CWE41740: NVCOMP: GPU COMPRESSION/DECOMPRESSION

EXPERTS

MAIN ROOM

Ben Karsin, Senior AI Developer Technology Engineer

Nikolay Sakharnykh, Senior Manager, Developer Technology

Guillaume Thomas-Collignon, DevTech Compute

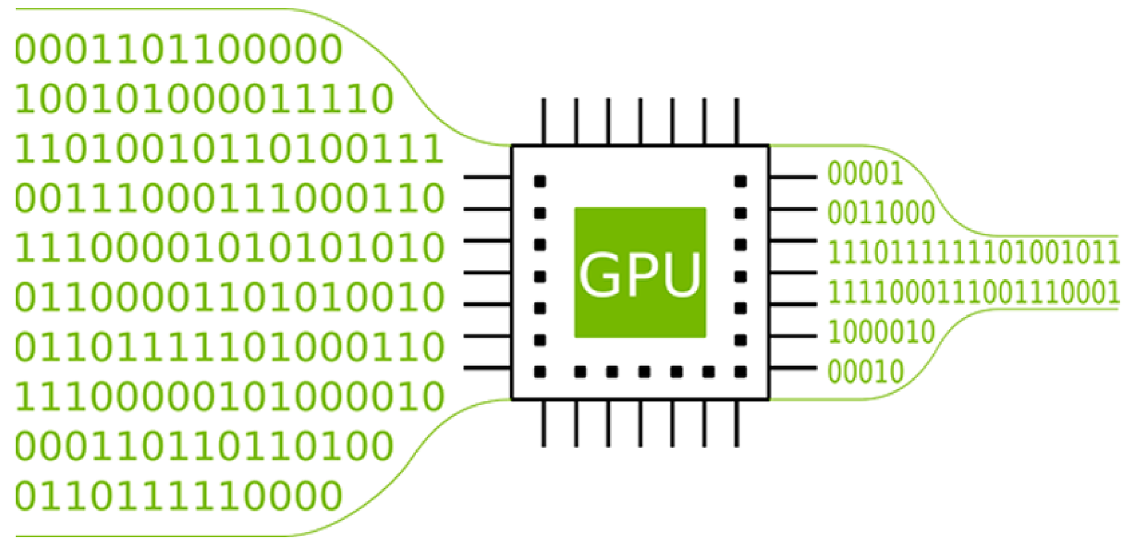
BREAKOUT ROOMS

Breakout room #1: Eric Schmidt, Topic: general nvCOMP topics, nvCOMP interfaces

Breakout room #2: Akshay Subramaniam, Topic: closed source compressors, LZ77-based compression/decompression

NVCOMP

CUDA library for GPU compression/decompression



- https://github.com/NVIDIA/nvcomp, https://developer.nvidia.com/nvcomp
- One-stop shop for optimized GPU implementations of lossless (de)compression methods with a standard C/C++ interface to switch between methods easily

- **Current release - new features in 2.2**
  - Redesigned easy to use high-level interface -> starting point for new users
  - 2 extremely fast compressors: ANS and GDeflate entropy-only

- **What's coming - new features in 2.3**
  - New compression algorithms: Zstd decompression, Deflate compression/decompression
  - High-level interface checksums

LOW-LEVEL INTERFACE

- Low-level interface targets advanced users:
  - metadata and chunking are handled *outside* of nvCOMP
  - supports batch compression/decompression of multiple streams
  - light-weight and fully asynchronous

```
// compute temporary memory size
nvcompBatchedLZ4DecompressGetTempSize(
  compress_data.size(), chunk_size, &decomp_temp_bytes);

// allocate GPU memory:
//   temporary storage, status pointers and decompression sizes
cudaMalloc(&d_decomp_temp, decomp_temp_bytes);
...

// run decompression kernel
nvcompBatchedLZ4DecompressAsync(
  compress_data.ptrs(), compress_data.sizes(),
  decomp_data.sizes(), d_decomp_sizes, compress_data.size(),
  d_decomp_temp, decomp_temp_bytes, decomp_data.ptrs(),
  d_status_ptrs,
  stream);
```

Example of decompressing a batch of buffers on the GPU

HIGH-LEVEL INTERFACE

- The **high-level** interface was redesigned for nvCOMP 2.2 and enables the easiest way to ramp up and use nvCOMP in applications
  - Metadata and chunking are handled *internally* by nvCOMP
  - Unlike the low-level interface, it is best used on large contiguous buffers
  - It can manage the required scratch space for the user.
  - In nvCOMP 2.2 all compressors are available through both low-level and high-level APIs.
  - Maintains a similar level of performance as the low-level interface for large input buffers.
  - Using the high-level interface metadata, you can decompress an HLIF-compressed buffer without knowing how it was compressed

```
// read the compressed data from the GDS file into the device buffer
cuFileRead(cf_handle, d_compressed, lcomp, 0, 0);

// nvcompManager configured using the compressed data (synchronous)
auto nvcomp_manager = create_manager(d_compressed, stream);

// Configure decompression (synchronous)
auto decomp_config = nvcomp_manager->configure_decompression(d_compressed);

// Decompress the data (asynchronous)
nvcomp_manager->decompress(d_output, d_compressed, decomp_config);
```

Example of reading a file from disk with GPUDirectStorage and decompressing with nvCOMP

NVCOMP 2.2 METHODS

- **Cascaded**: Novel high-throughput compressor ideal for analytical or structured/tabular data.
- **LZ4**: General-purpose no-entropy byte-level compressor well-suited for a wide range of datasets.
- **Snappy**: Similar to LZ4, this byte-level compressor is a popular existing format used for tabular data.
- **GDeflate**: Proprietary compressor with entropy encoding and LZ77, high compression ratios on arbitrary data.
- **Bitcomp**: Proprietary compressor designed for floating point data in Scientific Computing applications.
- **ANS**: Proprietary entropy encoder based on asymmetric numeral systems (ANS).

Using our benchmarking capability, you can quickly test the different methods to find the best one for your usecase. The below example benchmarks the low-level interface against a given file.

```
// You can replace lz4 below with one of <cascaded | snappy | gdeflate | bitcomp | ans> to test out the
// other algorithms on your dataset
./bin/benchmark_lz4_chunked -f /data/nvcomp/benchmark/mortgage-2009Q2-col4-float.bin
----------
files: 1
uncompressed (B): 164527964
comp_size: 148256777, compressed ratio: 1.11
compression throughput (GB/s): 7.00
decompression throughput (GB/s): 69.46
```

COMPRESSION RATIOS
Real-world data analytics
Cascaded
LZ4
Snappy
GDeflate-0
GDeflate-2
ANS
Bitcomp
Ratio (log scale)
128
64
32
16
8
4
2
1
col 0 long
col 12 string
col 2 string
col 3 decimal
col 30 string
col 4 decimal
col 5 decimal
col 6 decimal
col 7 decimal
col 9 decimal

COMPRESSION THROUGHPUT

A100 40GB

NVIDIA

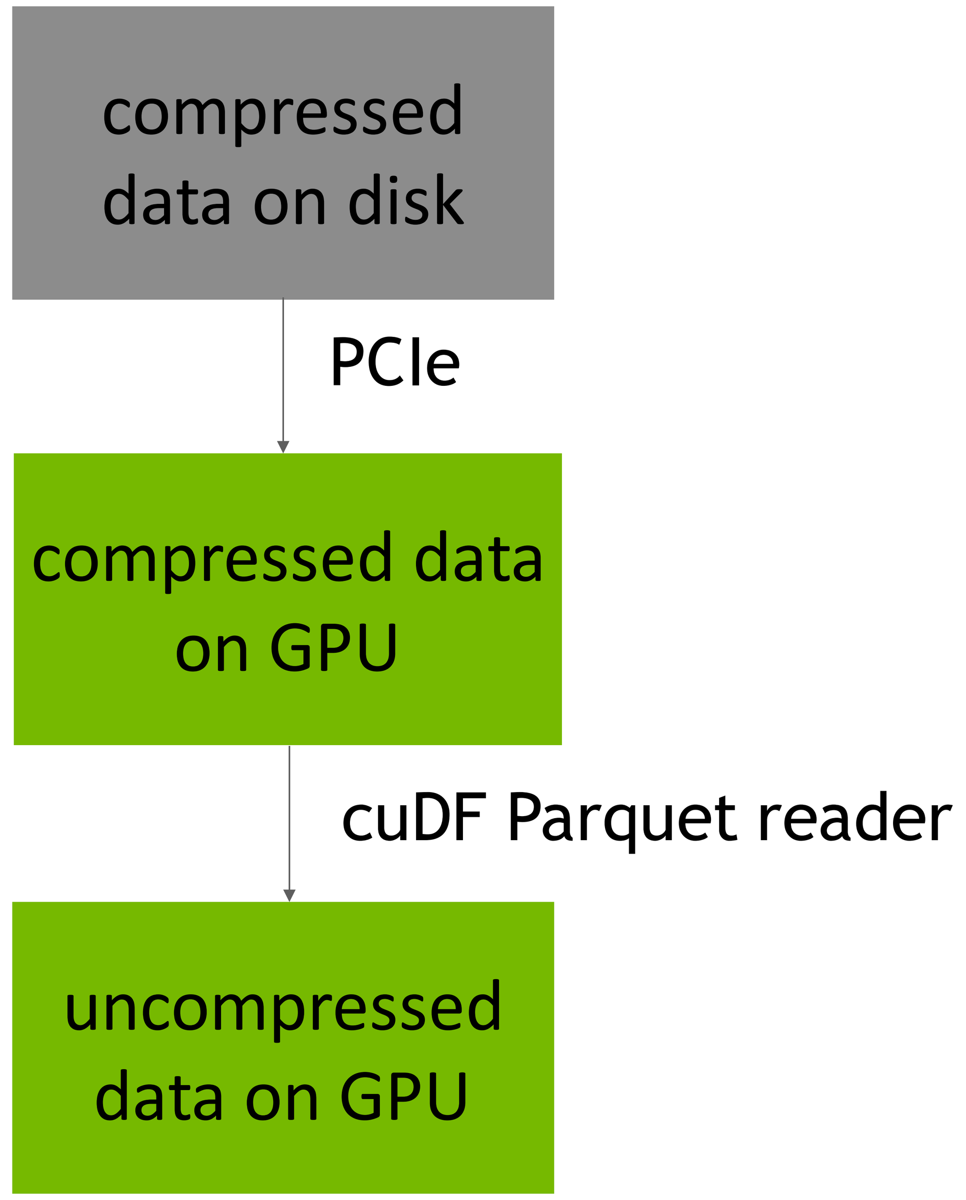
DECOMPRESSION THROUGHPUT

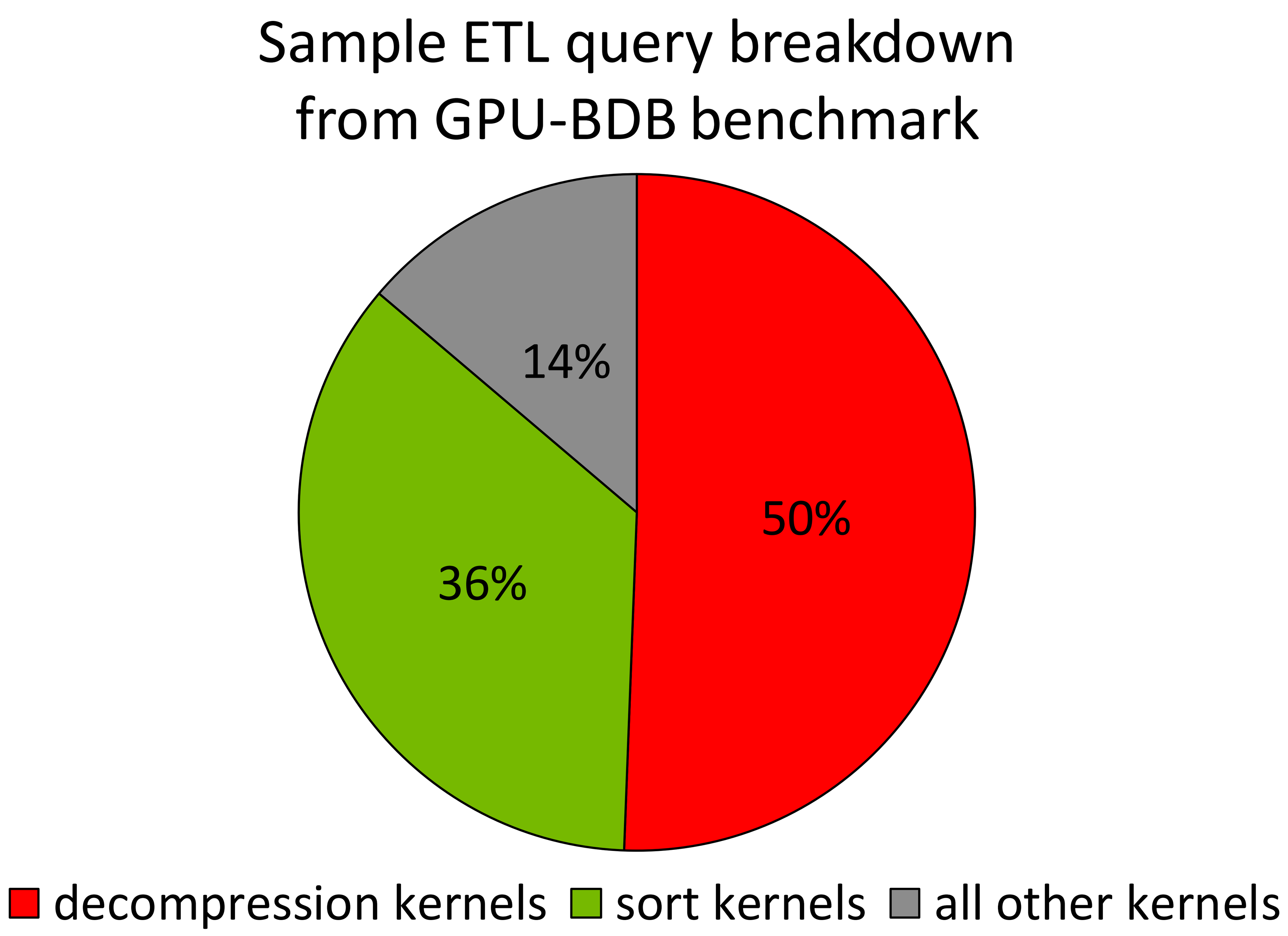
A100 40GB

USE CASES

ACCELERATING IO IN DATA CENTER

- Decompression is a huge bottleneck in RAPIDS – most expensive operation across many real-world queries



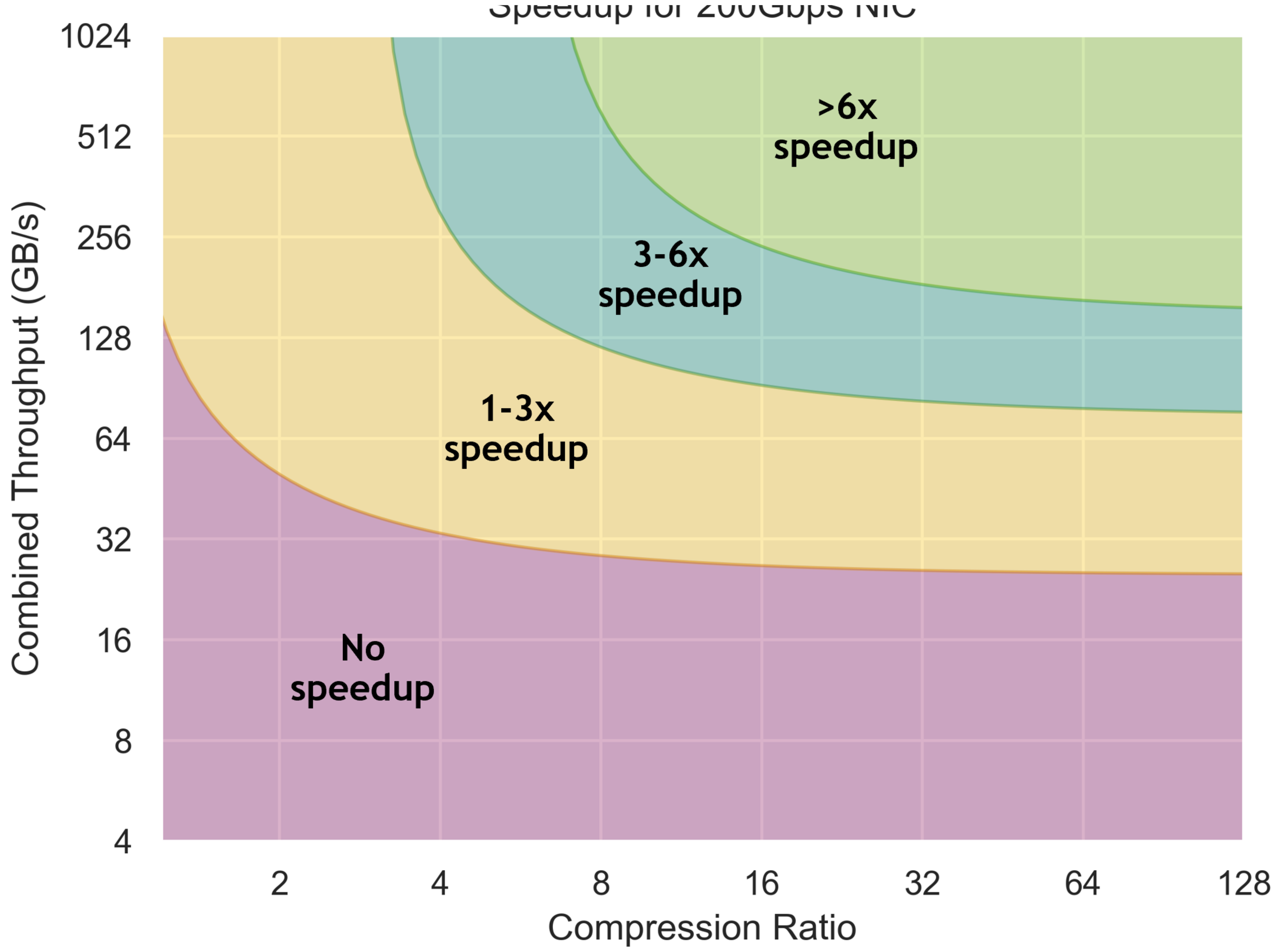


Decompression speed is more important than compression,
must support common formats

ACCELERATING COMMUNICATIONS

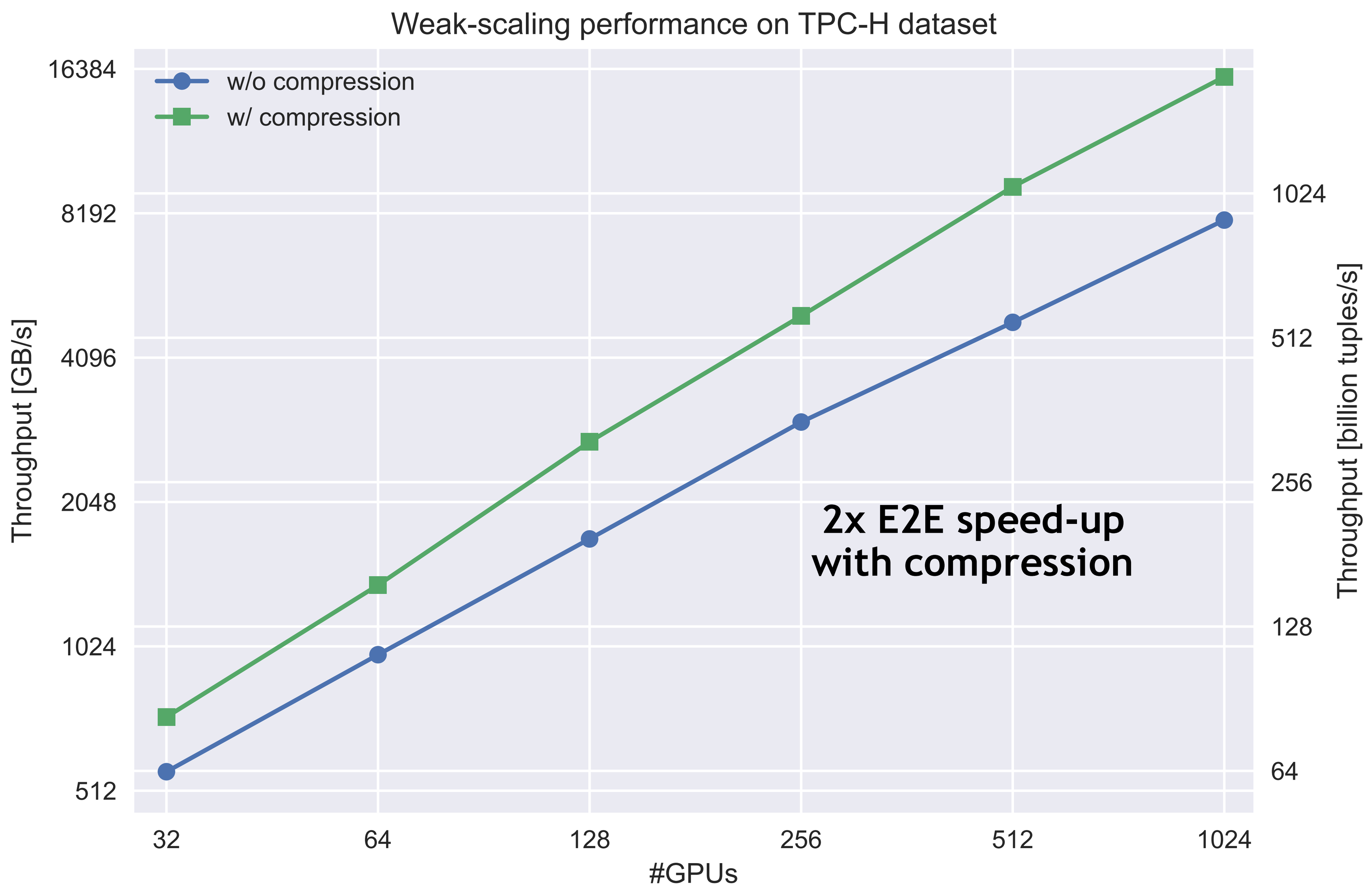
- Improve all-to-all communications on slow networks

Speed-up of applying compression vs sending raw data over 200Gbps NIC



"Scaling Joins to a Thousand GPUs", ADMS'21
http://www.adms-conf.org/2021-camera-ready/gao_adms21.pdf

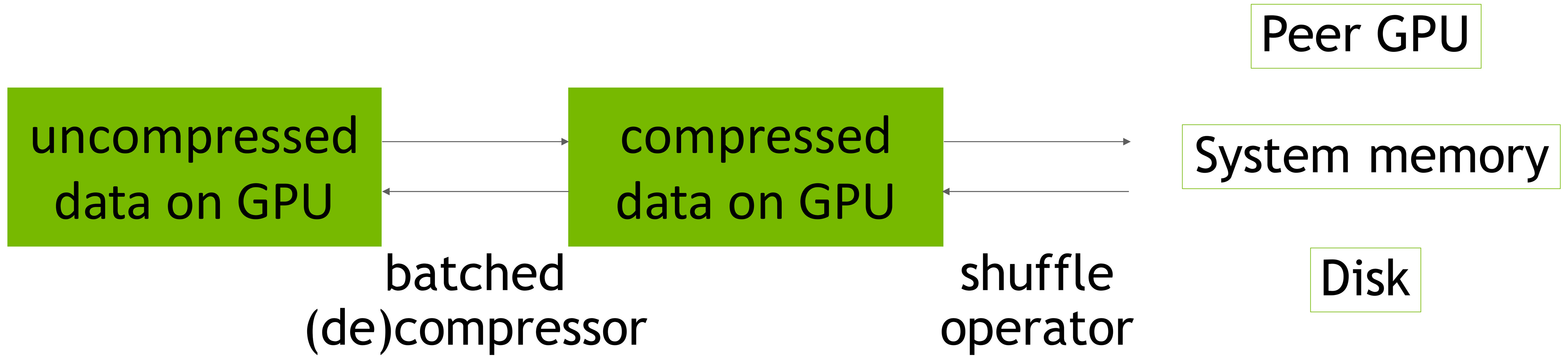
Distributed Join Performance
on DGX SuperPod (Selene)



Both compression and decompression speed are important,
flexible to use any codec

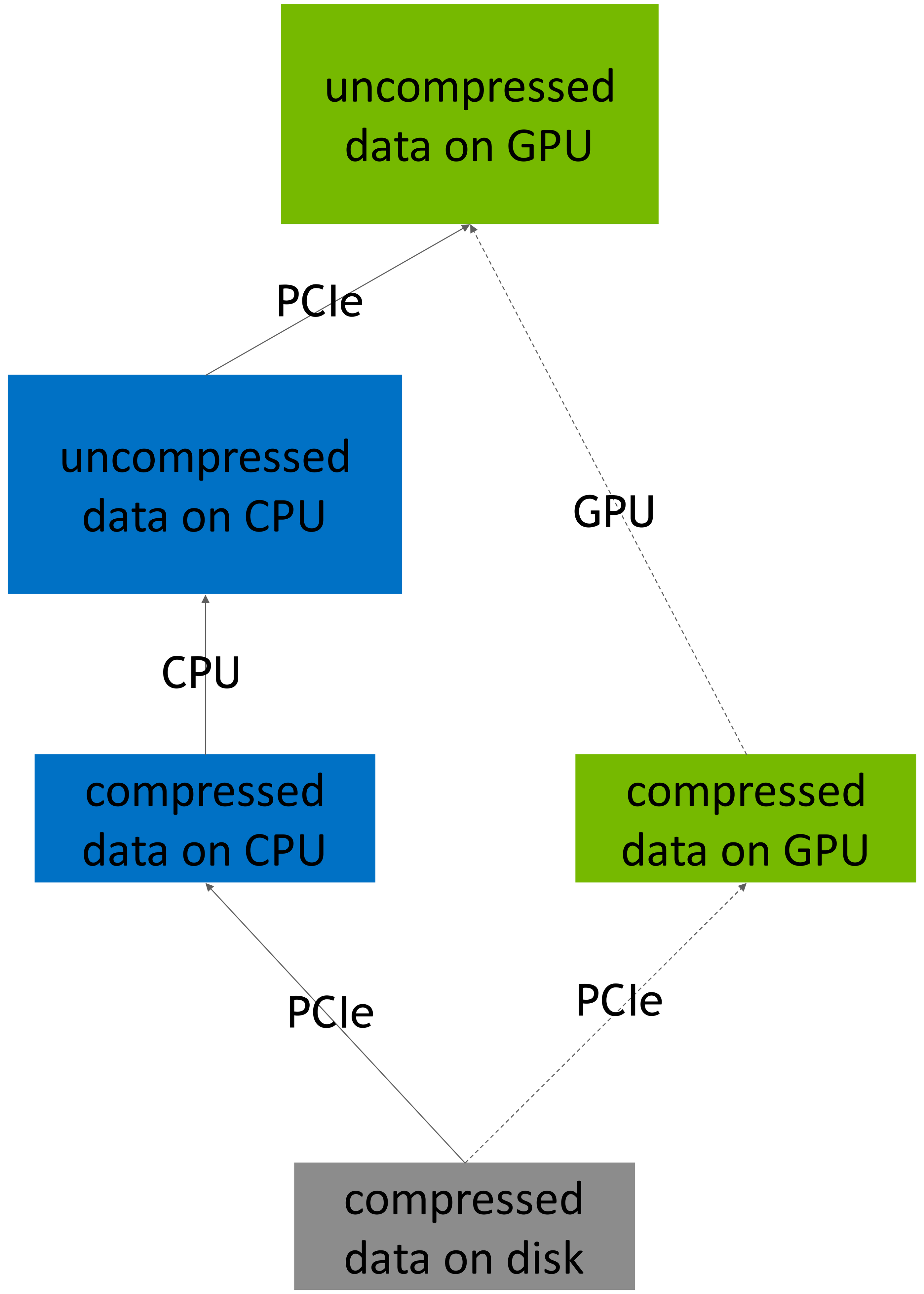
ACCELERATING SHUFFLE IN SPARK

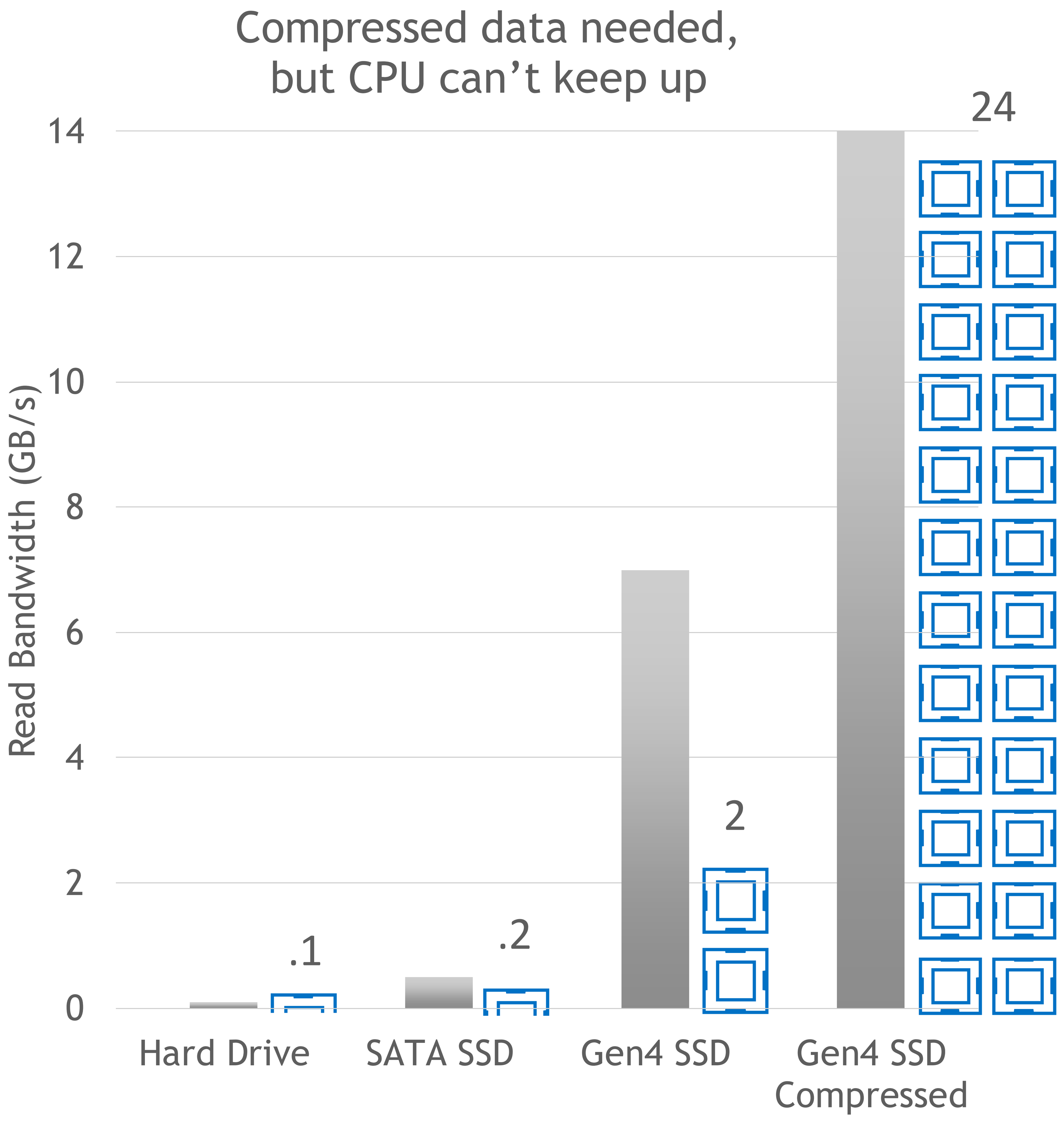
- Compress data during shuffle operation in Spark



Both compression and decompression speed are important, flexible to use any codec

ACCELERATING IO IN GRAPHICS





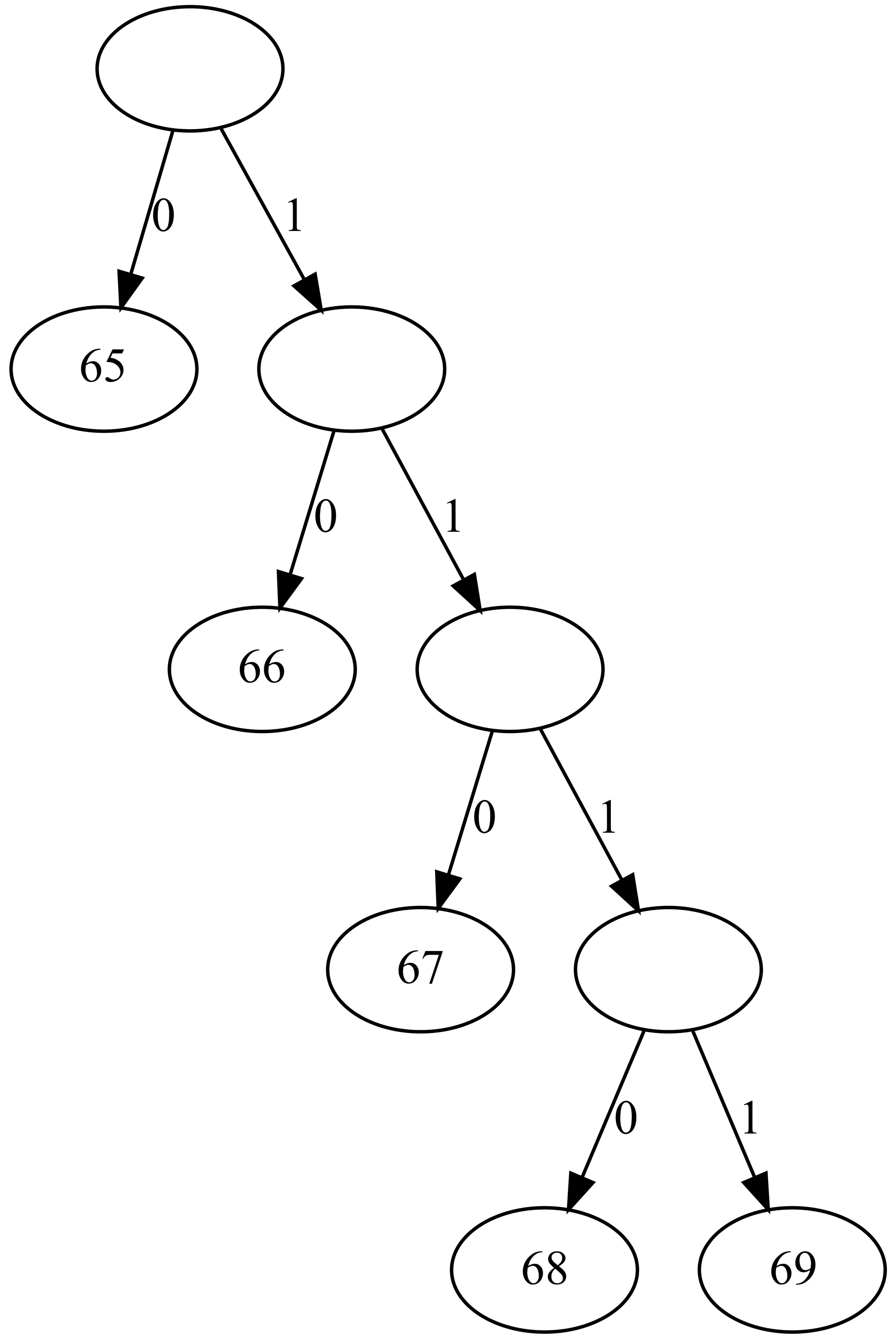
https://www.nvidia.com/en-us/geforce/news/rtx-io-gpu-accelerated-storage-technology/

ARE GPUS A GOOD FIT FOR (DE)COMPRESSION ALGORITHMS?

COMPRESSION TECHNIQUES

- Fundamental compression ops
    - **Data deduplication** – RLE, LZ, Dictionary
    - **Difference encoding** – Delta, Frame-of-reference
    - **Entropy encoding** – bit-packing, Huffman, ANS
- Many well-knowns schemes combine these together
    - Deflate = LZ + Huffman
    - zstd = LZ + Huffman + FSE/ANS
- Lots of pre-processing techniques (usually domain-specific)
    - Bit-plane transpose, sort, BWT

Huffman tree



Bit-shuffle/transpose

0	0	1	7	4	8	7	e
0	0	1	7	4	8	8	6
0	0	1	7	4	8	8	c
0	0	1	7	4	8	8	c
0	0	1	7	4	8	9	0

0	0	0	0	0
0	0	0	0	0
1	1	1	1	1
7	7	7	7	7
4	4	4	4	4
8	8	8	8	8
7	8	8	8	9
e	6	c	c	0

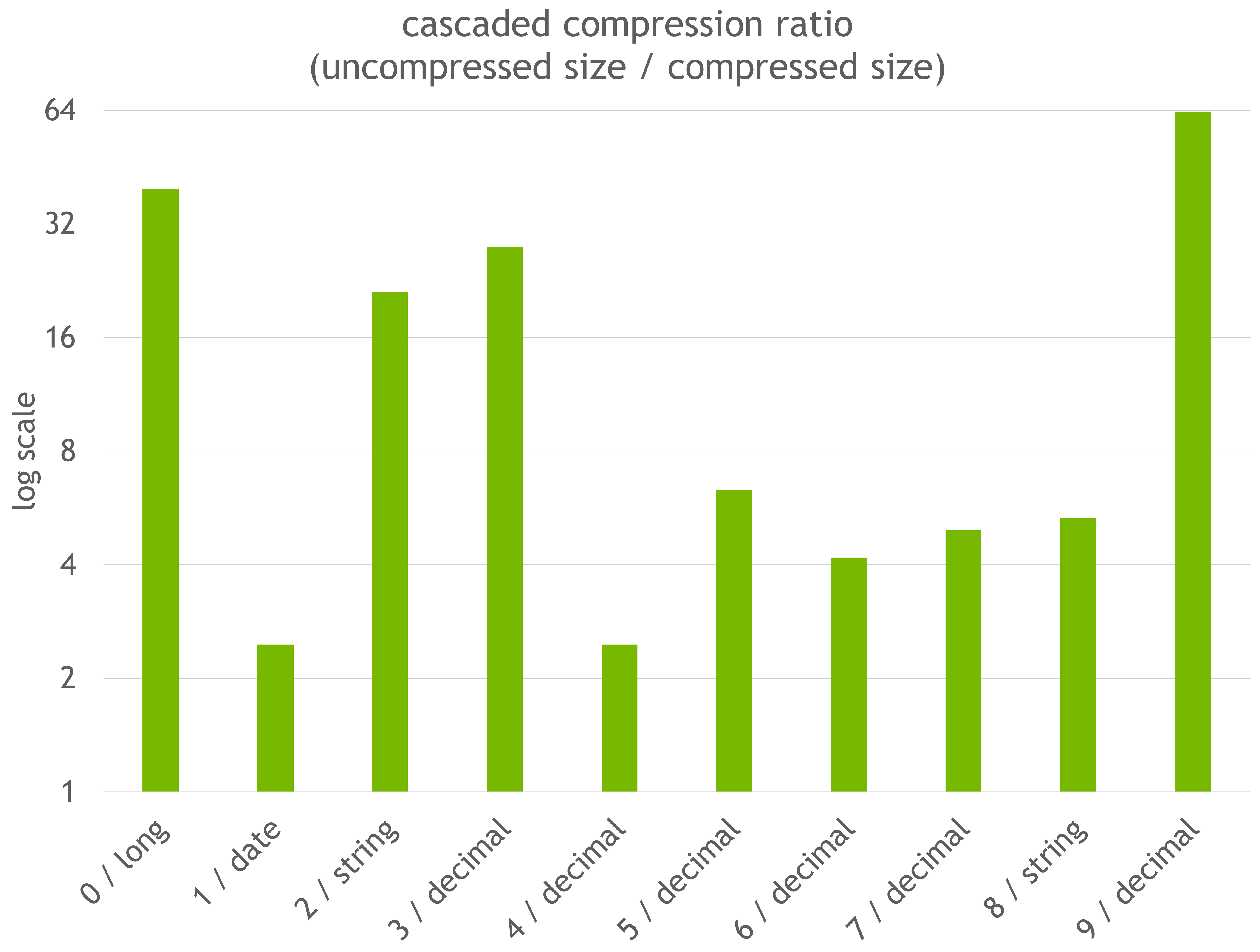
CASCADED COMPRESSOR

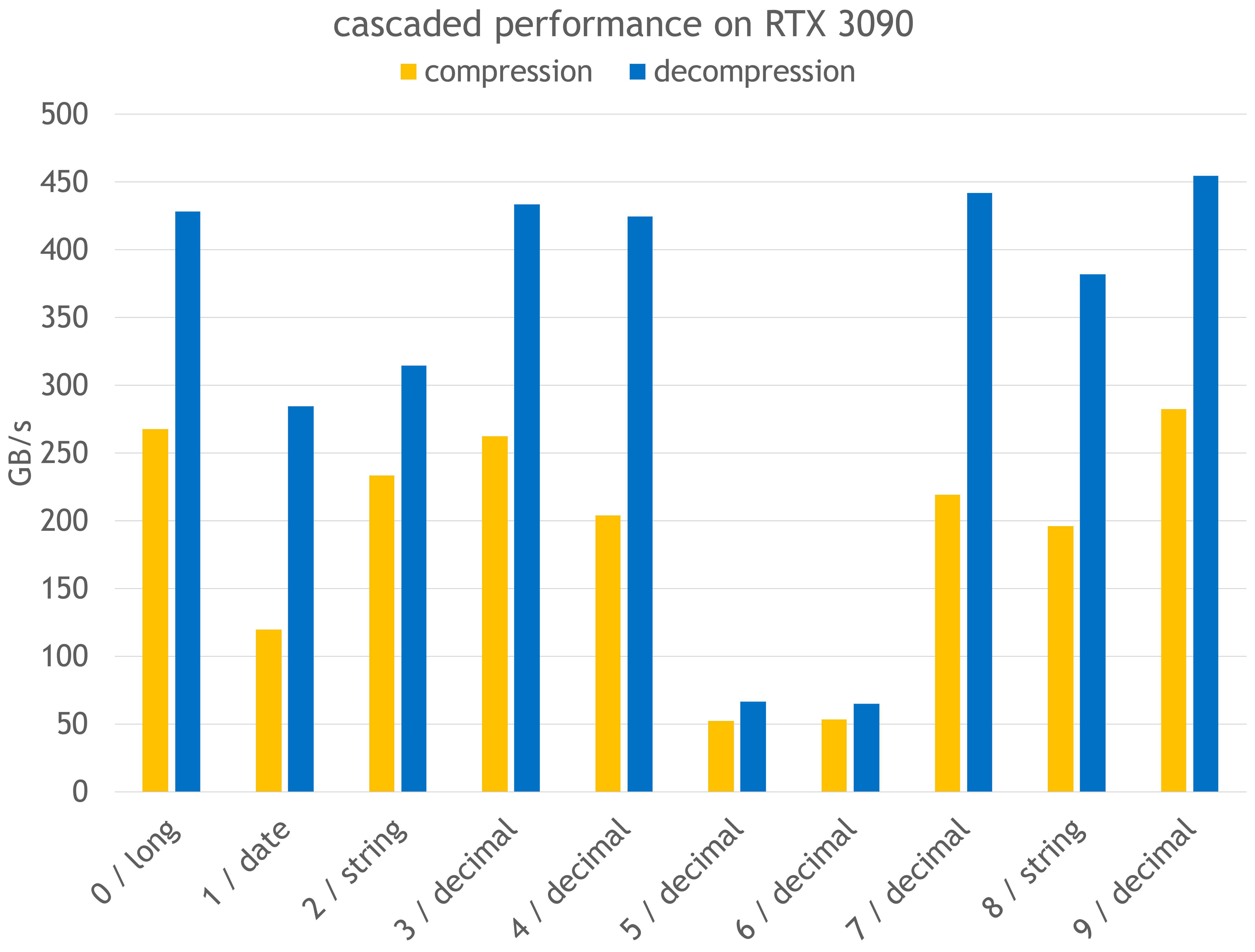
Combining the blocks together: RLE + Delta + FOR + bit-packing

More detail in the GTC'20 talk: [Software-Based Compression for Analytical Workloads]

CASCADED RATIOS AND PERF

Great fit for structured data from analytics datasets



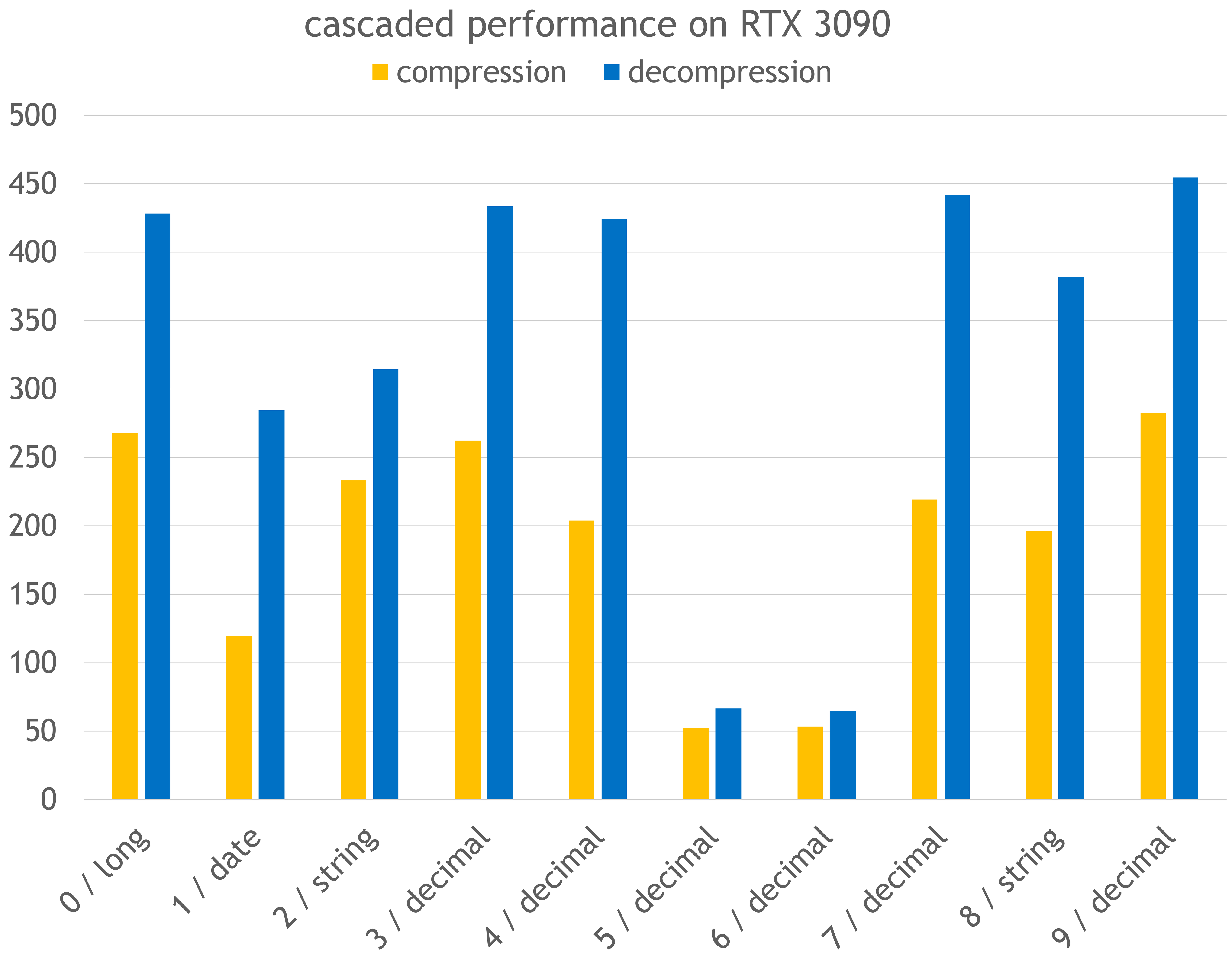


decimal interpreted as 8B integer, string and date as 4B integers

Dataset is derived from Fannie Mae's Single-Family Loan Performance Data and can be obtained here: https://rapidsai.github.io/demos/datasets/mortgage-data
Each column is 100-200MB of uncompressed data
Sample row from the dataset: `100005072756|12/01/2001|GMAC MORTGAGE, LLC|8.0|124352.34|12.0|348.0|0.0|12/2030|27100.0|...`

CASCADED RATIOS AND PERF

Great fit for structured data from analytics datasets



decimal interpreted as 8B integer, string and date as 4B integers

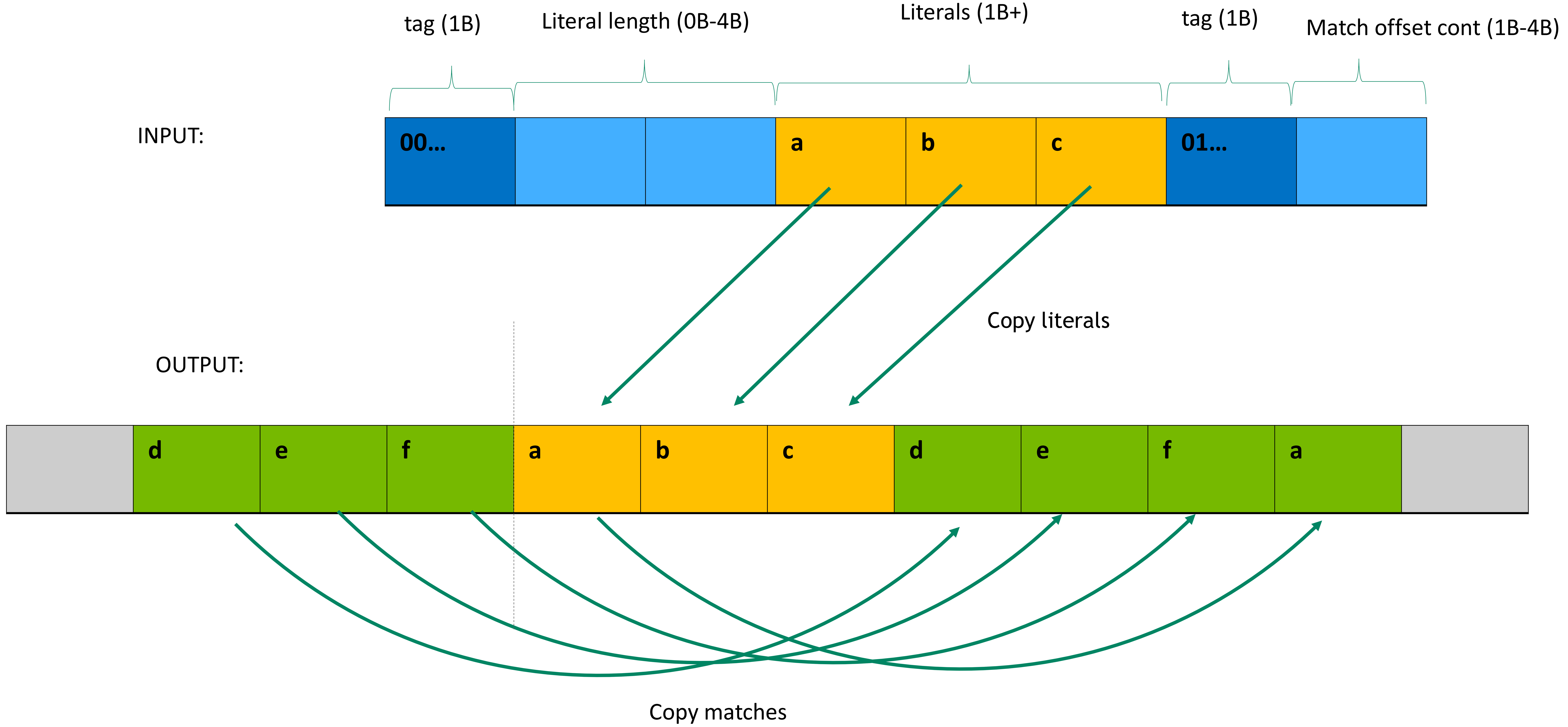
Dataset is derived from Fannie Mae's Single-Family Loan Performance Data and can be obtained here: https://rapidsai.github.io/demos/datasets/mortgage-data
Each column is 100-200MB of uncompressed data
Sample row from the dataset: `100005072756|12/01/2001|GMAC MORTGAGE, LLC|8.0|124352.34|12.0|348.0|0.0|12/2030|27100.0|...`

LZ4

4bit literal length
4bit match length
token (1B)
Literal length LSIC (0B+)
Literals (0B+)
offset (2B)
Match length LSIC (0B+)
INPUT:
255
34
a
b
c
255
255
12
Copy literals
OUTPUT:
d
e
f
a
b
c
d
e
f
a
Copy matches (min 4B)

SNAPPY

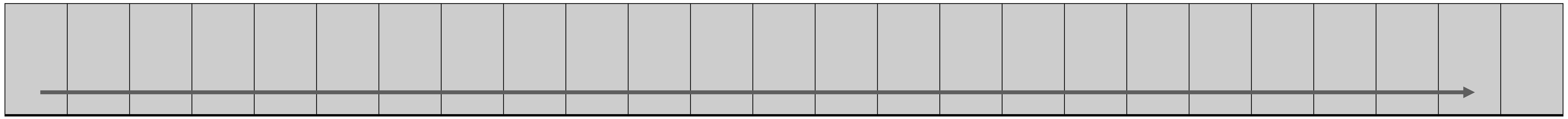
First 2 bits of the tag byte:
00 – literal, 6bit len, 0B-4B len
01 – copy, 3bit len, 3bit offset, 1B offset
10 – copy, 6bit len, 2B offset
11 – copy, 6bit len, 4B offset



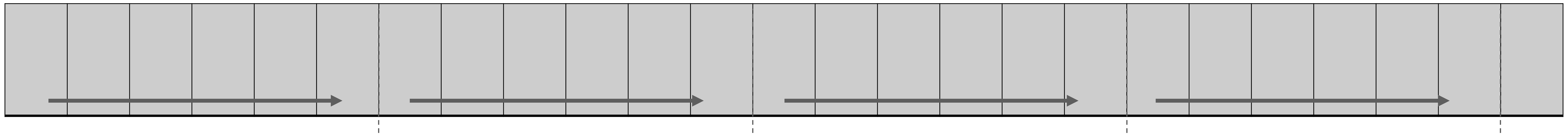
LEVELS OF PARALLELISM

Chunking is critical

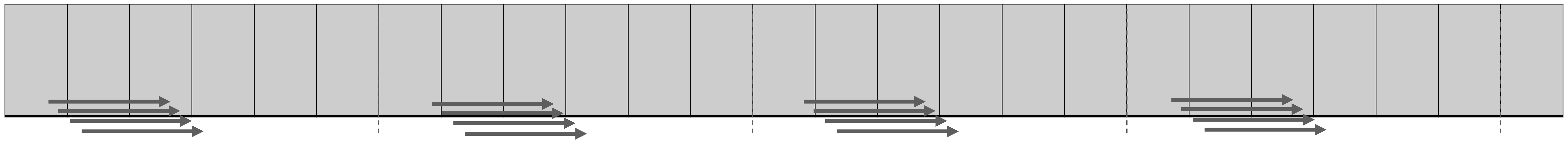
one LZ stream – difficult to map to the GPU, creates dependencies between copies, future copies depend on previous data



many smaller independent LZ streams – better mapping to the GPU, more concurrent tasks; need to store offsets/sizes



warp or threadblock-level parallelism within a LZ stream – threads cooperating on writing the output buffer



64KB / warp

More detail in the GTC’21 talk: [Optimizing Lossless Compression Algorithms on the GPU]

TAKEAWAY

1. GPU's fast memory and lots of compute power enables efficient compression/decompression
   - Now developers have access to these methods through a new core CUDA library nvCOMP: https://github.com/NVIDIA/nvcomp
2. Think about ways to leverage compression in your applications
   - We'd like to hear about your use cases!
3. Try nvCOMP and provide feedback - participate in shaping out the future of compression/decompression on GPU
   a) Have an idea about a new feature?
   b) Provide feedback on the interface?
   c) Report an issue?
   d) Would like to contribute?


NVIDIA