Improving Data Reuse in NPU On-chip Memory with Interleaved Gradient Order for DNN Training

Jungwoo Kim, Seonjin Na, Sanghyeon Lee,
Sunho Lee, and Jaehyuk Huh

Efficient Accelerator for DNN Training

- Training ML models require energy-efficient acceleration of NPU

- Data and model parallelism in NPU → high-performance DNN training
  - Over 90% of training at Google is on TPUs*

Google TPU

AWS Trainium

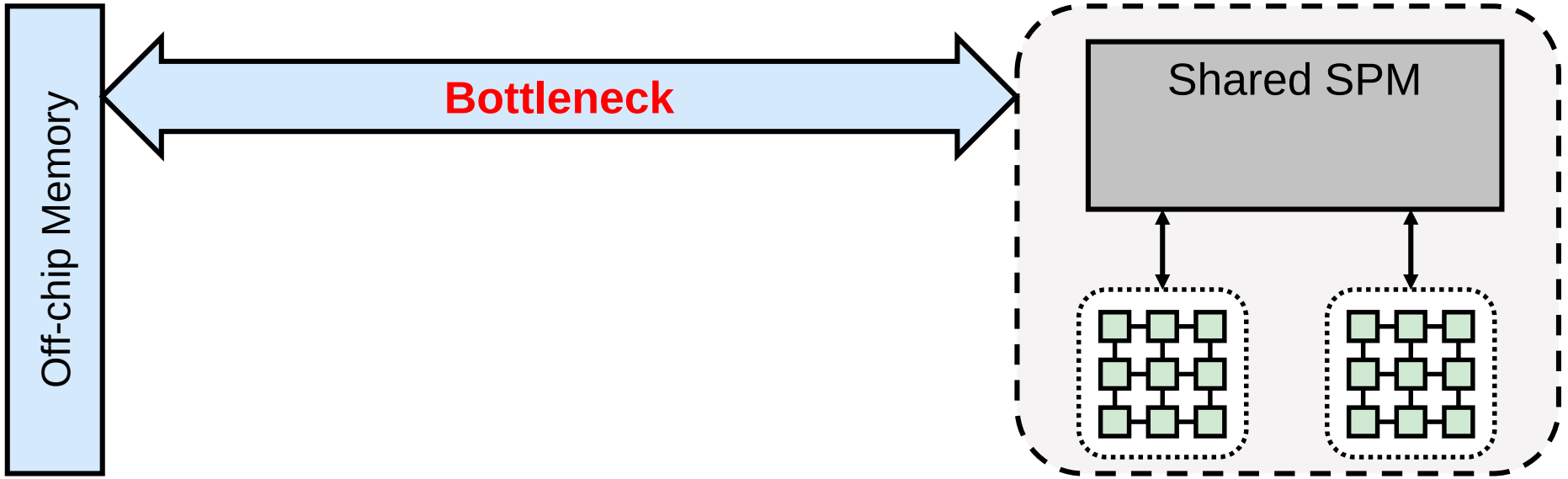
Tesla Dojo

Intel Gaudi2

* Jouppi, Norman P., et al. "TPU v4: An Optically Reconfigurable Supercomputer for Machine Learning with Hardware Support for Embeddings." (ISCA 2023)

Accelerator for Training Neural Networks

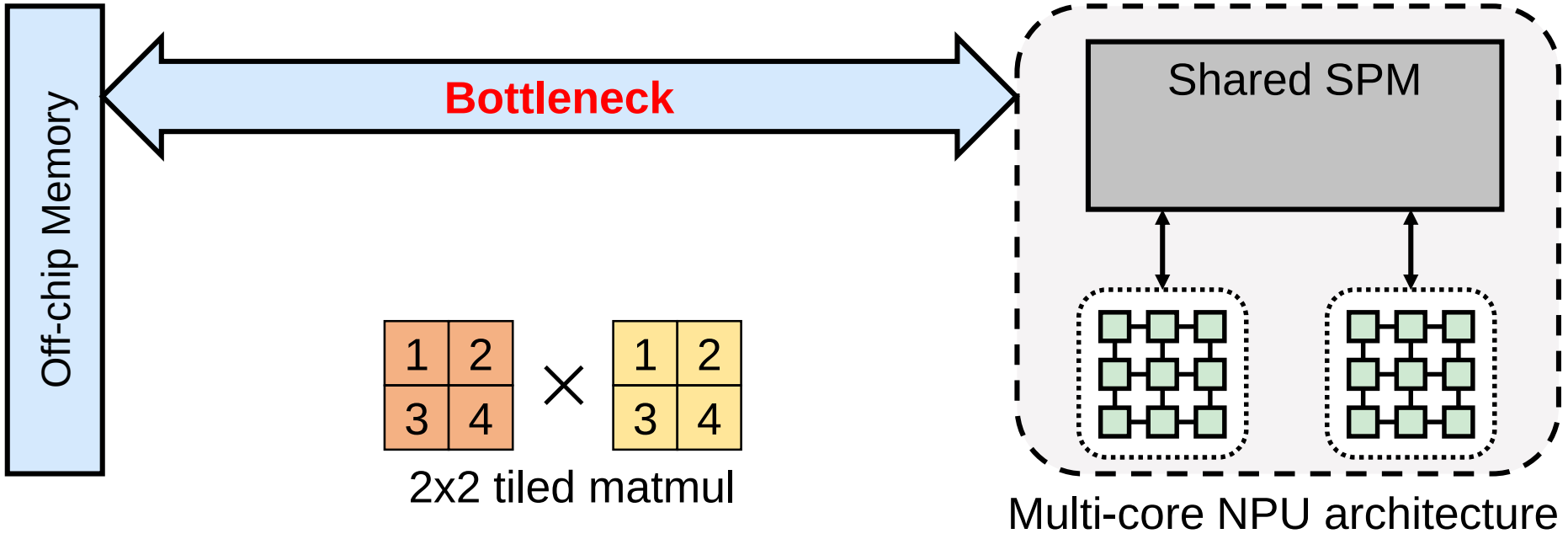
- Large models + complex datasets → **Memory bandwidth is bottleneck**
- Scratchpad memory (**SPM**) plays a significant role in NPU
- **Improving data reuse** → Reducing memory access → Better performance



Multi-core NPU architecture

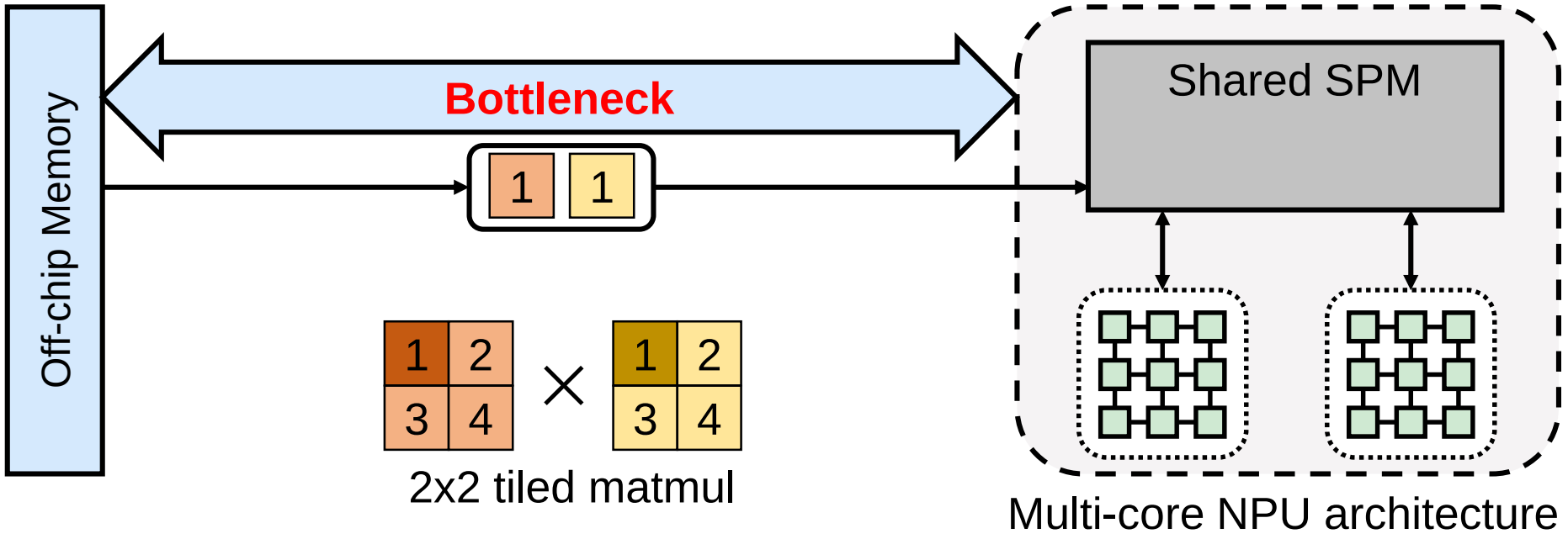
Accelerator for Training Neural Networks

- Large models + complex datasets → **Memory bandwidth is bottleneck**
- Scratchpad memory (**SPM**) plays a significant role in NPU
- **Improving data reuse** → Reducing memory access → Better performance



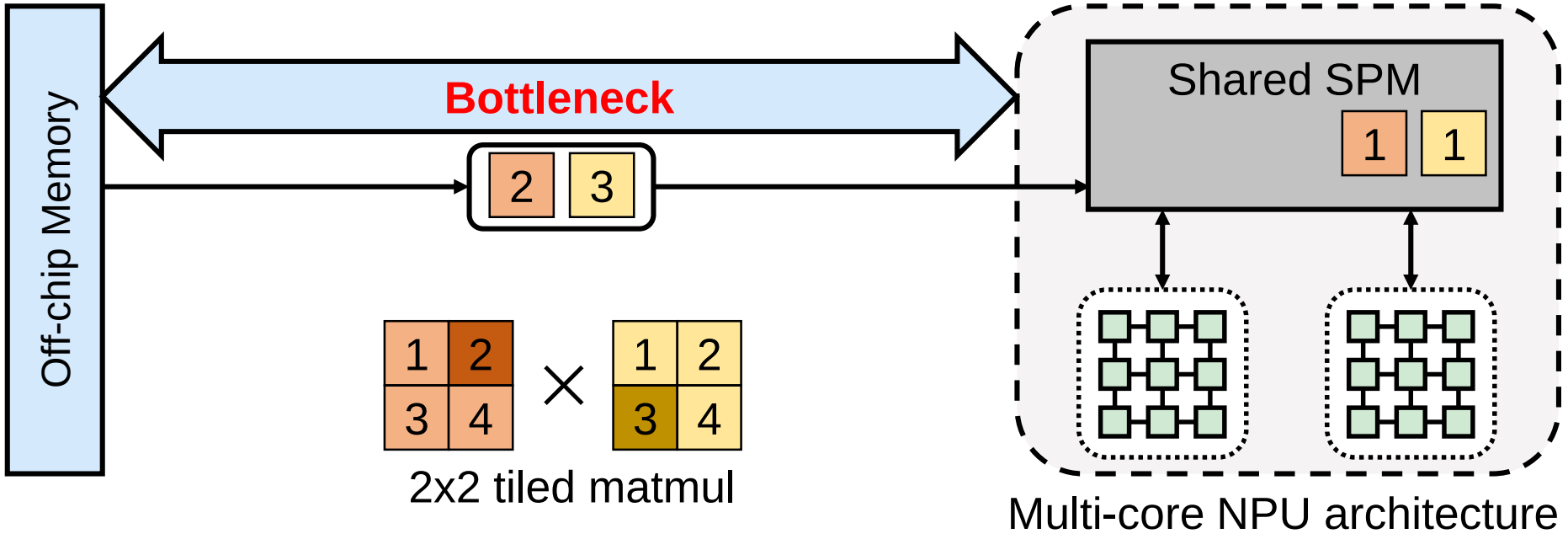
Accelerator for Training Neural Networks

- Large models + complex datasets → **Memory bandwidth is bottleneck**
- Scratchpad memory (**SPM**) plays a significant role in NPU
- **Improving data reuse** → Reducing memory access → Better performance



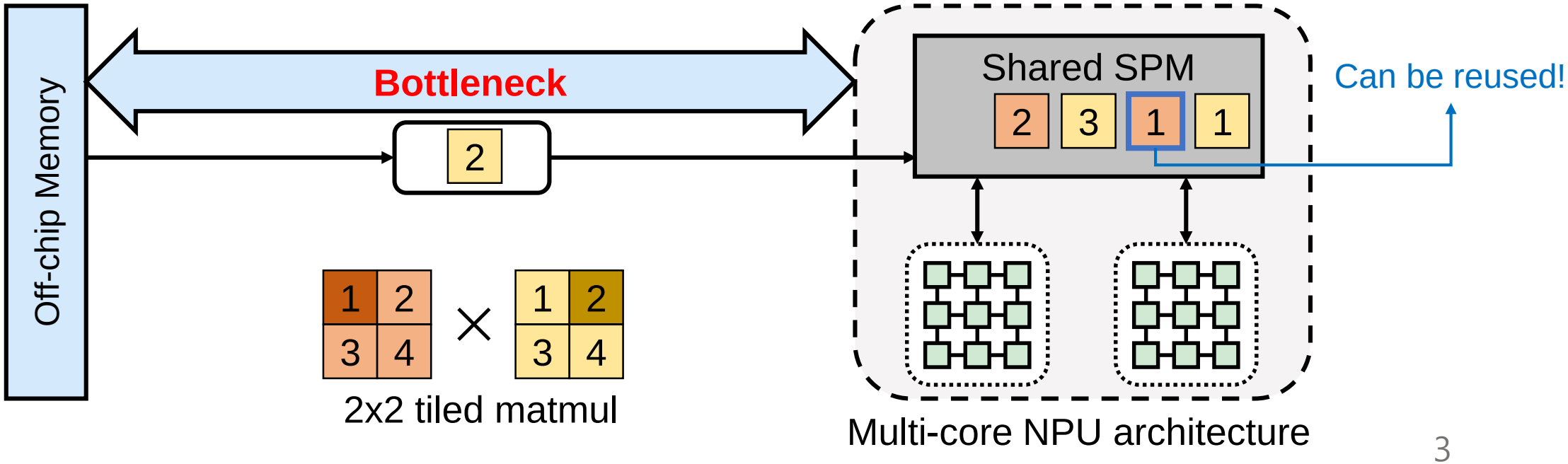
Accelerator for Training Neural Networks

- Large models + complex datasets → **Memory bandwidth is bottleneck**
- Scratchpad memory (**SPM**) plays a significant role in NPU
- **Improving data reuse** → Reducing memory access → Better performance



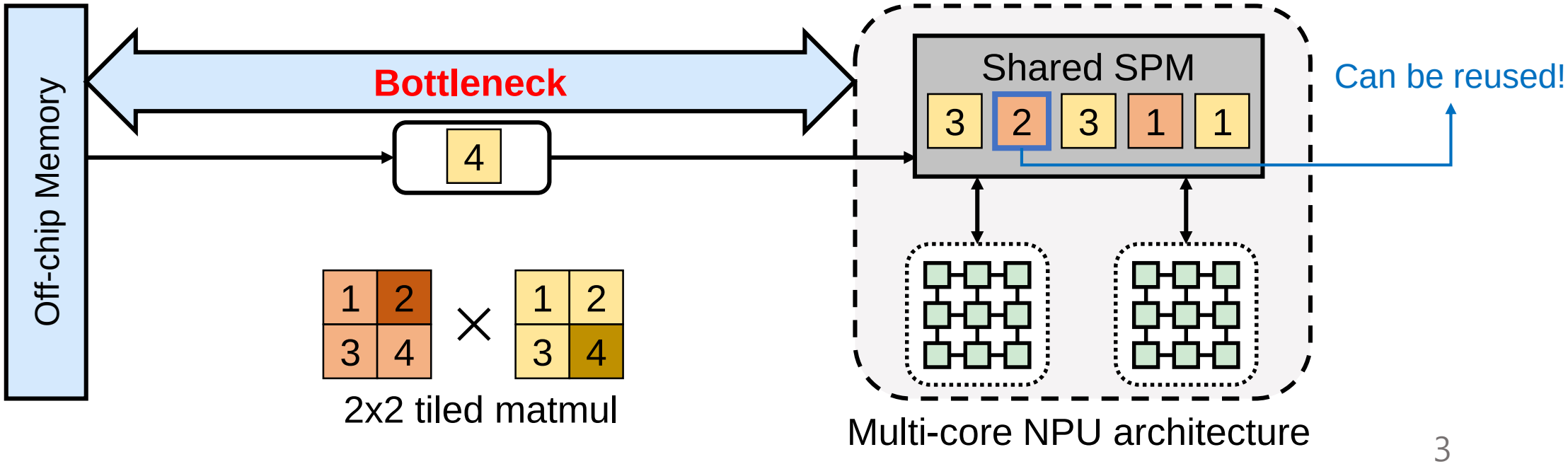
Accelerator for Training Neural Networks

- Large models + complex datasets → **Memory bandwidth is bottleneck**
- Scratchpad memory (**SPM**) plays a significant role in NPU
- **Improving data reuse** → Reducing memory access → Better performance



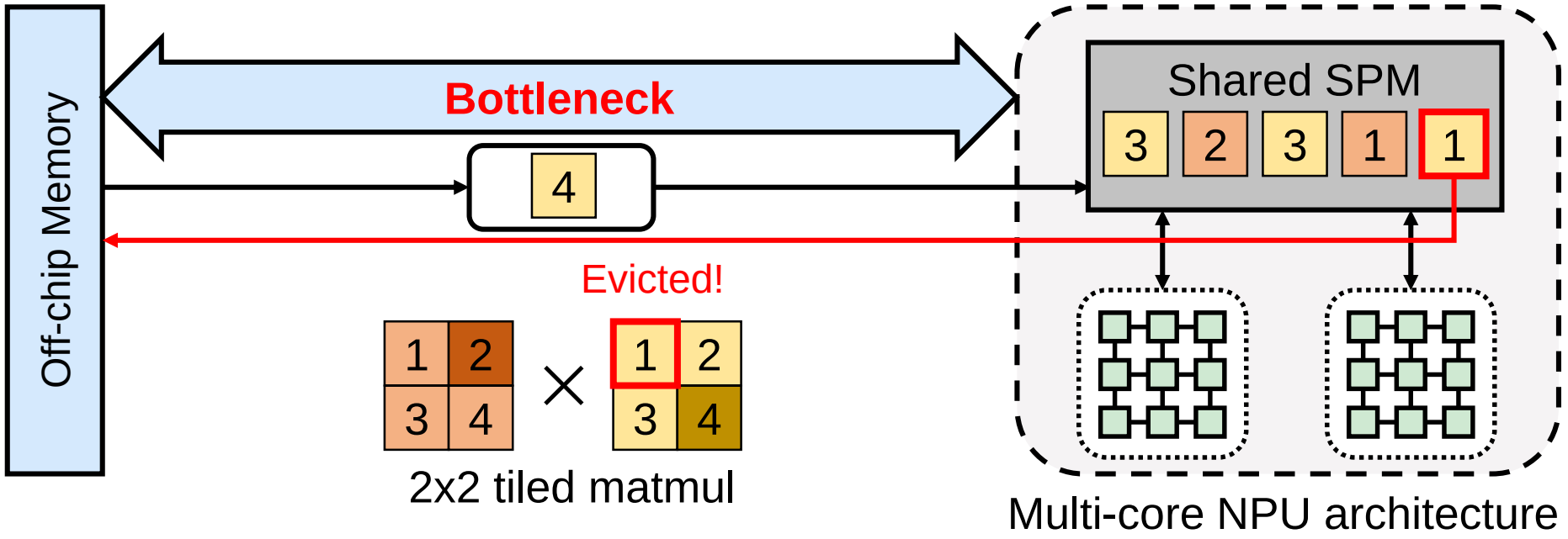
Accelerator for Training Neural Networks

- Large models + complex datasets → **Memory bandwidth is bottleneck**
- Scratchpad memory (**SPM**) plays a significant role in NPU
- **Improving data reuse** → Reducing memory access → Better performance



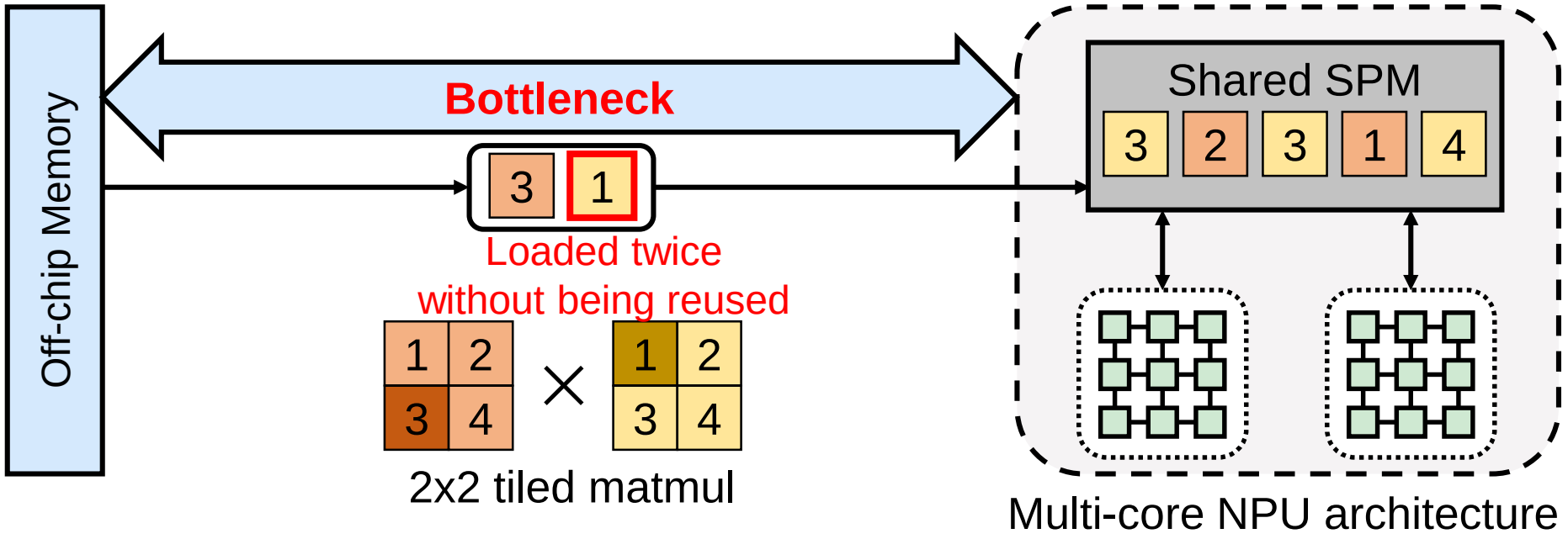
Accelerator for Training Neural Networks

- Large models + complex datasets → **Memory bandwidth is bottleneck**
- Scratchpad memory (**SPM**) plays a significant role in NPU
- **Improving data reuse** → Reducing memory access → Better performance



Accelerator for Training Neural Networks

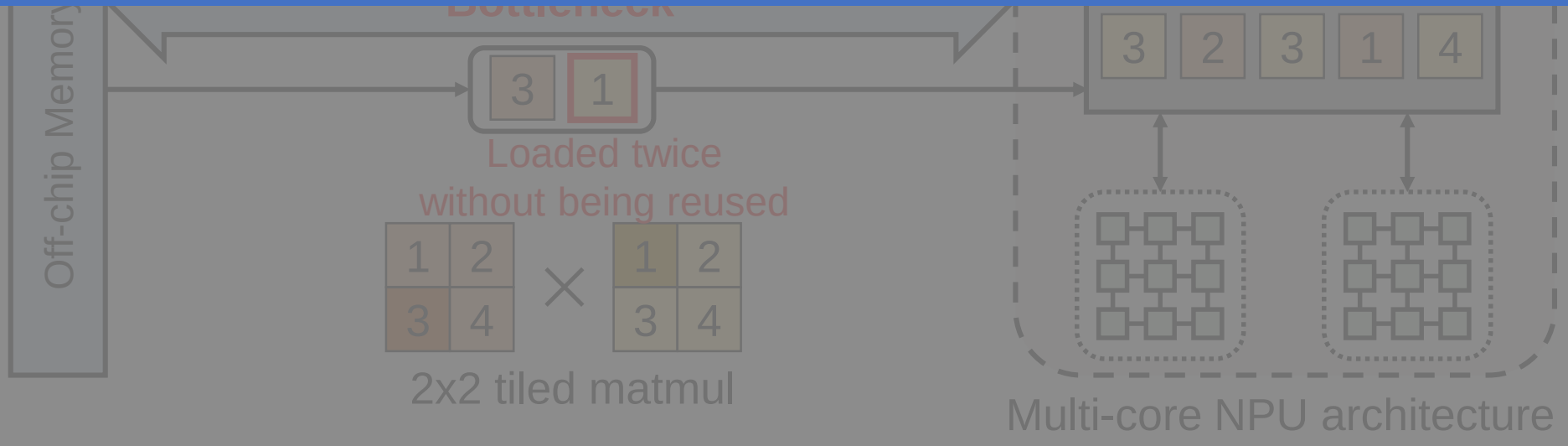
- Large models + complex datasets → **Memory bandwidth is bottleneck**
- Scratchpad memory (**SPM**) plays a significant role in NPU
- **Improving data reuse** → Reducing memory access → Better performance



Accelerator for Training Neural Networks

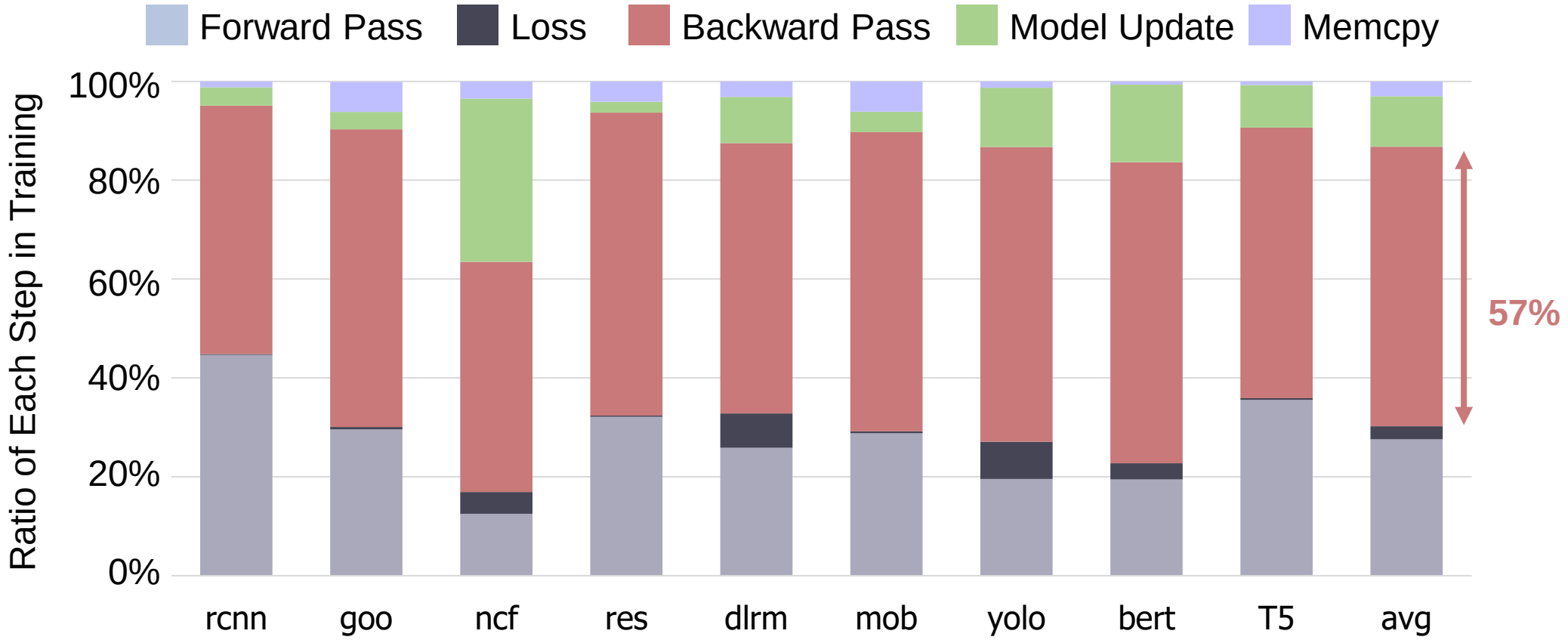
- Large models + complex datasets → **Memory bandwidth is bottleneck**
- Scratchpad memory (**SPM**) plays a significant role in NPU

Is there <u>a new data reuse chance</u> in DNN training?

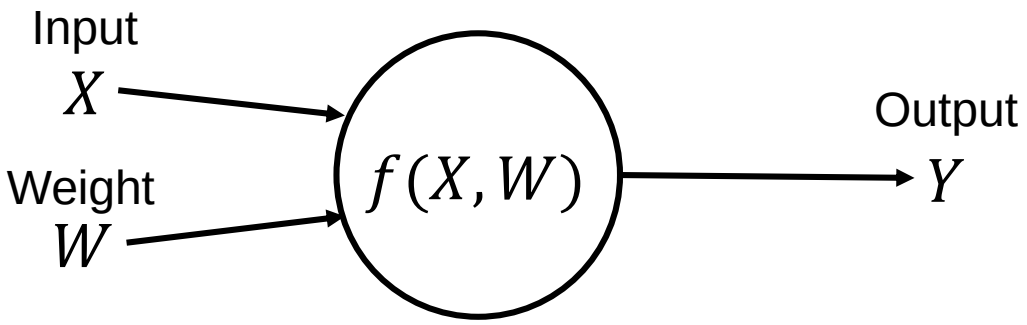


Breakdown of DNN Training

- Breakdown of the total training time for each step in DNN training
  - PyTorch 1.13, NVIDIA A100 40GB, 90 epochs, batch size ≥ 256
- **Backward pass** is the most critical step **(57%)** in the training procedures

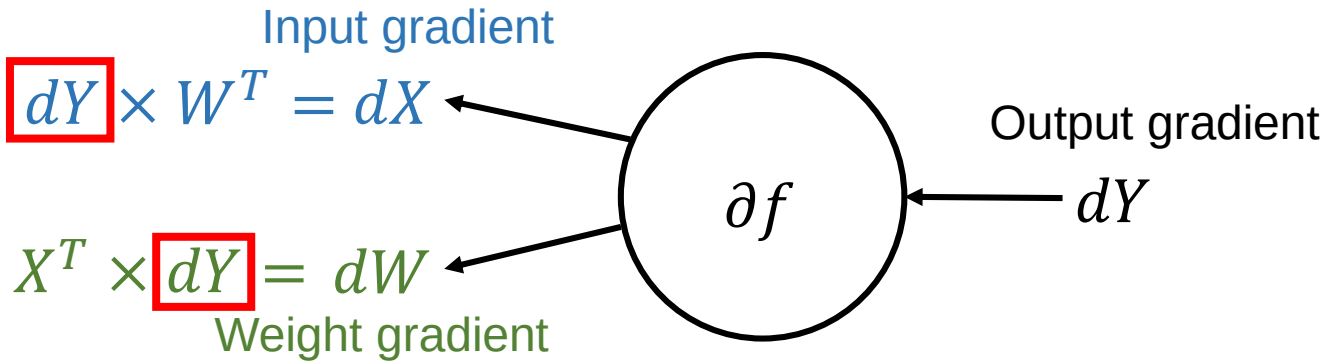


Operations in DNN Training



Forward Pass

- $\boldsymbol{dX}$, $\boldsymbol{dW}$, $\boldsymbol{dY}$= partial derivative of loss with respect to $\boldsymbol{X}$, $\boldsymbol{W}$, $\boldsymbol{Y}$

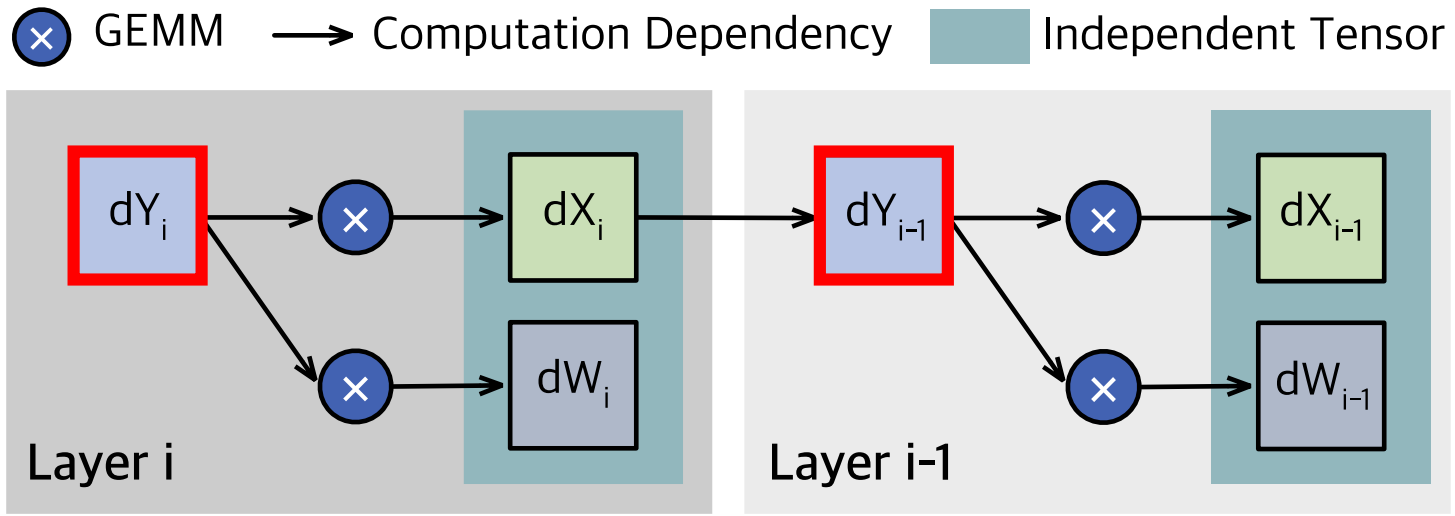


Backward Pass

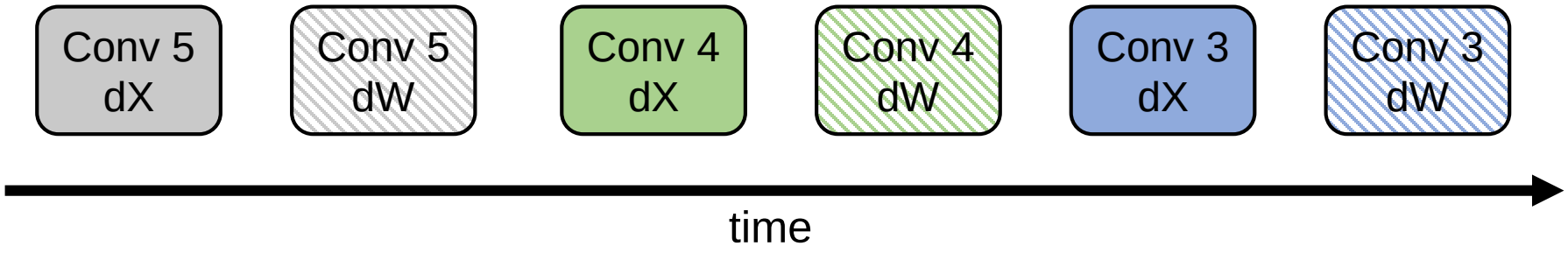
- **Same dY** is used in both dX and dW computations

Dependencies in Backward Pass

- **Two gradient computations (dX, dW)** in the same layer are **independent**



- dX and dW are **sequentially** computed in conventional accelerators



Ratio of Output Gradient (dY) in DNN Training

- Sequentially computing dX and dW → No dY reuse in SPM

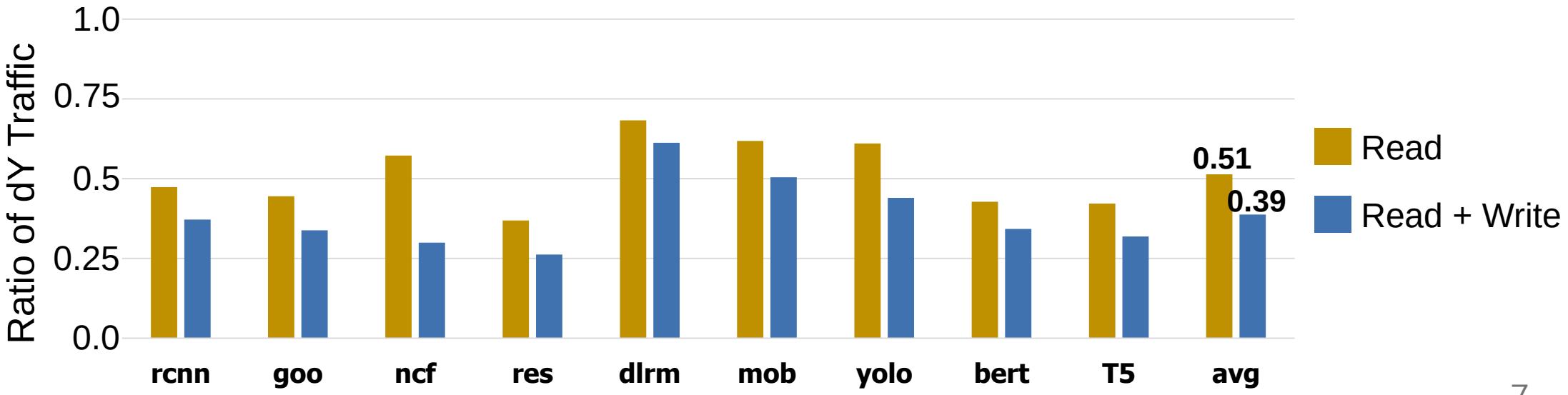
dY is loaded to SPM twice

$$X^T \times dY \longrightarrow dW$$

$$dY \times W^T \longrightarrow dX$$

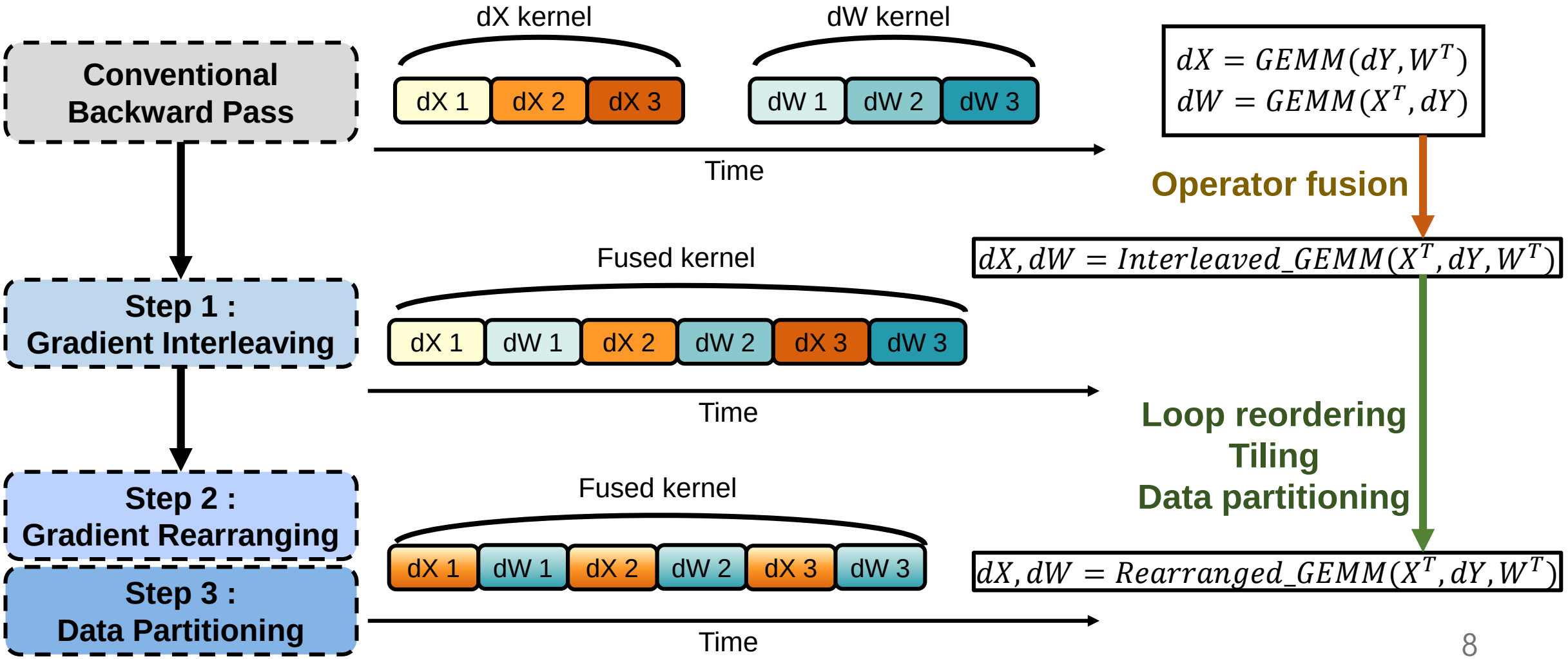
Read Write

- dY traffic occupies 51%/39% of read/total data traffic



Overview

- Software transformation : **load dY just once** to improve data reuse in SPM



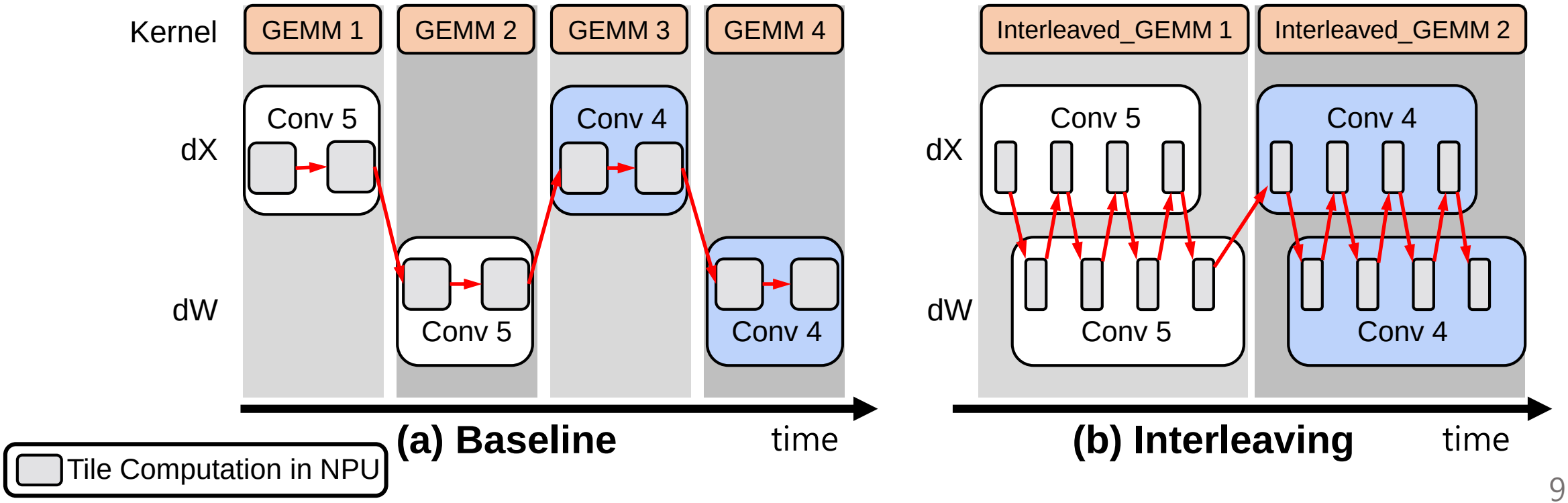
Step 1 : Interleaving Gradient Computations

- **Key idea :**

$$dX = GEMM(dY, W^T)$$
$$dW = GEMM(X^T, dY)$$

$$\rightarrow dX, dW = Interleaved\_GEMM(X^T, dY, W^T)$$

- **Independent** gradient computations **can be interleaved**
  - Computational instructions remain unchanged



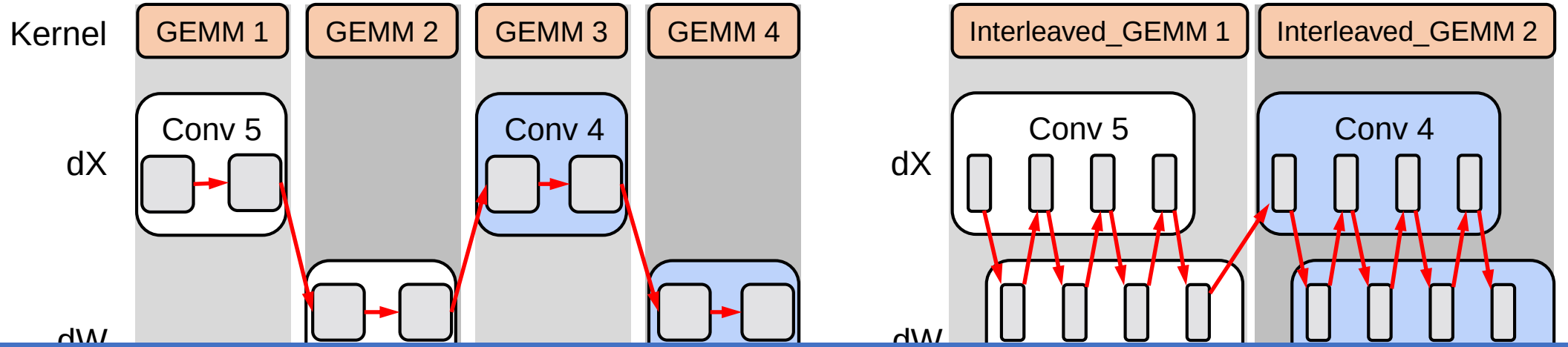
Step 1 : Interleaving Gradient Computations

- **Key idea :**

$$dX = GEMM(dY, W^T)$$
$$dW = GEMM(X^T, dY)$$

$$\rightarrow dX, dW = Interleaved\_GEMM(X^T, dY, W^T)$$

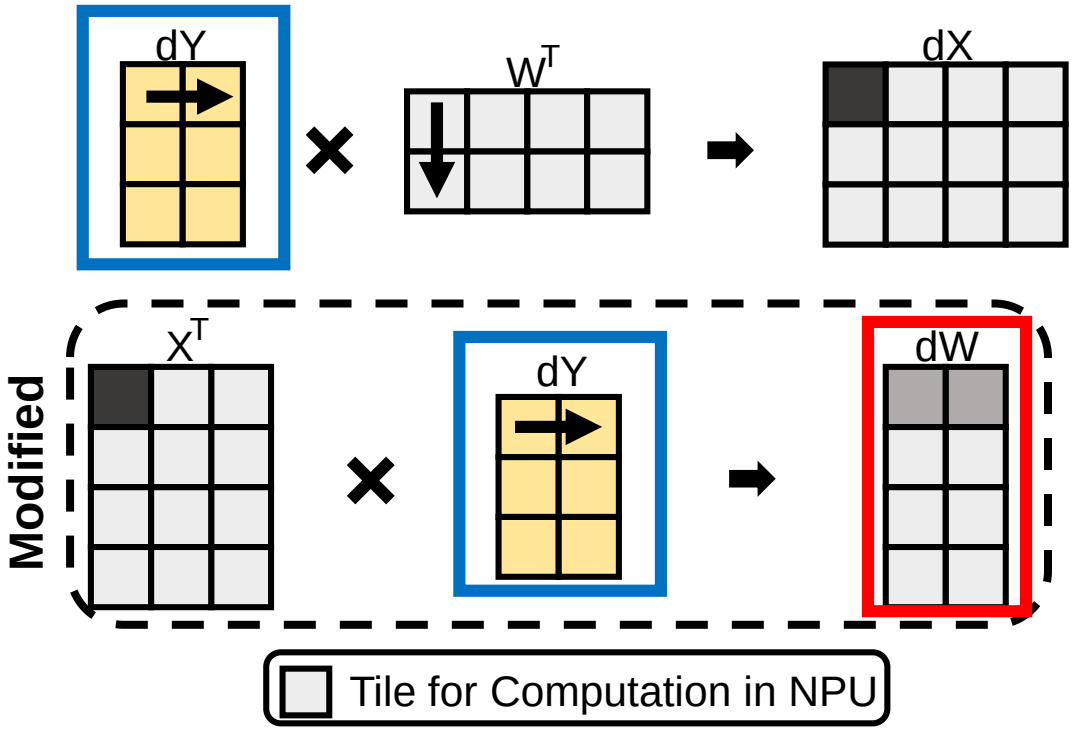
- **Independent** gradient computations **can be interleaved**
  - Computational instructions remain unchanged



How can we maximize dY reuse in interleaved computation?

Step 2 : Rearranging Gradient Computations

- We rearrange operations in an interleaved GEMM to reuse dY
- Interleaving**+dXmajor** : Modify dW computation to match dY access orders



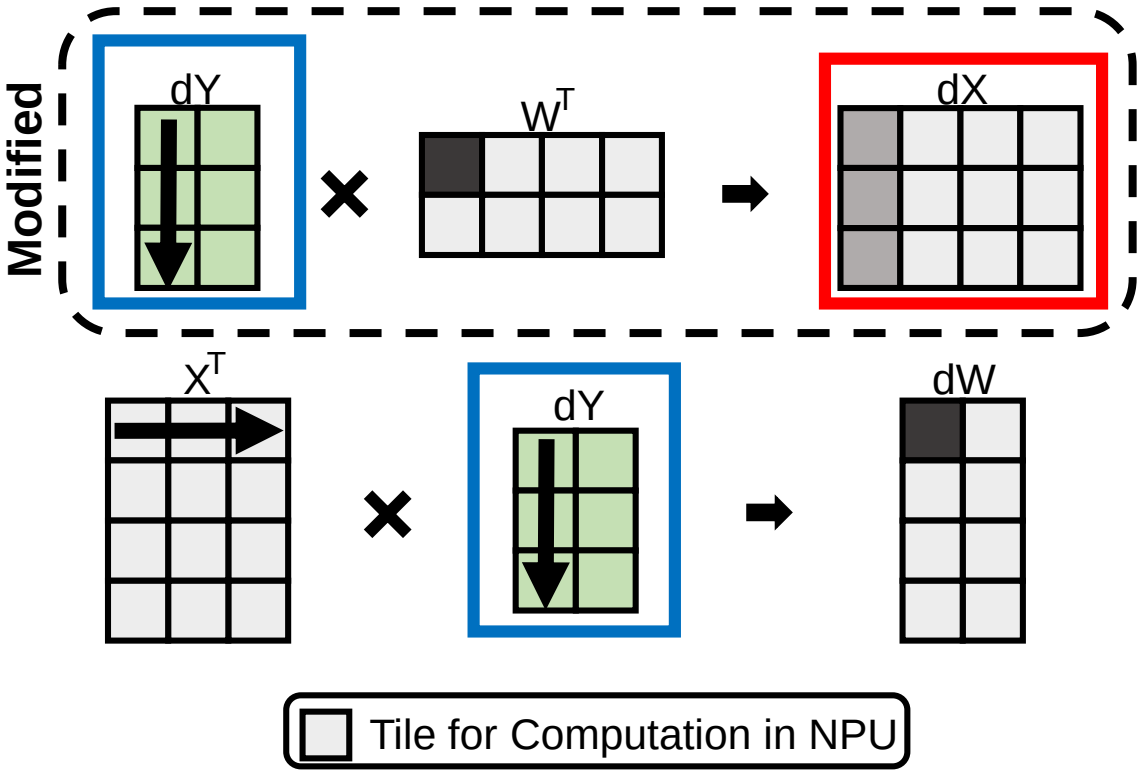
Trade-off

Always reuse **dY** (+)

Increase write traffic for **partial sum** of **dW** (−)

Step 2 : Rearranging Gradient Computations

- dXmajor : Too many write traffic for partial sum of dW in some layers
- Interleaving**+dWmajor** : Modify dX computation to match dY access orders



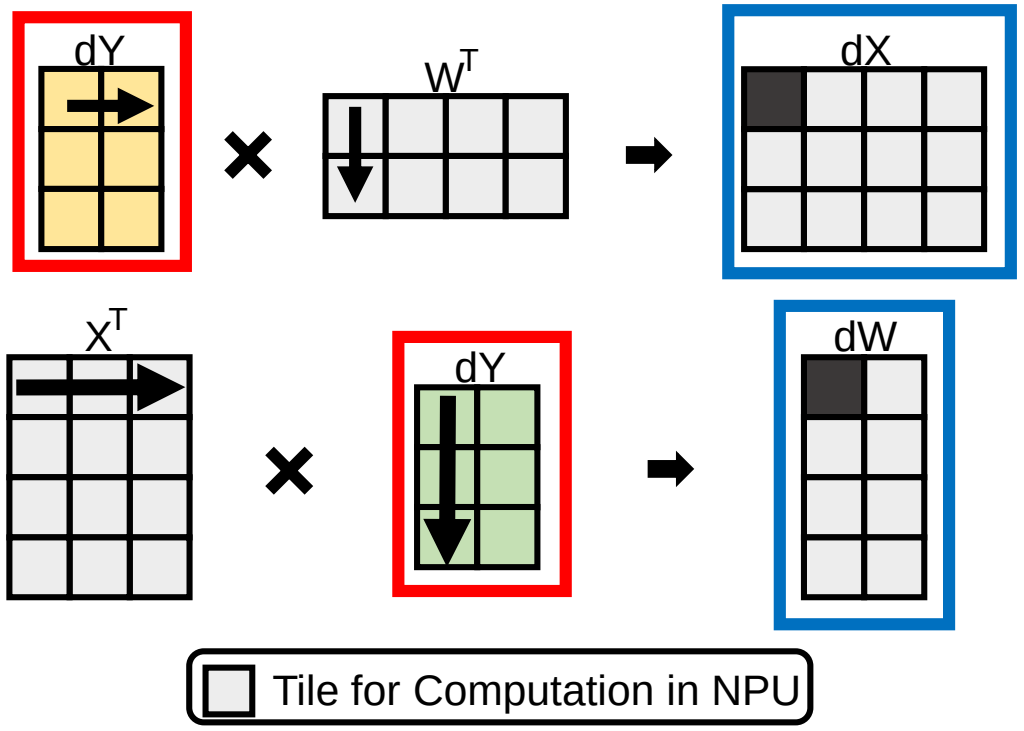
Trade-off

Always reuse **dY** (+)

Increase write traffic for **partial sum** of **dX** (−)

Step 2 : Rearranging Gradient Computations

- dX, dWmajor : Many write traffics for partial sum of dX or dW in some layers
- **Only** Interleaving : Use different dY access orders



Trade-off

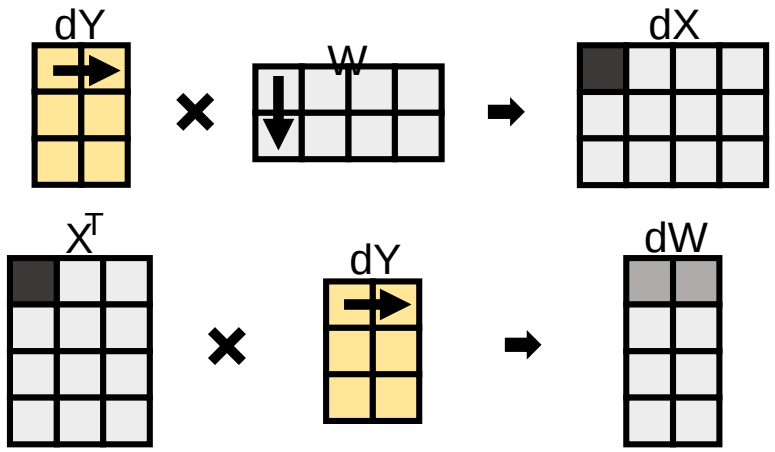
No partial sum of **dX** and **dW** (+)

Limited dY reuse (−)

Step 2 : Rearranging Gradient Computations

- Optimal rearrangement depends on **the shape of tensors**
  - If W is an extremely skinny matrix, choose **Interleaving+dWmajor**
  - If tensors are almost square matrices, choose **Only Interleaving**

Interleaving + dXmajor

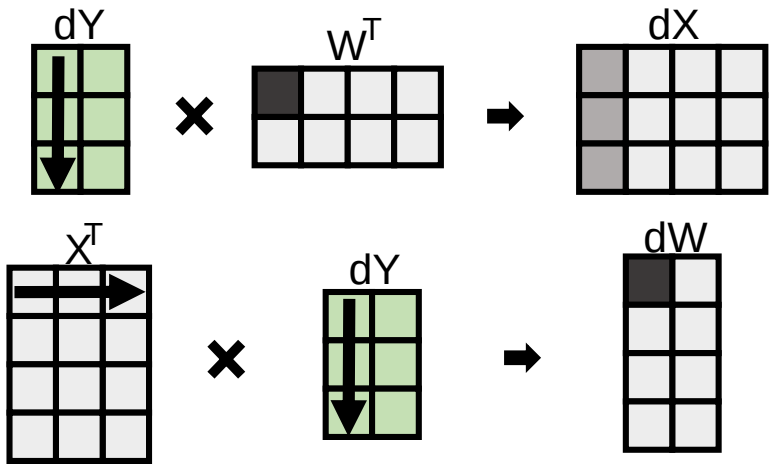


Trade-off

Always reuse **dY** (+)

Write traffic for **partial sum** of **dW** (−)

Interleaving + dWmajor

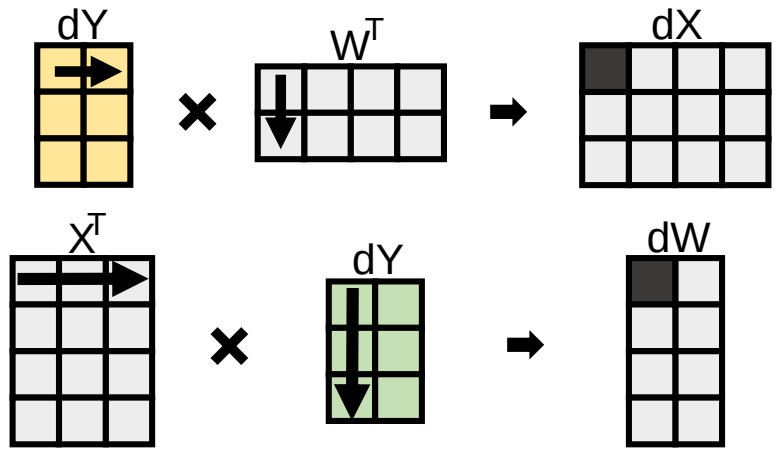


Trade-off

Always reuse **dY** (+)

Write traffic for **partial sum** of **dX** (−)

Only Interleaving



Trade-off

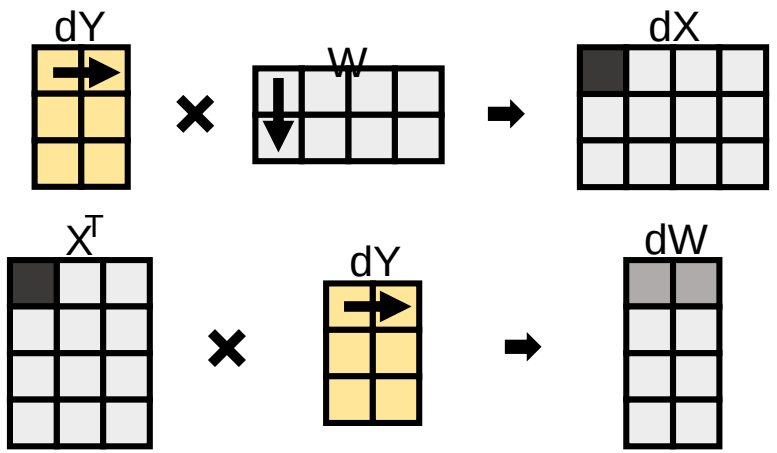
No partial sum of **dX** and **dW** (+)

Limited dY reuse (−)

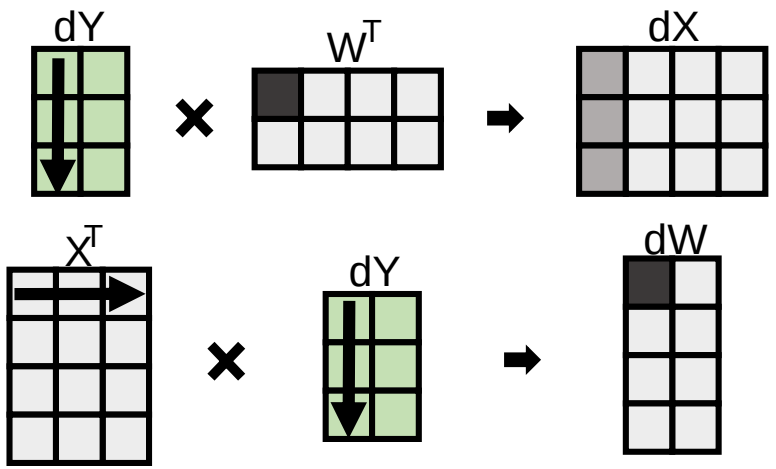
Step 2 : Rearranging Gradient Computations

- Optimal rearrangement depends on **the shape of tensors**
  - If W is an extremely skinny matrix, choose **Interleaving+dWmajor**
  - If tensors are almost square matrices, choose **Only Interleaving**

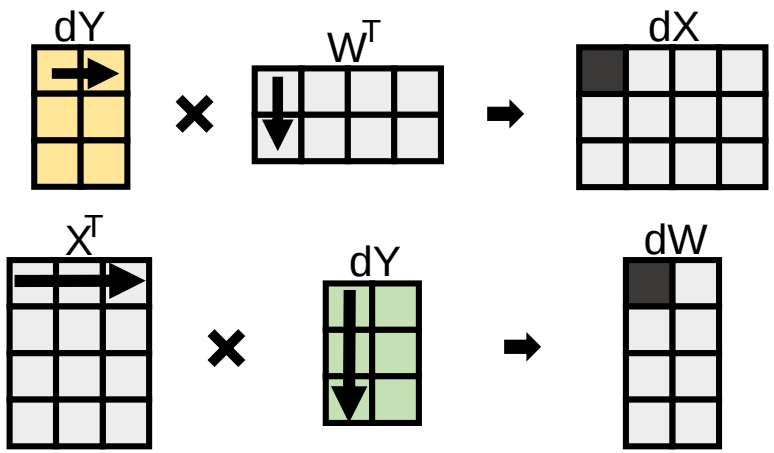
Interleaving + dXmajor



Interleaving + dWmajor



Only Interleaving

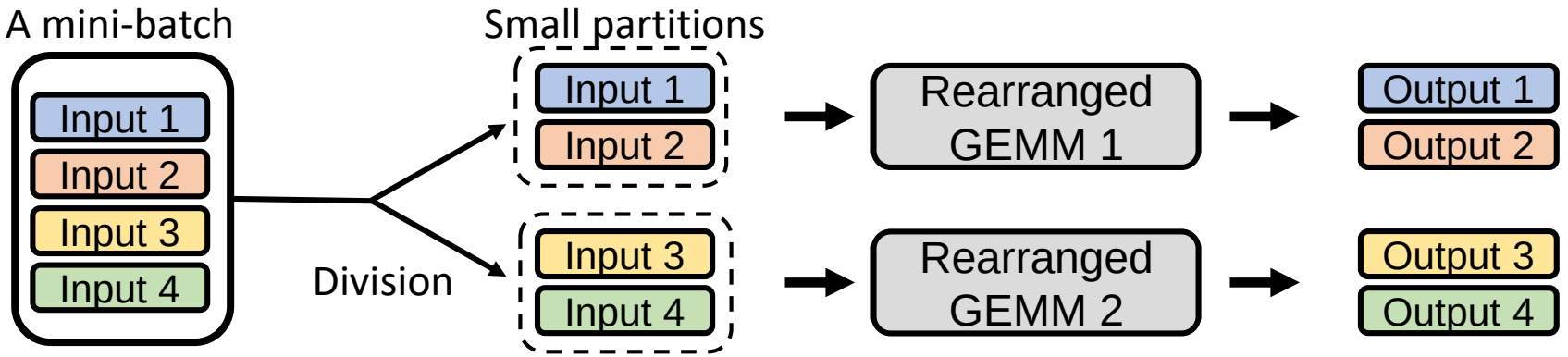


Can we improve rearrangement by reshaping tensors?

Step 3 : Data Partitioning for Fused GEMM

- Performance of rearranged GEMM depends on the shape of tensors

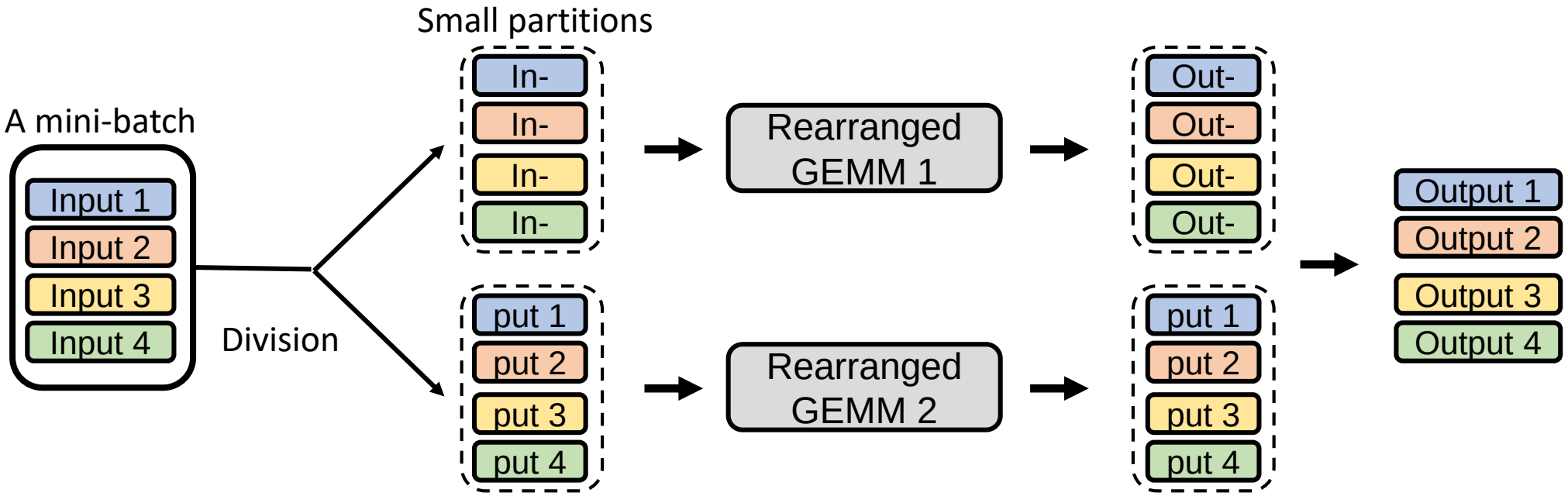
- A mini-batch is conventionally divided into smaller batches



- Divide data in different dimension to maximize rearrangement effect

Step 3 : Data Partitioning for Fused GEMM

- Divide in dimension orthogonal to input batches

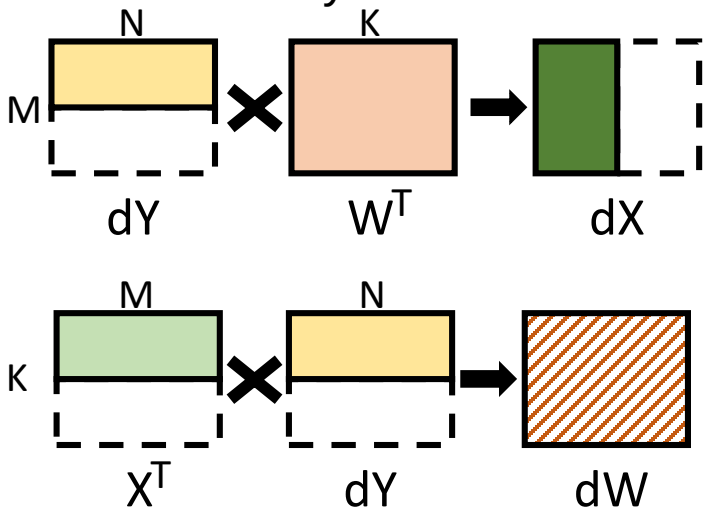


- GEMM can be defined by 3 dimensions (M,N,K) → 3 ways to partition data

Step 3 : Data Partitioning for Fused GEMM

- Optimal partitioning depends on **the tensor shape within rearranged GEMM**
  - Utilize KNN to predict optimal partitioning for each layer

Divided by dimension **M**

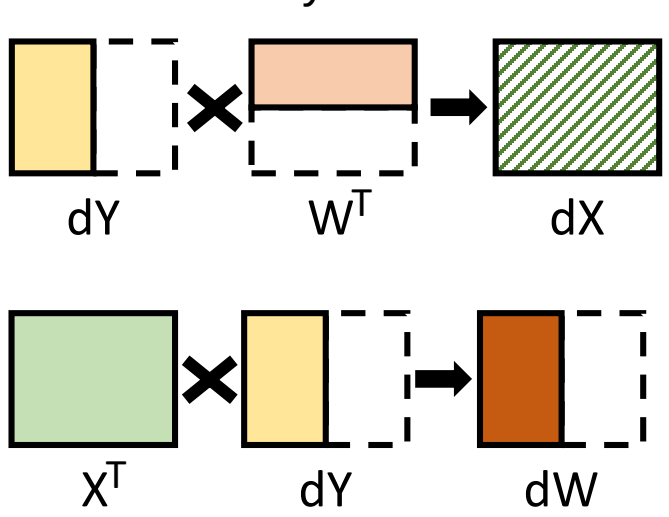


Trade-off

W is shared by all partitions (+)

Write traffic for **partial sum** of **dW** (−)

Divided by dimension **N**

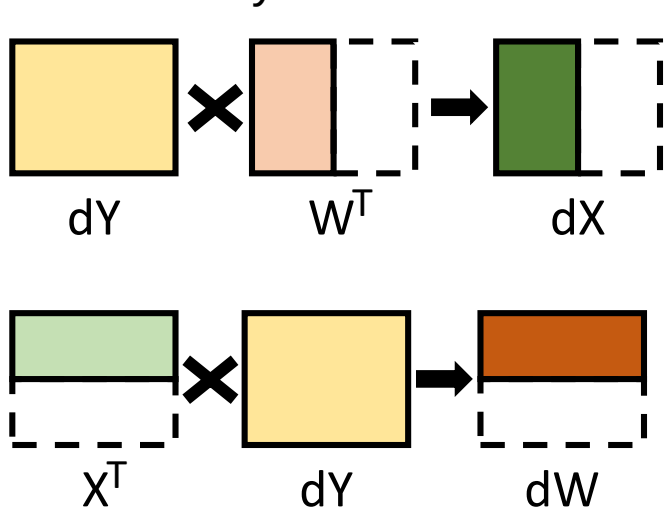


Trade-off

X is shared by all partitions (+)

Write traffic for **partial sum** of **dX** (−)

Divided by dimension **K**



Trade-off

No partial sum of **dX** and **dW** (+)

Inefficient for large **dY** (−)

Evaluation Methodology

- Cycle-level simulation modified from *SCALE-Sim
- Server-level and edge-level NPU configurations
  - Large NPU (Google TPUv4), Small NPU (ARM Ethos N77)
- Workloads : 9 models from various fields

Simulated NPU configurations

	Large NPU	Small NPU
Compute Unit	1-8 x (128x128 PE)	1 x (45x45 PE)
SPM Size	8MB per core	1MB
Clock Rate	1050MHz	1024MHz
Batch Size	8 (256 per TPUv4-8)	4

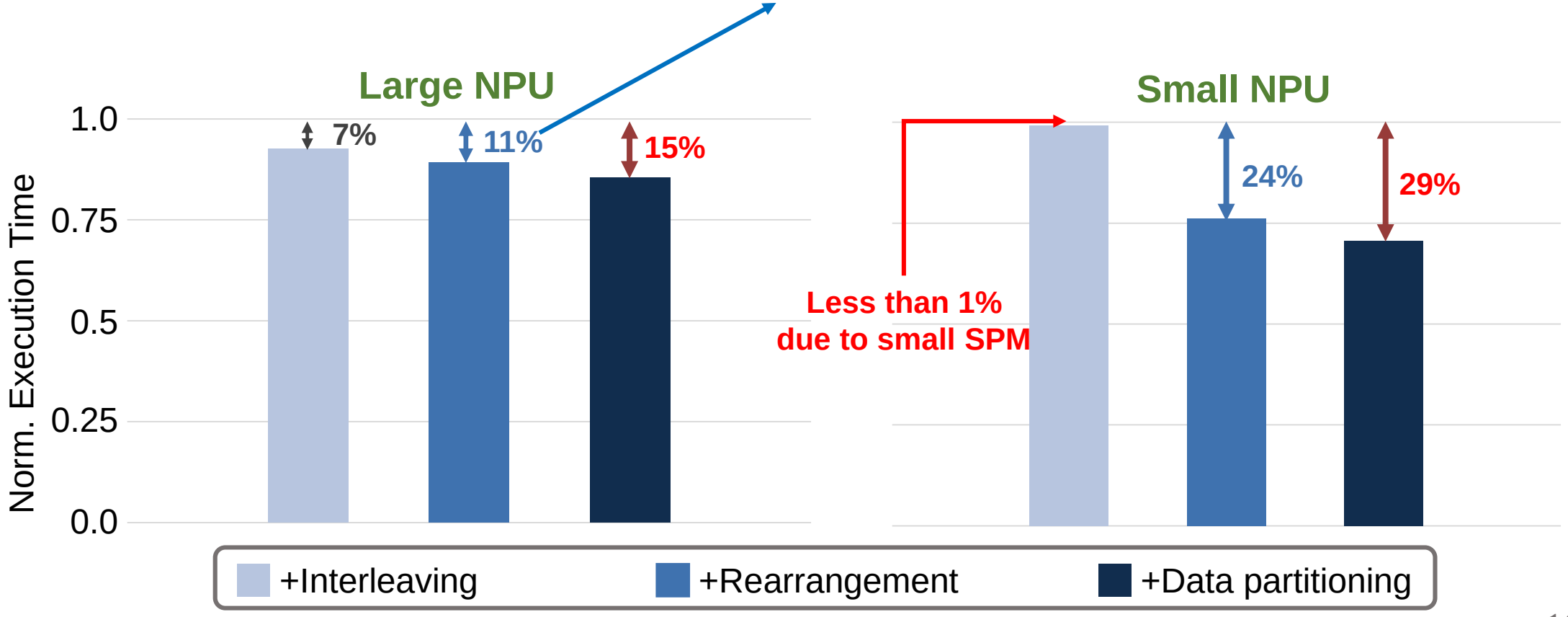
Workloads

Type	Model (Abbr)
Computer Vision	FasterRCNN (rcnn) Googlenet (goo) ResNet-50 (res) Mobilenet (mob) YOLO (v5/v2-tiny) (yolo)
Natural Language Processing	BERT (large/tiny) (bert) T5 (large/small) (T5)
Recommendation System	NCF-recommendation (ncf) DLRM (dlrm)

- A systematic methodology for characterizing scalability of dnn accelerators using scale-sim (ISPASS 2020)

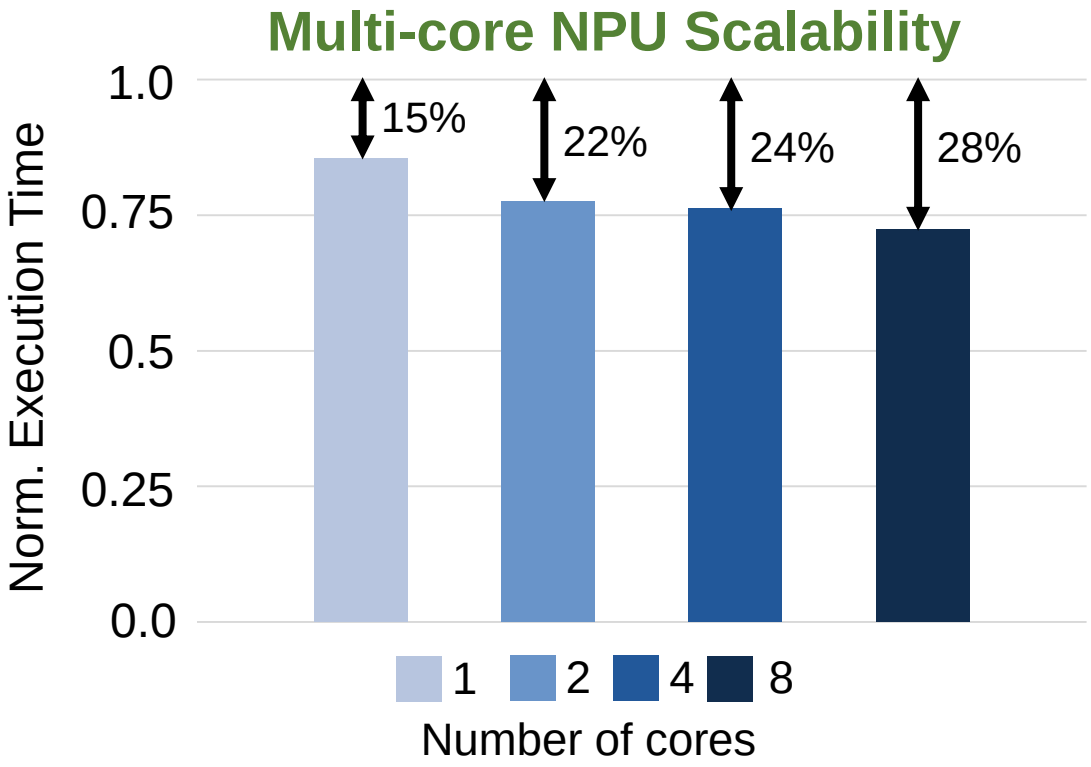
Evaluation Result (Single NPU)

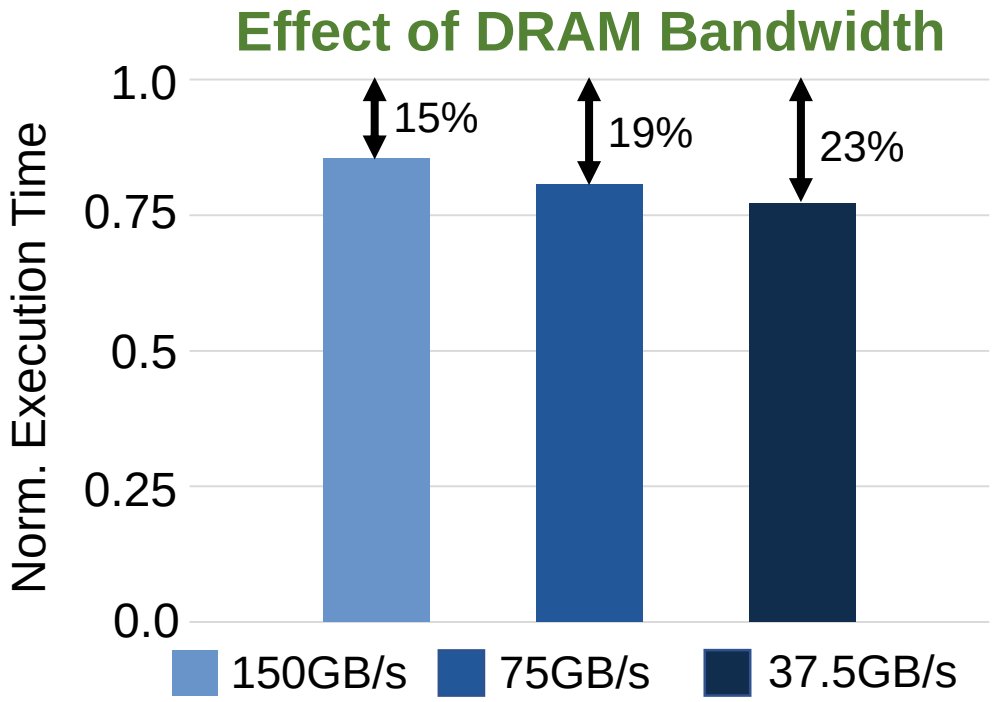
- Performance improvement: **15%** (Large NPU), **29%** (Small NPU)
- Performance improvement ∝ Reduced data traffic through data reuse



Evaluation Results (Large NPU)

- Interleaved gradient order is also efficient for multi-core NPU
  - Normalized to the baseline with the same number of cores
- Limited off-chip memory bandwidth → Greater performance improvement





Conclusion

- Efficiently utilizing SPM in NPU is crucial in DNN training

- Find a new data reuse chance by fusing gradient computations
  - **Reuse output gradient (dY)** in independent operations
  - Inspect **optimal memory access order** for interleaved gradient order
  - Propose a novel **computation partitioning** for interleaved gradient order

- **Improve performance by 15% and 29%** for server NPU and edge NPU
  - Without changing the accuracy or the amount of computation