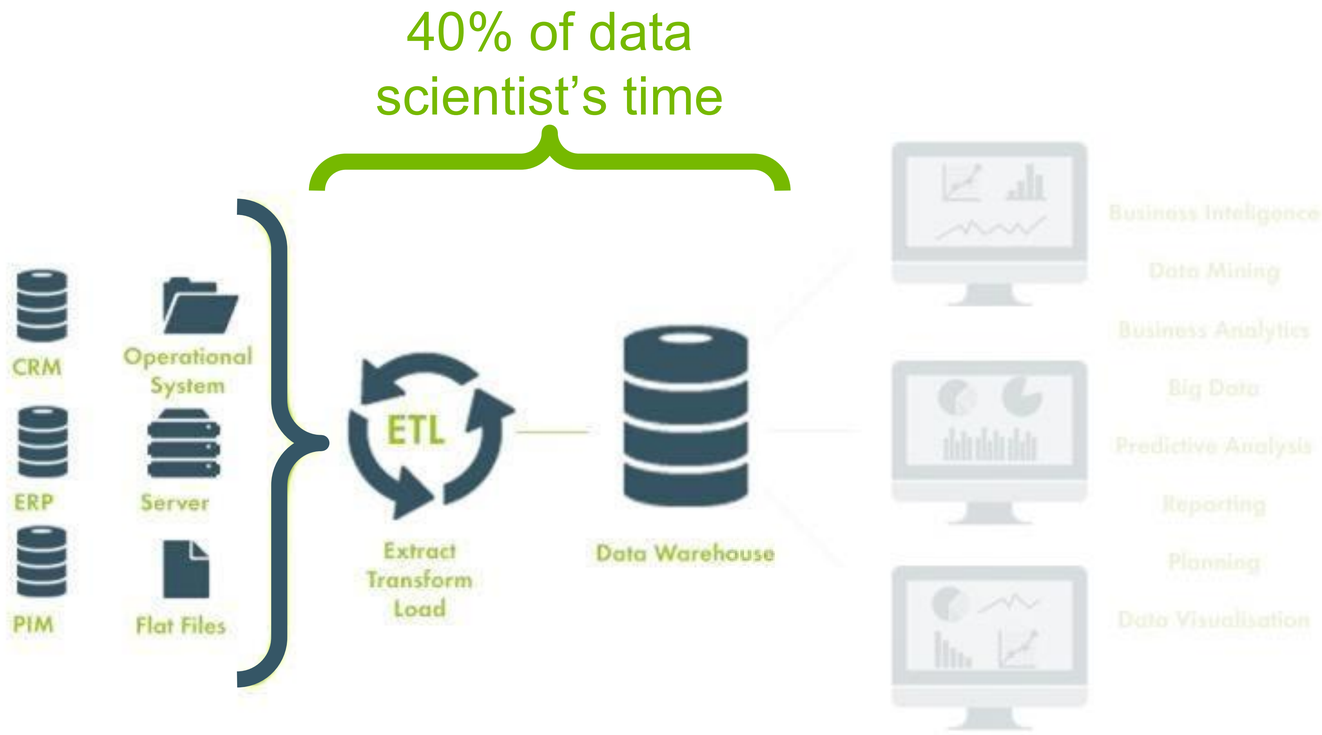
Non-analytical tasks related to data science are equally important

Analytical tasks include:

- Machine learning model development
- Measuring key metrics
- Statistical analysis
- Data visualizations

Non-analytical tasks (**upstream**) include:

- **Extract, transform, and load (ETL)**
  - Data collection
  - Data cleaning and preprocessing
  - Data modeling
- **Data engineering**



Challenges in Data Science Today

What needs to be solved to empower data scientists

INCREASING BIG DATA USE CASES

Data sets are continuing to dramatically increase in size

Multitude of sources

Different data formats with varying quality

SLOW CPU PROCESSING

End of Moore's law, CPUs aren't getting faster

Many popular data science tools are CPU-only

CPUs are not designed to parallelize process well

COMPLEX SOFTWARE MANAGEMENT

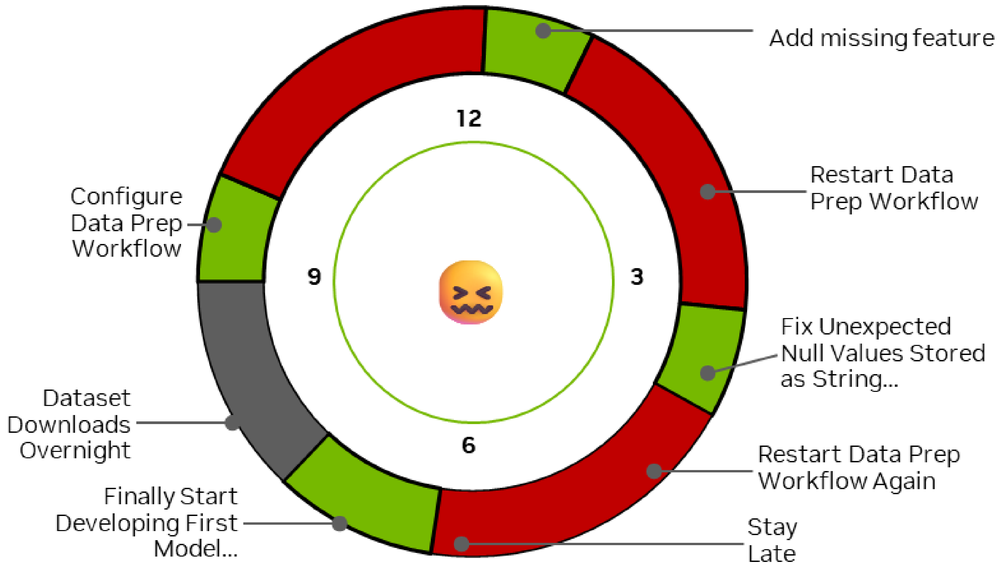
Time consuming to install software

Nearly impossible to manage all version conflicts

Updates often break other software

Less Waiting. More Building.

The right solution enables data scientists to focus on delivering value



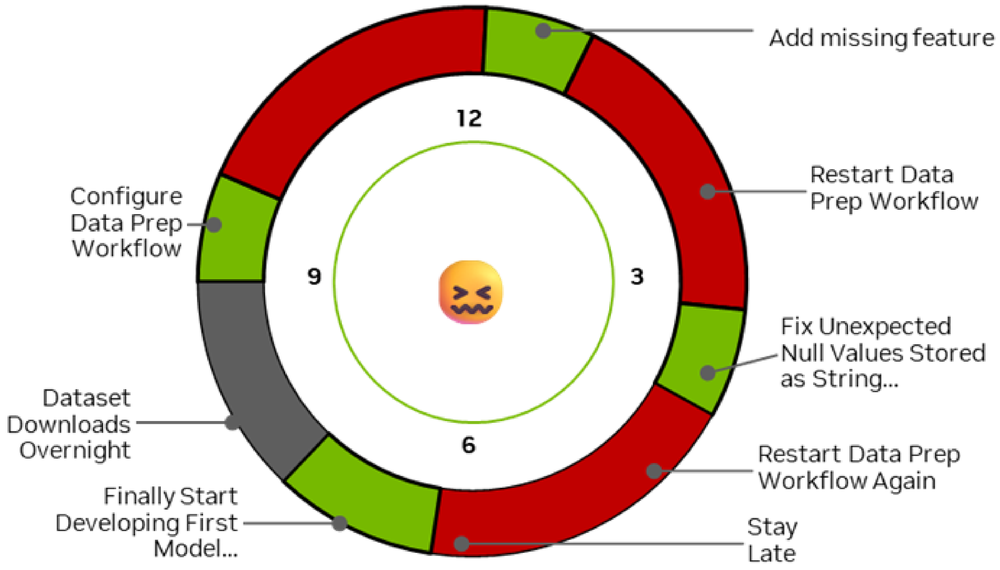
CPU WORKFLOW

ACCELERATED COMPUTING WORKFLOW

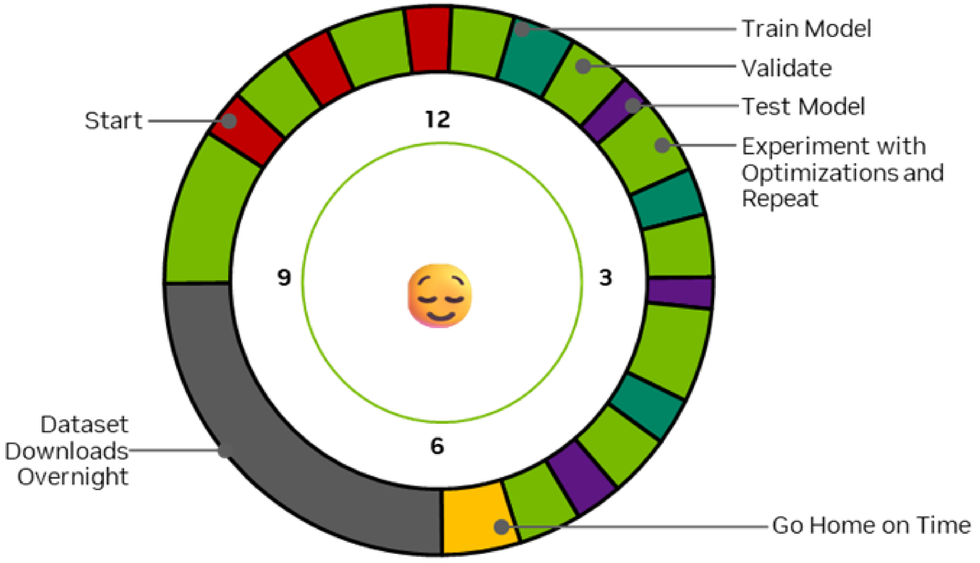
Waiting Building

Less Waiting. More Building.

The right solution enables data scientists to focus on delivering value



CPU WORKFLOW

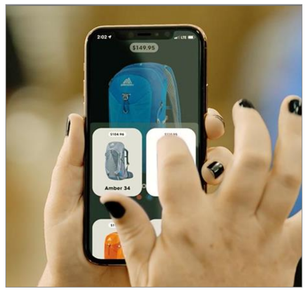


ACCELERATED COMPUTING WORKFLOW

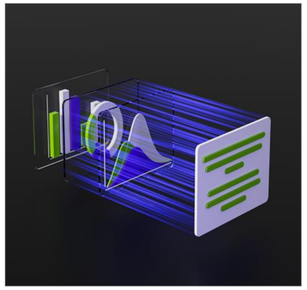
Waiting Building

Modern Applications Need Accelerated Computing

Petabyte scale data | Massive models | Real-time performance



Recommenders

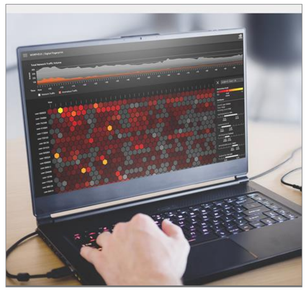


LLMs

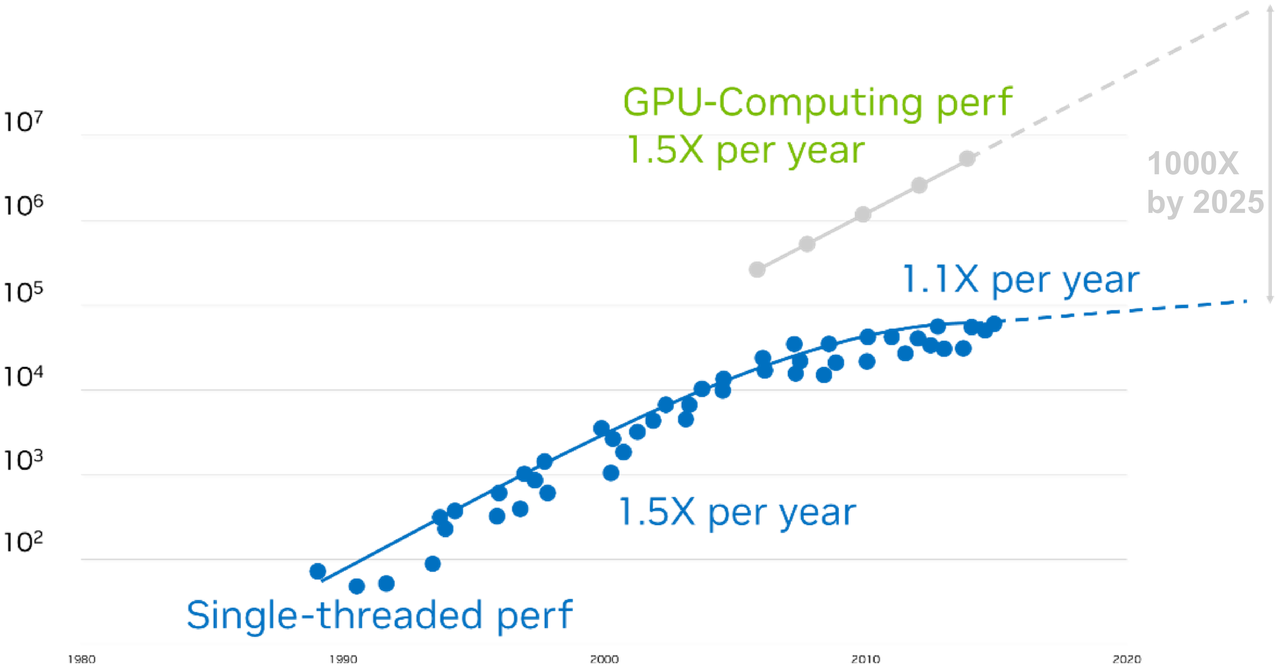
Forecasting

Fraud Detection

Genomic Analysis



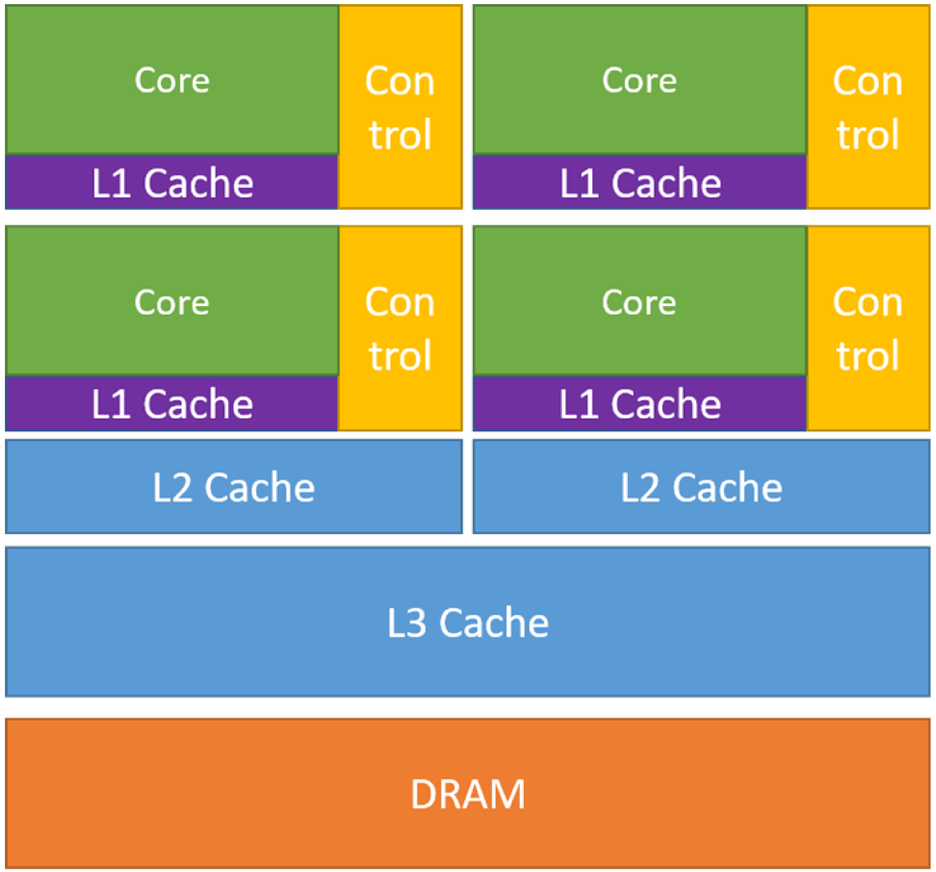
Cybersecurity

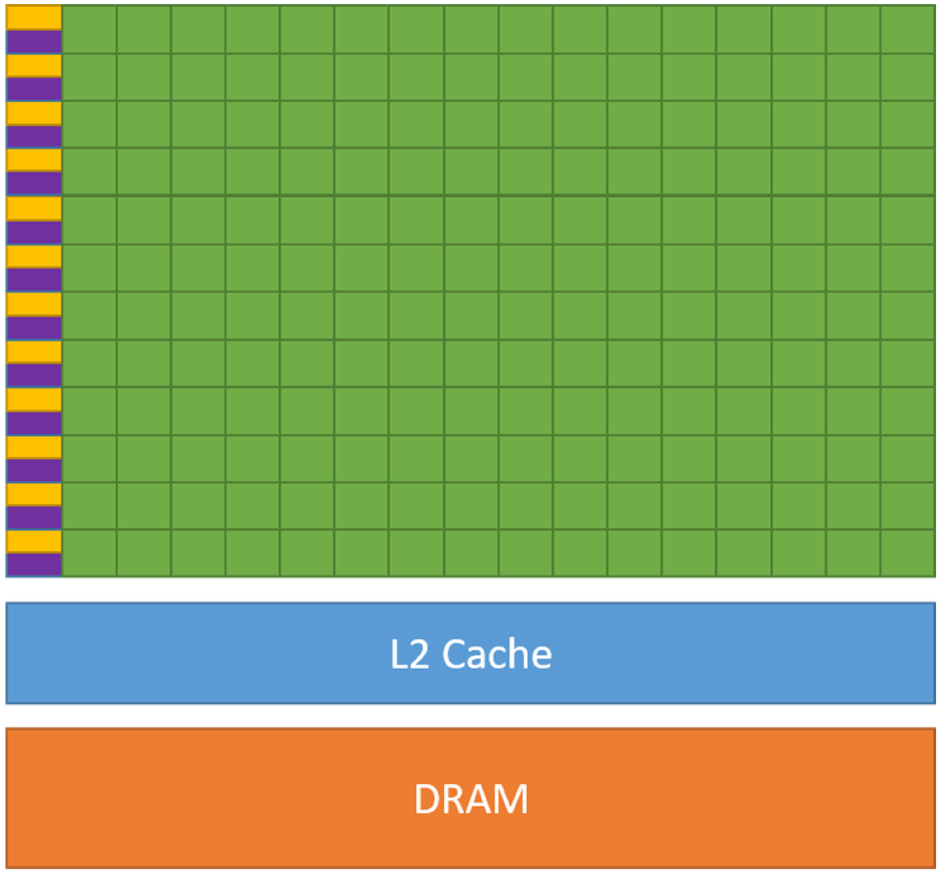


Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten New plot and data collected for 2010-2015 by K. Rupp

CPU and GPU Comparison

GPUs leverage parallel processing to handle large datasets





https://docs.nvidia.com/cuda/cuda-c-programming-guide/

DataFrame

Two-dimensional labeled data structure with columns of potentially different types

Structured (predefined format/schema):

- Tables
- Spreadsheets
- CSVs
- Relational databases

Semi-structured:

- Graph
- JSONs

Unstructured (doesn’t follow a specific data model or schema):

- Texts
- Images
- Audios
- Genes

DataFrame

Two-dimensional labeled data structure with columns of potentially different types

Structured (predefined format/schema):

- Tables
- Spreadsheets
- CSVs
- Relational databases

Semi-structured:

- Graph
- JSONs

Unstructured (doesn't follow a specific data model or schema):

- Texts
- Images
- Audios
- Genes

	Column 1	Column 2	Column 3
Row 1	1	2	3
Row 2	2	3	4
Row 3	3	4	5

DataFrame

Two-dimensional labeled data structure with columns of potentially different types

Structured (predefined format/schema):

- Tables
- Spreadsheets
- CSVs
- Relational databases

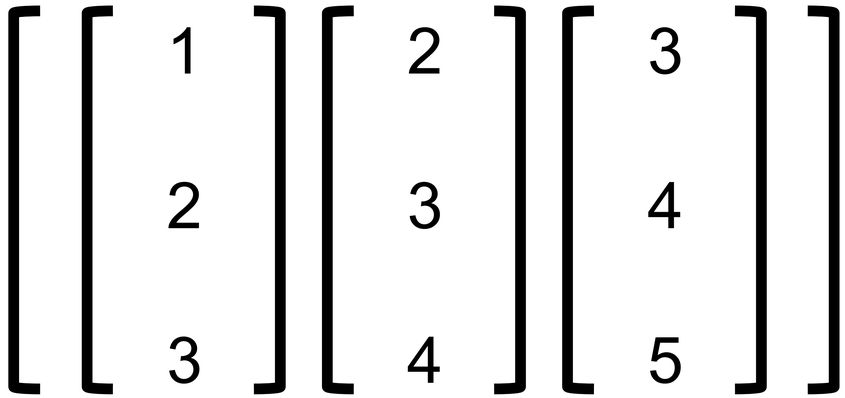
Semi-structured:

- Graph
- JSONs

Unstructured (doesn't follow a specific data model or schema):

- Texts
- Images
- Audios
- Genes

	Column 1	Column 2	Column 3
Row 1	1	2	3
Row 2	2	3	4
Row 3	3	4	5



Basic Idea

Vectorized operations are typically much faster than equivalent loops or iterations

Problem: (Column 1 - Column 2) x Column 3:

Sequential:

1. (1 - 2) x 3 = -3
2. (2 - 3) x 4 = -4
3. (3 - 4) x 5 = -5

	Column 1	Column 2	Column 3
Row 1	1	2	3
Row 2	2	3	4
Row 3	3	4	5

Parallel (able to leverage vector operations):

$$\left(\begin{bmatrix}1\\2\\3\end{bmatrix}-\begin{bmatrix}2\\3\\4\end{bmatrix}\right)\times\begin{bmatrix}3&4&5\end{bmatrix}=\begin{bmatrix}-3\\-4\\-5\end{bmatrix}$$

$$\left[\begin{bmatrix}1\\2\\3\end{bmatrix}\begin{bmatrix}2\\3\\4\end{bmatrix}\begin{bmatrix}3\\4\\5\end{bmatrix}\right]$$

Basic Idea

Total performance scales with both vector width and number of cores

Problem: (Column 1 - Column 2) x Column 3:

Sequential:

1.(1 - 2) x 3 = -3
2.(2 - 3) x 4 = -4
3.(3 - 4) x 5 = -5

	Column 1	Column 2	Column 3
Row 1	1	2	3
Row 2	2	3	4
Row 3	3	4	5

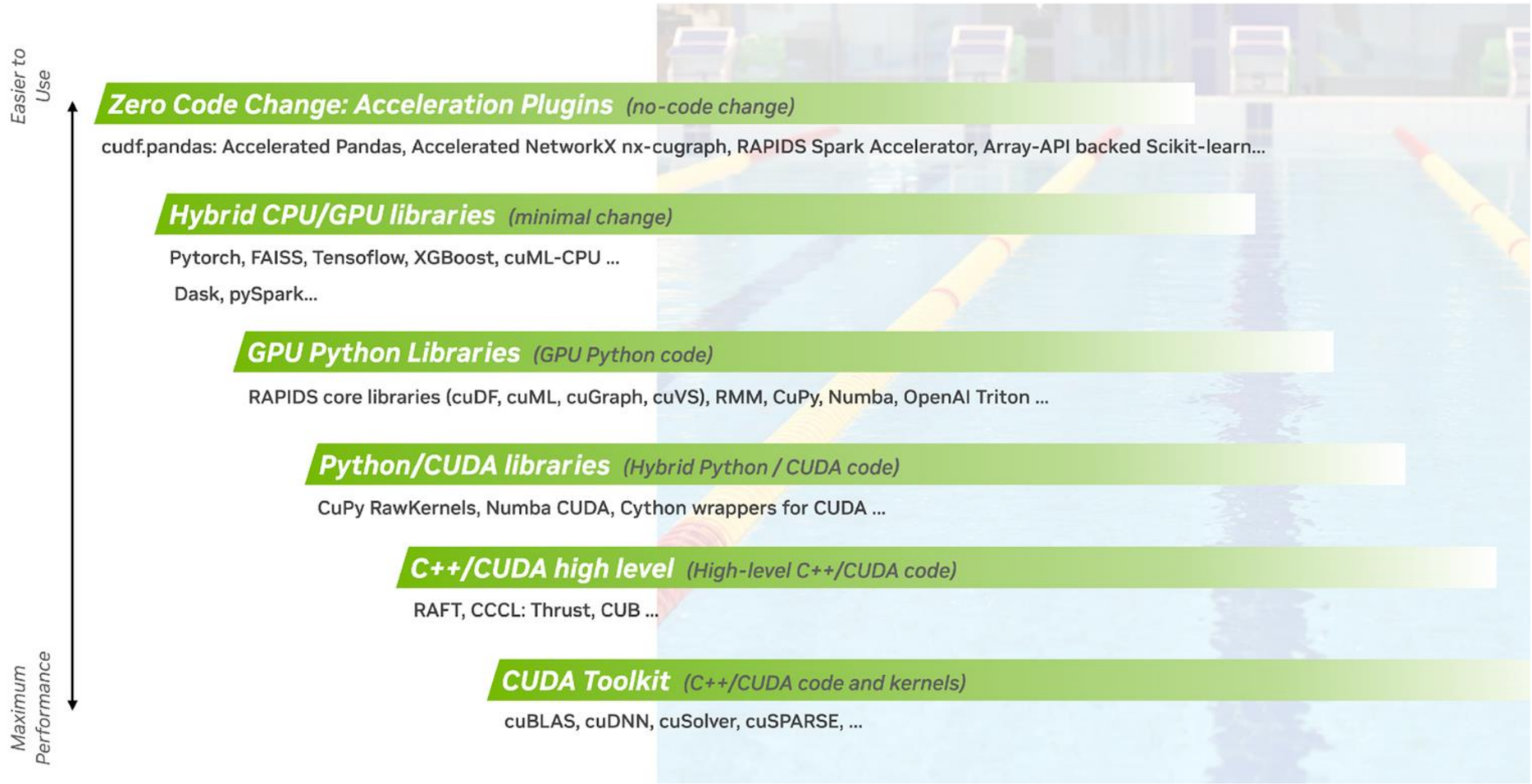
Parallel (able to leverage vector operations):

$$\left(\begin{bmatrix}1\\2\\3\end{bmatrix}-\begin{bmatrix}2\\3\\4\end{bmatrix}\right)\times\begin{bmatrix}3&4&5\end{bmatrix}=\begin{bmatrix}-3\\-4\\-5\end{bmatrix}$$

$$\left[\begin{bmatrix}1\\2\\3\end{bmatrix}\begin{bmatrix}2\\3\\4\end{bmatrix}\begin{bmatrix}3\\4\\5\end{bmatrix}\right]$$

Accelerated Computing Swim Lanes

RAPIDS makes GPU-acceleration more seamless while enabling specialization for maximum performance



How Data Science Uses cuDF

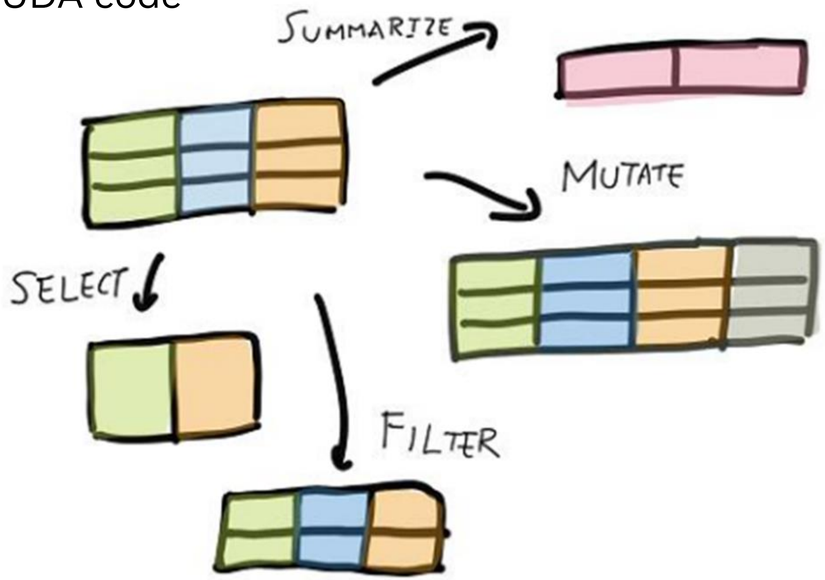
GPU DataFrame library accelerates ETL, analytical, and data processing for machine learning workflows

Python interface to CUDA C++ libraries for loading, joining, aggregating, filtering, and otherwise manipulating data using a DataFrame style API

- Accelerated file readers for prevalent formats in data science
- Uses Apache Arrow columnar memory format as its internal data format for efficient processing of tabular data
- Powerful aggregation functions using GroupBy
- Various string and list manipulation
- Ability to write user-defined functions (UDFs), without needing to write CUDA code
  - JIT compilation of User-Defined Functions (UDFs) using Numba

Resulting in:

- Faster statistical analysis
- Quicker response from interactive visualization
- Data preparation for downstream ML tasks
- Remove feature engineering bottlenecks



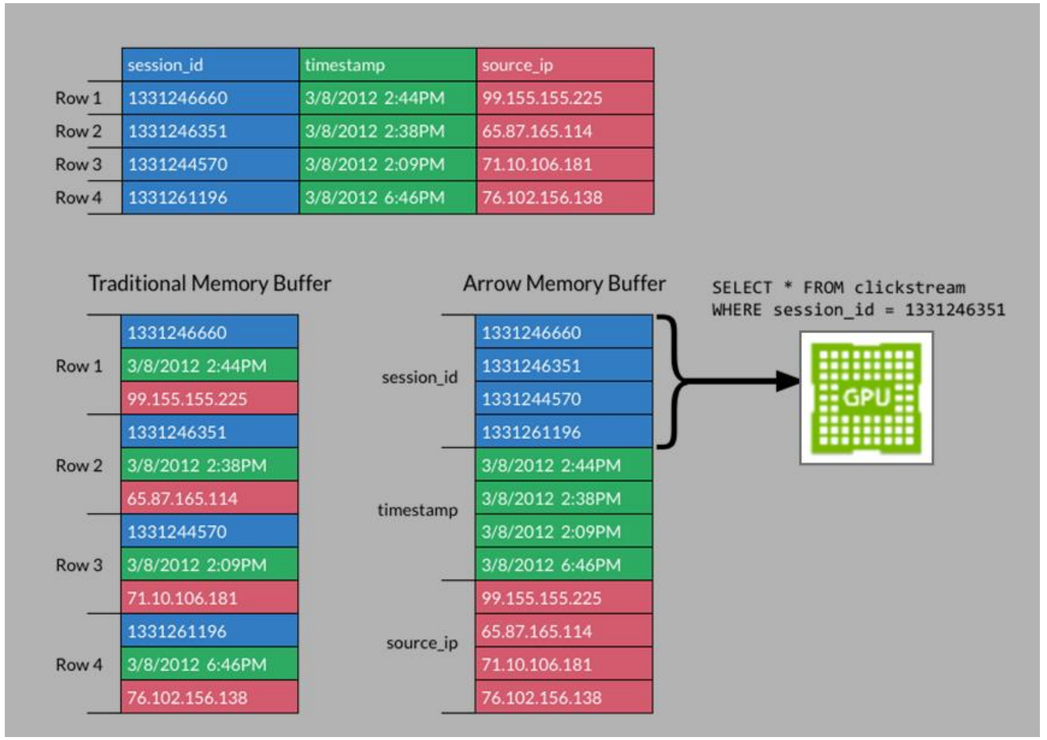
Apache Arrow

Open-source, cross-language development platform for in-memory data processing and analytics

- Columnar Format
  - Data is stored column-by-column instead of row-by-row
  - Efficient compression by grouping similar data types together
  - Leverages GPU strengths for vectorized processes using SIMD (single instruction, multiple data) operations
- In-memory data structure specification
- De-facto standard

Advantages:

- Fast access for operations like aggregation, filtering, and vectorized computations
- Zero-copy data transfer - can be shared across tools without being serialized, deserialized, or duplicated
- Reduced memory requirements enables efficient handling of large datasets
- Integrates well with columnar file formats such as Parquet, allowing for efficient reading of data from disk
- Interoperability

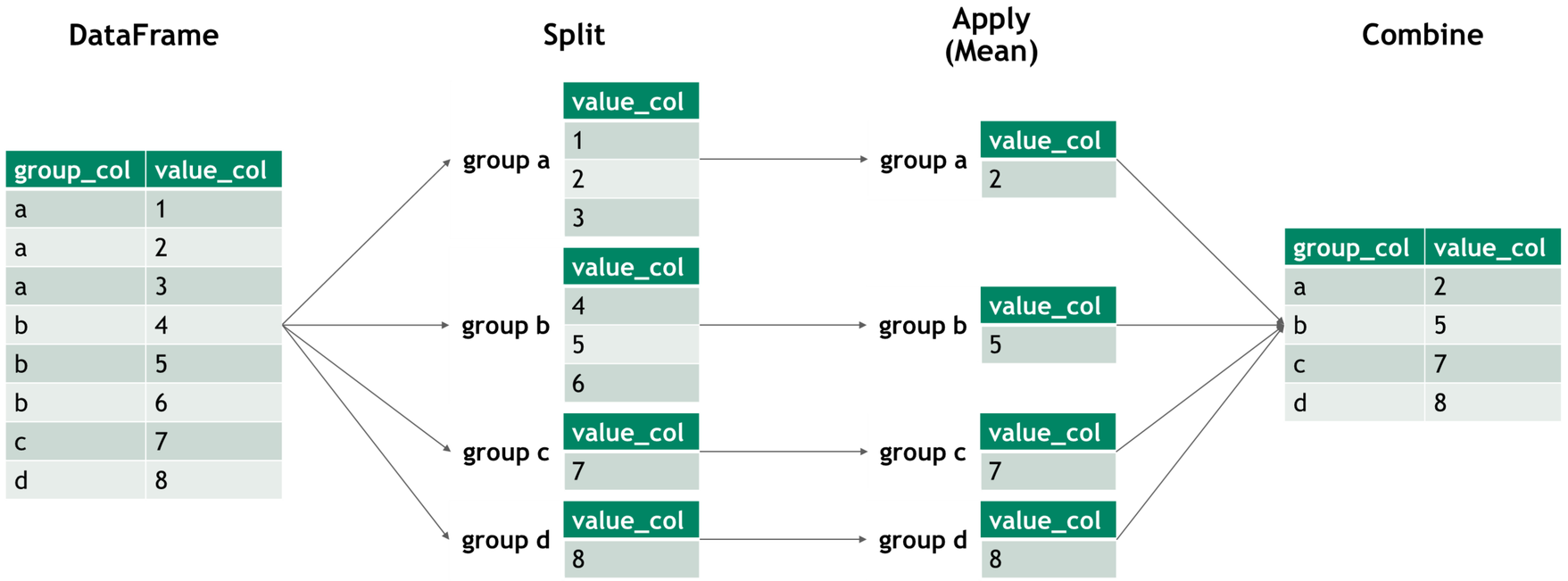


GroupBy Operations

Split-apply-combine

GroupBy operations allow users to separate data into groups based on common characteristics or a defined criteria

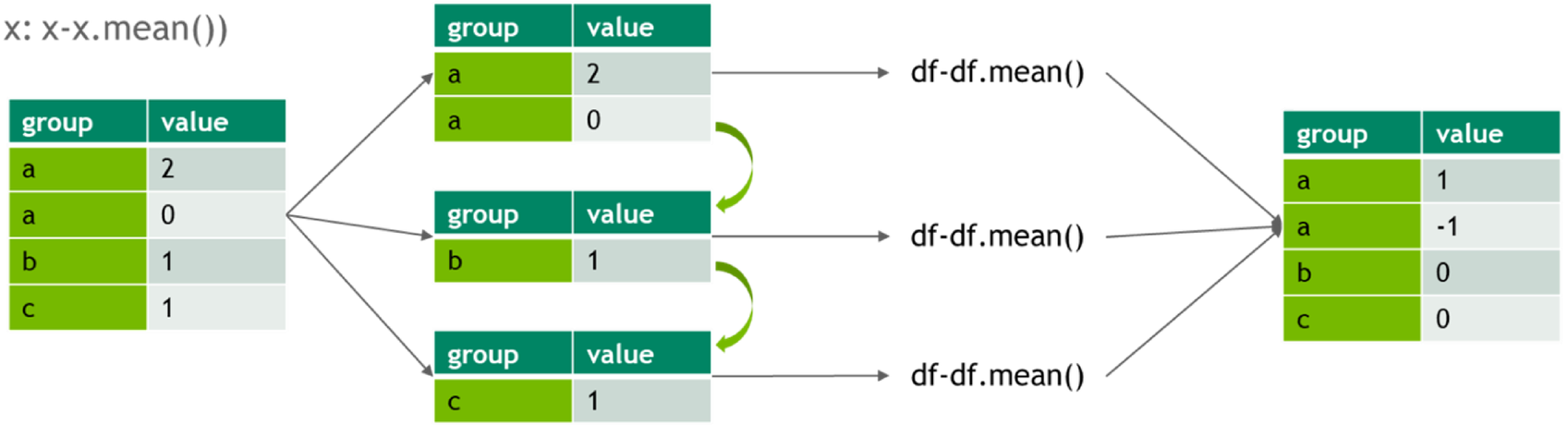
- .agg/.aggregate - compute a summary statistic (or statistics) for each group
- .transform - perform some group-specific computations and return a like-indexed object
- .apply - apply function group-wise and combine the results together



GroupBy Operations

Design for vector operations and minimize iterations

df.groupby.apply(lambda x: x-x.mean())

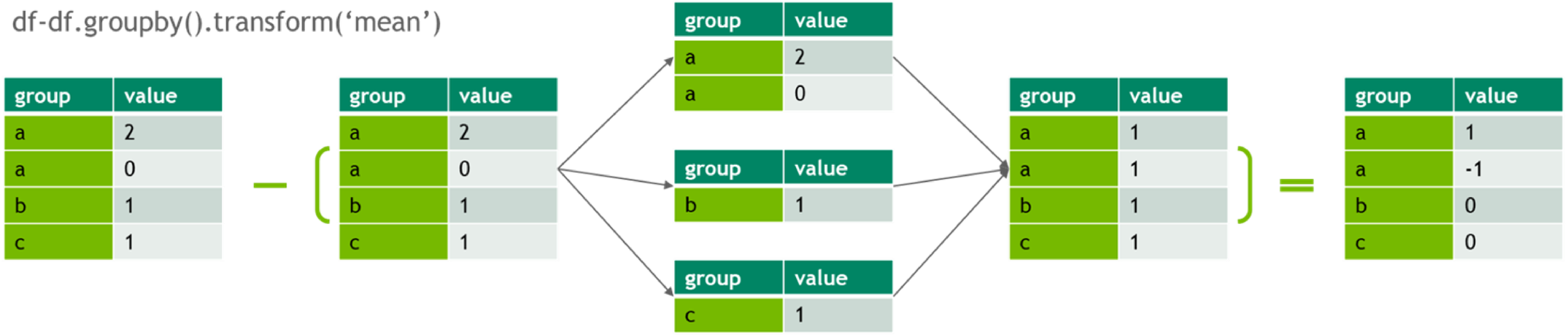


GroupBy Operations

Design for vector operations and minimize iterations

df.groupby.apply(lambda x: x-x.mean())

df-df.groupby().transform('mean')



RAPIDS cuDF

Enabler for popular DataFrame libraries

Several ways to use cuDF:

- Zero code change acceleration:
  - cudf.pandas
    - Write code with the full flexibility of pandas, then load cudf.pandas to accelerate on the GPU, with automatic CPU fallback if needed
    - Third-party library compatible and accelerates the entire pandas ecosystem
  - cuDF provides an in-memory, GPU-accelerated execution engine for Python users of the Polars Lazy API
- Write cuDF code using DataFrame style API
- Dask-cuDF extends Dask & RAPIDS Accelerator for Apache Spark

Announcing RAPIDS cuDF Accelerates pandas with Zero Code Change

World's fastest data analytics with pandas

- Most powerful and flexible open-source data analysis / manipulation tool
  - Over 9.5M pandas users
- Accelerated by cuDF
  - 150x Faster than CPU-only
  - Unified workflow on CPUs and GPUs across laptops, workstation & datacenter
  - Compatible with third-party libraries built on pandas

Meeting Data Scientists Where They Are

pandas

```
import pandas as pd
df = pd.read_csv("filepath")
df.groupby("col").mean()
df.rolling(window=3).sum()
```

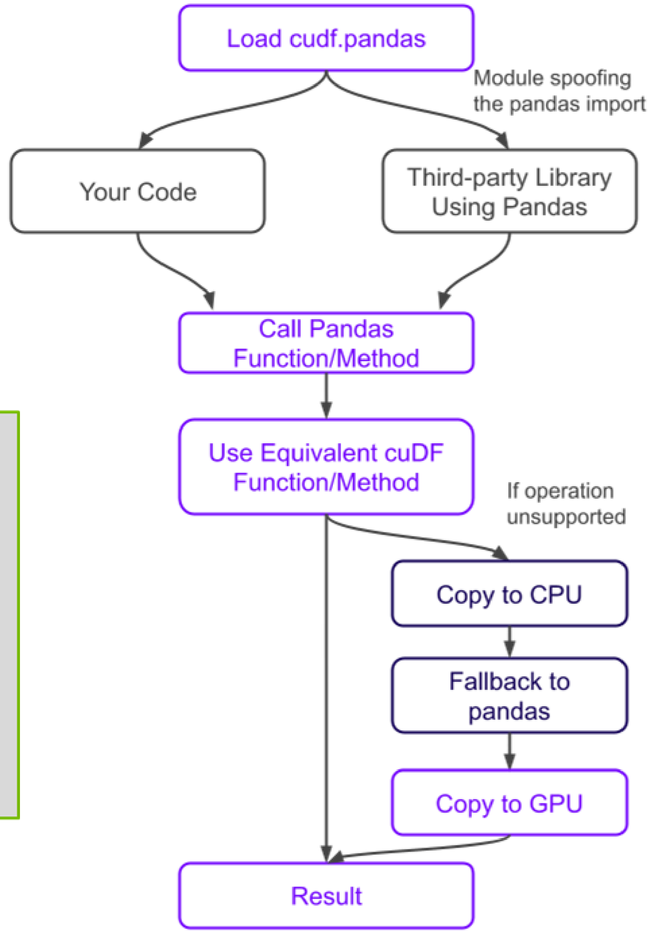
CPU

pandas
with RAPIDS

```
%load_ext cudf.pandas

import pandas as pd
df = pd.read_csv("filepath")
df.groupby("col").mean()
df.rolling(window=3).sum()
```

GPU-Accelerated



Polars + RAPIDS cuDF

Polars is the fastest growing data analytics library

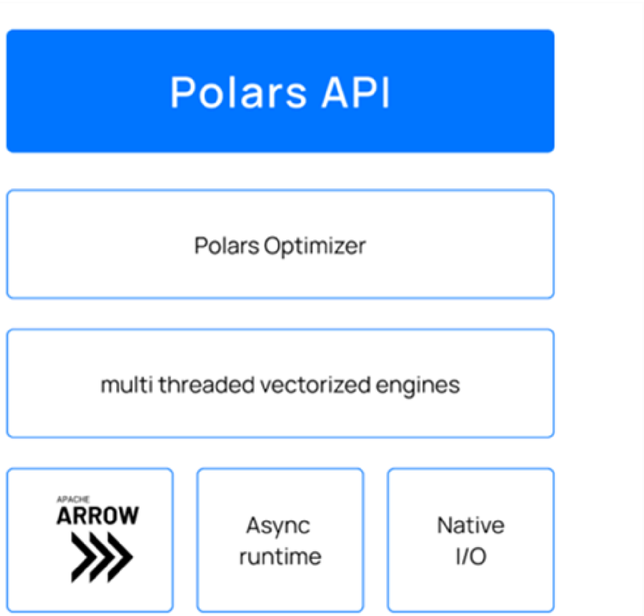
- High Performance DataFrame Library written in Rust
  - Build on Apache Arrow columnar memory format
  - Lazy Execution & Query Optimizer
- Gaining rapid adoption in data science community as a Pandas alternative

Polars is Accelerated with RAPIDS cuDF

- NVIDIA RAPIDS accelerates Polars through a GPU engine built into the lazy API
- The GPU Engine object can be passed to query execution for automatic acceleration (collect() function)

Execution Process

- Polars builds and optimizes the query plan
- cuDF inspects the plan, replacing supported operations with GPU-accelerated versions
- Unsupported operations fallback to CPU execution with minimal overhead

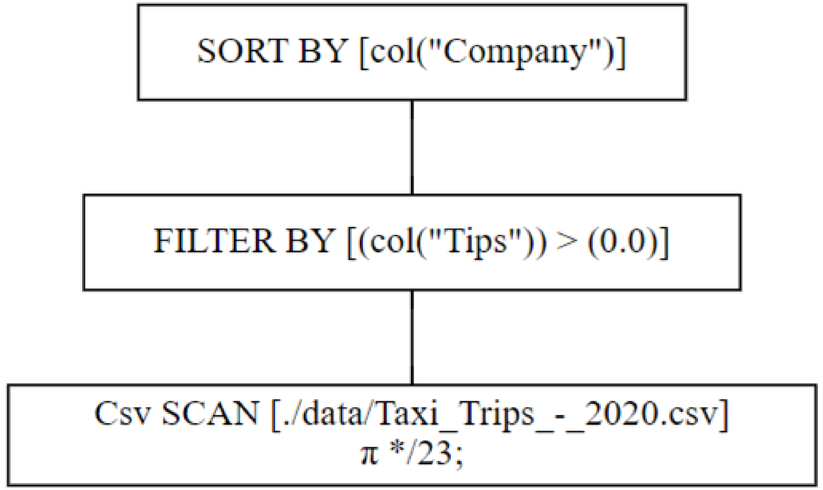


Lazy Execution and Query Optimization

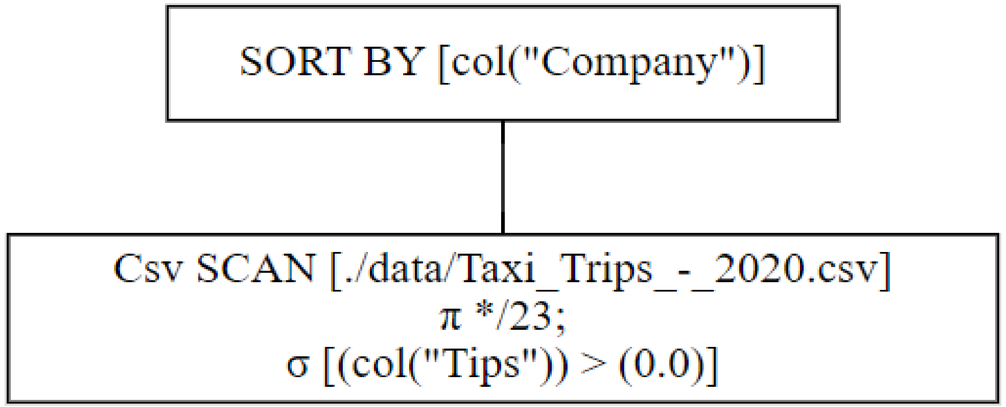
How does it work?

Lazy Execution

- Polars uses lazy evaluation by default, but has eager execution available
- Queries are not executed immediately, but built into an execution plan



Unoptimized Query



Optimized Query

Lazy Execution and Query Optimization

How does it work?

Lazy Execution

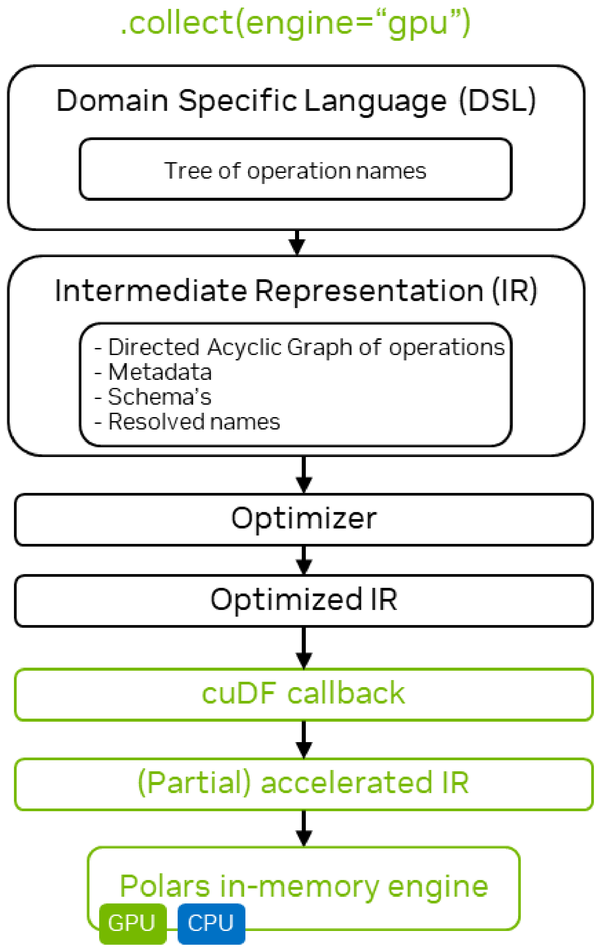
- Polars uses lazy evaluation by default, but has eager execution available
- Queries are not executed immediately, but built into an execution plan

Benefits of lazy evaluation:

- Allows global query optimization
- Minimizes unnecessary computations and memory usage
- Enables out-of-core processing for datasets larger than RAM

Query optimizer:

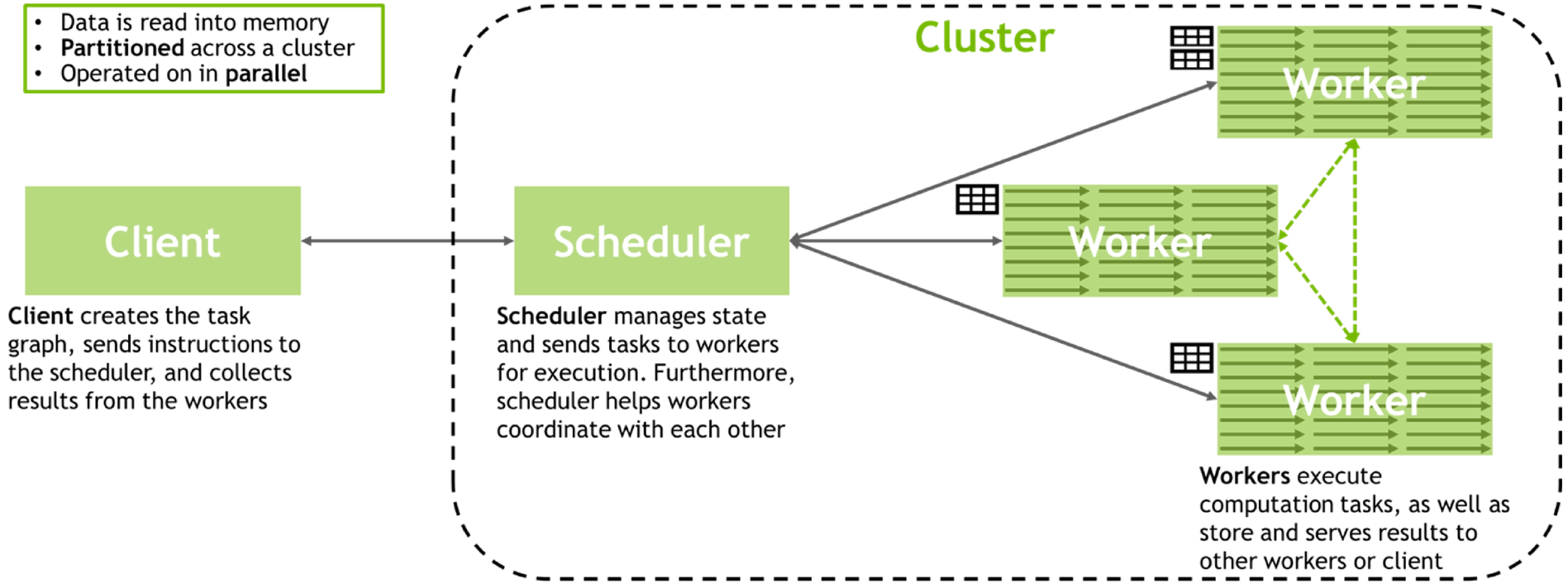
- Analyzes full query plan to determine most efficient execution
- Applies optimizations like predicate pushdown and projection pushdown
- Parallelizes operations across GPU/CPU cores automatically



RAPIDS + Dask

Distributed parallel computing on multi-GPU systems

- Dask is a parallel computing library built to scale Python workloads from single machines to supercomputer clusters
- Executes operations on chunks of data and aggregates results, thus enabling working with datasets that are larger than memory
- Scheduler coordinates the actions of several dask-worker processes spread across computing nodes
- Lazy execution - operations on a Dask DataFrame build a task graph, which is optimized prior to execution



Accelerating Data Processing

Choosing tools based on use case, data size, performance requirements, and existing infrastructure

- operates in-memory
- eager execution
- suited for smaller datasets (up to 1GB)

- handles larger-than-memory data
- lazy execution
- suited for medium to large datasets (1GB - low hundreds GB)

- distributed computing
- lazy execution
- suited for massive datasets across cluster of machines
- requires more setup, overhead, and resources

Accelerating Data Processing

Choosing tools based on use case, data size, performance requirements, and existing infrastructure

- operates in-memory
- eager execution
- suited for smaller datasets (up to 1GB)

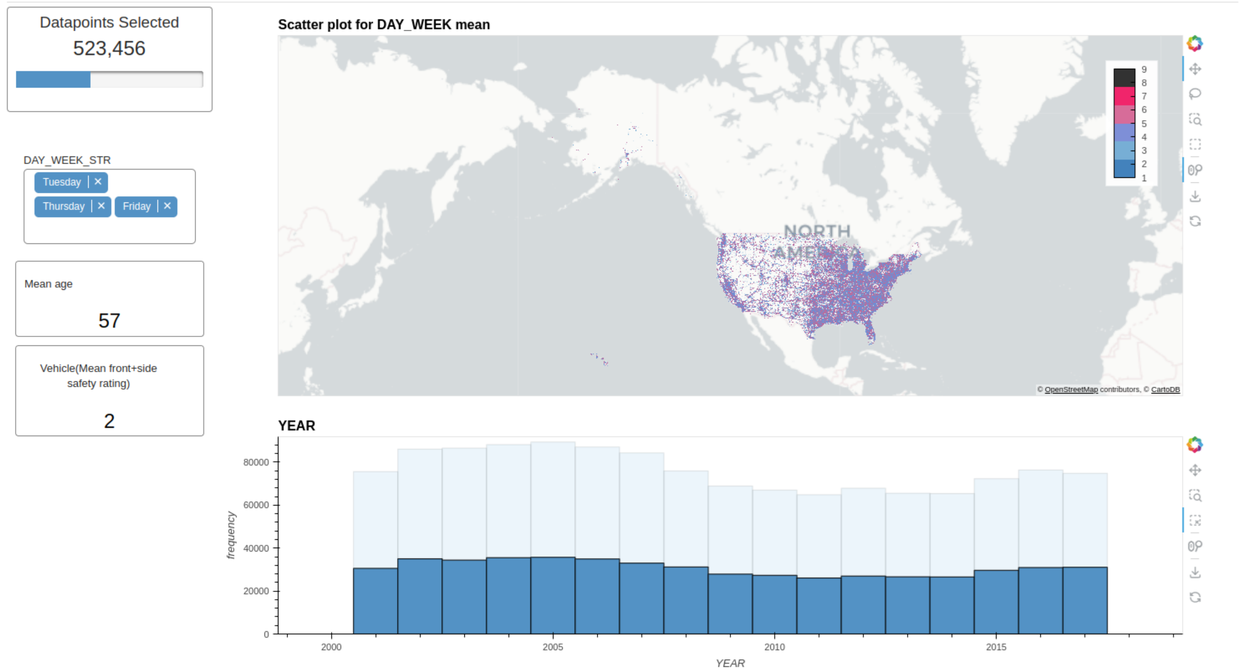
- handles larger-than-memory data
- lazy execution
- suited for medium to large datasets (1GB - low hundreds GB)

- distributed computing
- lazy execution
- suited for massive datasets across cluster of machines
- requires more setup, overhead, and resources

RAPIDS Data Visualization

Popular libraries offer direct cuDF support or easy integration

- Scatter, line, bar charts, and more
- Interactive dashboards
  - Disruption due to slow processing speed creates friction in the analysis process
- cuxfilter
  - Built on a host of pyViz ecosystem tools
  - Cross-filtering apply interactions from one chart as filters to other charts in the dashboard
- Efficient GPU-backed in-browser visualization
  - hvPlot + Datashader + cuxfilter + Plotly Dash

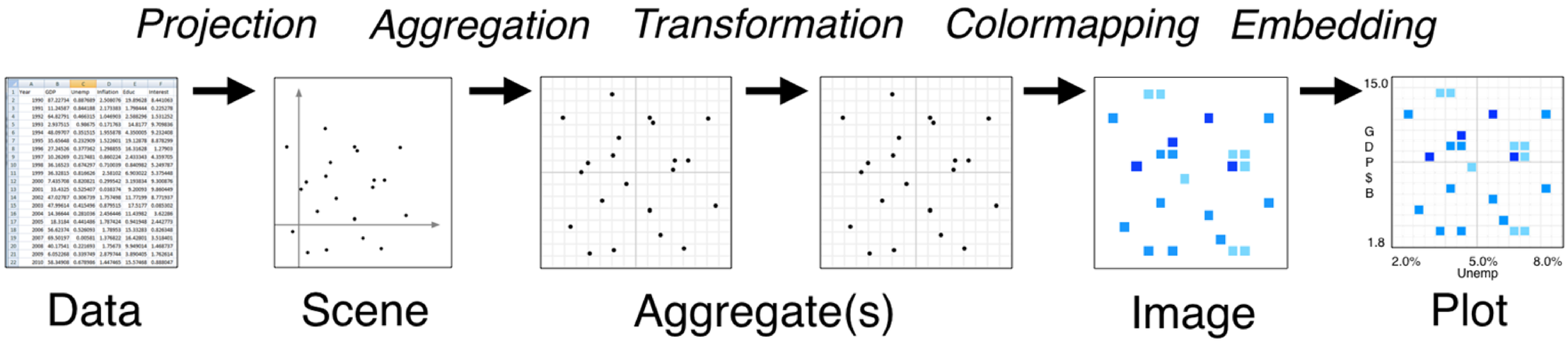


Datashader

Disruption due to slow processing speed creates friction in the analysis process

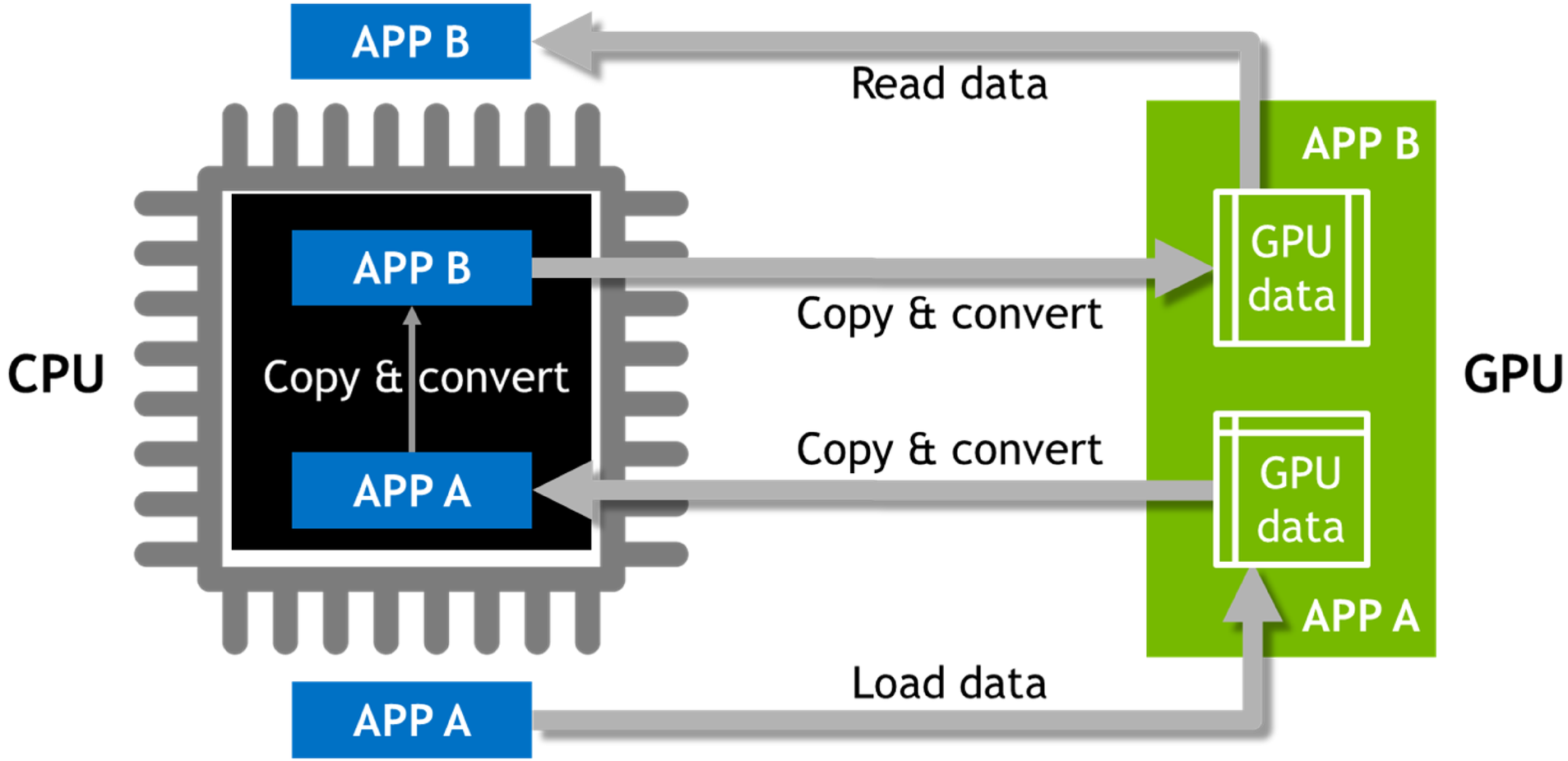
Datashader - a graphics pipeline system for creating meaningful representations of large datasets quickly and flexibly

- Work by rasterizing or aggregating large datasets into regular grids that can be viewed as images
- Can be GPU-accelerated by leveraging cuDF or other CPU-accelerated tools for the data aggregation step



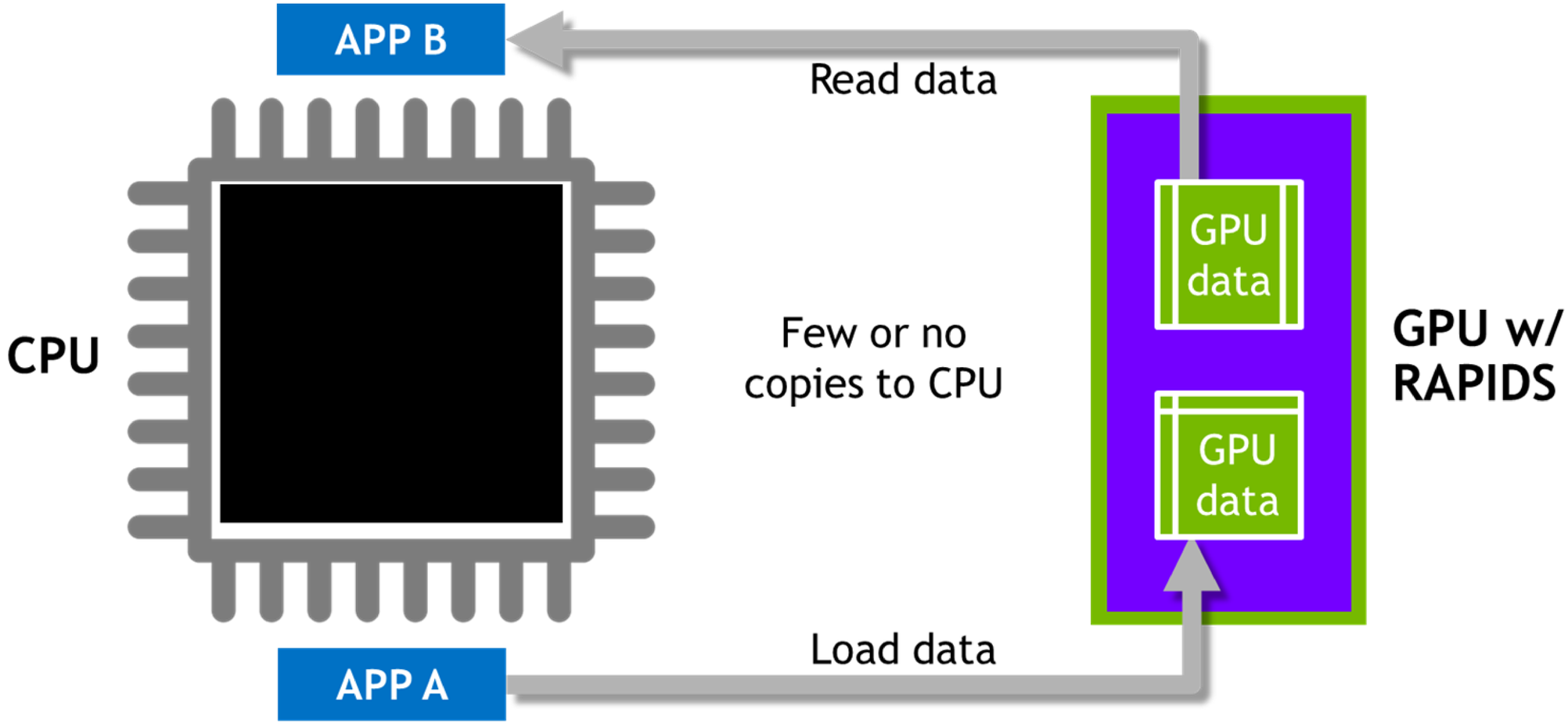
Traditional Model

Too much data movement and too many makeshift data formats require transformation



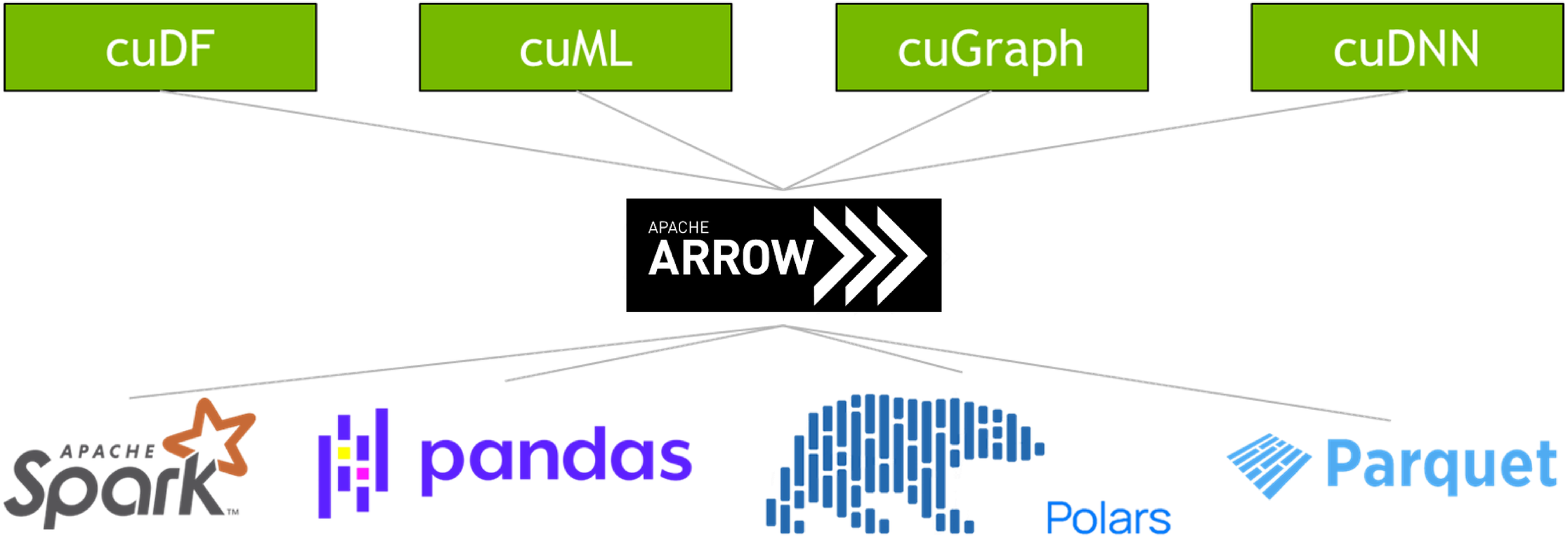
RAPIDS Model

Shared, standard data structures are essential to building an efficient workflow



One Format for Interoperability and Efficiency

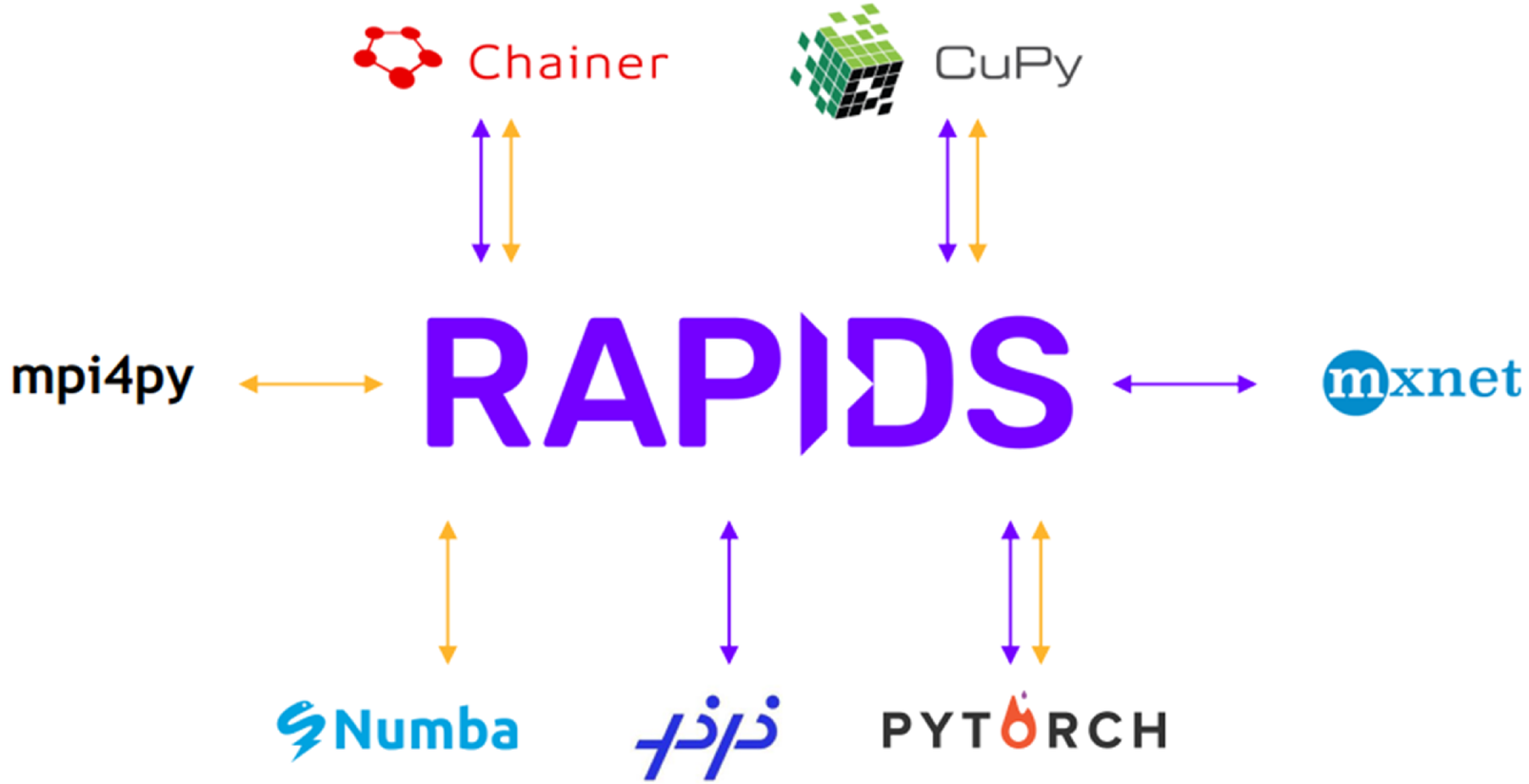
Standardized, language-agnostic in-memory columnar format for structured data



Interoperability with CuPy

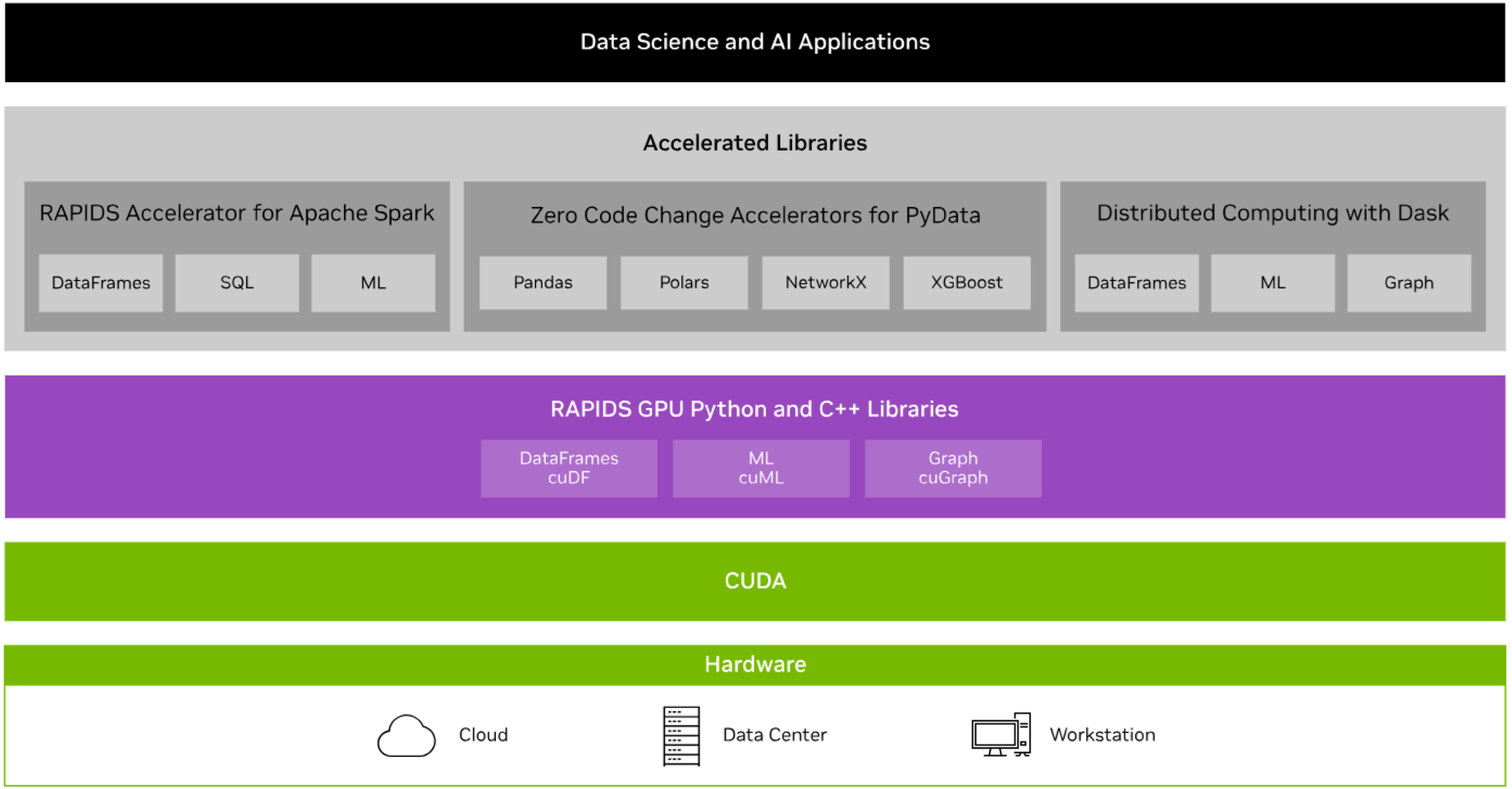
Interoperability of the GPU PyData Ecosystem

Using DLPack and __cuda_array_interface__ – the CUDA array interface is a standard format that describes a GPU array to allow sharing GPU arrays between different libraries without needing to copy or convert data



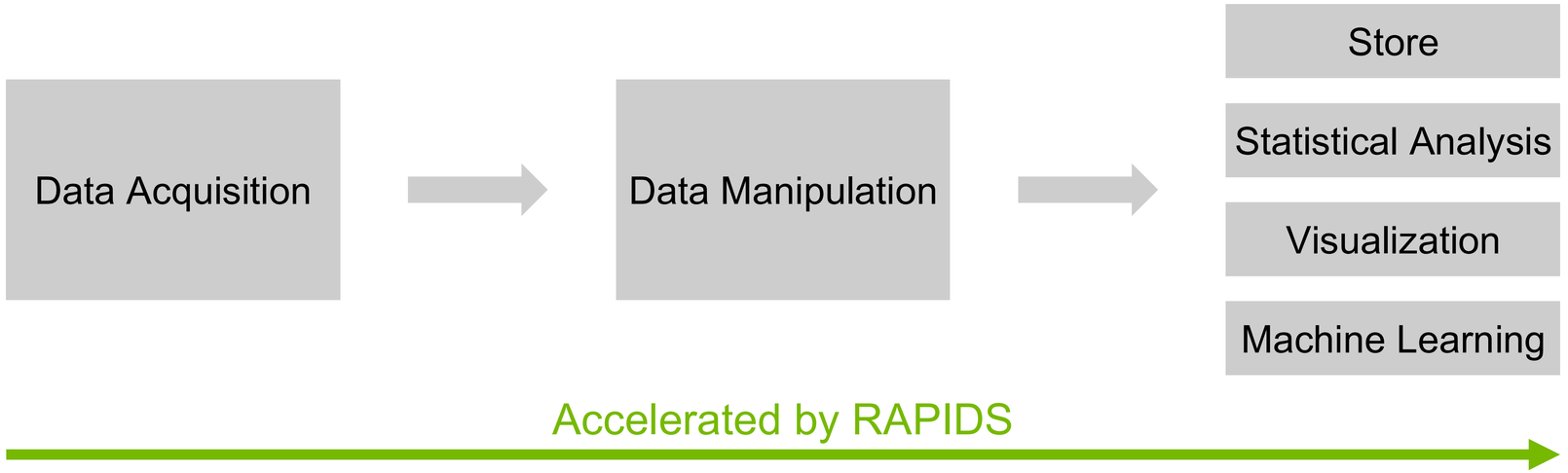
RAPIDS Transforms Data Science

An optimized hardware-to-software stack for the entire data science pipeline



Accelerating End-to-End Data Science Workflows

NVIDIA Data Science aims to provide seamless integration of NV libraries and SDKs for data scientists



Tips:

- Memory management
  - Some vectorized operations may create temporary copies of data, which can be problematic for very large datasets
- Leverage parallel processing when possible (GPU)
  - Design operations to use vector operations (groupby.apply is iterative and slow)
- Keep data on the GPU as much as possible
  - Minimize copy/convert operations
- Diagnose and remove bottlenecks

Hands-On Lab