# 15-779 Lecture 7:

# Advanced CUDA Programming: Mega-Kernel

**Zhihao Jia**

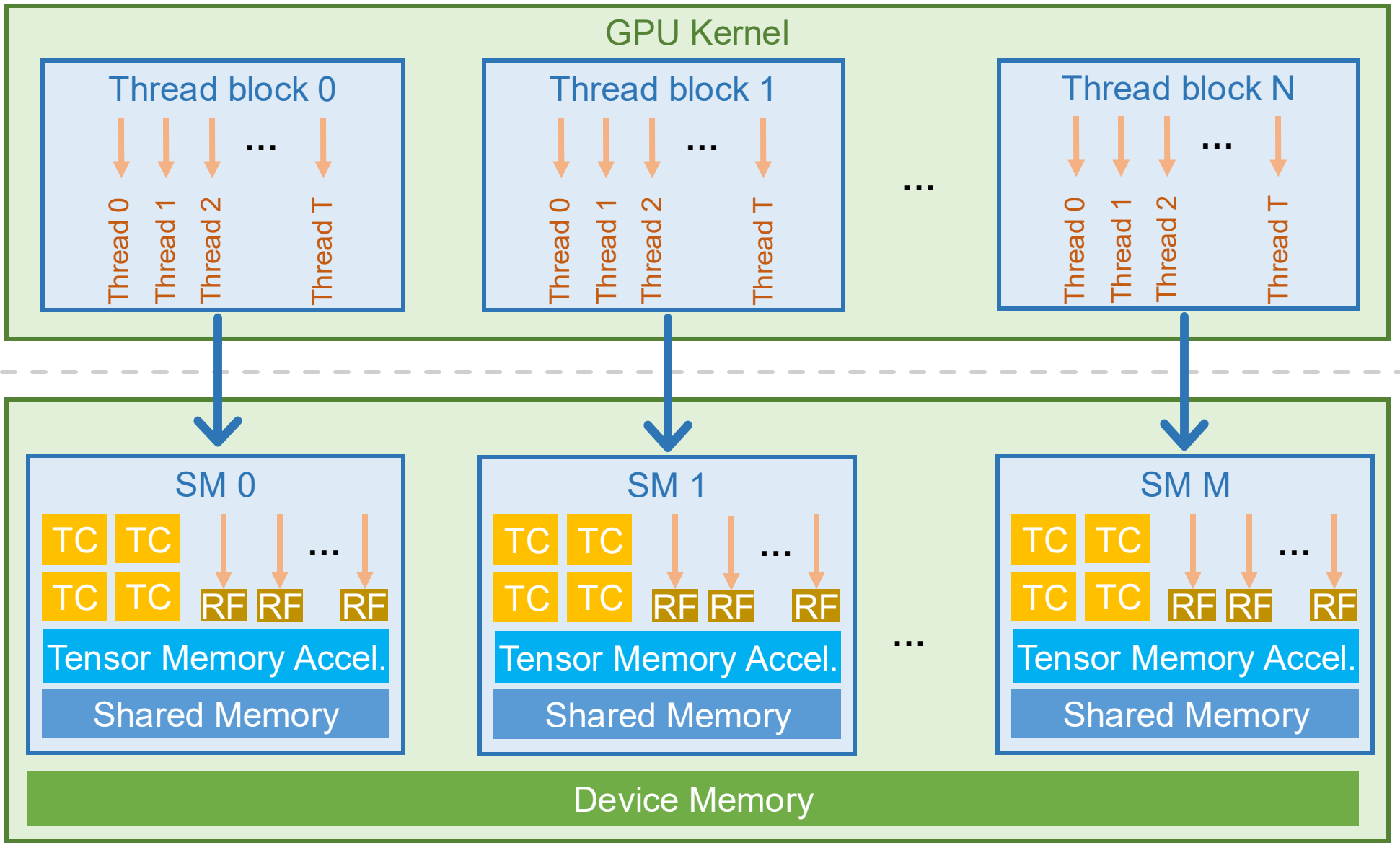
Computer Science Department

Carnegie Mellon University

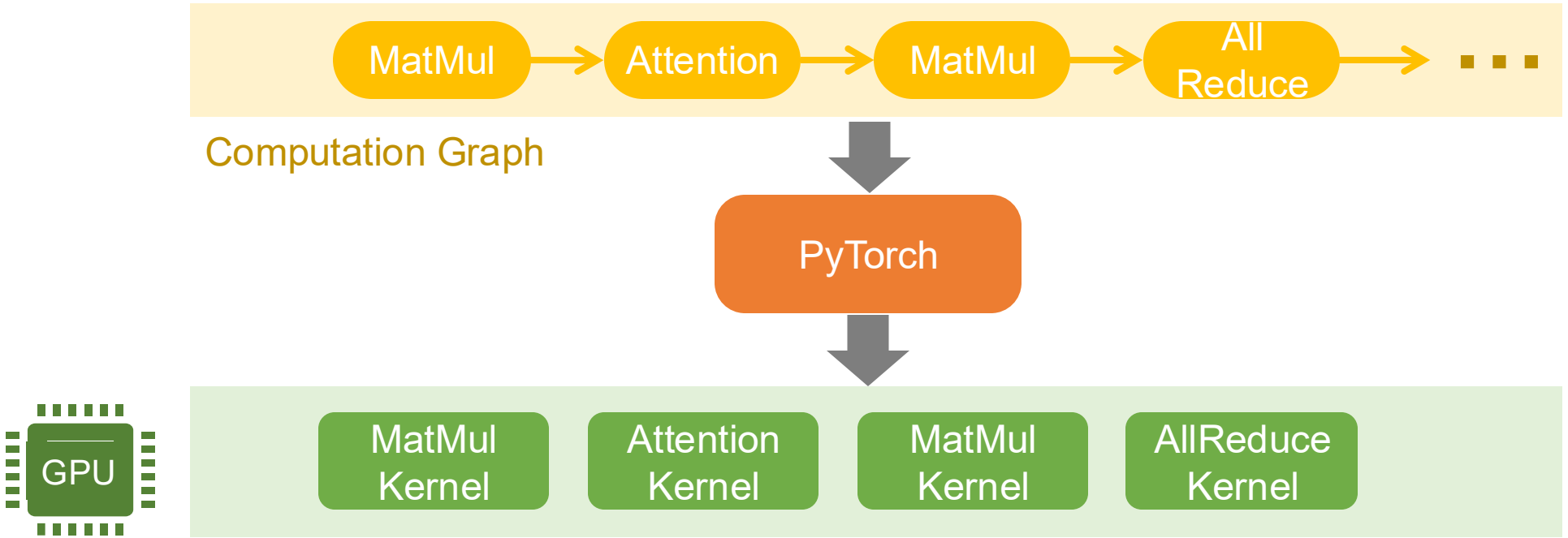
# What is a (Mega-)Kernel?

Programming Abstraction

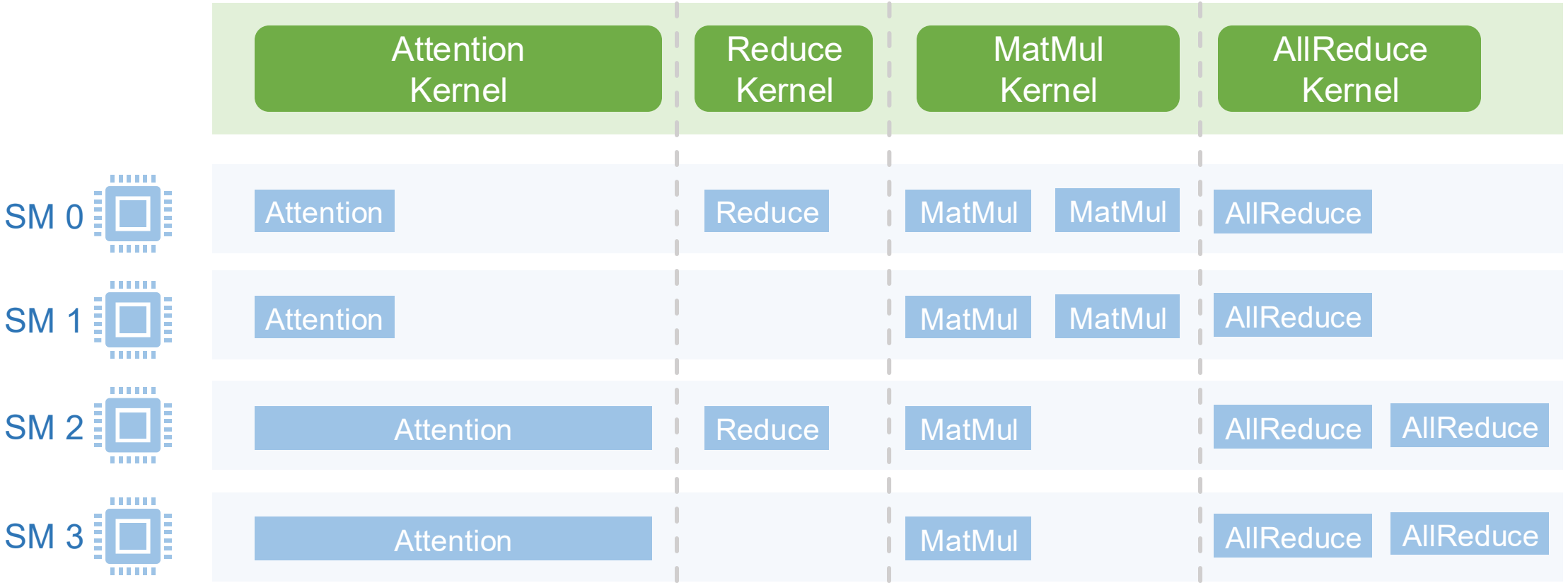
Hardware Architecture



# Existing Kernel-Per-Operator Approach



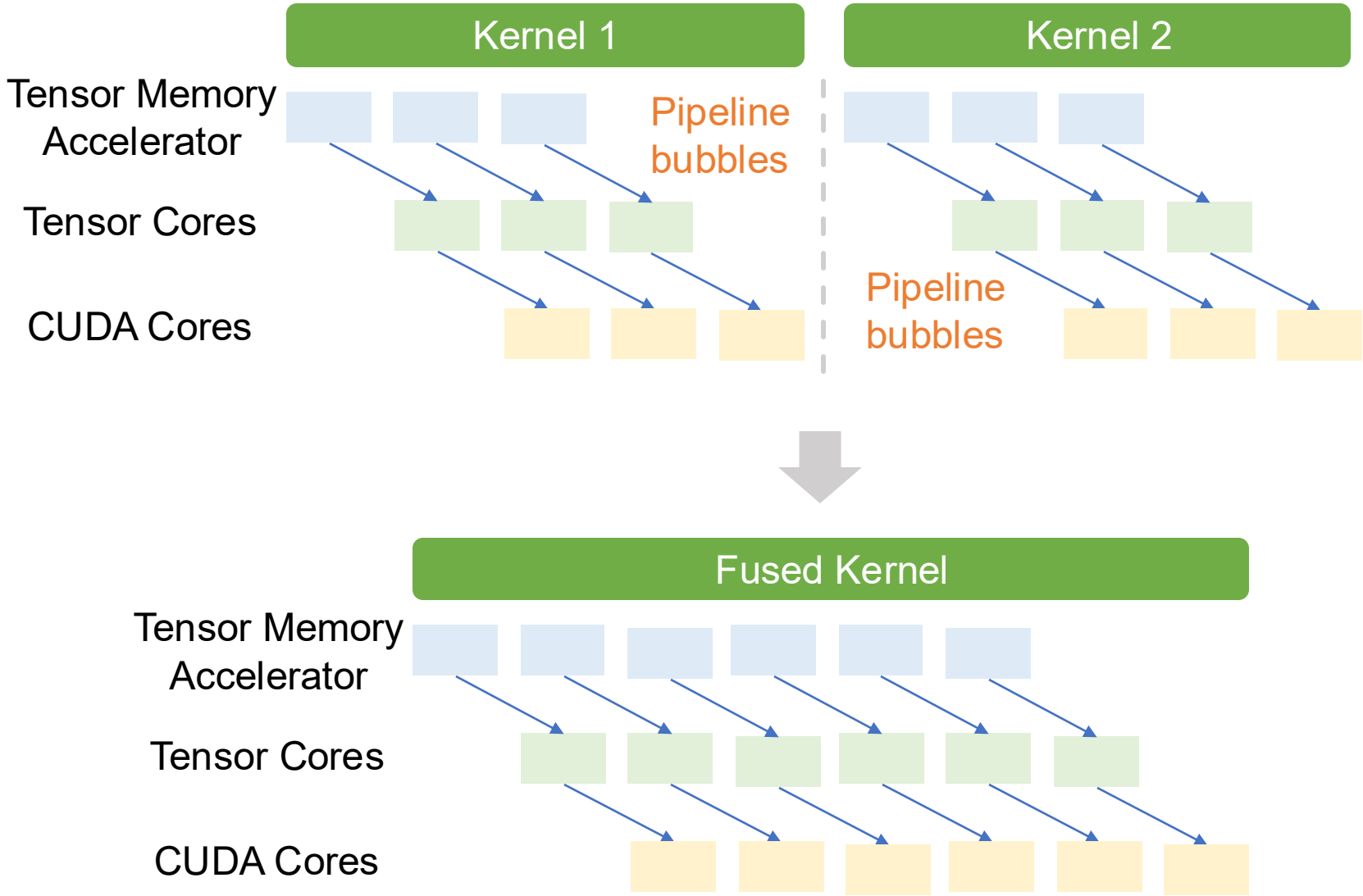
# Existing Kernel-Per-Operator Approach



# Limitations

**No Inter-Layer Pipelining**

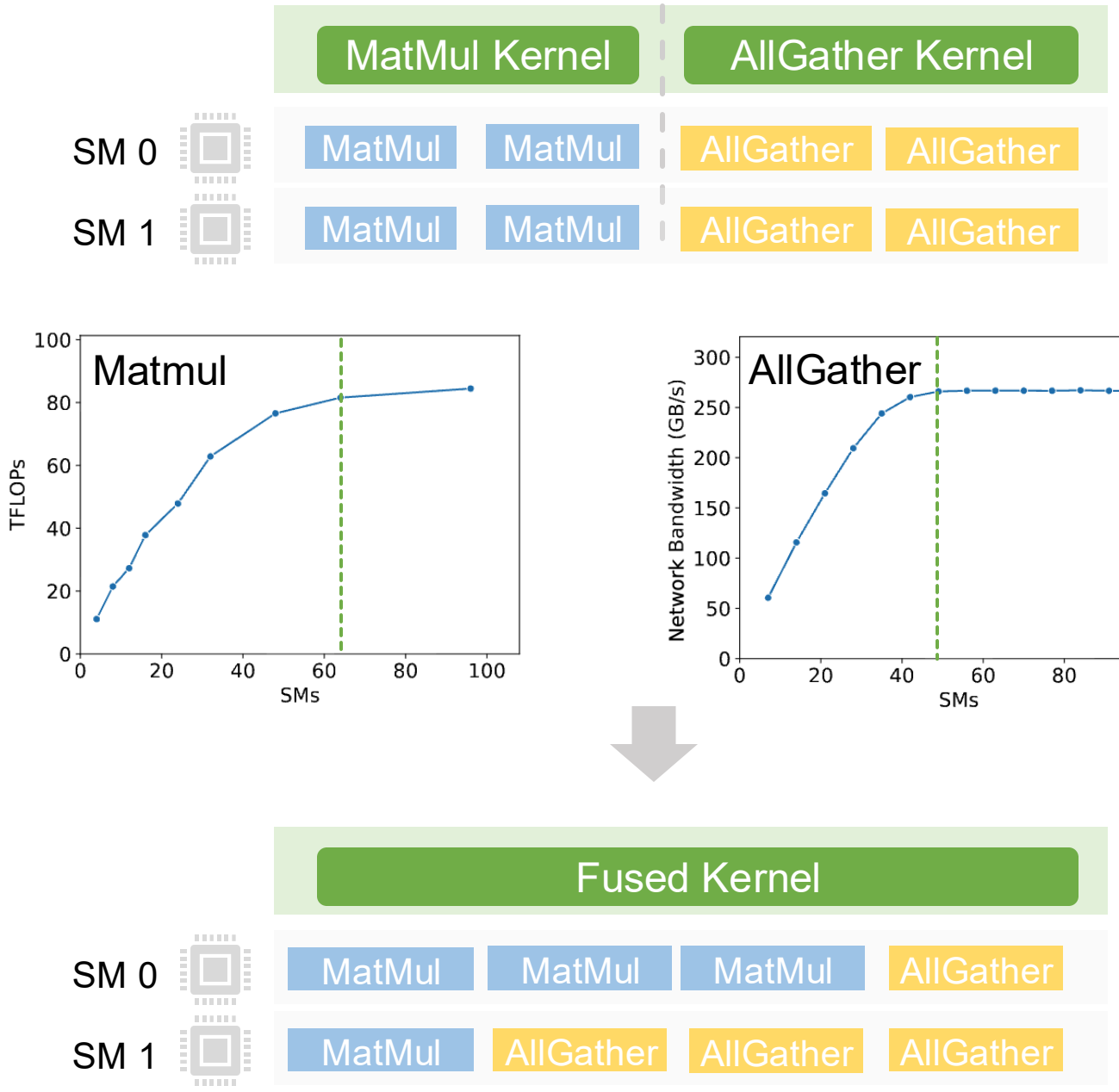
Kernel barriers prevent inter-layer pipelining



# Limitations

**No Inter-Layer Pipelining**
Kernel barriers prevent inter-layer pipelining

**No Overlapping**
Coarse-grained dependency prevents comp. & comm. overlap



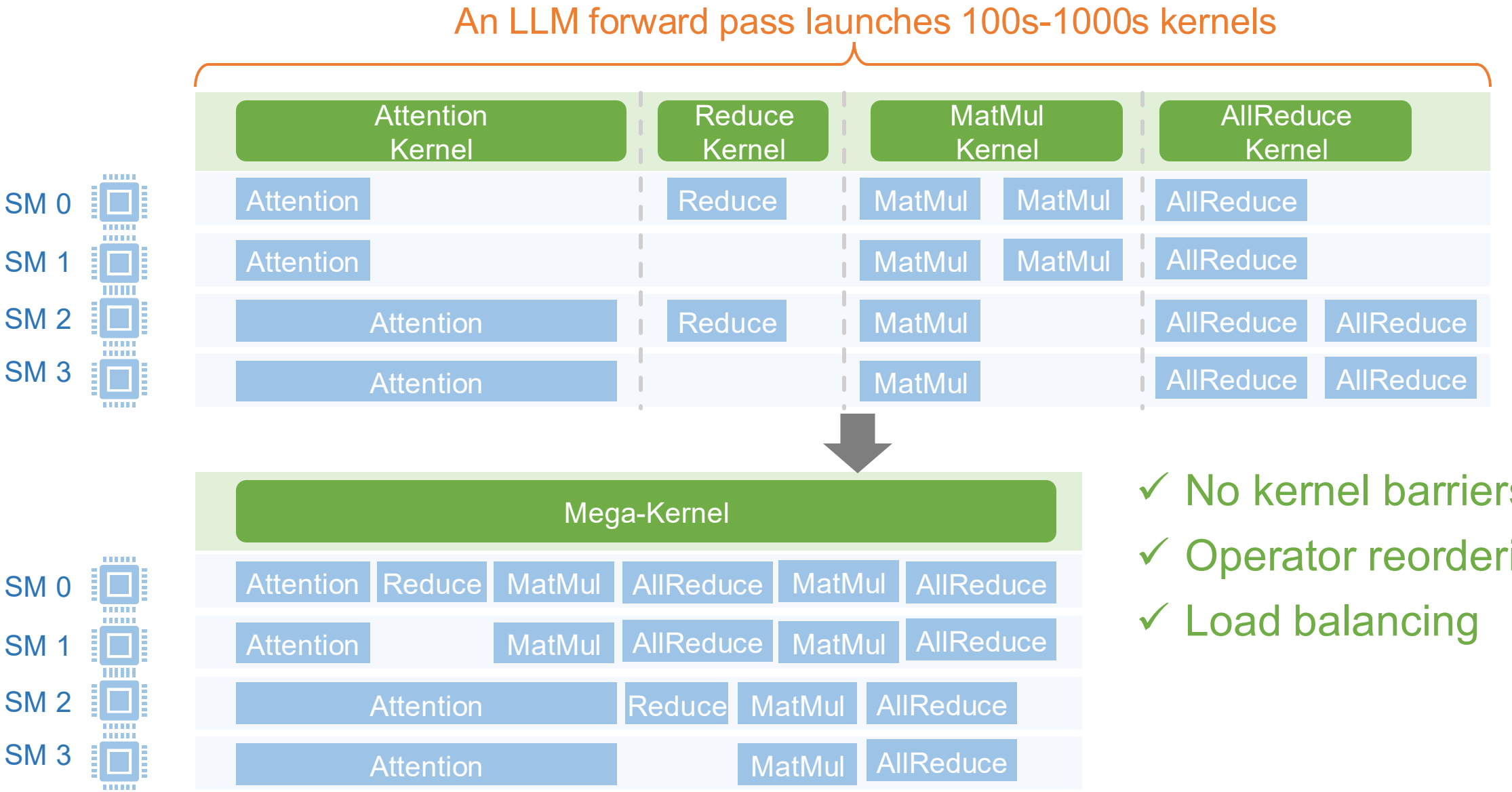
# Limitations

**No Inter-Layer Pipelining**
Kernel barriers prevent inter-layer pipelining

**No Overlapping**
Coarse-grained dependency prevents comp. & comm. overlap

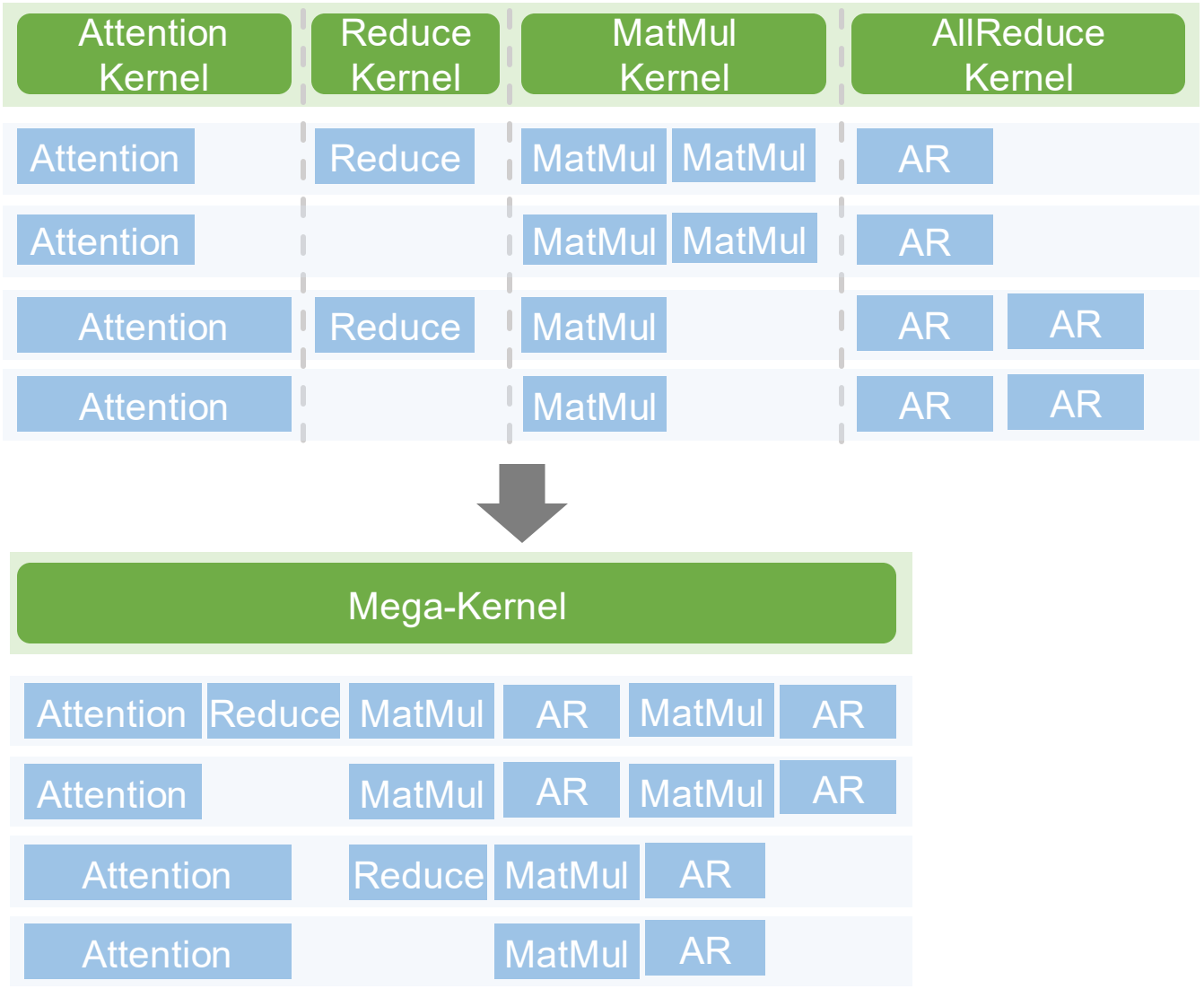
**Limited Dynamism**
Rely on CUDA graphs to reduce kernel launch overhead

# Kernel-Per-Operator v.s. Mega-Kernel




- ✓ No kernel barriers
- ✓ Operator reordering
- ✓ Load balancing

# Advantages of Mega-Kernel




**Inter-Layer Pipelining**
All layers are fused in the same mega-kernel

**Overlapping Comp/Comm**
Fine-grained dependency + operator reordering

**Dynamic Workloads**
No need for CUDA graphs + load balancing

# Key Challenges

1. How to manage dependency? → Task Graph

   No kernel barriers in mega-kernel

2. How to handle dynamism? → In-Kernel Parallel Runtime

   Continuous batching, prefill/decode,
   paged/radix attention, speculative decoding

3. How to optimize performance? → Mirage Superoptimizer*

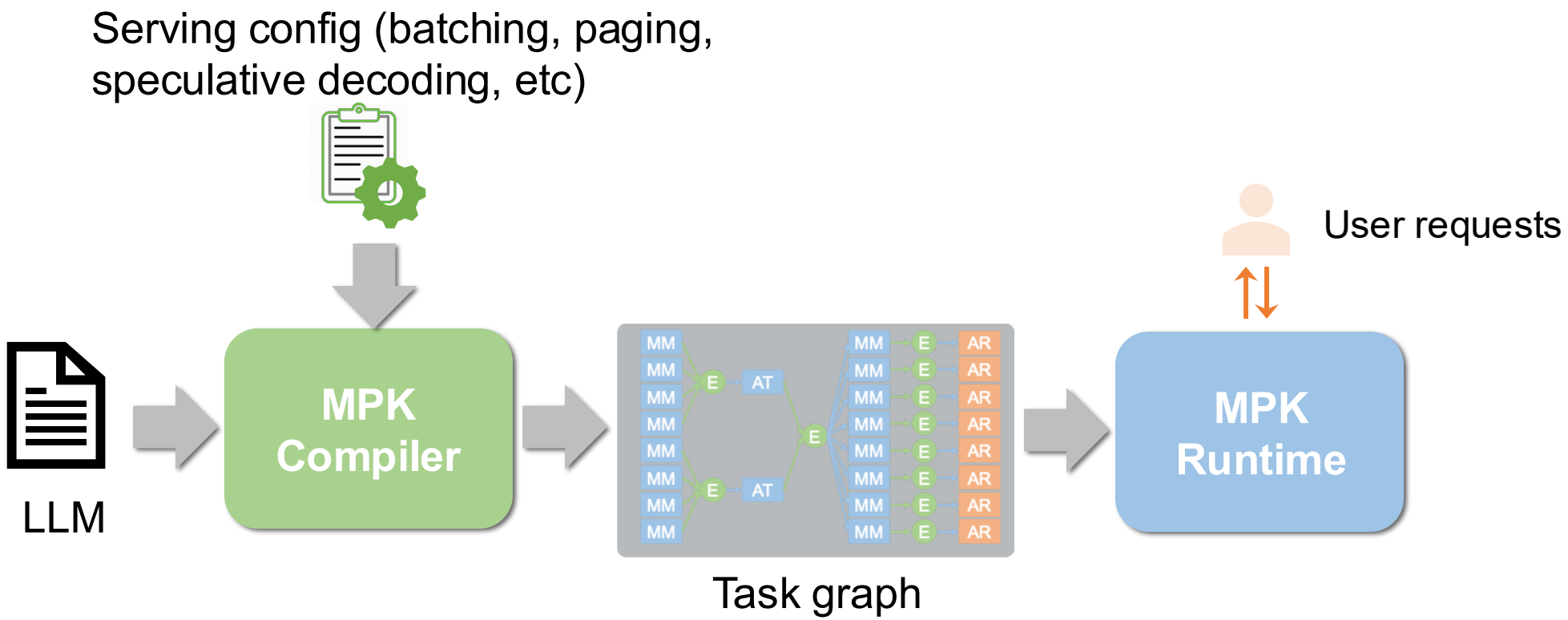
   Existing compilers target individual kernels

* Mirage: A Multi-Level Superoptimizer for Tensor Programs. OSDI'25.

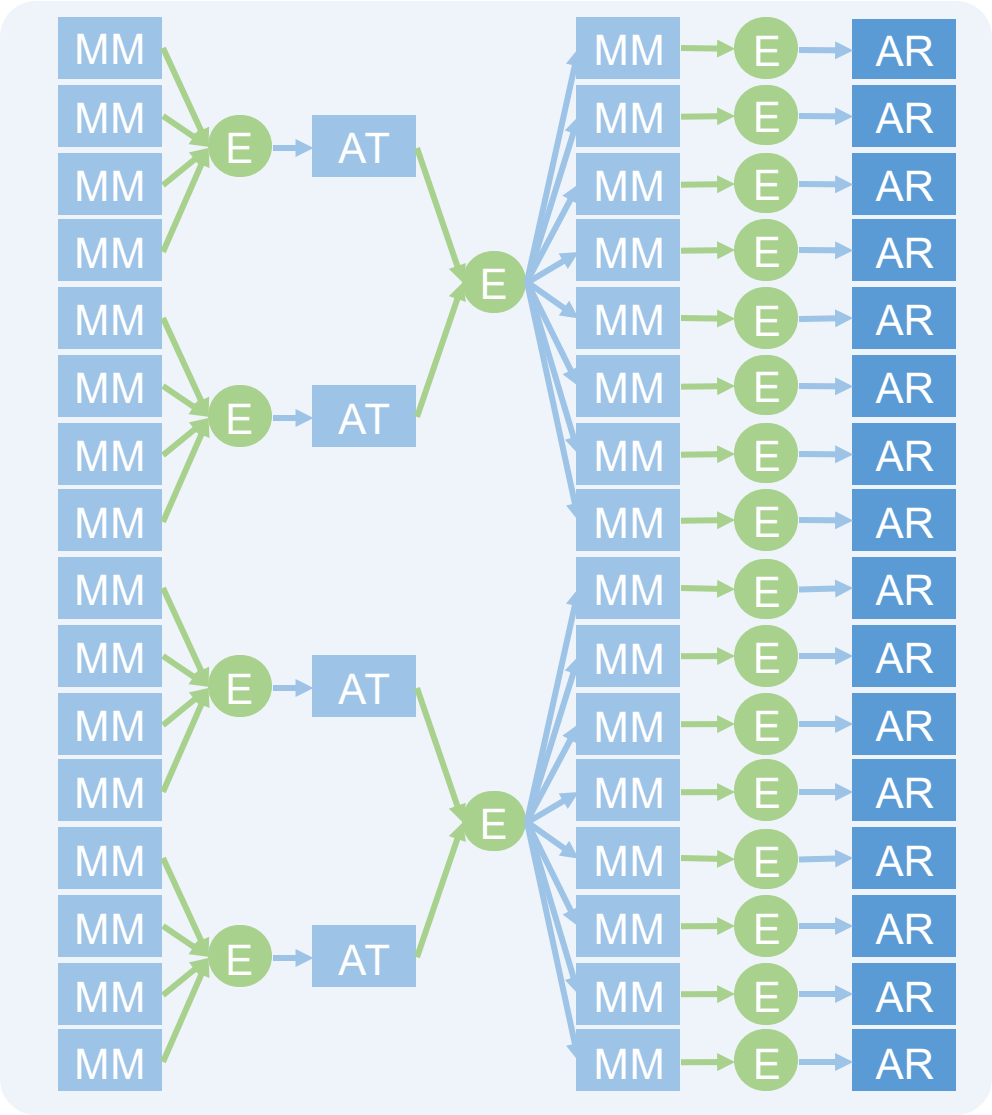
# Mirage Persistent Kernel: Compiling LLM Serving into a Mega-Kernel


- **Low engineering effort**: a few dozen lines of Python code to mega-kernelize an LLM
- **Better performance**: outperform existing systems by 1.2-6.7x
- **Day-0 support for new models:** do not rely on manual implementation

# MPK Overview

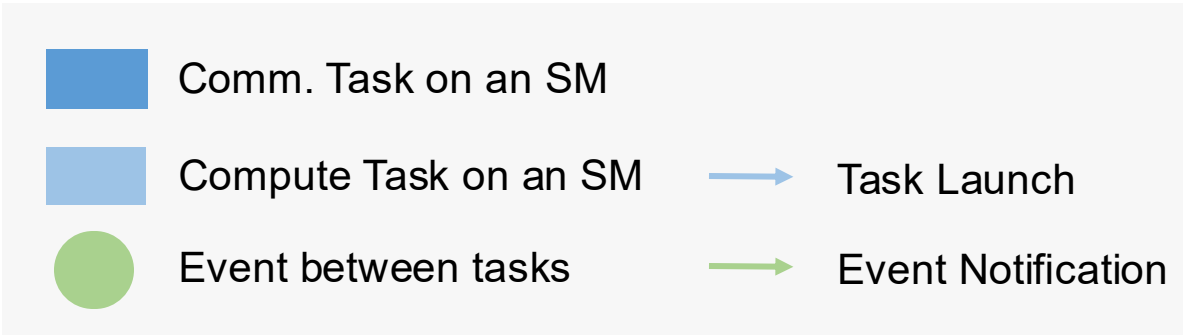


# Task Graph




Interleave tasks and events

- task: a unit of workload on one SM
- event: synchronization among tasks
- task→event: notify event once task is done
- event→task: launch task once event is triggered

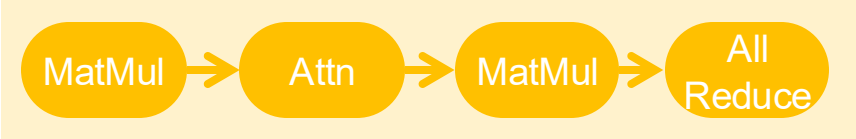


# Task Graph vs. CUDA Graph

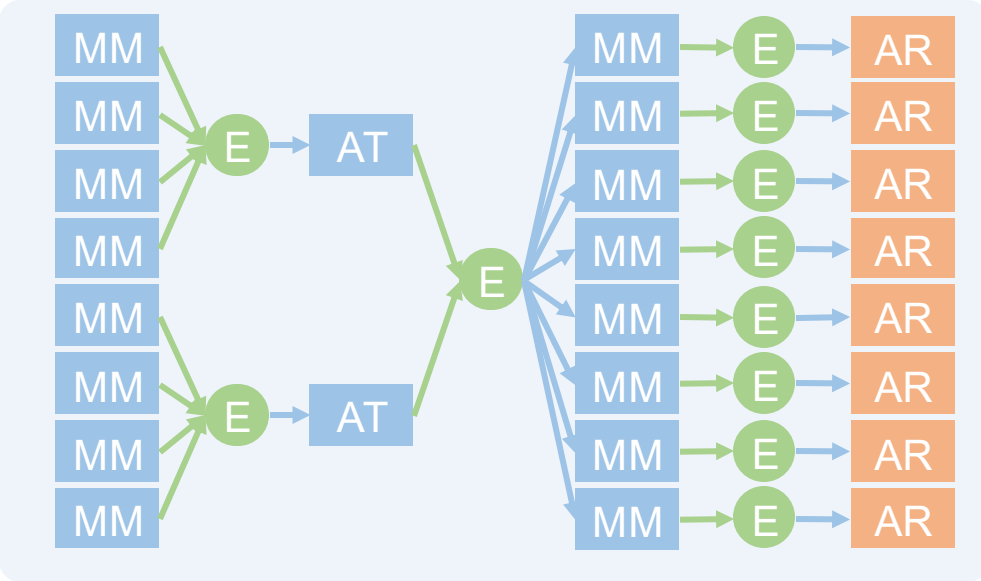
Task graph is a “lower-level” CUDA graph

- Capture sub-kernel dependency
- Static, immutable
- Constructed once and replayed many times

CUDA Graph: nodes are kernels on GPUs

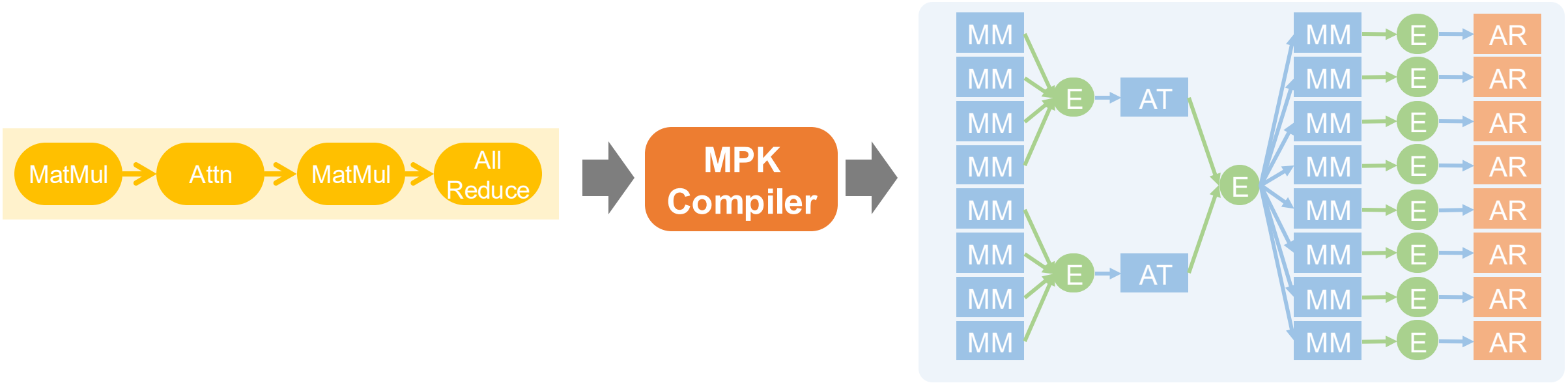



Task Graph: nodes are tasks on SMs

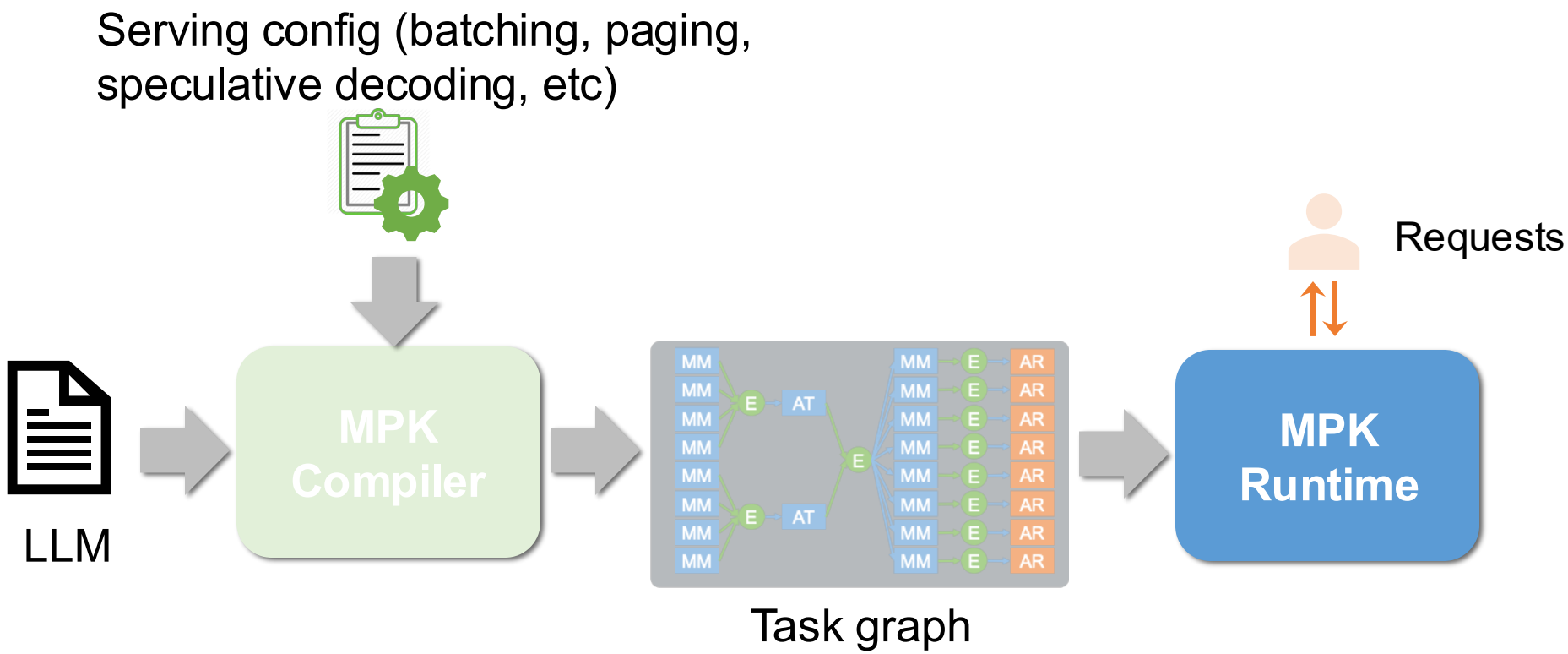


# The MPK Compiler

- Optimize operator-to-task decomposition based on available SMs
- Add synchronization events to capture precise task dependencies
- Generate high-performance CUDA implementation for each task



# MPK Overview

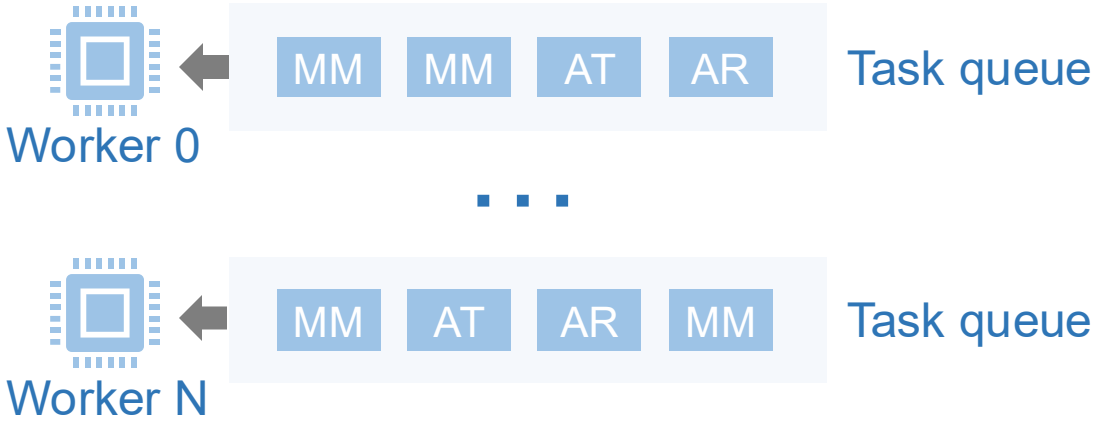


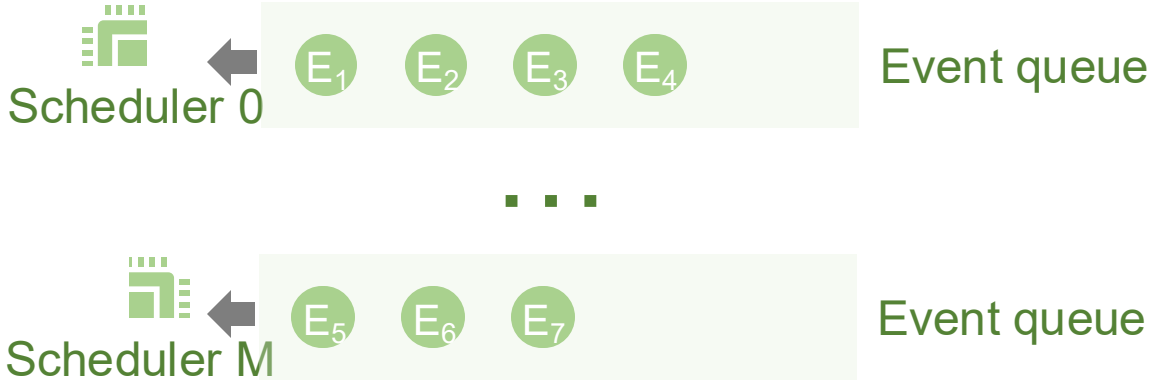
# The MPK Runtime

Each scheduler runs on one warp

Each worker runs on one SM

# Task-Based Parallel Runtime (Realm within a GPU Kernel)



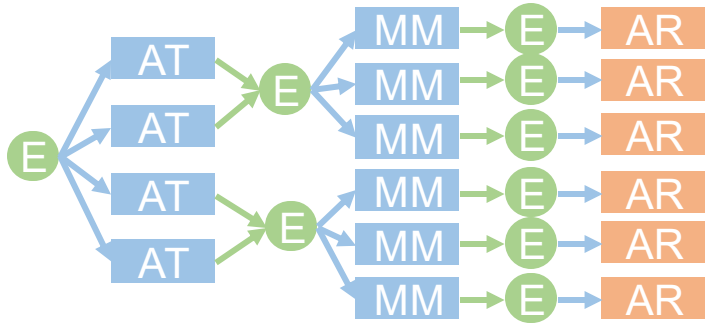

Repeatedly

1. Fetch a task from its queue
2. Execute the task
3. Trigger the completion event

Repeatedly

1. Dequeue fully triggered event
2. Launch all tasks depending on the event

# Event-Driven Execution




Scheduler
on a warp

Scheduler
on a warp

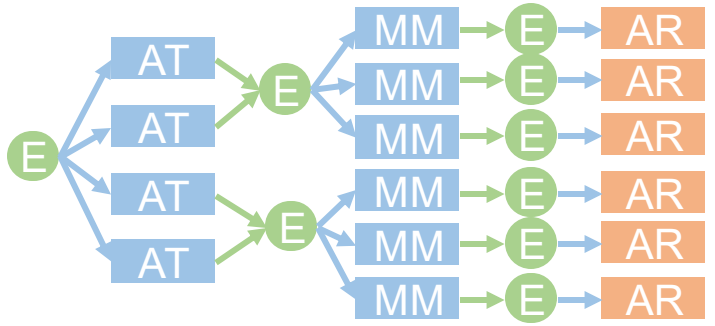
Worker on
an SM

Worker on
an SM

Worker on
an SM

Worker on
an SM

# Event-Driven Execution




Scheduler on a warp

Scheduler on a warp

E

Worker on an SM

Worker on an SM

Worker on an SM

Worker on an SM

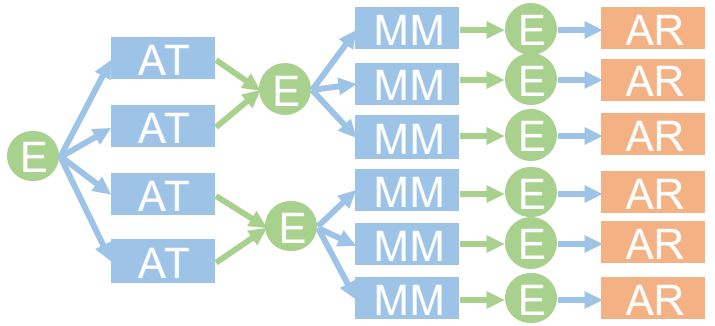
Attention

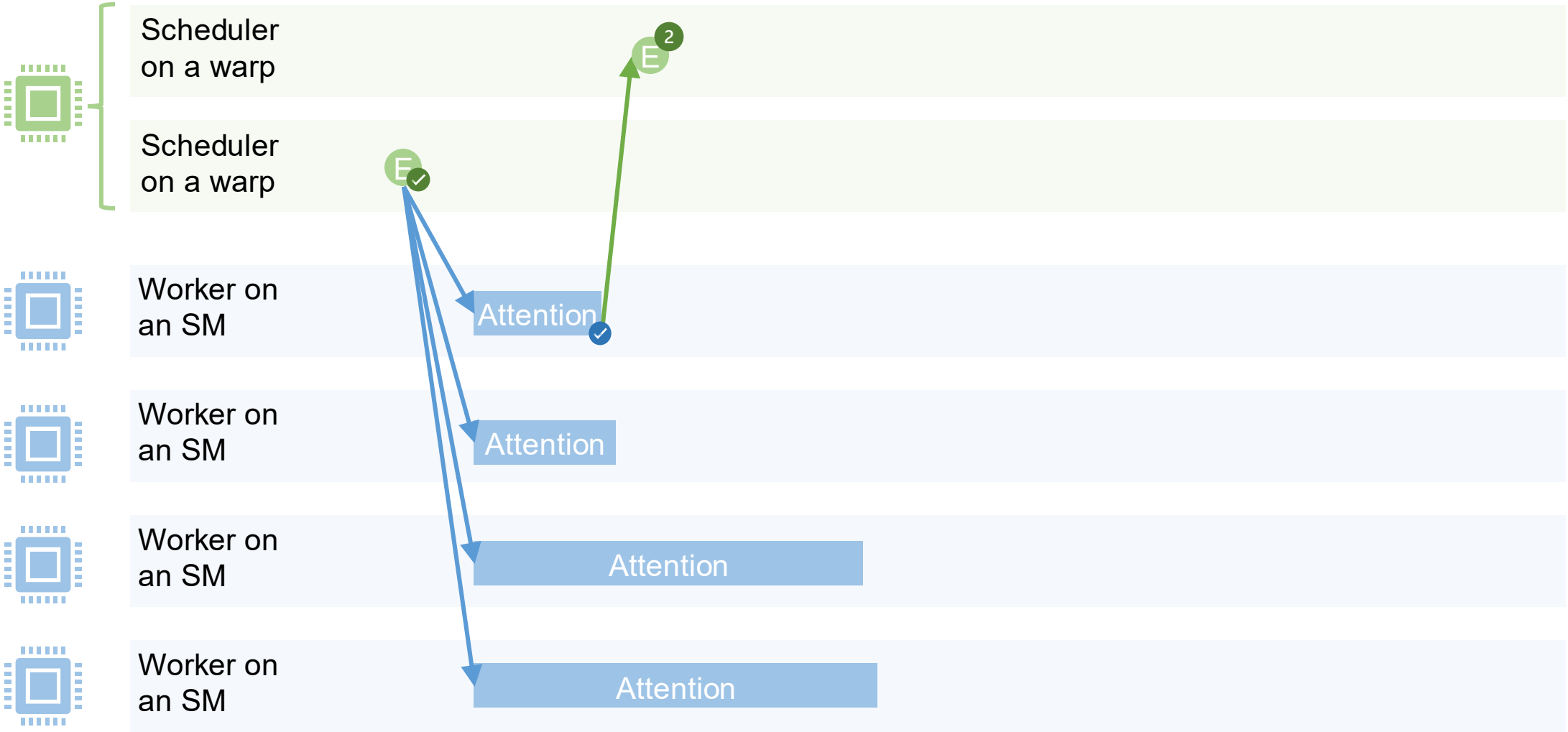
Attention

Attention

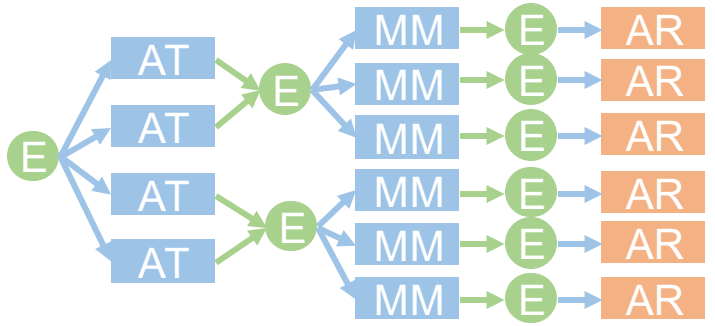
Attention

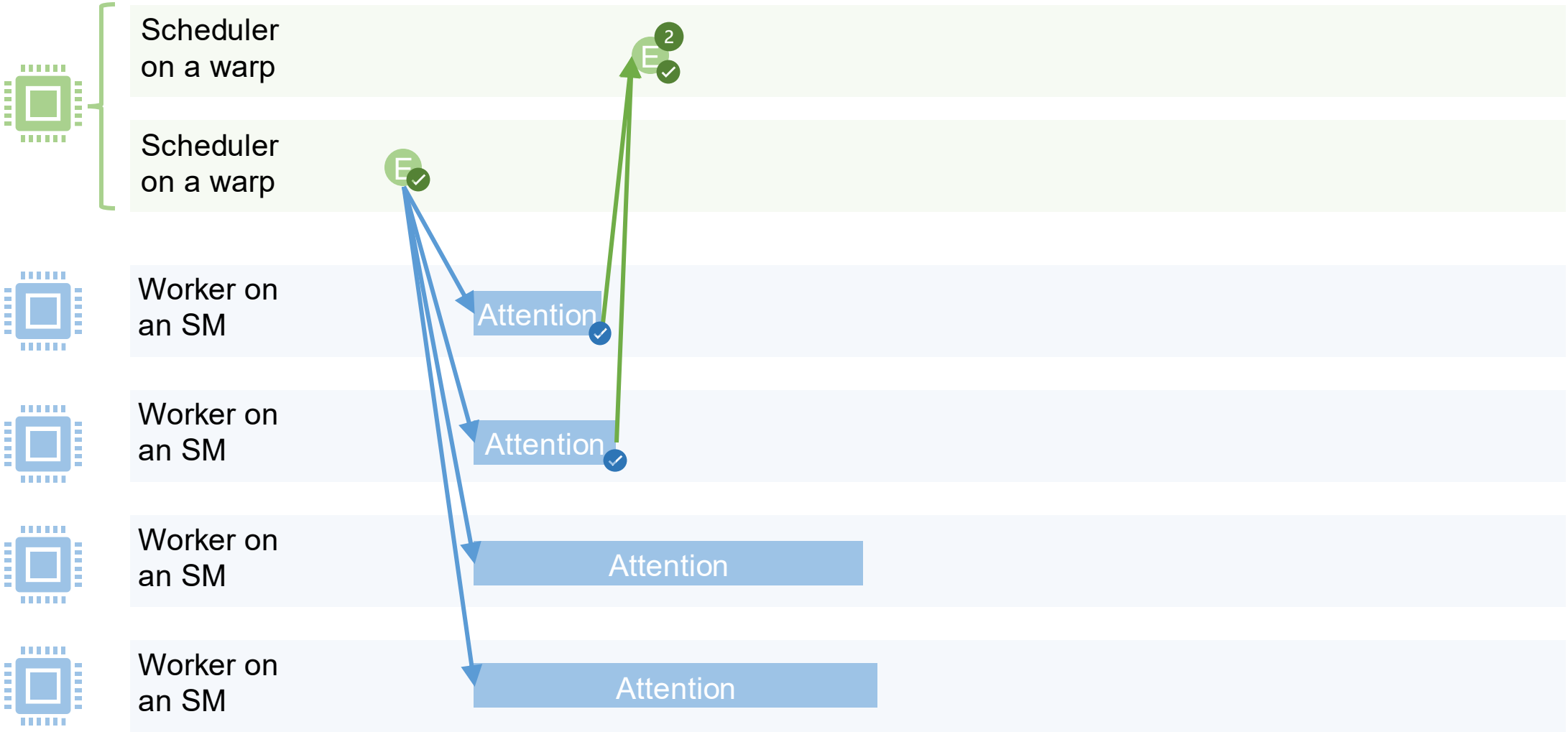
# Event-Driven Execution



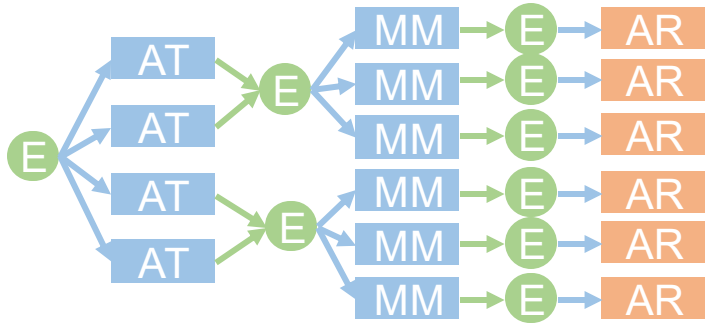



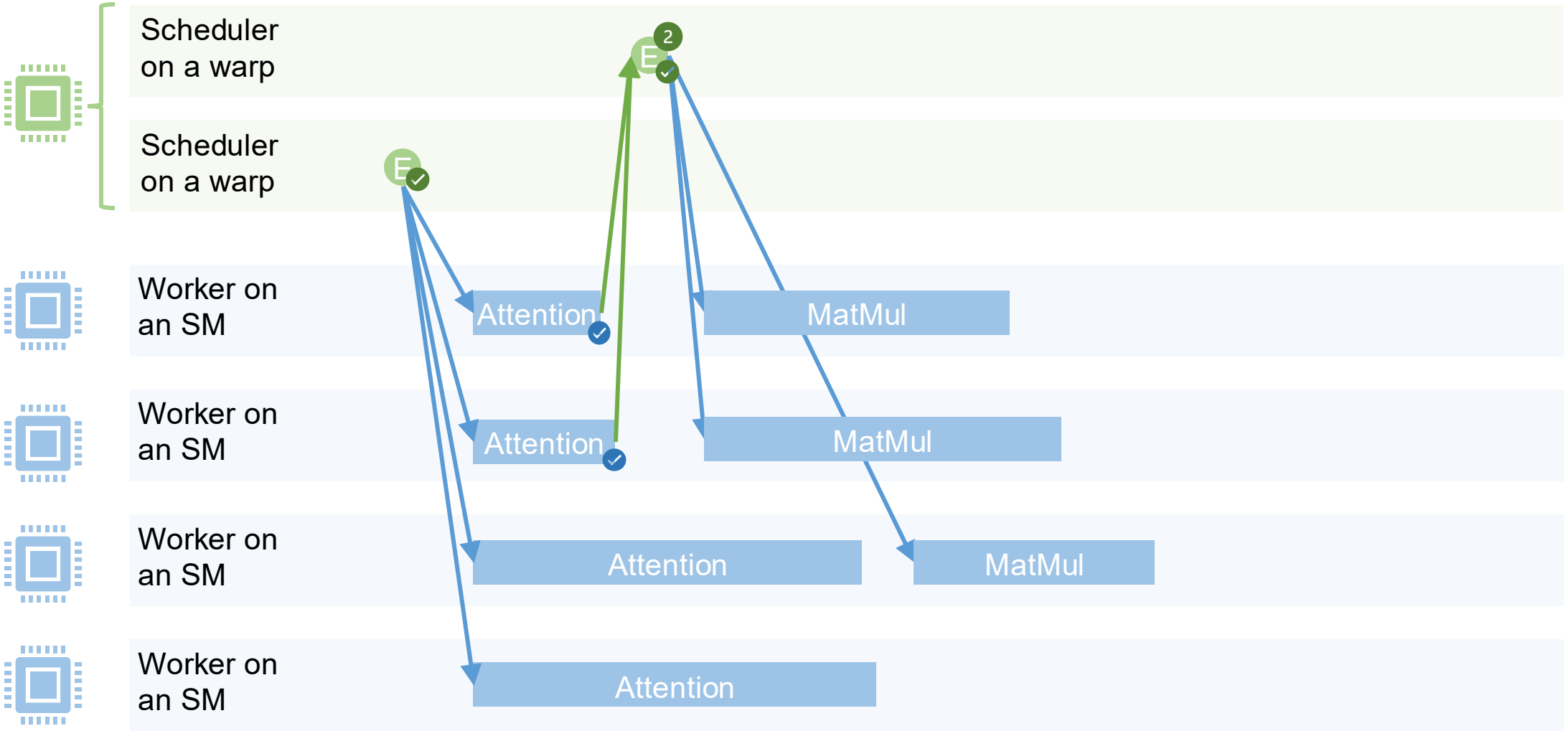
# Event-Driven Execution



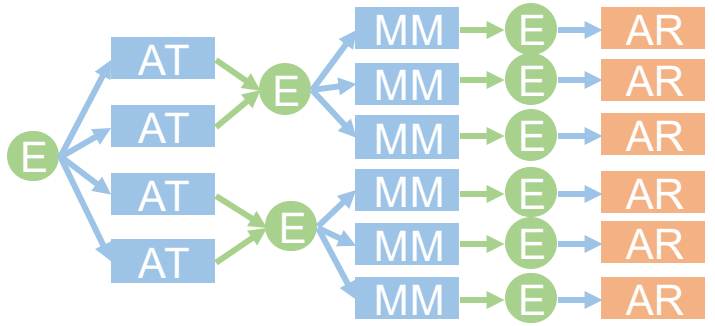



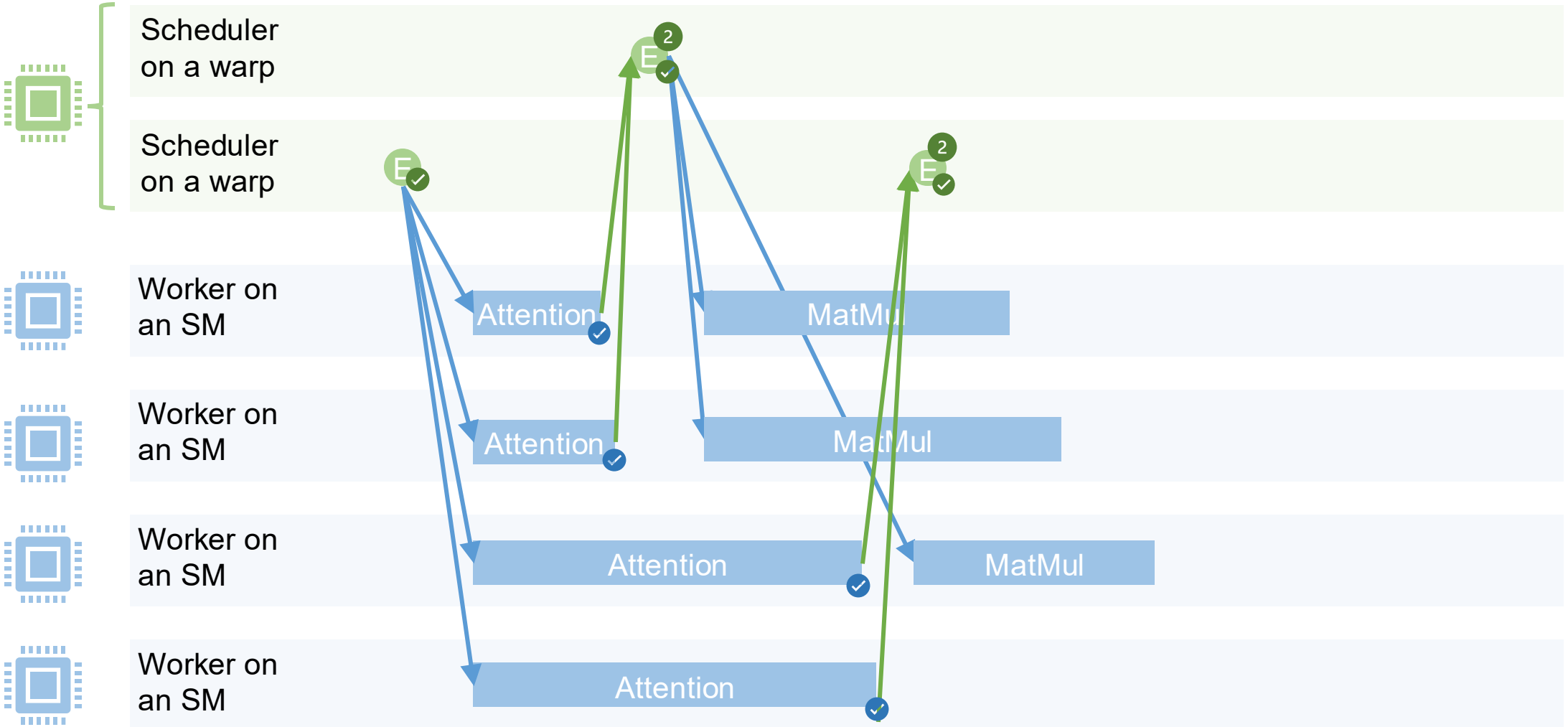
# Event-Driven Execution



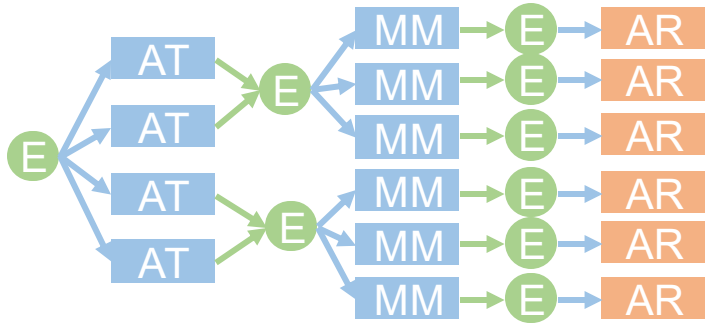



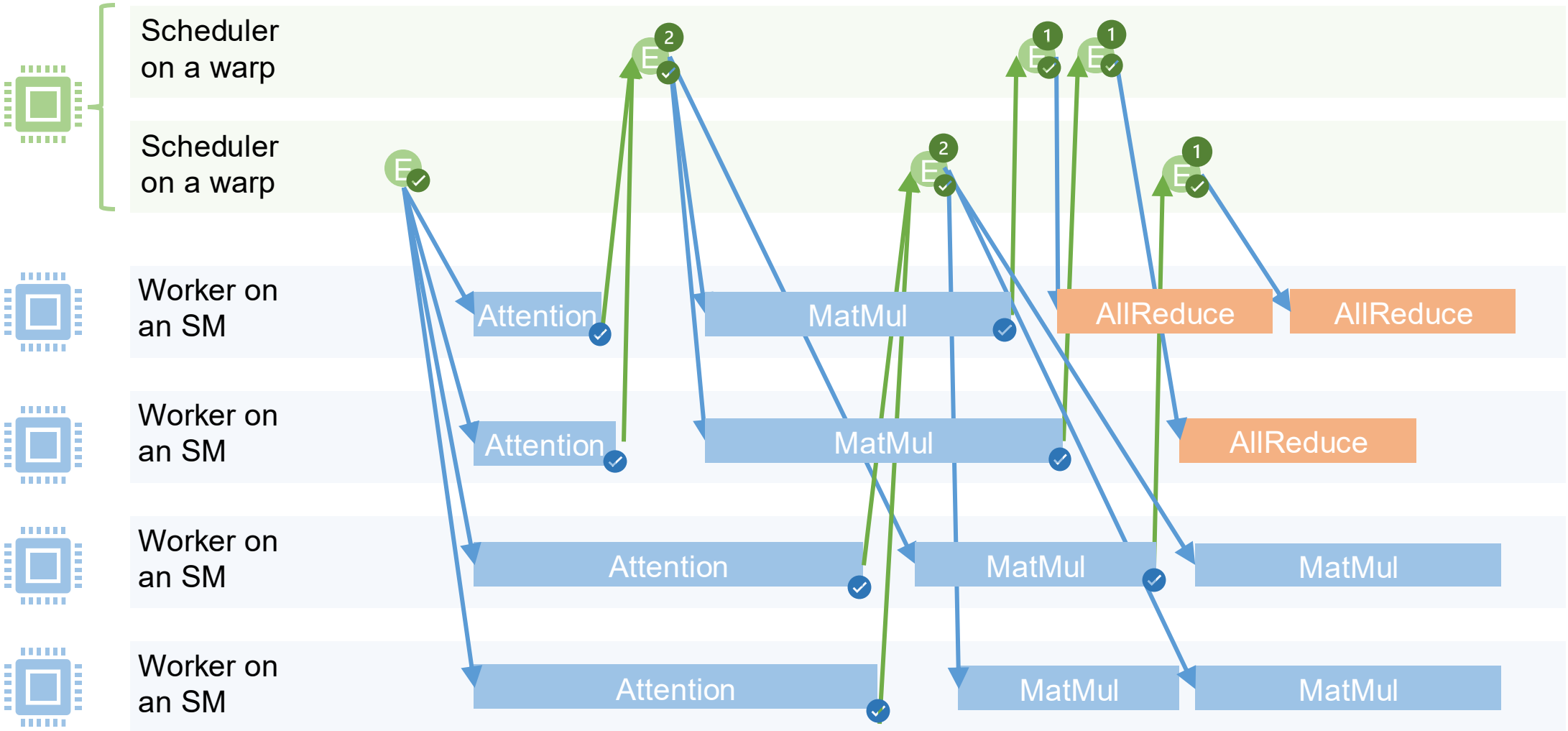
# Event-Driven Execution



# Event-Driven Execution



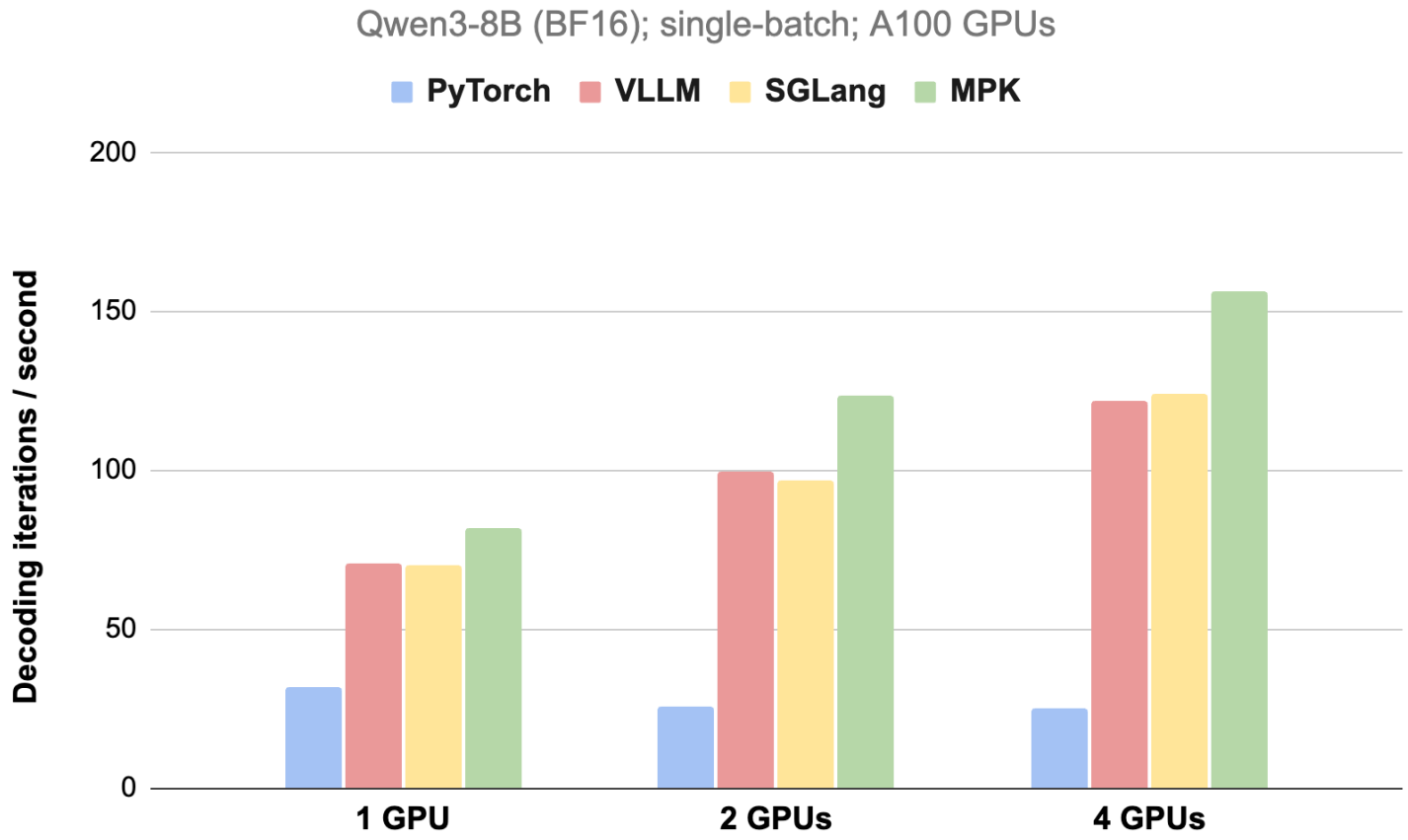
# Techniques to Reduce Task Launch Overheads

- **Lightweight workers and schedulers**
  - Task & event queues: circular buffers on device memory
  - Event notification, task enqueue/dequeue: atomic operations
- **Decentralized scheduling**
  - Schedulers assign tasks using only local information
- **Hybrid task launch**
  - Ahead-of-time: launched before event trigger to reduce latency
  - Just-in-time: launched after event trigger to balance load

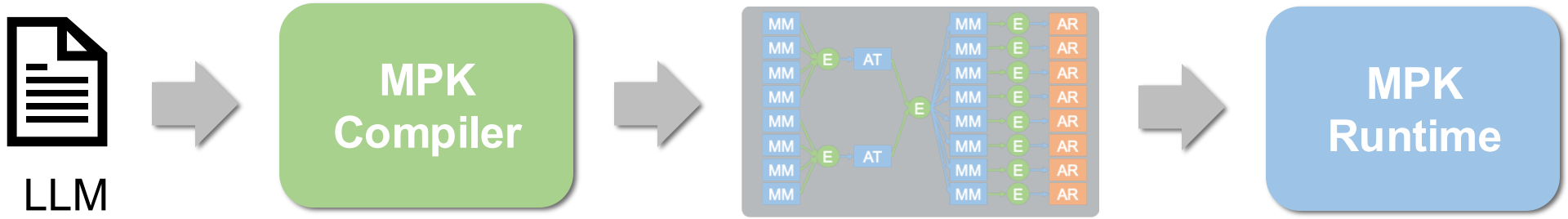
- Task launch overhead 2-3μs
  - Limited hardware support

# Pushing LLM Inference Latency Towards HW Limits

- Reducing Qwen3-8B per-token latency from 14.5ms to 12.5ms
- Approaching theoretical bound of 10ms



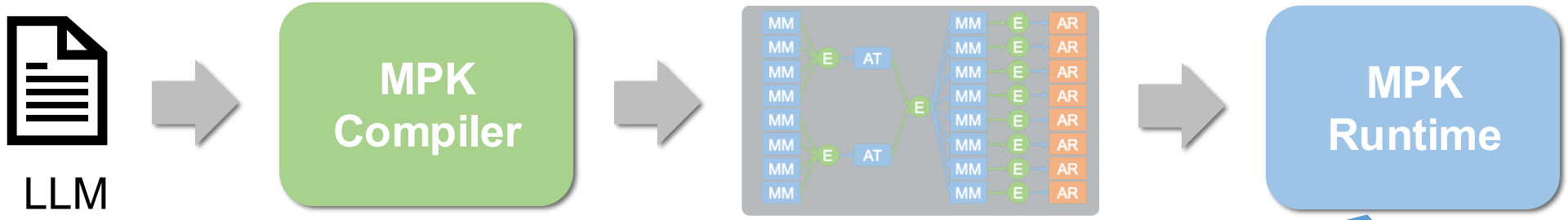
# Open Questions




**How to decompose layers into tasks?**

- Currently rely on heuristics + profiling-based tuning
- Optimized for A100, H100, B200
- Users want portability across diverse GPUs
- Need more automated methods

# Open Questions




**How to schedule tasks across workers?**

Round-Robin

✅ no synchronization with workers

❌ load imbalance

Dynamic Scheduling

✅ adapt to runtime load

❌ coordination overhead