40th IEEE International Conference on
Data Engineering
Utrecht, Netherlands | 13th - 17th May
2024

Dynamic Data Layout Optimization with Worst-case Guarantees

Kexin Rong[1,3], Paul Liu[2], Sarah Ashok Sonje[1], Moses Charikar[2]

Georgia Tech[1], Stanford University[2], VMware Research[3]

Data partition as a basic unit for storage

host	metric	time
s1	3	2015
s2	5	2015
…		
s1	500	2016
s3	100	2017
…		
s1	50	2019
…		

Partition 1

host	metric	time
s1	3	2015
…		

Parquet

Partition 2

host	metric	time
s1	500	2016
…		

Parquet

⋮

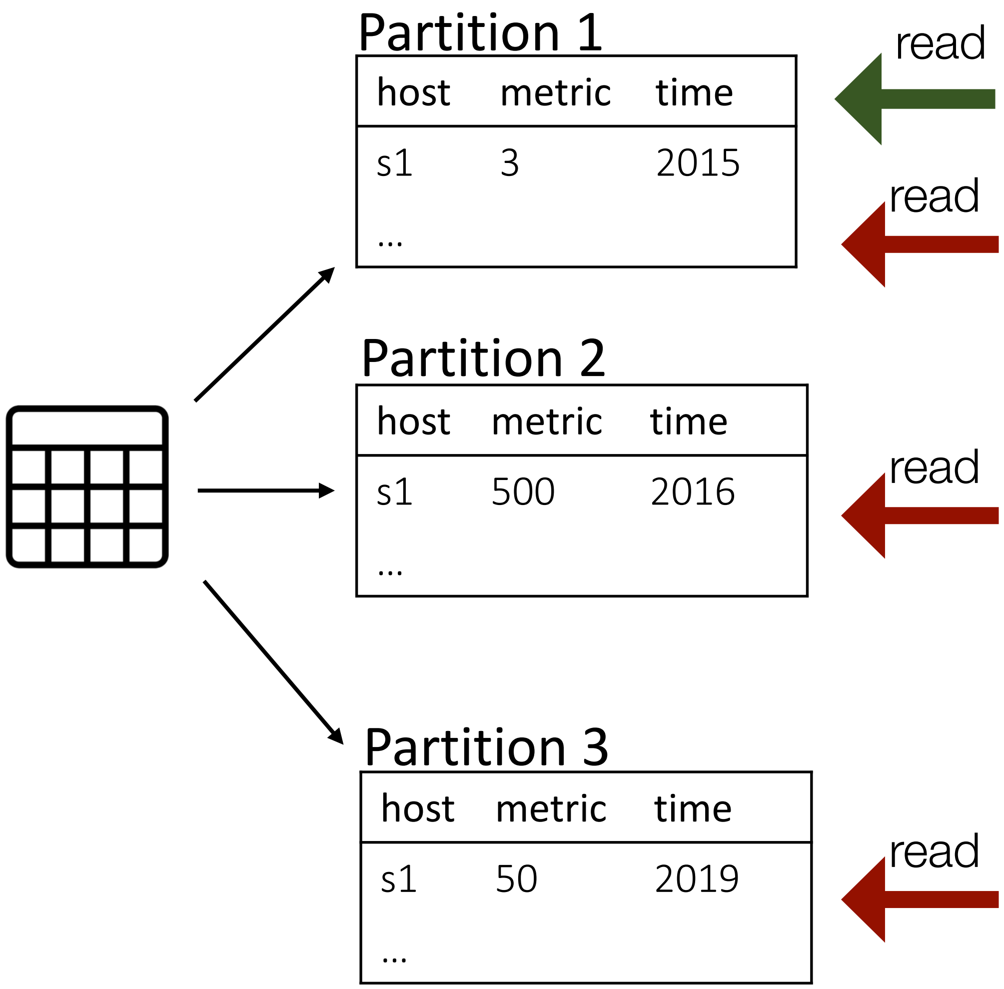
Partition N

host	metric	time
s1	50	2019
…		

Parquet

Data layout: $f(row\_id) \longrightarrow partition\_id$

Data layouts affect query performance



Partition-level metadata

Part	min(time)	max(time)	min(host)	max(host)
1	2015	2015	server1	server5
2	2016	2019	server1	server5
3	2019	2020	server1	server5

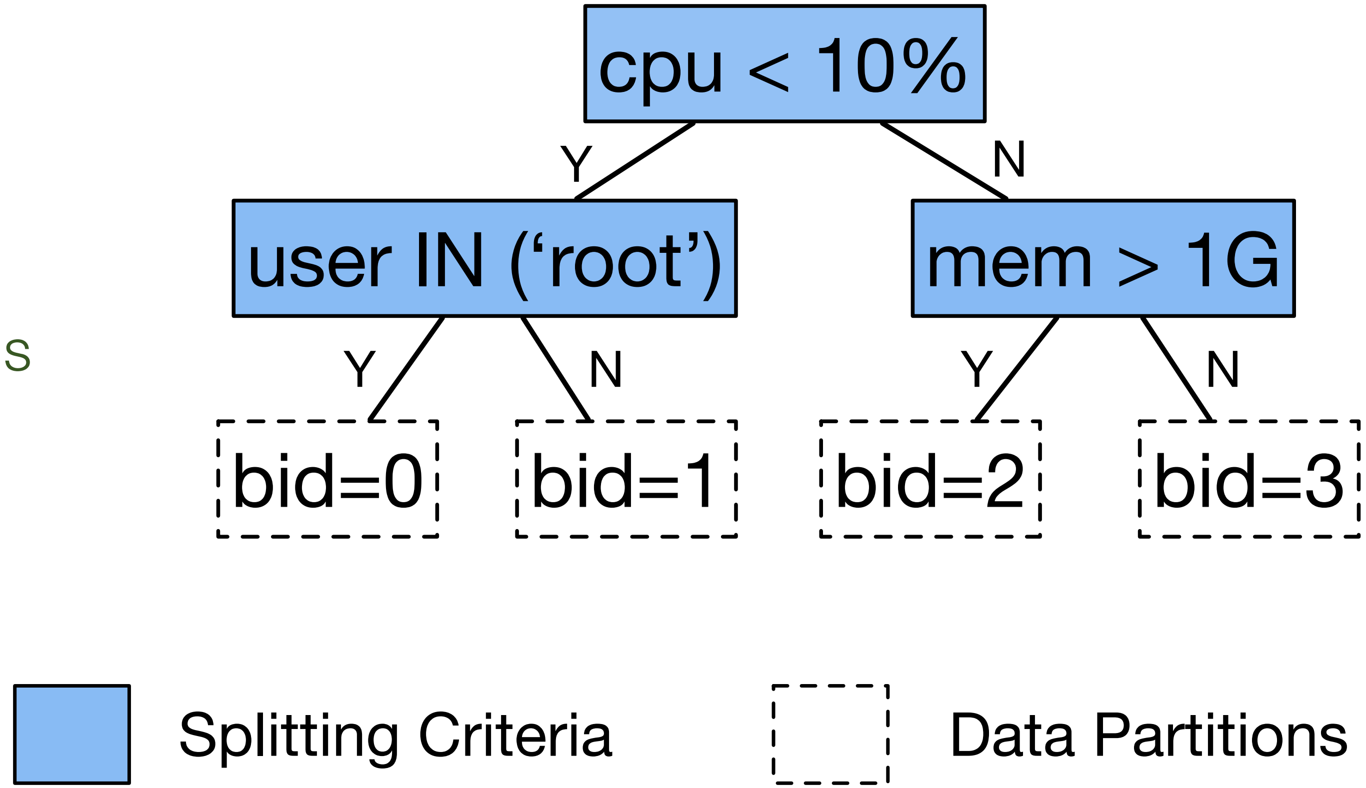
```
SELECT * FROM tbl
WHERE time = 2015
```

```
SELECT * FROM tbl
WHERE host = server2
```

Workload-aware layouts maximize data skipping

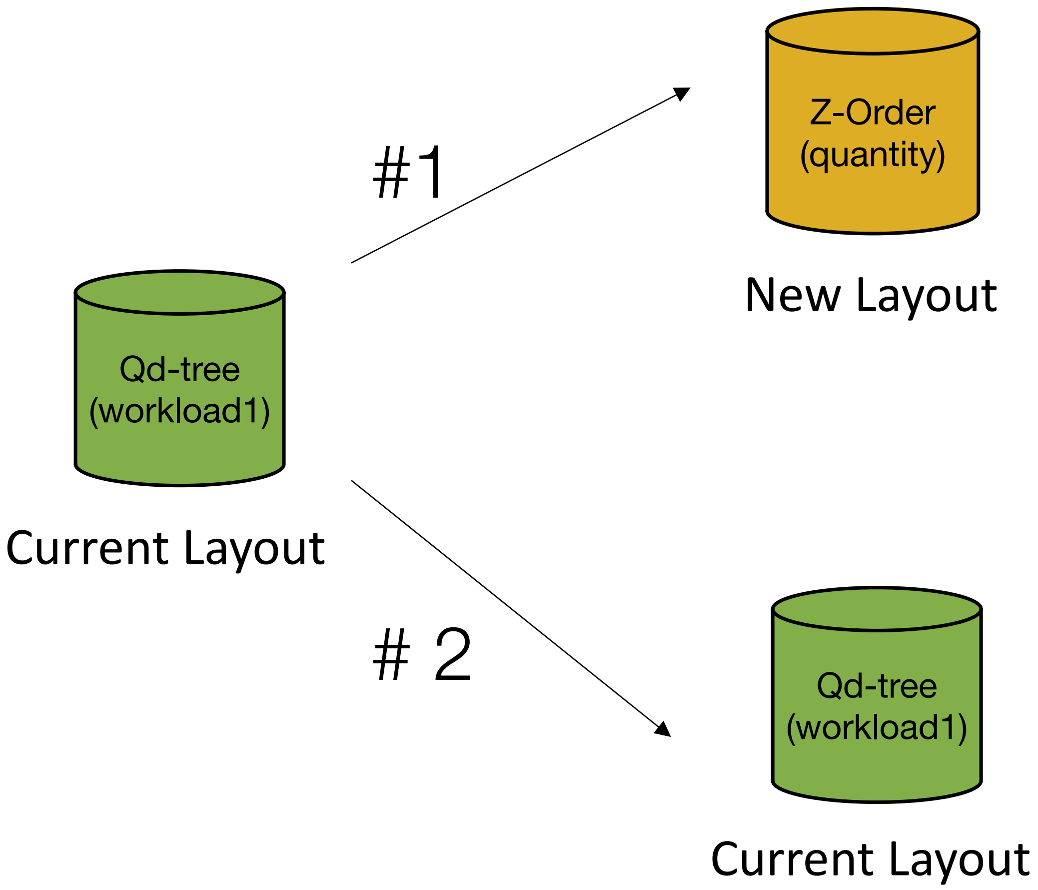
Example: Qd-tree [1]

- Use workload predicates to partition data
- Efficient for target query workloads
- Performance degrades when workload changes



[1] Z. Yang, et al. Qd-tree: Learning Data Layouts for Big Data Analytics. In SIGMOD 2020.

Challenge: dealing with workload changes



Option 1: Change layout

Reorganization cost +

Query cost -

Option 2: Do nothing

Reorganization cost

Query cost +

Problem Setup

system dependent parameter

Objective: minimize $\text{total cost} = \text{query} + \alpha \cdot \#\text{reorgs}$

Input: unknown sequence of queries

Output: *when* and *how* to reorganize

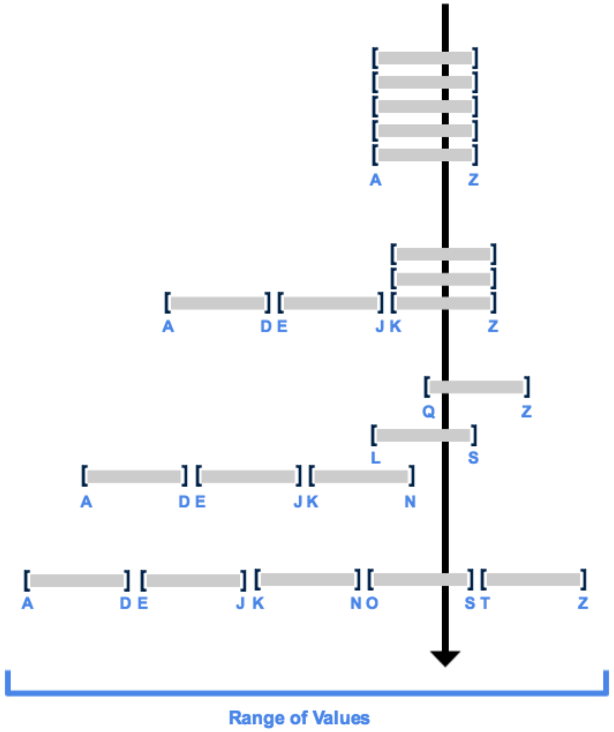
Current practices

Based on rules and heuristics

- Snowflake [1]: monitor “clustering depth” (#partitions that overlap at the same point)
- SAT [2]: monitor the ratio of actual query selectivity and data skipping rate
- No guarantee on performance

Based on future workload behaviors

- MTO [3]: can run q more queries from the same distribution before the next workload shift
- Assumptions on workload distribution



[1] https://docs.snowflake.com/en/user-guide/tables-clustering-micropartitions
[2] X. Xie, et al. “Sat: sampling acceleration tree for adaptive database repartition,” World Wide Web, vol. 26, no. 5, pp. 3503–3533, 2023.
[3] J. Ding, et al. “Instance-optimized data layouts for cloud analytics workloads,”, SIGMOD 2021.

Our idea: leveraging online algorithms

Specifically, we adapt classic algorithms for *Metrical Task Systems*[1]

- Does NOT rely on assumptions of future workload
- Provide guarantees in the form of competitive ratio

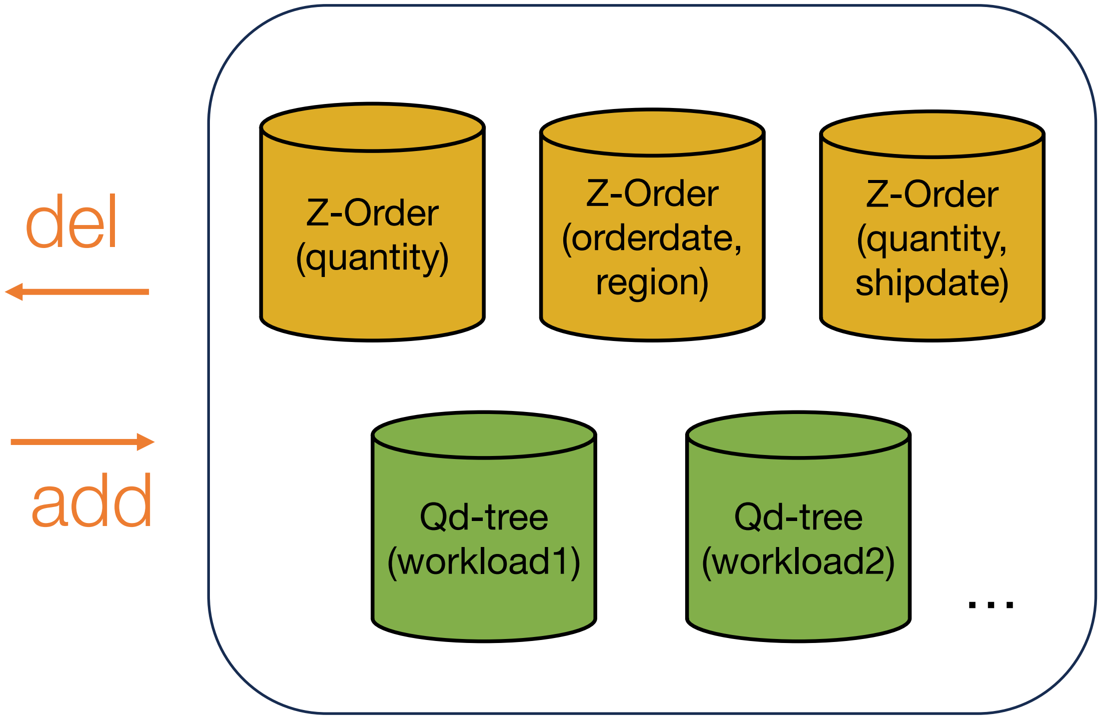
$$\sup_{I} \frac{cost(online\ algorithm)}{cost(offline\ algorithm)}$$

e.g., a single data layout optimized for the entire workload

[1] A. Borodin, N. Linial, and M. E. Saks, "An optimal on-line algorithm for metrical task system," J. ACM, vol. 39, no. 4, pp. 745–763, 1992.

Key Challenge: working with a dynamic state space

State Space S:



MTS:

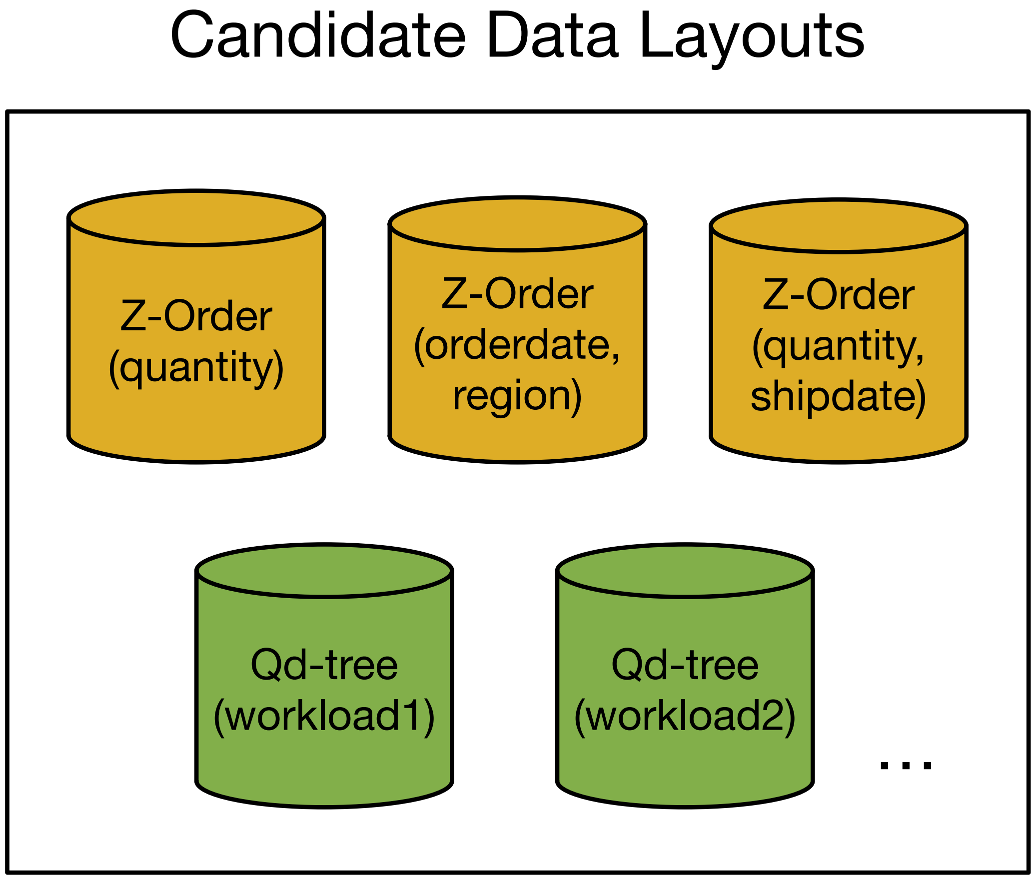
- Assume a fixed set S of candidate data layouts
- Competitive ratio $\sim \log(|S|)$

In practice:

- As new queries are processed, new layouts are generated on the fly

Our contribution: Introduce a dynamic variant of MTS (D-UMTS), and an algorithm that solves it with a tight competitive ratio $\sim \log(|S_{max}|)$

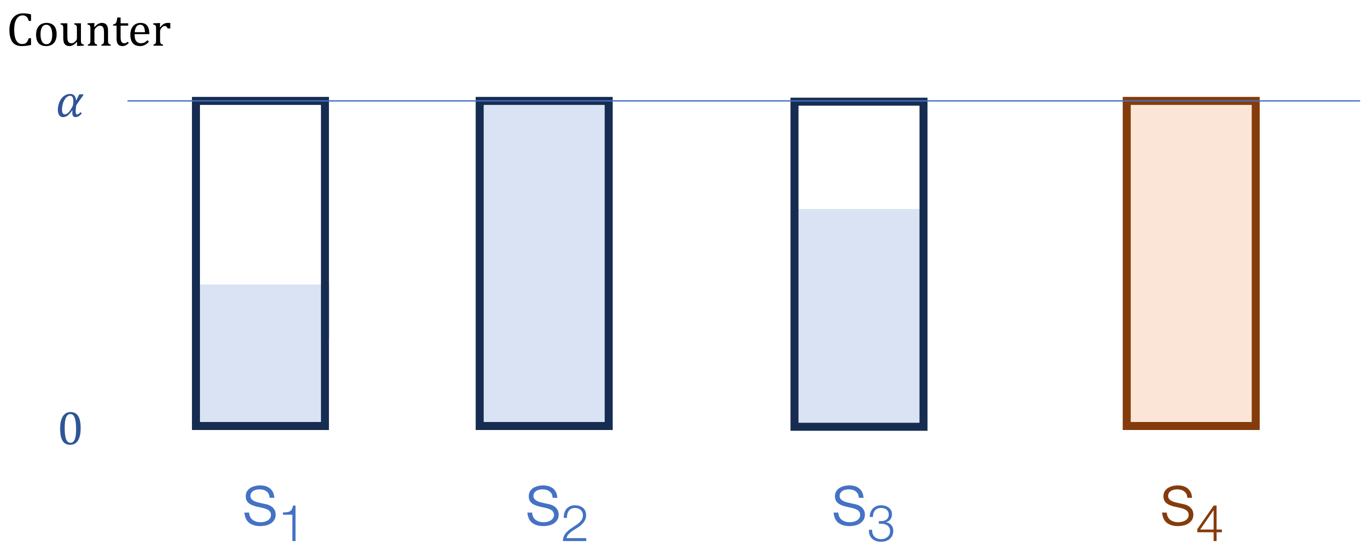
OREO Overview



* Reorganization happens in a separate background process and does not block queries

Reorganizer: how to switch?

The classic algorithm of Borodin, Linial, and Saks[1]

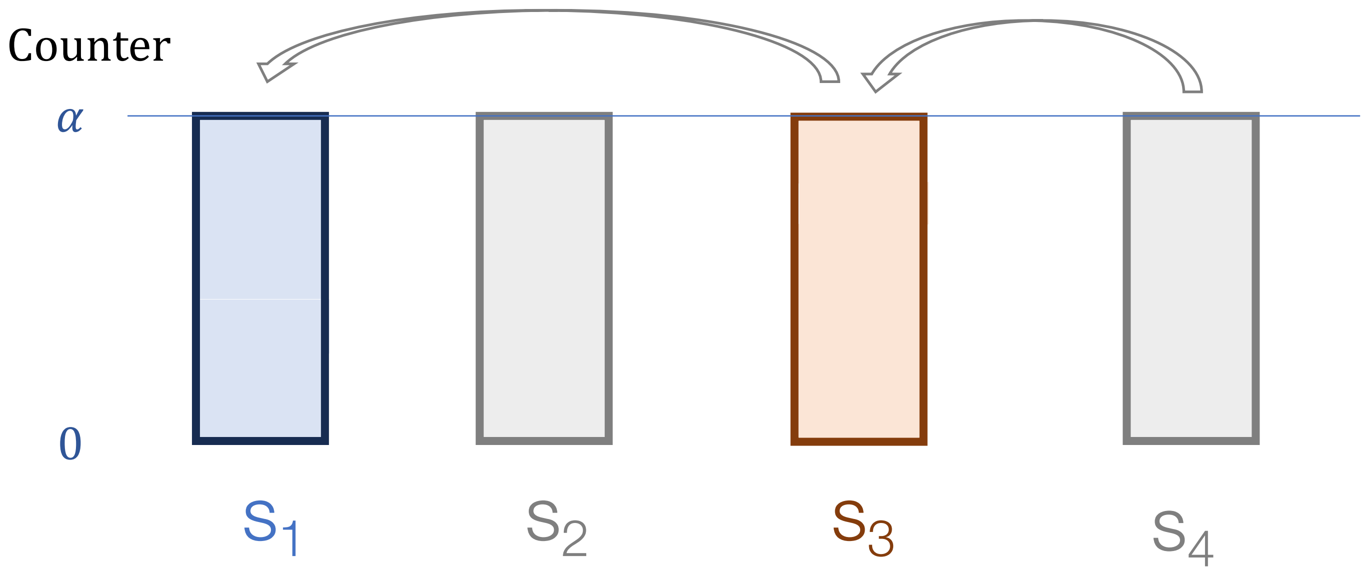


- For each query q: increase counter for S_i by $c(q, s_i)$
  - $c(q, s_i)$: % tables read
- When a counter is full ($>= \alpha$), *randomly* switch to a state whose counter is not full

[1] A. Borodin, N. Linial, and M. E. Saks, "An optimal on-line algorithm for metrical task system," J. ACM, vol. 39, no. 4, pp. 745–763, 1992.

Reorganizer: how to switch?

The classic algorithm of Borodin, Linial, and Saks[1]



- For each query q: increase counter for S_i by $c(q, s_i)$
  - $c(q, s_i)$: % tables read
- When a counter is full ($>= \alpha$), *randomly* switch to a state whose counter is not full

A phase ends when all counters are full. Reset all counters to 0

[1] A. Borodin, N. Linial, and M. E. Saks, "An optimal on-line algorithm for metrical task system," J. ACM, vol. 39, no. 4, pp. 745–763, 1992.

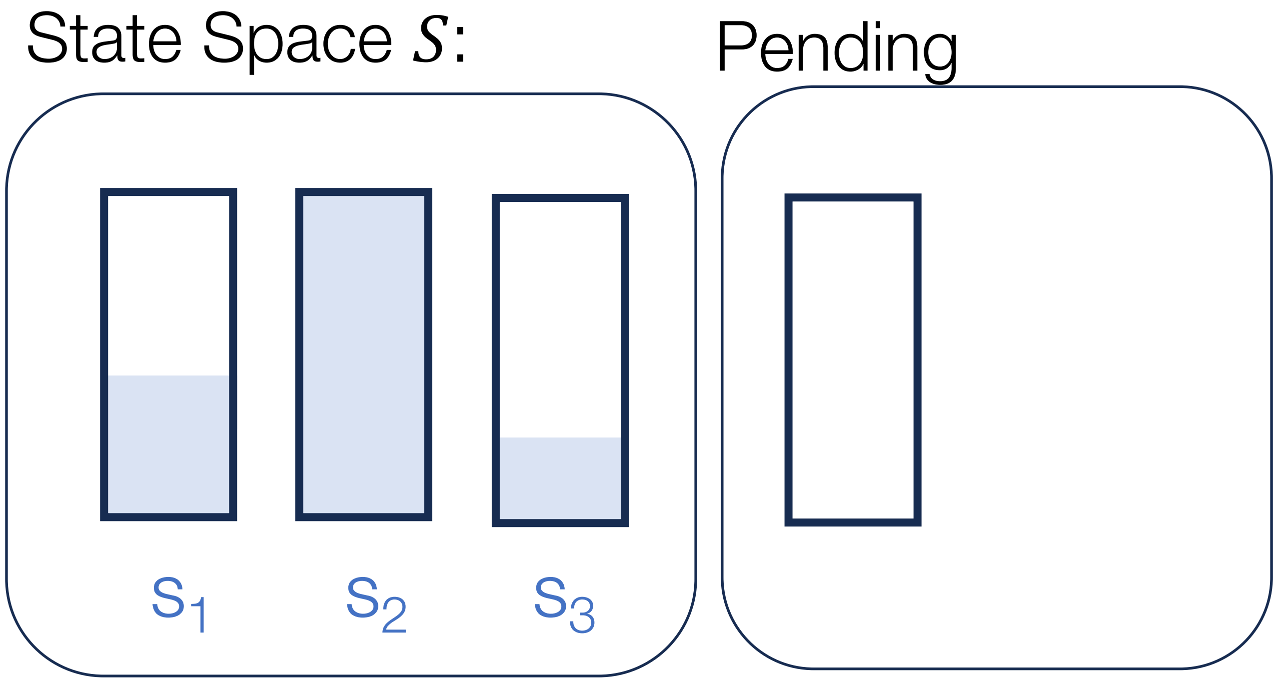
Reorganizer: how to switch?

Handling dynamic state space:

- Insert: delay until next phase
- Delete: set counter to full

Other enhancements

- Non-uniform transition distribution



Theorem. The modified algorithm solves D-UMTS with competitive ratio $2H(|S_{max}|) \leq 2(1 + log\,|S_{max}|)$.

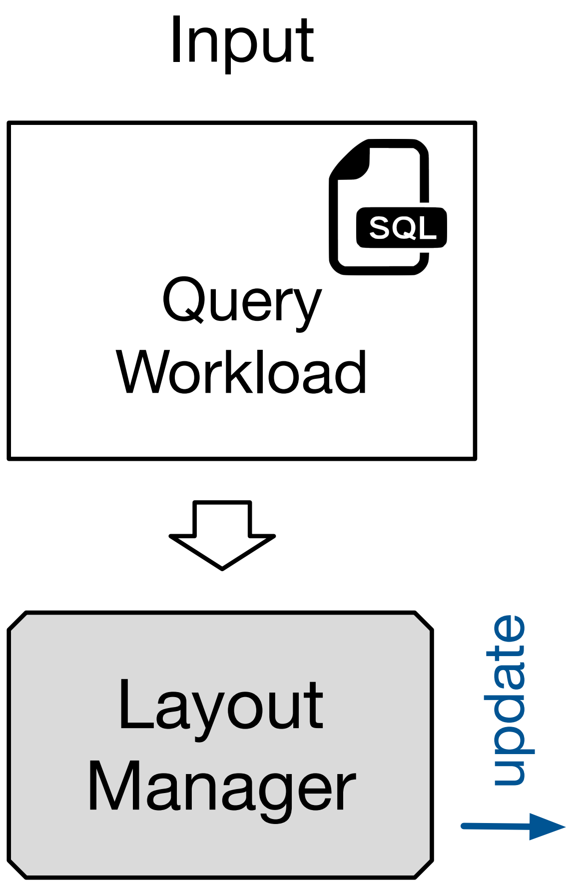
Layout Manager: which layouts to use?

generate_layout(D, Q, k):

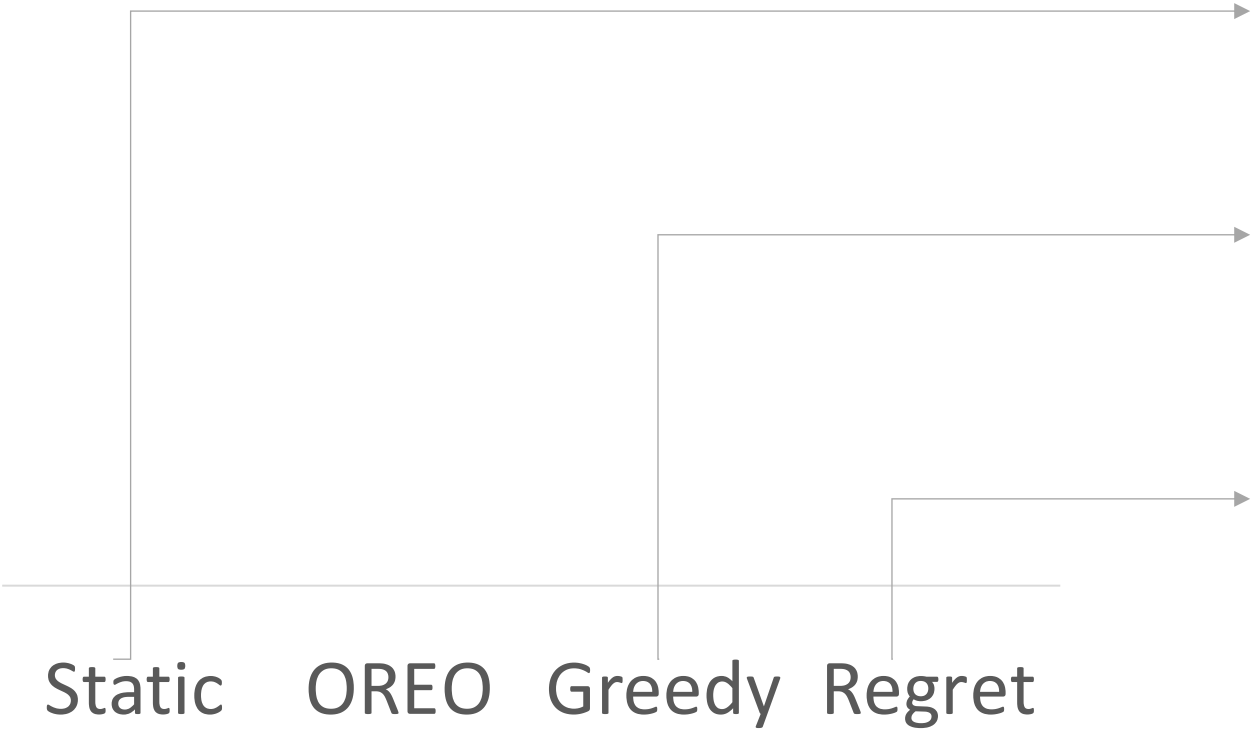
- D: a sample of datasets
- Q: a sliding window of recent queries
- k: # total partitions

Selectively admit new layouts

- competitive ratio $\sim \log(|S_{max}|)$
- Only admit a new layout to the state space if it's different enough from all existing ones



Evaluation: End-to-end Time in Spark



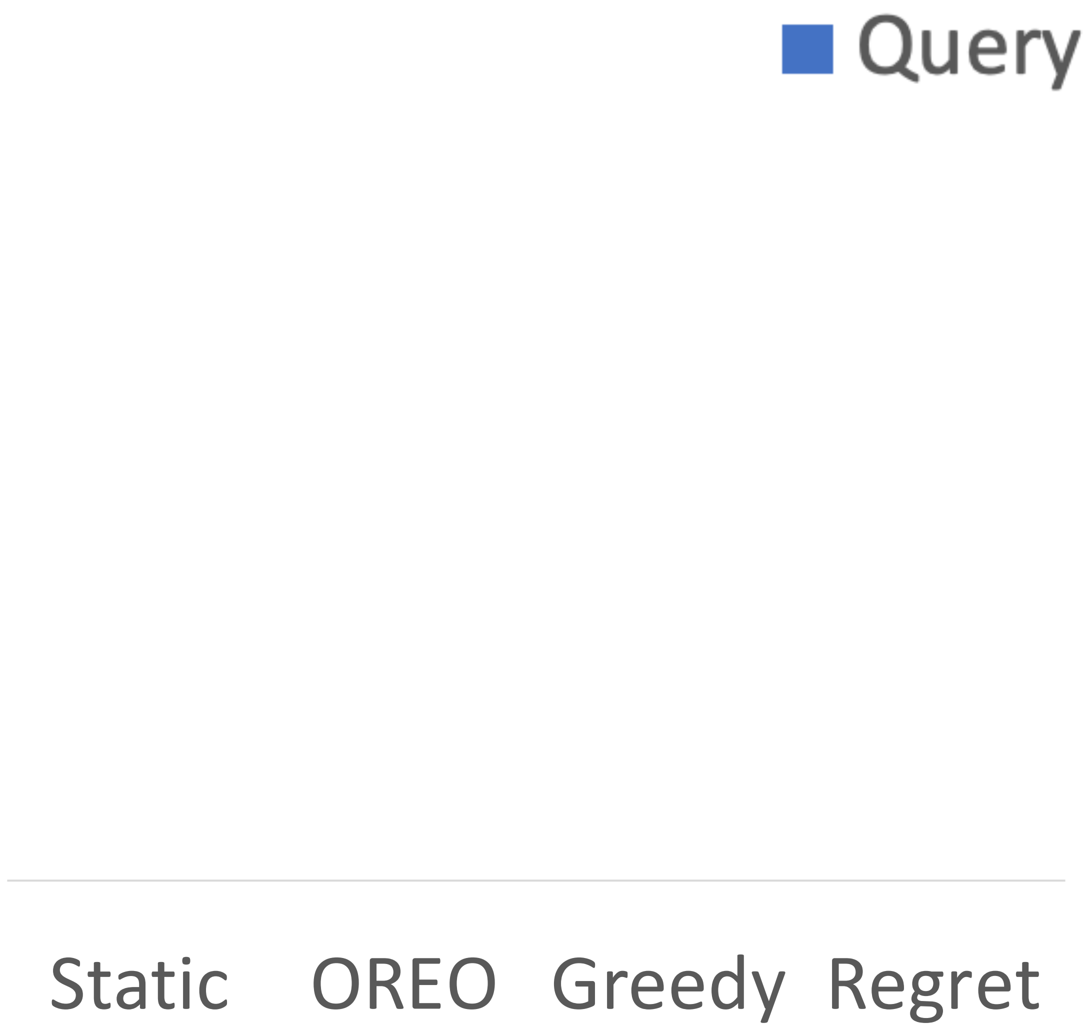
Best offline layout for the entire query workload

Switch to new layout if better

Switch to new layout if cumulative regret > reorganization cost [1]

[1] M. Daum et al., “Tasm: A tile-based storage manager for video analytics, ICDE 2021

Evaluation: End-to-end Time in Spark



Dataset

- TPC-H* (sf=100)

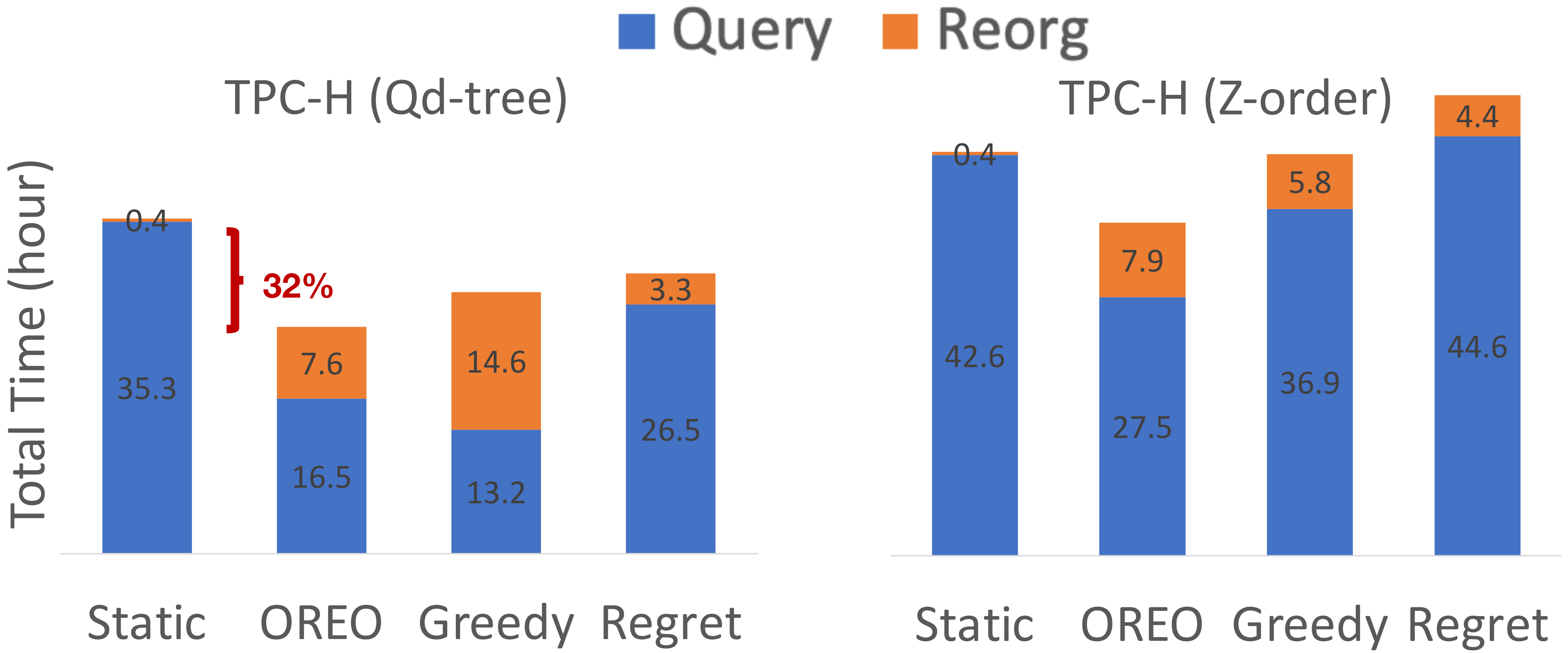
Workload

- 30k queries
- 20 templates

Metric:

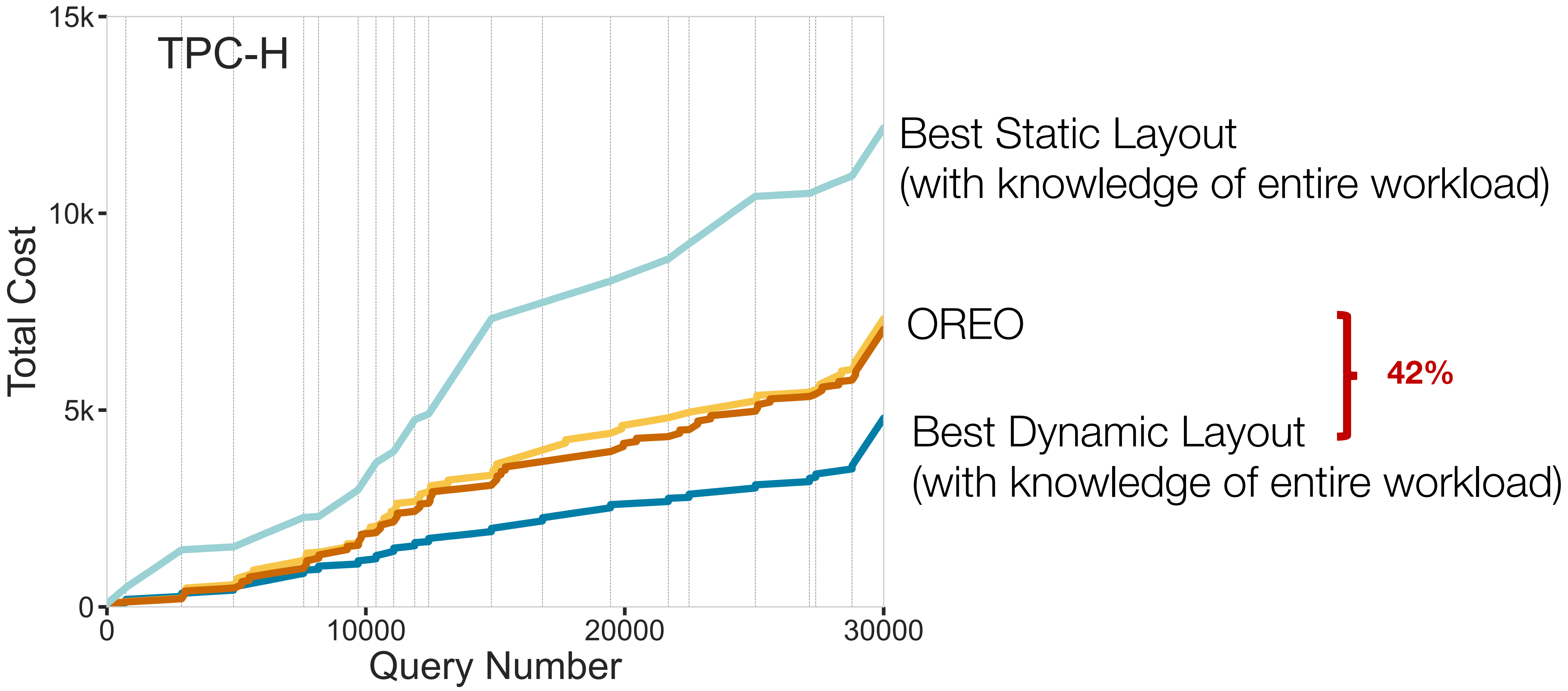
- query + reorganization time

Evaluation: End-to-end Time



Up to 32% improvement over a static data layout optimized for the entire workload

Evaluation: Gap to Offline Optimal



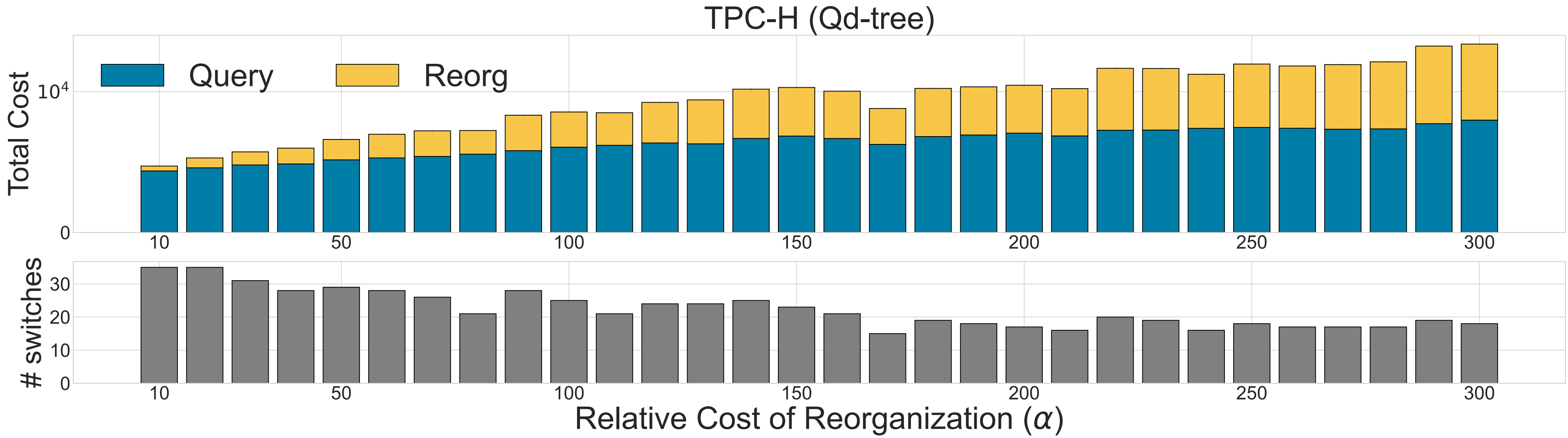
More experiments in the paper

Additional workloads:

- TPC-DS, production workload from an internal data platform

Detailed analysis

- Effect of reorganization cost α
- Sliding window vs reservoir sampling
- …



Conclusion and Future Works

OREO
- MTS with dynamic state space
- Up to 32% improvement in combined query and reorganization time compared to using a single, optimized data layout for the entire workload

Assumptions and limitations
- Static dataset
- Workload change rate

Future work
- Non-uniform MTS
- Additional storage budget for hosting multiple data layouts

Code: https://github.com/d2i-lab/oreo