NeurIPS 2025

FlashMoE: Fast Distributed MoE in a Single Kernel

Osayamen Jonathan Aimuyo*, Byungsoo Oh, Rachee Singh

November 6, 2025

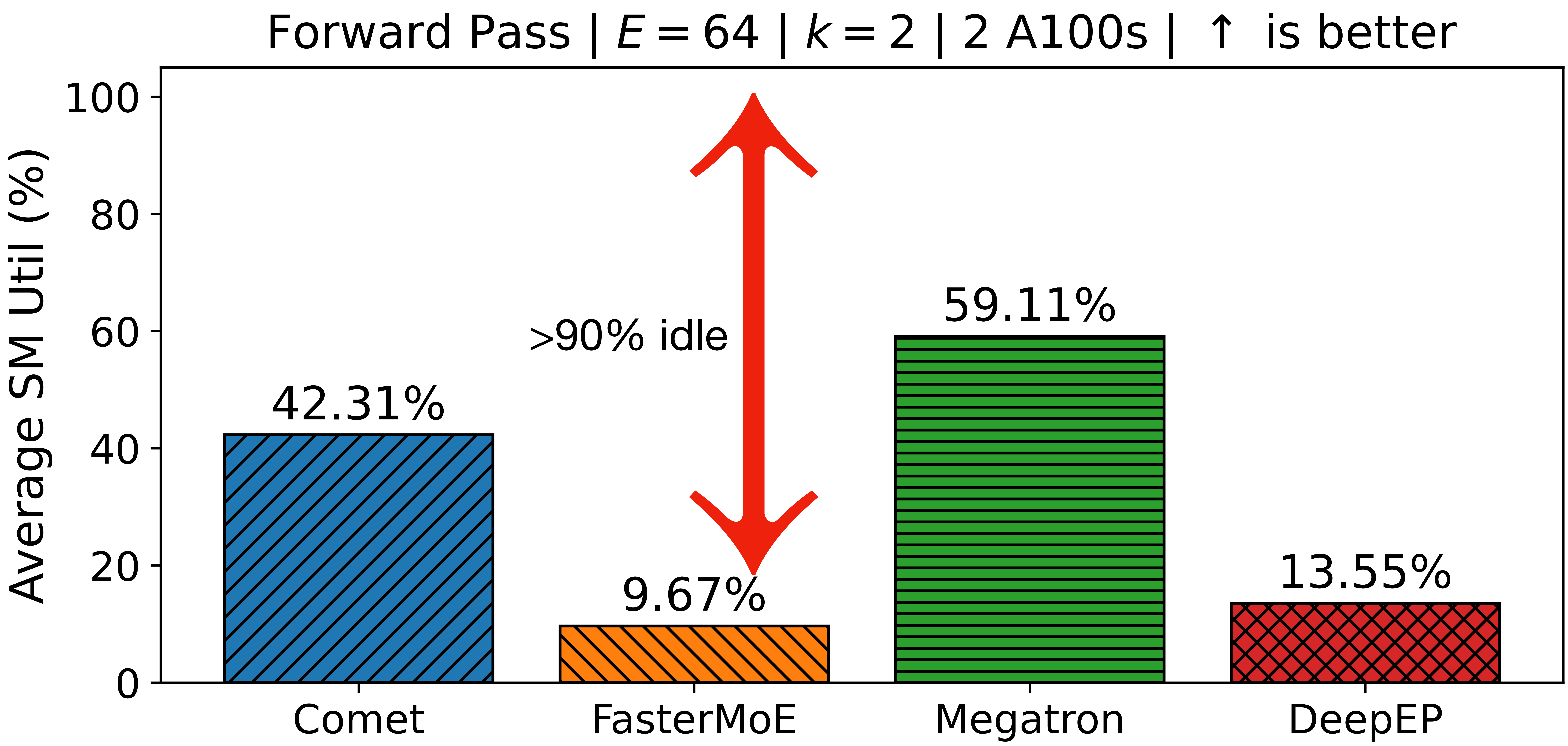
Cornell University Founded A.D. 1865
Cornell Bowers CIS
Computer Science

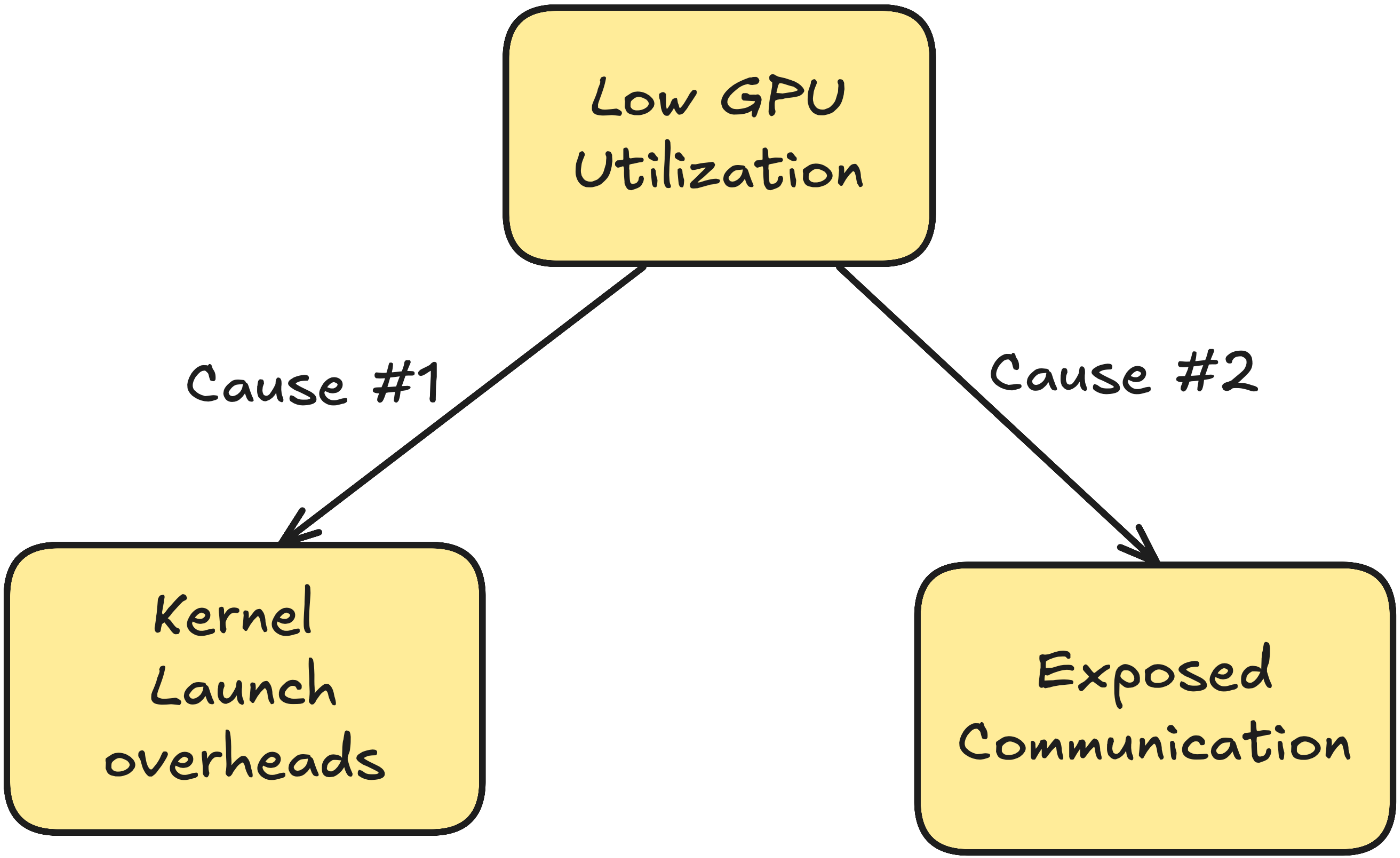
*now at Stanford University

Existing Distributed MoE implementations leave significant performance on the table!

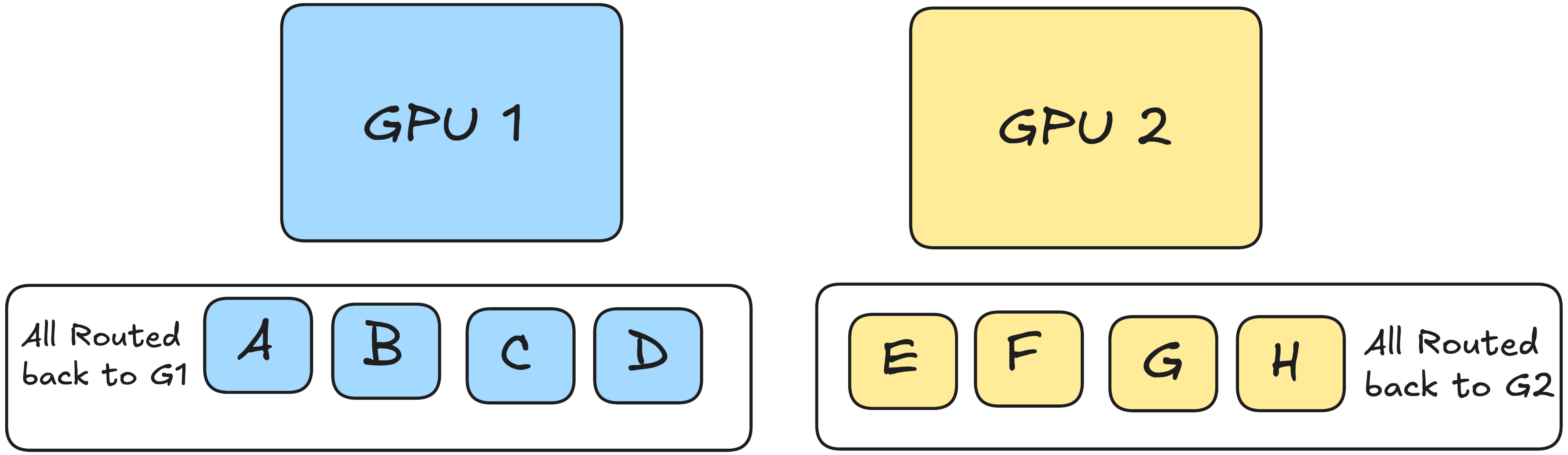
Claim

Challenge 1: GPUs are idle for up to 90% on average!

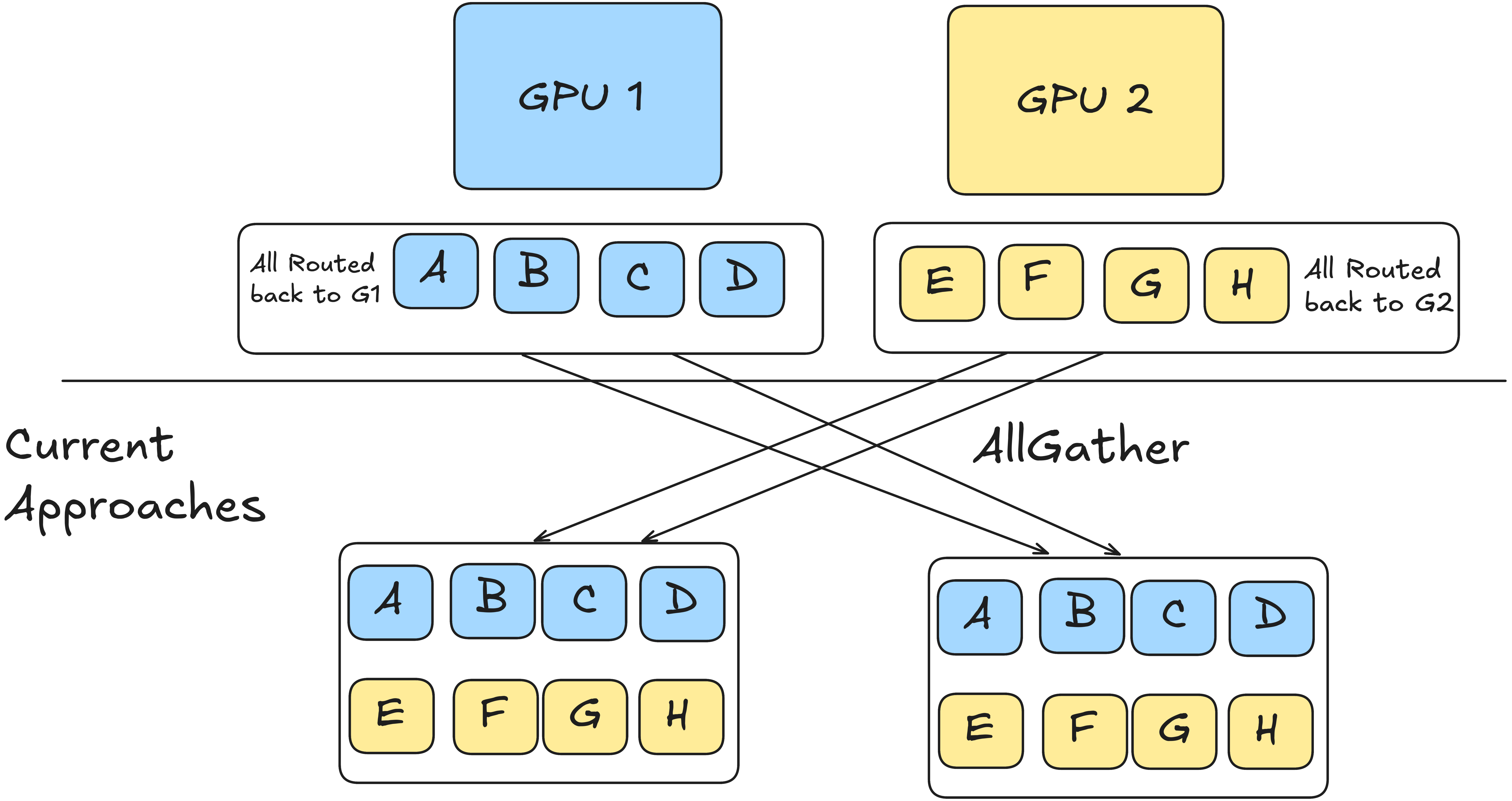



Low GPU Utilization
Cause #1
Cause #2
Kernel Launch overheads
Exposed Communication

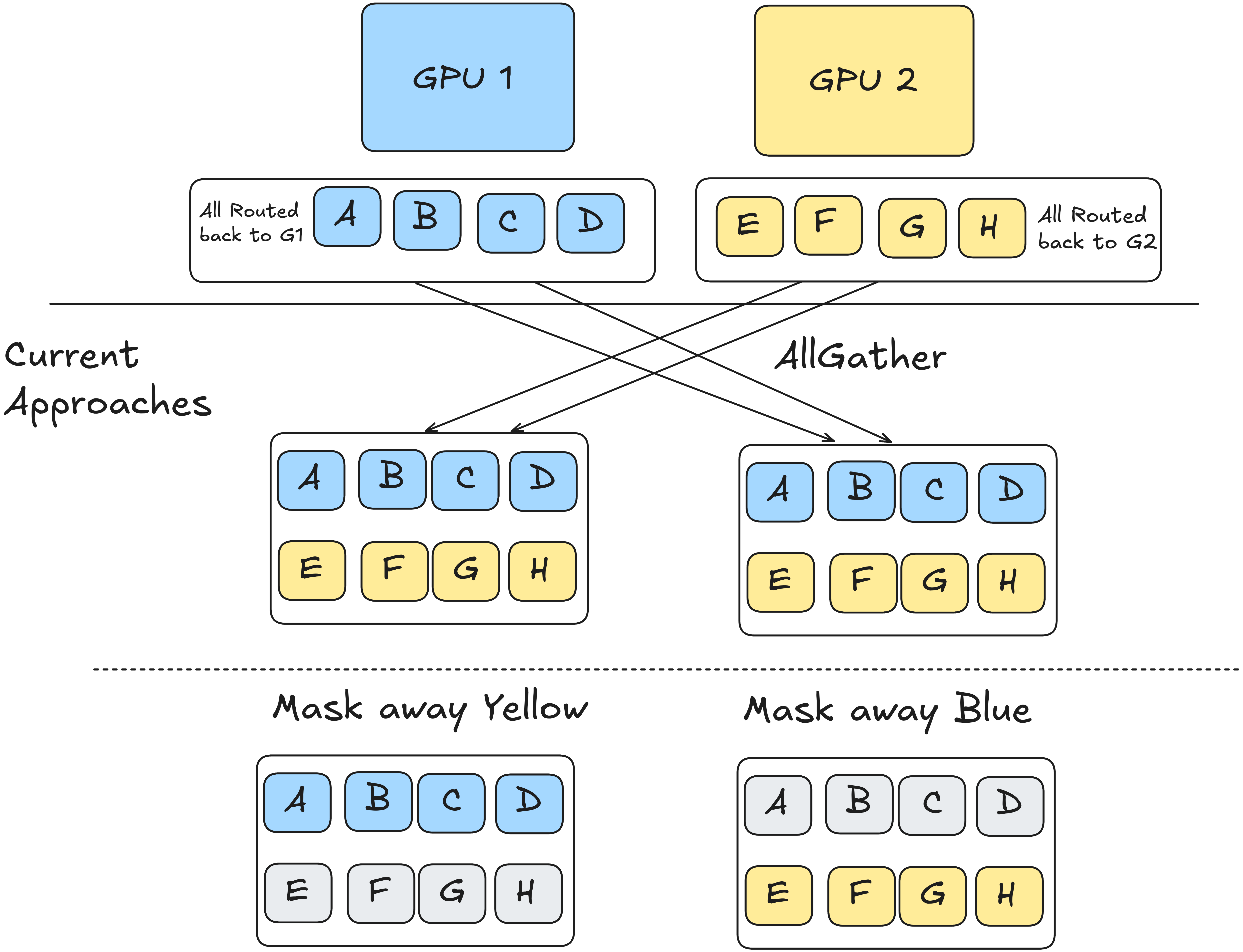
Challenges 2&3: Inefficient Communication And Lack of Task Locality



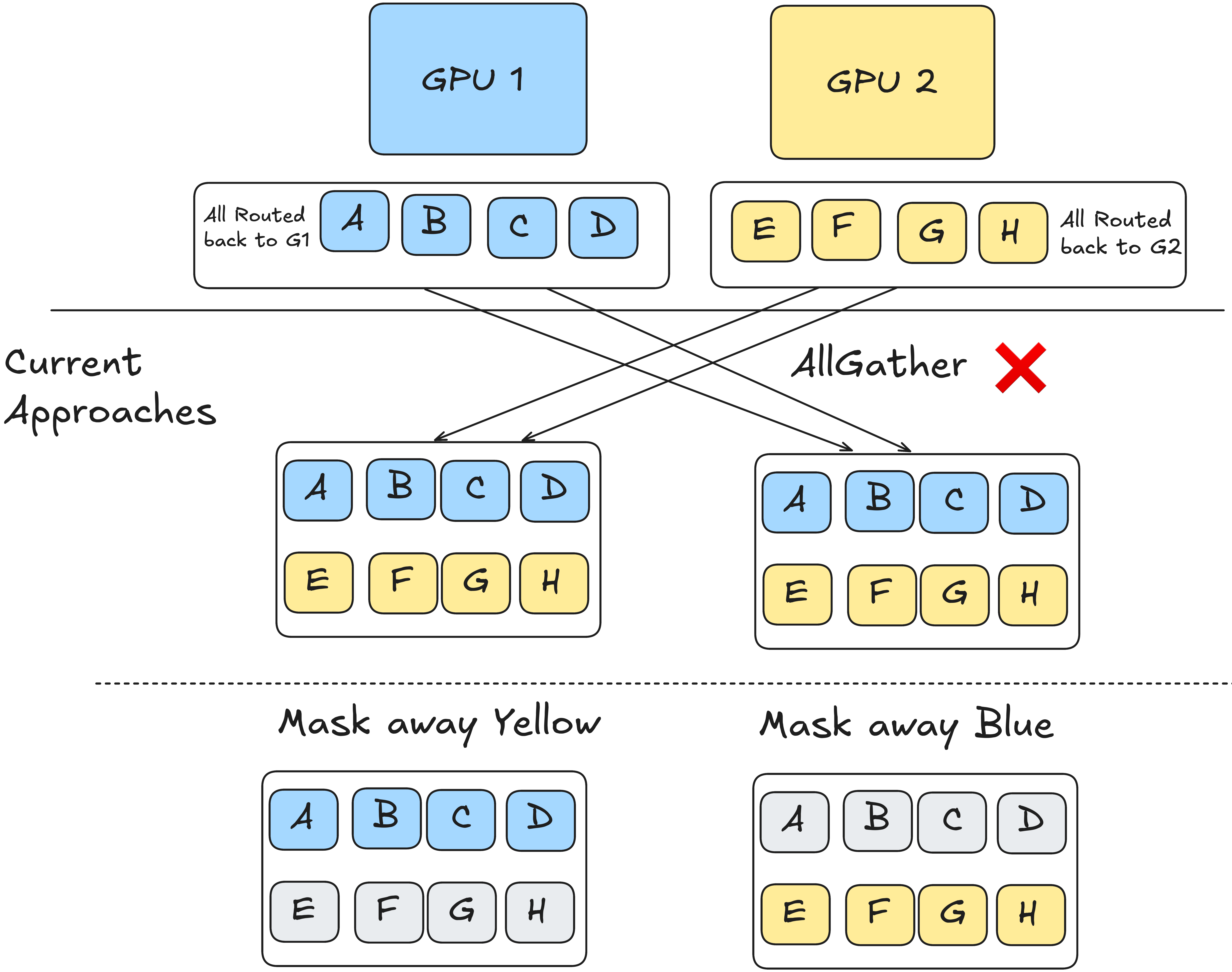
Challenges 2&3: Inefficient Communication And Lack of Task Locality



Challenges 2&3: Inefficient Communication And Lack of Task Locality



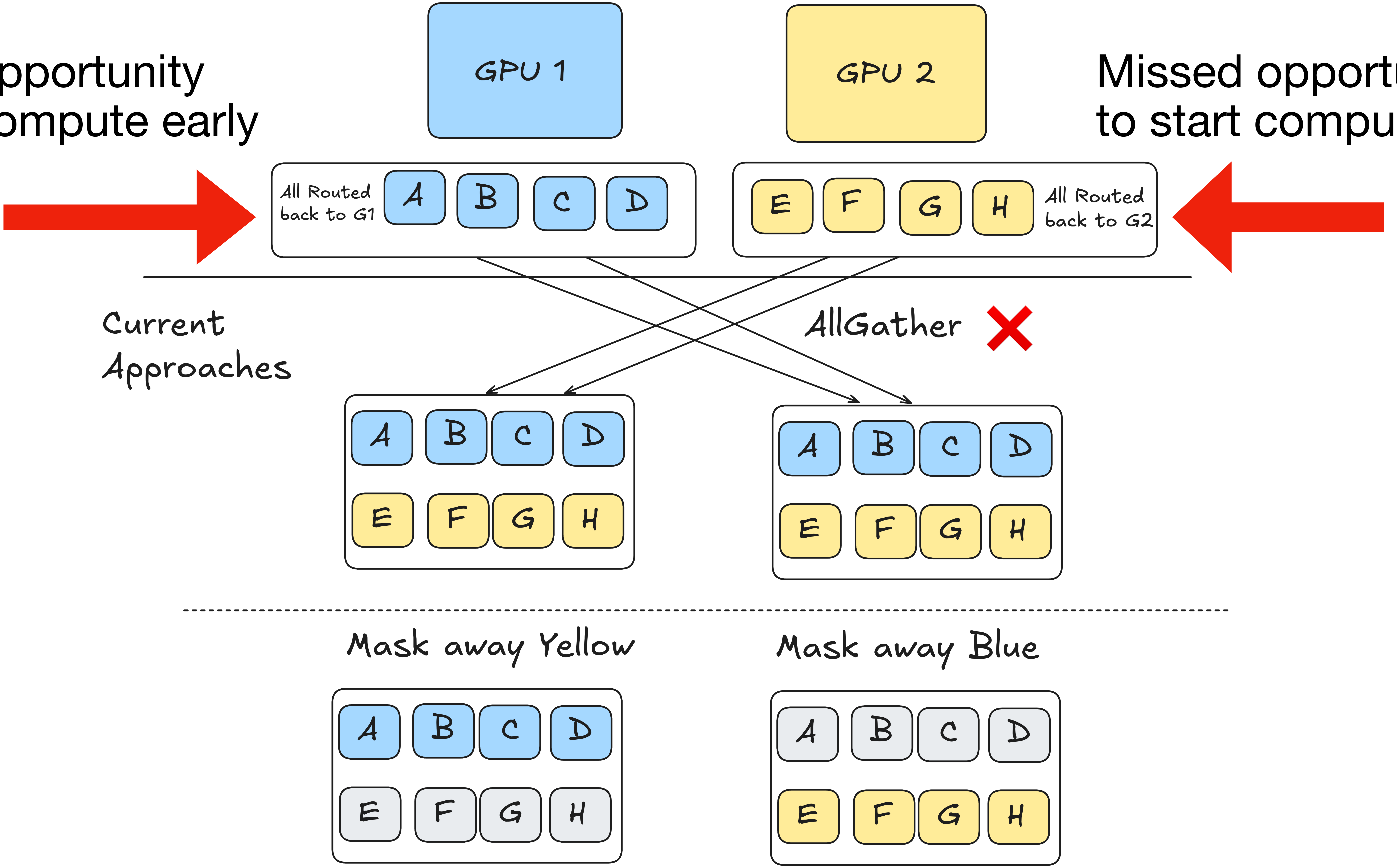
Challenges 2&3: Inefficient Communication And Lack of Task Locality



Challenges 2&3: Inefficient Communication And Lack of Task Locality

Missed opportunity to start compute early

Missed opportunity to start compute early

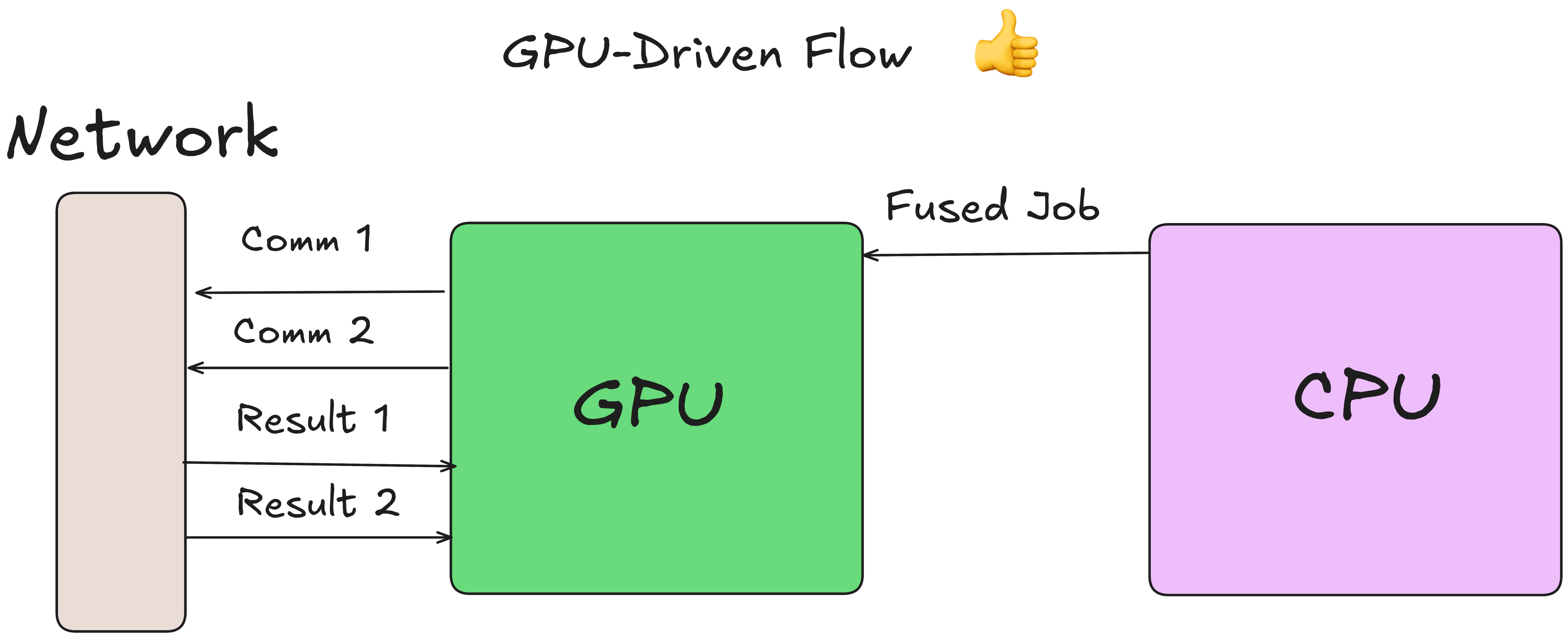


Non-fused CPU-Driven Flow 😢

Kernel Fusion To Tackle DMoE inefficiencies

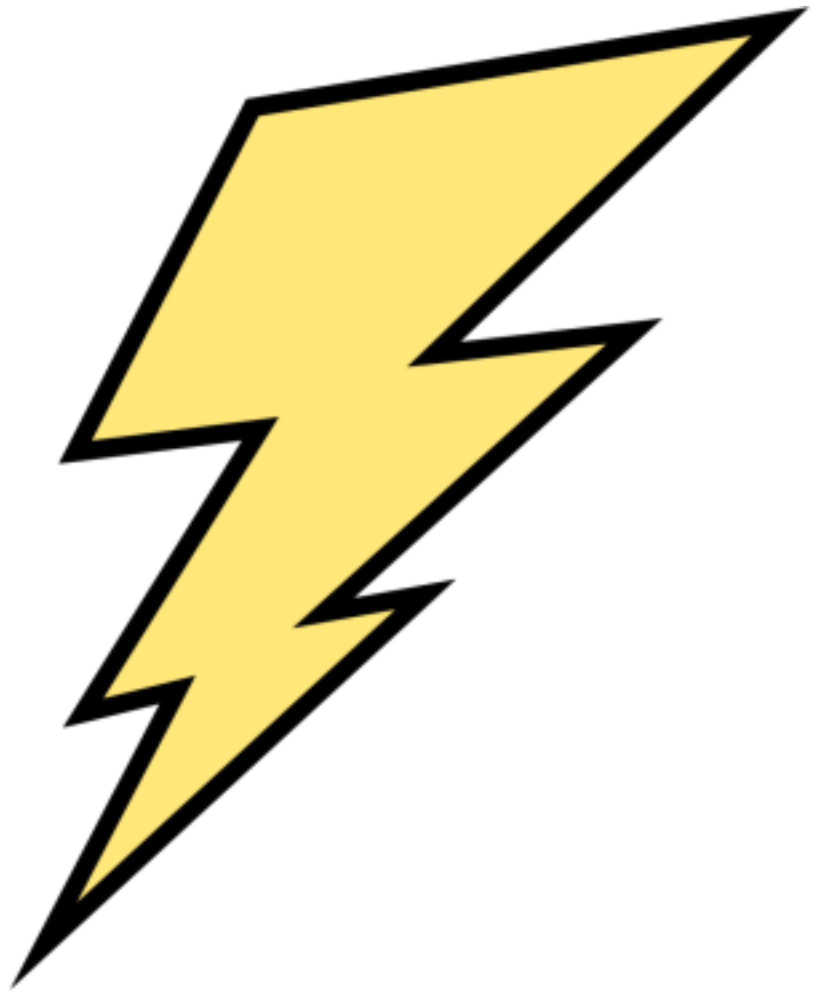
Kernel fusion:

- **eliminates** kernel launch overheads.
- **unlocks** fine-grained overlap of communication.
- **exploits** data locality: eliding unnecessary HBM roundtrips.
- **enables** low-latency, high-bandwidth GPU-initiated communication.
- **expands** the design space for communication and compute optimizations.
- **offloads** task dependency management to the GPU, making implementations very difficult



Key Challenge: How do we implement lightweight task management for a *completely fused* DMoE kernel?

FlashMoE



The Novelty of FlashMoE

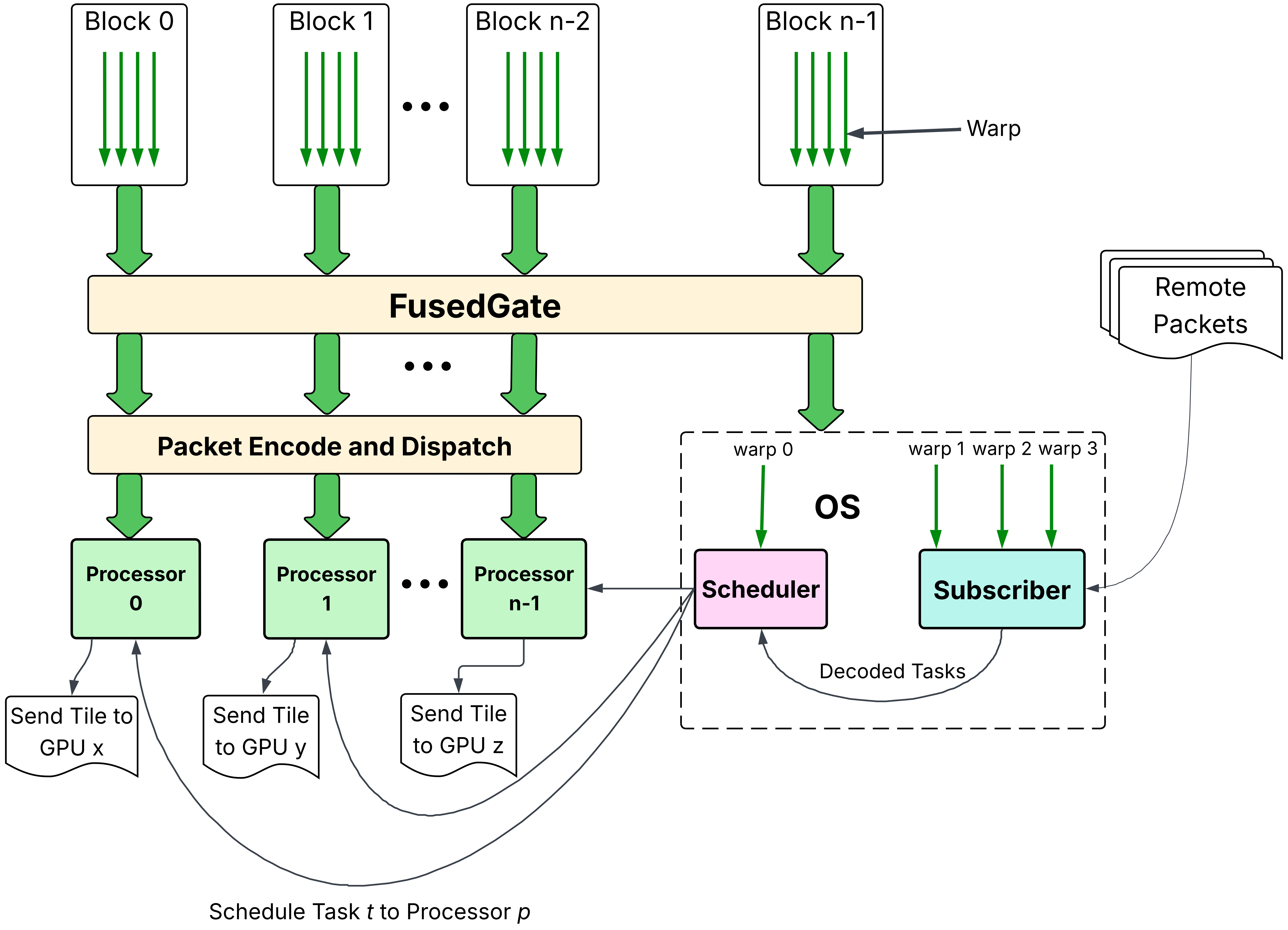
All at blazing speed!

☞ First to fuse all DMoE communication and computation into a *single kernel*

☞ First *in*-kernel, actor-style OS with work-conserving scheduling

☞ Formalize task abstraction for tile-level parallelism

☞ Introduces a provably correct, non-blocking layout for inter-GPU PGAS



Algorithm 1: *Flash Distributed MoE Fused Kernel*

Input: $A, O \in \mathbb{R}^{S \times H},\ E \in \mathbb{R}^{L \times H \times P},\ N$

begin
    $T, G_\phi \leftarrow$ **FusedGate**(A)
    if *blockId* $+ 1 < N$ **then**
        Dispatch(T, A)
        processor::start()
    else
        if $warpID == 0$ **then**
            scheduler::start()
        else
            subscriber::start(E, O)
        end if
    end if
end

Task Abstraction

$$t = (\mathscr{M}, \star, \phi)$$

Task descriptor

Metadata

Binary operator

Activation

Symmetric Tensor Layout

Non-blocking indexing

P_0 View

	R_0		R_1	
	B_0	B_1	B_0	B_1
P_0	E_0	E_0	E_0	E_0
	E_1	E_1	E_1	E_1
P_1	E_2	E_0	E_0	E_2
	E_3	E_1	E_1	E_3

P_1 View

	R_0		R_1	
	B_0	B_1	B_0	B_1
P_0	E_0	E_2	E_2	E_0
	E_1	E_3	E_3	E_1
P_1	E_2	E_2	E_2	E_2
	E_3	E_3	E_3	E_3

Theorem 1.1. *L is write-write conflict-free.*

Evaluation

Experimental Setup

- ☞ 4 Baselines: COMET, Megatron-[CUTLASS, TE], FasterMoE
- ☞ Flash: FP32, baselines: FP16.
- ☞ Testbed: 8 NVLink H100 RunPod VM.
- ☞ All results: averaged across 32 runs and preceded by 32 warmups
- ☞ Only forward pass

What we Evaluate

- ☞ GPU Utilization
- ☞ E2E Latency
- ☞ Experts Scalability
- ☞ Communication Efficiency

Table 1: Implementation metrics of *FlashMoE* using inlined NVSHMEM 3.2.5 on SM 80

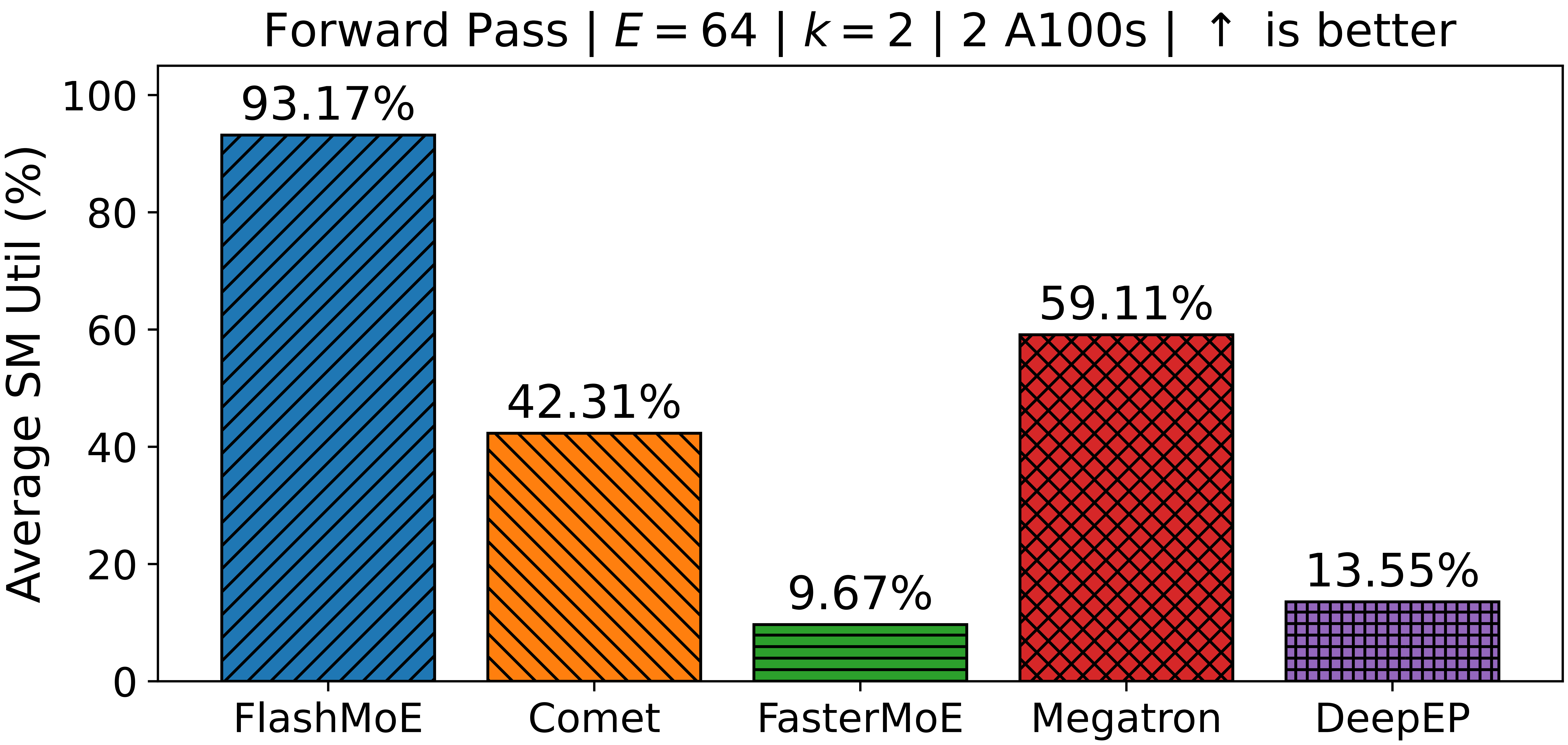
Metric	**Value**
Total lines of code (CUDA/C++)	6820
Kernel stack frame size	0 B
Spill stores (per thread)	0
Spill loads (per thread)	0
Shared memory usage (per block)	46 KB
Registers per thread	255
Max active blocks per SM	2
Compilation time	53 seconds
Binary size	29 MB

FlashMoE eliminates DMoE launch overheads!

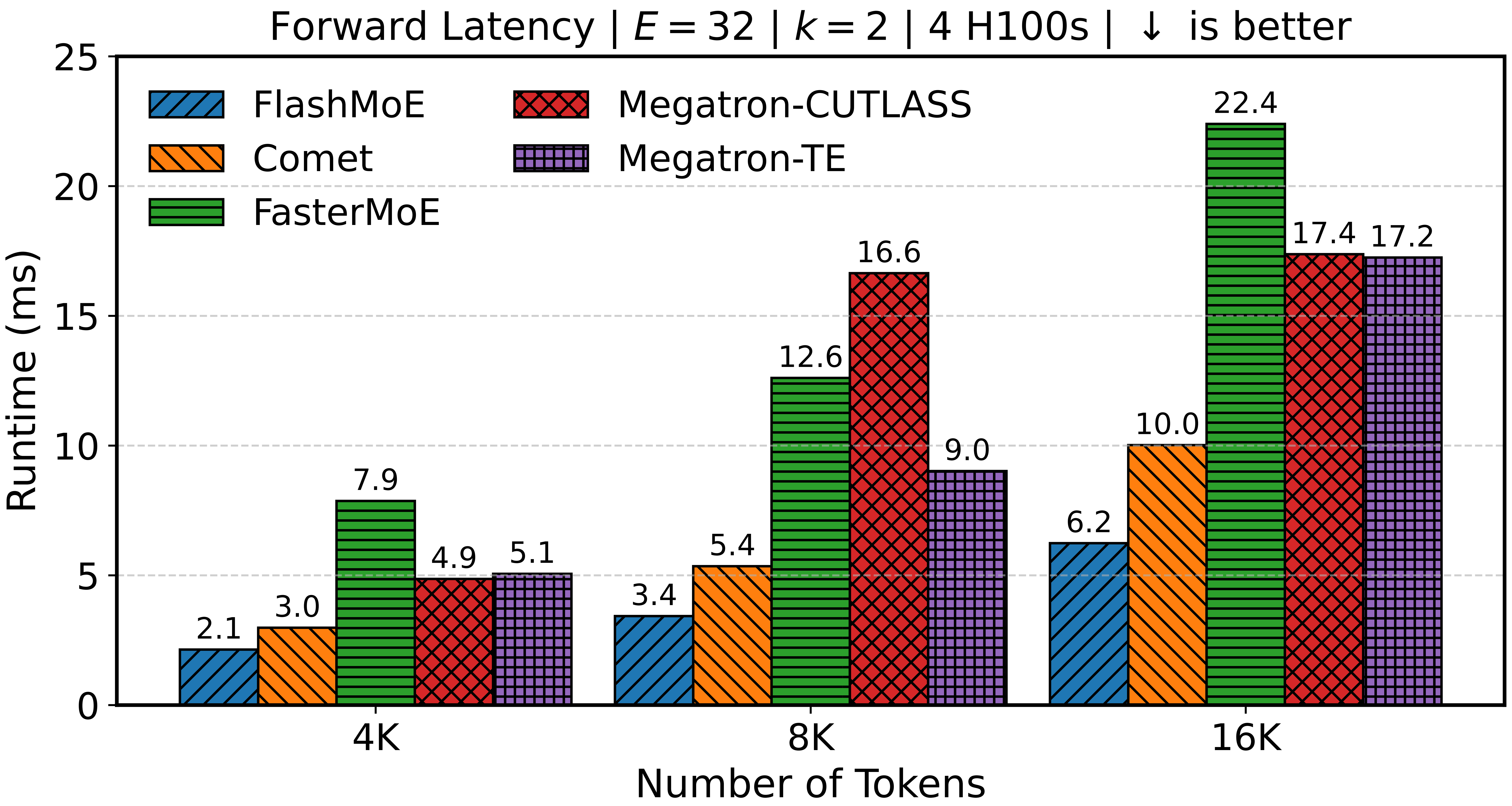
Works	**Launched GPU Ops**
FlashMoE (ours)	1
COMET	33
Megatron-LM CUTLASS	85
Megatron-LM TE	261
Megatron-LM + DeepEP	432
DeepSpeedMoE	550

Table 2: **Kernel Fusion Comparison.** We report GPU operations of from detailed profiling with Nsight Systems. Operations were from an MoE forward pass across 2 GPUs with 64 total experts.

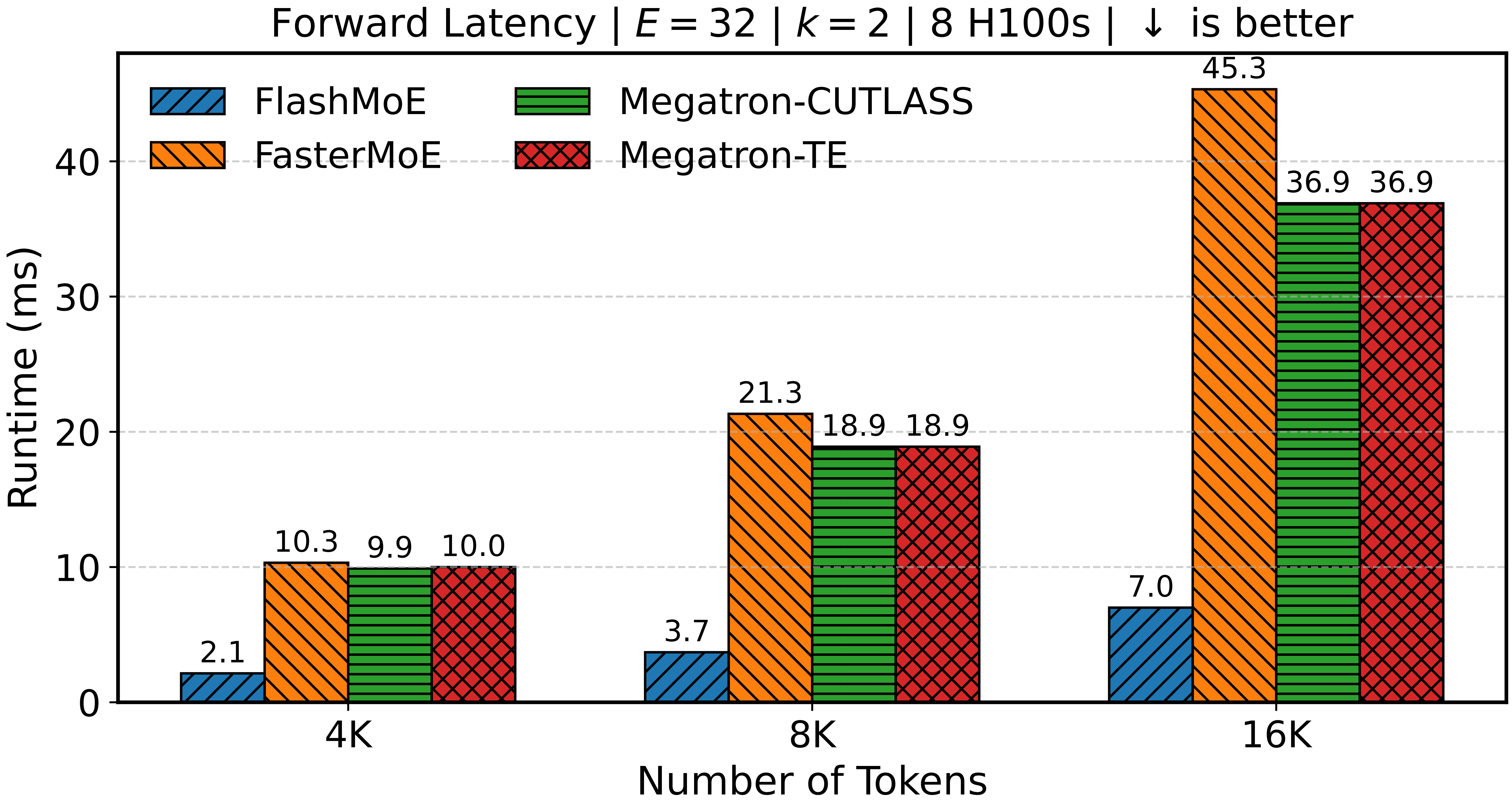
FlashMoE achieves 9x higher GPU Utilization!



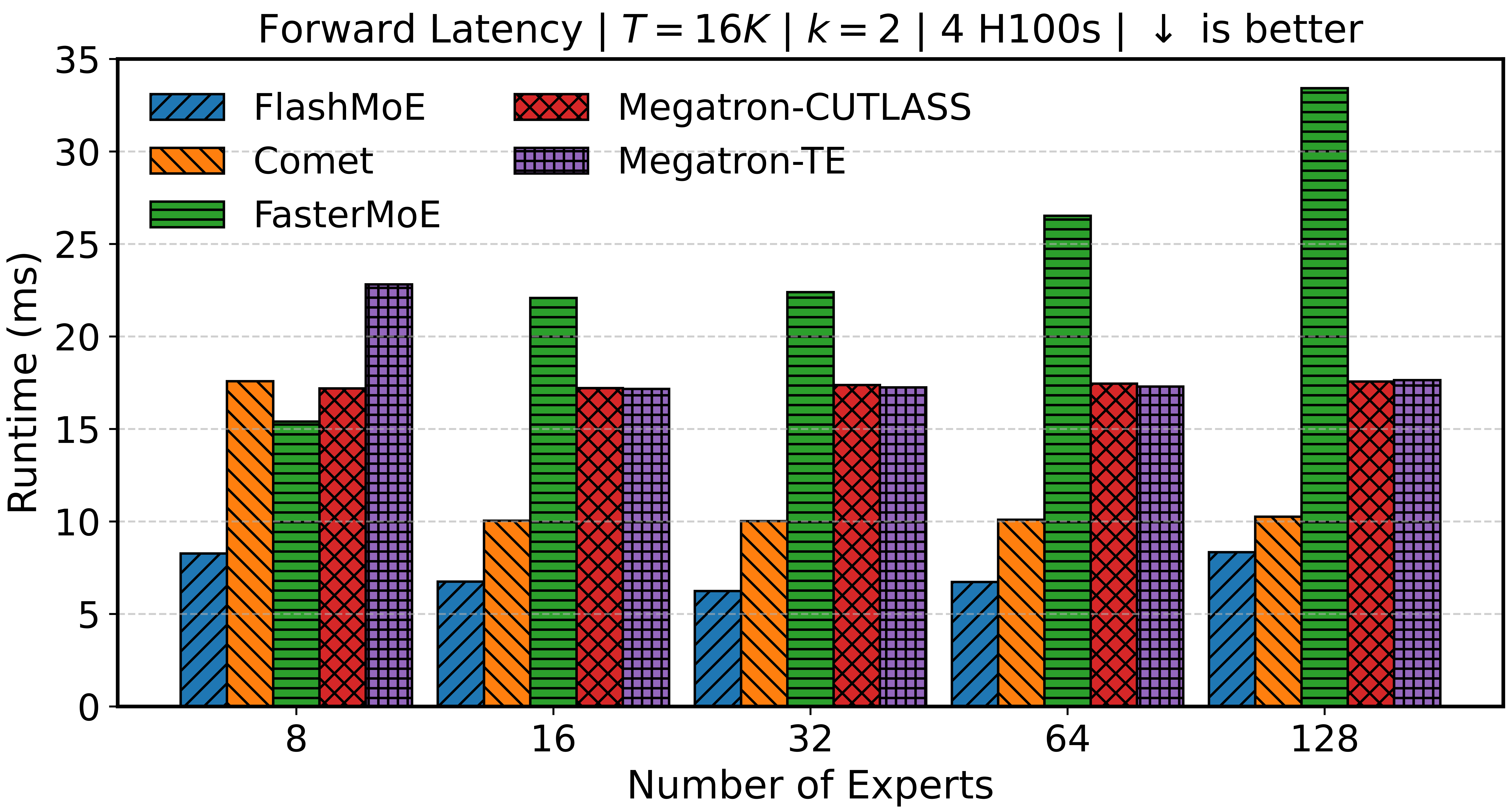
FlashMoE is 4.8x faster on 4 GPUs



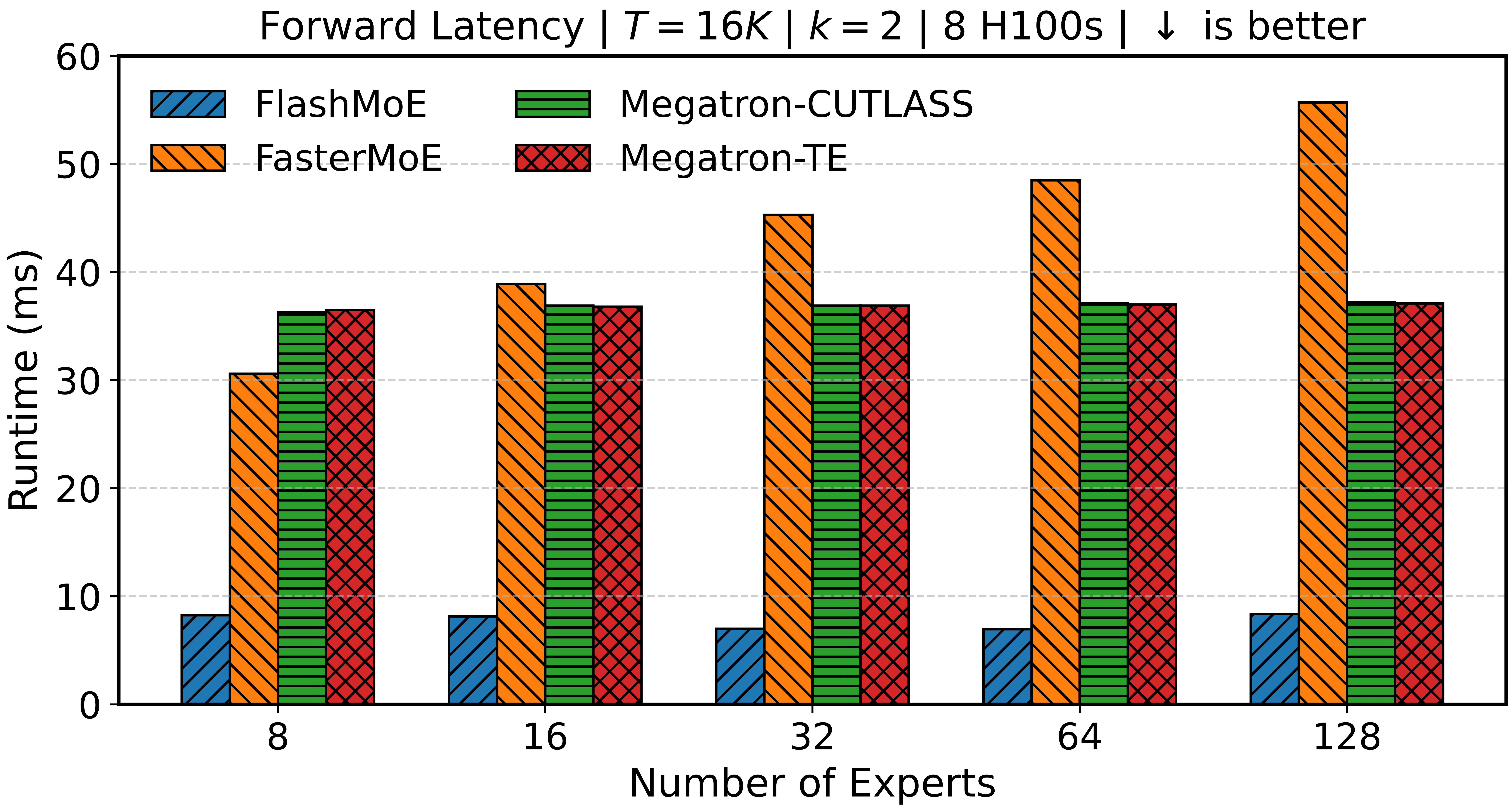
FlashMoE is 6x faster on 8 GPUs!



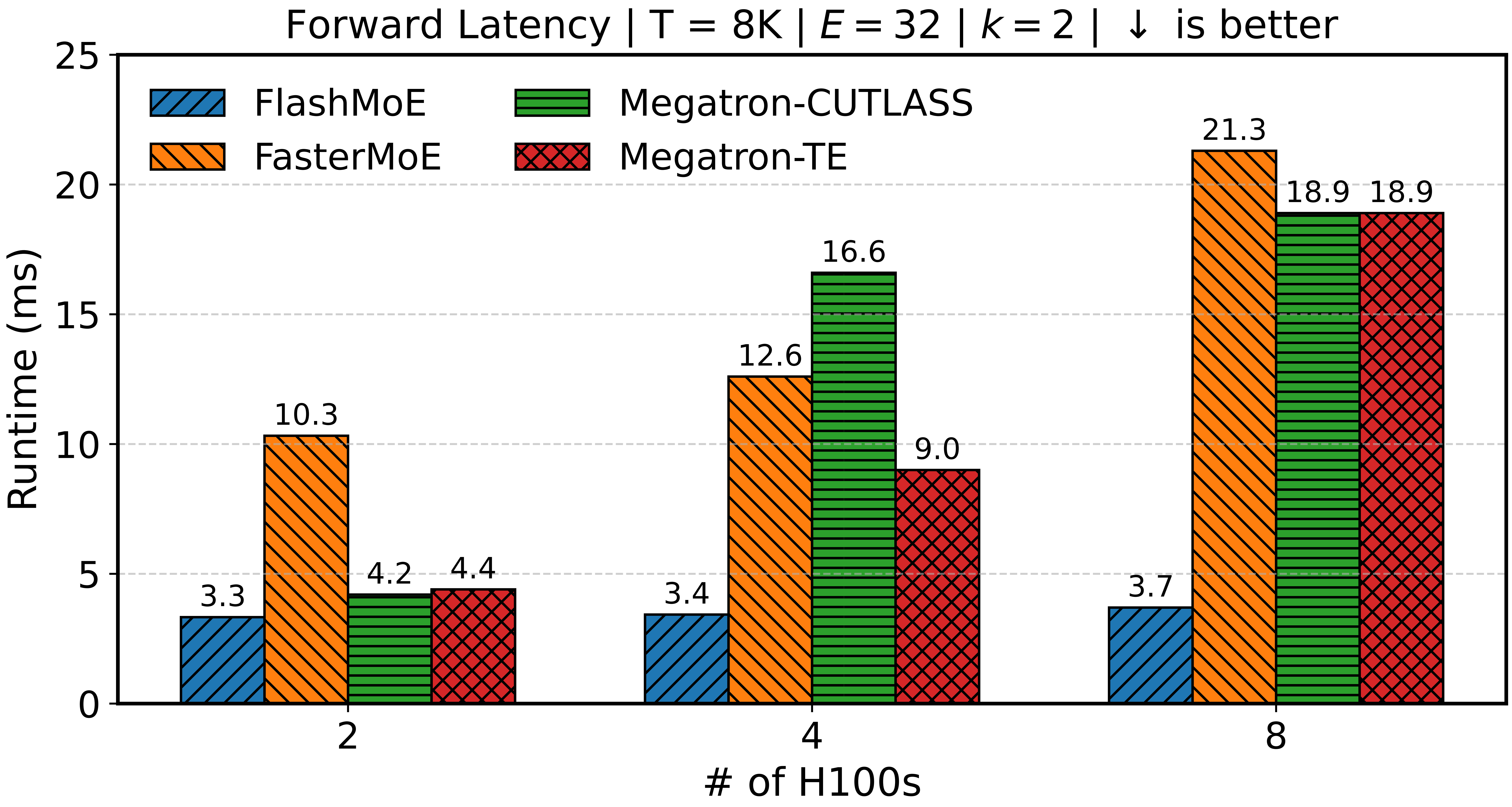
FlashMoE has uniform latency as experts increase!



FlashMoE has uniform latency as experts increase!



FlashMoE gives > 89% Communication Efficiency, 4x higher than baselines!

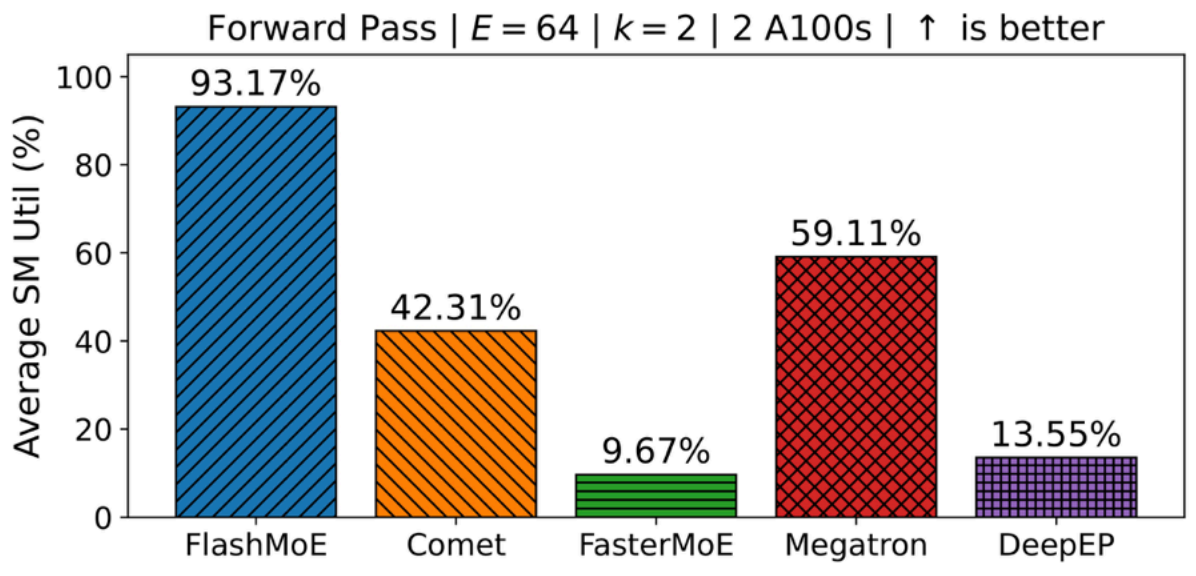


Conclusion

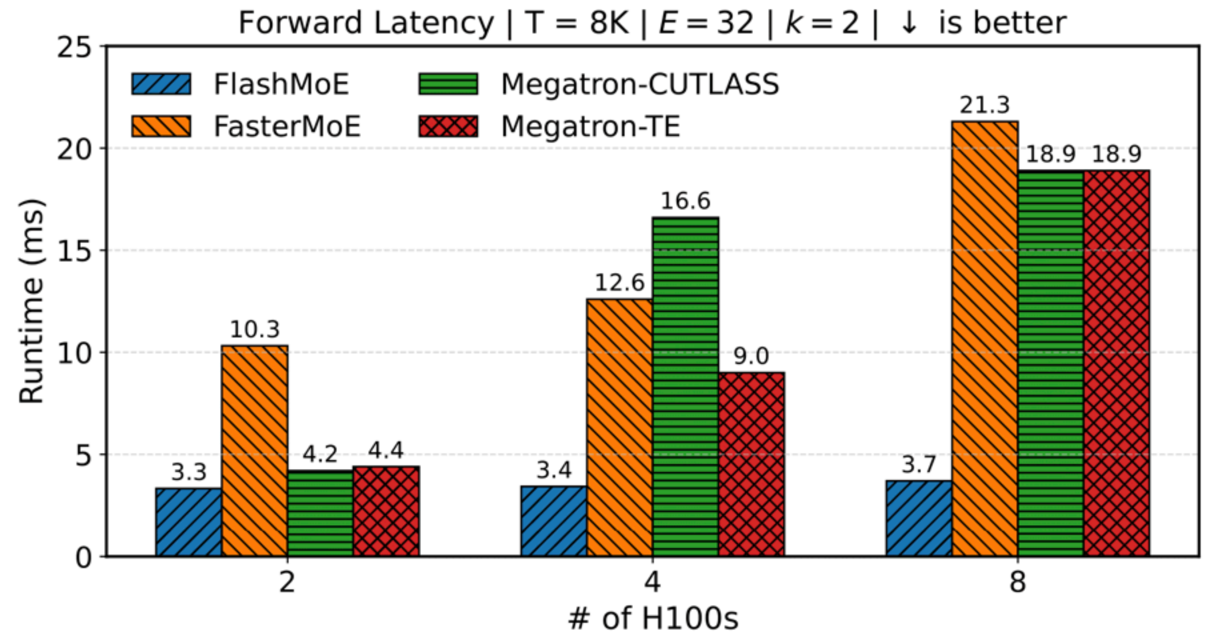
Complete DMoE Kernel Fusion

FlashMoE gives:

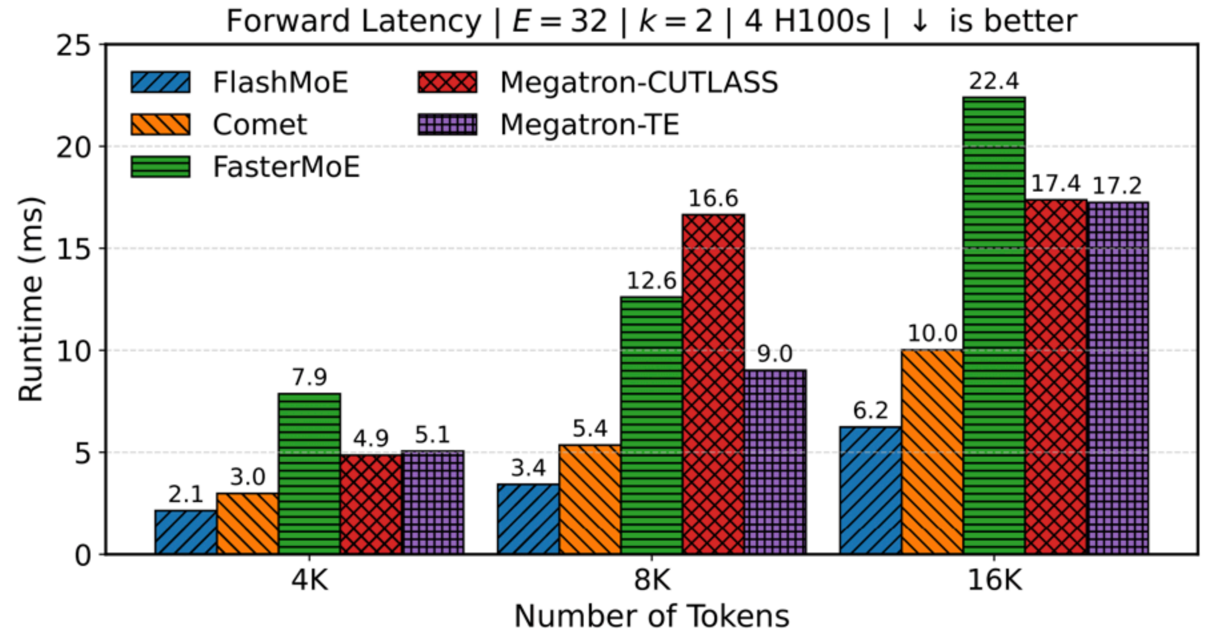
- **9x** higher GPU utilization
- **6x** faster E2E latency
- **Constant** expert scalability
- **4x** better communication efficiency



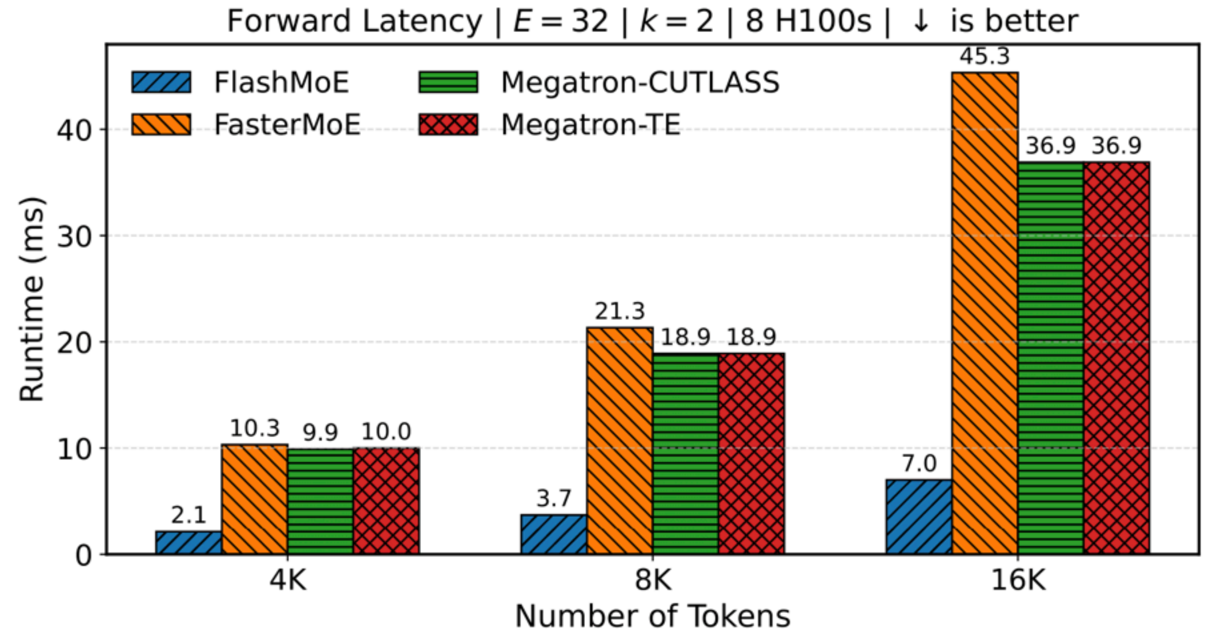
GPU SM Utilization



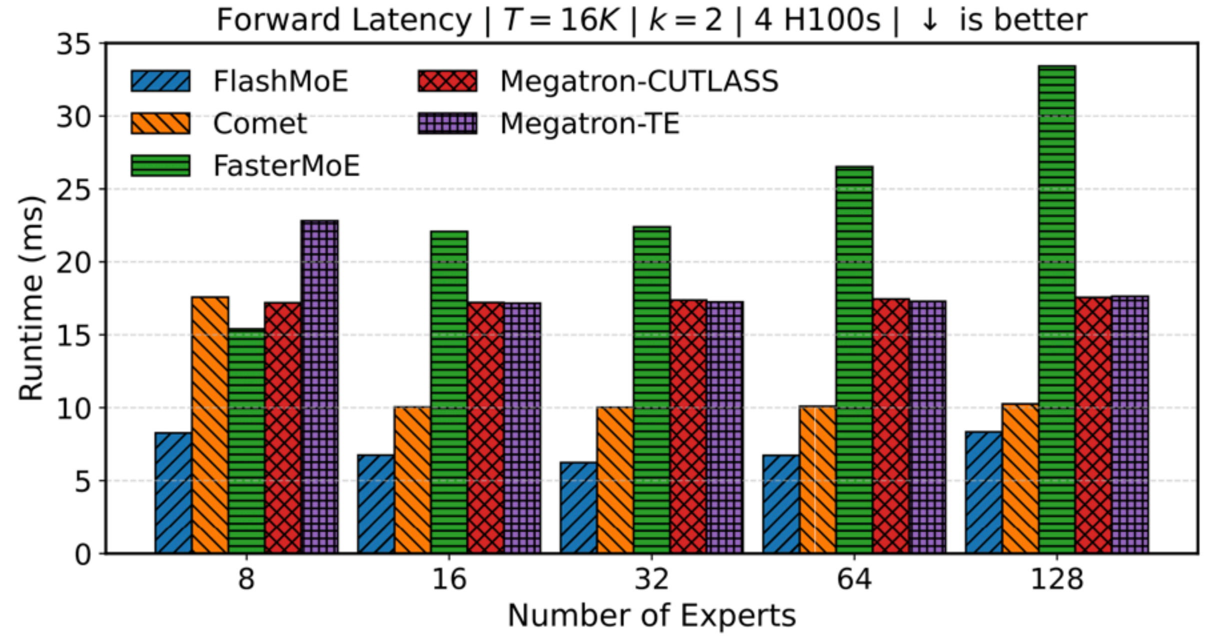
Scaling GPUs



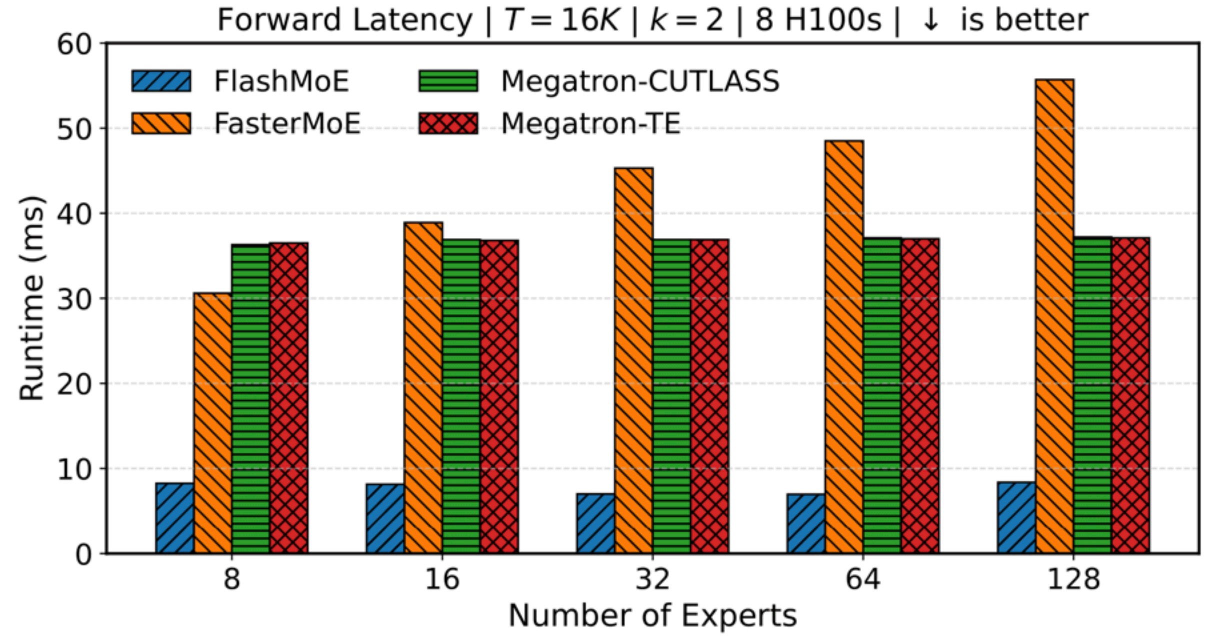
Scaling Tokens (4 GPUs)



Scaling Tokens (8 GPUs)



Scaling Experts (4 GPUs)



Scaling Experts (8 GPUs)