ApproxJoin

Approximate Distributed Joins

Do Le Quoc, Istemi Ekin Akkus,
Pramod Bhatotia, Spyros Blanas, Ruichuan Chen,
Christof Fetzer, Thorsten Strufe

TECHNISCHE UNIVERSITÄT DRESDEN

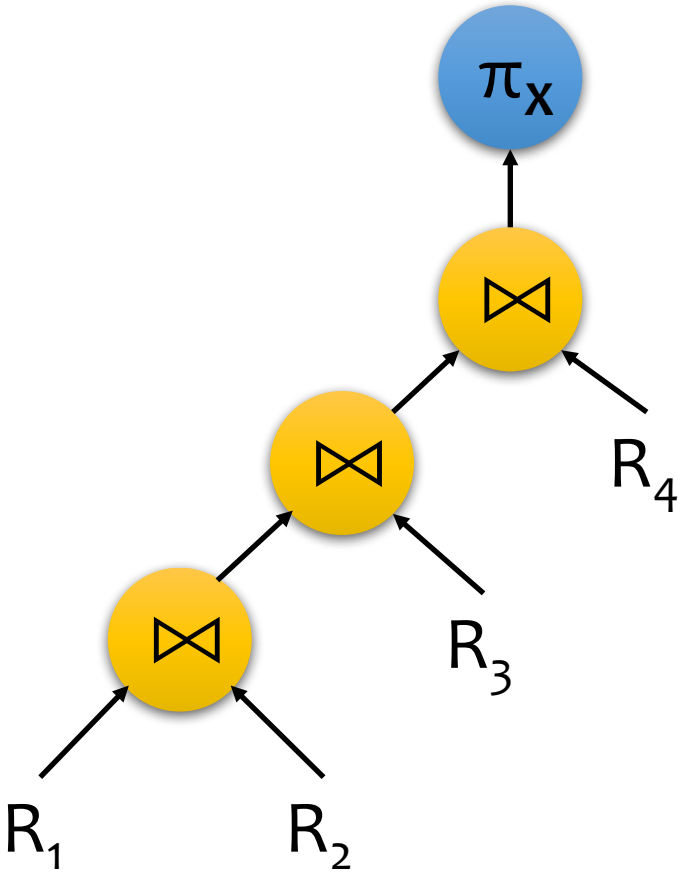
THE UNIVERSITY of EDINBURGH

NOKIA Bell Labs

THE OHIO STATE UNIVERSITY

10/2018

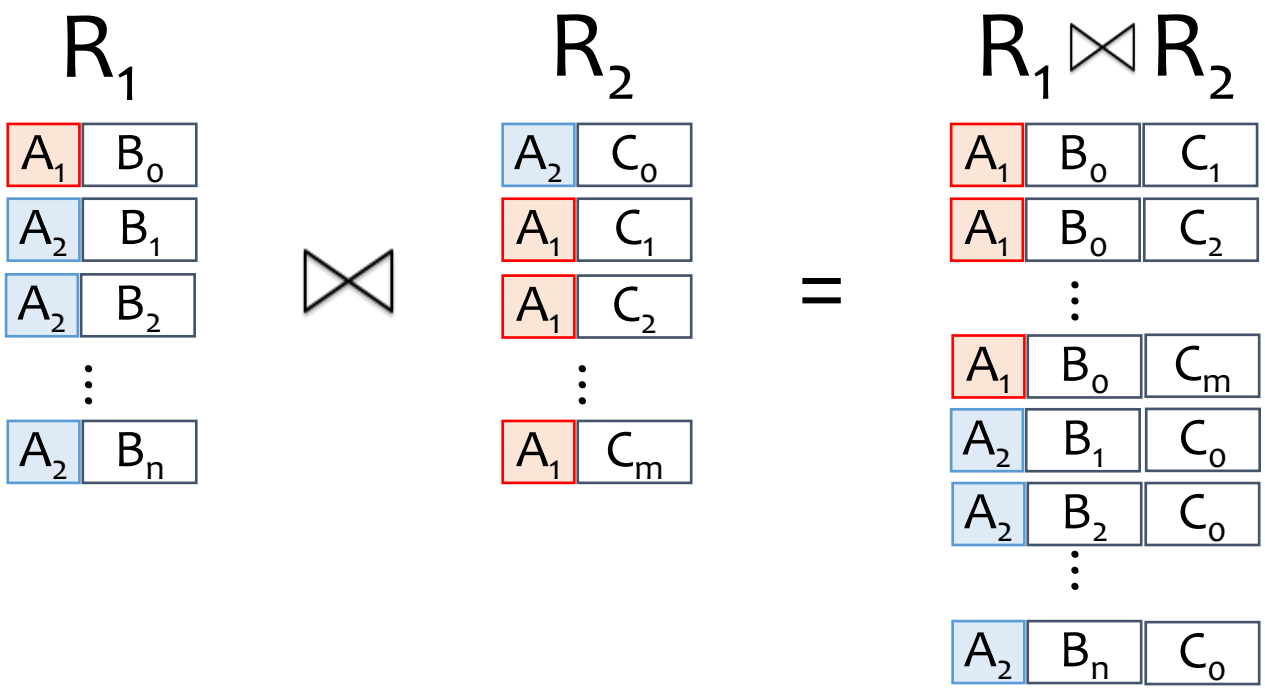
Motivation



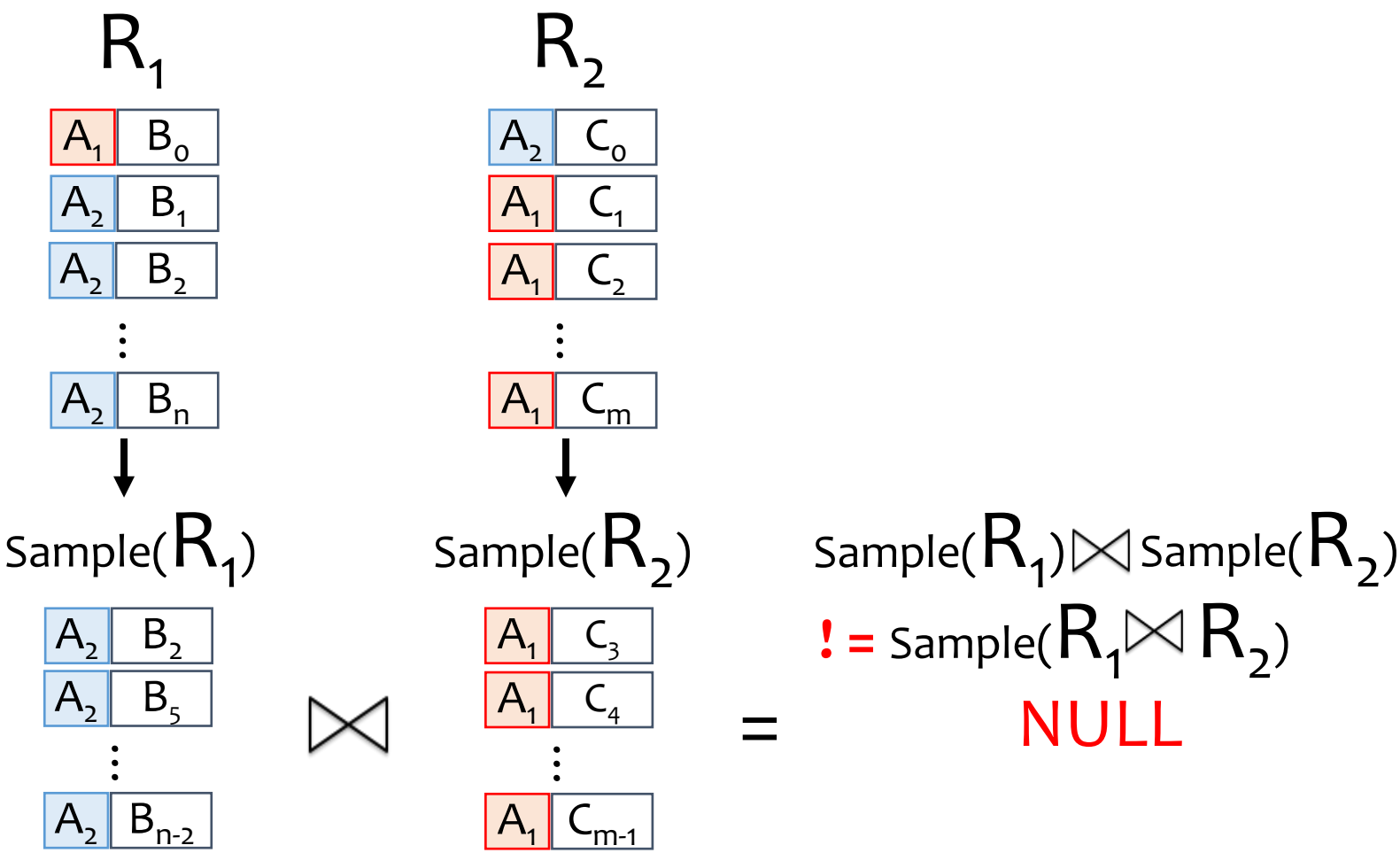
Join is a **critical** operation in big data analytics systems, but it is very **expensive**

Reduce the overhead of join operations using a **sampling-based** approach

Motivation

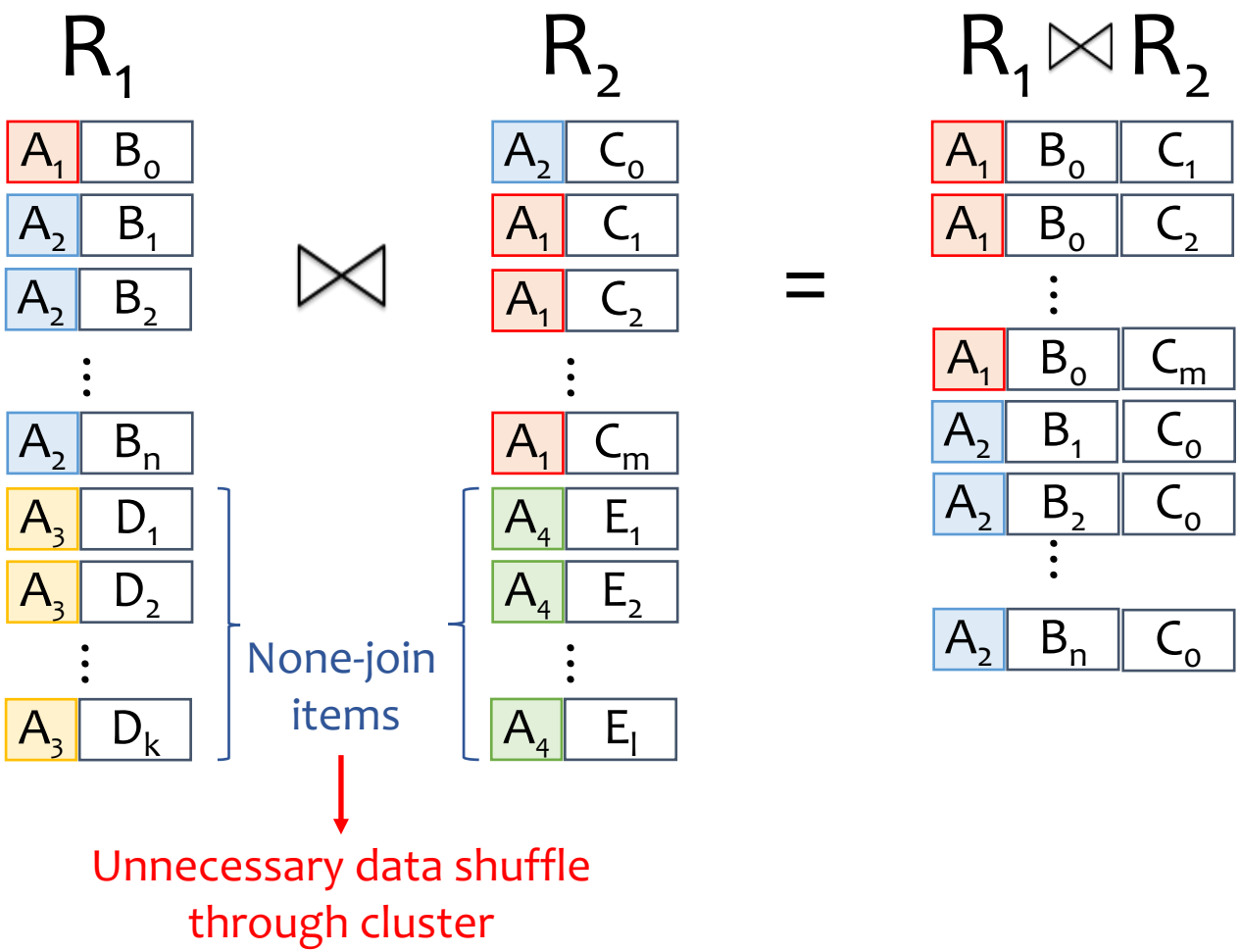

R_1
A_1 B_0
A_2 B_1
A_2 B_2
A_2 B_n
$\bowtie$
R_2
A_2 C_0
A_1 C_1
A_1 C_2
A_1 C_m
=
$R_1 \bowtie R_2$
A_1 B_0 C_1
A_1 B_0 C_2
A_1 B_0 C_m
A_2 B_1 C_0
A_2 B_2 C_0
A_2 B_n C_0

Motivation



Sampling over joins is a challenging task regarding the output quality

Motivation


R_1
R_2
$R_1 \bowtie R_2$
A_1 B_0
A_2 B_1
A_2 B_2
A_2 B_n
A_3 D_1
A_3 D_2
A_3 D_k
A_2 C_0
A_1 C_1
A_1 C_2
A_1 C_m
A_4 E_1
A_4 E_2
A_4 E_l
A_1 B_0 C_1
A_1 B_0 C_2
A_1 B_0 C_m
A_2 B_1 C_0
A_2 B_2 C_0
A_2 B_n C_0
None-join items
Unnecessary data shuffle through cluster

State-of-the-art Systems

AQUA (SIGMOD'99) Sampling over joins (SIGMOD'99)	Requiring priori knowledge of inputs (statistical info, indices)
RippleJoin (SIGMOD'99), WanderJoin (SIGMOD'16)	Using online aggregation approach for joins
SparkSQL (SIGMOD'15), SnappyData (SIGMOD'16)	Using pre-existing samples to serve queries

State-of-the-art Systems

AQUA (SIGMOD'99)
Sampling over joins (SIGMOD'9

Requiring priori knowledge of inputs
al info, indices)

Designed for single node system

RippleJoin (SIGMOD'99),
WanderJoin (SIGMOD'16)

Using online aggregation approach for joins

Do not support sampling over joins

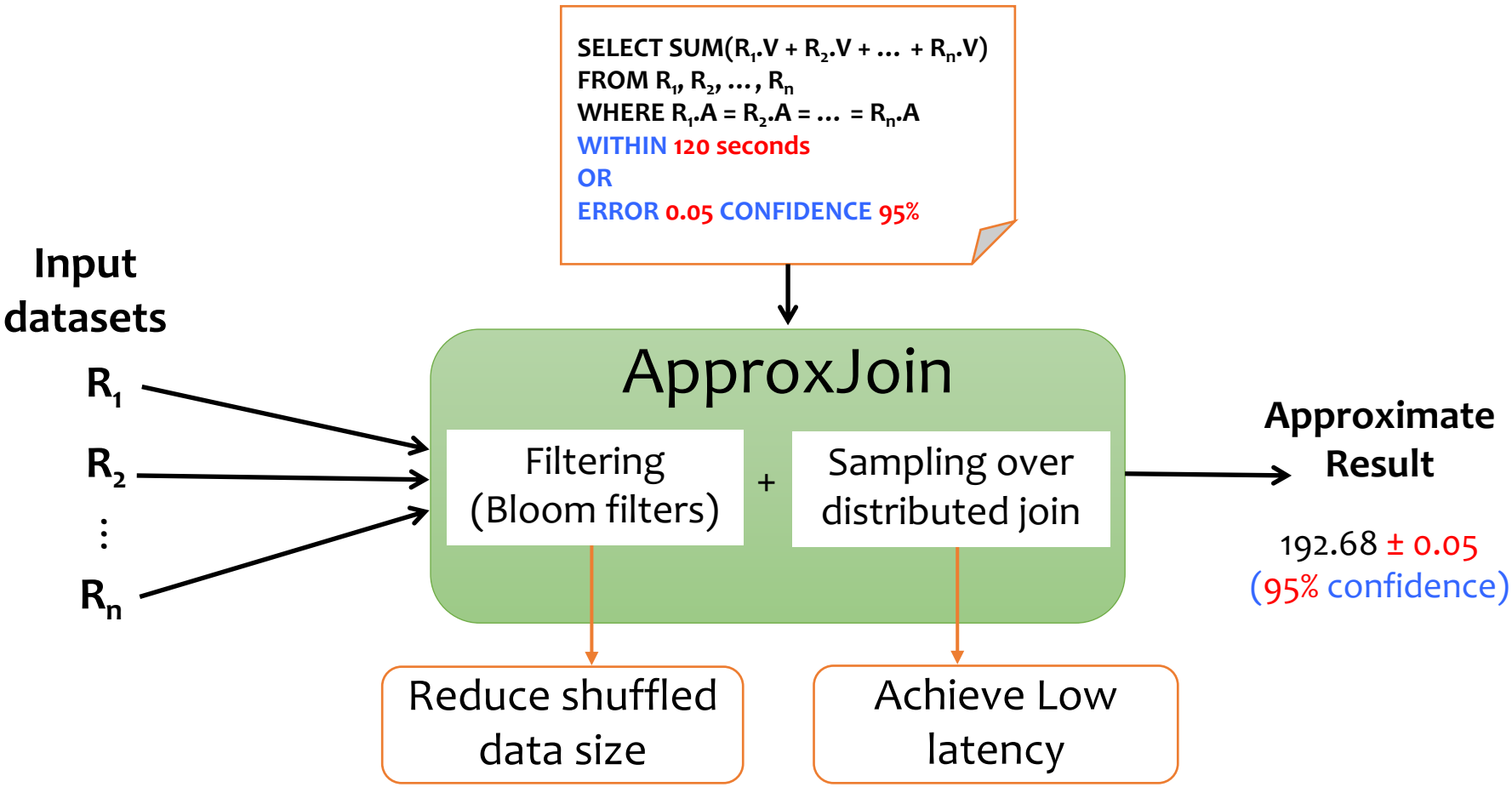
SparkSQL (SIGMOD
SnappyData (SIGMOD 16

ting samples to serve queries

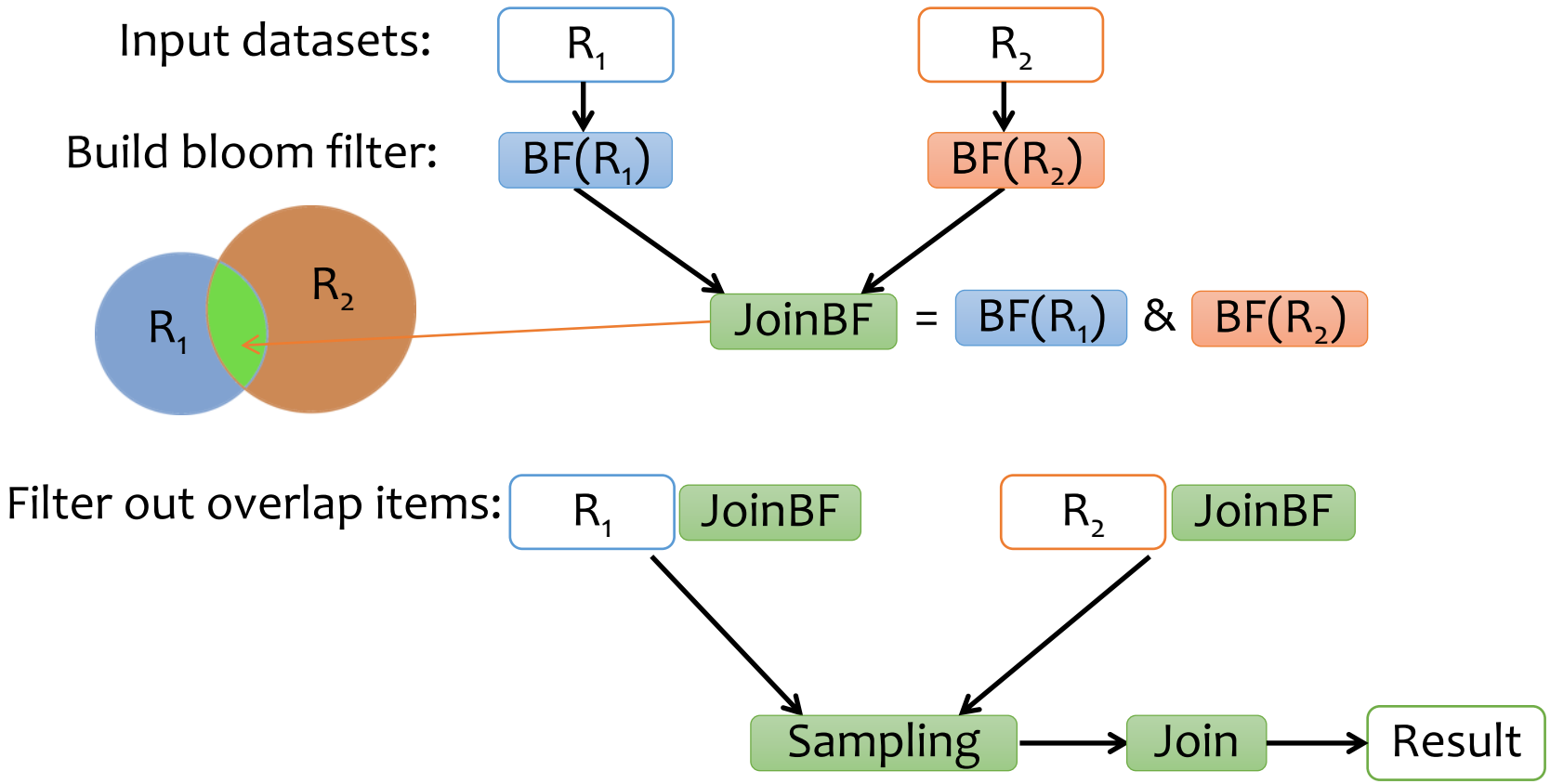
Outline

- ~~Motivation~~
- Design
- Evaluation

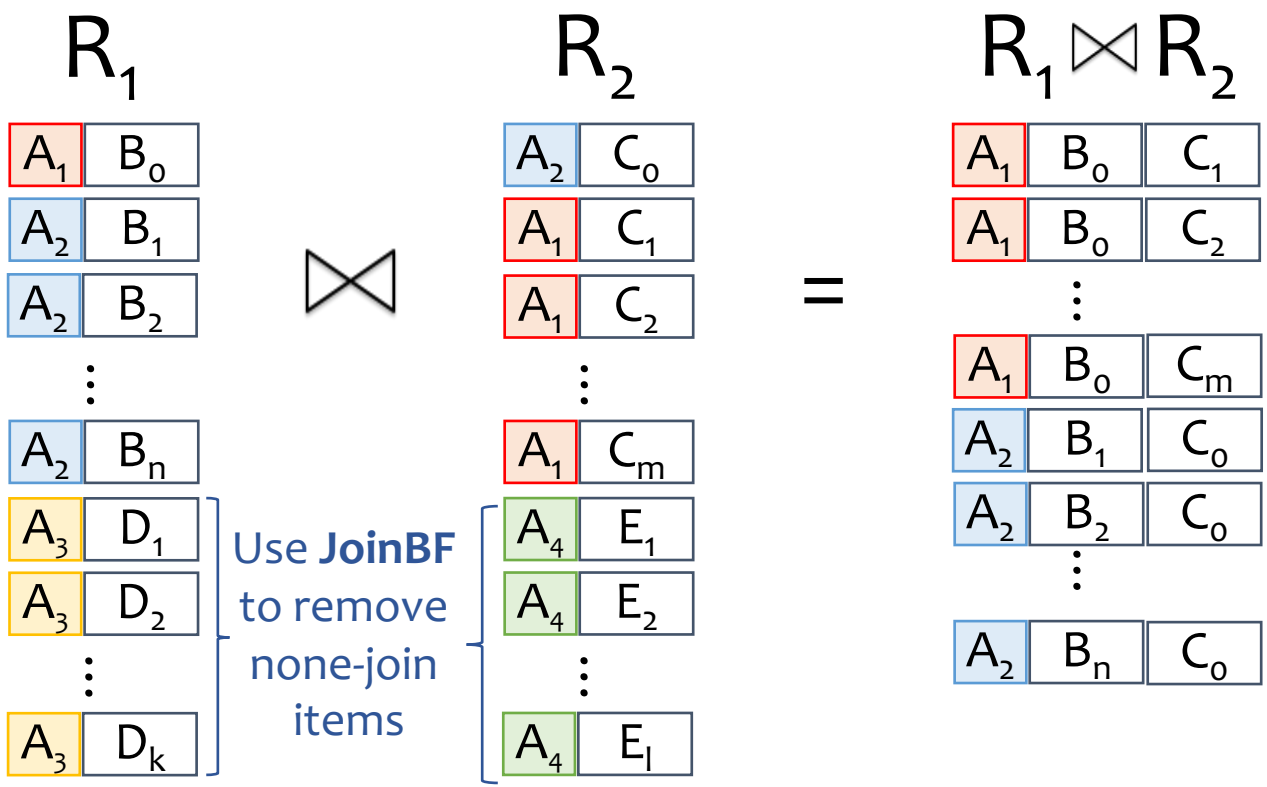
ApproxJoin: System Overview



ApproxJoin: Core Idea

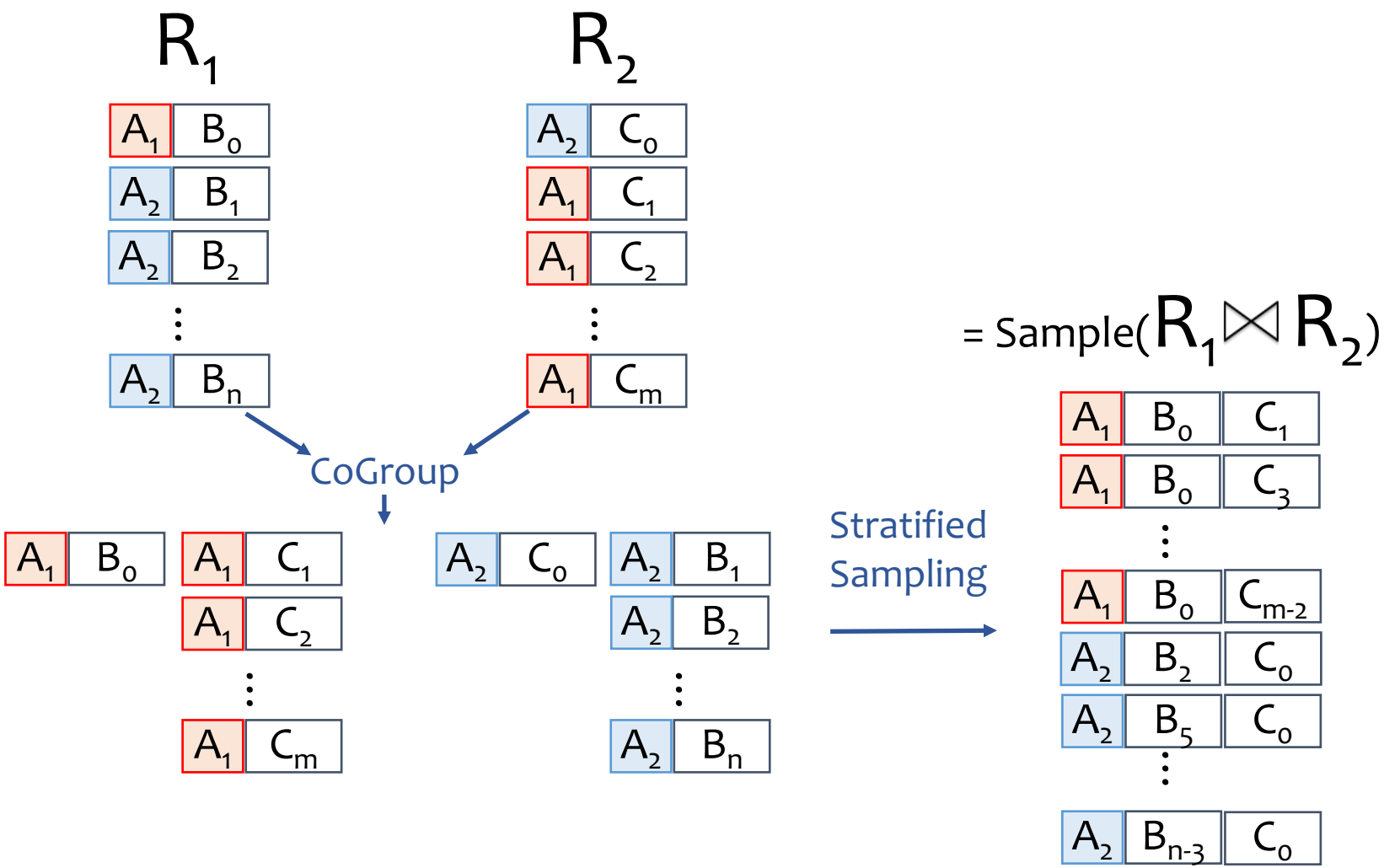


ApproxJoin: Filtering

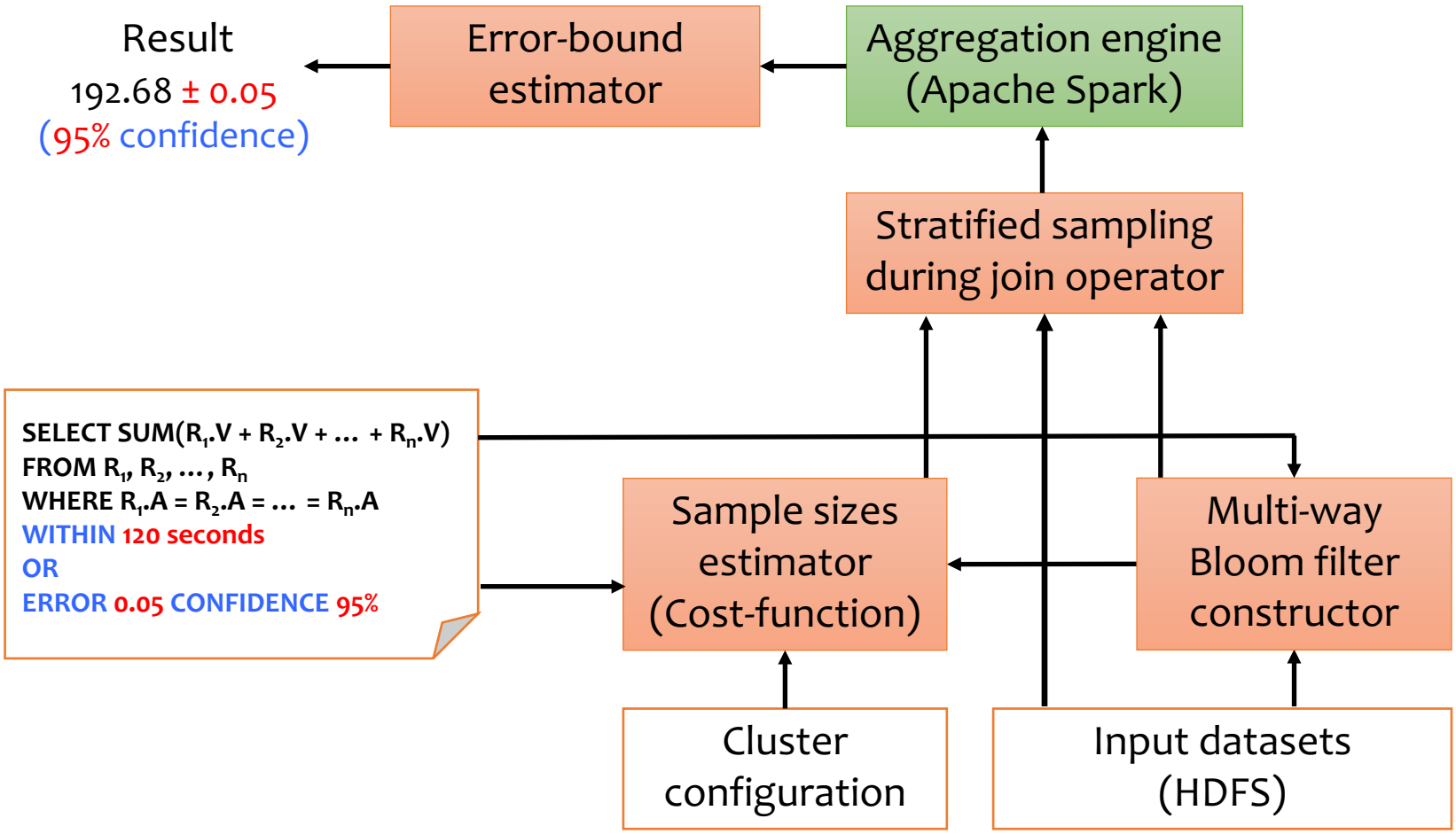


BF(R_1) = $\{A_1, A_2, A_3\}$ **BF(R_2)** = $\{A_1, A_2, A_4\}$ $\longrightarrow$ **JoinBF** = $\{A_1, A_2\}$

ApproxJoin: Sampling



ApproxJoin: Implementation



Outline

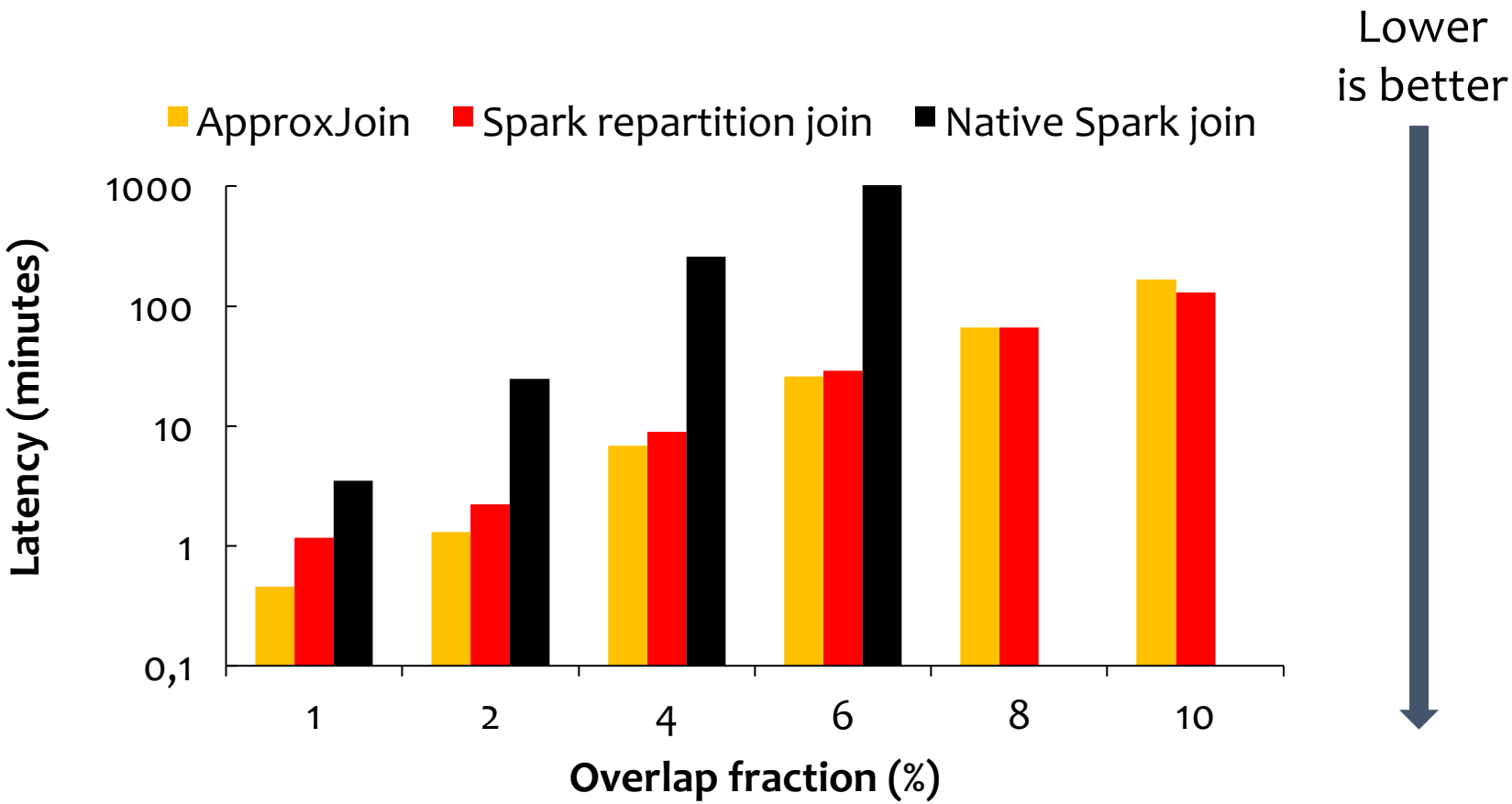
- ~~Motivation~~
- ~~Design~~
- Evaluation

Experimental Setup

- Evaluation questions
  - Latency vs overlap fraction
  - Shuffled data size vs overlap fraction
  - Latency vs sampling fraction
- Testbed
  - Cluster: 10 nodes
  - Datasets:
    - Synthesis: Poisson distribution datasets, TPC-H
    - CAIDA Network traffic traces; Netflix Prize

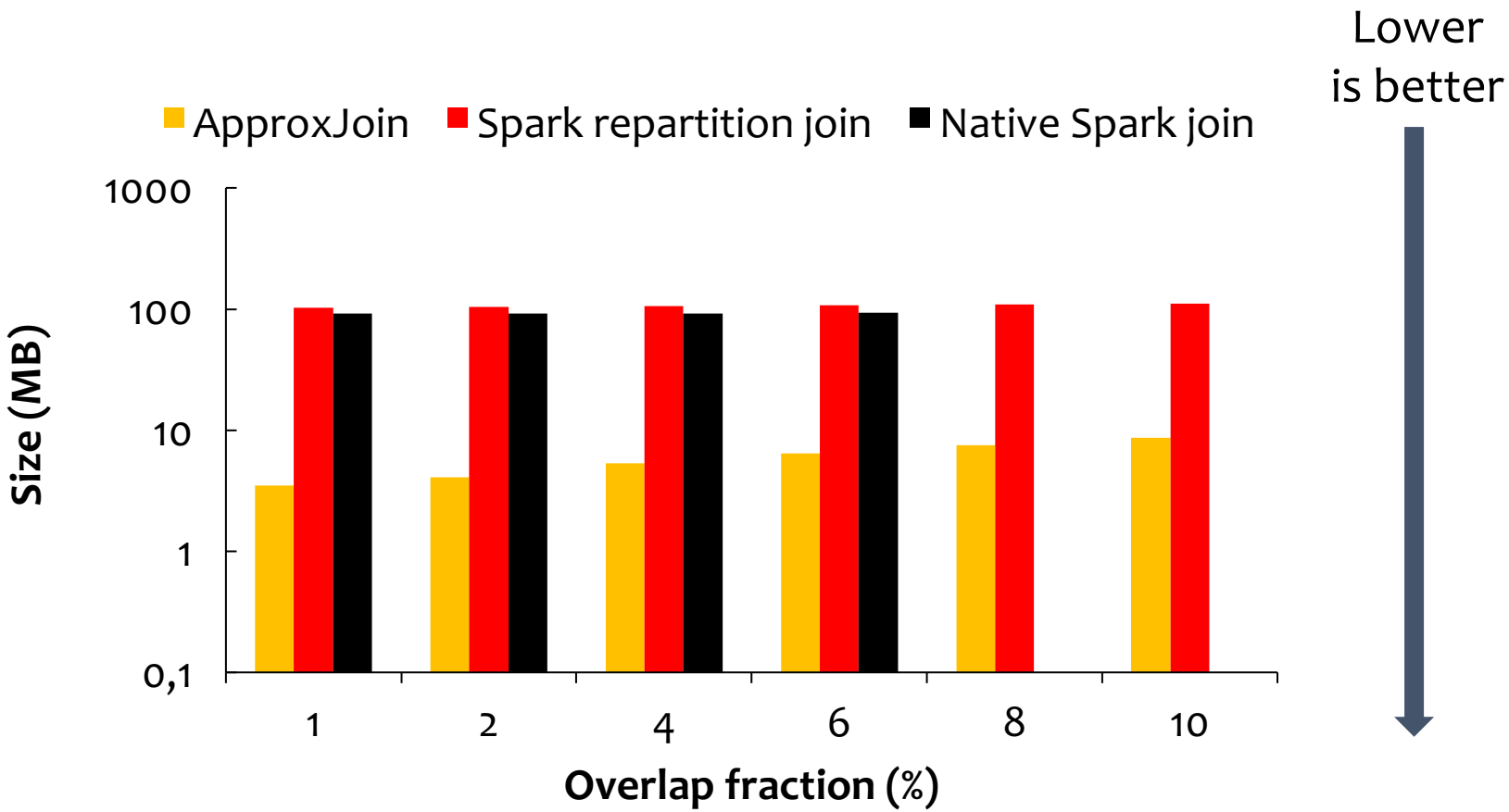
See the paper for more results!

Latency



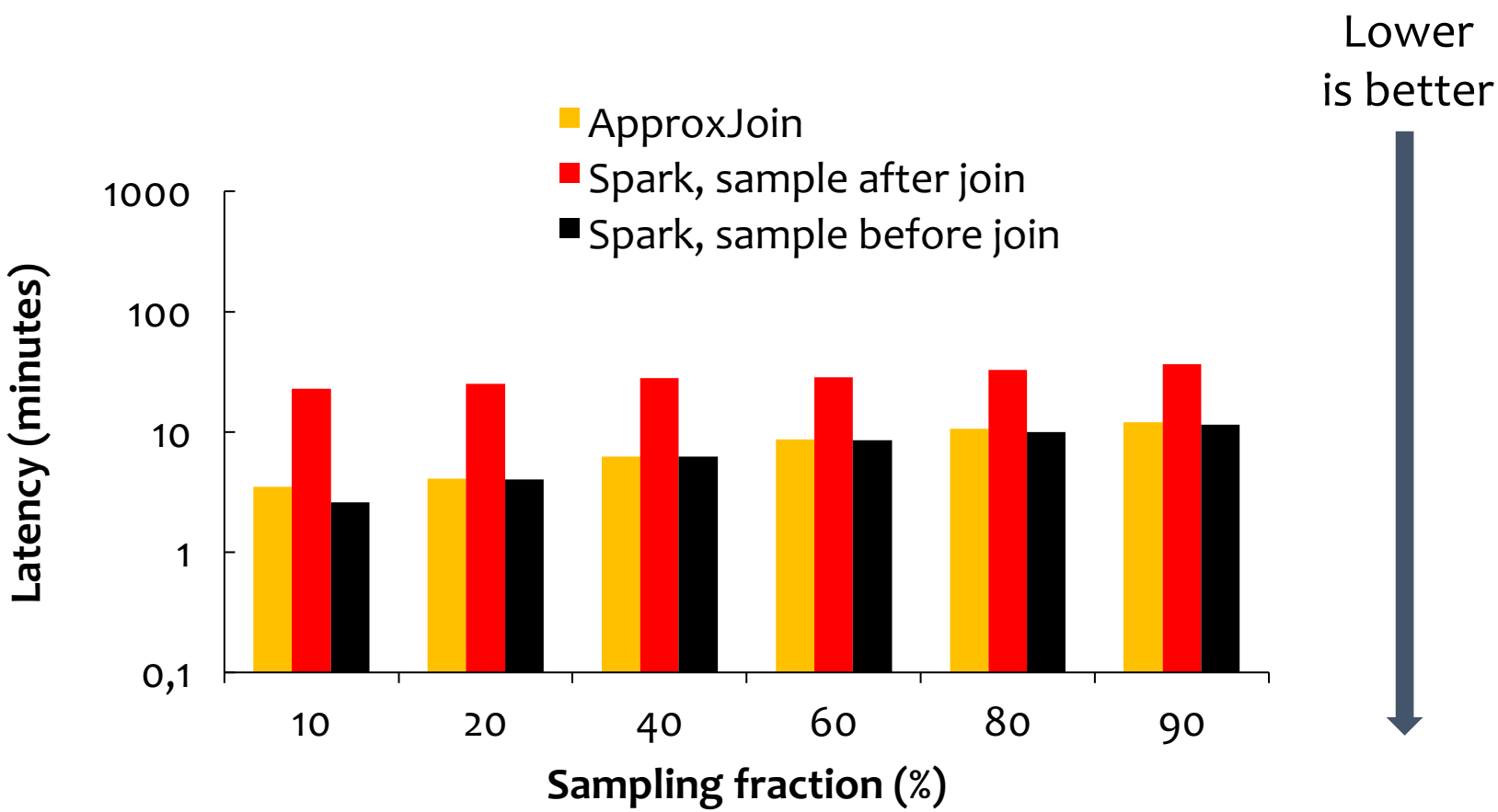
~2.6X and ~8X faster than Spark repartition join and native Spark join with overlap fraction of 1%

Shuffled Data Size



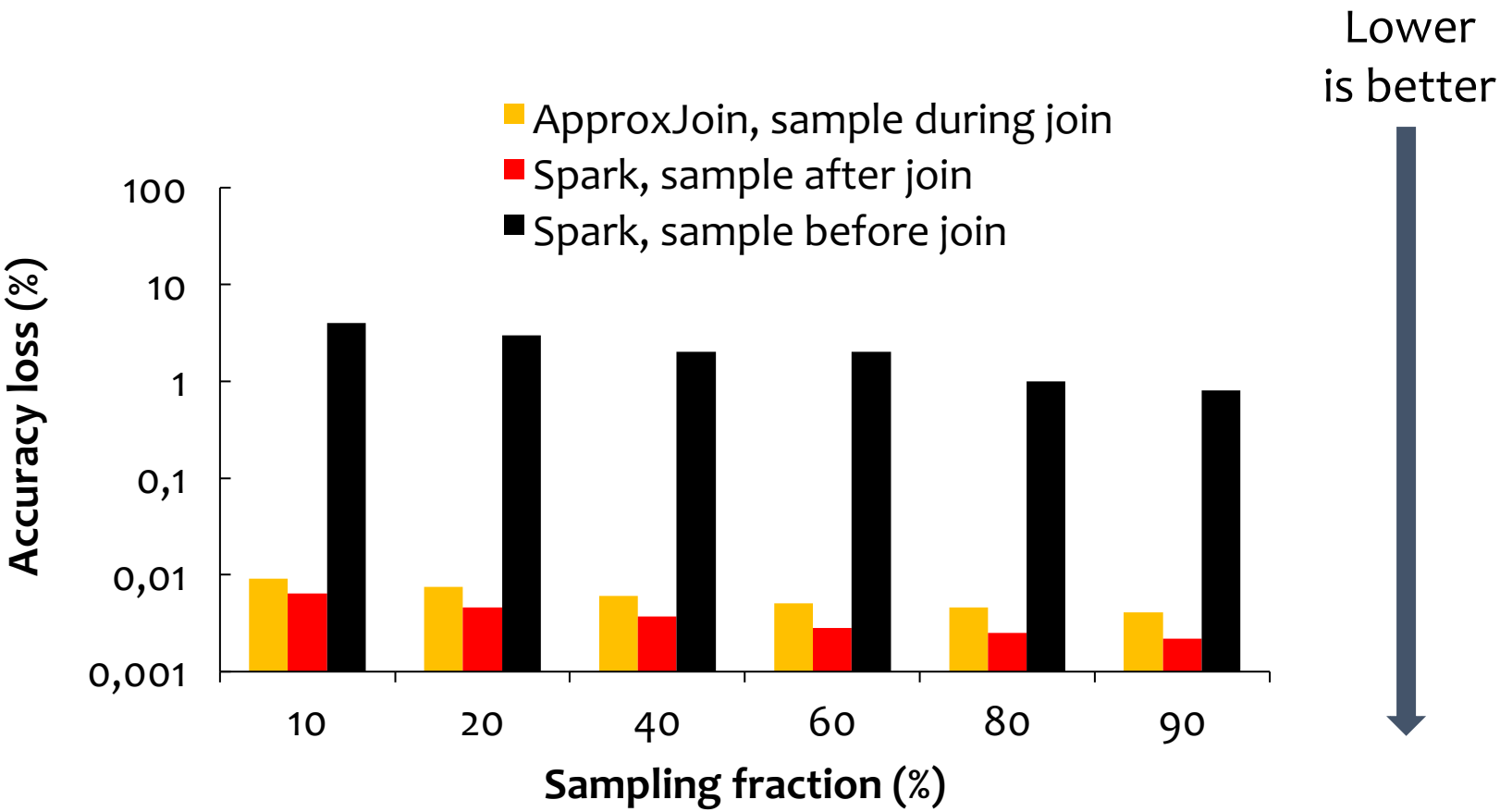
~29X and ~26X lower shuffled data size compared to Spark repartition join and native Spark join with overlap fraction of 1 %

Latency



(3X – 7X) faster than Spark with sampling *after* join
(1.01X – 1.3X) slower than Spark with sampling *before* join

Accuracy



Comparable accuracy to Spark with sampling *after* join
~42X more accurate than Spark with sampling *before* join

Outline

- ~~Motivation~~
- ~~Our work~~
- Conclusion

Conclusion

ApproxJoin: Approximate Distributed Joins

Transparent	Supports applications w/ minor code changes
Practical	Adaptive execution based on query budget
Efficient	Employs sketch & sampling techniques

Thank you!