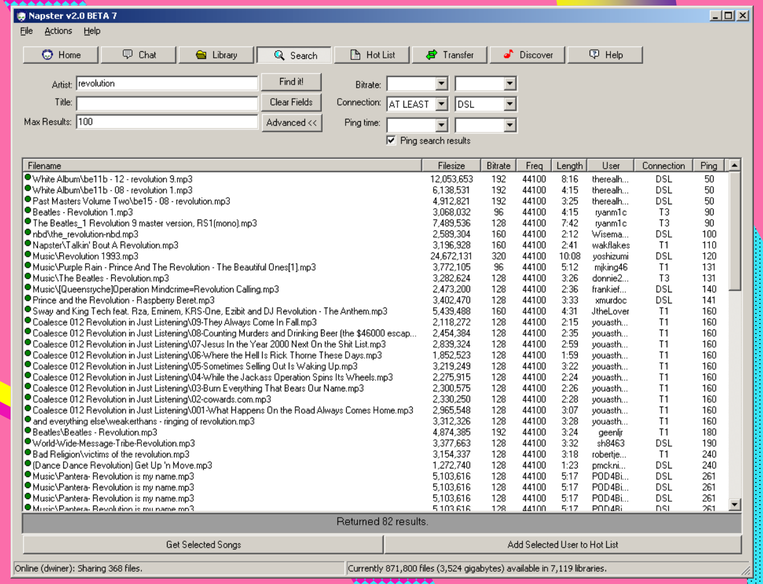
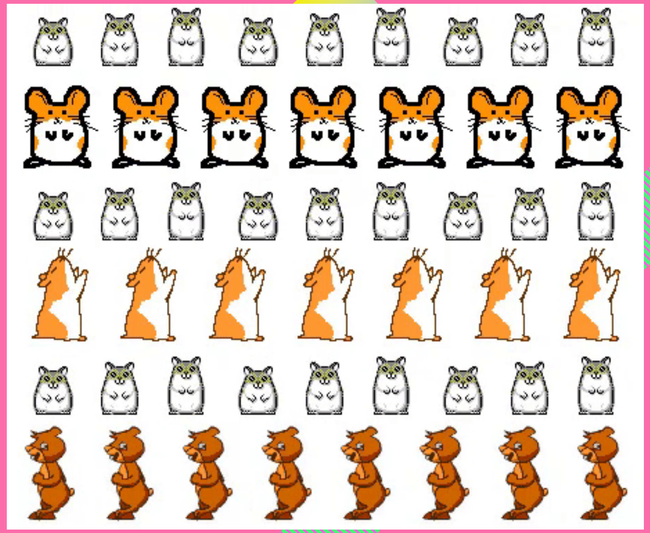
Cross-Server Data Joins on Slow Networks with Python

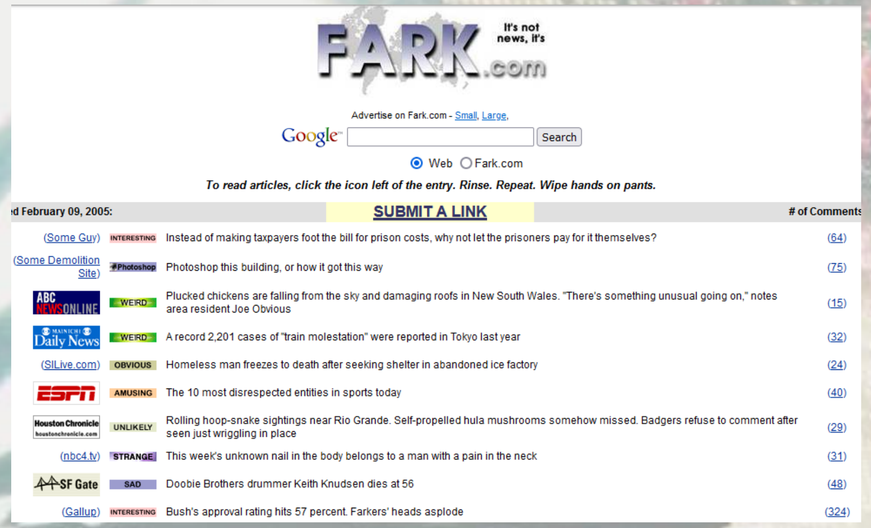
Bert Wagner
PyCon 2023

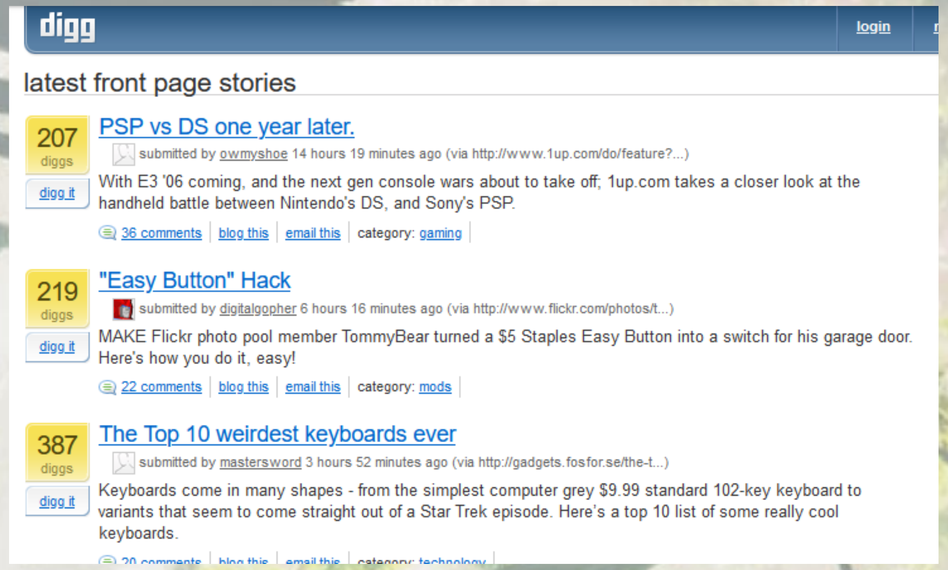
“MOM, DON’T USE THE PHONE”

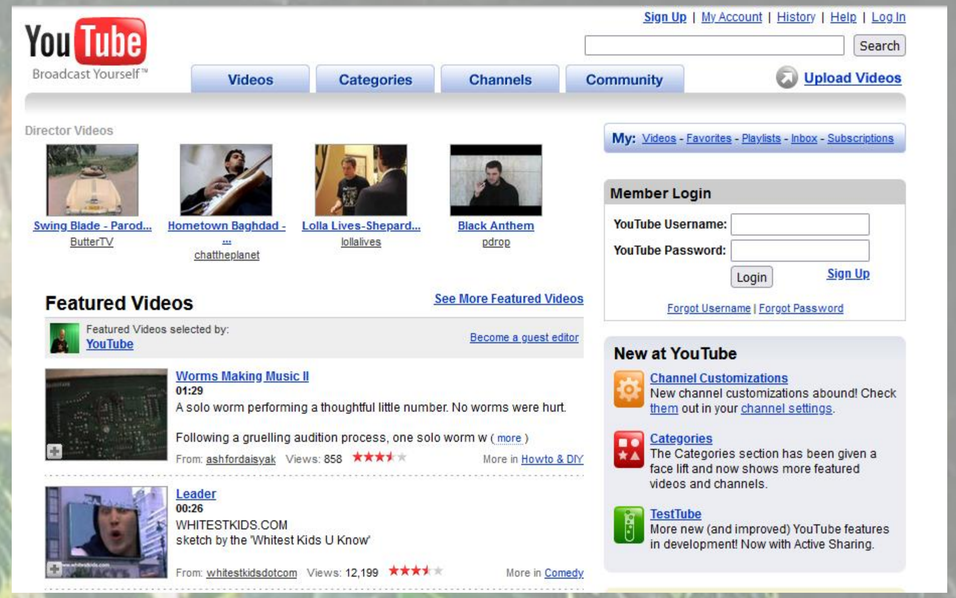




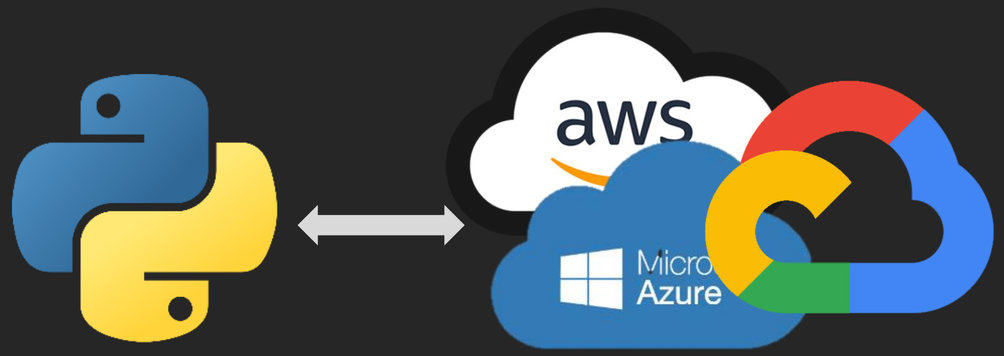
"CHECK THIS OUT"







☠️ “STOP STREAMING!” ☠️



BERT WAGNER

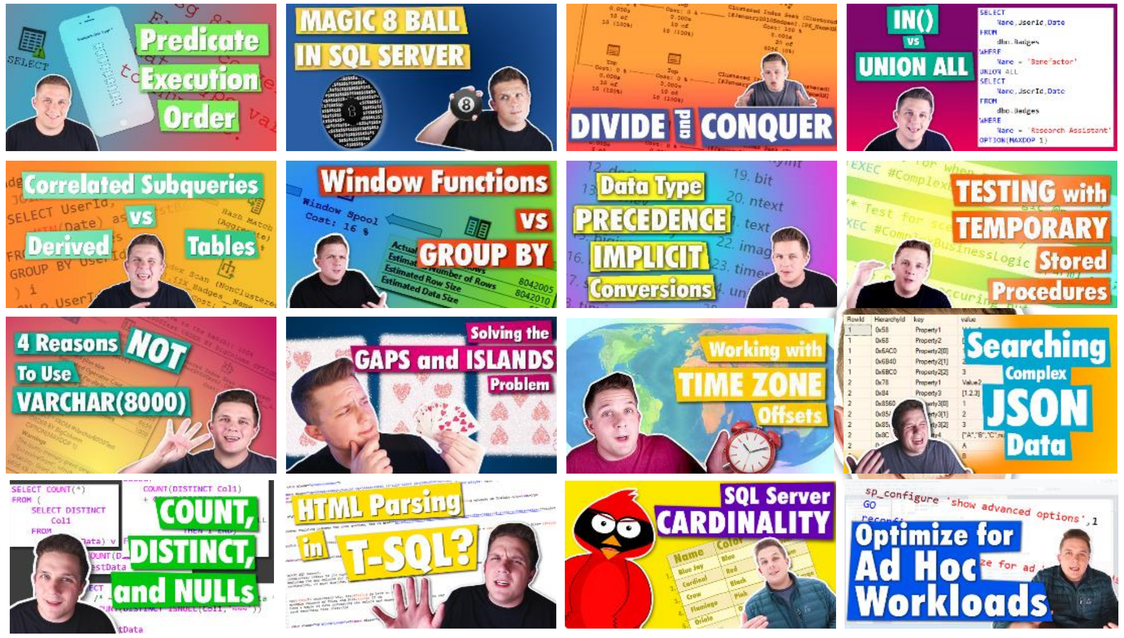
Data Scientist, YouTuber, Blogger

Code and slides:
bertwagner.com/crosstream

Contact:
bertwagner.com
bertwagner@bertwagner.com
@bertwagner

DATA WITH BERT

YouTube



AGENDA

Joining datasets across networks efficiently

- Great Solutions
- Not Bad Solutions
- Ugly Solutions

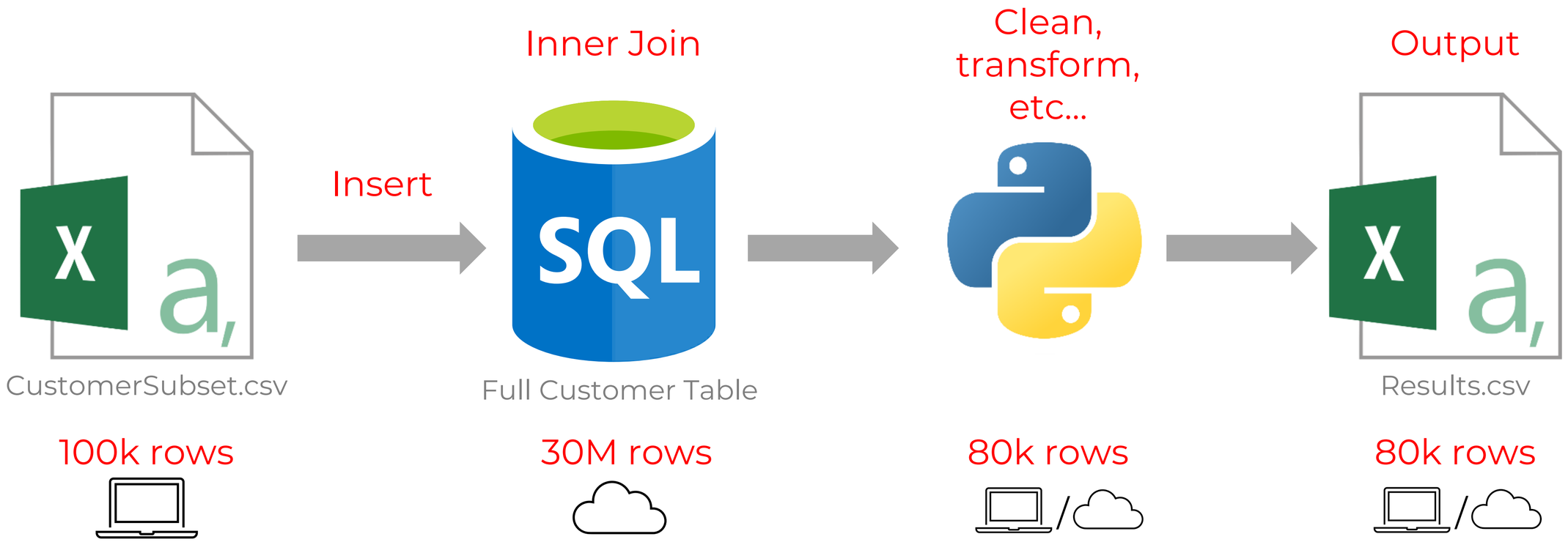
SCENARIO

CustomerSubset.csv
100k rows
2GB
Local

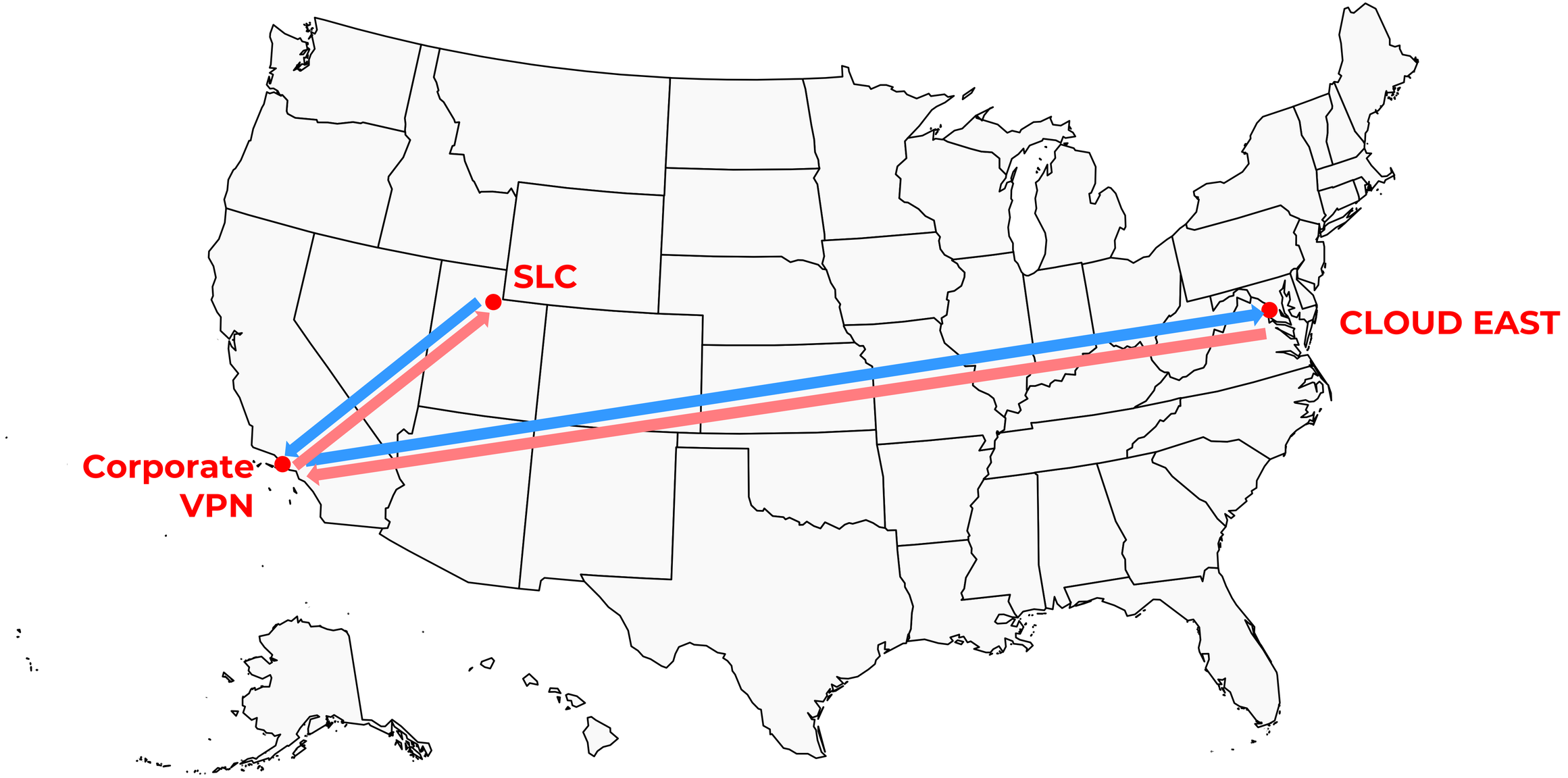
All Customer Data
30M rows
20GB
Remote

How do we join these datasets together?

SOLUTION: GREAT



SOLUTION: GREAT



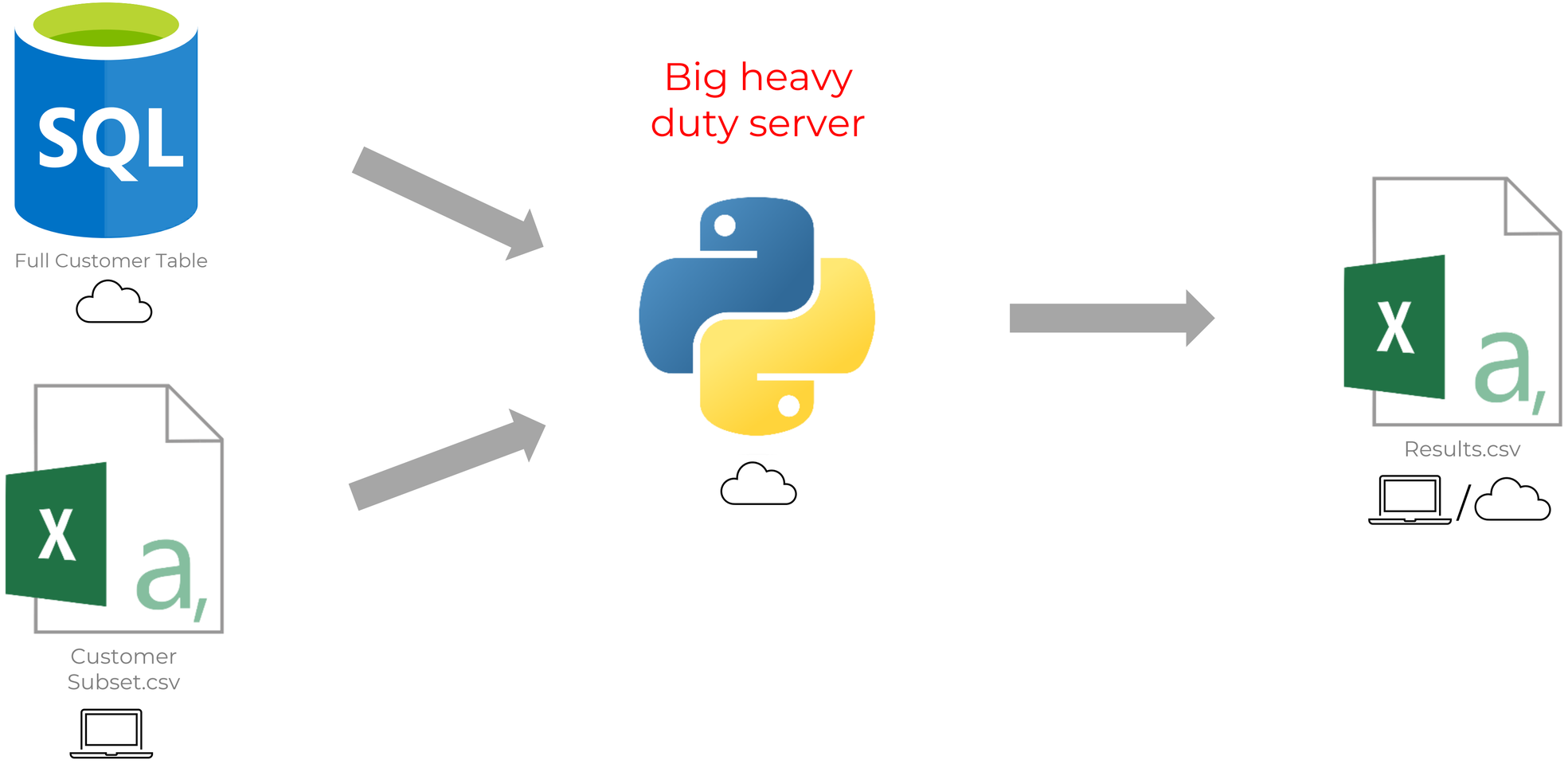
SUMMARY: GREAT SOLUTION

- PRO: Minimizes network movement
- PRO: Maximizes using tools for what they are good at
- CON: Might be paying for expensive cloud compute or licensing
- CON: Not good at doing transformations to data before you join

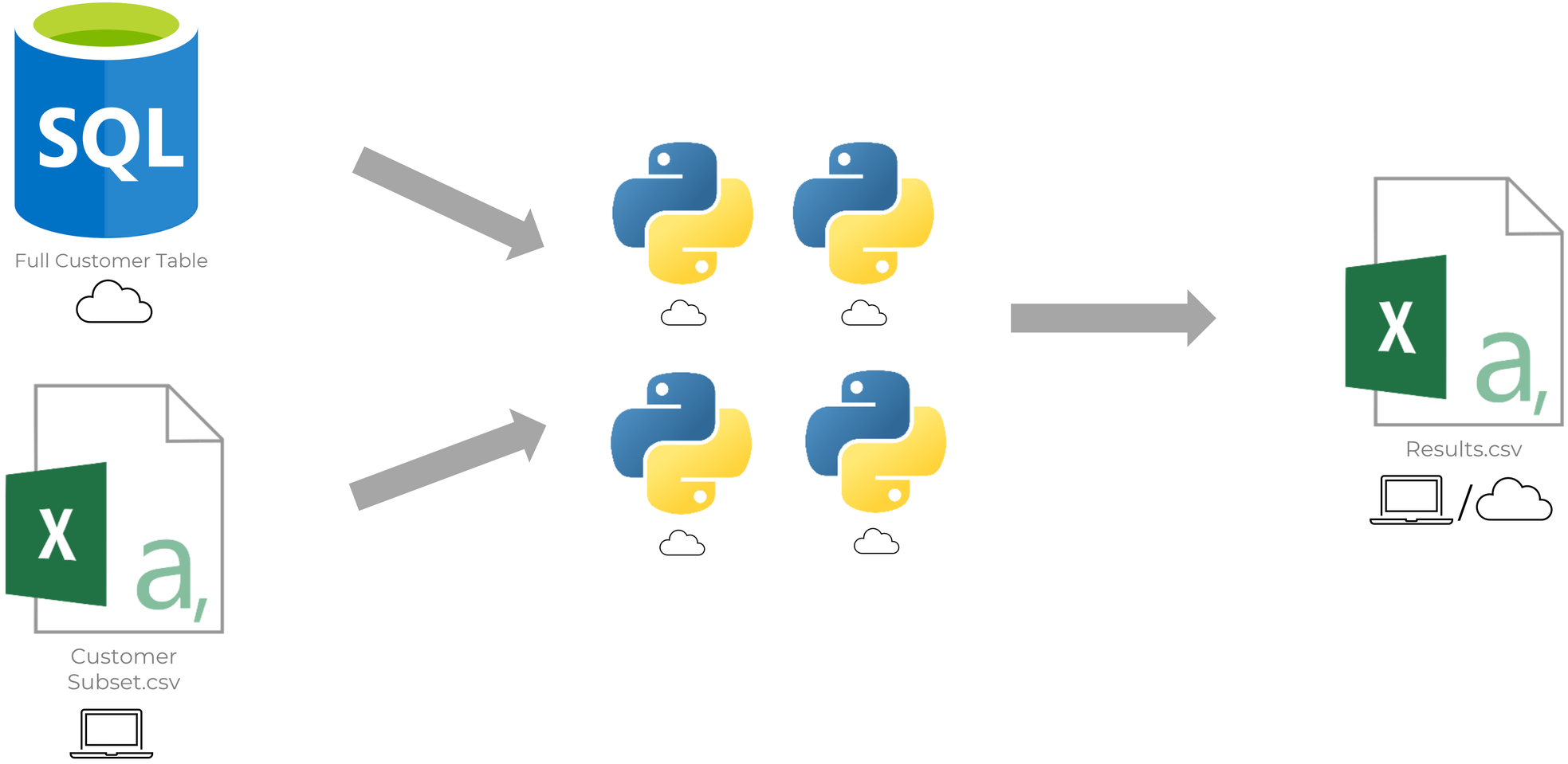
More resources:

- SQL Anti-Patterns for Analysts
- Database Indexing for Beginners

SOLUTION: NOT BAD



SOLUTION: NOT BAD



SUMMARY: NOT BAD SOLUTION

- PRO: If large data on same network as servers, performance is good
- PRO: If you have a pool of machines, might be able to parallelize
- PRO: Powerful server will allow you to transform data for your join keys
- CON: You need a big high performance server $$$
- CON: Your server needs to be on the same network as your large dataset

More resources:

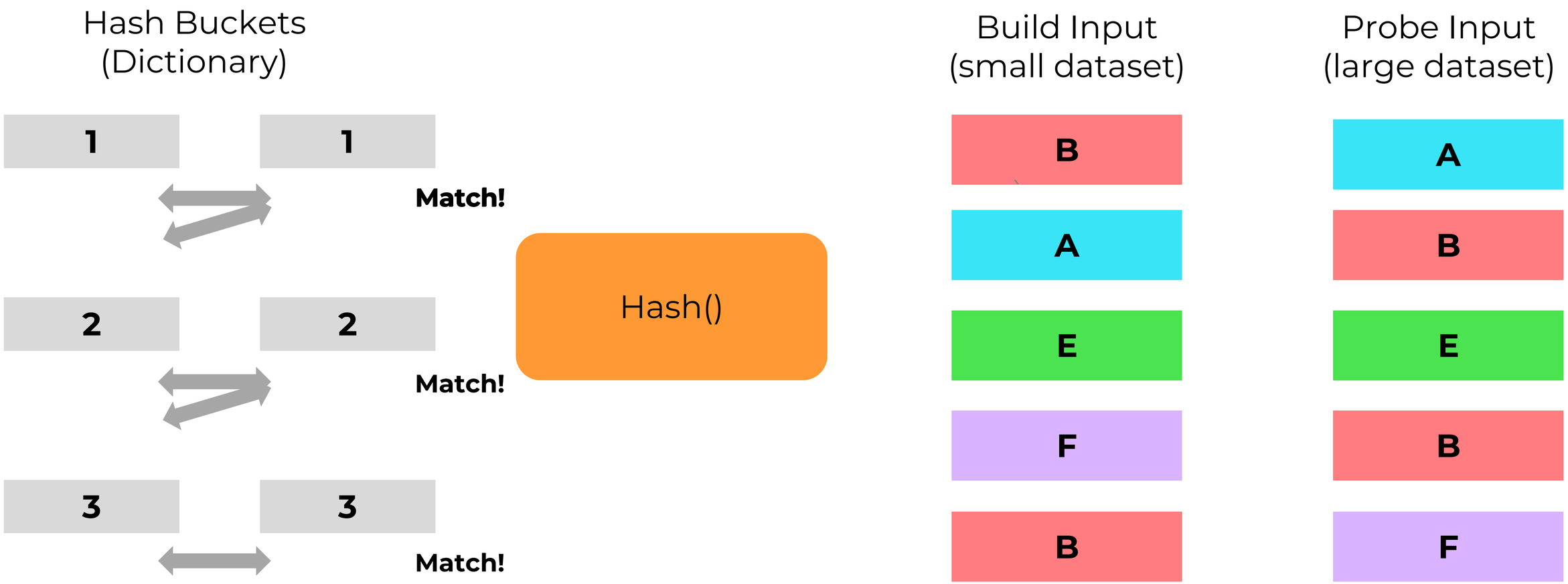
- Pandas: Scaling to large datasets

SOLUTION: UGLY

Let’s write our own hash join!

HASH JOIN THEORY

https://bertwagner.com/posts/hash-match-join-internals/



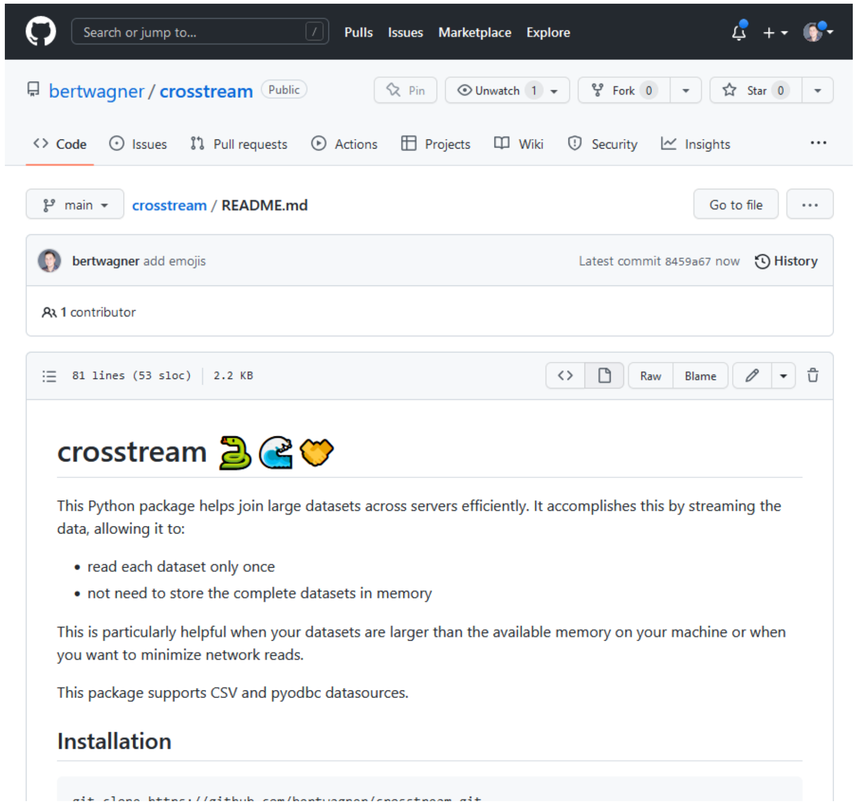
CROSSTREAM

Install from PyPI:

```
pip install crosstream
```

Or from source:

```
git clone https://github.com/bertwagner/crosstream.git
cd crosstream
pip install .
```



github.com/bertwagner/crosstream

DEMO:

BASIC USAGE

```
import crosstream as cs
import csv

file1 = 'small_dataset.csv'
file2 = 'large_dataset.csv'

# join using column indexes or column names
c1 = cs.read_csv(file1,True,[0,1])
c2 = cs.read_csv(file2, True, ['col1','col2'])

# specify the output file
with open('joined_output.csv', 'w') as f:
    w = csv.writer(f)

    # write header column names
    w.writerow(c1.column_names + c2.column_names)

    for row_left,row_right in cs.inner_hash_join(c1,c2):
        # write matched results to our joined_output.csv
        w.writerow(row_left + row_right)
```

DEMO:

CUSTOM JOIN KEYS

```
# define a function for joining on criteria that is modified before
# inserting into the hash table
def custom_join_key(row,indices):
    # calculate the hash of join values
    join_values = []
    join_key_values = []
    for col_index in indices:
        # here we transform our join key, removing any spaces from
        # our values
        col_value = str(row[col_index]).replace(' ','')
        join_values.append(str(hash(col_value)))
        join_key_values.append(col_value)
    join_key = ''.join(join_values)

    return join_key, join_key_values
```

```
...
for row_left,row_right in cs.inner_hash_join(c1,c2,
                override_build_join_key=custom_join_key):
    # write matched results to our joined_output.csv
    w.writerow(row_left + row_right)
...
```

DEMO:

CUSTOM MATCH PROCESSING

```
# define a function for performing additional transformations or
adding additional outputs before the columns are returned
def custom_process_matched_hashes(bucket_row,probe_row,
        bucket_join_column_indexes, probe_join_column_indexes):

    # adding a new column indicating the weights of these matches
    # are equal to 1
    weight=1.0
    return tuple(bucket_row),tuple(probe_row),(weight,)

...
for row_left,row_right in cs.inner_hash_join(c1,c2,
    override_process_matched_hashes=custom_process_matched_hashes):

    # write matched results to our joined_output.csv
    w.writerow(row_left + row_right)
...
```

SUMMARY: UGLY SOLUTION

- Last resort option
- Slow but reliable
- Can be programmed to restart on network failures
- Allows for heavy data transformations before and after joining
- Reads each dataset only once
- Works on CSV and ODBC
- Assuming you can't fit data in memory or on your laptop's disk
  - If only memory constrained, can use Dask to swap dataframes from disk

THANK YOU!

@bertwagner

bertwagner.com

youtube.com/DataWithBert

bertwagner@bertwagner.com