REVISITING REUSE

IN (MAIN-MEMORY) DATABASES

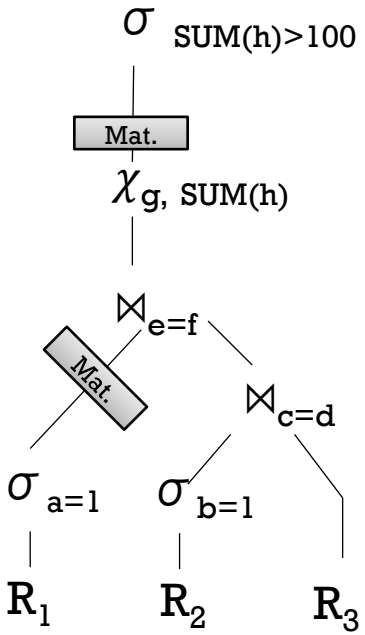
KAYHAN DURSUN*, CARSTEN BINNIG, UGUR CETINTEMEL, TIM KRASKA

REUSE IN THE REAL WORLD …

… MIGHT HELP!

… BUT MIGHT ALSO HURT!

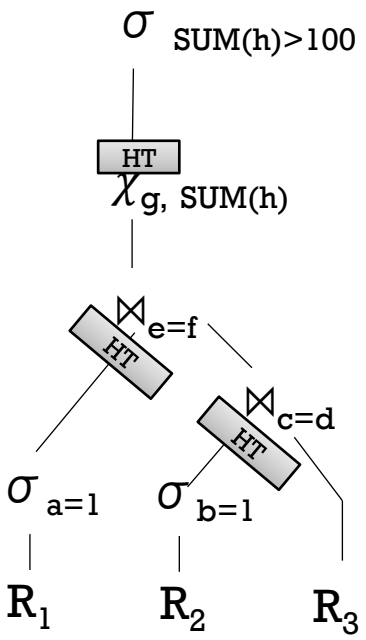
REUSE IN DATABASES MIGHT HURT …



1. **Reason about reuse** potential of intermediate results
2. **Add additional materialization** operations to query plan
3. Analyze if subsequent query can **reuse cached intermediates**

Unclear if extra costs pay off in future

REUSE WITHOUT REGRET …



Main idea: Reuse internal structures

- Keep internal data structures
- Reuse for subsequent queries

Savings are two-fold

- **No** additional **materialization costs**
- **No** need to **re-create internal structures**

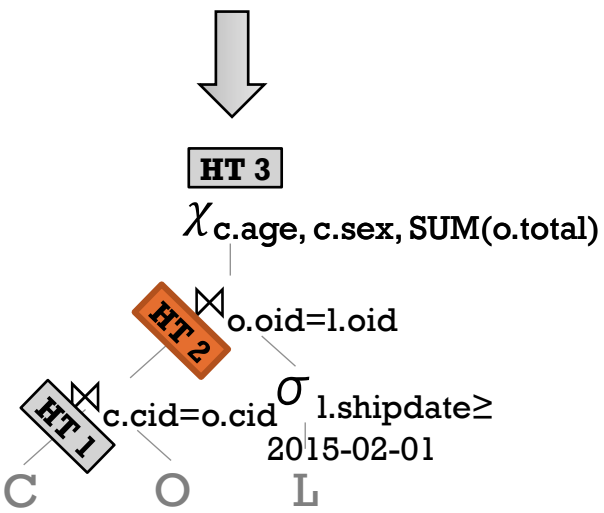
More robust towards different reuse-potentials

This talk: reuse hash tables for **joins + aggregations**

CASE I: EXACT REUSE

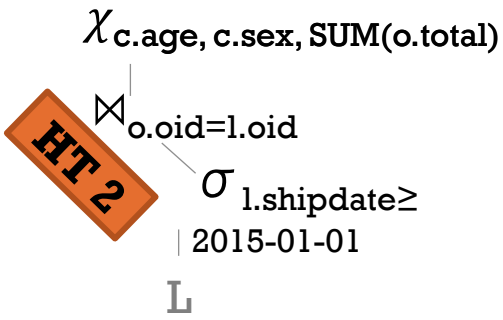
Initial Query

```
SELECT c.age, c.sex,
  SUM(o.total)
FROM C, O, L
WHERE <join-conditions>
AND l.shipdate≥2015-02-01
GROUP BY c.age, c.sex
```



Reuse Query

```
SELECT c.age, c.sex,
  SUM(o.total)
FROM C, O, L
WHERE <join-conditions>
AND l.shipdate≥2015-01-01
GROUP BY c.age, c.sex
```

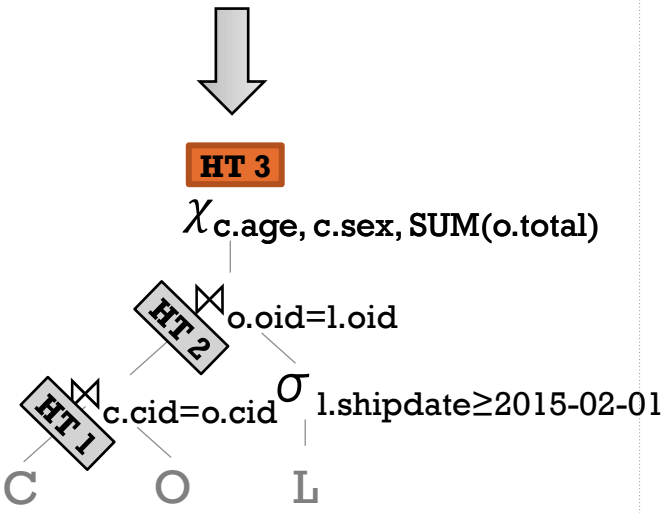


Exact-Reuse HT2

CASE II: PARTIAL REUSE

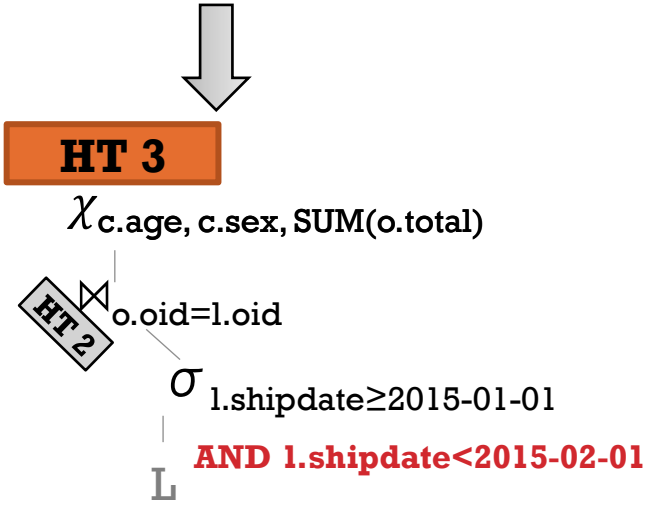
Initial Query

```
SELECT c.age, c.sex,
  SUM(o.total)
FROM C, O, L
WHERE <join-conditions>
AND l.shipdate≥2015-02-01
GROUP BY c.age, c.sex
```



Reuse Query

```
SELECT c.age, c.sex,
  SUM(o.total)
FROM C, O, L
WHERE <join-conditions>
AND l.shipdate≥2015-01-01
GROUP BY c.age, c.sex
```

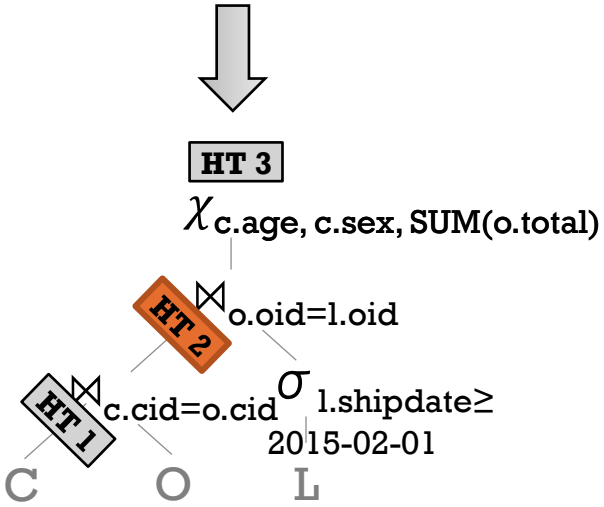


Partial-Reuse HT3: Add missing tuples

CASE III: SUBSUMING REUSE

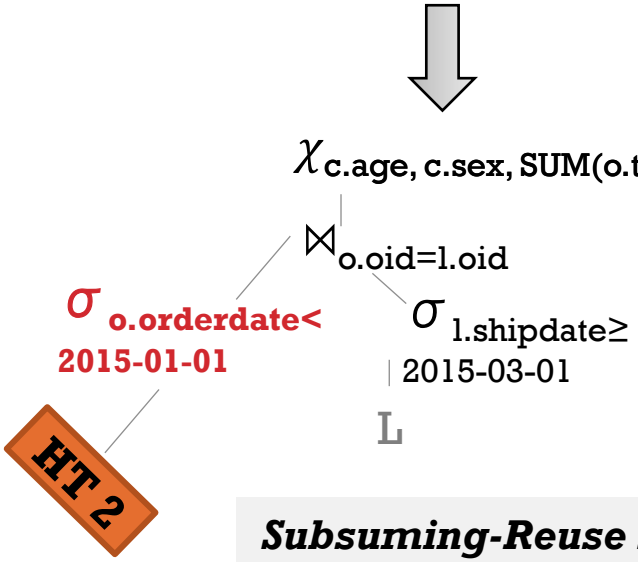
Initial Query

```
SELECT c.age, c.sex,
  SUM(o.total)
FROM C, O, L
WHERE <join-conditions>
AND l.shipdate≥2015-02-01
GROUP BY c.age, c.sex
```



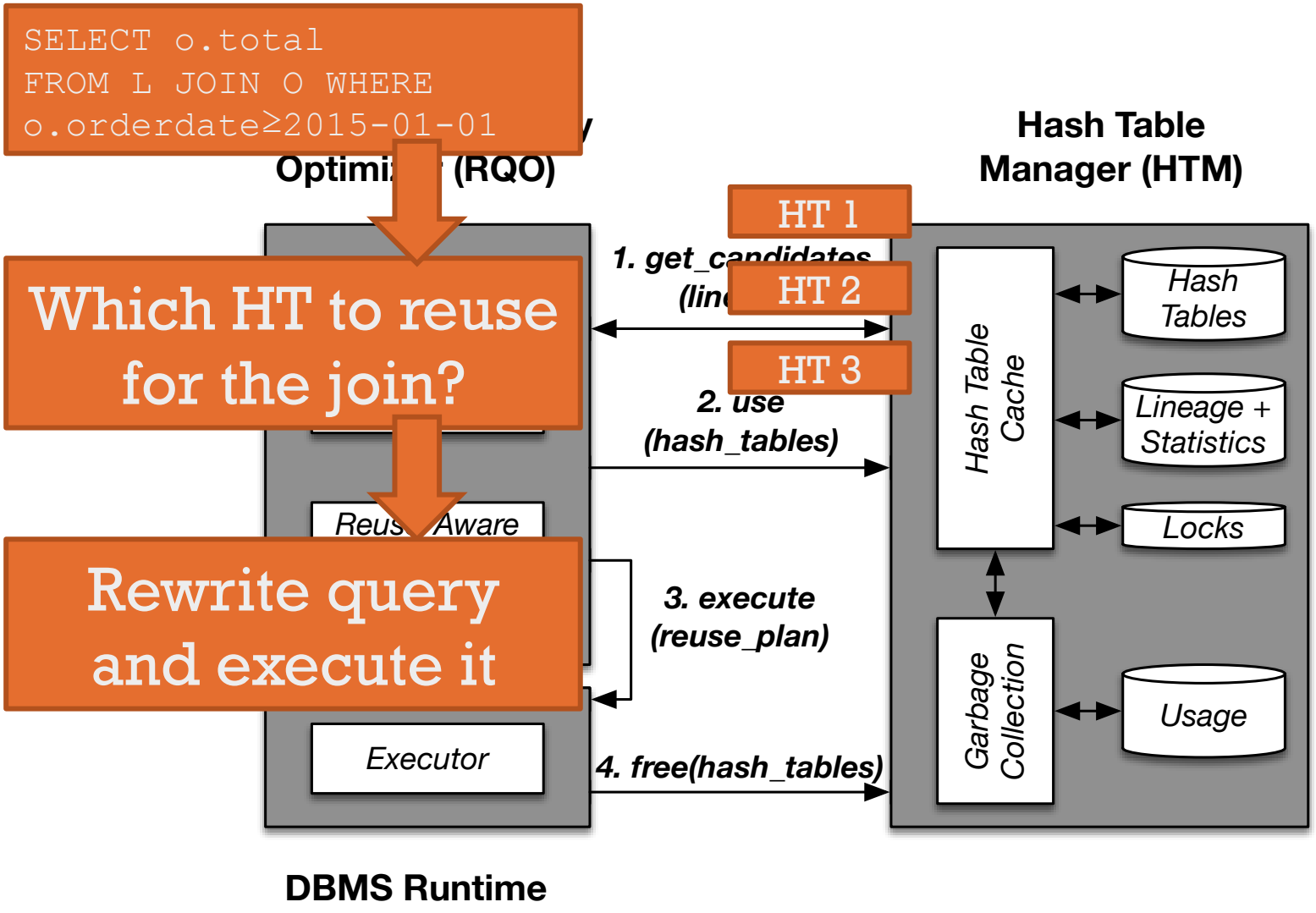
Reuse Query

```
SELECT c.age, c.sex,
  SUM(o.total)
FROM C, O, L
WHERE <join-conditions>
AND l.shipdate≥2015-02-01
AND o.oderdate<2015-01-01
GROUP BY c.age, c.sex
```



Subsuming-Reuse HT2: Filter false positives

HASH-STASH ARCHITECTURE



COST-MODEL FOR REUSE IN JOINS

Cost of a Hash-⋈: $c_{build}(HT) + c_{probe}(HT)$

How is that different for a RHJ?

$$c_{RHJ} = c_{resize}^{R}(HT) + c_{build}^{R}(HT) + c_{probe}^{R}(HT)$$

1. add only missing tuples

2. higher per-tuple cost (due to add. tuples)

3. post-filter false positives (i.e. a=0)

Reuse Query:

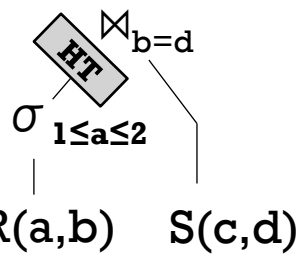


Table R

a	b
0	1
0	2
1	3
1	4
2	5
3	6

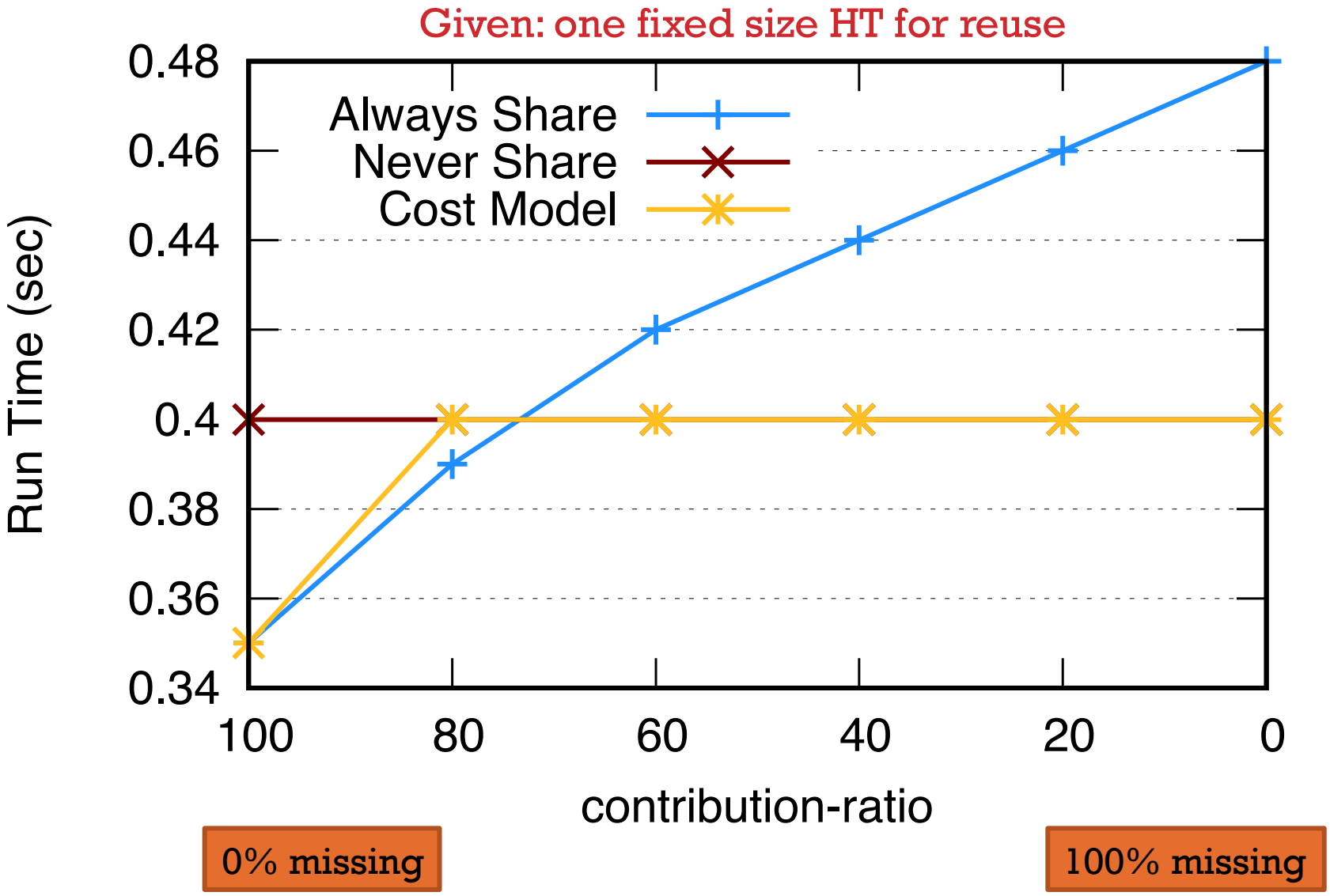
HT: **0≤a≤1**

b	[a,b]
1	[0,1]
2	[0,2]
3	[1,3]
4	[1,4]
5	[2,5]

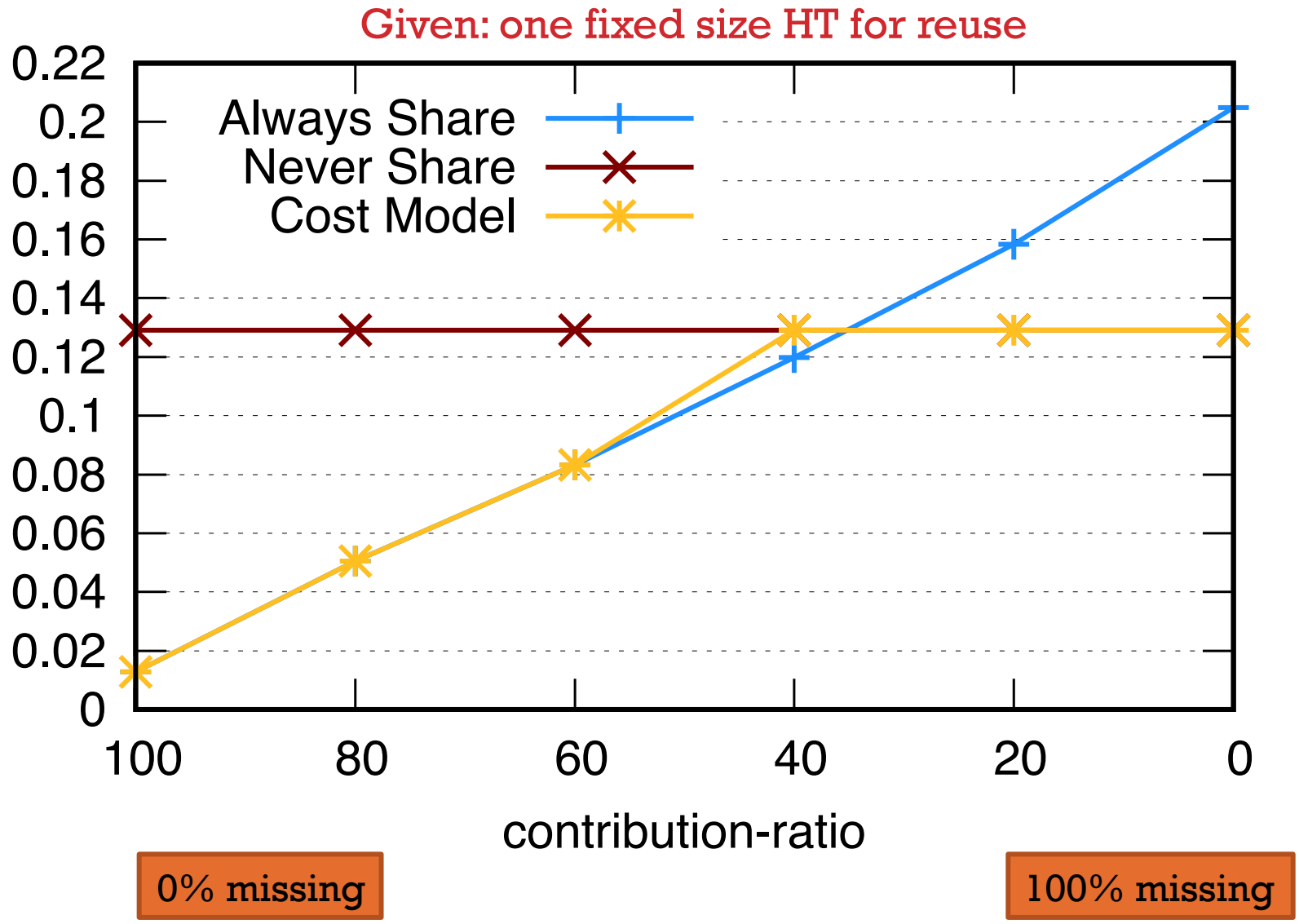
Table S

c	d
1	1
2	1
3	2
4	3
5	4
6	5

JOIN: REUSE OR NOT REUSE?



AGGREGATION: REUSE OR NOT REUSE?



ADDITIONAL OPTIMIZATIONS

Store additional attributes in HTs

- Enables post-filtering false positives
- Only if extension does not exceed cache line

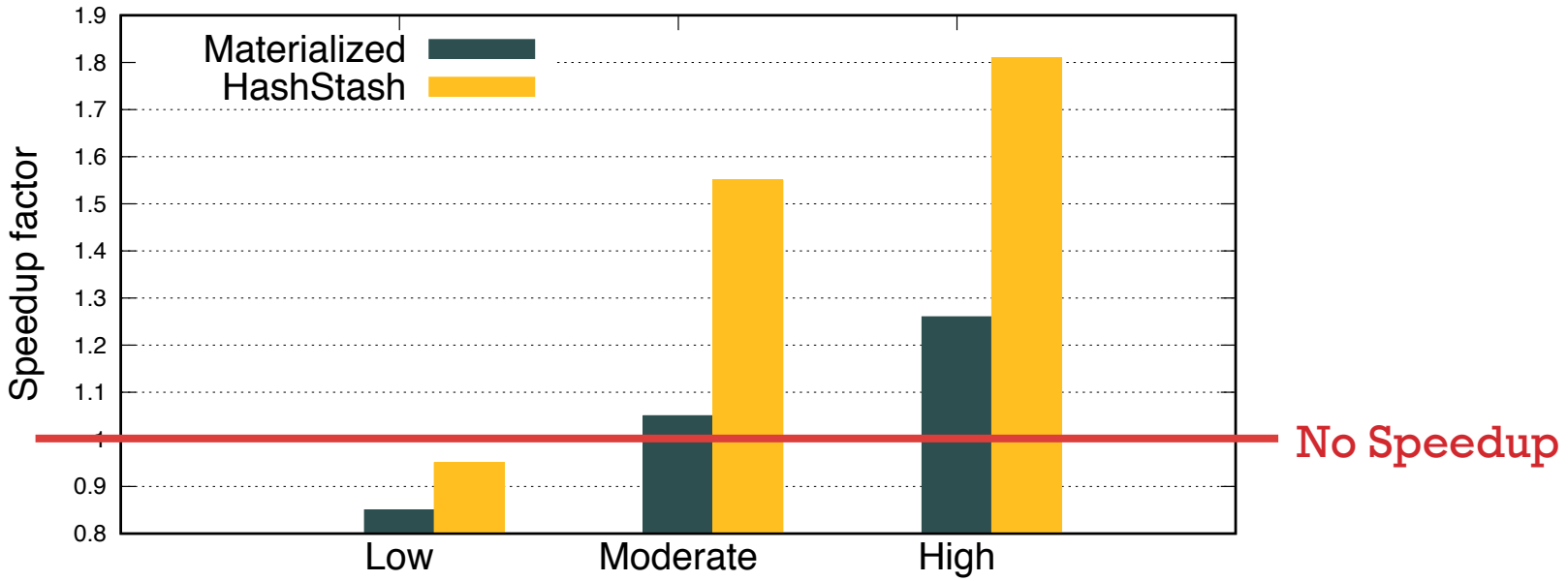
Aggregate Optimizations

- Always store COUNT and SUM instead of AVG
- Add additional aggregate functions

Speculative Optimizations

- Join re-ordering to produce "better" hash-tables
- Store aggregate hash table on finer granularity

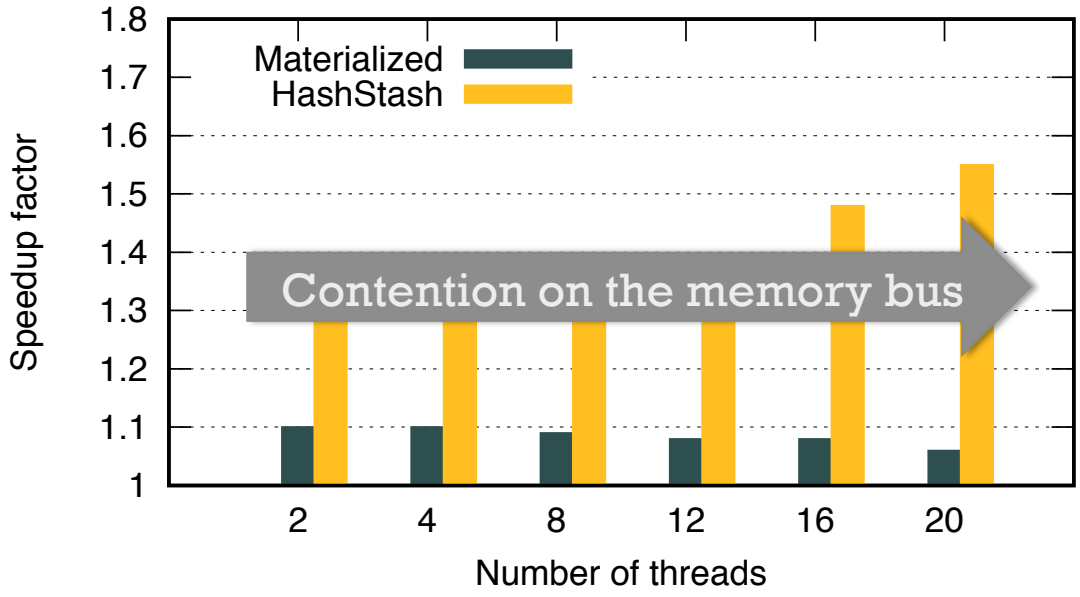
RESULTS: REUSE EFFICIENCY



Workloads: 64 queries with different reuse potentials

- **Low:** 1% of the cached data is reused on average
- **Medium:** 10% of the cached data is reused on average
- **High:** 50% of the cached data is reused on average

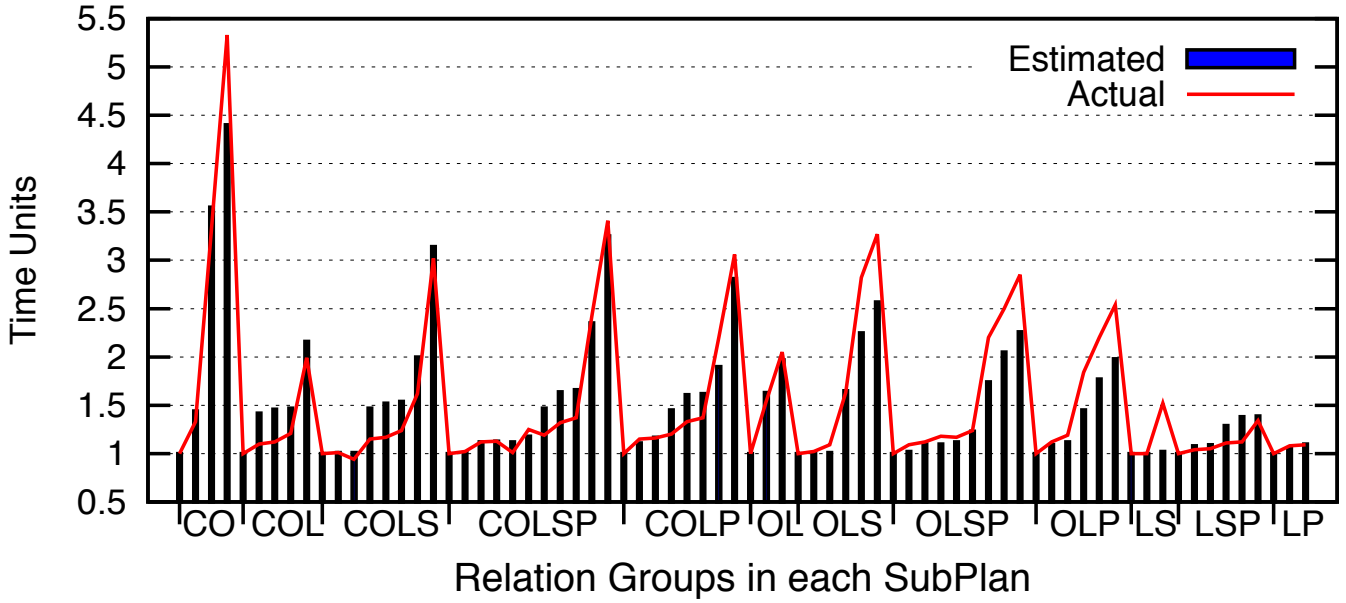
RESULTS: EFFECT OF MULTI-THREADING



Workload: Varying the degree of parallelism

- **Medium-reuse:** 64 queries
- **Thread:** 2-20 threads

RESULTS: ACCURACY OF COST MODEL



Workload:

- **5-way join:** Customer, Orders, Lineitem, Supplier, Part
- **For each sub-plan:** enumerated all HTs that qualify

SUMMARY AND OUTLOOK

Main idea:

- Reuse internal data structures of operators
- More robust towards different workloads

Other topics not covered in this talk

- Reuse-aware hash aggregate
- Top-down plan enumeration algorithm
- Query batch interface

Full paper at SIGMOD 2017