高速データフレームライブラリ FireDucksの紹介

2024/11/28

石坂一久

PyData Fukuoka Meetup #21

アジェンダ

1. FireDucksの紹介（30分）
2. 質疑（10分）
3. ハンズオン（10分）
4. 質疑（10分）

自己紹介

石坂 一久（福岡生まれ）

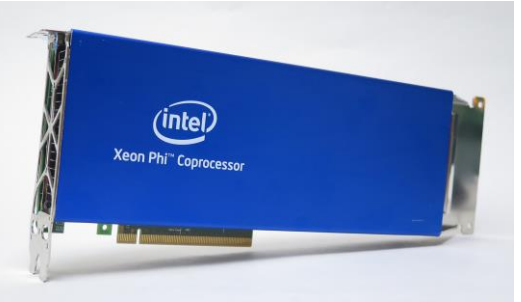
（NEC セキュアシステムプラットフォーム研究所所属）

＜これまでの関わってきた主な領域＞

自動並列化コンパイラ

並列処理・ベクトル処理

Intel Xeon Phi
（メニコア）



NEC SX-Aurora TSUBASA
（スパコン）

→ DataFrameコンパイラを作って，pandasを速くしよう！

https://pc.watch.impress.co.jp/docs/news/yajiuma/1238340.html
https://jpn.nec.com/hpc/sxauroratsubasa/specification/index.html

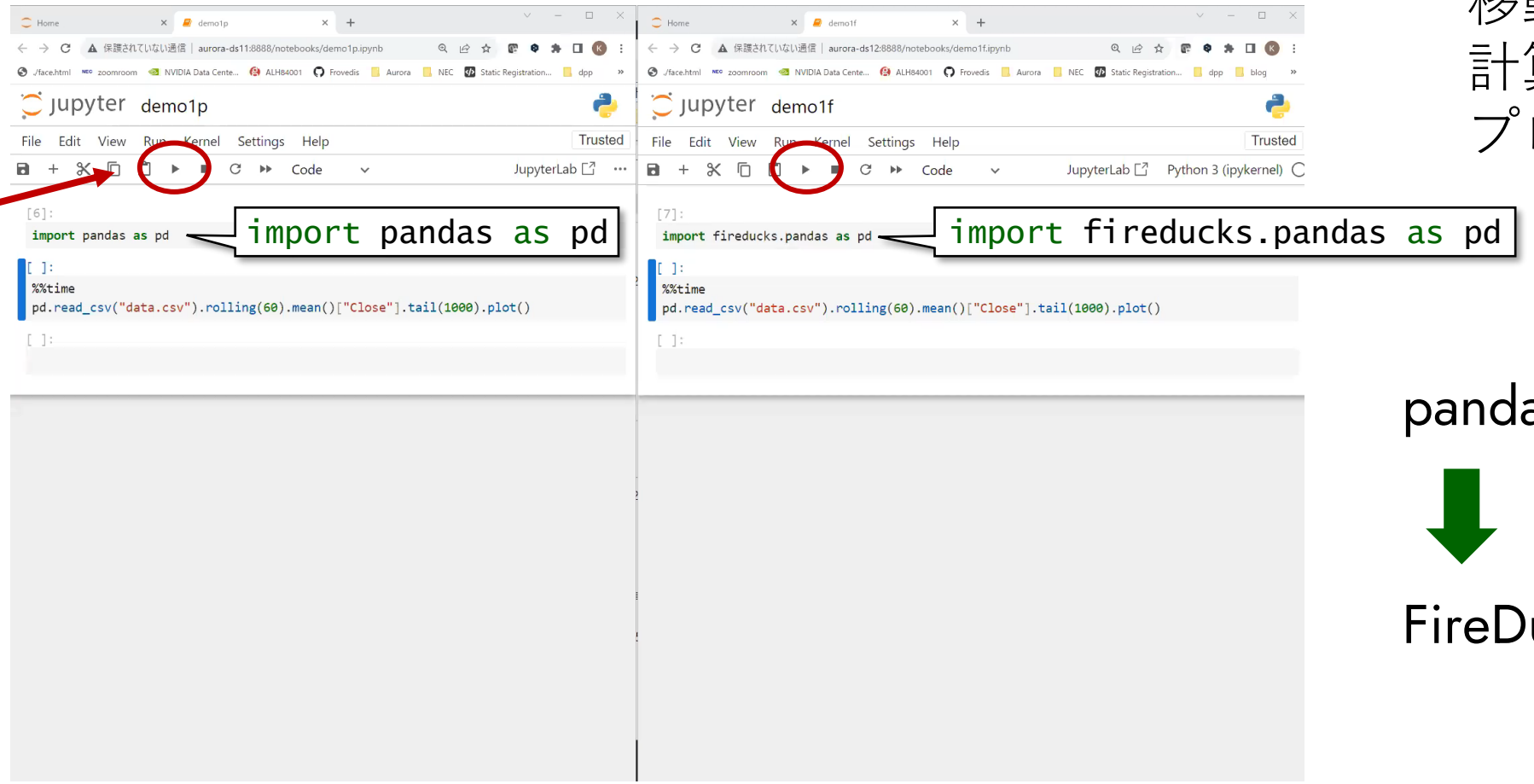
Demo

```
pd.read_csv(“data.csv”).rolling(60).mean()[“Close”].tail(1000).plot()
```

移動平均を計算するプログラム

pandas （違いはimport文だけ） **FireDucks**

実行開始ボタン



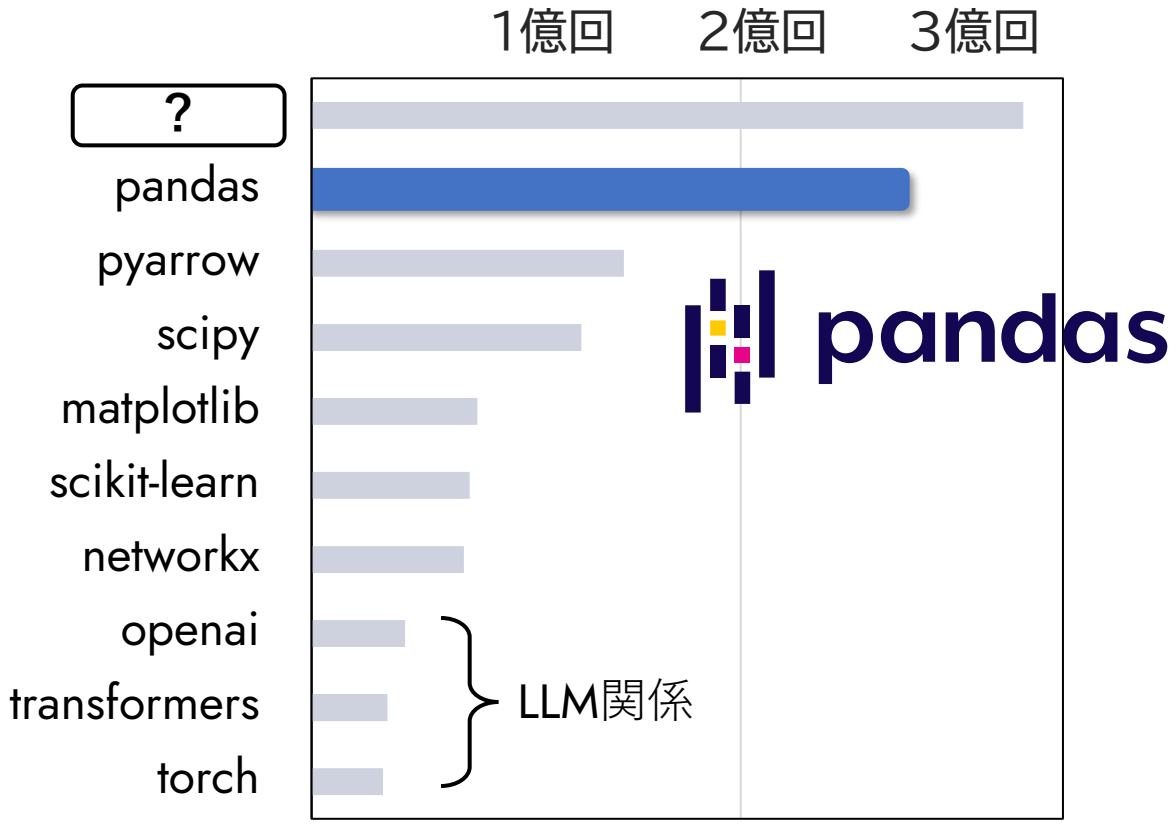
pandas: 4.06s

⬇ **約15倍**

FireDucks: 275ms

pandasとは？

月間2億回以上ダウンロードされるデータ分析の標準的なpythonライブラリ



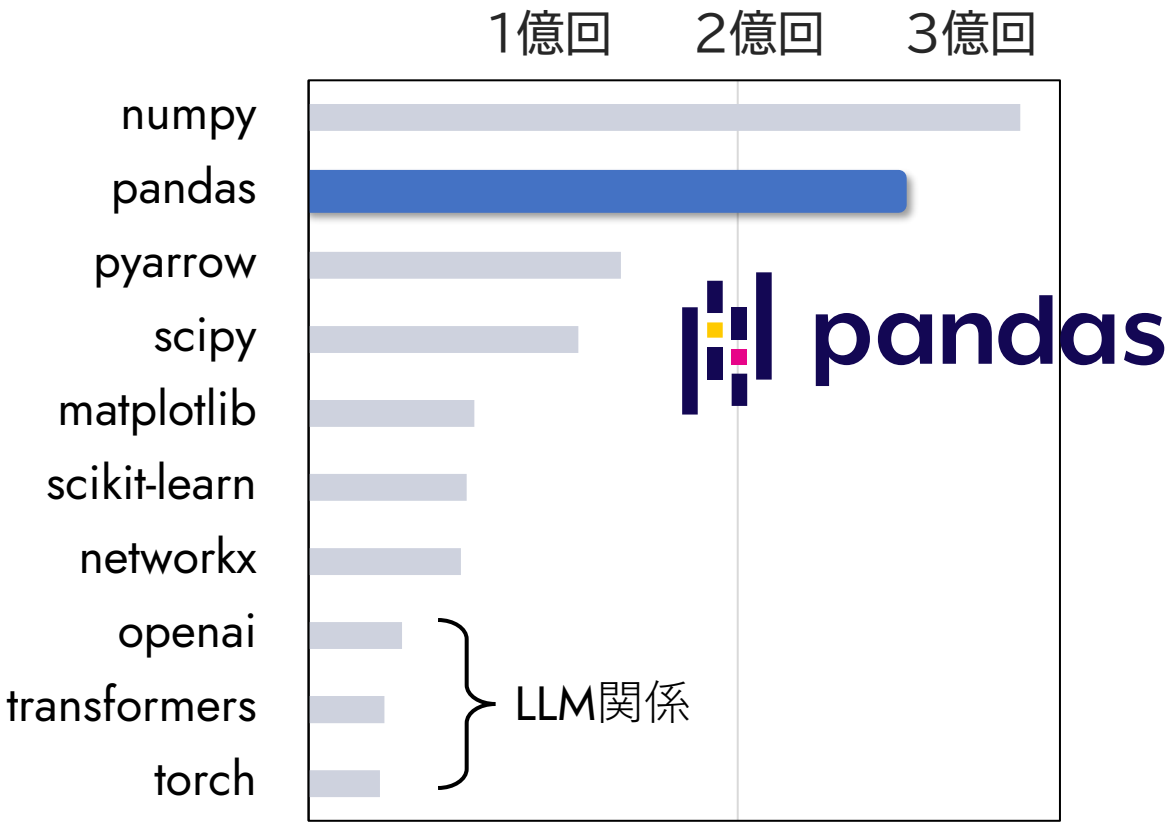
pypiの月間ダウンロード数
（データ分析関係のライブラリ 2024年10月）

https://www.amazon.co.jp/s?k=pandas+%E3%83%87%E3%83%BC%E3%82%BF&__mk_ja_JP=%E3%82%AB%E3%82%BF%E3%82%AB%E3%83%8A&crid=T44FOT0ODNPR&sprefix=pandas+%E3%83%87%E3%83%BC%E3%82%BF+%2Caps%2C165&ref=nb_sb_noss_2

https://www.udemy.com/ja/topic/pandas/

pandasとは？

月間2億回以上ダウンロードされるデータ分析の標準的なpythonライブラリ



pypiの月間ダウンロード数
（データ分析関係のライブラリ 2024年10月）

https://www.amazon.co.jp/s?k=pandas+%E3%83%87%E3%83%BC%E3%82%BF&__mk_ja_JP=%E3%82%AB%E3%82%BF%E3%82%AB%E3%83%8A&crid=T44FOT0ODNPR&sprefix=pandas+%E3%83%87%E3%83%BC%E3%82%BF+%2Caps%2C165&ref=nb_sb_noss_2

https://www.udemy.com/ja/topic/pandas/

データフレーム（表データ）

実際のデータ分析では表データは良く活用されている

（顧客データ，統計データ，センサーデータなど）

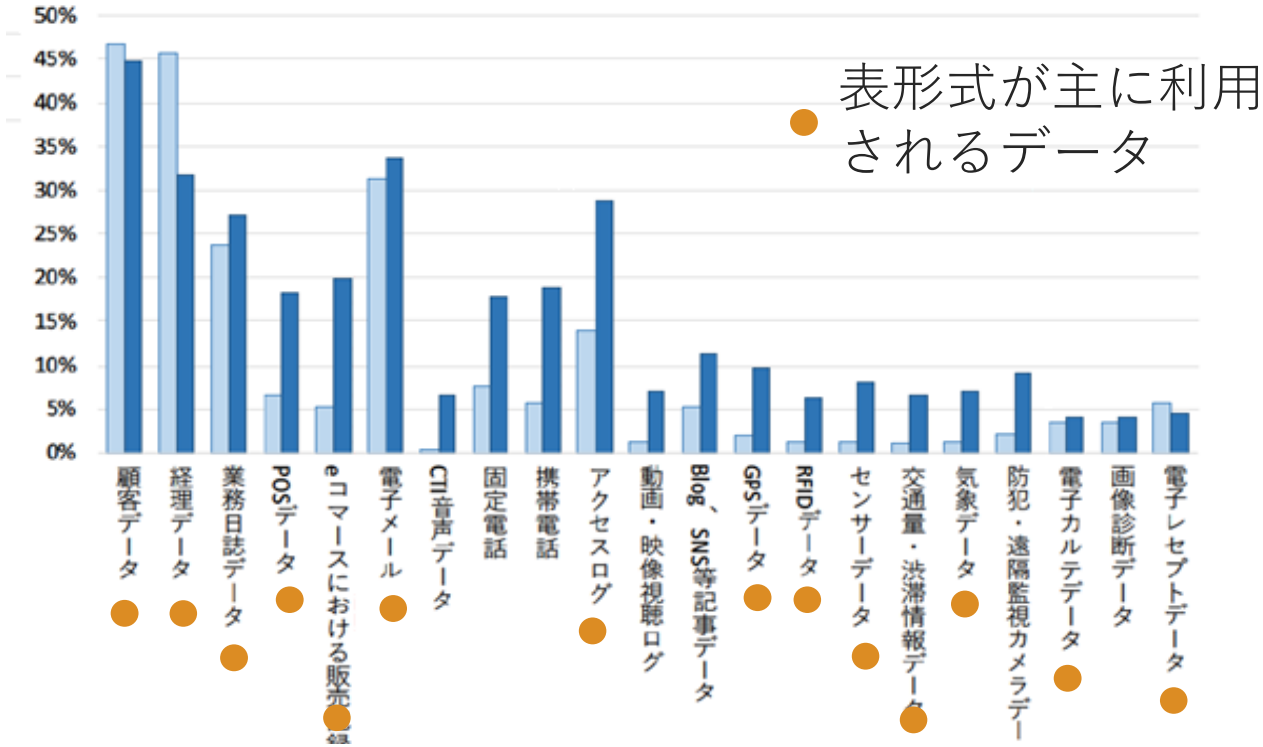
	Timestamp	Open	High	Low	Close	Volume_(BTC)	Volume_(Currency)	Weighted_Price
0	1325317920	4.39	4.39	4.39	4.39	0.455581	2.000000	4.390000
1	1325317980	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	1325318040	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	1325318100	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	1325318160	NaN	NaN	NaN	NaN	NaN	NaN	NaN
...	...	...	...	...	...	...	...	...
4857372	1617148560	58714.31	58714.31	58686.00	58686.00	1.384487	81259.372187	58692.753339
4857373	1617148620	58683.97	58693.43	58683.97	58685.81	7.294848	428158.146640	58693.226508
4857374	1617148680	58693.43	58723.84	58693.43	58723.84	1.705682	100117.070370	58696.198496
4857375	1617148740	58742.18	58770.38	58742.18	58760.59	0.720415	42332.958633	58761.866202
4857376	1617148800	58767.75	58778.18	58755.97	58778.18	2.712831	159417.751000	58764.349363

4857377 rows × 8 columns

bitcoinの1分ごとの価格情報

https://www.kaggle.com/datasets/mczielinski/bitcoin-historical-data

分析に活用しているデータ



総務省「デジタルデータの経済的価値の計測と活用の現状に関する調査研究」(2020) (一部編集)

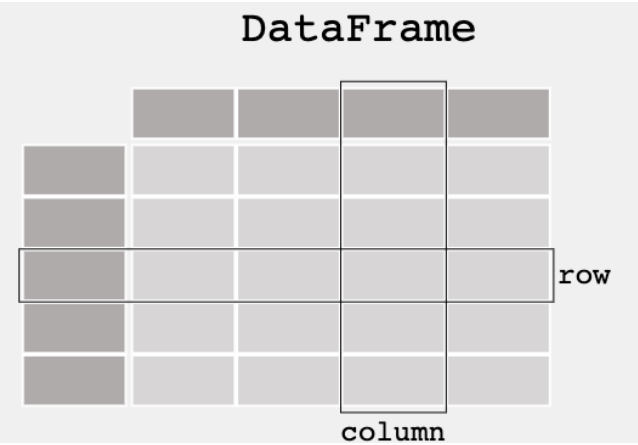
What is pandas? How important?

pandas introduction from NVIDIA

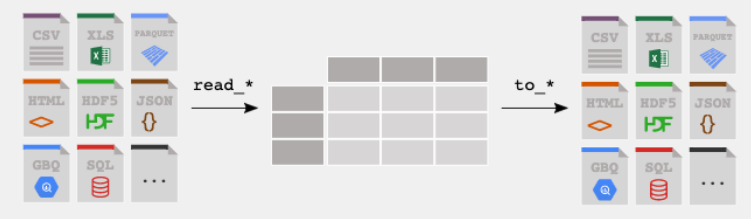
https://www.nvidia.com/en-us/glossary/pandas-python/

Pandas is the most popular software library for data manipulation and data analysis for the Python programming language.

Pandas addresses the many shortcomings that data scientists often encounter… In data science, working with data is usually sub-divided into multiple stages, including… **For these and other mission-critical data science tasks, Pandas excels.**



loading and storing



visualization

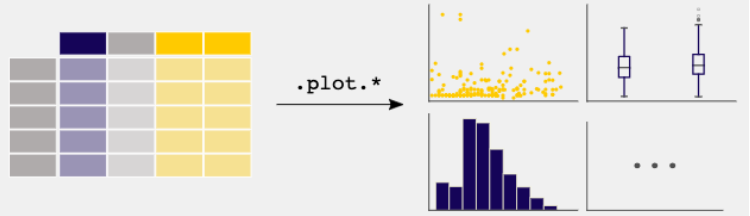
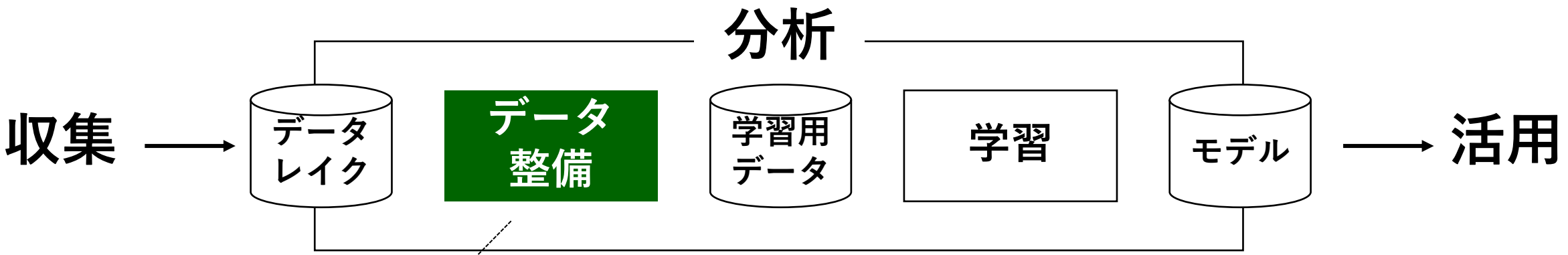


table merging

filtering

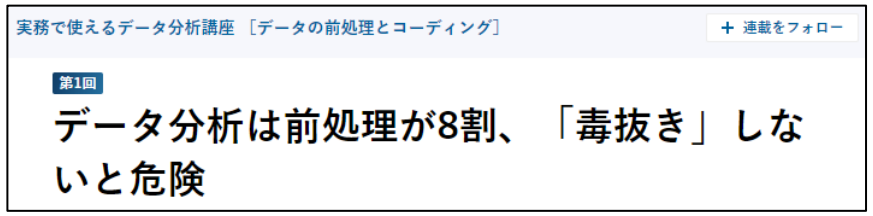
images: https://www.nvidia.com/en-us/glossary/pandas-python

pandas高速化のニーズ



pandasが良く使われるデータ整備がデータ分析のボトルネックに

- 探索的データ解析，学習用データの作成などの前処理
- 単純な整形だけでなく，複雑なアルゴリズムも登場

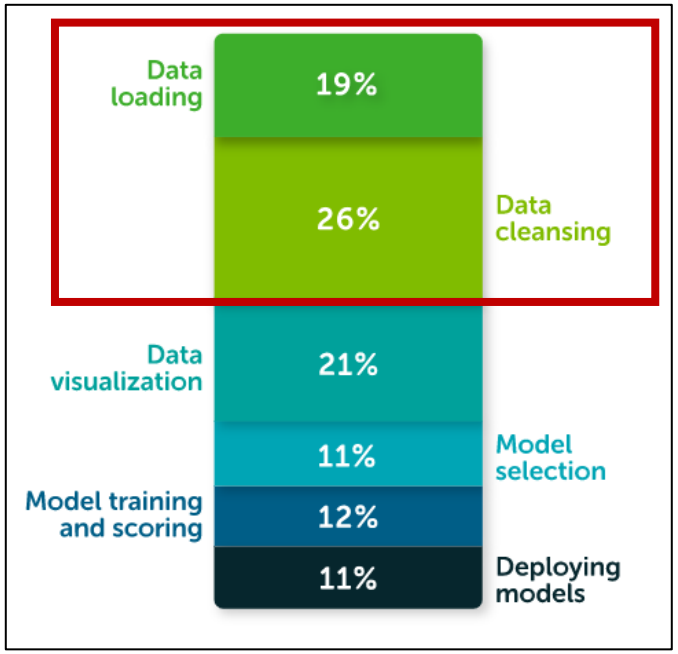

実務で使えるデータ分析講座［データの前処理とコーディング］ ＋ 連載をフォロー

第1回

データ分析は前処理が8割、「毒抜き」しないと危険

https://xtech.nikkei.com/atcl/learning/lecture/19/00110/00001/

データサイエンティストの時間の40%以上

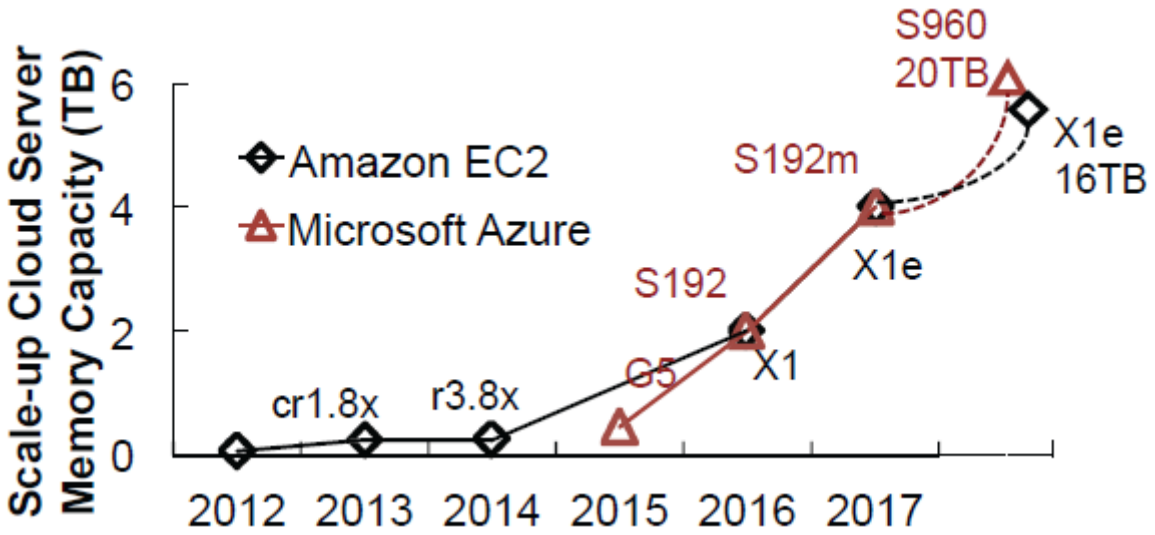


Anaconda The State of Data Science 2020

大量のデータを使えるようになったけど...

扱うデータ量や処理の複雑化に伴い速度課題が顕在化

クラウドサーバーのメモリ容量



コモディティサーバーでも
数百GBのメインメモリ

[5] Ogleari, M. et al.: String figure: A scalable and elastic memory network architecture, *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, IEEE, pp. 647–660 (2019)

pandasはなぜ遅い？

ほとんどの処理はシングルスレッド実行

Eager実行
（SQLのクエリプランナーが行うような最適化がされない）

遅い書き方ができてしまう

高速化の貢献

データ分析の効率向上

データ分析にかかった時間の7割が削減できて,
データ分析に集中できる！

クラウドコストの削減

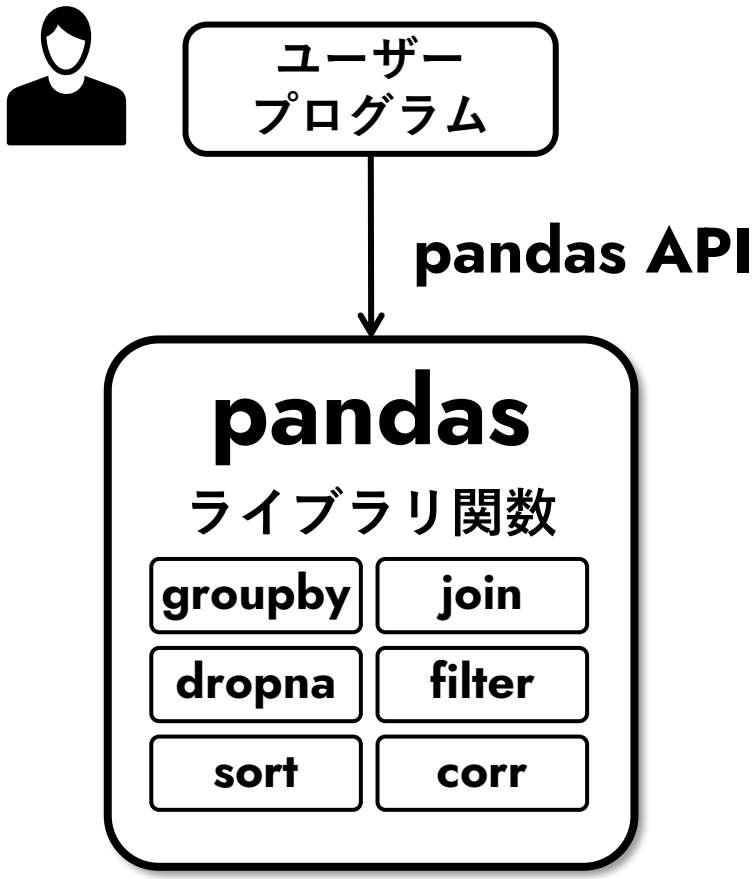
10倍速くなったから,
クラウドコストが**1/10**！

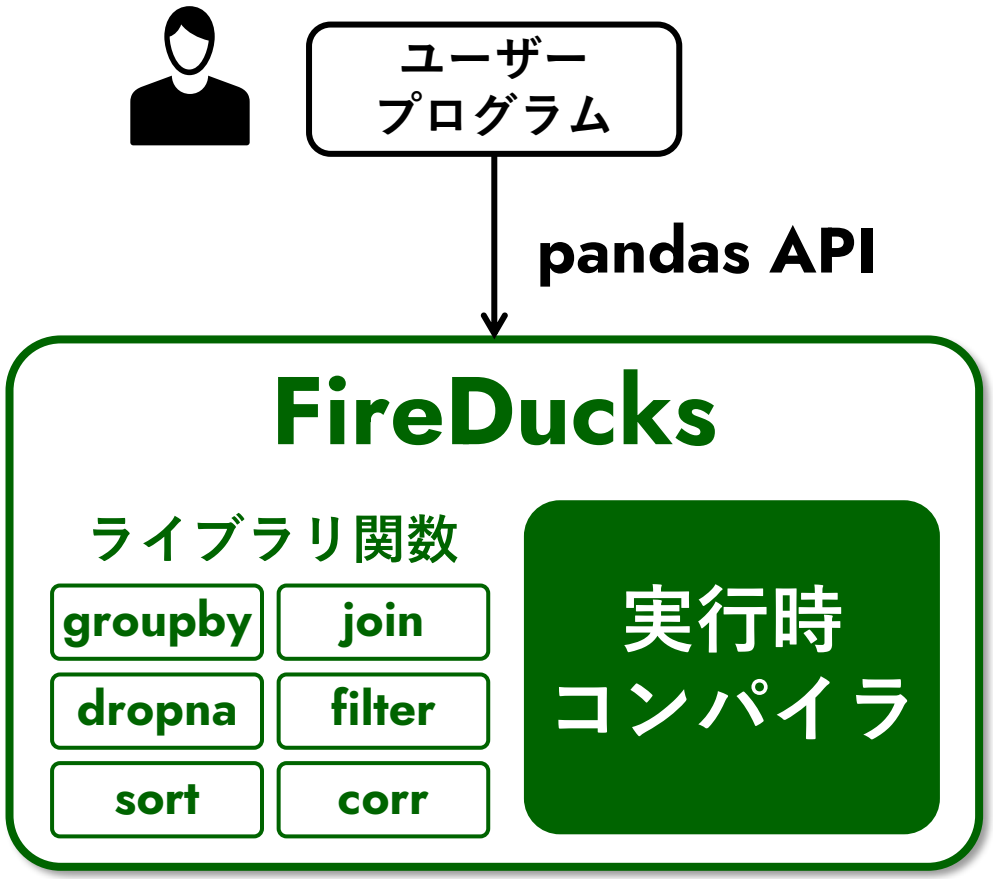
CO2排出量の削減

データ分析していると
CO2を垂れ流してるのではないかと気になっていた

実行時コンパイラで実現

ライブラリ中に埋め込まれた実行時コンパイラで，使い勝手やAPIを変えずに高速化

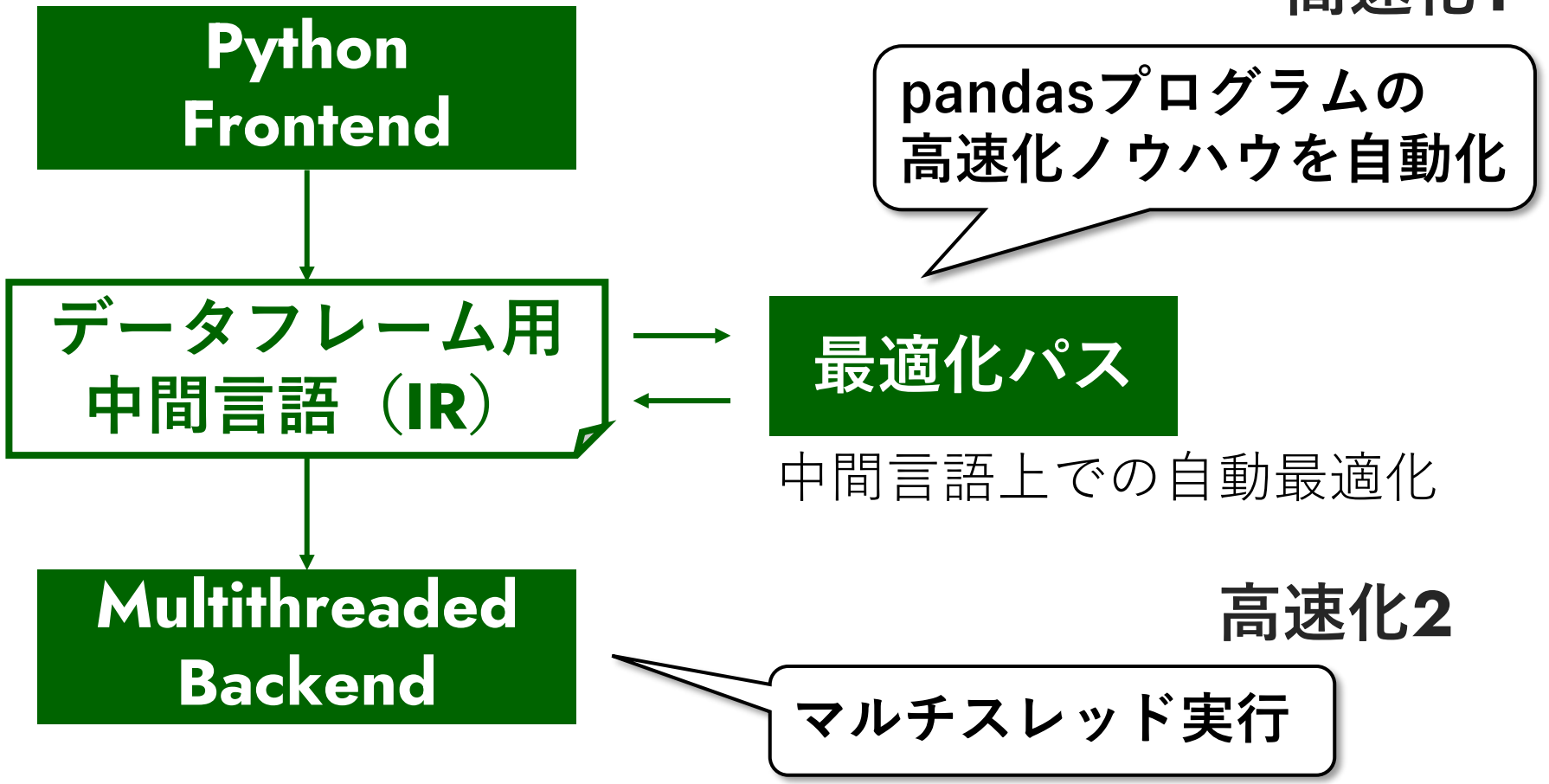




FireDucksのpandas互換性と高速化の仕組み

中間言語（IR）を介することで，APIの変更なく，**最適化や実行を改善**

ユーザープログラムを
一度中間言語に変換



ユーザーAPIとは
独立したバックエンド
で実行

データフレーム用中間言語

データフレームの要素処理を命令としたドメイン特化型の中間言語（IR）

pythonプログラム

```
pd.read_csv(“data.csv”)
  .rolling(60).mean()
  [“Close”]
  .tail(1000)
  .plot()
```

FireDucks IR

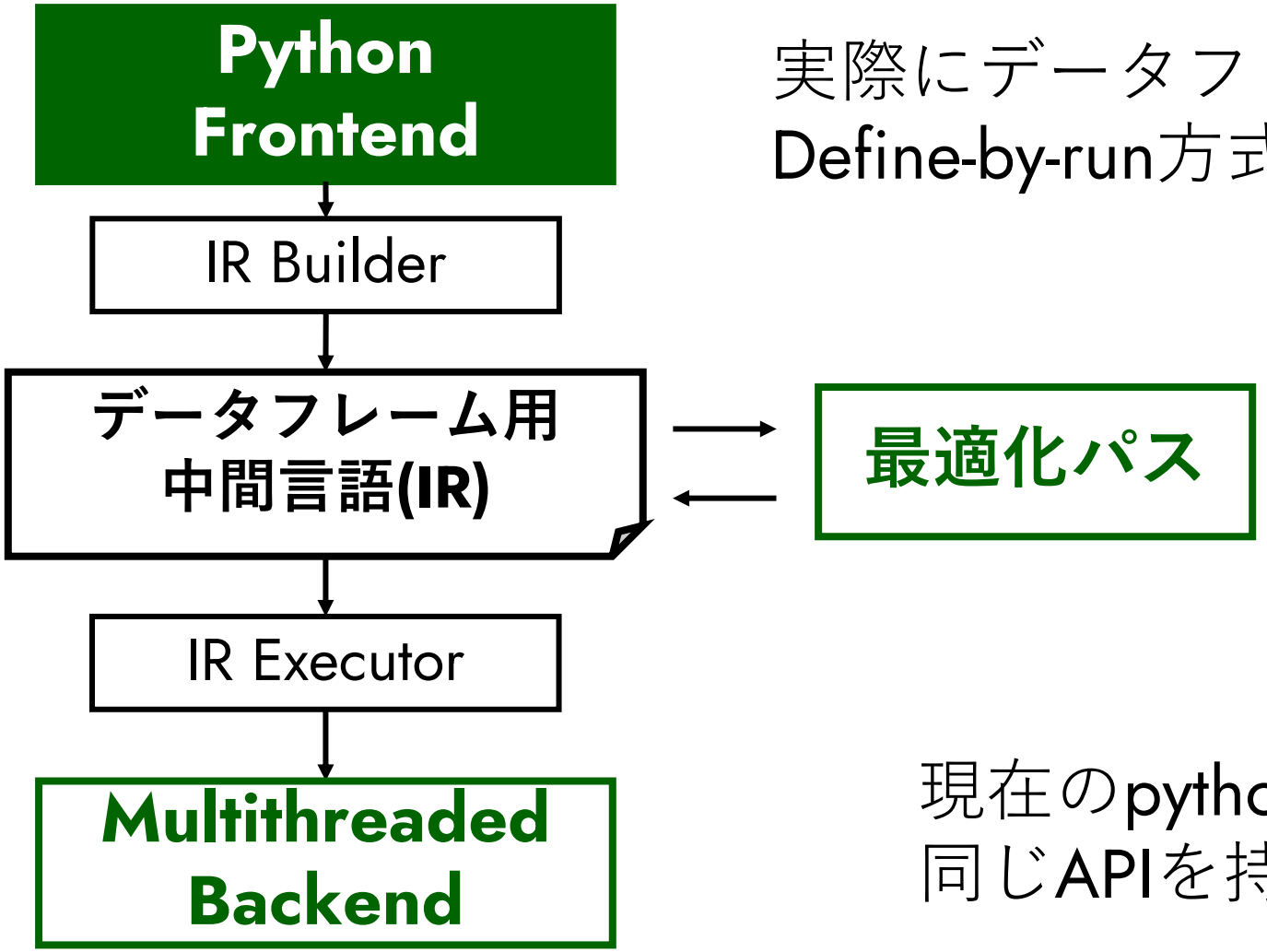
```
%t1 = read_csv('data.csv', %arg0)
%t2 = rolling_aggregate(%t1, 60, 60, 'mean')
%t3 = project(%t2, 'Close')
%t4 = slice(%t3, -1000, None, 1)
```

※: 最適化に適したSSA形式

1命令がデータフレーム操作の意味を持つので,
データフレーム特化の最適化を行いやすい

独自の中間言語を定義できるコンパイラフレームワークを利用
（LLVMのサブプロジェクト）

Python Frontend



FireDucksの構造

実際にデータフレーム処理をするのでなく，
Define-by-run方式で中間言語の生成を行う

現在のpython frontendはpandasと
同じAPIを持つ

Define-by-runによる中間言語生成

read_csvを実行

```
pd.read_csv(“data.csv”)
  .rolling(60).mean()
  [“Close”]
  .tail(1000)
  .plot()
```

FireDucks内部でread_csv opが生成される

```
%t1 = read_csv('data.csv', %arg0)
```

この時点では実際のcsvファイルの
読み込みは行われない

Define-by-runによる中間言語生成

read_csvを実行

```
pd.read_csv(“data.csv”)
  .rolling(60).mean()
  [“Close”]
  .tail(1000)
  .plot()
```

FireDucks内部でread_csv opが生成される

```
%t1 = read_csv('data.csv', %arg0)
```

この時点では実際のcsvファイルの
読み込みは行われない

FireDucksのread_csvの実装（簡略版）

```
def read_csv(filename):
    value = irbuilder.build_op(OP_read_csv, filename)
    return DataFrame(value)
```

Define-by-runによる中間言語生成

rolling.meanを実行

```
pd.read_csv(“data.csv”)
  .rolling(60).mean()
  [“Close”]
  .tail(1000)
  .plot()
```

rolling_aggregate opが生成される

```
%t1 = read_csv('data.csv', %arg0)
%t2 = rolling_aggregate(%t1, 60, 60, 'mean')
```

Define-by-runによる中間言語生成

__getitem__を実行
（列の取り出し）

```
pd.read_csv(“data.csv”)
  .rolling(60).mean()
  [“Close”]
  .tail(1000)
  .plot()
```

rolling_aggregate opが生成される

```
%t1 = read_csv('data.csv', %arg0)
%t2 = rolling_aggregate(%t1, 60, 60, 'mean')
%t3 = project(%t2, 'Close')
```

Define-by-runによる中間言語生成

tailを実行

```
pd.read_csv(“data.csv”)
  .rolling(60).mean()
  [“Close”]
  .tail(1000)
  .plot()
```

slice opが生成される

```
%t1 = read_csv('data.csv', %arg0)
%t2 = rolling_aggregate(%t1, 60, 60, 'mean')
%t3 = project(%t2, 'Close’)
%t4 = slice(%t3, -1000, None, 1)
```

中間言語の実行開始

特定のAPIが実行されると，中間言語の実行を開始（まとめて遅延実行）

plotを実行

```
pd.read_csv(“data.csv”)
  .rolling(60).mean()
  [“Close”]
  .tail(1000)
  .plot()
```

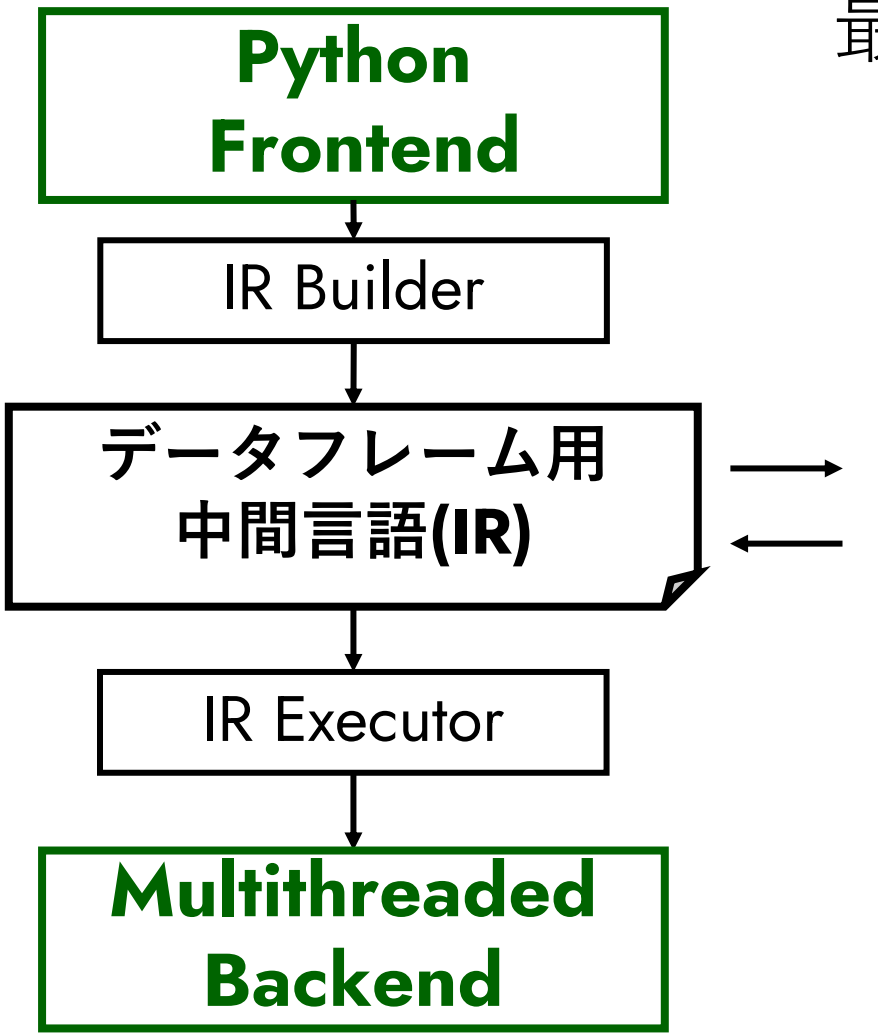
```
%t1 = read_csv('data.csv', %arg0)
%t2 = rolling_aggregate(%t1, 60, 60, 'mean')
%t3 = project(%t2, 'Close’)
%t4 = slice(%t3, -1000, None, 1)
```

plotはいくつかある評価ポイントの一つ（他には`__repr__`など）

```
print(df)
```

__repr__はprint内で利用される

最適化パスでの自動最適化



FireDucksの構造

最適化パスがIRをより良いIRに変換

- IR変換として，各種のデータフレーム高速化テクニックを実装

現在も鋭意拡充中

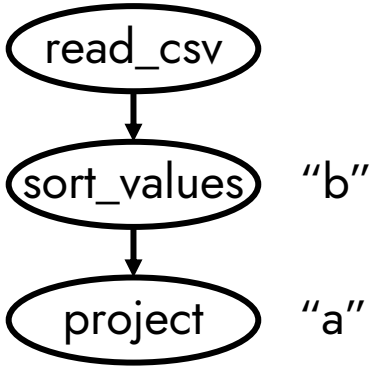
1) projection pushdowns最適化

projection（列の抽出）を前出しすることで，中間データを削減

ユーザーが書いたプログラム

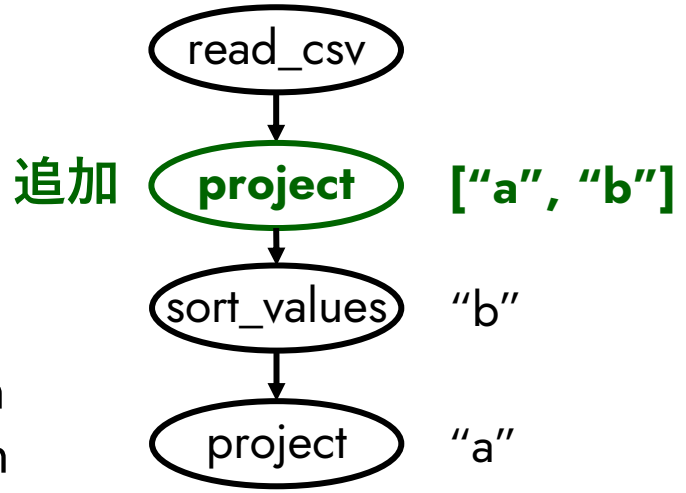
```
df = pd.read_csv("sample.csv")
sorted = df.sort_values("b")
result = sorted[["a"]]
```

中間言語生成



入力IR

projection pushdown



出力IR

実際に実行される処理

```
df = pd.read_csv("sample.csv")
df2 = df[["a", "b"]]
sorted = df2.sort_values("b")
result = sorted[["a"]]
```

実行

2) パターン最適化

特定の命令の組み合わせをより良い組み合わせに変換する

```
df[~df[“a”].isnull()]      # a列がnullの行を削除
```


```
df[“timestamp”].dt.strftime(“%Y”).astype(int)      # timestamp列から年の取り出し
```


```
groupby(“a”).sum().sort_values(“b”)      # groupby結果をb列でソート
```

2) パターン最適化

特定の命令の組み合わせをより良い組み合わせに変換する

```
df[~df["a"].isnull()]      # a列がnullの行を削除
```

➡ `df.dropna("a")`

```
df["timestamp"].dt.strftime("%Y").astype(int)      # timestamp列から年の取り出し
```

➡ `df["timestamp"].dt.year`

```
groupby("a").sum().sort_values("b")      # groupby結果をb列でソート
```

➡ `groupby("a", sort=False).sum().sort_values("b")`

sort=Falseを追加

パターン最適化の実装

df[~df[“a”].isnull()]　　# a列がnulの行を削除

➡ df.**dropna**(“a”)

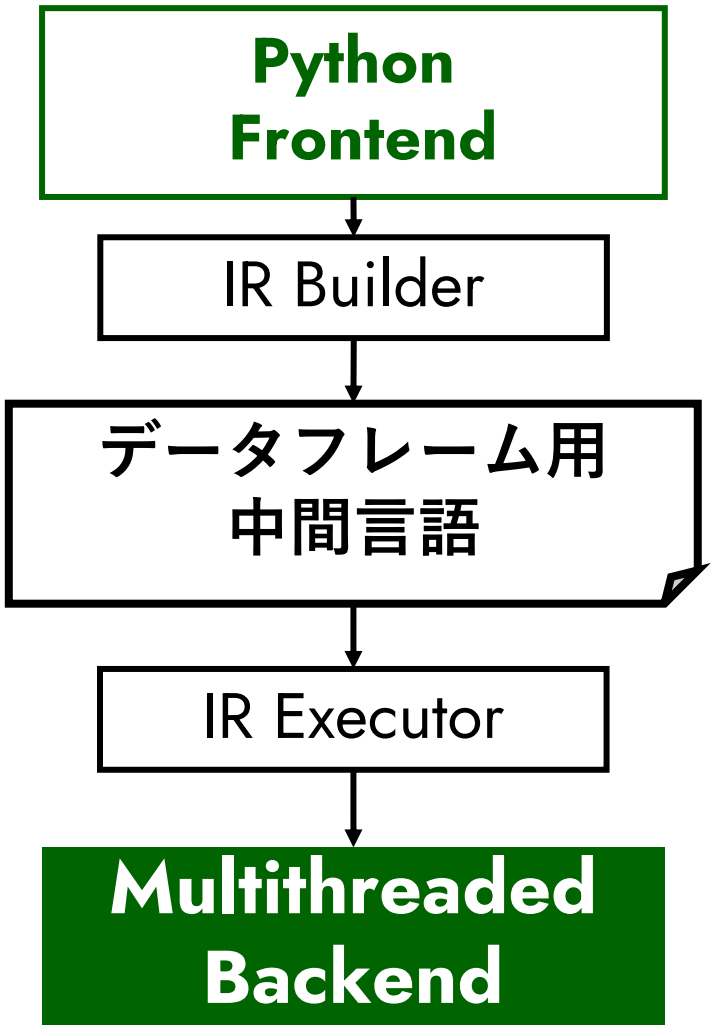
コンパイラフレームワーク

パターンの定義を書くだけ

```
def FilterToDropnaPat
    : Pat<(FilterOp $tbl,
           (InvertOp: $res_invert (IsNullOp(ProjectOp $proj_tbl, $cols, $chain_proj),
                     $chain_isnull),
            $chain_inv),
           $no_align,
           $chain),
          (DropnaOp $proj_tbl,
           (MakeTupleFromVectorOrScalarOfColumnNameOp $cols),
           (MakeScalarIntOp(ConstantI64Op ConstantAttr<I64Attr, "0">,
                            (returnType "$_builder.getI64Type()"))),
           ConstantAttr<I1Attr, "0">, ConstantAttr<I1Attr, "1">, ConstantAttr<I32Attr, "0">, $chain_proj),
          [(Constraint<CPred<"res_invert.getResult().hasOneUse()">>),(Constraint<CPred<"tbl == proj_tbl">>)]>;
```

このような仕組みを使い現在13個のパターンを実装

Backend



FireDucksの構造

中間言語中の各命令を実行するカーネルの集合

CPU用のマルチスレッドバックエンド（C++）

- データ構造にApache Arrowを利用
- Arrowが提供するカーネルに加えて，
  並列化・最適化を強化したカーネルを追加

APACHE ARROW

GPUバックエンドも開発中

Backendでの最適化例: groupby

```
df.groupby("x").sum()
```

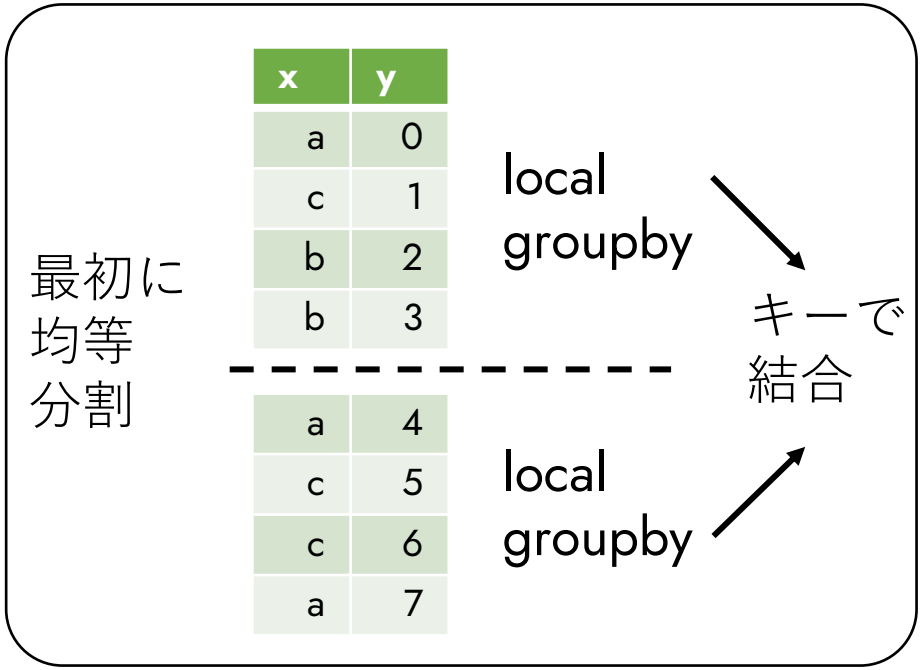
グループ数推定

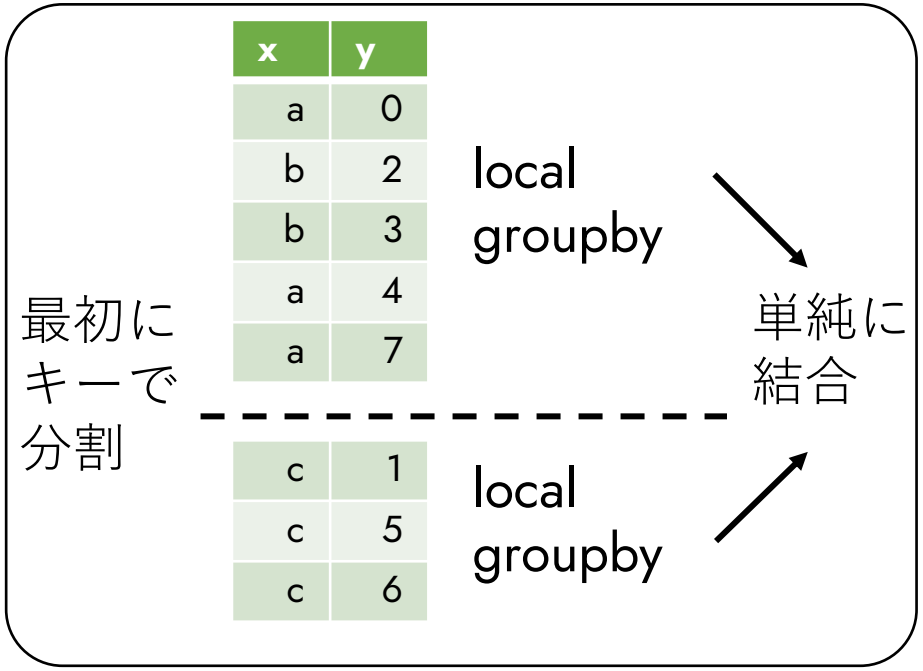
グループ数が少ないときに向いた
並列groupbyアルゴリズム

グループ数が多いときに向いた
並列groupbyアルゴリズム

元データ

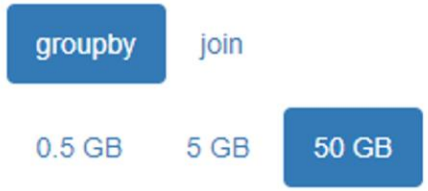
x	y
a	0
c	1
b	2
b	3
a	4
c	5
c	6
a	7





FireDucksの性能 (要素処理 groupby, join)

Database-like ops benchmark (https://duckdblabs.github.io/db-benchmark)



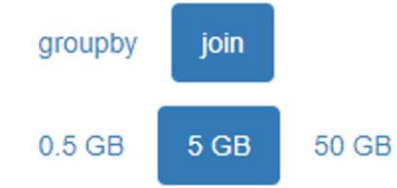
basic questions

Input table: 1,000,000,000 rows x 9 columns (50 GB)

rank-1

FireDucks	1.0.4	2024-09-10	15s
DuckDB	1.0.0	2024-07-04	25s
ClickHouse	24.5.1.1763	2024-06-07	28s
Polars	1.1.0	2024-07-09	47s
Datafusion	38.0.1	2024-06-07	56s
data.table	1.15.99	2024-06-07	88s
DataFrames.jl	1.6.1	2024-06-07	91s
InMemoryDatasets	0.7.13	2023-10-17	218s
spark	3.5.1	2024-06-07	261s
R-arrow	16.1.0	2024-06-07	378s
collapse	2.0.14	2024-06-07	411s
(py)datatable	1.2.0a0	2024-06-07	1022s
dplyr	1.1.4	2024-06-07	1104s
pandas	2.2.2	2024-06-07	1126s
dask	2024.5.2	2024-06-07	out of memory
Modin		see README	pending

Groupby



basic questions

Input table: 100,000,000 rows x 7 columns (5 GB)

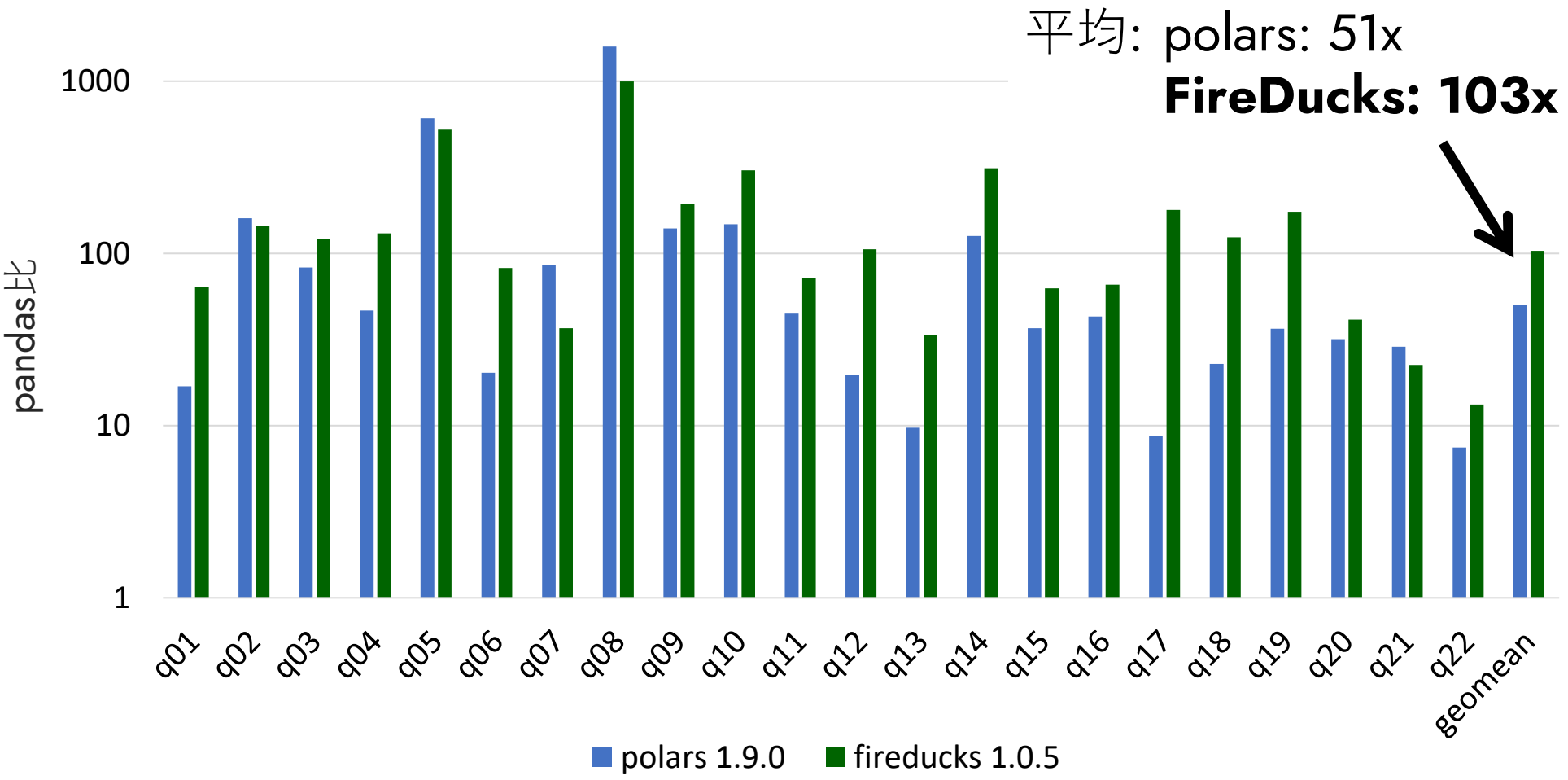
rank-1

FireDucks	1.0.4	2024-09-10	7s
DuckDB	1.0.0	2024-07-04	9s
Polars	1.1.0	2024-07-08	9s
Datafusion	38.0.1	2024-06-07	15s
InMemoryDatasets	0.7.13	2023-10-20	25s
ClickHouse	24.5.1.1763	2024-06-07	43s
data.table	1.15.99	2024-06-07	62s
collapse	2.0.14	2024-06-07	69s
DataFrames.jl	1.6.1	2024-06-07	77s
spark	3.5.1	2024-06-07	128s
dplyr	1.1.4	2024-06-07	214s
pandas	2.2.2	2024-06-07	244s
dask	2024.5.2	2024-06-07	635s
(py)datatable	1.2.0a0	2024-06-07	undefined exception
R-arrow	16.1.0	2024-06-07	out of memory
Modin		see README	pending

Join

FireDucksの性能 (TPC-Hベンチマーク Scale Factor=10)

pandasからコード変更なく最大996倍，平均103倍



評価環境

Intel(R) Xeon(R) Platinum 8488C
(32コア)

メモリ: 128GB
OS: Linux

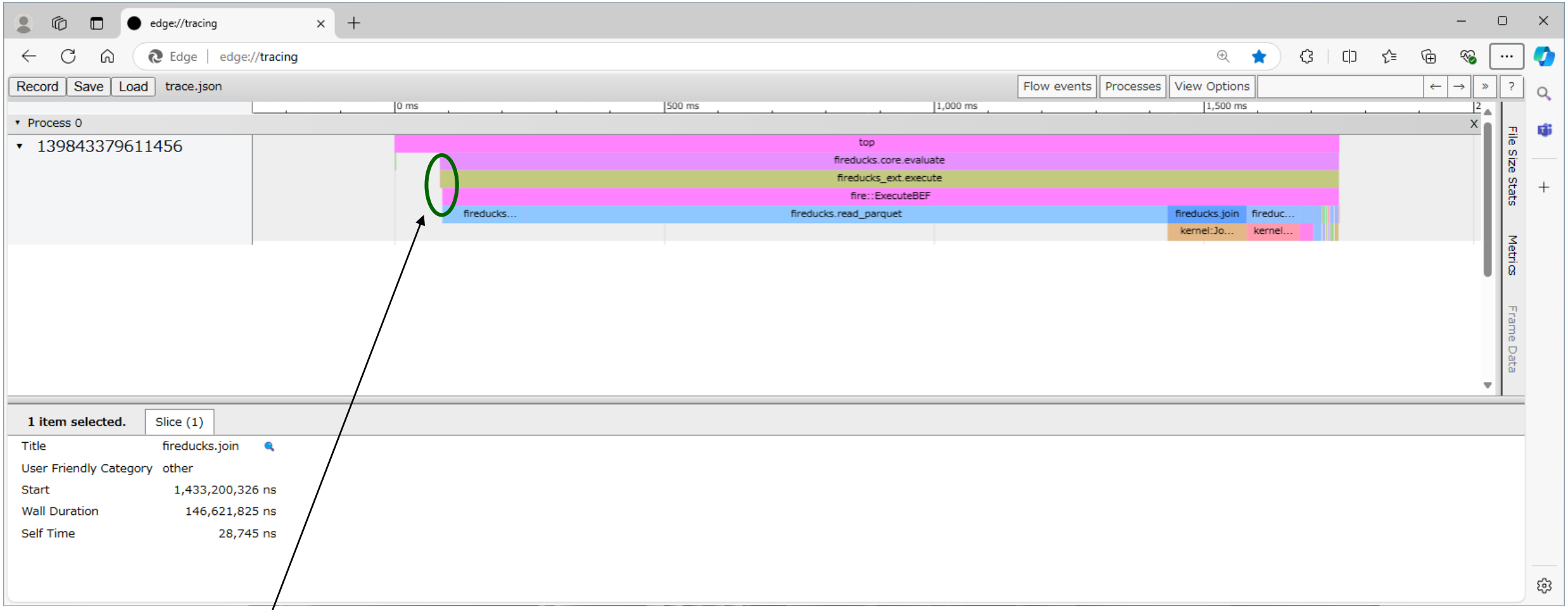
pandas 2.2.2
polars 1.9.0
FireDucks 1.0.5

ベンチマークコード

https://github.com/fireducks-dev/polars-tpch/tree/fireducks

IOを含まない時間で比較

コンパイル時間は十分短い



数~数十ms

DataFrameライブラリの比較

	pandas 互換性	シングルノード 性能	マルチノード 性能
FireDucks	○	○	×
Polars	×	○	×
Modin	○	△	○
Dask/Vaex	△	△	○
Pandas	○	×	×

FireDucksの利用

pipコマンドでインストール可能（BSDライセンス）

```
$ pip install fireducks
```

※ 現在はLinuxのみサポート（WSL可）

import文の書き換えだけでなく，pandasからの自動変換も可能

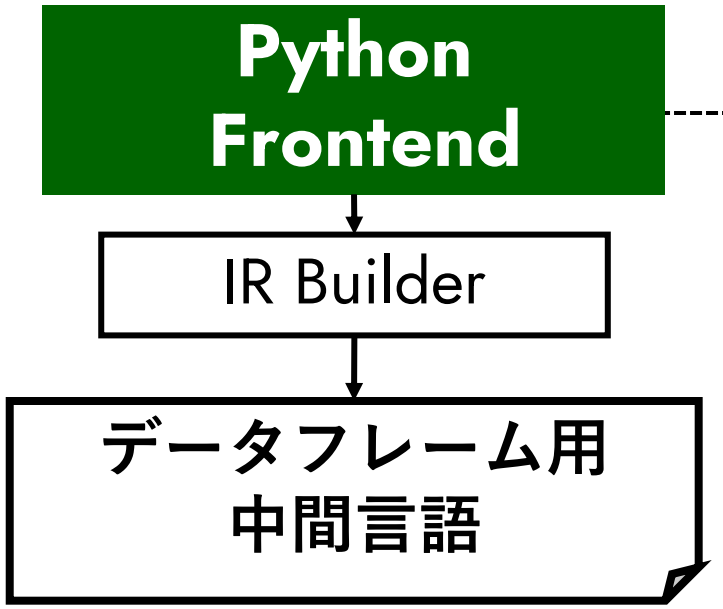
pythonコマンドの引数で指定

```
$ python3 -m fireducks.pandas program.py
```

jupyter notebookではマジックコマンド

```
%load_ext fireducks.pandas
import pandas
```

互換性向上の仕組み: Fallback



pandas

Fallback

中間言語を作らずに，Frontendからpandasを直接呼び出す

互換性は上がるけど，性能は上がらない（FireDucksで速くならないときはほぼこれ）

```
$ FIREDUCKS_FLAGS="-Wfallback" python -mfireducks.pandas demo.py
demo.py:4: FallbackWarning: Series.plot 0.201566 sec ...
```

※ Warningを出すことは可能（ご報告して下さい）

Tips

applyやループを利用しない

（A列が2より大きい行のB列の合計）

ループ

```
s = 0
for i in range(len(df)):
    if df["A"][i] > 2:
        s += df["B"][i]
```

apply

```
s = 0
def func(row):
    if row[“a”] > 2:
        s += row[“B”]

df.apply(func)
```

```
s = df[df["A"] > 2]["B"].sum()
```

時間計測の注意

```
t0 = time.time()
df.sort_values(“a”)
t1 = time.time()
print(t1 – t0)
```

正しい時間計測ができない（遅延実行）

```
df._evaluate()
t0 = time.time()
df.sort_values(“a”)._evaluate()
t1 = time.time()
print(t1 – t0)
```

明示的な実行の指示

Resource

Webサイト

https://fireducks-dev.github.io/ja/
(ユーザーガイド，ベンチマークなど)

github (issue report)

https://github.com/fireducks-dev/fireducks

slack (Q&A，雑談)

twitter/X (リリース情報)

https://x.com/fireducksdev

おわりに

FireDucksは，pandasのdrop-in replacementで使える高速データフレームライブラリです

実行時コンパイラ技術の活用により，pandasの弱点であるマルチスレッド実行，自動最適化を行います

ぜひご活用下さい

ご案内

エンタープライズサポート

FireDucks開発チームが所属するNECからエンタープライズサポートを提供予定

FireDucksに関するアンケート

FireDucksの今後の技術開発や事業展開のためのインプットをお願いします！

ハンズオン

Google Colabもしくはお手元のLinuxでお試しください

https://colab.research.google.com/github/fireducks-dev/fireducks/blob/main/notebooks/fireducks_pandas_nyc_demo.ipynb

- NYCタクシーデータを用いたpandasとFireDucksの比較
- 最適化の例