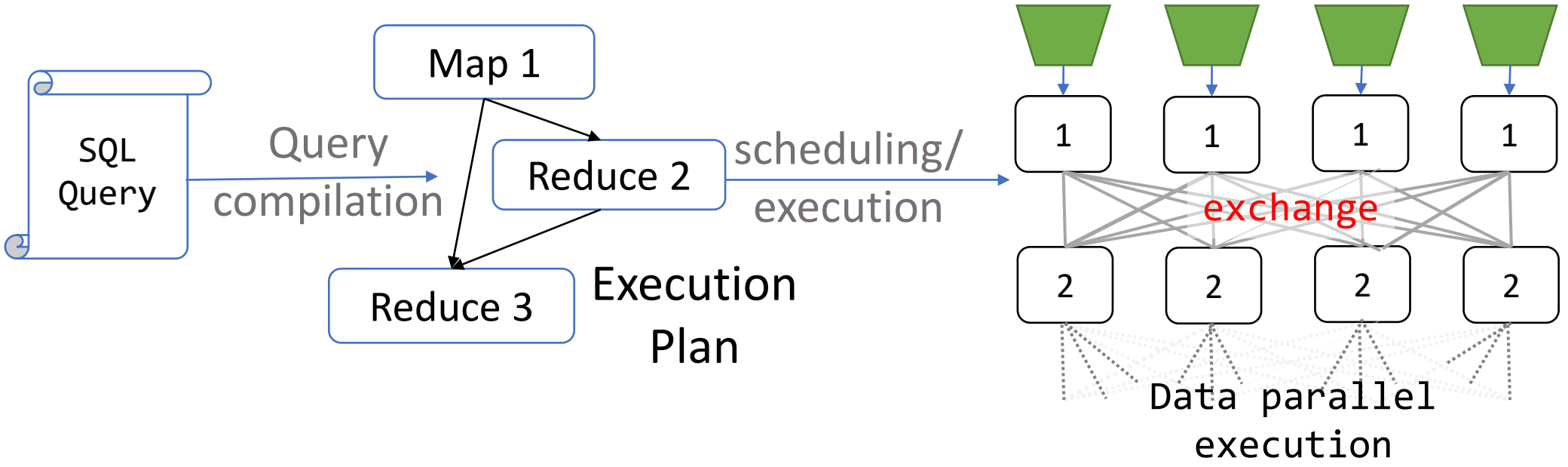
Generalized Sub-Query Fusion for Eliminating Redundant I/O from Big-Data Queries

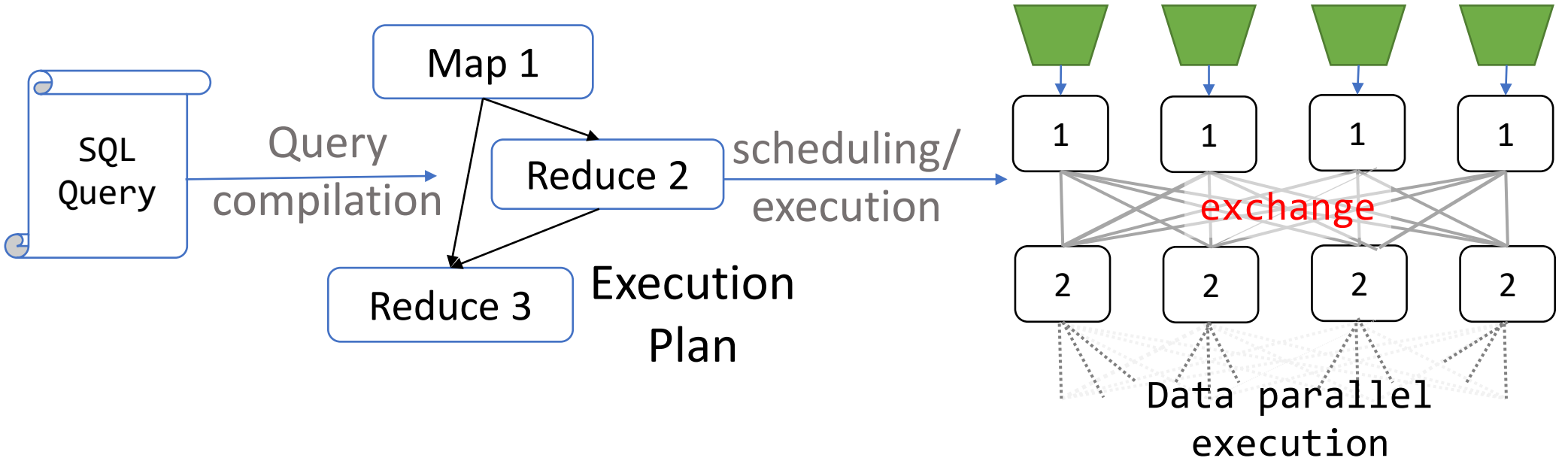
Kaushik Rajan,
Partho Sarthi, Akash Lal (Microsoft Research)
Abhishek Modi, Ashit Gosalia, Prakhar Jain, Mo Liu, (Microsoft)
Saurabh Kalikar (Intel, interned at Microsoft Research)

Big data query compilation



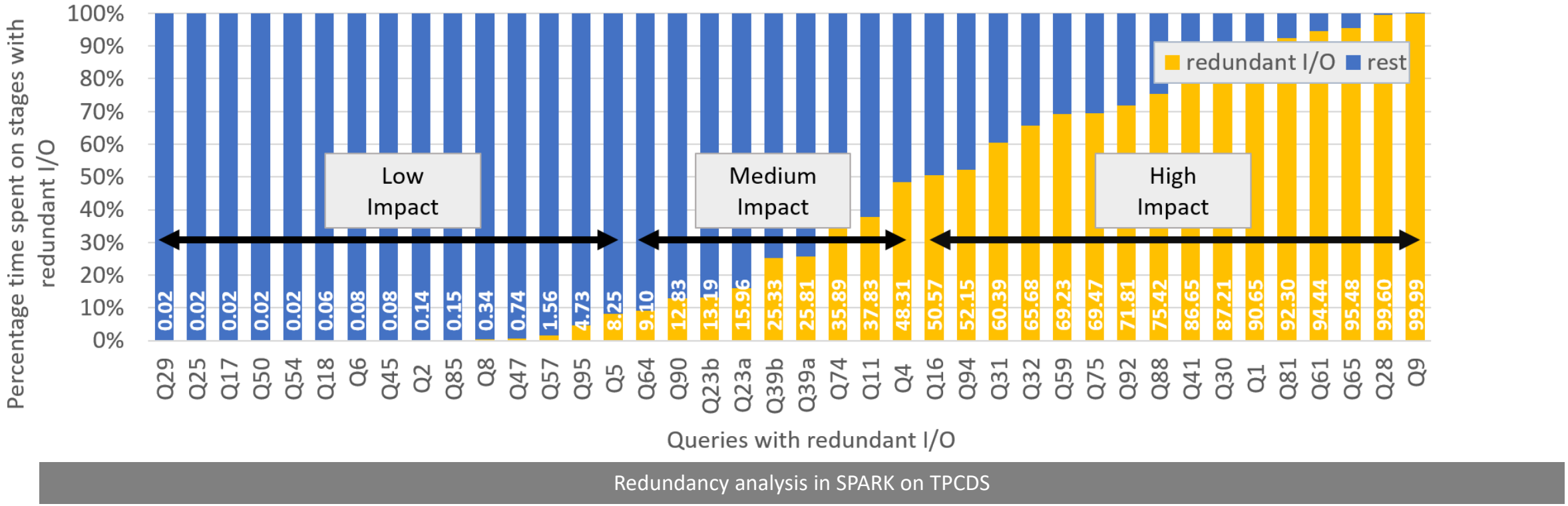
Query compilation and execution in big-data systems (Spark, Hadoop, Snowflake, Amazon Redshift, Google BigQuery, Azure Synapse)

Big data query compilation



Exchanges expensive as they induce disk and network I/O

Plans with fewer stages preferable

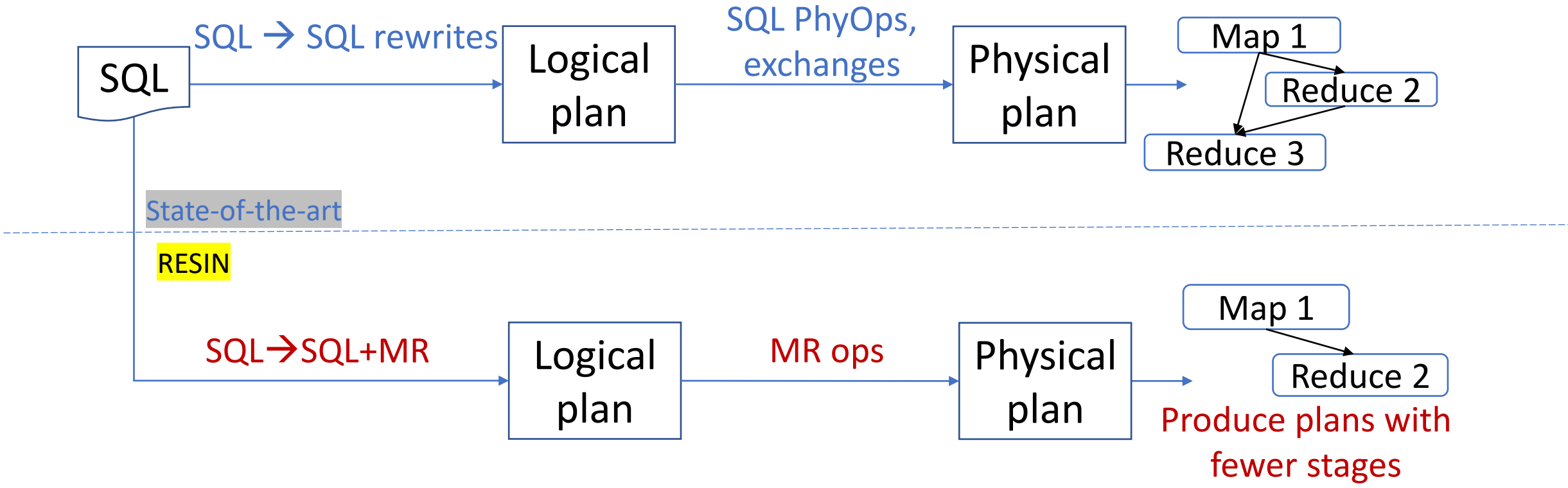


Redundancy analysis in SPARK on TPCDS

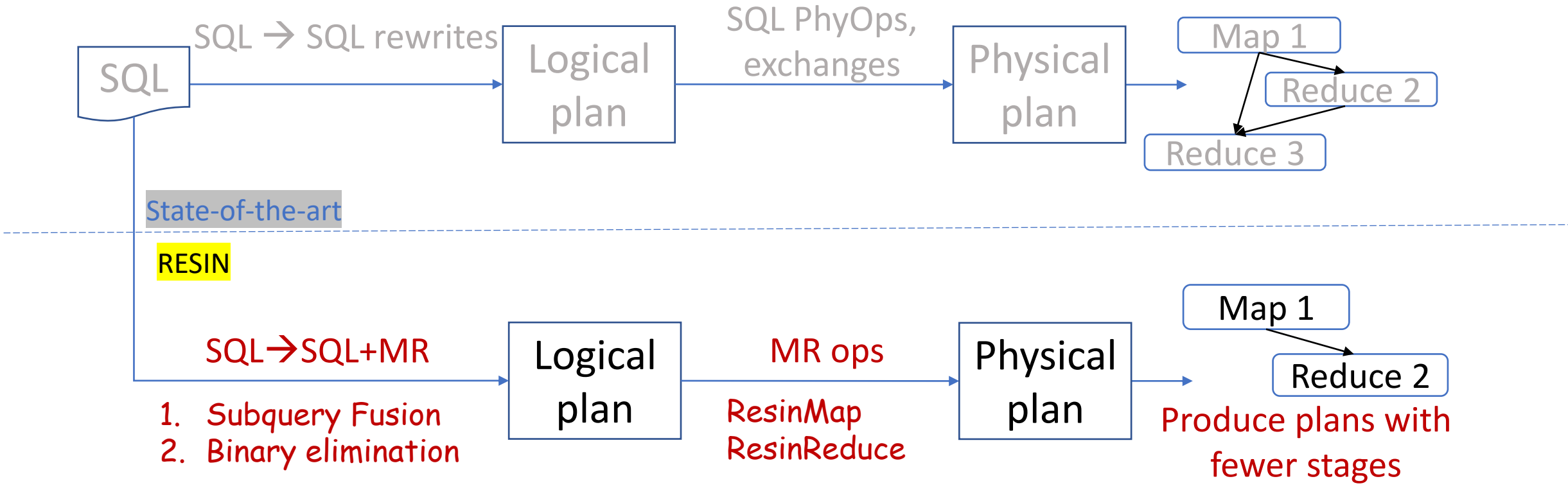
Redundant stages of processing

- TPCDS, 40% of queries have redundant I/O
- 16% of all queries, High-impact spend at least 50% time on stages with redundant I/O
- 9% medium impact, spend 10-50% time on stages with redundant I/O

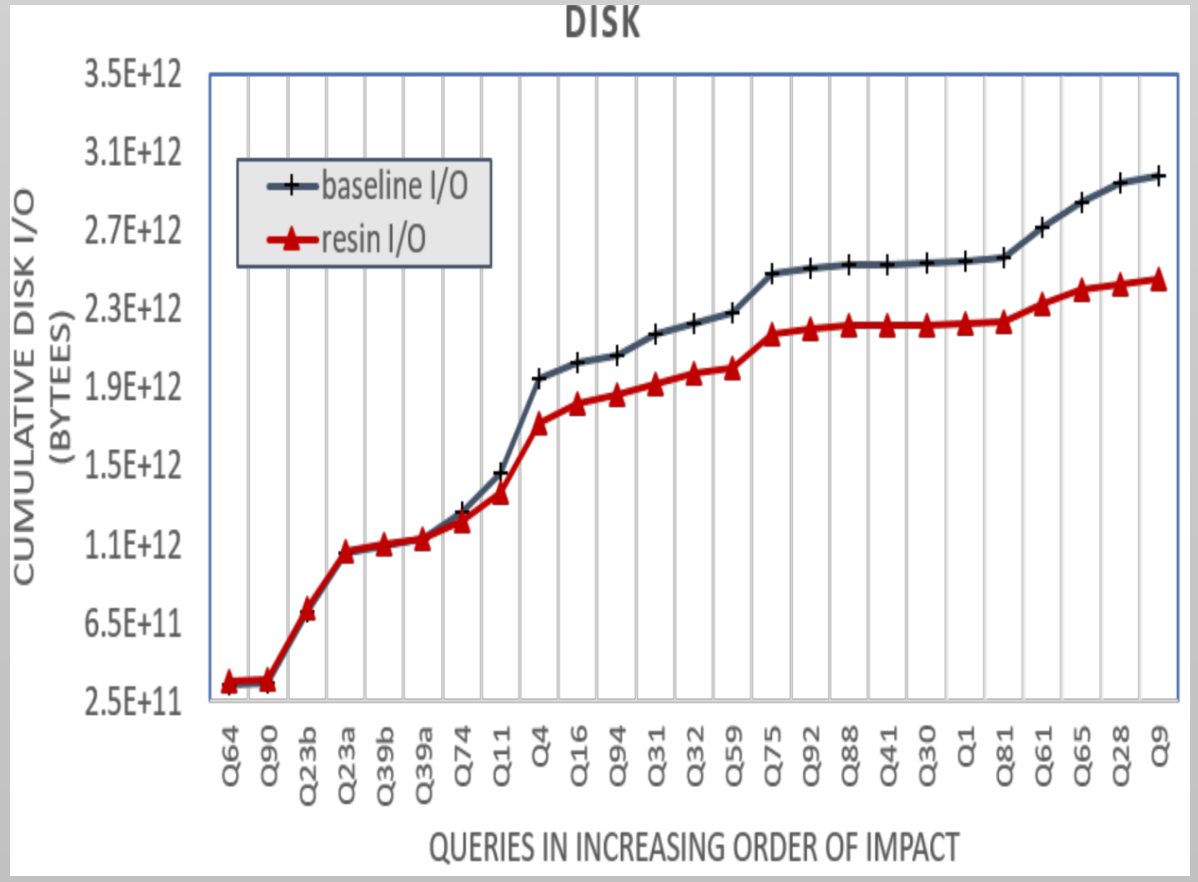
RESIN: MapReduce reasoning during optimization

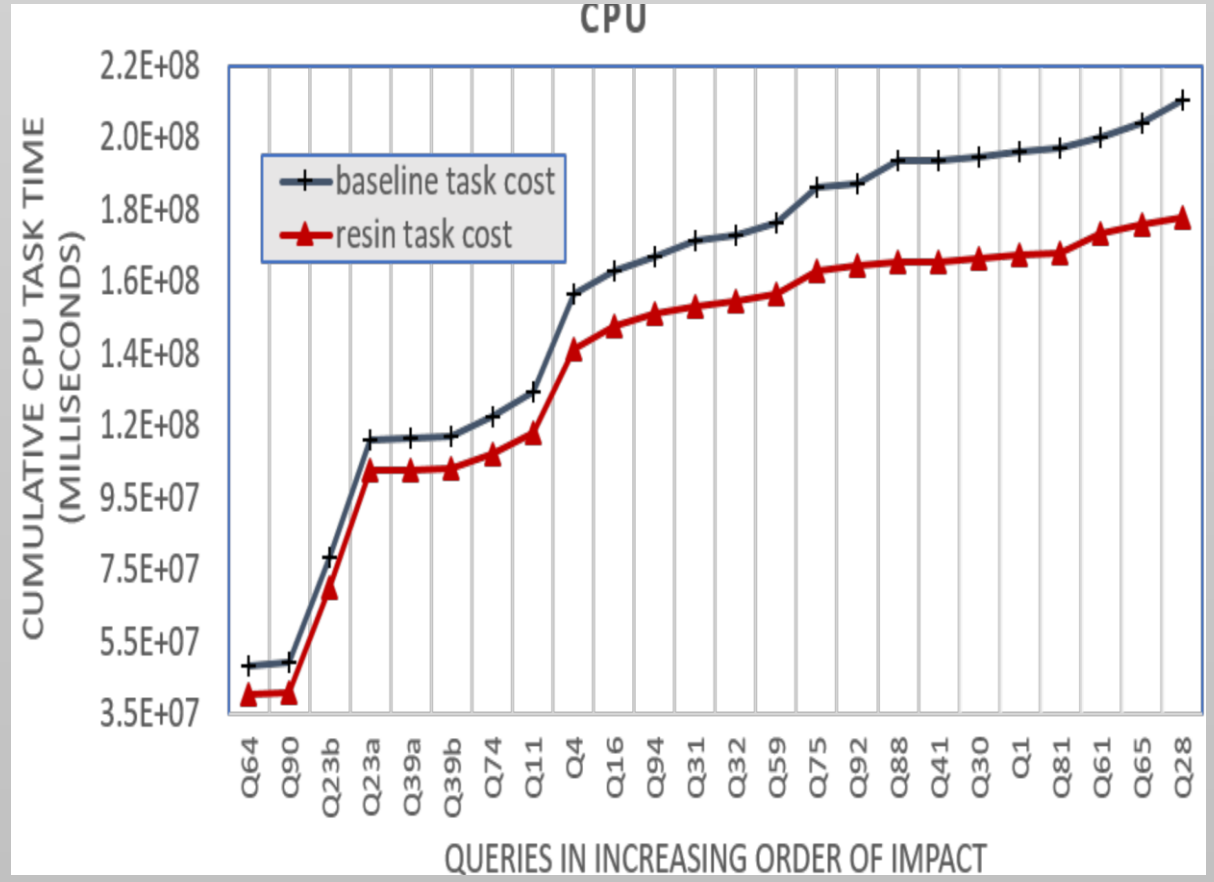


RESIN: MapReduce reasoning during optimization



Impact of RESIN on I/O and Memory





Rest of the talk

1. ResinMap and ResinReduce
2. Generalized sub-query fusion
3. Implementation on Spark
4. Experimental evaluation

ResinMap

Microsoft Research

A row-wise operator, can produce multiple output rows per input row

(a) SQL Query

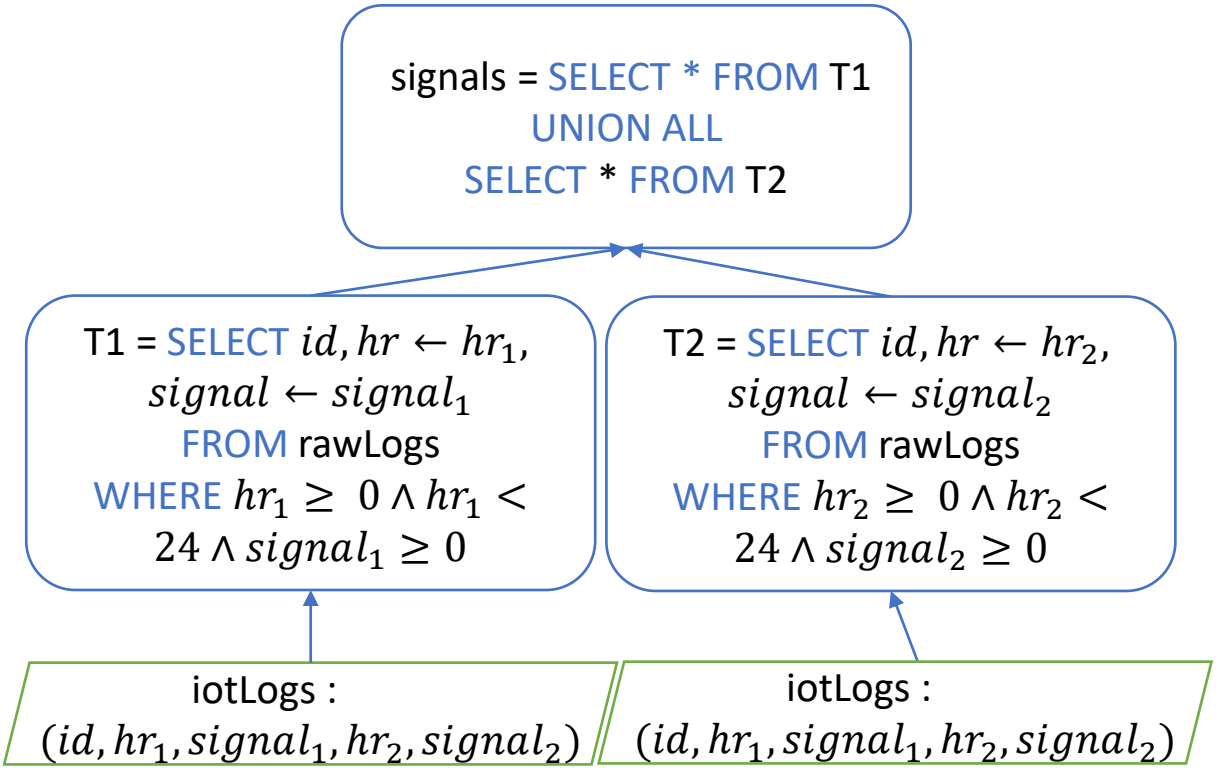
(b) Standard execution plan

(c) Optimized execution plan

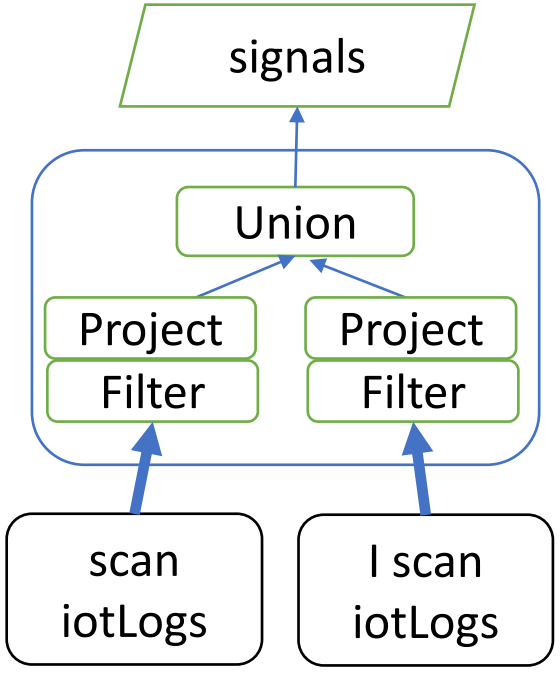
ResinMap

Microsoft Research

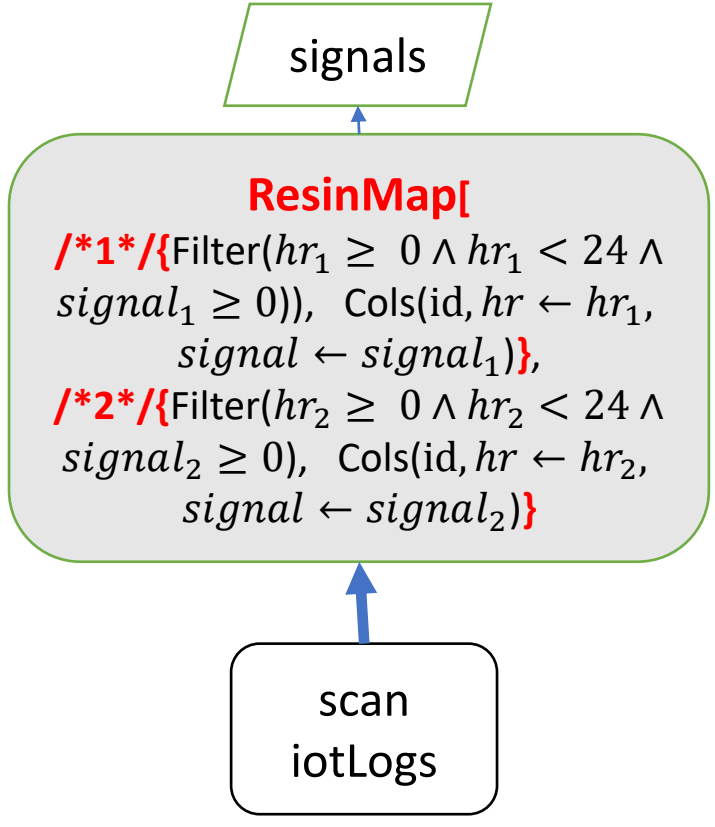
A row-wise operator, can produce multiple output rows per input row



(a) SQL Query



(b) Standard execution plan

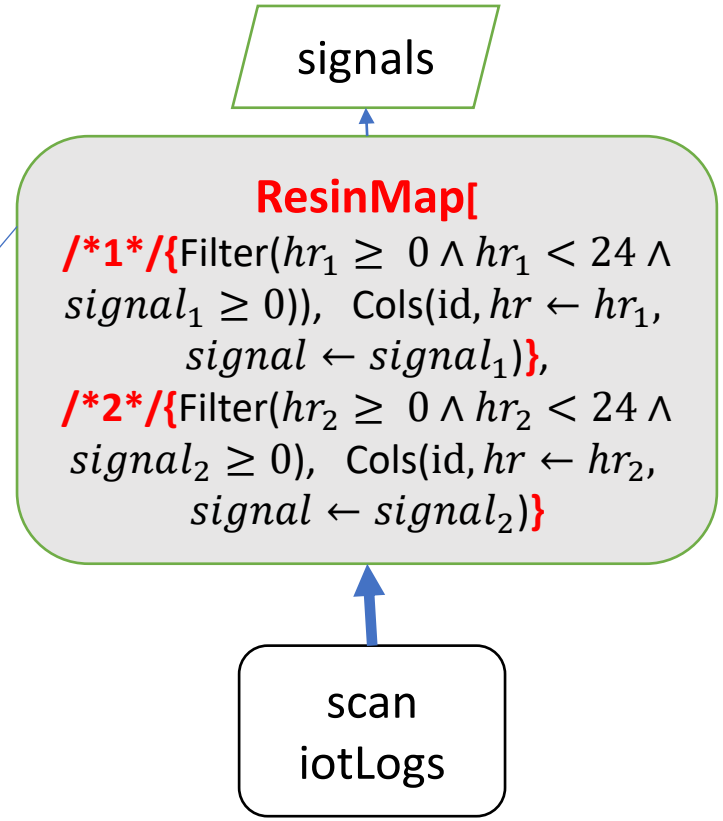


(c) Optimized execution plan

ResinMap

A row-wise operator, can produce multiple output rows per input row

```
//Each mapper m processes a partition rawlogs[m]
Method ResinMap(m) {
    foreach⟨id, hr_1, signal_1, hr_2, signal_2⟩ ∈ iotLogs[m] {
        if(hr_1 ≥ 0 ∧ hr_1 < 24 ∧ signal_1 ≥ 0) {
            hr = hr_1; signal = signal_1; output(id, hr, signal)
        }
        if(hr_2 ≥ 0 ∧ hr_2 < 24 ∧ signal_2 ≥ 0) {
            hr = hr_2; signal = signal_2; output(id, hr, signal)
        }
}
```

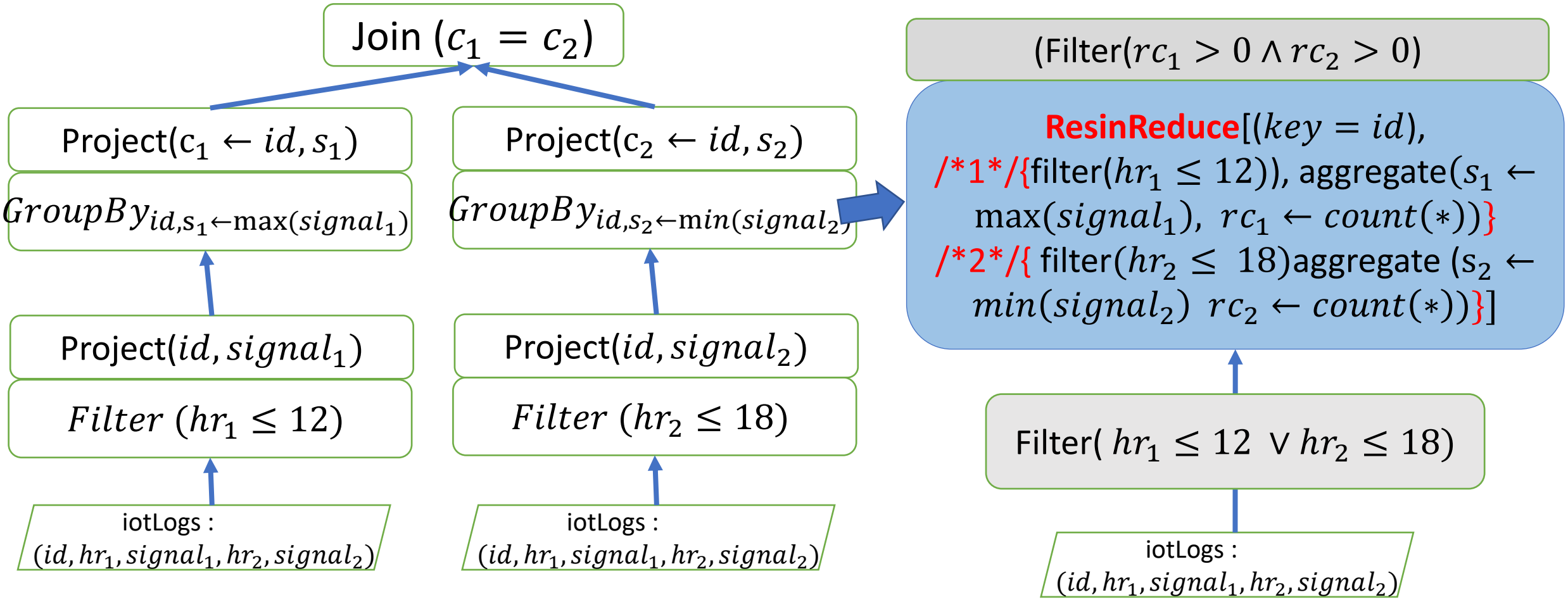


Single table *select*, *project*, *union* queries in one stage

ResinReduce

Microsoft Research

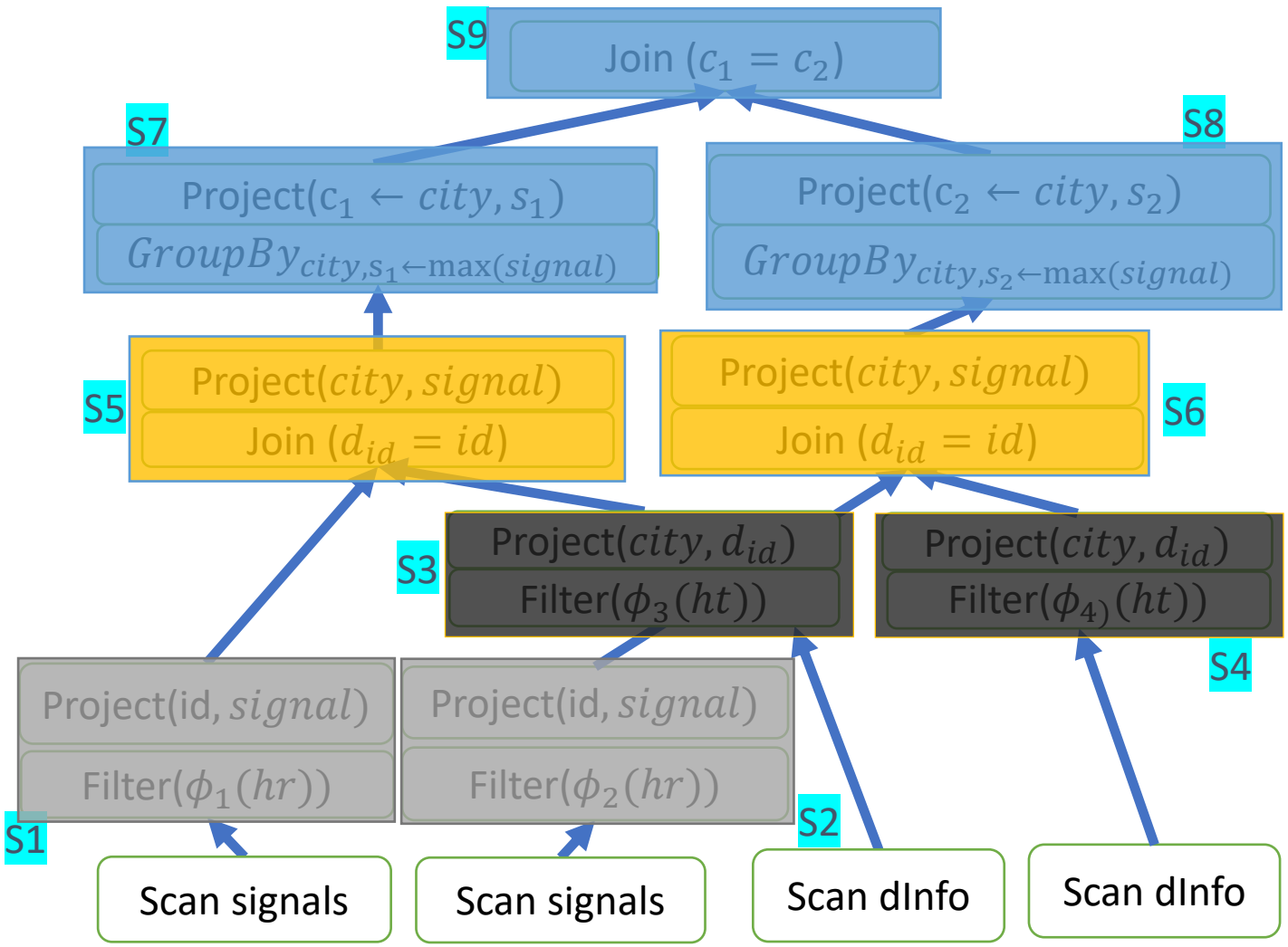
Key based operator, process rows sharing key, produce one row

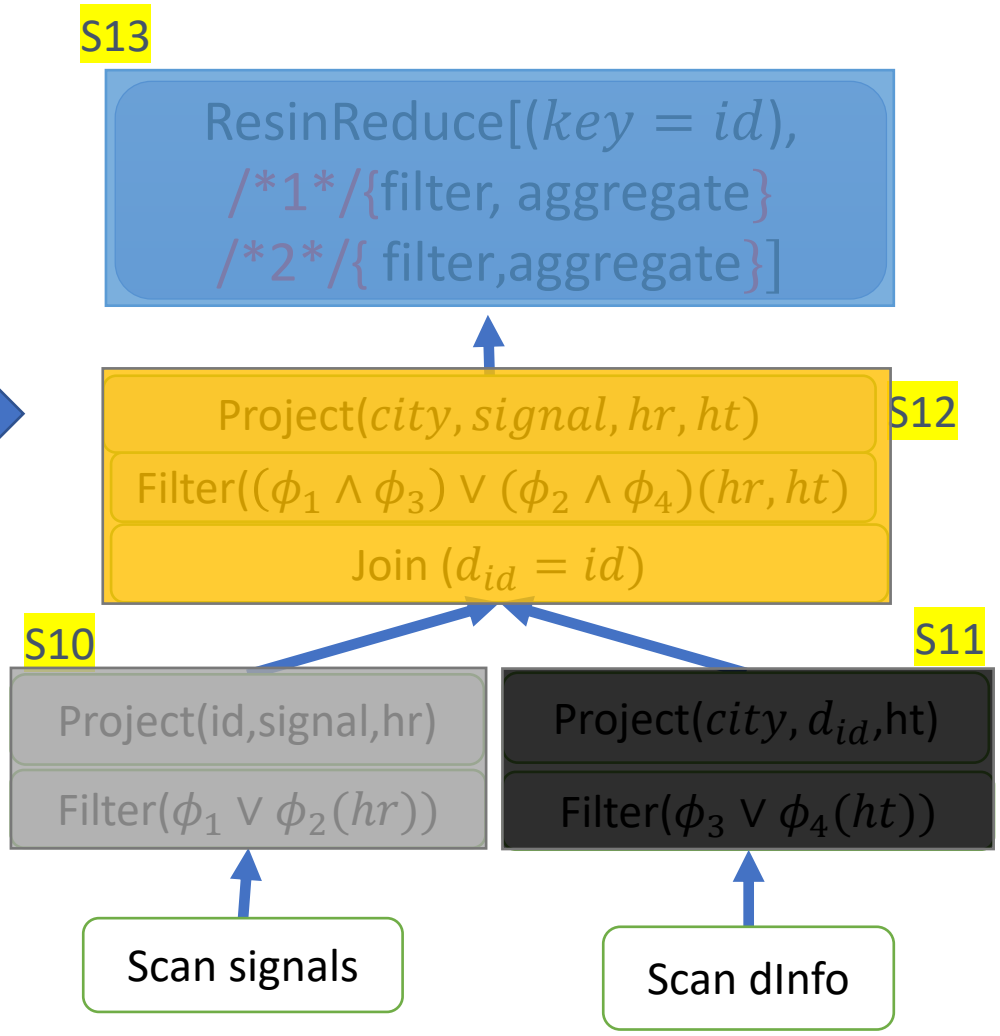


Eliminate multiple shuffles from single table join queries

Sub-query fusion

Microsoft Research

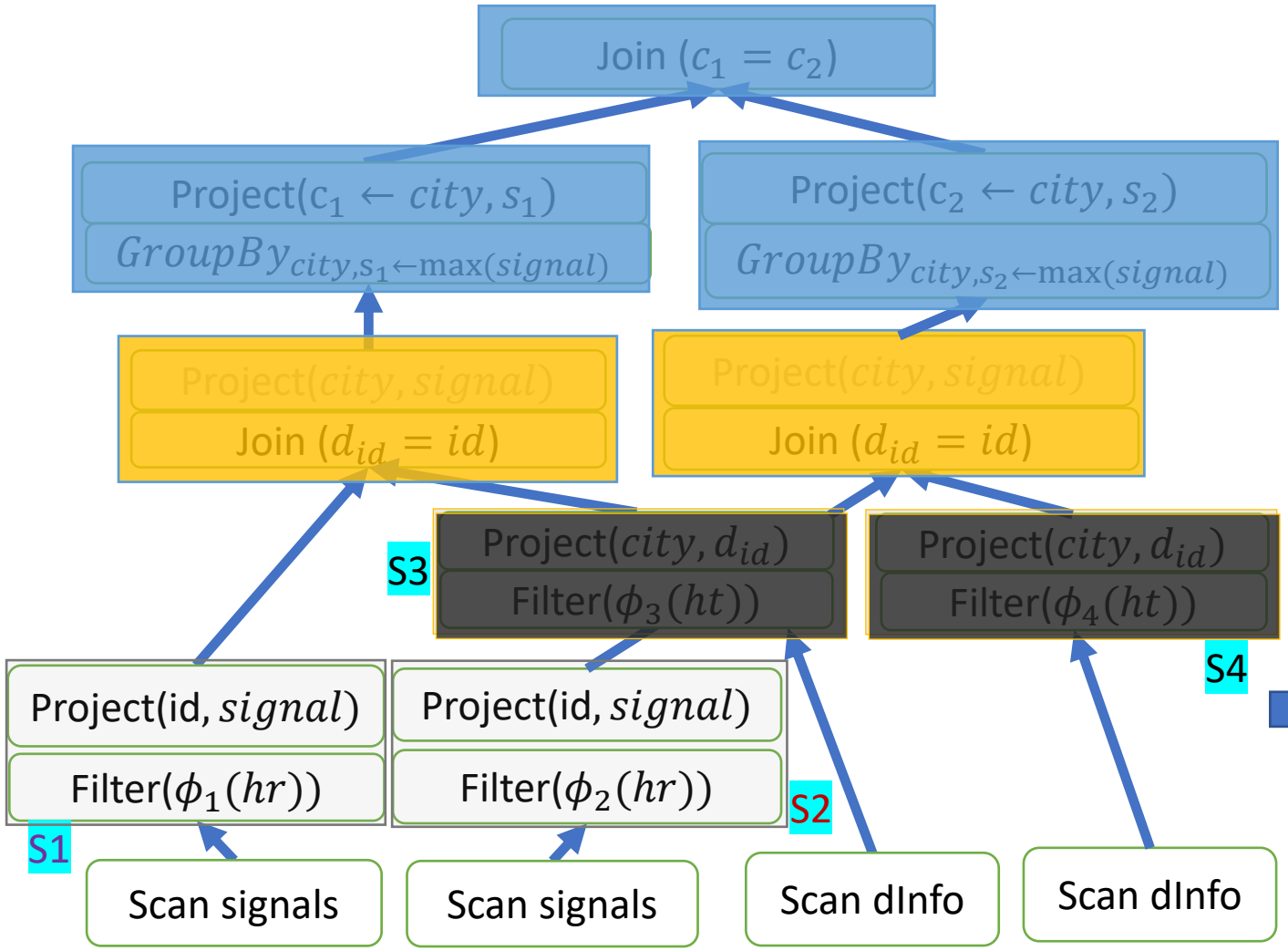


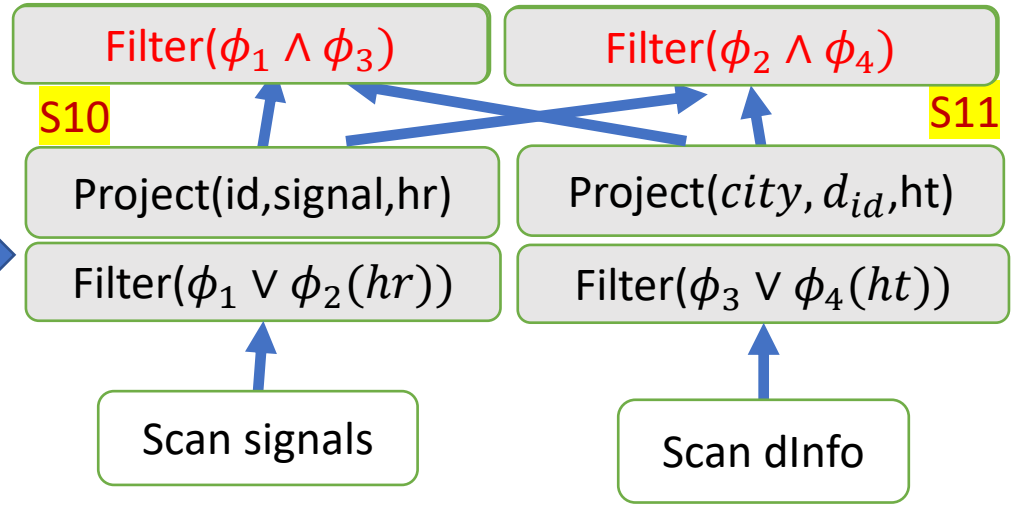


Eliminate scans/shuffles from multi-table queries

Sub-query fusion

Microsoft Research

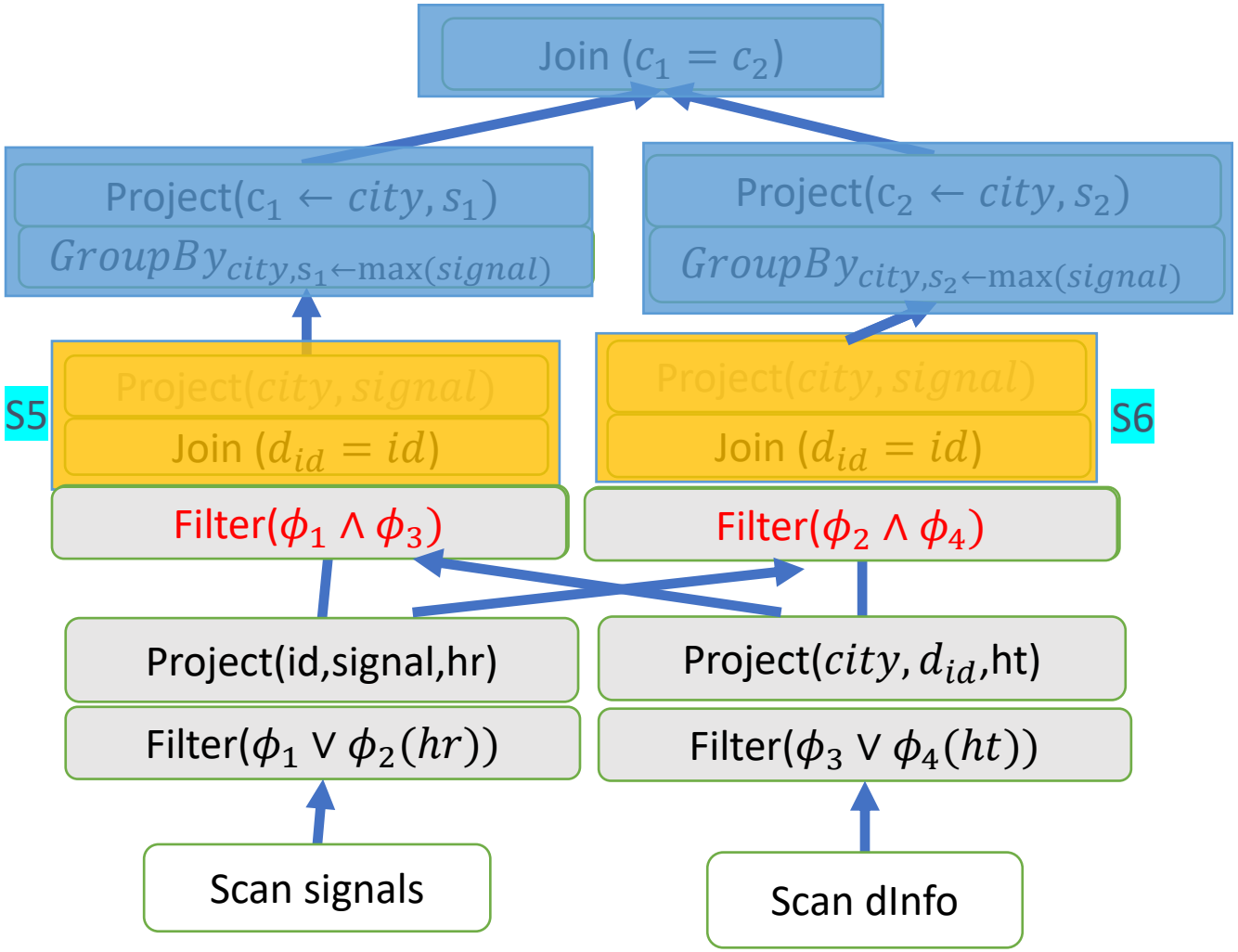


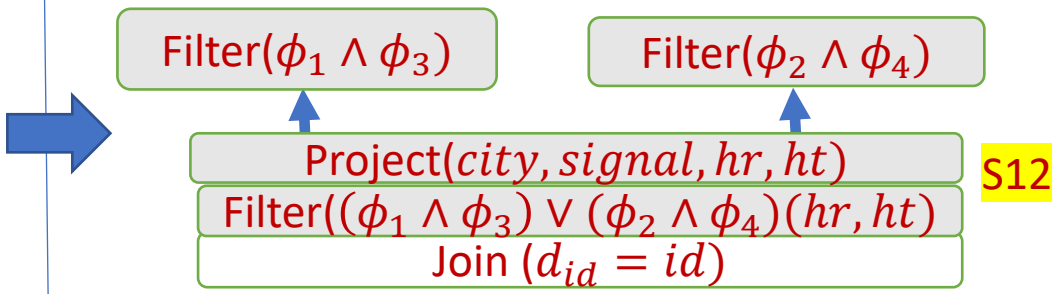


Eliminate scans/shuffles from multi-stage queries

Sub-query fusion

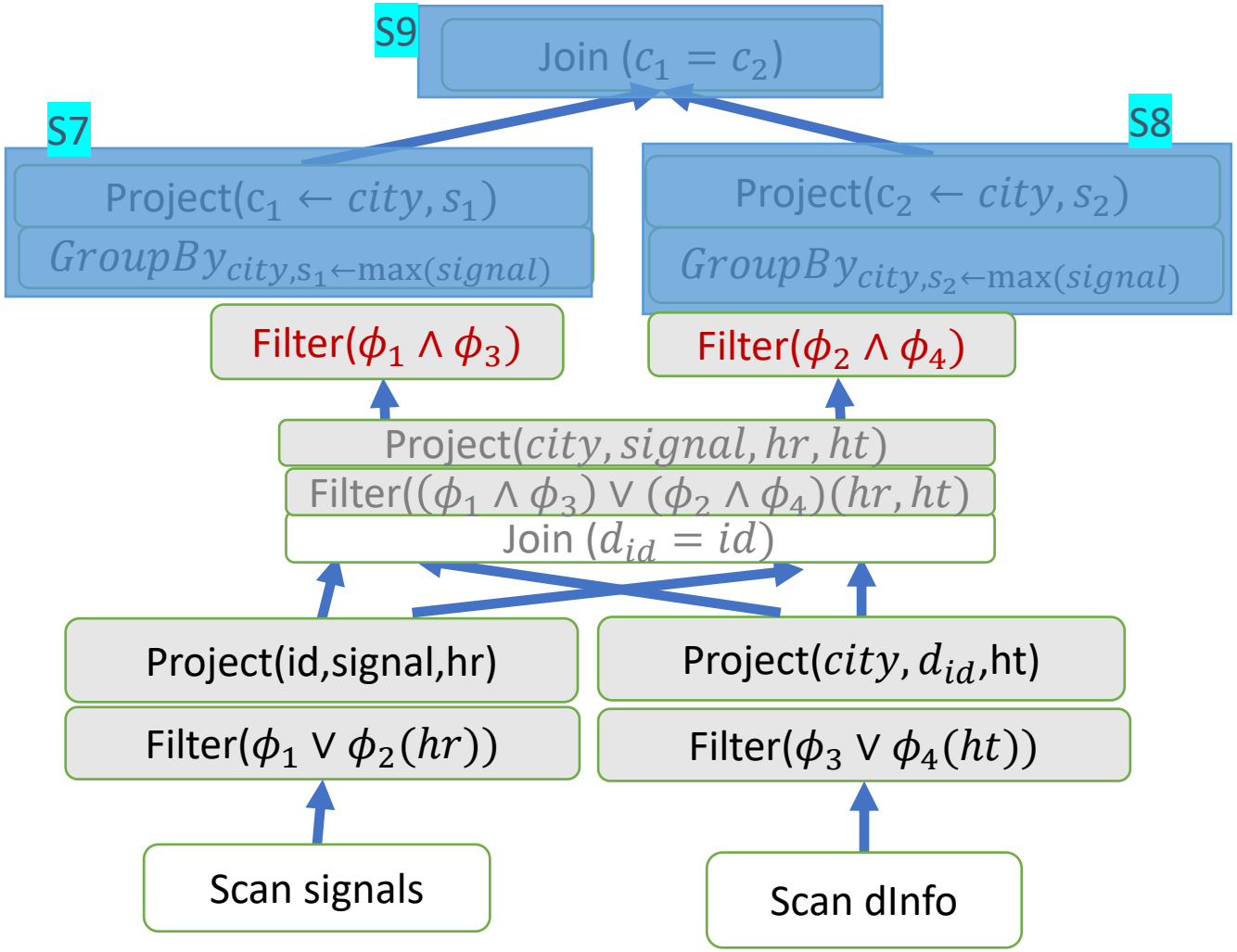
Microsoft Research

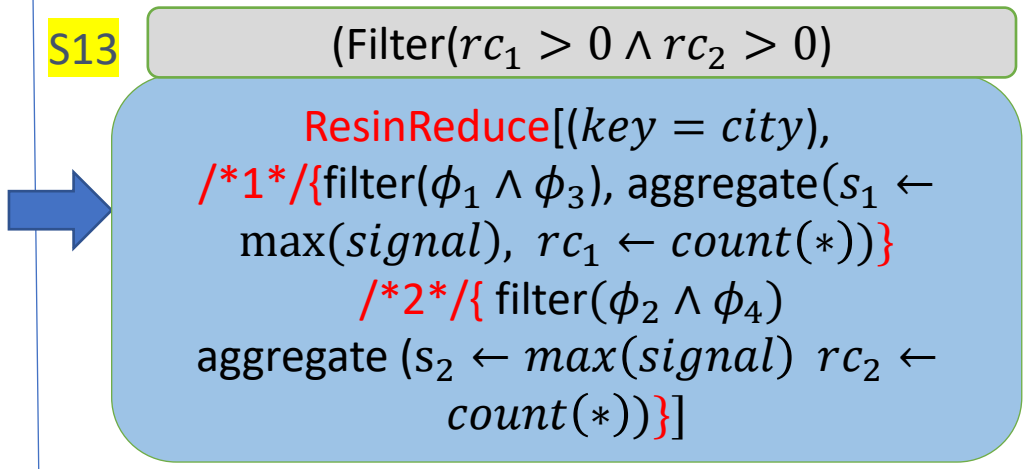




Eliminate scans/shuffles from multi-stage queries

Sub-query fusion

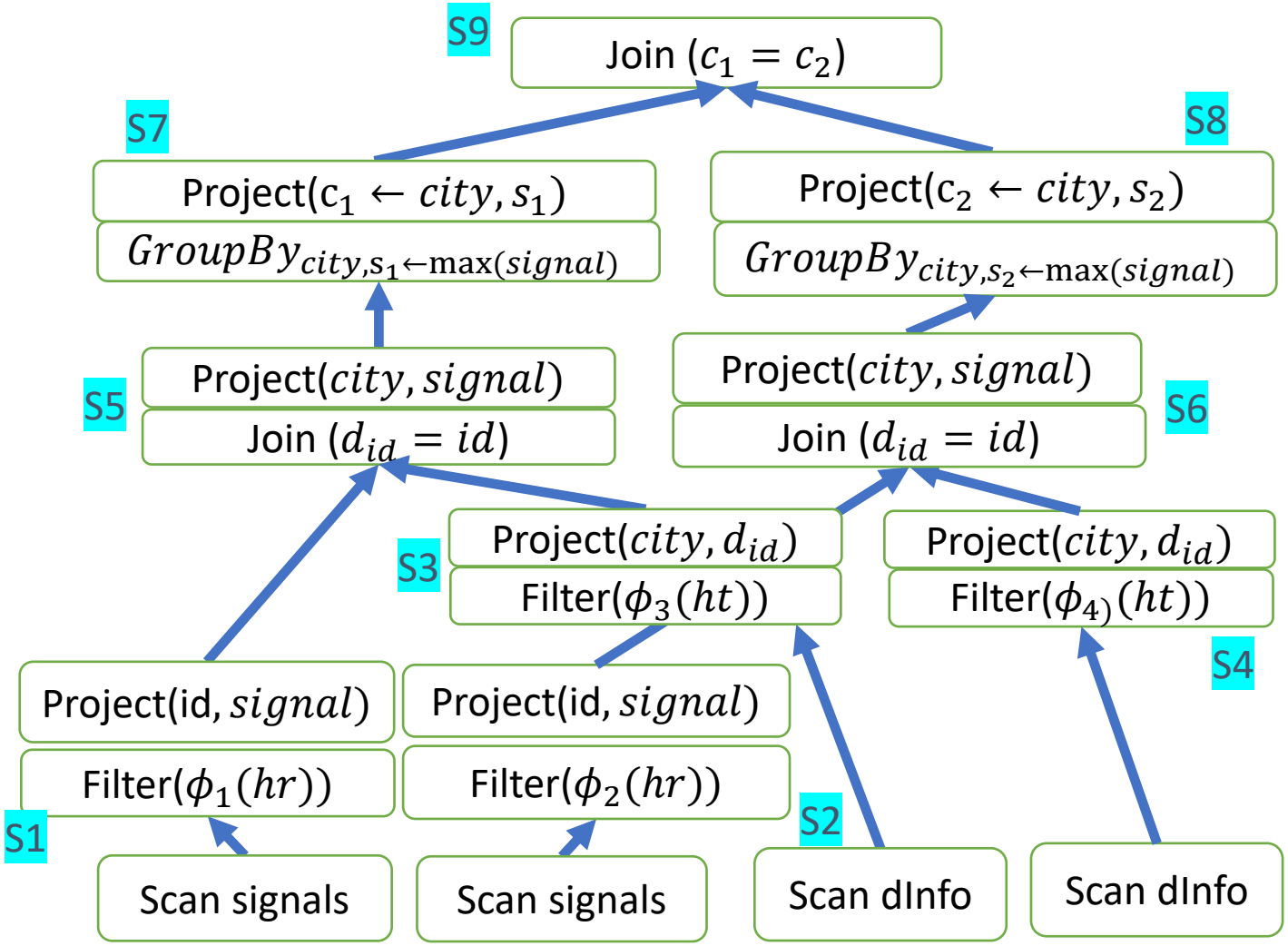


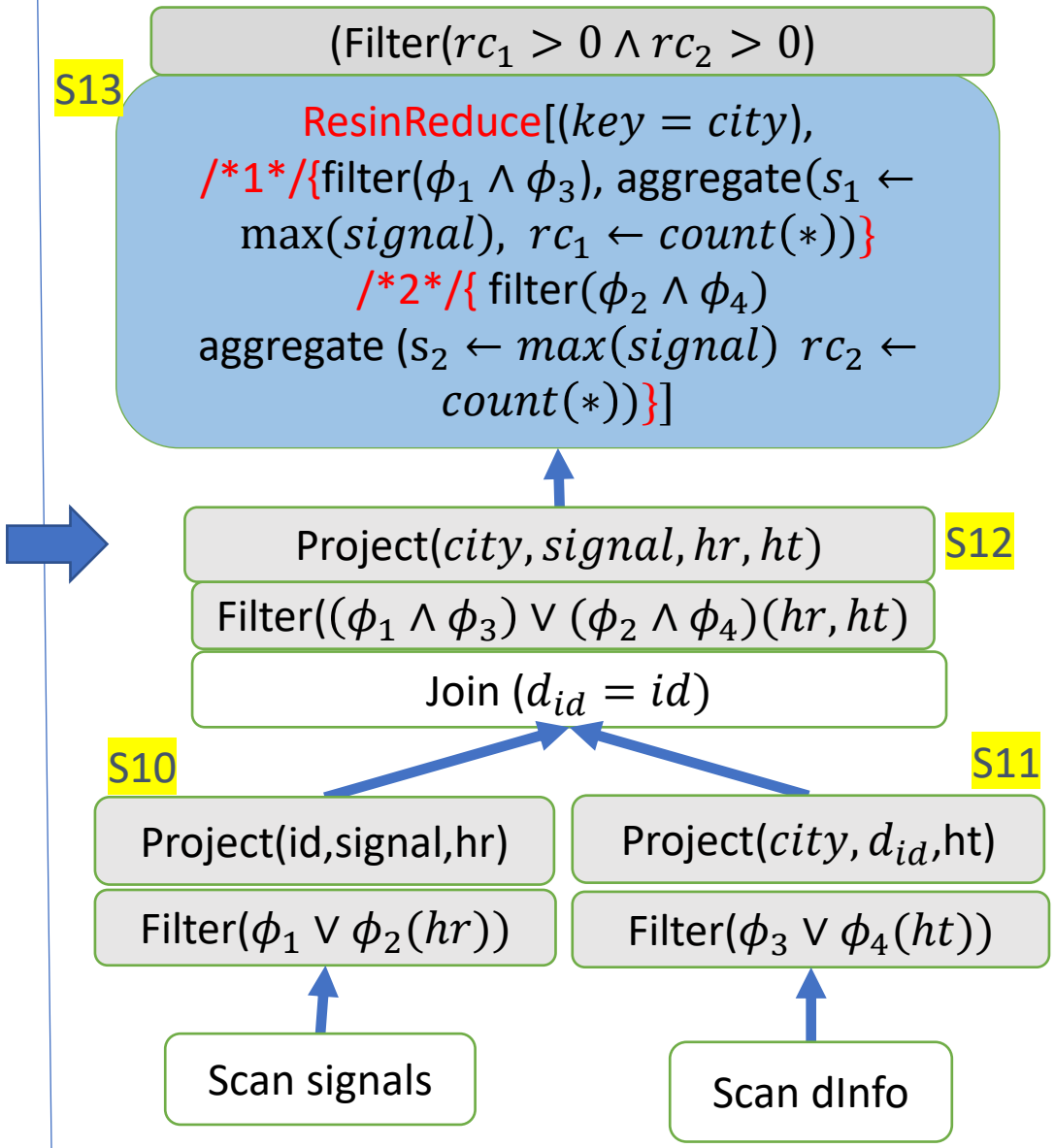


Eliminate scans/shuffles from multi-stage queries

Sub-query fusion

Microsoft Research





Eliminate scans/shuffles from multi-table queries

In the paper

- Parameters for ResinMap and ResinReduce operators, semantics and implementation
- Fusing of operators without increasing the number of rows shuffled
- Fusion rules for all sparkSQL operators, conditions under which fusion is possible

Rest of the talk

1. ResinMap and ResinReduce
2. Generalized sub-query fusion
3. Implementation on Spark
4. Experimental evaluation

Implementation

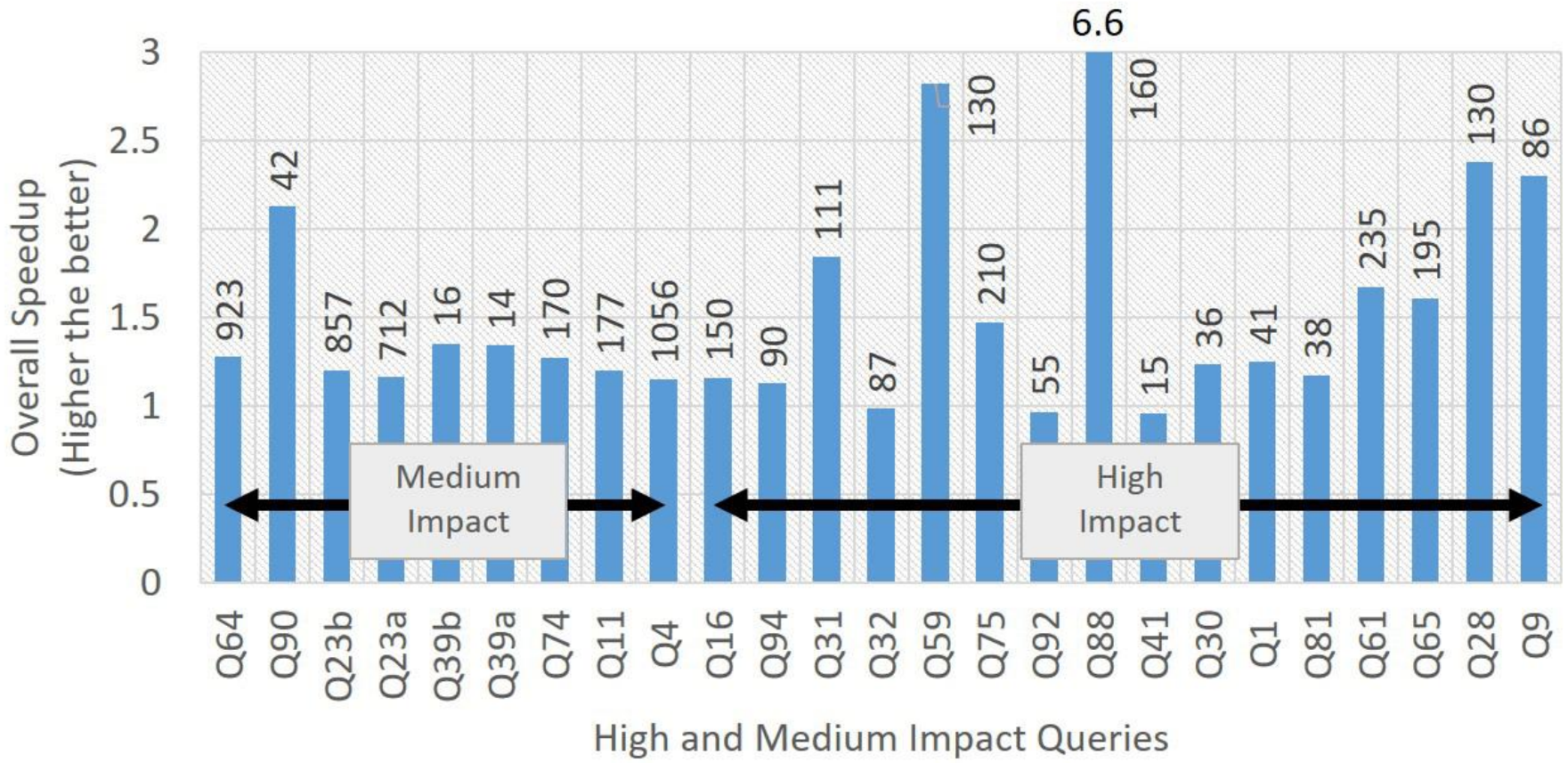
Implemented RESIN on catalyst optimizer in SPARK 2.4

1. Added logical and physical operators for **ResinMap** and **ResinReduce**
2. Added a new batch of optimization rules
   - Perform fusion in a single traversal of the tree
   - Perform Union and Join elimination by checking fused parent
   - Introduce exchanges if parent after fusion cannot be eliminated
3. Added implementations for our operators with codegen support

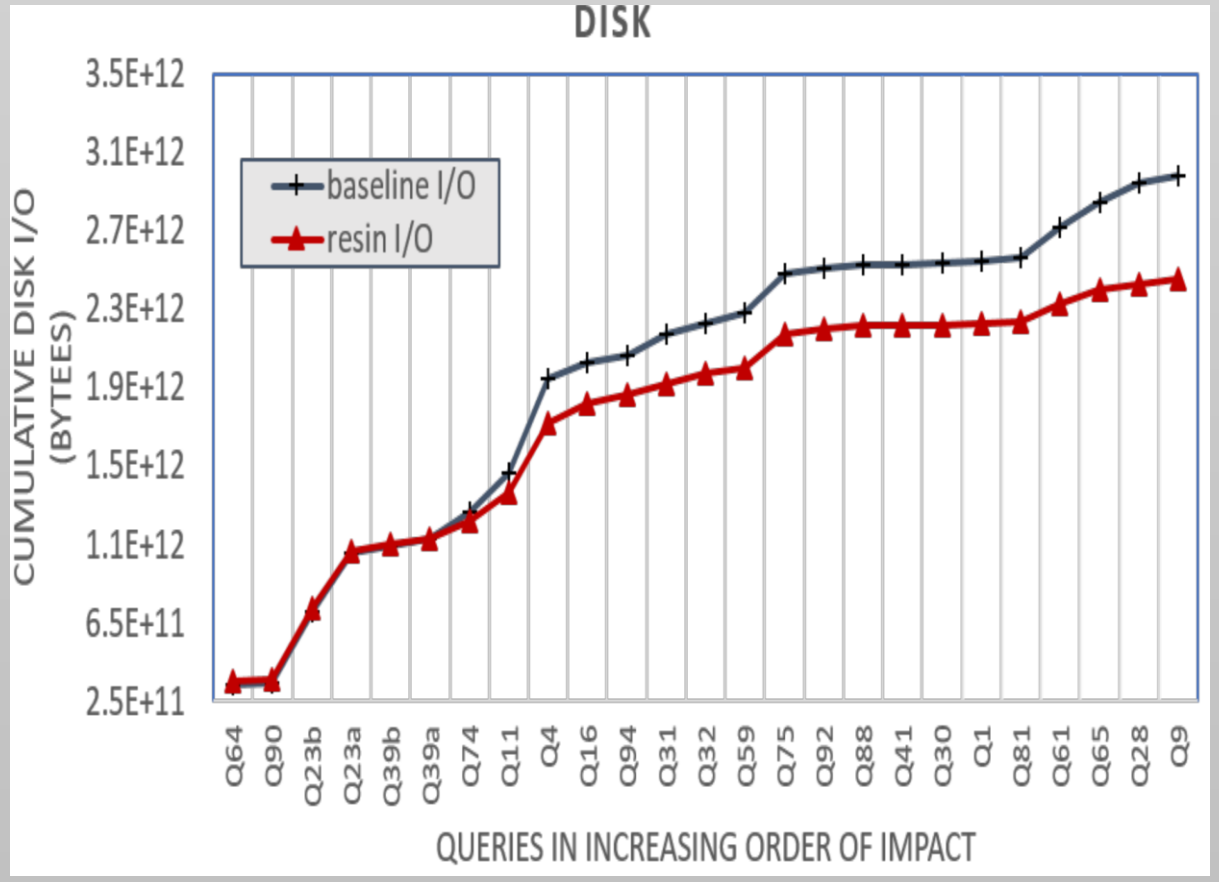
Evaluation

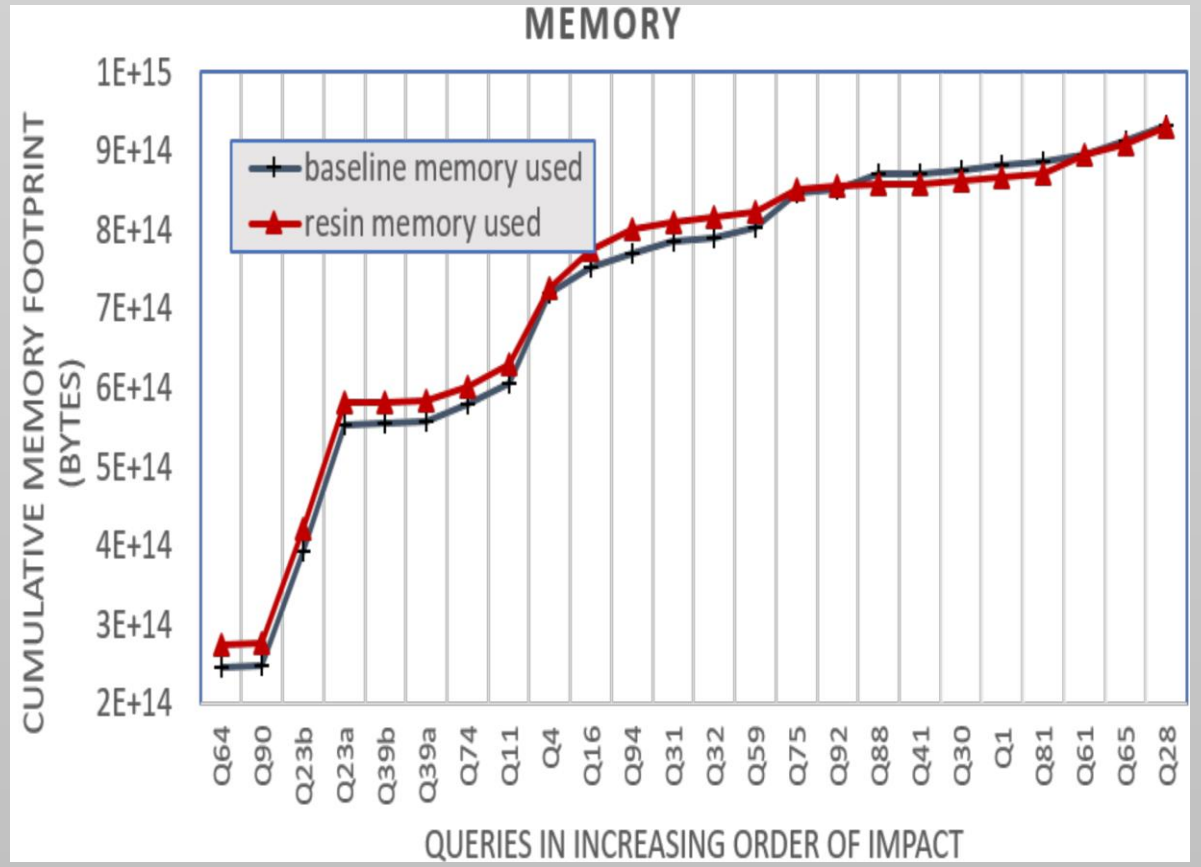
- Evaluated with TPCDS at 1TB and 10TB scale, data stored in *parquet*
- Two different clusters <120 cores, 480 GB memory> and <480 cores, 1.6TB memory>
- Detailed evaluation of 40 (out of 104) queries with redundant I/O
- *Note: baseline that already has basic I/O optimizations (predicate and project pushdown to store, exchange reuse)*

Speedup at 1TB



Impact of RESIN on I/O and Memory





Conclusions

- Big-data optimizers produce plans with redundant I/O and compute
- Proposed optimizer extensions to perform first class map-reduce reasoning
- Added generic map and reduce operators, rewrites that fuse stages and eliminate redundant I/O
- Demonstrated savings in terms of latency, disk and network I/O

Thank You

Email krajan@microsoft.com or any of the other authors to contact us