Do GPUs Really Need New Tabular File Formats?

DaMoN 2026 [Accelerator, Storage I/O]

Jigao Luo[1], Qi Chen[1], Carsten Binnig[1, 2]

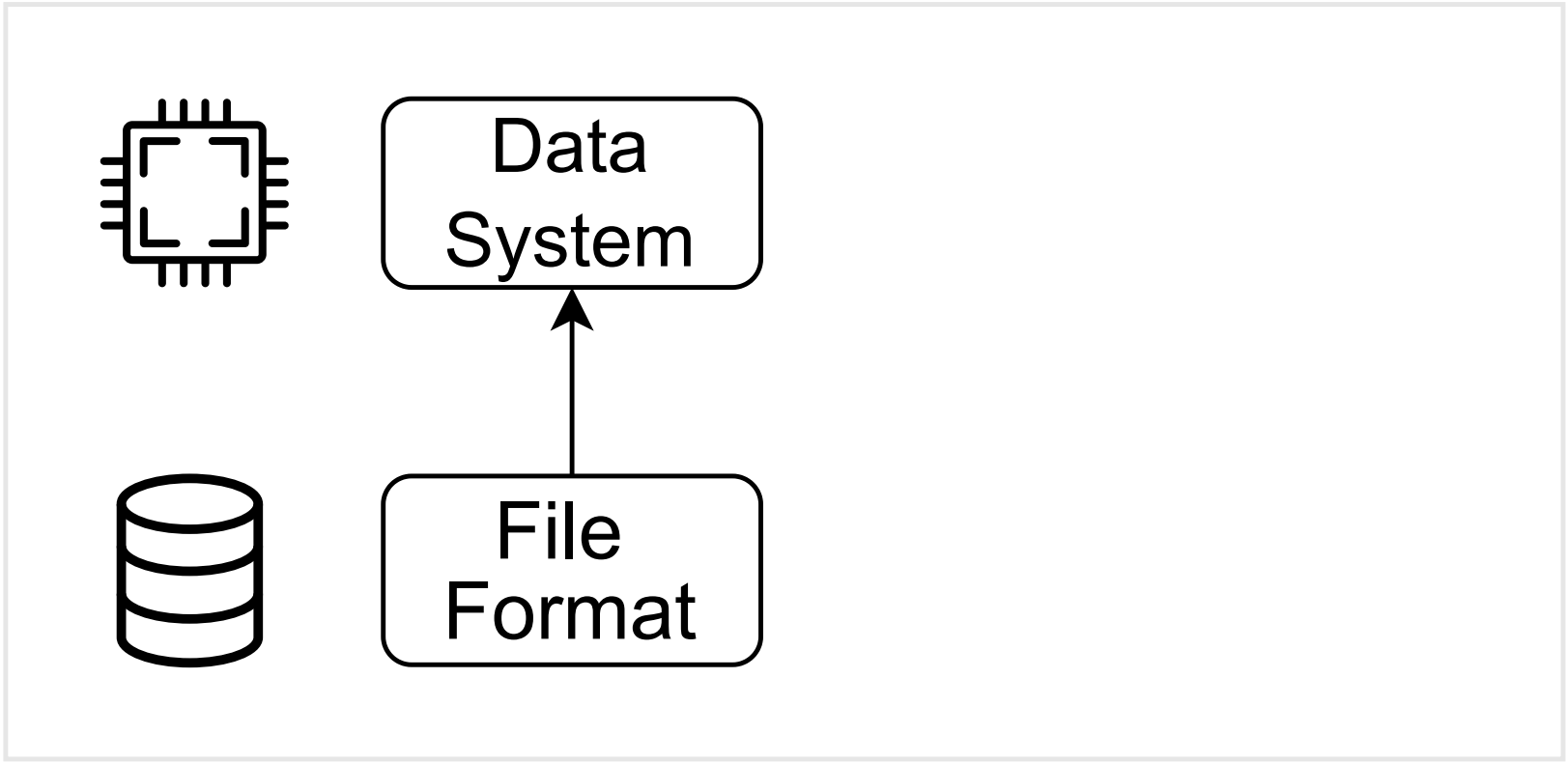
2026-06-01

[1]Systems Group, Technische Universität Darmstadt
[2]DFKI

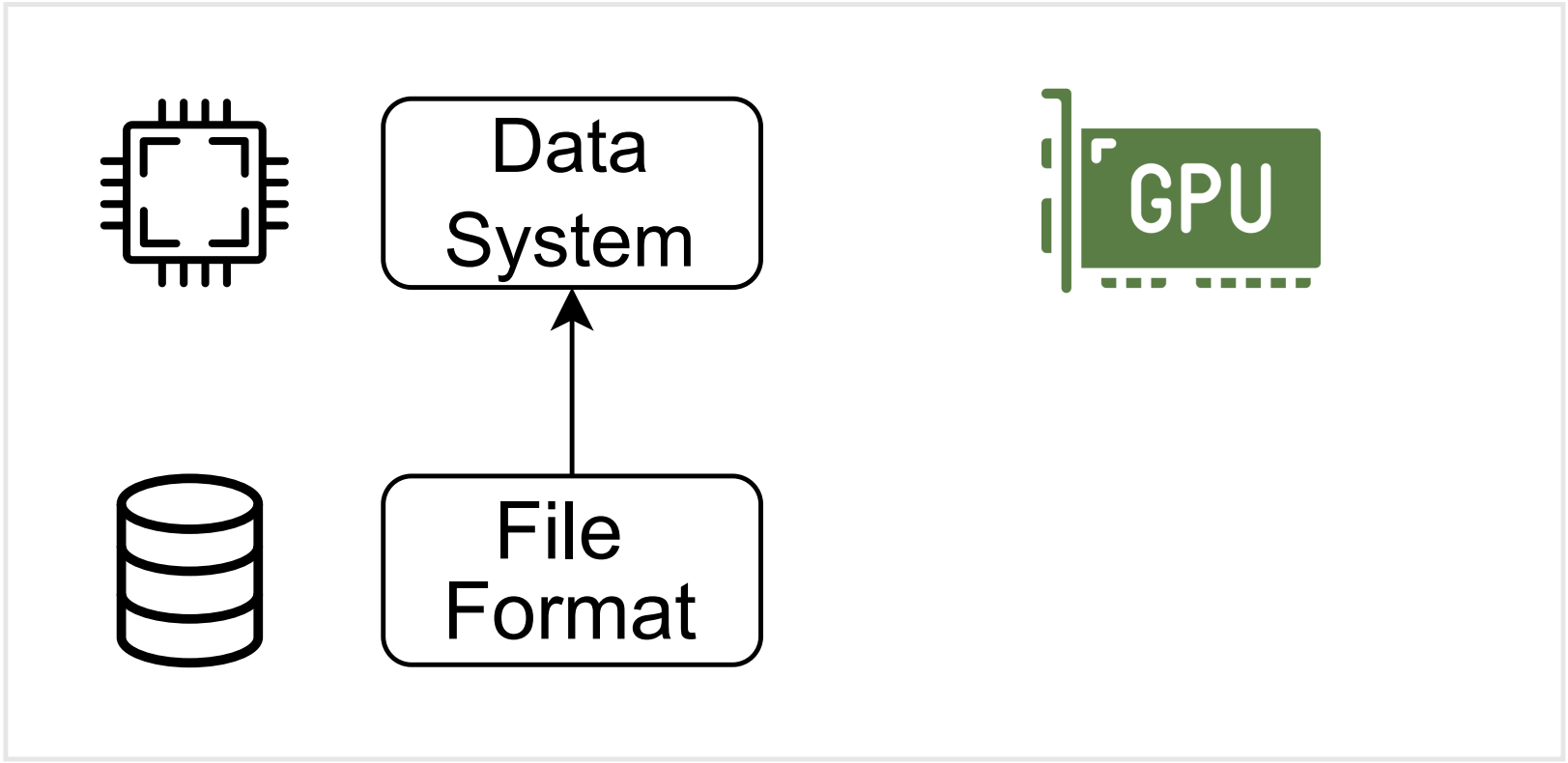
TECHNISCHE UNIVERSITÄT DARMSTADT

SYSTEMS

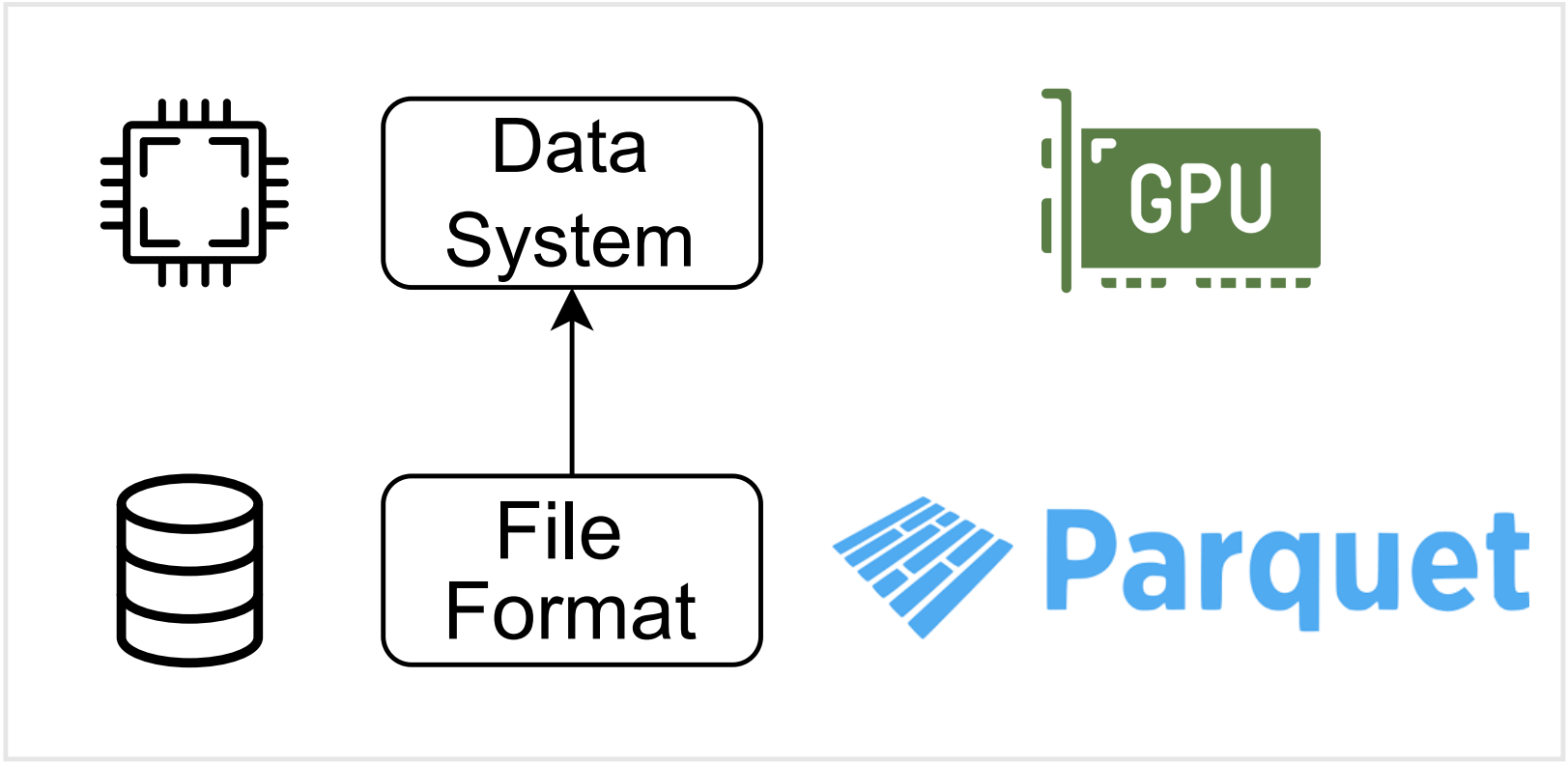
Data Processing: GPUs & File Formats



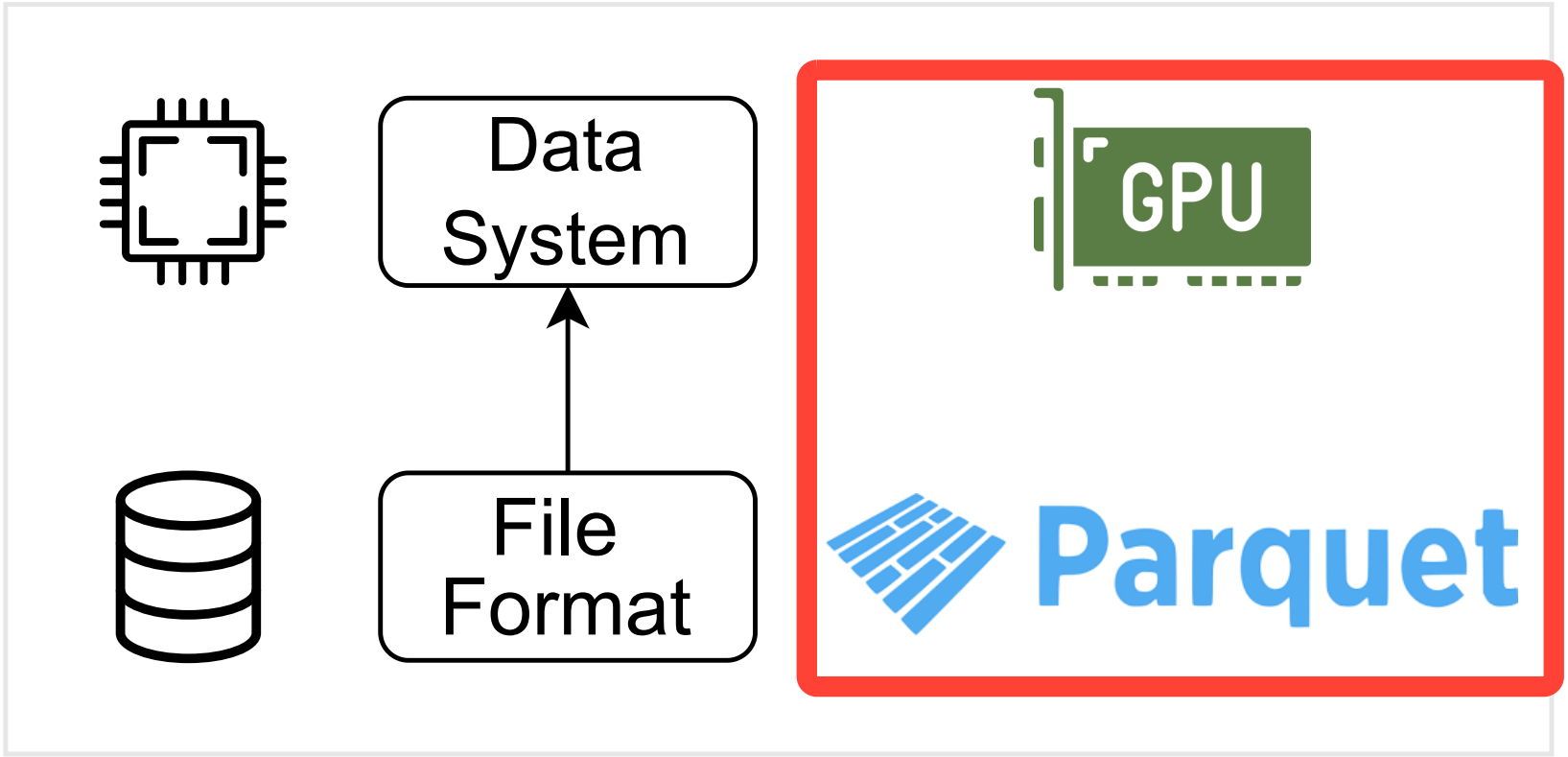
Data Processing: GPUs & File Formats



Data Processing: GPUs & File Formats



Data Processing: GPUs & File Formats

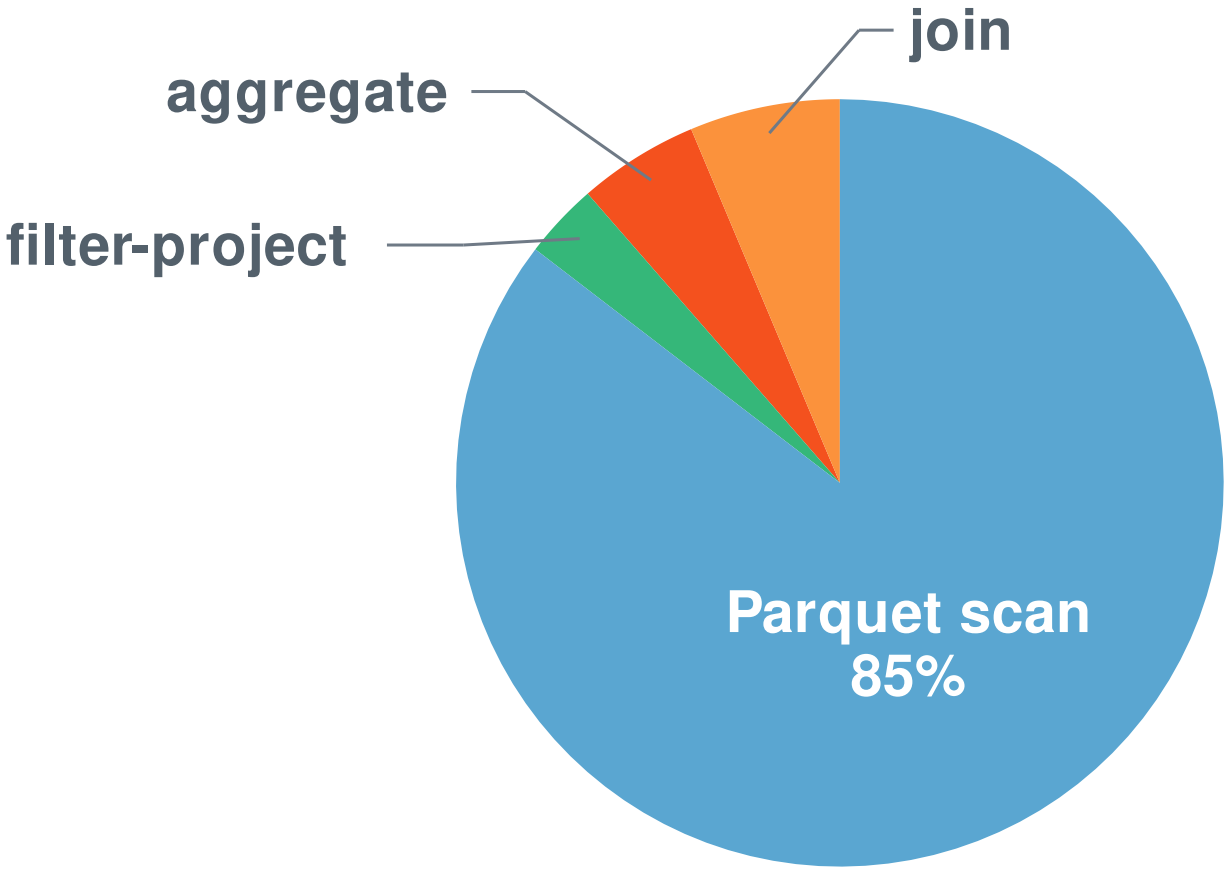


Problem: GPU Bottleneck With Parquet

Is Parquet Bad for GPUs?

Problem: GPU Bottleneck With Parquet

Is Parquet Bad for GPUs?



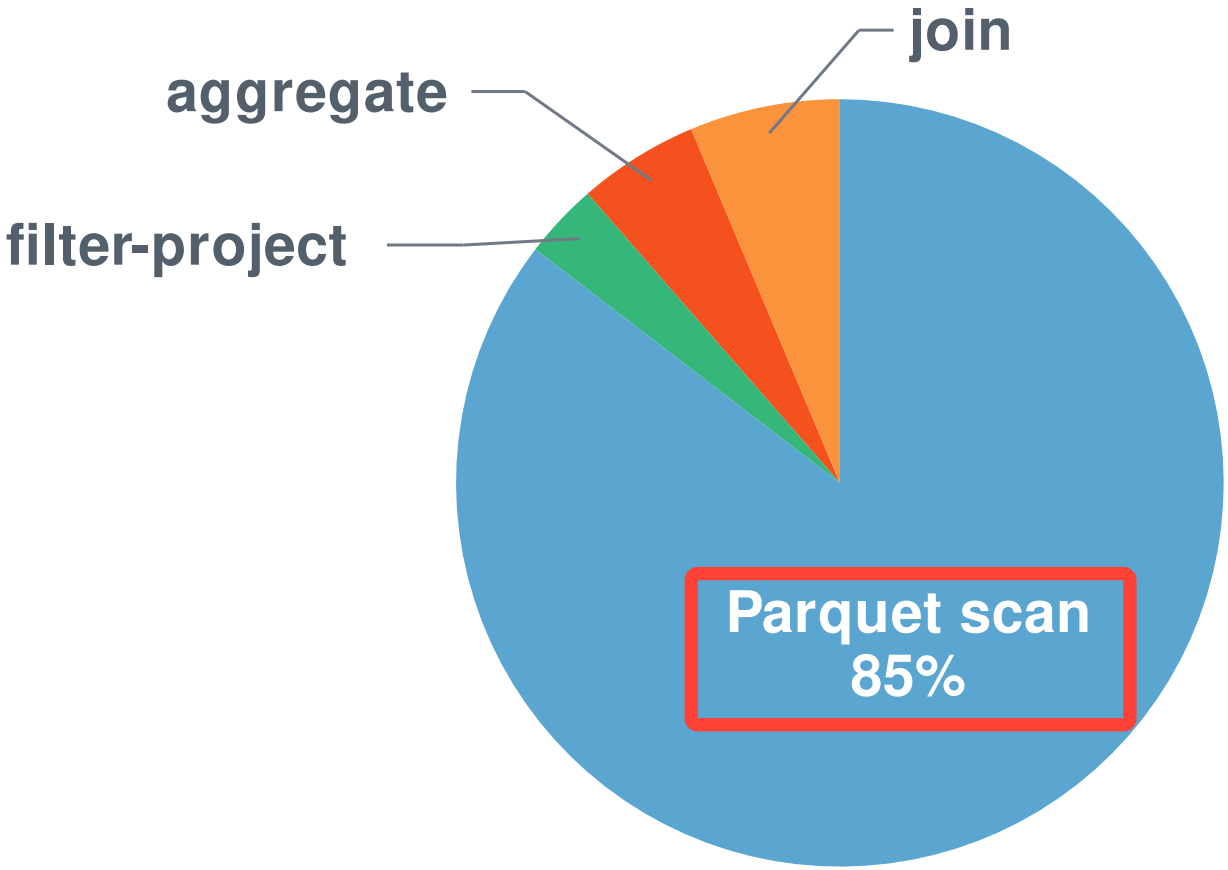
GPU TPC-H SF100 runtime breakdown.

Source: Accelerating Velox with RAPIDS cuDF, Velox-cuDF Team, 2025

Problem: GPU Bottleneck With Parquet

Is Parquet Bad for GPUs?

- **85%** in Parquet Scan



GPU TPC-H SF100 runtime breakdown.

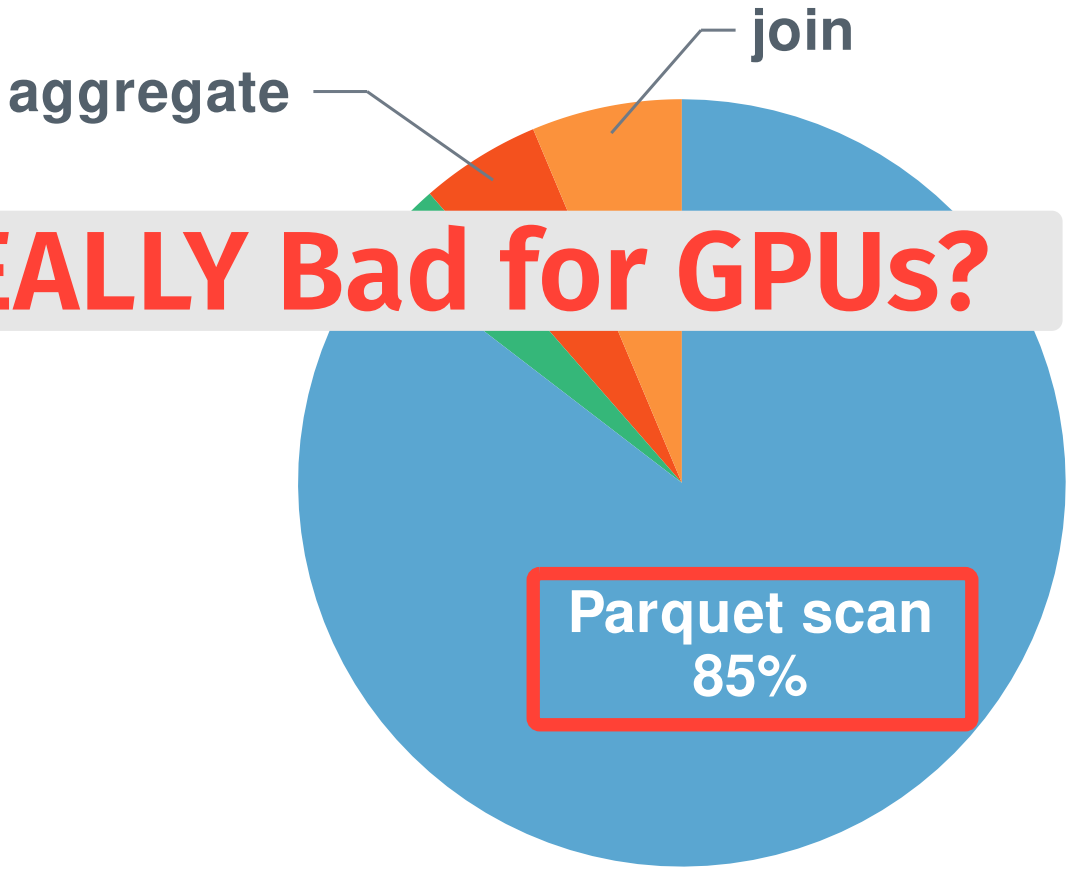
Source: Accelerating Velox with RAPIDS cuDF, Velox-cuDF Team, 2025

Problem: GPU Bottleneck With Parquet

Is Parquet REALLY Bad for GPUs?

Is Parquet Bad for GPUs?

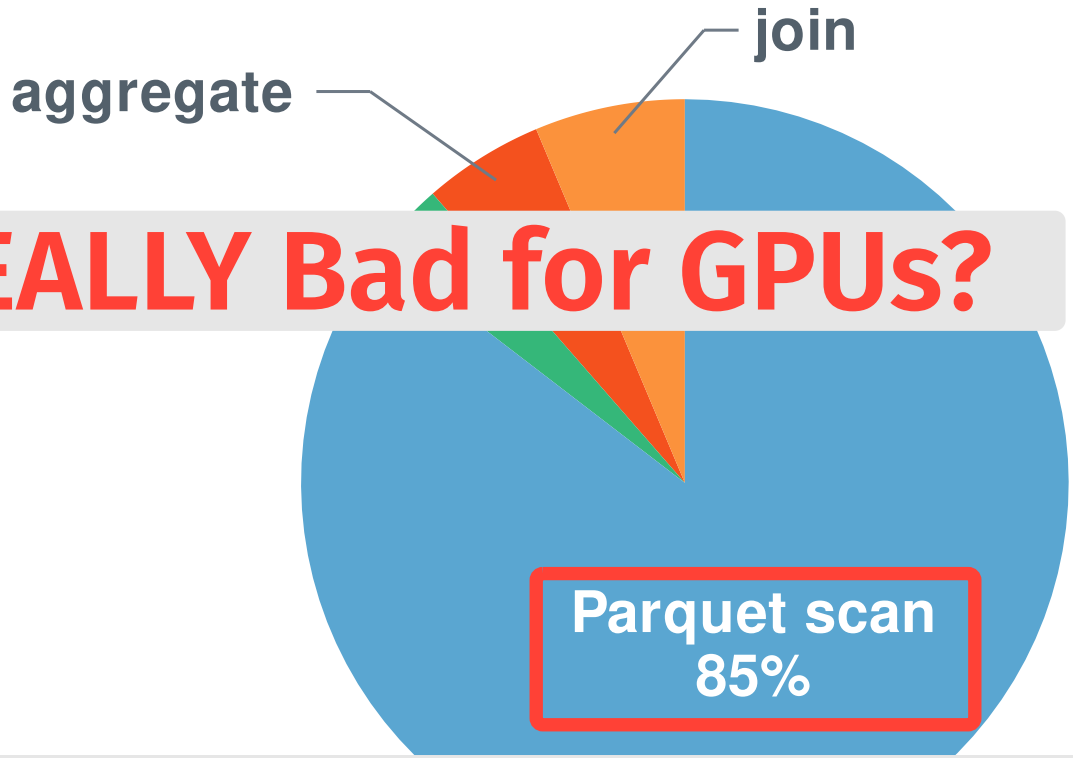
- **85%** in Parquet Scan



GPU TPC-H SF100 runtime breakdown.

Source: Accelerating Velox with RAPIDS cuDF, Velox-cuDF Team, 2025

Problem: GPU Bottleneck With Parquet



Is Parquet REALLY Bad for GPUs?

Is Parquet Bad for GPUs?

- **85%** in Parquet Scan

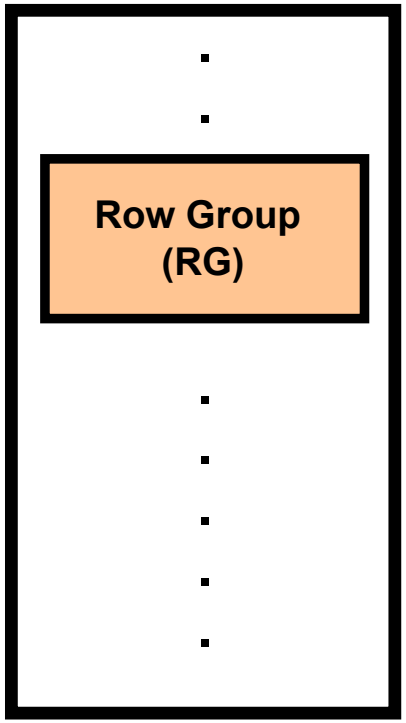
Do GPUs REALLY Need New File Formats?

GPU TPC-H SF100 runtime breakdown.

Source: Accelerating Velox with RAPIDS cuDF, Velox-cuDF Team, 2025

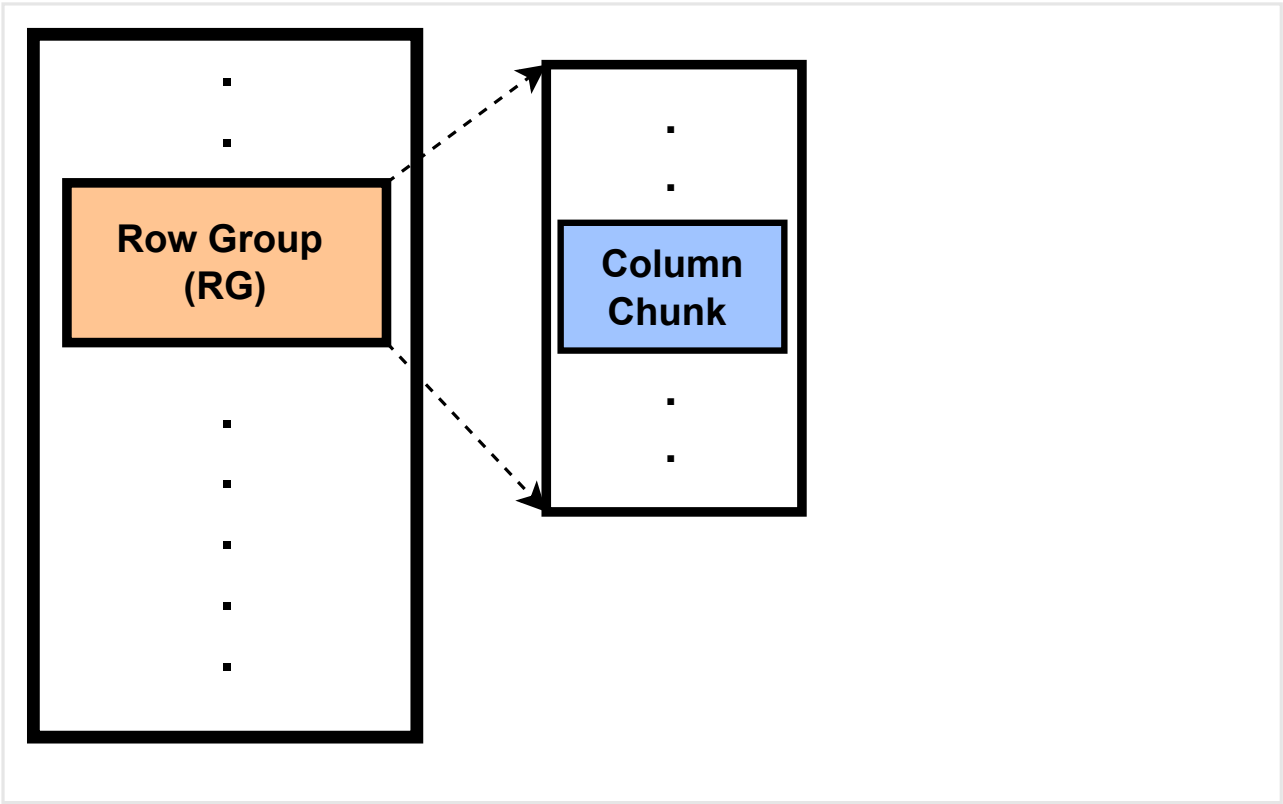
Background: Apache Parquet

- Row group



Background: Apache Parquet

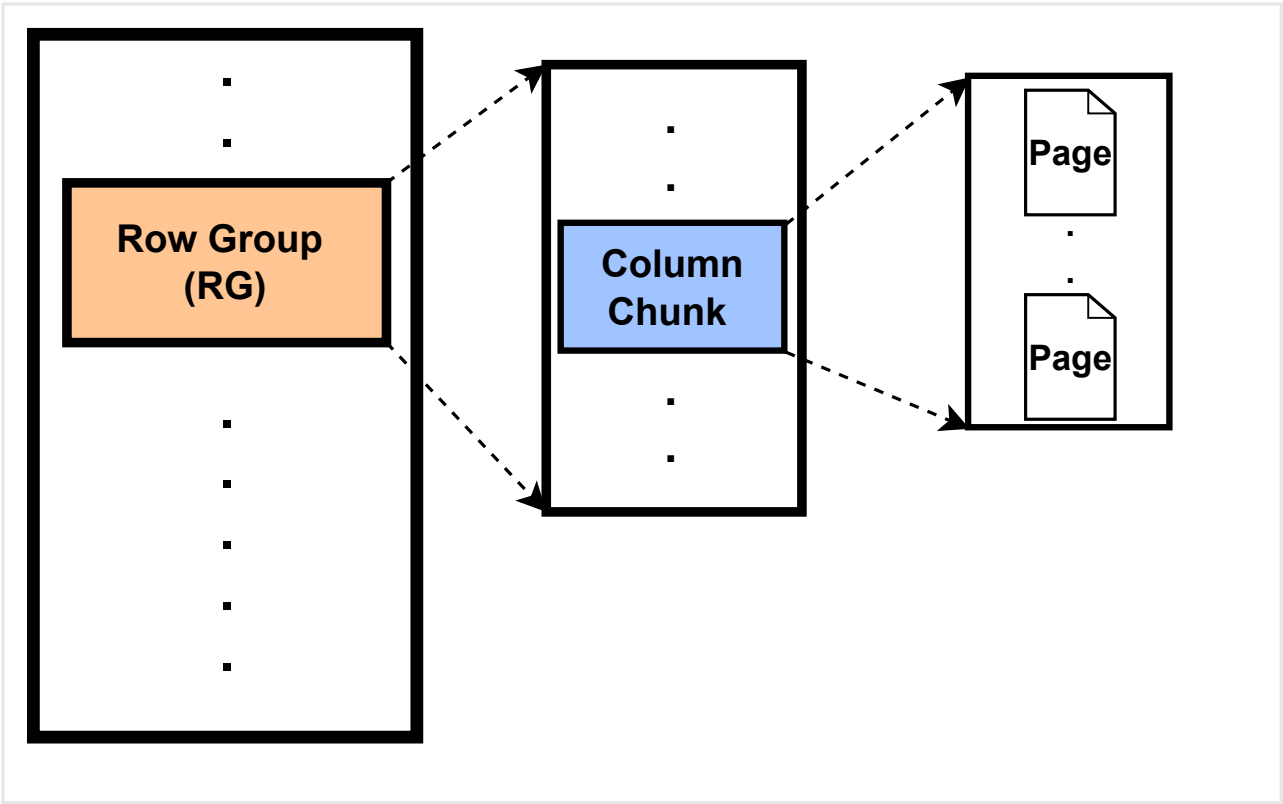
- Row group
- Column chunk



Parquet

Background: Apache Parquet

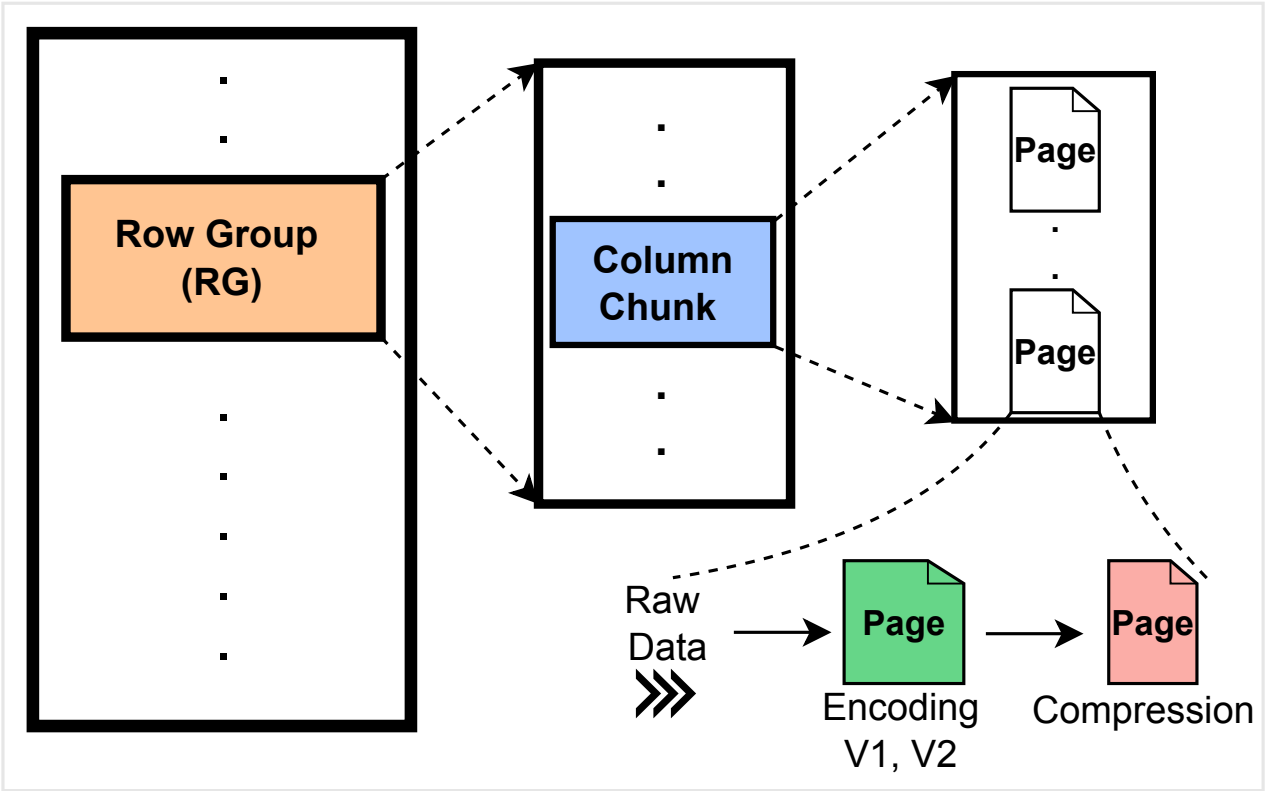
- Row group
- Column chunk
- Pages:



Parquet

Background: Apache Parquet

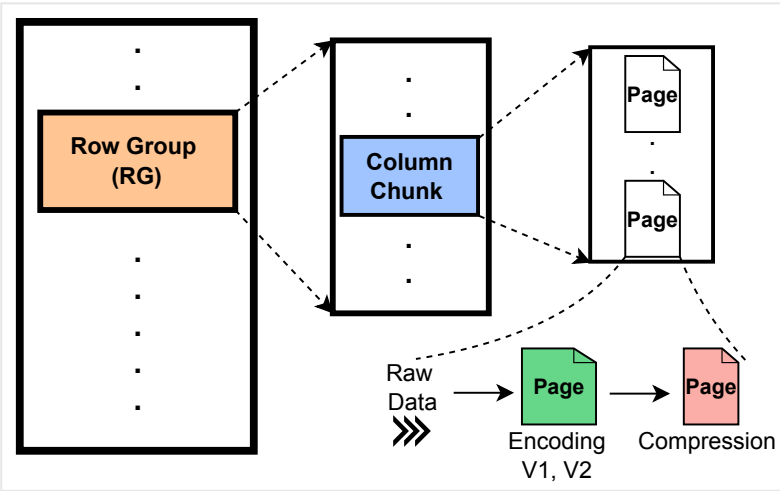
- Row group
- Column chunk
- Pages:
  - Encoded
  - Compressed



Parquet

Parquet Configuration Defaults for CPUs

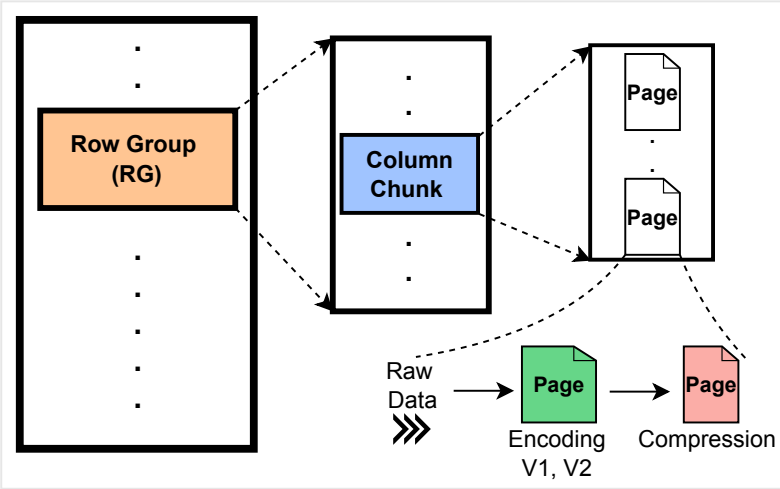
CPU Parquet Config. Defaults in **DuckDB**:



Parquet Configuration Defaults for CPUs

CPU Parquet Config. Defaults in DuckDB:

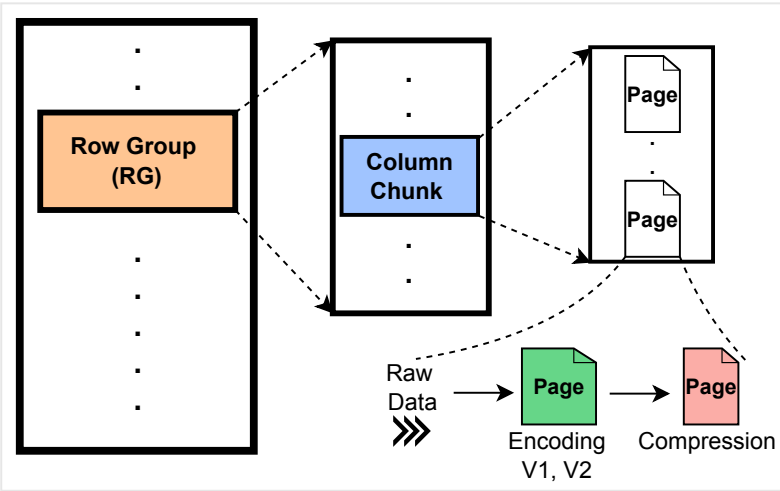
- Row group: 122880 rows



Parquet Configuration Defaults for CPUs

CPU Parquet Config. Defaults in DuckDB:

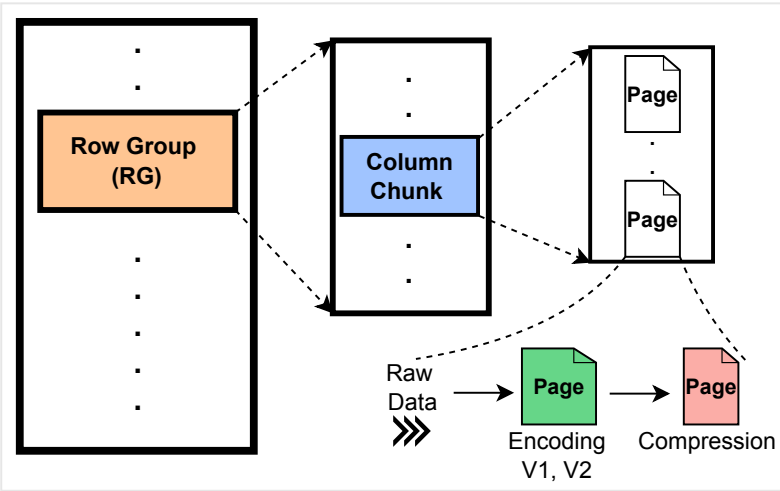
- Row group: 122880 rows
- Column chunk: 1 page only



Parquet Configuration Defaults for CPUs

CPU Parquet Config. Defaults in **DuckDB**:

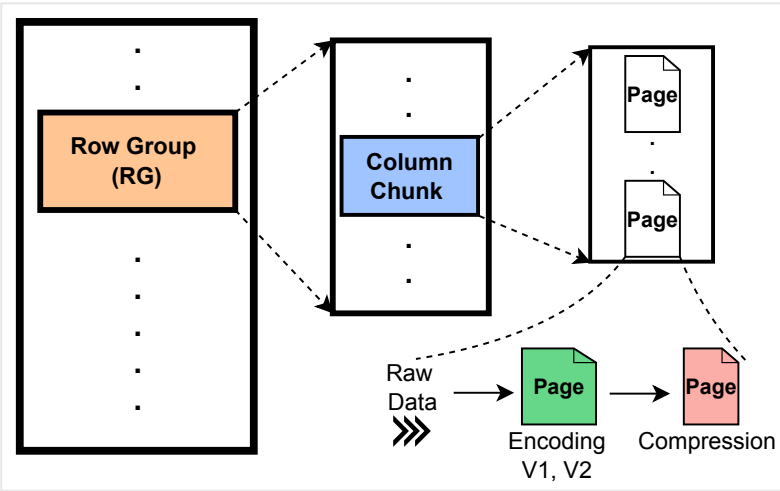
- Row group: 122880 rows
- Column chunk: 1 page only
- Encoding: V1 only



Parquet Configuration Defaults for CPUs

CPU Parquet Config. Defaults in DuckDB:

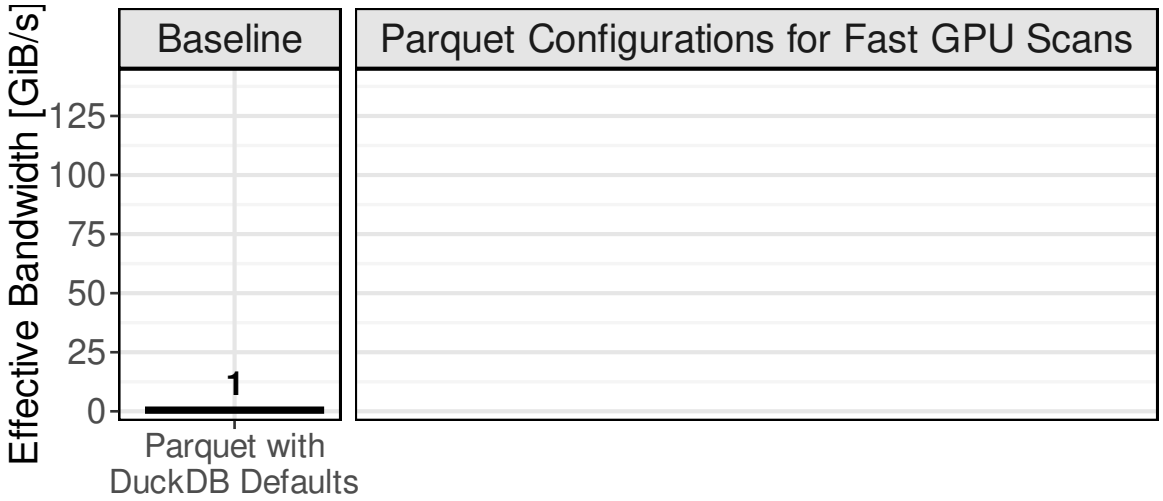
- Row group: 122880 rows
- Column chunk: 1 page only
- Encoding: V1 only
- Compression: used even without size reduction



Issue: CPU Defaults Config. ≠ GPU Optimal Config.

CPU Parquet Config. Defaults in **DuckDB**:

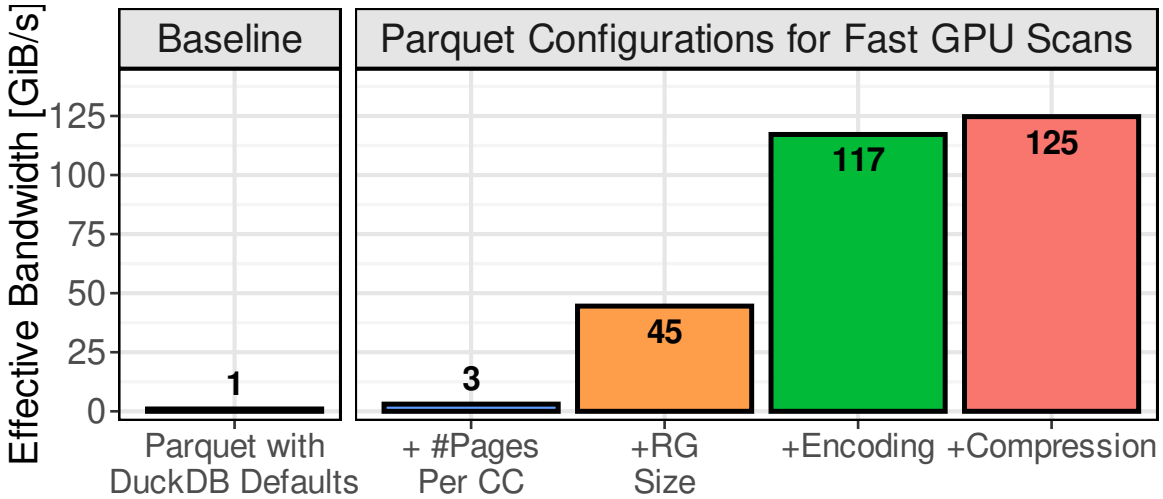
- Row group: 122880 rows
- Column chunk: 1 page only
- Encoding: V1 only
- Compression: used even without size reduction



Issue: CPU Defaults Config. ≠ GPU Optimal Config.

CPU Parquet Config. Defaults in DuckDB:

- Row group: 122880 rows
- Column chunk: 1 page only
- Encoding: V1 only
- Compression: used even without size reduction

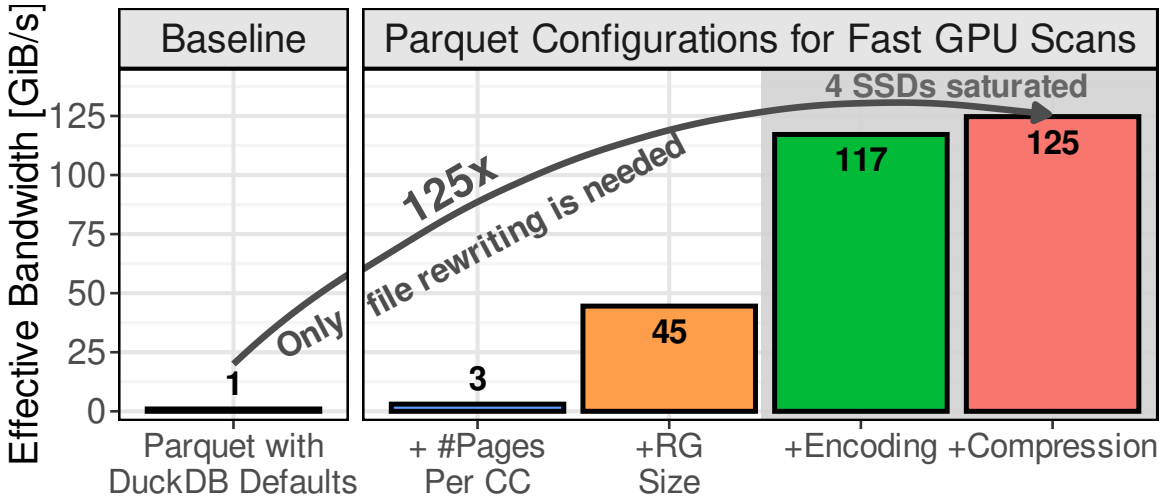


CPU Defaults Config. ≠ GPU Optimal Config.

Issue: CPU Defaults Config. ≠ GPU Optimal Config.

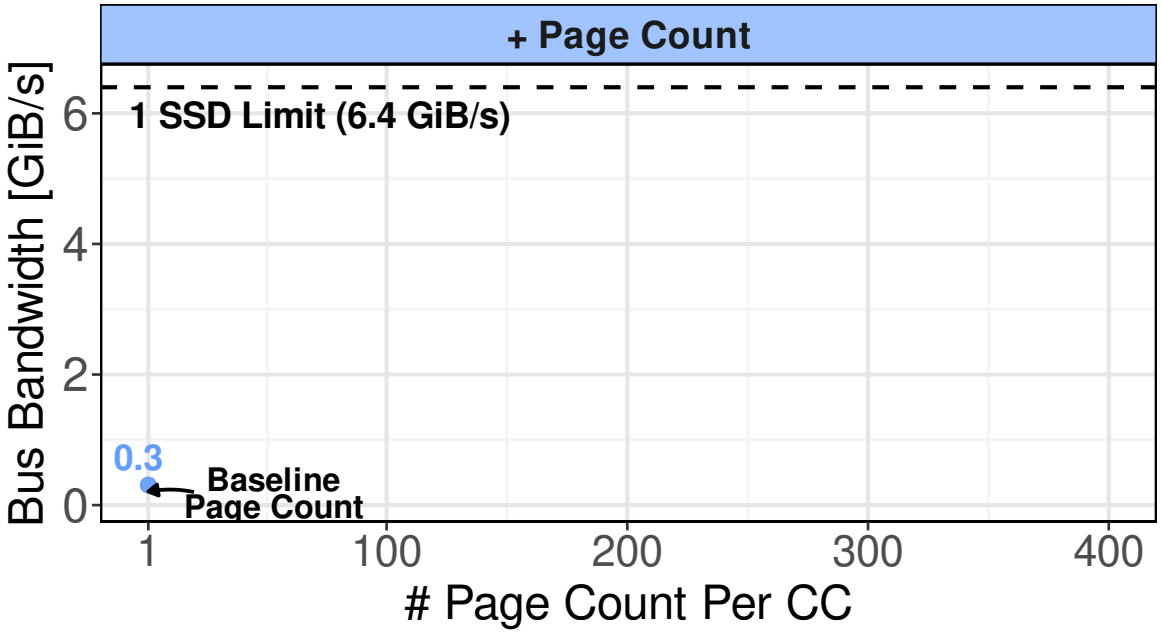
CPU Parquet Config. Defaults in **DuckDB**:

- Row group: 122880 rows
- Column chunk: 1 page only
- Encoding: V1 only
- Compression: used even without size reduction



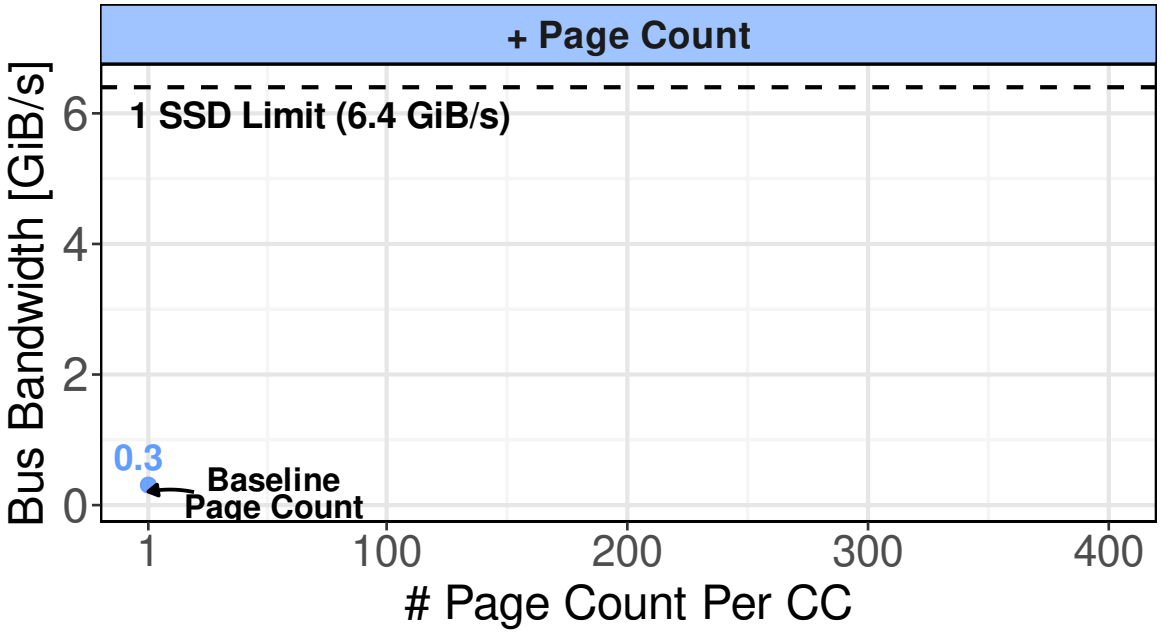
Config. 1: Page Count Per Column Chunk

- Baseline of **1** due to CPU parallelism



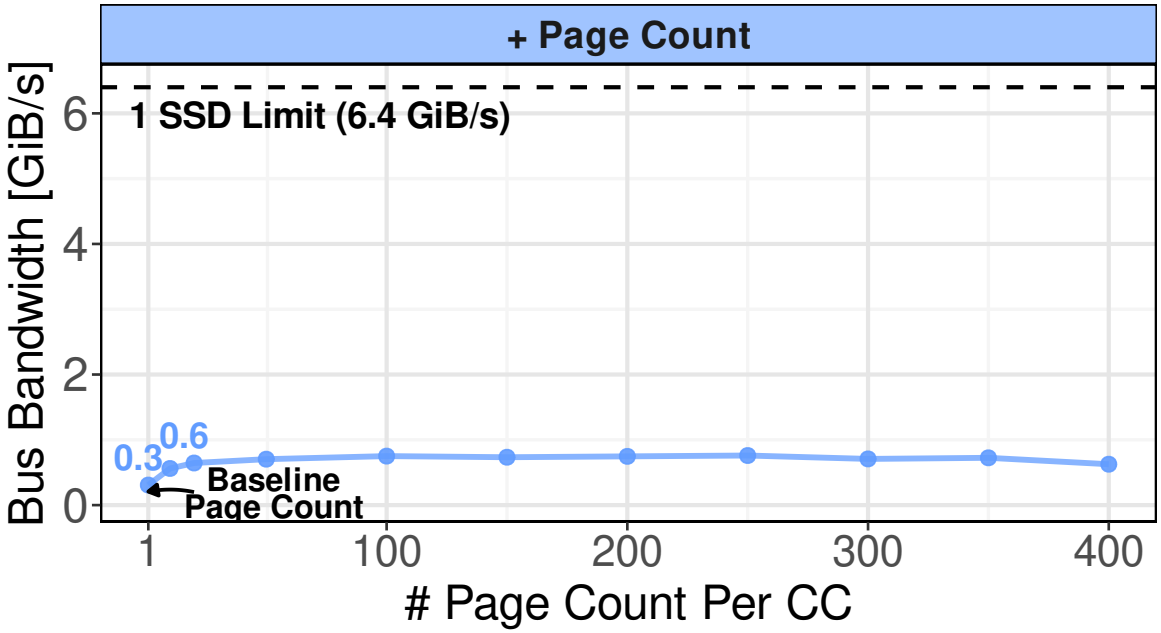
Config. 1: Page Count Per Column Chunk

- Baseline of **1** due to CPU parallelism
- #pages mapped to GPU kernel parameter



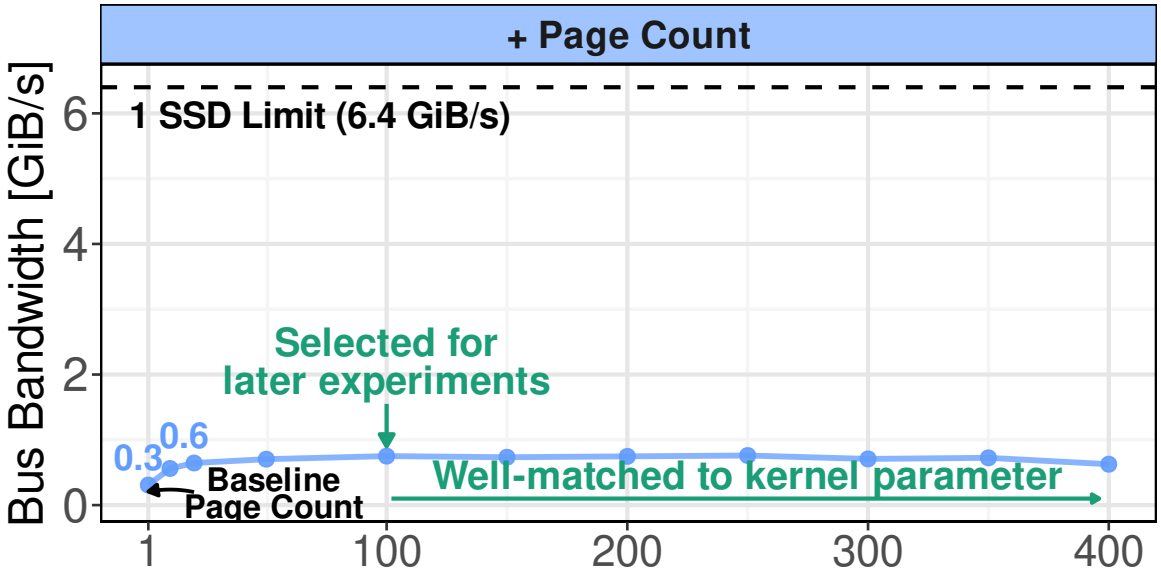
Config. 1: Page Count Per Column Chunk

- Baseline of **1** due to CPU parallelism
- #pages mapped to GPU kernel parameter
- Too few pages underutilize GPU



Config. 1: Page Count Per Column Chunk

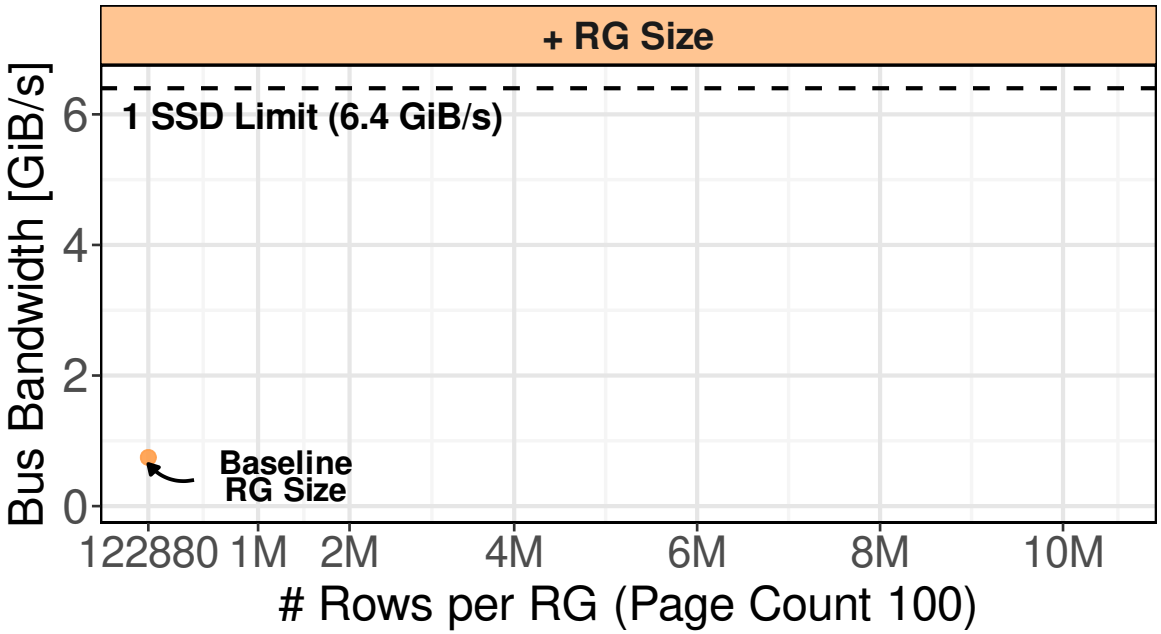
- Baseline of **1** due to CPU parallelism
- #pages mapped to GPU kernel parameter
- Too few pages underutilize GPU



Insight 1: Increase page count to match the recommended GPU kernel parameter

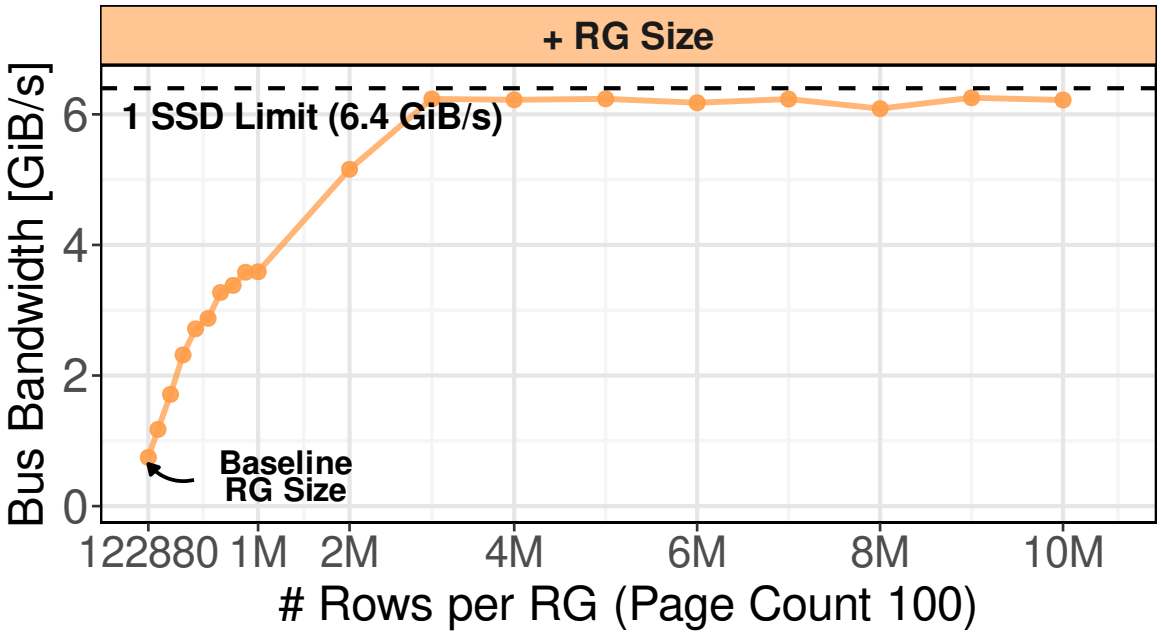
Config. 2: Row Group Size

- GPU I/O stack differs from CPU's



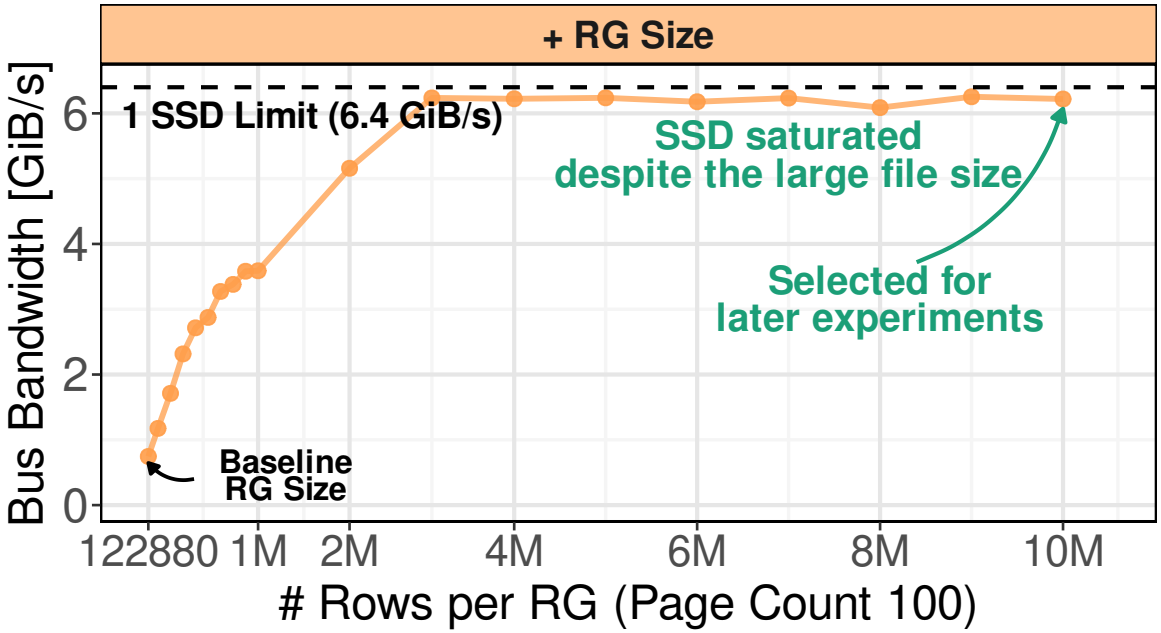
Config. 2: Row Group Size

- GPU I/O stack differs from CPU's
- Larger RG improve bandwidth



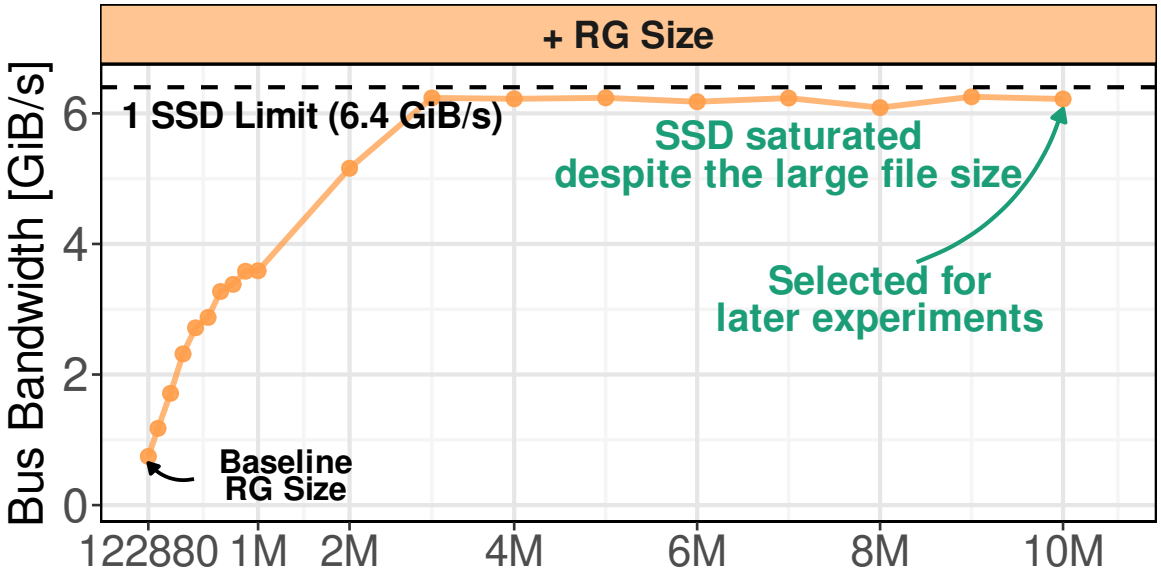
Config. 2: Row Group Size

- GPU I/O stack differs from CPU's
- Larger RG improve bandwidth



Config. 2: Row Group Size

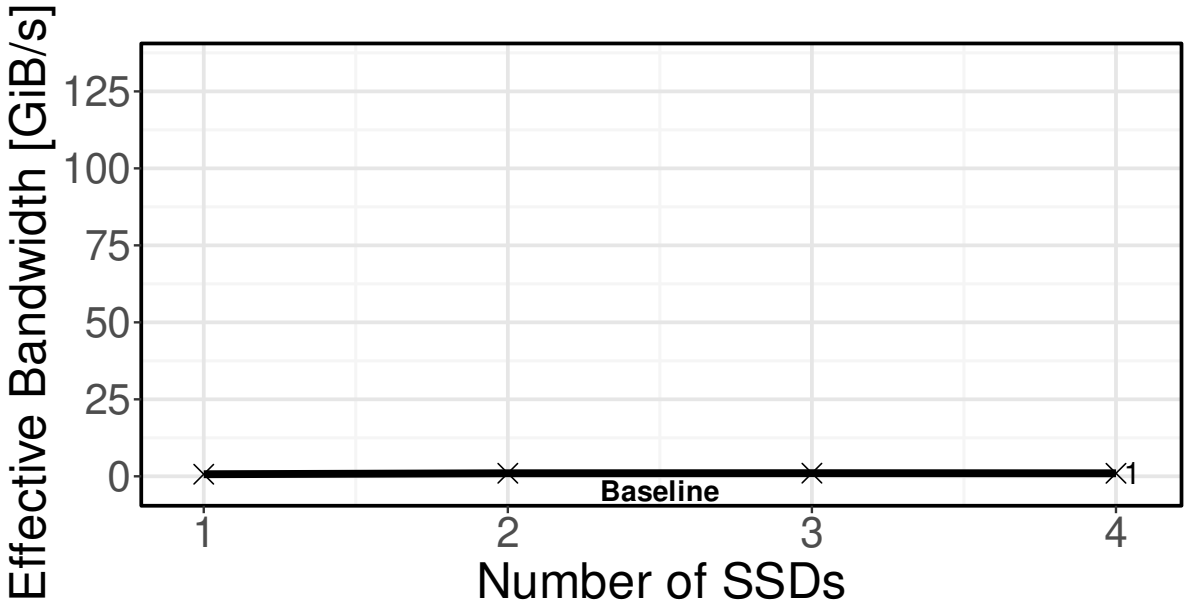
- GPU I/O stack differs from CPU's
- Larger RG improve bandwidth



Insight 2: Million-row RG sizes are preferred for GPU I/O stacks

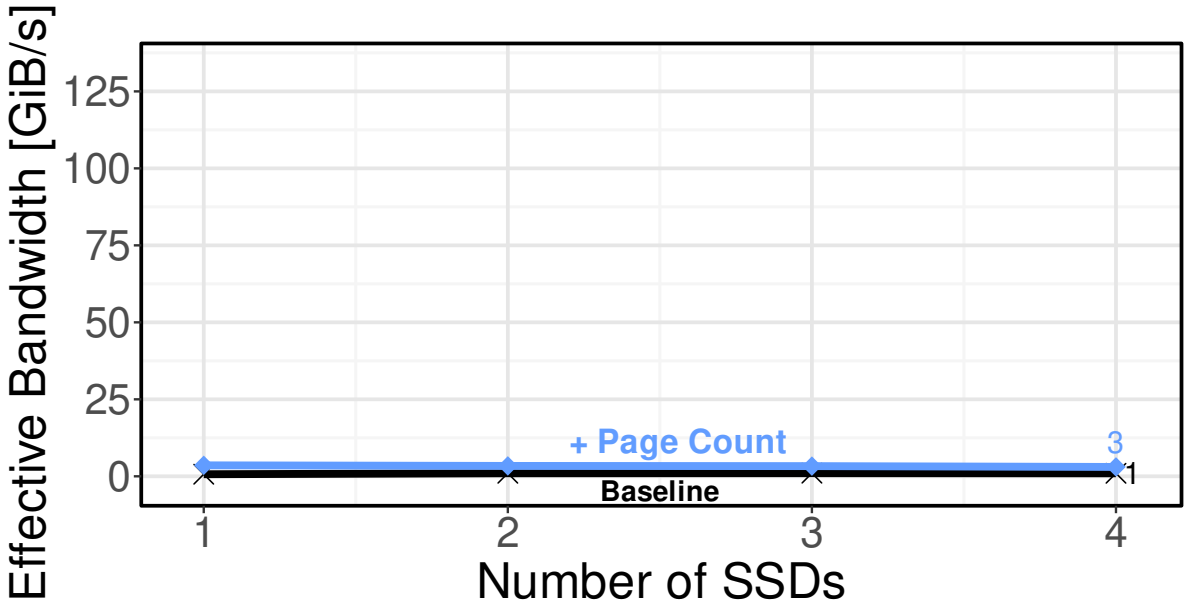
Reading 4 SSDs

- Single SSD saturated
- Scaling to multiple SSDs



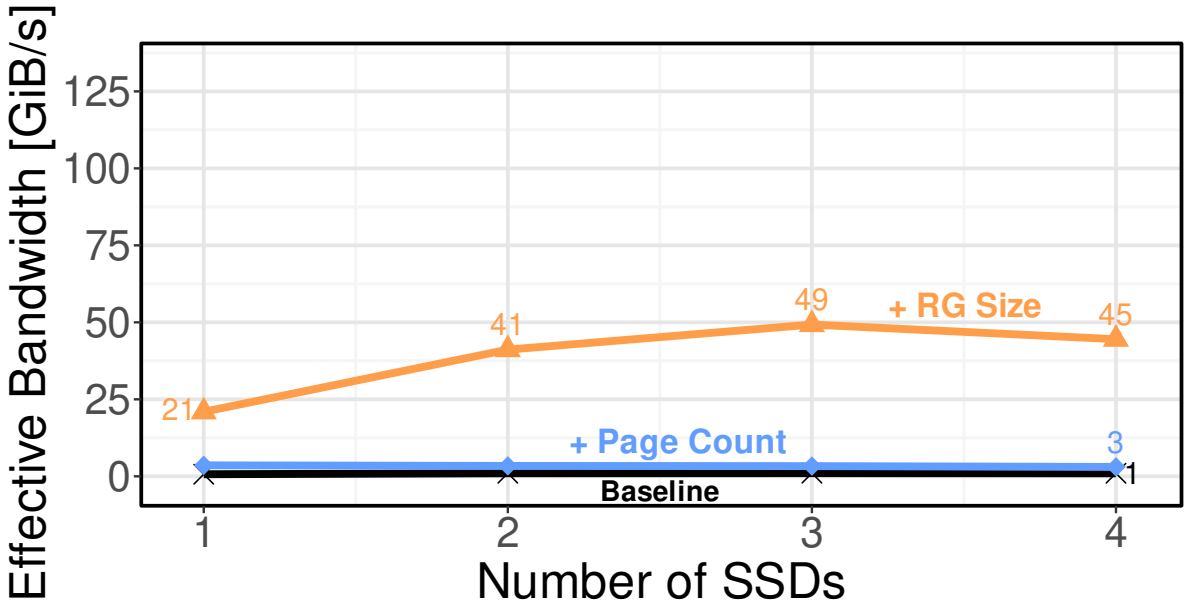
Reading 4 SSDs

- Single SSD saturated
- Scaling to multiple SSDs



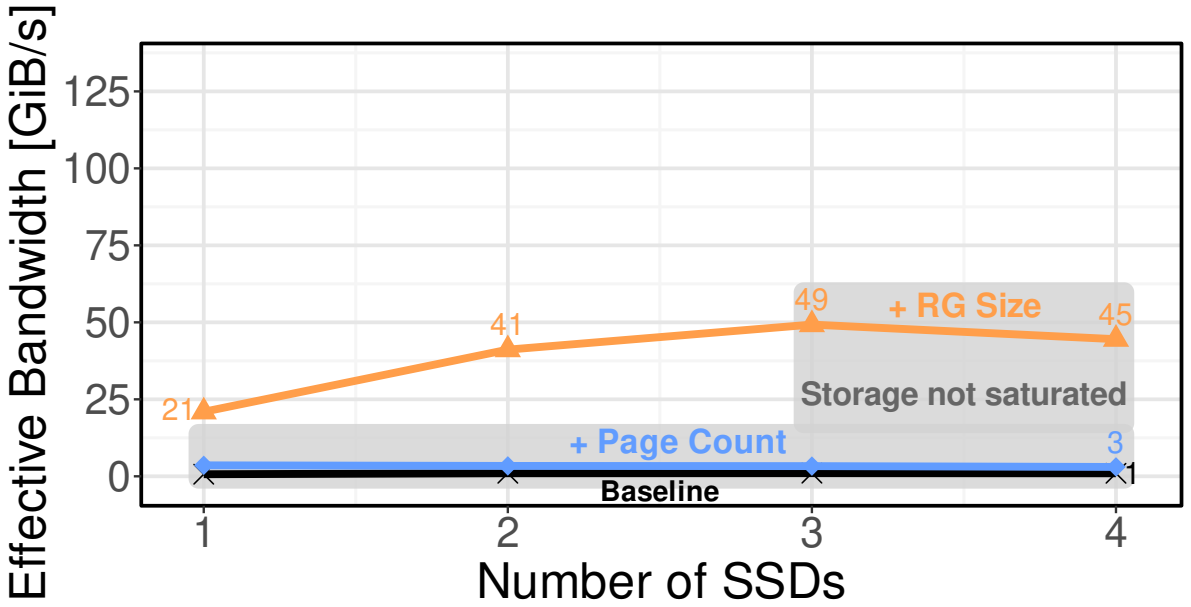
Reading 4 SSDs

- Single SSD saturated
- Scaling to multiple SSDs



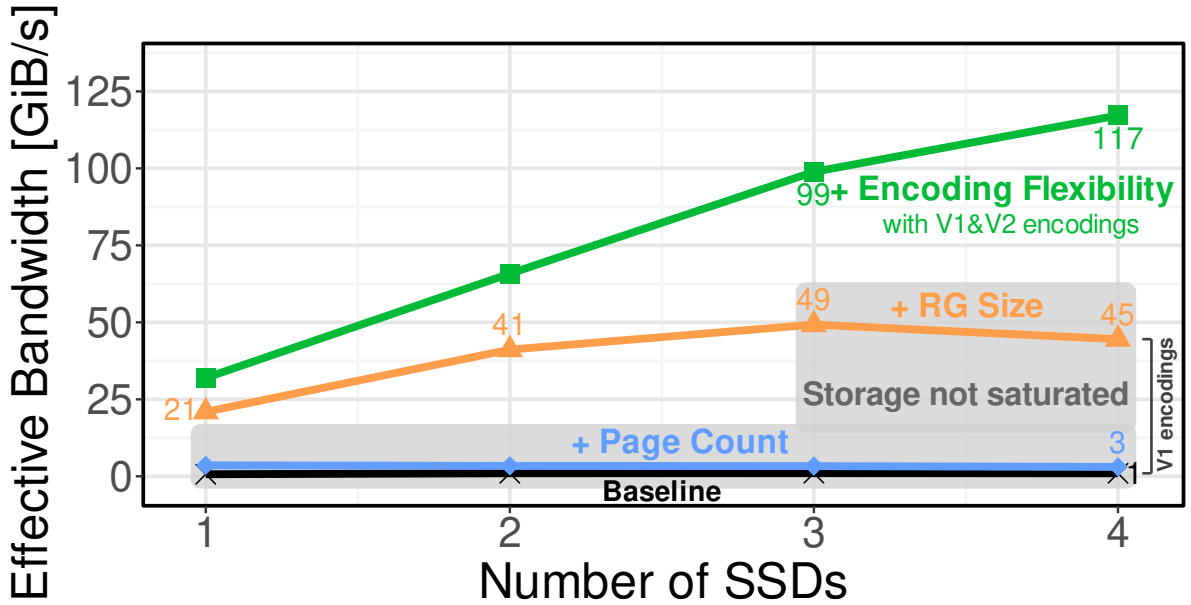
Reading 4 SSDs

- Single SSD saturated
- Scaling to multiple SSDs



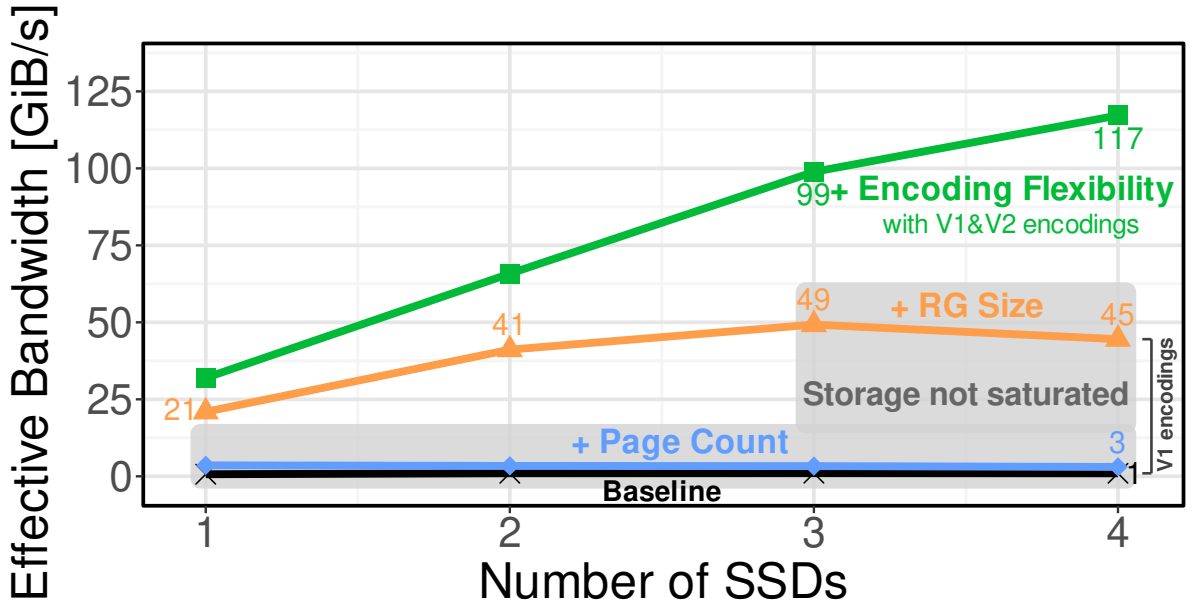
Config. 3: Encoding Flexibility

- Consider both V1 and V2



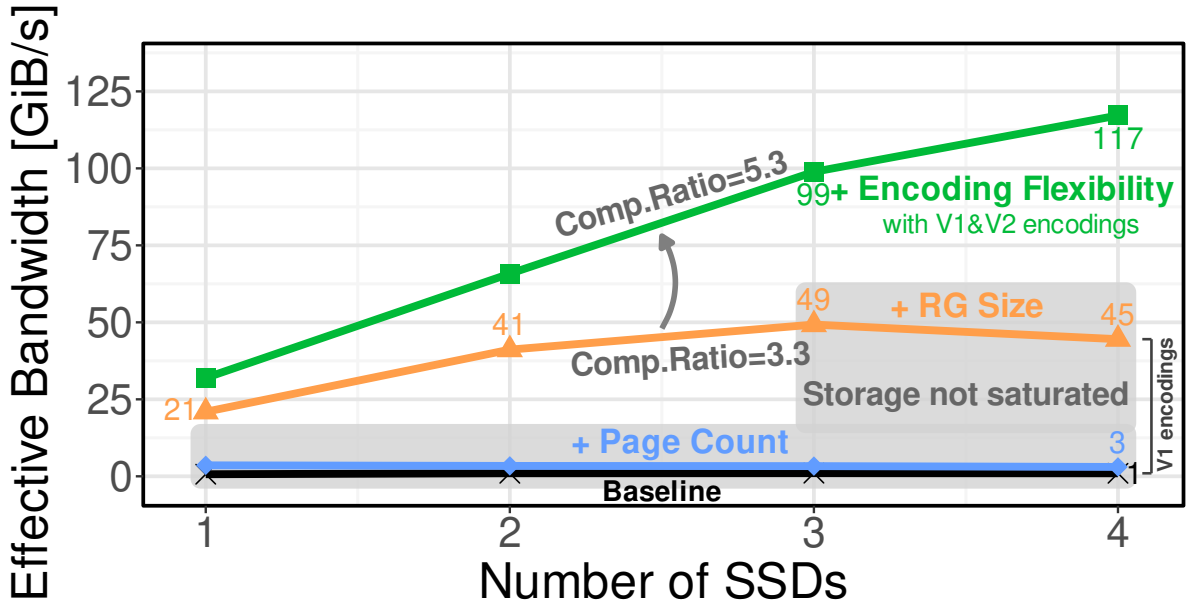
Config. 3: Encoding Flexibility

- Consider both V1 and V2
- Finalize the encoding producing the smallest size



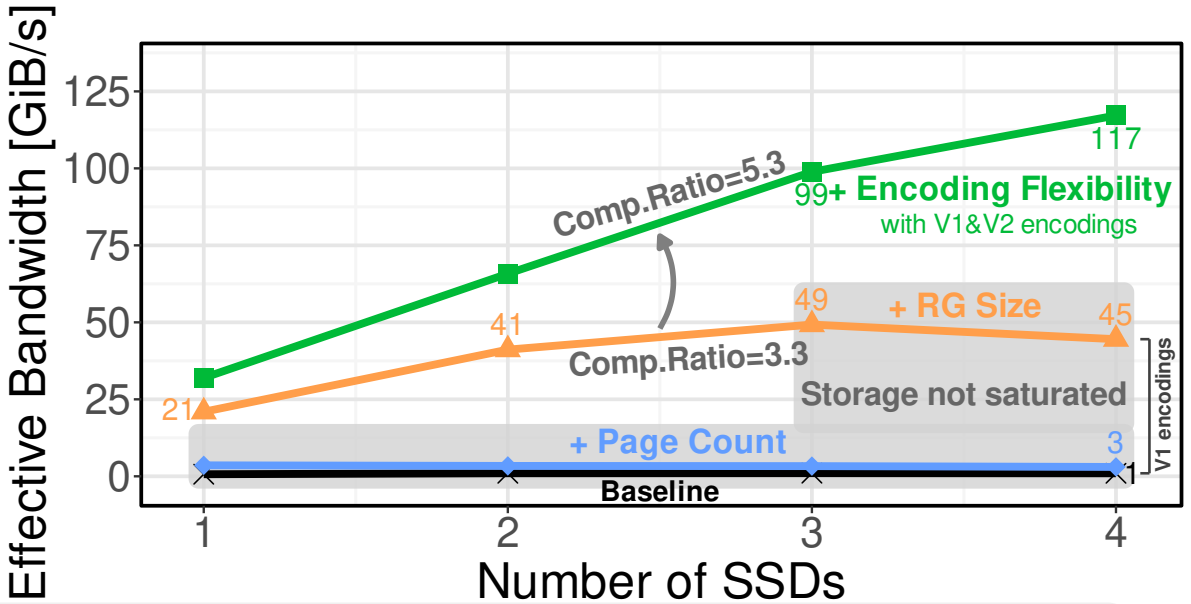
Config. 3: Encoding Flexibility

- Consider both V1 and V2
- Finalize the encoding producing the smallest size
- Improve compression ratio



Config. 3: Encoding Flexibility

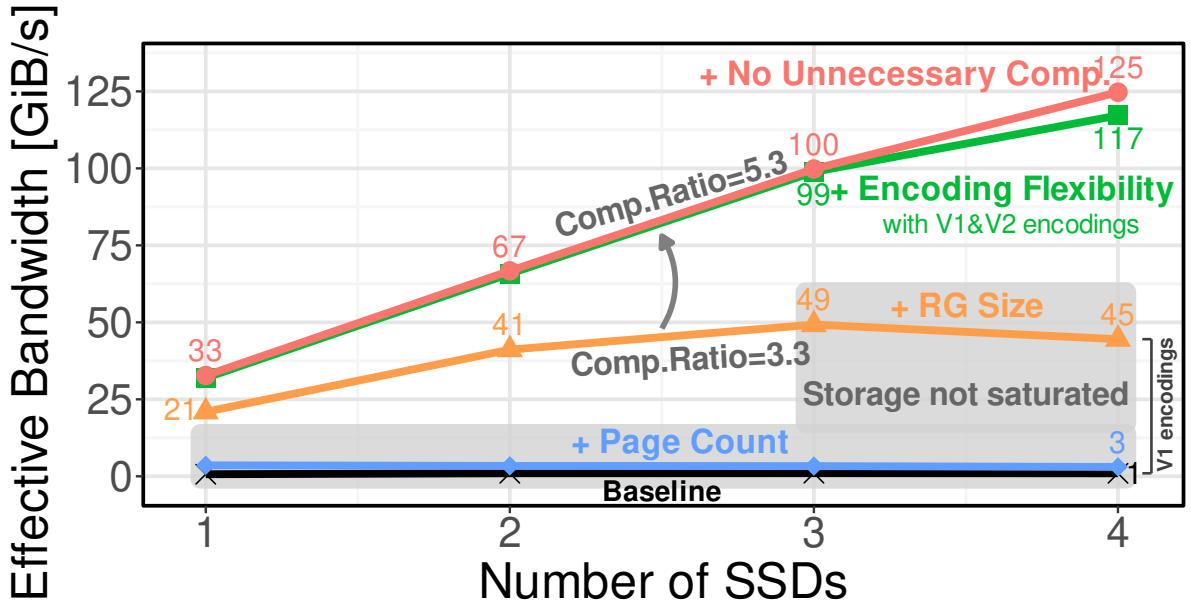
- Consider both V1 and V2
- Finalize the encoding producing the smallest size
- Improve compression ratio



Insight 3: Flexibly choosing encodings for size reduction

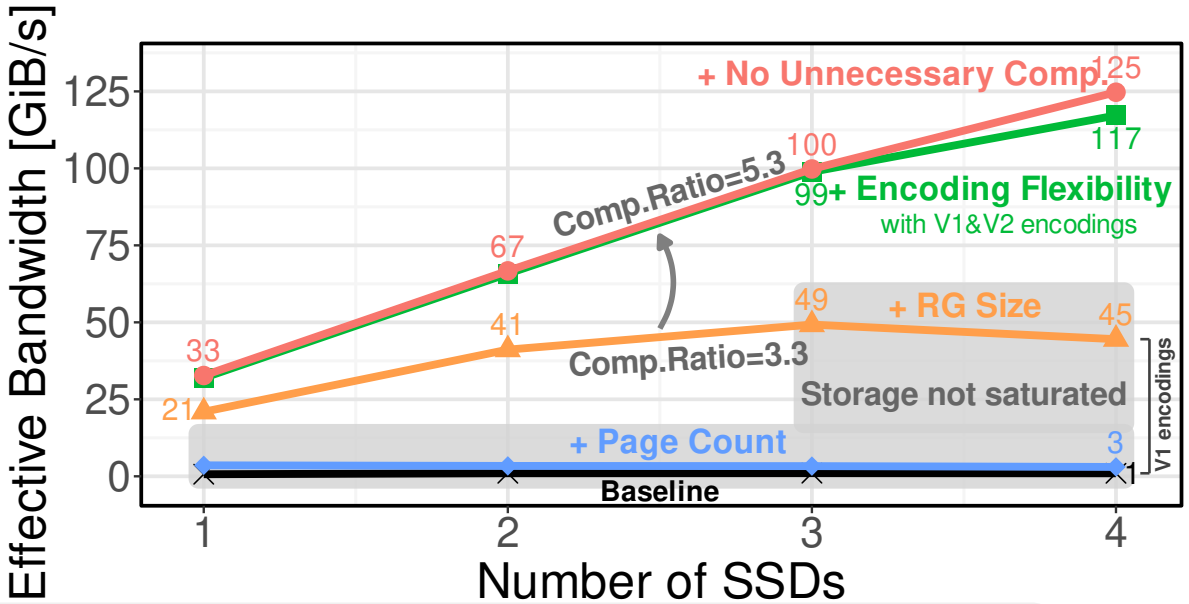
Config. 4: No Unnecessary Compression

- Compress only when size reduction is meaningful
- Otherwise, uncompressed



Config. 4: No Unnecessary Compression

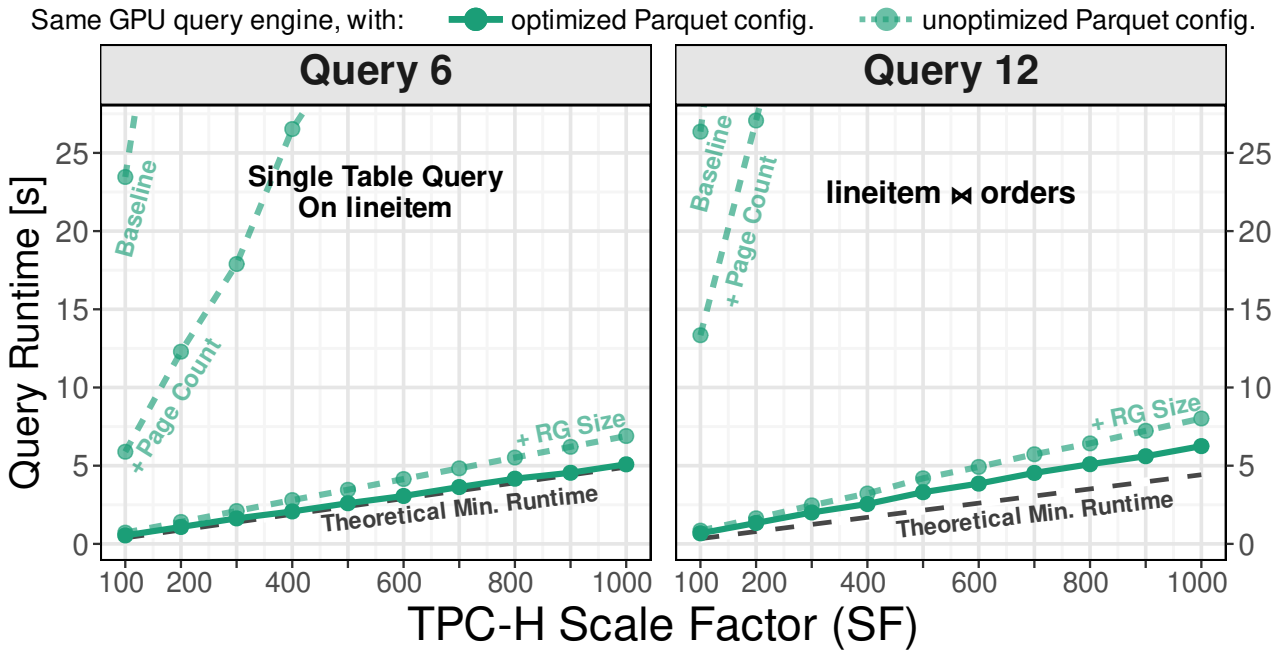
- Compress only when size reduction is meaningful
- Otherwise, uncompressed



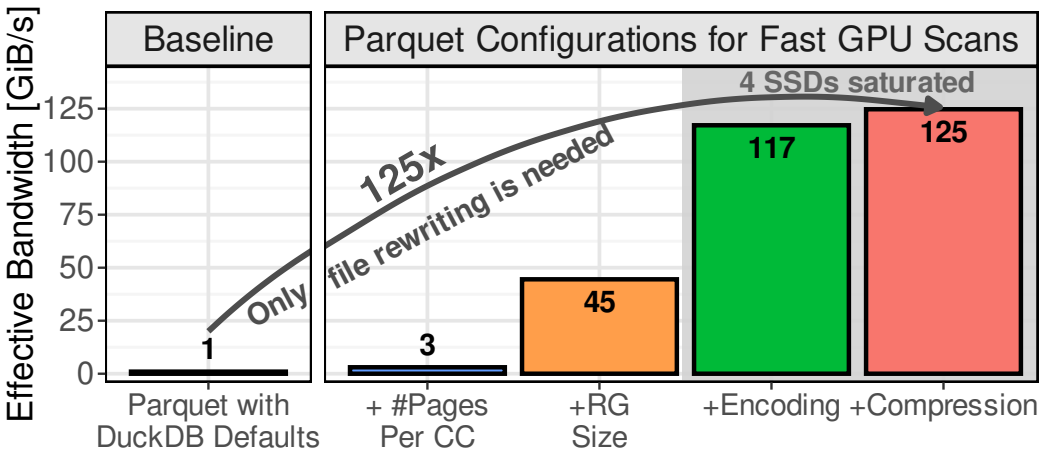
Insight 4: Skip compression if no size reduction

Query Processing With Parquet Improvements

- Parquet improvements carry to queries



Conclusion



- GPU-aware Parquet configuration matters: 4 Insights
- Parquet performs well once optimized for GPUs
- GPU-optimized config. ≠ CPU-optimized config.
- Understand & optimize Parquet **before** replacing it!

Paper

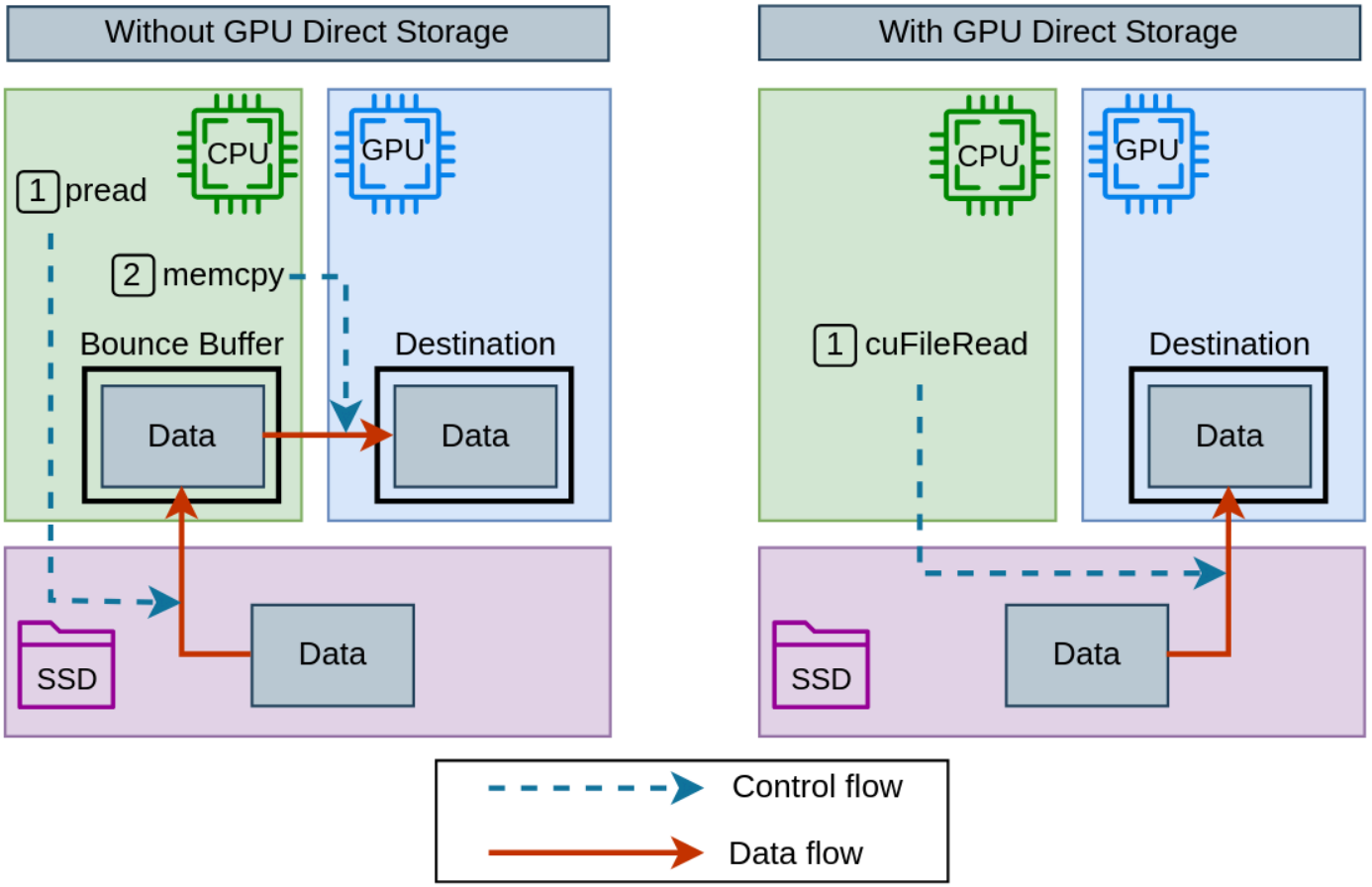
Rewriter Code

TECHNISCHE UNIVERSITÄT DARMSTADT

SYSTEMS

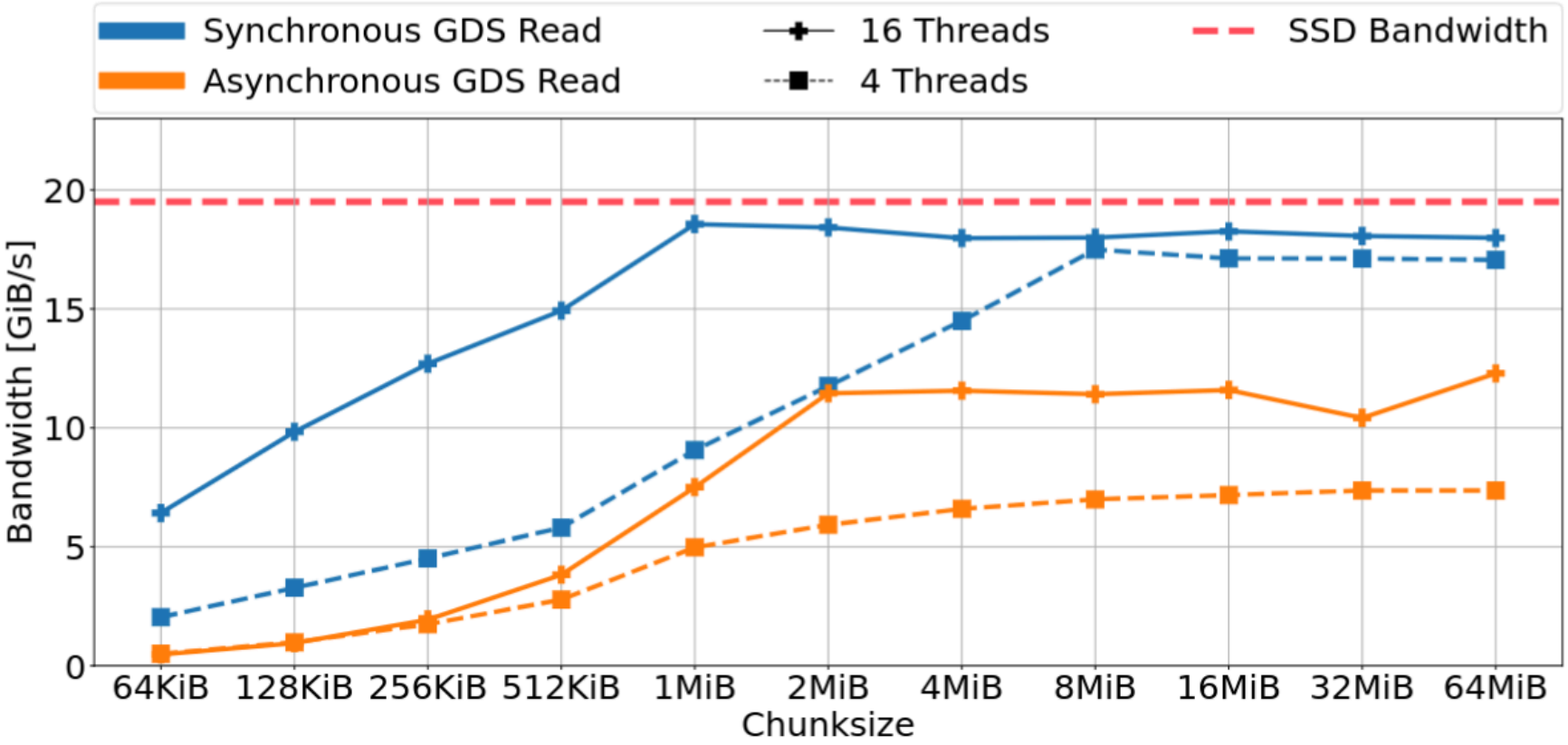
Thanks for your attention

NVIDIA GPU Direct Storage (GDS): I/O Stack



Source: Nils Boeschen 🎓, Tobias Ziegler, and Carsten Binnig. **GOLAP: A GPU-in-Data-Path Architecture for High-Speed OLAP**

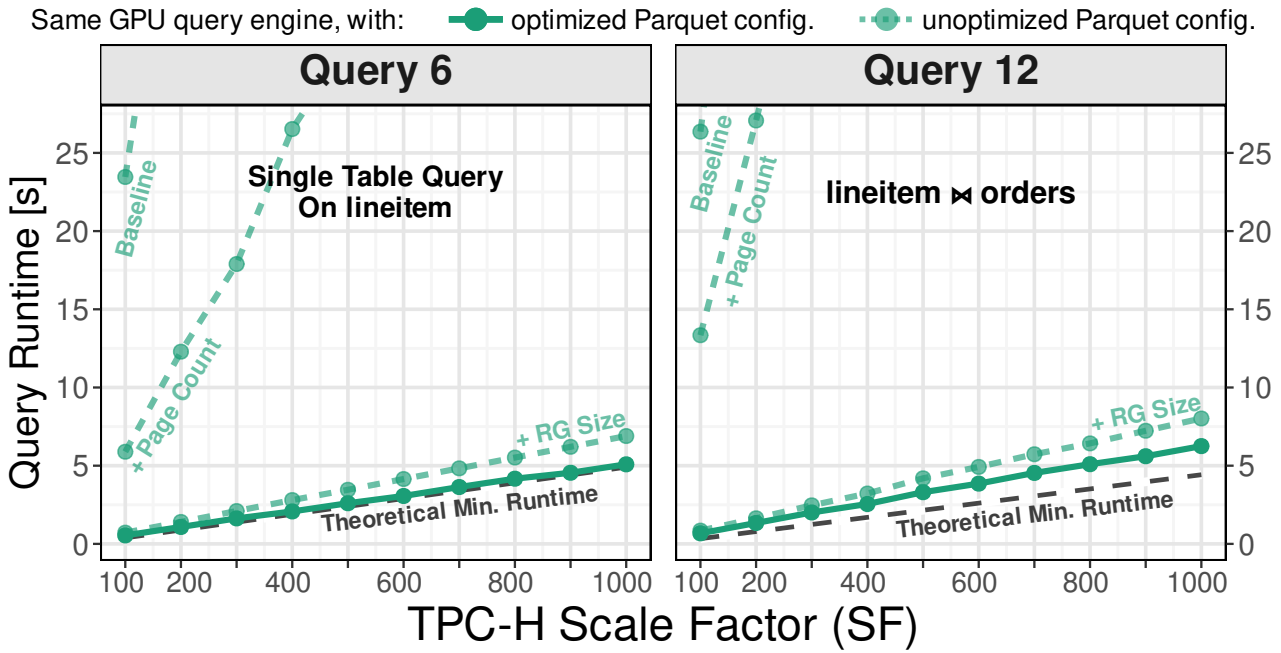
NVIDIA GPU Direct Storage (GDS): Read Size



Source: Nils Boeschen 🎓, Tobias Ziegler, and Carsten Binnig. **GOLAP: A GPU-in-Data-Path Architecture for High-Speed OLAP**

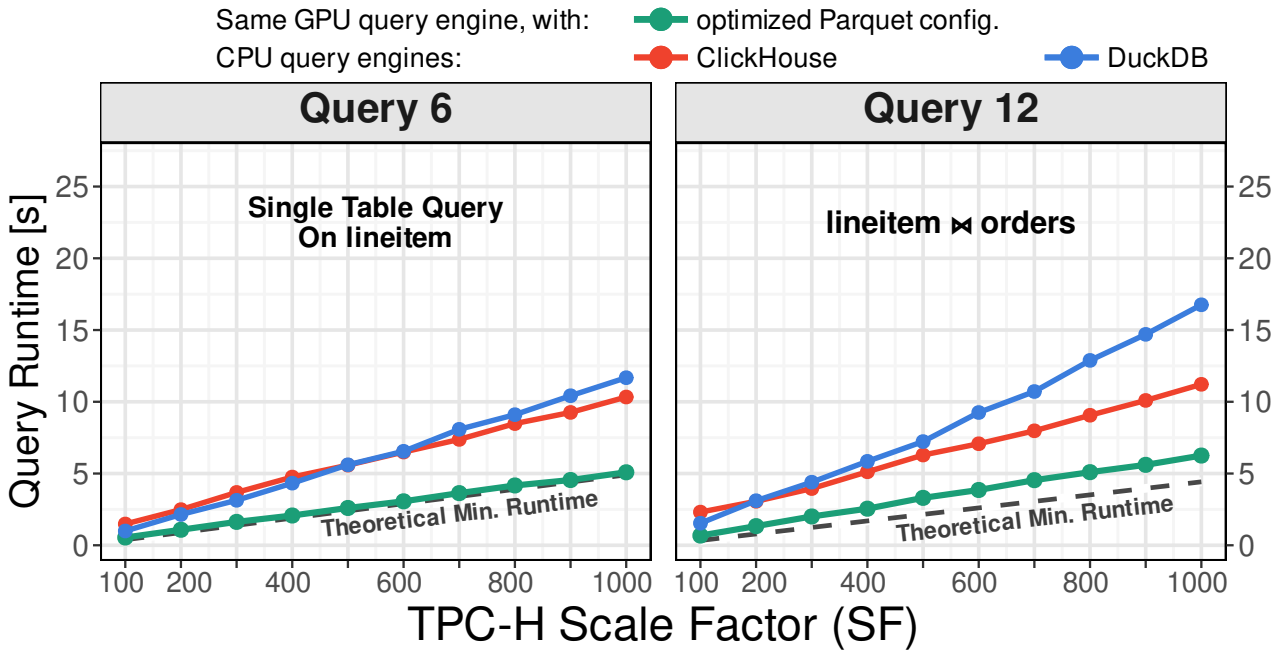
Query Processing With Parquet Improvements

- Parquet improvements carry to queries



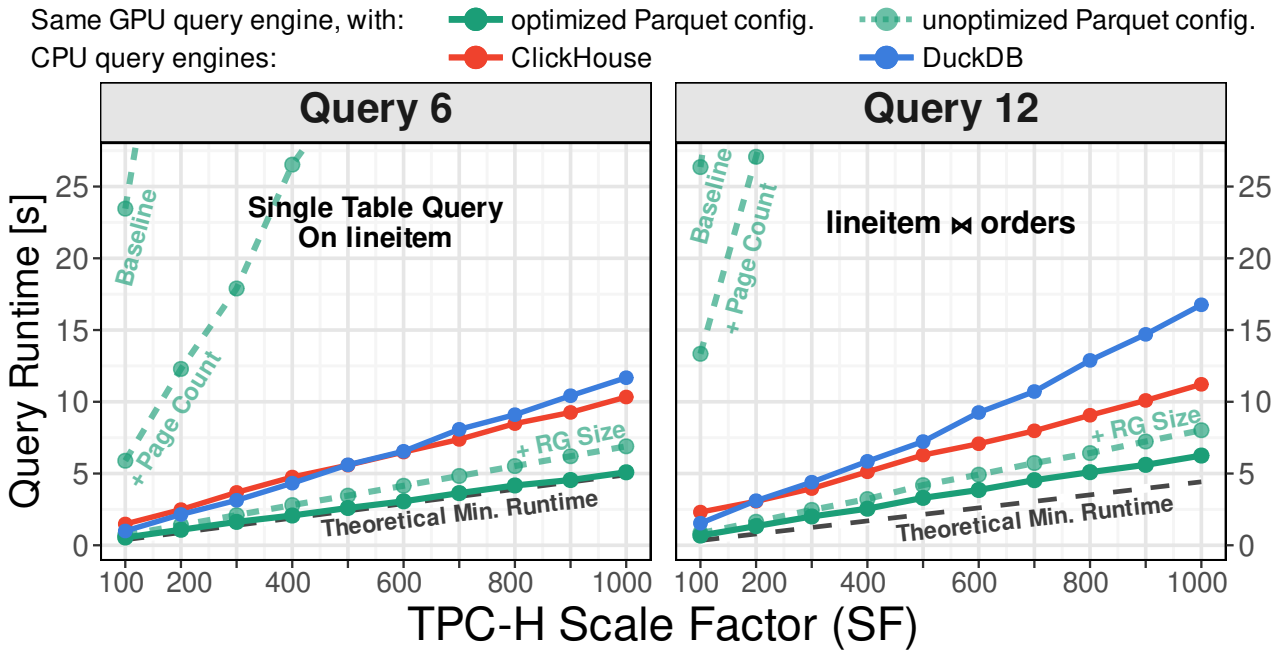
Query Processing With Parquet Improvements

- Parquet improvements carry to queries



Query Processing With Parquet Improvements

- Parquet improvements carry to queries



Additional Work: GPU Parquet Reader & Query Engine

Two Dimensions of GPU Parquet Scan

1. Parquet File configuration
2. GPU Parquet Reader design ⟶

VLDB 2026, PystachIO: https://arxiv.org/abs/2512.02862

PystachIO: Efficient Distributed GPU Query Processing with PyTorch over Fast Networks & Fast Storage

Jigao Luo*
TU Darmstadt

Nils Boeschen*
TU Darmstadt & hessian.AI

Muhammad El-Hindi
TU Munich

Carsten Binnig
TU Darmstadt & hessian.AI & DFKI Darmstadt

Abstract

The AI hardware boom has led modern data centers to adopt HPC-style architectures centered on distributed, GPU-centric computation. Large GPU clusters interconnected by fast RDMA networks and backed by high-bandwidth NVMe storage enable scalable computation and rapid access to storage-resident data. Tensor computation runtimes (TCRs), such as PyTorch, originally designed for AI workloads, have recently been shown to accelerate analytical workloads. However, prior work has primarily considered settings where the data fits in aggregated GPU memory. In this paper, we systematically study how TCRs can support scalable, distributed query processing for large-scale, storage-resident OLAP workloads. Although TCRs provide abstractions for network and storage I/O, naive use often underutilizes GPU and I/O bandwidth due to insufficient overlap between computation and data movement. As a core

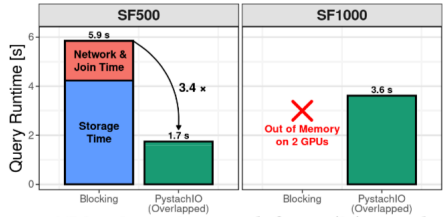


Figure 1: TPC-H Q3 runtime at scale factors (SF) 500 and 1000 on two GPUs over SSD-resident, non-co-partitioned tables. A blocking PyTorch-style baseline executes sequentially, leading to long runtimes and out-of-memory errors. In contrast, PystachIO overlaps storage I/O, networking, and computation to reduce query time by over 3×.

Additional Work: GPU Parquet Reader & Query Engine

Two Dimensions of GPU Parquet Scan

1. Parquet File configuration
2. GPU Parquet Reader design ⟶

GPU Query Engine:

Overlapping storage, network, and compute

VLDB 2026, PystachIO: https://arxiv.org/abs/2512.02862

PystachIO: Efficient Distributed GPU Query Processing with PyTorch over Fast Networks & Fast Storage

Jigao Luo*
TU Darmstadt

Nils Boeschen*
TU Darmstadt & hessian.AI

Muhammad El-Hindi
TU Munich

Carsten Binnig
TU Darmstadt & hessian.AI & DFKI Darmstadt

Abstract

The AI hardware boom has led modern data centers to adopt HPC-style architectures centered on distributed, GPU-centric computation. Large GPU clusters interconnected by fast RDMA networks and backed by high-bandwidth NVMe storage enable scalable computation and rapid access to storage-resident data. Tensor computation runtimes (TCRs), such as PyTorch, originally designed for AI workloads, have recently been shown to accelerate analytical workloads. However, prior work has primarily considered settings where the data fits in aggregated GPU memory. In this paper, we systematically study how TCRs can support scalable, distributed query processing for large-scale, storage-resident OLAP workloads. Although TCRs provide abstractions for network and storage I/O, naive use often underutilizes GPU and I/O bandwidth due to insufficient overlap between computation and data movement. As a core

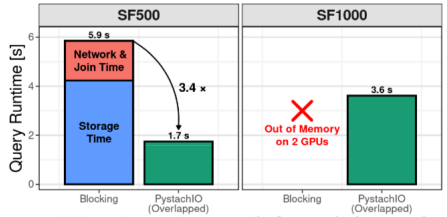
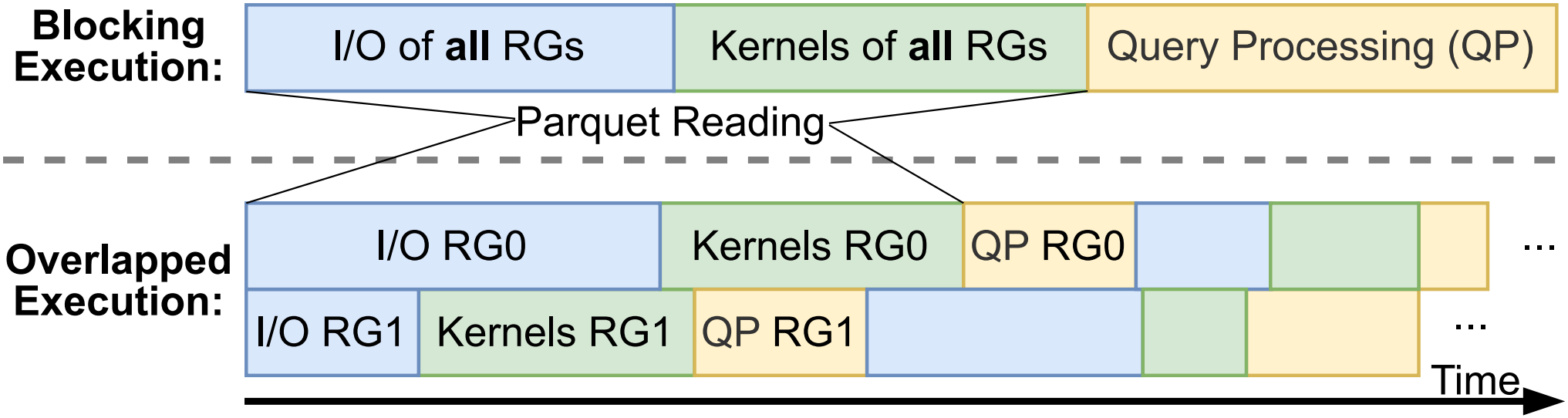


Figure 1: TPC-H Q3 runtime at scale factors (SF) 500 and 1000 on two GPUs over SSD-resident, non-co-partitioned tables. A blocking PyTorch-style baseline executes sequentially, leading to long runtimes and out-of-memory errors. In contrast, PystachIO overlaps storage I/O, networking, and computation to reduce query time by over 3×.

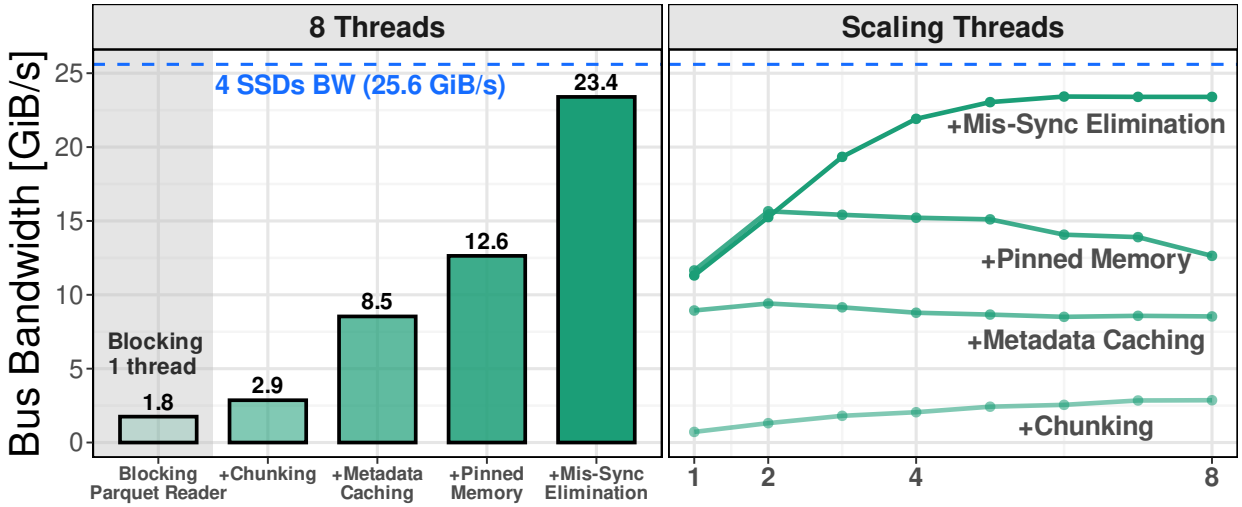
Overlapping: Reader & Query Engine



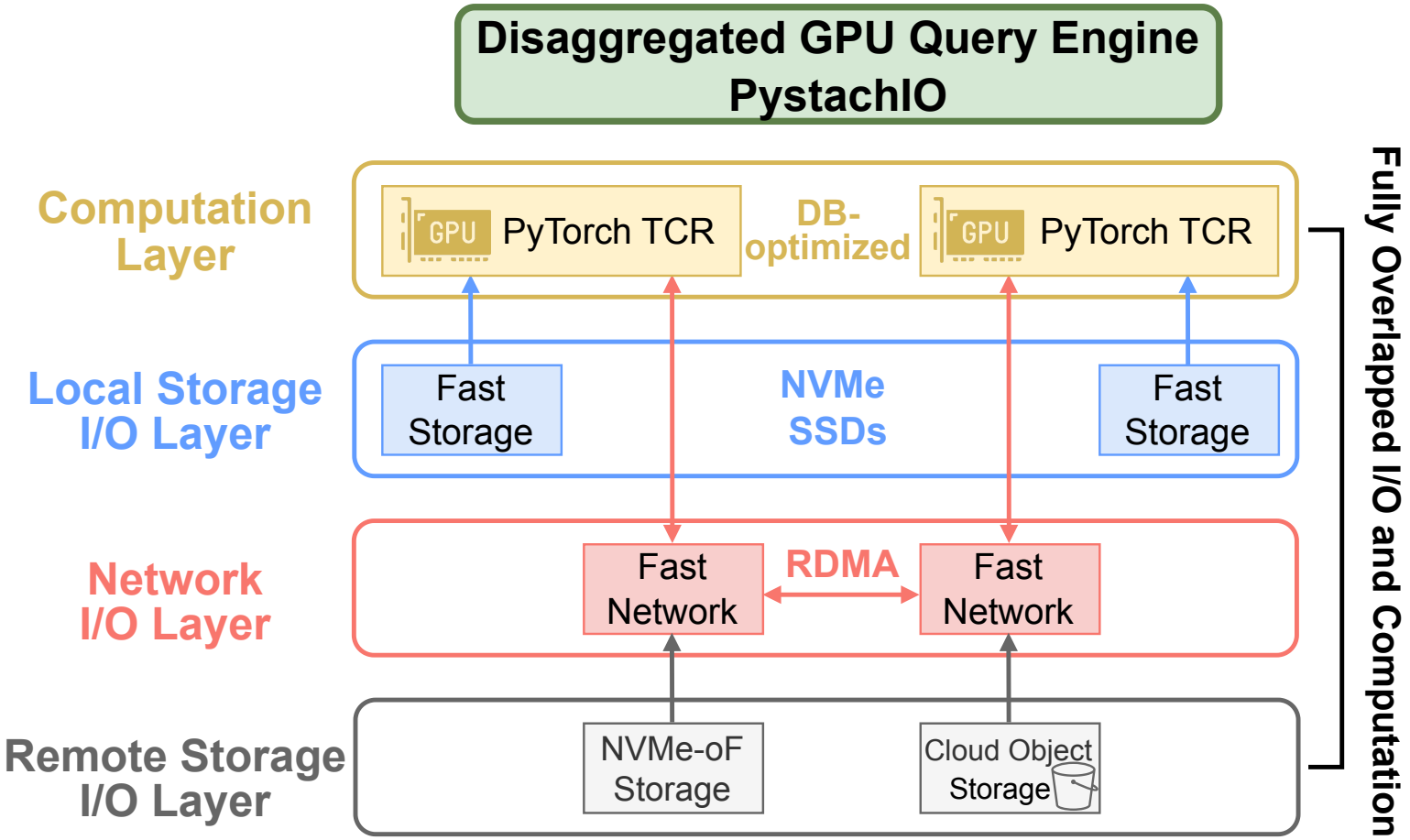
Overlapping Reader: Optimizations

- Built on NVIDIA RAPIDS cuDF
- Merged upstream

cuDF: [Story Issue] Towards a faster Parquet reader with pipelining and multistream optimization: https://github.com/rapidsai/cudf/issues/18892



PystachIO: Disaggreaged Architecture



“How do you deal with OOM?”

- Storage: chunked RG reads
- Network: scaling across GPUs via RDMA
- More info in ⟶

VLDB 2026, PystachIO: https://arxiv.org/abs/2512.02862

PystachIO: Efficient Distributed GPU Query Processing with PyTorch over Fast Networks & Fast Storage

Jigao Luo*
TU Darmstadt

Nils Boeschen*
TU Darmstadt & hessian.AI

Muhammad El-Hindi
TU Munich

Carsten Binnig
TU Darmstadt & hessian.AI & DFKI Darmstadt

Abstract

The AI hardware boom has led modern data centers to adopt HPC-style architectures centered on distributed, GPU-centric computation. Large GPU clusters interconnected by fast RDMA networks and backed by high-bandwidth NVMe storage enable scalable computation and rapid access to storage-resident data. Tensor computation runtimes (TCRs), such as PyTorch, originally designed for AI workloads, have recently been shown to accelerate analytical workloads. However, prior work has primarily considered settings where the data fits in aggregated GPU memory. In this paper, we systematically study how TCRs can support scalable, distributed query processing for large-scale, storage-resident OLAP workloads. Although TCRs provide abstractions for network and storage I/O, naive use often underutilizes GPU and I/O bandwidth due to insufficient overlap between computation and data movement. As a core

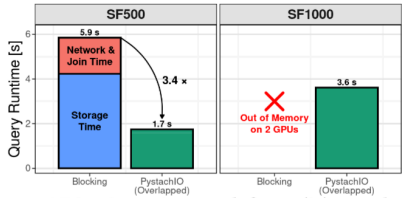
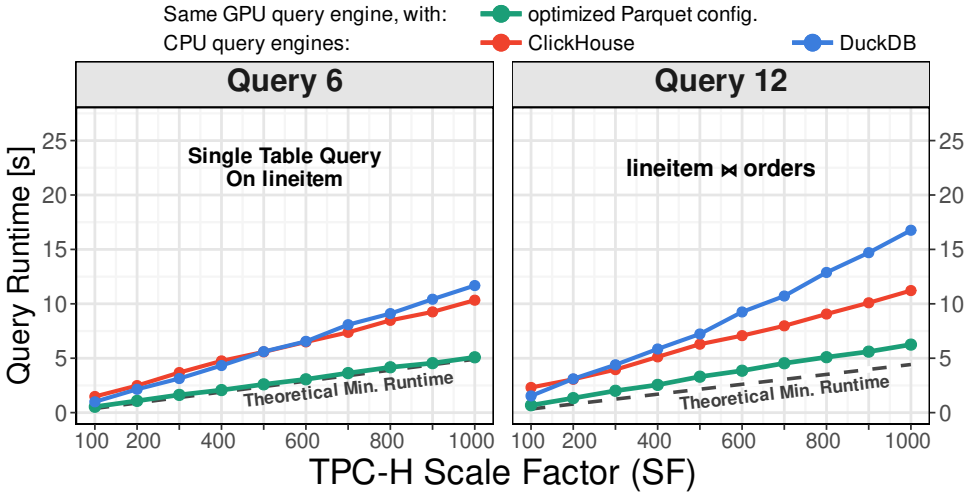
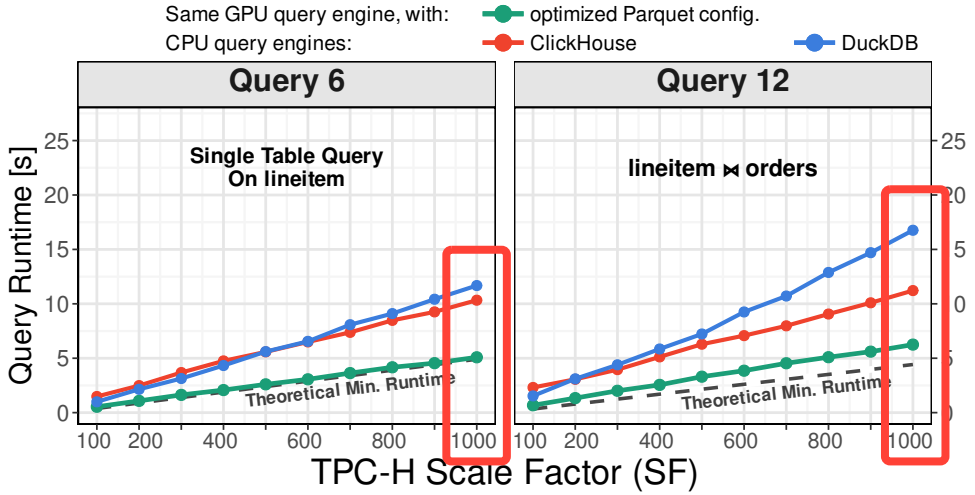


Figure 1: TPC-H Q3 runtime at scale factors (SF) 500 and 1000 on two GPUs over SSD-resident, non-co-partitioned tables. A blocking PyTorch-style baseline executes sequentially, leading to long runtimes and out-of-memory errors. In contrast, PystachIO overlaps storage I/O, networking, and computation to reduce query time by over 3×.

“Are GPUs Expensive?”



“Are GPUs Expensive?”



	Runtime Slowdown vs. PystachIO		Relative TCO vs. PystachIO	
Query	ClickHouse	DuckDB	ClickHouse	DuckDB
Q6	2.0x	2.3x	5.0x	5.6x
Q12	1.8x	2.7x	4.4x	6.6x

Table 1: SF1000 single-node runtime slowdown and relative TCO.