Image: National Geographic

Conclave

Secure Multi-Party Computation on Big Data

Scalable Oblivious Data Analytics

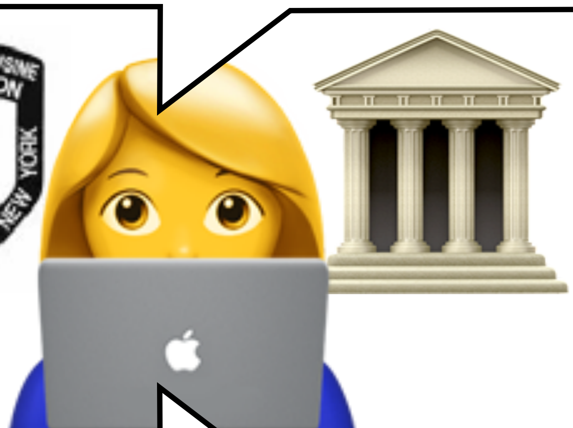
Nikolaj Volgushev Malte Schwarzkopf Ben Getchell
Andrei Lapets Mayank Varia Azer Bestavros

Pre-presentation disclaimers/ confessions

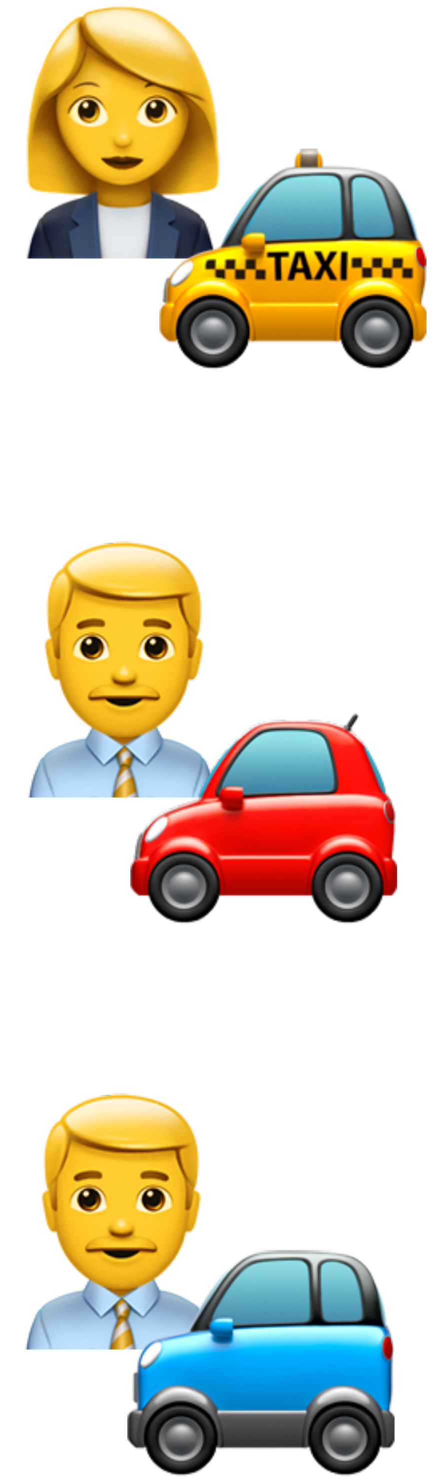
- Less focus on developing new MPC protocols
- Instead focus on integrating MPC into big data analytics domain where the current roadblocks are:
  - Accessibility (data analysts are not MPC experts)
  - Scalability (large data volumes)
- Semi-honest adversaries throughout! 😈

Is there healthy competition?

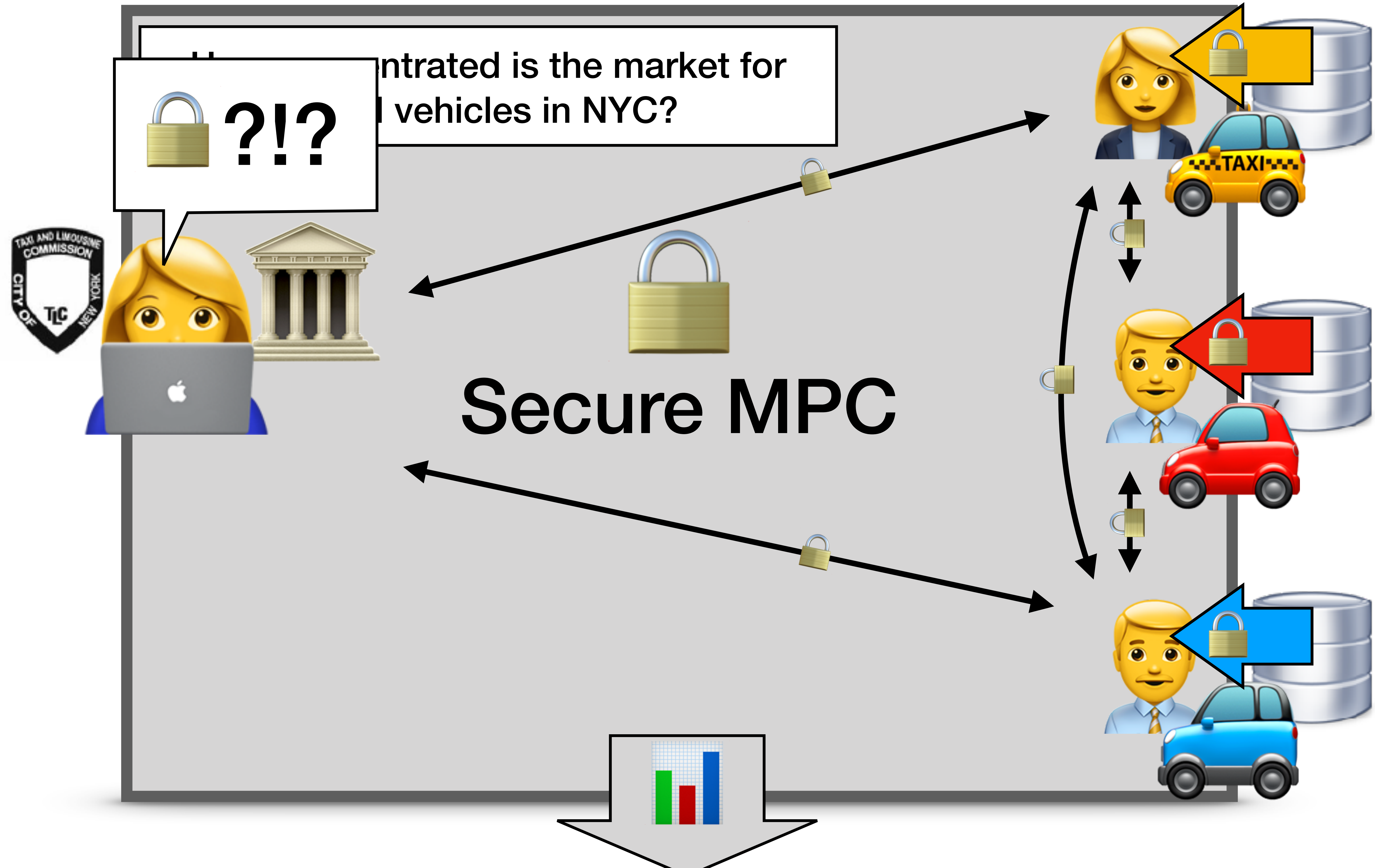
A dangerous monopoly forming?

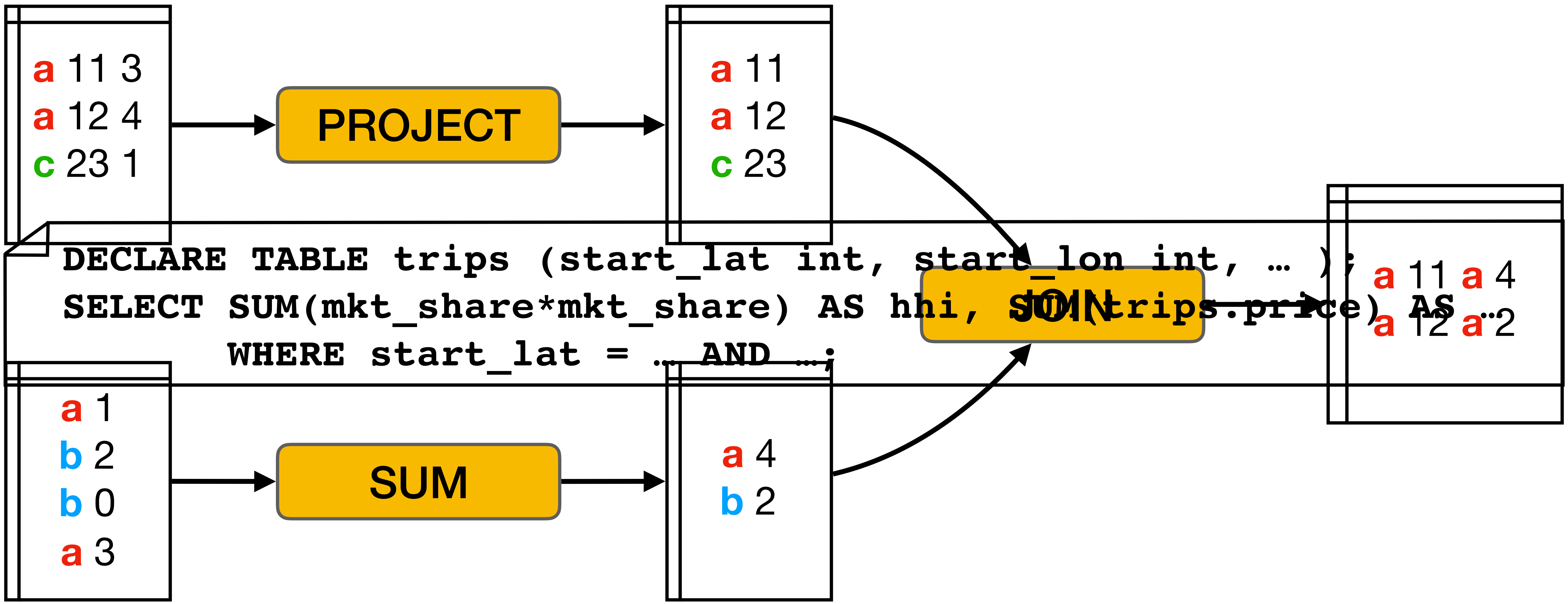


How concentrated is the market for hired vehicles in NYC?



A solution: Secure MPC

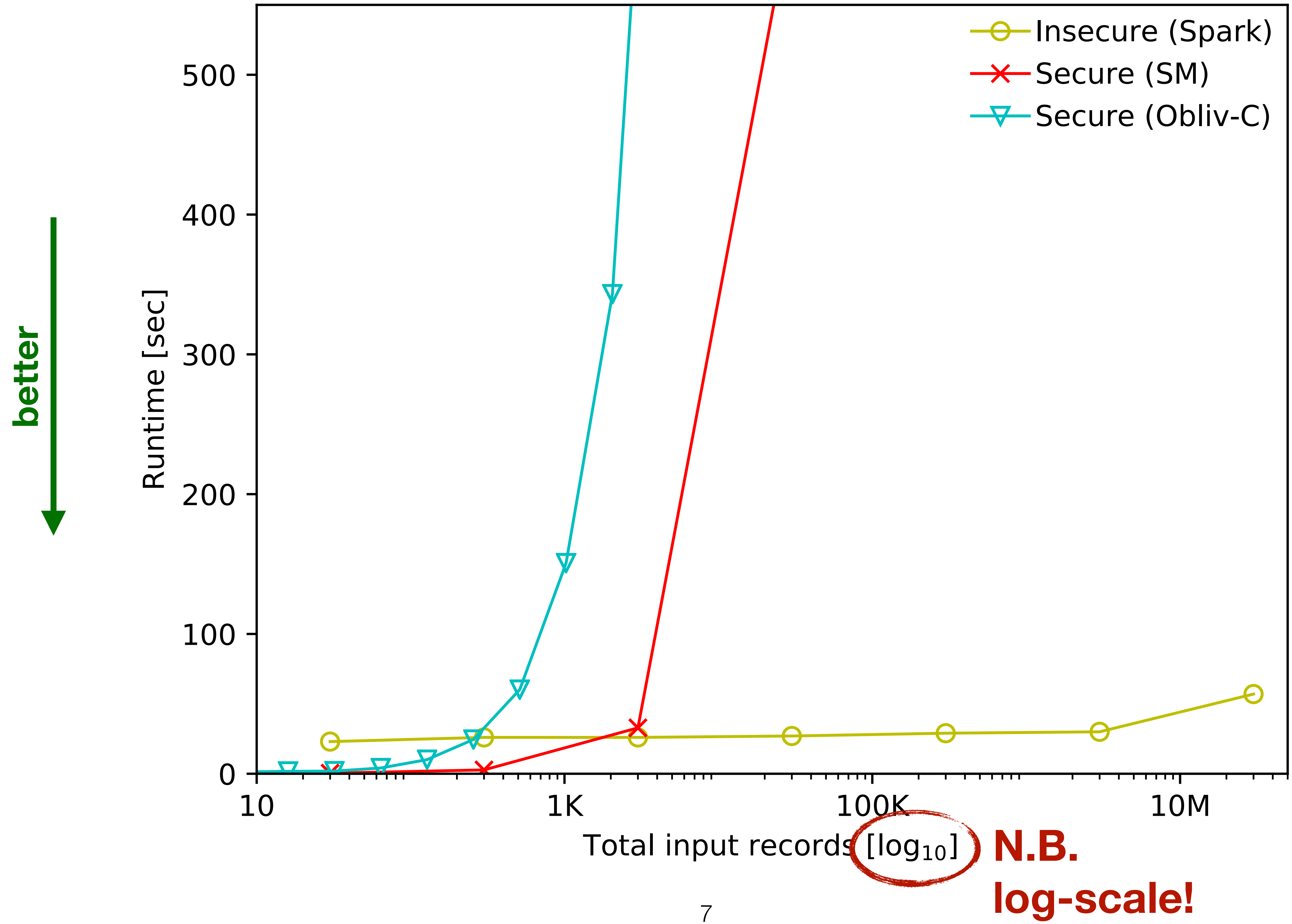




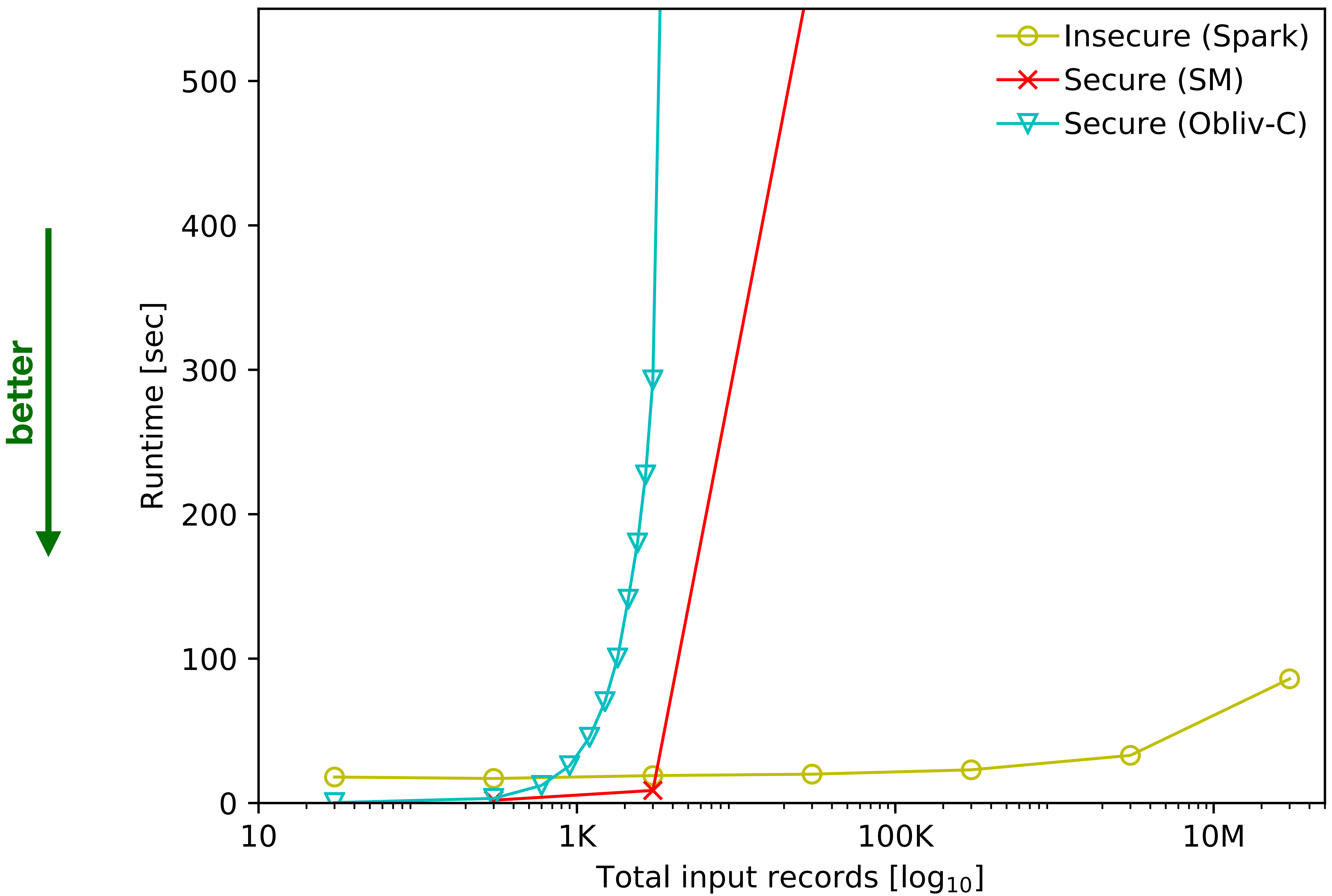

```
DECLARE TABLE trips (start_lat int, start_lon int, … );
SELECT SUM(mkt_share*mkt_share) AS hhi, SUM(trips.price) AS …
       WHERE start_lat = … AND …;
```


Secure MPC
175M annual trips!
TAXI

Does MPC scale? — Agg: SUM



Does MPC scale? — Join



Our `TODO` list

- Make MPC more *accessible* to data analysts
- Make MPC *scale* for common analytics queries

Conclave

Key insight:

For most queries, only some of the work **must** happen under MPC.

Automatically rewrite query to combine scalable local computation and MPC.

Contributions

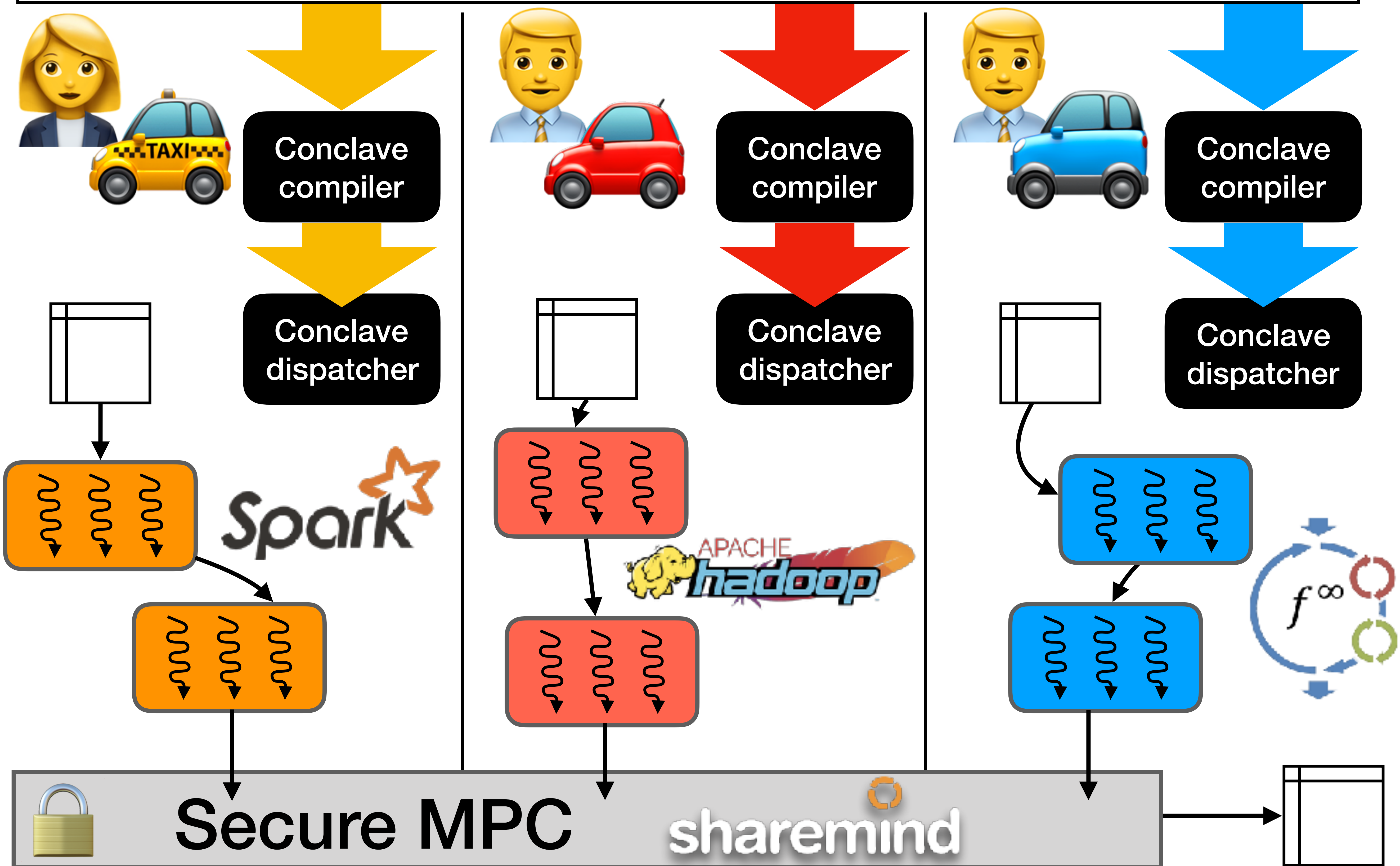
1. **MPC query compilation:** relational query compiler that optimizes for efficient MPC.

2. **Automated analysis** to minimize MPC use while maintaining required guarantees.

3. **Hybrid operators**: new MPC protocols that give the option to relax privacy requirements to further 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 …;
```



Relational query specification

Parties' data treated as single relation

```
# 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()
```

Relational query specification

```
import conclave as cc
pA, pB, pC = cc.Party(mpc.a.com), […], cc.Party(mpc.c.org)
schema = [Column("companyID", cc.INTEGER), …
          Column("price", cc.INTEGER)]
# 3 parties each contribute inputs with the same schema
taxi_data = cc.defineTable(schema, at=[pA, pB, pC])

# 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(to=[pA])
```

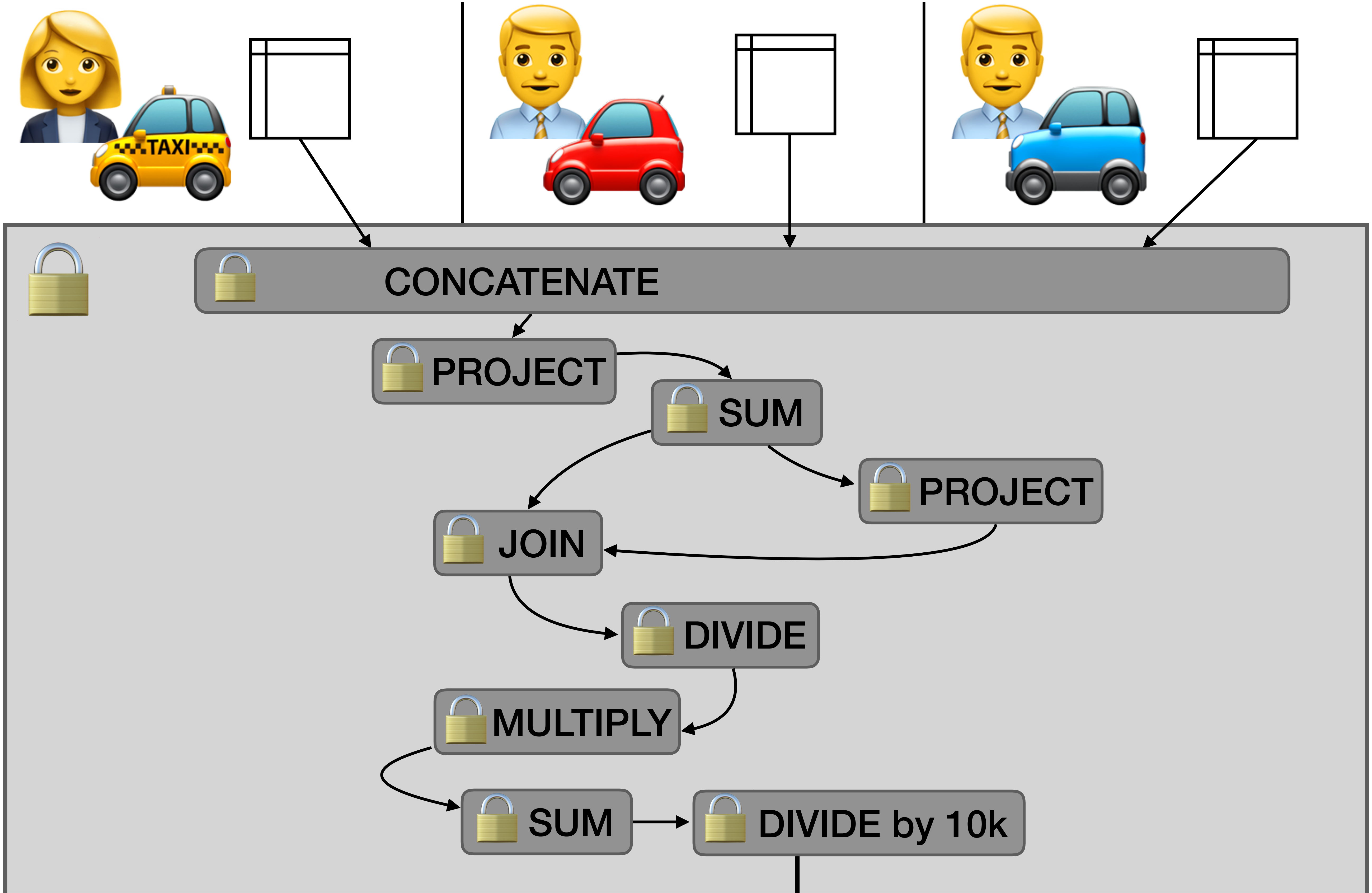
Contributions

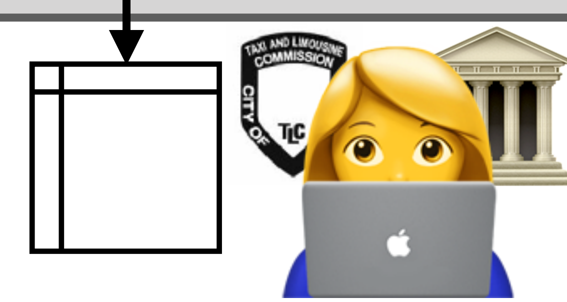
1. **MPC query compilation:** relational query compiler that optimizes for efficient MPC.

2. **Automated analysis** to minimize MPC use while maintaining required guarantees.

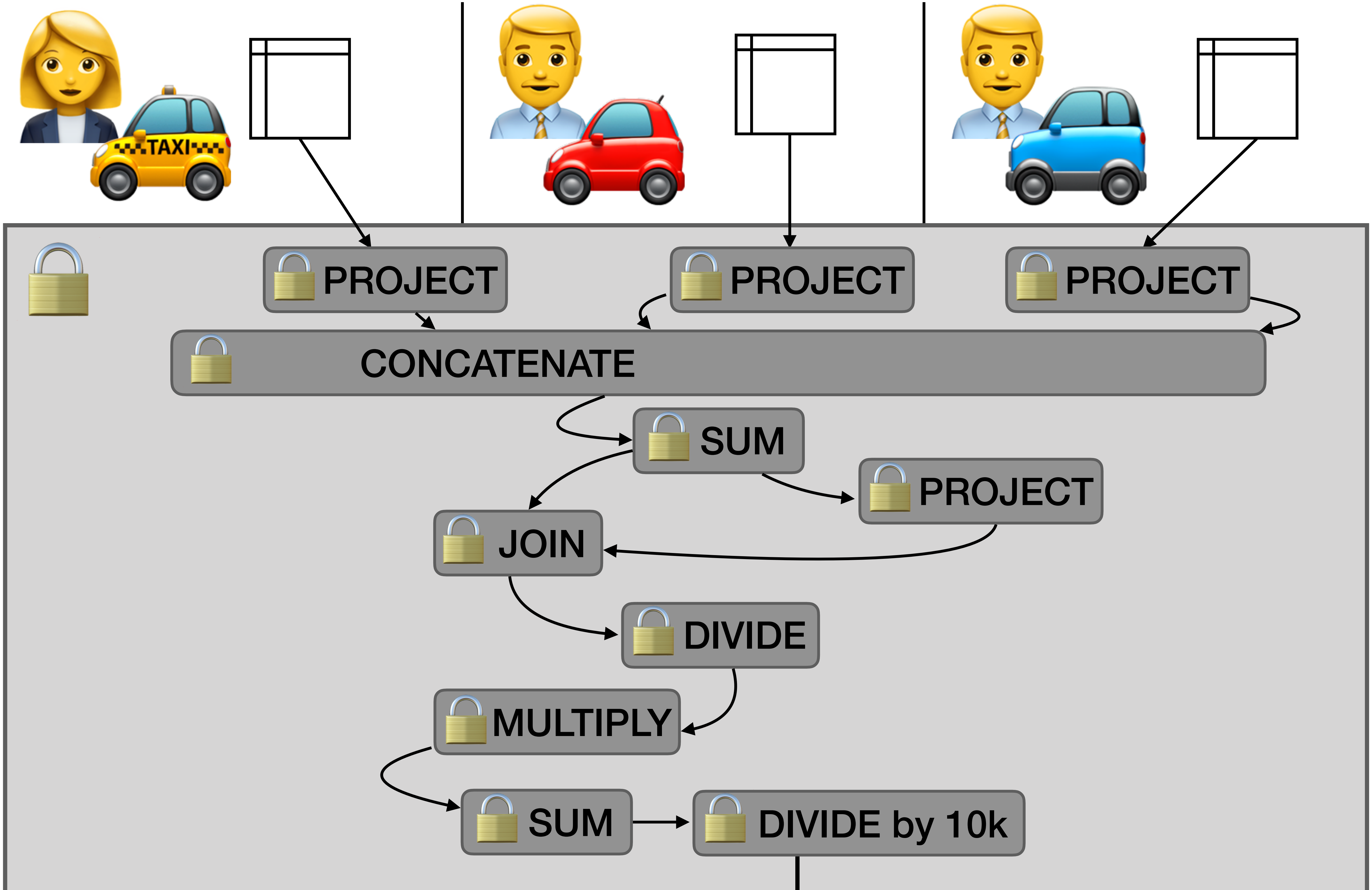
3. **Hybrid operators**: new MPC protocols that give the option to relax privacy requirements to further accelerate expensive operators under MPC.

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

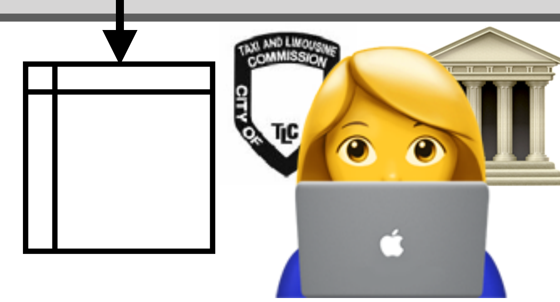


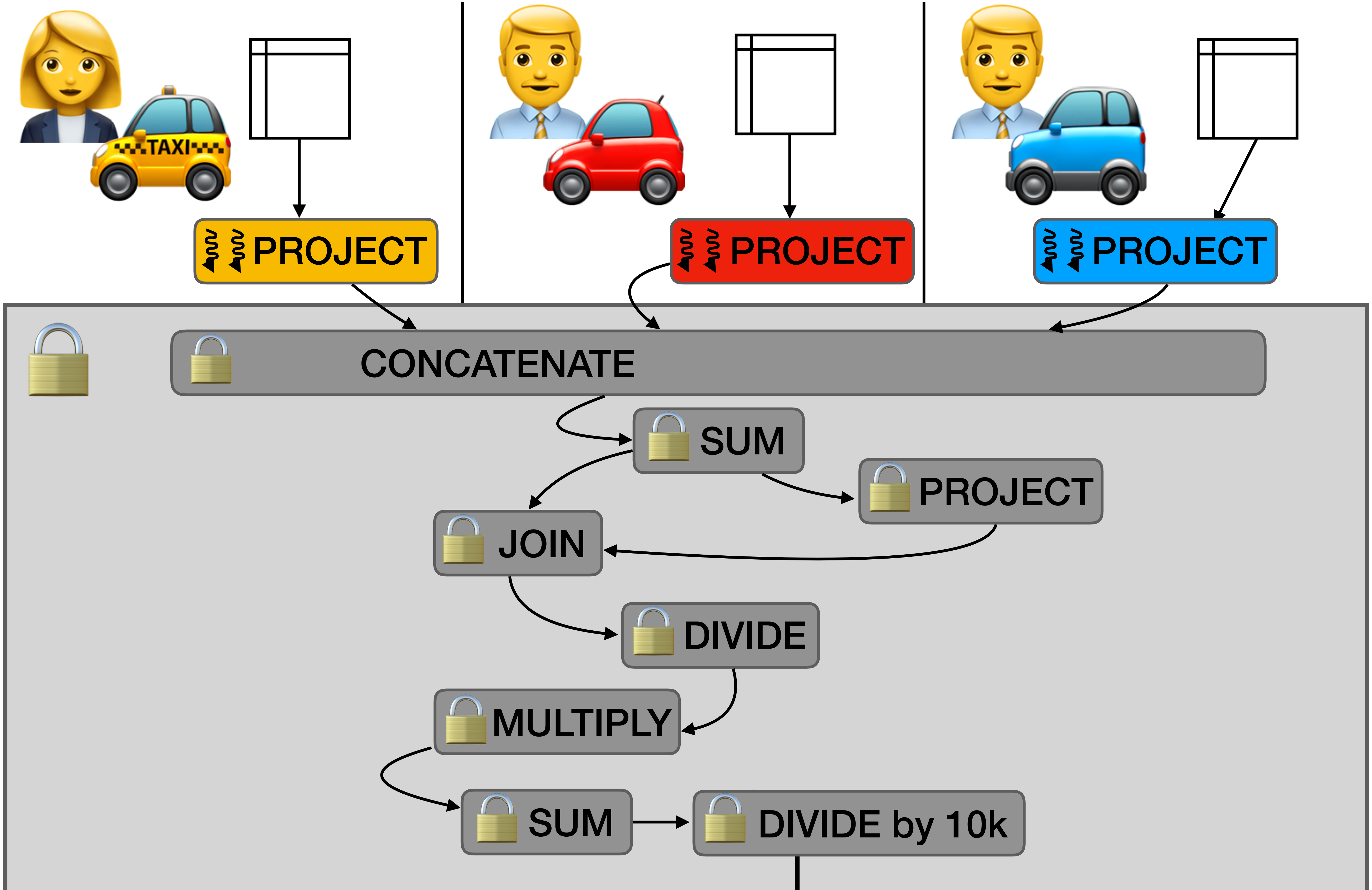


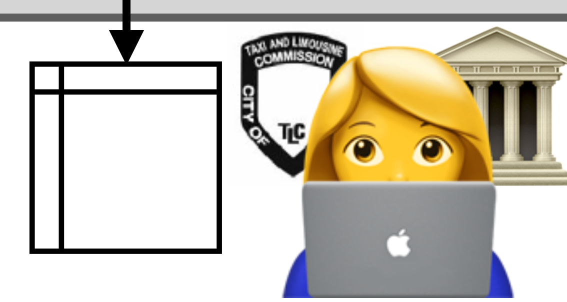
```
proj(concat(a, b)) =
concat(proj(a), proj(b))
```



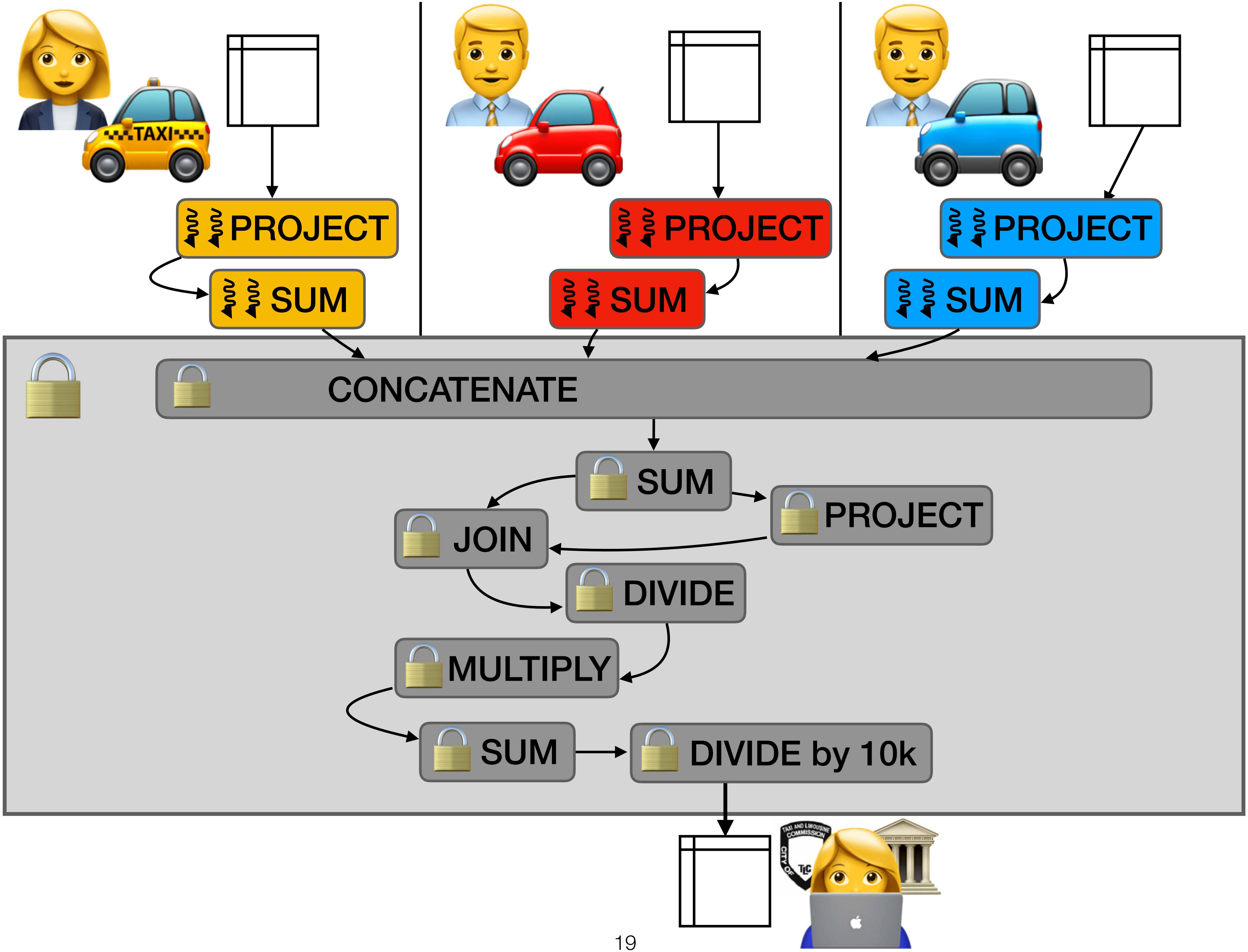

```
proj(concat(a, b)) =
concat(proj(a), proj(b))
```







```
sum(concat(a, b)) =
sum(concat(sum(a), sum(b)))
```



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

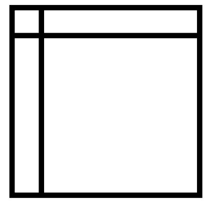
Contributions

1. **MPC query compilation:** relational query compiler that optimizes for efficient MPC.

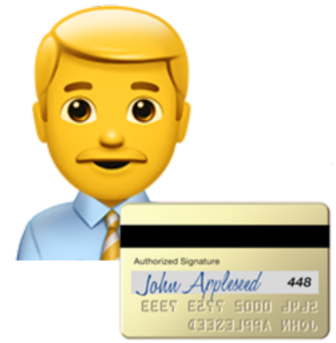
2. **Automated analyses** to determine which parts of a query must run under MPC.

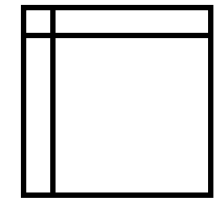
3. **Hybrid operators**: new MPC protocols that give the option to relax privacy requirements to further accelerate expensive operators under MPC.

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

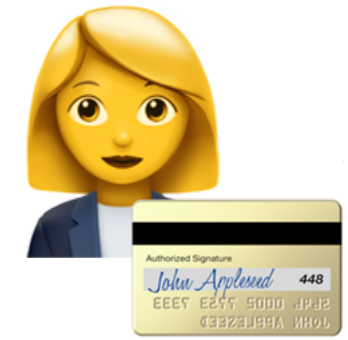


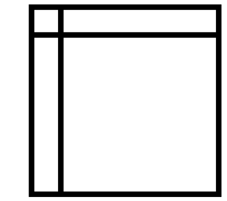
(ssn, zip)



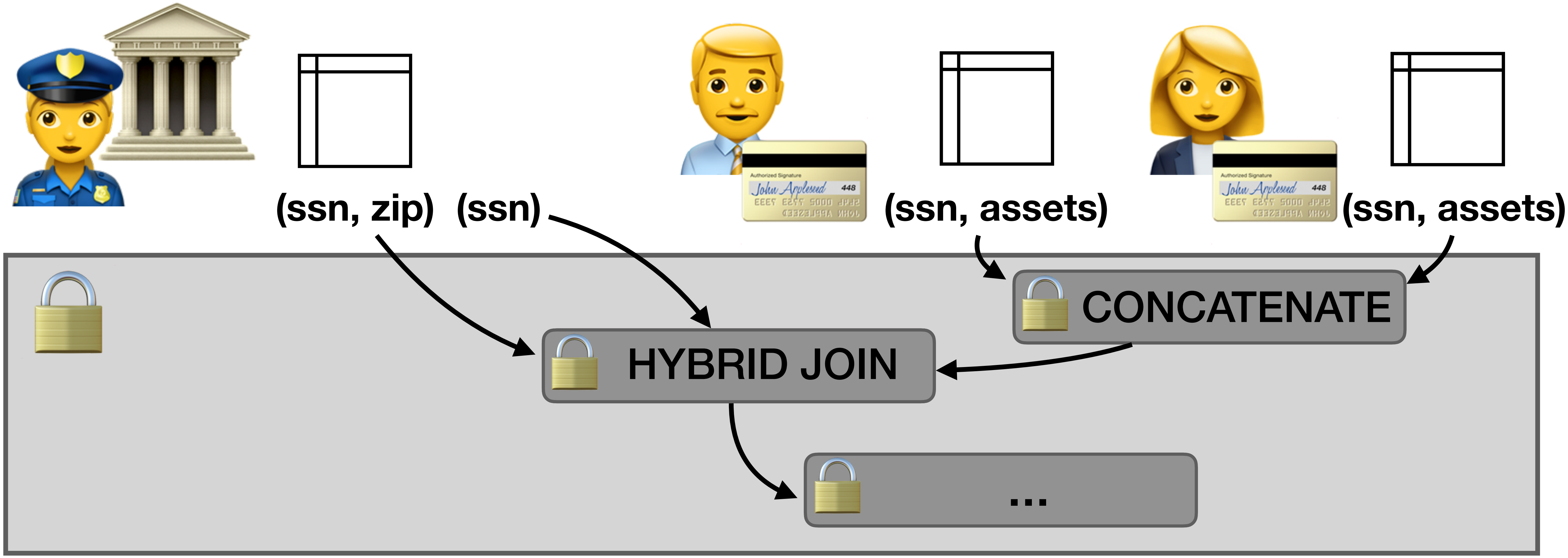


(ssn, assets)





(ssn, assets)




```
import conclave as cc
pA, pB, pC = cc.Party("mpc.ftc.gov"), cc.Party("mpc.a.com"), \
             cc.Party("mpc.b.cash")
demo_schema = [Column("ssn", cc.INTEGER),
               Column("zip", cc.INTEGER)]
demographics = cc.defineTable(demo_schema, at=pA)
# credit card companies trust the regulator to compute on SSNs
bank_schema = [Column("ssn", cc.INTEGER, trust=[pA]),
               Column("assets", cc.INTEGER)]
scores-1 = cc.defineTable(bank_schema, at=[pB])
...
```

Regulator trusted with SSN columns

Hybrid operators

- Two roles in hybrid operator scenario:
  - *Semi-trusted party* (STP) may learn a specific column in the clear; does not collude with other parties
  - *Untrusted parties* may not learn anything
- Goal:
  - Outsource expensive sub-steps to STP for local processing
  - **Without** leaking information to untrusted parties

	Complexity (Oblivious)	Complexity (Hybrid)	Bottle-neck operation (Oblivious)	Bottle-neck operation (Hybrid)
Join	$O(n^2)$ comparisons	$O(n+m \log (n+m))$ multiplications (where m is size of result)	Pair-wise comparison between all rows	Batched oblivious array access
Aggregation	$O(n \log^2 n)$ comparisons	$O(n \log n)$ multiplications	Oblivious sort	Oblivious shuffle

Evaluation

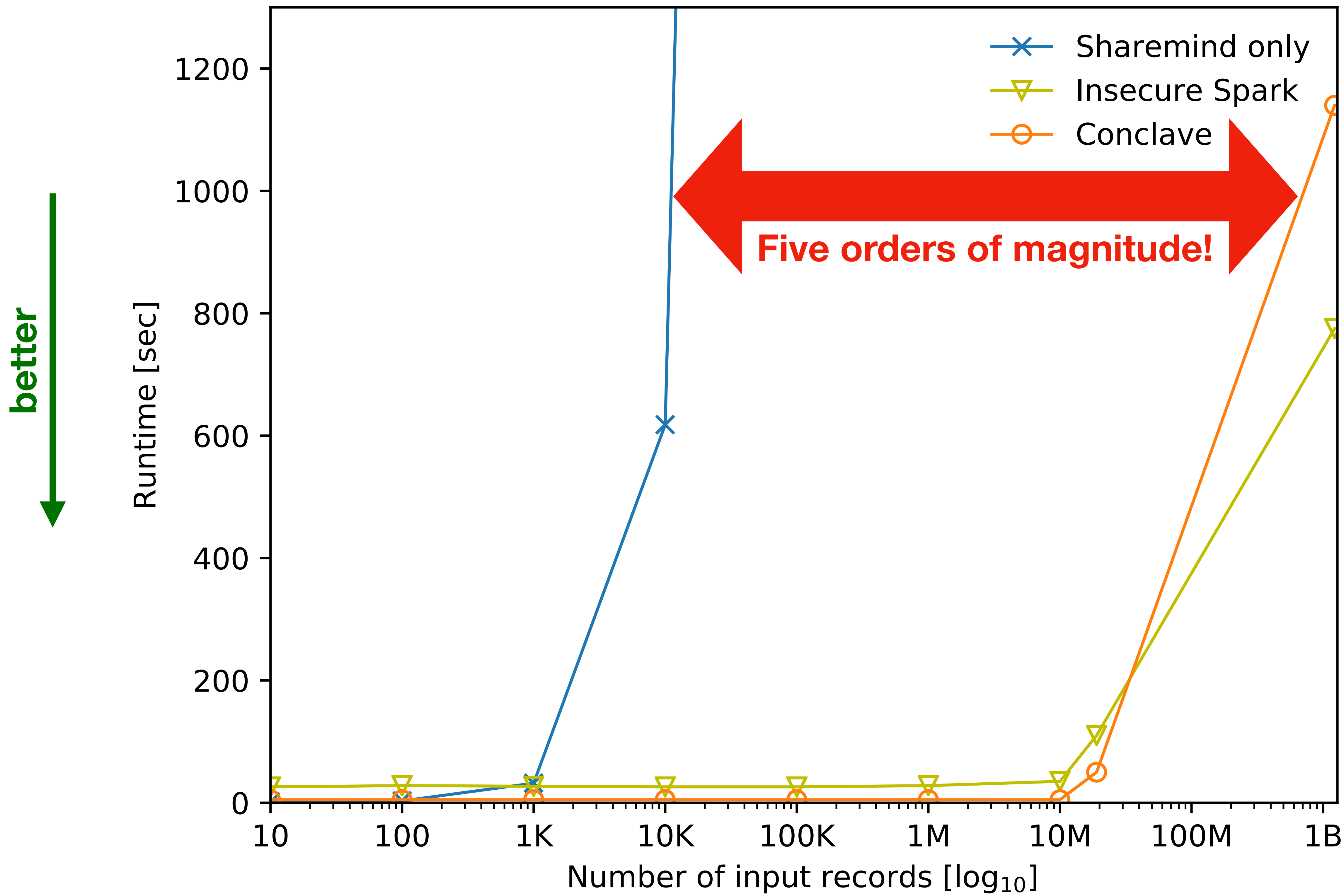
1. How does Conclave scale to increasingly large inputs?

2. How much does automatic MPC frontier placement reduce query runtime?

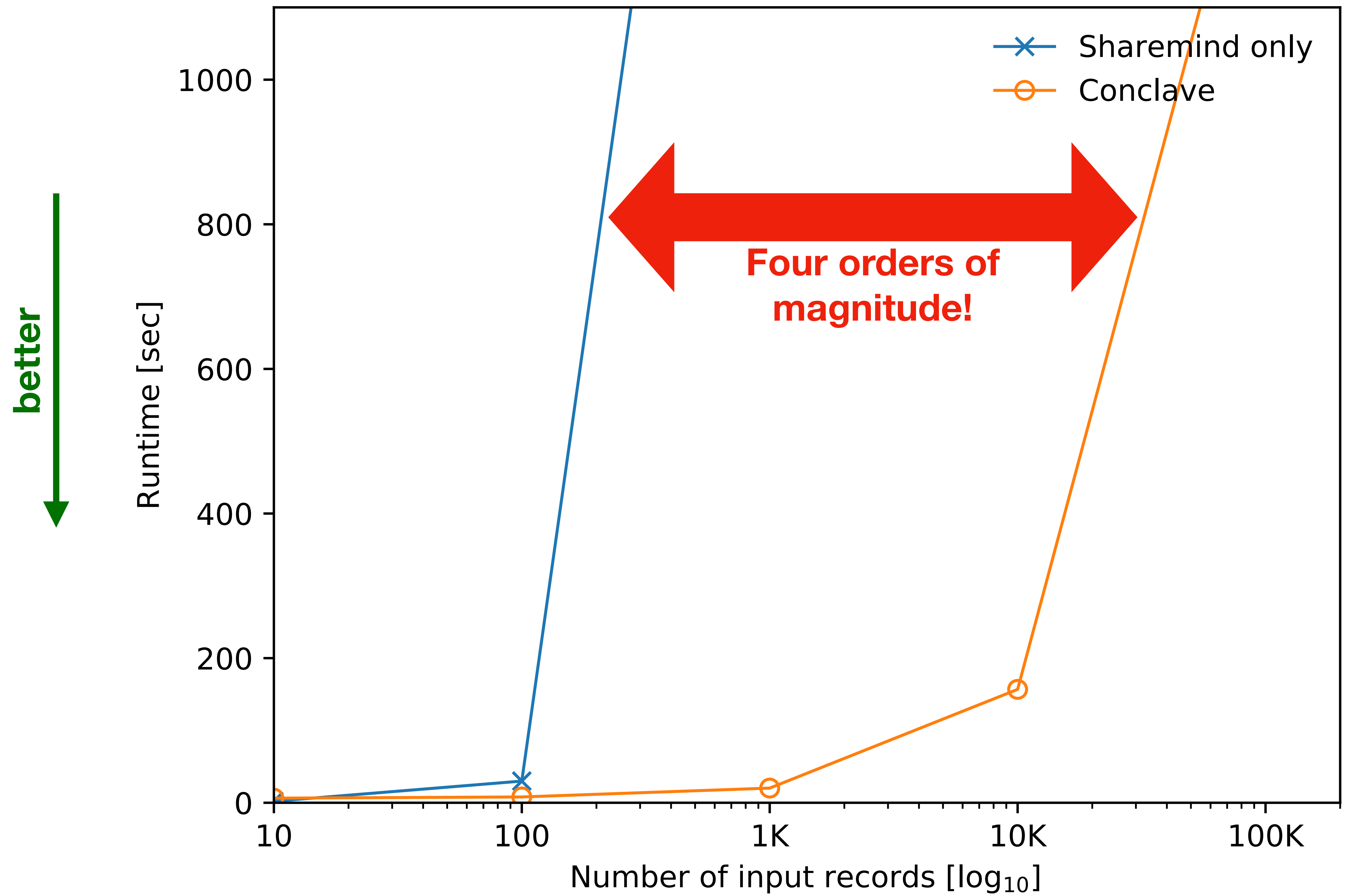
3. What impact do hybrid operators have on query runtime?

- **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



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

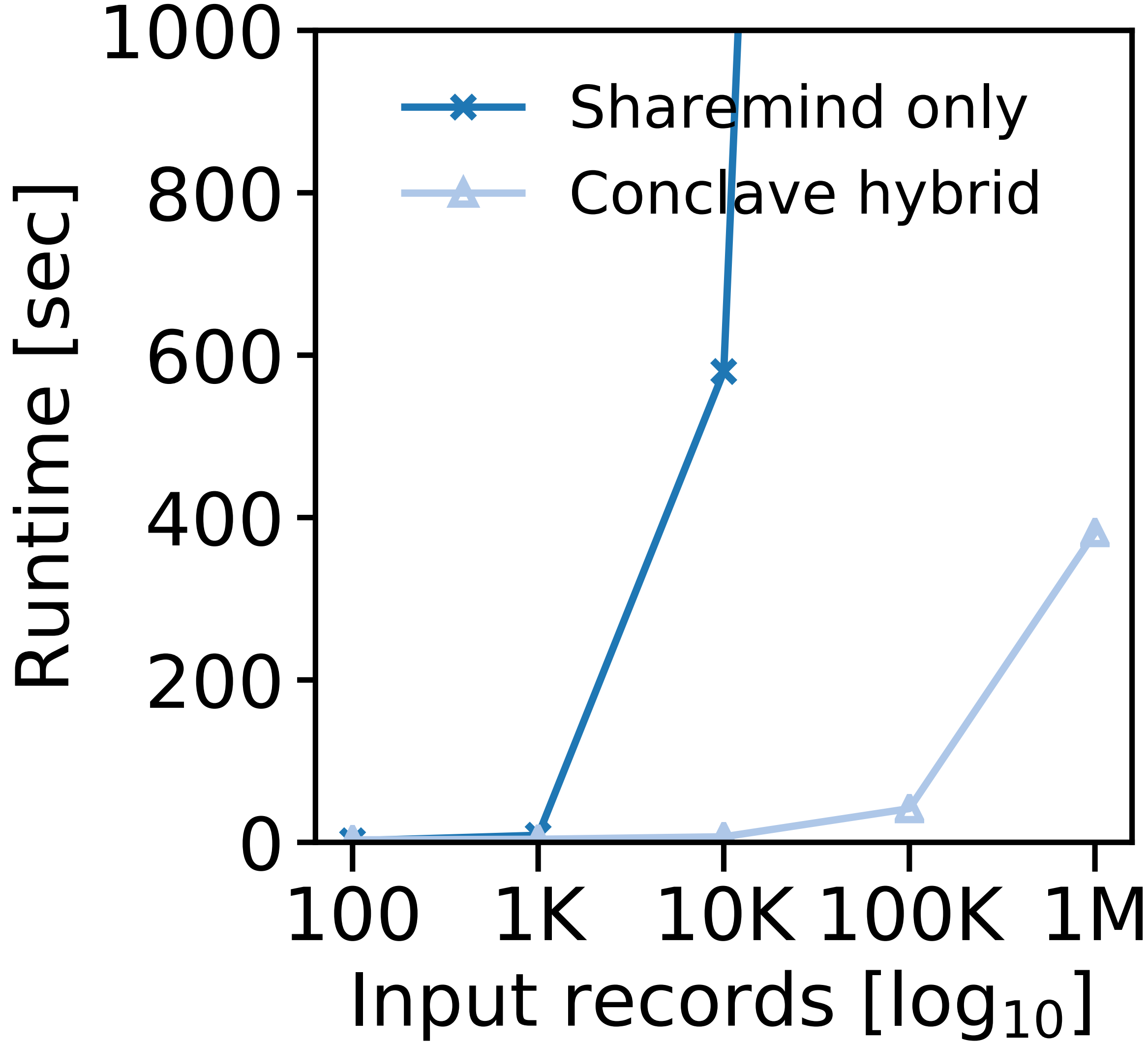
- Conclave is a query compiler for efficient MPC on “big data”
- Automatically shrinks MPC step to be as small as possible
- New hybrid MPC-cleartext protocols speed up operators
- Scales up to 5 orders of magnitude better than pure MPC

https://github.com/multiparty/conclave

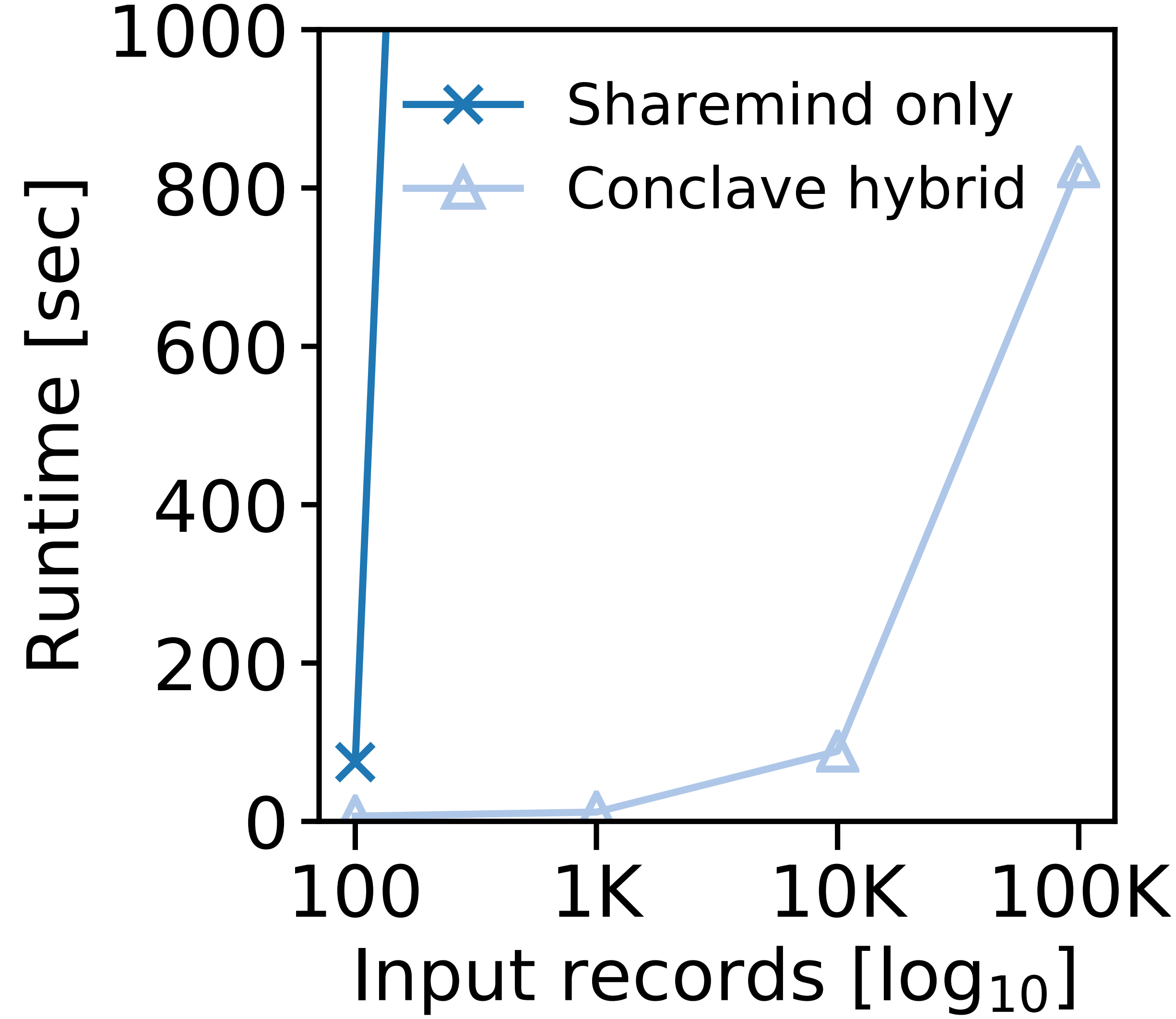
Conclave Implementation

- Relational front-end
- Rewrite rules on intermediate DAG of operators
- Back-ends generate code
  - Cleartext: Spark, sequential Python
  - MPC: Sharemind, Obliv-C (partial support)
- ~5,000 lines of Python

Hybrid MPC-cleartext operator impact



Join



Aggregation