DIFF: A Relational Interface for Large-Scale Data Explanation

Firas Abuzaid, Peter Kraft, Sahaana Suri, Edward Gan, Eric Xu, Atul Shenoy, Asvin Ananthanarayan, John Sheu, Erik Meijer, Xi Wu, Jeff Naughton, Peter Bailis, Matei Zaharia

STANFORD DAWN

Explaining trends in high-volume data remains a fundamental challenge for today's data analysts

- Example: tracking a mobile app's user engagement
  - Product manager wants to determine why the number of daily active users for her app declined in the last week
  - Has to inspect thousands of possible causes: user demographics, device and location metadata, temporal/seasonal factors
  - All combinations of these, too!

- With traditional OLAP and BI tools, PM has to manually search through all possible combos (i.e., series of `GROUP BY` and `CUBE` queries)

Explanation Engines automate this search process

- MacroBase [SIGMOD 2017]
- Scorpion [VLDB 2013]
- Data X-Ray [SIGMOD 2015]
- [Roy and Suciu, SIGMOD 2014]

- Identify features that are statistically significant in moving a particular metric that the user cares about
  - `device_make=“Apple”, os_version=“9.0.1”, app_version=“v50”` is 2x more likely to have lower DAU

Today’s Explanation Engines are lacking two things

1. Interoperability
   - » Analysts want to search for explanations as part of a larger workflow; the explanation query is only part of the pipeline (e.g., ETL, traditional OLAP queries, visualizations)
   - » Current explanation engines are usually standalone tools

2. Scalability
   - » Analysts want to their explanation queries to be interactive; explanation engines don’t scale to today’s data volumes

Our work: The DIFF operator

1. Declarative relational operator
   - » Unifies the core functionality of several explanation engines
   - » We can capture the semantics of MacroBase/Data X-Ray/Scorpion queries using a single interface

2. Logical and Physical Optimizations for DIFF
   - » Use `DIFF` query plan and apply new query optimization techniques

3. Scalable implementation of DIFF
   - » Integrate `DIFF` as an extension to MacroBase—MacroBase SQL
   - » Single node and Spark implementations

An example workflow using DIFF

Crash Logs

timestamp	app_version	device_type	os	crash
07-21-19 00:01	v1	iPhone X	11.0	false
...	...	...	...	...
07-28-19 12:00	v2	Galaxy S9	8.0	true
...	...	...	...	...
09-04-18 23:59	v3	HTC One	8.0	false

Analyzing crash logs with DIFF

```
SELECT * FROM
      (SELECT * FROM logs WHERE crash = true) crash_logs
DIFF
      (SELECT * FROM logs WHERE crash = false) success_logs
ON app_version, device_type, os
COMPARE BY risk_ratio >= 2.0, support >= 0.05;
```

Analyzing crash logs with DIFF

```
SELECT * FROM
     (SELECT * FROM logs WHERE crash = true) crash_logs
DIFF
     (SELECT * FROM logs WHERE crash = false) success_logs
ON app_version, device_type, os
COMPARE BY risk_ratio >= 2.0, support >= 0.05;
```

app_version	device_type	os	risk_ratio	support
v1	-	-	10.5	15%
v2	iPhone X	-	7.25	30%
v3	Galaxy S9	11.0	9.75	20%

Analyzing crash logs with DIFF

```
SELECT * FROM
     (SELECT * FROM logs WHERE crash = true) crash_logs
DIFF
     (SELECT * FROM logs WHERE crash = false) success_logs
ON app_version, device_type, os
COMPARE BY risk_ratio >= 2.0, support >= 0.05;
```

app_version	device_type	os	risk_ratio	support
v1	-	-	10.5	15%
v2	iPhone X	-	7.25	30%
v3	Galaxy S9	11.0	9.75	20%

explanation

Compare week to week using DIFF

```
SELECT * FROM
     (SELECT * FROM logs WHERE crash = true and timestamp
BETWEEN 08-28-  18 AND 09-04-18) this_week
DIFF
     (SELECT * FROM logs WHERE crash = true and timestamp
BETWEEN 08-21-  18 AND 08-28-18) last_week
ON app_version, device_type, os
COMPARE BY risk_ratio >= 2.0, support >= 0.05;
```

app_version	device_type	os	risk_ratio	support
v3	Galaxy S9	11.0	20.0	75%

DIFF operator has found successful use cases in many industrial and academic workloads

Elements of the DIFF operator

```
SELECT * FROM
     (SELECT * FROM logs WHERE crash = true) crash_logs
DIFF
     (SELECT * FROM logs WHERE crash = false) success_logs
ON app_version, device_type, os
COMPARE BY risk_ratio >= 2.0, support >= 0.05
```

Elements of the DIFF operator

```
                 SELECT * FROM
test relation    (SELECT * FROM logs WHERE crash = true) crash_logs
                 DIFF
control relation (SELECT * FROM logs WHERE crash = false) success_logs
                 ON app_version, device_type, os
                 COMPARE BY risk_ratio >= 2.0, support >= 0.05
```

Elements of the DIFF operator

```
SELECT * FROM
     (SELECT * FROM logs WHERE crash = true) crash_logs
DIFF
     (SELECT * FROM logs WHERE crash = false) success_logs
ON app_version, device_type, os  dimensions
COMPARE BY risk_ratio >= 2.0, support >= 0.05
```

Elements of the DIFF operator

```
SELECT * FROM
     (SELECT * FROM logs WHERE crash = true) crash_logs
DIFF
     (SELECT * FROM logs WHERE crash = false) success_logs
ON app_version, device_type, os
COMPARE BY risk_ratio >= 2.0, support >= 0.05
```

difference metrics

Elements of the DIFF operator

```
SELECT * FROM
     (SELECT * FROM logs WHERE crash = true) crash_logs
DIFF
     (SELECT * FROM logs WHERE crash = false) success_logs
ON app_version, device_type, os
COMPARE BY risk_ratio >= 2.0, support >= 0.05 MAX ORDER 3;
```

difference metrics

max order of combinations

Difference metrics allow us to generalize to other explanation engines

» MacroBase: Risk Ratio, Support

» Data X-Ray: Diagnosis Cost

» Scorpion: Influence

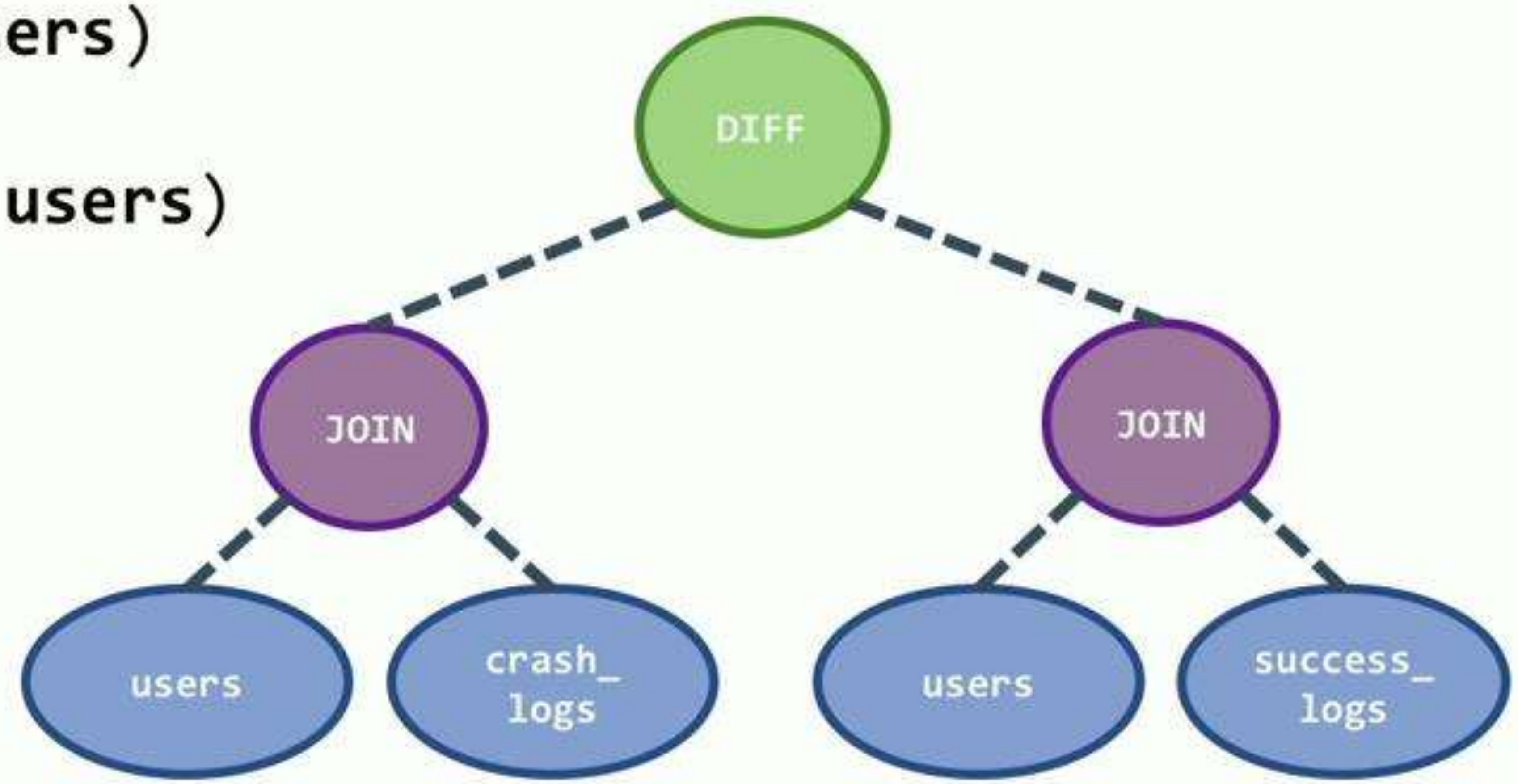
» [Roy and Suciu, SIGMOD 2014]: Intervention

» Frequent Itemset Mining: Support

Logical Optimizations: DIFF-JOIN Predicate Pushdown

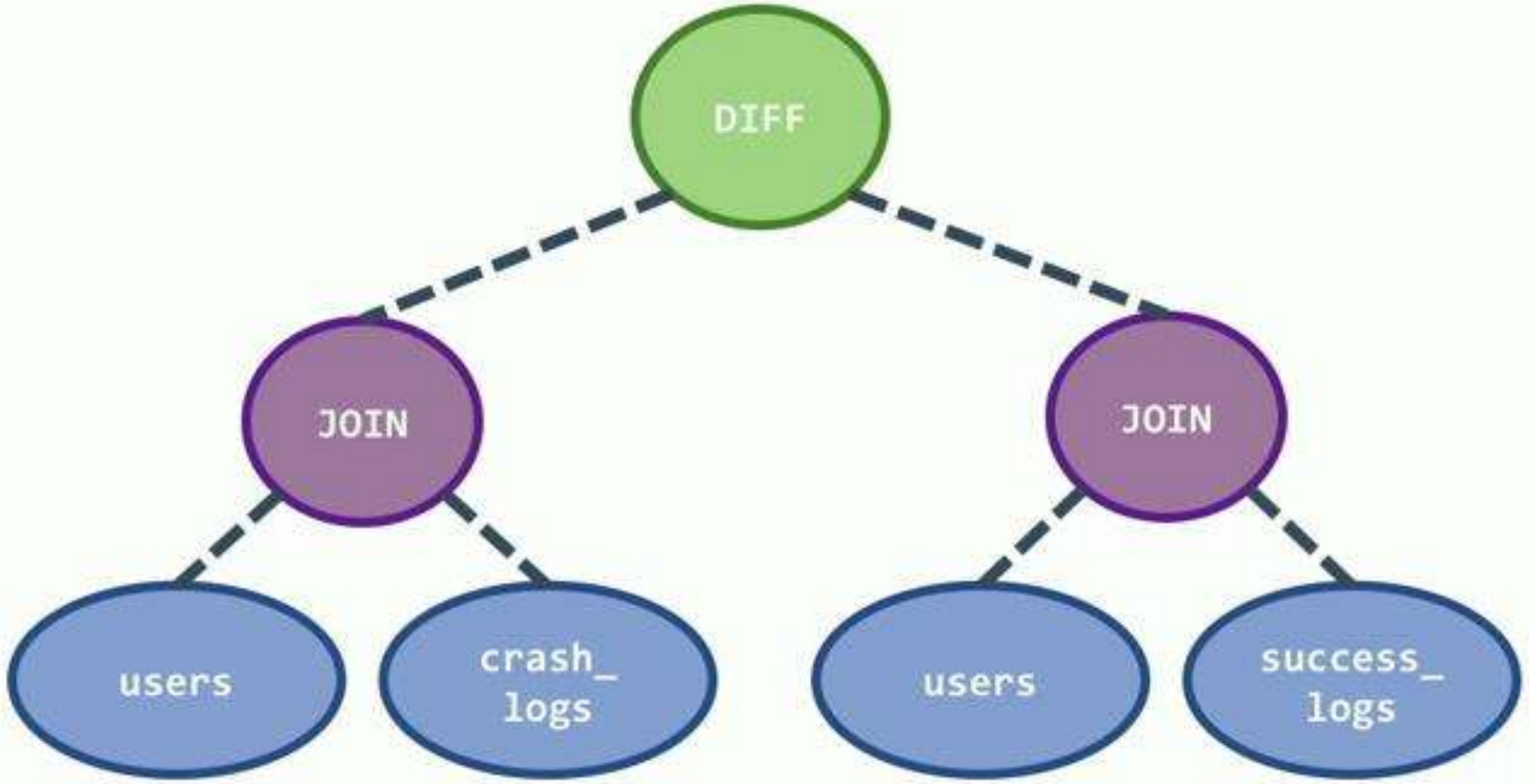
```
SELECT * FROM
      (crash_logs NATURAL JOIN users)
DIFF
      (success_logs NATURAL JOIN users)
ON app_version, device_type, os
COMPARE BY risk_ratio >= 2.0,
           support >= 0.05;
```

Query Plan



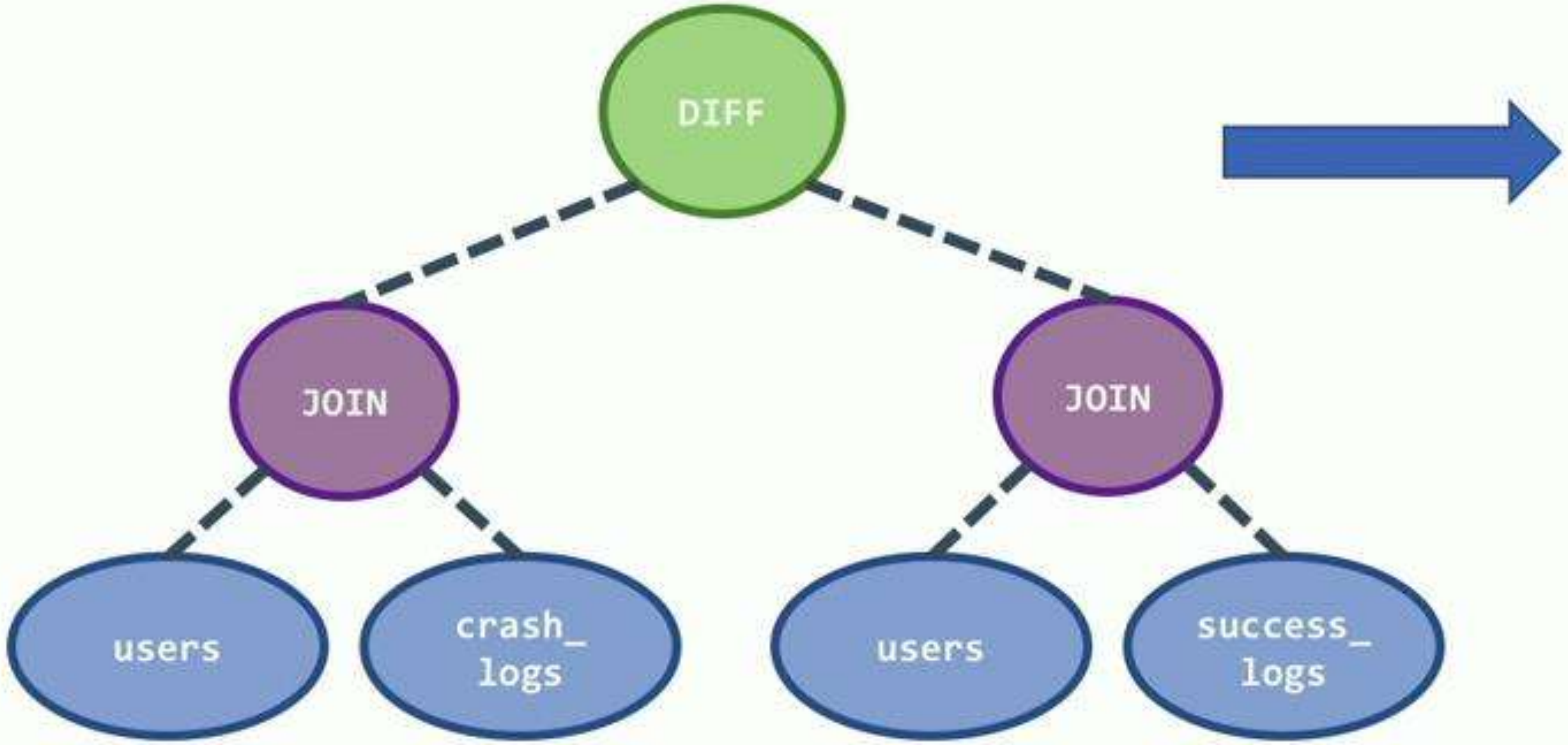
Idea: push the DIFF operator below the JOIN operator

Query Plan



Idea: push the DIFF operator below the JOIN operator

Query Plan

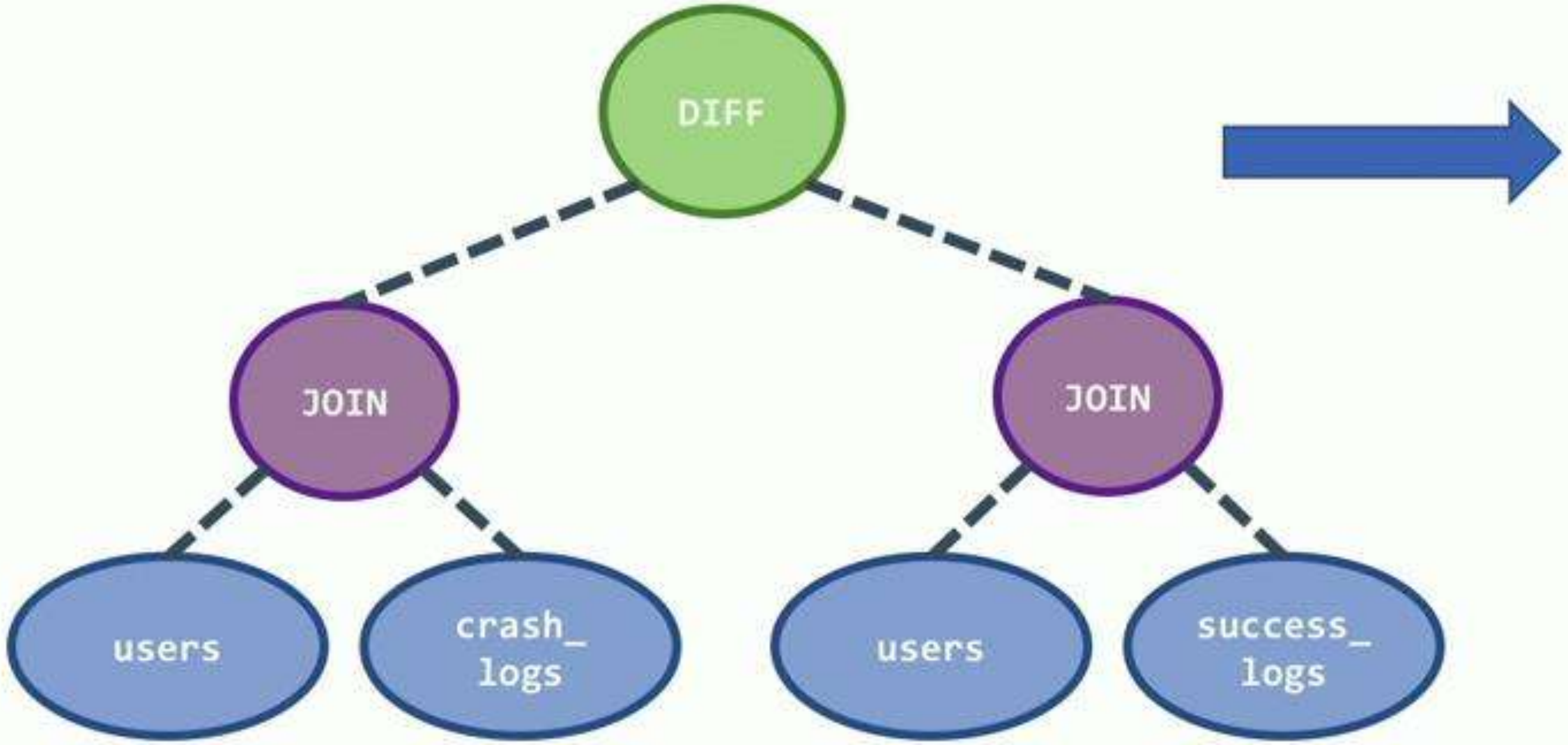


Adaptive DIFF-JOIN Algorithm

1. Evaluate **`DIFF`** on foreign key column of `crash_logs` and `success_logs` to find candidate keys

Idea: push the DIFF operator below the JOIN operator

Query Plan

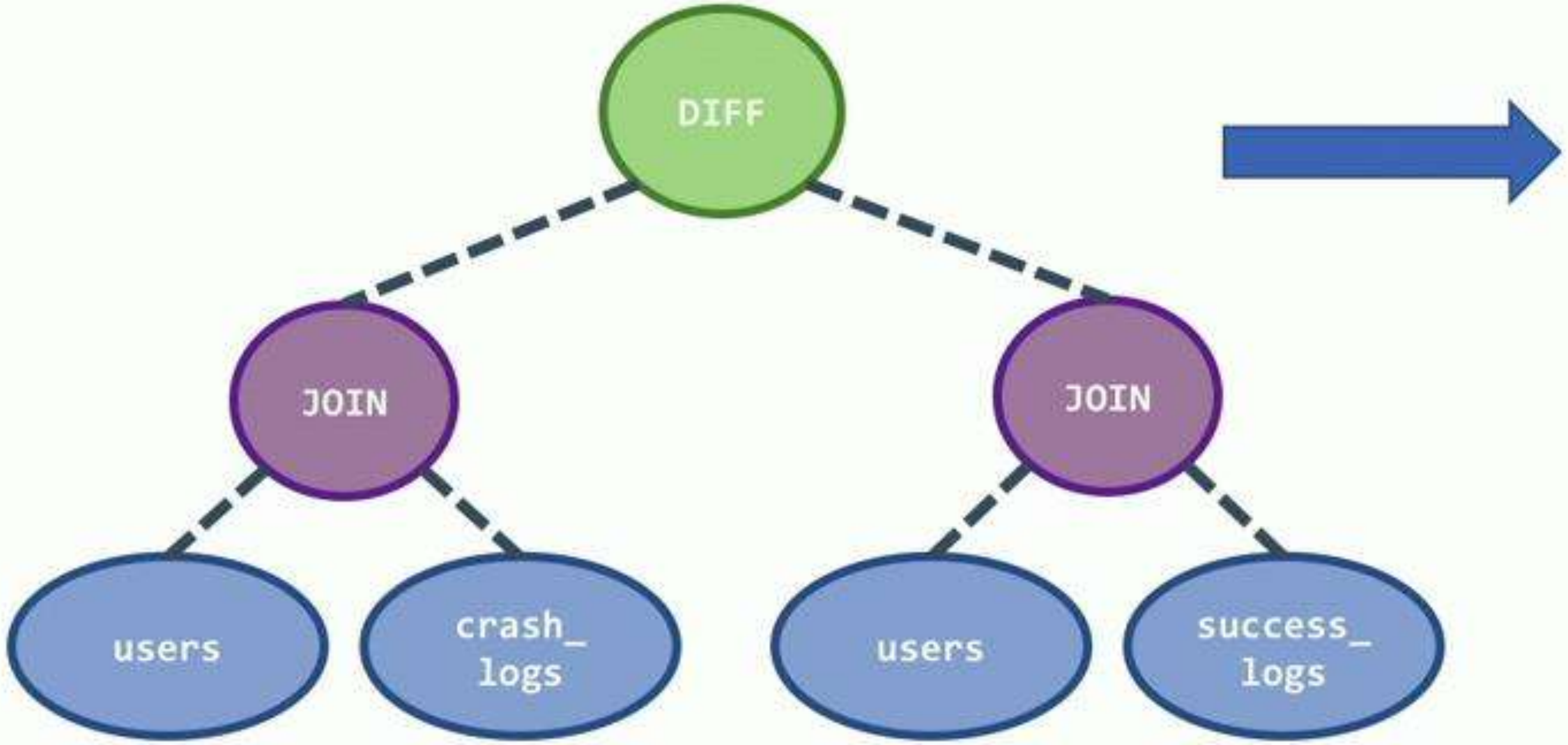


Adaptive DIFF-JOIN Algorithm

1. Evaluate **`DIFF`** on foreign key column of `crash_logs` and `success_logs` to find candidate keys
   - » If output is large, abort and use naïve approach

Idea: push the DIFF operator below the JOIN operator

Query Plan



Adaptive DIFF-JOIN Algorithm

1. Evaluate **`DIFF`** on foreign key column of `crash_logs` and `success_logs` to find candidate keys
   - » If output is large, abort and use naïve approach

2. Semi-join candidate keys with users to find candidate values

3. Evaluate `DIFF` on candidate values

Additional optimization:
prune search space using **functional dependencies**

Physical Optimizations for DIFF

1. Low-support attribute values pruned, remaining values **dictionary-encoded**
2. Low-cardinality columns bitmap-encoded based on **cost model**
3. Encoded data stored in **columnar format** for higher cache locality
4. Embarrassingly **parallel APriori** explores feature combinations for explanations

Implementation in MacroBase SQL

- Single node
  - Fork of the original MacroBase repo
  - 11.5 K lines of Java code
  - `DIFF` + core subset of ANSI SQL supported: `SELECT`, `WHERE`, `ORDER BY`, `JOIN`, `LIMIT`

- Spark
  - Integrated with Spark DataFrame API
  - For standard SQL queries, MB SQL defers execution to Spark SQL and Catalyst optimizations
  - All `DIFF` queries are i) optimized using our custom Catalyst rules, and ii) translated to equivalent Spark operators (e.g., `map`, `filter`, `reduce`, `groupBy`)
  - Pruning optimization to reduce communication costs
  - 1.6K lines of Java code

Implementation in MacroBase SQL

Open source:
https://macrobase.stanford.edu/docs

- Single node
  - Fork of the original MacroBase repo
  - 11.5 K lines of Java code
  - `DIFF` + core subset of ANSI SQL supported: `SELECT`, `WHERE`, `ORDER BY`, `JOIN`, `LIMIT`
- Spark
  - Integrated with Spark DataFrame API
  - For standard SQL queries, MB SQL defers execution to Spark SQL and Catalyst optimizations
  - All `DIFF` queries are i) optimized using our custom Catalyst rules, and ii) translated to equivalent Spark operators (e.g., `map`, `filter`, `reduce`, `groupBy`)
  - Pruning optimization to reduce communication costs
  - 1.6K lines of Java code

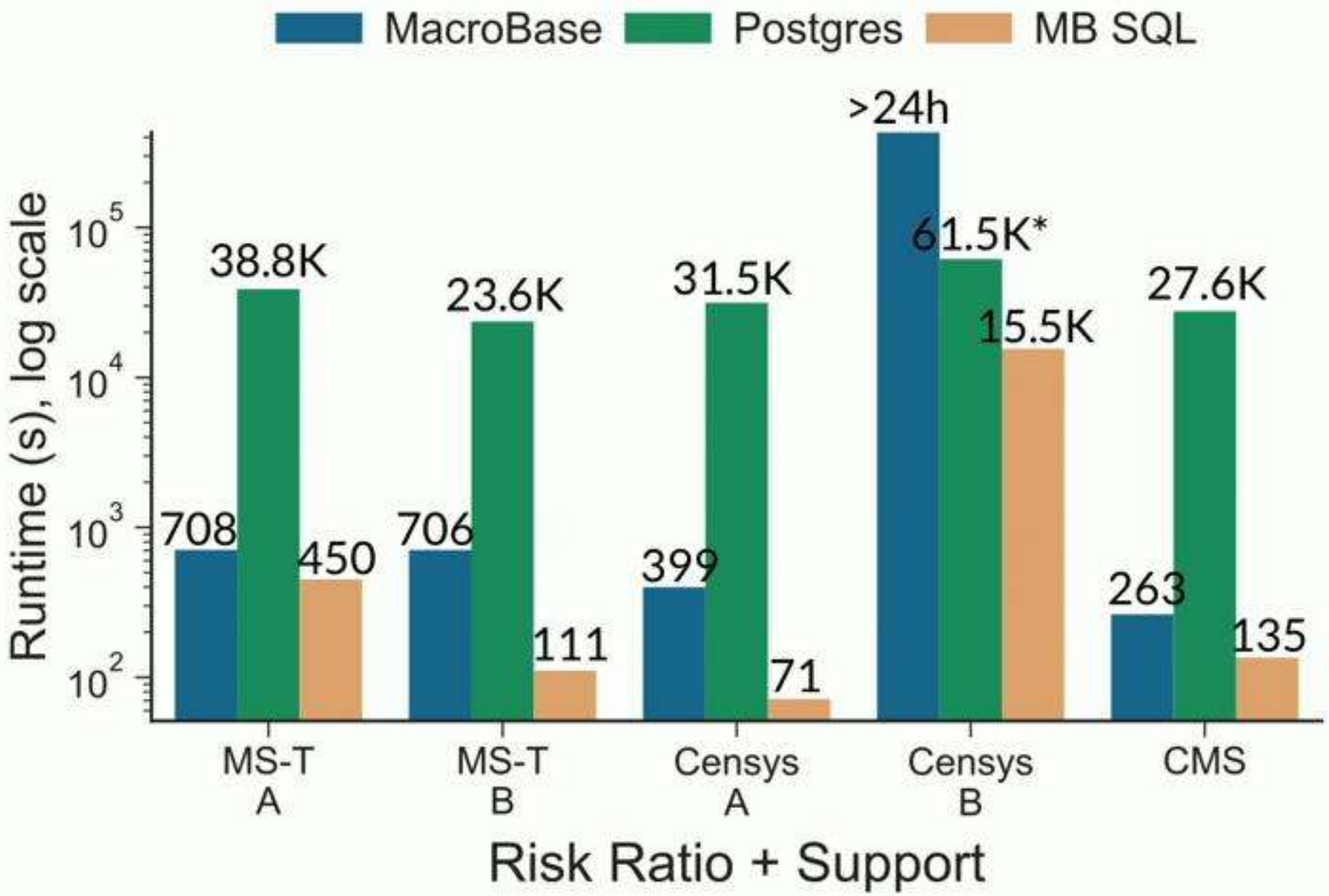
Evaluation: Single Node

Dataset	File size (CSV)	# rows	# columns	# 3-order combos
Censys A	3.6 GB	20 M	17	19.5 M
Censys B	2.6 GB	8 M	102	814.9 M
MS-Telemetry A	17 GB	50 M	13	73.4 M
MS-Telemetry B	13 GB	19 M	15	1.3 B
Center for Medicare Studies	7.7 GB	15 M	16	63.8 M

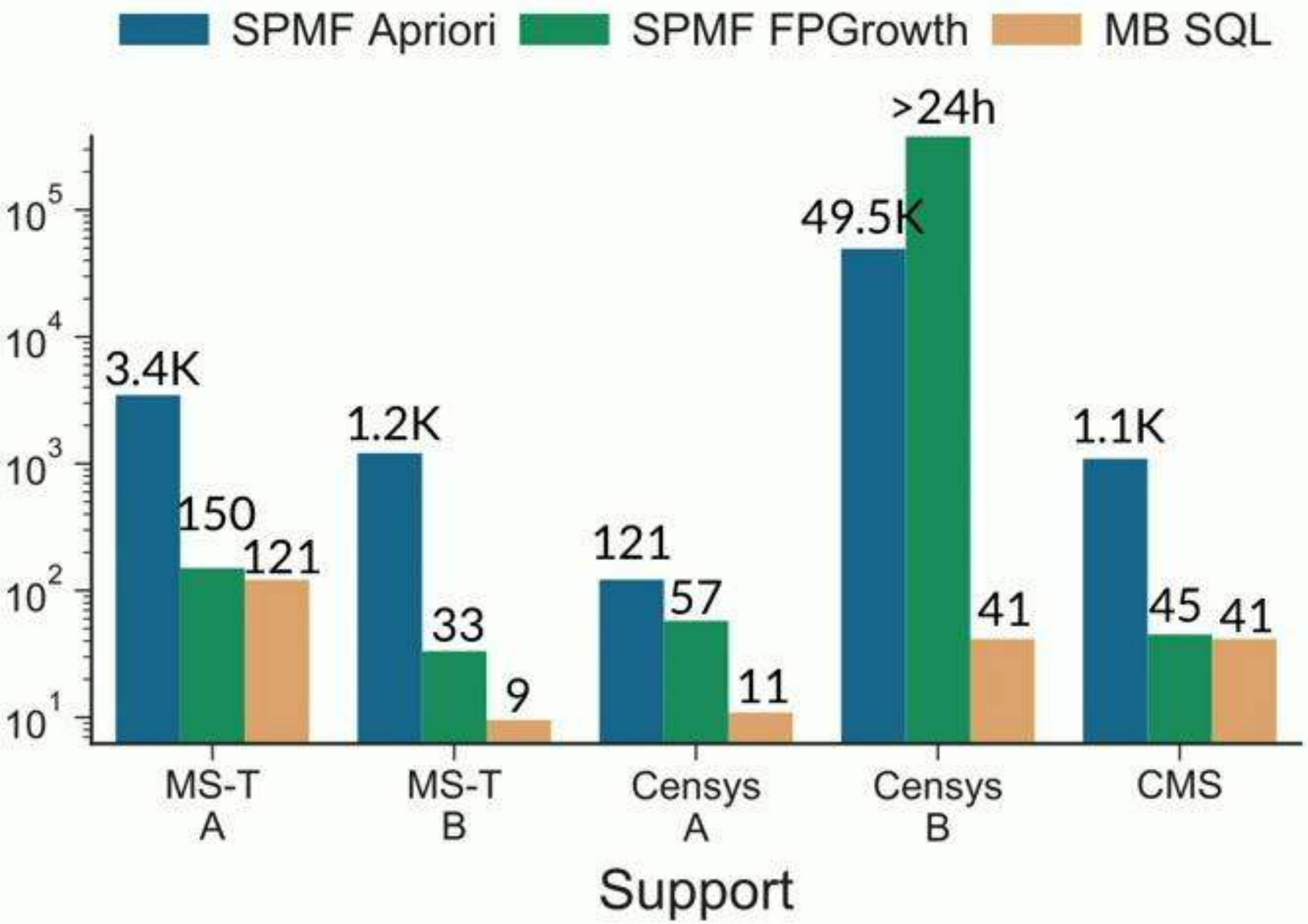
- Intel Xeon E5-2690 v4 CPU (Broadwell), 512GB of RAM
- Benchmarked MacroBase SQL against:
  - MacroBase
  - Postgres
  - RSExplain
  - Data X-Ray
  - SPMF Frequent Itemset Miners (Apriori, FPGrowth)

- For each dataset, execute `DIFF` query and measure end-to-end runtime (ingest time omitted)
  - When possible, `DIFF` query obtained from production workflow
  - Difference Metrics={Risk Ratio, Support}, `MAX ORDER` = 3, and dimensions are always categorical features

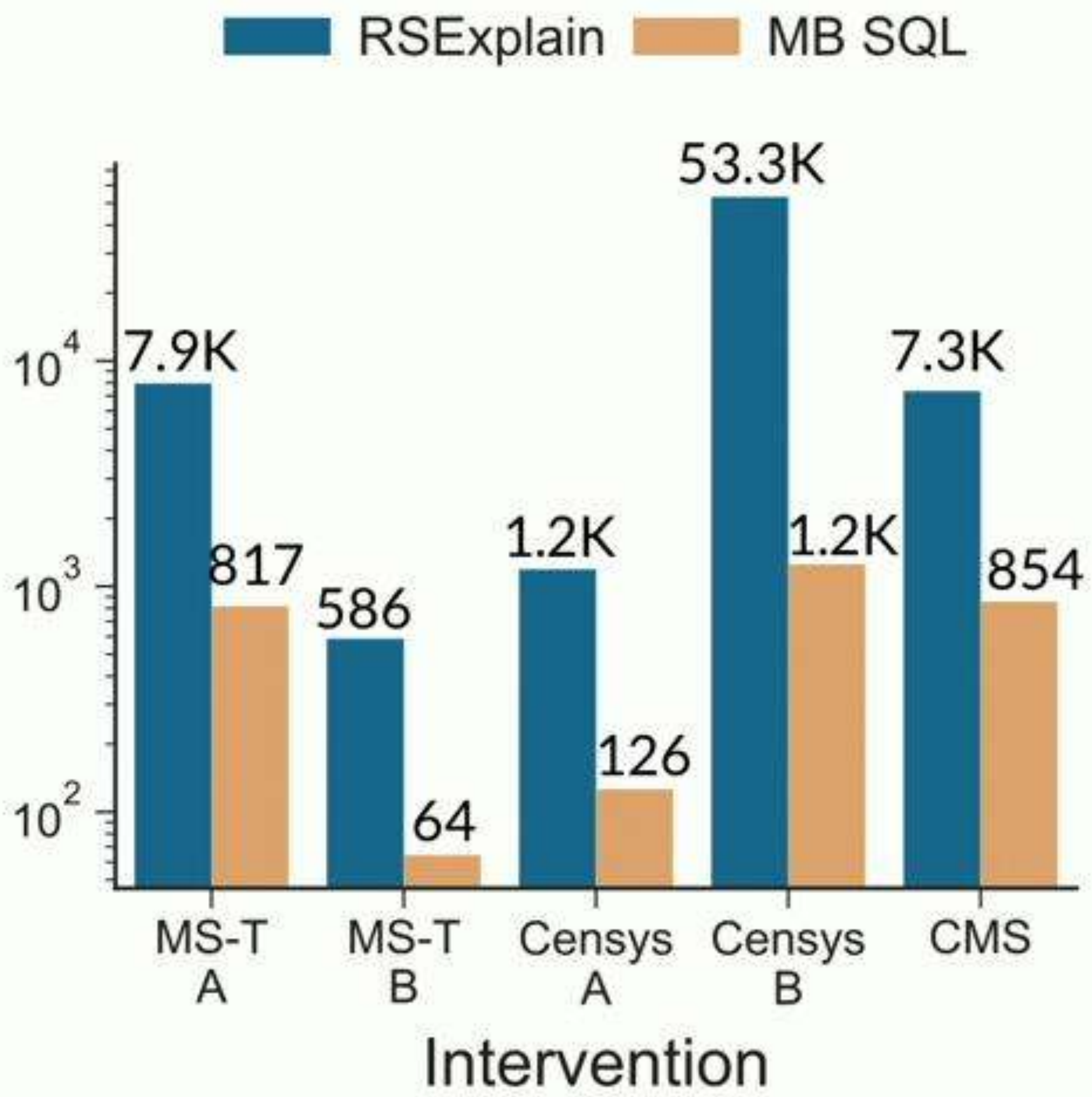
MB SQL outperforms both MacroBase and Postgres across DIFF queries

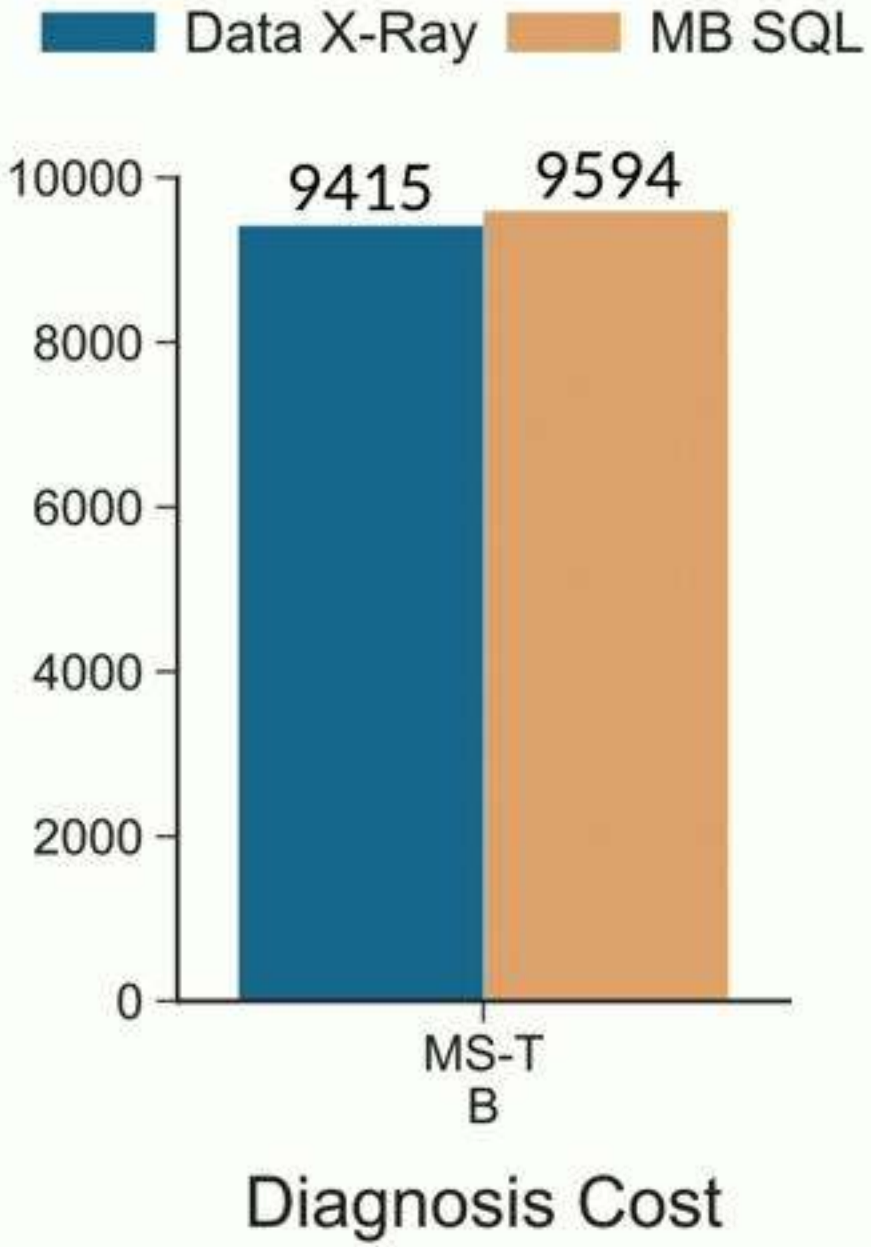


MB SQL also outperforms Frequent Itemset Miners

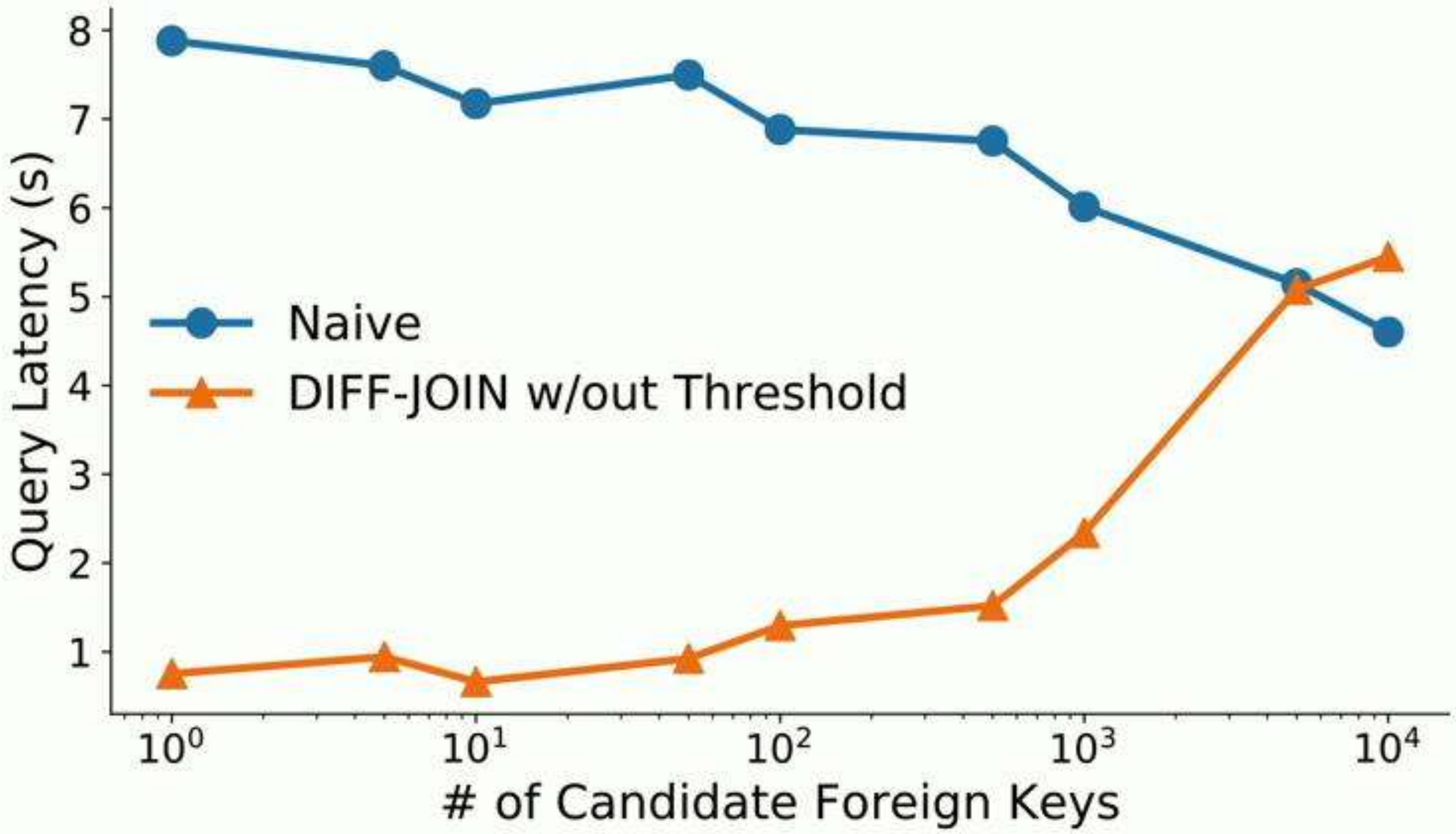


MB SQL generalizes to other Explanation Engines—with good performance

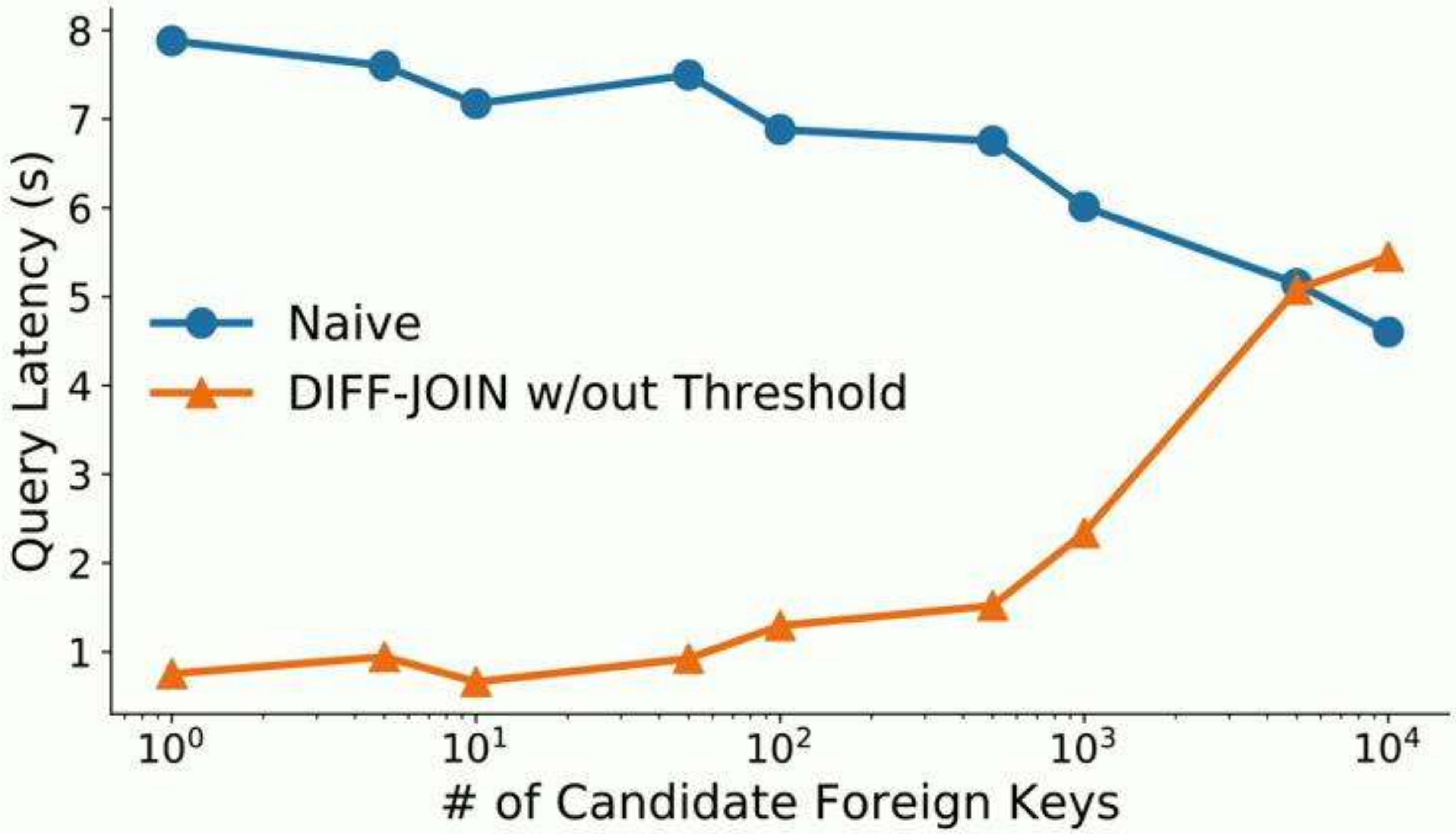




Adaptive Predicate Pushdown works for a broad range of DIFF-JOIN queries



Adaptive Predicate Pushdown works for a broad range of DIFF-JOIN queries



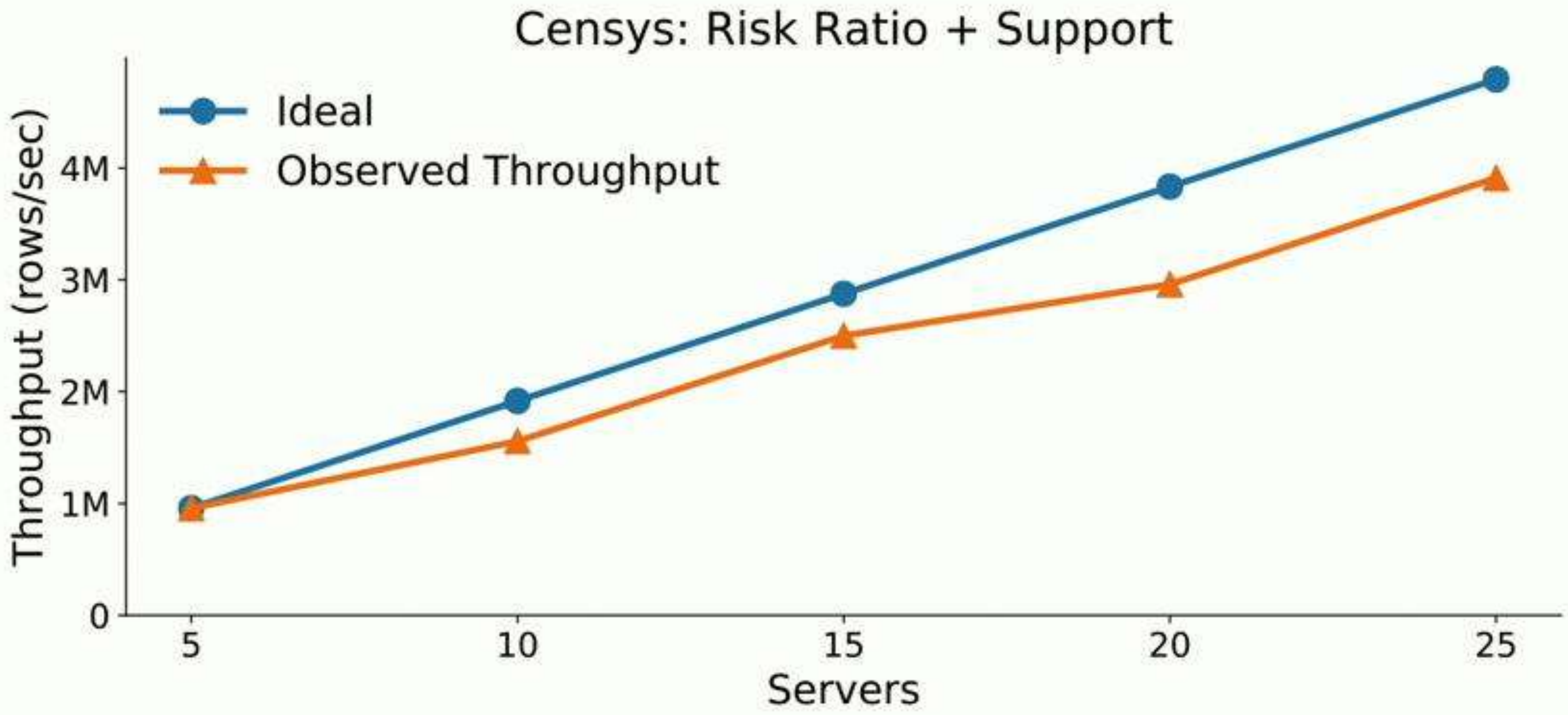
2x runtime improvement when applied to MS-Telemetry B

Evaluation: Spark

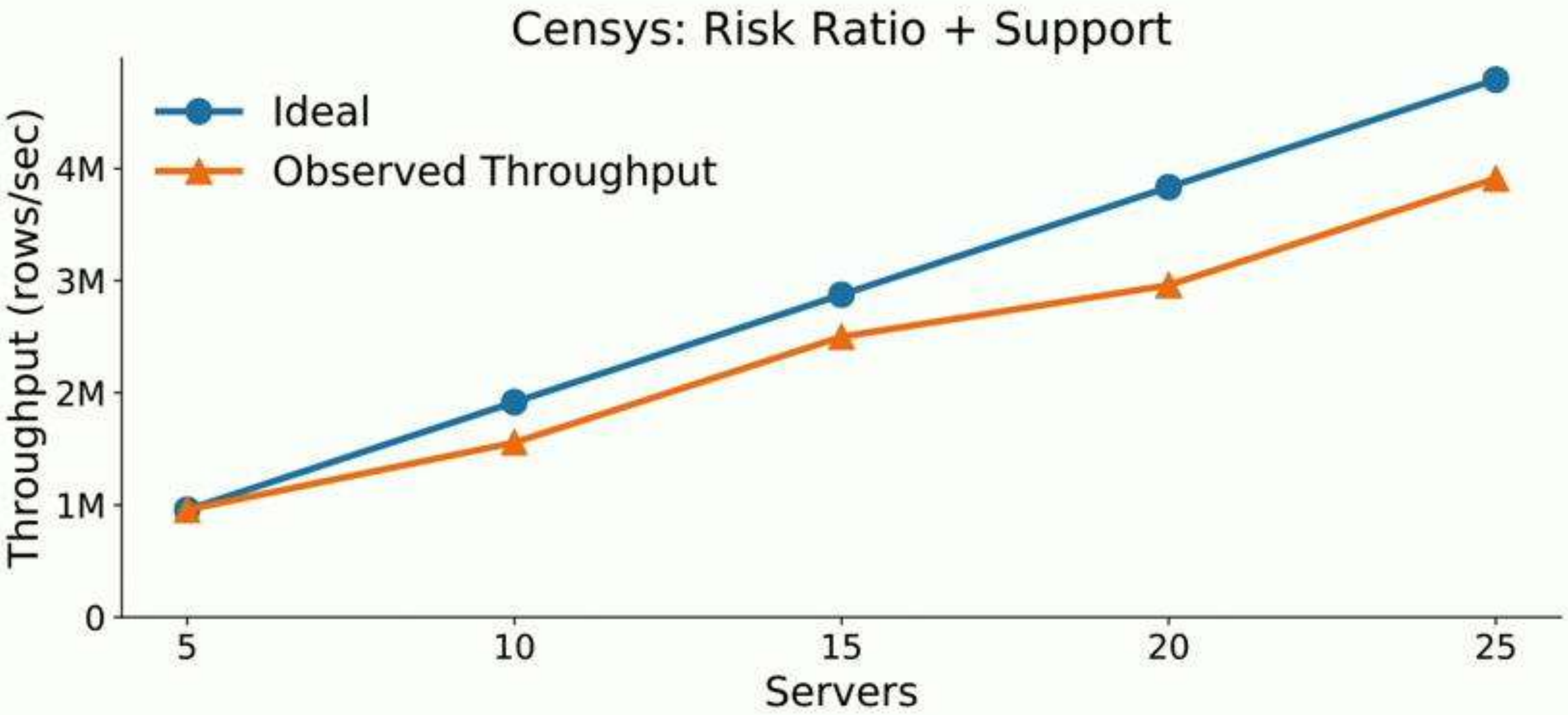
Dataset	File size (CSV)	# rows	# columns	# 3-order combos
Censys	75 GB	400 M	17	38 M
MS-Telemetry A	60 GB	175 M	13	132 M

- Spark v2.2.1
- GCP cluster comprised of n1-highmem-4 instances
- Each worker: 4 vCPUs, 2.2GHz Intel E5 v4 (Broadwell) processor, 26GB of RAM
- Benchmarked MacroBase SQL against:
    - Spark MLlib library (Apriori and FPGrowth implementations)

MB SQL in Spark exhibits near-linear scale up



MB SQL in Spark exhibits near-linear scale up



< 20 minutes on a day's worth of anonymous data used by a production service at Facebook (benchmarked on Facebook's production cluster)

Recap

- The DIFF operator captures the core semantics of several recent explanation engines—a singular relational interface that interoperates with traditional OLAP SQL
- DIFF generalizes to many industrial and academic use cases
- DIFF presents new opportunities for adaptive query optimization; what other SQL operators can we co-design with DIFF?
- We show that DIFF can be implemented in an efficient manner, both for the single-node and distributed cases. What are the limits of this scalability?

Google Microsoft

https://macrobase.stanford.edu