Liberate Analytical Data Management with DuckDB

Hannes Mühleisen

DuckDB Labs


D
S QL 1337

Act 1: The Backstory

2015


MARK RONSON
FEAT: BRUNO MARS
UPTOWN FUNK!
TRINIDAD JAMES REMIX


STAR WARS
THE FORCE AWAKENS
DECEMBER 18
IN 3D, REAL D 3D AND IMAX 3D

teradata.
VERTICA
Greenplum

ENTERPRISE SPECIAL

HIVE
APACHE Spark

HADOOP-ISTAN

Google Big Query
amazon REDSHIFT

CLOUD TODDLERS

ORACLE DATABASE
Microsoft SQL Server
IBM DB2

DINOSAUR ADD-ONS

2015 Analytics

vectorwise
HyPer

OBSCURE ACADEMIC SPIN-OFFS

pandas
GNUparallel
data.table

DESPAIR ENGINEERING

Can’t
pip import state_of_the_art

have to build
this ourselves?

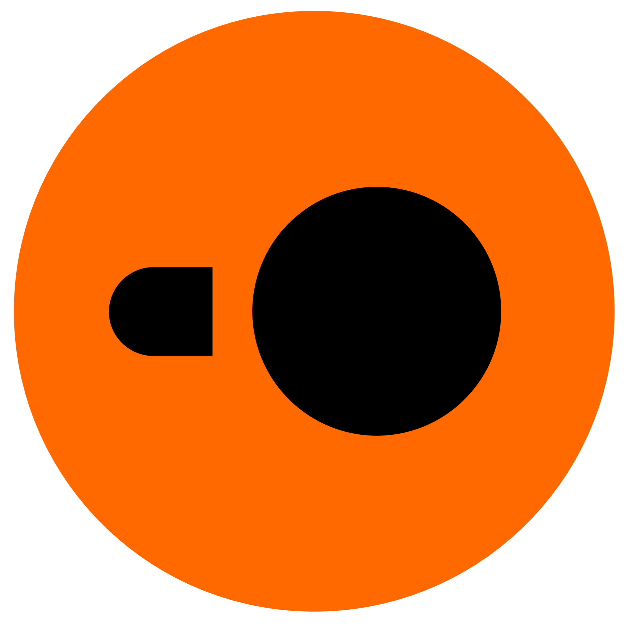
100 people, 10 years
Many $$$

Pause

Spite Engineering

SQLite

SQLite for Analytics!



Act 2: Design Decisions

Distributed?

Scalability! But at what COST?

Frank McSherry Unaffiliated
Michael Isard Unaffiliated[*]
Derek G. Murray Unaffiliated[†]

Abstract

We offer a new metric for big data platforms, COST, or the Configuration that Outperforms a Single Thread. The COST of a given platform for a given problem is the hardware configuration required before the platform outperforms a competent single-threaded implementation. COST weighs a system's scalability against the overheads introduced by the system, and indicates the actual performance gains of the system, without rewarding systems that bring substantial but parallelizable overheads.

We survey measurements of data-parallel systems recently reported in SOSP and OSDI, and find that many systems have either a surprisingly large COST, often

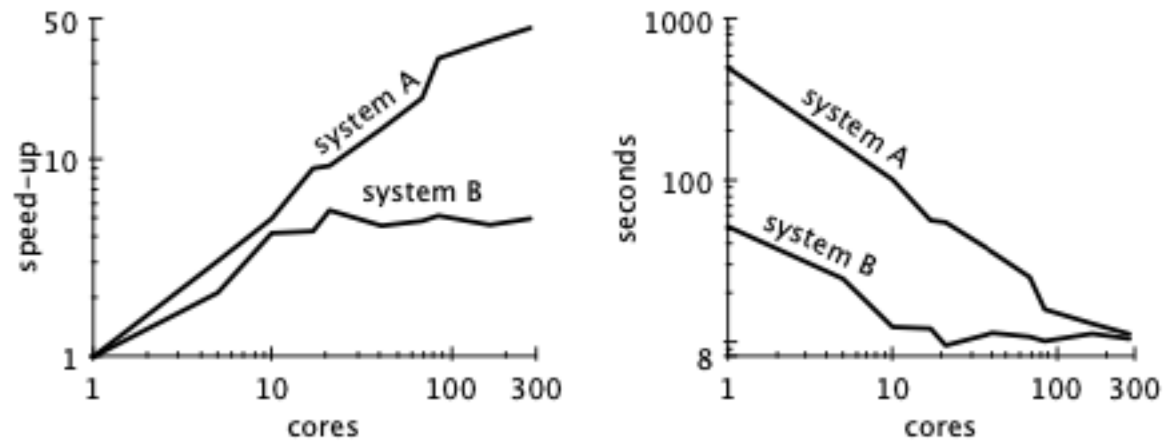
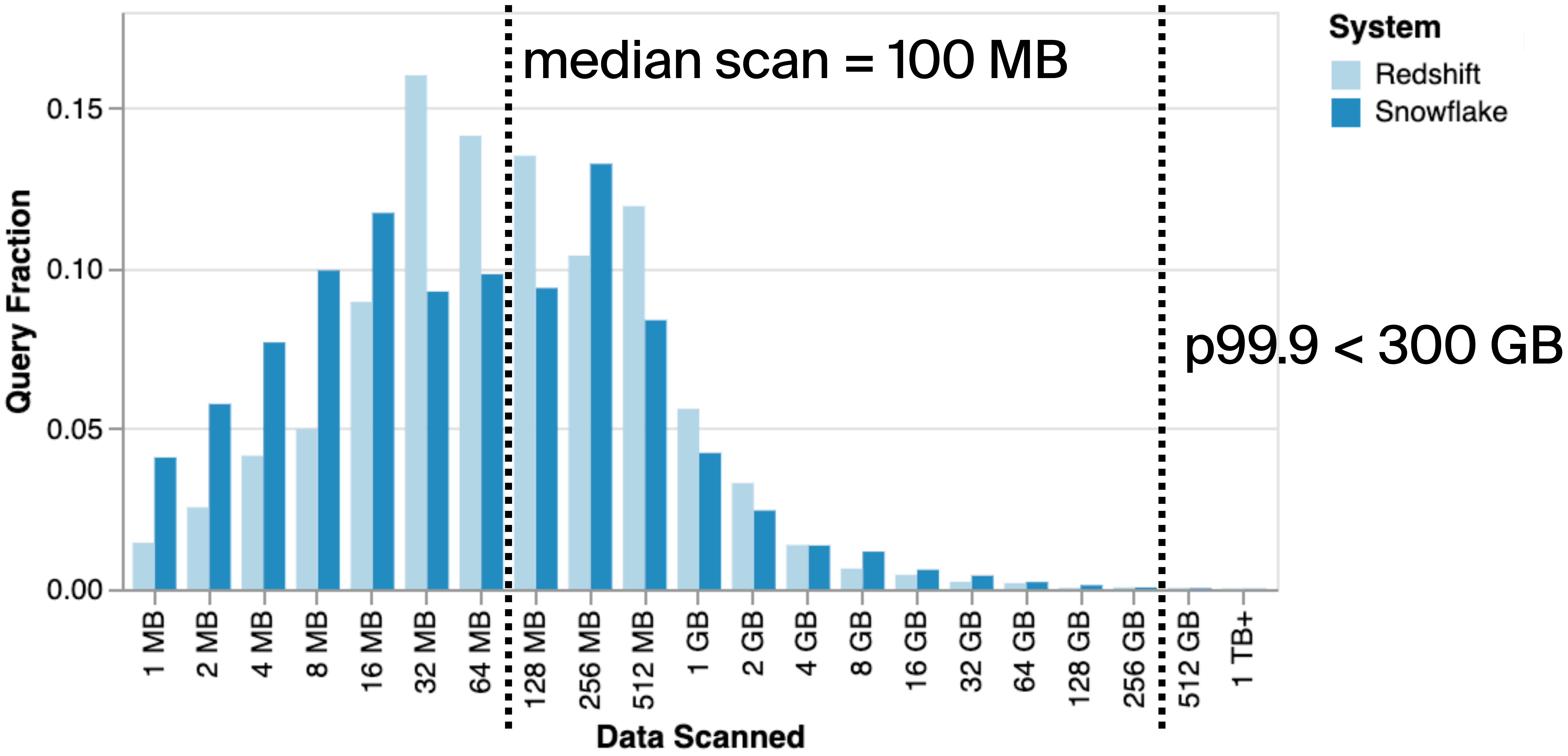


Figure 1: Scaling and performance measurements for a data-parallel algorithm, before (system A) and after (system B) a simple performance optimization. The unoptimized implementation "scales" far better, despite (or rather, because of) its poor performance.

2015!

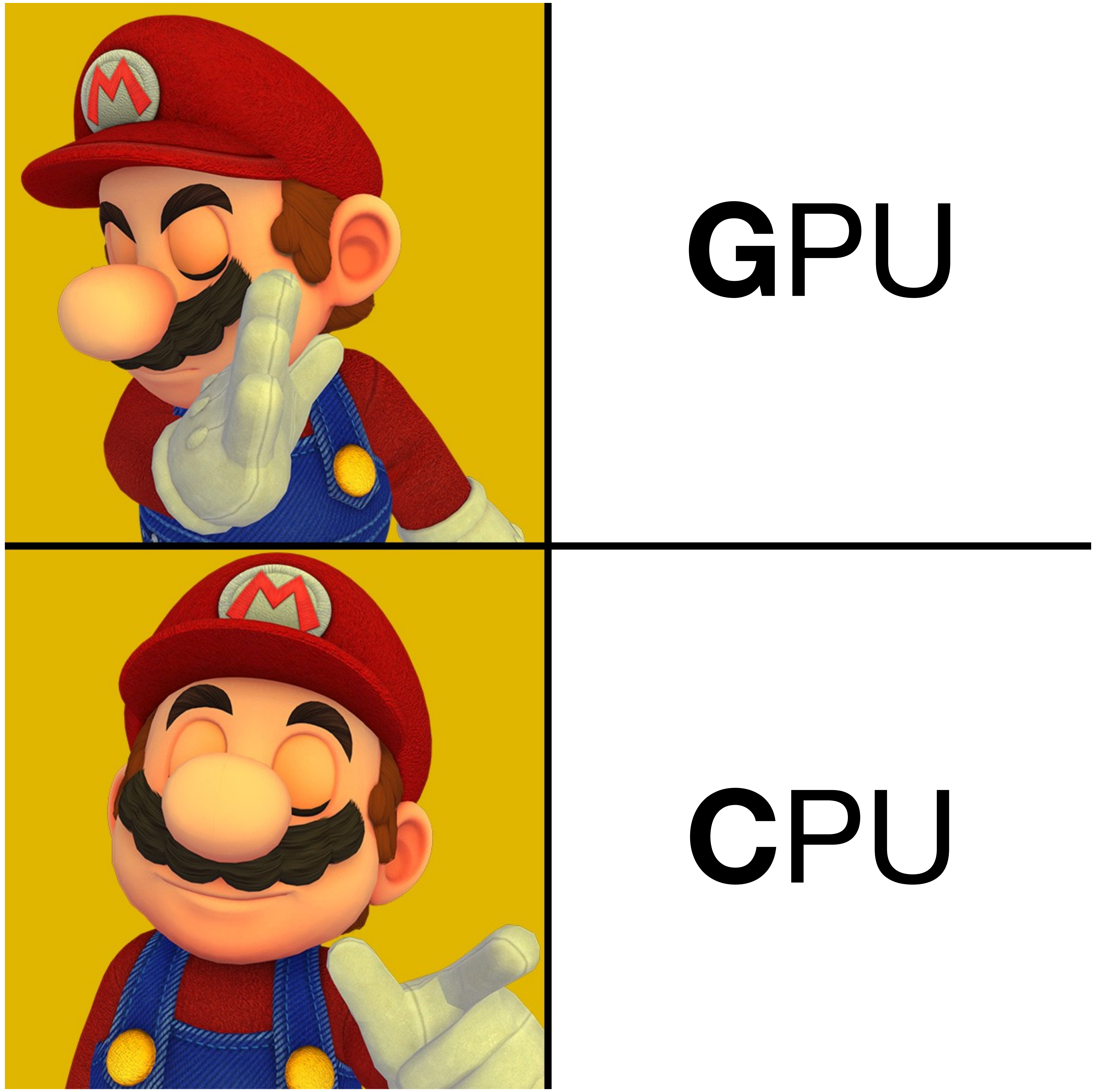


median scan = 100 MB
p99.9 < 300 GB
System
Redshift
Snowflake
Query Fraction
0.15
0.10
0.05
0.00
1 MB
2 MB
4 MB
8 MB
16 MB
32 MB
64 MB
128 MB
256 MB
512 MB
1 GB
2 GB
4 GB
8 GB
16 GB
32 GB
64 GB
128 GB
256 GB
512 GB
1 TB+
Data Scanned

Single Node!

JIT

Vectorized


GPU
CPU

SIMD

Scalar

Proprietary

MIT

DuckDB

Fair Benchmarking Considered Difficult: Common Pitfalls In Database Performance Testing

Mark Raasveldt, Pedro Holanda, Tim Gubner & Hannes Mühleisen
Centrum Wiskunde & Informatica (CWI)
Amsterdam, The Netherlands
[raasveld,holanda,tgubner,hannes]@cwi.nl

ABSTRACT

Performance benchmarking is one of the most commonly used methods for comparing different systems or algorithms, both in scientific literature and in industrial publications. While performance measurements might seem objective on the surface, there are many different ways to influence benchmark results to favor one system over the other, either by accident or on purpose. In this paper, we perform a study of the common pitfalls in DBMS performance comparisons, and give advice on how they can be spotted and avoided so a fair performance comparison between systems can be made. We illustrate the common pitfalls with a series of mock benchmarks, which show large differences in performance where none should be present.

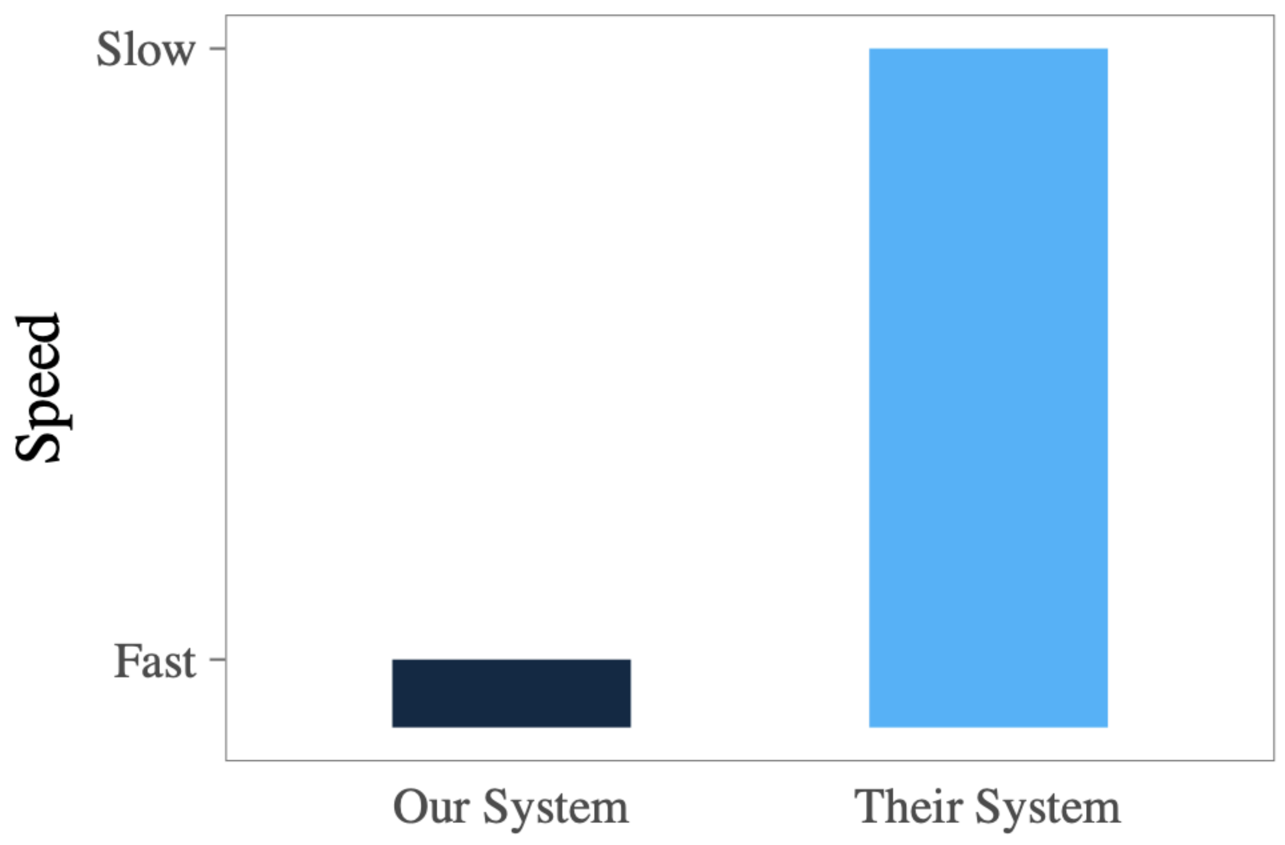


Figure 1: Generic benchmark results.

2018

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

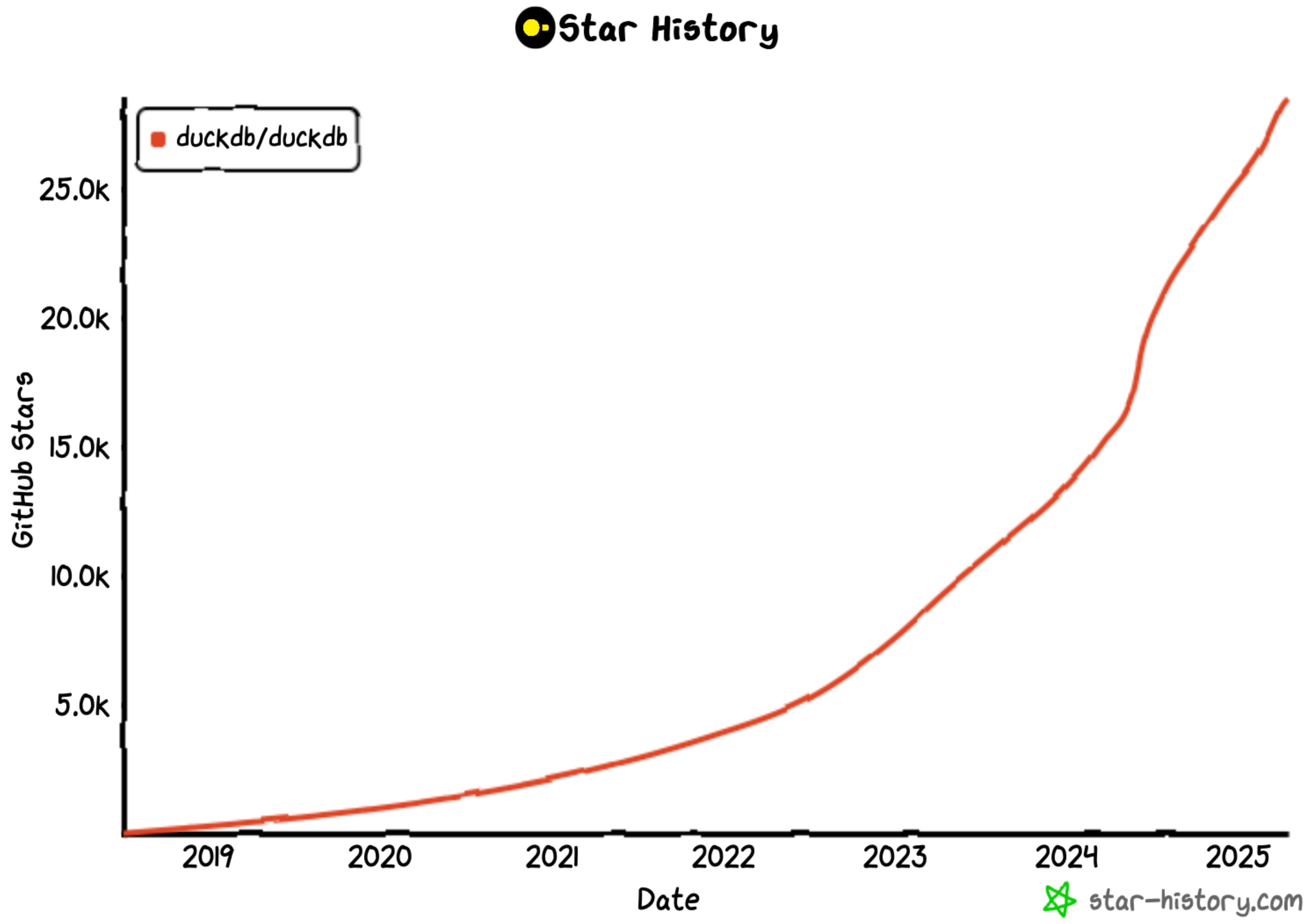
System and Machine

Relative time (lower is better)

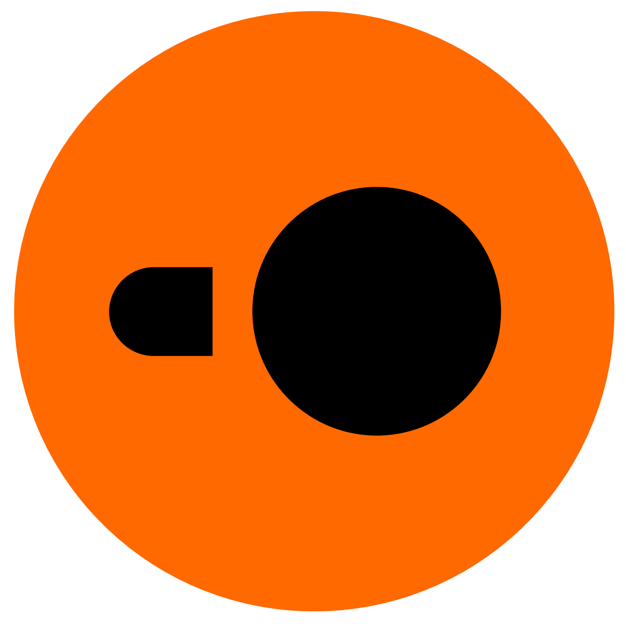
DuckDB (c6a.4xlarge, 500gb gp2) ×1.15

DuckDB (m5.4xlarge, 500gb gp2) ×1.51

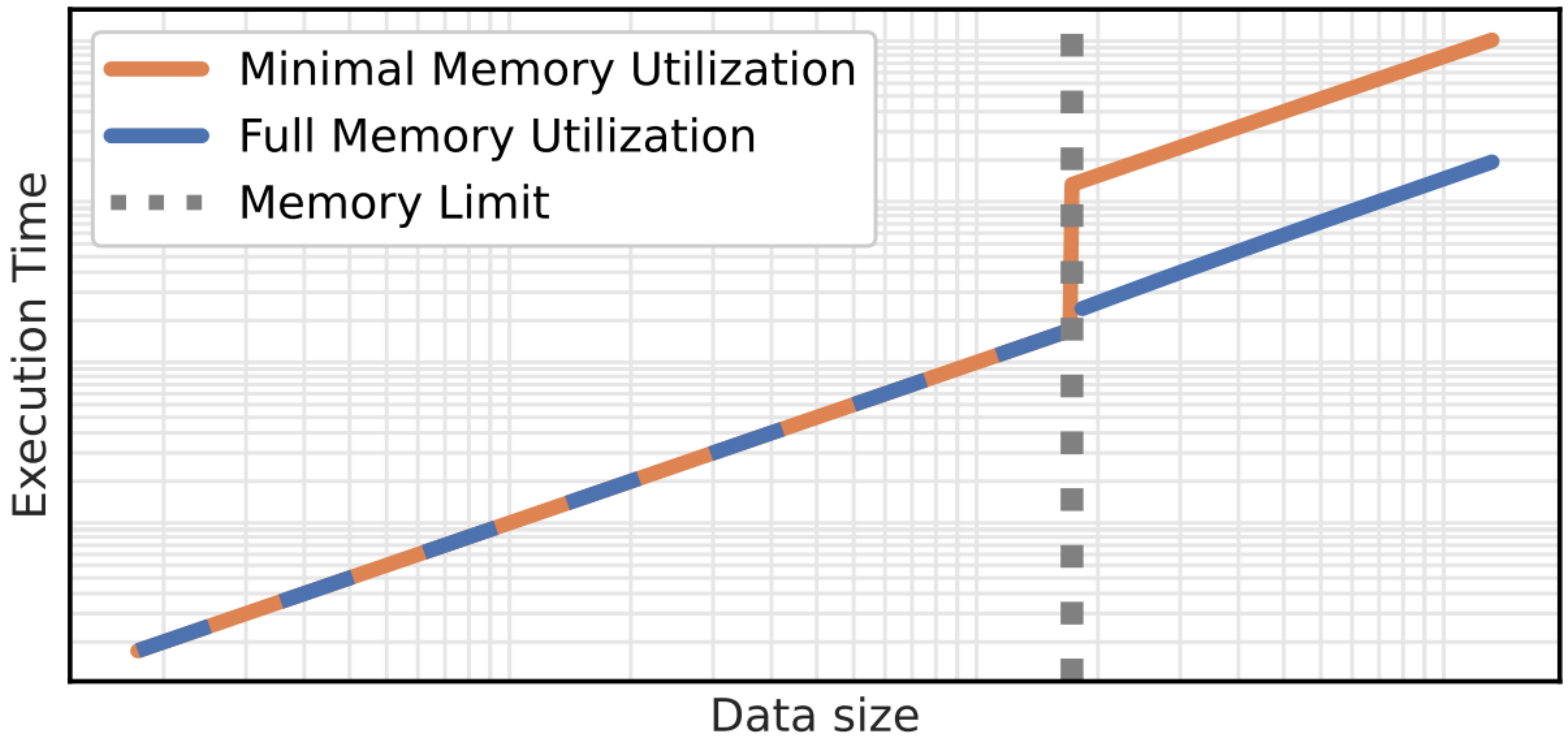
- **DuckDB isn't built for real-time analytics, so it's excluded from the main results, but it was the fastest in the benchmark.** Given its popularity, we included it in the benchmark to serve as a point of reference, and it surprised us: It was 3.5x faster than TimescaleDB and 7.3x faster than ClickHouse.


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

DuckDB Labs



Act 3: Going Deep

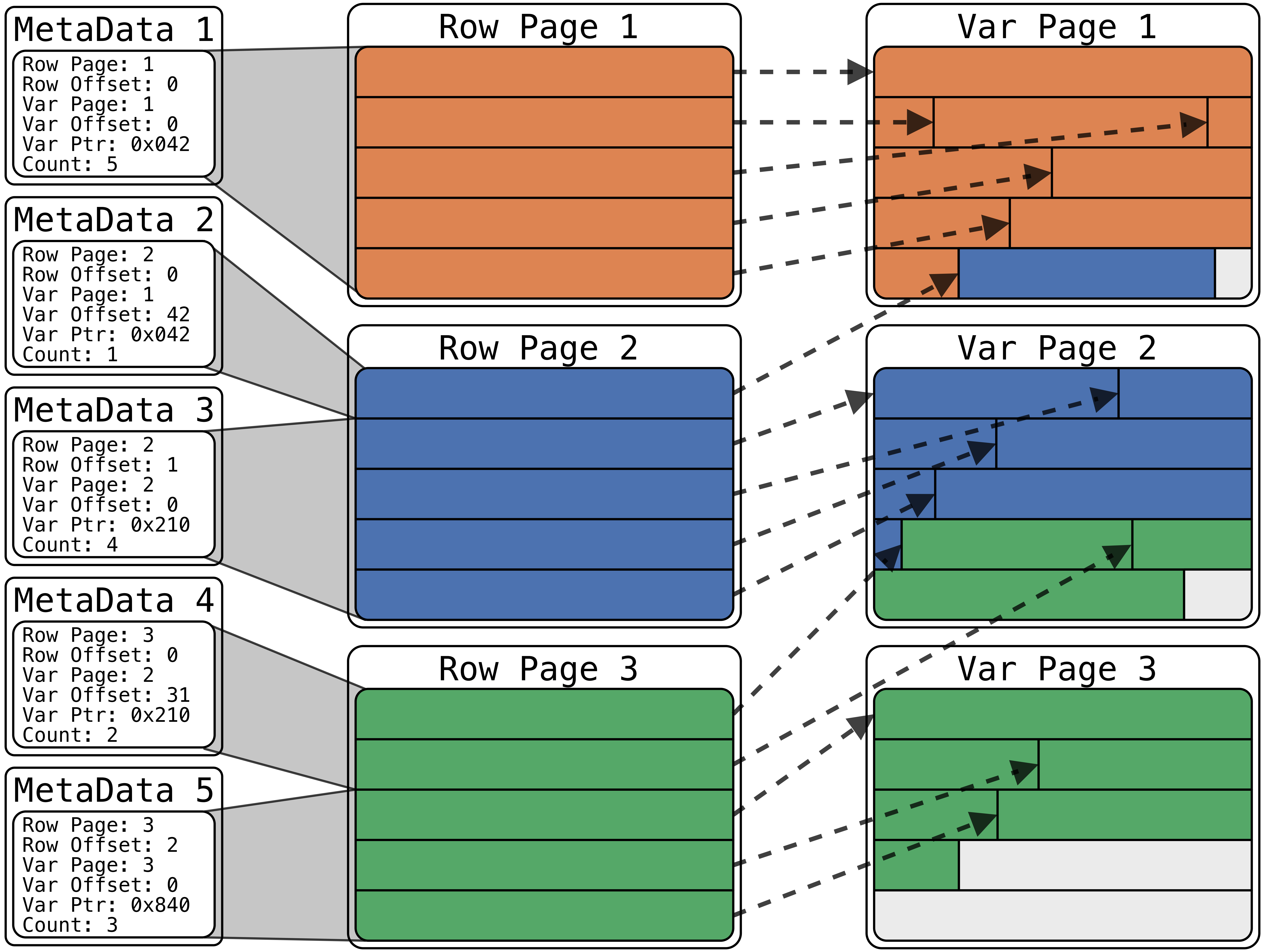

Minimal Memory Utilization
Full Memory Utilization
Memory Limit
Execution Time
Data size

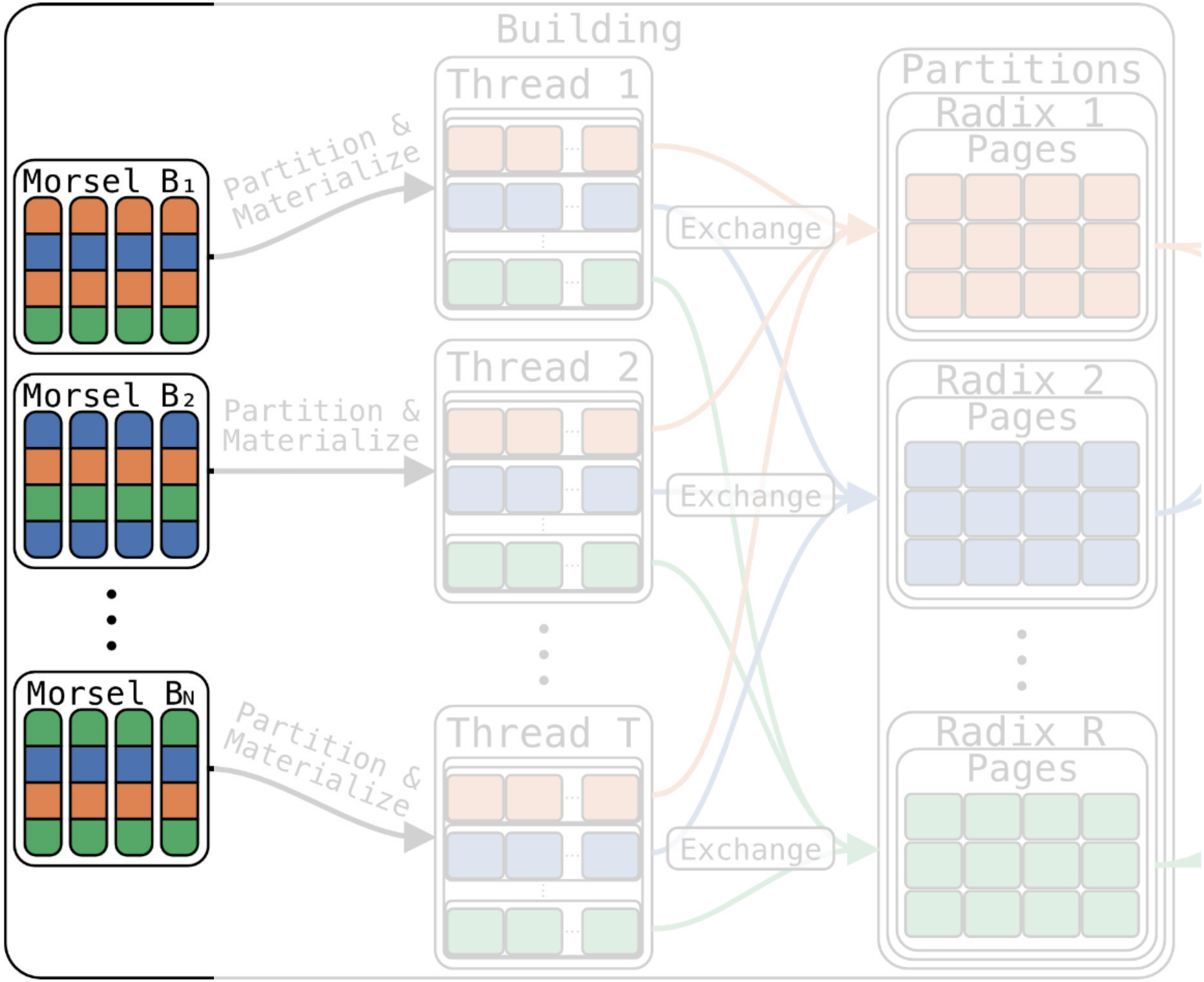
Or crash ^^

Saving Private Hash Join

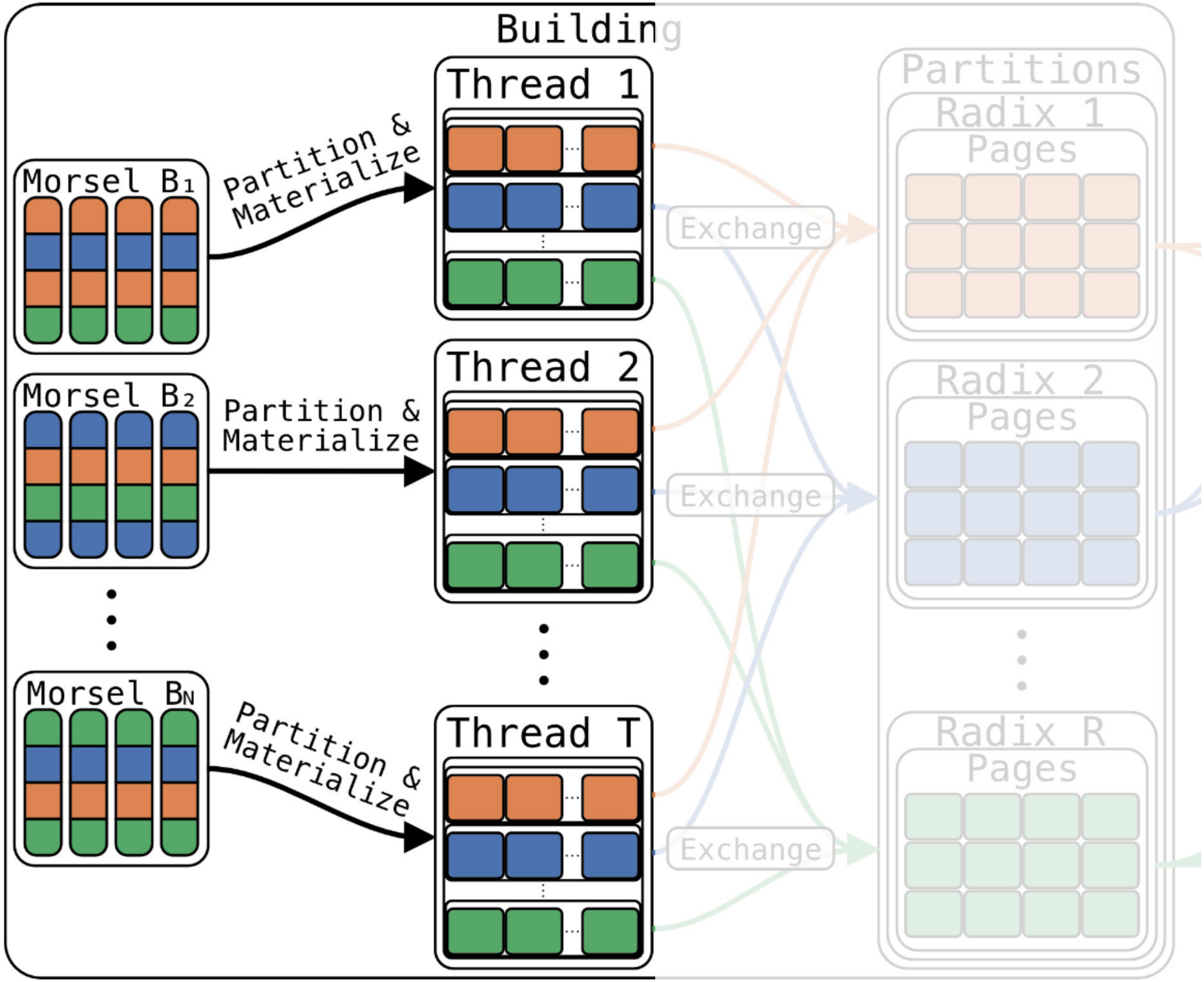
Laurens Kuiper, Paul Groß, Peter Boncz, Hannes Mühleisen
Centrum Wiskunde & Informatica
Amsterdam, The Netherlands
{laurens.kuiper,paul.gross,peter.boncz,hannes.muehleisen}@cwi.nl

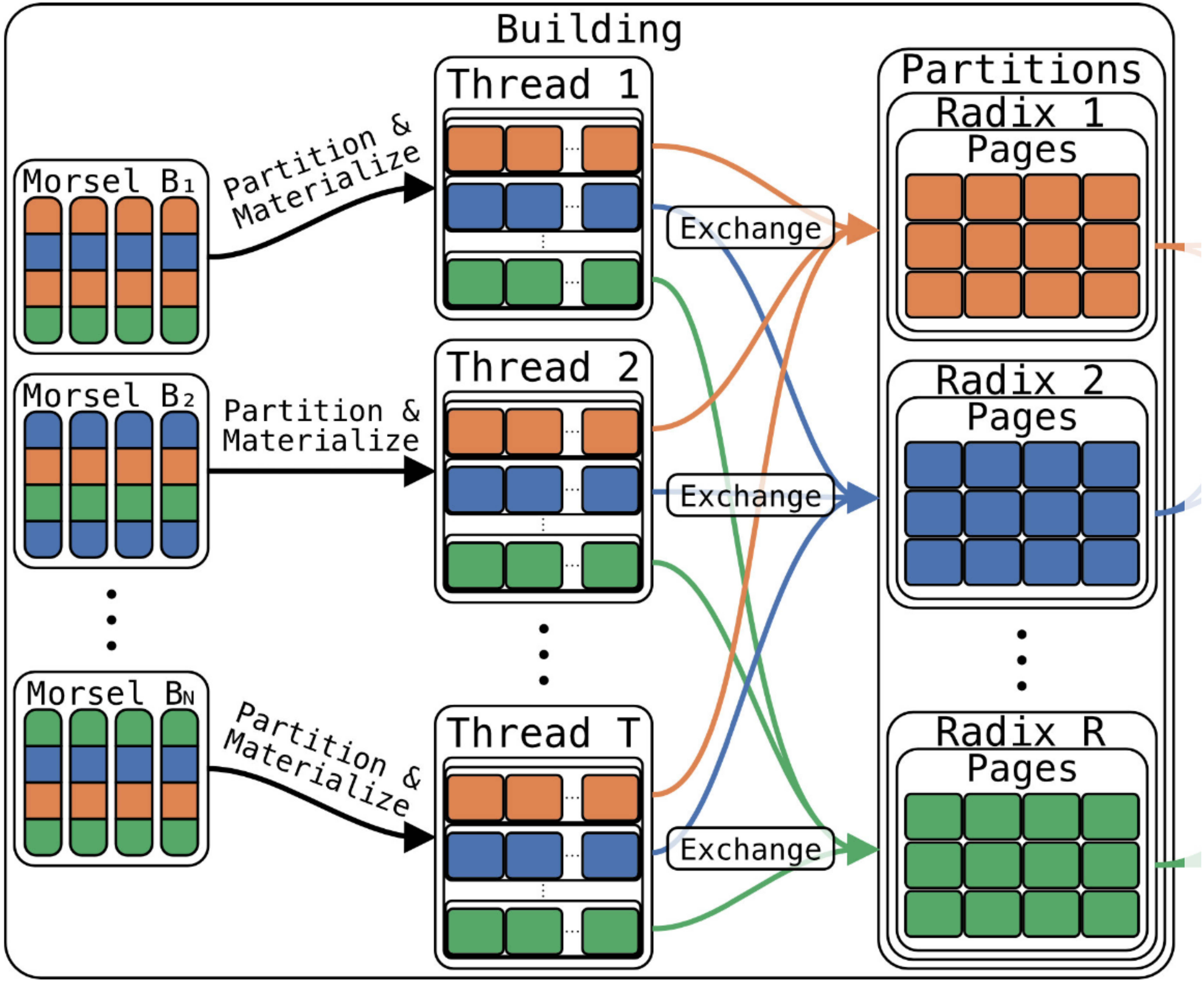
VLDB 2025!

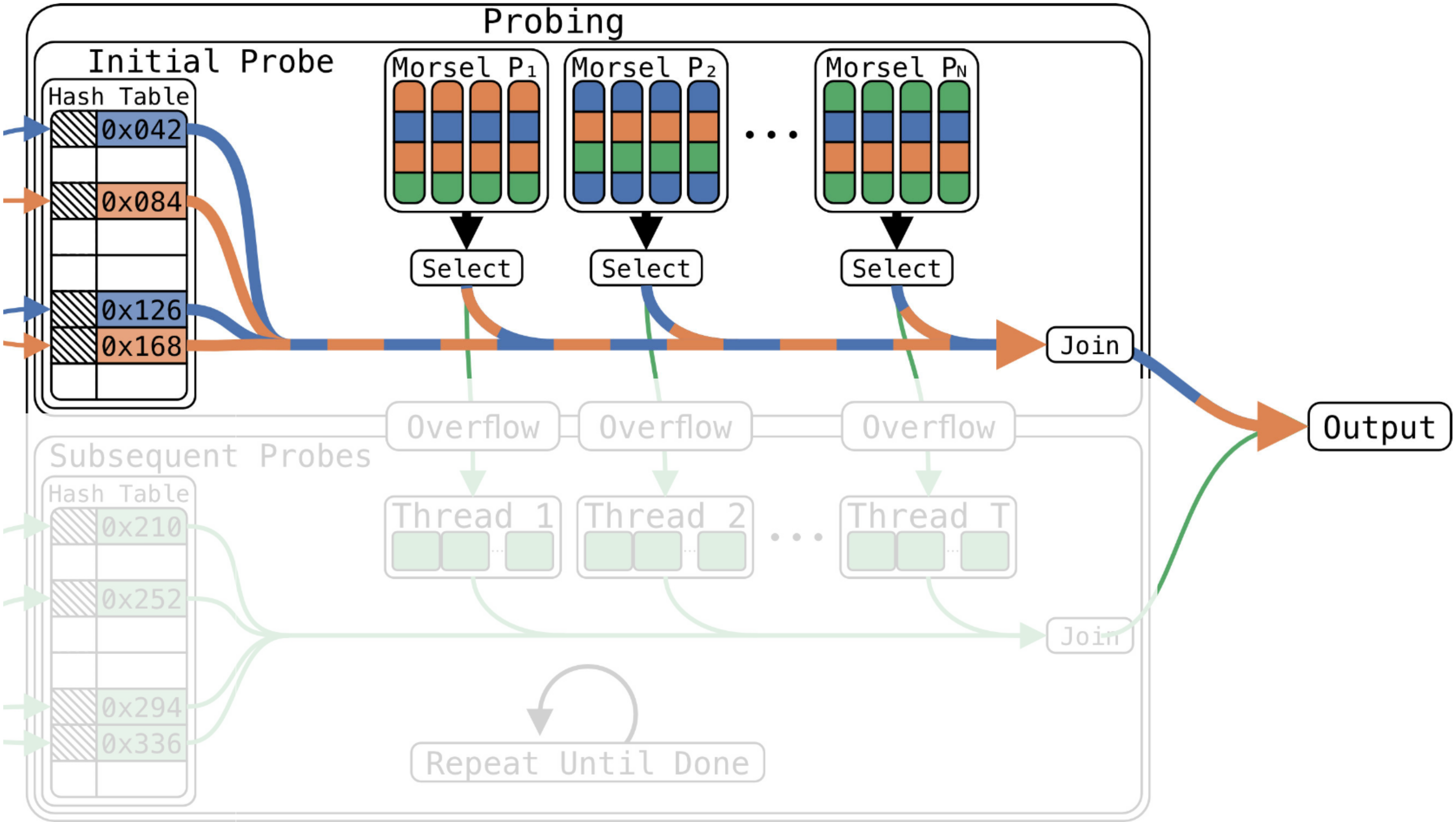

MetaData 1
Row Page: 1
Row Offset: 0
Var Page: 1
Var Offset: 0
Var Ptr: 0x042
Count: 5
MetaData 2
Row Page: 2
Row Offset: 0
Var Page: 1
Var Offset: 42
Var Ptr: 0x042
Count: 1
MetaData 3
Row Page: 2
Row Offset: 1
Var Page: 2
Var Offset: 0
Var Ptr: 0x210
Count: 4
MetaData 4
Row Page: 3
Row Offset: 0
Var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

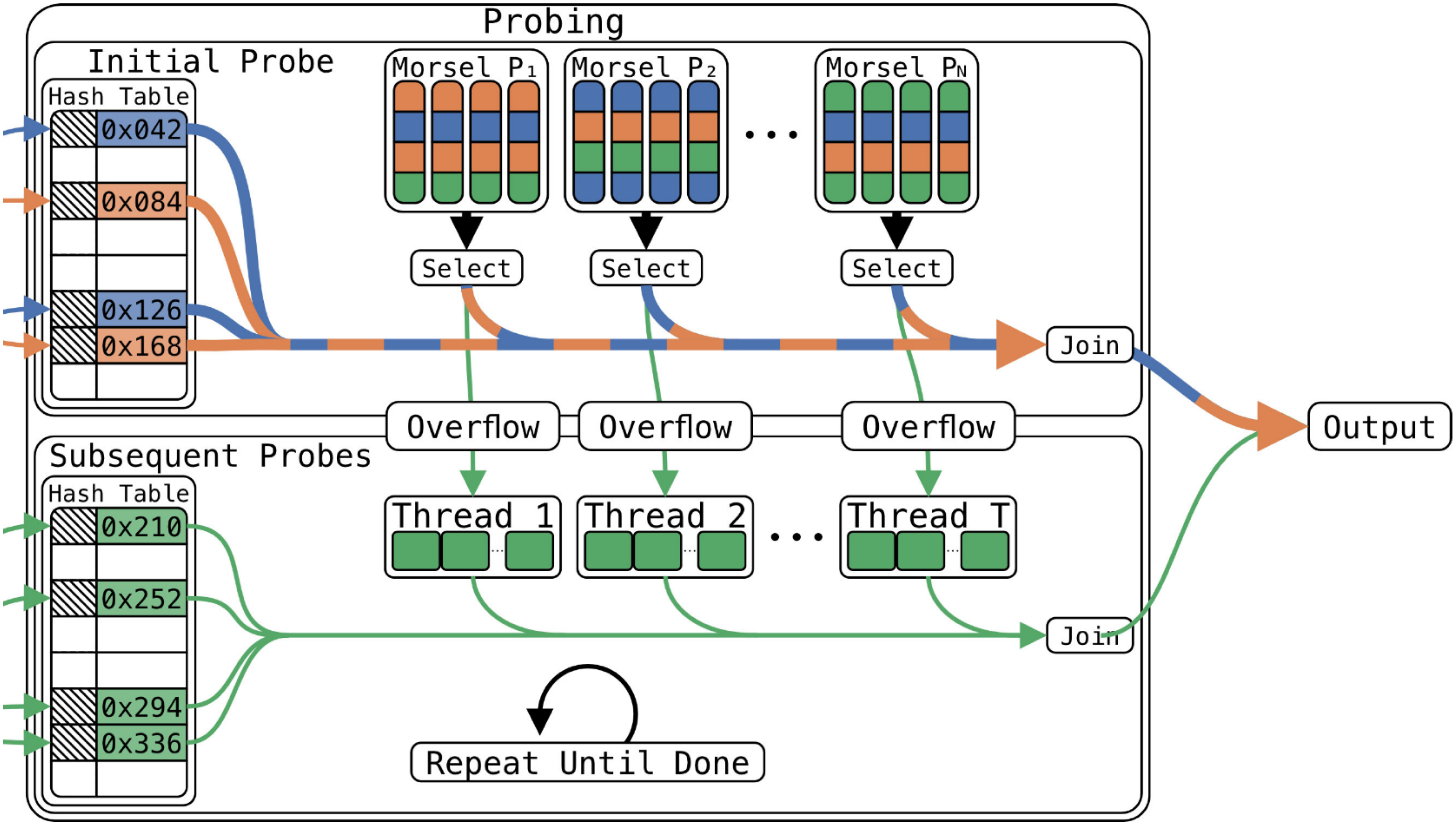


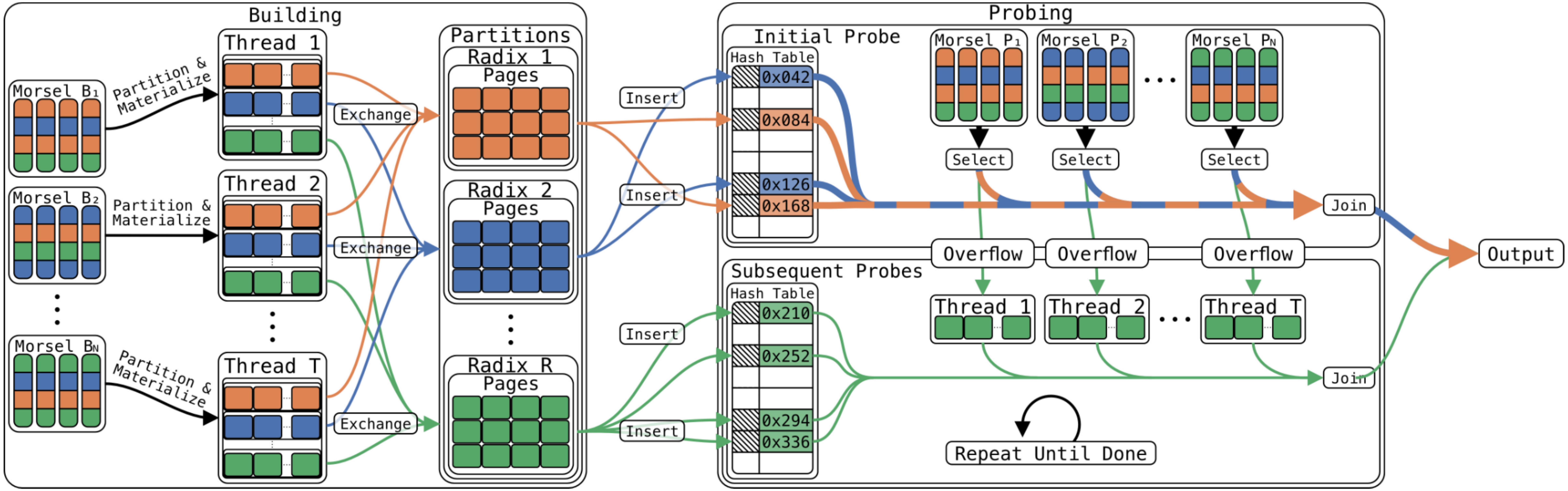
Building
Morsel B1
Morsel B2
Morsel BN
Partition & Materialize
Partition & Materialize
Partition & Materialize
Thread 1
Thread 2
Thread T
Exchange
Exchange
Exchange
Partitions
Radix 1
Pages
Radix 2
Pages
Radix R
Pages


Building
Morsel B_1
Partition & Materialize
Morsel B_2
Partition & Materialize
Morsel B_N
Partition & Materialize
Thread 1
Thread 2
Thread T
Exchange
Exchange
Exchange
Partitions
Radix 1
Pages
Radix 2
Pages
Radix R
Pages

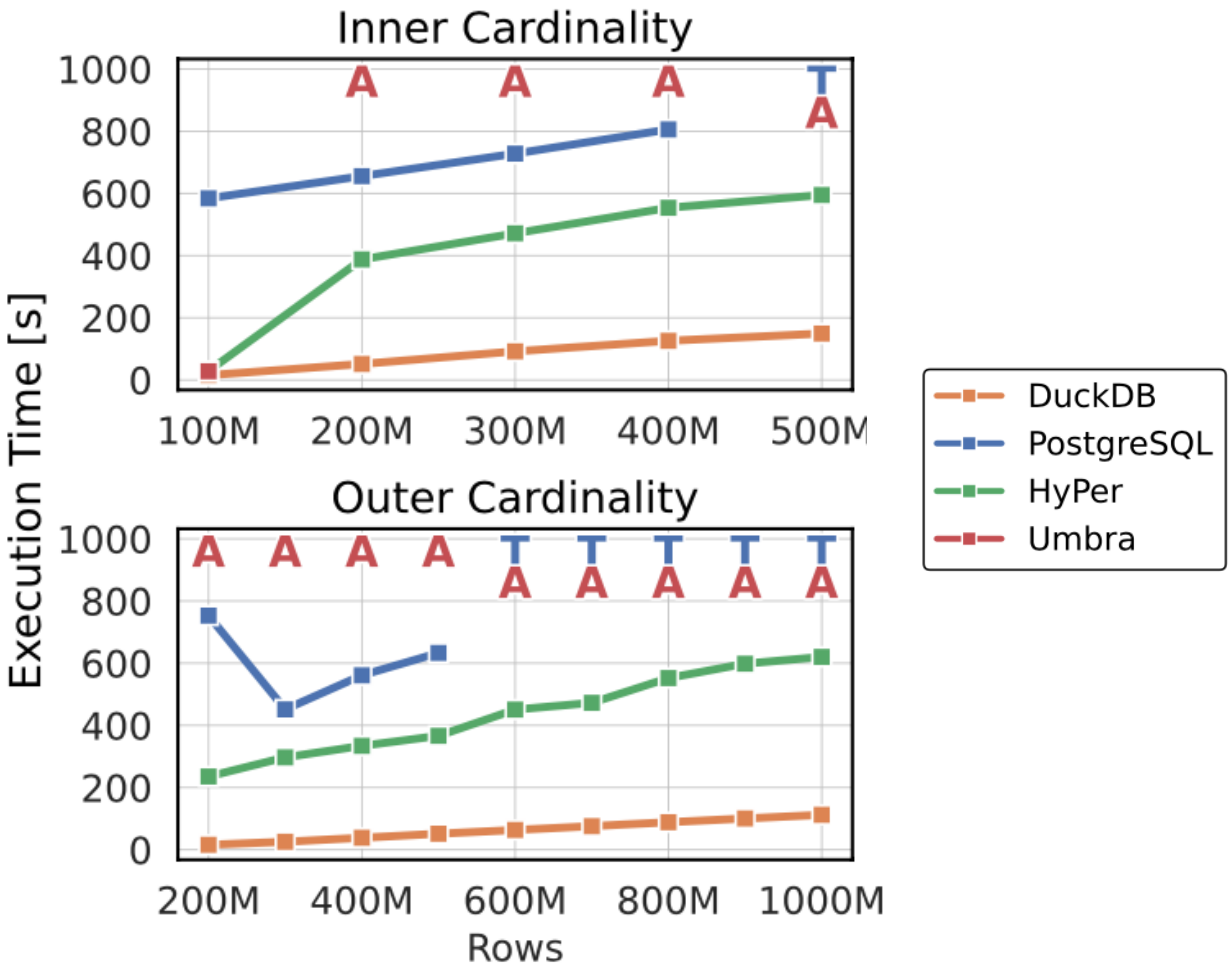

Building
Morsel B1
Morsel B2
Morsel BN
Partition & Materialize
Partition & Materialize
Partition & Materialize
Thread 1
Thread 2
Thread T
Exchange
Exchange
Exchange
Partitions
Radix 1
Pages
Radix 2
Pages
Radix R
Pages


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
Overflow
Overflow
Overflow
Output
Subsequent Probes
Hash Table
0x210
0x252
0x294
0x336
Thread 1
Thread 2
Thread T
Join
Repeat Until Done

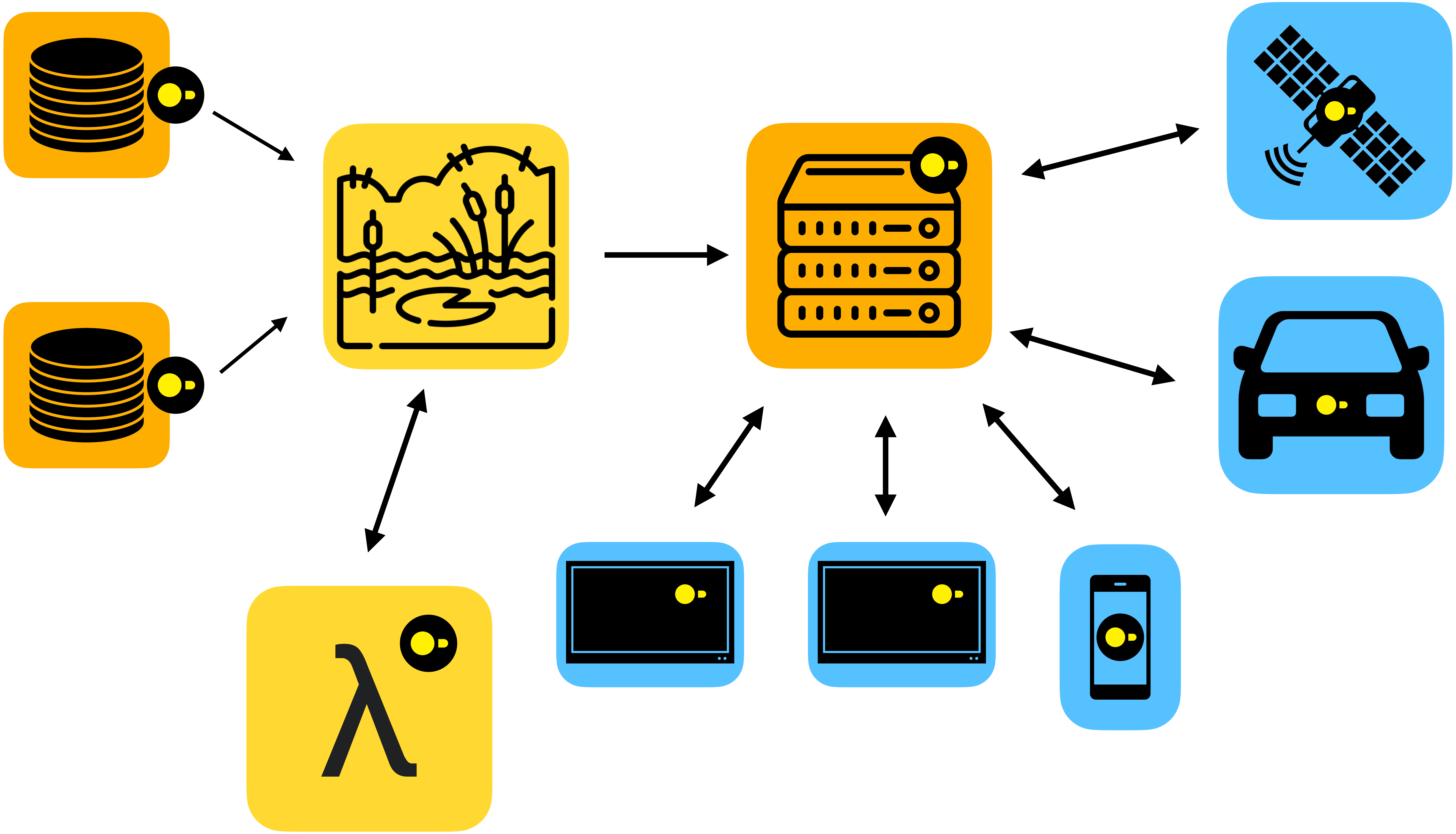


Building
Morsel B1
Morsel B2
Morsel BN
Partition & Materialize
Partition & Materialize
Partition & Materialize
Thread 1
Thread 2
Thread T
Exchange
Exchange
Exchange
Partitions
Radix 1
Pages
Radix 2
Pages
Radix R
Pages
Insert
Insert
Insert
Insert
Probing
Initial Probe
Hash Table
0x042
0x084
0x126
0x168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

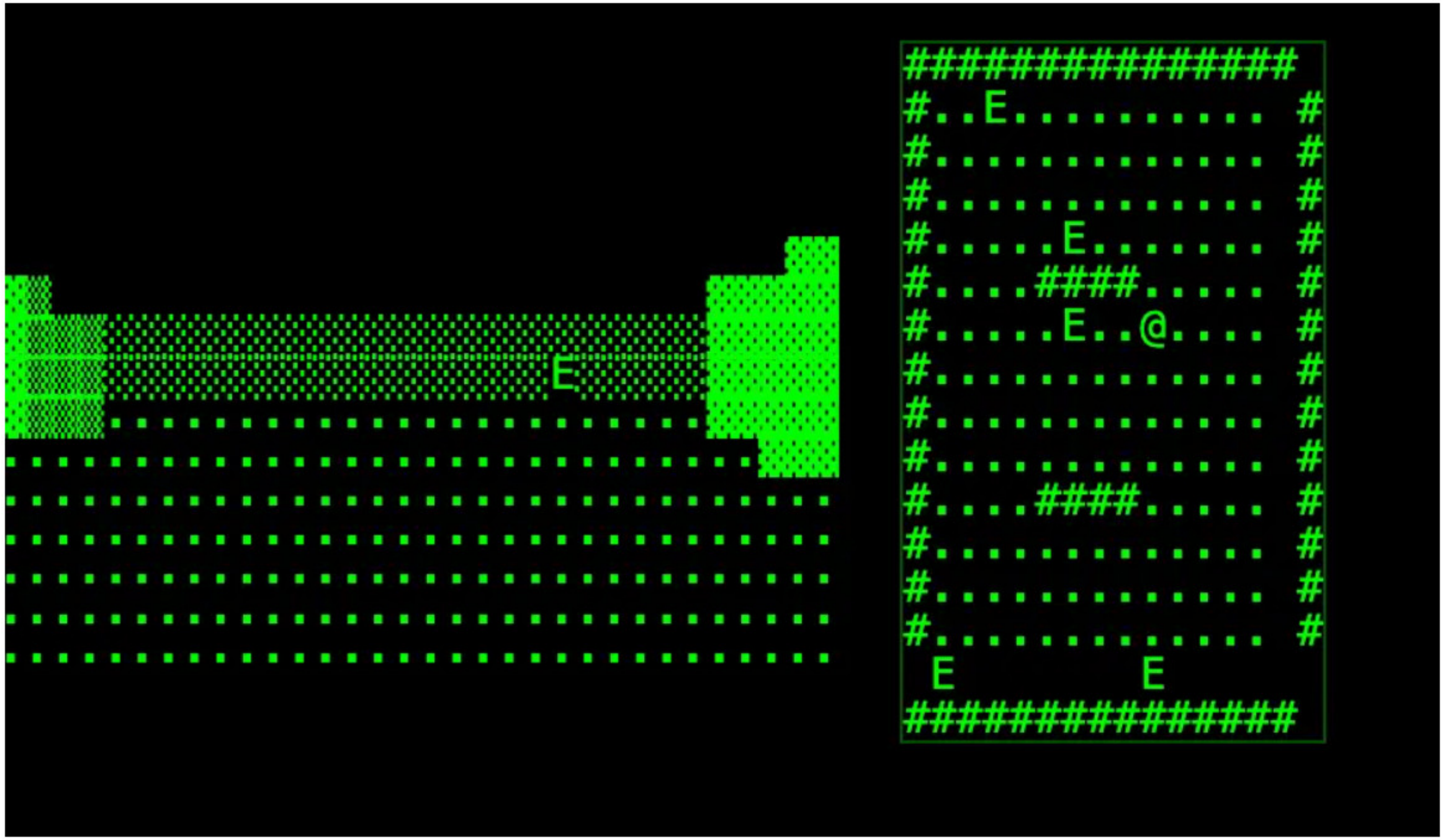


Inner Cardinality
1000
800
600
400
200
0
100M
200M
300M
400M
500M
Outer Cardinality
1000
800
600
400
200
0
200M
400M
600M
800M
1000M
Rows
Execution Time [s]
A
T
DuckDB
PostgreSQL
HyPer
Umbra

Going Back Up

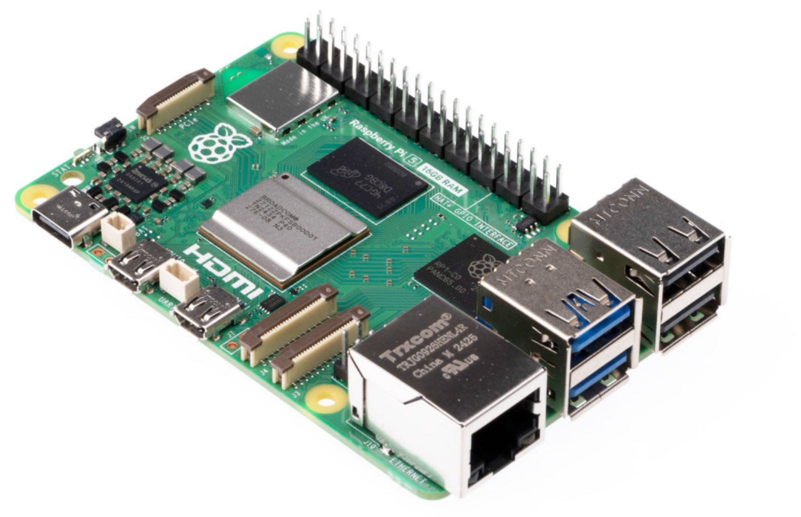


Building a SQL-Powered Doom Clone in the Browser



https://www.hey.earth/posts/duckdb-doom

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

@duckdb.org
@hannes.muehleisen.org