FlashInfer

Efficient and **Customizable Attention Engine** for LLM Inference **Serving**

MLSys25 Best Paper

Zihao Ye[1 2], Lequn Chen[3], Ruihang Lai[4], Wuwei Lin[2],
Yineng Zhang, Stephanie Wang[1], **Tianqi Chen**[2 4], Baris Kasikci[1],
Vinod Grover[2], Arvind Krishnamurthy[1], **Luis Ceze**[1 2]

[1]University of Washington, [2]NVIDIA
[3]Perplexity AI, [4]CMU

Zevin @ Reading Group 2025/6/10

FlashInfer

Efficient and **Customizable** **Attention Engine** for LLM Inference **Serving**

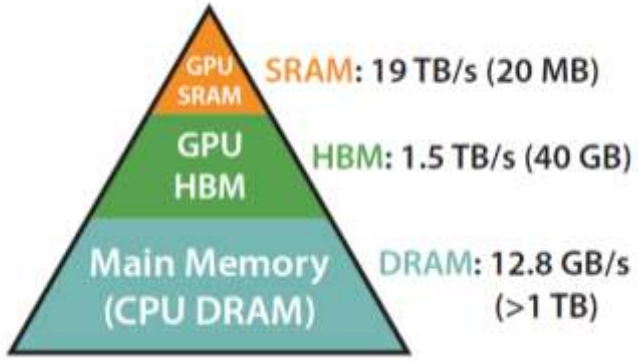
We already have
FlashAttention, PagedAttention, RadixAttention, SpecInfer,
StreamingLLM, Quest, DuoAttention,
FlexAttention, ...

Why do we still need a new attention engine?

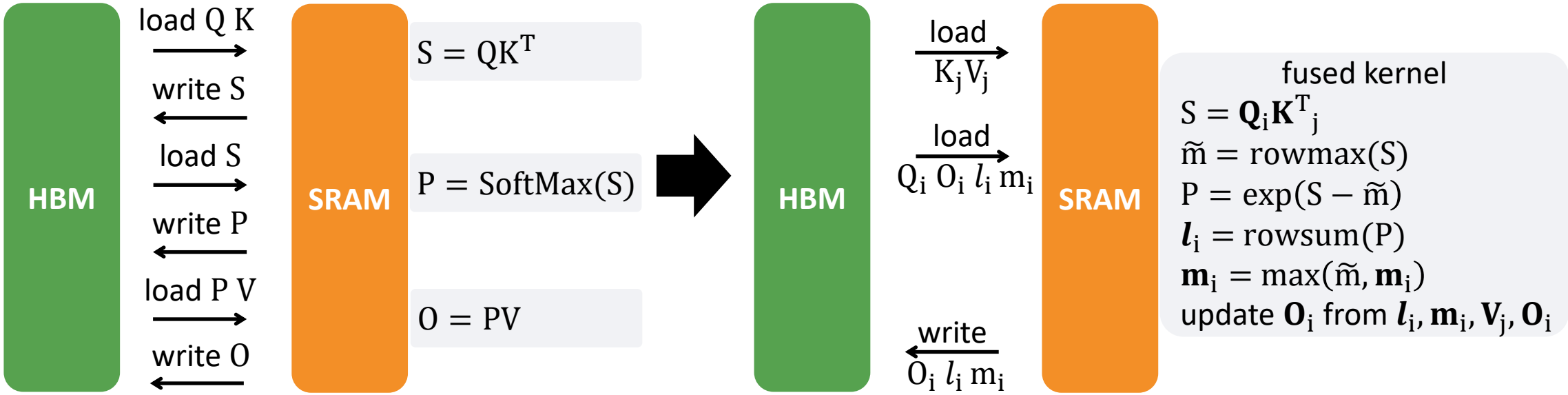
Background

❑FlashAttention

- ❖GPU-friendly memory access
- ❖FlashAttention2&3
  - ➢Loop optimization
  - ➢Pipeline optimization for Hopper



Memory hierarchy (A100-40GB)

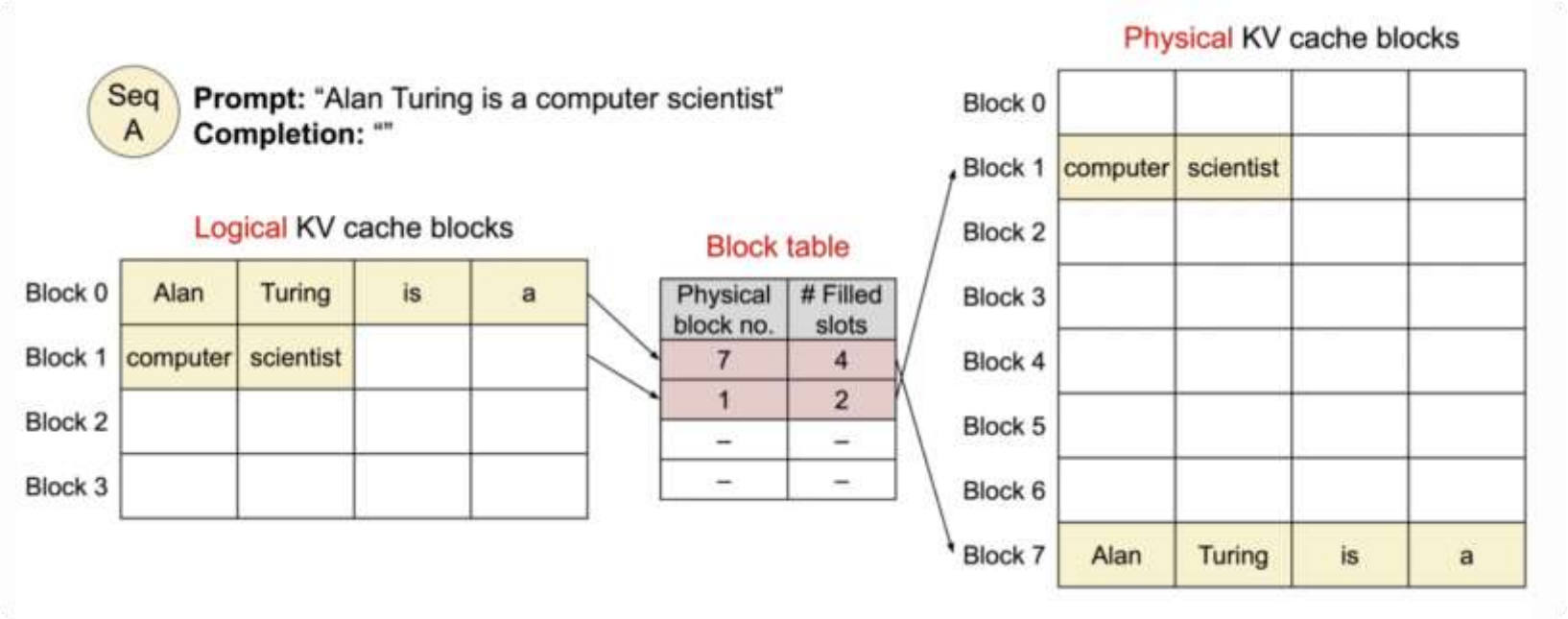


Standard attention **FlashAttention**

Background

❑PagedAttention (vLLM) [SOSP23]

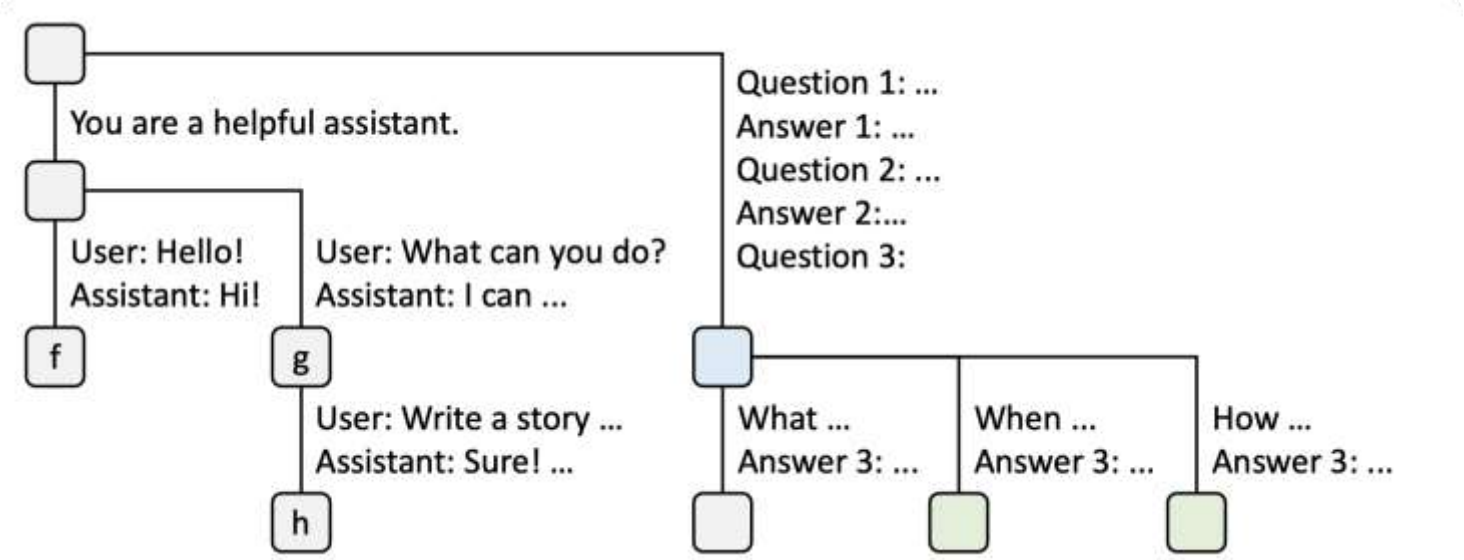
- ❖Paged KV cache management for variable sequence lengths
- ❖page_size=16 for tradeoff of throughput and fragmentation



Background

❑RadixAttention (SGLang) [NeurIPS24]

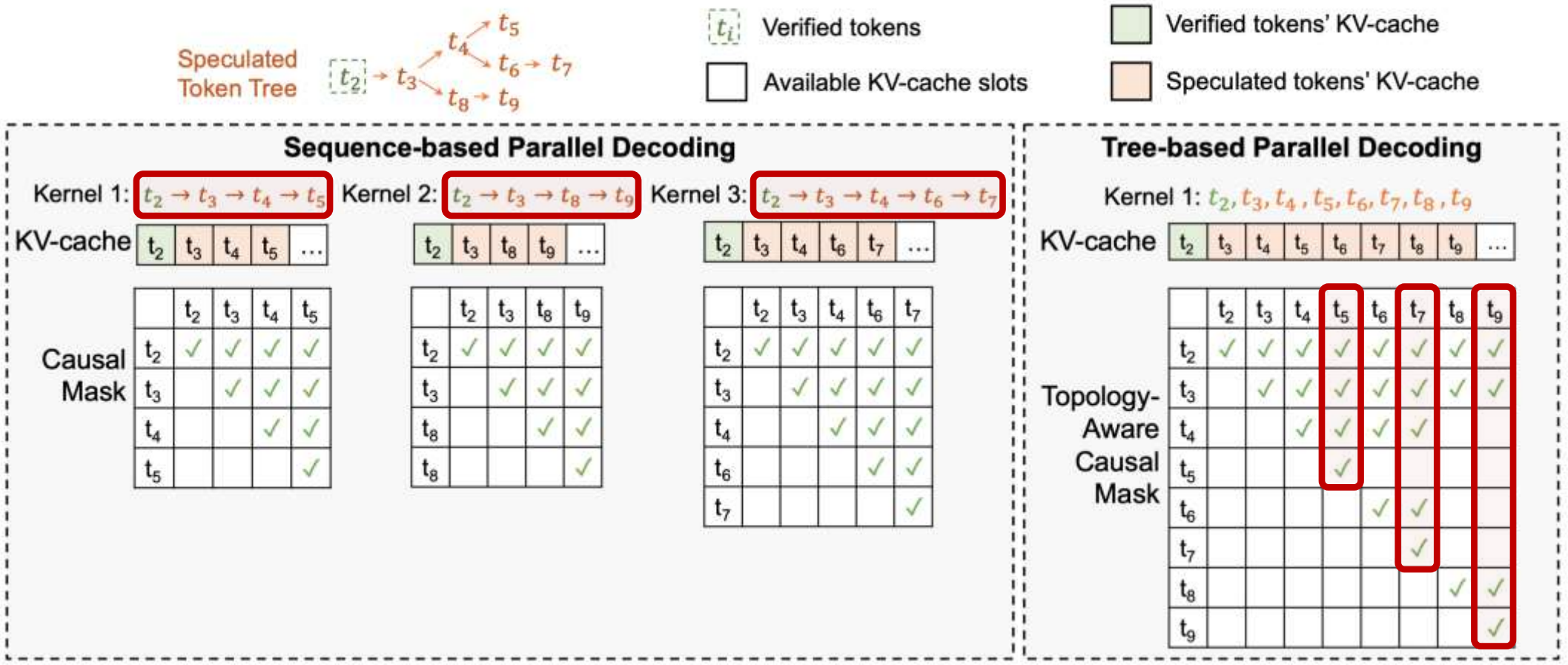
- ❖Designed for prefix-sharing
  - ➢multi-round conversation, AI agent, etc.
- ❖Shared prefix organized as a Radix tree
- ❖**page_size=1** for higher cache hit rate



Background

❑Speculative decoding (SpecInfer) [ASPLOS24]

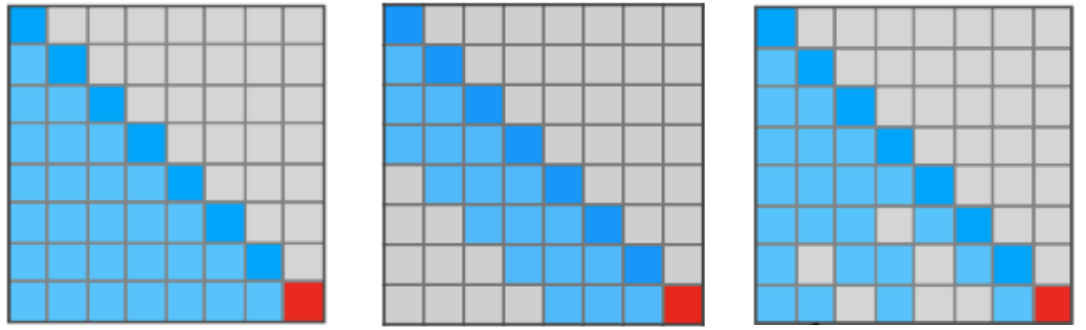
❖Tree-based parallel verification for speculative decoding



Motivation

❑New attention variants from researchers

- ❖KV cache compression:
  - ➢StreamingLLM
  - ➢Quest
- ❖Speculative decoding:
  - ➢SpecInfer
  - ➢Medusa



❑Efficient implementation in production

- ❖Requiring fusion with FlashAttention, PagedAttention, etc.

Issue 1: Hard to fuse variants with FA

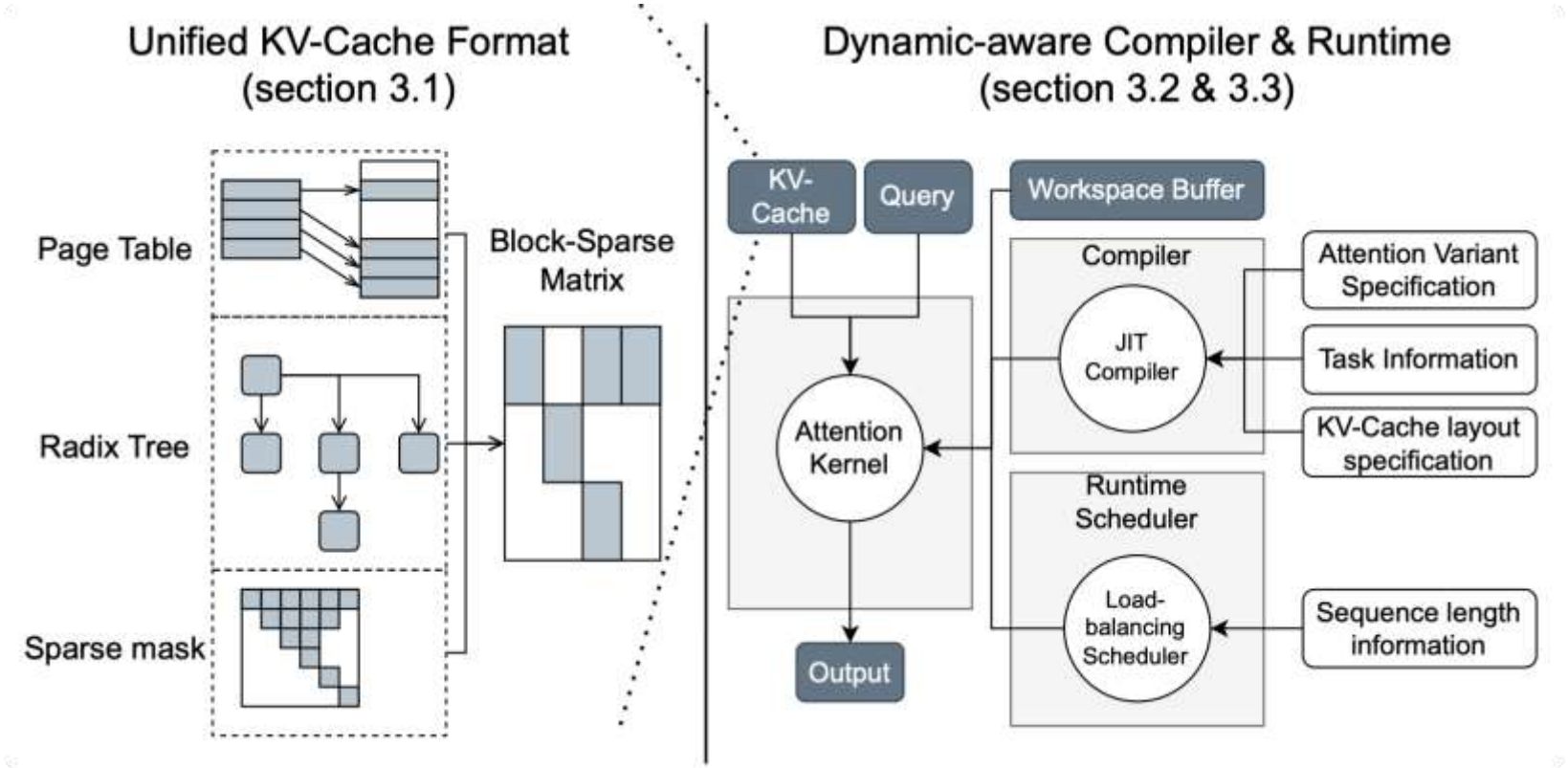
Motivation

- **FlashAttention can be further optimized**
  - For decoding (query_length = 1)
    - suboptimal tile size
    - tile size selected for prefill (64, 128)
  - For batch of different sequence length
    - low GPU utilization (some SMs left idle)

Issue 2: FlashAttention can be optimized

FlashInfer Design

❑Overview

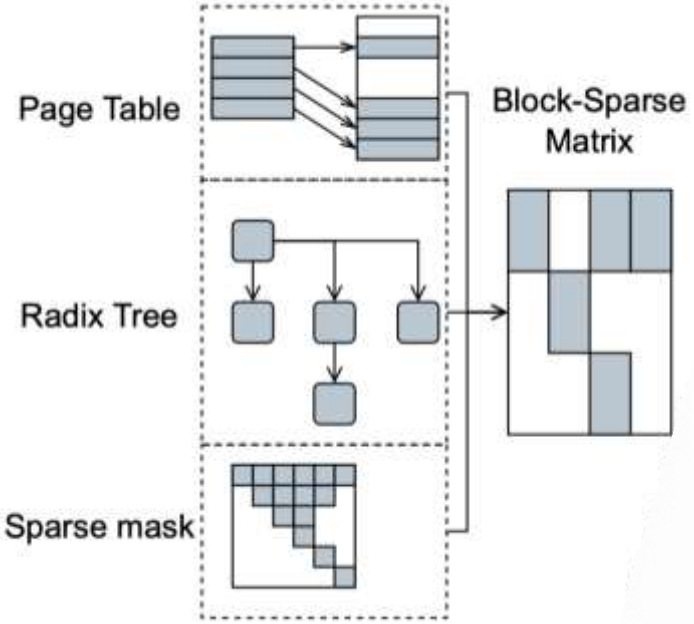


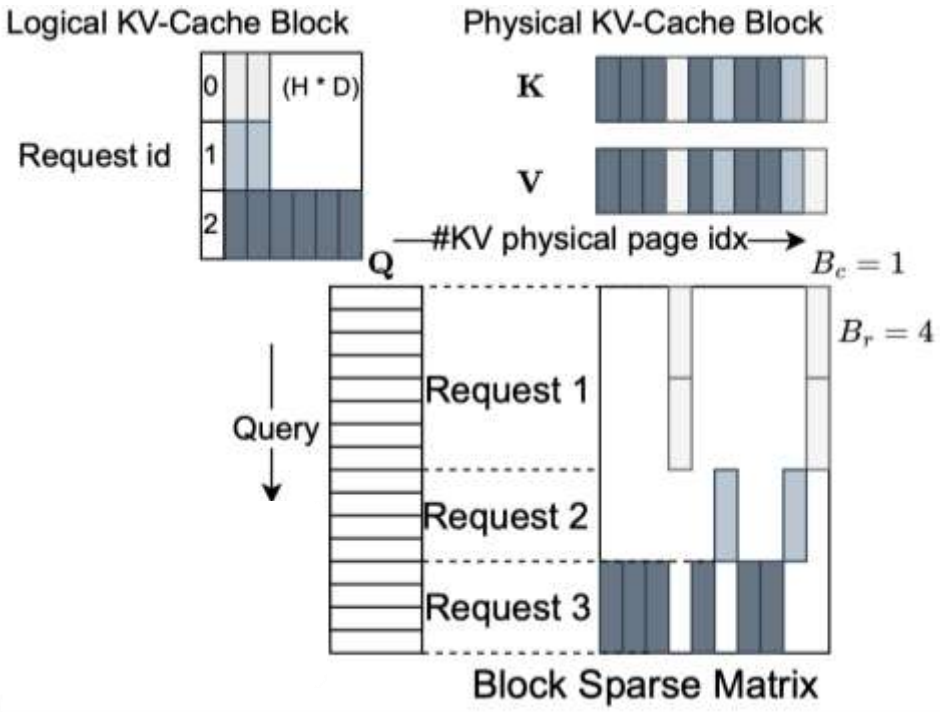
Unified KV Cache Format

❑Abstraction of Block Sparse Matrix

❖Expressing different structures

- ➢PagedAttention
- ➢RadixAttention
- ➢Masks

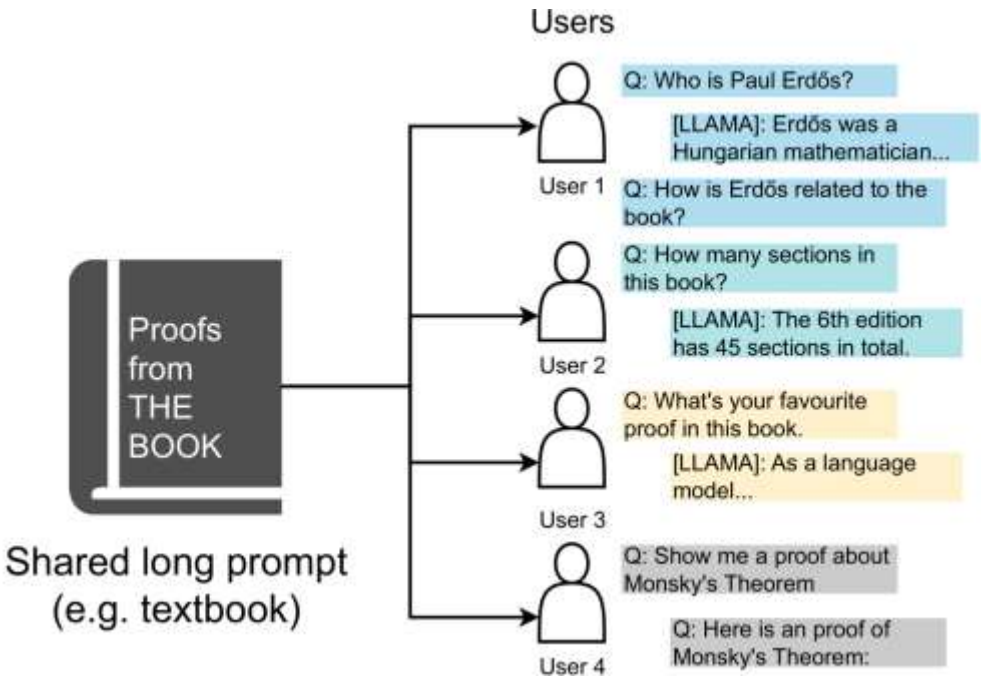


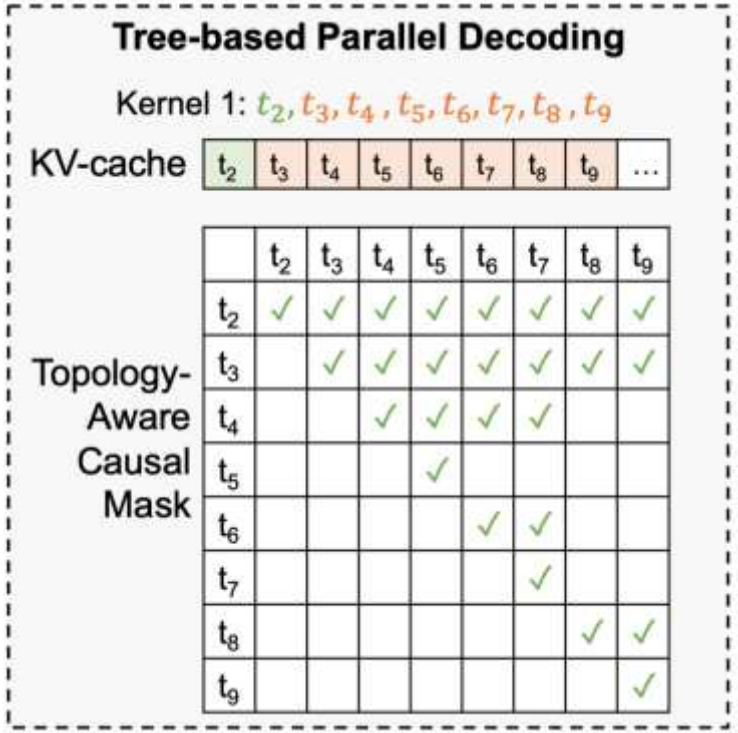


Unified KV Cache Format

❑Composable Formats for prefix sharing

- ❖Small block size for unique KV
- ❖Big block size for shared KV

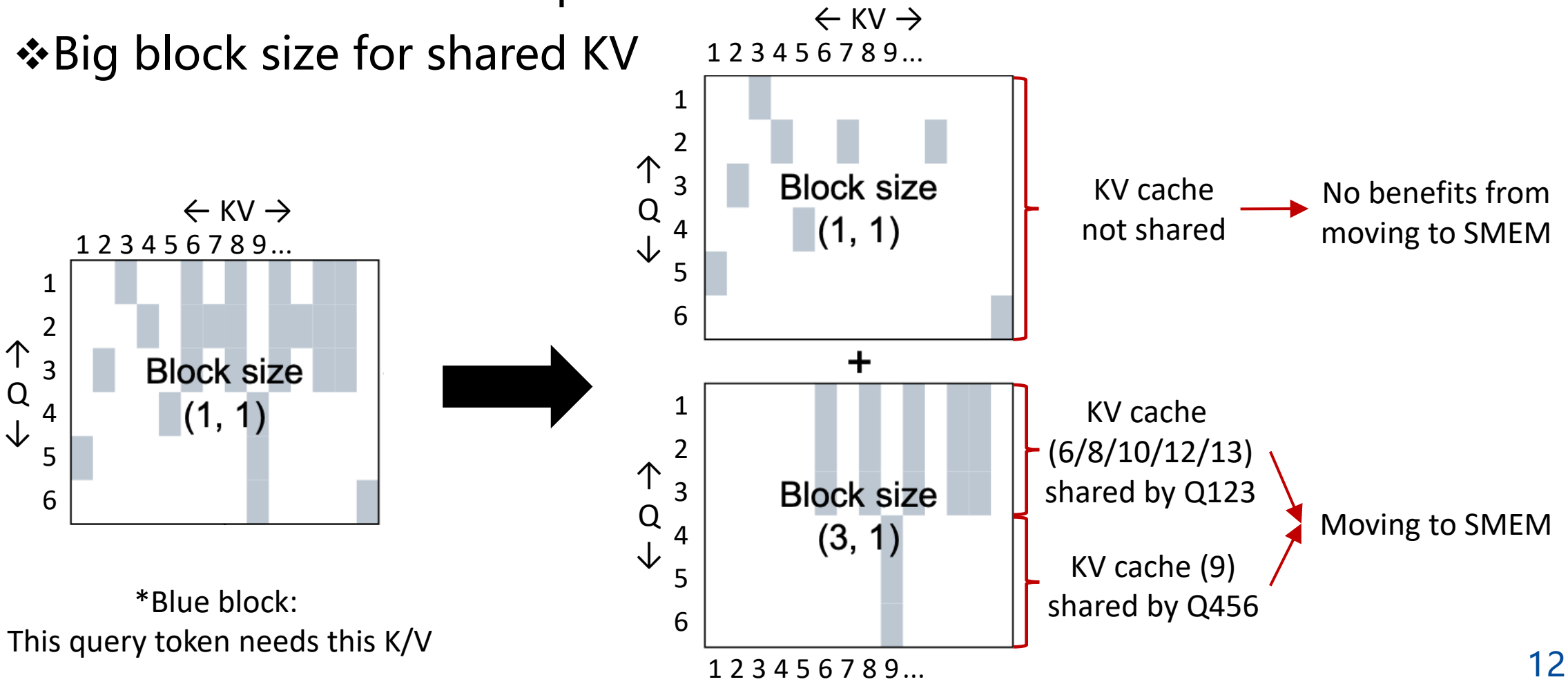




Unified KV Cache Format

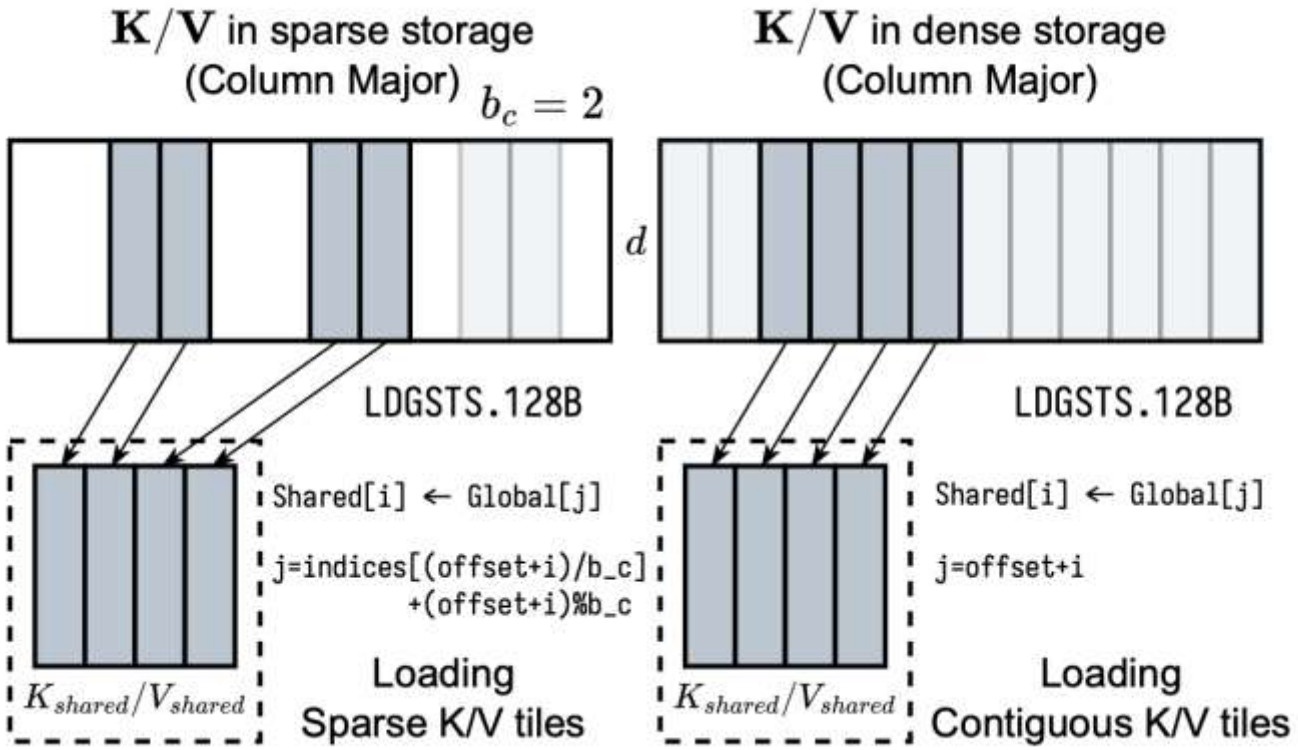
❑Composable Formats for prefix sharing

- ❖Small block size for unique KV
- ❖Big block size for shared KV



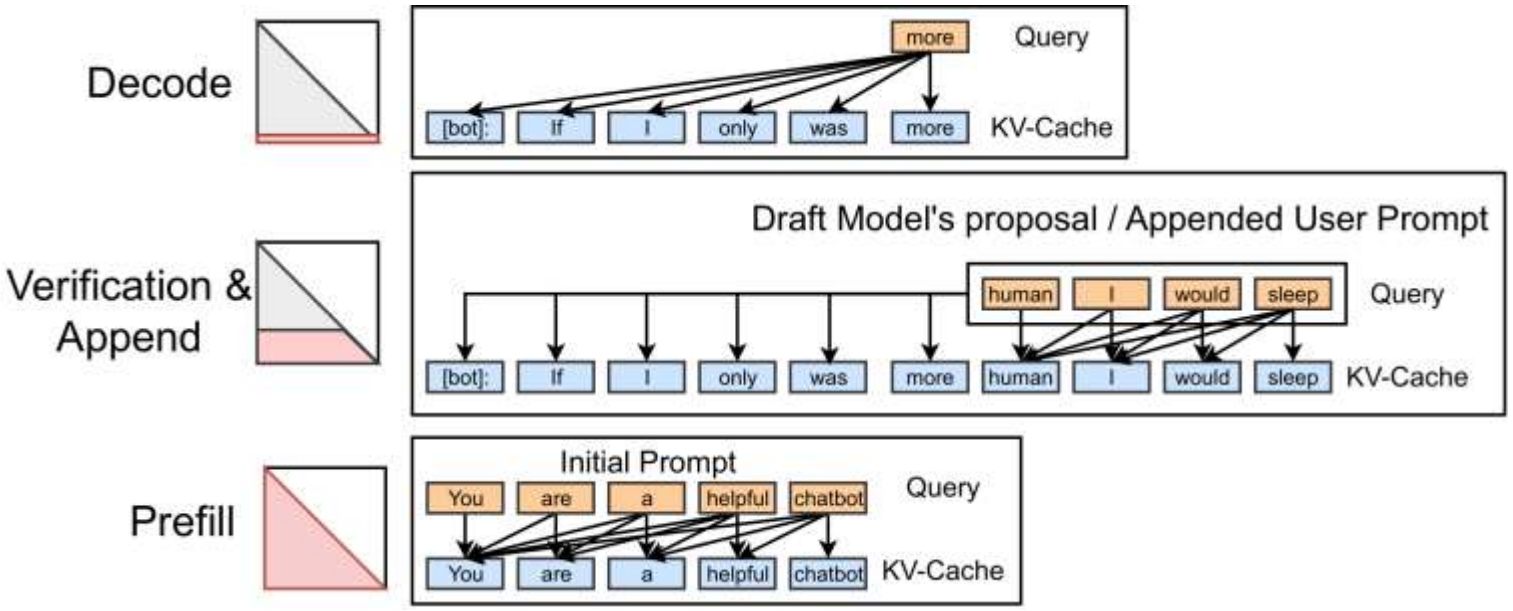
Sparse Row Gathering

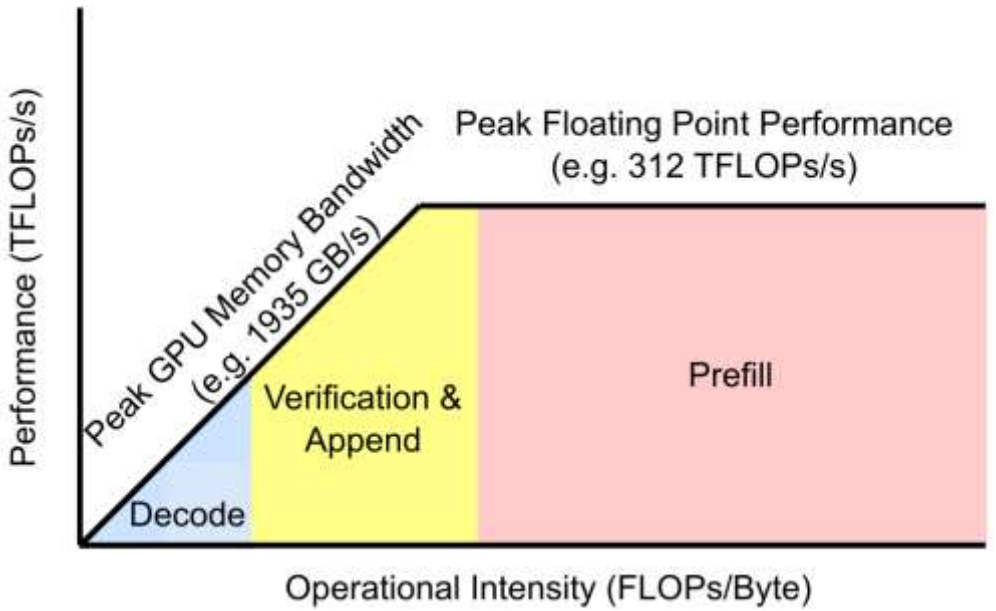
- **FlashInfer supports any block size**
- **To align tensor core shape**
  - Gathering from scattered HBM to contiguous SMEM



Tile Size Selection

- **Different query requires different tile size**
  - IO-bound: Decode, Append for small query length
  - Multiple tile-size options: (1, 16, 32, 64, 128) x (32, 64, 128)
    - FlashAttention only supports (64, 128) tiles
  - Heuristic selection based on hardware and workload

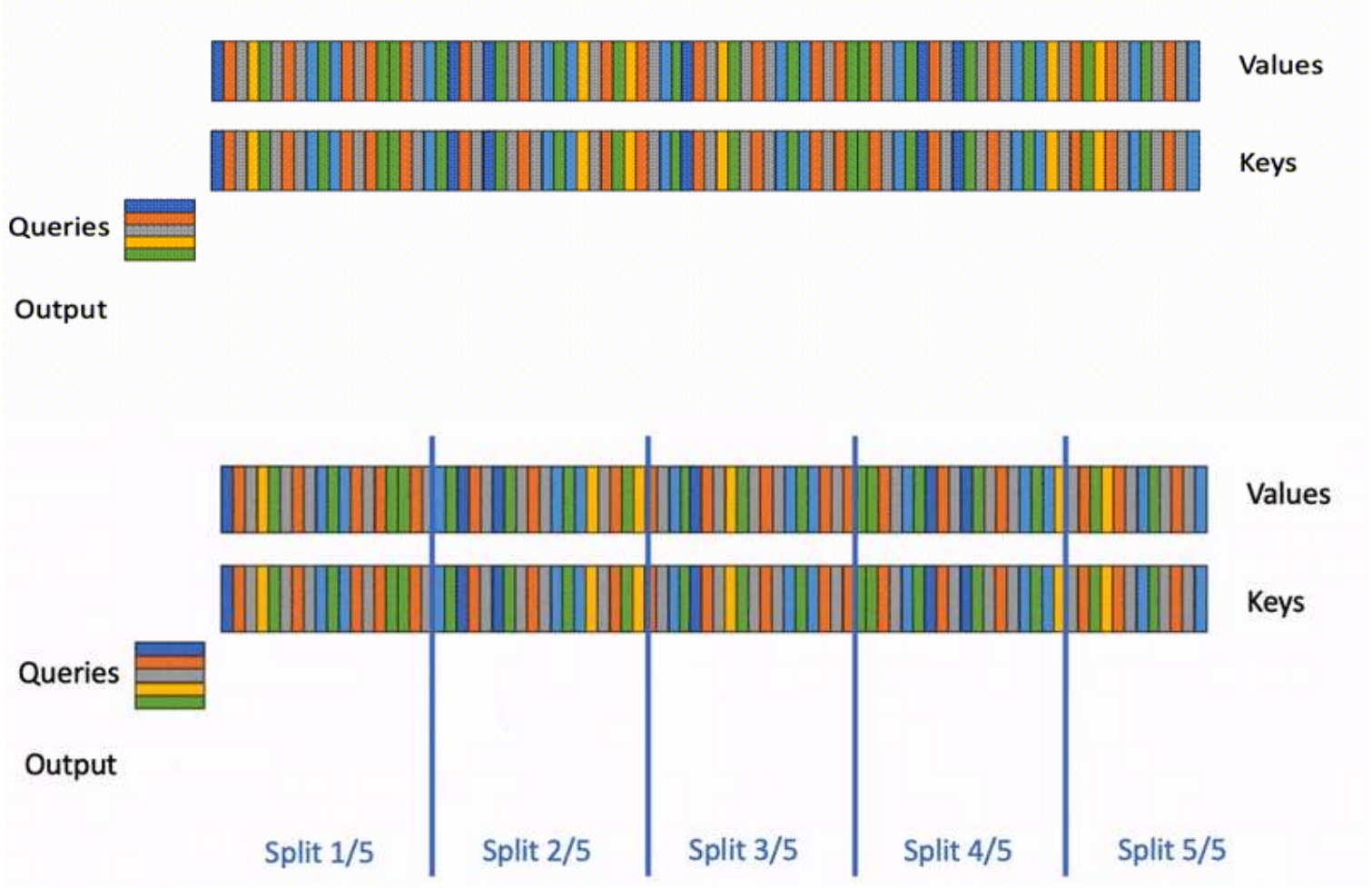




Load Balancing Schedule

❑FlashDecoding [MLSys24]

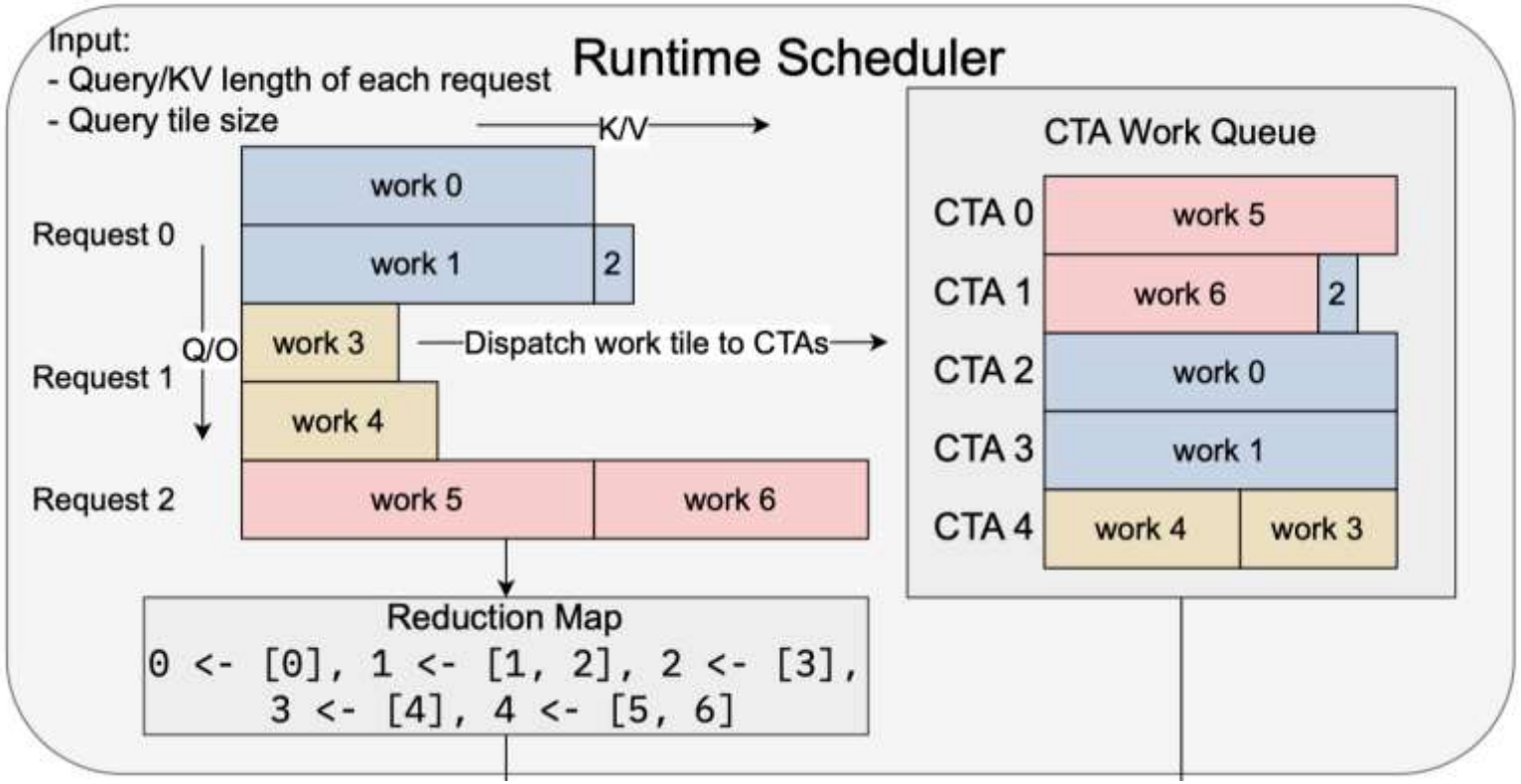
❖Split-K to efficiently leverage SMs



Load Balancing Schedule

❑Dispatching the attention work to CTAs

- ❖1 CTA (thread block) for 1 SM
- ❖Avoiding idle SM due to load imbalance



Attention Specification

- Similar to FlexAttention
- Accept CUDA code string

Attention Specification in Python

```
spec_decl = r"""
template <typename Params_, typename KernelTraits_>
struct FlashSigmoid {
  using Params = typename Params_;
  using KernelTraits = typename KernelTraits_;
  static constexpr bool use_softmax = false;
  float scale, bias;
  FlashSigmoid(const Params& params, int batch_idx, uint8_t*
smem_ptr) {
    // Copy from CUDA constant memory to registers
    scale = params.scale;
    bias = params.bias;
  }
  ...
  float LogitsTransform(const Params& params, float
logit_score, int batch_idx, int qo_idx, int kv_idx, int
qo_head_idx, int kv_head_idx) {
    return 1. / (1. + expf(-(logits_score * scale + bias)));
  }
};
"""
attn_spec = AttentionSpec(
  "FlashSigmoid",
  dtype_q, dtype_kv, dtype_o, idtype, head_dim, is_sparse,
  additional_vars=[("scale", "float"), ("bias", "float")],
  additional_tensors=[],
  spec_decl=spec_decl
)
```

Programming Interface

- **Init:**
  - JIT compilation
- **Plan:**
  - Dynamic load balancing schedule
  - Multiple layers use the same plan
    - cost amortized
- **Run:**
  - Select best CUDA graph
  - Replay CUDA graph

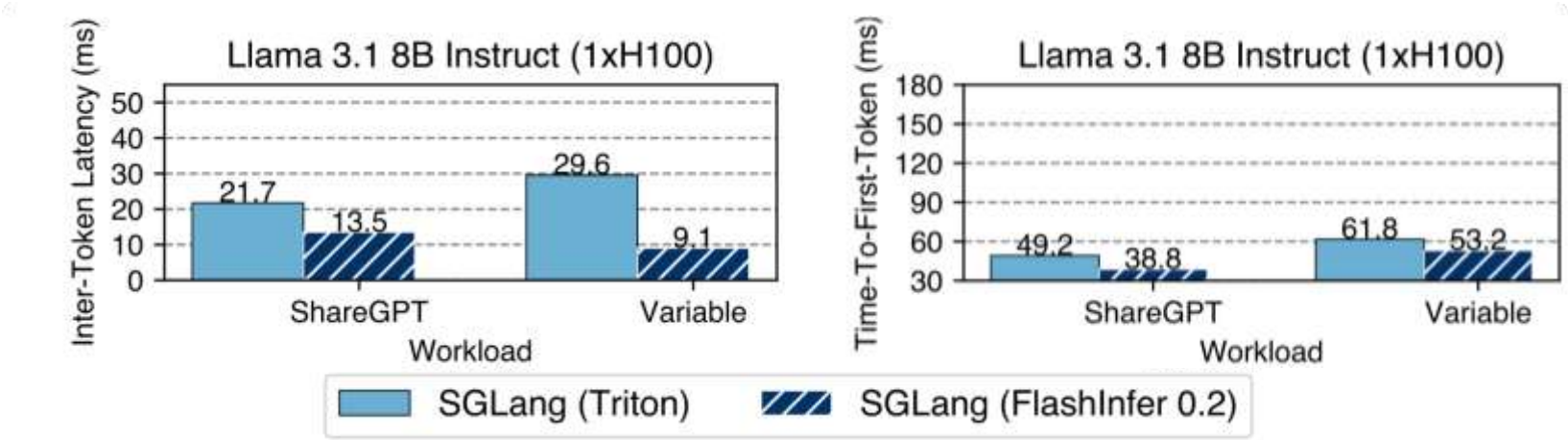
```
# Create workspace buffer
workspace = torch.empty(...)
seqlen_info.init()

# Compile: create CUDAGraphs
graphs = []
for task_info in task_infos:
  # Init: compile kernels according to spec
  attn = AttentionWrapper(attn_spec, task_info, workspace)
  g = torch.cuda.CUDAGraph()
  # Dummy plan
  attn.plan(seqlen_info)
  # Capture CUDA graphs
  with torch.cuda.graph(g):
    for i, layer in enumerate(layers):
      ...
      attn.run(...)
      ...
  graphs.append(g)

# Runtime: select the best CUDAGraph
g = select_graph(graphs)
finished = False
# Text generation loop
while not finished:
  seqlen_info.update()
  # Plan per generation step
  attn.plan(seqlen_info)
  # Replay CUDA-Graph
  g.replay()
```

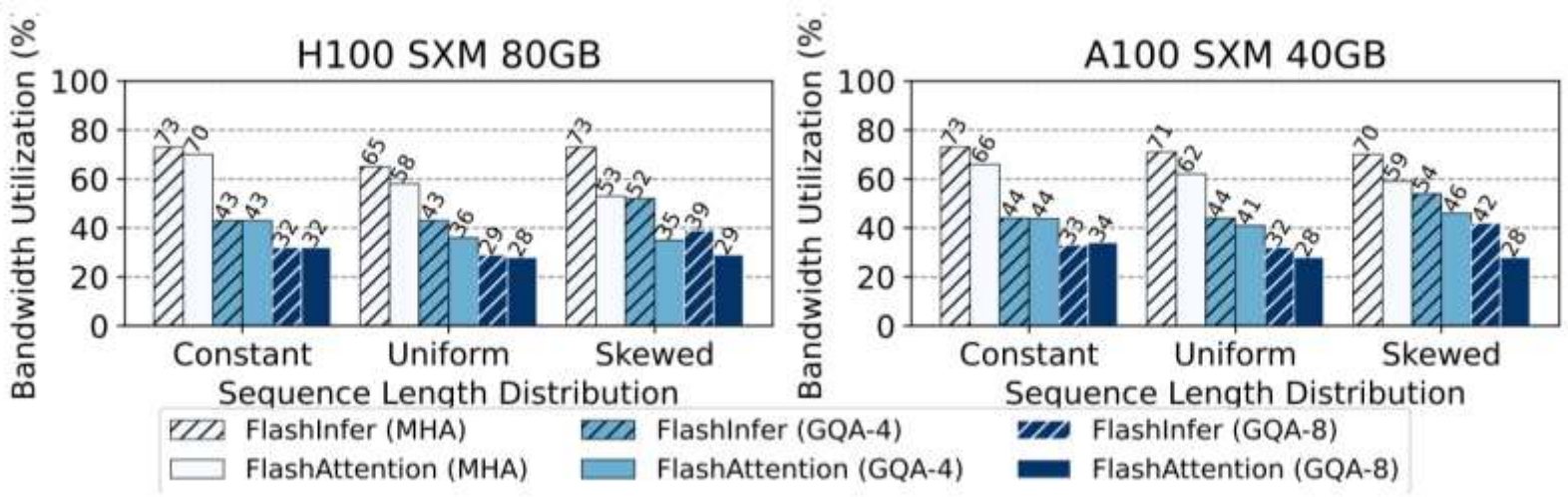
Evaluation: End-to-End on SGLang

- **FlashInfer outperforms Triton-based FlashAttention**



Evaluation: FA Kernel

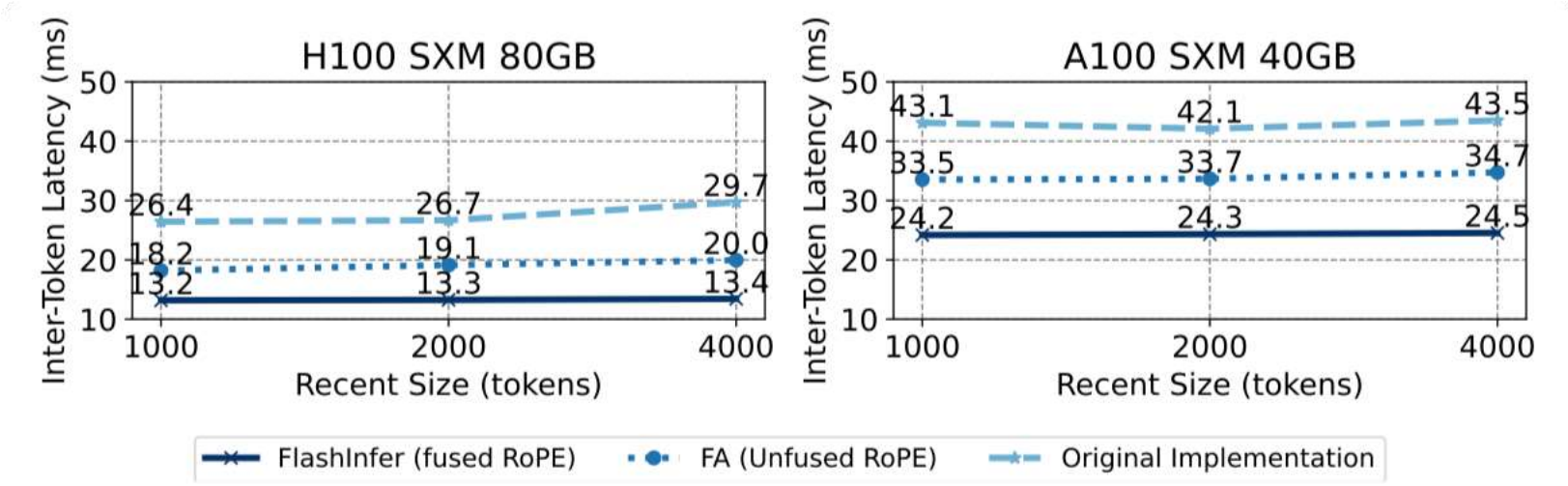
- **FlashInfer outperforms FlashAttention due to**
  - load-balancing scheduler
  - versatile tile-size selection



Decode kernel

Evaluation: StreamingLLM Use Case

- **RoPE of StreamingLLM should be modified**
  - Easy fusion with FlashInfer
    - only 20 lines of code
    - 28 – 30% latency reduction



Evaluation: FlexAttention

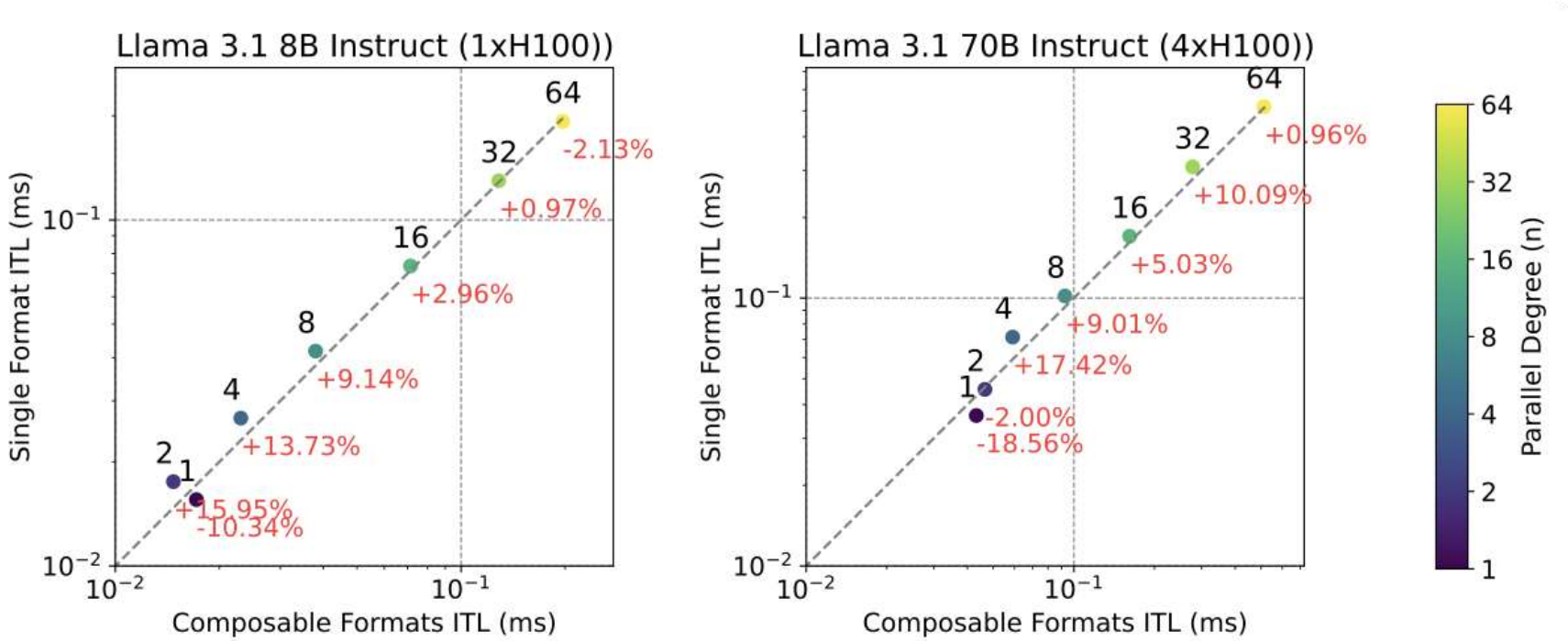
- **FlexAttention:**
  - User-friendly interface for attention variants
  - Generating block-sparse Triton-based FlashAttention
  - Problem:
    - Triton FA is slower than native FA
    - Triton lag in adopting new hardware features (e.g., Hopper)

Seq Length	FlexAttention	FlashInfer
512	209.11	**250.454**
1024	294.53	**406.554**
2048	376.90	**487.236**
4096	421.00	**548.388**
8192	441.26	**587.903**
16384	453.57	**612.259**

TFLOPS Comparison on Causal Attention

Ablation Study: Composable Formats

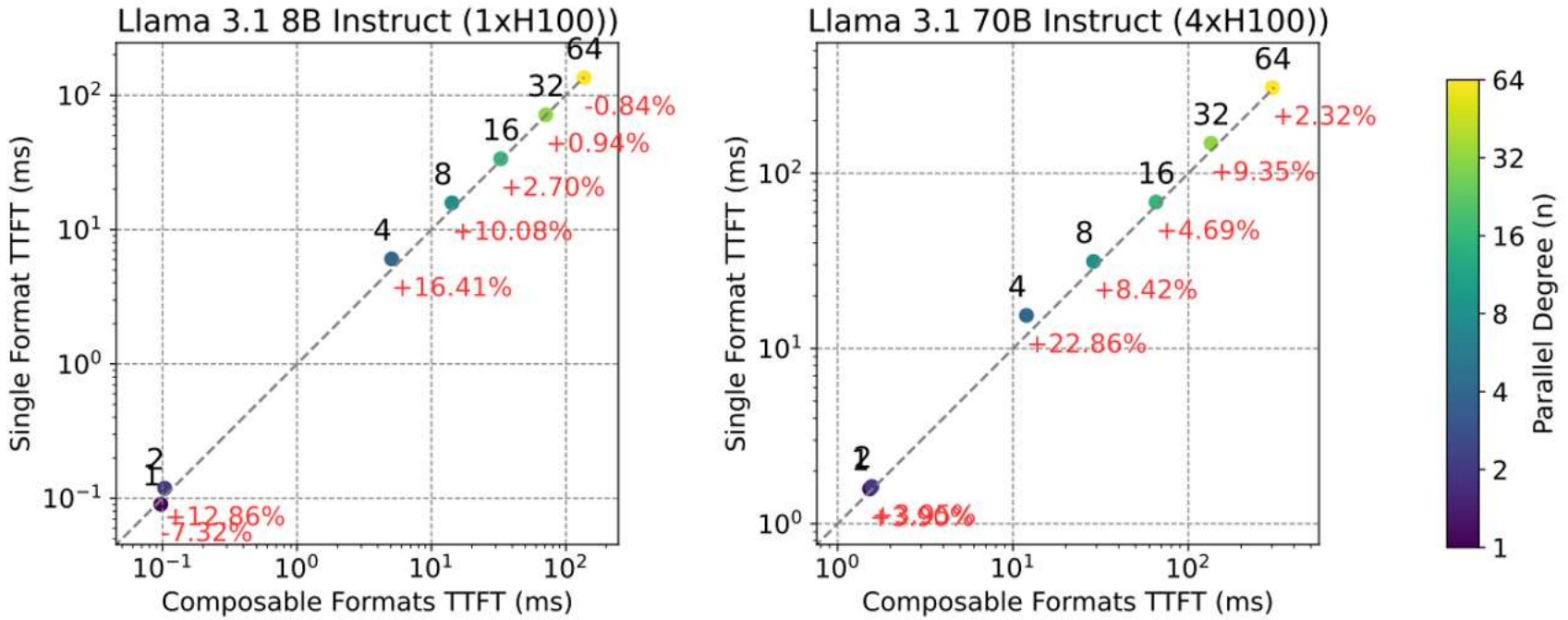
- **Scenario: parallel generation sharing the same prompt**
  - "n" parameter in OpenAI API
  - Used to select from multiple options
    - Chain-of-Thoughts, math solving, A/B test, etc.



Evaluation on MLC-Engine

Ablation Study: Composable Formats

- **Scenario: parallel generation sharing the same prompt**
  - "n" parameter in OpenAI API
  - Used to select from multiple options
    - Chain-of-Thoughts, math solving, A/B test, etc.



Evaluation on MLC-Engine

Ablation Study: Composable Formats

- **Scenario: prefix-sharing**

Table 5. Latency of Shared-Prefix Attention Kernels

Shared Prefix Length	Composable (BS=16)	Single (BS=16)	Composable (BS=64)	Single (BS=64)
1024	**45.17**	46.52	87.86	130.49
8192	**88.67**	226.57	125.76	931.75
32768	**217.42**	945.67	254.54	4090

Ablation Study: Load Balancing

- **Load balancing reduces ITL and TTFT**

Table 6. Load-balancing Scheduler Ablation Study (ITL)

Scenario	w/ Load-Balancing	w/o Load-Balancing	Triton
ShareGPT (RR=16)	**8.96**	9.16	9.36
$U(512, 2048)$ (RR=8)	**8.21**	8.42	8.49
$U(4096, 16384)$ (RR=1)	**8.63**	13.89	11.08

Table 7. Load-balancing Scheduler Ablation Study (TTFT)

Scenario	w/ Load-Balancing	w/o Load-Balancing	Triton
ShareGPT (RR=16)	**39.05**	39.42	52.92
$U(512, 2048)$ (RR=8)	**66.78**	67.38	68.48
$U(4096, 16384)$ (RR=1)	**411.02**	421.60	566.30

Ablation Study: Load Balancing

- **Load balancing reduces ITL and TTFT**

Table 6. Load-balancing Scheduler Ablation Study (ITL)

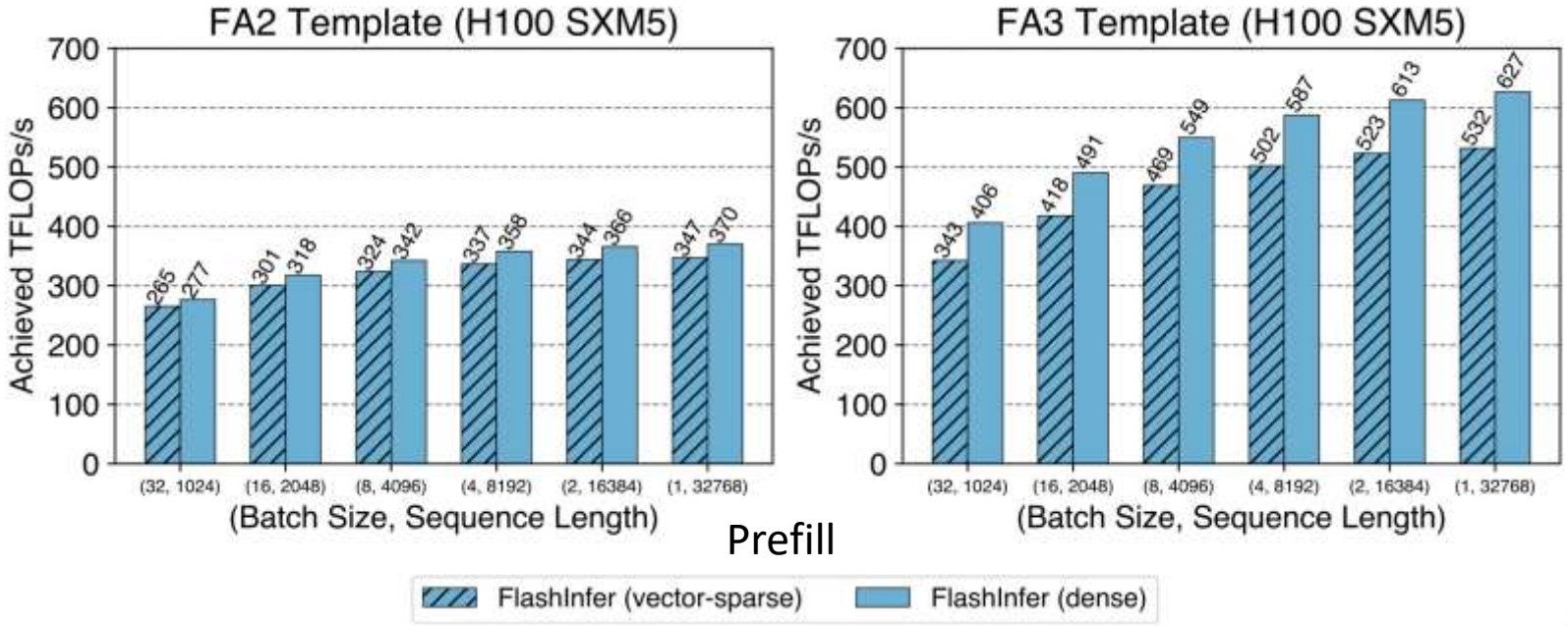
Scenario	w/ Load-Balancing	w/o Load-Balancing	Triton
ShareGPT (RR=16)	**8.96**	9.16	9.36
$U(512, 2048)$ (RR=8)	**8.21**	8.42	8.49
$U(4096, 16384)$ (RR=1)	**8.63**	13.89	11.08

Table 7. Load-balancing Scheduler Ablation Study (TTFT)

Scenario	w/ Load-Balancing	w/o Load-Balancing	Triton
ShareGPT (RR=16)	**39.05**	39.42	52.92
$U(512, 2048)$ (RR=8)	**66.78**	67.38	68.48
$U(4096, 16384)$ (RR=1)	**411.02**	421.60	566.30

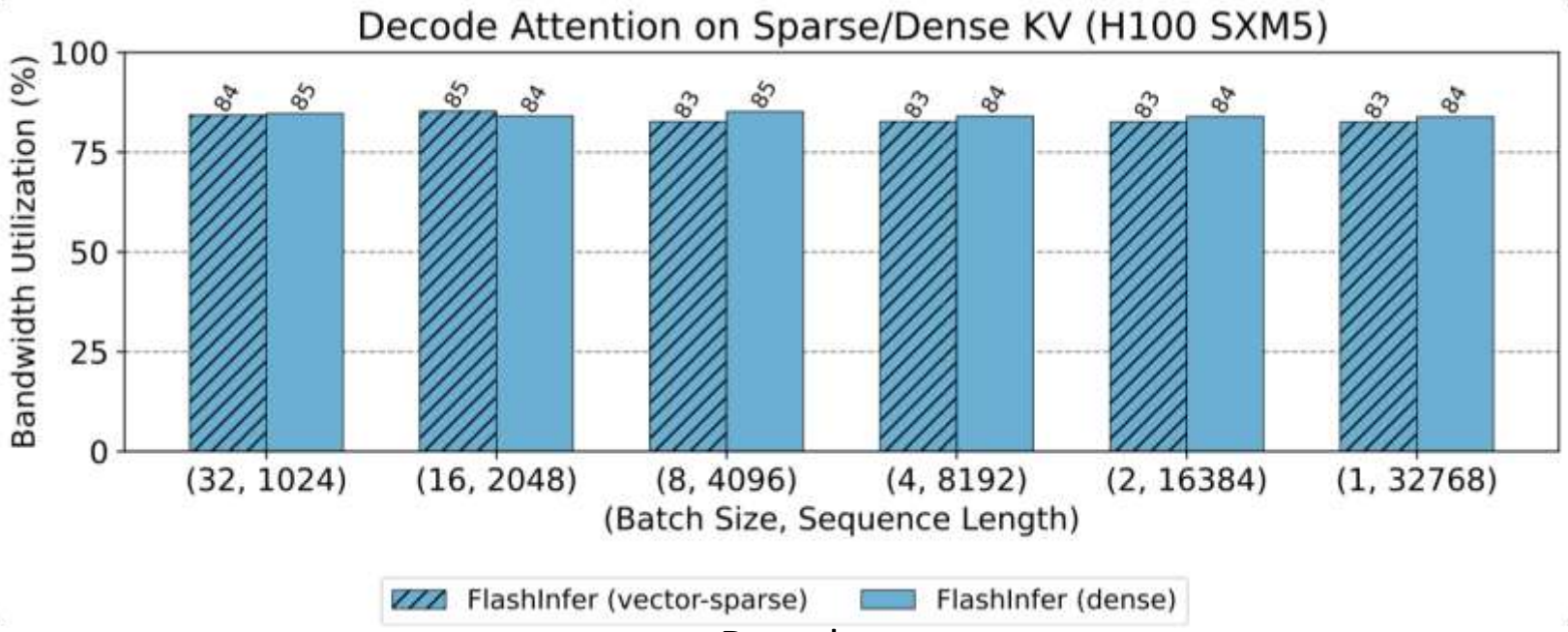
Ablation Study: Overhead of Sparse Gathering

- **Sparse row gathering involves overhead**
  - Moving Scattered HBM to contiguous SMEM, Index management
- **FA3 leverages TMA for KV loading**
  - TMA requires fixed-stride access
    - while sparse gathering of FlashInfer requires arbitrary row indices



Ablation Study: Overhead of Sparse Gathering

- **Sparse row gathering involves overhead**
  - Moving Scattered HBM to contiguous SMEM, Index management
  - No need of sparse gathering for contiguous blocks



Decode

Discussion

- **Minor issues:**
  - Some techniques are similar to existing papers
    - load-balancing scheduler
    - sparse-row gathering
- **Expertise in CUDA**
  - FlashInfer template requires a string of CUDA code
  - FlexAttention is enough for simple verfication
    - FlexAttention only requires PyTorch code
    - Triton now also supports Hopper advanced features

Outline

Baseline	Scenario	Baseline Feature	FlashInfer Feature
SGLang w/ Triton 4.1	E2E ITL, TTFT, ShareGPT, Variable		Overall
FlashAttention2&3 4.2	decoding of mha / gqa, prefill of mha		load-balancing scheduler decoding: versatile tile-size selection
FlashAttention2&3 4.3	StreamingLLM for RoPE fusion	hard to manually fuse RoPE in FA	only 20 PyTorch LOC to fuse
FlexAttention G.1	normal Attention variants (TFLOPS)	based on Triton (lagging in adopting new Hopper advanced features)	based on CUDA/CUTLASS
FlexAttention G.5	fine-grained sparsity of Quest (latency)	large block size	sparse-row gathering (tensor core for small block size)

FlashInfer Part	Scenario	Baseline
load-balancing scheduler G.3	var. seq. len., SGLang	Triton, w/, w/o load-balancing scheduler
composable format 4.4, G.2	parallel generation (MLC-Engine), shared-prefix	composable vs single format