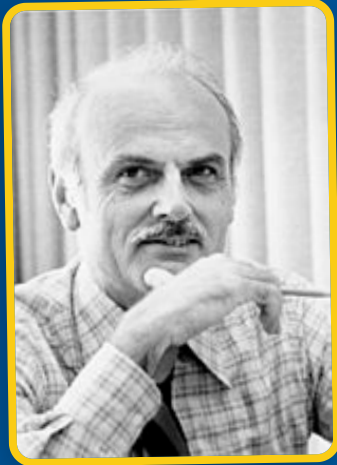




CoddSpeed!

Hardware Accelerated Analytics in Microsoft Fabric

PRESENTED BY

Matteo Interlandi

Microsoft Gray Systems Lab

HPTS October 4–7, 2026

Why GPU acceleration for the data warehouse?

CPUs have plateaued

Per-core perf is flat; scaling a DB engine across many cores is hard.

Accelerators everywhere

\$5T+ AI infra by 2030 — GPUs, ASICs, NVLink, CXL orders of magnitude faster than CPUs.

Agents = bursty workload

Latency-sensitive queries fanned out for users — exactly what accelerators are good for.

The hard part

Deliver this without locking the warehouse to one vendor, chip, or fabric.

THE CODDSPEED BET

Build **coprocessor** and **data-movement** abstractions that let the same SQL engine offload sub-plans to GPUs today and other hardware tomorrow — with the rest of the stack untouched.

Design choices

①

Two minimal abstractions

CAL (Coprocessor Abstraction Layer) — ships query fragments to the coprocessor.

DAL (Data Abstraction Layer) — moves the bytes between host and coprocessor.

Swap any hardware or network transport — the rest of the stack stays untouched.

②

Partitionable execution

To minimize **OOMs** on accelerators with minimal on-board memory, the host partitions large inputs and runs the same fragment per chunk.

③

Per-row-group CPU fallback

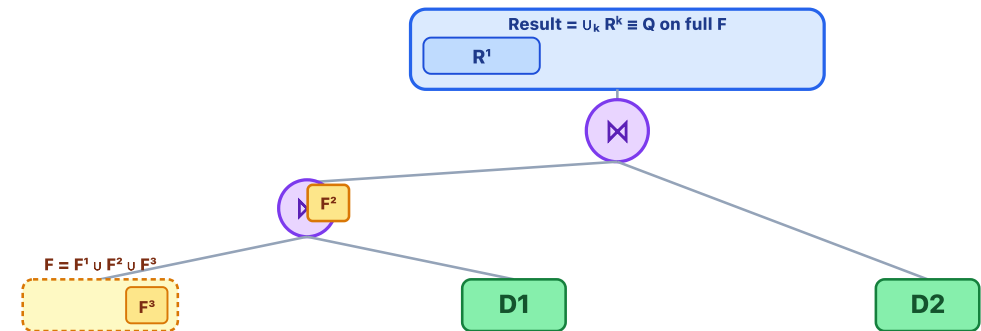
If a row group fails on GPU, the host re-runs **only that row group** on CPU.



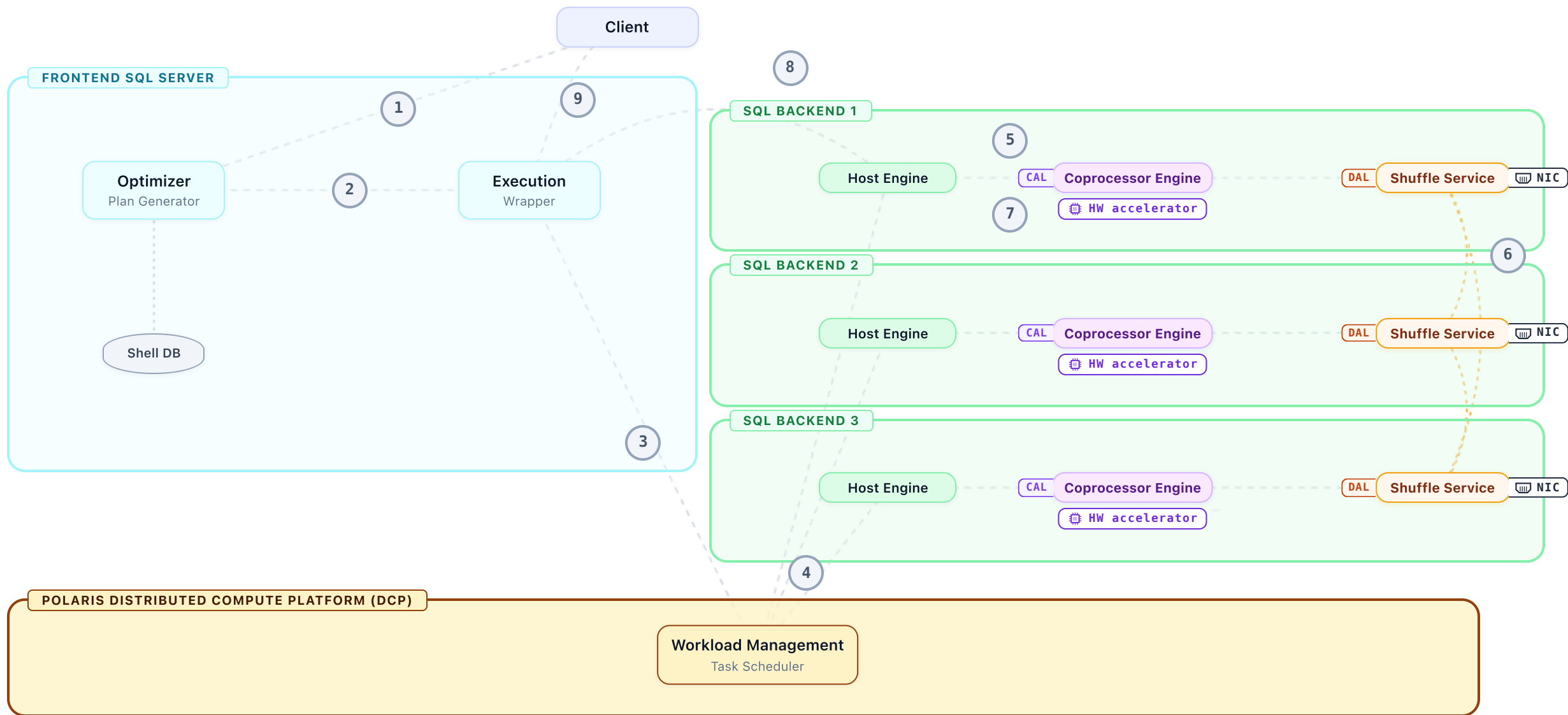
Partitionable execution

$$Q[T_1, \dots, T_i, \dots, T_n] = \cup_k Q[T_1, \dots, T_i^k, \dots, T_n]$$

for $T_i = \cup_k T_i^k$



Host splits a large input **F** into chunks **F¹...F^k**; the same offloaded fragment runs per chunk; per-chunk results **R^k** are unioned.



Inside the Optimizer — what to offload, and how

① Capability negotiation

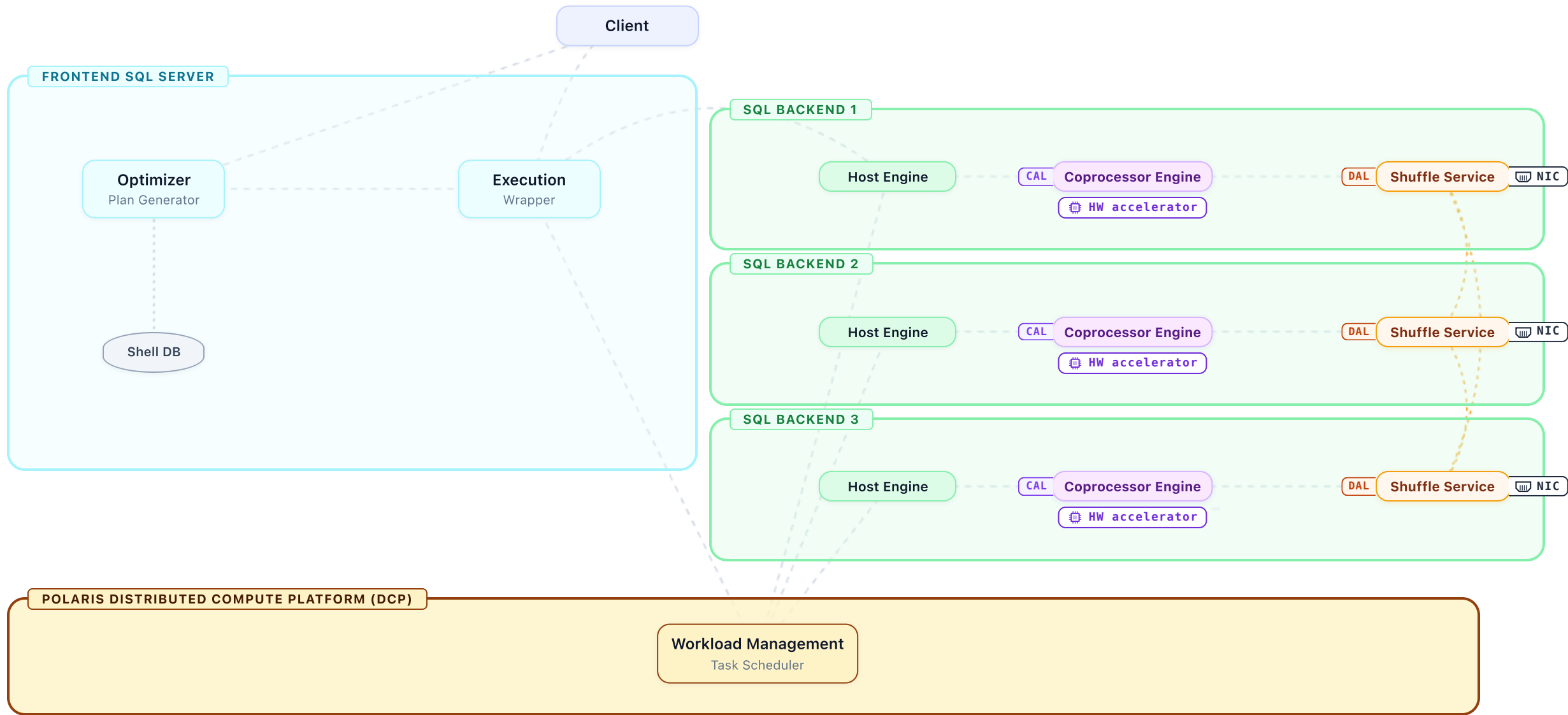
The optimizer asks CAL what this coprocessor can do — supported **operators**, **data types**, and **intrinsic**s. Only matching regions are offloaded; the rest stay on the host.

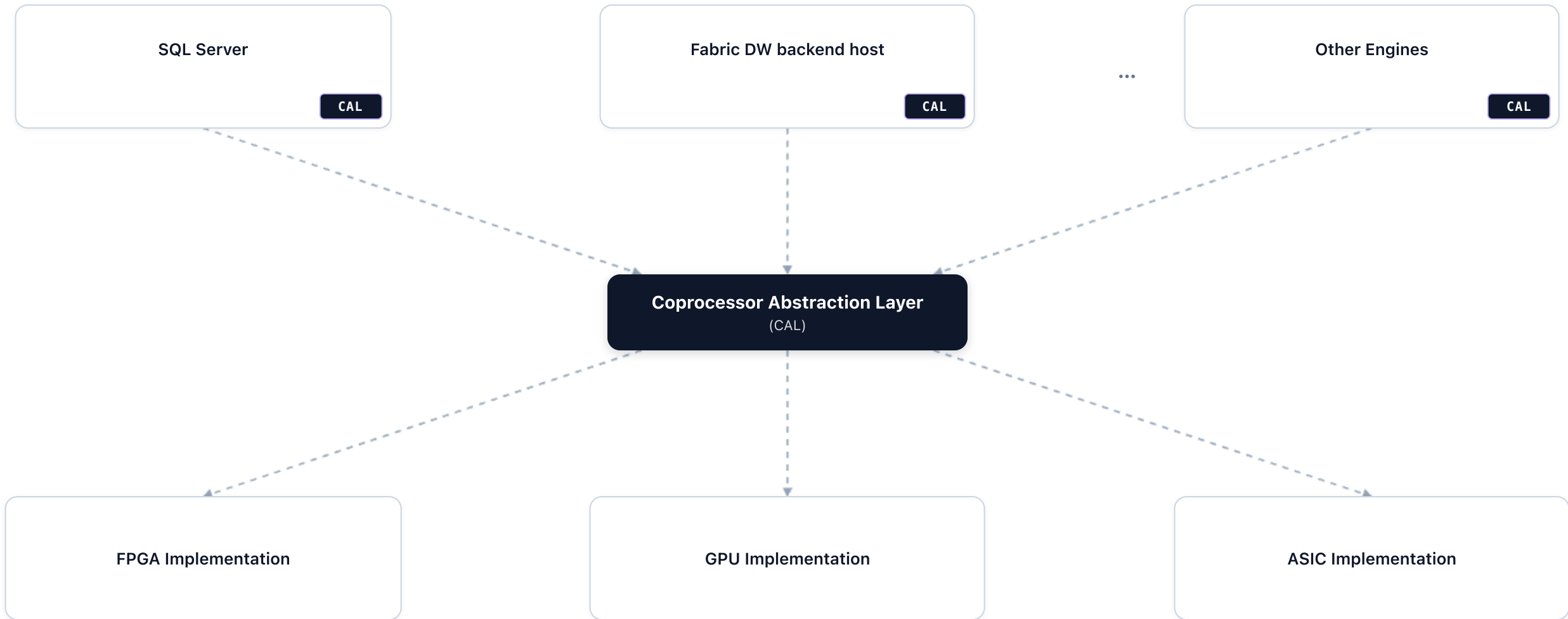
② Coprocessor as a physical operator

The host engine still owns parsing, optimization and distributed coordination. The optimizer learns to detect **eligible regions** of a plan, wrap them under a single **Coprocessor** operator, and let CAL execute the fragment.

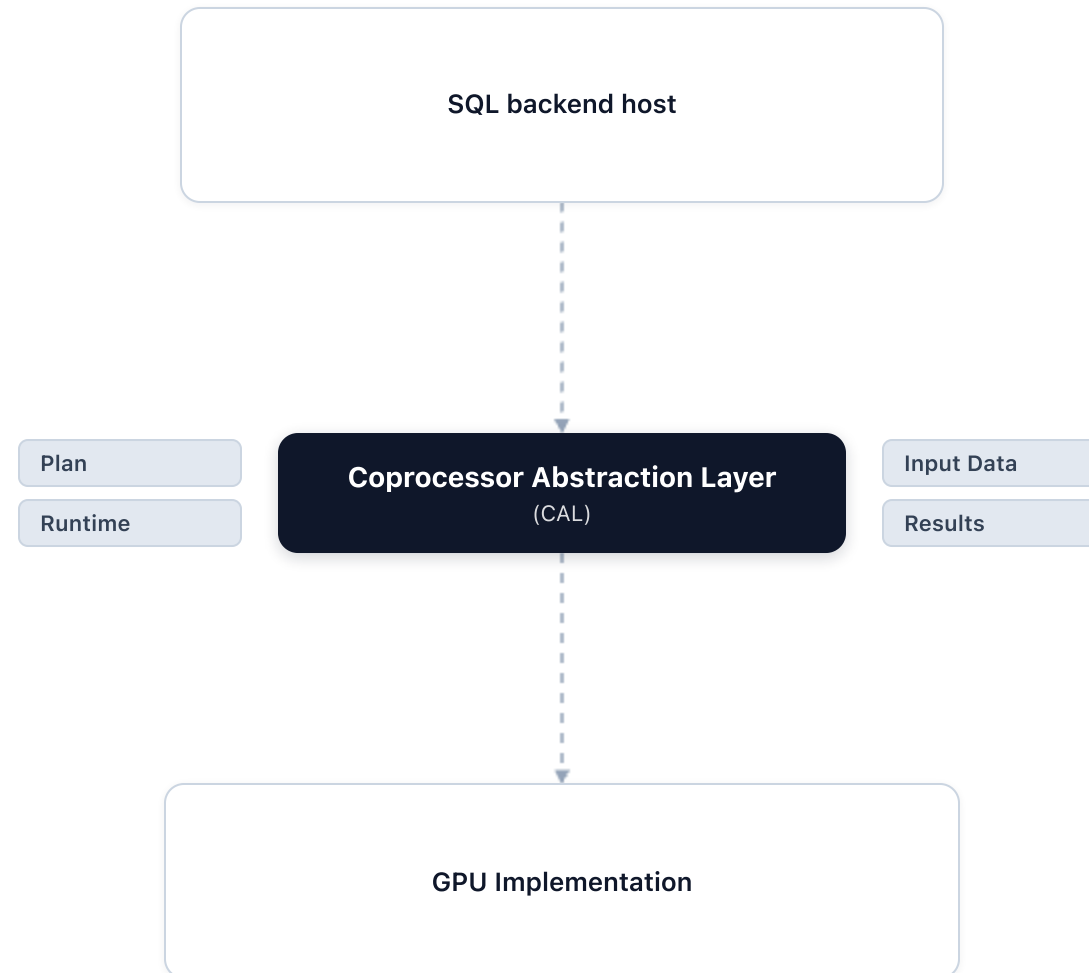
③ Subtrait — a portable plan

CAL relies on **Subtrait**, an open standard for serializing relational algebra. The host and the coprocessor evolve independently as long as they speak this shared plan language.





Zoomed in: one Fabric DW ↔ CAL ↔ GPU path, carried by four contracts.



Click → reveal the concrete Fabric DW ↔ GPU path

Inside the GPU Engine

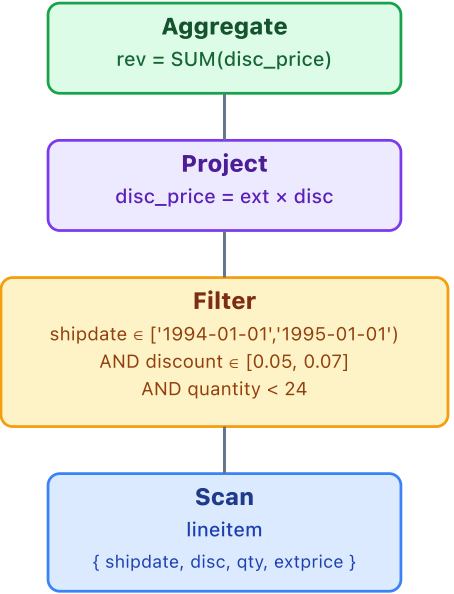
A productized version of TQP [VLDB 2022]

TQP (Tensor Query Processor) converts SQL physical plans into PyTorch programs that run on any hardware PyTorch supports. We hardened the same SQL-on-tensors idea for production workloads.

PyTorch + custom CUDA kernels

Like TQP, we use **PyTorch** for memory allocation and operator implementation. We rely **less on PyTorch ops** and **more on custom CUDA kernels** tuned for SQL semantics.

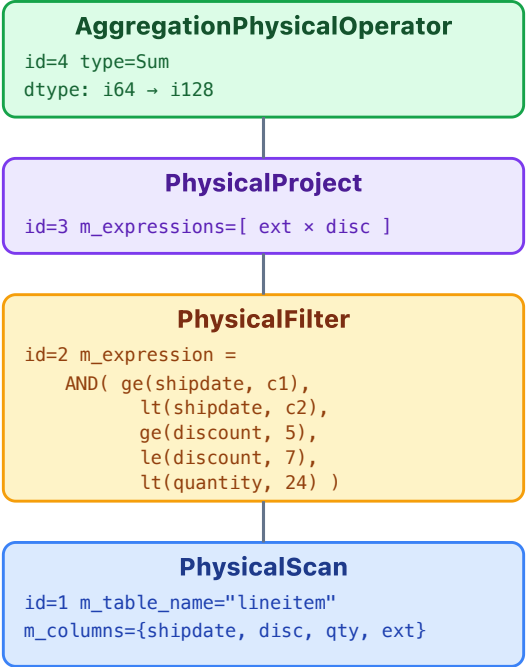
Substrait plan



physical plan from the host optimizer

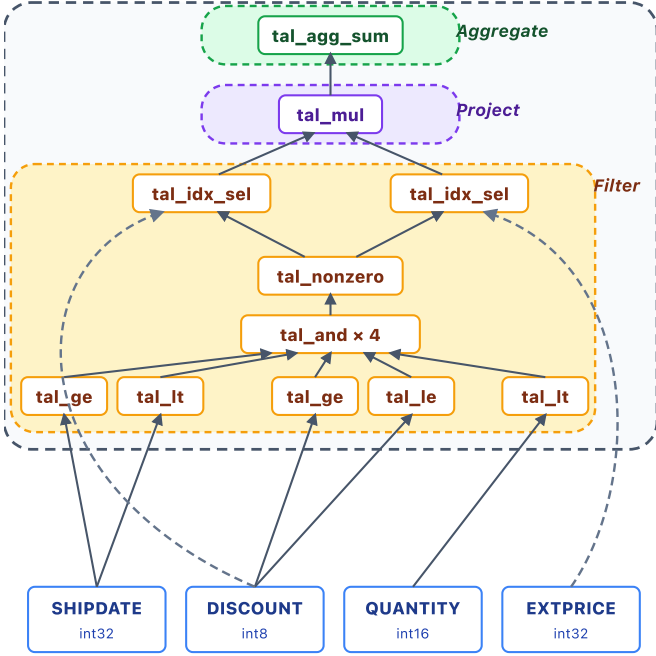
Parse
→

In-Memory Physical Plan



Bind & Build
→
+ inputs

Execution Plan



physical operators explode into many **sub-operators** on the GPU

Single A100, TPC-H SF=100

GPU

1× A100 (80 GB)

CPU BASELINE

EPYC 7V13 Milan · 24 cores / 220 GB

WORKLOAD

TPC-H SF 100

METRIC

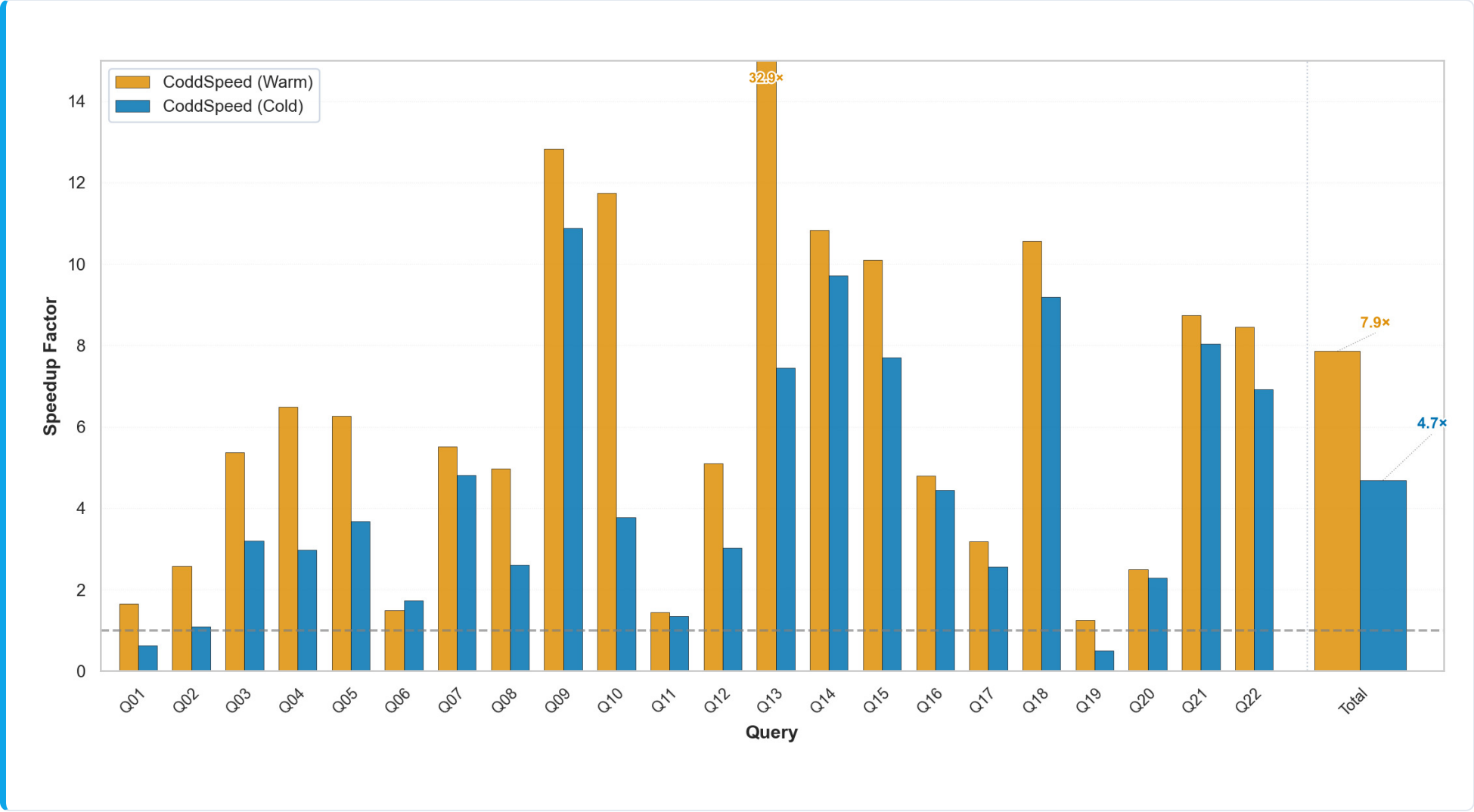
Speedup vs Fabric DW (single node)

RUNS

Median of 5

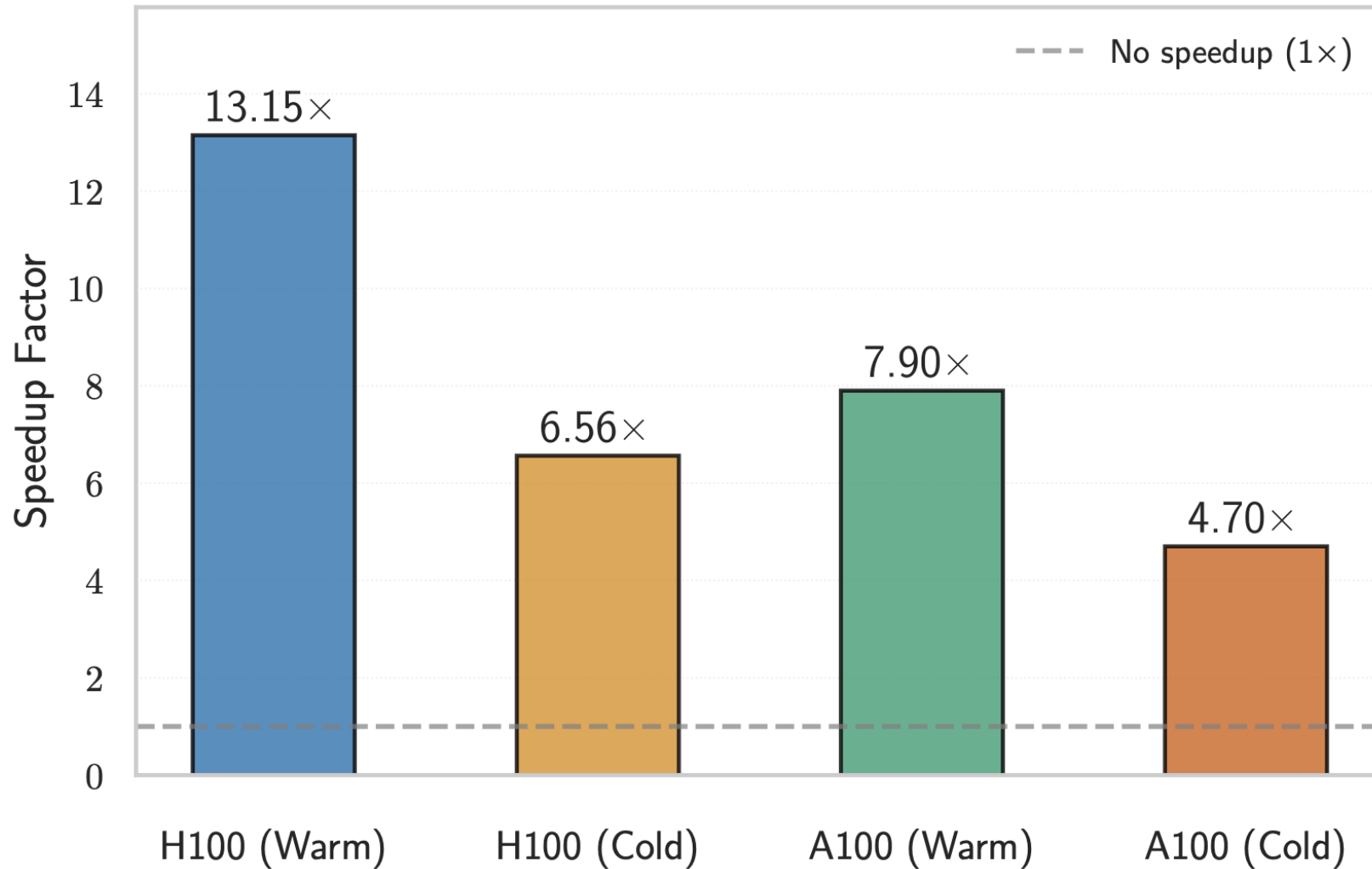
MODES

Cold (CPU→GPU at exec)
Warm (data in GPU)



The paper also reports results on an **internal production workload** (16 queries · 300 GB compressed real-world data): **up to ~15x warm speedup**.

Generation jump — H100 vs A100

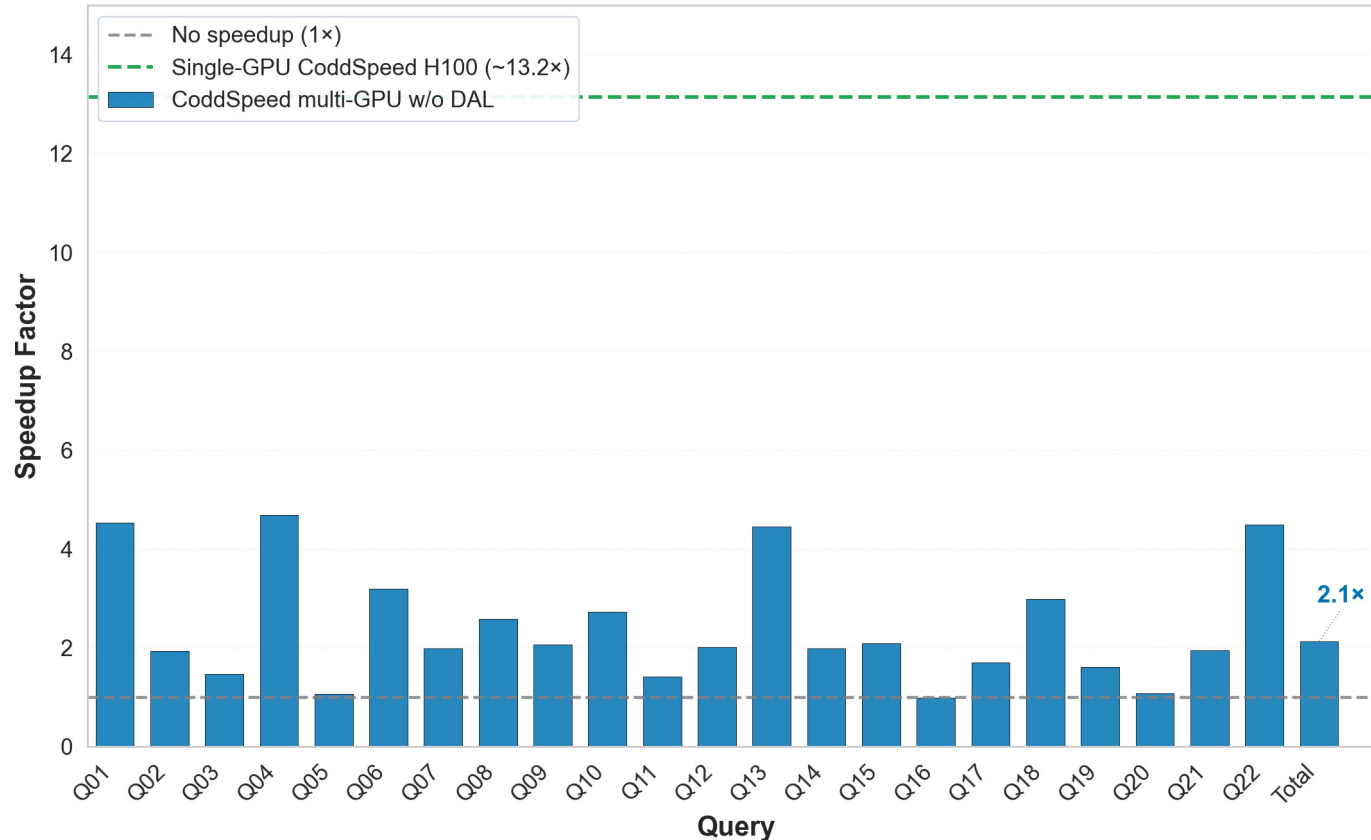


+50%

over A100

- HBM bandwidth dominates analytical workloads.
- H100 → 3 TB/s vs A100 → 2 TB/s.
- The CoddSpeed stack inherits this for free.

Fast compute is not enough — without a fast data plane



~2x

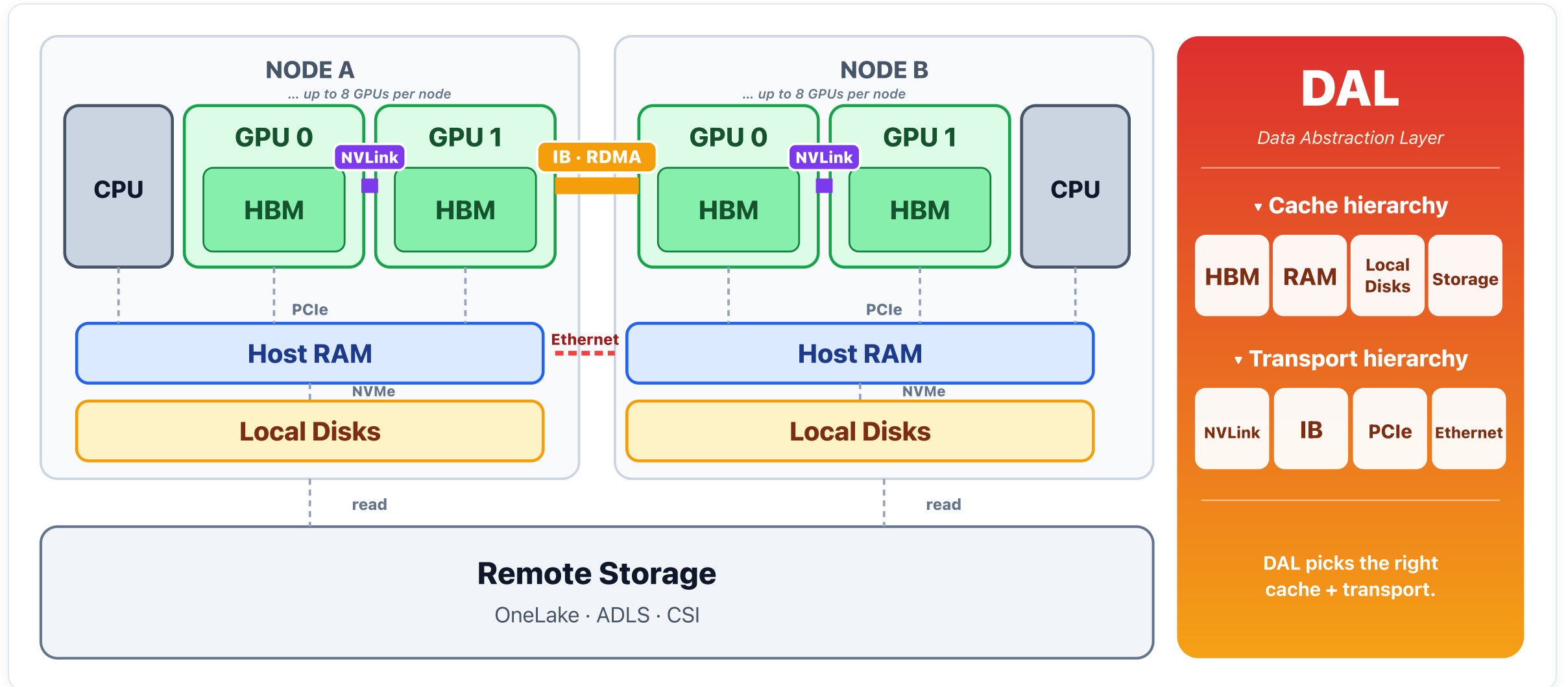
Multi-GPU H100 · TPC-H 1 TB

- Compute moved off the CPU, but across GPUs data is moved through a generic **TCP/IP shuffle**.
- **Amdahl law:** Communication is now the bottleneck.

vs Fabric DW CPU baseline · same kernels as the single-GPU run

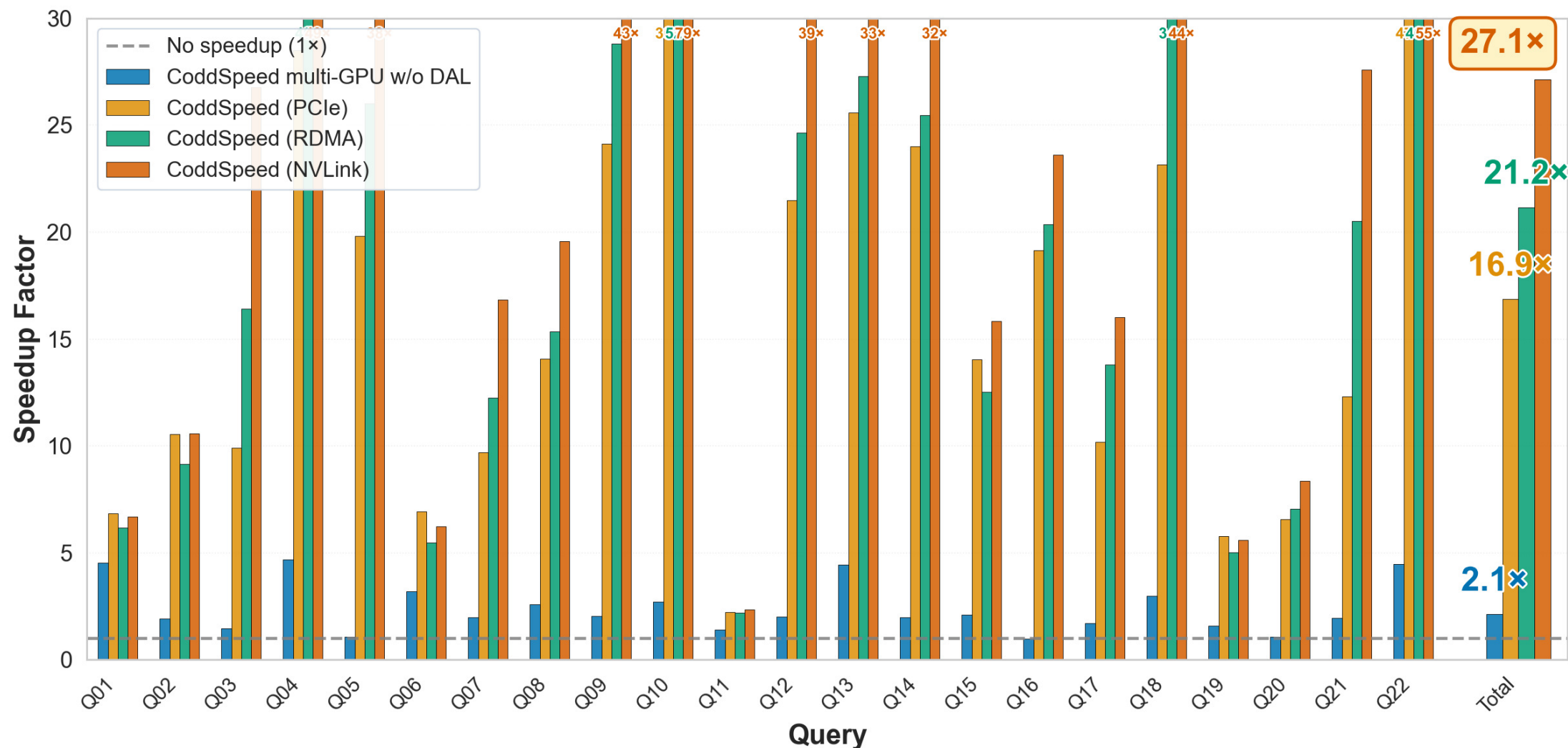
Once the compute gets fast enough, **data transfer becomes the bottleneck**. We need a memory-hierarchy and transport aware data plane → **DAL**.

DAL — one API across cache tiers AND interconnect



Scaling out — multi-GPU with DAL transports

TPC-H SF=1000 (1 TB) · 8 × H100 · warm cache · speedup over Fabric DW CPU



Beyond GPUs — FPGA and ASIC backends (Prototypes)



FPGA



GPU
COVERED ABOVE



ASIC

FPGA

- **Intel Agilex 7 I-Series AGI 041** — ~4M LEs, 1.3M ALMs, 377 Mb embedded memory.
- **PCIe Gen5 dual x8** (≈12 GB/s bidi) · 4× DDR4-2666 (85.3 GB/s peak).
- Built with **Kanagawa**, Microsoft's vendor-agnostic HLS toolchain.

Speeddata APU (ASIC)

- **C200 APU** — CGRA-based dataflow on a **PCIe 5.0 x16** board with 32 GB on-board RAM.
- Zero-copies Parquet from host memory; **decompression and decoding in hardware**.
- Native relational ops — select, project, join, aggregation; complex/nested types incl. strings.

TPC-H SF=100 — SPEEDUP VS SQL SERVER SINGLE CORE

SQL Server + FPGA	Q06 4.5×	Q12 14.5×	Q14 48.1×	Q19 6.6×	
Standalone APU	Q06 36.7×	Q07 40.7×	Q12 49.4×	Q13 259×	Q14 61.5×

Wrapping up

WHAT WE LEARNED



Nobody knows what the hardware landscape will look like in 5–10 years

CPUs, GPUs, TPUs, and whatever comes next will keep churning. Bet on abstractions that survive the churn—not on one device.



Our Bet: Two abstraction layers

CAL abstracts coprocessor compute; **DAL** abstracts data movement. New silicon can arrive without an engine rewrite.



Validated across hardware

Multiple hardware configurations and vendors stress-tested whether both APIs stay flexible enough.

We are considering **open sourcing the APIs** — let us know if you are interested.

To raise an Engine... it takes a village!



Microsoft®
Research



+ Product

CORE TEAM

- Dalitso Banda
- Nicolas Bruno
- Jesús Camacho-Rodríguez
- Carlo Curino
- Brandon Haynes
- Mingsheng Hong
- Matteo Interlandi
- Matthias Jasny
- Konstantinos Karanasos
- Brian Kroth
- Kwanghyun Park
- Raghu Ramakrishnan
- Karla Saur
- Rathijit Sen
- Markus Weimer
- Hangdong Zhao
- Dong He
- Supun Nakandala
- Bowen Wu
- **...and more**

RESEARCH COLLABORATORS

- Surajit Chaudhuri
- Peng Cheng
- Wei Cui
- Bailu Ding
- Johannes Gehrke
- Zezhou Huang
- Yinan Li
- Lukas M. Maas
- Kaushik Rajan
- Sampath Rajendra
- Ziyun Wei
- Lidong Zhou
- Spyros Blanas
- **...and more**

PRODUCT PARTNERS

- Josep Aguilar Saborit
- Momin Al-Ghosien
- Israel Arroyo
- Yuki Asada
- Evgeny Babin
- Ronak Bajaj
- Sharanya Bhat
- Kevin Bocksrocker
- Morten Borup Petersen
- Ron Boskovic
- Esteban Calvo
- Qi Chen
- Bogdan Crivat
- Conor Cunningham
- Pravija Danda
- Mandar Datar
- Marius Dumitru

PRODUCT PARTNERS

- Victor Fu
- Kevin Gaffney
- Cesar Galindo-Legaria
- Apurva Gandhi
- Advitya Gemawat
- Ashit Gosalia
- Vivek Gupta
- Zach Iracheta
- Anudeep Kambapu
- Sai Sumanth Kammal Shetty
- Babu Kandimalla
- Adarsh Kapil
- Maraki Ketema
- Aparna Konduri
- Igor Konev
- Chidamber Kulkarni
- YongChul Kwon
- Denis Lamtsov

PRODUCT PARTNERS

- Aaron Landy
- Hans Lehnert Marino
- Aaron Moore
- Nevena Nikolic
- Ehi Nosakhare
- Artem Oks
- Craig Peeper
- Blake Pelton
- Luis Penaranda
- Aaron Pitman
- Parminder Poonian
- Andrew Putnam
- Ravi Ramamurthy
- Minsoo Rhu
- Rob Rydberg
- Krystian Sakowski
- Zhiyang Shen
- Runbin Shi
- Nicholas Simmons
- Krish Srinivasan
- Prasanna Sundar
- Tino Teresko
- Gwangoo Yeo
- Gyeong-In Yu
- Lihao Zhang
- Qianxi Zhang
- Marko Zivanovic
- **+ Nvidia friends · ...**

★ THANK YOU ★

Questions?

CoddSpeed: Hardware-Accelerated Query Processing in Microsoft Fabric

PAPER

doi.org/10.1145/3788853.3803077

CONTACT

Matteo Interlandi · mainterl@microsoft.com

By the CoddSpeed team — Microsoft GSL · MSR · Azure Data · Hardware Systems