15-442/15-642: Machine Learning Systems

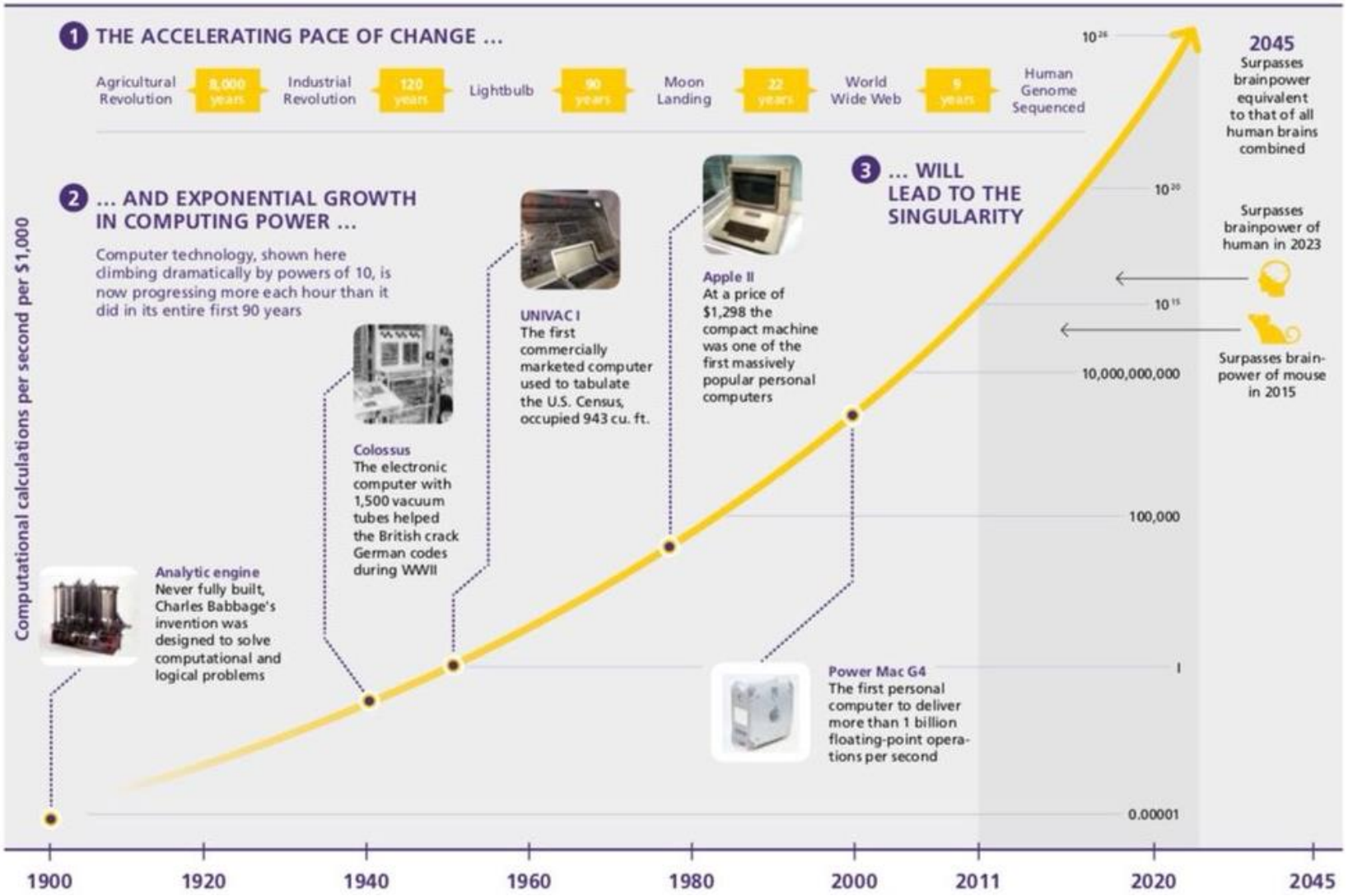
Advanced Topics: ML Superoptimization

Tianqi Chen and Zhihao Jia

Carnegie Mellon University

What are the fundamental driving forces behind the success of ML?

Compute Per Second Per Dollar

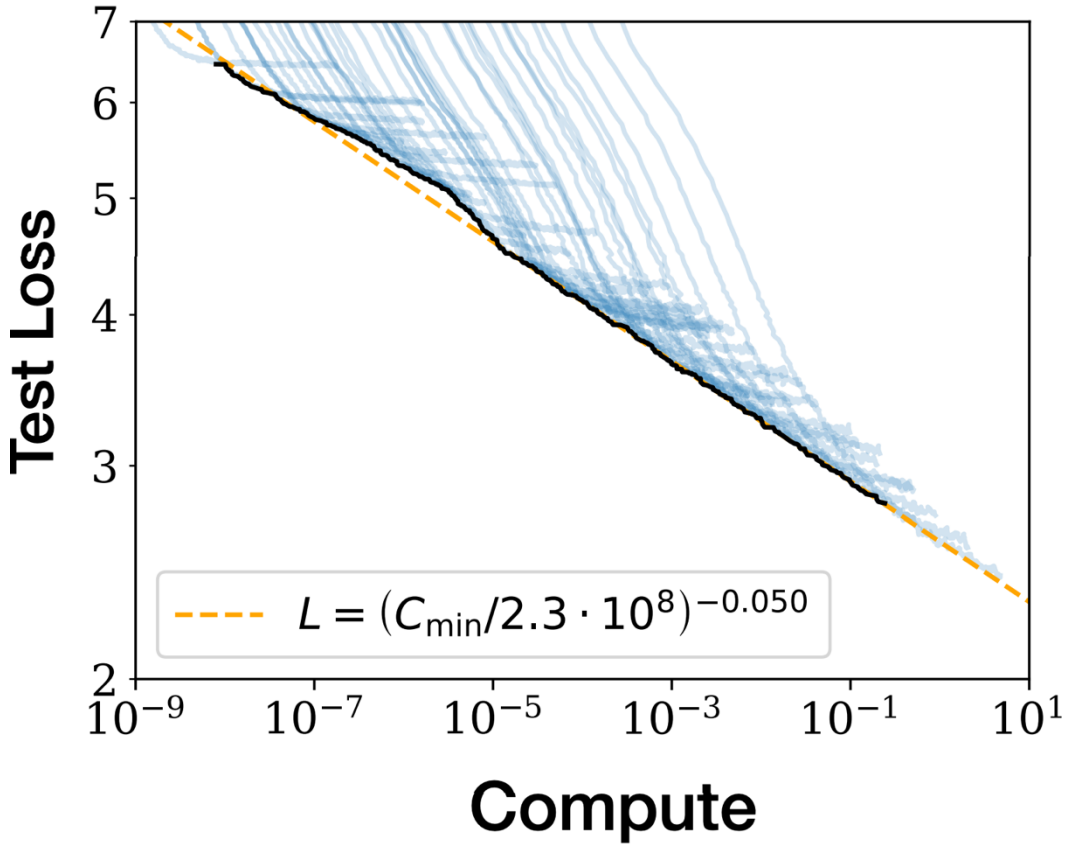


Surpass human brainpower in 2023

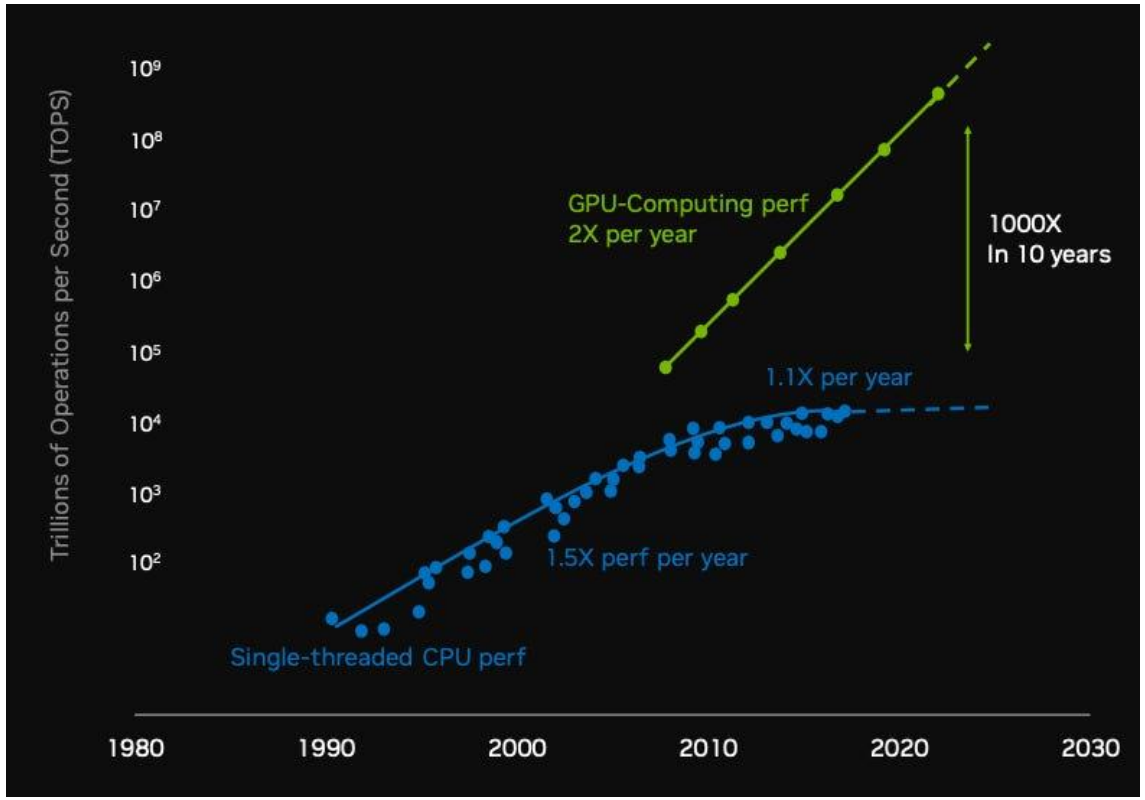
* Ray Kurzweil. *The Singularity Is Near: When Humans Transcend Biology*. 2005

Scaling Law in ML

Improving model accuracy & capability ← Scaling training & inference compute ← Hardware parallelization and specialization

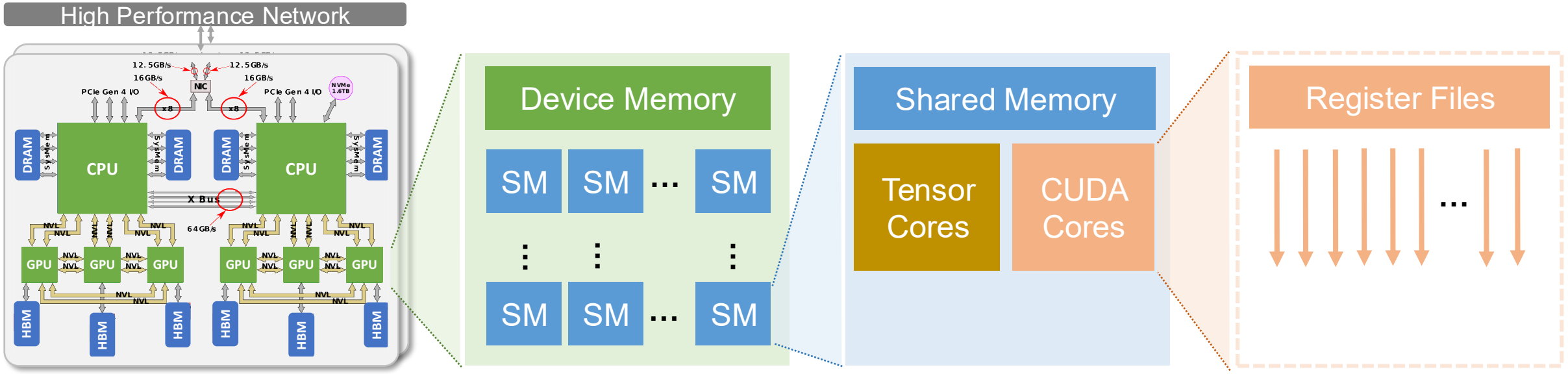


Source: OpenAI



Source: NVIDIA

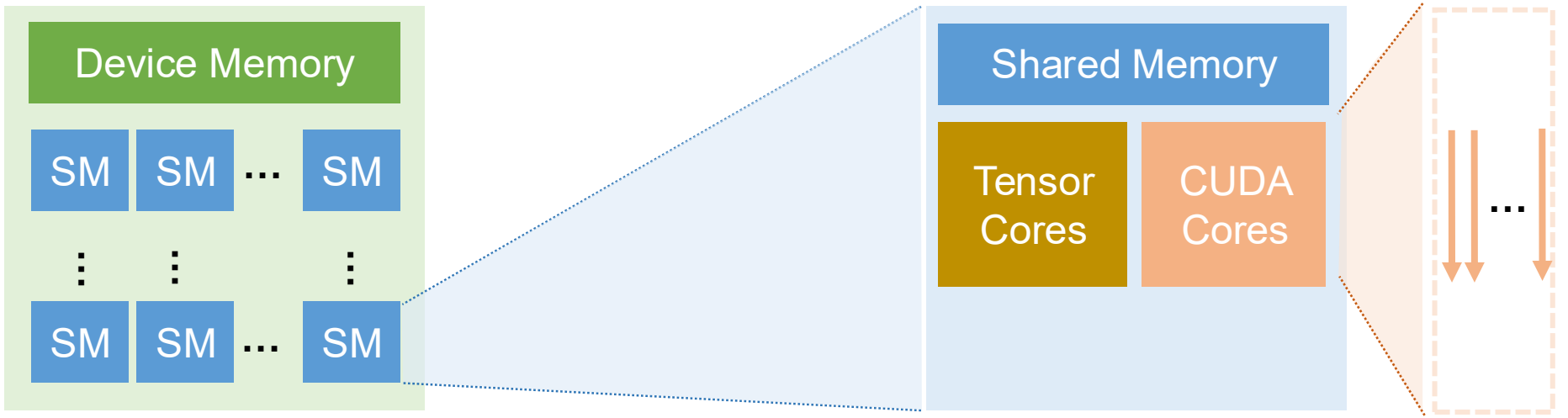
ML Hardware is Massively Parallel, Highly Heterogeneous



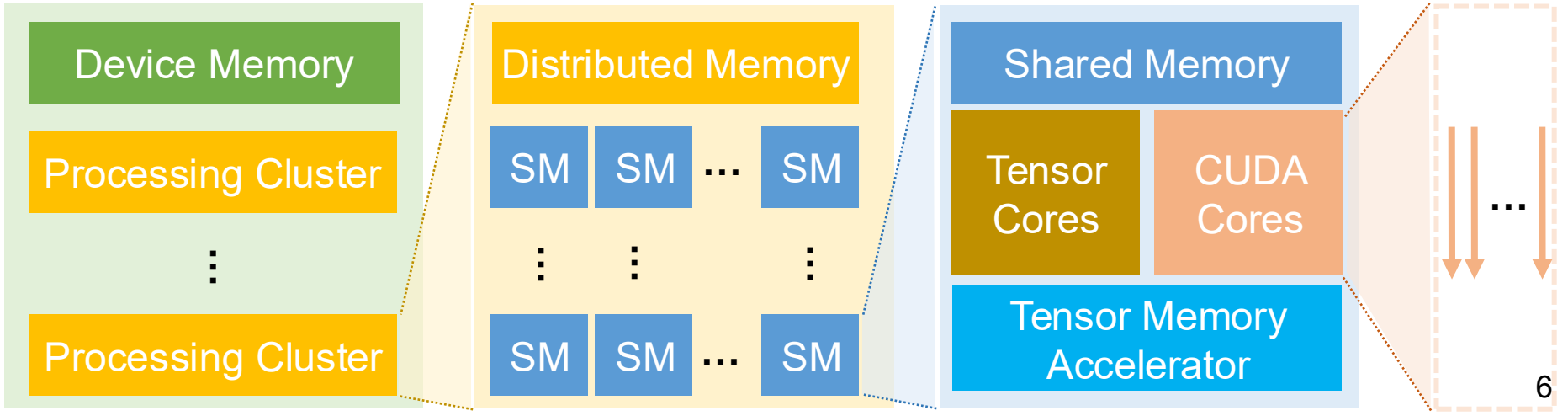
1K-100K GPUs × 100s streaming multiprocessors (SMs) × 128 CUDA cores 4 tensor cores = 10M-1B CUDA cores 400K-40M tensor cores

ML Hardware is Quickly Evolving

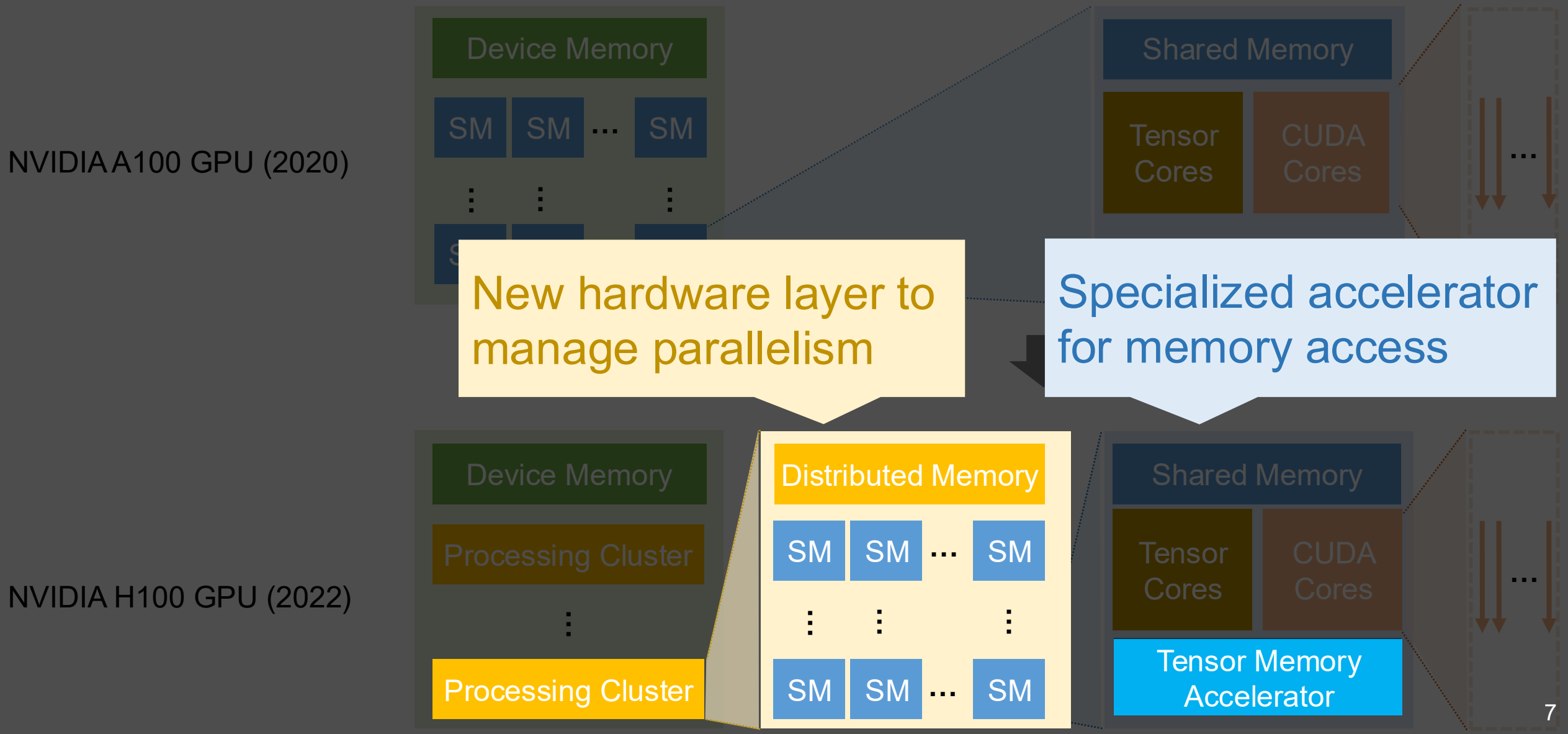
NVIDIA A100 GPU (2020)



NVIDIA H100 GPU (2022)



ML Hardware is Quickly Evolving



This Lecture: ML Superoptimization

Optimize ML performance by simultaneously considering

- Algebraic transformations
- Custom GPU kernels
- Schedule transformations

Mirage: A SuperOptimizer for ML

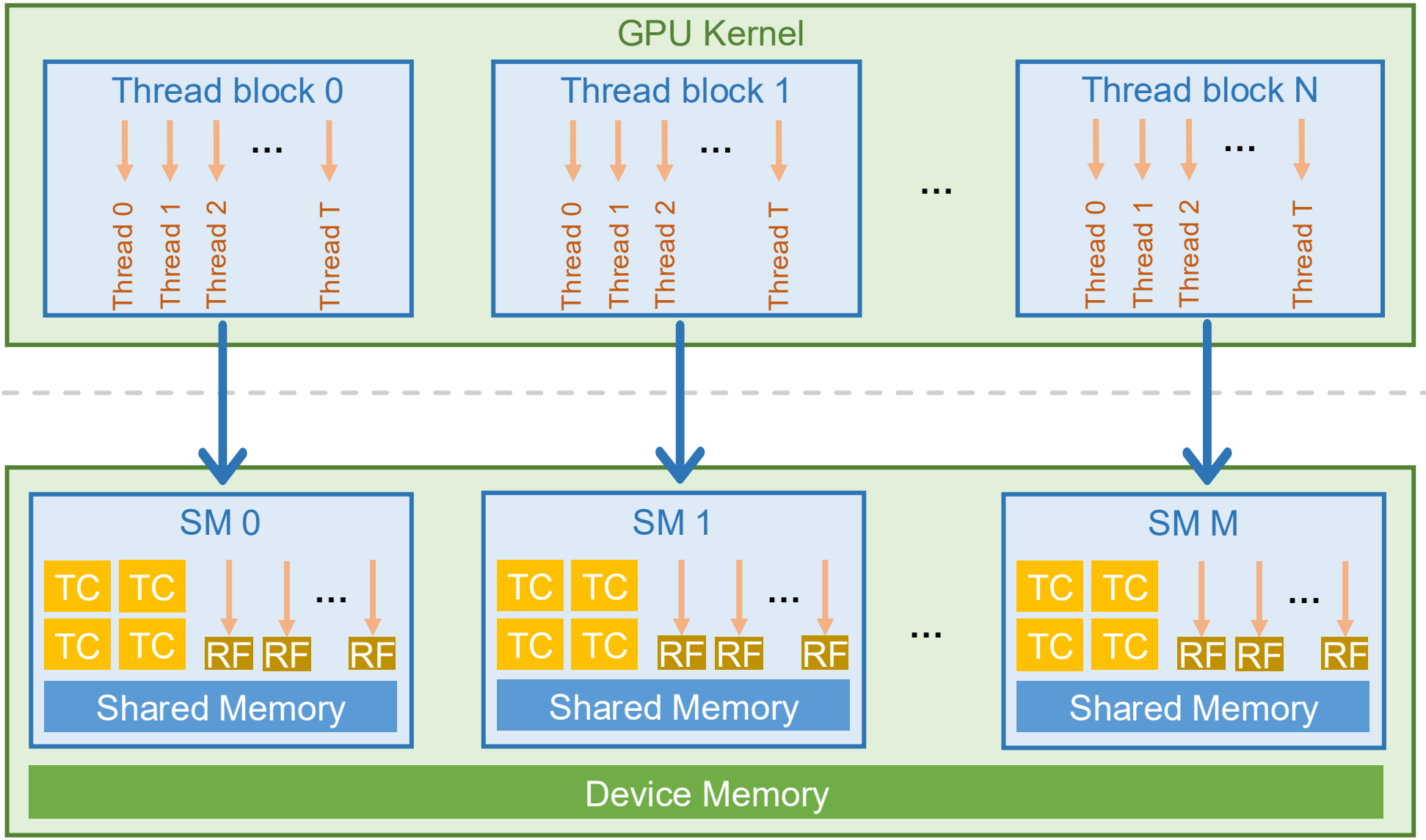
Key idea: automatically generate highly-optimized GPU kernels for DNNs

- **Less engineering effort**: thousands lines of CUDA code → a few lines of Python code in Mirage
- **Better performance**: outperform existing systems by 1.1-3.5x
- **Faster adaptation:** day-0 support for new models; no manual effort

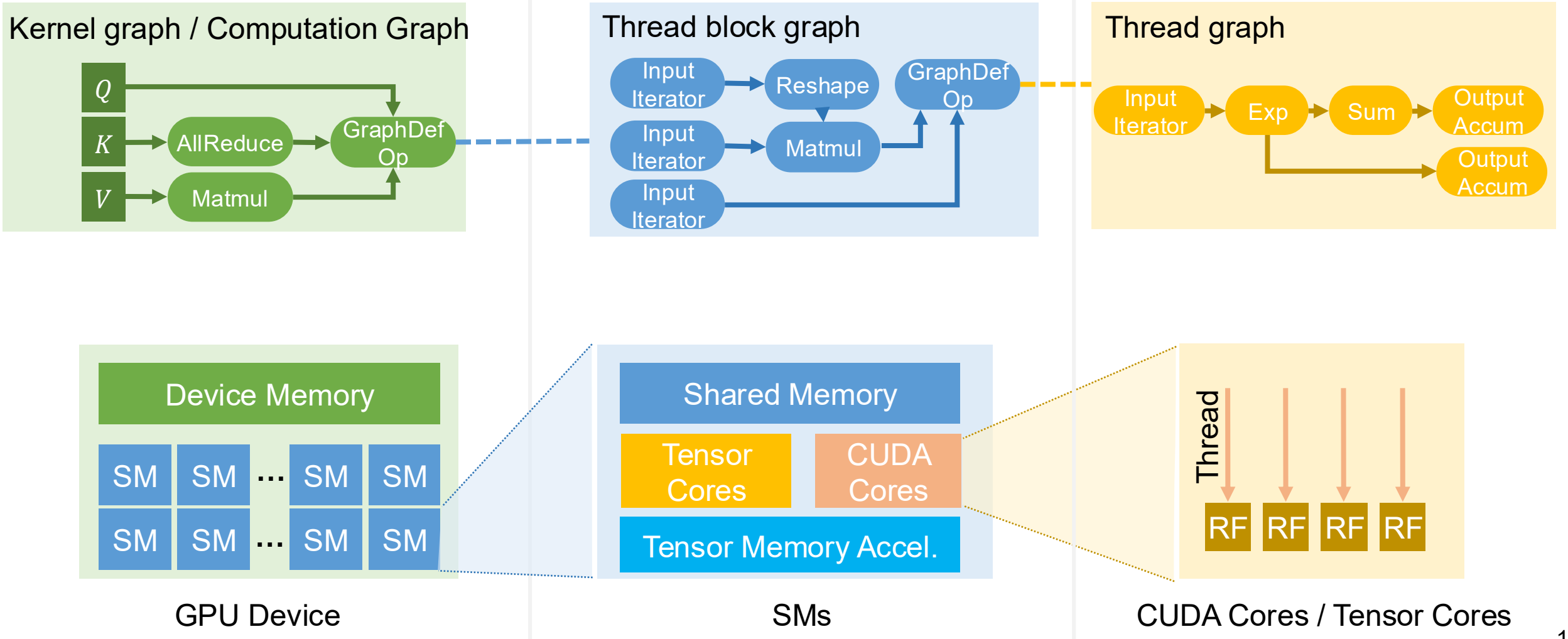
Recap: GPU Programming 101

Programming Abstraction

Hardware Architecture

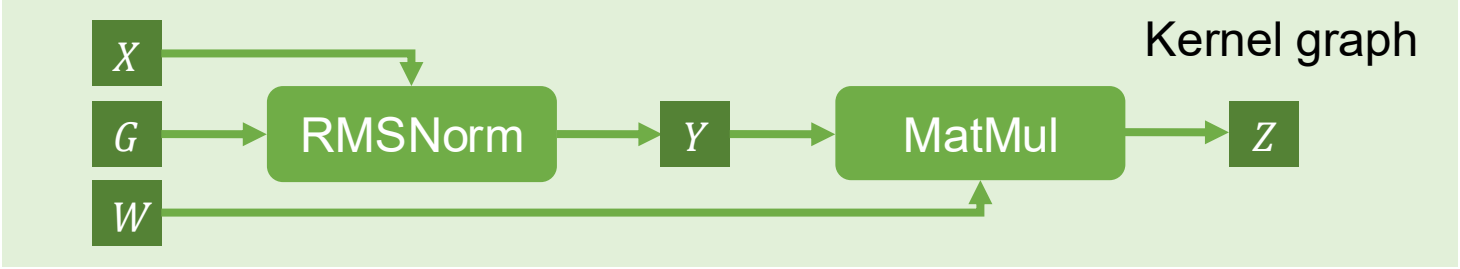


μGraphs: Hierarchical Graph Representation



Example: RMSNorm & MatMul in LLMs

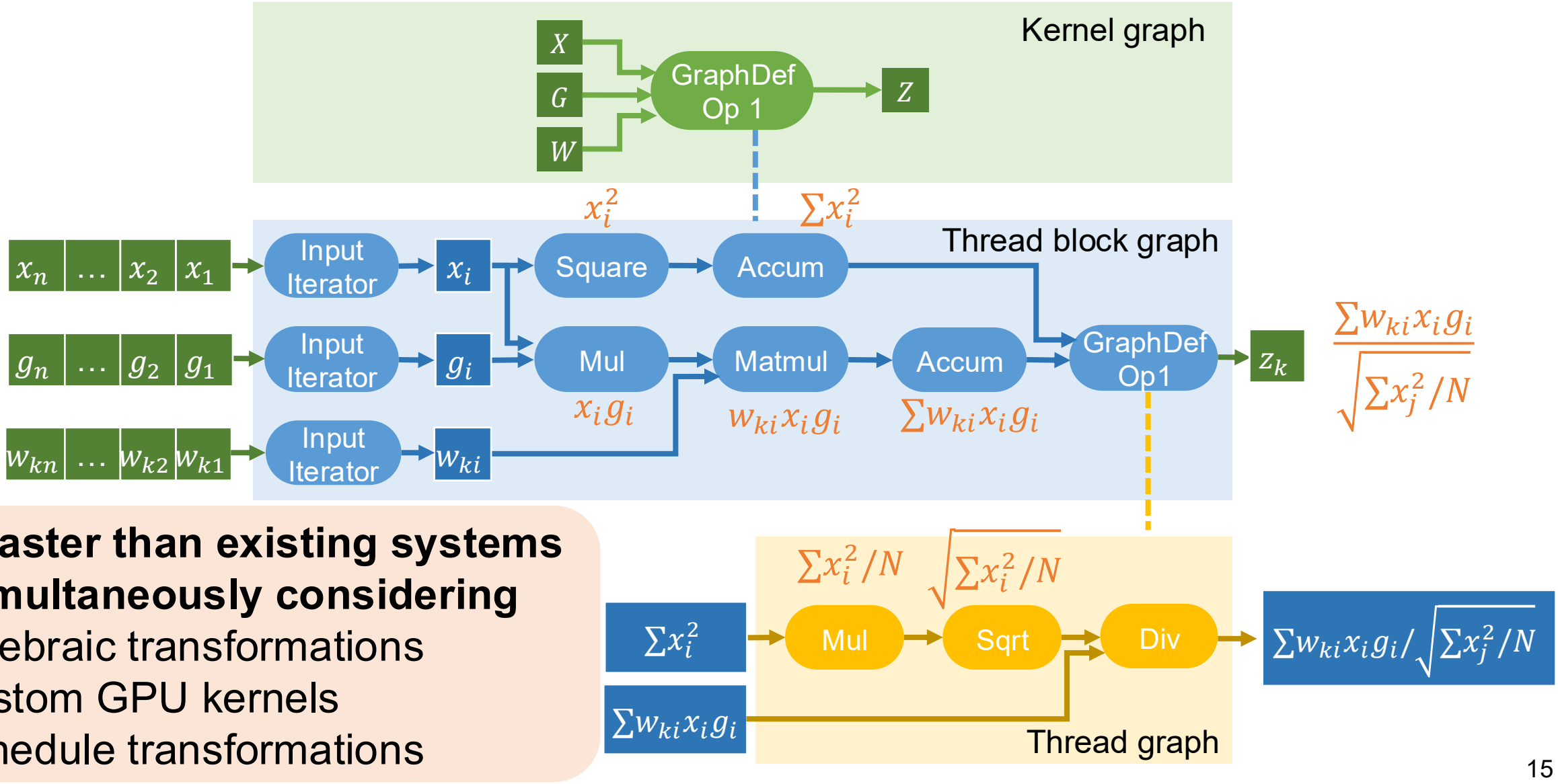
Existing systems launch two kernels since Y does not fit in shared memory



$$y_i = \frac{x_i g_i}{\sqrt{\frac{1}{N}\sum_j x_j^2}} \qquad z_i = \sum_k w_{ik} y_k$$

Performance issues:
1. No shared memory reuse
2. Kernel launch overhead

Best Discovered μGraph for RMSNorm & MatMul



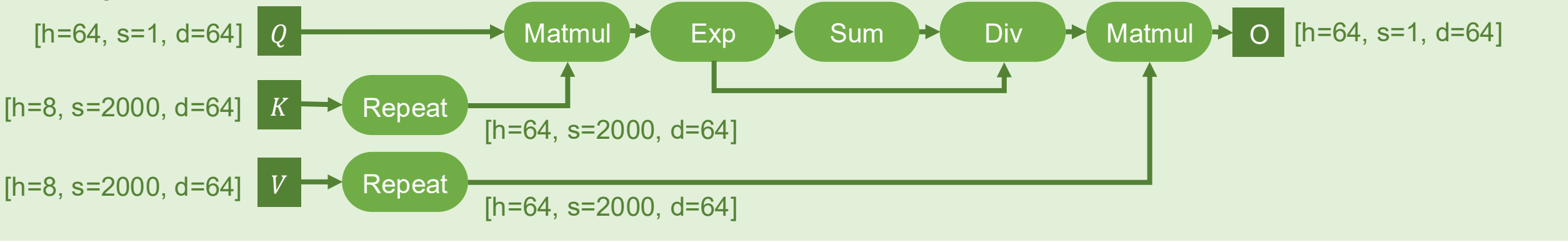
2.2x faster than existing systems by simultaneously considering

- Algebraic transformations
- Custom GPU kernels
- Schedule transformations

Example: Group-Query Attention in PyTorch

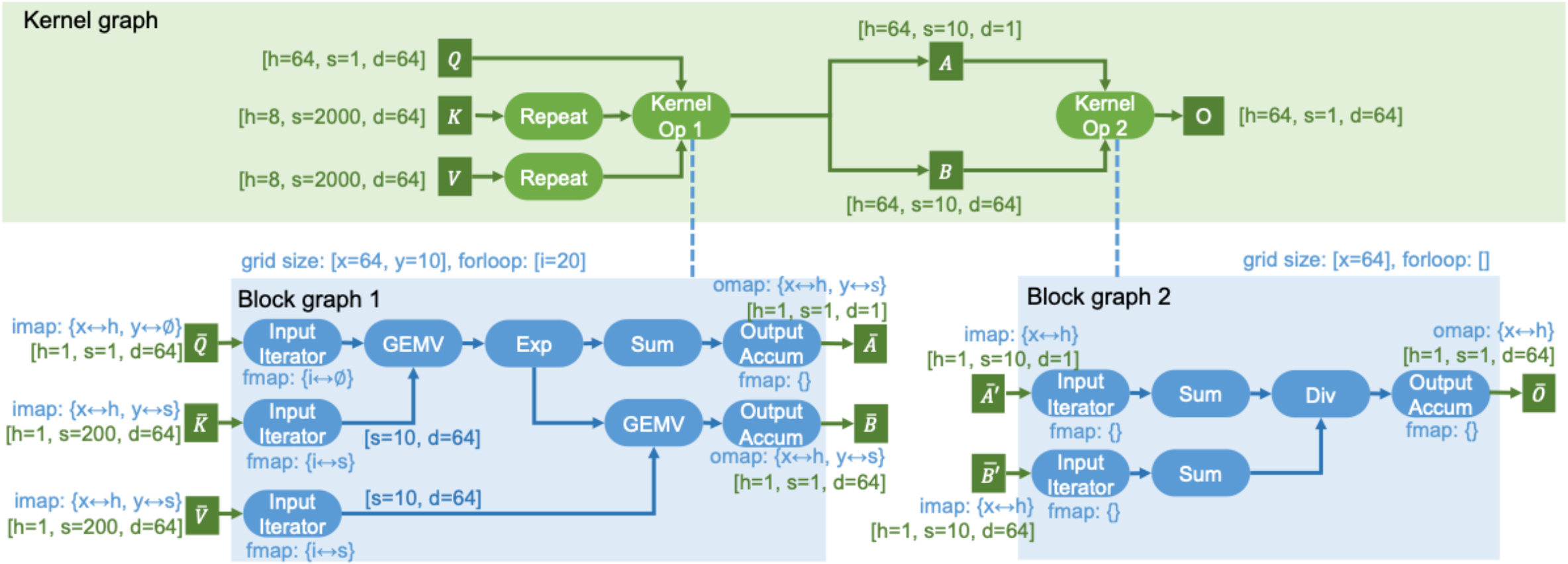
```
A = torch.matmul(Q, K.repeat(8))
S = torch.softmax(A)
O = torch.matmul(S, V.repeat(8))
```

Kernel graph



```
A = torch.matmul(Q, K.repeat(8))
S = torch.softmax(A)
O = torch.matmul(S, V.repeat(8))
```

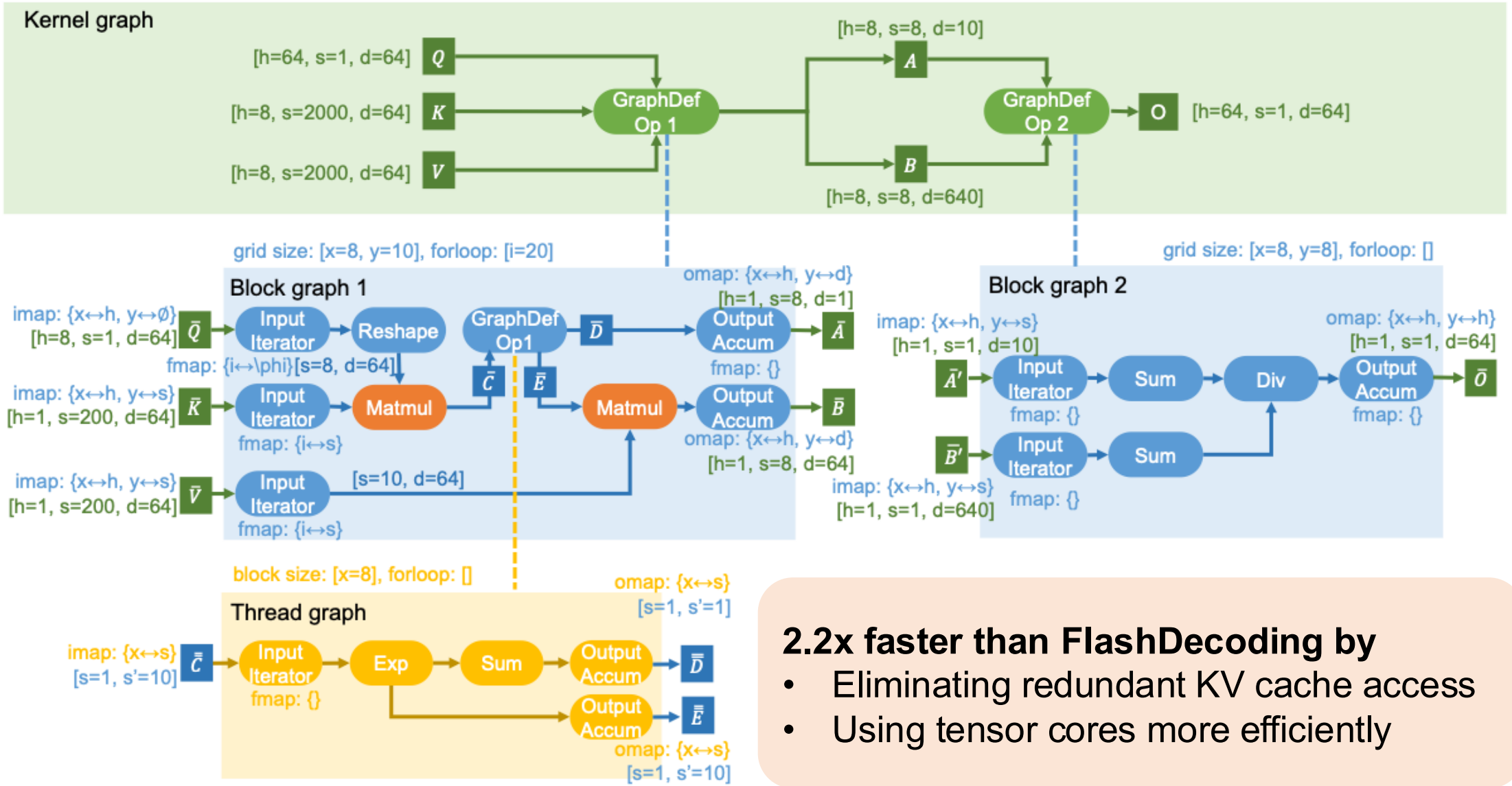
Group-Query Attention using FlashDecoding



Perform attention in two kernels by decomposing softmax into exp, 3 sums, and a div

Best μGraph for GQA

```
A = torch.matmul(Q, K.repeat(8))
S = torch.softmax(A)
O = torch.matmul(S, V.repeat(8))
```



2.2x faster than FlashDecoding by

- Eliminating redundant KV cache access
- Using tensor cores more efficiently

Key Challenges to Discover High-Performance μGraphs

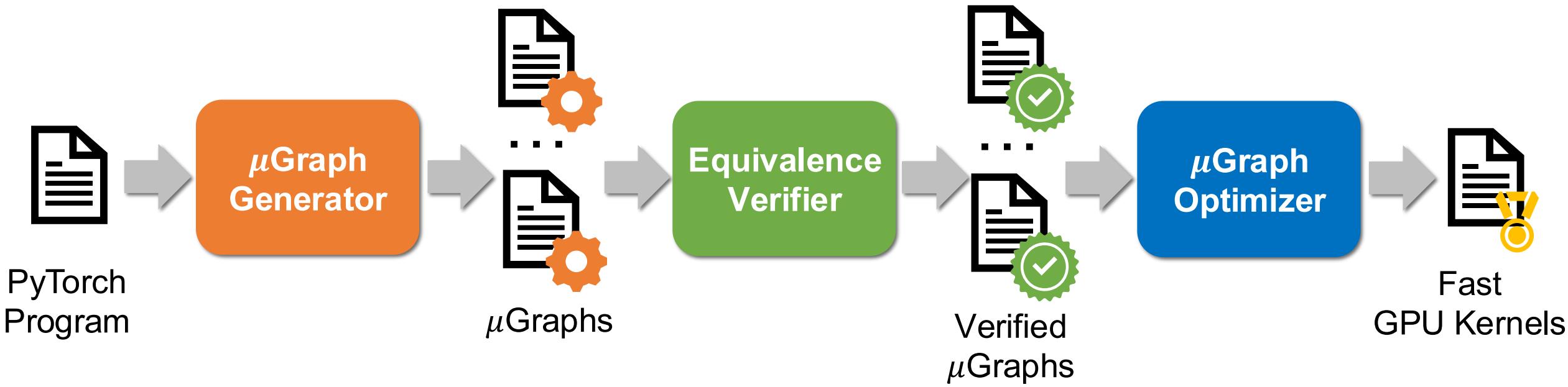
1. How to generate potential μGraphs?

Expression-guided search

2. How to verify their correctness?

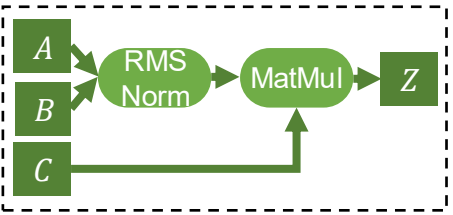
Probabilistic verification

Mirage Overview



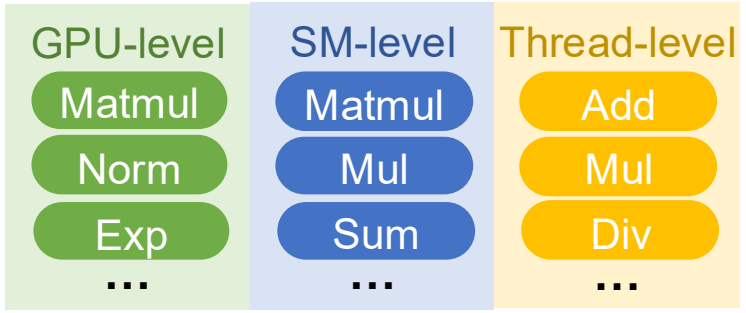
μGraph Generator

Consider all possible μGraphs using available operators

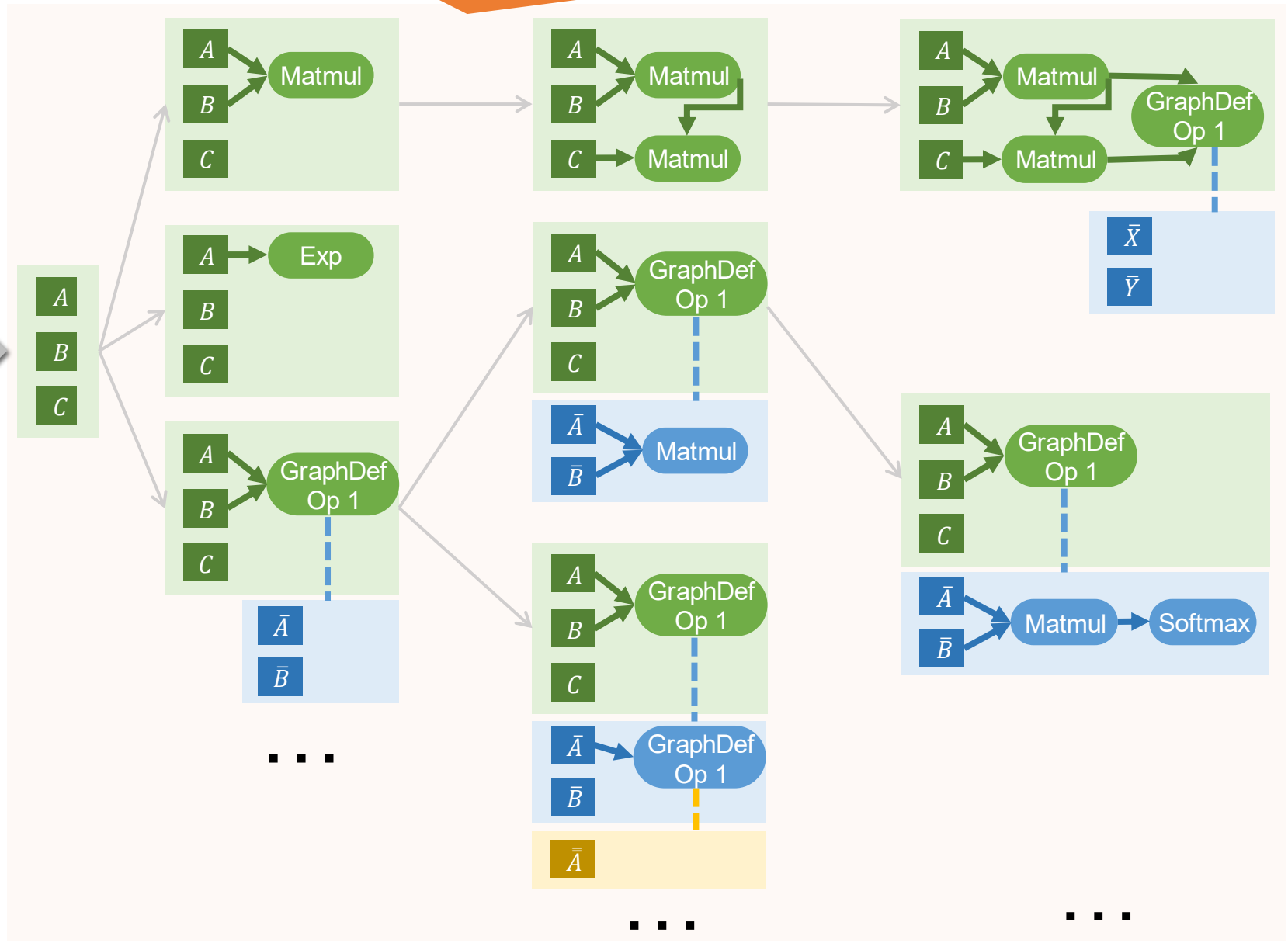


PyTorch Program

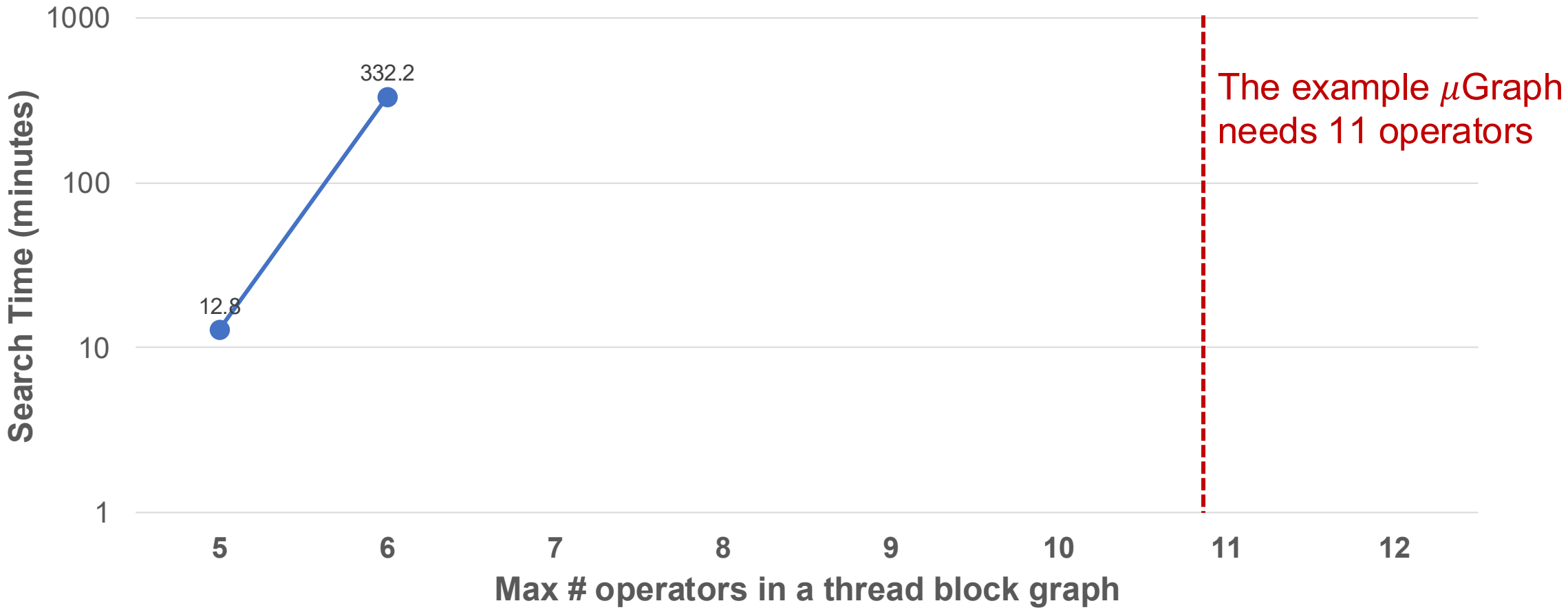
μGraph Generator



Operators at the kernel, thread block, and thread levels

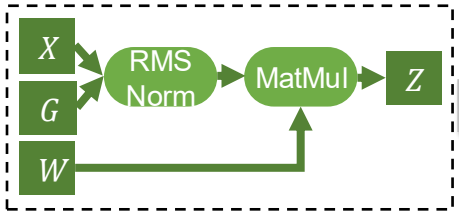


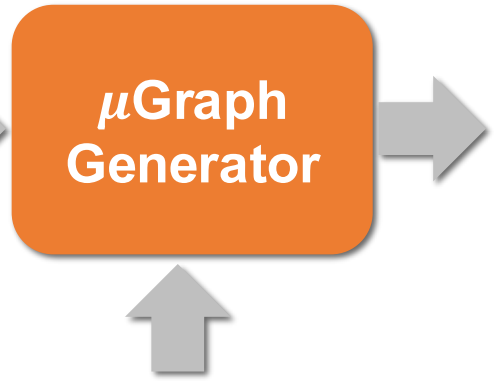
Challenge: Extremely Large Search Space

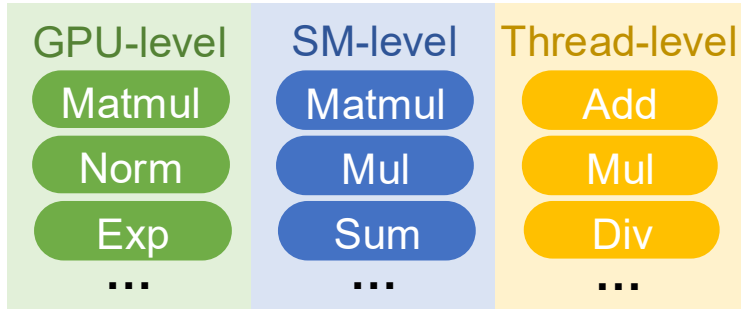


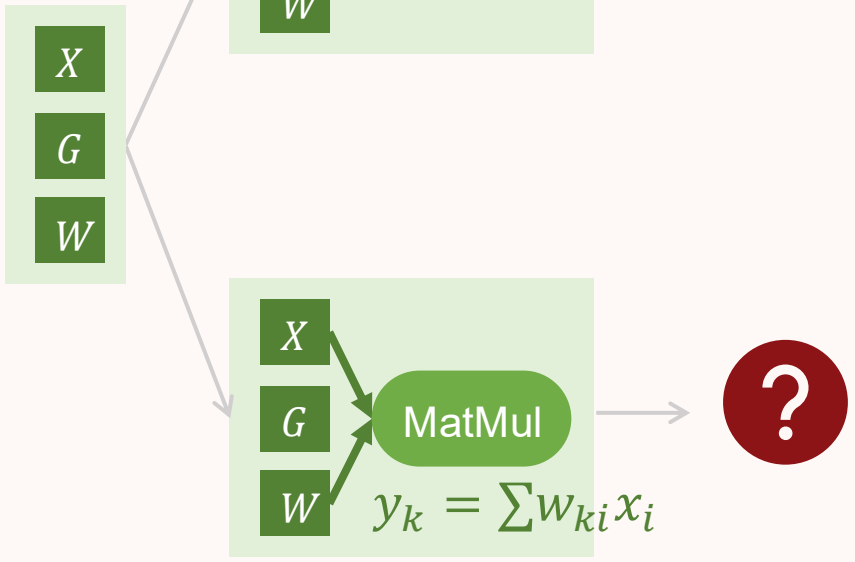
Can Algebraic Properties Guide Search?

$$z_k = \frac{\sum w_{ki} x_i g_i}{\sqrt{\sum x_j^2 / N}}$$





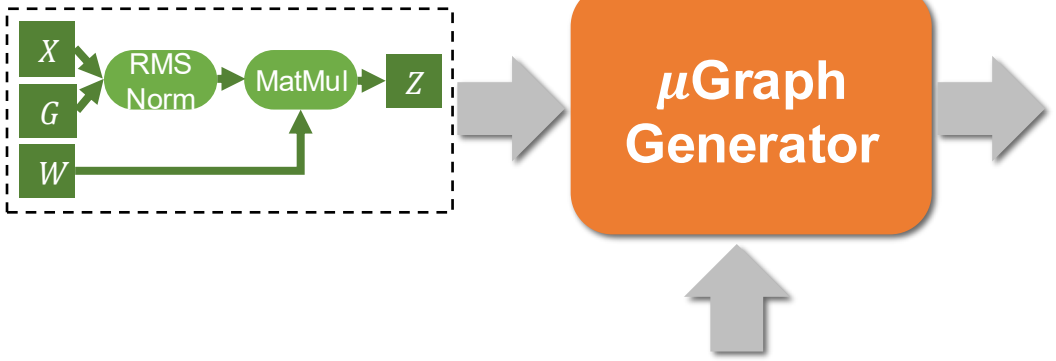




Can Algebraic Properties Guide Search?

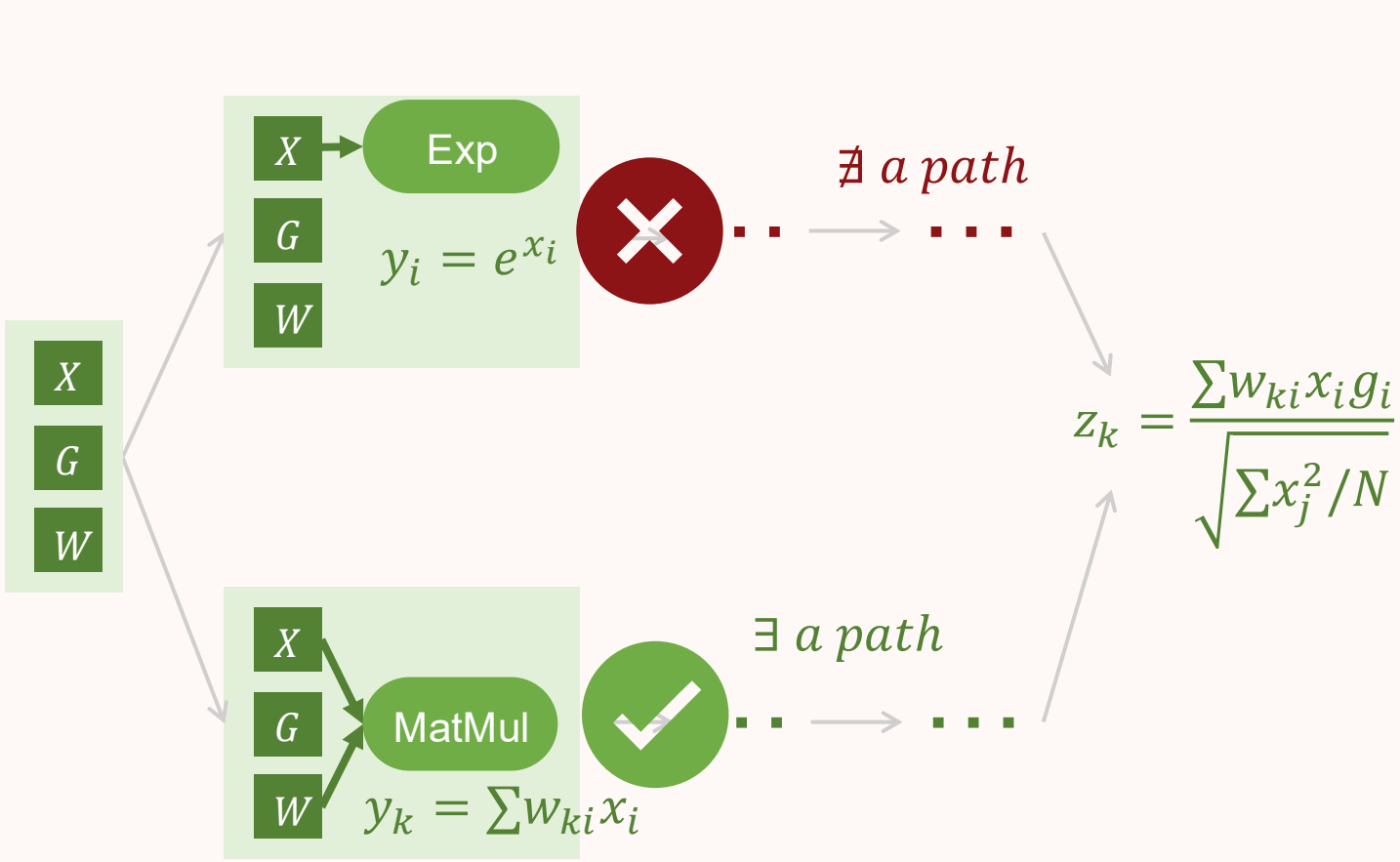
Key idea: prune candidates that do not have a path to desired computation using given operators

$$z_k = \frac{\sum w_{ki} x_i g_i}{\sqrt{\sum x_j^2 / N}}$$



μGraph Generator

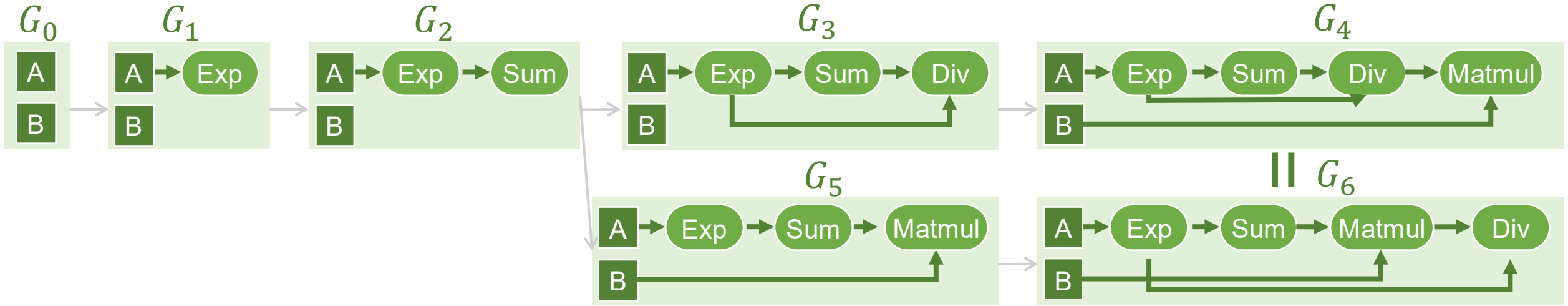
GPU-level	SM-level	Thread-level
Matmul	Matmul	Add
Norm	Mul	Mul
Exp	Sum	Div
...	...	...



Abstract Expression

Goal #1. capture subgraph relations: $E(G_0) \preccurlyeq E(G_1) \preccurlyeq E(G_2) \preccurlyeq E(G_3) \preccurlyeq E(G_4)$

Goal #2. capture graph equivalence: $E(G_4) = E(G_6)$



∃ a path from G to a μGraph G' equivalent to the input G_0?

$$G \longrightarrow \cdots \longrightarrow \cdots \longrightarrow G' \;\|\; G_0$$

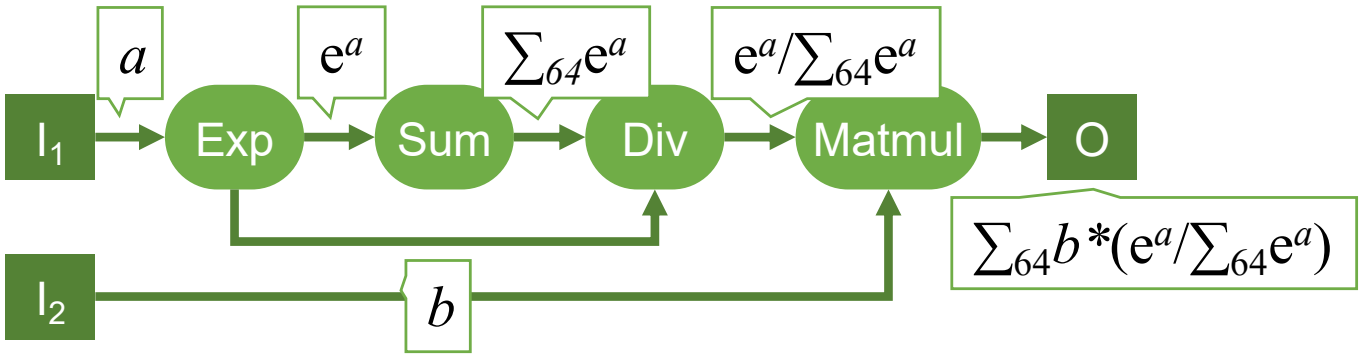
$$E(G) \preccurlyeq E(G_0)\ ?$$

Abstract Expression: An Implementation

Represent a tensor's computation by abstracting away index details

- E.g., $C = A \times B$: $c_{ij} = \sum_{k=1}^{64} a_{ik}\, b_{kj} \Rightarrow$ abstract expression is $c = \sum_{64} ab$

Recursively compute abstract expressions



Abstract Expression

Mirage uses first-order logic to reason about two relations

Goal 1: subexpression

Subexpression Axioms A_{sub}

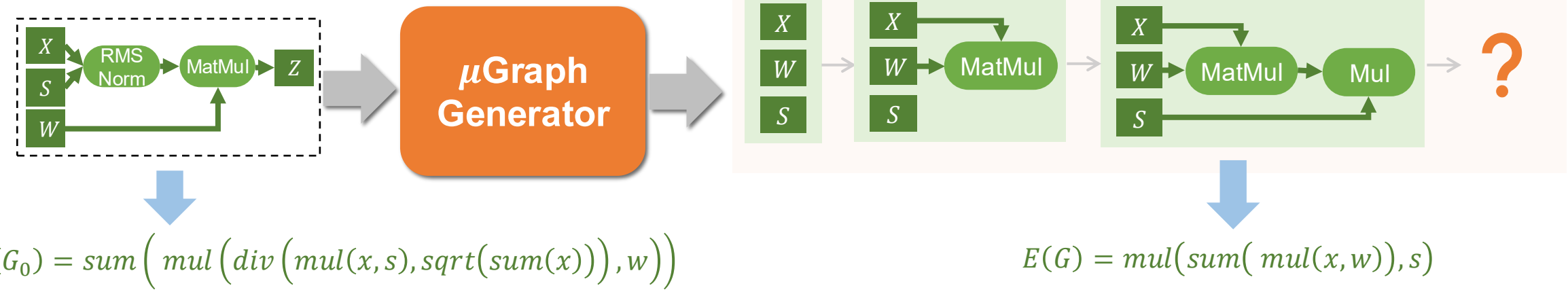
$\forall x, y.\ \text{subexpr}(x, \text{add}(x, y))$	
$\forall x, y.\ \text{subexpr}(x, \text{mul}(x, y))$	
$\forall x, y.\ \text{subexpr}(x, \text{div}(x, y))$	
$\forall x, y.\ \text{subexpr}(y, \text{div}(x, y))$	
$\forall x.\ \text{subexpr}(x, \text{exp}(x))$	
$\forall x, i.\ \text{subexpr}(x, \text{sum}(i, x))$	
$\forall x.\ \text{subexpr}(x, x)$	reflexivity
$\forall x, y, z.\ \text{subexpr}(x, y) \wedge \text{subexpr}(y, z) \rightarrow \text{subexpr}(x, z)$	transitivity

Goal 2: equivalence

Equivalence Axioms A_{eq}

$\forall x, y.\ \text{add}(x, y) = \text{add}(y, x)$	commutativity
$\forall x, y.\ \text{mul}(x, y) = \text{mul}(y, x)$	commutativity
$\forall x, y, z.\ \text{add}(x, \text{add}(y, z)) = \text{add}(\text{add}(x, y), z)$	associativity
$\forall x, y, z.\ \text{mul}(x, \text{mul}(y, z)) = \text{mul}(\text{mul}(x, y), z)$	associativity
$\forall x, y, z.\ \text{add}(\text{mul}(x, z), \text{mul}(y, z)) = \text{mul}(\text{add}(x, y), z)$	distributivity
$\forall x, y, z.\ \text{add}(\text{div}(x, z), \text{div}(y, z)) = \text{div}(\text{add}(x, y), z)$	associativity
$\forall x, y, z.\ \text{mul}(x, \text{div}(y, z)) = \text{div}(\text{mul}(x, y), z)$	associativity
$\forall x, y, z.\ \text{div}(\text{div}(x, y), z) = \text{div}(x, \text{mul}(y, z))$	associativity
$\forall x.\ x = \text{sum}(1, x)$	identity reduction
$\forall x, i, j.\ \text{sum}(i, \text{sum}(j, x)) = \text{sum}(i * j, x)$	associativity
$\forall x, y, i.\ \text{sum}(i, \text{add}(x, y)) = \text{add}(\text{sum}(i, x), \text{sum}(i, y))$	associativity
$\forall x, y, i.\ \text{sum}(i, \text{mul}(x, y)) = \text{mul}(\text{sum}(i, x), y)$	distributivity
$\forall x, y, i.\ \text{sum}(i, \text{div}(x, y)) = \text{div}(\text{sum}(i, x), y)$	distributivity

Abstract Expression-Guided Search



$$E(G_0) = sum\Big(\, mul\Big(div\Big(mul(x,s), sqrt\big(sum(x)\big)\Big), w\Big)\Big)$$

$$E(G) = mul\big(sum(\, mul(x,w)), s\big)$$

A1. $\forall x, y.\, x \preccurlyeq mul(x,y)$
A2. $\forall x, i.\, x \preccurlyeq sum(i,x)$
A3. …

Subexpression axioms

E1. $\forall x, y, z.$
$mul\big(x, div(y,z)\big) = div(mul(x,y), z)$
E2. …

Equivalence axioms

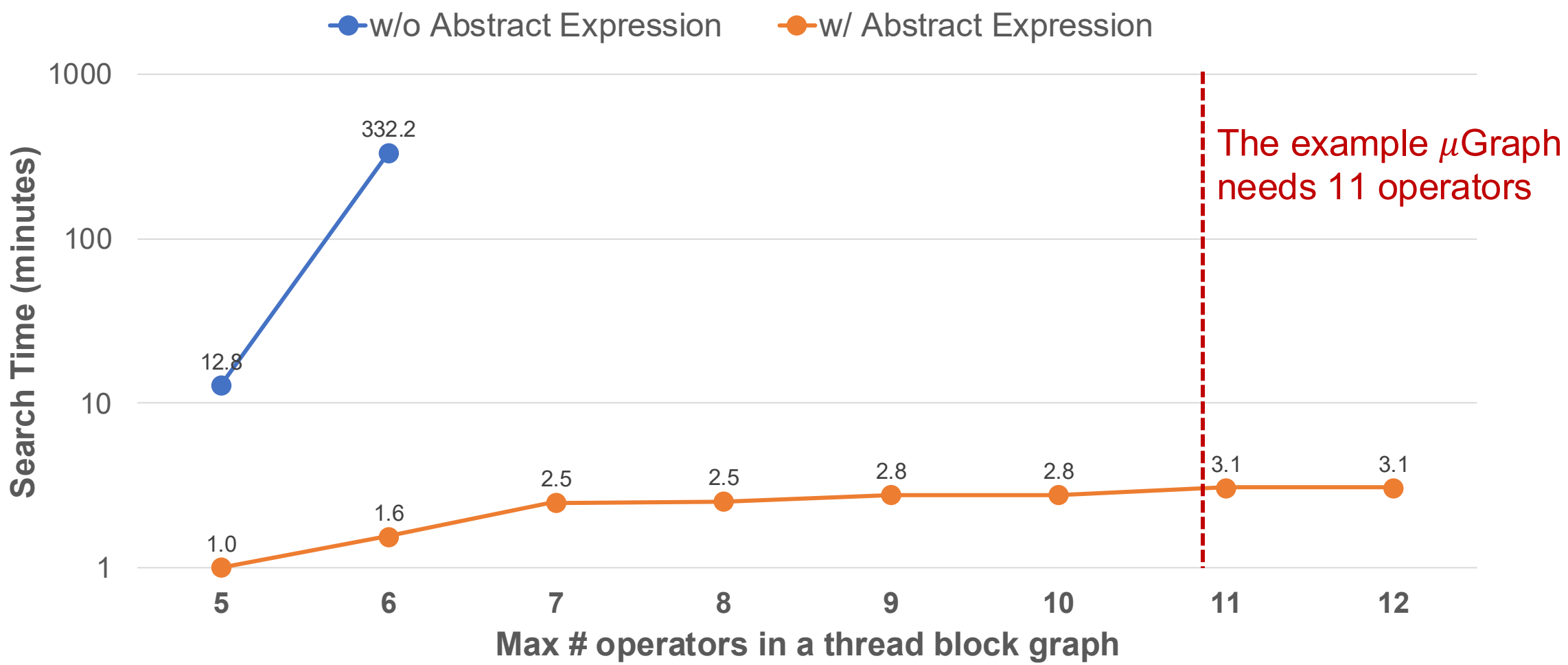
$E(G) \preccurlyeq E(G_0)$?

Automated Theorem Prover

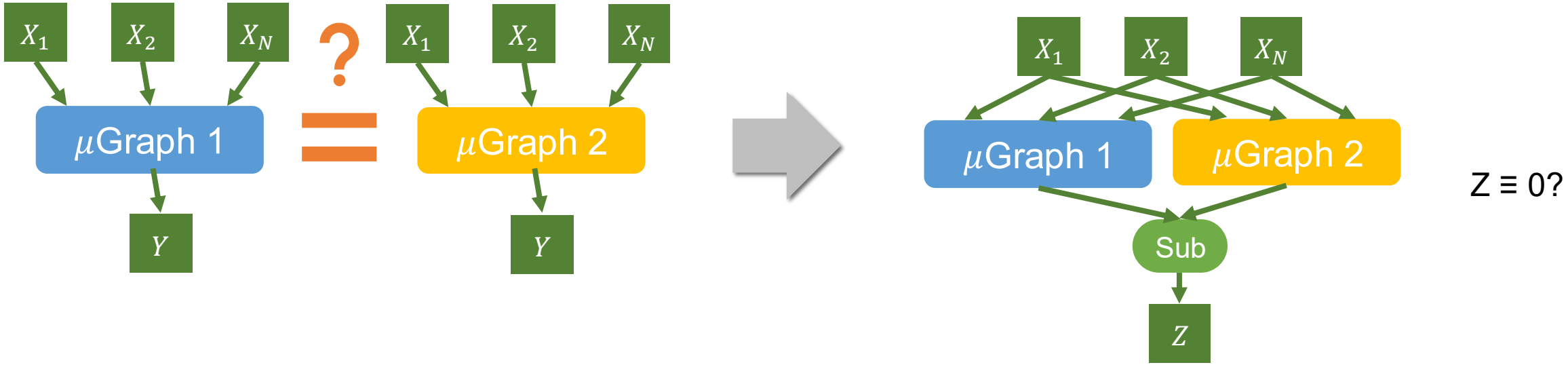
continue the search

prune candidate μGraph

Abstract Expression Significantly Improves Scalability



How to Verify Equivalence between μGraphs?

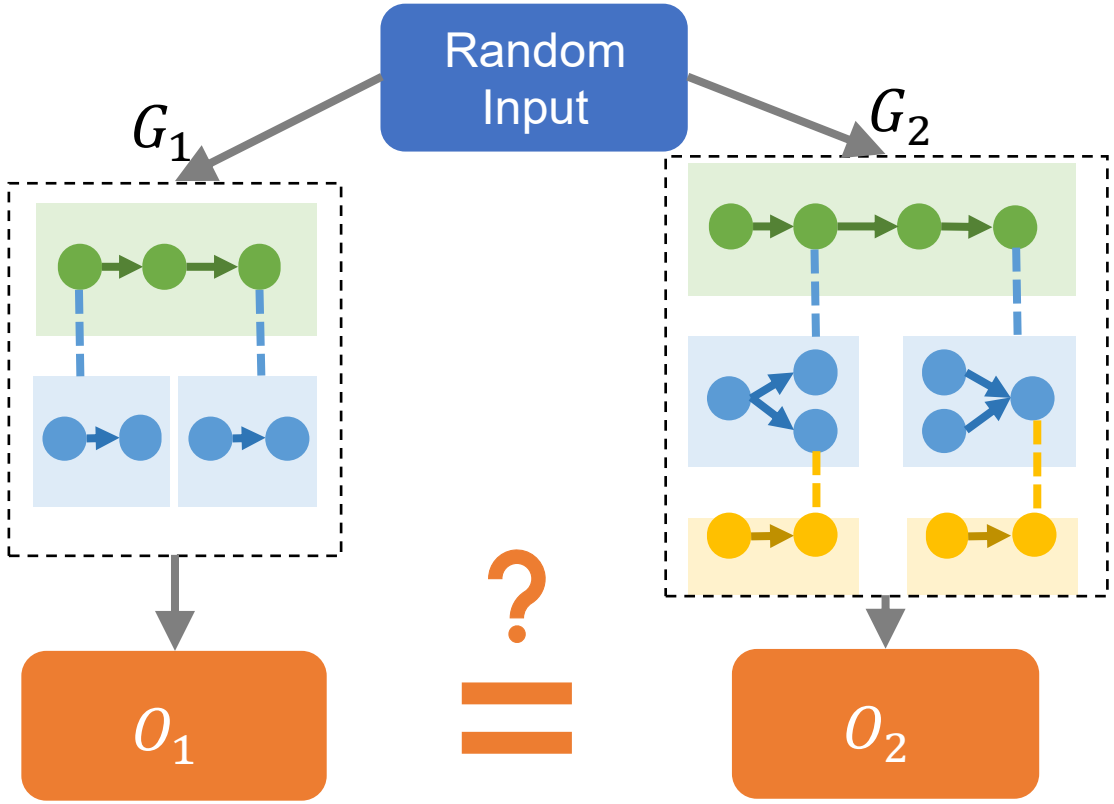


Schwartz–Zippel lemma: Let $P \in F[X_1, X_2, .., X_n]$ be a non-zero polynomial of total degree d over a finite field F. Let $r_1, r_2, .., r_n$ be selected randomly and uniformly from F. Then

$$\Pr[P(r_1, r_2, \ldots, r_n) = 0] \leq \frac{d}{|F|}$$

Probabilistic Equivalence Verifier

Idea: use random inputs in ***finite fields*** to examine μGraph equivalence



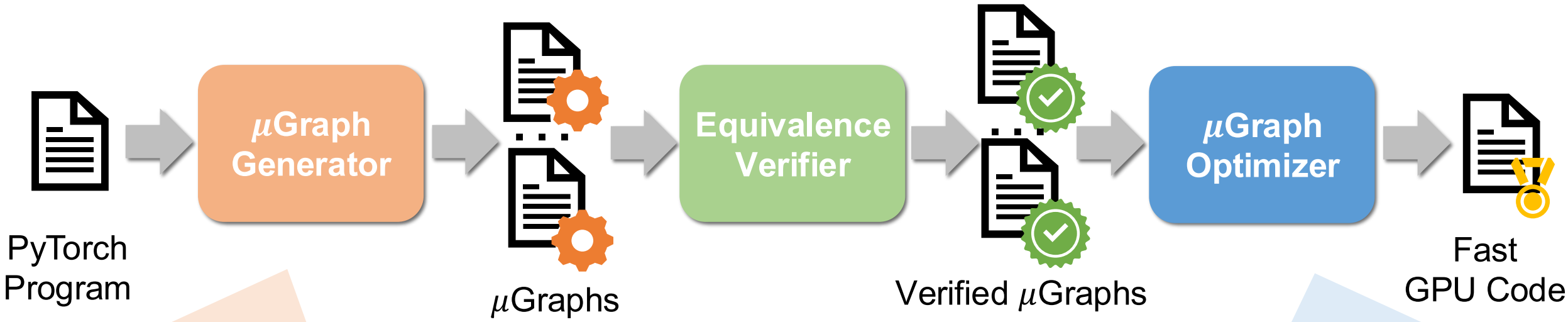
Theorem 1: if G_1 is equivalent to G_2, then $O_1 = O_2$

Theorem 2: if G_1 is not equivalent to G_2, then $O_1 \neq O_2$ with a certain probability p^*

Run t random tests, non-equivalent μGraphs pass all random tests with probability $(1-p)^t$

* $p \geq \frac{1}{T}$, where T is the size of intermediate tensors

μGraph Optimizer



Only consider output-alternating optimizations:

- Algebraic transformations
- Customized kernels
- Compute organization

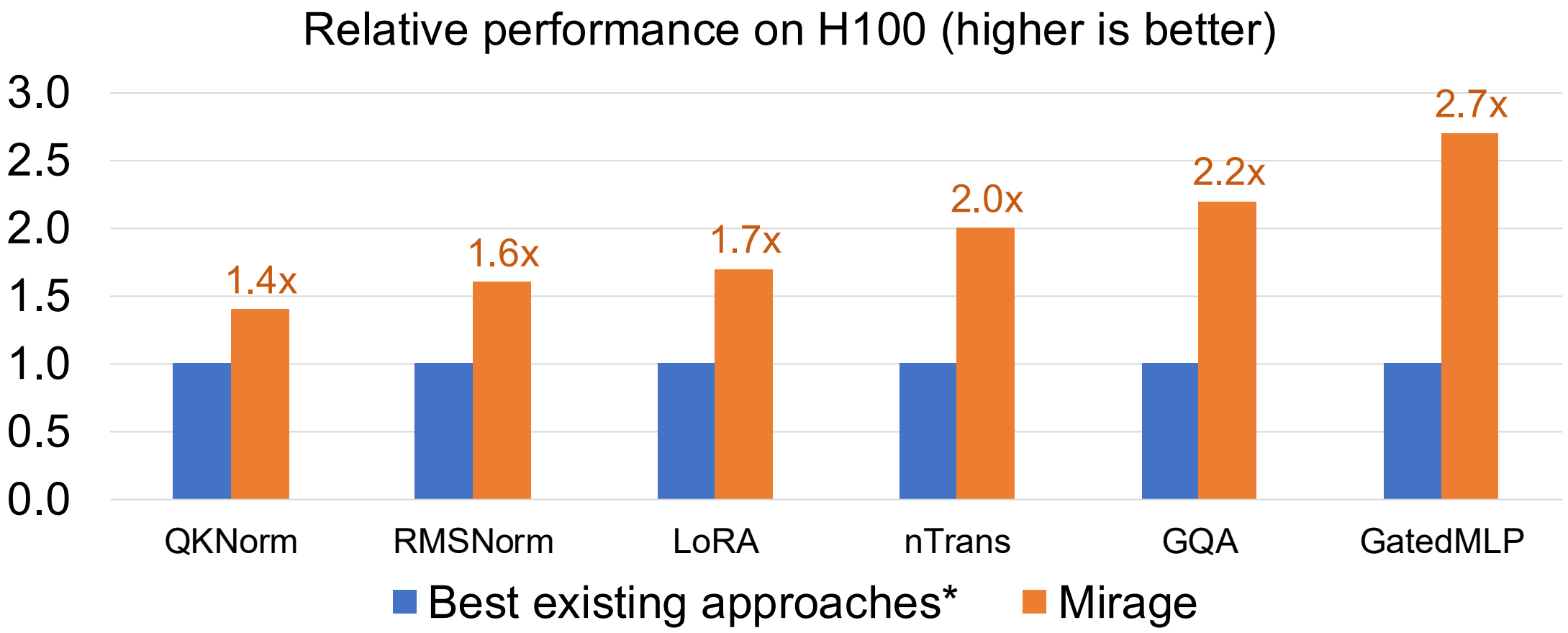
Reduce generator's search space

Other optimizations are deferred to μGraph Optimizer:

- Tensor layouts
- Memory planning
- Operator scheduling

Solve these tasks optimally

Mirage Outperforms Existing Approaches

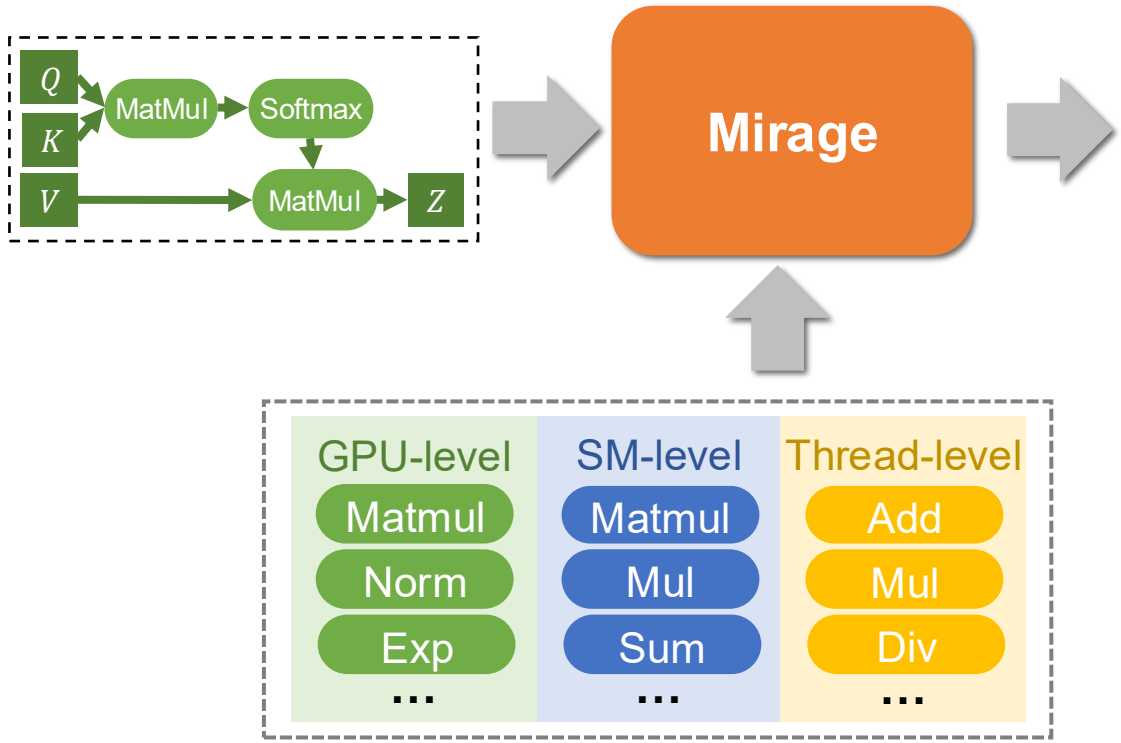


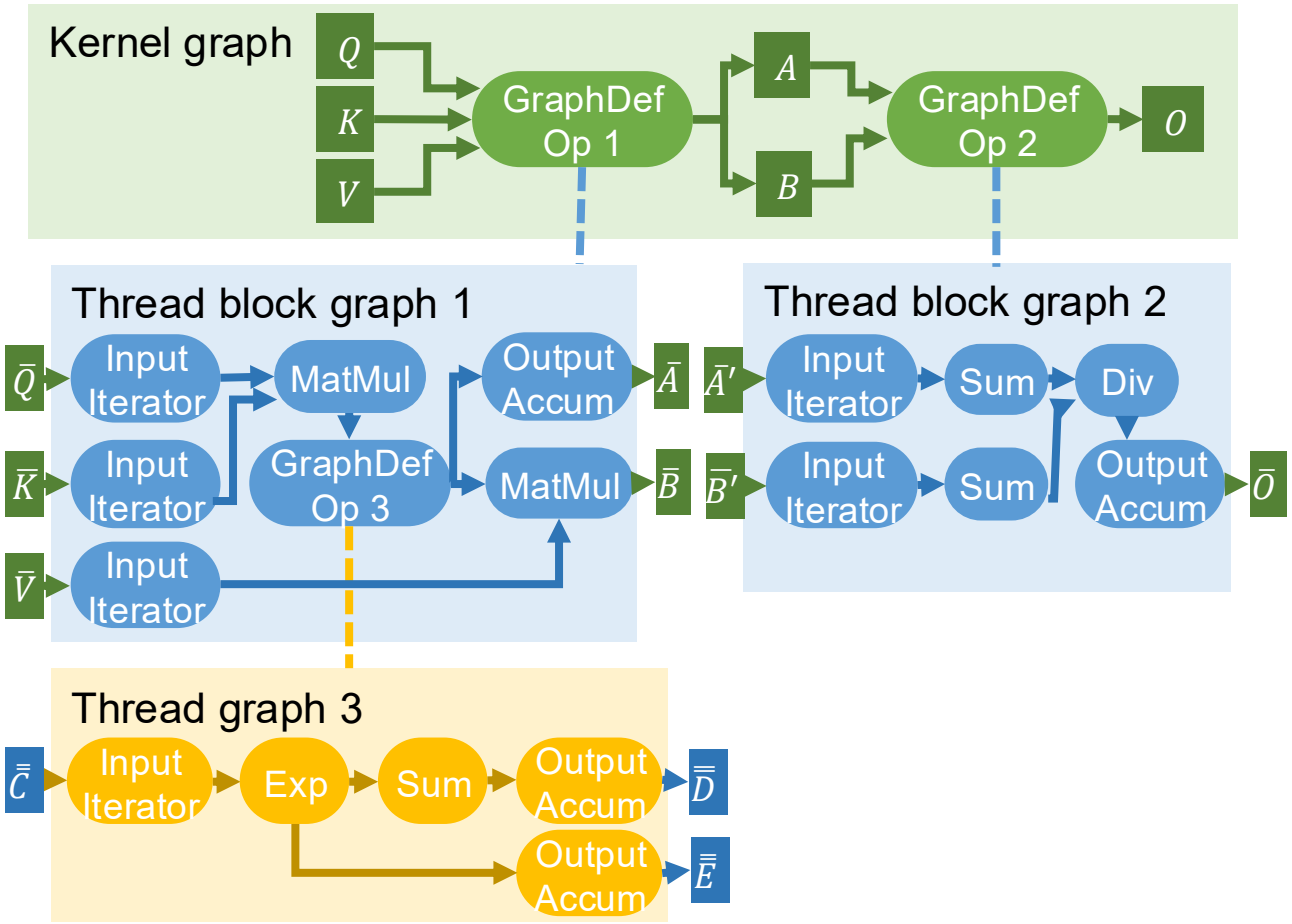
Vender-provided | Expert-written | Compiler-generated

*Best existing approaches are the best of cuDNN/cuBLAS, FlashAttention, PyTorch, TensorRT, Triton, and TVM.

Mirage Discovers Hardware-Customized μGraphs

Find μGraphs similar to expert-written implementations for attention on A100

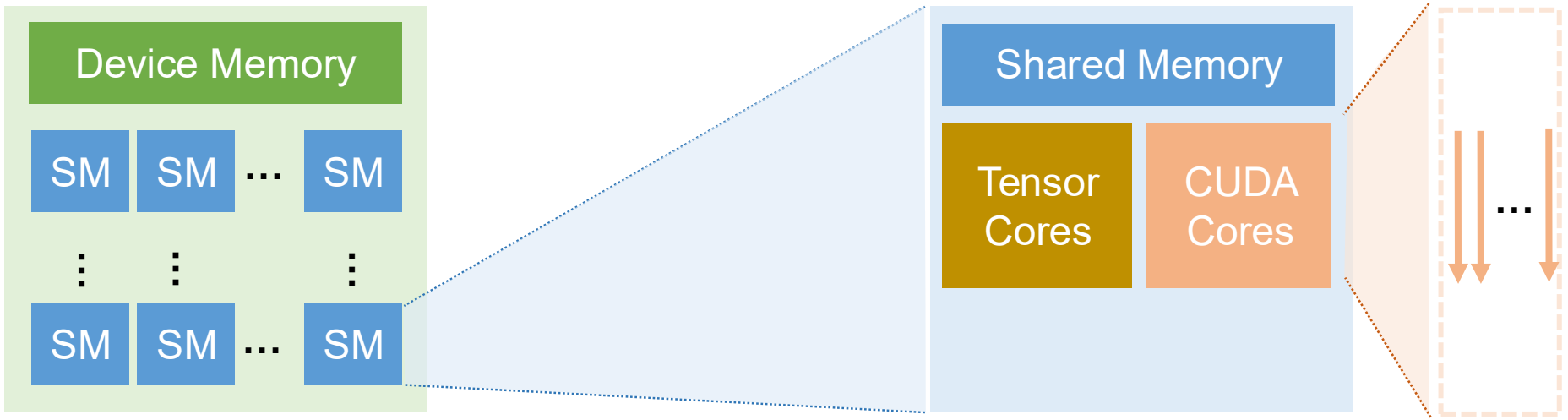




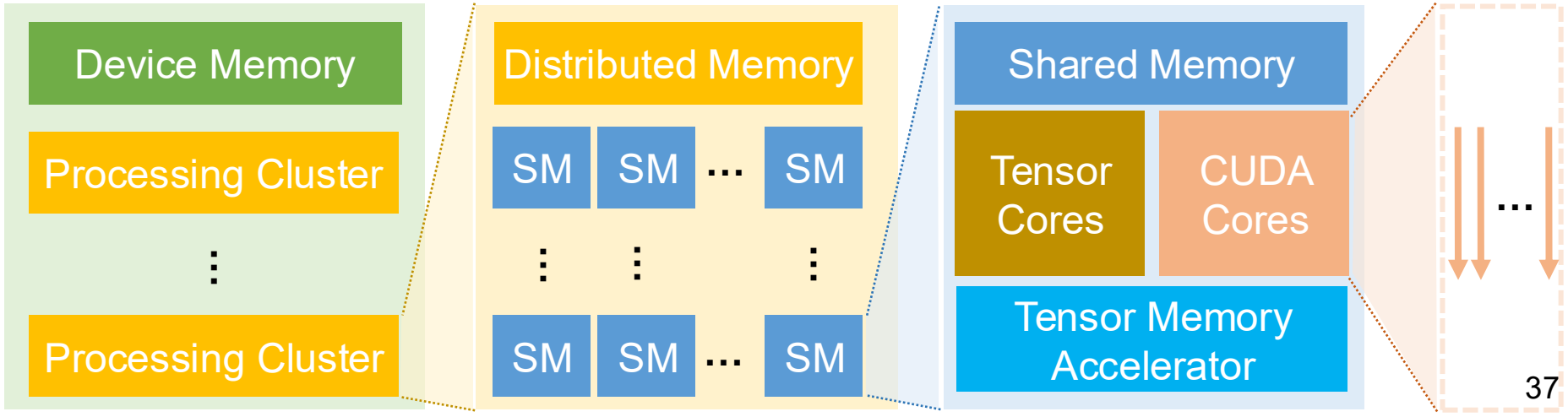
* FlashDecoding++: Faster LLM Inference with Asynchronization, Flat GEMM Optimization, and Heuristics. MLSys'24

H100: GPU Processing Clusters

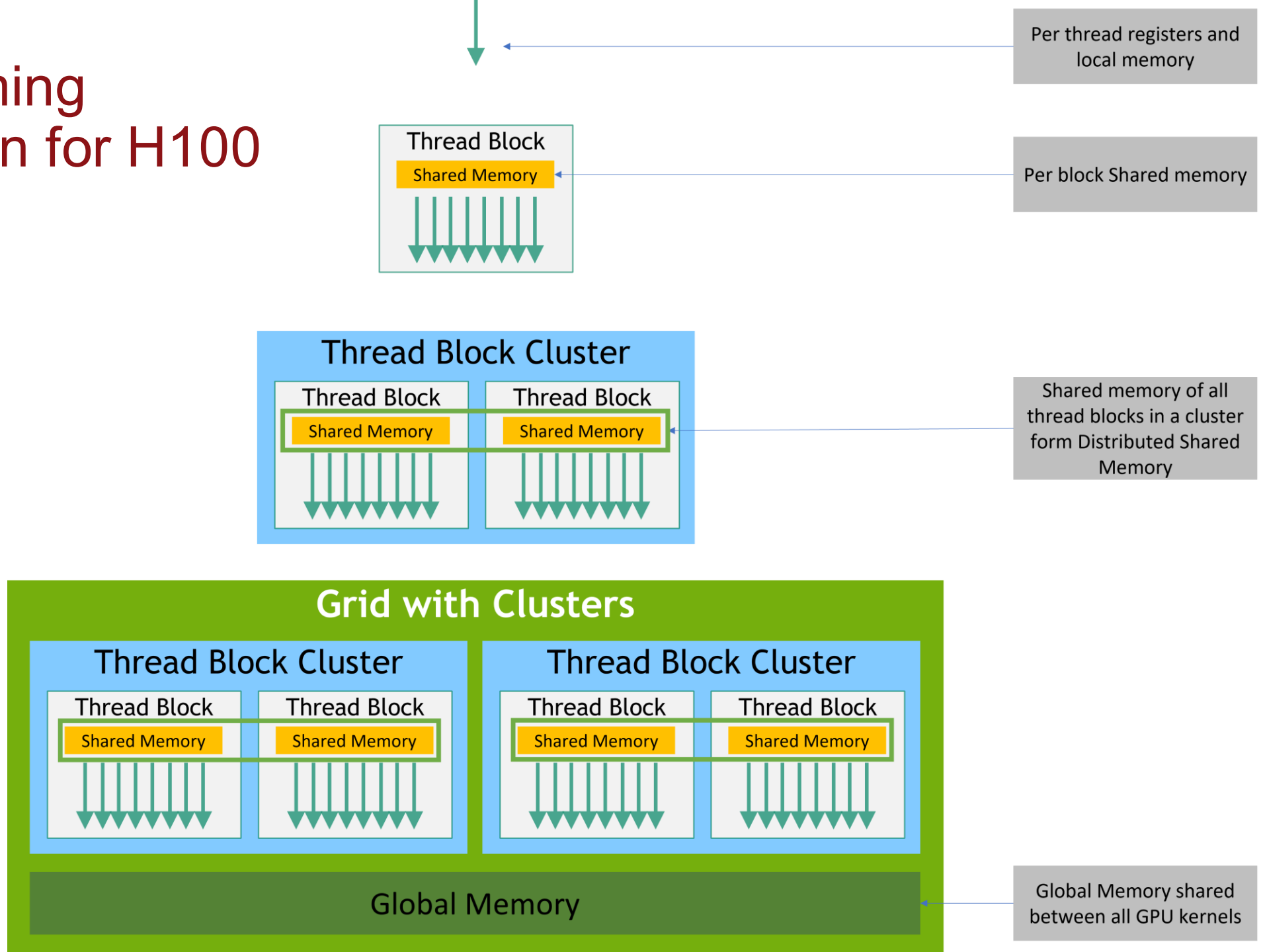
NVIDIA A100 GPU (2020)



NVIDIA H100 GPU (2022)



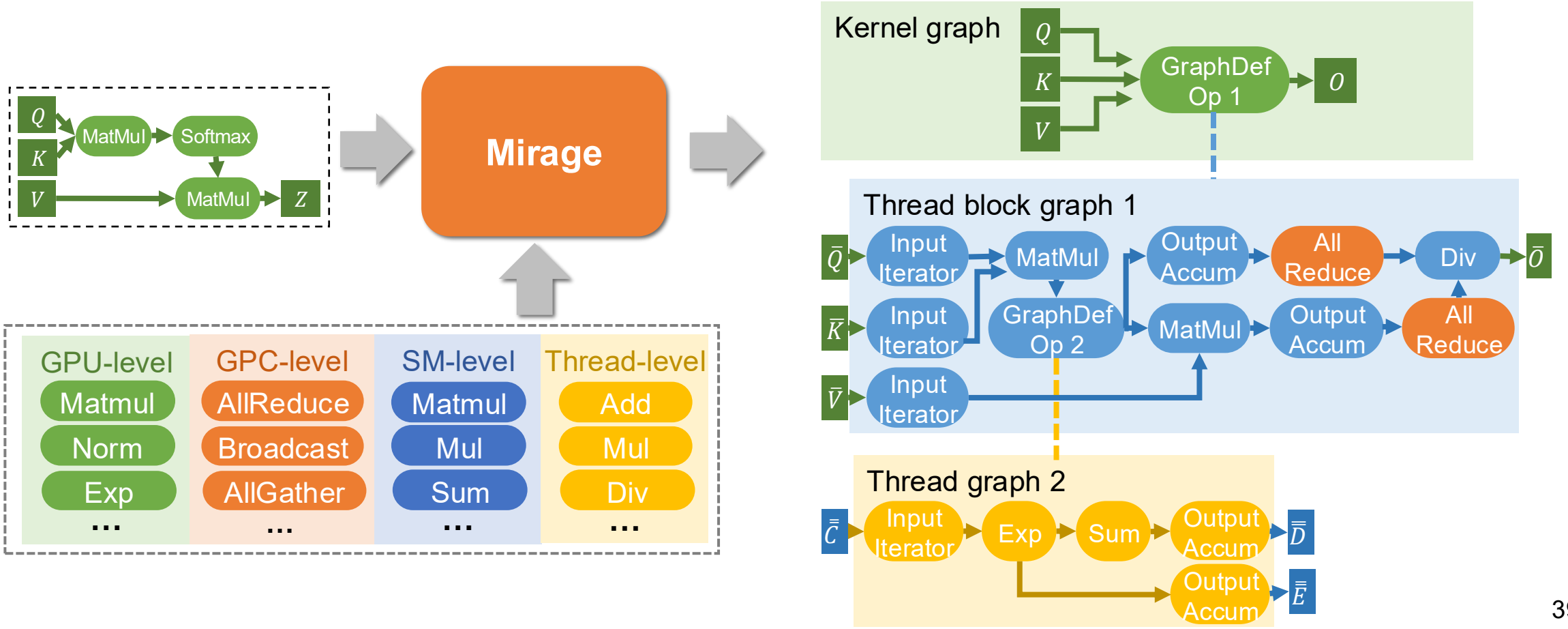
Programming Abstraction for H100



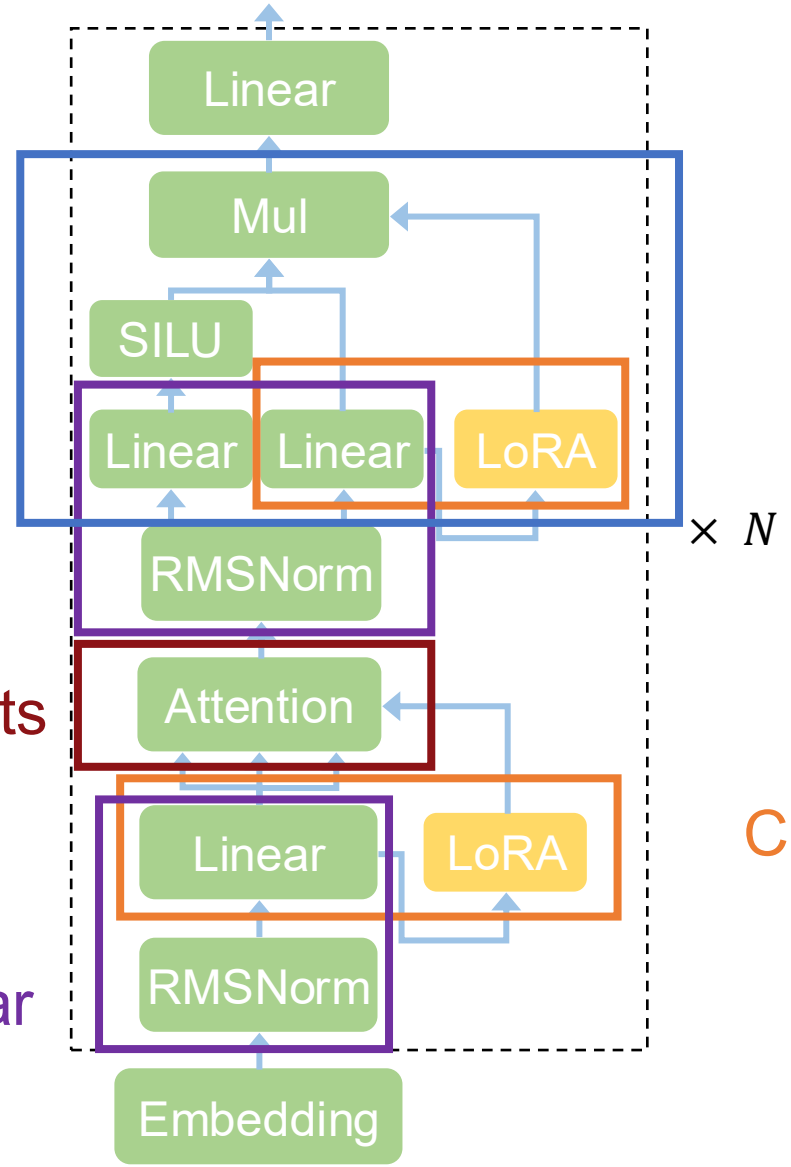
Mirage Discovers Hardware-Customized μGraphs

Leverage GPC-level AllReduce to accelerate attention on H100

- 2.2x faster than best existing kernels



Ubiquitous Superoptimization Opportunities in DNNs



Case 3: Gated MLP

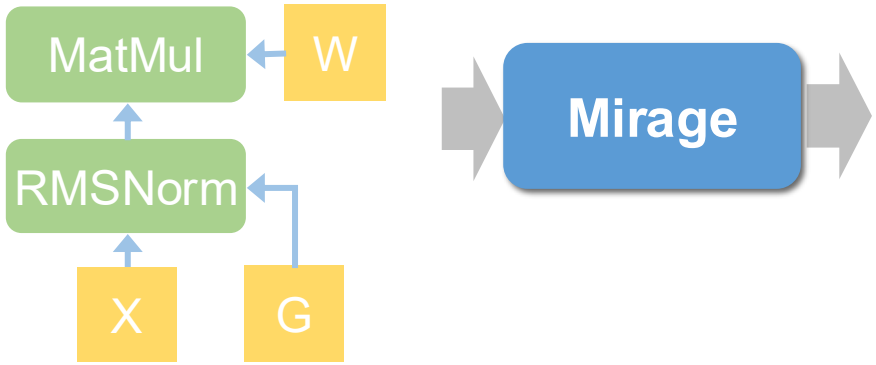
Case 4: Attention Variants

Case 2: LoRA + Linear

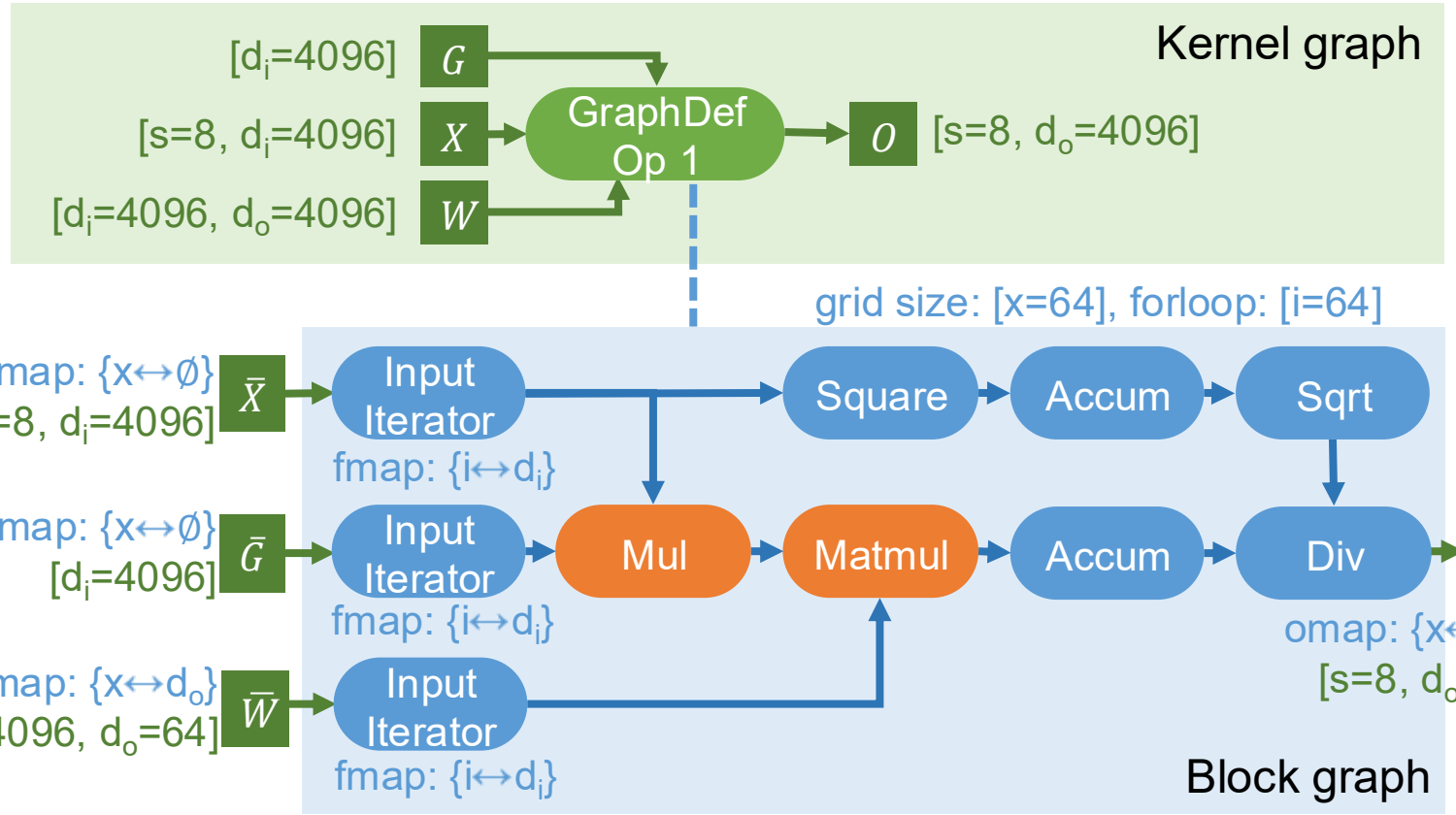
Case 1: Norm + Linear

Case 1: RMSNorm + Linear

- Key idea: move the denominator of RMSNorm after matrix multiplication to fuse kernels and reuse shared memory



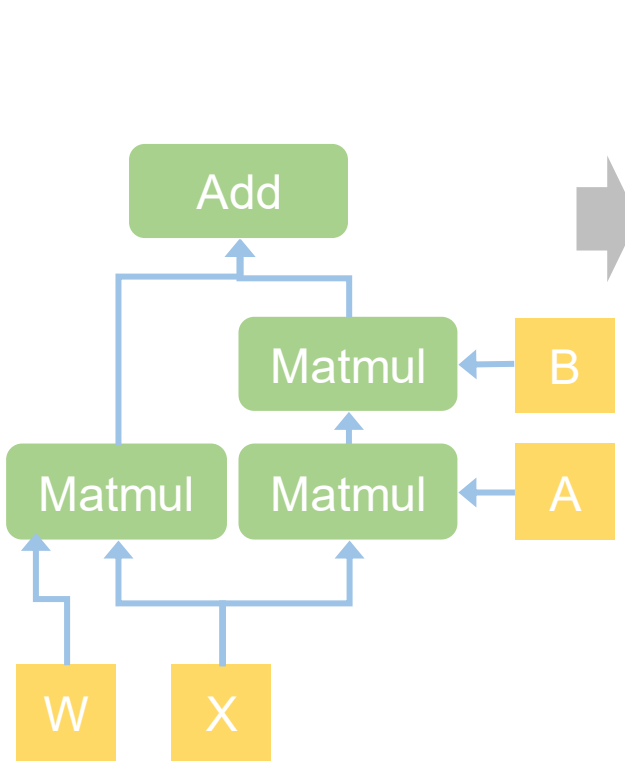
Mirage

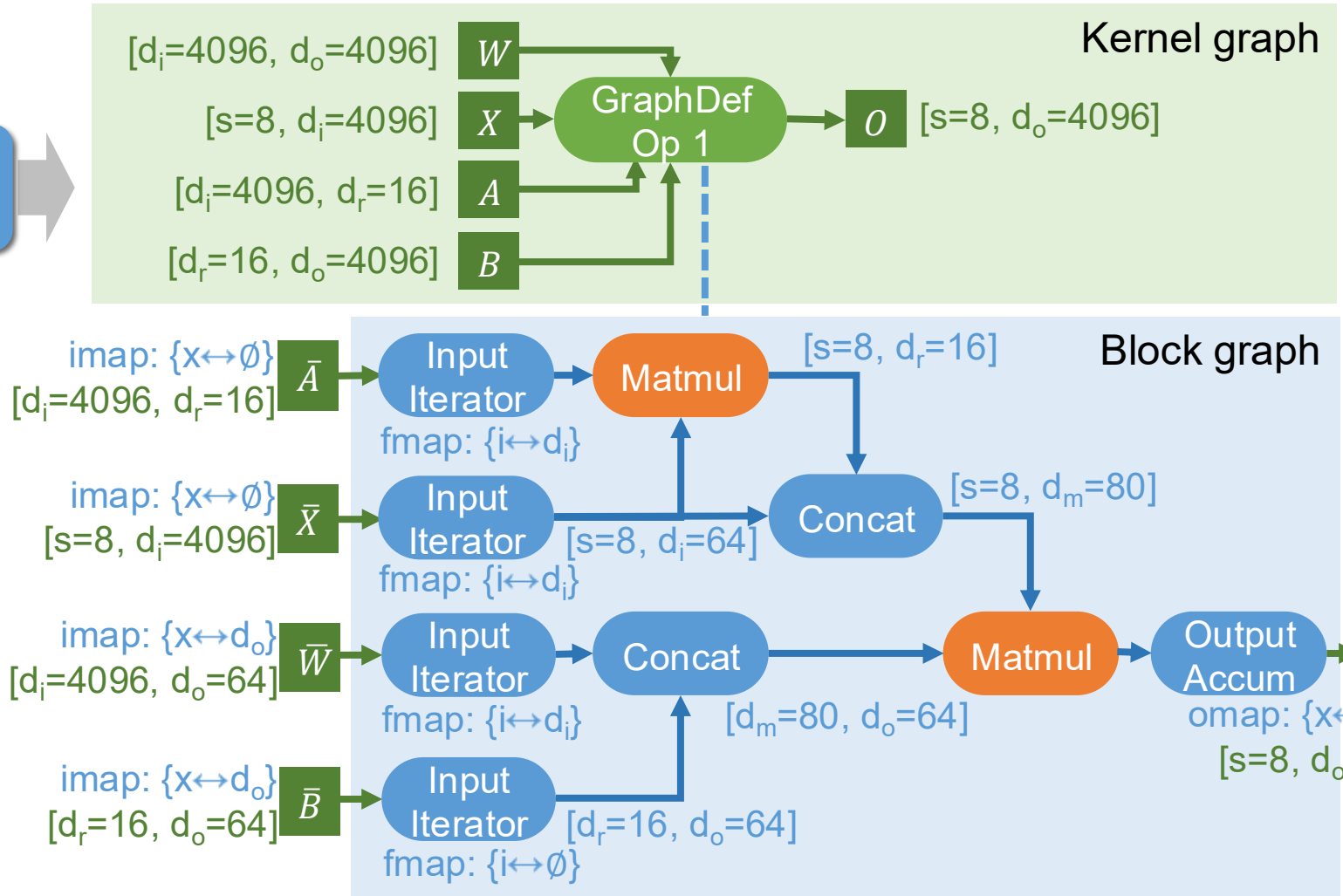


$$RMSNorm(x_i) = \frac{x_i * g_i}{\sqrt{\frac{1}{N}\sum_j x_j^2}}$$

Case 2: Optimized μGraph for LoRA (2.3x faster than Triton)

- Fuse three matmuls and an addition into a single kernel





Case 3: Gated MLP