

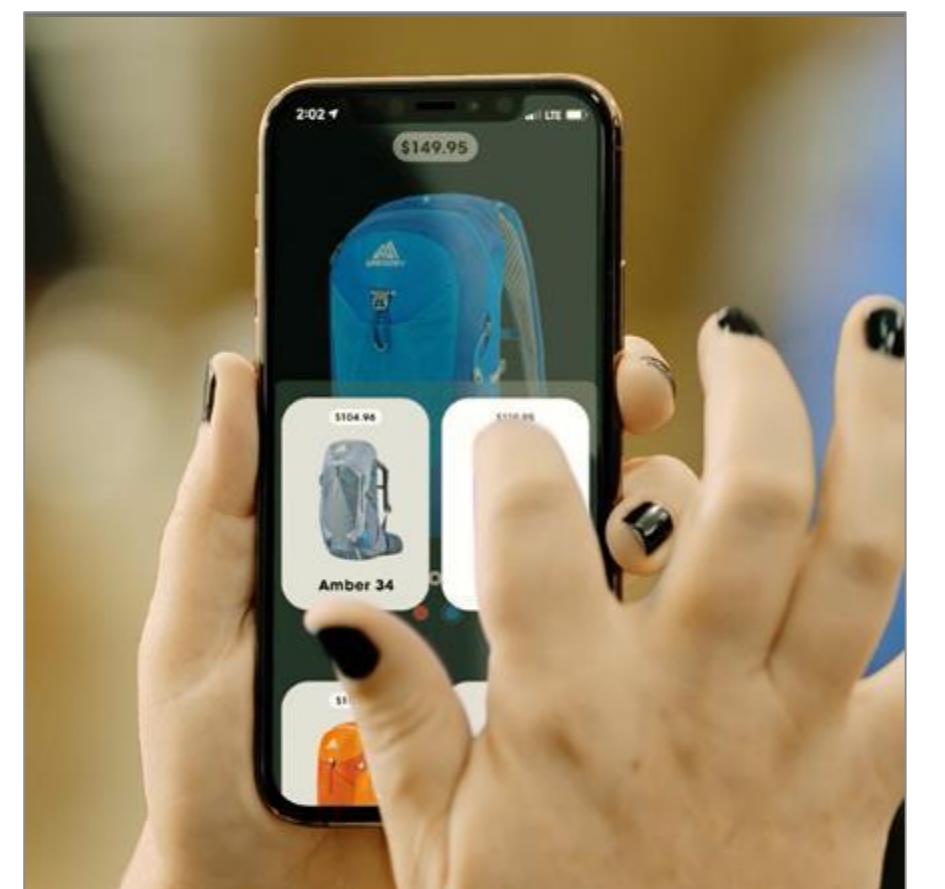


# Accelerated Data Science Everywhere

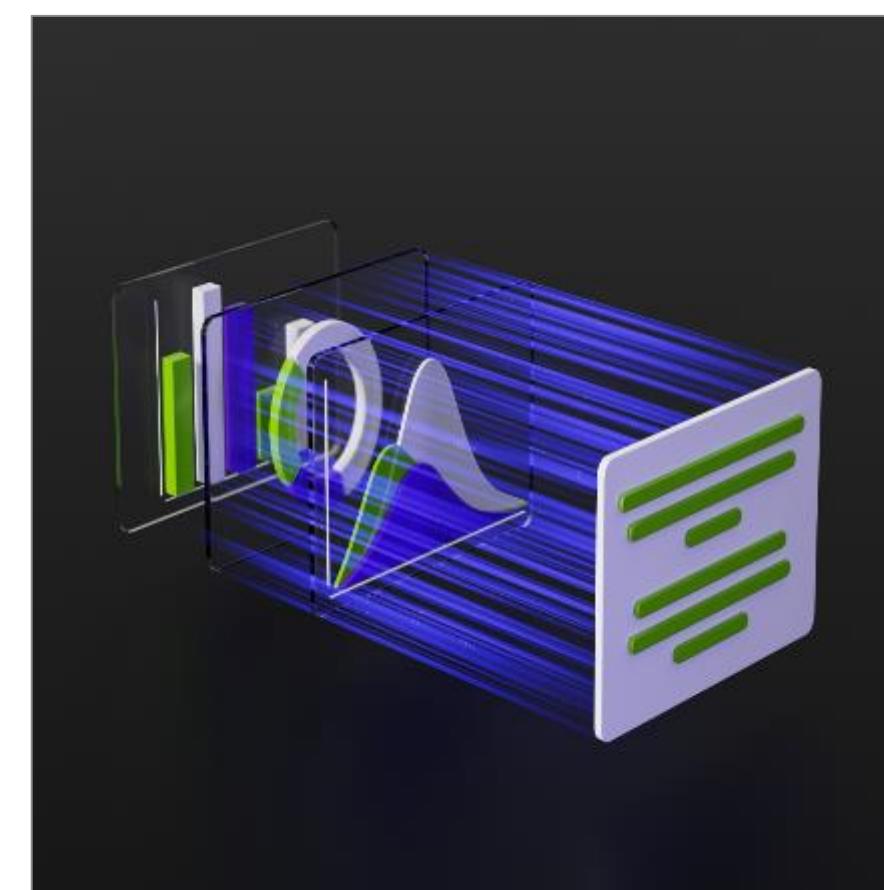
Nathan Stephens | Senior Manager, NVIDIA | ASU Research Computing Expo 2024

# Modern Enterprise Applications Need Accelerated Computing

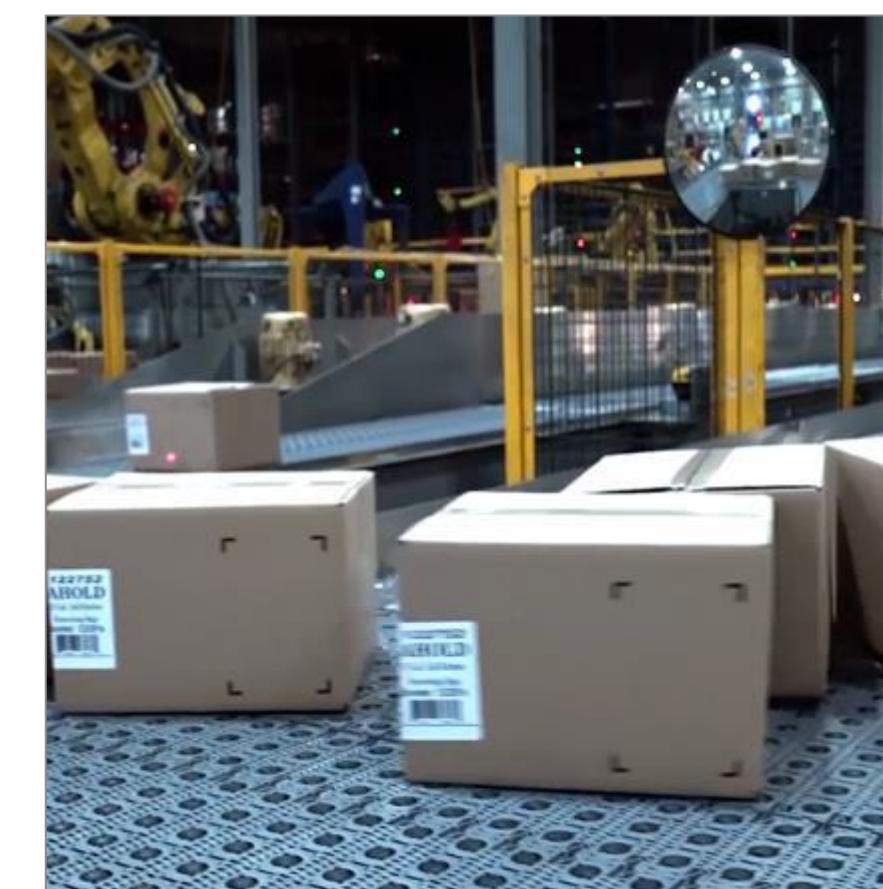
Internet scale data | Massive models | Real-time performance



Recommenders



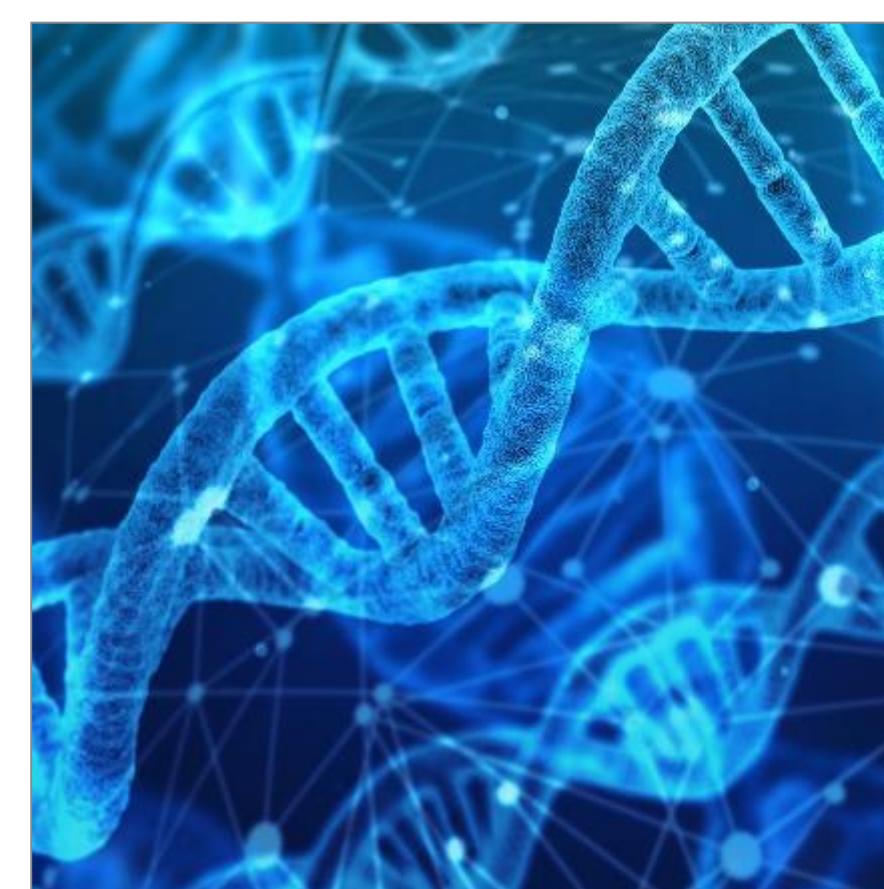
LLMs



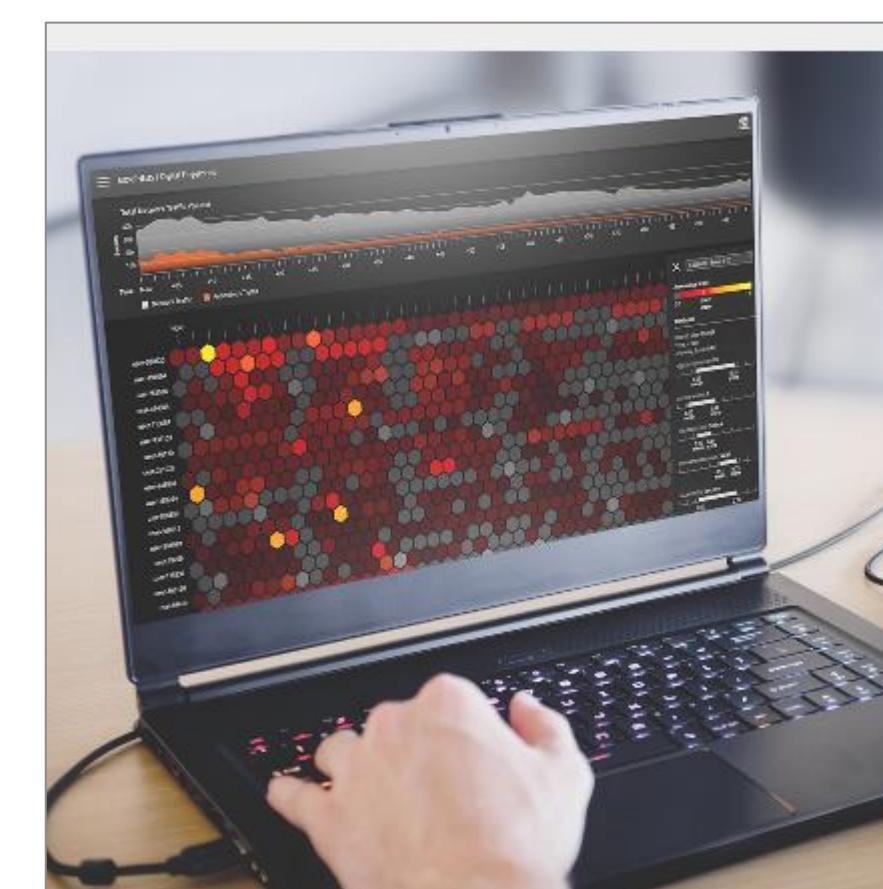
Forecasting



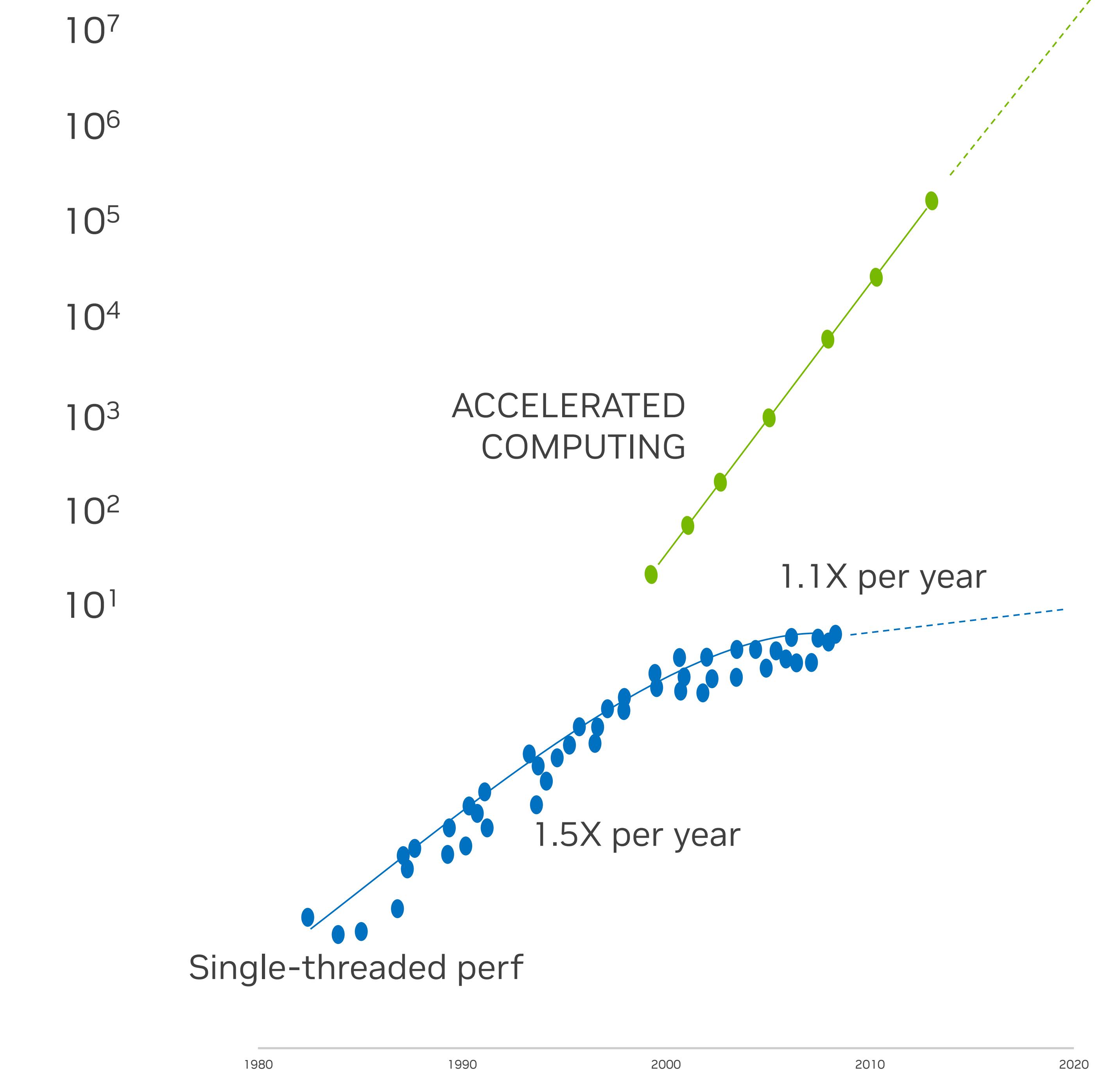
Fraud Detection



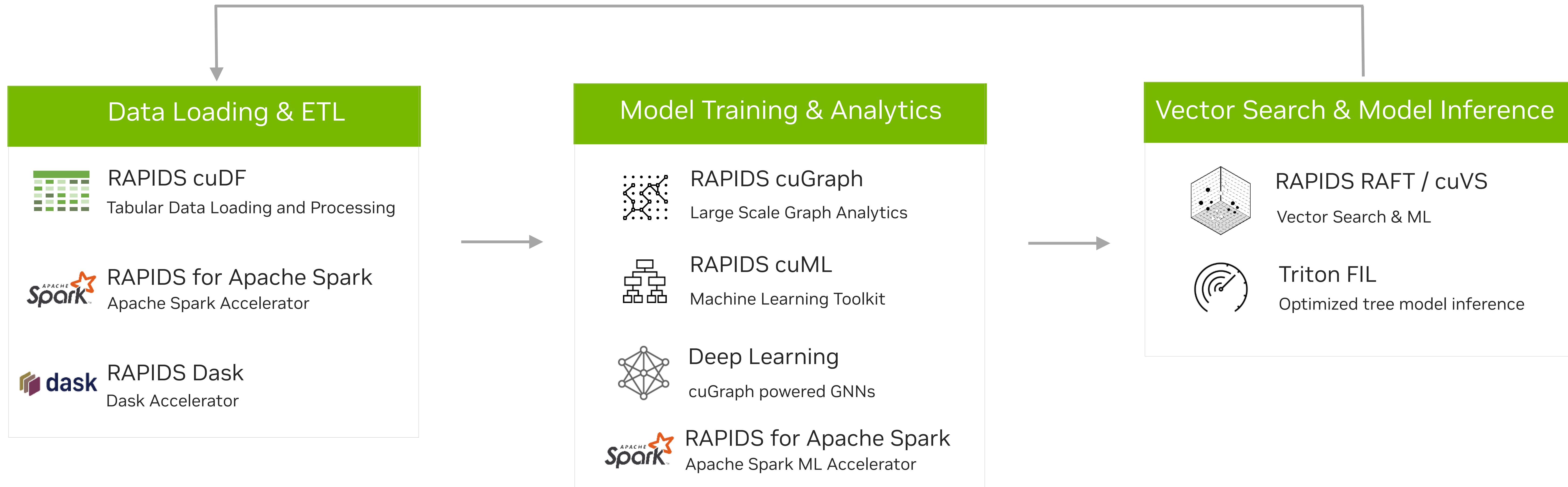
Genomic Analysis



Cybersecurity



# RAPIDS: Accelerating Data Science End-to-End

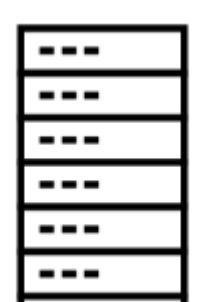


## NVIDIA AI Enterprise

Development Tools | Cloud Native Management and Orchestration | Infrastructure Optimization



Cloud



Data Center



Edge



RTX Laptop

# Accelerated Computing Swim Lanes

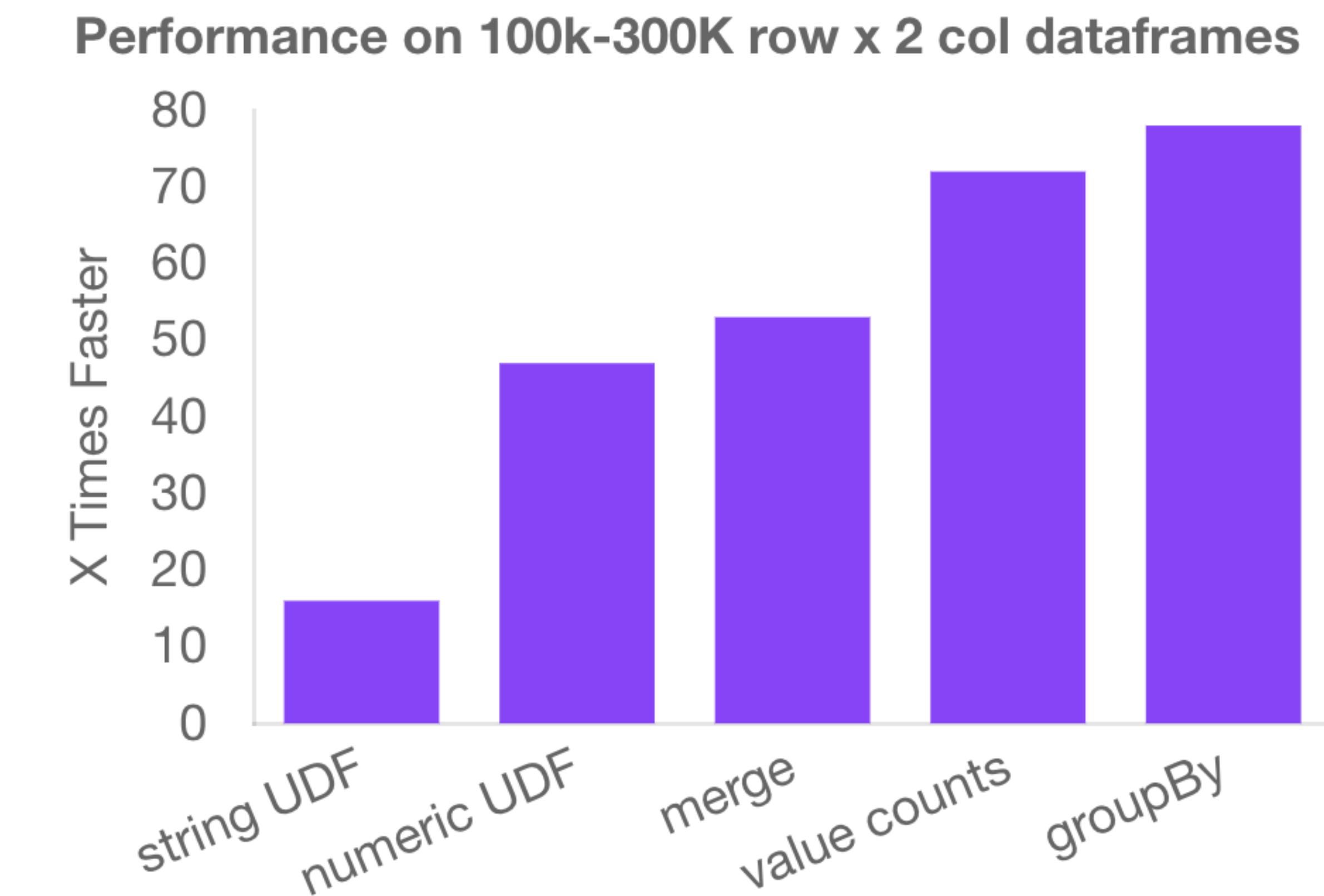
Making accelerated computing more seamless while enabling deep customization for maximum performance



# What is cuDF?

- Pandas-like data processing library for the GPU
  - Algorithms for loading, filtering, aggregating, joining
  - Numeric, datetime, categorical, string, and nested data
  - GPU accelerated I/O (incl. CSV, Parquet, ORC, JSON)
  - 10-100x faster than pandas\*
  - Provides 60-75% of the pandas API
- Built upon the **libcudf** CUDA C++ library
- Part of the wider RAPIDS ecosystem – cuML, cuGraph, cuSpatial, cuVS, and more
- Two modes of usage:
  - Standalone library (classic)
  - **cudf.pandas (this talk!)**

# RAPIDS



\* Benchmark on AMD EPYC 7642 (using 1x 2.3GHz CPU core) w/ 512GB and NVIDIA A100 80GB (1x GPU)  
w/ pandas v1.5 and cuDF v23.02

# What is cuDF?

- Pandas-like data processing library for the GPU
  - Algorithms for loading, filtering, aggregating, joining
  - Numeric, datetime, categorical, string, and nested data
  - GPU accelerated I/O (incl. CSV, Parquet, ORC, JSON)
  - 10-100x faster than pandas\*
  - Provides 60-75% of the pandas API
- Built upon the **libcudf** CUDA C++ library
- Part of the wider RAPIDS ecosystem – cuML, cuGraph, cuSpatial, cuVS, and more
- Two modes of usage:
  - Standalone library (classic)
  - **cudf.pandas (this talk!)**

```
import cudf
import 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())
```

```
%load_ext cudf.pandas # pandas operations now use the GPU!

import pandas as pd
import 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 = pd.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())
```

A large, abstract graphic on the left side of the slide features several overlapping, curved bands of color transitioning from bright lime green at the top to dark forest green at the bottom. The bands are slightly offset, creating a sense of depth and motion.

## Accelerating Pandas with Zero Code Change

- Pandas is great – let's make it fast
- cudf.pandas – what, why, and how?
- Benchmarks
- Example: Faster data question answering (cudf.pandas + LLM)
- Tips, tricks, and FAQ

# Pandas is everywhere

---

- The most used tool for working with tabular data<sup>[1]</sup>
- An ecosystem of software libraries built around pandas
- Flexible API lets you manipulate data easily
- Default tool used by major LLM-based chatbots to answer data analysis questions



[1]: StackOverflow Developer Survey, 2023

# Pandas is everywhere

---

You

How do I join two tables in Python?

ChatGPT

In Python, you can join two tables from a database using various libraries like SQLAlchemy, pandas, or even raw SQL queries with database connectors like psycopg2 for PostgreSQL, pymysql for MySQL, sqlite3 for SQLite, etc.

Here's an example of joining tables using pandas:

```
python
Copy code

import pandas as pd

# Suppose you have two tables as pandas DataFrames
df1 = pd.DataFrame({'A': [1, 2, 3], 'B': ['a', 'b', 'c']})
df2 = pd.DataFrame({'A': [1, 2, 3], 'C': ['x', 'y', 'z']})

# Joining tables based on a common column (A in this case)
result = pd.merge(df1, df2, on='A')

print(result)
```

# Performance can be limited

- Many Pandas functions are single threaded
- Not a query engine like DuckDB, Polars, PySpark, etc.,
  - Good: more interactive
  - Bad: can't do certain optimizations
- Great for prototyping and exploration, difficult to scale

## advanced questions

Input table: 100,000,000 rows x 9 columns ( 5 GB )				
duckdb-latest	0.9.1.1	2023-10-26	6s	
ClickHouse	23.10.4.25	2023-11-30	8s	
DuckDB	0.8.1.3	2023-10-26	9s	
Polars	0.19.8	2023-10-17	22s	
Datafusion	31.0.0	2023-10-24	31s	
InMemoryDataset	0.7.18	2023-10-17	38s	
data.table	1.14.9	2023-10-17	57s	
DataFrames.jl	1.6.1	2023-10-17	58s	
pandas	2.1.1	2023-10-17	277s	
dask	2023.10.0	2023-11-29	535s	
(py)datatable	1.1.0a0	2023-10-17	1288s	
dplyr	1.1.3	2023-10-17	1848s	
spark	3.5.0	2023-10-24	not yet implemented	
R-arrow	13.0.0.1	2023-10-17	not implemented	
collapse	2.0.3	2023-10-26	undefined exception	
Modin		see README	pending	
Seconds	5	10	15	20

Results of DuckDB/H2OAI database-like ops benchmark  
(groupby, 5 GB)

# Many fast alternatives to pandas

- Many alternatives:
  - Faster implementation (C++, Rust, CUDA)
  - Query engines
  - SQL-inspired
  - Distributed computing
  - Hardware accelerated (GPUs)
- Should I switch? Consider:
  - Data size and performance (pandas is getting faster!)
  - API coverage & feature growth
  - Collaboration / familiar tools
  - Compatibility & ecosystem
- If only we could bring the speed of these libraries to *pandas itself...*



*(just a handful of the many Python libraries for data science!)*

# **cudf.pandas – what, why, and how**

# cudf.pandas

The zero code change GPU accelerator for pandas built on cuDF

- Requires **no changes** to existing pandas code.

Just:

- %load\_ext cudf.pandas
- \$ python -m cudf.pandas <script.py>

- 100% of the pandas API
- Accelerates workflows up to 150x using the GPU
- Compatible with third-party libraries
- Falls back to using pandas on the CPU for unsupported functions and methods

```
%load_ext cudf.pandas
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

data = pd.read_parquet("data.parquet")
subset = data.index.indexer_between_time("09:30", "16:00")
data = data.iloc[subset]
results = data.groupby(pd.Grouper(freq="1D")).mean()

sns.lineplot(results)
plt.xticks(rotation=30)
```

# cudf.pandas: Automatic pandas acceleration

```
%time
import pandas as pd

weekday_names = {
    0: "Monday",
    1: "Tuesday",
    2: "Wednesday",
    3: "Thursday",
    4: "Friday",
    5: "Saturday",
    6: "Sunday",
}

df = pd.read_parquet("nyc_parking_violations_2022.parquet")
df["Issue Date"] = df["Issue Date"].astype("datetime64[ms]")
df["issue_weekday"] = df["Issue Date"].dt.weekday.map(weekday_names)
df.groupby(["issue_weekday"])["Summons Number"].count().sort_values()
```

CPU times: user 48.6 s, sys: 10.4 s, total: 59 s  
Wall time: 36.7 s

```
issue_weekday
Sunday      462992
Saturday    1108385
Monday      2488563
Wednesday   2760088
Tuesday     2809949
Friday       2891679
Thursday    2913951
Name: Summons Number, dtype: int64
```

```
%load_ext cudf.pandas
```

```
%time
import pandas as pd

weekday_names = {
    0: "Monday",
    1: "Tuesday",
    2: "Wednesday",
    3: "Thursday",
    4: "Friday",
    5: "Saturday",
    6: "Sunday",
}

df = pd.read_parquet("nyc_parking_violations_2022.parquet")
df["Issue Date"] = df["Issue Date"].astype("datetime64[ms]")
df["issue_weekday"] = df["Issue Date"].dt.weekday.map(weekday_names)
df.groupby(["issue_weekday"])["Summons Number"].count().sort_values()
```

CPU times: user 601 ms, sys: 141 ms, total: 742 ms  
Wall time: 720 ms

```
issue_weekday
Sunday      462992
Saturday    1108385
Monday      2488563
Wednesday   2760088
Tuesday     2809949
Friday       2891679
Thursday    2913951
Name: Summons Number, dtype: int64
```

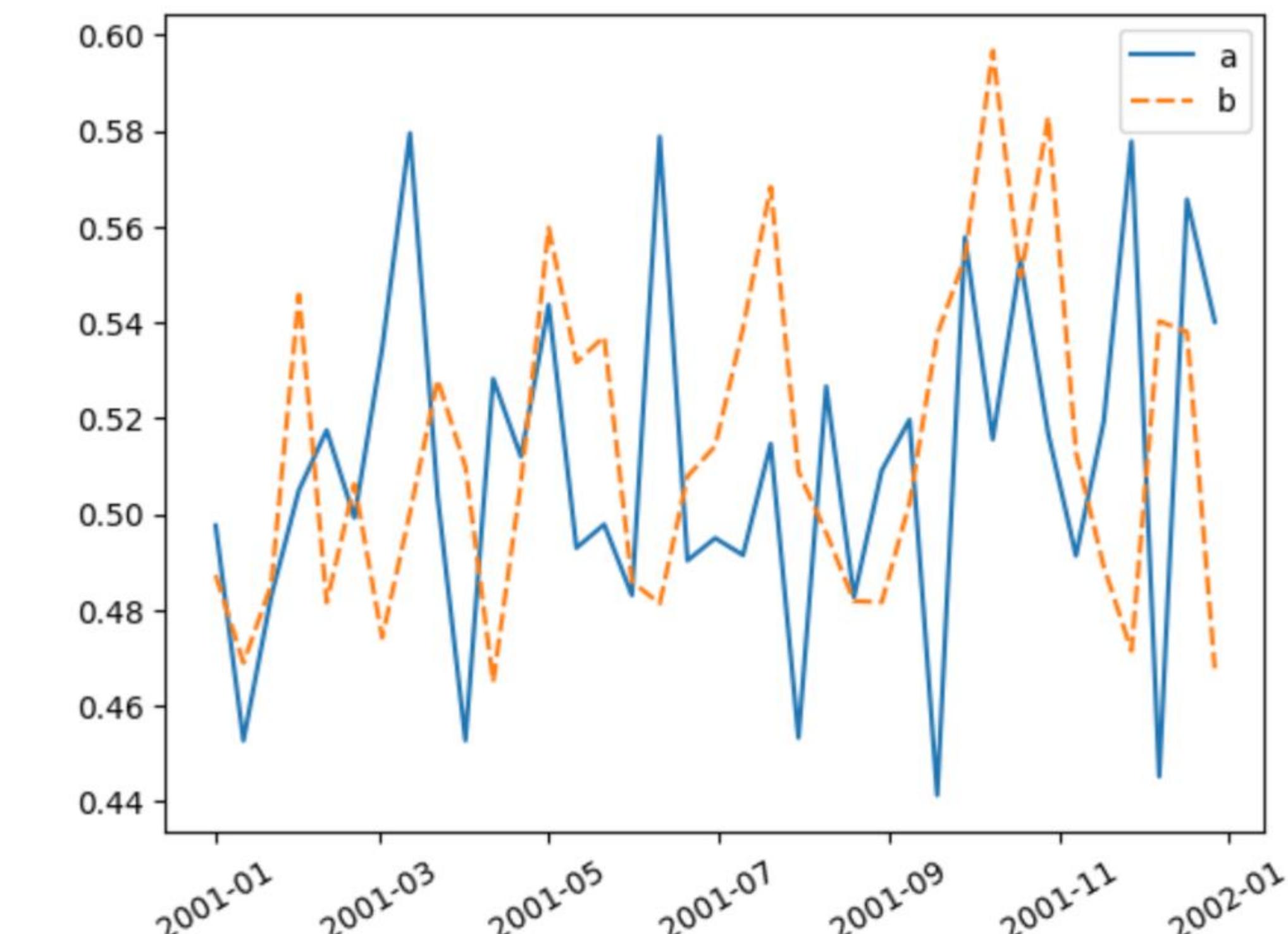
# Third-party library compatibility

- Works seamlessly with third-party libraries that expect pandas objects
- pandas operations within the libraries will also be GPU accelerated
- Integration tested with a variety of PyData libraries that use pandas, like SciPy, scikit-learn, XGBoost, Matplotlib, seaborn, HoloViews, PyTorch, TensorFlow, ...
- Not always faster
  - Depends on whether the pandas functions used are supported on the GPU

```
%load_ext cudf.pandas

import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

rng = pd.date_range("2001-01-01", "2002-01-01", freq="1h")
data = pd.DataFrame(
    {
        'a': np.random.rand(len(rng)),
        'b': np.random.rand(len(rng))
    },
    index=rng
)
data = data.iloc[data.indexer_between_time("09:30", "16:00")]
results = data.groupby(pd.Grouper(freq="10D")).mean()
_ = sns.lineplot(results)
_ = plt.xticks(rotation=30)
```



# Under the hood

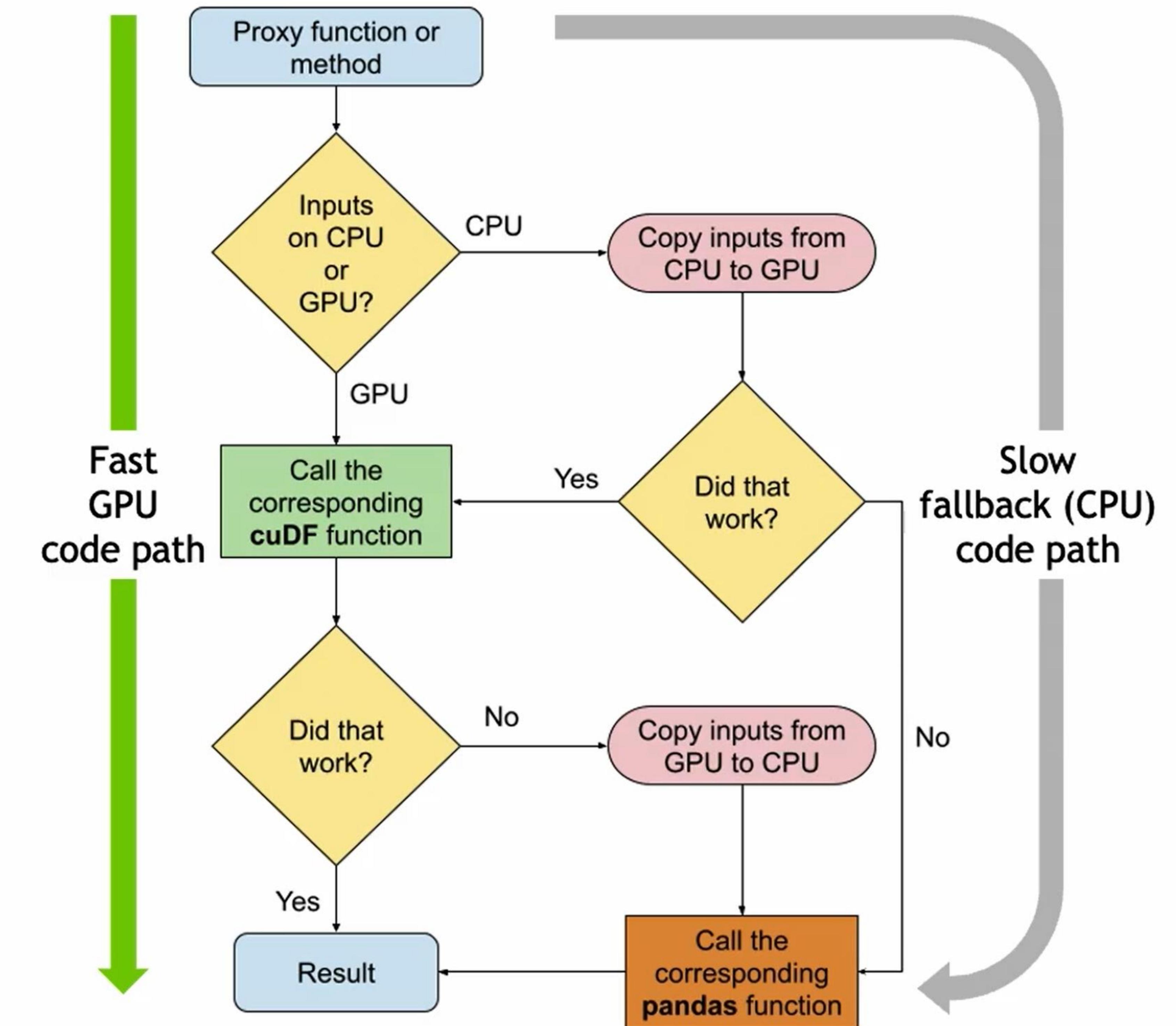
- `import pandas` imports a proxy module

```
In [1]: %load_ext cudf.pandas  
In [2]: import pandas as pd  
In [3]: print(pd)  
<module 'pandas' (ModuleAccelerator(fast=cudf,  
slow=pandas))>
```

- Proxy module composed of proxy types and functions:

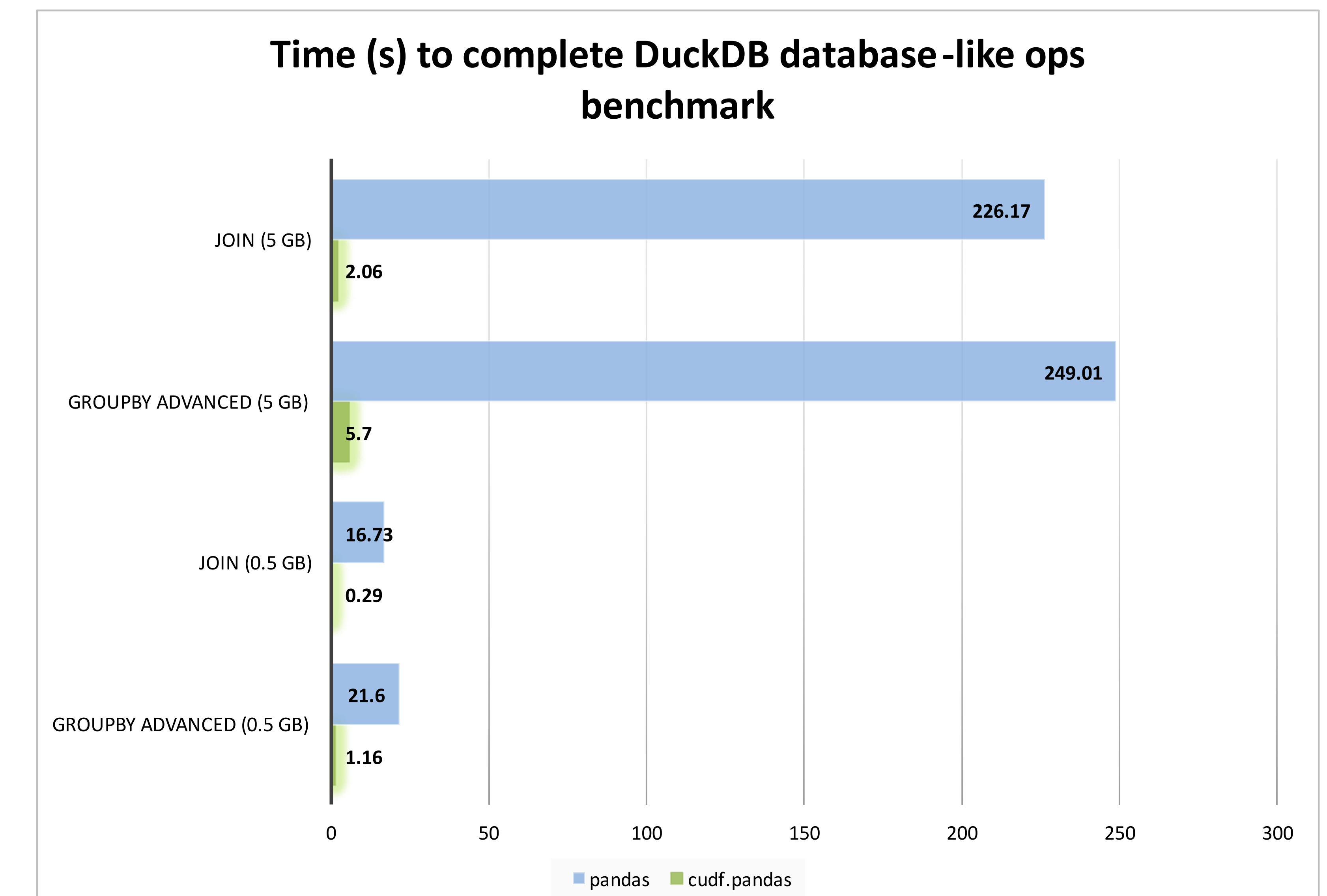
```
In [4]: print(type(pd.read_csv))  
<class 'cudf.pandas.fast_slow_proxy._FunctionProxy'>  
In [5]: print(type(pd.DataFrame))  
<class 'cudf.pandas.fast_slow_proxy._FastSlowProxyMeta'>
```

- Proxy functions and methods dispatch to cuDF or pandas



# Benchmarks

- >100x faster join and >40x faster groupby than pandas @ 5 GB on comparable CPU and GPU workstations
- pandas likely making insufficient use of available hardware
  - Parallel engines like polars or DuckDB perform much better than pandas on this benchmark
  - cudf.pandas is still faster, but by a smaller factor (1-4x)
  - *(Recently announced: Polars + cuDF collaboration on GPU engine for Polars, stay tuned!)*
- Takeaway: Existing pandas code can run orders of magnitude faster on the GPU with 0 effort



CPU: Xeon W9-3495X (1.9GHz [4.8GHz Turbo])  
GPU: RTX 5000 Ada 150W

# **Example: Faster data question answering with cudf.pandas and LLMs**

# Data question answering with pandas + LLMs

---

- LLMs can write pandas code on your behalf – just ask questions about your data
- Lots of history/knowledge about pandas
  - quality of code is reliably high
- Tools like Langchain and pandasai make them easy to use

```
import pandas as pd

from langchain.agents.agent_types import AgentType
from langchain_experimental.agents.agent_toolkits import create_pandas_dataframe_agent
from langchain_openai import ChatOpenAI

df2 = pd.read_csv("energy_use.csv")
df2 = pd.read_csv("household_info.csv")

agent = create_pandas_dataframe_agent(ChatOpenAI(temperature=0), [df1, df2], verbose=True)

agent.invoke("Which household has the highest average power consumption?")
agent.invoke("How many household names start with 'MAC0002'?")
agent.invoke("Which ACORN is the smallest consumer of power on average?")
```

# Pandas + LLMs

Analyst



DataFrame + query

PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
1	0	3	Braund, Mr. Owen Harris	male	22	1	0	AB 2117	72.0		S
2	1	1	Cumings, Mrs. John Bradley (Florence Brigitte Thayer)	female	38	1	0	PC 17599	71.2333	C85	C
3	1	3	Hikmette, Miss. Liane	female	26	0	0	STON/O2 310/322	71.05		S
4	1	1	Futrelle, Mrs. Jacques Heath (Cly May Peel)	female	35	1	0	113803	53.1	C123	S
5	0	3	Allen, Mr. William Henry	male	35	0	0	373450	8.05		S
6	0	3	Morin, Mr. James	male	35	0	0	333877	8.05		S
7	0	1	McCarthy, Mr. Timothy J.	male	54	0	0	17563	51.8625	E05	S
8	0	3	Peterson, Master. Gustav Leonard	male	2	0	1	248999	27.075		S
9	1	3	Johnson, Mrs. Oscar W. (Elisabeth Vilhelmina Berg)	female	27	0	2	347712	71.3333		S
10	1	2	Nasser, Mrs. Natacha (Makde Achrem)	female	16	1	0	237735	30.0008	C	S
11	1	3	Sandstrom, Miss. Marguerite Isold	female	4	1	1	PP 3945	16.7	GB	S
12	1	3	Bartell, Miss. Elizabeth	female	58	0	0	113813	28.65	C703	S
13	0	3	Santesson, Mr. William Henry	male	20	0	0	AB 2151	8.05		S
14	0	4	Anthonette, Mr. Andrew Johnson	male	38	1	0	343499	19.15		S

"How many passengers have more than 3 siblings?"

Answer

"30 people have more than 3 siblings"

Agent



LangChain

LLM



OpenAI

Generate code from metadata and query

PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
1	0	3	Braund, Mr. Owen Harris	male	22	1	0	AB 2117	72.0		S
2	1	1	Cumings, Mrs. John Bradley (Florence Brigitte Thayer)	female	38	1	0	PC 17599	71.2333	C85	C
3	1	3	Hikmette, Miss. Liane	female	26	0	0	STON/O2 310/322	71.05		S
4	1	1	Futrelle, Mrs. Jacques Heath (Cly May Peel)	female	35	1	0	113803	53.1	C123	S
5	0	3	Allen, Mr. William Henry	male	35	0	0	373450	8.05		S
6	0	3	Morin, Mr. James	male	35	0	0	333877	8.05		S
7	0	1	McCarthy, Mr. Timothy J.	male	54	0	0	17563	51.8625	E05	S
8	0	3	Peterson, Master. Gustav Leonard	male	2	0	1	248999	27.075		S
9	1	3	Johnson, Mrs. Oscar W. (Elisabeth Vilhelmina Berg)	female	27	0	2	347712	71.3333		S
10	1	2	Nasser, Mrs. Natacha (Makde Achrem)	female	16	1	0	237735	30.0008	C	S
11	1	3	Sandstrom, Miss. Marguerite Isold	female	4	1	1	PP 3945	16.7	GB	S
12	1	3	Bartell, Miss. Elizabeth	female	58	0	0	113813	28.65	C703	S
13	0	3	Santesson, Mr. William Henry	male	20	0	0	AB 2151	8.05		S
14	0	4	Anthonette, Mr. Andrew Johnson	male	38	1	0	343499	19.15		S

"How many passengers have more than 3 siblings?"

Execute code to answer the query

```
>>> df[df['SibSp'] > 3].shape[0]  
30
```

Generated pandas code

```
df[df['SibSp'] > 3].shape[0]
```

Just `%load_ext cudf.pandas` to accelerate on GPU!

# Faster data question answering

energy\_use.csv: ~160MM rows

acorns.csv: ~5K rows

```
%load_ext cudf.pandas
import pandas as pd
from langchain.agents.agent_types import AgentType
from langchain_experimental.agents.agent_toolkits import create_pandas_dataframe_agent
from langchain_openai import ChatOpenAI
df2 = pd.read_csv("energy_use.csv")
df2 = pd.read_csv("household_info.csv")
agent = create_pandas_dataframe_agent(OpenAI(temperature=0), [df1, df2], verbose=True)
agent.invoke("Which household has the highest average power consumption?")
agent.invoke("How many household names start with 'MAC002'?")
agent.invoke("Which ACORN is the smallest consumer of power on average?")
```

Query	Task	Pandas (CPU only)	cudf.pandas (CPU+GPU)
"Which household has the highest average power consumption?"	Groupby aggregation	13s	3s
"How many household names start with 'MAC002'?"	String operations	44s	3s
"Which ACORN is the smallest consumer of power?"	Join	49s	11s

# **FAQ's, Tips, and Tricks**

# FAQs, tips and tricks

## ***“When should I use cuDF (direct import) versus cudf.pandas?”***

- Use cudf.pandas if
  - You have existing pandas code and you want to run it on GPUs with 0 effort
  - The ability to run the same code on GPU-enabled as well as CPU-only systems is important
- Use cuDF (direct import) if:
  - You want everything to run on GPU (CPU fallback is prohibitively expensive)
  - You need functionality that cuDF provides but pandas does not

```
%load_ext cudf.pandas # pandas operations now use the GPU!

import pandas as pd
import 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 = pd.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())
```

```
import cudf
import 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())
```

# FAQs, tips and tricks

---

## ***“How do you ensure pandas compatibility?”***

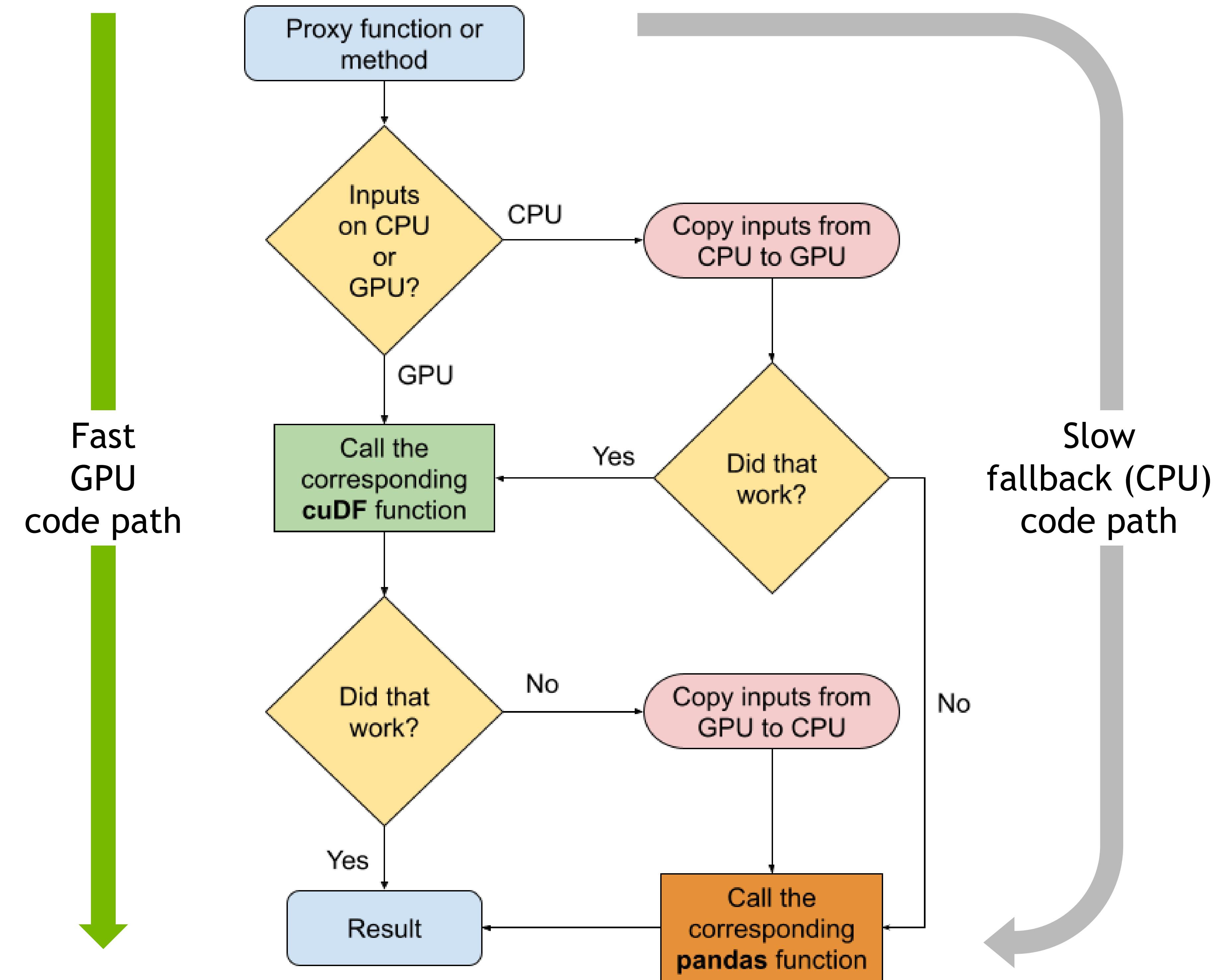
- We run the entire pandas unit test suite with cudf.pandas enabled
  - ~94% of the tests passing – a few minor differences
- We turn on cuDF’s “pandas compatibility mode” (ensures result ordering matches pandas, etc.)

```
cudf.set_option("mode.pandas_compatible", True)
```

# FAQs, tips and tricks

## ***Keep data on the GPU***

- Every function or method will attempt to run on the GPU first, and if that fails, will run on the CPU
- CPU fallback involves copying data between CPU and GPU – twice in the worst case
- As much as possible, you should use GPU-supported operations



# FAQs, tips and tricks

## The `%cudf.pandas.profile`

- Use the `%cudf.pandas.profile` cell magic in Jupyter, or import it directly:

```
from cudf.pandas import Profiler

with Profiler() as p:
    # code goes here

p.print_per_function_stats()
```

- Shows which functions ran on the CPU and which ran on the GPU
  - Note: doesn't report CPU<->GPU transfer time

```
%%cudf.pandas.profile

rng = pd.date_range("2023-01-01", "2023-02-01", freq="10ms")
data = pd.DataFrame(
{
    "a": np.random.rand(len(rng)),
    "b": np.random.rand(len(rng))
},
index=rng
)
data = data.iloc[rng.indexer_between_time("09:30", "16:00")]
results = data.groupby(pd.Grouper(freq="1D")).mean()
results.head()
```

Total time elapsed: 12.855 seconds  
11 GPU function calls in 1.322 seconds  
1 CPU function calls in 4.416 seconds  
*Stats*

Function	GPU ncalls	GPU cumtime	GPU percall	CPU ncalls	CPU cumtime	CPU percall
date_range	1	0.008	0.008	0	0.000	0.000
DatetimeIndex.__len__	2	0.000	0.000	0	0.000	0.000
DataFrame	2	0.873	0.436	0	0.000	0.000
DatetimeIndex.indexer_between...	0	0.000	0.000	1	4.416	4.416
_DataFrameIlocIndexer.__geti...	1	0.127	0.127	0	0.000	0.000
Grouper	1	0.000	0.000	0	0.000	0.000
DataFrame.groupby	1	0.021	0.021	0	0.000	0.000
DataFrameResampler.mean	1	0.259	0.259	0	0.000	0.000
DataFrame.head	1	0.001	0.001	0	0.000	0.000
DataFrame.__repr__	1	0.033	0.033	0	0.000	0.000

Not all pandas operations ran on the GPU. The following functions required CPU fallback:

- DatetimeIndex.indexer\_between\_time

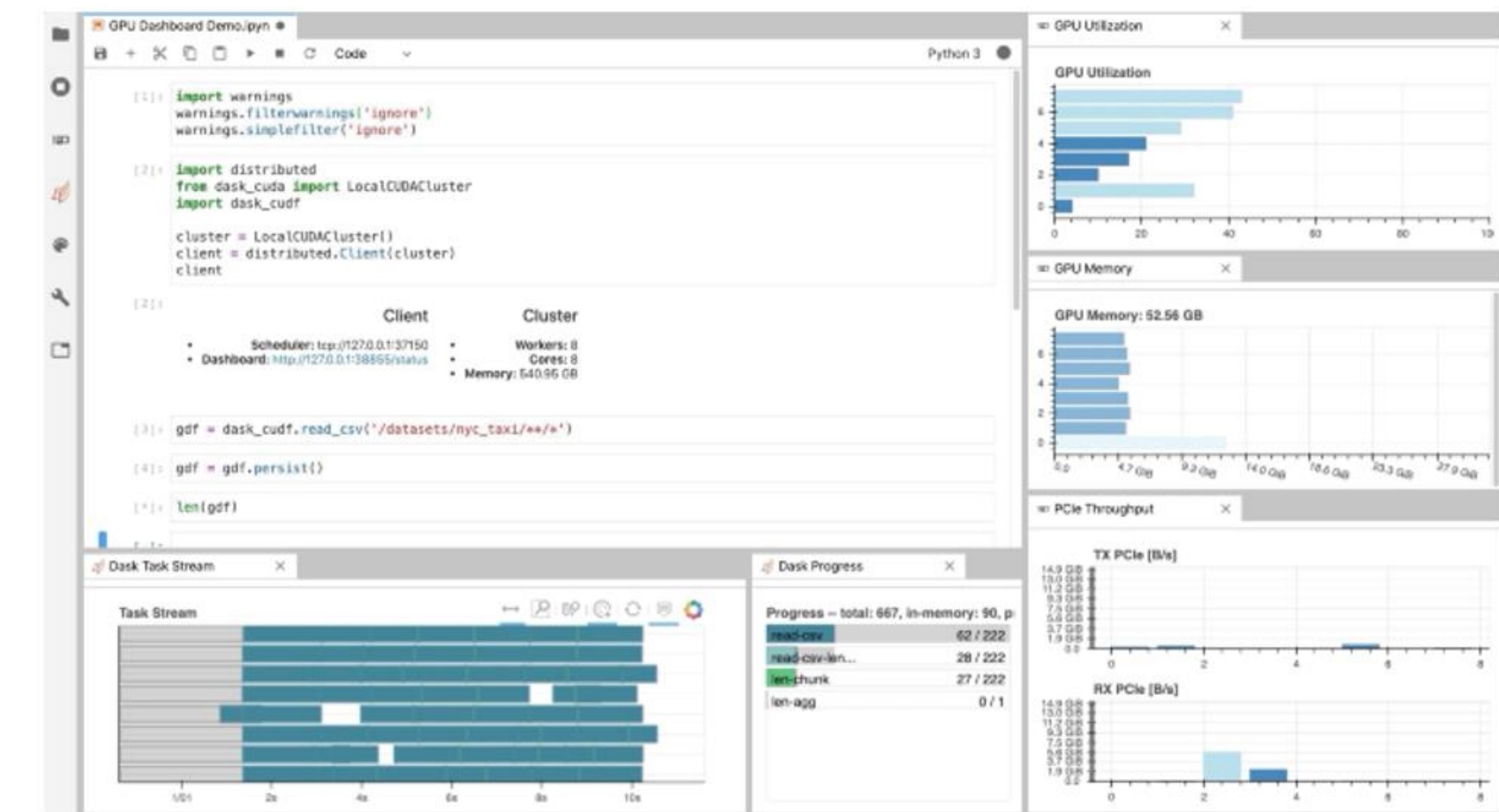
To request GPU support for any of these functions, please file a Github issue here:

<https://github.com/rapidsai/cudf/issues/new/choose>.

# FAQs, tips and tricks

## ***GPU memory is limited – use it efficiently!***

- Often, GPU memory is limited compared to CPU memory (RAM)
- cudf.pandas will fall back to using the CPU when you run out of GPU memory – unexpected slowdown!
- Important to manage GPU memory effectively:
  - Keep only the data you need
  - Filter early and often
  - Store sparse data efficiently
  - Monitor GPU usage



jupyterlab-nvdashboard plugin: measures GPU activity and memory usage

# FAQs, tips and tricks

## *Use idiomatic pandas patterns*

- Express computations in terms of native pandas operations rather than custom UDFs
  - These are better supported by the GPU
- Often, rewriting this way yields faster CPU-only performance as well
  - It's a win-win!

```
%%time

def func(x: str) -> int:
    count = 0
    for char in x:
        if char == "a":
            count += 1
    return count

s.apply(func).head()

CPU times: user 3.95 s, sys: 163 ms, total: 4.11 s
Wall time: 4.11 s
0    0
1    0
2    2
3    0
4    1
Name: s, dtype: int64
```

```
%%time
s.str.count("a").head()

CPU times: user 780 ms, sys: 74.3 ms, total: 854 ms
Wall time: 846 ms
0    0
1    0
2    2
3    0
4    1
Name: s, dtype: int32
```

# Demo: Accelerated pandas

cudf.pandas: the zero-code change GPU accelerator for pandas

The screenshot shows a Jupyter Notebook interface with the following details:

- Title Bar:** The notebook is titled "rapids-cudf-pandas-accelerator-mode.ipynb".
- Toolbar:** Includes "File", "Edit", "View", "Insert", "Runtime", "Tools", "Help", and a status message "All changes saved".
- Header Buttons:** "Comment", "Share", "Settings", and a user profile icon.
- Code/Text Buttons:** "+ Code" and "+ Text".
- Connect:** Shows "T4" connected.
- Cell Type:** A text cell.
- Content:**

10 Minutes to RAPIDS cuDF's pandas accelerator mode (cudf.pandas)

cuDF is a Python DataFrame library (built on the Apache Arrow columnar memory format) for loading, joining, aggregating, filtering, and otherwise manipulating tabular data using a DataFrame style API in the style of pandas.

cuDF now provides a pandas accelerator mode (`cudf.pandas`), allowing you to bring accelerated computing to your pandas workflows without requiring any code change.

This notebook is a short introduction to `cudf.pandas`.



## Accelerating Polars with zero code change

- New!
- Up to 13x speed up over CPUs
- The Polars GPU engine enables users to process 100s of millions of rows of data in seconds

# Many fast alternatives to pandas

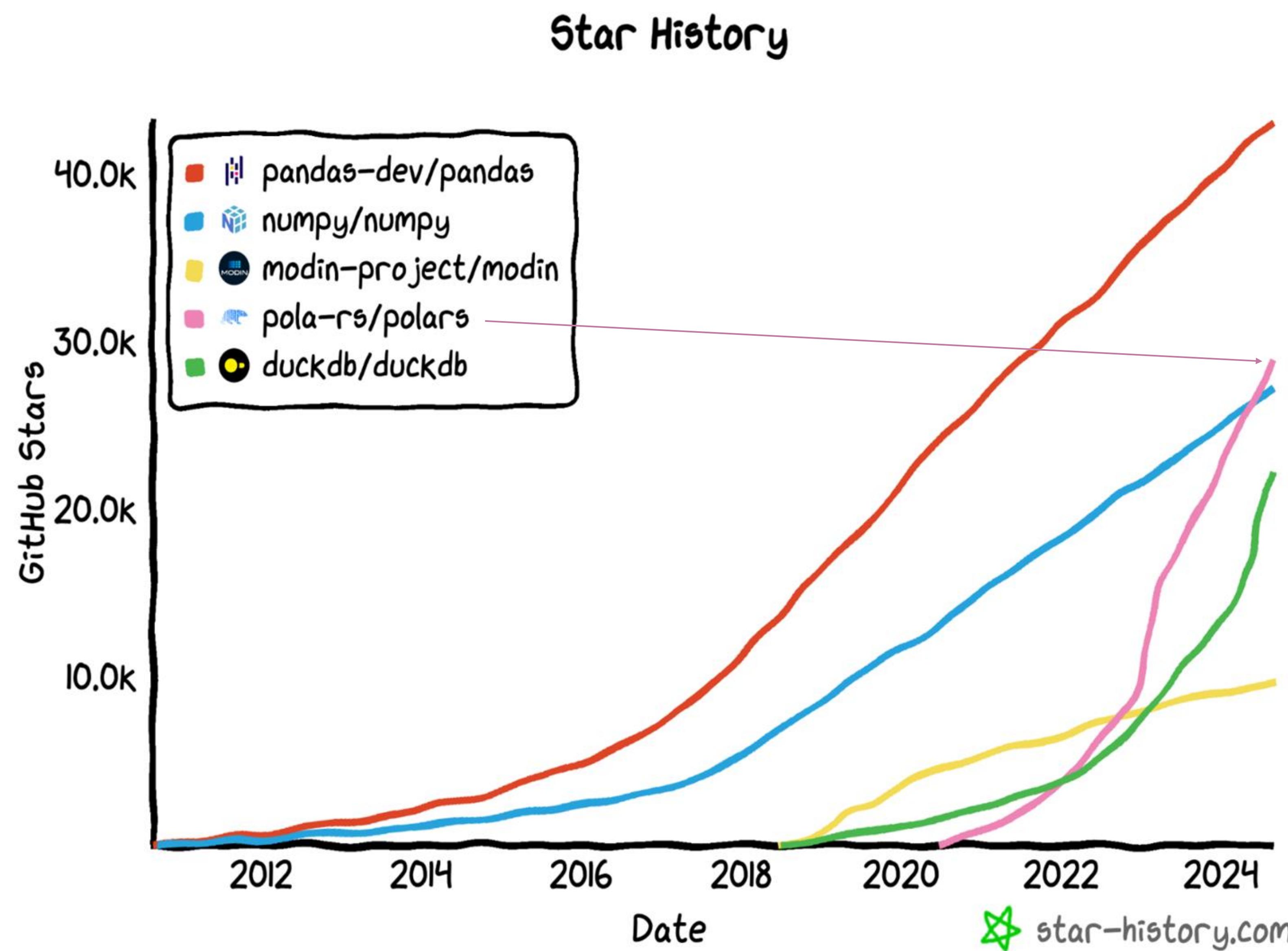
- Many alternatives:
  - Faster implementation (C++, Rust, CUDA)
  - Query engines
  - SQL-inspired
  - Distributed computing
  - Hardware accelerated (GPUs)
- Should I switch? Consider:
  - Data size and performance (pandas is getting faster!)
  - API coverage & feature growth
  - Collaboration / familiar tools
  - Compatibility & ecosystem



*(just a handful of the many Python libraries for data science!)*

# Polars: Fastest Growing Data Analytics Library

Open Source | Tens to Hundreds of GB Data | Single Machine



## Polars API

Polars Optimizer

multi threaded vectorized engines

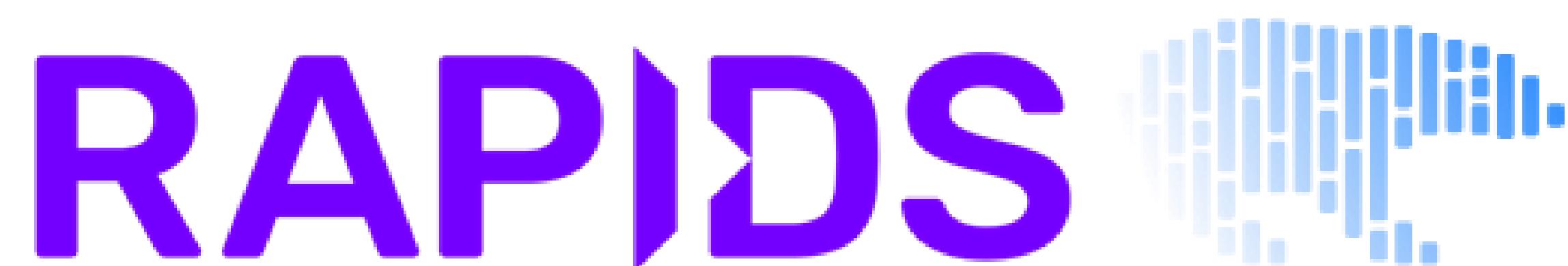


Async  
runtime

Native  
I/O

# Announcing: Polars is Accelerated with RAPIDS cuDF

100s of millions of rows in seconds



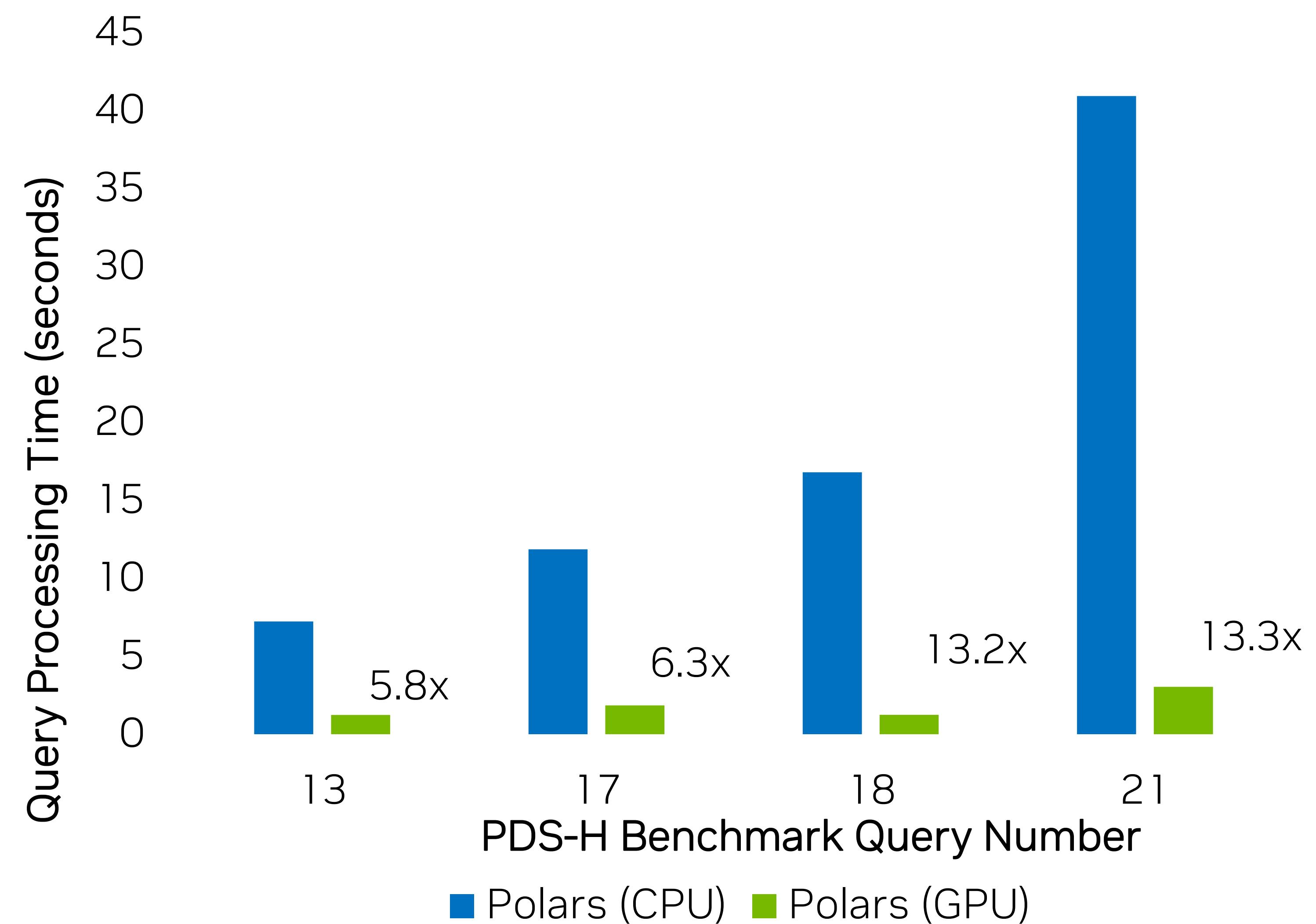
1. Zero code change (set: engine="gpu" in *collect* operation)

2. Compatible with the ecosystem of tools built for Polars

3. Graceful CPU fallback for unsupported queries

4. Available as beta version

Accelerate Polars up to 13x



# Zero Code Change



```
import polars as pl

(transactions
    .group_by("CUST_ID").agg(pl.col("AMOUNT").sum())
    .sort(by="AMOUNT", descending=True)
    .head()
    .collect())
```

cuDF OFF



```
import polars as pl

(transactions
    .group_by("CUST_ID").agg(pl.col("AMOUNT").sum())
    .sort(by="AMOUNT", descending=True)
    .head()
    .collect(engine="gpu"))
```

cuDF ON

# Demo: Accelerated Polars

The zero-code change GPU accelerator for Polars

The screenshot shows a Jupyter Notebook interface with the following details:

- Title Bar:** Shows the file name "polars\_gpu\_engine\_demo.ipynb" with a star icon.
- Toolbar:** Includes standard options like File, Edit, View, Insert, Runtime, Tools, Help, Comment, Share, and Settings.
- Runtime Selection:** Shows "T4" GPU selected with "RAM" and "Disk" options.
- Sidebar:** Features icons for Code (+), Text (+), Search (magnifying glass), and a workspace (x).
- Section Header:** A collapsed section titled "Introduction to the Polars GPU Engine".
- Text Content:**
  - An introduction stating: "Polars is a popular single machine DataFrame library powered by an OLAP Query Engine. Beginning in the v1.3 release, Polars can now leverage NVIDIA GPUs for even higher performance through its GPU engine (powered by RAPIDS cuDF)."
  - A note about design: "Designed to make processing 10-100+ GBs of data feel interactive with just a single GPU, this new engine is built directly into the Polars Lazy API – just pass `engine="gpu"` to the `collect` operation."
  - A statement about performance: "The GPU engine fully utilizes the Polars optimizer to ensure efficient execution and minimal memory usage, is compatible with the ecosystem of tools built for Polars, and has graceful CPU fallback for unsupported queries."
  - A final note: "This notebook is a short introduction to the Polars GPU engine, powered by cuDF."

**RAPIDS docs & installation guide:**  
<https://rapids.ai>

**Try cudf.pandas now on Colab:**  
<https://nvda.ws/rapids-cudf>

**Try cudf and polars now on Colab:**  
<https://nvda.ws/4eKIWZW>