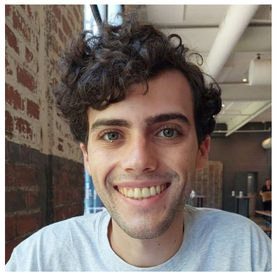
ORQ:
Complex Analytics on Private Data with Strong Security Guarantees

Eli Baum






joint work with Sam Buxbaum, Nitin Mathai, Muhammad Faisal, Vasia Kalavri, Mayank Varia, and John Liagouris

SOSP 2025 — 16 October 2025

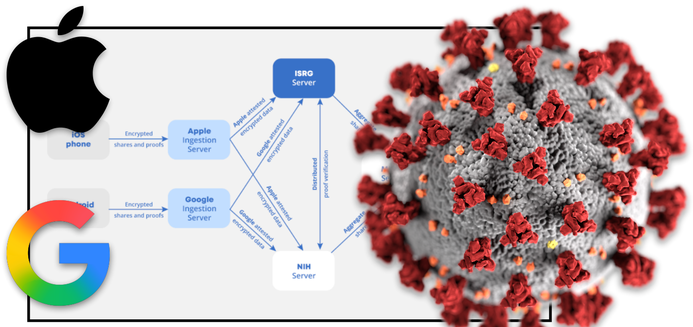
Motivation: Collaborative Analytics on Sensitive Data

Common interest

Private data

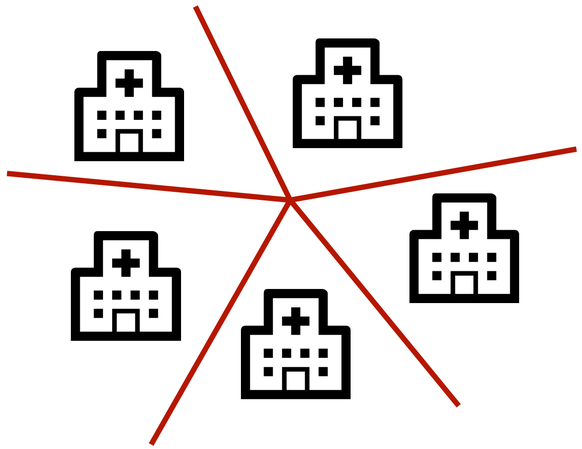
Motivation: Collaborative Analytics on Sensitive Data

Common interest



Disease Surveillance

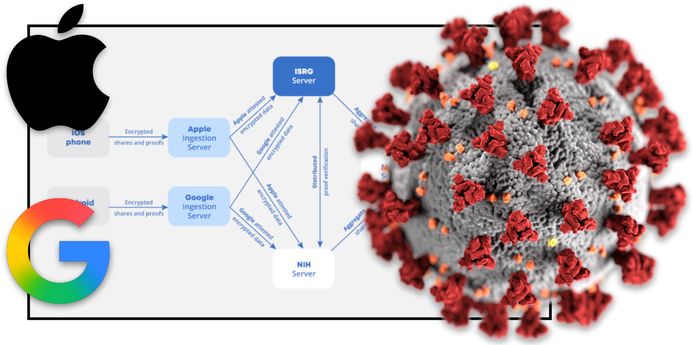
Private data



Hospitals

Motivation: Collaborative Analytics on Sensitive Data

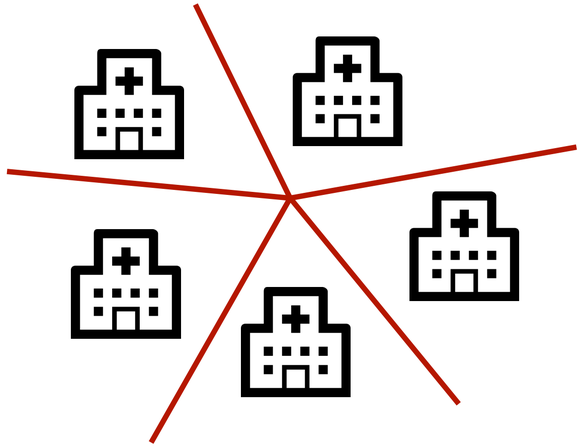
Common interest



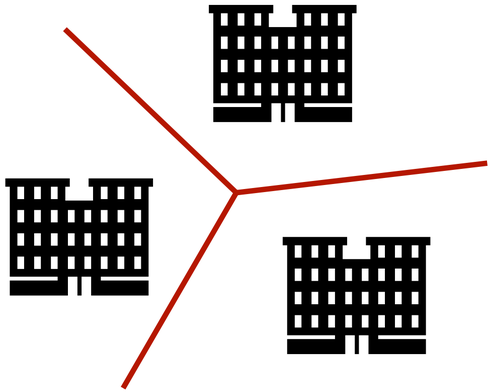
Disease Surveillance

Gender Wage Gap

Private data



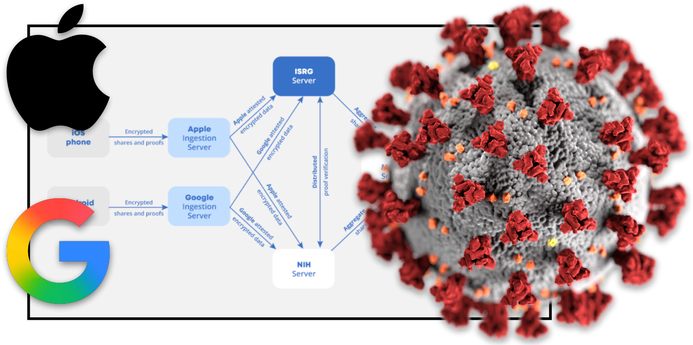
Hospitals



Employers

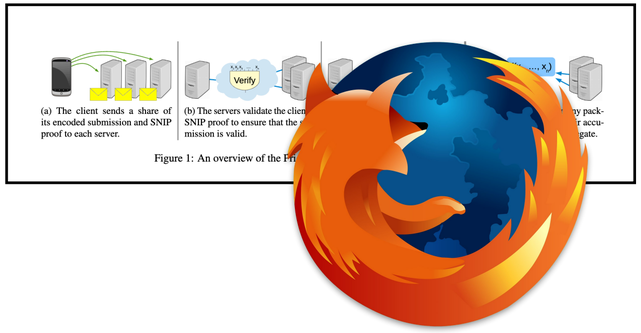
Motivation: Collaborative Analytics on Sensitive Data

Common interest



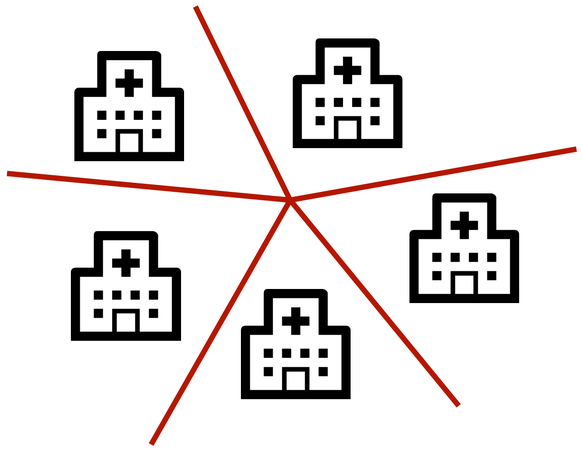
Disease Surveillance

Gender Wage Gap

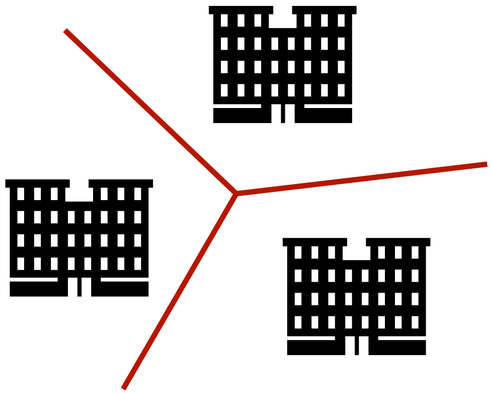


Browser Telemetry

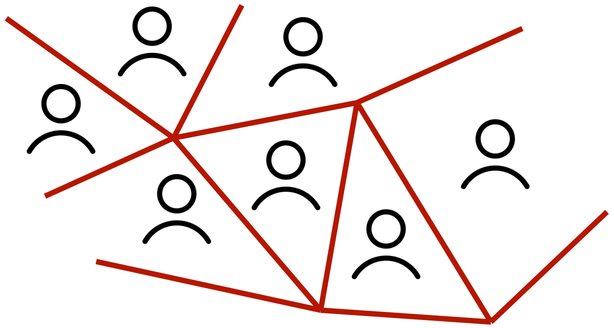
Private data



Hospitals



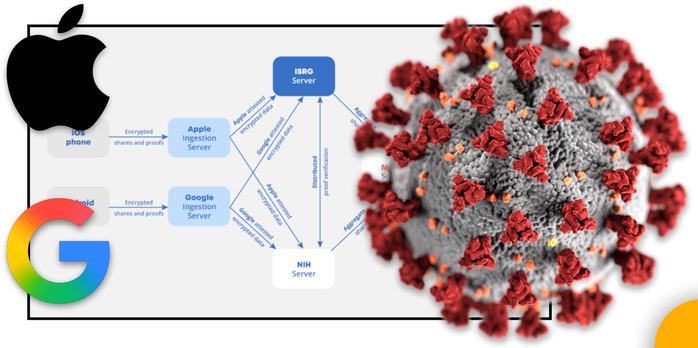
Employers



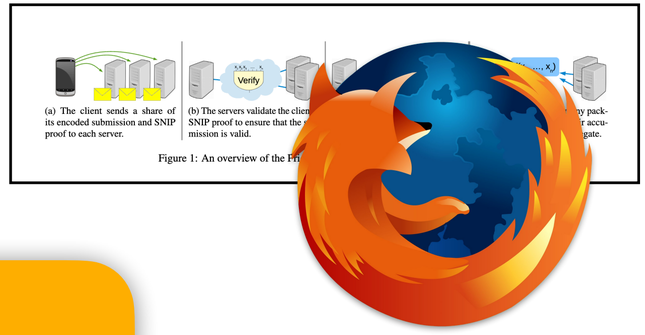
End Users

Motivation: Collaborative Analytics on Sensitive Data

Common interest



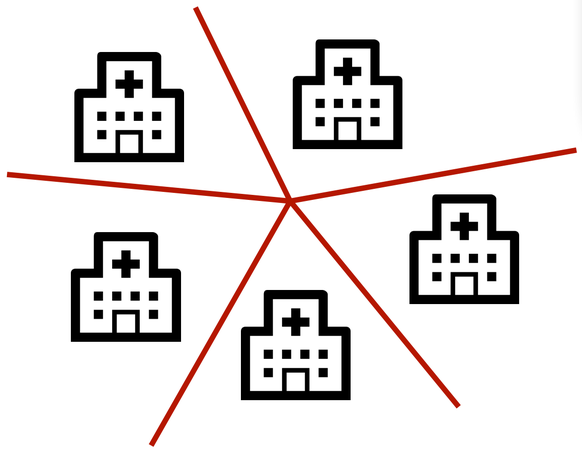
Disease Surveillance



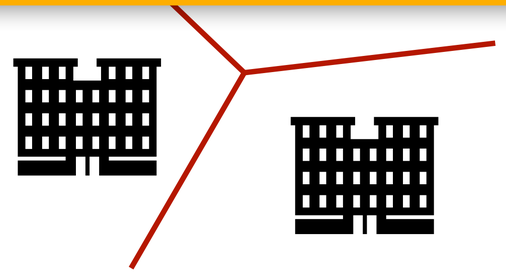
Browser Telemetry

Collaborative analytics requires securely joining data from multiple sources!

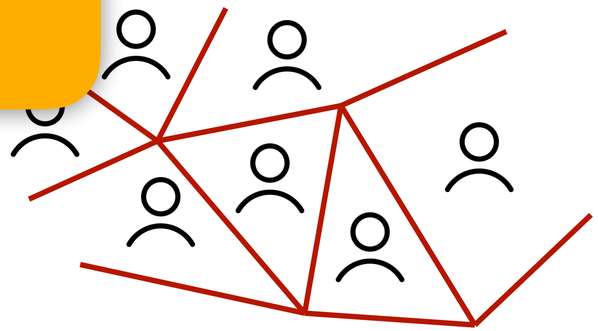
Private data



Hospitals



Employers



End Users

Outline

- Background on Oblivious Computation
- Our Approach
- Making the system work
- Results

Plaintext Analytics

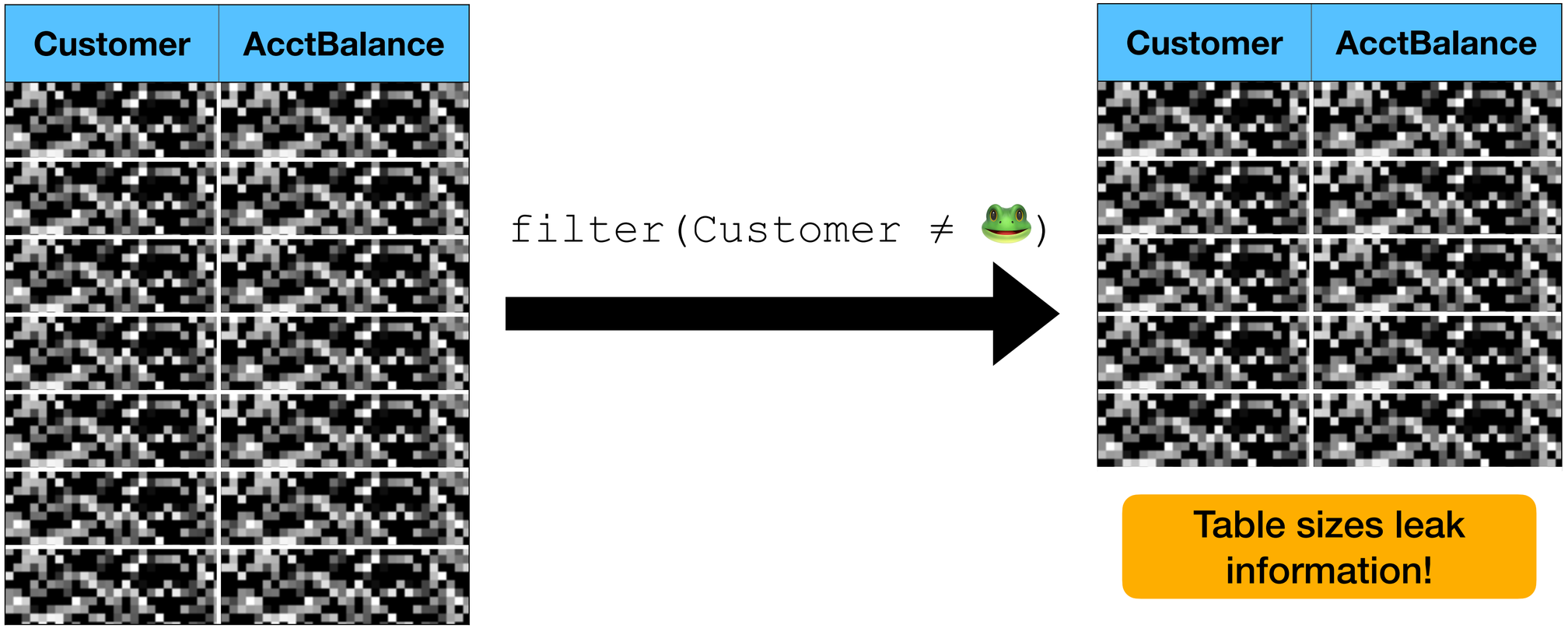
Customer	AcctBalance
🤡	$1000
😈	$2000
🤡	$500
🐸	$60
🥶	$8000
🥶	$700
🐸	-$200

`filter(Customer ≠ 🐸)`

Customer	AcctBalance
🤡	$1000
😈	$2000
🤡	$500
🥶	$8000
🥶	$700

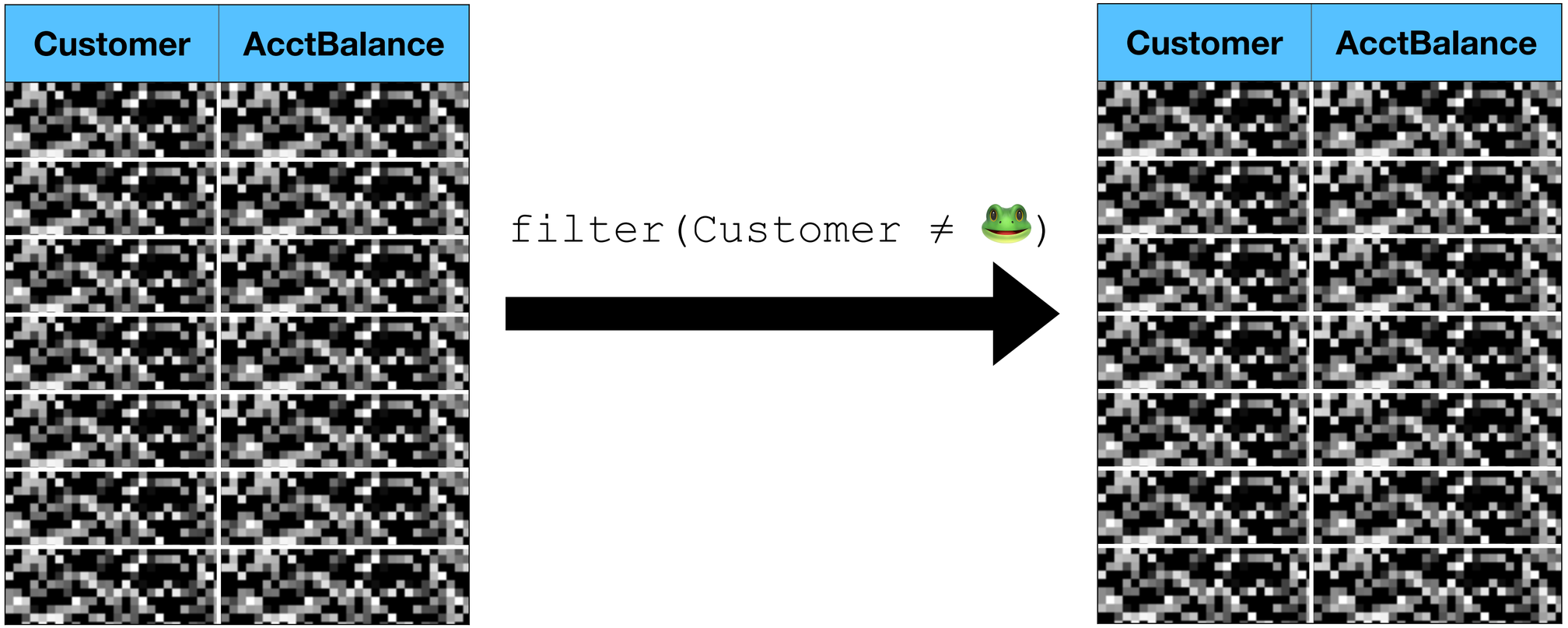
Plaintext Analytics

Adversary View



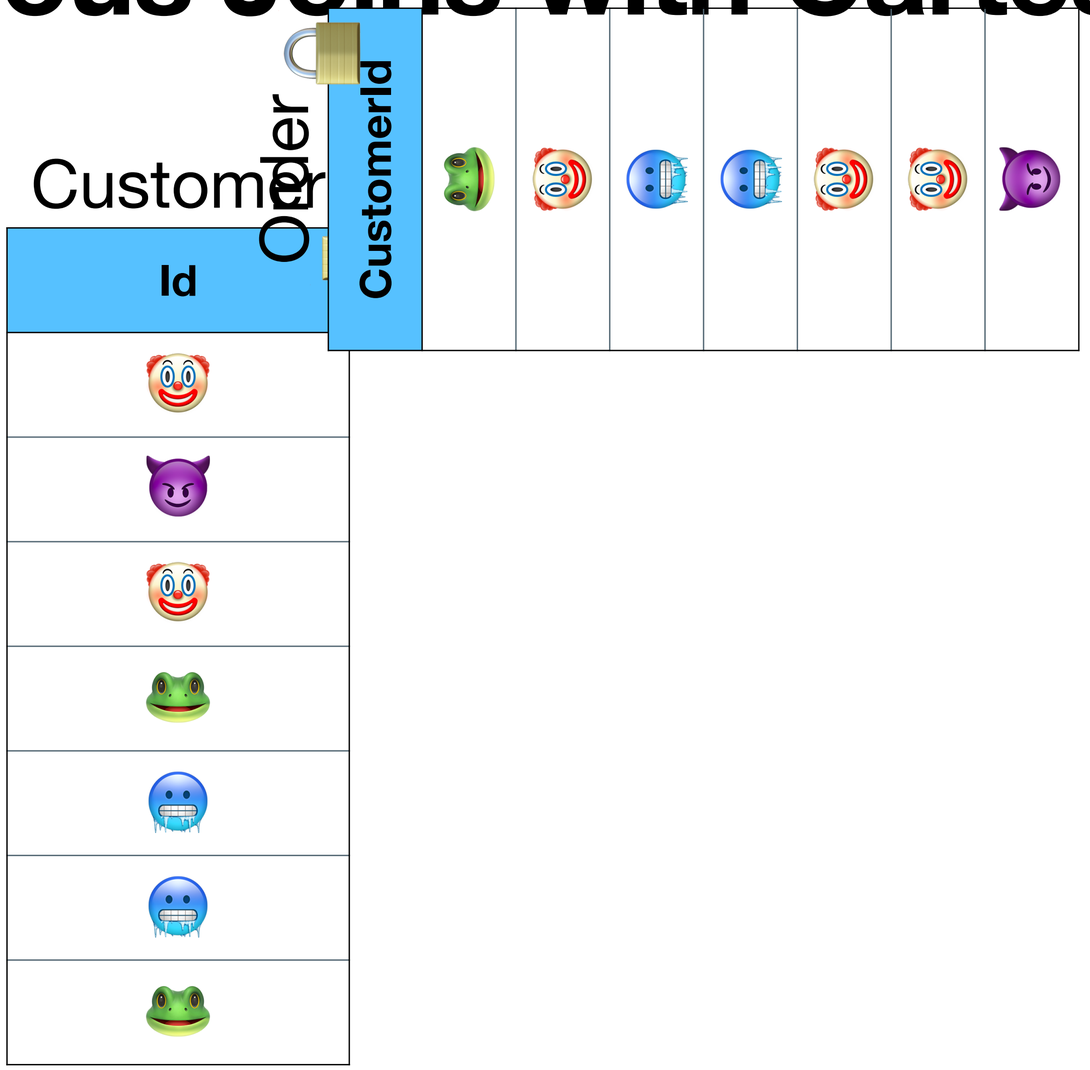
Oblivious Analytics

Complex oblivious operators must preserve **worst-case sizes**



Oblivious Joins with Cartesian Product

Oblivious Joins with Cartesian Product



Oblivious Joins with Cartesian Product

Cust \ Ord	🐸	🤡	🥶	🥶	🤡	🤡	😈

Oblivious Joins with Cartesian Product

Cust \ Ord	🐸	🤡	🥶	🥶	🤡	🤡	😈
🤡	❌	✅	❌	❌	✅	✅	❌
😈	❌	❌	❌	❌	❌	❌	✅
🤡	❌	✅	❌	❌	✅	✅	❌
🐸	✅	❌	❌	❌	❌	❌	❌
🥶	❌	❌	✅	✅	❌	❌	❌
🥶	❌	❌	✅	✅	❌	❌	❌
🐸	✅	❌	❌	❌	❌	❌	❌

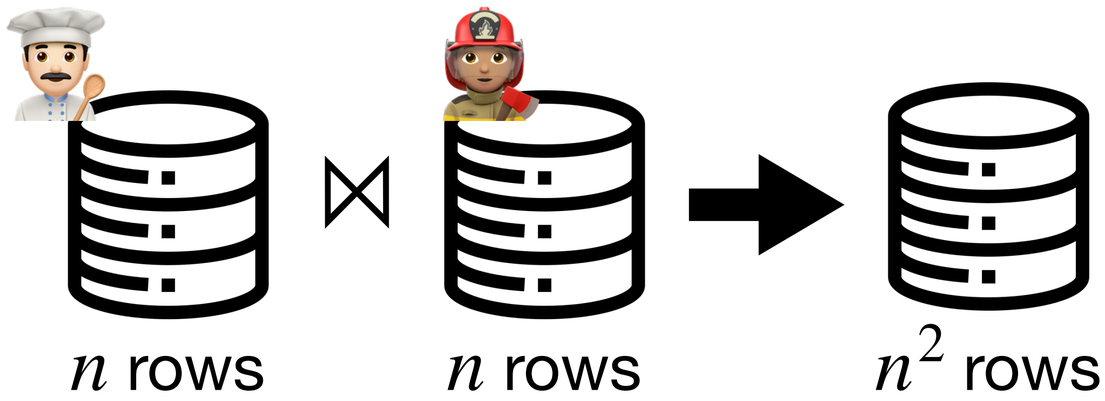
Oblivious Joins with Cartesian Product

Cust \ Ord	🐸	🤡	🥶	🥶	🤡	🤡	😈
🤡	❌	✅	❌	❌	✅	✅	❌
😈	❌	❌	❌	❌	❌	❌	✅
🤡	❌	✅	❌	❌	✅	✅	❌
🐸	✅	❌	❌	❌	❌	❌	❌
🥶	❌	❌	✅	✅	❌	❌	❌
🥶	❌	❌	✅	✅	❌	❌	❌
🐸	✅	❌	❌	❌	❌	❌	❌

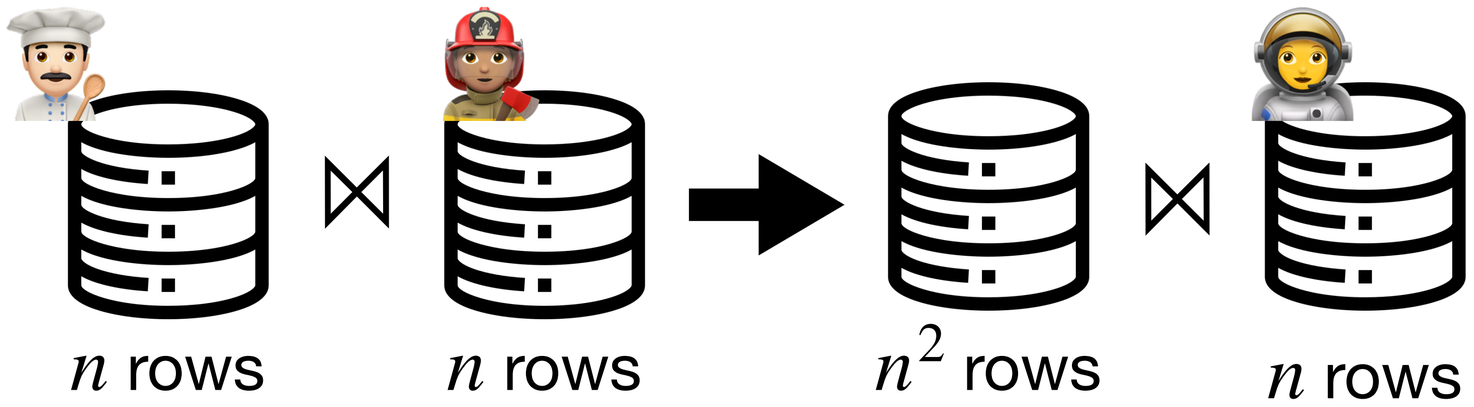
🔒

$O(n^2)$ work
$O(n^2)$ output size

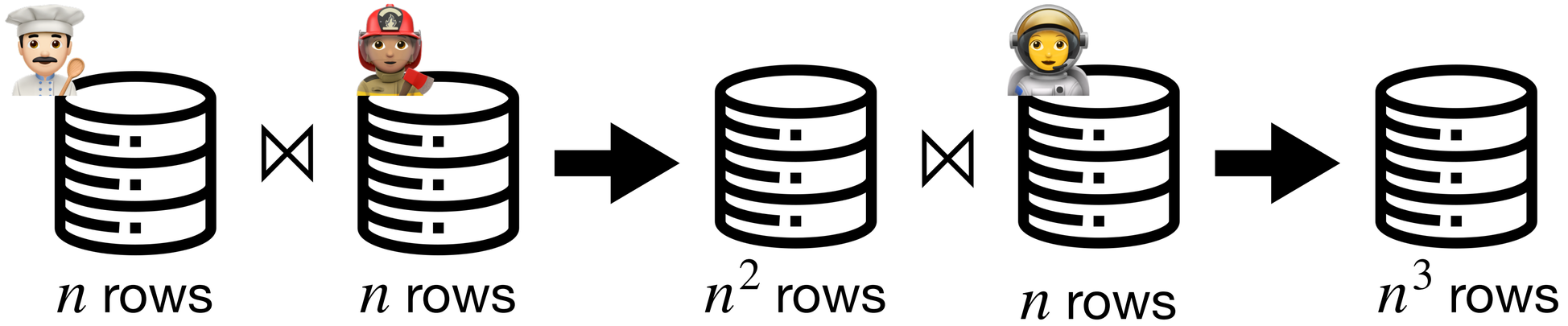
The Problem

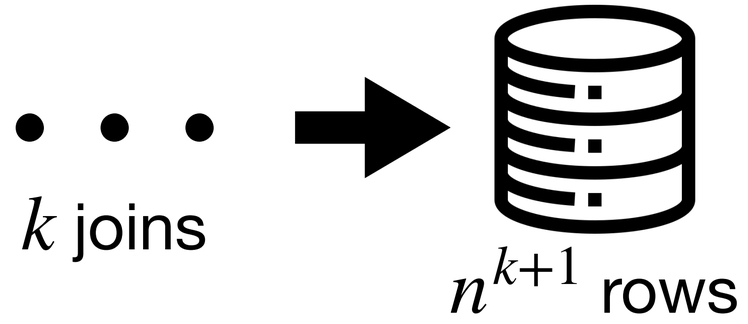


The Problem



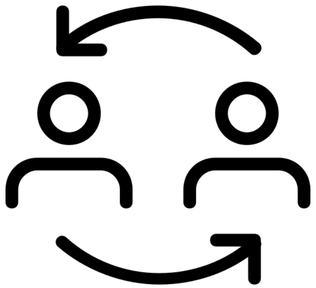
The Problem


n rows
$\bowtie$
n rows
n^2 rows
$\bowtie$
n rows
n^3 rows


k joins
n^{k+1} rows

Prior Approaches to Collaborative Analytics

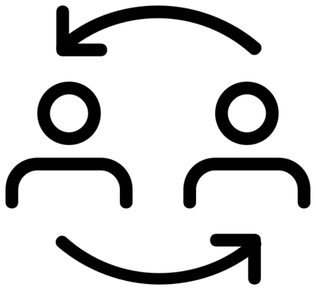
Prior Approaches to Collaborative Analytics



Peer-to-peer setting

e.g., Conclave (EuroSys'19),
Senate (Security'21), SMCQL (VLDB'17),
Shrinkwrap (VLDB'18)

Prior Approaches to Collaborative Analytics

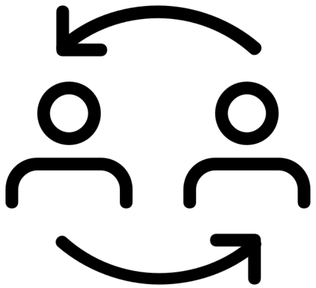


Peer-to-peer setting

e.g., Conclave (EuroSys'19),
Senate (Security'21), SMCQL (VLDB'17),
Shrinkwrap (VLDB'18)

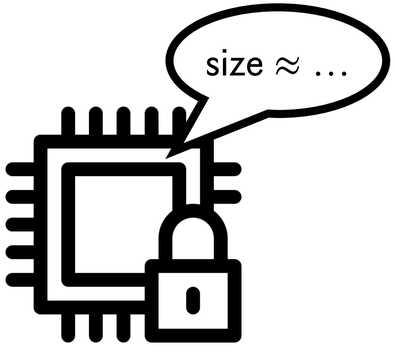
❌ Doesn't scale with more parties

Prior Approaches to Collaborative Analytics



Peer-to-peer setting

e.g., Conclave (EuroSys'19),
Senate (Security'21), SMCQL (VLDB'17),
Shrinkwrap (VLDB'18)

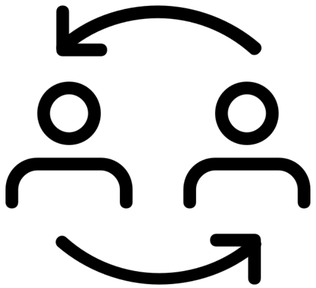


Leakage or upper bounds

e.g., Opaque (NSDI'17), Obliviator (Security'25),
SAQE (VLDB'20)

❌ Doesn't scale with more parties

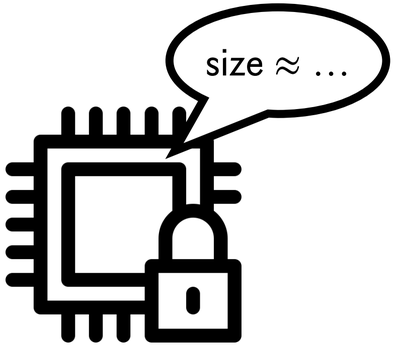
Prior Approaches to Collaborative Analytics



Peer-to-peer setting

e.g., Conclave (EuroSys'19),
Senate (Security'21), SMCQL (VLDB'17),
Shrinkwrap (VLDB'18)

❌ Doesn't scale with more parties

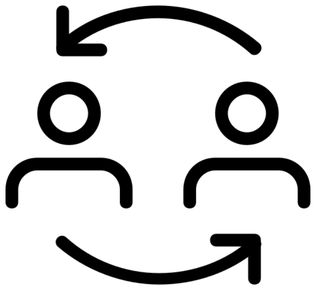


Leakage or upper bounds

e.g., Opaque (NSDI'17), Obliviator (Security'25),
SAQE (VLDB'20)

❌ Only provides partial security

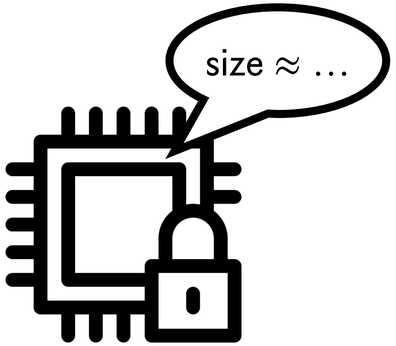
Prior Approaches to Collaborative Analytics



Peer-to-peer setting

e.g., Conclave (EuroSys'19),
Senate (Security'21), SMCQL (VLDB'17),
Shrinkwrap (VLDB'18)

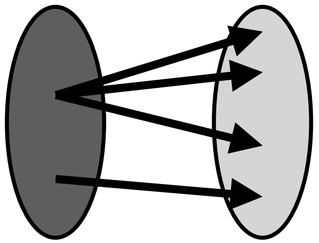
❌ Doesn't scale with more parties



Leakage or upper bounds

e.g., Opaque (NSDI'17), Obliviator (Security'25),
SAQE (VLDB'20)

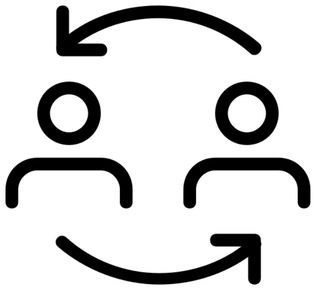
❌ Only provides partial security



Individual join operators

e.g., [MRR] (CCS'20), [BDG+] (CCS'22),
[AHK+] (CCS'23)

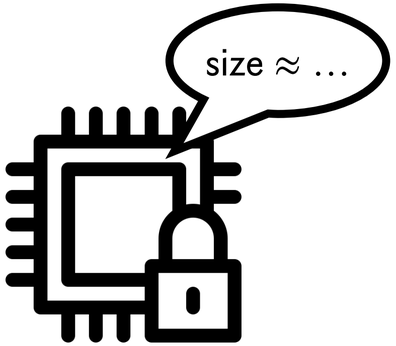
Prior Approaches to Collaborative Analytics



Peer-to-peer setting

e.g., Conclave (EuroSys'19), Senate (Security'21), SMCQL (VLDB'17), Shrinkwrap (VLDB'18)

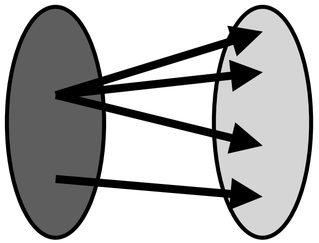
❌ Doesn't scale with more parties



Leakage or upper bounds

e.g., Opaque (NSDI'17), Obliviator (Security'25), SAQE (VLDB'20)

❌ Only provides partial security



Individual join operators

e.g., [MRR] (CCS'20), [BDG+] (CCS'22), [AHK+] (CCS'23)

❌ Cannot evaluate queries

Can we efficiently evaluate complex analytics without compromising security?

Can we efficiently evaluate complex analytics without compromising security?

Yes!

Can we efficiently evaluate complex analytics without compromising security?

Yes!

We are the first to show **multiple joins** with no security compromises

…and run **all queries** from prior work, with **orders-of-magnitude** larger inputs

What do real-world queries look like?

What do real-world queries look like?

We collect all **31 queries** from related work in MPC analytics

What do real-world queries look like?

We collect all **31 queries** from related work in MPC analytics

All have output size bound $O(n)$ instead of $O(n^k)$!

Why?

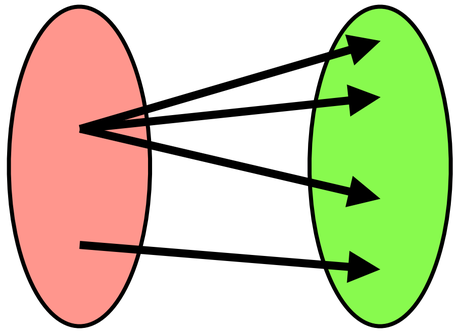
What do real-world queries look like?

We collect all **31 queries** from related work in MPC analytics

All have output size bound $O(n)$ instead of $O(n^k)$!

Why?

one-to-many



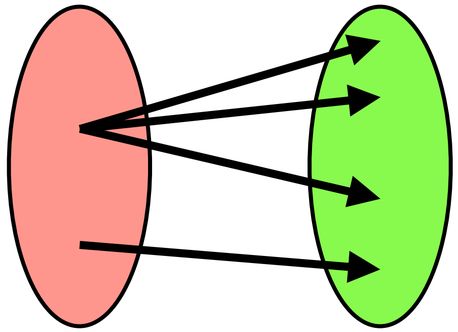
What do real-world queries look like?

We collect all **31 queries** from related work in MPC analytics

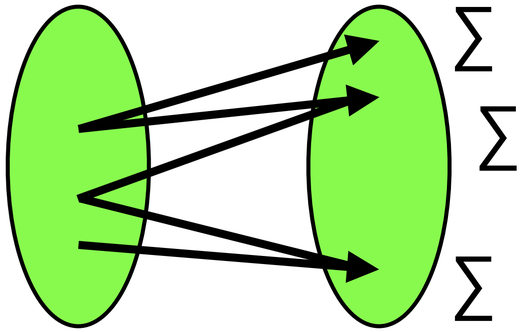
All have output size bound $O(n)$ instead of $O(n^k)$!

Why?

one-to-many



many-to-many + aggregation



What do real-world queries look like?

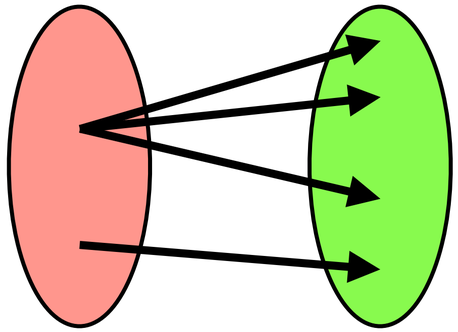
We collect all **31 queries** from related work in MPC analytics

All have output size bound $O(n)$ instead of $O(n^k)$!

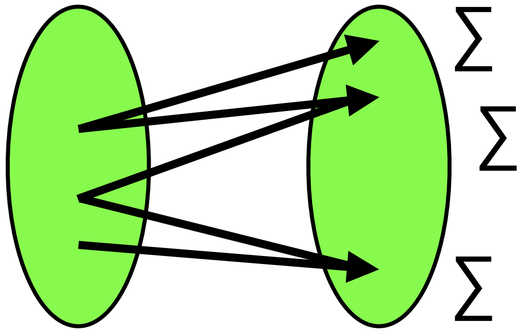
Why?

decomposable

one-to-many



many-to-many + aggregation



Our Insight

Our Insight

The general problem is impractically difficult

Our Insight

The general problem is impractically difficult

Practical instances are not!

Our Insight

The general problem is impractically difficult

Practical instances are not!

Common query semantics impose an
$O(n)$ bound on the output

Our Insight

The general problem is impractically difficult

Practical instances are not!

Common query semantics impose an
$O(n)$ bound on the output

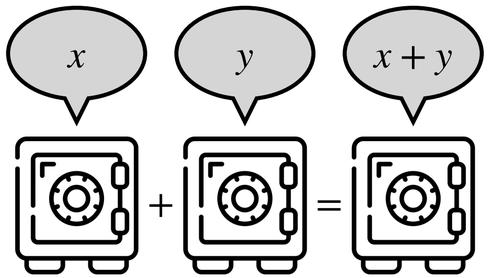
Decompose joins into sort + aggregation

Our Insight

The general problem is impractically difficult

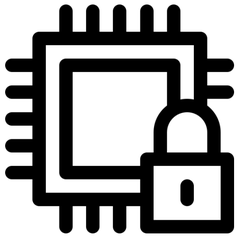
Practical instances are not!

Common query semantics impose an
$O(n)$ bound on the output



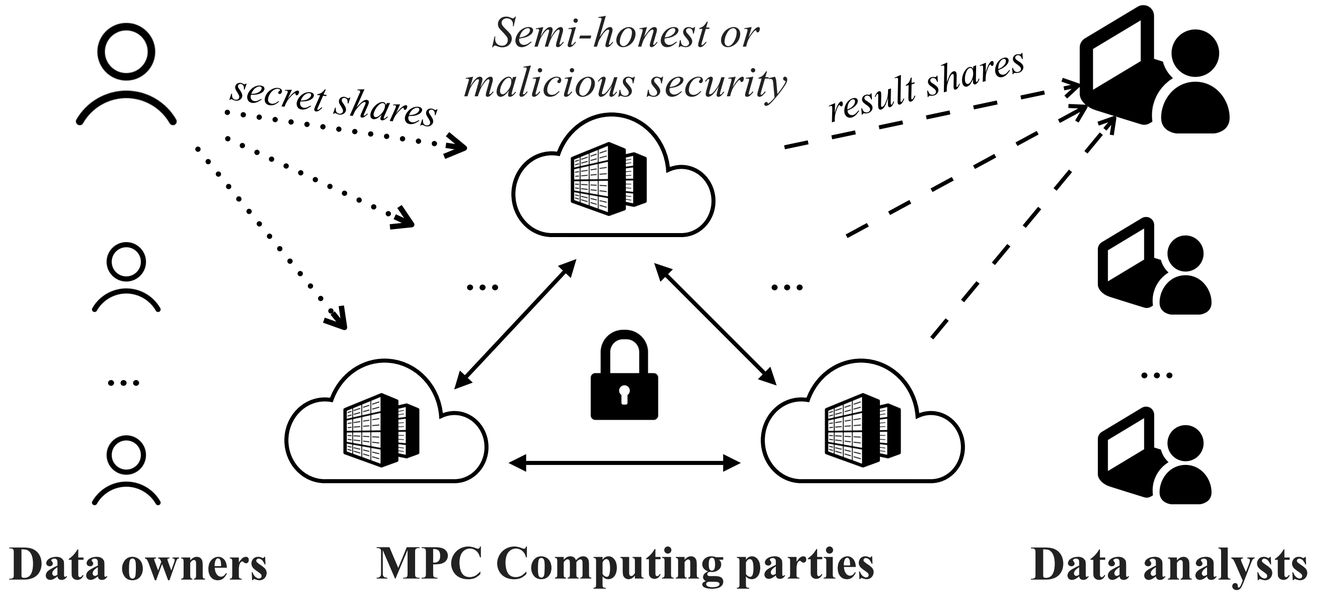
Decompose joins into sort + aggregation

Applies beyond multiparty computation!



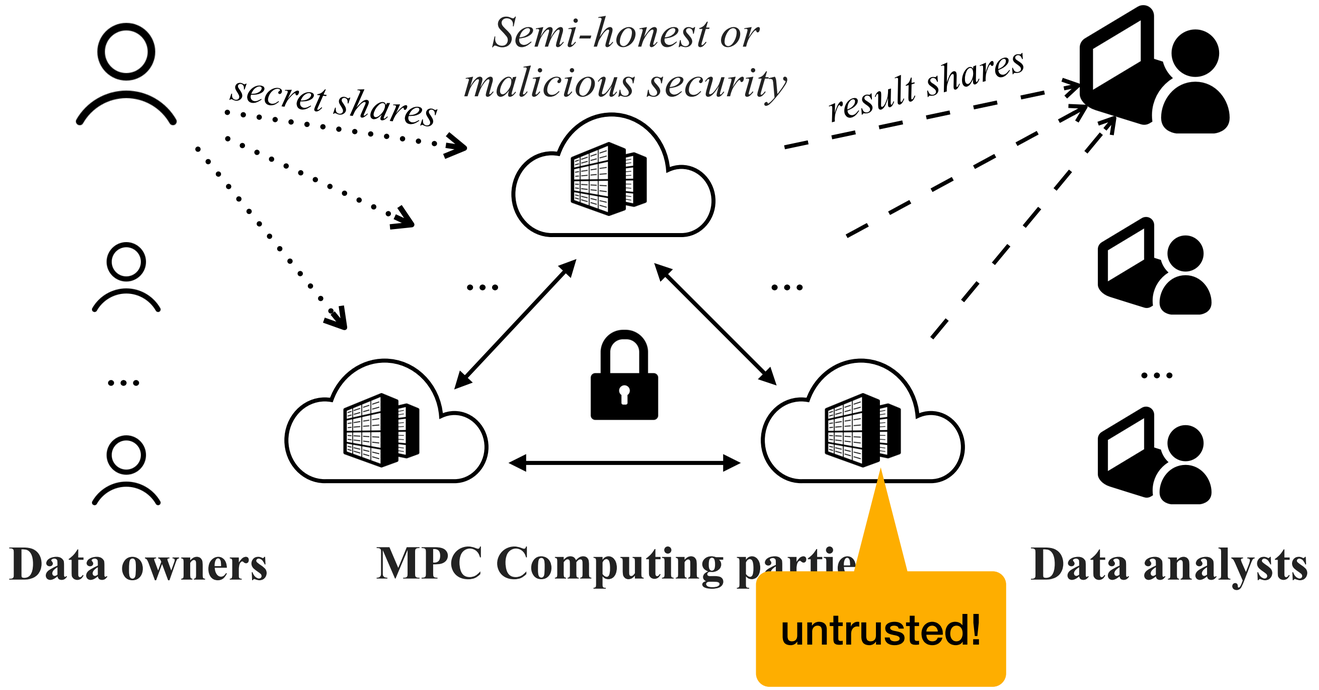
Secure Multiparty Computation (MPC)

Outsourced Setting



Secure Multiparty Computation (MPC)

Outsourced Setting



Secret Sharing: Data Owners

ID	Balance
1	$100
4	$50
9	$25
7	-$13

ID	Balance
3	$814
2	$312
7	$60
6	-$366

ID	Balance
14	$82
12	$14
0	-$8
3	$160

Secret Sharing: Data Owners

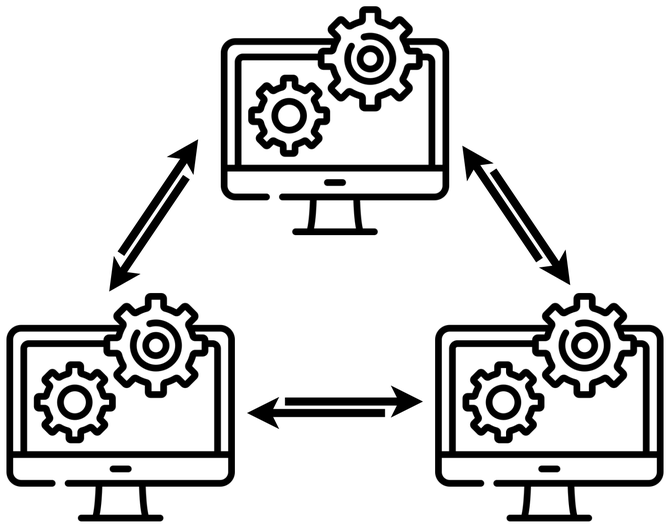
ID	Balance
1799JtRkU	NV9MVqRf
qicpdFpxn	ZgaoE7Hx
JtRkUmc5	FVwp7XTF
qSiJ3nBG	vhnCMIo7j

ID	Balance
UW6m9tx	EABqfgpRi
m6vQFTgb	9u19eWyV
qhq2JFbe	hM2hOg8
MQeNGgyI	q1uon8xM

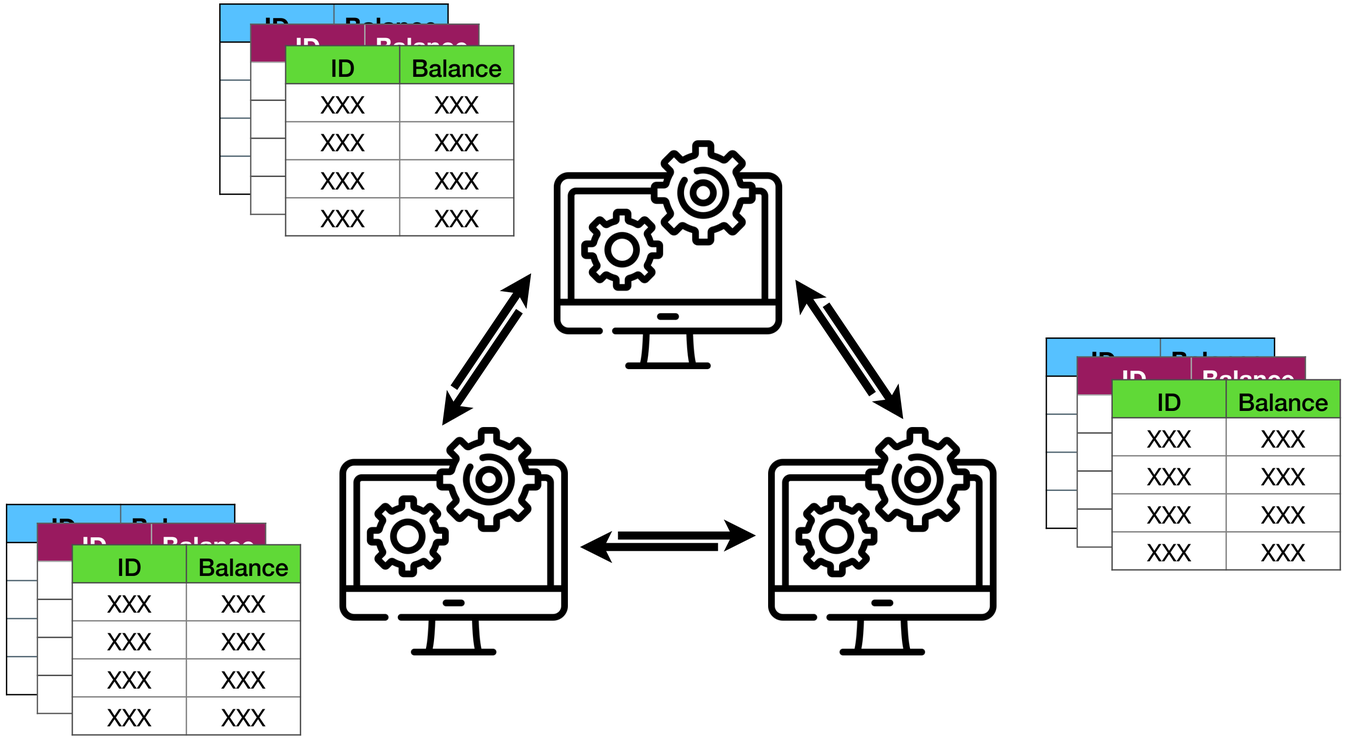
ID	Balance
hsIcAgWLy	OUPgb2V
MVAJi4Vnt	FzI0j2hH8
8KsbVnN	YIaNhg5py
ehbzAtm6j	LmWp415

Secret Sharing: Data Owners

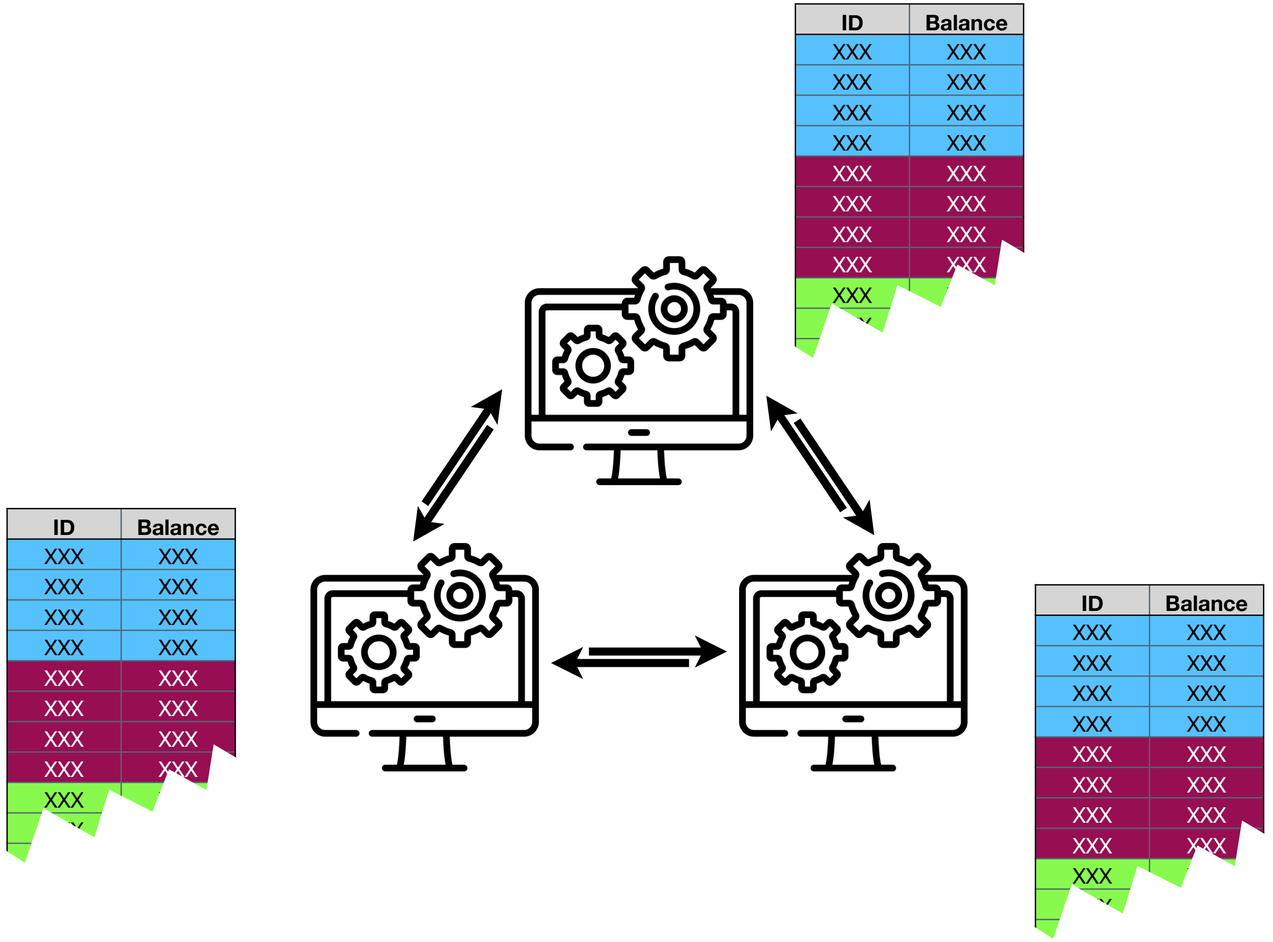
Secret Sharing: Computing Parties



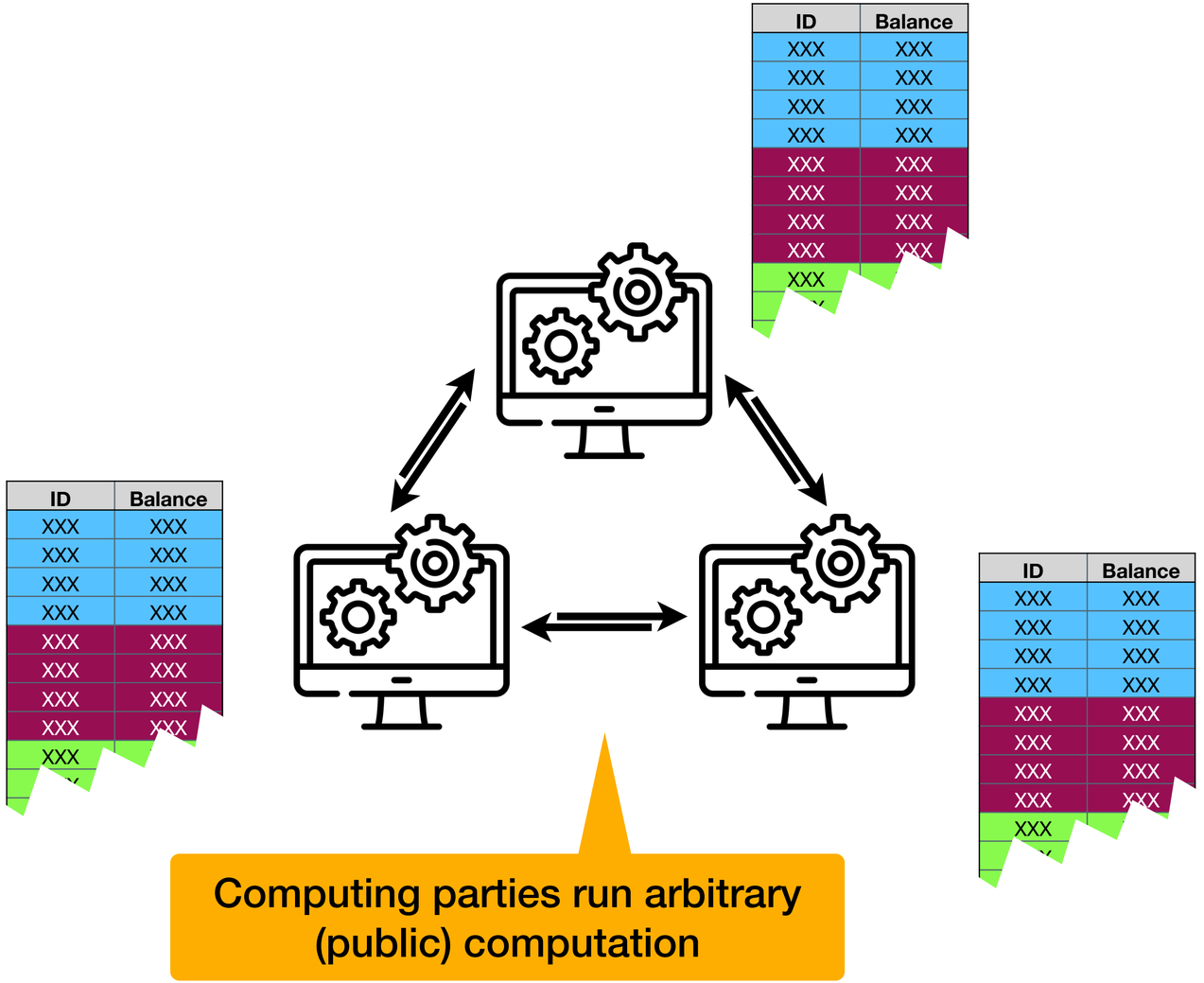
Secret Sharing: Computing Parties



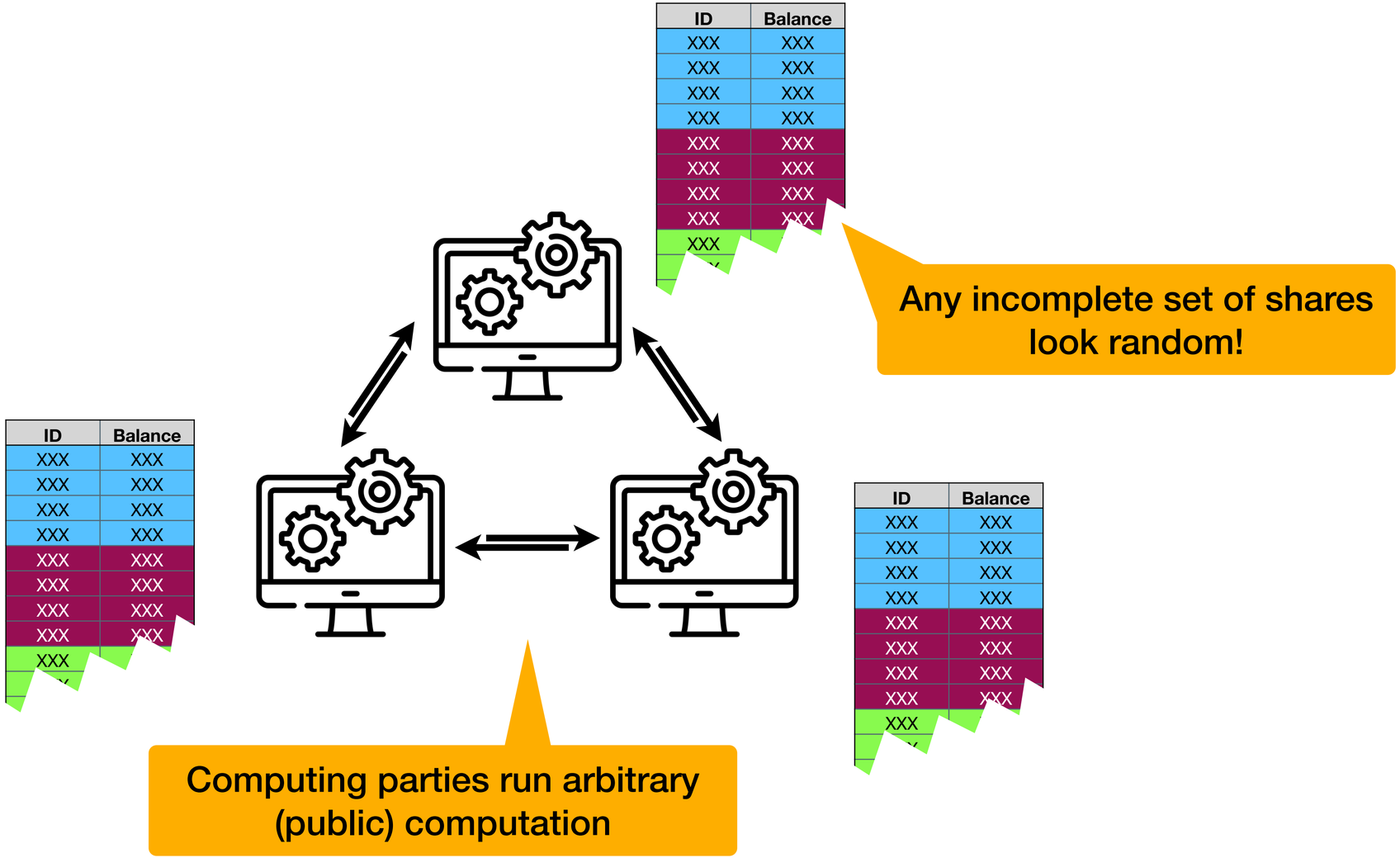
Secret Sharing: Computing Parties



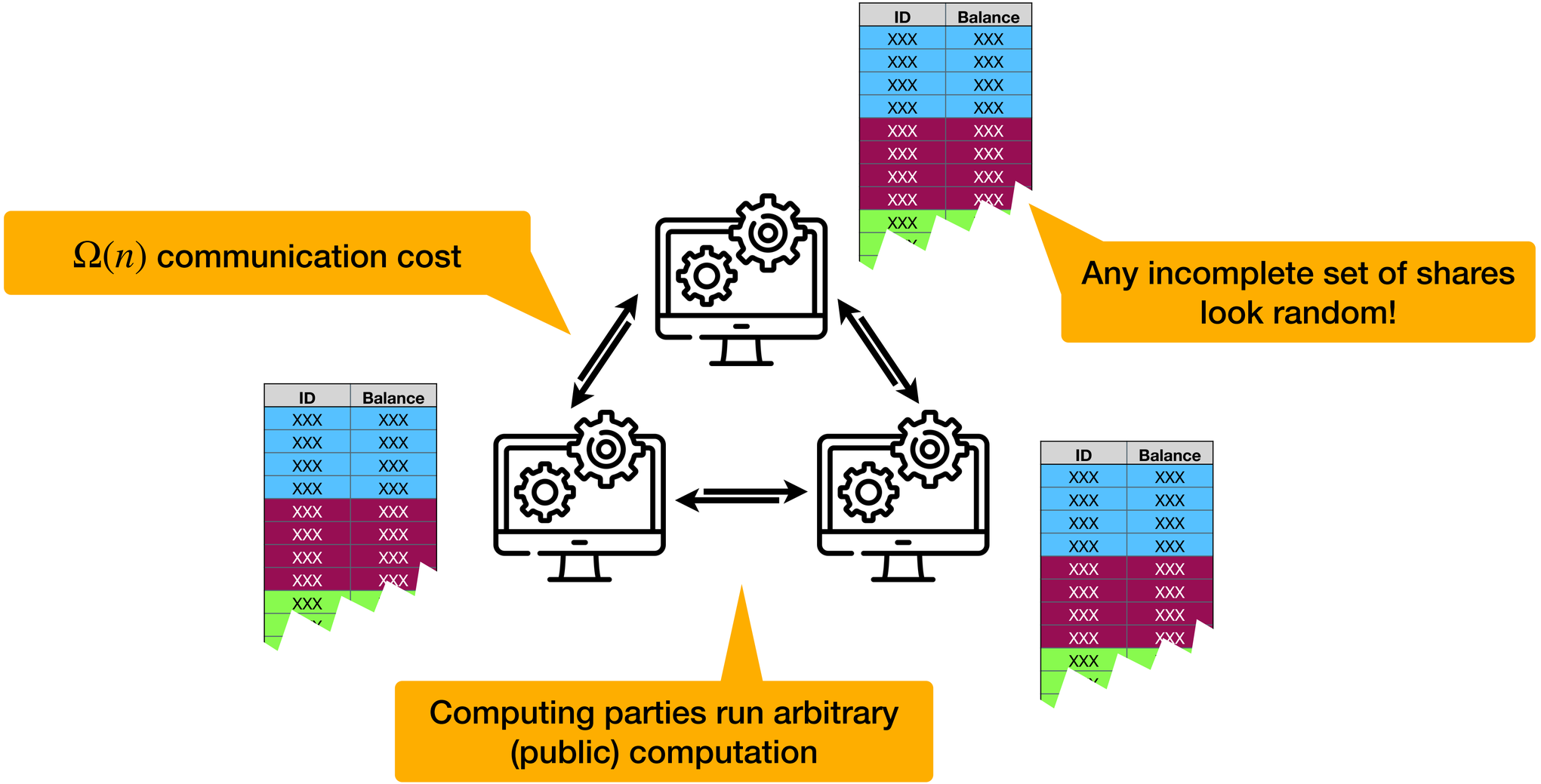
Secret Sharing: Computing Parties



Secret Sharing: Computing Parties



Secret Sharing: Computing Parties



Secret Shared Tables 🔒

ID	Balance
1799JtRkU	NV9MVqRf
qicpdFpxn	ZgaoE7Hx
JtRkUmc5	FVwp7XTF
qSiJ3nBG	vhnCMIo7j

ID	Balance
UW6m9tx	EABqfgpRi
m6vQFTgb	9u19eWyV
qhq2JFbe	hM2hOg8
MQeNGgyI	q1uon8xM

ID	Balance
hsIcAgWLy	OUPgb2V
MVAJi4Vnt	FzI0j2hH8
8KsbVnN	YIaNhg5py
ehbzAtm6j	LmWp415

Secret Shared Tables 🔒

ID	Balance
1	$100
4	$50
9	$25
7	-$13

ID	Balance
3	$814
2	$312
7	$60
6	-$366

ID	Balance
14	$82
12	$14
0	-$8
3	$160

Improving on Quadratic: Sort & Aggregate

One table must have unique keys

```
func join(L, R, keys):
   T := concat*(L, R)
   T.sort(keys)
   T.prefix_aggregate(…)
   return T
```

Improving on Quadratic: Sort & Aggregate

One table must have unique keys

$O(n \log n)$

```
func join(L, R, keys):
    T := concat*(L, R)
    T.sort(keys)
    T.prefix_aggregate(…)
    return T
```

$O(n \log n)$

Improving on Quadratic: Sort & Aggregate

One table must have unique keys

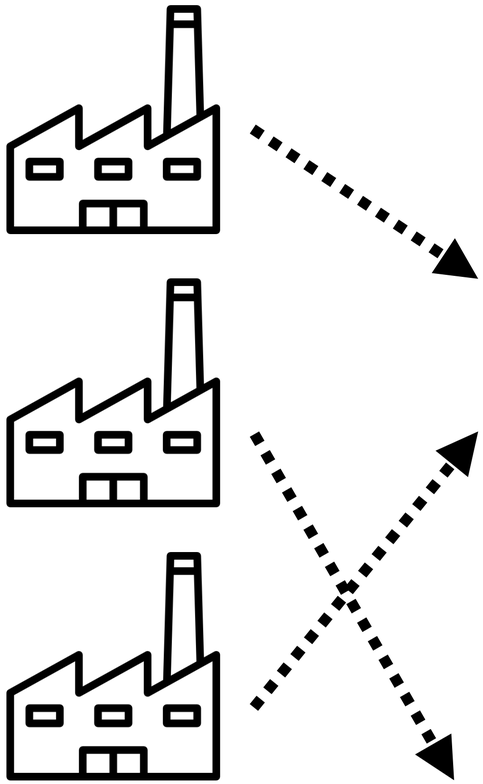
$O(n \log n)$

```
func join(L, R, keys):
    T := concat*(L, R)
    T.sort(keys)
    T.prefix_aggregate(…)
    return T
```

$O(n \log n)$

Extends to **inner, outer, anti,** and **semi** joins.

Example with Our Approach



Customers

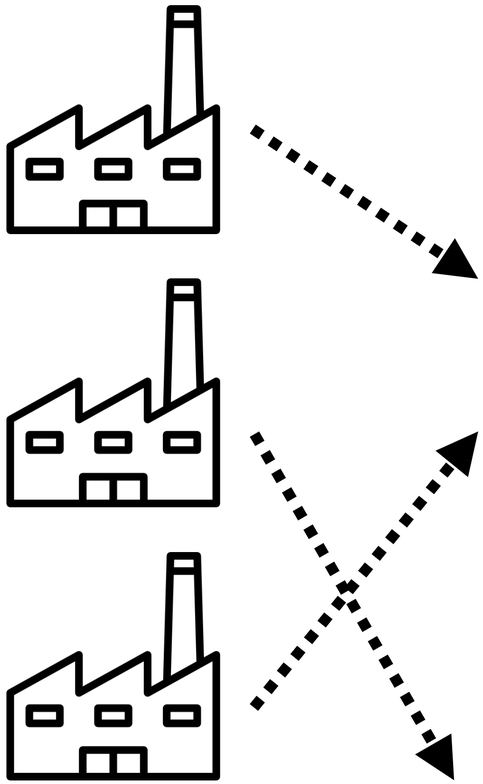
Customer	AcctBalance
🤡	$1000
😈	$2000
🤡	$500
🐸	$60
🥶	$8000
🥶	$700
🐸	-$200

⋈

Orders

Customer	OrdCost
🐸	$30
🤡	$20
🥶	$55
🥶	$18
🤡	$70
🤡	$12
😈	$8

Example with Our Approach



Customers

Customer	AcctBalance
🤡	$1000
😈	$2000
🤡	$500
🐸	$60
🥶	$8000
🥶	$700
🐸	-$200

⋈

Orders

Customer	OrdCost
🐸	$30
🤡	$20
🥶	$55
🥶	$18
🤡	$70
🤡	$12
😈	$8

duplicates!

Example with Our Approach

```
SELECT
    SPECIES, SUM(ACCTBALANCE) / SUM(ORDCOST)
FROM
    ORDERS, CUSTOMERS
WHERE
    ORDER.CUSTOMER = CUSTOMERS.CUSTOMER
GROUP BY
    SPECIES
```

Per customer group, what is the ratio of account balance to order total?

Customers

Customer	AcctBalance
🤡	$1000
😈	$2000
🤡	$500
🐸	$60
🥶	$8000
🥶	$700
🐸	-$200

Orders

Customer	OrdCost
🐸	$30
🤡	$20
🥶	$55
🥶	$18
🤡	$70
🤡	$12
😈	$8

Pre-Aggregate

Customers

Customer	AcctBalance
🤡	$1000
😈	$2000
🤡	$500
🐸	$60
🥶	$8000
🥶	$700
🐸	-$200

Orders

Customer	OrdCost
🐸	$30
🤡	$20
🥶	$55
🥶	$18
🤡	$70
🤡	$12
😈	$8

Pre-Aggregate

Customers

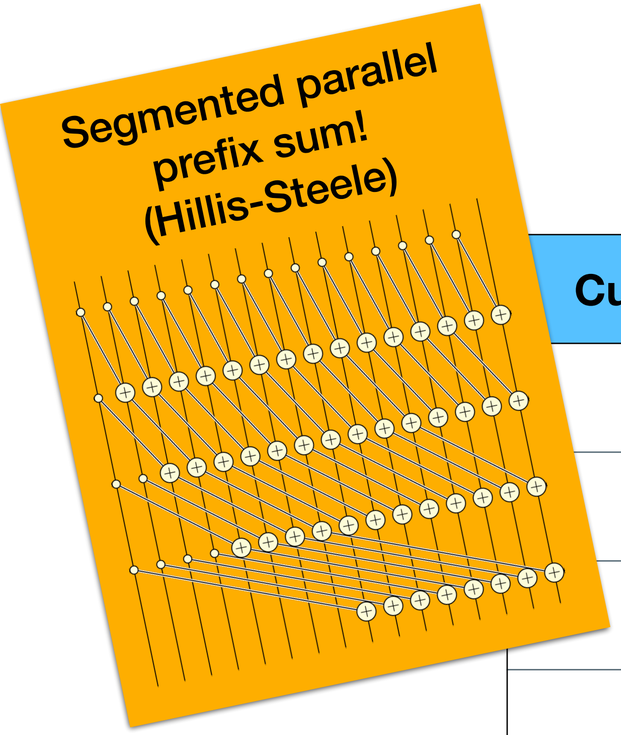
Customer	AcctBalance
🤡	$1000
😈	$2000
🤡	$500
🐸	$60
🥶	$8000
🥶	$700
🐸	-$200

Orders

Customer	OrdCost
🐸	$30
🤡	$20
🥶	$55
🥶	$18
🤡	$70
🤡	$12
😈	$8

After Pre-aggregation

One-to-many join



Customers

Customer	TotalAcctBalance
🤡	$1500
😈	$2000
🐸	-$140
🥶	$8700

unique keys!

pre-aggregated values

⋈

Orders

Customer	OrdCost
🐸	$30
🤡	$20
🥶	$55
🥶	$18
🤡	$70
🤡	$12
😈	$8

After Pre-aggregation

One-to-many join

Customers

Customer	TotalAcctBalance
🤡	$1500
😈	$2000
🐸	-$140
🥶	$8700

⋈

Orders

Customer	OrdCost
🐸	$30
🤡	$20
🥶	$55
🥶	$18
🤡	$70
🤡	$12
😈	$8

Concatenate

```
func join(L, R, keys):
    T := concat*(L, R)
    T.sort(keys)
    T.prefix_aggregate(…)
    return T
```

Customer	C.Total	O.Cost
🤡	$1500	
😈	$2000	
🐸	-$140	
🥶	$8700	
🐸		$30
🤡		$20
🥶		$55
🥶		$18
🤡		$70
🤡		$12
😈		$8

Sort

```
func join(L, R, keys):
    T := concat*(L, R)
    T.sort(keys)
    T.prefix_aggregate(…)
    return T
```

Customer	C.Balance	O.Cost
🤡	$1500	
🤡		$20
🤡		$70
🤡		$12
😈	$2000	
😈		$8
🐸	-$140	
🐸		$30
🥶	$8700	
🥶		$55
🥶		$18

Group Aggregate

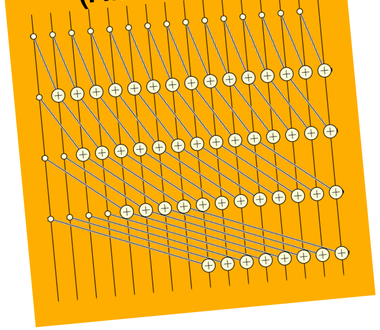
```
func join(L, R, keys):
    T := concat*(L, R)
    T.sort(keys)
    T.prefix_aggregate(…)
    return T
```

Customer	C.Balance	O.Cost	TotalCost
🤡	$1500	∅	
🤡	$1500	$20	
🤡	$1500	$70	
🤡	$1500	$12	$102
😈	$2000	∅	
😈	$2000	$8	$8
🐸	-$140	∅	
🐸	-$140	$30	$30
🥶	$8700	∅	
🥶	$8700	$55	
🥶	$8700	$18	$73

Group Aggregate

```
func join(L, R, keys):
    T := concat*(L, R)
    T.sort(keys)
    T.prefix_aggregate(…)
    return T
```

Segmented parallel prefix sum! (Hillis-Steele)



Customer	C.Balance	O.Cost	TotalCost
🤡	$1500	∅	
🤡	$1500	$20	
🤡	$1500	$70	
🤡	$1500	$12	$102
😈	$2000	∅	
😈	$2000	$8	$8
🐸	-$140	∅	
🐸	-$140	$30	$30
🥶	$8700	∅	
🥶	$8700	$55	
🥶	$8700	$18	$73

Mark Invalid Rows

```
func join(L, R, keys):
    T := concat*(L, R)
    T.sort(keys)
    T.prefix_aggregate(…)
    T.mark_invalid()
    return T
```

Customer	C.Balance	O.Cost	TotalCost
∅	∅	∅	∅
∅	∅	∅	∅
∅	∅	∅	∅
🤡	$1500	$12	$102
∅	∅	∅	∅
😈	$2000	$8	$8
∅	∅	∅	∅
🐸	-$140	$30	$30
∅	∅	∅	∅
∅	∅	∅	∅
🥶	$8700	$18	$73

Join Output

```
SELECT
    SPECIES, SUM(ACCTBALANCE) / SUM(ORDCOST)
FROM
    ORDERS, CUSTOMERS
WHERE
    ORDER.CUSTOMER = CUSTOMERS.CUSTOMER
GROUP BY
    SPECIES
```

Customer	C.Balance	TotalCost	Species
🤡	$1500	$102	Human
🥶	$8700	$73	Human
😈	$2000	$8	NonHuman
🐸	-$140	$30	NonHuman

Final Aggregation

```
SELECT
    SPECIES, SUM(ACCTBALANCE) / SUM(ORDCOST)
FROM
    ORDERS, CUSTOMERS
WHERE
    ORDER.CUSTOMER = CUSTOMERS.CUSTOMER
GROUP BY
    SPECIES
```

Species	Customer	C.Balance	TotalCost
Human	∅	∅	∅
Human	∅	$10,200	$175
NonHuman	∅	∅	∅
NonHuman	∅	$1,860	$38

Compute Ratio

```
SELECT
    SPECIES, SUM(ACCTBALANCE) / SUM(ORDCOST)
FROM
    ORDERS, CUSTOMERS
WHERE
    ORDER.CUSTOMER = CUSTOMERS.CUSTOMER
GROUP BY
    SPECIES
```

Species	C.Balance	TotalCost	Ratio
Human	$10,200	$174	**58**
NonHuman	$1,860	$38	**48**

Final Table

```
SELECT
    SPECIES, SUM(ACCTBALANCE) / SUM(ORDCOST)
FROM
    ORDERS, CUSTOMERS
WHERE
    ORDER.CUSTOMER = CUSTOMERS.CUSTOMER
GROUP BY
    SPECIES
```

Species	Ratio
Human	58
NonHuman	48

Final Table

```
SELECT
    SPECIES, SUM(ACCTBALANCE) / SUM(ORDCOST)
FROM
    ORDERS, CUSTOMERS
WHERE
    ORDER.CUSTOMER = CUSTOMERS.CUSTOMER
GROUP BY
    SPECIES
```

Still secret shared! More joins…

Species	Ratio
Human	58
NonHuman	48

Final Table

```
SELECT
    SPECIES, SUM(ACCTBALANCE) / SUM(ORDCOST)
FROM
    ORDERS, CUSTOMERS
WHERE
    ORDER.CUSTOMER = CUSTOMERS.CUSTOMER
GROUP BY
    SPECIES
```

Still secret shared! More joins…

Species	Ratio
Human	58
NonHuman	48

Oblivious output bound:
|Orders|

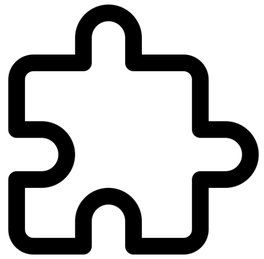
ORQ: an Oblivious Relational Query engine

ORQ: an Oblivious Relational Query engine

Familiar APIs &
simple orchestration

ORQ: an Oblivious Relational Query engine

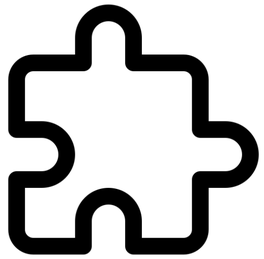
Familiar APIs &
simple orchestration



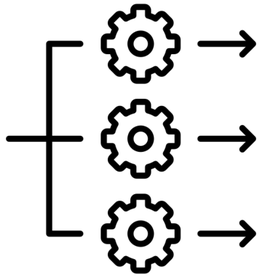
Support for multiple
protocols & threat models

ORQ: an Oblivious Relational Query engine

Familiar APIs &
simple orchestration



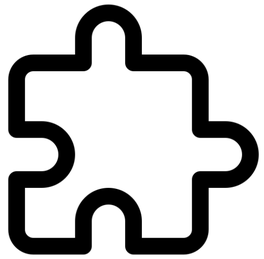
Support for multiple
protocols & threat models



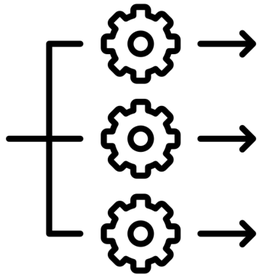
Data-parallel runtime

ORQ: an Oblivious Relational Query engine

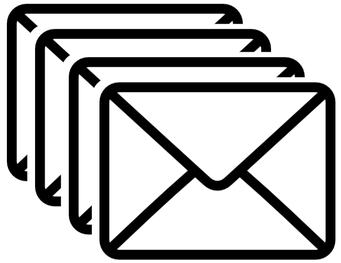
Familiar APIs &
simple orchestration



Support for multiple
protocols & threat models



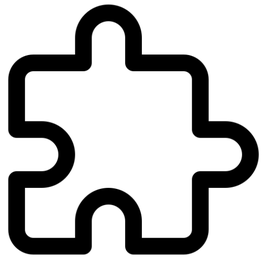
Data-parallel runtime



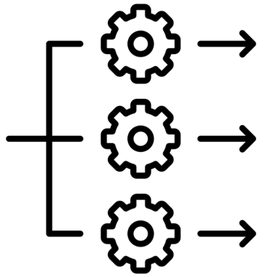
Vectorization &
message batching

ORQ: an Oblivious Relational Query engine

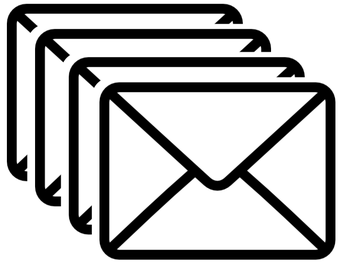
Familiar APIs &
simple orchestration



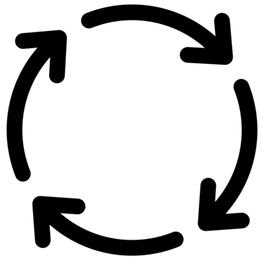
Support for multiple
protocols & threat models



Data-parallel runtime



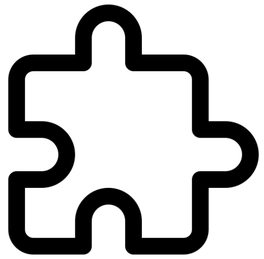
Vectorization &
message batching



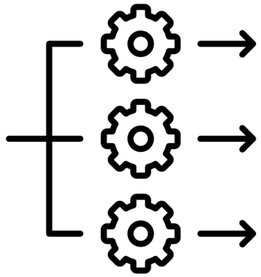
Custom TCP communicator

ORQ: an Oblivious Relational Query engine

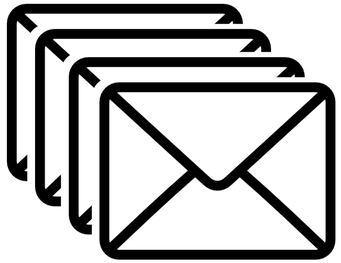
Familiar APIs &
simple orchestration



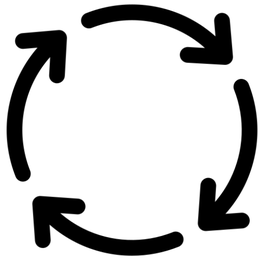
Support for multiple
protocols & threat models



Data-parallel runtime



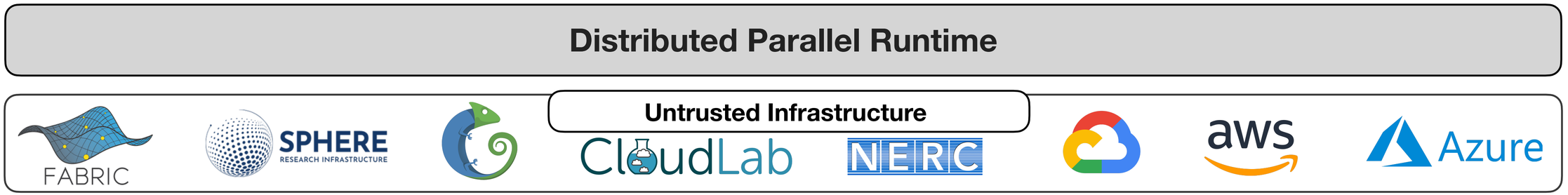
Vectorization &
message batching



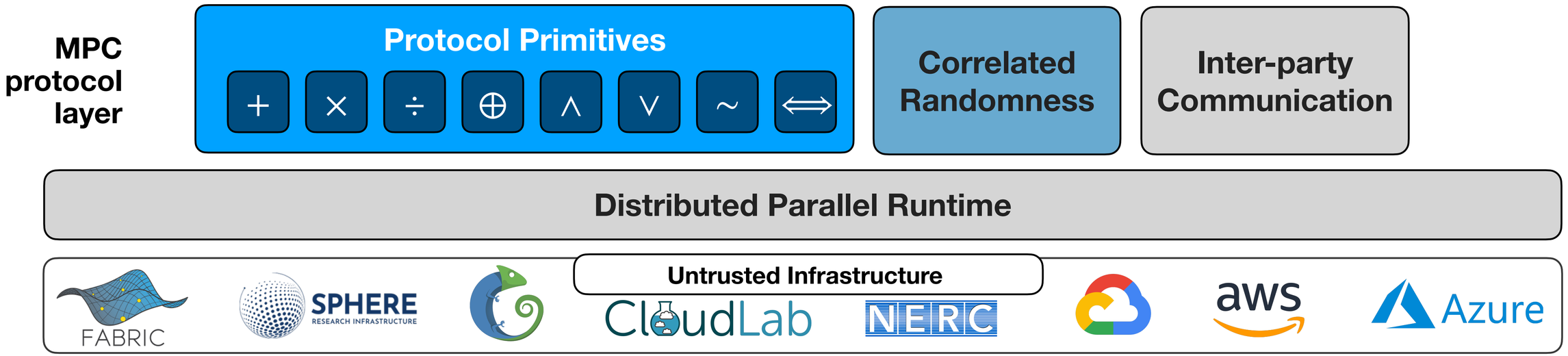
Custom TCP communicator

14,000 C++ LoC

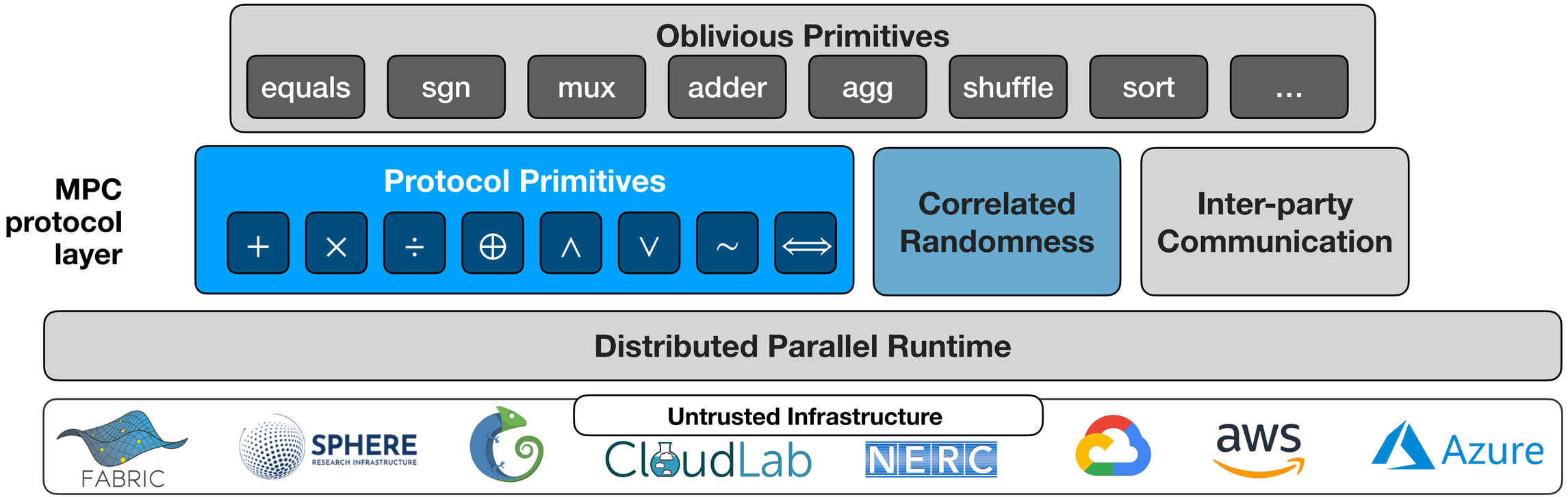
ORQ Stack


Distributed Parallel Runtime
Untrusted Infrastructure
FABRIC
SPHERE
RESEARCH INFRASTRUCTURE
CloudLab
NERC
aws
Azure

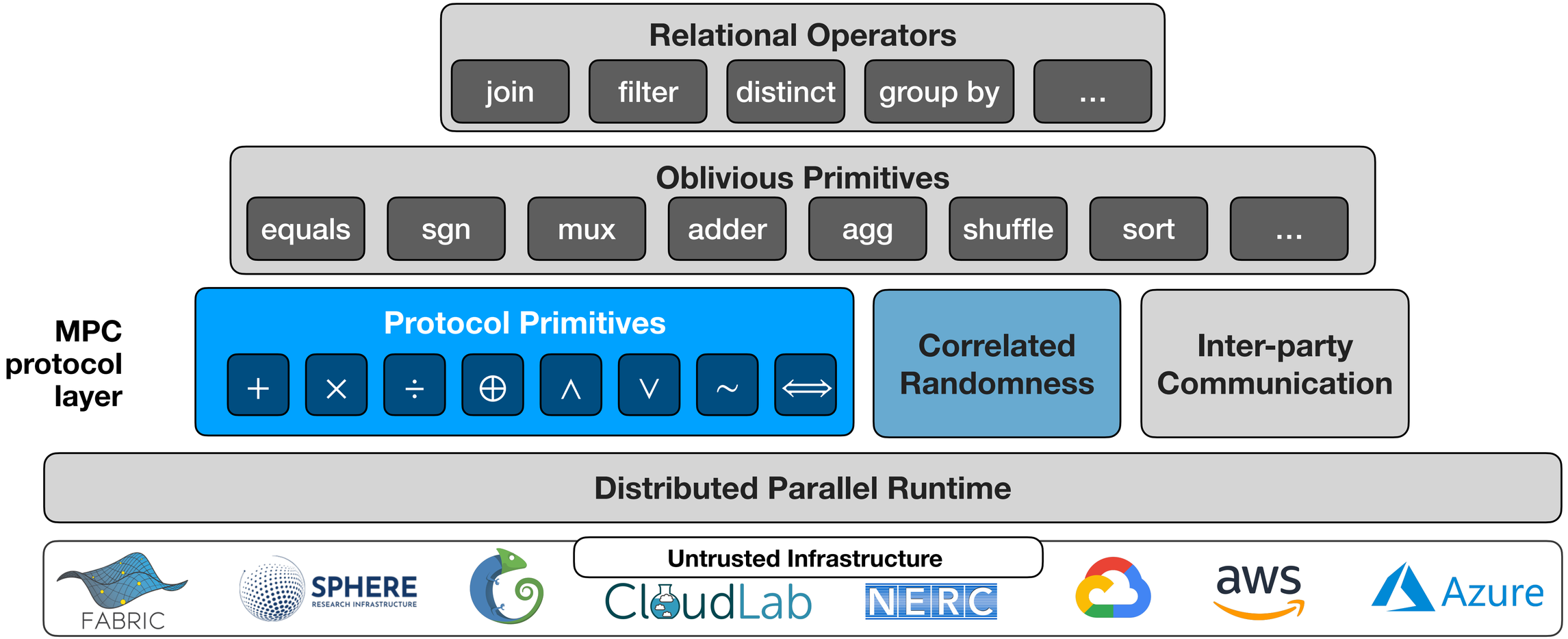
ORQ Stack


MPC protocol layer
Protocol Primitives
+
×
÷
⊕
∧
∨
~
⟺
Correlated Randomness
Inter-party Communication
Distributed Parallel Runtime
FABRIC
SPHERE
RESEARCH INFRASTRUCTURE
Untrusted Infrastructure
CloudLab
NERC
aws
Azure

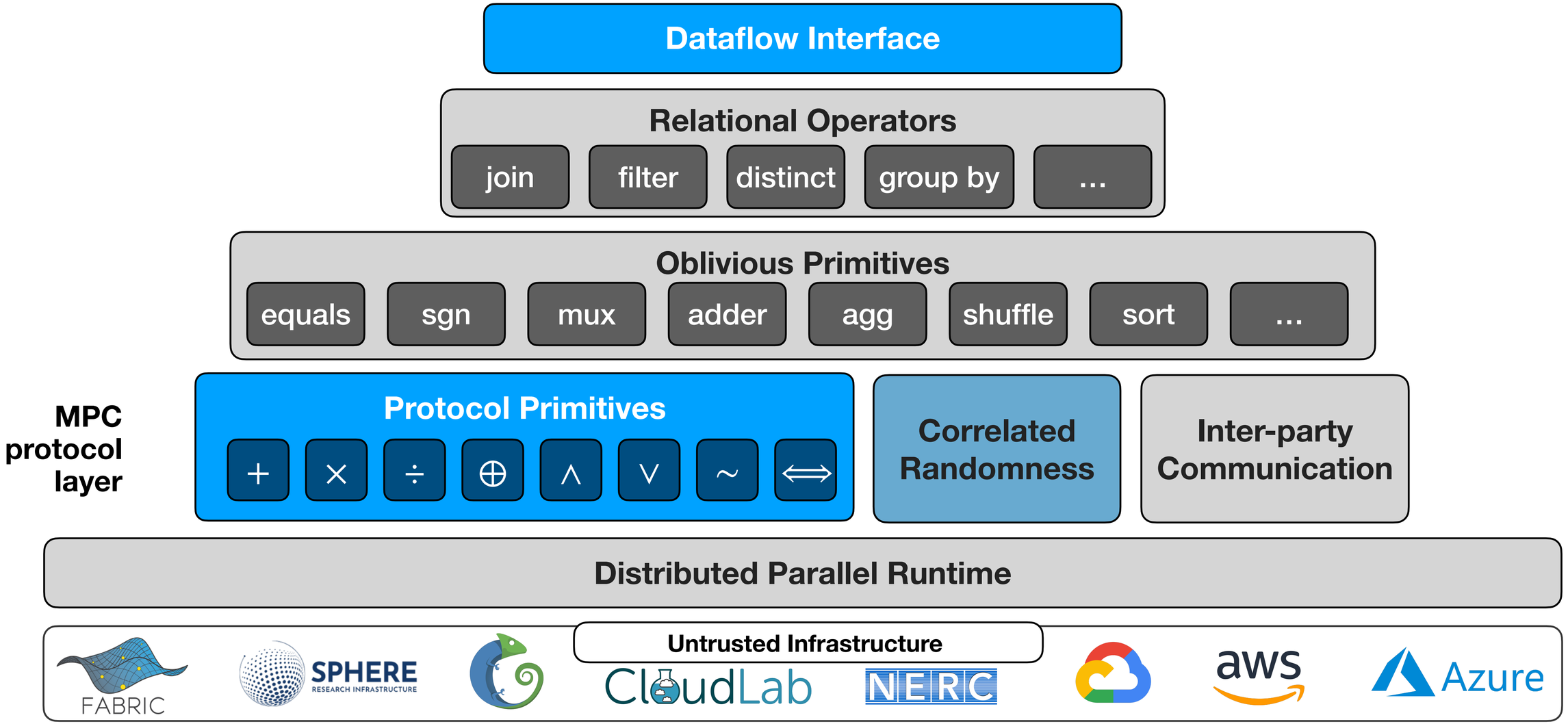
ORQ Stack


Oblivious Primitives
equals
sgn
mux
adder
agg
shuffle
sort
...
MPC protocol layer
Protocol Primitives
+
×
÷
⊕
∧
∨
~
⟺
Correlated Randomness
Inter-party Communication
Distributed Parallel Runtime
Untrusted Infrastructure
FABRIC
SPHERE
RESEARCH INFRASTRUCTURE
CloudLab
NERC
aws
Azure

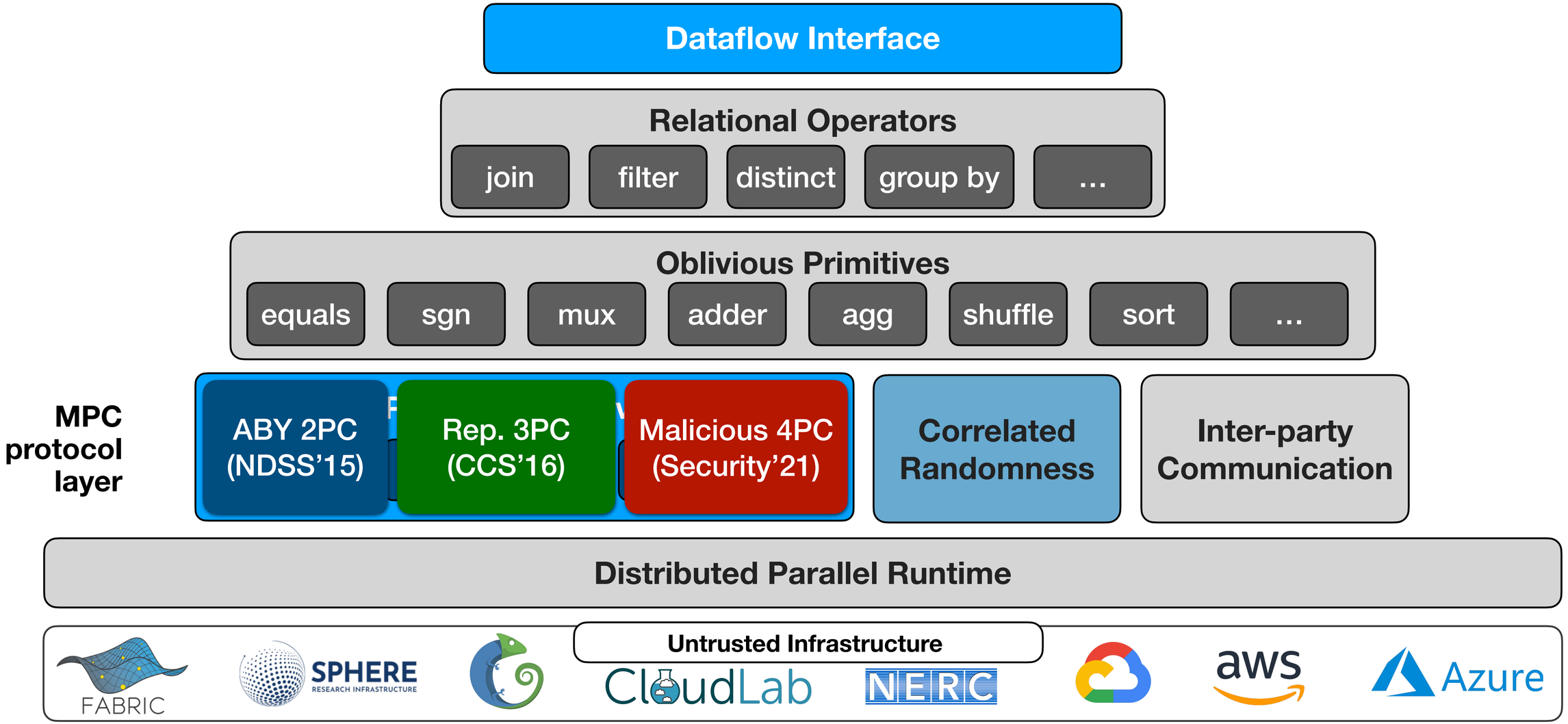
ORQ Stack


Relational Operators
join
filter
distinct
group by
...
Oblivious Primitives
equals
sgn
mux
adder
agg
shuffle
sort
...
MPC protocol layer
Protocol Primitives
+
×
÷
⊕
∧
∨
~
⟺
Correlated Randomness
Inter-party Communication
Distributed Parallel Runtime
Untrusted Infrastructure
FABRIC
SPHERE
RESEARCH INFRASTRUCTURE
CloudLab
NERC
aws
Azure

ORQ Stack


Dataflow Interface
Relational Operators
join
filter
distinct
group by
...
Oblivious Primitives
equals
sgn
mux
adder
agg
shuffle
sort
...
MPC protocol layer
Protocol Primitives
+
×
÷
⊕
∧
∨
~
⟺
Correlated Randomness
Inter-party Communication
Distributed Parallel Runtime
Untrusted Infrastructure
FABRIC
SPHERE
RESEARCH INFRASTRUCTURE
CloudLab
NERC
aws
Azure

ORQ Stack


Dataflow Interface
Relational Operators
join
filter
distinct
group by
...
Oblivious Primitives
equals
sgn
mux
adder
agg
shuffle
sort
...
MPC protocol layer
ABY 2PC (NDSS'15)
Rep. 3PC (CCS'16)
Malicious 4PC (Security'21)
Correlated Randomness
Inter-party Communication
Distributed Parallel Runtime
Untrusted Infrastructure
FABRIC
SPHERE
RESEARCH INFRASTRUCTURE
CloudLab
NERC
aws
Azure

API Example

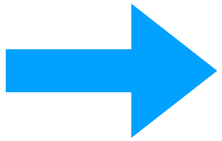
TPC-H Q13: SQL vs. ORQ Dataflow API

```
select
    c_count, count(*) as custdist
from (
    select
        c_custkey,
        count(o_orderkey)
    from
        customer left outer join orders on
            c_custkey = o_custkey
            and o_comment != 0
    group by
        c_custkey
    ) as c_orders (c_custkey, c_count)
group by
    c_count
order by
    custdist desc,
    c_count desc;
```

API Example

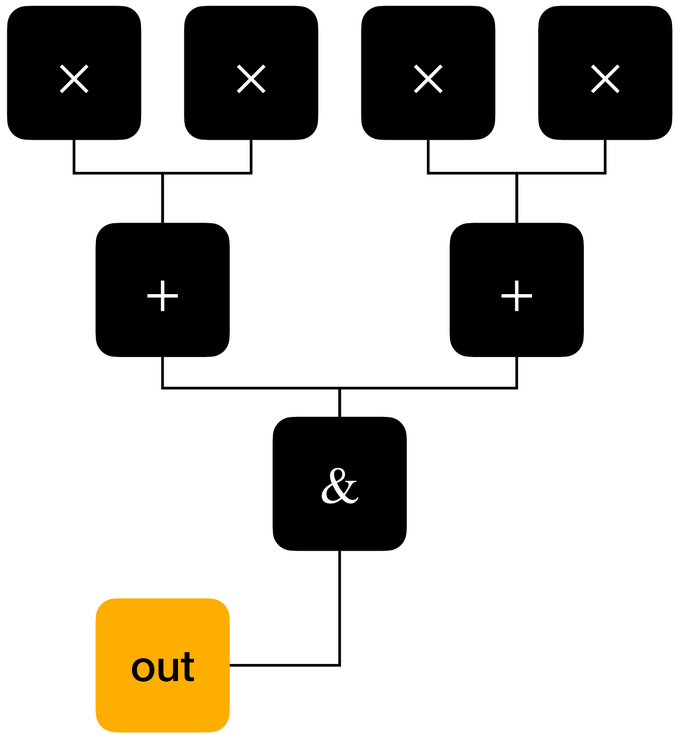
TPC-H Q13: SQL vs. ORQ Dataflow API

```
select
    c_count, count(*) as custdist
from (
    select
        c_custkey,
        count(o_orderkey)
    from
        customer left outer join orders on
            c_custkey = o_custkey
            and o_comment != 0
    group by
        c_custkey
    ) as c_orders (c_custkey, c_count)
group by
    c_count
order by
    custdist desc,
    c_count desc;
```

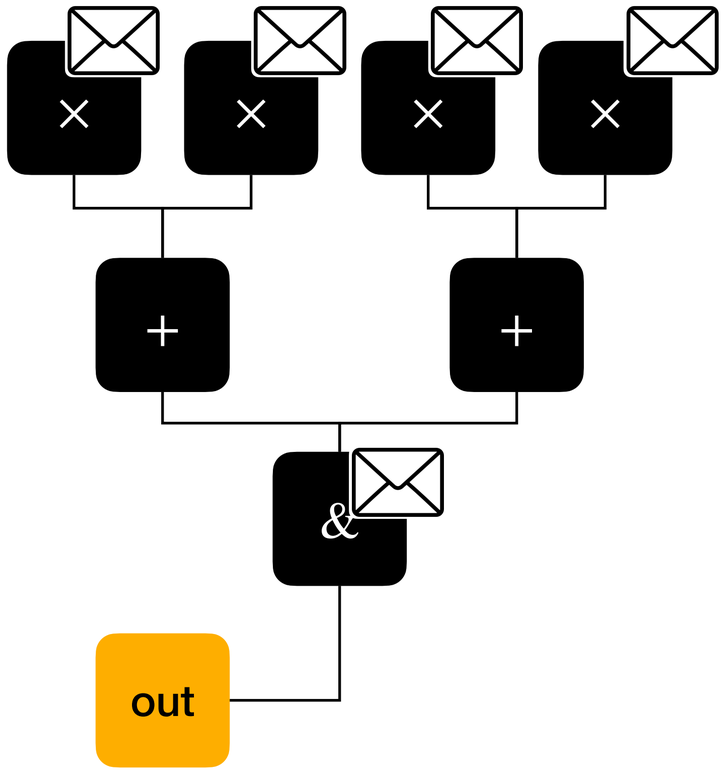


```
auto C = db.getCustomersTable();
auto O = db.getOrdersTable();

O.filter(O["[Comment]"] != 0);

auto T = C.left_outer_join(O,
    {"[CustKey]"},
    {{"Count", "Count", count}});

T.convert_a2b("Count", “[Count]");

auto F = T.aggregate(
    {"[Count]"},
    {{"CustDist", "CustDist", count}});

F.convert_a2b("CustDist", "[CustDist]");
F.sort({"[CustDist]", "[Count]"}, DESC);
```

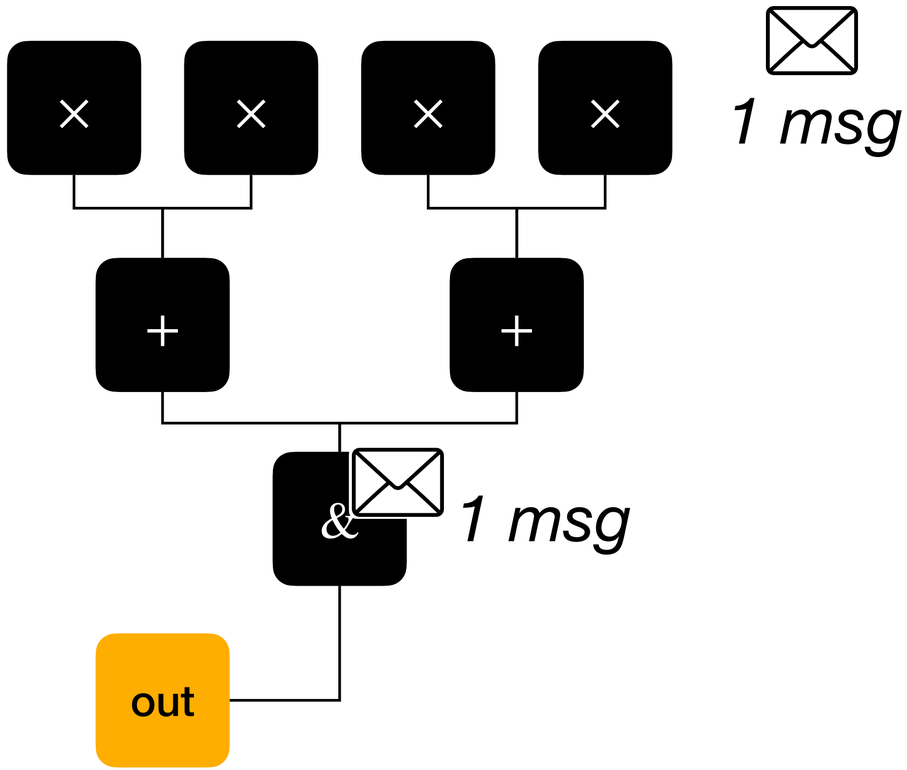
Vectorization & Message Batching



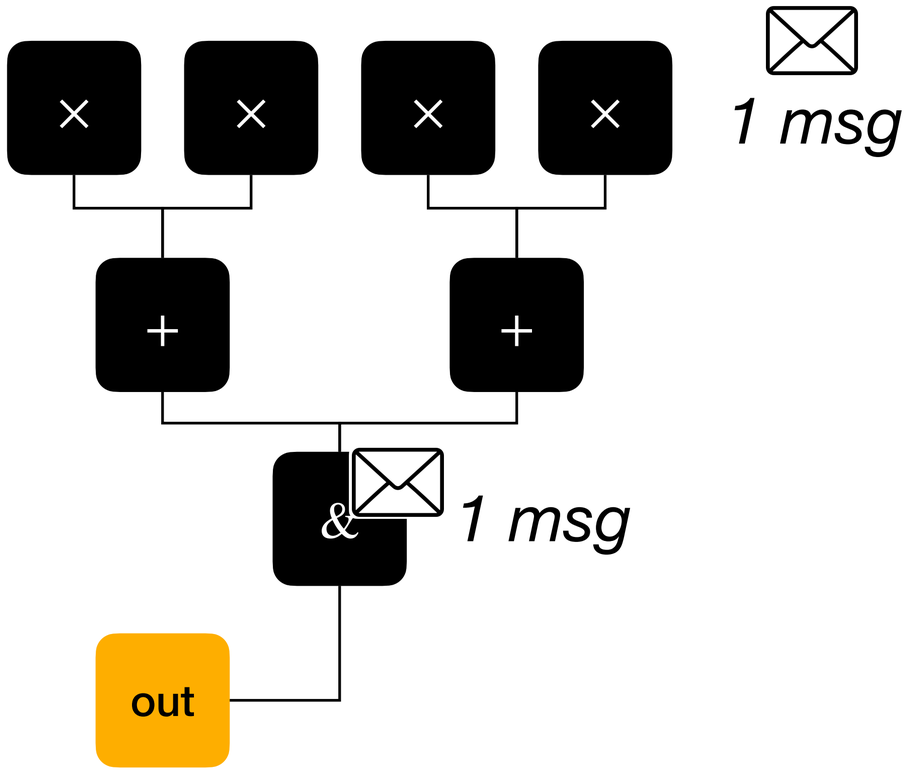
Vectorization & Message Batching

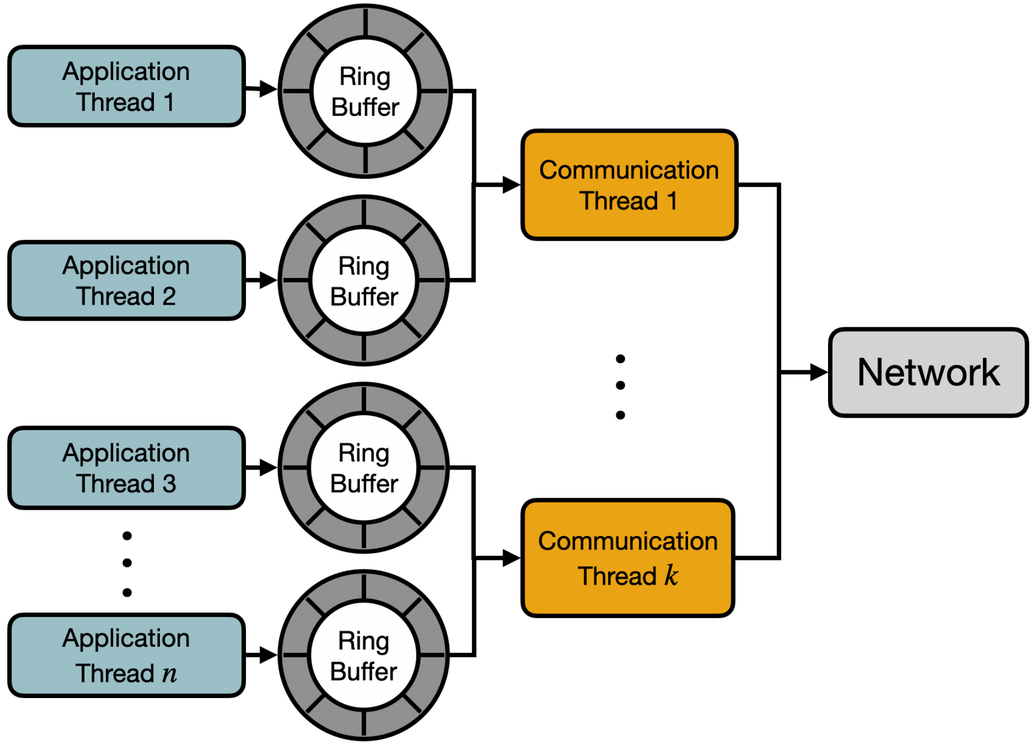


Vectorization & Message Batching



Vectorization & Message Batching





Flexible TCP Communicator

Prior MPC Benchmarks

with ~5M input rows

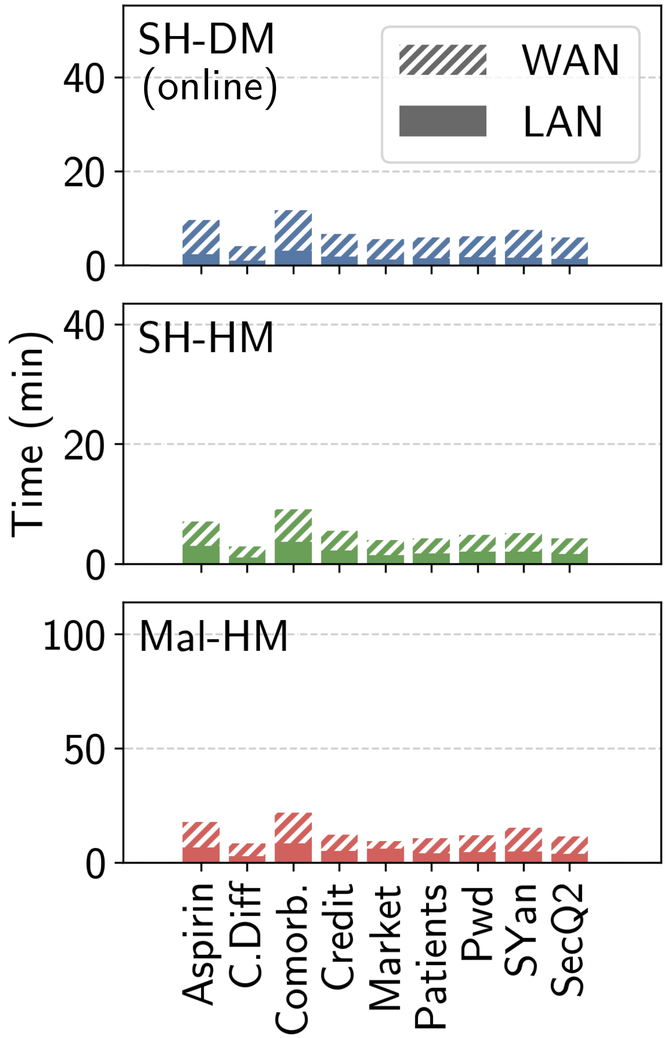
Prior MPC Benchmarks

with ~5M input rows

ABY 2PC
Semihonest / Dishonest Majority

Replicated 3PC
Semihonest / Honest Majority

Fantastic 4PC
Malicious / Honest Majority



all ≤ 3 min

all ≤ 4 min

all ≤ 9 min

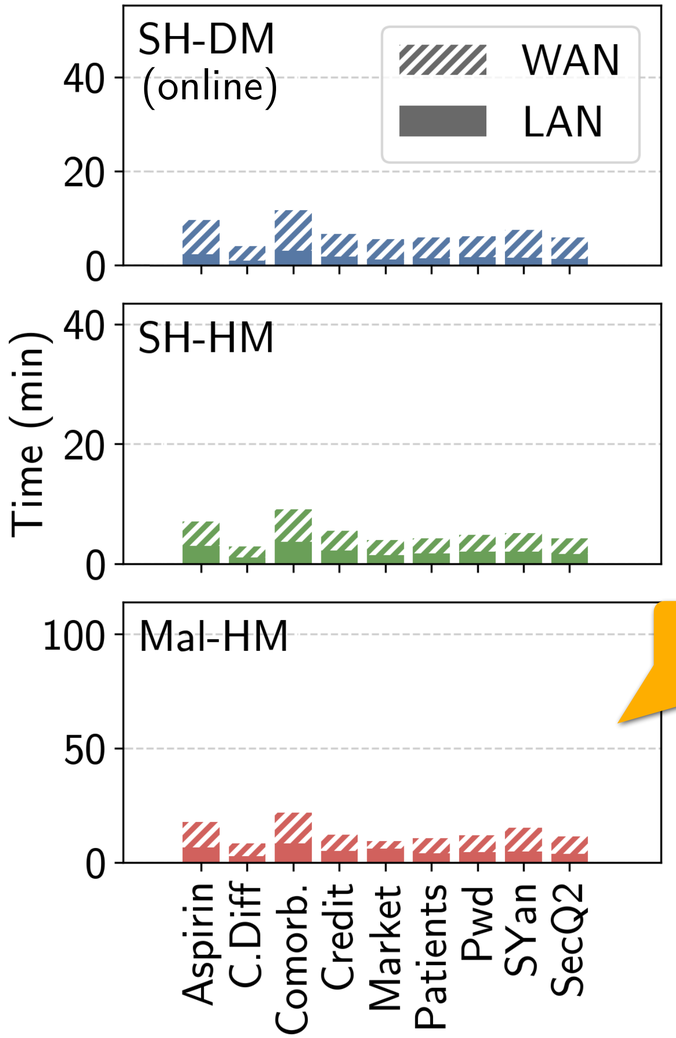
Prior MPC Benchmarks

with ~5M input rows

ABY 2PC
Semihonest / Dishonest Majority

Replicated 3PC
Semihonest / Honest Majority

Fantastic 4PC
Malicious / Honest Majority



all ≤ 3 min

all ≤ 4 min

all ≤ 9 min

Motivation for leakage!

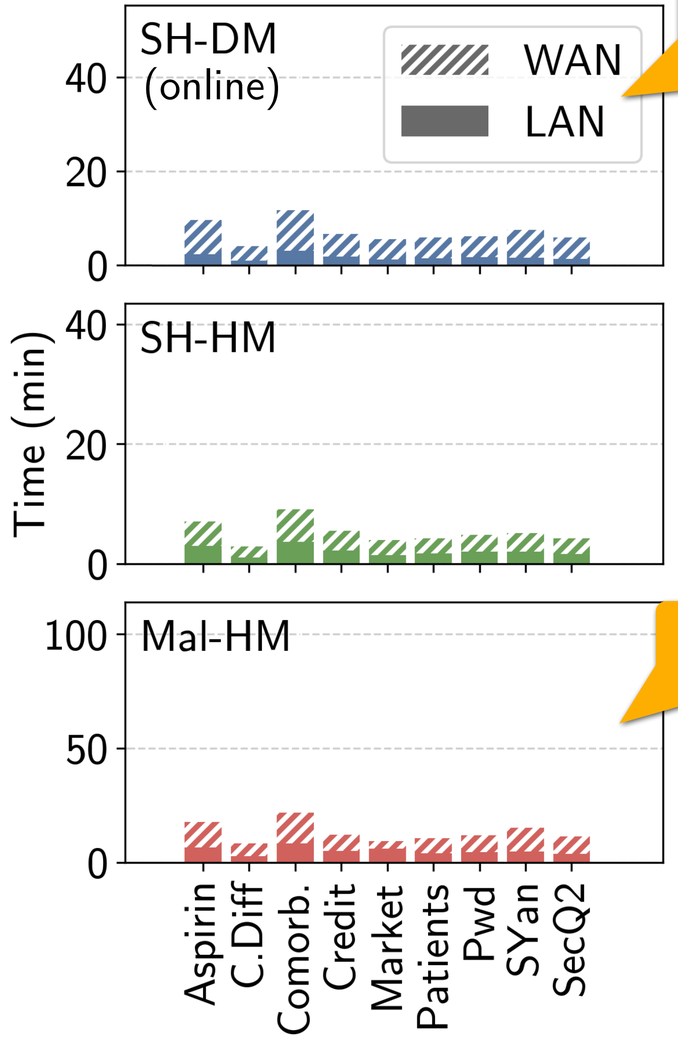
Prior MPC Benchmarks

with ~5M input rows

ABY 2PC
Semihonest / Dishonest Majority

Replicated 3PC
Semihonest / Honest Majority

Fantastic 4PC
Malicious / Honest Majority



6 Gbps 20 ms
25 Gbps 0.3 ms

all ≤ 3 min

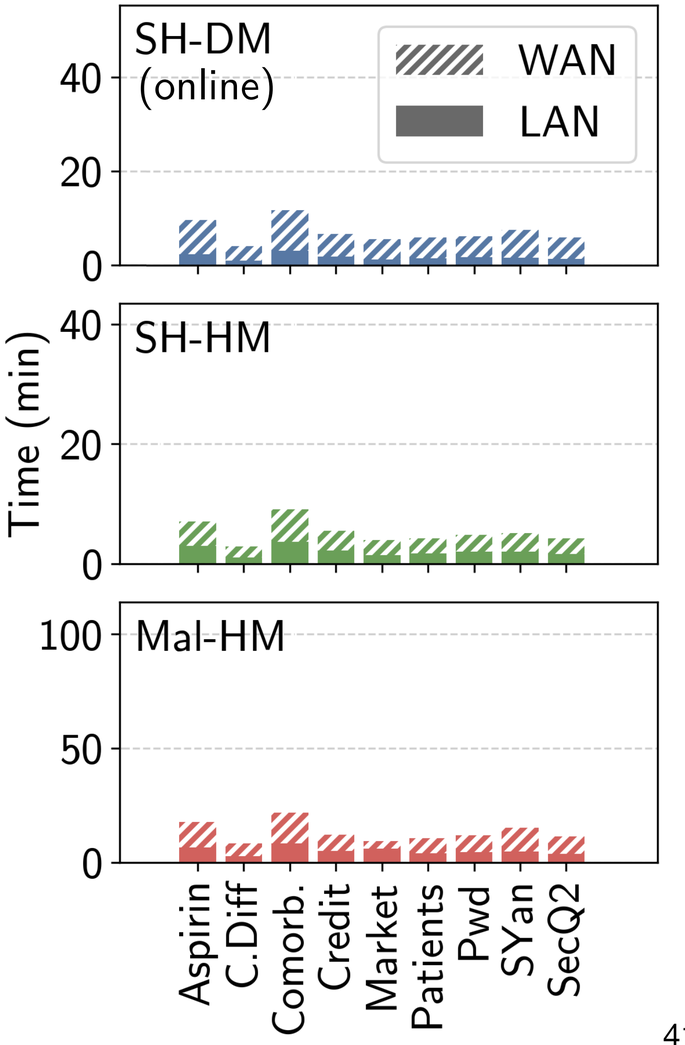
all ≤ 4 min

Motivation for leakage!

all ≤ 9 min

Prior MPC Benchmarks

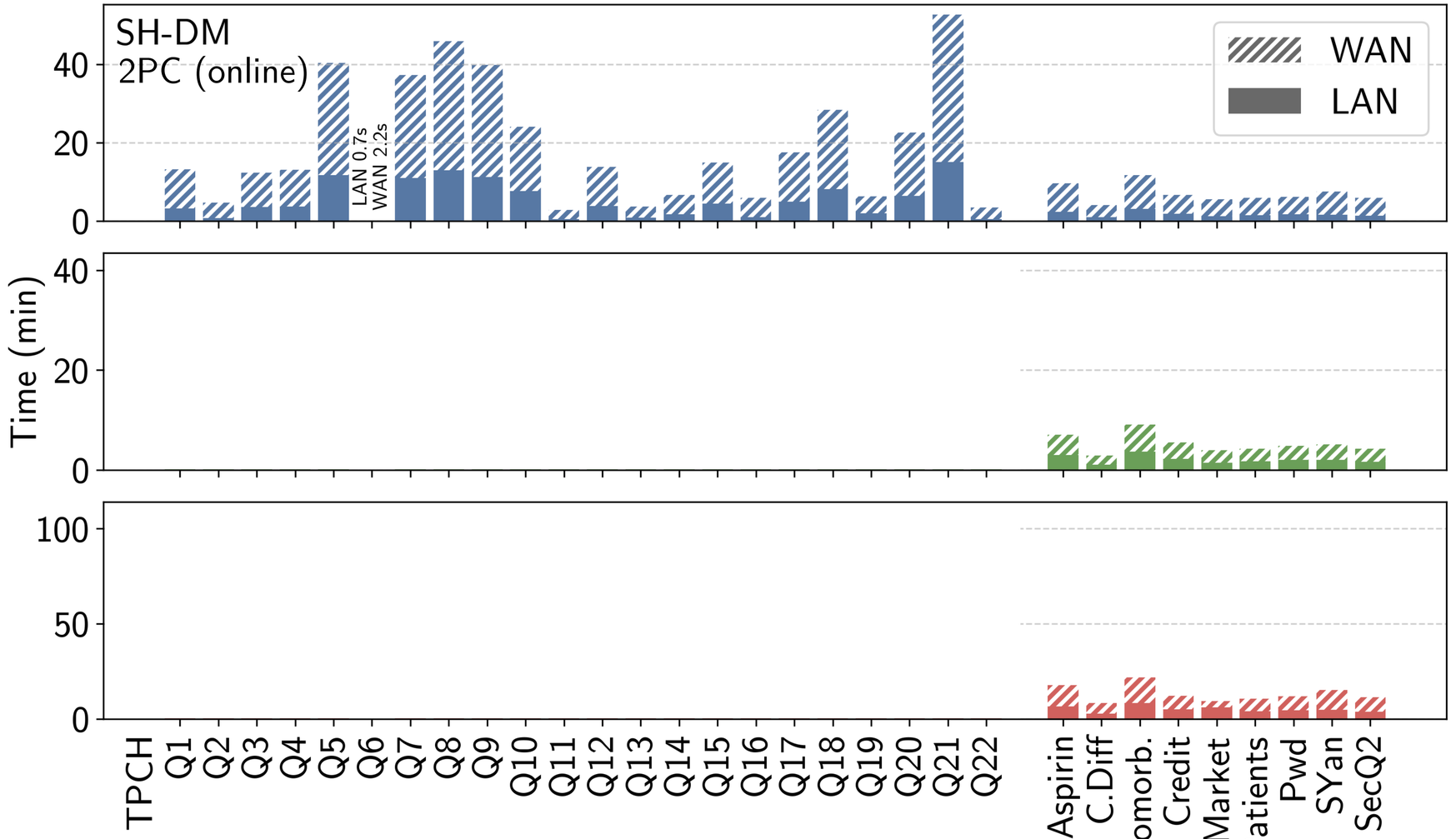
with ~5M input rows



Full TPC-H Benchmark under MPC

@ Scale Factor 1 (≈ 1 GB input)

	TPC-H	
minutes	Median	Max
SH-DM LAN	3.8	15.0
WAN	13.5	52.7

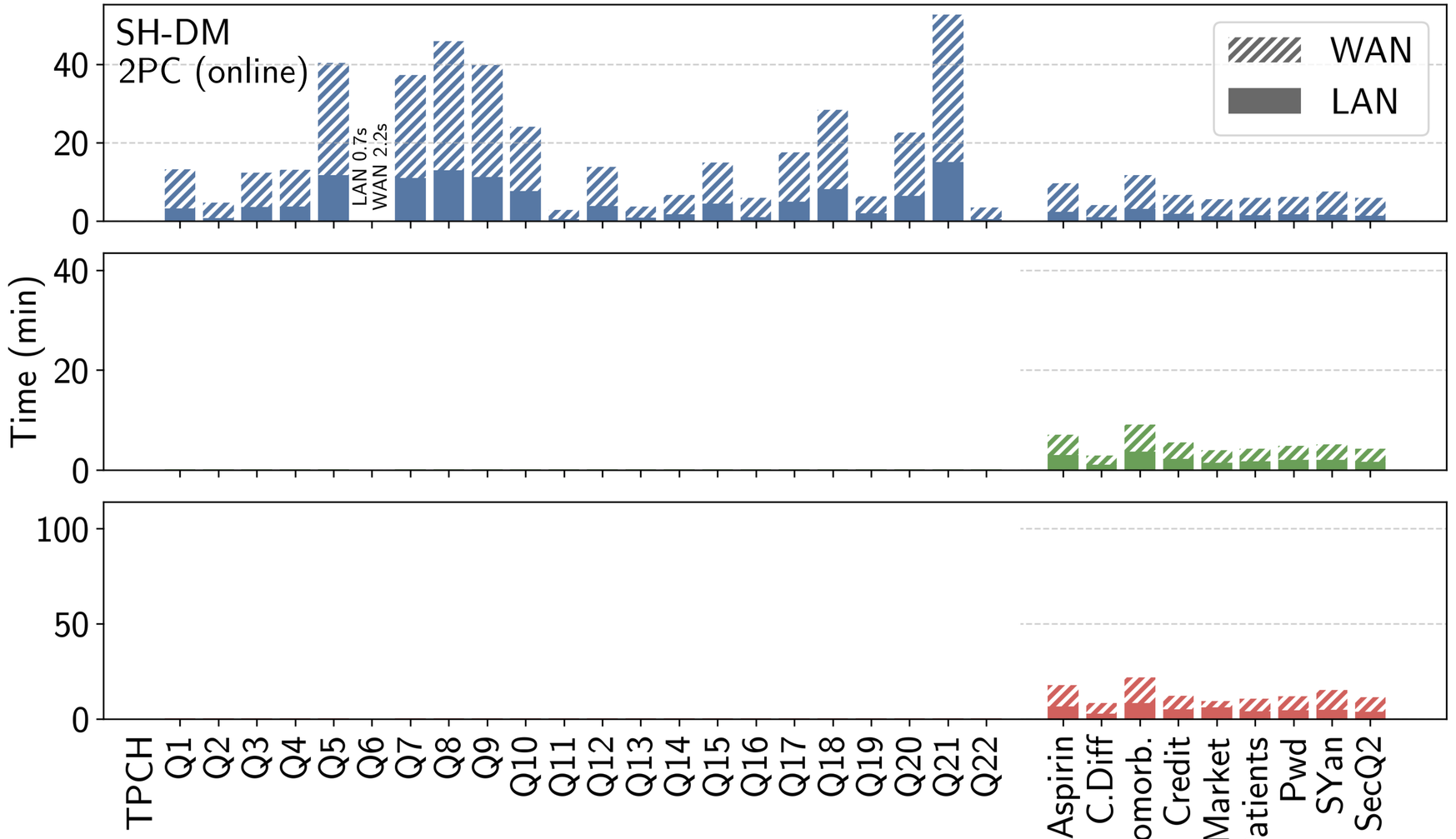


Machine type: AWS c7a.16xlarge

Full TPC-H Benchmark under MPC

@ Scale Factor 1 (≈ 1 GB input)

	TPC-H	
minutes	Median	Max
SH-DM LAN	3.8	15.0
WAN	13.5	52.7

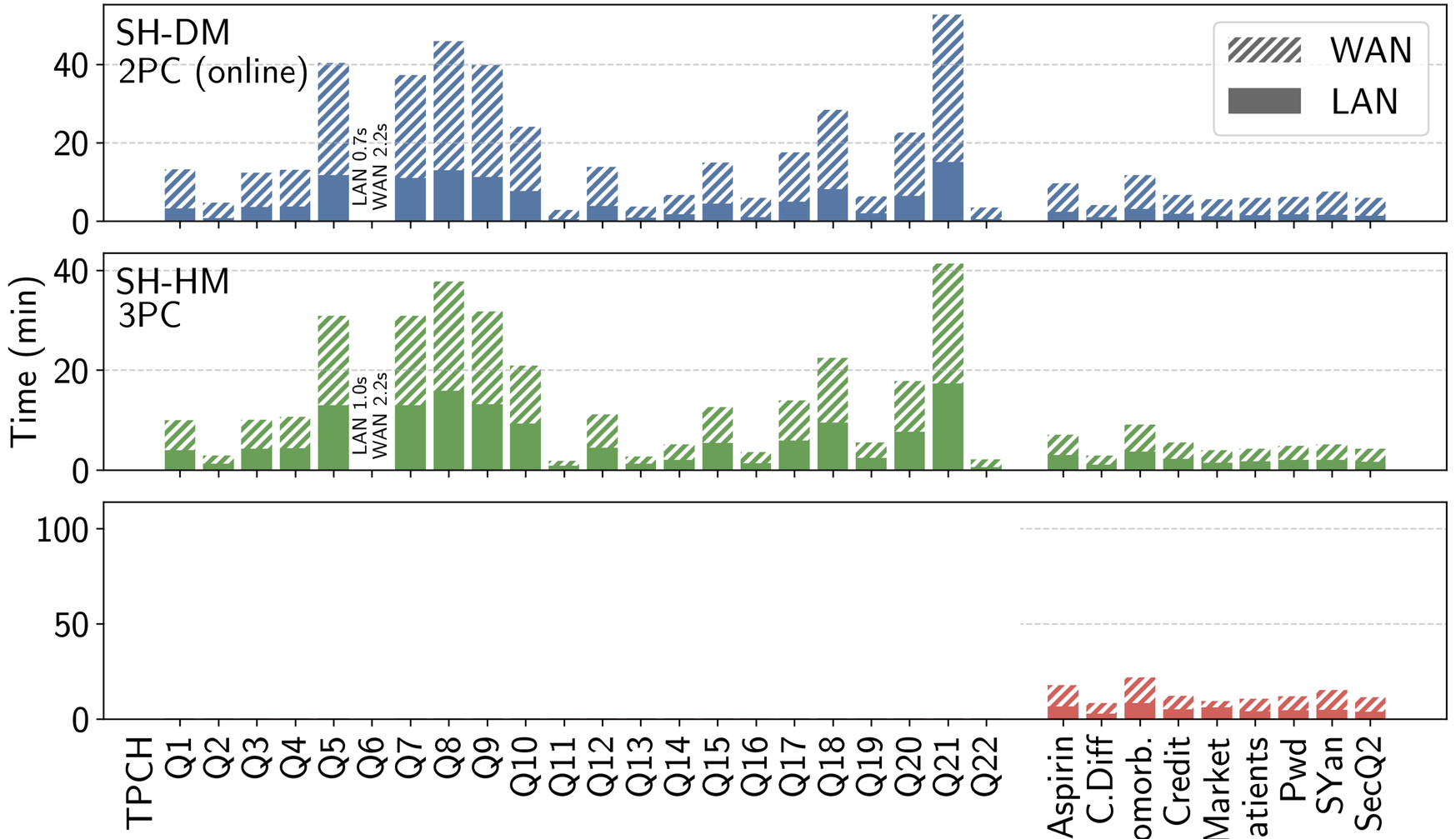


Machine type: AWS c7a.16xlarge

Full TPC-H Benchmark under MPC

@ Scale Factor 1 (≈ 1 GB input)

	TPC-H	
minutes	Median	Max
SH-DM LAN	3.8	15.0
WAN	13.5	52.7
SH-HM LAN	4.4	17.4
WAN	10.9	41.4

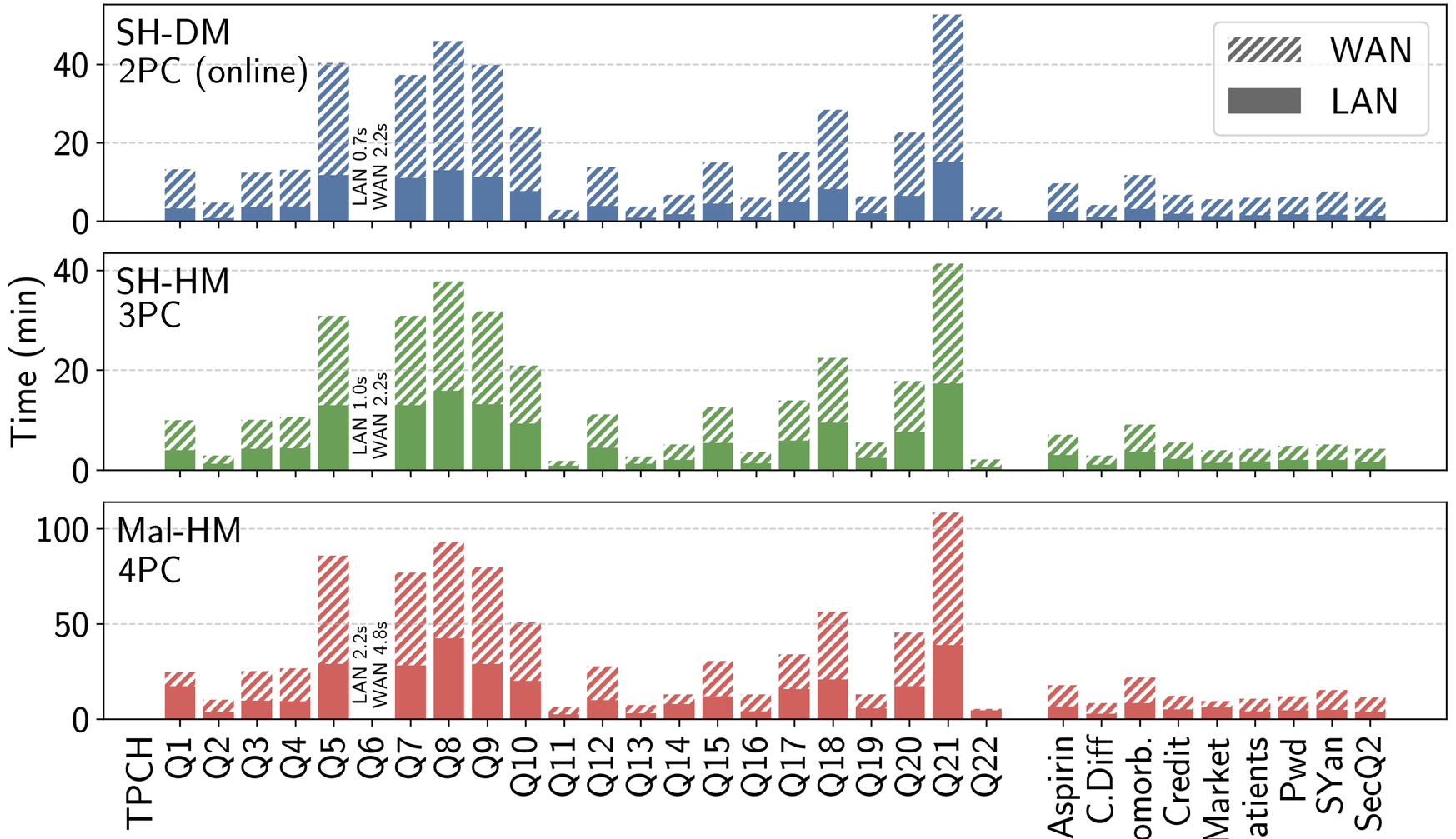


Machine type: AWS c7a.16xlarge

Full TPC-H Benchmark under MPC

@ Scale Factor 1 (≈ 1 GB input)

	TPC-H	
minutes	Median	Max
SH-DM LAN	3.8	15.0
WAN	13.5	52.7
SH-HM LAN	4.4	17.4
WAN	10.9	41.4
Mal-HM LAN	10.9	42.3
WAN	27.1	108.4

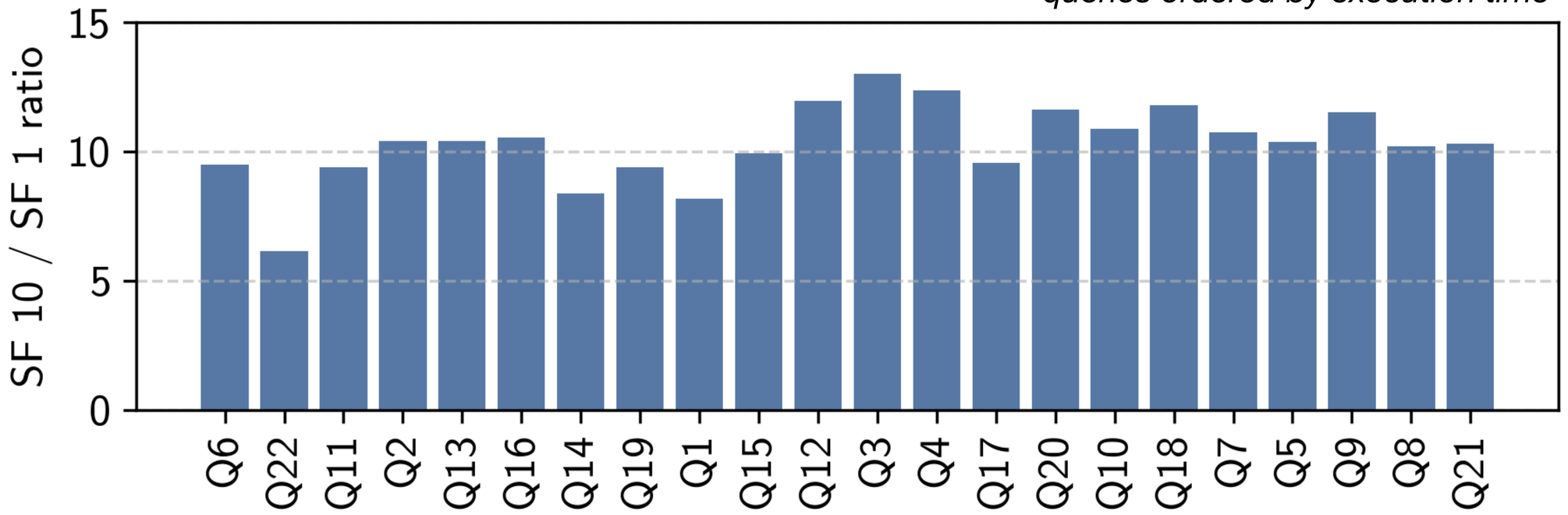


Machine type: AWS c7a.16xlarge

Scalability

10x larger inputs (SF 10)

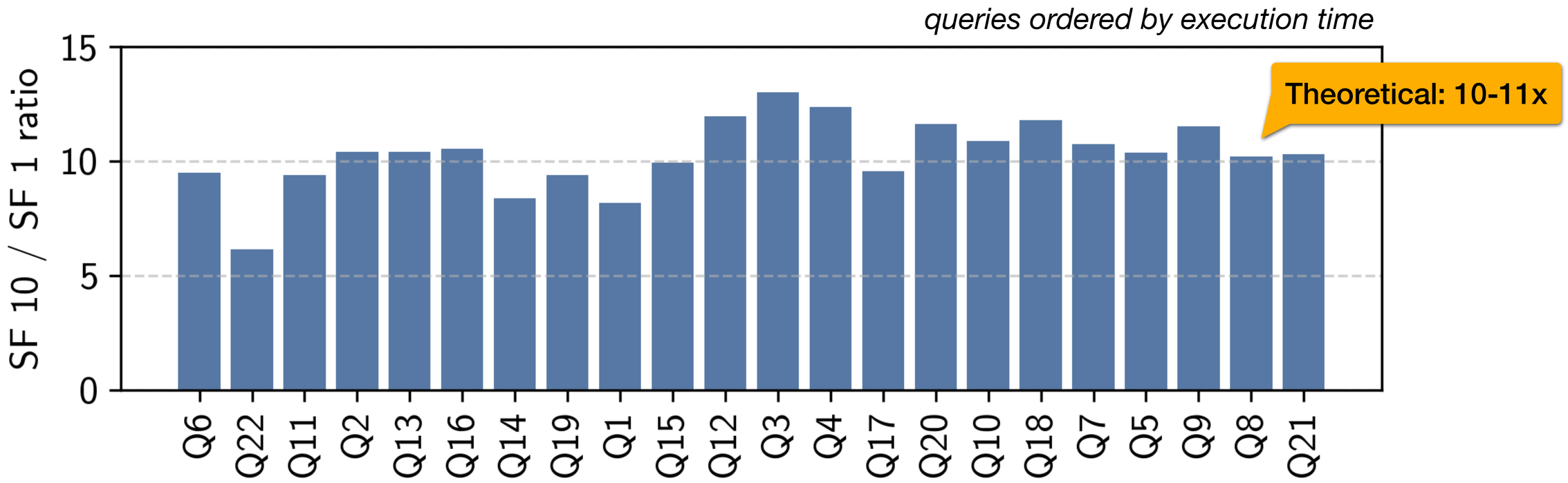
queries ordered by execution time



Machine type: AWS c7a.16xlarge

Scalability

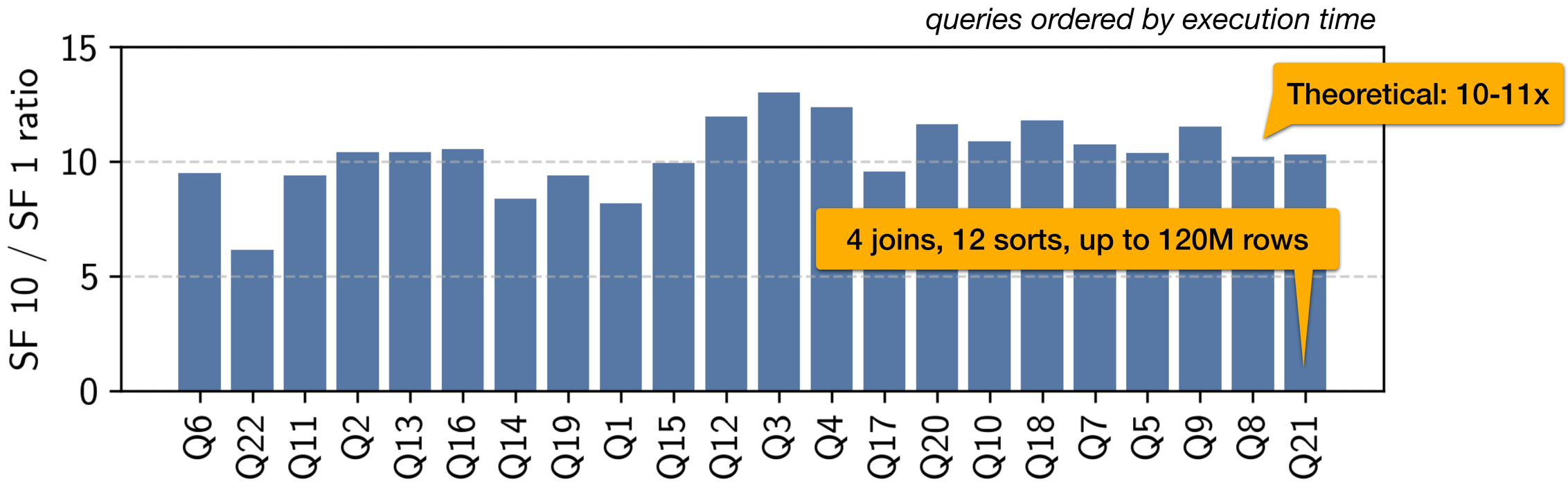
10x larger inputs (SF 10)



Machine type: AWS c7a.16xlarge

Scalability

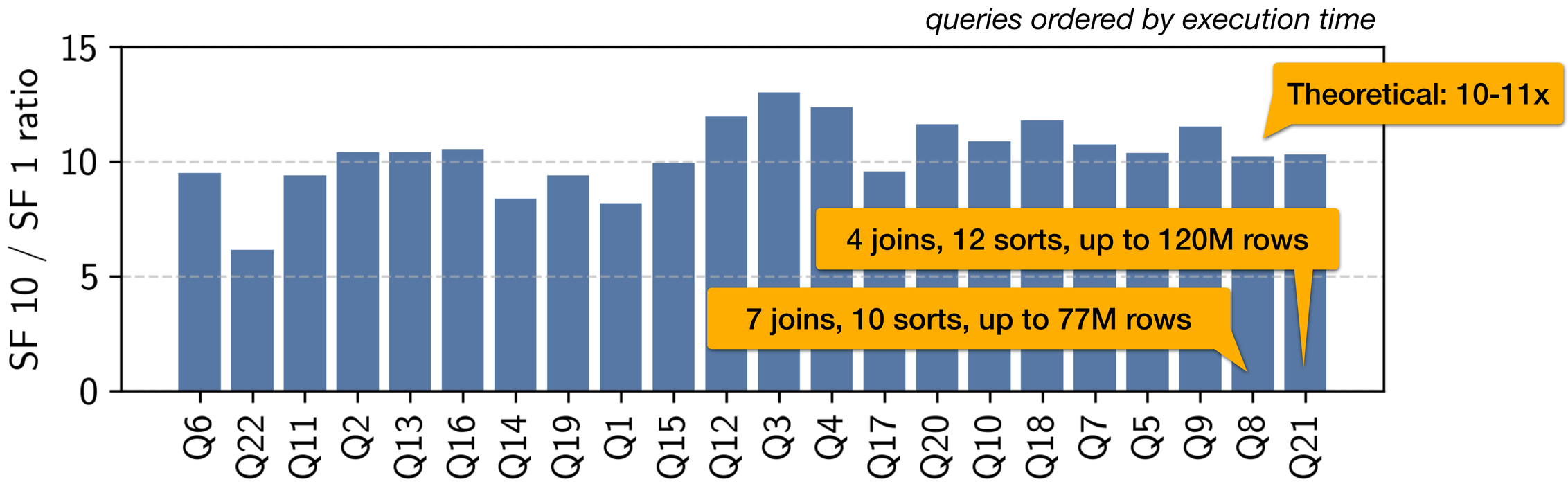
10x larger inputs (SF 10)



Machine type: AWS c7a.16xlarge

Scalability

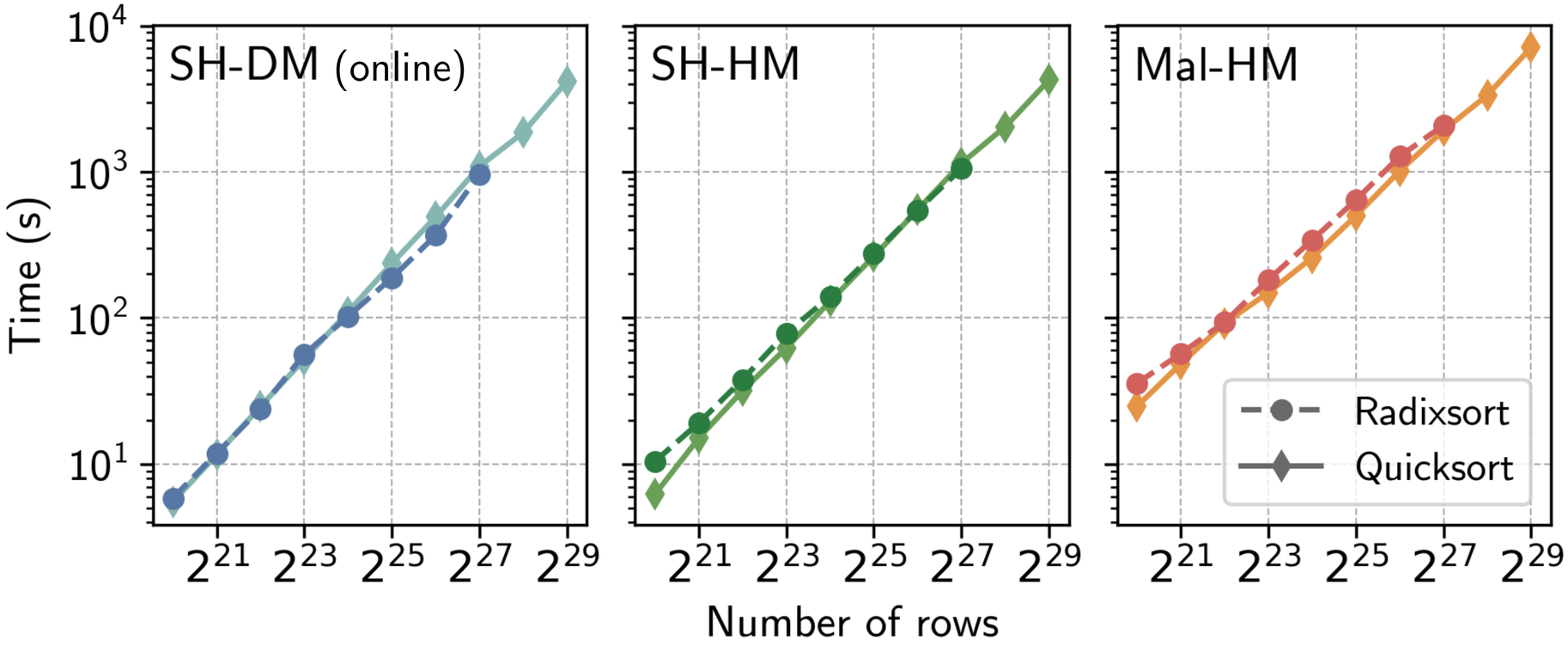
10x larger inputs (SF 10)



Machine type: AWS c7a.16xlarge

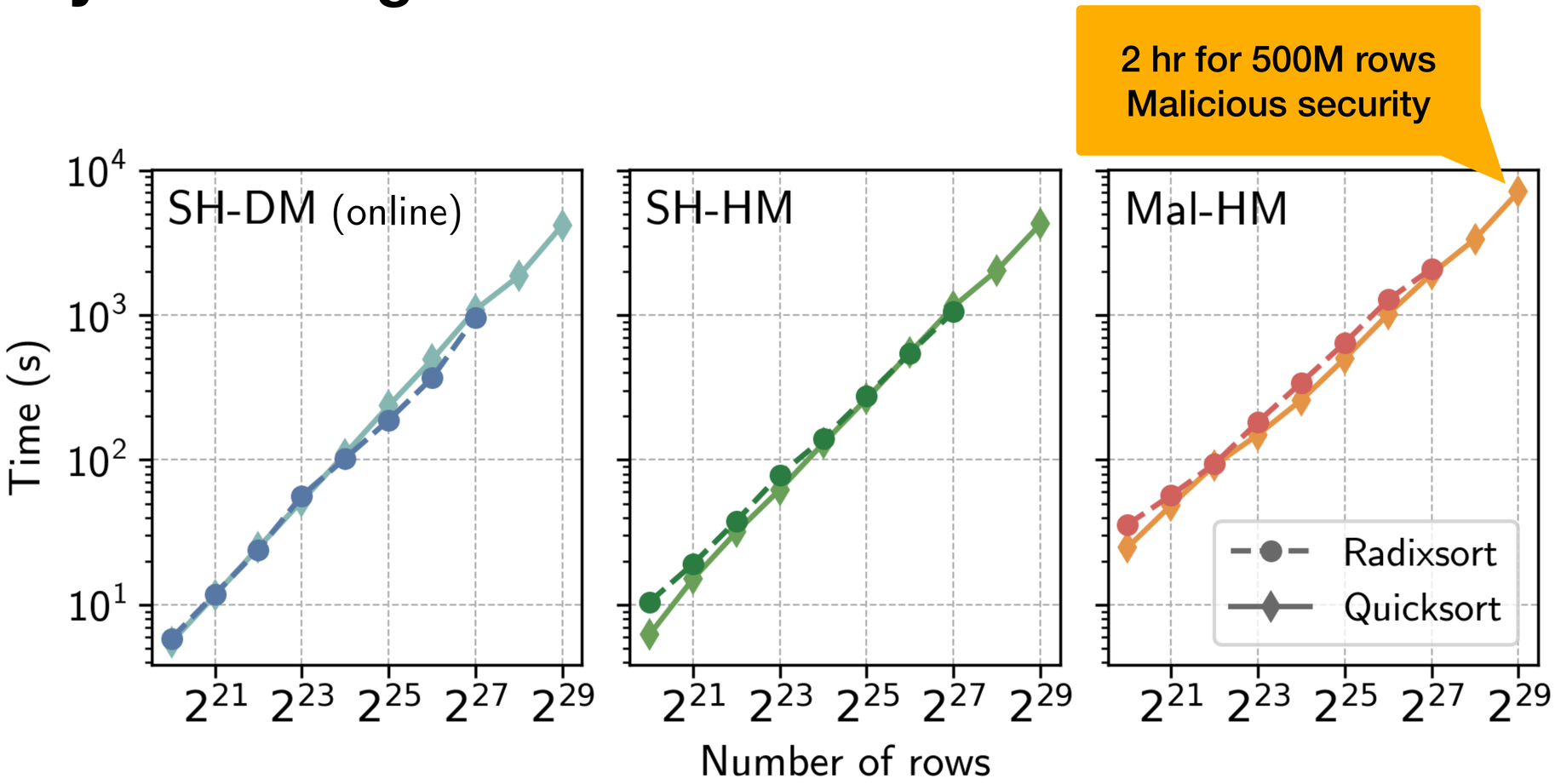
Going Even Further?

Scalability of Sorting



Going Even Further?

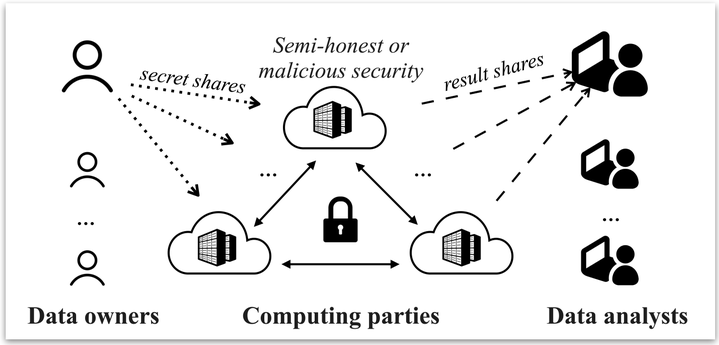
Scalability of Sorting



ORQ: Complex Analytics on Private Data with Strong Security Guarantees

email me: elibaum@bu.edu

Distinguished Artifact



Paper/Code/Docs

elibaum.com

Privacy

Runs on untrusted compute with end-to-end oblivious execution

Generality

Uses crypto protocols as a blackbox to meet different security requirements

Expressivity

Efficiently computes all queries from prior work & TPC-H, with no leakage

Scalability

Evaluates queries with millions of rows & efficiently uses available resources

CASPLab

We're hiring!

icons from flaticon.com