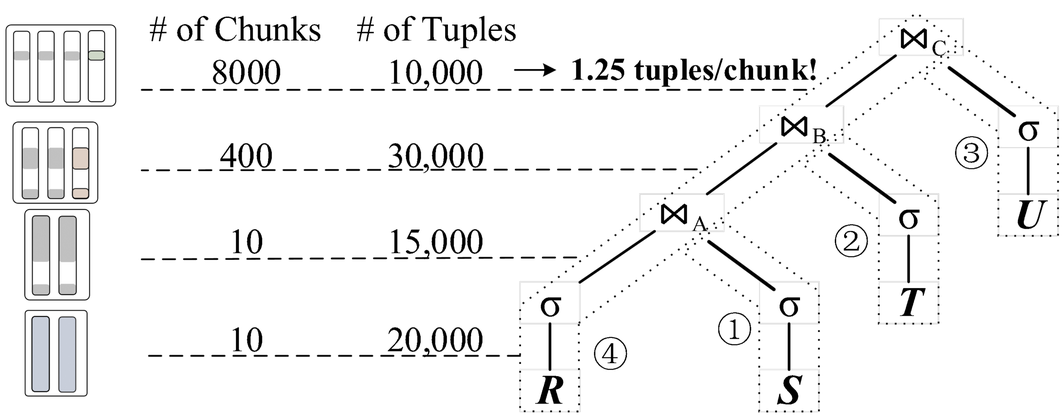
Data Chunk Compaction in Vectorized Execution

Yiming Qiao, Huanchen Zhang

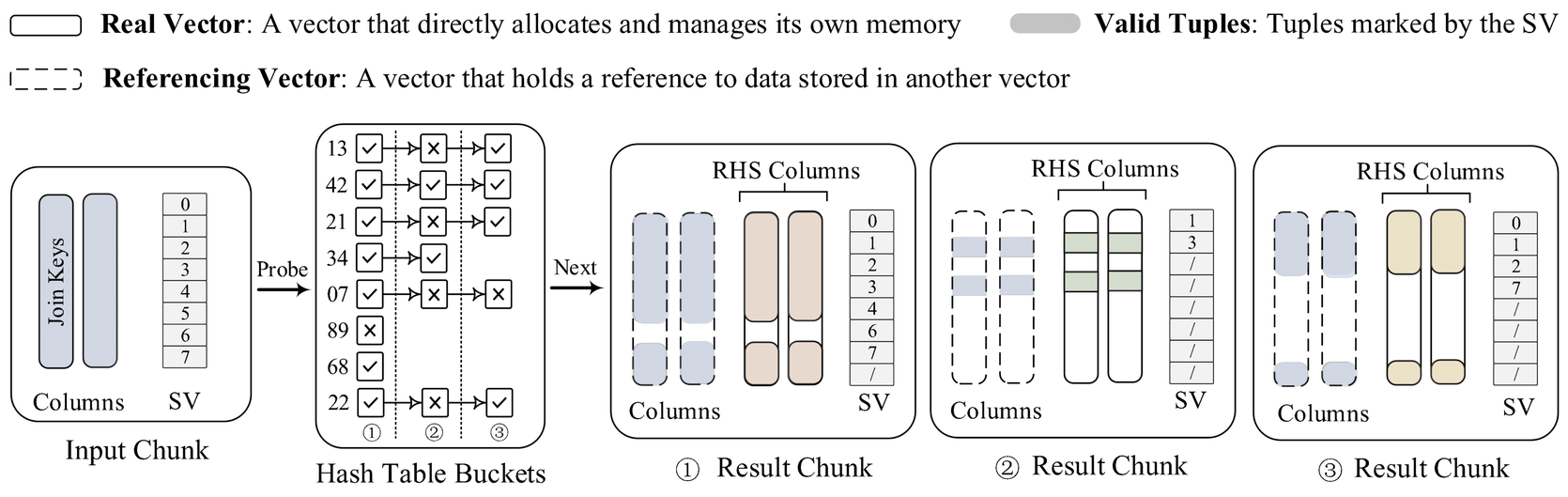
A Performance Issue Caused By Small Chunks In Vectorized Execution



- Vectorized engines process data in fixed-size chunks.
- **Filters and joins** often produce Small chunks.
- Example: 10,000 tuples in 8,000 chunks → **1.25 tuples/chunk**.
- Consequences:
  - ✗ High per-chunk overhead
  - ✗ Poor SIMD/cache utilization
  - Industry also Reports this Problem: Velox Issue-7801

Vectorized Hash Join: Zero-Copy Technique vs. Chunk Size

- Vectorized hash join probes the hash table row by row. Each probe may match a different number of tuples, causing output skew.
- To ensure **Zero-Copy**, each match set becomes a separate small chunk.
- This results in one **input chunk leading to Many Output Chunks.**



Zero-Copy Preferred Database: DuckDB, Velox

Chunk Size Preferred Database: Datafusion, CockroachDB

Data Chunk Compaction: Memory Copy vs. Interpretation Overhead

When to Compact?

- For **large** tuples, we should compact **conservatively**;
- For **small** tuples, we should compact **aggressively**.

How to Compact?

- Actually, not all memory copy is **necessary** for compaction...
- **We compact small chunks before they are generated.**

Our Solution 1: Learning Compaction

Pipeline Executor t_1 (Processing Latency) t_2 t_3

$R_{\text{chunk}} \rightarrow \bowtie_A \rightarrow \alpha_1 \rightarrow \sigma \rightarrow \alpha_2 \rightarrow \bowtie_B \rightarrow \alpha_3 \rightarrow \bowtie_C$

For each chunk from *R:*

For each threshold α_i: α_i = SelectArm(*i*); // before execution

For each threshold α_i: UpdateArm(*i*, α_i, $t_i + ... + t_3$) // after execution

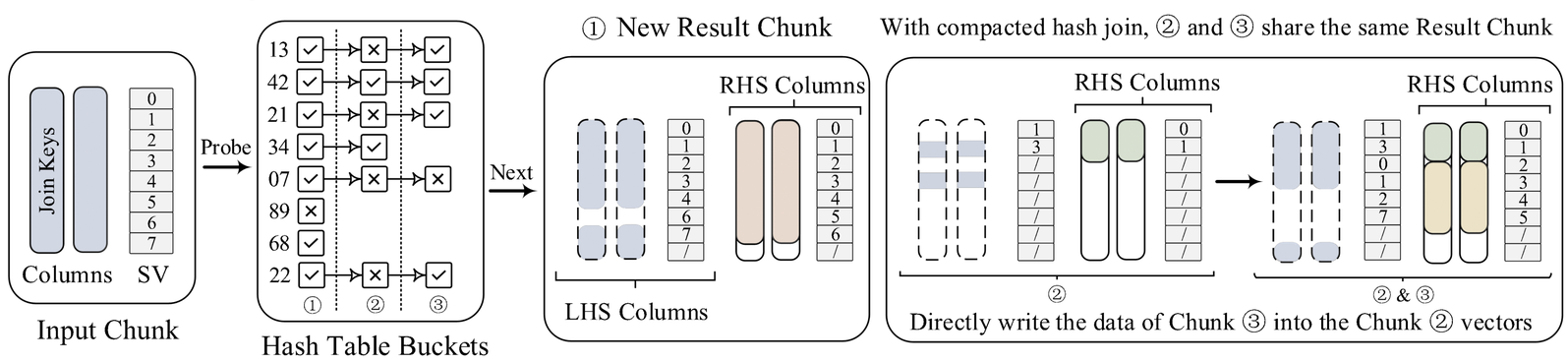
Compaction Learner has interfaces: SelectArm and UpdateArm.

Model 1			Model 2			Model 3		
Arm	Reward	Confidence	Arm	Reward	Confidence	Arm	Reward	Confidence
0	1.2	3	0	1.5	3	0	5.5	879
128	1.3	3	128	3.4	123	128	4.8	78
1024	2.4	97	1024	2.1	13	1024	4.1	9

SelectArm(*i*): returns the arm with highest reward from the Model *i*.

UpdateArm(*i*, α_i, *latency*): updates the arm α_i for the Model *i*.

Our Solution 2: Logical Compaction (Merged Into DuckDB 1.2.0)

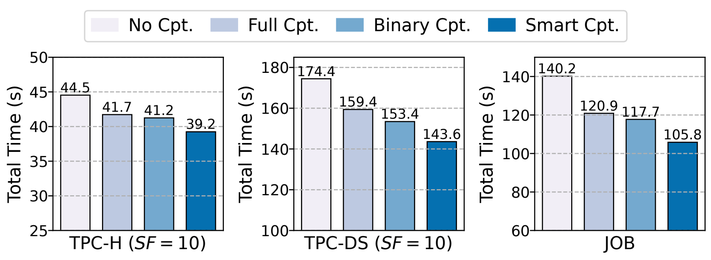


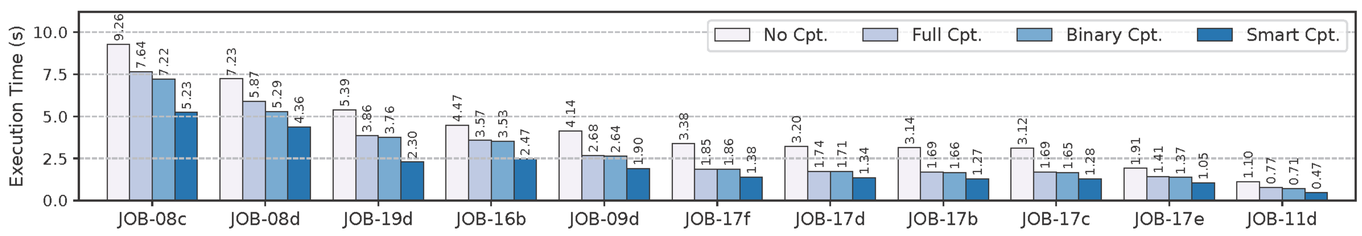
In Vectorized Execution, a Data Chunk can be considered a Sample.

The Proposed Vectorized Hash Join Outputs only one non-full chunk per input chunk, at the cost of adding additional selection vectors.

Stable Improvement Without Any Regressions on All Benchmarks

Our Solution is used by the Team Embryo (Third Prize) in the SIGMOD Programming Contest 2025.





Our Code

Contact: Yiming Qiao · qiaoym21@mails.tsinghua.edu.cn / yimingqiao3163@gmail.com