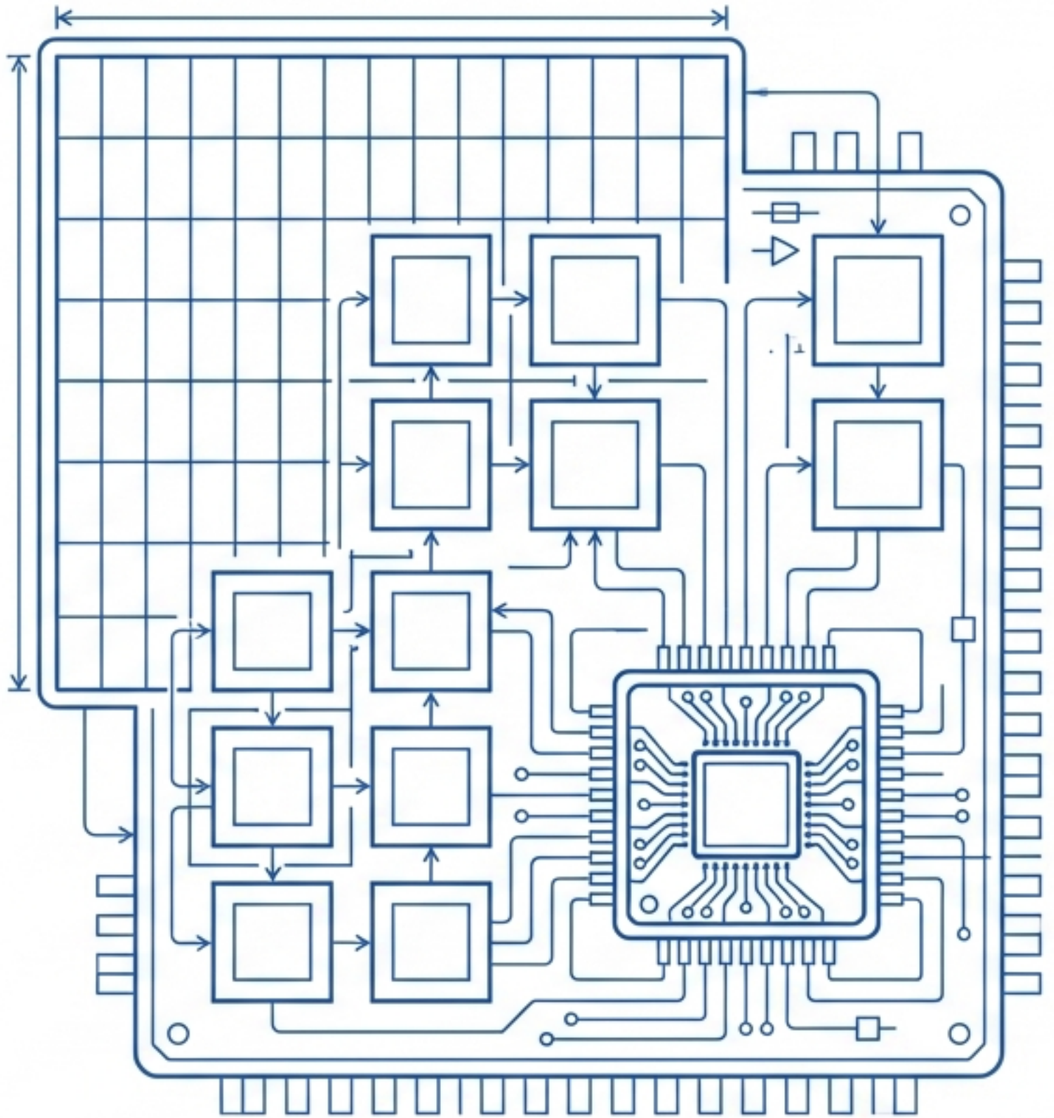
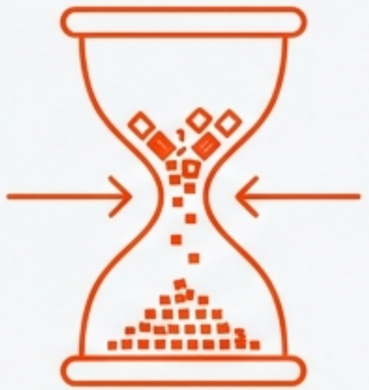
FlashAttention: Breaking the Memory Wall

The Evolution of Fast, IO-Aware Exact Attention (Featuring FlashAttention-3)



Based on research by Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré.

Accelerating Transformers by Minimizing Data Movement



The Problem

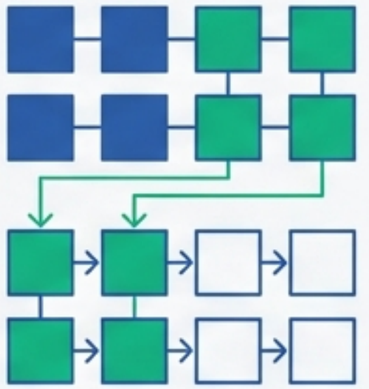
Attention mechanisms bottleneck Long Context LLMs.

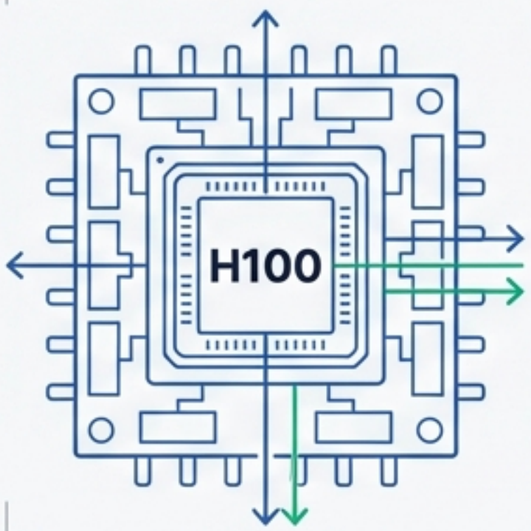
The “Memory Wall” means data movement is slower than computation.

The Paradigm Shift

FlashAttention-1 & 2 reordered computation (Tiling) to minimize memory access.

Result: 2-4x training speedup.





The New Frontier

FlashAttention-3 leverages NVIDIA Hopper (H100) hardware.

Uses Warp-Specialization and Asynchrony to overlap compute and data movement.

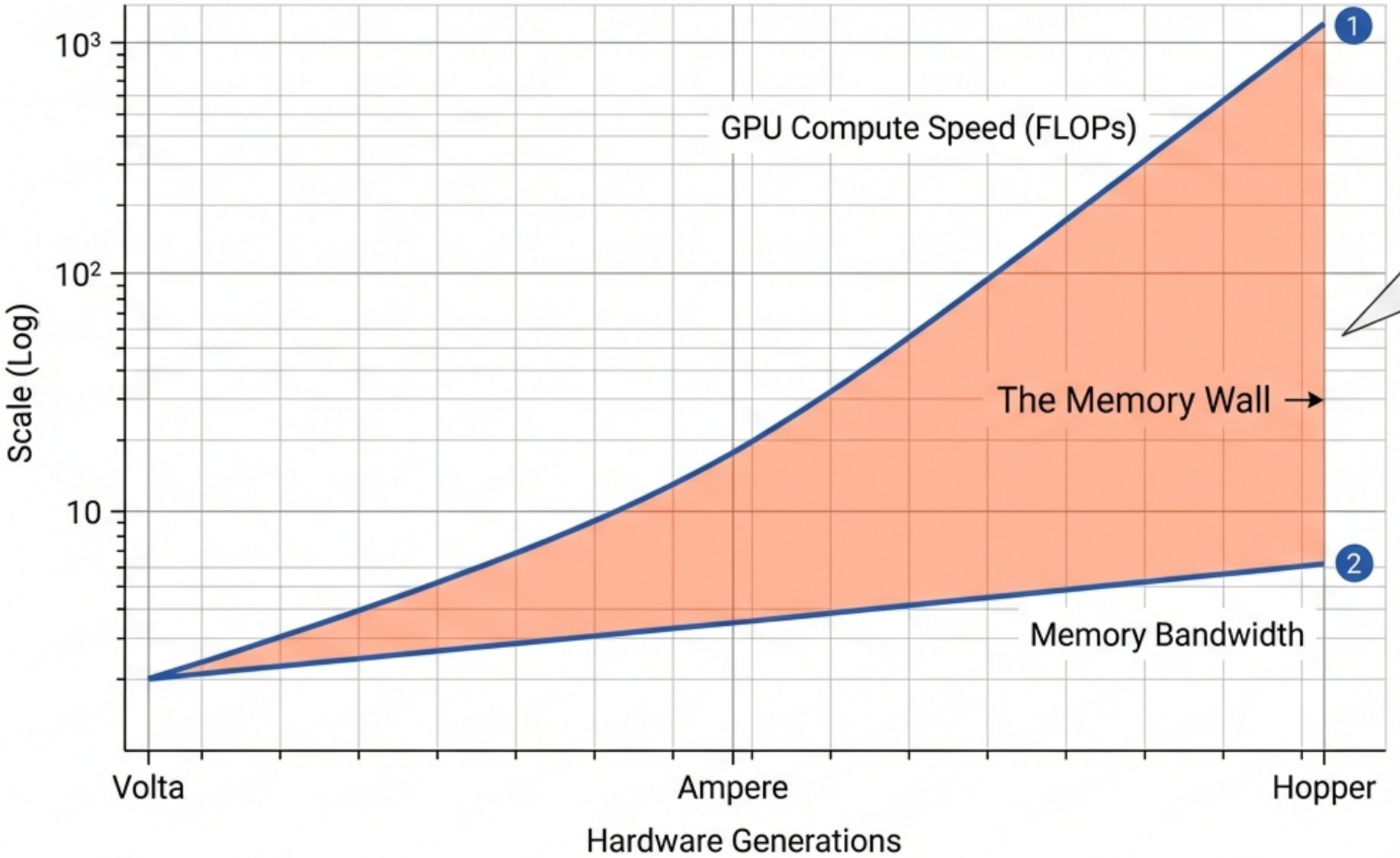
The Impact

75% Hardware Utilization (up from 35%).

Up to 1.2 PFLOPS in FP8.

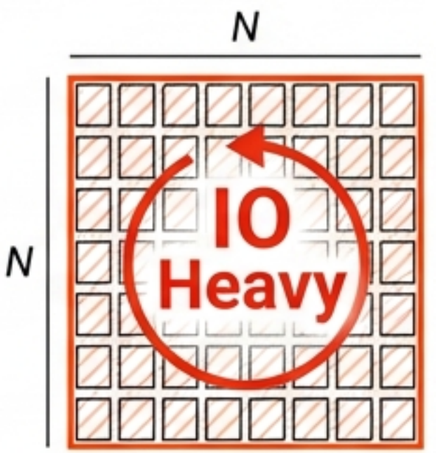
Enables 128k+ context windows.

The Bottleneck is Memory, Not Math



Computations are cheap; Data movement is expensive.

Operations like Softmax and Normalization are memory-bound, leaving GPU cores idle.

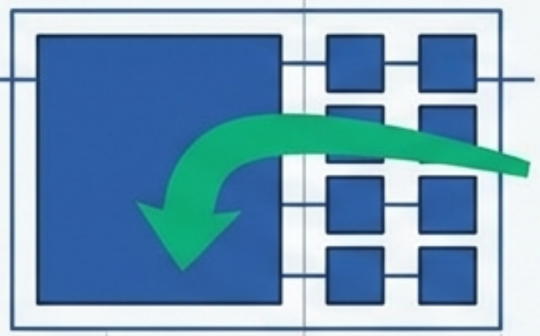


Attention Matrix $N \times N$.
Quadratic Scaling $O(N^2)$

The GPU Memory Hierarchy

SRAM (On-Chip Cache)

Capacity: 192KB/SM.
Bandwidth: ~19 TB/s.
“The Kitchen Counter”



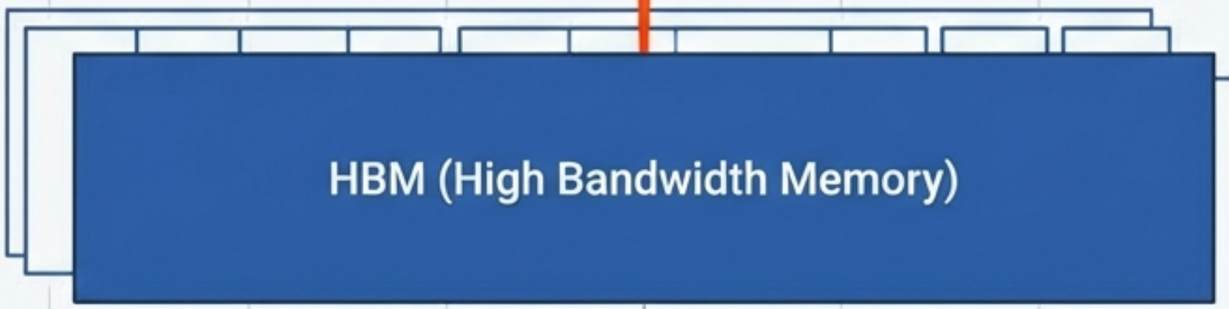
Compute Cores

Fast & Cheap

FlashAttention Goal:

Keep data in the “Kitchen” (SRAM) and minimize trips to the “Store” (HBM).

Slow & Expensive

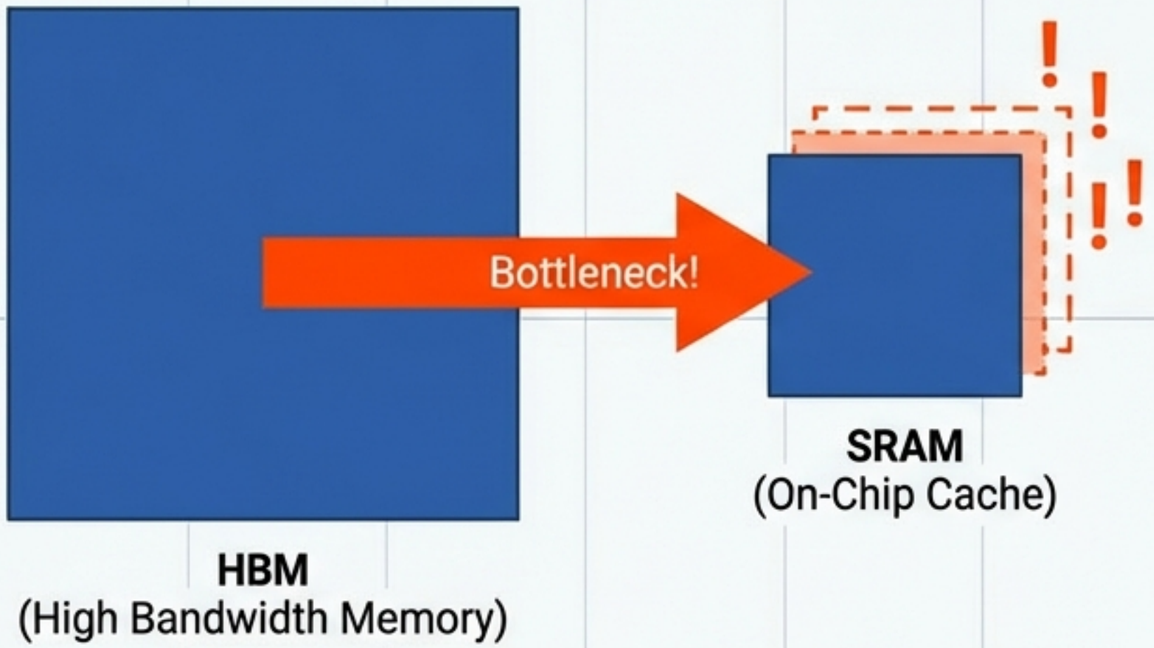


HBM

Capacity: 40-80GB.
Bandwidth: ~1.5 TB/s.
“The Grocery Store”

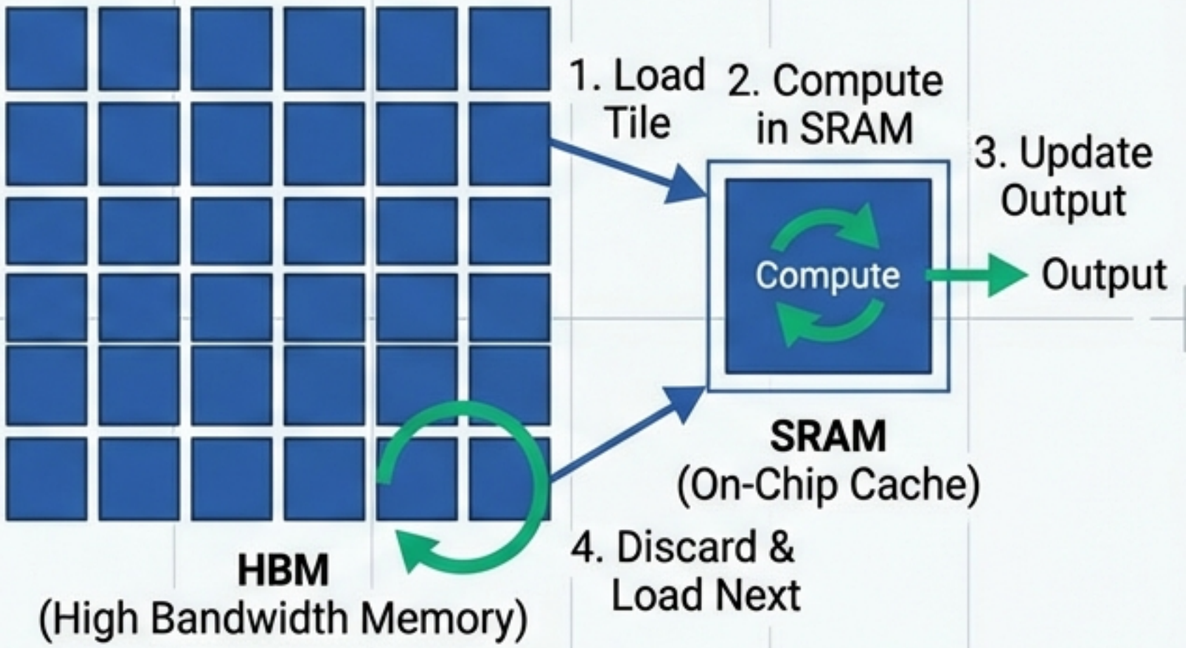
IO-Awareness: Tiling & Recomputation

Standard: Quadratic Memory $O(N^2)$



Quadratic Scaling $O(N^2)$ - Entire $N \times N$ matrix loaded, overflows cache.

Tiled: Linear Memory $O(N)$

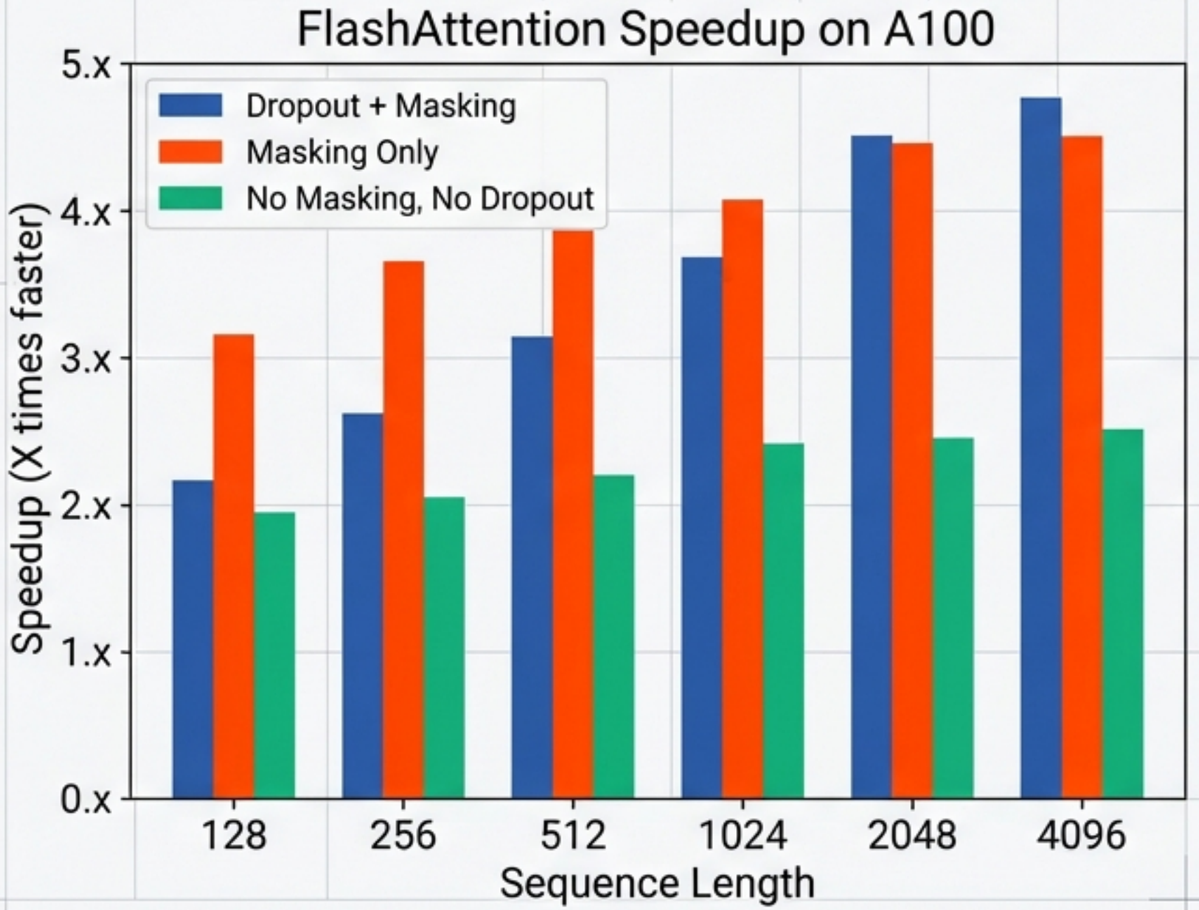


Linear Scaling $O(N)$ - Only small $B \times B$ tiles loaded, compute fits in cache.

Backward Pass Strategy: Instead of storing huge intermediate matrices, recompute them on-the-fly from inputs. Costs FLOPs, saves massive IO.

The Efficiency Gap on Hopper GPUs

The Baseline



Hardware outpaced Software.

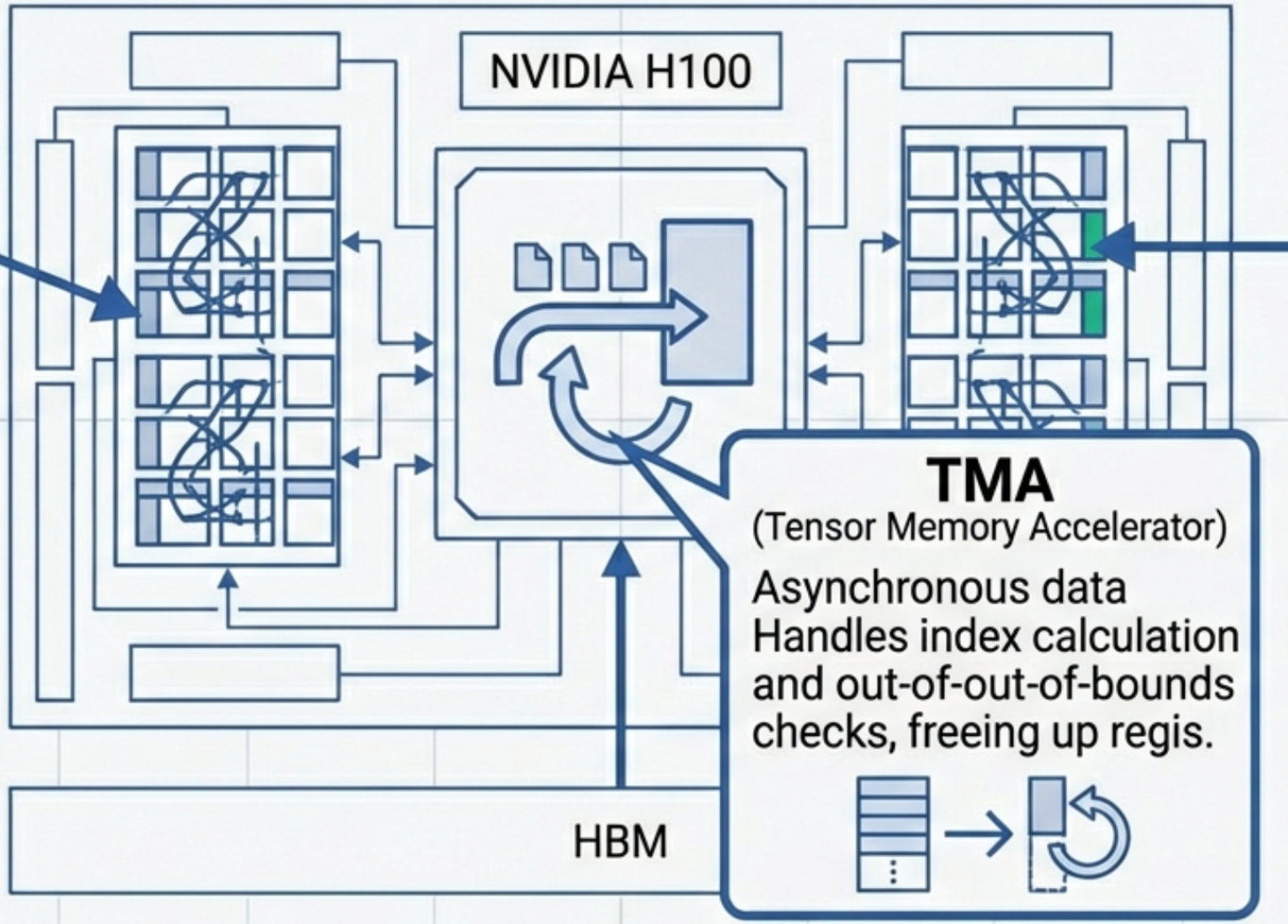
The Problem

On NVIDIA A100 (Ampere): **~70% Utilization.**

On NVIDIA H100 (Hopper): **~35% Utilization (FlashAttention-2).**

Why? Hopper introduced asynchronous instructions (`WGMMA`, `TMA`) that the old algorithm didn't exploit. Threads were left waiting for data.

Built for Hopper: New Hardware Capabilities



WGMMA

(Warp-group Matrix Multiply-Accumulate)

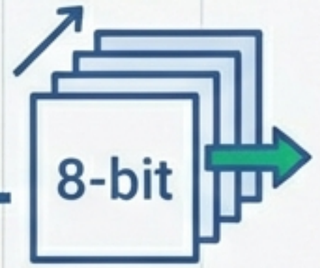
High throughput tensor core instructions. Much faster than `mma.sync`.

```
mma.sync << WGMMA
```

TMA

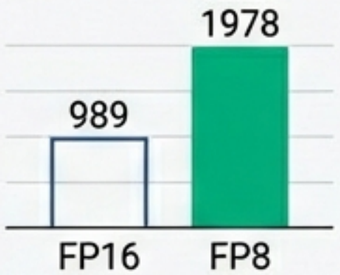
(Tensor Memory Accelerator)

Asynchronous data Handles index calculation and out-of-out-of-bounds checks, freeing up regis.



FP8 Support

Doubles Tensor Core throughput.
1978 TFLOPS (FP8)
vs 989 TFLOPS (FP16).

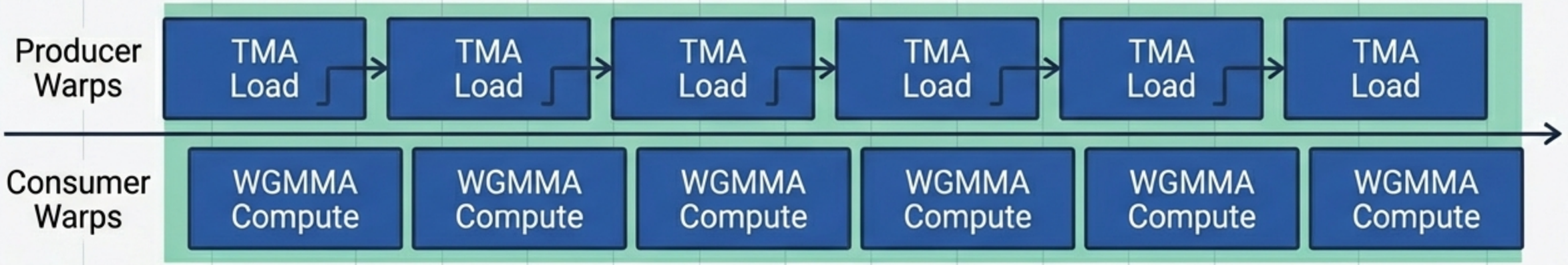


FlashAttention-3 uses NVIDIA's **CUTLASS** library to abstract these features.

Optimization 1: Asynchrony & Warp-Specialization

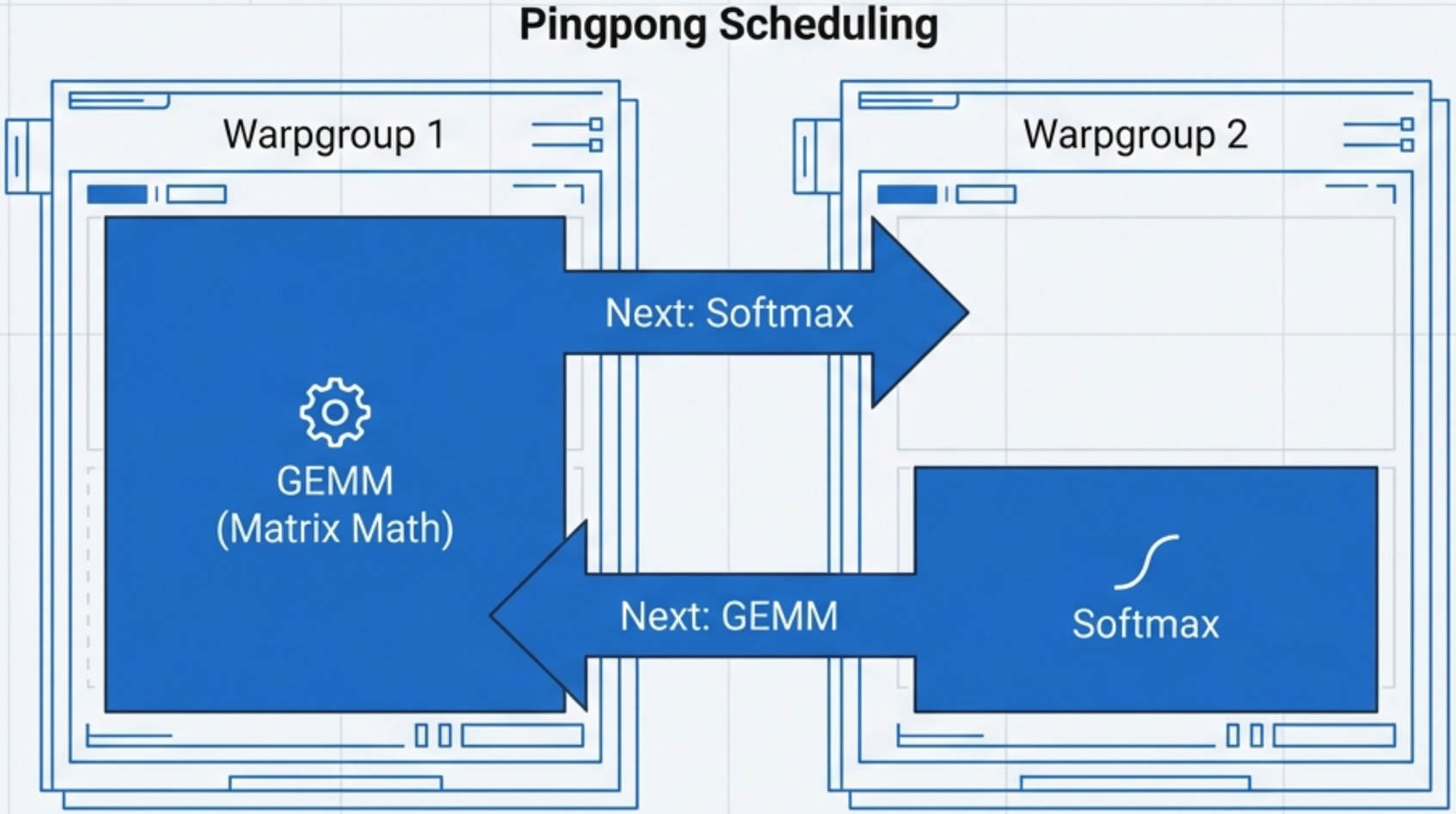
Sequential - The Old Way

Warp-Specialized - FA3



Producer Warps issue TMA instructions. Consumer Warps perform Math. Execution pipeline is saturated.

Optimization 2: Overlapping GEMM and Softmax



The Bottleneck:

GEMM is fast (Tensor Cores).

Softmax is slow (Multi-function units).

Softmax can **stall** the math.

The Fix:

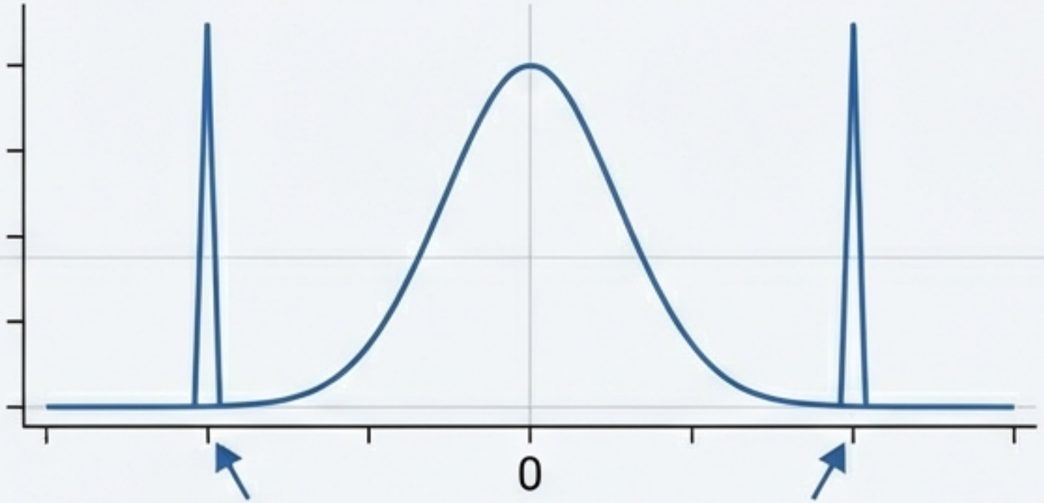
While Warpgroup 1 does heavy Matrix Math, Warpgroup 2 quietly handles Softmax in the background.

The **Tensor Cores never stop**.

Optimization 3: Low-Precision with FP8

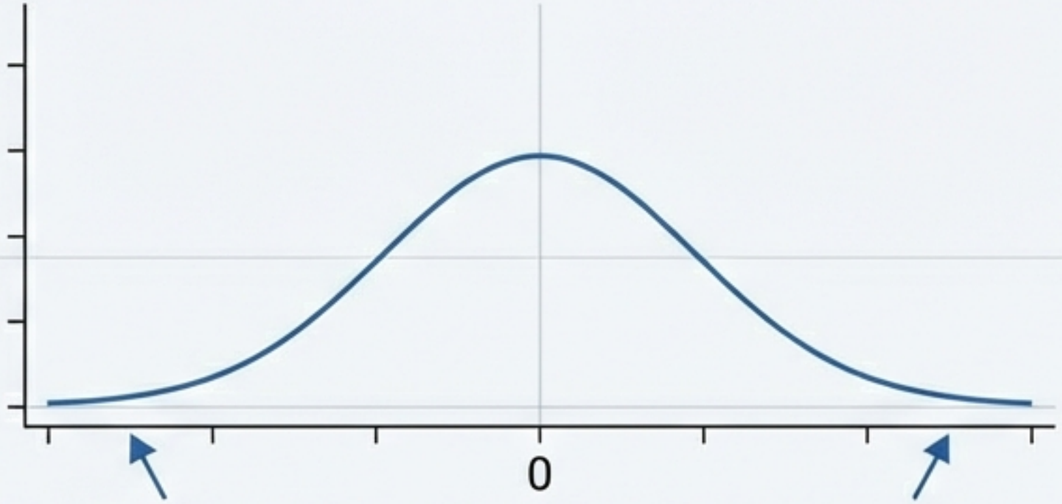
Solving the Accuracy vs. Speed Trade-off via Incoherent Processing

Standard FP8 Risk



Outliers cause quantization error

Incoherent Processing

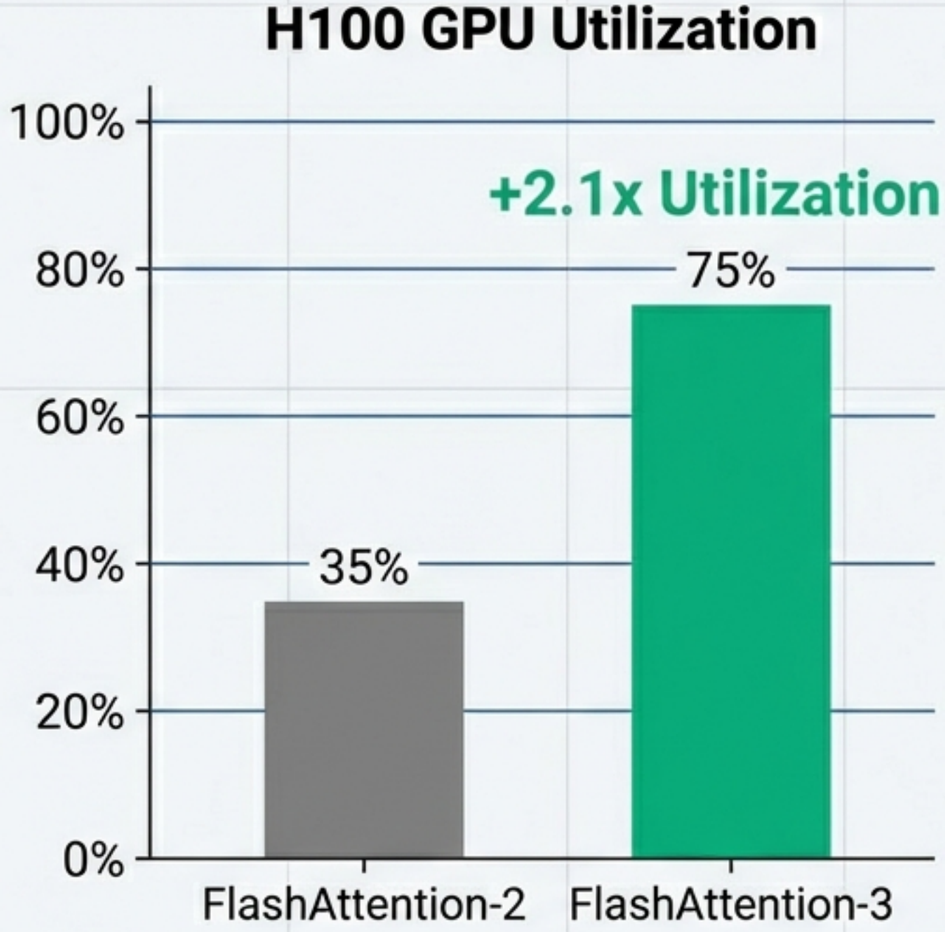


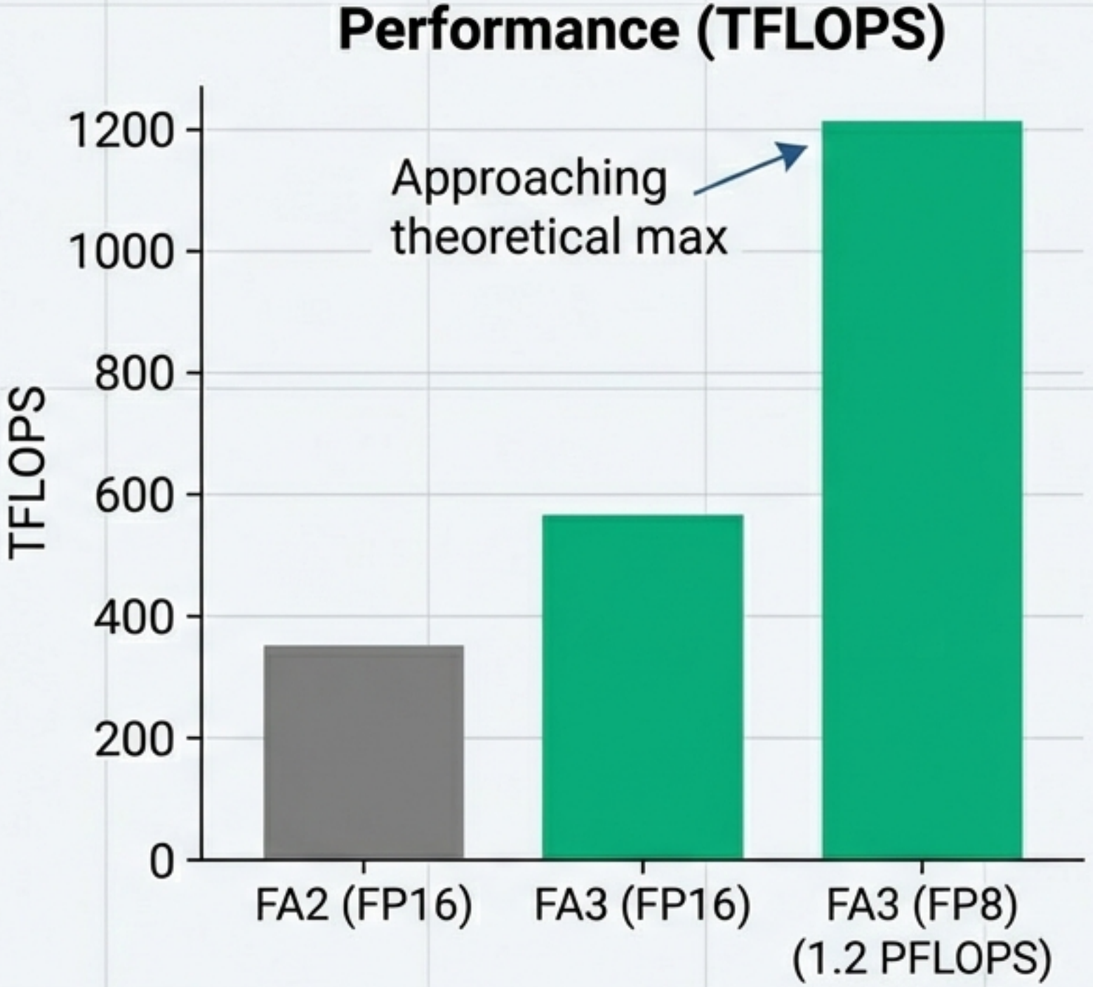
Hadamard Transform “spreads out” outliers

$$Q' = Q \times H \text{ (Hadamard Transform)}$$

Result: **1.2 PFLOPS** speed with **2.6x smaller error** than baseline FP8.

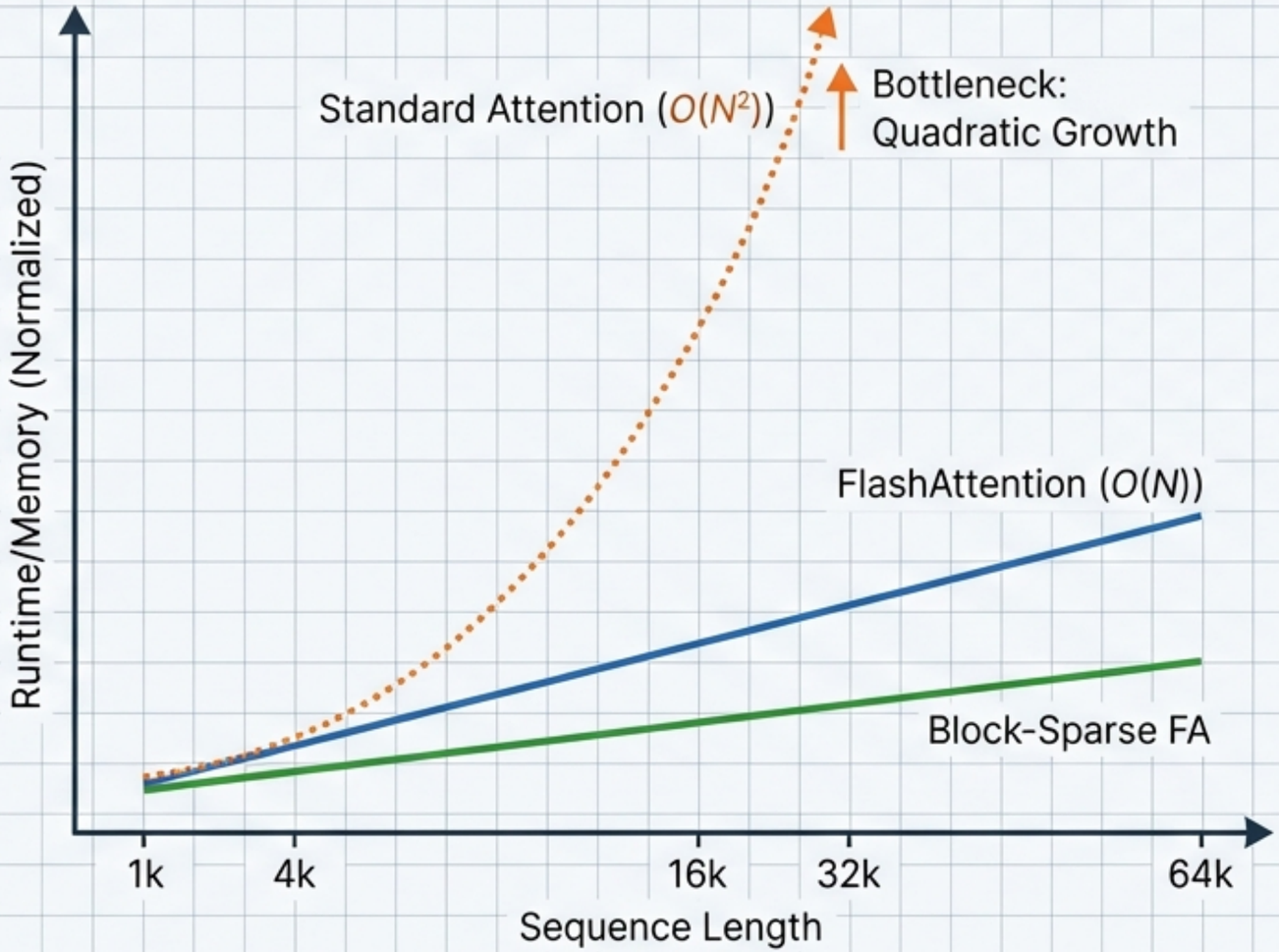
Benchmark Results: Speed and Utilization





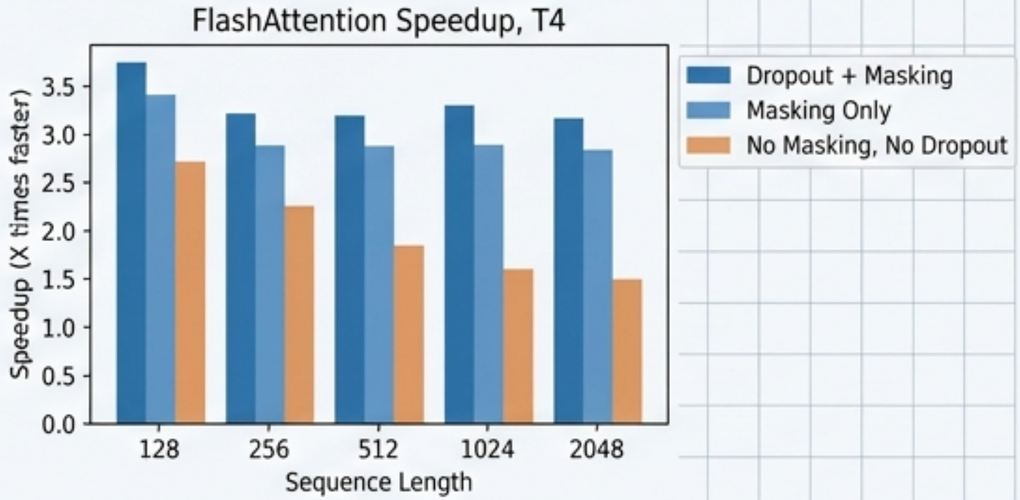
Unlocking Massive Context Windows

COMPARATIVE SCALING: RUNTIME & MEMORY vs. SEQUENCE LENGTH



KEY ACHIEVEMENTS

- **Path-X Benchmark:** First better-than-random performance on 16k sequence length.
- **Path-256 Benchmark:** Scaled to 64k sequence length via Block-Sparse extension.
- **Enables 128k+ contexts** (GPT-4, Llama 3).



The FlashAttention Genealogy

FlashAttention-1

- **Era:** 2022
- **Core Innovation:** IO-Awareness, Tiling, Recomputation.
- **Impact:** Linear Memory Complexity $O(N)$.

FlashAttention-2

- **Era:** 2023
- **Core Innovation:** Improved Parallelism, Work Partitioning.
- **Target:** Optimized for A100 (Ampere).

FlashAttention-3

- **Era:** 2024
- **Core Innovation:** Warp-Specialization, Asynchrony, FP8 Incoherence.
- **Target:** Optimized for H100 (Hopper).
- **Result:** **75% Utilization.**

Integration and Usage

```python
import flash_attn

# Scaled dot product attention
# Automatically selects best implementation
output = flash_attn.flash_attn_func(
    q, k, v,
    dropout_p=0.0,
    causal=True
)
```

Ecosystem & Requirements

Integrations

PyTorch 2.2+

Hugging Face Transformers

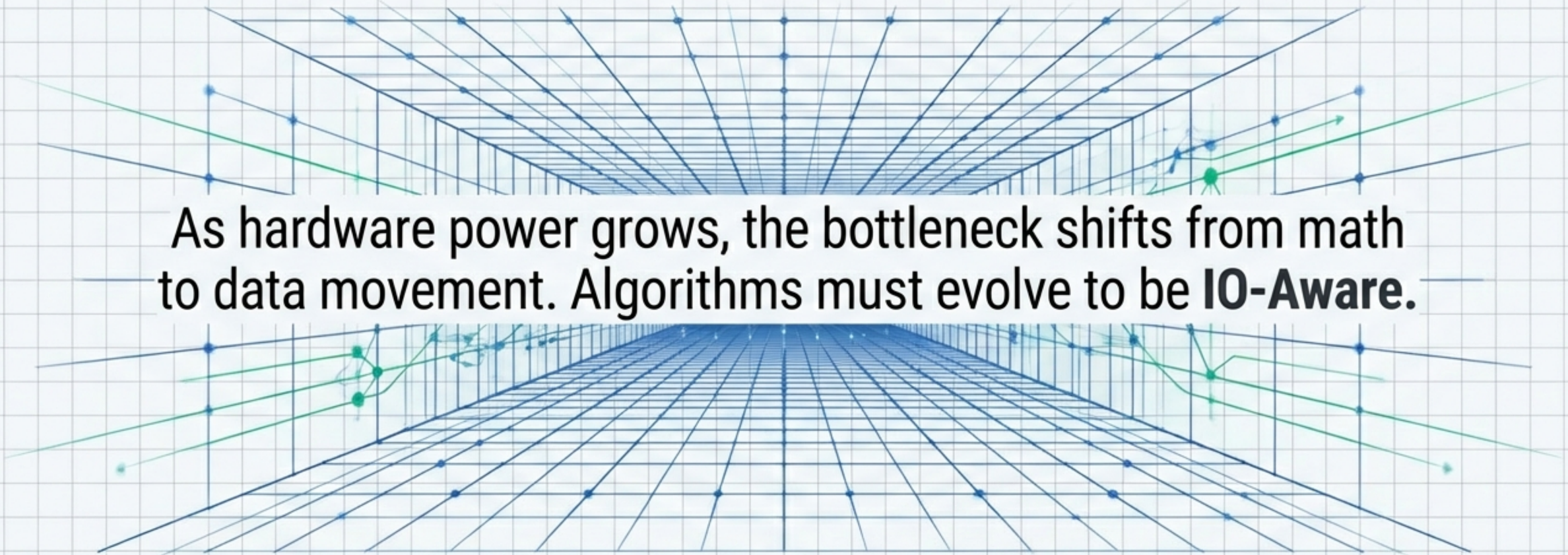
vLLM TGI

Requirements Checklist

- [x] NVIDIA H100/H800 GPU
- [x] CUDA 12.3+

```
pip install flash-attn --no-build-isolation
```

The Future is IO-Aware



As hardware power grows, the bottleneck shifts from math to data movement. Algorithms must evolve to be **IO-Aware.**

Resources:

GitHub: Dao-AILab/flash-attention

Paper: FlashAttention-3: Fast and Accurate Attention with Asynchrony and Low-precision