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

FlashNeuron: SSD-Enabled Large-Batch Training of Very Deep Neural Networks

Jonghyun Bae[1] Jongsung Lee[1,2] Yunho Jin[1] Sam Son[1] Shine Kim[1,2]
Hakbeom Jang[2] Tae Jun Ham[1] Jae W. Lee[1]

[1] Seoul National University [2] Samsung Electronics

Rise of DNNs

- **DNNs are the key enabler of today's AI application**

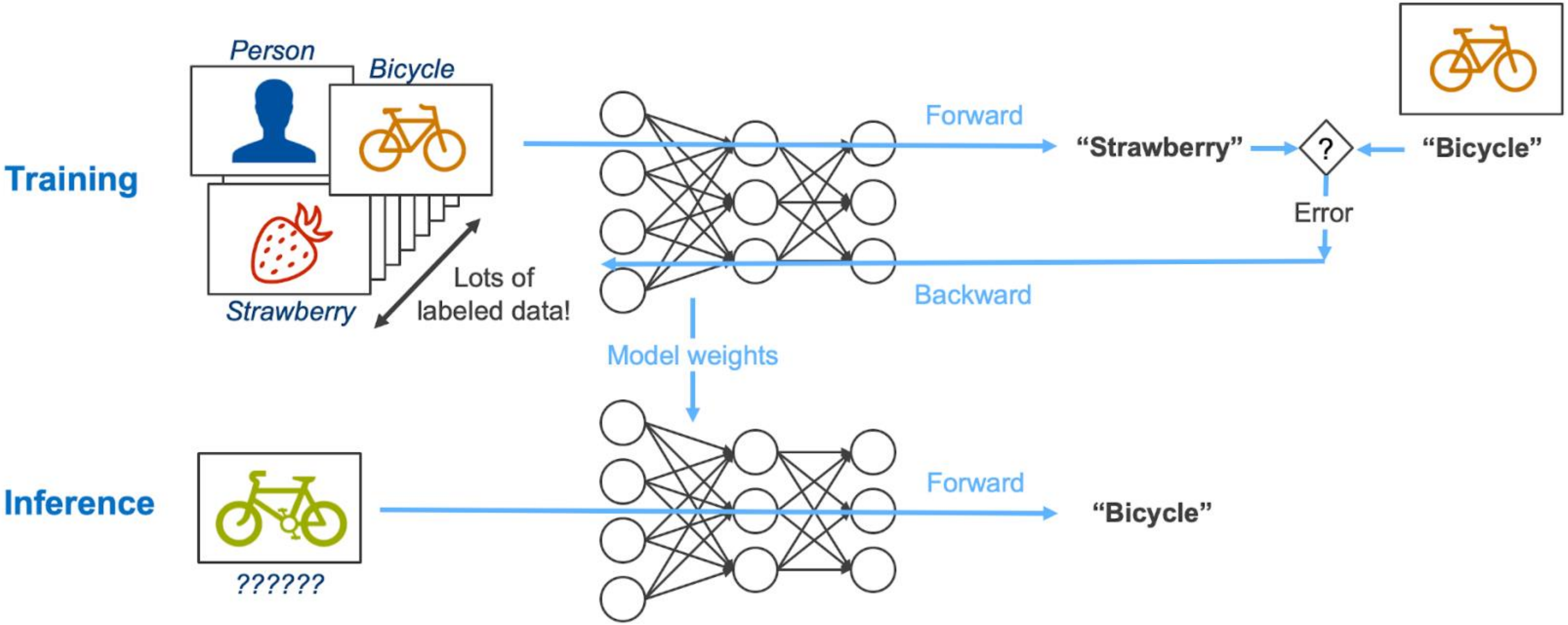
Object detection and classification [1]

Speech-to-text [2]

[1] Joseph Redmon et al., "You Only Look Once: Unified, Real-Time Object Detection," in *CVPR* 2016
[2] Image from https://nordicapis.com/5-best-speech-to-text-apis/

Rise of DNNs

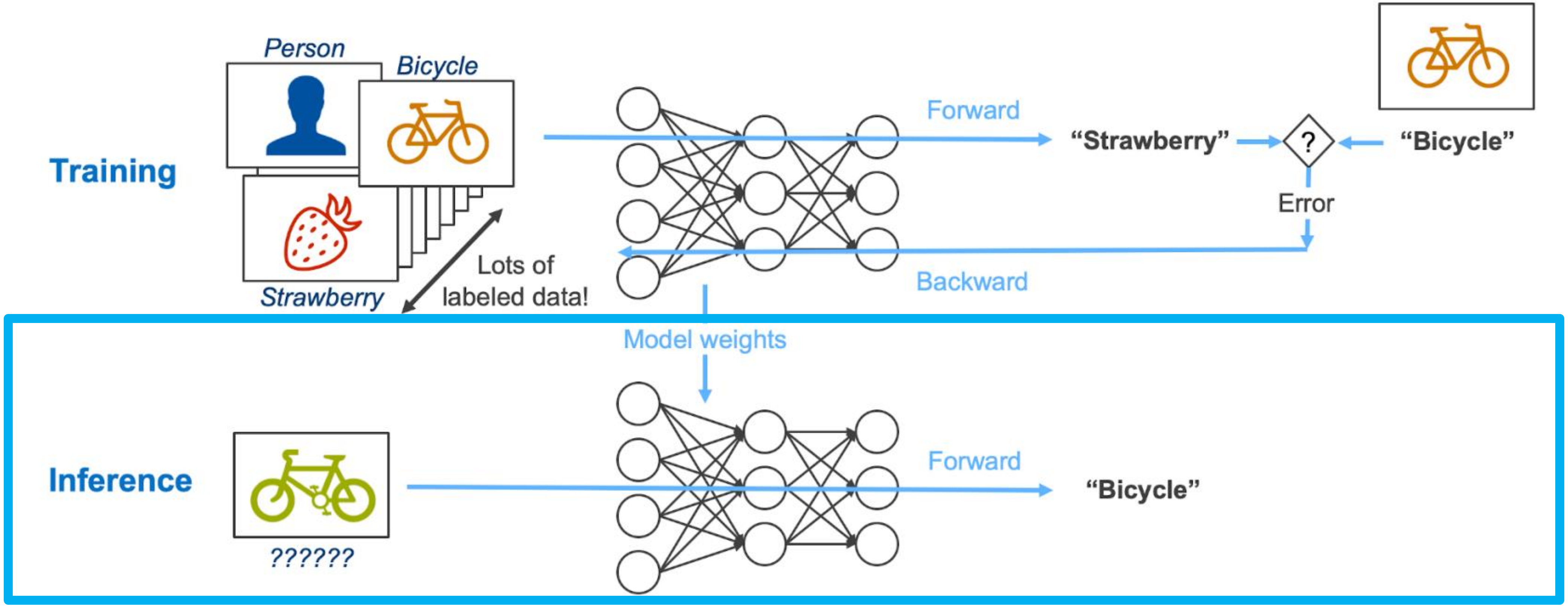
- **DNNs are the key enabler of today's AI application**
- **Two types of DNN workloads: Training >> Inference**
  - **3x** the computation: forward propagation, backward propagation, and weight update



***** Image from https://www.intel.com/content/www/us/en/artificial-intelligence/posts/deep-learning-training-and-inference.html

Rise of DNNs

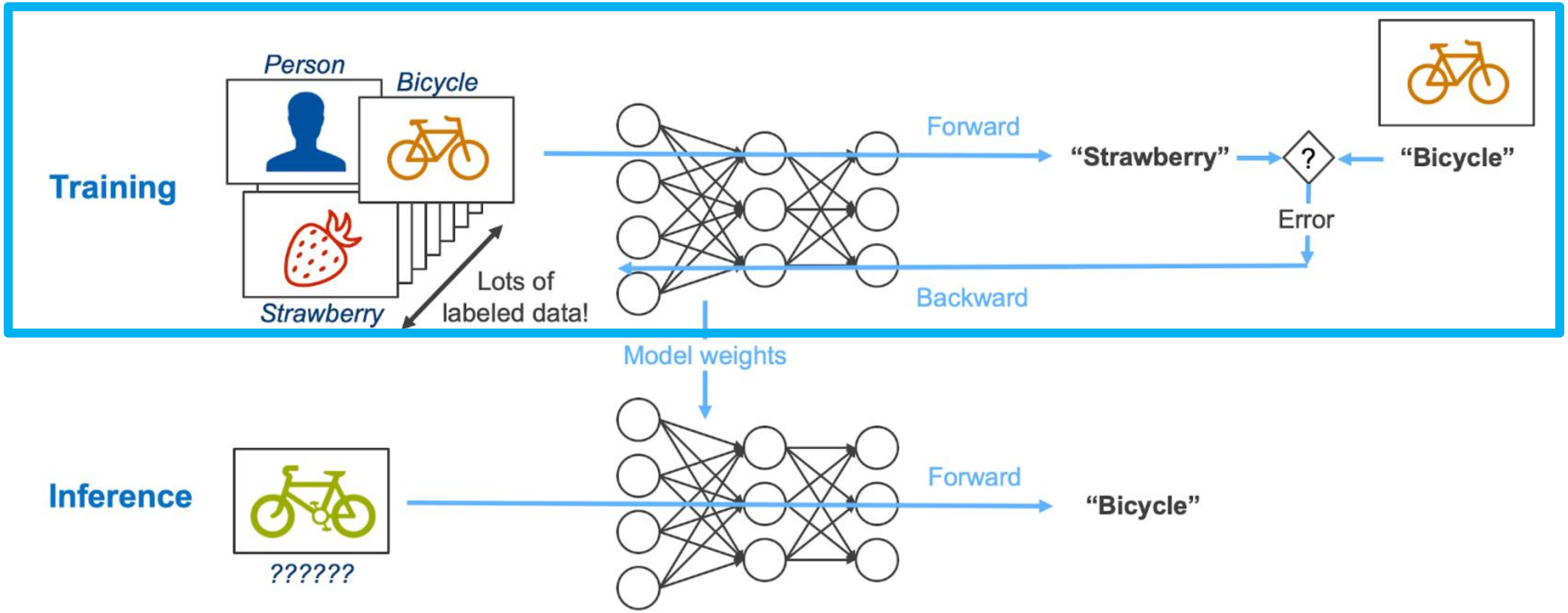
- **DNNs are the key enabler of today's AI application**
- **Two types of DNN workloads: Training >> Inference**
  - **3x** the computation: forward propagation, backward propagation, and weight update



* Image from https://www.intel.com/content/www/us/en/artificial-intelligence/posts/deep-learning-training-and-inference.html

Rise of DNNs

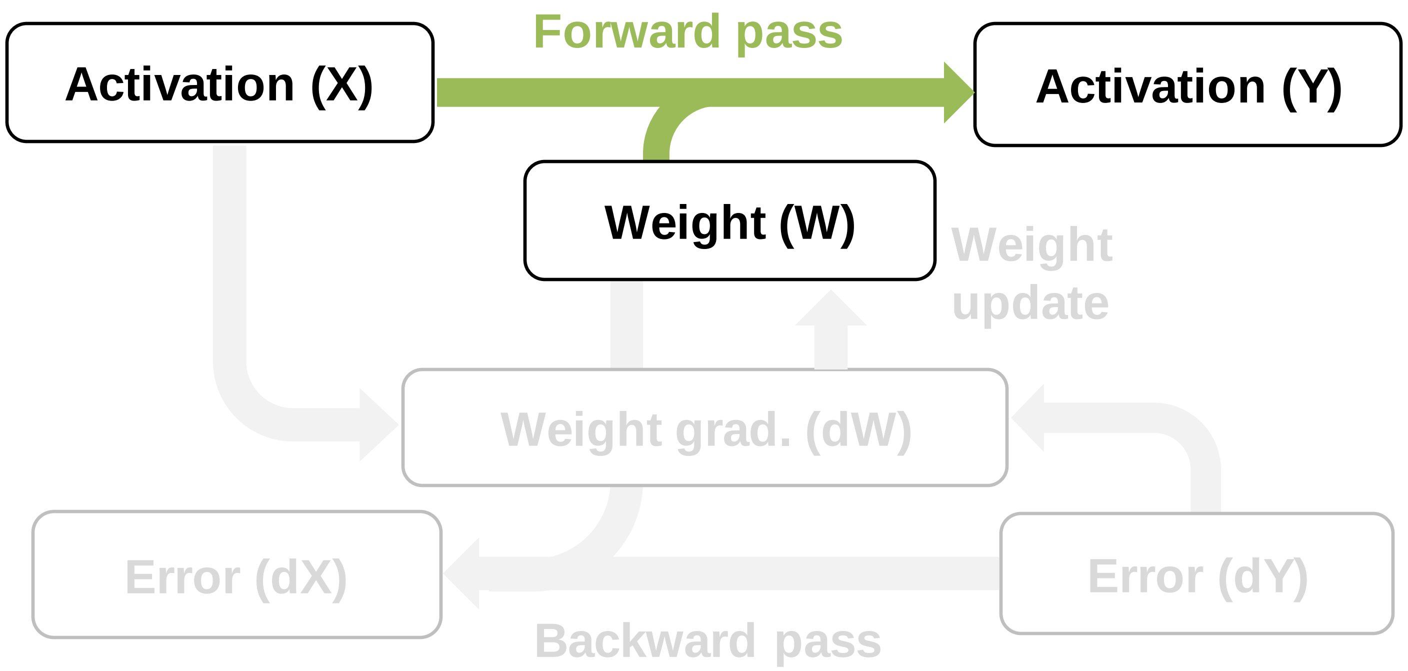
- **DNNs are the key enabler of today's AI application**
- **Two types of DNN workloads: Training >> Inference**
  - **3x** the computation: forward propagation, backward propagation, and weight update



* Image from https://www.intel.com/content/www/us/en/artificial-intelligence/posts/deep-learning-training-and-inference.html

Data Reuse in DNN Training

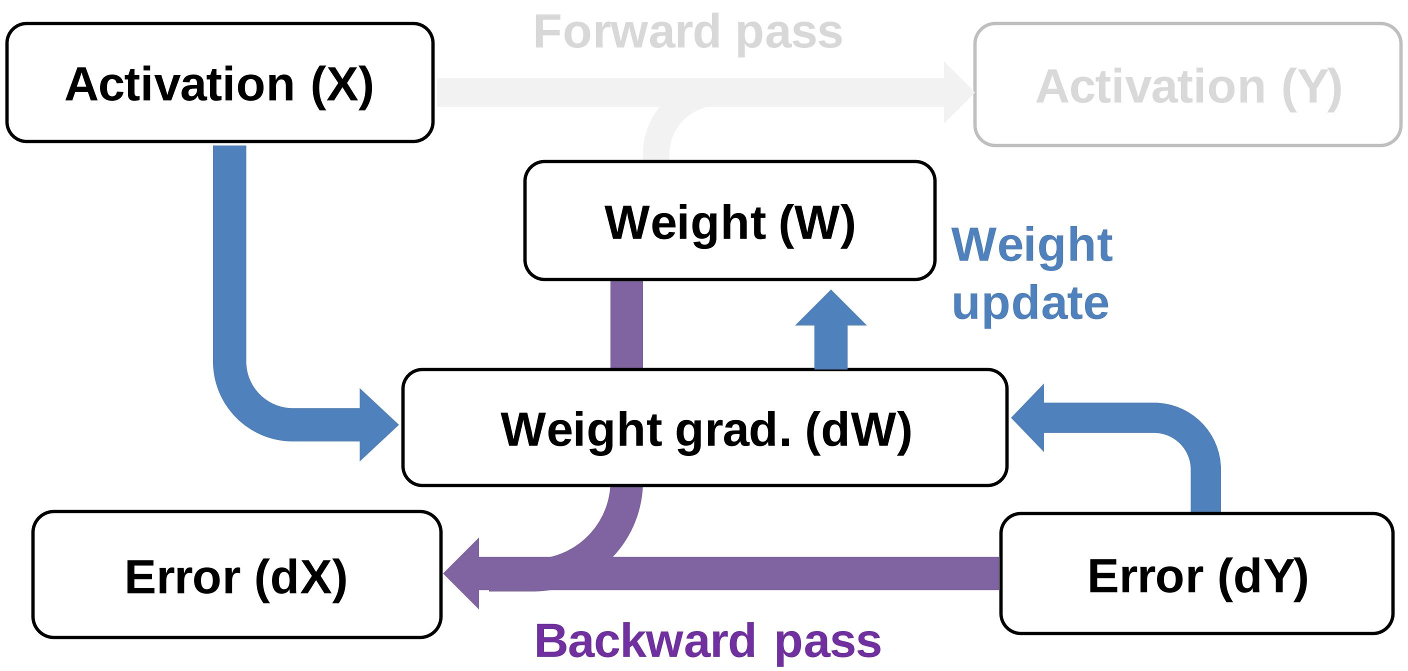
- **Data reuse pattern from forward propagation to backward propagation**
  - Requiring input activation (**X**), and output error (**dY**) to calculate input gradient map (**dX**), weight gradient (**dW**), and finally weight (**W**)



Simplified data reuse pattern in a layer

Data Reuse in DNN Training

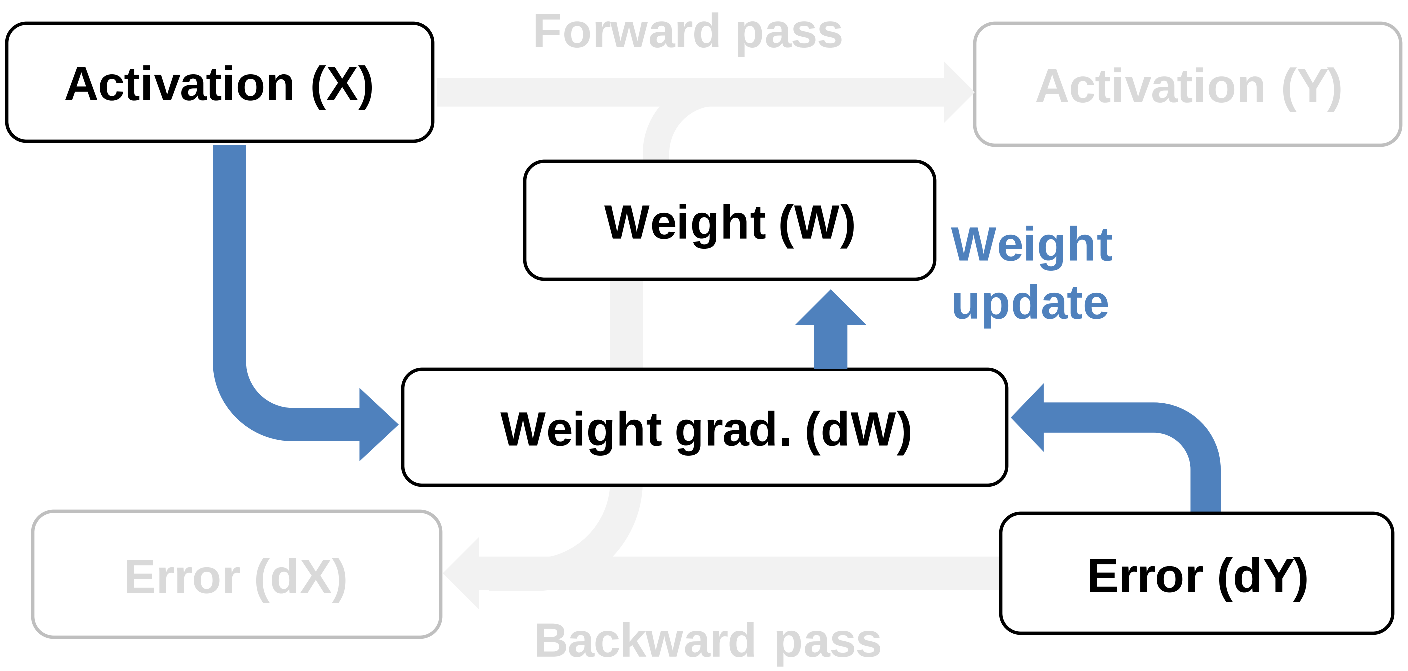
- **Data reuse pattern from forward propagation to backward propagation**
  - Requiring input activation (**X**), and output error (**dY**) to calculate input gradient map (**dX**), weight gradient (**dW**), and finally weight (**W**)



Simplified data reuse pattern in a layer

Data Reuse in DNN Training

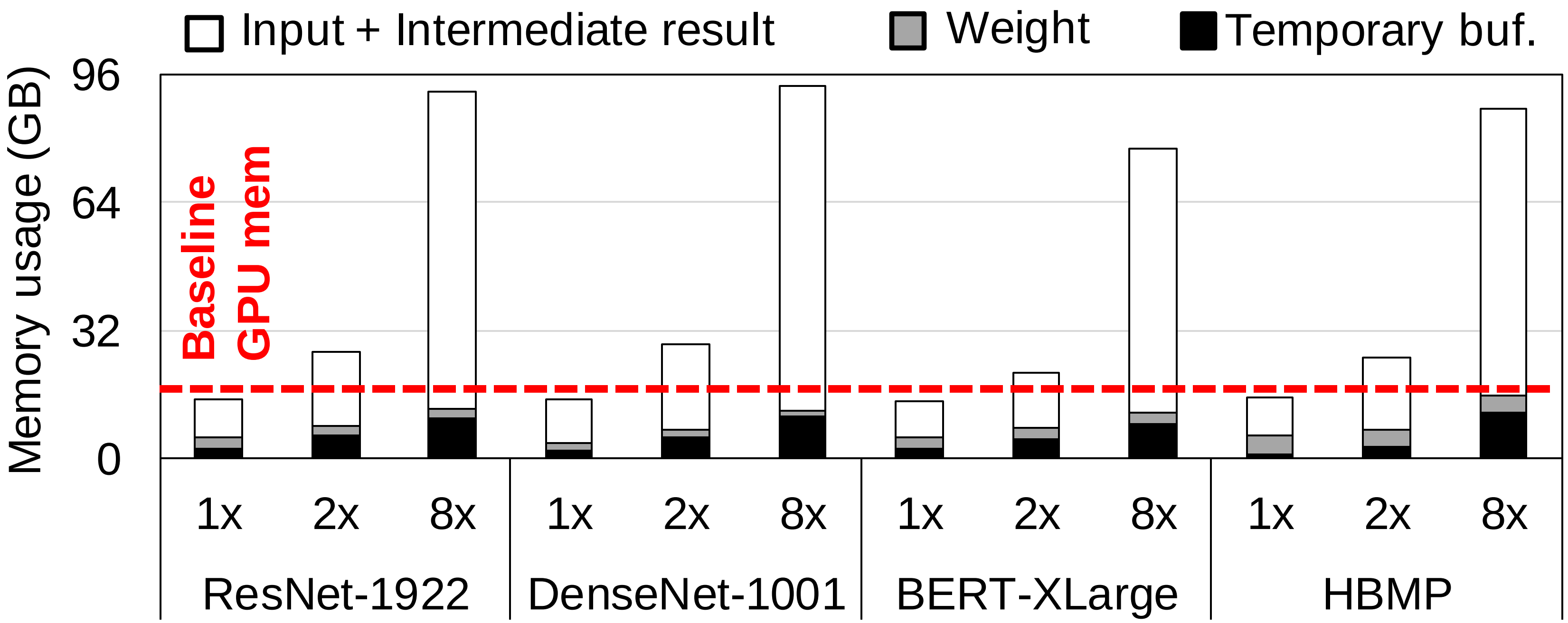
- **Data reuse pattern from forward propagation to backward propagation**
  - Requiring input activation (**X**), and output error (**dY**) to calculate input gradient map (**dX**), weight gradient (**dW**), and finally weight (**W**)



Simplified data reuse pattern in a layer

Memory Capacity Wall in DNN Training

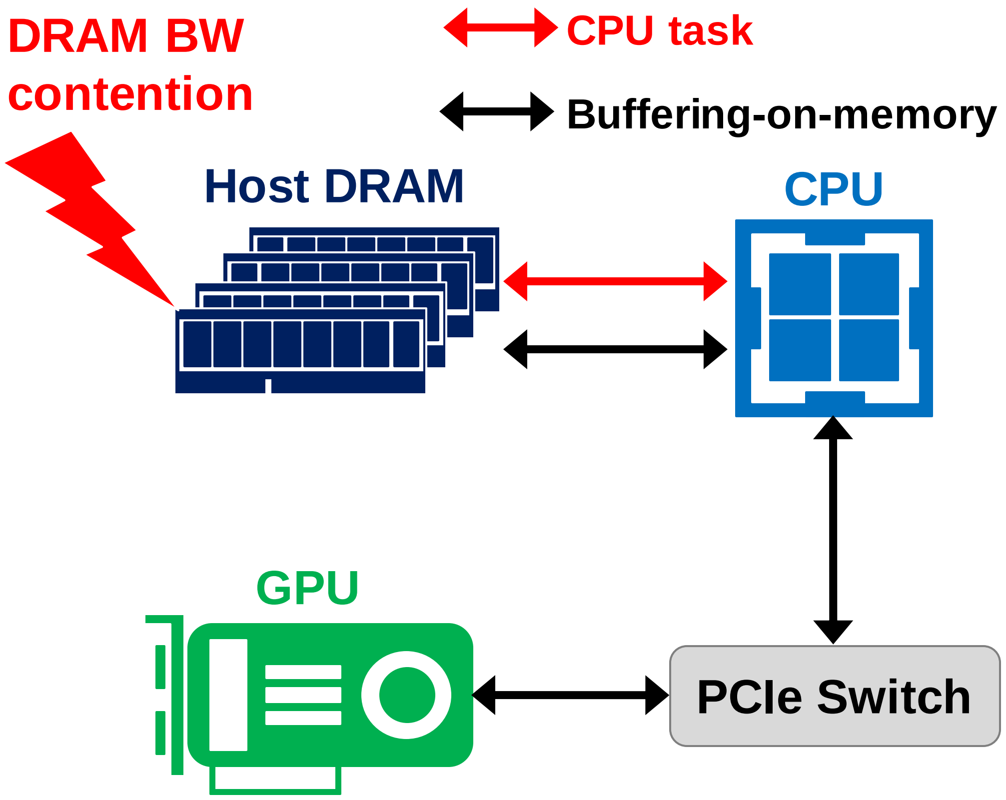
- **DRAM footprint increases with (1) deeper neural nets (for accuracy) and (2) larger batch size (for training throughput)**

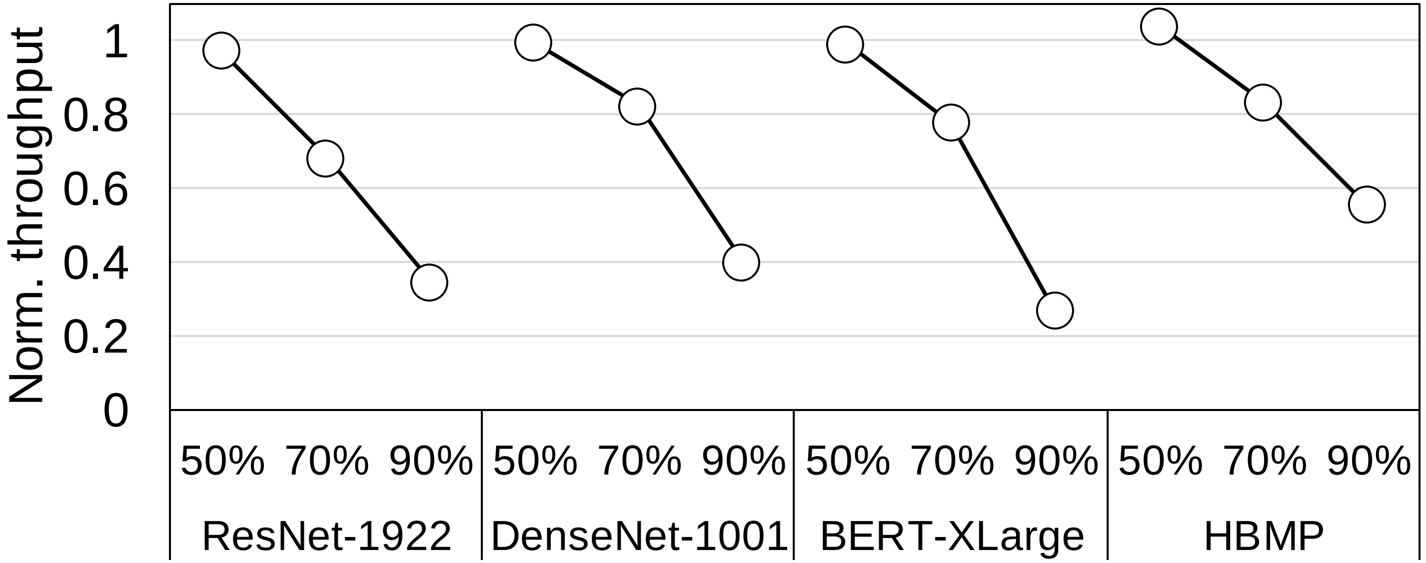


GPU memory usage for DNN training

Overcoming GPU Memory Capacity Wall

- **Previous approach: Buffering-on-memory**
  - Host DRAM BW contention by BW-intensive task on CPU (e.g., data augmentation)



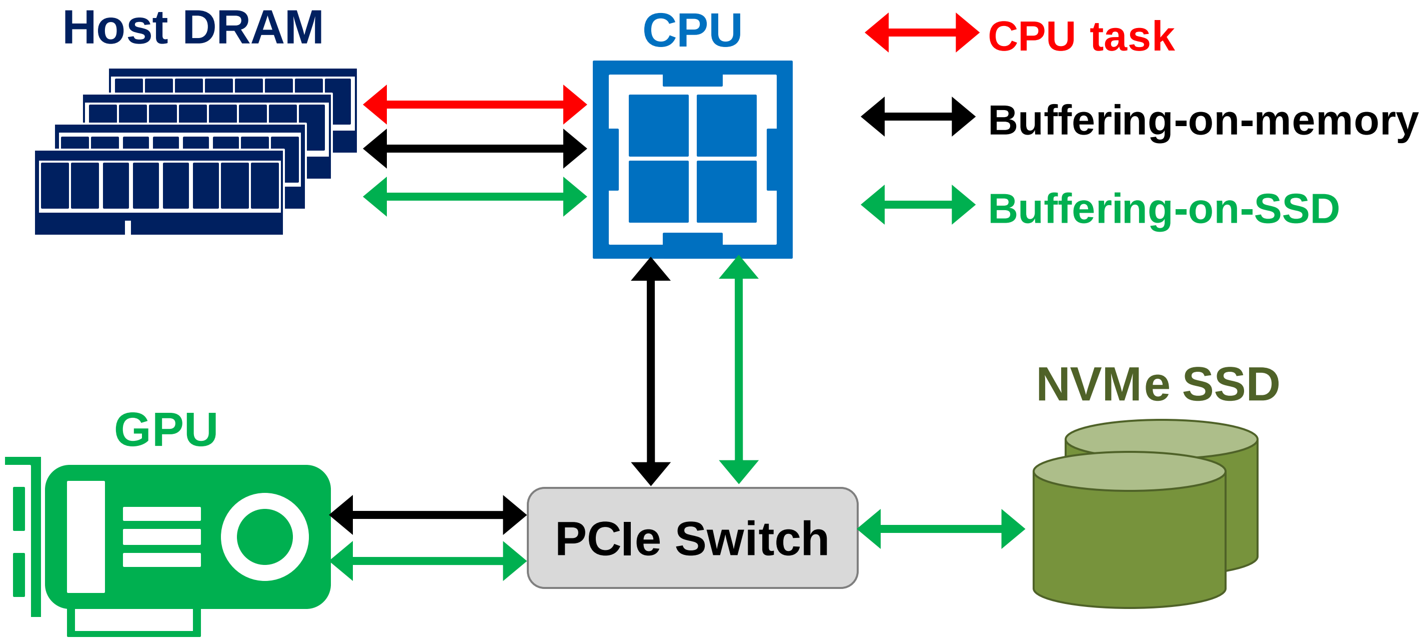


Normalized throughput of buffering-on-memory with bandwidth-intensive tasks on CPU

Overcoming GPU Memory Capacity Wall

- **Previous approach: Buffering-on-memory**
  - Host DRAM BW contention by BW-intensive task on CPU (e.g., data augmentation)

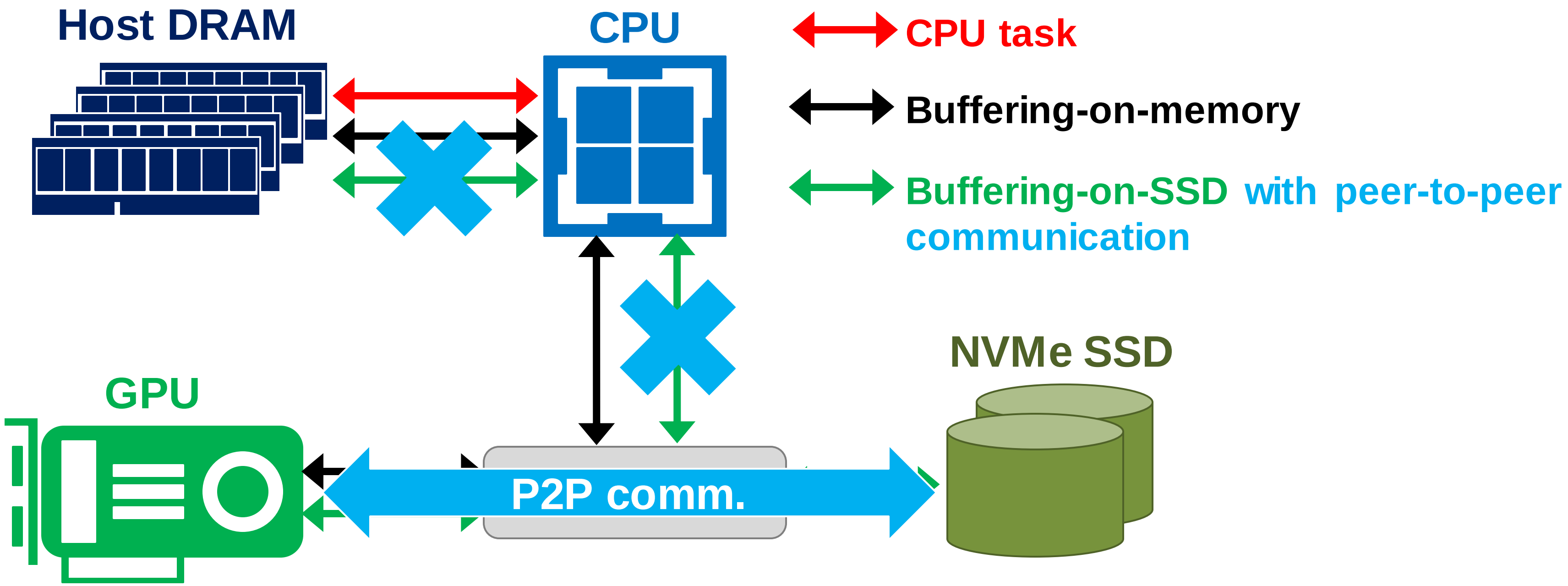
- **New solution: Buffering-on-SSD**



Overcoming GPU Memory Capacity Wall

- **Previous approach: Buffering-on-memory**
  - Host DRAM BW contention by BW-intensive task on CPU (e.g., data augmentation)

- **New solution: Buffering-on-SSD**
  - With peer-to-peer communication, no host DRAM bandwidth or CPU cycles consumed

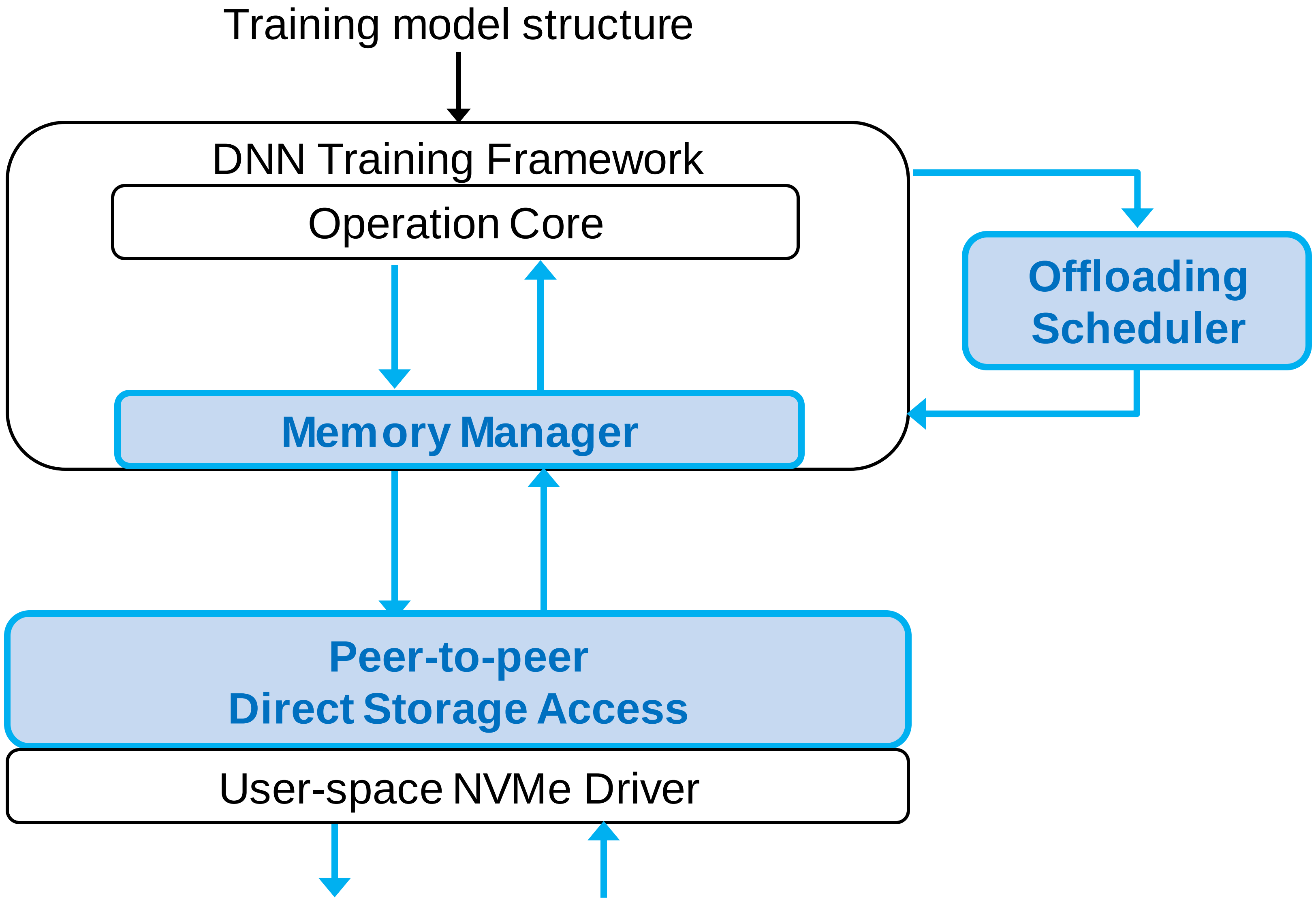


Our Proposal: FlashNeuron

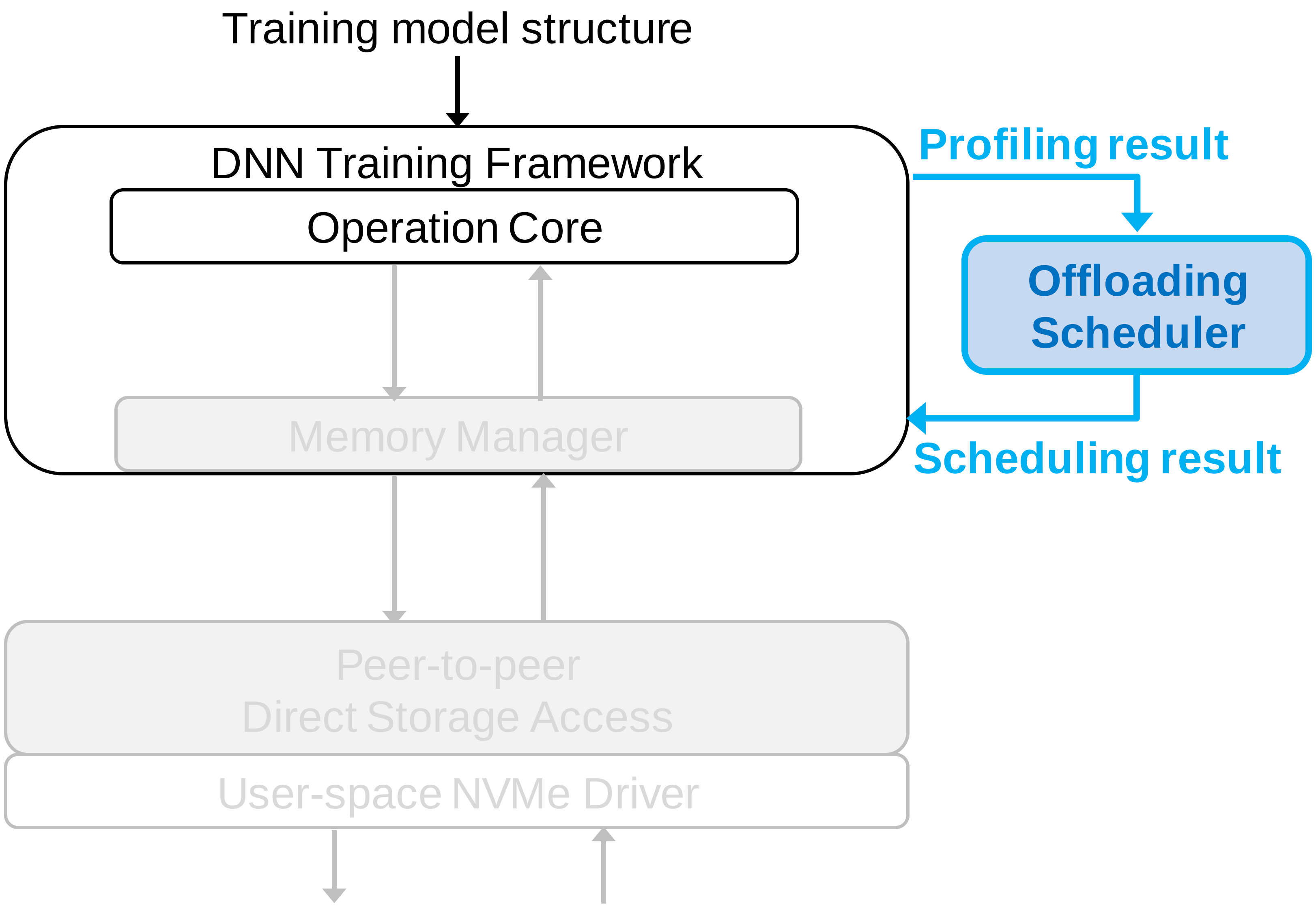
- **Key idea: DNN training using a high-performance SSD as a backing store**
  - **Offloading scheduler**: Identify a set of tensors to offload and generates an offloading schedule
  - **Memory manager**: Manage offloading/prefetching and tensor allocation/deallocation
  - **Lightweight user-level I/O stack**: Customized stack for p2p communication

- **Key results**
  - Batch size: **12.4x** to **14.0x** over the maximum allowable batch size on 16GB HBM
  - Training throughput improvement: Up to **37.8%** (**30.3%** on average) over the baseline
  - Cost efficiency: **35.3x** higher cost efficiency assuming the same capacity of DRAM and SSD

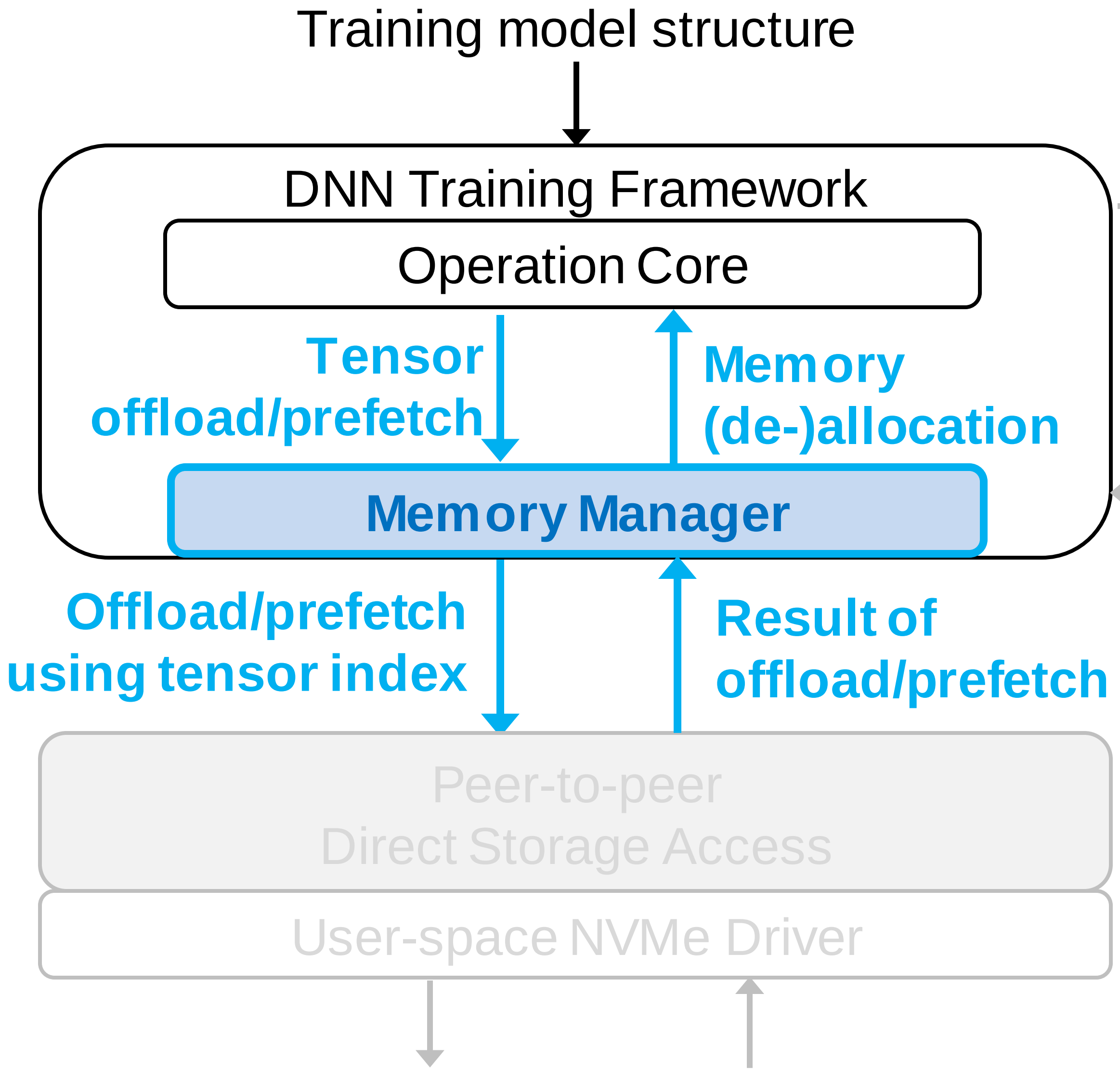
System Overview



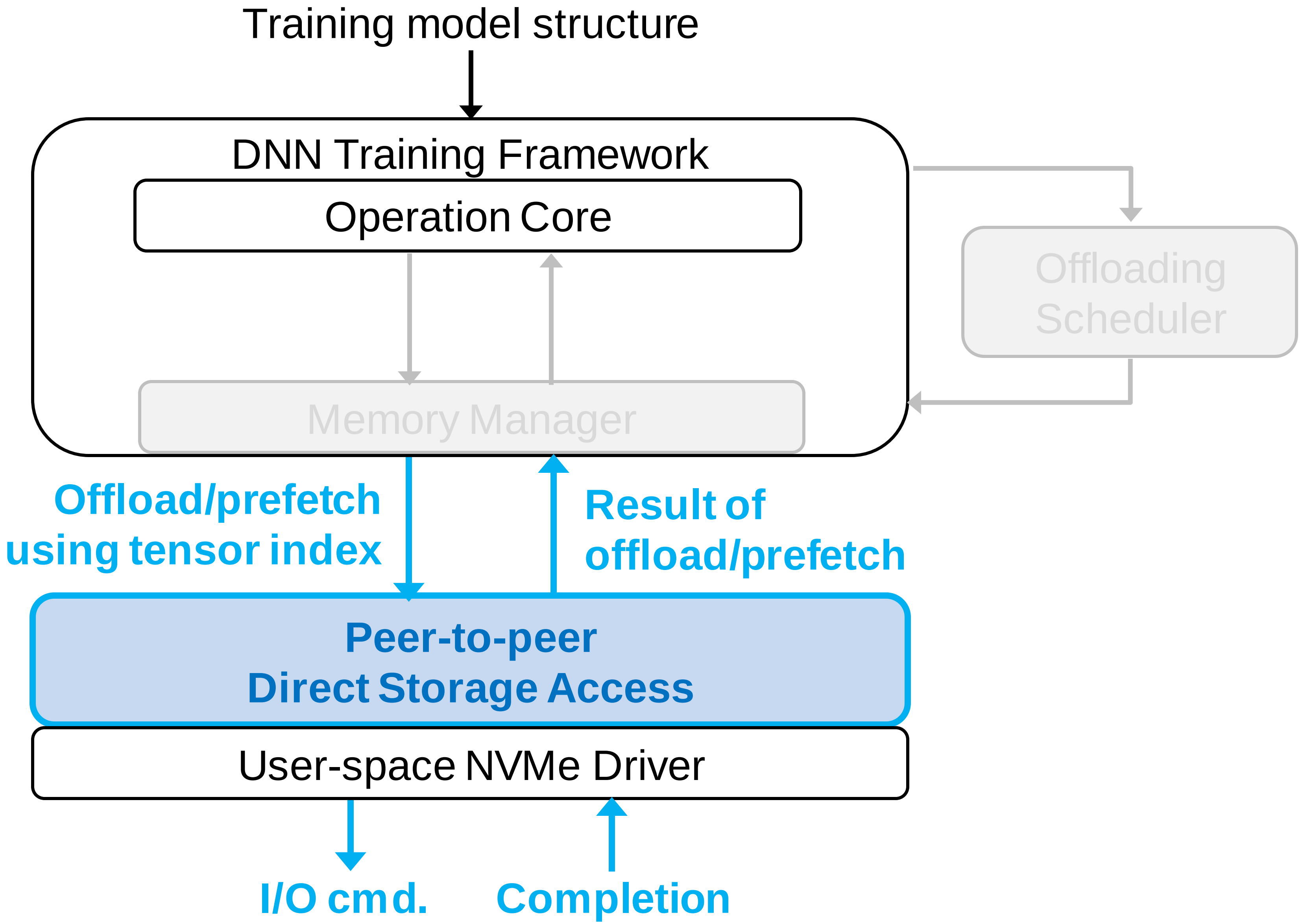
System Overview



System Overview

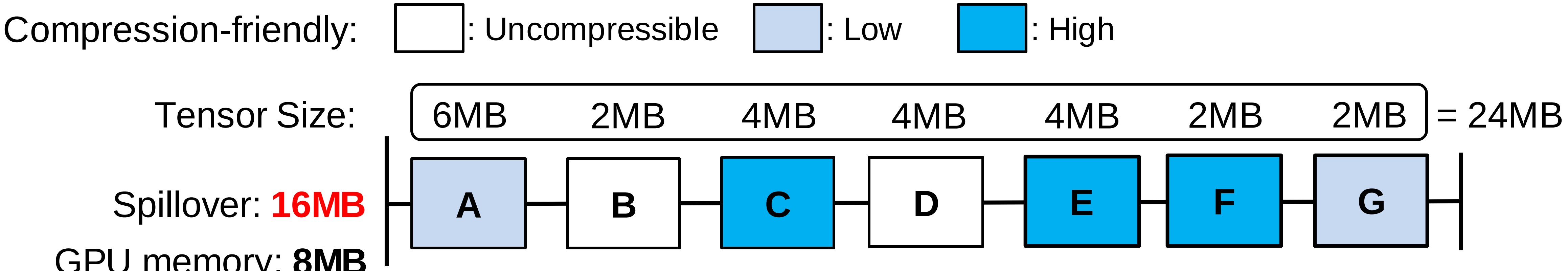


System Overview



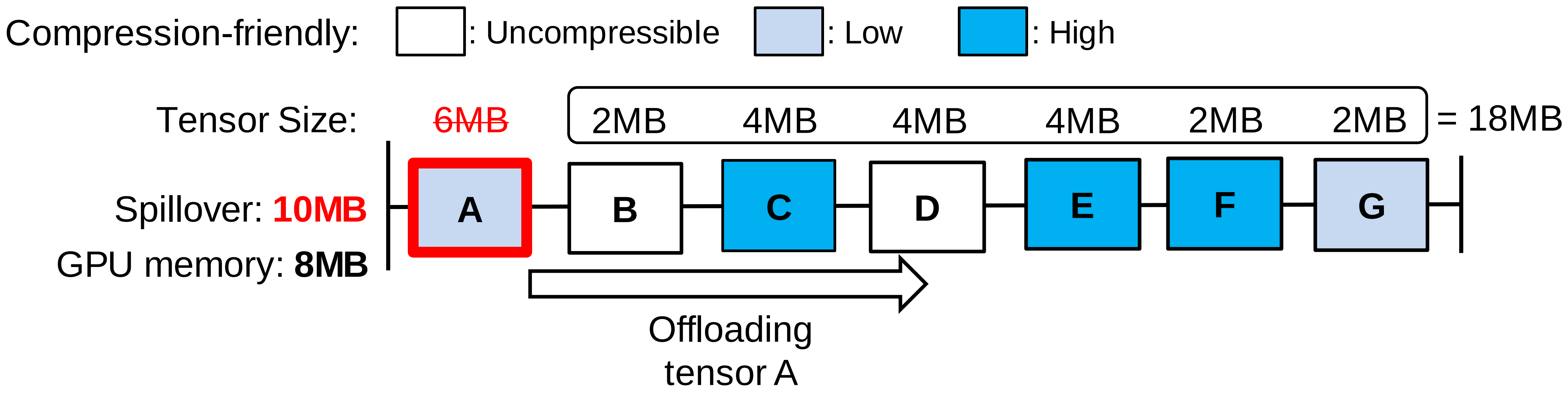
Offloading Scheduler: Phase 1

- **Finding an optimal scheduler for a given target batch size**
- **Phase 1**
  - **Iteratively select a certain number of tensors from the beginning**



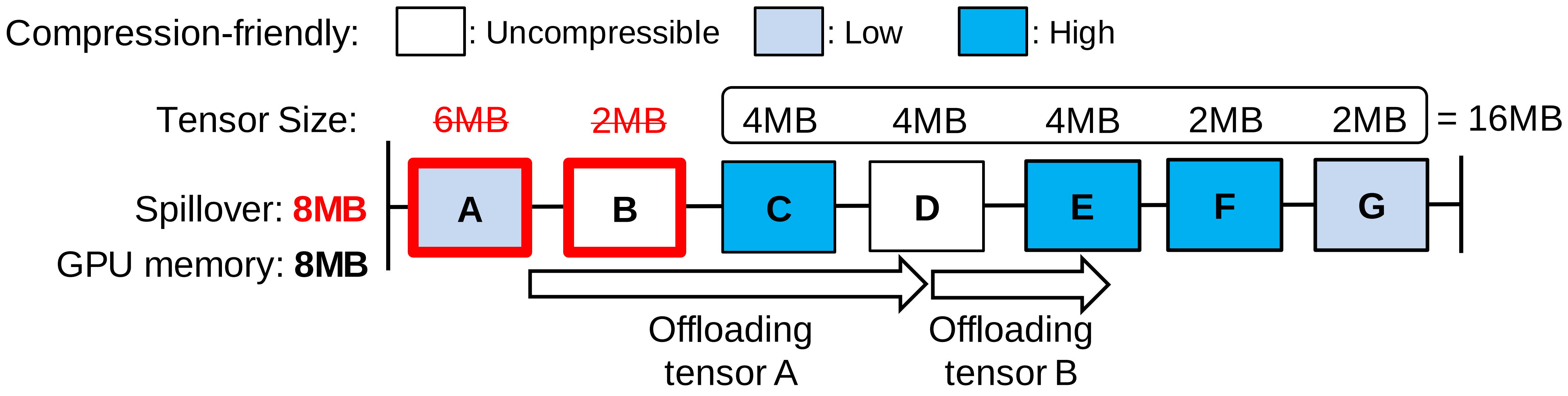
Offloading Scheduler: Phase 1

- **Finding an optimal scheduler for a given target batch size**
- **Phase 1**
  - **Iteratively select a certain number of tensors from the beginning**



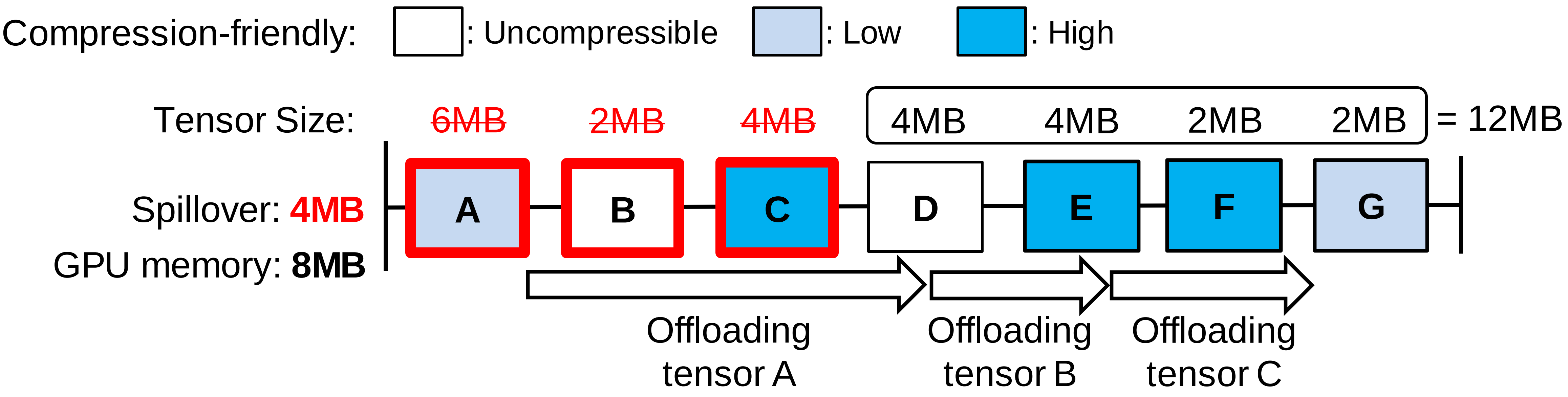
Offloading Scheduler: Phase 1

- **Finding an optimal scheduler for a given target batch size**
- **Phase 1**
  - **Iteratively select a certain number of tensors from the beginning**



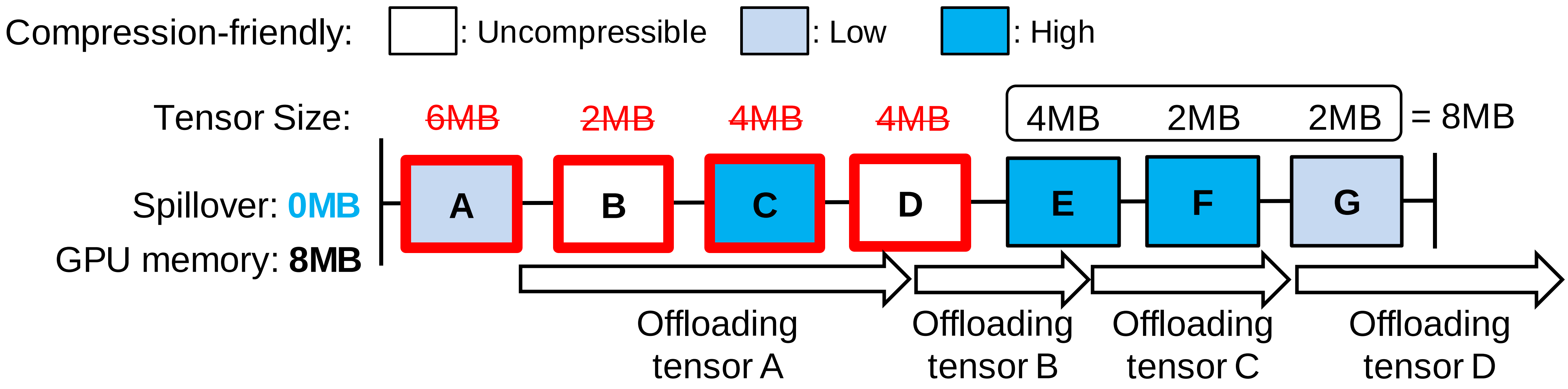
Offloading Scheduler: Phase 1

- **Finding an optimal scheduler for a given target batch size**
- **Phase 1**
  - **Iteratively select a certain number of tensors from the beginning**



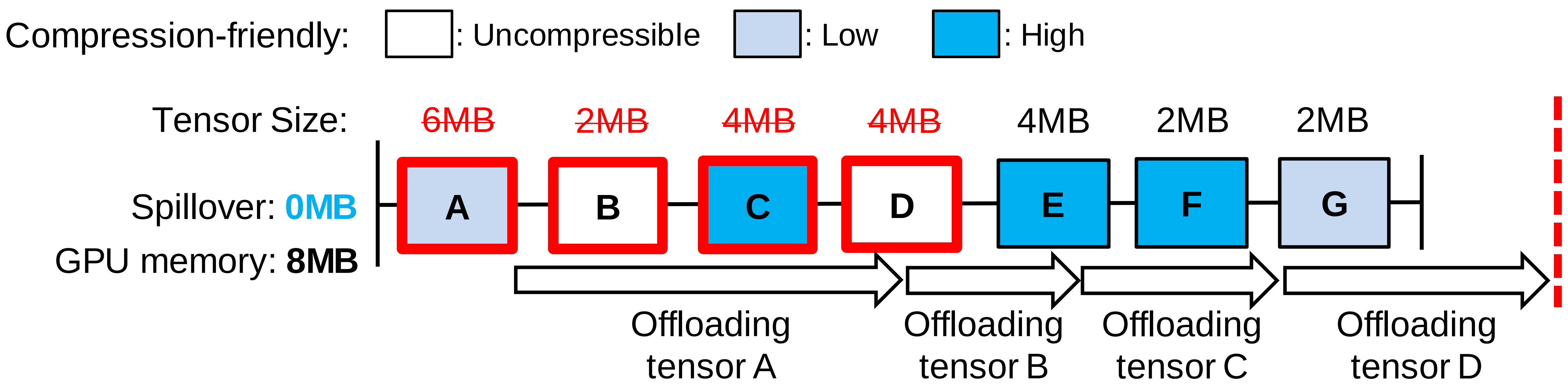
Offloading Scheduler: Phase 1

- **Finding an optimal scheduler for a given target batch size**
- **Phase 1**
  - **Iteratively select a certain number of tensors from the beginning**



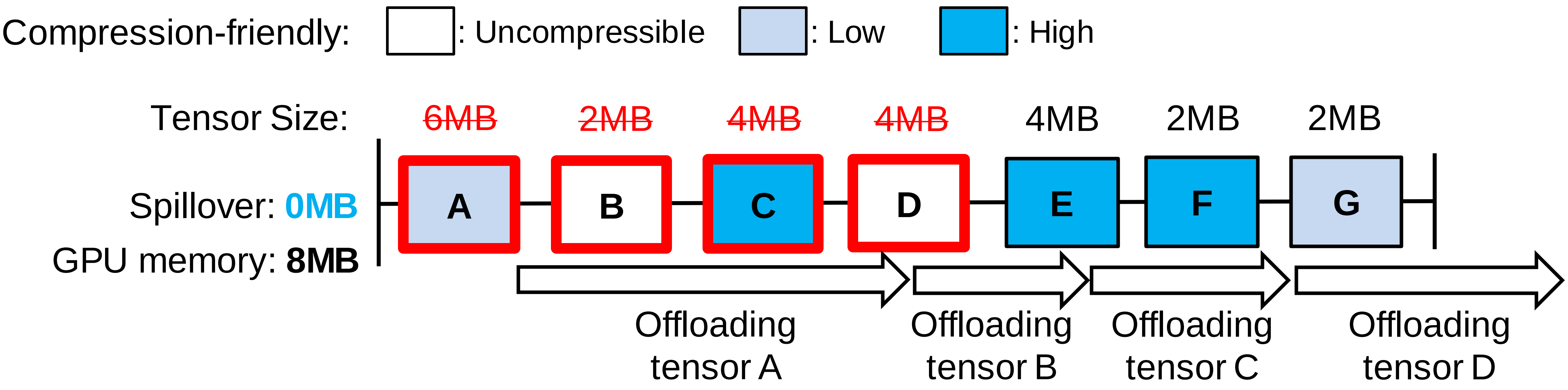
Offloading Scheduler: Phase 1

- **Finding an optimal scheduler for a given target batch size**
- **Phase 1**
  - **Iteratively select a certain number of tensors from the beginning**



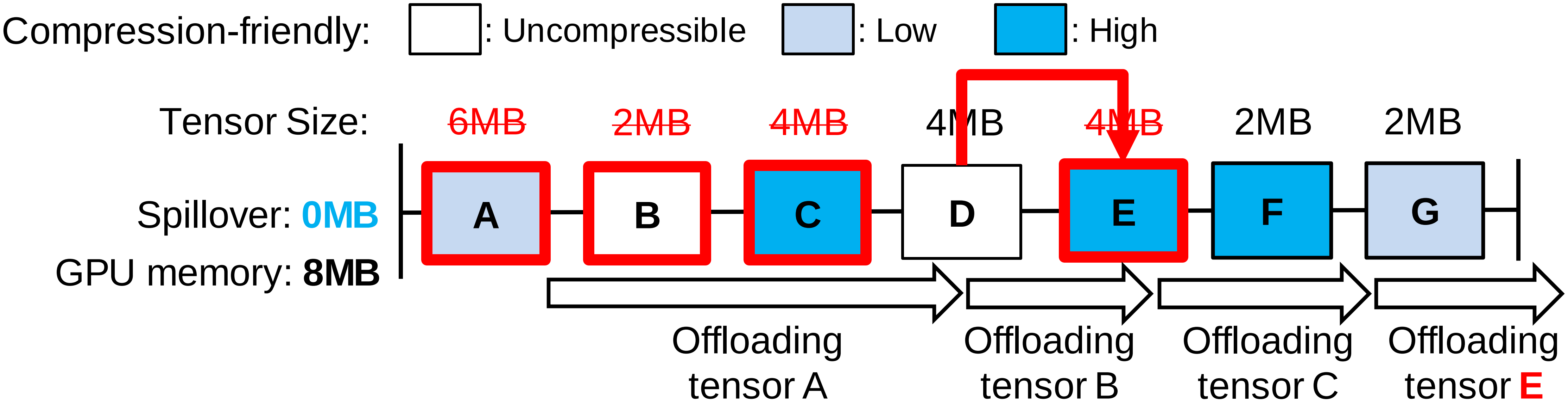
Offloading Scheduler: Phase 2

- **Finding an optimal scheduler for a given target batch size**
- **Phase 1**
  - Iteratively select a certain number of tensors from the beginning
- **Phase 2**
  - **Replace the tensors as offloading candidates with more compression-friendly tensors**



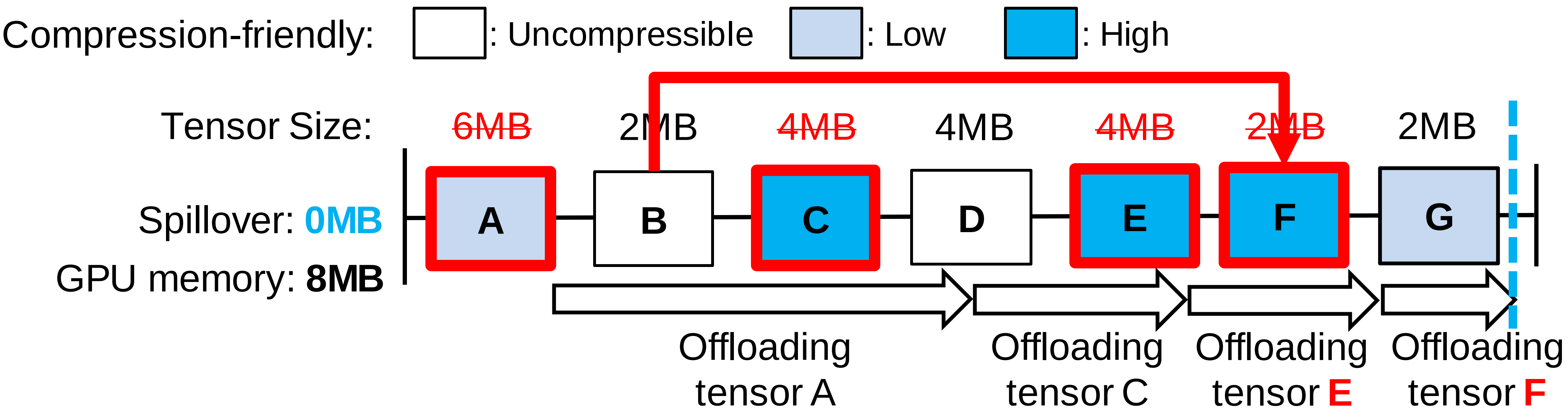
Offloading Scheduler: Phase 2

- **Finding an optimal scheduler for a given target batch size**
- **Phase 1**
  - Iteratively select a certain number of tensors from the beginning
- **Phase 2**
  - **Replace the tensors as offloading candidates with more compression-friendly tensors**



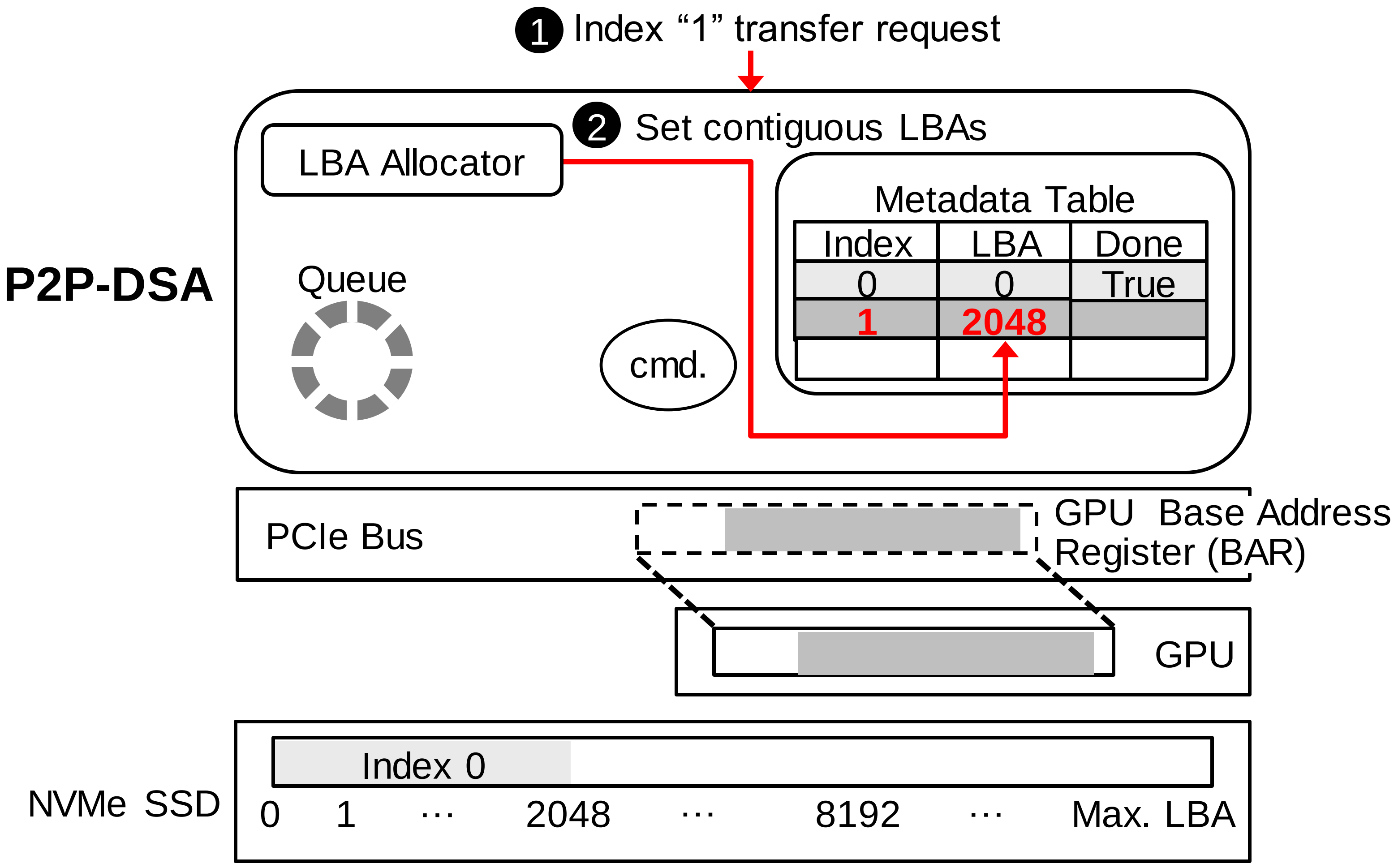
Offloading Scheduler: Phase 2

- **Finding an optimal scheduler for a given target batch size**
- **Phase 1**
  - Iteratively select a certain number of tensors from the beginning
- **Phase 2**
  - **Replace the tensors as offloading candidates with more compression-friendly tensors**



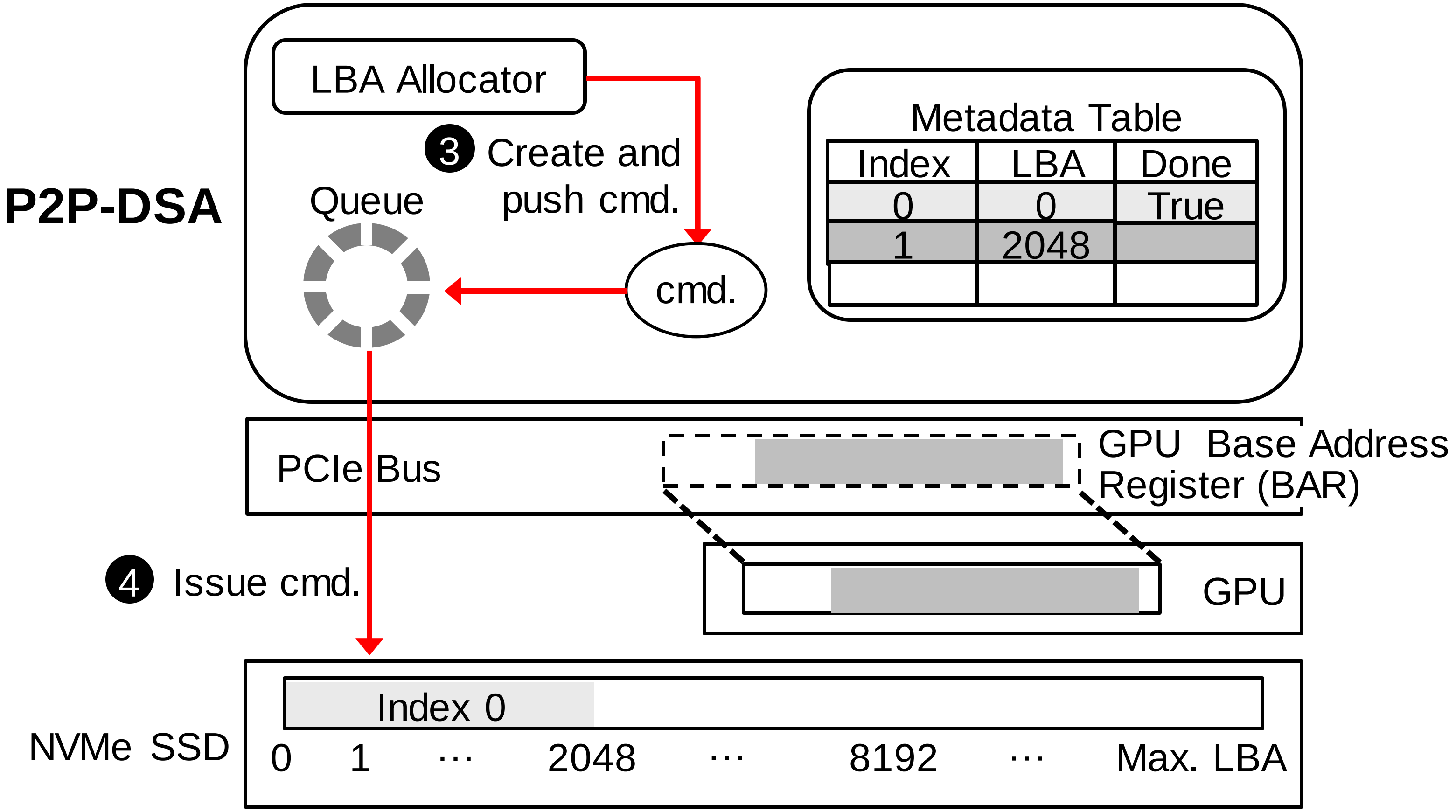
Peer-to-peer Direct Storage Access (P2P-DSA)

- **Lightweight I/O stack to enable direct tensor offloading/prefetching**
- **Example walk-through**



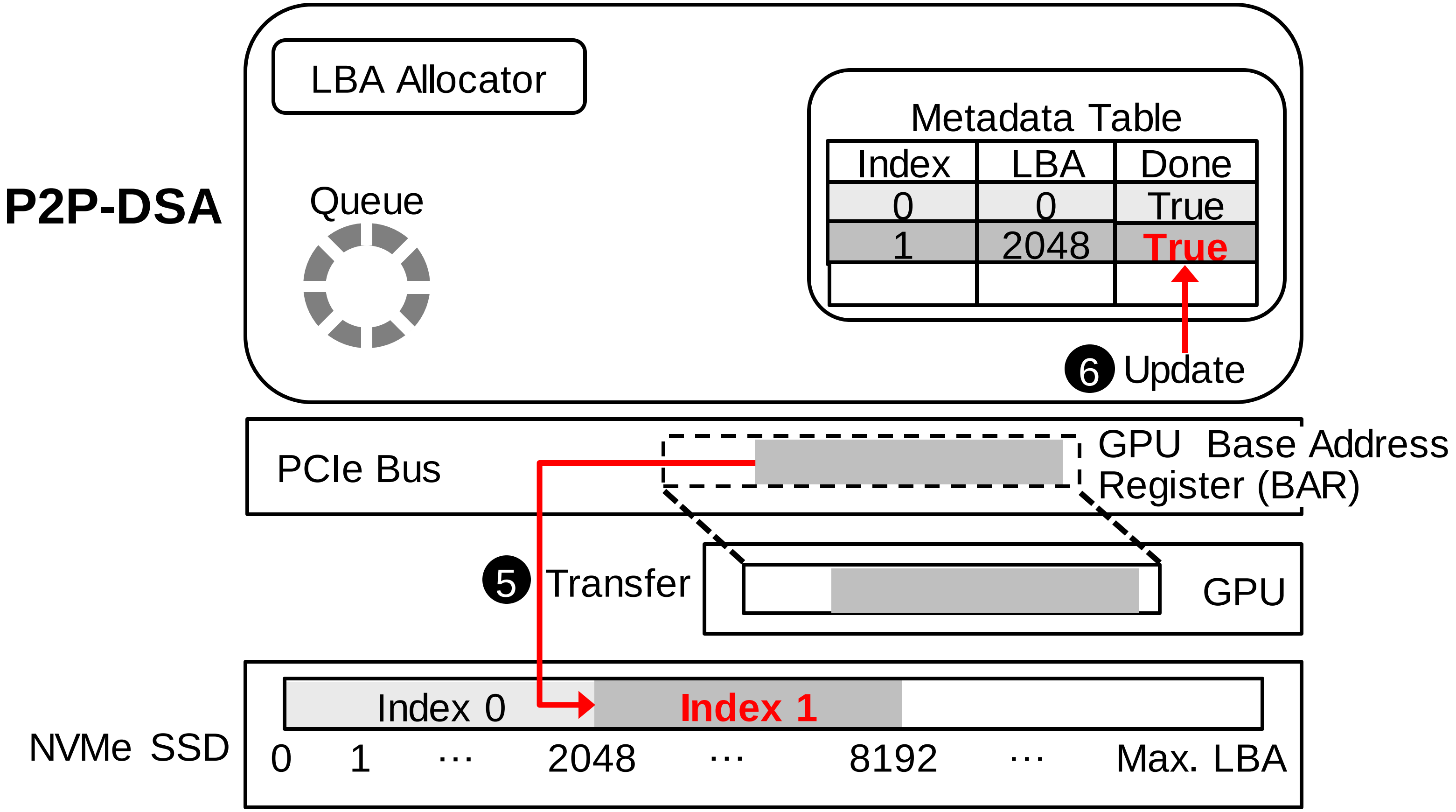
Peer-to-peer Direct Storage Access (P2P-DSA)

- **Lightweight I/O stack to enable direct tensor offloading/prefetching**
- **Example walk-through**



Peer-to-peer Direct Storage Access (P2P-DSA)

- **Lightweight I/O stack to enable direct tensor offloading/prefetching**
- **Example walk-through**



Methodology

- **System configurations**

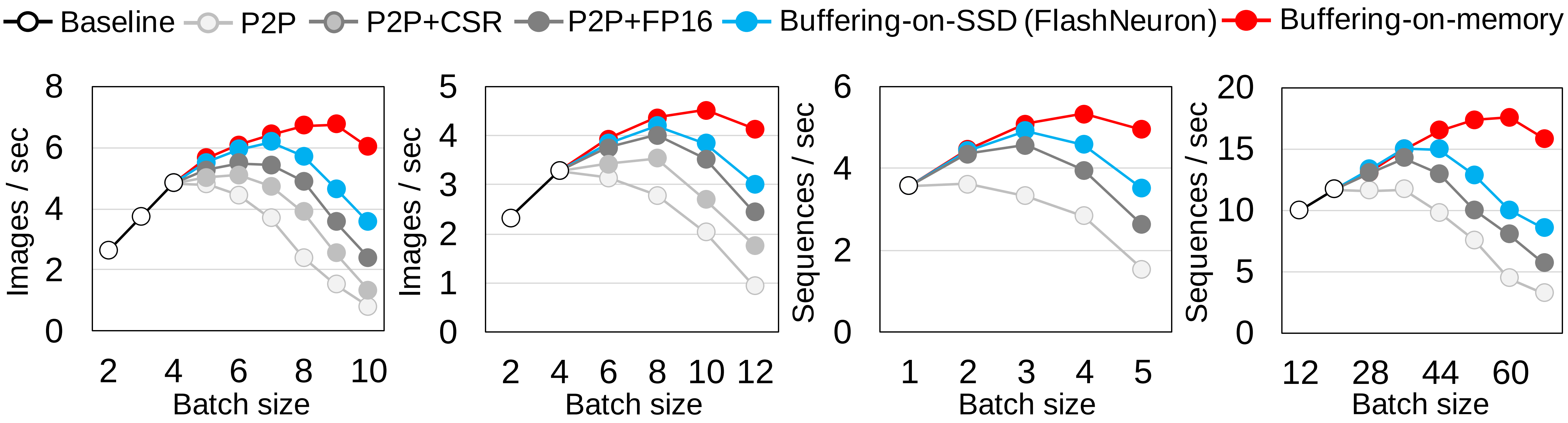
CPU	Intel Xeon Gold 6244 CPU 8 cores @ 3.60GHz
GPU	NVIDIA Tesla V100 16GB PCIe
Memory	Samsung DDR4-2666 64GB (32GB x 2)
Storage	Samsung PM1725b 8TB PCIe Gen3 8-lane x 2 (Seq. write: 3.3GB/s, seq. read: 6.3GB/s)
OS	Ubuntu server 18.04.3 LTS
Python	Version 3.7.3
PyTorch	Version 1.2

- **DNN models and datasets**

Network	**Dataset**	**# of layers**
ResNet-1922	ImageNet	1922
DenseNet-1001	ImageNet	1001
BERT-XLarge	SQuAD 1.1	48 transformer blocks
HBMP	SciTail	24 hidden layers

Evaluation: Overall Results

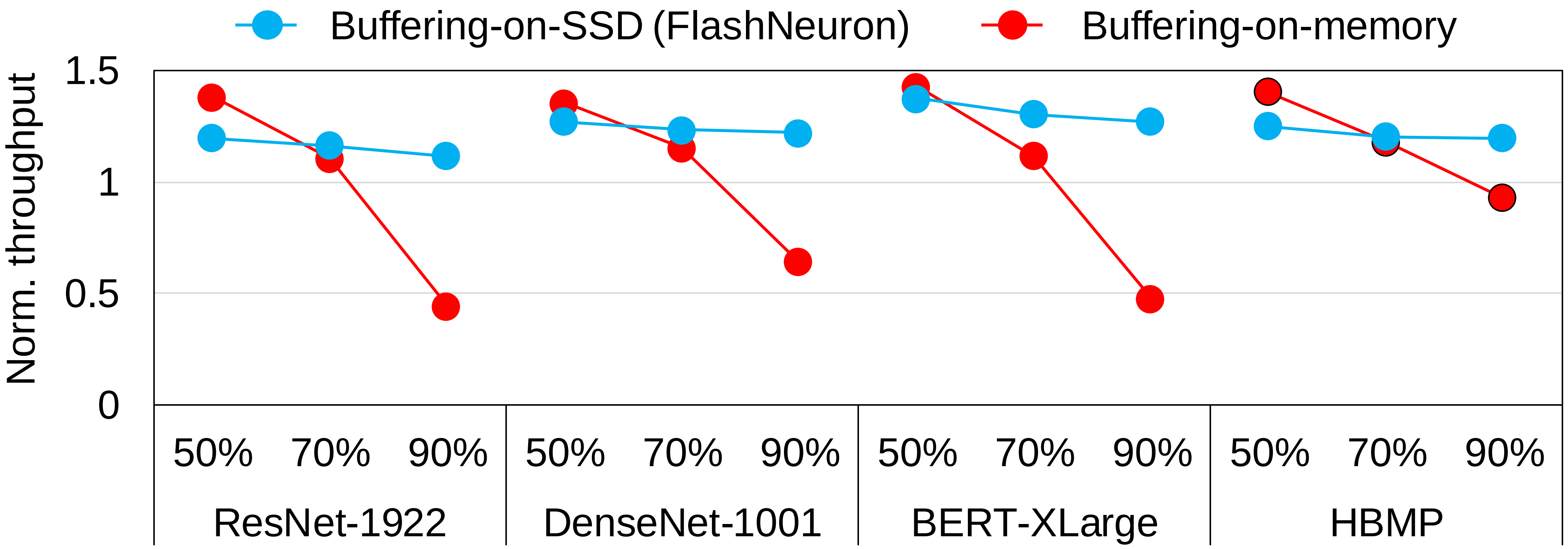
- 12.4x to 14x batch size increment compared to the baseline using GPU memory only
- Up to 37.8% (30.3% on average) training throughput improvement



ResNet-1922 **DenseNet-1001** **BERT-XLarge** **HBMP**

Evaluation: Co-locating Bandwidth-Intensive Tasks on CPU

- **Throughput of DNN training on GPU**
  - Buffering-on-memory: **40.2%** throughput degradation when CPU utilizes 90% of the memory BW
  - FlashNeuron: **20.2%** throughput gain when CPU utilizes 90% of the memory BW



Conclusion

FlashNeuron enables large-batch training of very deep and wide neural networks

- **Identify a bandwidth contention problem in recent buffering-on-memory proposal**
- **Introduce a novel offloading scheduler to fully utilize the scarce SSD write bandwidth**
- **Implement a lightweight user-space I/O stack customized for DNN training**

Thank You!

Source code of FlashNeuron is available at

https://github.com/SNU-ARC/flashneuron.git