CMSC828G: Special Topics in Systems for Machine Learning

Introduction to HPC

Instructor: Yafan Huang

DEPARTMENT OF
COMPUTER SCIENCE

Introduce Yourself

- Name
- PhD/MS/Undergraduate Student
- What do you expect from this course?
- Something interesting about yourself

- TA: Shweta Bhardwaj (Email: shweta12@umd.edu)
- TA Office Hour: IRB 3119, Friday, 12:30 pm to 2:30 pm.

Getting Start with Zaratan

ssh username@login.zaratan.umd.edu

- Zaratan is managed by UMD DIT.
- 360 compute nodes each with 128 CPU cores and 512 GiB of memory.
- 20 nodes with 4 NVIDIA A100 GPUs (40 GB per GPU).
- 8 nodes with 4 NVIDIA H100 GPUs (80 GB per GPU).

Details of the Zaratan cluster: [1]

Description	Number of nodes	Processor	Cores/node	Mem/node (GiB)	Mem/core	/tmp size (TB)	GPUs/node
Standard compute	360	AMD Zen3	128	512	4	0.75	none
Large memory	6	AMD Zen3	128	2048	16	6	none
H100	8	Intel SapphireRapids	96	512	5.3	12	4 x H100 (80 GB)
A100	20	AMD Zen3	128	512	4	0.75	4 x A100

[2]

Accelerate llama.cpp on AMD EPYC: ZenDNN Delivers Up to 4.5x the Prompt Processing

Aug 18, 2026

[1] https://hpcc.umd.edu/home/clusters/zaratan/
[2] https://www.amd.com/en/developer/resources/technical-articles/2026/llama-cpp-on-amd-epyc.html

Getting Start with Nexus (nexusclass)

- Nexus is managed by UMD CS/UMIACS.
  - *nexusclass* partition.
- 39 nodes with 4 NVIDIA A4000 GPUs.
- 16 nodes with 8 NVIDIA A5000 GPUs.

ssh username@nexusclass.umiacs.umd.edu

GPU Features	NVIDIA RTX A4000
GPU Memory	16GB GDDR6 with error-correction code (ECC)
Display Ports	4x DisplayPort 1.4
Max Power Consumption	140 W
Graphics Bus	PCI Express Gen 4 x 16
Form Factor	4.4" (H) x 9.5" (L) Single Slot
Thermal	Active
VR Ready	Yes

[1]

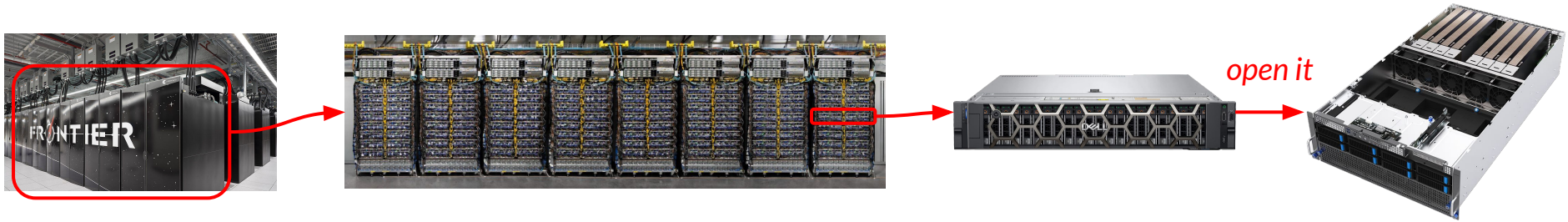
GPU Features	NVIDIA RTX A5000
GPU Memory	24GB GDDR6 with error-correction code (ECC)
Display Ports	4x DisplayPort 1.4*
Max Power Consumption	230 W
Graphics Bus	PCI Express Gen 4 x 16
Form Factor	4.4" (H) x 10.5" (L) Dual Slot
Thermal	Active
NVLink	2-way low profile (2-slot and 3-slot bridges)
Virtual GPU (vGPU) Software Support	NVIDIA Virtual PC (vPC) and Virtual Applications (vApps), NVIDIA RTX vWS, NVIDIA Virtual Compute Server
vGPU Profiles Supported	See the Virtual GPU Licensing Guide
VR Ready	Yes

[2]

[1] https://www.nvidia.com/en-us/products/workstations/rtx-a4000/
[2] https://www.nvidia.com/en-us/products/workstations/rtx-a5000/

High-Performance Computing

- High-performance computing (HPC) is the use of supercomputers and computer clusters to solve advanced problems. – *Definition from Wikipedia*.

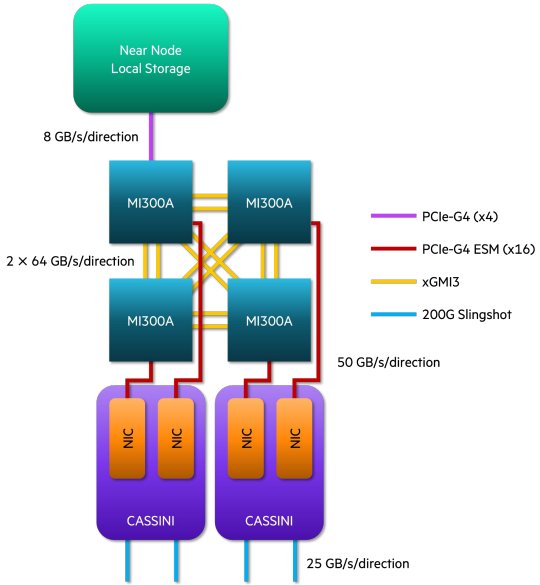


- Non-Uniform Memory Access (NUMA): a memory architecture where memory access time depends on the memory location relative to the processor.
- Most modern HPC systems are organized in this hierarchical way.

Top500 Supercomputer List

[1]

Rank	System	Cores	Rmax (PFlop/s)	Rpeak (PFlop/s)	Power (kW)
1	**LineShine** - LingKun, LX2 304C 1.55GHz, LingQi, Kylin OS, Shenzhen Cloud Computing Center Co., Ltd. National Supercomputing Centre in Shenzhen (NSCS) China	13,789,440	2,198.40	2,735.82	42,220
2	**El Capitan** - HPE Cray EX255a, AMD 4th Gen EPYC 24C 1.8GHz, AMD Instinct MI300A, Slingshot-11, TOSS, HPE DOE/NNSA/LLNL United States	11,340,000	1,809.00	2,821.10	29,685
3	**Frontier** - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE Cray OS, HPE DOE/SC/Oak Ridge National Laboratory United States	9,066,176	1,353.00	2,055.72	24,607
4	**Aurora** - HPE Cray EX - Intel Exascale Compute Blade, Xeon CPU Max 9470 52C 2.4GHz, Intel Data Center GPU Max, Slingshot-11, Intel DOE/SC/Argonne National Laboratory United States	9,264,128	1,012.00	1,980.01	38,698
5	**JUPITER Booster** - BullSequana XH3000, GH Superchip 72C 3GHz, NVIDIA GH200 Superchip, Quad-Rail NVIDIA InfiniBand NDR200, RedHat Enterprise Linux, Bull EuroHPC/FZJ Germany	4,801,344	1,000.00	1,226.28	15,794



Connection topology in one node of El Captain[2]

[1] https://top500.org/lists/top500/2026/06/
[2] https://hpc.llnl.gov/documentation/user-guides/using-el-capitan-systems/hardware-overview

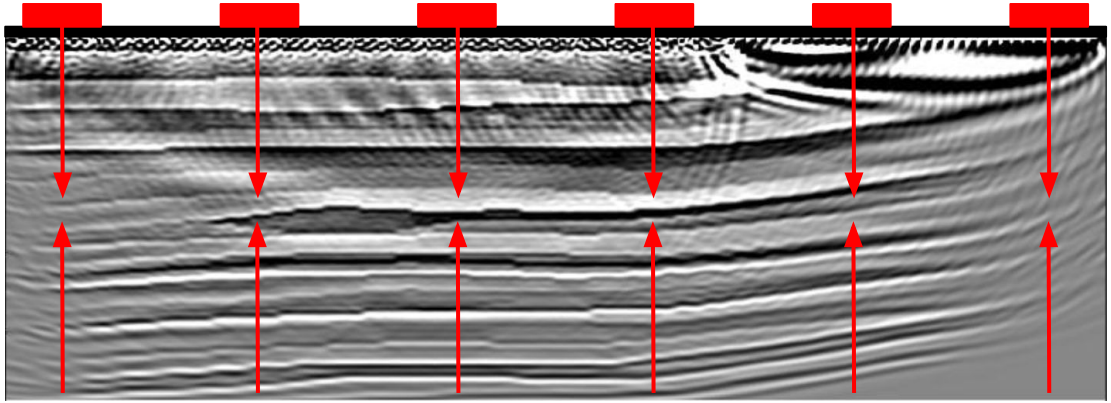
Case 1: Seismic Imaging

- Reverse time migration (RTM) recordes seismic waves to create a high-resolution image of the Earth's subsurface. – *Used a lot for finding oils or water resources.*
- RTM is highly computation-intensive, with each step designed to run in parallel.
- RTM is I/O-intensive, thousands of wave snapshots are sent into disk.

Supercomputer

Forward Waves

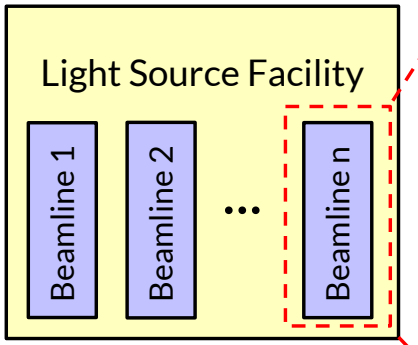
Backward Waves

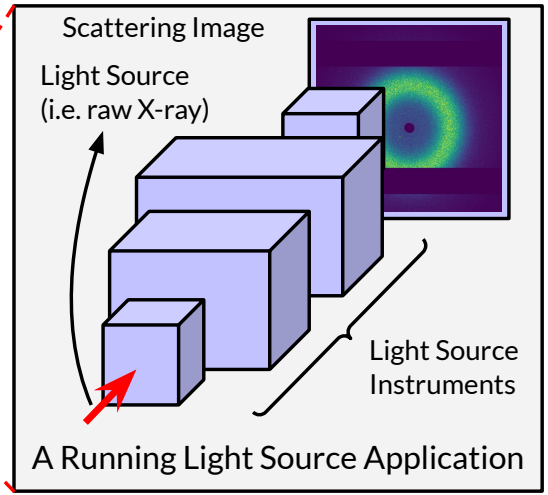


Case 2: Light Source

- Light source applications produce intense and tunable X-rays, enabling the study of both dynamic and static features of materials.
- Data generation speed for light source facility can be extremely high, requiring computation/storage from nearby supercomputers.

Light Source Facility at Argonne National Laboratory, IL





Case 3: Quantum Simulation

- Quantum simulation demands significant storage and computational resources.
- Storage constraints: only 48-qubit quantum circuit can fully occupy Frontier memory systems (4.6 Peta Bytes of DDR4 memory).
- Computation constraints: every small step in quantum simulation (a.k.a. state-vector simulation) is a large-scale vector matrix multiplication.

Frontier Supercomputer

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix} = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1n} \\ x_{21} & x_{22} & \cdots & x_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{m1} & x_{m2} & \cdots & x_{mn} \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix}$$

Example of Vector Matrix Multiplication

Case 4: LLM Training

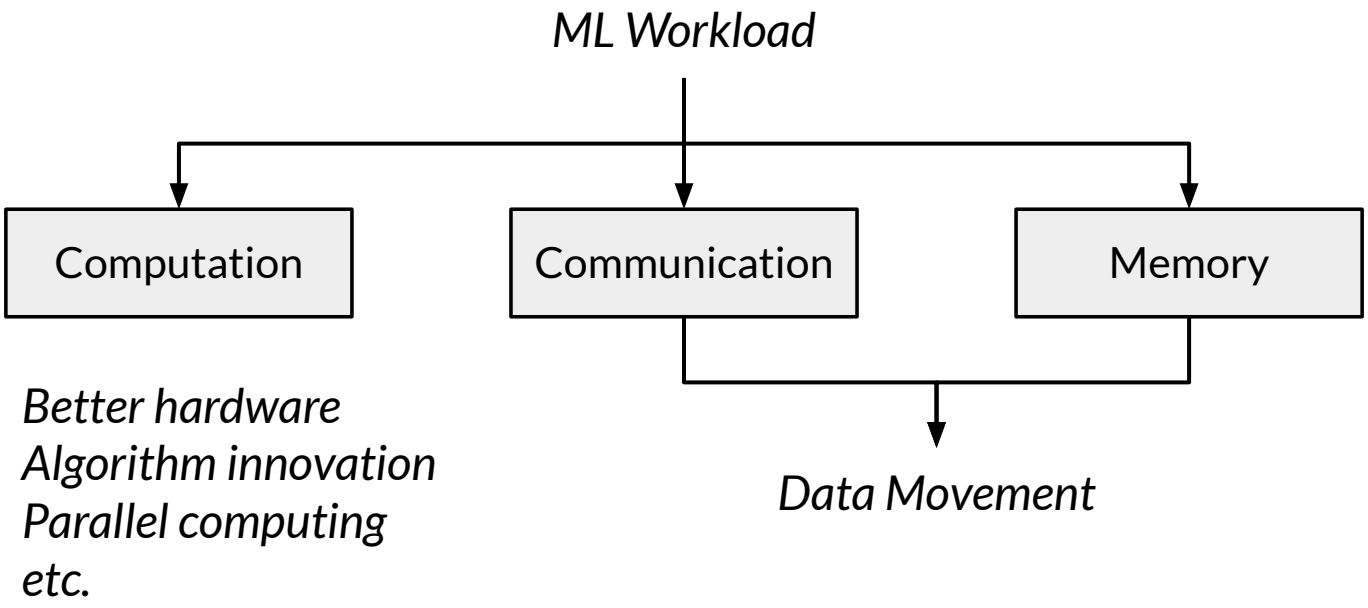
- LLM training has become a major HPC workload.
- GPT-4 Training Efforts: 25,000 GPUs, 90~100 Days.

Microsoft Azure

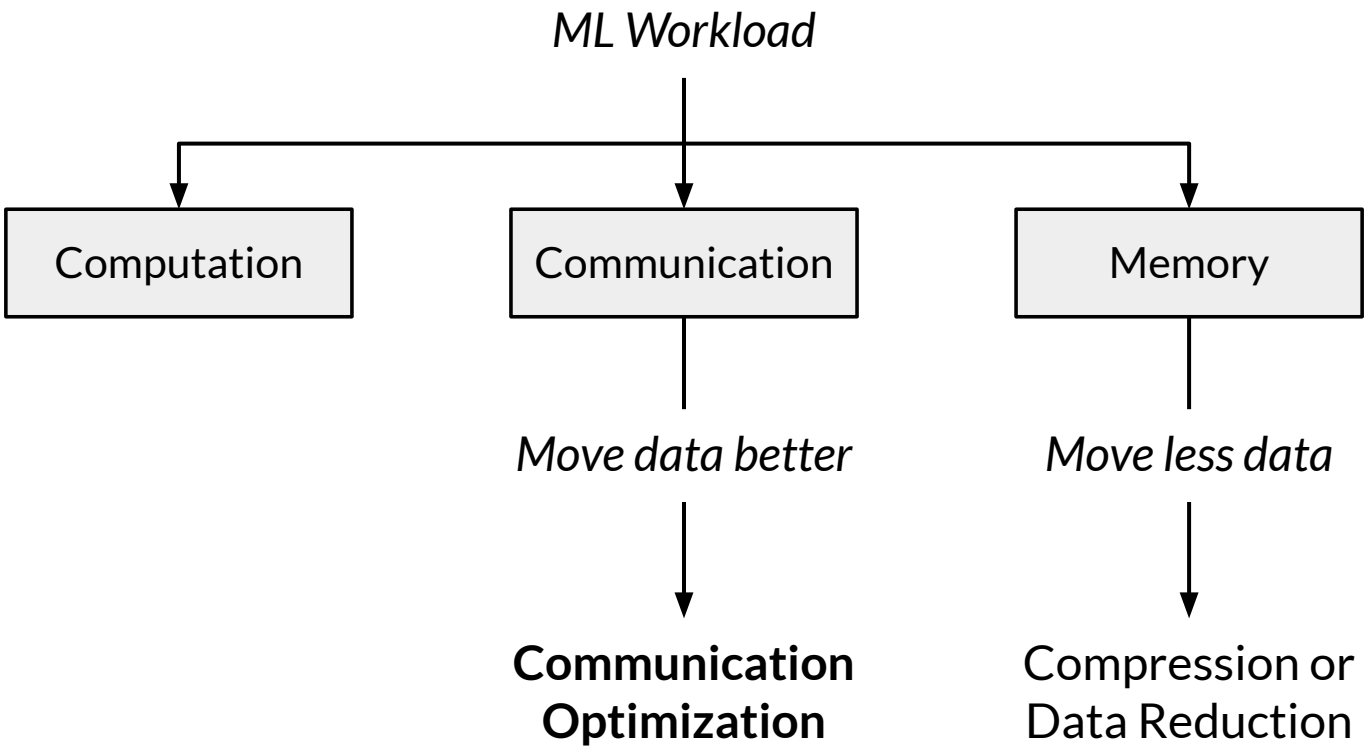
- What about LLM inference? Is it HPC?

[1] Everything we know about GPT-4: https://klu.ai/blog/gpt-4-llm

From HPC to ML Systems: A System View

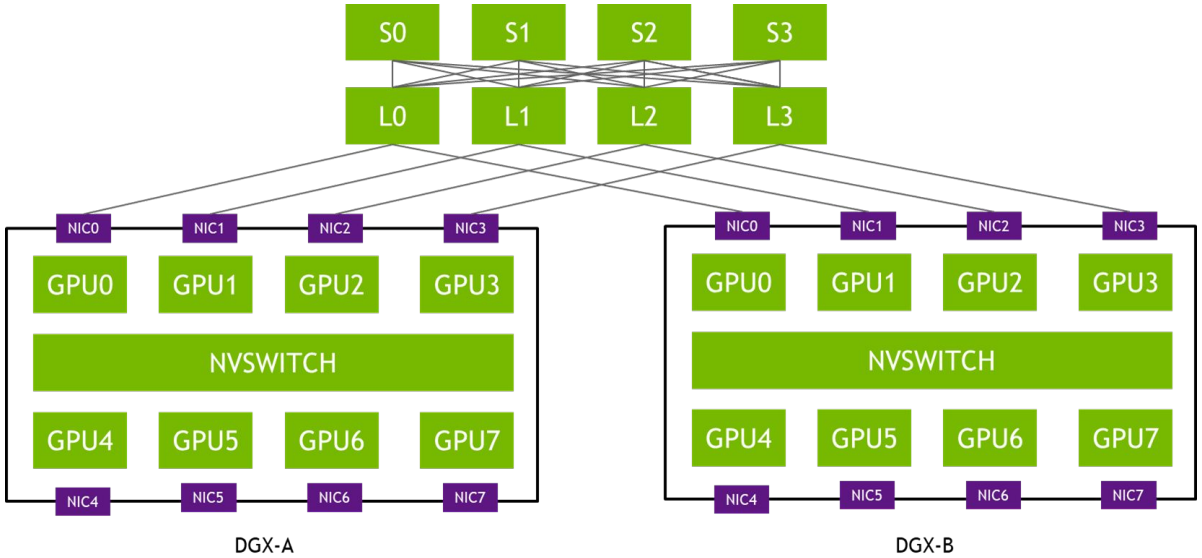


From HPC to ML Systems: A System View



Network Communication in NVIDIA

- Inter-node: S -> Spine; L -> Leaf; NIC -> network interface card.
- Intra-node: NVSwitch, NVLink, PCI-e.



NVIDIA GPU Cluster Network Topology[1]

[1] https://developer.nvidia.com/blog/doubling-all2all-performance-with-nvidia-collective-communication-library-2-12/

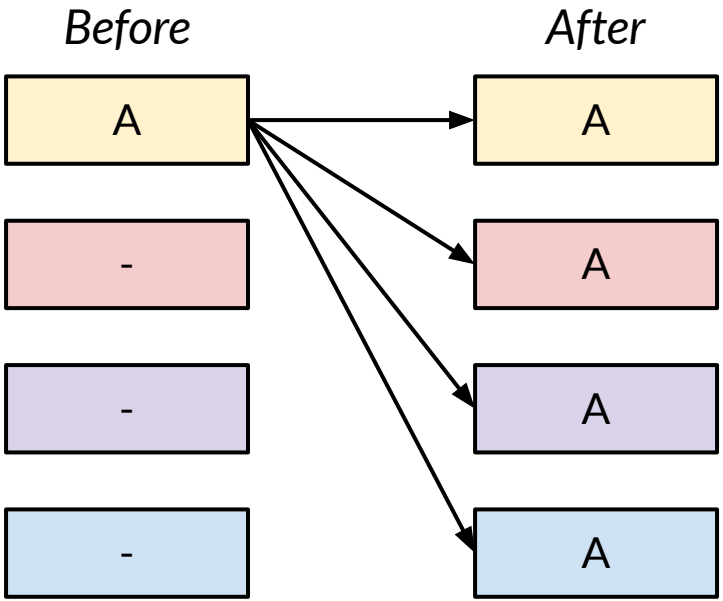
Communication Vocabulary

- Process: an independent running instance of a program.
- Rank: the unique ID number assigned to the process in a communication group.
- Messege/buffer: data that is communicated across each rank.
- Point-to-point: communication between a pair of process.
- Collectives: communication involve a group of processes.

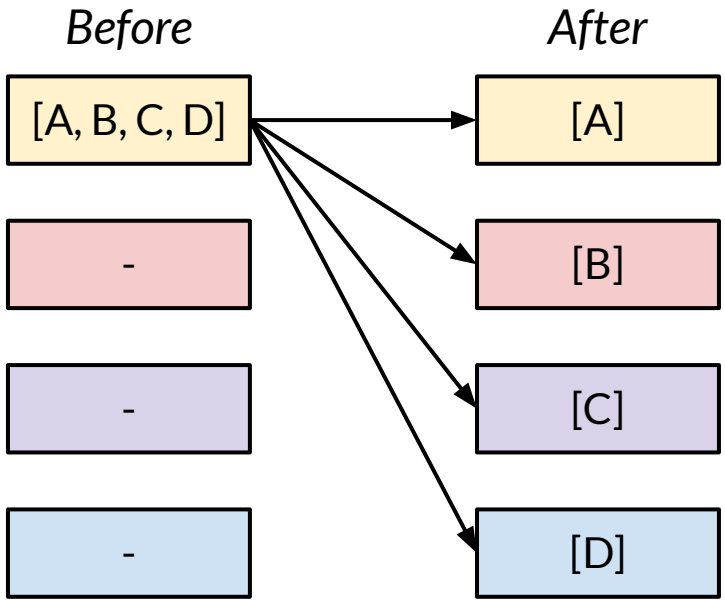
- Communication libraries: MPI, NCCL, RCCL, PyTorch Distributed, etc.

Collective Operation: One-to-Many

Rank 0 | Rank 1 | Rank 2 | Rank 3



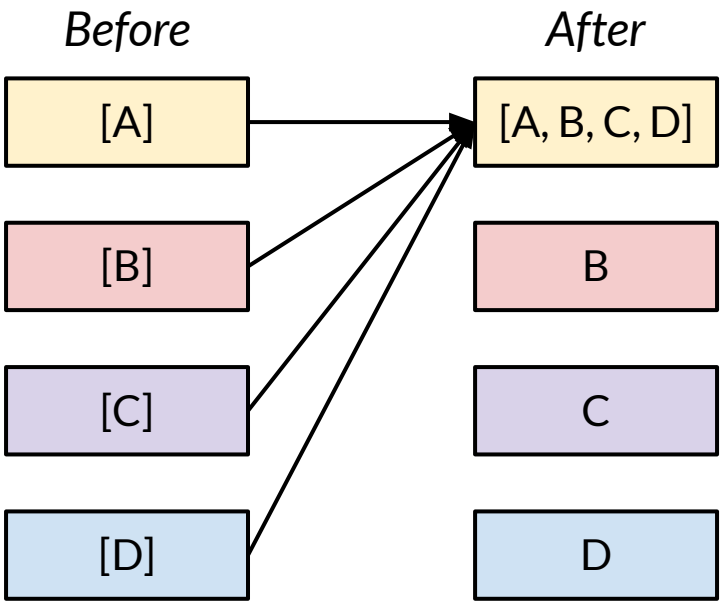
Broadcast



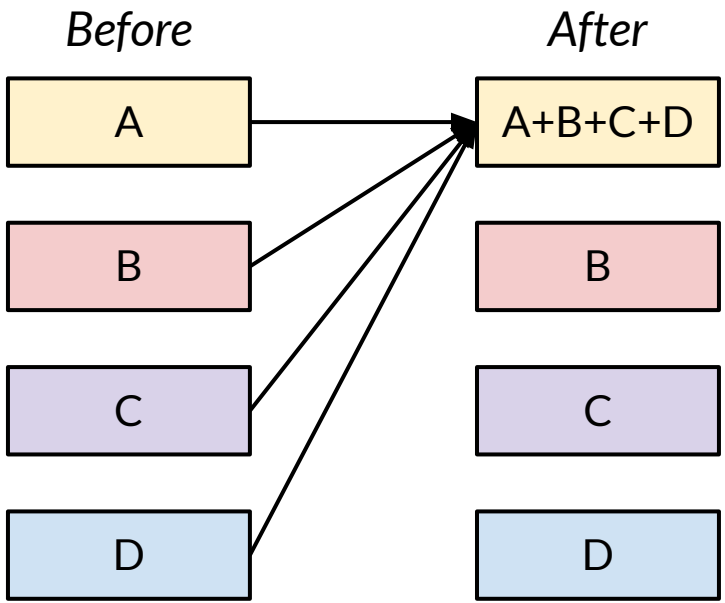
Scatter

Collective Operation: Many-to-One

Rank 0 | Rank 1 | Rank 2 | Rank 3

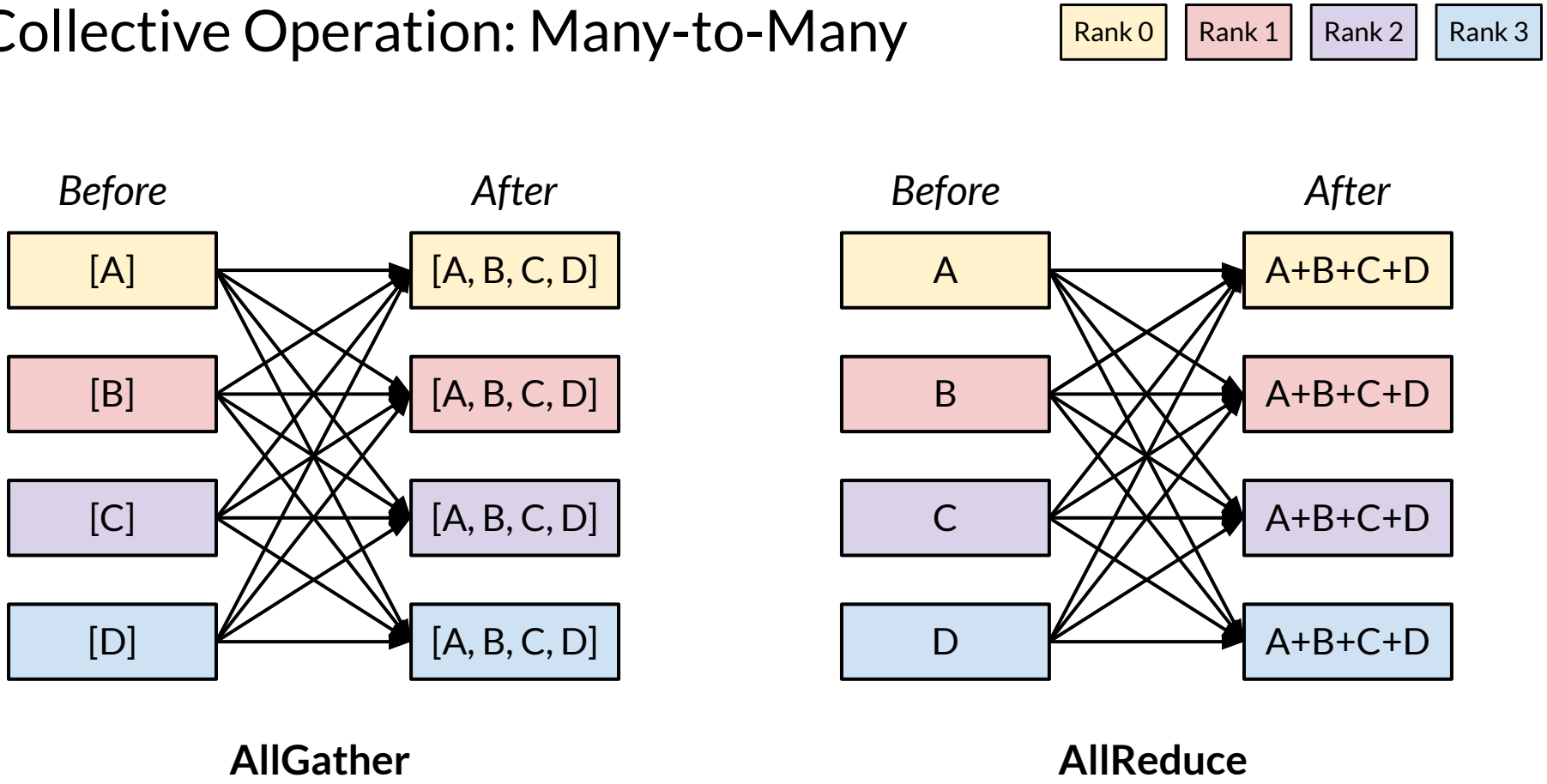


Gather

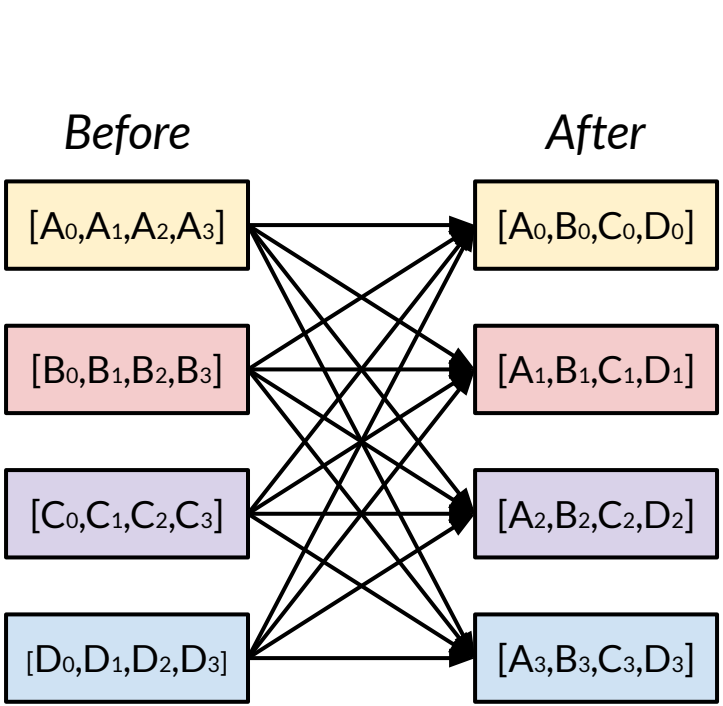


Reduce

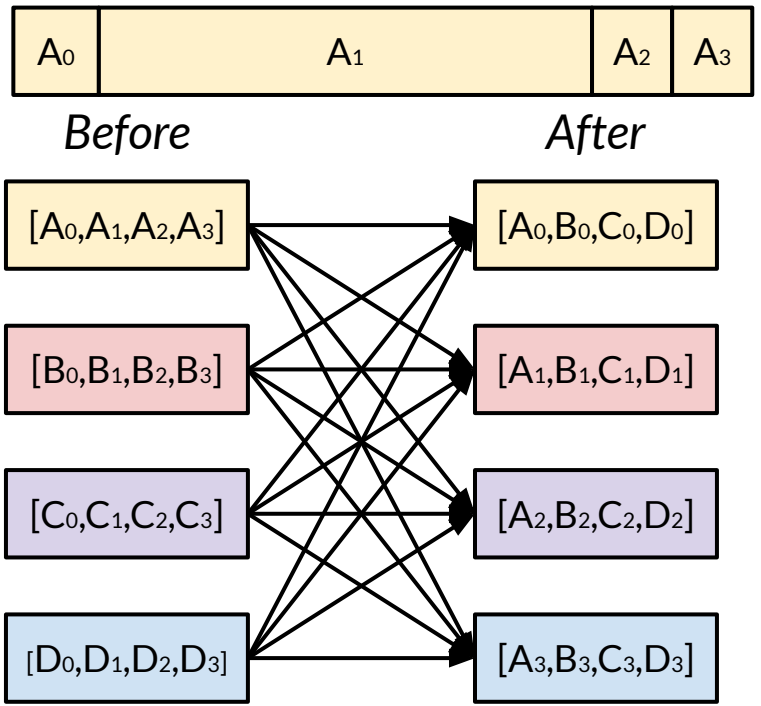
Collective Operation: Many-to-Many



Collective Operation: Many-to-Many



All-to-All

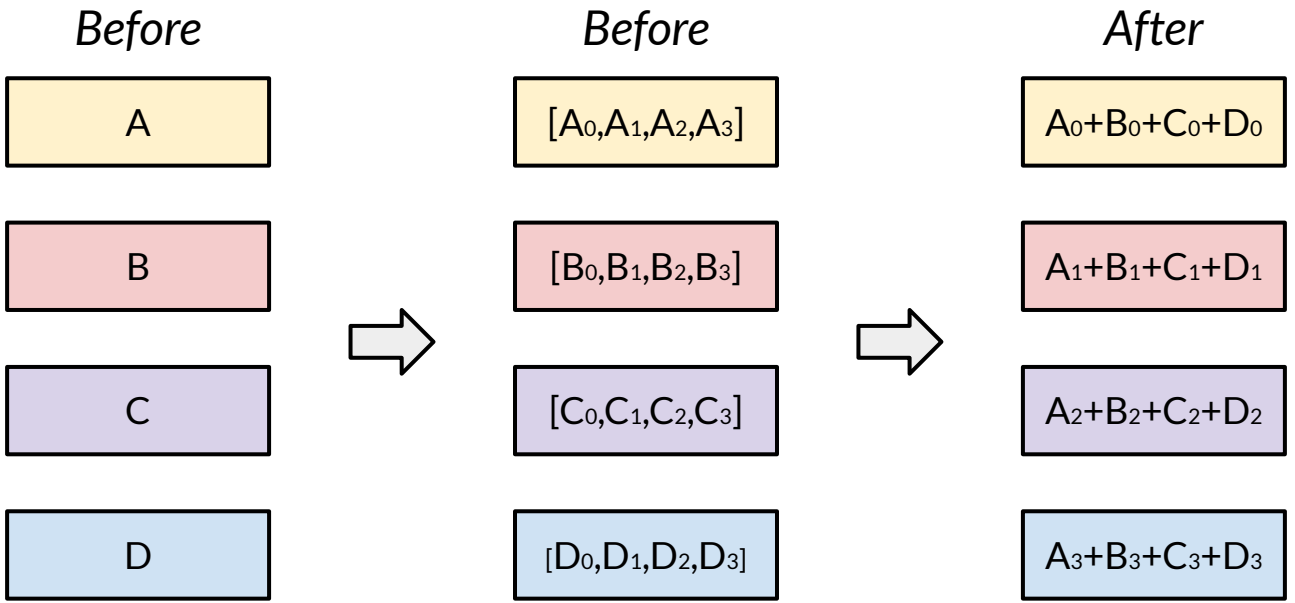


All-to-Allv

variable size/counts

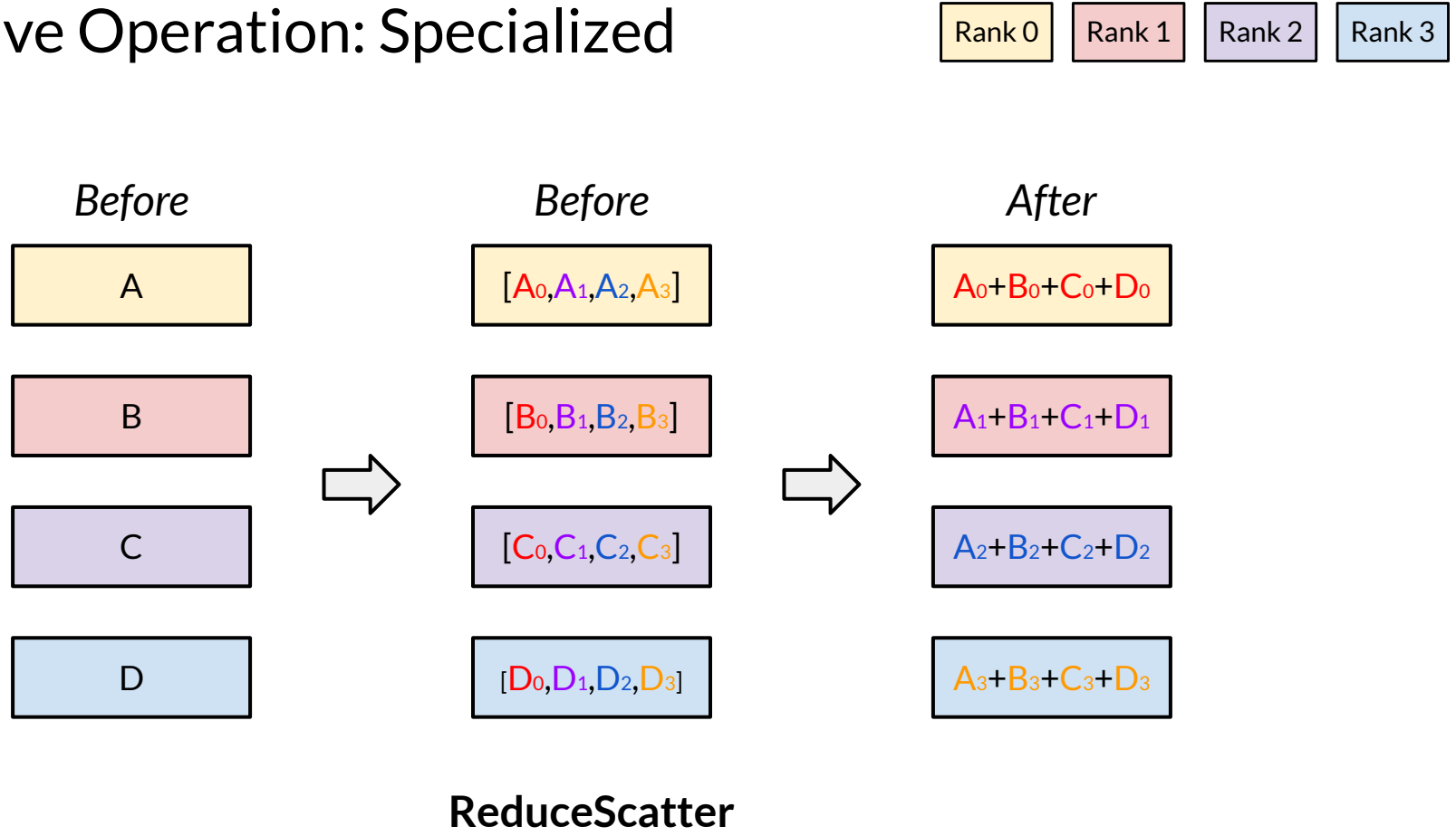
Collective Operation: Specialized

Rank 0 | Rank 1 | Rank 2 | Rank 3

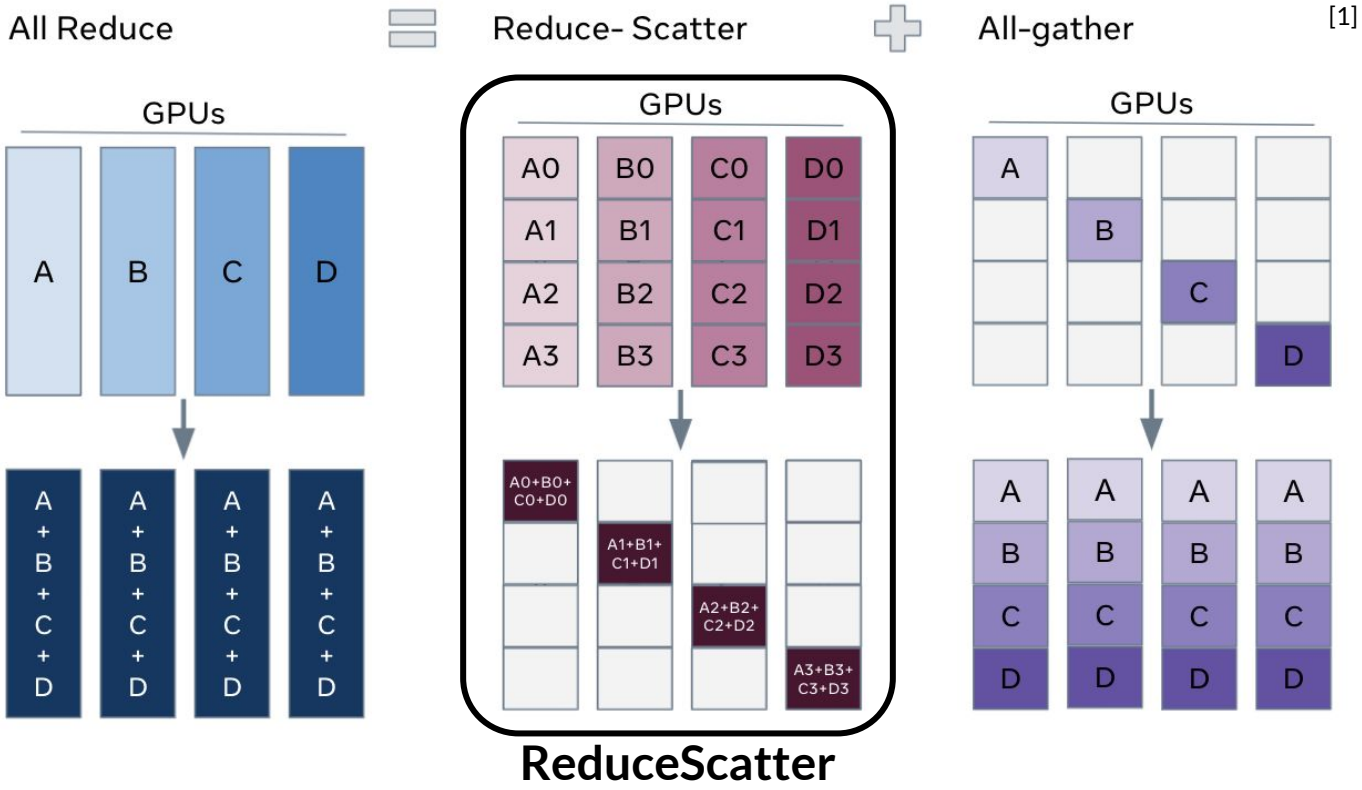


ReduceScatter

Collective Operation: Specialized



Collective Operation: Specialized



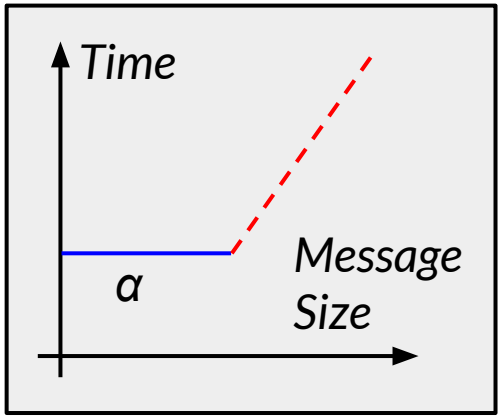
[1] https://engineering.fb.com/2021/07/15/open-source/fsdp/attachment/fsdp-graph-2a/

Collective Operation ≠ Collective Algorithm

- Operation / Primitive: specifies **what** data each rank should receive.
- Algorithm specifies **how** data moves among ranks.
- AllReduce algorithm:
  - Centralized
  - Ring
  - Recursive doubling

Communication Cost Model

- Modeling communication cost: $T_{msg}(n) = \alpha + \beta n$
  - α = latency / startup cost
  - β = Time per byte (≈ reciprocal of bandwidth)
  - n = message size
- Small messages -> latency-bound.
  - Minimize number of communication rounds.
- Large messages -> bandwidth-bound.
  - Minimize the total communicated bytes.
  - Maximize bandwidth utilization.



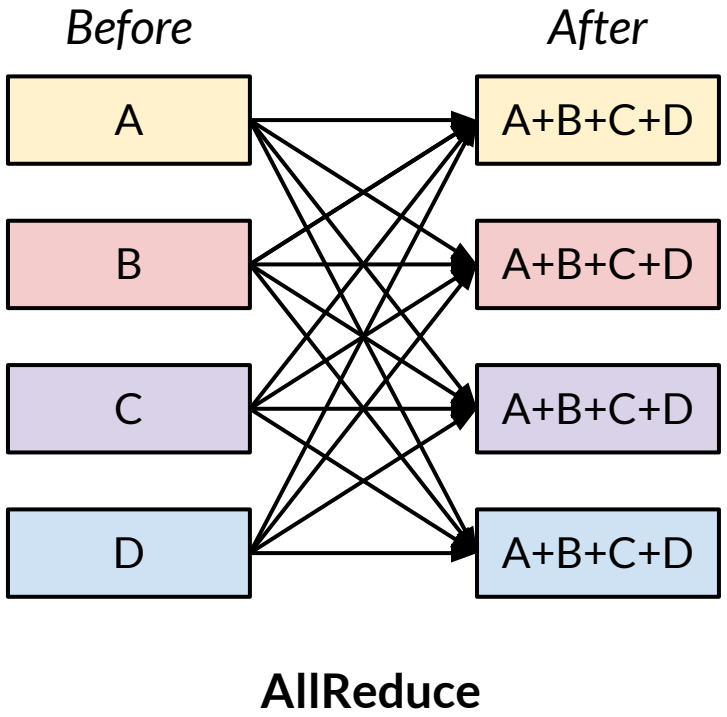
Collective Cost Generalization

- Modeling communication cost: $T_{msg}(n) = \alpha + \beta n$
- Collective communication cost: $T_{collective} \approx N_{steps}\alpha + V_{comm}\beta$
  - N_{steps} = communication rounds on critical path.
  - V_{comm} = bytes communicated by a rank.
- What makes a good collective algorithm?
  - Fewer steps.
  - Lower communication volume.
  - Higher memory bandwidth utilization.

Let’s understand cost with AllReduce

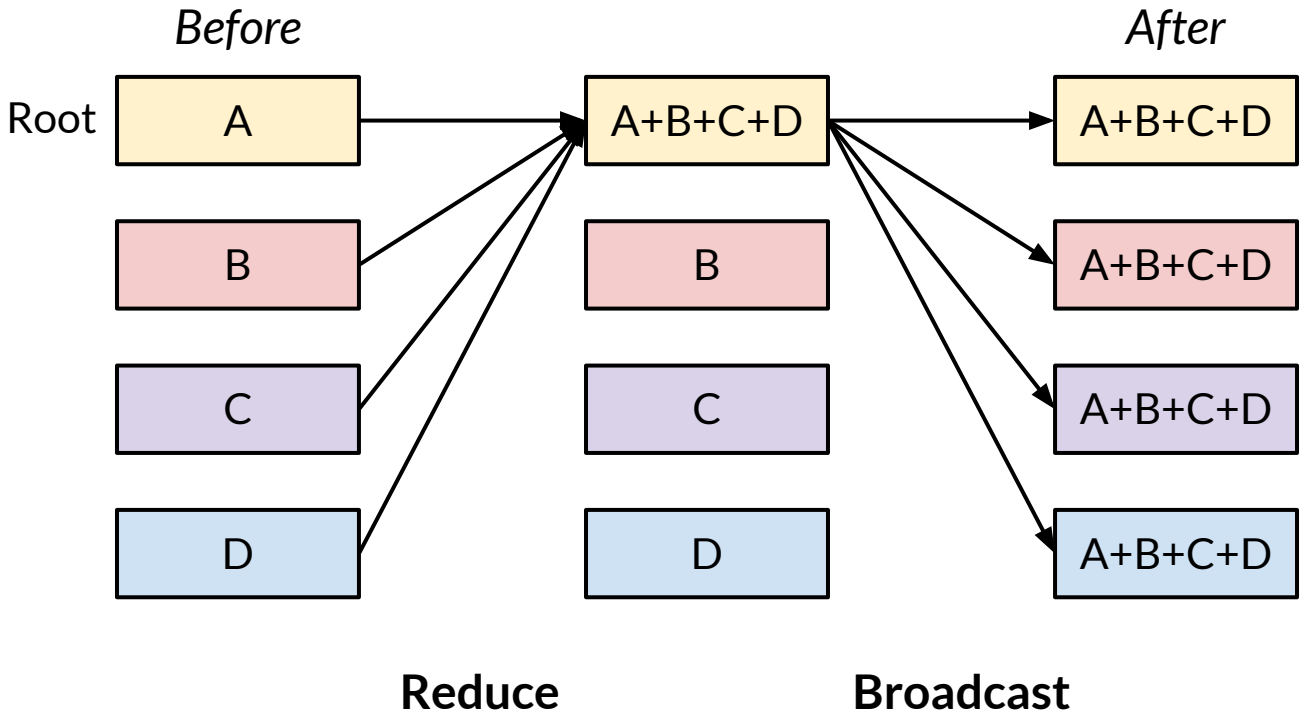
Rank 0 | Rank 1 | Rank 2 | Rank 3

- AllReduce algorithm:
  - Centralized
  - Ring
  - Recursive doubling
- Assumptions:
  - *p* ranks in total.
  - *n* bytes buffer on each rank.
  - Let’s ignore computation cost from Reduction on each rank.



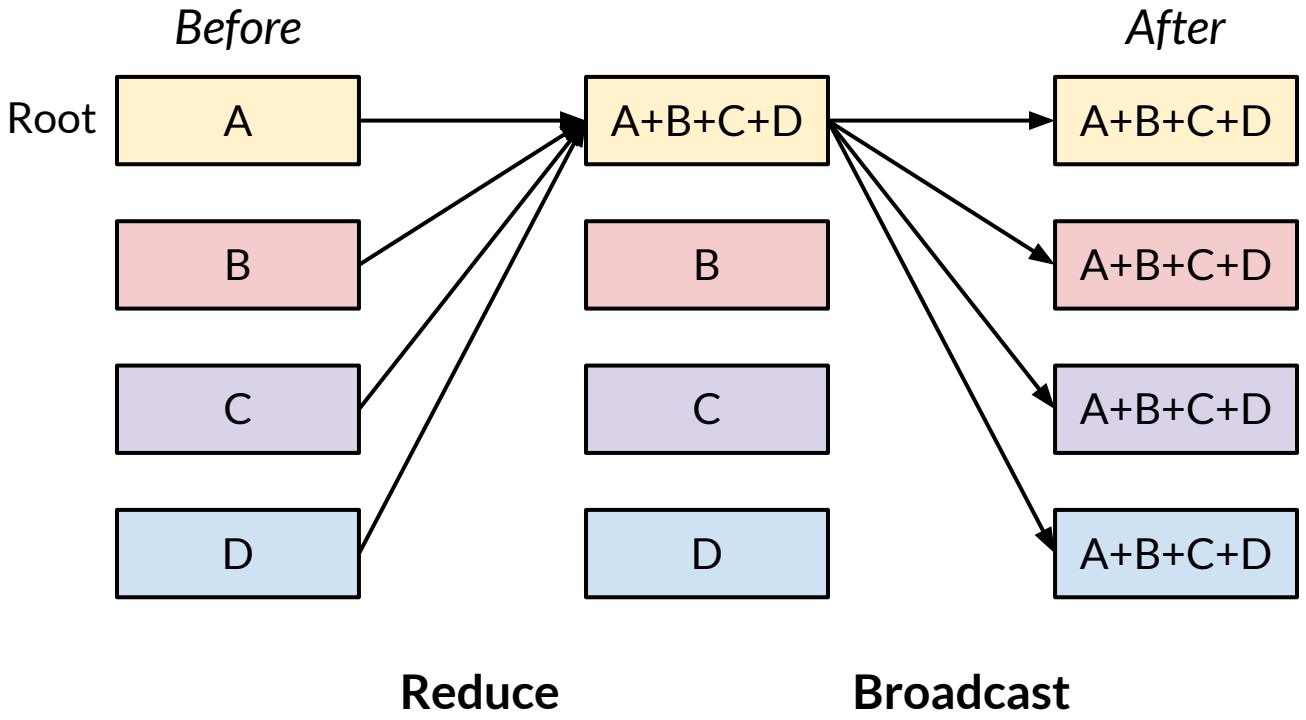
Centralized AllReduce

- Algorithm: find a root rank, Reduce (to root) + Broadcast (from Root).



Centralized AllReduce

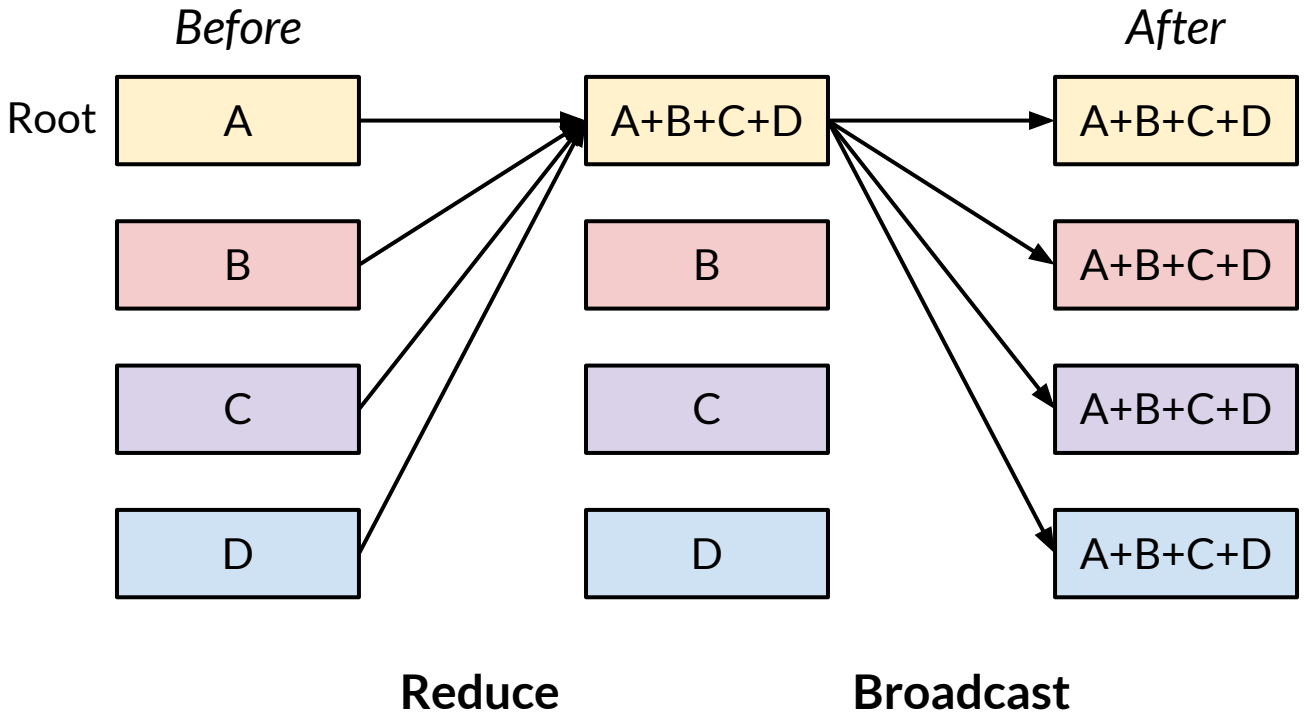
- Step count: *(p - 1)* rounds for both Reduce and Broadcast.



Centralized AllReduce

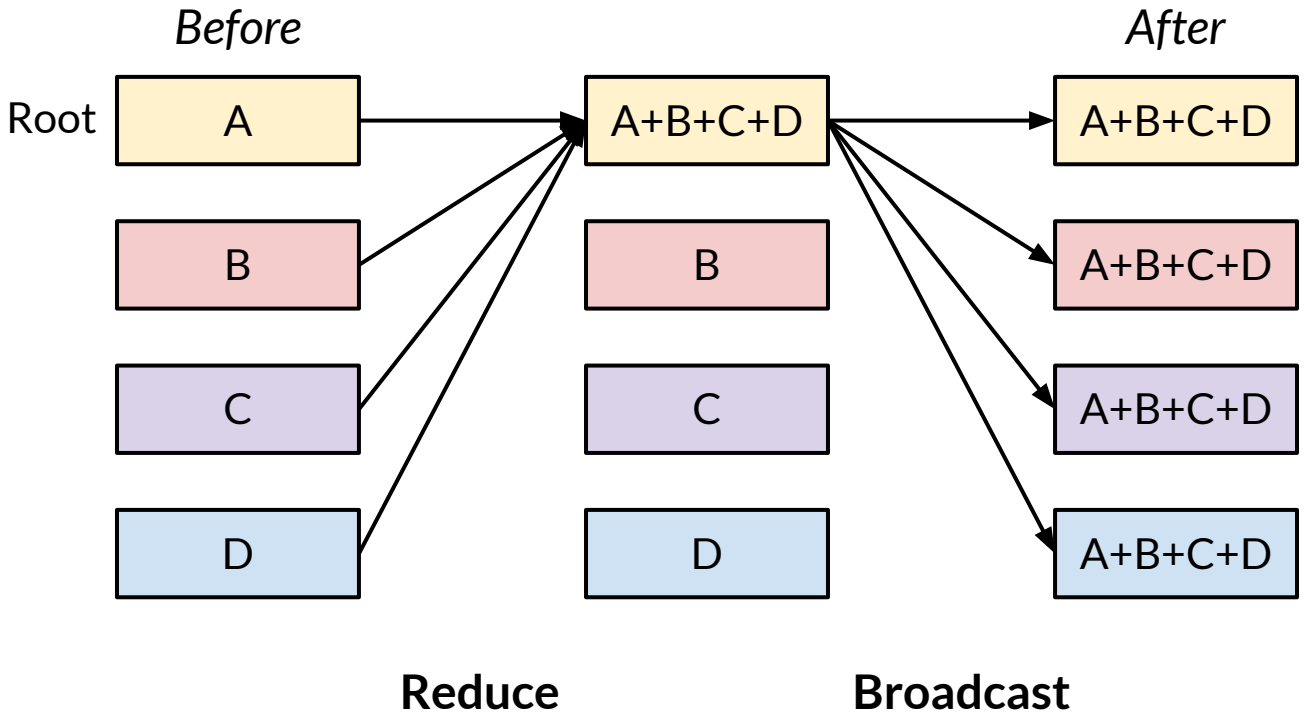
Root is the bottleneck!

- Communication volume: $(p - 1)n$ bytes for both Reduce and Broadcast.



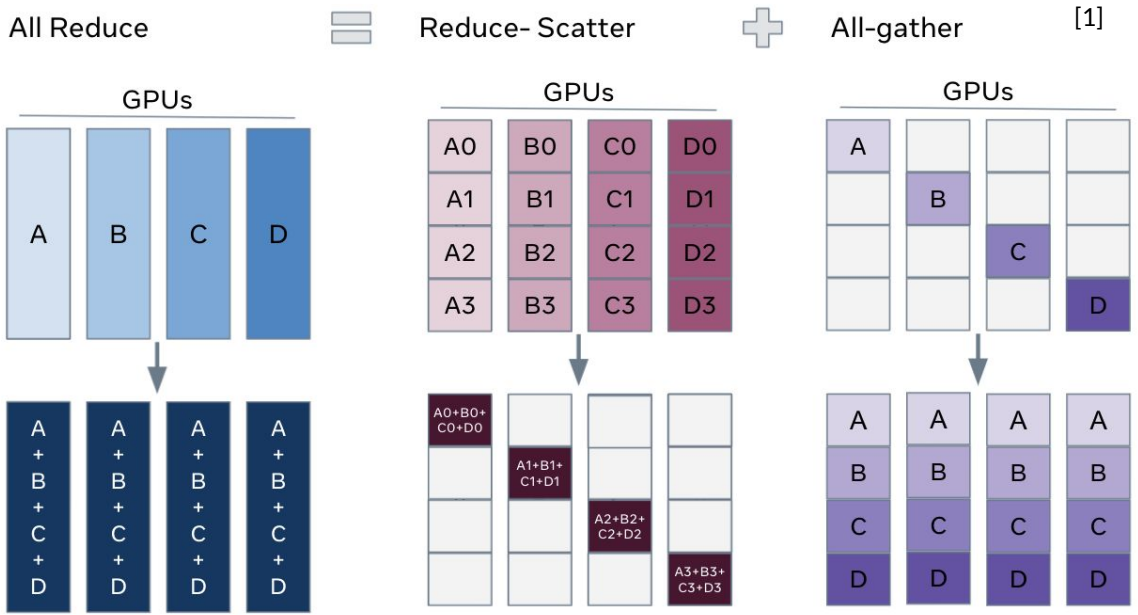
Centralized AllReduce

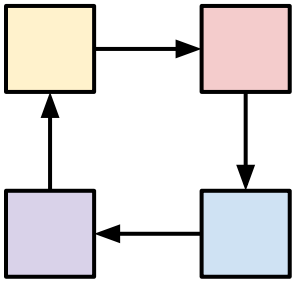
- Collective cost: $T_{centralized} \approx 2(p - 1)\alpha + 2(p - 1)n\beta$



Ring AllReduce

- Algorithm: ReduceScatter + AllGather.



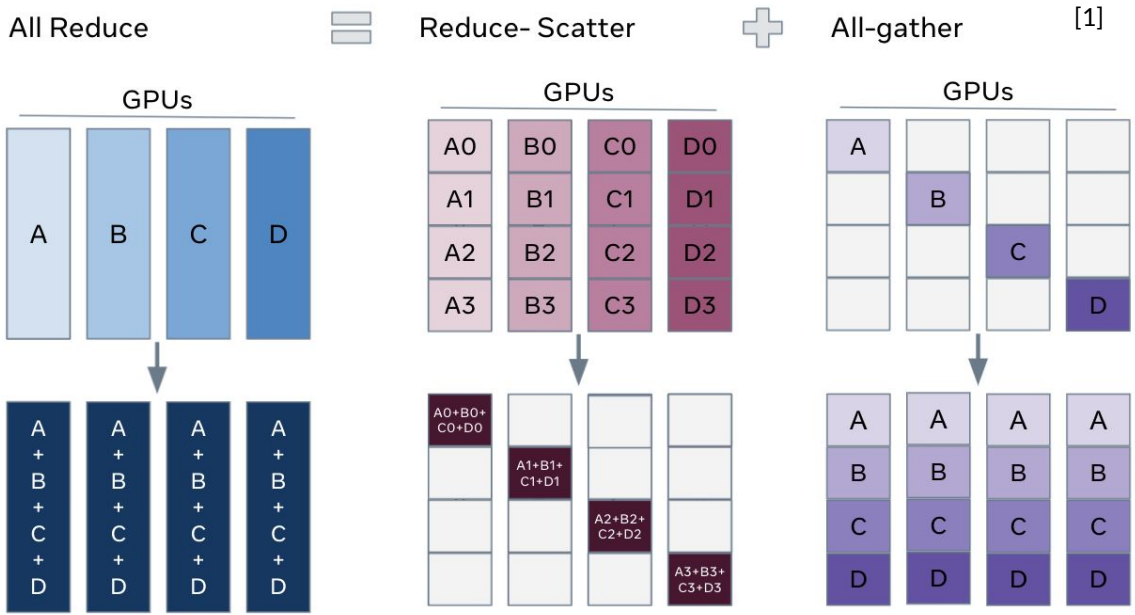


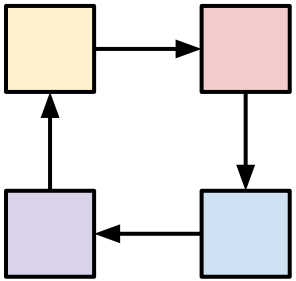
For each rank each round, send a chunk and receive a chunk.

[1] https://engineering.fb.com/2021/07/15/open-source/fsdp/attachment/fsdp-graph-2a/

Ring AllReduce

- Step count: *(p - 1)* rounds for both ReduceScatter and AllGather.





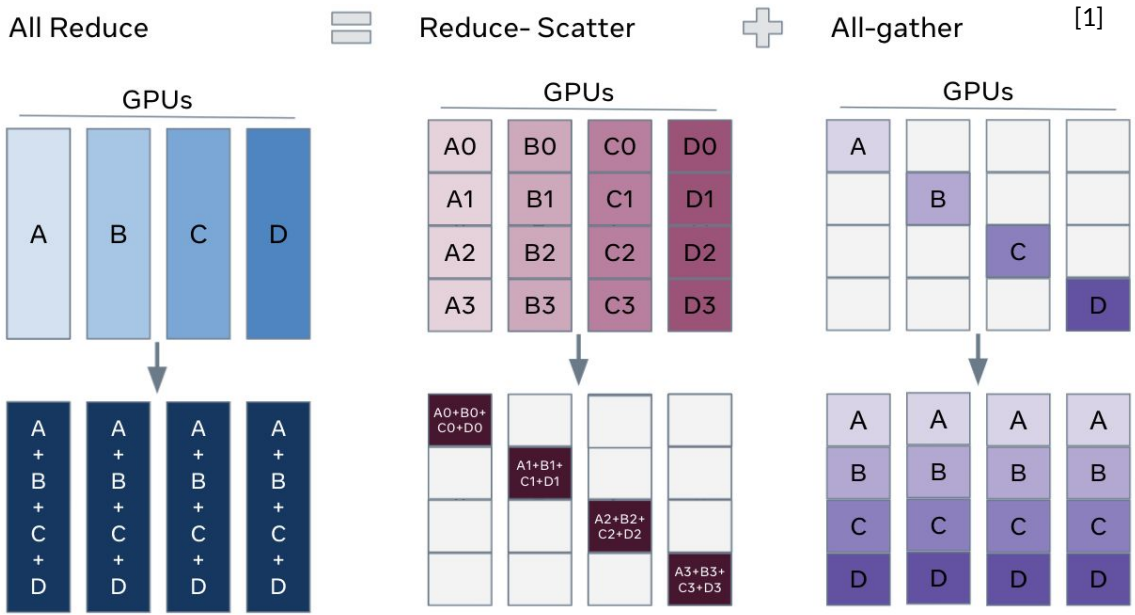
For each rank each round, send a chunk and receive a chunk.

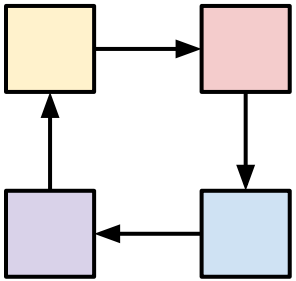
[1] https://engineering.fb.com/2021/07/15/open-source/fsdp/attachment/fsdp-graph-2a/

Ring AllReduce

(p - 1) rounds, each round n / p bytes.

- Communication volume: *(p - 1)n / p* bytes for both ReduceScatter + AllGather.



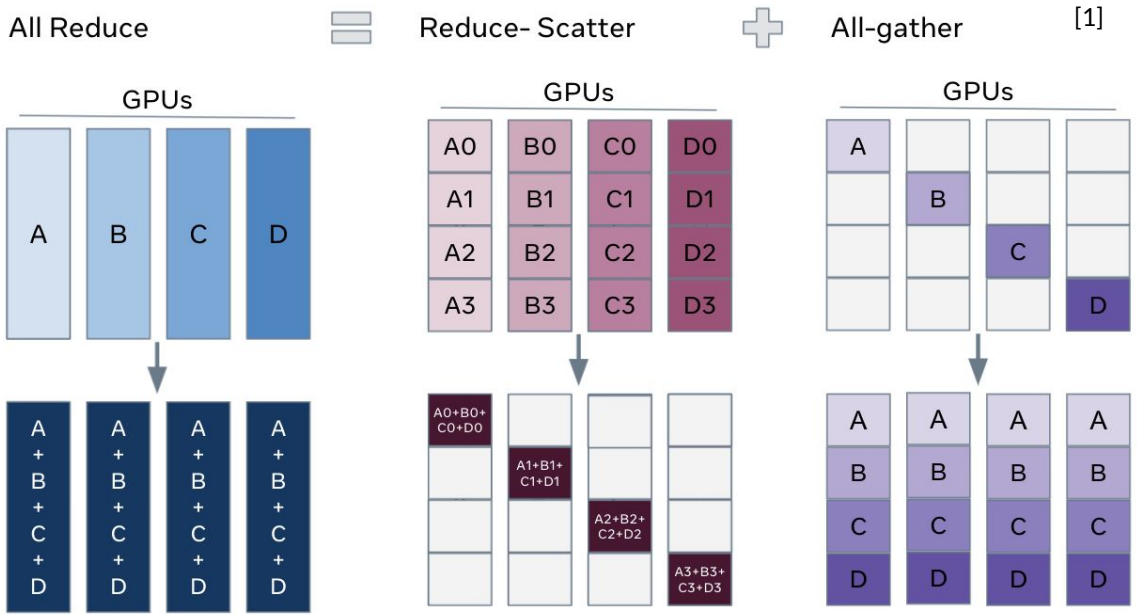


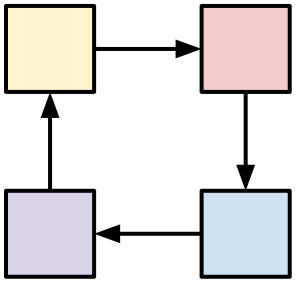
For each rank each round, send a chunk and receive a chunk.

[1] https://engineering.fb.com/2021/07/15/open-source/fsdp/attachment/fsdp-graph-2a/

Ring AllReduce

- Collective cost: $T_{ring} \approx 2(p - 1)\alpha + 2(p - 1)n\beta / p$



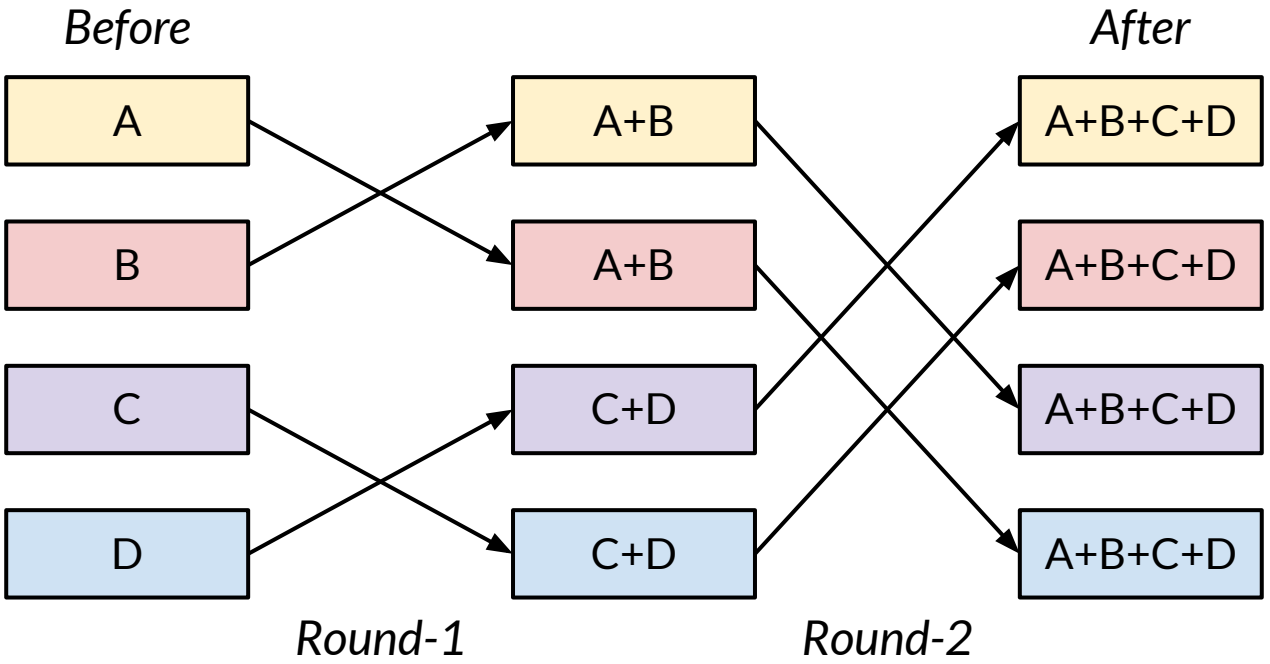


For each rank each round, send a chunk and receive a chunk.

[1] https://engineering.fb.com/2021/07/15/open-source/fsdp/attachment/fsdp-graph-2a/

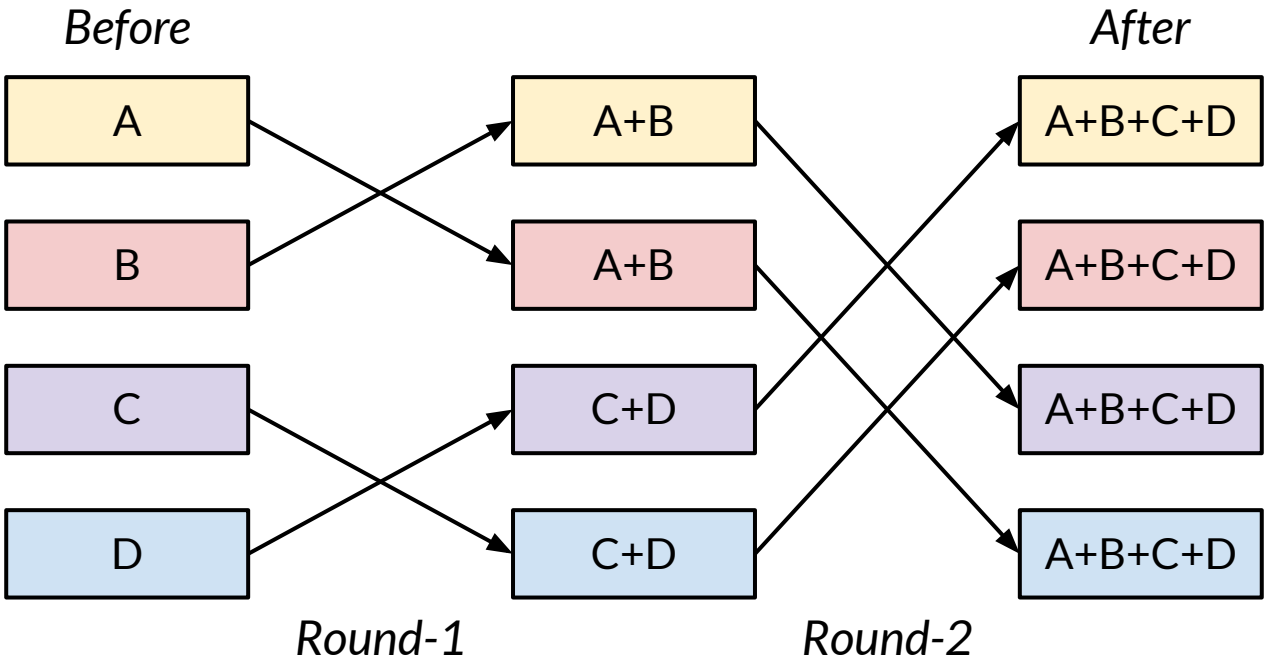
Recursive Doubling AllReduce

- Algorithm: In each round, each rank exchanges and reduces data with a partner, doubling the number of ranks represented in its partial result.



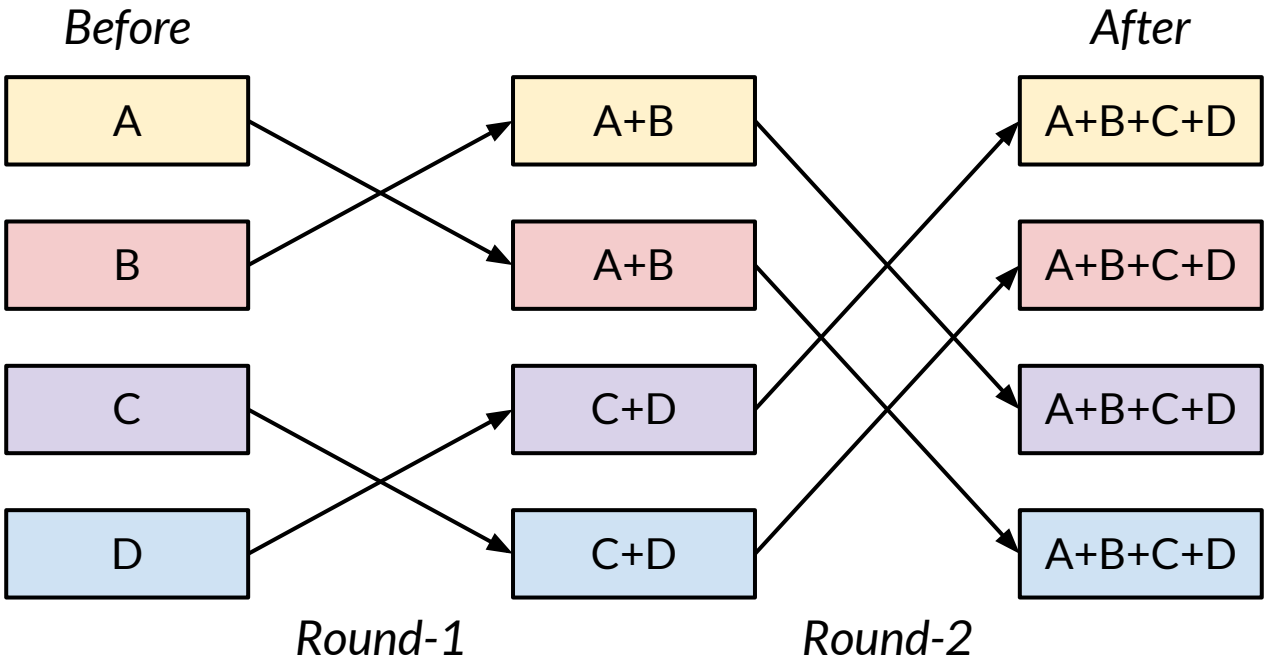
Recursive Doubling AllReduce

- Step count: $\log_2 p$ rounds in total.



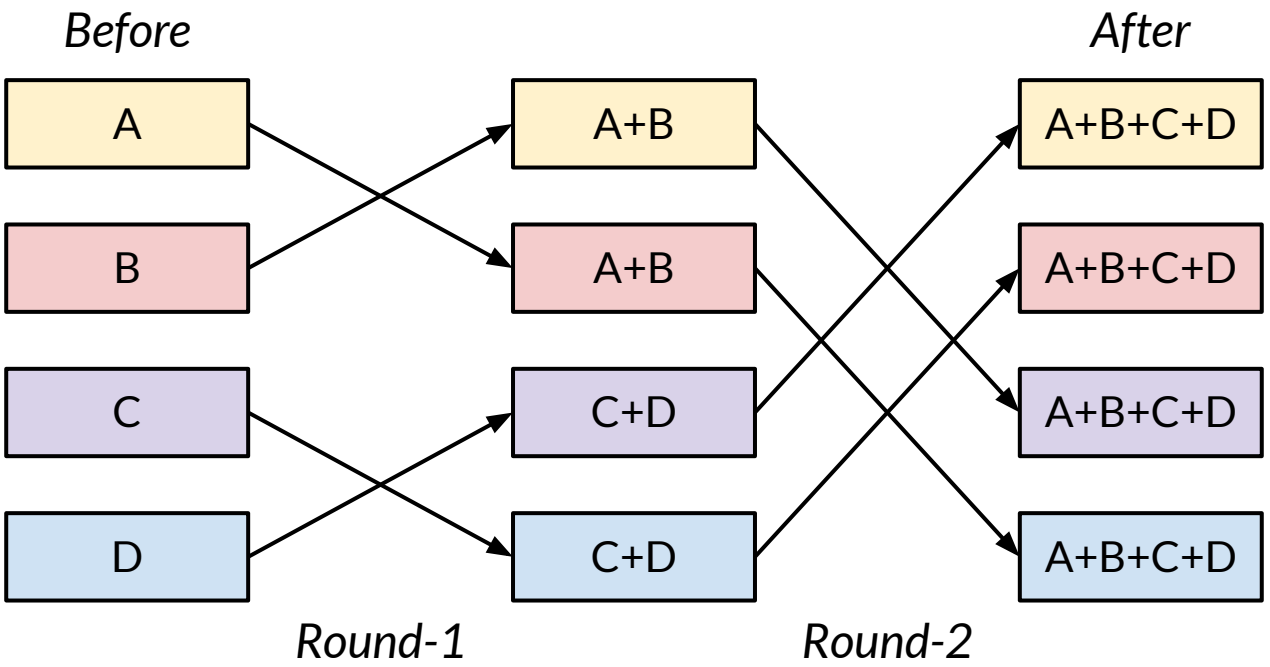
Recursive Doubling AllReduce

- Communication volume: n bytes for each rounds, so $n\log_2 p$ in total.



Recursive Doubling AllReduce

- Collective cost: $T_{RD} \approx \log_2 p\alpha + n\log_2 p\beta$



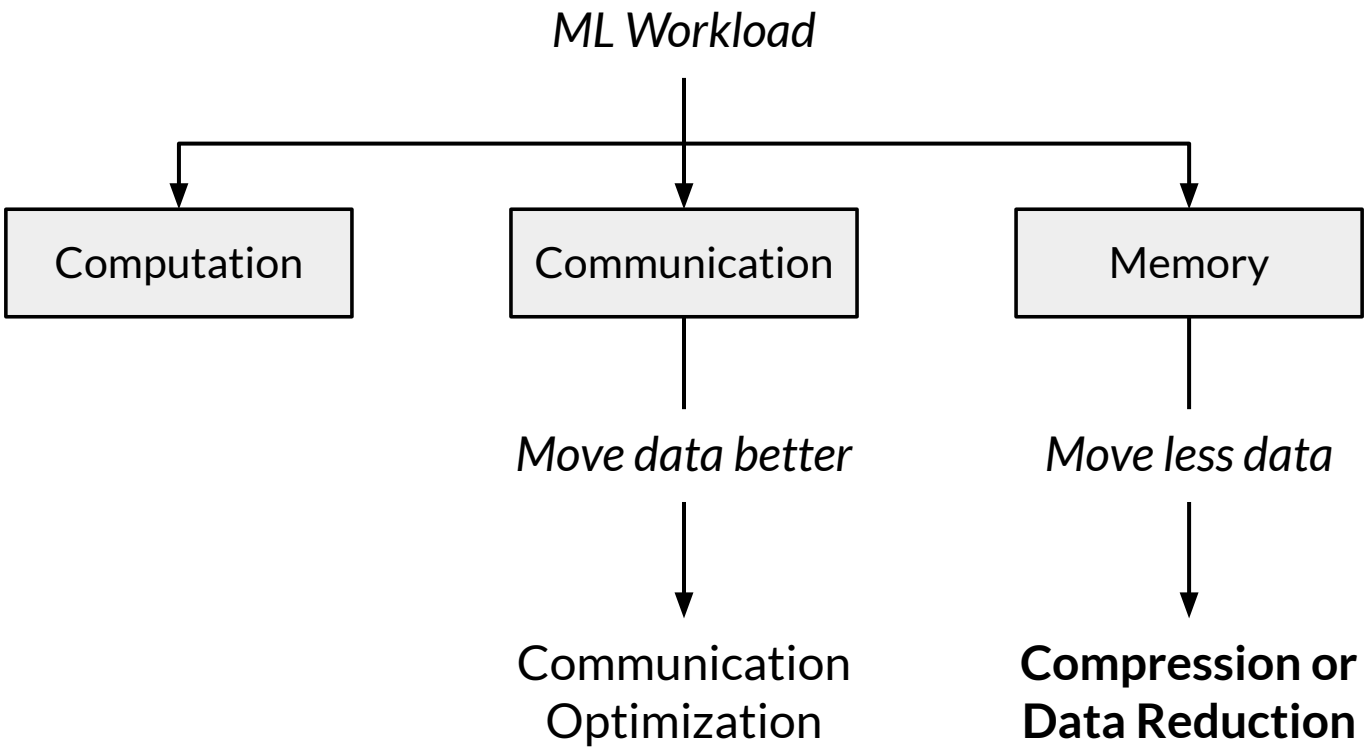
Summary

	Collective Cost	*Main Issue*	*Best for*
• $T_{centralized}$	$\approx 2(p-1)\alpha + 2(p-1)n\beta$	root bottleneck	baseline (or teaching)
• T_{ring}	$\approx 2(p-1)\alpha + 2(p-1)n\beta / p$	many rounds	large messages
• T_{RD}	$\approx \log_2 p\alpha + n\log_2 p\beta$	higher total bytes	small messages

Takeaway

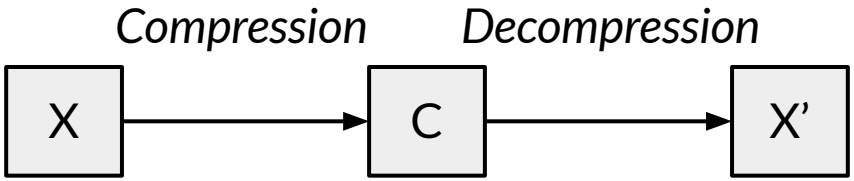
- There are always different algorithms/implementations for the same operation.
- Understanding trade-offs first: latency vs bandwidth in our case.
- There is no universally best AllReduce algorithm, so do other system tasks.

From HPC to ML Systems: A System View



Data Compression

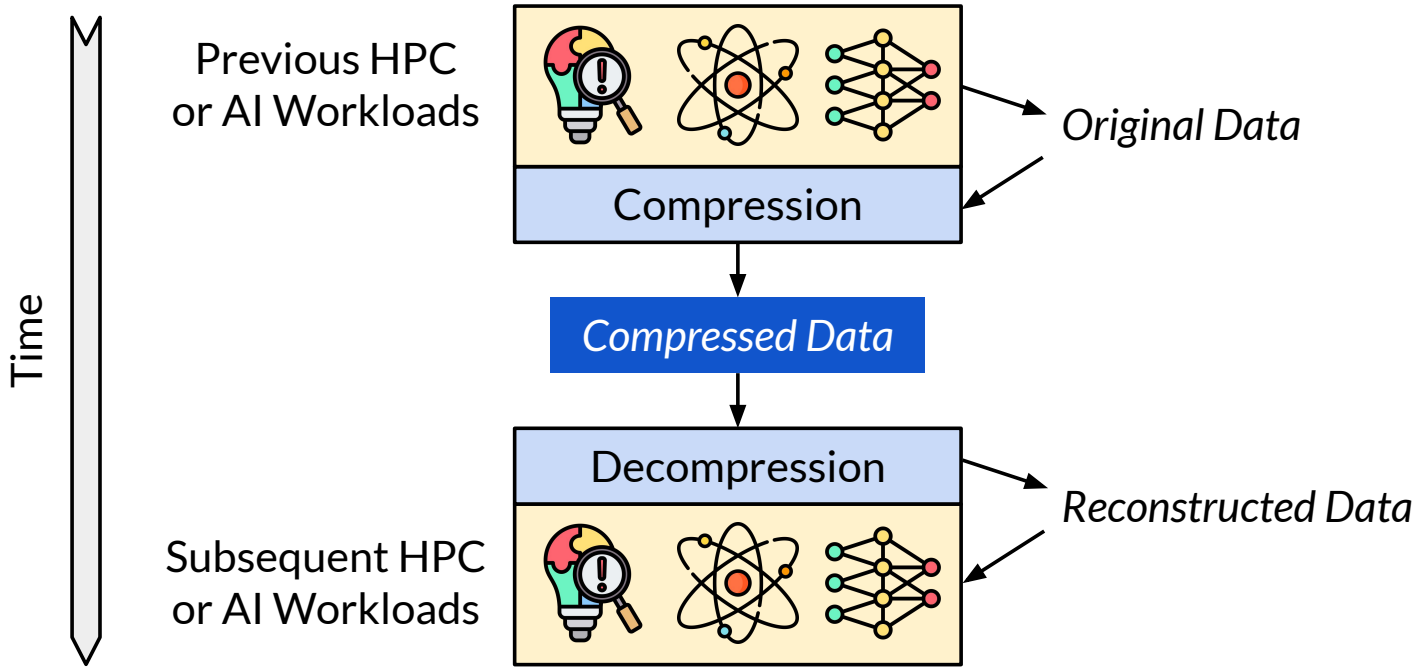
- Data compression: Represent data using fewer bits by exploiting redundancy or allowing controlled information loss.



- Lossless compression: X = X'
- Lossy compression: X ≈ X'
- Compression ratio: CR = Original Size / Compressed Size

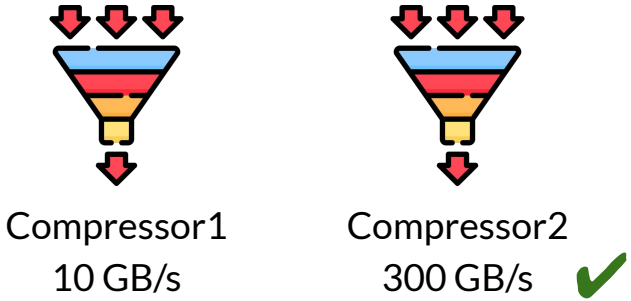
Compression as a Systems Optimization

- In system research, compression is not a single objective optimization task.

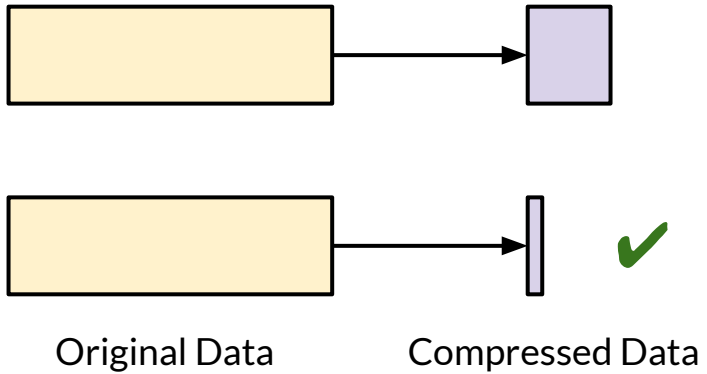


Compression as a Systems Optimization

- In system research, compression is not a single objective optimization task.
- Throughput: Original size / Compression or Decompression Time.



Compression is overhead in in-situ tasks!



- If lossy compression, data quality also needs to be considered.

Building Blocks of Compression

- In ML tasks, many numerical compressors combine data transformation, quantization, and lossless encoding to reduce redundancy.
  - Transform (discrete cosine transform, matrix matrix decomposition, etc)
  - Prediction (delta, Lorenzo, interpolation, regression, etc)
  - Quantization (converting floating point values to integers, lossy step)
  - Lossless Encoding (Huffman, Run-Length, Fixed-Length / Bit Packing)
  - …

Delta Encoding

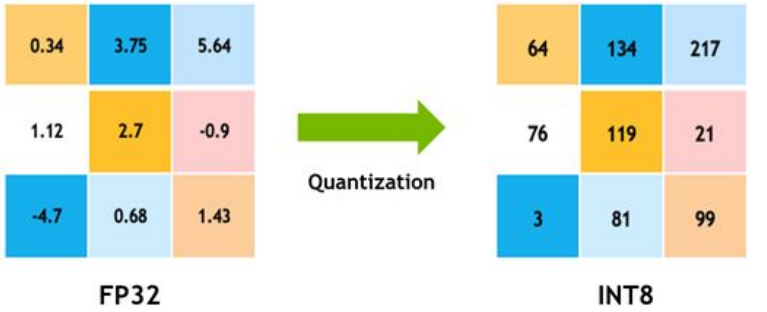
- Delta encoding replaces values with differences between consecutive values to exploit data correlation.
- Example: [100, 101, 102, 103, 104] -> [100, 1, 1, 1, 1]
- Small or repeated differences are more compressible than the original values.

- Is Delta Encoding parallel-friendly? 

Quantization

- Quantization maps **individual** continuous values or numbers independently to a smaller set of discrete values or lower bit-widths.
  - Uniform Quantization, Non-Uniform Quantization, Precision Scaling.

[1]



[2]

```
__device__ inline int quantization(double data, double recipPrecision)
{
    double dataRecip = data*recipPrecision;
    int s = dataRecip>=-0.5?0:1;
    return (int)(dataRecip+0.5) - s;
}
```

- Is Quantization parallel-friendly? 🤔

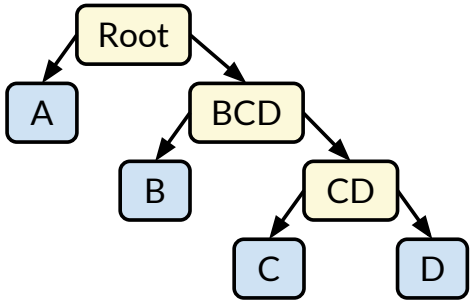
[1] https://developer.nvidia.com/blog/achieving-fp32-accuracy-for-int8-inference-using-quantization-aware-training-with-tensorrt/
[2] https://github.com/szcompressor/cuSZp/blob/master/src/cuSZp_kernels_1D_f64.cu

Huffman Encoding

- Huffman coding assigns shorter bit codes to more frequent symbols and longer codes to less frequent symbols.
- Example: Given a string array AAABBCD.

A	3
B	2
C	1
D	1

Step1:
Generating
Frequency Table



Step2:
Constructing
Huffman Tree

A	0
B	10
C	110
D	111

Step3:
Building Huffman
Codebook

AAABBCD

↓

000101011011111

Step4:
Compression

- Is Huffman Encoding parallel-friendly? 🤔

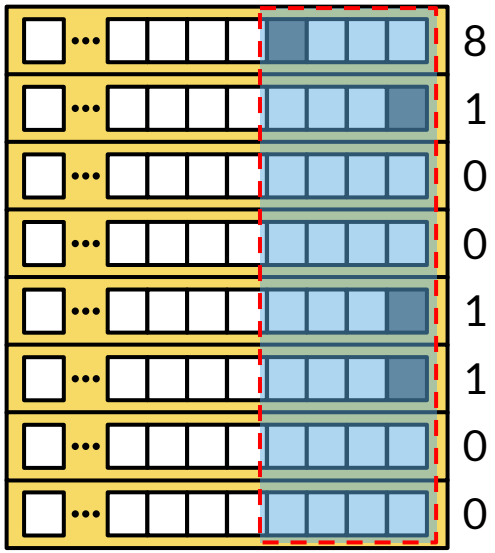
Run-Length Encoding

- Run-Length Encoding (RLE) replaces consecutive repeated values with a value and its repetition count.
- Example: AAAABBBCC -> (A,4) (B,3) (C,2)
- Example: ABCDEF -> (A,1) (B,1) (C,1) (D,1) (E,1) (F,1)
- RLE is effective when data contains long runs of repeated values.

- Is Run-Length Encoding parallel-friendly? 

Fixed-Length Encoding / Bit Packing

- Fixed-Length Encoding (or Bit Packing) packs values using the minimum number of bits required by their value range.
- Example: Given an integer array [8,1,0,0,1,1,0,0]



- Is Fixed-Length Encoding parallel-friendly? 🤔

Putting Compression Components Together

- A practice from HPC, cuSZp[1]
  - Compression Pipeline: Quantization + Delta + Fixed-Length Encoding
- Algorithm Optimizations:
  - Meta Data and Zero Blocks Preservation
  - Outlier Fixed-Length Encoding
  - Dimension-aware Delta Encoding
- Performance Optimizations:
  - Synchronization latency control, control-flow optimization, memory coalescing, vectorization, PTX inline, etc.

[1] https://github.com/szcompressor/cuSZp

Summary

	Opportunity
● Delta / Prediction	Correlation between values
● Quantization	Precision that can be discarded
● Huffman Encoding	Non-uniform symbol frequencies
● Run-Length Encoding	Consecutive repeated values
● Fixed-Length Encoding	Unused bits in fixed-width representations

Takeaway

- The best pipeline always depends on data characteristics and system constraints.