ACM Symposium on Cloud Computing

RIOS: Runtime Integrated Query Optimizer for Spark

Youfu Li(UCLA), Mingda Li(UCLA), Ling Ding(UCLA) and Matteo Interlandi(Microsoft)

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

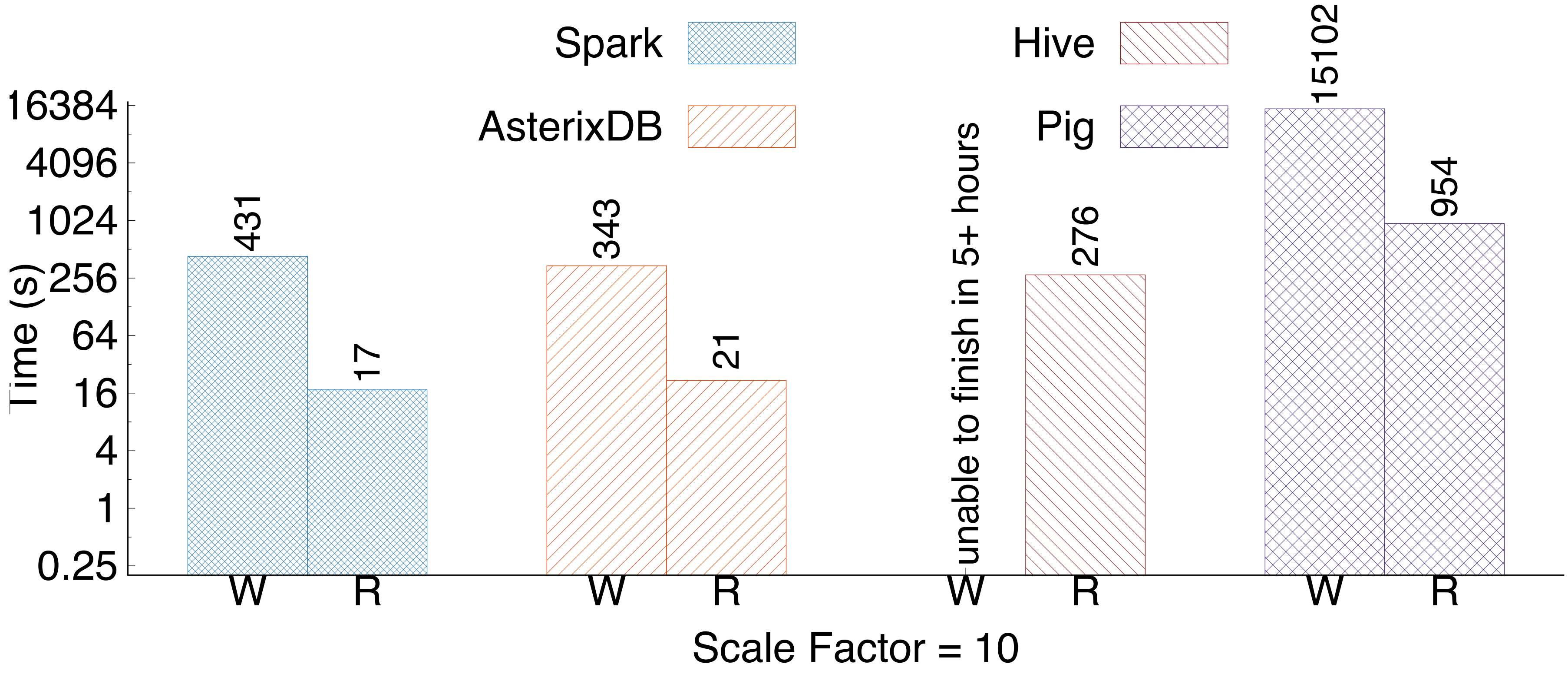
The role of the cost-based optimizer is to

(1) enumerate some set of equivalent plans

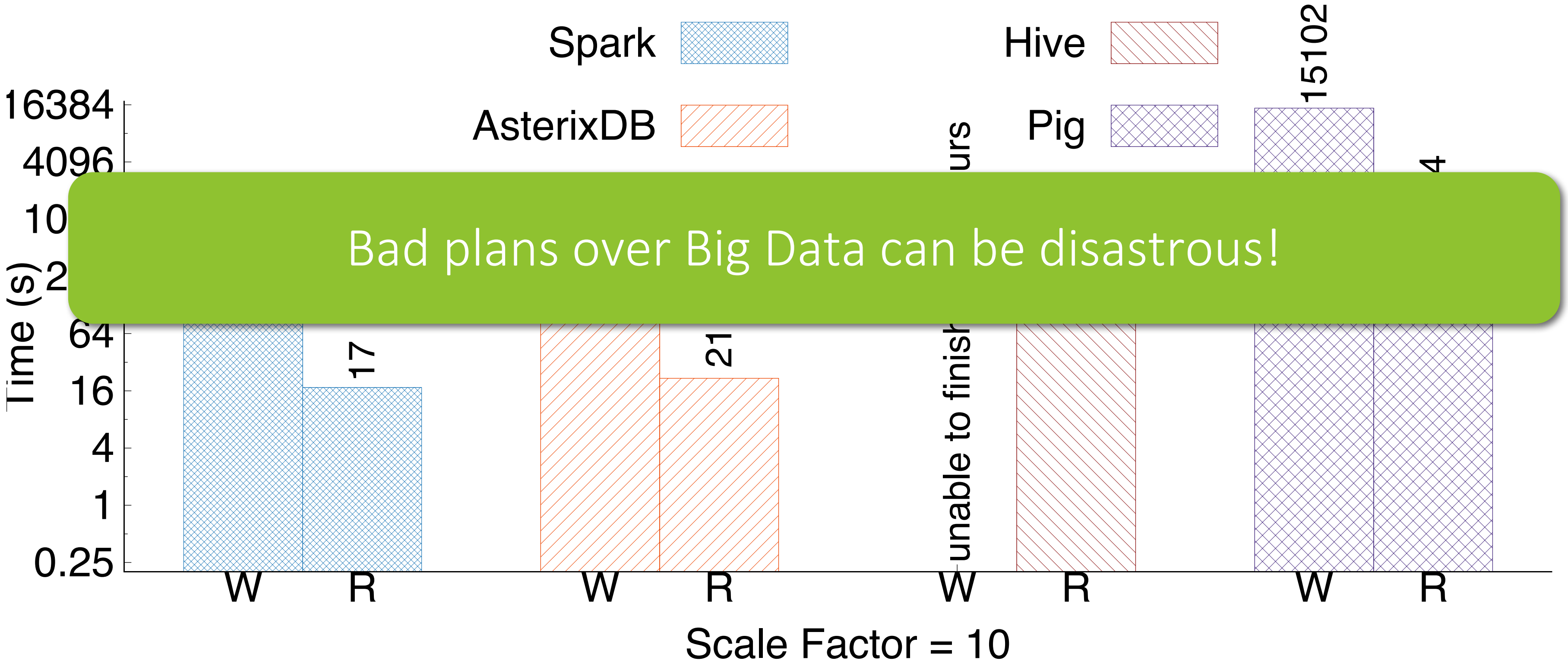
(2) estimate the cost of each

(3) select a sufficiently good plan

Query Optimization: Why Important?



Query Optimization: Why Important?



Bad plans over Big Data can be disastrous!

Challenges for Cost-based Optimizer in DISC

Lack of **upfront statistics**:

- data sits in HDFS and unstructured

Even if input statistics are available:

- **Correlations** between predicates
- Exponential error propagation in joins
- Arbitrary **UDFs**

Cost-based Optimizer in DISC: State of the Art

Pre-existing statistics

Bad statistics

No upfront statistics

Cost-based Optimizer in DISC: State of the Art

Pre-existing statistics

- Spark CBO[1]

- Collect and store statistics
- No runtime plan revision

Bad statistics

No upfront statistics

Cost-based Optimizer in DISC: State of the Art

Pre-existing statistics

- Spark CBO[1]

- Collect and store statistics
- No runtime plan revision

Bad statistics

- Adaptive Query planning[2]

Assumption is that some initial statistics exist

No upfront statistics

Cost-based Optimizer in DISC: State of the Art

Pre-existing statistics

- Spark CBO[1]

- Collect and store statistics
- No runtime plan revision

Bad statistics

- Adaptive Query planning[2]

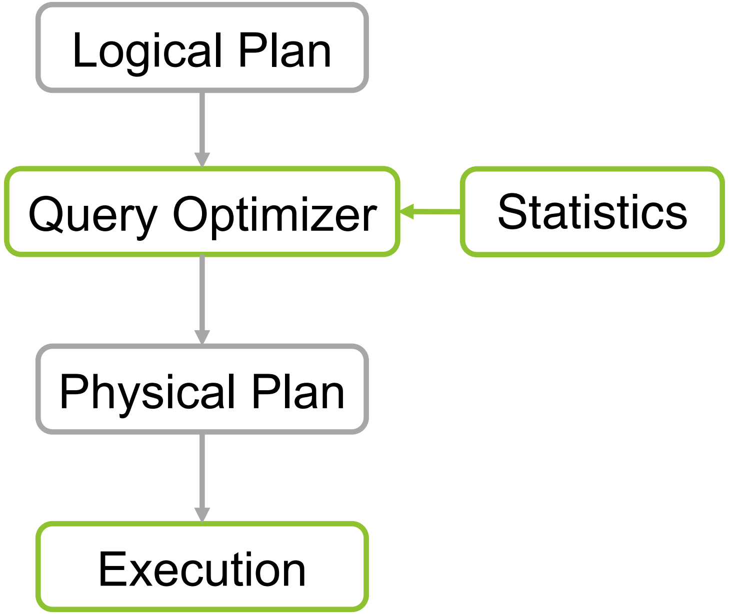
Assumption is that some initial statistics exist

No upfront statistics

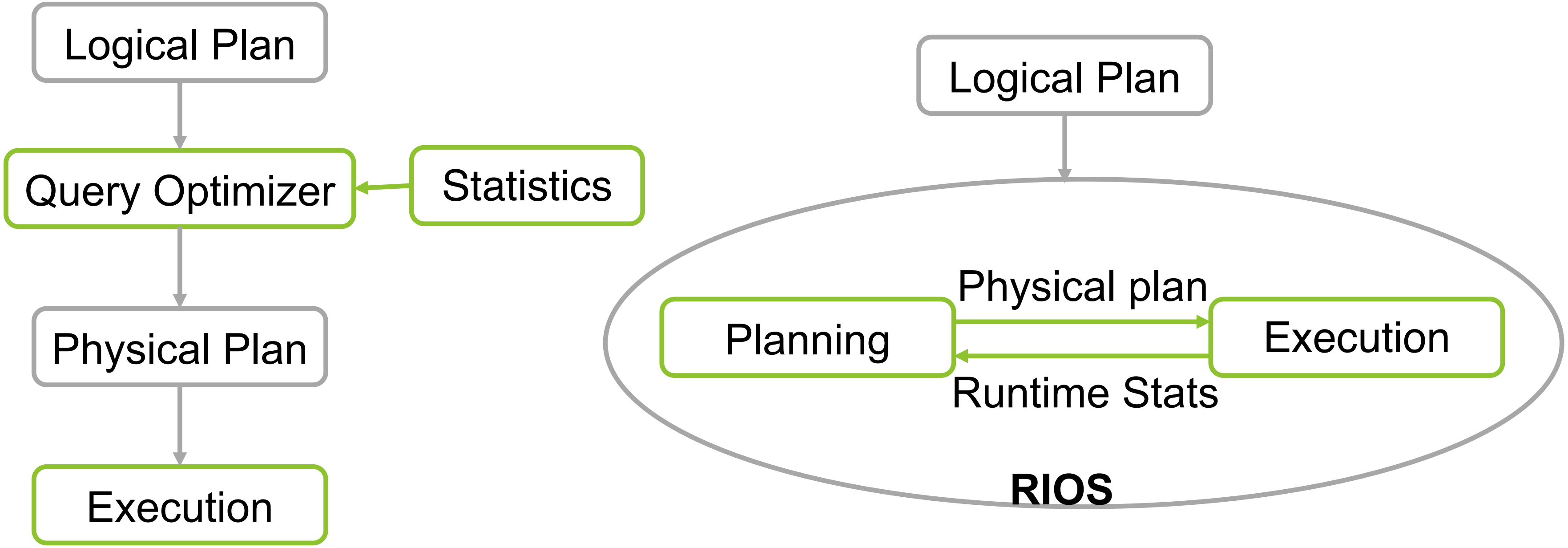
- Pilot runs (samples)[3]

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

Traditional Query Planning VS RIOS



Traditional Query Planning VS RIOS



Runtime Integrated Optimizer for Spark

Key idea: Execute-Gather-Aggregate-Plan strategy (EGAP)

- Query plans are lazily **executed**
- Statistics are **gathered** at runtime
- **Aggregate** statistics after gathering
- Joins are greedily **planned** for execution
- Plan can be **dynamically changed** if a **bad decision** was made

Neither upfront statistics nor pilot runs are required

- Raw dataset size is required for initial guess

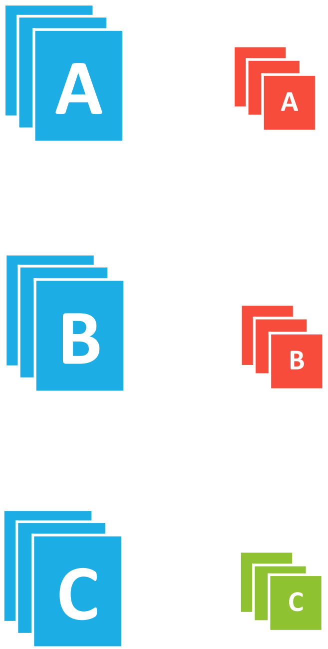
Support for not foreign-key joins

Runtime Optimizer: an Example

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

Assumption: $A < C$

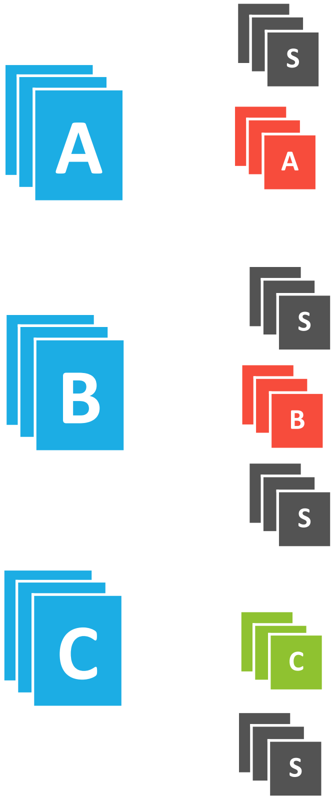
Runtime Optimizer: Execute Step



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

Assumption: $A < C$

Runtime Optimizer: Gather step



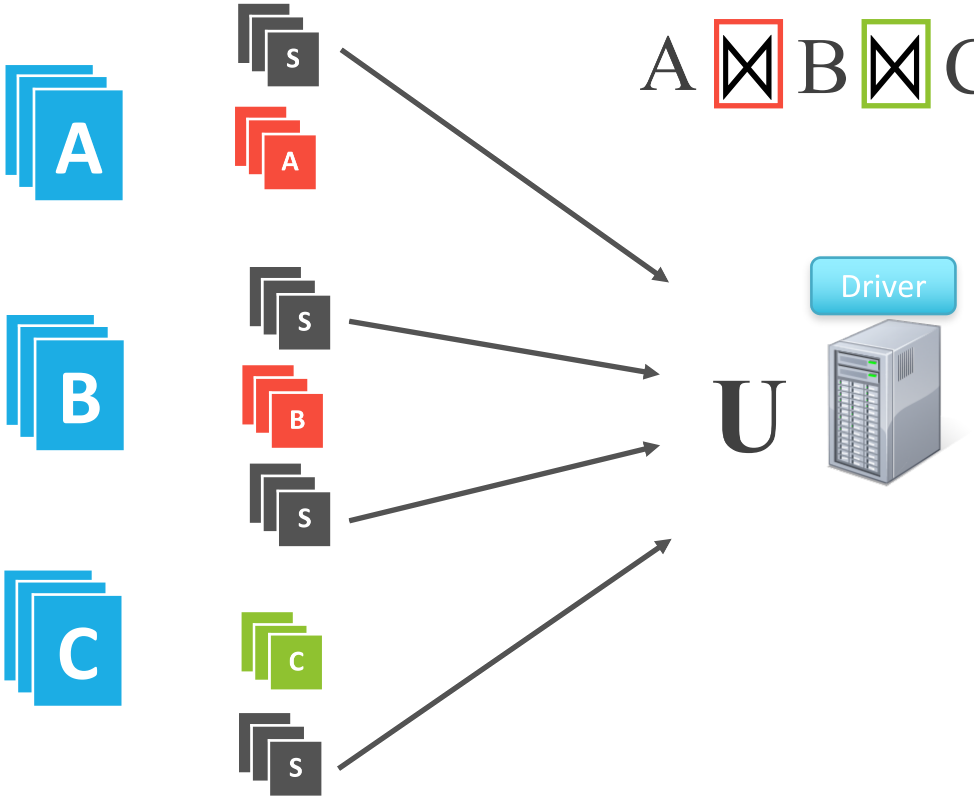
$$A \boxed{\bowtie} B \boxed{\bowtie} C$$

Assumption: $A < C$

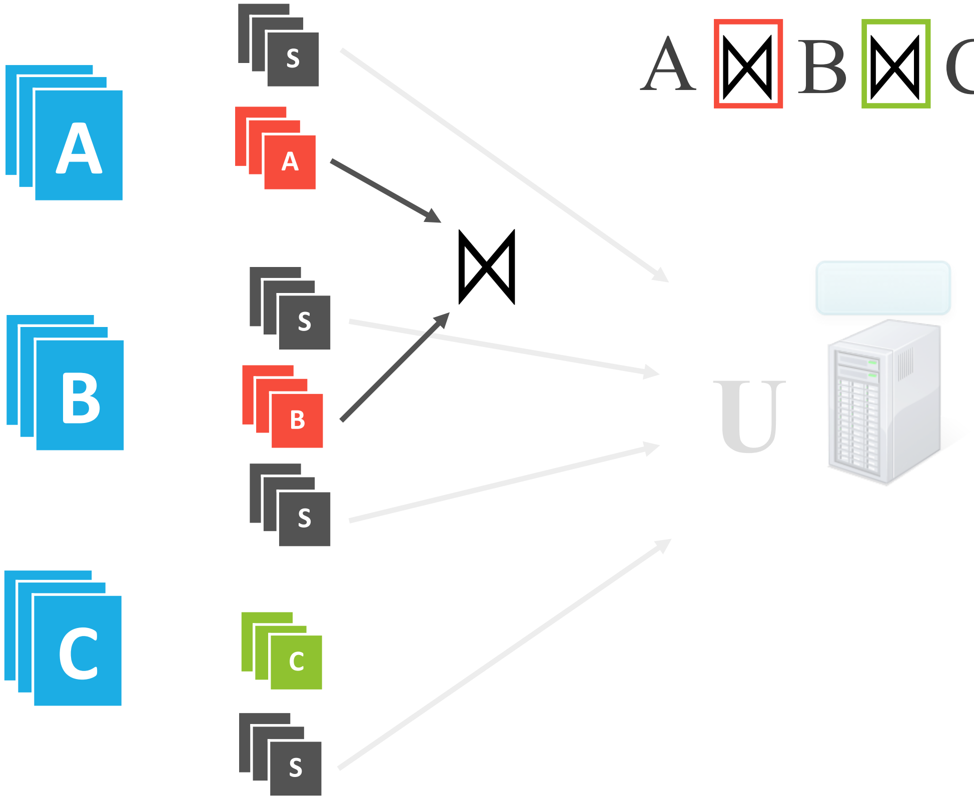
Runtime Optimizer: Aggregate step

$A \bowtie B \bowtie C$

Assumption: $A < C$



Runtime Optimizer: Plan step



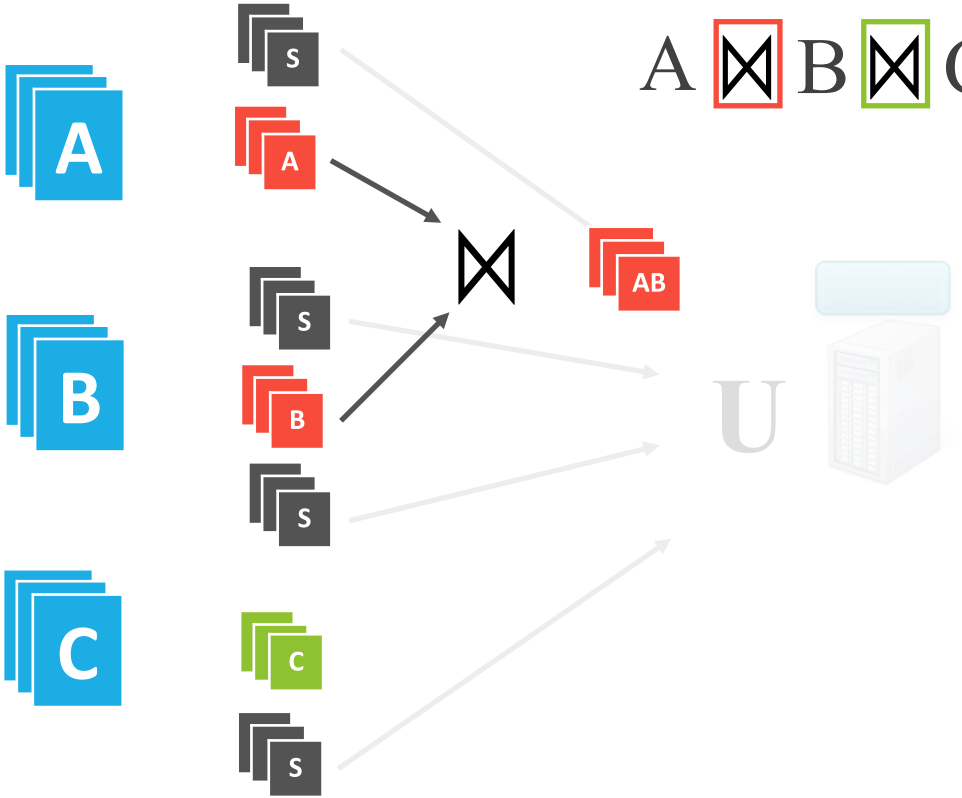
$A \bowtie B \bowtie C$

Assumption: $A < C$

Runtime Optimizer: Execute step

$A \bowtie B \bowtie C$

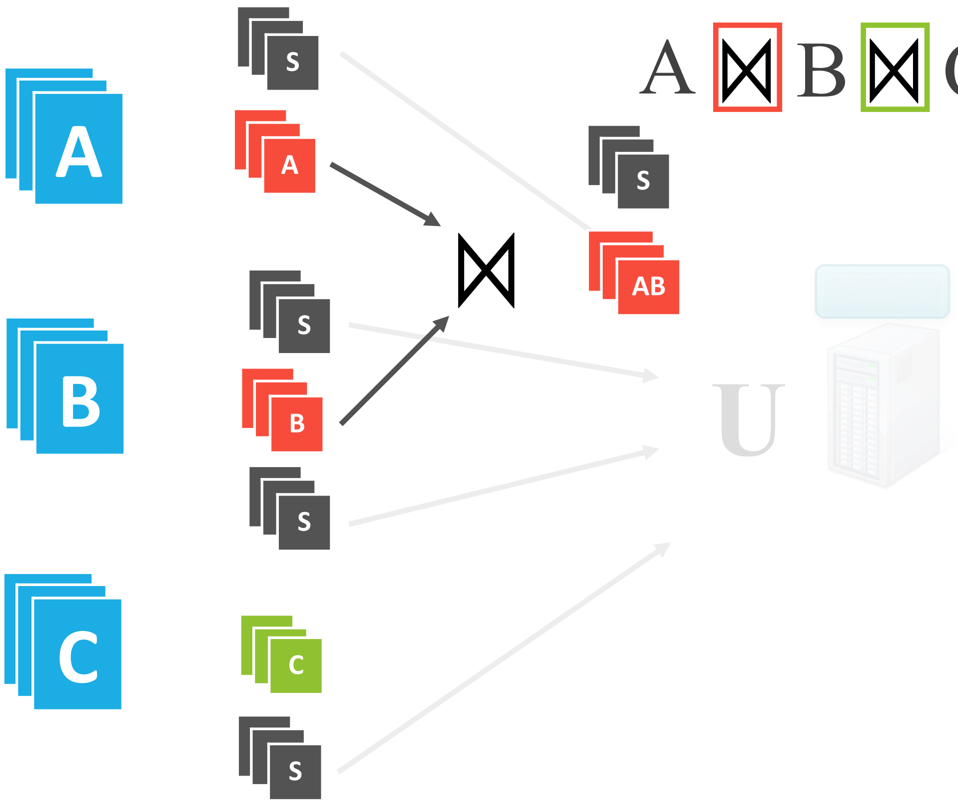
Assumption: $A < C$



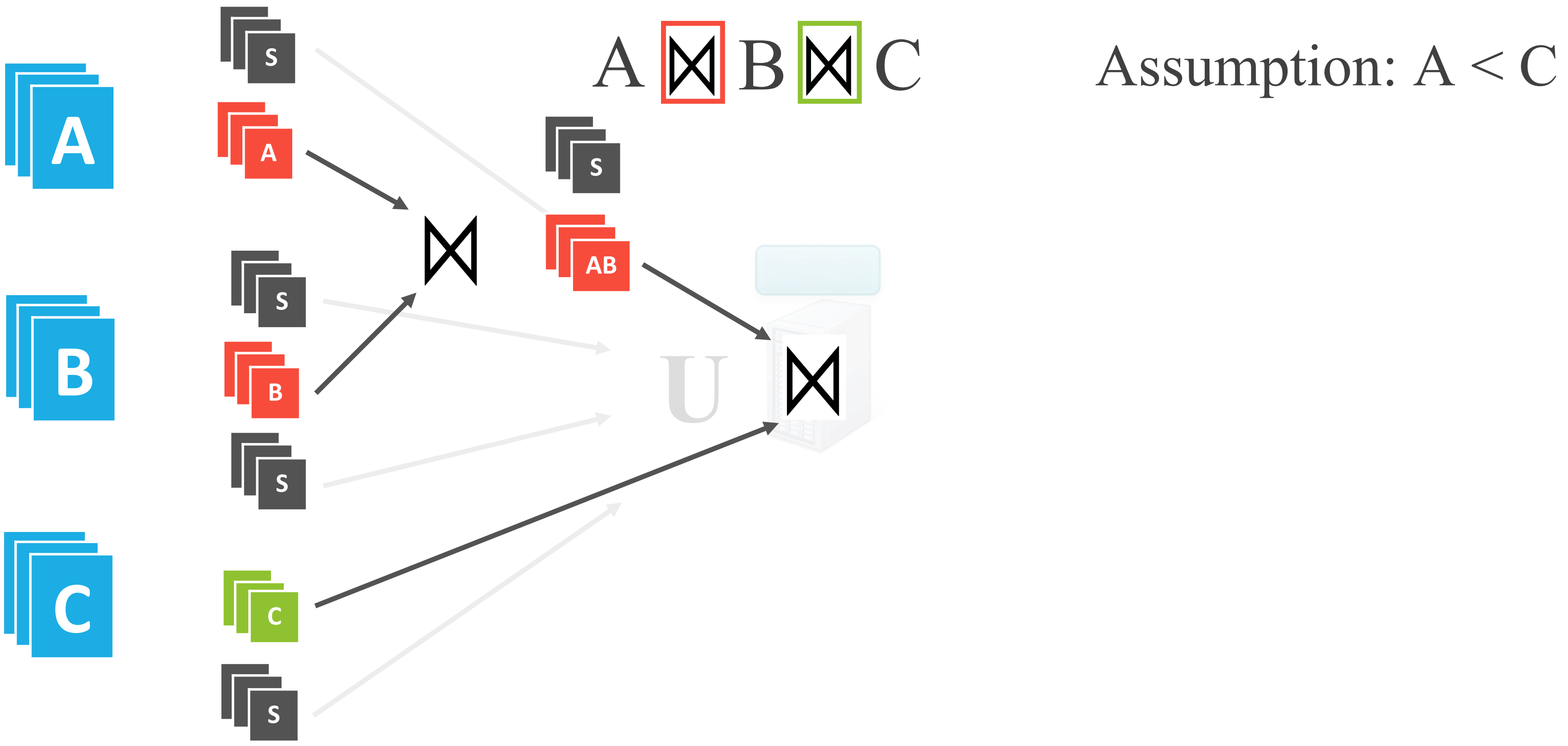
Runtime Optimizer: Gather step

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

Assumption: $A < C$



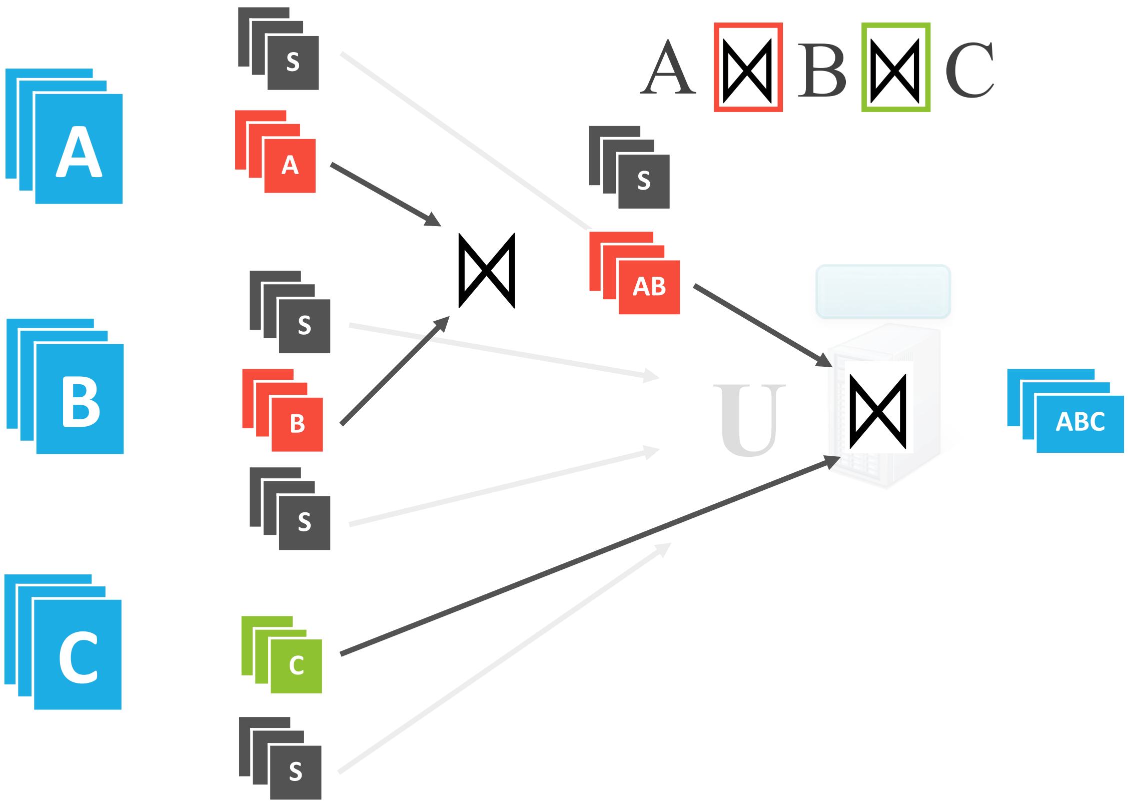
Runtime Optimizer: Plan step



Runtime Optimizer: Execute step

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

Assumption: $A < C$



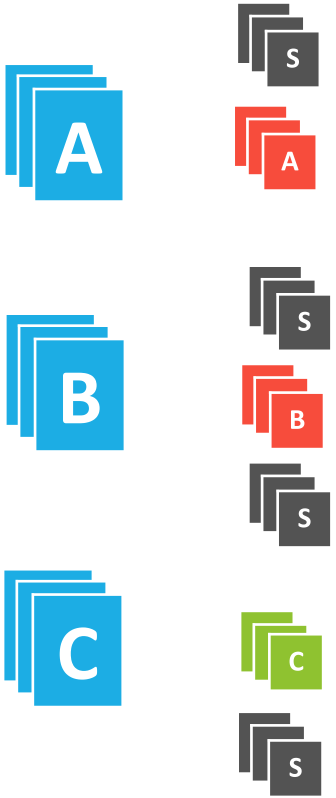
Runtime Optimizer: Wrong Guess

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

Assumption: $A < C$

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

Runtime Optimizer: Wrong Guess



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

Assumption: $A < C$

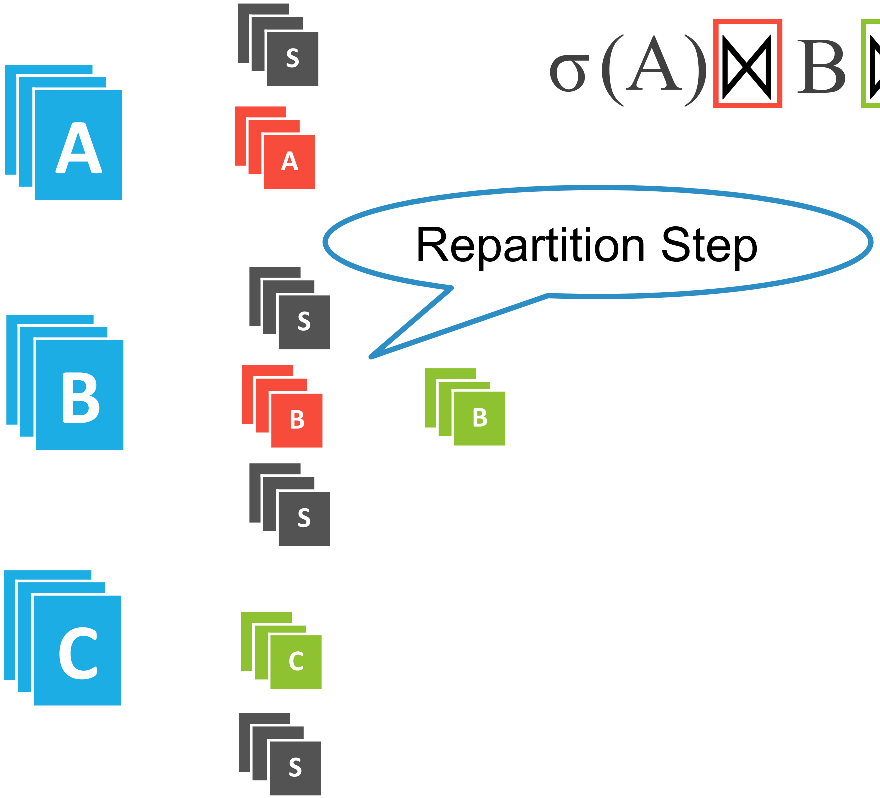
$\sigma(A) > \sigma(C)$

Runtime Optimizer: Wrong Guess

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

Assumption: $A < C$

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

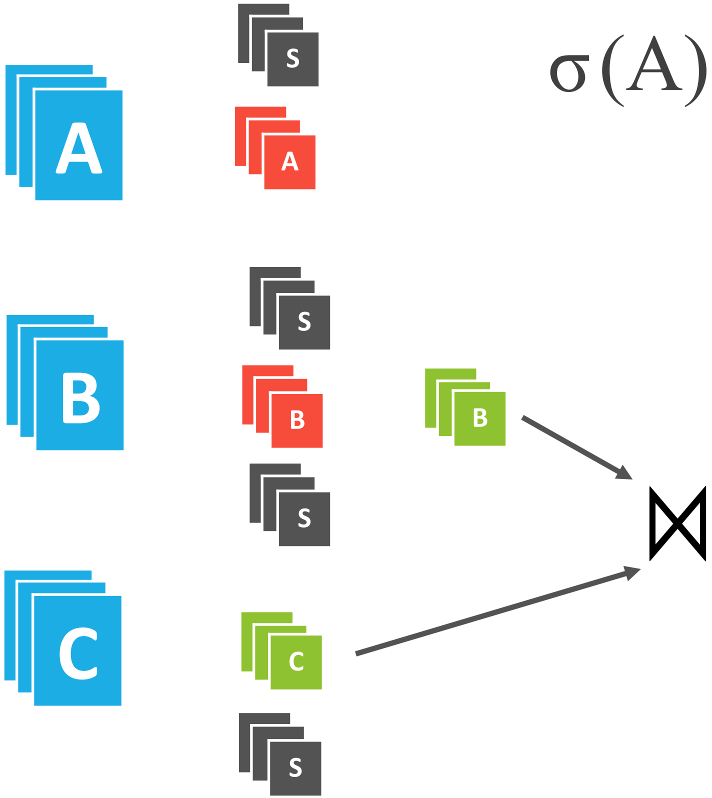


Runtime Optimizer: Wrong Guess

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

Assumption: $A < C$

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

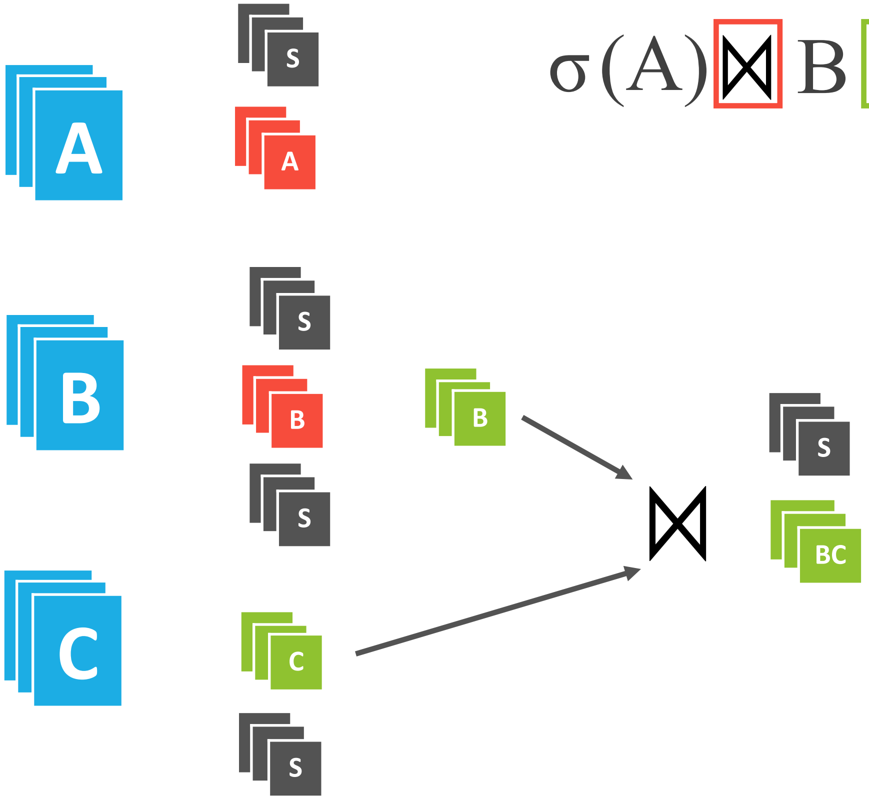


Runtime Optimizer: Wrong Guess

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

Assumption: $A < C$

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

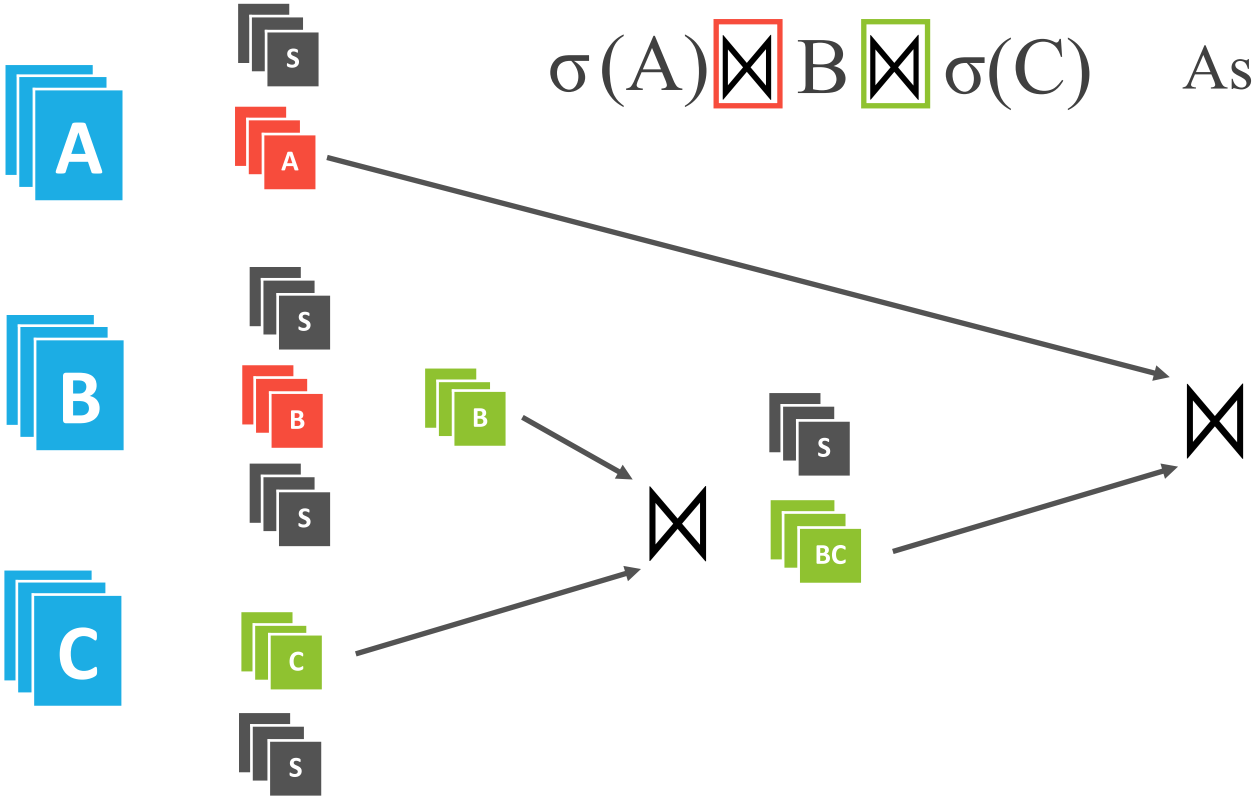


Runtime Optimizer: Wrong Guess

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

Assumption: A < C

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

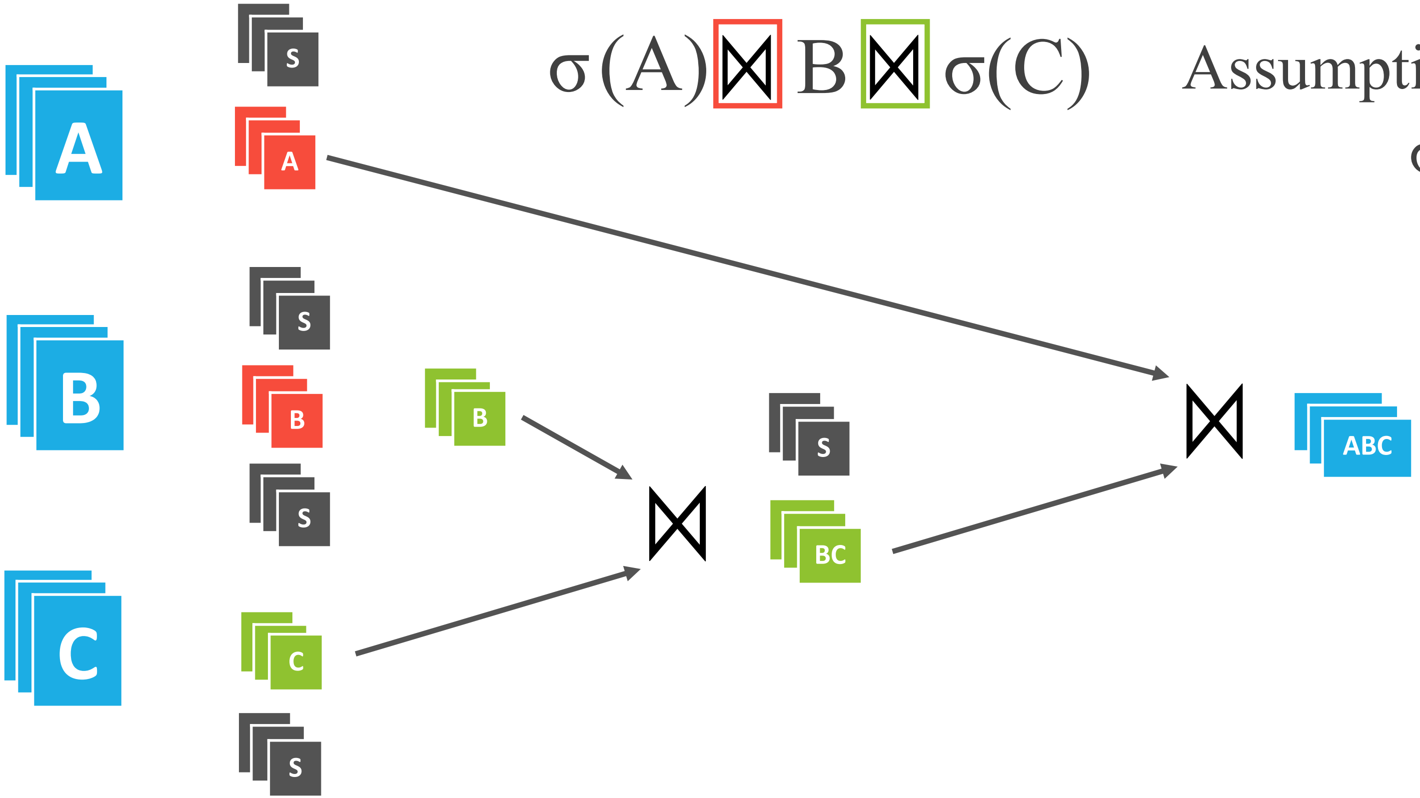


Runtime Optimizer: Wrong Guess

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

Assumption: A < C

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



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 aggregated 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

Aggregate - Efficient accumulation of statistics

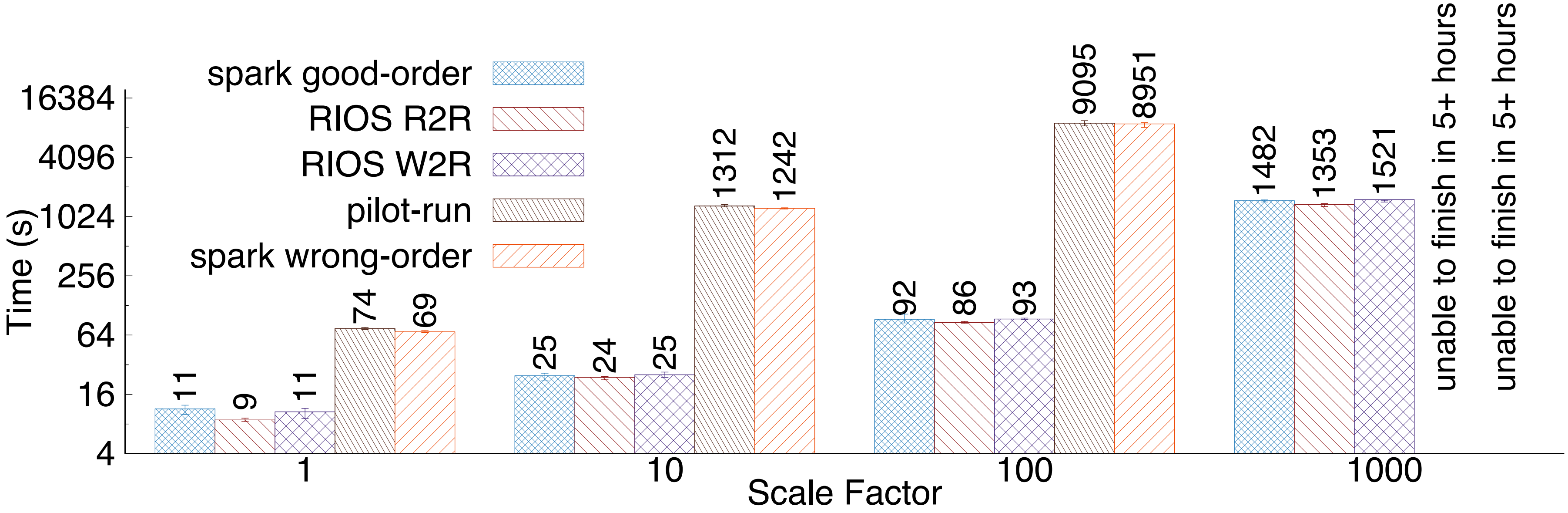
Plan - Try to schedule as many broadcast joins as possible

Experiments

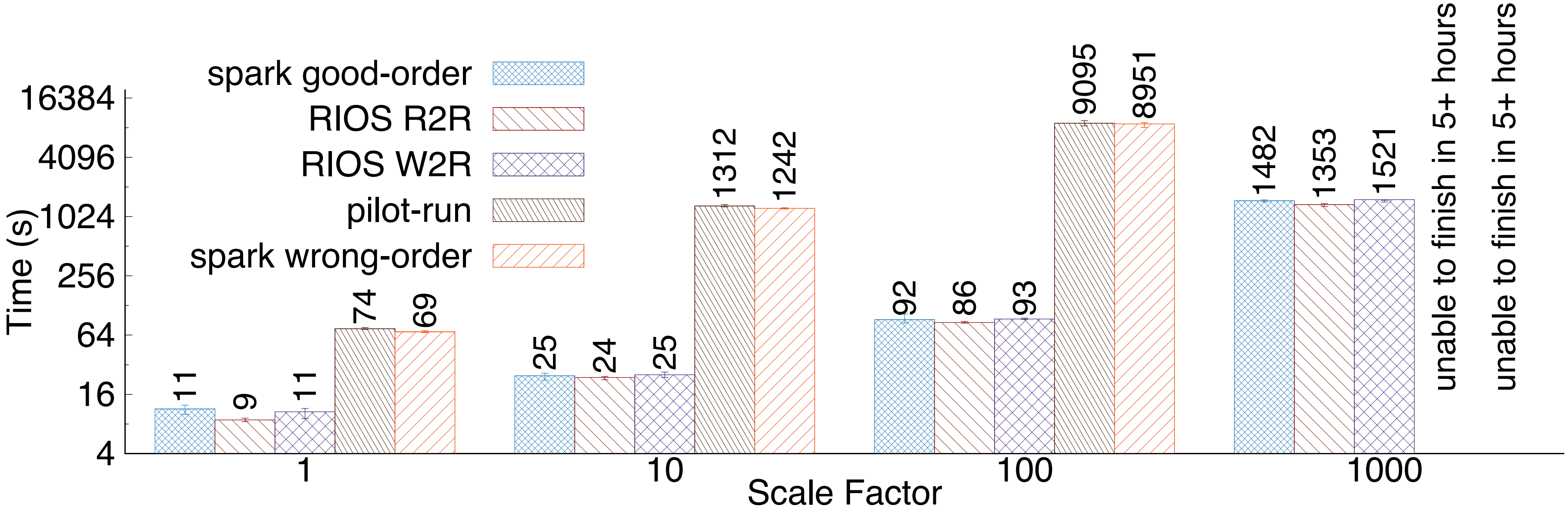
Q1: What are the performance of RIOS compared to regular Spark, pilot runs and Spark-CBO?

Q2: How expensive are wrong guesses?

Minibenchmark with 3 Fact Tables

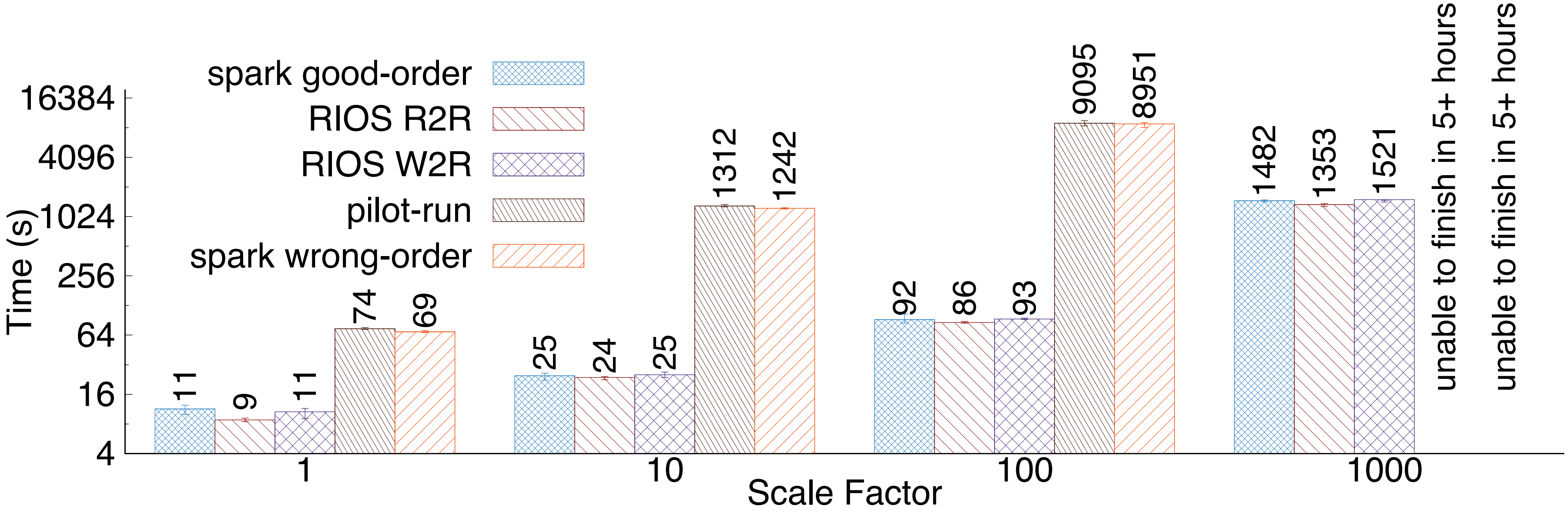


Minibenchmark with 3 Fact Tables



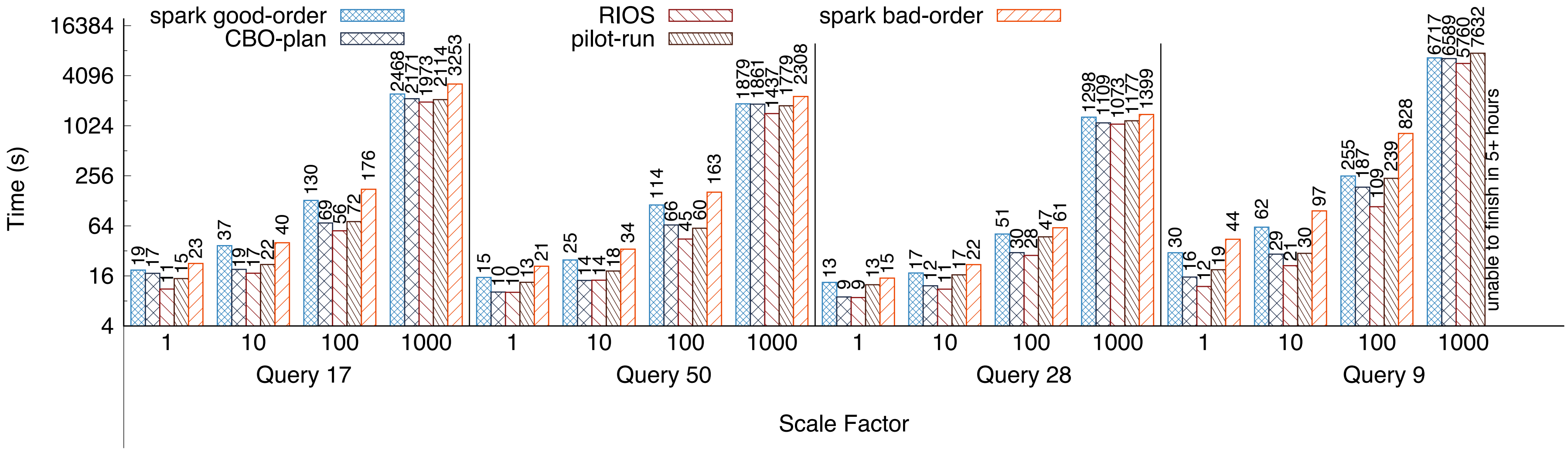
Q1: RIOS is always faster than Spark and pilot run

Minibenchmark with 3 Fact Tables

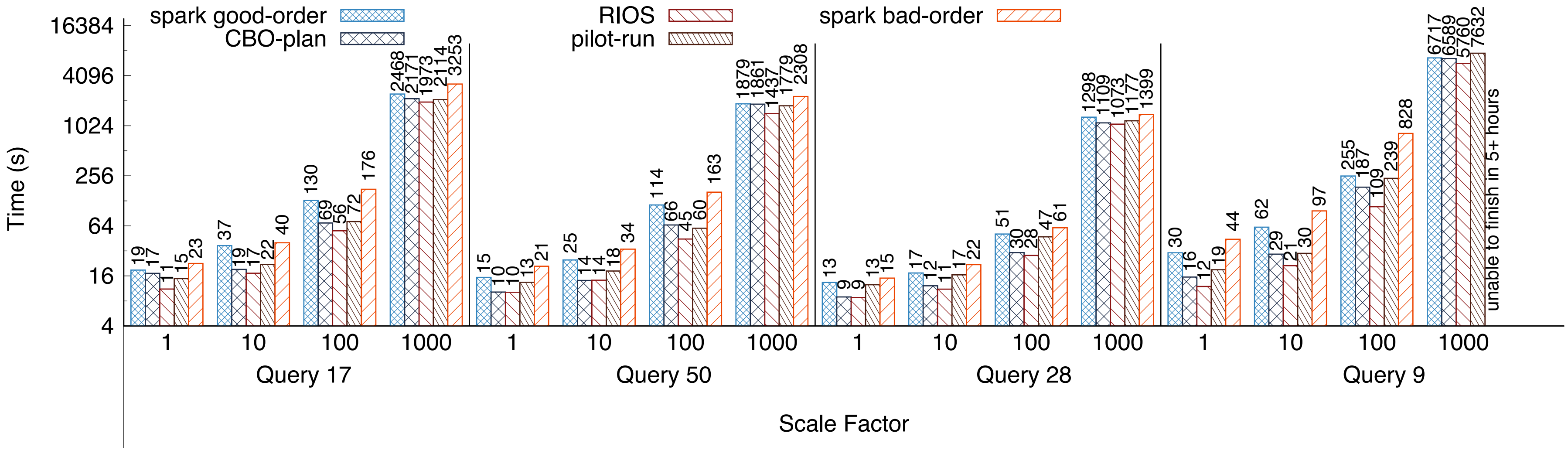


Q2: Not much, around 15% in the worst case

TPCDS and TPCH Queries



TPCDS and TPCH Queries



Q1: RIOS is always the faster approach

Conclusions

RIOS: cost-based 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 plan generated by pilot run, and > 100X than previous approaches for fact table joins.


thank you
danke
謝謝
ngiyabonga
рахмат
teşekkür ederim
Баярлалаа
spas
mersi
kia ora
barka
welalin
tack
mahalo
tapadh leat
спасибо
faafetai lava
vinaka
blagodaram
dank je
misaotra
matondo
paldies
grazzi
gracias
спасибі
kiitos
dankie
хвала
nanni
nandri
dhanyavad
asante
manana
bedankt
bayarlalaa
gràçie
hvala
maururu
köszönöm
dankon
ačiū
obrigada
tenki
murakoze
enkosi
akun
djiere dieuf
tau
mochchakkeram
dyakuyu
дякую
mamnun
chokrane
dziękuję
chnorakaloutioun
gratias ago
gràcies
sulpáy
go raibh maith agat
sobodi
děkuji
sagolun
sukriya
kop khun krap
taiku
obrigado
arigatô
takk
dakujem
trugarez
ありがとう
mèsi
didi madloba
najis tuke
tanemirt
grazie
terima kasih
rahmet
diolch
dhanyavadagalu
shukriya
mercé
мерси
kam sah hamnida
rahmat
ευχαριστώ
তোমাকে ধন্যবাদ
감사합니다
xièxie
merci

Experiment Configuration

- Datasets:
  - TPCDS
  - TPCH
- Configuration:
  - 16 machines, 4 cores (2 hyper threads per core) machines, 32GB of RAM, 1TB disk
  - Spark 2.2.1
  - Scale factor from 1 to 1000 (~1TB)

Reference

1. https://databricks.com/blog/2017/08/31/cost-based-optimizer-in-apache-spark-2-2.html.
2. S.Agarwal,S.Kandula,N.Bruno,M.-C.Wu,I.Stoica,andJ.Zhou.Re-optimizing data-parallel computing. In NSDI, 2012.
3. K. Karanasos, A. Balmin, M. Kutsch, F. Ozcan, V. Ercegovac, C. Xia, and J. Jackson. Dynamically optimizing queries over large scale data platforms. In SIGMOD, pages 943–954.
4. S. Chaudhuri, G. Das, and V. Narasayya. Optimized stratified sampling for approximate query processing. TODS, 32(2), jun 2007.
5. O. Papapetrou, W. Siberski, and W. Nejdl. Cardinality estimation and dynamic length adaptation for bloom filters. Distributed and Parallel Databases, 28(2):119– 156, 2010.

Thank you