
Image: National Geographic

Conclave

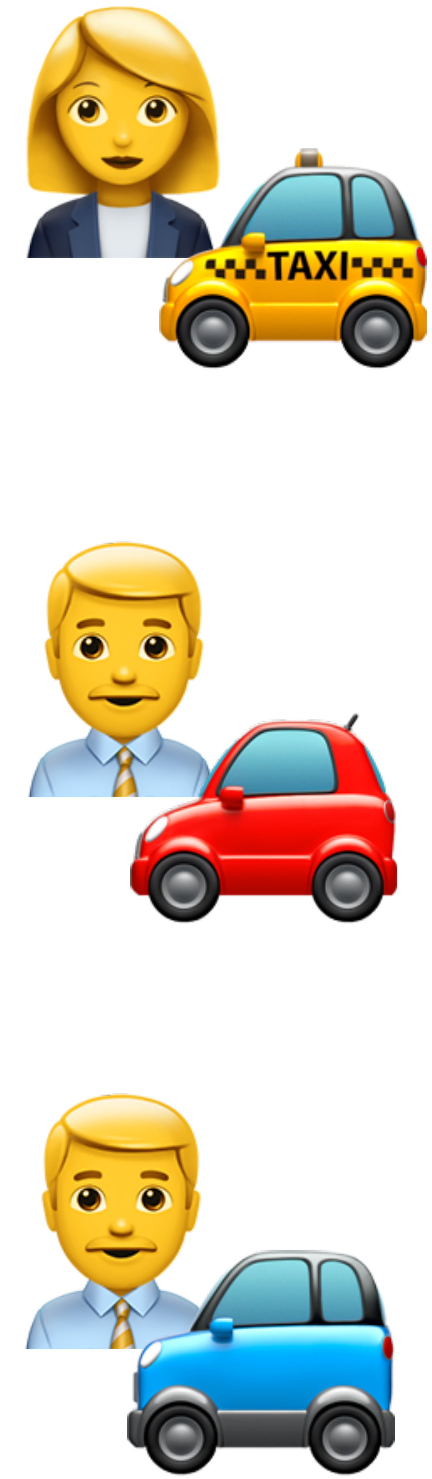
Secure Multi-Party Computation on Big Data

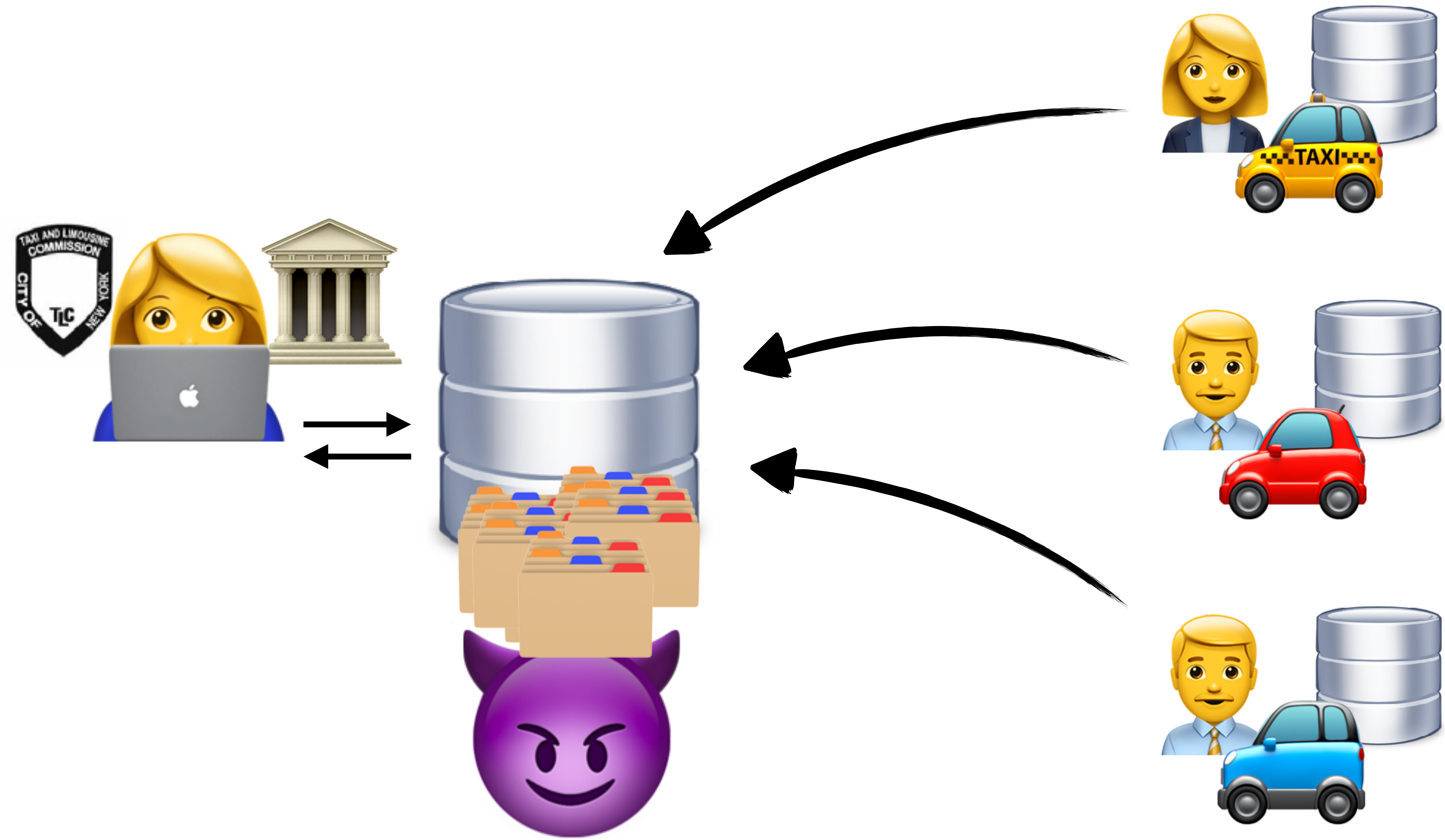
Nikolaj Volgushev **Malte Schwarzkopf** Ben Getchell
Andrei Lapets Mayank Varia Azer Bestavros

How about trips in Manhattan?

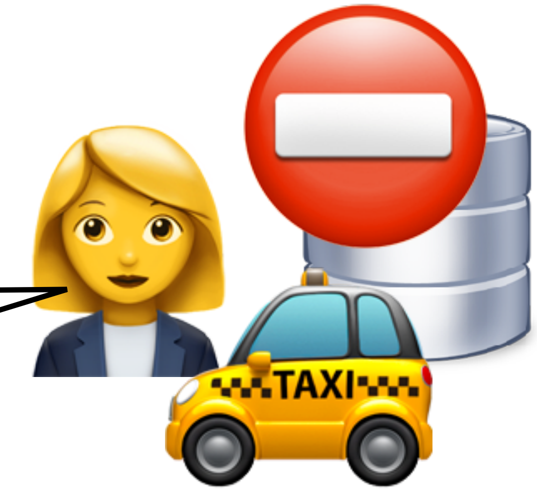
How about the morning rush hour?

How concentrated is the market airport transfers in hired vehicles in NYC?

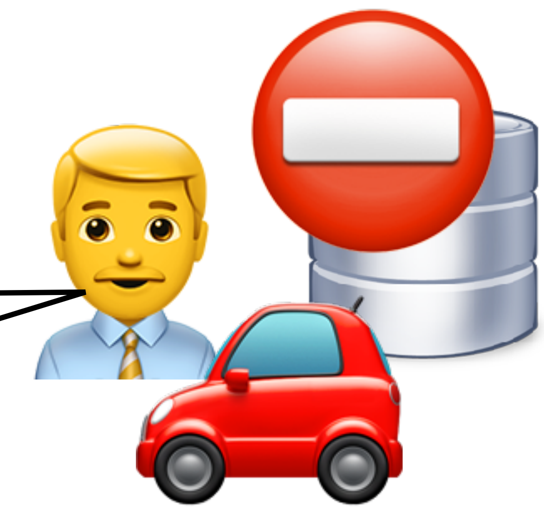



TAXI AND LIMOUSINE COMMISSION
CITY OF NEW YORK
TLC
TAXI

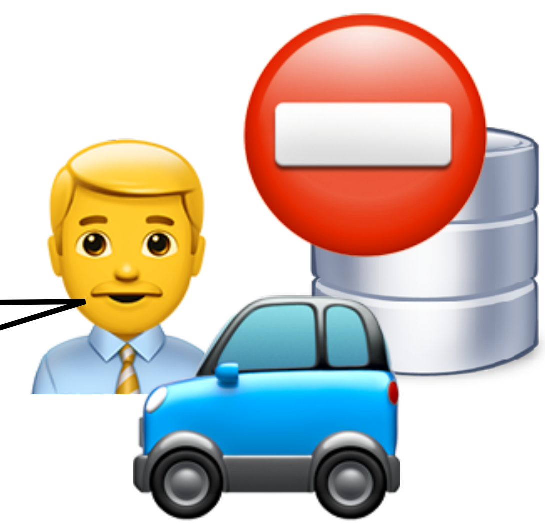
We’re worried about information leaking.



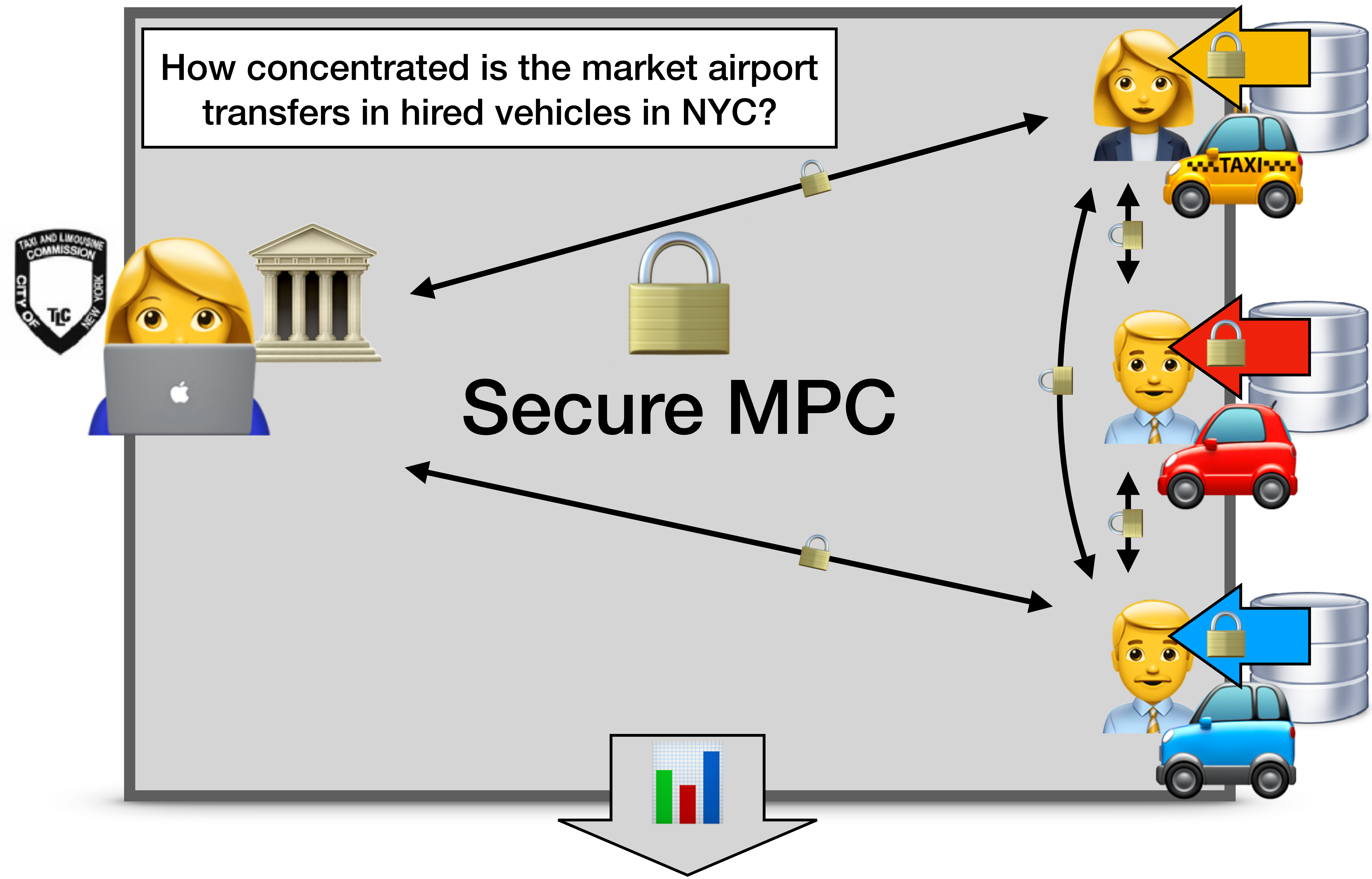
Competitors might see our pricing.



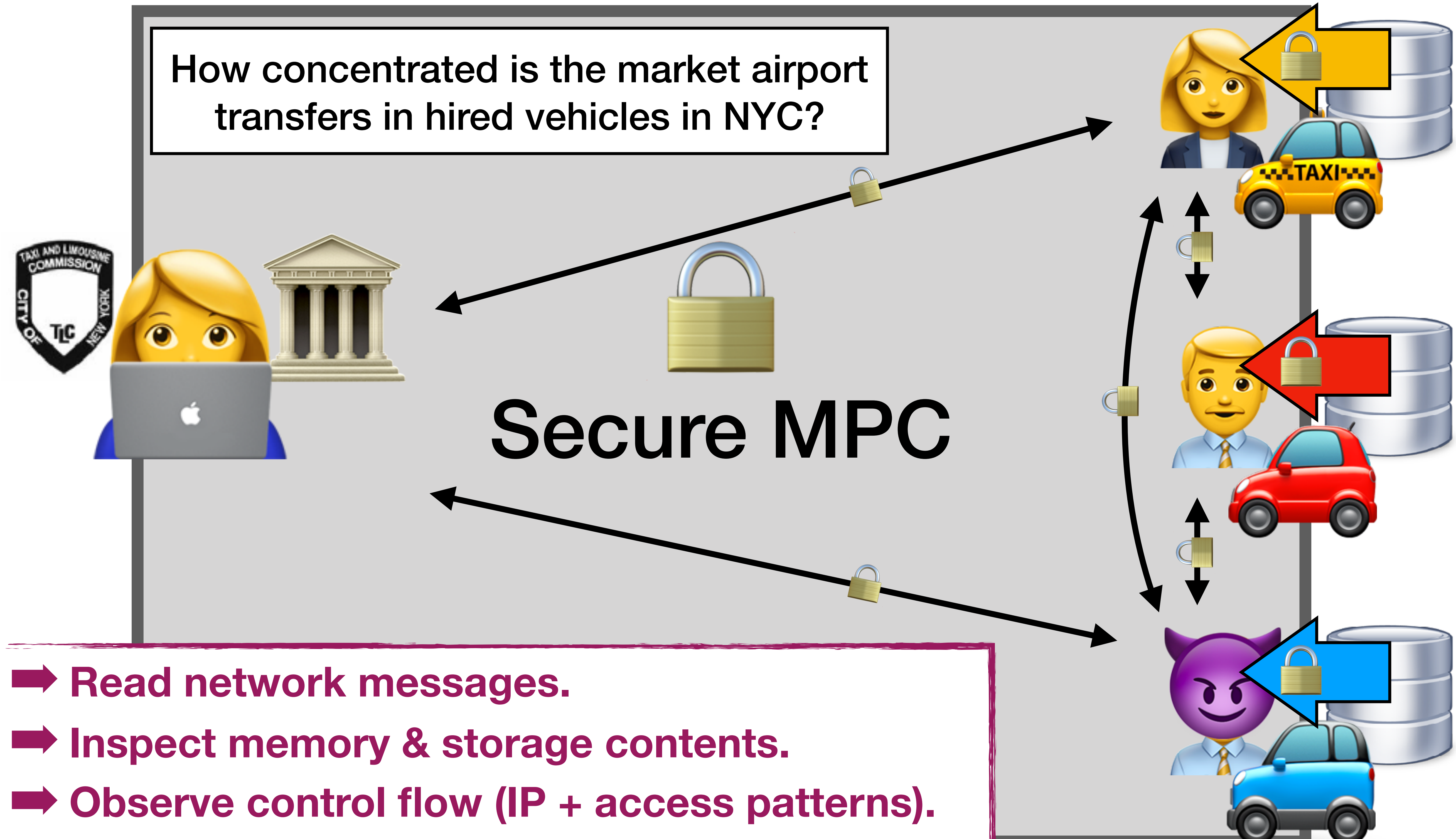
We’re a private company — no obligation to report.



A solution: Secure MPC



A solution: Secure MPC



➡ Read network messages.

➡ Inspect memory & storage contents.

➡ Observe control flow (IP + access patterns).

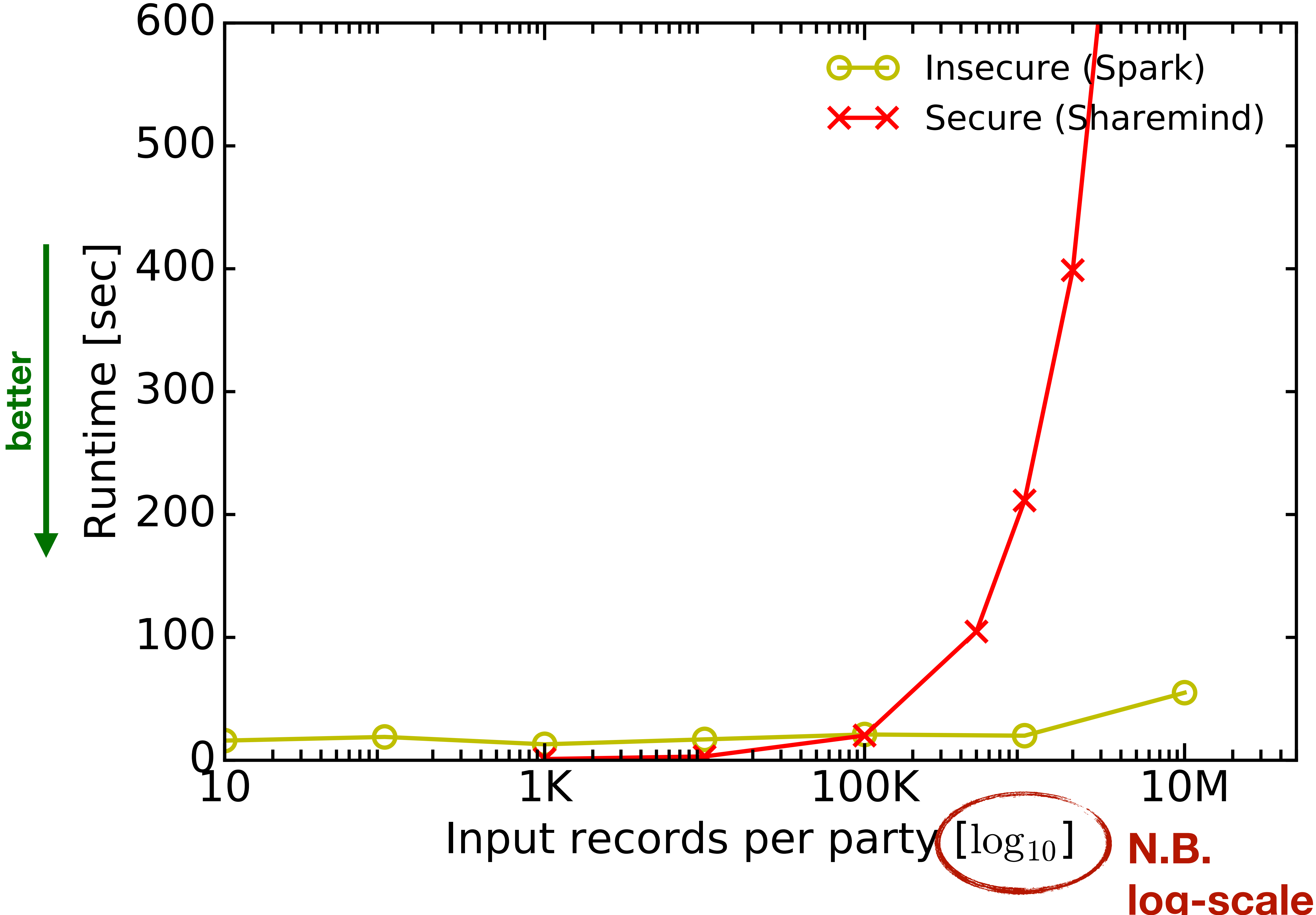
✓ Only local input and result are exposed.


TAXI AND LIMOUSINE COMMISSION
CITY OF NEW YORK
TLC
Secure MPC
175M annual trips!
TAXI

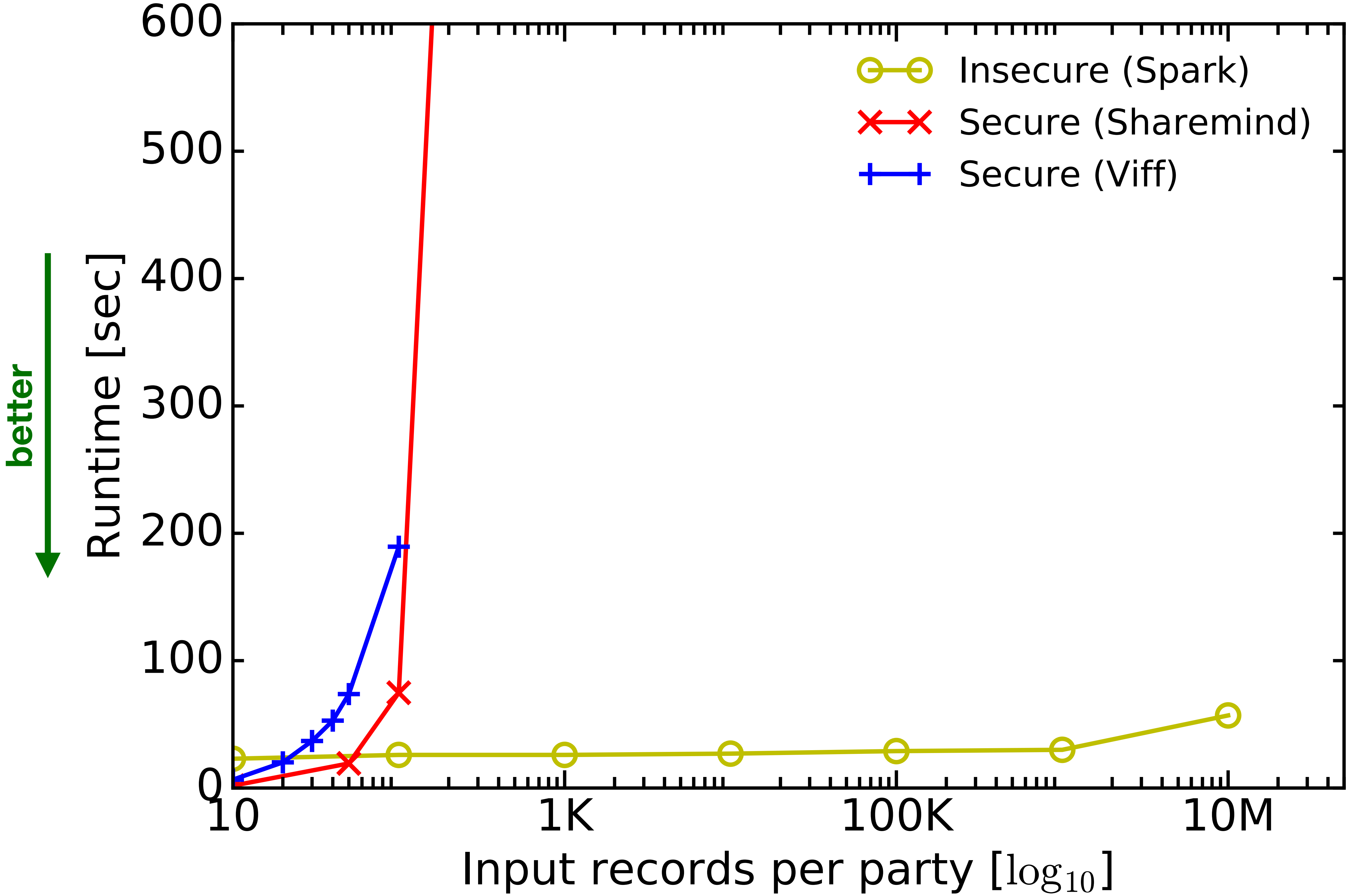
Does MPC scale?

- **Garbled circuits**: 1 wire = 1 bit = 1 label
  - Scales poorly in space: state >> input size
- **Secret sharing**: multiplication = network I/O
  - Vectorization & batching help, but only so much
  - Mostly small computations with few operations
- Lots of research on scaling to **many parties**, but little on scaling to **large input data**

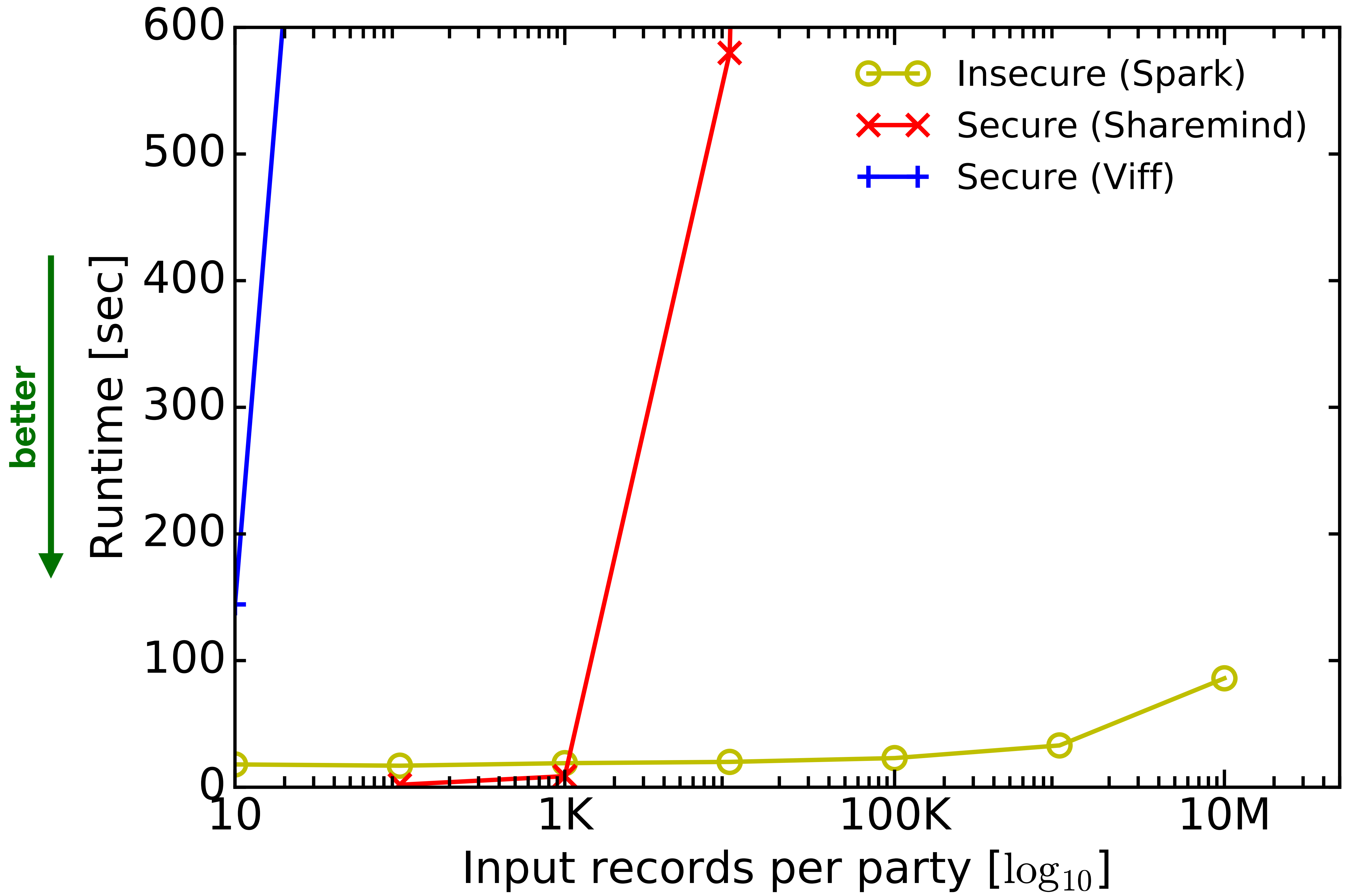
Does MPC scale? — Projection

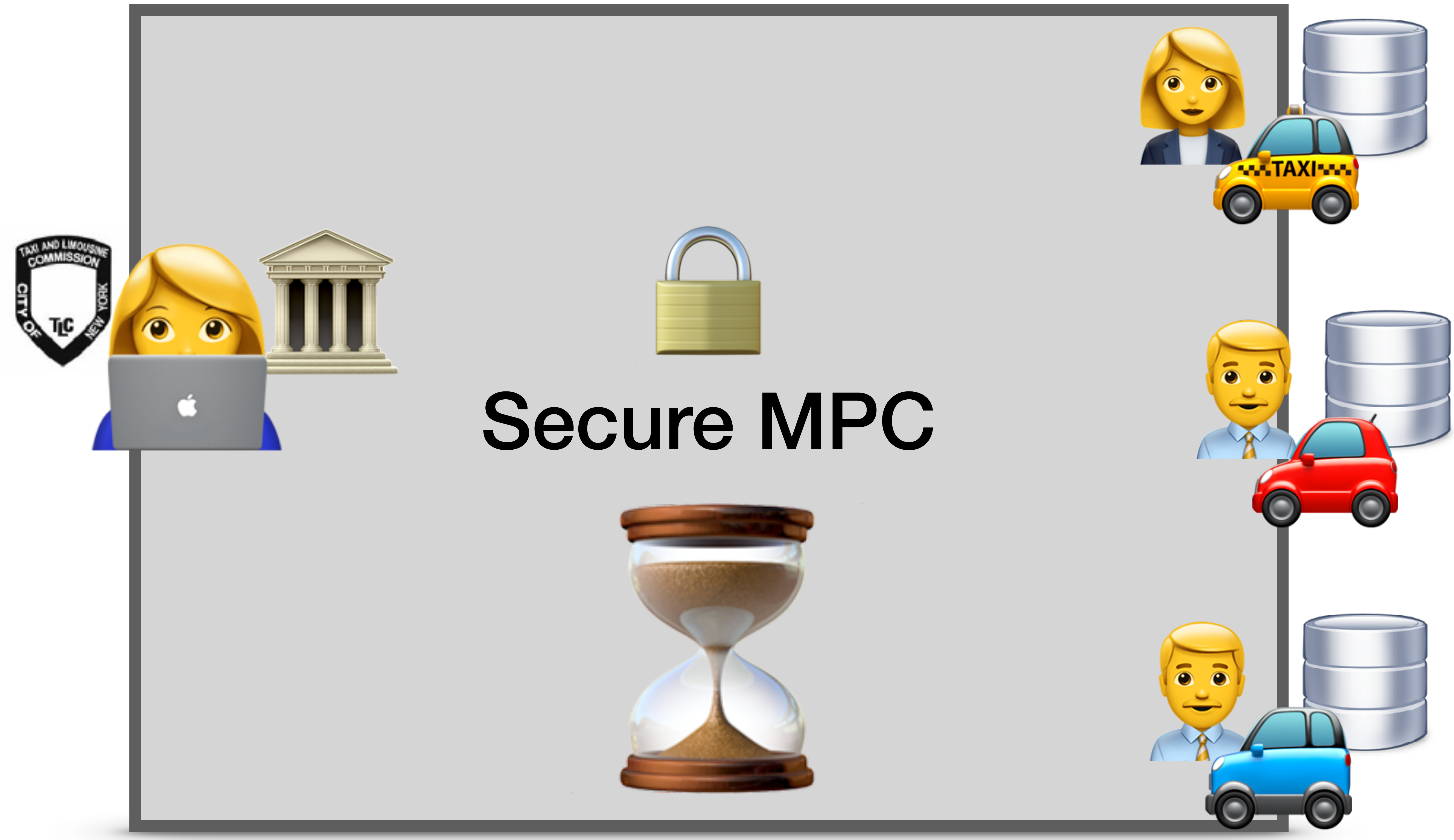


Does MPC scale? — Agg: SUM



Does MPC scale? — Join




Secure MPC
TAXI AND LIMOUSINE COMMISSION
CITY OF NEW YORK
TLC
TAXI

Conclave

Key insights:

For most queries, not all of the work **must** happen using cryptographic MPC techniques.

End-to-end guarantees can often be maintained even if part of the query is evaluated locally and in the clear by the parties.

We can automatically determine the MPC bounds.

Contributions

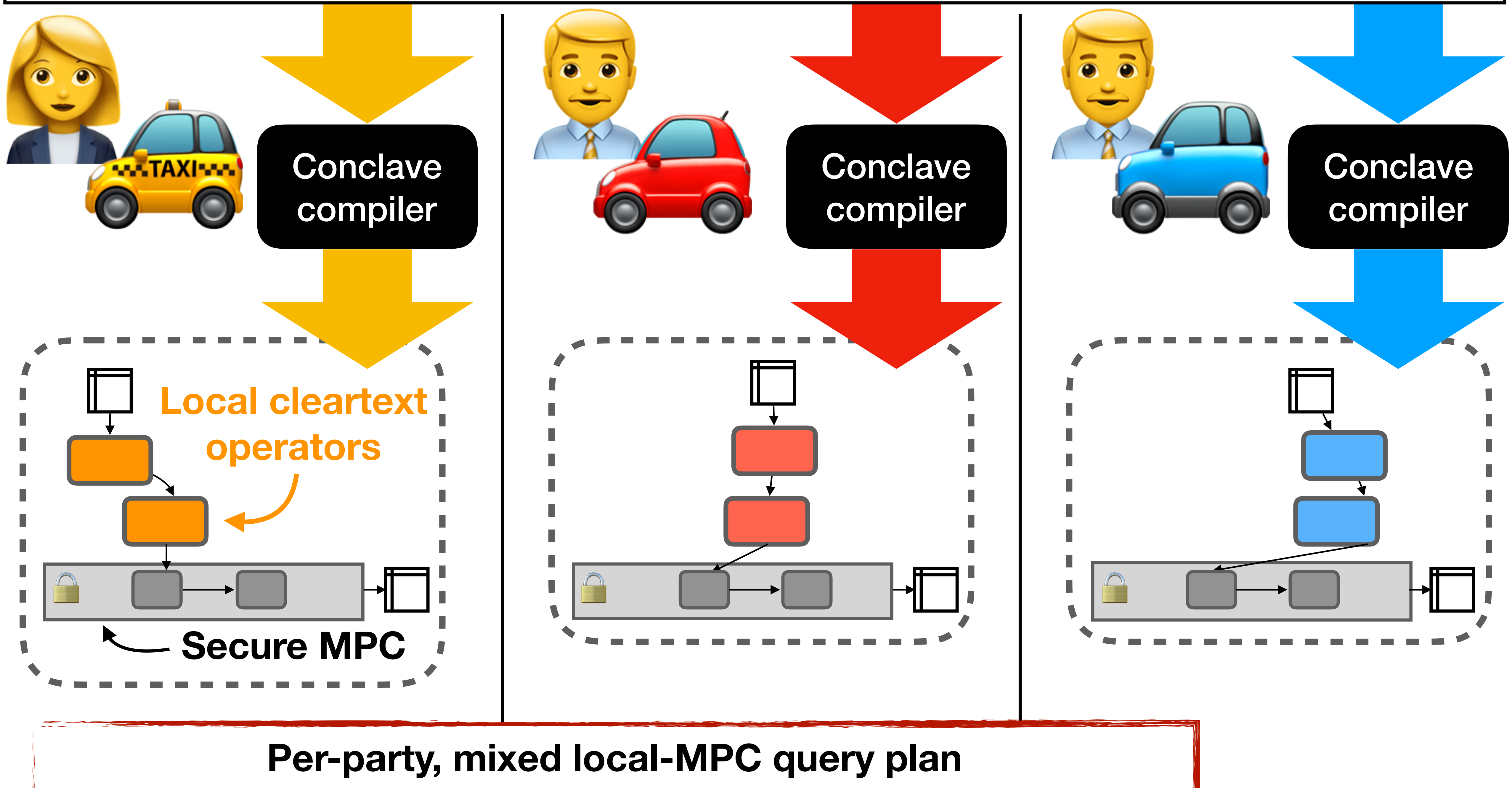
1. **Conclave paradigm:** run as little as possible, but as much as necessary under MPC

2. **Automated analyses** to determine which parts of a query must run under MPC

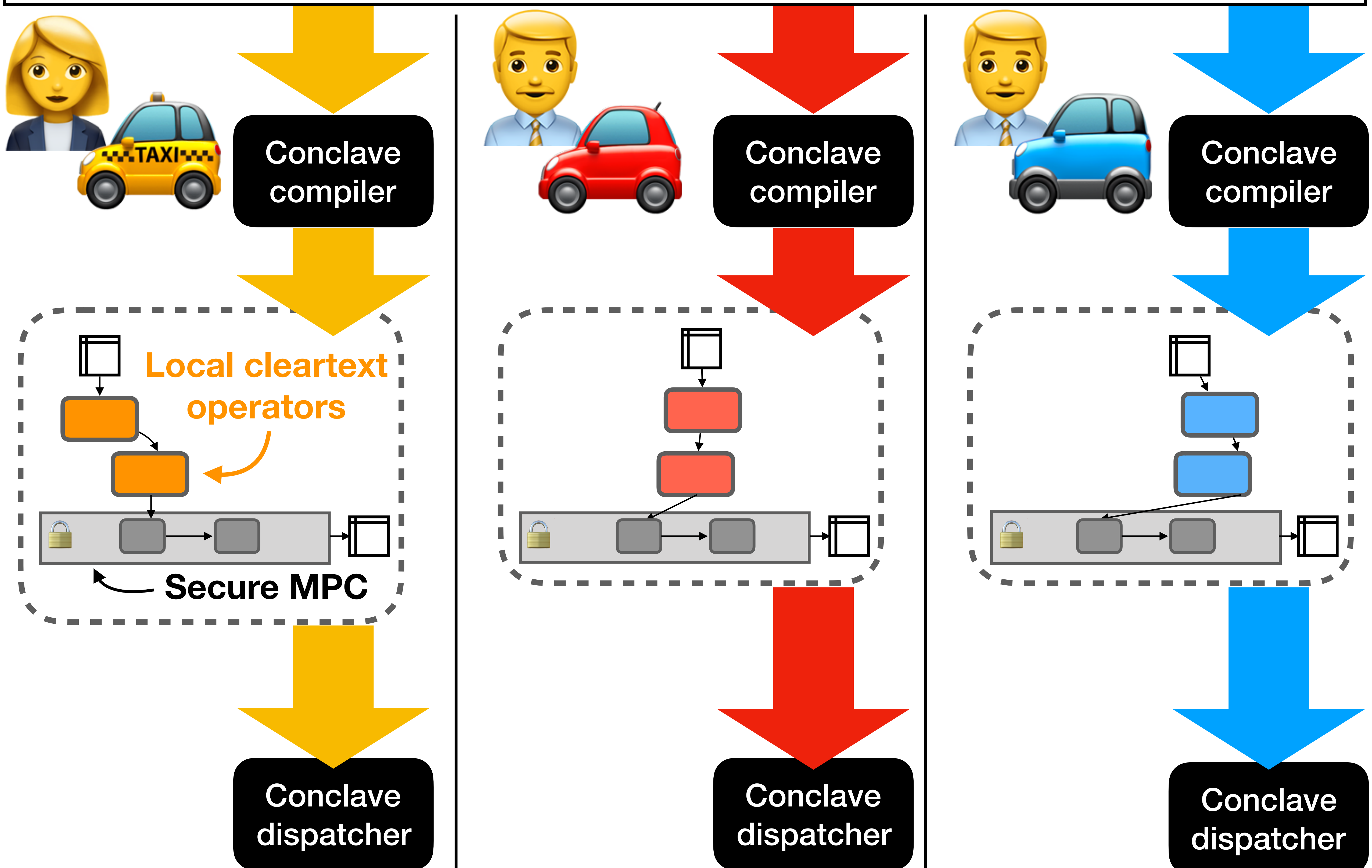
3. New **“hybrid” MPC-cleartext protocols** to accelerate expensive operators under MPC

4. **Prototype query compiler** implementation using Spark and Sharemind & performance evaluation

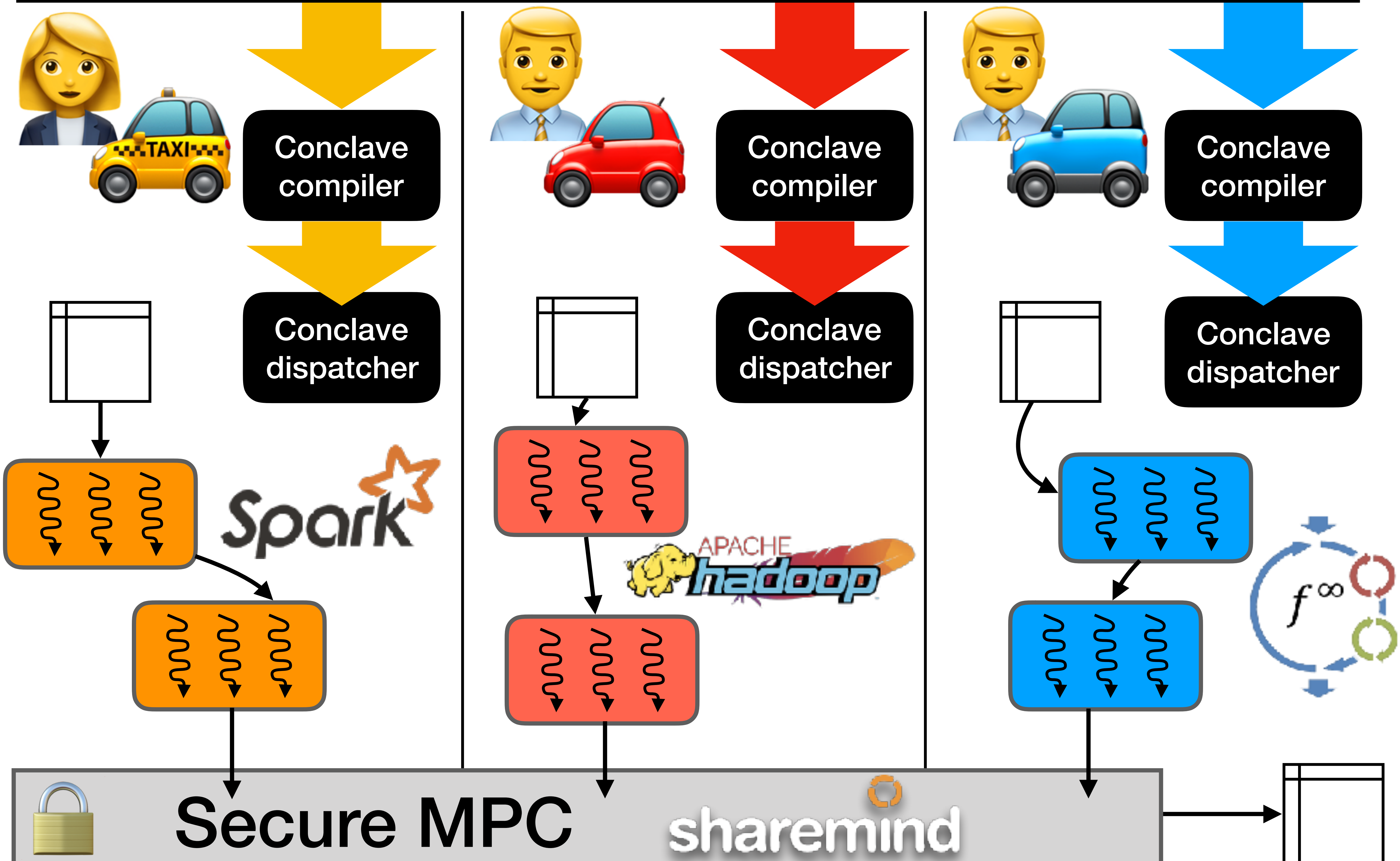
```
DECLARE TABLE trips (start_lat int, start_lon int, … );
SELECT SUM(mkt_share*mkt_share) AS hhi, SUM(trips.price) AS …
       WHERE start_lat = … AND …;
```



```
DECLARE TABLE trips (start_lat int, start_lon int, … );
SELECT SUM(mkt_share*mkt_share) AS hhi, SUM(trips.price) AS …
       WHERE start_lat = … AND …;
```



```
DECLARE TABLE trips (start_lat int, start_lon int, … );
SELECT SUM(mkt_share*mkt_share) AS hhi, SUM(trips.price) AS …
       WHERE start_lat = … AND …;
```



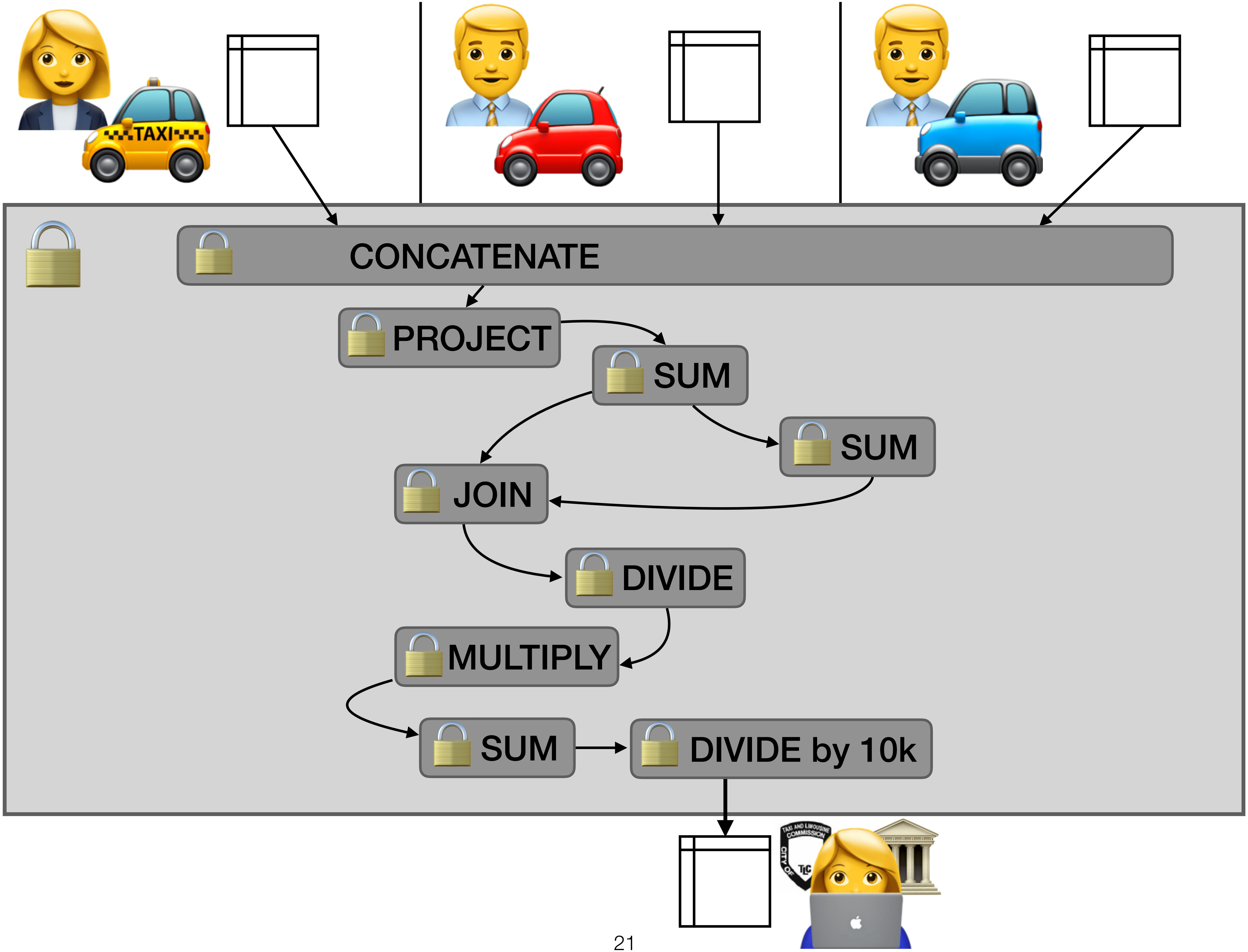
Assumptions

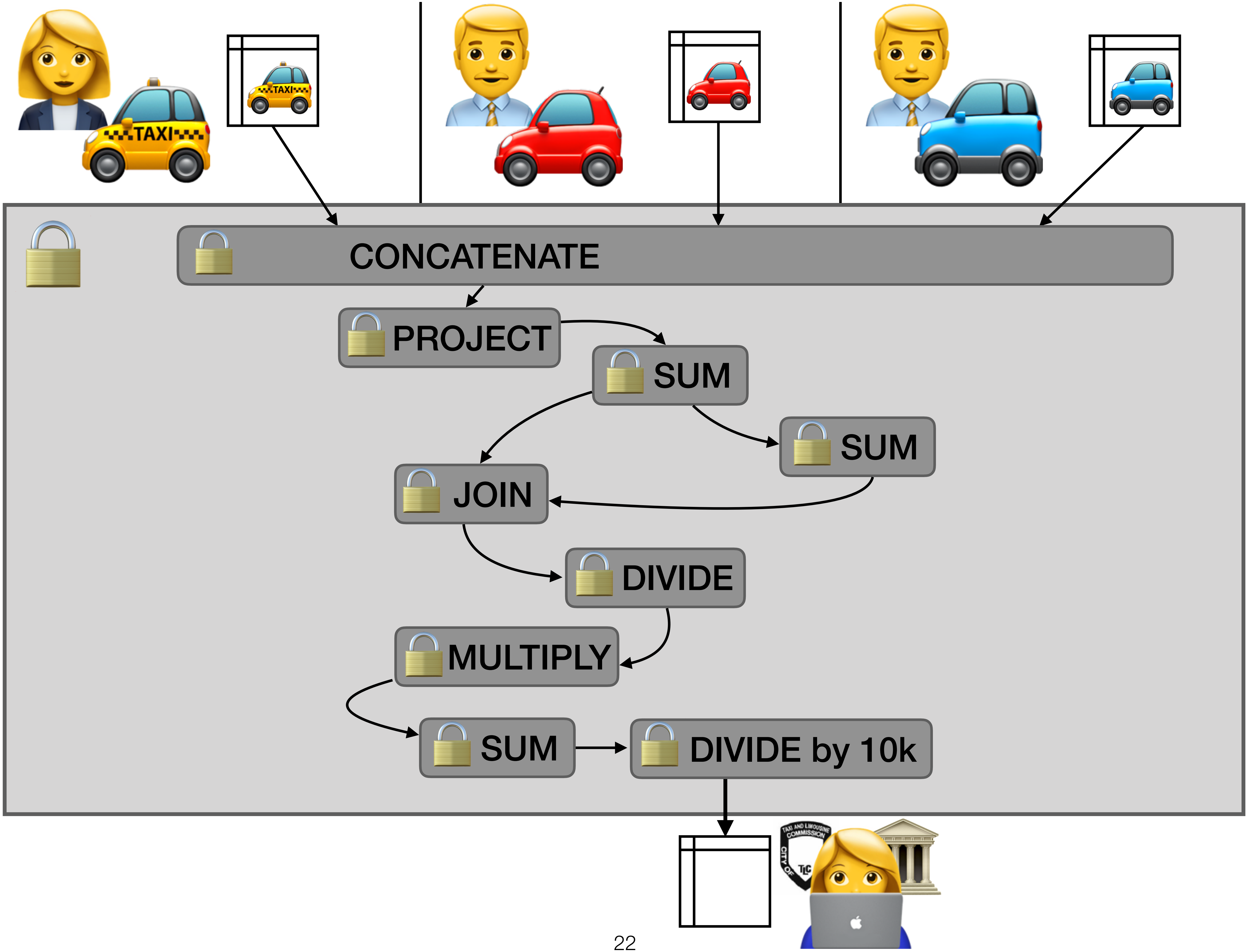
- Existing local data-parallel infrastructure (e.g., Spark cluster)
- Schemas are public or common schema agreed
- Honest-but-curious adversary
- Honest majority or anytrust model
- Okay to leak intermediate relation sizes
  - MPC backend might impose additional meta-data leakage
  - Sensitive values never never exposed to untrusted parties

LINQ-style relational query specification

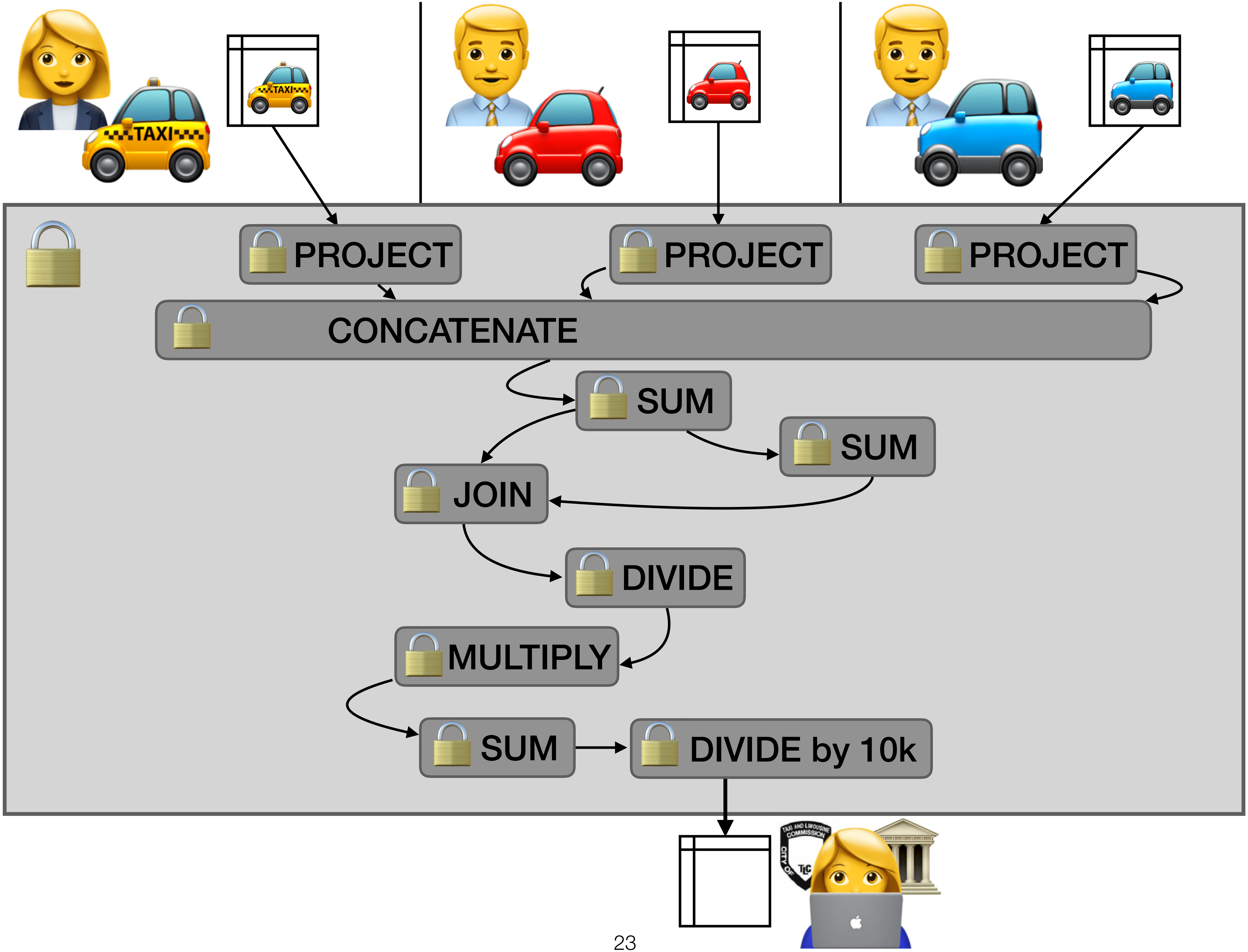
```
# compute the Herfindahl-Hirschman Index (HHI)
rev = taxi_data.project(["companyID", "price"])
          .sum("local_rev", group=[“companyID”], over="price")
          .project([0, "local_rev"])
market_size = rev.sum(“total_rev", over=“local_rev")
share = rev.join(market_size, left=[“companyID"],
                 right=[“companyID"])
           .divide("m_share", "local_rev", by="total_rev")
hhi = share.multiply(share, "ms_squared", "m_share")
           .sum(“hhi", on="ms_squared")
hhi.writeToCSV()
```

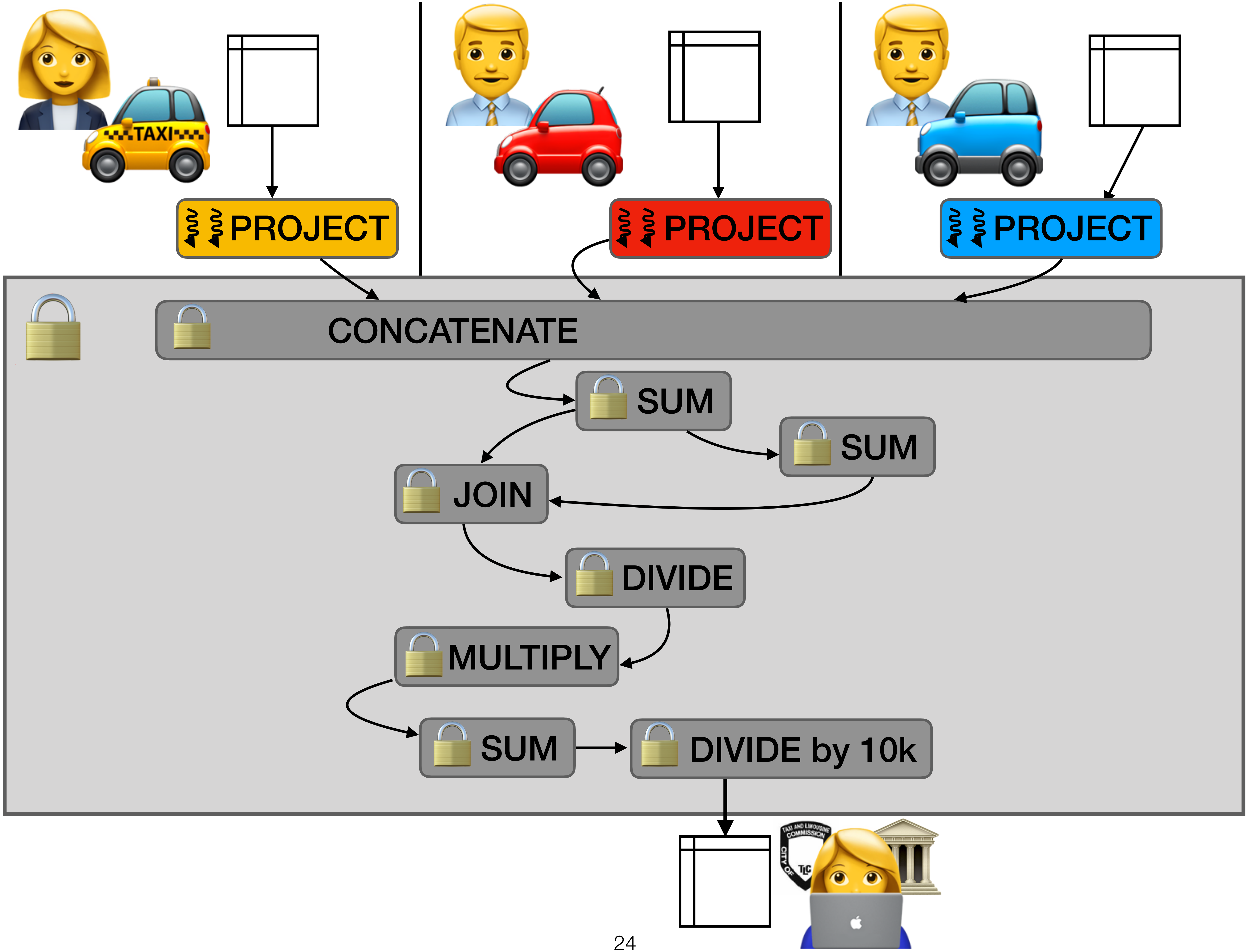
```
import conclave as cc
pA, pB, pC = cc.Party("A"), cc.Party("B"), cc.Party("C")
# 3 parties each contribute inputs with the same schema
schema = [Column("companyID", cc.INTEGER), …
          Column("price", cc.INTEGER)]
inputA = cc.defineTable(schema, owner=[pA])
inputB = cc.defineTable(schema, owner=[pB])
inputC = cc.defineTable(schema, owner=[pC])
# create multi-party input relation
taxi_data = cc.concat([inputA, inputB, inputC])

# compute the Herfindahl-Hirschman Index (HHI)
rev = taxi_data.project(["companyID", "price"])
          .sum("local_rev", group=[“companyID”], over="price")
          .project([0, "local_rev"])
market_size = rev.sum(“total_rev", over=“local_rev")
share = rev.join(market_size, left=[“companyID"],
                 right=[“companyID"])
           .divide("m_share", "local_rev", by="total_rev")
hhi = share.multiply(share, "ms_squared", "m_share")
           .sum(“hhi", on="ms_squared")
hhi.writeToCSV(owner=[pA])
```


TAXI
CONCATENATE
PROJECT
SUM
SUM
JOIN
DIVIDE
MULTIPLY
SUM
DIVIDE by 10k
TAXI AND LIMOUSINE COMMISSION
CITY OF
TLC

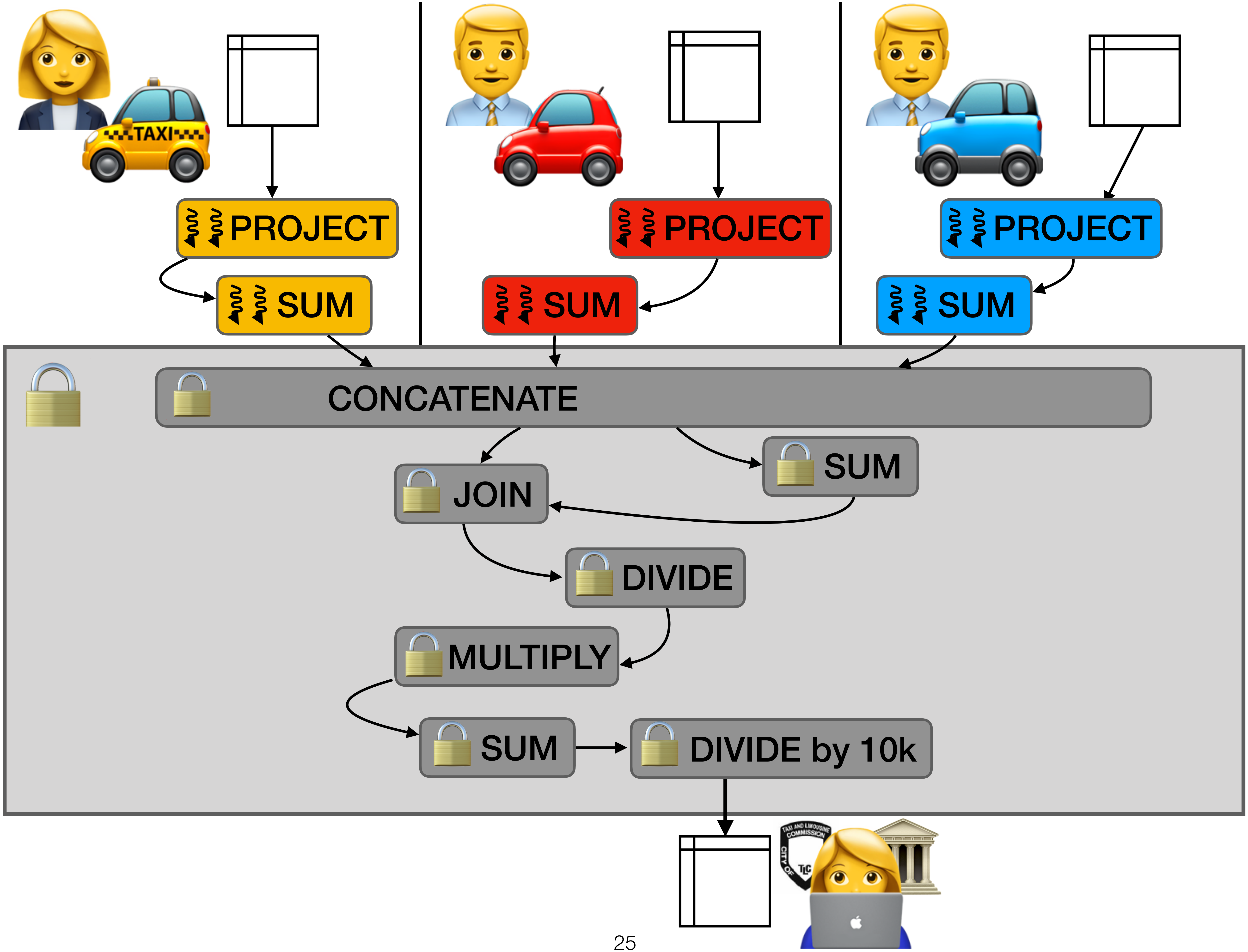


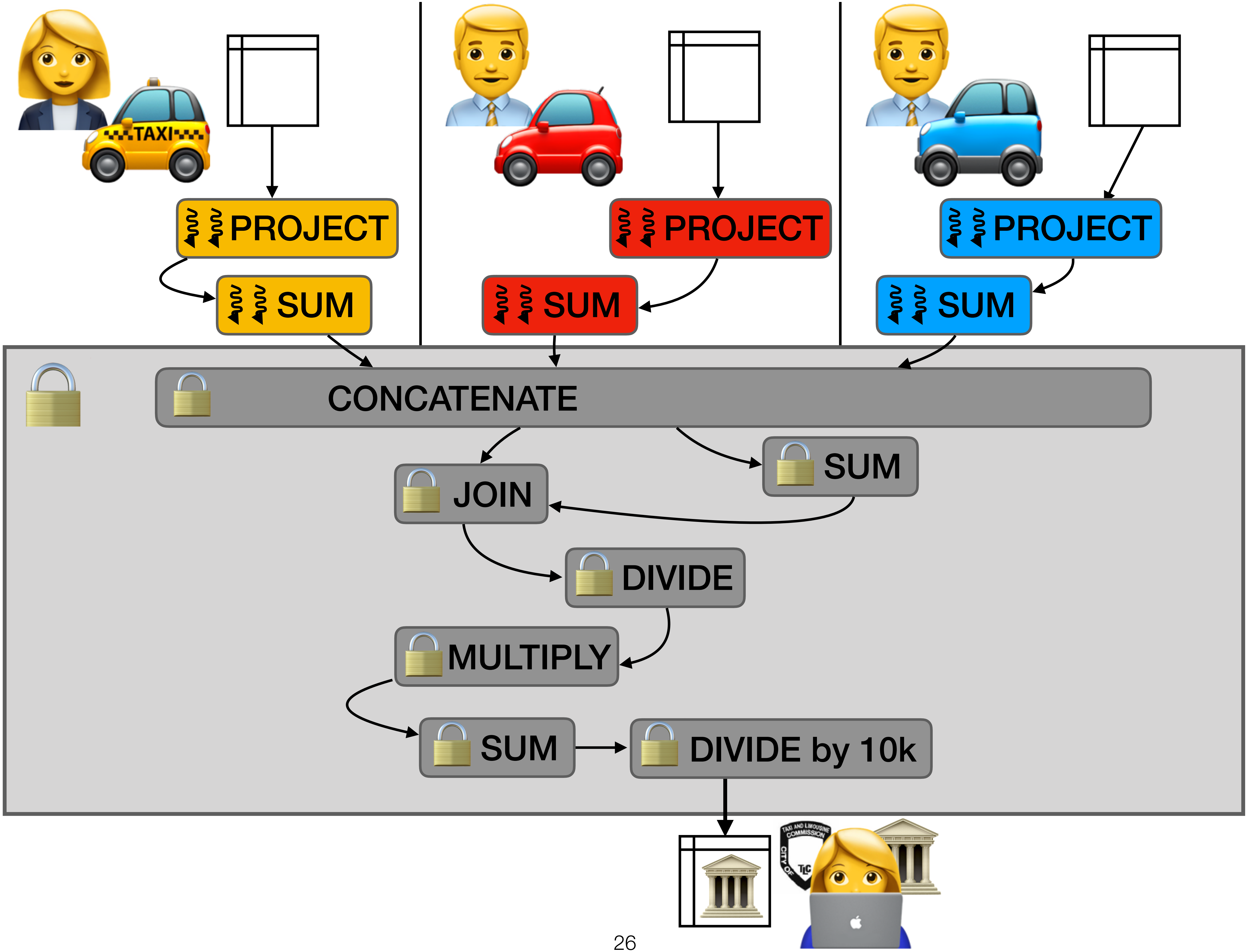
TAXI
CONCATENATE
PROJECT
SUM
SUM
JOIN
DIVIDE
MULTIPLY
SUM
DIVIDE by 10k

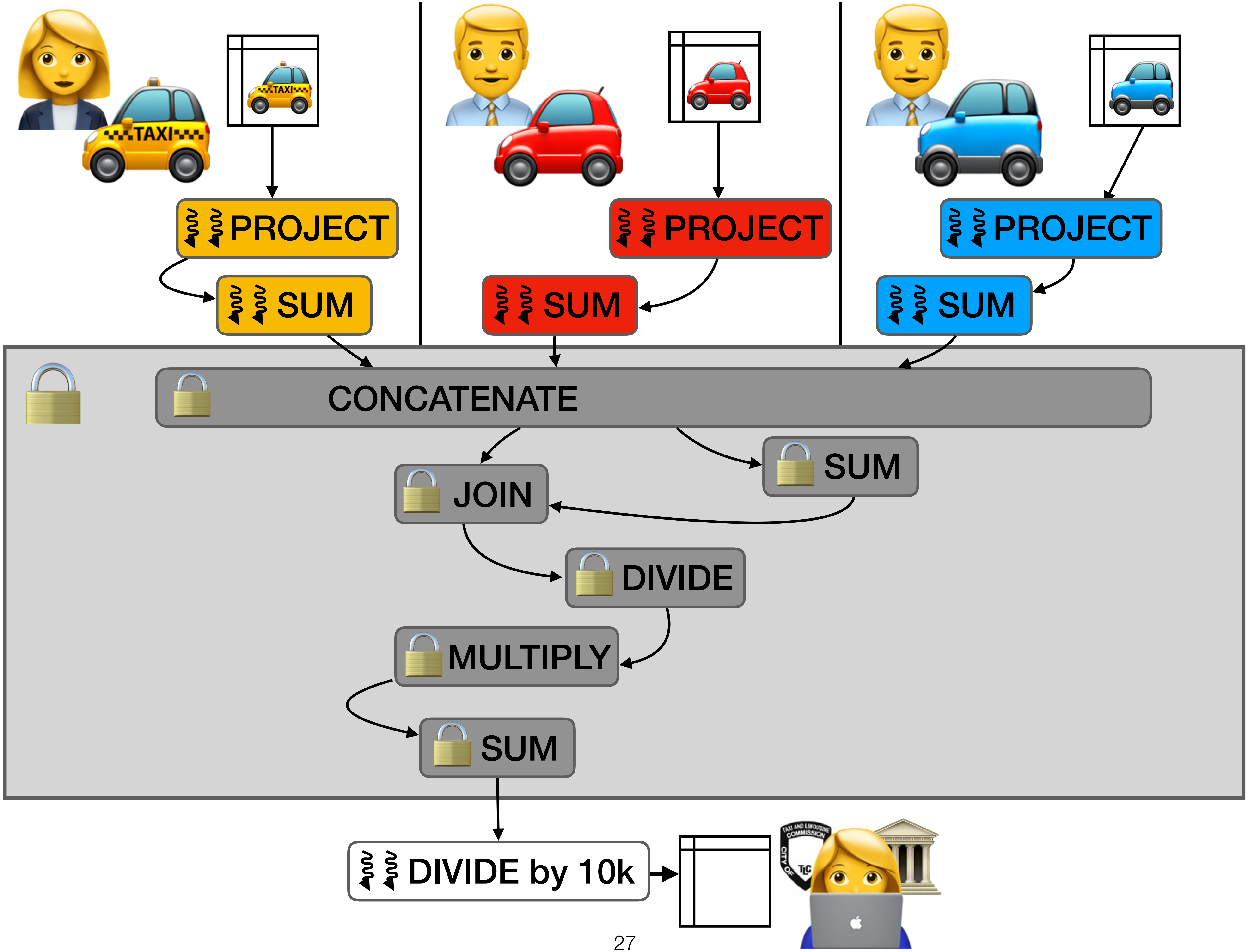

TAXI
PROJECT
PROJECT
PROJECT
CONCATENATE
SUM
SUM
JOIN
DIVIDE
MULTIPLY
SUM
DIVIDE by 10k
TAXI AND LIMOUSINE COMMISSION
CITY OF
TLC



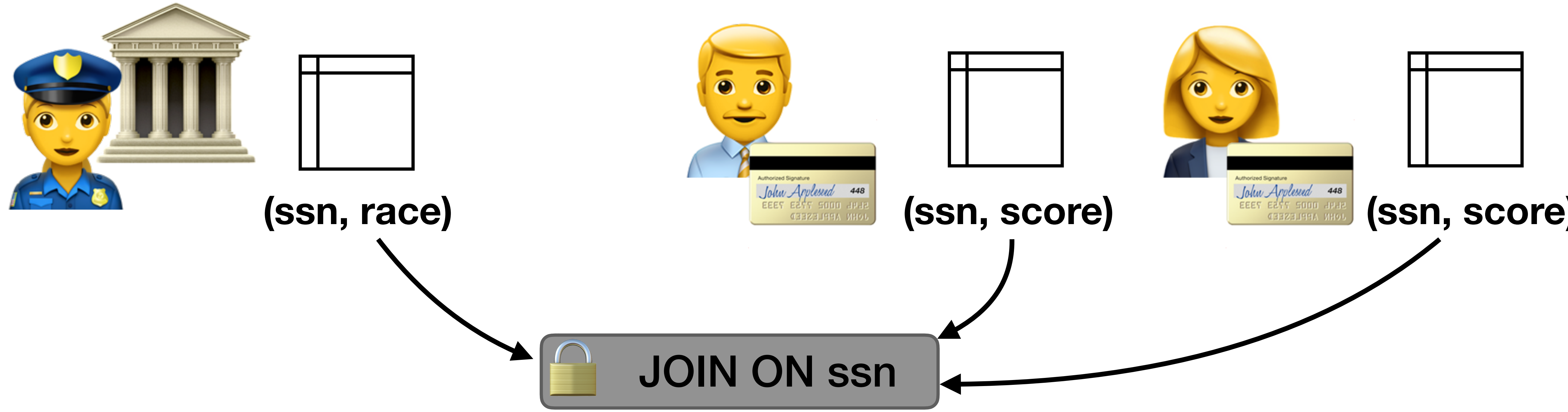
TAXI
PROJECT
PROJECT
PROJECT
CONCATENATE
SUM
SUM
JOIN
DIVIDE
MULTIPLY
SUM
DIVIDE by 10k


TAXI
PROJECT
SUM
PROJECT
SUM
PROJECT
SUM
CONCATENATE
SUM
JOIN
DIVIDE
MULTIPLY
SUM
DIVIDE by 10k


TAXI
PROJECT
SUM
PROJECT
SUM
PROJECT
SUM
CONCATENATE
SUM
JOIN
DIVIDE
MULTIPLY
SUM
DIVIDE by 10k

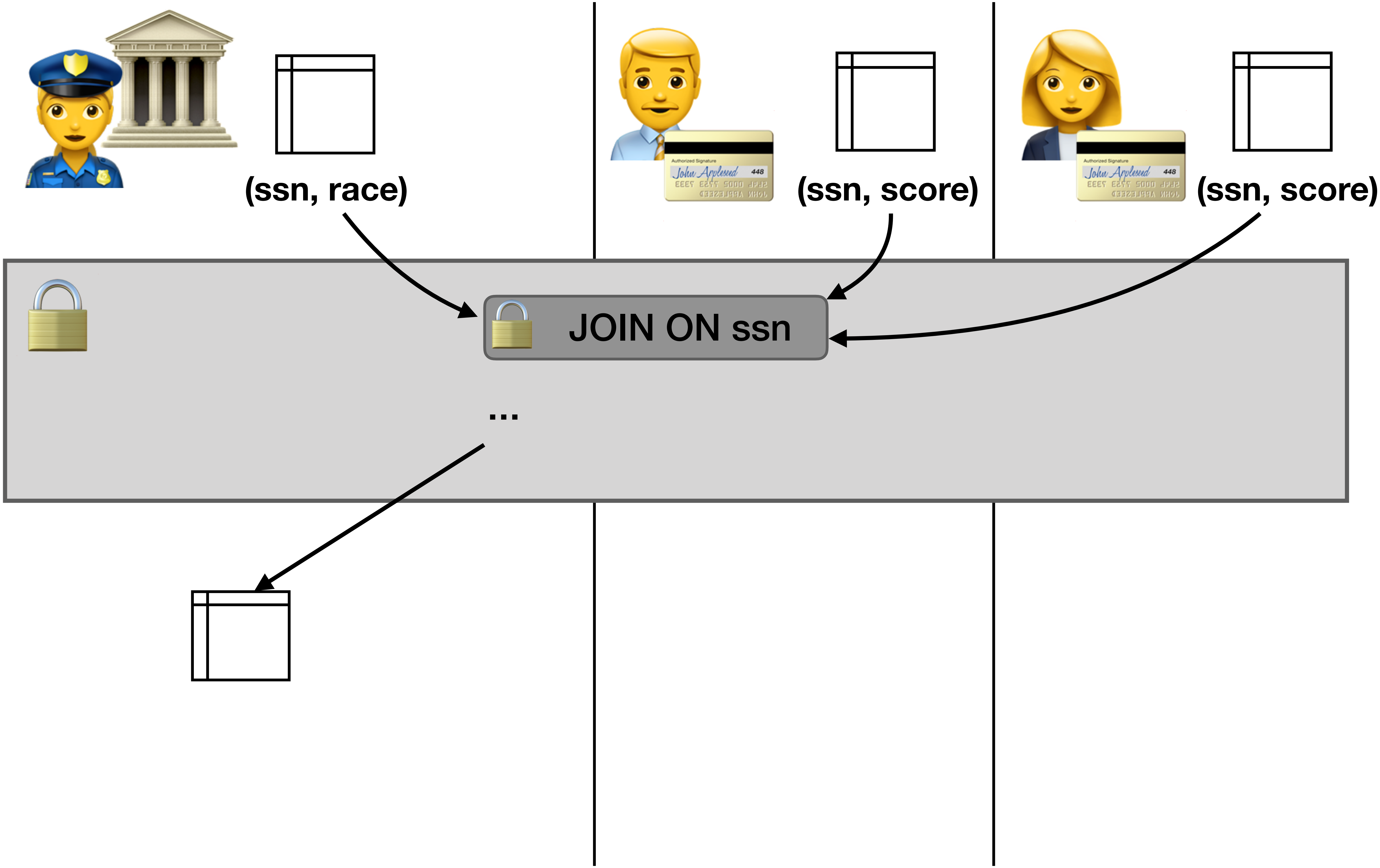


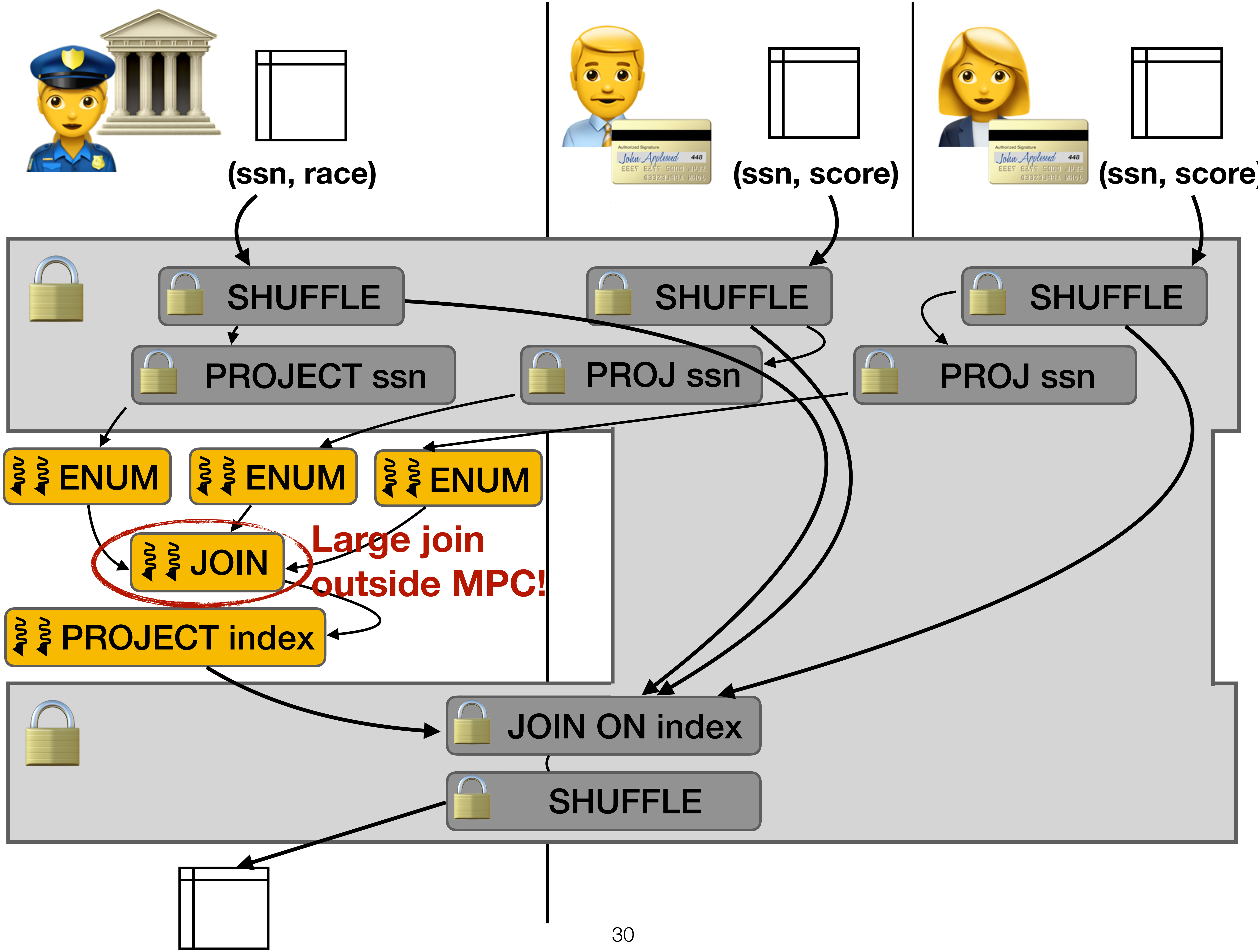
TAXI
PROJECT
SUM
PROJECT
SUM
PROJECT
SUM
CONCATENATE
SUM
JOIN
DIVIDE
MULTIPLY
SUM
DIVIDE by 10k
TAXI AND LIMOUSINE COMMISSION
CITY OF
TLC




```
import conclave as cc

pA, pB, pC = cc.Party(“regulator"), cc.Party(“bankA"), \
             cc.Party(“bankB")
demo_schema = [Column("ssn", cc.INTEGER),
               Column(“race", cc.INTEGER)]
demographics = cc.defineTable(demo_schema, owner=[pA])

# credit card companies trust the regulator to compute on SSNs
bank_schema = [Column("ssn", cc.INTEGER, trust=[pA]),
               Column("score", cc.INTEGER)]
scores1 = cc.defineTable(bank_schema, owner=[pB])
…
```

Regulator has SSNs anyway


(ssn, race)
(ssn, score)
(ssn, score)
JOIN ON ssn
...


(ssn, race)
(ssn, score)
(ssn, score)
SHUFFLE
SHUFFLE
SHUFFLE
PROJECT ssn
PROJ ssn
PROJ ssn
ENUM
ENUM
ENUM
JOIN
Large join
outside MPC!
PROJECT index
JOIN ON index
SHUFFLE

Implementation

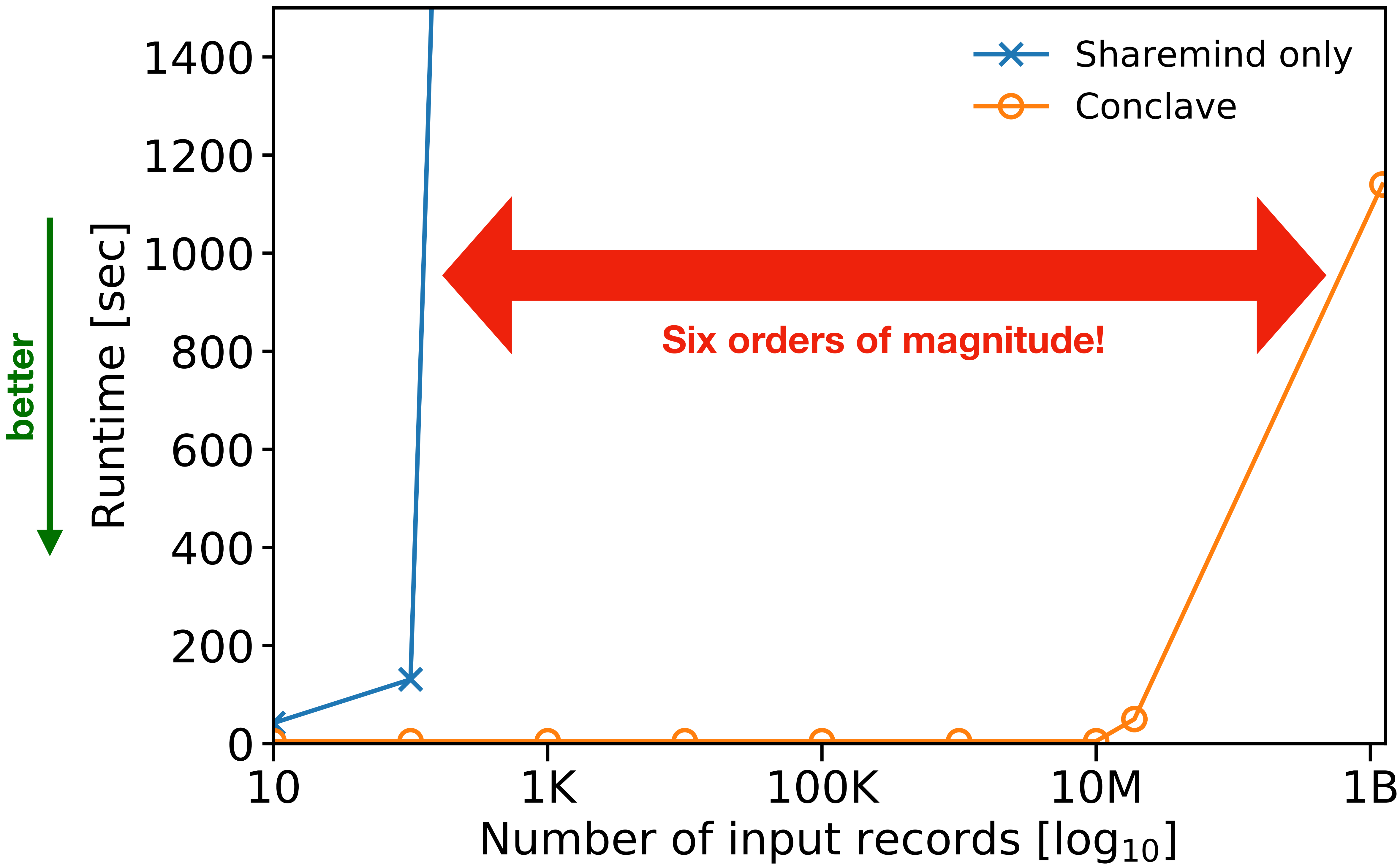
- LINQ-style relational front-end
- Rewrite rules on intermediate DAG of operators
- Back-ends generate code
  - Cleartext: Spark, sequential Python
  - MPC: Sharemind
- ~5,000 lines of Python

Evaluation

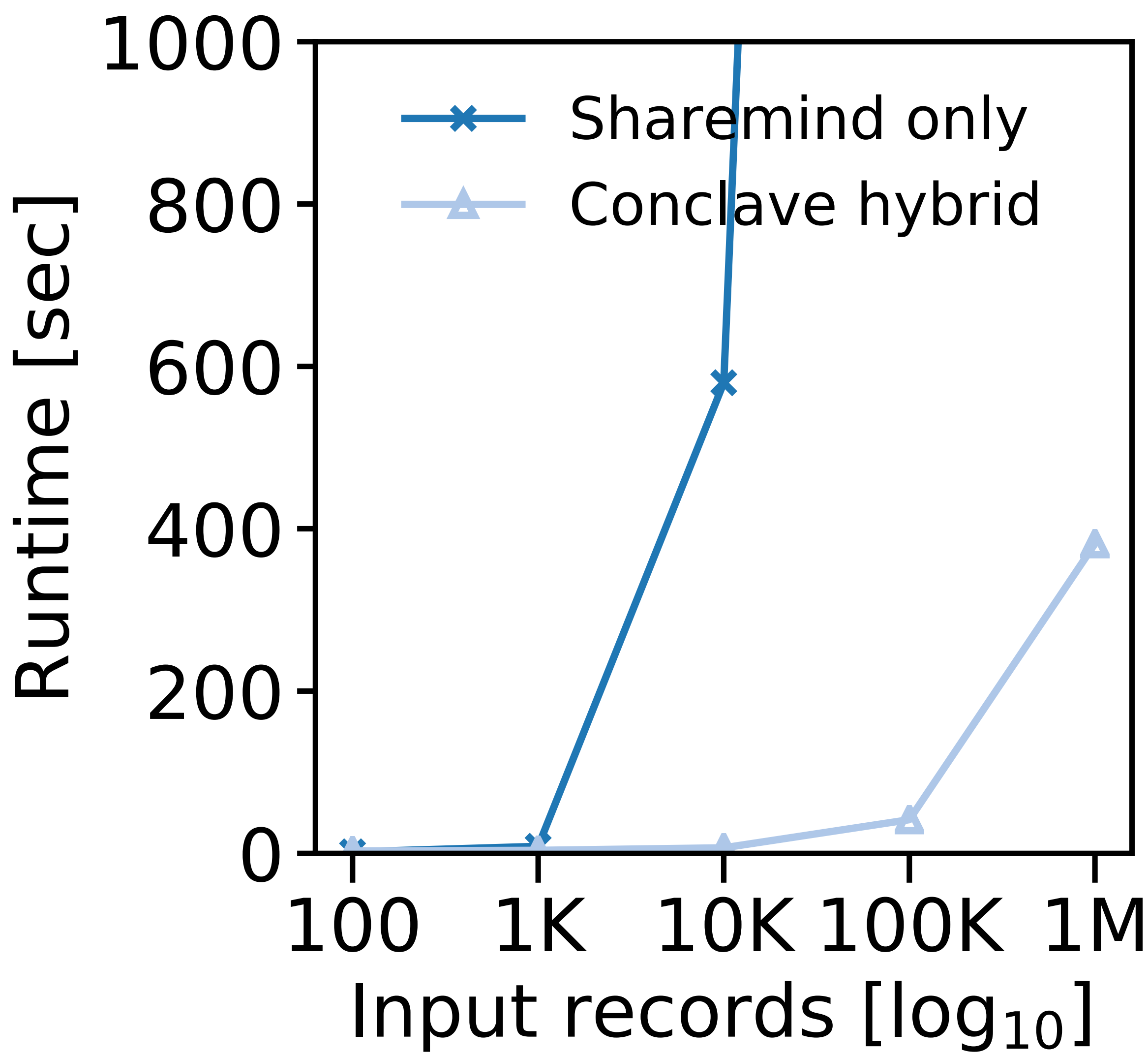
1. How does Conclave scale to increasingly large inputs?
2. How much does automatic MPC frontier placement reduce query runtime?
3. What impact do hybrid MPC-cleartext operators have on query runtime?

- **Three parties**
  3 VM Spark cluster + Sharemind endpoint at each
- **Two queries**
  1. Taxi market concentration: up to 1.3B trip records
  2. Credit card regulation: up to 100k SSNs

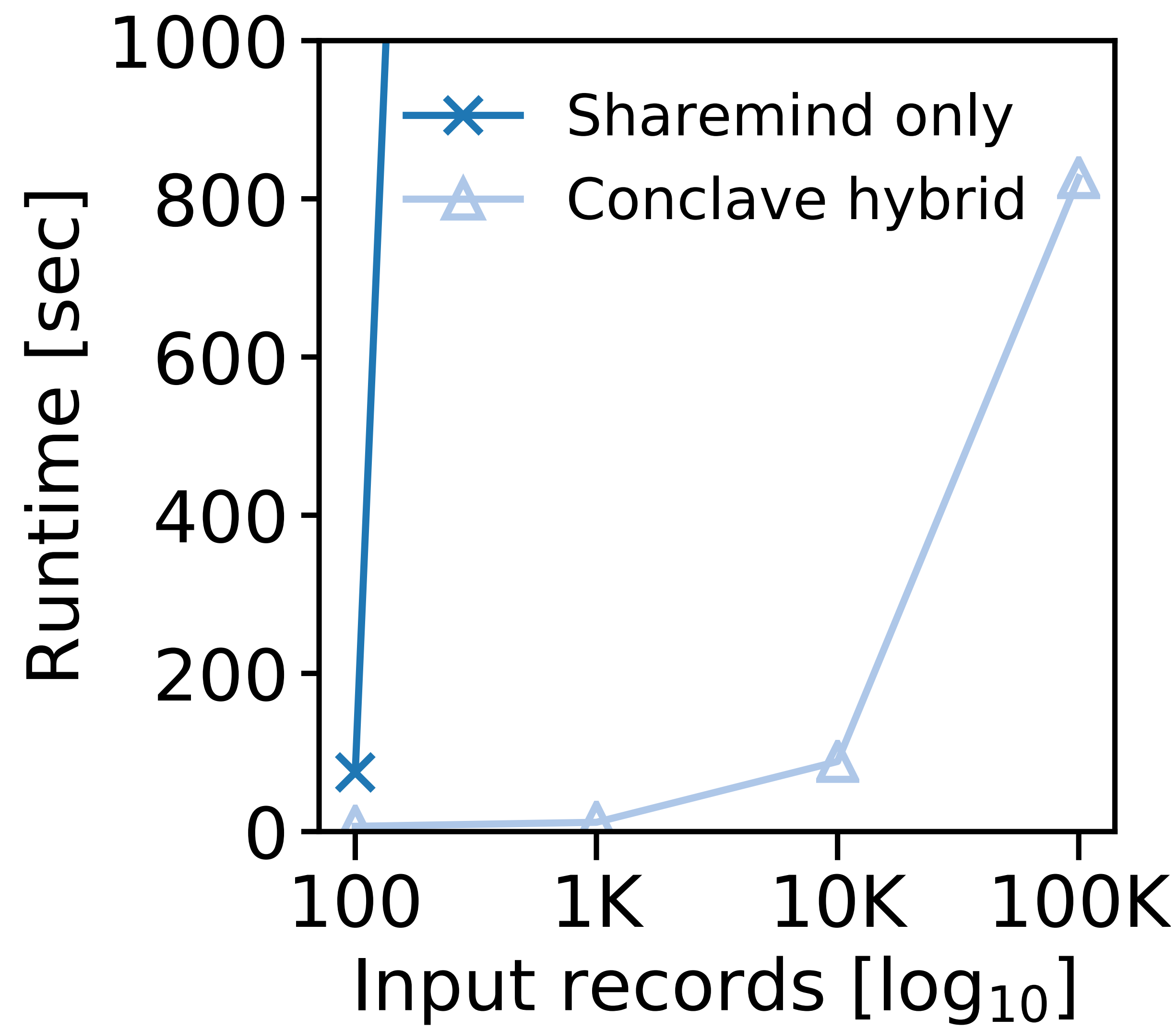
Taxi market concentration query



Hybrid MPC-cleartext operator impact

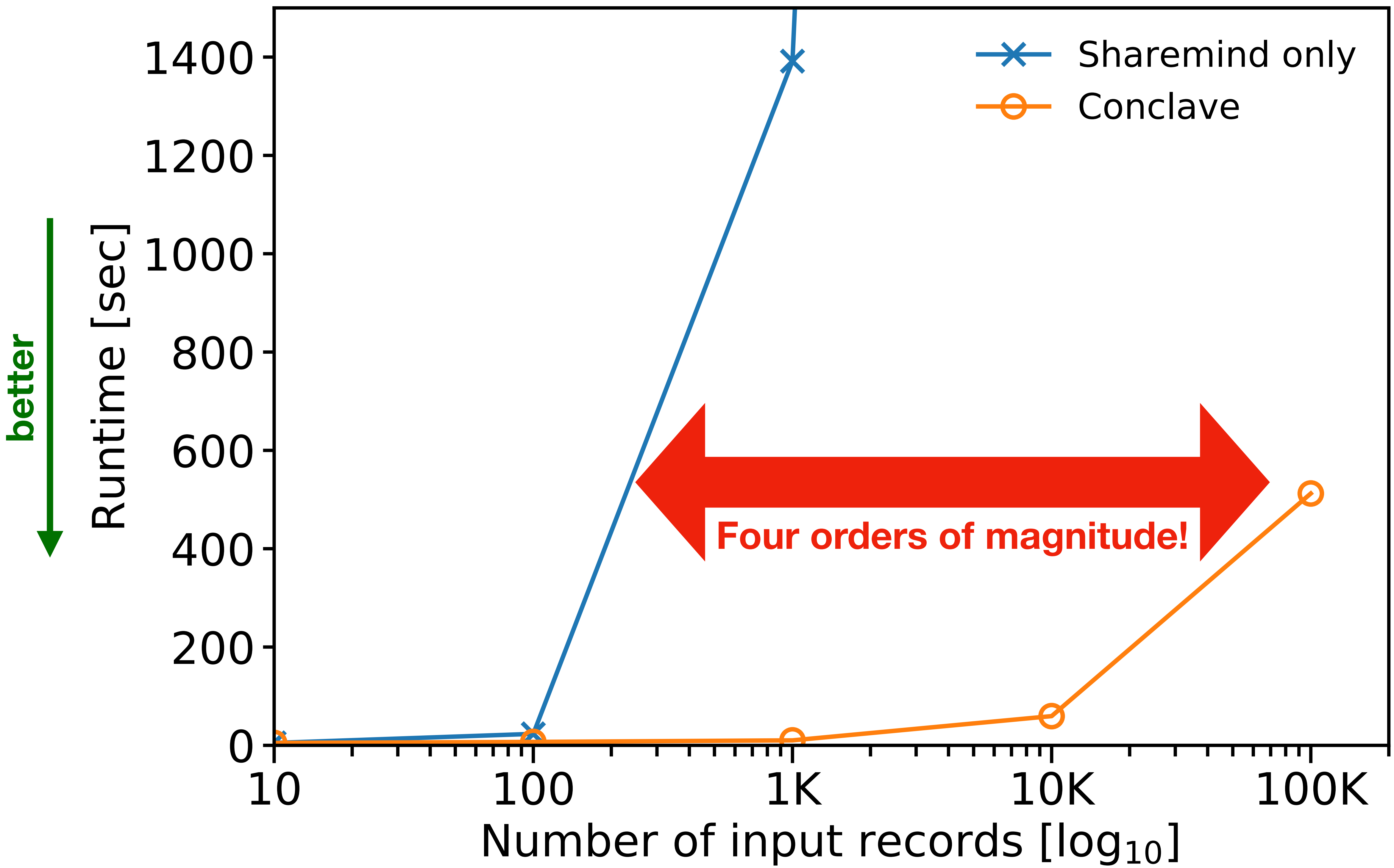


Join



Aggregation

Credit card regulation query



Related work

- Mixed-mode MPC: **Wysteria** [S&P 2014] — custom DSL
- Query rewriting for MPC
  - **SMCQL** [VLDB 2017]: binary public/private columns, no hybrid operators
  - **Opaque** [NSDI 2017]: computation under SGX, focus on reducing oblivious shuffles

Summary

- Conclave is a query compiler for efficient MPC on “big data”
- Computes as much as possible locally in the clear
- Automatically shrinks MPC step to be as small as possible
- New hybrid MPC-cleartext protocols speed up operators
- Scales up to 7 orders of magnitude better than pure MPC

`https://github.com/multiparty/conclave`

Ask me for our draft paper if you’re interested! :)