SECURE MPC FROM A SYSTEMS PERSPECTIVE

A HANDS-ON TUTORIAL WITH ORQ

John Liagouris
liagos@bu.edu

Vasiliki Kalavri
vkalavri@bu.edu

Eli Baum
elibaum@bu.edu

sites.bu.edu/casp

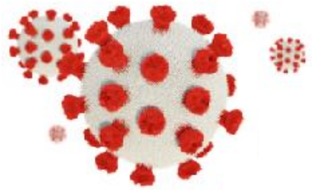
TUTORIAL OVERVIEW

- **Part I: Fundamentals of secure Multi-Party Computation (MPC)**
  - MPC use cases and basic concepts
  - Vectorization and message batching in MPC
  - Oblivious relational analytics with ORQ
- **Part II: Hands-on programming session with ORQ**
  - Deploying ORQ on CloudLab
  - ORQ example code walk-through
  - Implement and run your own secure data analysis pipeline using ORQ

SECURE COLLABORATIVE ANALYTICS

Common interest

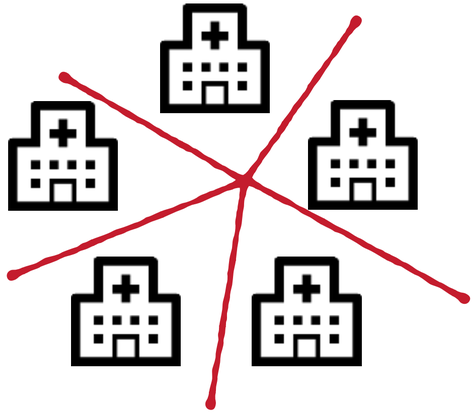
Disease surveillance



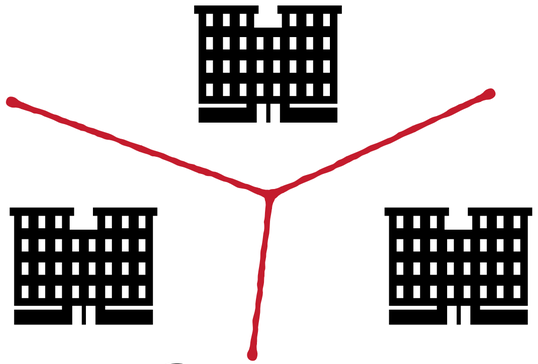
Gender or racial wage gap

Privacy-preserving advertising

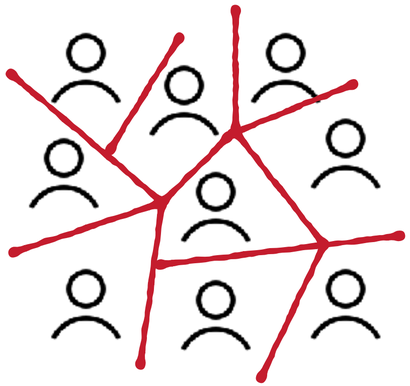
Private data



Healthcare providers



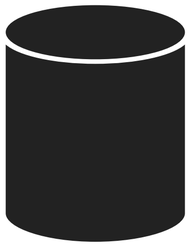
Companies

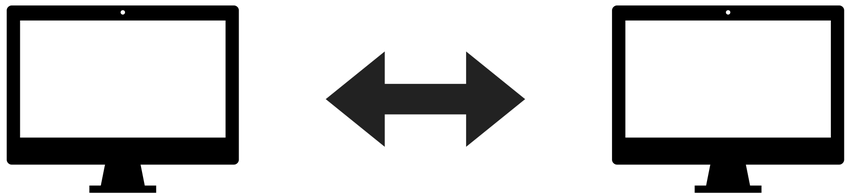


End-users

END-TO-END DATA PROTECTION

Goal: exact results on secret inputs





Data "at rest"

Data "in transit"

Data "in use"

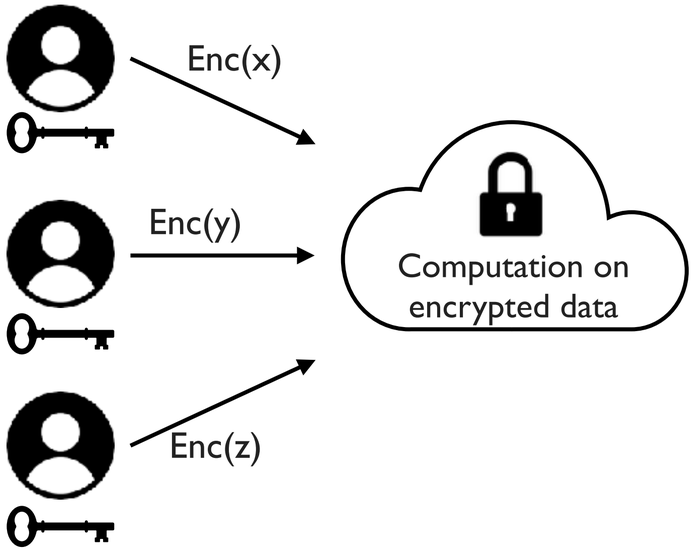
Advanced Encryption Standard (AES)

Transport Layers Security (TLS)

How can we protect data in use?

APPROACHES TO PROTECTING DATA "IN USE"

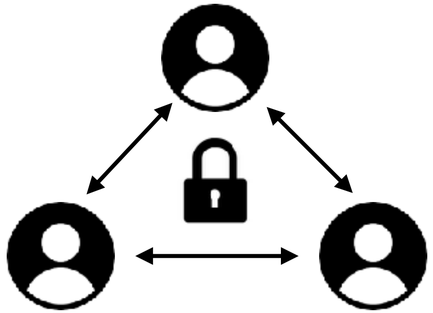
Fully Homomorphic Encryption (FHE)



Security via homomorphic encryption

(very high computational cost)

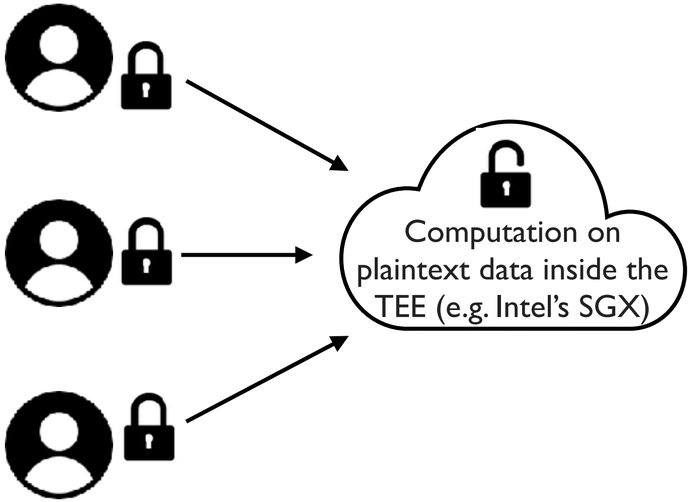
Secure Multi-Party Computation (MPC)



Security via decentralized trust

(high communication cost)

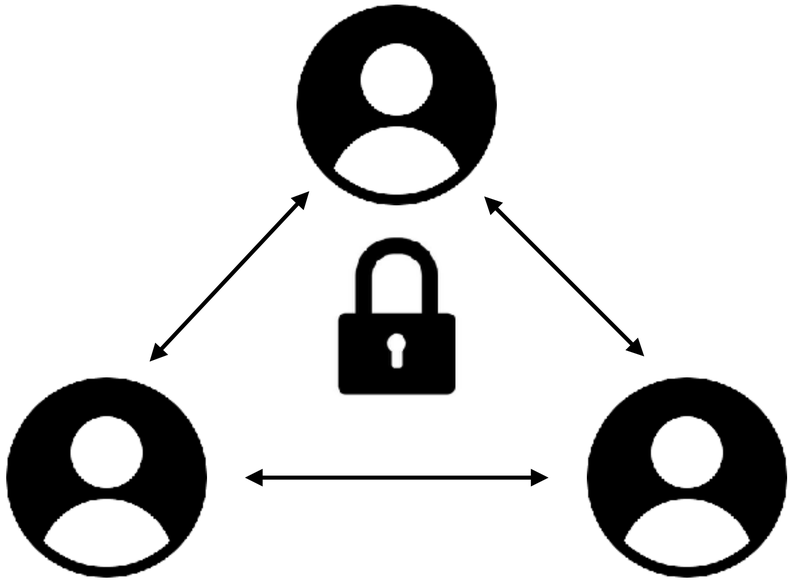
Trusted Execution Environments (TEEs)



Security via physically protected HW

(prone to side-channel attacks)

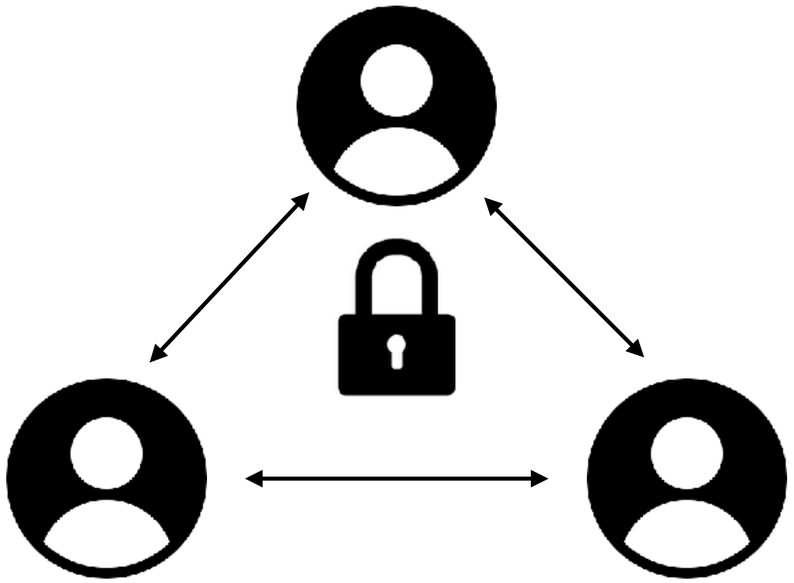
SECURE MULTI-PARTY COMPUTATION (MPC)



MPC is a family of **cryptographic protocols** that allows **mutually distrusting parties** to perform **arbitrary computations** on their **private data** while keeping the data siloed from each other and from external entities.

Yehuda Lindell. *Secure multiparty computation*. Commun. ACM, 64(1):86–96, December 2020.

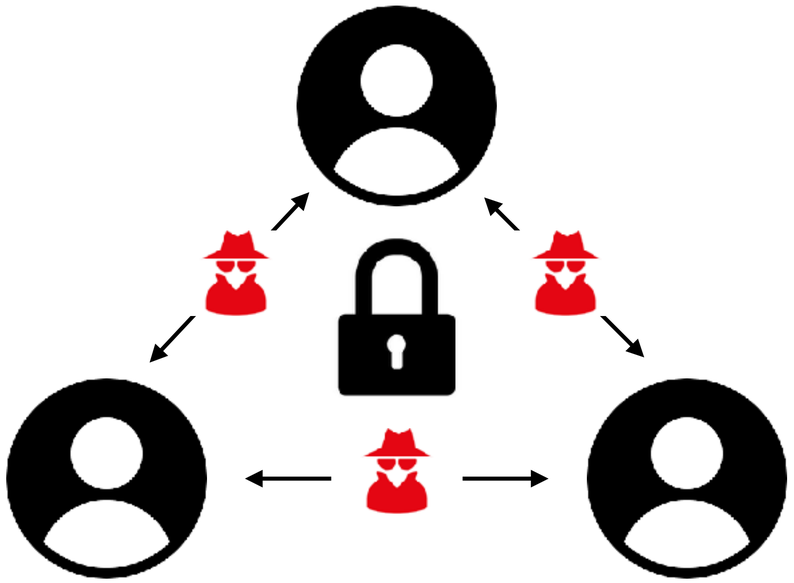
SECURE MULTI-PARTY COMPUTATION (MPC)



- Any number of parties

Yehuda Lindell. *Secure multiparty computation*. Commun. ACM, 64(1):86–96, December 2020.

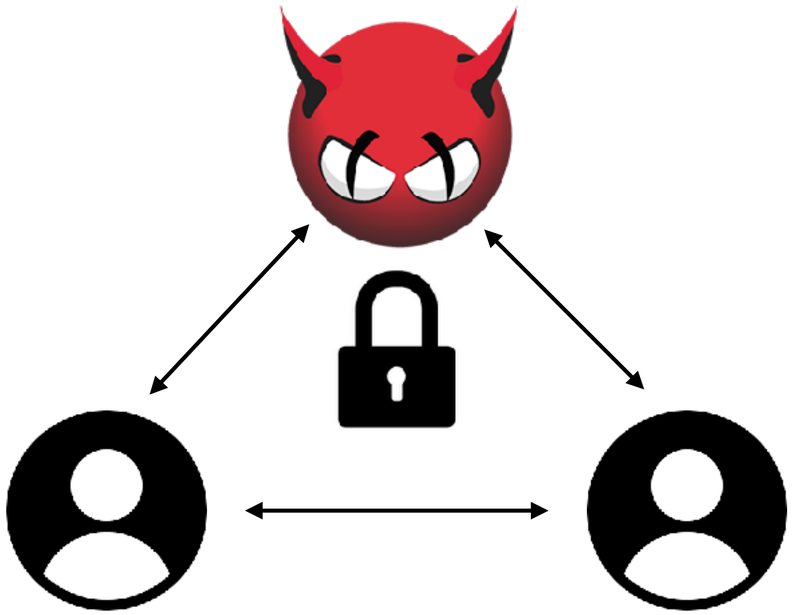
SECURE MULTI-PARTY COMPUTATION (MPC)



- Any number of parties
- Protection against external adversaries

Yehuda Lindell. *Secure multiparty computation*. Commun. ACM, 64(1):86–96, December 2020.

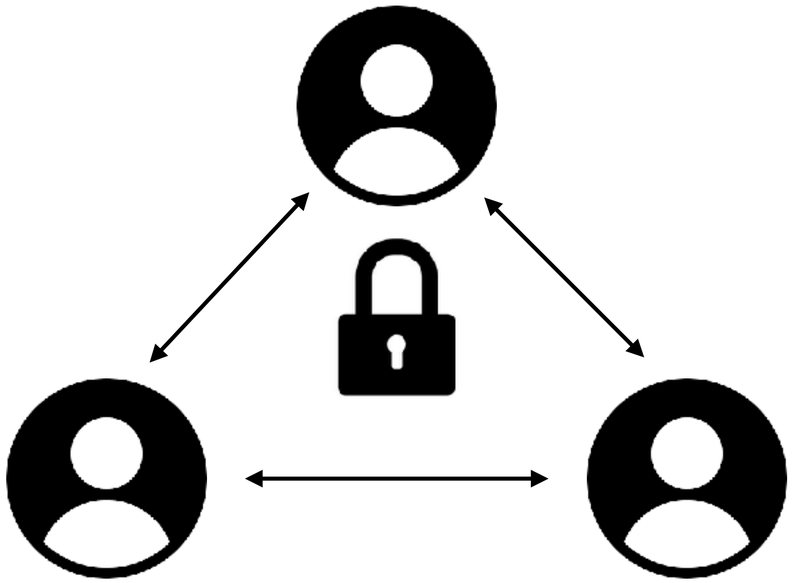
SECURE MULTI-PARTY COMPUTATION (MPC)



- Any number of parties
- Protection against external adversaries
- Semi-honest or malicious parties
  (honest majority or dishonest majority)

Yehuda Lindell. *Secure multiparty computation*. Commun. ACM, 64(1):86–96, December 2020.

SECURE MULTI-PARTY COMPUTATION (MPC)

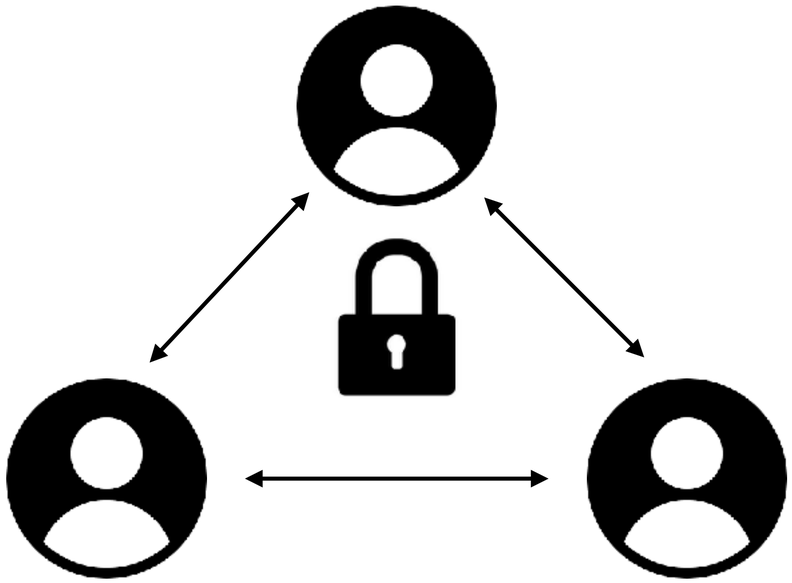


- Any number of parties
- Protection against external adversaries
- Semi-honest or malicious parties
- Arbitrary computations

  (relational analytics, machine learning, …)

Yehuda Lindell. *Secure multiparty computation*. Commun. ACM, 64(1):86–96, December 2020.

SECURE MULTI-PARTY COMPUTATION (MPC)

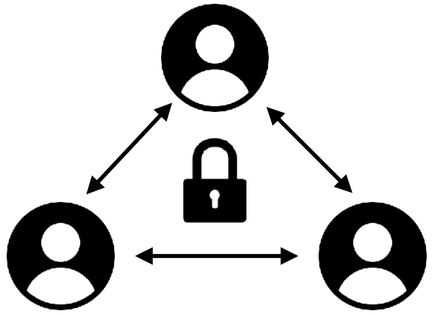


- Any number of parties
- Protection against external adversaries
- Semi-honest or malicious parties
- Arbitrary computations
- Easy to explain

…but not so easy to make it practical!

Yehuda Lindell. *Secure multiparty computation*. Commun. ACM, 64(1):86–96, December 2020.

DATA OWNERS CAN ACT AS COMPUTING PARTIES

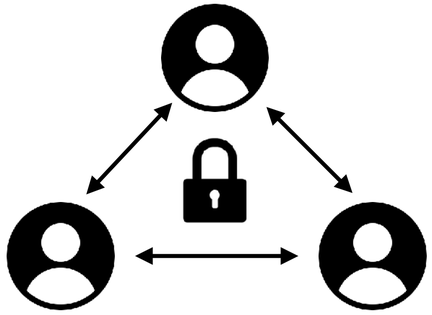


Peer-to-peer MPC

Data owners act as computing parties using trusted resources

- Data owners may not have domain expertise or private infrastructure
- MPC does not scale well with the number of data owners

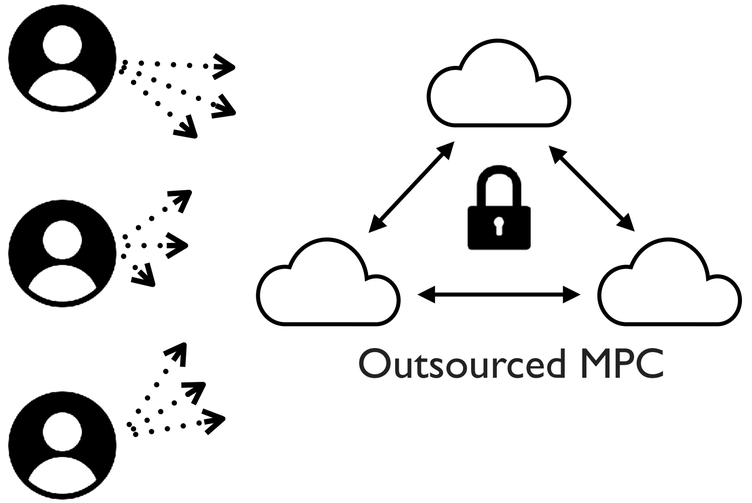
OUR FOCUS: OPTIMIZE MPC IN THE CLOUD



Peer-to-peer MPC

Data owners act as computing parties using trusted resources

- ⊖ Data owners may not have domain expertise or private infrastructure
- ⊖ MPC does not scale well with the number of data owners



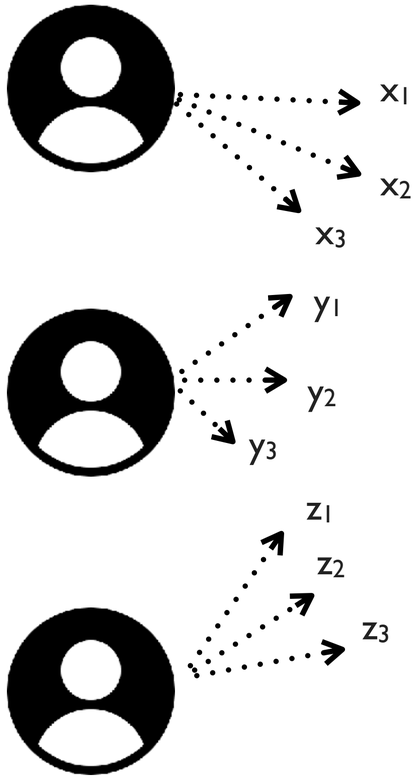
Outsourced MPC

Data owners outsource secret shares of their data to untrusted third parties

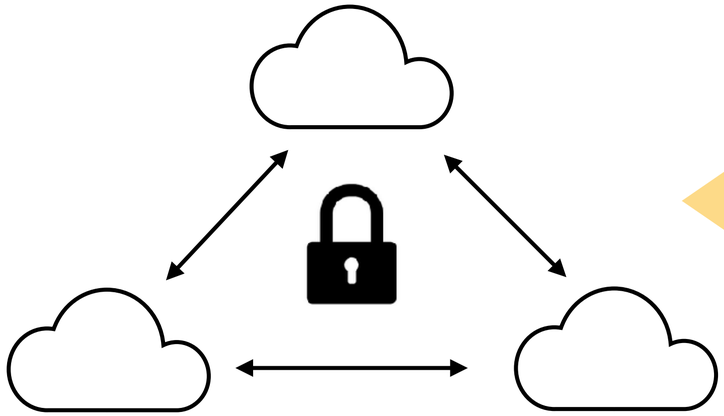
- ⊕ Data owners can use untrusted cloud resources on demand
- ⊕ A small number of third parties can support a large number of data owners

OUTSOURCED MPC: COMPUTING PARTIES ARE SEPARATE FROM DATA OWNERS

Data owners



Computing parties



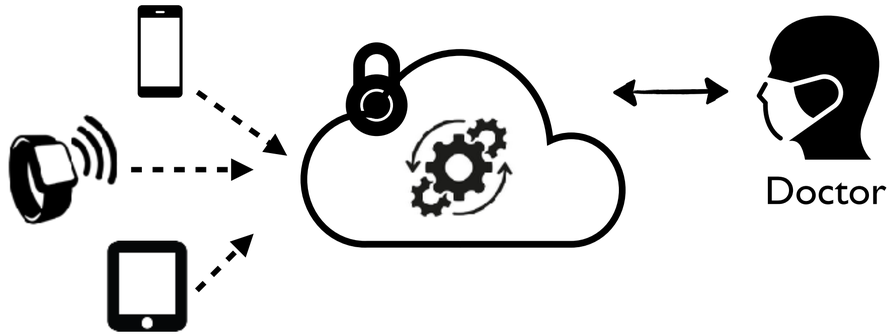
- Competing cloud providers
- Resources managed by different entities in a federated cloud

Untrusted non-colluding entities compute collectively on random "data shares"

REAL-WORLD USE CASES

Digital health analytics

(BU Medical & Hariri Institute for Computing)

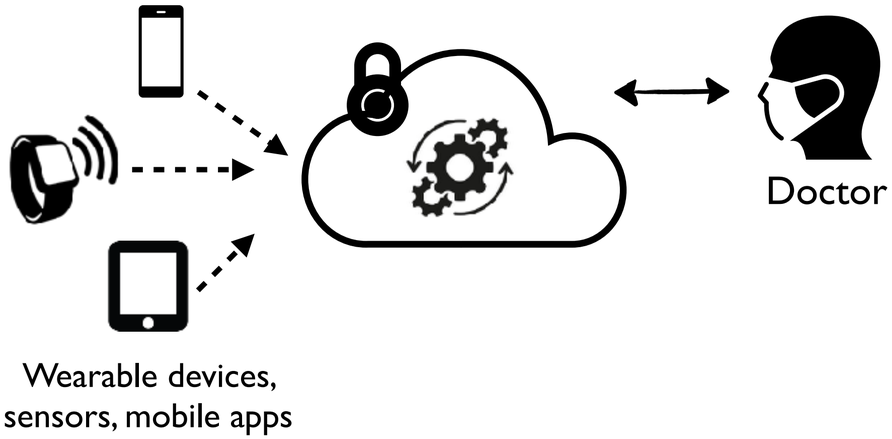


What is the % of non-diabetic patients whose glucose levels reach 180mg/dL while eating?

REAL-WORLD USE CASES

Digital health analytics

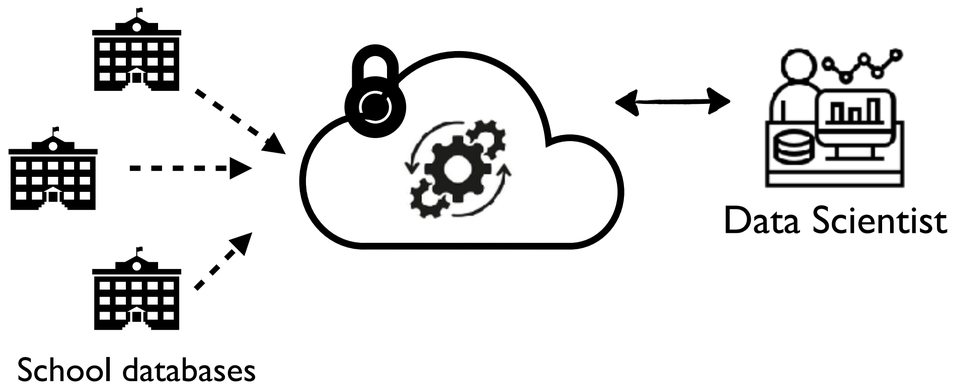
(BU Medical & Hariri Institute for Computing)



What is the % of non-diabetic patients whose glucose levels reach 180mg/dL while eating?

Predictive models in education

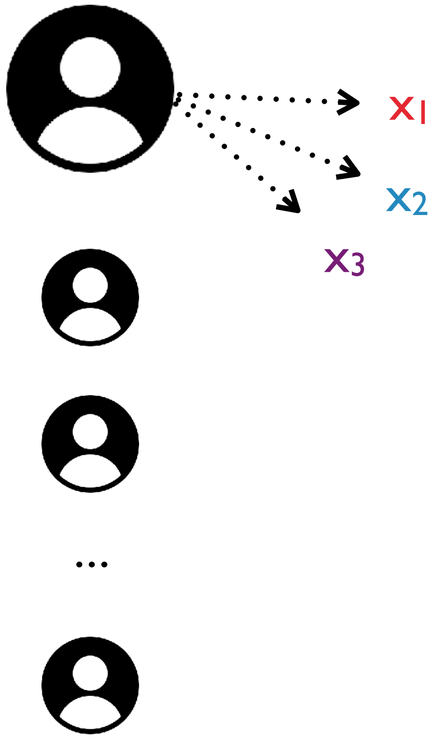
(BU Wheelock & McGovern Institute for Brain Research)



What is the expected student reading performance given past exam scores and family histories?

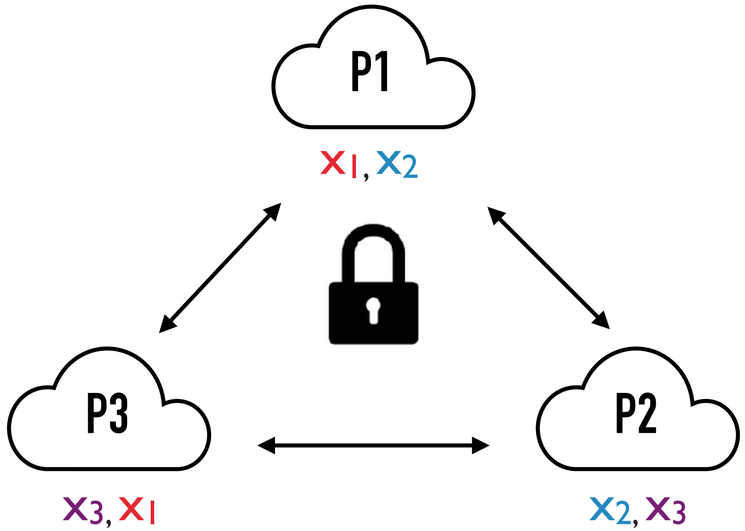
A brief introduction to MPC with replicated secret sharing

REPLICATED SECRET SHARING



Any number of data owners

3 computing parties



Each data owner encodes data item x using either arithmetic or boolean secret sharing:

Arithmetic sharing: $x = x_1 + x_2 + x_3 \pmod{2^k}$

Boolean sharing: $x = x_1 \oplus x_2 \oplus x_3$

where x_1, x_2, x_3 are uniformly random.

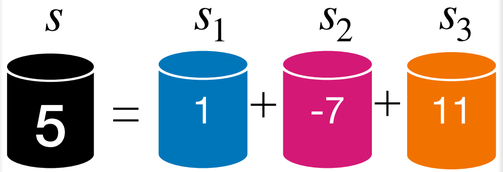
Each computing party receives 2 shares,
so that one party alone cannot reconstruct the original data item.

EXAMPLE: SECURE ADDITION

Arithmetic sharing: $x = x_1 + x_2 + x_3 \pmod{2^{64}}$

Party 1

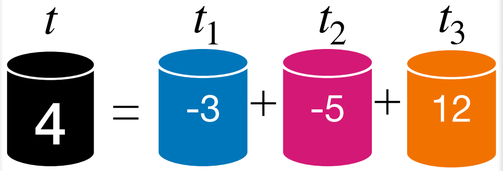
Data Owner A



Party 3

Party 2

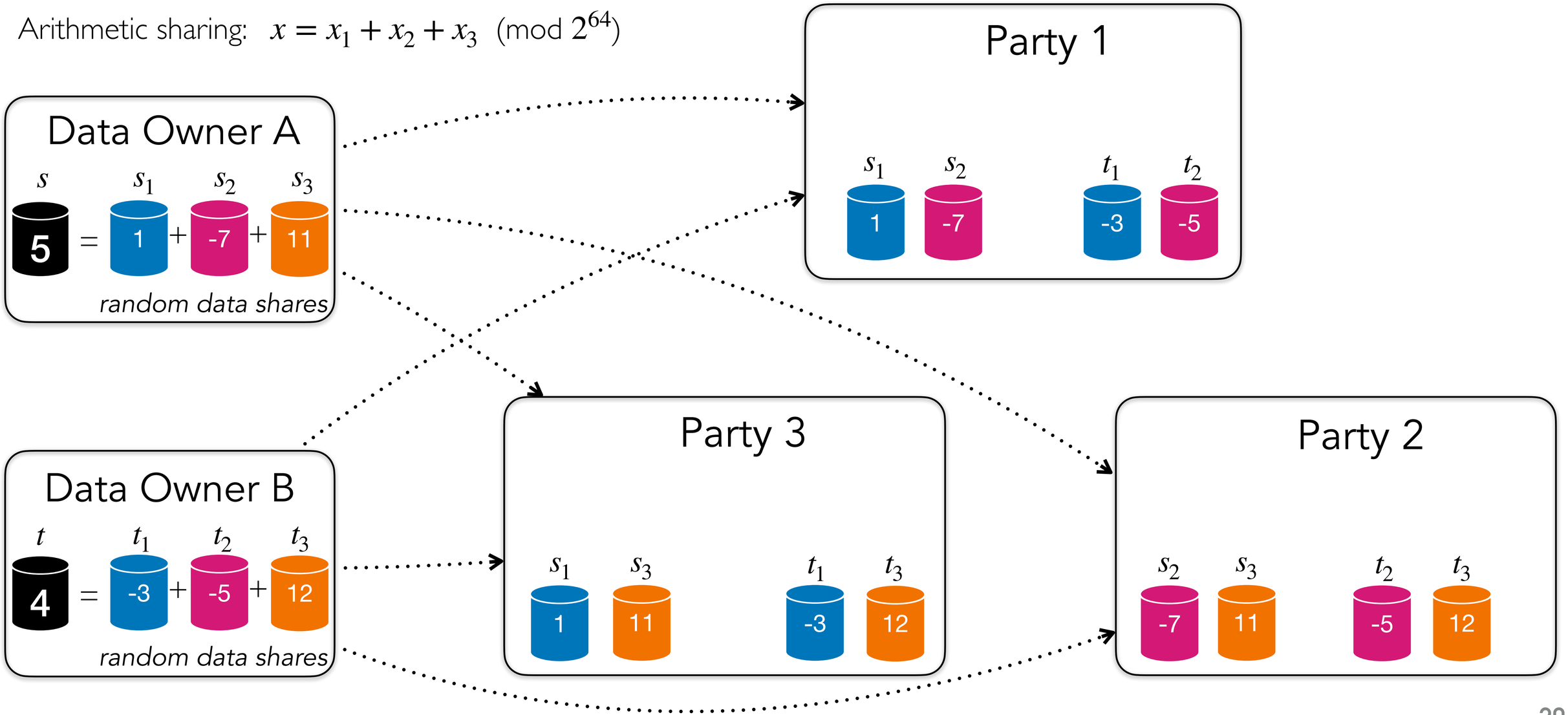
Data Owner B



T. Araki, J. Furukawa, Y. Lindell, A. Nof, and K. Ohara. *High-Throughput Semi-Honest Secure Three-Party Computation with an Honest Majority*. CCS, 2016.

EXAMPLE: SECURE ADDITION

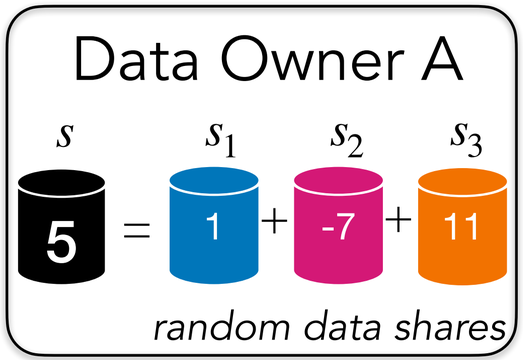
Arithmetic sharing: $x = x_1 + x_2 + x_3 \pmod{2^{64}}$

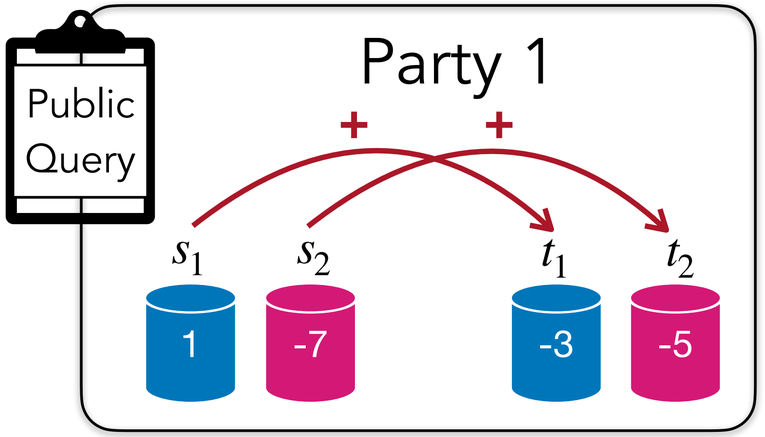


T. Araki, J. Furukawa, Y. Lindell, A. Nof, and K. Ohara. *High-Throughput Semi-Honest Secure Three-Party Computation with an Honest Majority*. CCS, 2016.

EXAMPLE: SECURE ADDITION

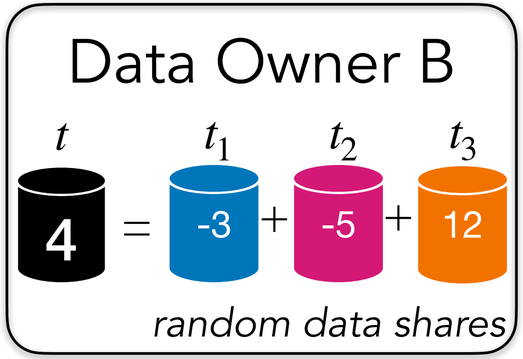
Arithmetic sharing: $x = x_1 + x_2 + x_3 \pmod{2^{64}}$

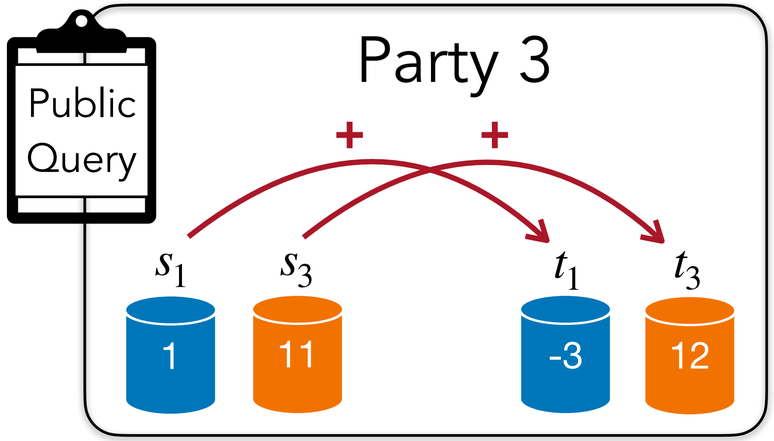


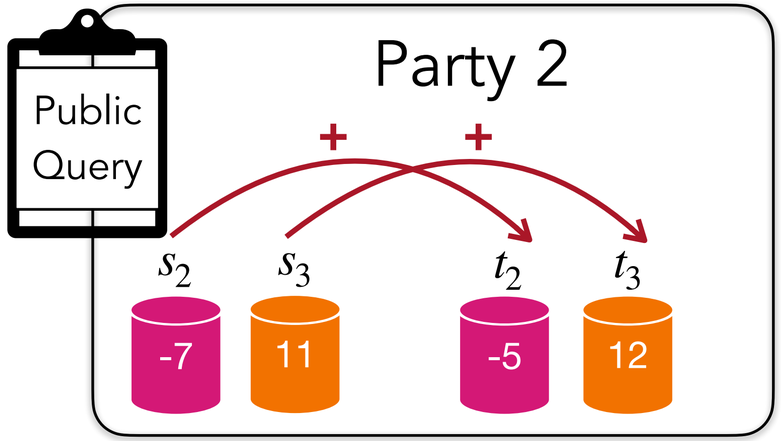


Query: $s + t$

$$s + t = (s_1 + t_1) + (s_2 + t_2) + (s_3 + t_3)$$



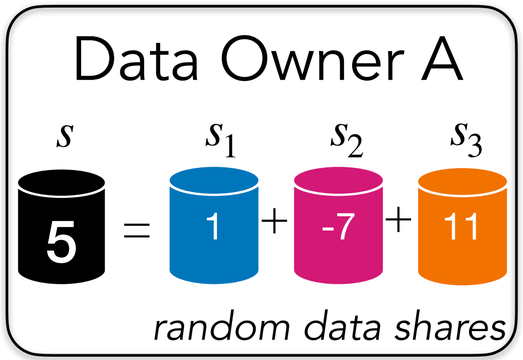


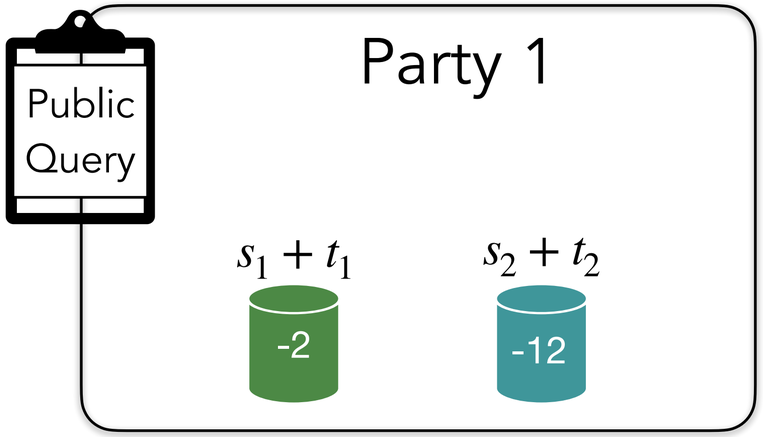


T. Araki, J. Furukawa, Y. Lindell, A. Nof, and K. Ohara. *High-Throughput Semi-Honest Secure Three-Party Computation with an Honest Majority*. CCS, 2016.

EXAMPLE: SECURE ADDITION

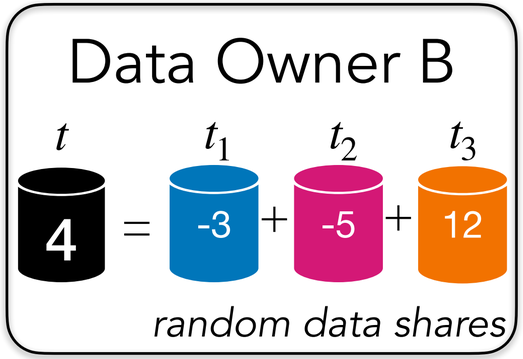
Arithmetic sharing: $x = x_1 + x_2 + x_3 \pmod{2^{64}}$

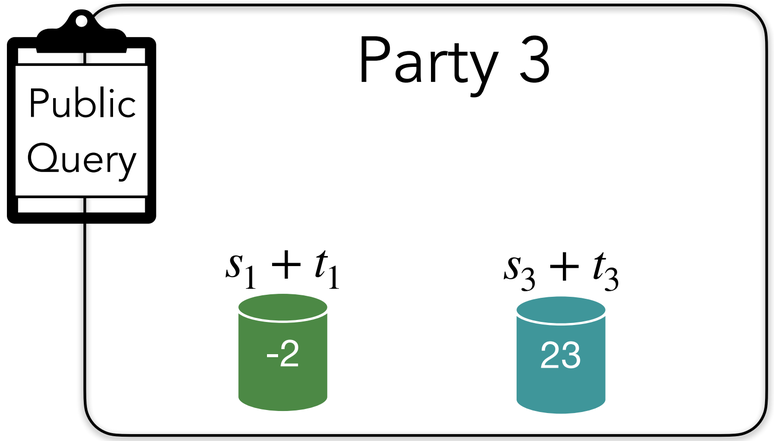


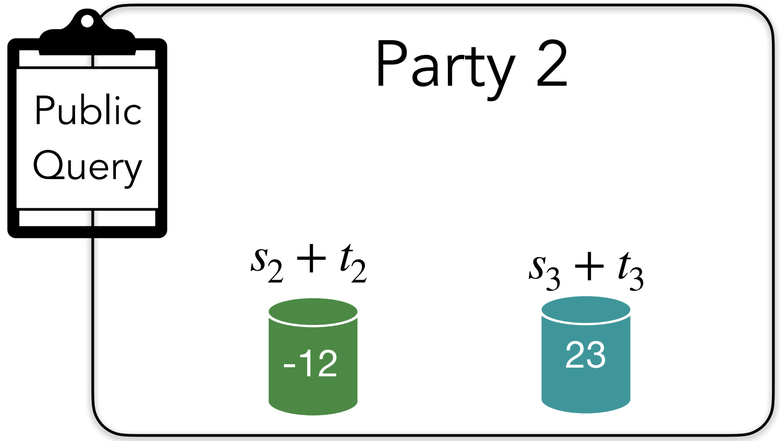


Query: $s + t$

$$s + t = (s_1 + t_1) + (s_2 + t_2) + (s_3 + t_3)$$



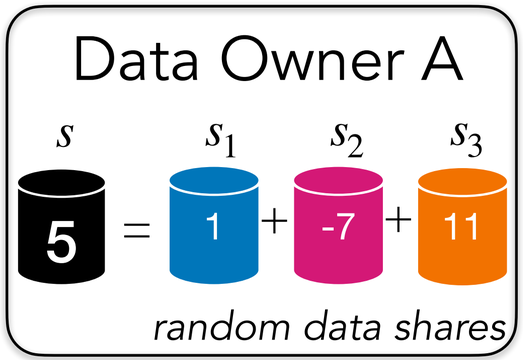


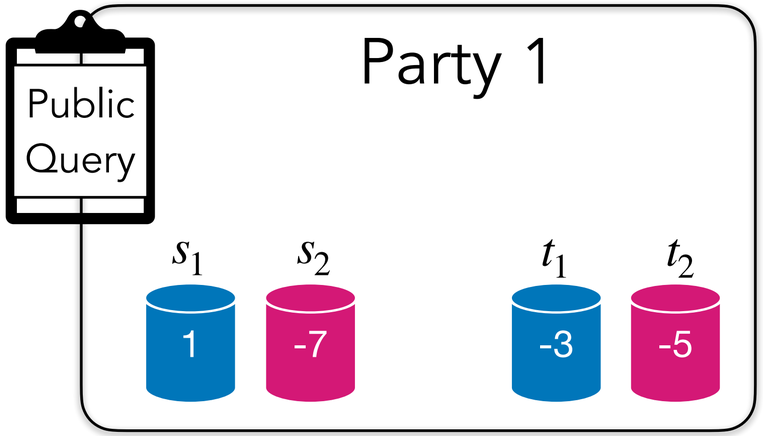


T. Araki, J. Furukawa, Y. Lindell, A. Nof, and K. Ohara. *High-Throughput Semi-Honest Secure Three-Party Computation with an Honest Majority*. CCS, 2016.

EXAMPLE: SECURE MULTIPLICATION

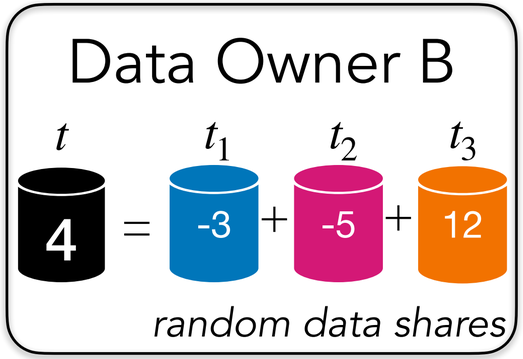
Arithmetic sharing: $x = x_1 + x_2 + x_3 \pmod{2^{64}}$

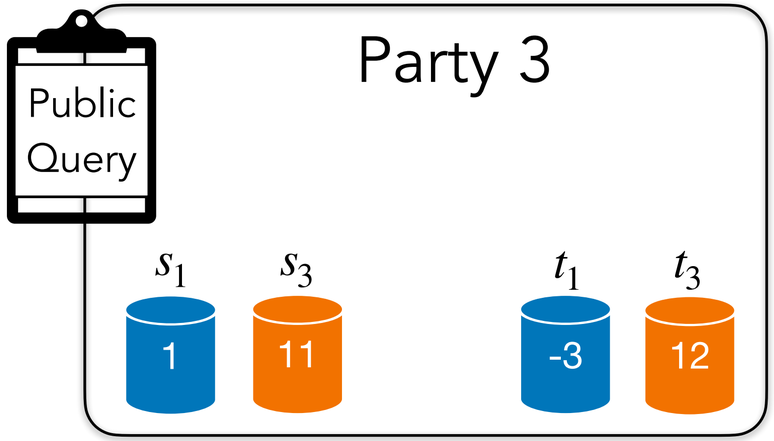


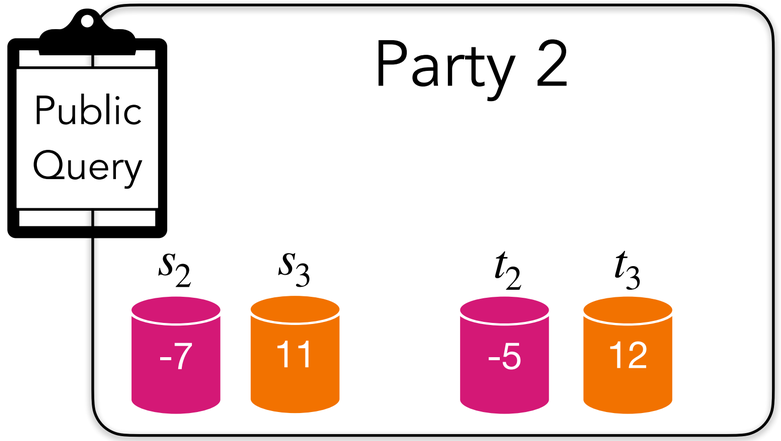


Query: $s \times t$

$$s \times t = (s_1 + s_2 + s_3) \cdot (t_1 + t_2 + t_3)$$



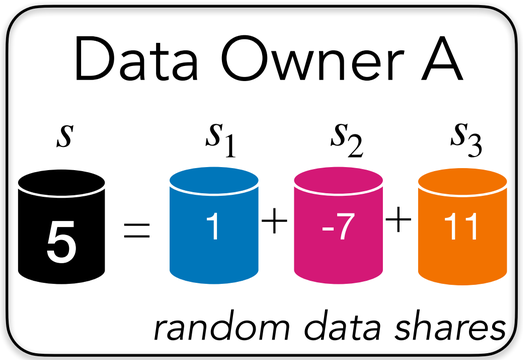


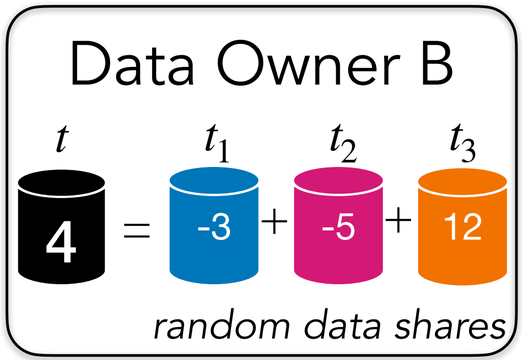


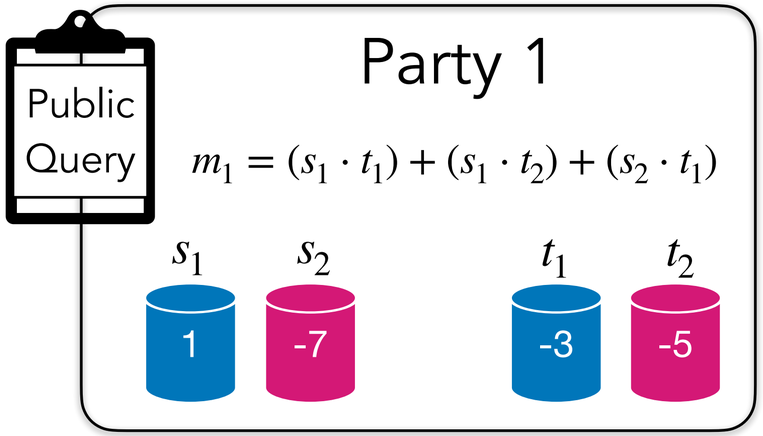
T. Araki, J. Furukawa, Y. Lindell, A. Nof, and K. Ohara. *High-Throughput Semi-Honest Secure Three-Party Computation with an Honest Majority*. CCS, 2016.

EXAMPLE: SECURE MULTIPLICATION

Arithmetic sharing: $x = x_1 + x_2 + x_3 \pmod{2^{64}}$

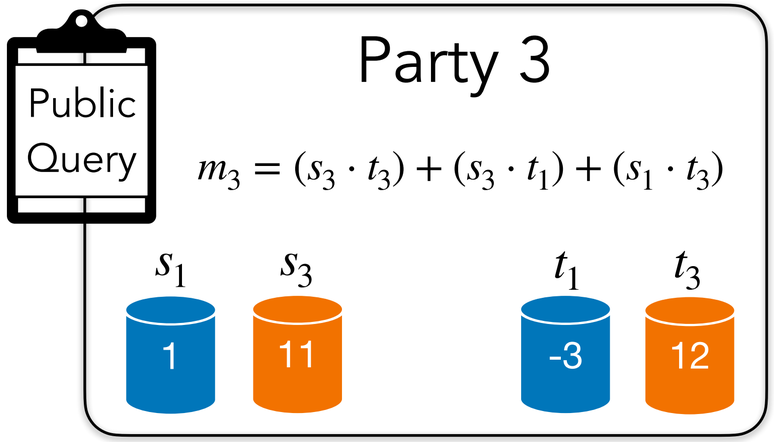


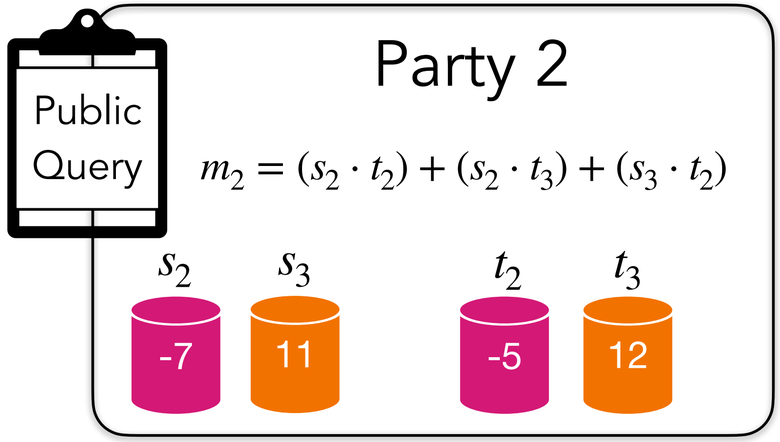




Query: $s \times t$

$$s \times t = (s_1 + s_2 + s_3) \cdot (t_1 + t_2 + t_3) = \cdots = m_1 + m_2 + m_3$$

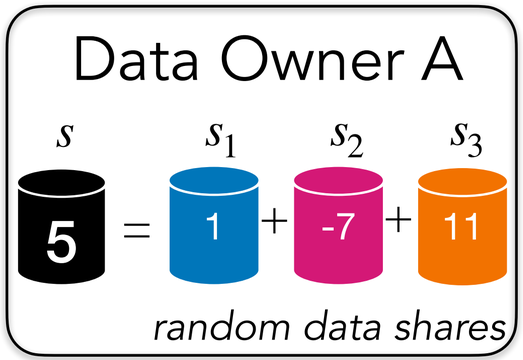


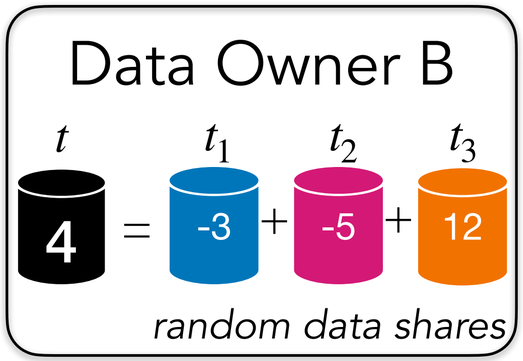


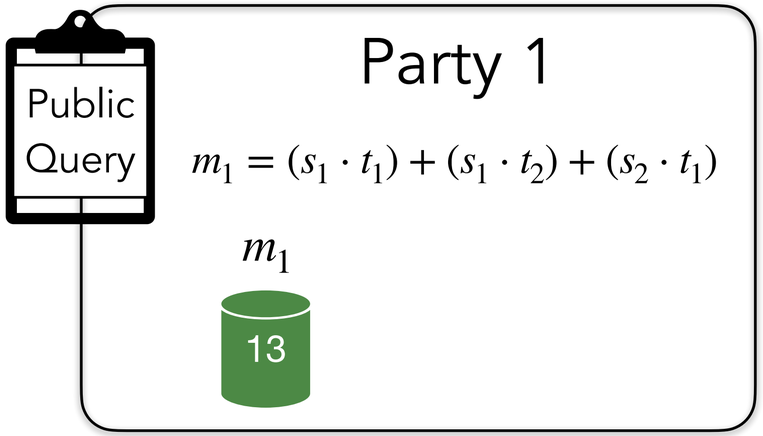
T. Araki, J. Furukawa, Y. Lindell, A. Nof, and K. Ohara. *High-Throughput Semi-Honest Secure Three-Party Computation with an Honest Majority*. CCS, 2016.

EXAMPLE: SECURE MULTIPLICATION

Arithmetic sharing: $x = x_1 + x_2 + x_3 \pmod{2^{64}}$

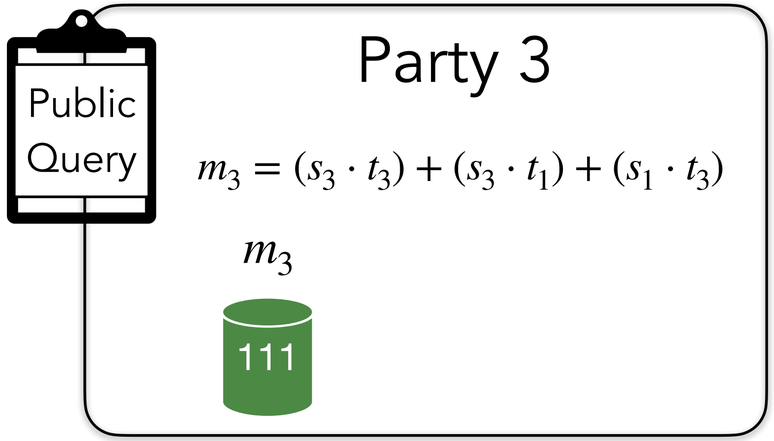


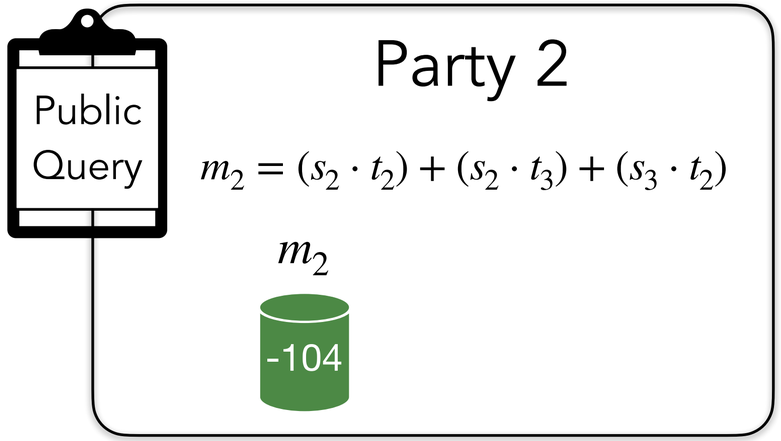




Query: $s \times t$

$$s \times t = (s_1 + s_2 + s_3) \cdot (t_1 + t_2 + t_3) = \cdots = m_1 + m_2 + m_3$$

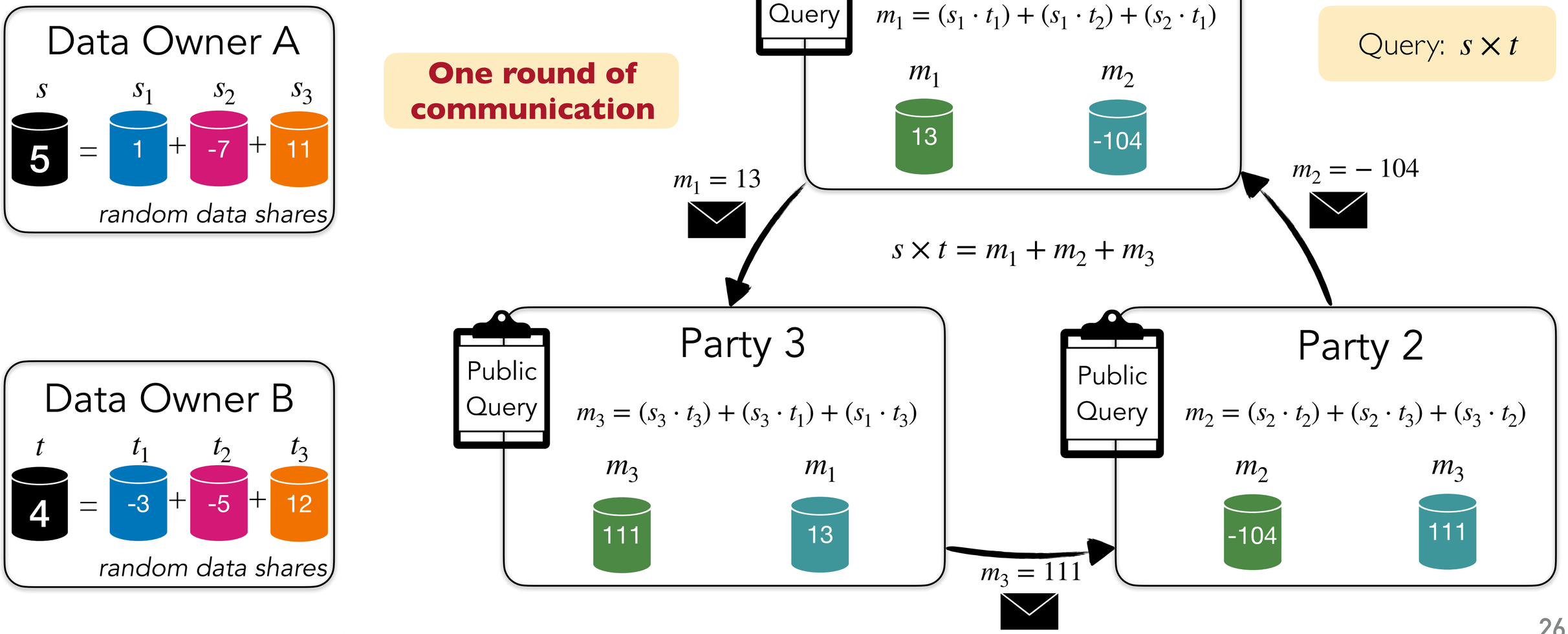




T. Araki, J. Furukawa, Y. Lindell, A. Nof, and K. Ohara. *High-Throughput Semi-Honest Secure Three-Party Computation with an Honest Majority*. CCS, 2016.

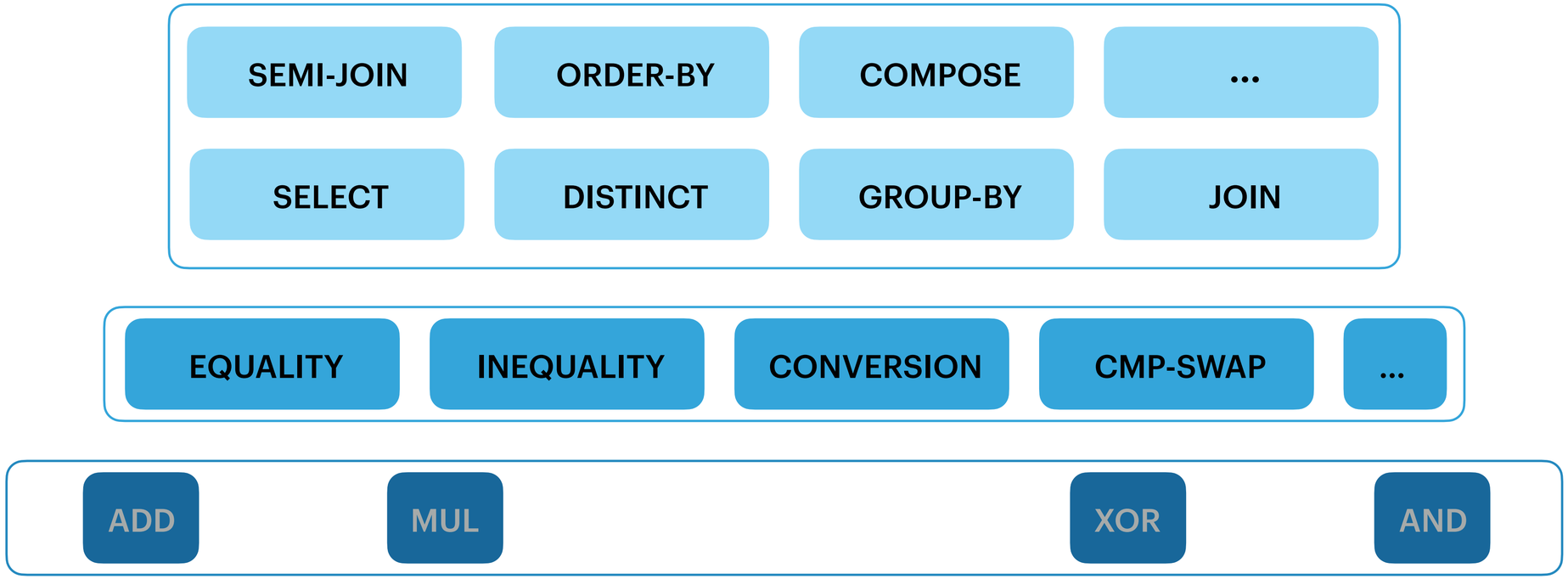
EXAMPLE: SECURE MULTIPLICATION

Arithmetic sharing: $x = x_1 + x_2 + x_3 \pmod{2^{64}}$



T. Araki, J. Furukawa, Y. Lindell, A. Nof, and K. Ohara. *High-Throughput Semi-Honest Secure Three-Party Computation with an Honest Majority*. CCS, 2016.

FROM SECURE PRIMITIVES TO SECURE ANALYTICS



Arithmetic sharing: $x = x_1 + x_2 + x_3 \pmod{2^{64}}$

Boolean sharing: $x = x_1 \oplus x_2 \oplus x_3$

SECURITY COMES AT A COST

Every MUL / AND operation requires **data exchange**

- Billions of primitive operations to support analytics
- High **latency** and **bandwidth** requirements when parties are deployed in different networks

No leakage relies on **oblivious computation**

- The computing parties perform identical computation that is **data-independent**
  - Worst-case padding
  - All data will be accessed at least once (cannot achieve sublinear complexity)

AVOIDING INFORMATION LEAKAGE

Information leakage can lead to **reconstruction attacks** and may reveal sensitive data:

```
SELECT *
FROM EMPLOYEE
WHERE GENDER='F'
AND RANK='manager'
AND SALARY>50000
```

```
if (p1.salary)>(p2.salary) => gap
```

OBLIVIOUS COMPUTATION

To prevent information leakage, the computing parties perform an identical computation that is data-independent

- Data access patterns do not depend on the actual shares
- No conditionals (if-then-else)
- No data reduction

* *Both a and b are secret-shared*

For 3-bit numbers: $a : a_2a_1a_0$ $b : b_2b_1b_0$

If $(a > b)$ {…}

$$\phi = a \overset{?}{>} b = (a_2 \oplus b_2) \wedge a_2$$
$$\oplus\ (a_2 \oplus b_2 \oplus 1) \wedge (a_1 \oplus b_1) \wedge a_1$$
$$\oplus\ (a_2 \oplus b_2 \oplus 1) \wedge (a_1 \oplus b_1 \oplus 1) \wedge ((b_0 \oplus 1) \wedge a_0)$$

"If the most significant bits are not the same, then a is greater than b when a_2 is set"

Cleartext

Oblivious

For k-bit values, each inequality requires $O(\log k)$ communication rounds under MPC

OBLIVIOUS COMPUTATION

To prevent information leakage, the computing parties perform an identical computation that is data-independent

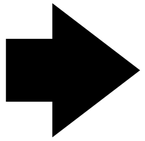
- Data access patterns do not depend on the actual shares
- No conditionals (if-then-else)
- No data reduction

**all values are secret-shared*

R

Employee	Salary
Kim	2000
Jane	1500
Alex	4500

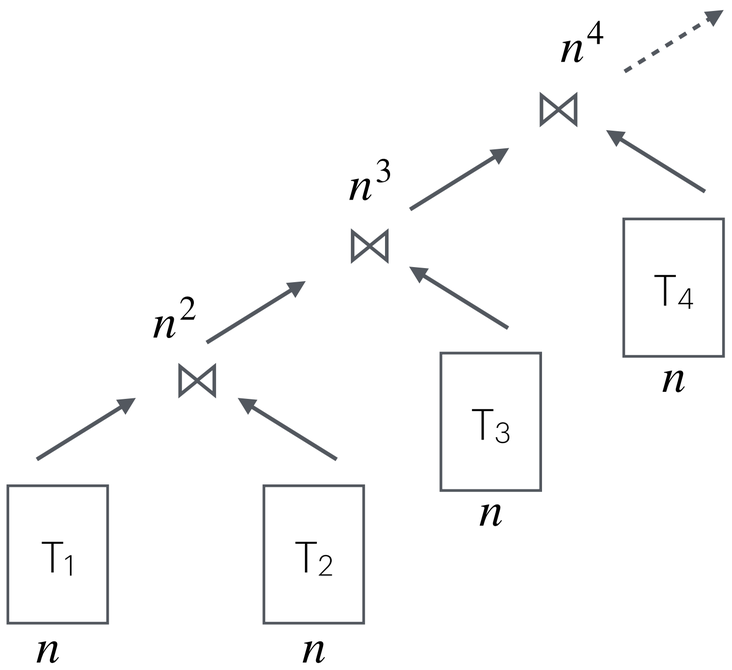
$\sigma(Salary > 3000)$



R'

Employee	Salary	ϕ
Kim	2000	0
Jane	1500	0
Alex	4500	1

CORE PROBLEM: THE CASCADING EFFECT



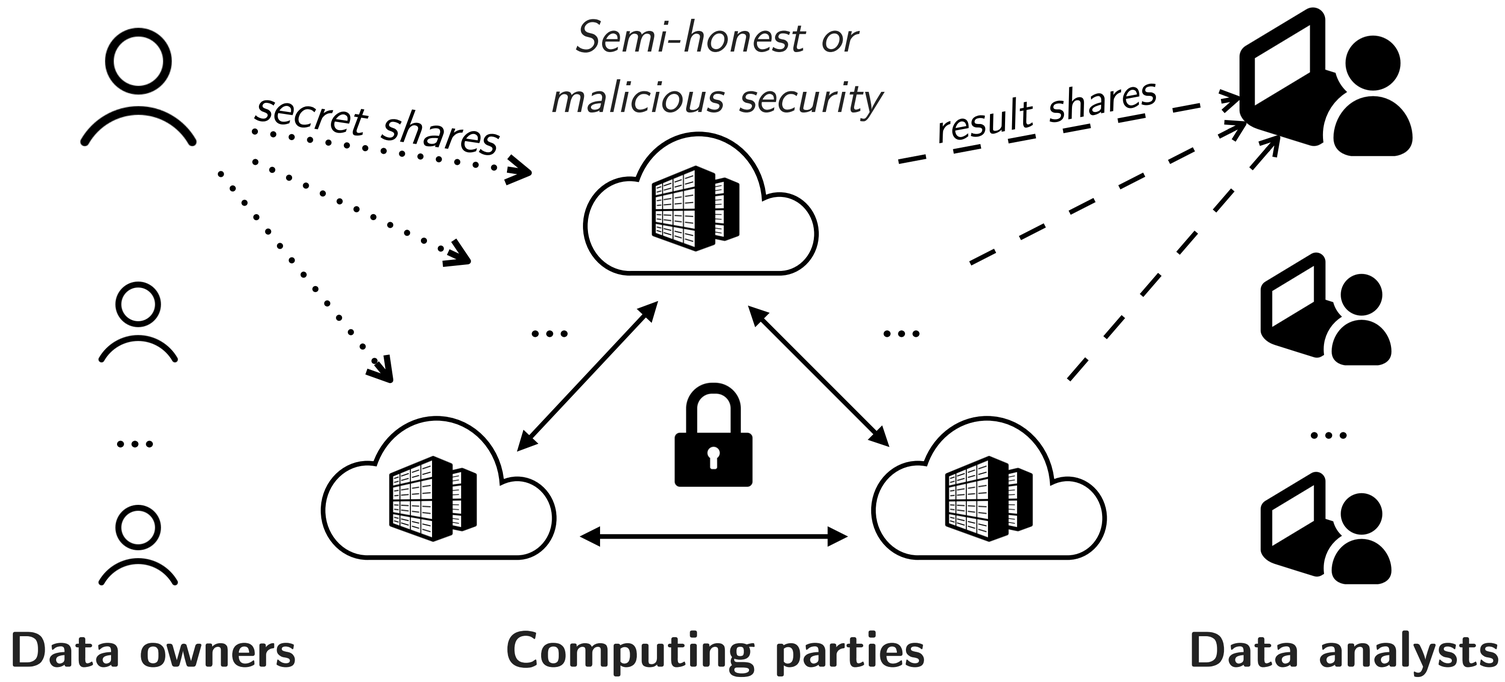
Joining k tables of n rows each would require $O(n^k)$ operations under MPC

- Oblivious operators produce **worst-case** outputs
- Potential **duplicates** in both join inputs require computing the **cartesian product**
- Duplicates are the **norm** – multiple data owners may contribute to the same tables
- A challenging open problem in oblivious analytics (MPC, TEEs, FHE, etc.)

ORQ: Complex Analytics on Private Data with Strong Security Guarantees

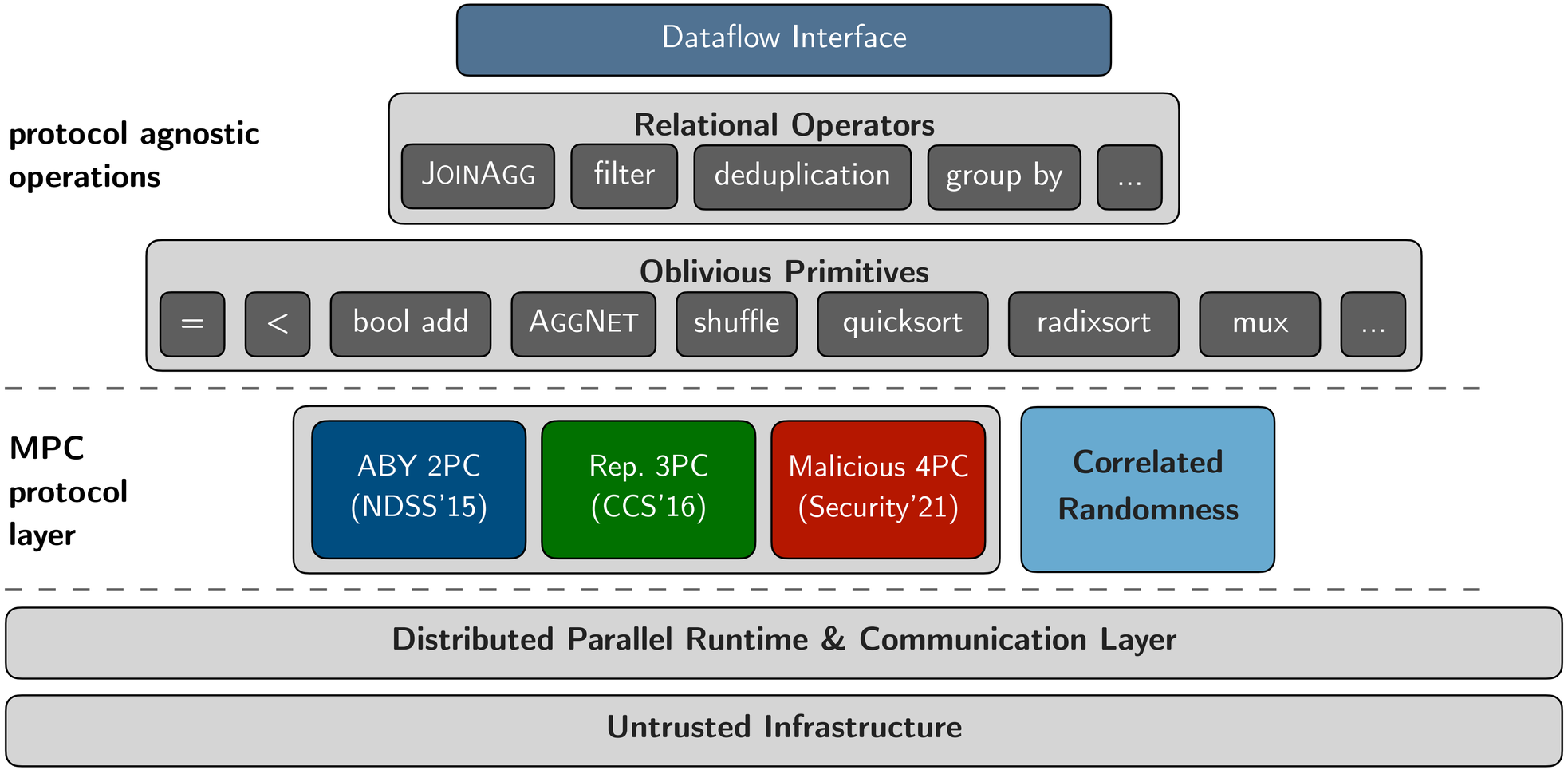
E. Baum, S. Buxbaum, N. Mathai, M. Faisal, V. Kalavri, M. Varia, J. Liagouris. *ORQ: Complex Analytics on Private Data with Strong Security Guarantees*. SOSP, 2025.

ORQ SETTING



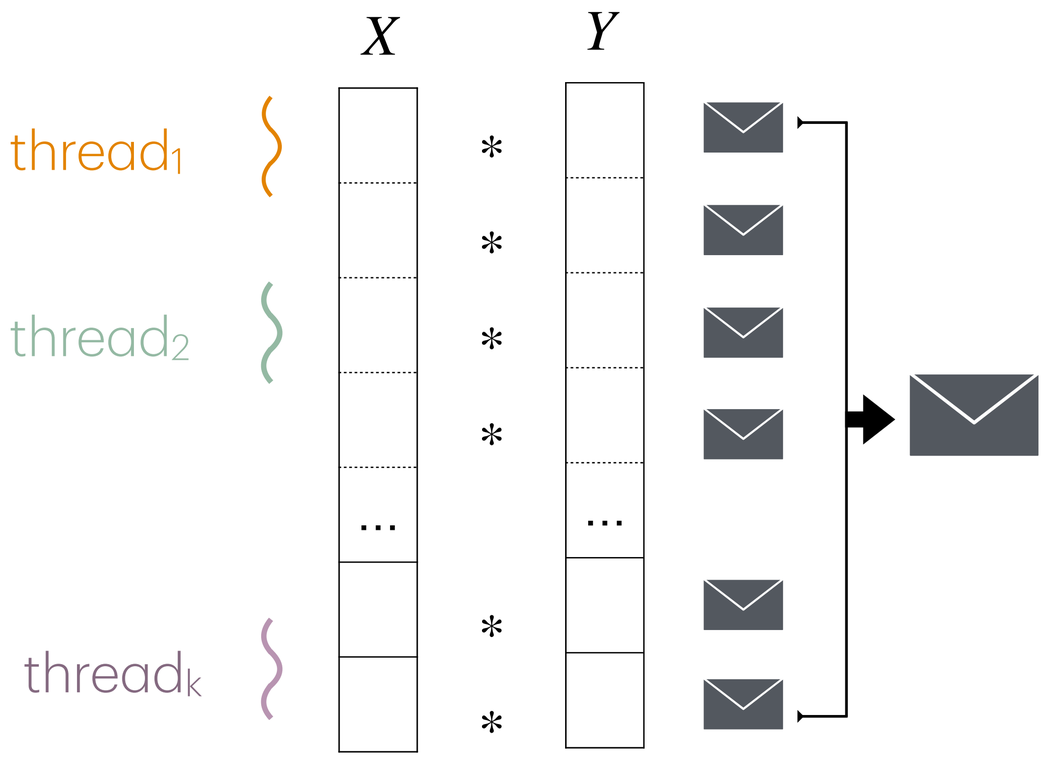
ORQ targets the typical outsourced setting that uses a small set of computing parties to support any number of data owners and analysts

ORQ STACK


Dataflow Interface
protocol agnostic operations
Relational Operators
JoinAgg
filter
deduplication
group by
...
Oblivious Primitives
=
<
bool add
AggNet
shuffle
quicksort
radixsort
mux
...
MPC protocol layer
ABY 2PC (NDSS'15)
Rep. 3PC (CCS'16)
Malicious 4PC (Security'21)
Correlated Randomness
Distributed Parallel Runtime & Communication Layer
Untrusted Infrastructure

Mitigating MPC communication costs

VECTORIZATION FACILITATES MESSAGE BATCHING AND DATA PARALLELISM

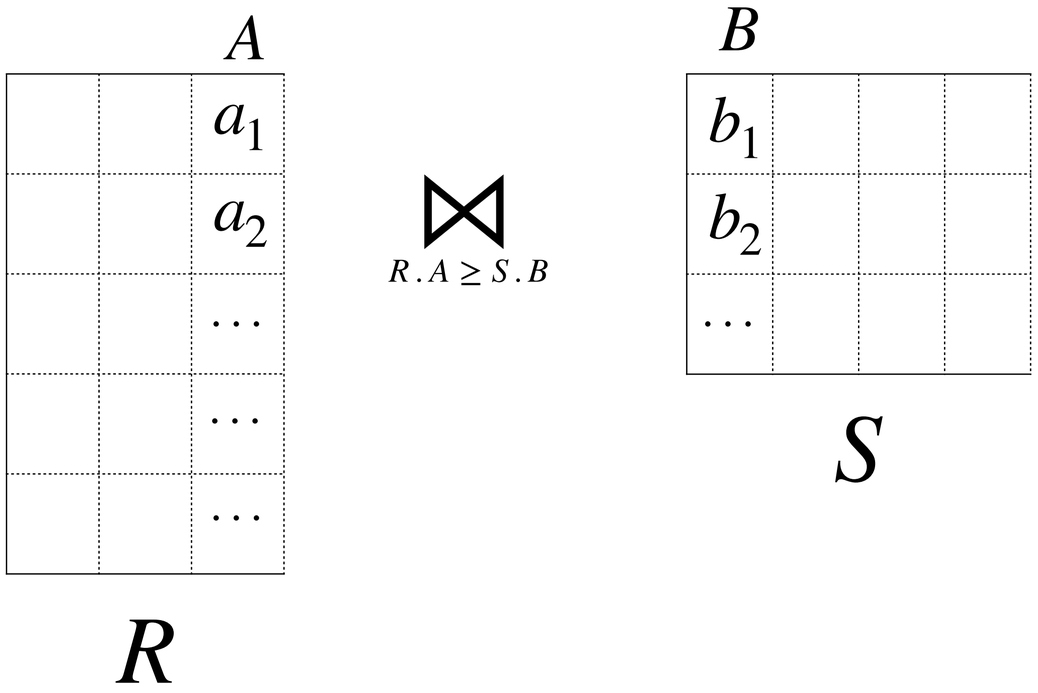


vectorized multiplication

One communication round
for the entire vector

(independent of the number of elements)

MESSAGE BATCHING EXAMPLE

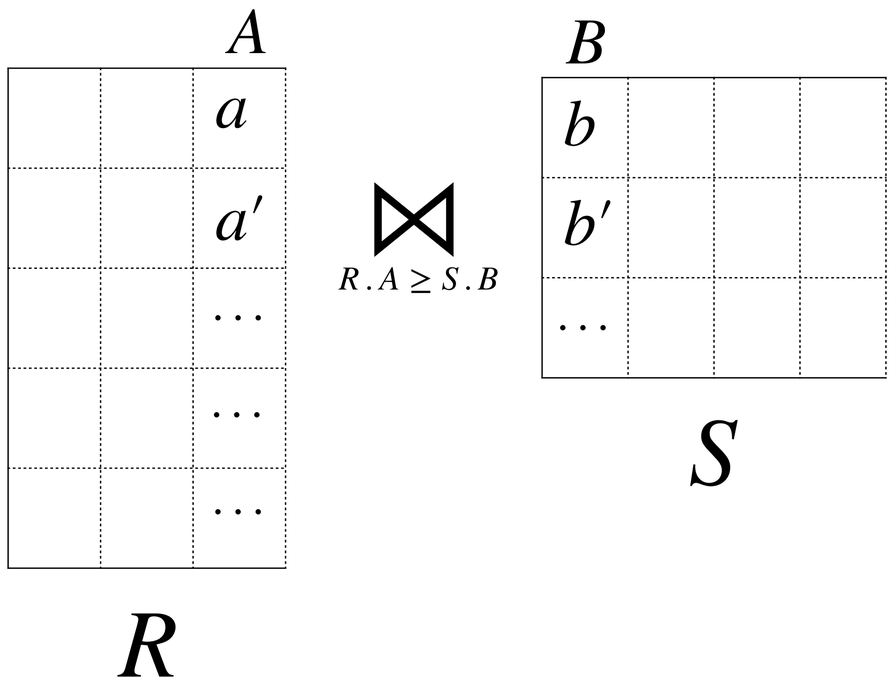


$$[\phi_1 \leftarrow a_1 \overset{?}{\geq} b_1]$$
$$[\phi_2 \leftarrow a_1 \overset{?}{\geq} b_2]$$
$$\cdots$$
$$[\phi_\kappa \leftarrow a_2 \overset{?}{\geq} b_1]$$
$$[\phi_{\kappa+1} \leftarrow a_2 \overset{?}{\geq} b_2]$$
$$\cdots$$

MESSAGE BATCHING EXAMPLE

$$\ell = 4 \text{ bits}$$

$$\underbrace{a_3a_2a_1a_0}_{a} \quad \underbrace{b_3b_2b_1b_0}_{b} \quad \underbrace{a'_3a'_2a'_1a'_0}_{a'} \quad \underbrace{b'_3b'_2b'_1b'_0}_{b'}$$



$R.A \geq S.B$

$$a \overset{?}{\geq} b \iff (a_3 \oplus b_3) \wedge a_3$$
$$\oplus\ ((a_3 \oplus b_3) \oplus 1) \wedge (a_2 \oplus b_2) \wedge a_2$$
$$\oplus\ ((a_3 \oplus b_3) \oplus 1) \wedge ((a_2 \oplus b_2) \oplus 1) \wedge (a_1 \oplus b_1) \wedge a_1$$
$$\oplus\ ((a_3 \oplus b_3) \oplus 1) \wedge ((a_2 \oplus b_2) \oplus 1) \wedge ((a_1 \oplus b_1) \oplus 1) \wedge (((a_0 \oplus 1) \wedge b_0) \oplus 1)$$

$$a \overset{?}{\geq} b' \iff (a_3 \oplus b'_3) \wedge a'_3$$
$$\oplus\ ((a_3 \oplus b'_3) \oplus 1) \wedge (a_2 \oplus b'_2) \wedge a_2$$
$$\oplus\ ((a_3 \oplus b'_3) \oplus 1) \wedge ((a_2 \oplus b'_2) \oplus 1) \wedge (a_1 \oplus b'_1) \wedge a_1$$
$$\oplus\ ((a_3 \oplus b'_3) \oplus 1) \wedge ((a_2 \oplus b'_2) \oplus 1) \wedge ((a_1 \oplus b'_1) \oplus 1) \wedge (((a_0 \oplus 1) \wedge b'_0) \oplus 1)$$

$$a' \overset{?}{\geq} b \iff \cdots$$

MESSAGE BATCHING EXAMPLE

$\ell = 4$ *bits*

$$\underbrace{a_3a_2a_1a_0}_{a} \quad \underbrace{b_3b_2b_1b_0}_{b} \quad \underbrace{a'_3a'_2a'_1a'_0}_{a'} \quad \underbrace{b'_3b'_2b'_1b'_0}_{b'}$$

		A
		a
		a'
		$\cdots$
		$\cdots$
		$\cdots$

R

$\bowtie_{R.A \geq S.B}$

B			
b			
b'			
$\cdots$			

S

$$a \overset{?}{\geq} b \iff (a_3 \oplus b_3) \wedge a_3$$
$$\oplus\ ((a_3 \oplus b_3) \oplus 1) \wedge (a_2 \oplus b_2) \wedge a_2$$
$$\oplus\ ((a_3 \oplus b_3) \oplus 1) \wedge ((a_2 \oplus b_2) \oplus 1) \wedge (a_1 \oplus b_1) \wedge a_1$$
$$\oplus\ ((a_3 \oplus b_3) \oplus 1) \wedge ((a_2 \oplus b_2) \oplus 1) \wedge ((a_1 \oplus b_1) \oplus 1) \wedge (((a_0 \oplus 1) \wedge b_0) \oplus 1)$$

Compute XORs
(no communication)

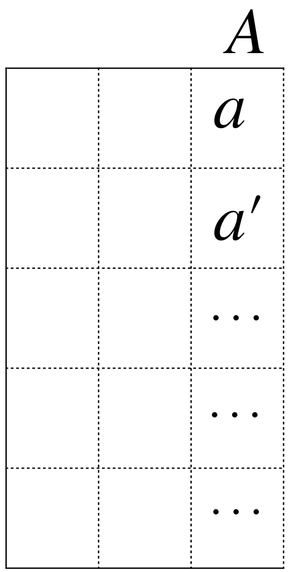
$$a \overset{?}{\geq} b' \iff (a_3 \oplus b'_3) \wedge a'_3$$
$$\oplus\ ((a_3 \oplus b'_3) \oplus 1) \wedge (a_2 \oplus b'_2) \wedge a_2$$
$$\oplus\ ((a_3 \oplus b'_3) \oplus 1) \wedge ((a_2 \oplus b'_2) \oplus 1) \wedge (a_1 \oplus b'_1) \wedge a_1$$
$$\oplus\ ((a_3 \oplus b'_3) \oplus 1) \wedge ((a_2 \oplus b'_2) \oplus 1) \wedge ((a_1 \oplus b'_1) \oplus 1) \wedge (((a_0 \oplus 1) \wedge b'_0) \oplus 1)$$

$$a' \overset{?}{\geq} b \iff \cdots$$

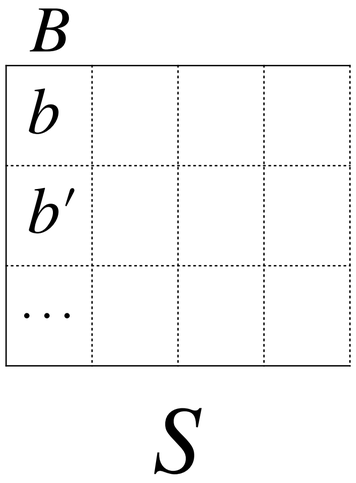
MESSAGE BATCHING EXAMPLE

$\ell = 4$ *bits*

$$\underbrace{a_3a_2a_1a_0}_{a} \quad \underbrace{b_3b_2b_1b_0}_{b} \quad \underbrace{a'_3a'_2a'_1a'_0}_{a'} \quad \underbrace{b'_3b'_2b'_1b'_0}_{b'}$$



$\bowtie_{R.A \geq S.B}$



$$a \overset{?}{\geq} b \iff c_1 \wedge a_3 \oplus c_2 \wedge c_3 \wedge a_2 \oplus c_4 \wedge c_5 \wedge c_6 \wedge a_1 \oplus c_4 \wedge c_5 \wedge c_7 \wedge (c_8 \wedge b_0)$$

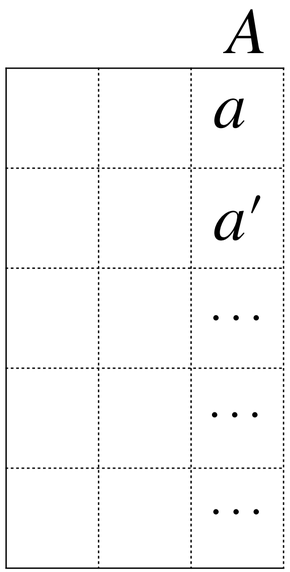
$$a \overset{?}{\geq} b' \iff c'_1 \wedge a_3 \oplus c'_2 \wedge c'_3 \wedge a_2 \oplus c'_4 \wedge c'_5 \wedge c'_6 \wedge a_1 \oplus c'_4 \wedge c'_5 \wedge c'_7 \wedge (c_8 \wedge b'_0)$$

$$a' \overset{?}{\geq} b \iff \cdots$$

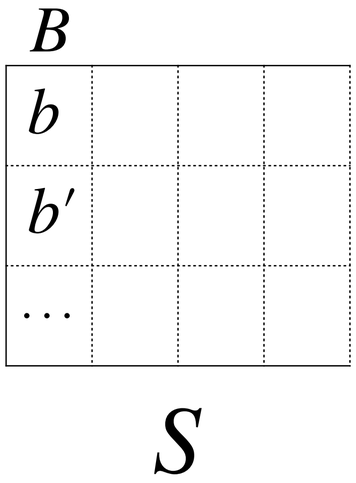
MESSAGE BATCHING EXAMPLE

$\ell = 4$ *bits*

$$\underbrace{a_3a_2a_1a_0}_{a} \quad \underbrace{b_3b_2b_1b_0}_{b} \quad \underbrace{a'_3a'_2a'_1a'_0}_{a'} \quad \underbrace{b'_3b'_2b'_1b'_0}_{b'}$$



$\bowtie_{R.A \geq S.B}$



$$a \overset{?}{\geq} b \iff c_1 \wedge a_3 \oplus c_2 \wedge c_3 \wedge a_2 \oplus c_4 \wedge c_5 \wedge c_6 \wedge a_1 \oplus c_4 \wedge c_5 \wedge c_7 \wedge (c_8 \wedge b_0)$$

$$a \overset{?}{\geq} b' \iff c'_1 \wedge a_3 \oplus c'_2 \wedge c'_3 \wedge a_2 \oplus c'_4 \wedge c'_5 \wedge c'_6 \wedge a_1 \oplus c'_4 \wedge c'_5 \wedge c'_7 \wedge (c_8 \wedge b'_0)$$

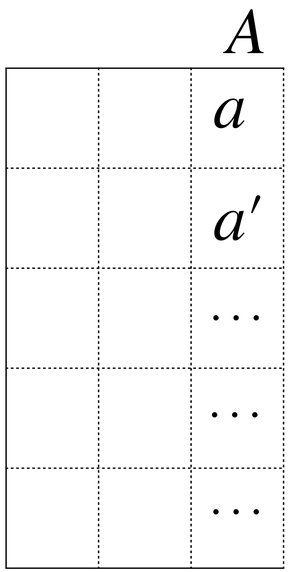
$$a' \overset{?}{\geq} b \iff \cdots$$

Compute independent ANDs
(1 round of communication)

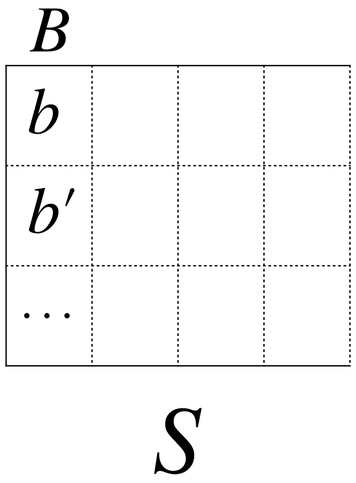
MESSAGE BATCHING EXAMPLE

$\ell = 4$ *bits*

$$\underbrace{a_3a_2a_1a_0}_{a} \quad \underbrace{b_3b_2b_1b_0}_{b} \quad \underbrace{a'_3a'_2a'_1a'_0}_{a'} \quad \underbrace{b'_3b'_2b'_1b'_0}_{b'}$$



$\bowtie_{R.A \geq S.B}$



$$a \overset{?}{\geq} b \iff c_1 \wedge a_3 \oplus c_2 \wedge c_3 \wedge a_2 \oplus c_4 \wedge c_5 \wedge c_6 \wedge a_1 \oplus c_4 \wedge c_5 \wedge c_7 \wedge (c_8 \wedge b_0)$$

$$a \overset{?}{\geq} b' \iff c'_1 \wedge a_3 \oplus c'_2 \wedge c'_3 \wedge a_2 \oplus c'_4 \wedge c'_5 \wedge c'_6 \wedge a_1 \oplus c'_4 \wedge c'_5 \wedge c'_7 \wedge (c_8 \wedge b'_0)$$

$$a' \overset{?}{\geq} b \iff \cdots$$

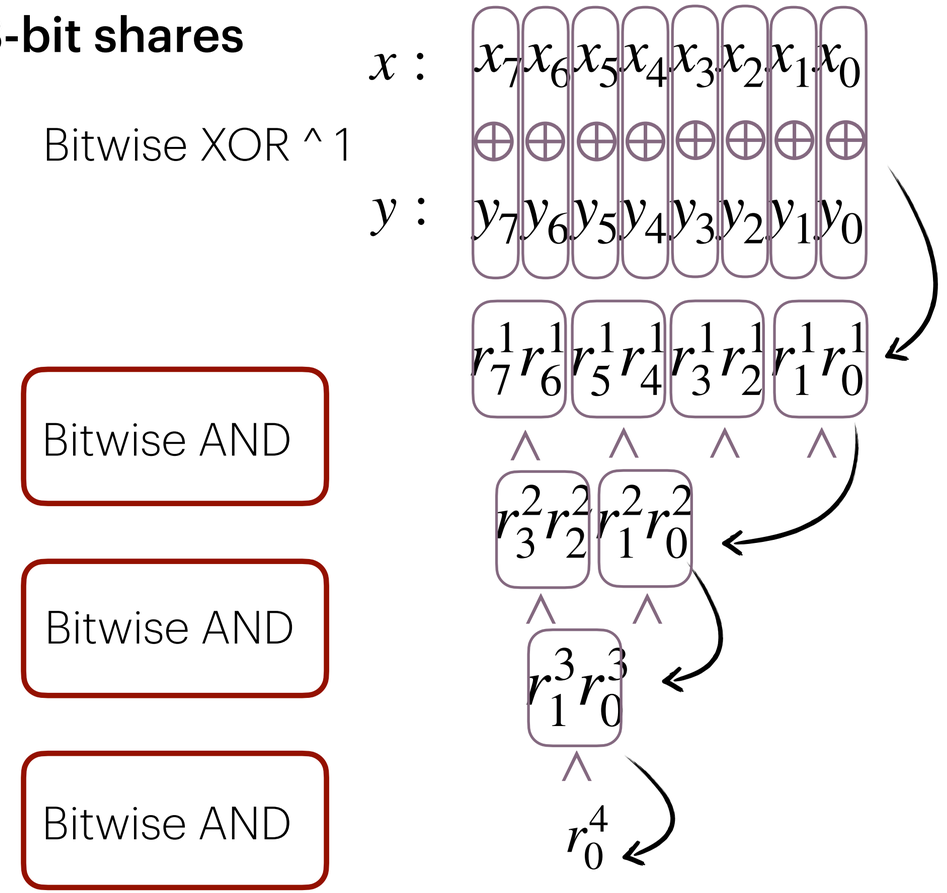
$\log \ell + 1$ communication rounds in total (for the whole join)

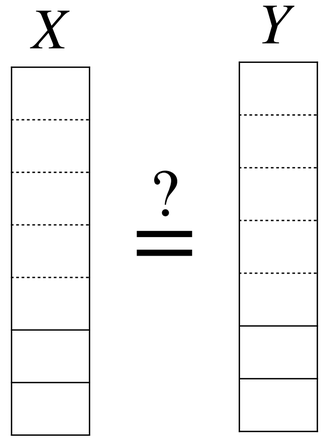
Independent on the cardinality of R and S

EXAMPLE: EQUALITY PRIMITIVE

x and y are equal if their bit representation is identical

For 8-bit shares



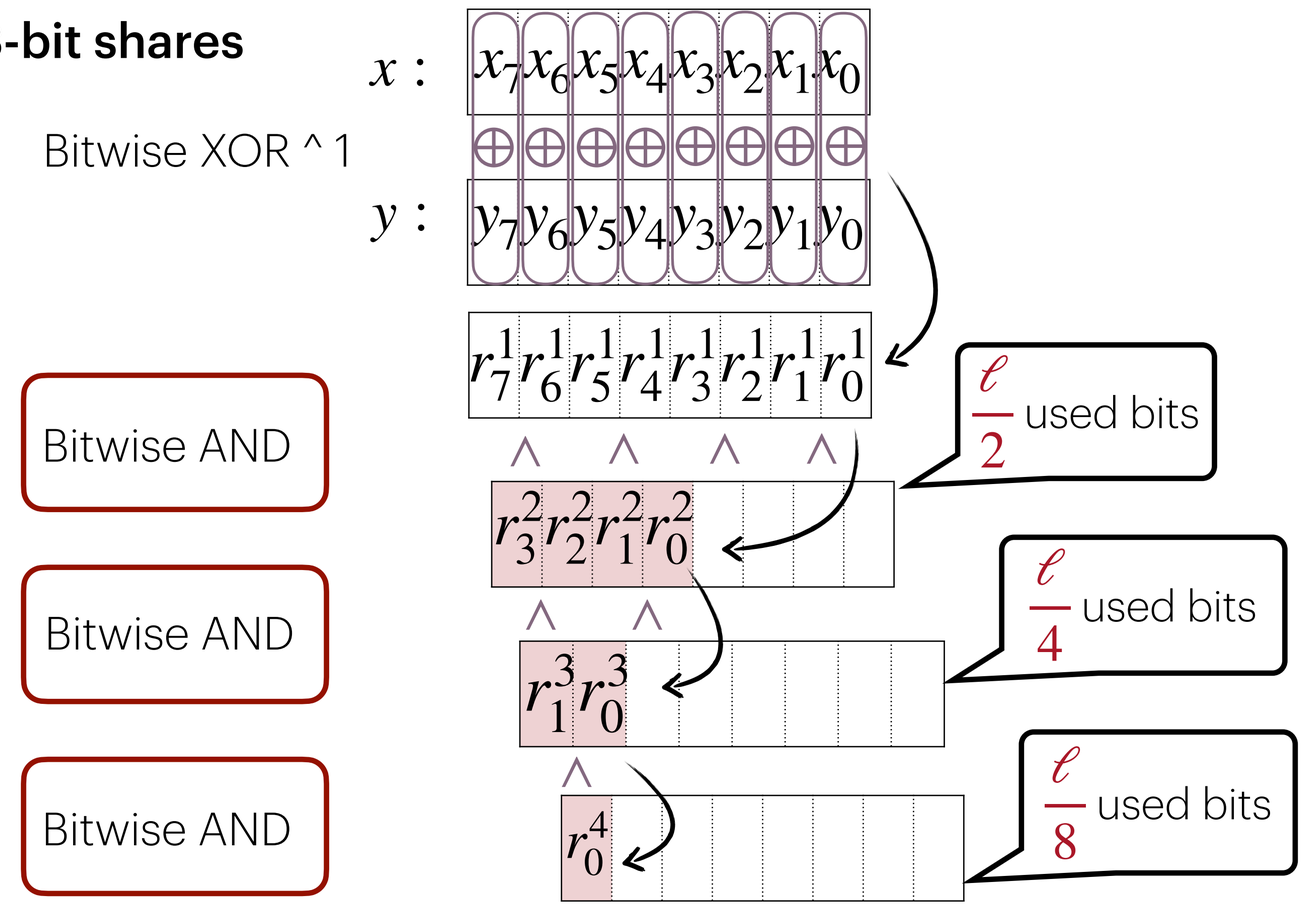


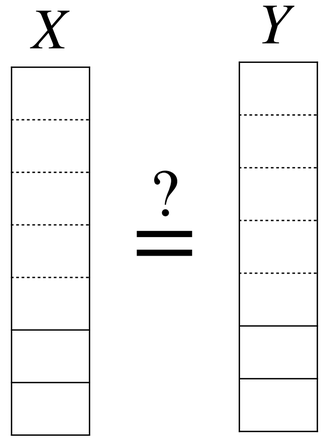
With vectorized AND, we can apply equality on entire vectors in $O(log\ell)$ rounds,
where ℓ is the share length in bits

EXAMPLE: EQUALITY PRIMITIVE

x and y are equal if their bit representation is identical

For 8-bit shares



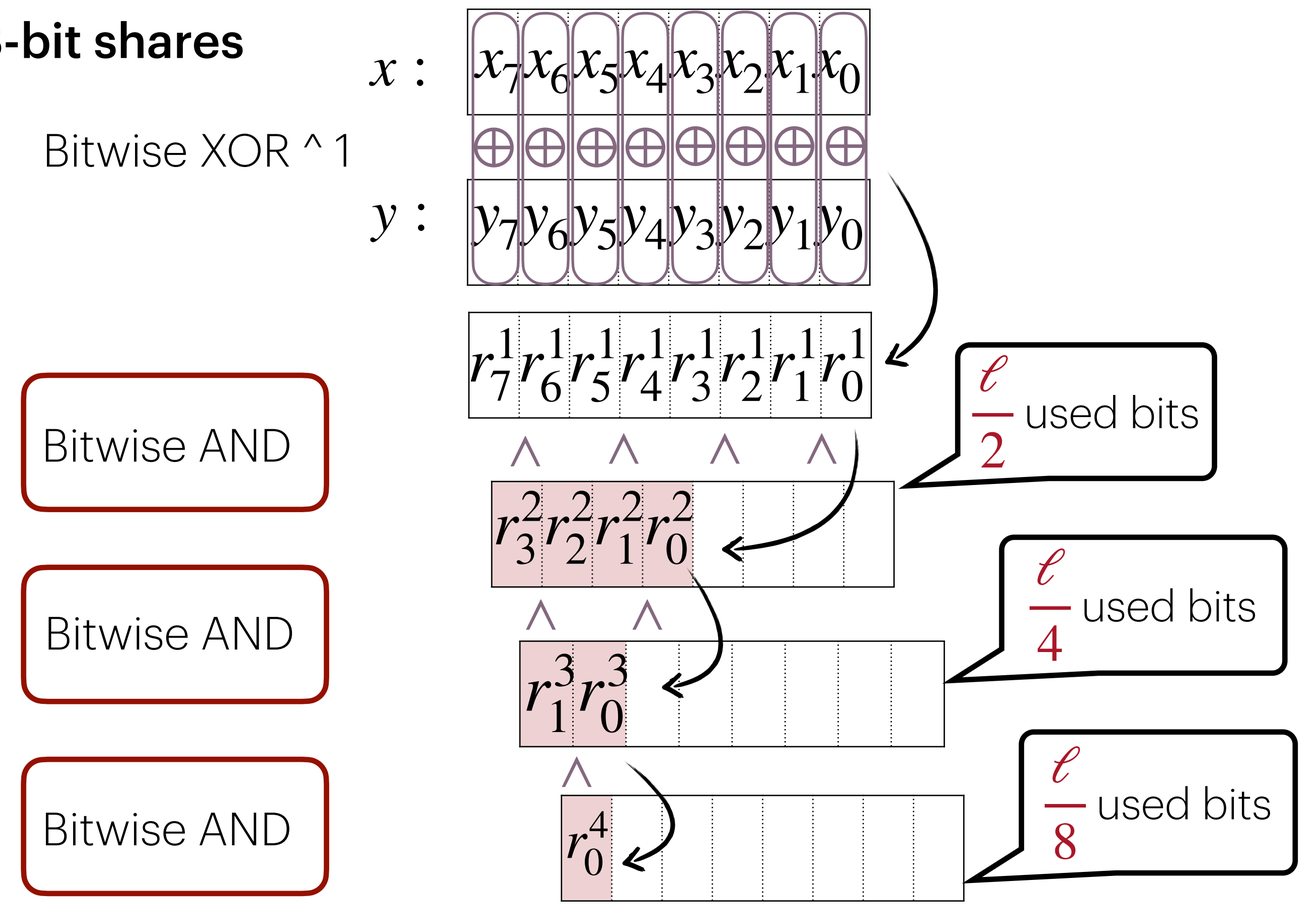


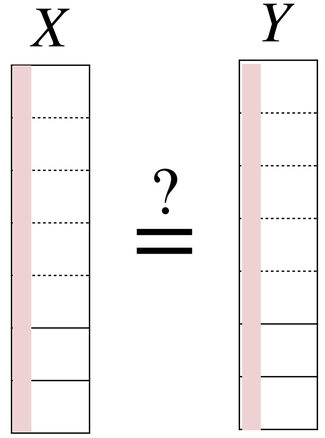
At each round, data are stored in elements of ℓ bits, but only a subset of those contain useful information

EXAMPLE: EQUALITY PRIMITIVE

x and y are equal if their bit representation is identical

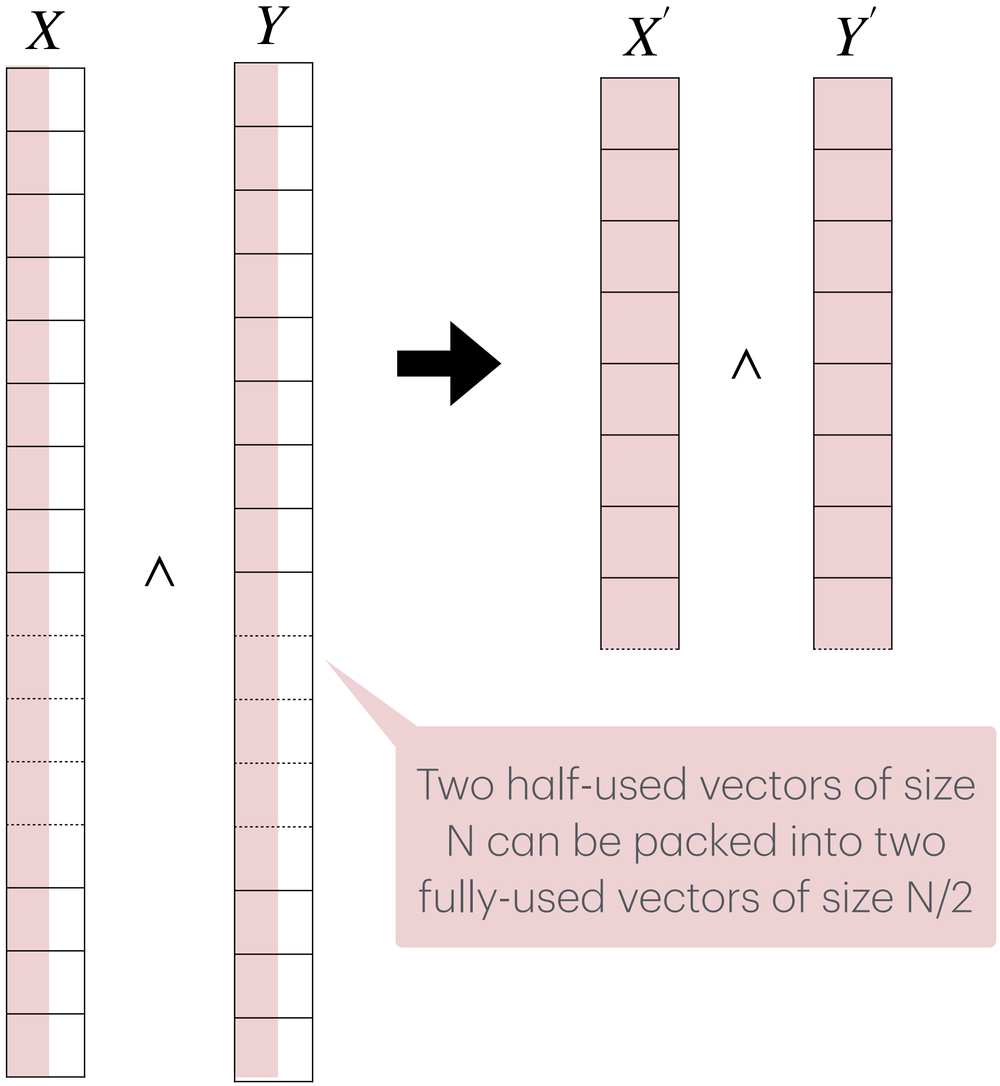
For 8-bit shares





Naively applying the vectorized AND primitive would **waste computation and bandwidth**

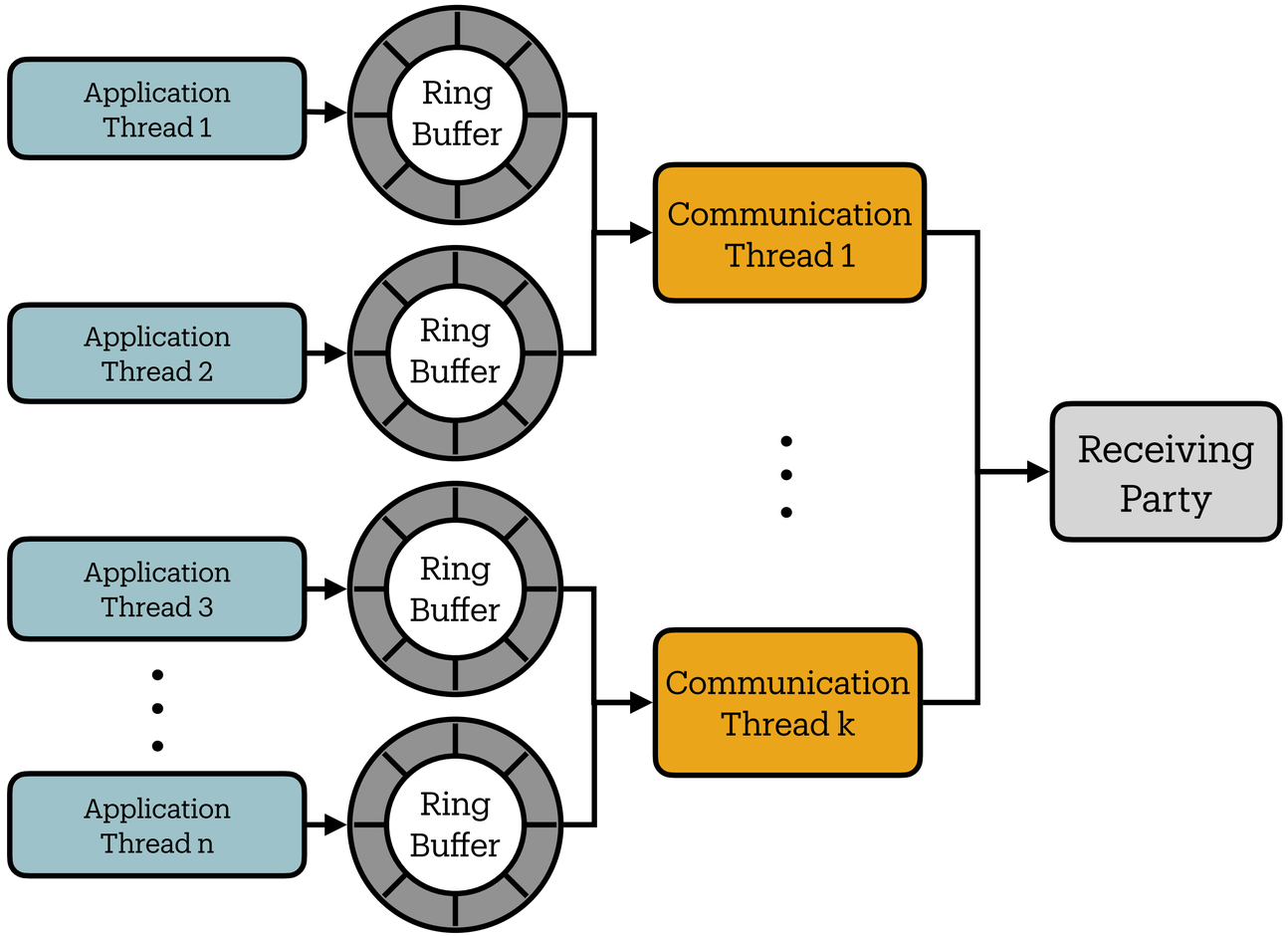
SOLUTION: BIT PACKING



- Bits from independent elements are packed together for efficient vectorized **computation** and batched **communication**
- Each parallel thread can easily perform its own bit (un)packing **without coordination**

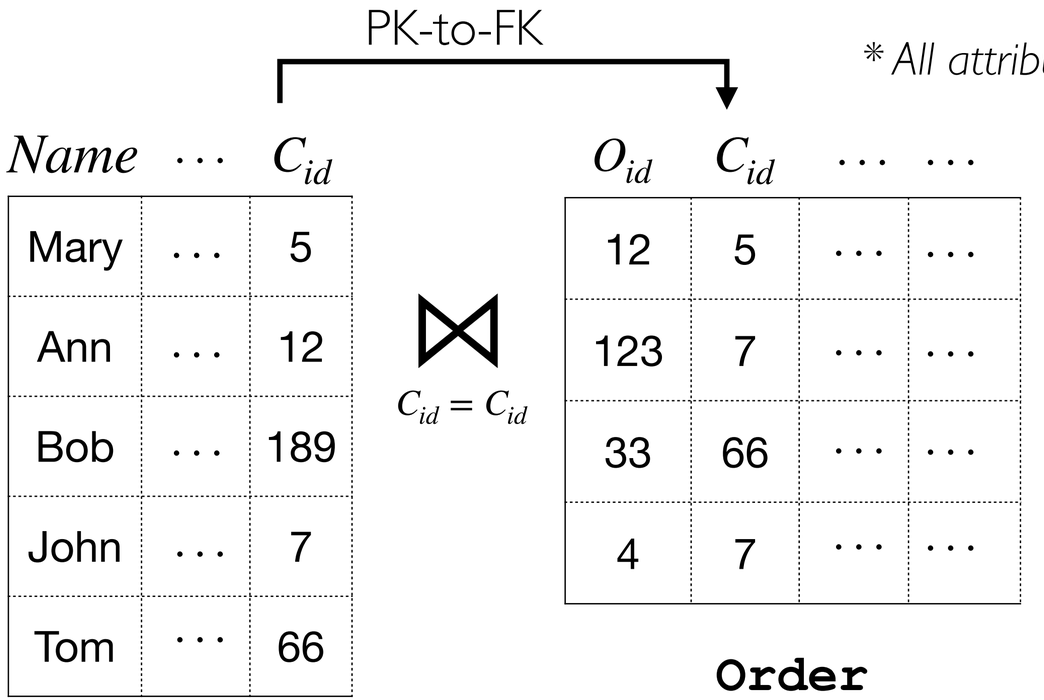
CUSTOM NO-COPY COMMUNICATOR

- **Dedicated communication threads** for send and receive
- **Lock-free ring buffers** between application and communication threads
- **Configurable** number of communication threads and **parallel TCP connections**
- WAN deployments benefit from more connections



Addressing the cascading effect in oblivious relational analytics

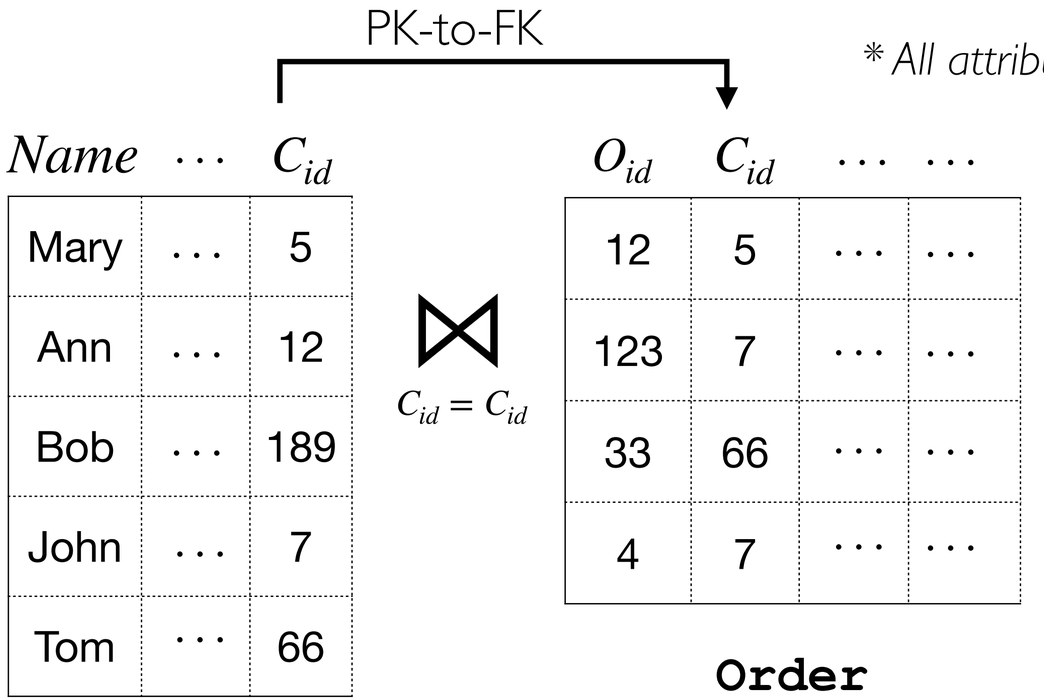
EVALUATING JOINs USING SORT AND A PREFIX NETWORK



* *All attributes are secret-shared*

```
SELECT O.Oid
FROM Customer as C, Orders as O
WHERE C.Cid = O.Cid AND C.Name=John
```

EVALUATING JOINs USING SORT AND A PREFIX NETWORK



R_{id}	Name	C_{id}	O_{id}
C	Mary	5	
C	Ann	12	
C	Bob	189	
C	John	7	
C	Tom	66	
O		5	12
O		7	123
O		66	33
O		7	4

$S \cup T$

```
SELECT O.Oid
FROM Customer as C, Orders as O
WHERE C.Cid = O.Cid AND C.Name=John
```

EVALUATING JOINs USING SORT AND A PREFIX NETWORK

PK-to-FK

* *All attributes are secret-shared*

$Name$	...	C_{id}
Mary	...	5
Ann	...	12
Bob	...	189
John	...	7
Tom	...	66

Customer

$\bowtie$ $C_{id} = C_{id}$

O_{id}	C_{id}	...	...
12	5	...	...
123	7	...	...
33	66	...	...
4	7	...	...

Order

```
SELECT O.Oid
FROM Customer as C, Orders as O
WHERE C.Cid = O.Cid AND C.Name=John
```

Padded attributes

R_{id}	$Name$	C_{id}	O_{id}
C	Mary	5	
C	Ann	12	
C	Bob	189	
C	John	7	
C	Tom	66	
O		5	12
O		7	123
O		66	33
O		7	4

$S \cup T$

EVALUATING JOINs USING SORT AND A PREFIX NETWORK

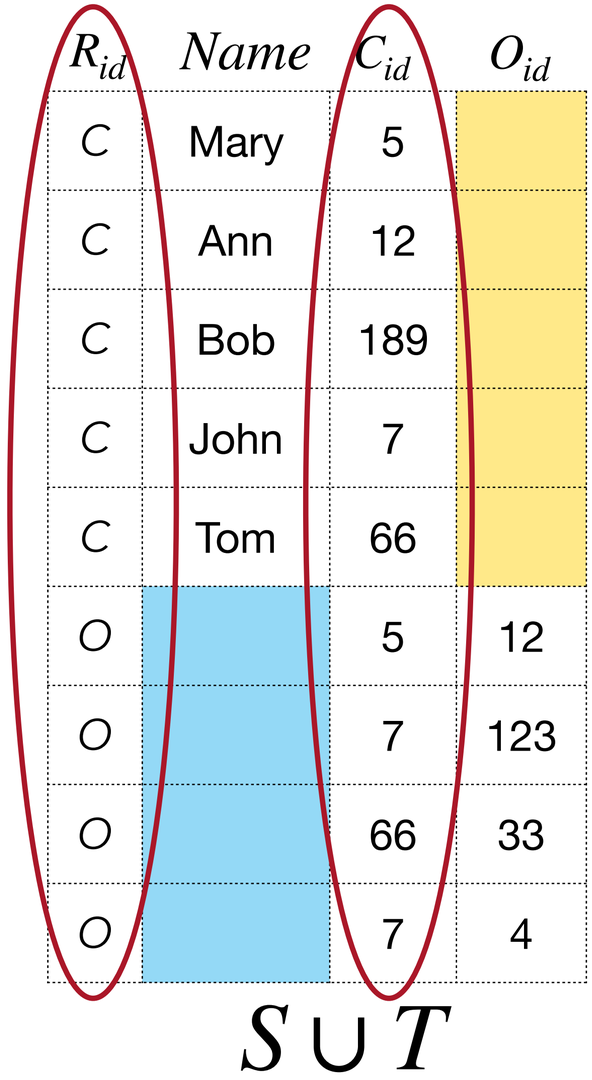
** All attributes are secret-shared*

R_{id}	$Name$	C_{id}	O_{id}
C	Mary	5	
C	Ann	12	
C	Bob	189	
C	John	7	
C	Tom	66	
O		5	12
O		7	123
O		66	33
O		7	4

$S \cup T$

EVALUATING JOINs USING SORT AND A PREFIX NETWORK

* *All attributes are secret-shared*



R_{id}	$Name$	C_{id}	O_{id}
C	Mary	5	
C	Ann	12	
C	Bob	189	
C	John	7	
C	Tom	66	
O		5	12
O		7	123
O		66	33
O		7	4

$S \cup T$

Sort on C_{id}, R_{id}

$O(n \log n)$ operations

$O(\log n)$ rounds

$O(n)$ space

$n = |S \cup T|$

EVALUATING JOINs USING SORT AND A PREFIX NETWORK

** All attributes are secret-shared*

R_{id}	$Name$	C_{id}	O_{id}
C	Mary	5	
C	Ann	12	
C	Bob	189	
C	John	7	
C	Tom	66	
O		5	12
O		7	123
O		66	33
O		7	4

$S \cup T$

R_{id}	$Name$	C_{id}	O_{id}
O		5	12
C	Mary	5	
O		7	123
O		7	4
C	John	7	
C	Ann	12	
O		66	33
C	Tom	66	
C	Bob	189	

$S \cup T$

EVALUATING JOINs USING SORT AND A PREFIX NETWORK

** All attributes are secret-shared*

R_{id}	$Name$	C_{id}	O_{id}
C	Mary	5	
C	Ann	12	
C	Bob	189	
C	John	7	
C	Tom	66	
O		5	12
O		7	123
O		66	33
O		7	4

$S \cup T$

R_{id}	$Name$	C_{id}	O_{id}
O		5	12
C	Mary	5	
O		7	**123**
O		7	**4**
C	John	7	
C	Ann	12	
O		66	33
C	Tom	66	
C	Bob	189	

$S \cup T$

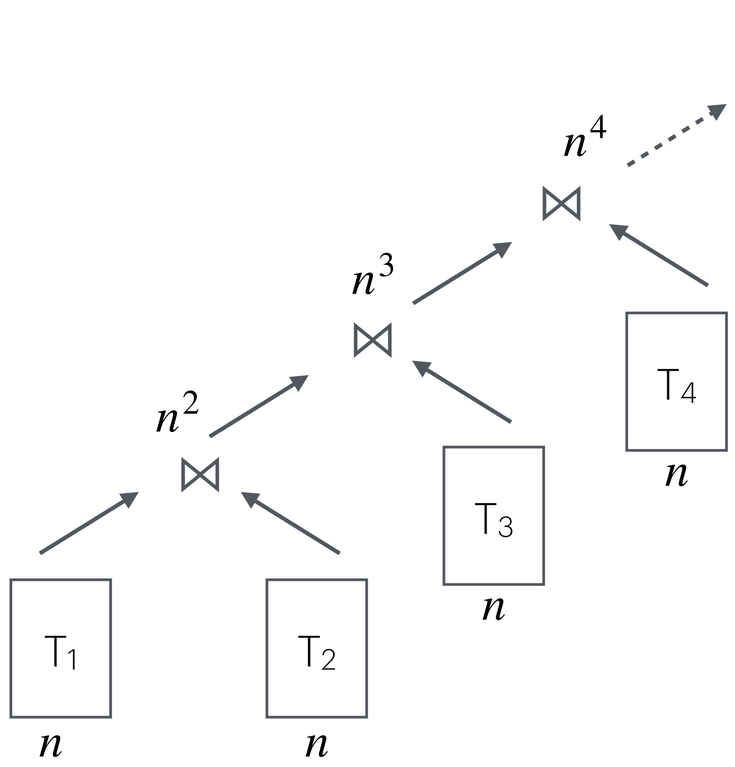
Apply prefix network to find matched pairs of rows and mask the rest

$O(n \log n)$ operations

$O(\log n)$ rounds $\quad n = |S \cup T|$

$O(n)$ space

AVOIDING THE CASCADING EFFECT IN QUERIES WITH NO PK-FK CONSTRAINTS

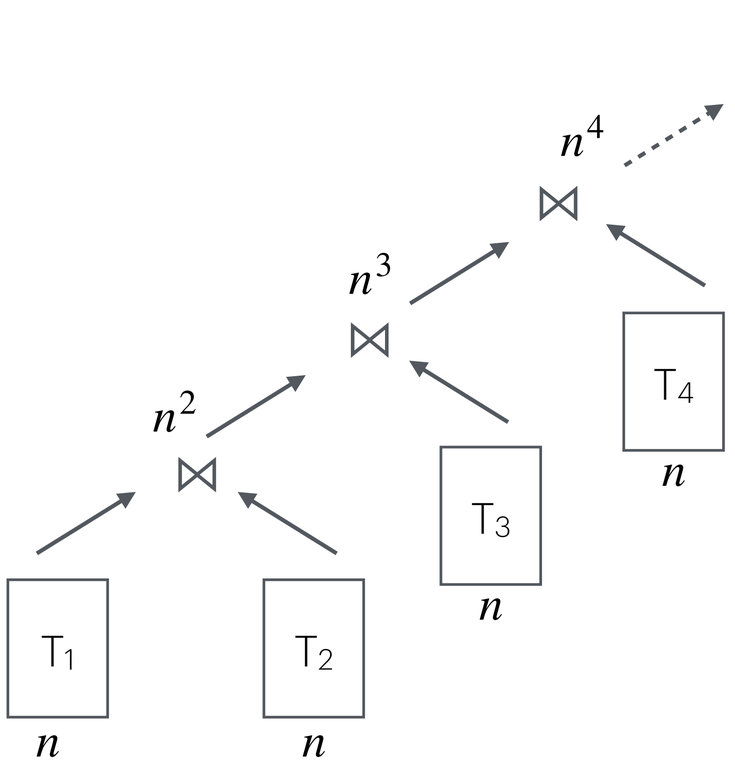


Joining k tables of n rows each would require $O(n^k)$ operations under MPC

MPC queries typically return **aggregated** results

- The query result size is bounded by the size of the aggregated result (e.g., the number of groups)
- **Interesting fact**: this gives us an $O(n)$ size bound for a broad class of analytics, including all queries used in prior MPC works
- We leverage this bound to avoid computing the cartesian product and evaluate the entire query in $O(n \log n)$

AVOIDING THE CASCADING EFFECT IN QUERIES WITH NO PK-FK CONSTRAINTS



Joining k tables of n rows each would require $O(n^k)$ operations under MPC

MPC queries typically return **aggregated** results

- The query result size is bounded by the size of the aggregated result (e.g., the number of groups)
- **Interesting fact**: this gives us an $O(n)$ size bound for a broad class of analytics, including all queries used in prior MPC works
- We leverage this bound to avoid computing the cartesian product and evaluate the entire query in $O(n \log n)$

Key idea: Apply the aggregation incrementally to bound the size of intermediate results

Example: TPC-H Q3

```
select
        l_orderkey,
        sum(l_extendedprice*(1-l_discount)) as revenue,
        o_orderdate,
        o_shippriority
from
        customer,
        orders,
        lineitem
where
        c_mktsegment = '[SEGMENT]'
        and c_custkey = o_custkey
        and l_orderkey = o_orderkey
        and o_orderdate < date '[DATE]'
        and l_shipdate > date '[DATE]'
group by
        l_orderkey,
        o_orderdate,
        o_shippriority
order by
        revenue desc,
        o_orderdate;
```

Custom aggregation per group

3 input tables

2 joins and 3 filters

1 group-by and 1 order-by on the join output

The query retrieves the shipping priority and potential revenue of the orders having the largest revenue among those that had not been shipped as of a given date. Orders are listed in decreasing order of revenue.

TPC-H Benchmark specification:
https://www.tpc.org/TPC_Documents_Current_Versions/pdf/TPC-H_v3.0.1.pdf

Example: TPC-H Q3

SQL

```
select
        l_orderkey,
        sum(l_extendedprice*(1-l_discount)) as revenue,
        o_orderdate,
        o_shippriority
from
        customer,
        orders,
        lineitem
where
        c_mktsegment = '[SEGMENT]'
        and c_custkey = o_custkey
        and l_orderkey = o_orderkey
        and o_orderdate < date '[DATE]'
        and l_shipdate > date '[DATE]'
group by
        l_orderkey,
        o_orderdate,
        o_shippriority
order by
        revenue desc,
        o_orderdate;
```

ORQ API

```
// Initialize C (Customers), O (Orders), LI (Lineitem)
...

// Apply filters
C.filter("MktSegment" == SEGMENT);
O.filter("OrdDate" < DATE);
LI.filter("ShipDate" > DATE);

// Calculate the revenue
LI["Revenue"] = LI["Price"]*(100-LI["Discount"])/100;

// Joins and group-by with aggregation
auto RES = C.inner_join(O, {"CustKey"})
            .inner_join(LI, {"OrdKey"}, {
                {"OrdDate","OrdDate",   copy},
                {"Priority","Priority", copy}})
            .aggregate({"OrdKey","OrdDate","Priority"},
                {"Revenue","TotalRevenue", sum})
            .project({"OrdKey","TotalRevenue",
                      "OrdDate","Priority"})
            .sort({{"TotalRevenue", DESC},
                   {"OrdDate",      ASC}})
```

TPC-H Benchmark specification:
https://www.tpc.org/TPC_Documents_Current_Versions/pdf/TPC-H_v3.0.1.pdf

STEP-BY-STEP EVALUATION IN ORQ

Assuming no PK-FK constraints

```
select
        l_orderkey,
        sum(l_extendedprice*(1-l_discount)) as revenue,
        o_orderdate,
        o_shippriority
from
        customer,
        orders,
        lineitem
where
        c_mktsegment = '[SEGMENT]'
        and c_custkey = o_custkey
        and l_orderkey = o_orderkey
        and o_orderdate < date '[DATE]'
        and l_shipdate > date '[DATE]'
group by
        l_orderkey,
        o_orderdate,
        o_shippriority
order by
        revenue desc,
        o_orderdate;
```

LineItem (LI)

OrdKey	SDate	Revenue	V
1	2023	19	1
1	2018	24	0
5	2023	70	1
5	2024	80	1
5	2001	55	0

Customers (C)

CustKey	MktSeg	V
303	S	1
303	S	1
222	x	0
255	S	1
999	y	0

Orders (O)

OrdKey	CustKey	OrdDate	Priority	V
1	303	2021	1	1
2	255	2019	3	1
3	101	2023	2	0
4	303	2024	5	0
5	303	2020	6	1
5	255	2020	6	1

STEP-BY-STEP EVALUATION IN ORQ

Assuming no PK-FK constraints

```
select
        l_orderkey,
        sum(l_extendedprice*(1-l_discount)) as revenue,
        o_orderdate,
        o_shippriority
from
        customer,
        orders,
        lineitem
where
        c_mktsegment = '[SEGMENT]'
        and c_custkey = o_custkey
        and l_orderkey = o_orderkey
        and o_orderdate < date '[DATE]'
        and l_shipdate > date '[DATE]'
group by
        l_orderkey,
        o_orderdate,
        o_shippriority
order by
        revenue desc,
        o_orderdate;
```

LineItem (LI)

OrdKey	SDate	Revenue	V
1	2023	19	1
1	2018	24	0
5	2023	70	1
5	2024	80	1
5	2001	55	0

Customers (C)

CustKey	MktSeg	V
303	S	1
303	S	1
222	x	0
255	S	1
999	y	0

key

⋈ CustKey

Orders (O)

OrdKey	CustKey	OrdDate	Priority	V
1	303	2021	1	1
2	255	2019	3	1
3	101	2023	2	0
4	303	2024	5	0
5	303	2020	6	1
5	255	2020	6	1

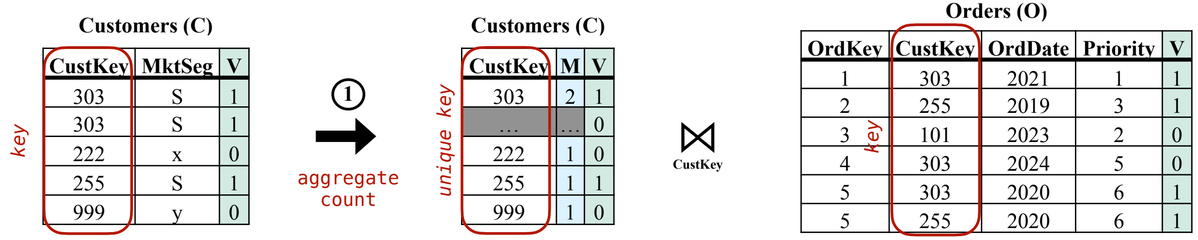
key

STEP-BY-STEP EVALUATION IN ORQ

```
select
        l_orderkey,
        sum(l_extendedprice*(1-l_discount)) as revenue,
        o_orderdate,
        o_shippriority
from
        customer,
        orders,
        lineitem
where
        c_mktsegment = '[SEGMENT]'
        and c_custkey = o_custkey
        and l_orderkey = o_orderkey
        and o_orderdate < date '[DATE]'
        and l_shipdate > date '[DATE]'
group by
        l_orderkey,
        o_orderdate,
        o_shippriority
order by
        revenue desc,
        o_orderdate;
```

LineItem (LI)

OrdKey	SDate	Revenue	V
1	2023	19	1
1	2018	24	0
5	2023	70	1
5	2024	80	1
5	2001	55	0



STEP-BY-STEP EVALUATION IN ORQ

```
select
        l_orderkey,
        sum(l_extendedprice*(1-l_discount)) as revenue,
        o_orderdate,
        o_shippriority
from
        customer,
        orders,
        lineitem
where
        c_mktsegment = '[SEGMENT]'
        and c_custkey = o_custkey
        and l_orderkey = o_orderkey
        and o_orderdate < date '[DATE]'
        and l_shipdate > date '[DATE]'
group by
        l_orderkey,
        o_orderdate,
        o_shippriority
order by
        revenue desc,
        o_orderdate;
```

Customers ⋈ Orders (CO)

OrdKey	CustKey	OrdDate	Priority	M	V
…	…	…	…		0
…	…	…	…		0
5	255	2020	6	1	1
5	303	2020	6	2	1
1	303	2021	1	2	1

LineItem (LI)

OrdKey	SDate	Revenue	V
1	2023	19	1
1	2018	24	0
5	2023	70	1
5	2024	80	1
5	2001	55	0

Customers (C)

CustKey	MktSeg	V
303	S	1
303	S	1
222	x	0
255	S	1
999	y	0

key

① aggregate count

Customers (C)

CustKey	M	V
303	2	1
…	…	0
222	1	0
255	1	1
999	1	0

unique key

② ⋈ CustKey

Orders (O)

OrdKey	CustKey	OrdDate	Priority	V
1	303	2021	1	1
2	255	2019	3	1
3	101	2023	2	0
4	303	2024	5	0
5	303	2020	6	1
5	255	2020	6	1

key

STEP-BY-STEP EVALUATION IN ORQ

```
select
        l_orderkey,
        sum(l_extendedprice*(1-l_discount)) as revenue,
        o_orderdate,
        o_shippriority
from
        customer,
        orders,
        lineitem
where
        c_mktsegment = '[SEGMENT]'
        and c_custkey = o_custkey
        and l_orderkey = o_orderkey
        and o_orderdate < date '[DATE]'
        and l_shipdate > date '[DATE]'
group by
        l_orderkey,
        o_orderdate,
        o_shippriority
order by
        revenue desc,
        o_orderdate;
```

Customers ⋈ Orders (CO)

	OrdKey	CustKey	OrdDate	Priority	M	V
key	…	…	…	…		0
key	…	…	…	…		0
key	5	255	2020	6	1	1
key	5	303	2020	6	2	1
key	1	303	2021	1	2	1

⋈ OrdKey

LineItem (LI)

	OrdKey	SDate	Revenue	V
key	1	2023	19	1
key	1	2018	24	0
key	5	2023	70	1
key	5	2024	80	1
key	5	2001	55	0

Customers (C)

	CustKey	MktSeg	V
key	303	S	1
key	303	S	1
key	222	x	0
key	255	S	1
key	999	y	0

① aggregate count

Customers (C)

	CustKey	M	V
unique key	303	2	1
	…	…	0
unique key	222	1	0
unique key	255	1	1
unique key	999	1	0

② ⋈ CustKey

Orders (O)

	OrdKey	CustKey	OrdDate	Priority	V
	1	303	2021	1	1
	2	255	2019	3	1
key	3	101	2023	2	0
	4	303	2024	5	0
	5	303	2020	6	1
	5	255	2020	6	1

STEP-BY-STEP EVALUATION IN ORQ

```
select
        l_orderkey,
        sum(l_extendedprice*(1-l_discount)) as revenue,
        o_orderdate,
        o_shippriority
from
        customer,
        orders,
        lineitem
where
        c_mktsegment = '[SEGMENT]'
        and c_custkey = o_custkey
        and l_orderkey = o_orderkey
        and o_orderdate < date '[DATE]'
        and l_shipdate > date '[DATE]'
group by
        l_orderkey,
        o_orderdate,
        o_shippriority
order by
        revenue desc,
        o_orderdate;
```

Customers ⋈ Orders (CO)

key

OrdKey	CustKey	OrdDate	Priority	M	V
…	…	…	…		0
…	…	…	…		0
5	255	2020	6	1	1
5	303	2020	6	2	1
1	303	2021	1	2	1

⋈ OrdKey

LineItem (LI)

unique key

OrdKey	SDate	Revenue	RevPre	V
1	2023	19	19	1
…	…	…	…	0
5	2023	70	150	1
…	…	…	…	0
…	…	…	…	0

③ aggregate sum

LineItem (LI)

key

OrdKey	SDate	Revenue	V
1	2023	19	1
1	2018	24	0
5	2023	70	1
5	2024	80	1
5	2001	55	0

Customers (C)

key

CustKey	MktSeg	V
303	S	1
303	S	1
222	x	0
255	S	1
999	y	0

① aggregate count

Customers (C)

unique key

CustKey	M	V
303	2	1
…	…	0
222	1	0
255	1	1
999	1	0

② ⋈ CustKey

Orders (O)

key

OrdKey	CustKey	OrdDate	Priority	V
1	303	2021	1	1
2	255	2019	3	1
3	101	2023	2	0
4	303	2024	5	0
5	303	2020	6	1
5	255	2020	6	1

STEP-BY-STEP EVALUATION IN ORQ

```
select
        l_orderkey,
        sum(l_extendedprice*(1-l_discount)) as revenue,
        o_orderdate,
        o_shippriority
from
        customer,
        orders,
        lineitem
where
        c_mktsegment = '[SEGMENT]'
        and c_custkey = o_custkey
        and l_orderkey = o_orderkey
        and o_orderdate < date '[DATE]'
        and l_shipdate > date '[DATE]'
group by
        l_orderkey,
        o_orderdate,
        o_shippriority
order by
        revenue desc,
        o_orderdate;
```

Customers ⋈ Orders (CO)

key

OrdKey	CustKey	OrdDate	Priority	M	V
…	…	…	…		0
…	…	…	…		0
5	255	2020	6	1	1
5	303	2020	6	2	1
1	303	2021	1	2	1

④ ⋈ OrdKey

LineItem (LI)

unique key

OrdKey	SDate	Revenue	RevPre	V
1	2023	19	19	1
…	…	…	…	0
5	2023	70	150	1
…	…	…	…	0
…	…	…	…	0

③ aggregate sum

LineItem (LI)

key

OrdKey	SDate	Revenue	V
1	2023	19	1
1	2018	24	0
5	2023	70	1
5	2024	80	1
5	2001	55	0

Customers (C)

key

CustKey	MktSeg	V
303	S	1
303	S	1
222	x	0
255	S	1
999	y	0

① aggregate count

Customers (C)

unique key

CustKey	M	V
303	2	1
…	…	0
222	1	0
255	1	1
999	1	0

② ⋈ CustKey

Orders (O)

key

OrdKey	CustKey	OrdDate	Priority	V
1	303	2021	1	1
2	255	2019	3	1
3	101	2023	2	0
4	303	2024	5	0
5	303	2020	6	1
5	255	2020	6	1

STEP-BY-STEP EVALUATION IN ORQ

```
select
        l_orderkey,
        sum(l_extendedprice*(1-l_discount)) as revenue,
        o_orderdate,
        o_shippriority
from
        customer,
        orders,
        lineitem
where
        c_mktsegment = '[SEGMENT]'
        and c_custkey = o_custkey
        and l_orderkey = o_orderkey
        and o_orderdate < date '[DATE]'
        and l_shipdate > date '[DATE]'
group by
        l_orderkey,
        o_orderdate,
        o_shippriority
order by
        revenue desc,
        o_orderdate;
```

CO ⋈ LI (COL)

OrdKey	OrdDate	Priority	RevPre	M	V
…	…	…	…		0
1	NULL	NULL	19	NULL	0
1	2021	1	19	2	1
5	NULL	NULL	150	NULL	0
5	2020	6	150	1	1
5	2020	6	150	2	1

Customers ⋈ Orders (CO)

key

OrdKey	CustKey	OrdDate	Priority	M	V
…	…	…	…		0
…	…	…	…		0
5	255	2020	6	1	1
5	303	2020	6	2	1
1	303	2021	1	2	1

④ ⋈ OrdKey

LineItem (LI)

unique key

OrdKey	SDate	Revenue	RevPre	V
1	2023	19	19	1
…	…	…	…	0
5	2023	70	150	1
…	…	…	…	0
…	…	…	…	0

③ aggregate sum

LineItem (LI)

key

OrdKey	SDate	Revenue	V
1	2023	19	1
1	2018	24	0
5	2023	70	1
5	2024	80	1
5	2001	55	0

Customers (C)

key

CustKey	MktSeg	V
303	S	1
303	S	1
222	x	0
255	S	1
999	y	0

① aggregate count

Customers (C)

unique key

CustKey	M	V
303	2	1
…	…	0
222	1	0
255	1	1
999	1	0

② ⋈ CustKey

Orders (O)

key

OrdKey	CustKey	OrdDate	Priority	V
1	303	2021	1	1
2	255	2019	3	1
3	101	2023	2	0
4	303	2024	5	0
5	303	2020	6	1
5	255	2020	6	1

STEP-BY-STEP EVALUATION IN ORQ

```
select
        l_orderkey,
        sum(l_extendedprice*(1-l_discount)) as revenue,
        o_orderdate,
        o_shippriority
from
        customer,
        orders,
        lineitem
where
        c_mktsegment = '[SEGMENT]'
        and c_custkey = o_custkey
        and l_orderkey = o_orderkey
        and o_orderdate < date '[DATE]'
        and l_shipdate > date '[DATE]'
group by
        l_orderkey,
        o_orderdate,
        o_shippriority
order by
        revenue desc,
        o_orderdate;
```

CO ⋈ LI (COL)

keys

OrdKey	OrdDate	Priority	RevPre	M	V
…	…	…	…		0
1	NULL	NULL	19	NULL	0
1	2021	1	19	2	1
5	NULL	NULL	150	NULL	0
5	2020	6	150	1	1
5	2020	6	150	2	1

copy, agg

⑤ aggregate sum

RESULT

OrdKey	OrdDate	Priority	TotalR	V
…	…	…	…	0
…	…	…	…	0
…	…	…	…	0
1	2021	1	38	1
5	2020	6	450	1

④ ⋈ OrdKey

Customers ⋈ Orders (CO)

key

OrdKey	CustKey	OrdDate	Priority	M	V
…	…	…	…		0
…	…	…	…		0
5	255	2020	6	1	1
5	303	2020	6	2	1
1	303	2021	1	2	1

LineItem (LI)

unique key

OrdKey	SDate	Revenue	RevPre	V
1	2023	19	19	1
…	…	…	…	0
5	2023	70	150	1
…	…	…	…	0
…	…	…	…	0

③ aggregate sum

LineItem (LI)

key

OrdKey	SDate	Revenue	V
1	2023	19	1
1	2018	24	0
5	2023	70	1
5	2024	80	1
5	2001	55	0

Customers (C)

key

CustKey	MktSeg	V
303	S	1
303	S	1
222	x	0
255	S	1
999	y	0

① aggregate count

Customers (C)

unique key

CustKey	M	V
303	2	1
…	…	0
222	1	0
255	1	1
999	1	0

② ⋈ CustKey

Orders (O)

key

OrdKey	CustKey	OrdDate	Priority	V
1	303	2021	1	1
2	255	2019	3	1
3	101	2023	2	0
4	303	2024	5	0
5	303	2020	6	1
5	255	2020	6	1

STEP-BY-STEP EVALUATION IN ORQ

```
select
        l_orderkey,
        sum(l_extendedprice*(1-l_discount)) as revenue,
        o_orderdate,
        o_shippriority
from
        customer,
        orders,
        lineitem
where
        c_mktsegment = '[SEGMENT]'
        and c_custkey = o_custkey
        and l_orderkey = o_orderkey
        and o_orderdate < date '[DATE]'
        and l_shipdate > date '[DATE]'
group by
        l_orderkey,
        o_orderdate,
        o_shippriority
order by
        revenue desc,
        o_orderdate;
```

CO ⋈ LI (COL)

keys

OrdKey	OrdDate	Priority	RevPre	M	V
…	…	…	…		0
1	NULL	NULL	19	NULL	0
1	2021	1	19	2	1
5	NULL	NULL	150	NULL	0
5	2020	6	150	1	1
5	2020	6	150	2	1

copy, agg

⑤ aggregate sum

RESULT

OrdKey	OrdDate	Priority	TotalR	V
…	…	…	…	0
…	…	…	…	0
…	…	…	…	0
1	2021	1	38	1
5	2020	6	450	1

④ ⋈ OrdKey

Customers ⋈ Orders (CO)

key

OrdKey	CustKey	OrdDate	Priority	M	V
…	…	…	…		0
…	…	…	…		0
5	255	2020	6	1	1
5	303	2020	6	2	1
1	303	2021	1	2	1

LineItem (LI)

unique key

OrdKey	SDate	Revenue	RevPre	V
1	2023	19	19	1
…	…	…	…	0
5	2023	70	150	1
…	…	…	…	0
…	…	…	…	0

③ aggregate sum

LineItem (LI)

key

OrdKey	SDate	Revenue	V
1	2023	19	1
1	2018	24	0
5	2023	70	1
5	2024	80	1
5	2001	55	0

Customers (C)

key

CustKey	MktSeg	V
303	S	1
303	S	1
222	x	0
255	S	1
999	y	0

① aggregate count

Customers (C)

unique key

CustKey	M	V
303	2	1
…	…	0
222	1	0
255	1	1
999	1	0

② ⋈ CustKey

Orders (O)

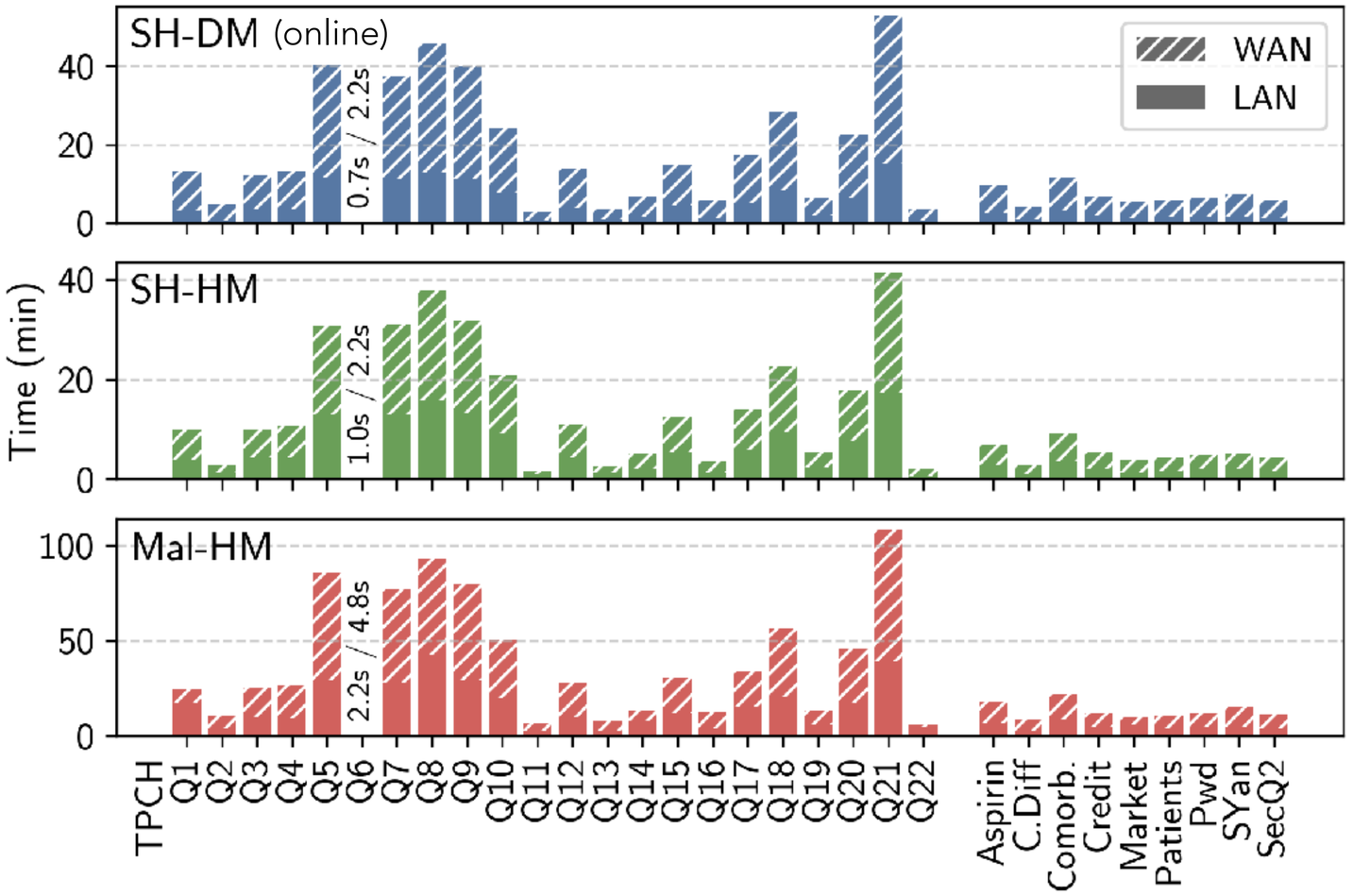
key

OrdKey	CustKey	OrdDate	Priority	V
1	303	2021	1	1
2	255	2019	3	1
3	101	2023	2	0
4	303	2024	5	0
5	303	2020	6	1
5	255	2020	6	1

The query result size is bounded by the size of Orders (O)

Result highlights

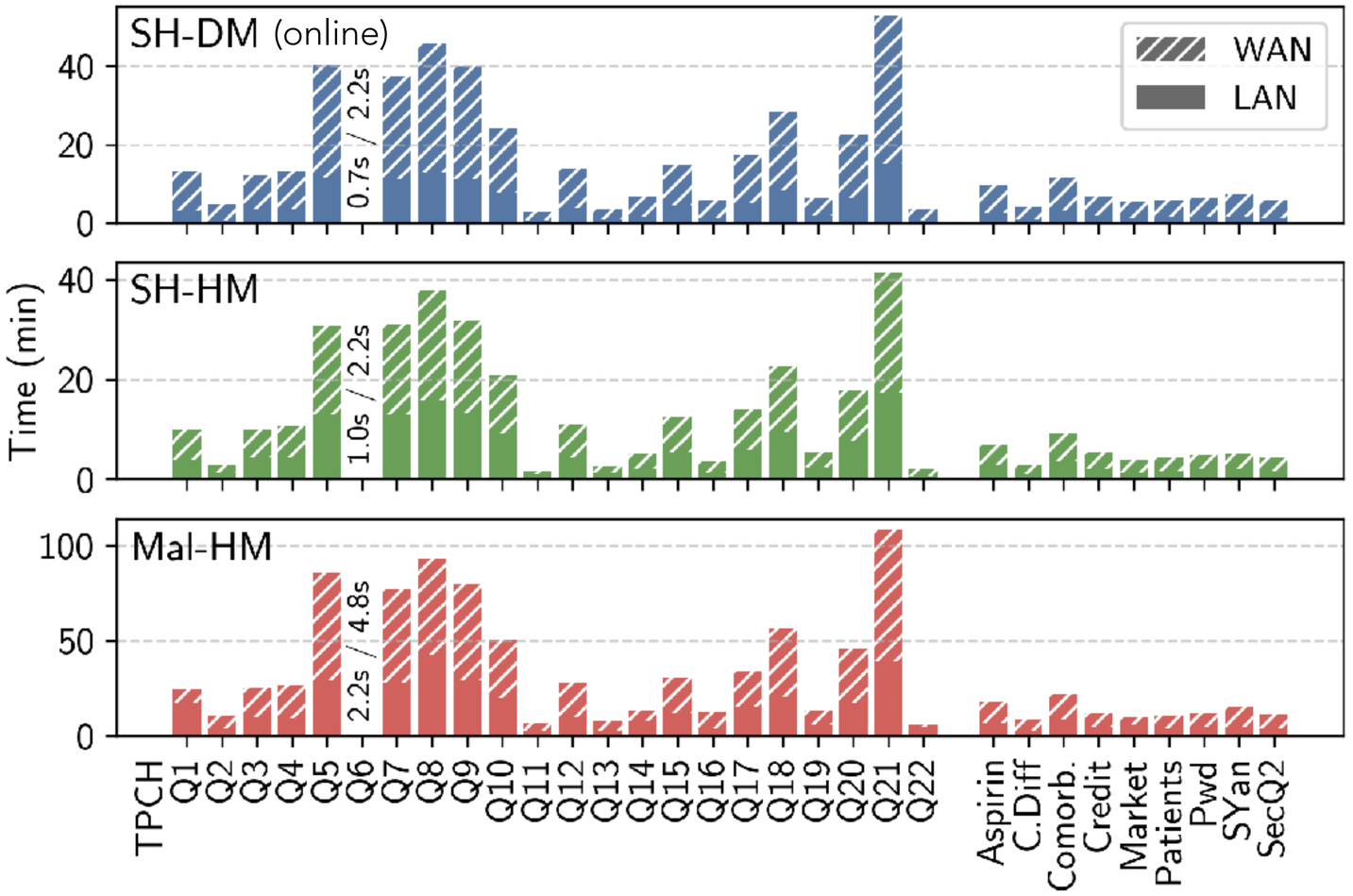
COMPLEX ANALYTICS UNDER MPC ARE PRACTICAL



- **31** queries with **multiple joins** and **aggregations**, including all TPC-H queries at **SF1 and SF10** (up to 85M input records – 110M+ intermediate records)
- **3 protocols** with semi-honest and malicious security: Demmler et al. (SH-DM), Araki et al. (SH-HM), Dalskov et al. (Mal-HM)

Each computing party uses one `c7a.16xlarge` AWS instance (64 vCPUs, 128GB memory)

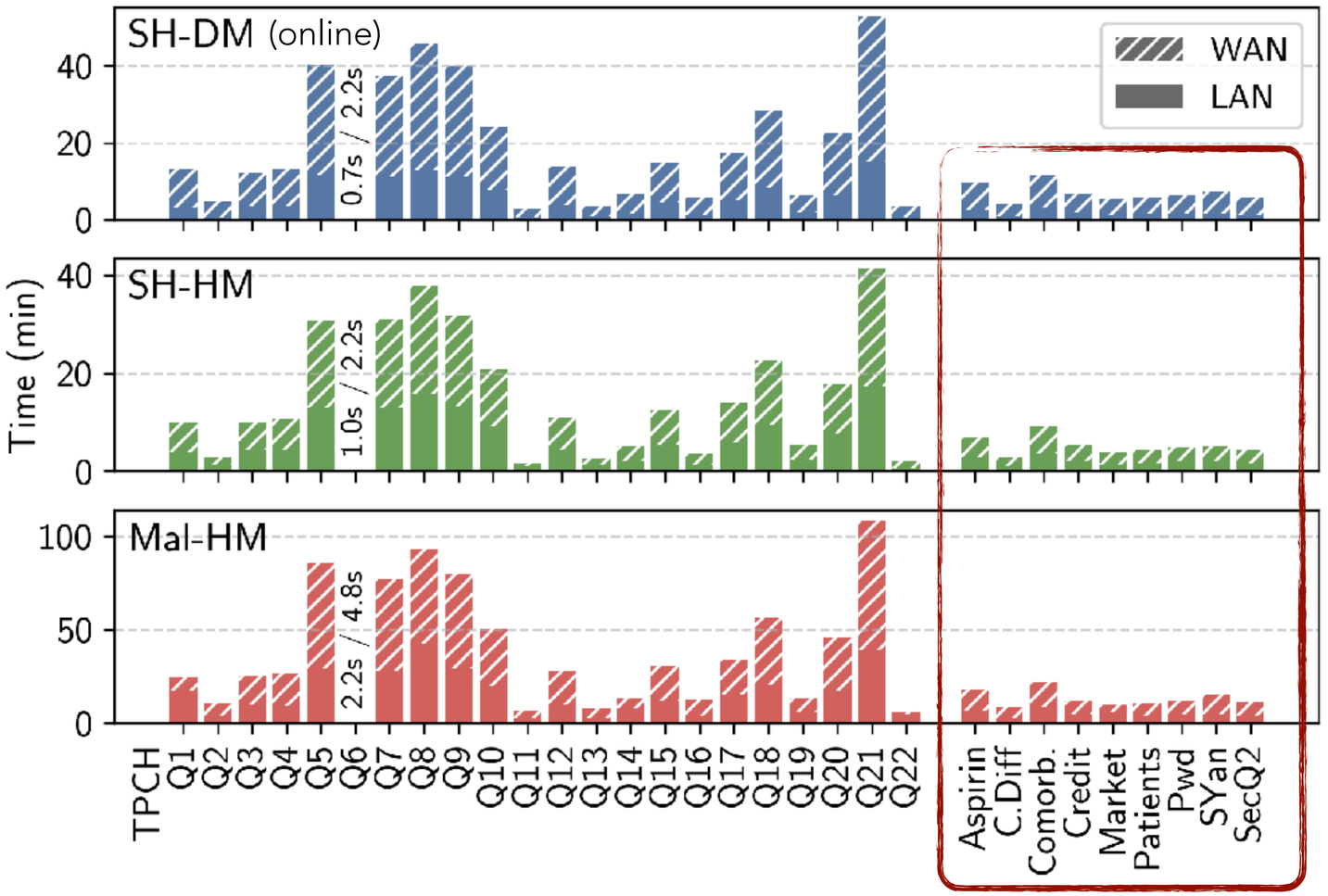
COMPLEX ANALYTICS UNDER MPC ARE PRACTICAL



- **31** queries with **multiple joins** and **aggregations**, including all TPC-H queries at **SF1 and SF10** (up to 85M input records – 110M+ intermediate records)
- **3 protocols** with semi-honest and malicious security: Demmler et al. (SH-DM), Araki et al. (SH-HM), Dalskov et al. (Mal-HM)
- The most expensive query, **Q21** (12 oblivious sort calls) completes in **42min**, under **malicious** security

Each computing party uses one `c7a.16xlarge` AWS instance (64 vCPUs, 128GB memory)

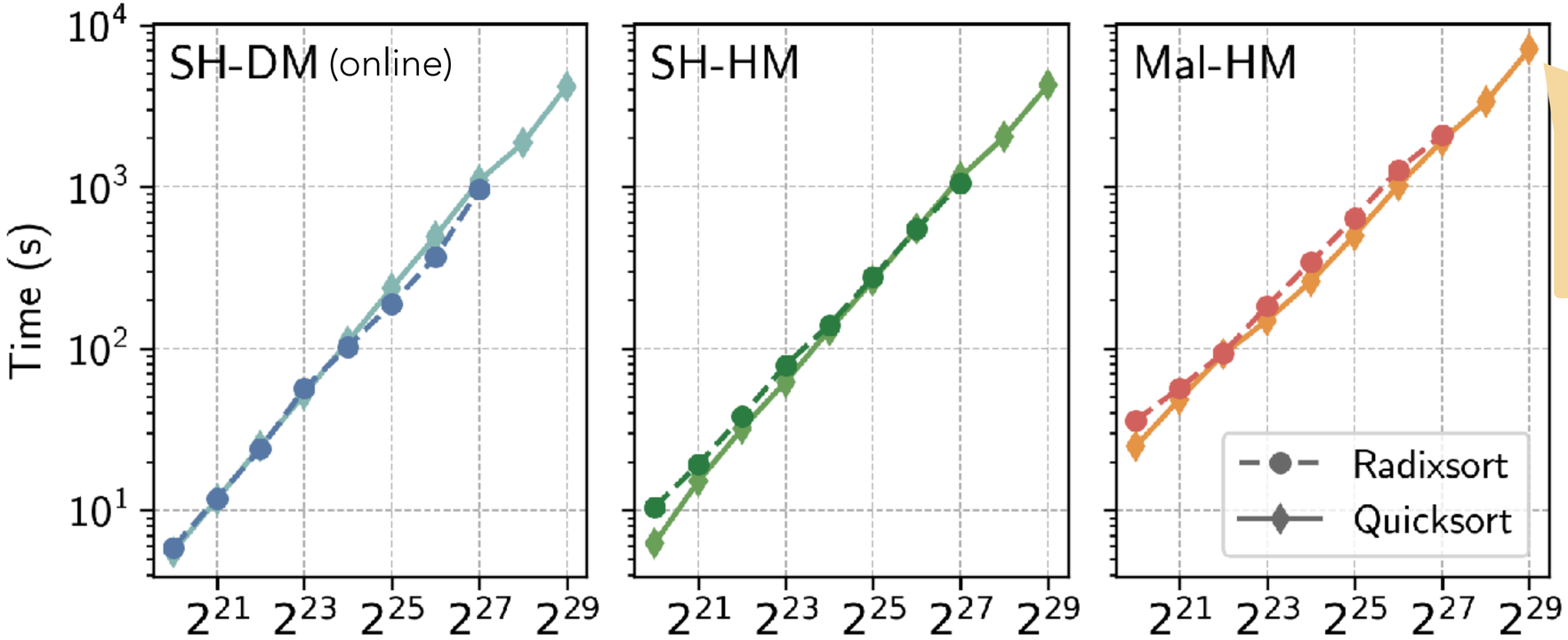
COMPLEX ANALYTICS UNDER MPC ARE PRACTICAL



- **31** queries with **multiple joins** and **aggregations**, including all TPC-H queries at **SF1 and SF10** (up to 85M input records – 110M+ intermediate records)
- **3 protocols** with semi-honest and malicious security: Demmler et al. (SH-DM), Araki et al. (SH-HM), Dalskov et al. (Mal-HM)
- The most expensive query, **Q21** (12 oblivious sort calls) completes in **42min**, under **malicious** security
- All other queries from prior work in under **10min**

Each computing party uses one `c7a.16xlarge` AWS instance (64 vCPUs, 128GB memory)

OBLIVIOUSLY SORTING HALF A BILLION ELEMENTS



~2 hours for 512M elements with malicious security

LAN setting with one `c7a.16xlarge` AWS instance (64 vCPUs, 128GB memory) per party

TUTORIAL OVERVIEW

- **Part I: Fundamentals of secure Multi-Party Computation (MPC)**
  - MPC use cases and basic concepts
  - Vectorization and message batching in MPC
  - Oblivious relational analytics with ORQ
- **Part II: Hands-on programming session with ORQ**
  - Deploying ORQ on CloudLab
  - ORQ example code walk-through
  - Implement and run your own secure data analysis pipeline using ORQ

We are hiring students and postdocs

https://sites.bu.edu/casp/

with generous support from

BOSTON UNIVERSITY

NSF

BOSCH

amazon