Lazy Join Optimizations without Upfront Statistics

MATTEO INTERLANDI

SPARK SUMMIT 2017

Cloud Computing Programs

Open Source Data-Intensive Scalable Computing (DISC) Platforms: Hadoop MapReduce and Spark

- functional API
- **map** and **reduce** User-Defined Functions
- RDD transformations (**filter**, **flatMap**, **zipPartitions**, etc.)

Several years later, introduction of high-level SQL-like declarative query languages (and systems)

- Conciseness
- Pick a physical execution plan from a number of alternatives

Query Optimization

Two steps process

- **Logical** optimizations (e.g., filter pushdown)
- **Physical** optimizations (e.g., join orders and implementation)

Physical optimizer in RDMBS:

- **Cost-base**
- **Data statistics** (e.g., predicate selectivities, cost of data access, etc.)

The role of the cost-based optimizer is to

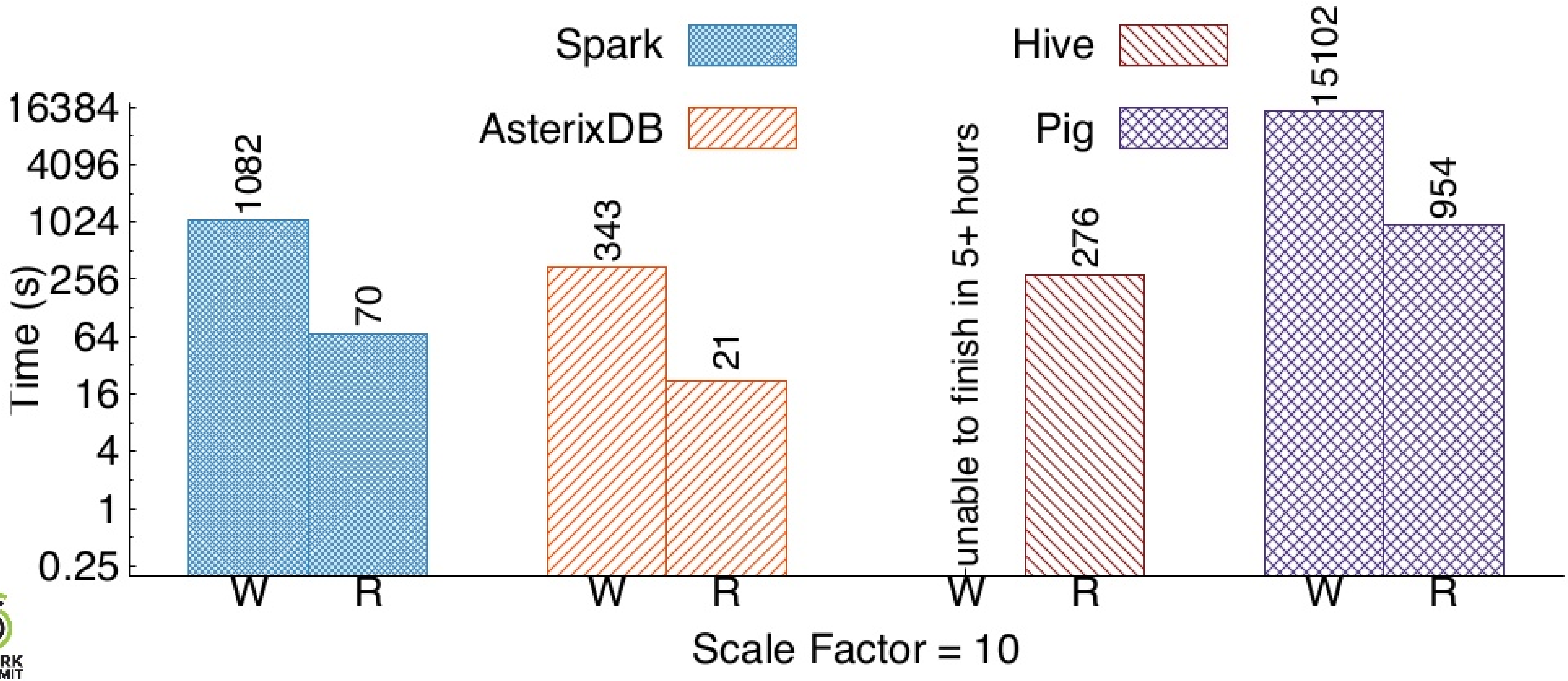
(1) enumerate some set of equivalent plans

(2) estimate the cost of each plan

(3) select a sufficiently good plan

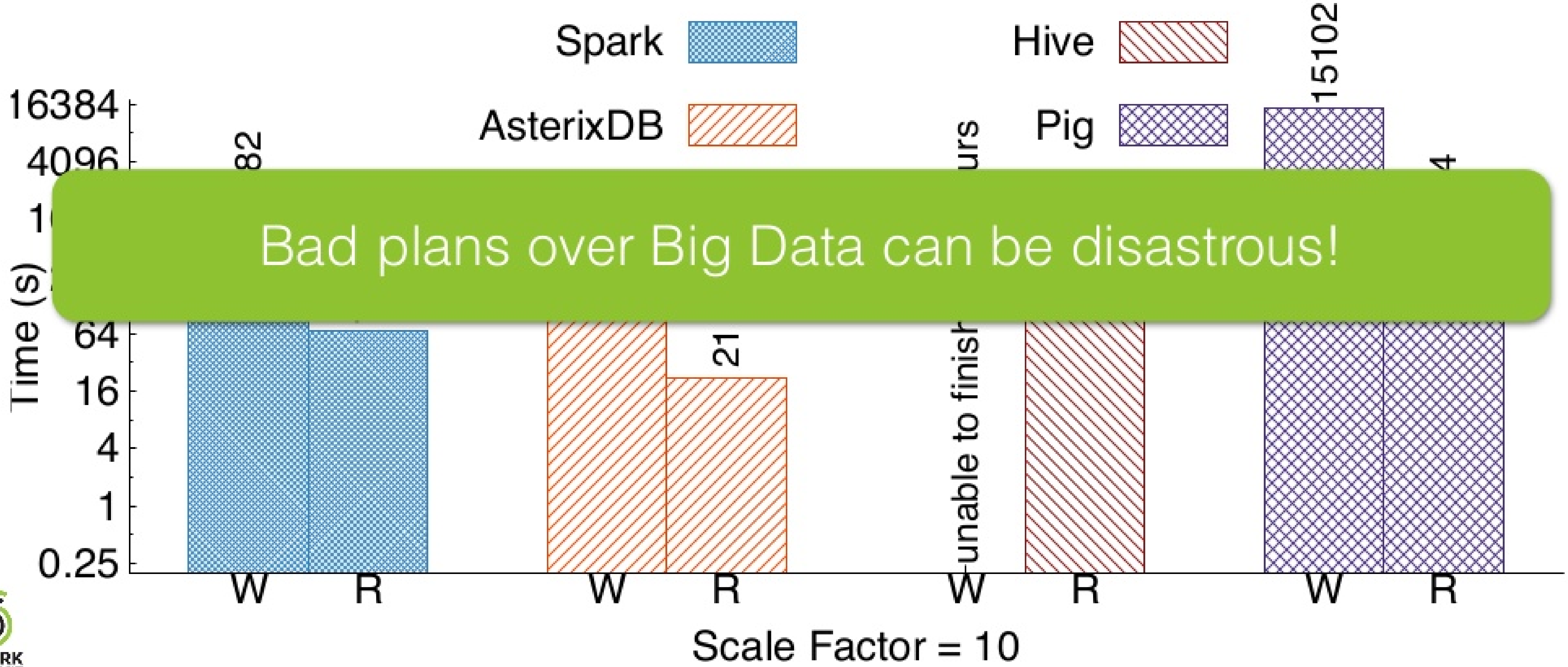
SPARK SUMMIT 2017

Query Optimization: Why Important?



SPARK SUMMIT 2017

Query Optimization: Why Important?



Bad plans over Big Data can be disastrous!

Cost-base Optimizer in DISC

No **cost-based join enumeration**
- Rely on order of relations in FROM clause
- Left-deep plans

No **upfront statistics**:
- Often data sits in HDFS and unstructured

Even if input statistics are available:
- **Correlations** between predicates
- Exponential error propagation in joins
- Arbitrary **UDFs**

Cost-base Optimizer in DISC: State of the Art

Bad statistics

No upfront statistics

Cost-base Optimizer in DISC: State of the Art

Bad statistics

- Adaptive Query planning
- RoPe [NSDI 12, VLDB 2013]

No upfront statistics

Cost-base Optimizer in DISC: State of the Art

Bad statistics

- Adaptive Query planning
- RoPe [NSDI 12, VLDB 2013]

Assumption is that some initial statistics exist

No upfront statistics

SPARK SUMMIT 2017

Cost-base Optimizer in DISC: State of the Art

Bad statistics

- Adaptive Query planning
- RoPe [NSDI 12, VLDB 2013]

No upfront statistics

- Pilot runs (samples)
- DynO [SIGMOD 2014]

Assumption is that some initial statistics exist

Cost-base Optimizer in DISC: State of the Art

Bad statistics

- Adaptive Query planning
- RoPe [NSDI 12, VLDB 2013]

Assumption is that some initial statistics exist

No upfront statistics

- Pilot runs (samples)
- DynO [SIGMOD 2014]

- Samples are expensive
- Only foreign-key joins
- No runtime plan revision

Lazy Cost-base Optimizer for Spark

Key idea: interleave query **planning** and **execution**

- Query plans are **lazily** executed
- Statistics are **gathered at runtime**
- Joins are **greedly** scheduled
- Next join can be **dynamically changed** if a **bad decision** was made
- Execute-Gather-Aggregate-Plan strategy (EGAP)

Neither upfront statistics nor pilot runs are required

- Raw dataset size for initial guess

Support for not foreign-key joins

Lazy Optimizer: an Example

$A \boxed{\bowtie} B \boxed{\bowtie} C$

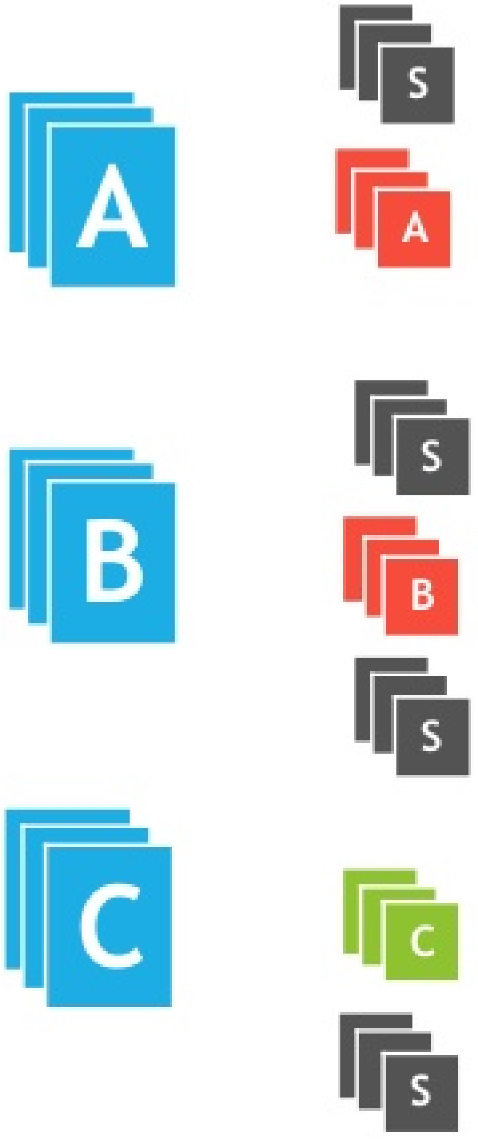
Assumption: A < C

Lazy Optimizer: Execute Step

$A \boxed{\bowtie} B \boxed{\bowtie} C$

Assumption: A < C

Lazy Optimizer: Gather step



$A \boxed{\bowtie} B \boxed{\bowtie} C$

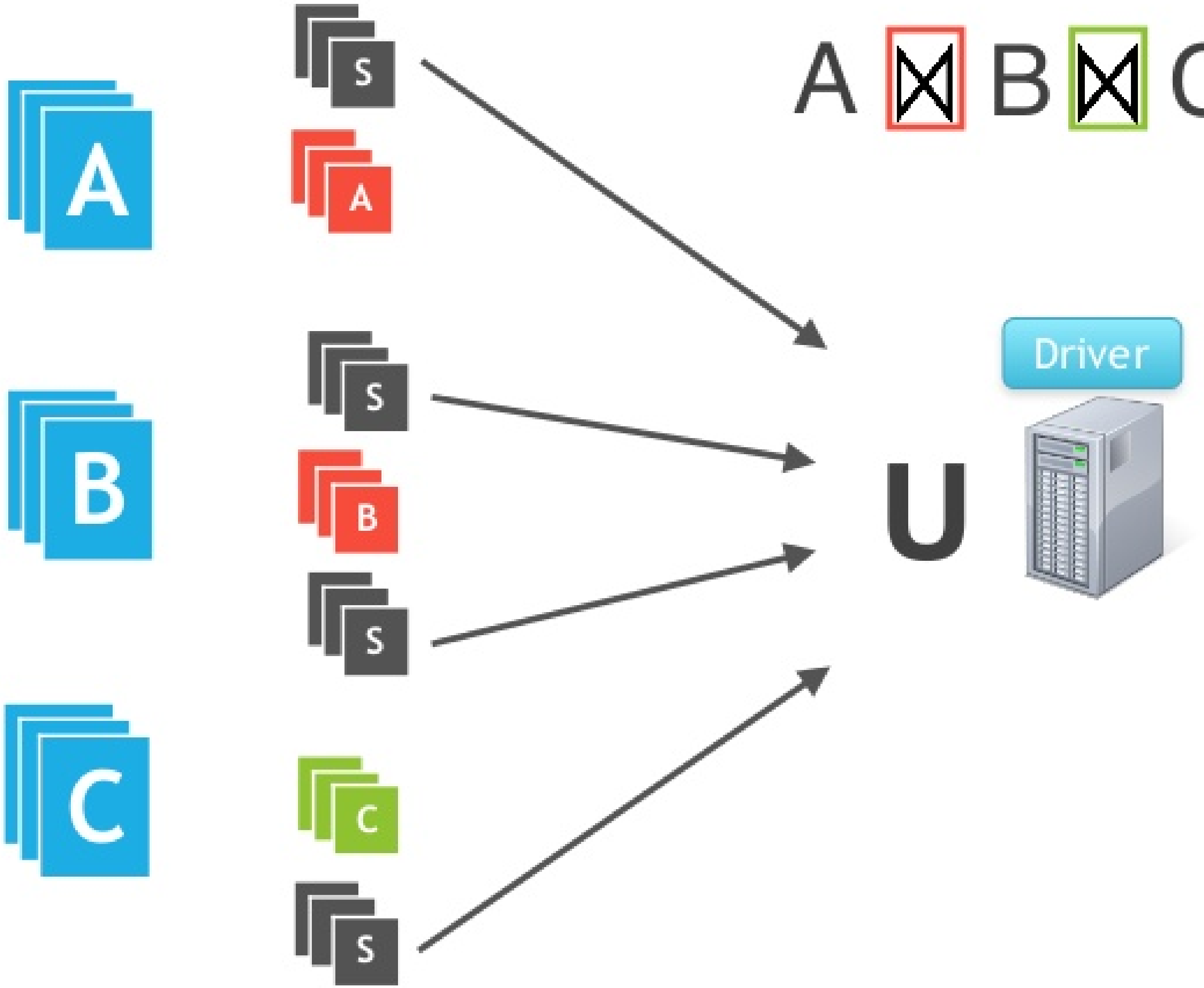
Assumption: A < C

SPARK SUMMIT 2017

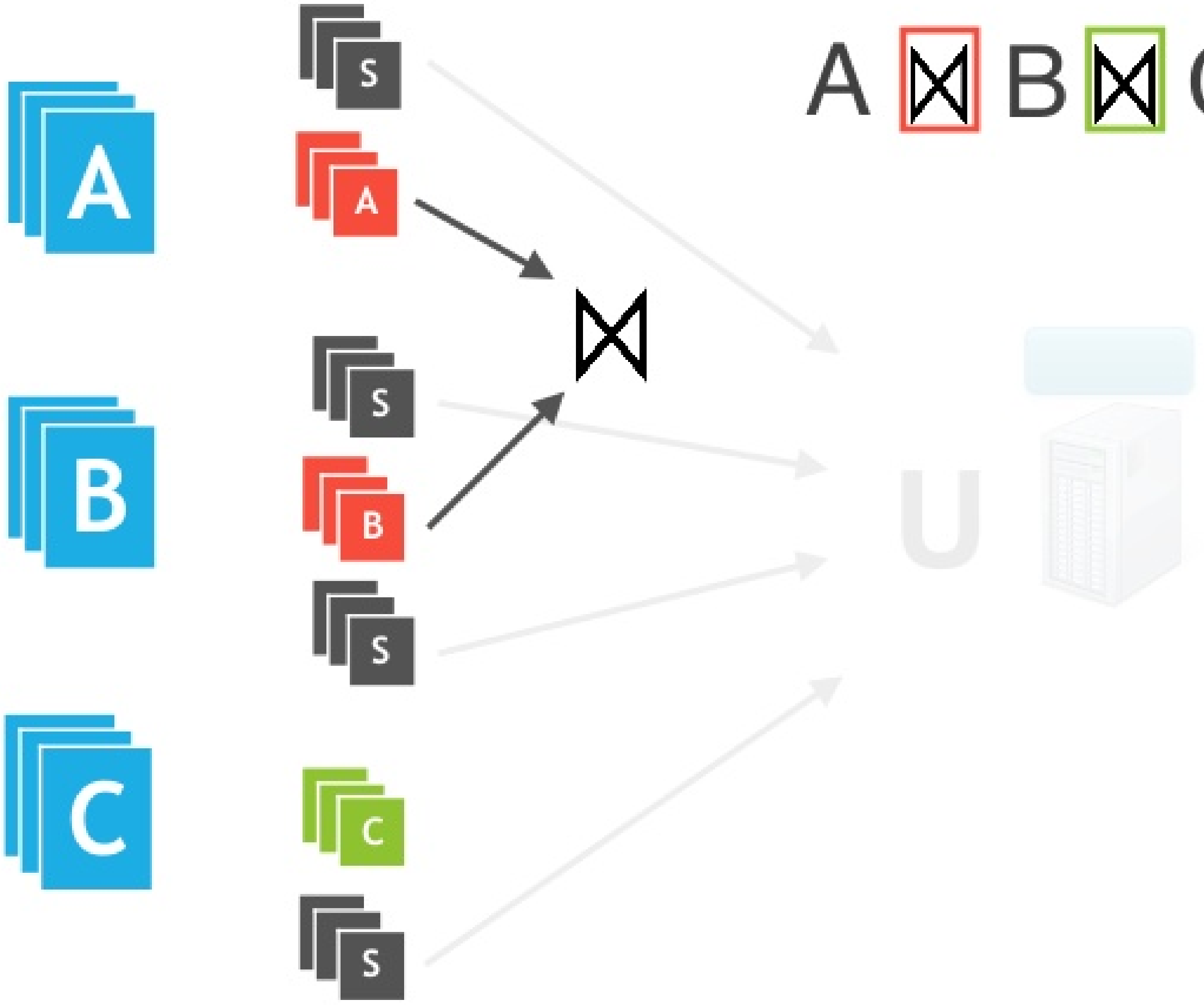
Lazy Optimizer: Aggregate step

$A \boxed{\bowtie} B \boxed{\bowtie} C$

Assumption: A < C



Lazy Optimizer: Plan step



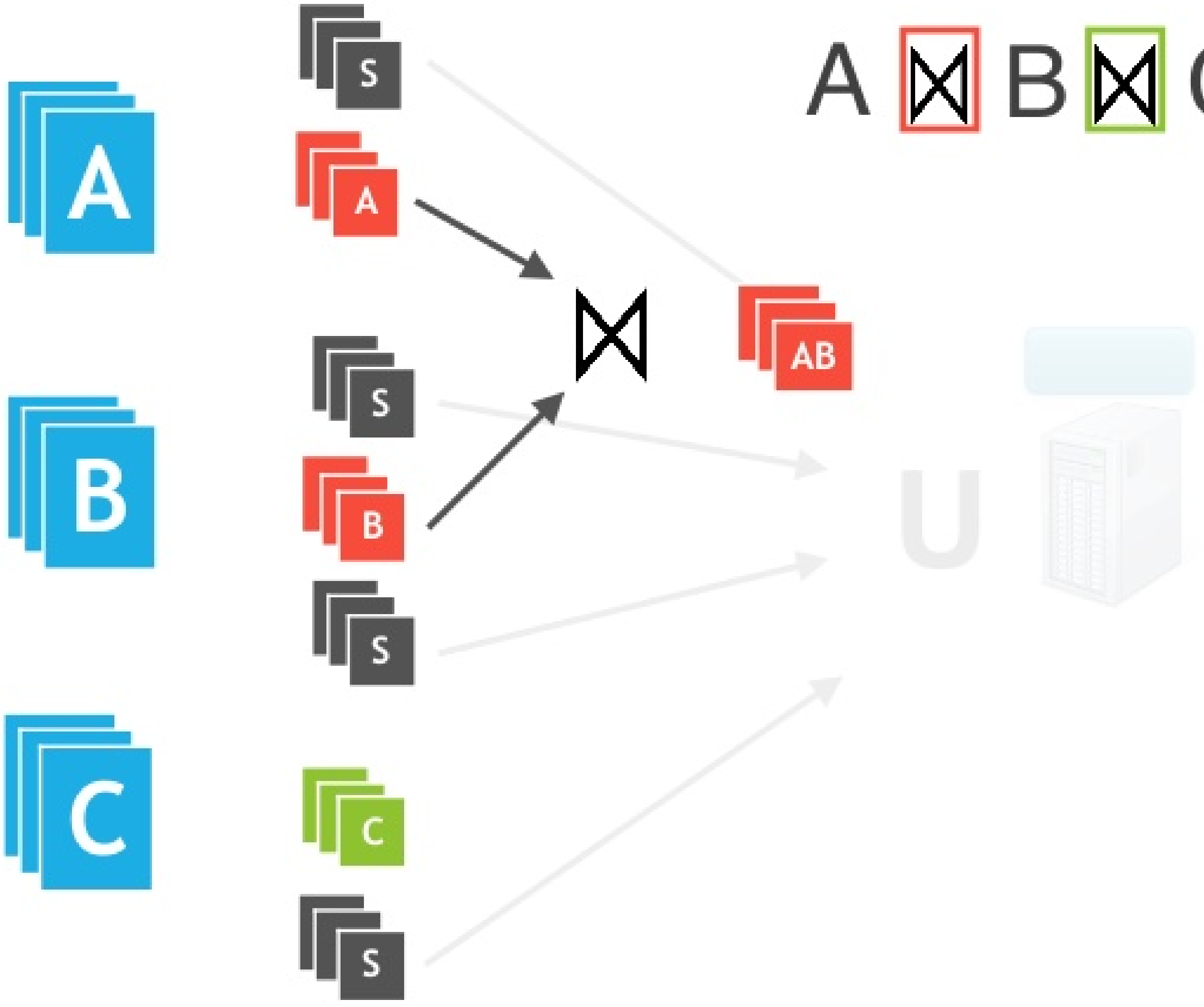
$A \bowtie B \bowtie C$

Assumption: A < C

Lazy Optimizer: Execute step

$A \bowtie B \bowtie C$

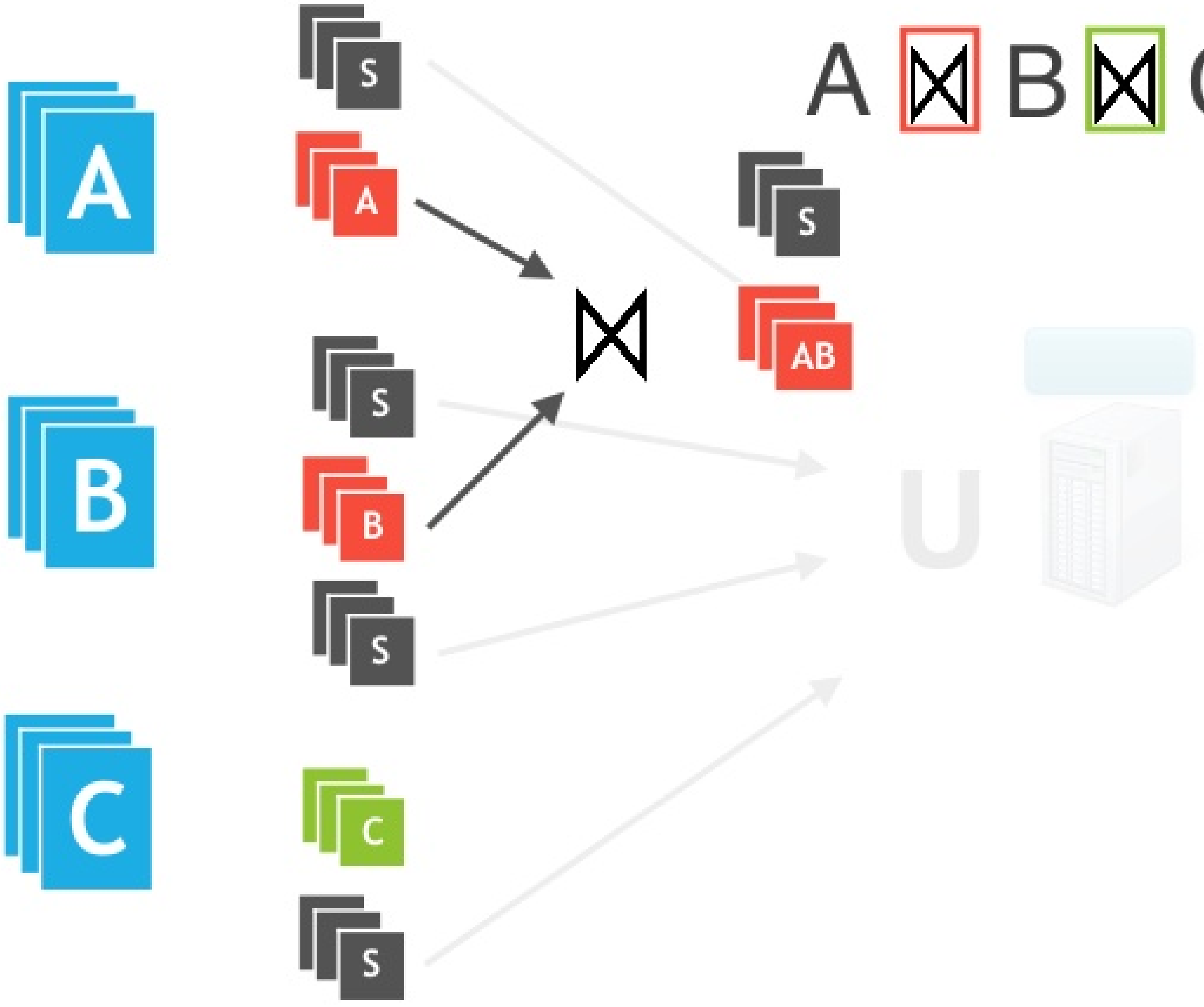
Assumption: A < C



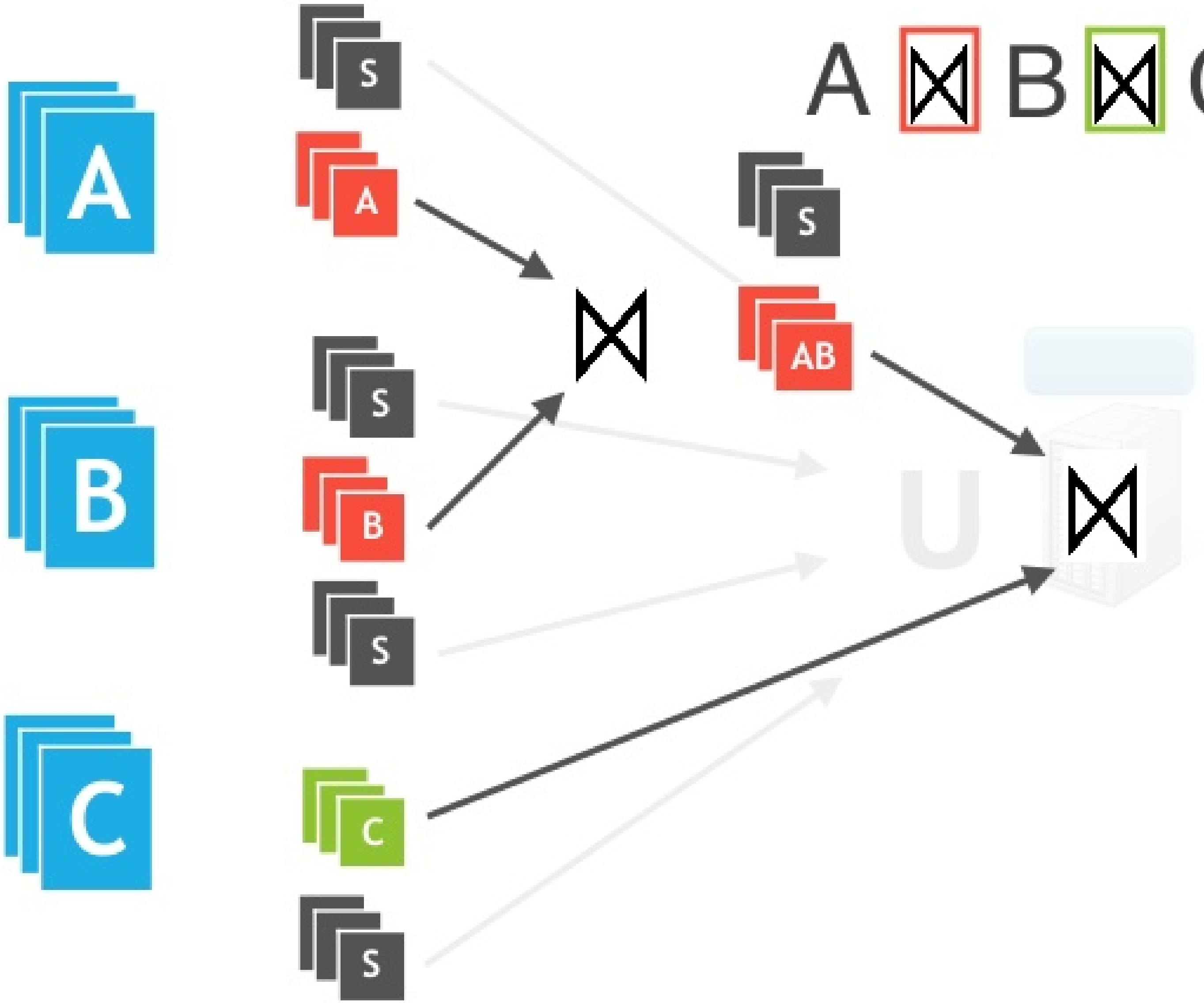
Lazy Optimizer: Gather step

$A \bowtie B \bowtie C$

Assumption: A < C



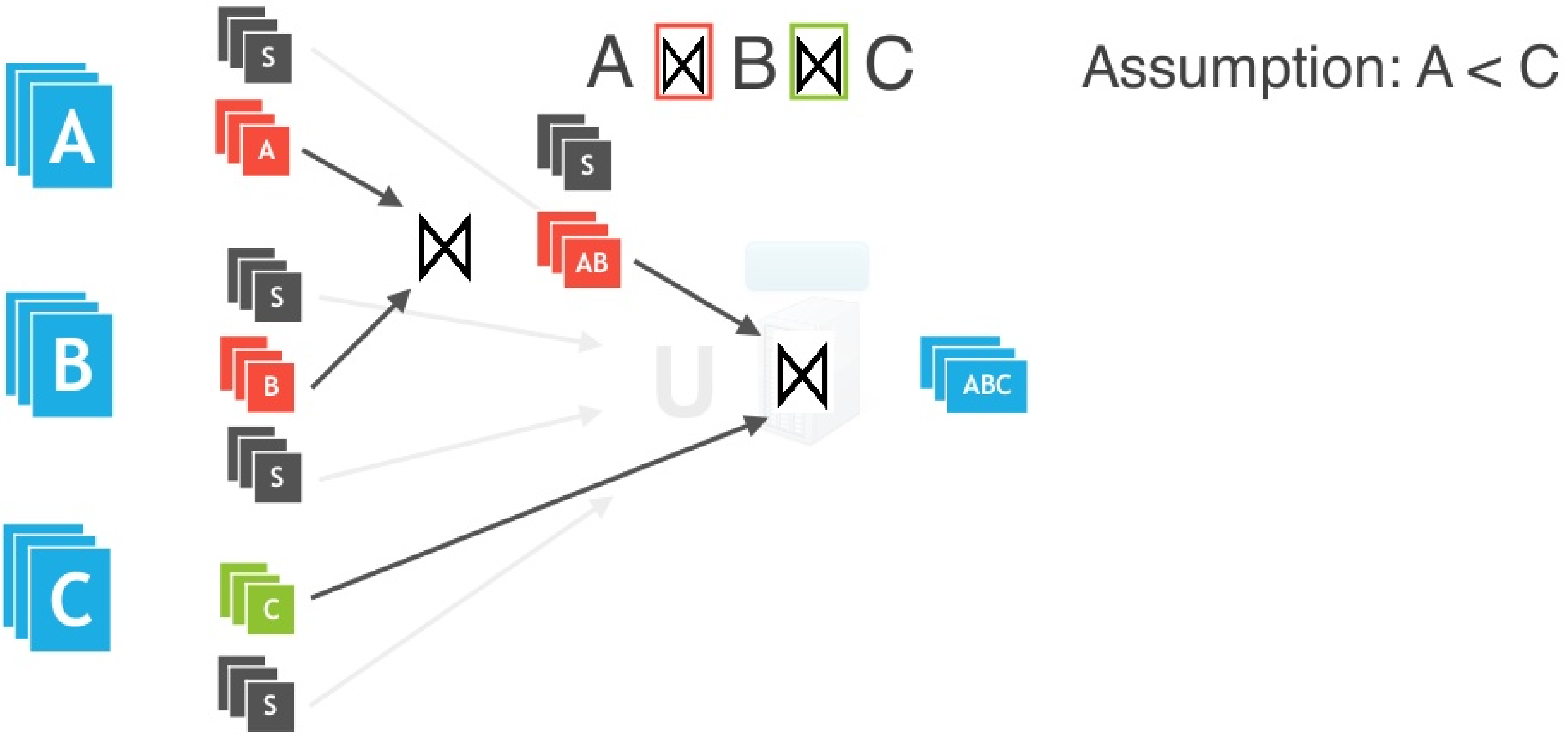
Lazy Optimizer: Plan step



$A \bowtie B \bowtie C$

Assumption: A < C

Lazy Optimizer: Execute step



Lazy Optimizer: Wrong Guess

$\sigma(A) \boxed{\bowtie} B \boxed{\bowtie} \sigma(C)$

Assumption: $A < C$

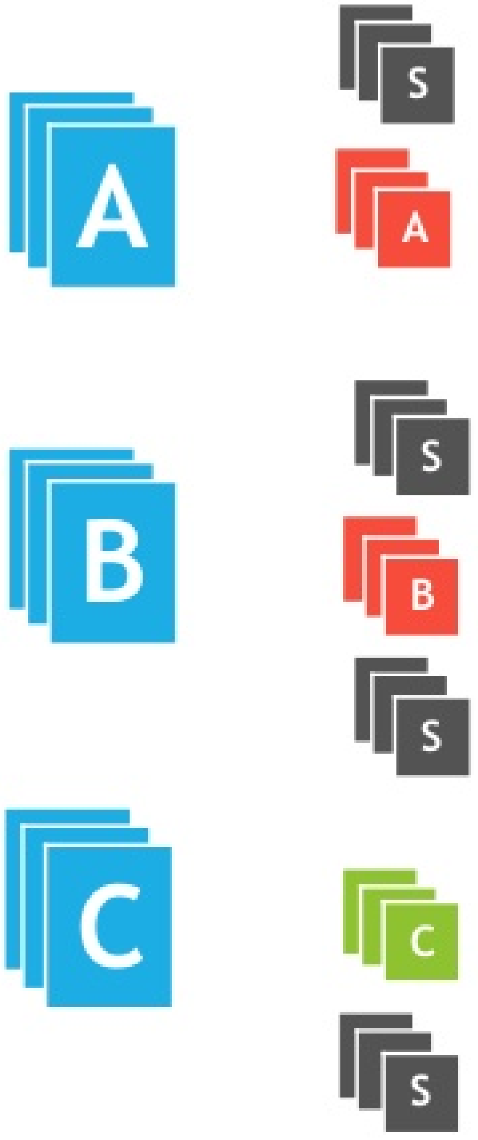
$\sigma(A) > \sigma(C)$

A

B

C

Lazy Optimizer: Wrong Guess



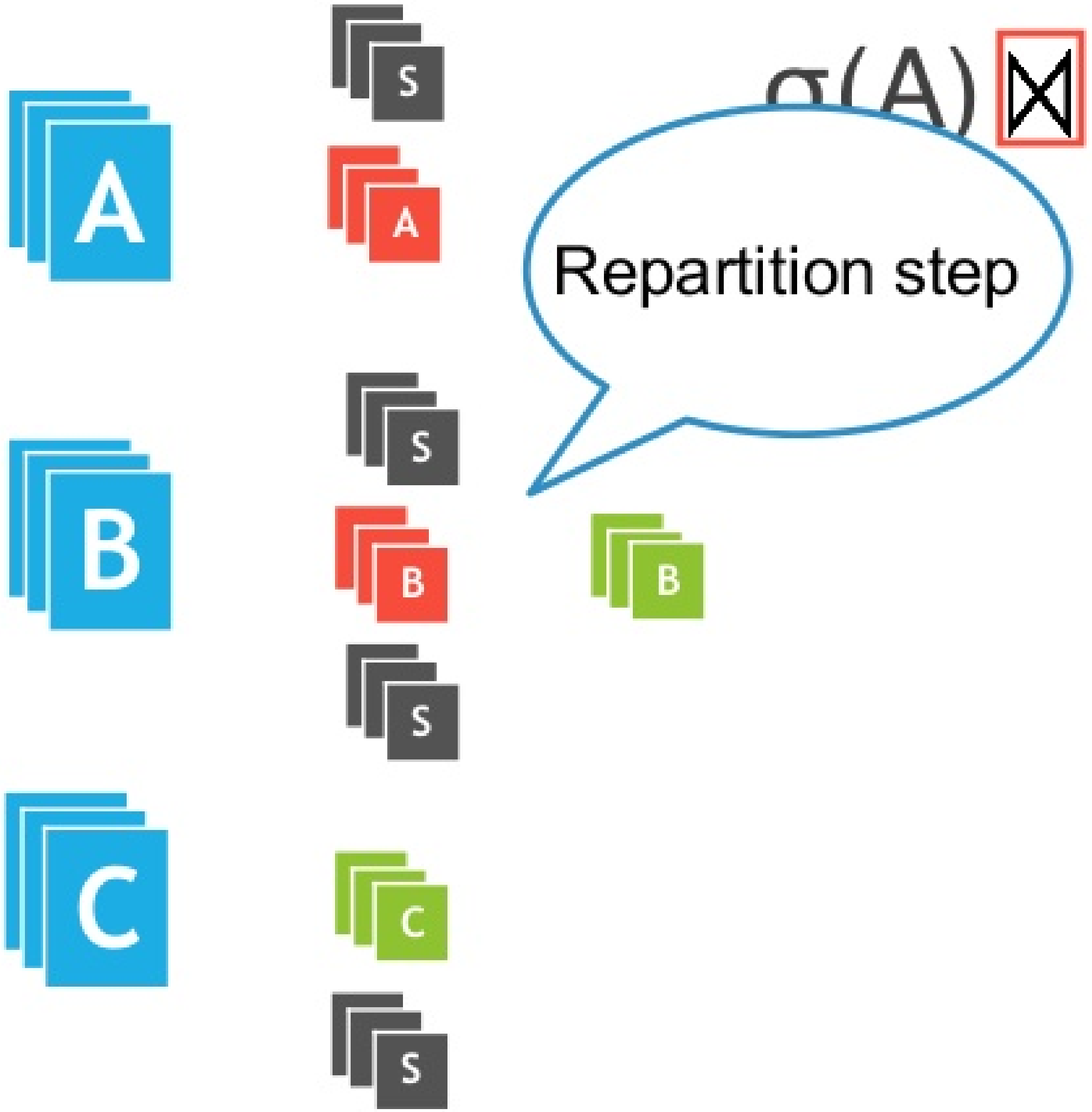
$$\sigma(A) \boxed{\bowtie} B \boxed{\bowtie} \sigma(C)$$

Assumption: $A < C$

$\sigma(A) > \sigma(C)$

SPARK SUMMIT 2017

Lazy Optimizer: Wrong Guess



$\sigma(A) \bowtie B \bowtie \sigma(C)$

Assumption: A < C

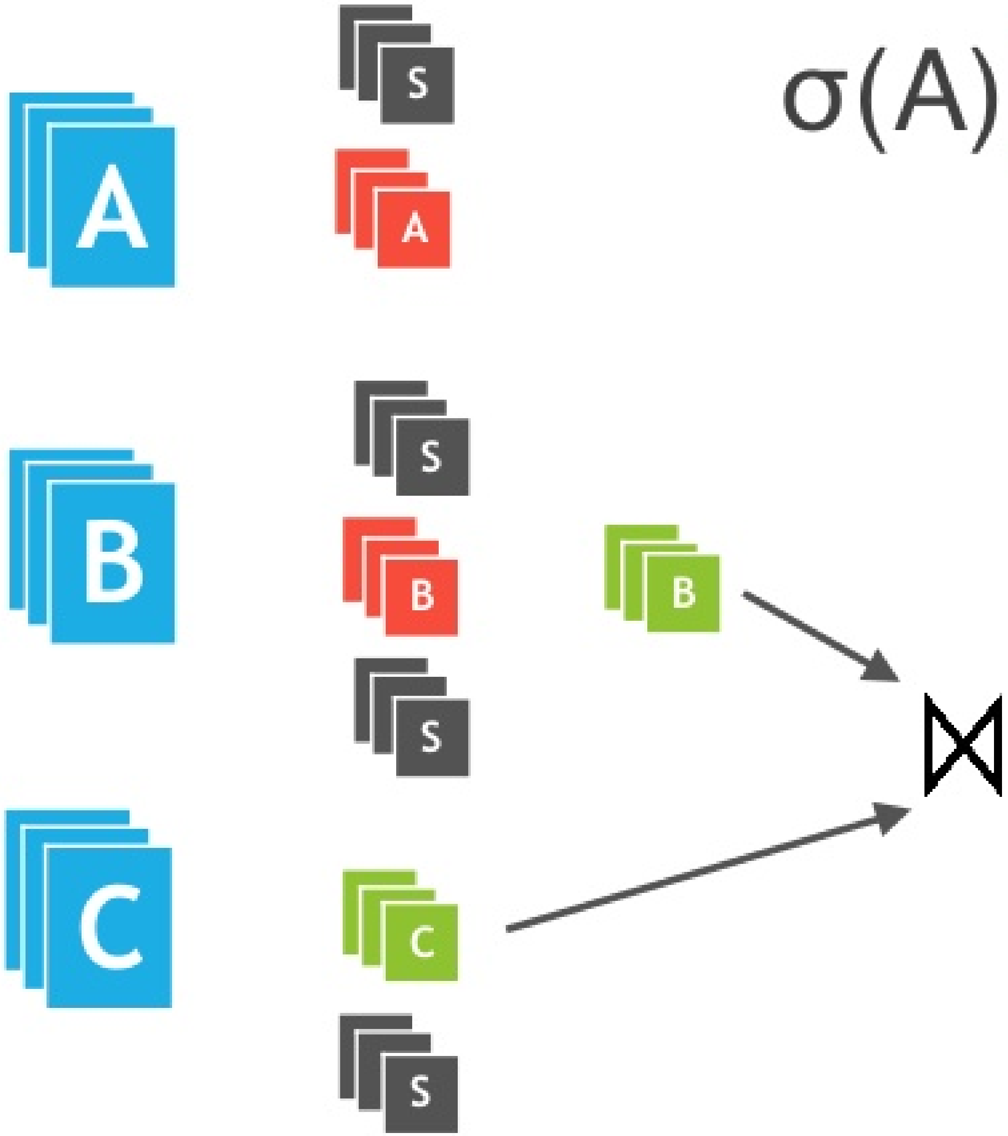
$\sigma(A) > \sigma(C)$

Lazy Optimizer: Wrong Guess

$\sigma(A) \boxed{\bowtie} B \boxed{\bowtie} \sigma(C)$

Assumption: A < C

$\sigma(A) > \sigma(C)$



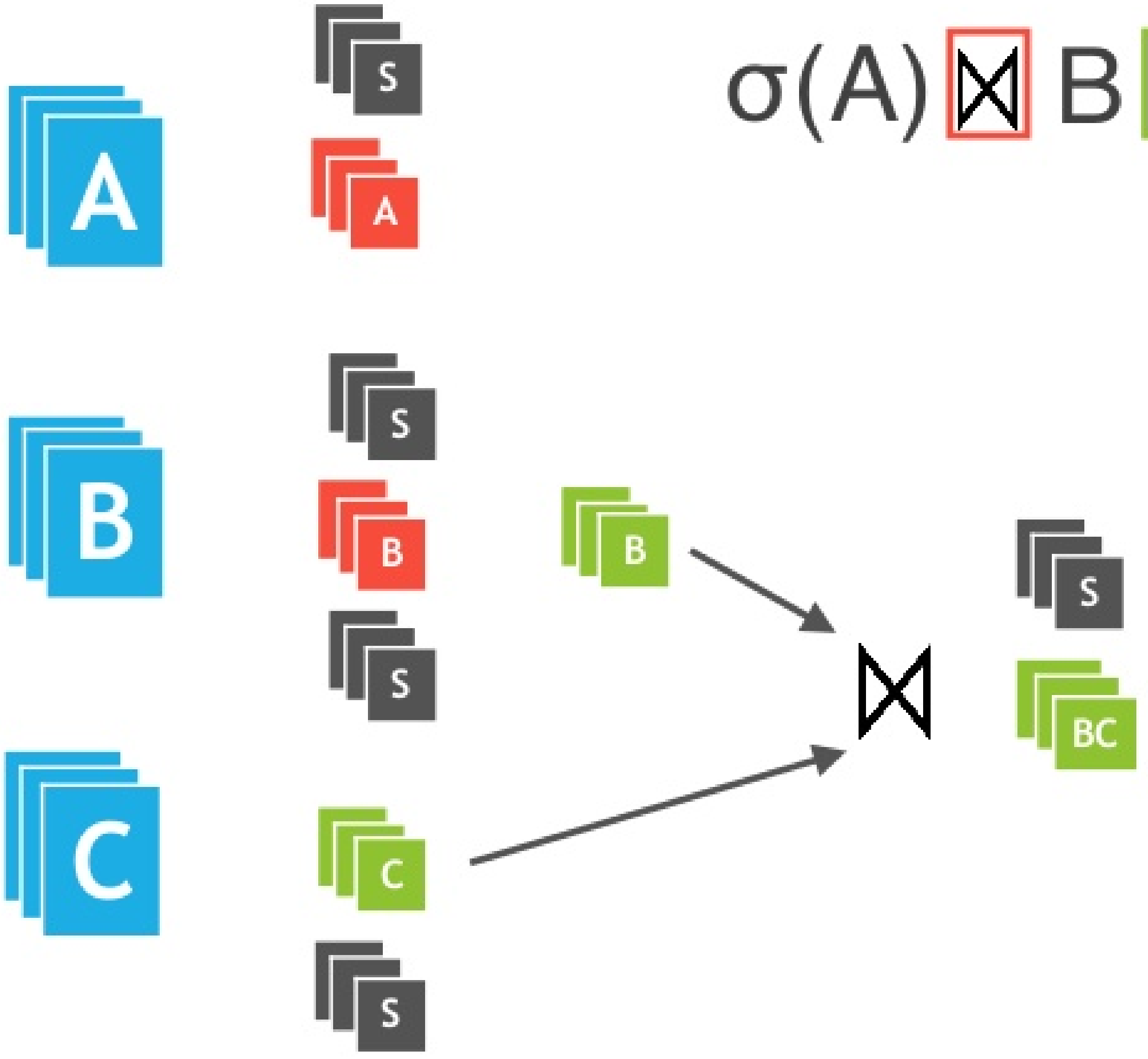
SPARK SUMMIT 2017

Lazy Optimizer: Wrong Guess

$\sigma(A) \bowtie B \bowtie \sigma(C)$

Assumption: A < C

$\sigma(A) > \sigma(C)$

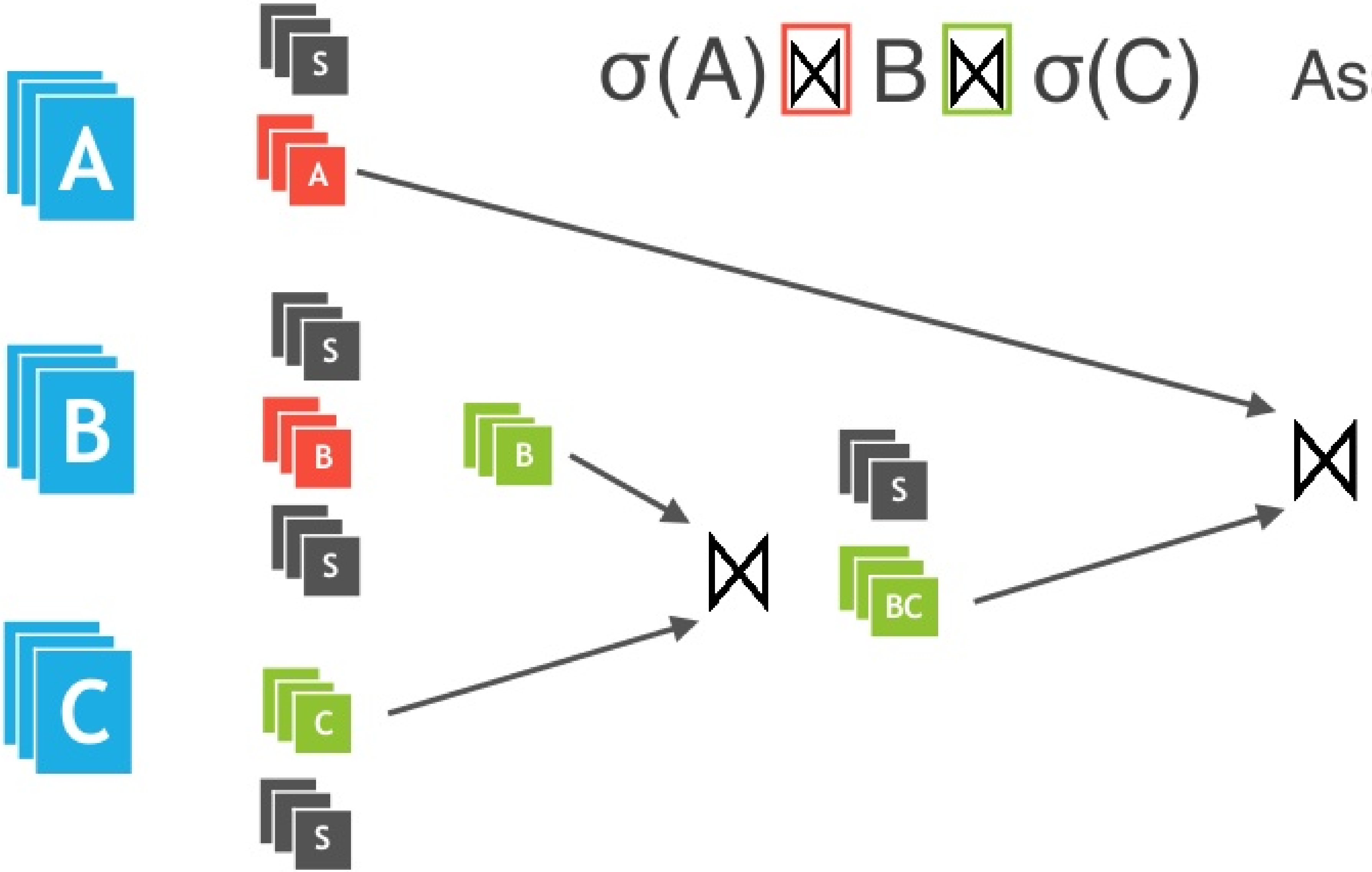


Lazy Optimizer: Wrong Guess

$\sigma(A) \boxed{\bowtie} B \boxed{\bowtie} \sigma(C)$

Assumption: A < C

$\sigma(A) > \sigma(C)$



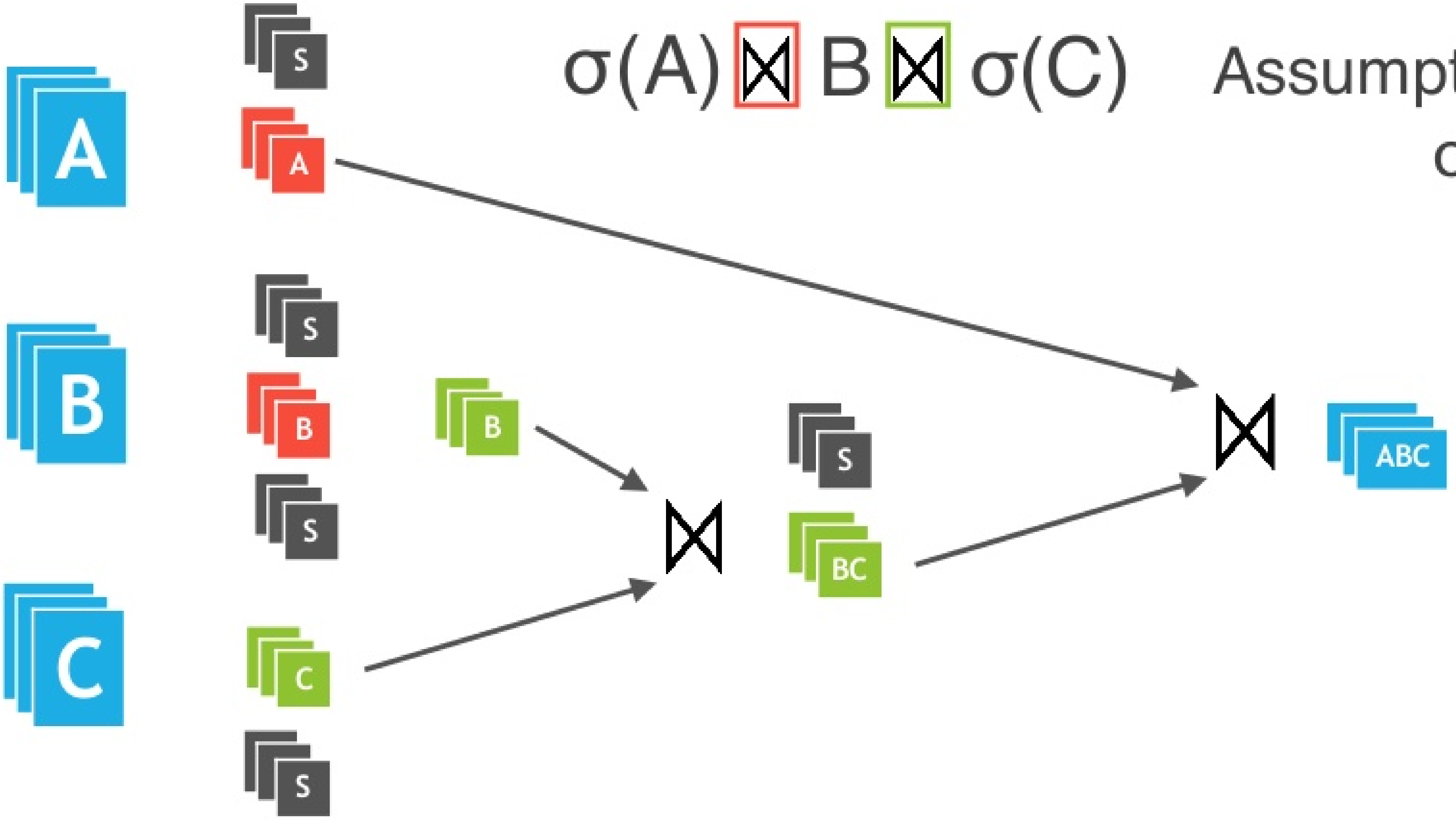
SPARK SUMMIT 2017

Lazy Optimizer: Wrong Guess

$\sigma(A) \bowtie B \bowtie \sigma(C)$

Assumption: A < C

$\sigma(A) > \sigma(C)$



SPARK SUMMIT 2017

Runtime Integrated Optimizer for Spark

Spark **batch execution** model allows late binding of joins

Set of Statistics:

- Join estimations (based on sampling or sketches)
- Number of records
- Average size of each record

Statistics are aggregates using a Spark job or accumulators

Join **implementations** are picked based on thresholds

Challenges and Optimizations

Execute - Block and revise execution plans **without wasting computation**

Gather - **Asynchronous** generation of statistics

Aggregate - Efficient accumulation of statistics

Plan - Try to schedule as many **broadcast joins** as possible

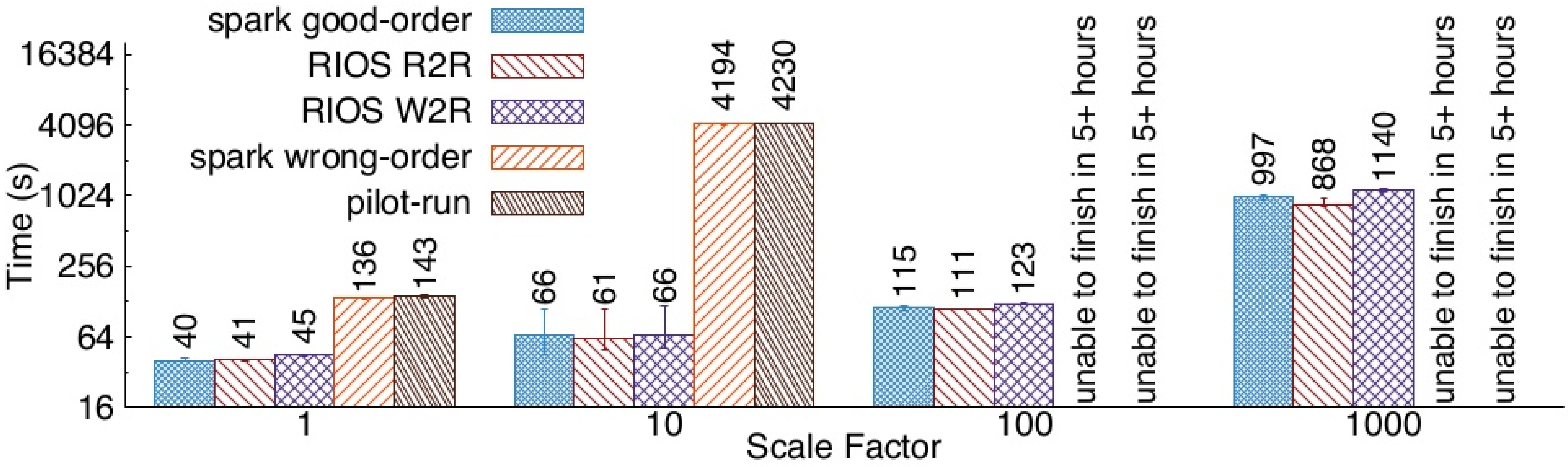
Experiments

Q1: Is RIOS able to generate good query plans?

Q2: What are the performance of RIOS compared to regular Spark and pilot runs?

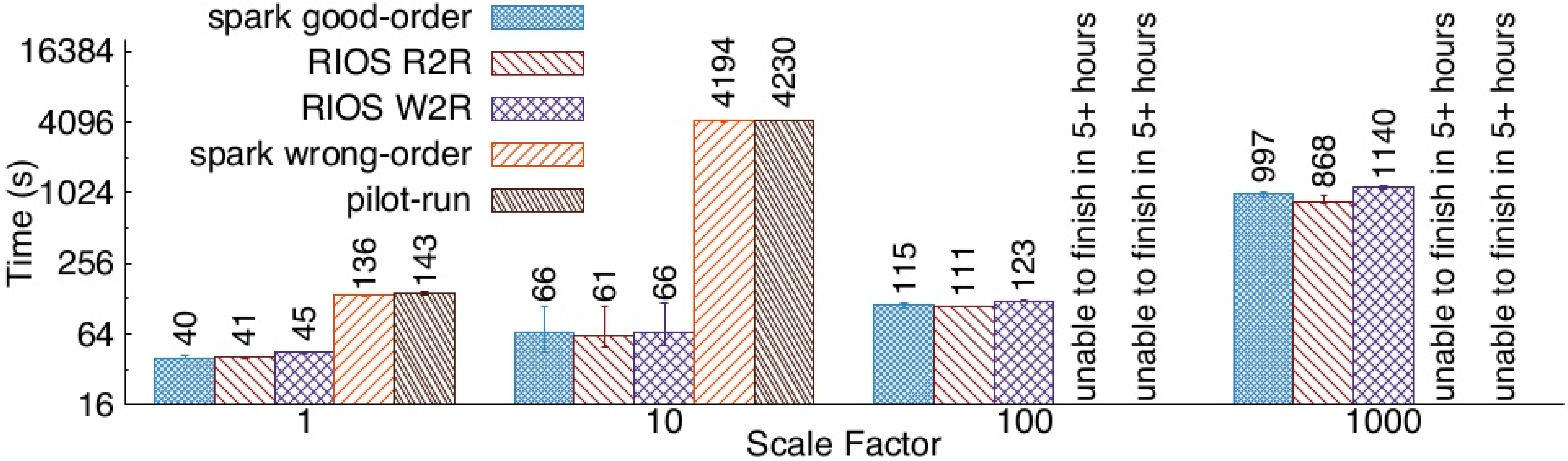
Q3: How expensive are wrong guesses?

Minibenchmark with 3 Fact Tables



SPARK SUMMIT 2017

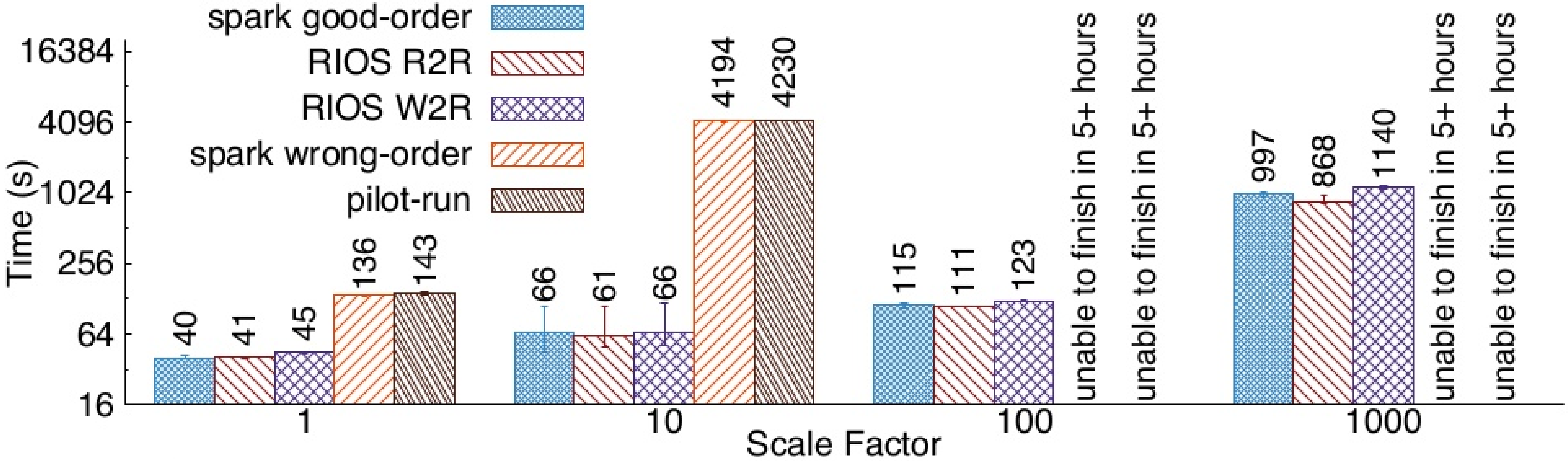
Minibenchmark with 3 Fact Tables



Q1: RIOS is able to avoid bad plans

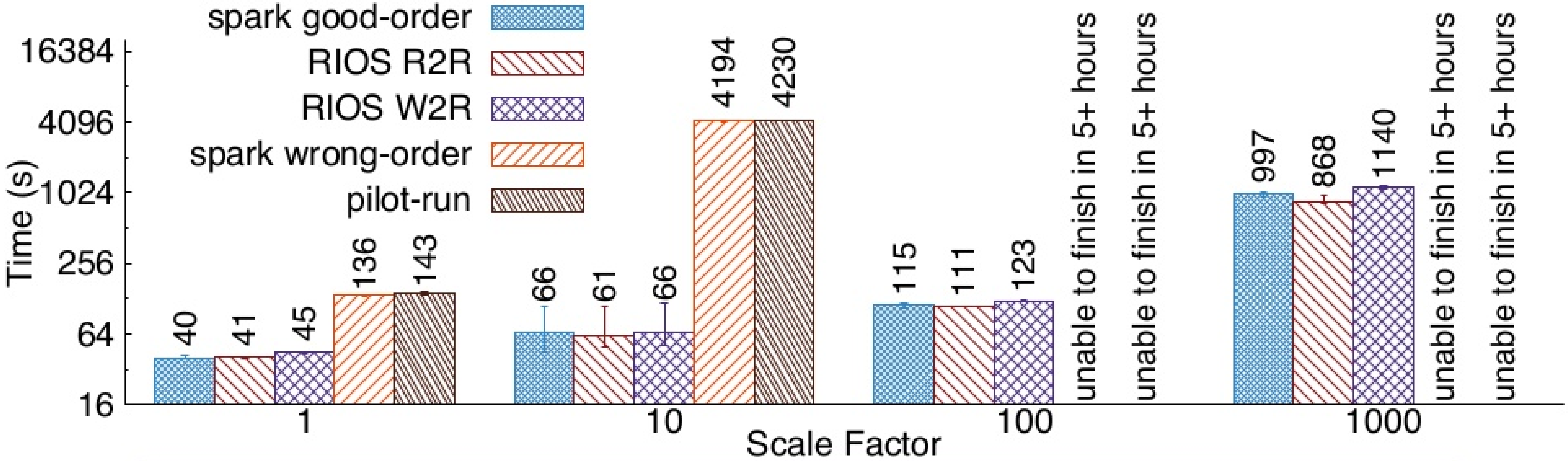
SPARK SUMMIT 2017

Minibenchmark with 3 Fact Tables



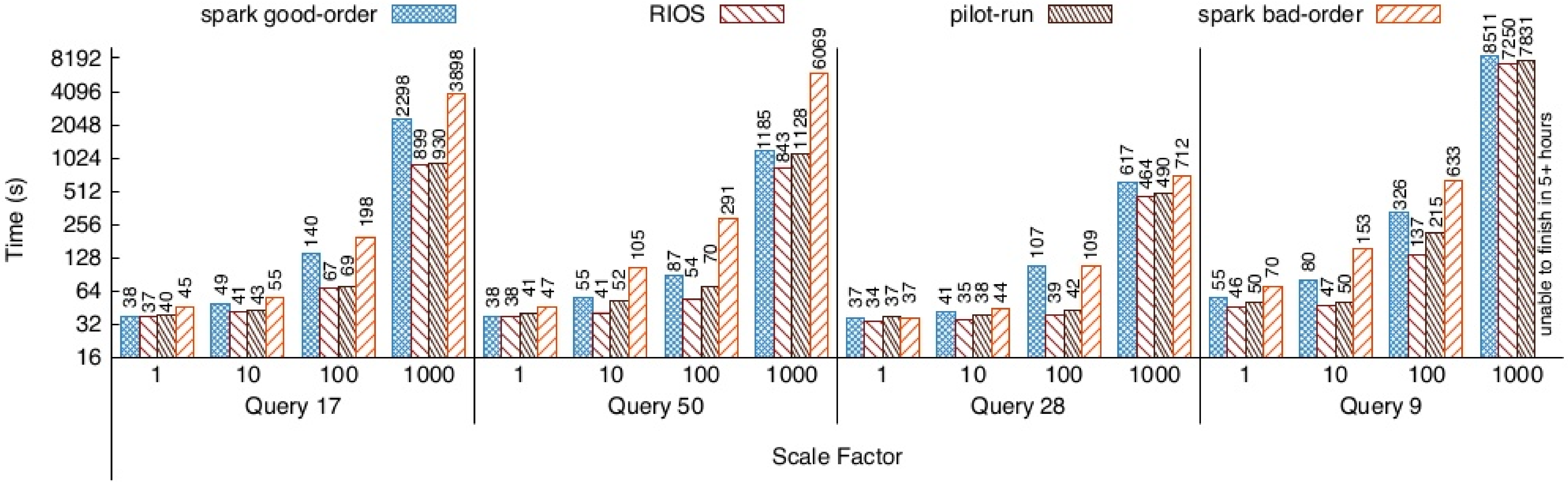
Q2: RIOS is always faster than pilot run approach

Minibenchmark with 3 Fact Tables



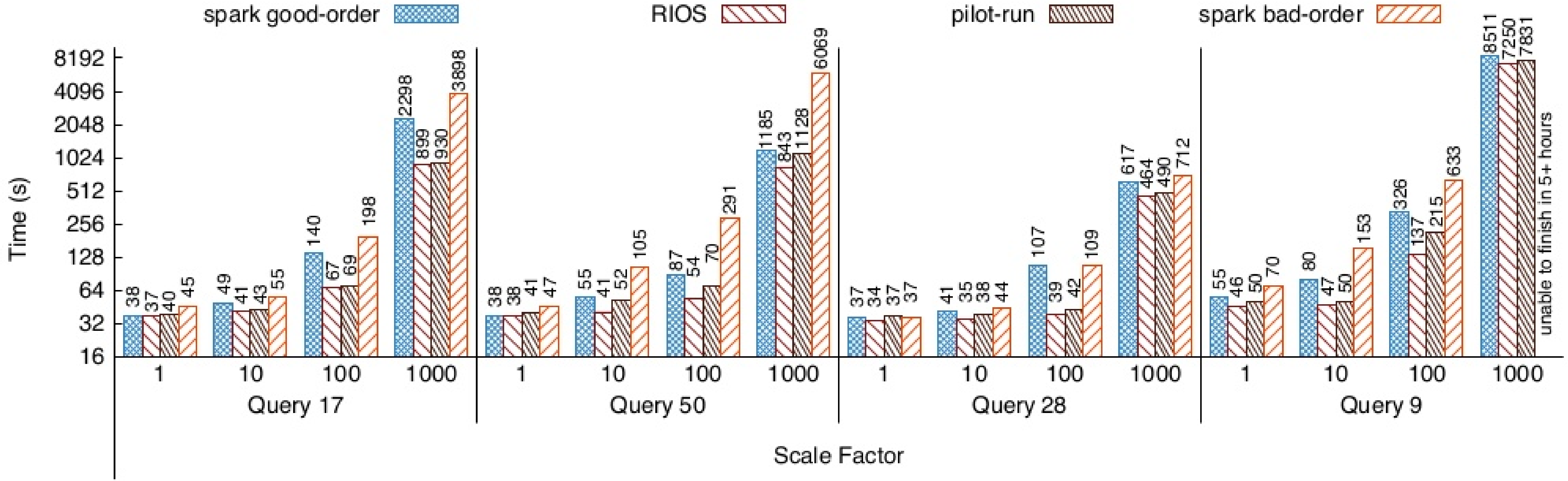
Q3: Bad guesses cost around 15% in the worst case

TPCDS and TPCH Queries



SPARK SUMMIT 2017

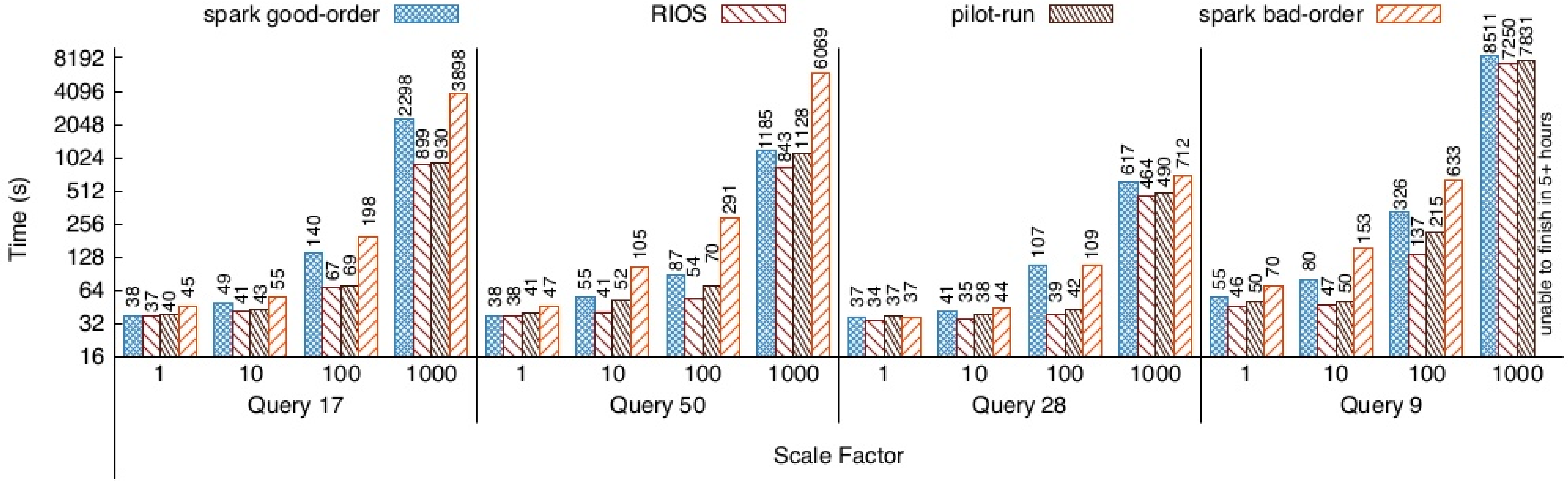
TPCDS and TPCH Queries



Q1: RIOS generates optimal plans

SPARK SUMMIT 2017

TPCDS and TPCH Queries



Q2: RIOS is always the faster approach

Conclusions

RIOS: cost-base query optimizer for Spark

Statistics are **gathered at runtime** (no need for initial statistics or pilot runs)

Late bind of joins

Up to **2x faster** than the best left-deep plans (Spark), and **> 100x** than previous approaches for fact table joins.

Future Work

More **flexible** shuffle operations:

- Efficient switch from shuffle-base joins to broadcast joins
- Allow records to be partitioned in different ways

Take in consideration **interesting orders** and **partitions**

Add aggregation and additional statistics (I\O and network cost)

SPARK SUMMIT 2017

SPARK SUMMIT 2017

Thank you

Experiment Configuration

- Datasets:
  - TPCDS
  - TPCH
- Configuration:
  - 16 machines, 4 cores (2 hyper threads per core) machines, 32GB of RAM, 1TB disk
  - Spark 1.6.3
  - Scale factor from 1 to 1000 (~1TB)