



# **SECRECY: SECURE COLLABORATIVE ANALYTICS ON SECRET-SHARED DATA**

John Liagouris, Vasiliki Kalavri, Muhammad Faisal, Mayank Varia

Boston University

# ABOUT US



John Liagouris



Research Scientist

*distributed systems  
databases*

Vasia Kalavri



Assistant Professor

*stream processing  
graph analytics*

Muhammad Faisal



PhD Student

*security  
distributed systems*

Mayank Varia



Research Professor

*theoretical & applied  
cryptography*

# SYSTEMS FOR SECURE COLLABORATIVE ANALYTICS

General-purpose  
MPC frameworks

Sharemind  
SCALE-MAMBA  
EMP  
ABY<sup>3</sup>  
OblivM  
Obliv-c  
JIFF  
Wysteria  
MP-SPDZ  
...

*Limited support for  
relational operators*

Enclave-based  
query processors

Opaque  
StealthDB  
ObliDB  
OCQ  
...

*Rely on trusted hardware  
(e.g. Intel's SGX)*

MPC query  
processors

Conclave  
Shrinkwrap  
SMCQL  
Senate  
SAQE  
...

*Minimize the secure part of  
the computation or relax the  
security guarantees (or both)*



# SYSTEMS FOR RELATIONAL MPC

Some systems relax the full MPC security guarantees to speed up queries

Hybrid execution requires that data owners participate in the computation

Framework	Information Leakage	Trusted Party	Query Execution	Optimization Objective	Optimization Conditions
Conclave	Controlled (Hybrid operators)	Yes	Hybrid	Minimize the use of secure computation	1. Data owners participate in computation 2. Data owners provide privacy annotations 3. There exists a (fourth) trusted party
SMCOT	No	No	Hybrid	Minimize the use of secure computation	1. Data owners participate in computation 2. Data owners provide privacy annotations
Senate	No	No	Hybrid	Calibrate padding of intermediate results	1. Data owners participate in computation 2. Data owners provide privacy annotations and intermediate result sensitivities
				Choose sampling rate for approximate answers	1. Data owners participate in computation 2. Data owners provide privacy annotations and privacy budget
Senate	No	No	Hybrid	Reduce joint computation to subsets of parties	1. Data owners participate in computation 2. Input or intermediate relations are owned by subsets of the computing parties
SDB	Yes (operator dependent)	No	Hybrid	Reduce data encryption and decryption costs	1. Data owner participates in computation 2. Data owner provides privacy annotations
Secrecy	No	No	End-to-end under MPC	Reduce MPC costs	None

Senate further optimizes plans *inside* MPC without information leakage by leveraging information about data ownership

Various optimizations are applicable under certain conditions (e.g. data are annotated as sensitive or non-sensitive)

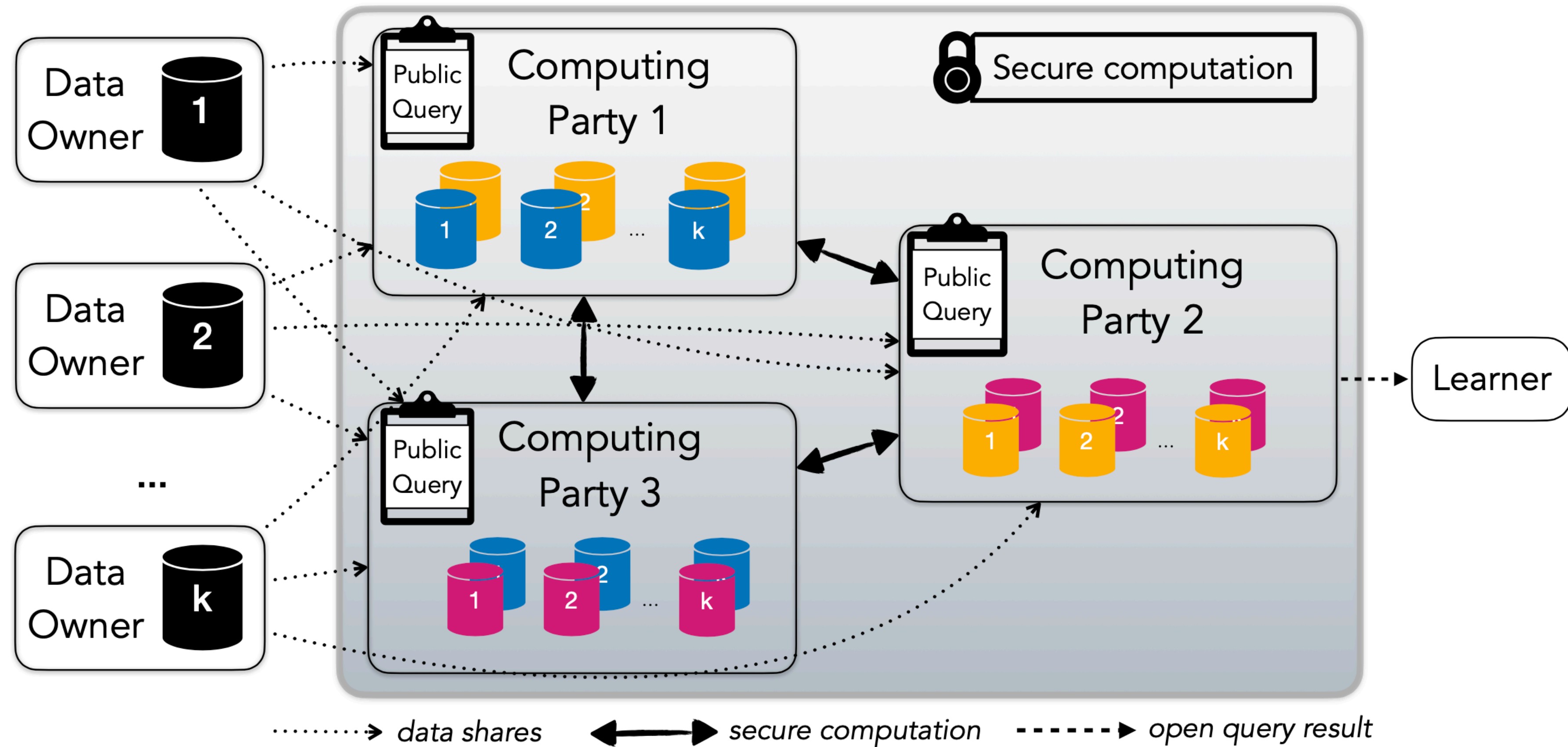
---

# OPTIMIZING RELATIONAL QUERIES UNDER MPC

*Can we improve MPC query performance when **all** data are sensitive and data owners **may not** have the private resources to participate in the computation?*

*...and do so with no information leakage and without relying on trusted execution environments?*

# MPC SETTING OVERVIEW



Each party receives two shares of each input dataset  
(replicated secret-sharing<sup>1</sup>)

<sup>1</sup> 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.

---

# THREAT MODEL AND GUARANTEES

## Semi-honest model

- Parties do not deviate from the protocol (“honest but curious”)
- Adversary has complete control over the network and can also compromise one computing party (but cannot alter its execution)

## Security guarantees

- Untrusted parties do not learn anything about:
  - The actual data (input, output, intermediate) and their sizes
  - The data access patterns during query execution



# SECRECY DESIGN PRINCIPLES

## Decoupling data owners from computing parties

- No assumptions about data ownership
- All optimizations are applicable even if none of the data owners participates in the computation

## No reliance on trusted execution environments

- Do not rely on hardware enclaves, honest brokers or secure co-processors
- Do not rely on (semi-)trusted parties



## No information leakage

- Hide all access patterns and the size of intermediate data
- All optimizations retain the full MPC security guarantees

## End-to-end MPC execution

- Execute all operators under MPC
- Do not require data owners to annotate attributes as sensitive or not sensitive

## General and composable operators

- No assumptions about the data schema
- Operators can be combined to construct arbitrary end-to-end oblivious queries



---

# MPC COSTS

---

# MPC COSTS

## Operation cost

- The number of MPC operations required
- An operation includes **local computation** plus a number of **messages** exchanged among computing parties

---

# MPC COSTS

## Operation cost

- The number of MPC operations required
- An operation includes **local computation** plus a number of **messages** exchanged among computing parties

## Synchronization cost

- The number of communication rounds among computing parties
- Each communication round is a **barrier** in the distributed execution



---

# MPC COSTS

## Operation cost

- The number of MPC operations required
- An operation includes **local computation** plus a number of **messages** exchanged among computing parties

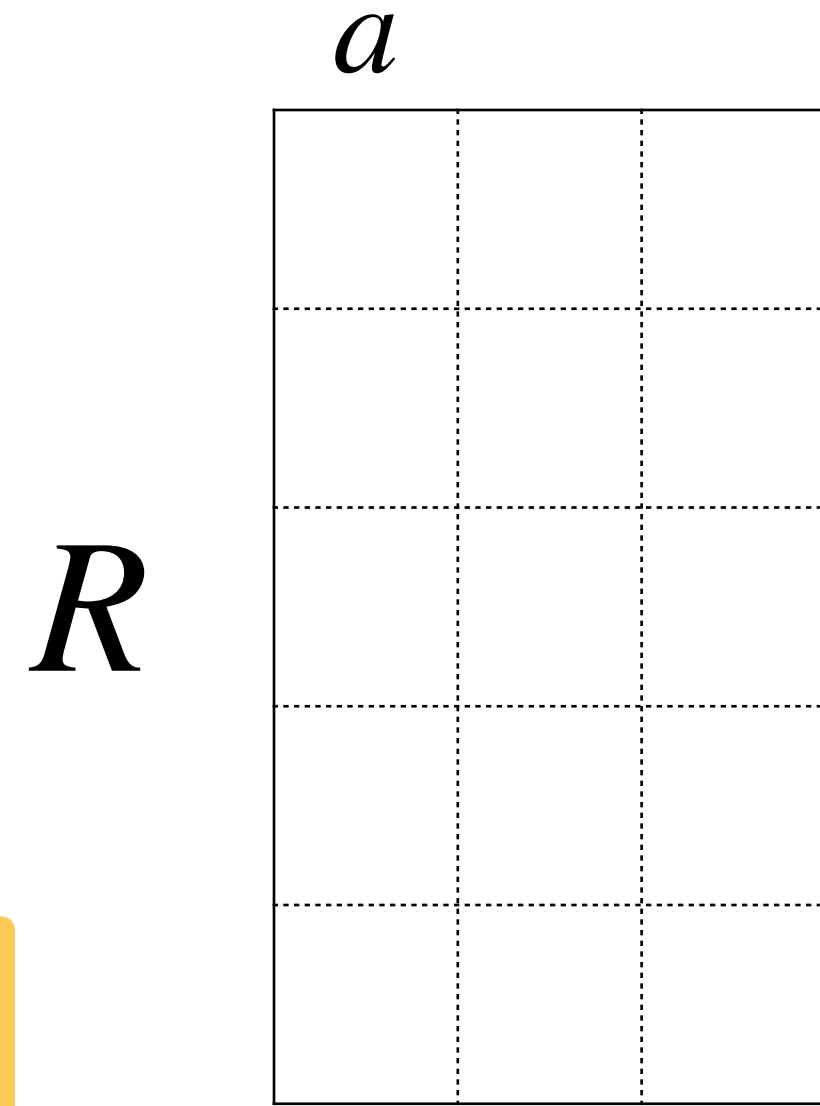
## Synchronization cost

- The number of communication rounds among computing parties
- Each communication round is a **barrier** in the distributed execution

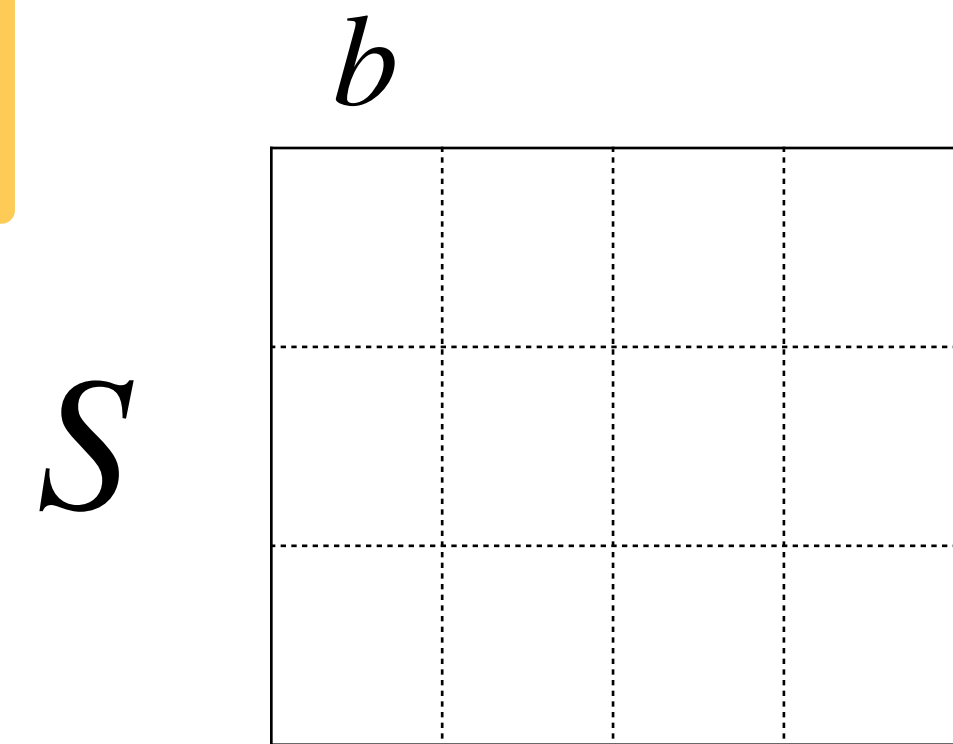
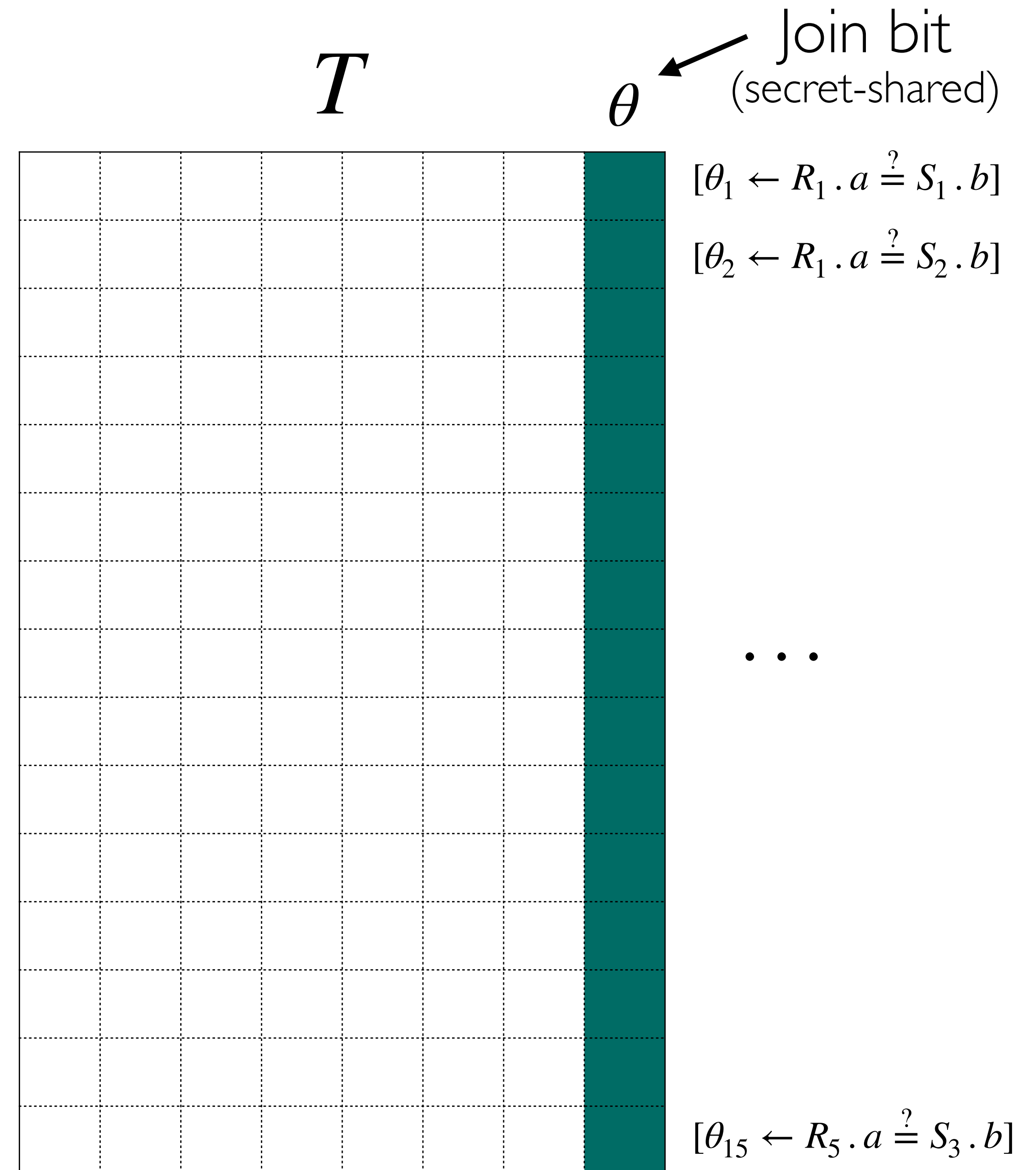
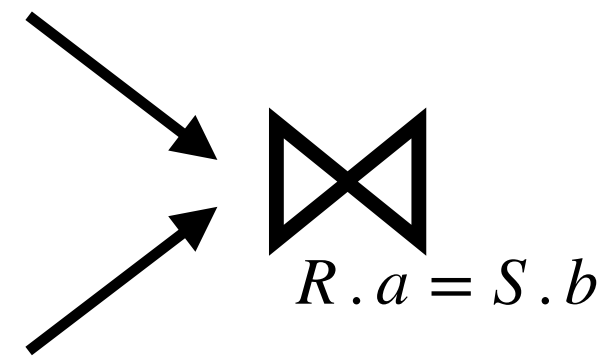
## Composition cost

- The **extra cost** of composing oblivious operators under MPC
- Measured in number of MPC operations and communication rounds

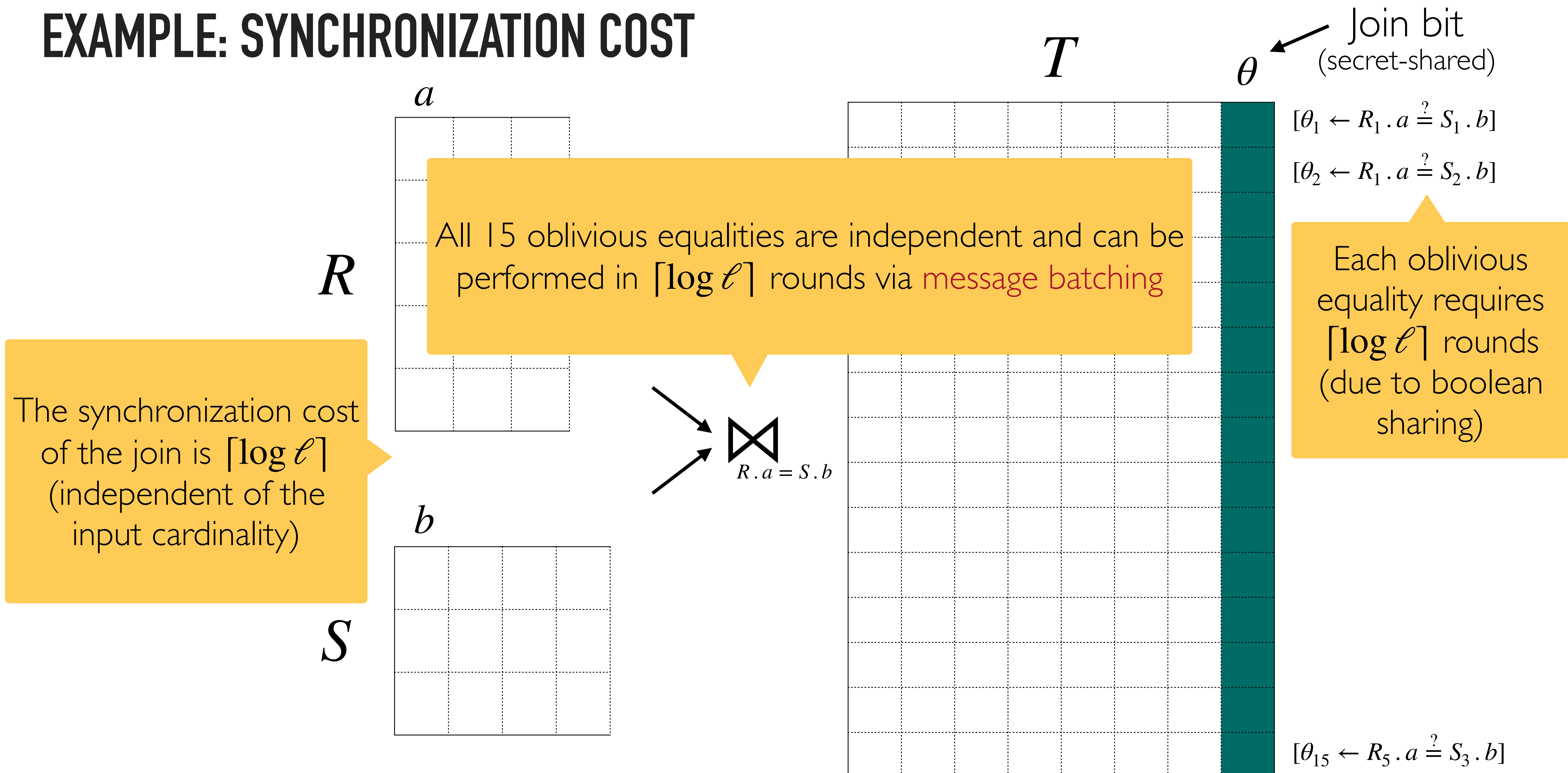
## EXAMPLE: OPERATION COST



The operation cost is  $|R| \times |S| = 15$  oblivious equalities

 $|R|, |S|$ : cardinalities of the input relations

# EXAMPLE: SYNCHRONIZATION COST

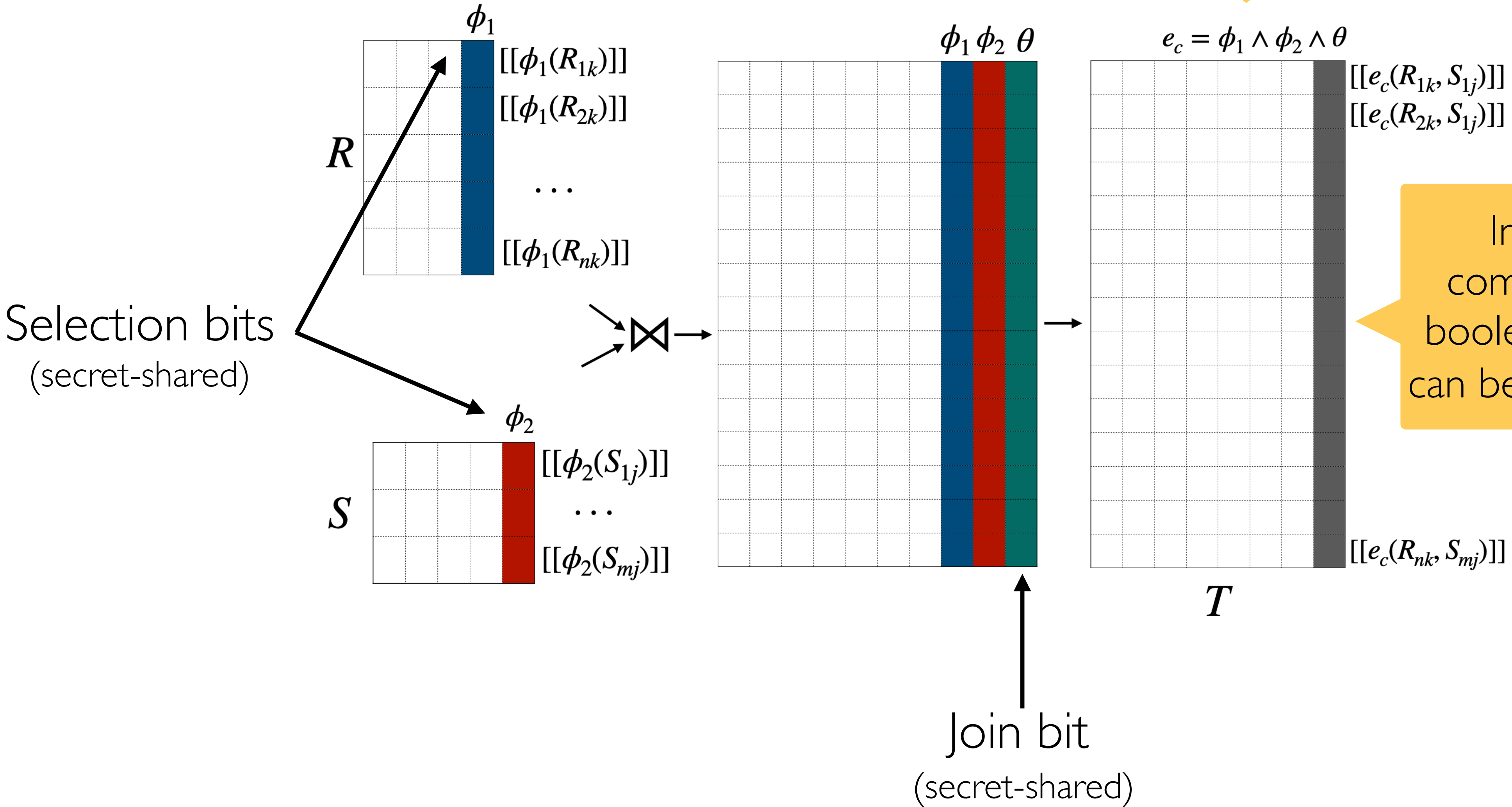


$\ell$ : the length of attributes  $a$  and  $b$  in number of bits



# EXAMPLE: COMPOSITION COST

The extra cost of composition is the cost of evaluating  $e_c$  under MPC



In this example, the composition requires 30 boolean ANDs all of which can be evaluated in 2 rounds

---

# MPC Query Optimizations

---

# OPTIMIZATIONS RATIONALE

Secrecy aims to reduce **one or more** of the three MPC costs:

1. Operation cost
2. Synchronization cost
3. Composition cost



---

# OPTIMIZATIONS RATIONALE

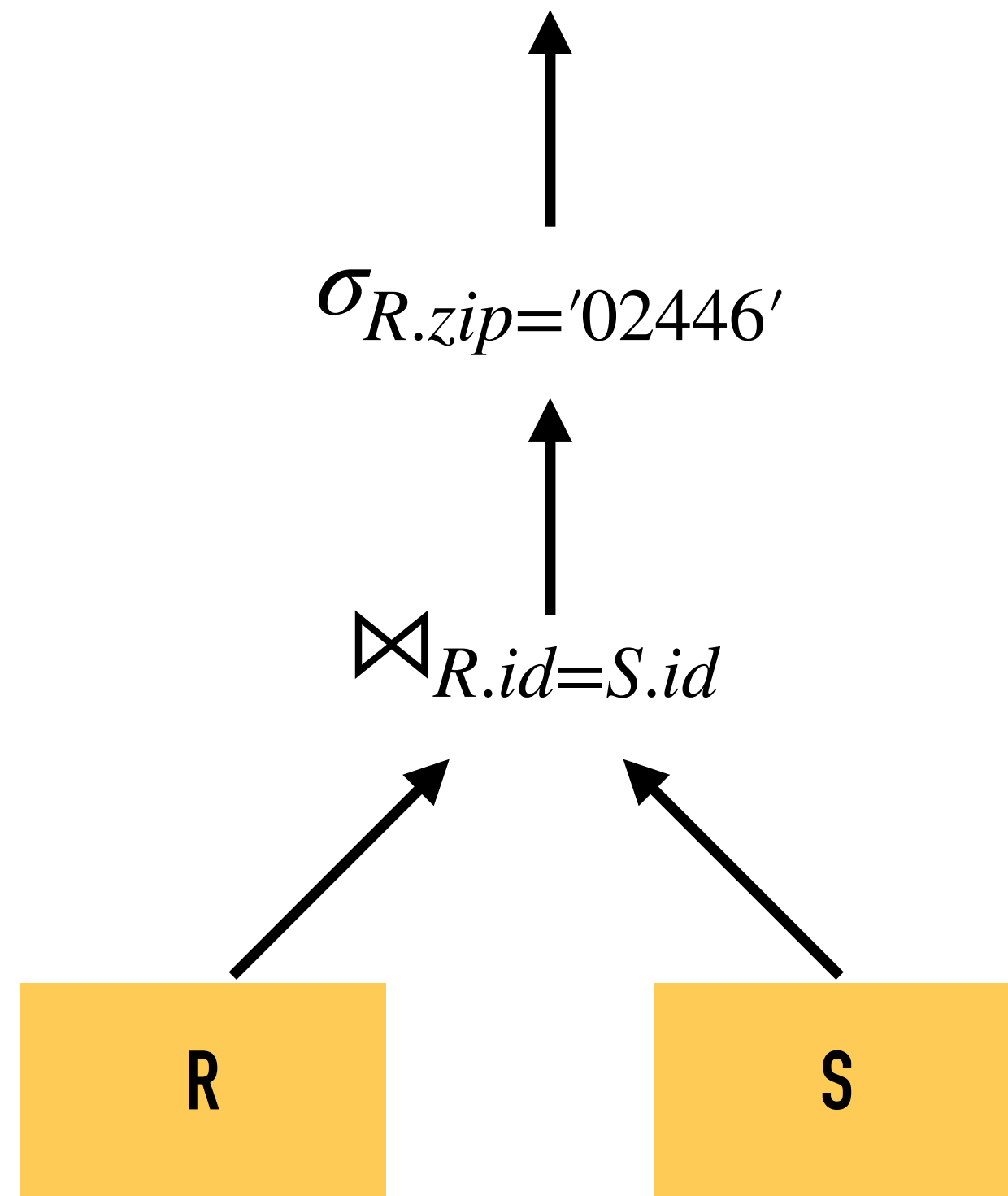
Secrecy aims to reduce **one or more** of the three MPC costs:

1. Operation cost
2. Synchronization cost
3. Composition cost

We introduce three different types of optimizations:

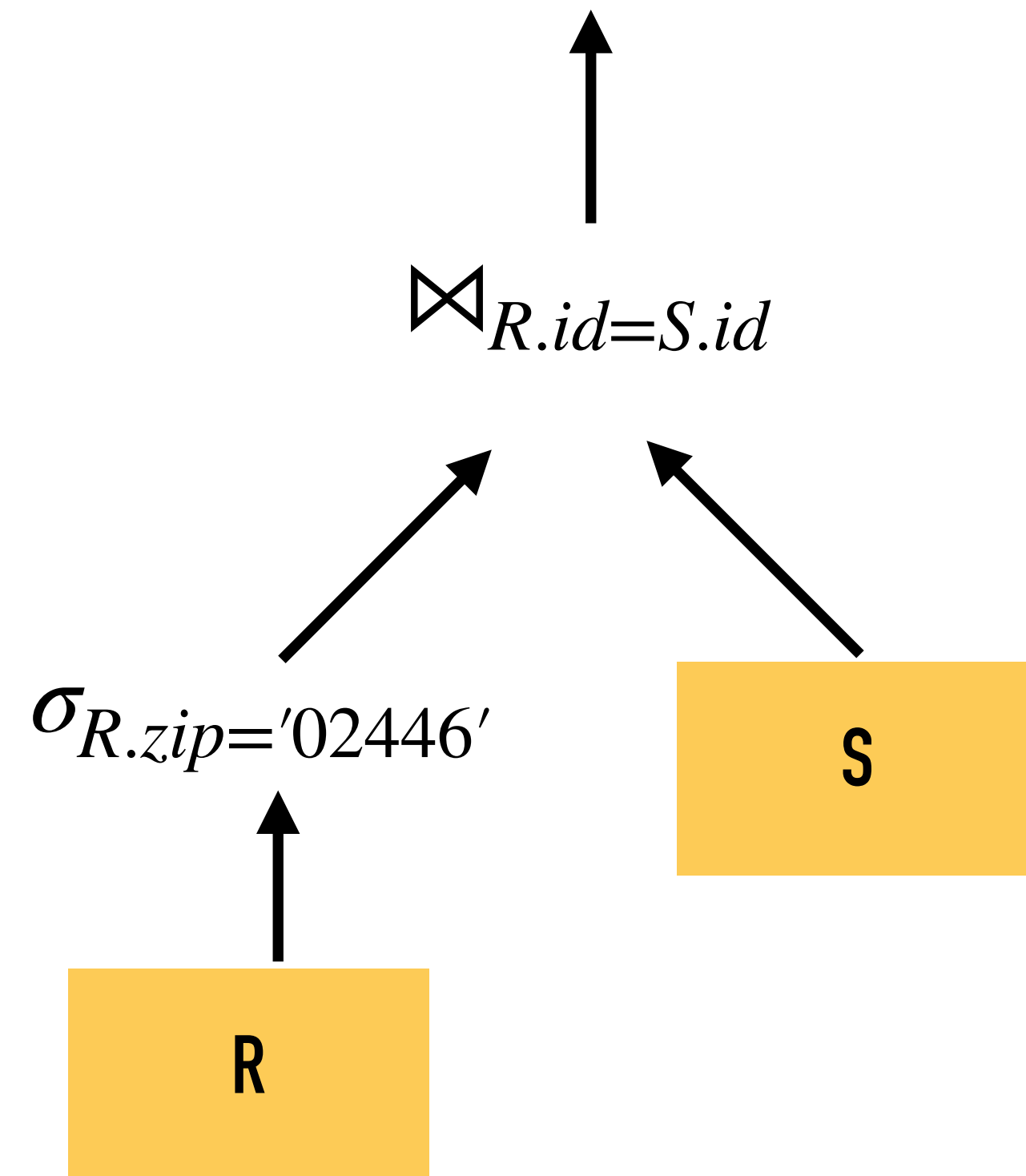
1. Logical transformation rules
2. Physical optimizations
3. Secret-sharing optimizations

# LOGICAL TRANSFORMATION RULES



```
SELECT R.id
FROM R, S
WHERE R.id = S.id
AND R.zip='02446'
```

# LOGICAL TRANSFORMATION RULES

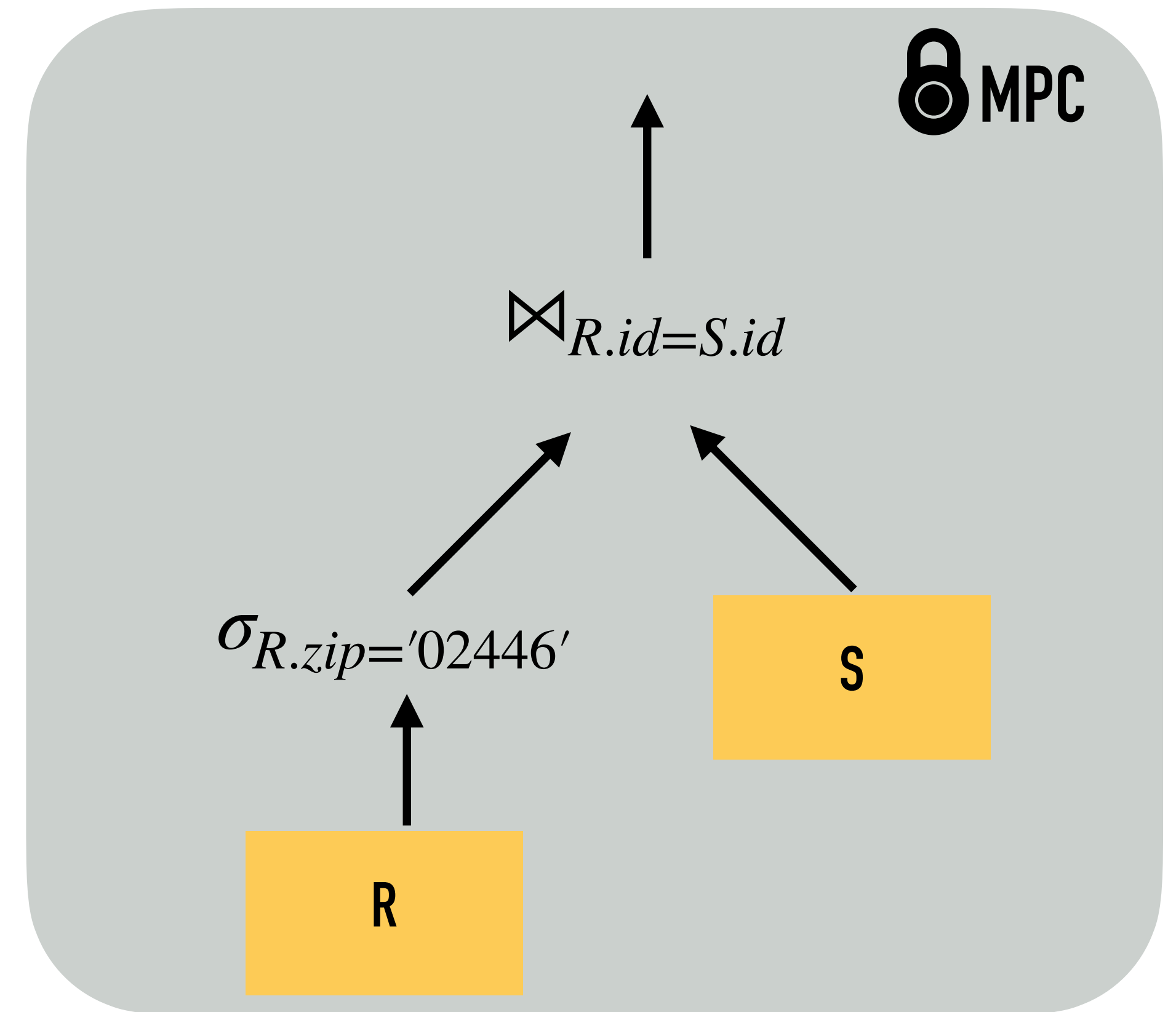
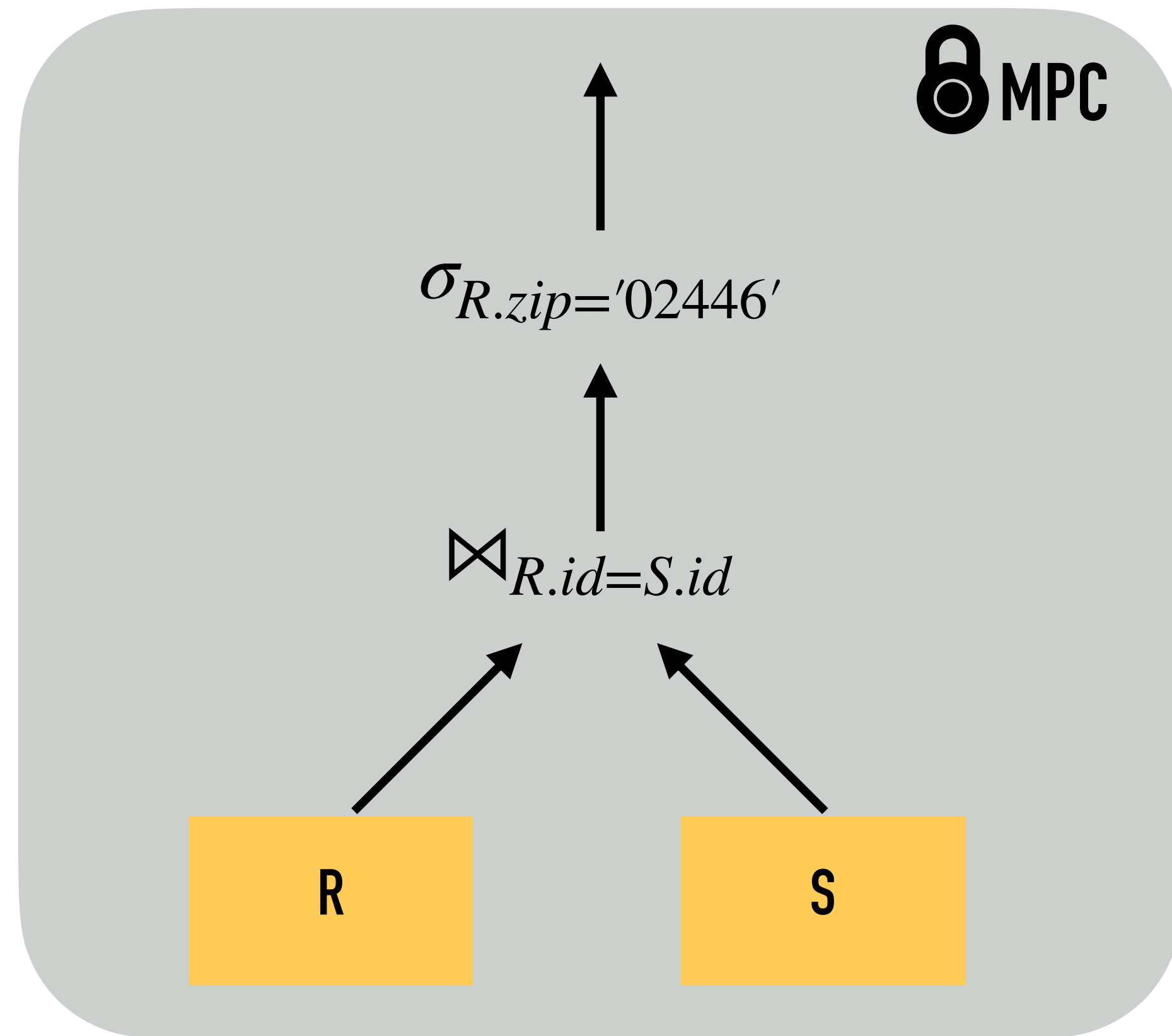


```
SELECT R.id
FROM R, S
WHERE R.id = S.id
AND R.zip = '02446'
```

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

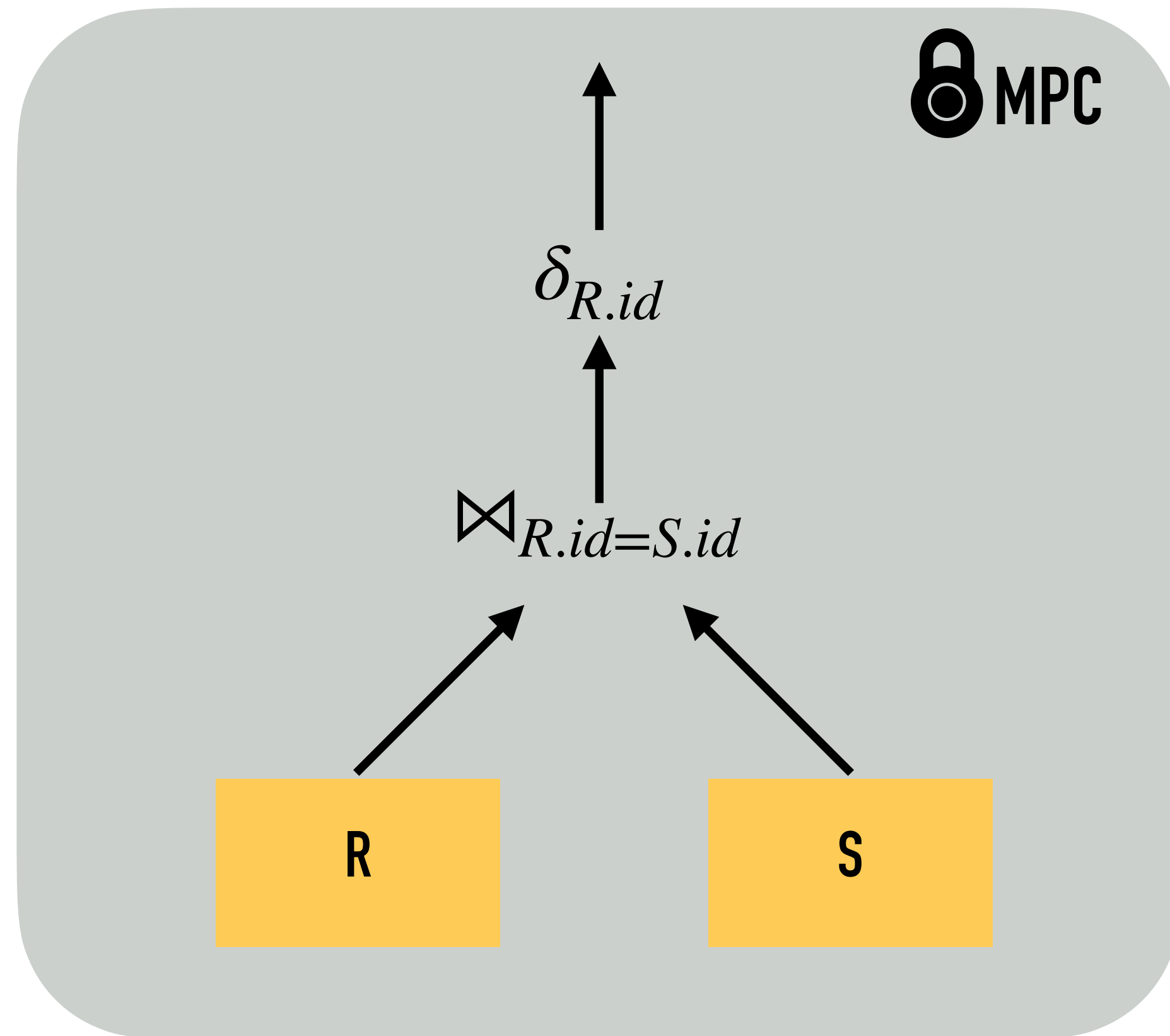


# PLAINTEXT OPTIMIZATIONS ARE NOT ALWAYS EFFECTIVE UNDER MPC



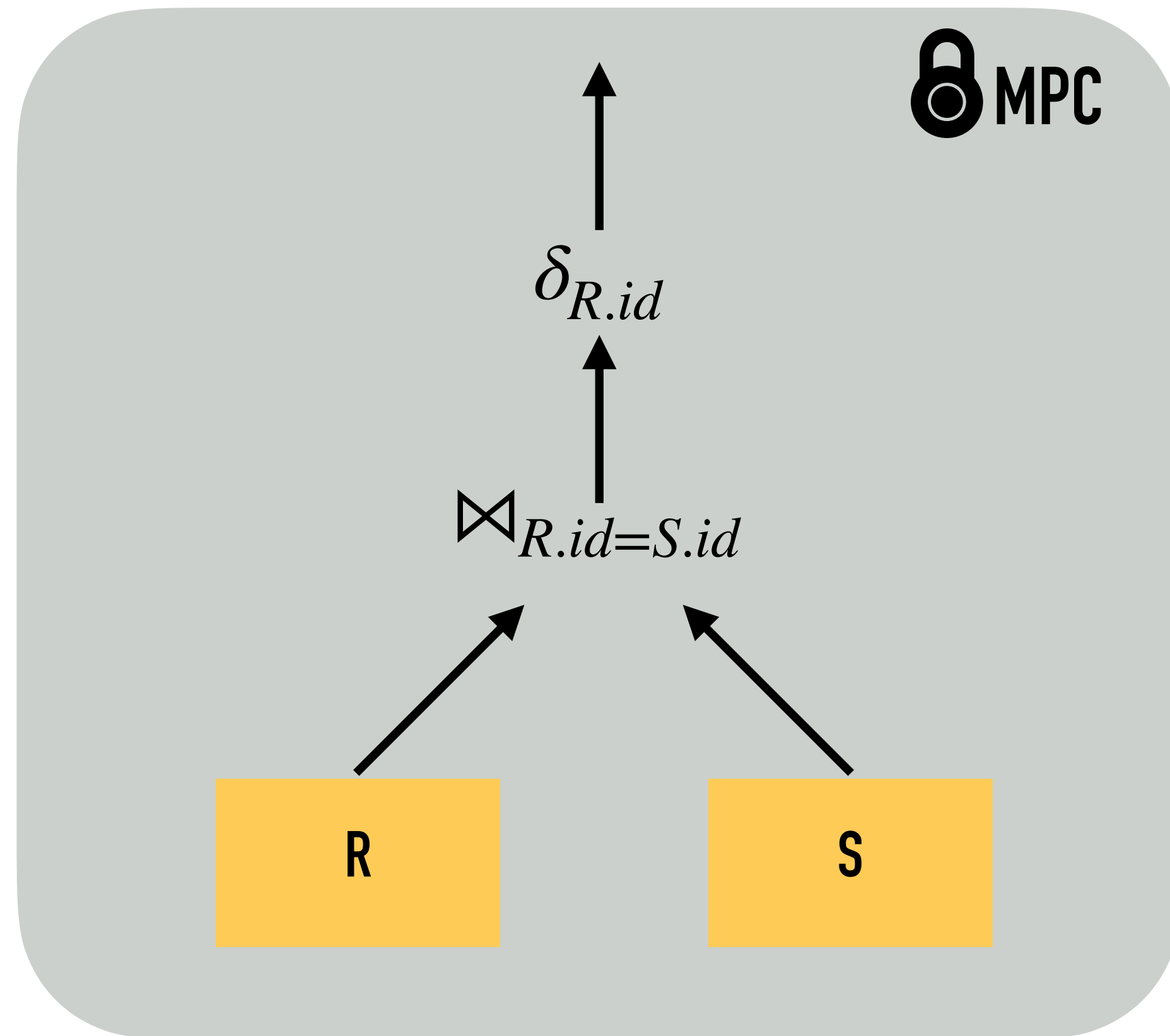
*Pushing the selection before the JOIN does not improve JOIN's performance under MPC*  
(since the oblivious selection does not remove any tuples from R)

# OPERATOR REORDERING STILL MAKES SENSE UNDER MPC



```
SELECT DISTINCT R.id
FROM R, S
WHERE R.id = S.id
```

# OPERATOR REORDERING STILL MAKES SENSE UNDER MPC



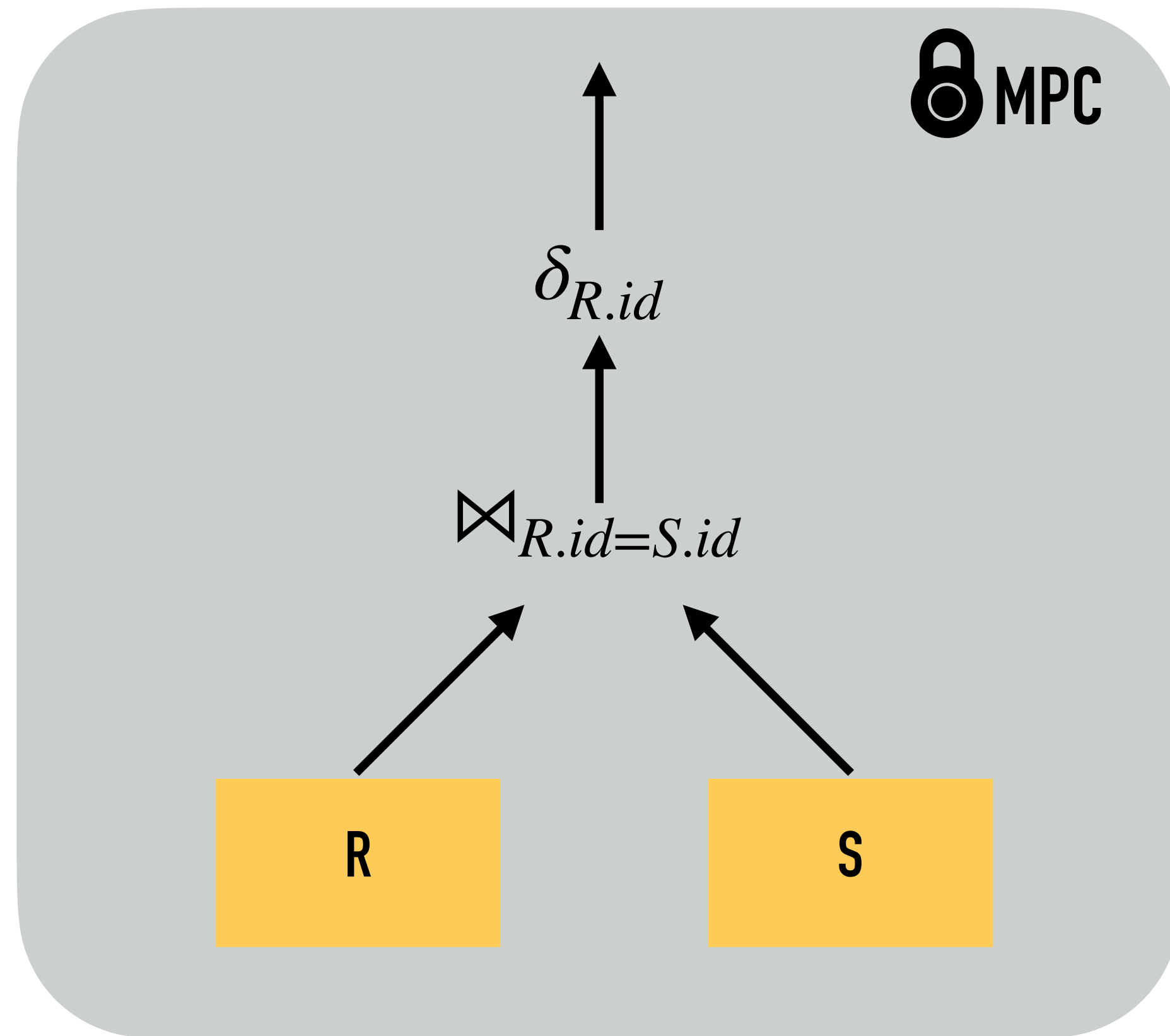
```
SELECT DISTINCT R.id  
FROM R, S  
WHERE R.id = S.id
```

$$|R| = |S| = n$$

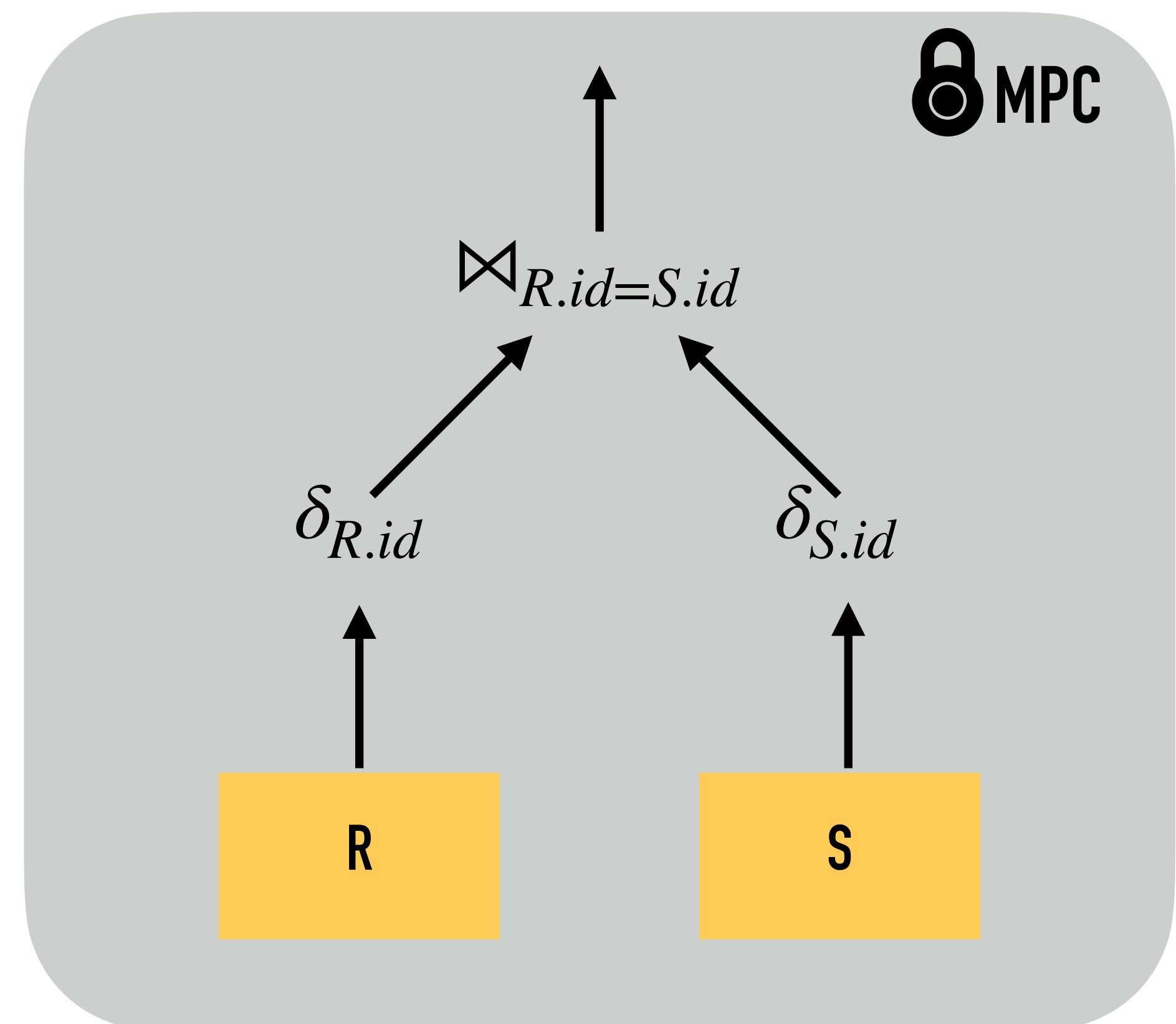
$O(n^2 \log^2 n)$  operations / messages

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

# OPERATOR REORDERING STILL MAKES SENSE UNDER MPC



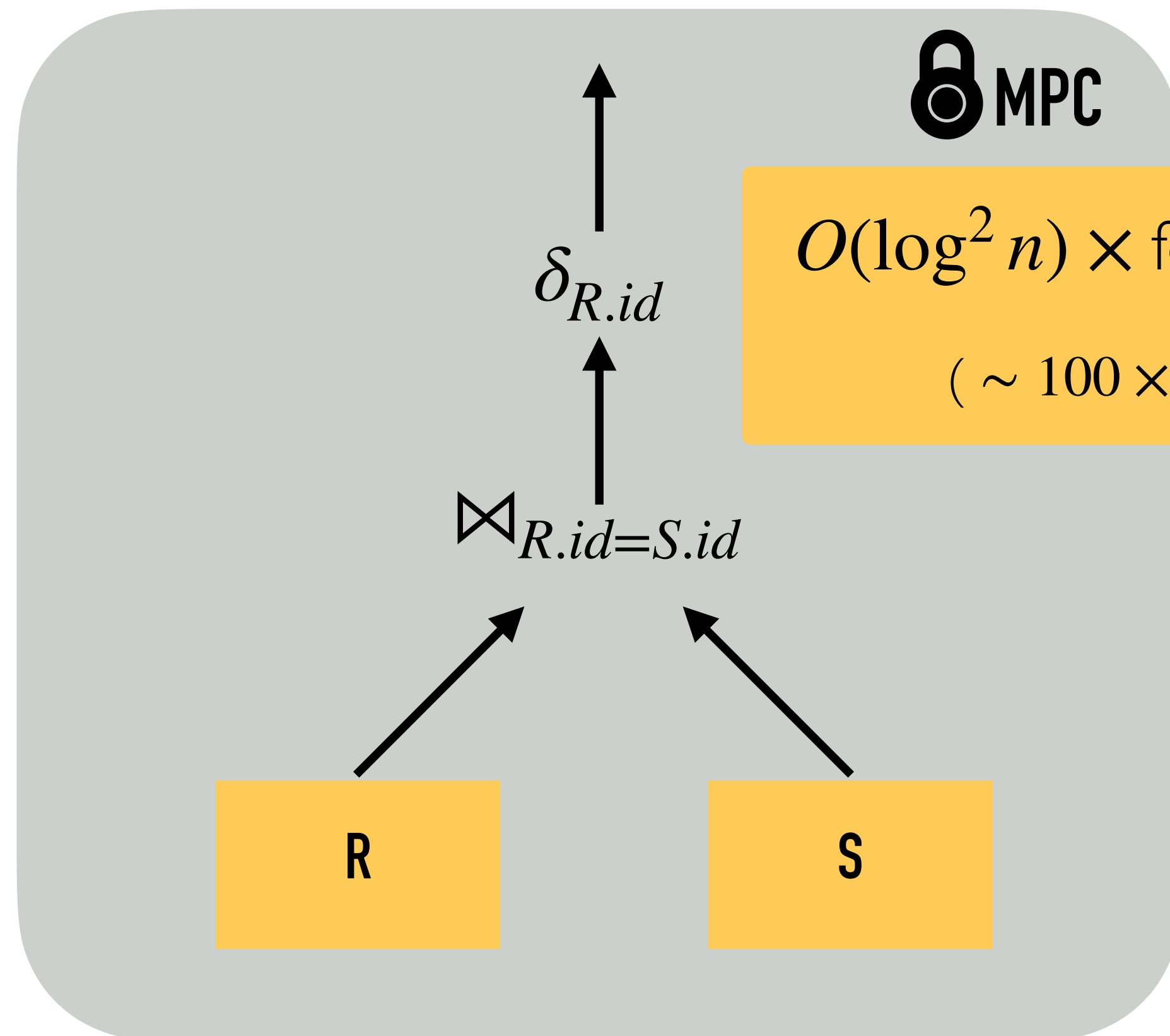
$O(n^2 \log^2 n)$  operations / messages  
 $O(\log^2 n)$  rounds       $O(n^2)$  space



$O(n^2)$  operations / messages  
 $\sim 4 \times$  fewer rounds       $O(n)$  space

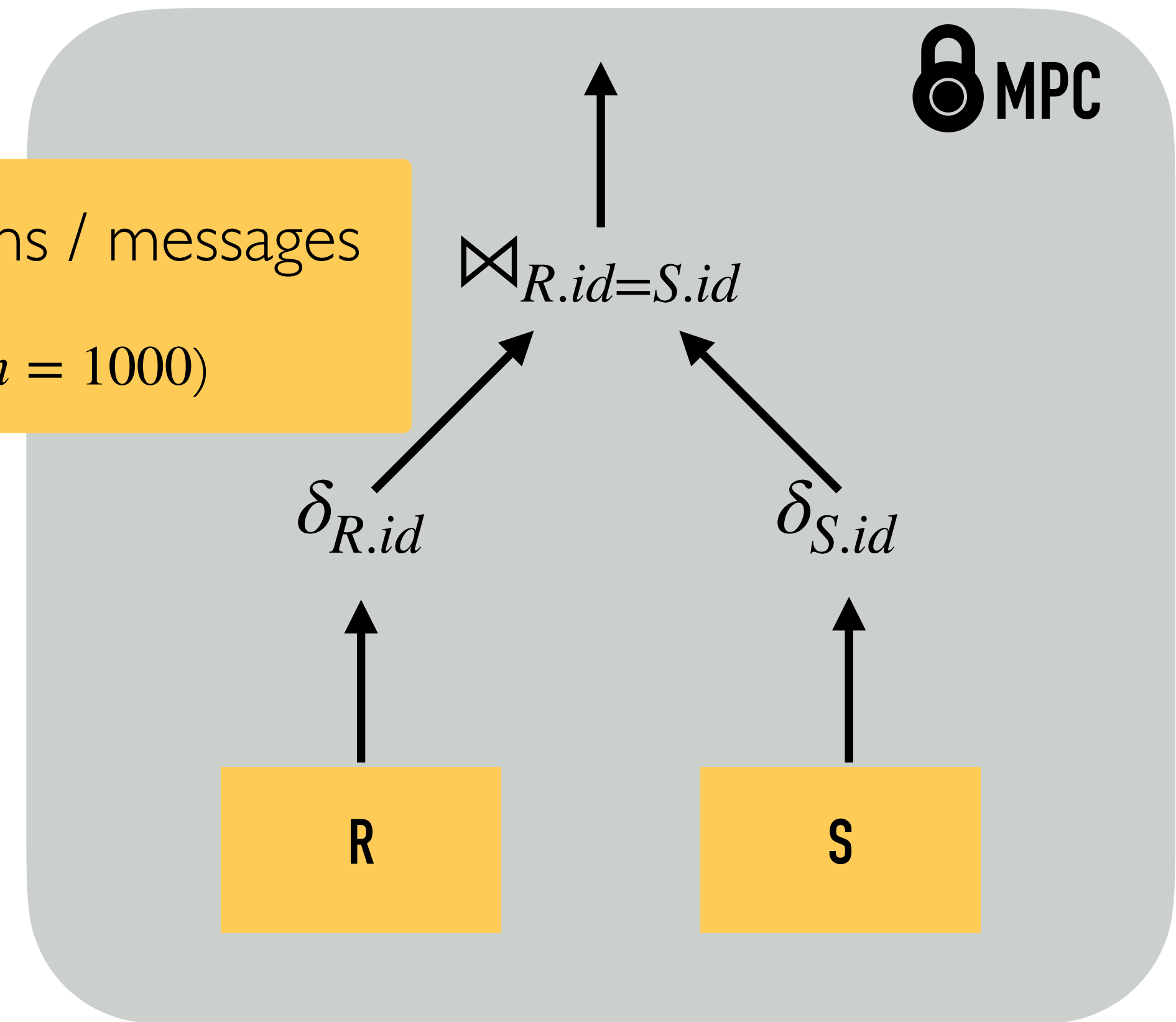


# OPERATOR REORDERING STILL MAKES SENSE UNDER MPC



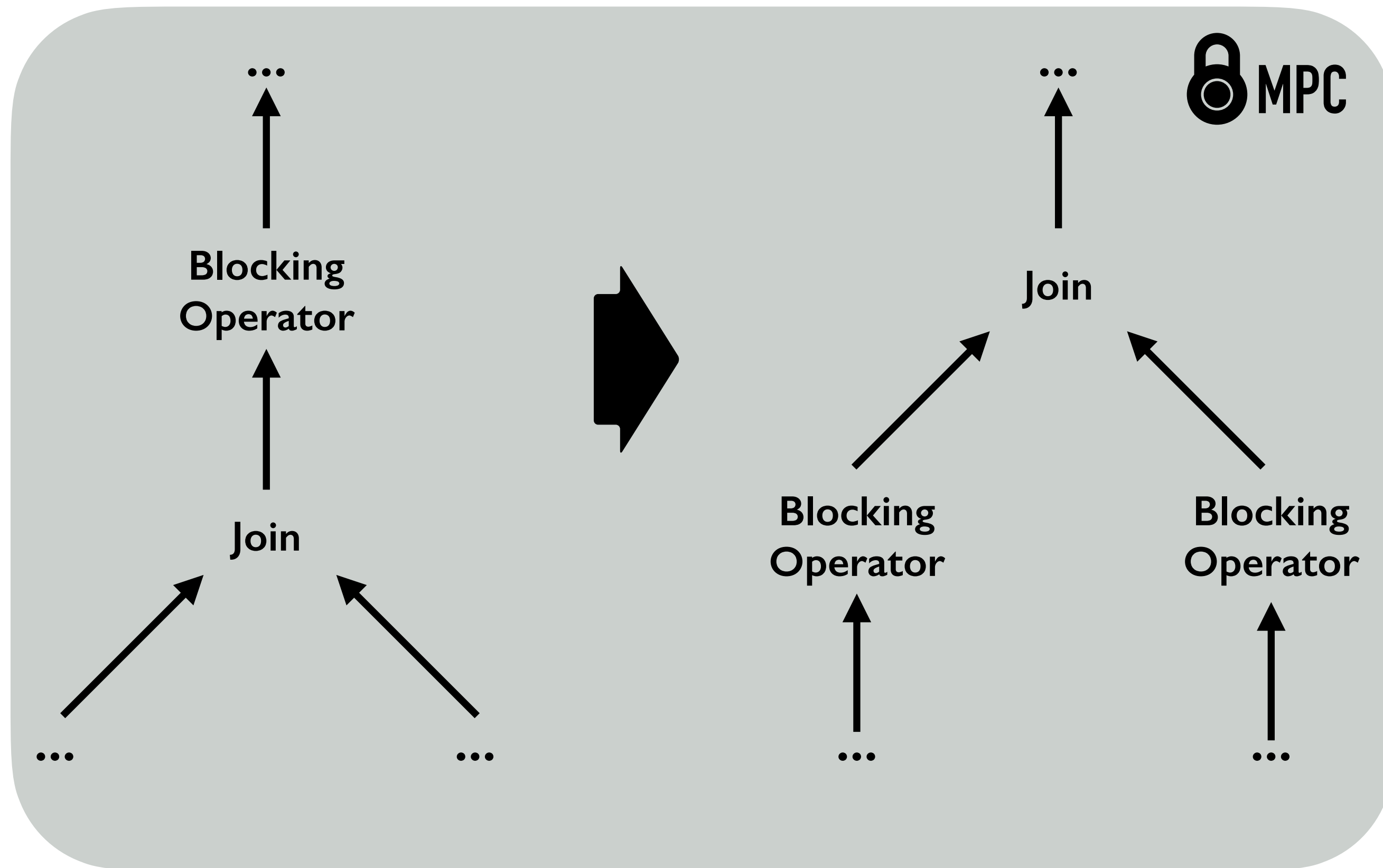
$O(n^2 \log^2 n)$  operations / messages  
 $O(\log^2 n)$  rounds     $O(n^2)$  space

$O(\log^2 n) \times$  fewer operations / messages  
(  $\sim 100 \times$  improvement for  $n = 1000$  )



$O(n^2)$  operations / messages  
 $\sim 4 \times$  fewer rounds     $O(n)$  space

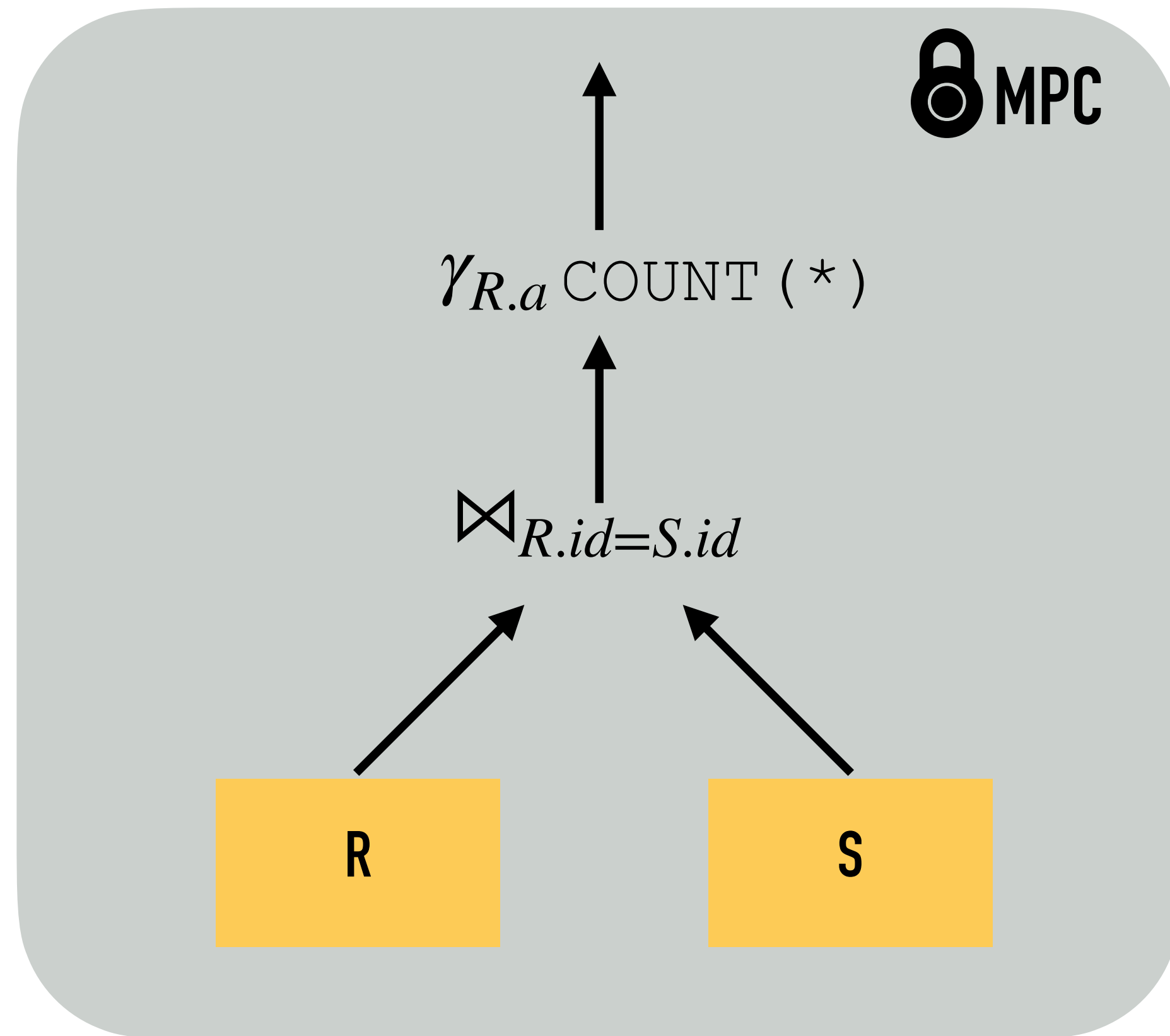
# LOGICAL TRANSFORMATION RULES IN SECRECY



- *Push blocking operators down to the input*
  - \* Blocking operators materialize and sort their input — the **earlier** we apply them the better
- *Push joins up to the root*
  - \* Joins produce outputs larger than their inputs — the **later** we apply them the better

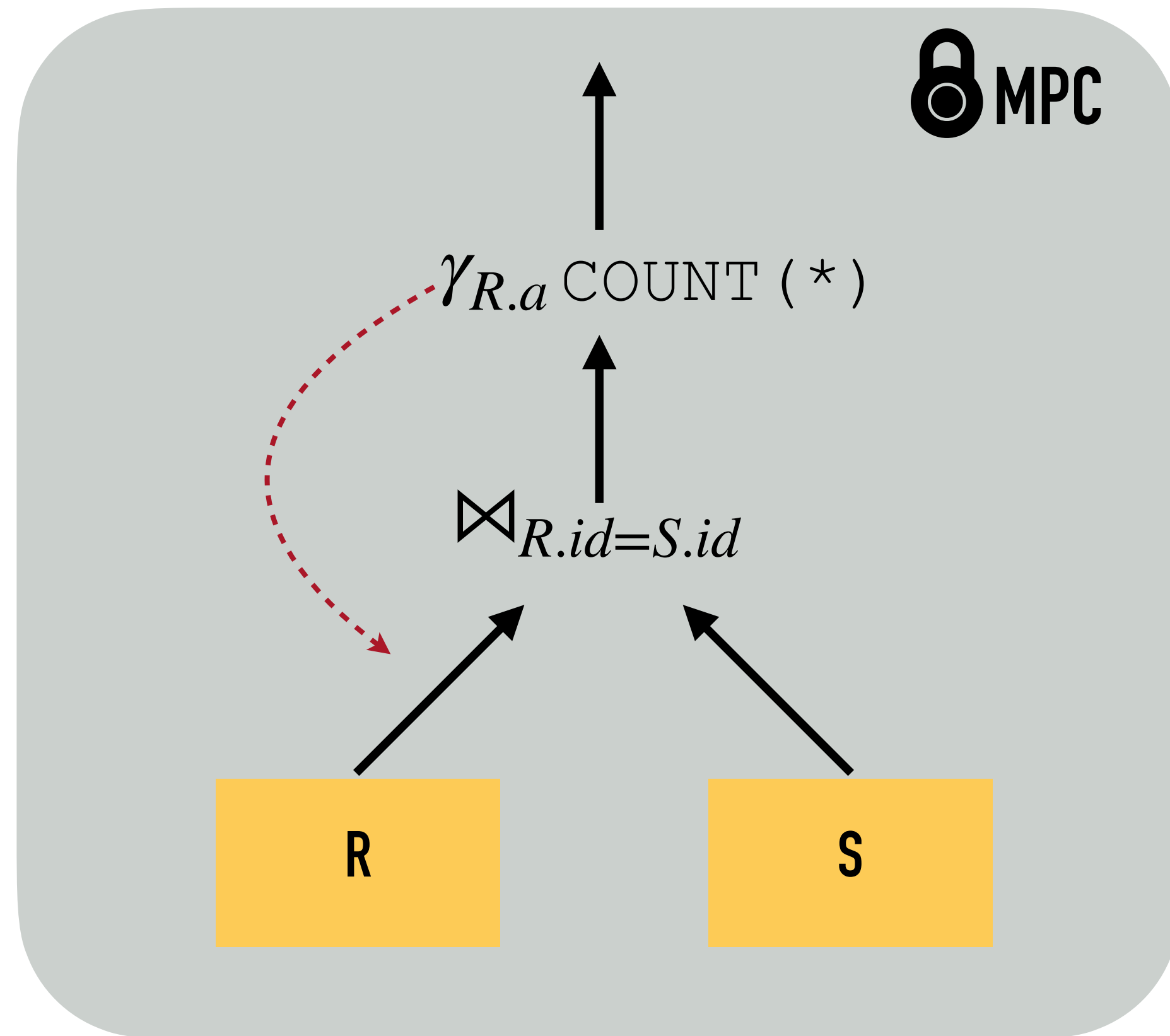
*Blocking operators: GROUP-BY, ORDER-BY, DISTINCT  
(all based on oblivious sorting)*

# EXAMPLE: OPERATOR DECOMPOSITION



```
SELECT R.a, COUNT (*)  
FROM R, S  
WHERE R.id = S.id  
GROUP-BY R.a
```

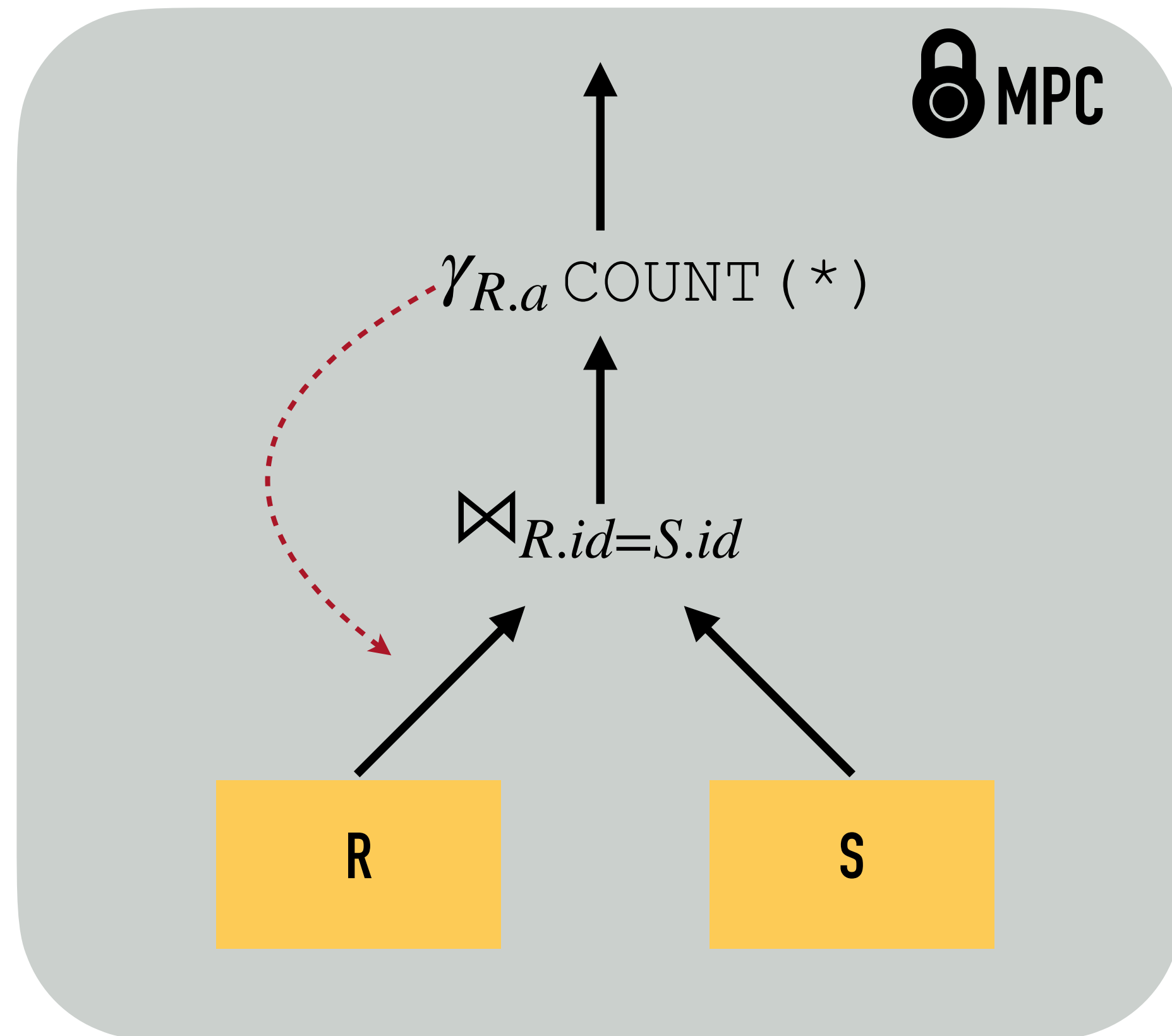
# EXAMPLE: OPERATOR DECOMPOSITION



```
SELECT R.a, COUNT (*)  
FROM R, S  
WHERE R.id = S.id  
GROUP-BY R.a
```

*Pushing GROUP-BY down does not  
produce a semantically equivalent plan*

# EXAMPLE: OPERATOR DECOMPOSITION

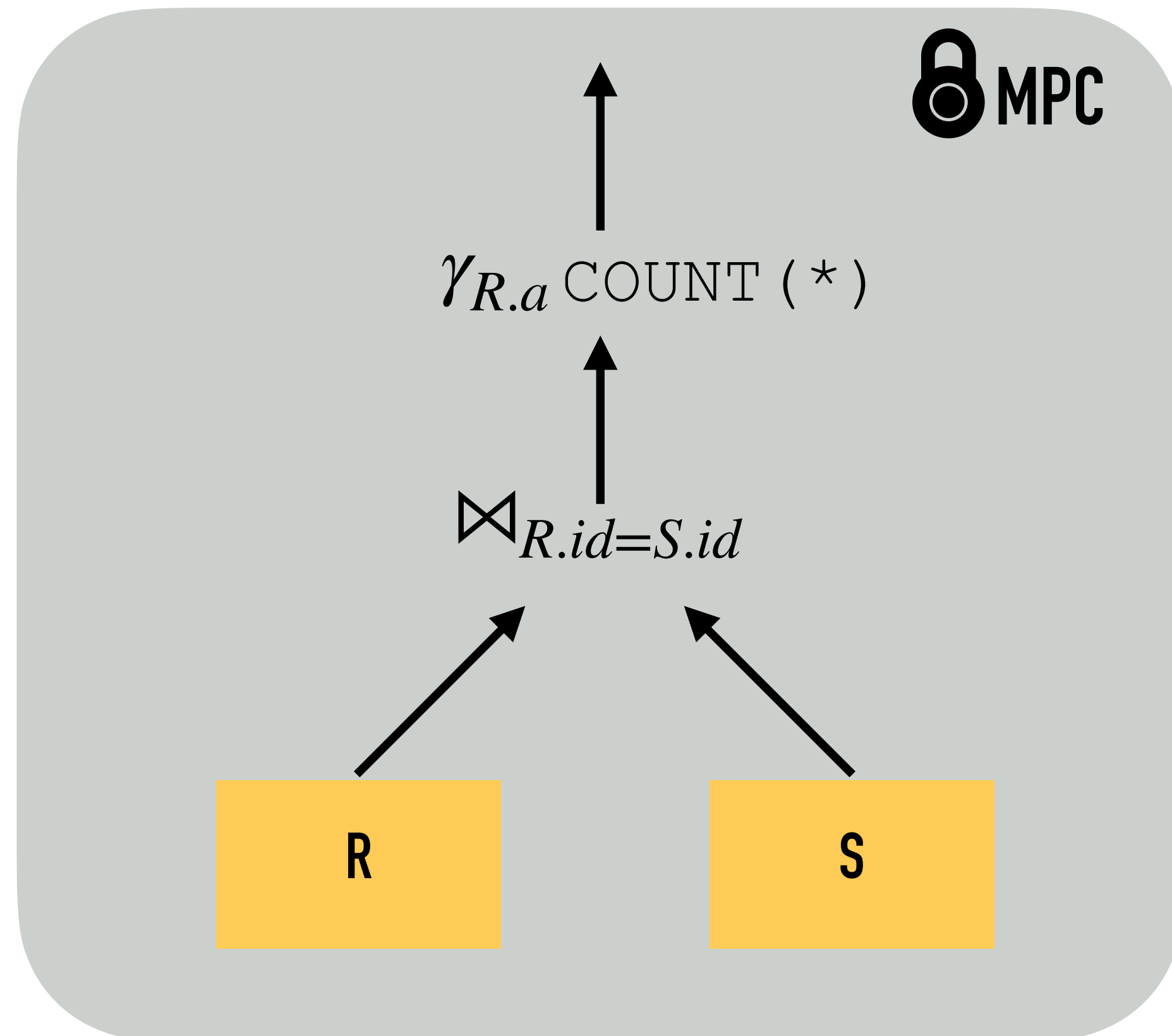


```
SELECT R.a, COUNT (*)  
FROM R, S  
WHERE R.id = S.id  
GROUP-BY R.a
```

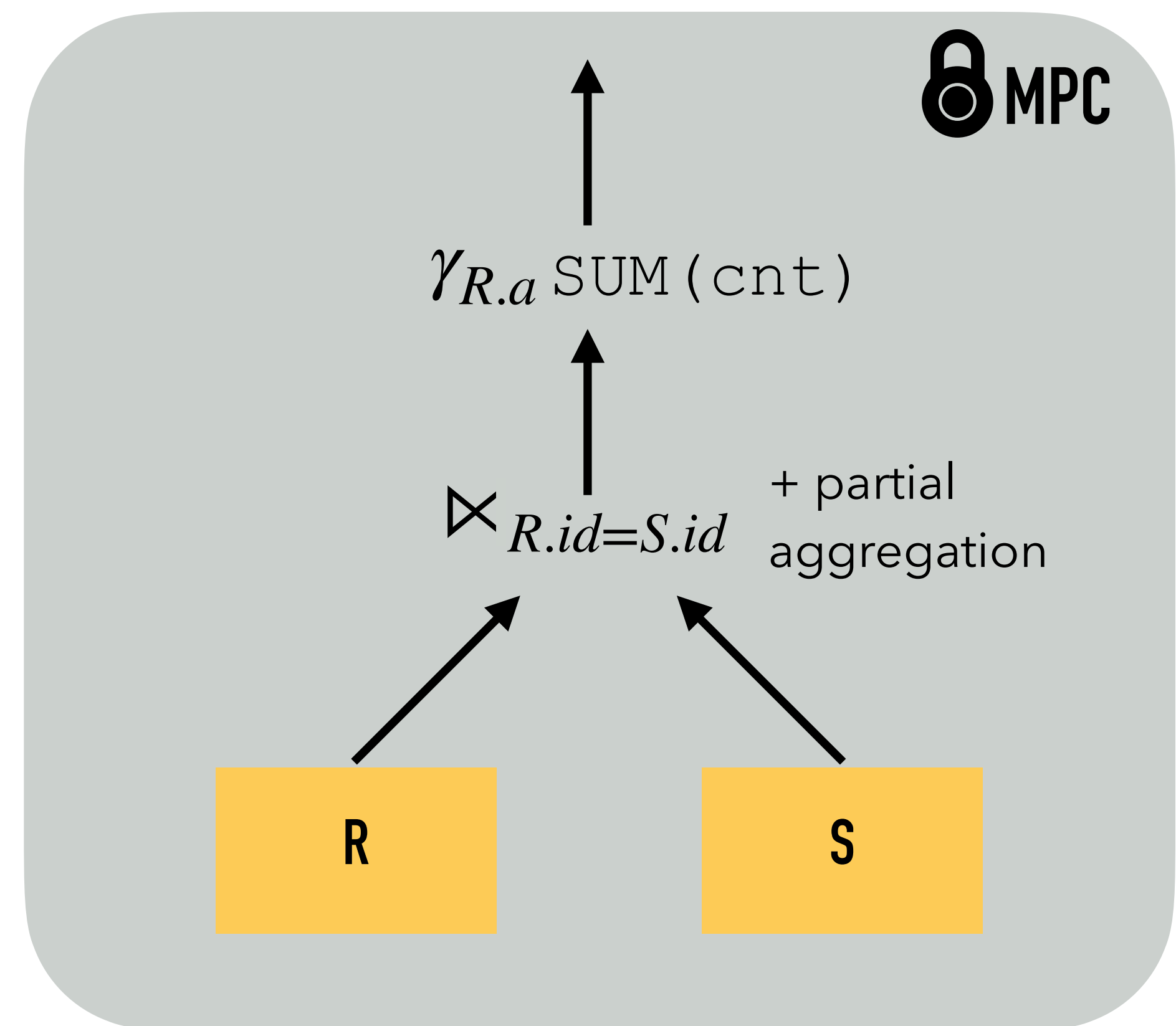
*We can instead decompose the aggregation in two parts and push the first (and most expensive one) down*

*Pushing GROUP-BY down does not produce a semantically equivalent plan*

# EXAMPLE: OPERATOR DECOMPOSITION



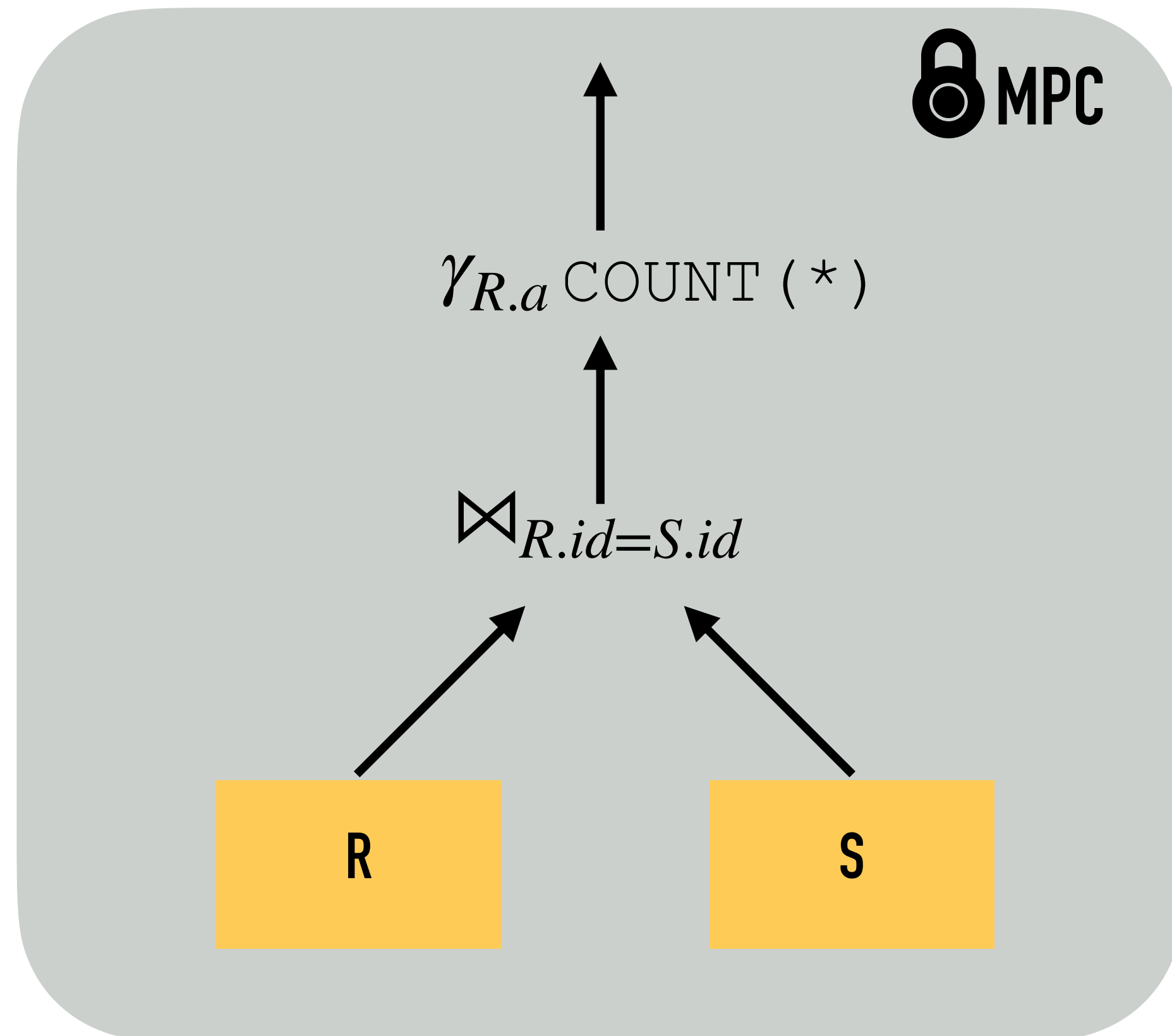
*Pushing GROUP-BY down does not produce a semantically equivalent plan*



*cnt is the number of times each id in R matched with an id in S during the SEMI-JOIN*

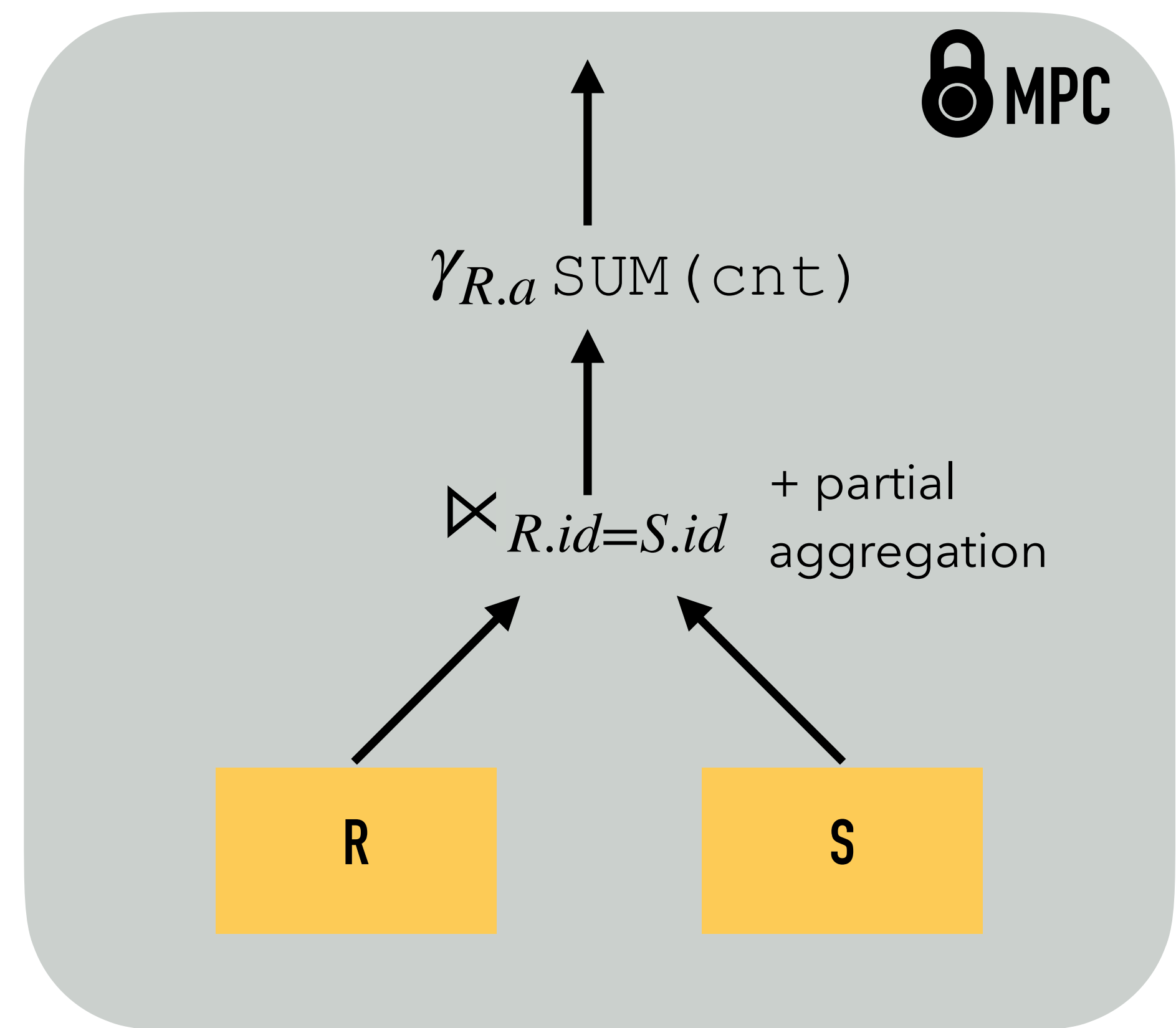


# EXAMPLE: OPERATOR DECOMPOSITION



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

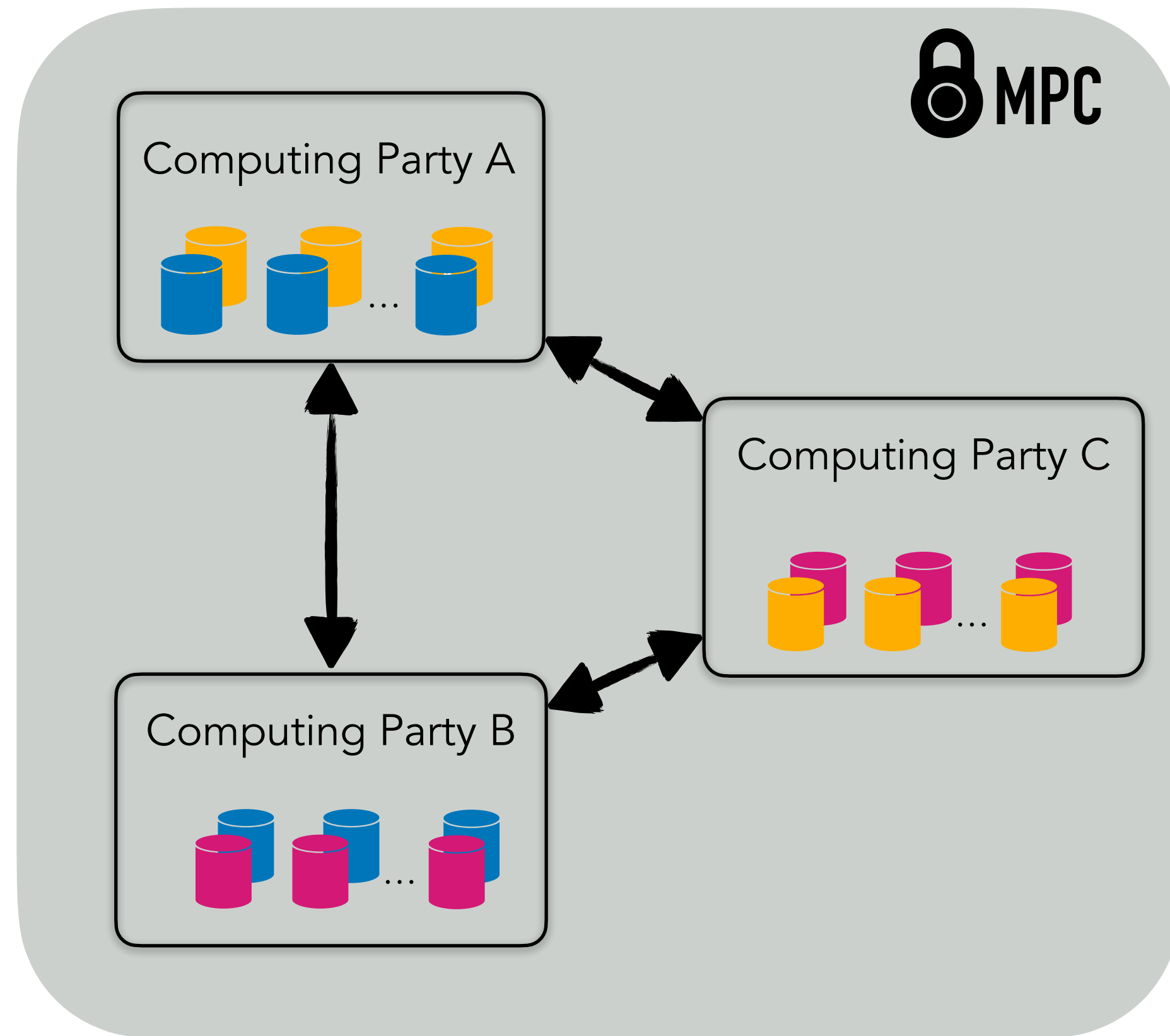
$O(n^2)$  rounds  $O(n^2)$  space



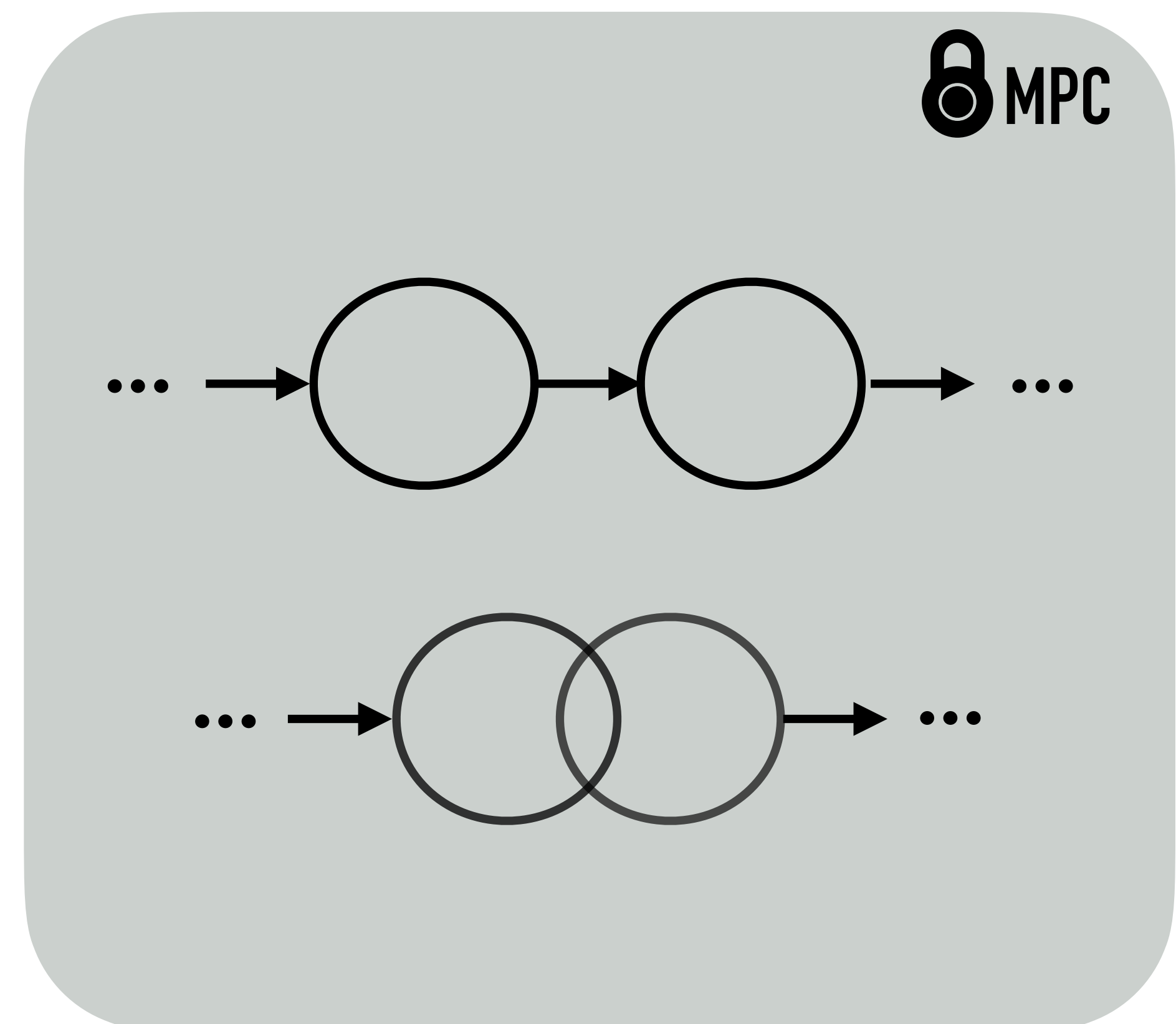
$O(n^2)$  operations / messages

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

# PHYSICAL OPTIMIZATIONS IN SECRECY

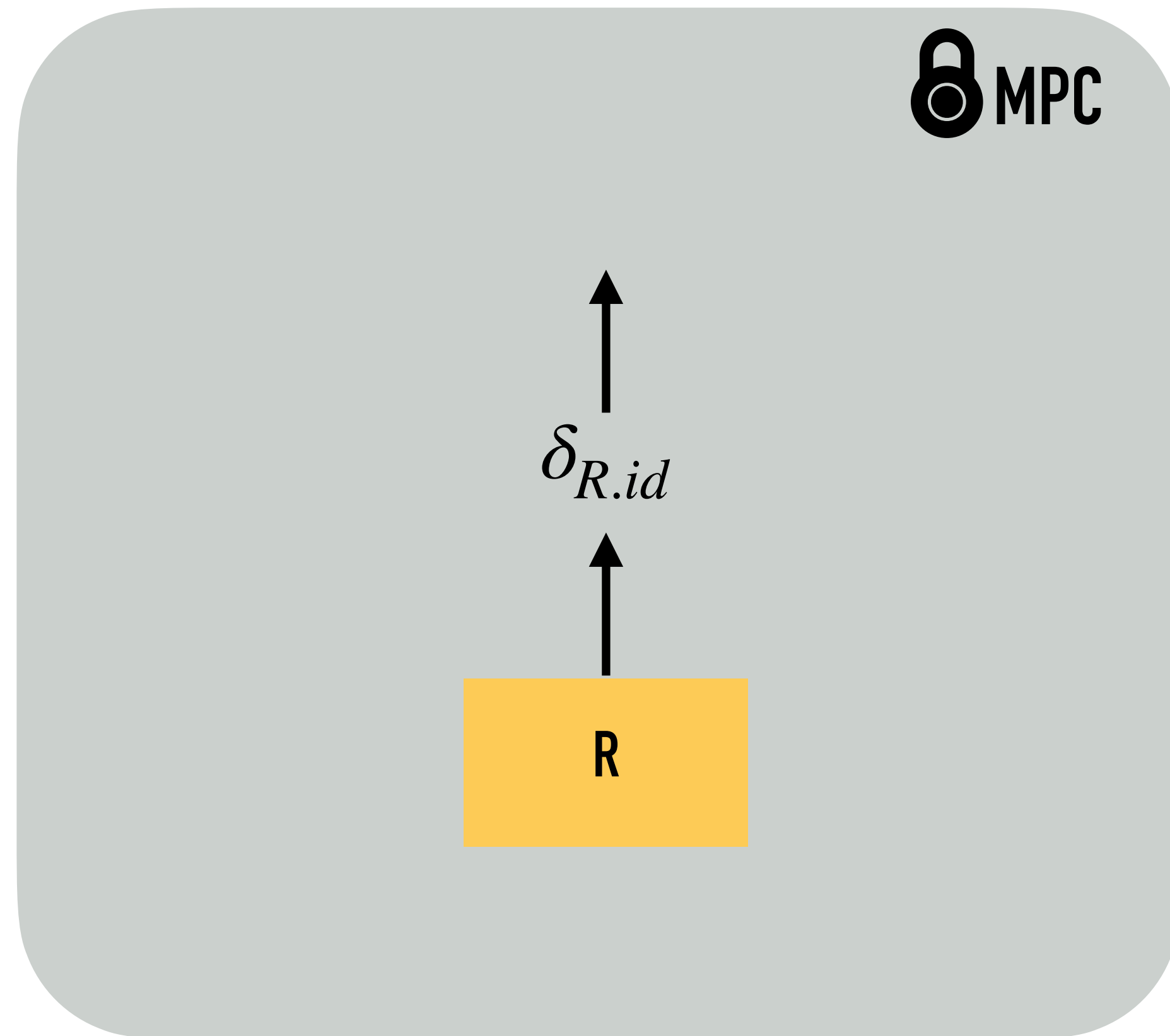


*Message Batching*

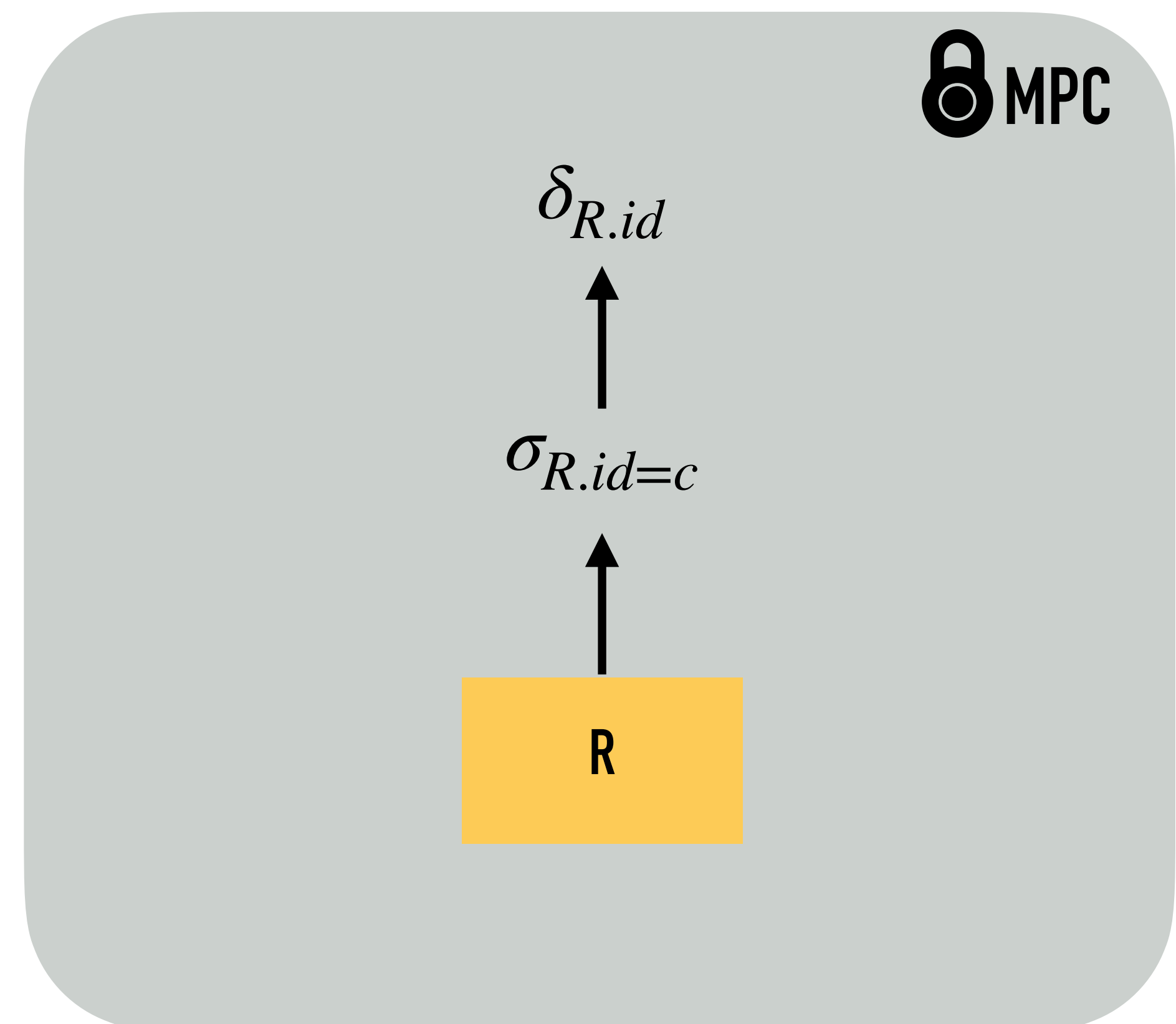


*Operator Fusion*

# EXAMPLE: OPERATOR FUSION

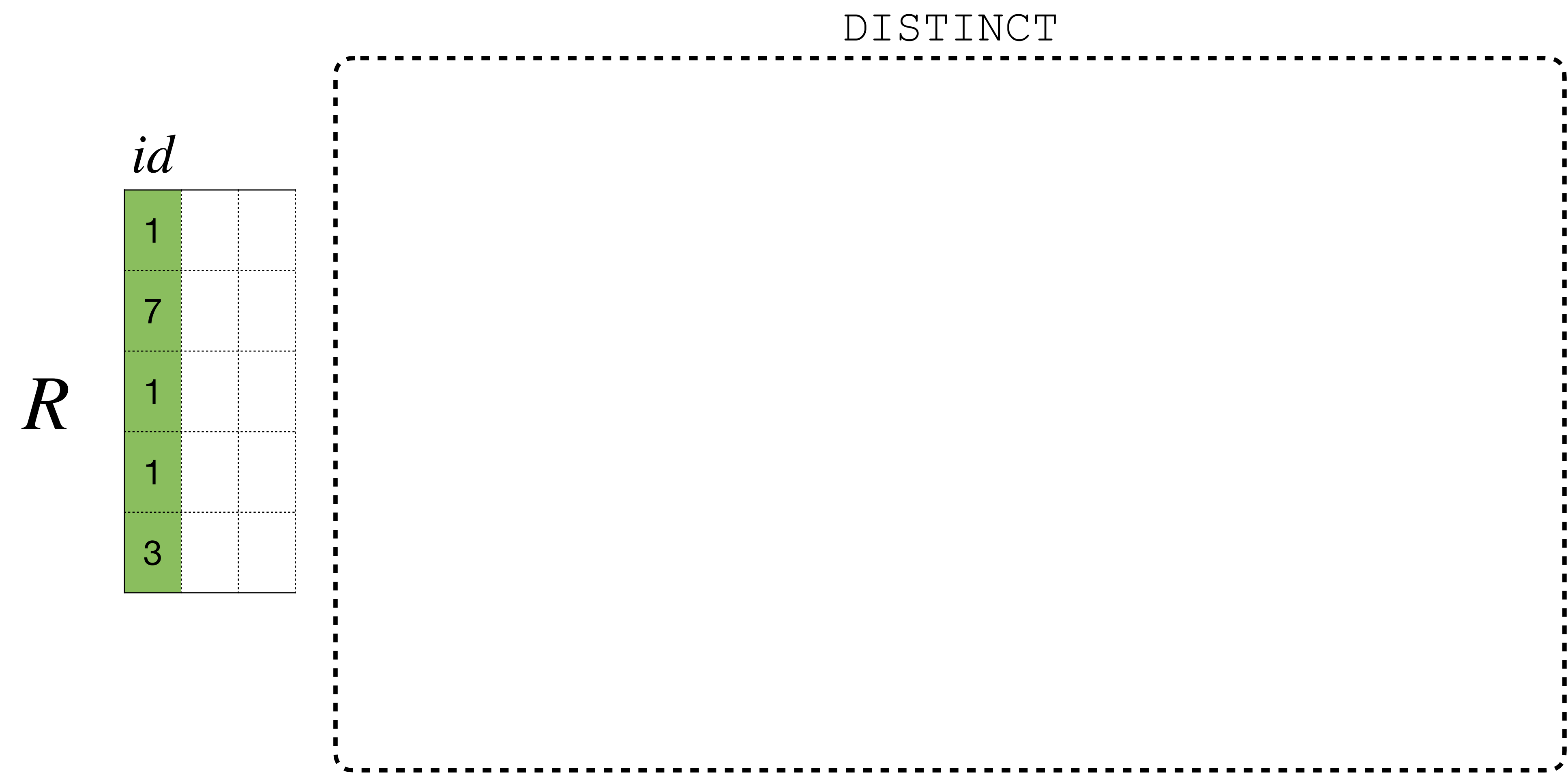


*Applying DISTINCT to a base relation requires  $O(\log^2 n)$  rounds*

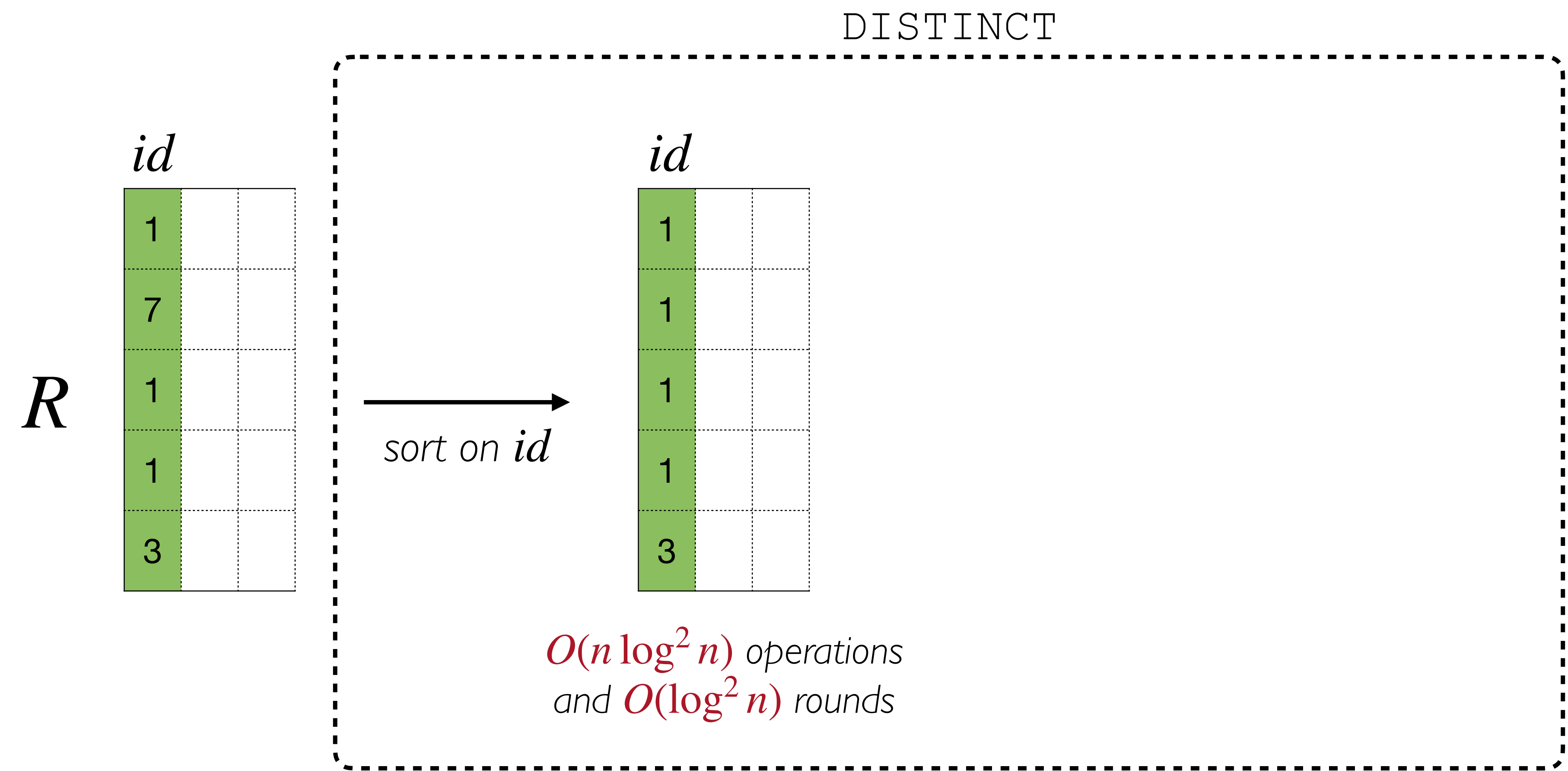


*Applying DISTINCT to the output of another operator requires  $O(n)$  rounds*

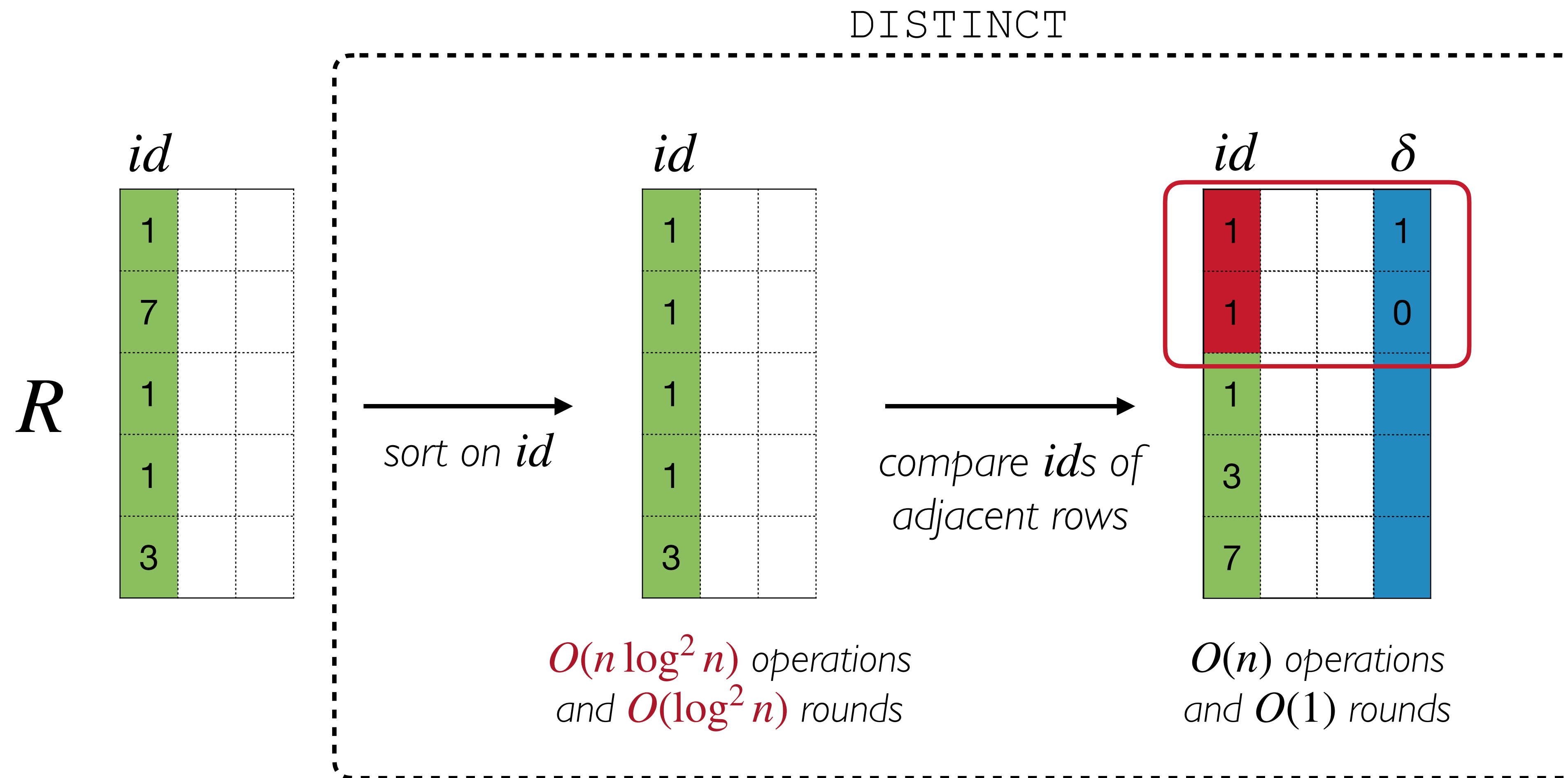
# EXAMPLE: OPERATOR FUSION



# EXAMPLE: OPERATOR FUSION



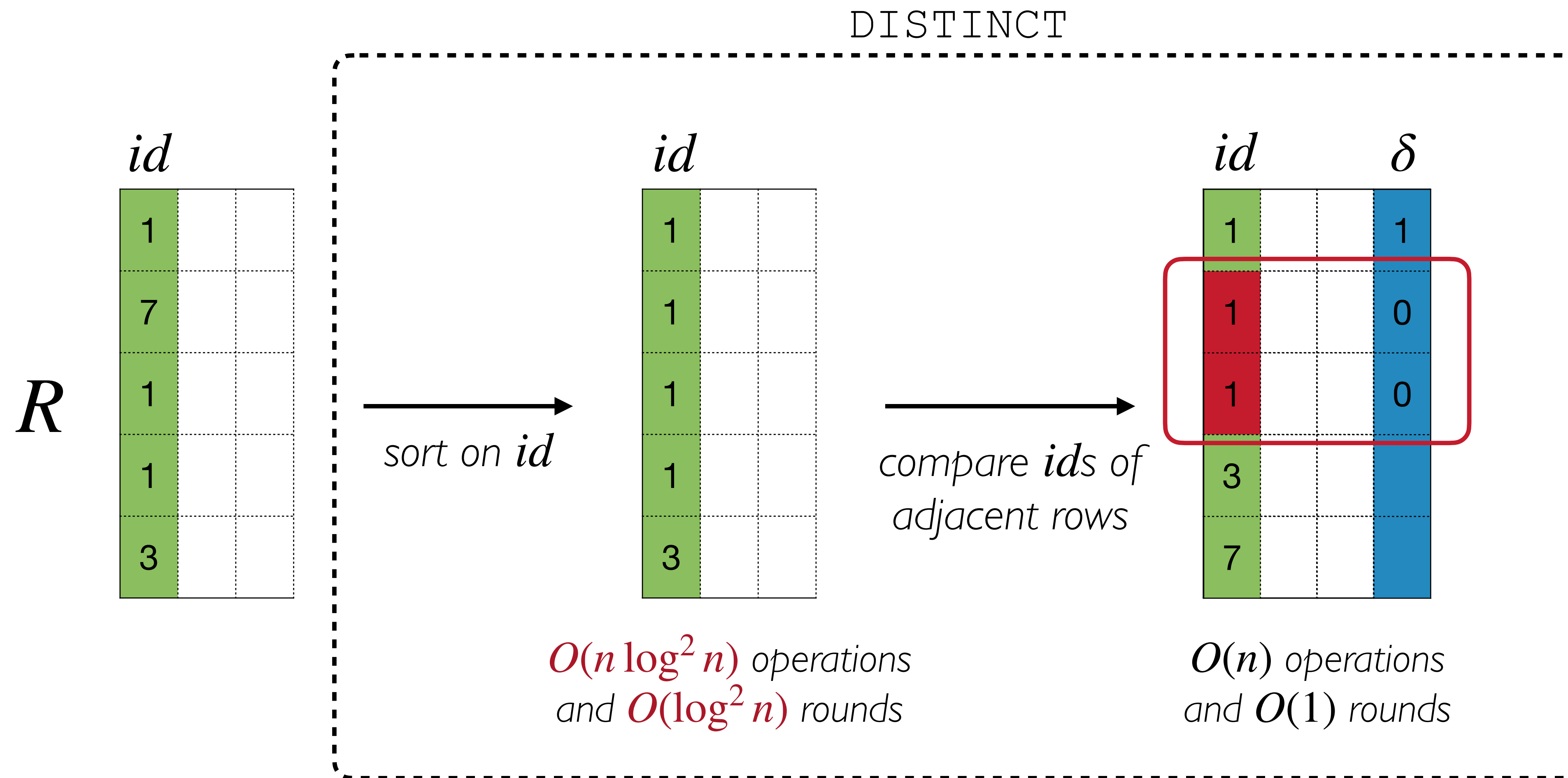
# EXAMPLE: OPERATOR FUSION



All adjacent equality comparisons in the second phase of DISTINCT are independent and can be performed in bulk

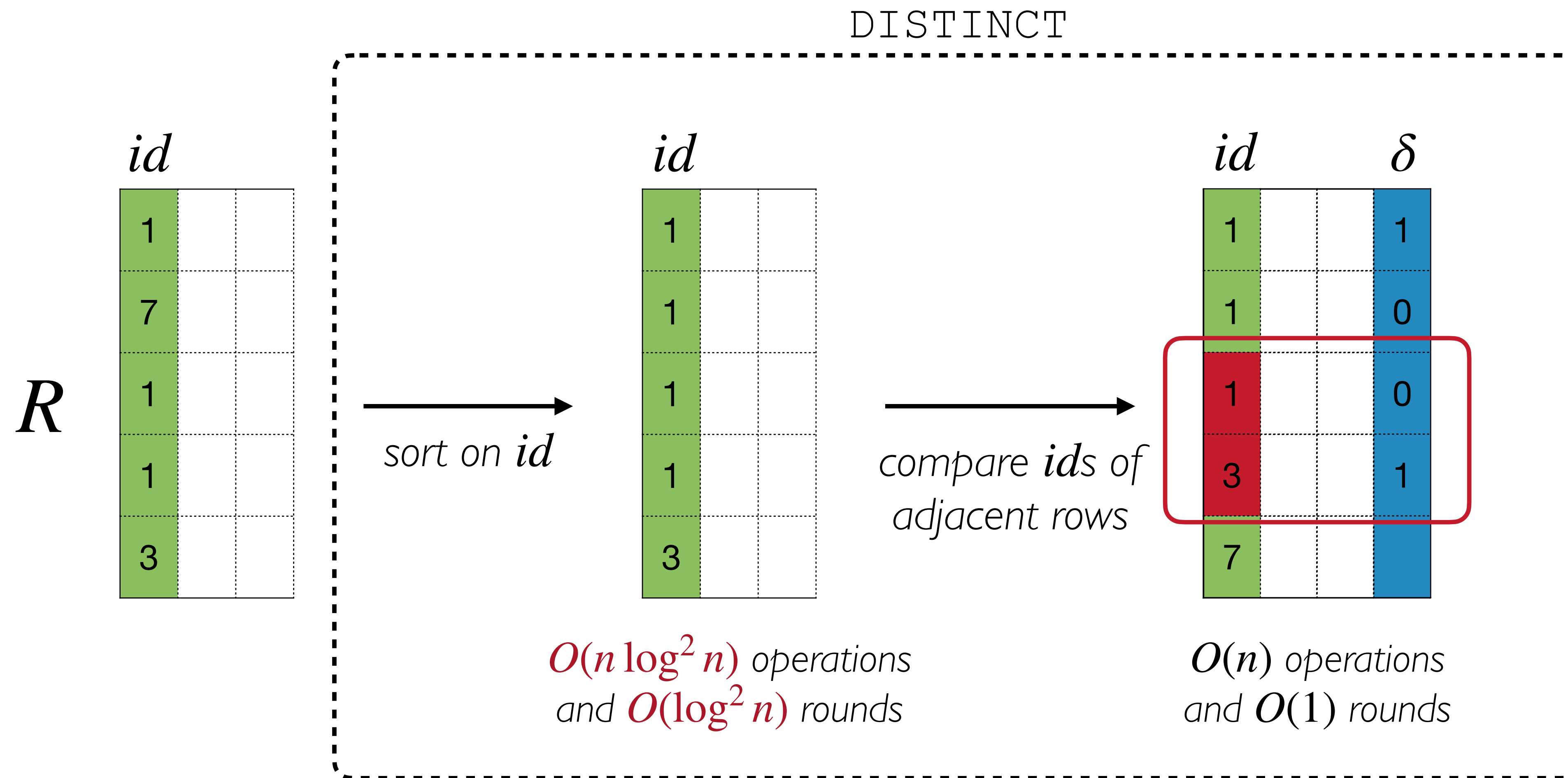


# EXAMPLE: OPERATOR FUSION



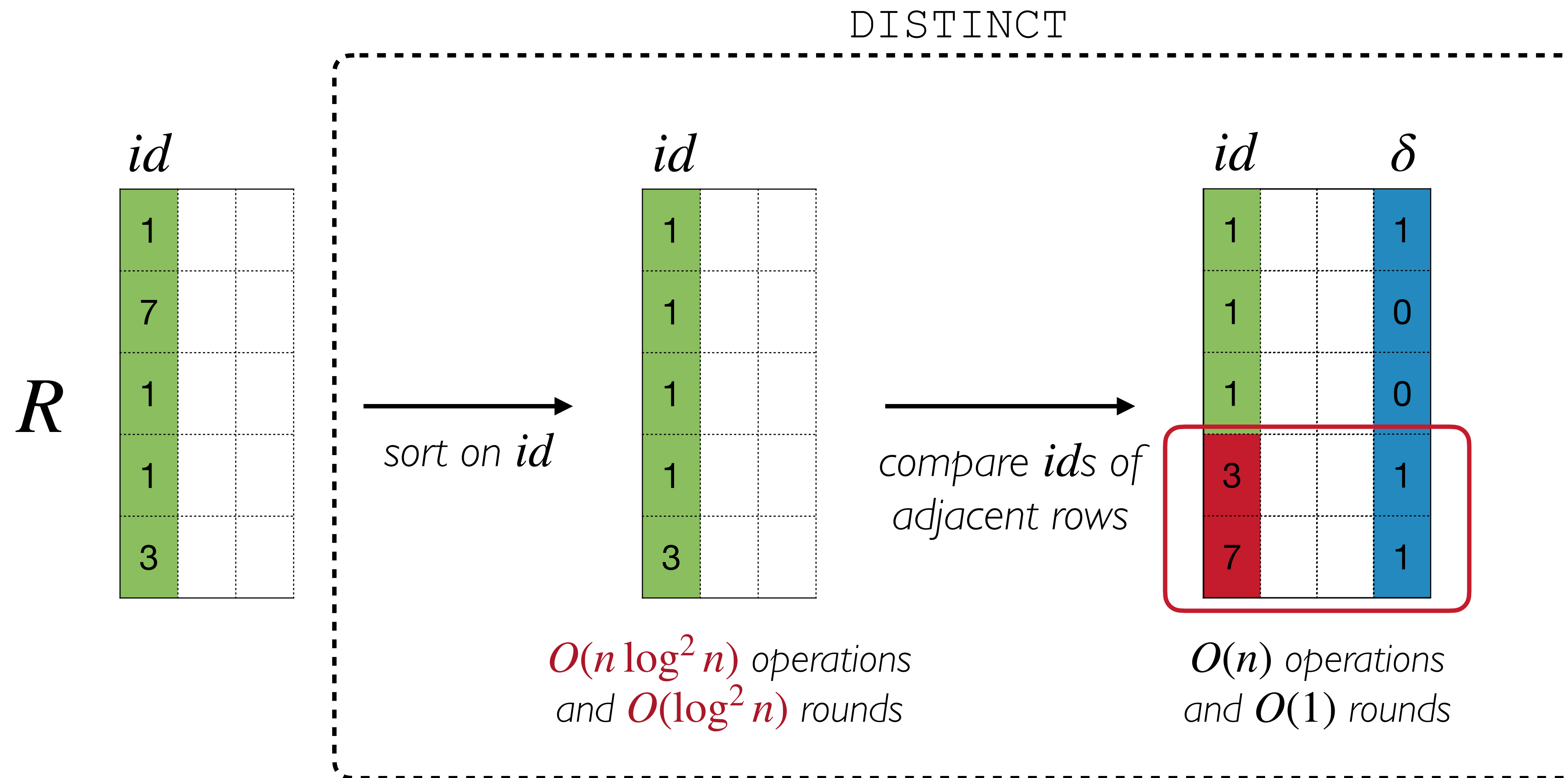
All adjacent equality comparisons in the second phase of DISTINCT are independent and can be performed in bulk

# EXAMPLE: OPERATOR FUSION



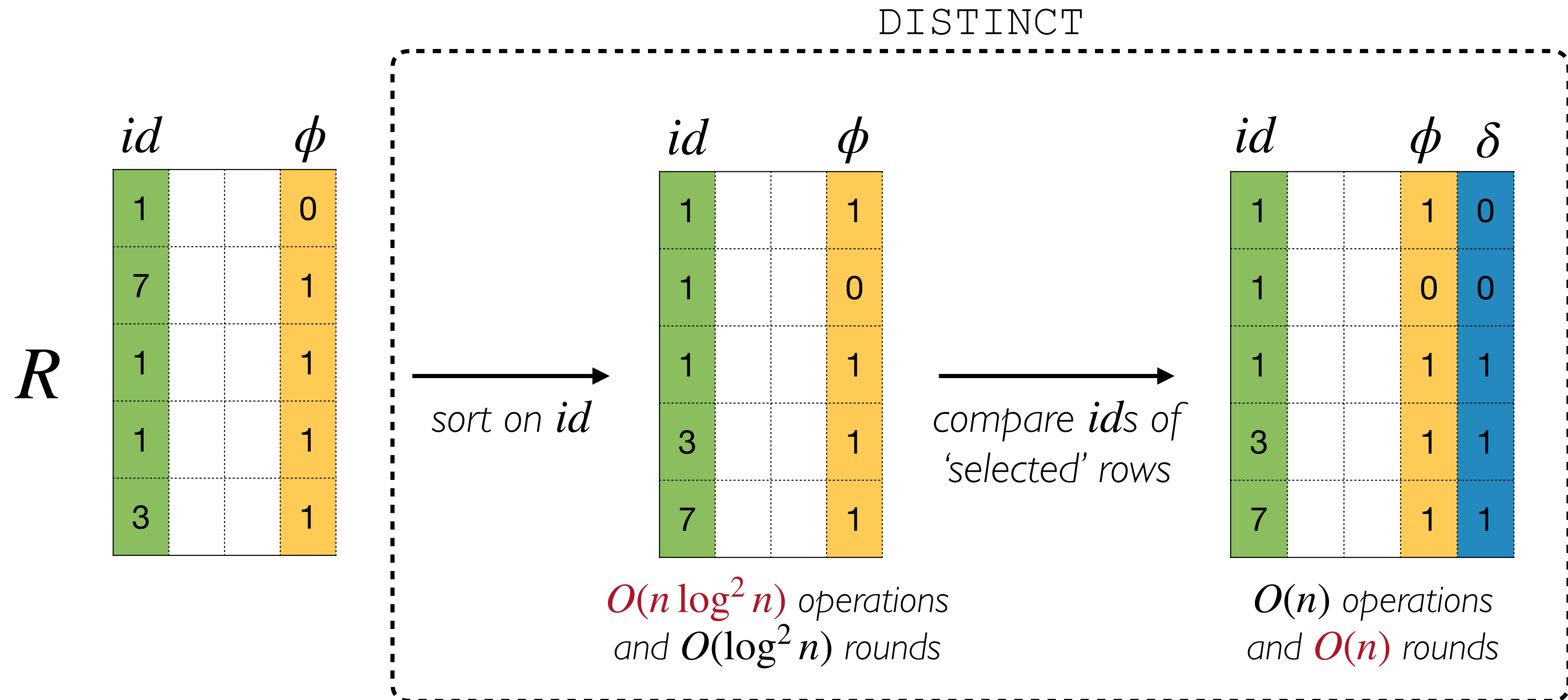
All adjacent equality comparisons in the second phase of DISTINCT are independent and can be performed in bulk

# EXAMPLE: OPERATOR FUSION



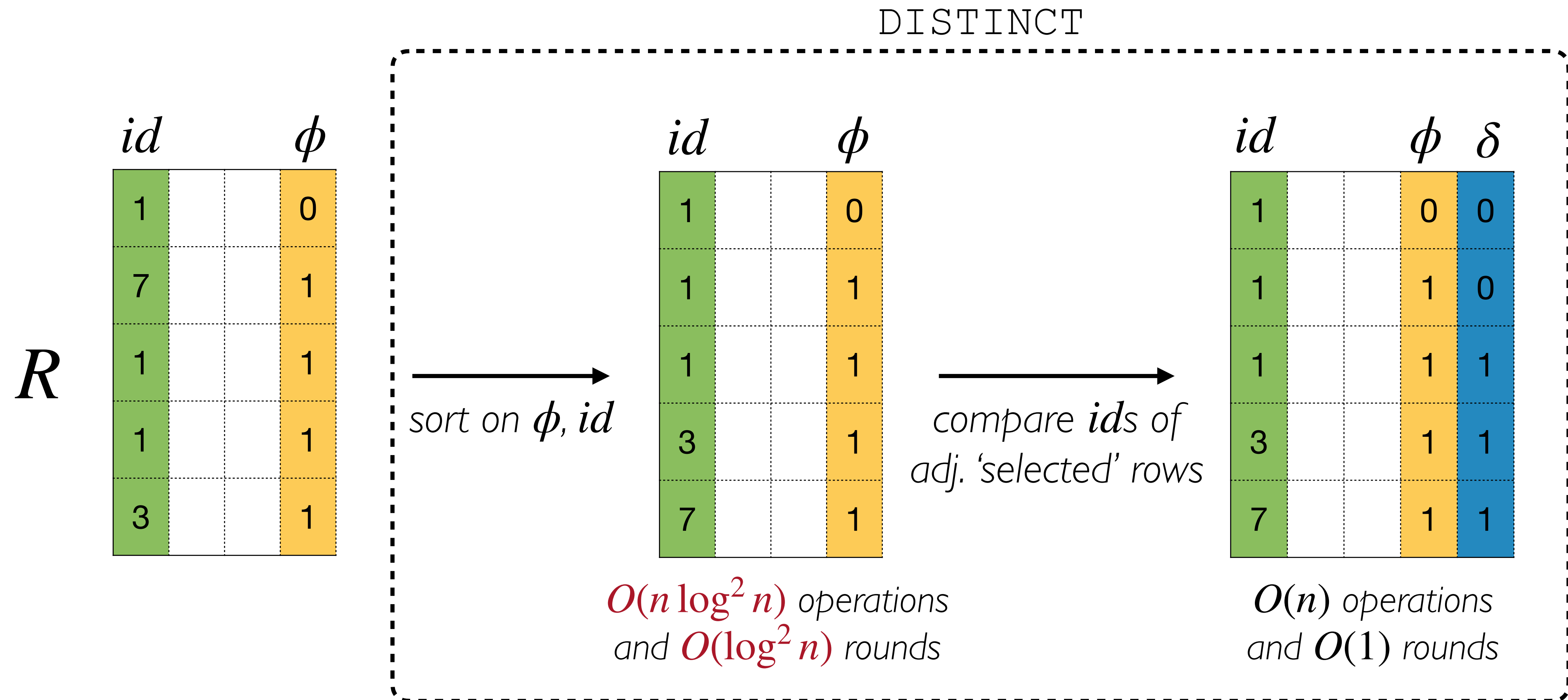
All adjacent equality comparisons in the second phase of DISTINCT are independent and can be performed in bulk

# EXAMPLE: OPERATOR FUSION



Equality checks in the second phase of DISTINCT must remember previously 'selected' rows with the same  $id$  and  $\delta = 1$

# EXAMPLE: OPERATOR FUSION



Taking  $\phi$  into account in the the two phases of DISTINCT allows setting  $\delta$  in bulk and reduces the overall composition cost in number of rounds

# SECRET-SHARING OPTIMIZATIONS IN SECRECY



patient_id (boolean)	diagnosis_code (boolean)	#visits (arithmetic)
...	...	...
...	...	...
...	...	...
...	...	...
...	...	...


```
SELECT COUNT(visits)
FROM Diagnosis
WHERE diagnosis_code=cdiag
```

“Count the total number of visits for all patients with diagnosis code cdiag”

Dual Sharing



# SECRET-SHARING OPTIMIZATIONS IN SECRECY

 MPC

patient_id (boolean)	diagnosis_code (boolean)	#visits (arithmetic)	$\phi$
...	...	...	...
...	...	...	...
...	...	...	...
...	...	...	...
...	...	...	...

Dual Sharing

```
SELECT COUNT(visits)
FROM Diagnosis
WHERE diagnosis_code=cdiag
```

“Count the total number of visits for all patients with diagnosis code cdiag”

Secrecy constructs boolean and arithmetic shares of the selection bit  $\phi$  to speedup multiplication with #visits under MPC  
(boolean-to-arithmetic single-bit conversion requires two rounds)

# SECRET-SHARING OPTIMIZATIONS IN SECRECY




patient_id (boolean)	diagnosis_code (boolean)	last_visit (boolean)
...	...	...
...	...	...
...	...	...
...	...	...
...	...	...

```
SELECT DISTINCT patient_id
FROM Diagnosis as d, Medication as m
WHERE m.date - d.last_visit > 10 days
```

“Find all patients who have been prescribed medication in more than 10 days after their last visit”

*Proactive Sharing*

# SECRET-SHARING OPTIMIZATIONS IN SECRECY



patient_id (boolean)	diagnosis_code (boolean)	last_visit+10 (boolean)
...	...	...
...	...	...
...	...	...
...	...	...
...	...	...

```
SELECT DISTINCT patient_id
FROM Diagnosis as d, Medication as m
WHERE m.date - d.last_visit > 10 days
```

“Find all patients who have been prescribed medication in more than 10 days after their last visit”

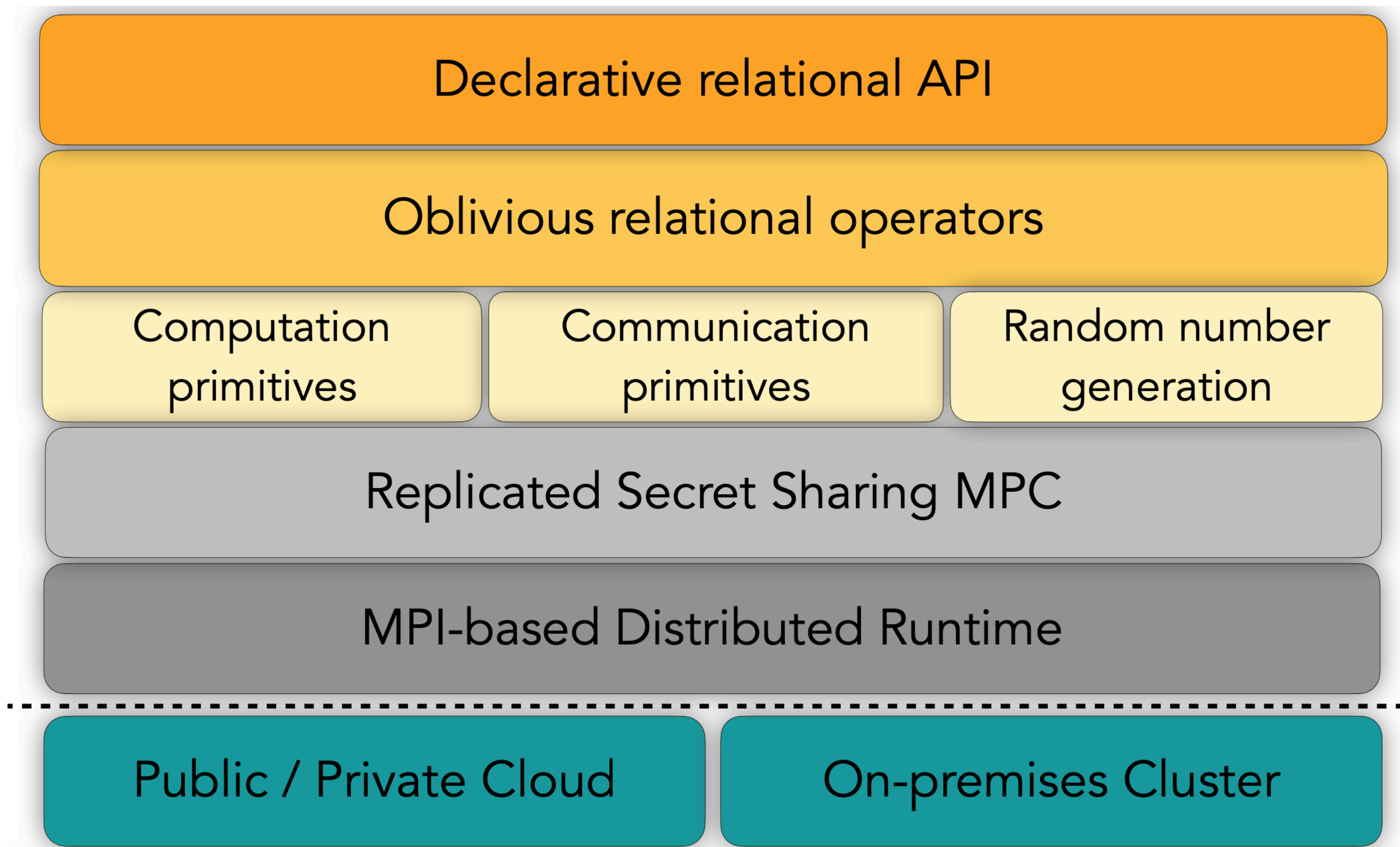
*Secrecy secret-shares the result of last\_visit+10 to avoid using a boolean Ripple-Carry Adder under MPC*

*Proactive Sharing*

---

# System Implementation

# THE SECRECY STACK



# DECLARATIVE RELATIONAL API

## Query in SQL syntax

```
SELECT diag, COUNT(*) cnt
FROM diagnosis
WHERE pid IN cdiff_cohort
GROUP BY diag
ORDER BY cnt DESC
LIMIT 10
```

*“Select the 10 most common  
diagnosis codes among patients  
in a cohort”*

## Query in Secrecy's API

---

```
1  /** Commorbidity Query */
2  BTable t1 = get_shares(diagnosis);
3  BTable t2 = get_shares(cohort);
4
5  // Sort t1 on diag (at index 2)
6  bitonic_sort(&t1, 2, ASC);
7  in(&t1, &t2, 0, 0); // Semi-join on pid
8  group_by_count(&t1, 2); // Group-by on diag
9  // Sort t1 on count (at index 4)
10 bitonic_sort(&t1, 4, ASC);
11 open(t1, 10); // Open first 10 rows
```

---



---

# Experiments

---

# EVALUATION

1. Performance on real and synthetic queries
2. Comparison with other MPC frameworks
3. Scaling behavior
4. Benefits of individual optimizations
5. Micro-benchmarks

---

# EVALUATION

1. Performance on real and synthetic queries
2. Comparison with other MPC frameworks
3. Scaling behavior
4. Benefits of individual optimizations
5. Micro-benchmarks

# EXPERIMENTAL SETUP

\* More queries in our extended TR

Experiments with 8 relational queries\*:

- 5 real-world queries used in previous works on relational MPC:  
*“Comorbidity”, “Recurrent C. Diff.”, “Aspirin Count”, “Credit Score”, “Password Reuse”*
- 3 TPC-H queries (Q4, Q6, Q13)

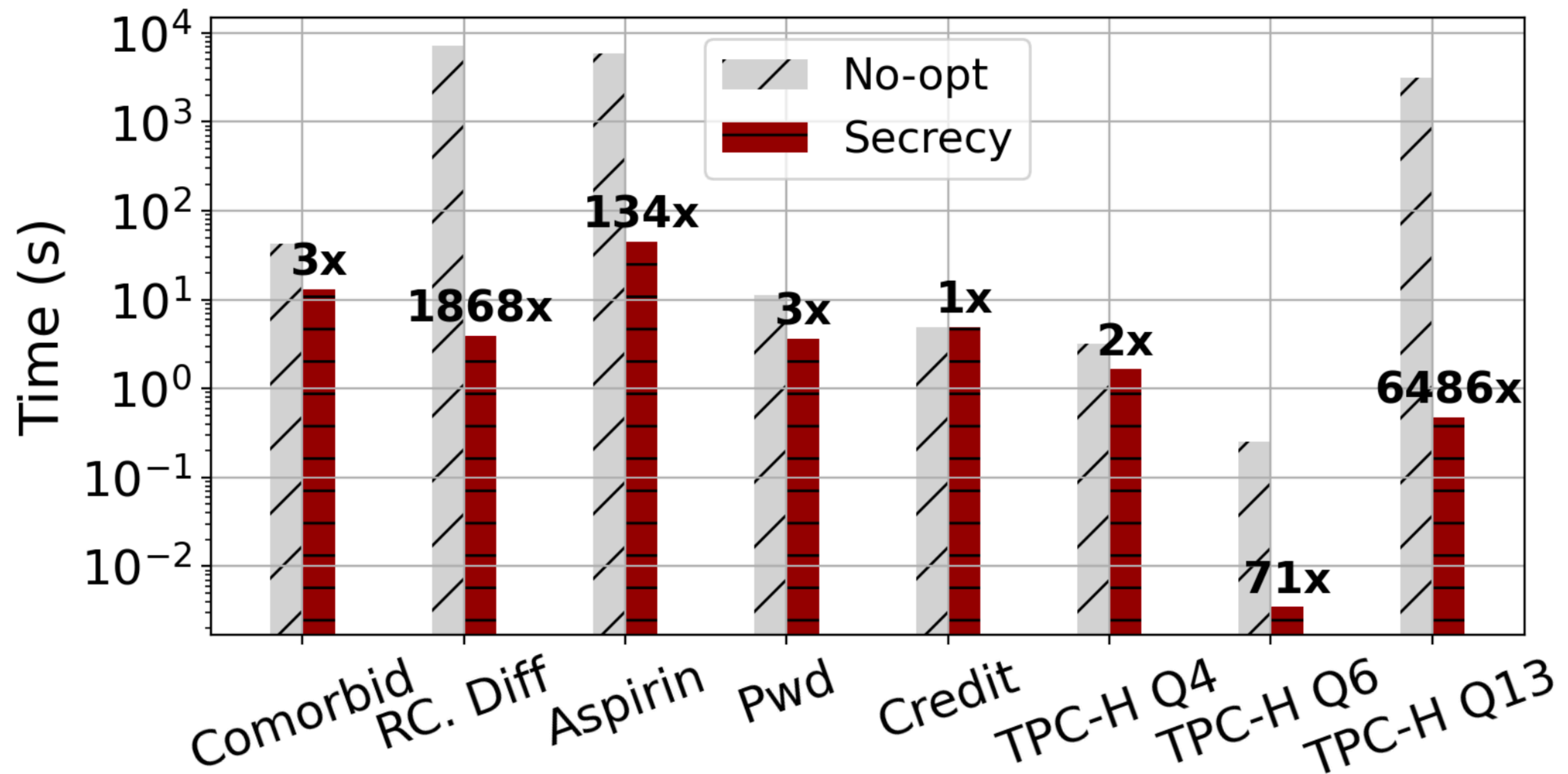
All experiments use randomly generated 64-bit shares

- Using real data is no different — parties always operate on random shares
- Share size can be increased to any  $2^k$  without modifying the protocols

All experiments are on the Massachusetts Open Cloud (MOC)

- 3 VMs (one per party) with 32GB RAM each running MPICH 1.4
- **One CPU thread per party** (for both computation and communication)

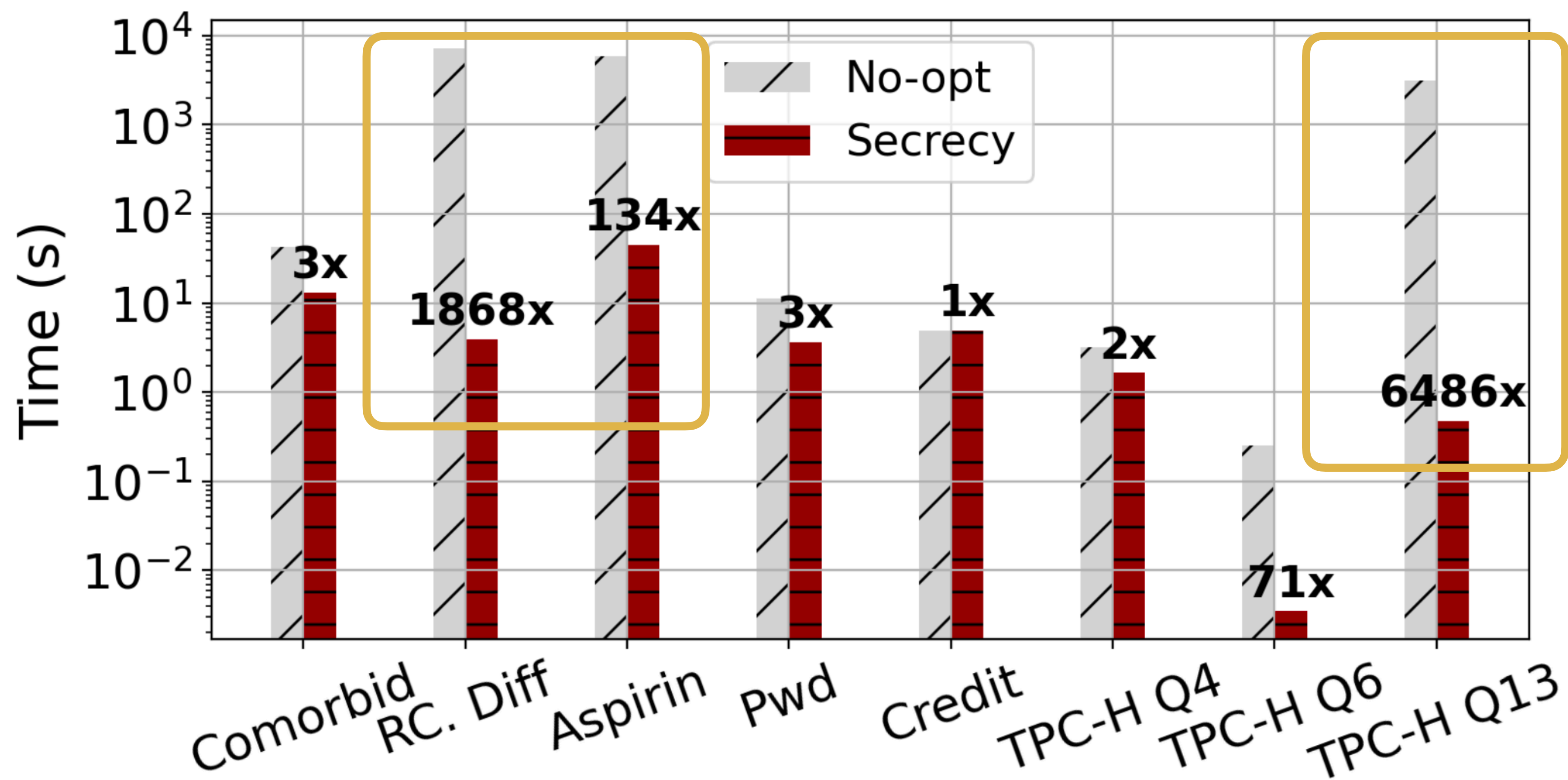
# PERFORMANCE ON REAL AND SYNTHETIC QUERIES



*Non-optimized plans use message batching too, otherwise the cost of MPC is prohibitive and these queries cannot scale to more than a few hundred input rows*

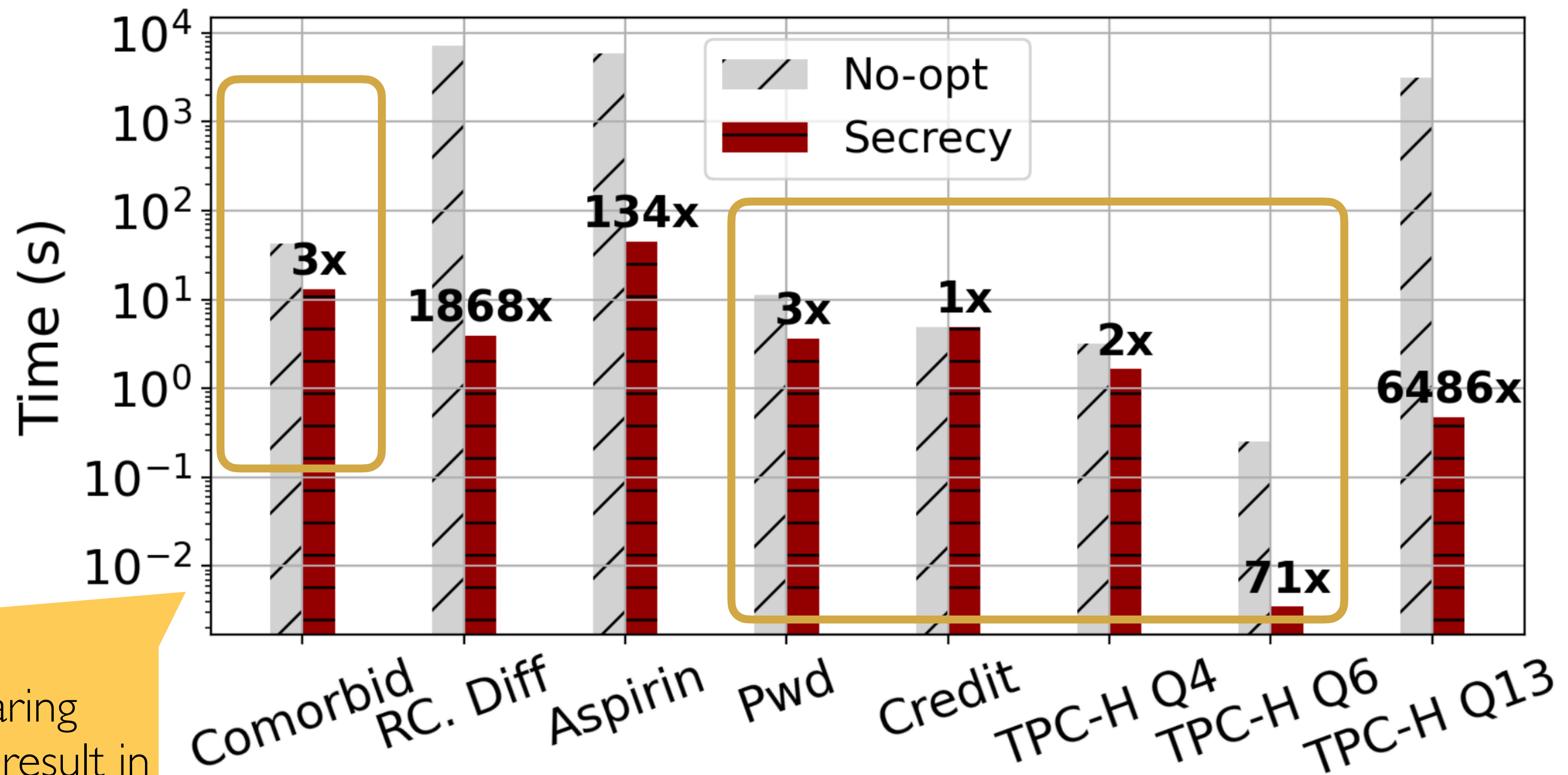
# PERFORMANCE ON REAL AND SYNTHETIC QUERIES

Logical and physical optimizations result in over 100 × speedups





# PERFORMANCE ON REAL AND SYNTHETIC QUERIES



Secret-sharing optimizations result in up to 71 × speedups

# COMPARISON WITH OTHER RELATIONAL MPC FRAMEWORKS

Framework	MPC Protocol	Information Leakage	Trusted Party	Query Execution
Conclave	Secret Sharing / Garbled Circuits	Controlled (Hybrid operators)	Yes	Hybrid
SMCQL	Garbled Circuits / ORAM	No	No	Hybrid
Shrinkwrap	Garbled Circuits / ORAM	Controlled (Diff. Privacy)	No	Hybrid
SAQE	Garbled Circuits	Controlled (Diff. Privacy)	No	Hybrid
Senate	Garbled Circuits	No	No	Hybrid
SDB	Secret Sharing	Yes (operator dependent)	No	Hybrid
<i>Secrecy</i>	Repl. Secret Sharing	No	No	End-to-end under MPC



# COMPARISON WITH OTHER RELATIONAL MPC FRAMEWORKS

Framework	MPC Protocol	Information Leakage	Trusted Party	Query Execution
Conclave	Secret Sharing / Garbled Circuits	Controlled (Hybrid operators)	Yes	Hybrid
SMCQL	Garbled Circuits / ORAM	No	No	Hybrid
Shrinkwrap	Garbled Circuits / ORAM	Controlled (Diff. Privacy)	No	Hybrid
SAQE	Garbled Circuits	Controlled (Diff. Privacy)	No	Hybrid
Senate	Garbled Circuits	No	No	Hybrid
SDB	Secret Sharing	Yes (operator dependent)	No	Hybrid
<i>Secrecy</i>	Repl. Secret Sharing	No	No	End-to-end under MPC

The only publicly available framework with a semi-honest model and no information leakage

# COMPARISON WITH OTHER RELATIONAL MPC FRAMEWORKS

Framework	MPC Protocol	Information Leakage	Trusted Party	Query Execution
Conclave	Secret Sharing / Garbled Circuits	Controlled (Hybrid operators)	Yes	Hybrid
SMCQL	Garbled Circuits / ORAM	No	No	Hybrid
Shrinkwrap	Garbled Circuits / ORAM	Controlled (Diff. Privacy)	No	Hybrid
SAQE	Garbled Circuits	Controlled (Diff. Privacy)	No	Hybrid
Senate	Garbled Circuits	No	No	Hybrid
SDB	Secret Sharing	Yes (operator dependent)	No	Hybrid
<b>Secrecy</b>	Repl. Secret Sharing	No	No	End-to-end under MPC

Senate also builds on the malicious version EMP

These two (and a new version of SMCQL) are not publicly available but are based on EMP<sup>1</sup>

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

# SECRECY VS SMCQL

Experiments with 25 rows per input relation

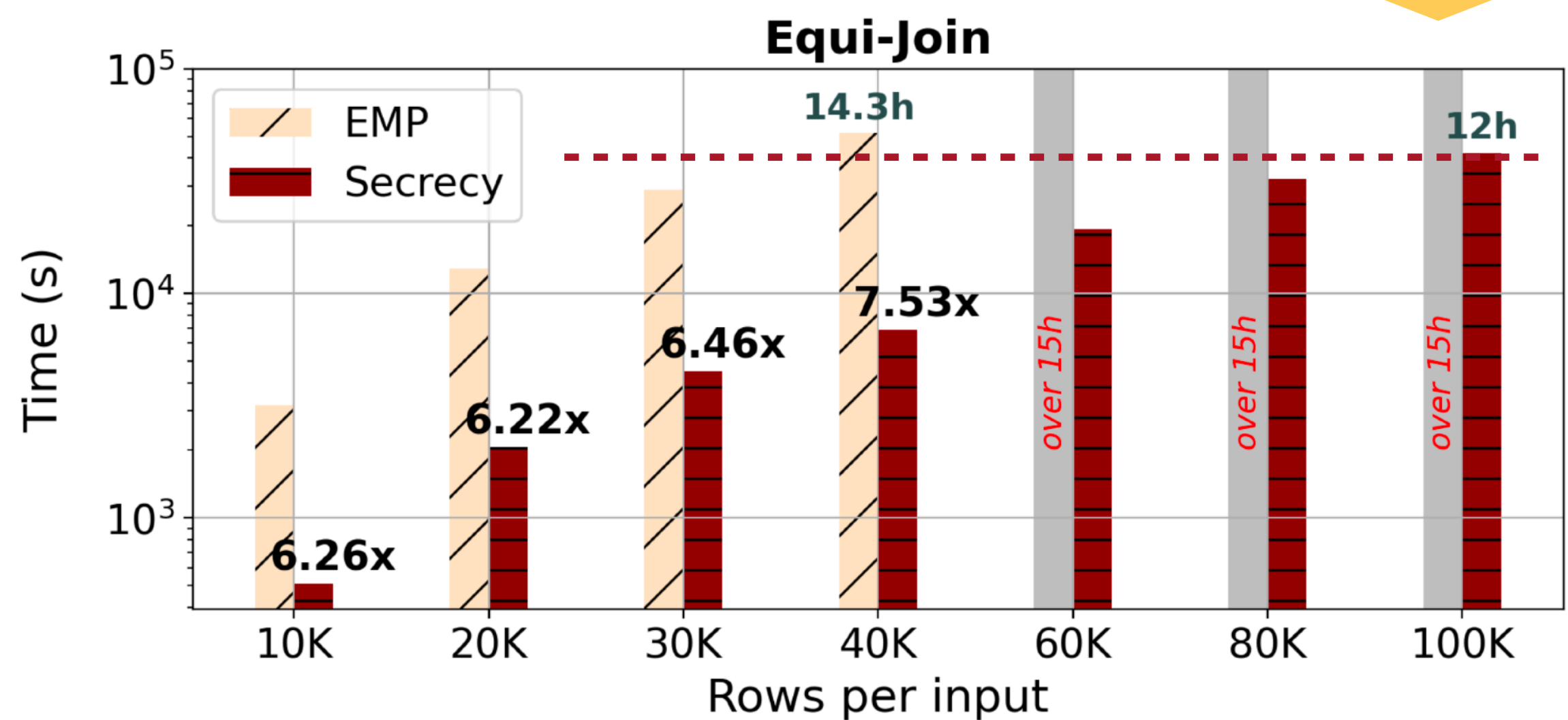
	Comorbidity	Recurrent C. Diff.	Aspirin Count
<b>SMCQL</b>	197s	804s	796s
<b>Secrecy</b>	0.083s	0.092s	0.171s

Secrecy processes all tuples under MPC whereas SMCQL filters many of the rows “in the clear”  
(only 8 of 25 tuples entered the MPC circuit in SMCQL)

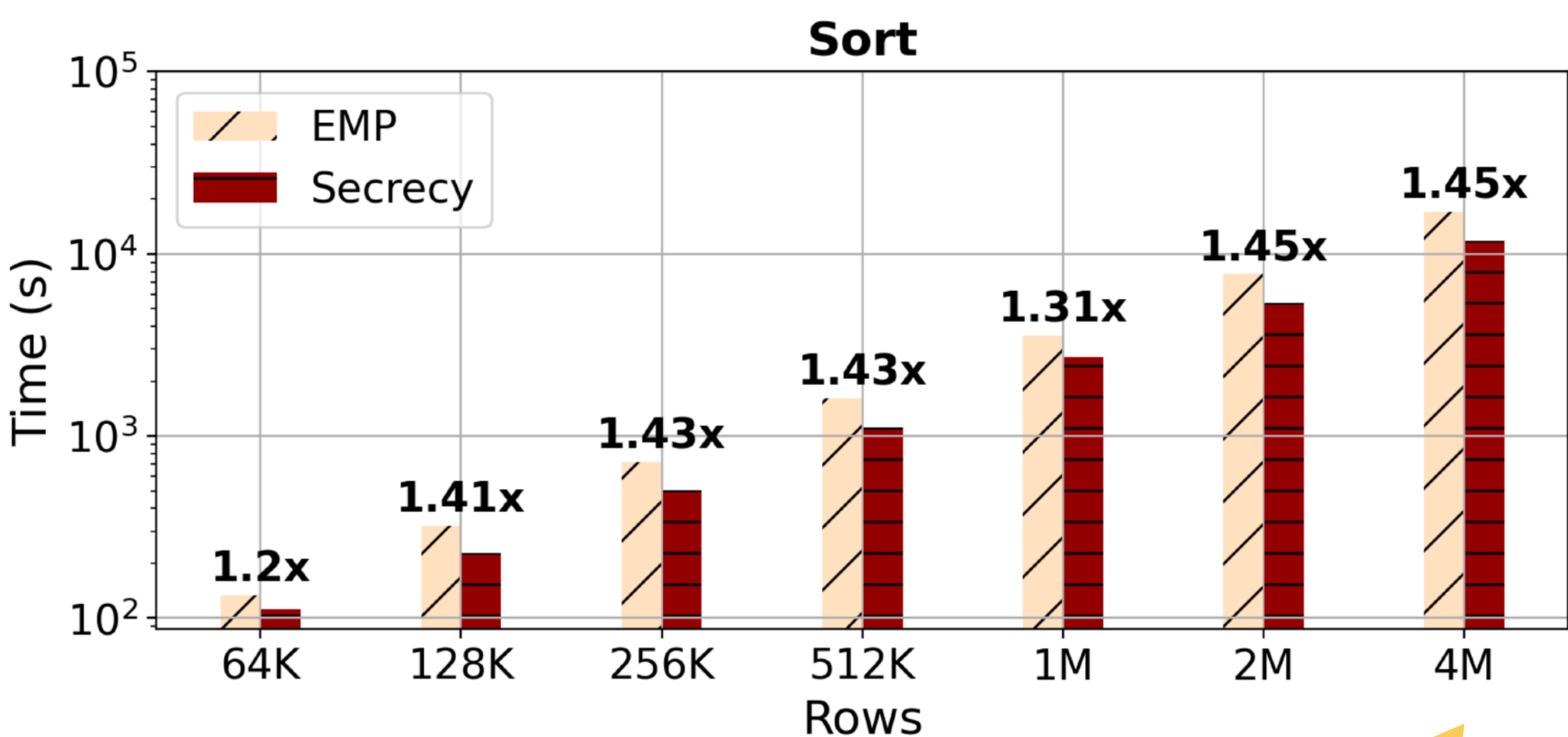
<sup>1</sup> J. Bater, G. Elliott, C. Eggen, S. Goel, A. N. Kho, and J. Rogers. *SMCQL: secure querying for federated databases*. PVLDB, 10(6):673–684, 2017.

# SECRECY VS EMP

Secrecy executes the join on 100K rows per input in ~12h



EMP requires ~14h for 40K rows per input



Secrecy sorts 4M rows in ~3.3h whereas EMP requires ~4.7h

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

# SECRECY SCALING BEHAVIOR

We group queries into three categories:

- Category A: queries with selections and global aggregations
- Category B: queries with select and group-by operators
- Category C: queries with select, group-by, join, and semi-join operators

Category A

TPC-H Q6

Category B

“PASSWORD REUSE”,  
“CREDIT SCORE”,  
“COMORBIDITY”,  
“RECURRENT C. DIFF.”

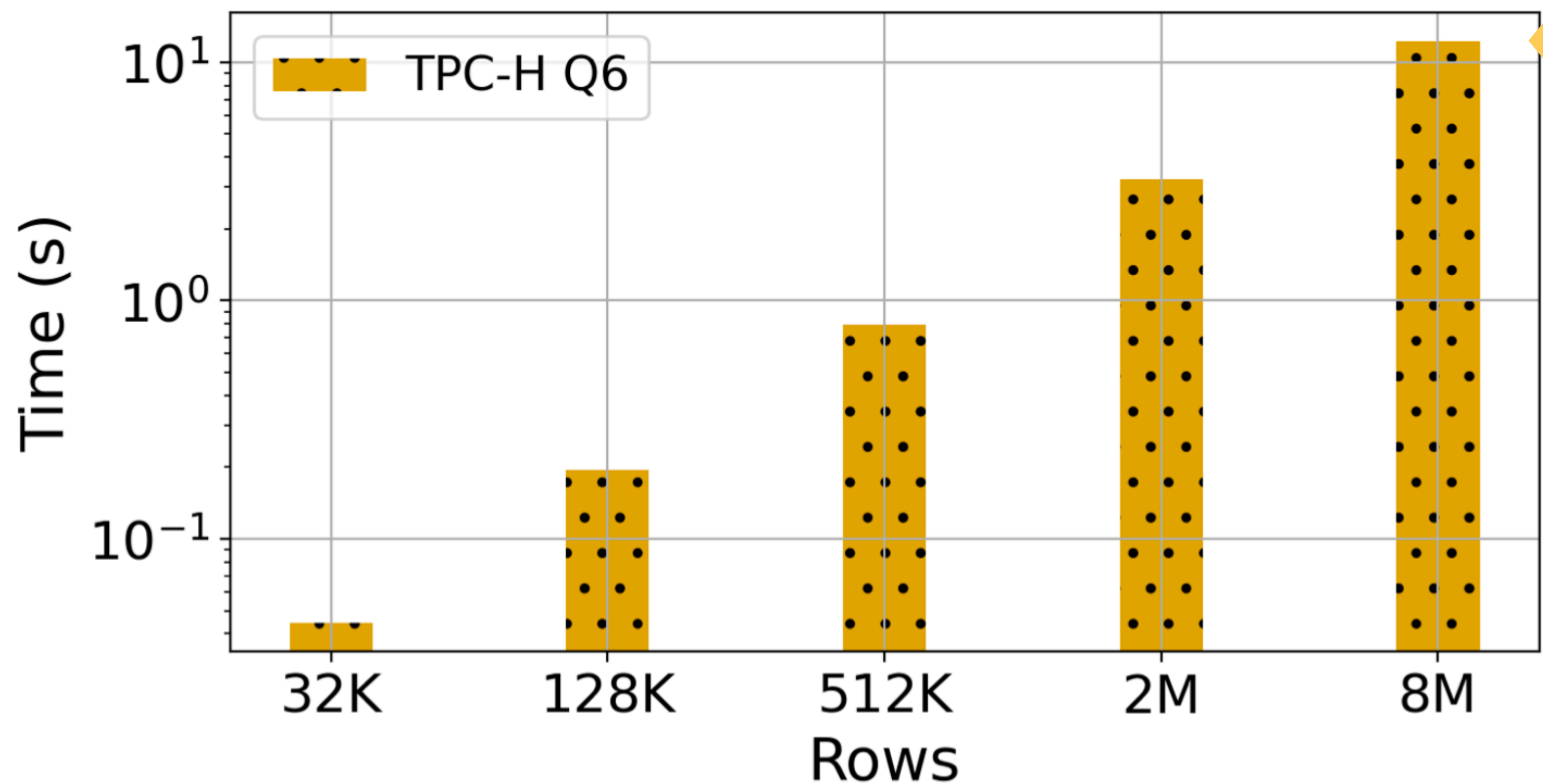
Category C

TPC-H Q4, TPC-H Q13,  
“ASPIRIN COUNT”



# SCALING BEHAVIOR: CATEGORY A

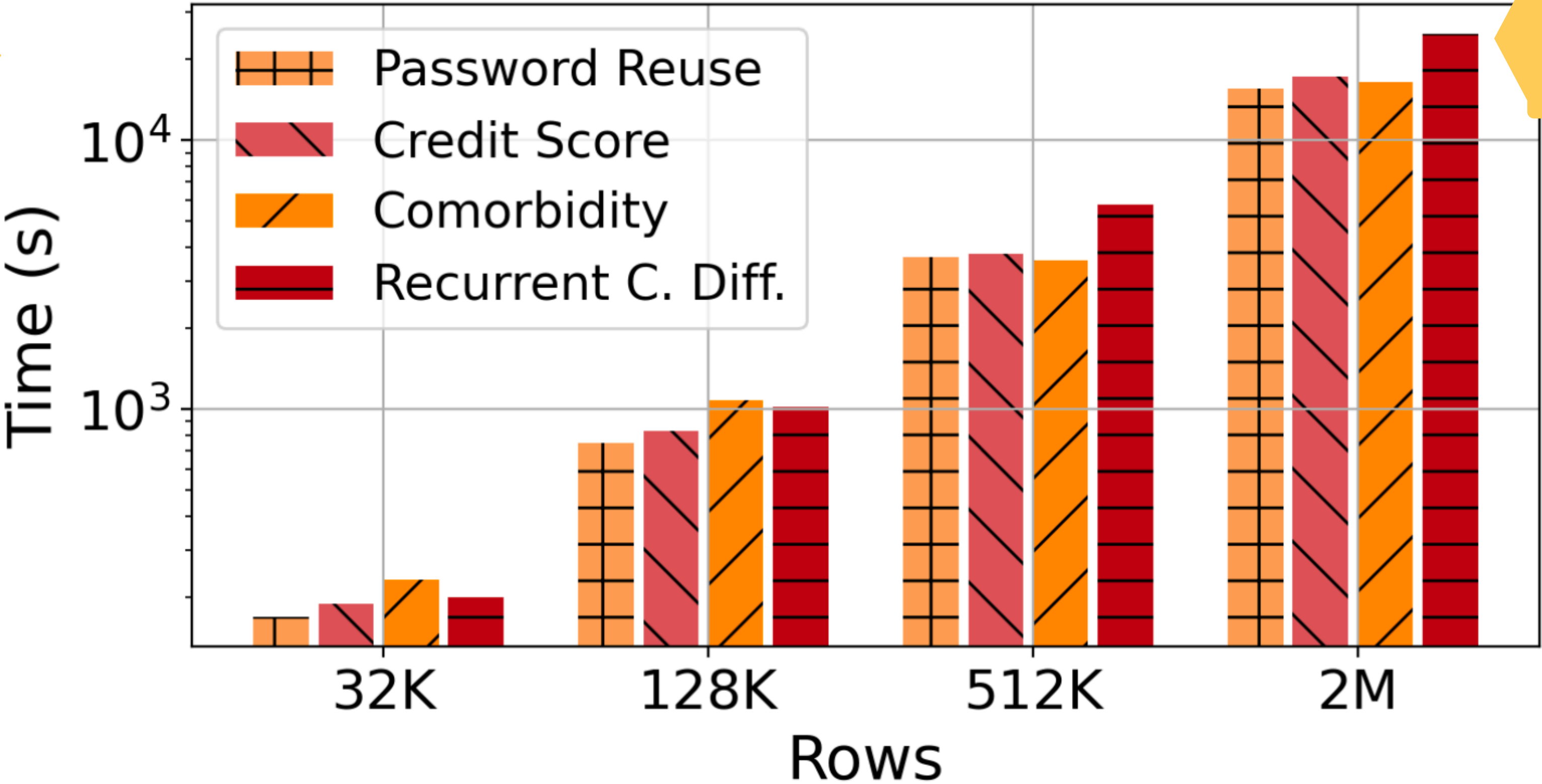
Requires limited communication rounds that do not depend on the input size



Scales comfortably to millions of input rows

# SCALING BEHAVIOR: CATEGORY B

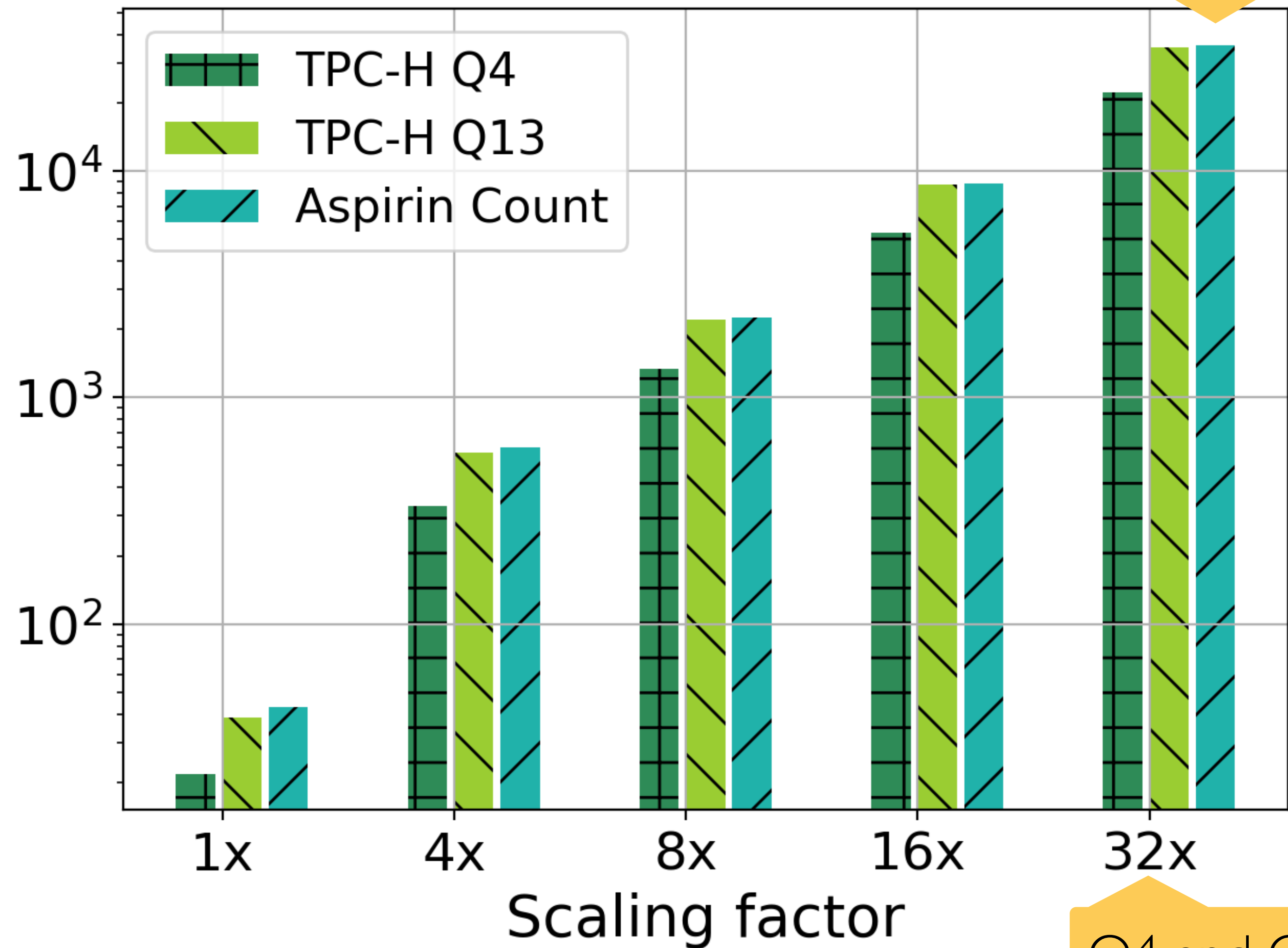
The cost of these queries is dominated by the oblivious GROUP-BY and DISTINCT operators



Scale to millions of input rows but with higher execution times

# SCALING BEHAVIOR: CATEGORY C

The cost of these queries is dominated by the oblivious JOIN and SEMI-JOIN operators



Aspirin Count scales to 32K rows per input

Q4 and Q13 scale to 164K and 300K rows respectively



---

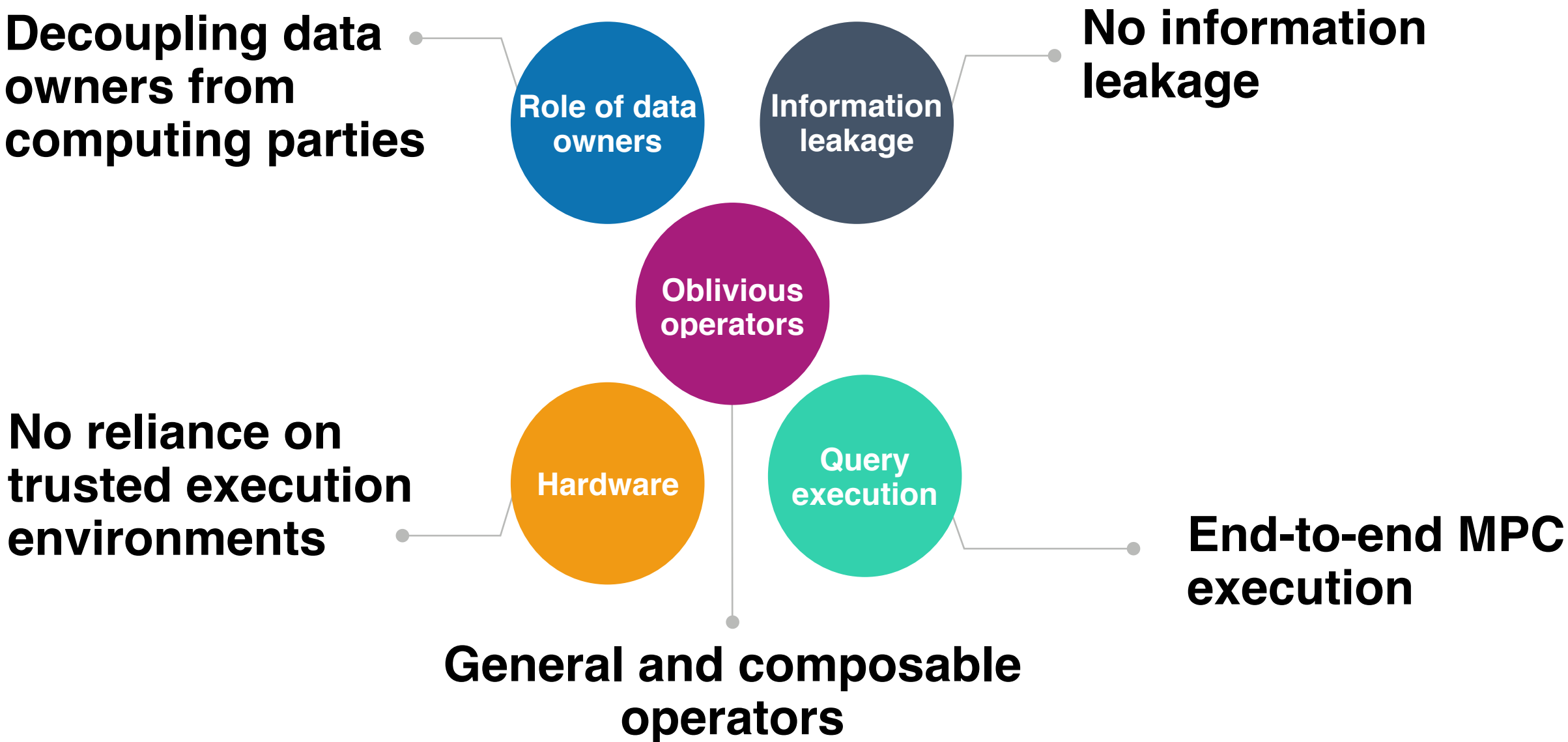
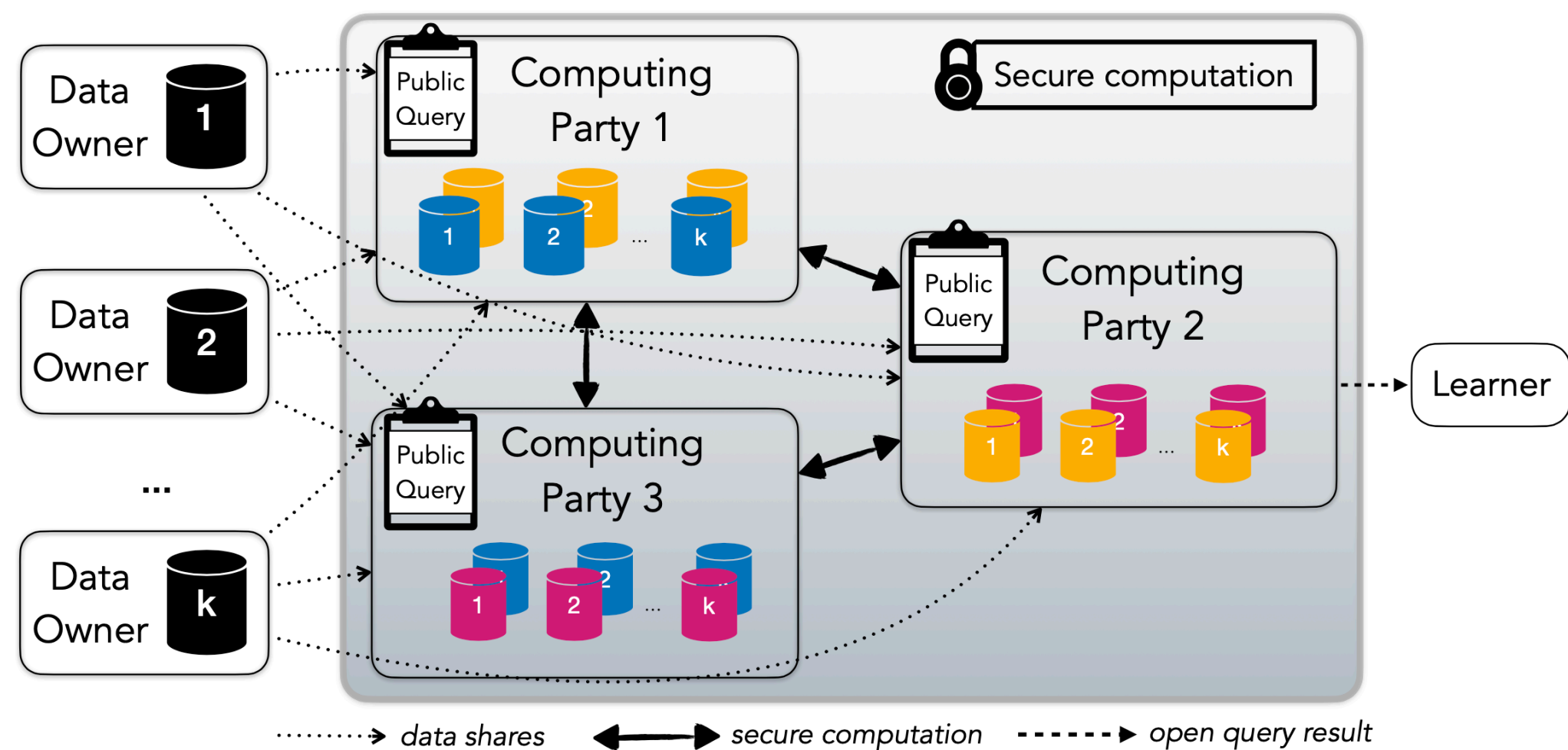
# Future Work

---

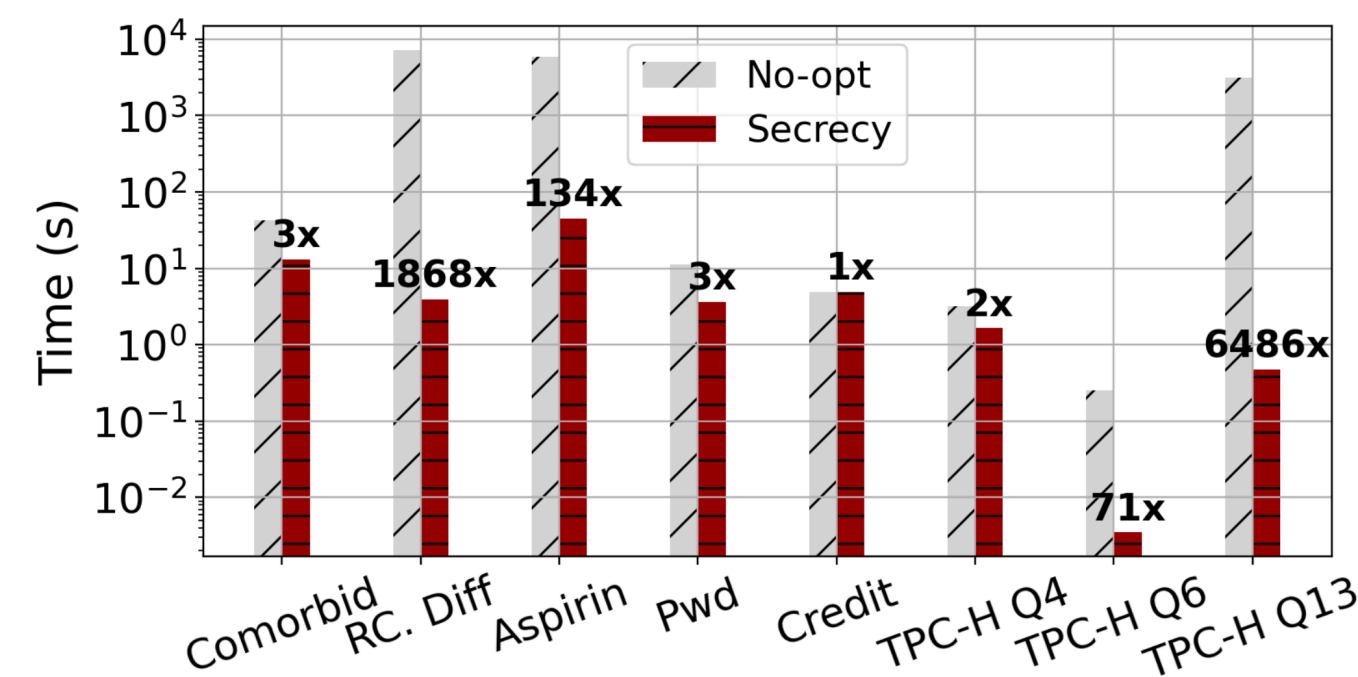
# WHAT'S NEXT?

- Building robust MPC query optimizers
- Add support for task- and data-parallelism (ongoing)
- Design efficient oblivious relational operators
- Hardware acceleration for MPC
- Malicious security (ongoing)
- ...

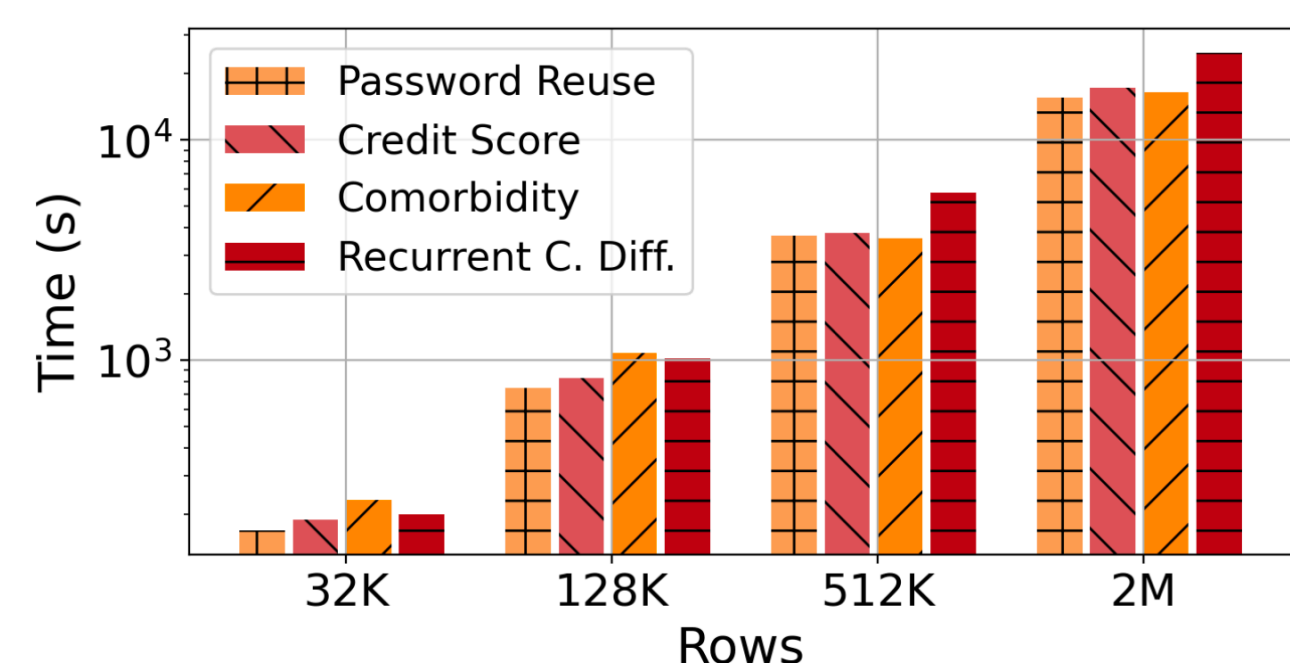
# SECRECY SUMMARY



Up to 1000x speedups in real and synthetic queries



Up to millions of input rows entirely under MPC



Scales to much larger inputs compared to prior works

