SCALING JOINS TO A THOUSAND GPUS

Hao Gao, Nikolay Sakharnykh, August 16, 2021

MEMORY PERFORMANCE

Why GPUs are suitable for join?

- **AMD EPYC 7742 CPU (Rome)**
  - DDR4: 8 channels, 64-bit per channel, 3200MT/s
  - Peak memory bandwidth **205GB/s**
  - Measured random 8B access **4.52GB/s**[1]
- **NVIDIA A100 GPU (Ampere)**
  - HBM2e: 5120-bit bus width, 1512MHz
  - Peak memory bandwidth **1935GB/s**
  - Measured random 8B access **134GB/s**
  - But only **80GB** capacity

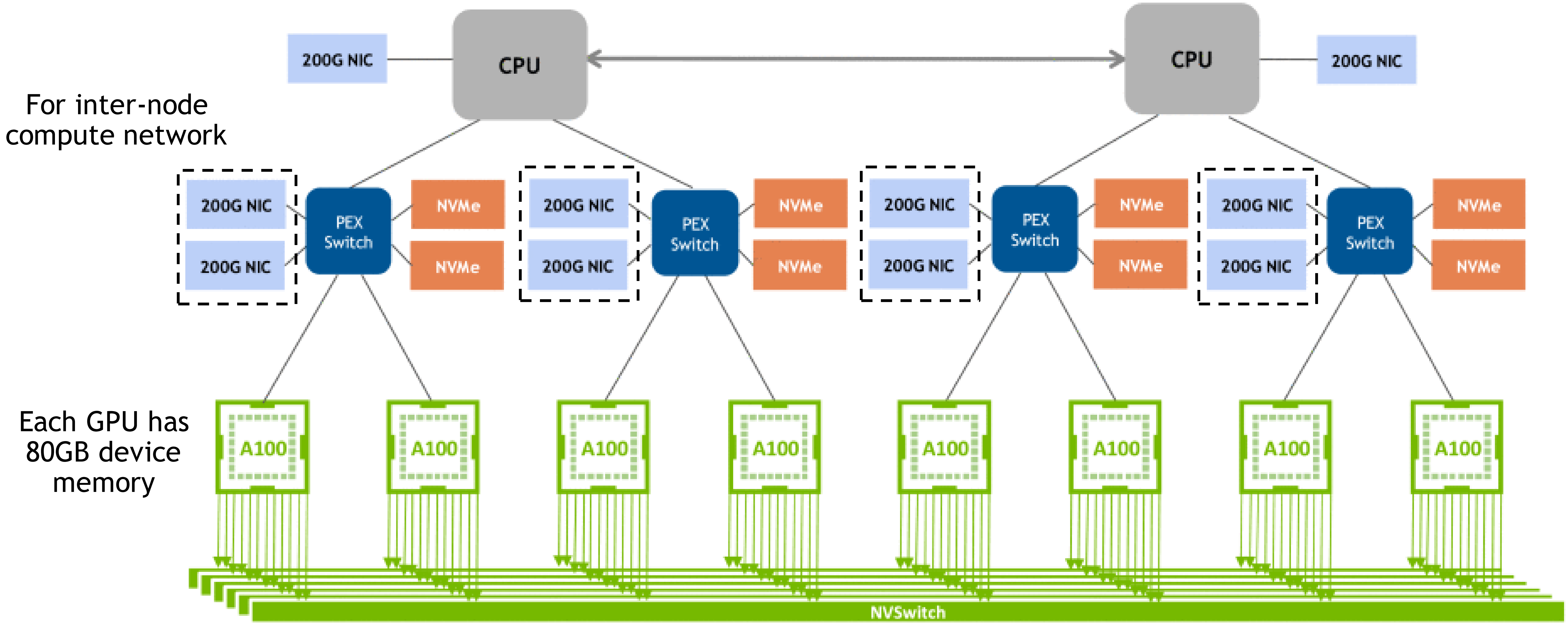
What if we need more than 80GB?

1) Spill to CPU memory

2) **Scale out to multiple GPUs** ---> This talk.

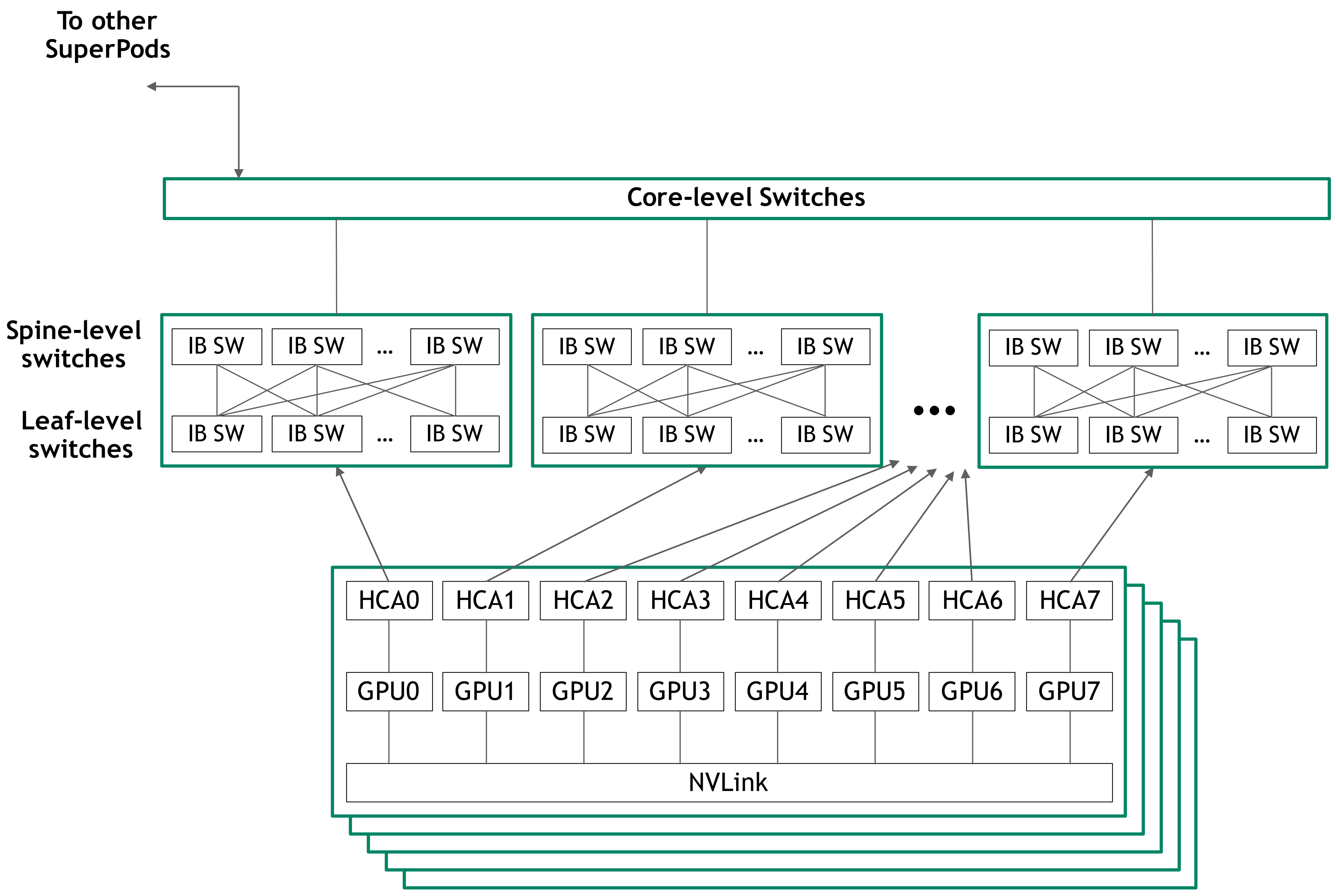
1. *Measured with open-source benchmark X-Mem: https://github.com/microsoft/X-Mem*

DGX A100 NODE



- Critical for performance to specify **GPU-NIC affinity**

CLUSTER TOPOLOGY

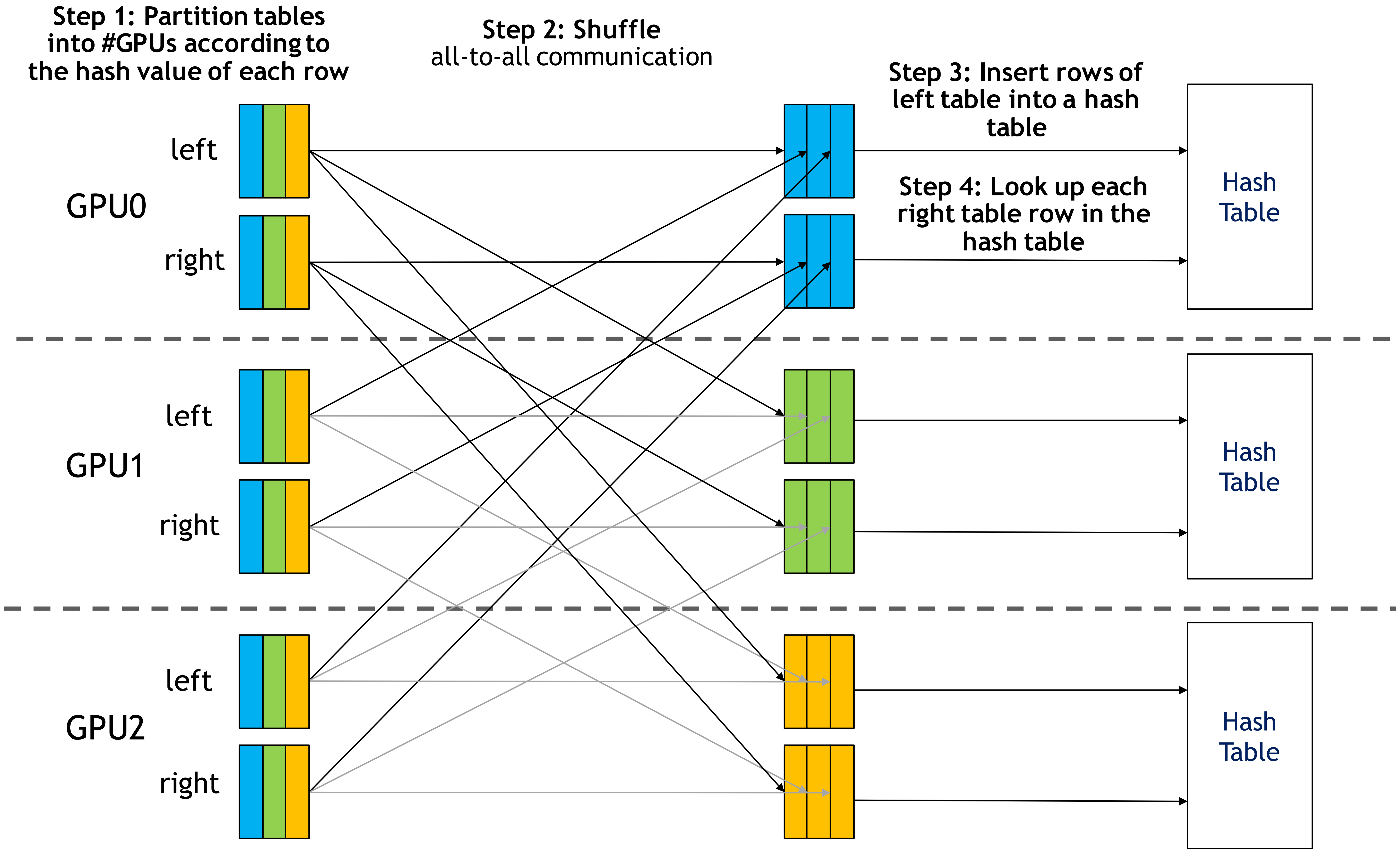


SuperPod (140x DGX A100 nodes)

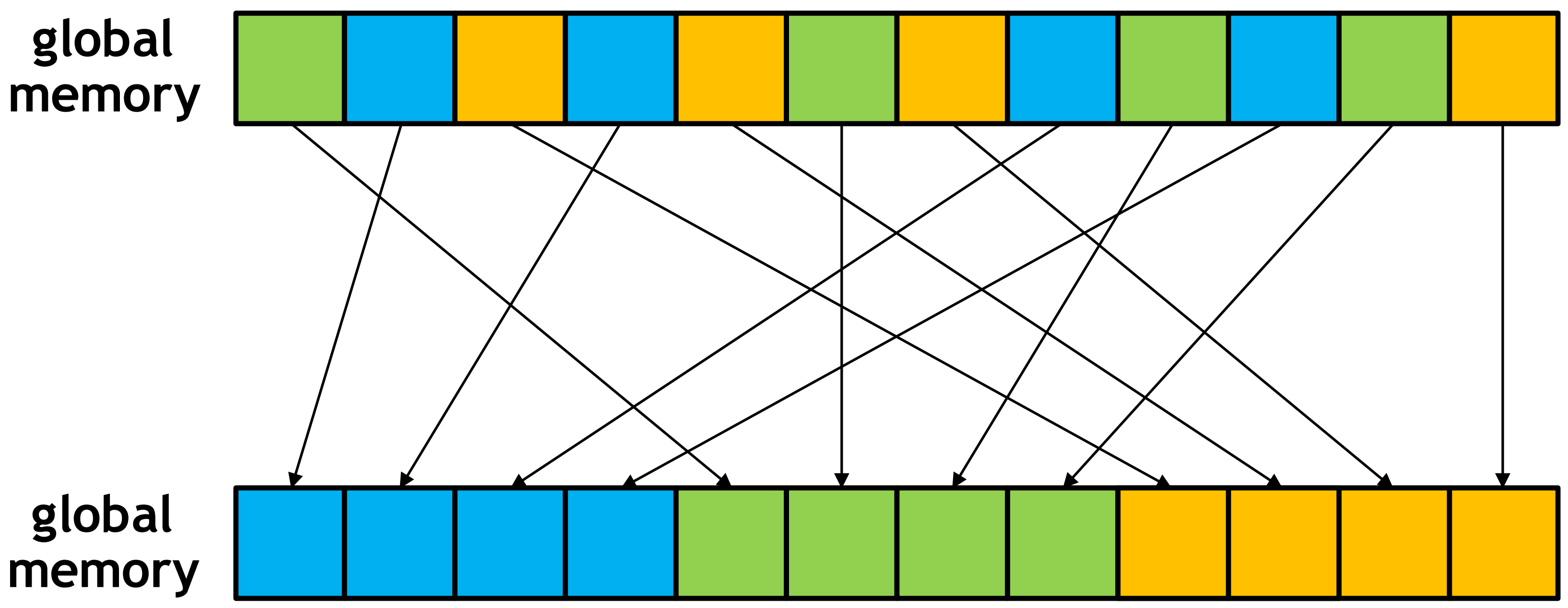
- Full fat-tree topology
- Three levels of switches: leaf level, spine level and core level
- Rail optimized: core-level switches are only used for cross-rail or cross-SuperPod traffic

GTC’21: NVIDIA Selene: Leadership-Class AI Supercomputing Infrastructure

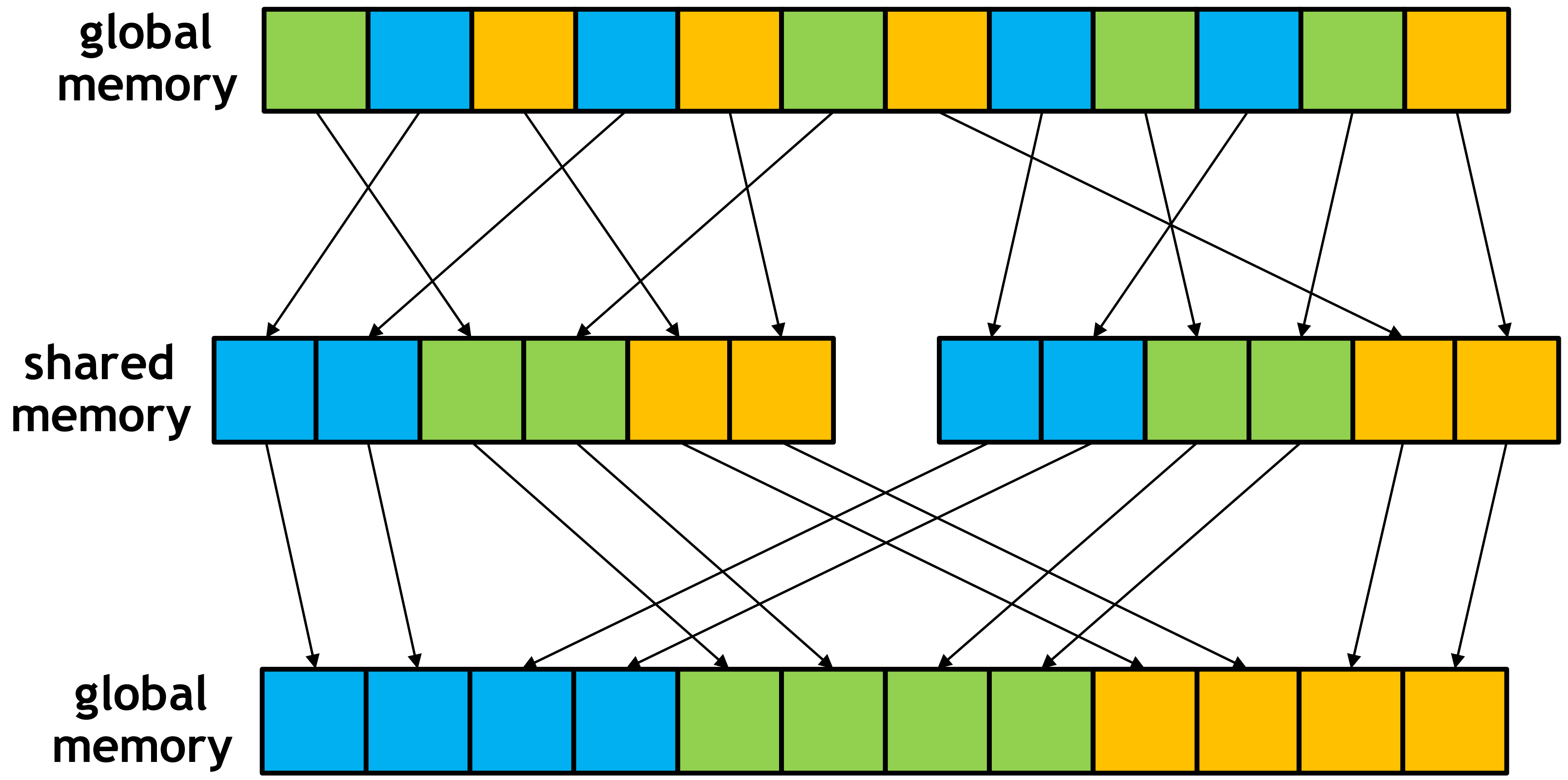
REPARTITIONED JOIN



HASH PARTITION



Naive implementation:
Dominated by the global memory
random accesses



Optimized implementation:
Random in shared memory,
sequential in global memory

LOCAL JOIN

- No-partitioned hash-based join
- Store references instead of values inside the hash table

Probe Table

l_orderkey
23
14
56
11
39
27
23

Hash Table

Hash value	Row idx
hash(27)	2
hash(23)	1
hash(11)	0
hash(29)	3

Build Table

o_orderkey	o_orderpriority
11	1
23	5
27	2
29	4

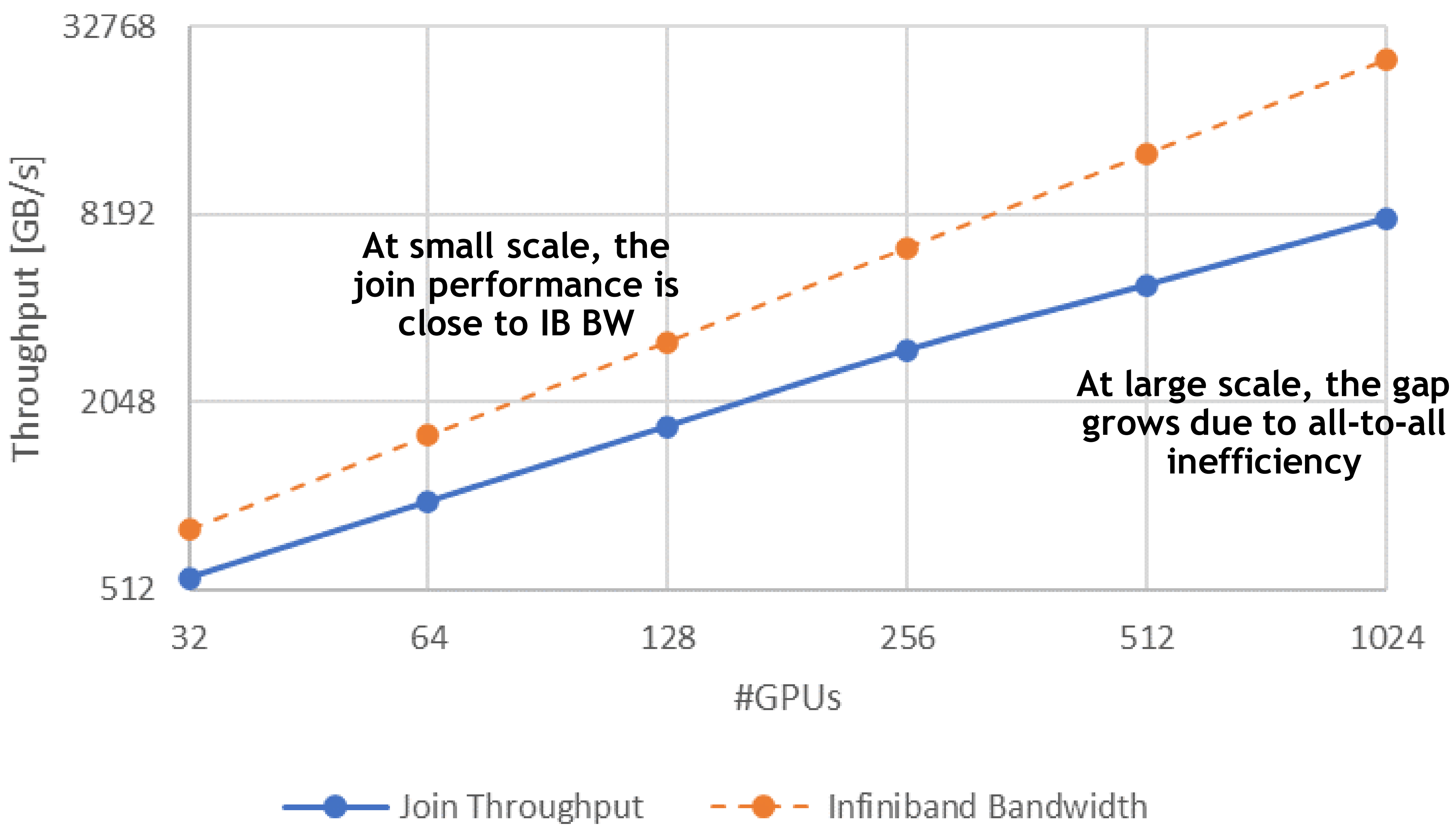
Key is not found

LOCAL JOIN

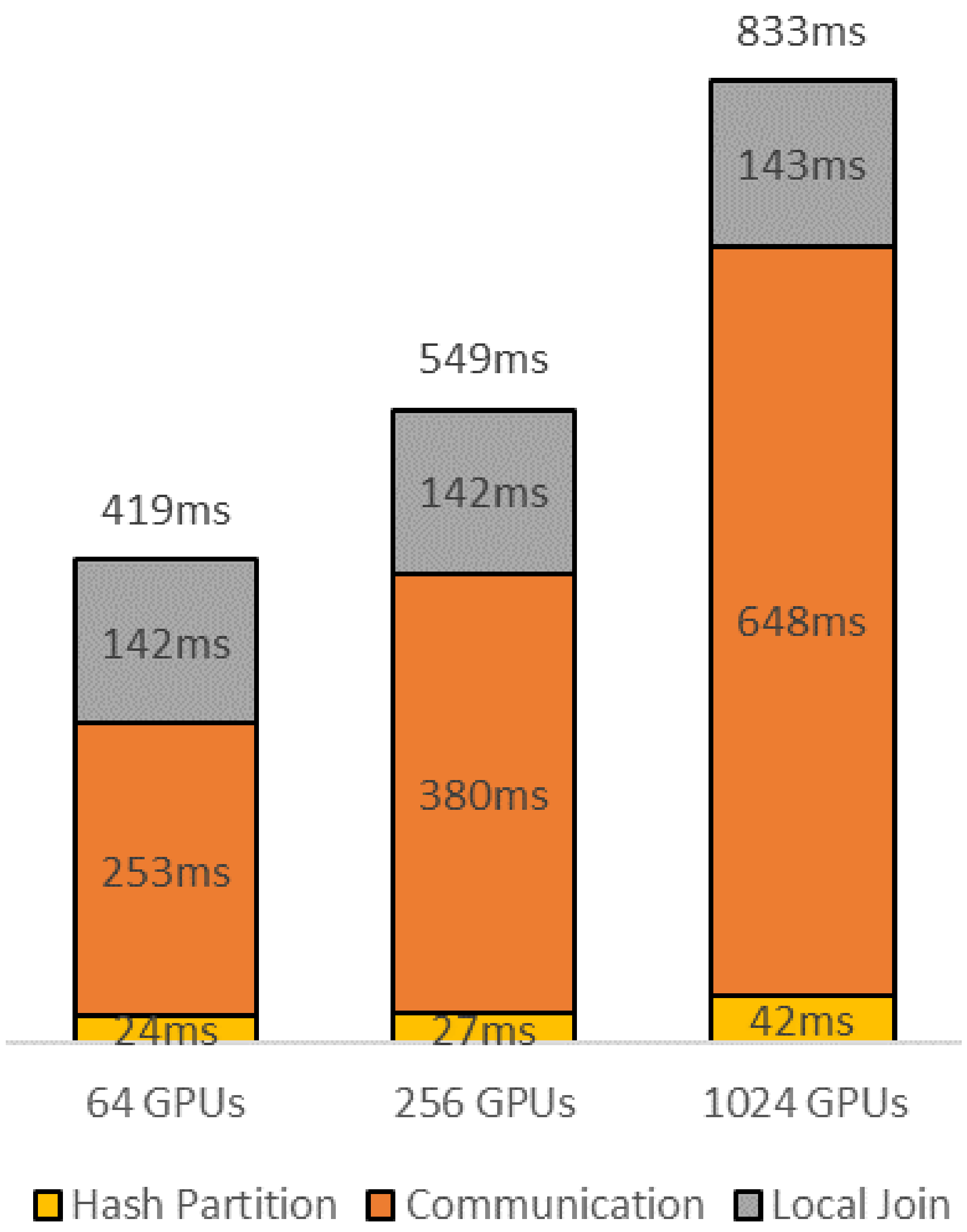
- No-partitioned join vs. partitioned join
  - Advantages: lower memory consumption
  - Disadvantages: higher cache miss rate
- Store references vs. store values
  - Advantages: can scale to large keys/payloads
  - Disadvantages: extra random reads during probing
- In principle repartitioned distributed join can use another single-GPU join as well (including partitioned join)

WEAK-SCALING PERFORMANCE ON TPC-H DATASET

Left table: **l_orderkey**
Right table: **o_orderkey** and **o_orderpriority**



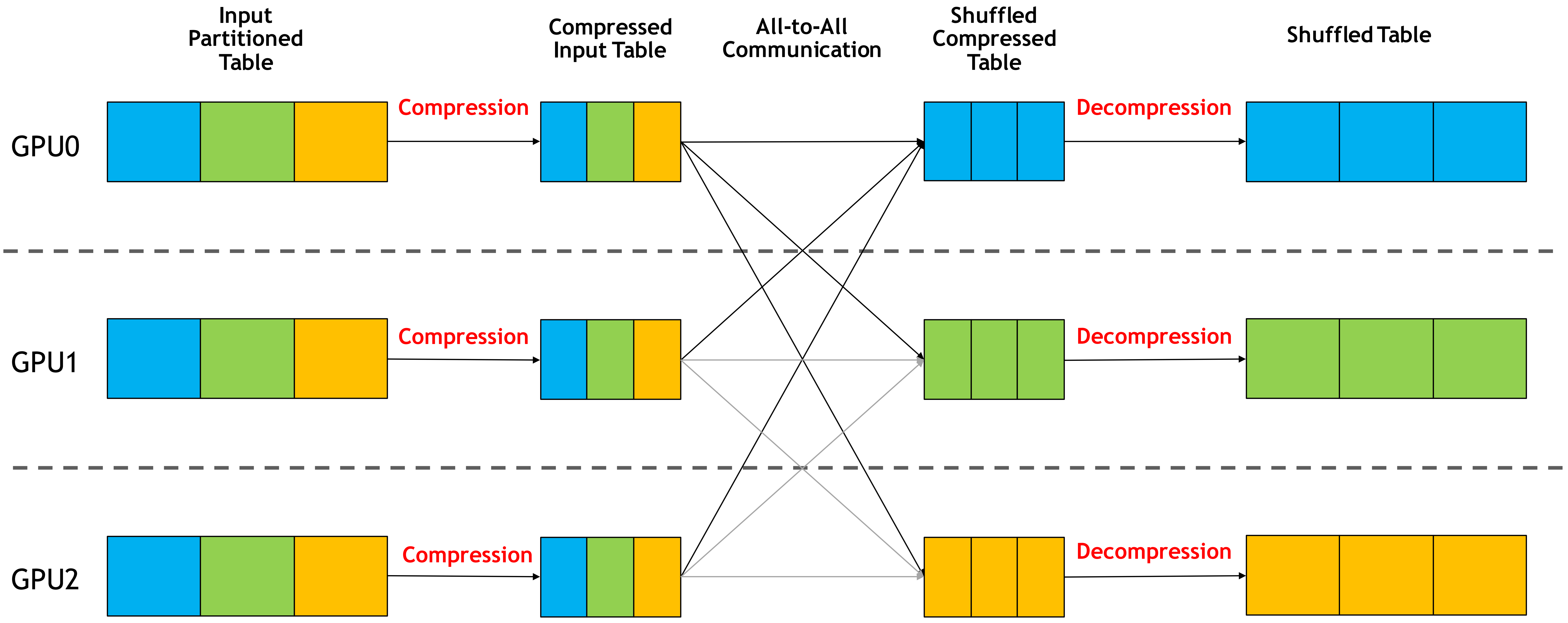
Time Breakdown



Dominated by communication time

Solution: use compression

COMMUNICATION WITH COMPRESSION



RUN LENGTH ENCODING (RLE)

- Idea: compress repeated values into (value, run length) pair

Input Sequence

3	9	9	4	4	4	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1

Value

3	9	4	0	1

Run Length

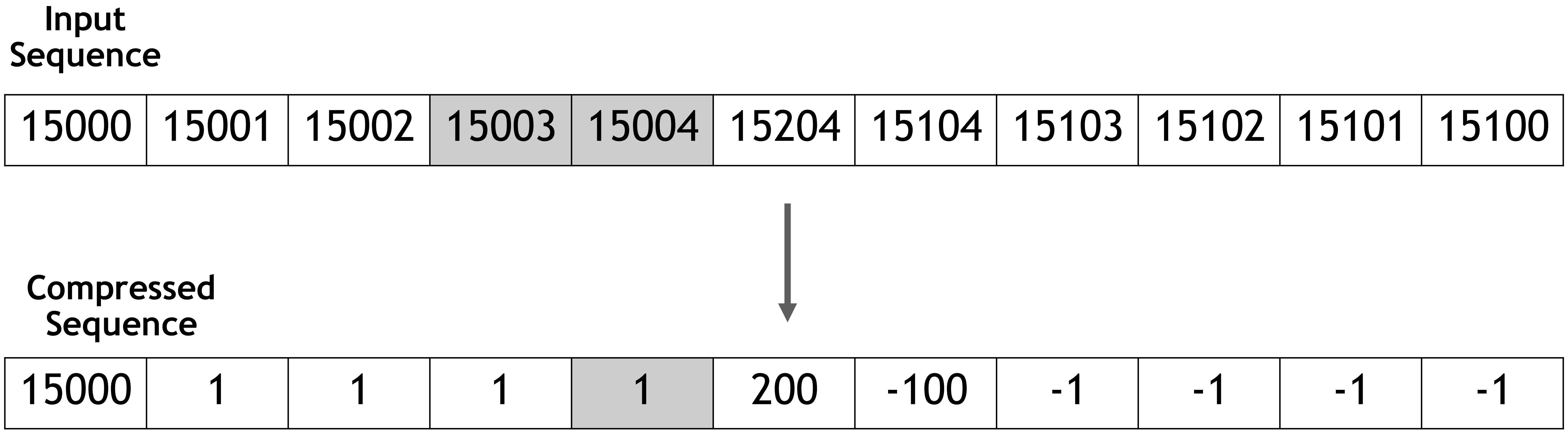
1	2	3	10	6

Compress 22 integers into 10!

In general, compression ratio is data dependent. Worst-case scenario, RLE can increase input by a factor of 2.

DELTA ENCODING

- Idea: compute the difference relative to the previous value



- Does not compress by itself
- But the output is easier to compress by RLE or bitpacking

FOR AND BIT-PACKING

- Idea: use smallest number of bits to represent a range

Input Sequence

15000	15001	15002	15003	15004	15204	15104	15103	15102	15101	15100

Compressed Sequence

0	1	2	3	4	204	104	103	102	101	100

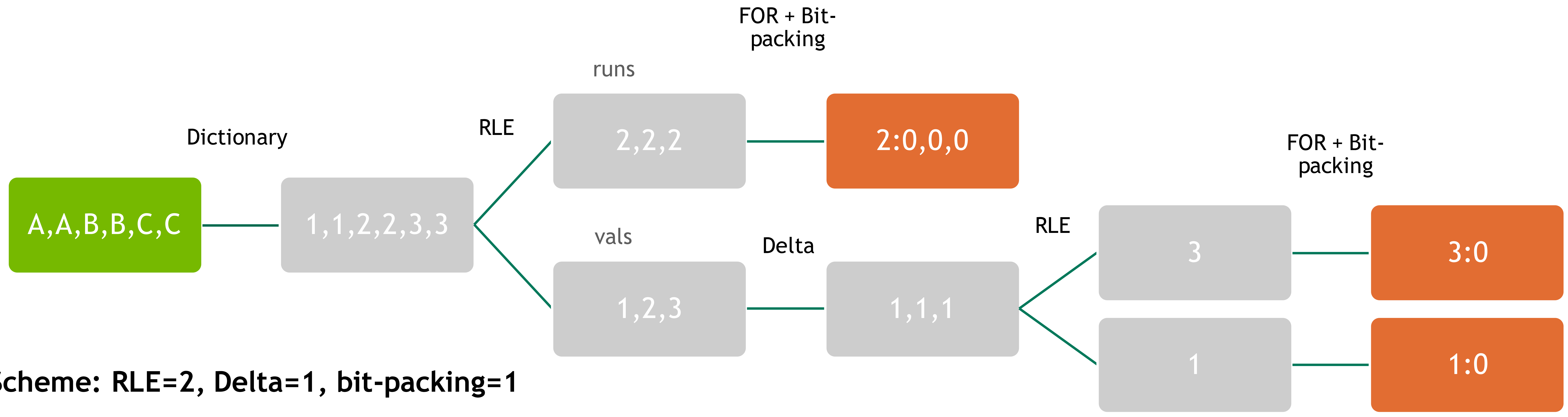
\+ 15000

Frame of reference (FOR)

all values < 256; use 8 bits

CASCADED COMPRESSOR

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



Scheme: RLE=2, Delta=1, bit-packing=1

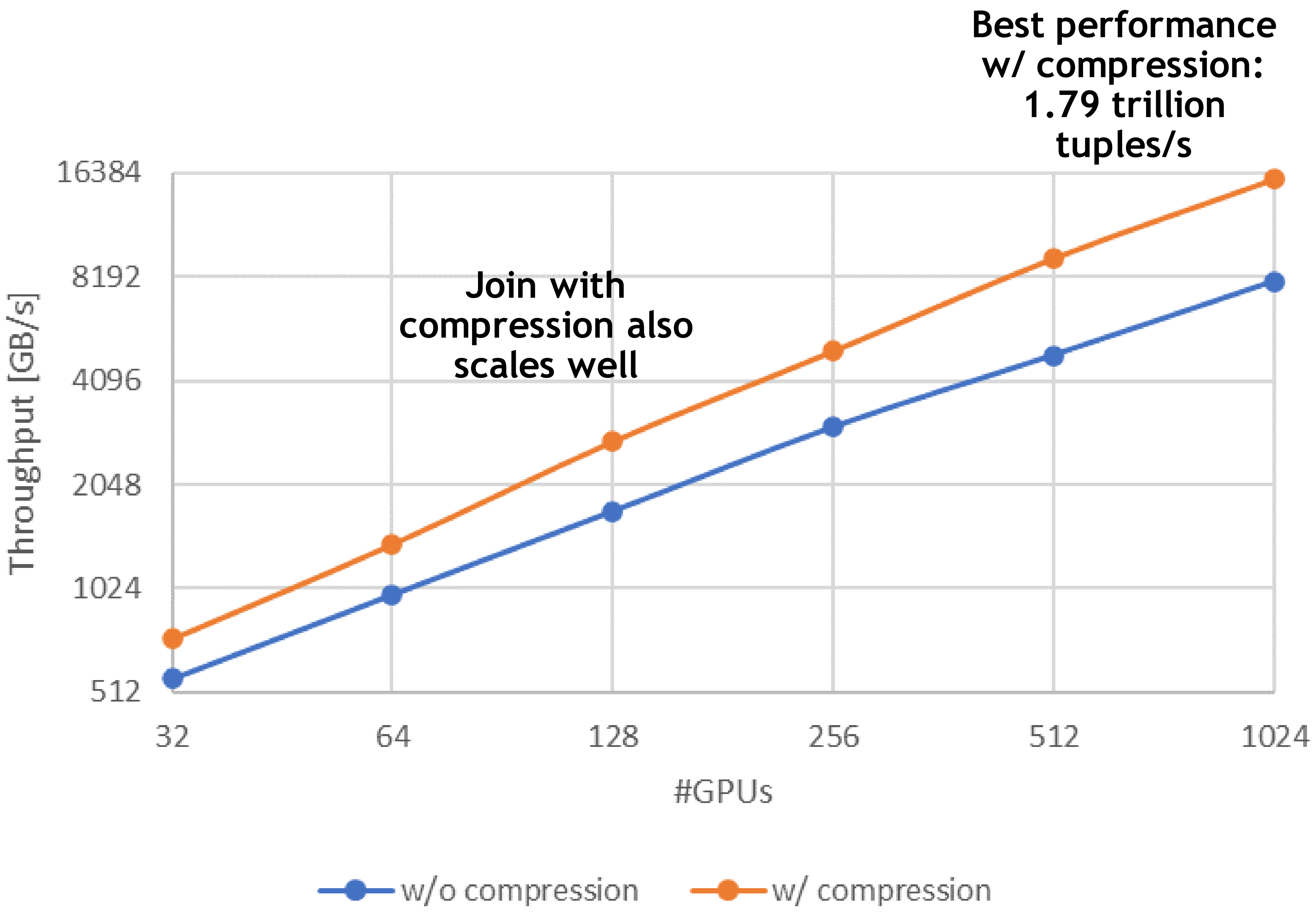
Uncompressed ⟷ Compressed

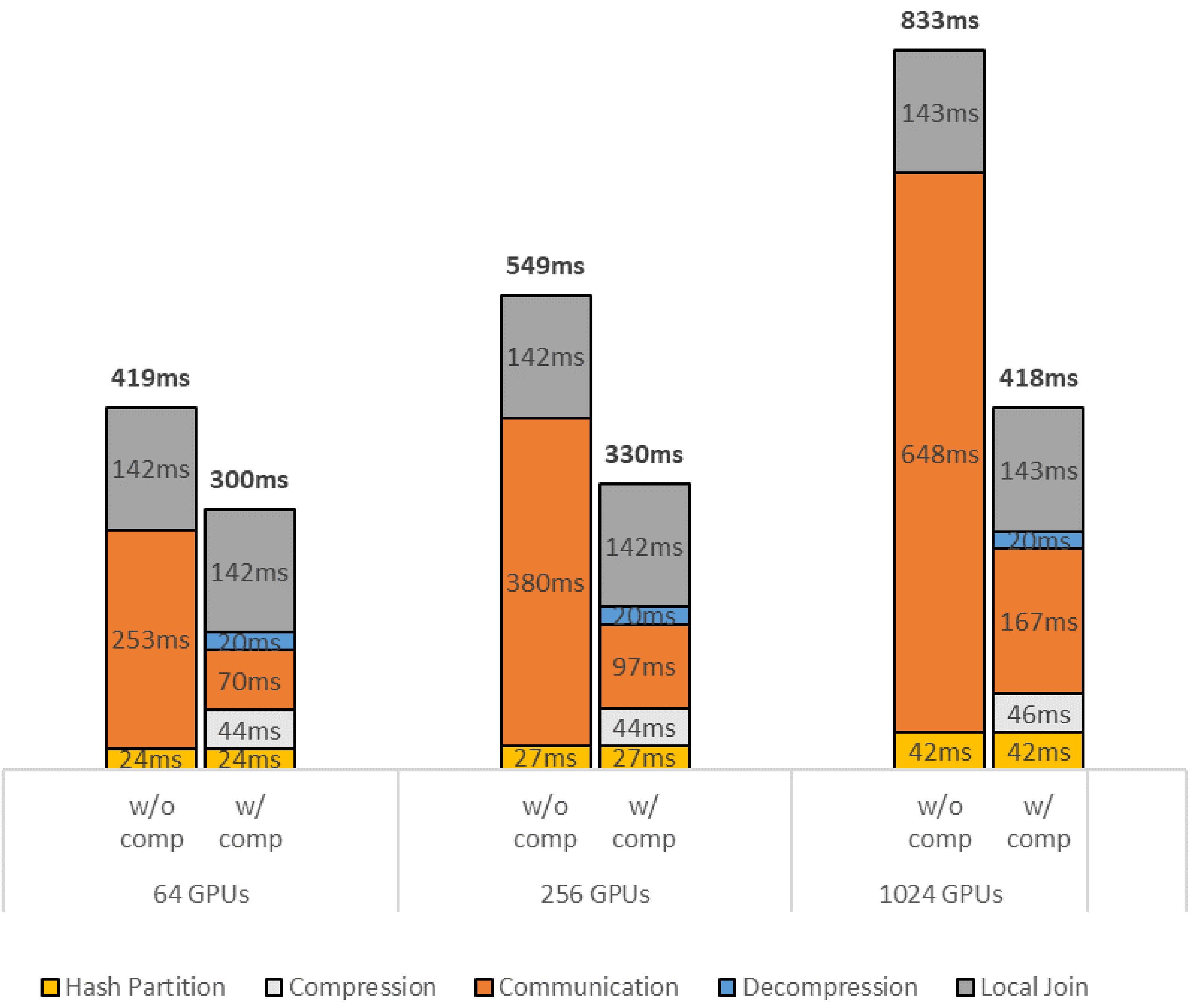
IMPLEMENTING CASCADED COMPRESSOR

- Important to **batch** and **fuse** all layers (i.e., all layers of all partitions are within a single kernel)
- Compression scheme (number of RLE/Delta/bitpacking layers) important for both ratio and throughput
- **Sample** and **benchmark** the input columns for choosing the best scheme (time not included for join).

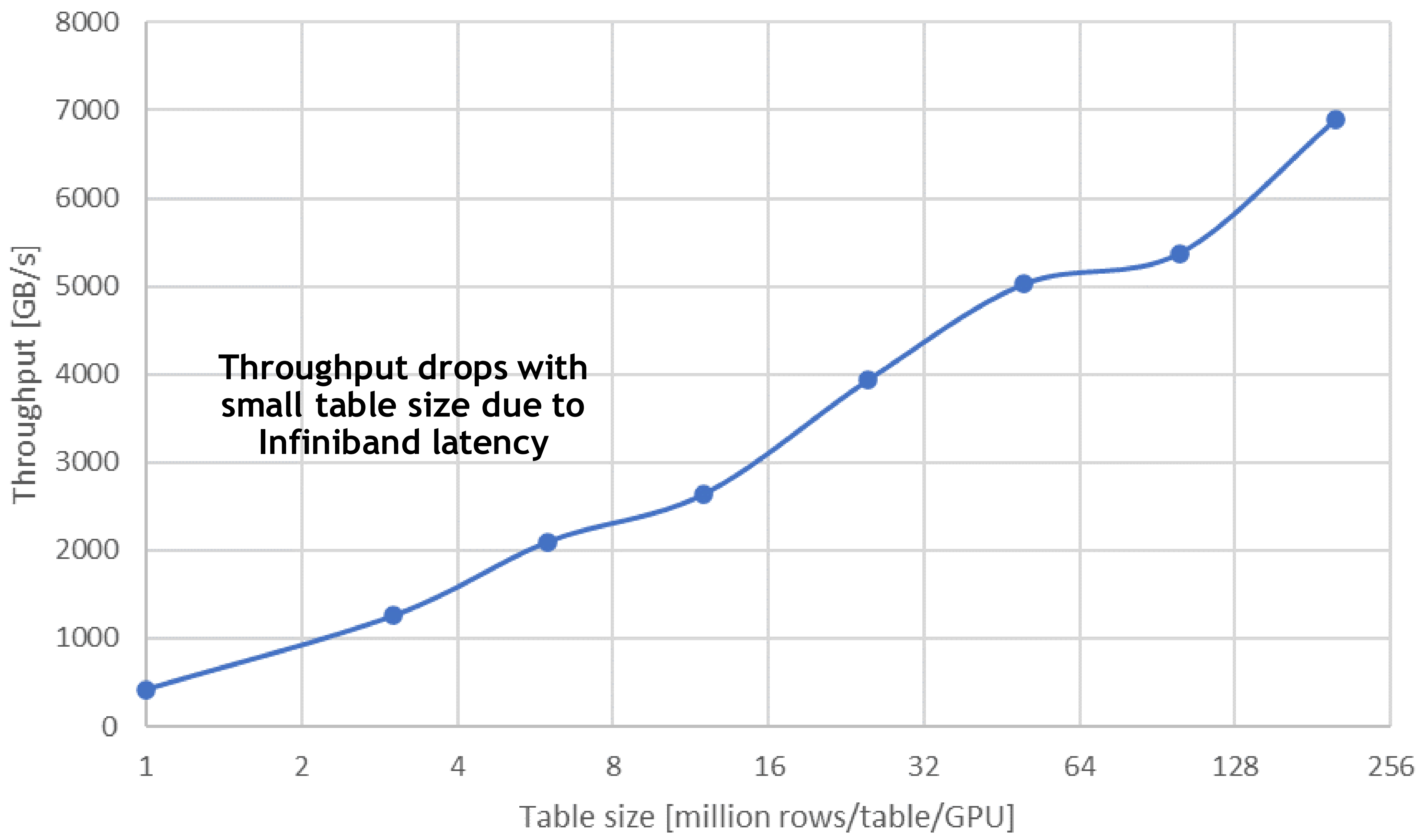
	RLE	Delta	bitpacking
O_ORDERKEY	2	1	True
O_ORDERPRIORITY	0	0	True
L_ORDERKEY	2	1	True

WEAK-SCALING PERFORMANCE WITH COMPRESSION





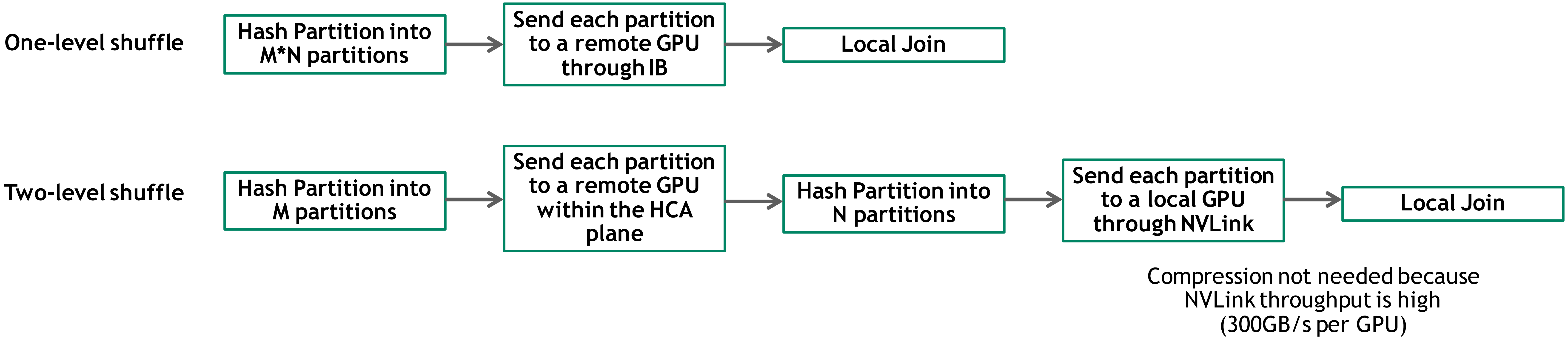
SMALL TABLES PERFORMANCE



- Fixed 512 GPUs
- Each table has two columns
  keys: random unique 8B integers
  payloads: 8B row index
- Selectivity = 0.3
- No compression

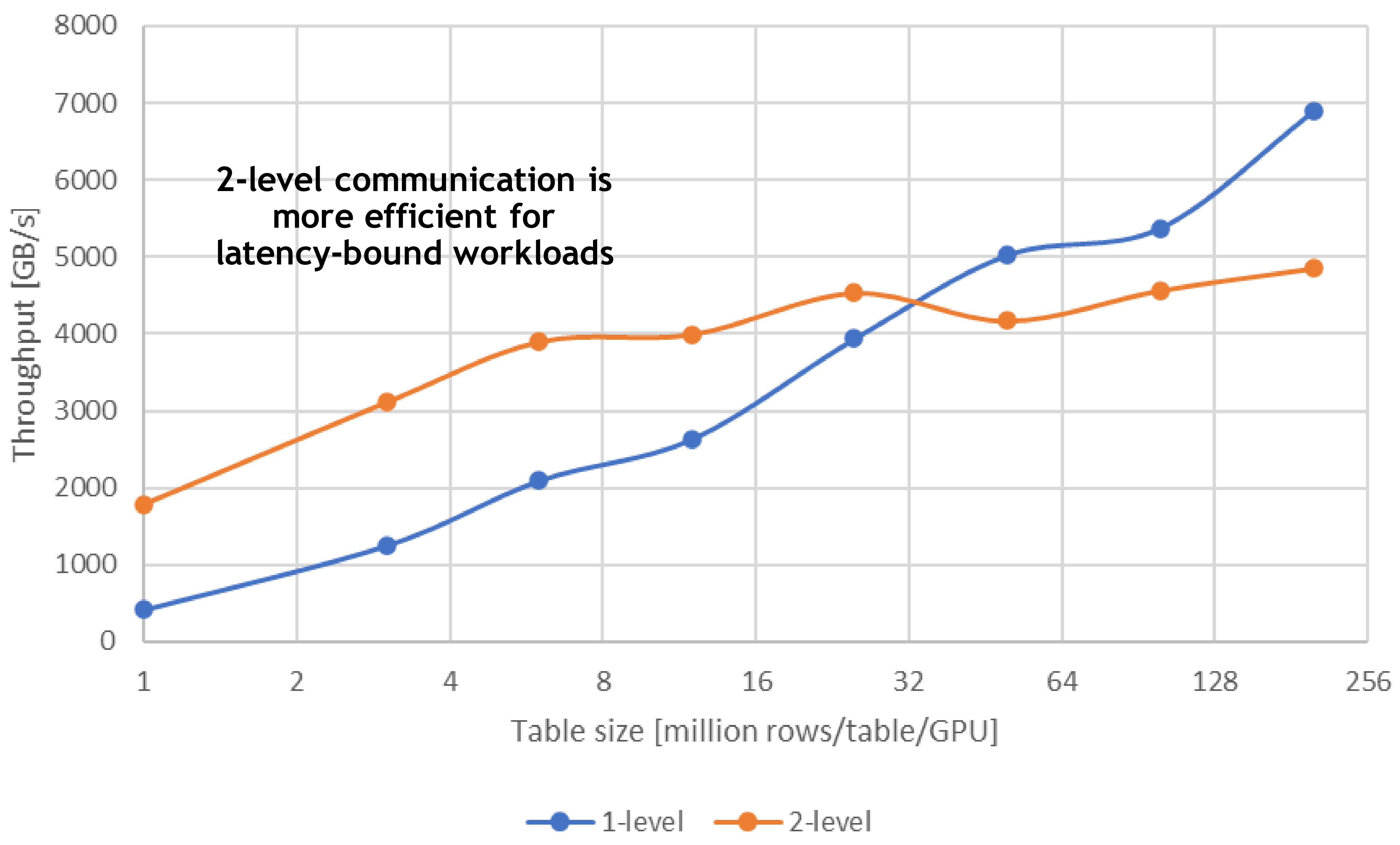
TWO-LEVEL COMMUNICATION

Suppose there are M nodes and N GPUs/node.



- Advantage: take advantage of the rail-optimized topology
- Advantage: less Infiniband messages
- Disadvantage: extra hash partition and communication stages
- Forward-looking: use shared-memory instead of message-passing within a node (help overlap and avoid copies)

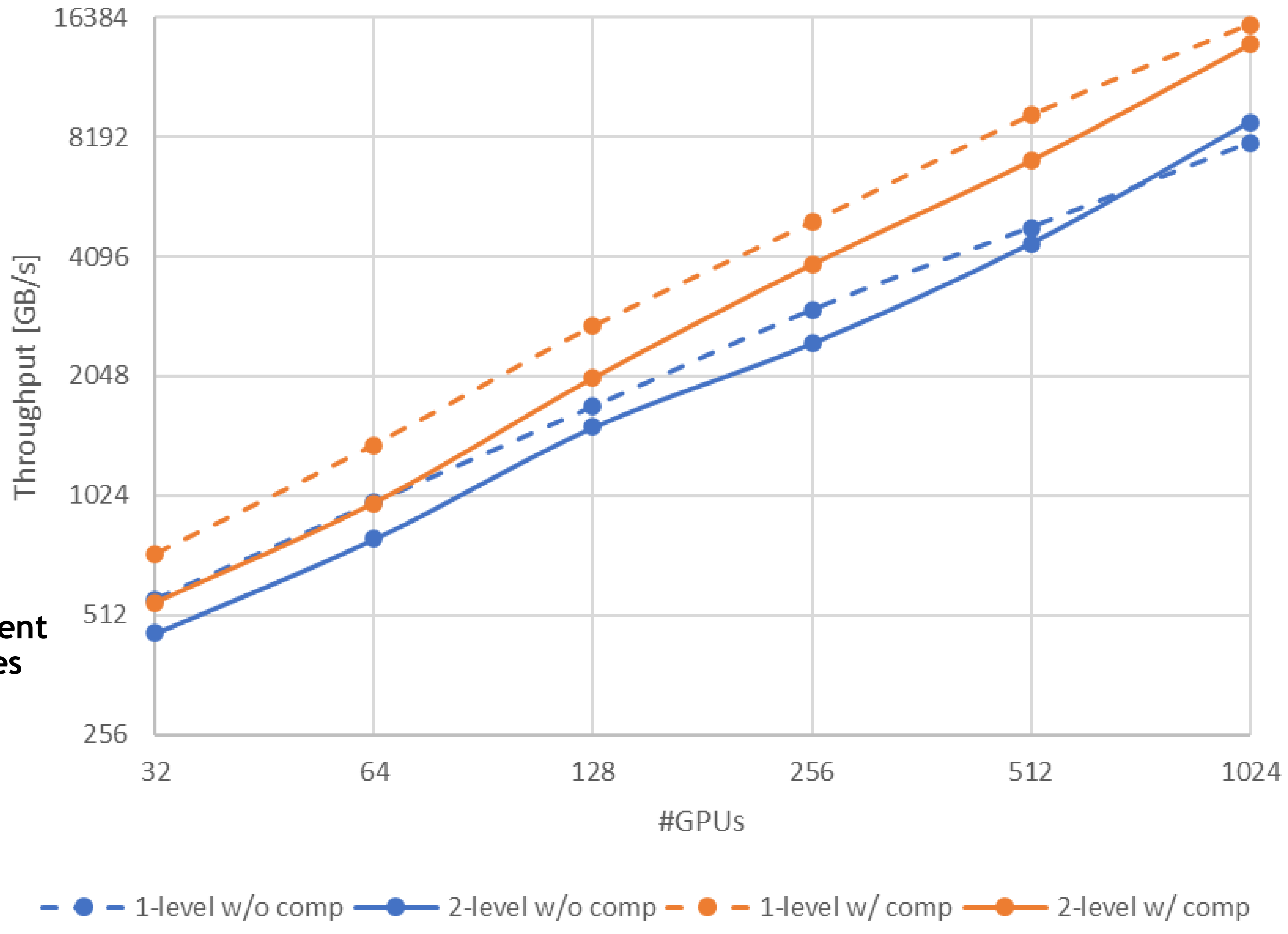
TWO-LEVEL COMMUNICATION – SMALL TABLES



- Fixed 512 GPUs
- Each table has two columns
  keys: random unique 8B integers
  payloads: 8B row index
- Selectivity = 0.3
- No compression
- 261632 IB messages for 1-level
  4032 IB messages for 2-level

NVIDIA

TWO-LEVEL COMMUNICATION - PERFORMANCE



Performance gap is closed when #GPUs is large due to less IB messages

At small scale, 2-level is less efficient due to extra stages

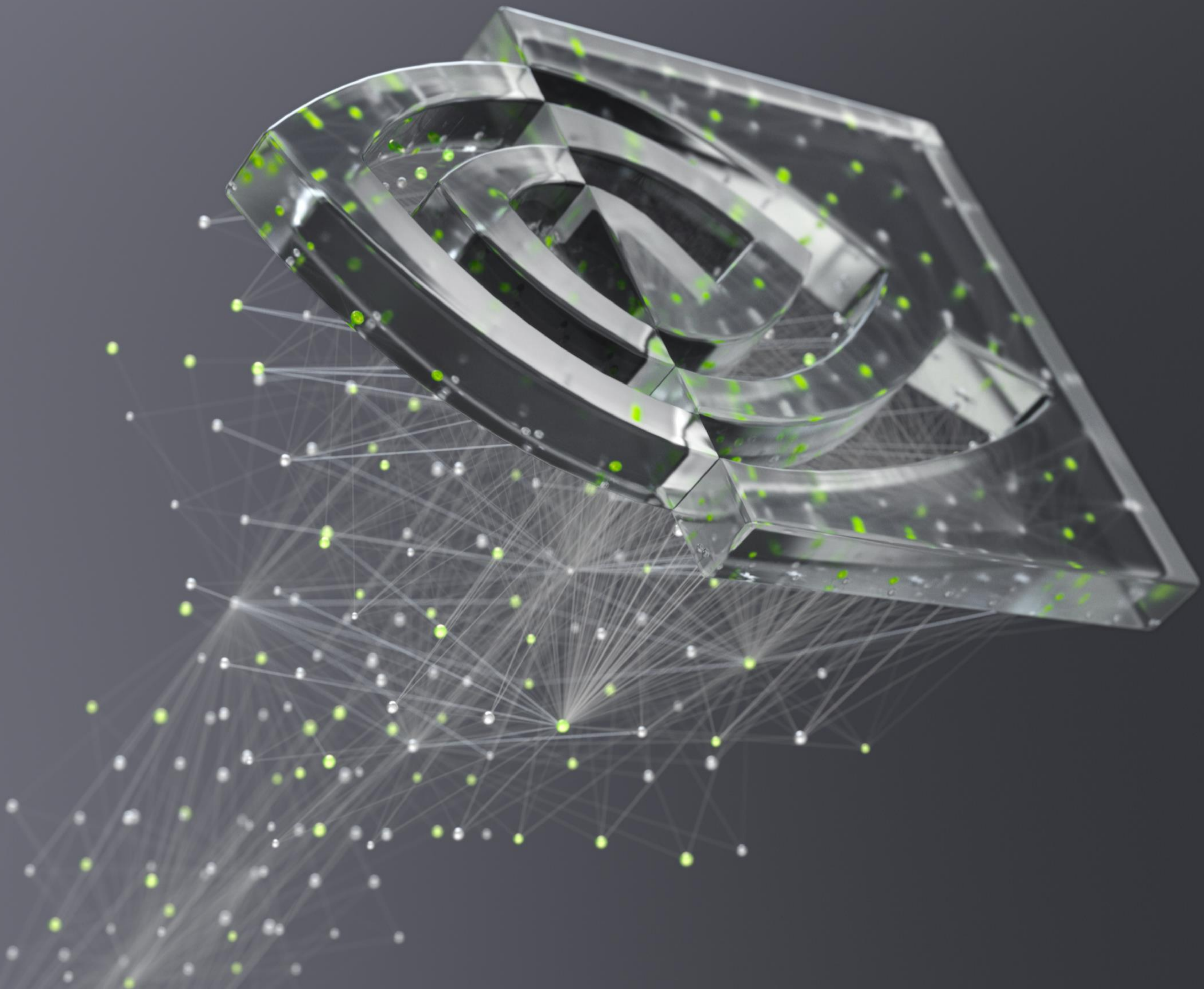
CONCLUSION

- Distributed **repartitioned hash-join** is scalable up to **1024 GPUs**
- Without compression, join time is dominated by the communication stage
- **Cascaded compression** is efficient and scalable. Improves performance by 1.77x on 1024 GPUs
- **Two-level communication** is helpful for latency-limited scenarios

Our code: *https://github.com/rapidsai/distributed-join*

FUTURE IMPROVEMENTS

- Computation and communication overlap
- Within a node, use shared-memory instead of message-passing
- Compression on strings
- Dataset with skewed keys
- Cache-friendly local join



NVIDIA