



From MapD to Sirius: Co-Designing GPU-Native OLAP

ADMS Keynote / VLDB Boston | August 31, 2026

Joshua Patterson, VP Solutions Architecture, NVIDIA



Agenda

- 01 What's Changed Since the Origin of Data Processing
- 02 Four Engine Eras: MapD, BlazingSQL, Spark RAPIDS (cuDF), Theseus
- 03 Reference Design Evolution and Cumulative Advantages
- 04 Modernizing Existing Engines: Benchmarks
- 05 Sirius: Blackwell-Native Results and Prototypes
- 06 Research Agenda and Direction

What's Changed Since the Origin of Data Processing



GPU Data Analytics at an Inflection Point

Database operations are parallel by design, so analytics has always been a natural fit for GPU architecture. What held it back was never the math — it was three hardware constraints. Each one has now moved.

01 CHALLENGE

Limited GPU memory capacity

Working sets had to fit in a handful of gigabytes, so anything real spilled off the device.

02 CHALLENGE

CPU–GPU PCIe bottleneck

Feeding the GPU cost more than computing on it. The link, not the kernel, set the ceiling.

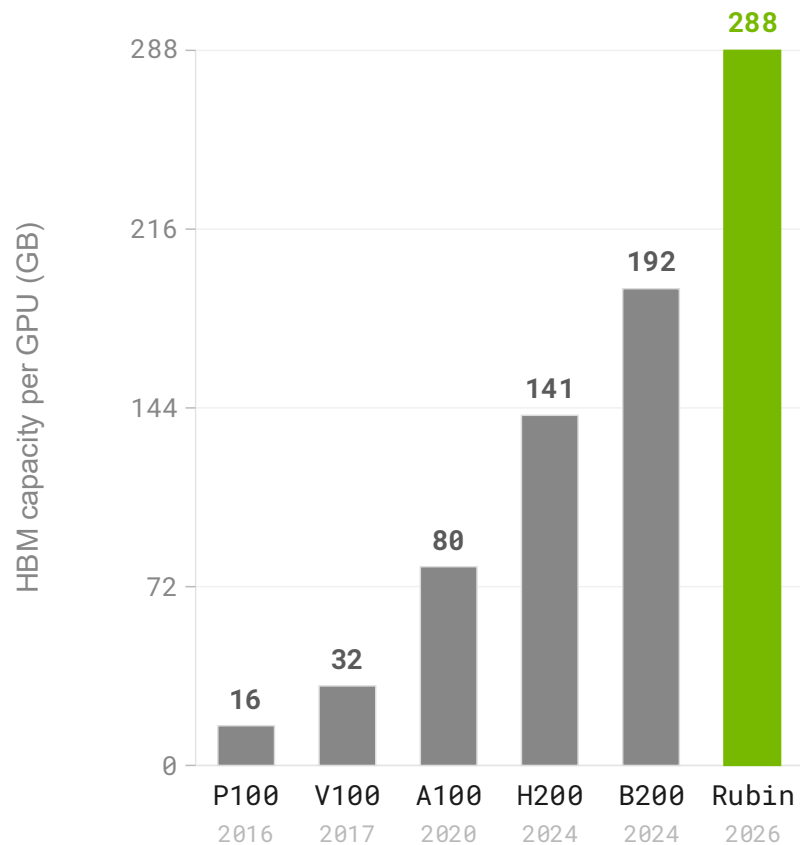
03 CHALLENGE

Engineering complexity

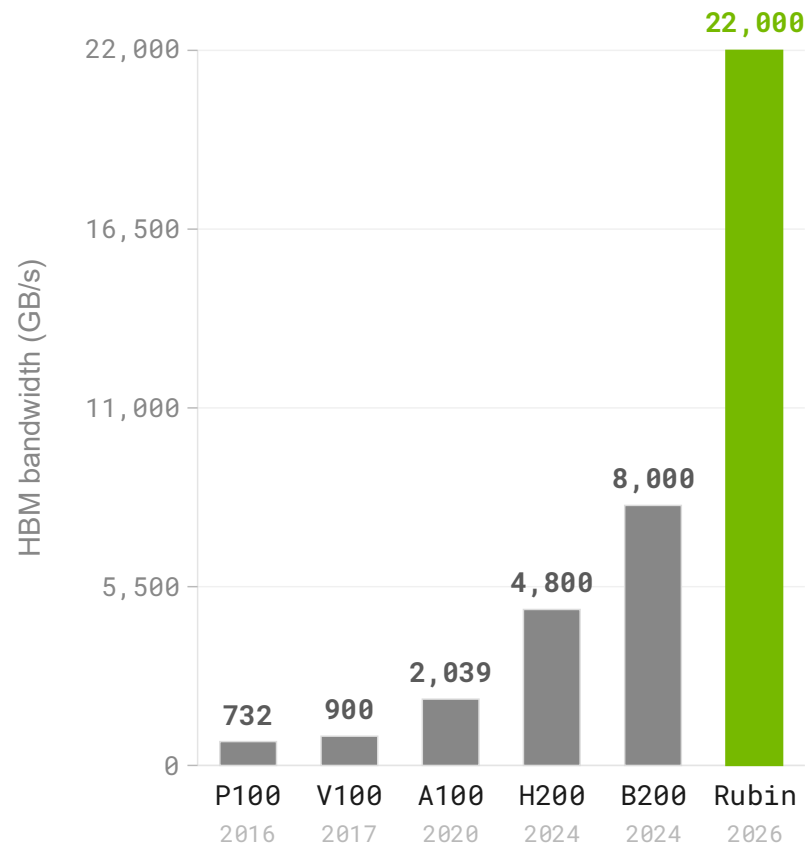
Every engine rebuilt scans, joins and shuffles from scratch with no shared GPU substrate.

Larger and Faster HBM

GPU MEMORY CAPACITY



HBM BANDWIDTH



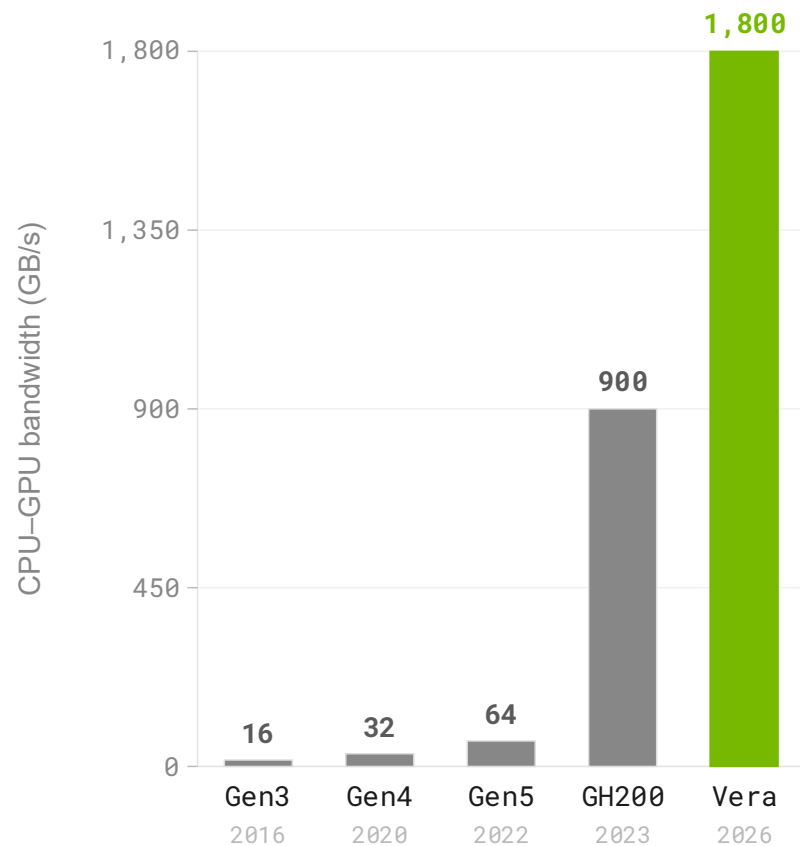
THE HEADROOM

288 GB at 22 TB/s

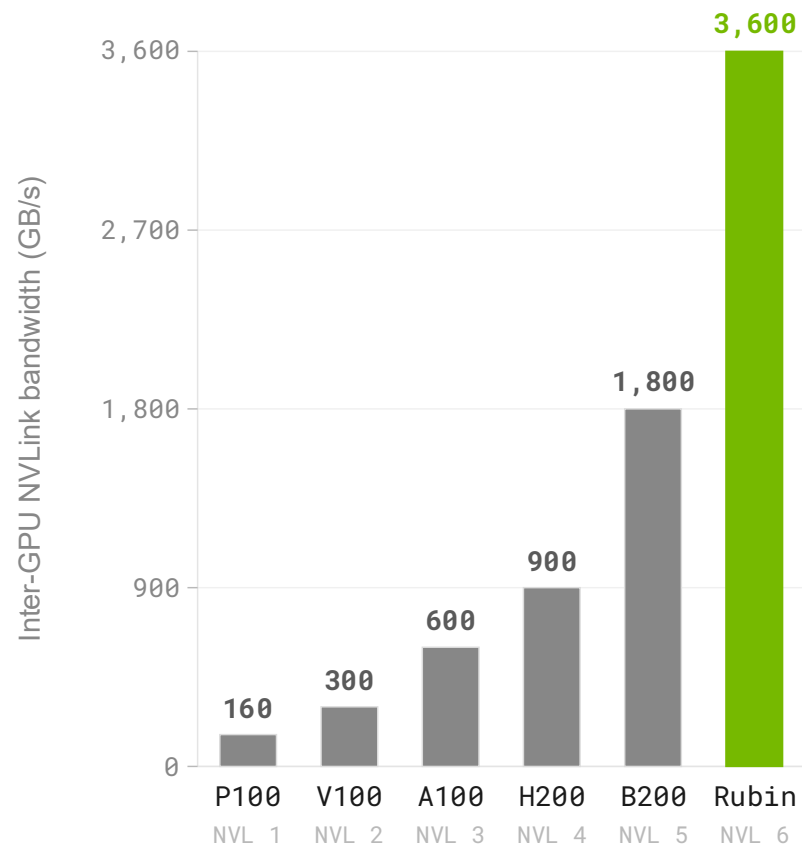
A decade ago a working set had to fit in 16 GB. Rubin carries 18x the capacity and 30x the bandwidth, so hot data now lives on the GPU.

Faster Interconnect

CPU-TO-GPU BANDWIDTH



INTER-GPU NVLINK BANDWIDTH



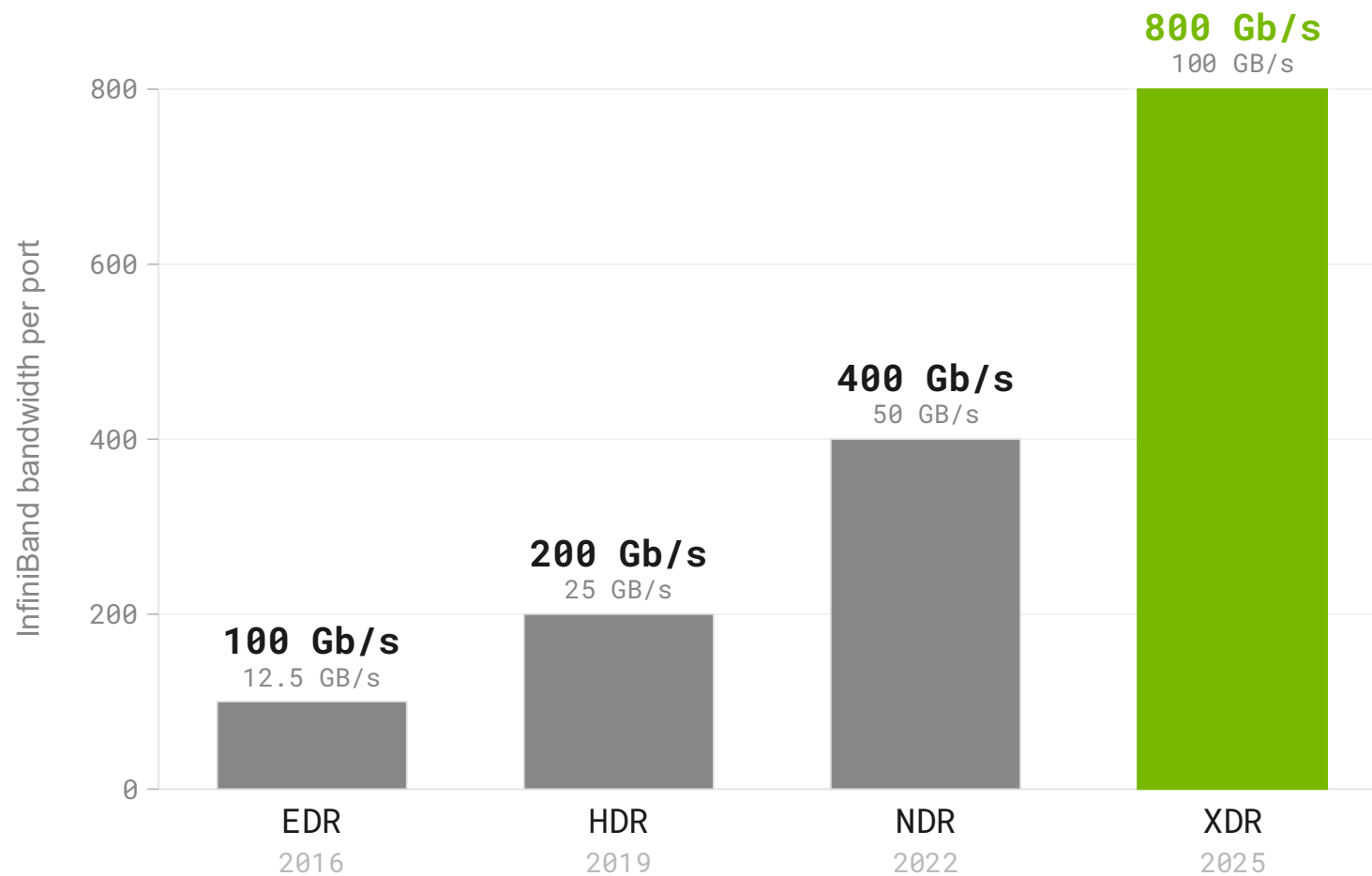
THE BYPASS

NVLink-C2C at 1.8 TB/s

Coherent NVLink-C2C replaces PCIe between CPU and GPU, and NVLink 6 pushes 3.6 TB/s between GPUs — host memory becomes a usable tier.

Faster Network Fabric

INFINIBAND GENERATIONS



THE SHUFFLE PATH

800 Gb/s per XDR port

InfiniBand has doubled every generation. An 8-port node now moves 800 GB/s to the fabric — the same order as a single GPU's local memory a few years ago.

WHY IT MATTERS

Distributed joins and shuffles stop being network-bound, so a cluster of GPUs behaves like one large machine.

Four Engine Eras



MapD and the In-Memory Pioneers

ARCHITECTURE (2013)

Massively parallel database leveraging columnar execution, LLVM JIT query compilation, and immense raw memory bandwidth.

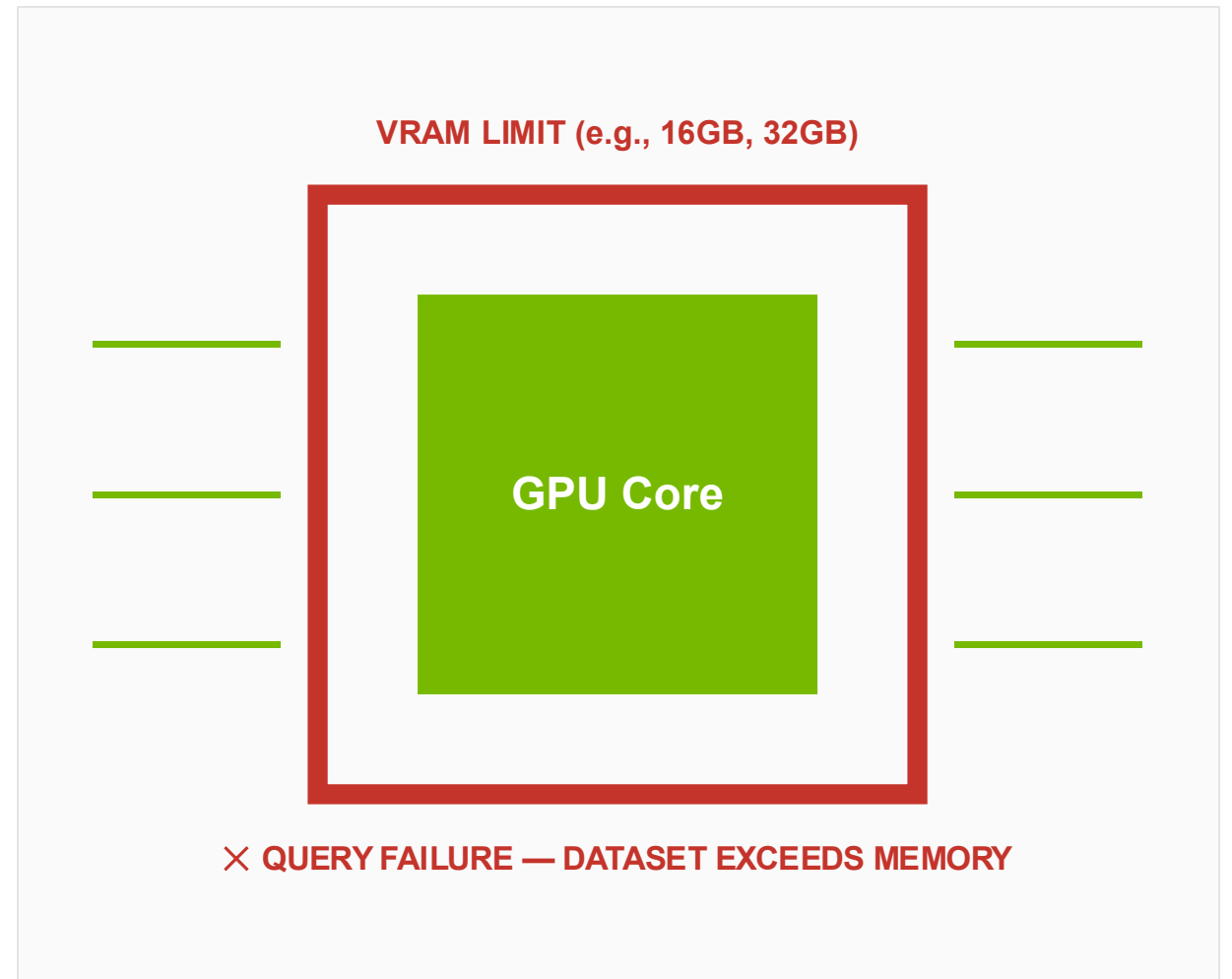
CAPABILITY

Interactive, real-time visual analytics scanning billions of rows in milliseconds.

THE CONSTRAINT: THE VRAM CLIFF

Strictly device-bound. The architecture hit a brick wall the moment a dataset exceeded physical GPU memory limits, resulting in immediate query failure.

BASE SYSTEM SCHEMATIC 1 (THE VRAM BOX)



BlazingSQL and Out-of-Core Asynchronous Execution

ARCHITECTURE

An accelerator-native, distributed query engine built from the ground up in a modular fashion to use NVIDIA cuDF for compute primitives and Apache Calcite for its query planner.

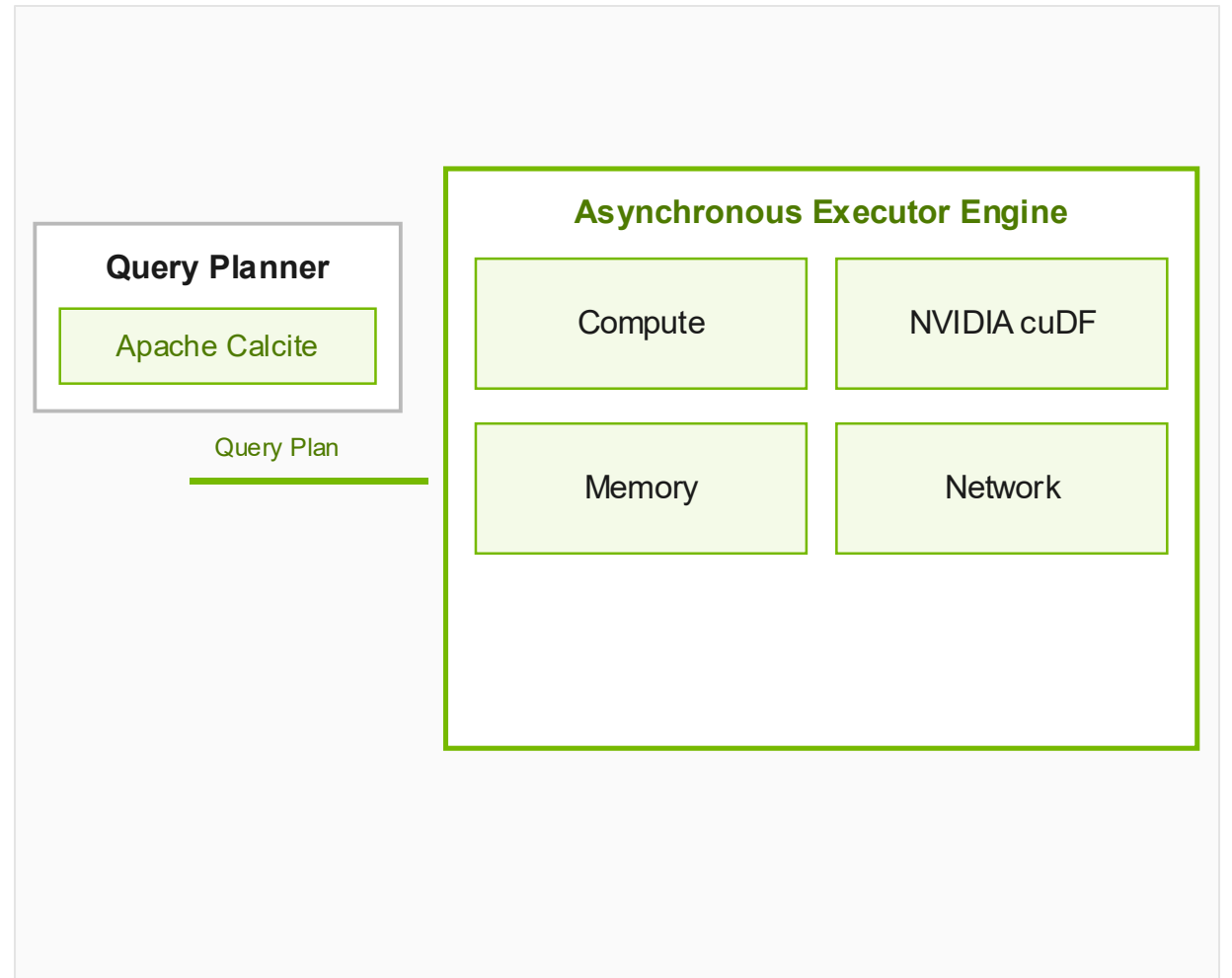
CAPABILITY: TRANSPARENT DATA SPILLING

Gracefully orchestrates 'Batch Holders' across host memory and storage without user intervention.

THE CONSTRAINT: BOTTLENECKED BY SCANS

BlazingSQL lacked a cache layer, and top speed was bottlenecked by how fast data could be read from storage.

SYSTEM SCHEMATIC 2 (COMPOSABLE ASYNCHRONOUS EXECUTOR)



NVIDIA cuDF for Apache Spark and the Retrofitting Friction

ARCHITECTURE

Grafting GPU acceleration onto a legacy CPU-first ecosystem by translating logical query plans directly into GPU-accelerated libcudf operators.

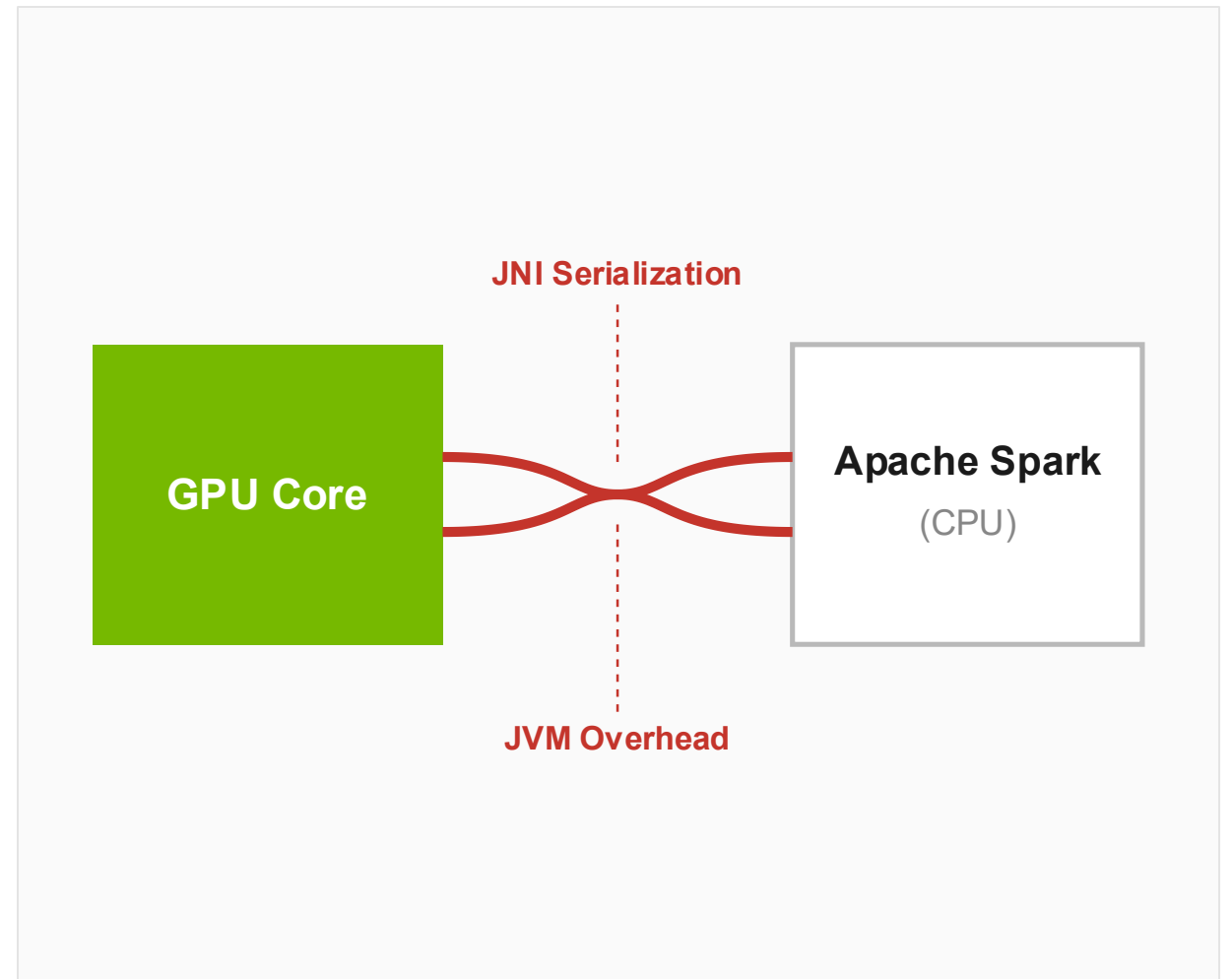
CAPABILITY

Extended the lifespan and speed of existing Spark deployments without altering the end-user interface.

THE CONSTRAINT: HITTING THE CPU WALL

Fundamentally bottlenecked by legacy architecture. Suffered from severe JVM memory management overhead and was unable to exploit high-speed NVLink or GPU-direct memory tiers.

SYSTEM SCHEMATIC 3 (THE CHOKED BRIDGE)



Theseus and Out-of-Core Asynchronous Execution

ARCHITECTURE

The successor of BlazingSQL, driven by a custom memory-centric cost model, a fixed-size buffer pool and an async IO layer to achieve better performance.

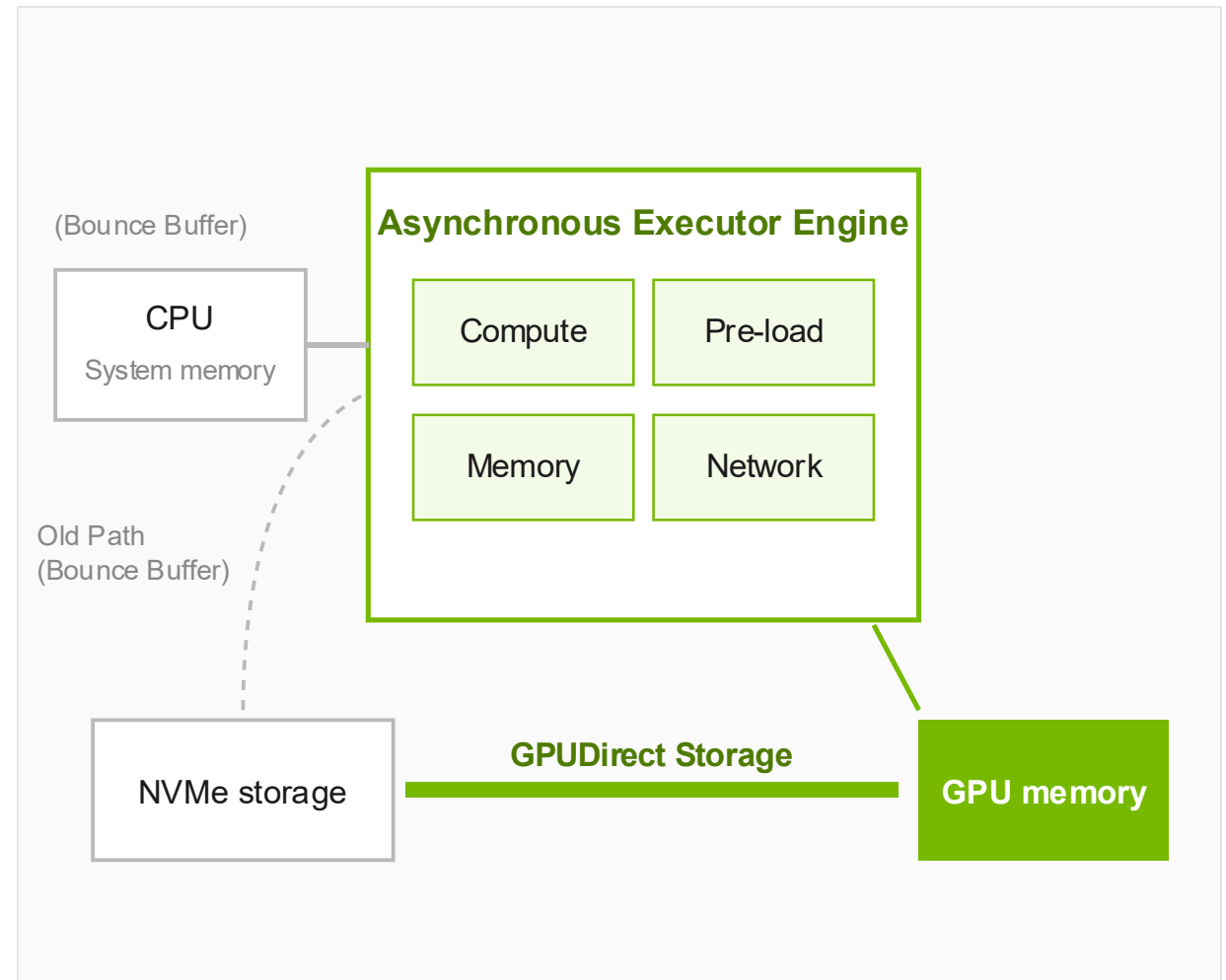
CAPABILITY: ASYNC DATA LOADING

Accelerated data scanning by an optimized, asynchronous IO layer which supports GPUDirect Storage.

SCALE

Executes 100 TB datasets efficiently across just 2 DGX nodes, structurally bypassing CPU main memory bounce buffers via GPUDirect Storage.

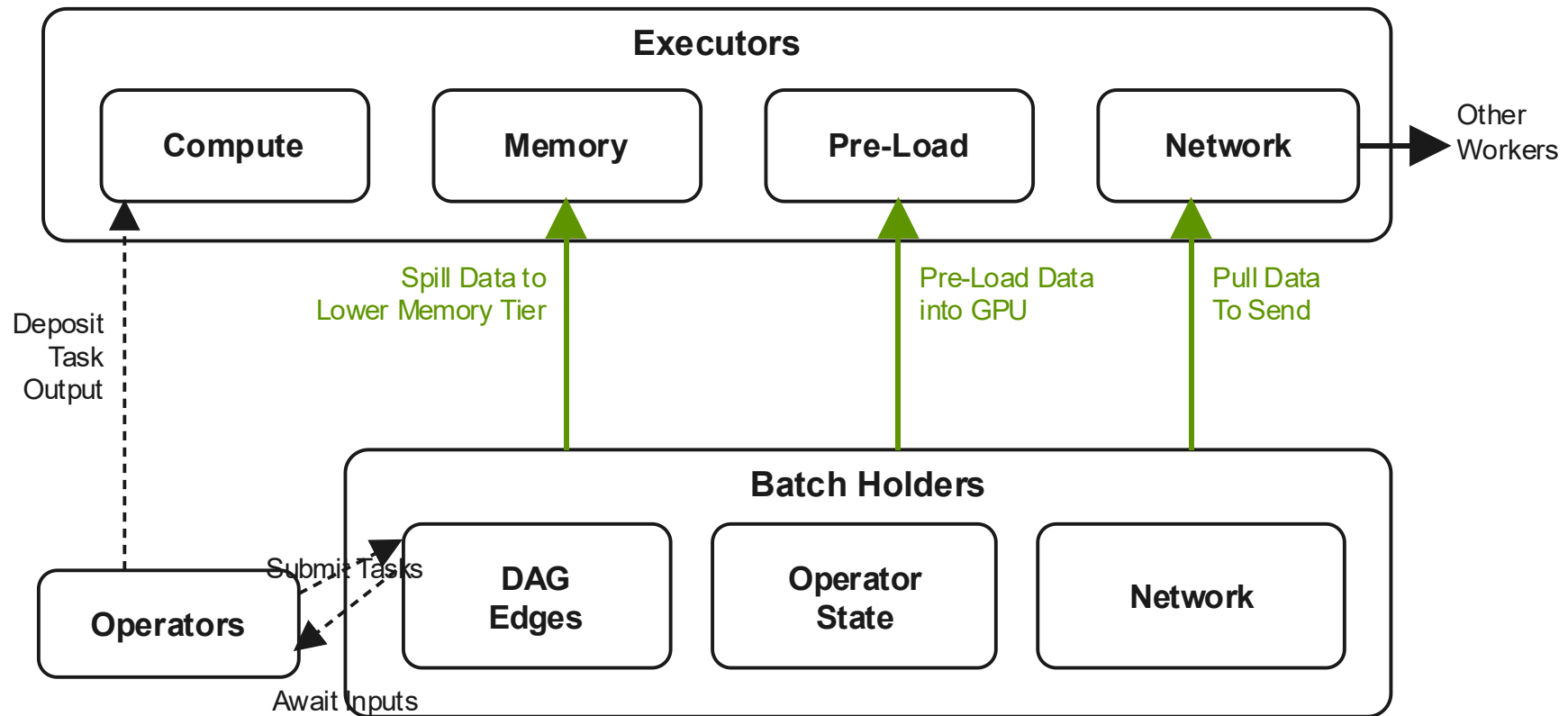
SYSTEM SCHEMATIC 4 (THE MEMORY BYPASS)



Task Execution and Batch Holder Memory Management

Executors run compute, memory, pre-load and network as parallel roles; Batch Holders track task state and spill it across memory tiers without stalling the pipeline

SYSTEM SCHEMATIC — EXECUTORS AND BATCH HOLDERS

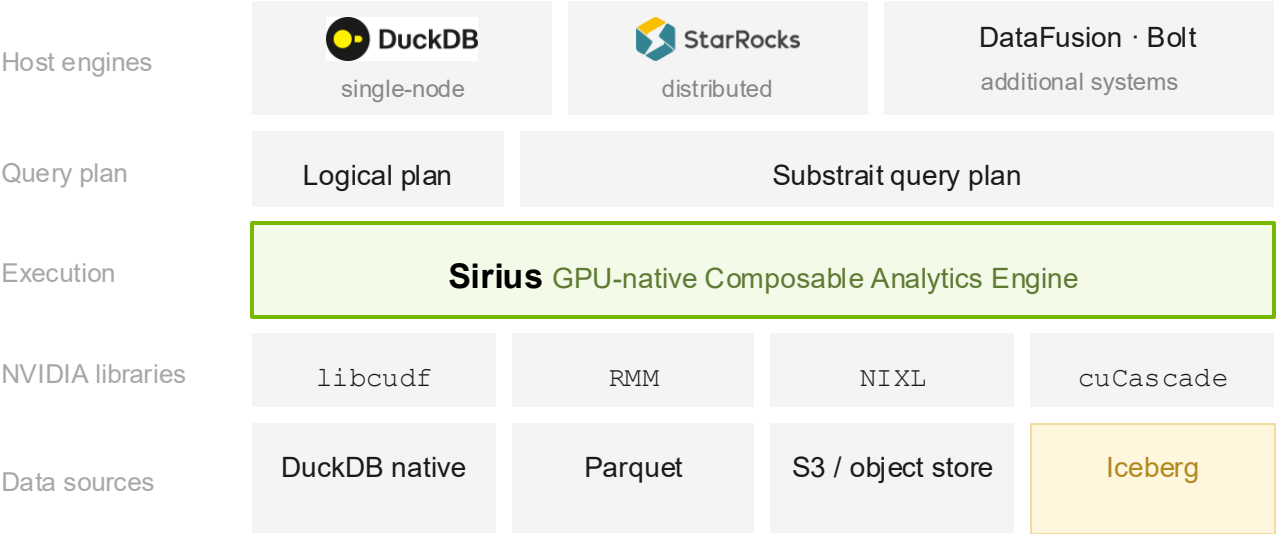


Sirius - Composable GPU Execution for Drop In Acceleration



Sirius — A GPU-Native, Composable SQL Engine

THE SIRIUS STACK



GitHub repo:
github.com/sirius-db/sirius



DESIGN PRINCIPLE 01

GPU-Native Execution

The GPU is the primary execution engine — buffer manager and query engine designed for it from the start, not a CPU engine retrofitted with kernels.

DESIGN PRINCIPLE 02

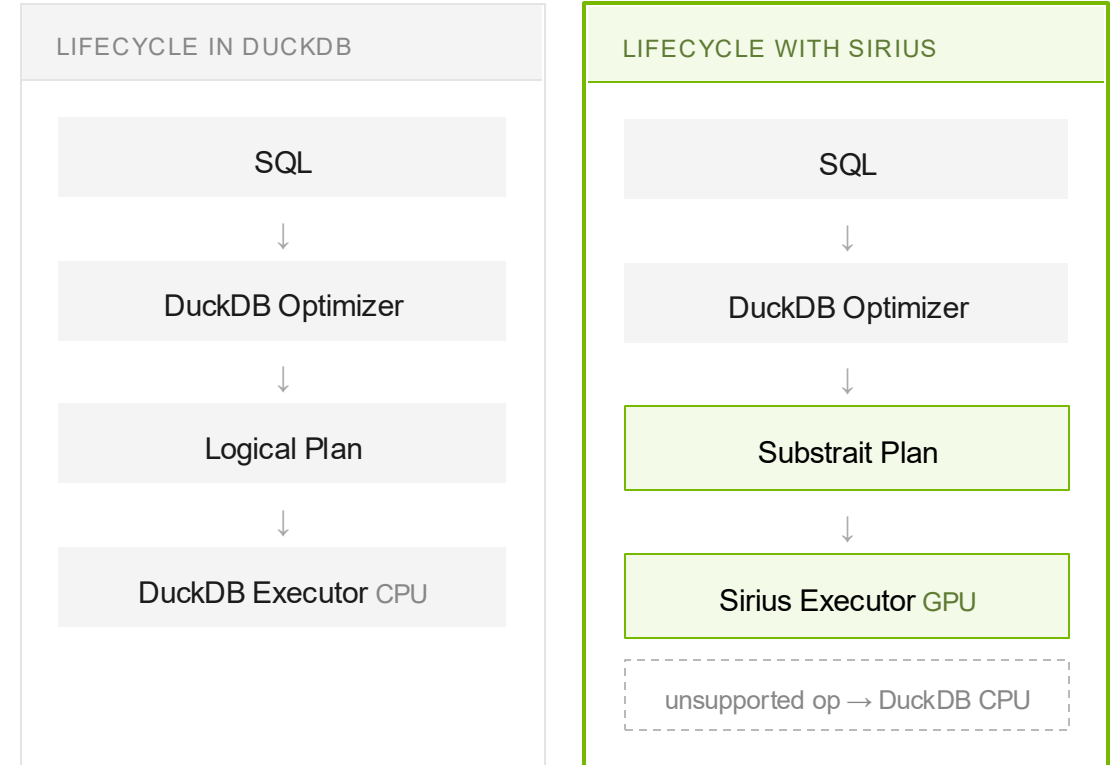
MICE — Modular, Interoperable, Composable, Extensible

Drop-in acceleration for a host system with minimal modification to it. Built on CUDA-X libraries; new operators plug in without touching the engine core.

Drop-In Acceleration, With a Safety Net

Shipped as a DuckDB extension — same SQL, same client, same files

- No modification to the DuckDB codebase or its user interface
- Unsupported operators **fall back to host CPU execution** automatically — a query never fails for want of a GPU kernel
- Reads what you already have: DuckDB native files, Parquet via the GPU reader, S3 and object stores, Iceberg tables
- Extensible — new database operators drop in without engine surgery



Scaling Past the Device

GPU memory is 5-10x smaller than host memory, so spilling and scale-out are first-class, not afterthoughts

LARGER THAN GPU MEMORY

Asynchronous Spilling with cuCascade

A data repository connects Sirius pipelines and asynchronously downgrades tiers — GPU to host memory to storage — with custom converters between representations. Working sets are not capped by HBM.

SINGLE NODE

Multi-GPU Execution

Partitioning and inter-GPU exchange over NVLink let one node put every GPU behind a single query, with the buffer pool managed across devices.

MULTI NODE

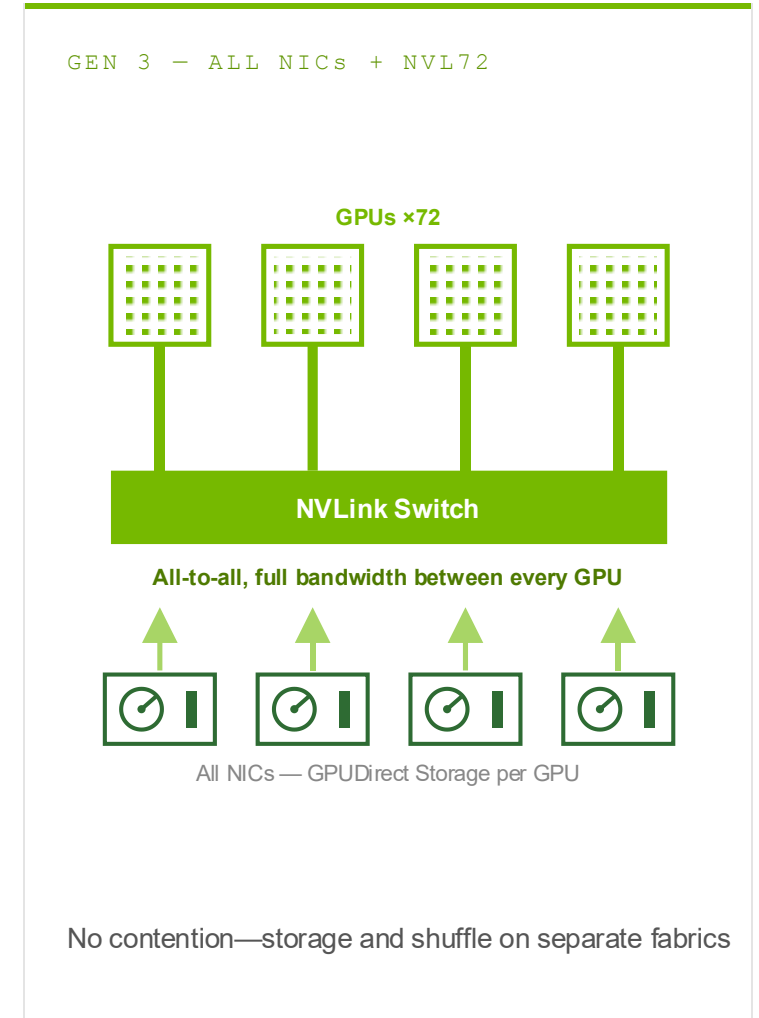
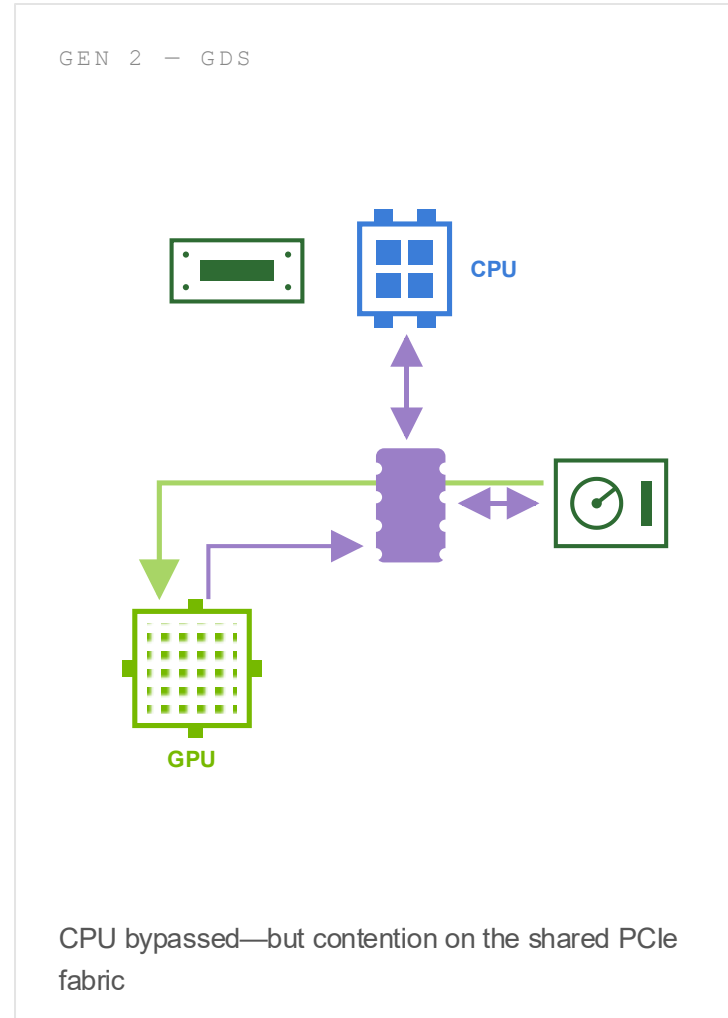
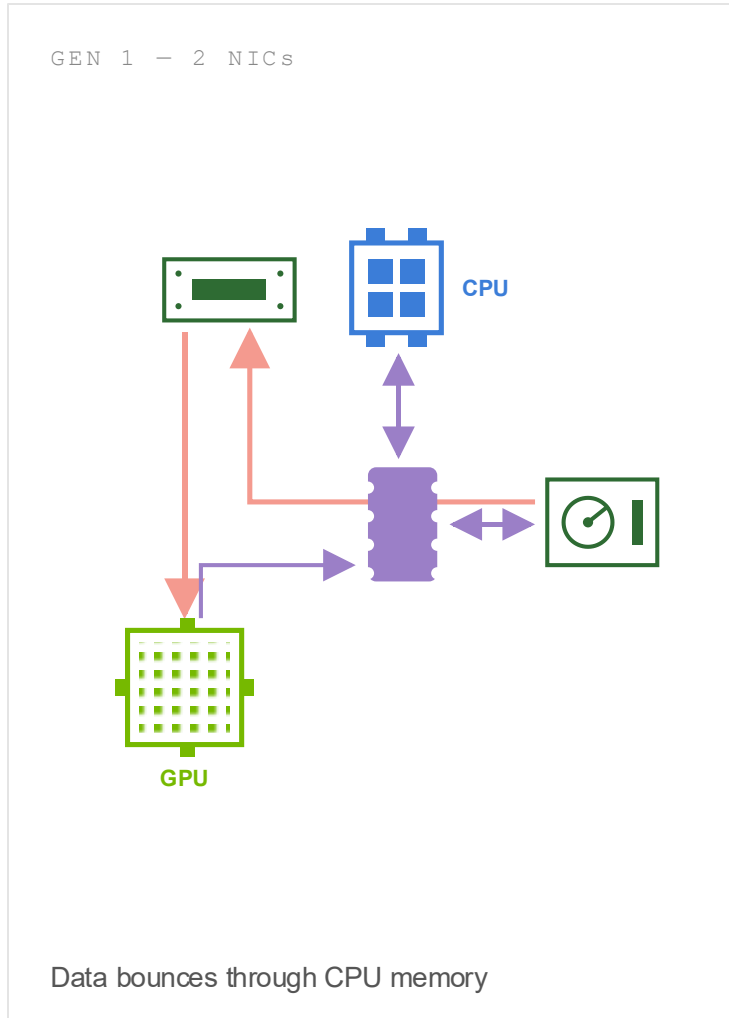


Distributed Acceleration

The same Substrait interface plugs Sirius under a distributed host engine, so cluster-scale queries pick up GPU execution without changing the deployment.

■ Available today ■ On roadmap

Reference Design Evolution



System memory / storage

CPU

PCIe switch

GPU

Bounce buffer

PCIe

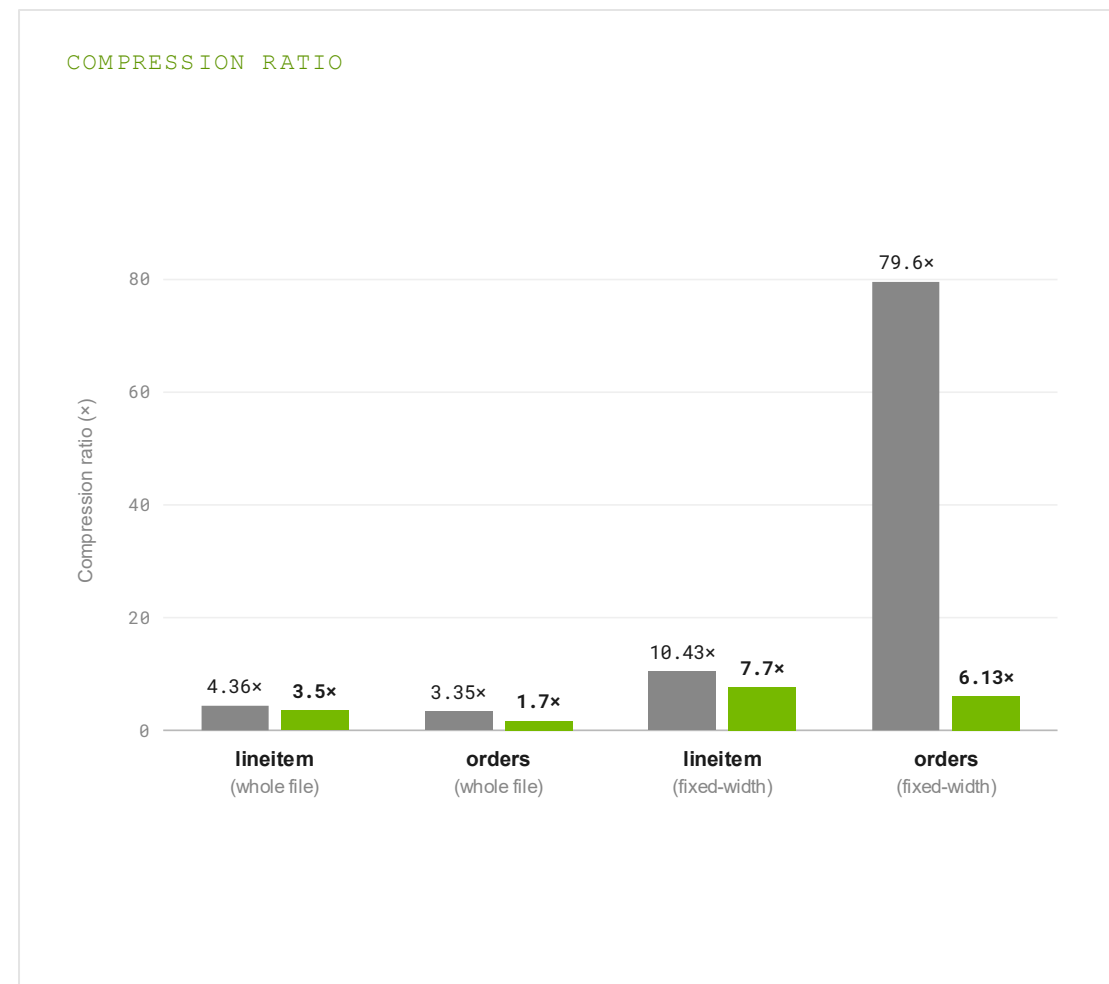
GPUDirect Storage

NVLink

Can You Utilize All This I/O? Yes

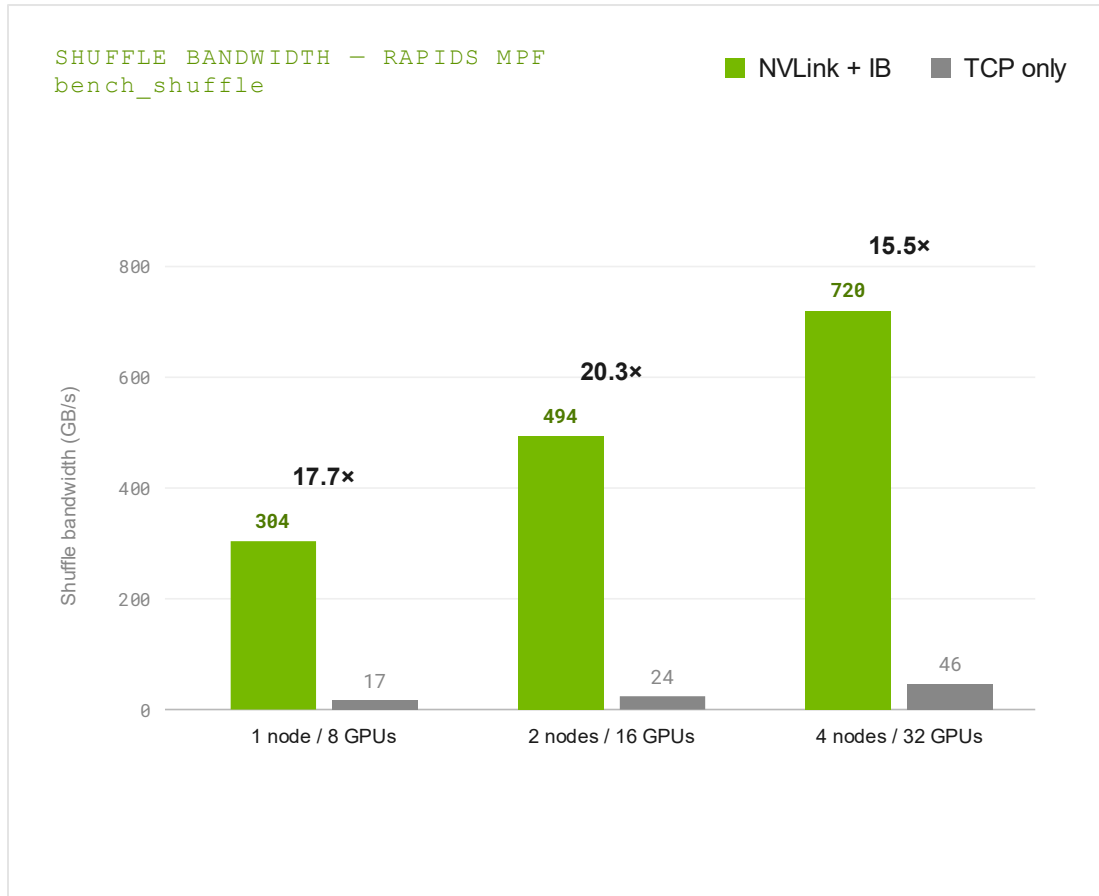
Pure GPU compute throughput on a GB300—I/O and metadata parsing excluded from the timed region

■ Parquet ■ Simpatico



Fixed-width decode hits **2.2 TB/s** on a single GPU—decompression keeps pace with everything the fabric can deliver

Reduce Shuffle Times on Accelerated Fabrics



15–20× faster exchange phase than a non-accelerated network—shuffle time is bytes / bandwidth

Shuffle measured in isolation; query-level gains depend on the plan's exchange share. Per GPU: ~38 GB/s vs ~2 GB/s.

SB 14/34 — BREAKER

What Happens When We Modernize Existing Engines

Modernizing HeavyDB —1.7× Faster in Eight Weeks

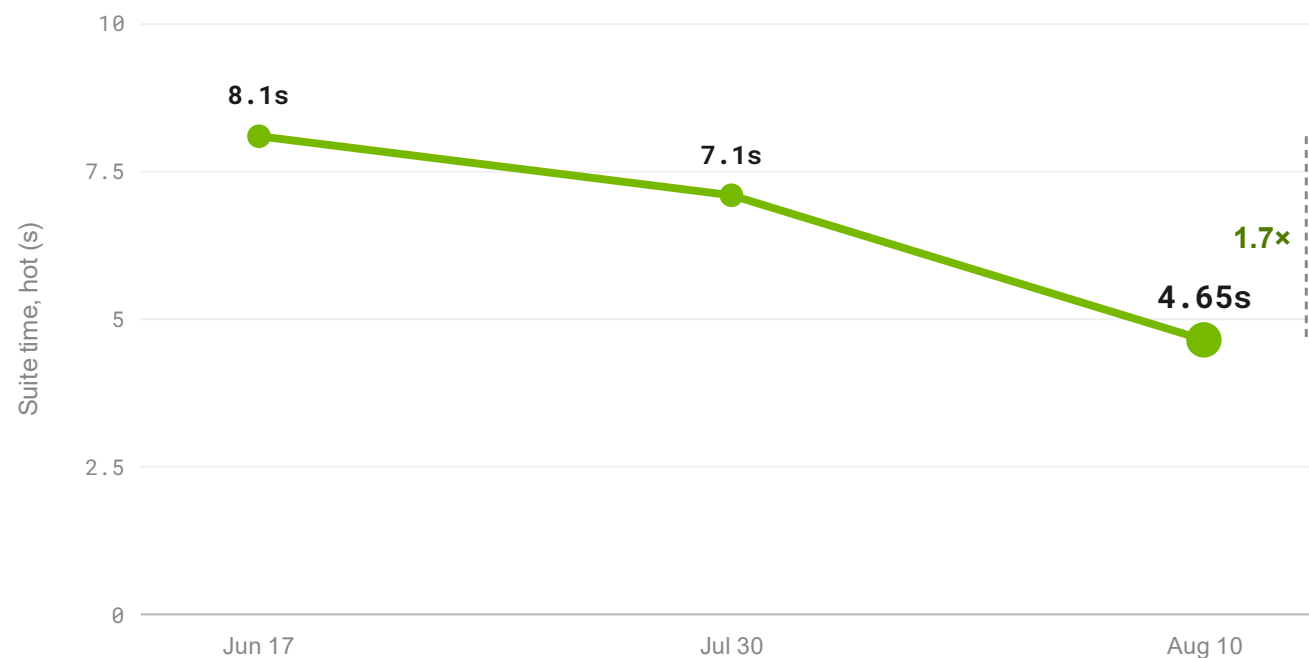
4.65s

TPC-H SF 1,000, best hot total

8× H100, SINGLE NODE

Bigger memory and faster GPUs set the floor—
eight weeks of modernization took another 1.7× off,
same box

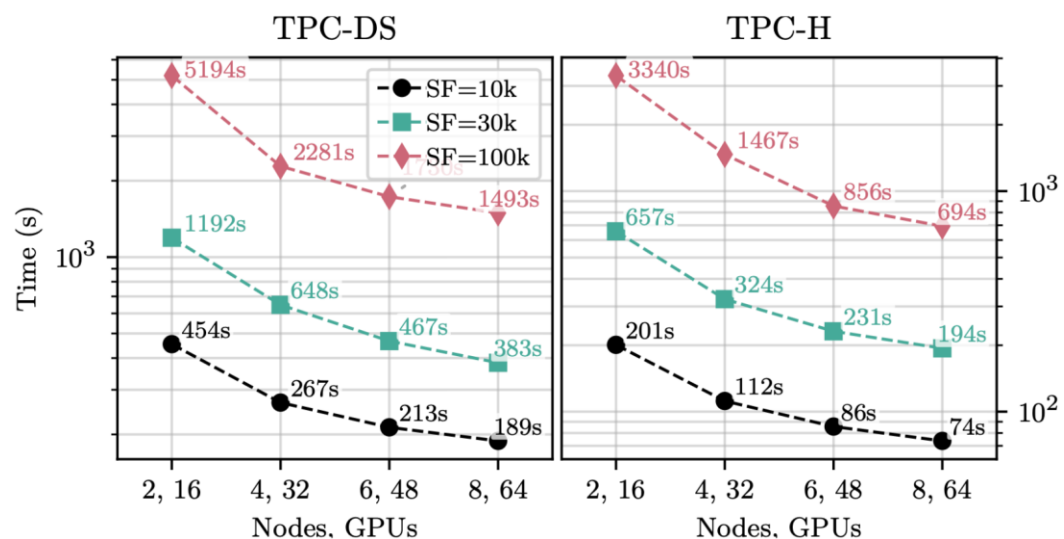
MODERNIZATION TRAJECTORY — BEST HOT TOTAL, 8× H100



Theseus Performance at Scale

TPC Benchmarks On-Premise and in the Cloud

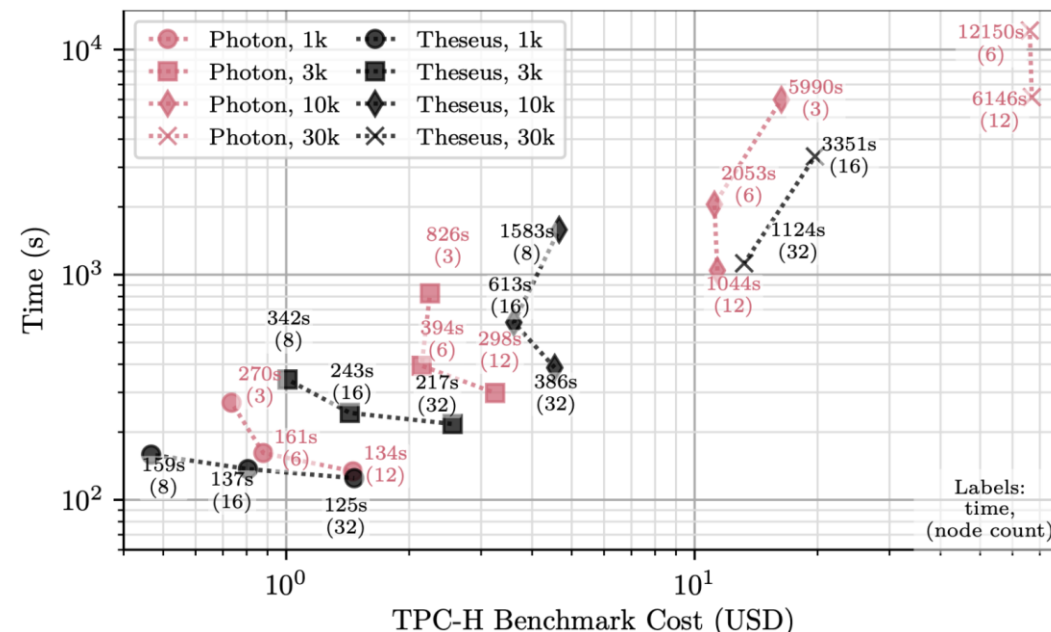
On-Premise



COMPUTE DGX A100 — Xeon Platinum 8380 · 4 TiB · 8× A100 80GB · 200Gb/s InfiniBand

STORAGE Parquet on a distributed cluster with GPUDirect Storage and RDMA

Cloud



THESEUS AWS g6.4xlarge — 16 vCPUs · 64 GiB · 1× L4 24 GiB GPU · 25 Gbps

PHOTON AWS r7gd.12xlarge — 48 vCPUs · 384 GiB · 22.5 Gbps

STORAGE Parquet on AWS S3

Source for both: Theseus paper — arxiv.org/pdf/2508.05029

Apache Spark GPU Streaming Pipeline Acceleration

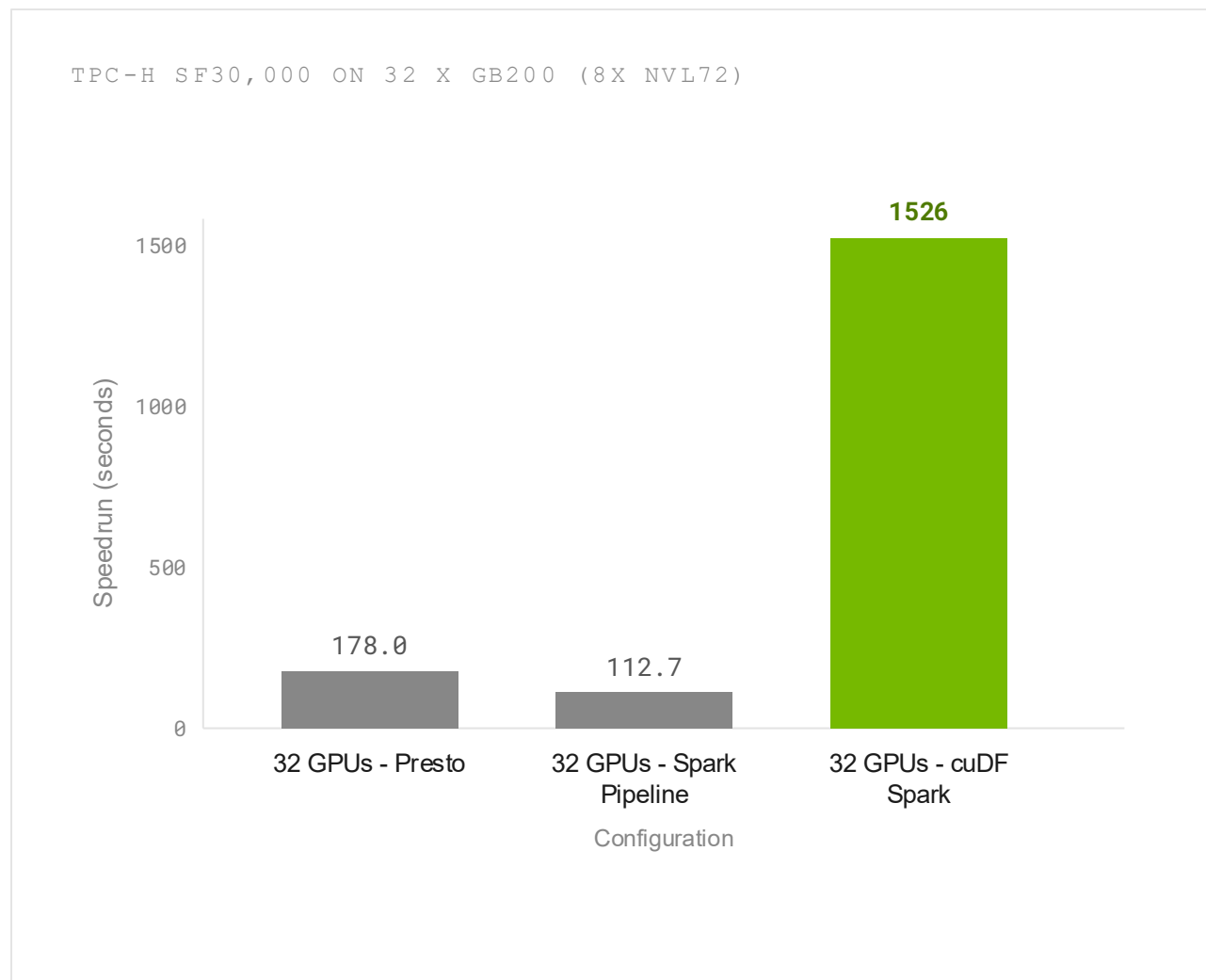
- **Concurrent fragments execution**

- **Streaming GPU exchange**

- **Velox-cuDF hardening**

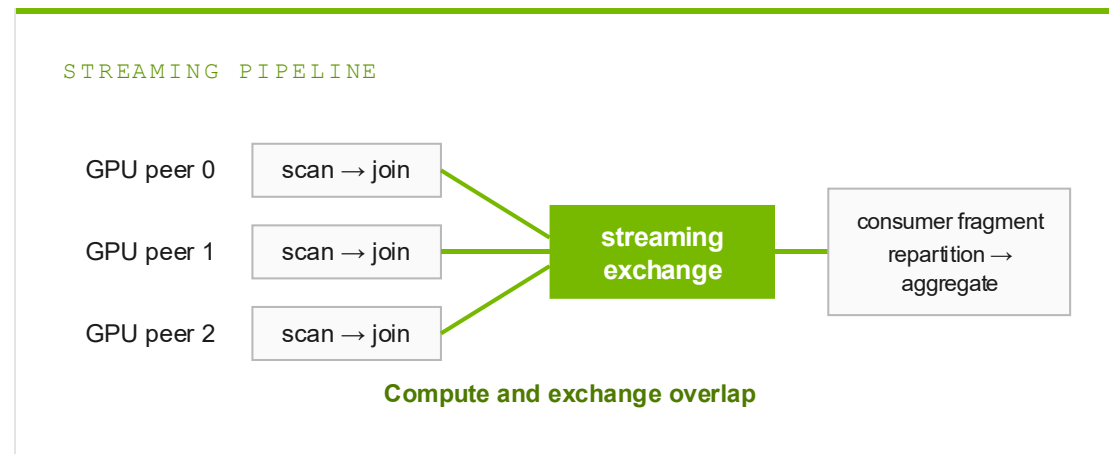
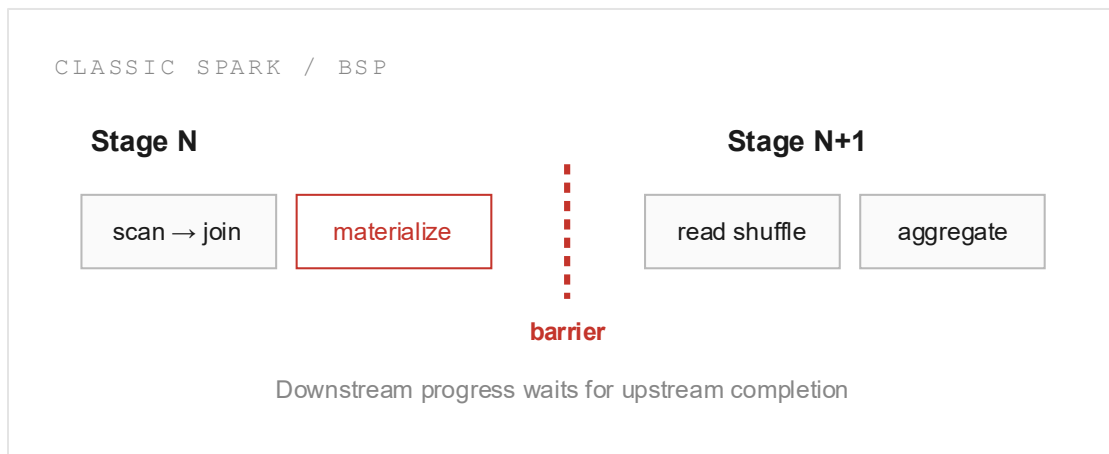
- **Architecture and OSS Integration**

Built on Gluten-Velox-cuDF with NVIDIA GPUs,
experimenting with Spark's SPIP Real-time mode, Laptop
mode



Break Stage Barriers – Parallelizing Pipelined Fragments

The architectural shift: fragments exchange columnar batches while producers and consumers run concurrently.



WHAT CHANGES TECHNICALLY

- ExchangeSpec decomposes the plan into producer/consumer fragments.
- Each GPU executor is a peer; native Velox tasks exchange through UCX, with cuda_ipc on the same host when available.
- Distribution semantics remain explicit: HASH, RANGE, SINGLE, ROUND_ROBIN and BROADCAST.

Faster and Cheaper in the Cloud

TPC-H SF 1,000 on AWS, on-demand—GPU against CPU. Lower left is better



■ Spark cuDF Pipelined (g7, RTX PRO 4500)

■ CPU—Amazon EMR Spark (Graviton)

~14x faster · ~17x cheaper per run



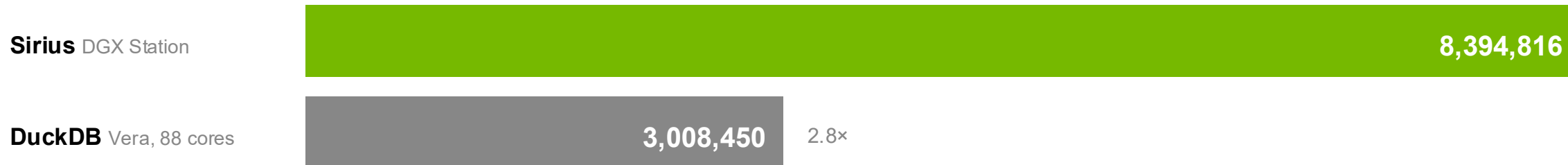
SB 23/34 — SIRIUS

What Happens When You Start With Blackwell—and Have a Decade of Experience

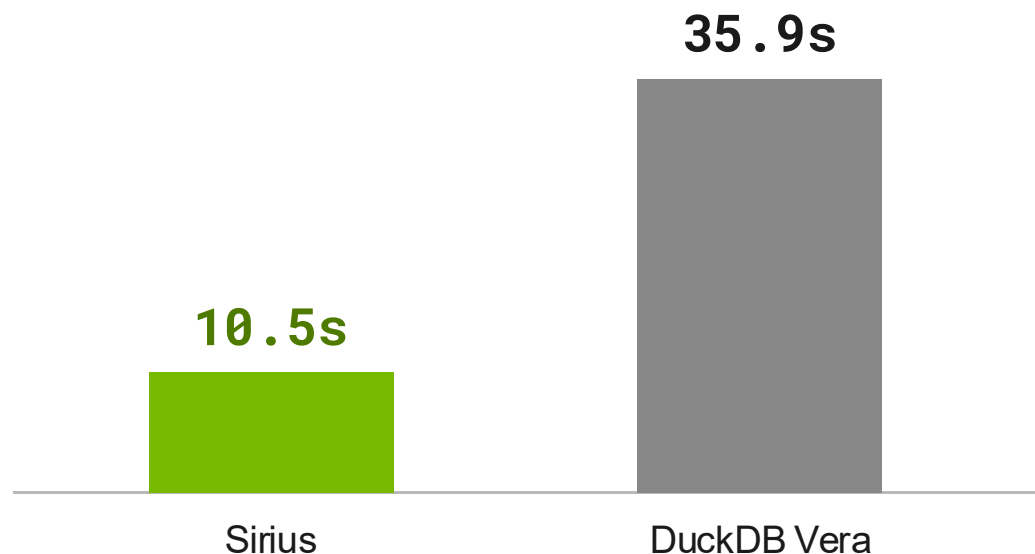
Sirius Runs the Complete TPC-H Suite

Power and throughput runs, refresh functions, full ACID—end to end

TPC-H QphH@SIZE, SF 1,000 — HIGHER IS BETTER



POWER RUN, 22 QUERIES — RUNTIME (S), LOWER IS BETTER

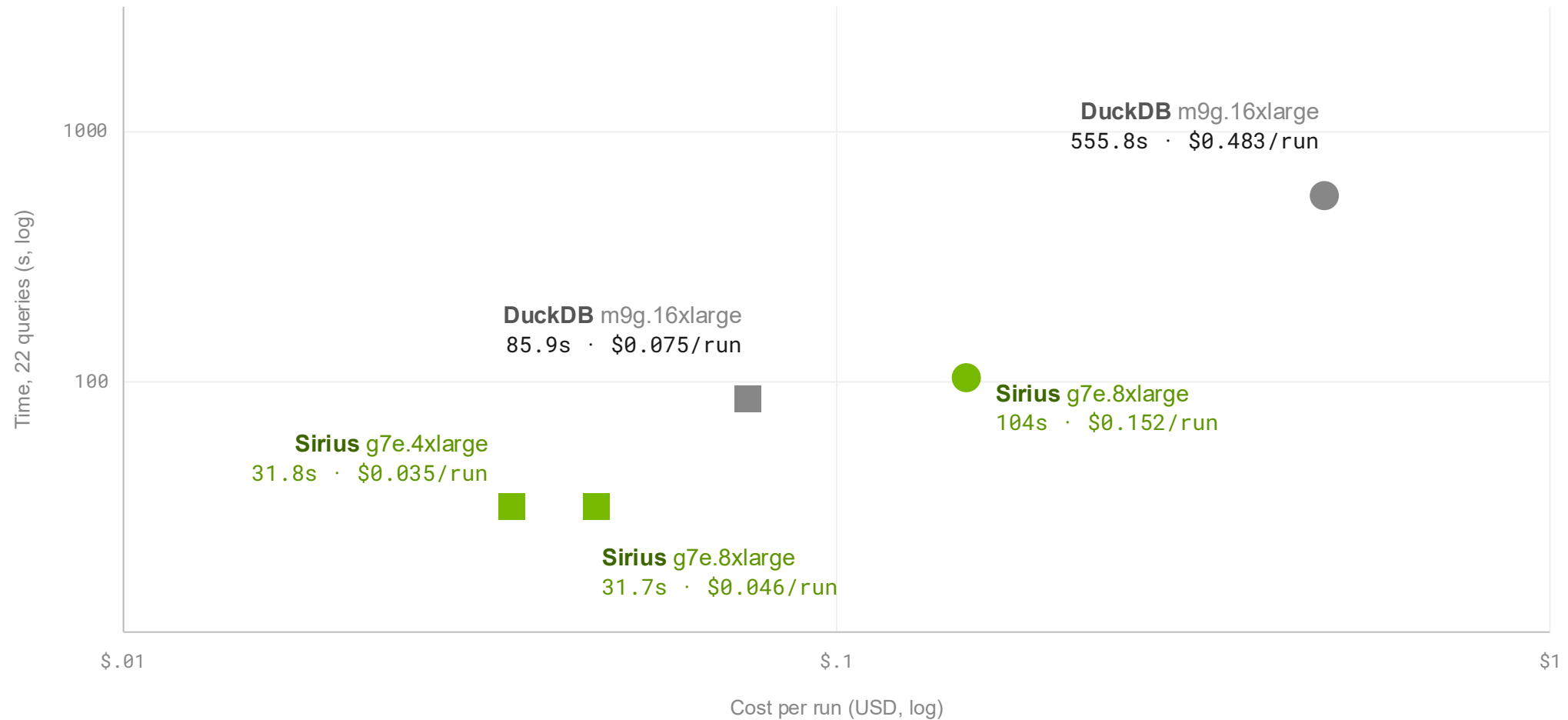


5.9s

Fastest TPC-H runtime currently achievable on the DGX Station—standalone power run, without refresh functions or the full-suite protocol

Faster and Cheaper in the Cloud

TPC-H SF 1,000 on AWS, on-demand—GPU against CPU. Lower left is better



■ Sirius (g7e, RTX PRO 6000)

■ CPU—DuckDB v1.5.5 (Graviton5)

□ Hot

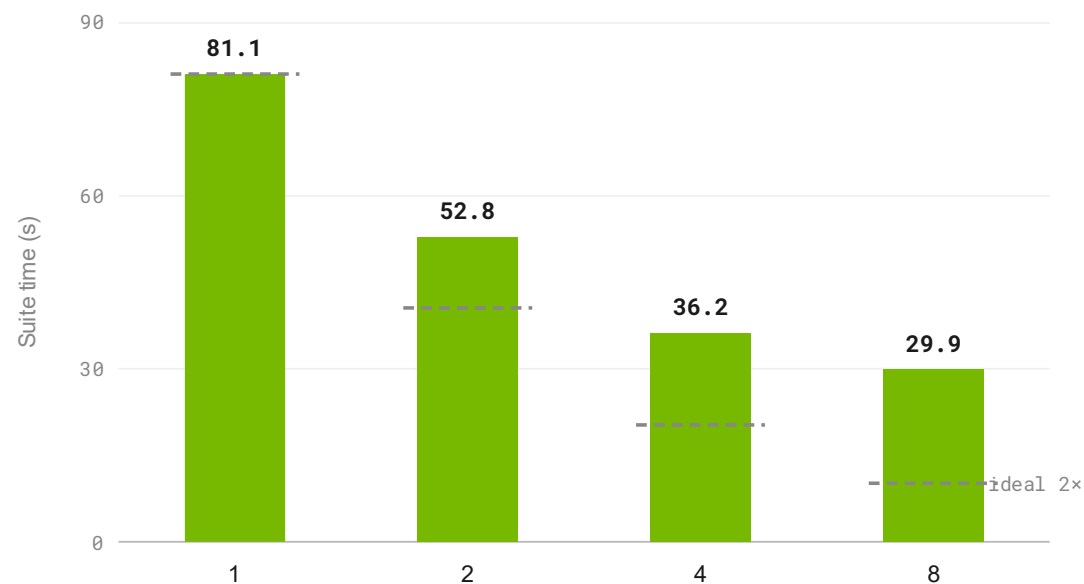
○ Cold



Sirius + StarRocks Prototype

Scaling out across GPUs, scaling up to 3× the data

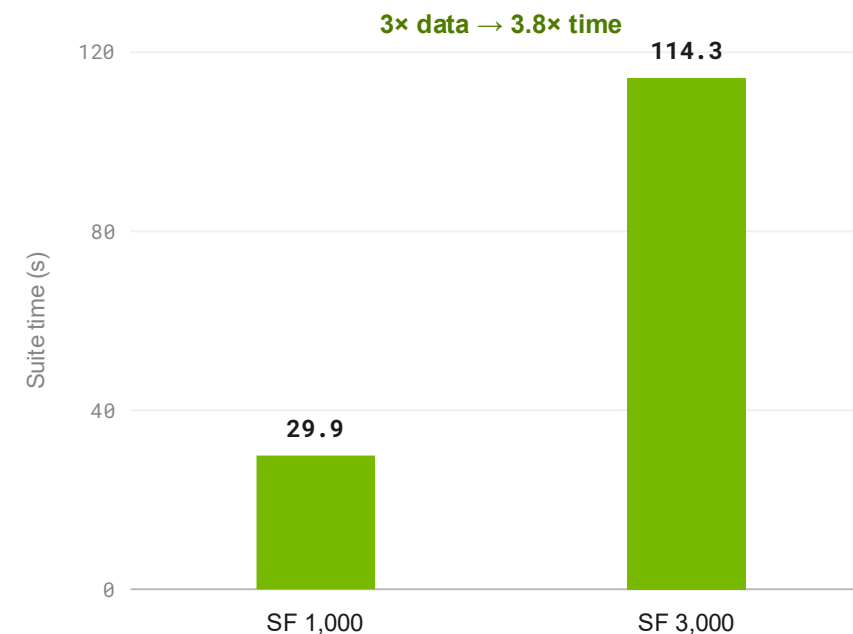
TPC-H SF 1,000 — GPU SCALE-OUT



GB200 compute nodes — 22 warm medians

Near-linear early (1.54×, 1.46× per doubling), flattening at 8 as per-query fixed costs dominate the short queries

DATASET SCALE-UP — FIXED 8 GB200

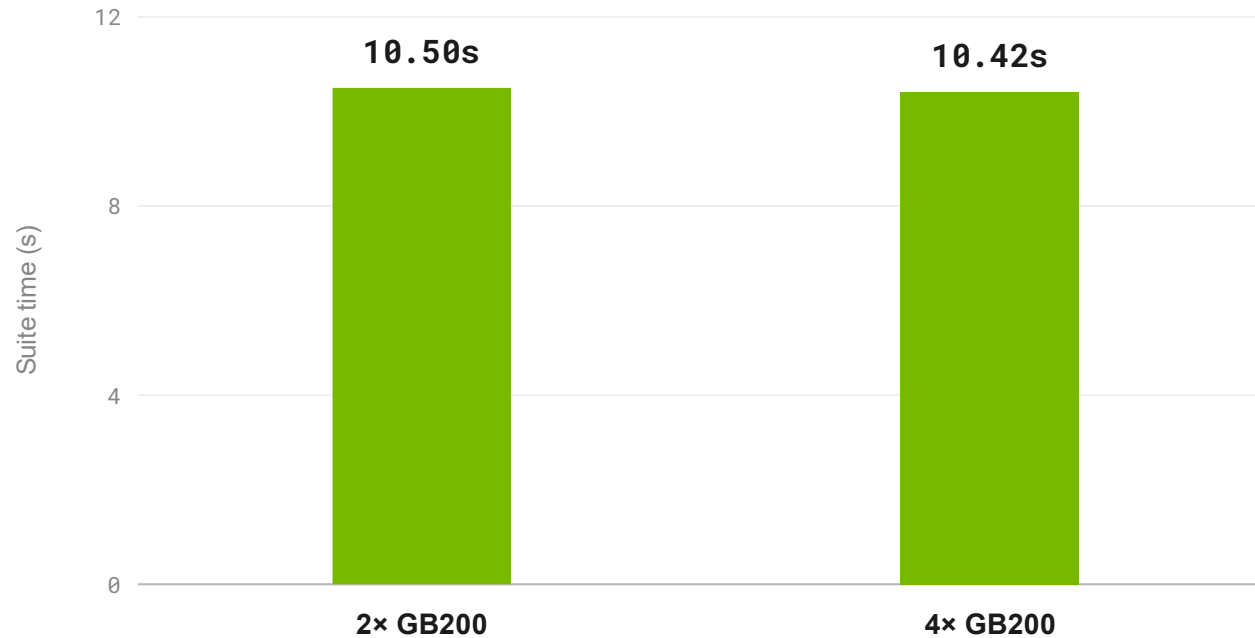


Full 22-query suite, warm — same 8 GB200 nodes

Near-proportional runtime growth past device memory—tiered (pinned host) memory keeps the full suite finishing at 3× the data

Sirius on NVL4: SF 1,000 in Ten Seconds

SUITE RUNTIME — HOT



TPC-H SF 1,000 suite, hot — NVL4, Aug 2026 nightly

10.4s

Hot suite on 4x GB200—and 10.5s on just 2. The suite is no longer compute-bound at this scale

WHAT IT SHOWS

- 2→4 GPUs buys almost nothing hot: NVLink already keeps two GPUs fed
- Early prototype—cold and lukewarm paths aren't dialed in yet

PROTOTYPE

Sirius + cuVS — Vector Search on the GPU

Extensibility in action — cuVS plugs in behind the same SQL surface, no engine surgery

THE INTERFACE — NOTHING NEW TO LEARN

```
-- build an IVF-Flat index on an embedding column
CREATE INDEX docs_idx ON docs USING IVFFLAT (embedding);

-- top-k search, composed with ordinary SQL
SELECT id, title
FROM docs
WHERE published >= '2026-01-01'
ORDER BY array_distance(embedding, $query)
LIMIT 10;
```

LANDING IN THE OPEN REPO

EXACT

PR #1204

Brute-Force Search (ENN)

Exact nearest neighbors as a GPU operator — the correctness baseline.

INDEX

PR #1205

IVF-Flat Index Creation

Index built on device, beside the relational data.

APPROXIMATE

PR #1206

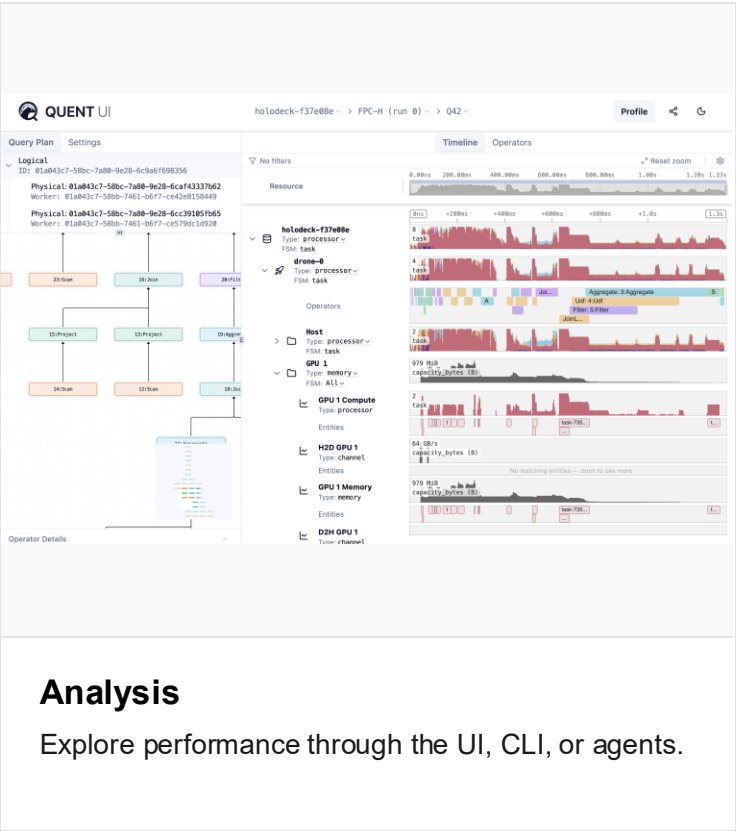
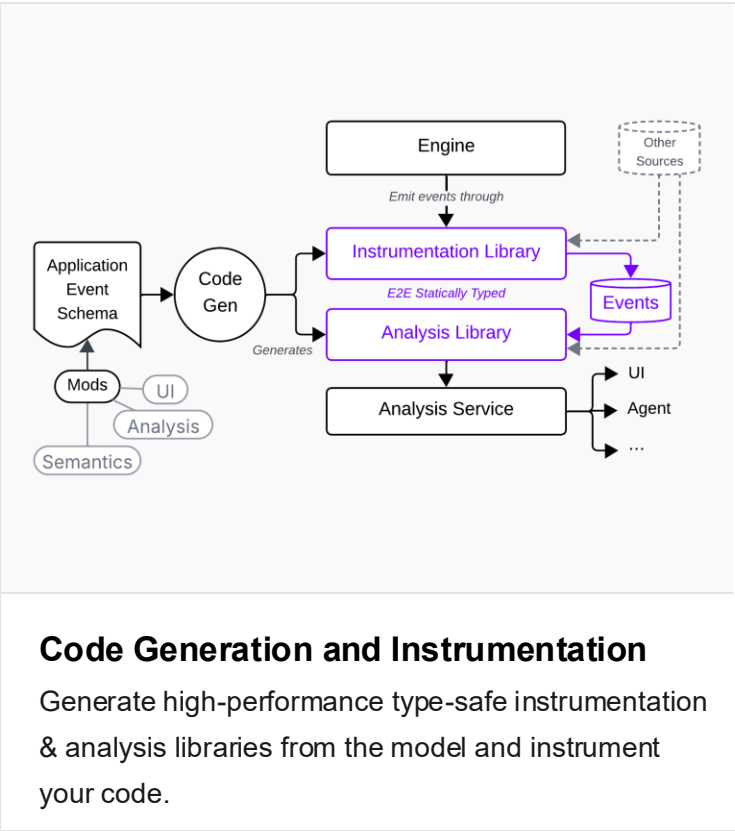
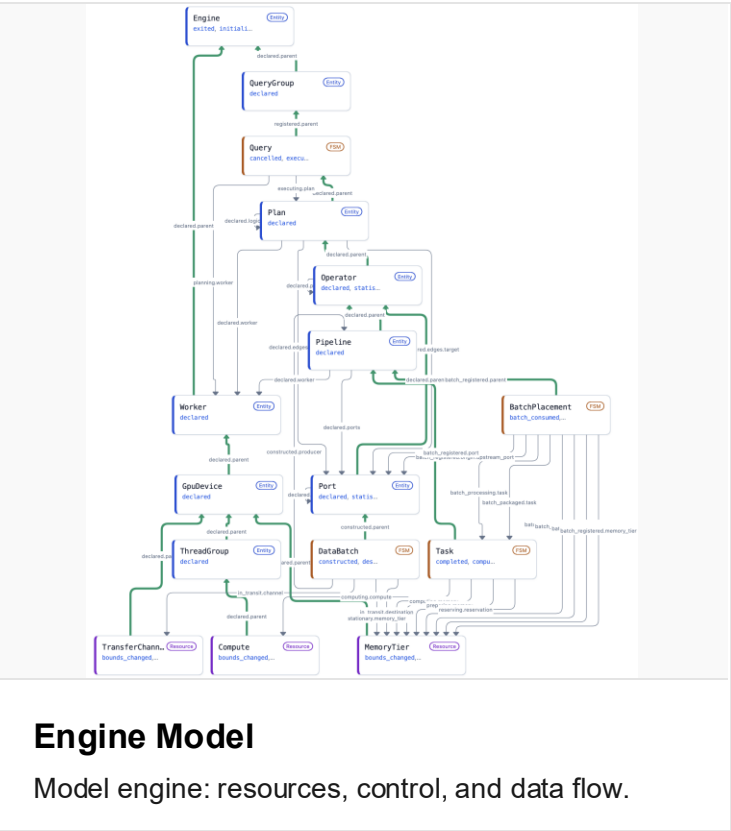
Vector Search (ANN)

Approximate top-k composed into ordinary SQL plans.



Toolkit to build application-specific performance analysis tools for accelerated data processing systems

github.com/rapidsai/quent



Work With Us

Open problems across query optimizations, out-of-core execution, kernel fusion, communication, and cross-engine composability



**GitHub
repo**

github.com/sirius-db/sirius



Open research questions

[github.com/sirius-
db/sirius/blob/dev/docs/sirius-for-
research.md](https://github.com/sirius-db/sirius/blob/dev/docs/sirius-for-research.md)



**Join our
Slack**

[Join the Sirius Slack](#)