VLDB 2026 · INDUSTRY TRACK

Microsoft

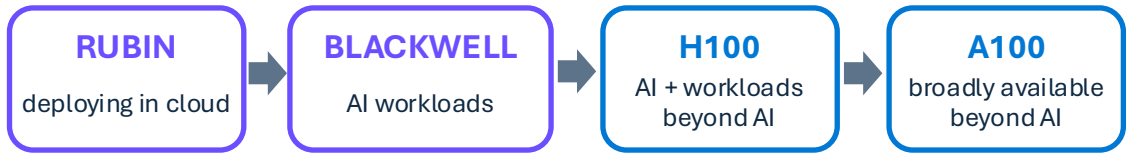
TQP++

Leveraging AI-native Compilation for Query Processing

Wei Cui · Peng Cheng · Carlo Curino · Rathijit Sen · **Matteo Interlandi**

Convergence of two trends

AI FLEET REFRESH



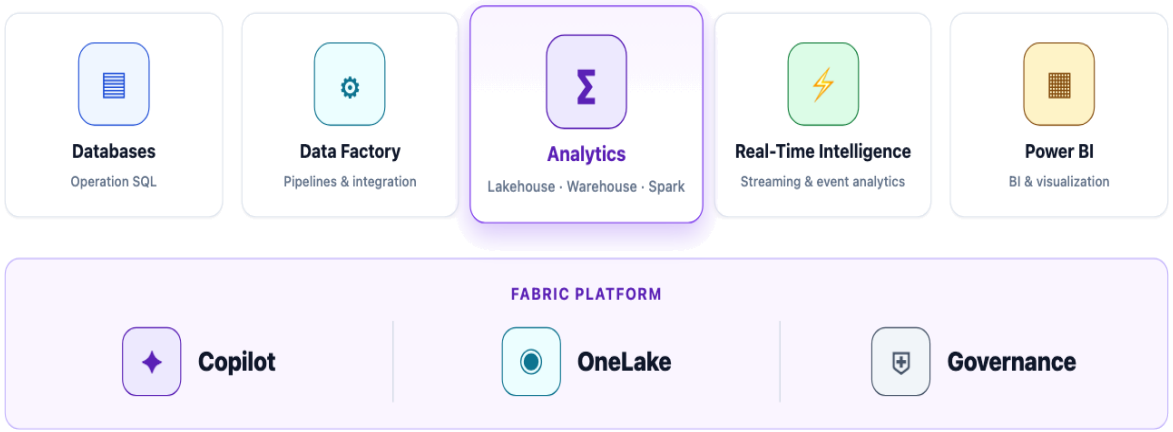
As AI fleets advance, earlier GPU generations become available for workloads beyond AI

~15% **A100 VM price premium**
vs comparable CPU VM

Microsoft Fabric

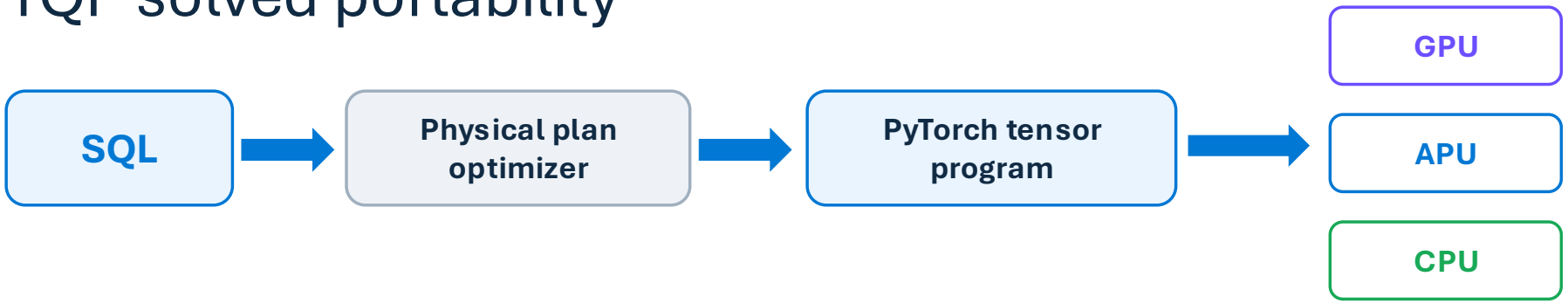
The unified data platform for AI transformation

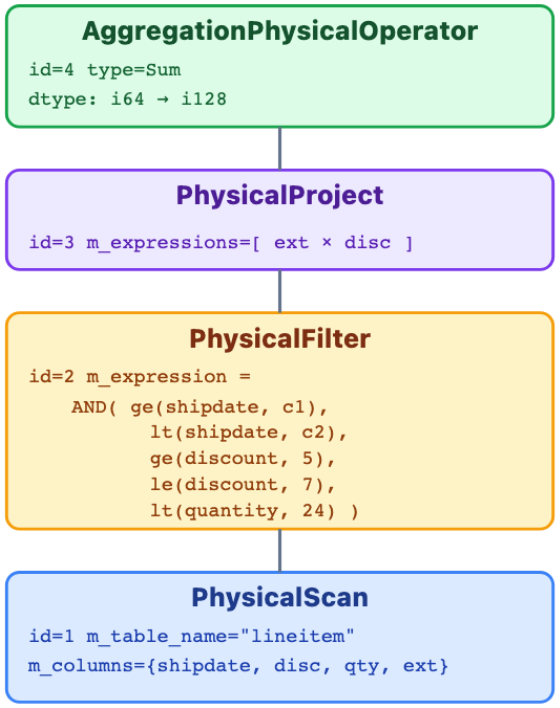
31,000+ Fabric customers **>90%** of the Fortune 500

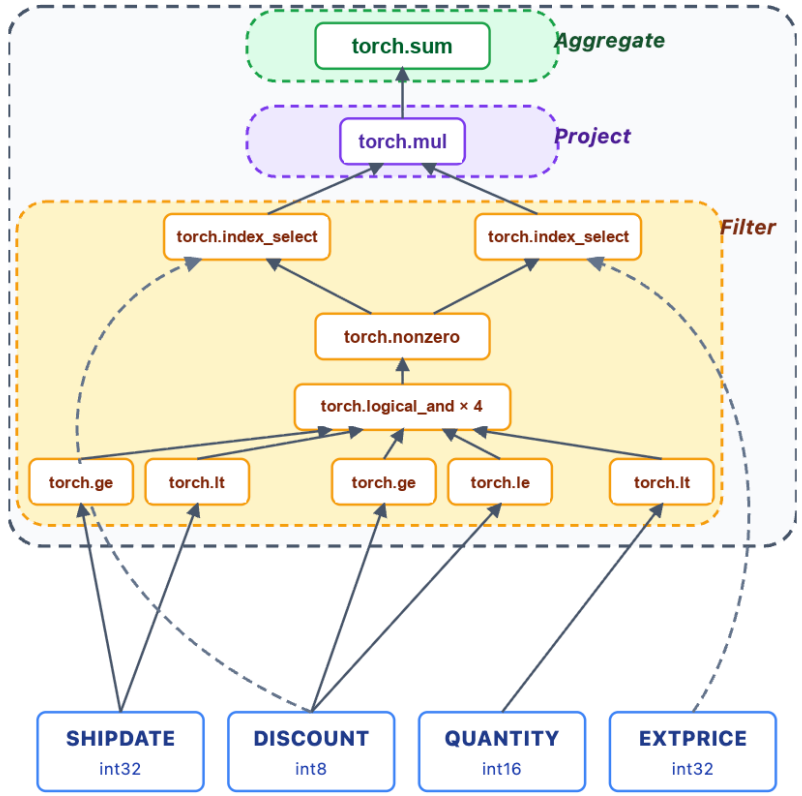


Opportunity: run analytics on hardware originally deployed for AI

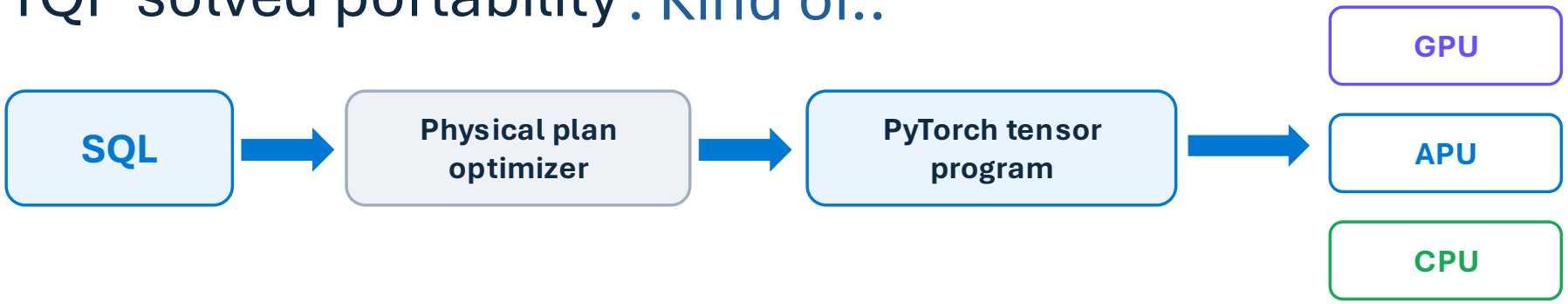
TQP solved portability

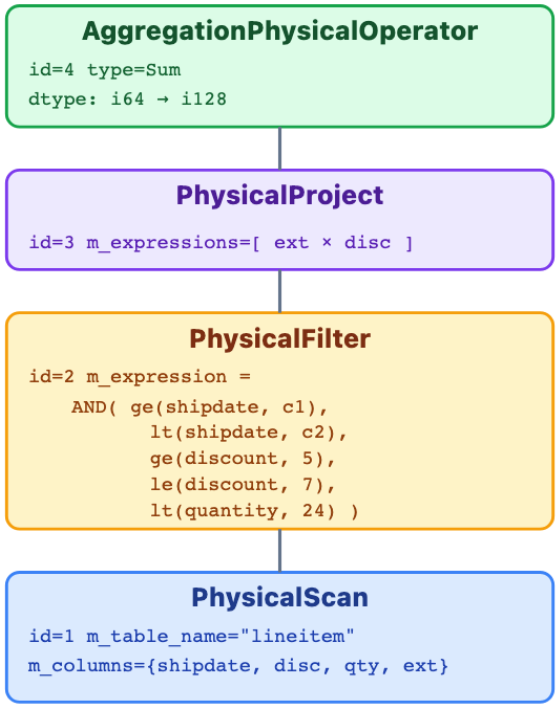


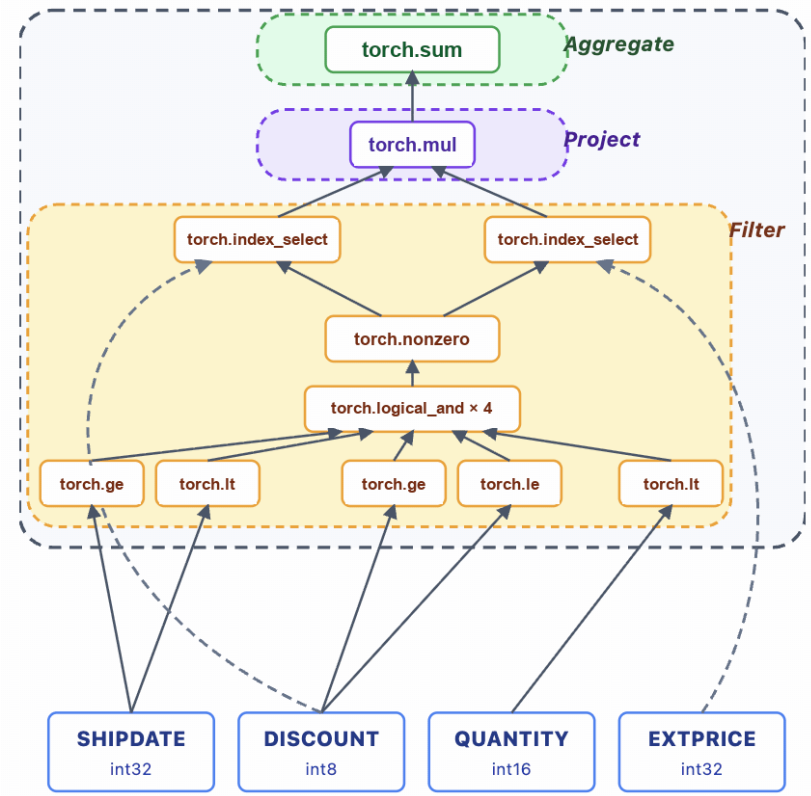


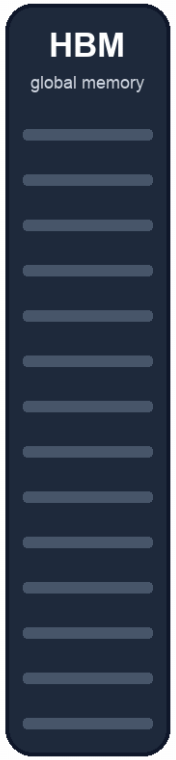


TQP solved portability : Kind of..









PROBLEM 1

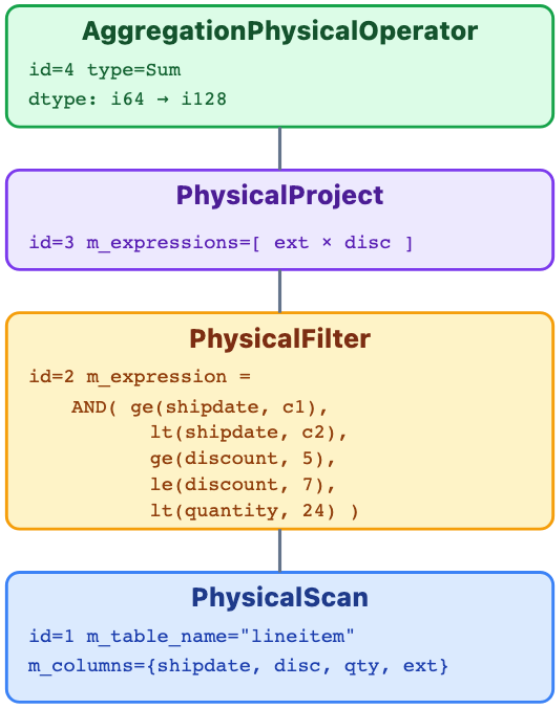
Every operator boundary materializes data

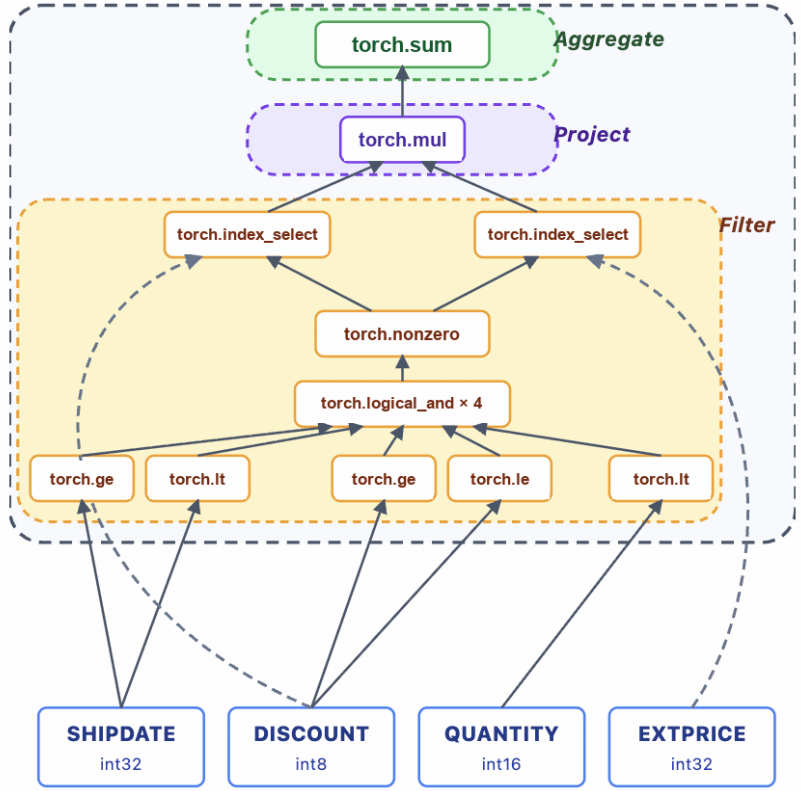
PROBLEM 2

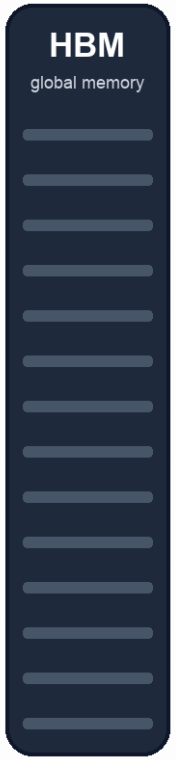
Portable PyTorch or fast custom operators?

TQP solved portability: Kind of..

microsoft / antares







PROBLEM 1

Every operator boundary materializes data

PROBLEM 2

Portable PyTorch or fast custom operators?

SOLUTION

Use compilers for operator fusion and multi-HW code generation

An IR that speaks both tensor algebra and SQL

DENSE ML

Einsum gives the compiler structure

C[N,M] +=! A[N,K] * B[K,M]

LOWERED LOOP NEST

```
parallel_for (n = 0; n < N; ++n)
  parallel_for (m = 0; m < M; ++m) {
    float acc = 0;
    for (k = 0; k < K; ++k)
      acc += A[n][k] * B[k][m];
    C[n][m] = acc;
  }
```

N,M: parallel output loops K: reduction loop acc: aggregation

SPARSE SQL

Antares IR adds the missing primitives

Predicate filter

output[N] = data[N].when(pred[N])

Scatter-sum

output[ids[N]] +=! data[N].when(mask[N])

External Functions (UDFs)

build_key[N].ext_call(sql_hash_build, ...)

Sparse · data-dependent · still optimizable

Expressive enough for relational operators. Declarative enough to optimize

Two SQL patterns, two optimized GPU schedules

LIKE

Compile LIKE as a 1D convolution

Conv1D: out[n] += x[n+k] × w[k]
LIKE: hit[n] &= text[n+k] == pattern[k]

Same overlapping sliding-window access pattern → reuse the same compiler optimization

1	**Infer overlap**	Compute the union of text bytes required by every thread in the block
2	**Load once**	Issue one coalesced HBM → shared-memory tile load
3	**Reuse on-chip**	Threads read their overlapping windows from shared memory into registers
4	**Emit matches**	Only final predicate results leave the block; source bytes are not reloaded

AGGREGATION

Statistics choose the highest tier that fits

state size = group cardinality × value width

The compiler uses state sizes to emit the paths that maximizes on-chip residency

groups ≤ K_1 → registers
thread-private accumulation · no atomics

↓

K_1 < groups ≤ K_2 → shared memory
block-local atomic accumulation

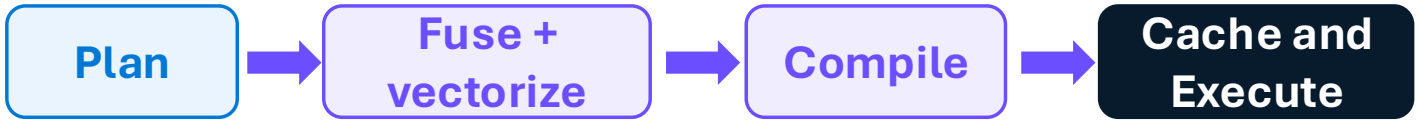
↓

groups > K_2 → sharded HBM buffers
spill only state that cannot stay on-chip

Compilation-Time Specialization + Runtime adaptation

TWO COMPILATION MODES

Just-in-time Fusion (JTF)



Recurring query shape

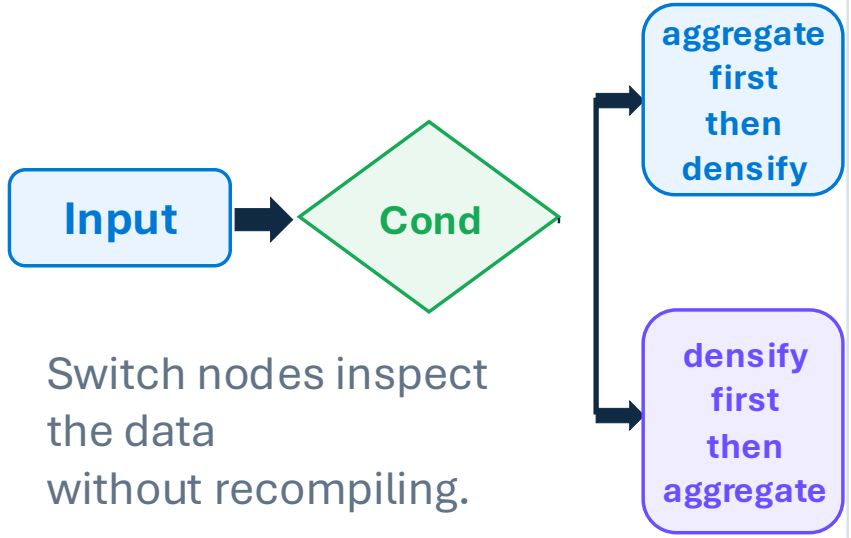
0.1–2.7 s compile time

Ahead-of-time Fusion (ATF)

One-shot or ad-hoc query

RUNTIME ADAPTATION

Multi-gated execution graph



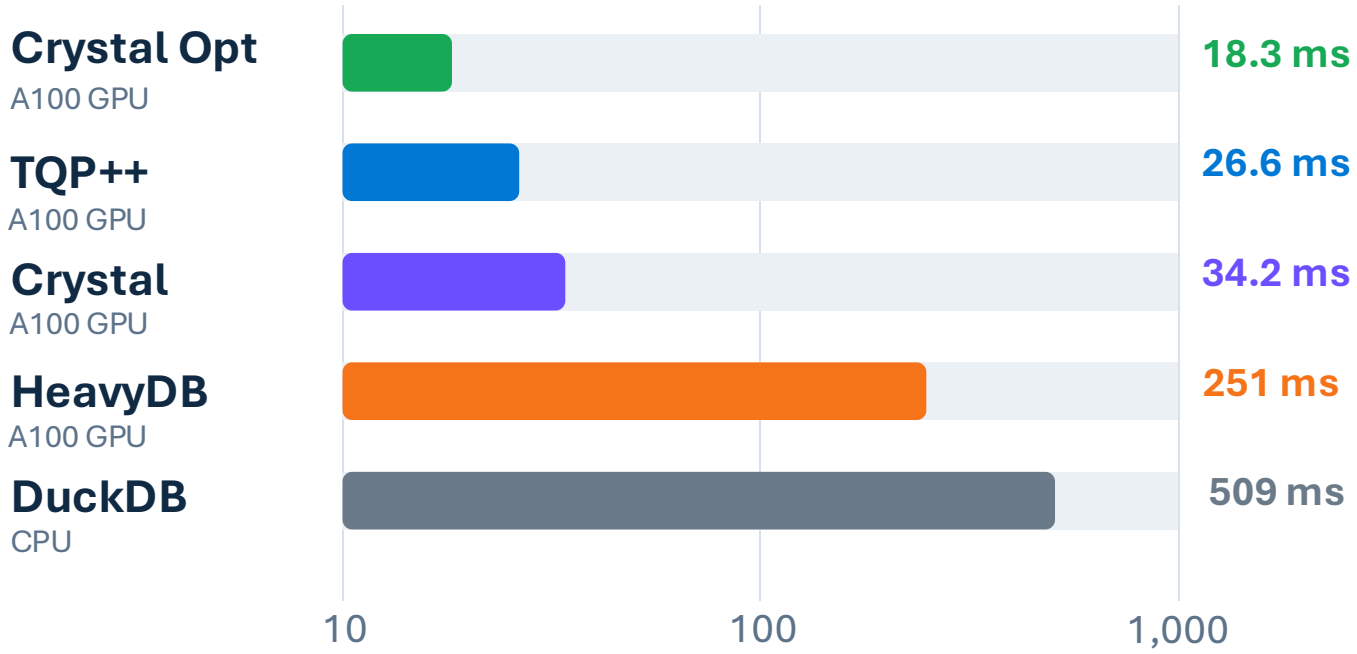
Switch nodes inspect the data without recompiling.

group cardinality · join selectivity

SSB and TPC-H performance

SSB SF 16

Total runtime across systems



TQP++ is competitive vs hand optimized queries

TPC-H SF 100

TQP++ total runtime

same codebase · data resident in GPU memory

A100	1.11 s
H100	0.69 s
MI300	0.65 s

A100: 15× faster than DuckDB

The research became a foundation

TQP++ seeded a broader accelerated analytics program

TQP++

Single GPU foundation

0.65 s

TPC-H SF 100 · MI300

This paper

SCALE OUT

Distributed TQP++

0.53 s

TPC-H 1 TB · 40 H100s

Wed 13:45 – 15:15

ZIPFLOW

Compressed data movement

2.08×

vs nvCOMP end-to-end

Earlier this session

CODDSPEED

Fabric DW productization

GPU

SIGMOD 2026 Best Industry Paper Award

What transferred

tiered scheduling · dispatch logic · fusion patterns

What production didn't adopt

JTF (overheads) · the compiler (maturity)

AI-native compilation brings performance without compromising on portability