Superoptimizing Machine Learning Systems

Zhihao Jia

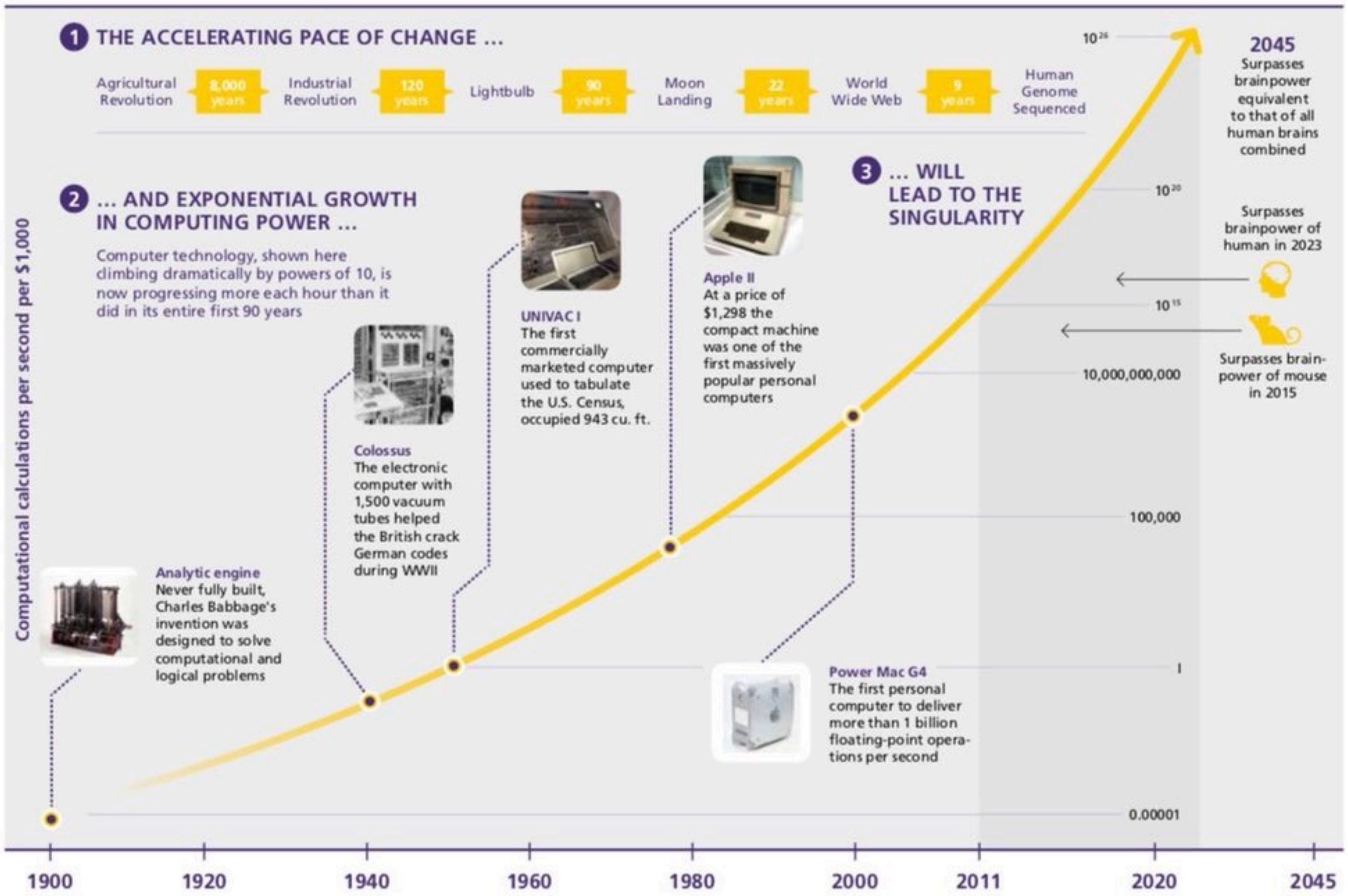
Computer Science Department
Carnegie Mellon University

catalyst

https://catalyst.cs.cmu.edu/

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

Compute Per Second Per Dollar

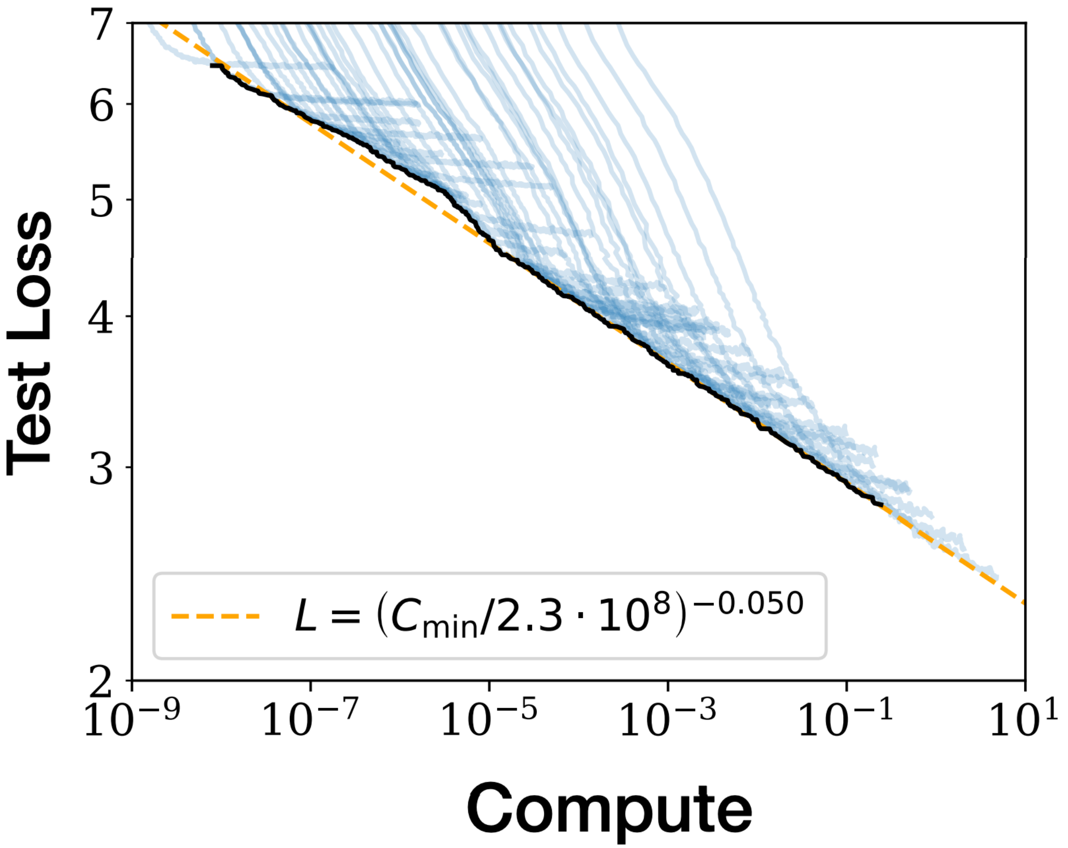


Surpass human brainpower in 2023

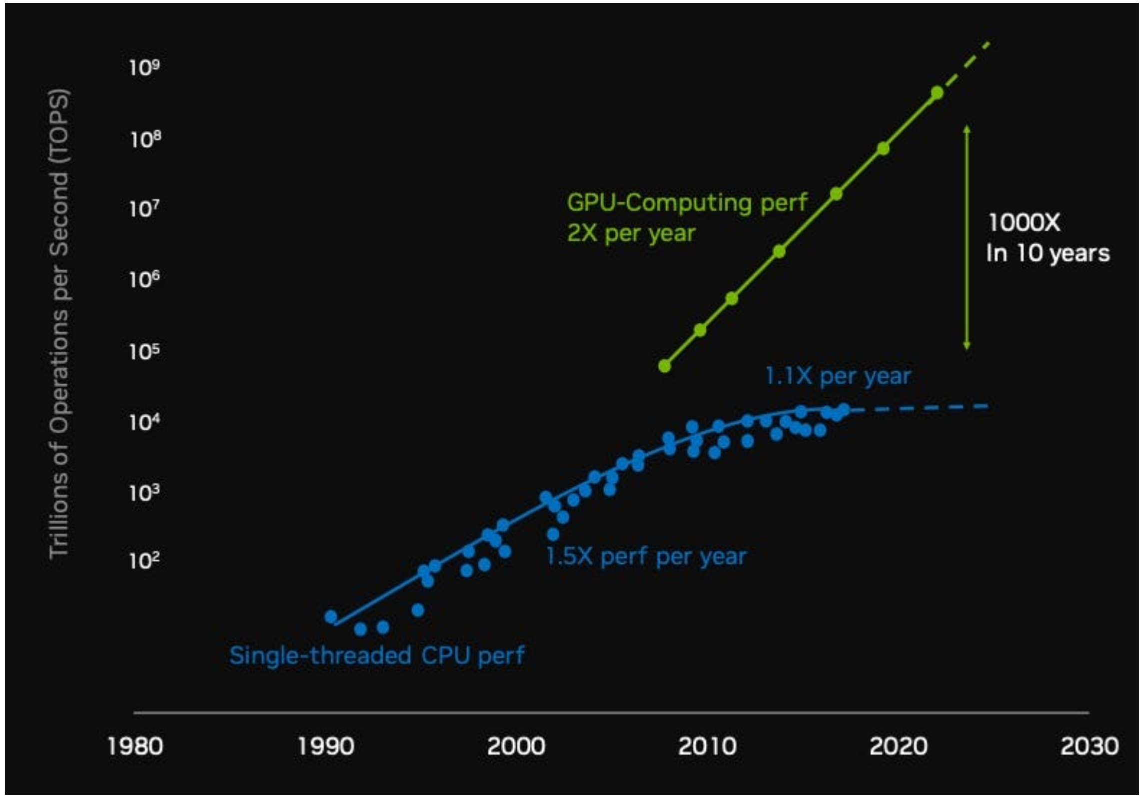
* Ray Kurzweil. *The Singularity Is Near: When Humans Transcend Biology.* 2005

Scaling Law in ML

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

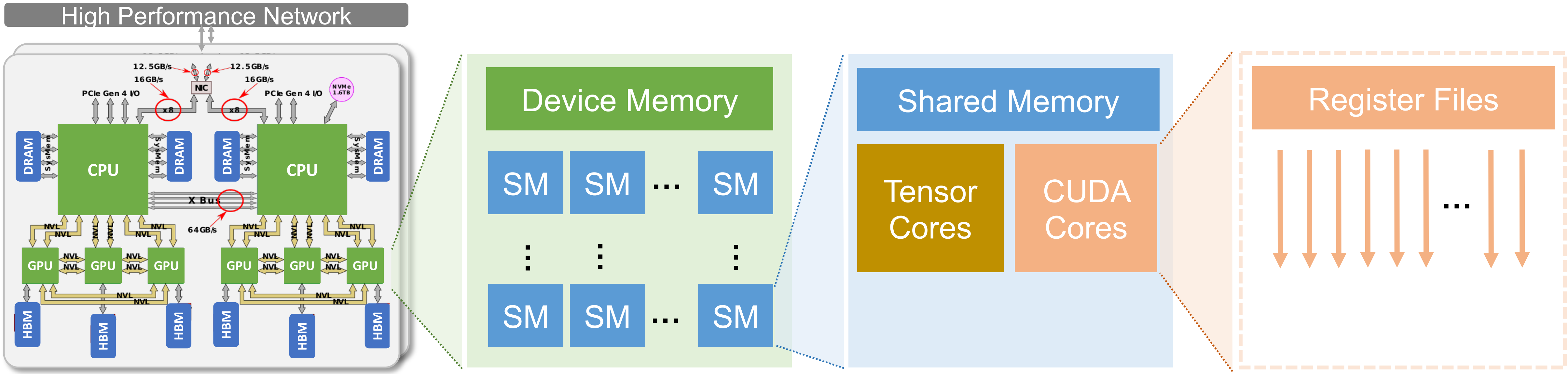


Source: OpenAI



Source: NVIDIA

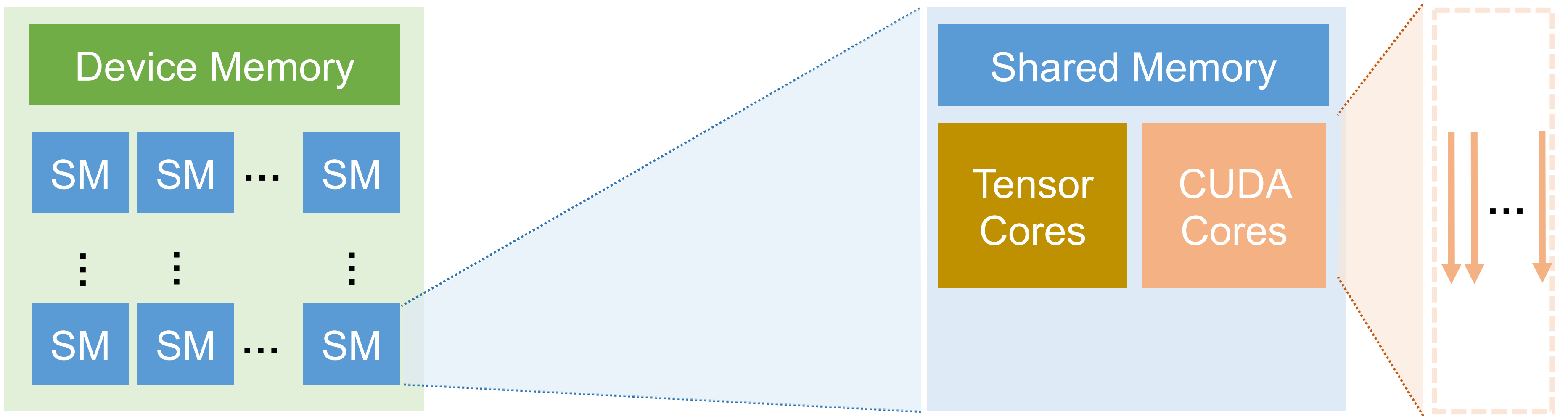
ML Hardware is Massively Parallel, Highly Heterogeneous



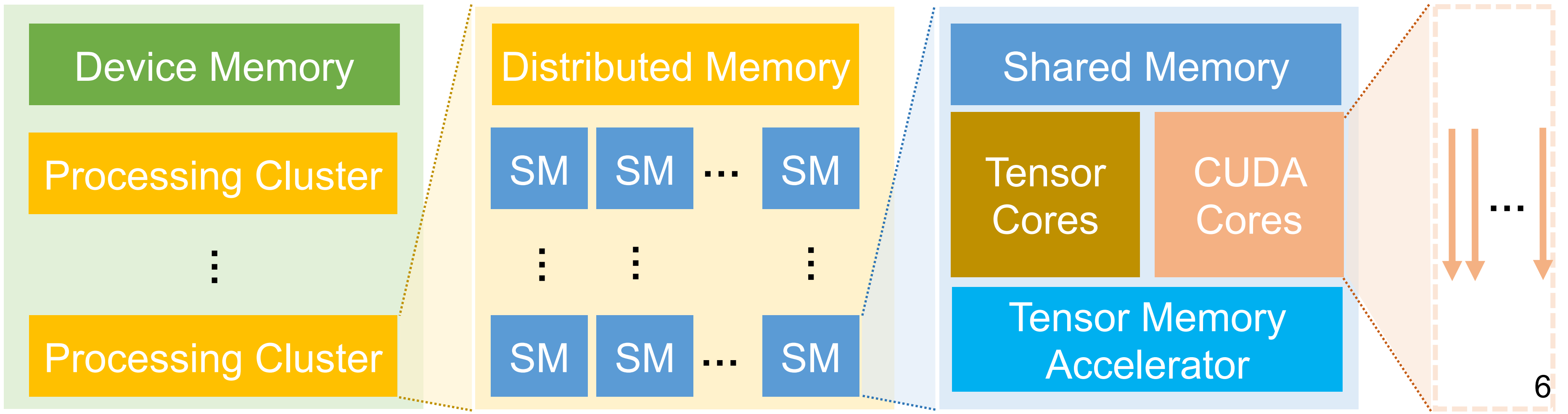
1K-100K GPUs × 100s streaming multiprocessors (SMs) × 128 CUDA cores 4 tensor cores = 10M-1B CUDA cores 400K-40M tensor cores

ML Hardware is Quickly Evolving

NVIDIA A100 GPU (2020)



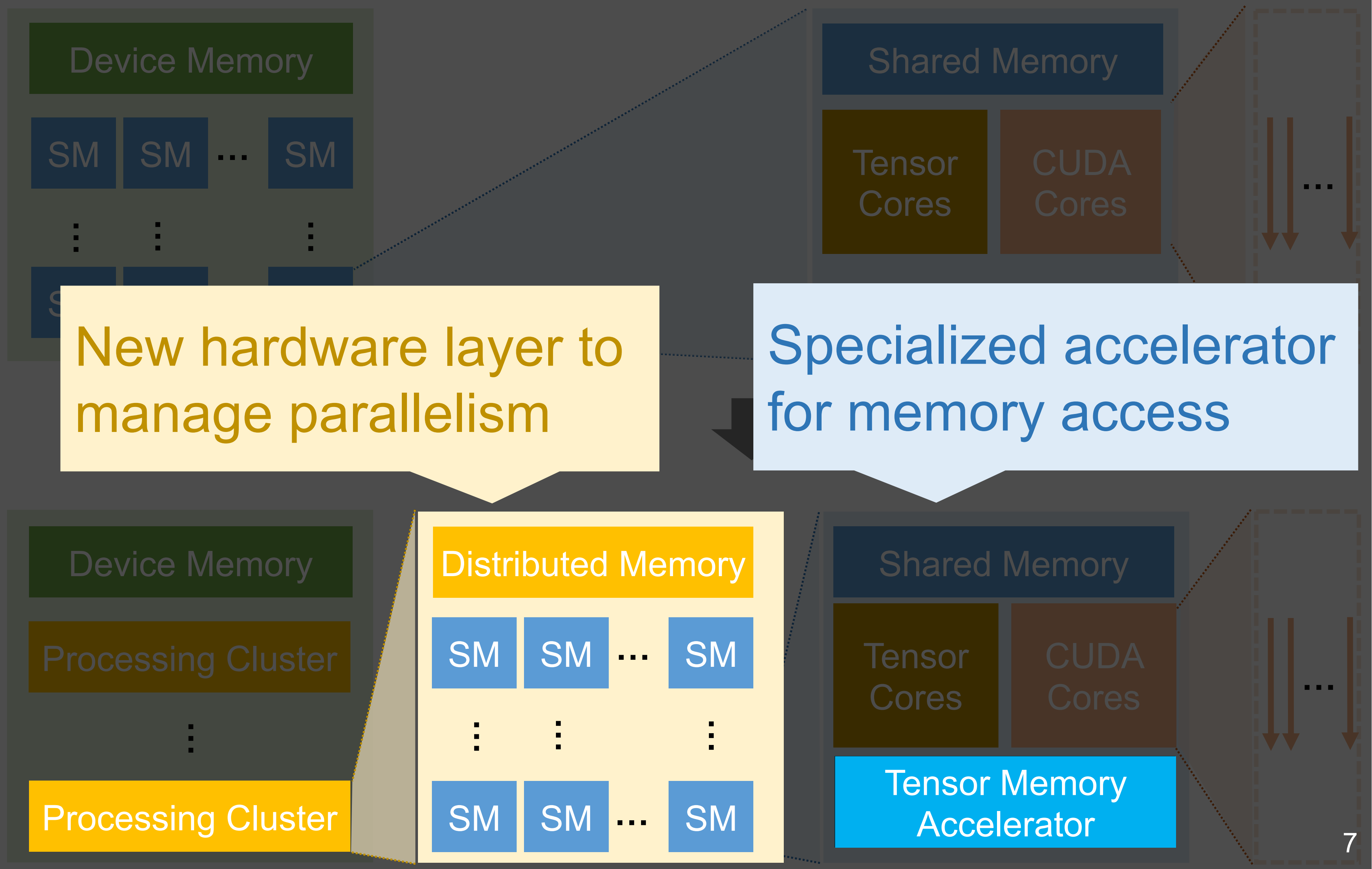
NVIDIA H100 GPU (2022)



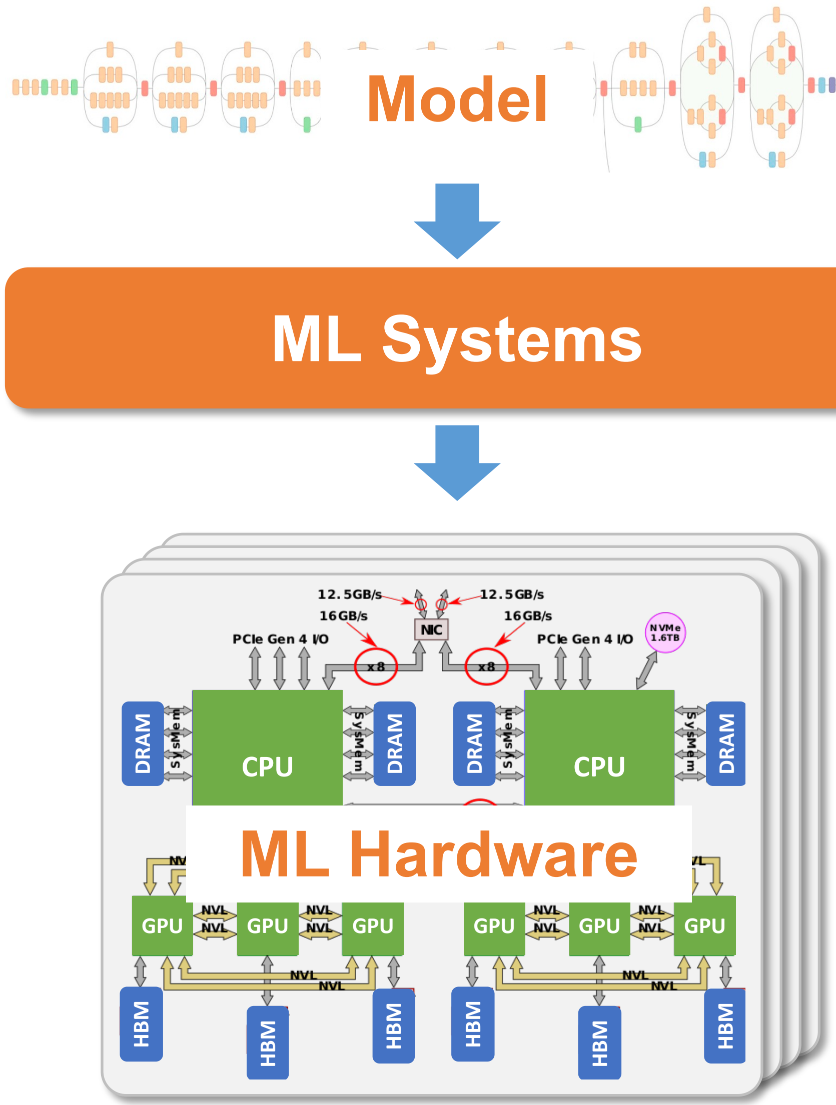
ML Hardware is Quickly Evolving

NVIDIA A100 GPU (2020)

NVIDIA H100 GPU (2022)



Our Research: ML Systems



Goal:
Efficiently deploying ML applications on <u>massively parallel</u>, <u>increasingly heterogeneous</u>, <u>rapidly evolving</u> hardware platforms

Key Challenges for Developing ML Systems

Massively Parallel

Billions of compute units on modern ML hardware

How can we find the best way to parallelize ML computation?

Increasingly Heterogeneous

CUDA cores, tensor cores, tensor memory accelerators, processing clusters, and more

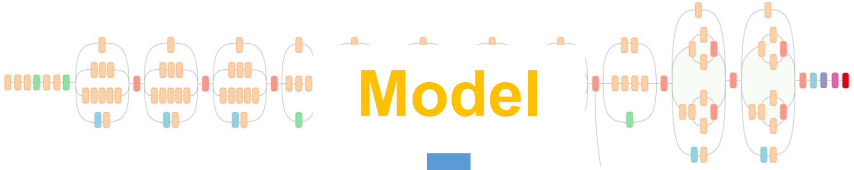
How can we handle different accelerator types and complex memory hierarchy?

Rapidly Evolving

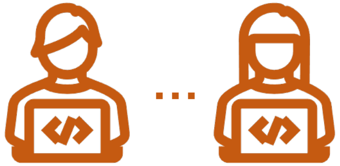
New generation every 2-3 years; but building high-quality systems & compilers takes much longer

How can we deal with the rapid evolution of ML hardware?

Current Practice: Rely on Engineers to Handle HW Complexity



Model



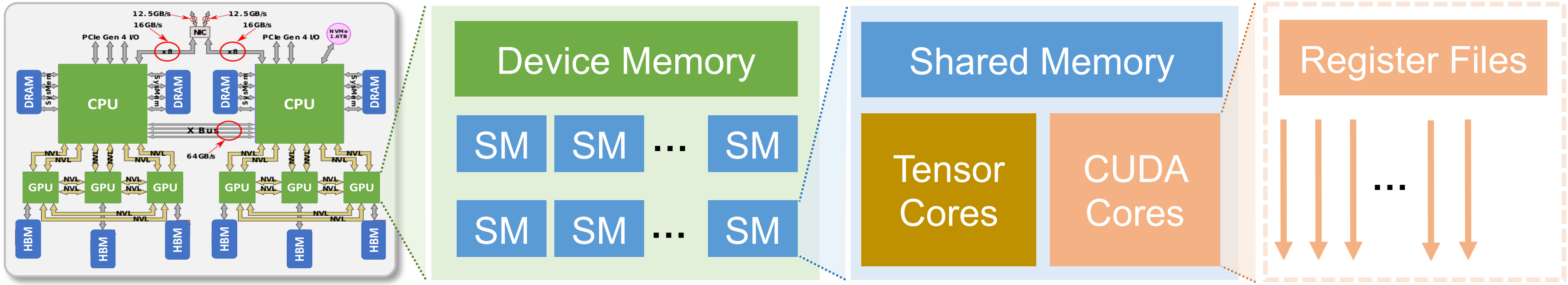
Manually design & implement using HW libraries, compilers, ISAs

ML Systems

Systems optimizations at different hardware levels

Workload partition	Kernel fusion	Shared mem reuse	Instruction selection
Comm/comp. overlap	Kernel orchestration	Warp scheduling	Register allocation
…	…	…	…

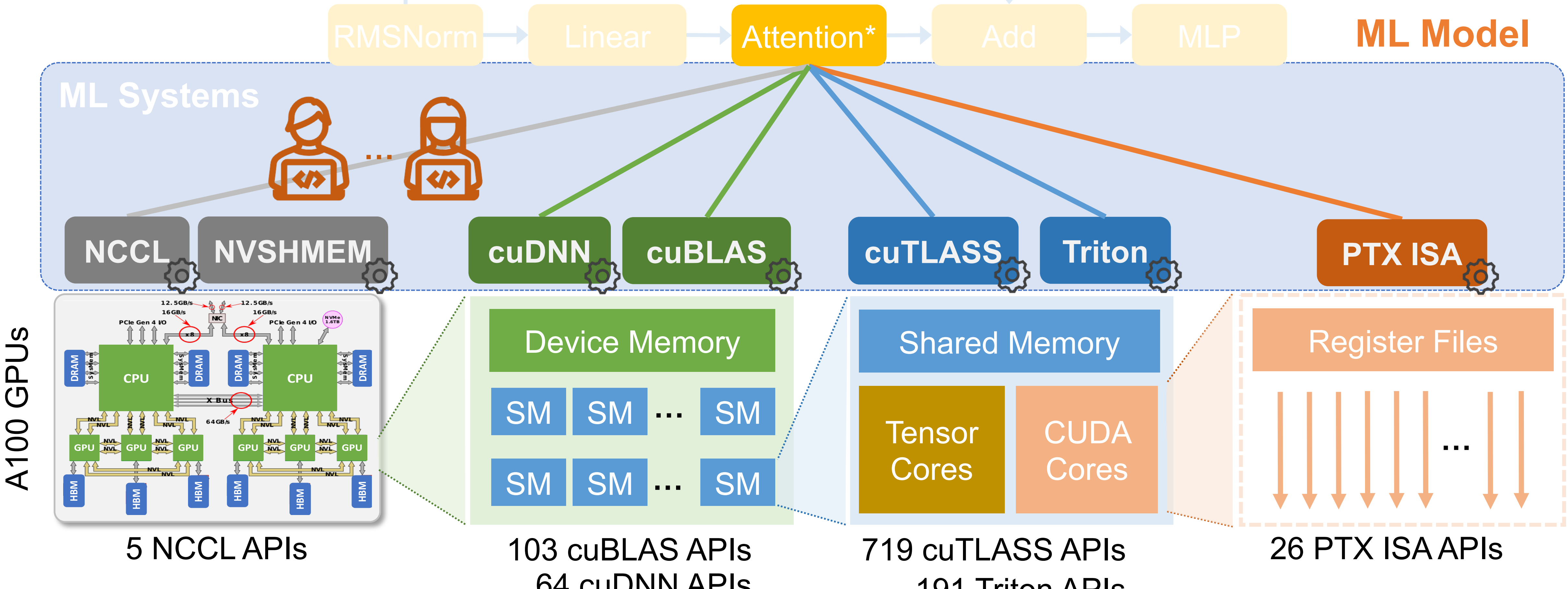
NCCL | NVSHMEM | cuDNN | cuBLAS | cuTLASS | Triton | PTX ISA



Machine-level GPU-level SM-level Thread-level

Issue 1: Time Intensive to Manually Design and Implement

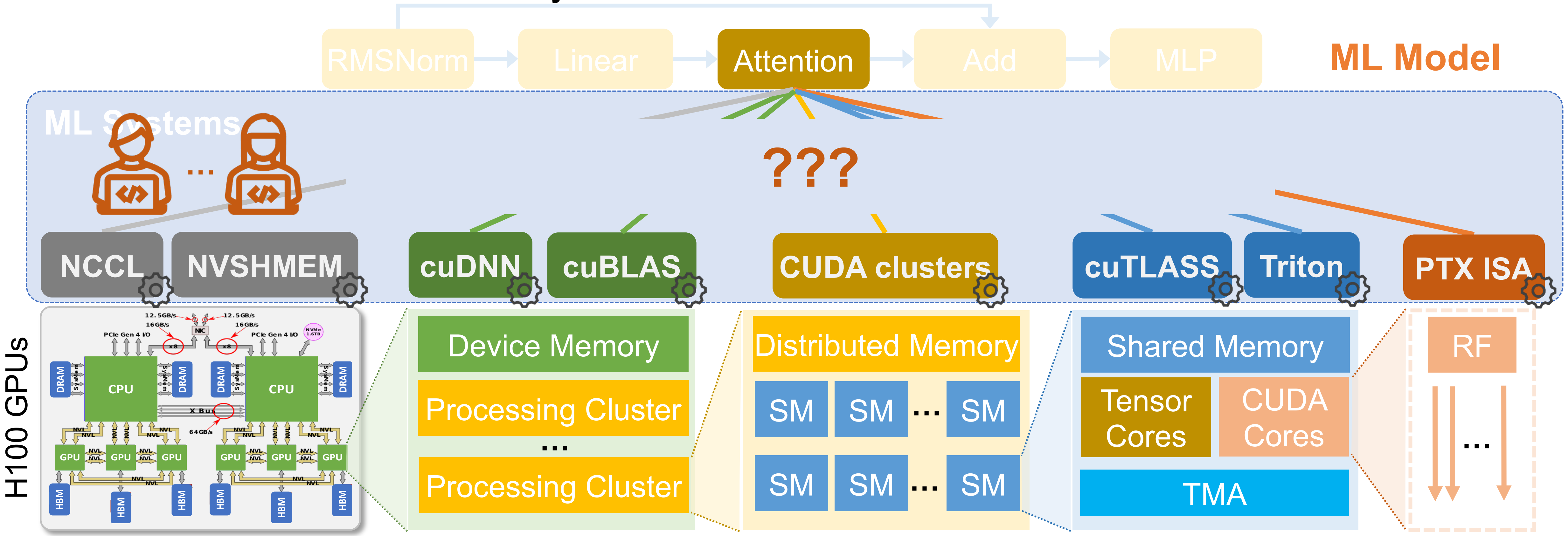
- Complex interactions between HW levels
- Optimizing attention takes many months; 1000+ calls to HW libraries; 60K LOC



* FlashAttention: Fast and Memory-Efficient Exact Attention

Issue 2: New HW Requires New Design and Implementation

- New HW features affect optimization landscape of others
- Reimplement kernels to optimize attention for H100 (15K LOC)
- Not available until two years after H100's release



Can ML systems discover and deploy these optimizations automatically?

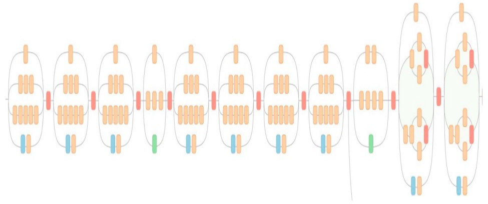
~~Many engineering months, 60K LOC to optimize attention on A100~~

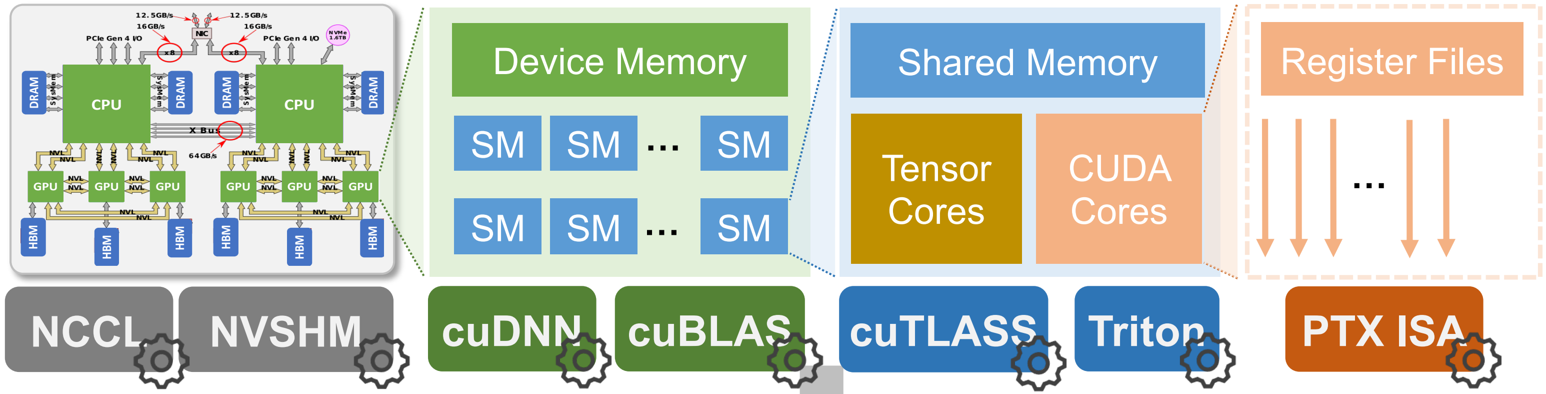
~~Reimplementing kernels, modifying 15K LOC for H100~~

ML systems automate these optimizations with minimal or even no manual effort

Our Vision: ML Systems *Automatically* Discover and Deploy Optimizations at All Levels

ML Model





Hardware Specification

Numerous ways to map ML computation onto parallel hardware

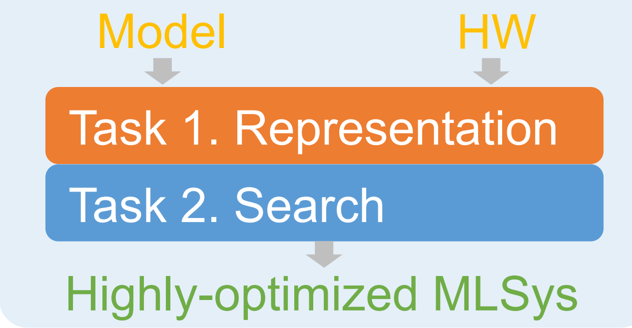
Very Large and Complex Space of Possible Systems Optimizations

Changes made at one level affect optimizations for others

Automated Search Guided by Algebraic Properties of ML

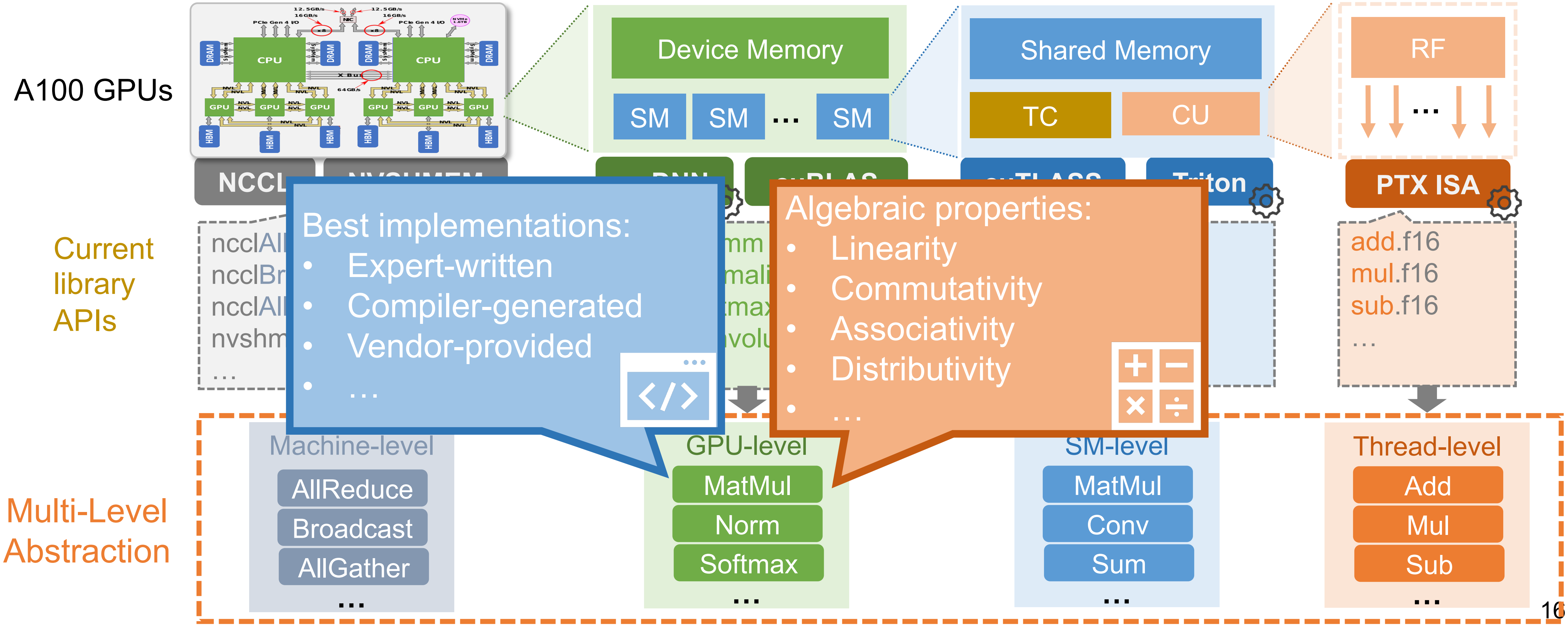
Quickly discover high-quality optimizations

Highly-optimized ML implementations



Task 1: How to Represent Optimizations?

Desiderata: a unified representation across HW layers and architectures



Model HW

Task 1. Representation

Task 2. Search

Highly-optimized MLSys

Task 2: How to Find Performant Systems?

- Need to simultaneously consider many tasks

Machine-level	GPU-level	GPC-level	SM-level	Thread-level
AllReduce	MatMul	AllReduce	MatMul	Add
Broadcast	RMSNorm	Broadcast	Mul	Mul
…	…	…	…	…

Algebraic Properties

Best Implementations

Automated Search

ML Model

Compilation | Parallelization | Scheduling | Resource Allocation | …

Algebraic properties of ML + Techniques in systems, PL, compilers

Highly optimized ML systems

Our Research: Automated End-to-end ML Systems

Multi-Level Abstraction

Machine-level	GPU-level	GPC-level	SM-level	Thread-level
AllReduce	MatMul	AllReduce	MatMul	Add
Broadcast	Norm	Broadcast	Mul	Mul
…	…	…	…	…

Algebraic Properties

Best Impl.

Algebraic properties of ML + Techniques in systems, PL, compilers

Compilation
- TASO (SOSP'19)
- PET (OSDI'21)
- EinNet (OSDI'23)
- Mirage (OSDI'25)

Parallelization
- FlexFlow (MLSys'19)
- Unity (OSDI'22)
- TopoOpt (NSDI'23)
- GPP (ASPLOS'25)

Scheduling
- Sia (SOSP'23)
- Parcae (NSDI'24)
- Korch (ASPLOS'24)
- Helix (ASPLOS'25)

Resource Allocation
- Dorylus (OSDI'21)
- Bamboo (NSDI'23)
- SpotServe (ASPLOS'24)

How to Address Three Challenges?

Capture model- and HW-specific optimizations across all levels

Massively Parallel

Increasingly Heterogeneous

Rapidly Evolving

Multi-Level Abstraction

Machine-level	GPU-level	GPC-level	SM-level	Thread-level
AllReduce	MatMul	AllReduce	MatMul	Add
Broadcast	Norm	Broadcast	Mul	Mul
…	…	…	…	…

Algebraic Properties

Best Impl.

Algebraic properties of ML + Techniques in systems, PL, compilers

Compilation
- TASO (SOSP'19)
- PET (OSDI'21)
- EinNet (OSDI'23)
- Mirage (OSDI'25)

Parallelization
- FlexFlow (MLSys'19)
- Unity (OSDI'22)
- TopoOpt (NSDI'23)
- GPP (ASPLOS'25)

Scheduling
- Sia (SOSP'23)
- Parcae (NSDI'24)
- Korch (ASPLOS'24)
- Helix (ASPLOS'25)

Resource Alloc.
- Dorylus (OSDI'21)
- Bamboo (NSDI'23)
- SpotServe (ASPLOS'24)

Automatically discover efficient ways to parallelize ML

Much less engineering efforts; faster adaptation to new HW

Our Techniques Generalize Beyond ML Systems

Multi-Level Abstraction

Machine-level	GPU-level	GPC-level	SM-level	Thread-level
AllReduce	MatMul	AllReduce	MatMul	Add
Broadcast	Norm	Broadcast	Mul	Mul
…	…	…	…	…

Algebraic Properties

Best Impl.

Algebraic properties of ML + Techniques in systems, PL, compilers

Compilation
- TASO (SOSP'19)
- PET (OSDI'21)
- EinNet (OSDI'23)
- Mirage (OSDI'25)

Parallelization
- FlexFlow (MLSys'19)
- Unity (OSDI'22)
- TopoOpt (NSDI'23)
- GPP (ASPLOS'25)

Scheduling
- Sia (SOSP'23)
- Parcae (NSDI'24)
- Korch (ASPLOS'24)
- Helix (ASPLOS'25)

Resource Allocation
- Dorylus (OSDI'21)
- Bamboo (NSDI'23)
- SpotServe (ASPLOS'24)

Quantum Circuit Optimizer
- Quartz (PLDI'22)
- Quarl (OOPSLA'24)

Quantum Algebra

Database Query Optimizer
- TOD (VLDB'22)
- SDPipe (VLDB'23)

Relational Algebra

Our Techniques Generalize Beyond ML Systems

Multi-Level Abstraction

Machine-level	GPU-level	GPC-level	SM-level	Thread-level
AllReduce	MatMul	AllReduce	MatMul	Add
Broadcast	Norm	Broadcast	Mul	Mul
…	…	…	…	…

Algebraic Properties

Best Impl.

Algebraic properties of ML + Techniques in systems, PL, compilers

Compilation
- TASO (SOSP'19)
- PET (OSDI'21)
- EinNet (OSDI'23)
- Mirage (OSDI'25)

Parallelization
- FlexFlow (MLSys'19)
- Unity (OSDI'22)
- TopoOpt (NSDI'23)
- GPP (ASPLOS'25)

Scheduling
- Sia (SOSP'23)
- Parcae (NSDI'24)
- Korch (ASPLOS'24)
- Helix (ASPLOS'25)

Resource Allocation
- Dorylus (OSDI'21)
- Bamboo (NSDI'23)
- SpotServe (ASPLOS'24)

Quantum Circuit Optimizer
- Quartz (PLDI'22)
- Quarl (OOPSLA'24)

Quantum Algebra

Database Query Optimizer
- TOD (VLDB'22)
- SDPipe (VLDB'23)

Relational Algebra

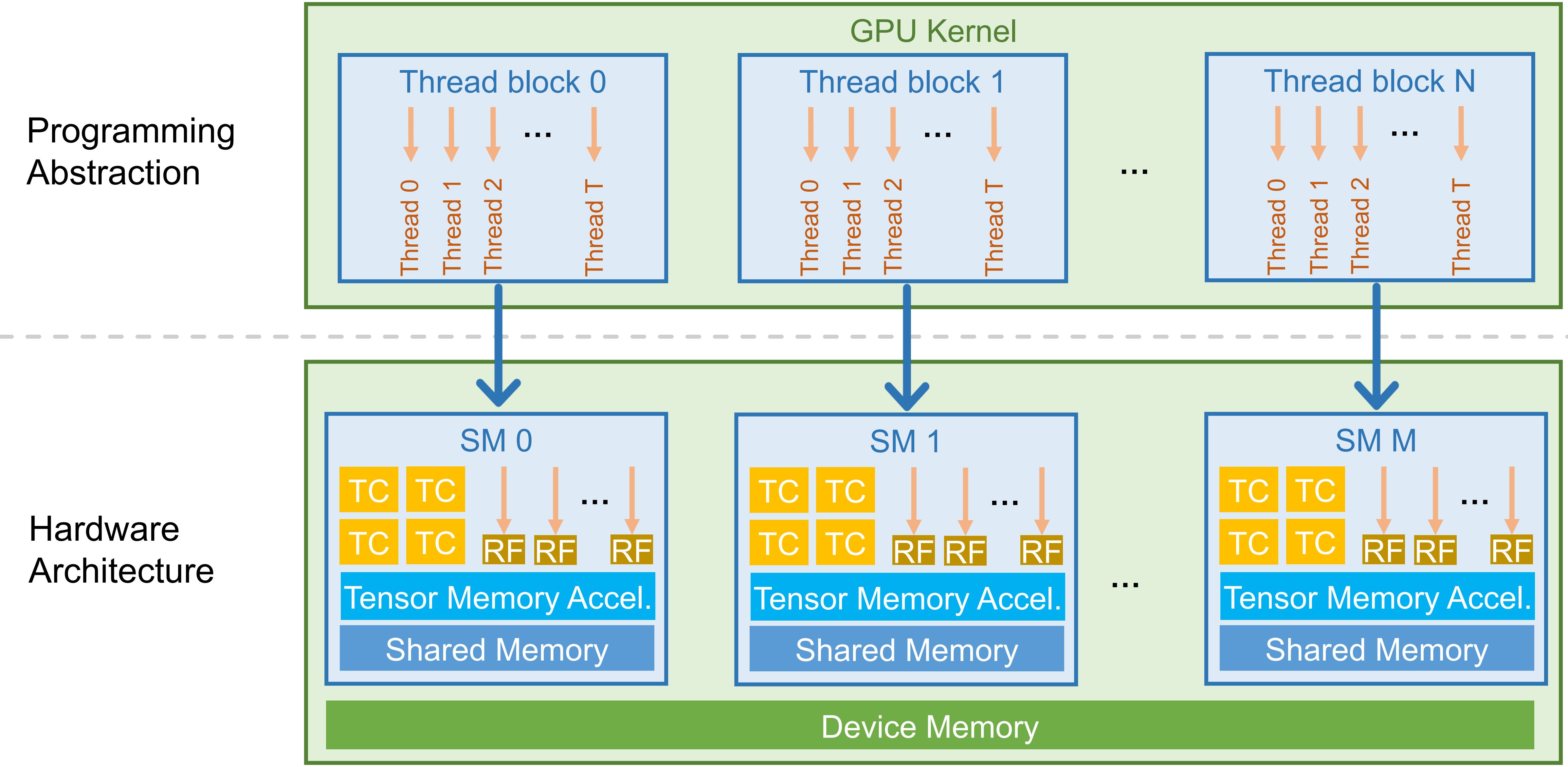
Mirage: A Multi-Level SuperOptimizer for ML

Key idea: automatically translates pure math definition of ML models into highly optimized GPU code

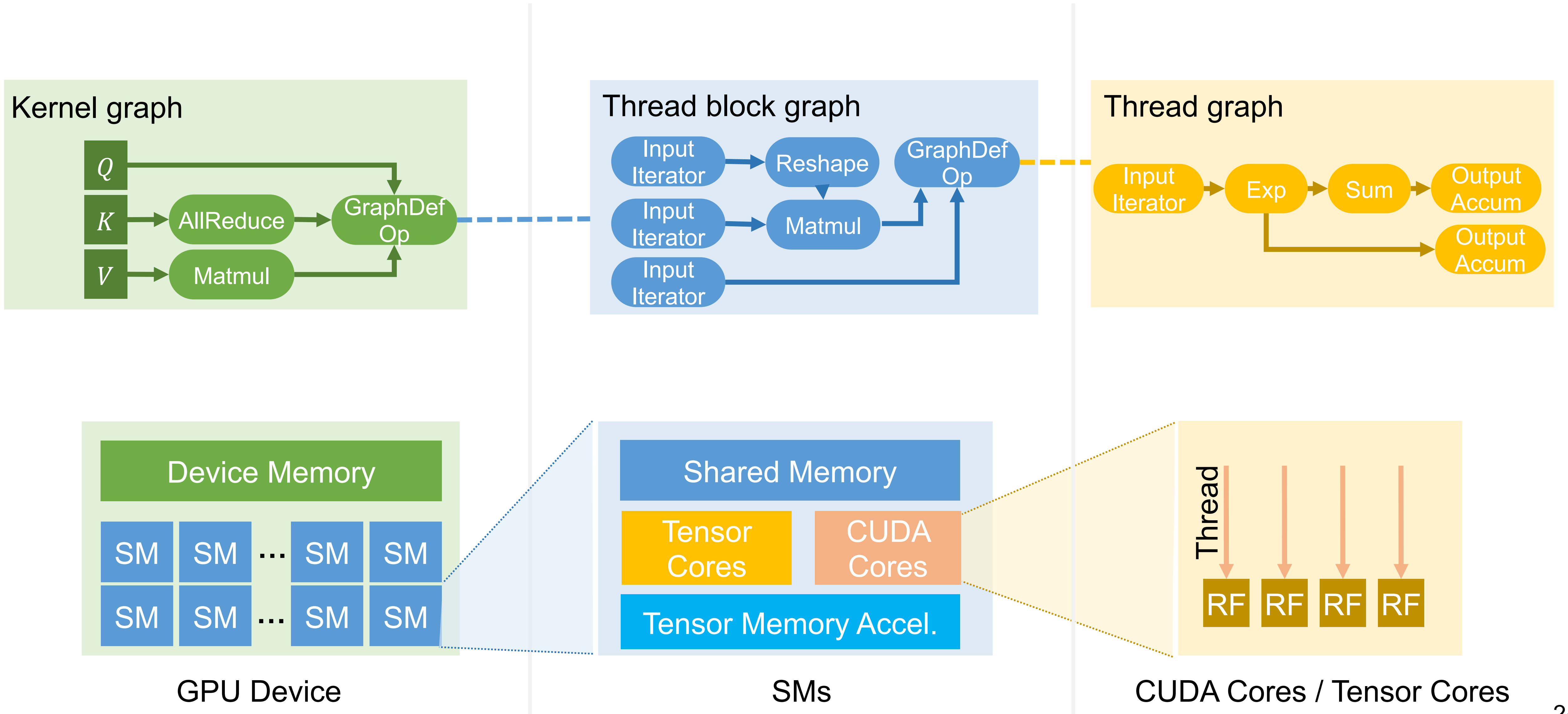
- **Less engineering effort**: thousands of lines of CUDA code → a few lines of code in Mirage
- **Better performance**: outperform existing systems by 1.1-2.9x
- **Faster adaptation:** day-0 support for new models; no manual effort

1. A Multi-Level Superoptimizer for Tensor Programs. OSDI’25.
2. EinNet: Optimizing Tensor Programs with Derivation-Based Transformations. OSDI’23.
3. PET: Optimizing Tensor Programs with Partially Equivalent Transformations and Automated Corrections. OSDI’21.
4. TASO: Optimizing Deep Learning Computation with Automated Generation of Graph Substitutions. SOSP’19

GPU Programming 101

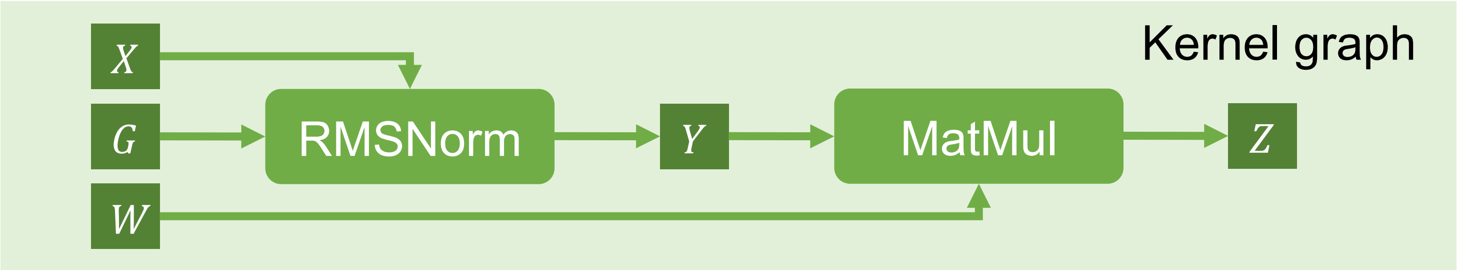


μGraphs: Hierarchical Graph Representation



Motivating Example: RMSNorm & MatMul in LLMs

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



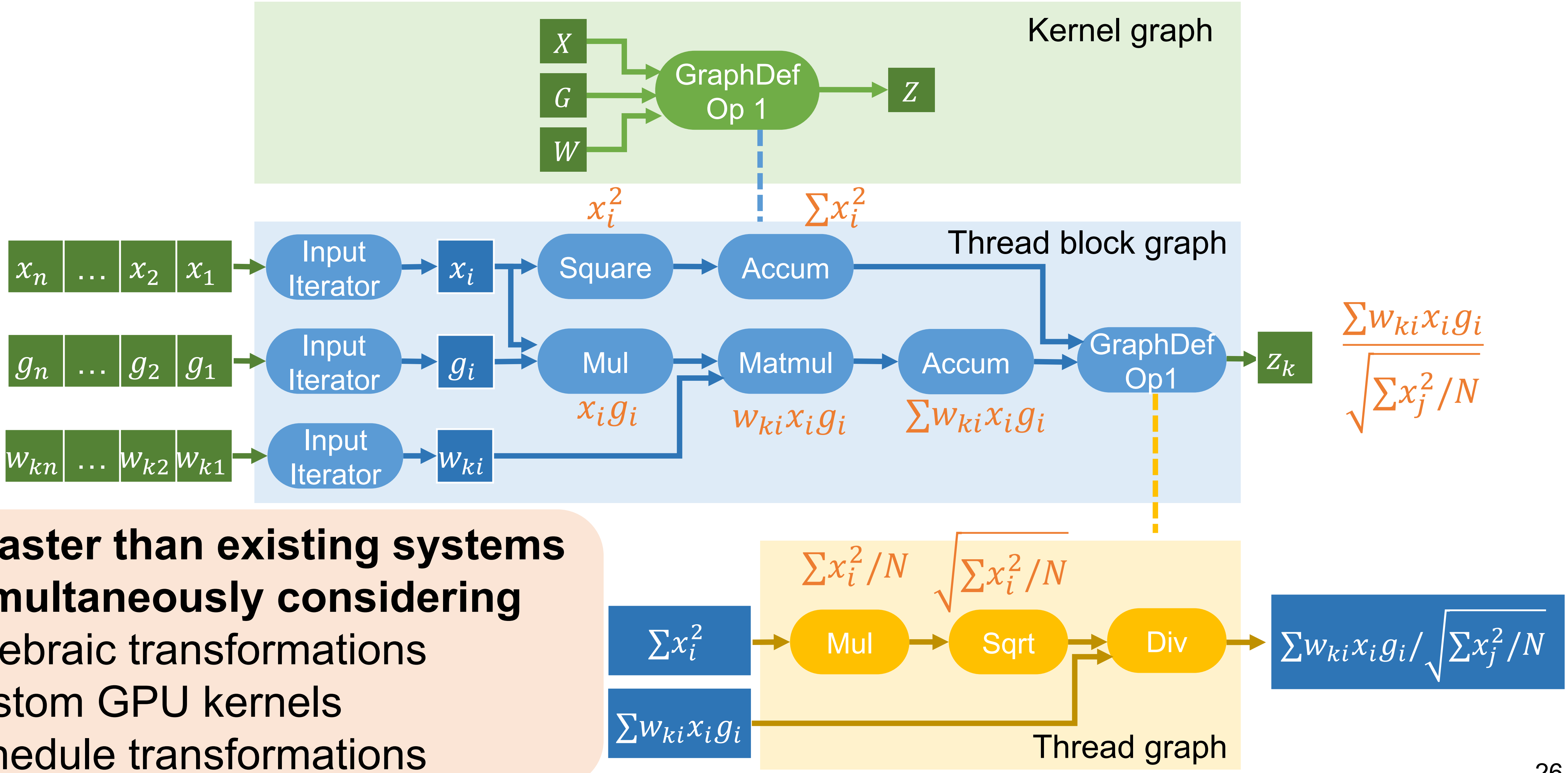
$$y_i = \frac{x_i g_i}{\sqrt{\frac{1}{N}\sum_j x_j^2}}$$

$$z_i = \sum_k w_{ik} y_k$$

Performance issues:

1. No shared memory reuse
2. Kernel launch overhead

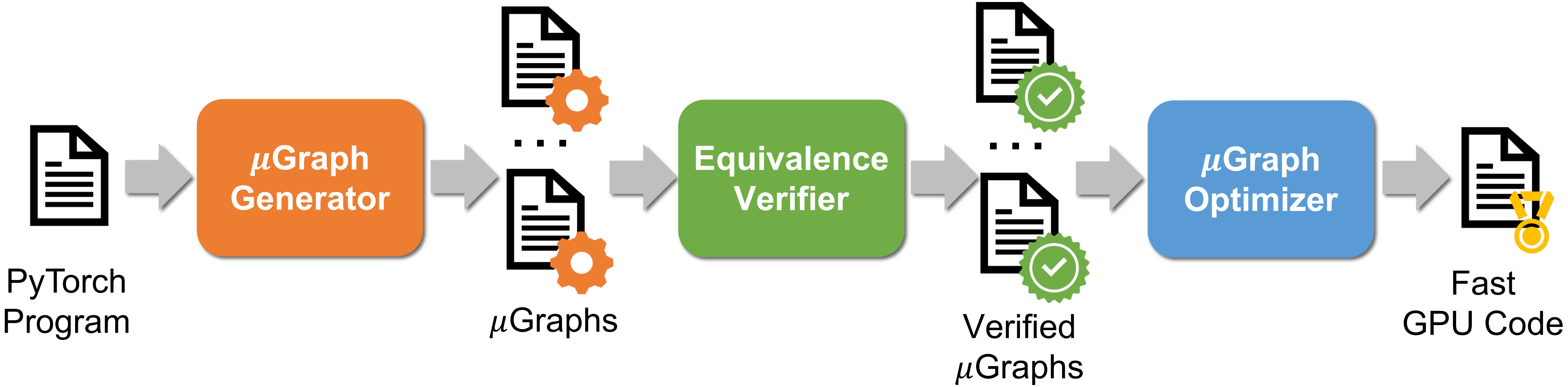
Best Discovered μGraph for RMSNorm & MatMul



2.2x faster than existing systems by simultaneously considering

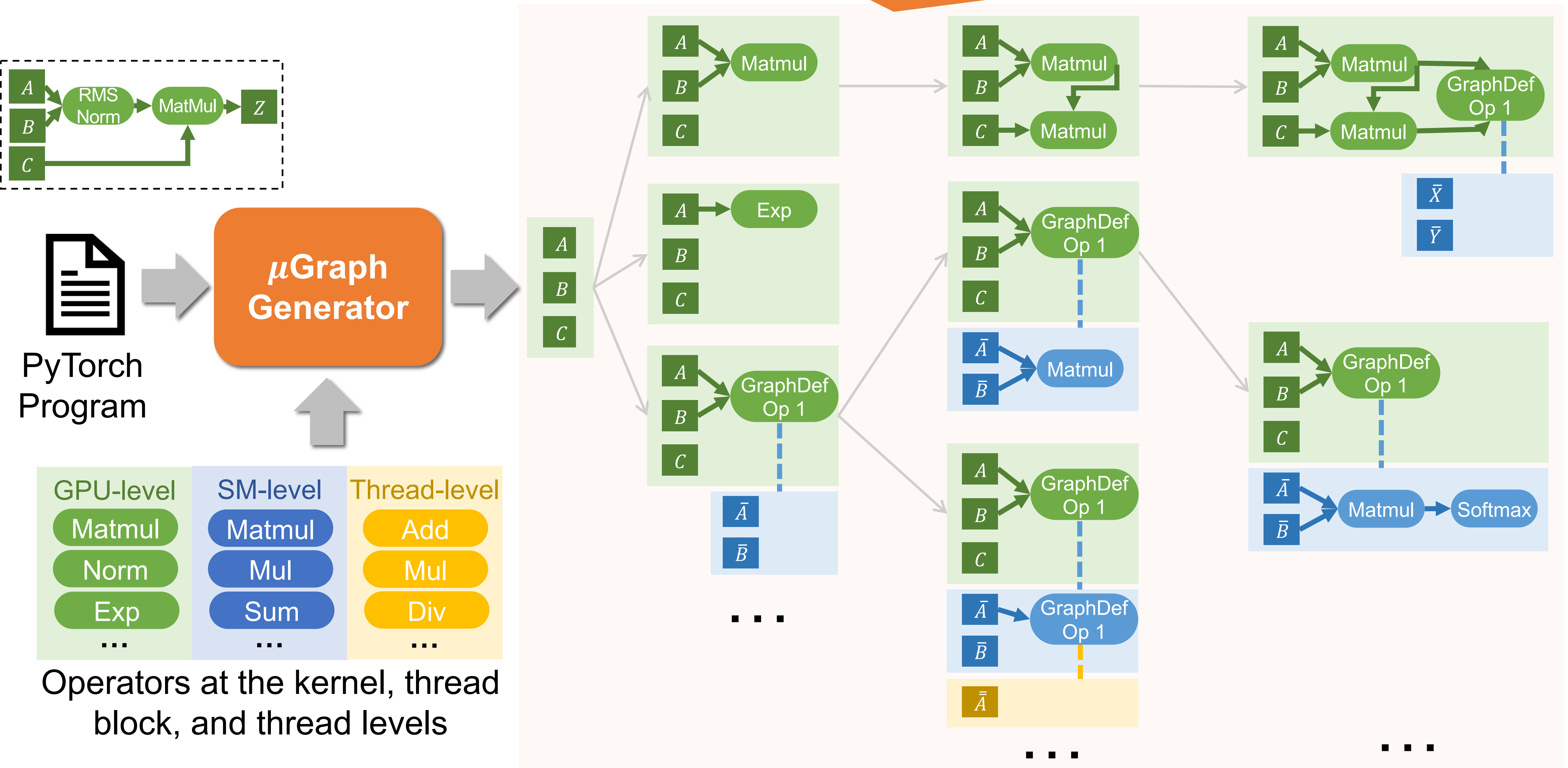
- Algebraic transformations
- Custom GPU kernels
- Schedule transformations

Mirage Overview

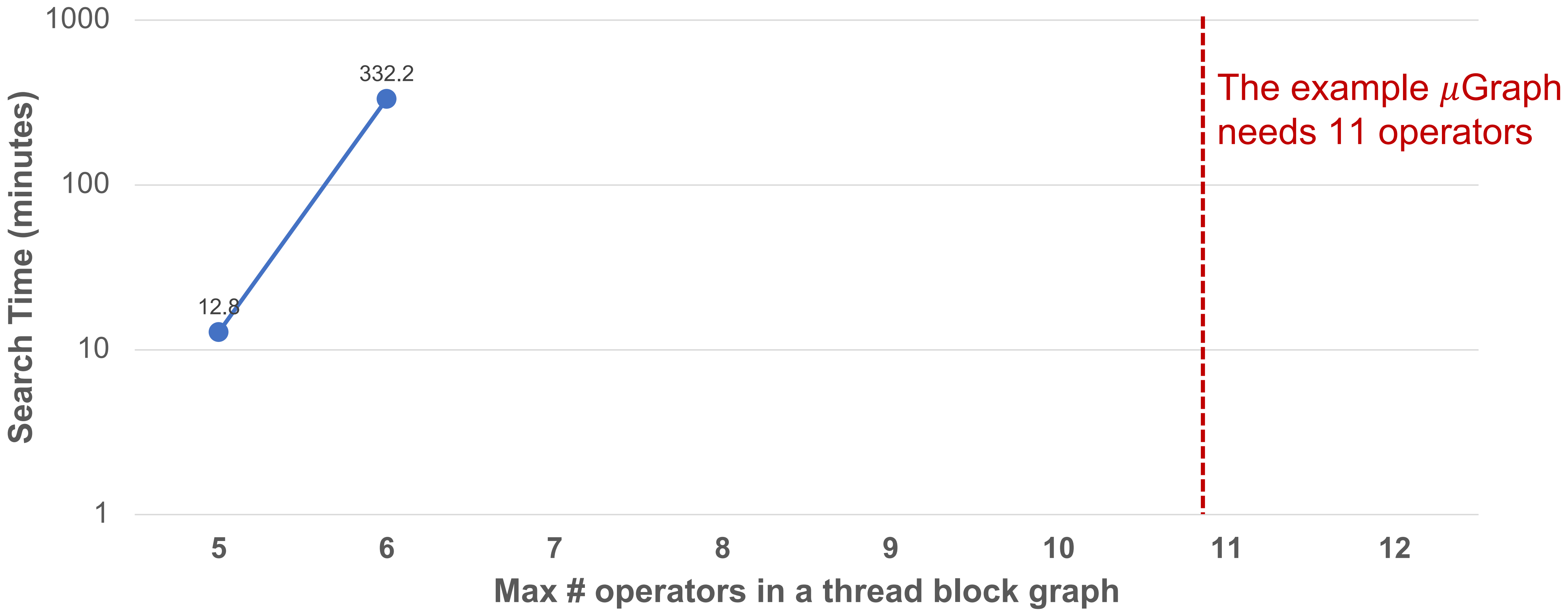


μGraph Generator

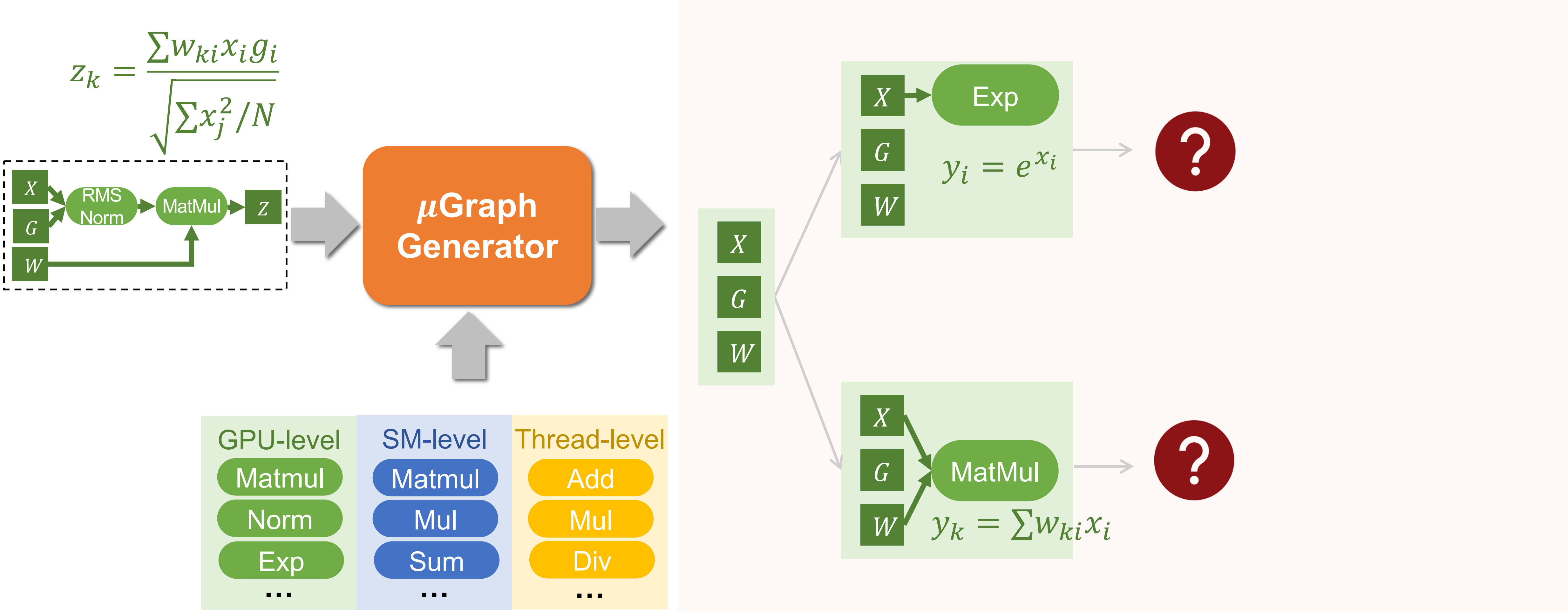
Consider all possible μGraphs using available operators



Challenge: Extremely Large Search Space

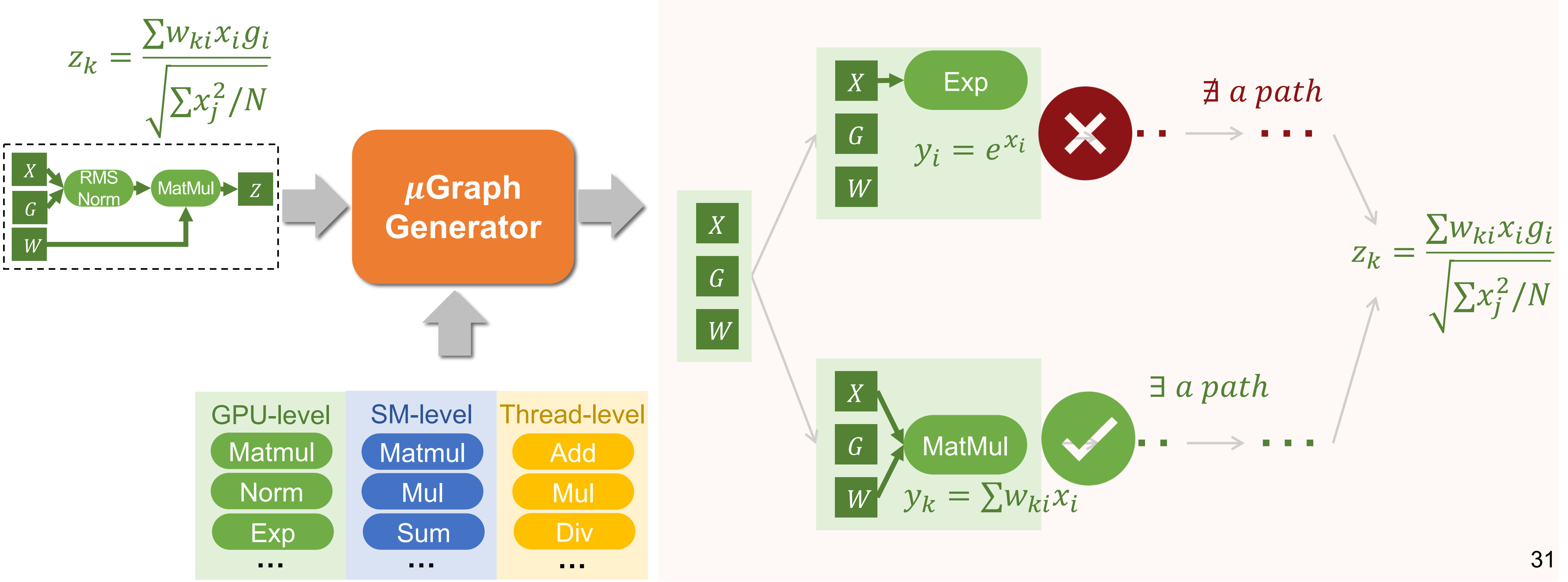


Can Algebraic Properties Guide Search?



Can Algebraic Properties Guide Search?

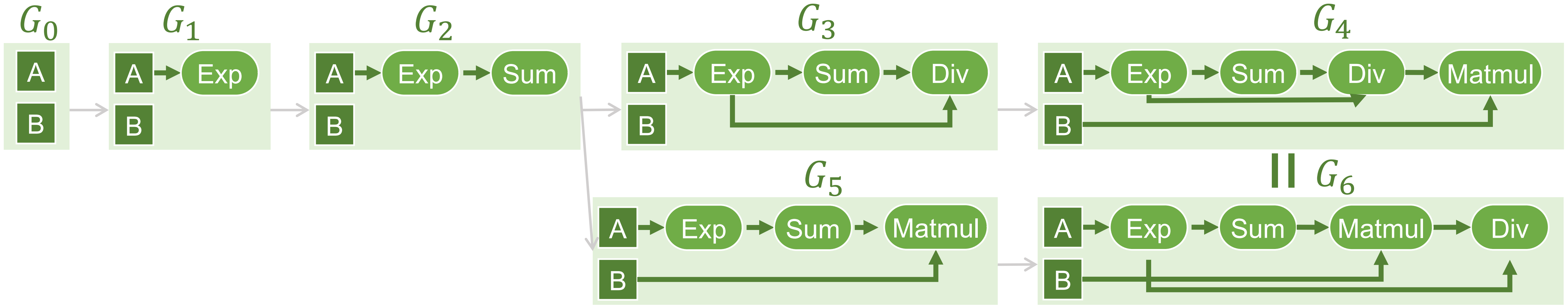
Key idea: prune candidates that do not have a path to desired computation using given operators



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: $E(G_4) = E(G_6)$



$\exists$ a path from G to a μGraph G' equivalent to the input G_0?

$$G \longrightarrow \cdots \longrightarrow \cdots \longrightarrow \underset{G_0}{\underset{||}{G'}}$$

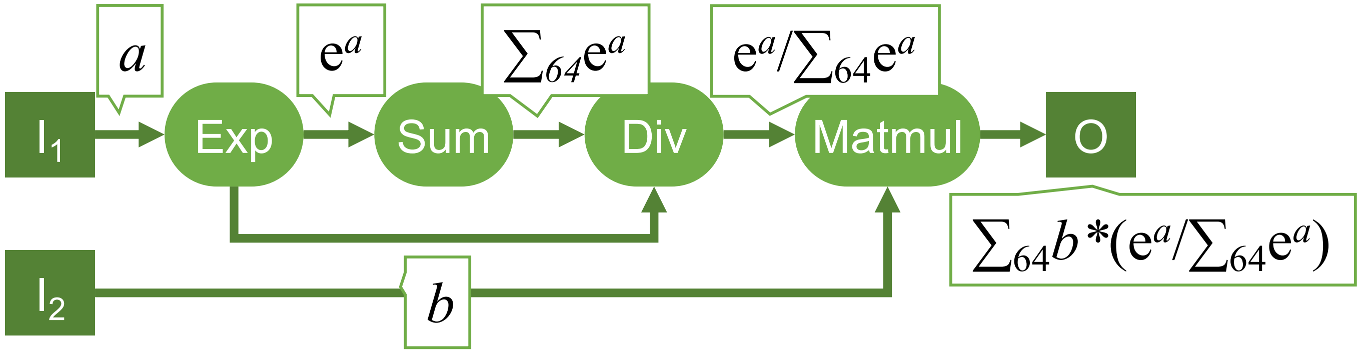
$E(G) \preccurlyeq E(G_0)$?

Abstract Expression: An Implementation

Represent a tensor's computation by abstracting away index details

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

Recursively compute abstract expressions



Abstract Expression: An Implementation

Mirage uses first-order logic to reason about two relations

Goal 1: subexpression

Subexpression Axioms A_{sub}

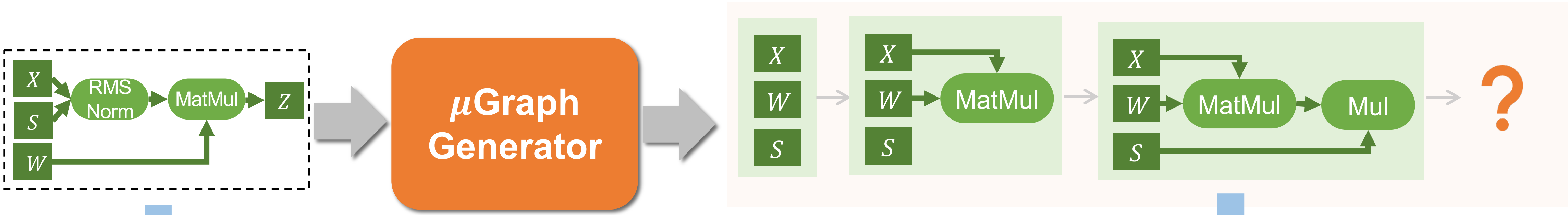
$\forall x, y.\ \mathsf{subexpr}(x, \mathsf{add}(x, y))$	
$\forall x, y.\ \mathsf{subexpr}(x, \mathsf{mul}(x, y))$	
$\forall x, y.\ \mathsf{subexpr}(x, \mathsf{div}(x, y))$	
$\forall x, y.\ \mathsf{subexpr}(y, \mathsf{div}(x, y))$	
$\forall x.\ \mathsf{subexpr}(x, \mathsf{exp}(x))$	
$\forall x, i.\ \mathsf{subexpr}(x,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

Goal 2: equivalence

Equivalence Axioms A_{eq}

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

Abstract Expression-Guided Search



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

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

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

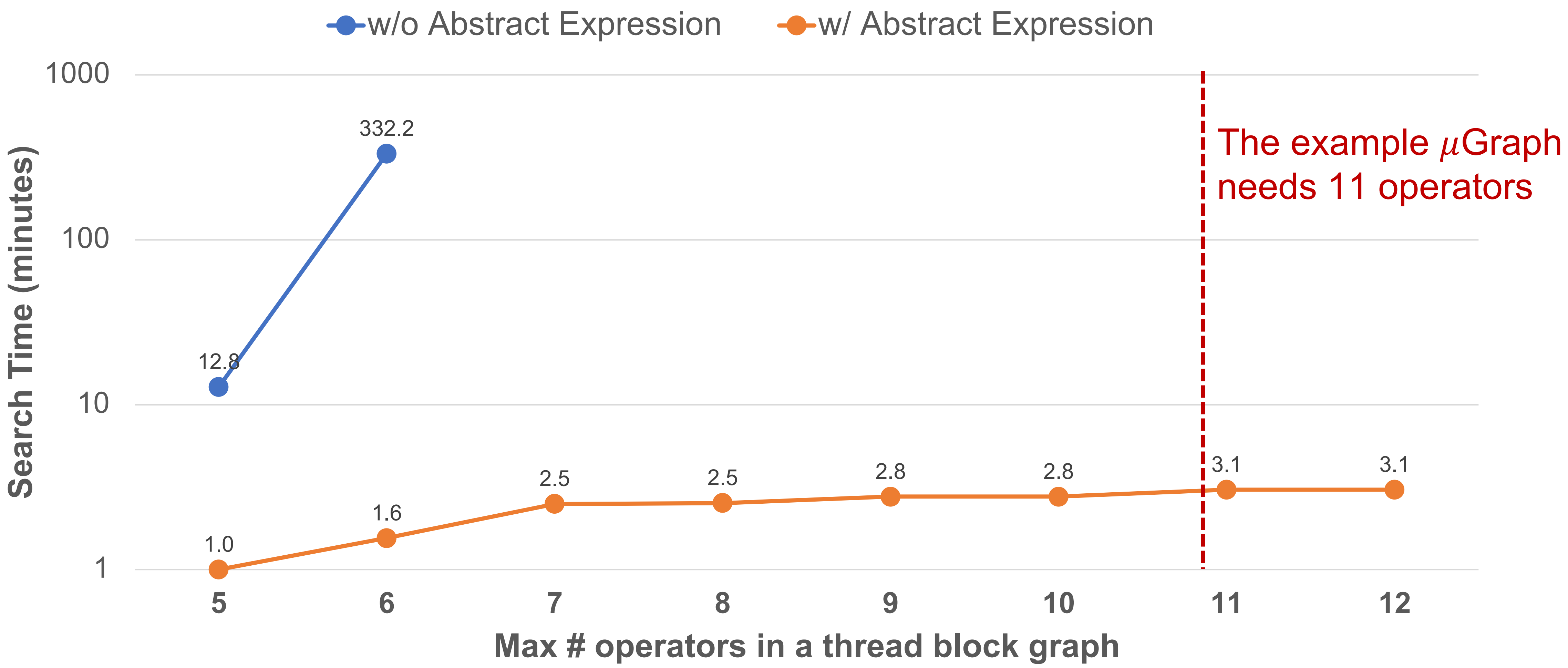
$E(G) \preccurlyeq E(G_0)$?

Automated Theorem Prover

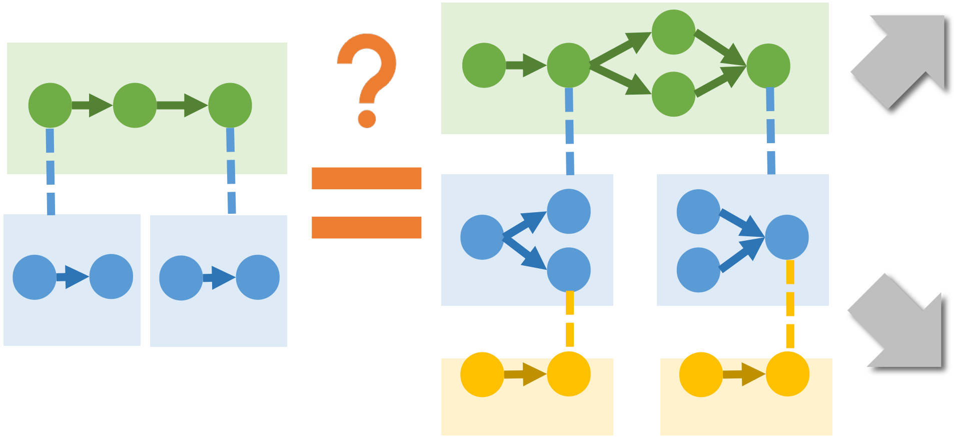
continue the search

prune candidate μGraph

Abstract Expression Significantly Improves Scalability



μGraph Verifiers



Probabilistic Equivalence Verifier

Random tests over finite fields

- Pro: directly support new operators
- Con: limited to a subset of models (i.e., those w/o ReLU)

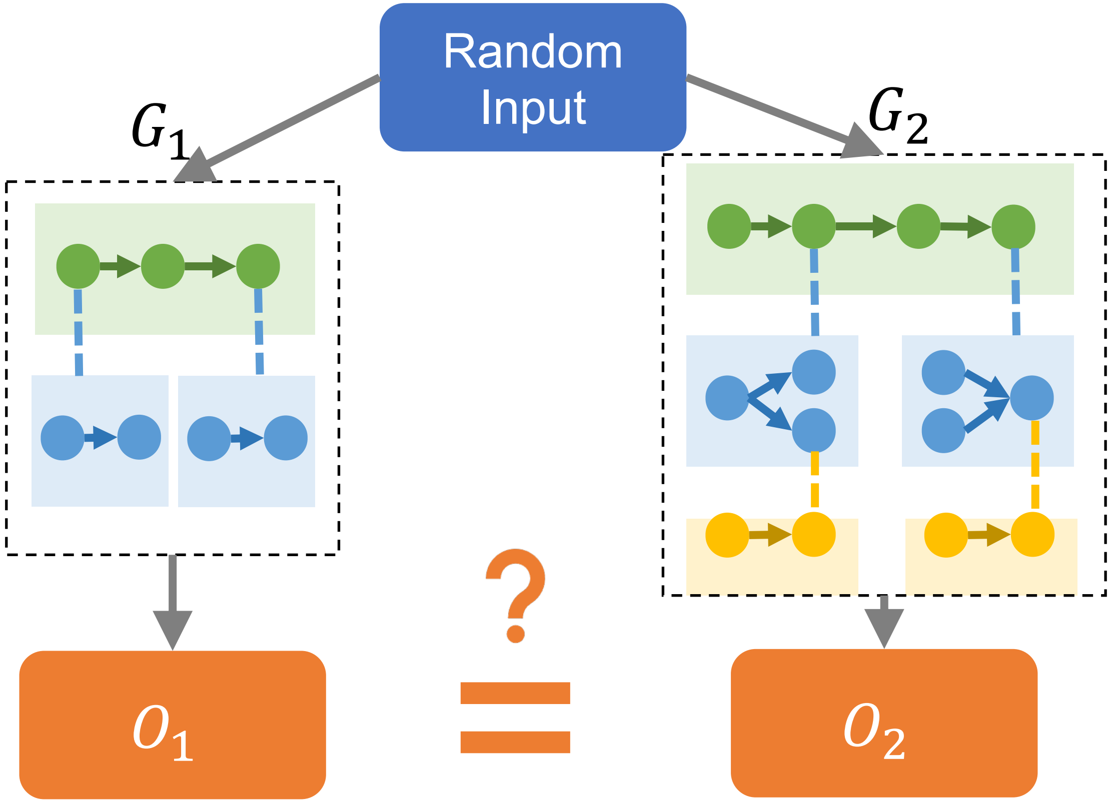
Solver-based Equivalence Verifier

Formal verification

- Pro: support arbitrary ML operators
- Con: manual effort to support new operators

Probabilistic Equivalence Verifier

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

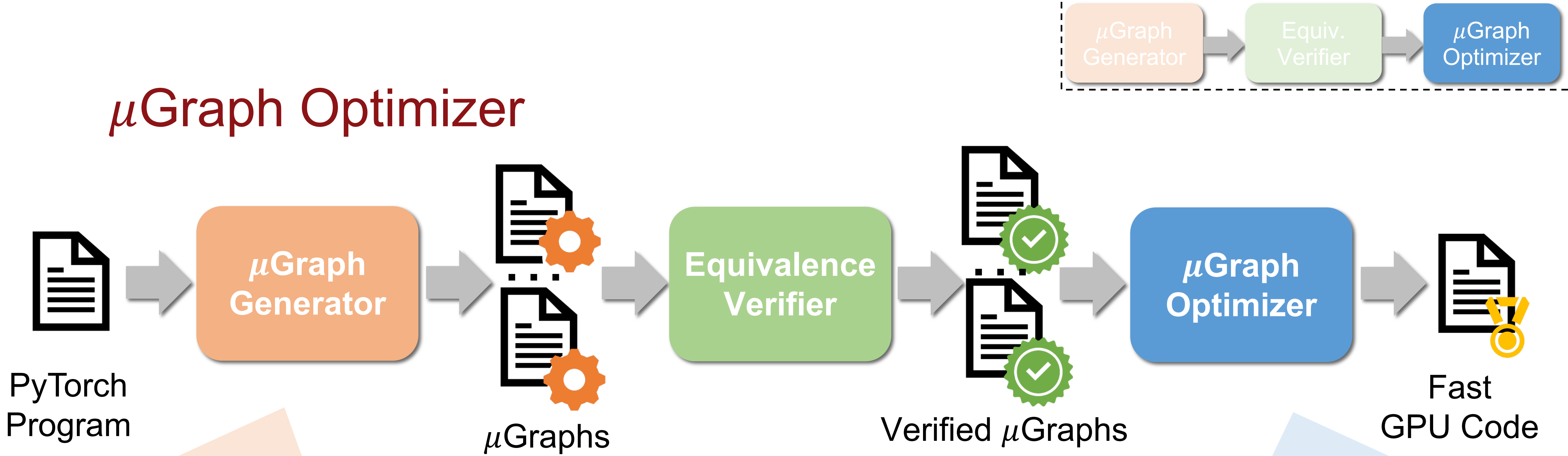


Theorem 1: if G_1 is equivalent to G_2, then $O_1 = O_2$

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

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

μGraph Optimizer



Only consider output-alternating optimizations:

- Algebraic transformations
- Kernel instantiation
- Compute organization

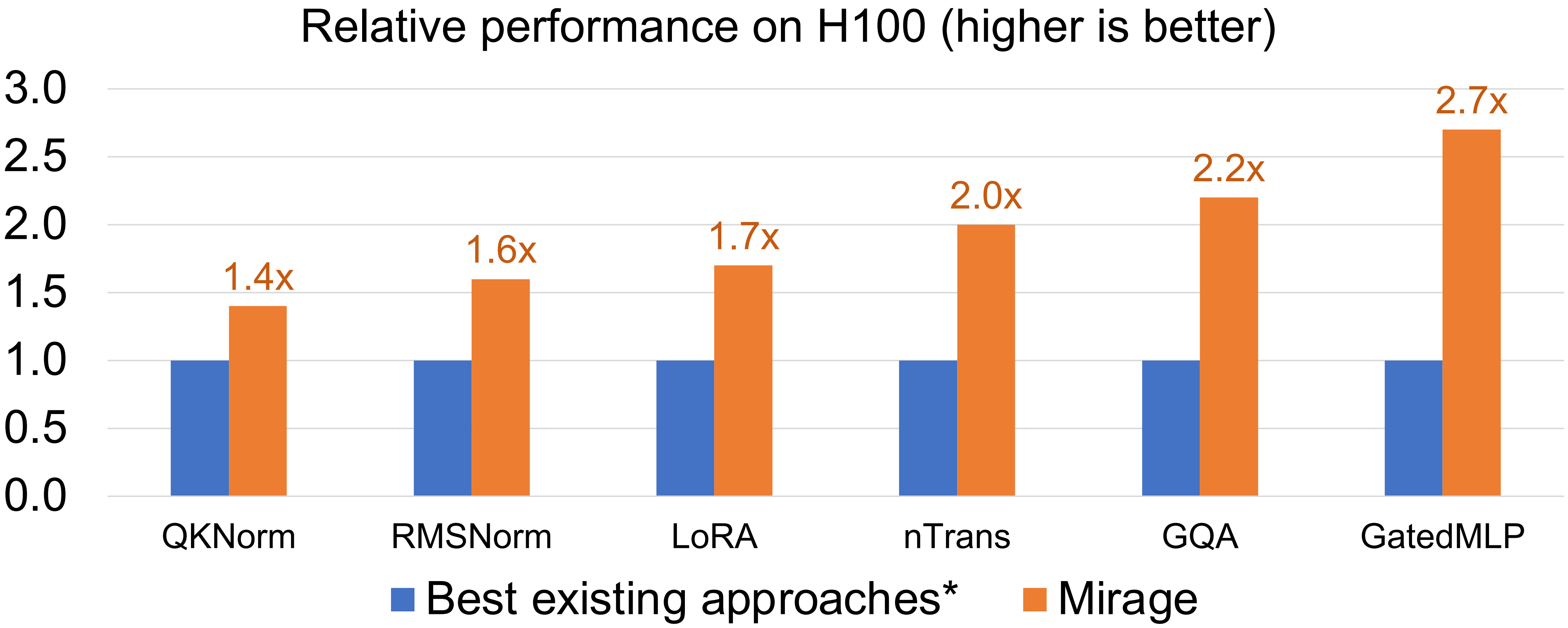
Reduce generator's search space

Other optimizations are deferred to μGraph Optimizer:

- Tensor layouts
- Memory planning
- Operator scheduling

Solve these tasks optimally

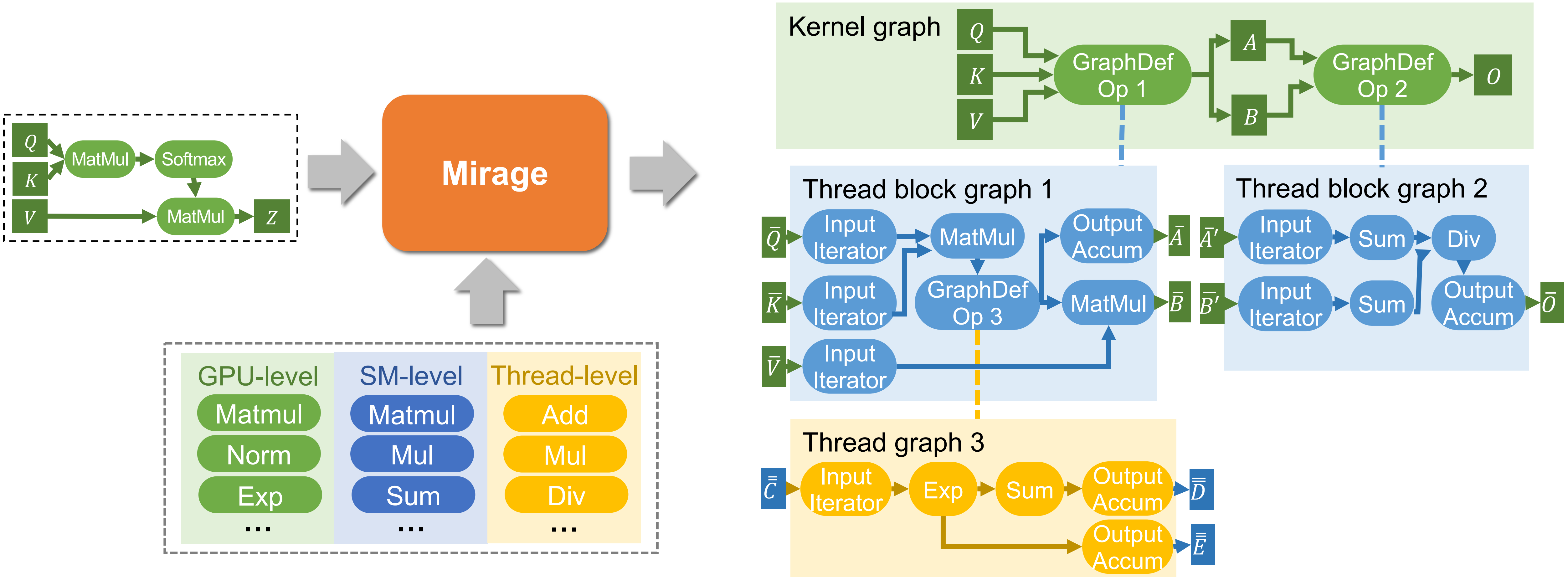
Mirage Outperforms Existing Approaches



$$\text{*Best existing approaches are the best of } \underbrace{\text{cuDNN/cuBLAS}}_{\text{Vender-provided}}\text{, } \underbrace{\text{FlashAttention, PyTorch, TensorRT}}_{\text{Expert-written}}\text{, } \underbrace{\text{Triton, and TVM}}_{\text{Compiler-generated}}\text{.}$$

Mirage Discovers Hardware-Customized μGraphs

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

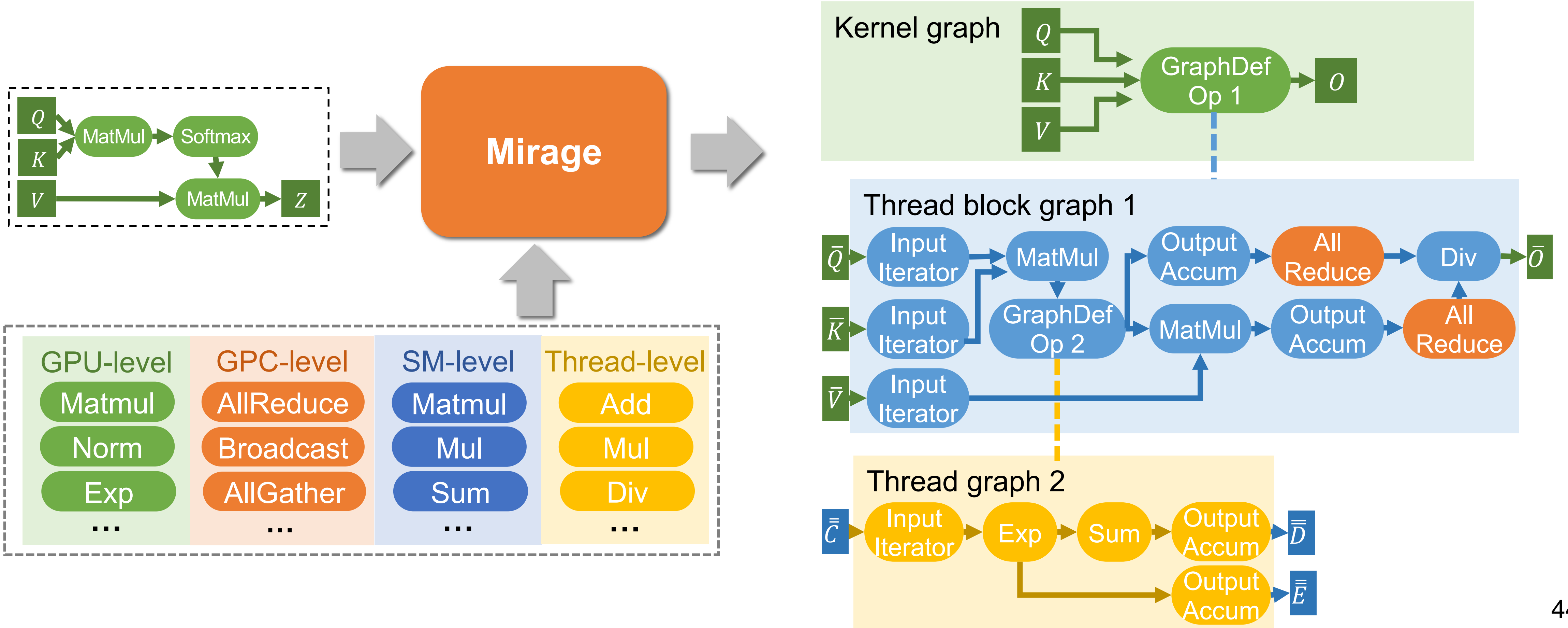


* FlashDecoding++: Faster LLM Inference with Asynchronization, Flat GEMM Optimization, and Heuristics. MLSys'24

Mirage Discovers Hardware-Customized μGraphs

Leverage GPC-level AllReduce to accelerate attention on H100

- 2.2x faster than best existing kernels



Our Research: Superoptimizing ML Systems

Multi-Level Abstraction

Machine-level	GPU-level	GPC-level	SM-level	Thread-level
AllReduce	MatMul	AllReduce	MatMul	Add
Broadcast	Norm	Broadcast	Mul	Mul
…	…	…	…	…

Algebraic Properties

Best Impl.

Algebraic properties of ML + Techniques in systems, PL, compilers

Compilation
- TASO (SOSP'19)
- PET (OSDI'21)
- EinNet (OSDI'23)
- Mirage (OSDI'25)

Parallelization
- FlexFlow (MLSys'19)
- Unity (OSDI'22)
- TopoOpt (NSDI'23)
- GPP (ASPLOS'25)

Scheduling
- Sia (SOSP'23)
- Parcae (NSDI'24)
- Korch (ASPLOS'24)
- Helix (ASPLOS'25)

Resource Allocation
- Dorylus (OSDI'21)
- Bamboo (NSDI'23)
- SpotServe (ASPLOS'24)

catalyst https://catalyst.cs.cmu.edu/