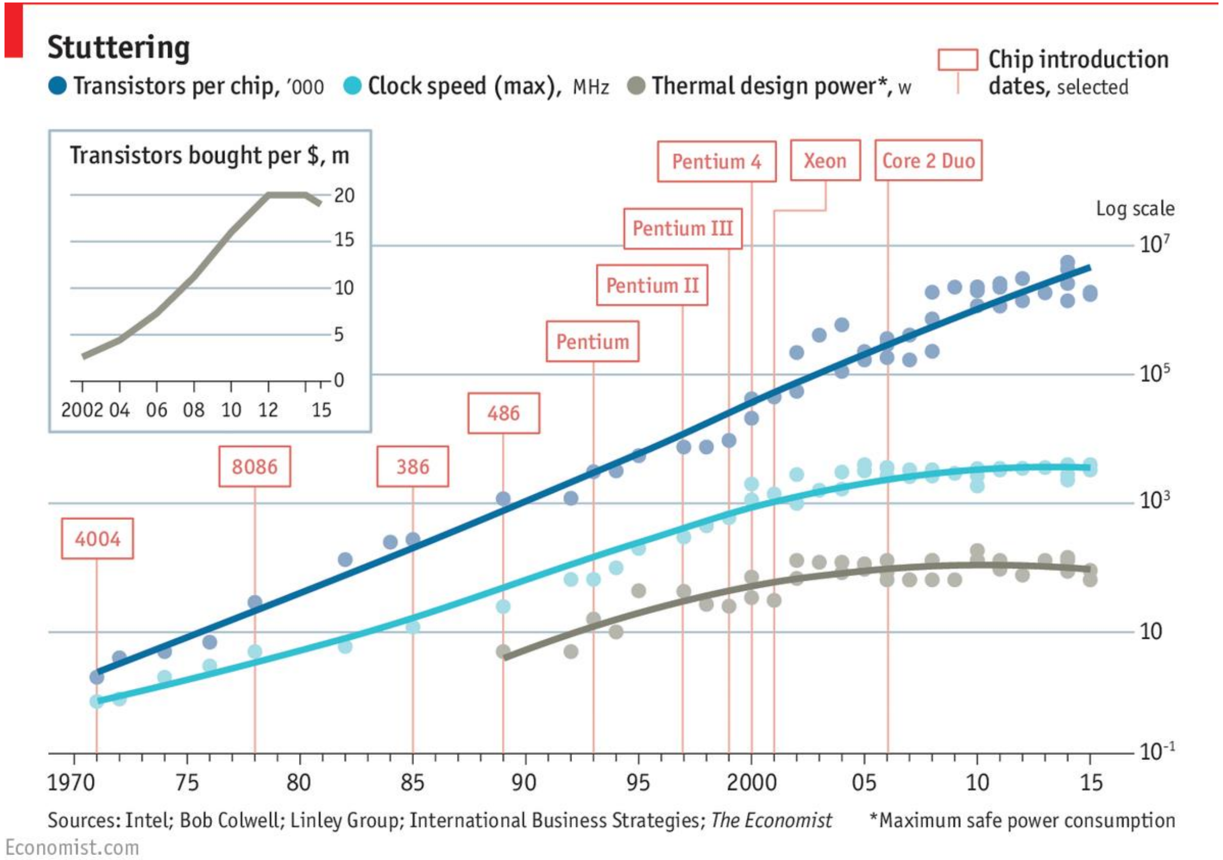
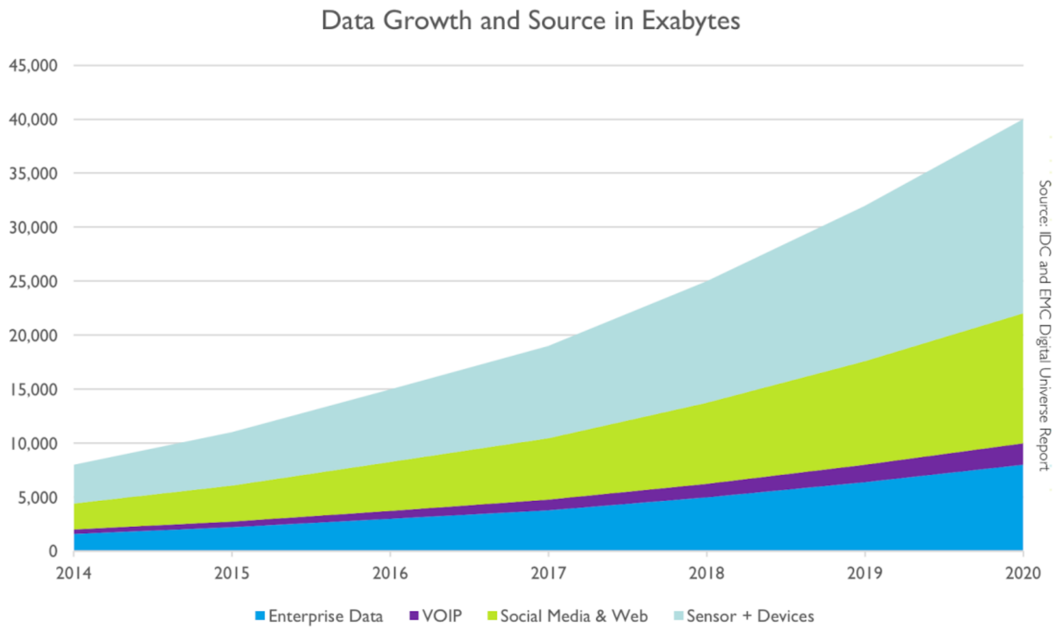
GPU Accelerated Data Analytics in Python

Mads R. B. Kristensen, NVIDIA



Stuttering
Transistors per chip, '000
Clock speed (max), MHz
Thermal design power*, w
Chip introduction dates, selected
Transistors bought per $, m
20
15
10
5
0
2002 04 06 08 10 12 15
4004
8086
386
486
Pentium
Pentium II
Pentium III
Pentium 4
Xeon
Core 2 Duo
Log scale
10^7
10^5
10^3
10
10^{-1}
1970 75 80 85 90 95 2000 05 10 15
Sources: Intel; Bob Colwell; Linley Group; International Business Strategies; *The Economist*
*Maximum safe power consumption
Economist.com



Data Growth and Source in Exabytes
45,000
40,000
35,000
30,000
25,000
20,000
15,000
10,000
5,000
0
2014 2015 2016 2017 2018 2019 2020
Enterprise Data
VOIP
Social Media & Web
Sensor + Devices
Source: IDC and EMC Digital Universe Report

Scale up and out with RAPIDS and Dask

Scale Up / Accelerate

RAPIDS and Others

Accelerated on single GPU

NumPy -> CuPy/PyTorch/..
Pandas -> cuDF
Scikit-Learn -> cuML
Numba -> Numba

Dask + RAPIDS

Multi-GPU
On single Node (DGX)
Or across a cluster

PyData

NumPy, Pandas, Scikit-Learn
and many more

Single CPU core
In-memory data

Dask

Multi-core and Distributed PyData

NumPy -> Dask Array
Pandas -> Dask DataFrame
Scikit-Learn -> Dask-ML
… -> Dask Futures

Scale out / Parallelize

Scale up and out with RAPIDS and Dask

RAPIDS and Others

Accelerated on single GPU

NumPy -> CuPy/PyTorch/..
Pandas -> cuDF
Scikit-Learn -> cuML
Numba -> Numba

PyData

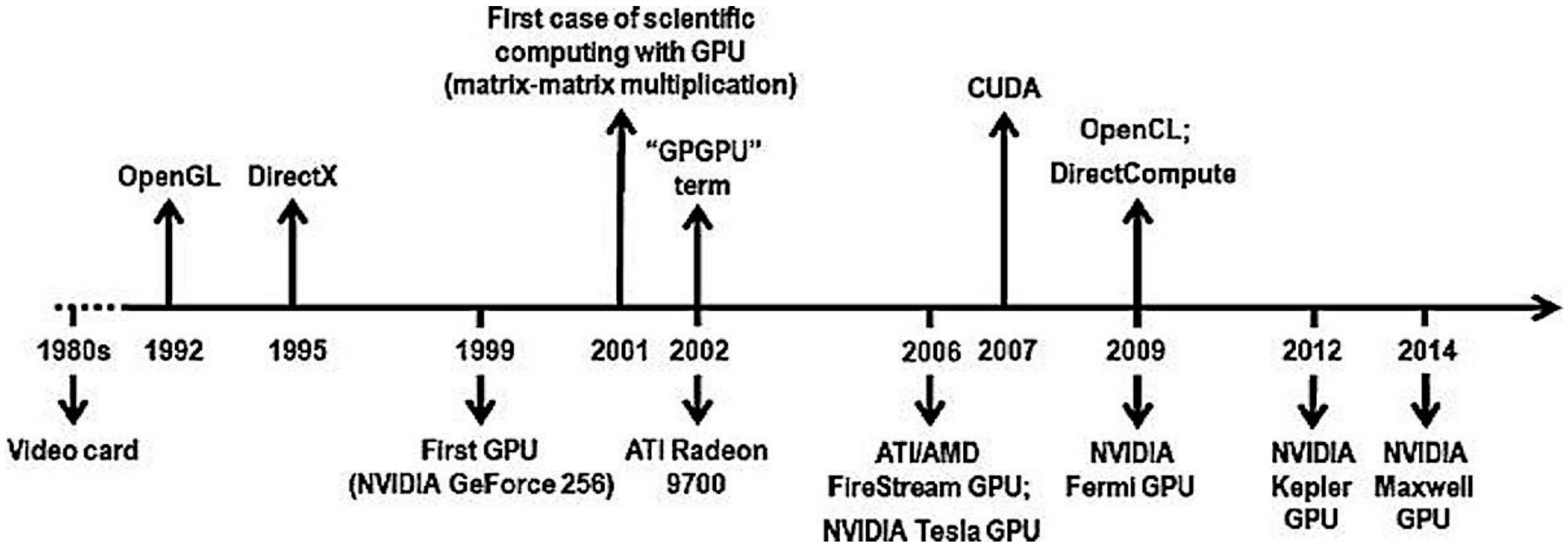
NumPy, Pandas, Scikit-Learn and many more

Single CPU core
In-memory data

Scale Up / Accelerate

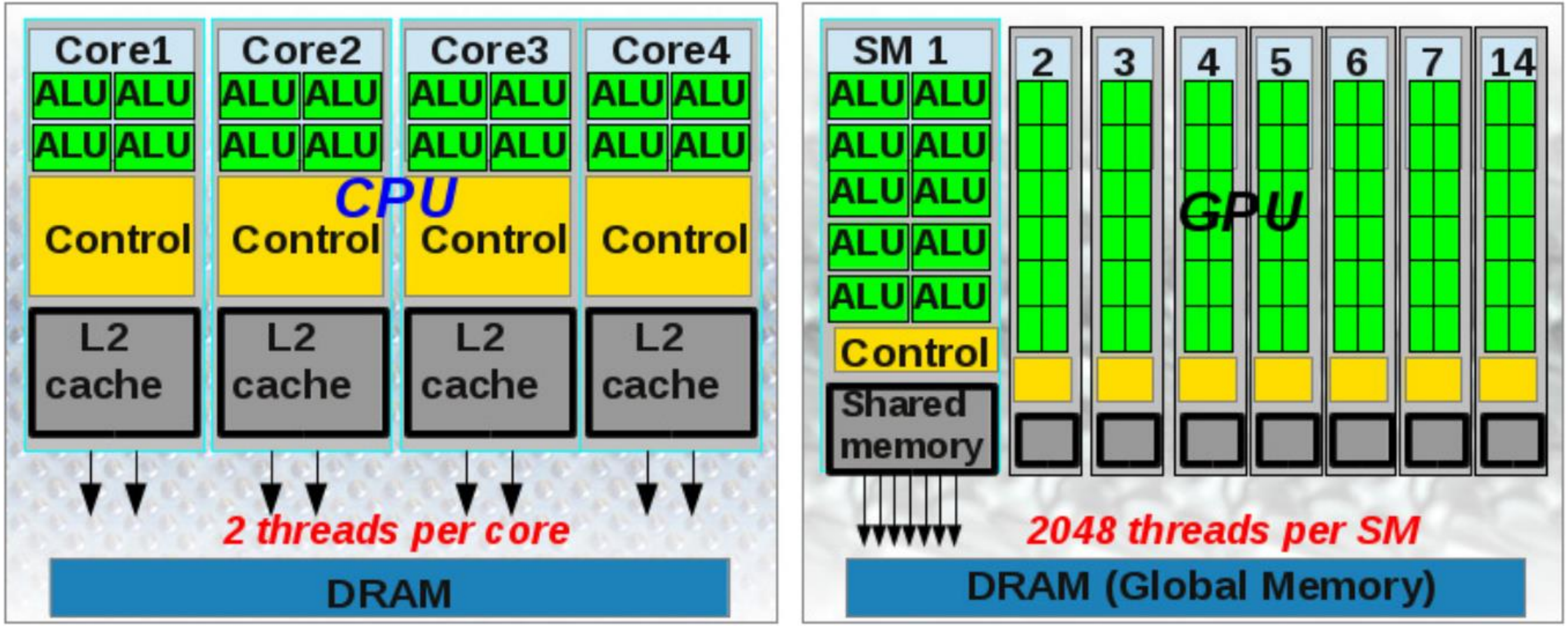
Scale out / Parallelize

History of the GPU



DOI: https://doi.org/10.1007/978-3-319-17885-1_1606

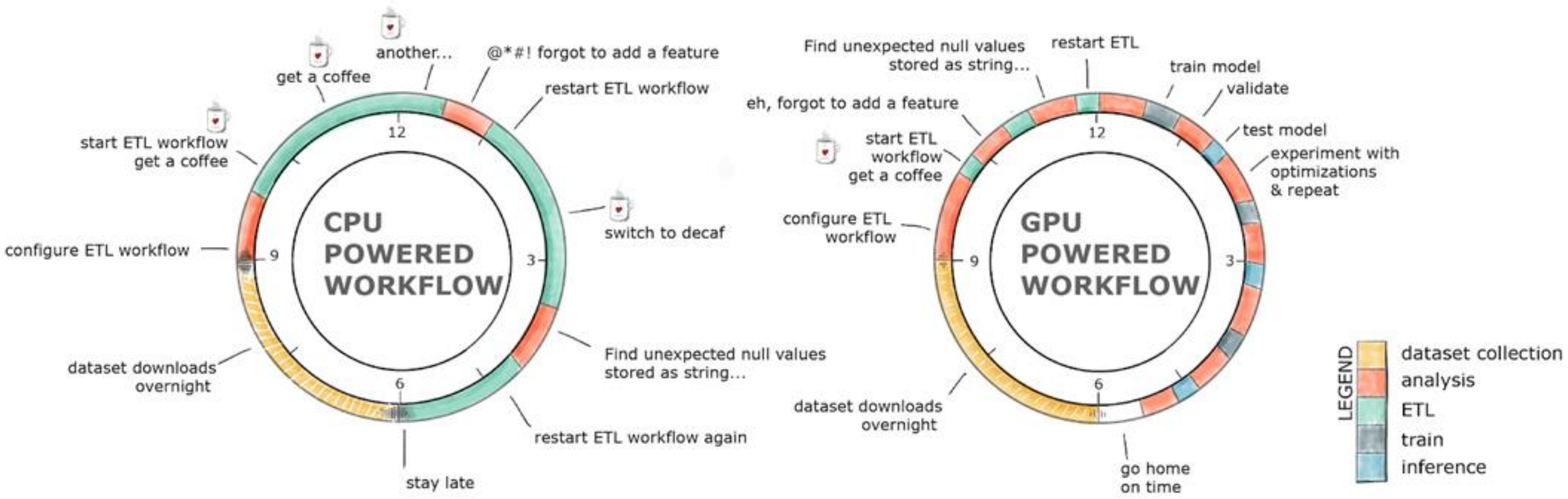
CPU vs GPU



DOI: 10.1016/j.cam.2013.12.032.

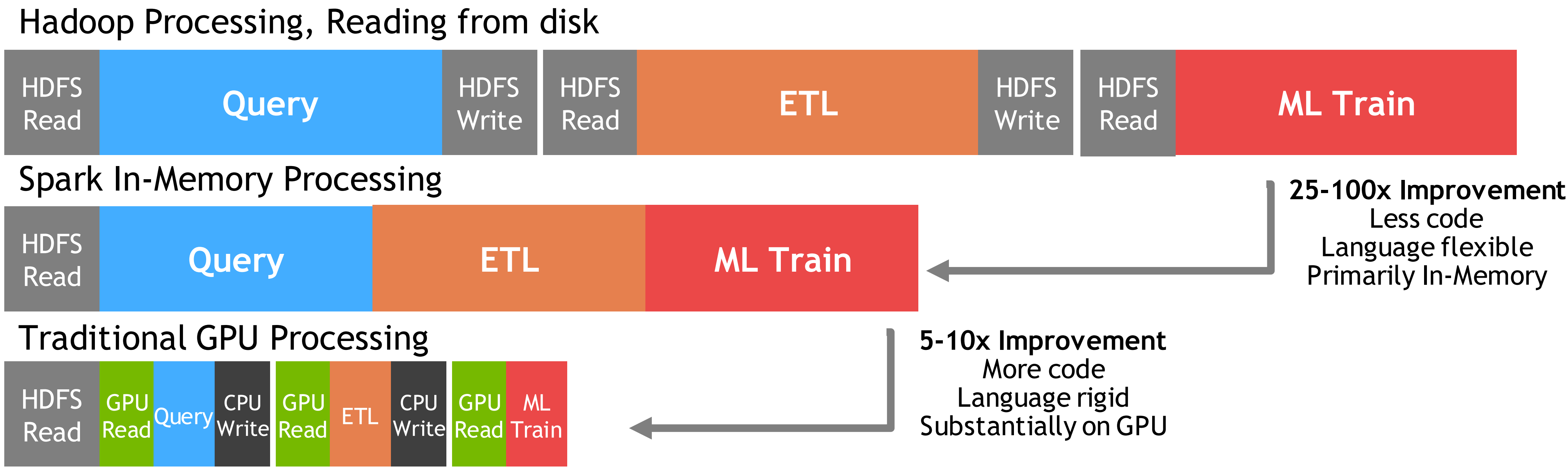
GPU-Accelerated ETL

The average data scientist spends 90+% of their time in ETL as opposed to training models



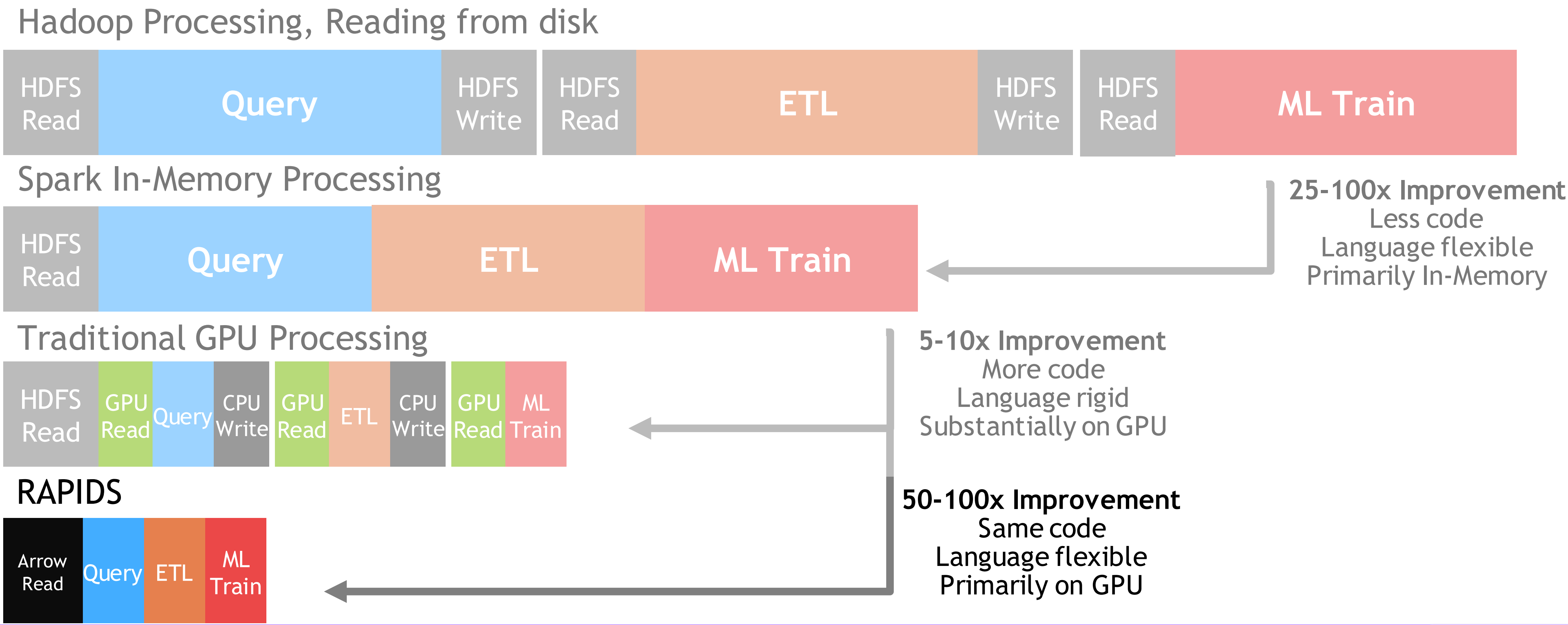
Data Processing Evolution

Faster data access, less data movement



Data Processing Evolution

Faster data access, less data movement



cuDF

cuDF

A GPU DataFrame library in Python with a pandas-like API built into the PyData ecosystem

Pandas-like API on the GPU

CPU

pandas

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

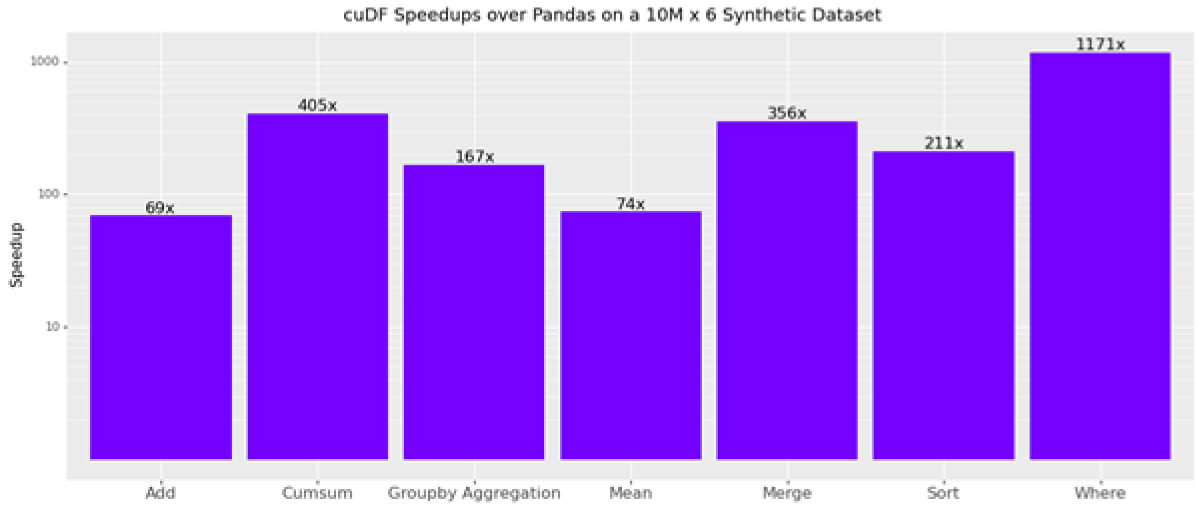
GPU

cuDF

```
>>> import cudf
>>> df = cudf.read_csv("filepath")
>>> df.groupby(“col”).mean()
>>> df.rolling(window=3).sum()
```

Average Speed-Ups: *10-100x*

Best-in-Class Performance (Benchmark)



Groupby | Strings and Regex | UDFs | Nested Types | Time Series

Indexing | Missing Data | CuPy Interoperability | Rolling Windows

10 Minutes to cuDF

NVIDIA A100 vs. AMD EPYC 7642 48-Core Processor
cuDF Python vs. Pandas

KvikIO

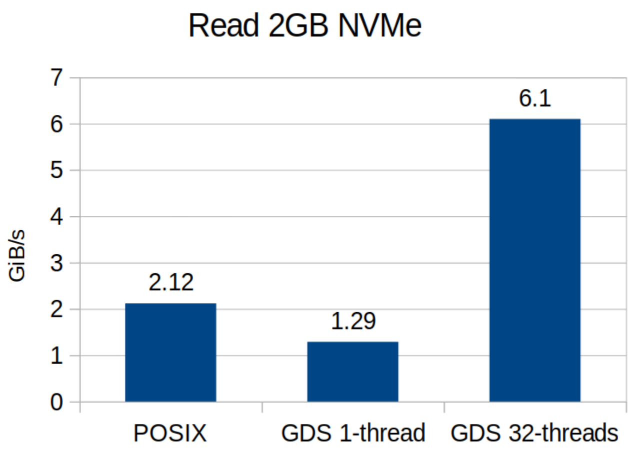
RAPIDS KvikIO

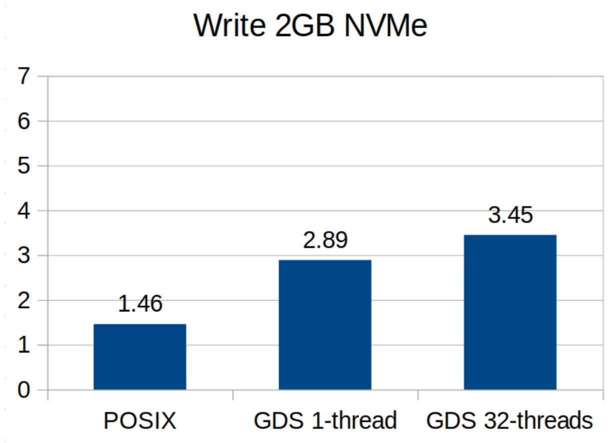
KvikIO is a C++ and Python frontend for **cuFile** that provide features such as an object-oriented API, exception handling, RAII semantic, multithreading IO, fallback mode, and a **Zarr backend**.

Using KvikIO should feel natural to C++ and Python developers.

```cpp
#include <cuda_runtime.h>
#include <kvikio/file_handle.hpp>
using namespace std;

int main() {
  void *a = nullptr;
  cudaMalloc(&a, 80);
  // Read file into `a` in parallel using 16 threads
  kvikio::default_thread_pool::reset(16);
  {
    kvikio::FileHandle f("/nvme/input.raw", "r");
    future<size_t> fut = f.pread(a, sizeof(a), 0);
    size_t read = fut.get(); // Blocking
    // Note, `f` closes automatically on destruction.
  }
}
```

Comparing KvikIO's Zarr backend versus manually copying between GPU and host memory before accessing the Zarr array using POSIX





NVIDIA DGX A100 (using one of the GPUs)
2x AMD EPYC 7742 64-Core@3.4GHz (max boost)
1x NVMe Samsung PM1733 SSD (MZWLJ3T8HBLS-00007)

KvikIO: https://github.com/rapidsai/kvikio

```python
# Write CuPy array to disk
import cupy
import kvikio
a = cupy.arange(10)
with kvikio.CuFile("/nvme/input.raw", "w") as f:
    f.write(a)

# Write same CuPy array to a Zarr store
import zarr
from kvikio.zarr import GDSStore
z = zarr.array(a,
    compressor=None,
    store=GDSStore("/nvme/store"),
    meta_array=cupy.empty(()),
)
# We can not access the Zarr array `z` as a
# regular CuPy array.
b = z[:]  # Read from disk to GPU seamlessly
```

cuML

cuML

Accelerated Machine Learning with a scikit-learn API

CPU

Scikit-learn

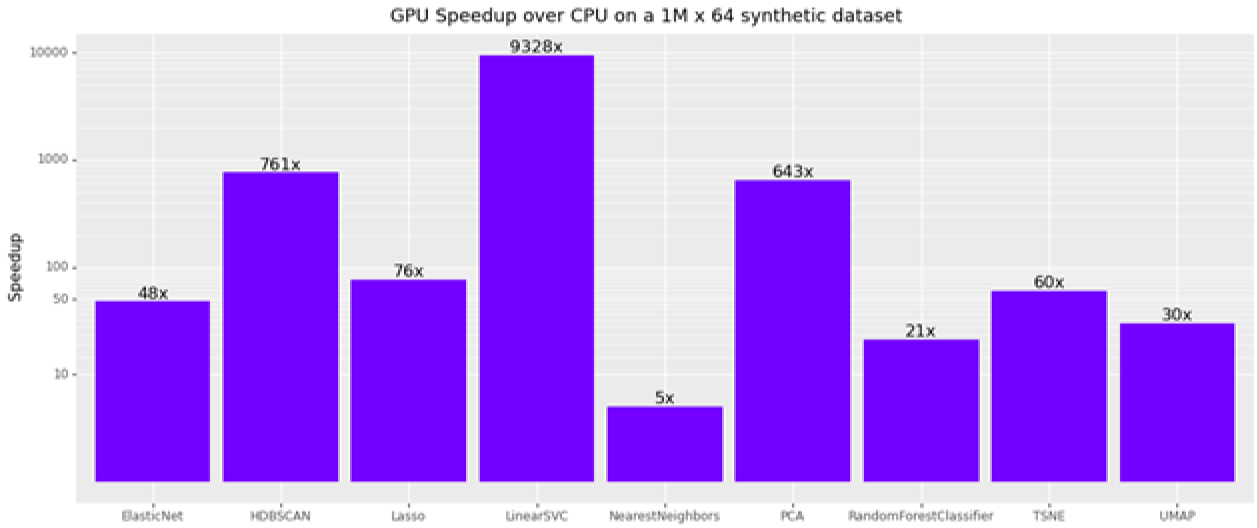
```
>>> from sklearn.ensemble import
RandomForestClassifier
>>> clf = RandomForestClassifier()
>>> clf.fit(x, y)
```

GPU

cuML

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

50+ GPU-Accelerated Algorithms & Growing



Time Series	Classification	Regression	Clustering	Preprocessing
Cross Validation	Tree Models	Dimensionality Reduction	Explainability	

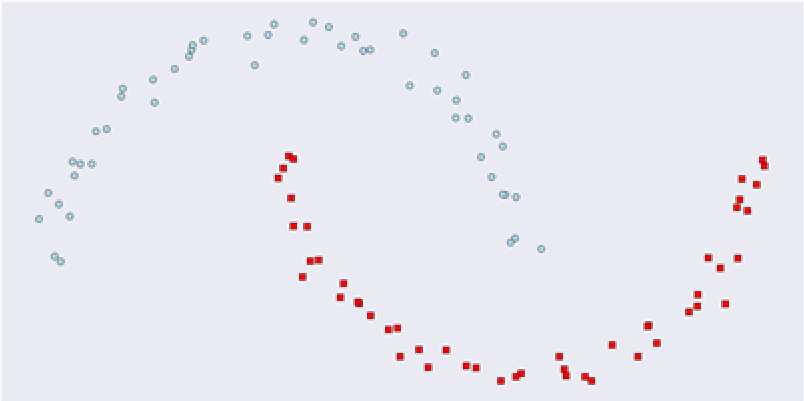
A100 GPU vs. AMD EPYC 7642 (96 logical cores)
cuML 23.04, scikit-learn 1.2.2, umap-learn 0.5.3

RAPIDS matches common Python APIs

CPU-Based Clustering

```
from sklearn.datasets import make_moons
import pandas

X, y = make_moons(n_samples=int(1e2),
                  noise=0.05, random_state=0)

X = pandas.DataFrame({'fea%d'%i: X[:, i]
                     for i in range(X.shape[1])})
```

```
from sklearn.cluster import DBSCAN
dbscan = DBSCAN(eps = 0.3, min_samples = 5)

dbscan.fit(X)

y_hat = dbscan.predict(X)
```

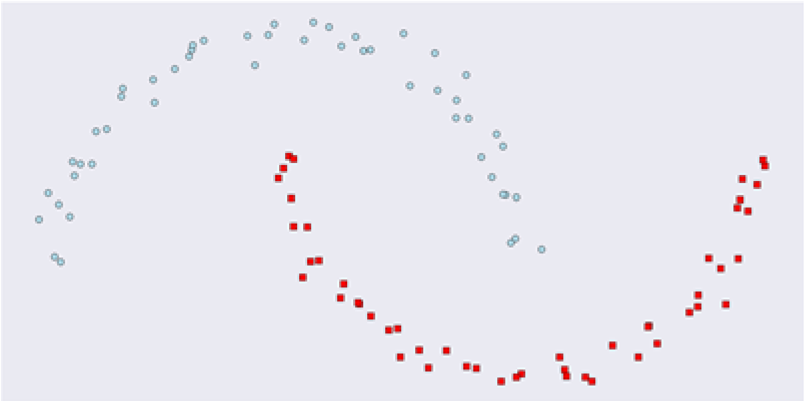


RAPIDS matches common Python APIs

GPU-Accelerated Clustering

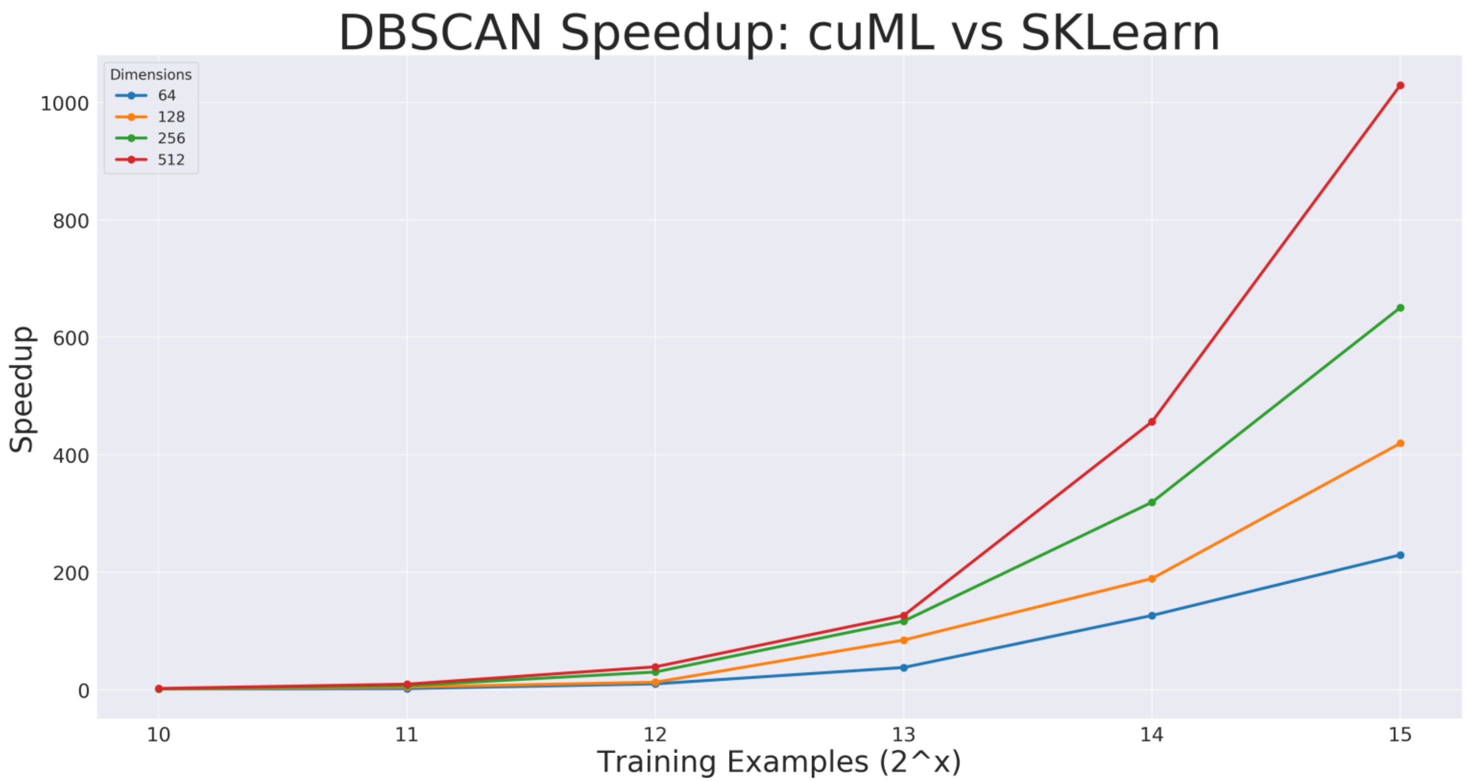
```
from sklearn.datasets import make_moons
import cudf

X, y = make_moons(n_samples=int(1e2),
                  noise=0.05, random_state=0)

X =    cudf.DataFrame({'fea%d'%i: X[:, i]
                     for i in range(X.shape[1])})
```

```
from cuml                import DBSCAN
dbscan = DBSCAN(eps = 0.3, min_samples = 5)

dbscan.fit(X)

y_hat = dbscan.predict(X)
```

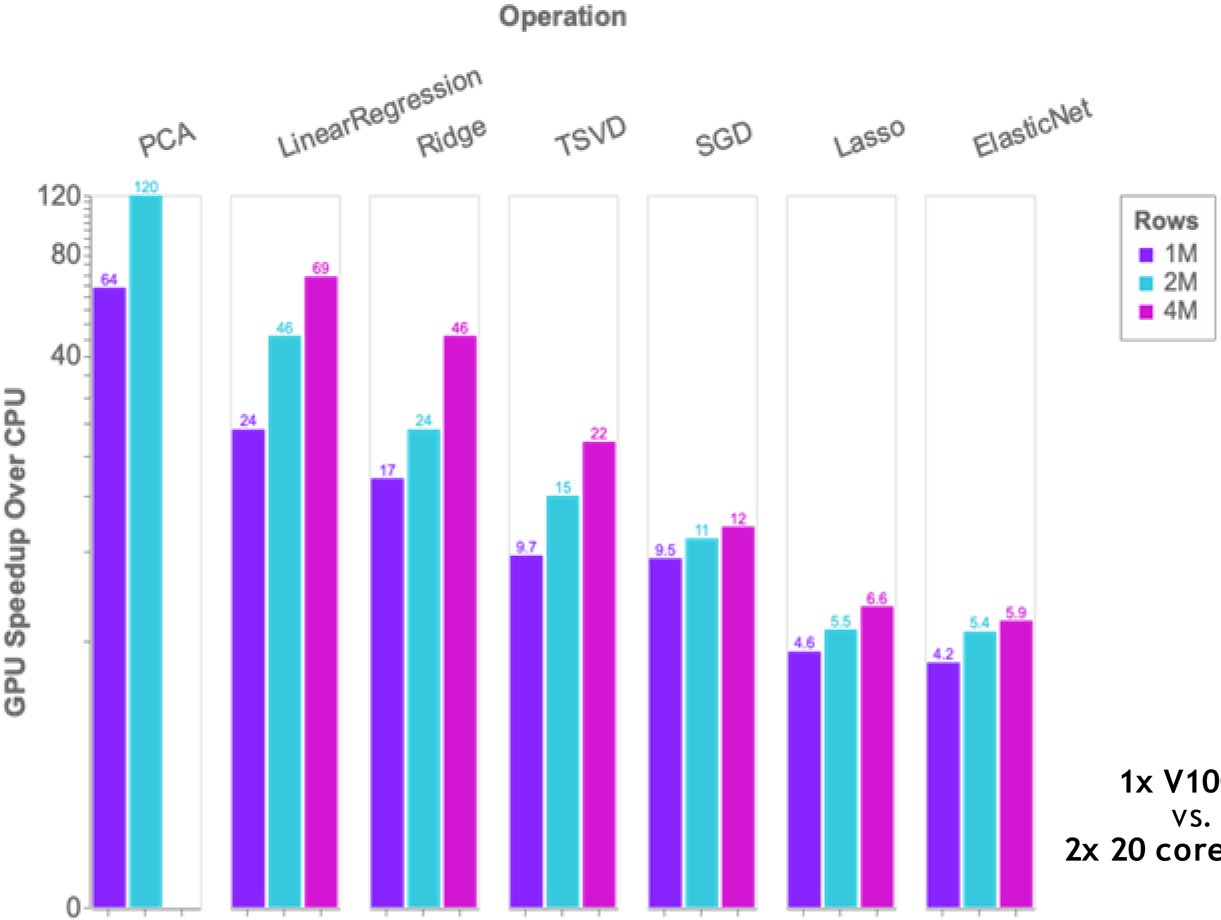


CLUSTERING

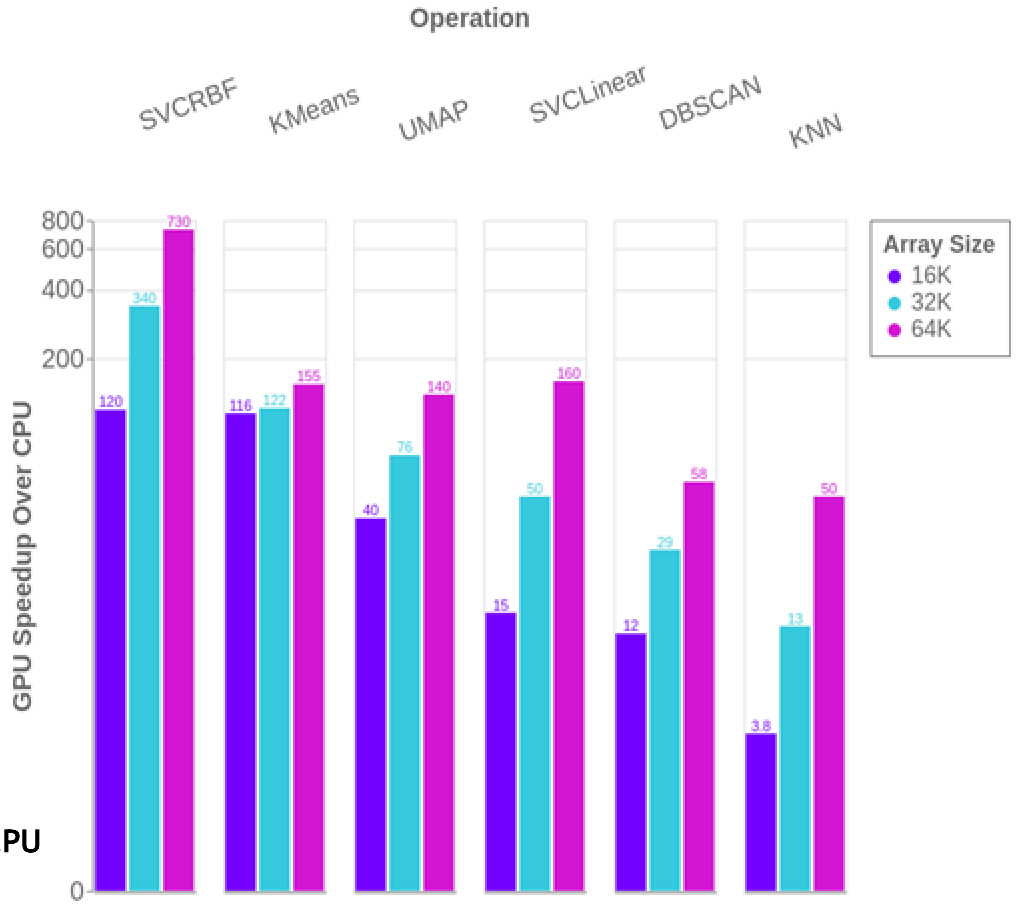
Benchmark



Benchmarks: single-GPU cuML vs scikit-learn



1x V100
vs.
2x 20 core CPU



XGBoost

Accelerated XGBoost

"XGBoost is All You Need" – Bojan Tunguz, 4x Kaggle Grandmaster

CPU

XGBoost

```
>>> from xgboost import XGBClassifier
>>> clf = XGBClassifier()
>>> clf.fit(x, y)
```

GPU

XGBoost

```
>>> from xgboost import XGBClassifier
>>> clf =
XGBClassifier(tree_method="gpu_hist")
>>> clf.fit(x, y)
```

Up to 20x Speedups

- One line of code change to unlock up to 20x speedups with GPUs
- Scalable to the world's largest datasets with Dask and PySpark
- Built-in SHAP support for model explainability
- Deployable with Triton for lighting-fast inference in production
- RAPIDS helps maintain the XGBoost project

RAPIDS ❤ dmlc XGBoost

libcudf

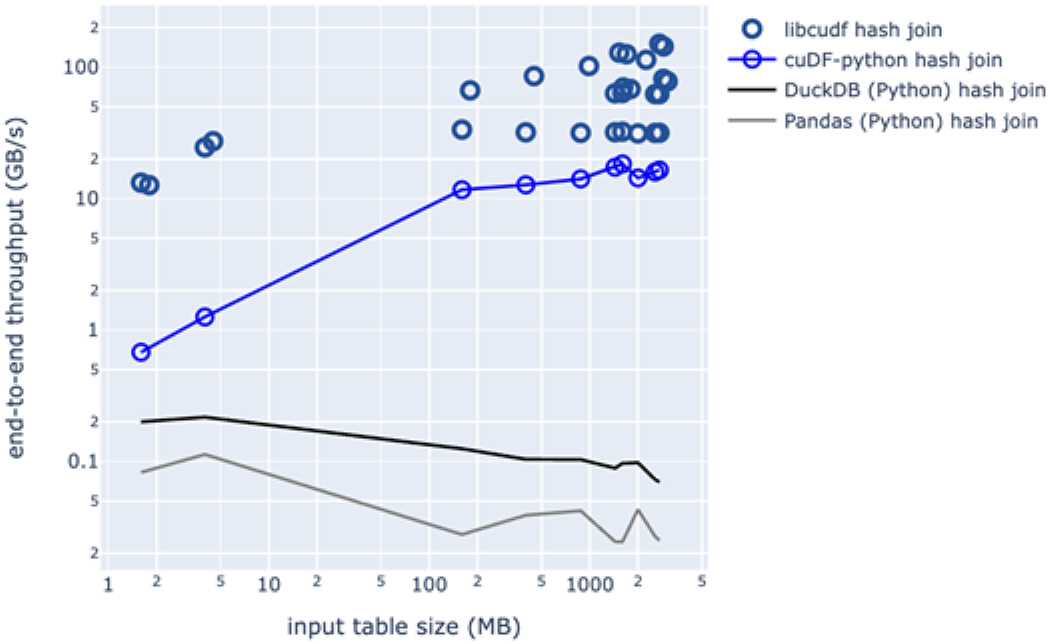
libcudf

The engine powering GPU-accelerated Apache Spark, Dask, cuDF, and high-performance data analytics

- **libcudf** is the CUDA/C++ framework for tabular data analysis
  - Data ingestion and parsing, joins, aggregations, filters, window functions, regular expressions, nested types, and more
  - Built on the Apache Arrow memory specification
  - Consistent *C++17* RAII-based APIs

- *Fastest library* for joins, aggregations, sorting, and more
  - Traditional and conditional joins
  - Nested-type sorting and aggregations

Up to 100x faster joins than DuckDB



Documentation

cuSignal

cuSignal

A GPU signal processing library interoperable with PyTorch with a SciPy Signal API

Drop-in Replacement for Real and Complex Numbers

CPU

SciPy Signal

```
>>> from scipy import signal
>>> cf = signal.resample_poly(cy,
up, down, window=("kaiser", 0.5)
```

GPU

cuSignal

```
>>> import cusignal
>>> cf = cusignal.resample_poly(cy,
up, down, window=("kaiser", 0.5)
```

Average Speed-Ups: *10-100x*

Method	SciPy Signal (ms)	cuSignal (ms)	Speedup (xN)
fftconvolve	27300	46.6	585.8
correlate	4020	28.3	142.0
resample	14700	15.4	954.5
resample_poly	2360	4.6	513.0
welch	4870	23.5	207.2
spectrogram	2520	13.2	190.9
convolve2d	8410	6.04	1392.3

- Filtering and Filter Design
- Peak Finding
- Waveform Generation
- Spectral Analysis
- Window Functions
- Wavelets
- Convolution

API Documentation   Getting Started Notebook

Data Visualization

cuxfilter and Node-RAPIDS

Visual Insight into the Largest Datasets

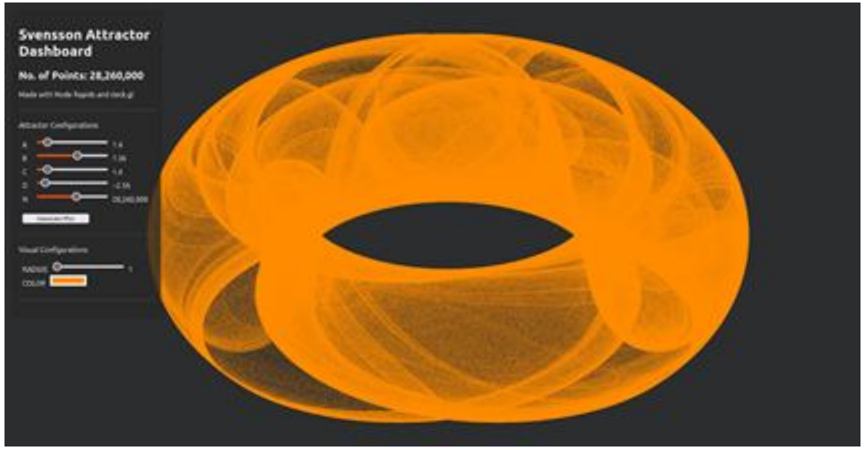
- cuxfilter makes it possible for Python users to visualize billions of points in real time *without pre-processing*

- Node-RAPIDS enables browser-based ETL and data visualization with Node.js

- Integration with common PyViz libraries

HoloViews hvPlot

Datashader bokeh plotly | Dash



Scale up and out with RAPIDS and Dask

RAPIDS and Others

Accelerated on single GPU

NumPy -> CuPy/PyTorch/..
Pandas -> cuDF
Scikit-Learn -> cuML
Numba -> Numba

Dask + RAPIDS

Multi-GPU
On single Node (DGX)
Or across a cluster

PyData

NumPy, Pandas, Scikit-Learn
and many more

Single CPU core
In-memory data

Dask

Multi-core and Distributed PyData

NumPy -> Dask Array
Pandas -> Dask DataFrame
Scikit-Learn -> Dask-ML
… -> Dask Futures

Scale Up / Accelerate

Scale out / Parallelize

Dask

Scale up and out with RAPIDS and Dask

Scale Up / Accelerate

PyData

NumPy, Pandas, Scikit-Learn
and many more

Single CPU core
In-memory data

Dask

Multi-core and Distributed PyData

NumPy -> Dask Array
Pandas -> Dask DataFrame
Scikit-Learn -> Dask-ML
… -> Dask Futures

Scale out / Parallelize

Dask Parallelizes

Natively

- **Support existing data science libraries**
  - Built on top of NumPy, Pandas, Scikit-Learn, … (easy to migrate)
  - With the same APIs (easy to train)
- **Scales**
  - Scales out to thousand-node clusters
  - Easy to install and use on a laptop
- **Popular**
  - Most common parallelism framework today at PyData and SciPy conferences
- **Deployable**
  - HPC: SLURM, PBS, LSF, SGE
  - Cloud: Kubernetes
  - Hadoop/Spark: Yarn

Parallel NumPy

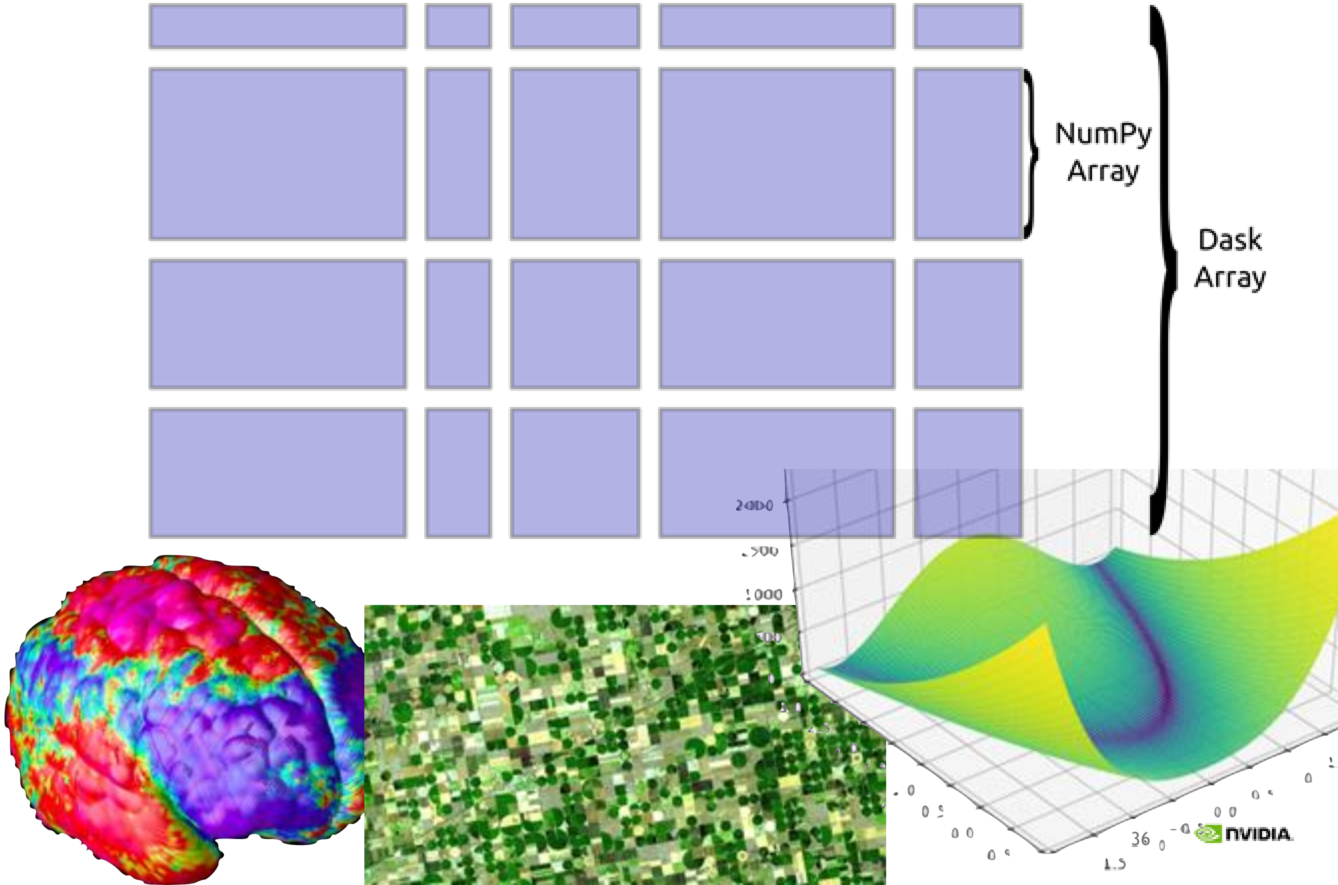
For imaging, simulation analysis, machine learning

- Same API as NumPy

  ```
  import dask.array as da
  x = da.from_hdf5(...)
  x + x.T - x.mean(axis=0)
  ```

- One Dask Array is built from many NumPy arrays

  Either lazily fetched from disk
  Or distributed throughout a cluster



Parallel Pandas

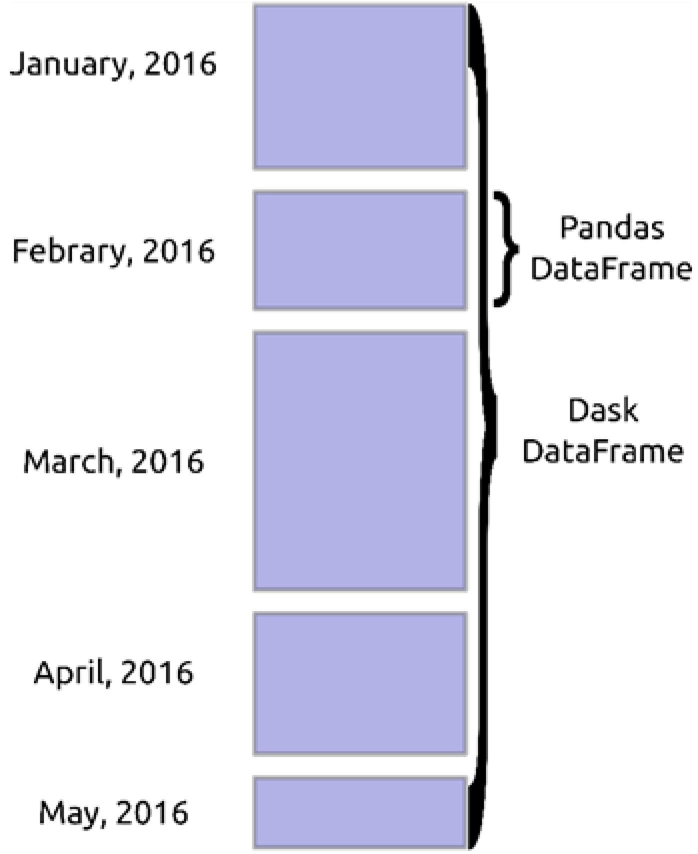
For ETL, time series, data munging

- Same API as Pandas

```
import dask.dataframe as dd
df = dd.read_csv(...)
df.groupby('name').balance.max()
```

- One Dask DataFrame is built from many Pandas DataFrames

  Either lazily fetched from disk
  Or distributed throughout a cluster



Parallel Python

For custom systems, ML algorithms, workflow engines

- Parallelize existing codebases

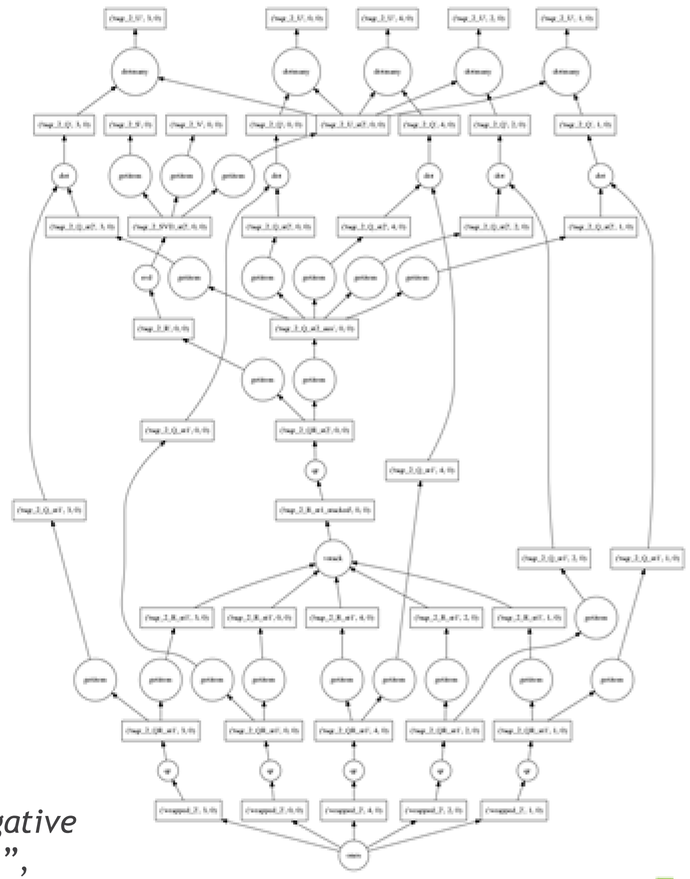
```
results = {}

for x in X:
  for y in Y:
    if x < y:
      result = f(x, y)
    else:
      result = g(x, y)
    results.append(result)
```

Parallel Python

For custom systems, ML algorithms, workflow engines

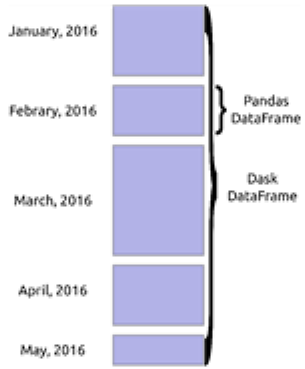
- Parallelize existing codebases

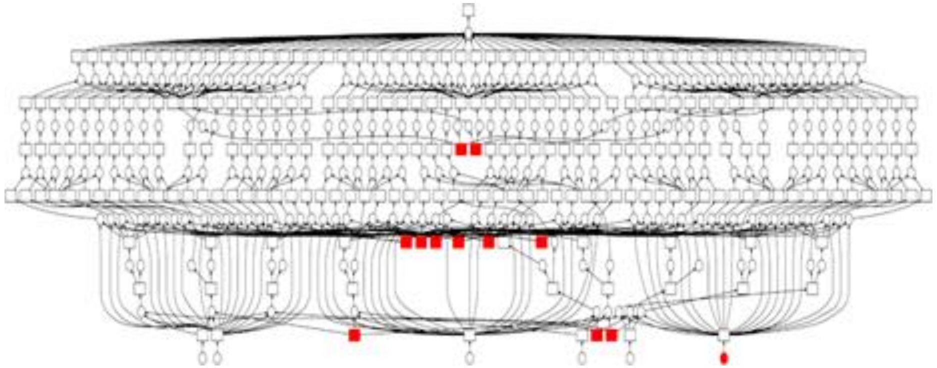
```
f = dask.delayed(f)
g = dask.delayed(g)

results = {}

for x in X:
  for y in Y:
    if x < y:
      result = f(x, y)
    else:
      result = g(x, y)
    results.append(result)

result = dask.compute(results)
```



M Tepper, G Sapiro “*Compressed nonnegative matrix factorization is fast and accurate*”, IEEE Transactions on Signal Processing, 2016

Dask Connects Python users to Hardware





User

Writes high level code (NumPy/Pandas/Scikit-Learn)

Turns into a task graph

Execute on distributed hardware

Scale up and out with RAPIDS and Dask

Scale Up / Accelerate

RAPIDS and Others

Accelerated on single GPU

NumPy -> CuPy/PyTorch/..
Pandas -> cuDF
Scikit-Learn -> cuML
Numba -> Numba

Dask + RAPIDS

Multi-GPU
On single Node (DGX)
Or across a cluster

PyData

NumPy, Pandas, Scikit-Learn
and many more

Single CPU core
In-memory data

Dask

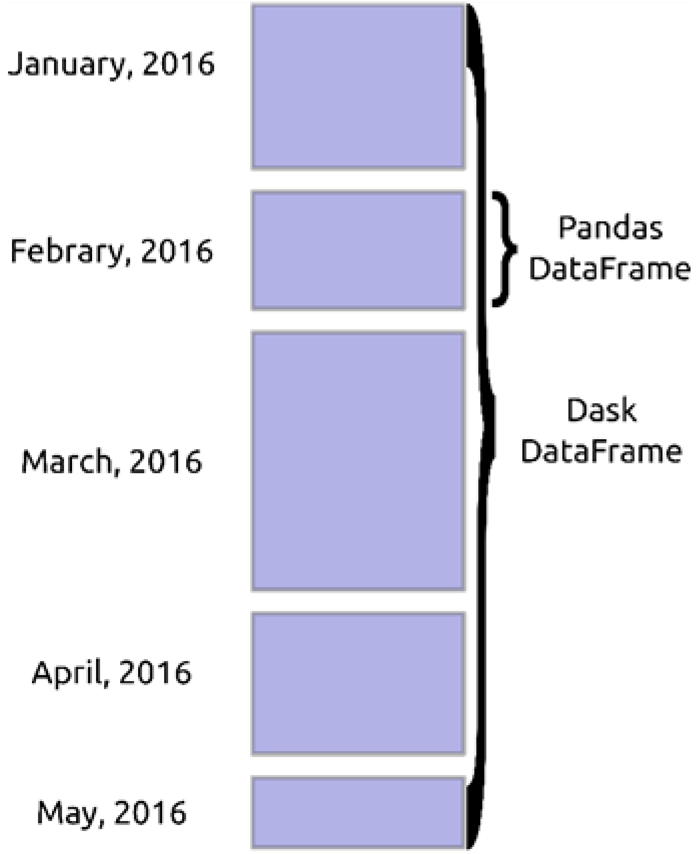
Multi-core and Distributed PyData

NumPy -> Dask Array
Pandas -> Dask DataFrame
Scikit-Learn -> Dask-ML
… -> Dask Futures

Scale out / Parallelize

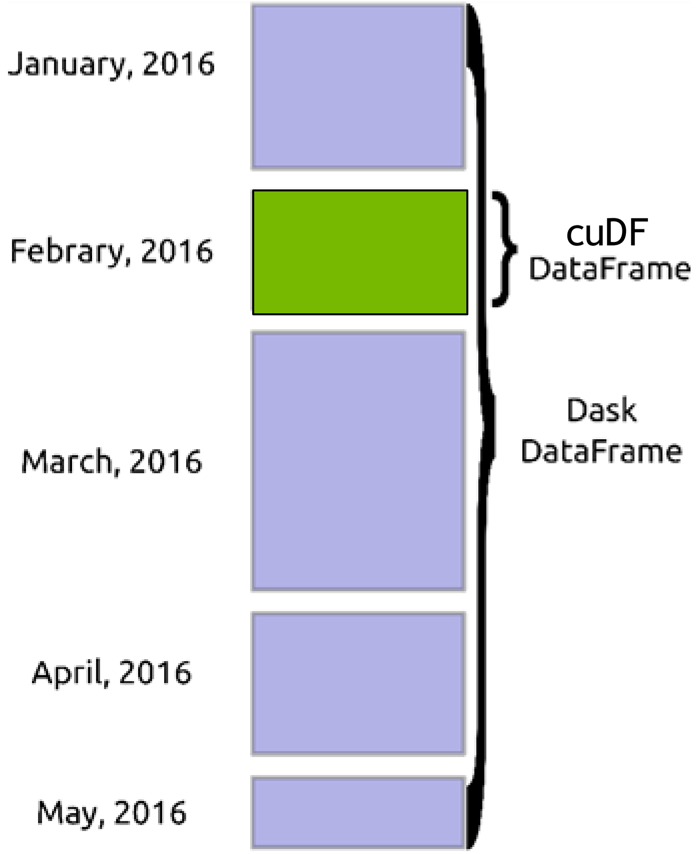
Combine Dask with cuDF

Many GPU DataFrames form a distributed DataFrame



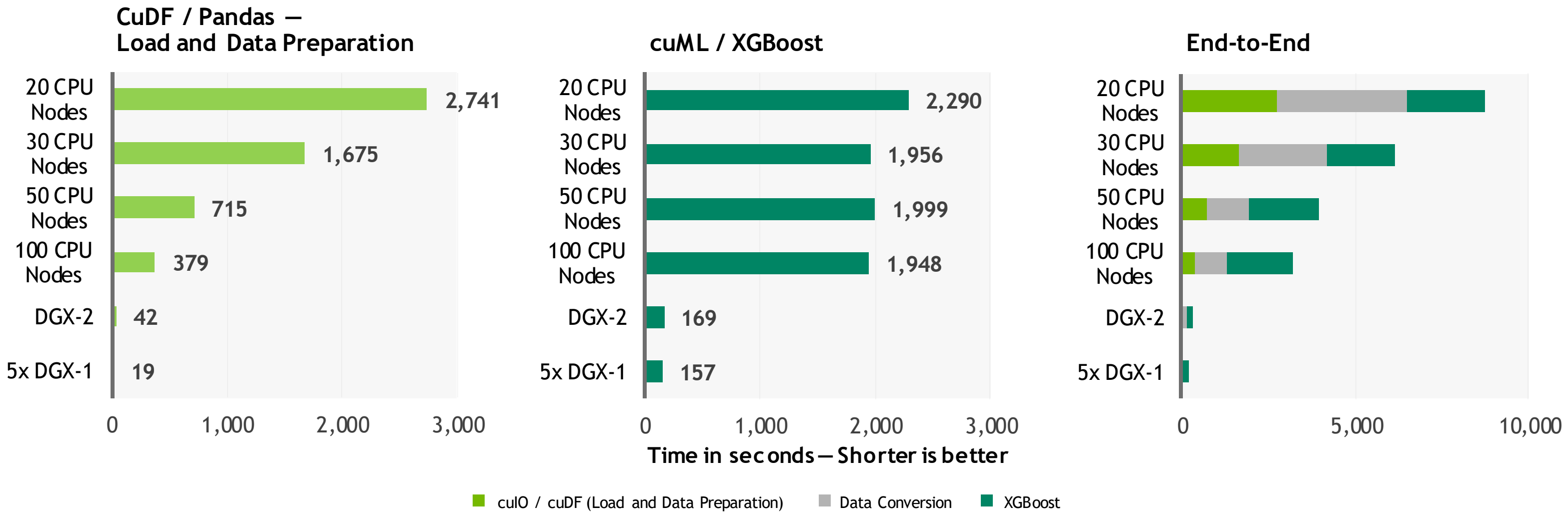
Combine Dask with cuDF

Many GPU DataFrames form a distributed DataFrame



Dask + RAPIDS

END-TO-END BENCHMARKS



Benchmark

200GB CSV dataset; Data preparation includes joins, variable transformations.

CPU Cluster Configuration

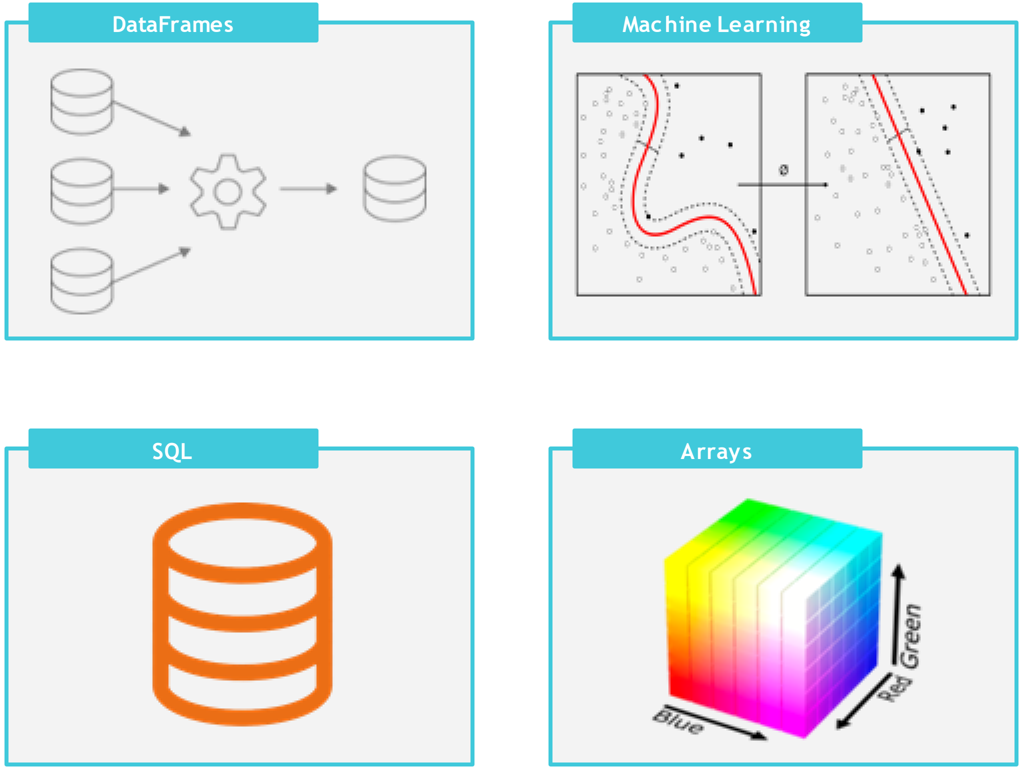
CPU nodes (61 GiB of memory, 8 vCPUs, 64-bit platform), Apache Spark

DGX Cluster Configuration

5x DGX-1 on InfiniBand network

Dask

The distributed computing framework built for the Python analytics ecosystem



- **Foundational**: Scales pandas, NumPy, Scikit-learn, XGBoost, and more
- **Familiar**: Dask matches PyData library APIs
- **Popular**: 7M+ monthly downloads, 2x growth in 2022
- **Observable**: Real-time, interactive cluster dashboards and profiling
- **Deployable**: Kubernetes, Yarn, SLURM, and all cloud platforms

[Getting Started with Dask]

Dask and RAPIDS

Drop-In Acceleration for DataFrames, SQL, and Machine Learning

CPU

Dask DataFrame

```
>>> import dask.dataframe as dd

>>> df = dd.read_parquet("file")
>>> df.groupby(“col”).mean()
>>> df.rolling(window=3).sum()
```

Dask SQL

```
>>> from dask_sql import Context
>>> c = Context()
>>> c.create_table("timeseries", df)

>>> result = c.sql("""
SELECT name, SUM(x) as "sum"
FROM timeseries
GROUP BY name
""")
```

Dask XGBoost

```
>>> import xgboost as xgb

>>> dtrain = xgb.dask.DaskDMatrix(...)
>>> output = xgb.dask.train(...,
{'tree_method': “hist”})
>>> prediction = xgb.dask.predict(client,
output, dtrain)
```

GPU

Dask DataFrame

```
>>> import dask.dataframe as dd
>>> import dask
>>> dask.config.set(
{“dataframe.backend”: “cudf”})

>>> df = dd.read_parquet("file")
>>> df.groupby(“col”).mean()
>>> df.rolling(window=3).sum()
```

Average Speedup: *10-20x*

Dask SQL

```
>>> from dask_sql import Context
>>> c = Context()
>>> c.create_table("timeseries", df,
gpu=True)

>>> result = c.sql("""
SELECT name, SUM(x) as "sum"
FROM timeseries
GROUP BY name
""")
```

Average Speedup: *10-20x*

Dask XGBoost

```
>>> import xgboost as xgb

>>> dtrain = xgb.dask.DaskDMatrix(...)
>>> output = xgb.dask.train(...,
{'tree_method': “gpu_hist”})
>>> prediction = xgb.dask.predict(client,
output, dtrain)
```

Average Speedup: *Up to 20x*

[10 Minutes to Dask and cuDF]

[Dask-SQL Getting Started]

[Dask XGBoost Getting Started]

Getting Started

Explore: RAPIDS Github

https://github.com/rapidsai

RAPIDS

RAPIDS

Open GPU Data Science

http://rapids.ai

Repositories 92 | Packages | People 135 | Teams 138 | Projects 6

Pinned repositories

cudf
cuDF - GPU DataFrame Library
Cuda ★ 2.5k 336

cuml
cuML - RAPIDS Machine Learning Library
C++ ★ 1.1k 169

cugraph
cuGraph - RAPIDS Graph Analytics Library
Cuda ★ 331 64

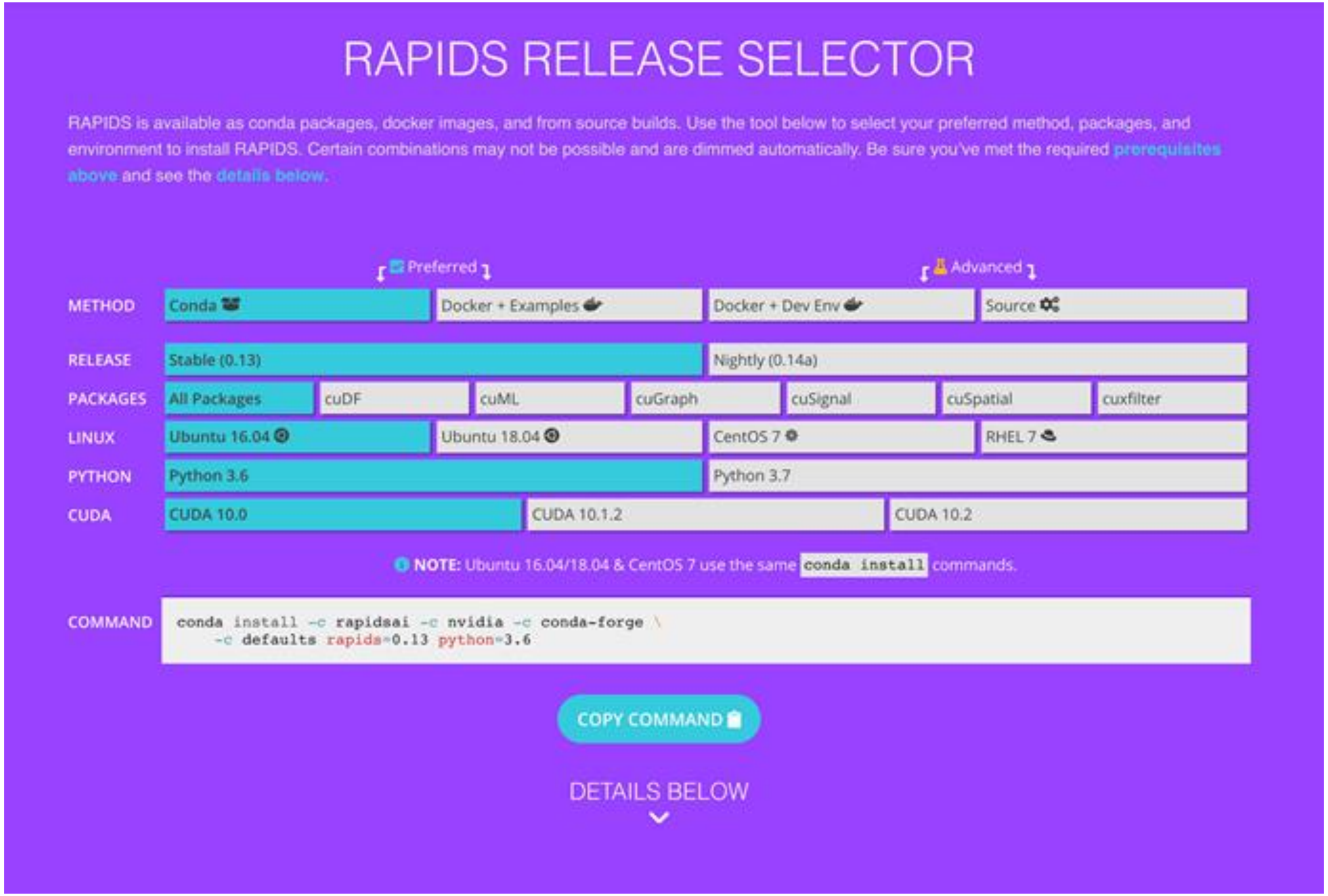
cusignal
cuSignal
Jupyter Notebook ★ 229 23

cuspatial
CUDA-accelerated GIS and spatiotemporal algorithms
Python ★ 90 21

notebooks
RAPIDS Sample Notebooks
Jupyter Notebook ★ 319 144

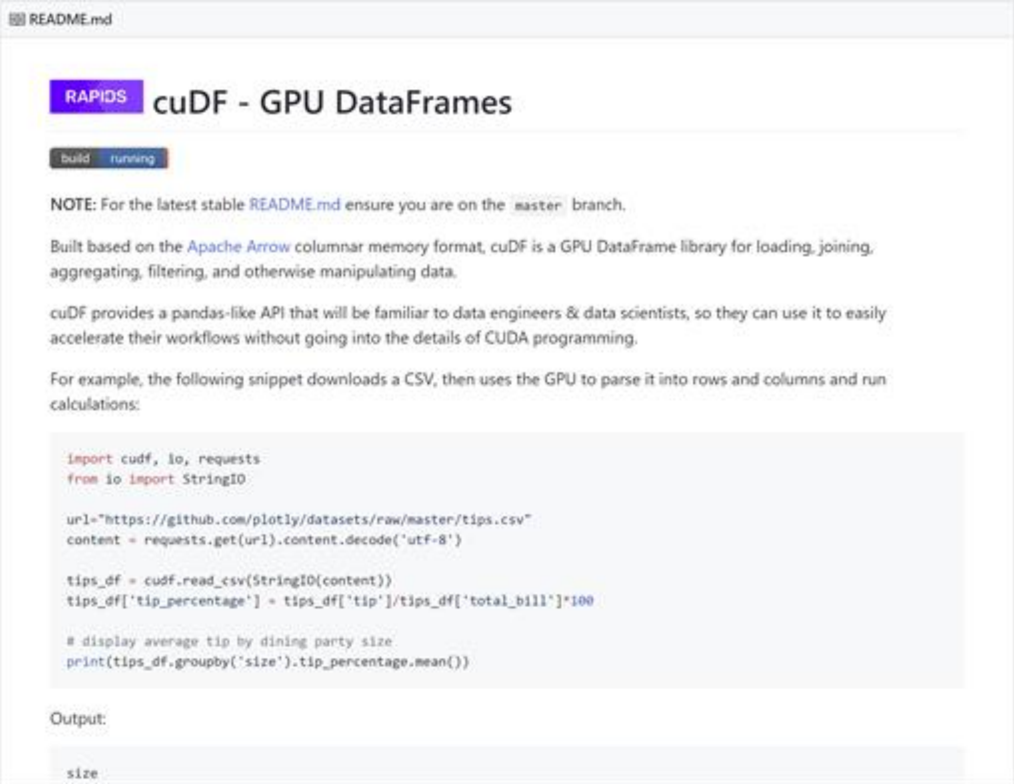
Easy Installation

Interactive Installation Guide



Explore: RAPIDS Code and Blogs

Check out our code and how we use it



README.md

RAPIDS **cuDF - GPU DataFrames**

build running

NOTE: For the latest stable README.md ensure you are on the `master` branch.

Built based on the Apache Arrow columnar memory format, cuDF is a GPU DataFrame library for loading, joining, aggregating, filtering, and otherwise manipulating data.

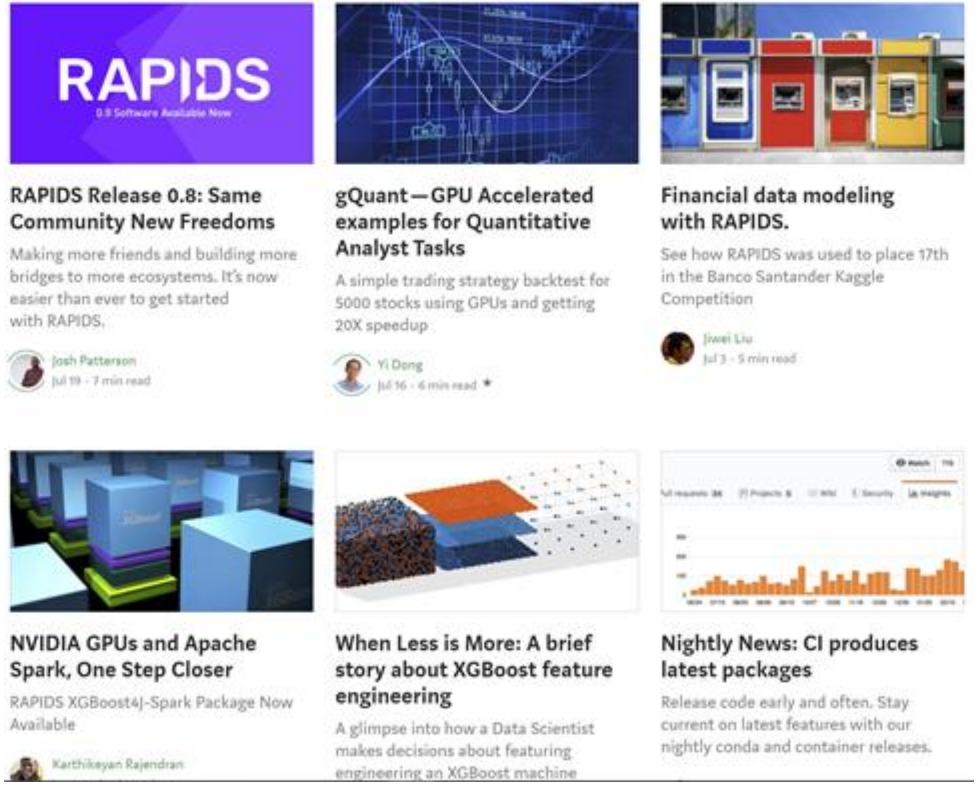
cuDF provides a pandas-like API that will be familiar to data engineers & data scientists, so they can use it to easily accelerate their workflows without going into the details of CUDA programming.

For example, the following snippet downloads a CSV, then uses the GPU to parse it into rows and columns and run calculations:

```
import cudf, io, requests
from io import StringIO

url="https://github.com/plotly/datasets/raw/master/tips.csv"
content = requests.get(url).content.decode('utf-8')

tips_df = cudf.read_csv(StringIO(content))
tips_df['tip_percentage'] = tips_df['tip']/tips_df['total_bill']*100

# display average tip by dining party size
print(tips_df.groupby('size').tip_percentage.mean())
```

Output:

```
size
```



https://github.com/rapidsai

https://medium.com/rapids-ai