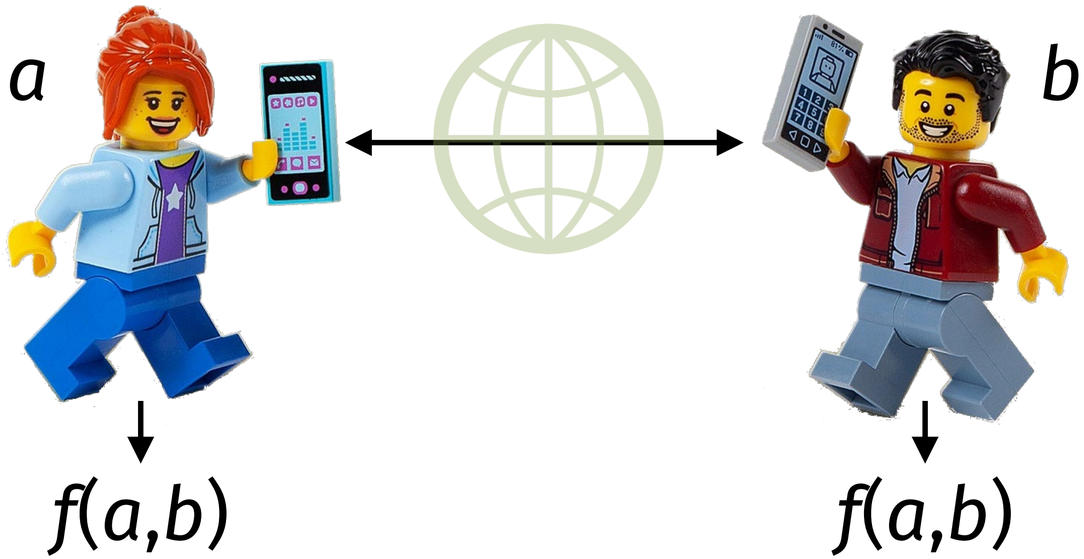
COURSE ANNOUNCEMENTS

- Homework 5 has been posted, due Friday 3/24 on Gradescope
- Homework 6 will be posted this week, and due Friday 3/31

Lecture 15: Protected Database Search

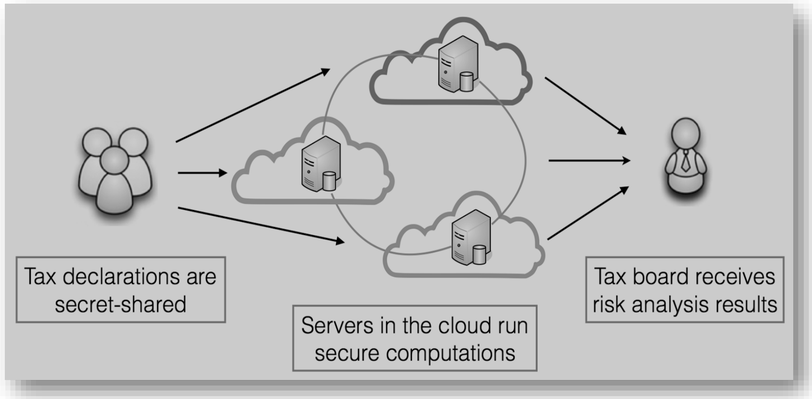
REVIEW: MPC

“Secure multi-party computation … enables *different participating entities* in possession of private sets of data to *link and aggregate their data sets* for the exclusive purpose of performing a finite number of pre-approved computations *without transferring or otherwise revealing any private data* to each other or anyone else.”

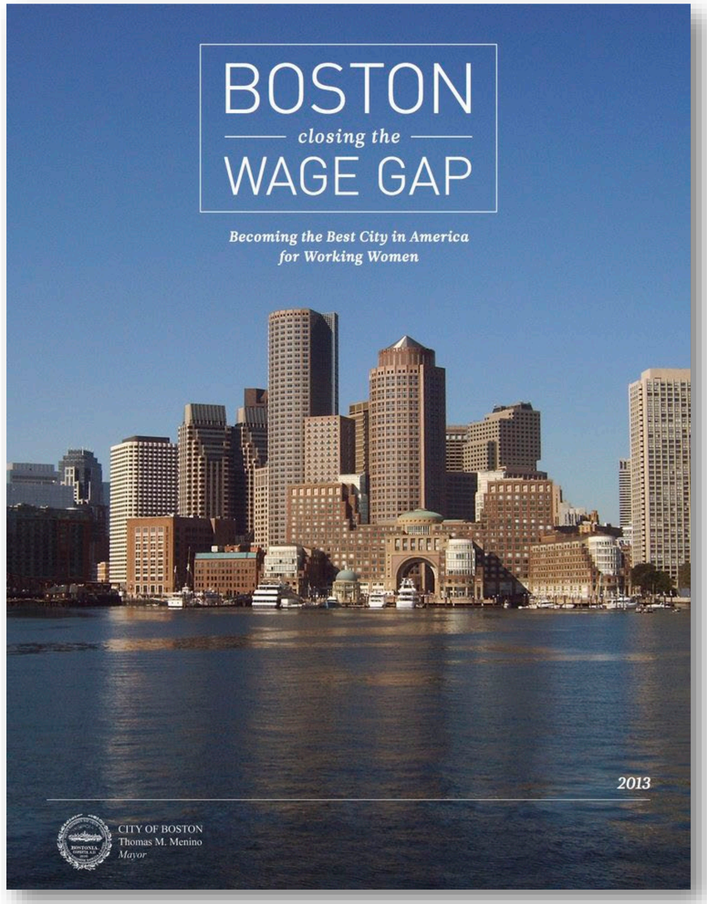


MPC DEPLOYMENTS

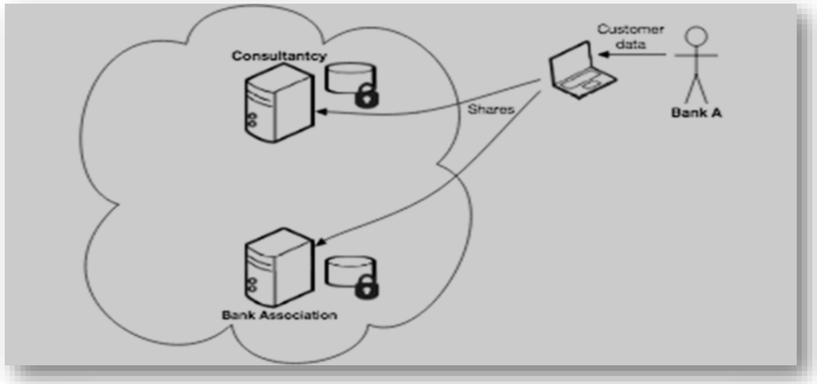
Cybernetica: VAT tax audits



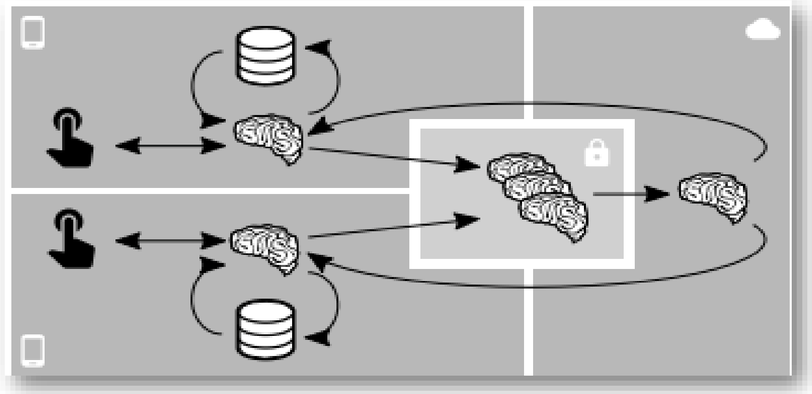
BU: Pay equity in Boston



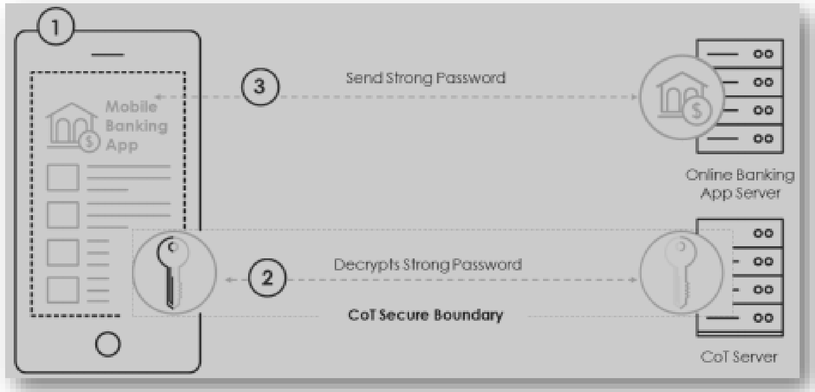
Partisia: Rate credit of farmers



Google: Federated machine learning



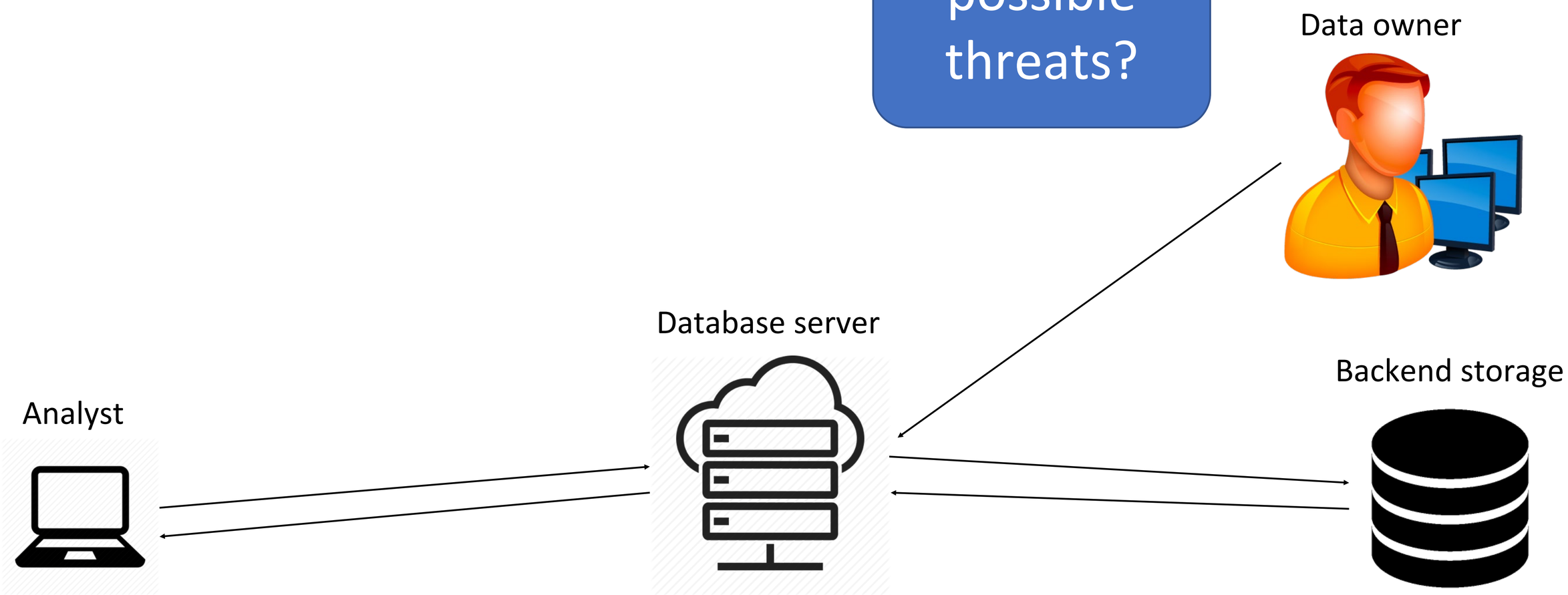
Unbound: Protect cryptographic keys



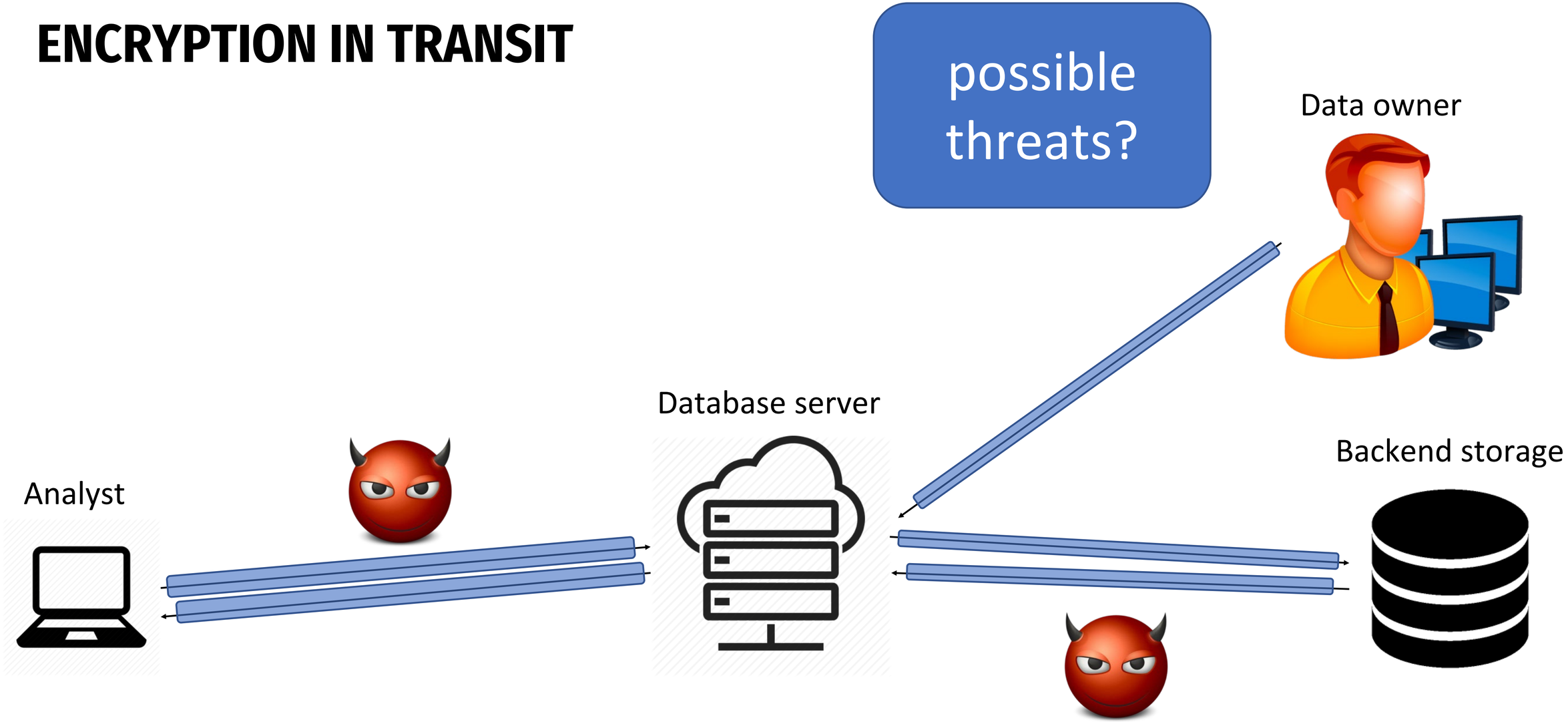
15.1 From Data to Databases

LET'S PROTECT A DATABASE

possible threats?

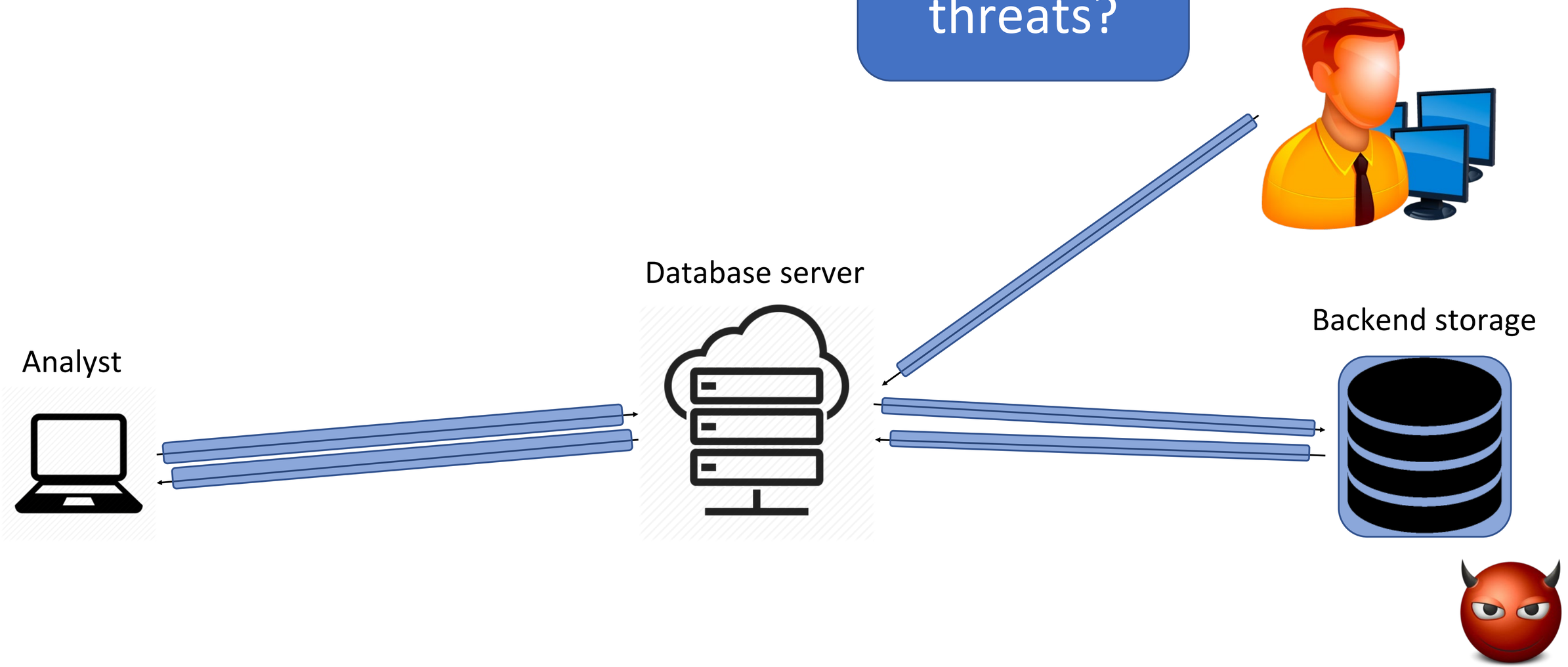


ENCRYPTION IN TRANSIT



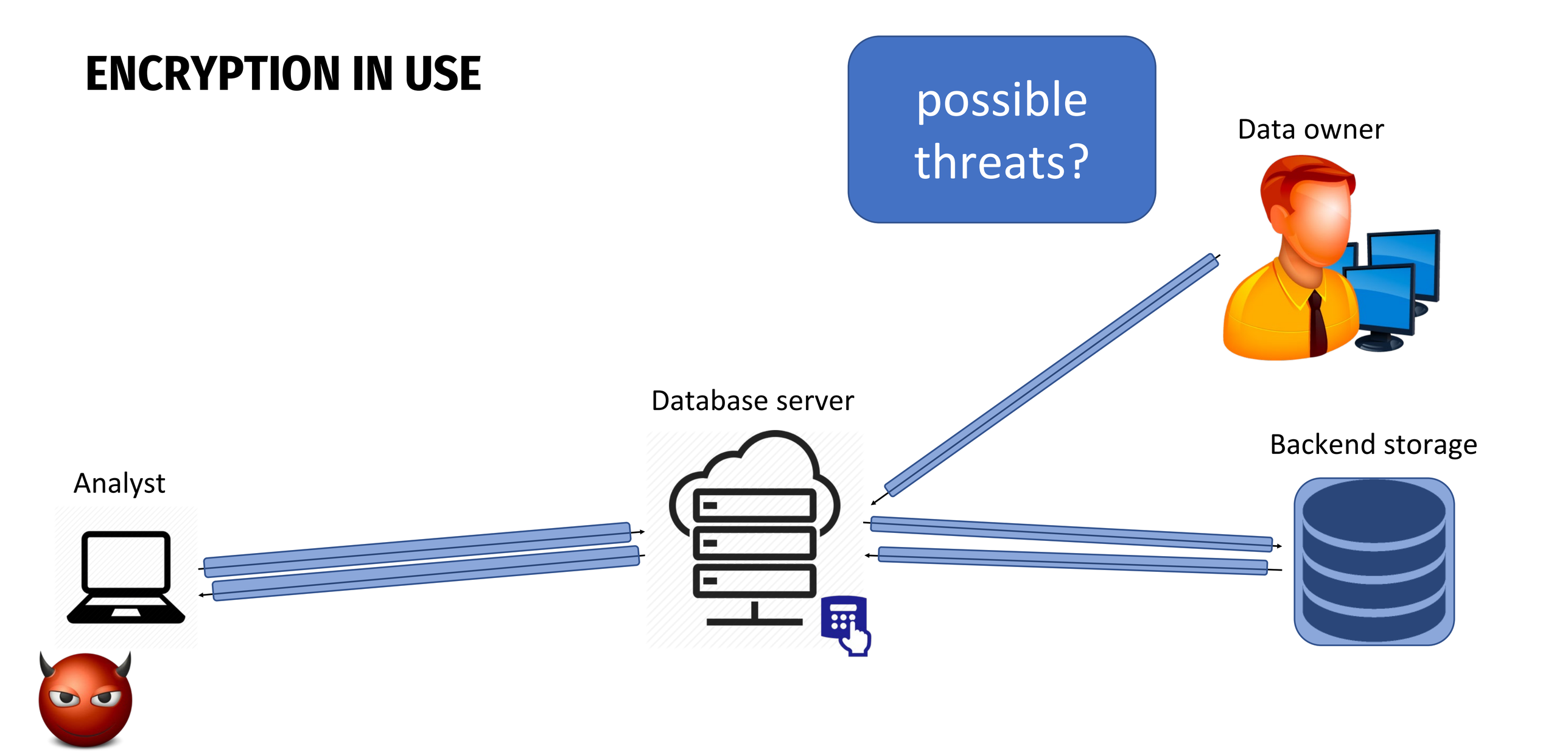
ENCRYPTION AT REST

possible threats?



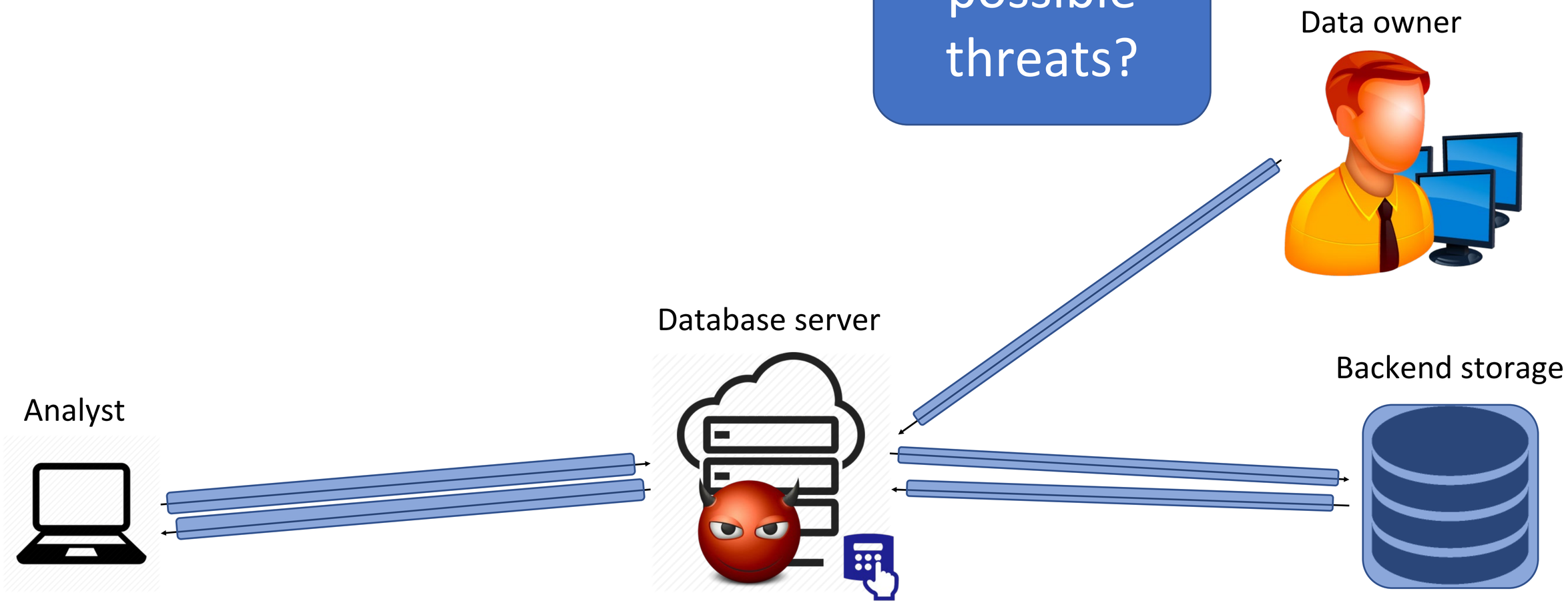
ENCRYPTION IN USE

possible threats?

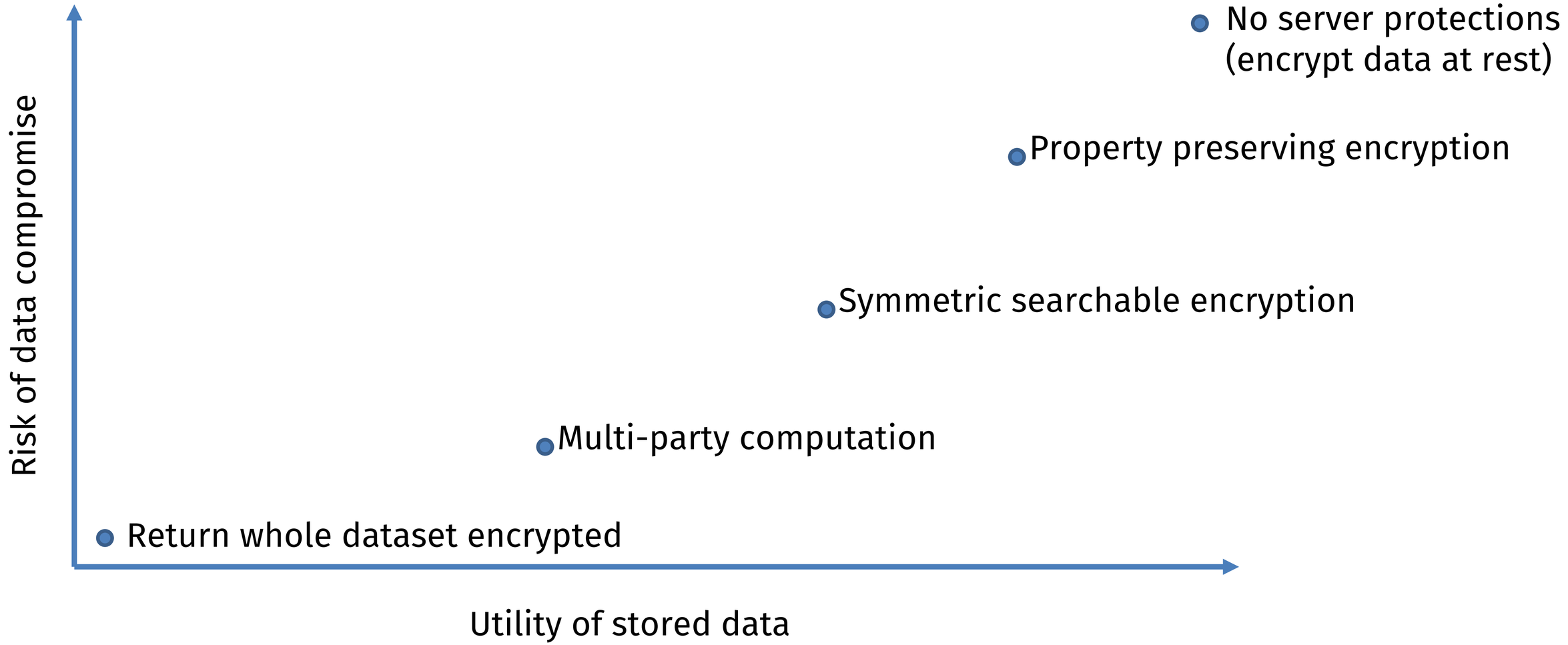


ENCRYPTION IN USE

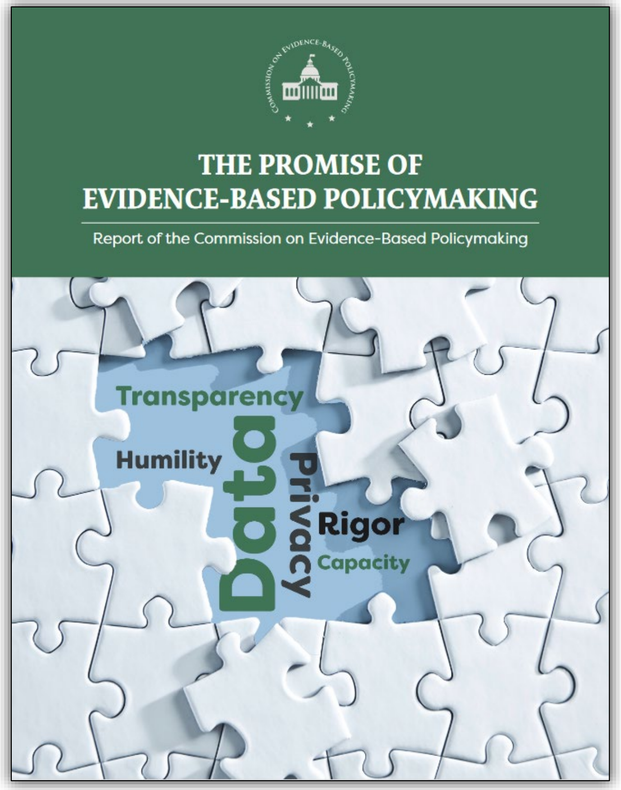
possible threats?

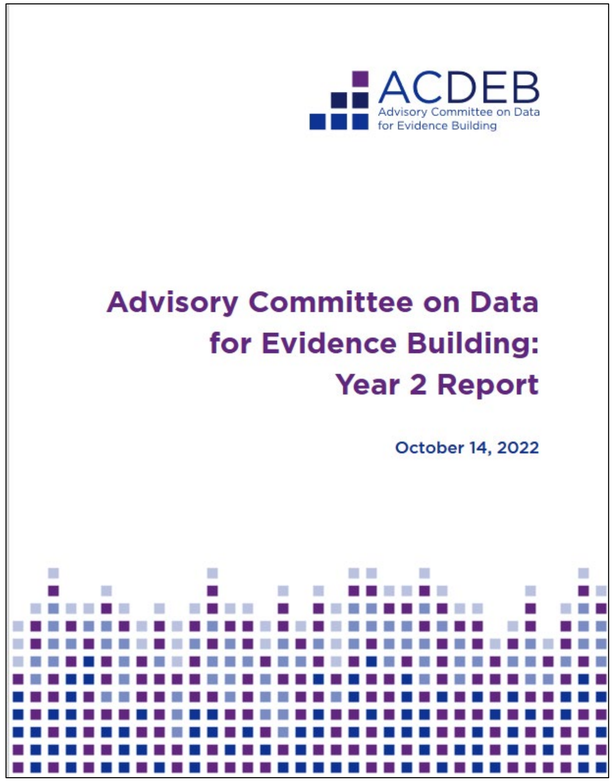


OBJECTIVE: CRYPTOGRAPHICALLY PROTECTED DATABASE SEARCH



INTEREST BY GOVERNMENT STATISTICAL AGENCIES





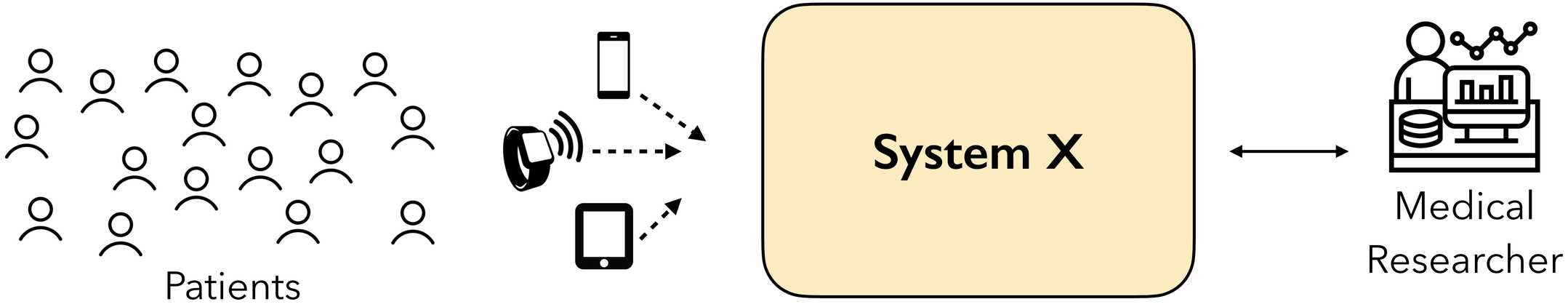
bea.gov/evidence

unstats.un.org/bigdata/task-teams/privacy

15.2 Designing MPC for Databases

MOTIVATING USE CASE: DIGITAL HEALTH ANALYTICS

(BU Medical & Hariri Institute for Computing)

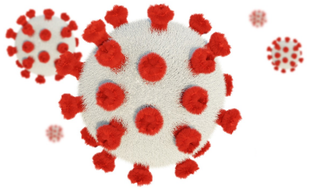


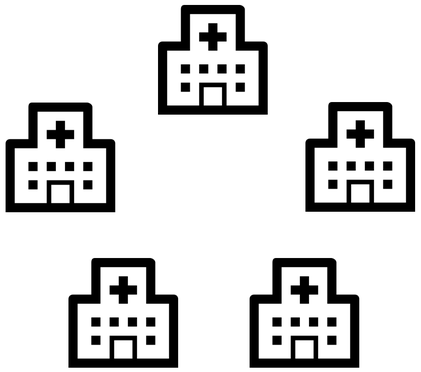
What would be the right System X for this use case?

https://www.bu.edu/hic/research/focused-research-programs/continuous-analysis-of-mobile-health-data-among-medically-vulnerable-populations/

SECURE COLLABORATIVE ANALYTICS

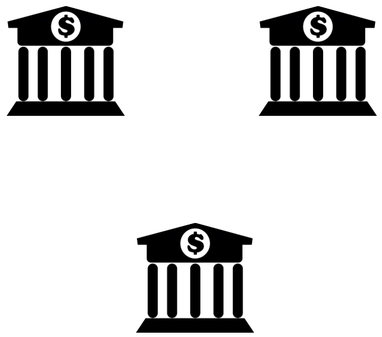
Medical Studies





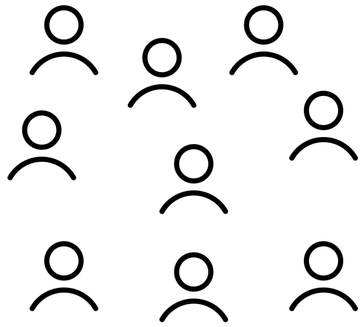
Healthcare providers

Market Analyses



Credit score agencies

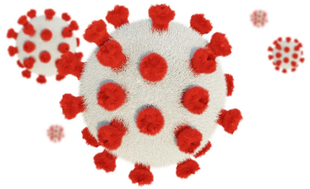
Privacy-preserving advertising



End-users

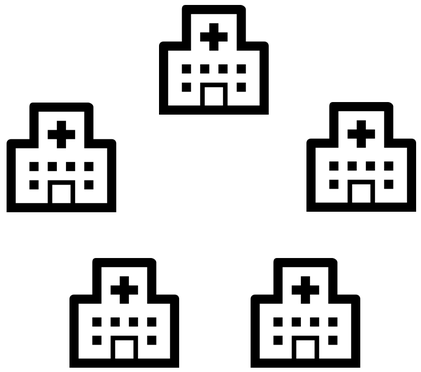
SECURE COLLABORATIVE ANALYTICS

Medical Studies

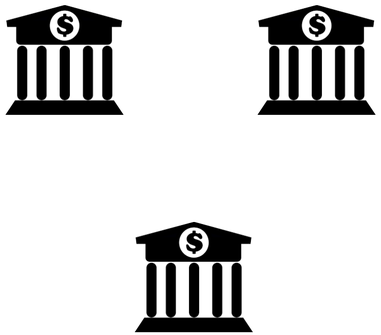


Market Analyses

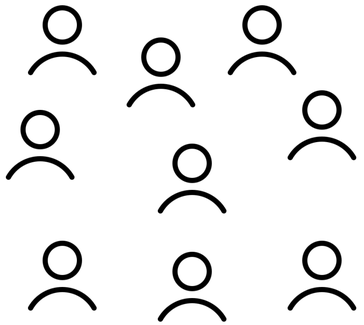
Privacy-preserving advertising



Healthcare providers



Credit score agencies

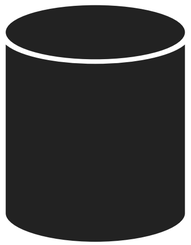


End-users

<u>Requirements</u>:

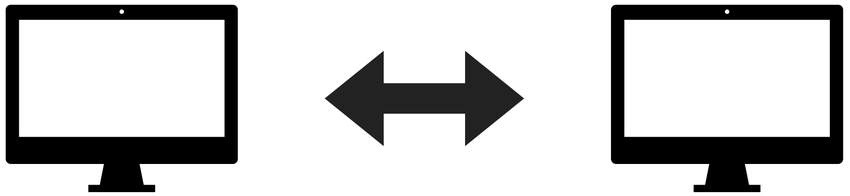
- ***No information leakage to untrusted entities***
- ***No reliance on trusted resources***
- ***Relational analytics***
- ***Practical performance***

GOAL: END-TO-END DATA PROTECTION



Data "at rest"

Advanced Encryption Standard (AES)



Data "in transit"

Transport Layers Security (TLS)

Data "in use"

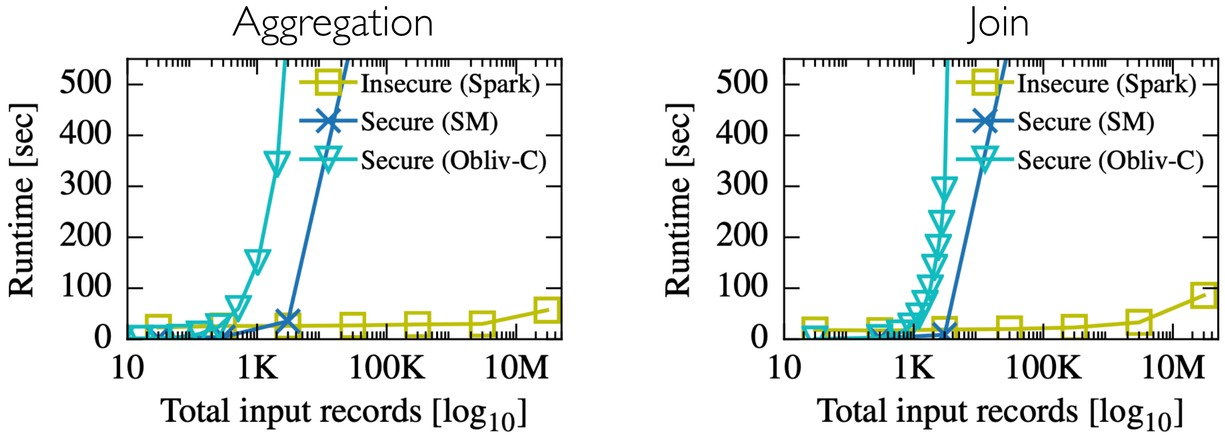
Why should we protect data in use?

CHALLENGE: HOW TO REDUCE THE MPC COST?

"Running the query entirely under MPC […] ***fails to scale beyond 3,000 total records****…"*

"Computing a function f on millions of client inputs […] could potentially take an ***astronomical amount of time*** *in a full MPC."*

"The primary source of the slowdown arises from their join operators that have ***hundreds of input tuples****…"*



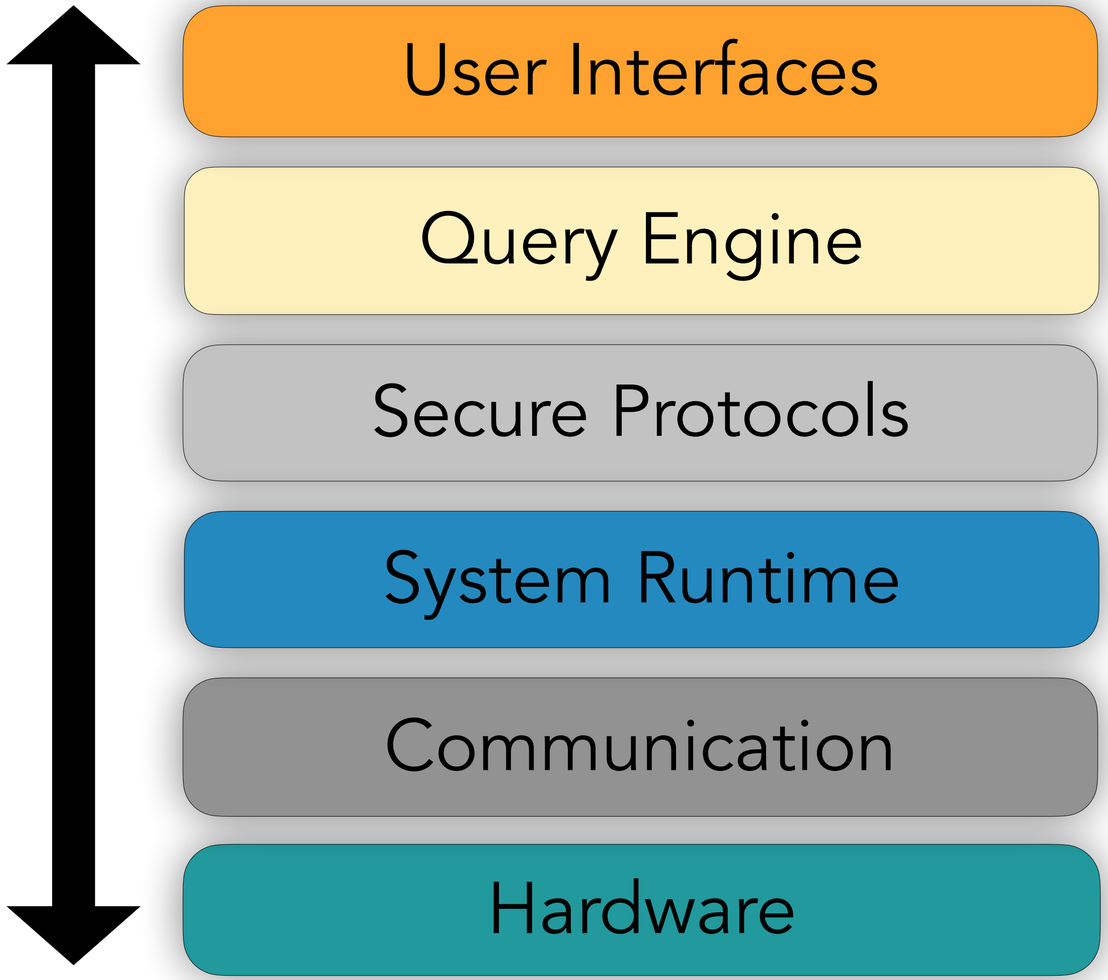
Plaintext	Secure	Slowdown
158	253,894	1,609X
165	159,145	967X
193	8,195,317	43,337X

[1] N. Volgushev, M. Schwarzkopf, B. Getchell, M. Varia, A. Lapets, and A. Bestavros. *Conclave: secure multi-party computation on big data*. EuroSys, 2019.

[2] J. Bater, G. Elliott, C. Eggen, S. Goel, A. N. Kho, and J. Rogers. *SMCQL: secure querying for federated databases*. PVLDB, 10(6):673–684, 2017.

[3] H. Corrigan-Gibbs and D. Boneh. *Prio: Private, Robust, and Scalable Computation of Aggregate Statistics*, NSDI, 2017.

TO MAKE MPC PRACTICAL WE NEED TO RETHINK THE SYSTEM STACK



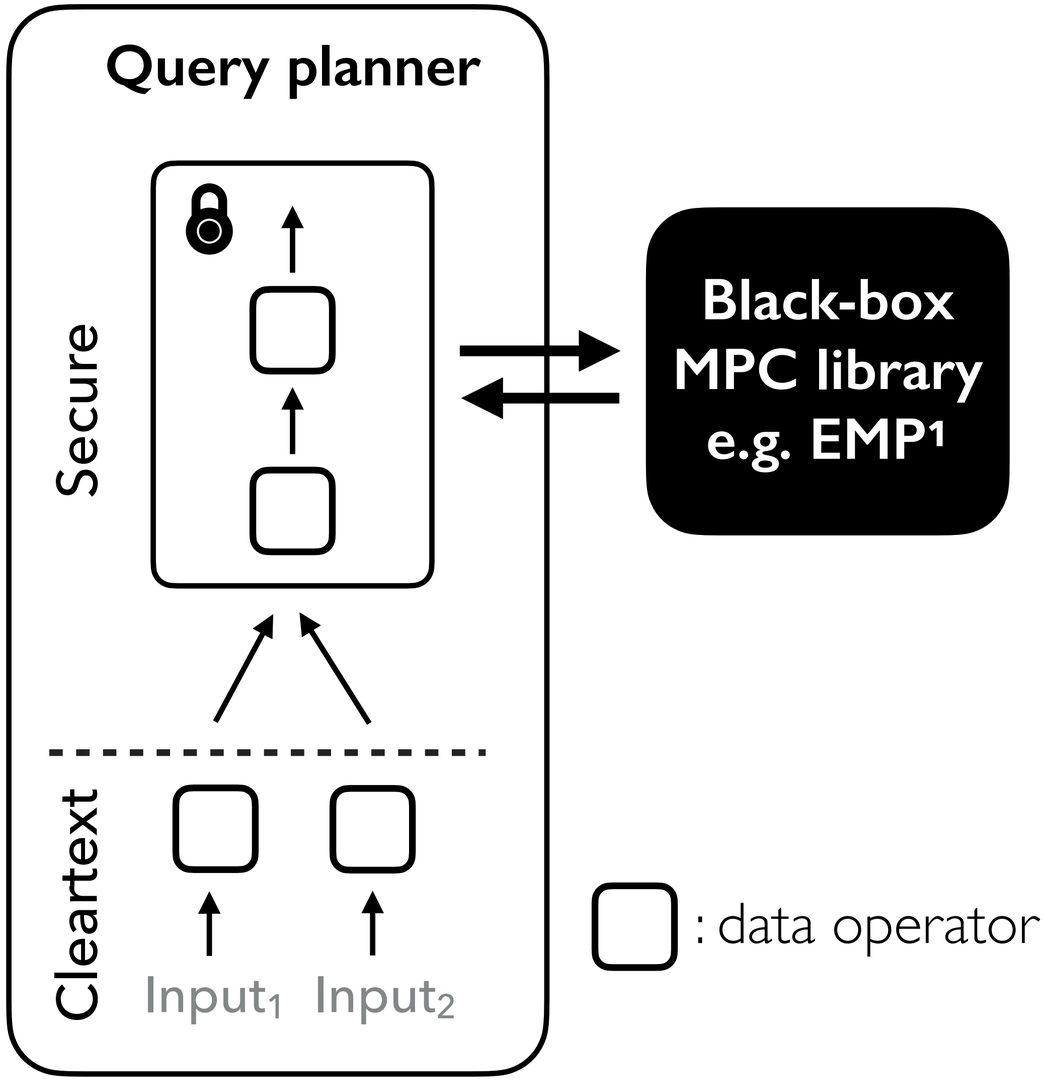
The Secrecy Framework

No information leakage

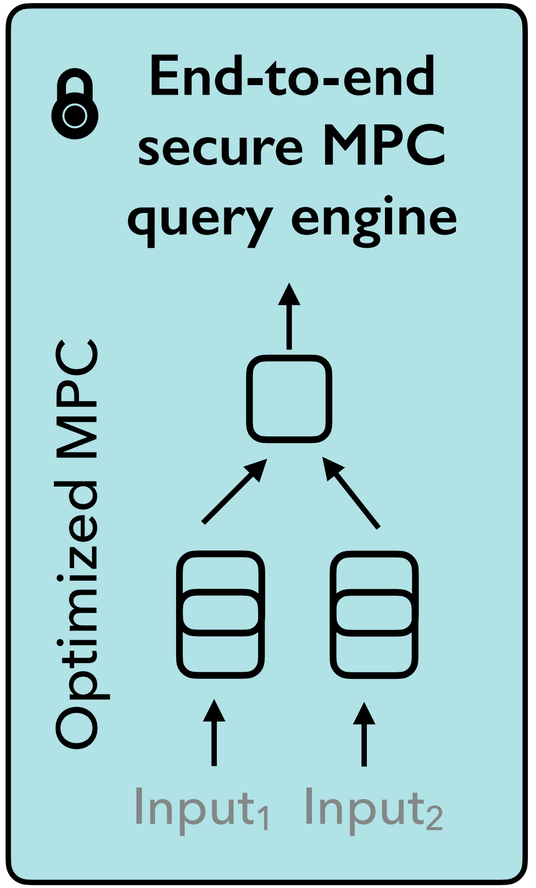
No trusted resources

Complex data analytics

OPENING THE MPC BLACK BOXES



: data operator



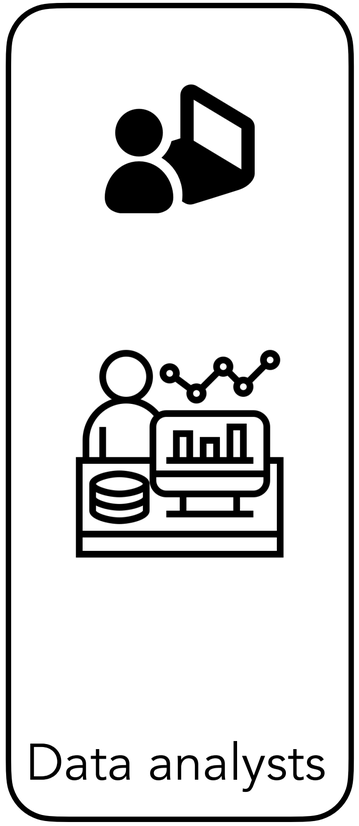
Secrecy

Supported optimizations:

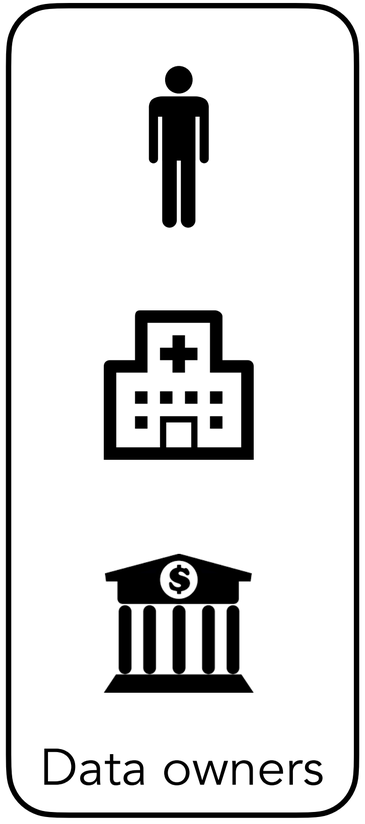
- Logical (e.g. operator reordering)
- Physical (e.g. message batching, operator fusion)
- Protocol-specific (e.g. dual sharing)

[1] X. Wang, A. J. Malozemoff, and J. Katz. *EMP-toolkit: Efficient MultiParty computation toolkit*, 2016. https://github.com/emp-toolkit

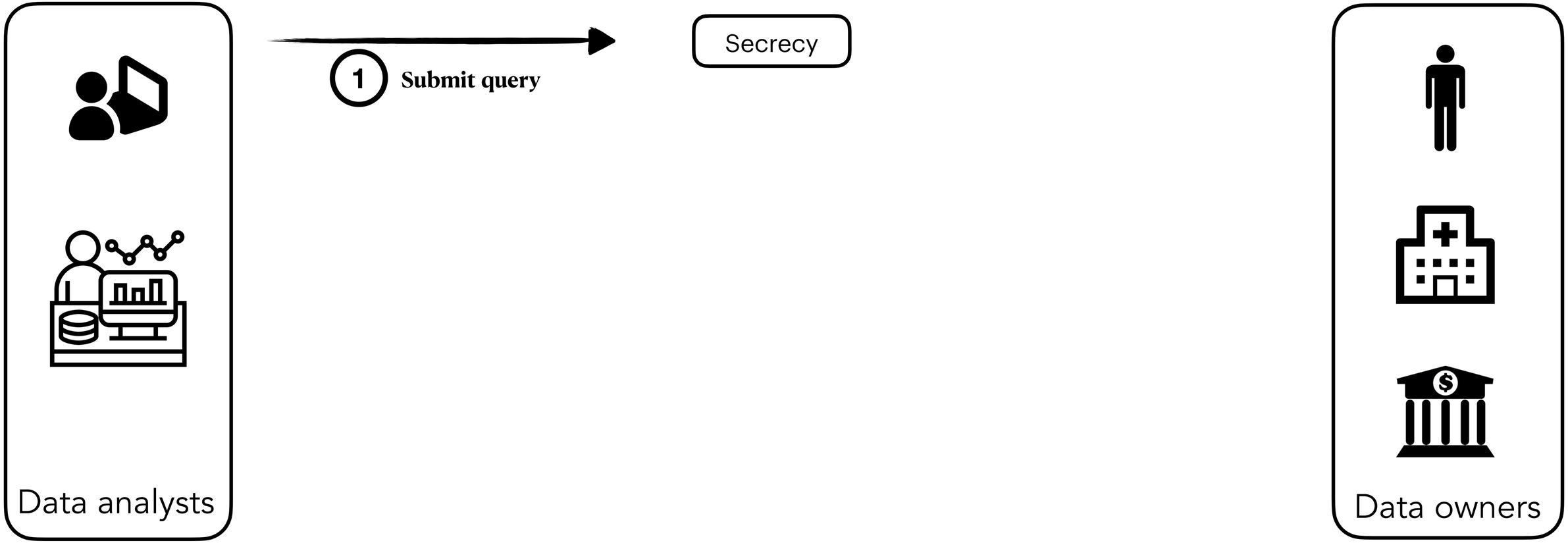
SECRECY AS A SERVICE



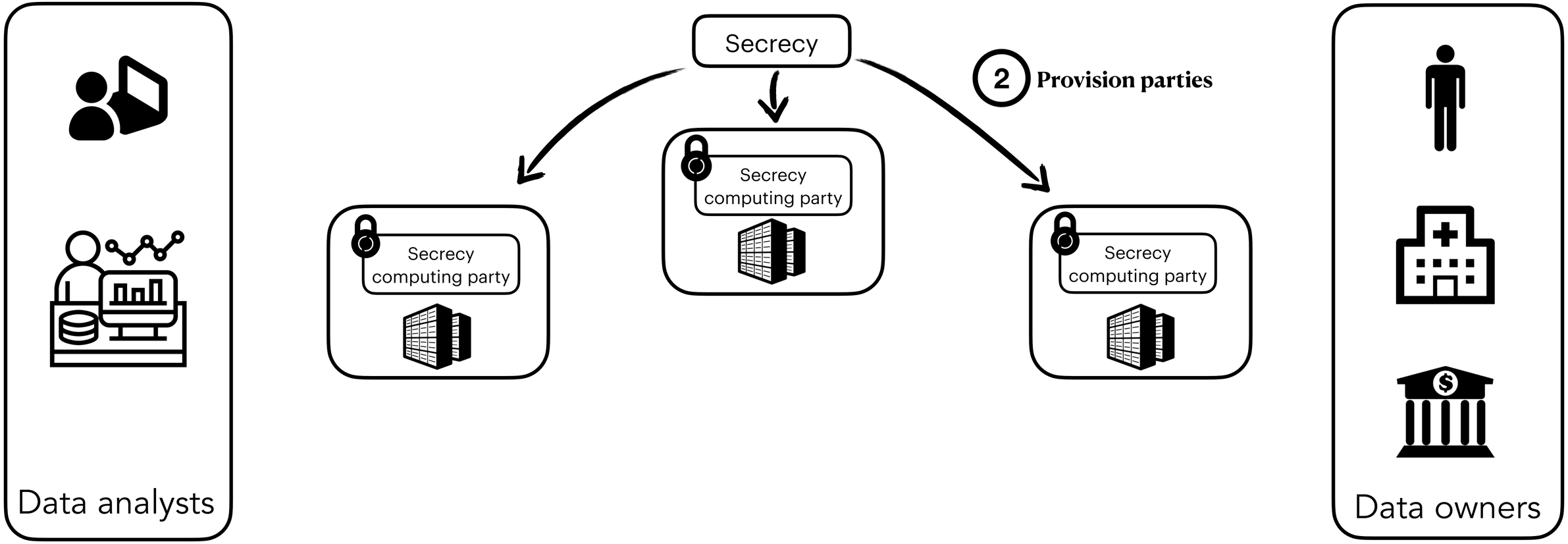
Secrecy



SECRECY AS A SERVICE

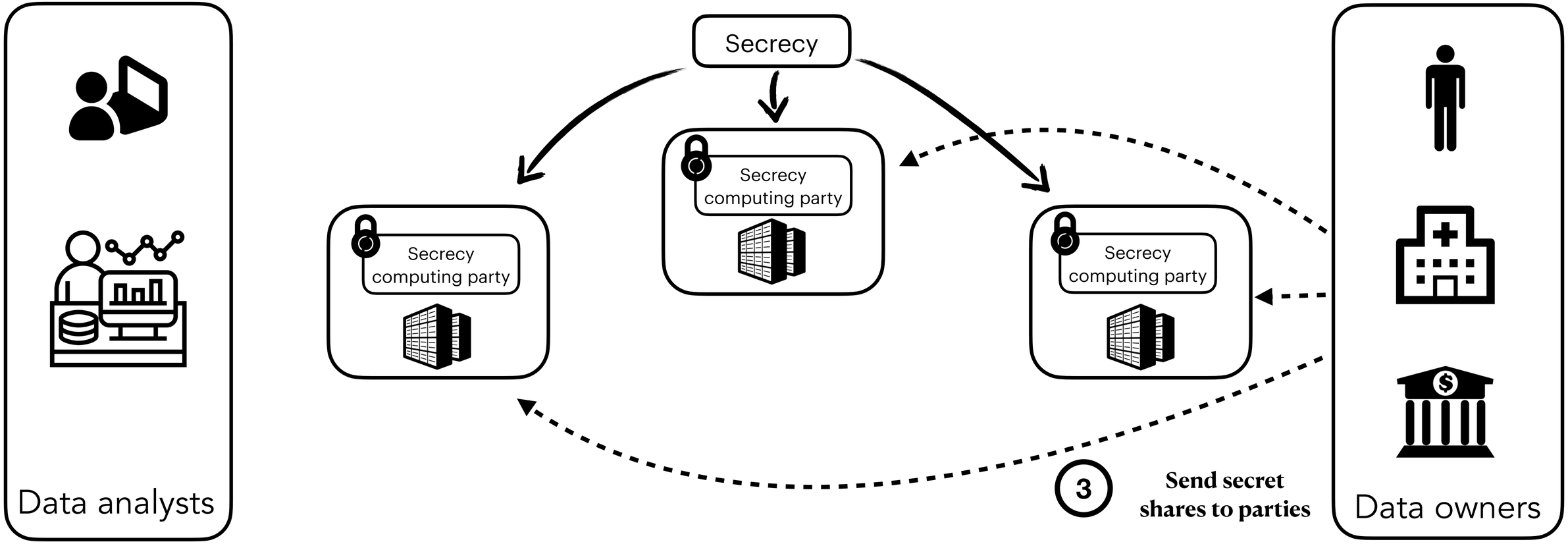


SECRECY AS A SERVICE

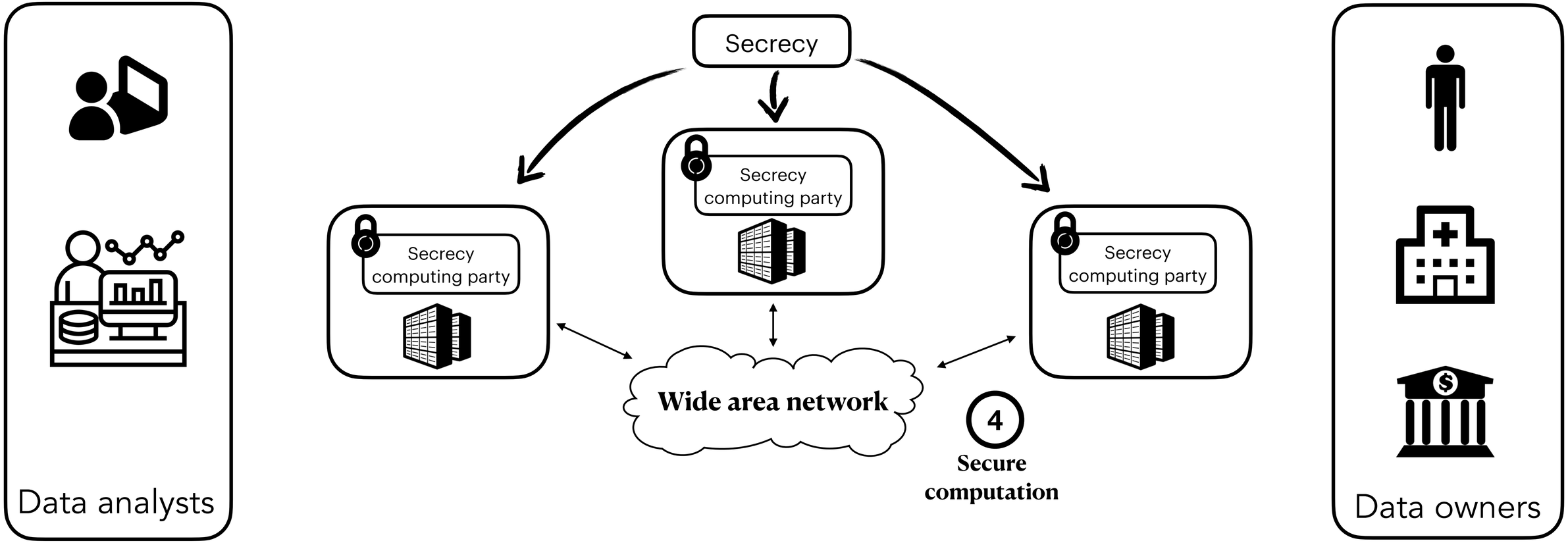


Supported Cloud providers:

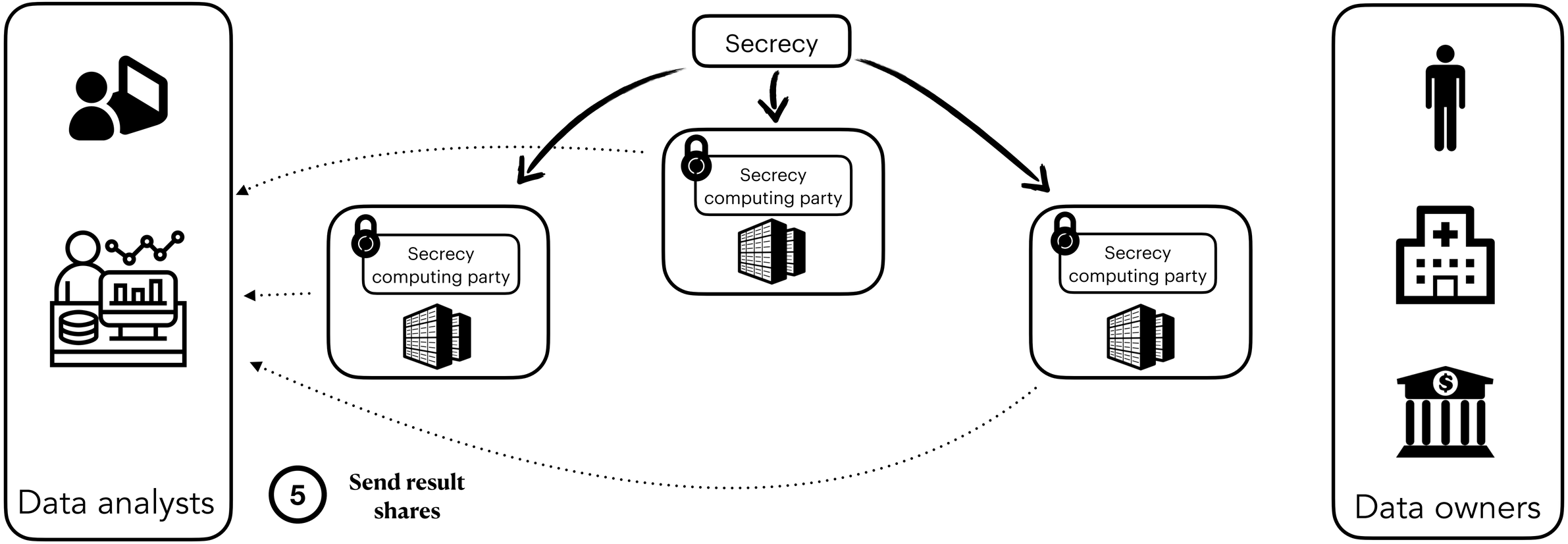
SECRECY AS A SERVICE



SECRECY AS A SERVICE



SECRECY AS A SERVICE



THREAT MODEL AND GUARANTEES

Semi-honest model

- Computing parties do not deviate from the protocol ("honest but curious")
- Adversary can monitor the network and can also compromise one computing party (but cannot alter its execution)

Security guarantees

- Untrusted parties do not learn anything about:
  - The original data, intermediate or output result sizes
  - The data access patterns during query execution

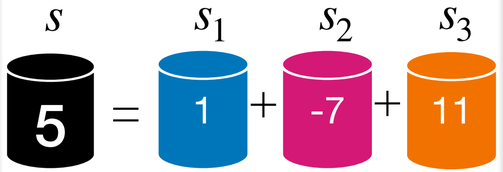
* *Adding support for maliciously secure primitives in progress*

EXAMPLE: SECURE ADDITION IN SECRECY

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

Party 1

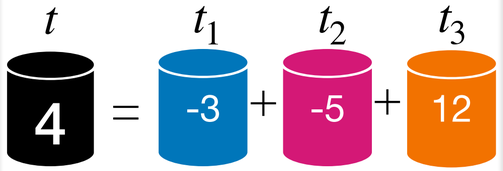
Data Owner A



Party 3

Party 2

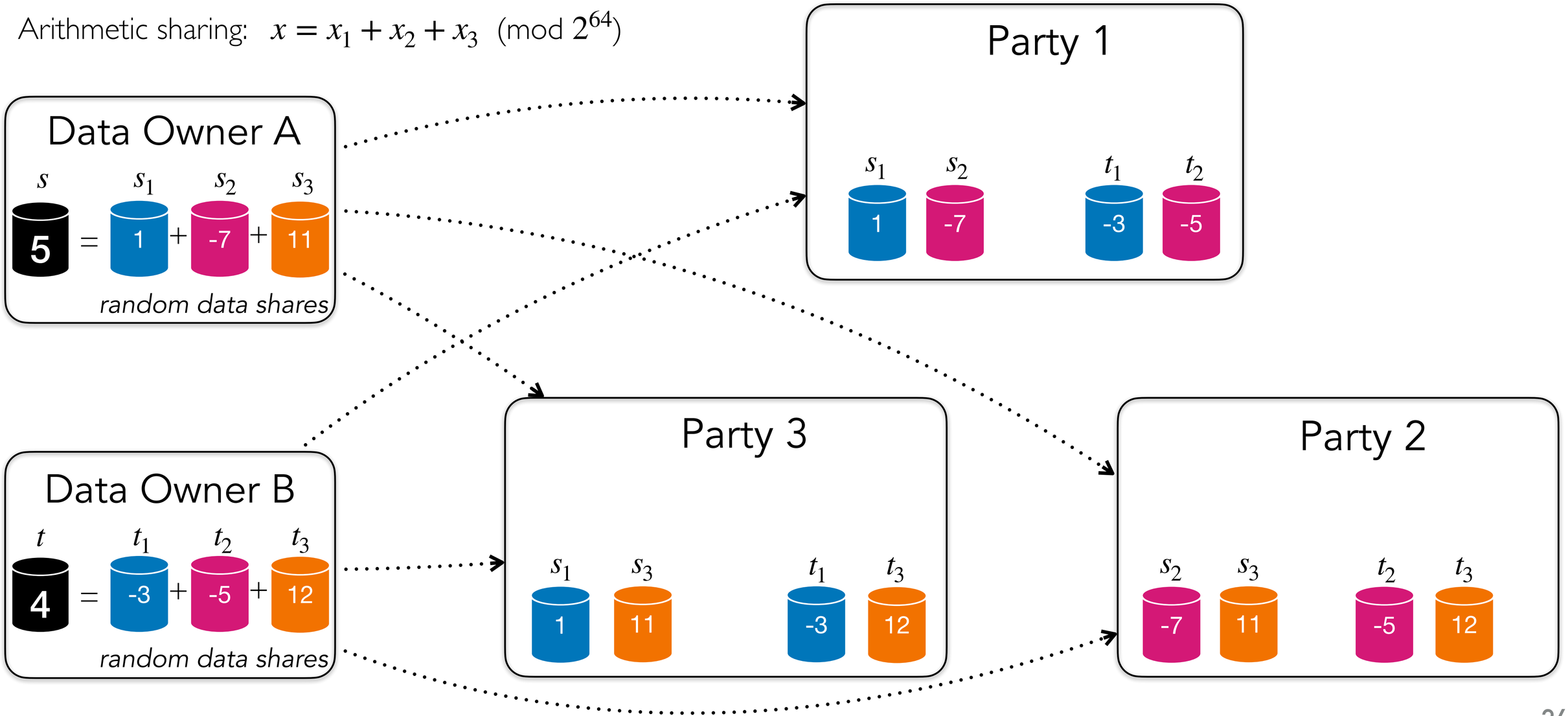
Data Owner B



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 IN SECRECY

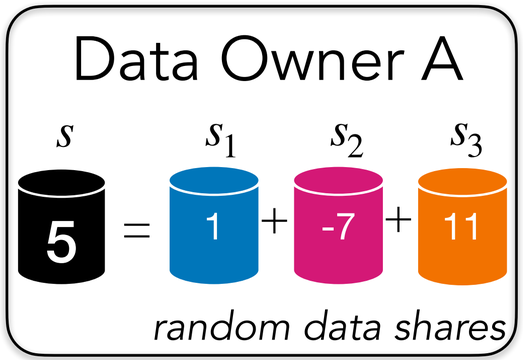
Arithmetic sharing: $x = x_1 + x_2 + x_3 \pmod{2^{64}}$

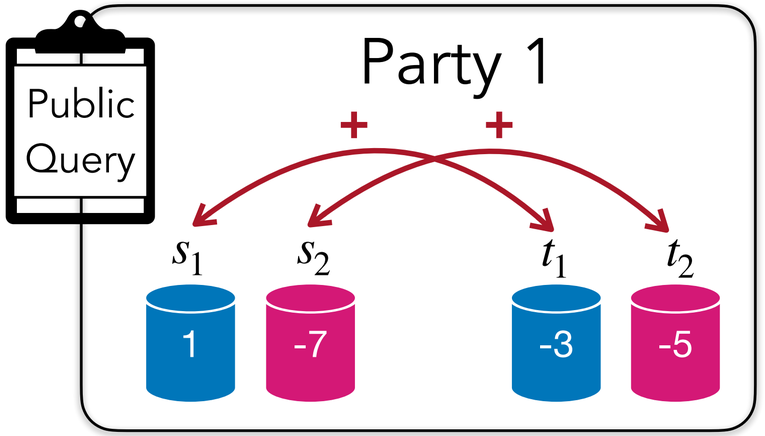


T. Araki, J. Furukawa, Y. Lindell, A. Nof, and K. Ohara. *High-Throughput Semi-Honest Secure Three-Party Computation with an Honest Majority*. CCS, 2016.

EXAMPLE: SECURE ADDITION IN SECRECY

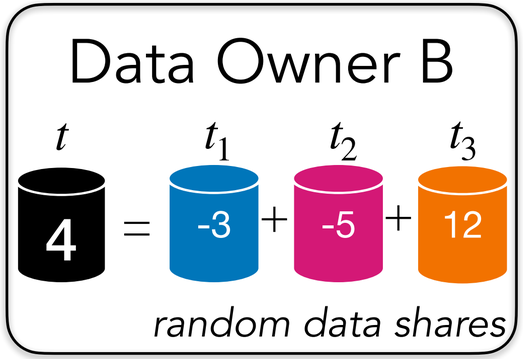
Arithmetic sharing: $x = x_1 + x_2 + x_3 \pmod{2^{64}}$

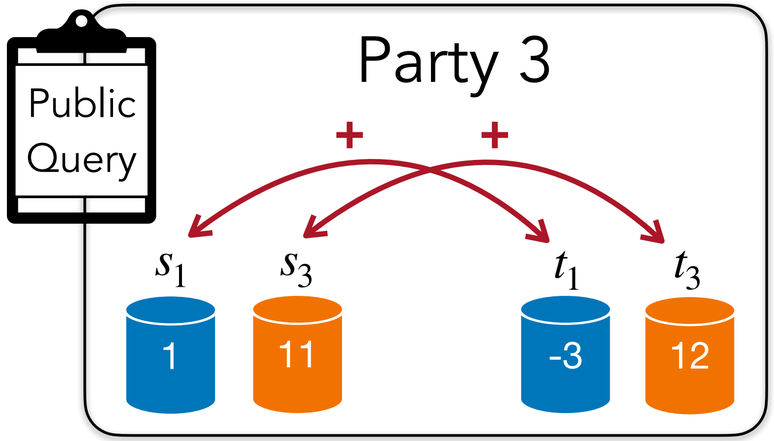


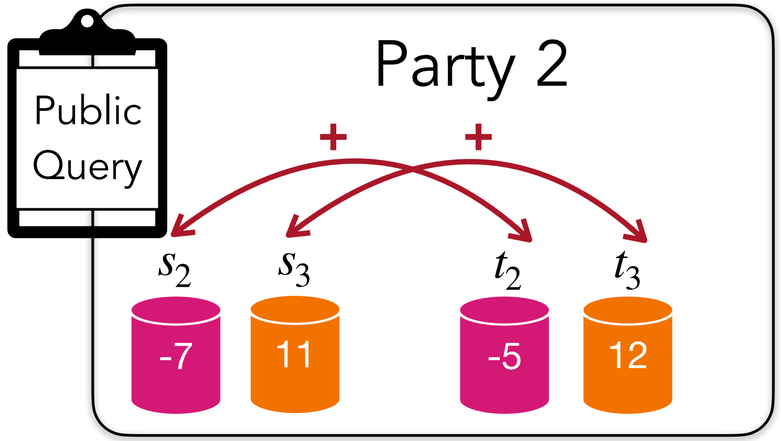


Query: $s + t$

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



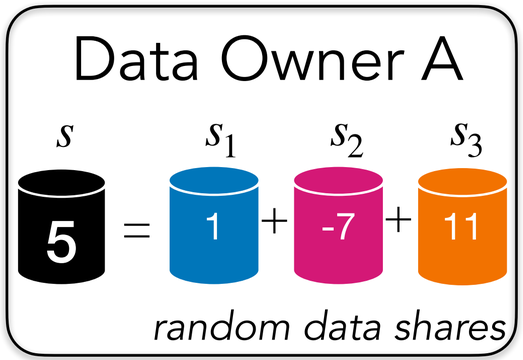


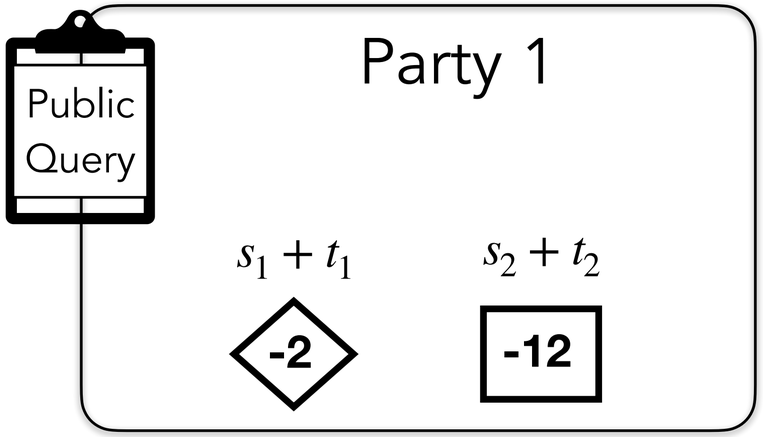


T. Araki, J. Furukawa, Y. Lindell, A. Nof, and K. Ohara. *High-Throughput Semi-Honest Secure Three-Party Computation with an Honest Majority*. CCS, 2016.

EXAMPLE: SECURE ADDITION IN SECRECY

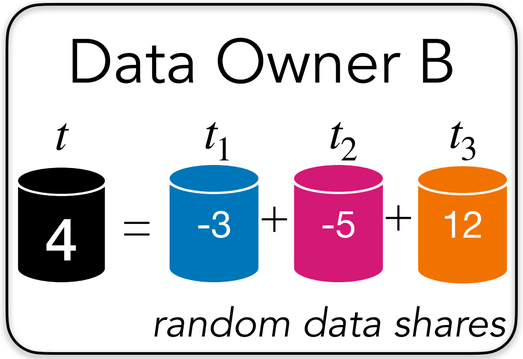
Arithmetic sharing: $x = x_1 + x_2 + x_3 \pmod{2^{64}}$

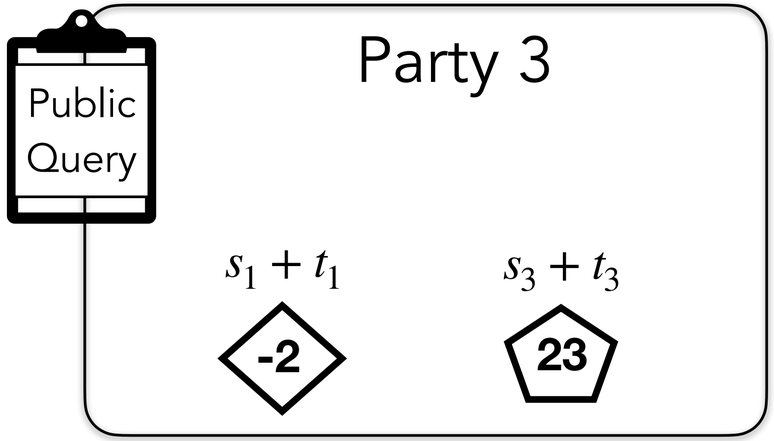


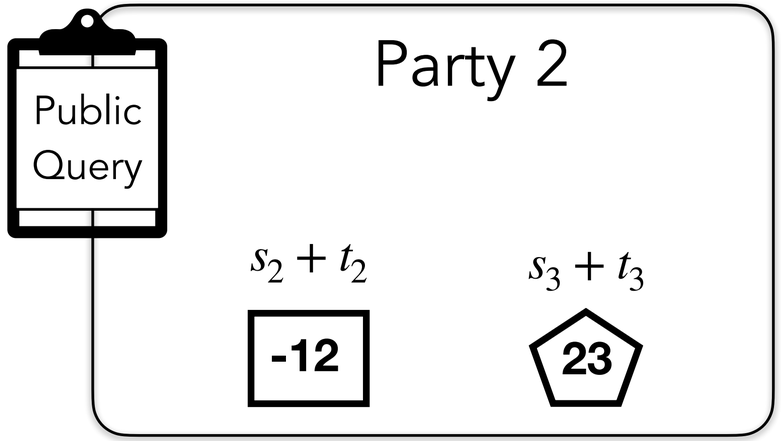


Query: $s + t$

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



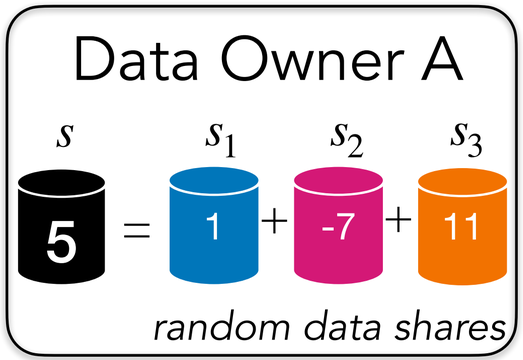


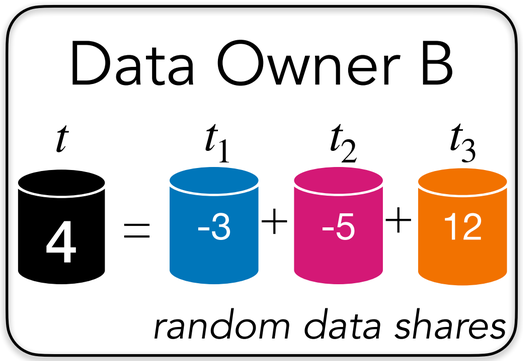


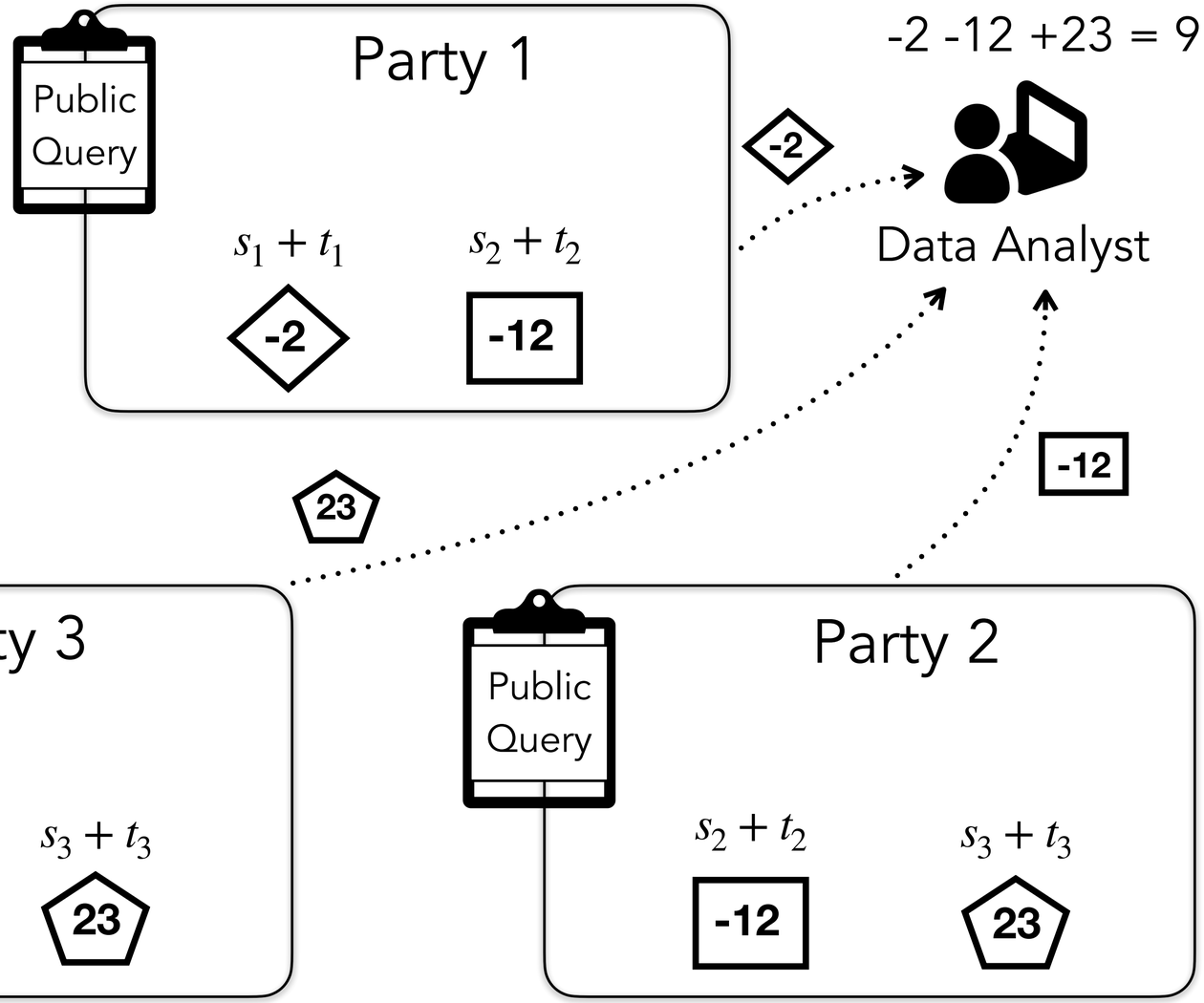
T. Araki, J. Furukawa, Y. Lindell, A. Nof, and K. Ohara. *High-Throughput Semi-Honest Secure Three-Party Computation with an Honest Majority*. CCS, 2016.

EXAMPLE: SECURE ADDITION IN SECRECY

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



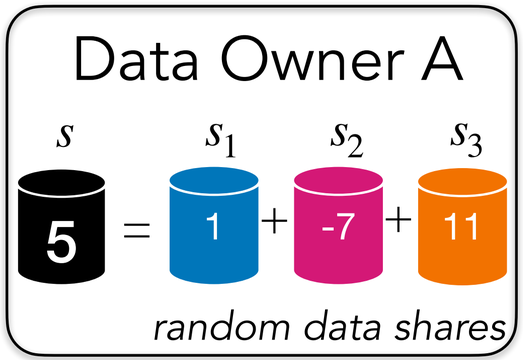


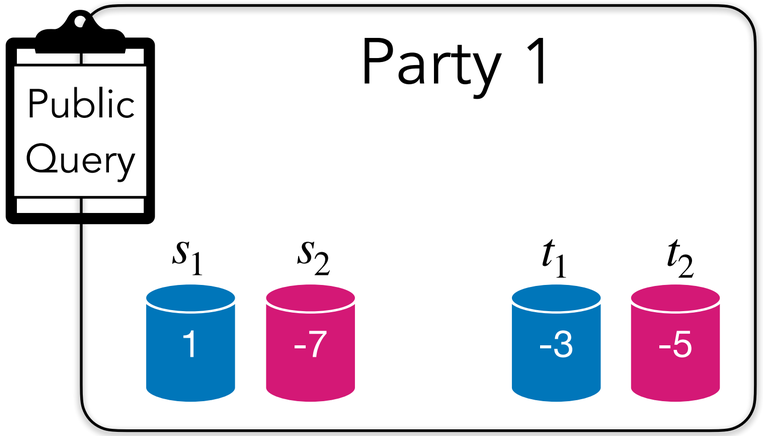


T. Araki, J. Furukawa, Y. Lindell, A. Nof, and K. Ohara. *High-Throughput Semi-Honest Secure Three-Party Computation with an Honest Majority*. CCS, 2016.

EXAMPLE: SECURE MULTIPLICATION IN SECRECY

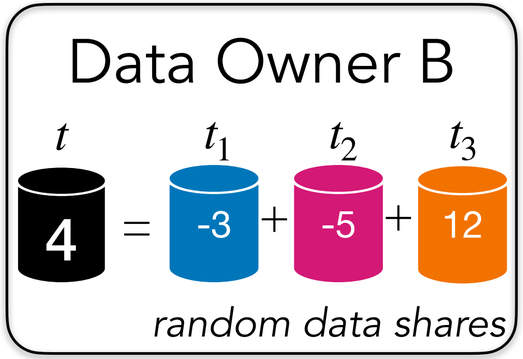
Arithmetic sharing: $x = x_1 + x_2 + x_3 \pmod{2^{64}}$

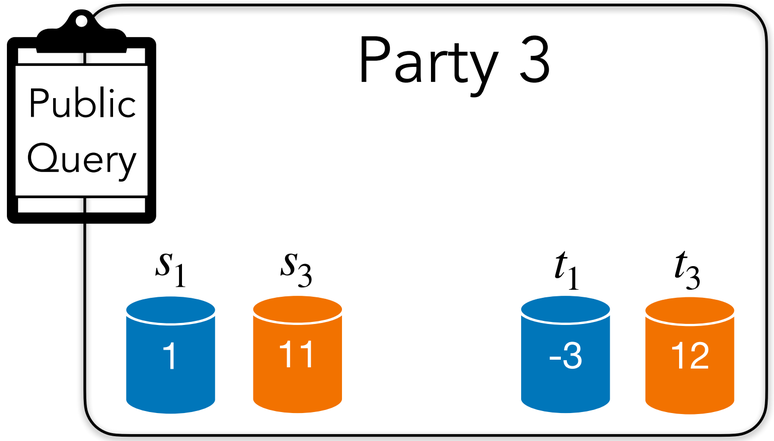


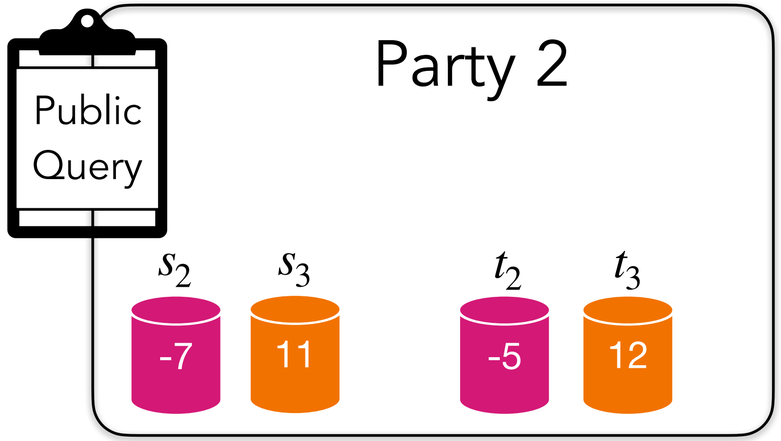


Query: $s \times t$

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



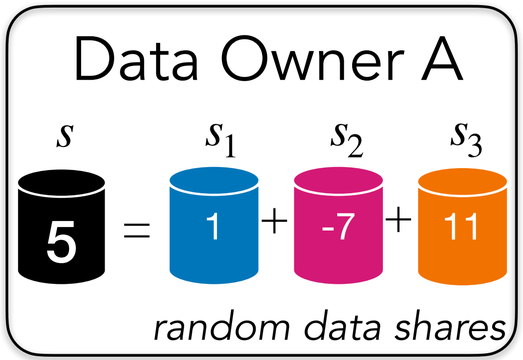


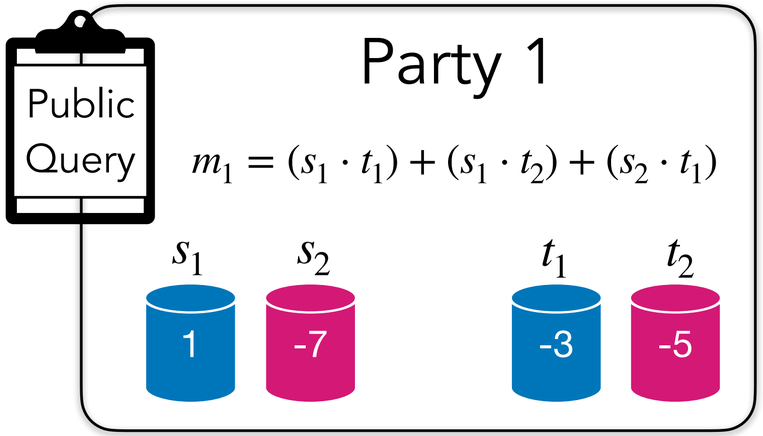


T. Araki, J. Furukawa, Y. Lindell, A. Nof, and K. Ohara. *High-Throughput Semi-Honest Secure Three-Party Computation with an Honest Majority*. CCS, 2016.

EXAMPLE: SECURE MULTIPLICATION IN SECRECY

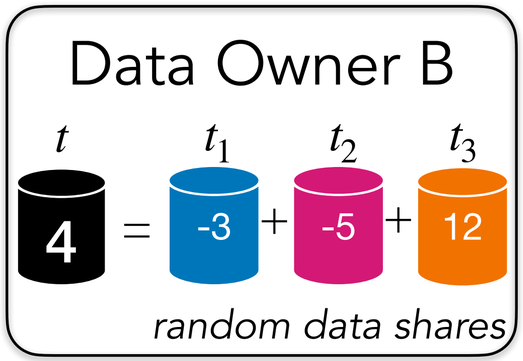
Arithmetic sharing: $x = x_1 + x_2 + x_3 \pmod{2^{64}}$

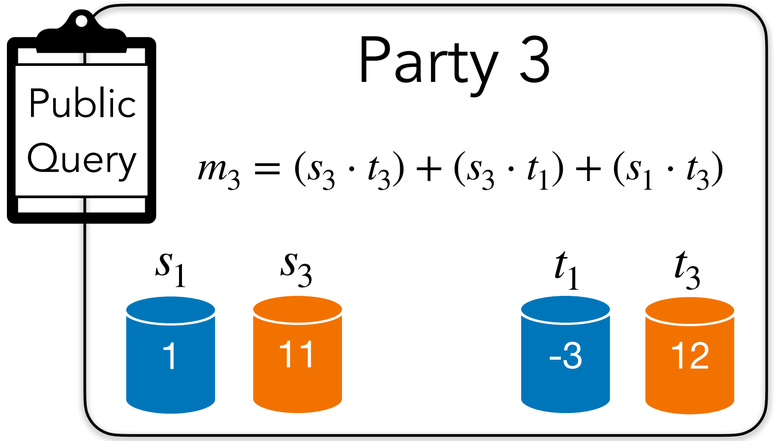


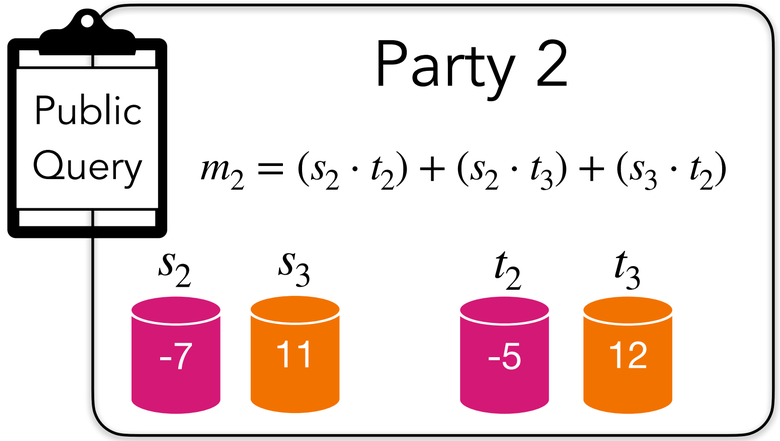


Query: $s \times t$

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



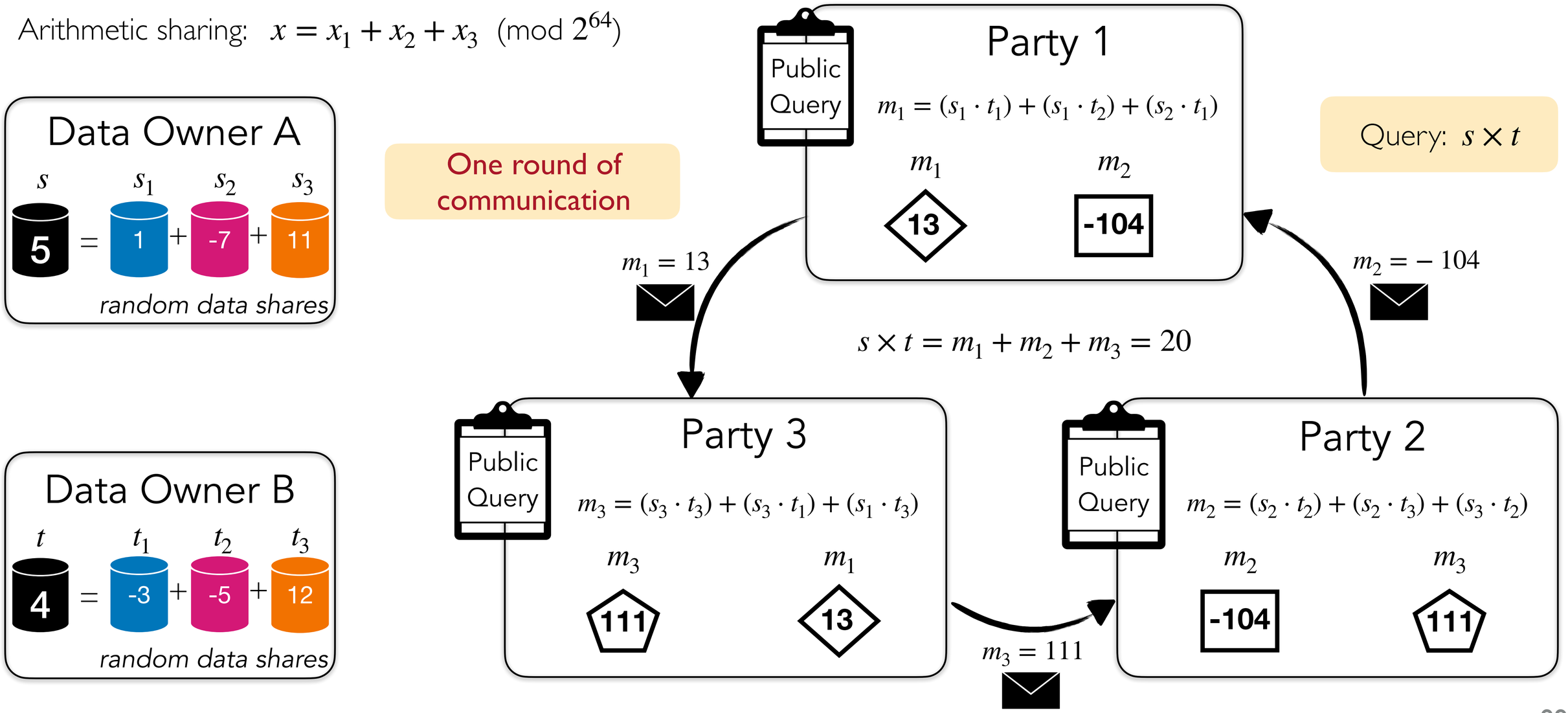




T. Araki, J. Furukawa, Y. Lindell, A. Nof, and K. Ohara. *High-Throughput Semi-Honest Secure Three-Party Computation with an Honest Majority*. CCS, 2016.

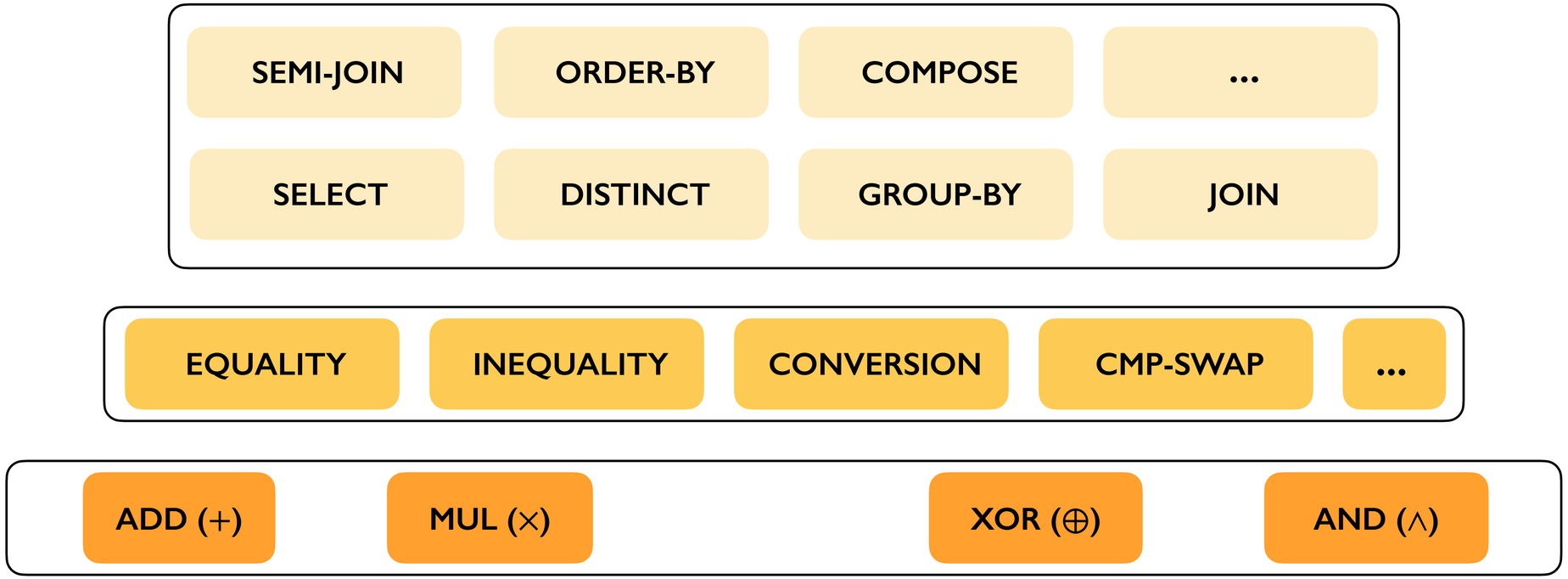
EXAMPLE: SECURE MULTIPLICATION IN SECRECY

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 ADD & MUL TO RELATIONAL ANALYTICS



Arithmetic sharing: $s = s_1 + s_2 + s_3 \pmod{2^{64}}$

Boolean sharing: $s = s_1 \oplus s_2 \oplus s_3$

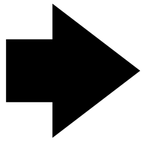
OBLIVIOUS COMPUTATION

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*

If $(a > b)$ {…}



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

"Number a is greater than b if the first bit they differ as we go from left to right is 1 *for a and* 0 *for b"*

Cleartext

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: $a : a_2 a_1 a_0$ $b : b_2 b_1 b_0$

If $(a > b)$ $\{\ldots\}$ ➡

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

Cleartext

Oblivious

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: $a : a_2a_1a_0$ $b : b_2b_1b_0$

Cleartext

$$\text{If } (a > b)\ \{\ldots\}$$

Oblivious

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

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

OBLIVIOUS COMPUTATION

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

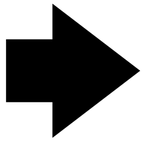
- Data access patterns do not depend on the actual shares
- No conditionals (if-then-else)
- No data reduction

* *Both a and b are secret-shared*

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

"Else, a is greater than b when the second most significant bits are not the same and a_1 is set"

If $(a > b)$ $\{\ldots\}$



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

Cleartext

Oblivious

OBLIVIOUS COMPUTATION

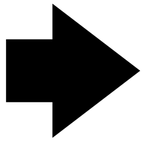
To prevent information leakage, the computing parties perform an identical computation that is data-independent

- 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: $a : a_2 a_1 a_0$ $b : b_2 b_1 b_0$

If $(a > b)$ $\{\ldots\}$



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

"Else, a is greater than b when a_0 is set and b_0 is not set"

Cleartext

Oblivious

OBLIVIOUS COMPUTATION

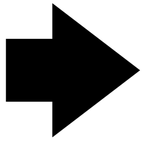
To prevent information leakage, the computing parties perform an identical computation that is data-independent

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

R

Employee	Salary
Kim	2000
Jane	1500
Alex	4500

$\sigma(Salary > 3000)$



R'

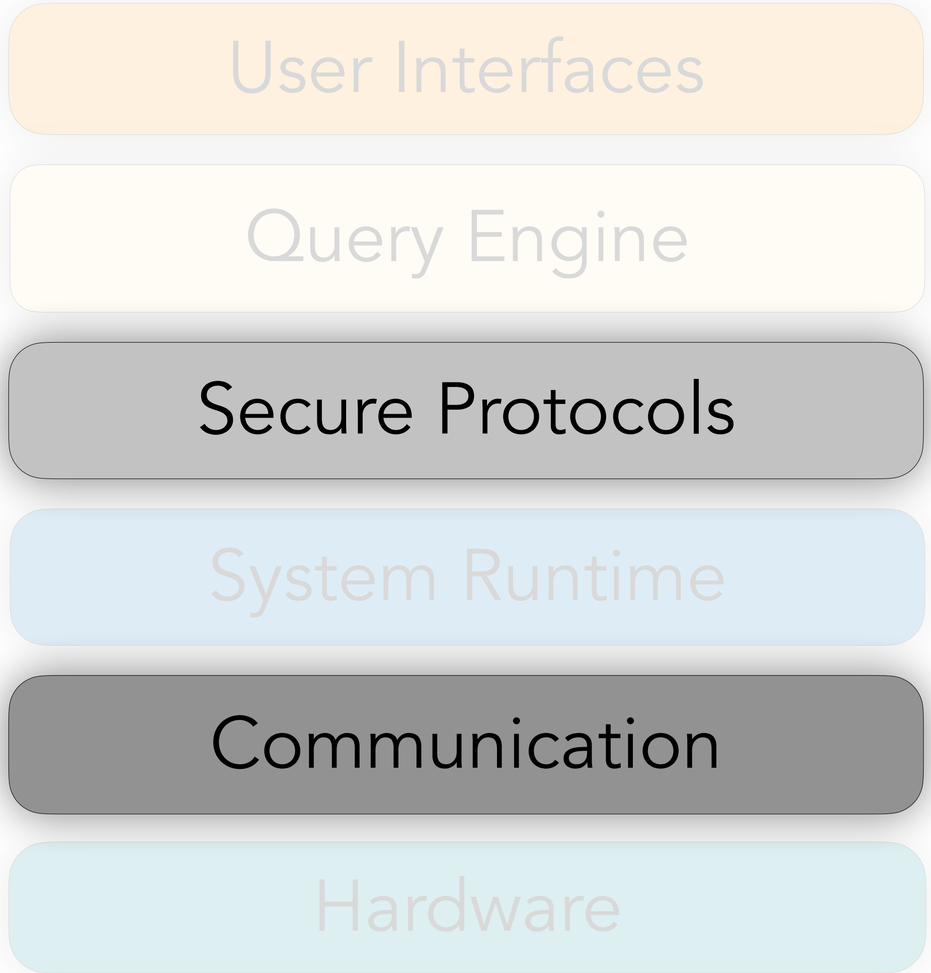
Employee	Salary	ϕ
Kim	2000	0
Jane	1500	0
Alex	4500	1

* *All attributes are secret-shared*

SECRECY's CORE CONTRIBUTIONS

1. ***Redesigned MPC primitives that work directly on relations instead of individual records***
   - Amortize network I/O
   - Make secret-sharing competitive in high-latency (WAN) environments

EXAMPLE: MESSAGE BATCHING



EXAMPLE: MESSAGE BATCHING

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

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

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

R

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

B			
b			
b'			
$\cdots$			

S

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

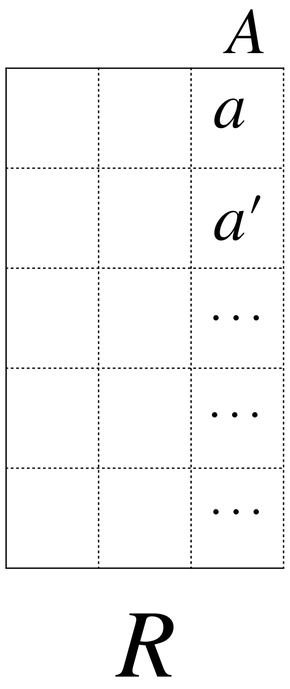
$$a \overset{?}{\geq} b' \iff (a_3 \oplus b'_3) \wedge a'_3$$
$$\oplus\ ((a_3 \oplus b'_3) \oplus 1) \wedge (a_2 \oplus b'_2) \wedge a_2$$
$$\oplus\ ((a_3 \oplus b'_3) \oplus 1) \wedge ((a_2 \oplus b'_2) \oplus 1) \wedge (a_1 \oplus b'_1) \wedge a_1$$
$$\oplus\ ((a_3 \oplus b'_3) \oplus 1) \wedge ((a_2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

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

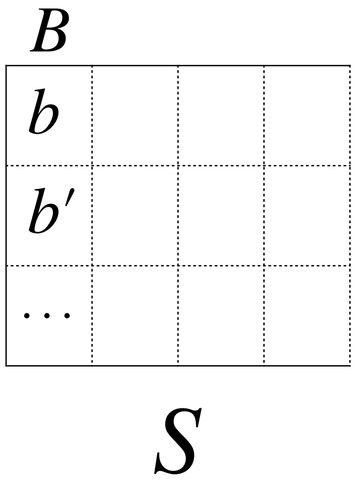
EXAMPLE: MESSAGE BATCHING

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

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



$R.A \geq S.B$



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

Compute XORs
(no communication)

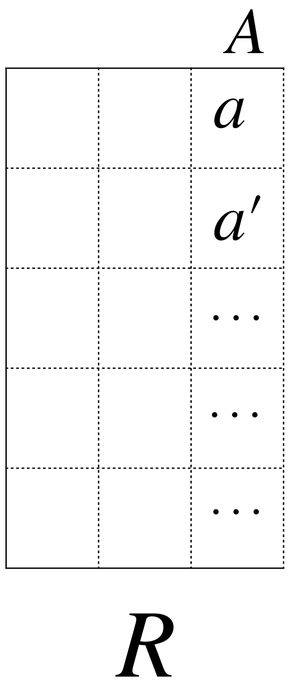
$$a \overset{?}{\geq} b' \iff (a_3 \oplus b'_3) \wedge a'_3$$
$$\oplus\ ((a_3 \oplus b'_3) \oplus 1) \wedge (a_2 \oplus b'_2) \wedge a_2$$
$$\oplus\ ((a_3 \oplus b'_3) \oplus 1) \wedge ((a_2 \oplus b'_2) \oplus 1) \wedge (a_1 \oplus b'_1) \wedge a_1$$
$$\oplus\ ((a_3 \oplus b'_3) \oplus 1) \wedge ((a_2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

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

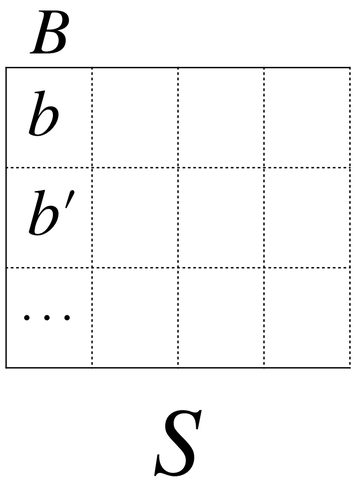
EXAMPLE: MESSAGE BATCHING

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

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



$R.A \geq S.B$



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

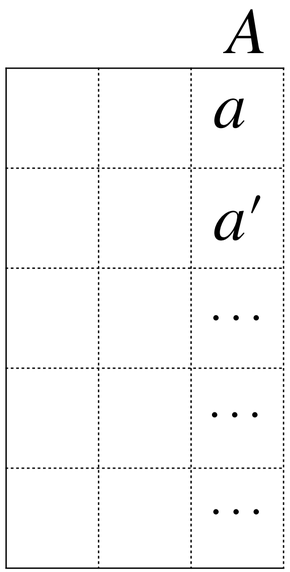
$$a \overset{?}{\geq} b' \iff c'_1 \wedge a_3 \oplus c'_2 \wedge c'_3 \wedge a_2 \oplus c'_4 \wedge c'_5 \wedge c'_6 \wedge a_1 \oplus c'_4 \wedge c'_5 \wedge c'_7 \wedge (c_8 \wedge b'_0)$$

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

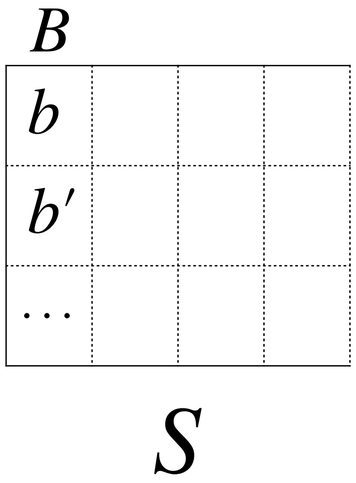
EXAMPLE: MESSAGE BATCHING

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

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



$R.A \geq S.B$



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

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

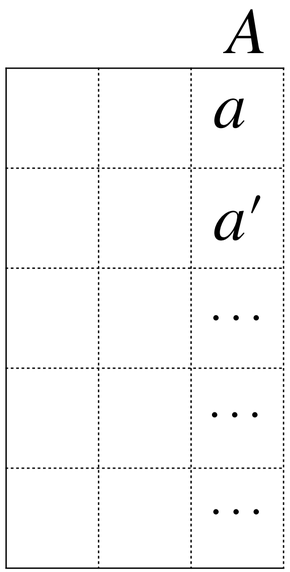
$$a' \overset{?}{\geq} b \iff \cdots$$

Compute independent ANDs
(1 round of communication)

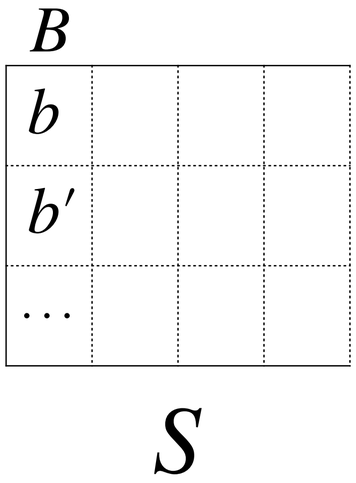
EXAMPLE: MESSAGE BATCHING

$\ell = 4$ *bits*

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



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



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

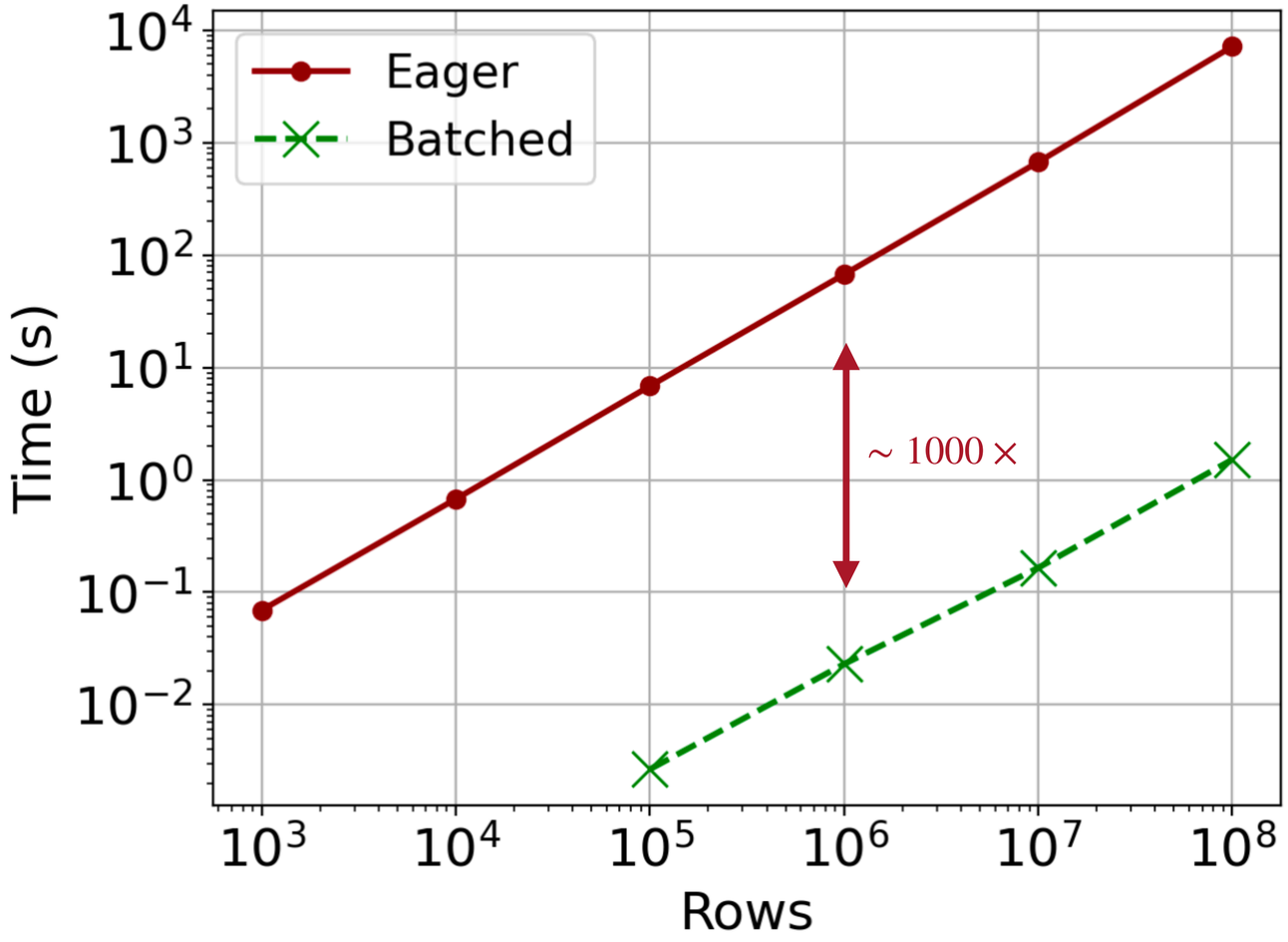
$$a \overset{?}{\geq} b' \iff c'_1 \wedge a_3 \oplus c'_2 \wedge c'_3 \wedge a_2 \oplus c'_4 \wedge c'_5 \wedge c'_6 \wedge a_1 \oplus c'_4 \wedge c'_5 \wedge c'_7 \wedge (c_8 \wedge b'_0)$$

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

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

Independent on the cardinality of R and S

EFFECT OF MESSAGE BATCHING (LAN)



- Eager: Message batching disabled (one network I/O per row)
- Batched: Message batching enabled

Lower is better

* Parties deployed on AWS EC2 r5.xlarge instances (us-east-2)

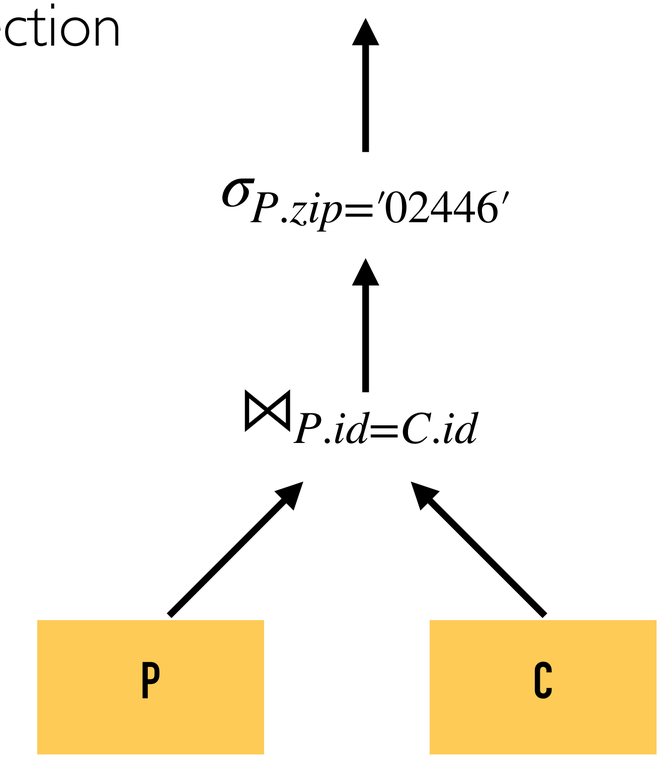
SECRECY's CORE CONTRIBUTIONS

1. ***Redesigned MPC primitives that work directly on relations instead of individual records***
   - Amortize network I/O
   - Make secret-sharing competitive in high-latency (WAN) environments
2. ***Analytical cost model based on secure computation and communication primitives***
   - Operation cost (number of MPC operations)
   - Synchronization cost (number of communication rounds)
   - Composition cost (extra cost of composing relational operators)

LOGICAL TRANSFORMATION RULES (CLEARTEXT DATABASES)

σ : selection

$\bowtie$: join

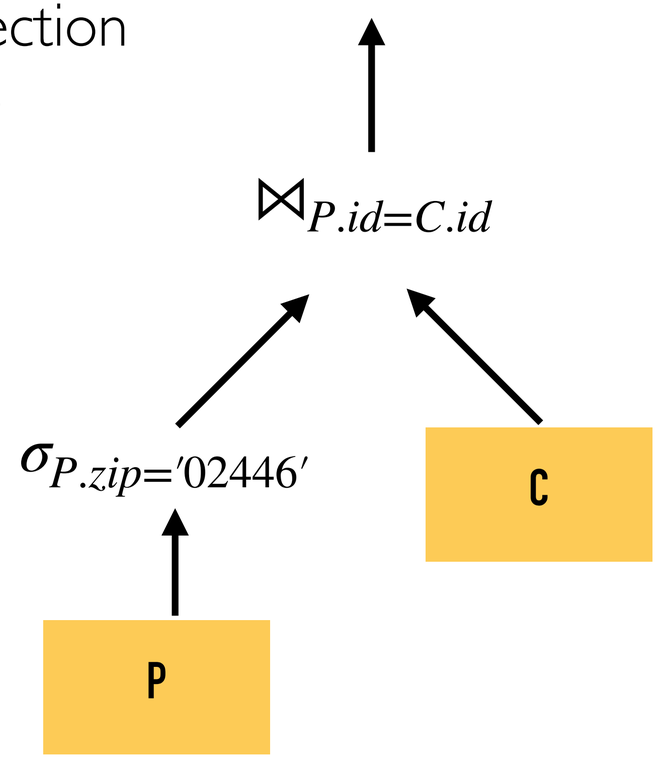



```
SELECT P.id
FROM Patients as P, Clients as C
WHERE P.id = C.id
AND P.zip='02446'
```

"Find the IDs of patients who are also clients (of an insurance company) and live in Brookline"

LOGICAL TRANSFORMATION RULES (CLEARTEXT DATABASES)

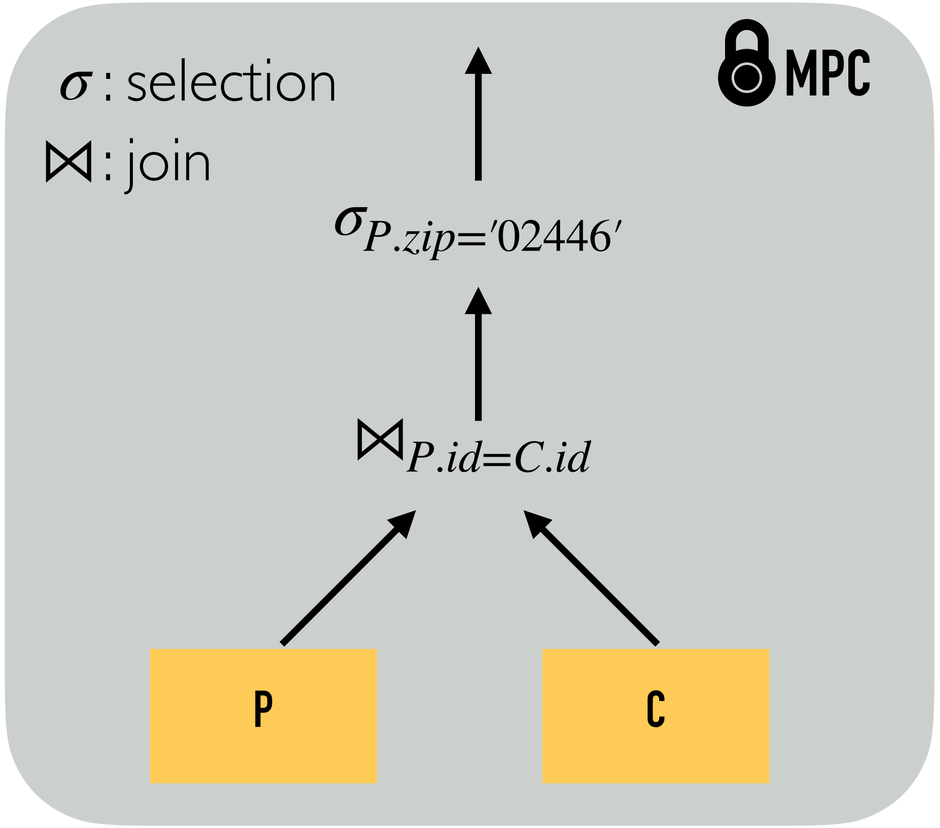
σ : selection
$\bowtie$: join

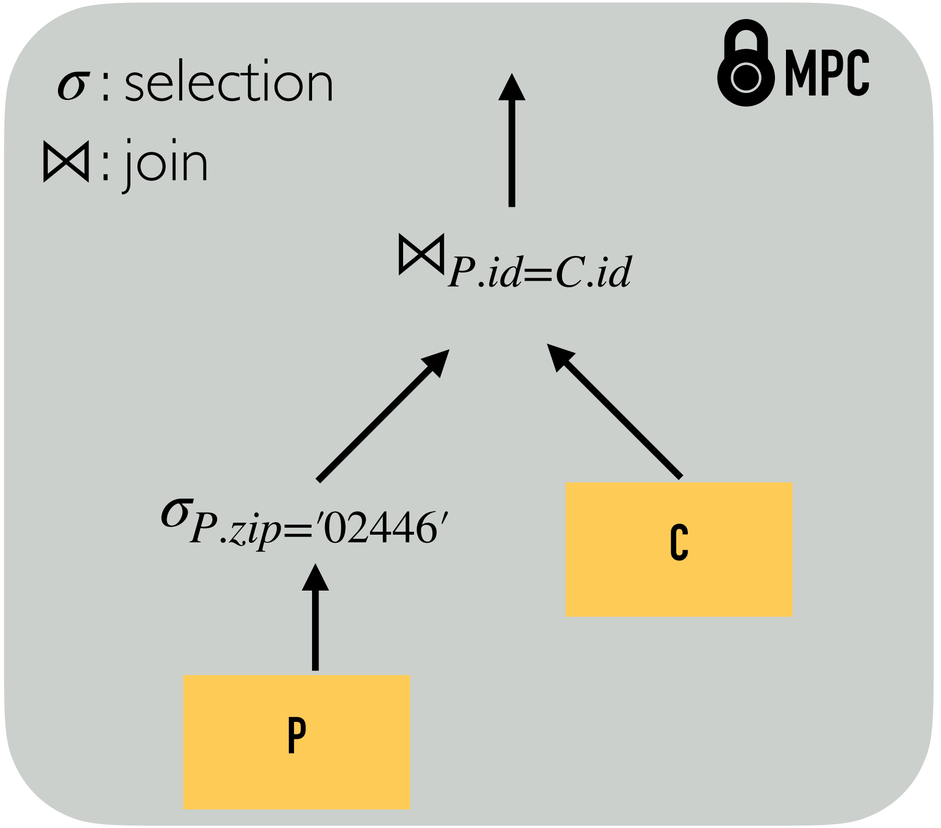



```
SELECT P.id
FROM Patients as P, Clients as C
WHERE P.id = C.id
AND P.zip='02446'
```

Pushing the selection down reduces the size of intermediate data and improves performance

CLEARTEXT OPTIMIZATIONS ARE NOT ALWAYS EFFECTIVE UNDER MPC

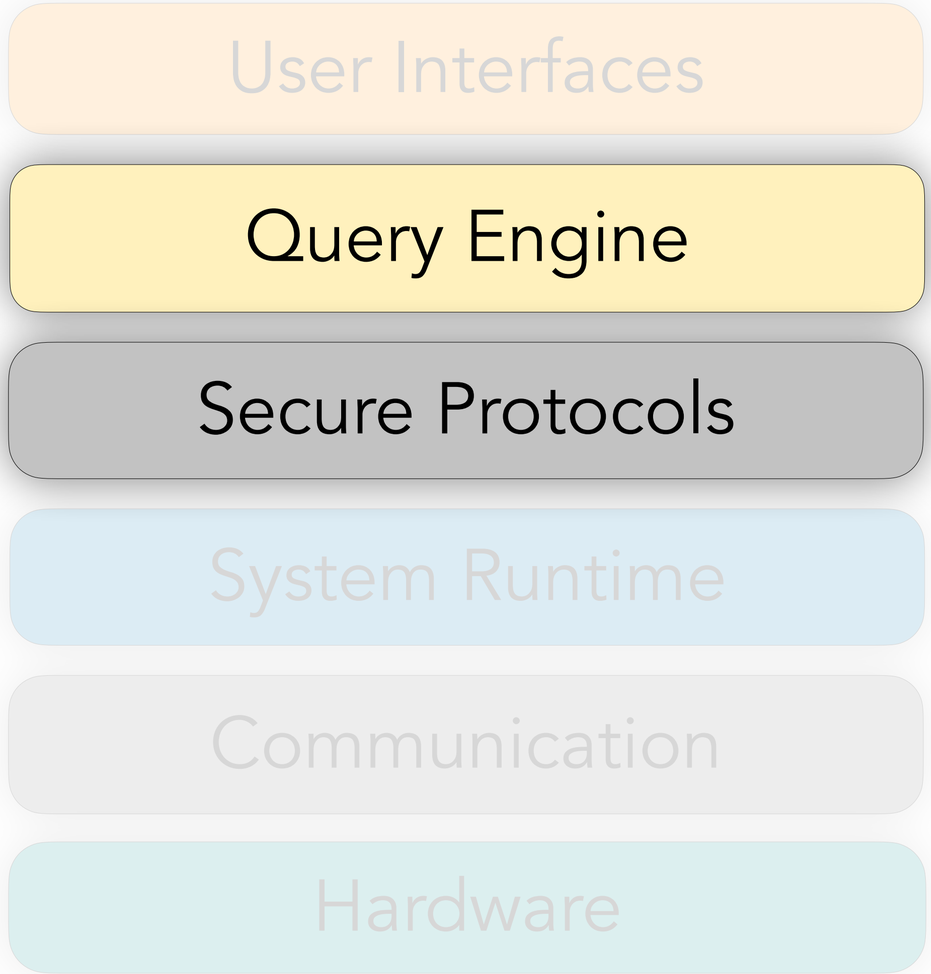




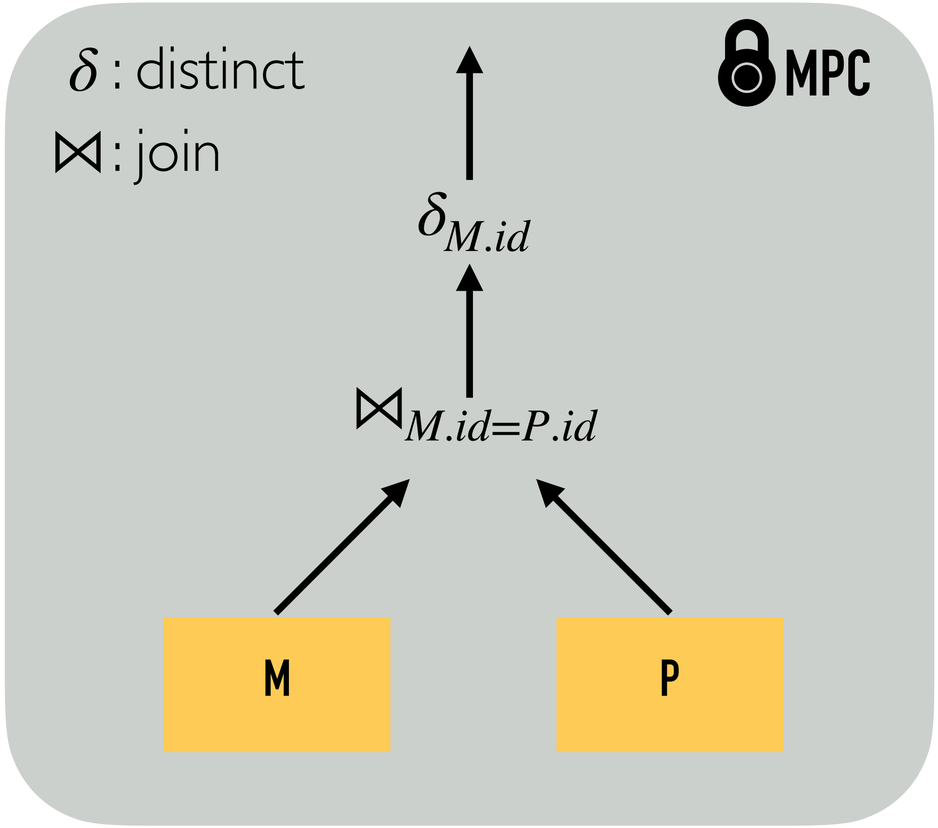
Pushing the selection before the `JOIN` *does not improve* `JOIN`*'s performance under MPC*

(since the oblivious selection does not remove any tuples from P)

OPERATOR REORDERING STILL MAKES SENSE UNDER MPC



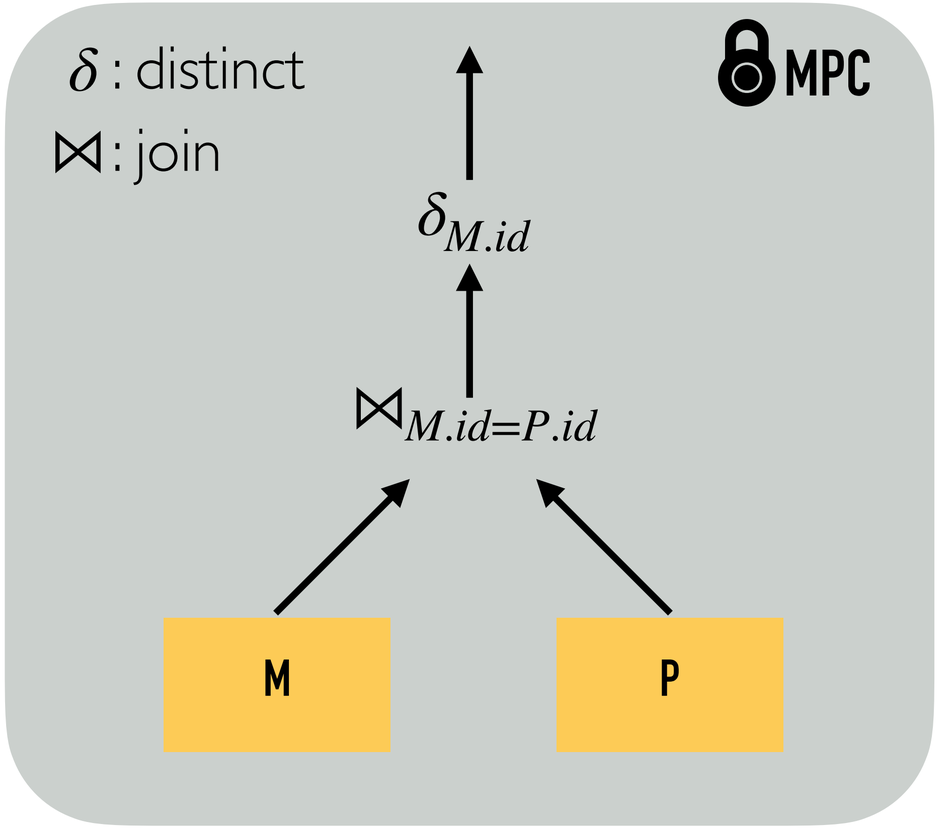
OPERATOR REORDERING STILL MAKES SENSE UNDER MPC




```
SELECT DISTINCT M.id
FROM Medication as M, Prescribed as P
WHERE M.id = P.id
```

"Find the distinct medication IDs that have been prescribed to patients"

OPERATOR REORDERING STILL MAKES SENSE UNDER MPC



$O(n^2 \log^2 n)$ operations / messages

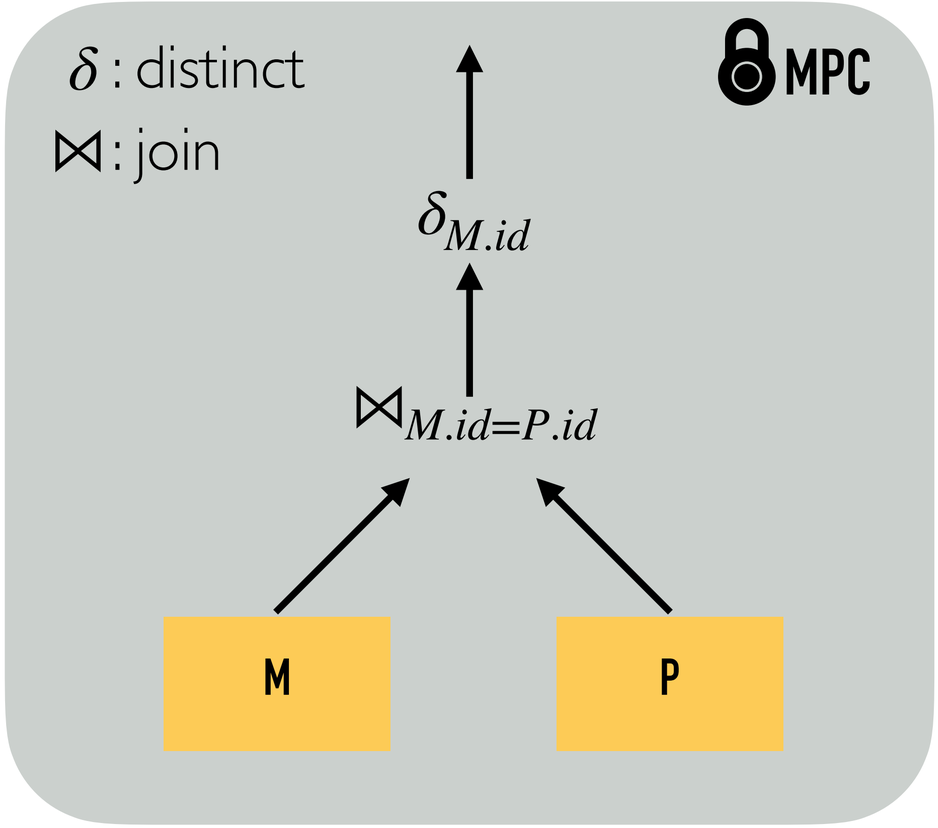
$O(\log^2 n)$ rounds $O(n^2)$ space

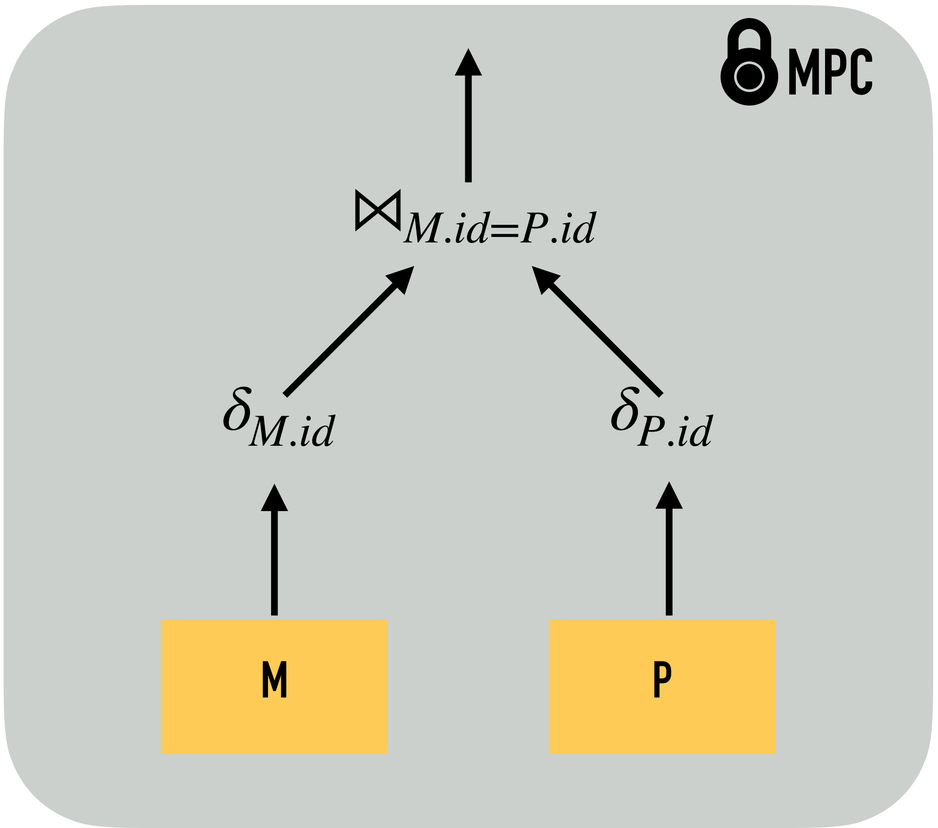
```
SELECT DISTINCT M.id
FROM Medication as M, Prescribed as P
WHERE M.id = P.id
```

$$|\mathtt{M}| = |\mathtt{P}| = n$$

* Assuming the distinct operator is based on a sorting network

EXAMPLE: DISTINCT PUSH-DOWN IN SECRECY





$O(n^2 \log^2 n)$ operations / messages

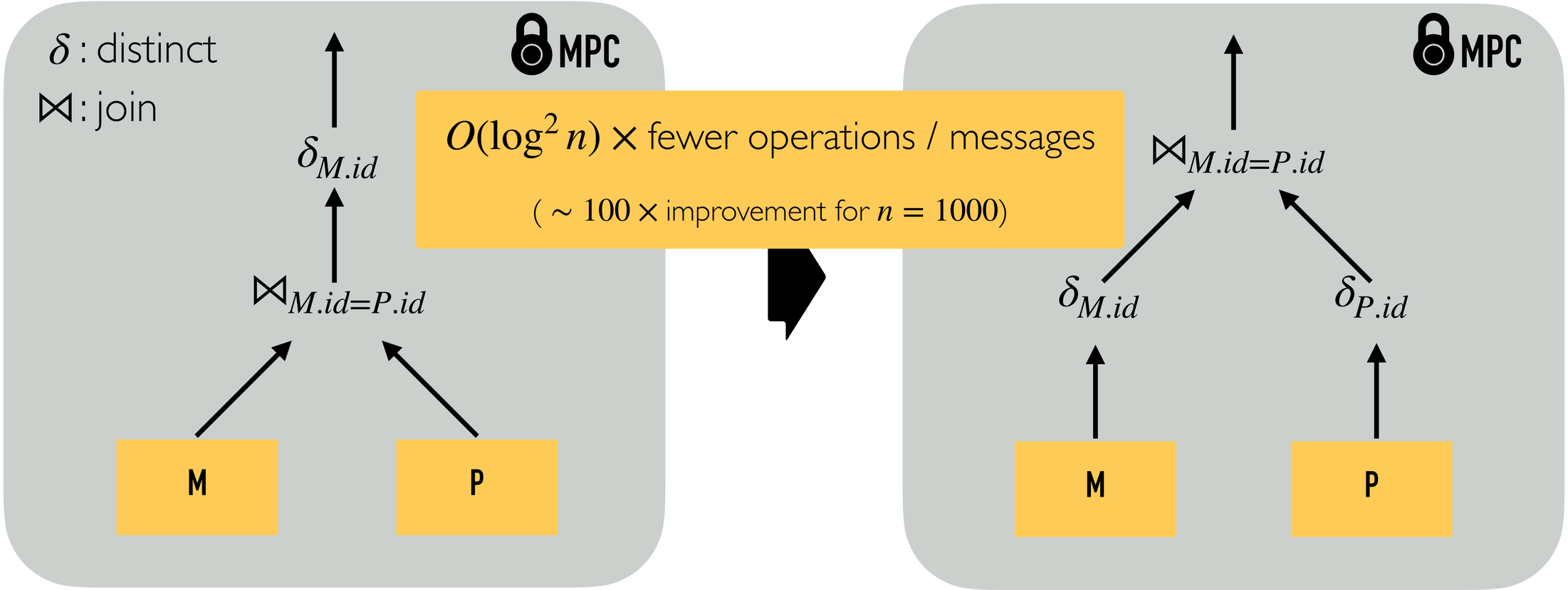
$O(\log^2 n)$ rounds $O(n^2)$ space

$O(n^2)$ operations / messages

$\sim 4\times$ fewer rounds $O(n)$ space

* Assuming the distinct operator is based on a sorting network

EXAMPLE: DISTINCT PUSH-DOWN IN SECRECY



$O(n^2 \log^2 n)$ operations / messages

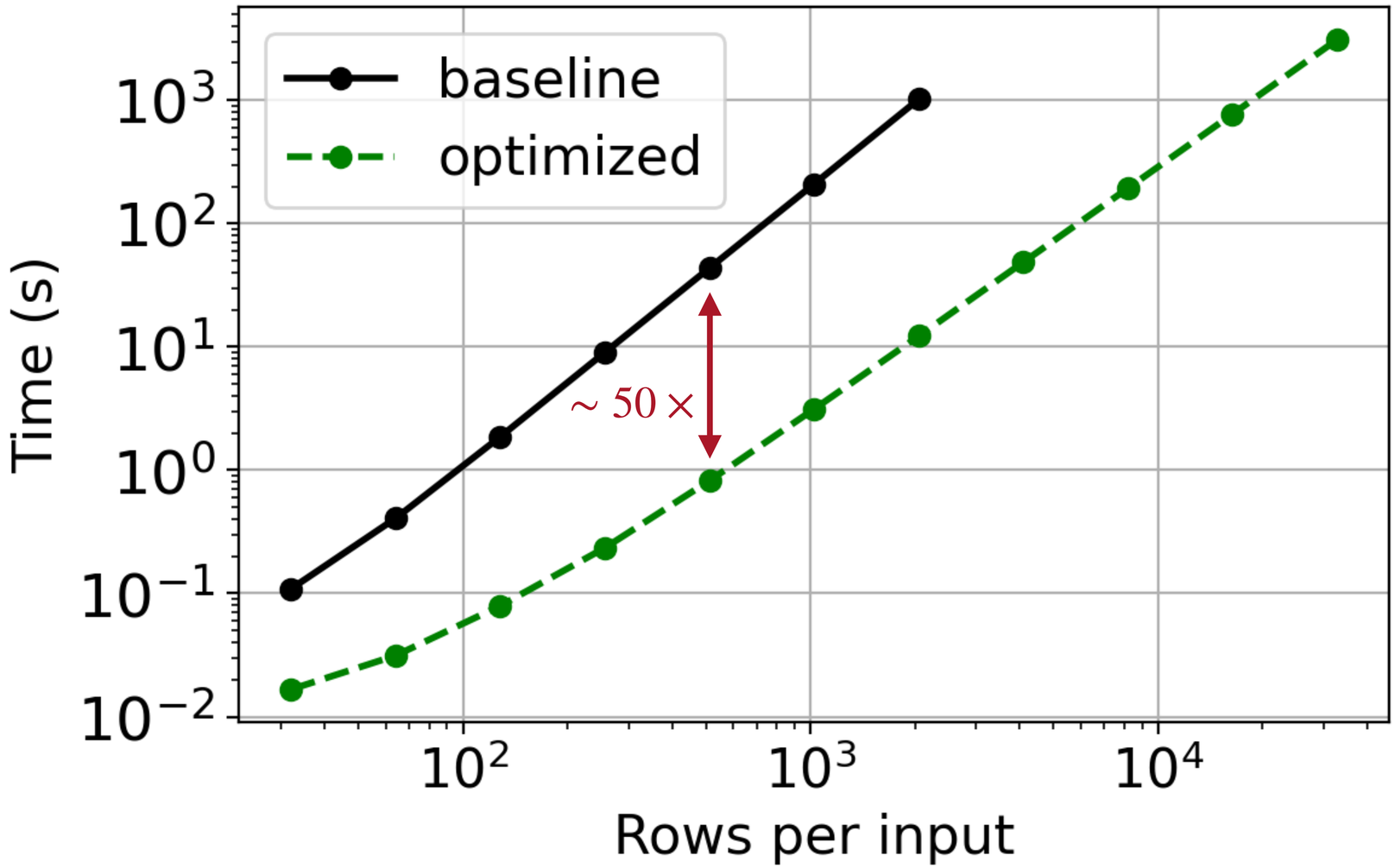
$O(\log^2 n)$ rounds $O(n^2)$ space

$O(n^2)$ operations / messages

~ 4 × fewer rounds $O(n)$ space

* Assuming the distinct operator is based on a sorting network

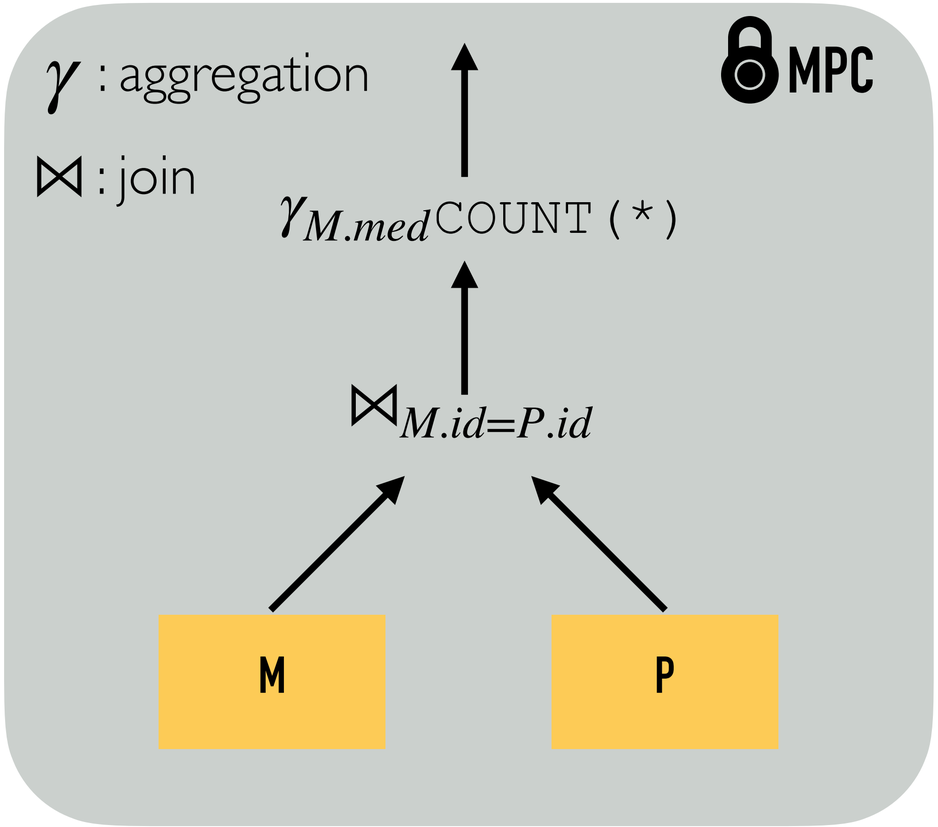
EFFECT OF DISTINCT PUSH-DOWN (LAN)



Lower is better

* Parties deployed on AWS EC2 r5.xlarge instances (us-east-2)

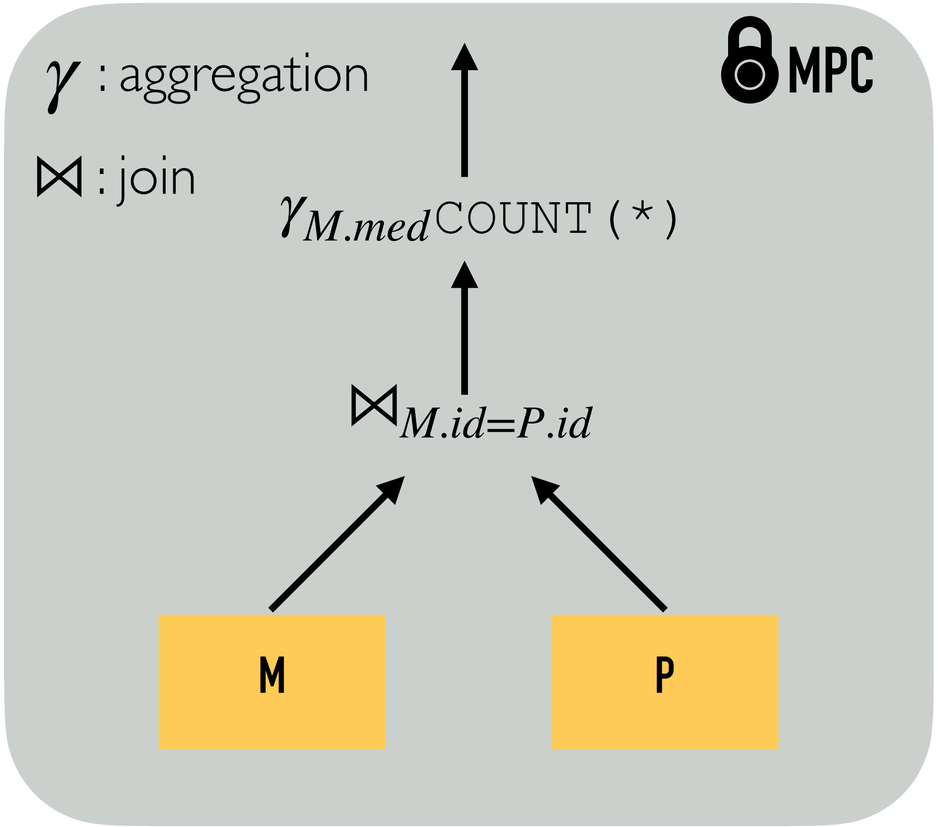
EXAMPLE: OPERATOR DECOMPOSITION IN SECRECY




```
SELECT M.med, COUNT(*)
FROM Medication as M, Patients as P
WHERE M.id = P.id
GROUP-BY M.med
```

"Count the number of patients per prescribed medication"

EXAMPLE: OPERATOR DECOMPOSITION IN SECRECY

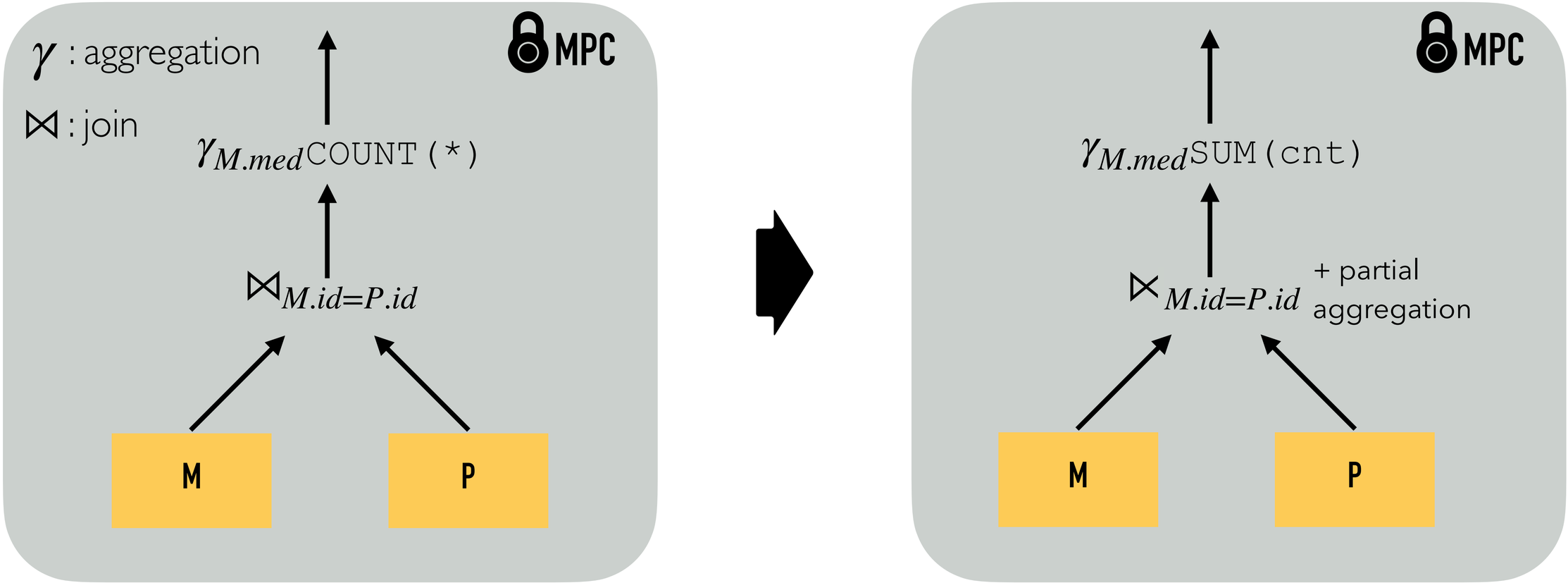



```
SELECT M.med, COUNT(*)
FROM Medication as M, Patients as P
WHERE M.id = P.id
GROUP-BY M.med
```

We can decompose the aggregation in two parts and push the first (and most expensive one) down

Applying `GROUP-BY` *after the join will require materializing the cartesian product* $M \times P$

EXAMPLE: OPERATOR DECOMPOSITION IN SECRECY

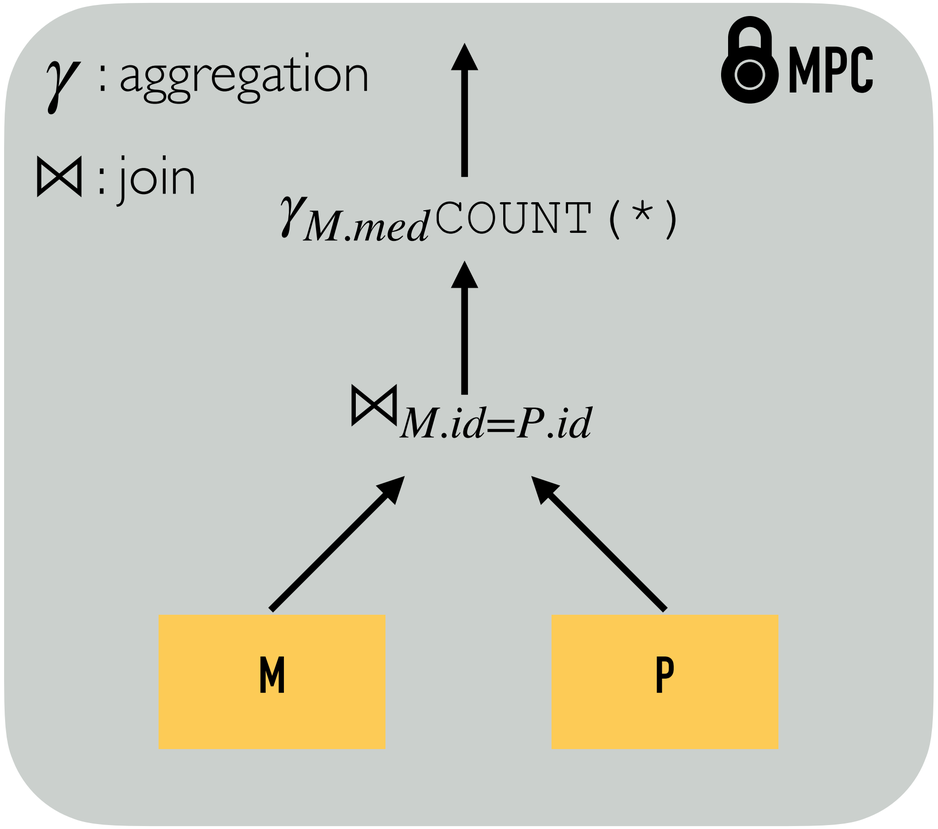


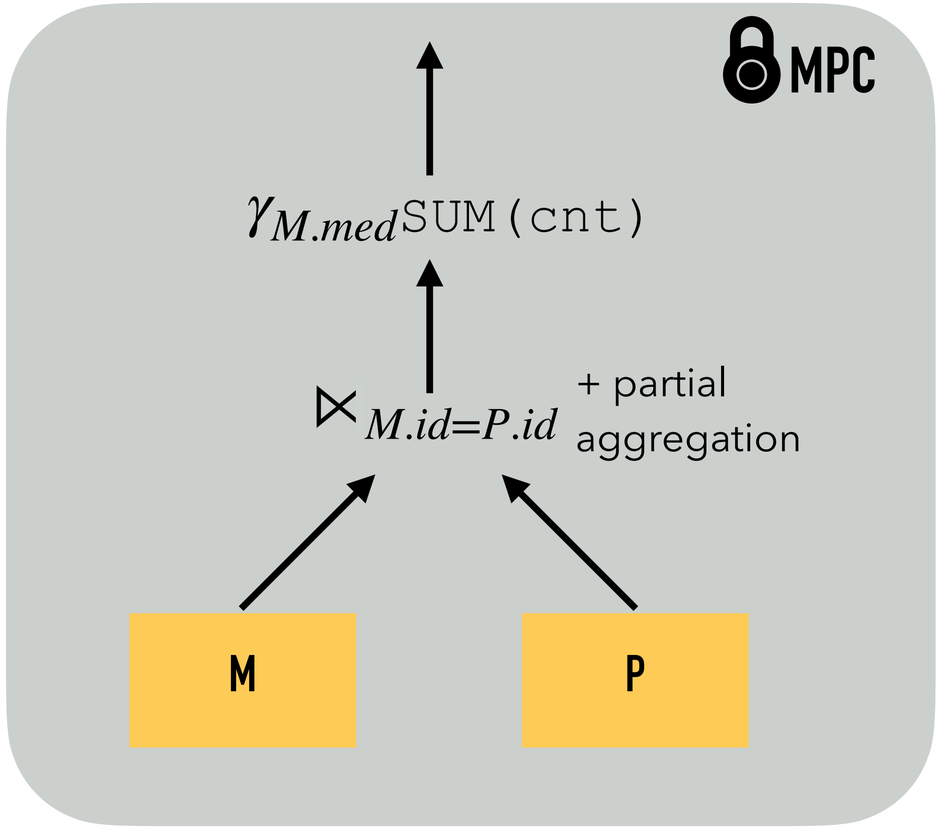
Applying `GROUP-BY` *after the join will require materializing the cartesian product* $M \times P$

`cnt` *is the number of times each* `id` *in* `M` *matched with an* `id` *in* `P` *during the* `SEMI-JOIN`

EXAMPLE: OPERATOR DECOMPOSITION IN SECRECY

* Assuming the group-by operator is based on a sorting network





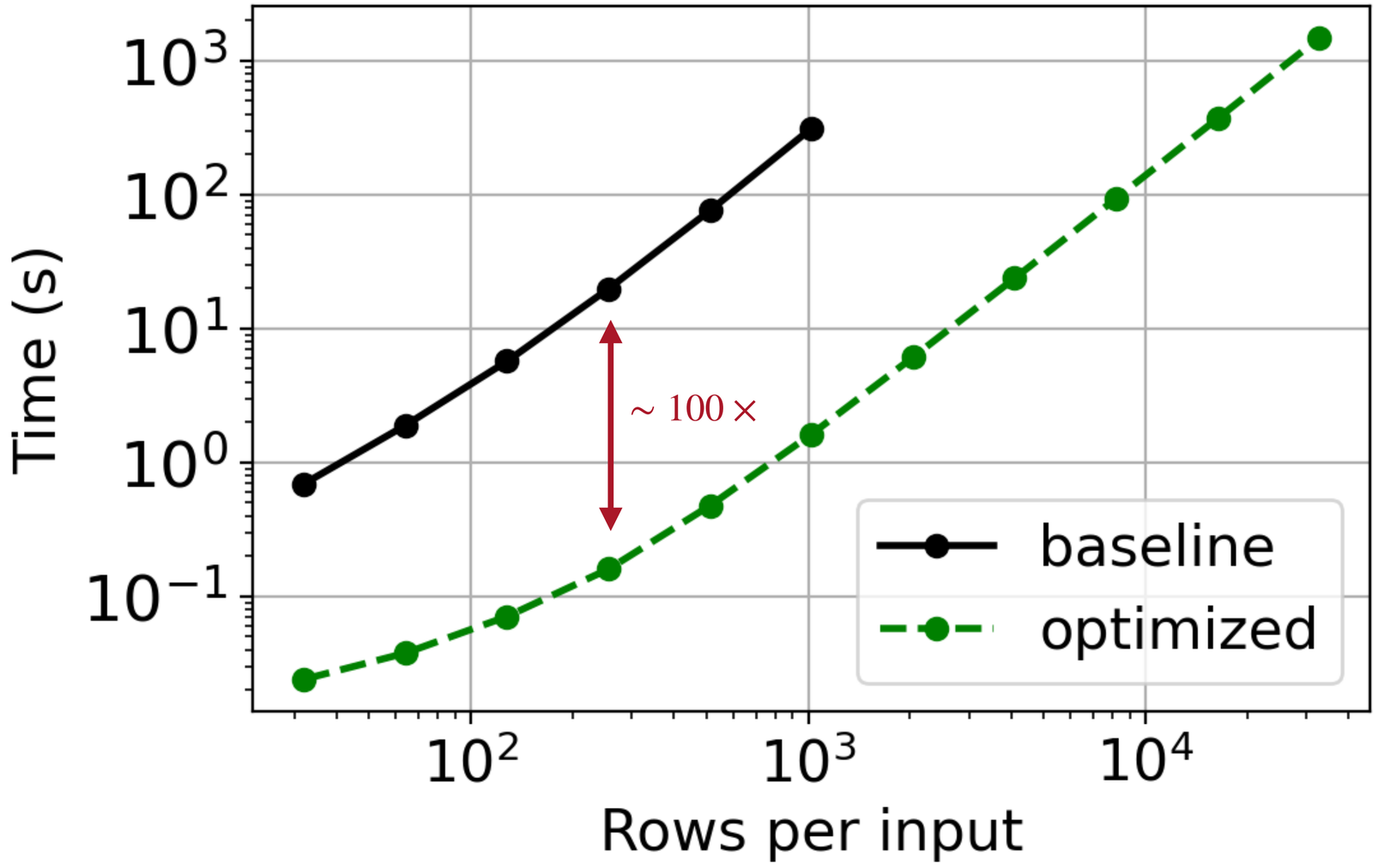
$O(n^2 \log^2 n)$ *operations / messages*

$O(n^2)$ *rounds* $O(n^2)$ *space*

$O(n^2)$ operations / messages

$O(n)$ *rounds* $O(n)$ *space*

EFFECT OF JOIN-AGGREGATION DECOMPOSITION (LAN)



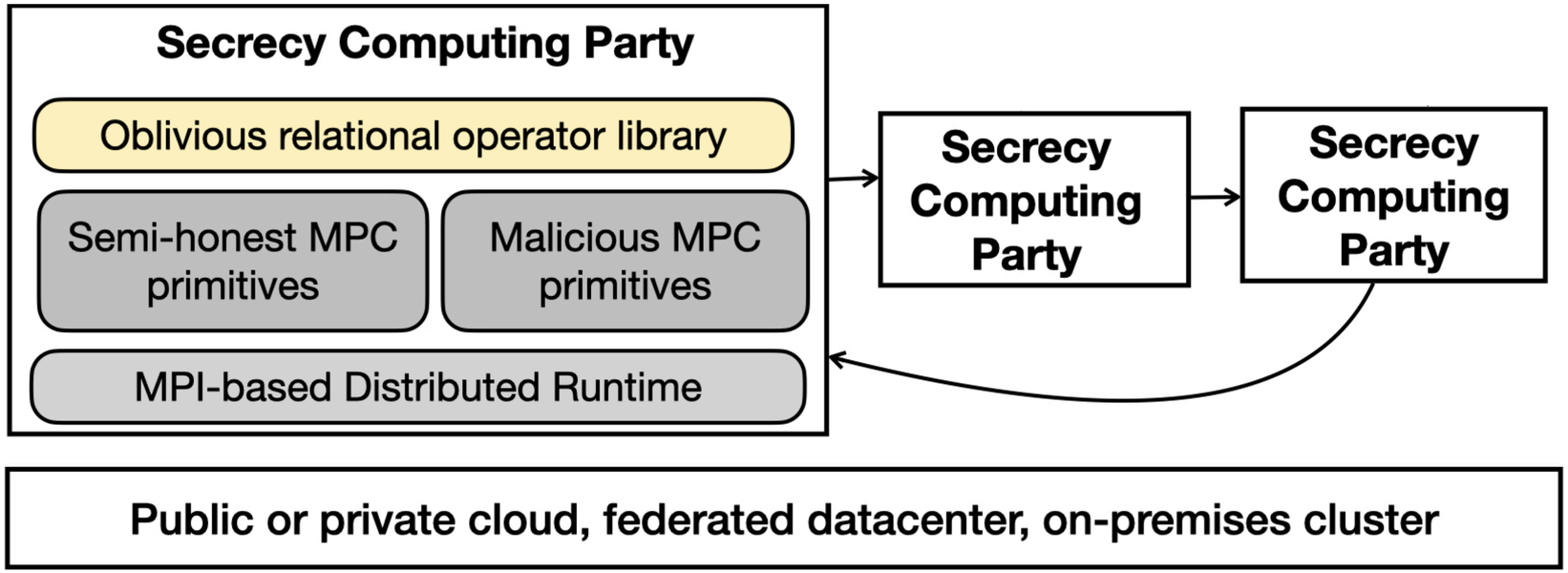
Lower is better

* Parties deployed on AWS EC2 r5.xlarge instances (us-east-2)

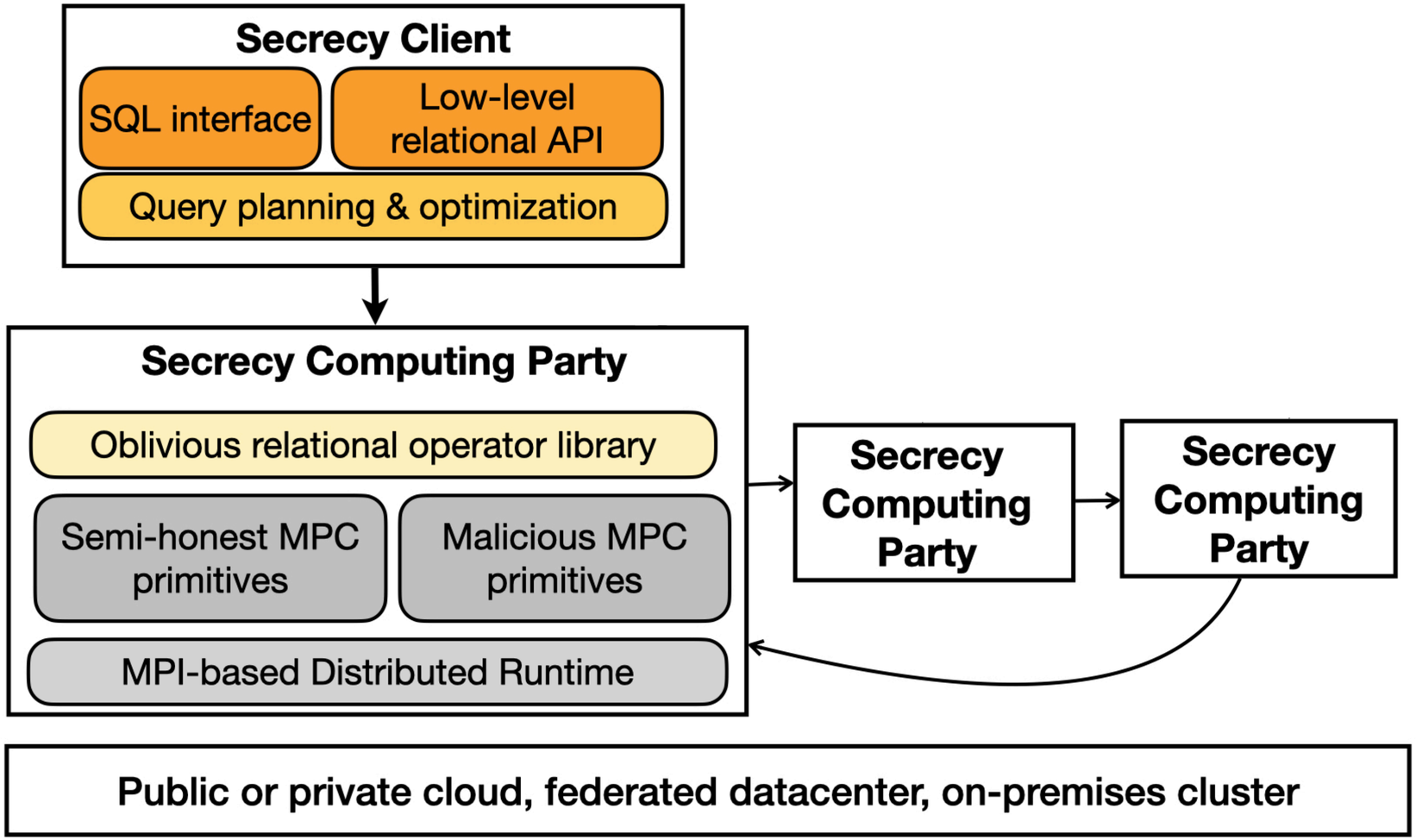
SECRECY's CORE CONTRIBUTIONS

1. ***Redesigned MPC primitives that work directly on relations instead of individual records***
   - Amortize network I/O
   - Make secret-sharing competitive in high-latency (WAN) environments
2. ***Analytical cost model based on secure computation and communication primitives***
   - Operation cost (number of MPC operations)
   - Synchronization cost (number of communication rounds)
   - Composition cost (extra cost of composing relational operators)
3. ***Volcano-style query processor for vectorized MPC execution***
   - Logical optimizations (e.g., operator reordering and decomposition)
   - Physical optimizations (e.g., message batching, operator fusion)
   - Protocol-specific optimizations (e.g., dual sharing)

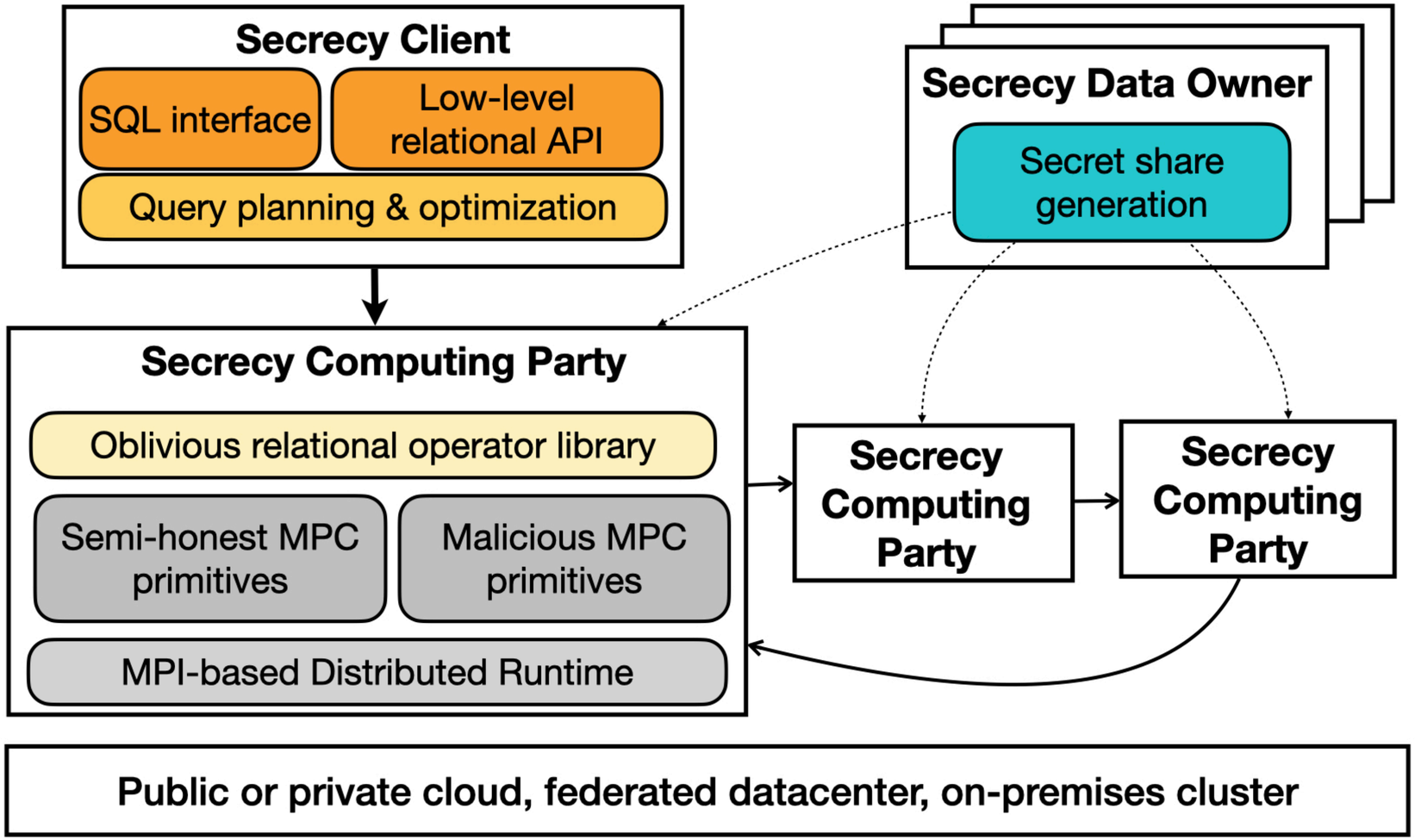
SECRECY OVERVIEW



SECRECY OVERVIEW



SECRECY OVERVIEW



Performance on real and synthetic queries

SUMMARY OF RESULTS

1. Secrecy optimizations can improve performance by orders of magnitude
2. Secrecy scales to millions of input rows
3. Secrecy outperforms state-of-the-art frameworks
4. Relational operators scale well with configurable memory footprints
5. Secrecy can perform millions of primitive MPC operations per second

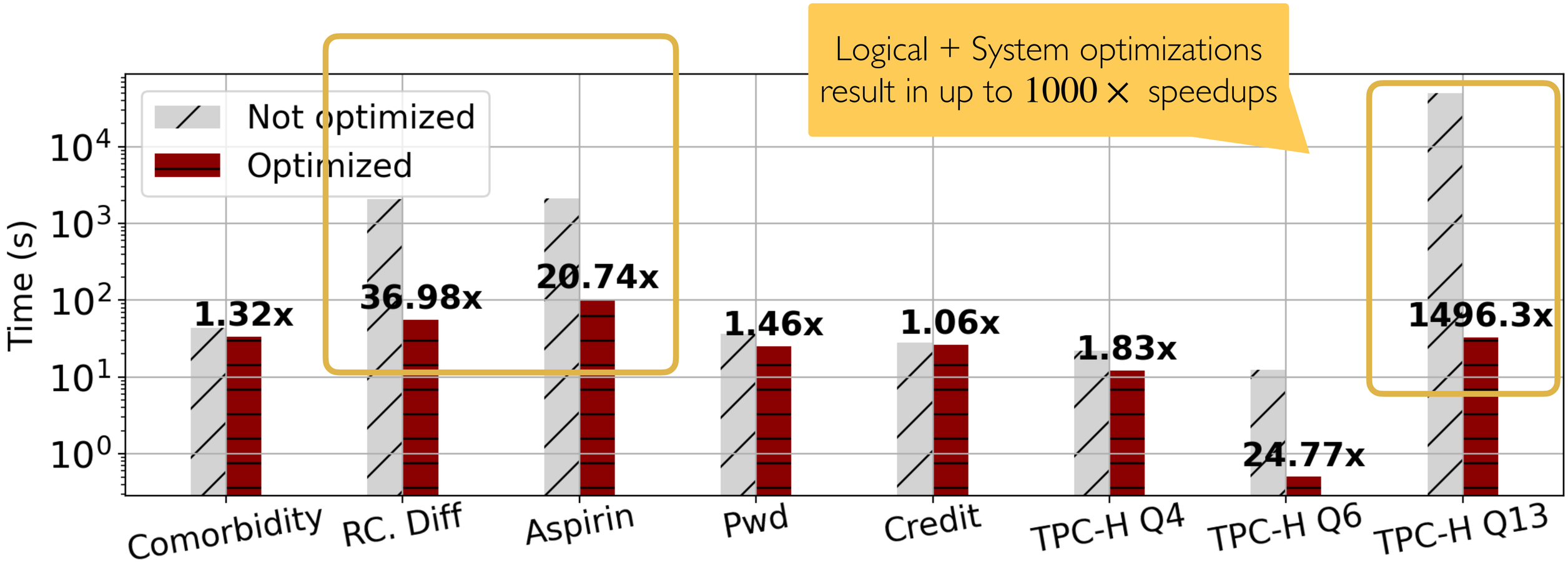
J. Liagouris, V. Kalavri, M. Faisal, M. Varia. *Secrecy: Secure Collaborative Analytics on Secret-shared Data*. arXiv:2102.01048, 2021.

SUMMARY OF RESULTS

1. Secrecy optimizations can improve performance by orders of magnitude
2. Secrecy scales to millions of input rows
3. Secrecy outperforms state-of-the-art frameworks
4. Relational operators scale well with configurable memory footprints
5. Secrecy can perform millions of primitive MPC operations per second

J. Liagouris, V. Kalavri, M. Faisal, M. Varia. *Secrecy: Secure Collaborative Analytics on Secret-shared Data*. arXiv:2102.01048, 2021.

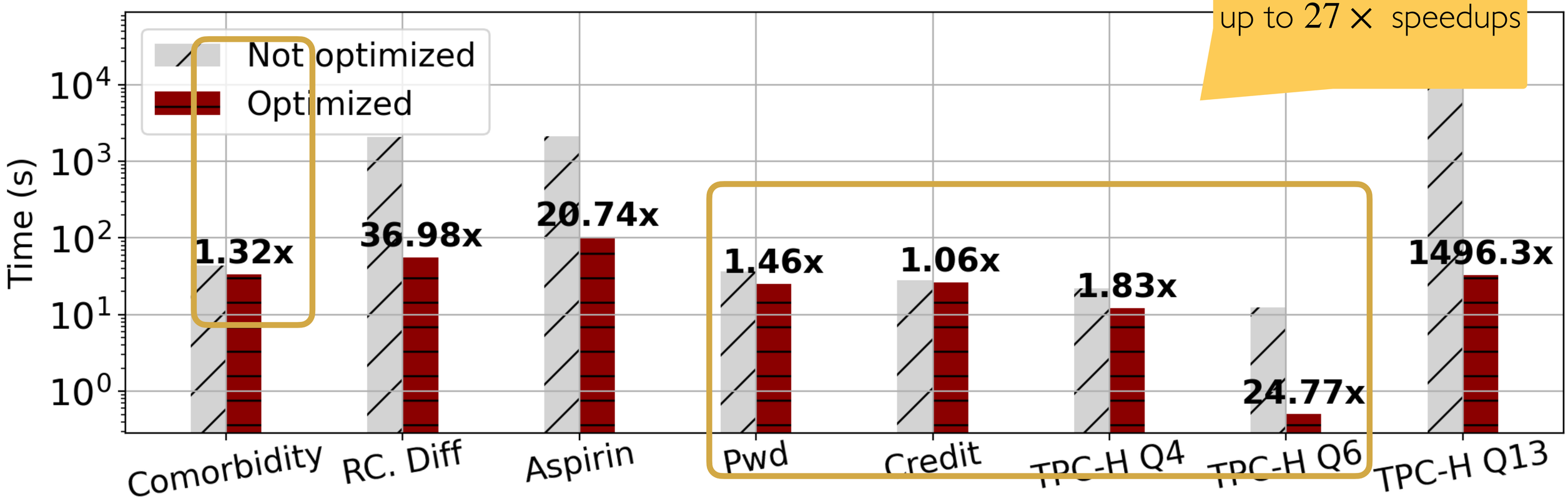
EFFECT OF SECRECY OPTIMIZATIONS ON REAL QUERIES (WAN)



* Parties deployed in three AWS regions: us-east-2 (Ohio), us-east-1 (Virginia), and us-west-1 (California)
* Reported times are for 1000 rows per input relation
* Not optimized plans use message batching too (otherwise the cost of MPC is prohibitive)

EFFECT OF SECRECY OPTIMIZATIONS ON REAL QUERIES (WAN)

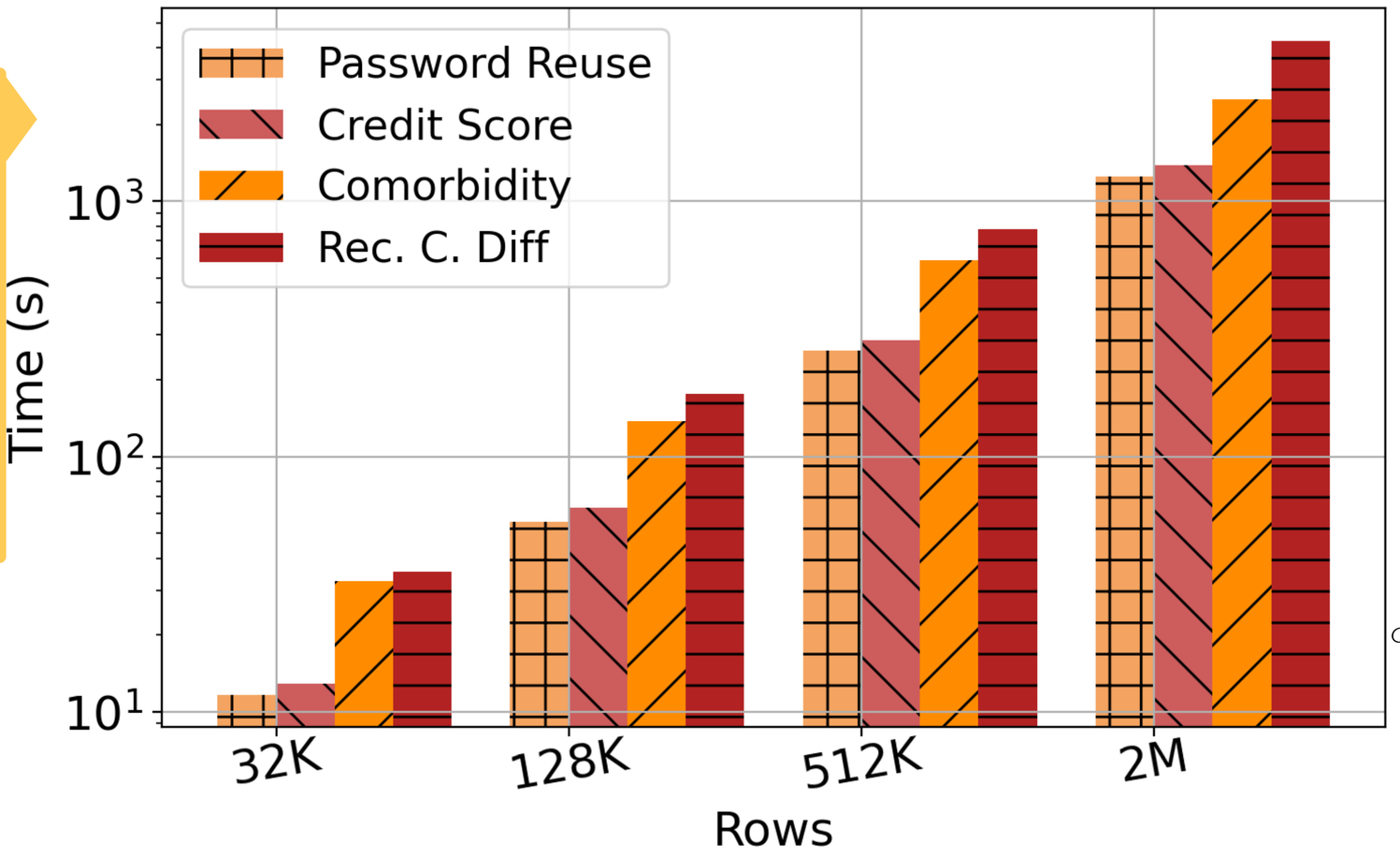
Protocol-specific optimizations result in up to 27 × speedups



* Parties deployed in three AWS regions: us-east-2 (Ohio), us-east-1 (Virginia), and us-west-1 (California)
* Reported times are for 1000 rows per input relation
* Not optimized plans use message batching too (otherwise the cost of MPC is prohibitive)

SECRECY's SCALING BEHAVIOR (LAN)

The cost of these queries is dominated by the oblivious `GROUP-BY` and `DISTINCT` operators



Rec. C. Diff scales to 2 million rows in ~1.2h

```
WITH rcd AS (
    SELECT pid, time, row_no() OVER
    (PARTITION BY pid ORDER BY time)
    FROM diagnosis
    WHERE diag=cdiff)
SELECT DISTINCT pid
FROM rcd r1 JOIN rcd r2 ON r1.pid = r2.pid
WHERE r2.time - r1.time >= 15 DAYS
AND r2.time - r1.time <= 56 DAYS
AND r2.row_no = r1.row_no + 1
```

"Find the distinct ids of patients who have been diagnosed with `cdiff` and have two consecutive infections between 15 and 56 days apart"

* Parties deployed on AWS EC2 r5.xlarge instances (us-east-2)

What about more complex queries?

TPC-H Q8: 7 JOINS + GROUP-BY + ORDER-BY

```
select
        o_year,
        sum(case
                when nation = '[NATION]'
                then volume
                else 0
        end) / sum(volume) as mkt_share
from (
        select
                extract(year from o_orderdate) as o_year,
                l_extendedprice * (1-l_discount) as volume,
                n2.n_name as nation
        from
                part,
                supplier,
                lineitem,
                orders,
                customer,
                nation n1,
                nation n2,
                region
        where
                p_partkey = l_partkey
                and s_suppkey = l_suppkey
                and l_orderkey = o_orderkey
                and o_custkey = c_custkey
                and c_nationkey = n1.n_nationkey
                and n1.n_regionkey = r_regionkey
                and r_name = '[REGION]'
                and s_nationkey = n2.n_nationkey
                and o_orderdate between date '1995-01-01' and date '1996-12-31'
                and p_type = '[TYPE]'
        ) as all_nations
group by
        o_year
order by
        o_year;
```

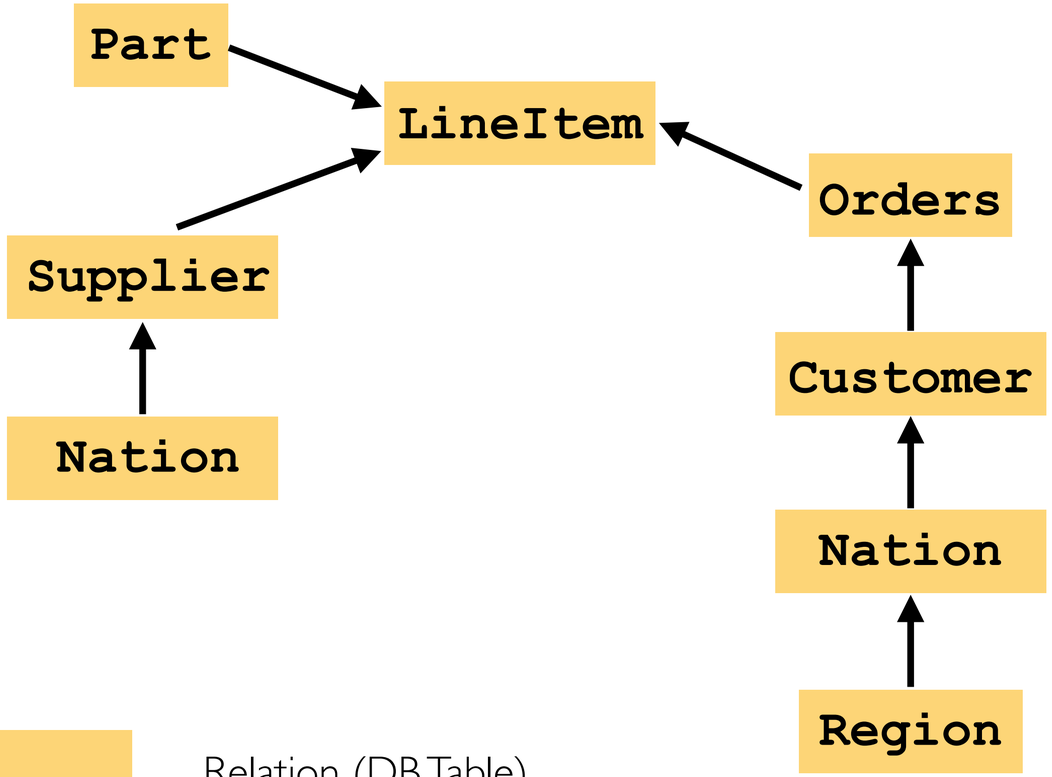
"This query determines how the market share of a given nation within a given region has changed over two years for a given part type"

A naive approach would require $O(n^8 \log^2 n)$ operations under MPC

TPC-H Q8: 7 JOINS + GROUP-BY + ORDER-BY

```
select
        o_year,
        sum(case
                when nation = '[NATION]'
                then volume
                else 0
        end) / sum(volume) as mkt_share
from (
        select
                extract(year from o_orderdate) as o_year,
                l_extendedprice * (1-l_discount) as volume,
                n2.n_name as nation
        from
                part,
                supplier,
                lineitem,
                orders,
                customer,
                nation n1,
                nation n2,
                region
        where
                p_partkey = l_partkey
                and s_suppkey = l_suppkey
                and l_orderkey = o_orderkey
                and o_custkey = c_custkey
                and c_nationkey = n1.n_nationkey
                and n1.n_regionkey = r_regionkey
                and r_name = '[REGION]'
                and s_nationkey = n2.n_nationkey
                and o_orderdate between date '1995-01-01' and date '1996-12-31'
                and p_type = '[TYPE]'
        ) as all_nations
group by
        o_year
order by
        o_year;
```

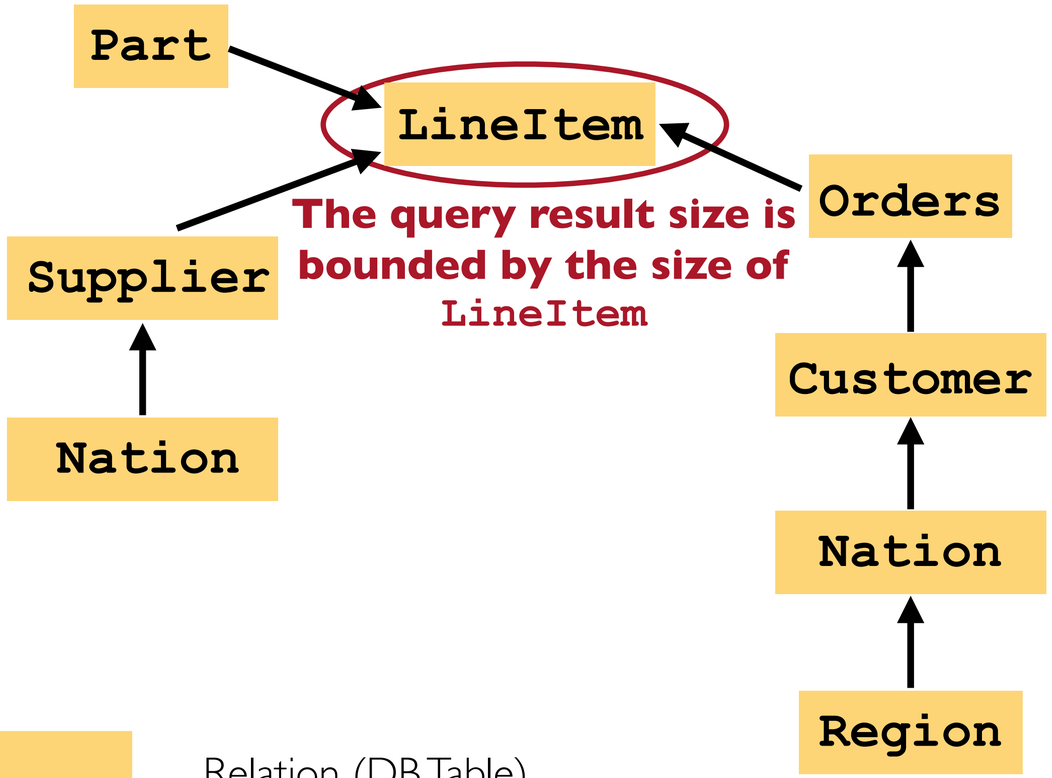
"This query determines how the market share of a given nation within a given region has changed over two years for a given part type"



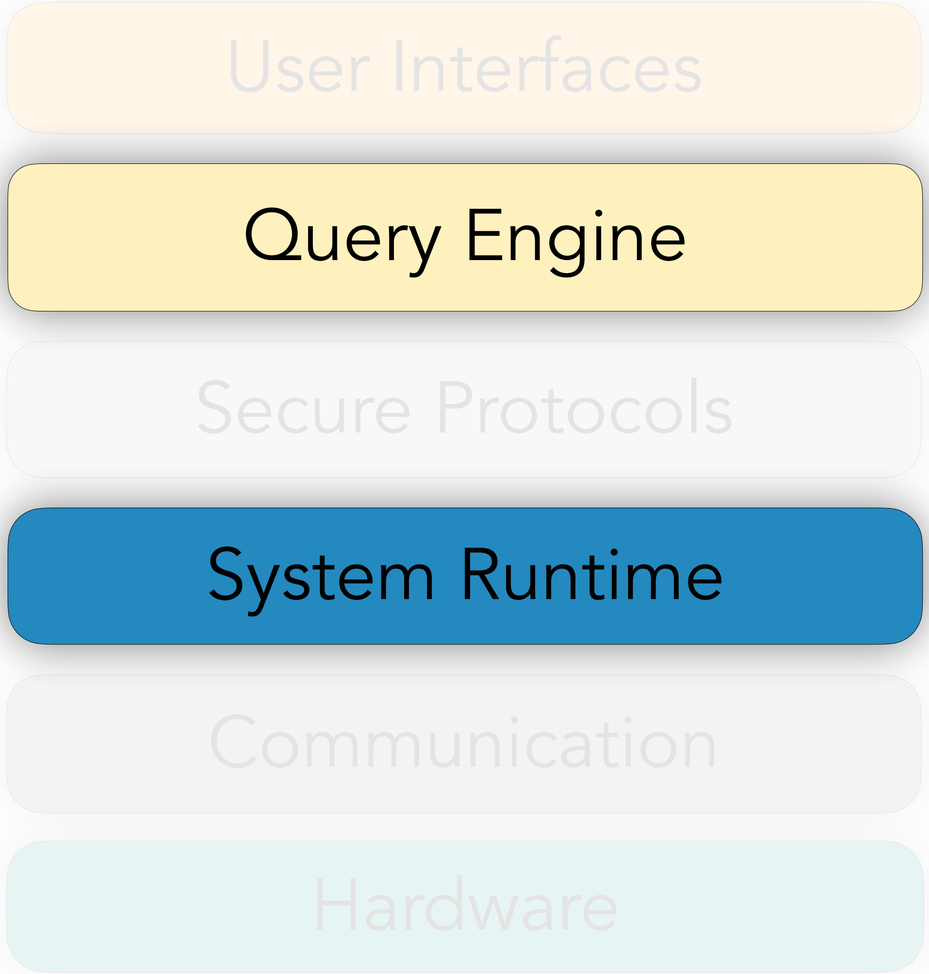
TPC-H Q8: 7 JOINS + GROUP-BY + ORDER-BY

```
select
        o_year,
        sum(case
                when nation = '[NATION]'
                then volume
                else 0
        end) / sum(volume) as mkt_share
from (
        select
                extract(year from o_orderdate) as o_year,
                l_extendedprice * (1-l_discount) as volume,
                n2.n_name as nation
        from
                part,
                supplier,
                lineitem,
                orders,
                customer,
                nation n1,
                nation n2,
                region
        where
                p_partkey = l_partkey
                and s_suppkey = l_suppkey
                and l_orderkey = o_orderkey
                and o_custkey = c_custkey
                and c_nationkey = n1.n_nationkey
                and n1.n_regionkey = r_regionkey
                and r_name = '[REGION]'
                and s_nationkey = n2.n_nationkey
                and o_orderdate between date '1995-01-01' and date '1996-12-31'
                and p_type = '[TYPE]'
        ) as all_nations
group by
        o_year
order by
        o_year;
```

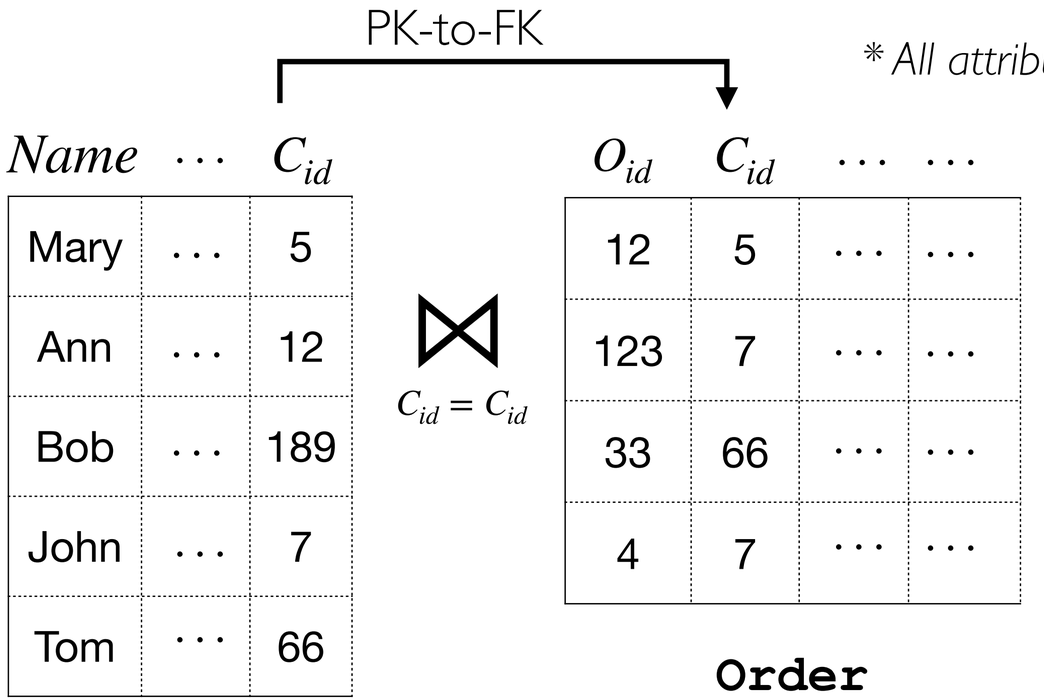
"This query determines how the market share of a given nation within a given region has changed over two years for a given part type"



LOGICAL + SYSTEM OPTIMIZATIONS TO SCALE COMPLEX QUERIES



EVALUATING JOINs USING SORT AND ODD-EVEN AGGREGATION



* *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 ODD-EVEN AGGREGATION

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

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 ODD-EVEN AGGREGATION

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 ODD-EVEN AGGREGATION

** 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 ODD–EVEN AGGREGATION

** 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}

$O(n \log^2 n)$ operations

$O(\log^2 n)$ rounds

$O(n)$ space

$n = |S \cup T|$

* Assuming the distinct operator is based on a sorting network

EVALUATING JOINs USING SORT AND ODD-EVEN AGGREGATION

** 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 ODD-EVEN AGGREGATION

** 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 odd-even aggregation phase to find "matched pairs of tuples" and mask the rest

$O(n \log n)$ operations

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

$O(n)$ space

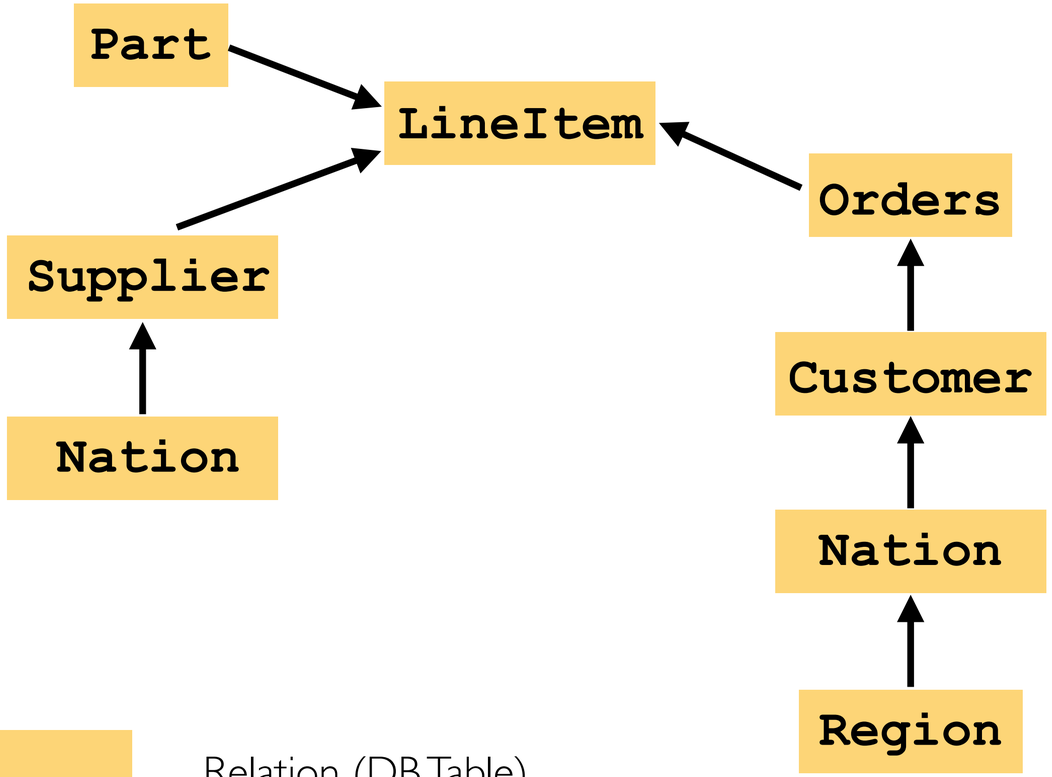
TPC-H Q8: 7 JOINS + GROUP-BY + ORDER-BY

"This query determines how the market share of a given nation within a given region has changed over two years for a given part type"

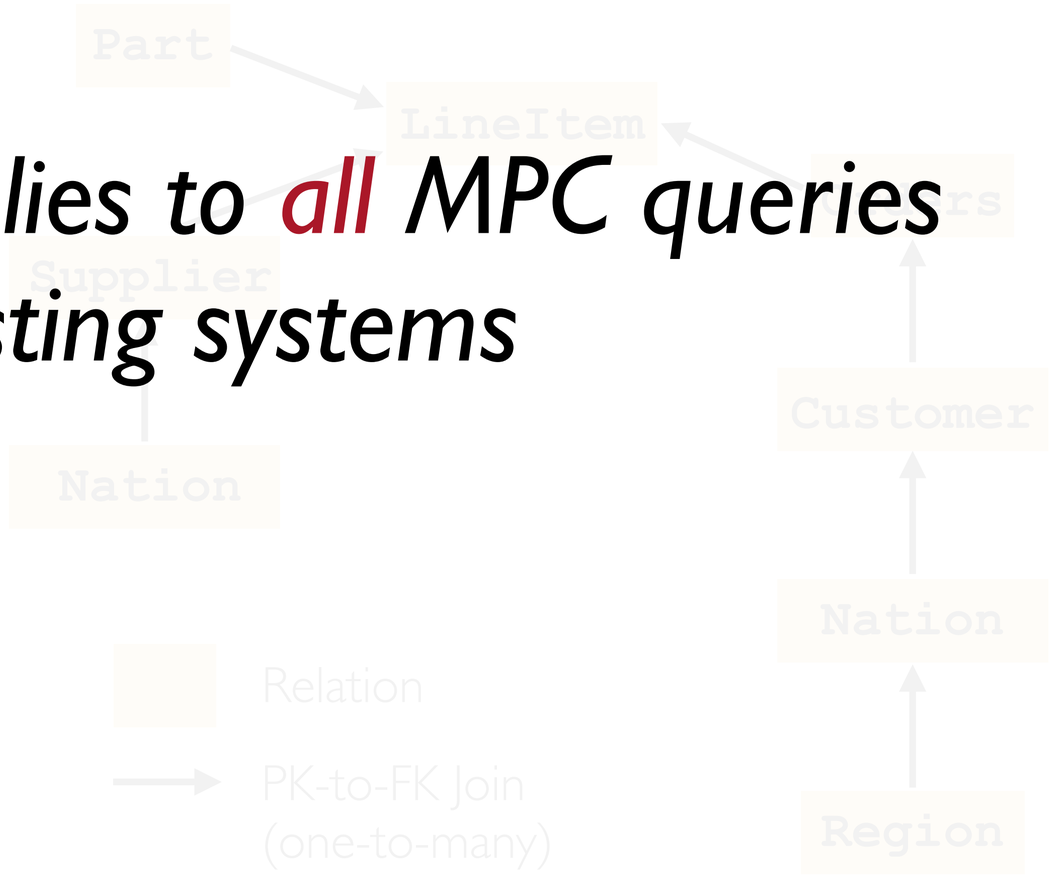
Using this simple idea, we can evaluate the **whole query** in:

- $O(n \log^2 n)$ operations
- $O(\log^2 n)$ rounds
- $O(n)$ space

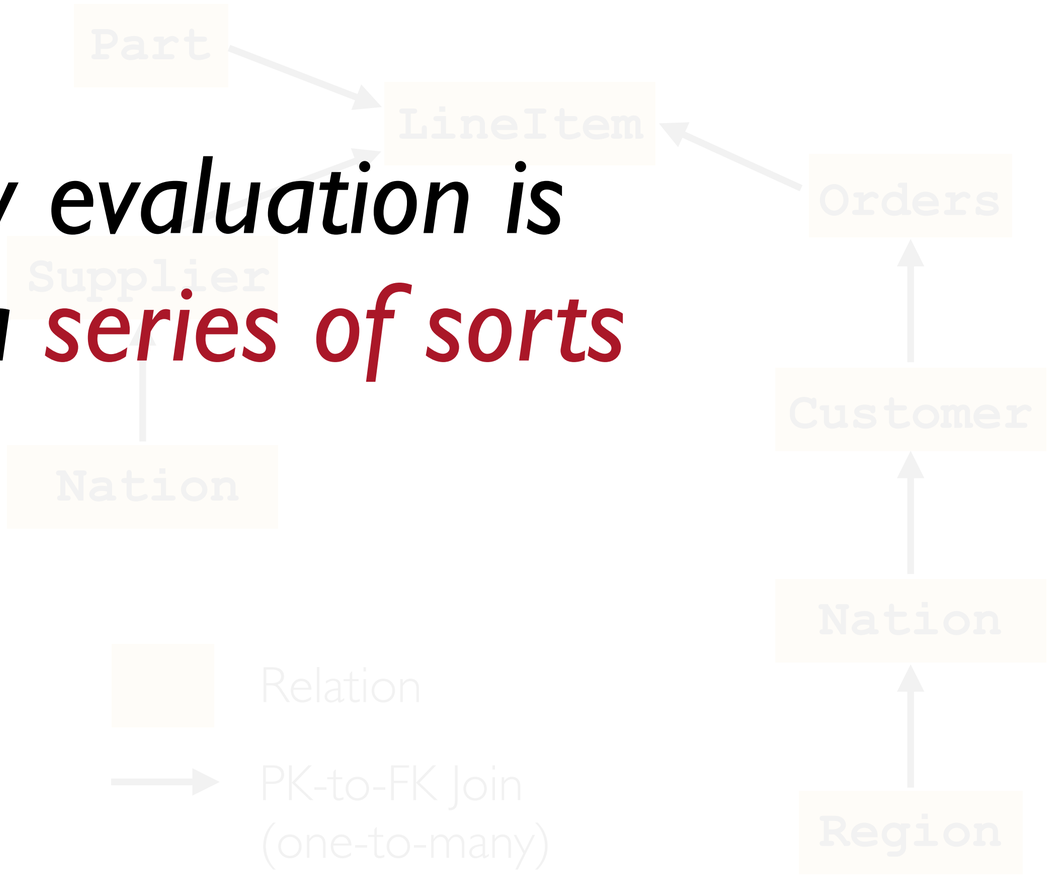
where n is the total number of input rows across all input relations



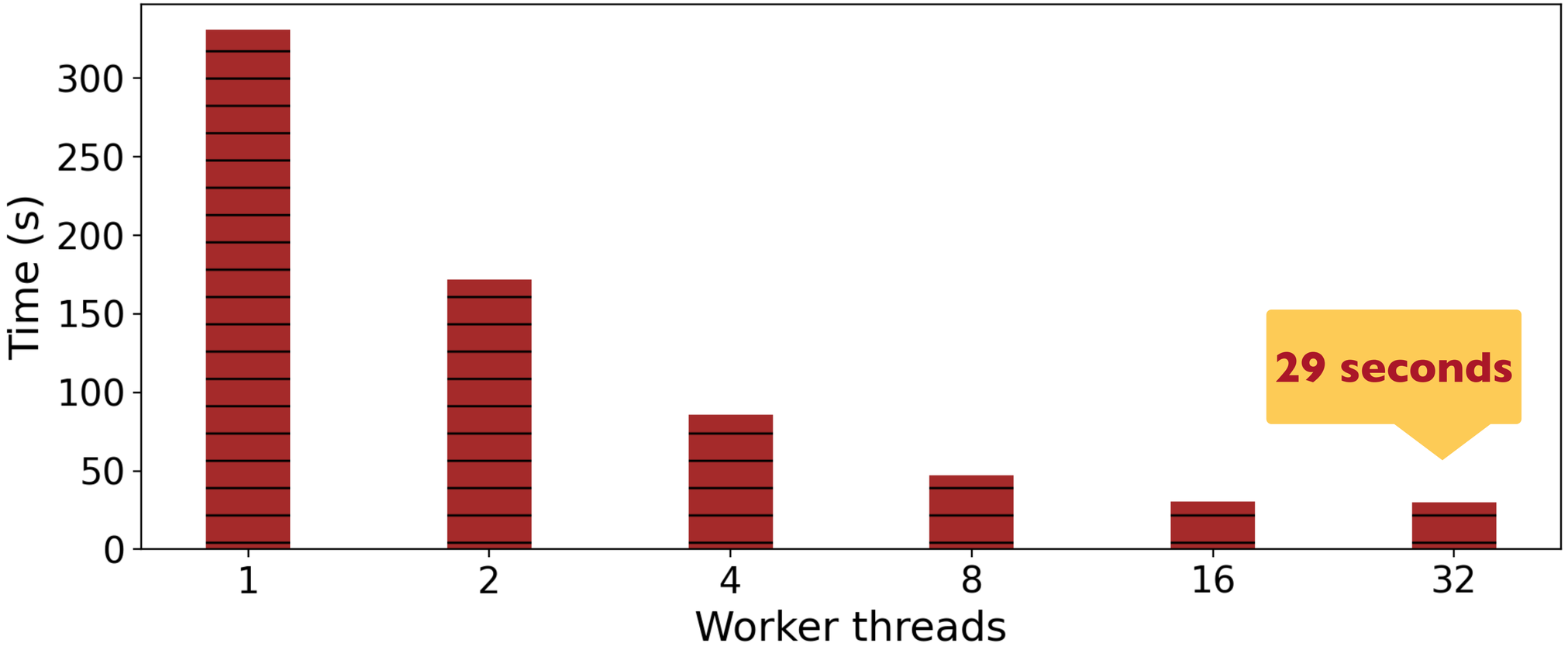
The optimization applies to all MPC queries used in existing systems



Secure query evaluation is reduced into a series of sorts



PARALLEL OBLIVIOUS SORT ON SECRECY (LAN)



* Reported times are for 2M input rows

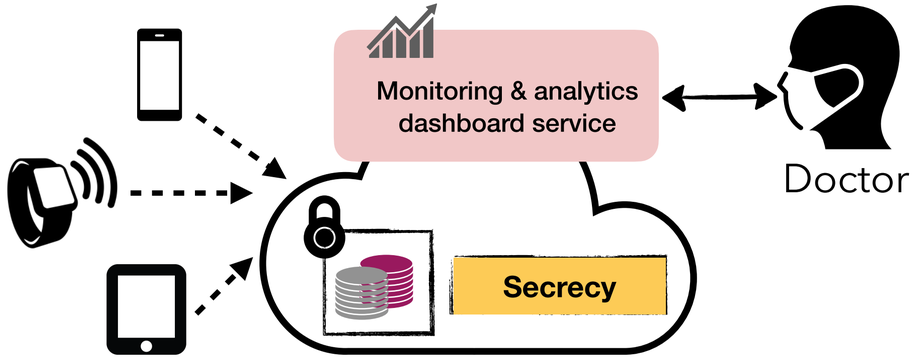
* Parties deployed on AWS EC2 r5.xlarge instances (us-east-2)

Ongoing and Future Work

REAL-WORLD SECRECY USE CASES

Digital Health Analytics

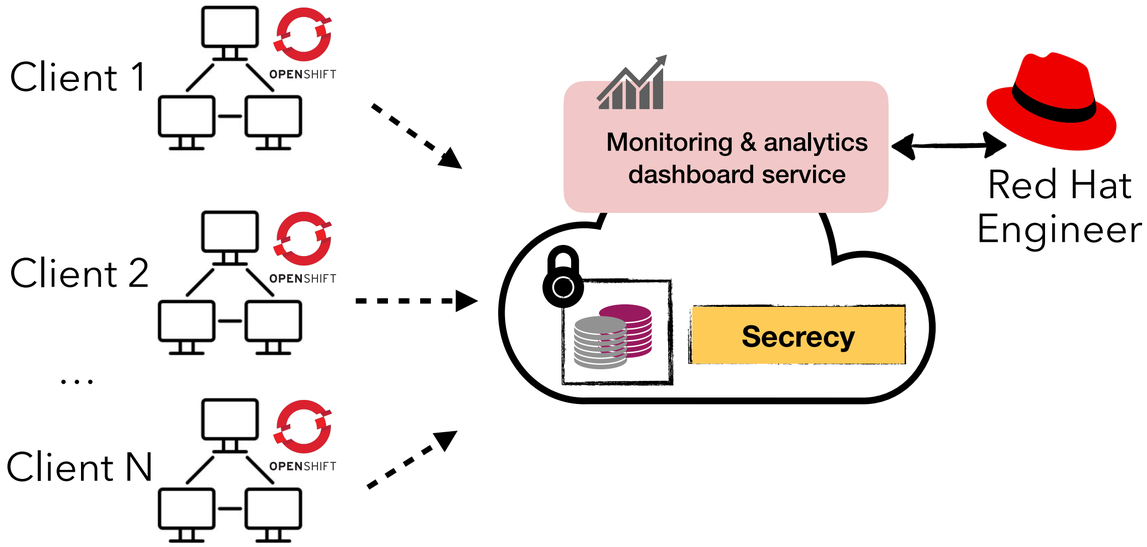
(BU Medical & Hariri Institute for Computing)



https://www.bu.edu/hic/research/focused-research-programs/continuous-analysis-of-mobile-health-data-among-medically-vulnerable-populations/

Secure cross-site analytics on OpenShift logs

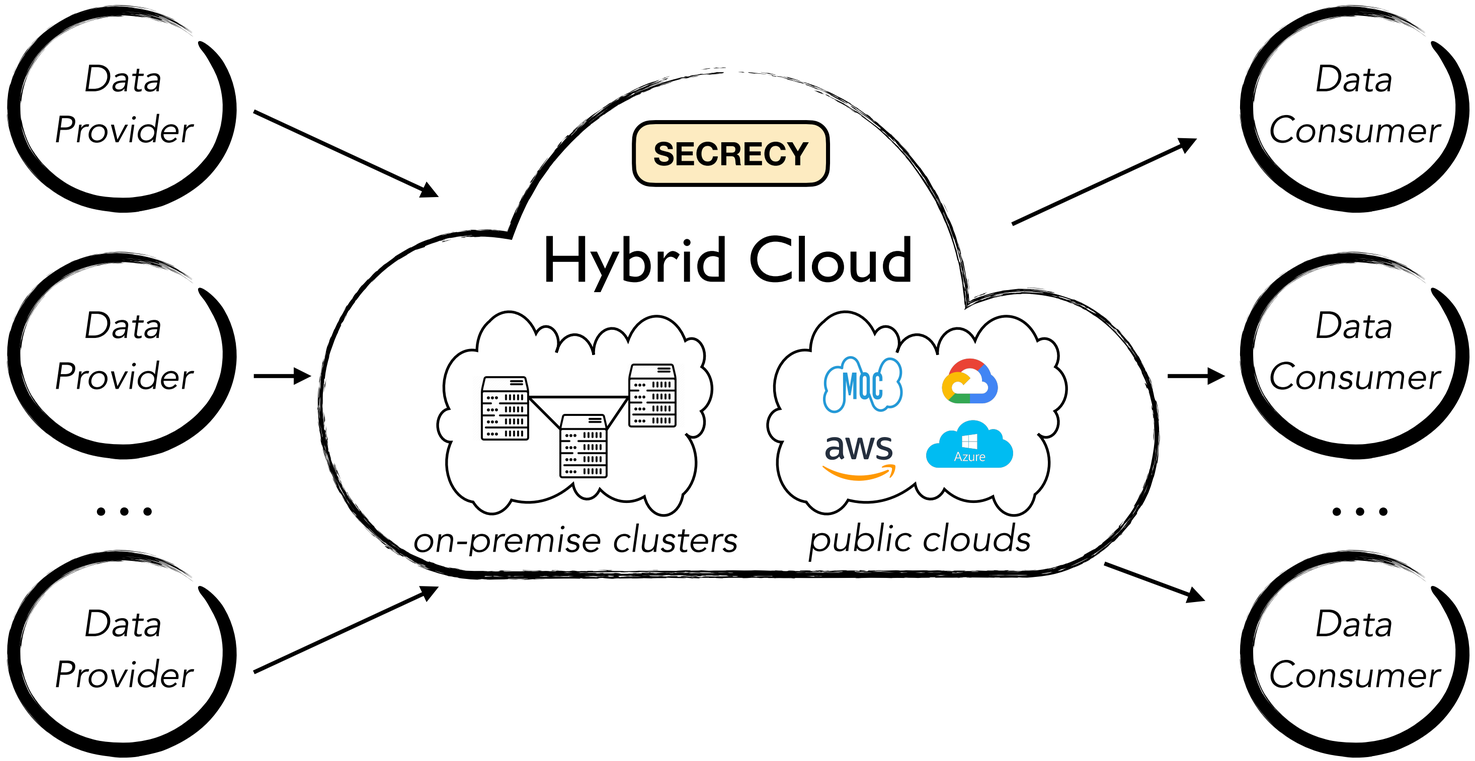
(BU RedHat Collaboratory)



https://www.bu.edu/rhcollab/projects/security-privacy/secure-cross-site-analytics-on-openshift-logs/

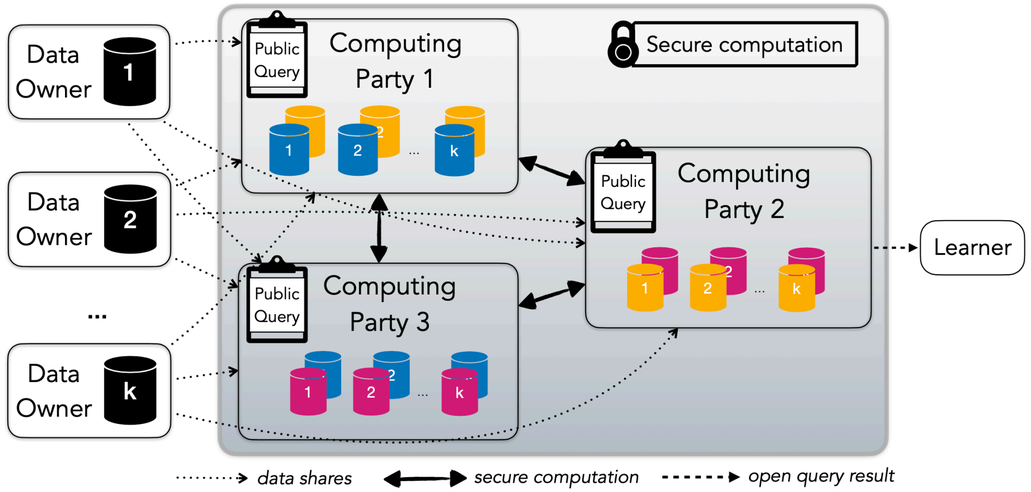
BROADER VISION

A general-purpose framework for private data analysis in untrusted clouds



+ Relational analytics
+ ML workloads
+ Fully Homomorphic Encryption primitives
+ Differential Privacy
+ Hardware acceleration for secure computation

SECRECY SUMMARY



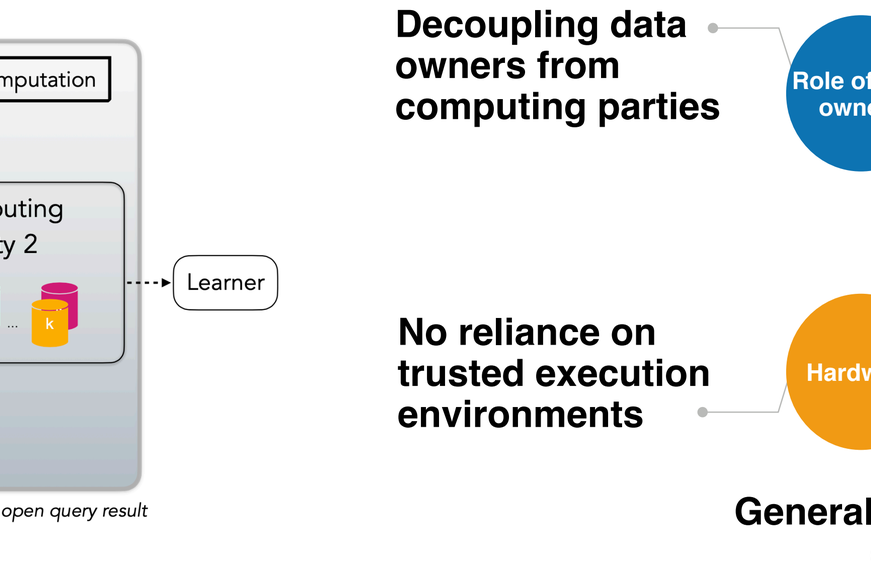
Decoupling data owners from computing parties

No information leakage

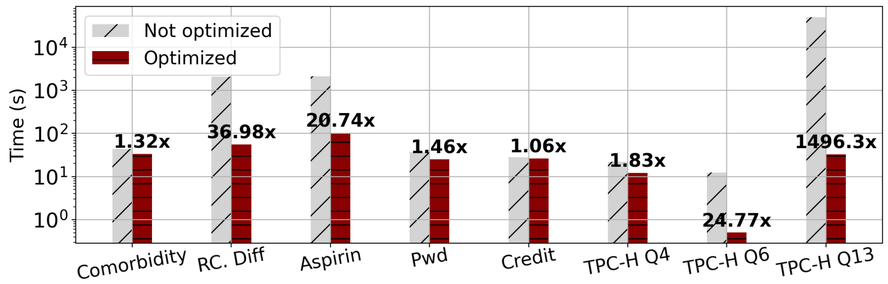
No reliance on trusted execution environments

End-to-end MPC execution

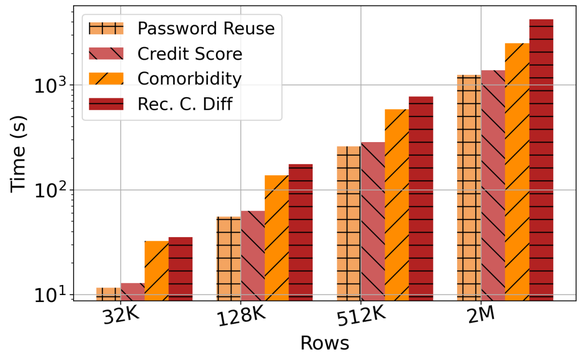
General and composable operators



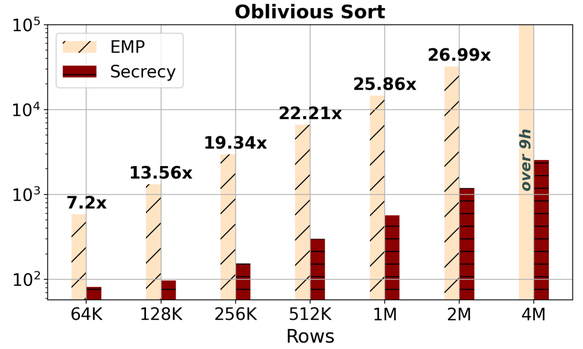
Up to 1000x speedups for real and synthetic queries



Millions of input rows entirely under MPC



Much larger inputs compared to prior works



J. Liagouris, V. Kalavri, M. Faisal, M. Varia. *Secrecy: Secure Collaborative Analytics on Secret-shared Data*. arXiv:2102.01048, 2021.