Working with Skewed Data: The Iterative Broadcast

Rob Keevil - KnowIT

Fokko Driesprong - GoDataDriven

#EUde11

SPARK SUMMIT EUROPE 2017

About Rob Keevil

- Philosophy, Politics & Economics
- Big Data Architect
- Financial and Anti-Fraud Domains
- Scala Programmer

KNOWIT services

About Fokko Driesprong

- Software Engineering and Distributed Systems
- Data Engineer at GoDataDriven

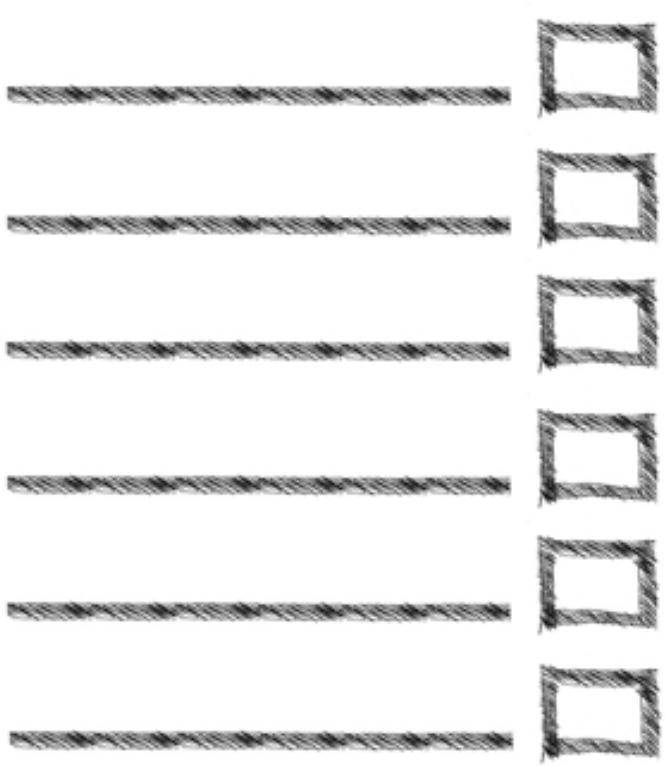
GoDataDriven
proudly part of Xebia Group

- Committer & PPMC Member, Apache Airflow
- Open source enthusiast

https://godatadriven.com/players/fokko-driesprong

Overview

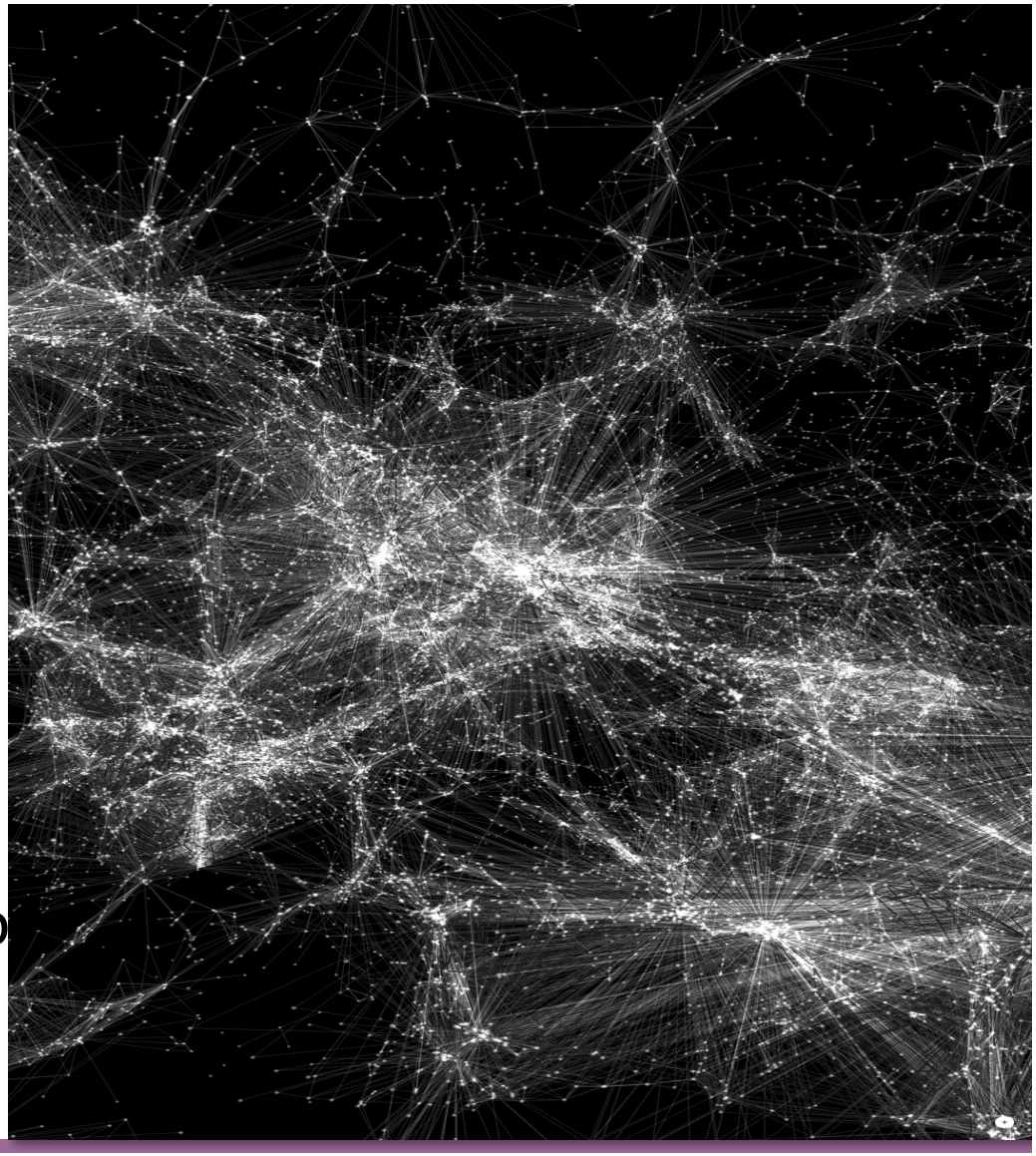
- The Skew Problem at ING
- Join Strategies in Spark
- Our Solution
- Performance Analysis
- Considerations
- Future Work



The Skew Problem at

ING

- Billions of transactions…
- Billions of accounts…
- Millions of companies…
- …with a very uneven distributio



Familiar?

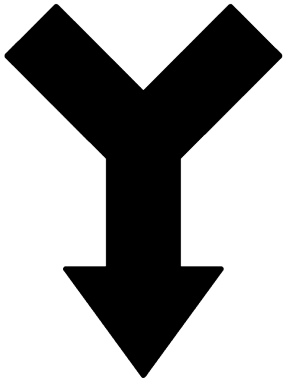
ed	Duration	Tasks: Succeeded/Total	Input	Output	Shuff
18 19:41:01	5.1 min	199/200			6.1 G

Summary Metrics for 199 Completed Tasks

Metric	Min	25th percentile	Median	75th percentile	Max
Duration	4 s	7 s	9 s	13 s	2.2 min
GC Time	0.2 s	0.4 s	0.6 s	2 s	8 s
Shuffle Read Size / Records	9.7 MB / 3542436	14.8 MB / 6104493	16.0 MB / 7704481	18.4 MB / 10158526	40.0 MB / 105456519

Join Types Native to Spark

- Sort Merge Join
- Broadcast Hash Join
- Shuffled Hash Join
- Broadcast Nested Loop Join
- Cartesian Product

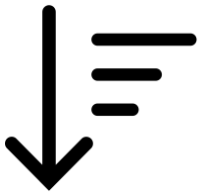


The Sort Merge Join

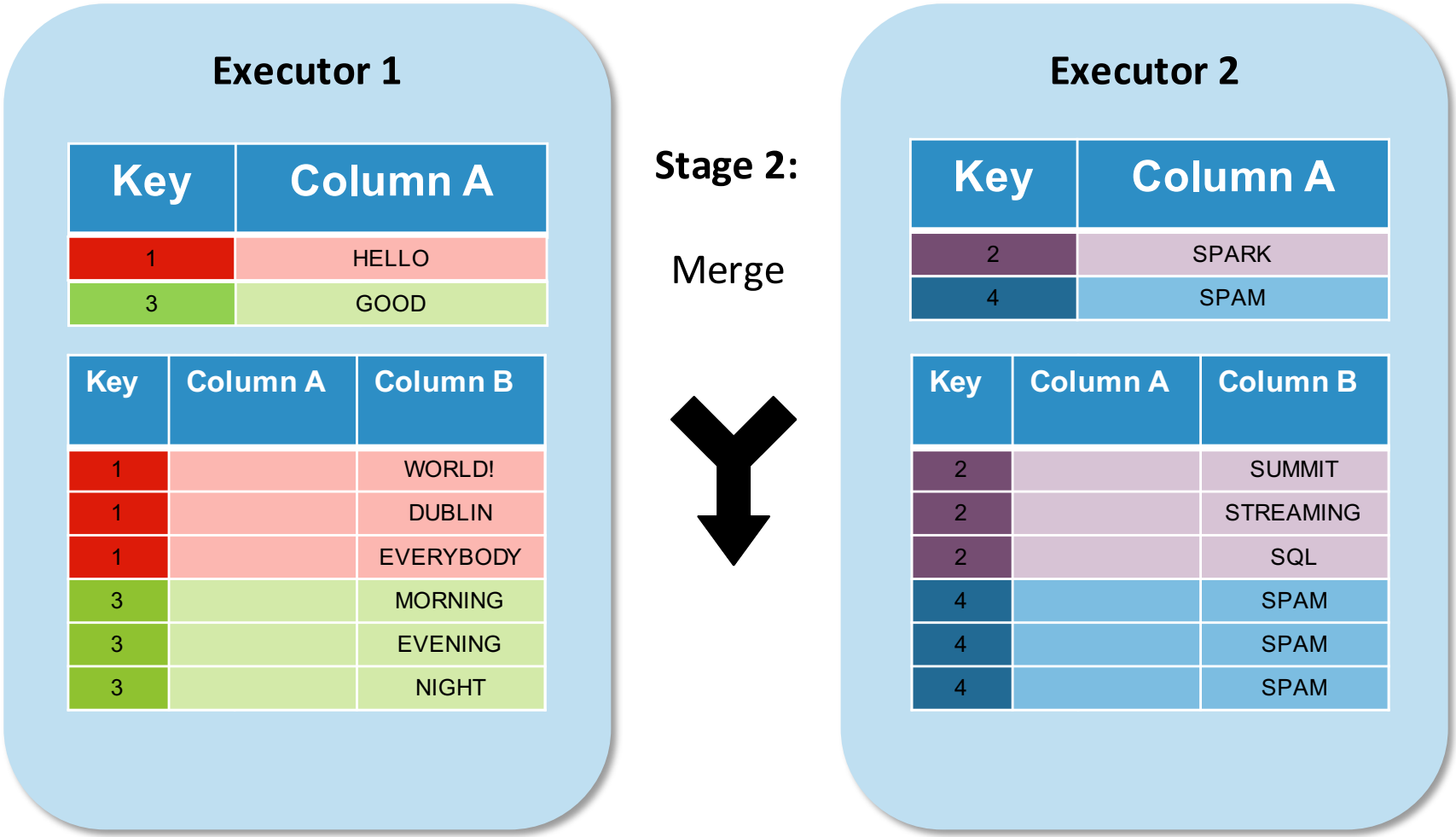
Key	Column A
1	HELLO
2	SPARK
4	SPAM
3	GOOD

Stage 1:

Determine partitions by hashing the key(s)



Key	Column B
1	WORLD!
4	SPAM
3	MORNING
2	SUMMIT
1	DUBLIN
4	SPAM
3	EVENING
2	STREAMING
1	EVERYBODY
4	SPAM
3	NIGHT
2	SQL

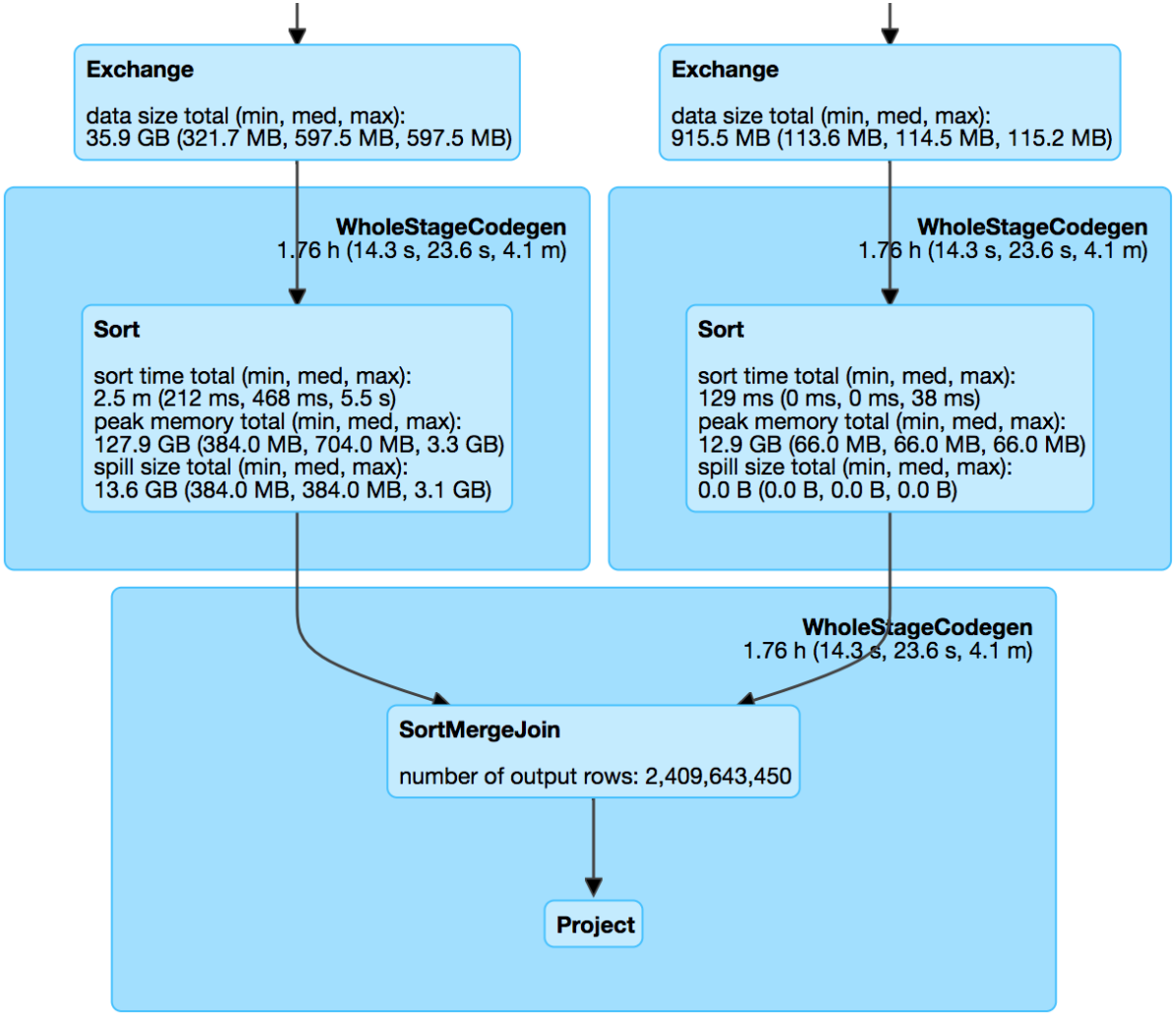

Executor 1
Key
Column A
1
HELLO
3
GOOD
Key
Column A
Column B
1
WORLD!
1
DUBLIN
1
EVERYBODY
3
MORNING
3
EVENING
3
NIGHT
Stage 2:
Merge
Executor 2
Key
Column A
2
SPARK
4
SPAM
Key
Column A
Column B
2
SUMMIT
2
STREAMING
2
SQL
4
SPAM
4
SPAM
4
SPAM

Executor 1

Key	Column A	Column B
1	HELLO	WORLD!
1	HELLO	DUBLIN
1	HELLO	EVERYBODY
3	GOOD	MORNING
3	GOOD	EVENING
3	GOOD	NIGHT

Executor 2

Key	Column A	Column B
2	SPARK	SUMMIT
2	SPARK	STREAMING
2	SPARK	SQL
4	SPAM	SPAM
4	SPAM	SPAM
4	SPAM	SPAM

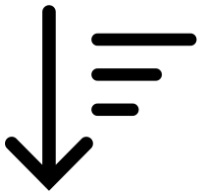

Exchange
data size total (min, med, max):
35.9 GB (321.7 MB, 597.5 MB, 597.5 MB)
Exchange
data size total (min, med, max):
915.5 MB (113.6 MB, 114.5 MB, 115.2 MB)
WholeStageCodegen
1.76 h (14.3 s, 23.6 s, 4.1 m)
WholeStageCodegen
1.76 h (14.3 s, 23.6 s, 4.1 m)
Sort
sort time total (min, med, max):
2.5 m (212 ms, 468 ms, 5.5 s)
peak memory total (min, med, max):
127.9 GB (384.0 MB, 704.0 MB, 3.3 GB)
spill size total (min, med, max):
13.6 GB (384.0 MB, 384.0 MB, 3.1 GB)
Sort
sort time total (min, med, max):
129 ms (0 ms, 0 ms, 38 ms)
peak memory total (min, med, max):
12.9 GB (66.0 MB, 66.0 MB, 66.0 MB)
spill size total (min, med, max):
0.0 B (0.0 B, 0.0 B, 0.0 B)
WholeStageCodegen
1.76 h (14.3 s, 23.6 s, 4.1 m)
SortMergeJoin
number of output rows: 2,409,643,450
Project

Regular Table

Key	Column A
1	HELLO
2	SPARK
4	SPAM
3	GOOD

Stage 1:

Determine partitions by hashing the key(s)



Skewed Table

Key	Column B
4	SPAM
4	SPAM
4	SPAM
2	SUMMIT
3	MORNING
4	SPAM
4	SPAM
4	SPAM
1	WORLD!
4	SPAM
4	SPAM
4	SPAM

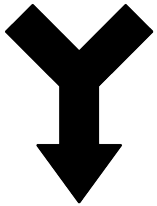
Executor 1

Key	Column A
1	HELLO
3	GOOD

Key	Column A	Column B
1		WORLD!
3		MORNING

Stage 2:

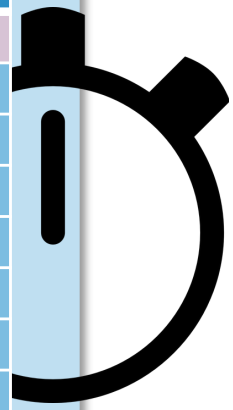
Merge



Executor 2

Key	Column A
2	SPARK
4	SPAM

Key	Column A	Column B
2		SUMMIT
4		SPAM
4		SPAM
4		SPAM
4		SPAM
4		SPAM
4		SPAM
4		SPAM
4		SPAM
4		SPAM



Executor 1

Key	Column A	Column B
1	HELLO	WORLD!
3	GOOD	MORNING

Executor 2

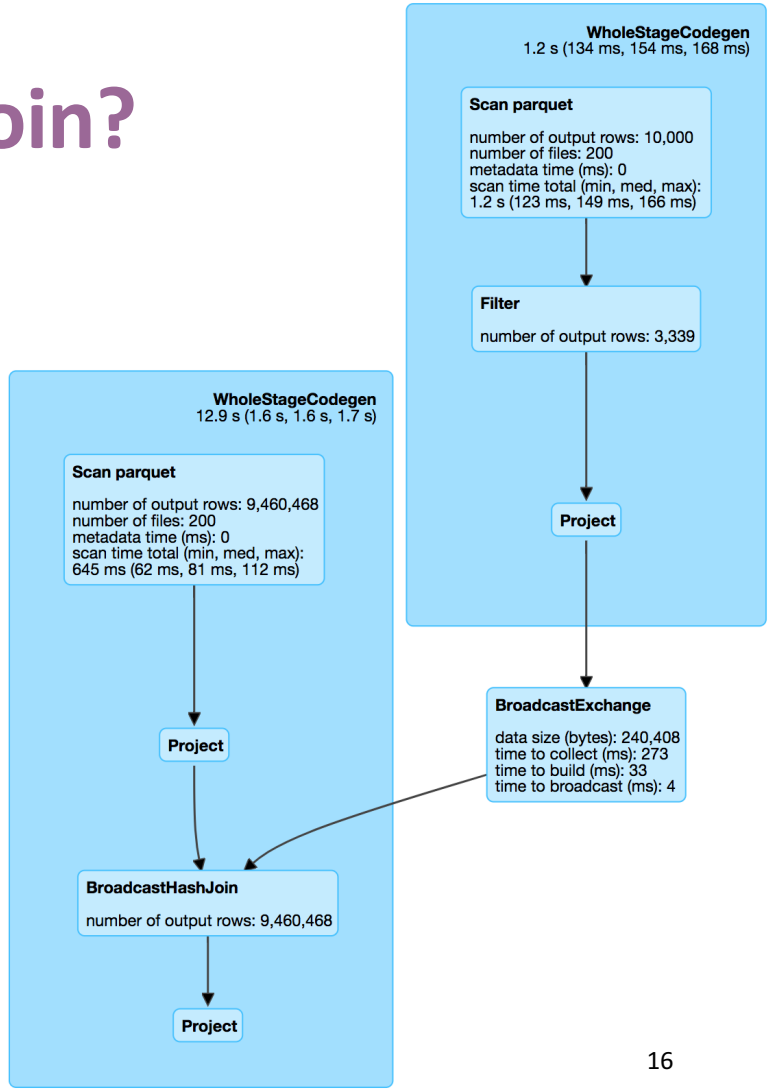
Key	Column A	Column B
2	SPARK	SUMMIT
4	SPAM	SPAM
4	SPAM	SPAM
4	SPAM	SPAM
4	SPAM	SPAM
4	SPAM	SPAM
4	SPAM	SPAM
4	SPAM	SPAM
4	SPAM	SPAM
4	SPAM	SPAM

Potential Solutions

- Give it more resources?
- Repartition the data?

What about Broadcast Hash Join?

- Leaves the partitions larger table untouched
- Copies the entire smaller table to every executor
- Iterate over the large table, and join using a HashMap



Smaller Table

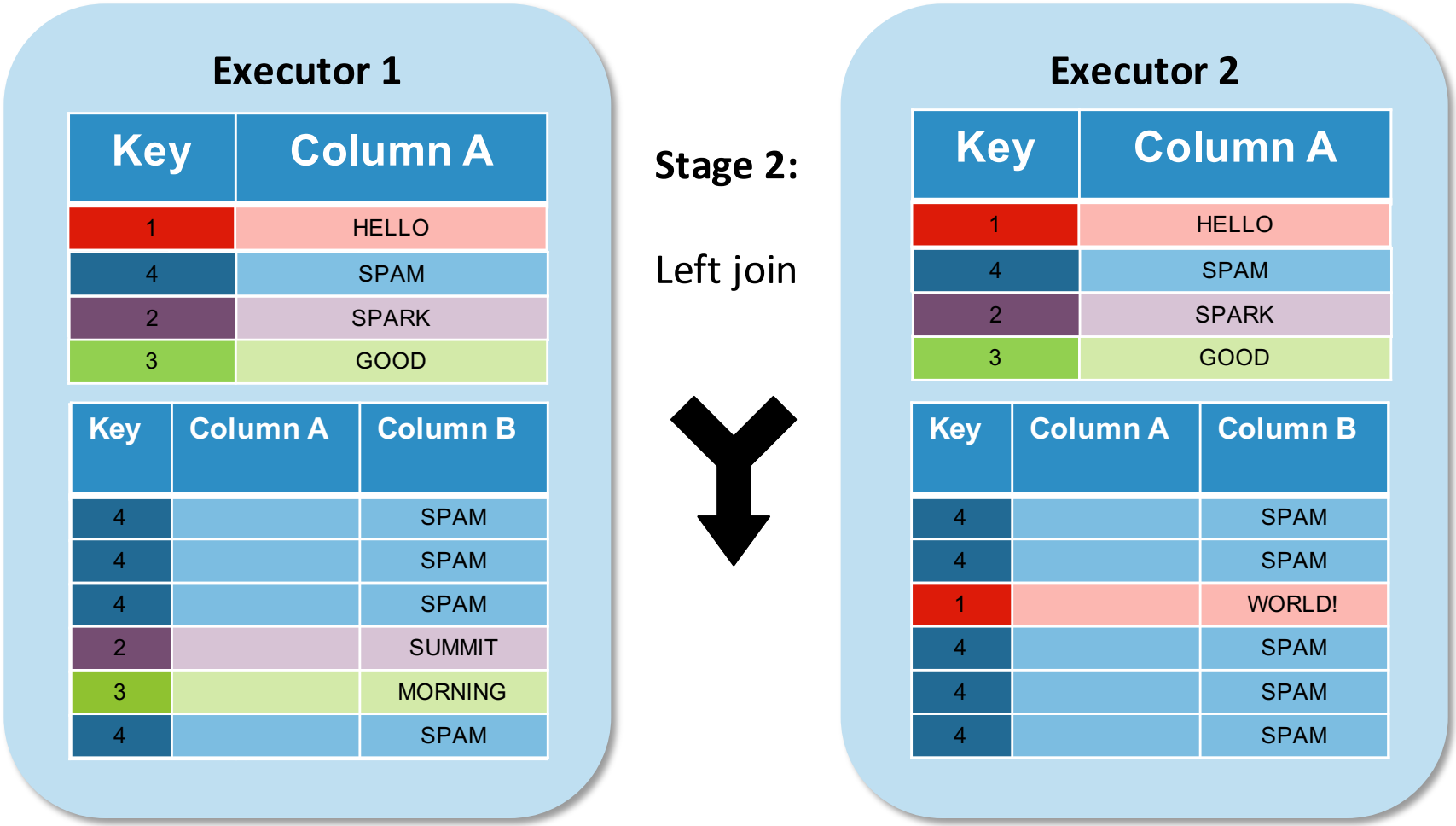
Key	Column A
1	HELLO
4	SPAM
2	SPARK
3	GOOD

Stage 1:

Broadcast smaller dataset to all executors

Larger Table

Key	Column B
4	SPAM
4	SPAM
4	SPAM
2	SUMMIT
3	MORNING
4	SPAM
4	SPAM
4	SPAM
1	WORLD!
4	SPAM
4	SPAM
4	SPAM


Executor 1
Key
Column A
1
HELLO
4
SPAM
2
SPARK
3
GOOD
Key
Column A
Column B
4
SPAM
4
SPAM
4
SPAM
2
SUMMIT
3
MORNING
4
SPAM
Stage 2:
Left join
Executor 2
Key
Column A
1
HELLO
4
SPAM
2
SPARK
3
GOOD
Key
Column A
Column B
4
SPAM
4
SPAM
1
WORLD!
4
SPAM
4
SPAM
4
SPAM

Executor 1

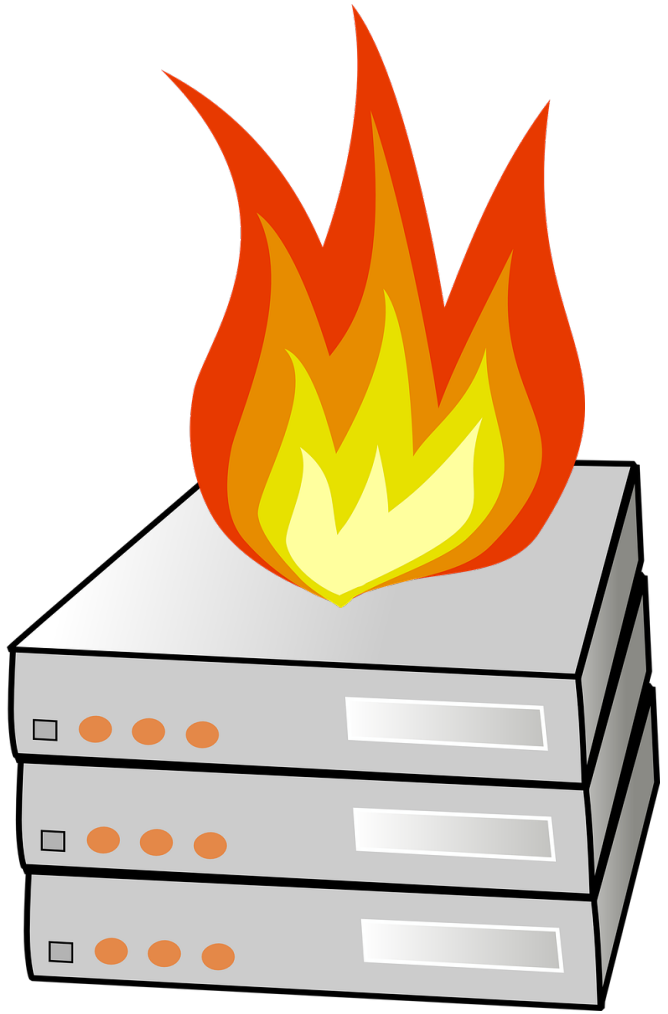
Key	Column A	Column B
4	SPAM	SPAM
4	SPAM	SPAM
4	SPAM	SPAM
2	SPARK	SUMMIT
3	GOOD	MORNING
4	SPAM	SPAM

Executor 2

Key	Column A	Column B
4	SPAM	SPAM
4	SPAM	SPAM
1	HELLO	WORLD!
4	SPAM	SPAM
4	SPAM	SPAM
4	SPAM	SPAM

But…

- …What if your “smaller” table isn’t so small?
- The whole small table needs to fit in driver memory and every executor



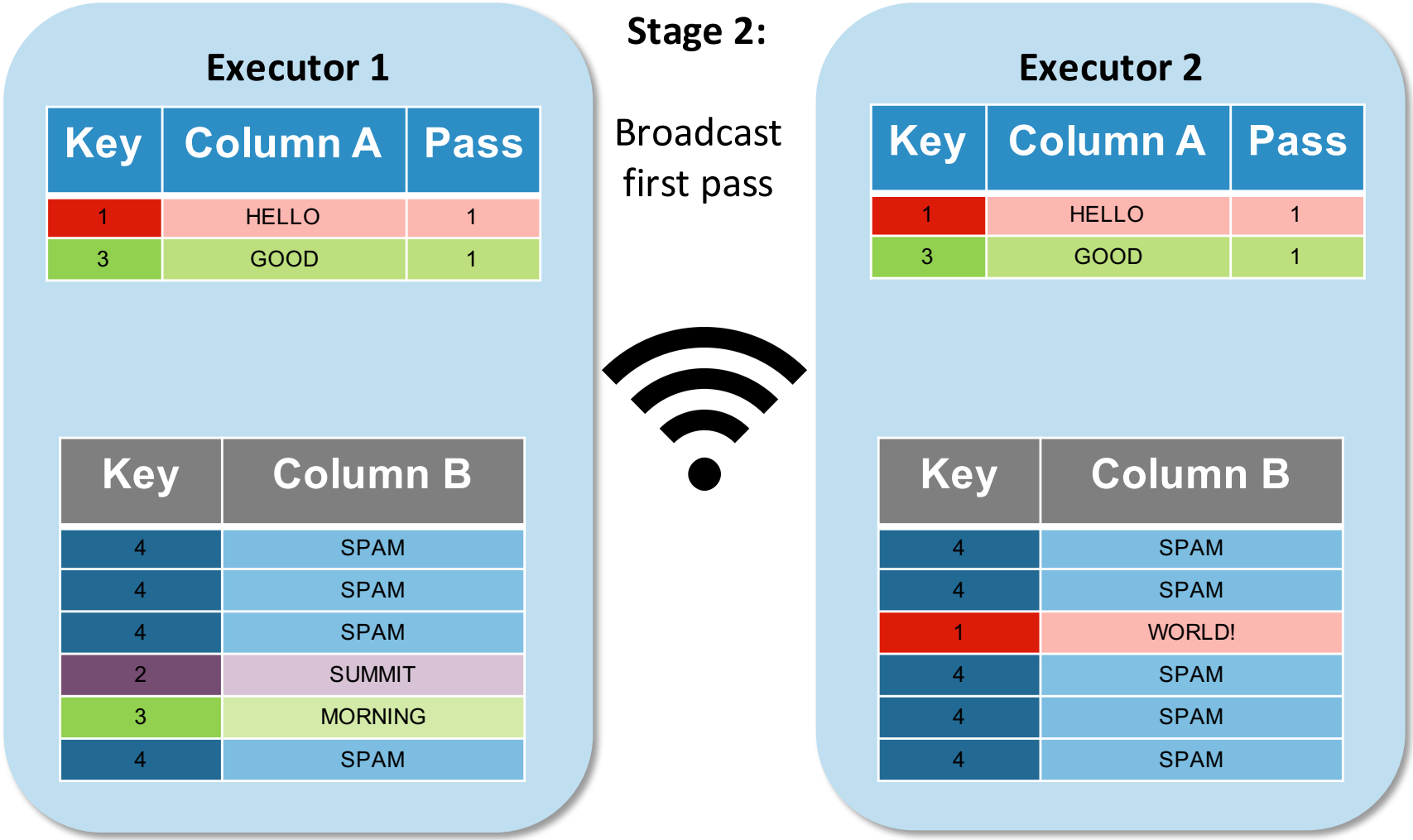
Introducing: The Iterative Broadcast!

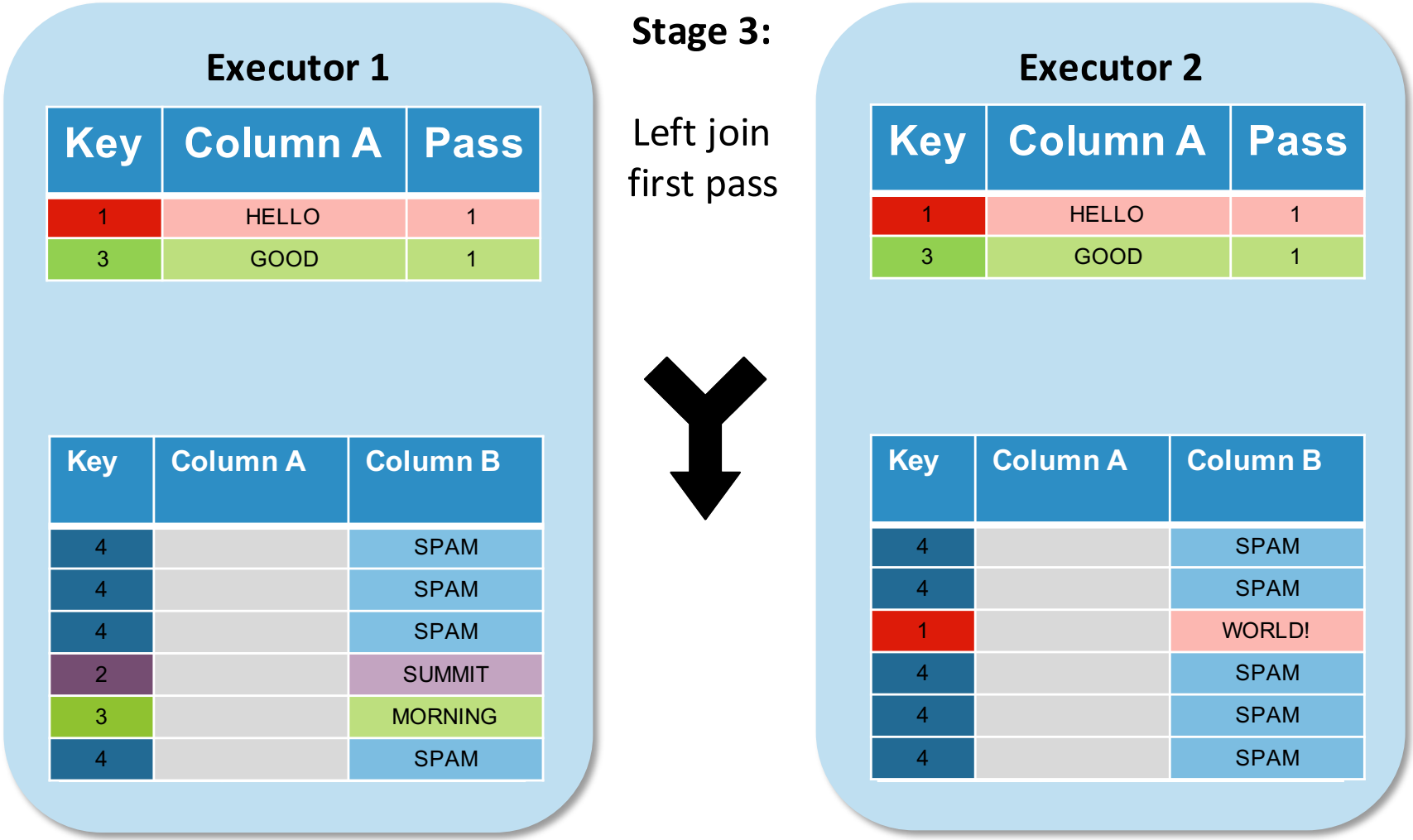
- Divide the smaller table into “passes”
- Broadcast a pass & left join to the larger dataset
- Clear the broadcast partition from memory
- Repeat until all passes are processed

Stage 1:

Assign Pass Number
to Smaller Table

Key	Column A	Pass
1	HELLO	1
4	SPAM	2
2	SPARK	2
3	GOOD	1


Executor 1
Key
Column A
Pass
1
HELLO
1
3
GOOD
1
Key
Column B
4
SPAM
4
SPAM
4
SPAM
2
SUMMIT
3
MORNING
4
SPAM
Stage 2:
Broadcast first pass
Executor 2
Key
Column A
Pass
1
HELLO
1
3
GOOD
1
Key
Column B
4
SPAM
4
SPAM
1
WORLD!
4
SPAM
4
SPAM
4
SPAM


Executor 1
Key
Column A
Pass
1
HELLO
1
3
GOOD
1
Key
Column A
Column B
4
SPAM
4
SPAM
4
SPAM
2
SUMMIT
3
MORNING
4
SPAM
Stage 3:
Left join first pass
Executor 2
Key
Column A
Pass
1
HELLO
1
3
GOOD
1
Key
Column A
Column B
4
SPAM
4
SPAM
1
WORLD!
4
SPAM
4
SPAM
4
SPAM

Stage 4:

Broadcast second pass

Executor 1

Key	Column A	Pass
2	SPARK	2
4	SPAM	2

Key	Column A	Column B
4		SPAM
4		SPAM
4		SPAM
2		SUMMIT
3	GOOD	MORNING
4		SPAM

Executor 2

Key	Column A	Pass
2	SPARK	2
4	SPAM	2

Key	Column A	Column B
4		SPAM
4		SPAM
1	HELLO	WORLD!
4		SPAM
4		SPAM
4		SPAM

Stage 5:

Left join second pass

Executor 1

Key	Column A	Pass
2	SPARK	2
4	SPAM	2

Key	Column A	Column B
4		SPAM
4		SPAM
4		SPAM
2		SUMMIT
3	GOOD	MORNING
4		SPAM

Executor 2

Key	Column A	Pass
2	SPARK	2
4	SPAM	2

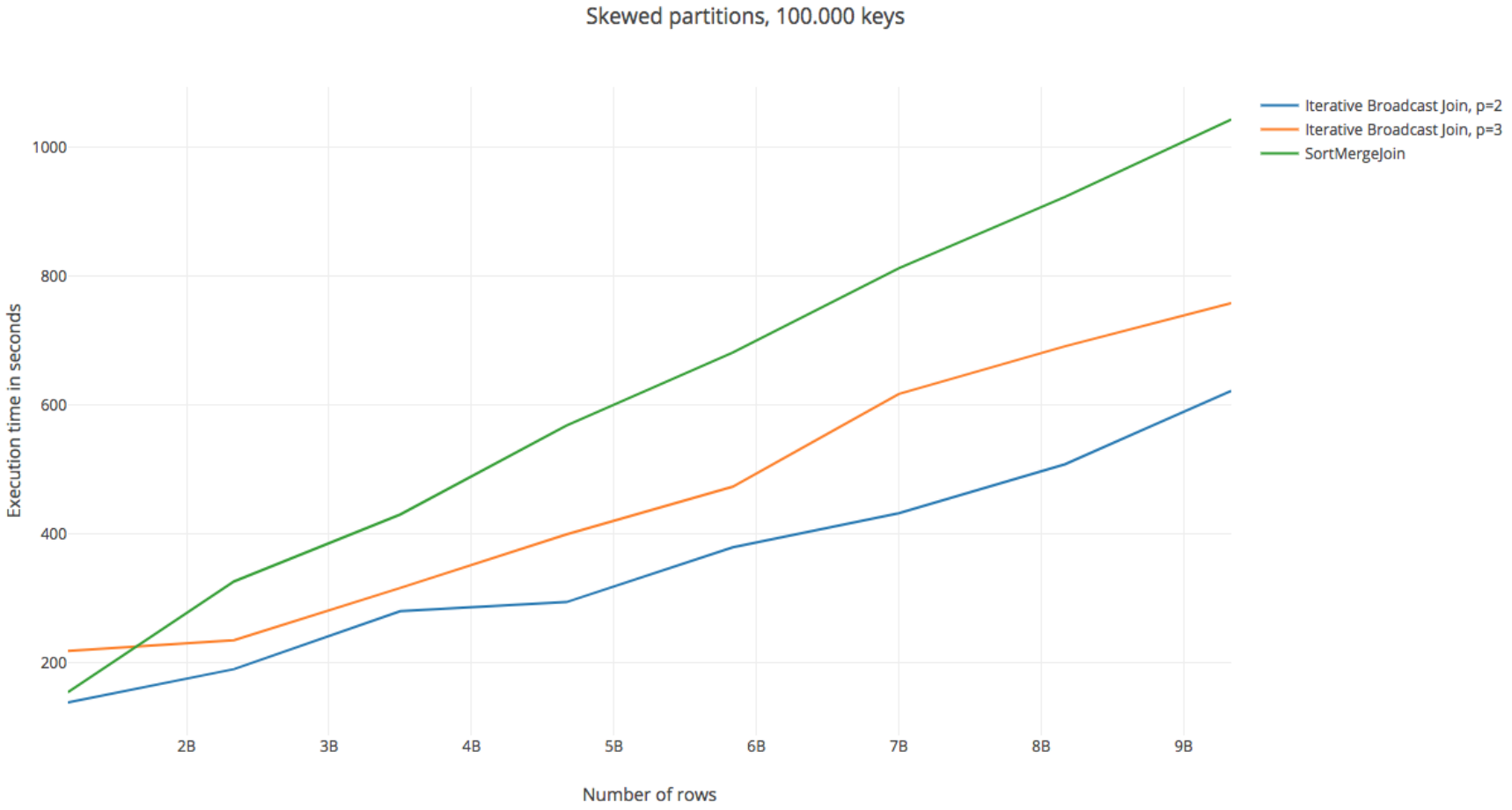
Key	Column A	Column B
4		SPAM
4		SPAM
1	HELLO	WORLD!
4		SPAM
4		SPAM
4		SPAM

Performance Analysis

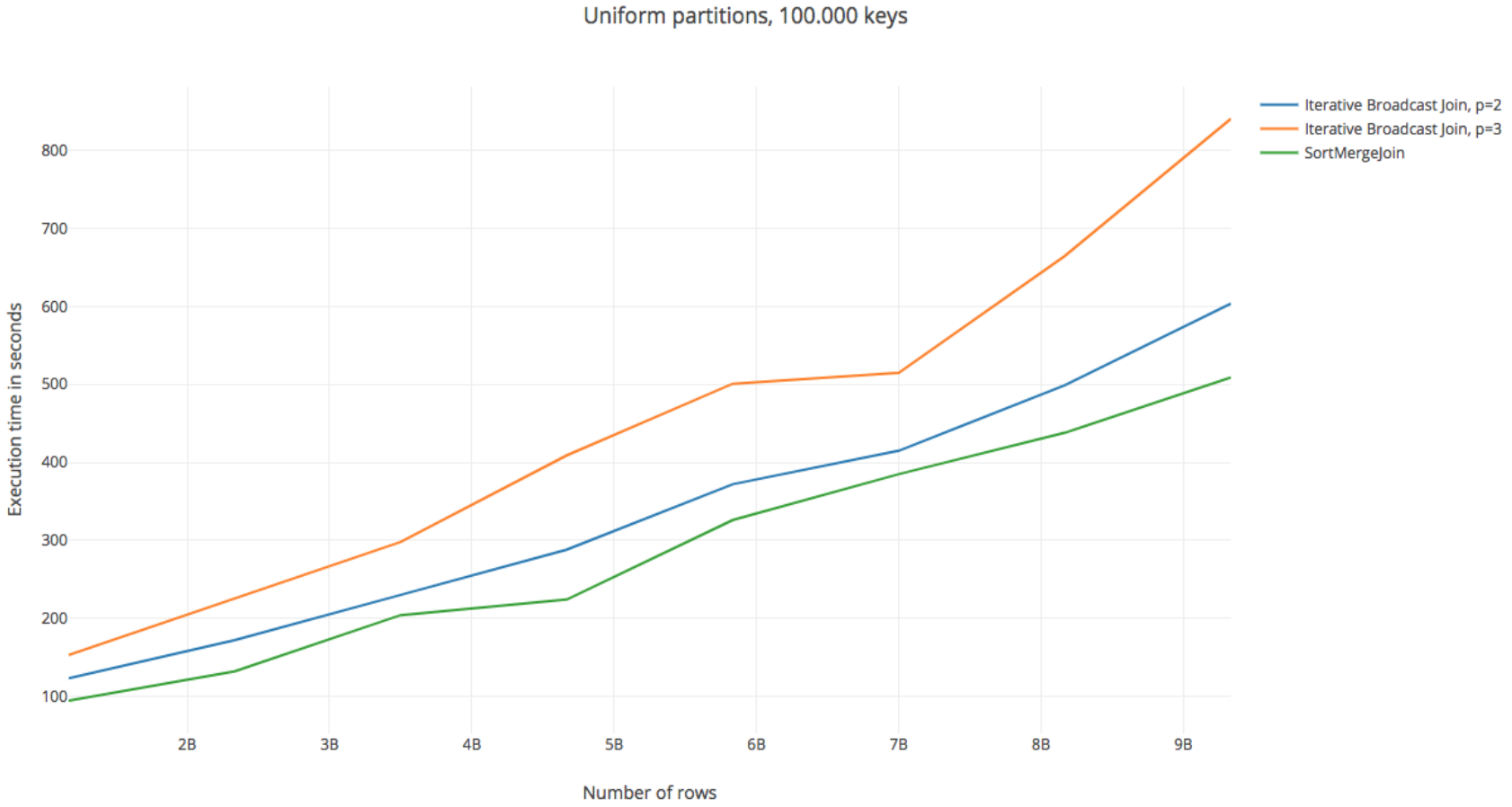
- Setup notes
  - Tests on Amazon EMR using 4x m4.4xlarge workers
  - 5 cores and 18gb memory per executor
  - Spark 2.1
  - Benchmark on GitHub

  - Skewed dataset, 10000 keys
    - Largest key 10000 rows, 2nd key 5000 rows, 3rd key 2500 rows … 10000th key 1 row
  - Uniform dataset, evenly spread

Performance Analysis – Skewed Data



Performance Analysis – Uniform Data



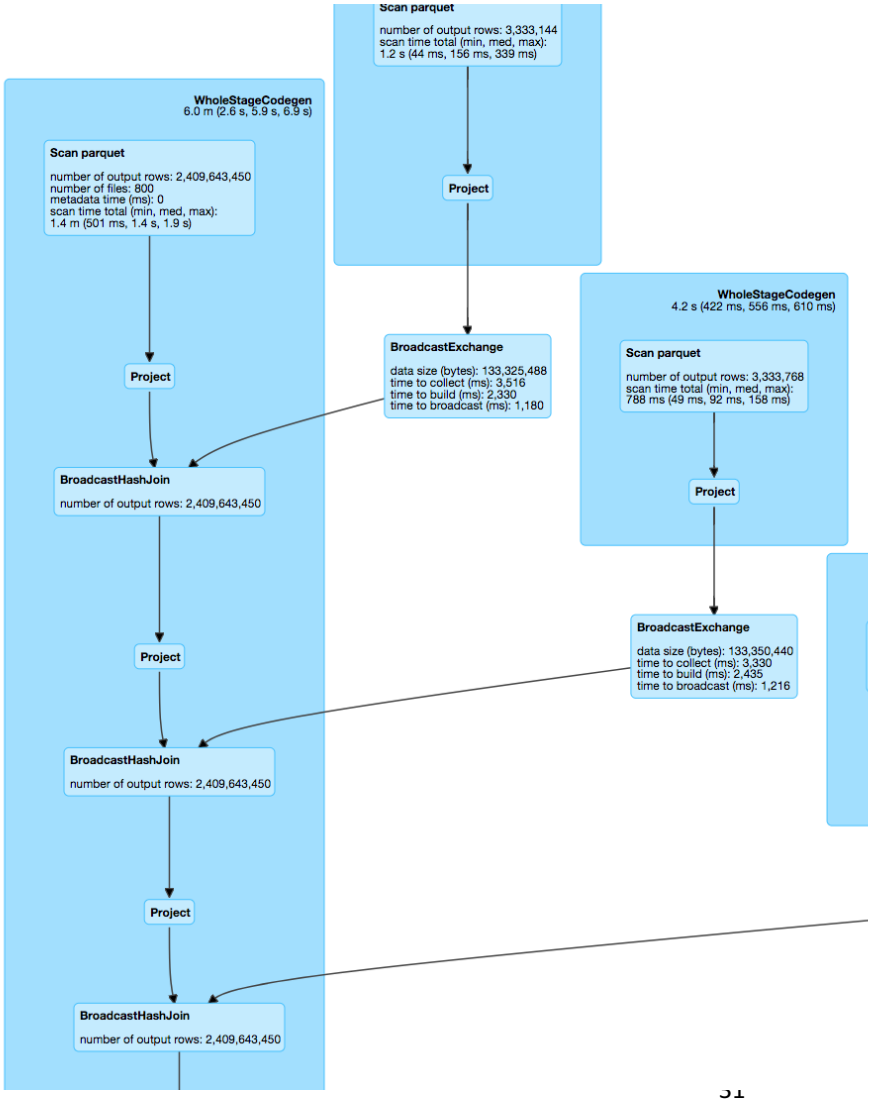
Considerations

- Hyper parameters
  - Size of the broadcast variable
  - The number of passes

- Additional complexity

Future Work

- In-memory iterations
- Native to Spark



Recap

- Skew hurts Spark parallelism and stability
- Default join types have issues with skewed data
- Iterative Broadcast Join can be used to process skewed data while maintaining parallelism

That's All Folks!

https://github.com/godatadriven/iterative-broadcast-join