Incremental Adoption of Spark Dask and Ray

Kevin Kho

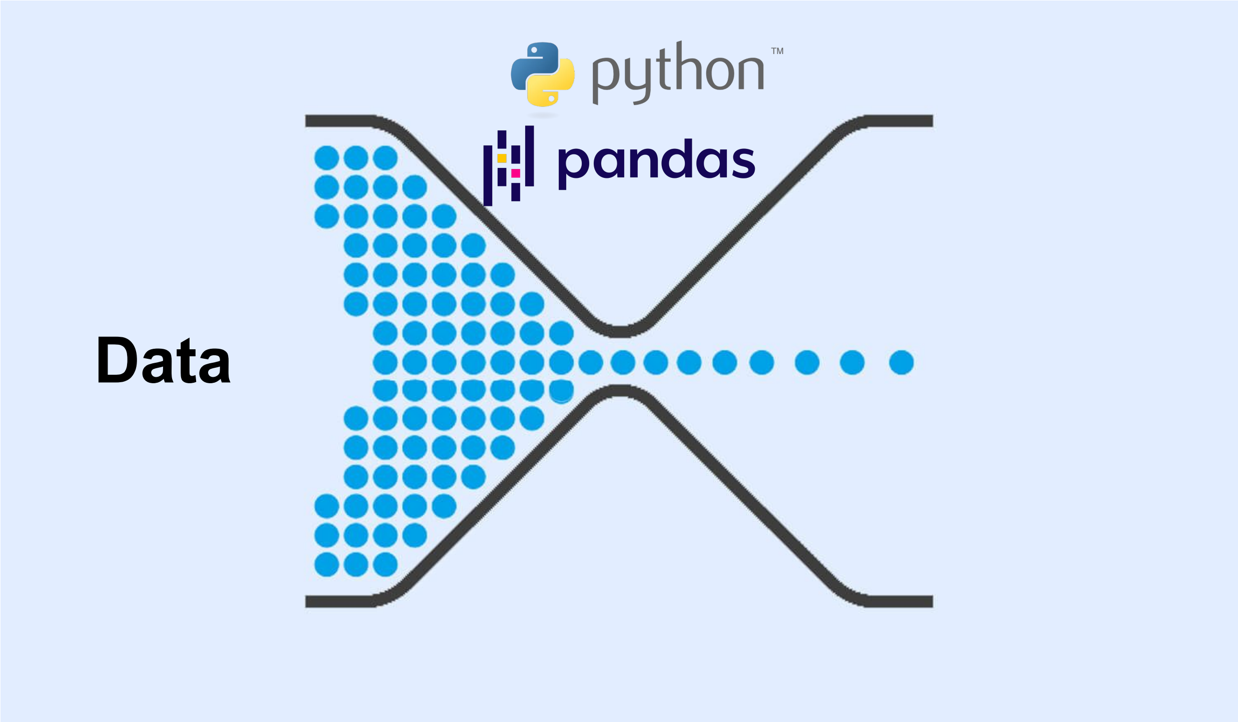
Jun 8, 2023 @ Budapest Data Forum

Agenda

- Why incremental adoption?
- What prevent incremental adoption?
- How Fugue helps? (Python API and Fugue SQL)

Why Incremental Adoption?

Needs for Distributed Computing



Needs for Distributed Computing

How to migrate existing code to these frameworks that have different syntax?

Incrementally moving portions of workloads that can truly benefit from additional resources

- Training several machine learning models in parallel
- Expensive feature engineering
- Data preprocessing and sampling
- …

Benefits of Incremental Adoption

- Can reduce all-in risk, be more cost effective
- Can minimize the adoption effort to achieve business objectives
- Can be more flexible on technical decisions
- …

What Prevents Incremental Adoption?

An example case

- For each uid, replace each numeric column with its z score over the timestamp

ts	_0	_1	_2	_3	_4	_5	_6	_7	_8	_9	uid
2023-01-28	0.188811	0.021881	0.550565	0.773350	0.071387	0.865288	0.342135	0.574775	0.357540	0.371238	77
2023-01-19	0.782489	0.469113	0.769810	0.077082	0.760960	0.764667	0.592024	0.805782	0.912301	0.096970	196
2023-01-23	0.285164	0.751222	0.674446	0.499522	0.447089	0.534925	0.501569	0.821843	0.794972	0.305882	552
2023-01-14	0.298401	0.524305	0.020982	0.987472	0.093021	0.853935	0.154098	0.057437	0.477783	0.951546	758
2023-01-25	0.139258	0.322477	0.567103	0.837987	0.234033	0.639060	0.096734	0.262131	0.995676	0.286171	682
...	...	...	...	...	...	...	...	...	...	...	...

Scaling out the Pandas solution

```python
def zscore_pd_gp(df:pd.DataFrame, n) -> pd.DataFrame:
    idf = df.sort_values(["uid","ts"]).set_index("uid")
    subdf = idf[COLS]
    x = subdf.groupby("uid", sort=False).shift(1).rolling(n)
    z=(subdf-x.mean()).abs()/x.std()
    return z.assign(ts=idf.ts).dropna().reset_index()[df.columns]
```

- **False Belief 1: Zero Rewrite Works**
- **False Belief 2: Full Rewrite == Best Performance**

False Belief 1: Zero Rewrite Works

- ❌The drop-in replacement solutions will let us fully adopt distributed systems without rewrite
- ❌The performance will be the *local performance * cluster size*

Pandas on Spark

```
import pyspark.pandas as ps

zscore_pd_gp(ps.DataFrame(pd_df), 2)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Input In [15], in <cell line: 3>()
      1 import pyspark.pandas as ps
----> 3 zscore_pd_gp(ps.DataFrame(pd_df), 2)

Input In [14], in zscore_pd_gp(df, n)
     19 idf = df.sort_values(["uid","ts"]).set_index("uid")
     20 subdf = idf[COLS]
---> 21 x = subdf.groupby("uid", sort=False).shift(1).rolling(n)
     22 z=(subdf-x.mean()).abs()/x.std()
     23 return z.assign(ts=idf.ts).dropna().reset_index()[df.columns]

TypeError: groupby() got an unexpected keyword argument 'sort'
```

Pandas on Spark (remove sort)

```
import pyspark.pandas as ps

zscore_pd_gp(ps.DataFrame(pd_df), 2)
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
Input In [13], in <cell line: 3>()
      1 import pyspark.pandas as ps
----> 3 zscore_pd_gp(ps.DataFrame(pd_df), 2)

Input In [11], in zscore_pd_gp(df, n)
     19 idf = df.sort_values(["uid","ts"]).set_index("uid")
     20 subdf = idf[COLS]
---> 21 x = subdf.groupby("uid").shift(1).rolling(n)
     22 z=(subdf-x.mean()).abs()/x.std()
     23 return z.assign(ts=idf.ts).dropna().reset_index()[df.columns]

File /usr/local/lib/python3.8/site-packages/pyspark/pandas/frame.py:13255, in
```

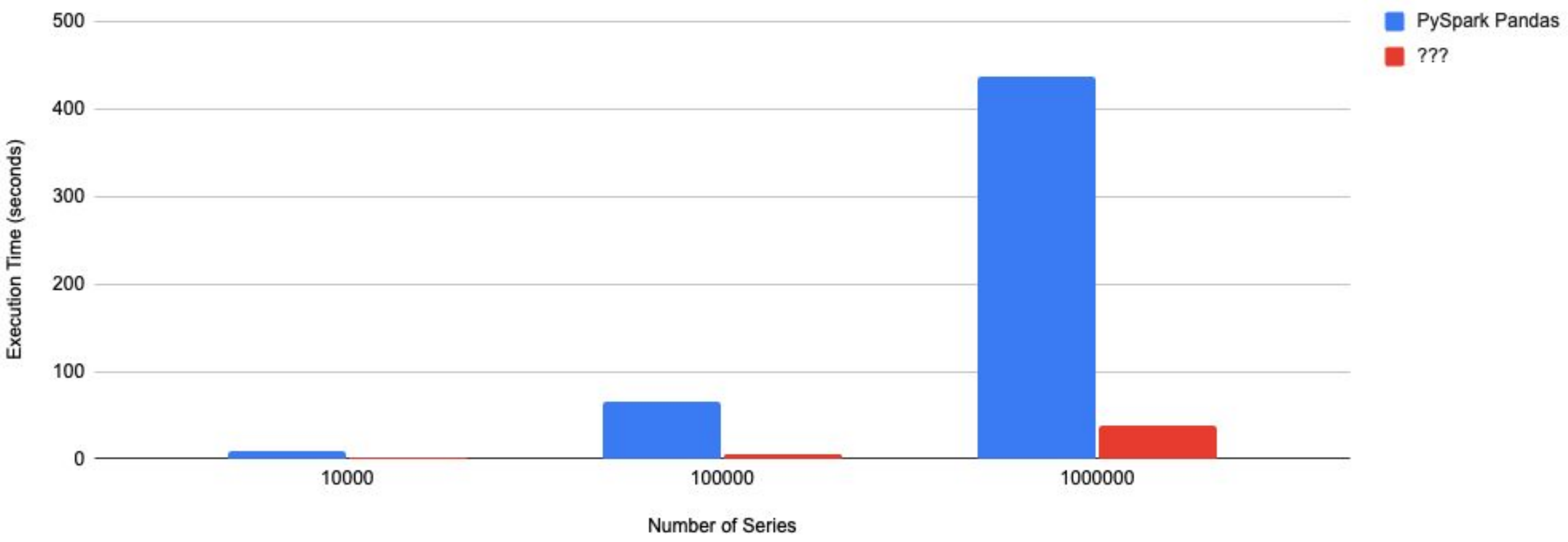
Drop In replacement cannot be 100% consistent with the original solution.

Rewrite to make it work

```
def zscore_pd(df:pd.DataFrame, n) -> pd.DataFrame:
    df = df.sort_values("ts").reset_index(drop=True)
    subdf = df[COLS]
    x = subdf.shift(1).rolling(n)
    z=(subdf-x.mean()).abs()/x.std()
    return z.assign(uid=df.uid, ts=df.ts).dropna()[df.columns]
```

```
ps.DataFrame(pd_df).groupby("uid").apply(lambda df:zscore_pd(df,n=2)).reset_index(drop=True)
```

Suboptimal Performance



A better implementation on spark can be 10x faster.

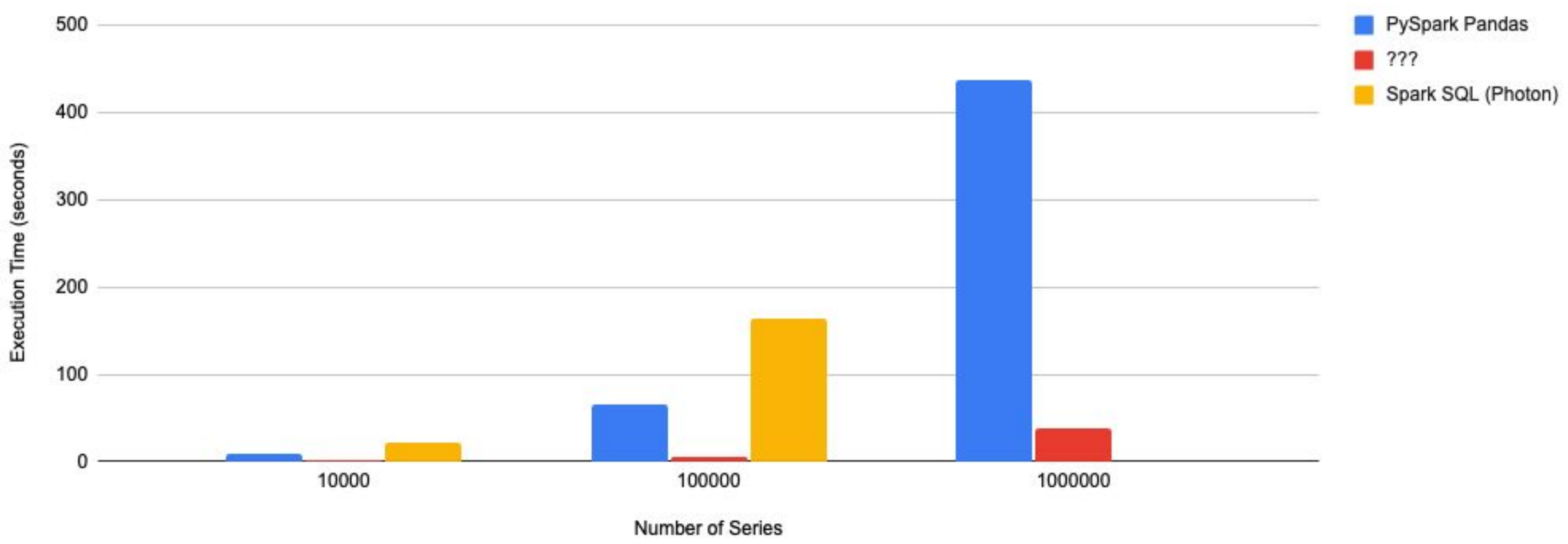
False Belief 2: Full Rewrite == Best Performance

- ❌Whatever a distributed framework provides, we should try to leverage, because they will yield the best performance

Rewrite to bypass Python and Pandas

```
%sql
WITH
    mean_std AS (
        SELECT
            uid, ts,_0,_1,_2,_3,_4,_5,_6,_7,_8,_9,
            AVG(_0) OVER (PARTITION BY uid ORDER BY ts ROWS BETWEEN {PERIOD} PRECEDING AND 1 PRECEDING) AS mean_0,
            STDDEV(_0) OVER (PARTITION BY uid ORDER BY ts ROWS BETWEEN {PERIOD} PRECEDING AND 1 PRECEDING) AS std_0,
            AVG(_1) OVER (PARTITION BY uid ORDER BY ts ROWS BETWEEN {PERIOD} PRECEDING AND 1 PRECEDING) AS mean_1,
            STDDEV(_1) OVER (PARTITION BY uid ORDER BY ts ROWS BETWEEN {PERIOD} PRECEDING AND 1 PRECEDING) AS std_1,
            AVG(_2) OVER (PARTITION BY uid ORDER BY ts ROWS BETWEEN {PERIOD} PRECEDING AND 1 PRECEDING) AS mean_2,
            ROW_NUMBER() OVER (PARTITION BY uid ORDER BY ts) AS rn
        FROM parquet.`{path}`
    ),
    z AS (
        SELECT
            uid, ts,
            abs((_0 - mean_0)/std_0) AS z_0,
            abs((_1 - mean_1)/std_1) AS z_1,
            abs((_2 - mean_2)/std_2) AS z_2
        FROM mean_std
        WHERE rn>{PERIOD} AND mean_0 IS NOT NULL AND std_0 IS NOT NULL
    )
SELECT
    SUM(z_0) AS z_0,
    SUM(z_1) AS z_1,
    SUM(z_2) AS z_2f
FROM z
```

Suboptimal Performance



The most native solution is not necessarily the best.

What we learned from the above examples

Drop-in replacement != Zero effort migration

Semantic consistency != Optimal performance

Full rewrite != Best performance

How Fugue Helps?

— Python API and Fugue SQL

What does Fugue do?

What does Fugue do?

Thin Wrapper

- python
- pandas
- polars
- Apache Arrow

Local Python Environment

- Apache Spark
- dask
- RAY

Scaling out the Pandas solution

```python
def zscore_pd(df:pd.DataFrame, n) -> pd.DataFrame:
    df = df.sort_values("ts").reset_index(drop=True)
    subdf = df[COLS]
    x = subdf.shift(1).rolling(n)
    z=(subdf-x.mean()).abs()/x.std()
    return z.assign(uid=df.uid, ts=df.ts).dropna()[df.columns]
```

```python
ps.DataFrame(pd_df).groupby("uid").apply(lambda df:zscore_pd(df,n=2)).reset_index(drop=True)
```

```python
from fugue import transform

transform(
    any_dataframe,
    zscore_pd,
    partition="uid",
    params=dict(n=PERIOD),
    schema="*",
)
```

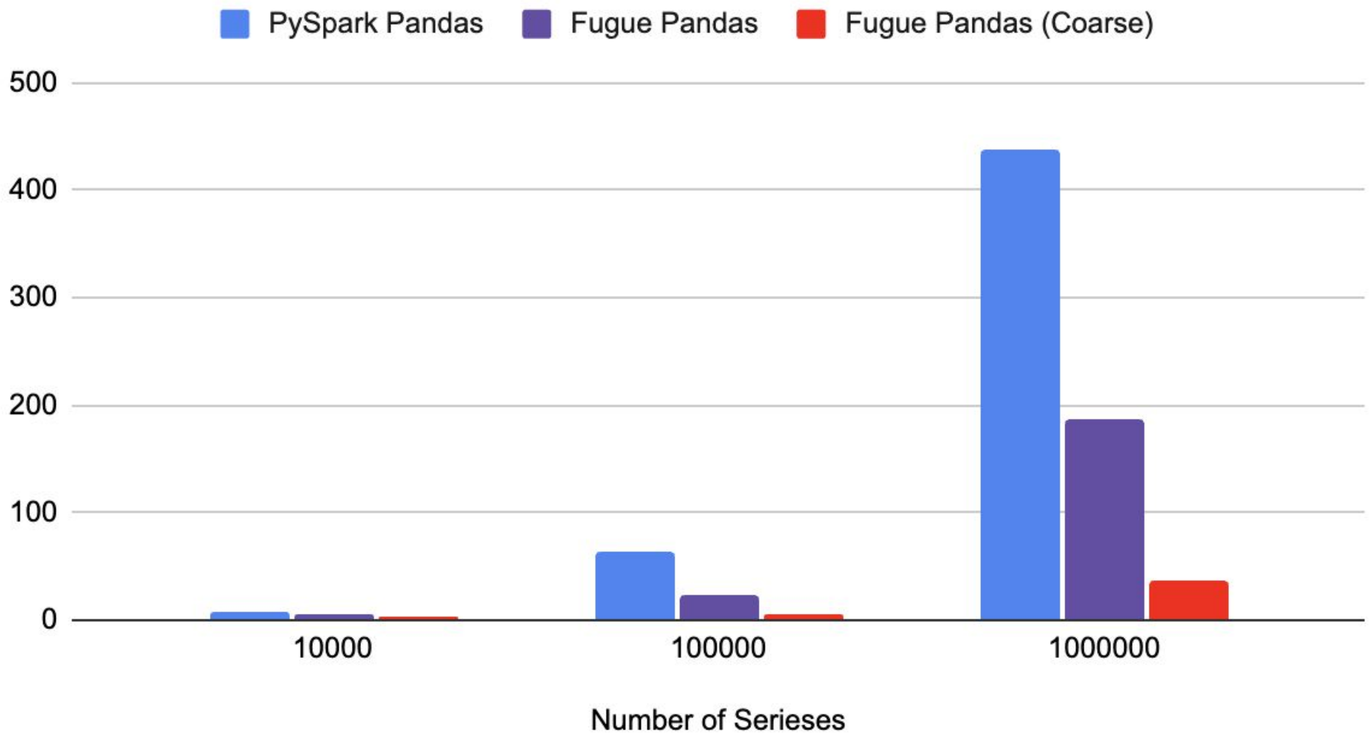
Scaling out the Pandas solution (Coarse)

```python
def zscore_pd_gp(df:pd.DataFrame, n) -> pd.DataFrame:
    idf = df.sort_values(["uid","ts"]).set_index("uid")
    subdf = idf[COLS]
    x = subdf.groupby("uid", sort=False).shift(1).rolling(n)
    z=(subdf-x.mean()).abs()/x.std()
    return z.assign(ts=idf.ts).dropna().reset_index()[df.columns]
```

```python
from fugue import transform

transform(
    any_dataframe,
    zscore_pd_gp,
    partition=dict(by="uid", algo="coarse"),
    params=dict(n=PERIOD),
    schema="*",
)
```

Performance Comparison



How to move one step to Spark/Ray?

```
transform(
    pd_sample_df,
    zscore_pd,
    partition="uid",
    params=dict(n=PERIOD),
    schema="*",
)
```

```
transform(
    spark_sample_df,
    zscore_pd,
    partition="uid",
    params=dict(n=PERIOD),
    schema="*",
)
```

```
transform(
    spark_df,
    zscore_pd,
    partition="uid",
    params=dict(n=PERIOD),
    schema="*",
)
```

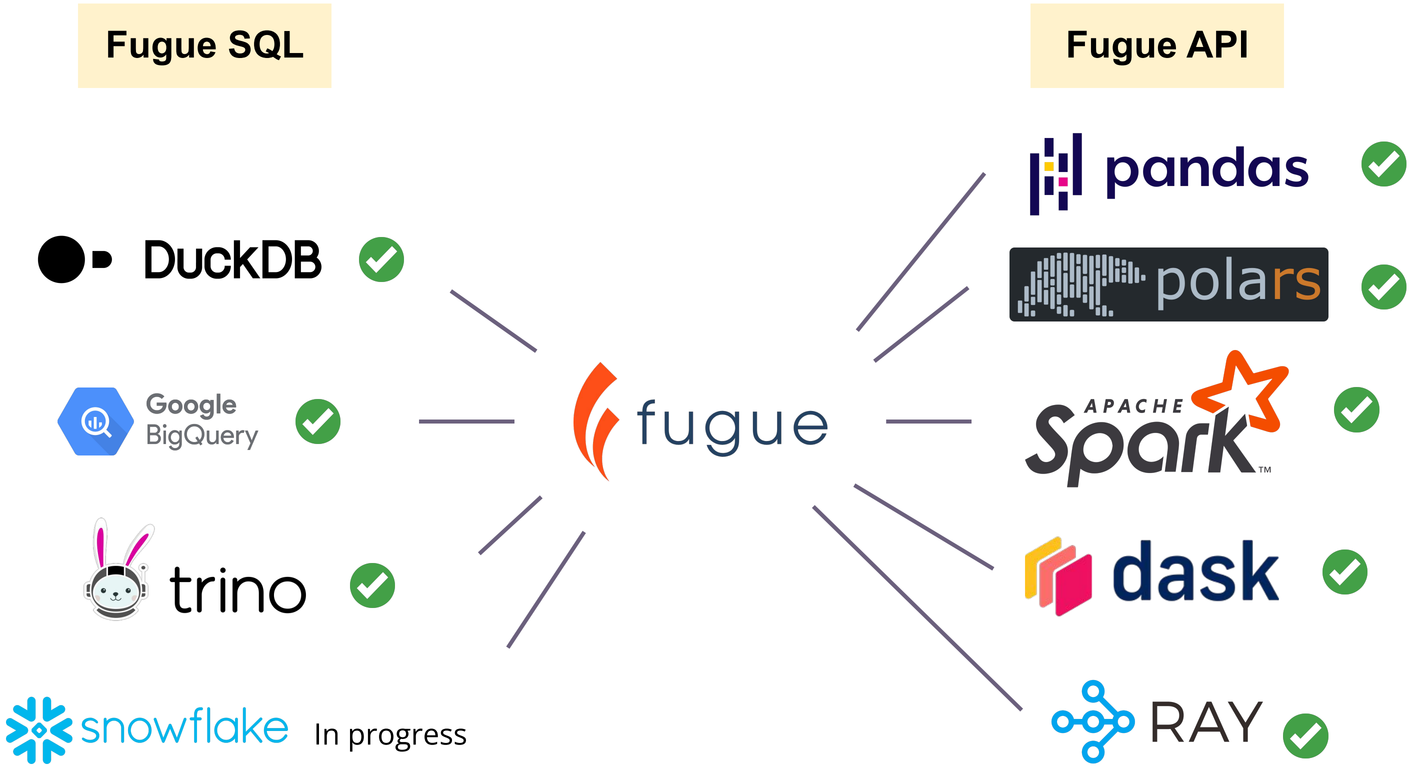
```
transform(
    ray_sample_dataset,
    zscore_pd,
    partition="uid",
    params=dict(n=PERIOD),
    schema="*",
)
```

```
transform(
    ray_dataset,
    zscore_pd,
    partition="uid",
    params=dict(n=PERIOD),
    schema="*",
)
```

Why Fugue?

- The core computing logic can remain untouched
- The migration is always reversible
- Existing unit/integration tests can still work
- The iterations will be faster
- The workflow becomes scale and platform agnostic

The Full Picture of Fugue Backends



Incremental SQL Development

A Complex SQL (TPC-DS #78)

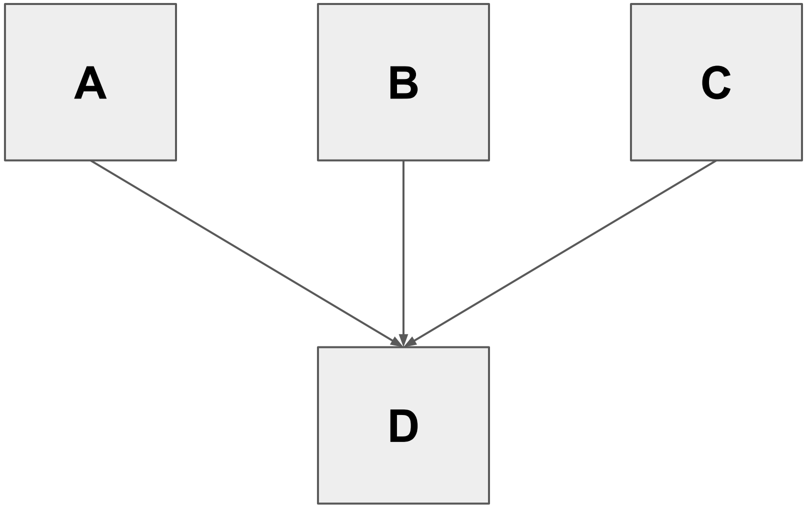
```sql
WITH ws
     AS (SELECT d_year                 AS ws_sold_year,
                ws_item_sk,
                ws_bill_customer_sk    ws_customer_sk,
                Sum(ws_quantity)       ws_qty,
                Sum(ws_wholesale_cost) ws_wc,
                Sum(ws_sales_price)    ws_sp
         FROM   web_sales
                LEFT JOIN web_returns
                       ON wr_order_number = ws_order_number
                          AND ws_item_sk = wr_item_sk
                JOIN date_dim
                  ON ws_sold_date_sk = d_date_sk
         WHERE  wr_order_number IS NULL
         GROUP  BY d_year,
                   ws_item_sk,
                   ws_bill_customer_sk),
     cs
     AS (SELECT d_year                 AS cs_sold_year,
                cs_item_sk,
                cs_bill_customer_sk    cs_customer_sk,
                Sum(cs_quantity)       cs_qty,
                Sum(cs_wholesale_cost) cs_wc,
                Sum(cs_sales_price)    cs_sp
         FROM   catalog_sales
                LEFT JOIN catalog_returns
                       ON cr_order_number = cs_order_number
                          AND cs_item_sk = cr_item_sk
                JOIN date_dim
                  ON cs_sold_date_sk = d_date_sk
         WHERE  cr_order_number IS NULL
         GROUP  BY d_year,
                   cs_item_sk,
                   cs_bill_customer_sk),
     ss
     AS (SELECT d_year                 AS ss_sold_year,
                ss_item_sk,
                ss_customer_sk,
                Sum(ss_quantity)       ss_qty,
                Sum(ss_wholesale_cost) ss_wc,
                Sum(ss_sales_price)    ss_sp
         FROM   store_sales
                LEFT JOIN store_returns
                       ON sr_ticket_number = ss_ticket_number
                          AND ss_item_sk = sr_item_sk
                JOIN date_dim
                  ON ss_sold_date_sk = d_date_sk
         WHERE  sr_ticket_number IS NULL
         GROUP  BY d_year,
                   ss_item_sk,
                   ss_customer_sk)
SELECT ss_item_sk,
               Round(ss_qty / ( COALESCE(ws_qty + cs_qty, 1) ), 2) ratio,
               ss_qty
store_qty,     ss_wc
               store_wholesale_cost,
               ss_sp
               store_sales_price,
               COALESCE(ws_qty, 0) + COALESCE(cs_qty, 0)
               other_chan_qty,
               COALESCE(ws_wc, 0) + COALESCE(cs_wc, 0)
               other_chan_wholesale_cost,
               COALESCE(ws_sp, 0) + COALESCE(cs_sp, 0)
               other_chan_sales_price
FROM   ss
       LEFT JOIN ws
              ON ( ws_sold_year = ss_sold_year
                   AND ws_item_sk = ss_item_sk
                   AND ws_customer_sk = ss_customer_sk )
       LEFT JOIN cs
              ON ( cs_sold_year = ss_sold_year
                   AND cs_item_sk = cs_item_sk
                   AND cs_customer_sk = ss_customer_sk )
WHERE  COALESCE(ws_qty, 0) > 0
       AND COALESCE(cs_qty, 0) > 0
       AND ss_sold_year = 1999
ORDER  BY ss_item_sk,
          ss_qty DESC,
          ss_wc DESC,
          ss_sp DESC,
          other_chan_qty,
          other_chan_wholesale_cost,
          other_chan_sales_price,
          Round(ss_qty / ( COALESCE(ws_qty + cs_qty, 1) ), 2)
```

A

B

C

D



Why SQL development is hard

- How to iterate on each individual step?
- If it is slow, what is the cause?
- How to test?

Remove CTE and break up steps (sample to local)

```
WITH ws
     AS (SELECT d_year                      AS ws_sold_year,
                ws_item_sk,
                ws_bill_customer_sk        ws_customer_sk,
                Sum(ws_quantity)           ws_qty,
                Sum(ws_wholesale_cost) ws_wc,
                Sum(ws_sales_price)        ws_sp
         FROM   web_sales
                LEFT JOIN web_returns
                       ON wr_order_number = ws_order_number
                          AND ws_item_sk = wr_item_sk
                JOIN date_dim
                  ON ws_sold_date_sk = d_date_sk
         WHERE  wr_order_number IS NULL
         GROUP  BY d_year,
                   ws_item_sk,
                   ws_bill_customer_sk),
     cs
     AS (SELECT d_year                      AS cs_sold_year,
                cs_item_sk,
```

```
%%fsql spark
SELECT d_year                      AS ws_sold_year,
         ws_item_sk,
         ws_bill_customer_sk        ws_customer_sk,
         Sum(ws_quantity)           ws_qty,
         Sum(ws_wholesale_cost) ws_wc,
         Sum(ws_sales_price)        ws_sp
 FROM    web_sales
         LEFT JOIN web_returns
                ON wr_order_number = ws_order_number
                   AND ws_item_sk = wr_item_sk
         JOIN date_dim
           ON ws_sold_date_sk = d_date_sk
 WHERE   wr_order_number IS NULL
 GROUP   BY d_year,
            ws_item_sk,
            ws_bill_customer_sk

SAMPLE 1%
YIELD LOCAL DATAFRAME AS ws
```

Remove CTE and break up steps (to file)

```
cs
AS (SELECT d_year                  AS cs_sold_year,
           cs_item_sk,
           cs_bill_customer_sk     cs_customer_sk,
           Sum(cs_quantity)        cs_qty,
           Sum(cs_wholesale_cost)  cs_wc,
           Sum(cs_sales_price)     cs_sp
    FROM   catalog_sales
           LEFT JOIN catalog_returns
                  ON cr_order_number = cs_order_number
                     AND cs_item_sk = cr_item_sk
           JOIN date_dim
             ON cs_sold_date_sk = d_date_sk
    WHERE  cr_order_number IS NULL
    GROUP  BY d_year,
              cs_item_sk,
              cs_bill_customer_sk),
ss
AS (SELECT d_year                  AS ss_sold_year,
```

```
%%fsql spark
SELECT d_year                  AS cs_sold_year,
       cs_item_sk,
       cs_bill_customer_sk     cs_customer_sk,
       Sum(cs_quantity)        cs_qty,
       Sum(cs_wholesale_cost)  cs_wc,
       Sum(cs_sales_price)     cs_sp
 FROM  catalog_sales
       LEFT JOIN catalog_returns
              ON cr_order_number = cs_order_number
                 AND cs_item_sk = cr_item_sk
       JOIN date_dim
         ON cs_sold_date_sk = d_date_sk
 WHERE cr_order_number IS NULL
 GROUP BY d_year,
          cs_item_sk,
          cs_bill_customer_sk

YIELD FILE AS cs
```

Switch to local development

```
SELECT ss_item_sk,
               Round(ss_qty / ( COALESCE(ws_qty + cs_qty, 1) ), 2) ratio,
               ss_qty                                              store_qty,
               ss_wc
               store_wholesale_cost,
               ss_sp
               store_sales_price,
               COALESCE(ws_qty, 0) + COALESCE(cs_qty, 0)
               other_chan_qty,
               COALESCE(ws_wc, 0) + COALESCE(cs_wc, 0)
               other_chan_wholesale_cost,
               COALESCE(ws_sp, 0) + COALESCE(cs_sp, 0)
               other_chan_sales_price
FROM   ss
       LEFT JOIN ws
              ON ( ws_sold_year = ss_sold_year
                   AND ws_item_sk = ss_item_sk
                   AND ws_customer_sk = ss_customer_sk )
       LEFT JOIN cs
              ON ( cs_sold_year = ss_sold_year
                   AND cs_item_sk = cs_item_sk
                   AND cs_customer_sk = ss_customer_sk )
```

```
%%fsql duckdb
SELECT ss_item_sk,
               Round(ss_qty / (COALESCE(ws_qty+cs_qty,1)),2) ratio,
               ss_qty                                        store_qty,
               ss_wc
               store_wholesale_cost,
               ss_sp
               store_sales_price,
               COALESCE(ws_qty, 0) + COALESCE(cs_qty, 0)
               other_chan_qty,
               COALESCE(ws_wc, 0) + COALESCE(cs_wc, 0)
               other_chan_wholesale_cost,
               COALESCE(ws_sp, 0) + COALESCE(cs_sp, 0)
               other_chan_sales_price
FROM   ss
       LEFT JOIN ws
              ON ( ws_sold_year = ss_sold_year
                   AND ws_item_sk = ss_item_sk
                   AND ws_customer_sk = ss_customer_sk )
       LEFT JOIN cs
              ON ( cs_sold_year = ss_sold_year
                   AND cs_item_sk = cs_item_sk
                   AND cs_customer_sk = ss_customer_sk )

PRINT
```

Assemble the final SQL

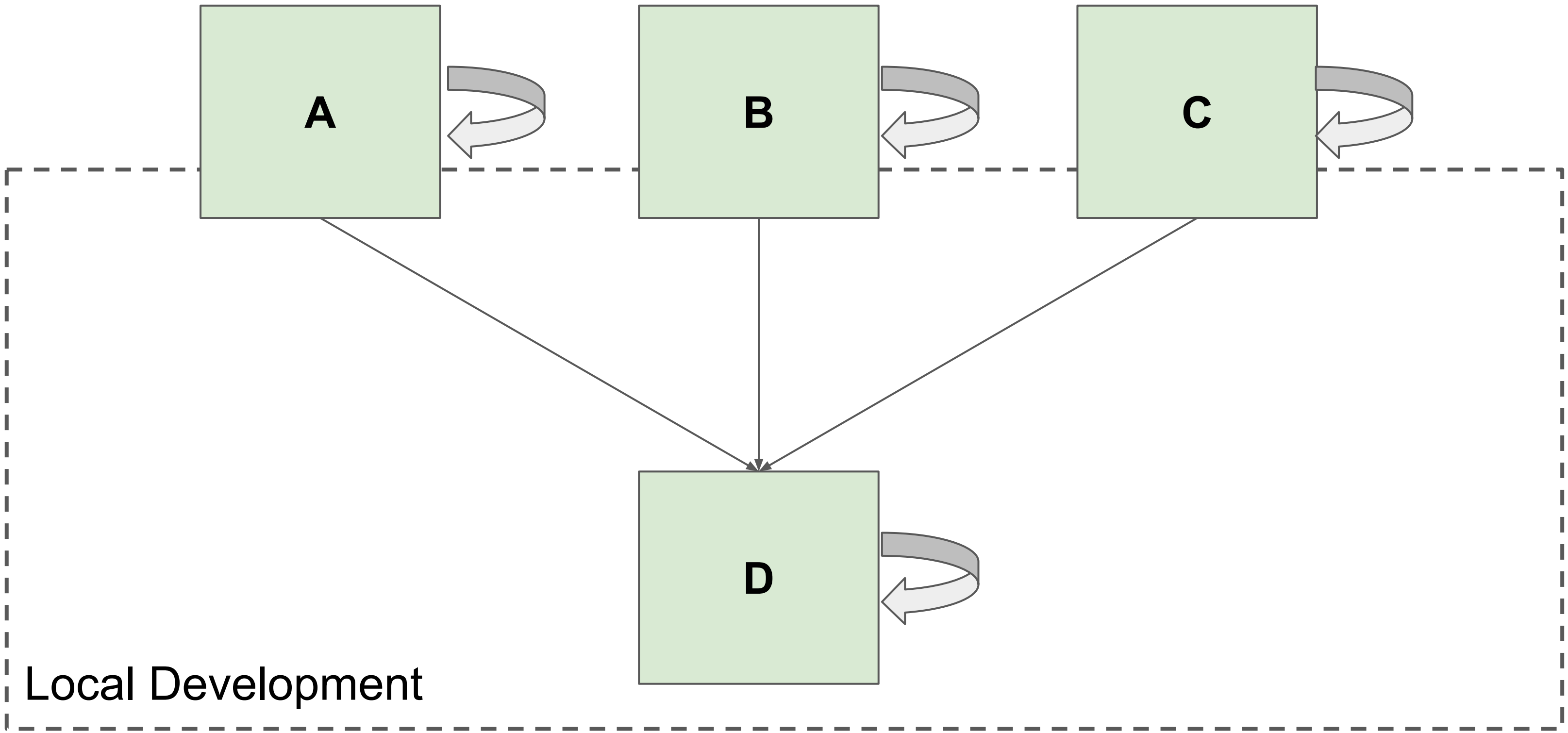
```
%%fsql spark
ss =
    SELECT ...

ws =
    SELECT ...

cs =
    SELECT ...

result =
    SELECT ...
    FROM ss
        LEFT JOIN ws ON ...
        LEFT JOIN cs ON ...
    YIELD DATAFRAME
```

The Whole Process



Test SQL

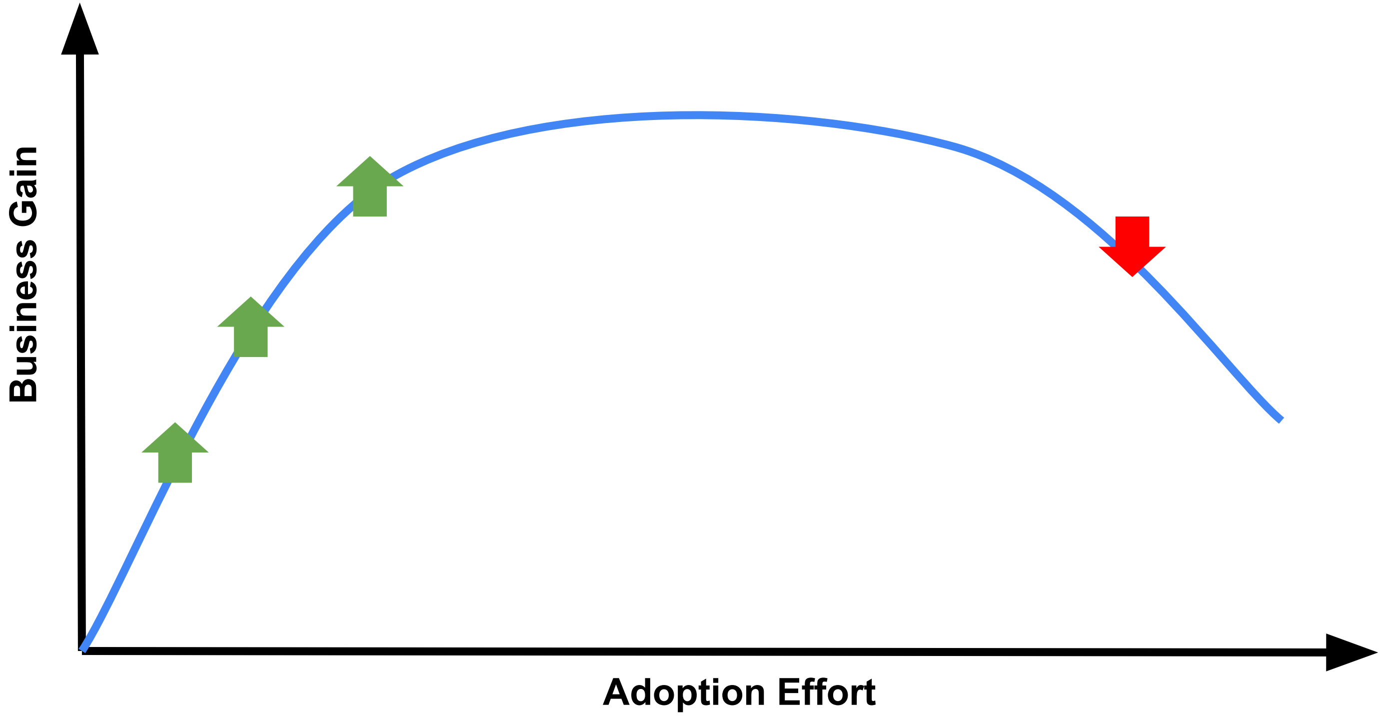
```sql
SELECT
    DATE_TRUNC("HOUR", pickup_datetime) AS ts,
    pu_location,
    COUNT(*) AS pu
FROM hive.src_table
GROUP BY 1, 2
```

```python
from fugue_sql import fsql

fsql(sql, {"hive.src_table": pd_dataframe}).run("duckdb")
```

Key Takeaways

Strategical incremental adoption to maximize business gain



Thank You

https://github.com/fugue-project/fugue