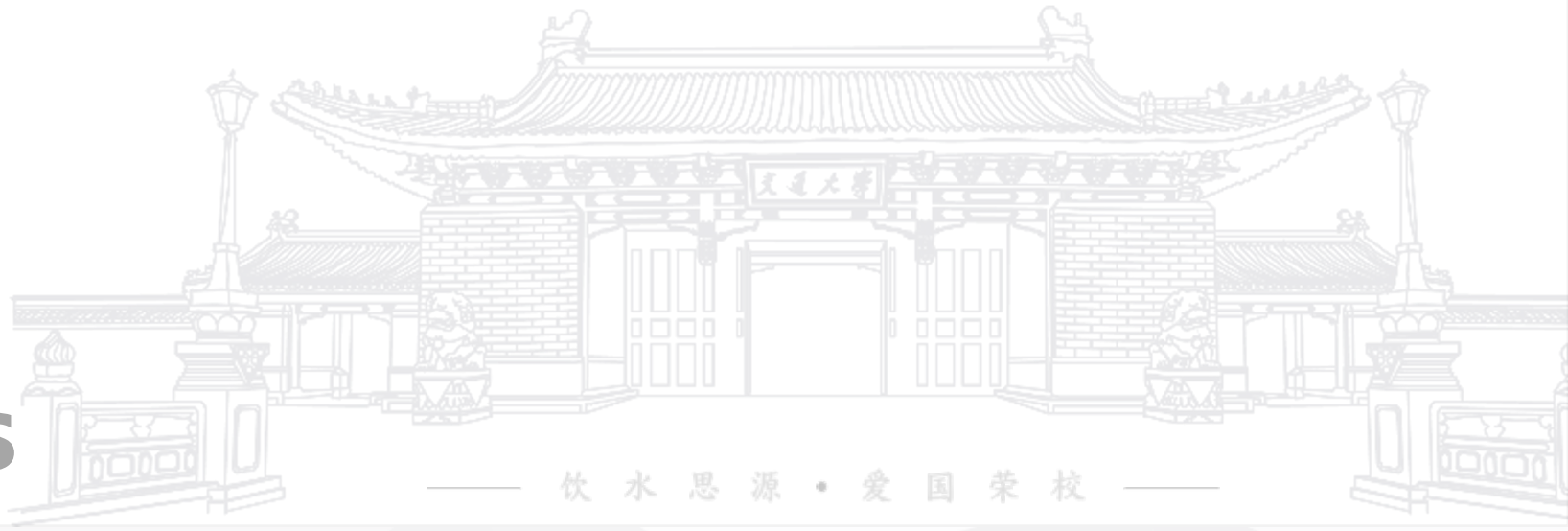
Towards Compute-Aware In-Switch Computing for LLMs Tensor-Parallelism on Multi-GPU Systems

Chen Zhang[1], Qijun Zhang[1]*, Zhuoshan Zhou[1], Yijia Diao[1],
Haibo Wang[2], Zhe Zhou[2], Zhipeng Tu[2], Zhiyao Li[2],
Guangyu Sun[3], Zhuoran Song[1], Zhigang Ji[1], Jingwen Leng[1]*, Minyi Guo

[1]Shanghai Jiao Tong University, [2]Huawei Technologies Co. Ltd., [3]Peking University

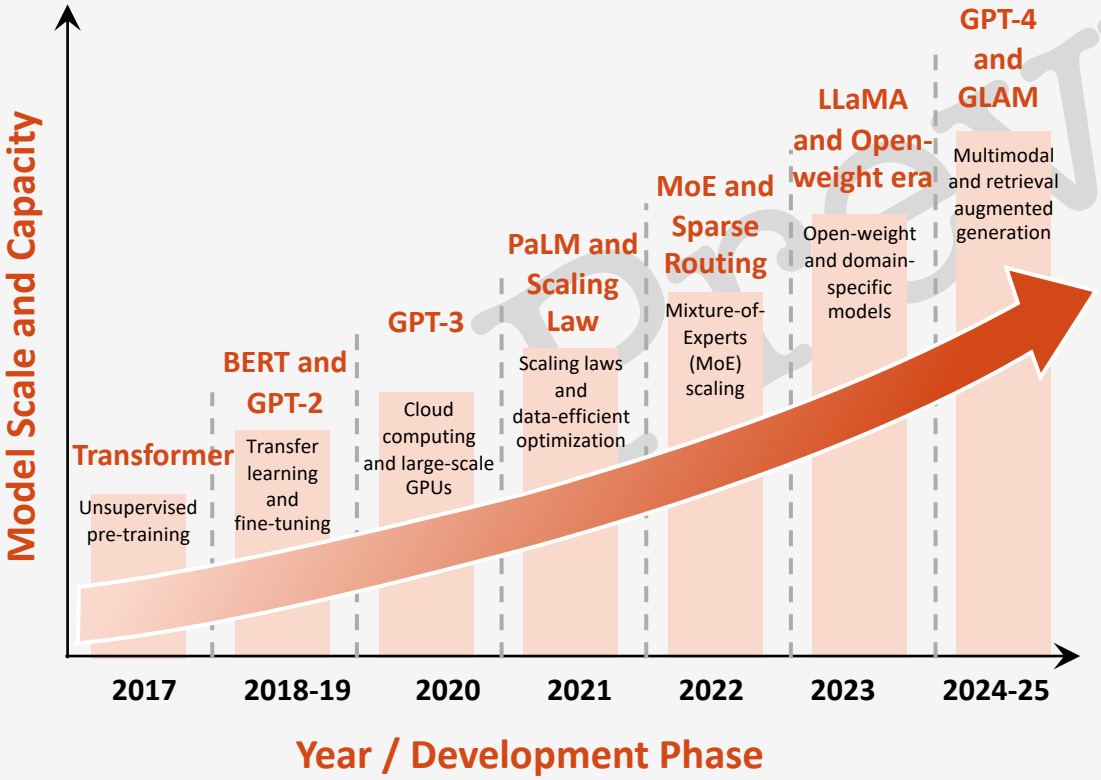
上海交通大学
SHANGHAI JIAO TONG UNIVERSITY

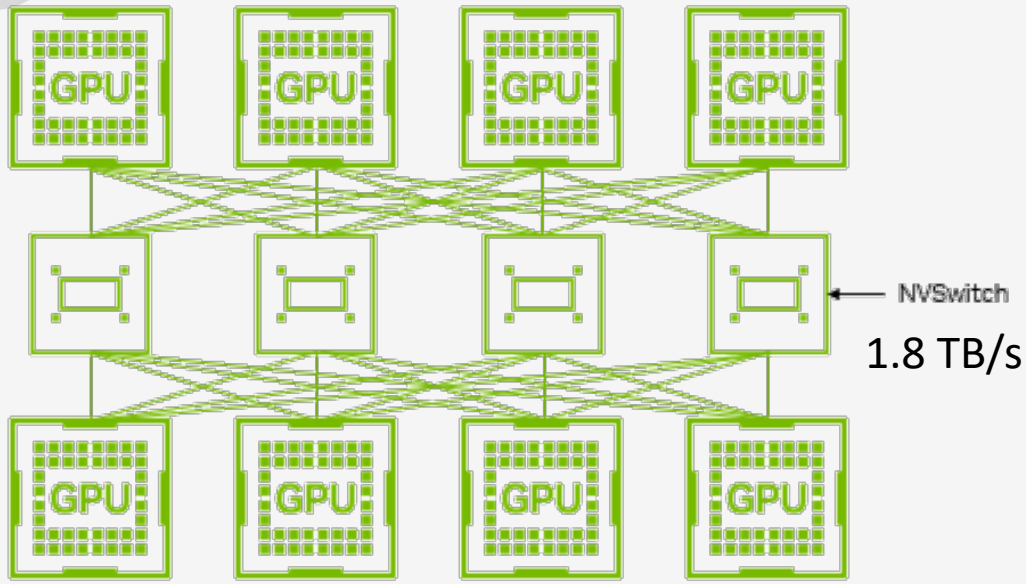
1. Background
2. Motivation
3. Solutions
4. Experiments
5. Conclusion



LLM Scaling Laws Drive Multi-GPU System Scale-Up

- **LLM scaling laws incentivize ever-larger models, pushing parameter growth beyond single-GPU capacity and motivating tightly coupled multi-GPU nodes interconnected by NVLink/NVSwitch.**





Ghaffar Nia, N., Amiri, A., Luo, Y. et al. Ethical perspectives on deployment of large language model agents in biomedicine: a survey. AI Ethics 2026.

Tensor Parallelism Contributes to 97% Traffic

- **Unlike DP and PP, TP incurs collective communication at every forward and backward pass, e.g., All-Reduce (basic TP) or Reduce-Scatter / All-Gather (TP+SP), making its communication overhead the dominant bottleneck as models and GPU counts scale.**

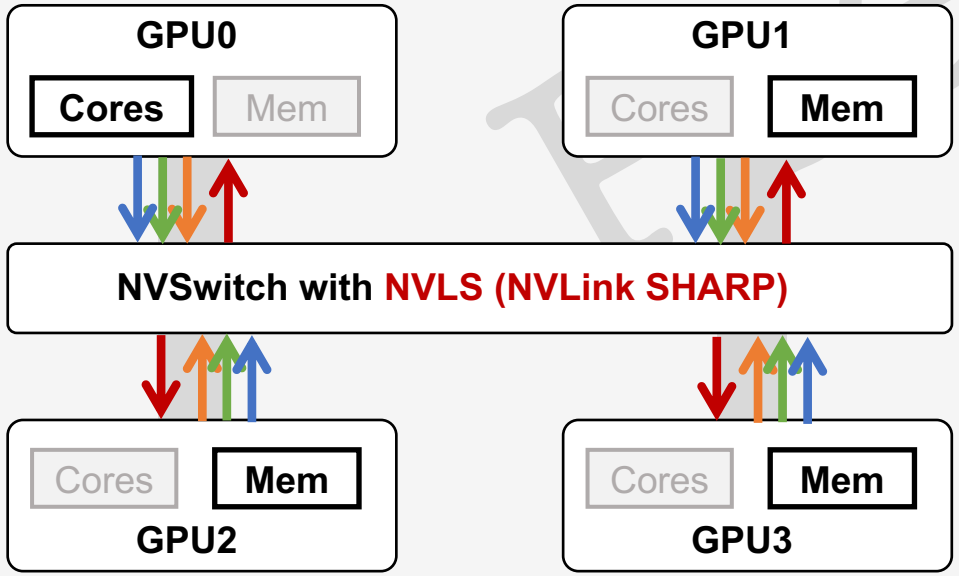
Communication Volume and Bandwidth Requirements

Parallelism	Traffic per Round (MB)	# Rounds	Total Traffic (GB)
PP	96	120	11
DP	1277	32	40
EP	6	7680	42
TP	180	7680	1350
SP	762	11520	8573

H. Liao, B. Liu, X. Chen, *et al.*, “Ub-mesh: a hierarchically lo- calized nd-fullmesh datacenter network architecture,” *HotChips* 2025.

NVLink SHARP (a.k.a. NVLS)

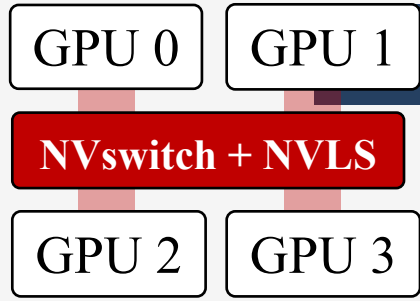
- **Dedicated to NVSwitch-based Multi-GPU Systems**
  - Introduced since Hopper Architecture
- **Native, instruction-level support for in-switch operations**
  - **multimem.*** instructions via **inline PTX** to implement the corresponding collectives
- **2–8× speedups for collectives compared to GPU-driven implementations**



Instructions	Accelerated Collectives
multimem.st	AllGather
multimem.ld_reduce	Reduce-Scatter
multimem.red	AllReduce

Klenk B, Jiang N, Thorson G, et al. An in-network architecture for accelerating shared-memory multiprocessor collectives, ISCA 2020.

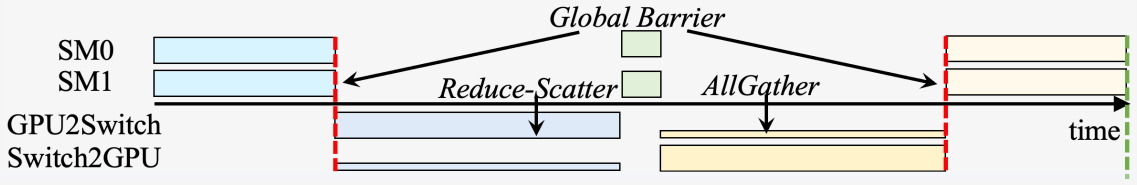
CAIS: Computation-aware In-Switch Computing



Existing NVLS design is

Communication Centric

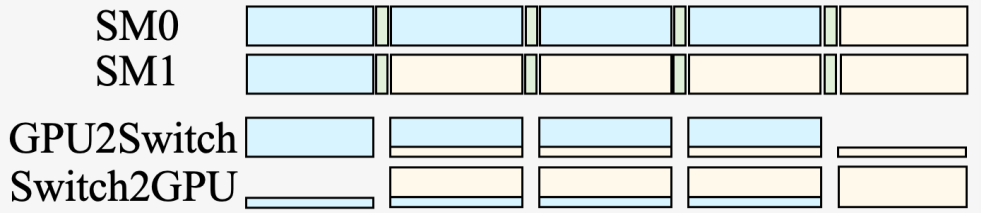
Instructions	Accelerated Collectives
multimem.st	**AllGather**
multimem.ld_reduce	**Reduce-Scatter**
multimem.red	**AllReduce**



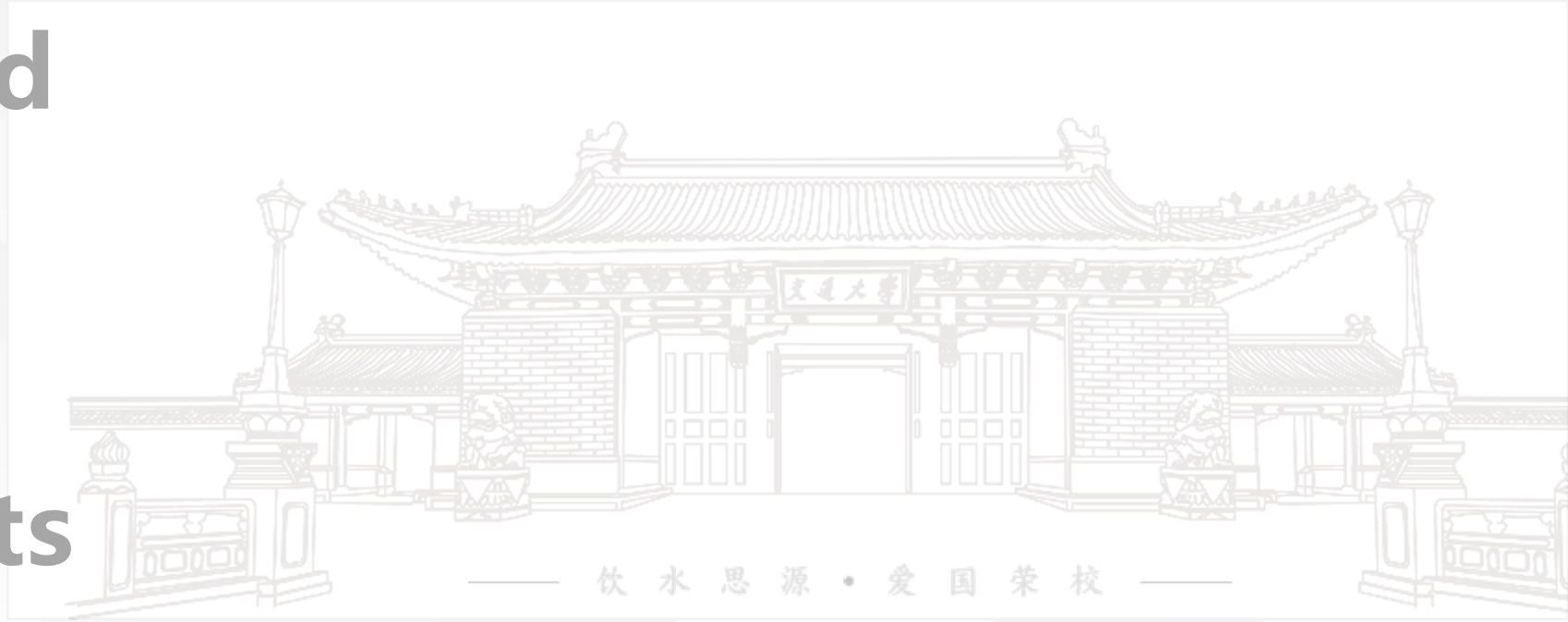
Our proposal:

Computation Aware

Tenor Parallel Operations	
TP with SP	**AllGather** + GEMM
	GEMM + **ReduceScatter**
Basic TP	**AllReduce** + GEMM
	GEMM + **AllReduce**

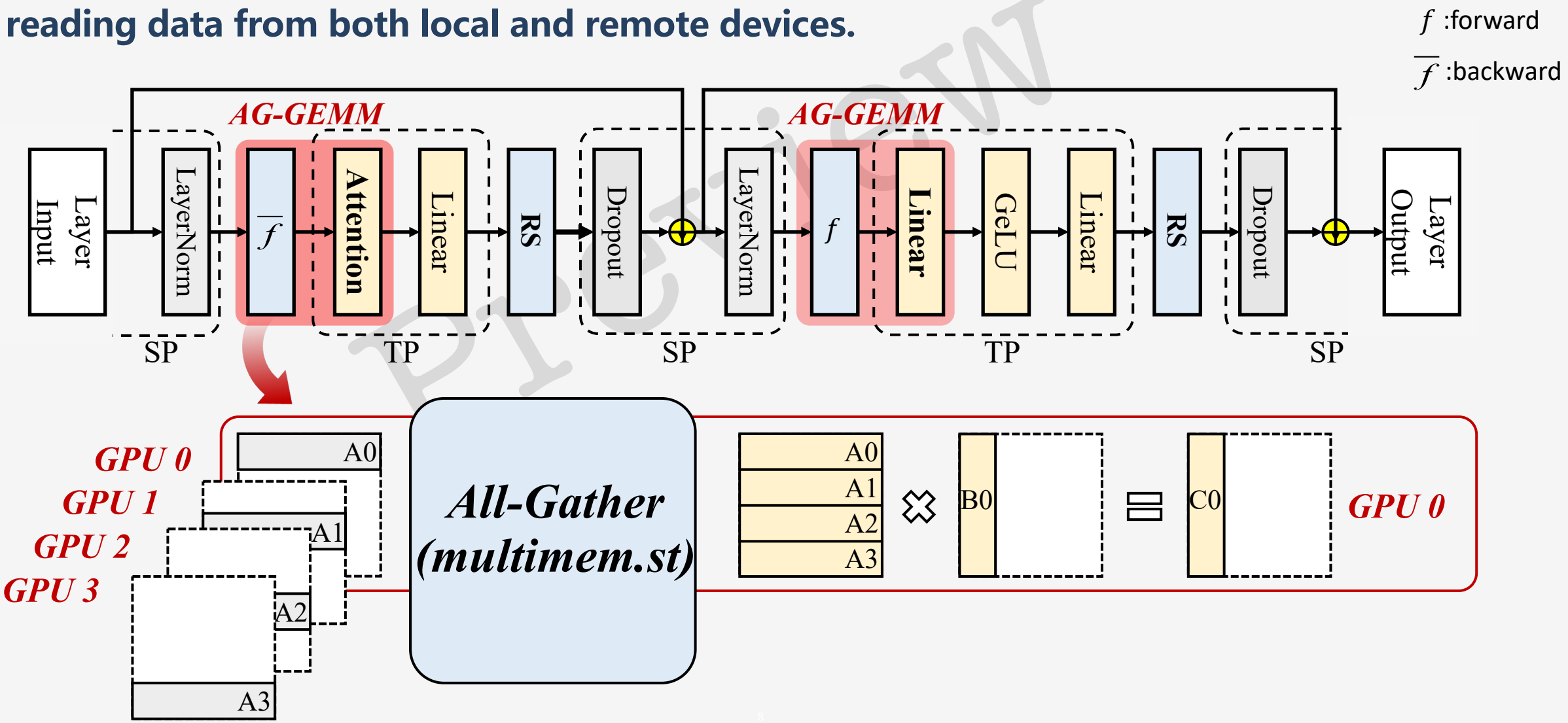


1. Background
2. Motivation
3. Solutions
4. Experiments
5. Conclusion



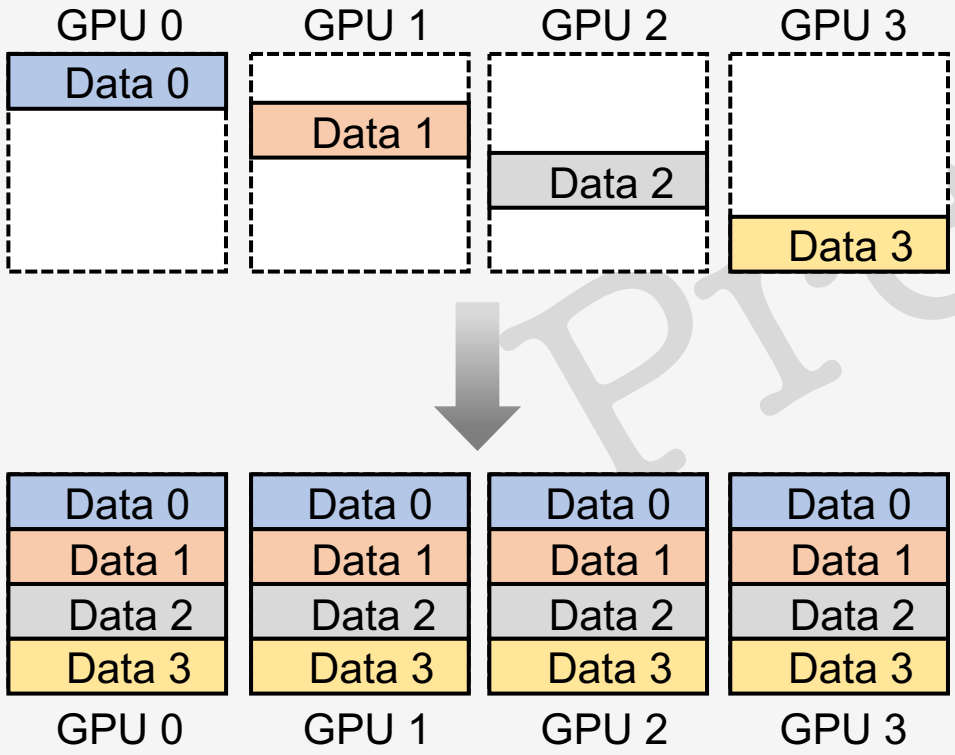
AllGather + GEMM Example

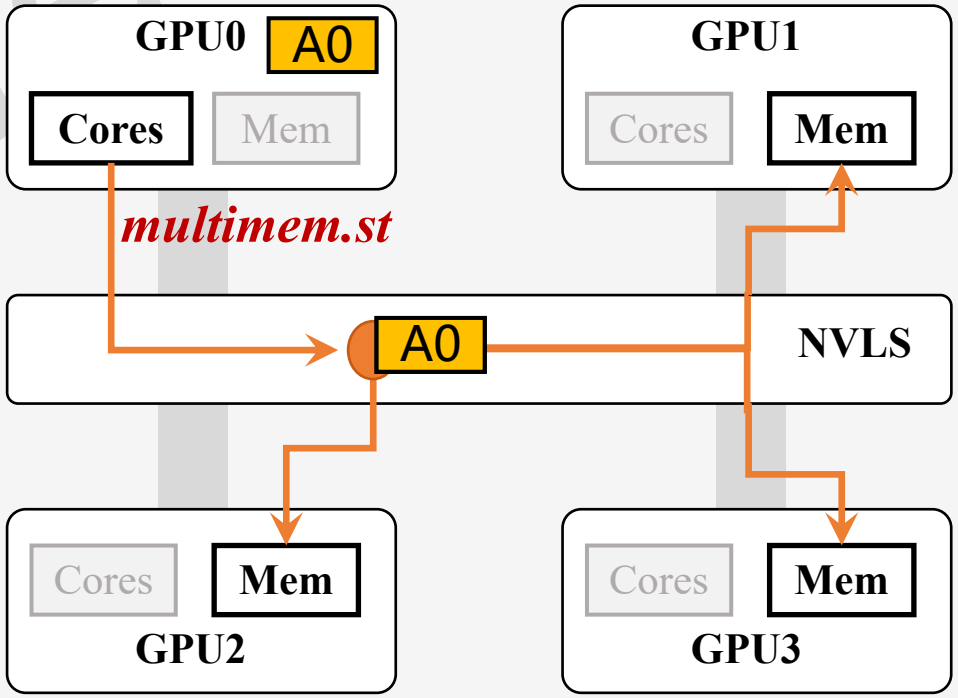
- All-Gather is often followed by a GEMM (e.g., attention or FFN), which requires reading data from both local and remote devices.



AllGather works in PUSH mode

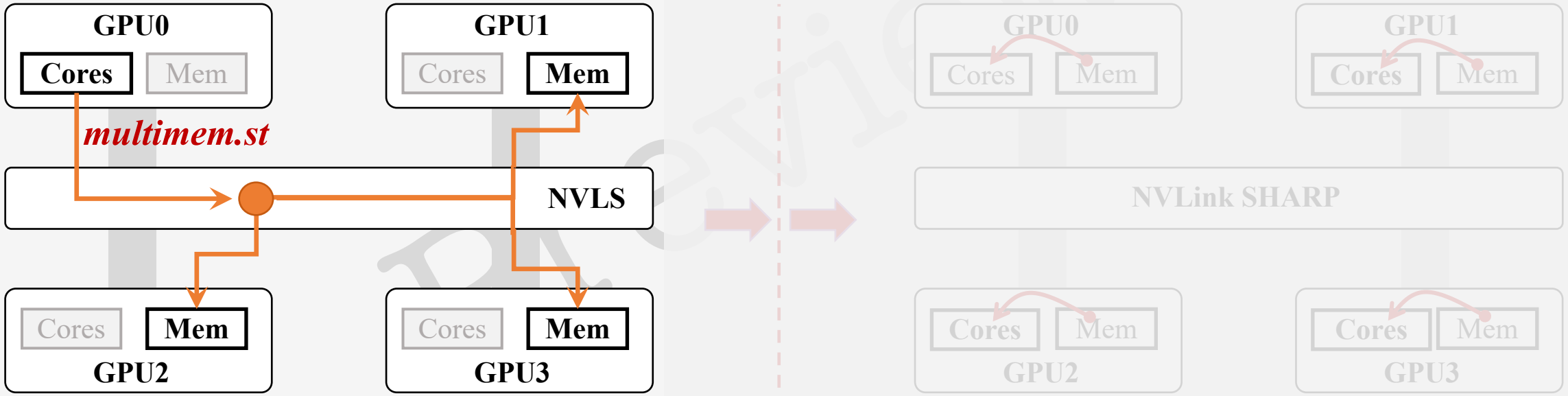
- **Each GPU pushes its local data to other GPUs via the *multimem.st* instruction**
- Inline PTX for in-switch computing function
- Automatic Data Duplication/Reduction in the Switch





Execution Details

All-Gather* + *GEMM*: *remote store* → *syn.* → *local read



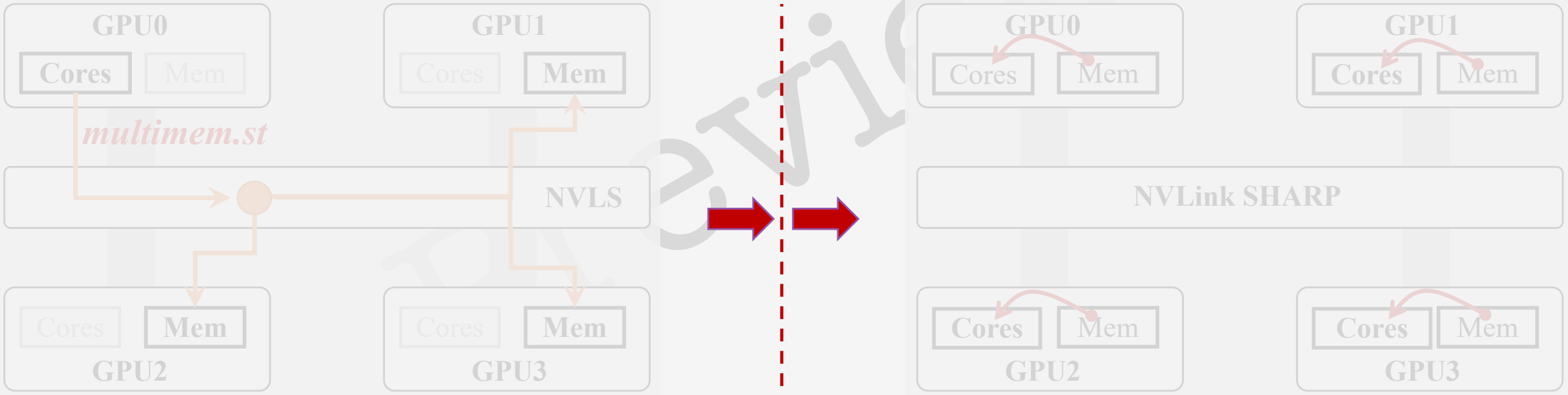
① **Producer GPU: proactively push via** *multimem.st*

② Global Synchronization

③ Consumer GPU: TBs read local memory in GEMM Kernel

Execution Details

All-Gather + GEMM: *remote store* → *syn.* → *local read*



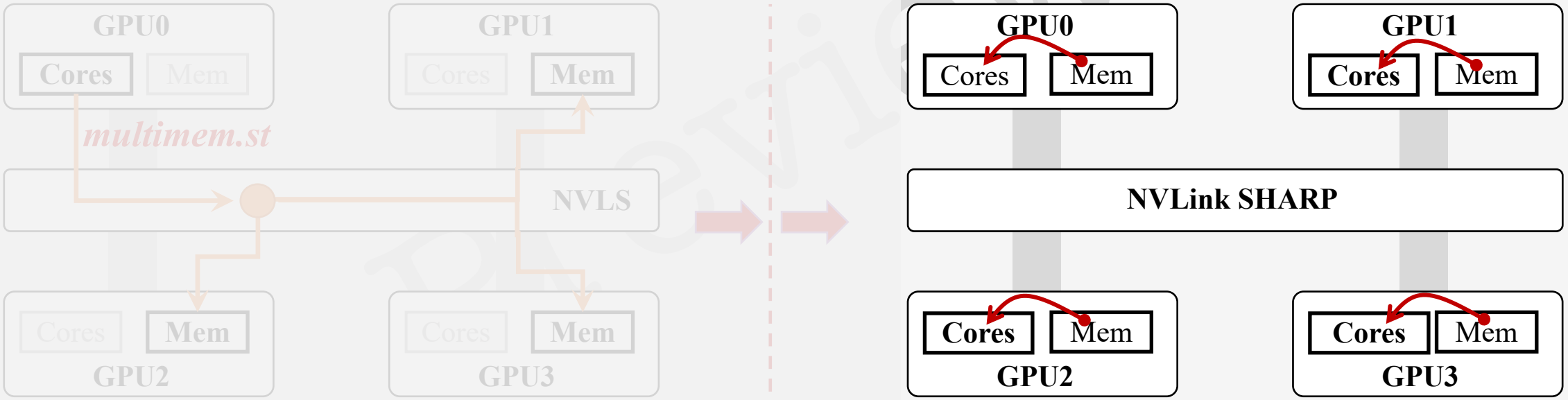
① Producer GPU: proactively push via *multimem.st*

② Global Synchronization

③ Consumer GPU: TBs read local memory in GEMM Kernel

Execution Details

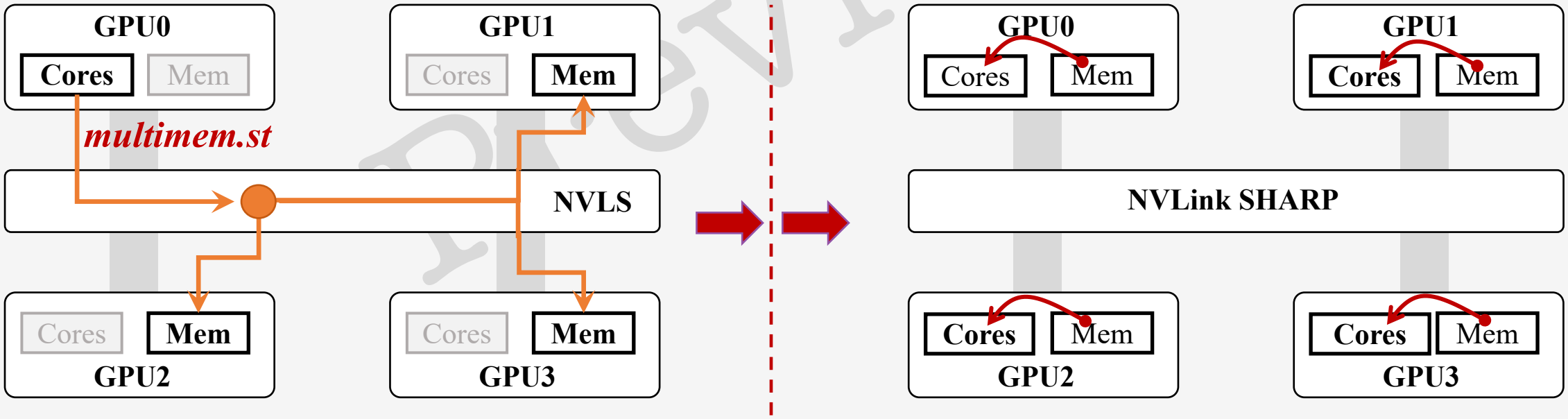
All-Gather + GEMM: *remote store → syn. → local read*



In summary: Producer PUSH, then Consumer READ

- The GEMM computation requires memory reads from both local and remote devices, yet *multimem.st*, the in-switch instruction used for All-Gather, operates in push mode.

All-Gather + GEMM: remote store → syn. → local read

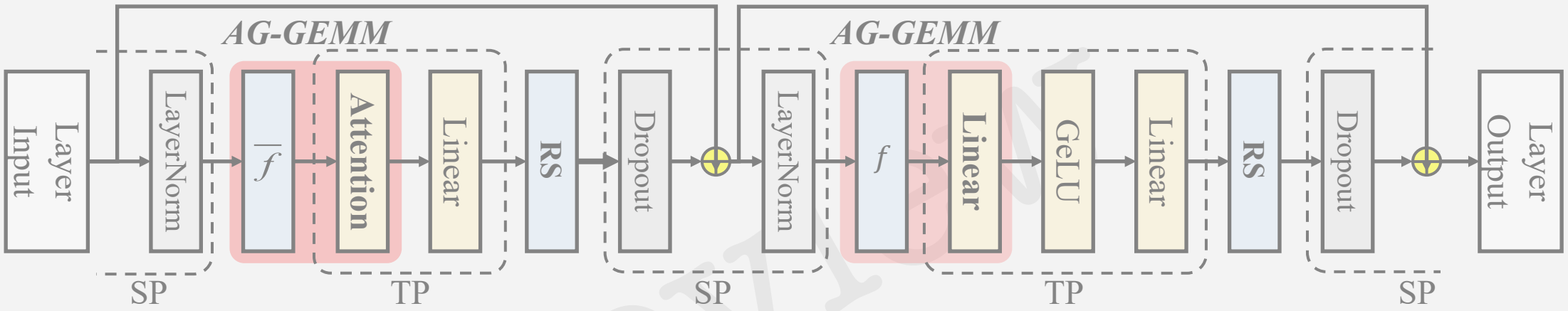


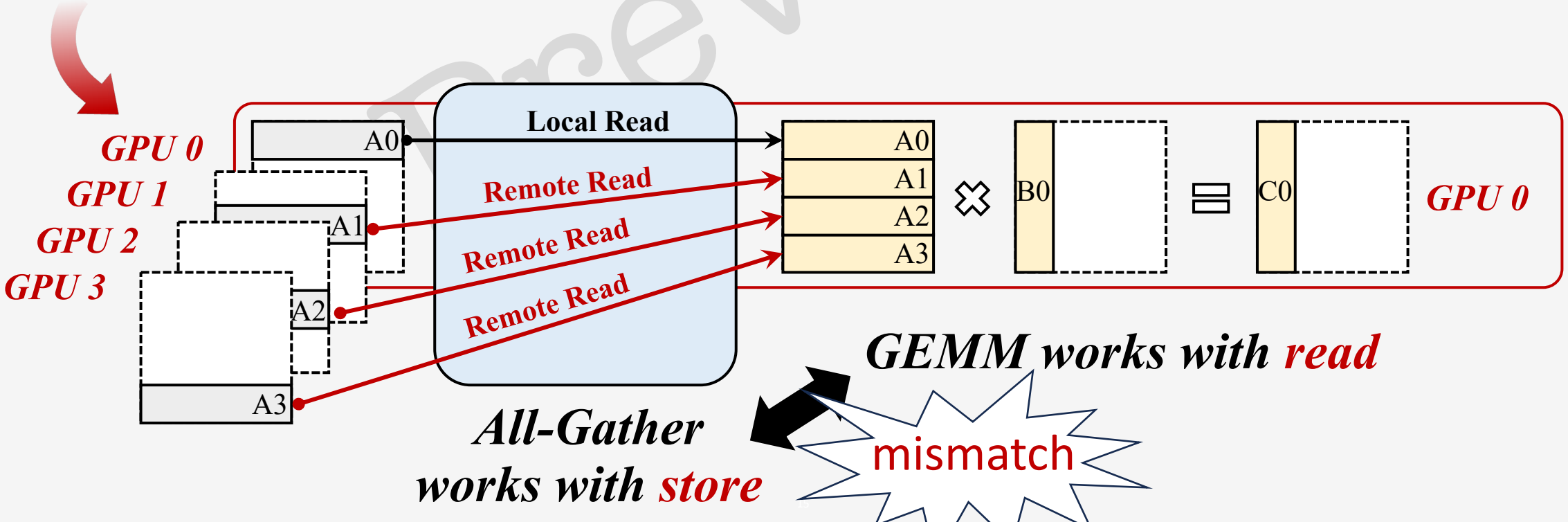
① Producer GPU: proactively push via *multimem.st*

② Global Synchronization

③ Consumer GPU: TBs read local memory in GEMM Kernel

From GEMM side: Rethinking All-Gather+GEMM

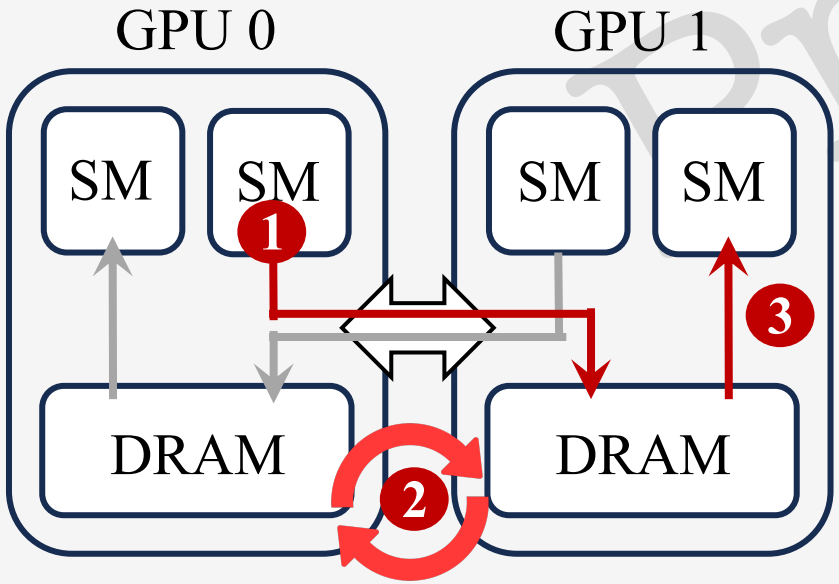




From GEMM side: Rethinking All-Gather+GEMM

- Issue 1: isolated data-transfer & compute in separate phases (GPU 0 vs. 1)
- Issue 2: frequent global synchronization
- Issue 3: more programming effort & software cost

→ in-efficiency in fine-grained compute-communicate overlap

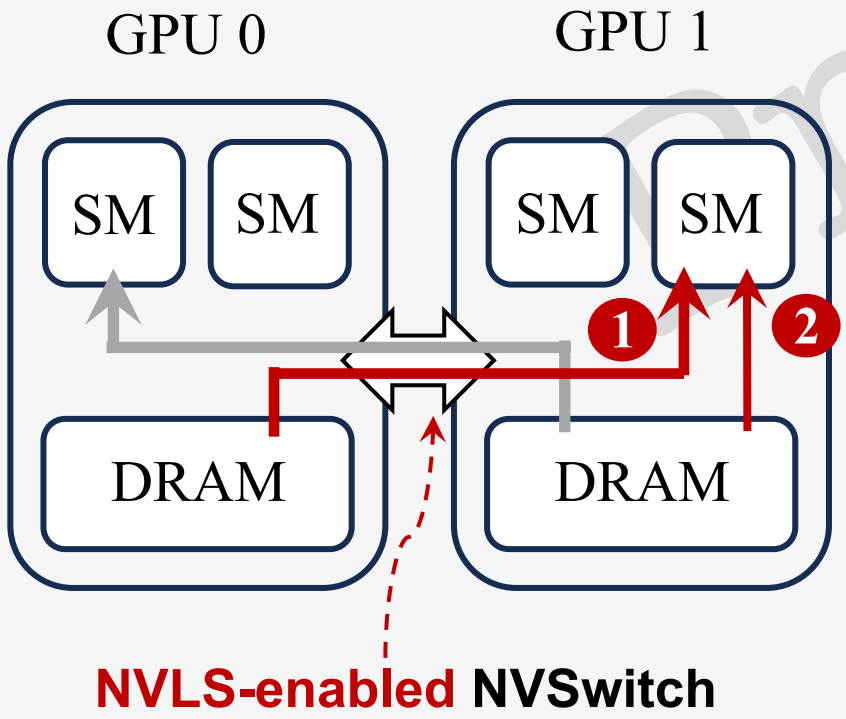



```
__global__ AG_GEMM_MultimemSt(......) {
  // Perform AllGather with multimem.st
1 AllGather_MultimemSt();
  // Guarantee remote GPU has pushed data to local DRAM
2 Global_Barrier();
  // Load data to shared memory from local DRAM
3 Local_Load();
  // Guarantee data has been in shared memory
  Local_Barrier();
  // Perform computation with data in shared memory
  Computation();}
```

What if AllGather in "Pull Mode" ?

- Benefit 1: ~~disjoint~~ data-move & compute **unified** in one single end
- Benefit 2: ~~frequent~~ **less** global synchronization
- Benefit 3: ~~more~~ **less** programming effort & software cost

→ **smoother data-flow & compute-communicate overlap**

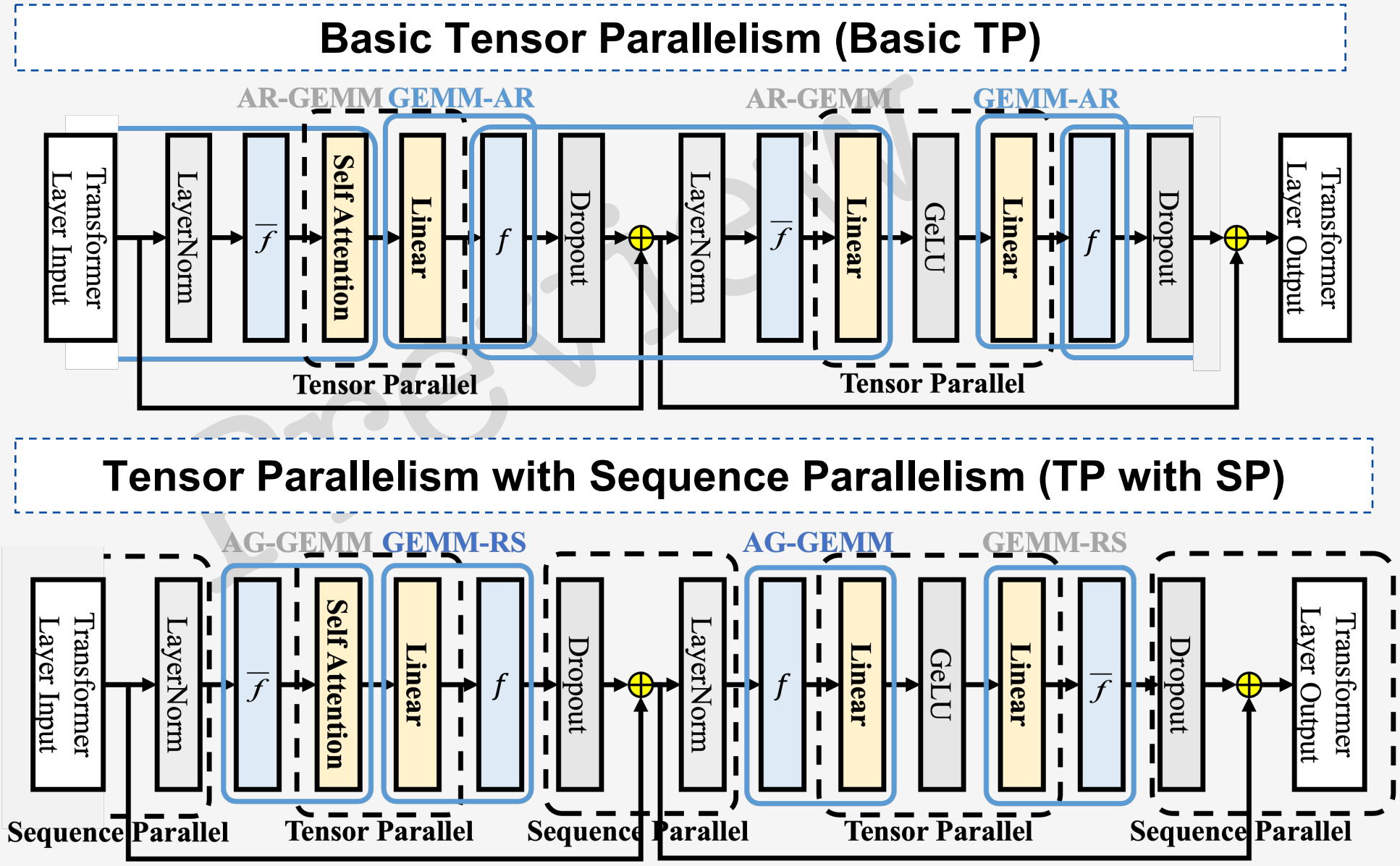



```
__global__ AG_GEMM_Load(......) {
  // Directly read remote data to shared memory with load
1 Remote_Load();
  // Load data to shared memory from local DRAM
2 Local_Load();
  // Guarantee data has been in shared memory
  Local_Barrier();
  // Perform computation with data in shared memory
  Computation();
}
```

(TP | SP) & (Forward | Backward)

f :forward

$\overline{f}$:backward



Semantic Mismatch between Comp. & Comm.

Existing NVLink SHARP Primitives

Existing Primitives	Collective Op	Comm. Mode
multimem.st	**AllGather**	**Push**
multimem.ld_reduce	**Reduce-Scatter**	**Pull**
multimem.red	**AllReduce**	**Push**

Tensor Parallelism Requirement

TP Operation		Required Memory Semantics
TP with SP	**AG-GEMM**	**Read Memory**
	GEMM-RS	**Write Memory**
Basic TP	**AR-GEMM**	**Read Memory**
	GEMM-AR	**Write Memory**

Mis-Matches

- ✗ **"Read-Push"**
- ✗ **"Write-Pull"**
- ✗ **"Read-Push"**

CAIS: Re-design NVLS in a Compute-aware Fashion

Existing NVLink SHARP Primitives

Existing Primitives	Collective Op	Comm. Mode
multimem.st	AllGather	Push
multimem.ld_reduce	Reduce-Scatter	Pull
multimem.red	AllReduce	Push

Our Proposal

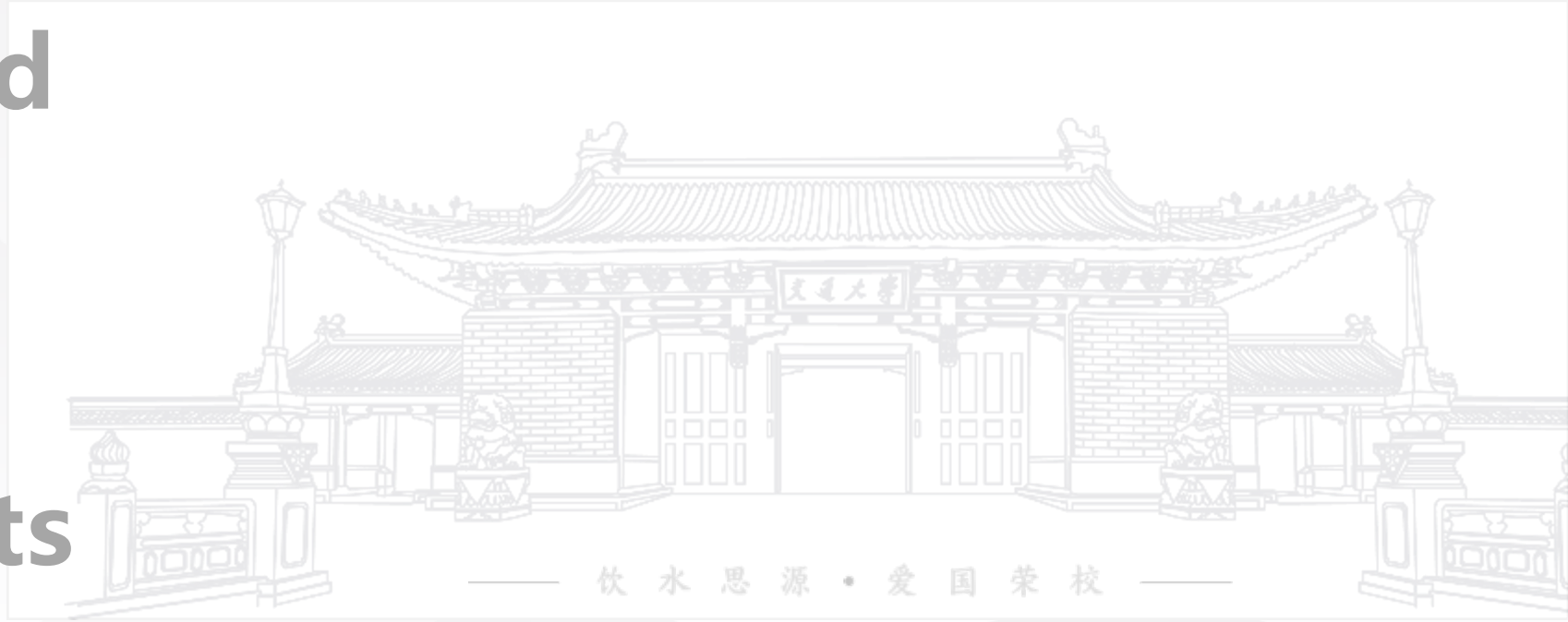
Primitives	Comm. Mode
ld.cais	Pull
red.cais	Push

Tensor Parallelism Requirement

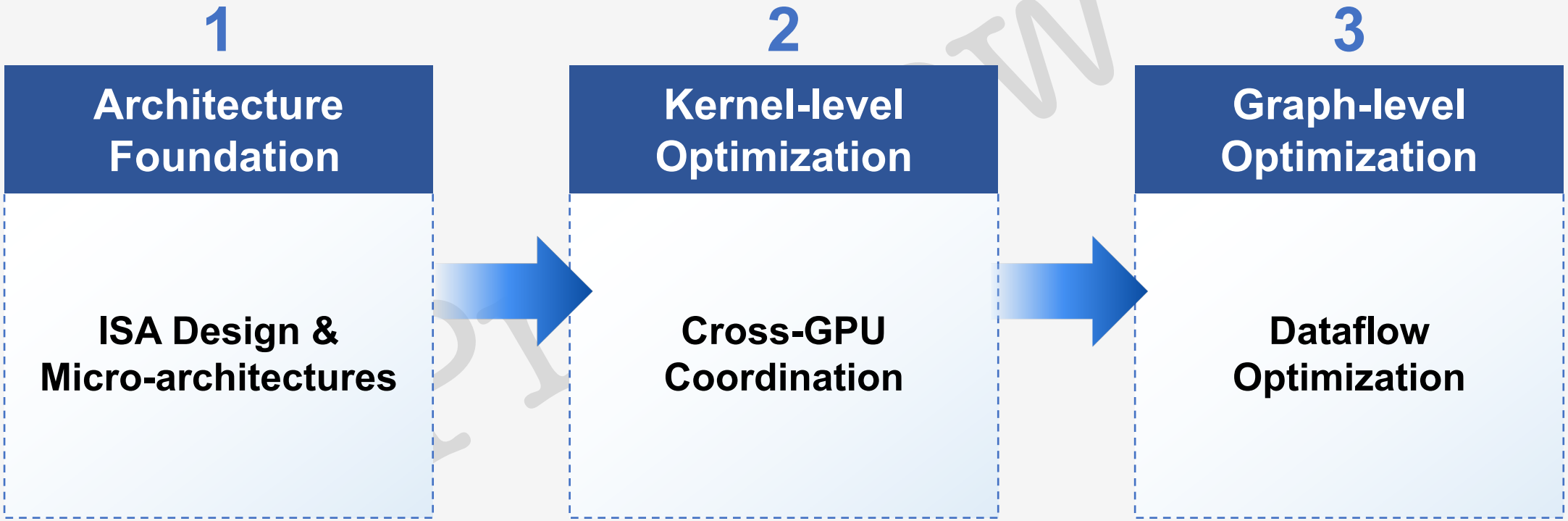
TP Operation		Required Memory Semantics
TP with SP	AG-GEMM	Read Memory
	GEMM-RS	Write Memory
Basic TP	AR-GEMM	Read Memory
	GEMM-AR	Write Memory

Match!

1. Background
2. Motivation
3. Solutions
4. Experiments
5. Conclusion



Three Key Pieces in CAIS

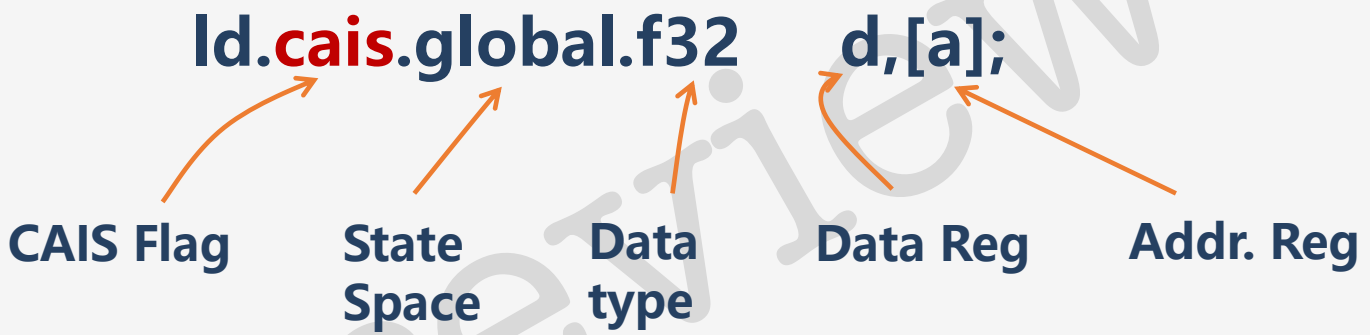


Tech 1： ISA Extensions

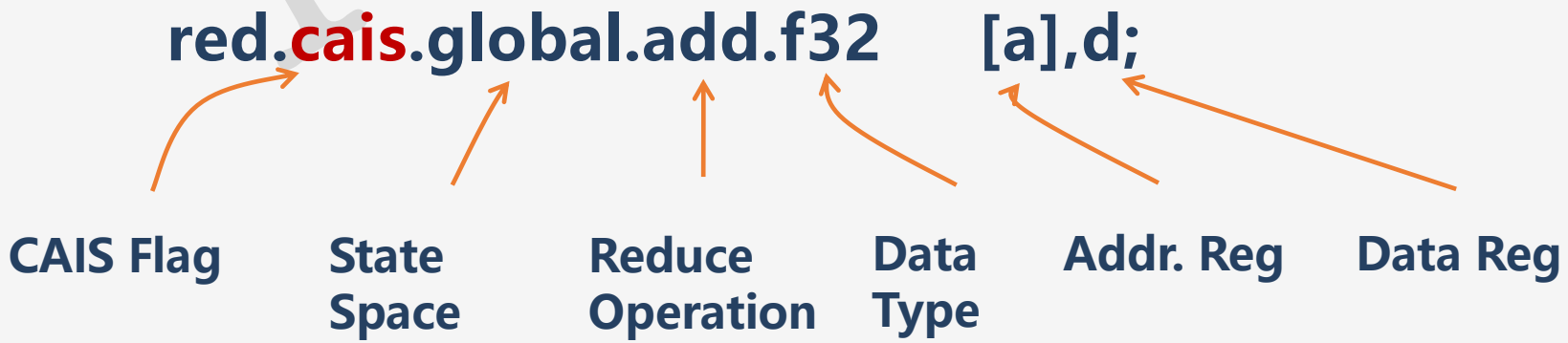
	Original *Inst.* in Existing Arch	The Proposed *Inst.* in CAIS
Instruction 1: *load*	ld.global.f32	ld.cais.global.f32
Instruction 2: reduce	red.global.add.f32	red.cais.global.add.f32

Tech 1： ISA Extensions

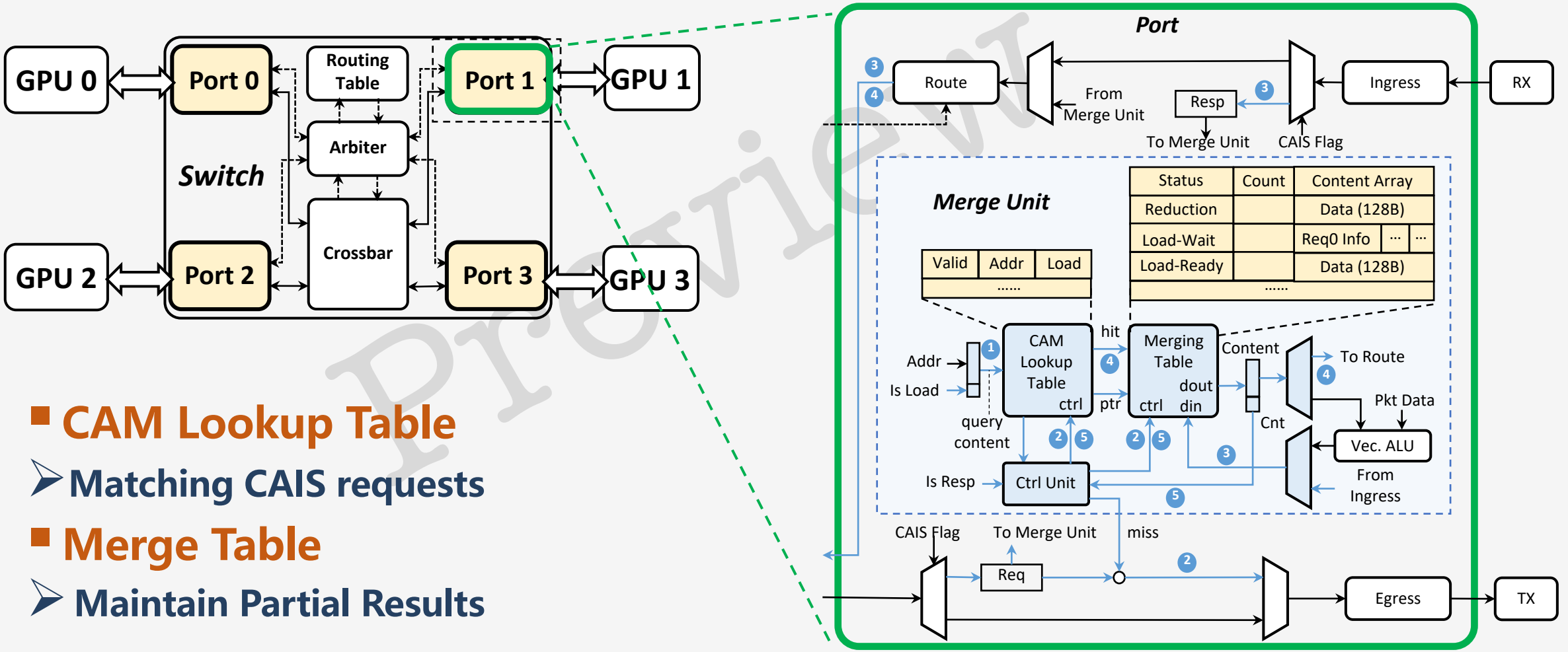
- Instruction 1: ***load***



- Instruction 2: ***reduce***

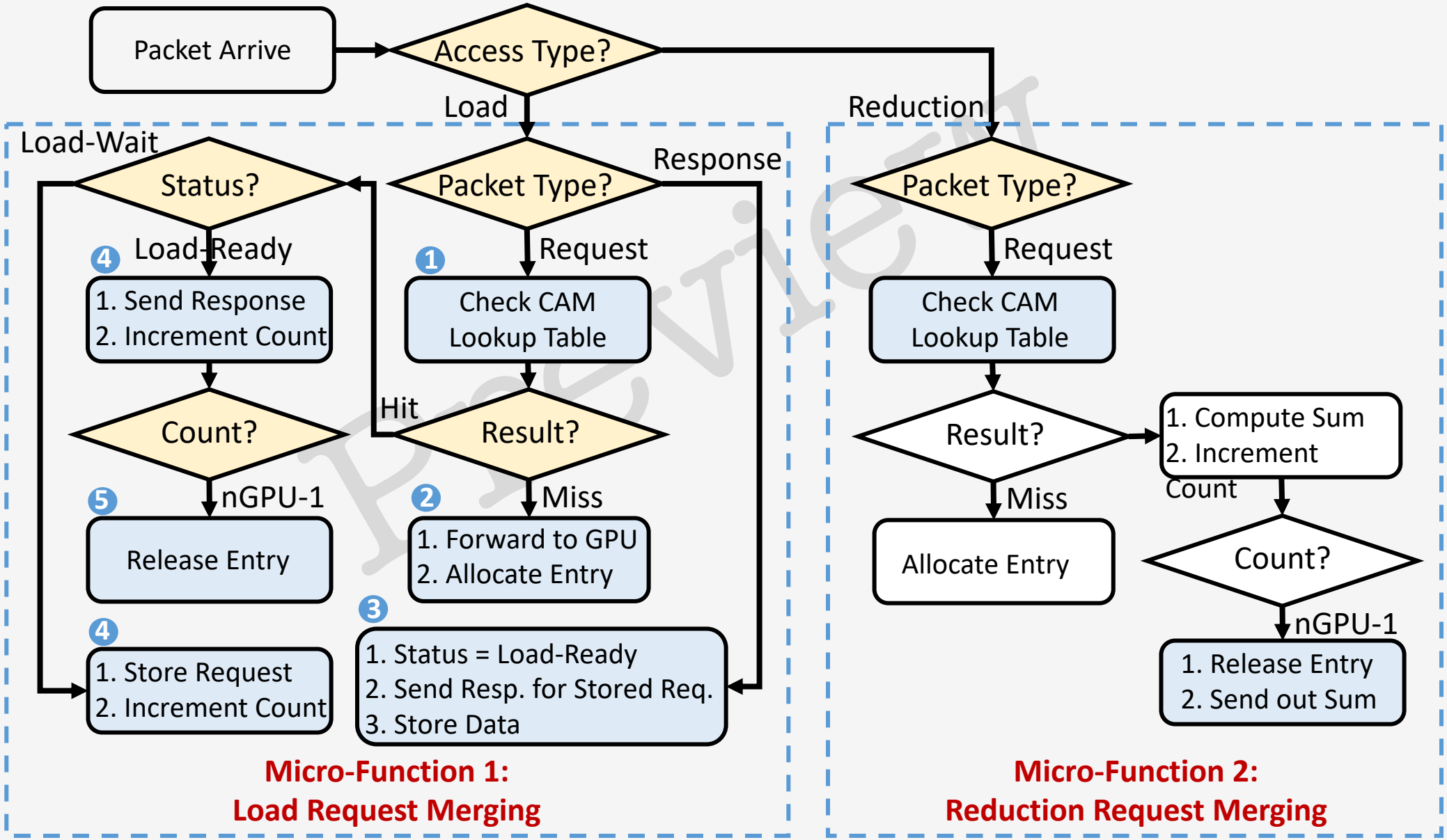


Tech 1: Micro-architectures



- **CAM Lookup Table**
  - Matching CAIS requests
- **Merge Table**
  - Maintain Partial Results

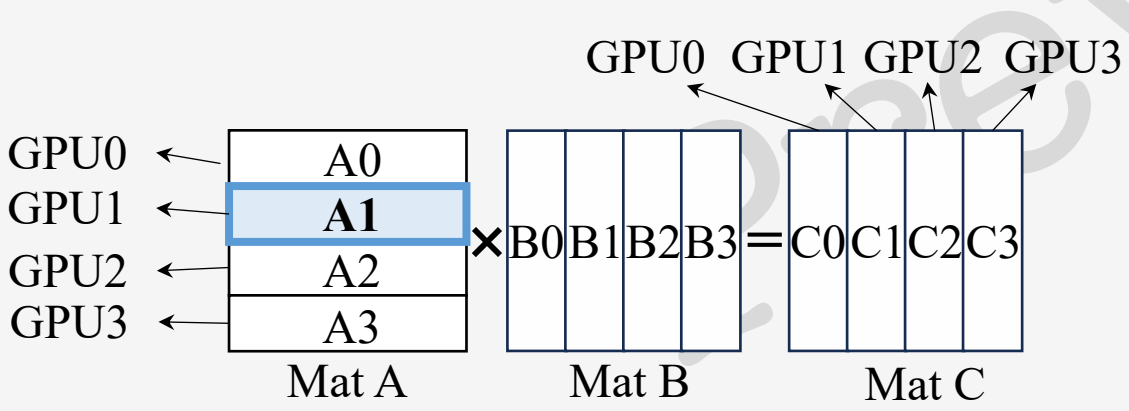
Tech 1: In-Switch Micro-Functions



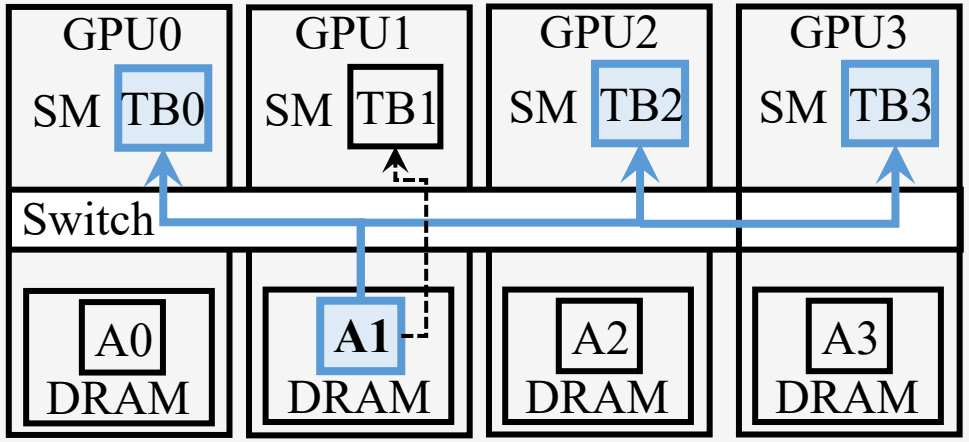
Tech 2: Cross-GPU TB Coordination

- **AllGather-GEMM Example:**

"A1"data is requested by multiple GPUs (0/2/3)



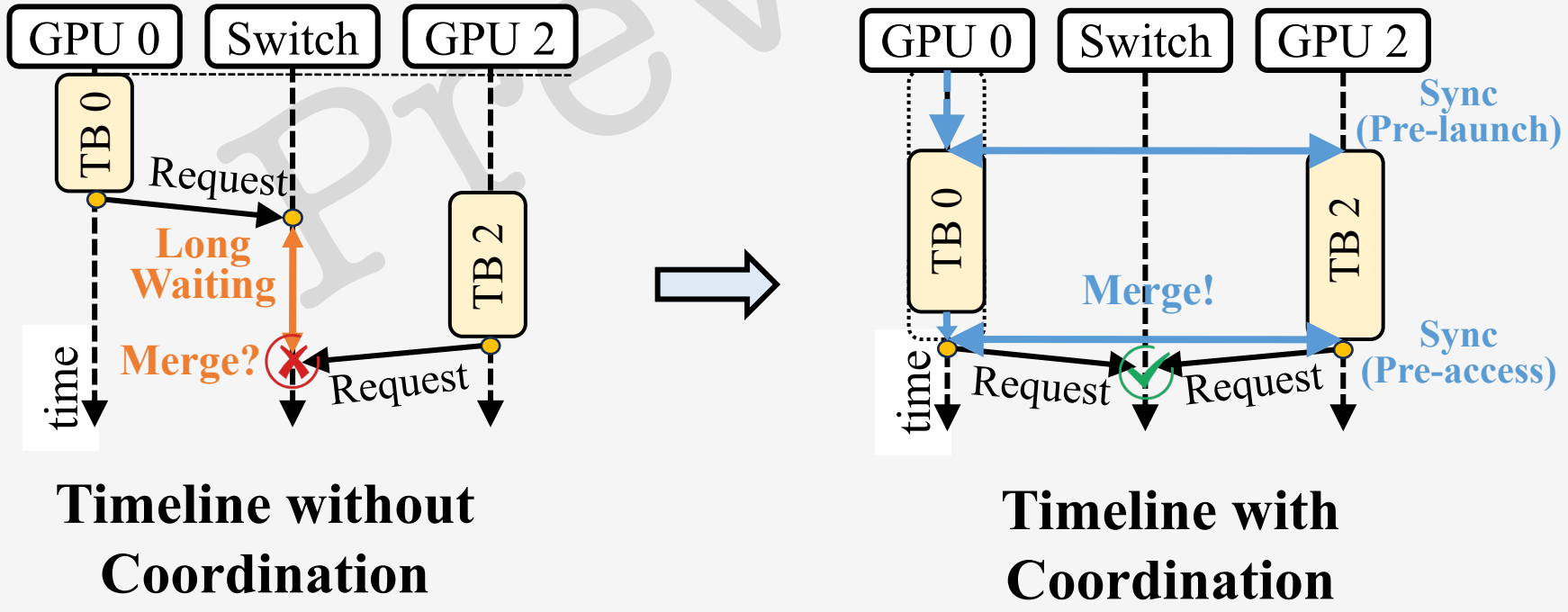
AllGather-GEMM Example



In-switch Request Broadcasting

Tech 2: Cross-GPU TB Coordination

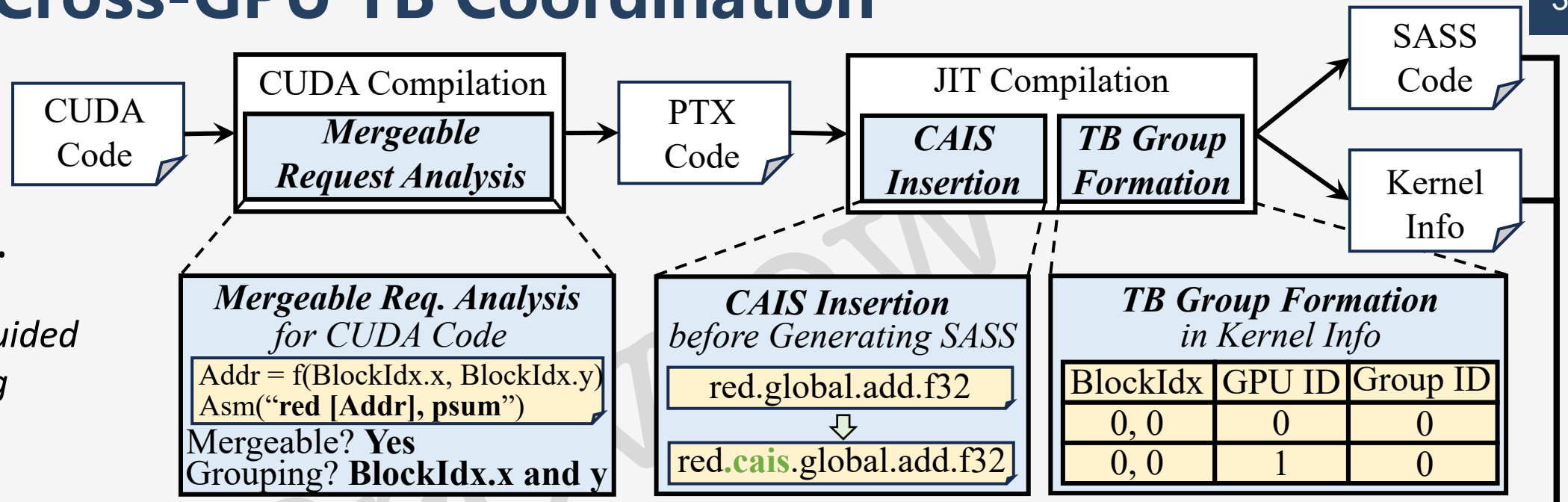
- Without coordination, mergeable load or reduction requests from different GPUs may **arrive** at the switch **at different times**, resulting in missed merging opportunities or buffer pressure due to delayed aggregation.



Tech 2: Cross-GPU TB Coordination

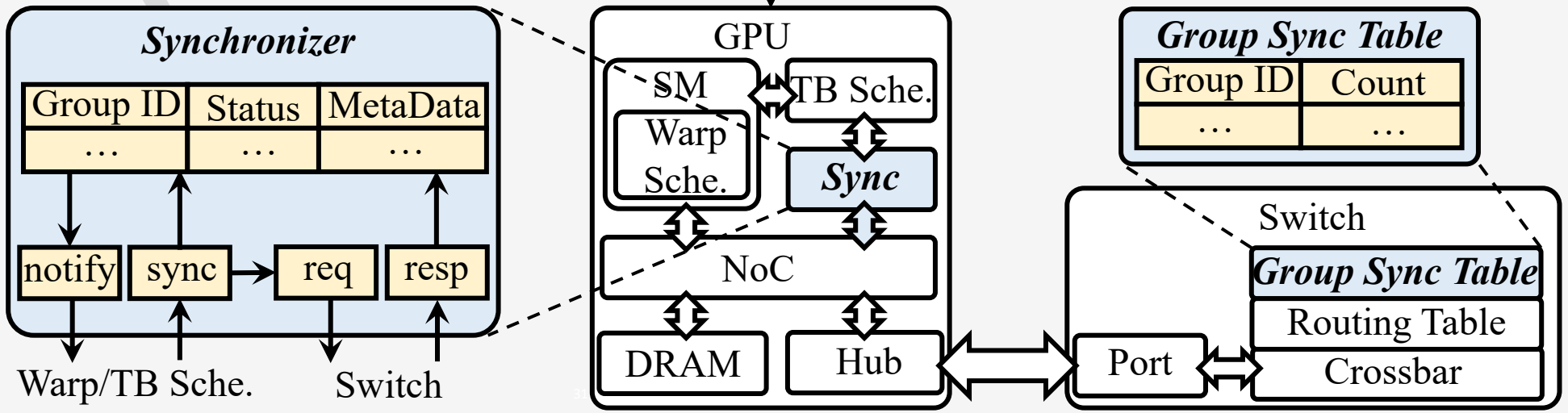
Compiler

Compiler-Guided TB Grouping

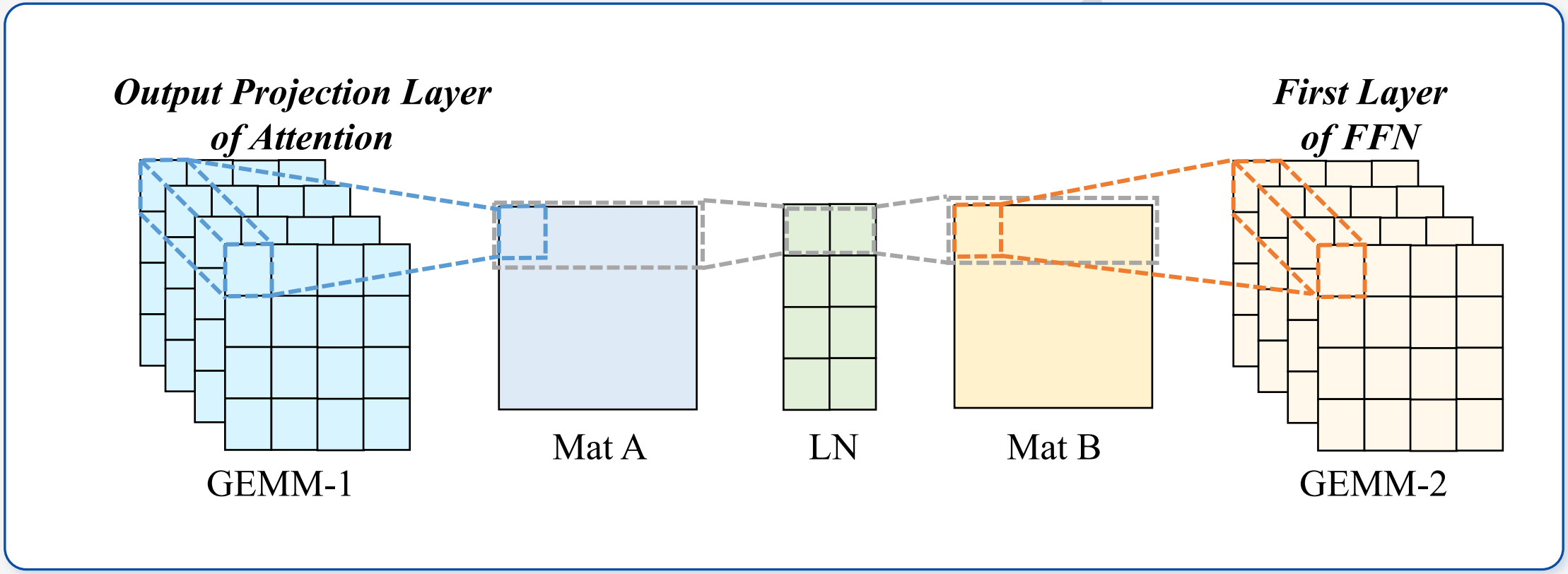


Architecture

Group-Aware Synchronization Mechanisms

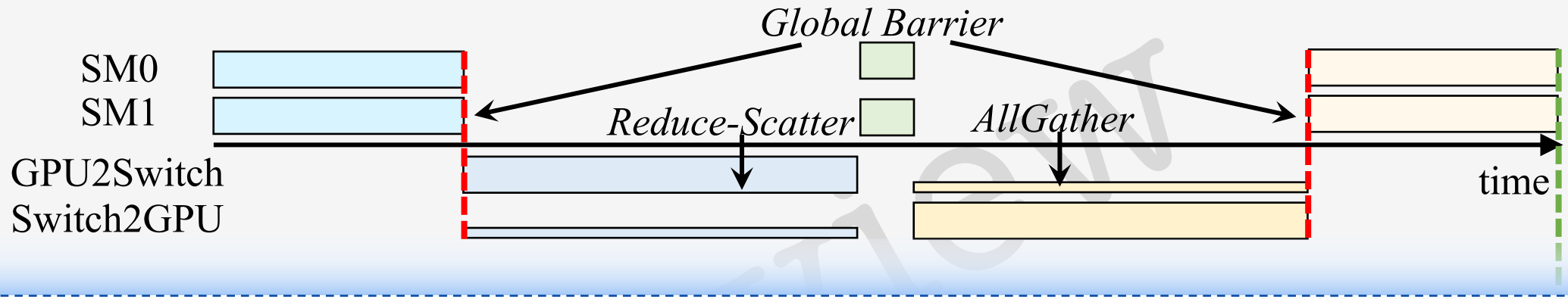


Tech 3: Graph-level dataflow optimization

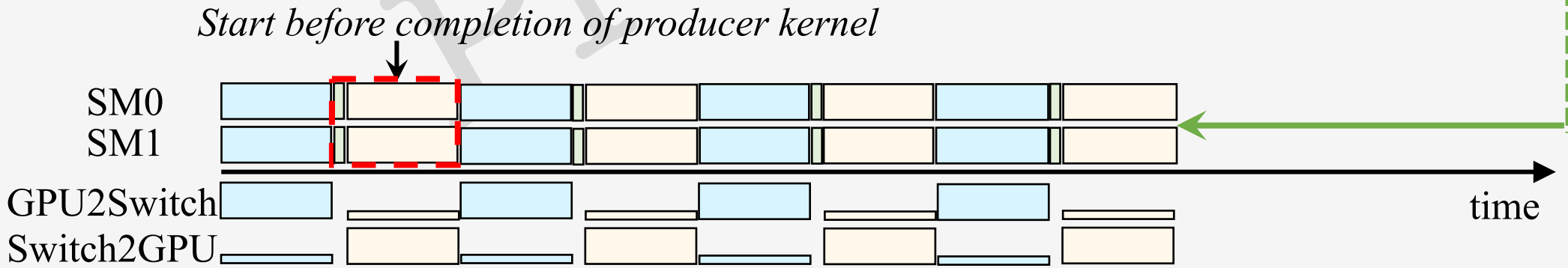


Sub-layer of Transformer layer and its fine-grained dependency

Tech 3: CAIS-enabled Kernel Fusion

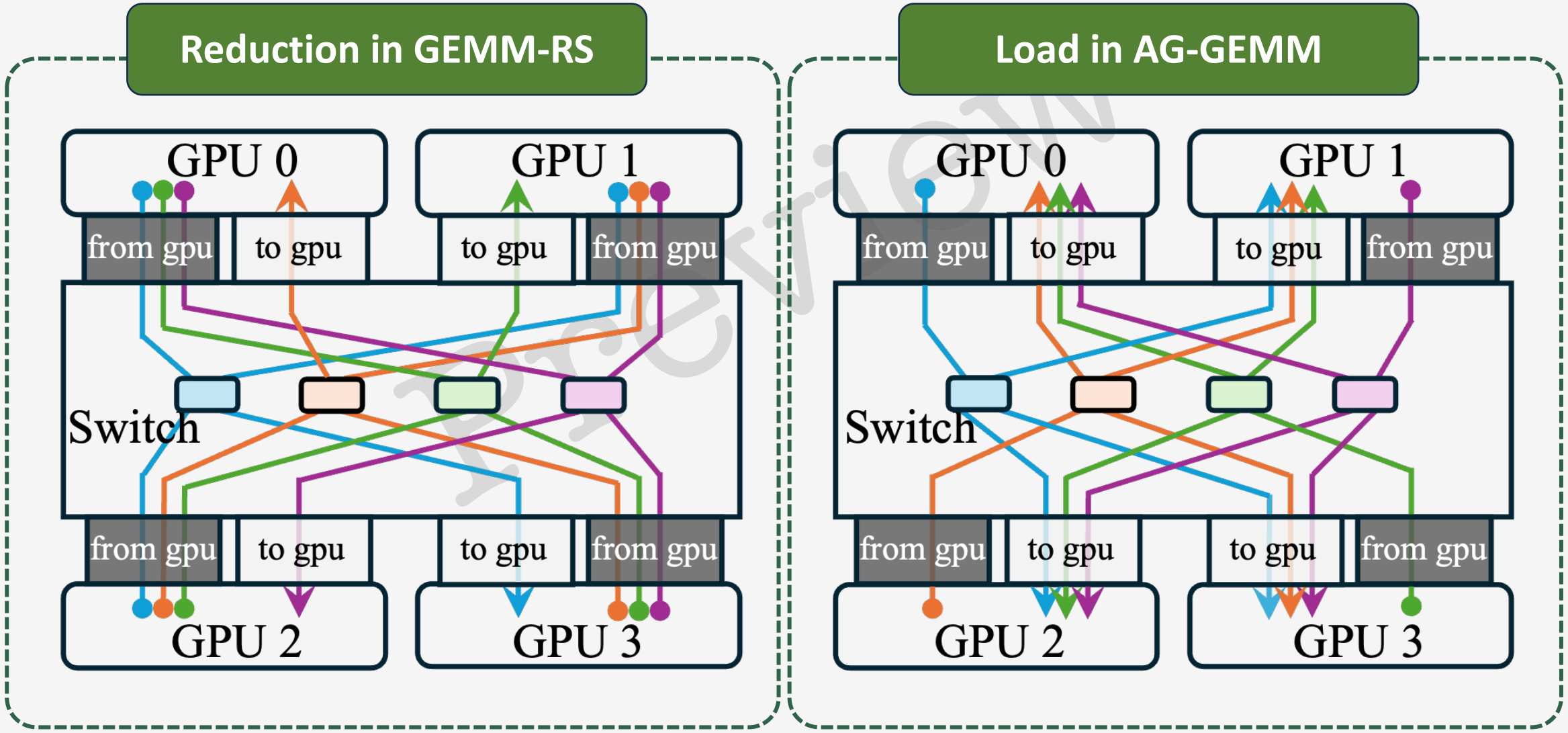


Timeline of communication-centric in-switch computing

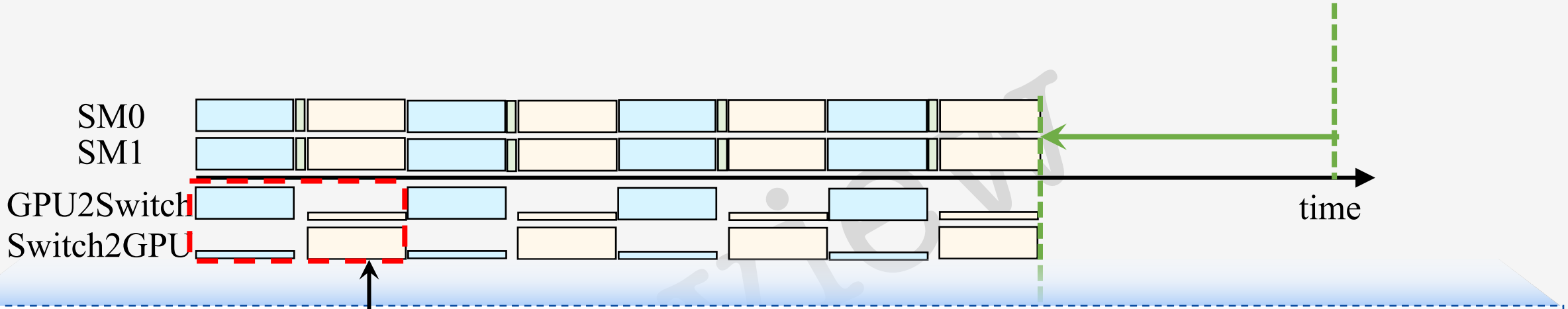


Timeline of compute-aware in-switch computing + fine-grained dependency

Tech 3: Opportunities of Complementary Traffic

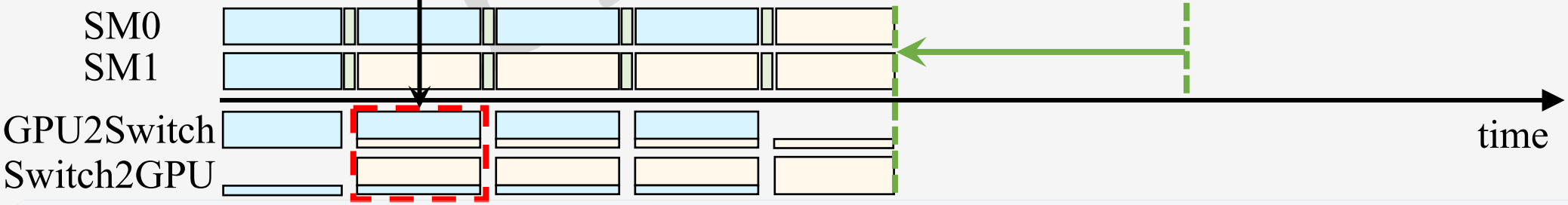


Tech 3: Kernel Fusion for Complementary Traffic



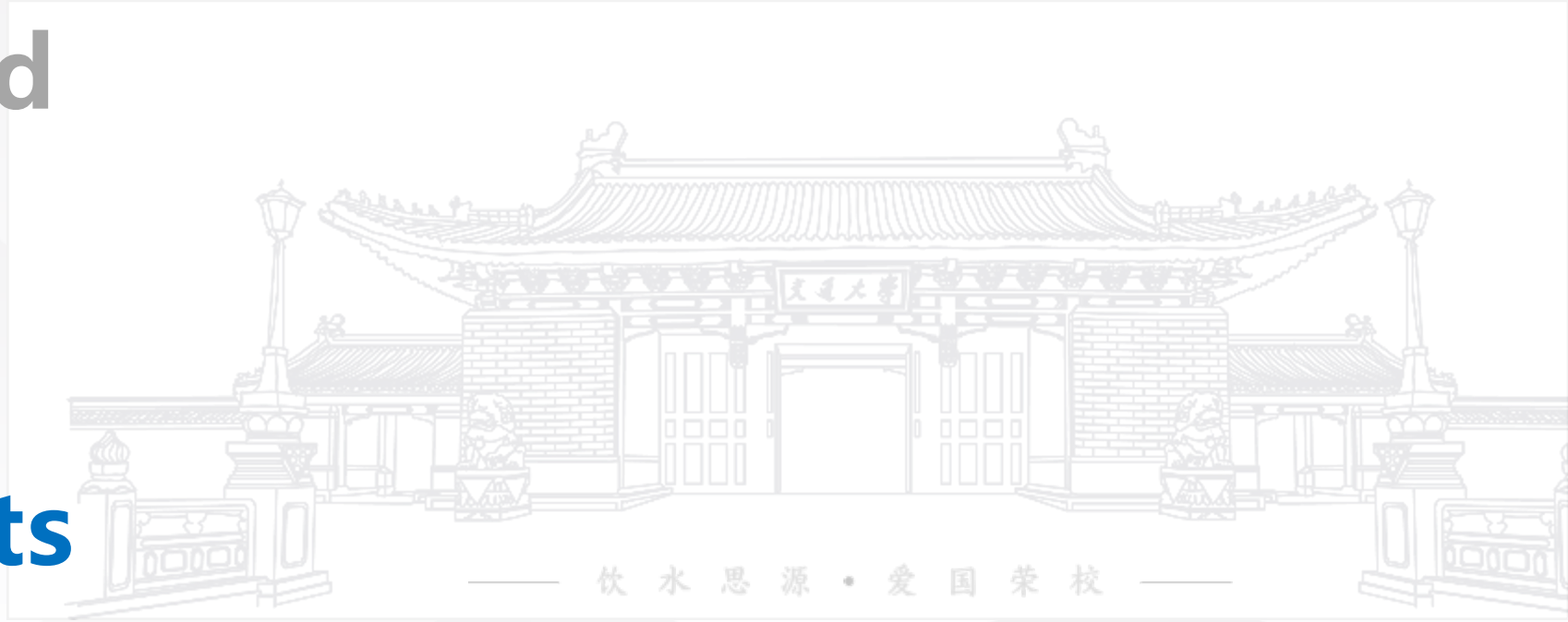
Timeline of compute-aware in-switch computing + fine-grained dependency

Utilize complementary Communication pattern



Timeline of compute-aware in-switch computing + fine-grained dependency + asymmetric kernel overlapping

1. Background
2. Motivation
3. Solutions
4. **Experiments**
5. Conclusion



Experimental Methodology

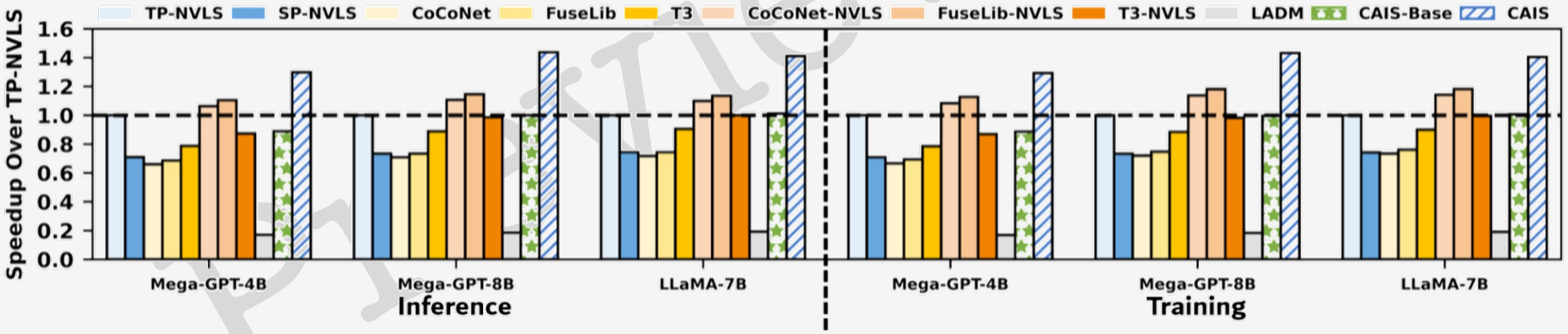
- **Machine Setup:**
  - 8 GPUs + 4 NVSwitches, similar to NVDIA DGX-H100
  - Modified Accel-Sim + BookSim2
- **Workload:**
  - Mega-GPT-4B，Mega-GPT-8B，LLaMA-7B
  - Training and Inference
- **Previous work:**

	Baseline	MICRO'20	ASPLOS'22	SC'24	ASPLOS'24
Original	-	LADM	CoCoNet	FuseLib	T3
+NVLS	TP/SP+NVLS	-	CoCoNet+NVLS	FuseLib+NVLS	T3+NVLS

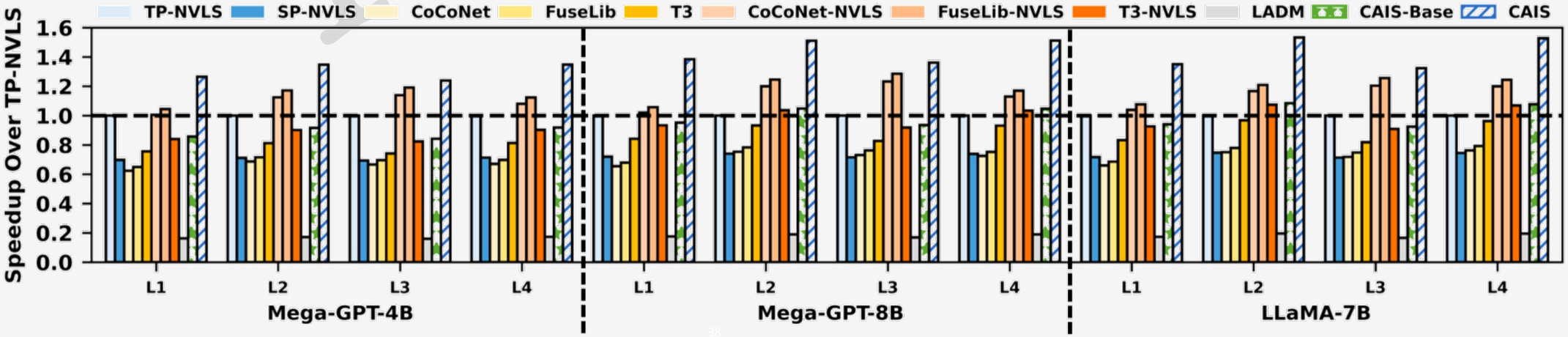
Result 1: Model Speedup (Training/Inference)

- Inference: 1.21×~ 1.99×
- Training: 1.25×~2.03×

End-to-End

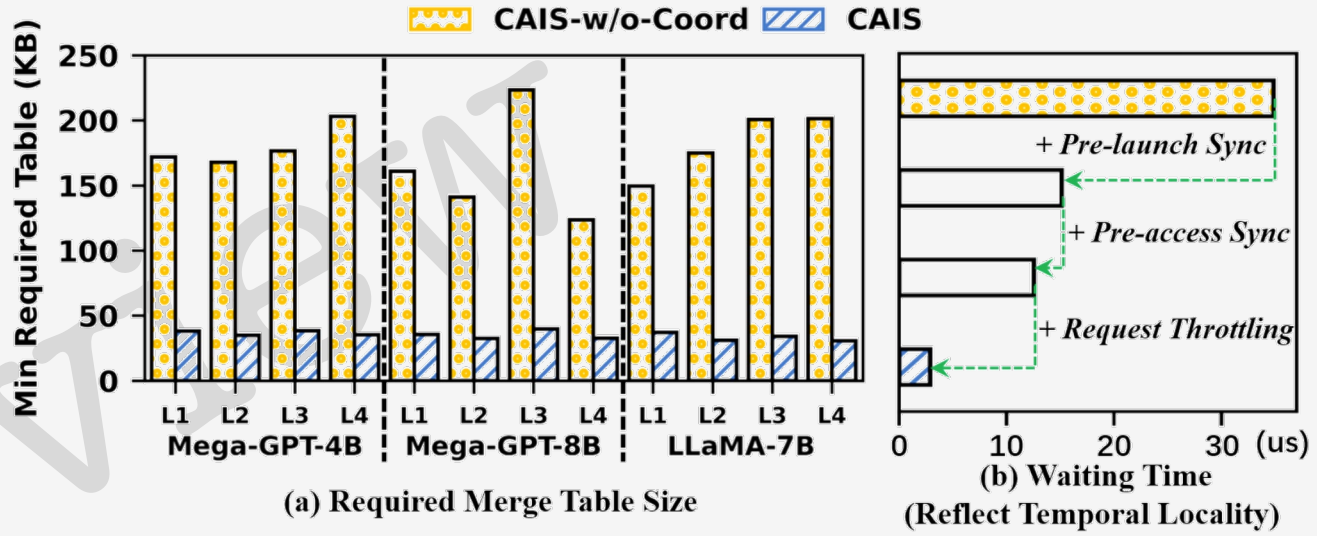


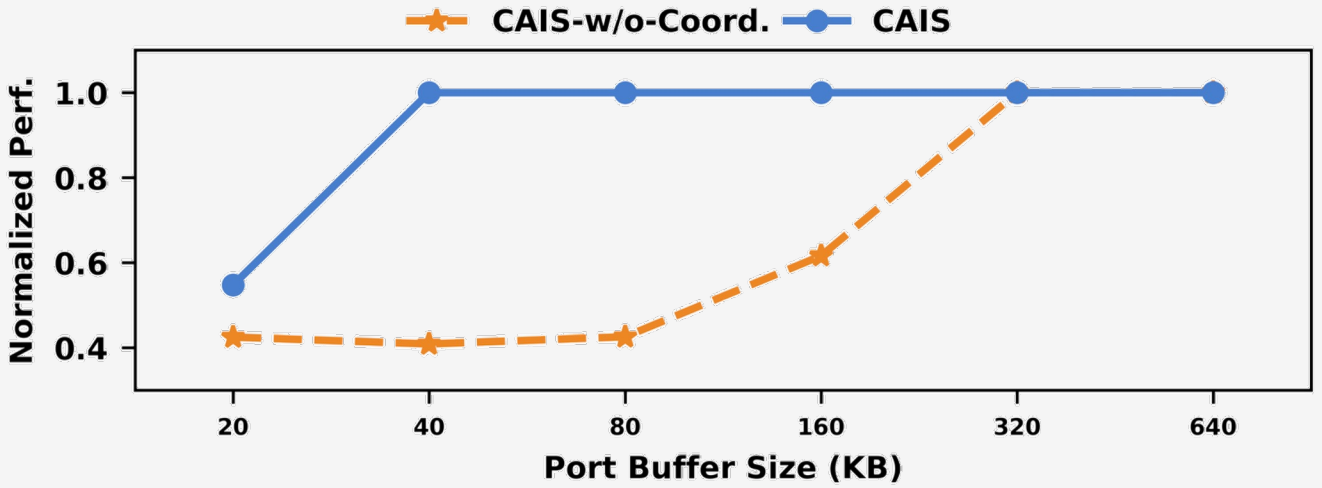
Layer wise



Result 2: Benefits of TB Coordination

- **Higher Performance:**
  - The waiting time was reduced from **35** to **3** microseconds for merge requests.
- **Lower Hardware Cost:**
  - This further decreases the required capacity of the lookup table, leading to higher performance with fewer in-switch caches.



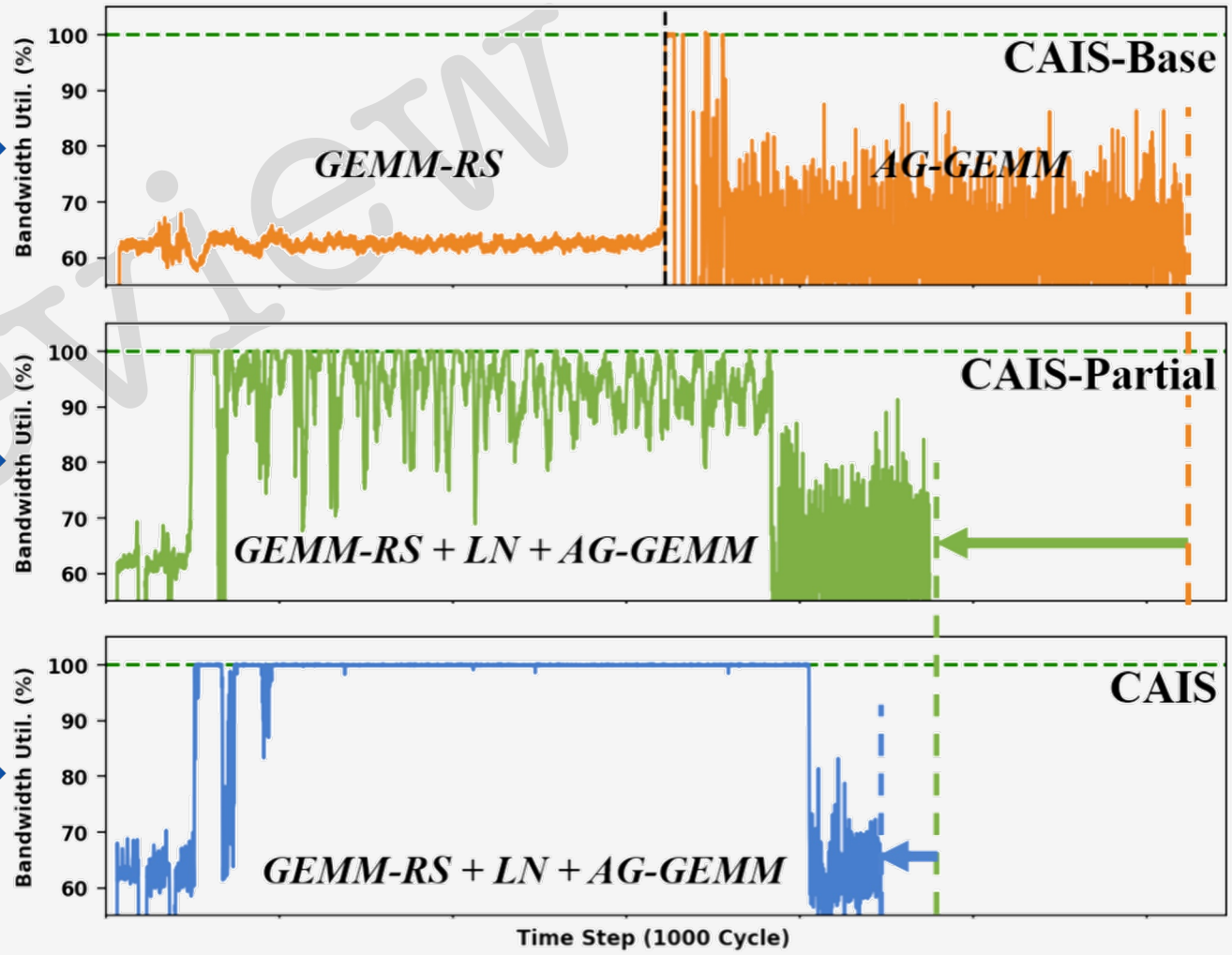


Result 3: Bandwidth Utilization Improvements

CAIS base: 62.4%
Only ISA & Arch

CAIS Partial: 84.7%
ISA & Arch + Graph

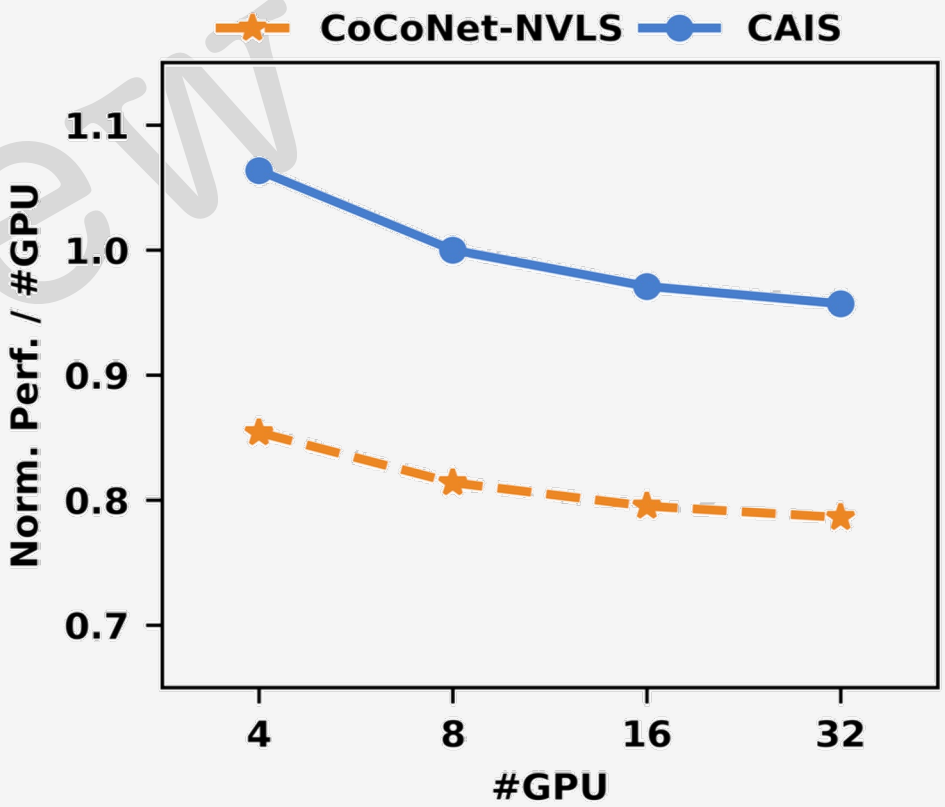
CAIS Full: 90.2%
ISA & Arch + Graph + Trfc Ctl



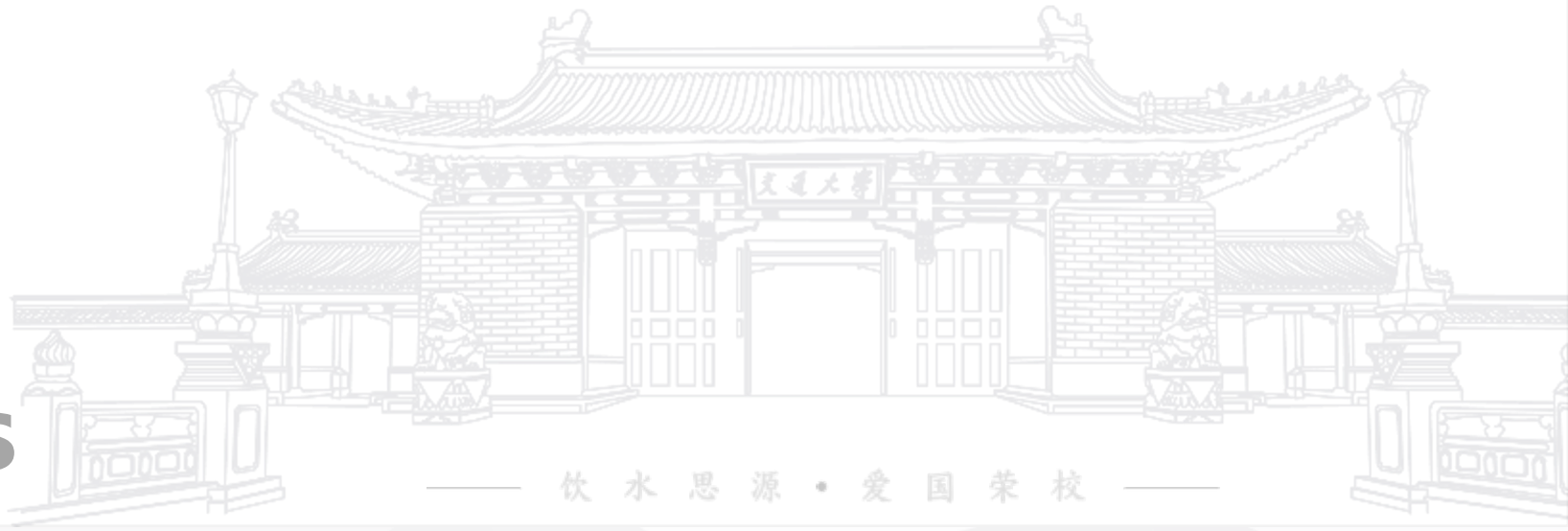
Result 4: Scalability and HW Cost

- **Scalability:**
  - The performance with 32 GPUs is less than 5% lower than with 8 GPUs.
  - The hardware overhead does not increase with the number of GPUs

- **HW Cost:**
  - Switch: 0.5 mm2, < 1% of NVswitch.
  - GPU: 0.019 mm2, <0.01% of H100.
  - 12 nm process



1. Background
2. Motivation
3. Solutions
4. Experiments
5. Conclusion



Takeaways & Open questions

- The existing NVSwitch is designed with a **communication-centric** approach, **lacking consideration for up/down-stream operators** (e.g., GEMM), limiting the performance of in-switch computing in Scale-up GPU System.
- This work provides a detailed analysis of the TP dataflow during both LLM training and inference, and subsequently redesigns a **Compute-aware In-switch Computing** communication mechanism.
- Building upon this mechanism, we propose corresponding designs at the **instruction set**, **architecture**, and **software** stacks. These designs substantially enhance the collective performance of multi-GPU systems, thereby demonstrating the advantages of the proposed scheme.

上海交通大学
SHANGHAI JIAO TONG UNIVERSITY

Thank You !

Chen Zhang

Assistant Professor

Shanghai Jiao Tong University

chenzhang.sjtu@sjtu.edu.cn