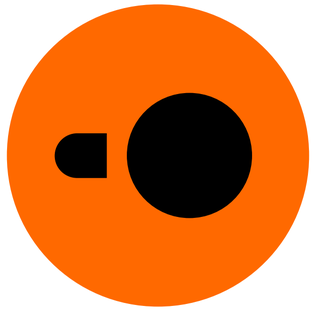
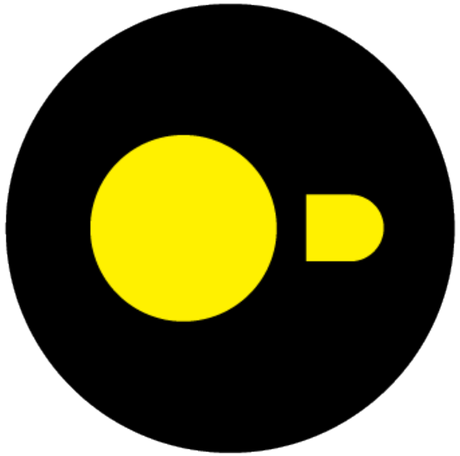


Wrangling Data with DuckDB

Hannes Mühleisen

DuckDB Labs

ALL OPINIONS AND RANTS
IN THIS TALK -- EVEN THOSE THAT
ARE JUSTIFIED -- ARE ONLY MEANT TO
SPARK DISCUSSION. ALL LUMINARY
QUOTES ARE MADE UP ... POORLY.
THE FOLLOWING TALK CONTAINS A
GERMAN ACCENT AND DUE TO ITS
CONTENT SHOULD NOT BE TAKEN
SERIOUSLY BY ANYONE



DuckDB

SeQeL

Simple

```
pip install duckdb

curl install.duckdb.org | bash
```


BATTERIES
INCLUDED

Fast

ClickBench — a Benchmark For Analytical DBMS

Methodology | Reproduce and Validate the Results | Add a System | Hardware Benchmark | Versions Benchmark | See also: JSONBench

System: All | AlloyDB | AlloyDB (tuned) | Athena (partitioned) | Athena (single) | Aurora for MySQL | Aurora for PostgreSQL | Bigquery | ByConity | ByteHouse | chDB (DataFrame) | chDB (Parquet, partitioned) | chDB | CHYT | Citus | ClickHouse Cloud (aws) | ClickHouse Cloud (azure) | ClickHouse Cloud (gcp) | ClickHouse (data lake, partitioned) | ClickHouse (data lake, single) | ClickHouse (Parquet, partitioned) | ClickHouse (Parquet, single) | ClickHouse (web) | ClickHouse | ClickHouse (tuned) | ClickHouse (tuned, memory) | Cloudberry | CrateDB (tuned) | CrateDB

...

c6a.4xlarge, 1500gb gp2 | XL | Jumbo | Pulse | Standard | 16 vCPU 32GB | dc2.8xlarge | ra3.16xlarge | ra3.4xlarge | ra3.xlplus | c6a.4xlarge, 700gb gp2 | S2 | S24 | 2XL | 3XL | 4XL | L1 - 16CPU 32GB | c6a.4xlarge, 500gb gp3 | 16 vCPU 64GB | 4 vCPU 16GB | 8 vCPU 32GB | 64 vCPU 256GB

Cluster size: All | 1 | 2 | 3 | 4 | 8 | 9 | 16 | 32 | 64 | 128 | serverless | 1 | 2 | 3 | undefined

Metric: Cold Run | Hot Run | Load Time | Storage Size

System & Machine	Relative time (lower is better)	
Umbra (c6a.metal, 500gb gp2):		×1.60
Salesforce Hyper (c6a.metal, 500gb gp2):		×1.68
DuckDB (c6a.metal, 500gb gp2):		**×2.15**
ClickHouse (tuned, memory) (c6a.metal, 500gb gp2):		×2.15
ClickHouse (tuned) (c6a.metal, 500gb gp2):		×2.29
ClickHouse (c6a.metal, 500gb gp2):		×2.51

RTABench

a Benchmark For Real Time Analytics

Repo

System: All | TimescaleDB | ClickHouse | Timescale Cloud | MongoDB | DuckDB | Postgres | ClickHouse Cloud (aws) | MySQL

Database Type: All | General Purpose | Real-time Analytics | Batch Analytics

Machine: All | m5.4xlarge, 500gb gp2 | c6a.4xlarge, 500gb gp2 | 4 vCPU 16GB | 12 vCPU 48 GB (3x: 4vCPU 16GB) | 16 vCPU 64GB | 6 vCPU 24 GB (3x: 2vCPU 8GB) | 8 vCPU 32GB | 24 vCPU 96 GB (3x: 8vCPU 32GB)

Cluster size: All | 1 | 3

Metric: Cold Run | Hot Run | Load Time | Storage Size

System and Machine

Relative time (lower is better)

TimescaleDB (c6a.4xlarge, 500gb gp2) ×1.44

TimescaleDB (m5.4xlarge, 500gb gp2) ×1.79

RTABench

a Benchmark For Real Time Analytics

Repo

System: All | TimescaleDB | ClickHouse | Timescale Cloud | MongoDB | DuckDB | Postgres | ClickHouse Cloud (aws) | MySQL

Database Type: All | General Purpose | Real-time Analytics | Batch Analytics

Machine: All | m5.4xlarge, 500gb gp2 | c6a.4xlarge, 500gb gp2 | 4 vCPU 16GB | 12 vCPU 48 GB (3x: 4vCPU 16GB) | 16 vCPU 64GB | 6 vCPU 24 GB (3x: 2vCPU 8GB) | 8 vCPU 32GB | 24 vCPU 96 GB (3x: 8vCPU 32GB)

Cluster size: All | 1 | 3

Metric: Cold Run | Hot Run | Load Time | Storage Size

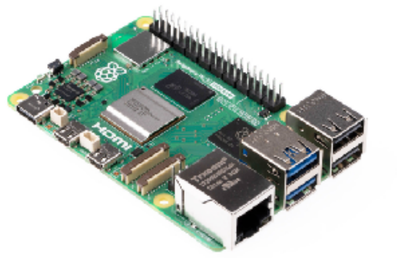
System and Machine

Relative time (lower is better)

- **DuckDB isn't built for real-time analytics, so it's excluded from the main results, but it was the fastest in the benchmark.** Given its popularity, we included it in the benchmark to serve as a point of reference, and it surprised us: It was 3.5x faster than TimescaleDB and 7.3x faster than ClickHouse.

It Scales!

SF 1 000



Raspberry Pi
16 GB RAM

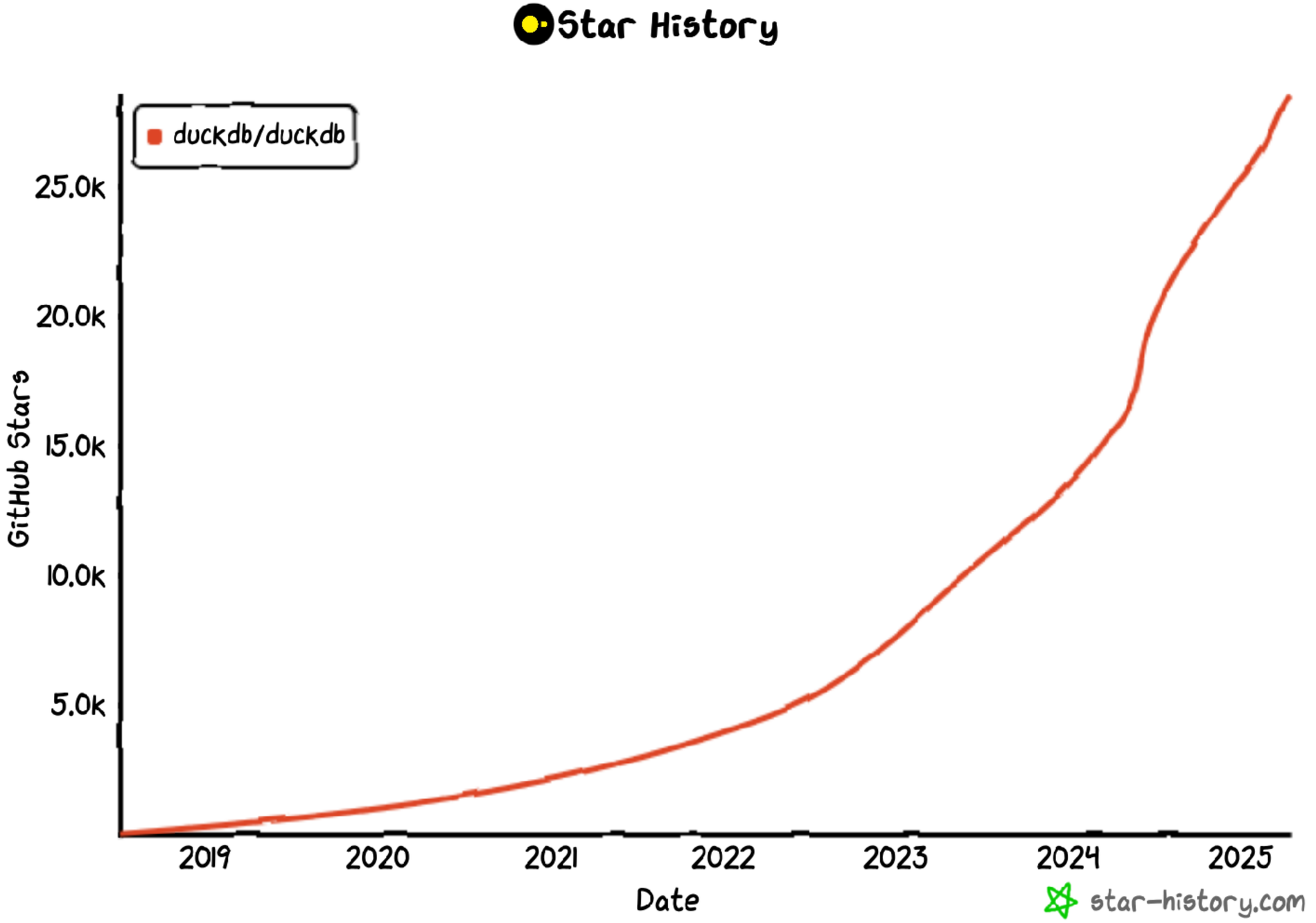
SF 10 000

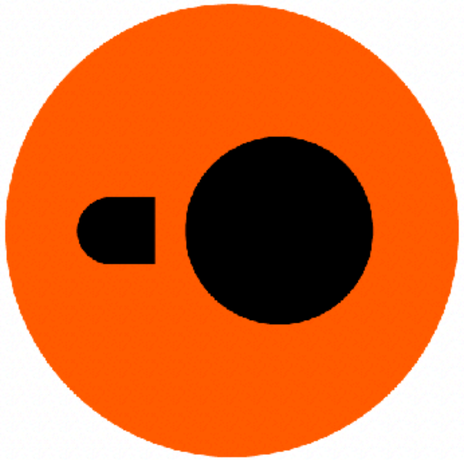
MacBook Pro
128 GB RAM

SF 100 000

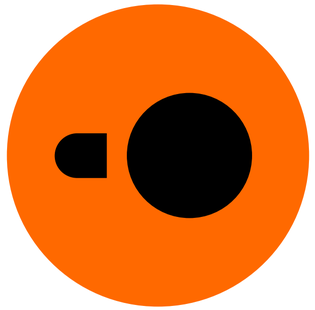
EC2 i7ie.48xlarge
1.5 TB RAM

Free


Star History
duckdb/duckdb
25.0k
20.0k
15.0k
10.0k
5.0k
GitHub Stars
2019
2020
2021
2022
2023
2024
2025
Date
star-history.com



DuckDB Labs



Parallelism

Volcano—An Extensible and Parallel Query Evaluation System

Goetz Graefe

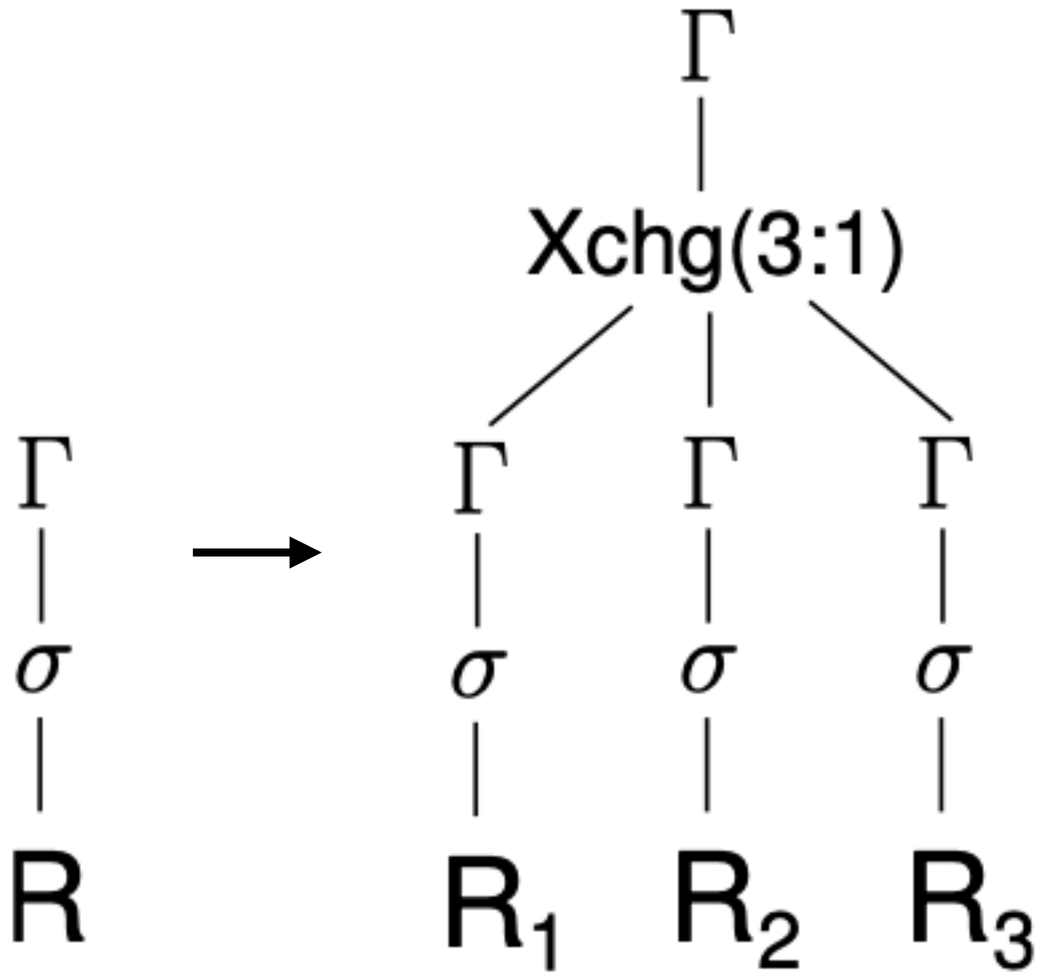
Abstract—**To investigate the interactions of extensibility and parallelism in database query processing, we have developed a new dataflow query execution system called Volcano. The Volcano effort provides a rich environment for research and education in database systems design, heuristics for query optimization, parallel query execution, and resource allocation.**

Volcano uses a standard interface between algebra operators, allowing easy addition of new operators and operator implementations. Operations on individual items, e.g., predicates, are imported into the query processing operators using *support functions*. The semantics of support functions is not prescribed; any data type including complex objects and any operation can be realized. Thus, Volcano is *extensible* with new operators, algorithms, data types, and type-specific methods.

Volcano includes two novel *meta-operators*. The *choose-plan* meta-operator supports *dynamic query evaluation plans* that allow delaying selected optimization decisions until run-time, e.g., for embedded queries with free variables. The *exchange* meta-operator supports *intra-operator parallelism* on partitioned datasets and both *vertical and horizontal inter-operator parallelism*, translating between demand-driven dataflow within

tem as it lacks features such as a user-friendly query language, a type system for instances (record definitions), a query optimizer, and catalogs. Because of this focus, Volcano is able to serve as an experimental vehicle for a multitude of purposes, all of them open-ended, which results in a combination of requirements that have not been integrated in a single system before. First, it is modular and extensible to enable future research, e.g., on algorithms, data models, resource allocation, parallel execution, load balancing, and query optimization heuristics. Thus, Volcano provides an infrastructure for experimental research rather than a final research prototype in itself. Second, it is simple in its design to allow student use and research. Modularity and simplicity are very important for this purpose because they allow students to begin working on projects without an understanding of the entire design and all its details, and they permit several concurrent student projects. Third, Volcano's design does not presume any

1994

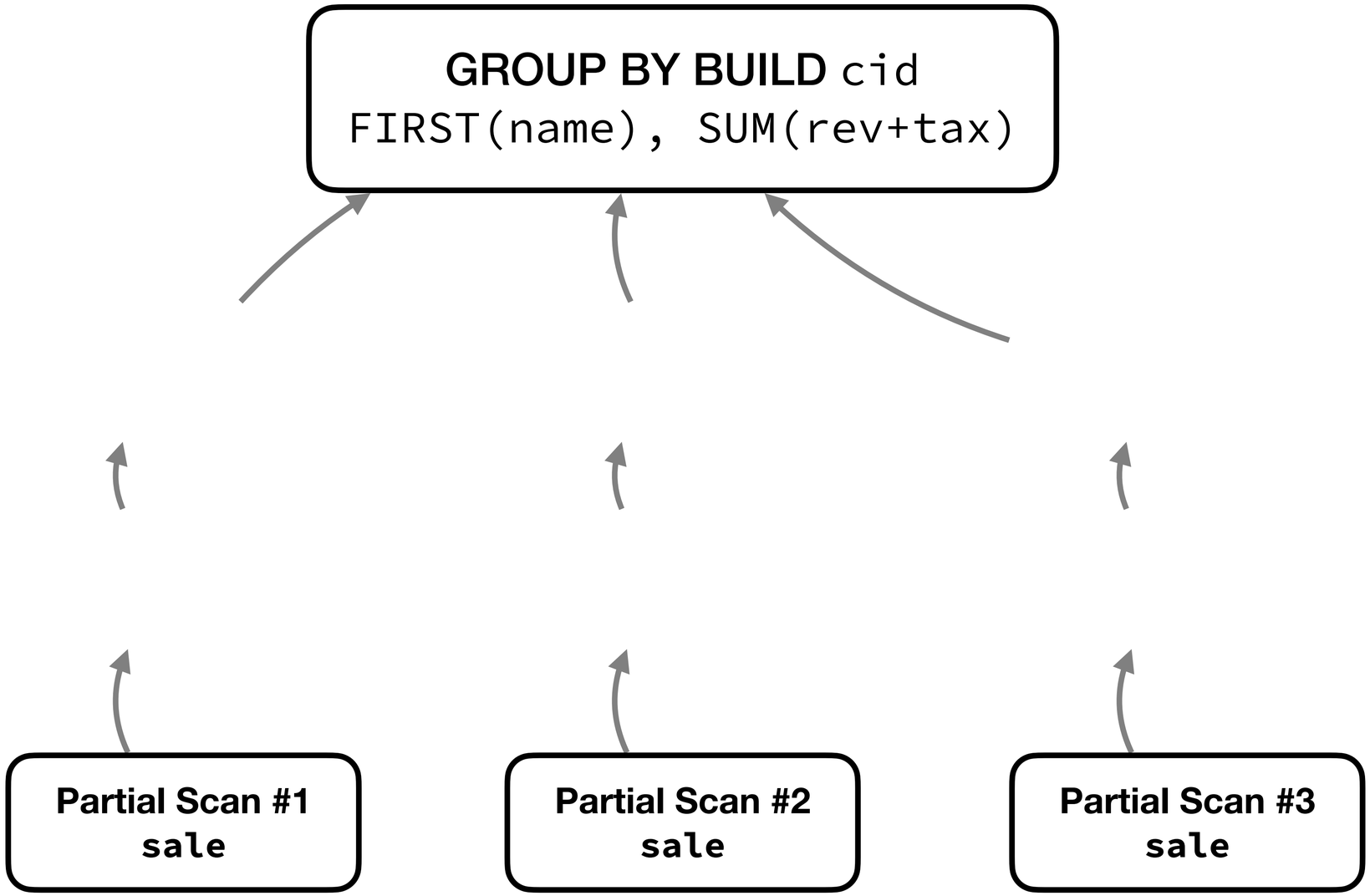

Γ
σ
R
Γ
Xchg(3:1)
Γ
σ
R1
Γ
σ
R2
Γ
σ
R3

Morsel-Driven Parallelism: A NUMA-Aware Query Evaluation Framework for the Many-Core Age

Viktor Leis* Peter Boncz† Alfons Kemper* Thomas Neumann*

2014

```
SELECT FIRST(name), SUM(rev+tax)
FROM cust JOIN sale USING(cid) GROUP BY cid
```



GROUP BY BUILD `cid`

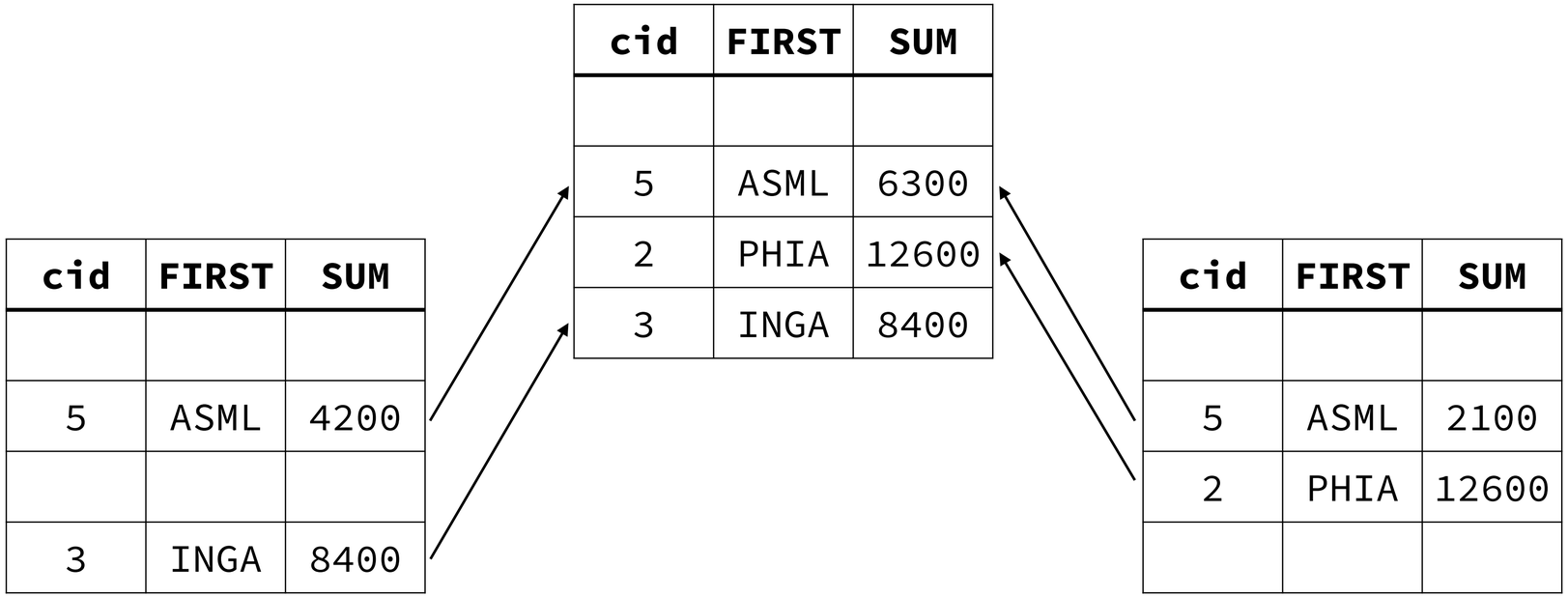
`FIRST(name), SUM(rev+tax)`

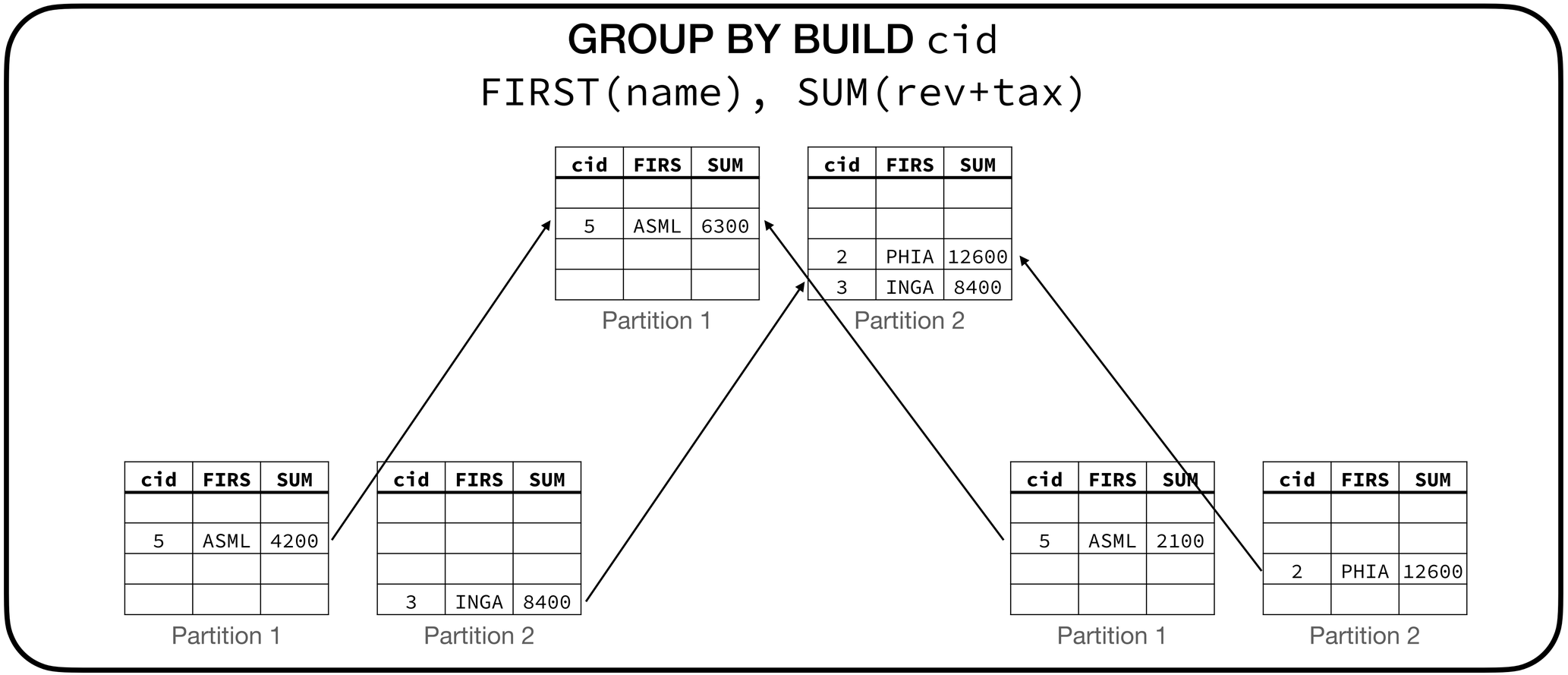
cid	FIRST	SUM
5	ASML	4200
3	INGA	8400

cid	FIRST	SUM
5	ASML	2100
2	PHIA	12600

GROUP BY BUILD `cid`

`FIRST(name), SUM(rev+tax)`




GROUP BY BUILD cid
FIRST(name), SUM(rev+tax)
cid FIRS SUM
5 ASML 6300
Partition 1
cid FIRS SUM
2 PHIA 12600
3 INGA 8400
Partition 2
cid FIRS SUM
5 ASML 4200
Partition 1
cid FIRS SUM
3 INGA 8400
Partition 2
cid FIRS SUM
5 ASML 2100
Partition 1
cid FIRS SUM
2 PHIA 12600
Partition 2

ORDER BY

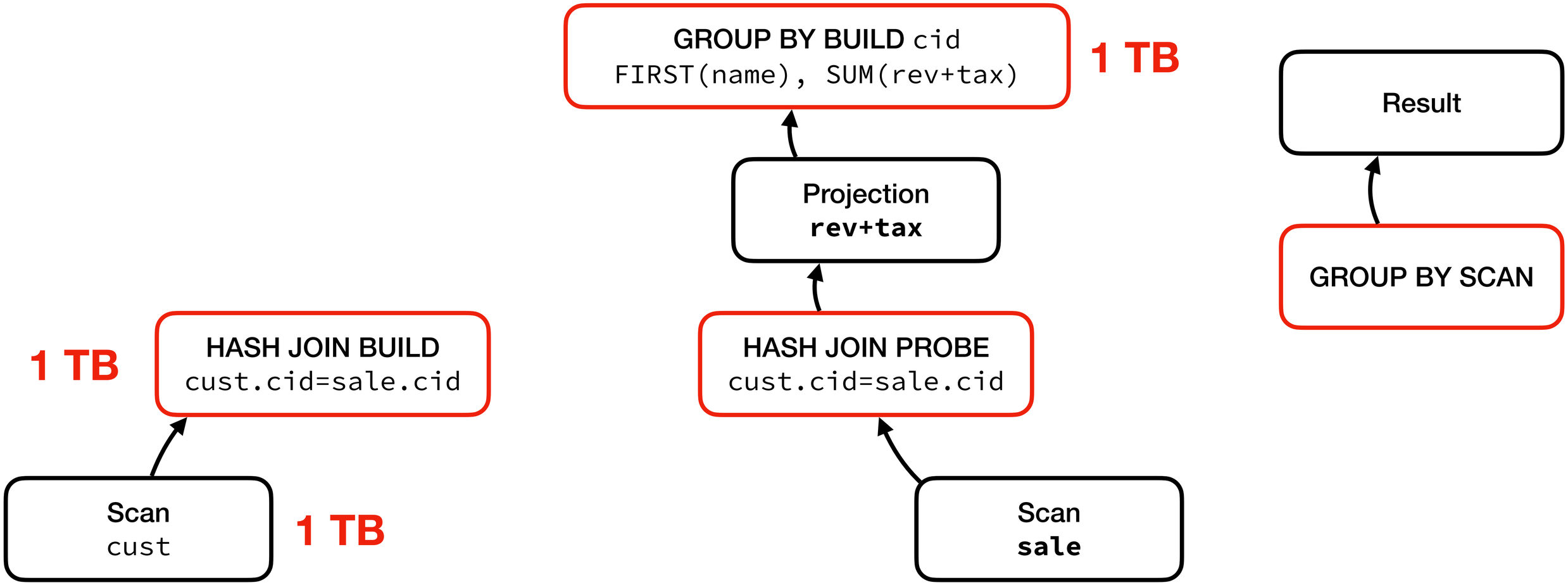
~~GROUP BY~~

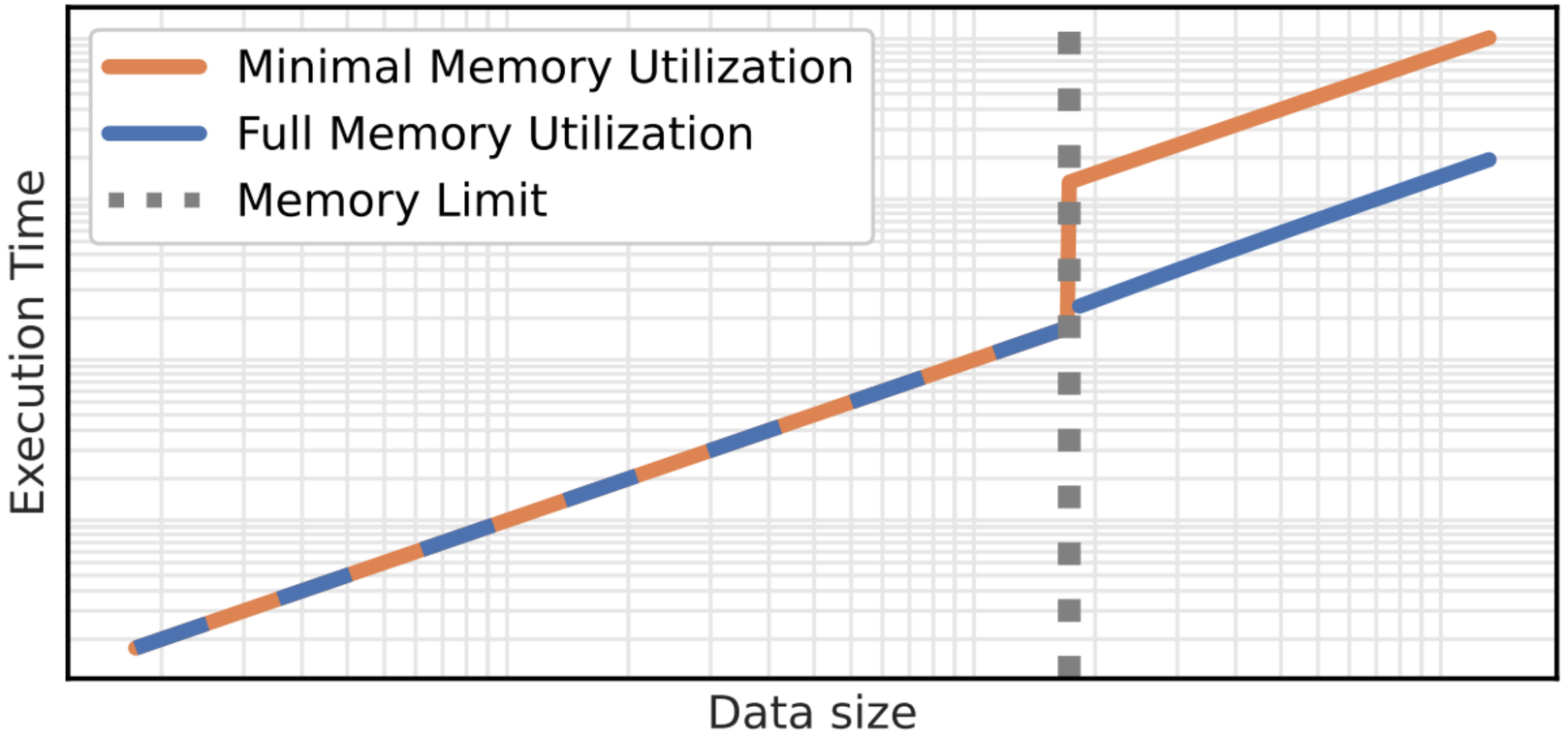
JOIN

OVER

COPY

```
SELECT FIRST(name), SUM(rev+tax)
FROM cust JOIN sale USING(cid) GROUP BY cid
```




Minimal Memory Utilization
Full Memory Utilization
Memory Limit
Execution Time
Data size

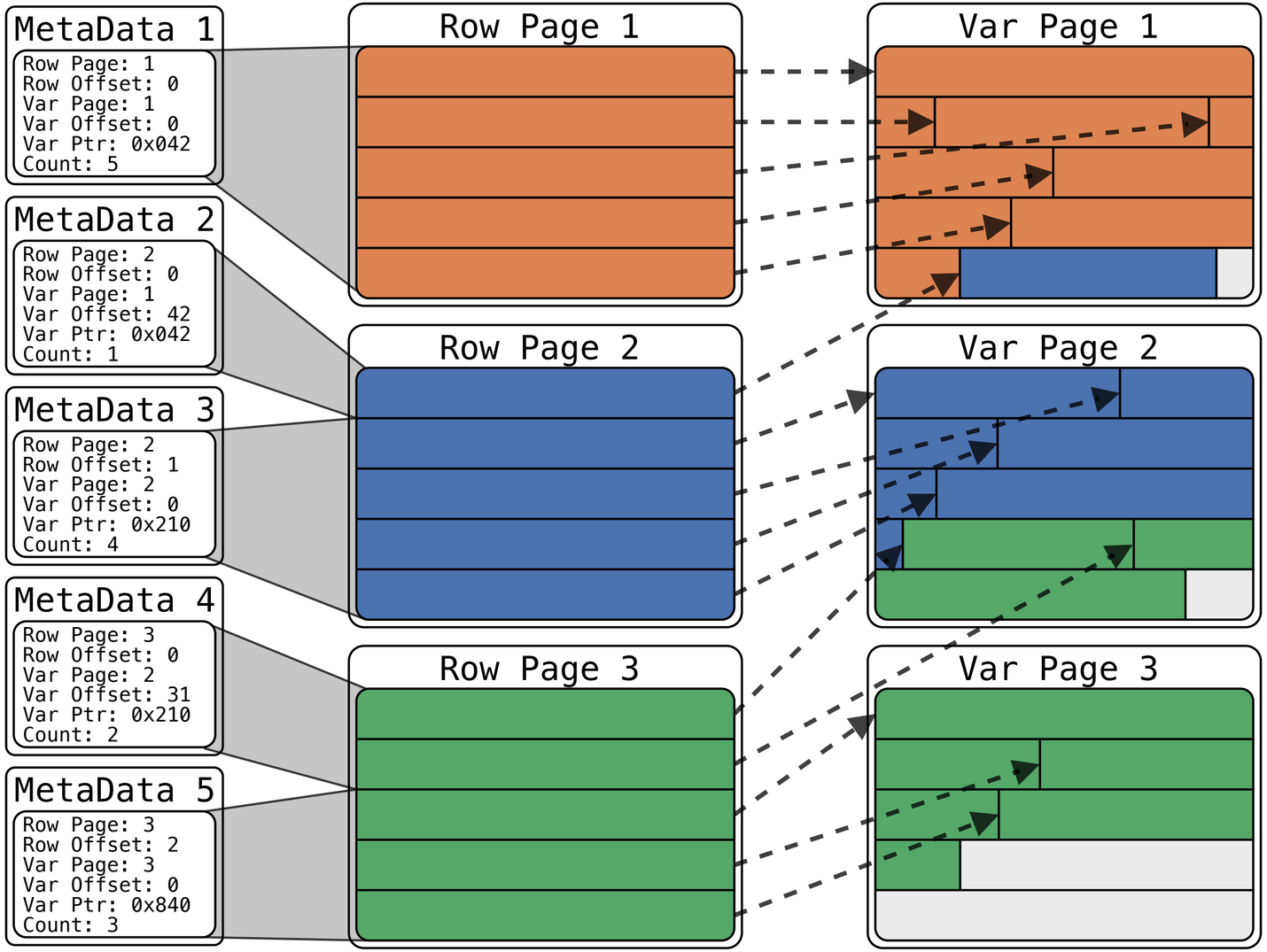
Or crash ^^

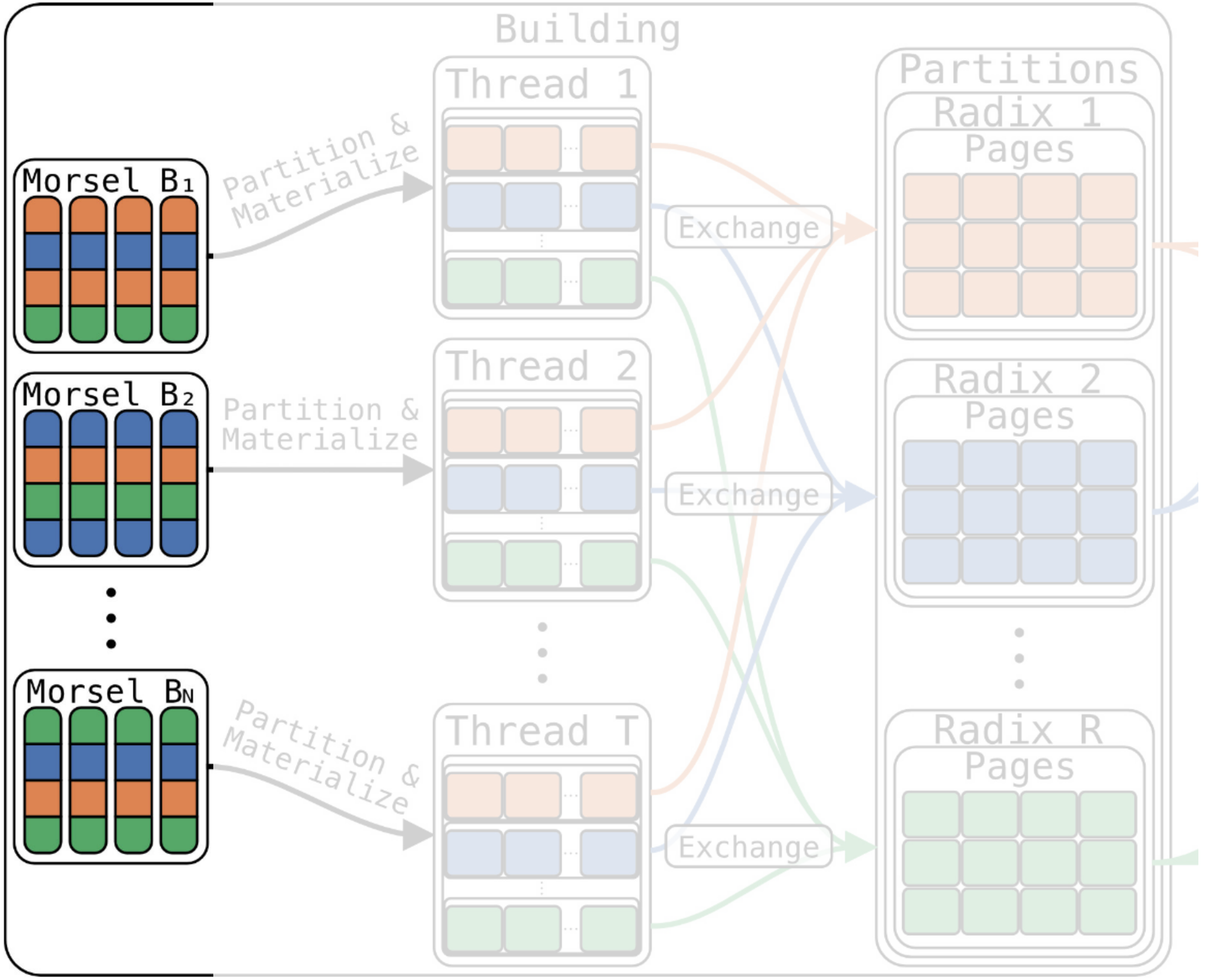
Out-Of-Core

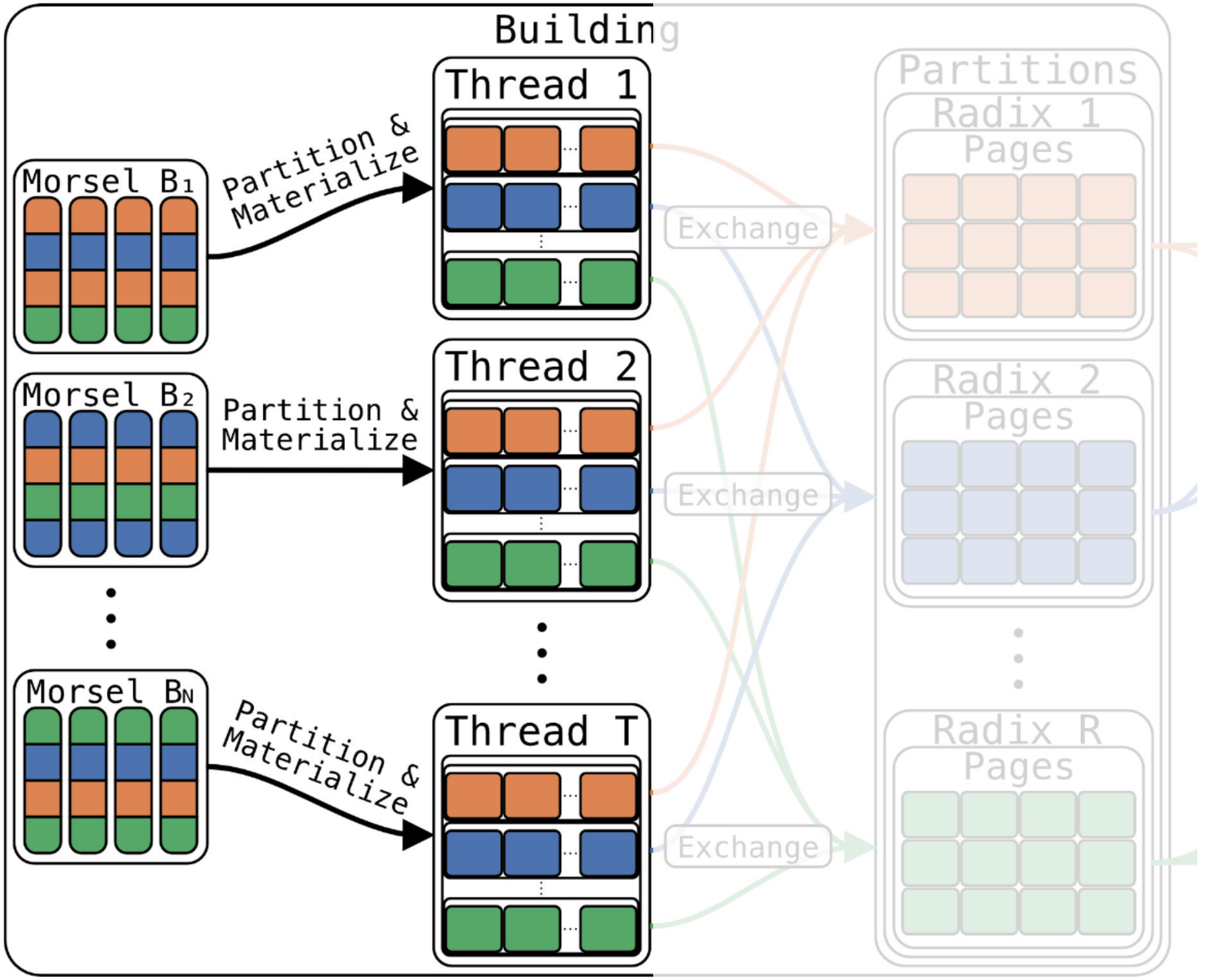
Saving Private Hash Join

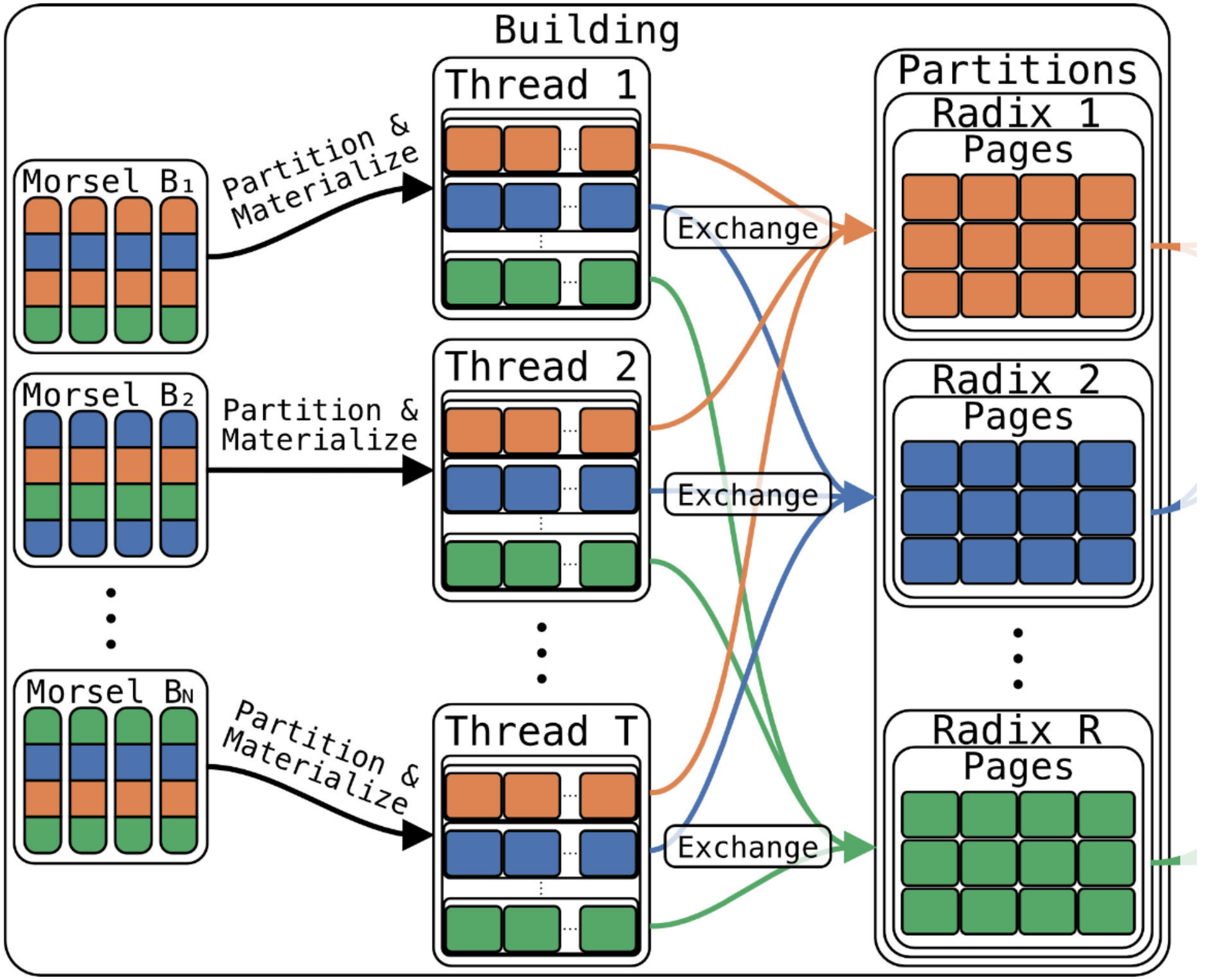
Laurens Kuiper, Paul Groß, Peter Boncz, Hannes Mühleisen
Centrum Wiskunde & Informatica
Amsterdam, The Netherlands
{laurens.kuiper,paul.gross,peter.boncz,hannes.muehleisen}@cwi.nl

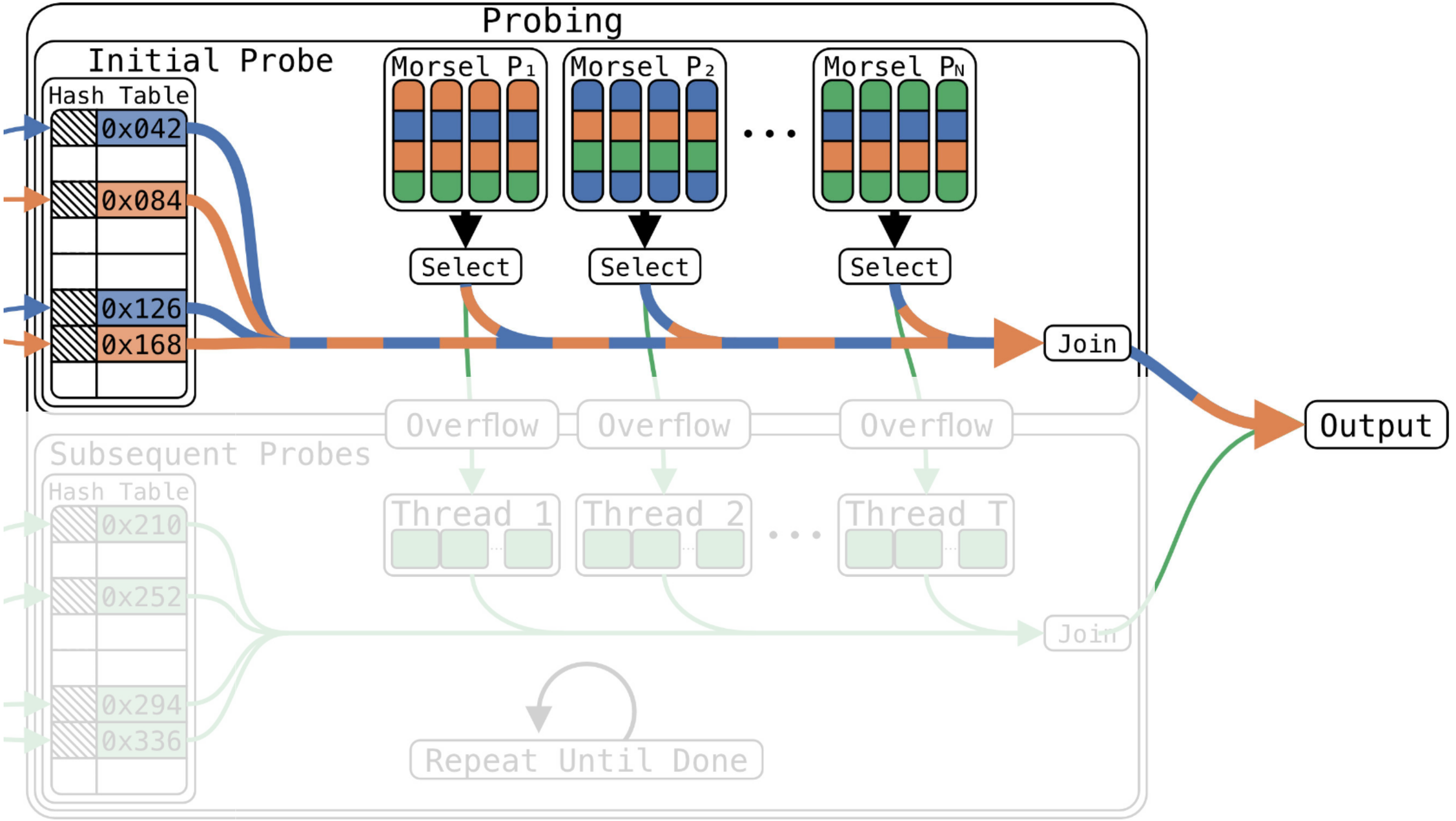
VLDB 2025!

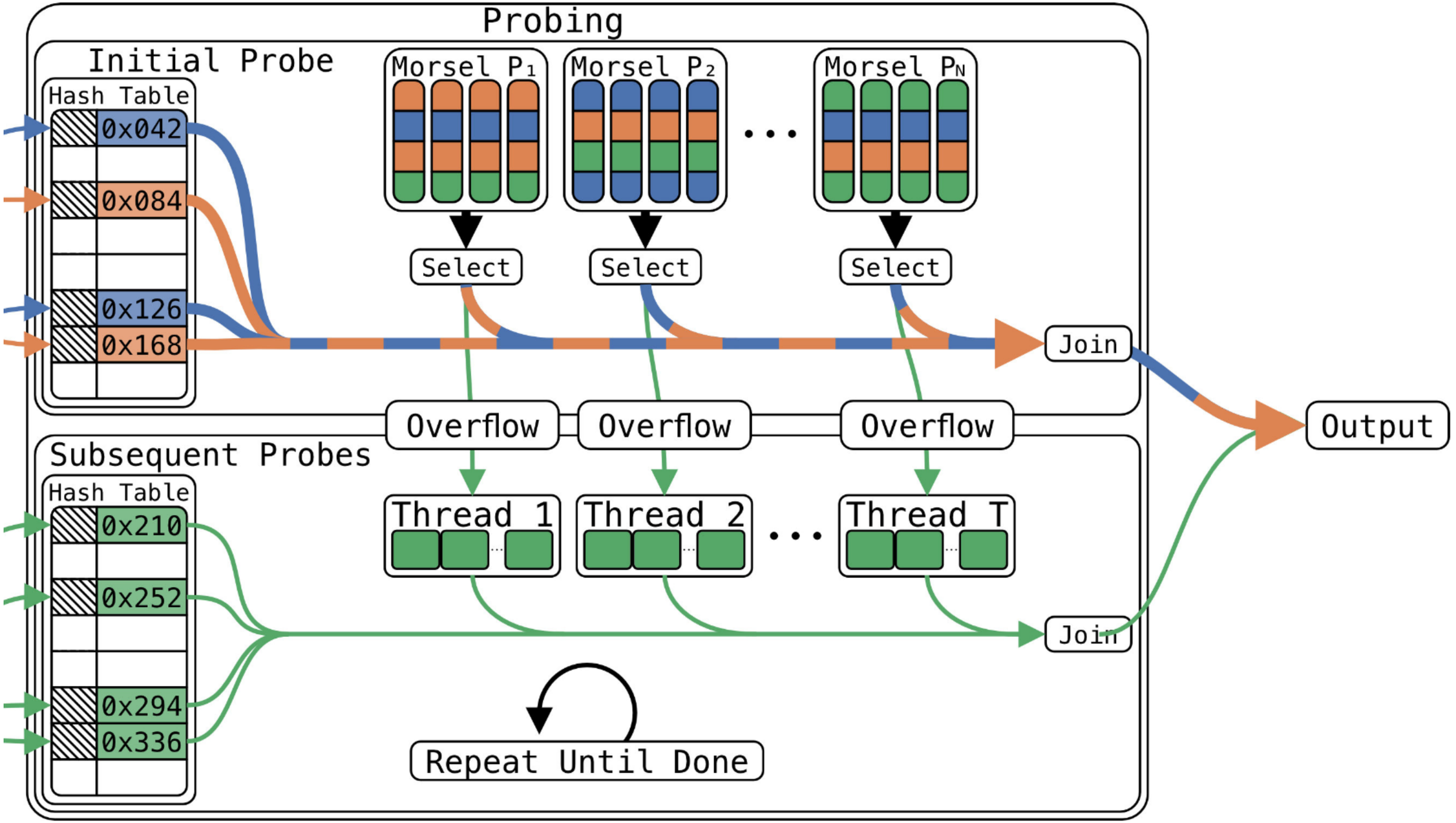

MetaData 1
Row Page: 1
Row Offset: 0
Var Page: 1
Var Offset: 0
Var Ptr: 0x042
Count: 5
MetaData 2
Row Page: 2
Row Offset: 0
Var Page: 1
Var Offset: 42
Var Ptr: 0x042
Count: 1
MetaData 3
Row Page: 2
Row Offset: 1
Var Page: 2
Var Offset: 0
Var Ptr: 0x210
Count: 4
MetaData 4
Row Page: 3
Row Offset: 0
Var Page: 2
Var Offset: 31
Var Ptr: 0x210
Count: 2
MetaData 5
Row Page: 3
Row Offset: 2
Var Page: 3
Var Offset: 0
Var Ptr: 0x840
Count: 3
Row Page 1
Row Page 2
Row Page 3
Var Page 1
Var Page 2
Var Page 3

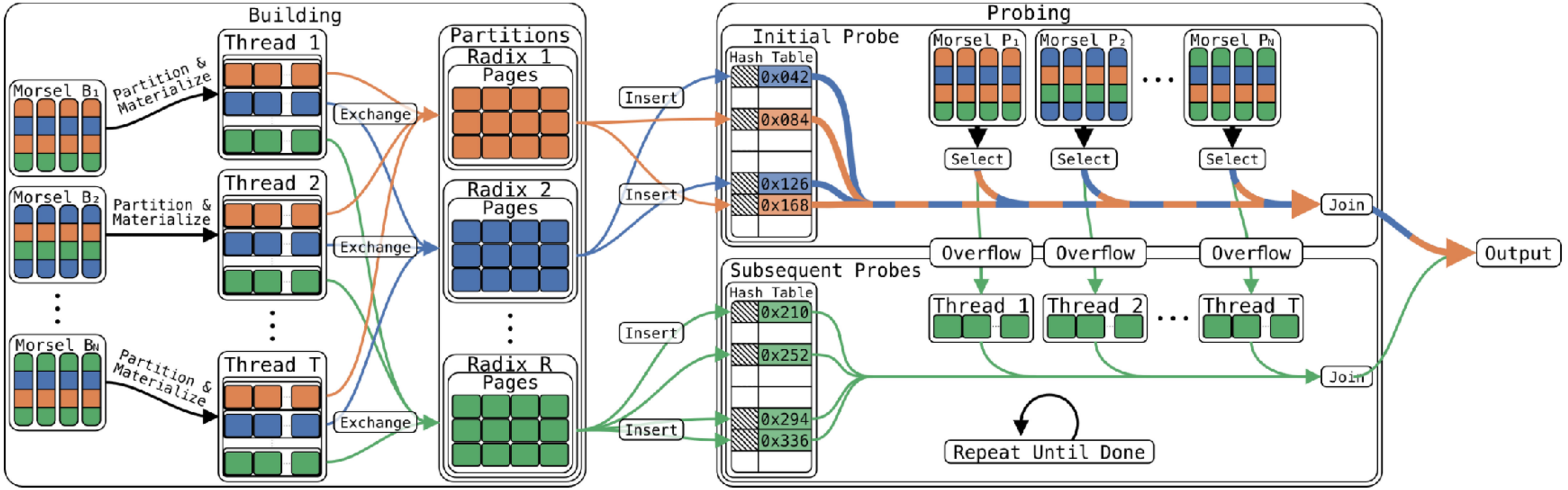

Building
Morsel B1
Morsel B2
Morsel BN
Partition & Materialize
Partition & Materialize
Partition & Materialize
Thread 1
Thread 2
Thread T
Exchange
Exchange
Exchange
Partitions
Radix 1
Pages
Radix 2
Pages
Radix R
Pages

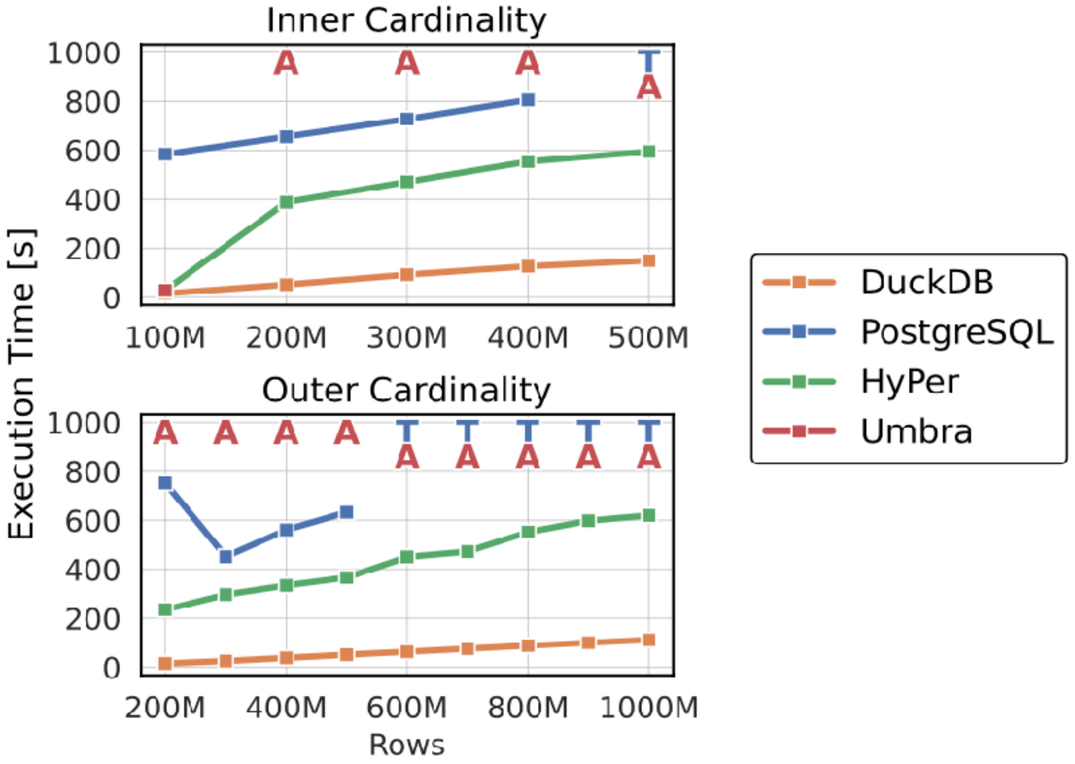

Building
Thread 1
Thread 2
Thread T
Morsel B1
Morsel B2
Morsel BN
Partition & Materialize
Partition & Materialize
Partition & Materialize
Exchange
Exchange
Exchange
Partitions
Radix 1
Pages
Radix 2
Pages
Radix R
Pages

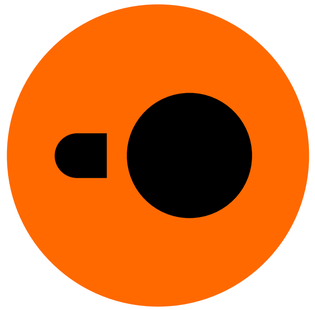

Building
Thread 1
Thread 2
Thread T
Morsel B1
Morsel B2
Morsel BN
Partition & Materialize
Partition & Materialize
Partition & Materialize
Exchange
Exchange
Exchange
Partitions
Radix 1
Pages
Radix 2
Pages
Radix R
Pages


Probing
Initial Probe
Hash Table
0x042
0x084
0x126
0x168
Morsel P1
Morsel P2
Morsel PN
Select
Select
Select
Join
Output
Overflow
Overflow
Overflow
Subsequent Probes
Hash Table
0x210
0x252
0x294
0x336
Thread 1
Thread 2
Thread T
Join
Repeat Until Done


Probing
Initial Probe
Hash Table
0x042
0x084
0x126
0x168
Morsel P1
Morsel P2
Morsel PN
Select
Select
Select
Join
Output
Overflow
Overflow
Overflow
Subsequent Probes
Hash Table
0x210
0x252
0x294
0x336
Thread 1
Thread 2
Thread T
Join
Repeat Until Done


Building
Morsel B1
Morsel B2
Morsel BN
Partition & Materialize
Partition & Materialize
Partition & Materialize
Thread 1
Thread 2
Thread T
Exchange
Exchange
Exchange
Partitions
Radix 1 Pages
Radix 2 Pages
Radix R Pages
Insert
Insert
Insert
Insert
Probing
Initial Probe
Hash Table
0x042
0x084
0x126
0x168
Morsel P1
Morsel P2
Morsel PN
Select
Select
Select
Join
Overflow
Overflow
Overflow
Subsequent Probes
Hash Table
0x210
0x252
0x294
0x336
Thread 1
Thread 2
Thread T
Join
Repeat Until Done
Output


Inner Cardinality
Outer Cardinality
Execution Time [s]
1000
800
600
400
200
0
100M
200M
300M
400M
500M
200M
400M
600M
800M
1000M
Rows
A
T
DuckDB
PostgreSQL
HyPer
Umbra



@duckdb.org
@hannes.muehleisen.org