19th USENIX Conference on File and Storage Technologies (FAST '21)

Behemoth: A Flash-centric Training Accelerator for Extreme-scale DNNs

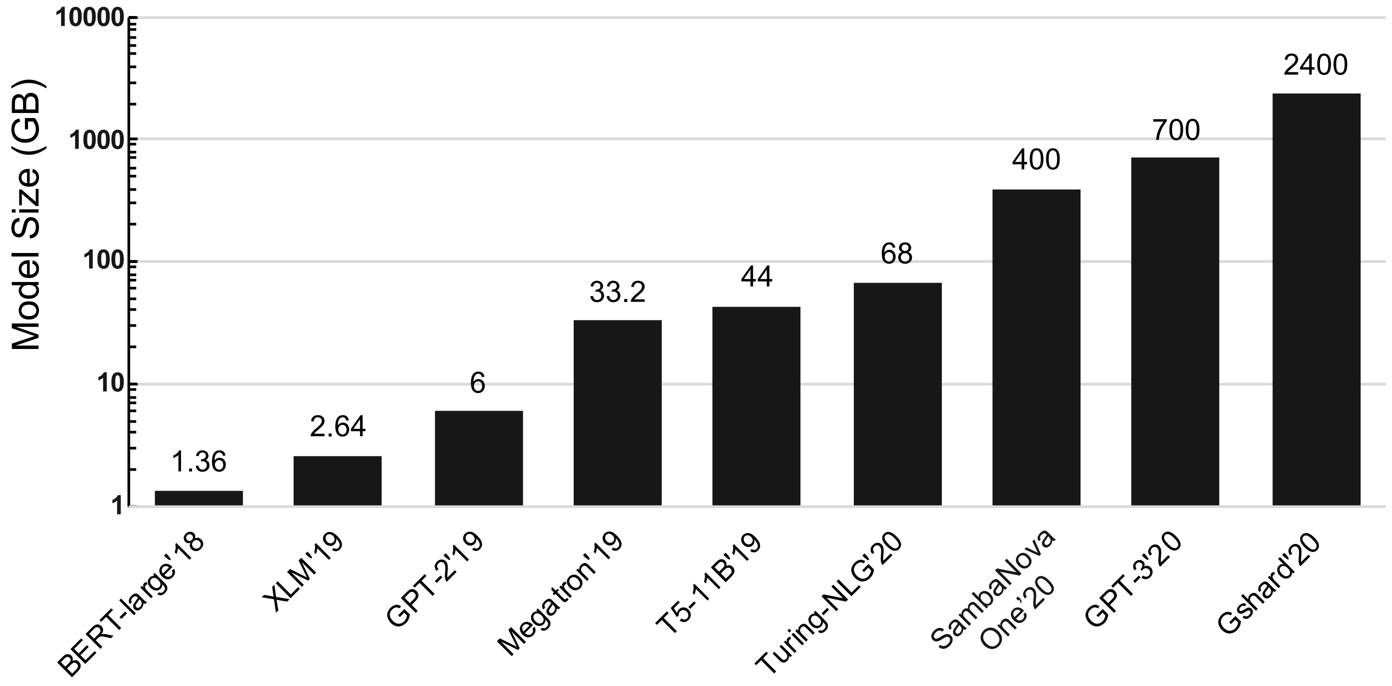
Shine Kim[1,2,*] Yunho Jin[1,*] Gina Sohn[1] Jonghyun Bae[1] Tae Jun Ham[1] Jae W. Lee[1]

[1] Seoul National University [2] Samsung Electronics

*Equal Contributions

Explosive expansion of DNNs

- Deep Neural Networks have become widespread in various application domains
  - Natural language processing, computer vision, recommendation, and so on
- Increasing the model size is crucial to improve accuracy of DNNs
  - Extreme-scale models demand a tremendous amount of computation and memory capacity



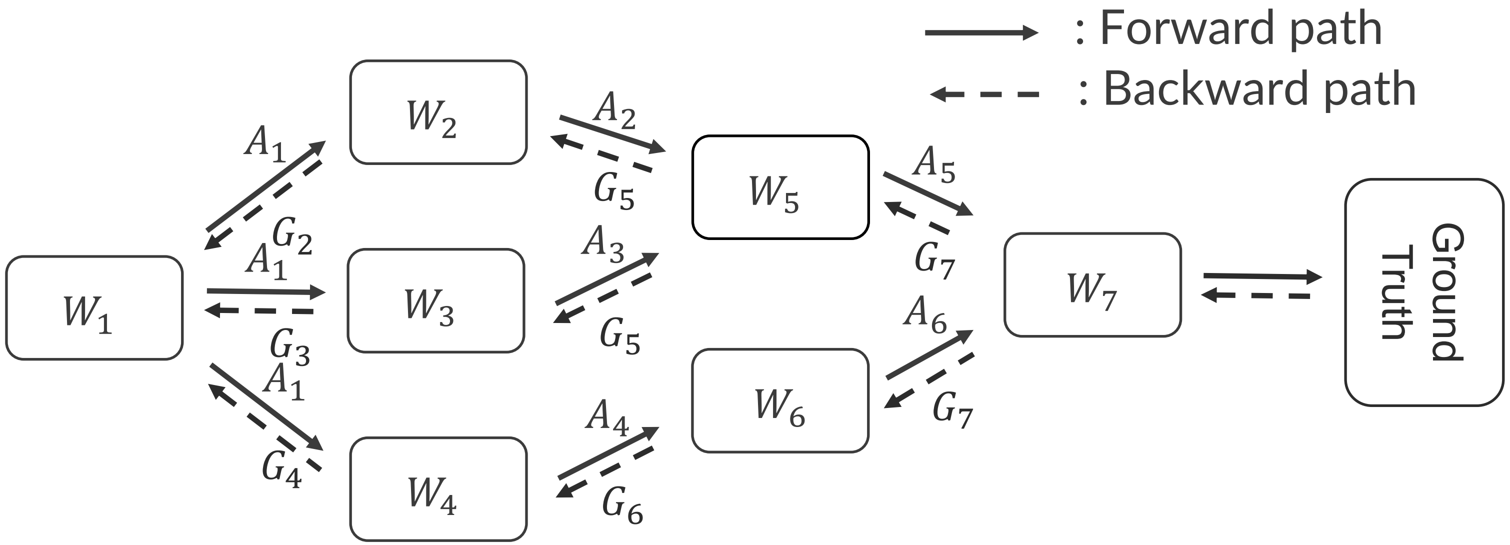
DNN Training Process and Dataflow

- DNN training is a repetitive process of matrix operation

DNN Training Process and Dataflow

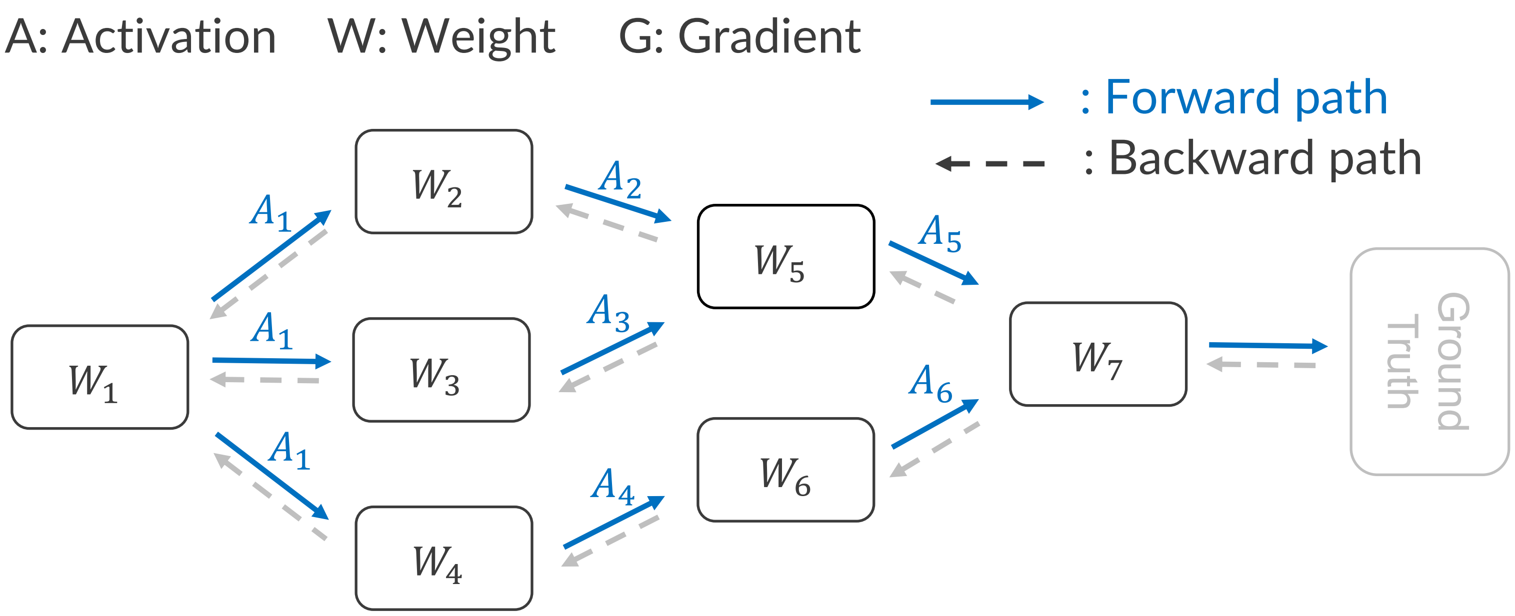
- DNN training is a repetitive process of matrix operation
  - Forward path: multiply activation and weights to generate expected value
  - Calculate the difference (loss) between expected value and ground truth
  - Backward path: propagate the loss in backward order and update weights

A: Activation W: Weight G: Gradient



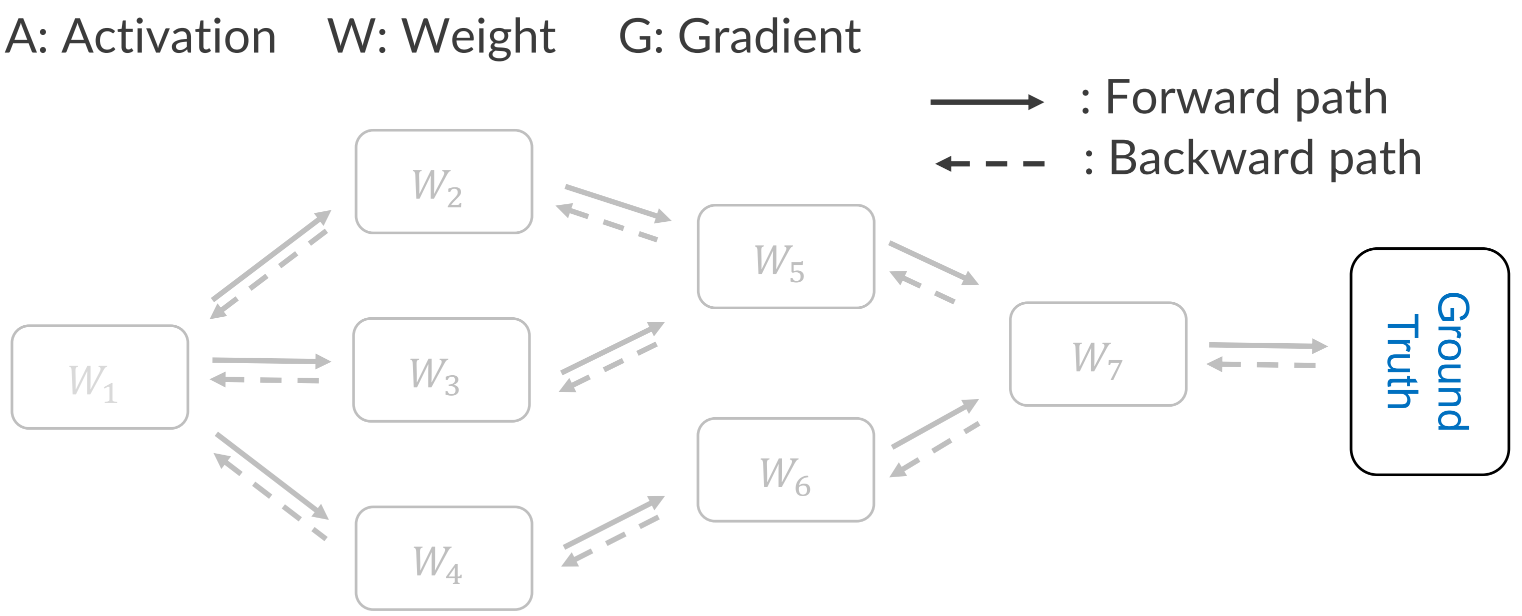
DNN Training Process and Dataflow

- DNN training is a repetitive process of matrix operation
  - **Forward path: multiply activations and weights to generate expected value**
  - Calculate the difference (loss) between expected value and ground truth
  - Backward path: propagate the loss in backward order and update weights



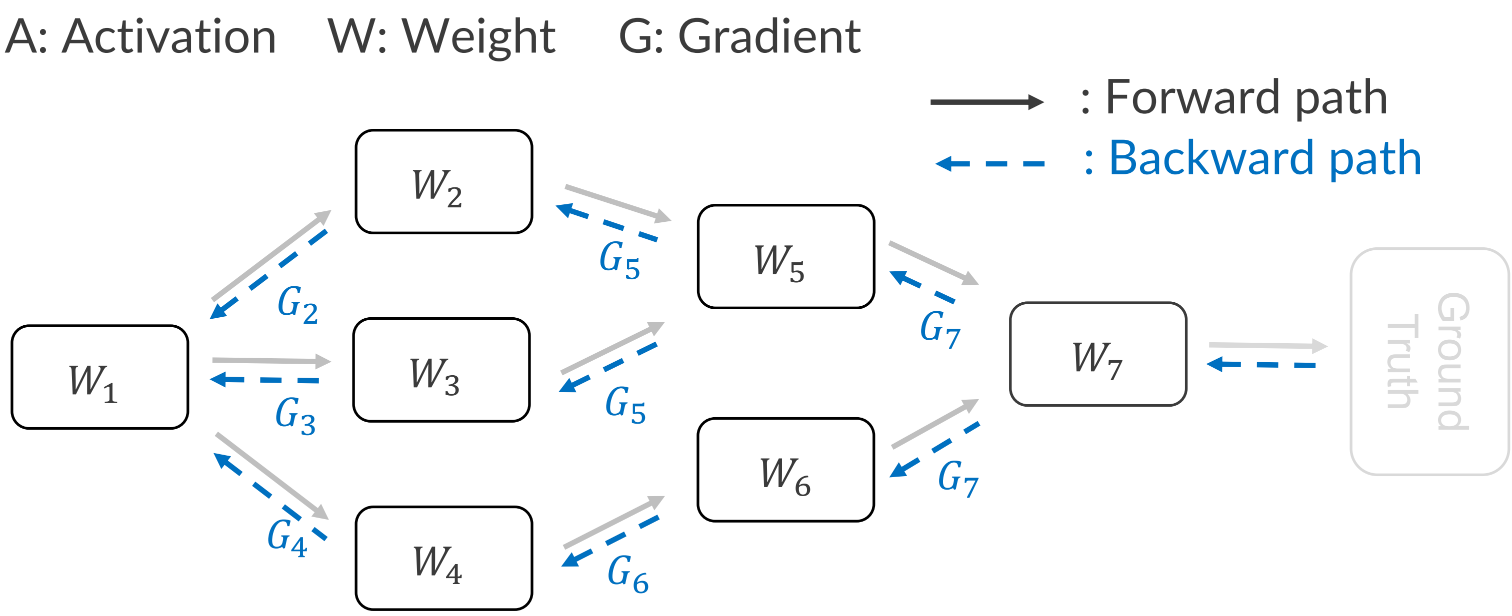
DNN Training Process and Dataflow

- DNN training is a repetitive process of matrix operation
  - Forward path: multiply activations and weights to generate expected value
  - **Calculate the difference (loss) between expected value and ground truth**
  - Backward path: propagate the loss in backward order and update weights



DNN Training Process and Dataflow

- DNN training is a repetitive process of matrix operation
  - Forward path: multiply activation and weights to generate expected value
  - Calculate the difference (loss) between expected value and ground truth
  - **Backward path: propagate the loss in backward order and update weights**

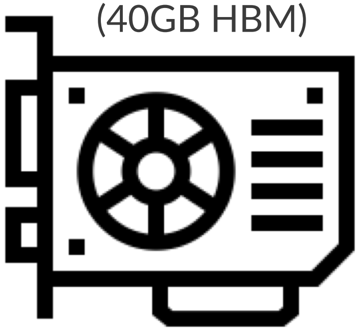


Challenges for Extreme-scale NLP Model Training

- Memory capacity wall
  - DNN model size exceeds memory capacity of a single GPU
  - Forces users to partition the model and distribute to HBM DRAM on GPU (Model Parallelism)

Split a model into layers

GPU A100
(40GB HBM)



X 18

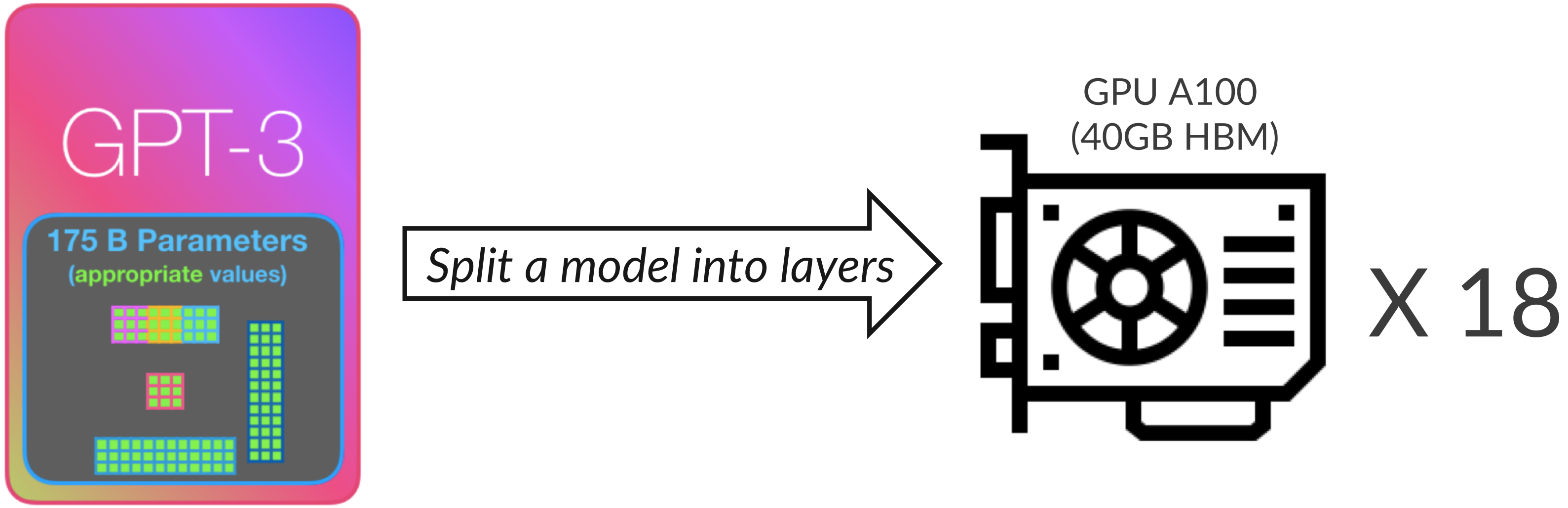
700GB parameters

720GB memory (HBM)

*figure borrowed from http://jalammar.github.io/how-gpt3-works-visualizations-animations/

Challenges for Extreme-scale NLP Model Training

- Memory capacity wall
  - DNN model size exceeds memory capacity of a single GPU
  - Forces users to partition the model and distribute to HBM DRAM on GPU (Model Parallelism)

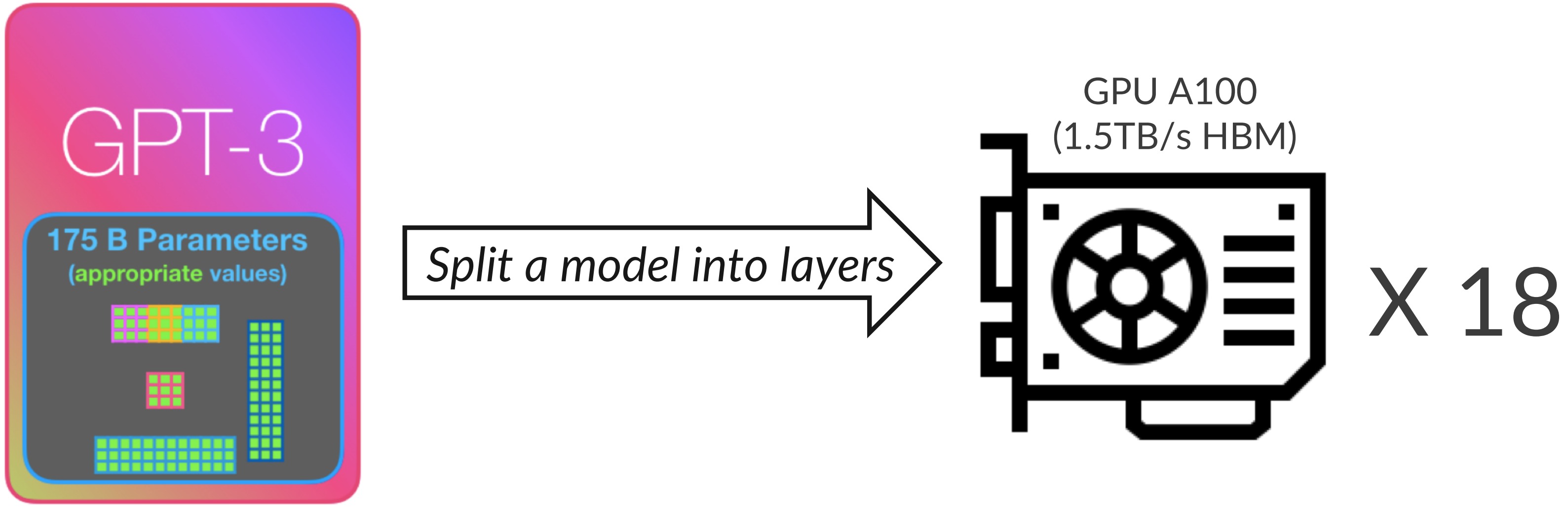


700GB parameters *720GB memory (HBM)*

Careful load balancing is required!

Challenges for Extreme-scale NLP Model Training

- Memory B/W underutilization
  - As a DNN model (matrix) size increased, each value in the matrix is reused more often
  - The memory B/W requirement does not increase as the computation amount increases

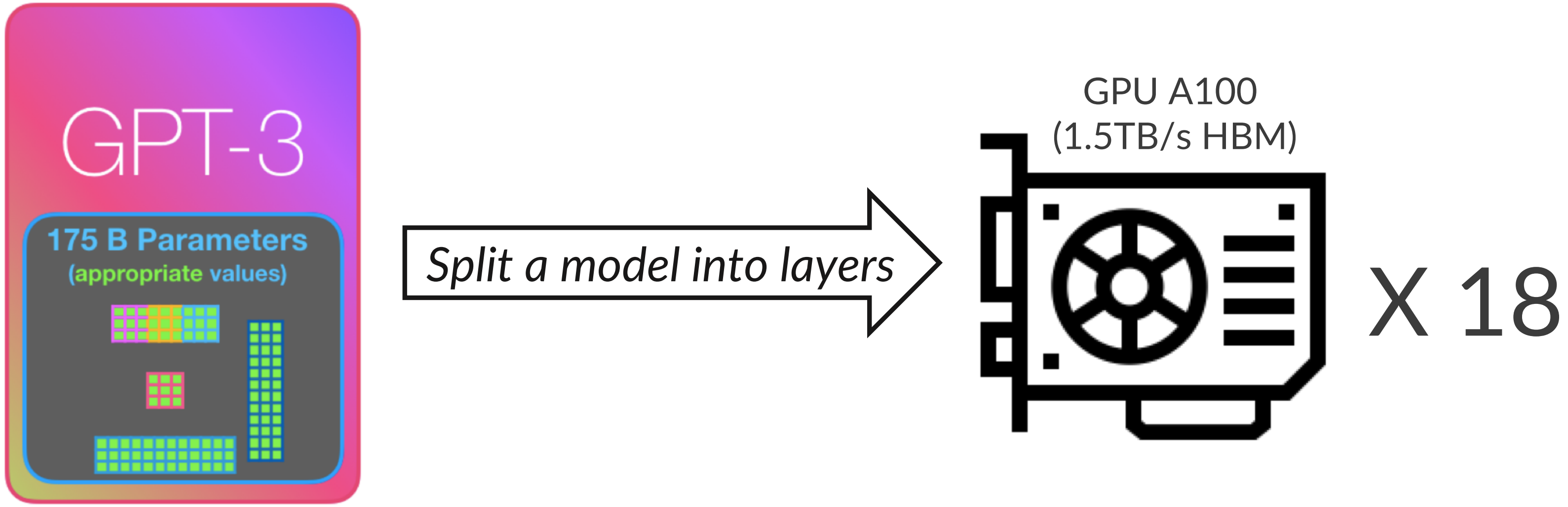


50GB/s[1] (required B/W) ***Total 28TB/s (HBM B/W)***

1) *Training with batch size of 16 on 840 TFLOPs compute core*

Challenges for Extreme-scale NLP Model Training

- Memory B/W underutilization
  - As a DNN model (matrix) size increased, each value in the matrix is reused more often
  - The memory B/W requirement does not increase as the computation amount increases



50GB/s[1] (required B/W) *Total 28TB/s (HBM B/W)*

Significant memory B/W underutilization occurs!

1) *Training with batch size of 16 on 840 TFLOPs compute core*

Challenges for Extreme-scale NLP Model Training

Scaling of DNNs necessitates

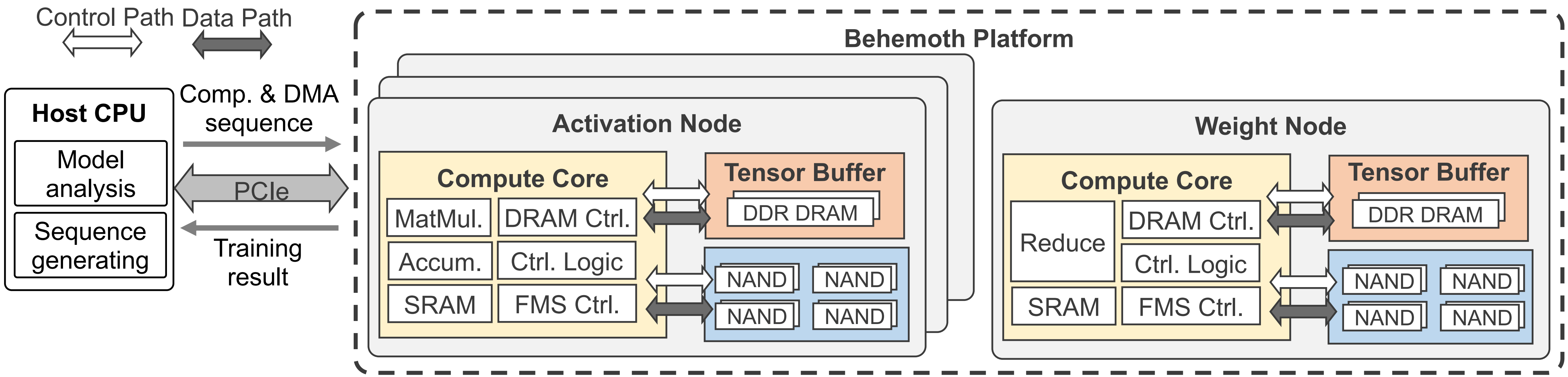
a new memory system with

high-capacity and low-cost

(replacing low-capacity, high-cost HBM)

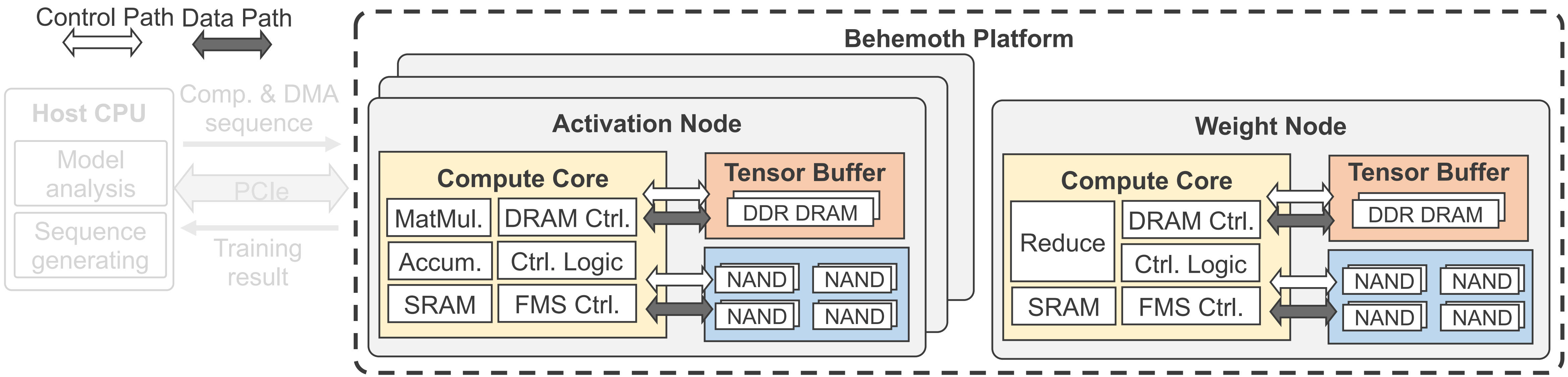
System Overview

Behemoth holds the entire DNN model in a single node to enable **data-parallel** training



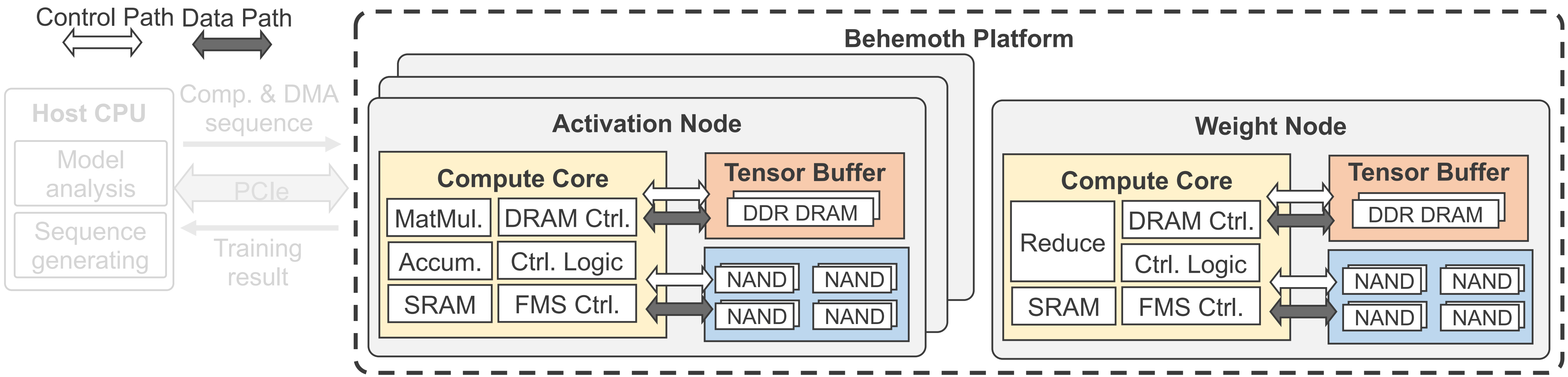
System Overview

Behemoth holds the entire DNN model in a single node to enable **data-parallel** training



System Overview

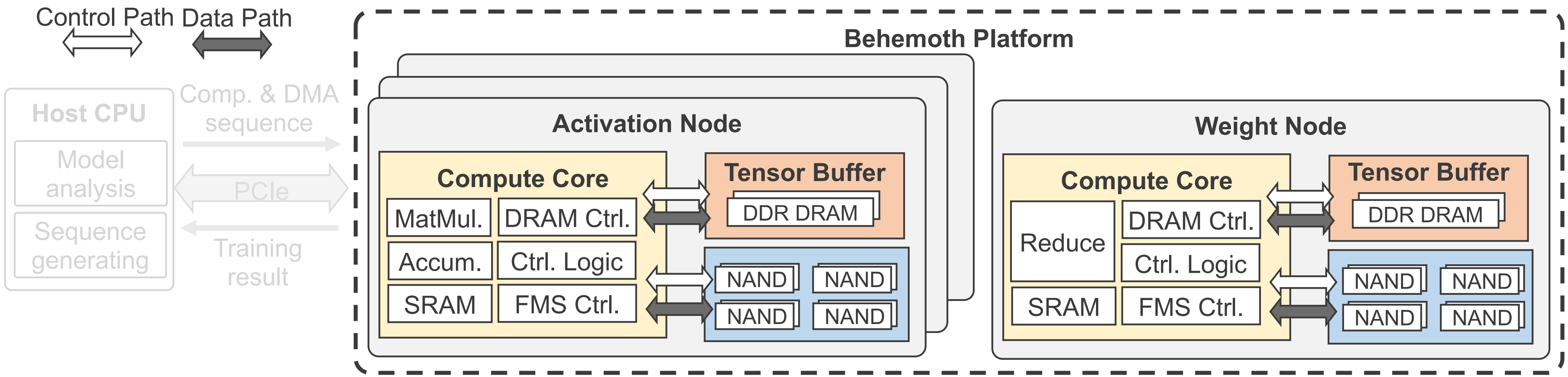
Behemoth holds the entire DNN model in a single node to enable **data-parallel** training



- Compute Core: compute tensors and transfer data between Tensor Buffer and NANDs

System Overview

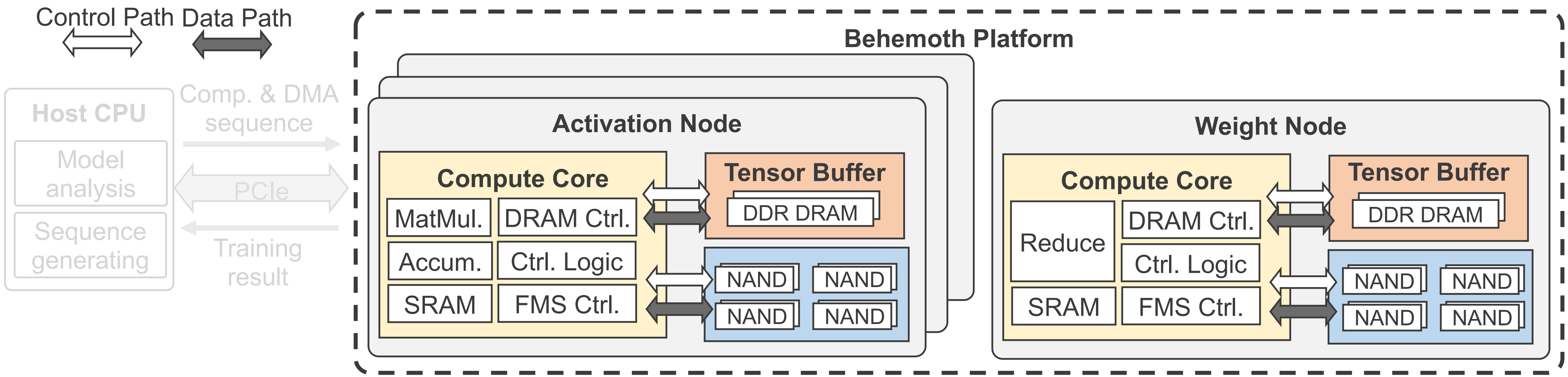
Behemoth holds the entire DNN model in a single node to enable **data-parallel** training



- Compute Core: compute tensors and transfer data between Tensor Buffer and NANDs
- Tensor Buffer: keep tensors in DDR DRAM serving as a staging (prefetching/offloading) area for NANDs

System Overview

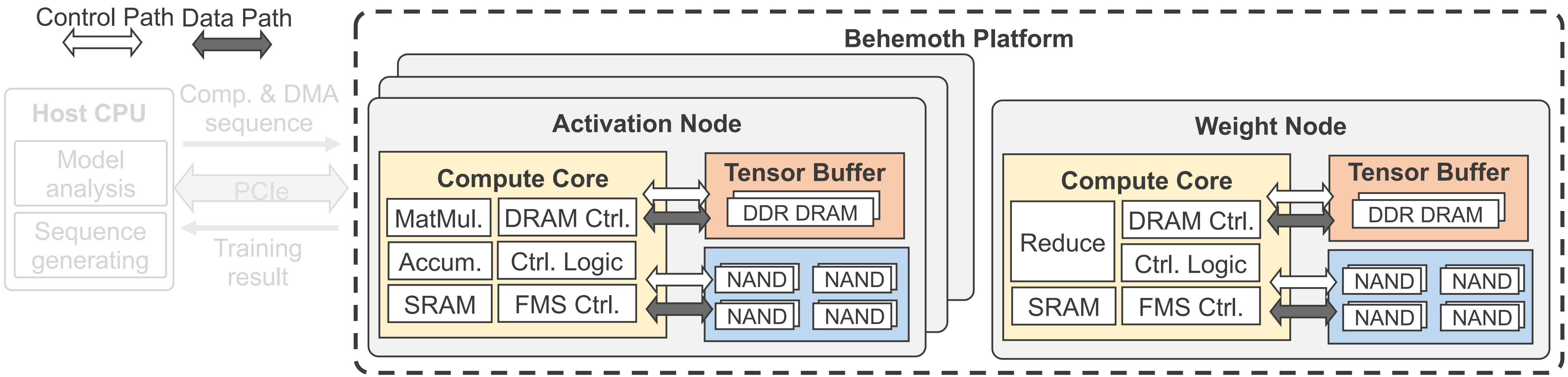
Behemoth holds the entire DNN model in a single node to enable **data-parallel** training



- Compute Core: compute tensors and transfer data between Tensor Buffer and NANDs
- Tensor Buffer: keep tensors in DDR DRAM serving as a staging (prefetching/offloading) area for NANDs
- NANDs: store tensors like HBMs in conventional training system

System Overview

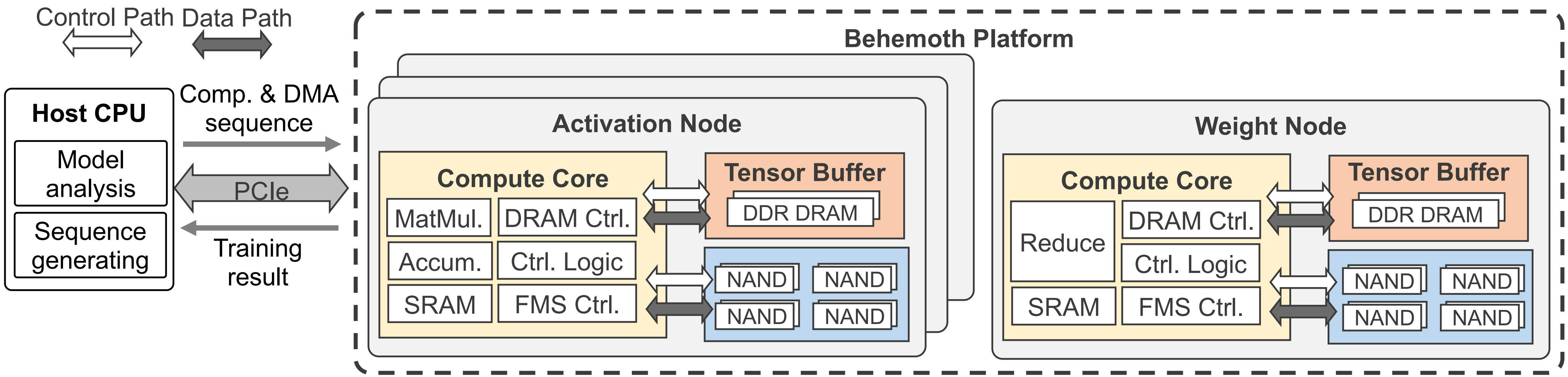
Behemoth adopts a **two-level** memory architecture using DDR DRAM and NAND flash to reduce the DNN training cost



- Compute Core: compute tensors and transfer data between Tensor Buffer and NANDs
- Tensor Buffer: keep tensors in DDR DRAM serving as a staging (prefetching/offloading) area for NANDs
- NANDs: store tensors like HBMs in conventional training system

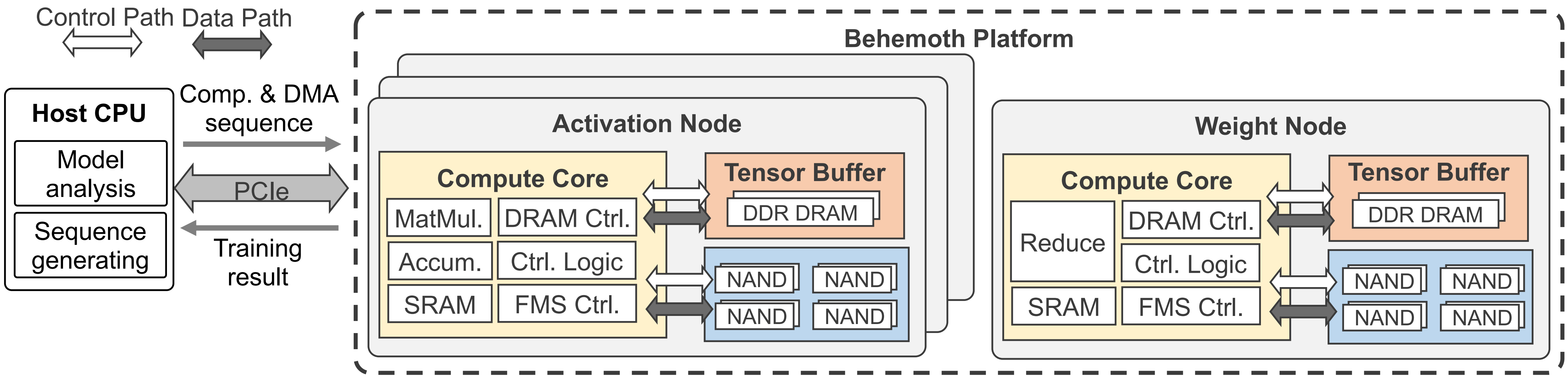
System Overview

Behemoth platform consist of multiple Activation Nodes and a single Weight Node enabling data-parallel training



System Overview

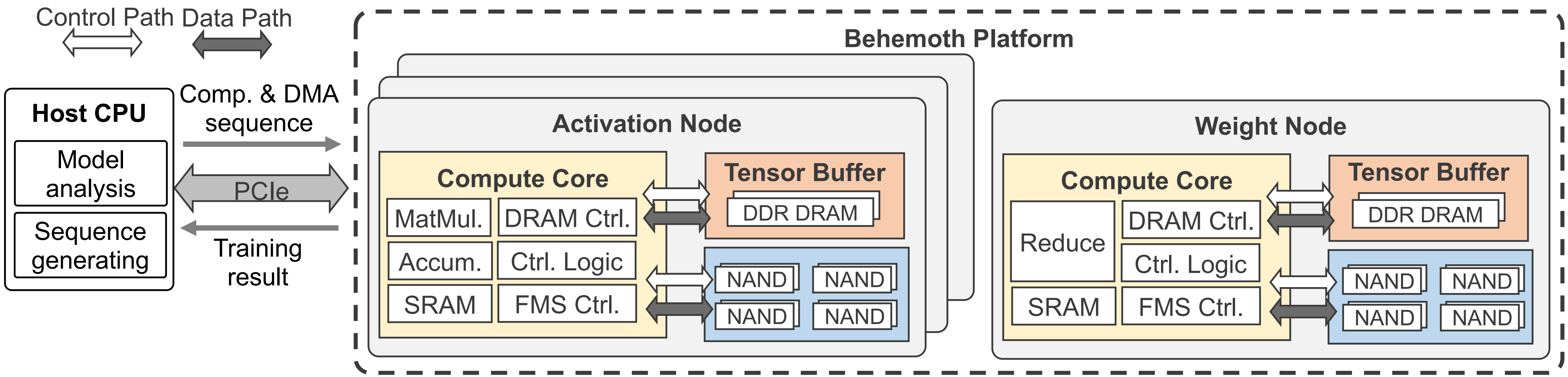
Behemoth platform consist of multiple Activation Nodes and a single Weight Node enabling data-parallel training



- Activation Node: compute and store activations

System Overview

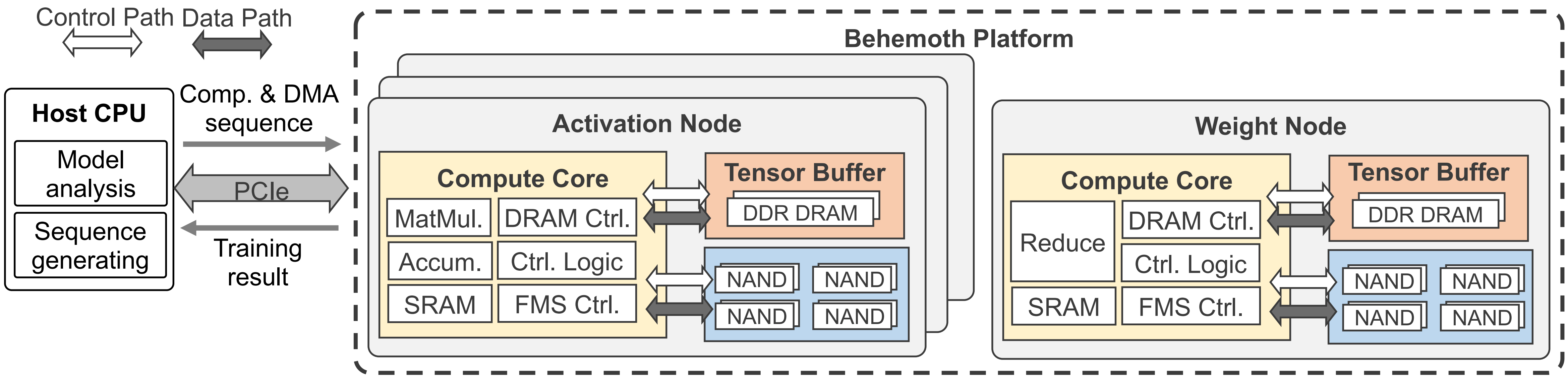
Behemoth platform consist of multiple Activation Nodes and a single Weight Node enabling data-parallel training



- Activation Node: compute and store activations
- Weight Node: update and store weights

System Overview

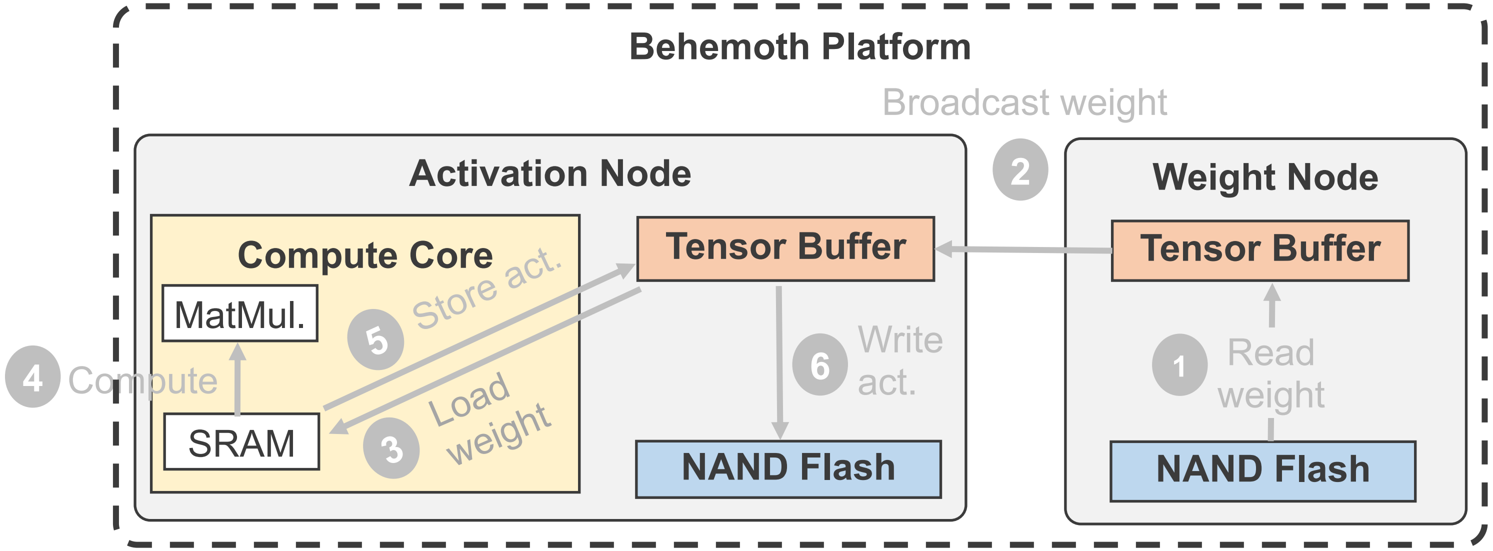
Behemoth platform consist of multiple Activation Nodes and a single Weight Node enabling data-parallel training



- Activation Node: compute and store activations
- Weight Node: update and store weights
- Host system: transfer training command sequence to Behemoth

Example Execution Walk-Through

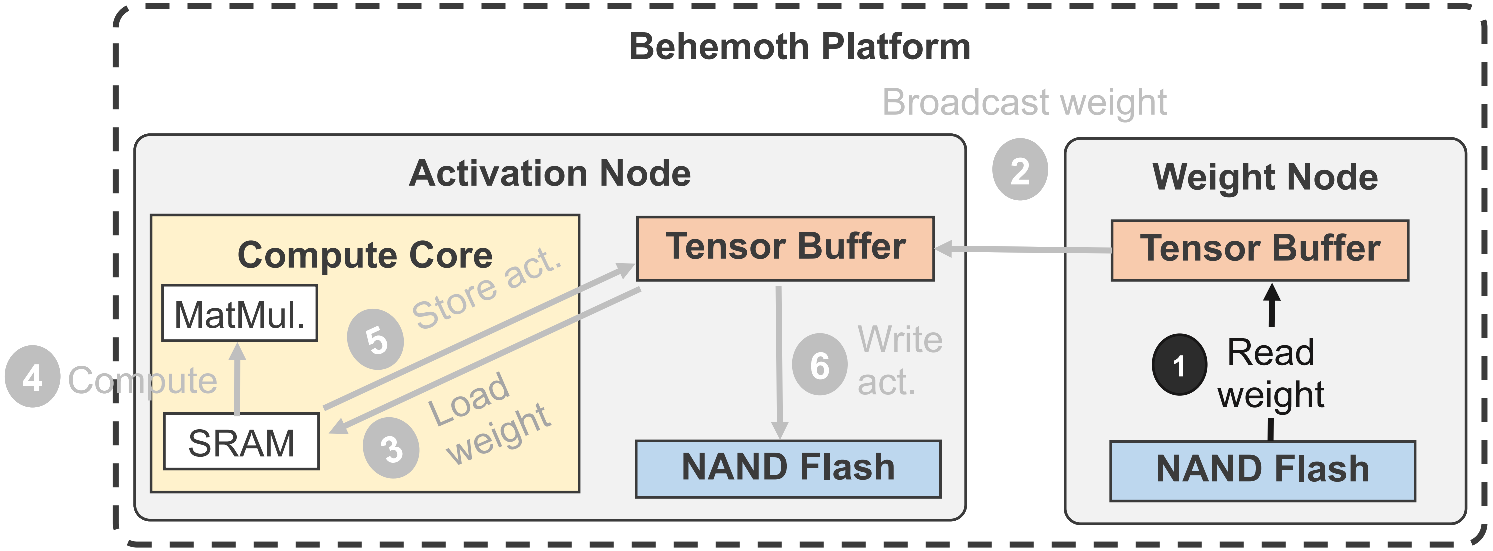
DNN training is a repetitive process of *forward/backward* propagation



- Activation Node: compute and store activations
- Weight Node: update and store weights

Example Execution Walk-Through

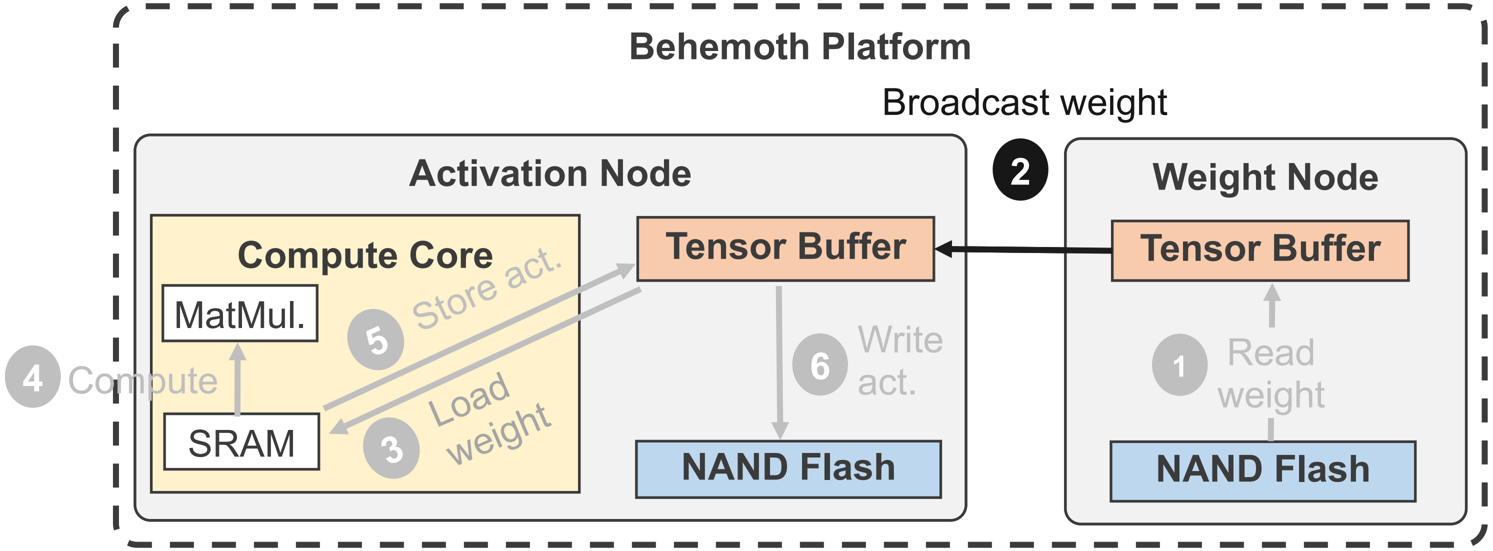
DNN training is a repetitive process of ***forward***/*backward* propagation



- Activation Node: compute and store activations
- Weight Node: update and store weights

Example Execution Walk-Through

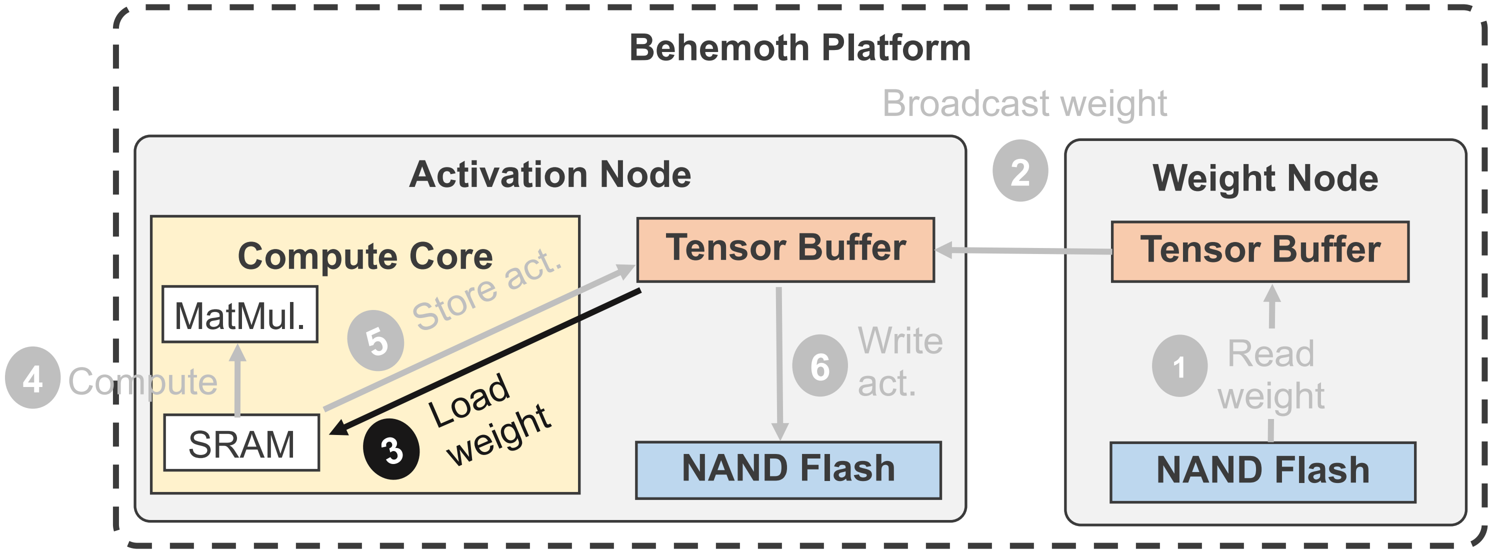
DNN training is a repetitive process of ***forward***/*backward* propagation



- Activation Node: compute and store activations
- Weight Node: update and store weights

Example Execution Walk-Through

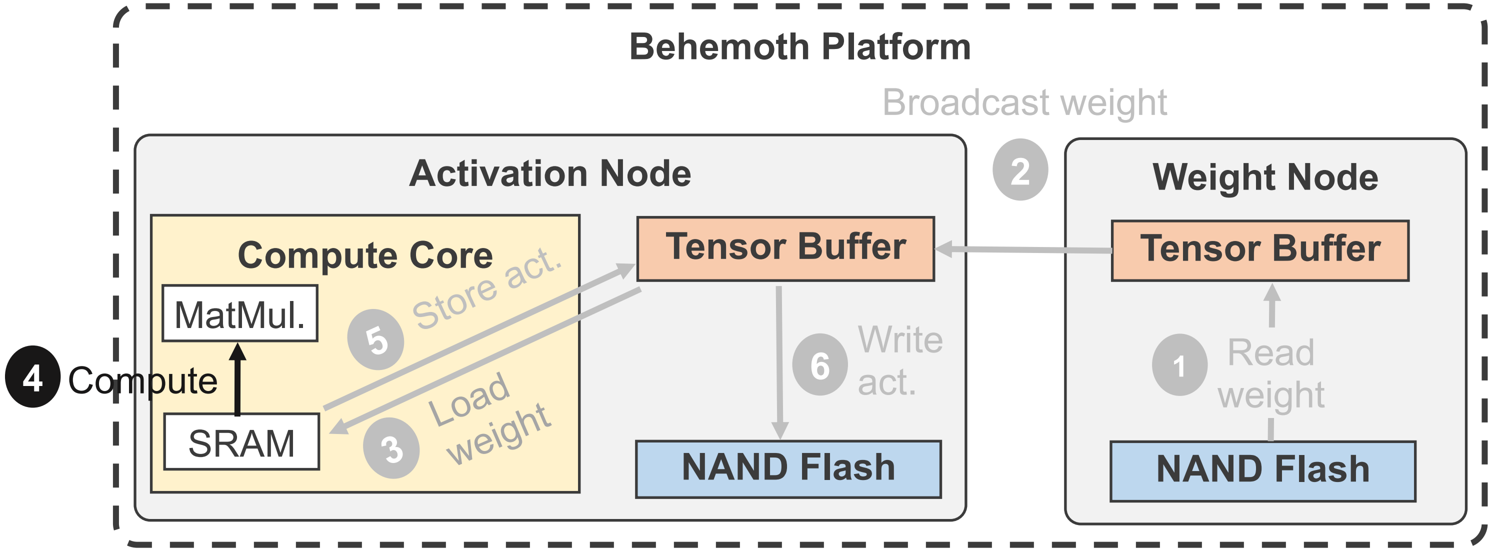
DNN training is a repetitive process of ***forward***/*backward* propagation



- Activation Node: compute and store activations
- Weight Node: update and store weights

Example Execution Walk-Through

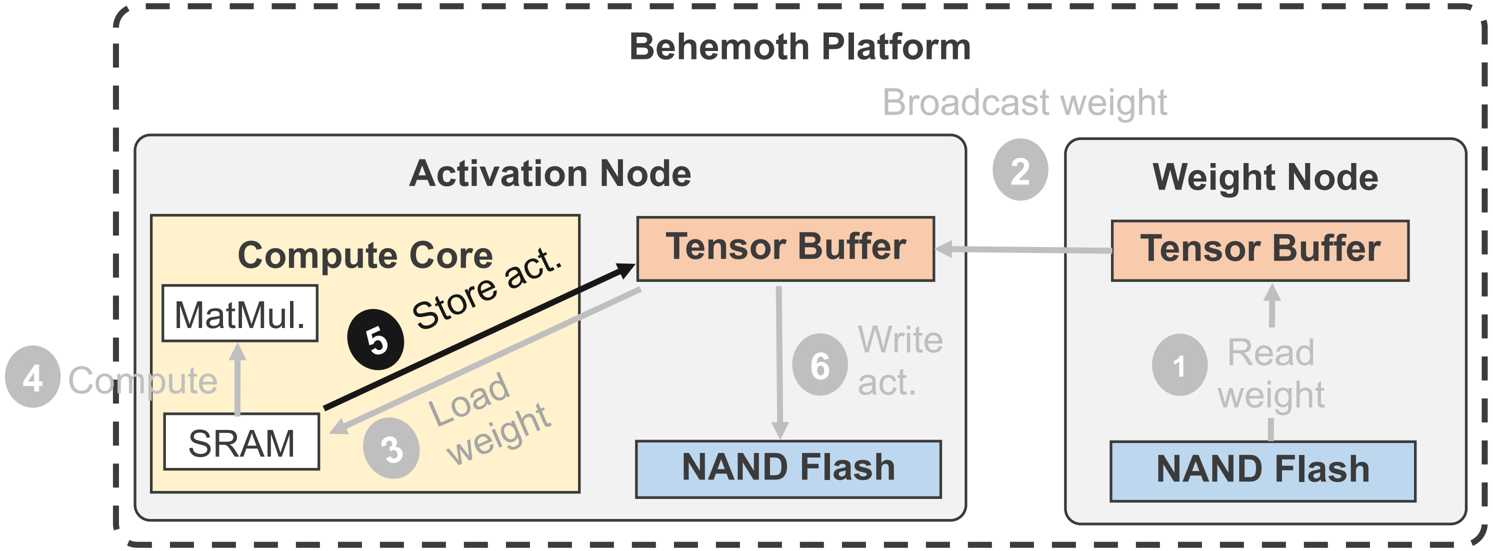
DNN training is a repetitive process of ***forward***/*backward* propagation



- Activation Node: compute and store activations
- Weight Node: update and store weights

Example Execution Walk-Through

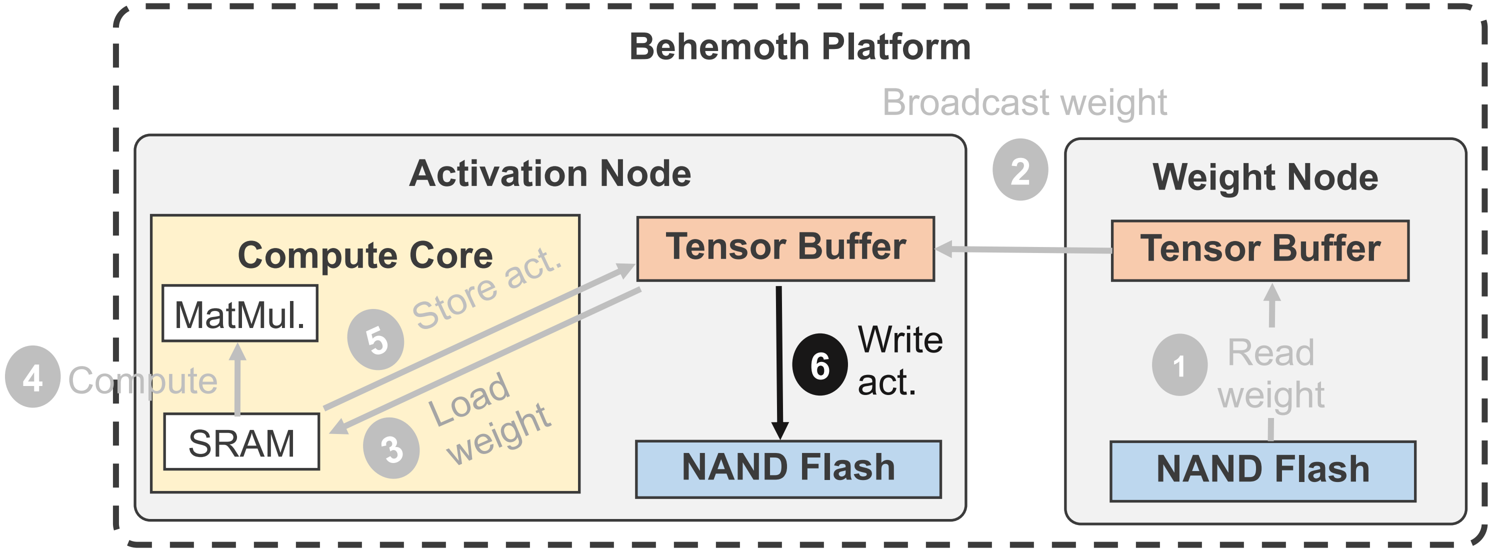
DNN training is a repetitive process of ***forward***/*backward* propagation



- Activation Node: compute and store activations
- Weight Node: update and store weights

Example Execution Walk-Through

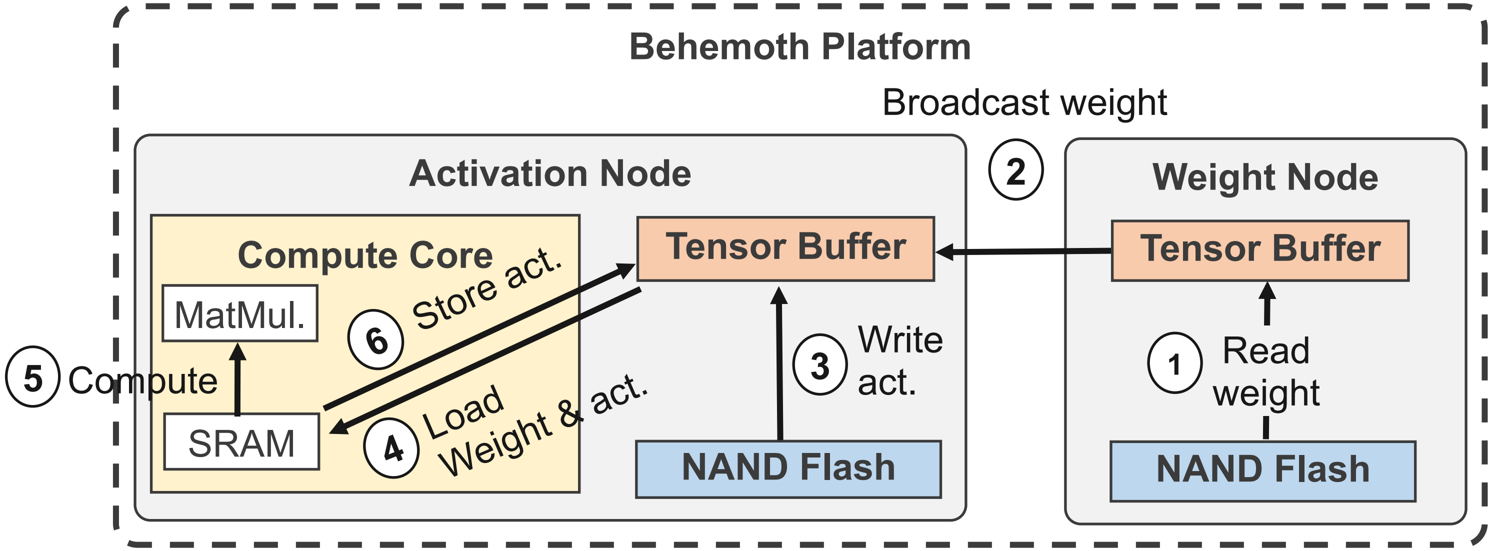
DNN training is a repetitive process of ***forward****/backward* propagation



- Activation Node: compute and store activations
- Weight Node: update and store weights

Example Execution Walk-Through

DNN training is a repetitive process of *forward*/***backward*** propagation



- Activation Node: compute and store activations
- Weight Node: update and store weights

Architecting Specialized FMS for DNN Training

Flash Memory System (FMS) is the main storage in Behemoth to meet the bandwidth and endurance requirements of extreme scale DNN training

Architecting Specialized FMS for DNN Training

Flash Memory System (FMS) is the main storage in Behemoth to meet the bandwidth and endurance requirements of extreme scale DNN training

>50GB/s Read and Write Bandwidth

5-year Endurance

Improving Bandwidth of FMS

- SSD firmware has become a bottleneck for scalable performance

Improving Bandwidth of FMS

- SSD firmware has become a bottleneck for scalable performance
- H/W implemented (automated) data-path can be a solution

Improving Bandwidth of FMS

- SSD firmware has become a bottleneck for scalable performance
- H/W implemented (automated) data-path can be a solution
- Complex functions of FTL make data-path automation difficult
  - Garbage Collection (GC), Wear-leveling (WL), Metadata management for persistency, and so on

Improving Bandwidth of FMS

FMS separates data types and adopts lightweight FTL to implement H/W automated data path

Improving Bandwidth of FMS

FMS separates data types and adopts lightweight FTL to implement H/W automated data path

#: Stream name (Act. Node / Weight Node)	**Persistency**	**Retention**	**Access permission**	
			Host	**Behemoth**
1: NV-Stream (Training inputs / –)	Non-volatile	Years	Append-only seq. write	Read only
2: V-Stream (Activations / Interm. weights)	Volatile	Minutes	N/A	Read & Append-only seq. write
3: NV-Stream (– / Trained weights)	Non-volatile	Years	Read only	Read & Append-only seq. write

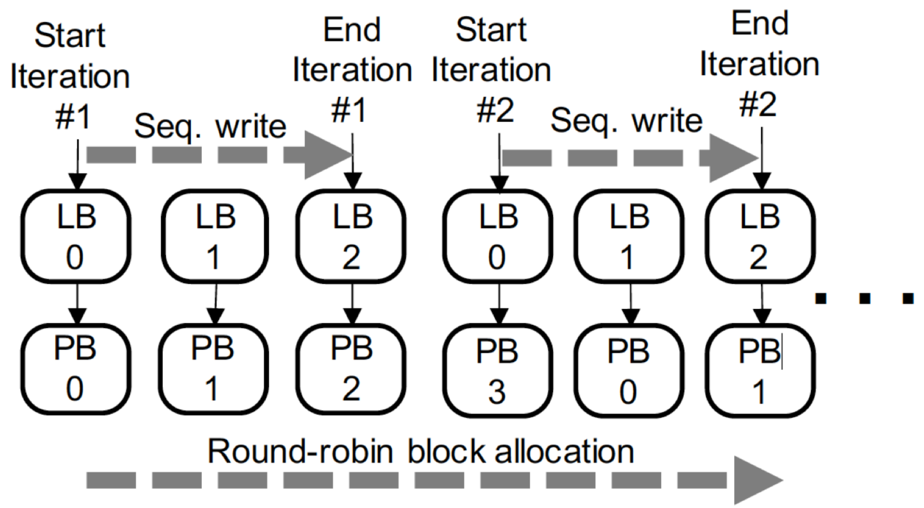
Multi-stream support for data seperation

Improving Bandwidth of FMS

FMS separates data types and adopts lightweight FTL to implement H/W automated data path

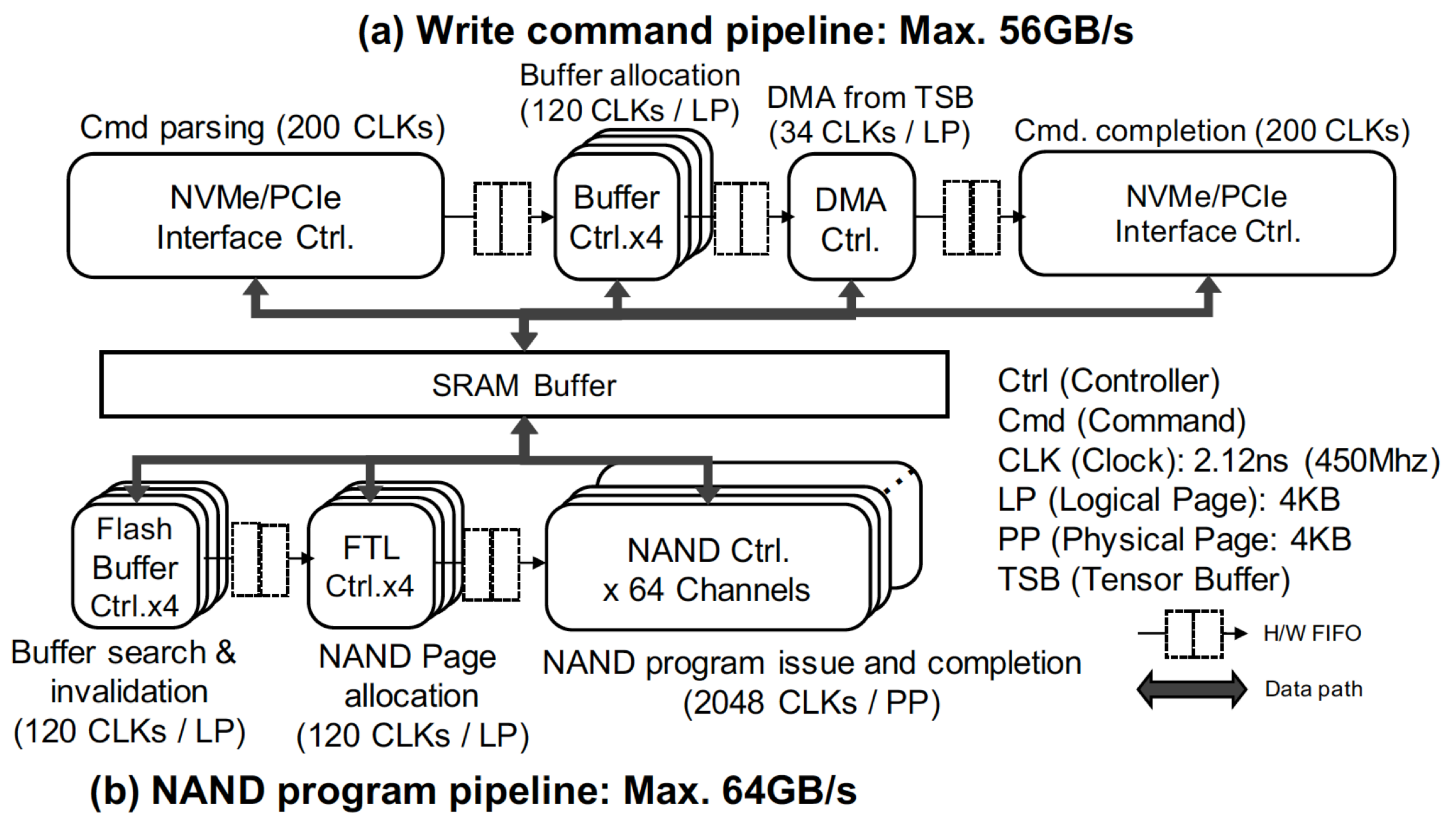
#: Stream name (Act. Node / Weight Node)	Persistency	Retention	Access permission	
			Host	Behemoth
1: NV-Stream (Training inputs / –)	Non-volatile	Years	Append-only seq. write	Read only
2: V-Stream (Activations / Interm. weights)	Volatile	Minutes	N/A	Read & Append-only seq. write
3: NV-Stream (– / Trained weights)	Non-volatile	Years	Read only	Read & Append-only seq. write

Multi-stream support for data seperation



Strict append-only seq. write for lightweight FTL

Improving Bandwidth of FMS



H/W automated write data path of FMS

(a) write command pipeline: transfers data from TSB to an SRAM buffer in the FMS controller

(b) NAND program pipeline: programs data in the SRAM to NANDs

Improving Endurance of FMS

- Endurance of SSD relies on the Program/Erase (P/E) cycles for NAND block

Improving Endurance of FMS

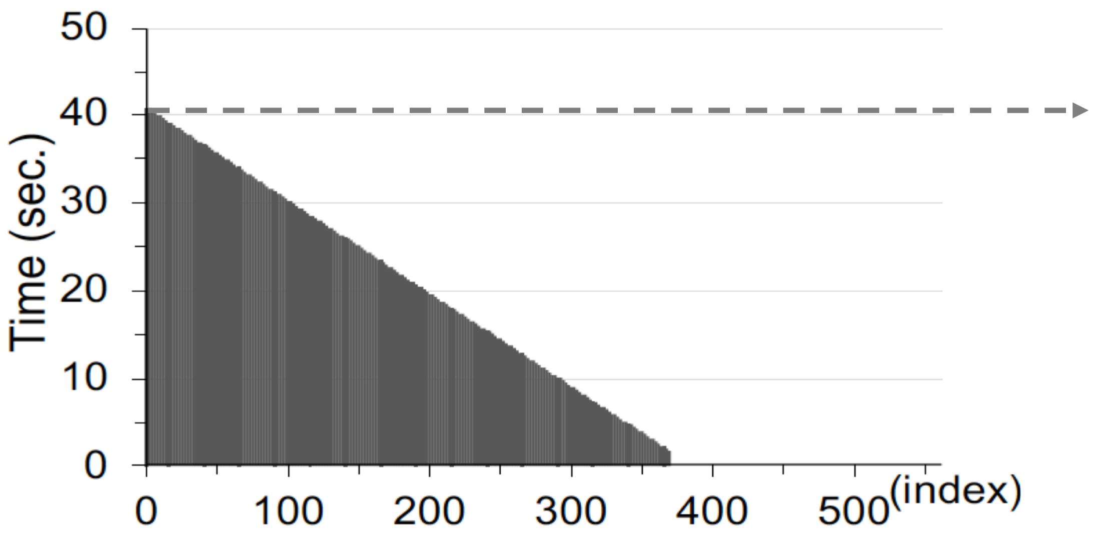
- Endurance of SSD relies on the Program/Erase (P/E) cycles for NAND block
- DNN training workloads cause frequent P/E operation

Improving Endurance of FMS

Behemoth reduces the data retention time
and maintains very low WAF (~1)

Improving Endurance of FMS

Behemoth reduces the data retention time
and maintains very low WAF (~1)



Tensor lifespan for a training iteration of GTP-3 on Behemoth

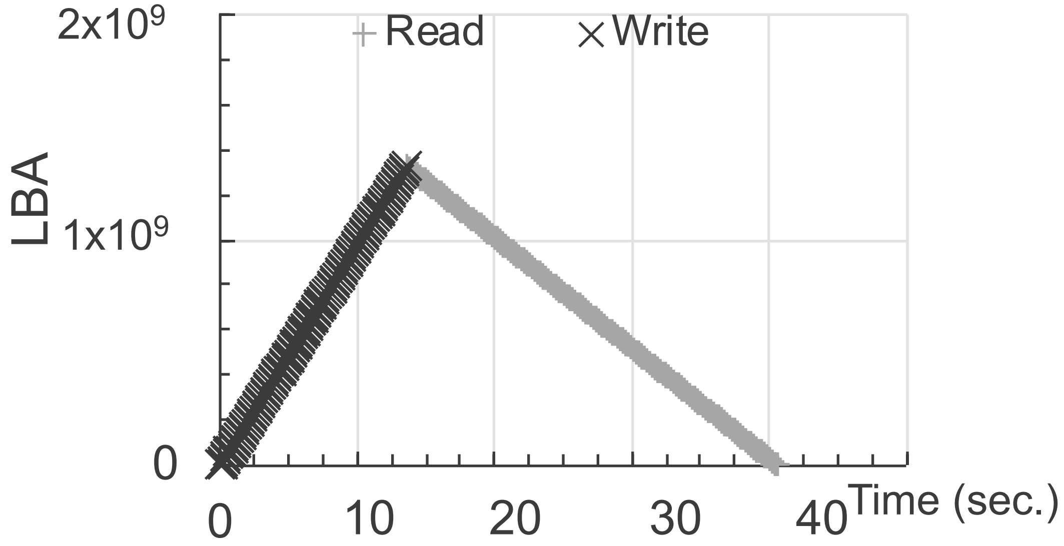
- Max. data lifespan: 41 sec.
- 1 year retention → 3 days
  - P/E cycle can be increased by at least 40x [1, 2]
  - e.g., 50K P/E cycle → 200K P/E cycle

1) *Yu cai et al*, *ICCD'12*, Flash correct-and-refresh: Retention-aware error management for increased flash memory lifetime
2) Ren-Shuo Liu *et al*, *FAST'12*, Optimizing NAND flash-based SSDs via retention relaxation

Improving Endurance of FMS

Behemoth reduces the data retention time
and maintains very low WAF (~1)

- Only performs monotonic sequential ***writes*** and *reads*
  - No garbage collection → WAF 1



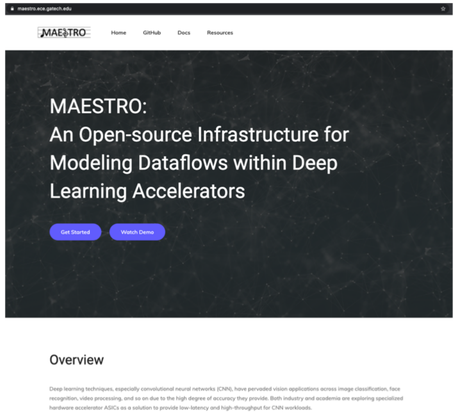
GPT-3 training workloads running on FMS

Evaluation Methodology

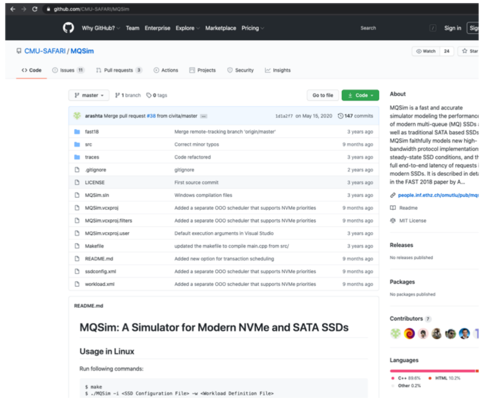
- We evaluate our platform's effectiveness by
  1) Comparing the memory cost of Behemoth against the conventional TPU-based DNN training system
  2) Comparing the training throughput of FMS against conventional SSDs

Evaluation Methodology

- We evaluate our platform's effectiveness by
  1) Comparing the memory cost of Behemoth against the conventional TPU-based DNN training system
  2) Comparing the training throughput of FMS against conventional SSDs



NPU Simulator:
MAESTRO[1)]

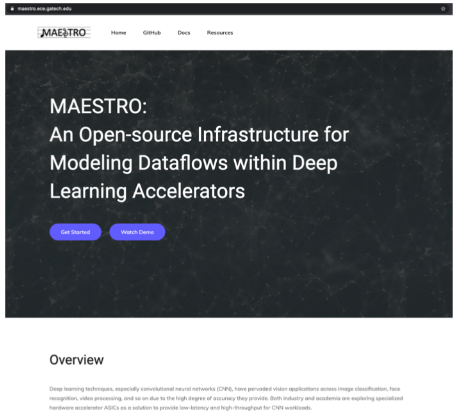


SSD Simulator:
MQ-Sim[2)]

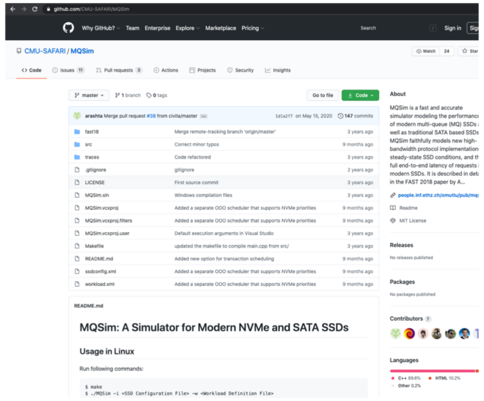
1) https://maestro.ece.gatech.edu/
2) https://github.com/CMU-SAFARI/MQSim

Evaluation Methodology

- We evaluate our platform's effectiveness by
  1) Comparing the memory cost of Behemoth against the conventional TPU-based DNN training system
  2) Comparing the training throughput of FMS against conventional SSDs



NPU Simulator: MAESTRO[1)]



SSD Simulator: MQ-Sim[2)]

Model	Size	Total act. (GB)	Total weight (GB)	PFLOP
BERT/GPT3-like	1×1	44	350	2.15
	1×2	88	698	4.42
	1×4	175	1393	8.56
	2×1	88	1395	8.56
	2×2	175	2786	17.12
	2×4	349	5569	34.21
T5-like	1×1	40	305	0.62
	1×2	80	609	1.25
	1×4	160	1218	2.49
	2×1	80	1218	2.49
	2×2	160	2436	4.99
	2×4	319	4871	9.97

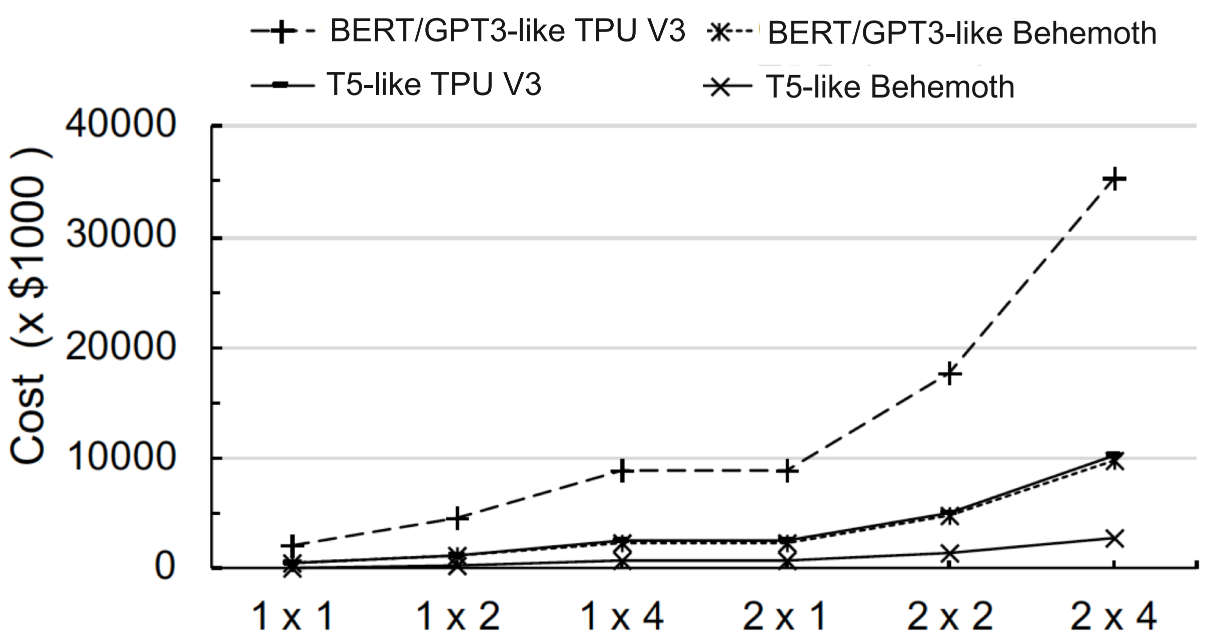
Evaluation workloads

1) *https://maestro.ece.gatech.edu/*
2) *https://github.com/CMU-SAFARI/MQSim*

Memory Cost Evaluation

NPU Parameters		
Number of Cores	16 cores (52.5 TFLOPs per core)	
Number of PEs	524,288	
Peak throughput	840 TFLOPs	
Host I/F conf.	PCIe Gen4 x 32 lane	
Node and Memory Parameters		
	Resemble TPU	**Behemoth**
Number of Nodes	864	432
Buffer conf.	128GB HBM (16GB x 8ea)	16GB DDR4 DRAM + 2TB NAND flash
Peak bandwidth	4.8TB/s	50GB/s
Compute Parameters		
Parallel comp. method	Model parallelism	Data parallelism

Platform configurations for the cost comparison



Memory cost[1)] comparison between TPU V3 and Behemoth

- To maintain the same training throughput, TPUv3-like platform costs up to 3.65x the memory cost

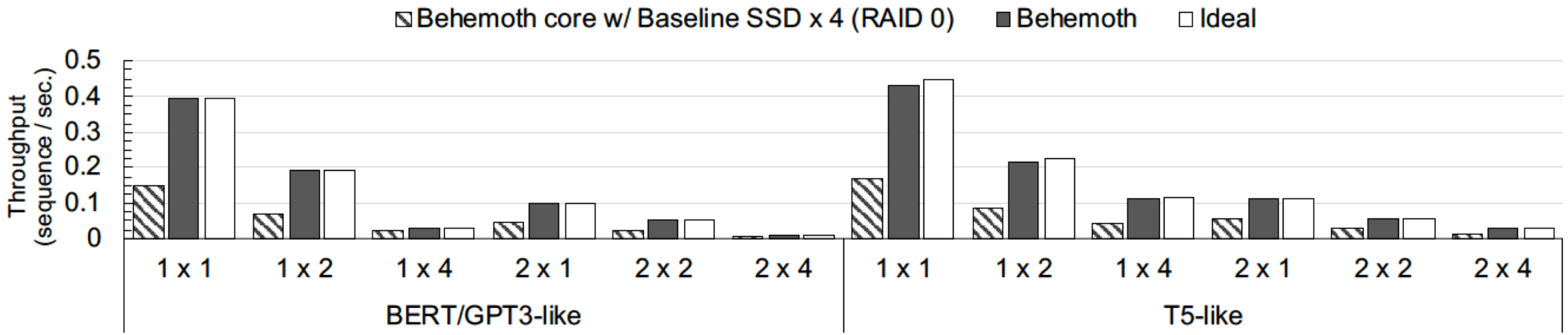
1) HBM: $20/GB, SLC NAND: $0.67/GB, DDR DRAM: $4/GB

Training Throughput Evaluation

- We compare Behemoth and baseline system utilizing the commodity SSDs
  - Behemoth with 2TB FMS
  - Behemoth core with 500GB of 4x SSDs (RAID 0)

Storage Parameters		
	Behemoth FMS	**Baseline SSD**
NAND Configurations	2TB, 64 channels, 2 chips/channel, 1 die/chip	500GB, 16 channels, 2 chips/channel, 1 die/chip
Channel Speed Rate	1200MT/s (MT/s: Mega Transfers per Second [20])	
NAND Structure	128Gb SLC / die: 8 planes / die, 683 blocks / plane, 768 pages / block, 4KB page	
NAND Latency	Read: 3μs, Program: 100μs, Block erase: 5ms	
Buffer Configurations	SRAM 16MB: 6MB for FTL metadata, 10MB for I/O buffer	DRAM 512GB: FTL metadata SRAM 8MB: I/O buffer, GC Buffer
FTL Schemes	Block mapping	Page mapping, Preemtible GC [38]
OP ratio	N/A	7%
Firmware Latency	N/A	Write: 1.45μs / a page (4KB)
Contoller Latency	Read: 1.93μs / an NVMe Cmd, Write: 1.18μs / an NVMe Cmd	Read: 1.93μs / an NVMe Cmd

Training Throughput Evaluation



- Behemoth is close to the ideal case
- Conventional SSDs show much lower training throughput (up-to 2.05x)
  - SSD firmware bottleneck is major cause for performance degradation

Summary

Behemoth enables efficient data-parallel training of extreme-scale DNN models

- Analyze the memory capacity problem for extreme-scale DNN model training
- Identify new opportunities to leverage NAND flash devices to hold those models
- Present a novel flash-centric DNN training accelerator
- Show 3.65x memory cost savings over TPUv3 and 2.05x training throughput impr ovement over conventional SSDs

Thank you !

Additional details in the paper:

- Analysis of Transformer: a key enabling primitive for extreme-scale DNNs
- Discussion of architectural decisions
- Coverage analysis for various DNN models
- Endurance evaluation