Do GPUs Really Need New Tabular File Formats?

DaMoN 2026 Short Paper

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 With File Formats

- AI Training
- AI Inference
- Query Processing

Data Processing With File Formats

- AI Training
- AI Inference
- Query Processing

File Formats, e.g.,

Data Processing With File Formats

- AI Training
- AI Inference
- Query Processing

File Formats, e.g.,

Why Parquet?

- De facto standard

Data Processing With File Formats

- AI Training
- AI Inference
- Query Processing

File Formats, e.g.,

Why Parquet?

- De facto standard
- Widely supported

Data Processing With File Formats

- AI Training
- AI Inference
- Query Processing

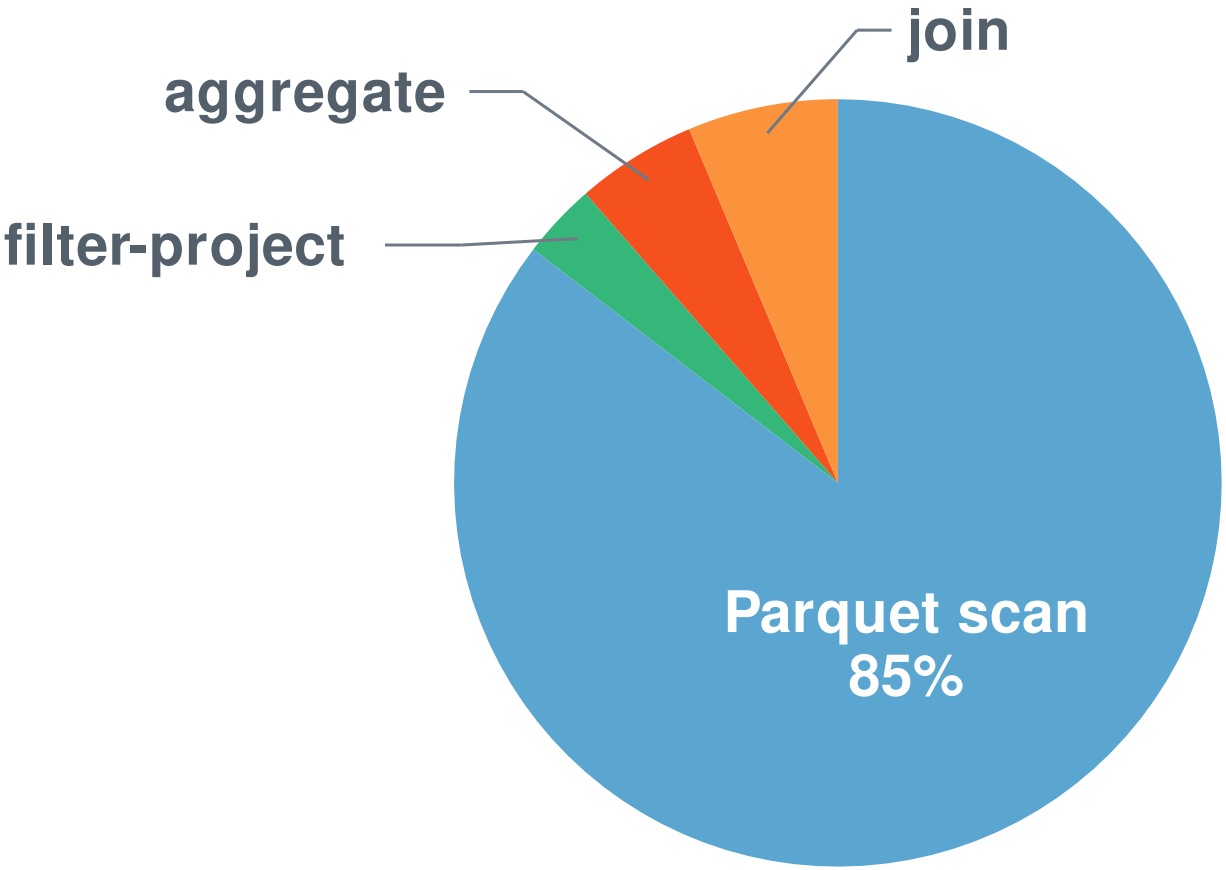
File Formats, e.g.,

Why Parquet?

- De facto standard
- Widely supported
- GPU impl. available

Problem: GPU Bottleneck With Parquet

- **Fast**: Join, Agg, Filter

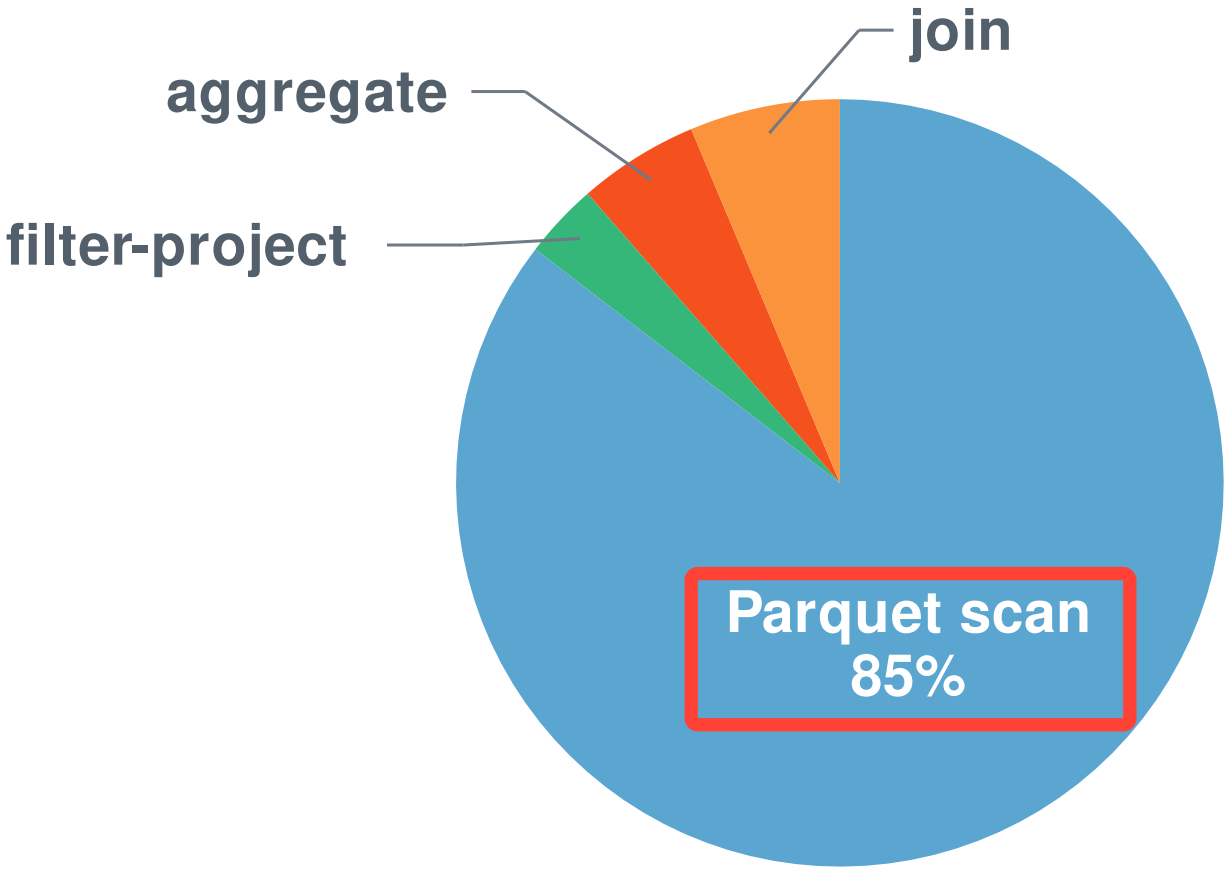


TPC-H SF100 runtime breakdown.

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

Problem: GPU Bottleneck With Parquet

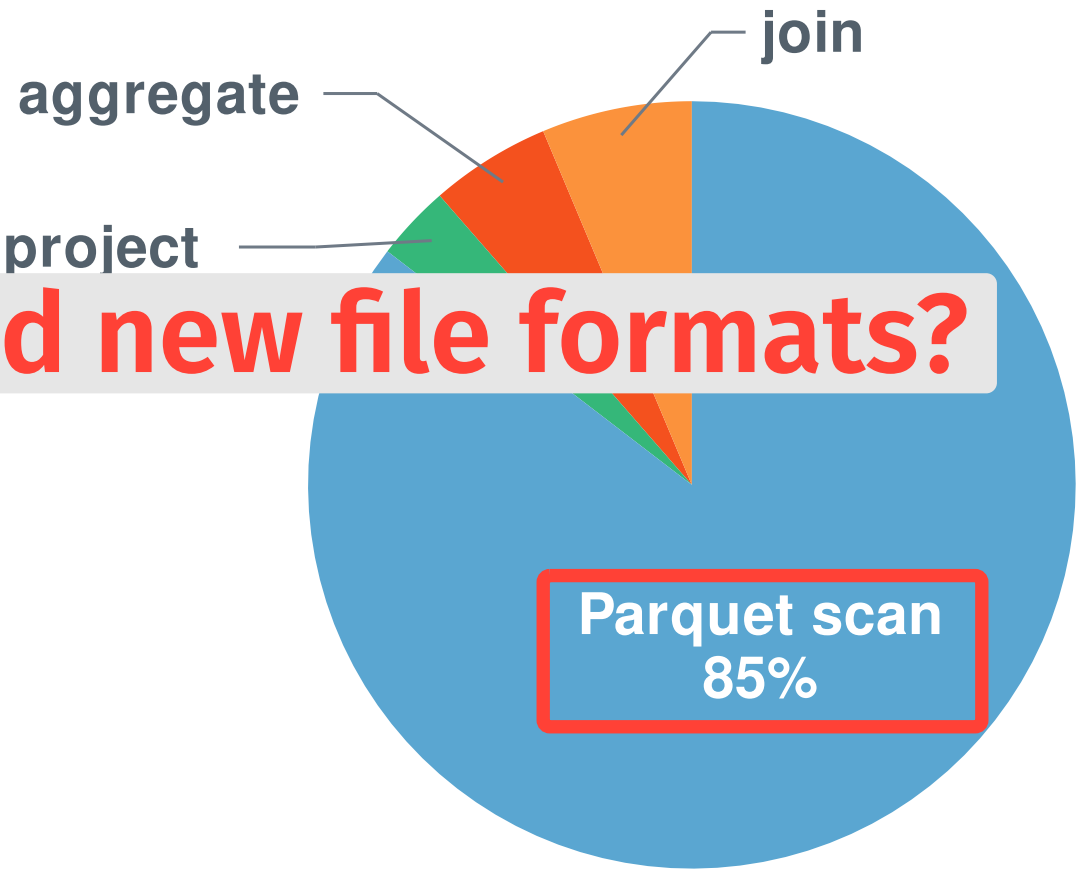
- **Fast**: Join, Agg, Filter
- **Slow**: Parquet Scan



TPC-H SF100 runtime breakdown.

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

Problem: GPU Bottleneck With Parquet



Do GPUs need new file formats?

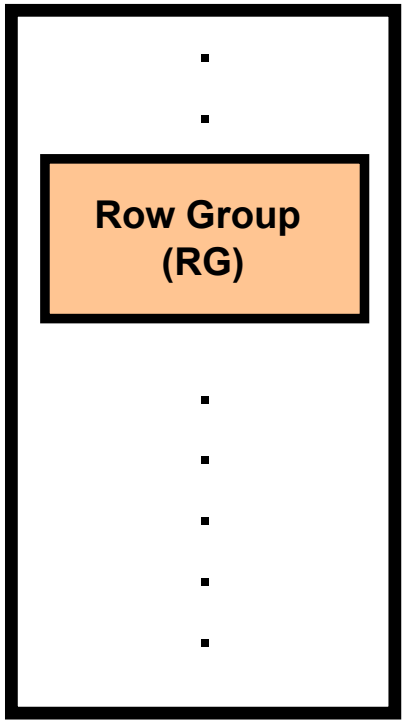
- **Fast**: Join, Agg, Filter
- **Slow**: Parquet Scan

TPC-H SF100 runtime breakdown.

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

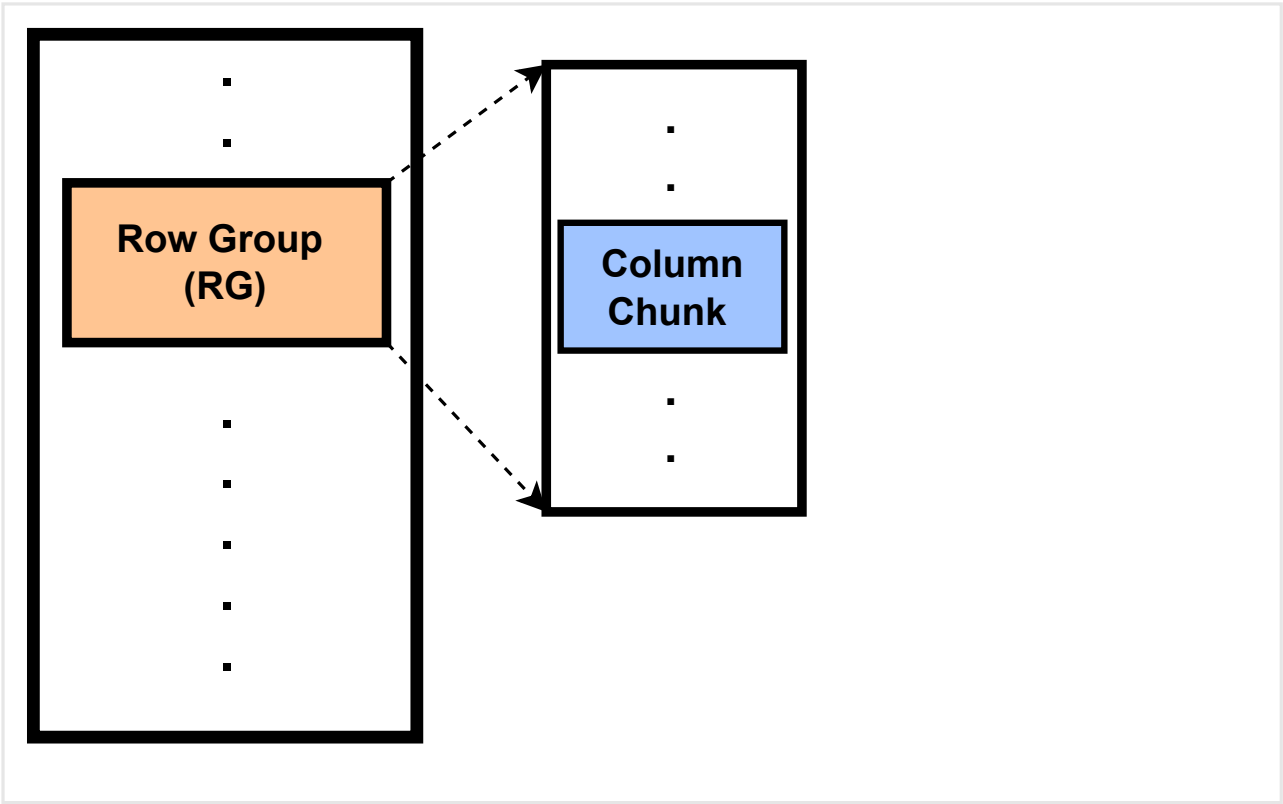
Background: Apache Parquet

- Row group



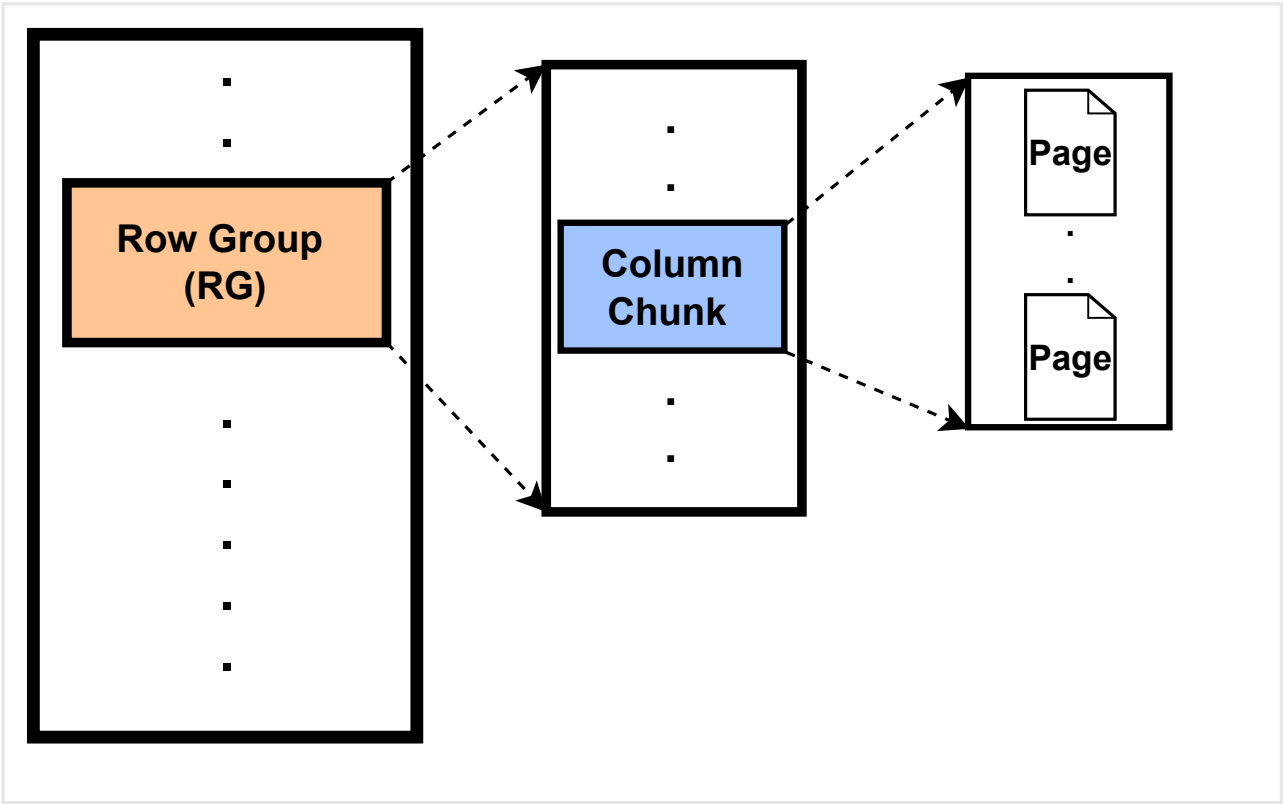
Background: Apache Parquet

- Row group
- Column chunk



Background: Apache Parquet

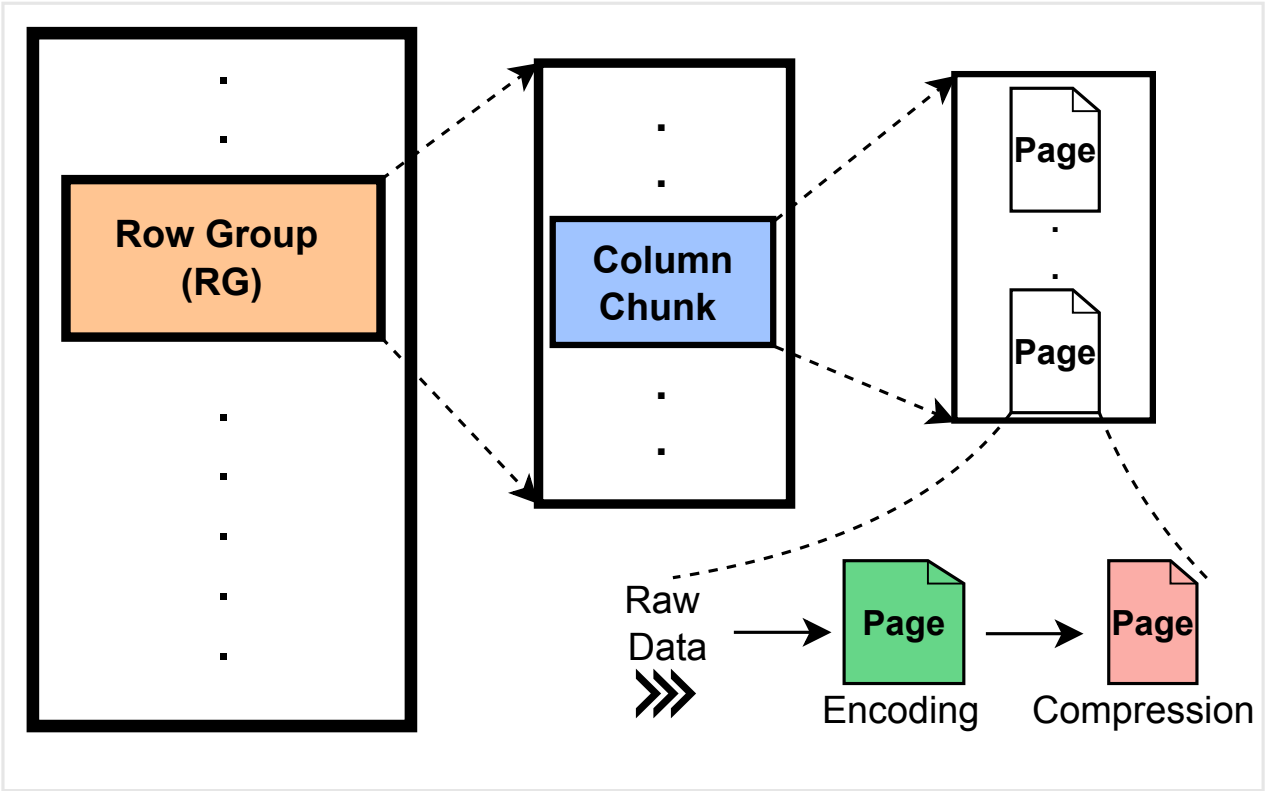
- Row group
- Column chunk
- Pages:



Parquet

Background: Apache Parquet

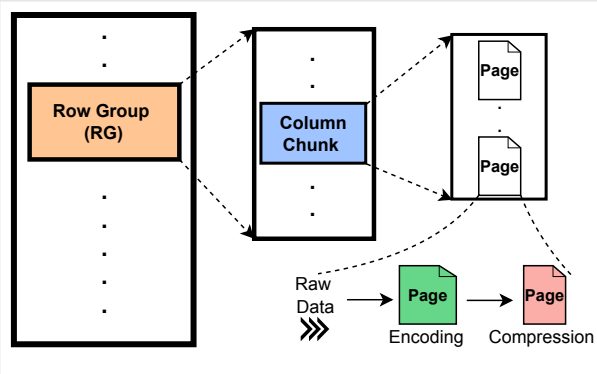
- Row group
- Column chunk
- Pages:
  - Encoded
  - Compressed



Parquet

Issue: CPU-Oriented Parquet Configuration Defaults

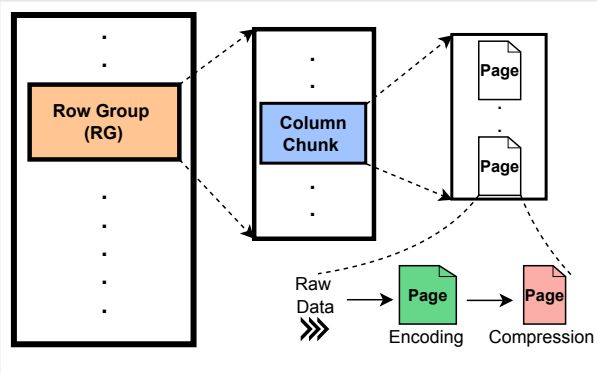
Parquet Config. Defaults for **CPUs**:



Issue: CPU-Oriented Parquet Configuration Defaults

Parquet Config. Defaults for **CPUs**:

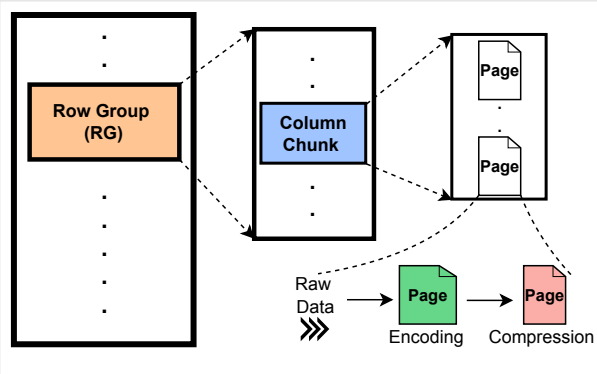
- Row group: 122880 rows



Issue: CPU-Oriented Parquet Configuration Defaults

Parquet Config. Defaults for **CPUs**:

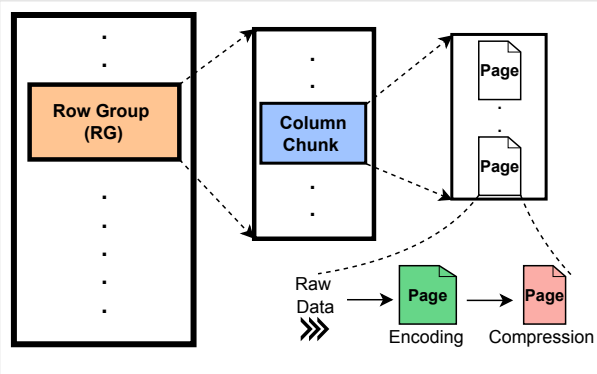
- Row group: 122880 rows
- Column chunk: 1 page only



Issue: CPU-Oriented Parquet Configuration Defaults

Parquet Config. Defaults for **CPUs**:

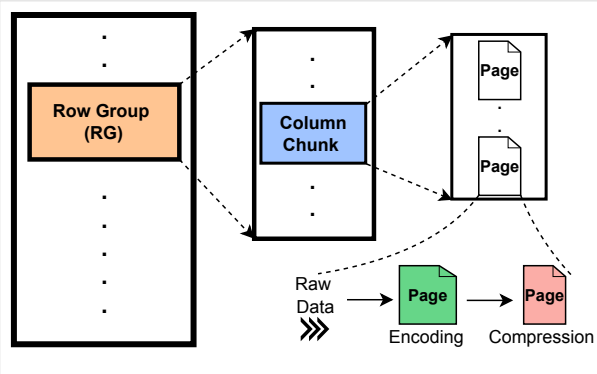
- Row group: 122880 rows
- Column chunk: 1 page only
- Encoding: V1 only



Issue: CPU-Oriented Parquet Configuration Defaults

Parquet Config. Defaults for **CPUs**:

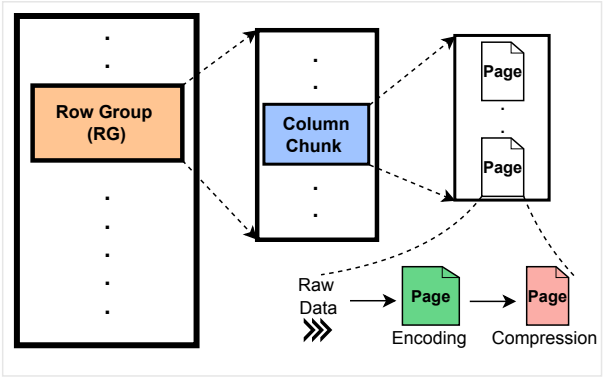
- Row group: 122880 rows
- Column chunk: 1 page only
- Encoding: V1 only
- Compression: used even without size reduction

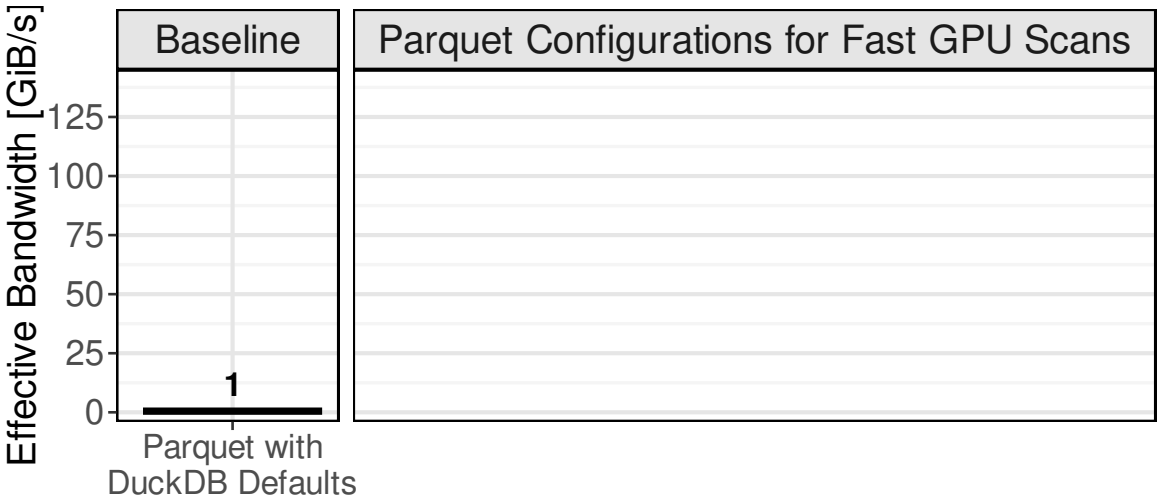


Issue: CPU-Oriented Parquet Configuration Defaults

Parquet Config. Defaults for **CPUs**:

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

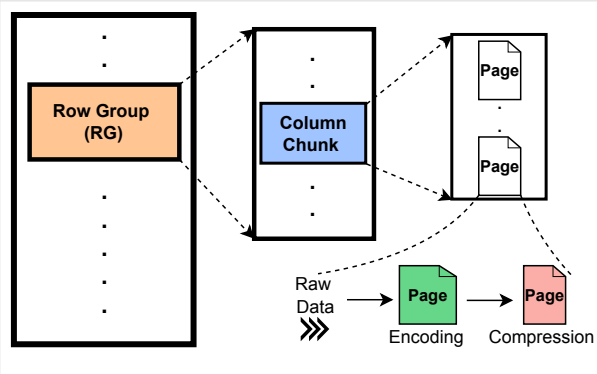


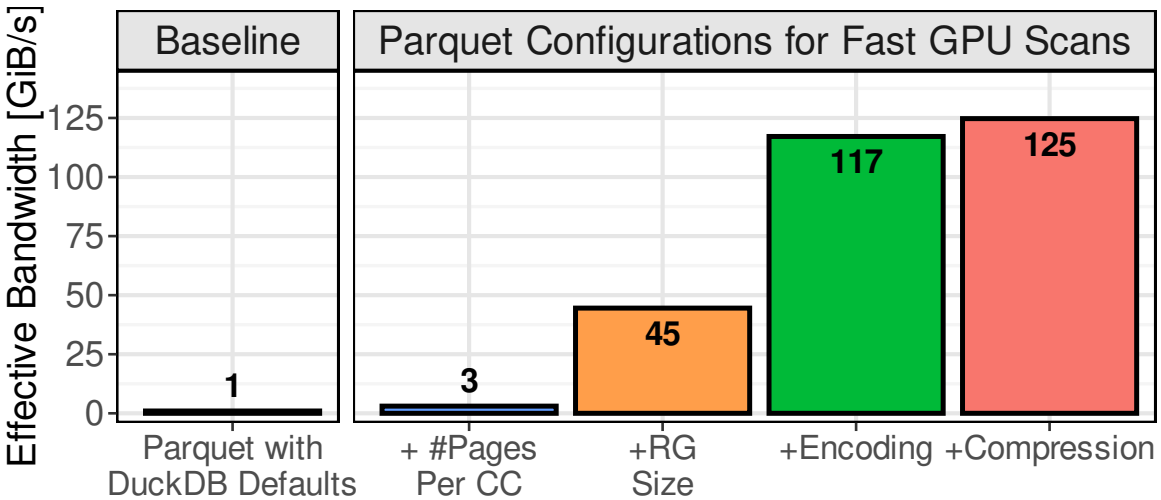


Issue: CPU-Oriented Parquet Configuration Defaults

Parquet Config. Defaults for **CPUs**:

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

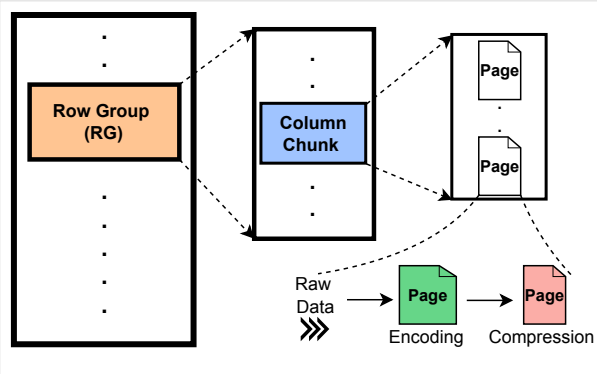


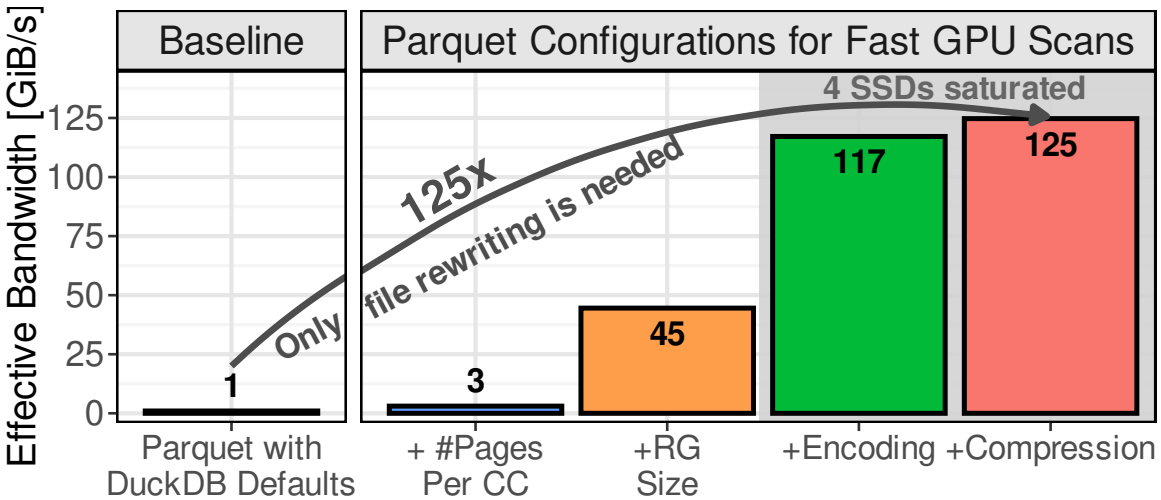


Issue: CPU-Oriented Parquet Configuration Defaults

Parquet Config. Defaults for **CPUs**:

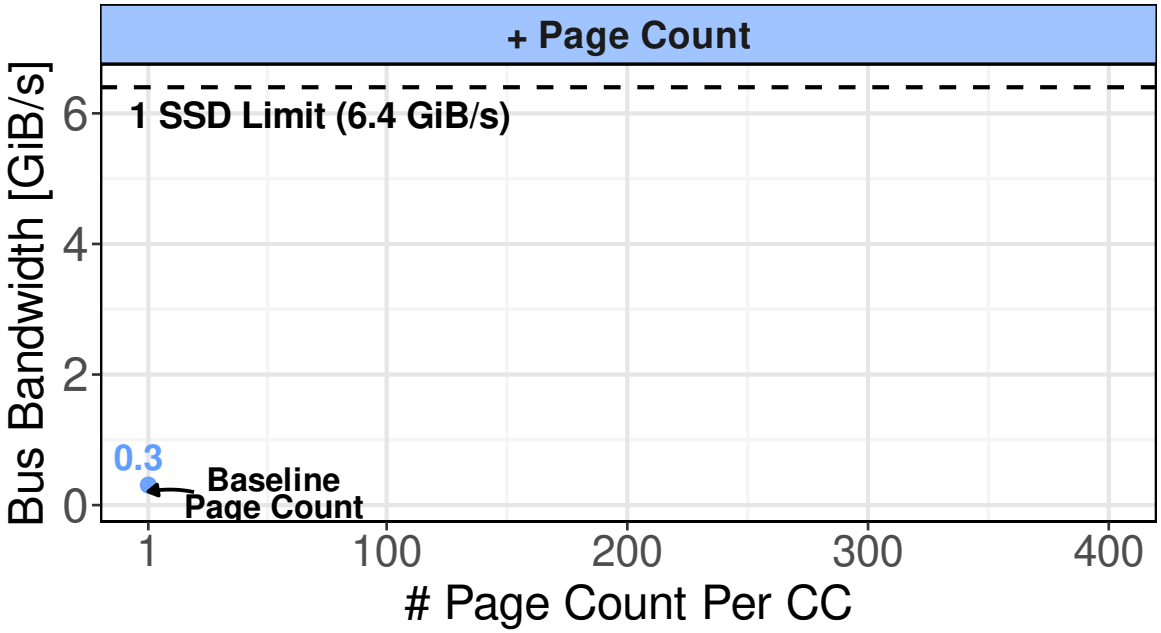
- Row group: 122880 rows
- Column chunk: 1 page only
- Encoding: V1 only
- Compression: used even without size reduction





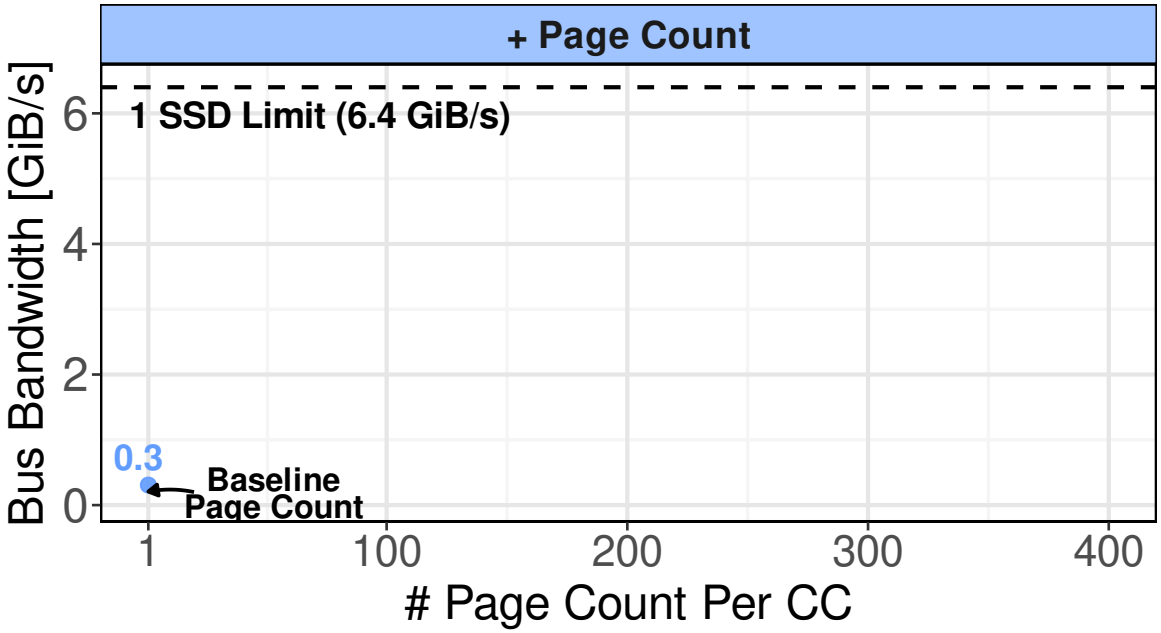
Config.: Page Count Per Column Chunk

- Baseline of 1 due to CPU parallelism



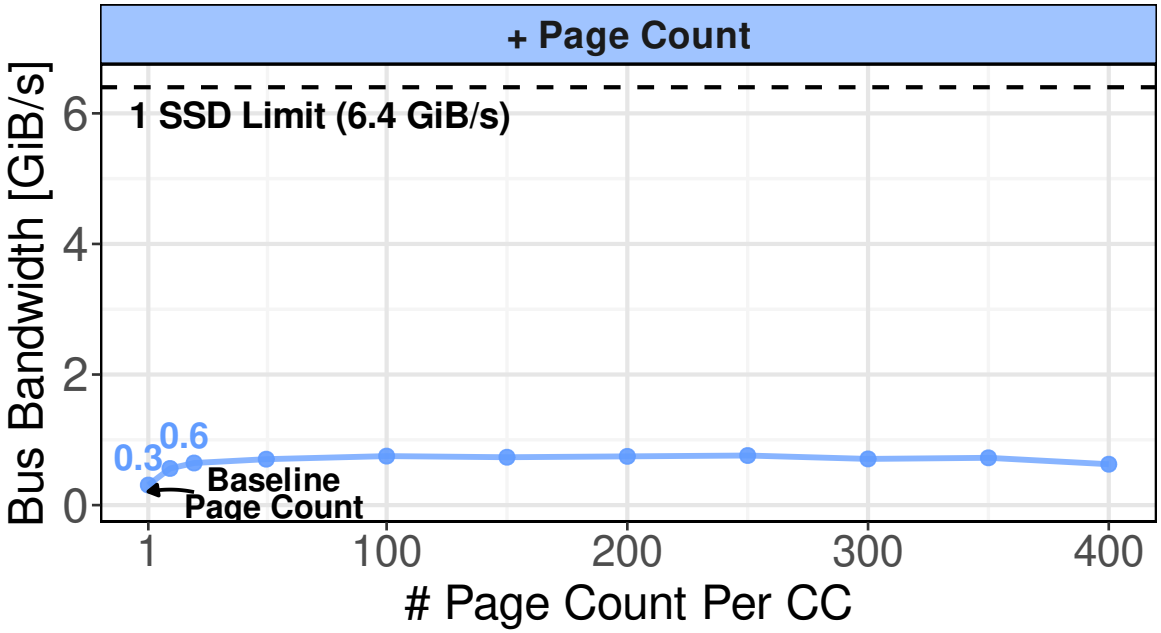
Config.: Page Count Per Column Chunk

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



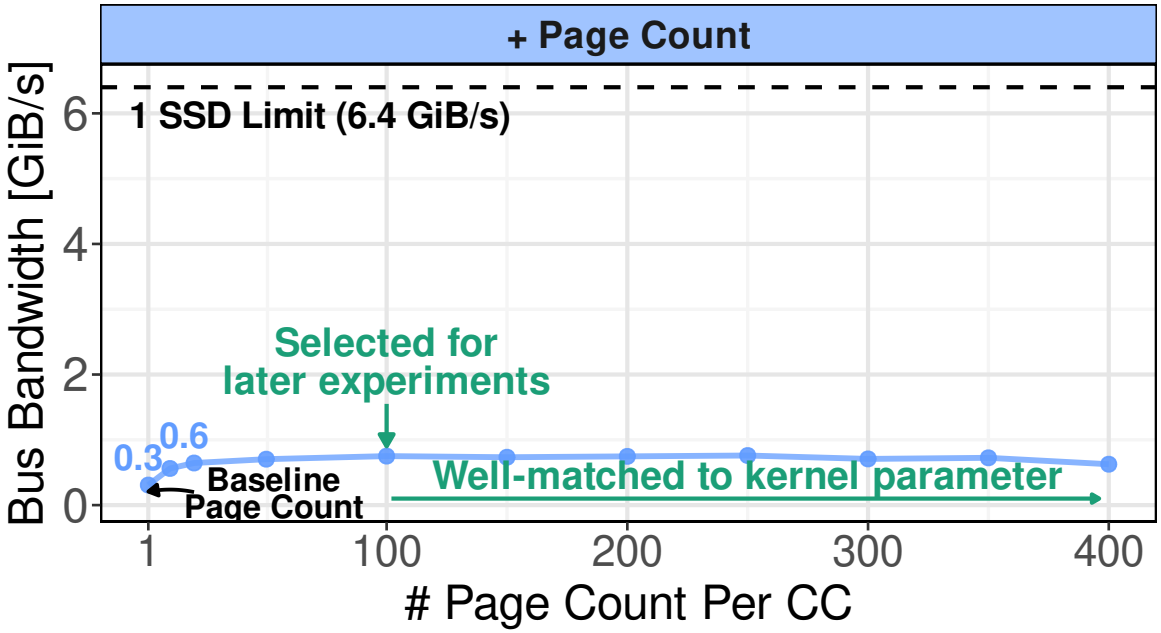
Config.: Page Count Per Column Chunk

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



Config.: Page Count Per Column Chunk

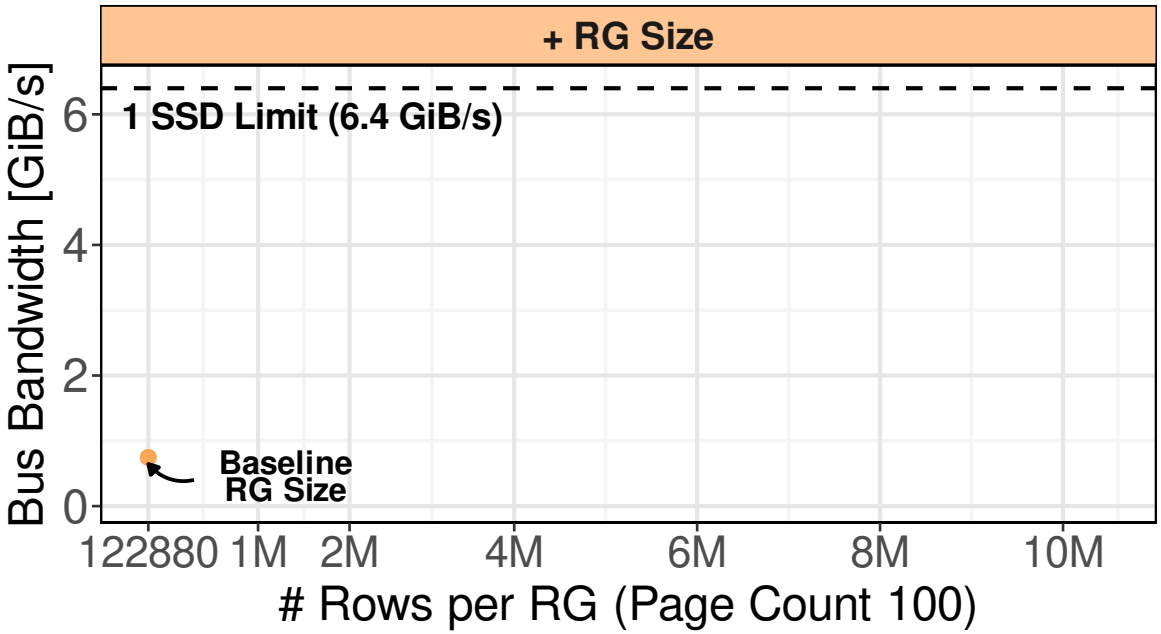
- Baseline of 1 due to CPU parallelism
- #pages mapped to GPU kernel parameter
- Too few pages underutilize GPU



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

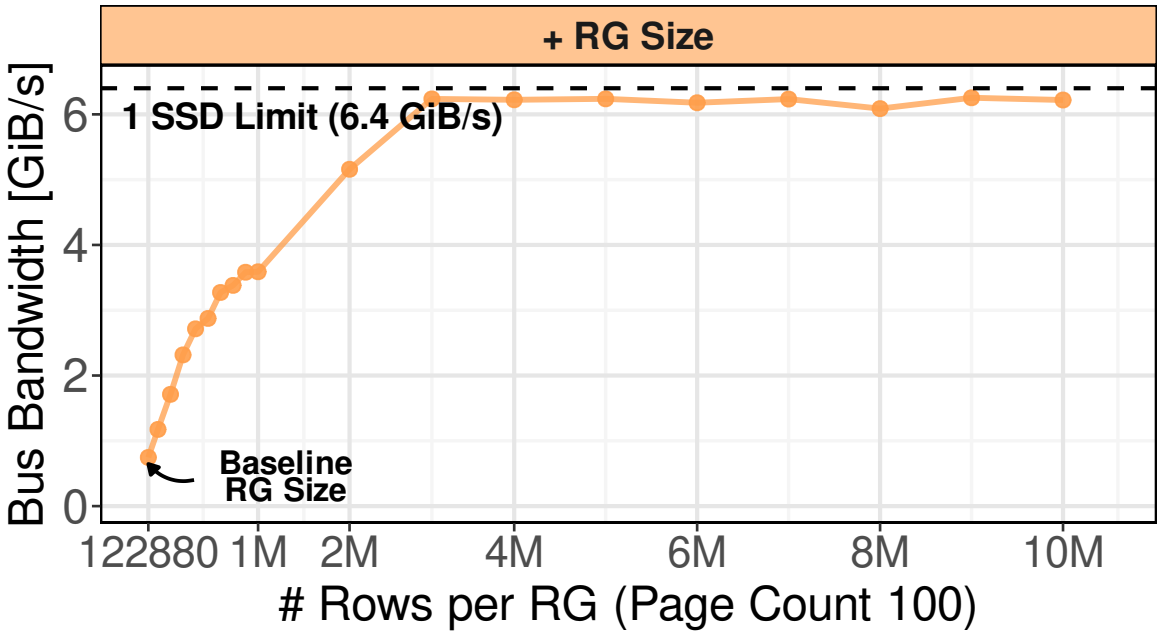
Config.: Row Group Size

- GPU I/O stack differs from CPU's
- RG size controls I/O granularity



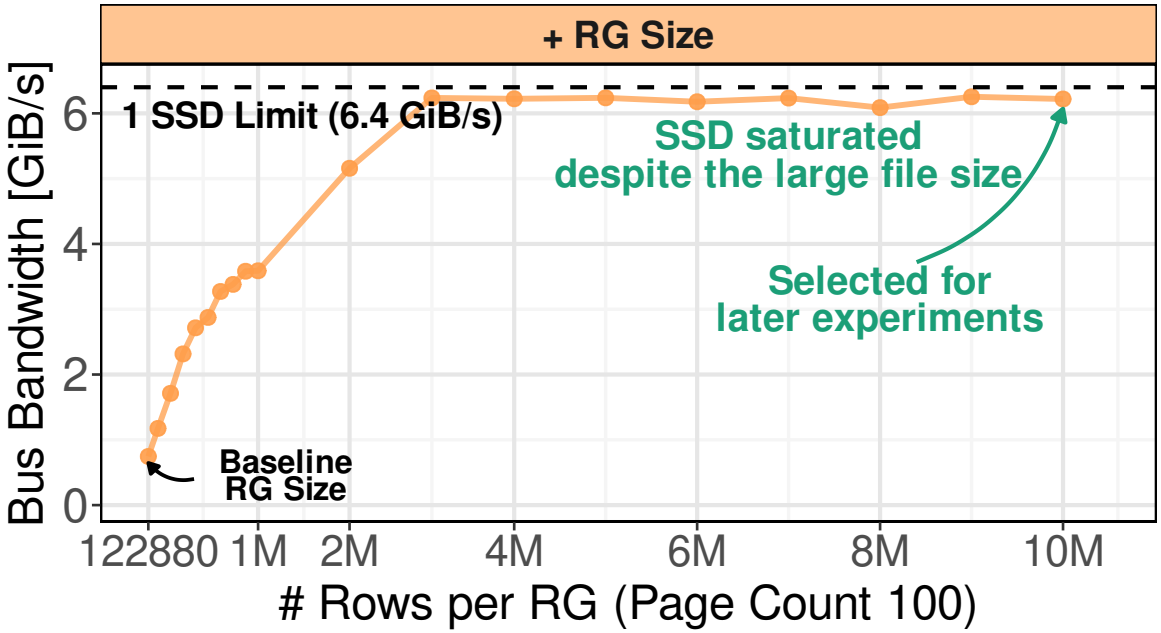
Config.: Row Group Size

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



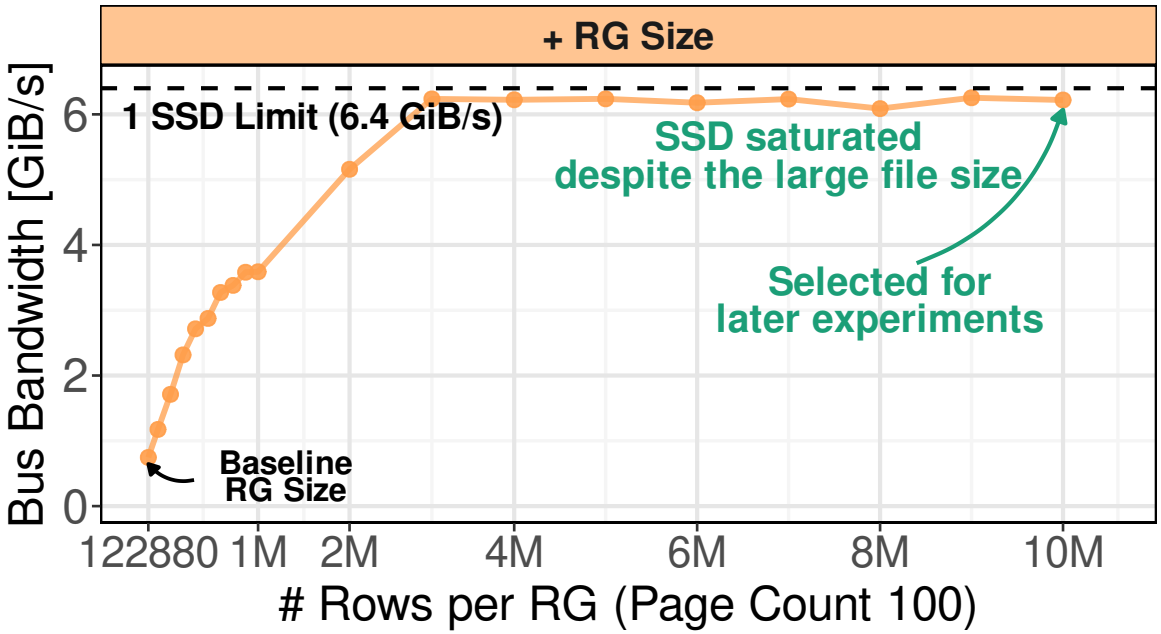
Config.: Row Group Size

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



Config.: Row Group Size

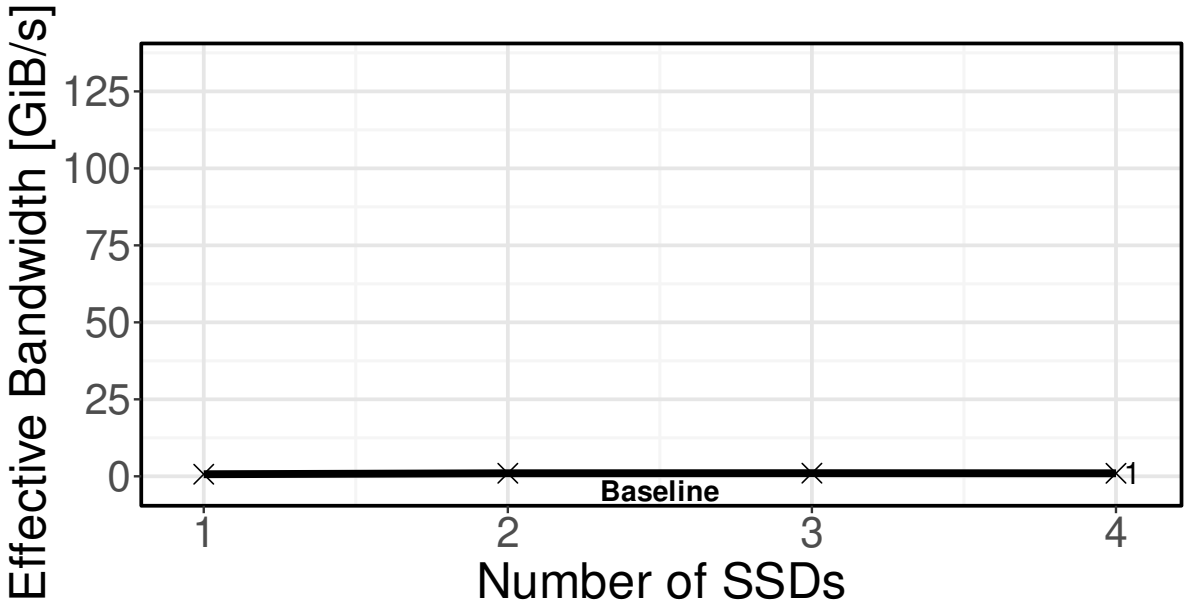
- GPU I/O stack differs from CPU's
- RG size controls I/O granularity
- Larger RG improve bandwidth



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

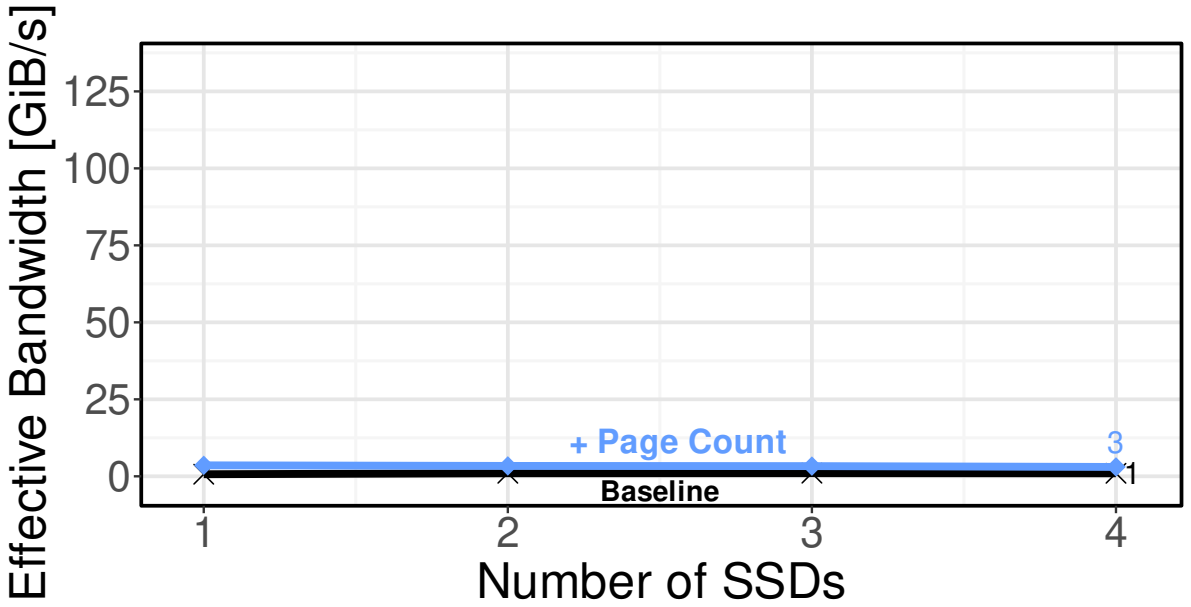
Reading 4 SSDs

- Single SSD saturated
- Scaling to multiple SSDs



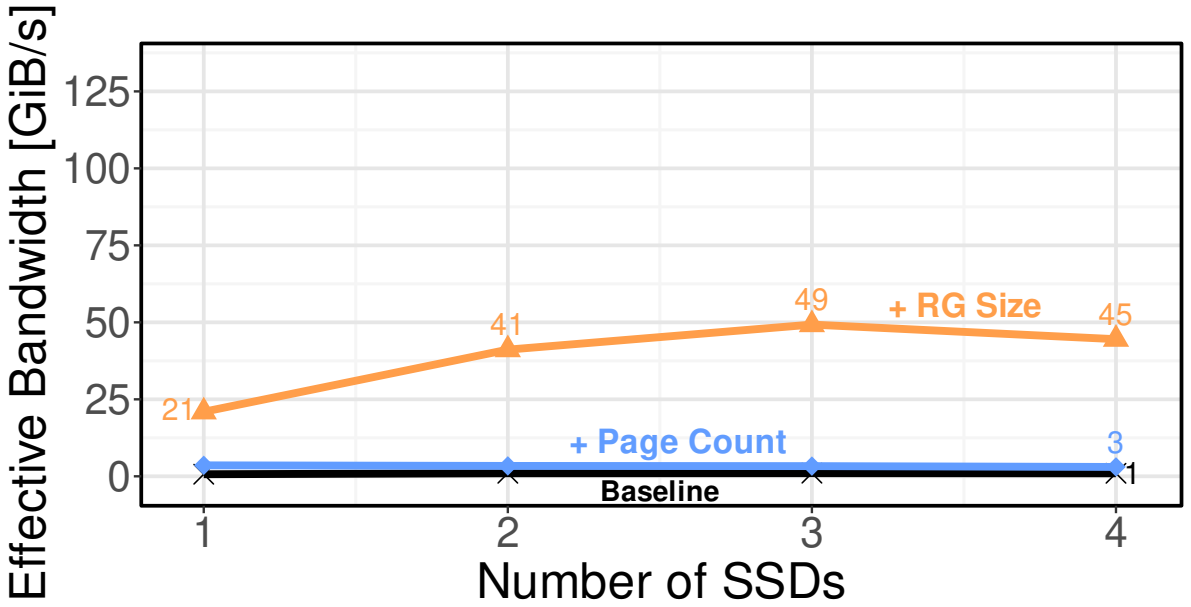
Reading 4 SSDs

- Single SSD saturated
- Scaling to multiple SSDs



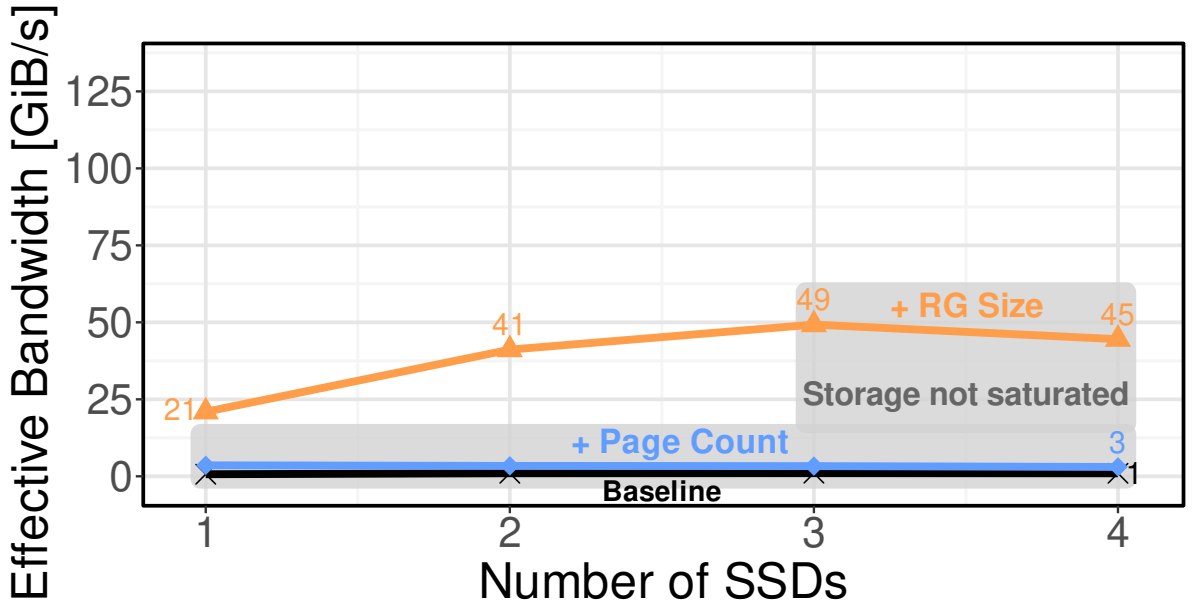
Reading 4 SSDs

- Single SSD saturated
- Scaling to multiple SSDs



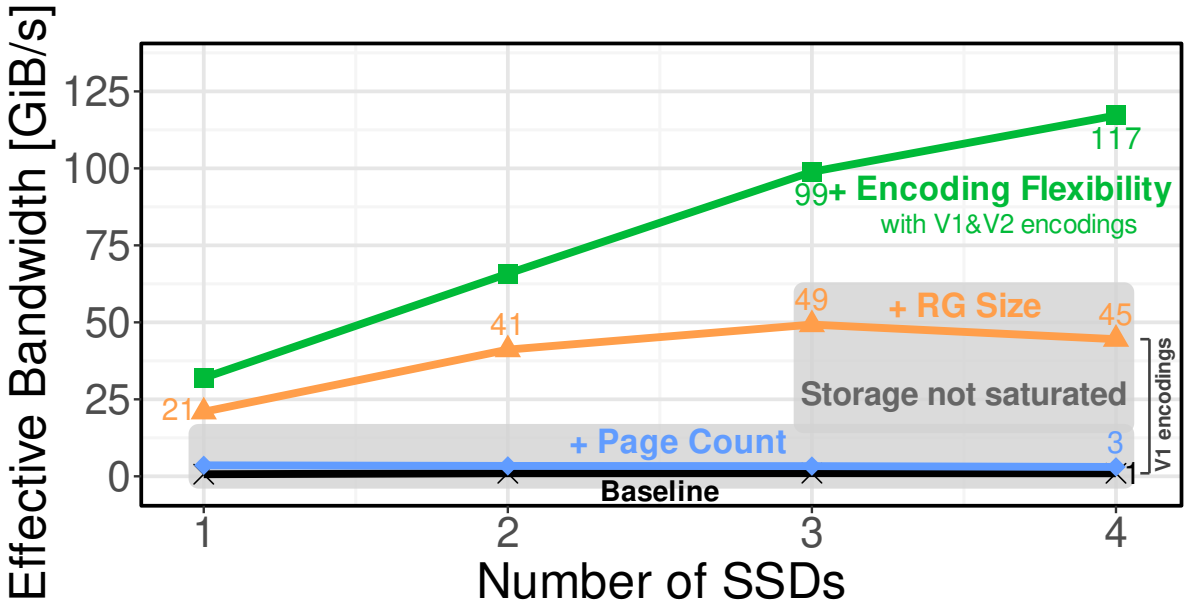
Reading 4 SSDs

- Single SSD saturated
- Scaling to multiple SSDs



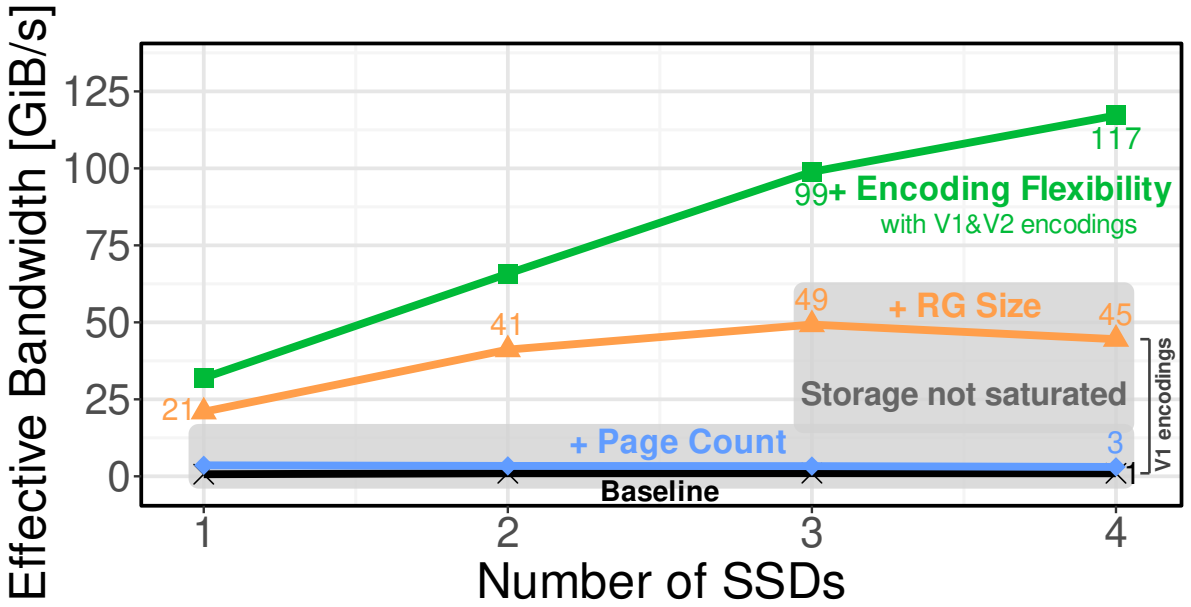
Config.: Encoding Flexibility

- Consider both V1 and V2



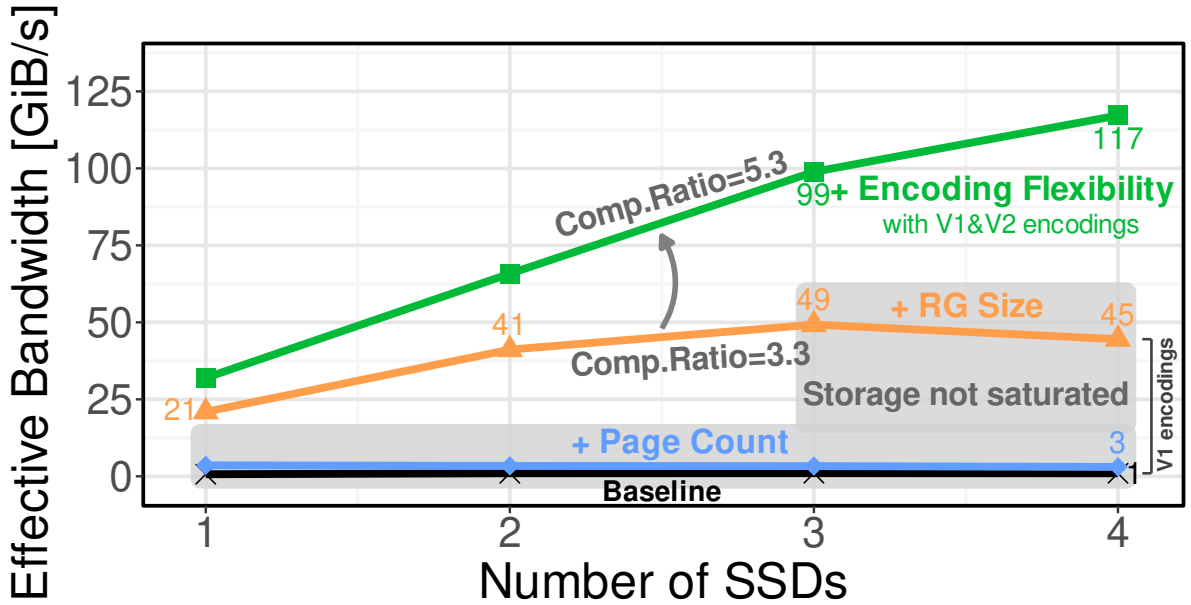
Config.: Encoding Flexibility

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



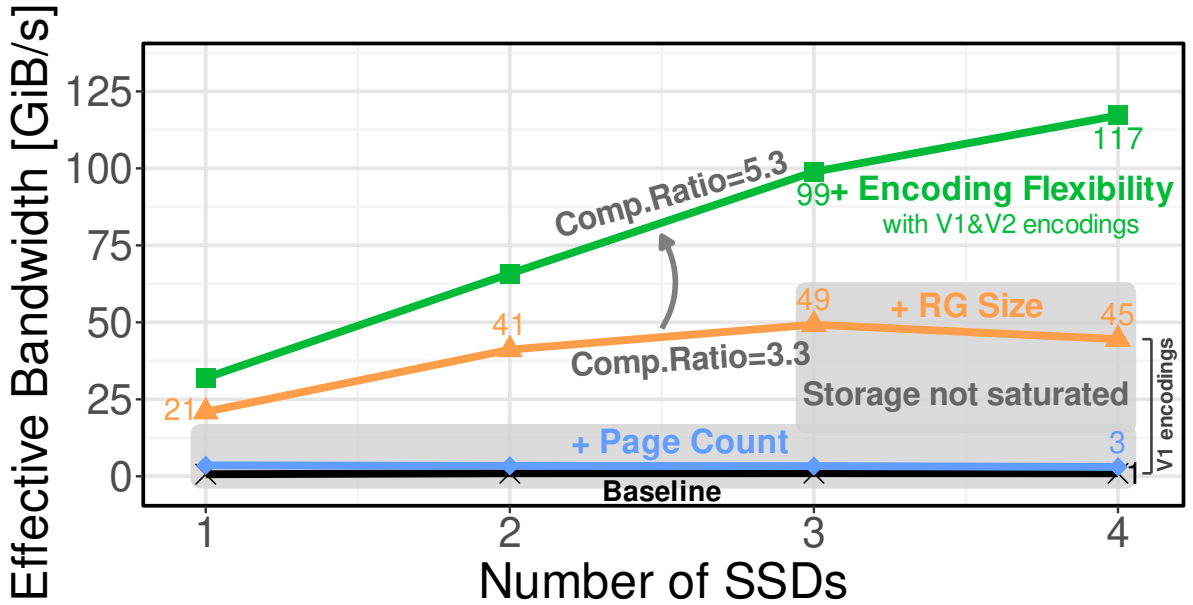
Config.: Encoding Flexibility

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



Config.: Encoding Flexibility

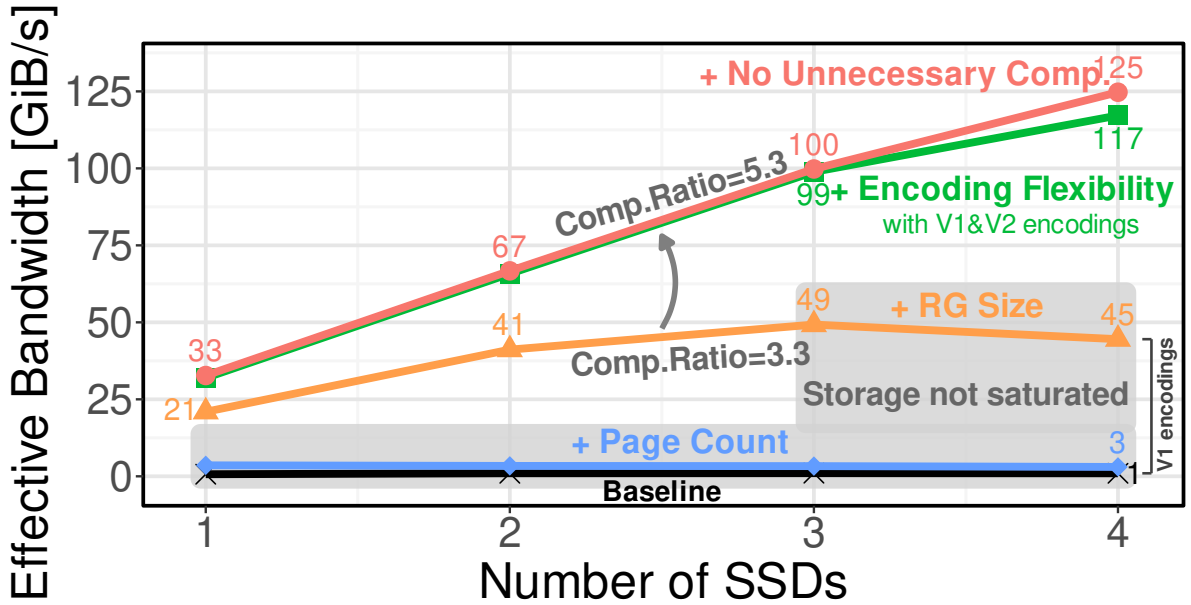
- Consider both V1 and V2
- Finalize the encoding producing the smallest size
- Improve compression ratio



Insight 3: Flexibly choosing encodings to achieve higher size reduction

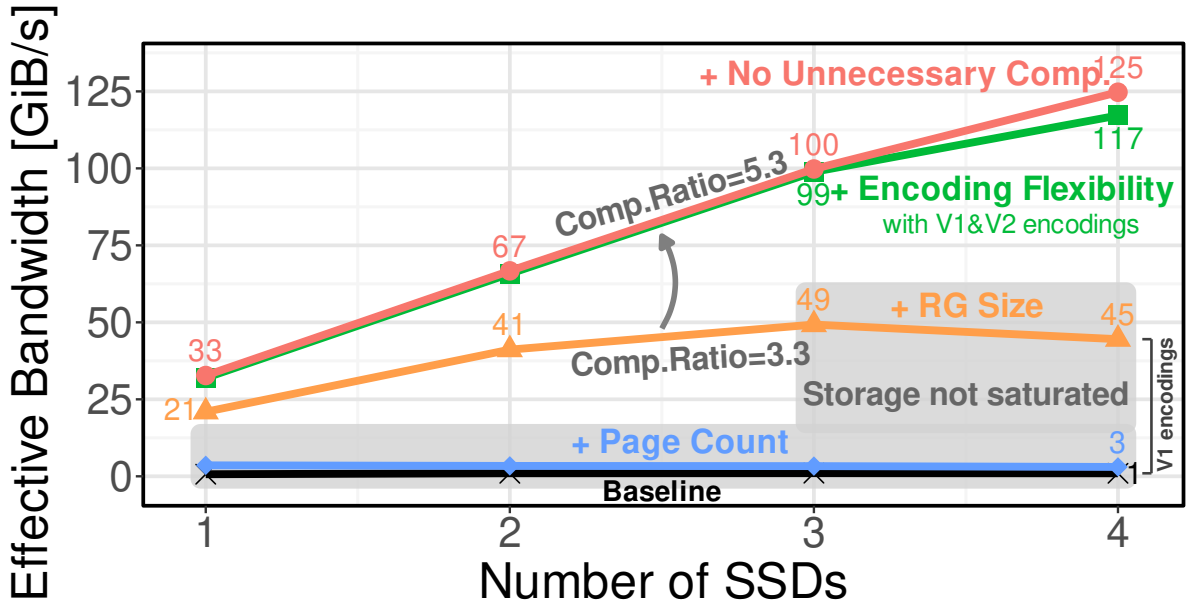
Config.: No Unnecessary Compression

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



Config.: No Unnecessary Compression

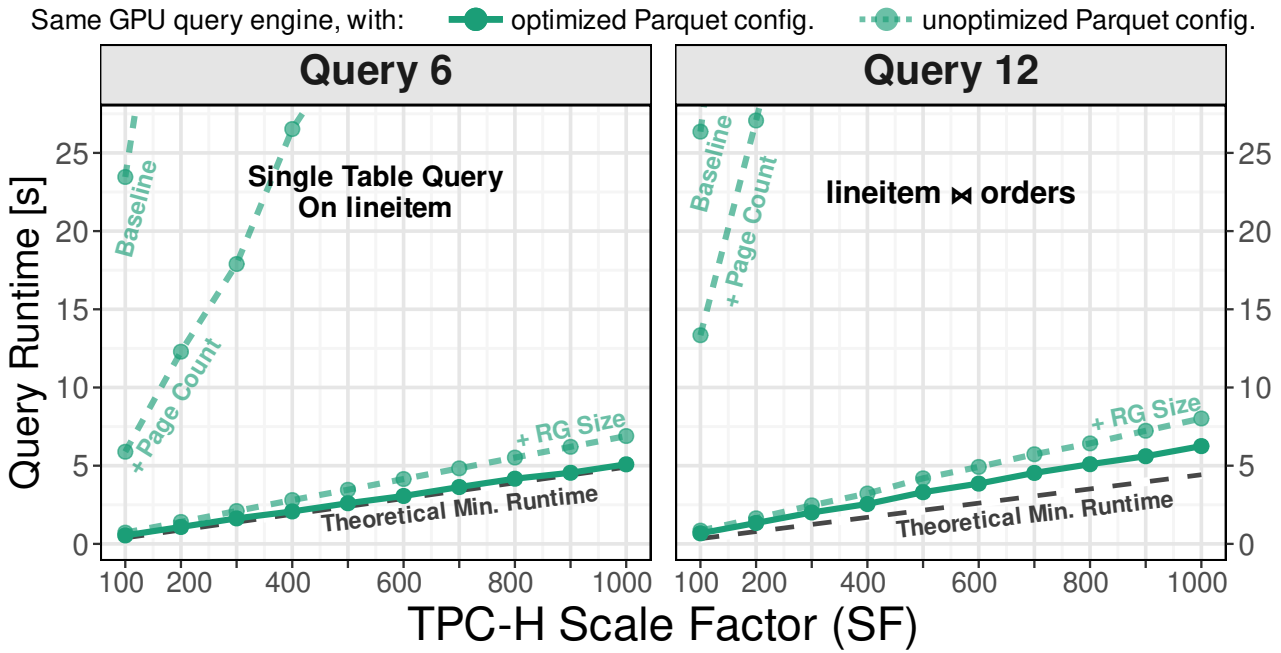
- Compress only when size reduction is meaningful
- Otherwise, uncompressed



Insight 4: Skip unnecessary compression if no size reduction

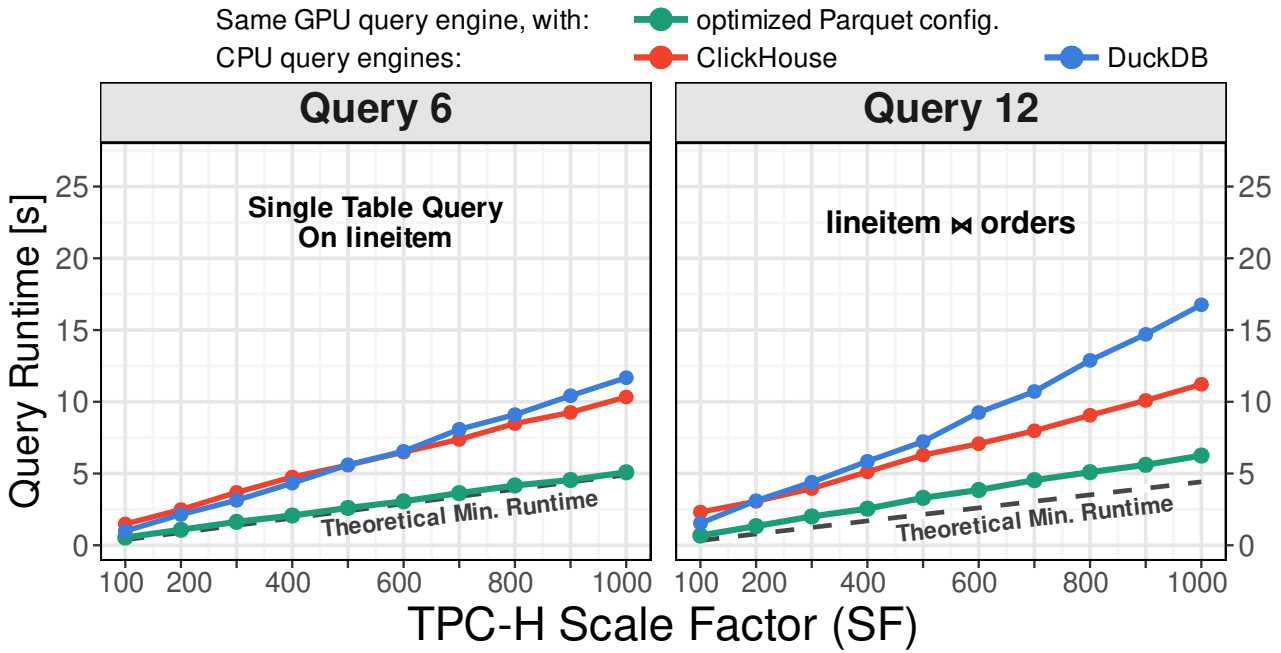
Query Processing With Parquet Improvements

- 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 ⟶

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

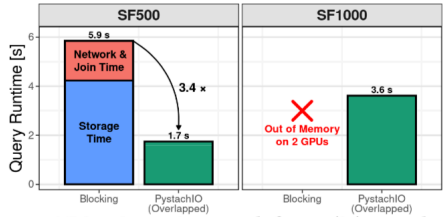


Figure 1: TPC-H Q3 runtime at scale factors (SF) 500 and 1000 on two GPUs over SSD-resident, non-co-partitioned tables. A blocking PyTorch-style baseline executes sequentially, leading to long runtimes and out-of-memory errors. In contrast, PystachIO overlaps storage I/O, networking, and computation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

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

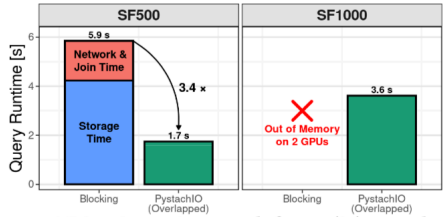


Figure 1: TPC-H Q3 runtime at scale factors (SF) 500 and 1000 on two GPUs over SSD-resident, non-co-partitioned tables. A blocking PyTorch-style baseline executes sequentially, leading to long runtimes and out-of-memory errors. In contrast, PystachIO overlaps storage I/O, networking, and computation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

Additional Work: Parquet Tools for CPUs & GPUs

Improving Parquet quality:

- **Viewing**
- **Linting** for suggestions
- **Rewriting** for optimization

Parquet Tools: A Better Parquet Is Parquet Itself

Jigao Luo*
TU Darmstadt
jigao.luo@tu-darmstadt.de

Xiangpeng Hao*
University of Wisconsin-Madison
xiangpeng.hao@wisc.edu

Andrew Lamb
InfluxData
alamb@influxdata.com

Carsten Binnig
TU Darmstadt & hessian.AI & DFKI
carsten.binnig@tu-darmstadt.de

ABSTRACT

Apache Parquet [2] is the dominant file format for analytical workloads, yet its performance is often criticized. When properly configured, Parquet can perform several times better than the default settings. However, configuration remains difficult in practice because effective tools are still lacking. In this demonstration, we present a workflow for optimizing Parquet using three tools: a viewer, a linter, and a rewriter. Used together, they can often substantially improve the realized performance when using Parquet. By better understanding the actual existing boundaries of Parquet, we hope to help future research efforts focus on the highest-value improvements for future file formats.

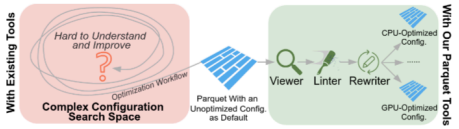
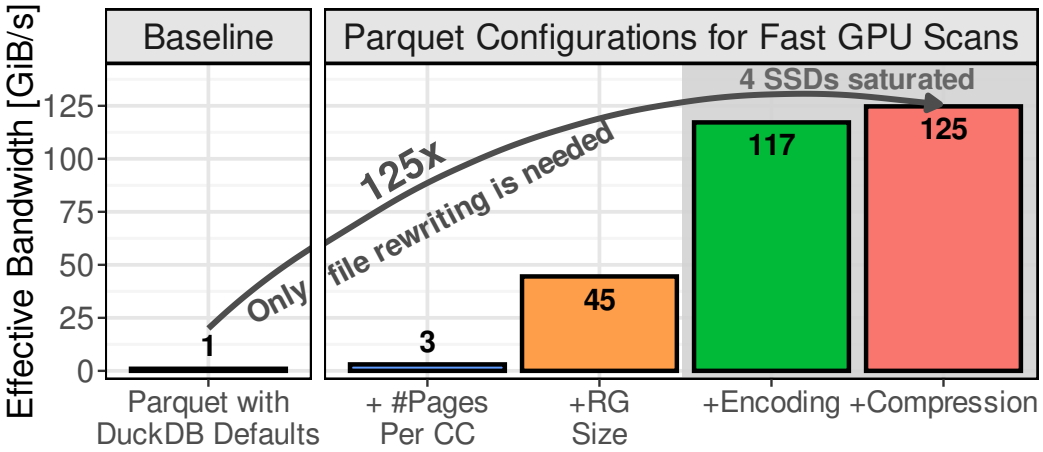


Figure 1: Parquet configuration workflows with existing tools (*left*) and our dedicated Parquet tools (*right*). *Left:* With existing tools, Parquet internal settings are difficult to understand, and most systems use suboptimal defaults. *Right:* Our tools visualize Parquet configurations, revealing opportunities to optimize for specific workloads within the existing format.

with:

Conclusion



- GPU-aware Parquet configuration matters: 4 Insights
- Parquet performs well once optimized
- Optimize Parquet before replacing it

Paper

Rewriter Code

TECHNISCHE UNIVERSITÄT DARMSTADT

SYSTEMS

Thanks for your attention

“How do you deal with OOM?”

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

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

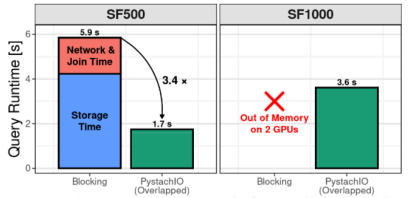
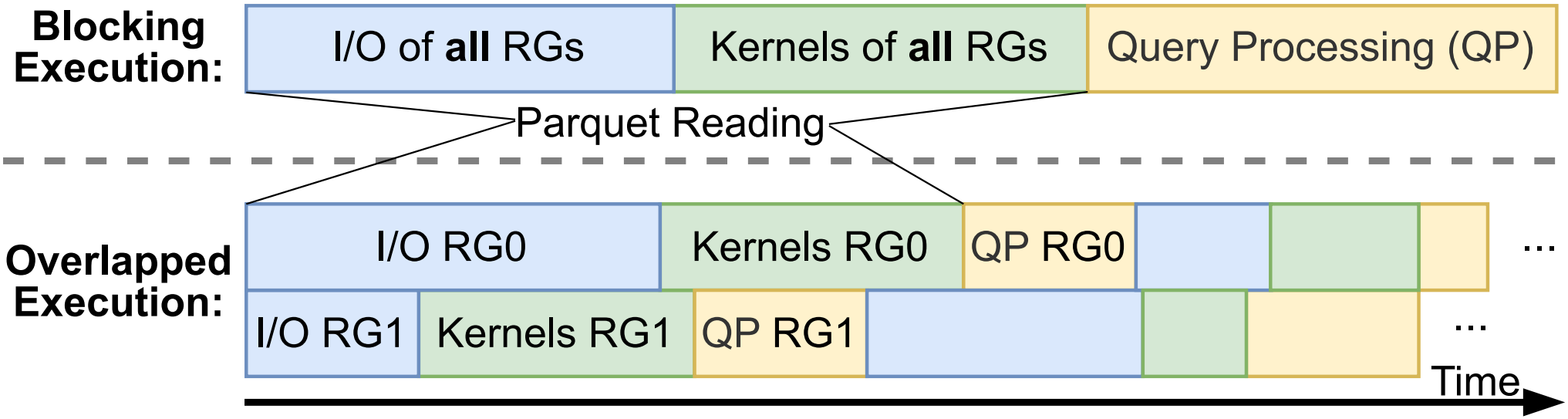
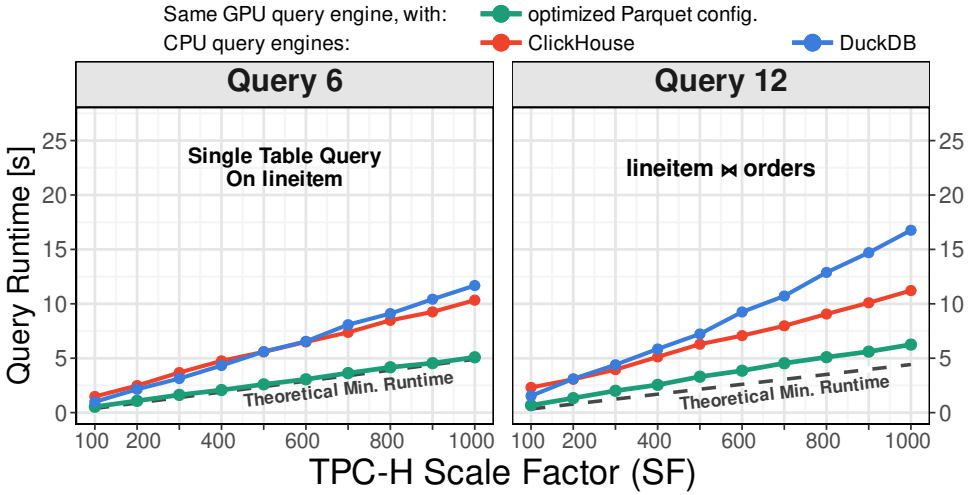


Figure 1: TPC-H Q3 runtime at scale factors (SF) 500 and 1000 on two GPUs over SSD-resident, non-co-partitioned tables. A blocking PyTorch-style baseline executes sequentially, leading to long runtimes and out-of-memory errors. In contrast, PystachIO overlaps storage I/O, networking, and computation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

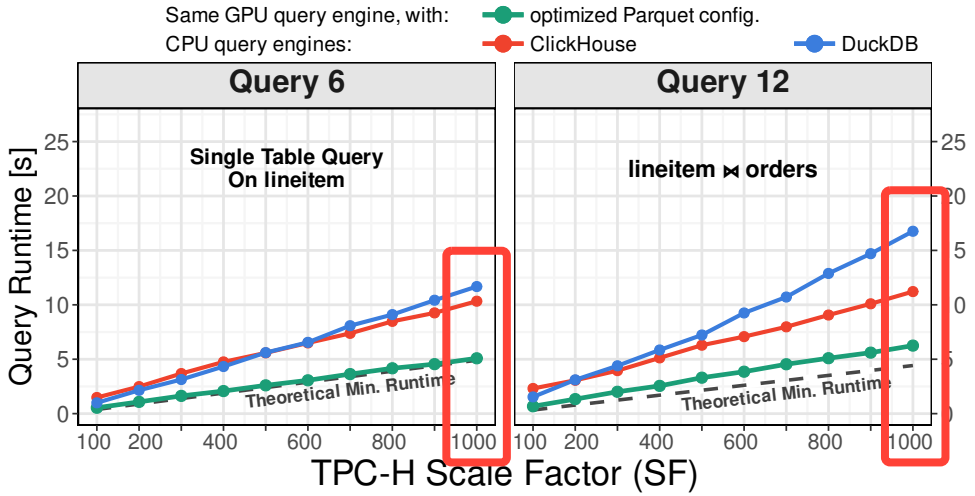
Overlapping: Reader & Query Engine



“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.