Rethinking Analytical Processing in the GPU Era

Bobbi Yogatama*, Yifei Yang*, Kevin Kristensen, Devesh Sarda, Abigale Kim,
Adrian Cockcroft, Yu Teng, Joshua Patterson, Gregory Kimball, Wes McKinney,
Weiwei Gong, **Xiangyao Yu**

1/21/2026

Rethinking Analytical Processing in the GPU Era

Bobbi Yogatama[2*], Yifei Yang[1*], Kevin Kristensen[1], Devesh Sarda[1], Abigale Kim[1], Adrian Cockcroft[5], Yu Teng, Joshua Patterson[2], Gregory Kimball[2], Wes McKinney[4], Weiwei Gong[3], Xiangyao Yu[1]
[1]University of Wisconsin-Madison, [2]NVIDIA, [3]Oracle, [4]Posit PBC, [5]OrionX
siriusdb@cs.wisc.edu

Abstract

The era of GPU-powered data analytics has arrived. In this paper, we argue that recent advances in hardware (e.g., larger GPU memory, faster interconnect and IO, and declining cost) and software (e.g., composable data systems and mature libraries) have removed the key barriers that have limited the wider adoption of GPU data analytics. We present Sirius, a prototype open-source GPU-native SQL engine that offers drop-in acceleration for diverse data systems. Sirius treats GPU as the primary engine and leverages libraries like libcudf for high-performance relational operators. It provides drop-in acceleration for existing databases by leveraging the standard Substrait query representation, replacing the CPU engine without changing the user-facing interface. Sirius achieves 8.3× and 7.4× better cost efficiency on TPC-H and ClickBench, respectively, when integrated with single-node DuckDB, and delivers up to 12.5× speedup when integrated with Apache Doris distributed engine.

1 Introduction

The performance of a SQL engine is ultimately driven by the capabilities of the underlying hardware—especially compute throughput and memory bandwidth. Modern GPUs are rapidly outpacing CPUs on both fronts and have become the default hardware choice for data- and compute-intensive applications such as machine learning. Table 1 compares recent CPU and GPU architectures, highlighting the contrast in core count and memory bandwidth. Given the inherently parallel nature of relational data analytics, GPUs are a perfect fit for future analytical databases.

Despite their great potential, however, GPU-based SQL engines [6, 9, 10, 15, 18, 18] have not yet seen mainstream adoption. Prior efforts, both academic and commercial, have been constrained by the following four key challenges and thereby remain niche solutions.

- **Limited GPU memory capacity**: Up to 288 GB in a latest GPU device in contrast to several TBs in CPUs.
- **Data movement bottleneck**: Traditionally, PCIe has relatively low bandwidth, creating a bottleneck when transferring data between GPU and the rest of the system.
- **Expensive GPU hardware**: High-end GPUs remain significantly more expensive than CPUs and are often in short supply.

*Equal contributions

CIDR'26, Chaminade, USA

ACM ISBN 978-1-4503-XXXX-X/2018/06
https://doi.org/XXXXXXX.XXXXXXX

Table 1: Comparison of CPU and GPU Instances

	Amazon c6a.metal (AMD EPYC CPU)	GH200 (NVIDIA GPU)
Core Count	192 (vCPUs)	14,000+ (CUDA cores)
Memory BW	~400 GB/s	3,000 GB/s (HBM)
Memory Size	384 GB	96 GB (HBM)
Rental Cost	$7.344/h (AWS)	$1.5/h (Lambda Labs)

- **Engineering cost**: Building a full SQL engine optimized for GPUs from the ground up is a major engineering challenge.

In this paper, we argue that the hardware and software foundations are finally in place today to overcome these challenges and enable scalable GPU data analytics. On the hardware side, GPU memory capacity doubles almost every generation, starting from Volta (32 GB), to Ampere (80 GB), Hopper (192 GB), and most recently Blackwell (288 GB). Better interconnects such as PCIe Gen6, NVLink-C2C [13], and GPU Direct [8] significantly reduce the data movement overhead between GPU and other system components. This allows a GPU to process data beyond on-device memory with very fast speed, enabling TBs of analytics even on a single GPU and more with distribution. Meanwhile, GPUs are increasingly affordable and accessible, especially for older generations which are already sufficient for data analytics workloads.

On the software side, the GPU ecosystem has significantly matured. Libraries such as `libcudf` [6] provides high-performance primitives like joins and aggregations that a GPU SQL engine can build upon. The rise of composable data systems [25]—driven by open standards like Substrait [16] for query representation, Apache Arrow and Parquet for columnar data formats—enable greater interoperability. A modern GPU engine can reuse existing components, such as SQL parser and query optimizer, to drastically reduce the complexity of building the system from the ground up.

To demonstrate the feasibility of GPU-native databases, we have built Sirius[1], a prototype open-source SQL engine that uses GPU as the primary execution device and delivers drop-in acceleration across diverse data systems. Sirius integrates seamlessly with existing database systems via the standard Substrait query representation. For example, plugging Sirius into DuckDB causes minimal change to the user-facing interface. Instead of executing the query using DuckDB's CPU engine, the Substrait plan is routed to Sirius for GPU-native execution. Sirius builds on GPU libraries such as `libcudf` [6], `RMM` [14], and `NCCL` [11], reusing optimized implementations of core relational operators like joins, filters, aggregations, and data shuffle. Thanks to its modular design, Sirius also allows developers to easily switch the operator implementation between these GPU libraries and custom CUDA kernels. This flexibility

[1]https://github.com/sirius-db/sirius

Outline

Part I → Why GPU Databases?

Hardware trend

Software trend

Part II → Sirius: A GPU-Native SQL Engine

Design

Evaluation

Future Direction

CPU vs. GPU Hardware Comparison

Table 1: Comparison of CPU and GPU Instances

	Amazon c6a.metal (AMD EPYC CPU) since 2022	GH200 (NVIDIA GPU) since 2023	Rubin (NVIDIA GPU) since 2026
Core Count	192 (vCPUs)	14,000+ (CUDA cores)	-
Memory BW	~400 GB/s	3,000 GB/s (HBM3)	22,000 GB/s (HBM4)
Memory Size	384 GB	96 GB (HBM3)	288 GB (HBM4)
Network	6.25 GB/s (Ethernet)	900 GB/s (NVLink)	3,600 GB/s (NVLink)
Rental Cost	$7.344/h (AWS)	$1.5/h (Lambda Labs)	-

CPU vs. GPU Hardware Comparison

Table 1: Comparison of CPU and GPU Instances

	Amazon c6a.metal (AMD EPYC CPU) since 2022	GH200 (NVIDIA GPU) since 2023	Rubin (NVIDIA GPU) since 2026
Core Count	192 (vCPUs)	14,000+ (CUDA cores)	-
Memory BW	~400 GB/s	3,000 GB/s (HBM3)	22,000 GB/s (HBM4)
Memory Size	384 GB	96 GB (HBM3)	288 GB (HBM4)
Network	6.25 GB/s (Ethernet)	900 GB/s (NVLink)	3,600 GB/s (NVLink)
Rental Cost	$7.344/h (AWS)	$1.5/h (Lambda Labs)	-

GPUs excel at:

Compute power

CPU vs. GPU Hardware Comparison

Table 1: Comparison of CPU and GPU Instances

	Amazon c6a.metal (AMD EPYC CPU) since 2022	GH200 (NVIDIA GPU) since 2023	Rubin (NVIDIA GPU) since 2026
Core Count	192 (vCPUs)	14,000+ (CUDA cores)	-
Memory BW	~400 GB/s	3,000 GB/s (HBM3)	22,000 GB/s (HBM4)
Memory Size	384 GB	96 GB (HBM3)	288 GB (HBM4)
Network	6.25 GB/s (Ethernet)	900 GB/s (NVLink)	3,600 GB/s (NVLink)
Rental Cost	$7.344/h (AWS)	$1.5/h (Lambda Labs)	-

GPUs excel at:

- **Compute power**
- **Memory bandwidth**

CPU vs. GPU Hardware Comparison

Table 1: Comparison of CPU and GPU Instances

	Amazon c6a.metal (AMD EPYC CPU) since 2022	GH200 (NVIDIA GPU) since 2023	Rubin (NVIDIA GPU) since 2026
Core Count	192 (vCPUs)	14,000+ (CUDA cores)	-
Memory BW	~400 GB/s	3,000 GB/s (HBM3)	22,000 GB/s (HBM4)
Memory Size	384 GB	96 GB (HBM3)	288 GB (HBM4)
Network	6.25 GB/s (Ethernet)	900 GB/s (NVLink)	3,600 GB/s (NVLink)
Rental Cost	$7.344/h (AWS)	$1.5/h (Lambda Labs)	-

GPUs excel at:

- **Compute power**
- **Memory bandwidth**
- **Network bandwidth**

CPU vs. GPU Hardware Comparison

Table 1: Comparison of CPU and GPU Instances

	Amazon c6a.metal (AMD EPYC CPU) since 2022	GH200 (NVIDIA GPU) since 2023	Rubin (NVIDIA GPU) since 2026
Core Count	192 (vCPUs)	14,000+ (CUDA cores)	-
Memory BW	~400 GB/s	3,000 GB/s (HBM3)	22,000 GB/s (HBM4)
Memory Size	384 GB	96 GB (HBM3)	288 GB (HBM4)
Network	6.25 GB/s (Ethernet)	900 GB/s (NVLink)	3,600 GB/s (NVLink)
Rental Cost	$7.344/h (AWS)	$1.5/h (Lambda Labs)	-

GPUs excel at:

- **Compute power**
- **Memory bandwidth**
- **Network bandwidth**

Relational analytics is inherently parallel → A natural fit for GPUs

Yet most data analytics today remains CPU-centric—missing the GPU revolution

Why GPU SQL Not Mainstream Yet?

Key challenges in GPU data analytics

- Limited GPU memory capacity
- Data movement bottleneck
- GPU hardware cost
- Engineering complexity

GPU SQL at a Tipping Point

Challenge #1: Limited GPU memory capacity

- GPU memory is smaller than CPU memory

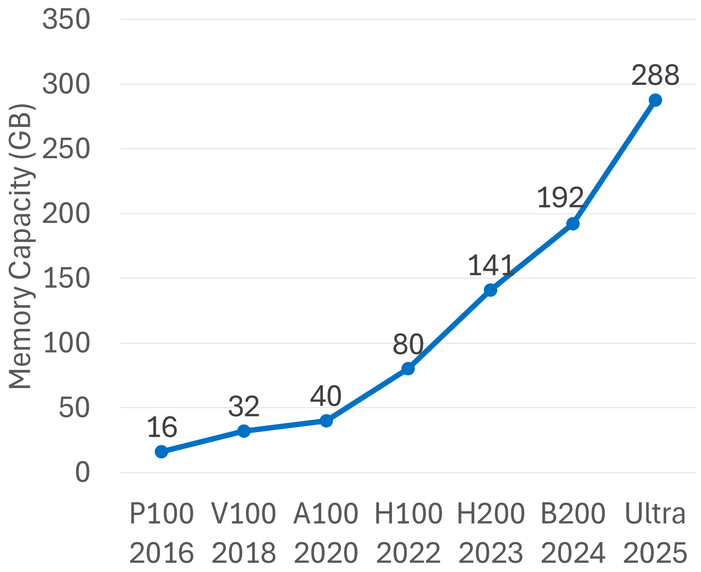
GPU SQL at a Tipping Point

Challenge #1: Limited GPU memory capacity

- GPU memory is smaller than CPU memory

Hardware trend #1: Bigger Memory at Faster Speed

- **GPU memory capacity**: up to **288 GB** HBM per GPU



GPU Memory Capacity

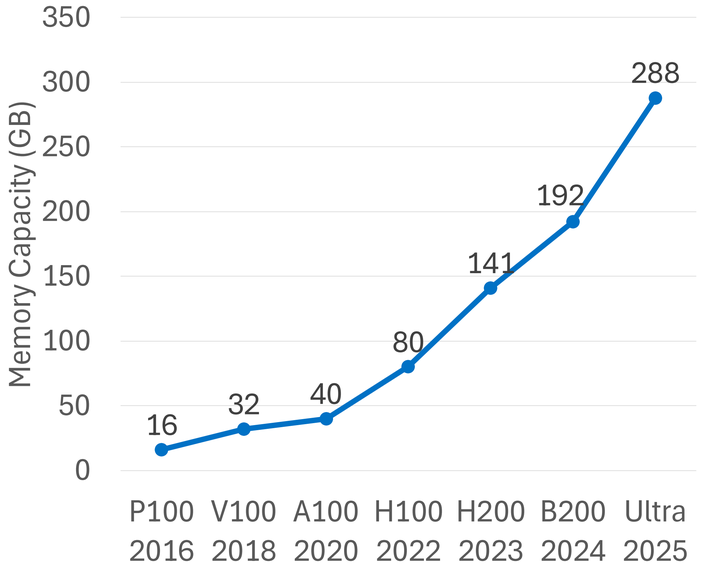
GPU SQL at a Tipping Point

Challenge #1: Limited GPU memory capacity

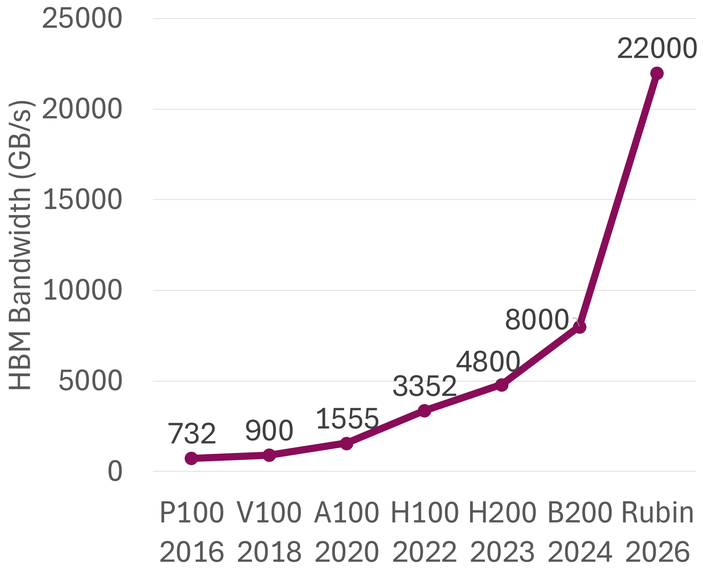
- GPU memory is smaller than CPU memory

Hardware trend #1: Bigger Memory at Faster Speed

- **GPU memory capacity**: up to **288 GB** HBM per GPU
- **HBM bandwidth**: up to **22,000 GB/s** in Rubin GPU



GPU Memory Capacity



HBM Bandwidth

GPU SQL at a Tipping Point

Challenge #2: Data movement bottleneck

- Loading data into GPU is bottlenecked by PCIe

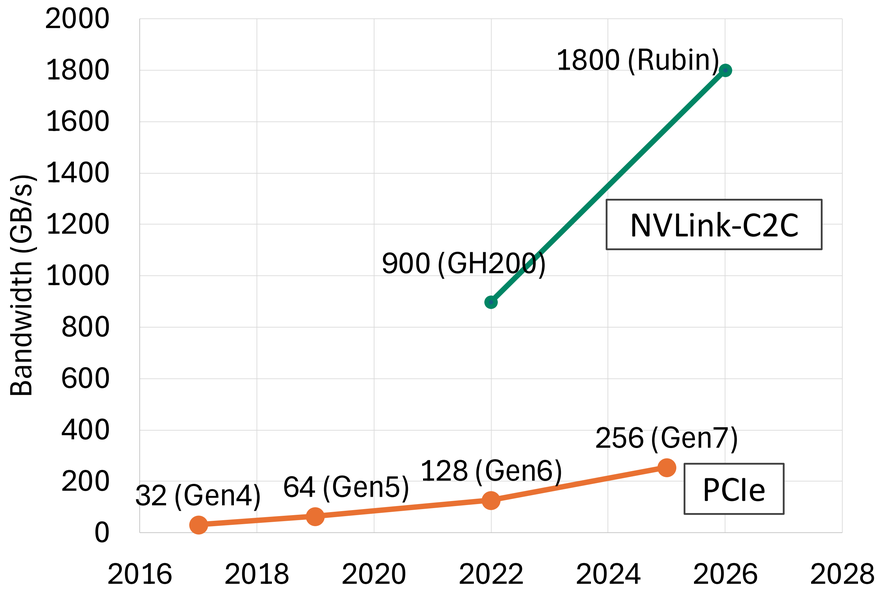
GPU SQL at a Tipping Point

Challenge #2: Data movement bottleneck

- Loading data into GPU is bottlenecked by PCIe

Hardware trend #2: Faster Network and Storage

- **CPU-GPU bandwidth**: NVLinkC2C provides up to **1800 GB/s**



CPU-GPU bandwidth

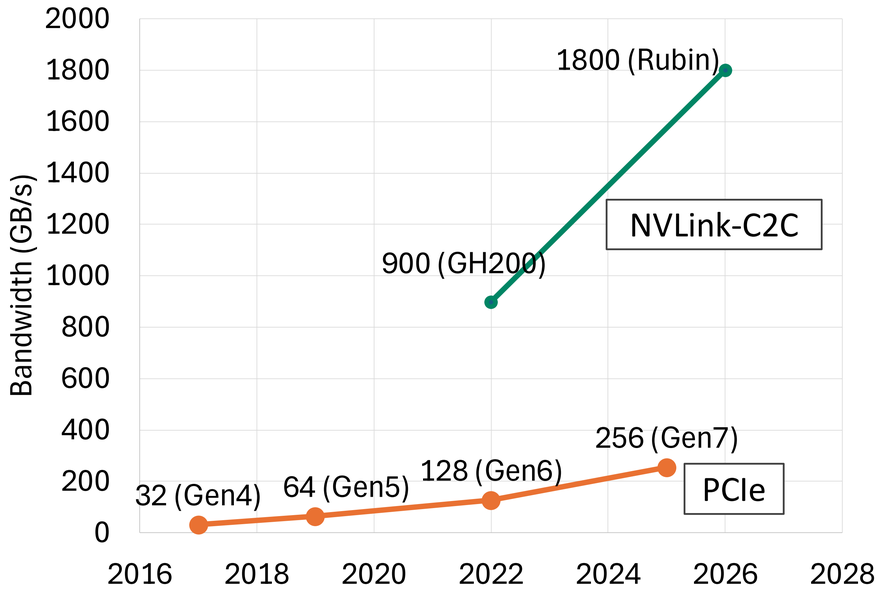
GPU SQL at a Tipping Point

Challenge #2: Data movement bottleneck

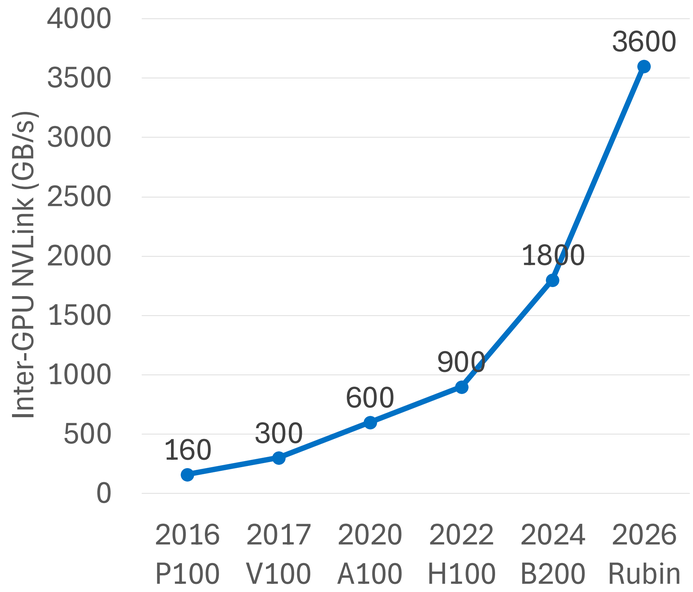
- Loading data into GPU is bottlenecked by PCIe

Hardware trend #2: Faster Network and Storage

- **CPU-GPU bandwidth**: NVLinkC2C provides up to **1800 GB/s**
- **GPU-GPU network**: NVLink provides up to **3600 GB/s**



CPU-GPU bandwidth



Inter-GPU NVLink Bandwidth

GPU SQL at a Tipping Point

Challenge #3: GPU hardware cost

- High-end GPUs are expensive and often in short supply

GPU SQL at a Tipping Point

Challenge #3: GPU hardware cost

- High-end GPUs are expensive and often in short supply

Hardware trend #3: Older-generation GPUs accessible at lower cost

- On-demand H100 prices drop from **$8/hour in March 2023** to around **$3/hour in 2025**
- In AWS, in 2025 alone, the A100, H100, and H200 on demand price have dropped by **33%**, **44%**, and **25%** respectively

Table 1: Comparison of CPU and GPU Instances

	Amazon c6a.metal (AMD EPYC CPU) since 2022	GH200 (NVIDIA GPU) since 2023	Rubin (NVIDIA GPU) since 2026
Core Count	192 (vCPUs)	14,000+ (CUDA cores)	-
Memory BW	~400 GB/s	3,000 GB/s (HBM3)	22,000 GB/s (HBM4)
Memory Size	384 GB	96 GB (HBM3)	288 GB (HBM4)
Network	6.25 GB/s (Ethernet)	900 GB/s (NVLink)	3,600 GB/s (NVLink)
Rental Cost	$7.344/h (AWS)	$1.5/h (Lambda Labs)	-

GPU SQL at a Tipping Point

Challenge #4: Engineering complexity

- Building SQL engine from the ground up is challenging

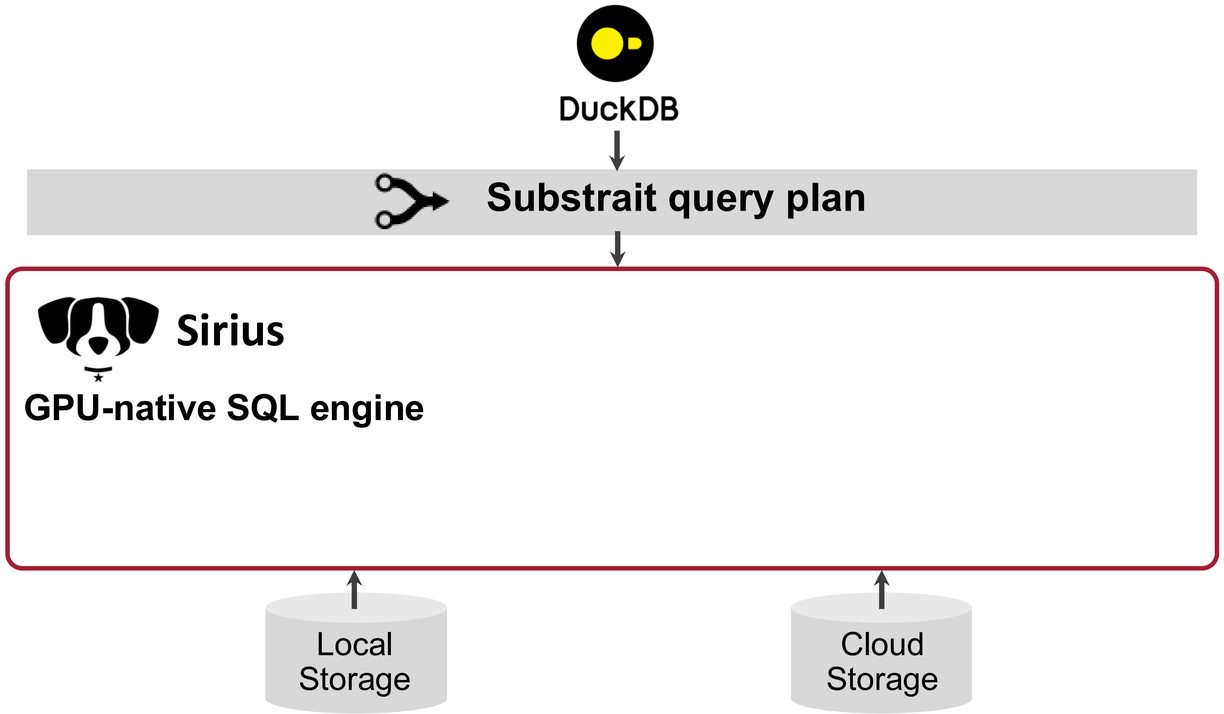
GPU SQL at a Tipping Point

Challenge #4: Engineering complexity

- Building SQL engine from the ground up is challenging

Software trend #1: Composable Data Systems

- Reuse existing components (e.g., SQL frontend, query optimizer)
- Substrait: A standardized IR for query plans, enabling portability across query engines



GPU SQL at a Tipping Point

Challenge #4: Engineering complexity

- Building SQL engine from the ground up is challenging

Software trend #1: Composable Data Systems

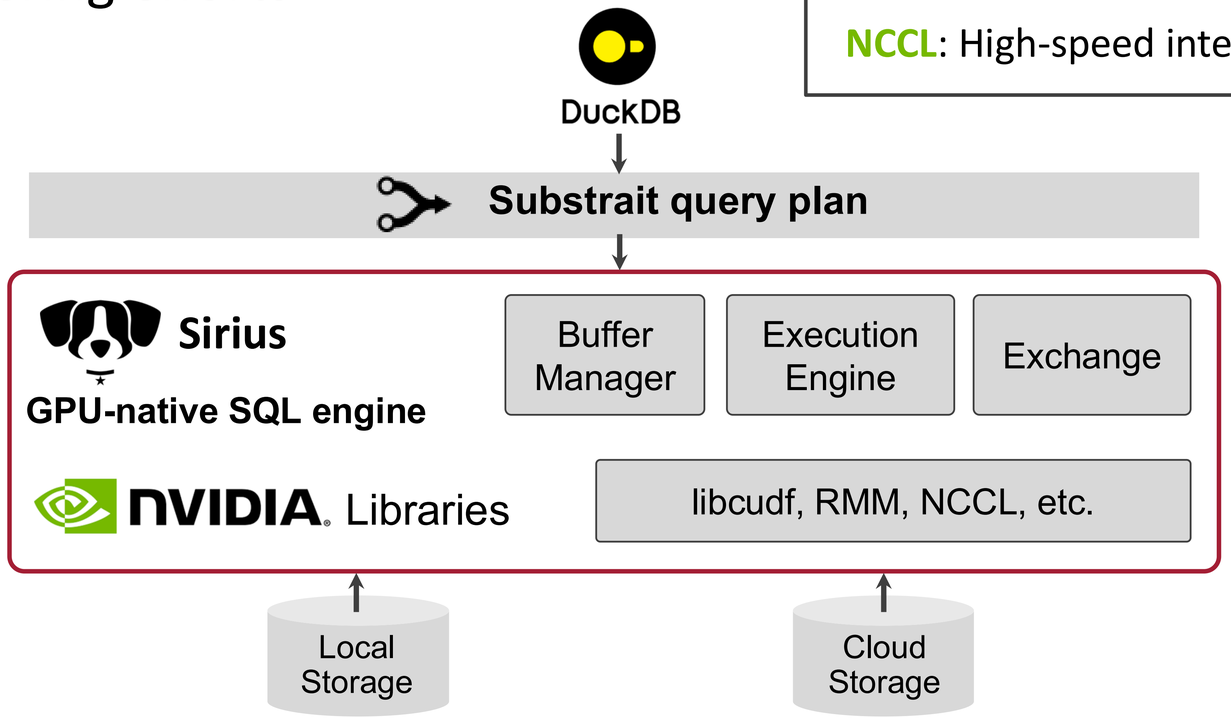
Software trend #2: Maturing GPU Libraries

- Minimizing engineering efforts

libcudf: Columnar-oriented relational operators (e.g., joins, aggregations) on GPU

RMM: An efficient GPU memory allocator

NCCL: High-speed inter-GPU communication library



Summary: GPU SQL at a Tipping Point

Modern hardware and software trends make GPU databases viable at scale

- Challenge #1: Limited GPU memory capacity
- Challenge #2: Data movement bottleneck
- Challenge #3: GPU hardware cost
- Challenge #4: Engineering complexity

Hardware Trends

#1: Bigger Memory at Faster Speed
#2: Faster Network and Storage
#3: Declining GPU Cost

Software Trends

#1: Composable data systems
#2: Maturing GPU Libraries

The GPU era of data analytics has arrived

Rethinking Analytical Processing in the GPU Era

Bobbi Yogatama[2*], Yifei Yang[1*], Kevin Kristensen[1], Devesh Sarda[1], Abigale Kim[1], Adrian Cockcroft[5], Yu Teng, Joshua Patterson[2], Gregory Kimball[2], Wes McKinney[4], Weiwei Gong[3], Xiangyao Yu[1]
[1]University of Wisconsin-Madison, [2]NVIDIA, [3]Oracle, [4]Posit PBC, [5]OrionX
siriusdb@cs.wisc.edu

Abstract

The era of GPU-powered data analytics has arrived. In this paper, we argue that recent advances in hardware (e.g., larger GPU memory, faster interconnect and IO, and declining cost) and software (e.g., composable data systems and mature libraries) have removed the key barriers that have limited the wider adoption of GPU data analytics. We present Sirius, a prototype open-source GPU-native SQL engine that offers drop-in acceleration for diverse data systems. Sirius treats GPU as the primary engine and leverages libraries like libcudf for high-performance relational operators. It provides drop-in acceleration for existing databases by leveraging the standard Substrait query representation, replacing the CPU engine without changing the user-facing interface. Sirius achieves 8.3× and 7.4× better cost efficiency on TPC-H and ClickBench, respectively, when integrated with single-node DuckDB, and delivers up to 12.5× speedup when integrated with Apache Doris distributed engine.

1 Introduction

The performance of a SQL engine is ultimately driven by the capabilities of the underlying hardware—especially compute throughput and memory bandwidth. Modern GPUs are rapidly outpacing CPUs on both fronts and have become the default hardware choice for data- and compute-intensive applications such as machine learning. Table 1 compares recent CPU and GPU architectures, highlighting the contrast in core count and memory bandwidth. Given the inherently parallel nature of relational data analytics, GPUs are a perfect fit for future analytical databases.

Despite their great potential, however, GPU-based SQL engines [6, 9, 10, 15, 18, 18] have not yet seen mainstream adoption. Prior efforts, both academic and commercial, have been constrained by the following four key challenges and thereby remain niche solutions.

- **Limited GPU memory capacity**: Up to 288 GB in a latest GPU device in contrast to several TBs in CPUs.
- **Data movement bottleneck**: Traditionally, PCIe has relatively low bandwidth, creating a bottleneck when transferring data between GPU and the rest of the system.
- **Expensive GPU hardware**: High-end GPUs remain significantly more expensive than CPUs and are often in short supply.

*Equal contributions

CIDR'26, Chaminade, USA

ACM ISBN 978-1-4503-XXXX-X/2018/06
https://doi.org/XXXXXXX.XXXXXXX

Table 1: Comparison of CPU and GPU Instances

	Amazon c6a.metal (AMD EPYC CPU)	GH200 (NVIDIA GPU)
Core Count	192 (vCPUs)	14,000+ (CUDA cores)
Memory BW	~400 GB/s	3,000 GB/s (HBM)
Memory Size	384 GB	96 GB (HBM)
Rental Cost	$7.344/h (AWS)	$1.5/h (Lambda Labs)

- **Engineering cost**: Building a full SQL engine optimized for GPUs from the ground up is a major engineering challenge.

In this paper, we argue that the hardware and software foundations are finally in place today to overcome these challenges and enable scalable GPU data analytics. On the hardware side, GPU memory capacity doubles almost every generation, starting from Volta (32 GB), to Ampere (80 GB), Hopper (192 GB), and most recently Blackwell (288 GB). Better interconnects such as PCIe Gen6, NVLink-C2C [13], and GPU Direct [8] significantly reduce the data movement overhead between GPU and other system components. This allows a GPU to process data beyond on-device memory with very fast speed, enabling TBs of analytics even on a single GPU and more with distribution. Meanwhile, GPUs are increasingly affordable and accessible, especially for older generations which are already sufficient for data analytics workloads.

On the software side, the GPU ecosystem has significantly matured. Libraries such as `libcudf` [6] provides high-performance primitives like joins and aggregations that a GPU SQL engine can build upon. The rise of composable data systems [25]—driven by open standards like Substrait [16] for query representation, Apache Arrow and Parquet for columnar data formats—enable greater interoperability. A modern GPU engine can reuse existing components, such as SQL parser and query optimizer, to drastically reduce the complexity of building the system from the ground up.

To demonstrate the feasibility of GPU-native databases, we have built Sirius[1], a prototype open-source SQL engine that uses GPU as the primary execution device and delivers drop-in acceleration across diverse data systems. Sirius integrates seamlessly with existing database systems via the standard Substrait query representation. For example, plugging Sirius into DuckDB causes minimal change to the user-facing interface. Instead of executing the query using DuckDB's CPU engine, the Substrait plan is routed to Sirius for GPU-native execution. Sirius builds on GPU libraries such as `libcudf` [6], `RMM` [14], and `NCCL` [11], reusing optimized implementations of core relational operators like joins, filters, aggregations, and data shuffle. Thanks to its modular design, Sirius also allows developers to easily switch the operator implementation between these GPU libraries and custom CUDA kernels. This flexibility

[1]https://github.com/sirius-db/sirius

Outline

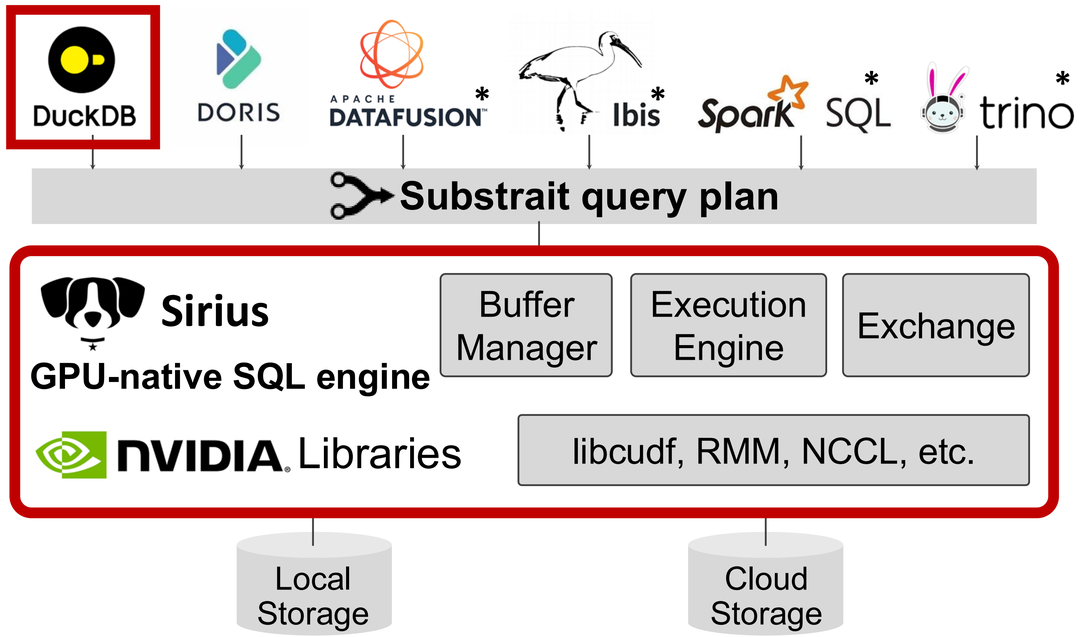
Part I → Why GPU Databases?

- Hardware trend
- Software trend

Part II → Sirius: A GPU-Native SQL Engine

- Design
- Evaluation
- Future Direction

Sirius: A GPU-Native SQL Engine



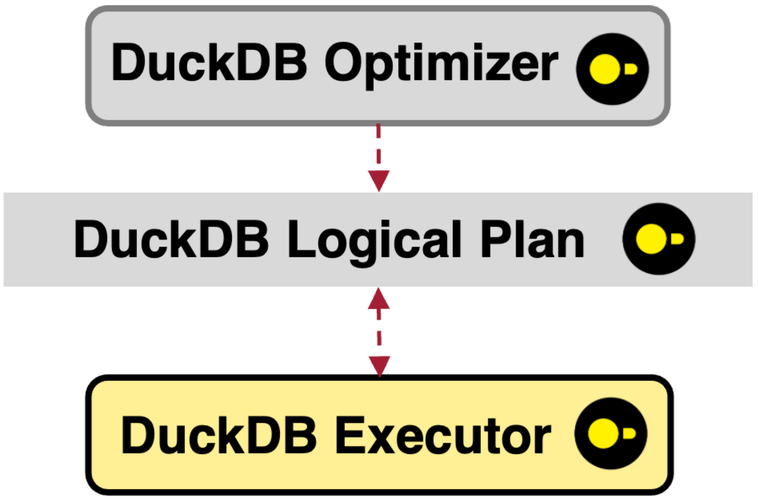
Design principle #1: **GPU-native solution** (vs. retrofitted design)

- Core components (buffer manager, query engine) are designed assuming GPU execution

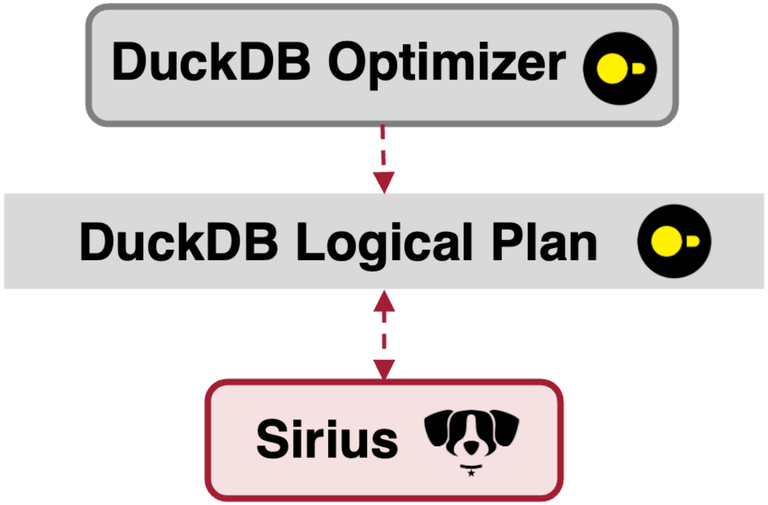
Design principle #2: **MICE (Modular, Interoperable, Composable, Extensible)**

- Drop-in accelerate data systems with minimal modification to the host systems

Query Lifecycle in Sirius



Query Lifecycle in DuckDB



Query Lifecycle in Sirius

Developed as a DuckDB Extension:

- ➔No modification to DuckDB codebase
- ➔No modification to DuckDB User Interface
- ➔Fallback execution to DuckDB

How Does Sirius Work?

Running queries in DuckDB vs Sirius

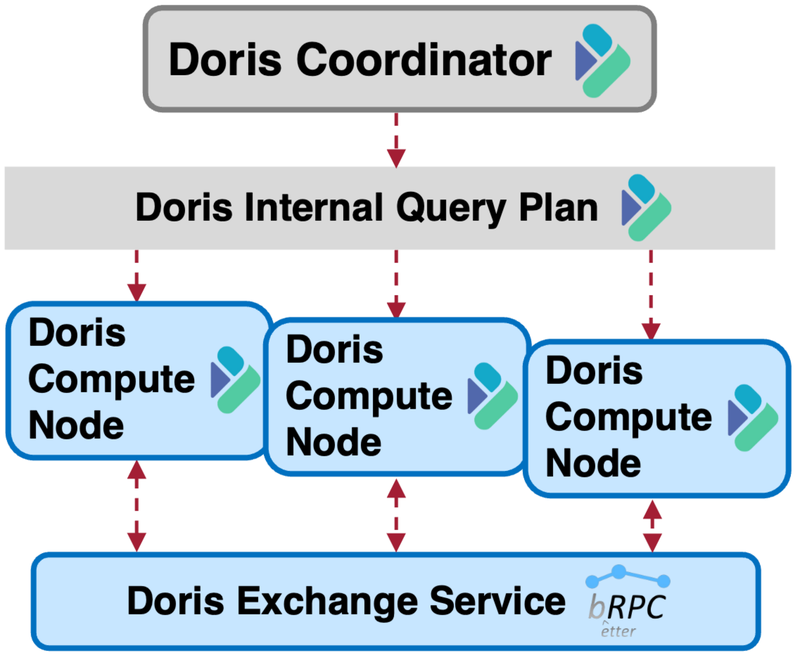
```
D select
    l_orderkey, sum(l_extendedprice * (1 - l_discount)) as revenue,
    o_orderdate, o_shippriority
  from
    customer, orders, lineitem
  where
    c_mktsegment = 1
    and c_custkey = o_custkey
    and l_orderkey = o_orderkey
    and o_orderdate < 19950315
    and l_shipdate > 19950315
  group by
    l_orderkey, o_orderdate, o_shippriority
  order by
    revenue desc, o_orderdate
  limit 5;
┌─────────────┬────────────────────┬─────────────┬──────────────────┐
│ L_ORDERKEY  │      revenue       │ O_ORDERDATE │ O_SHIPPRIORITY   │
│    int64    │       double       │    int64    │      int64       │
├─────────────┼────────────────────┼─────────────┼──────────────────┤
│     2456423 │        406181.0111 │    19950305 │                0 │
│     3459808 │ 405838.69889999996 │    19950304 │                0 │
│      492164 │         390324.061 │    19950219 │                0 │
│     1188320 │        384537.9359 │    19950309 │                0 │
│     2435712 │ 378673.05580000003 │    19950226 │                0 │
└─────────────┴────────────────────┴─────────────┴──────────────────┘
Run Time (s): real 0.066 user 0.364014 sys 0.076839
D
```

DuckDB

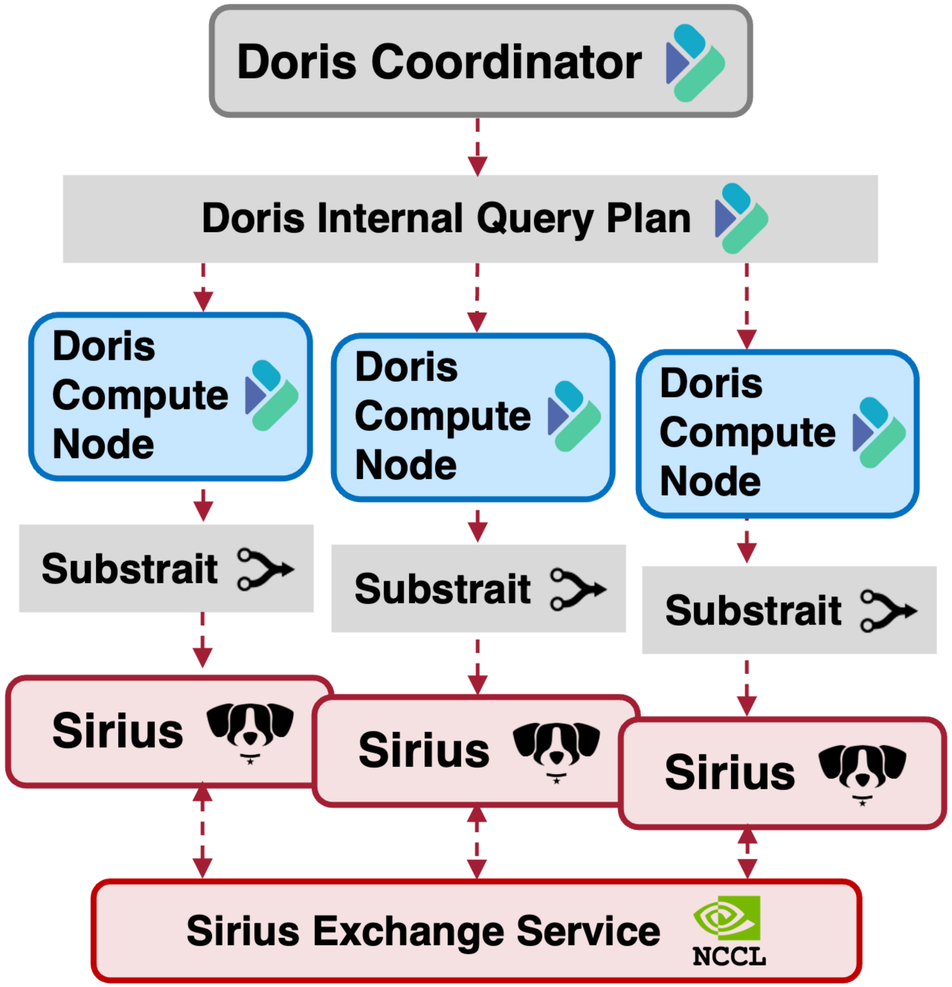
```
D call gpu_processing("select
    l_orderkey, sum(l_extendedprice * (1 - l_discount)) as revenue,
    o_orderdate, o_shippriority
  from
    customer, orders, lineitem
  where
    c_mktsegment = 1
    and c_custkey = o_custkey
    and l_orderkey = o_orderkey
    and o_orderdate < 19950315
    and l_shipdate > 19950315
  group by
    l_orderkey, o_orderdate, o_shippriority
  order by
    revenue desc, o_orderdate
  limit 5;");
┌─────────────┬────────────────────┬─────────────┬──────────────────┐
│ L_ORDERKEY  │      revenue       │ O_ORDERDATE │ O_SHIPPRIORITY   │
│    int64    │       double       │    int64    │      int64       │
├─────────────┼────────────────────┼─────────────┼──────────────────┤
│     2456423 │        406181.0111 │    19950305 │                0 │
│     3459808 │ 405838.69889999996 │    19950304 │                0 │
│      492164 │         390324.061 │    19950219 │                0 │
│     1188320 │        384537.9359 │    19950309 │                0 │
│     2435712 │ 378673.05580000003 │    19950226 │                0 │
└─────────────┴────────────────────┴─────────────┴──────────────────┘
Run Time (s): real 0.012 user 0.008004 sys 0.004387
D
```

Sirius

Query Lifecycle in Sirius – Distributed



Query Lifecycle in Doris



Query Lifecycle in Sirius-Accelerated Doris

Rethinking Analytical Processing in the GPU Era

Bobbi Yogatama[2*], Yifei Yang[1*], Kevin Kristensen[1], Devesh Sarda[1], Abigale Kim[1], Adrian Cockcroft[5], Yu Teng, Joshua Patterson[2], Gregory Kimball[2], Wes McKinney[4], Weiwei Gong[3], Xiangyao Yu[1]
[1]University of Wisconsin-Madison, [2]NVIDIA, [3]Oracle, [4]Posit PBC, [5]OrionX
siriusdb@cs.wisc.edu

Abstract

The era of GPU-powered data analytics has arrived. In this paper, we argue that recent advances in hardware (e.g., larger GPU memory, faster interconnect and IO, and declining cost) and software (e.g., composable data systems and mature libraries) have removed the key barriers that have limited the wider adoption of GPU data analytics. We present Sirius, a prototype open-source GPU-native SQL engine that offers drop-in acceleration for diverse data systems. Sirius treats GPU as the primary engine and leverages libraries like libcudf for high-performance relational operators. It provides drop-in acceleration for existing databases by leveraging the standard Substrait query representation, replacing the CPU engine without changing the user-facing interface. Sirius achieves 8.3× and 7.4× better cost efficiency on TPC-H and ClickBench, respectively, when integrated with single-node DuckDB, and delivers up to 12.5× speedup when integrated with Apache Doris distributed engine.

1 Introduction

The performance of a SQL engine is ultimately driven by the capabilities of the underlying hardware—especially compute throughput and memory bandwidth. Modern GPUs are rapidly outpacing CPUs on both fronts and have become the default hardware choice for data- and compute-intensive applications such as machine learning. Table 1 compares recent CPU and GPU architectures, highlighting the contrast in core count and memory bandwidth. Given the inherently parallel nature of relational data analytics, GPUs are a perfect fit for future analytical databases.

Despite their great potential, however, GPU-based SQL engines [6, 9, 10, 15, 18, 18] have not yet seen mainstream adoption. Prior efforts, both academic and commercial, have been constrained by the following four key challenges and thereby remain niche solutions.

- **Limited GPU memory capacity**: Up to 288 GB in a latest GPU device in contrast to several TBs in CPUs.
- **Data movement bottleneck**: Traditionally, PCIe has relatively low bandwidth, creating a bottleneck when transferring data between GPU and the rest of the system.
- **Expensive GPU hardware**: High-end GPUs remain significantly more expensive than CPUs and are often in short supply.

*Equal contributions

CIDR'26, Chaminade, USA

ACM ISBN 978-1-4503-XXXX-X/2018/06
https://doi.org/XXXXXXX.XXXXXXX

Table 1: Comparison of CPU and GPU Instances

	Amazon c6a.metal (AMD EPYC CPU)	GH200 (NVIDIA GPU)
Core Count	192 (vCPUs)	14,000+ (CUDA cores)
Memory BW	~400 GB/s	3,000 GB/s (HBM)
Memory Size	384 GB	96 GB (HBM)
Rental Cost	$7.344/h (AWS)	$1.5/h (Lambda Labs)

- **Engineering cost**: Building a full SQL engine optimized for GPUs from the ground up is a major engineering challenge.

In this paper, we argue that the hardware and software foundations are finally in place today to overcome these challenges and enable scalable GPU data analytics. On the hardware side, GPU memory capacity doubles almost every generation, starting from Volta (32 GB), to Ampere (80 GB), Hopper (192 GB), and most recently Blackwell (288 GB). Better interconnects such as PCIe Gen6, NVLink-C2C [13], and GPU Direct [8] significantly reduce the data movement overhead between GPU and other system components. This allows a GPU to process data beyond on-device memory with very fast speed, enabling TBs of analytics even on a single GPU and more with distribution. Meanwhile, GPUs are increasingly affordable and accessible, especially for older generations which are already sufficient for data analytics workloads.

On the software side, the GPU ecosystem has significantly matured. Libraries such as `libcudf` [6] provides high-performance primitives like joins and aggregations that a GPU SQL engine can build upon. The rise of composable data systems [25]—driven by open standards like Substrait [16] for query representation, Apache Arrow and Parquet for columnar data formats—enable greater interoperability. A modern GPU engine can reuse existing components, such as SQL parser and query optimizer, to drastically reduce the complexity of building the system from the ground up.

To demonstrate the feasibility of GPU-native databases, we have built Sirius[1], a prototype open-source SQL engine that uses GPU as the primary execution device and delivers drop-in acceleration across diverse data systems. Sirius integrates seamlessly with existing database systems via the standard Substrait query representation. For example, plugging Sirius into DuckDB causes minimal change to the user-facing interface. Instead of executing the query using DuckDB's CPU engine, the Substrait plan is routed to Sirius for GPU-native execution. Sirius builds on GPU libraries such as `libcudf` [6], `RMM` [14], and `NCCL` [11], reusing optimized implementations of core relational operators like joins, filters, aggregations, and data shuffle. Thanks to its modular design, Sirius also allows developers to easily switch the operator implementation between these GPU libraries and custom CUDA kernels. This flexibility

[1]https://github.com/sirius-db/sirius

Outline

Part I → Why GPU Databases?

- Hardware trend
- Software trend

Part II → Sirius: A GPU-Native SQL Engine

- Design
- **Evaluation**
- Future Direction

Performance Evaluation – Single Node

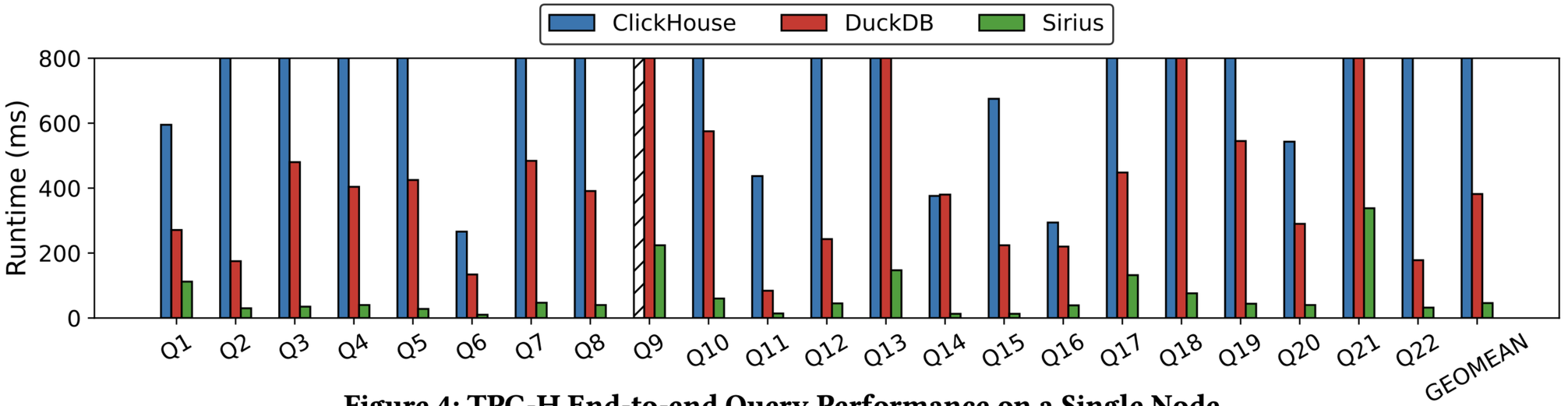


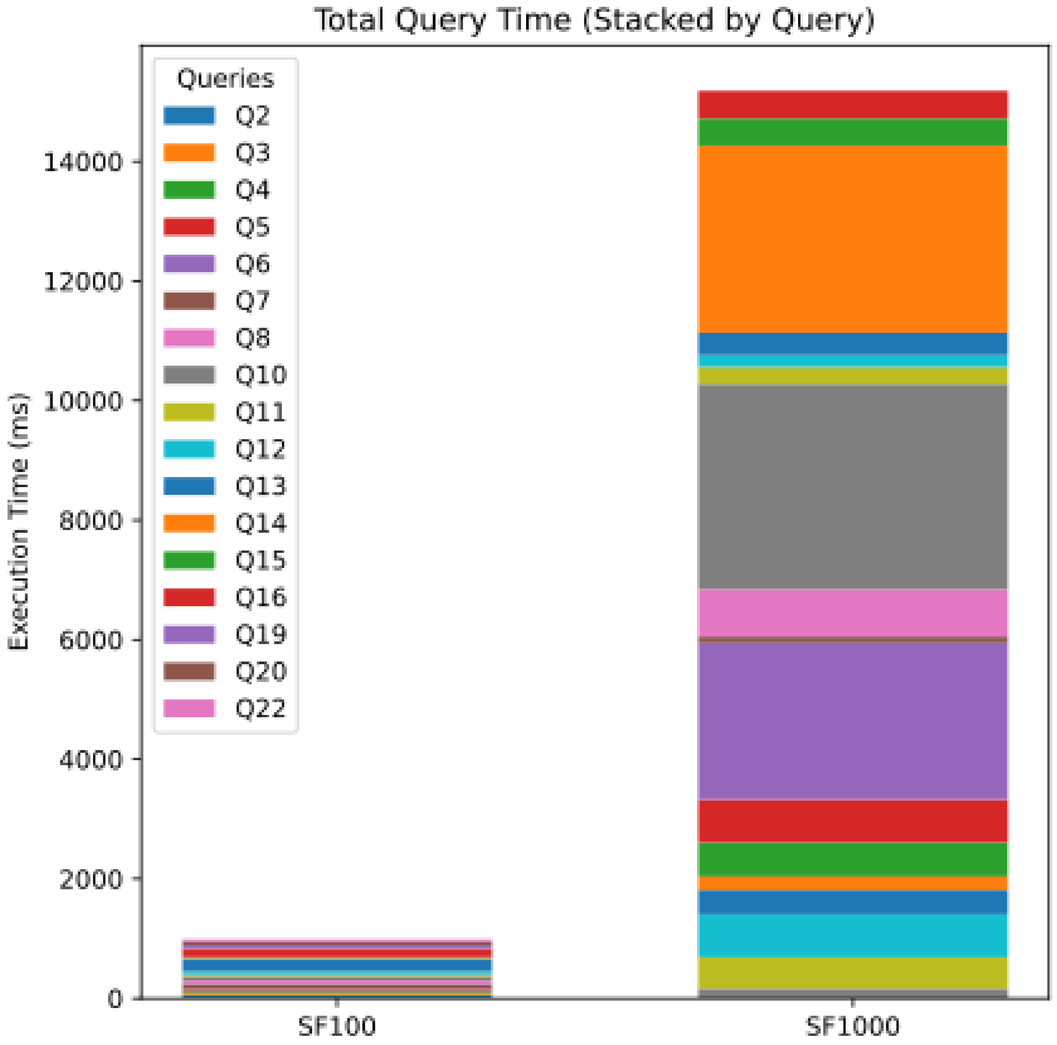
Figure 4: TPC-H End-to-end Query Performance on a Single Node.

TPC-H 100GB

- CPU: c8i.8xlarge@AWS ($1.5 per hour)
- GPU: GH200@LambdaLabs ($1.5 per hour)

Sirius accelerates DuckDB by 8.2x

Performance Evaluation – Single Node

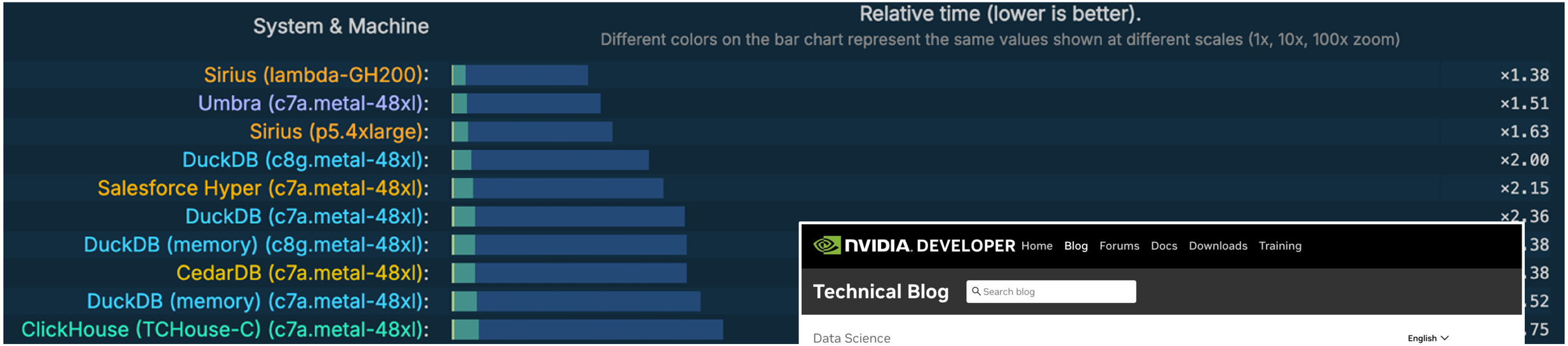


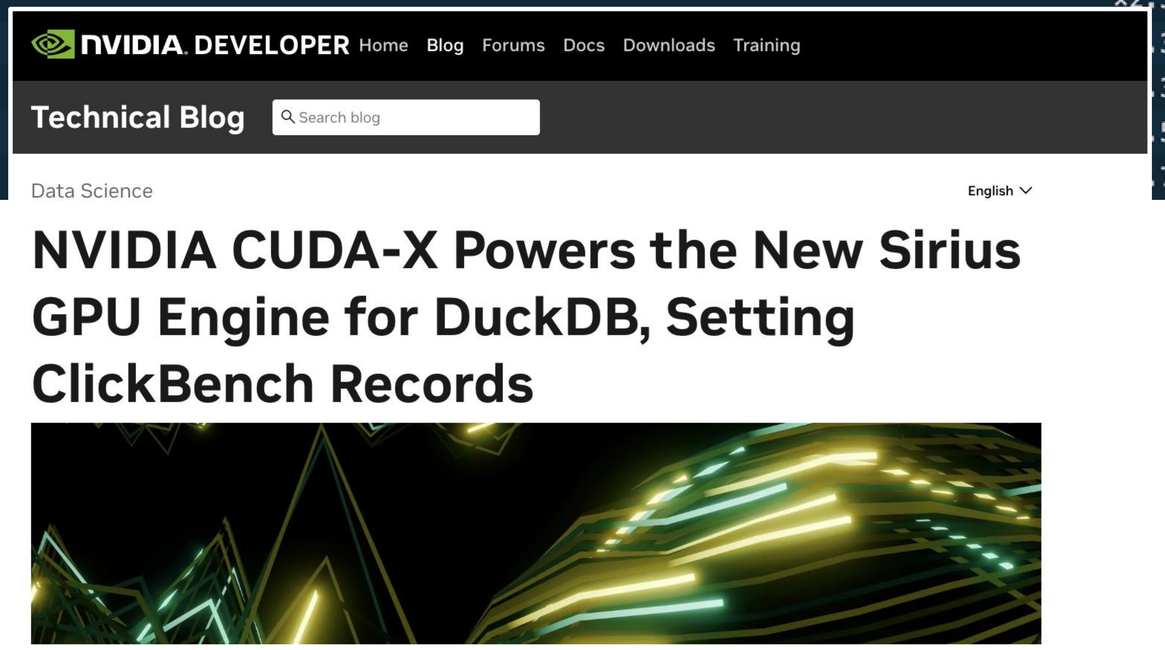
TPC-H SF100 vs 1000

GPU: GH200@LambdaLabs ($1.5 per hour)

Scaling Sirius to 1TB! (ongoing)

Performance Evaluation – ClickBench





Breaking the ClickBench Record!!!

Running on hardware that is 7x cheaper

Performance Evaluation – Multi-Node

Table 2: TPC-H End-to-End Query Performance in the Distributed Setting.

	Doris	ClickHouse	Sirius	Breakdown in Sirius		
				Compute	Exchange	Other
Q1	1193	393	**97**	33	3	61
Q3	838	12785	**341**	43	233	75
Q6	199	294	**84**	36	1	47

TPC-H 100GB

- 4-node cluster
- Each node: NVIDIA A100 GPU + 64-cores Intel Xeon Gold 6526Y CPUs

Sirius accelerates Doris by 2.4–12.5x

Rethinking Analytical Processing in the GPU Era

Bobbi Yogatama[2*], Yifei Yang[1*], Kevin Kristensen[1], Devesh Sarda[1], Abigale Kim[1], Adrian Cockcroft[5], Yu Teng, Joshua Patterson[2], Gregory Kimball[2], Wes McKinney[4], Weiwei Gong[3], Xiangyao Yu[1]
[1]University of Wisconsin-Madison, [2]NVIDIA, [3]Oracle, [4]Posit PBC, [5]OrionX
siriusdb@cs.wisc.edu

Abstract

The era of GPU-powered data analytics has arrived. In this paper, we argue that recent advances in hardware (e.g., larger GPU memory, faster interconnect and IO, and declining cost) and software (e.g., composable data systems and mature libraries) have removed the key barriers that have limited the wider adoption of GPU data analytics. We present Sirius, a prototype open-source GPU-native SQL engine that offers drop-in acceleration for diverse data systems. Sirius treats GPU as the primary engine and leverages libraries like libcudf for high-performance relational operators. It provides drop-in acceleration for existing databases by leveraging the standard Substrait query representation, replacing the CPU engine without changing the user-facing interface. Sirius achieves 8.3× and 7.4× better cost efficiency on TPC-H and ClickBench, respectively, when integrated with single-node DuckDB, and delivers up to 12.5× speedup when integrated with Apache Doris distributed engine.

1 Introduction

The performance of a SQL engine is ultimately driven by the capabilities of the underlying hardware—especially compute throughput and memory bandwidth. Modern GPUs are rapidly outpacing CPUs on both fronts and have become the default hardware choice for data- and compute-intensive applications such as machine learning. Table 1 compares recent CPU and GPU architectures, highlighting the contrast in core count and memory bandwidth. Given the inherently parallel nature of relational data analytics, GPUs are a perfect fit for future analytical databases.

Despite their great potential, however, GPU-based SQL engines [6, 9, 10, 15, 18, 18] have not yet seen mainstream adoption. Prior efforts, both academic and commercial, have been constrained by the following four key challenges and thereby remain niche solutions.

- **Limited GPU memory capacity**: Up to 288 GB in a latest GPU device in contrast to several TBs in CPUs.
- **Data movement bottleneck**: Traditionally, PCIe has relatively low bandwidth, creating a bottleneck when transferring data between GPU and the rest of the system.
- **Expensive GPU hardware**: High-end GPUs remain significantly more expensive than CPUs and are often in short supply.

*Equal contributions

CIDR'26, Chaminade, USA

ACM ISBN 978-1-4503-XXXX-X/2018/06
https://doi.org/XXXXXXX.XXXXXXX

Table 1: Comparison of CPU and GPU Instances

	Amazon c6a.metal (AMD EPYC CPU)	GH200 (NVIDIA GPU)
Core Count	192 (vCPUs)	14,000+ (CUDA cores)
Memory BW	~400 GB/s	3,000 GB/s (HBM)
Memory Size	384 GB	96 GB (HBM)
Rental Cost	$7.344/h (AWS)	$1.5/h (Lambda Labs)

- **Engineering cost**: Building a full SQL engine optimized for GPUs from the ground up is a major engineering challenge.

In this paper, we argue that the hardware and software foundations are finally in place today to overcome these challenges and enable scalable GPU data analytics. On the hardware side, GPU memory capacity doubles almost every generation, starting from Volta (32 GB), to Ampere (80 GB), Hopper (192 GB), and most recently Blackwell (288 GB). Better interconnects such as PCIe Gen6, NVLink-C2C [13], and GPU Direct [8] significantly reduce the data movement overhead between GPU and other system components. This allows a GPU to process data beyond on-device memory with very fast speed, enabling TBs of analytics even on a single GPU and more with distribution. Meanwhile, GPUs are increasingly affordable and accessible, especially for older generations which are already sufficient for data analytics workloads.

On the software side, the GPU ecosystem has significantly matured. Libraries such as `libcudf` [6] provides high-performance primitives like joins and aggregations that a GPU SQL engine can build upon. The rise of composable data systems [25]—driven by open standards like Substrait [16] for query representation, Apache Arrow and Parquet for columnar data formats—enable greater interoperability. A modern GPU engine can reuse existing components, such as SQL parser and query optimizer, to drastically reduce the complexity of building the system from the ground up.

To demonstrate the feasibility of GPU-native databases, we have built Sirius[1], a prototype open-source SQL engine that uses GPU as the primary execution device and delivers drop-in acceleration across diverse data systems. Sirius integrates seamlessly with existing database systems via the standard Substrait query representation. For example, plugging Sirius into DuckDB causes minimal change to the user-facing interface. Instead of executing the query using DuckDB's CPU engine, the Substrait plan is routed to Sirius for GPU-native execution. Sirius builds on GPU libraries such as `libcudf` [6], `RMM` [14], and `NCCL` [11], reusing optimized implementations of core relational operators like joins, filters, aggregations, and data shuffle. Thanks to its modular design, Sirius also allows developers to easily switch the operator implementation between these GPU libraries and custom CUDA kernels. This flexibility

[1]https://github.com/sirius-db/sirius

Outline

Part I → Why GPU Databases?

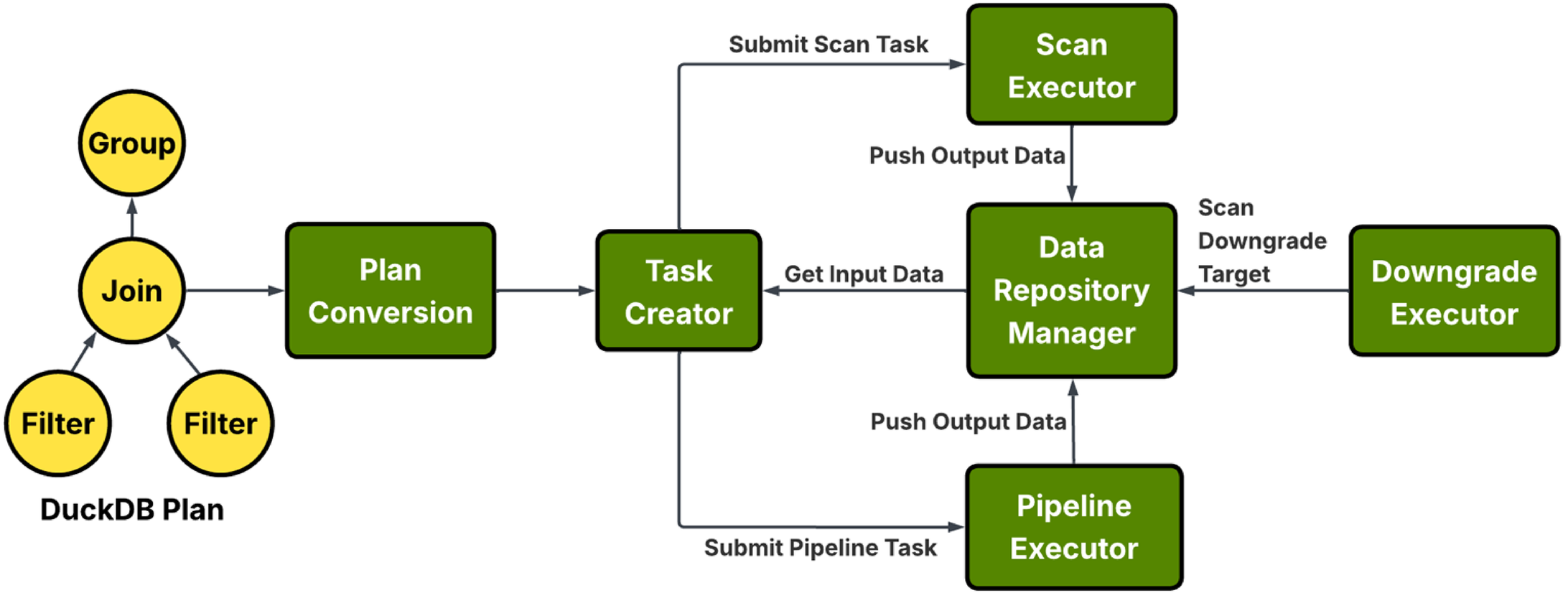
- Hardware trend
- Software trend

Part II → Sirius: A GPU-Native SQL Engine

- Design
- Evaluation

Future Direction

What's Next?



Inspired by [1], an upcoming framework in Sirius will enable acceleration at TB's scale.

To be announced at GTC 2026 (March 16-19)

[1] Aramburu, Malpica, et al. *Theseus: A Distributed and Scalable GPU-Accelerated Query Processing Engine Optimized for Efficient Data Movement*, **arXiv 2025**

Future Direction

- Out-of-core execution
- Supporting GPU Reader
- Expanding SQL Coverage
  - E.g., UDF, LIST, nested structures, ASOF join, vector data type, etc.
- DuckDB Community Extension

Sirius Contributors

Jointly Developed by UW-Madison and NVIDIA

NVIDIA

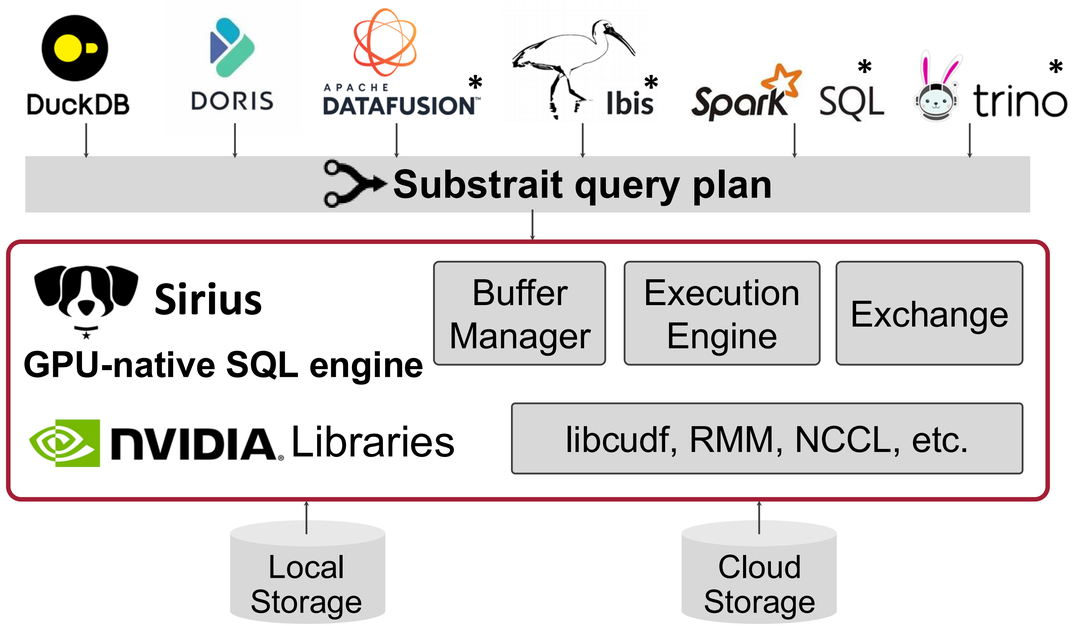
- Bobbi Yogatama
- Felipe Aramburu
- Amin Aramoon
- William Malpica
- Joost Hoozemans
- Matthijs Brobbel
- Dhruv Vats
- Johan Peltenburg
- Rodrigo Aramburu
- Joshua Patterson

University of Wisconsin-Madison

- Xiangyao Yu
- Yifei Yang
- Kevin Kristensen
- Devesh Sarda
- Abigale Kim

We welcome contributions both from Academia and Industry!!!

Sirius: GPU-Native Execution Engine



Project website: sirius-db.com

Open-sourcing on github:
https://github.com/sirius-db/sirius

Interested in Sirius? Contact

siriusdb@cs.wisc.edu

[1] Bobbi Yogatama*, Yifei Yang*, et al. *Rethinking Analytical Processing in the GPU Era*, **CIDR 2026**
[2] Aramburu, Malpica, et al. *Theseus: A Distributed and Scalable GPU-Accelerated Query Processing Engine Optimized for Efficient Data Movement*, **arXiv 2025**
[3] Yifei Yang, Xiangyao Yu, *Accelerate Distributed Joins with Predicate Transfer*, **SIGMOD 2025**
[4] Junyi Zhao, et al. *Debunking the Myth of Join Ordering: Toward Robust SQL Analytics*, **SIGMOD 2025**
[5] Yifei Yang, et al. *Predicate Transfer: Efficient Pre-Filtering on Multi-Join Queries*, **CIDR 2024**
[6] Bobbi Yogatama, Weiwei Gong, Xiangyao Yu. *Scaling your Hybrid CPU-GPU DBMS to Multiple GPUs,* **VLDB 2024**
[7] Bobbi Yogatama et al. *Accelerating User-Defined Aggregate Functions with Block-wide Execution and JIT Compilation on GPUs*, **DaMoN@SIGMOD 2023**
[8] Bobbi Yogatama, Weiwei Gong, Xiangyao Yu. *Orchestrating data placement and query execution in heterogeneous CPU-GPU DBMS*, **VLDB 2022**
[9] Anil Shanbhag*, Bobbi Yogatama*, Xiangyao Yu, and Samuel Madden. *Tile-based Lightweight Integer Compression in GPU*, **SIGMOD 2022**
[10] Anil Shanbhag, Sam Madden, Xiangyao Yu. *A Study of the Fundamental Performance Characteristics of GPUs and CPUs for Database Analytics*, **SIGMOD 2020**