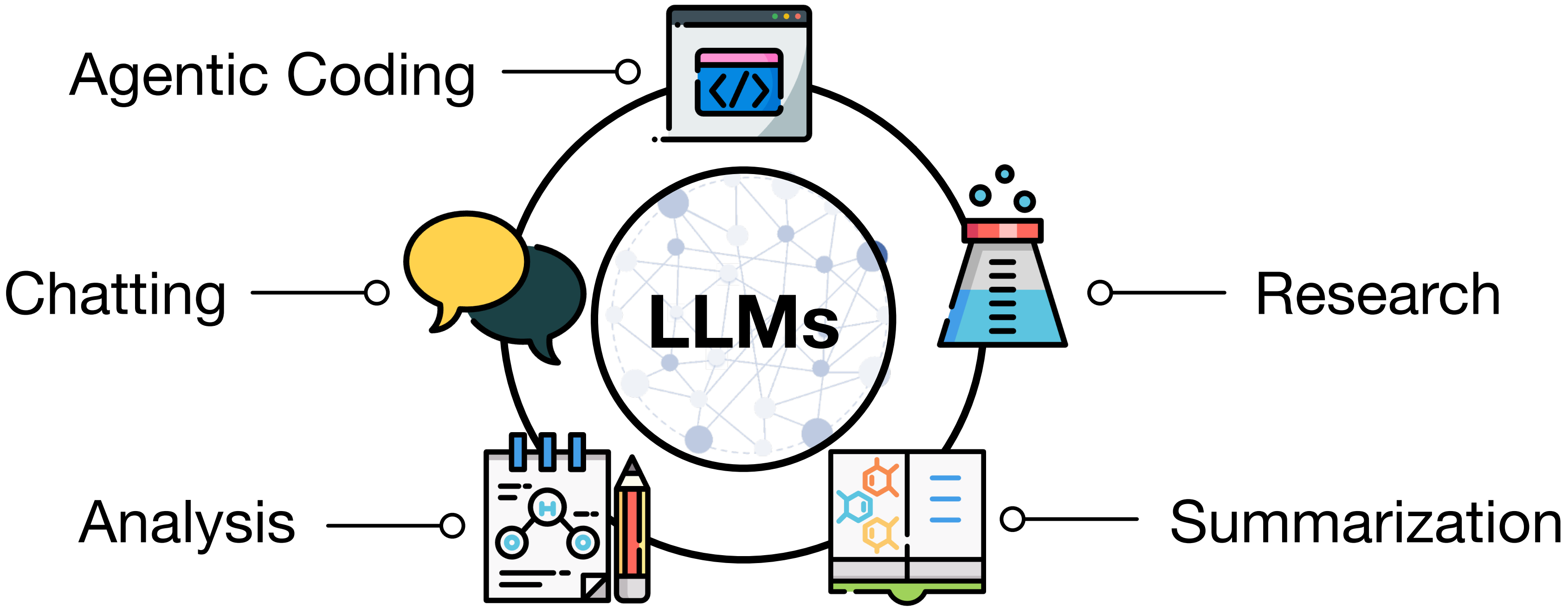
PIMBA: A Processing-in-Memory Acceleration for Post-Transformer Large Language Model Serving

Wonung Kim
Yubin Lee
Yoonsung Kim
Jinwoo Hwang
Seongryong Oh
Jiyong Jung
Aziz Huseynov
Woong Gyu Park
Chang Hyun Park§
Divya Mahajan†
Jongse Park

KAIST CASYS Lab § Uppsala University † Georgia Institute of Technology

KAIST
CASYS | KAIST Computer Architecture & System Lab
UPPSALA UNIVERSITET
Georgia Tech

Large Language Models Are Pervasive

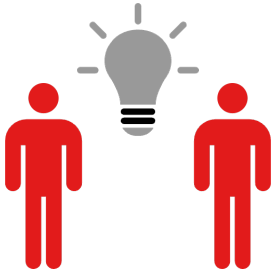


ChatGPT Gemini Claude

Modern LLM Trends

RAG

Knowledge



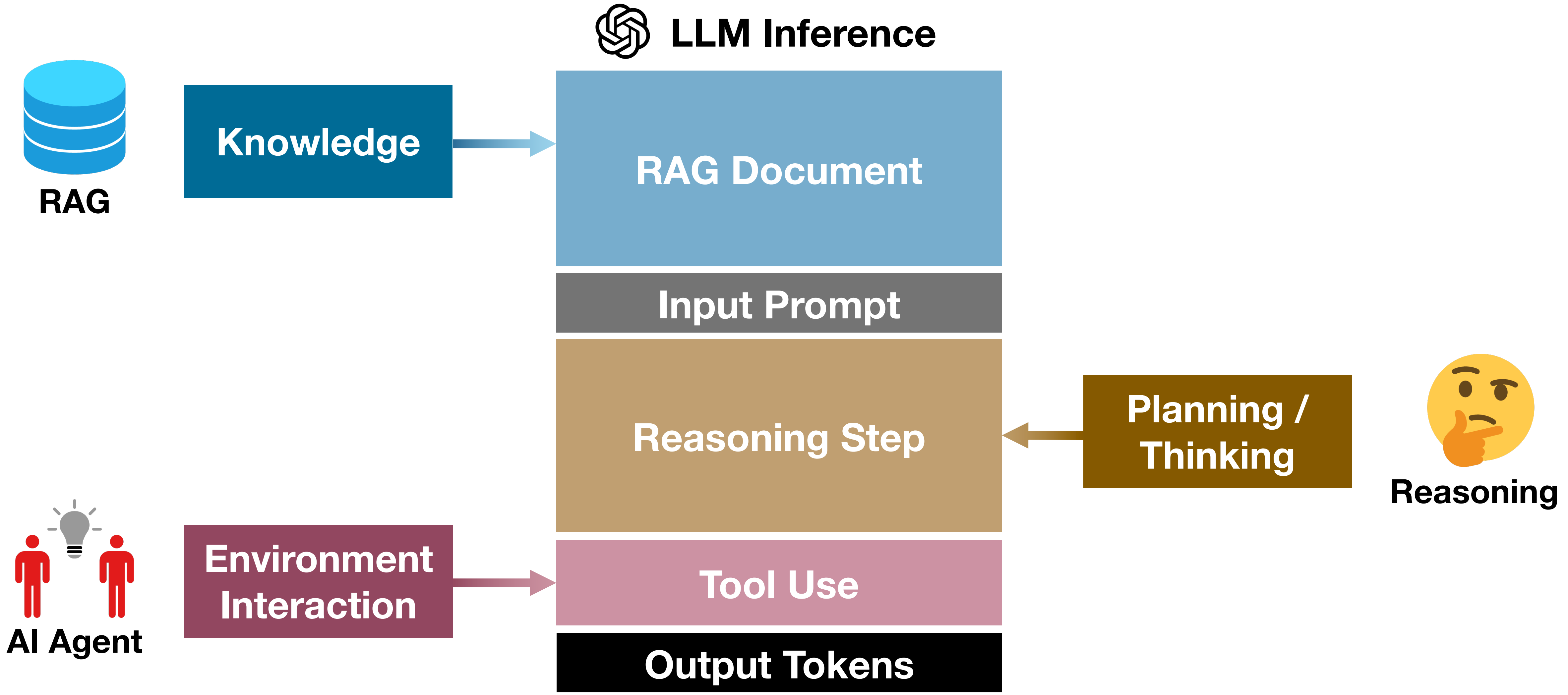
AI Agent

Environment Interaction

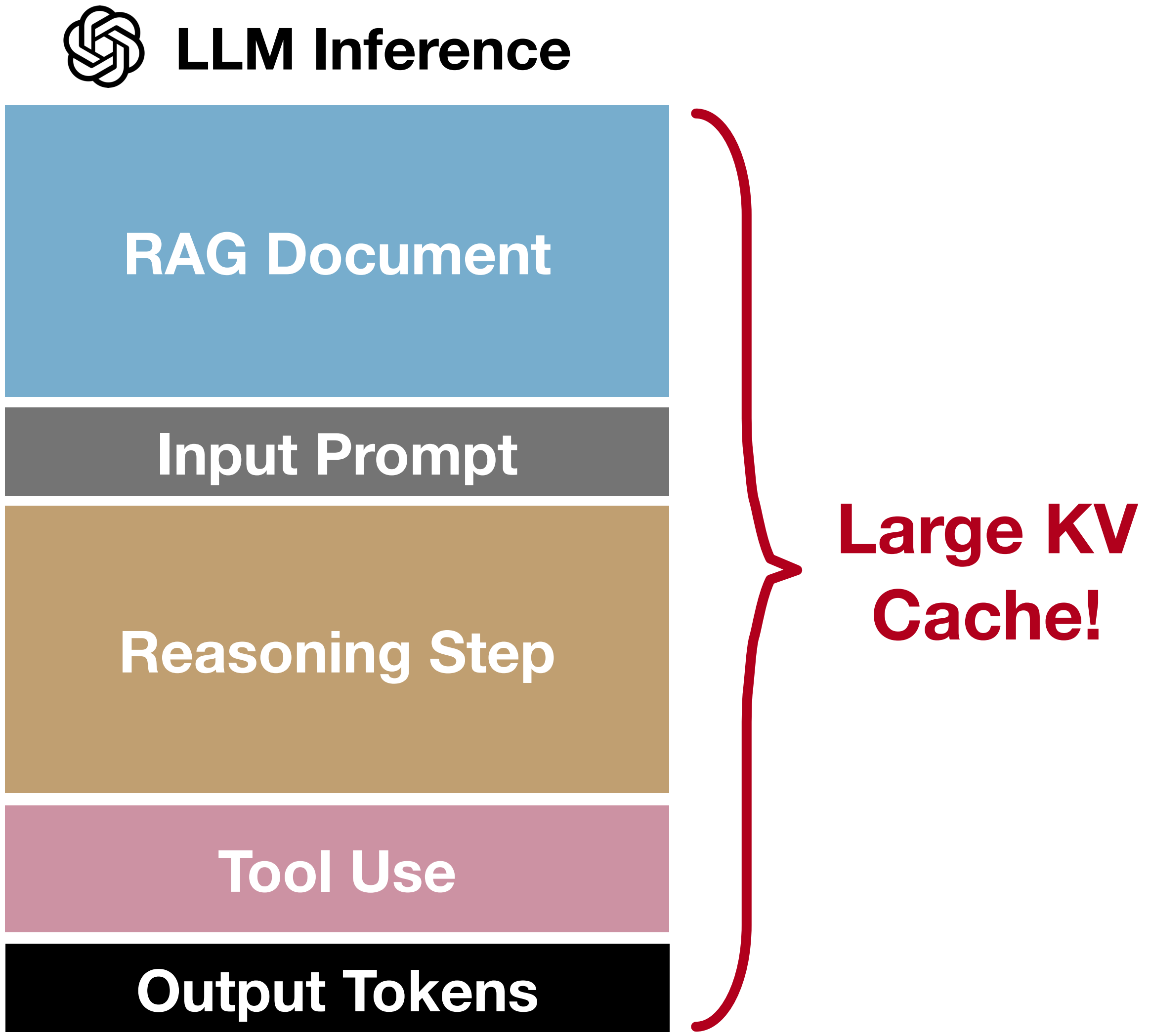
Planning / Thinking

Reasoning

Modern LLM Trends



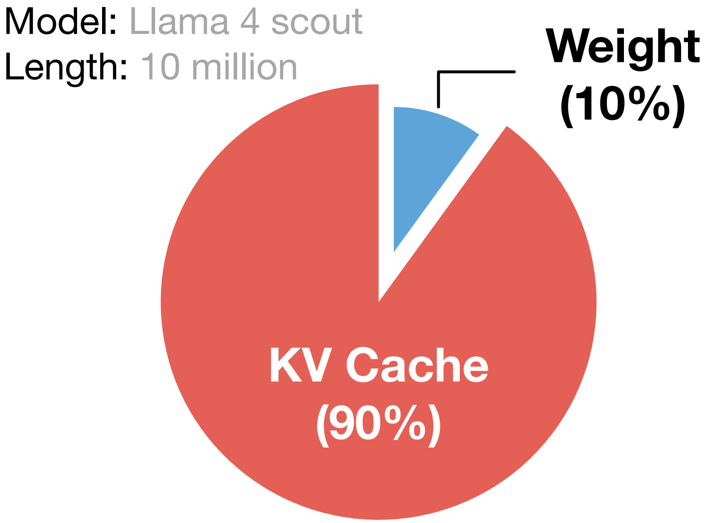
Modern LLM Trends: Large KV Cache



Modern LLM Trends: Large KV Cache

LLM Inference

Model	Context Length
Llama 4	10M
Grok 4	2M
Gemini 2.5	1M
Claude 4	1M



Modern LLMs now support up to 10M context length

Even with GQA, MoE, KV Cache dominates!

Modern LLM Trends: Large KV Cache

LLM Inference

RAG Document

Reasoning Step

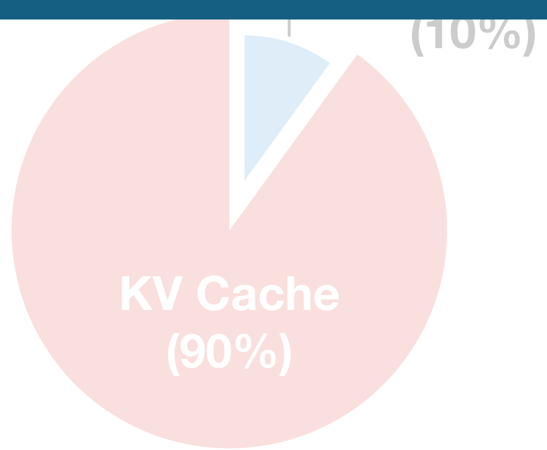
Tool Use

Output Tokens

Model	Context Length
Llama 4	10M
Grok 4	2M

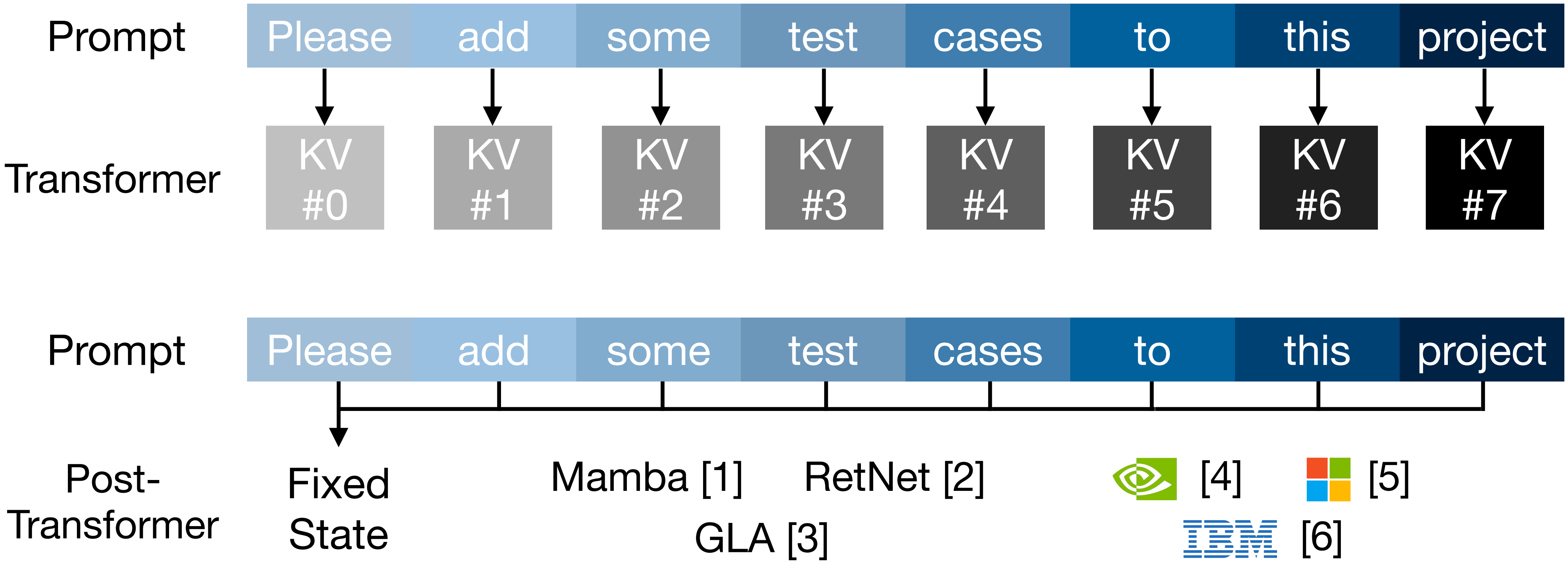
Modern LLMs now support up to 10M

It has become inevitable to confront the KV cache memory bottleneck



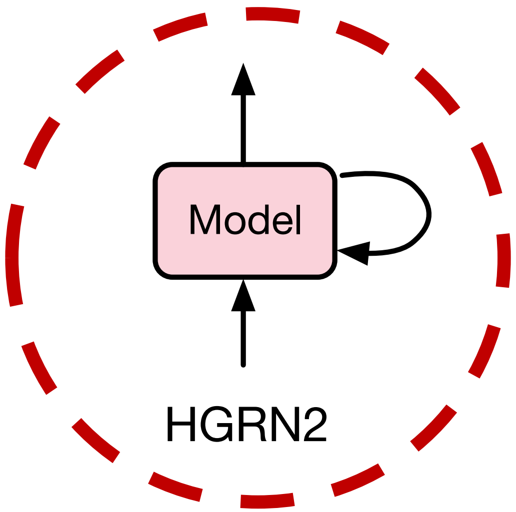
Even with GQA, MoE, KV Cache dominates!

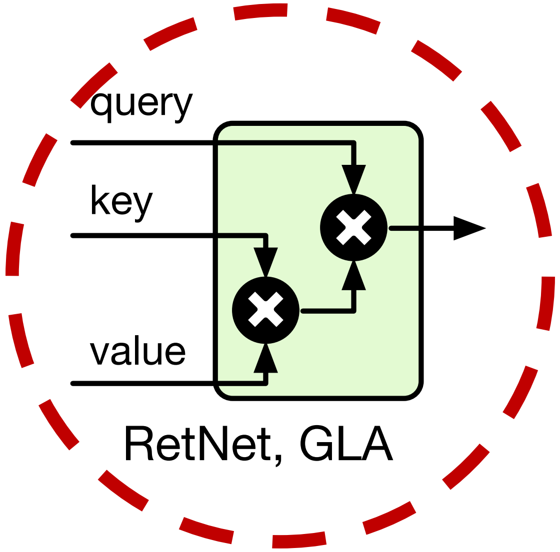
Post-Transformer Models



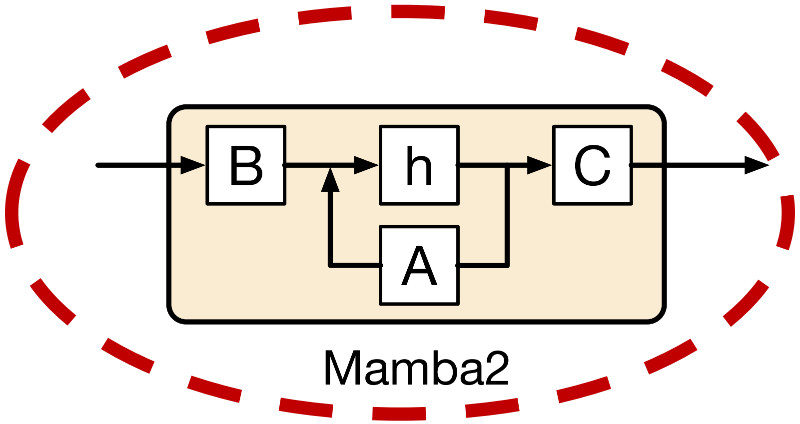
[1] Mamba: Linear-Time Sequence Modeling with Selective State Spaces
[2] Retentive Network: A Successor to Transformer for Large Language Models
[3] Gated Linear Attention Transformers with Hardware-Efficient Training
[4] Nemotron-H: A Family of Accurate and Efficient Hybrid Mamba-Transformer Models
[5] Samba: Simple Hybrid State Space Models for Efficient Unlimited Context Language Modeling
[6] IBM Granite 4.0: hyper-efficient, high performance hybrid models for enterprise

We begin by analyzing how post-transformers operate to identify performance bottlenecks





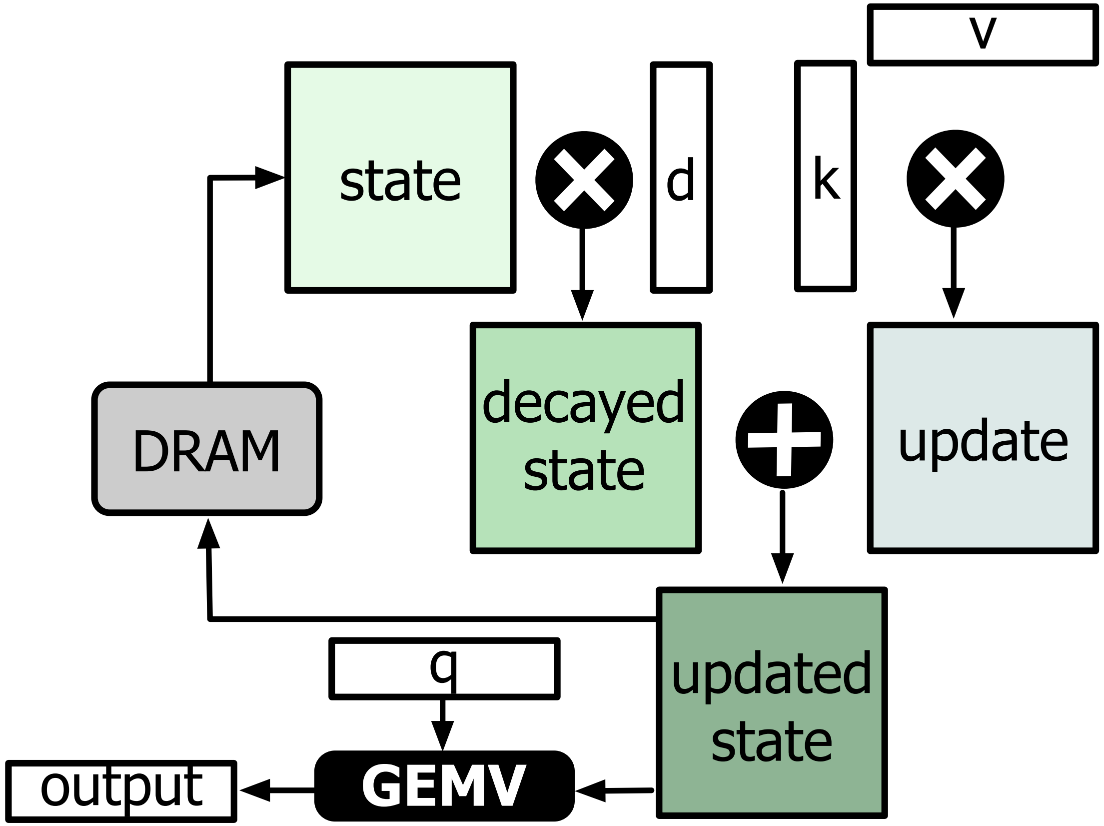
But, unlike transformers, post-transformers exhibit diverse algorithmic forms



We identified a common operator shared across these algorithms, which we call it as,

State Update

State Update Operation



❶ **Weight decay**

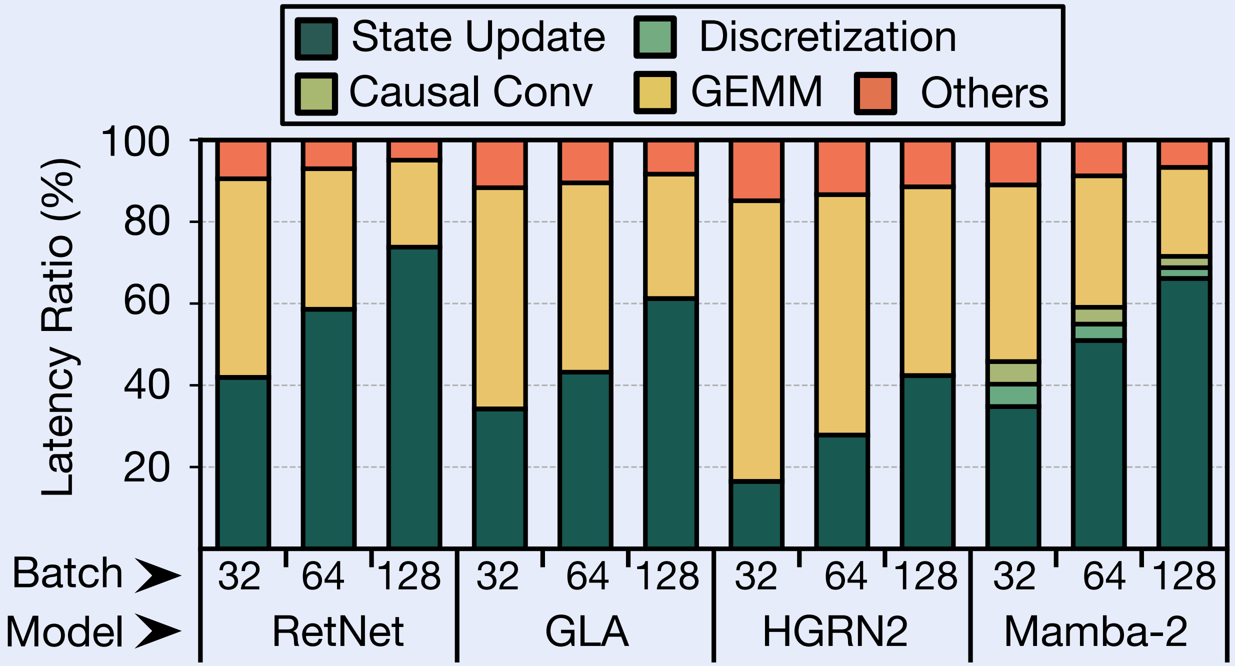
❷ **Outer product**

❸ **Update**

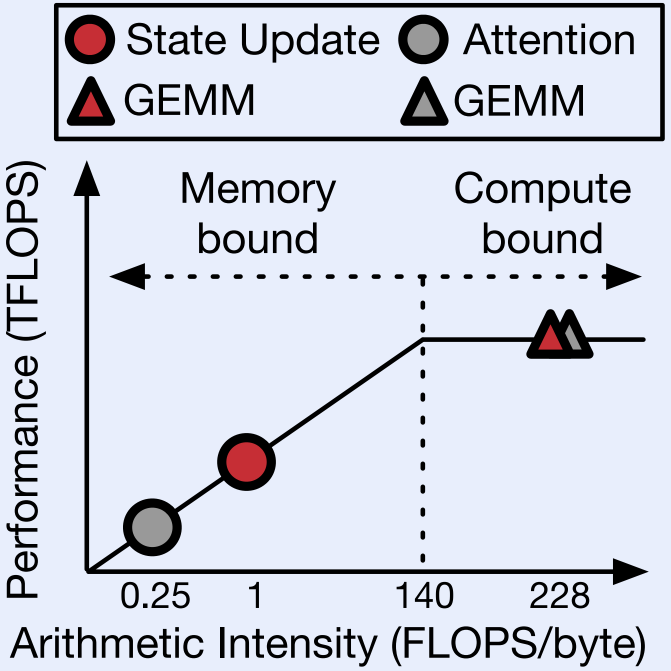
❹ **GEMV**

Characterizing Post-Transformers

Latency Breakdown

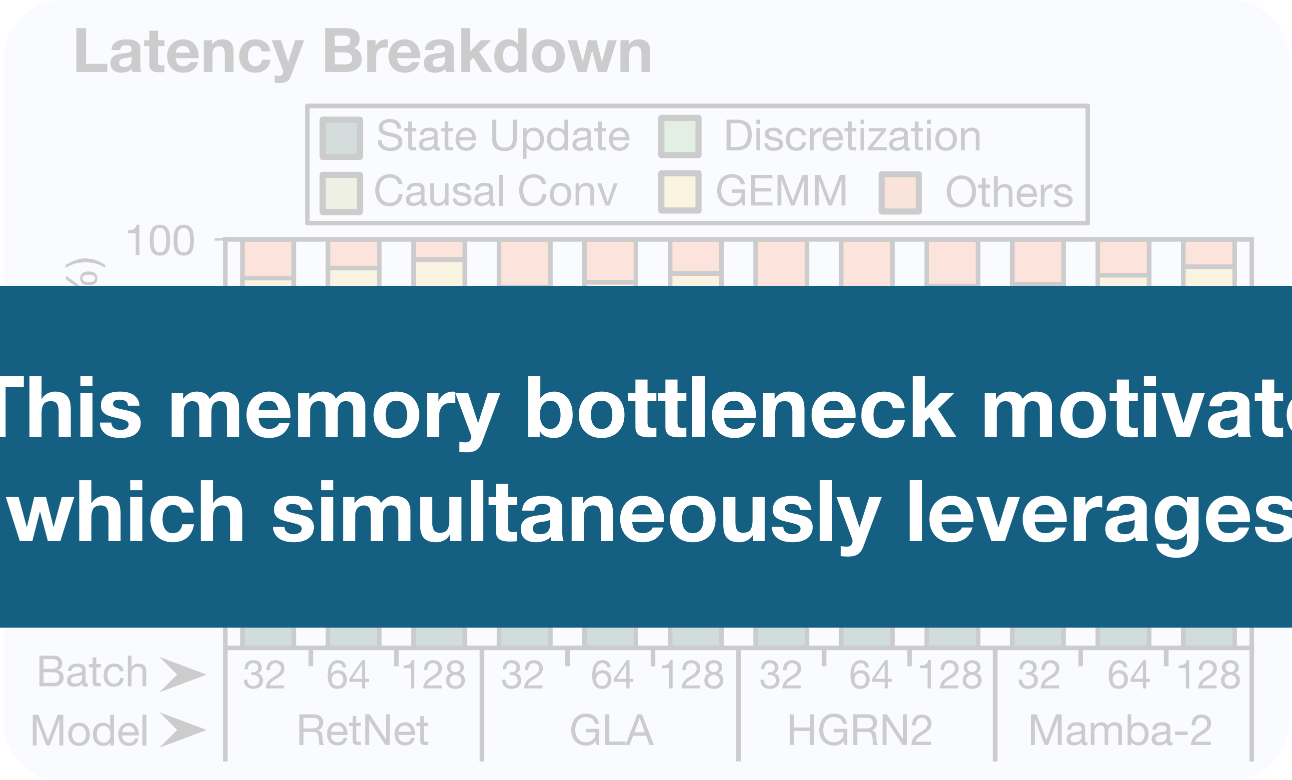


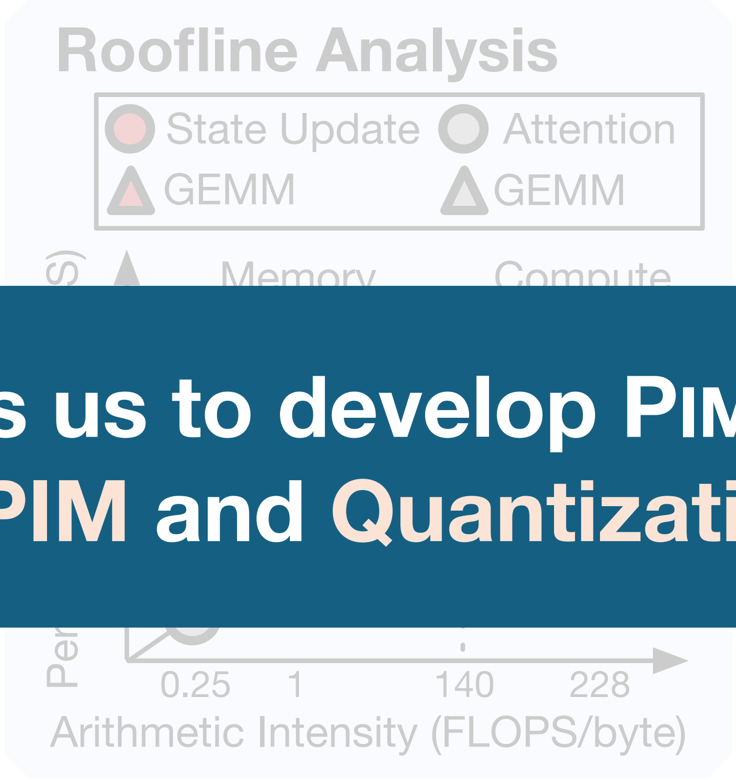
Roofline Analysis



- **State update operations dominate**
  - Due to lack of parameter reuse, state updates **cannot** be efficiently batched
  - Unlike GEMM, they have low arithmetic intensity, thus **memory-bound**

Characterizing Post-Transformers

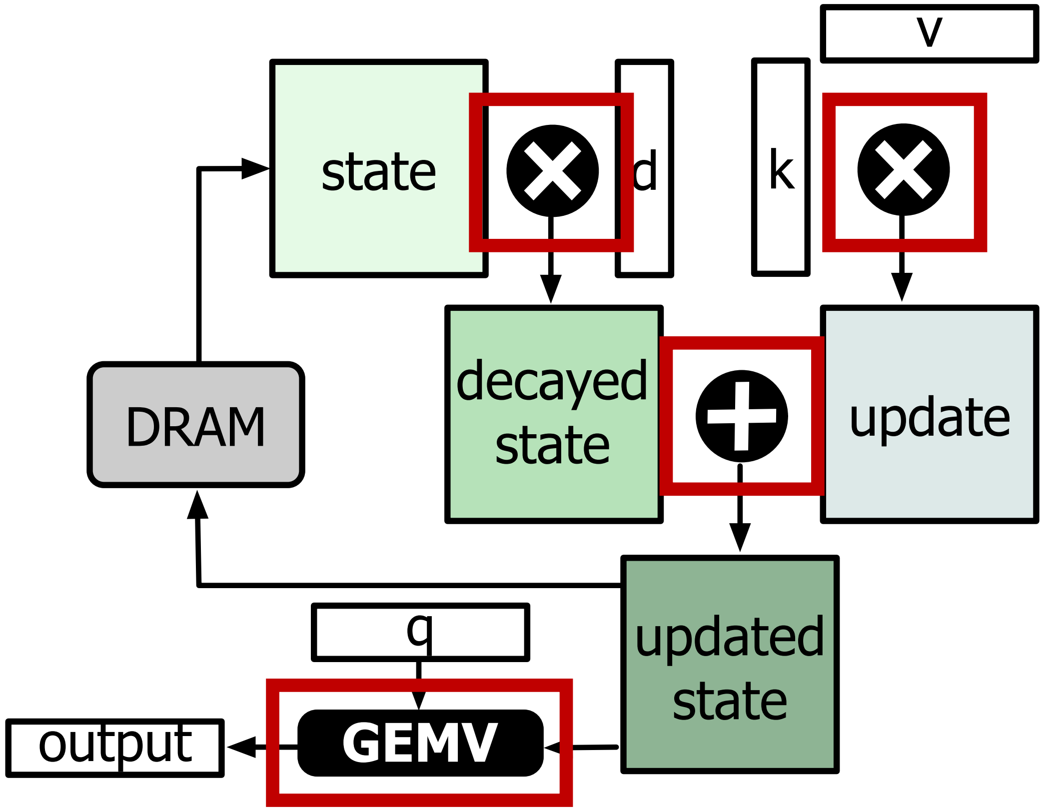




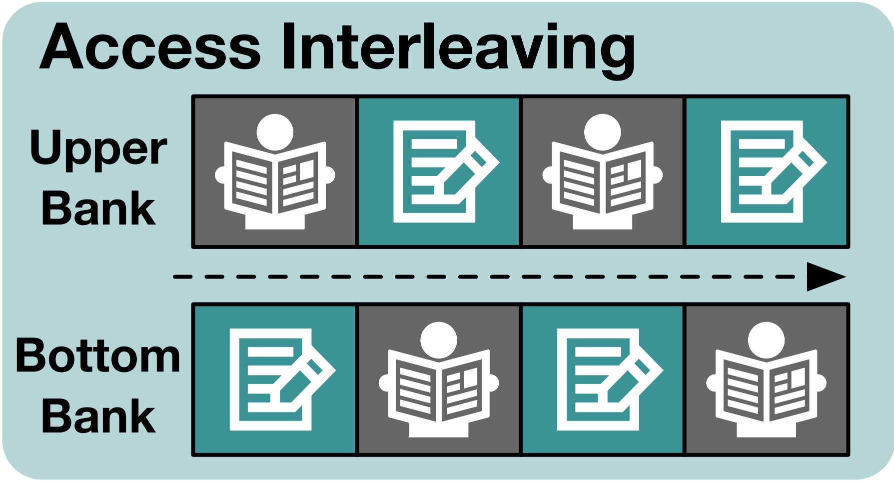
This memory bottleneck motivates us to develop PIMBA, which simultaneously leverages PIM and Quantization

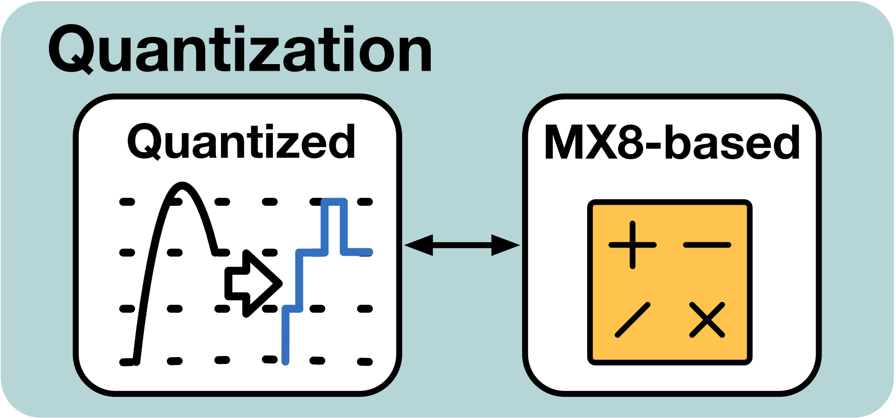
- **State update operations dominate**
  - Due to lack of parameter reuse, state updates **cannot** be efficiently batched
  - Unlike GEMM, they have low arithmetic intensity, thus **memory-bound**

PIMBA Overview



How?



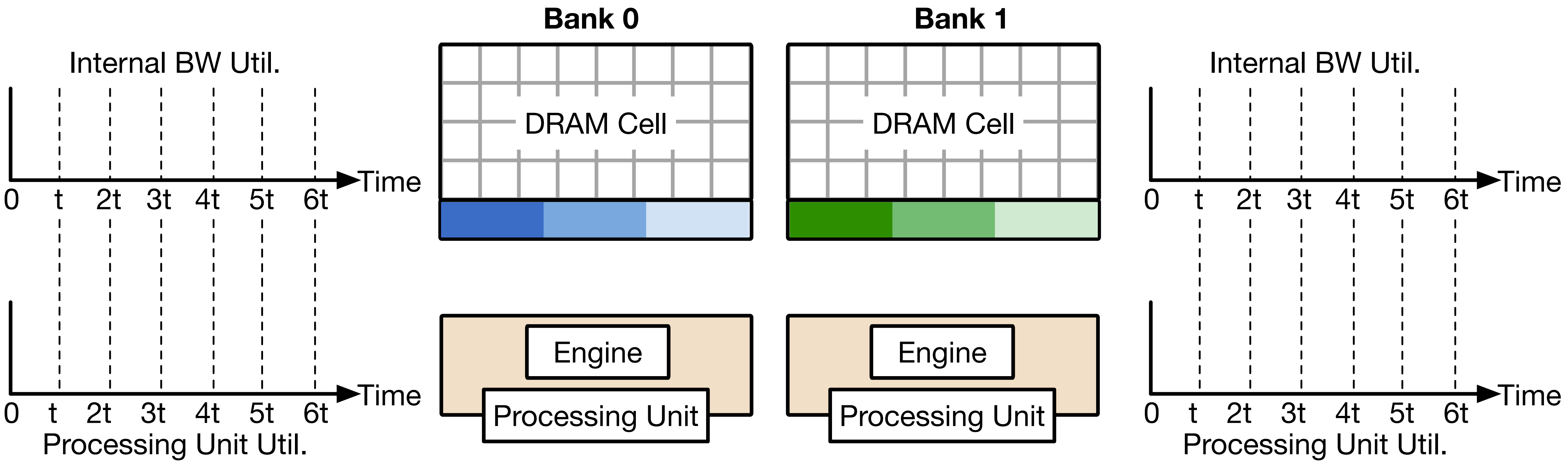


Diverse operations needed!

We focus on optimizing area overhead

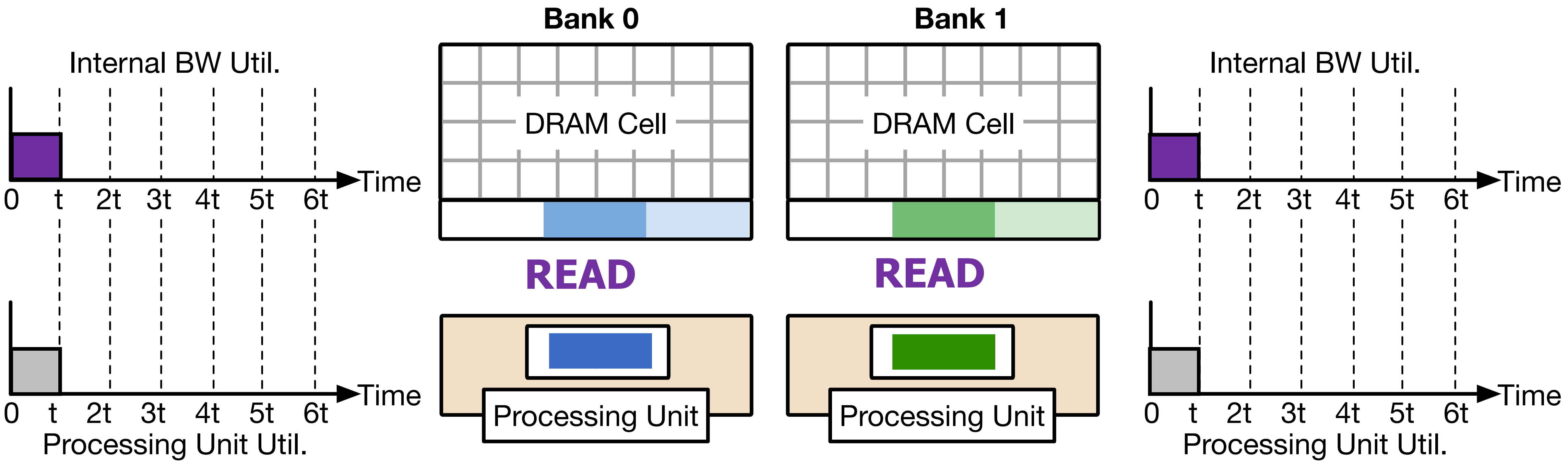
State Updates in Existing PIM

Access Interleaving
Quantization



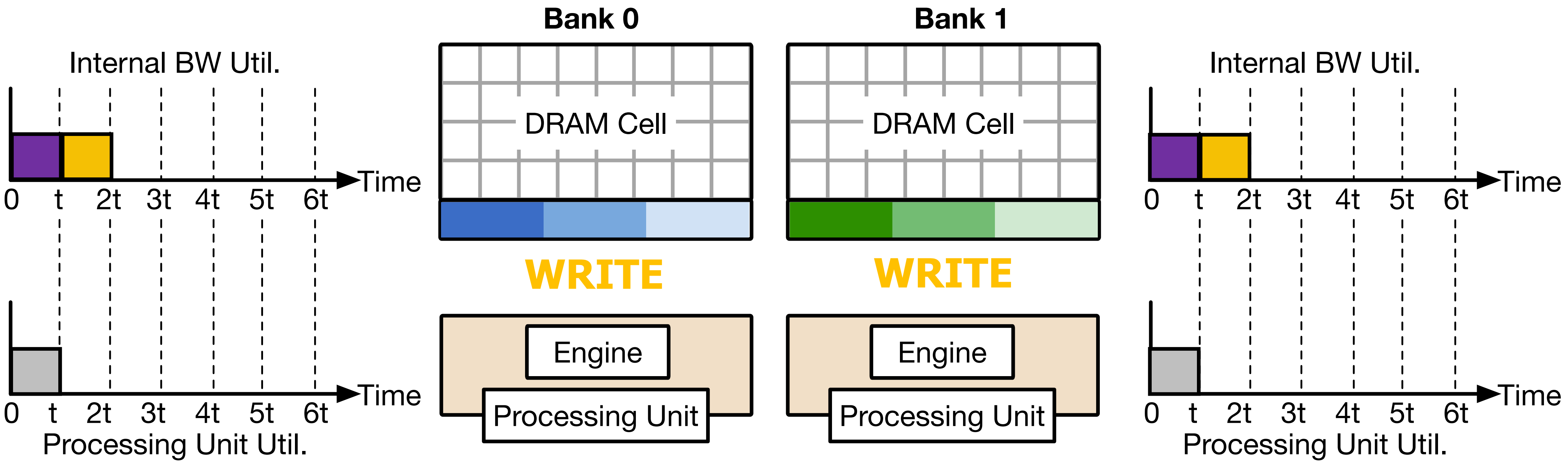
State Updates in Existing PIM

Access Interleaving
Quantization



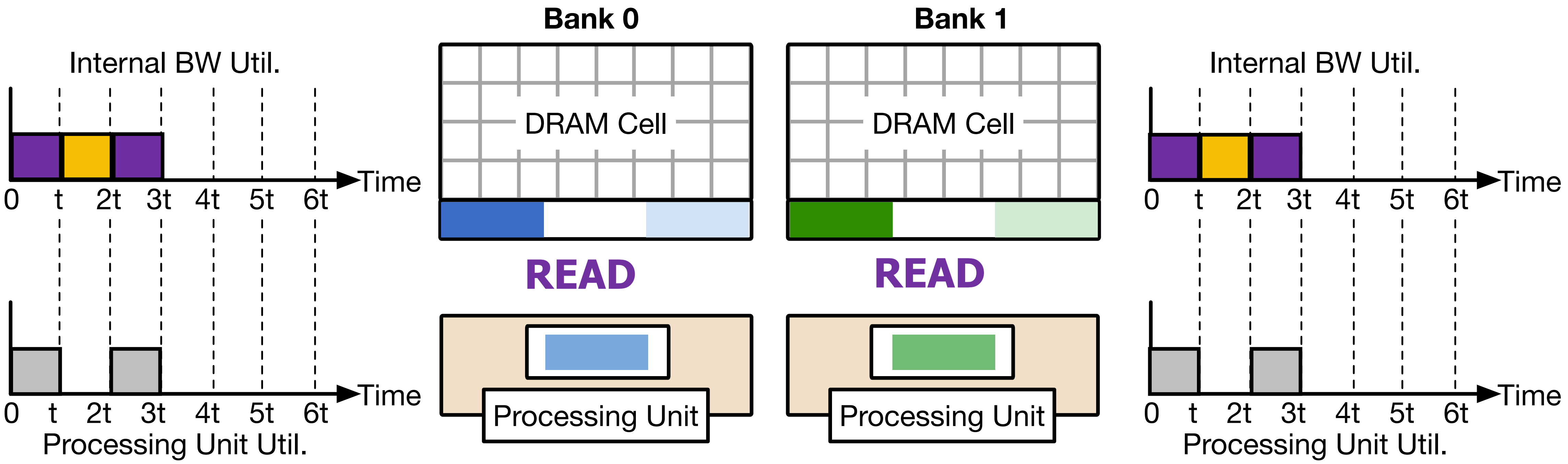
State Updates in Existing PIM

Access Interleaving
Quantization



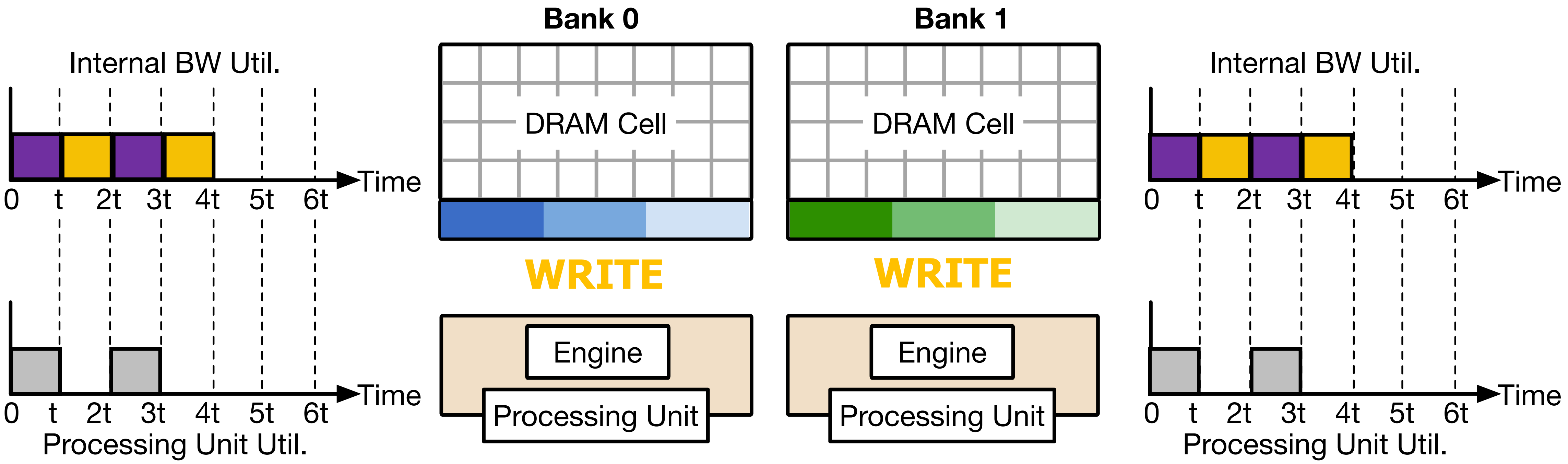
State Updates in Existing PIM

Access Interleaving
Quantization



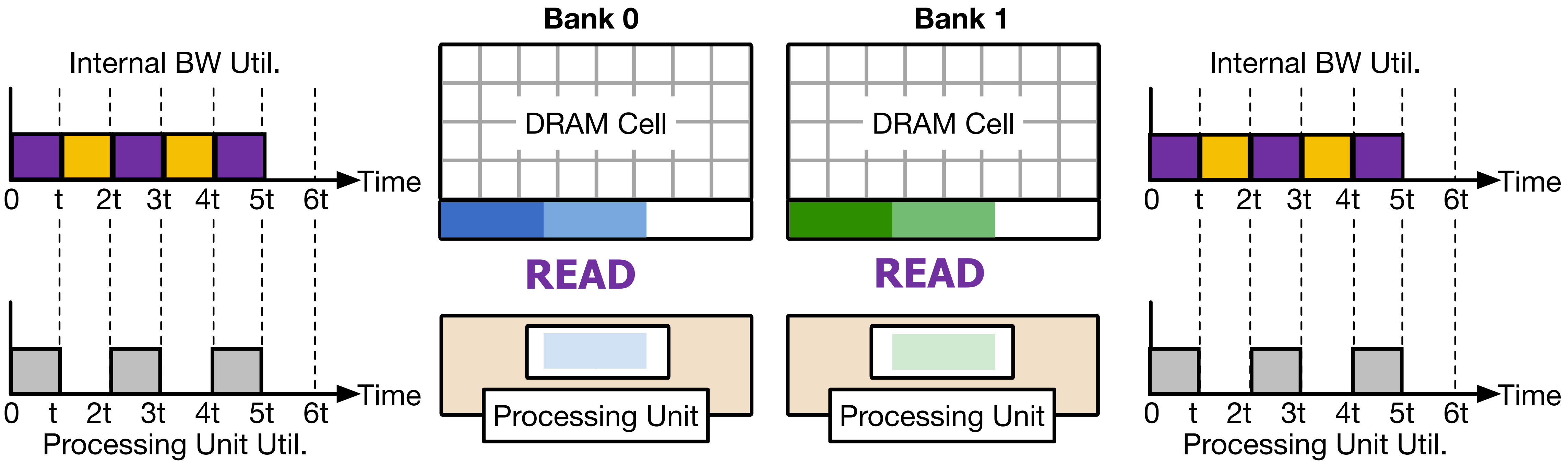
State Updates in Existing PIM

Access Interleaving
Quantization



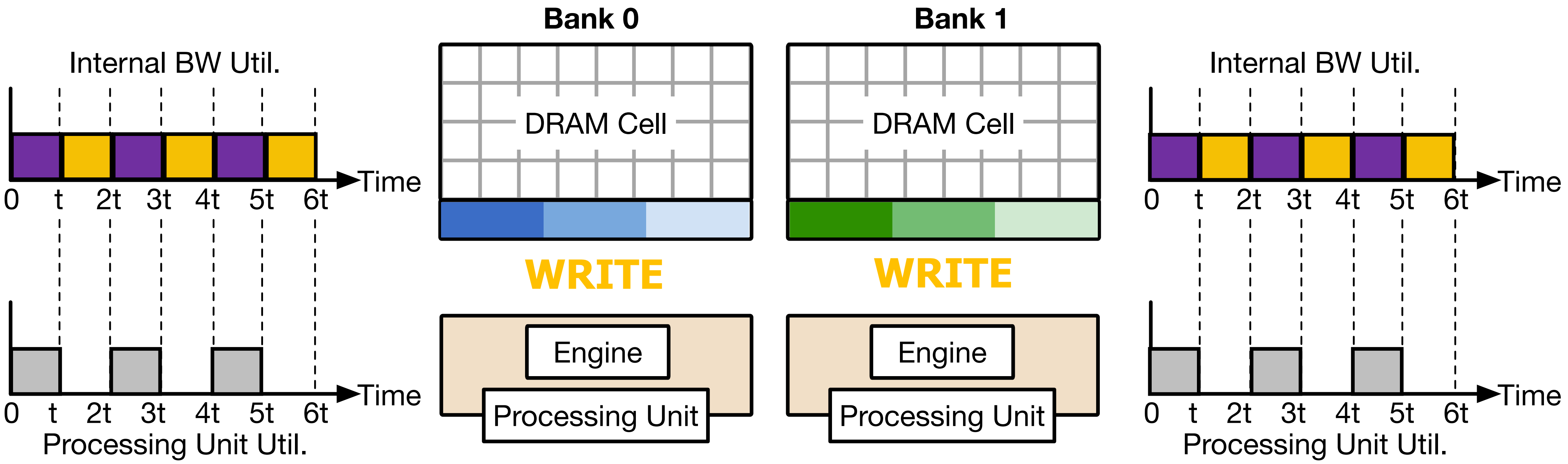
State Updates in Existing PIM

Access Interleaving
Quantization



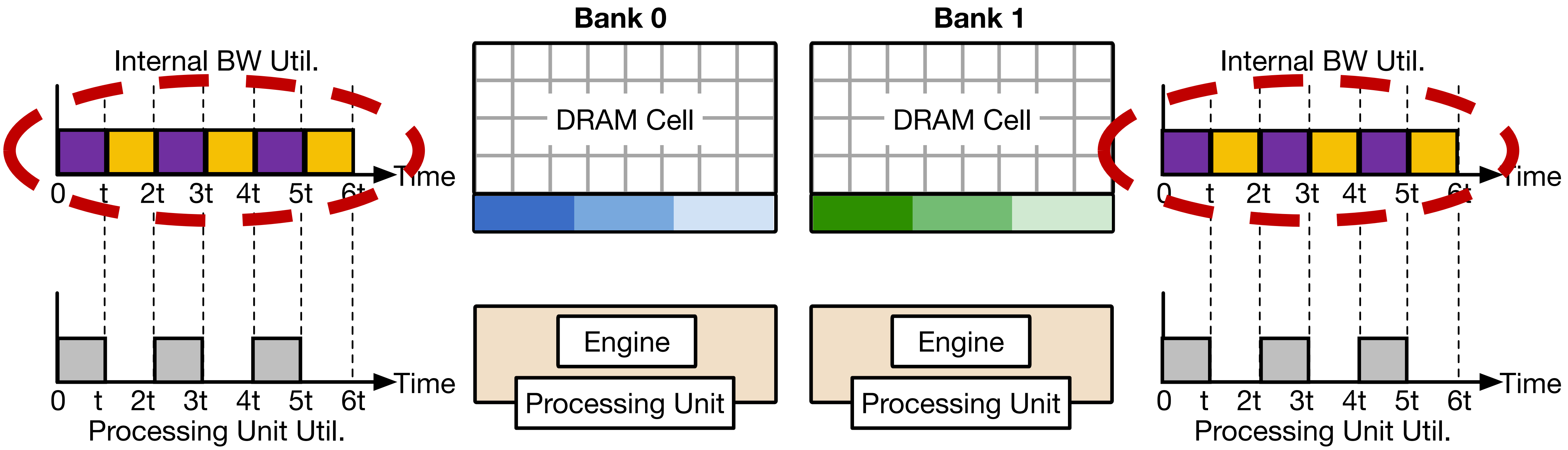
State Updates in Existing PIM

Access Interleaving
Quantization



State Updates in Existing PIM

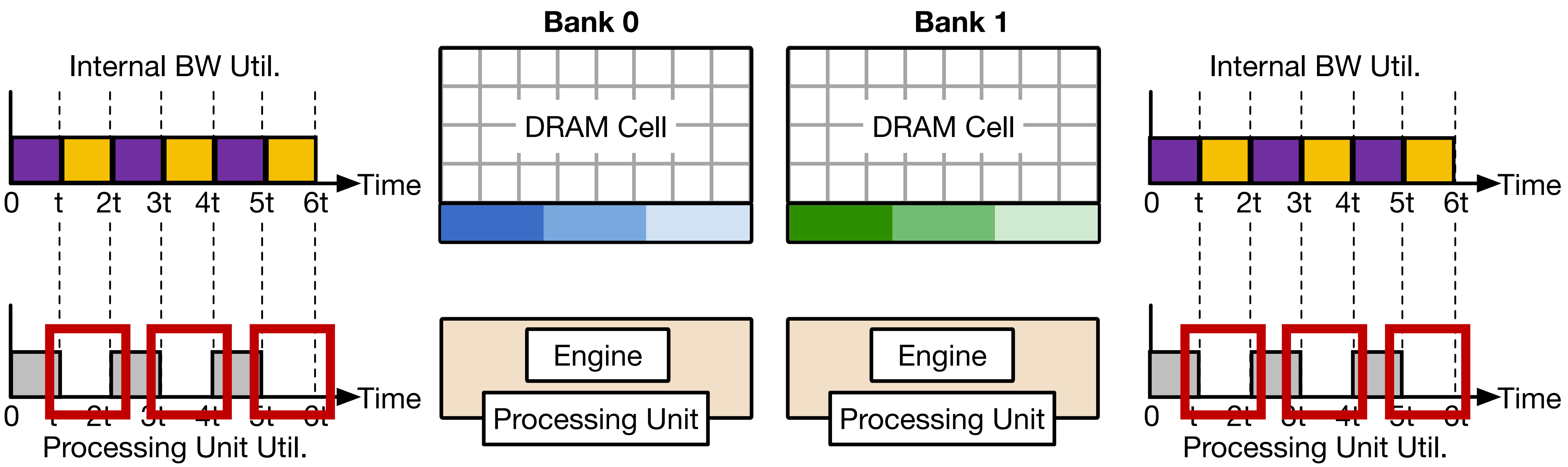
Access Interleaving
Quantization



Unlike dot-product operations,
state update operations require both reads and writes

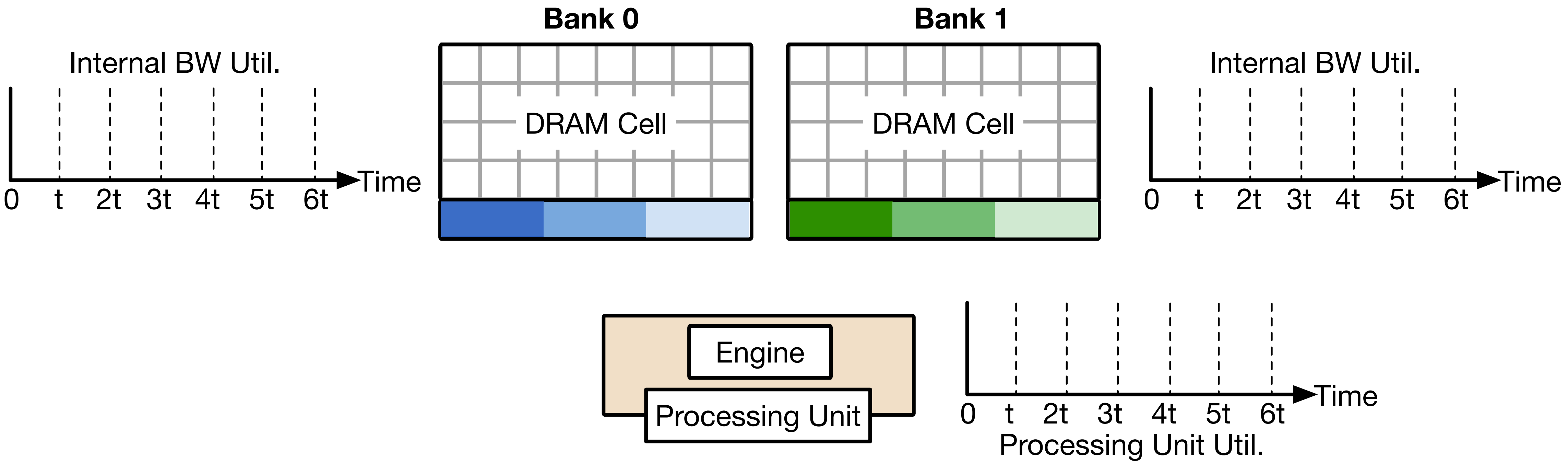
State Updates in Existing PIM

Access Interleaving
Quantization

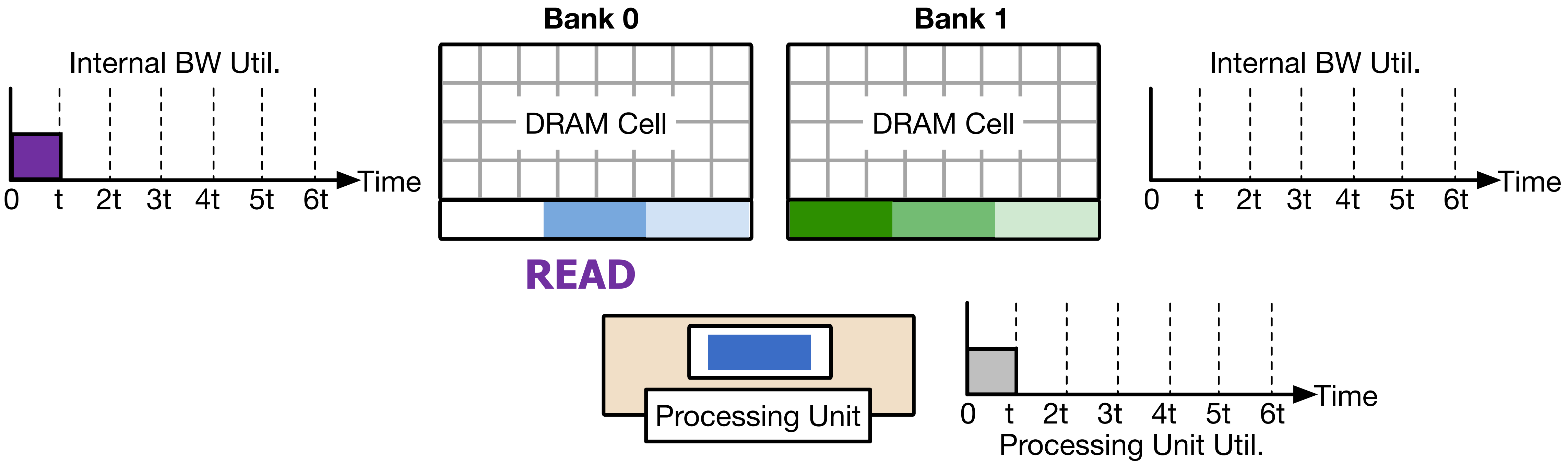


This leads to underutilization of processing units during writes

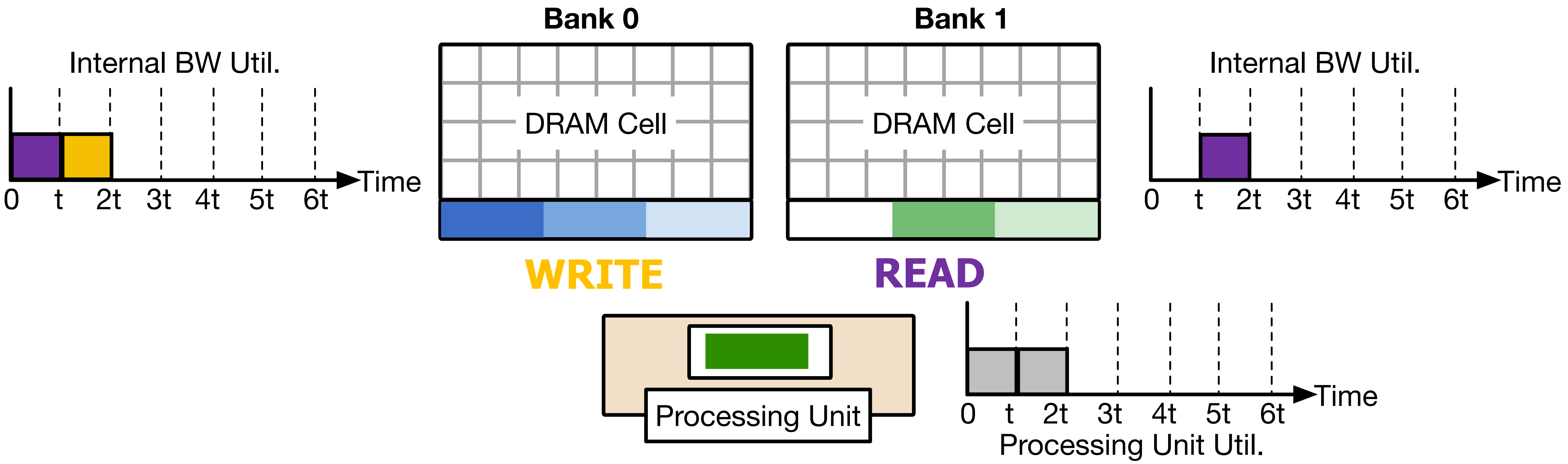
Access Interleaving



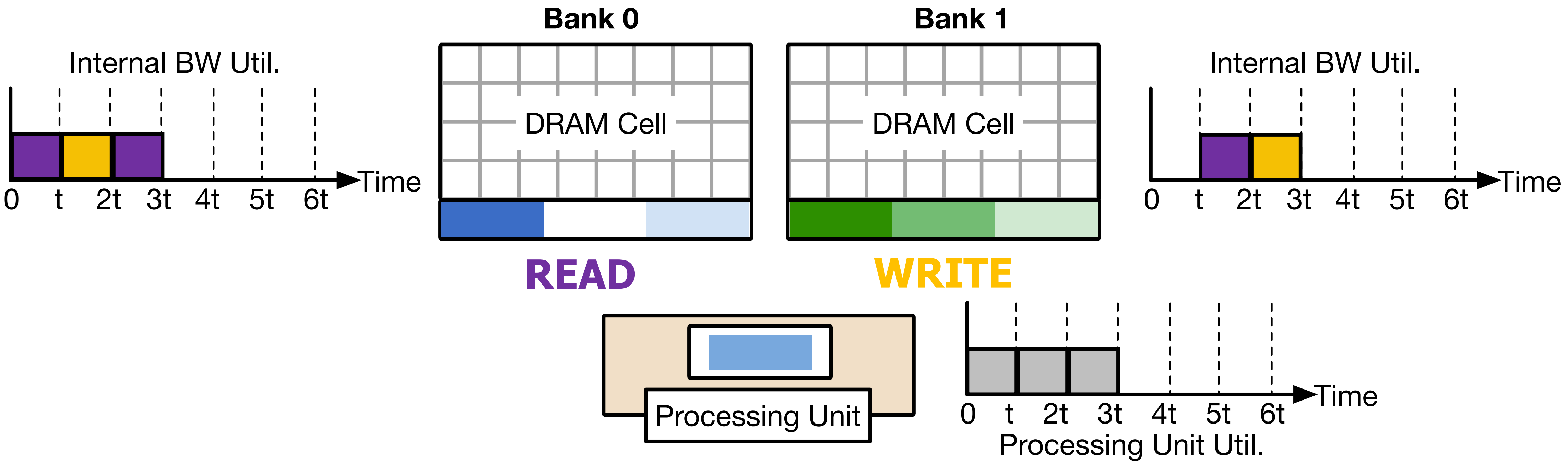
Access Interleaving



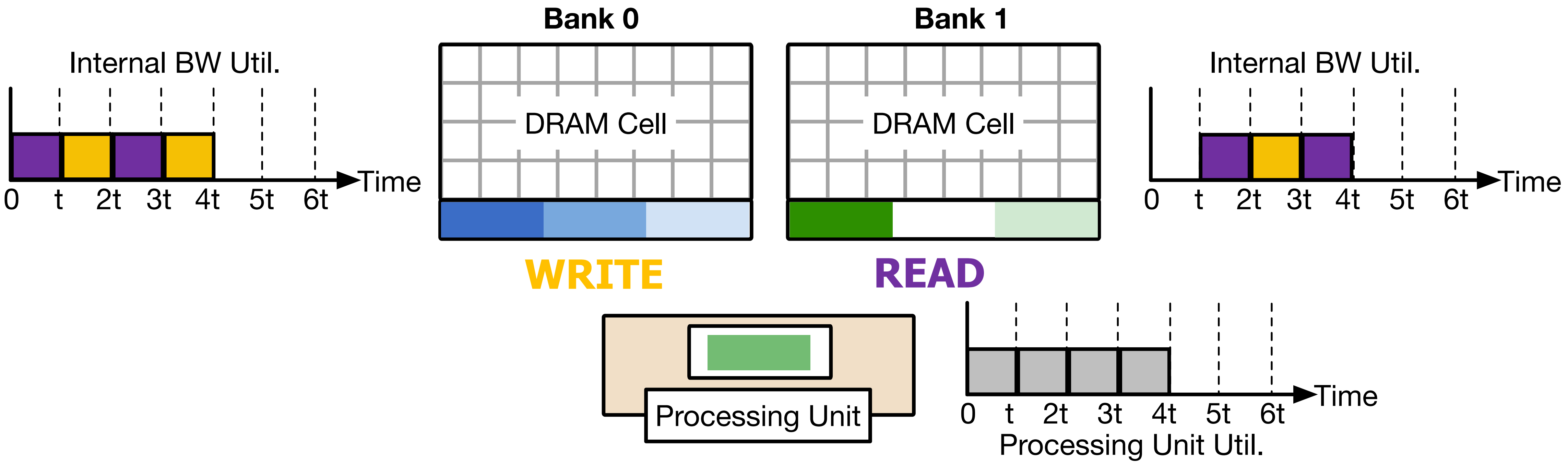
Access Interleaving



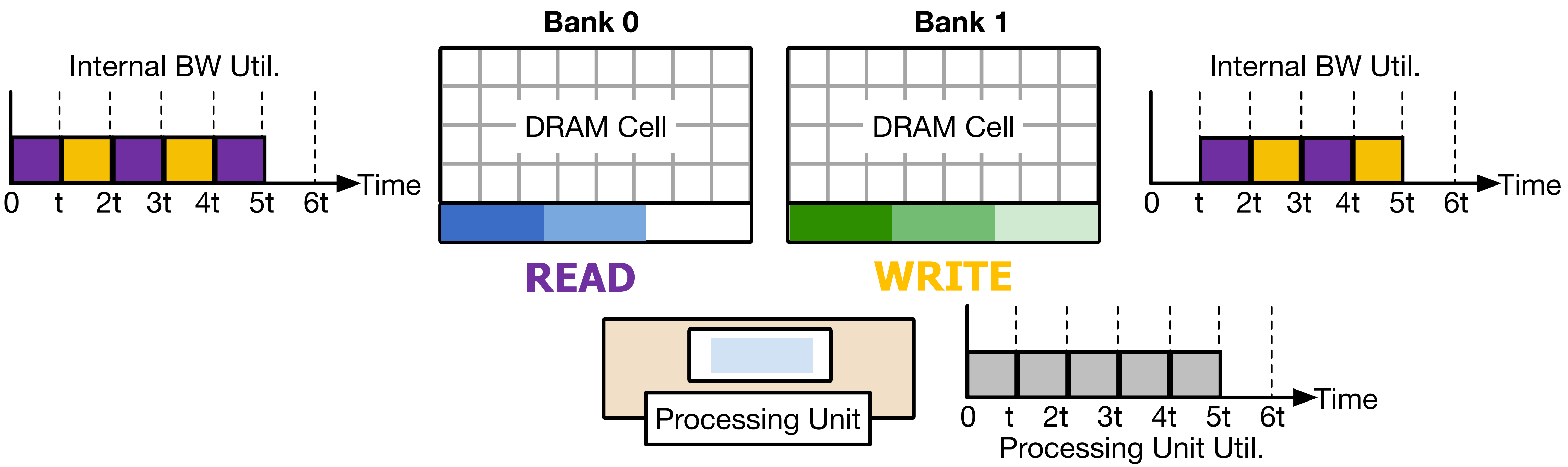
Access Interleaving



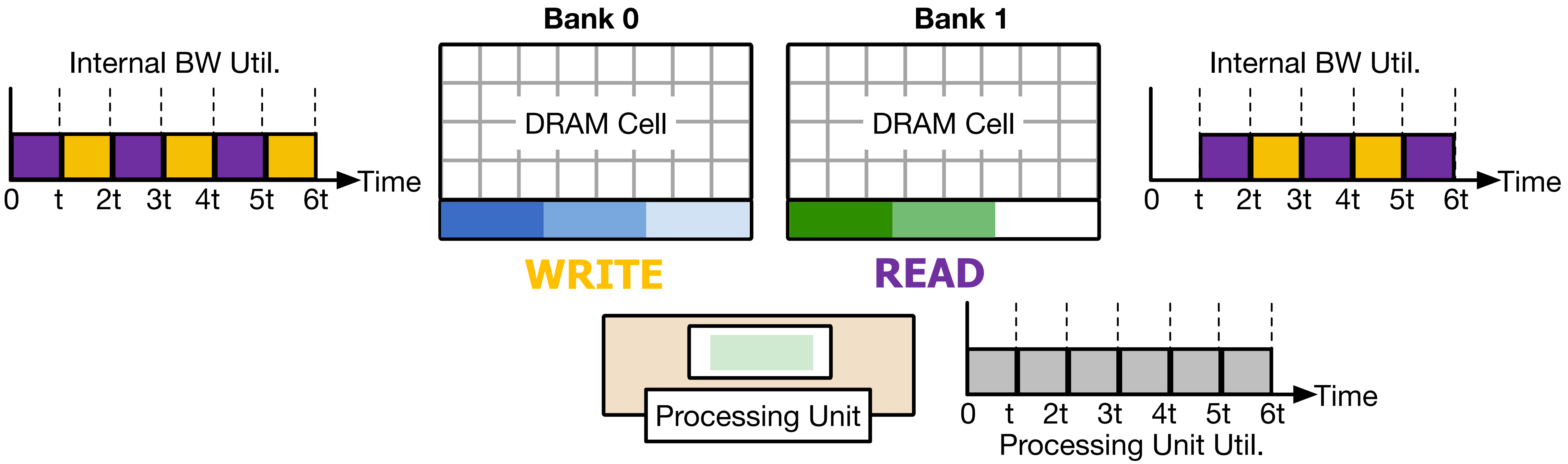
Access Interleaving



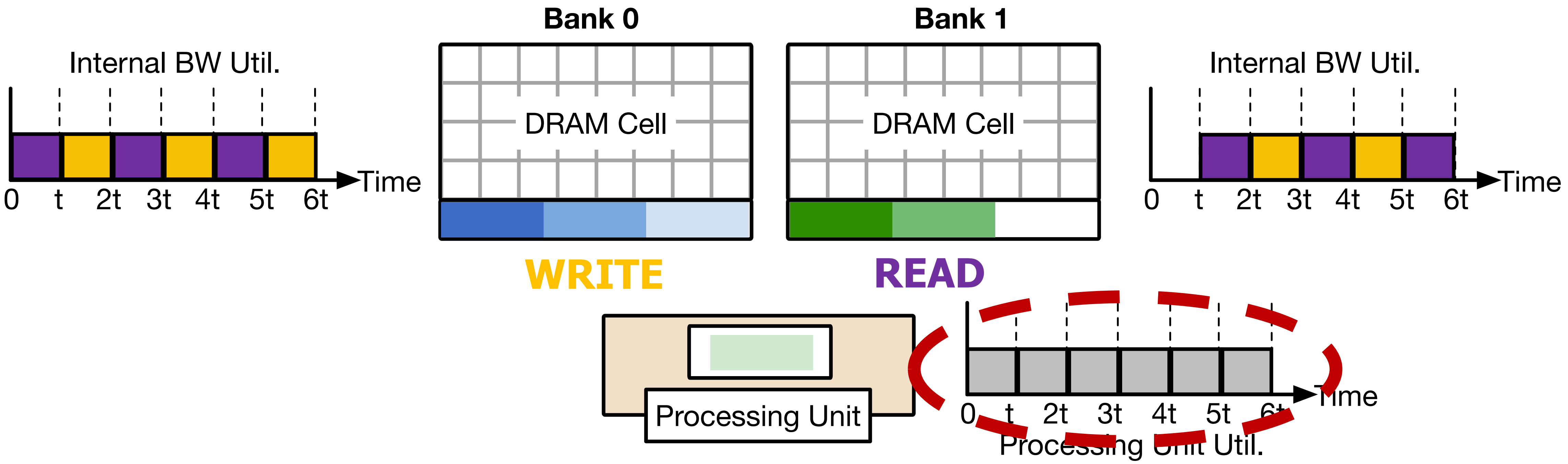
Access Interleaving



Access Interleaving



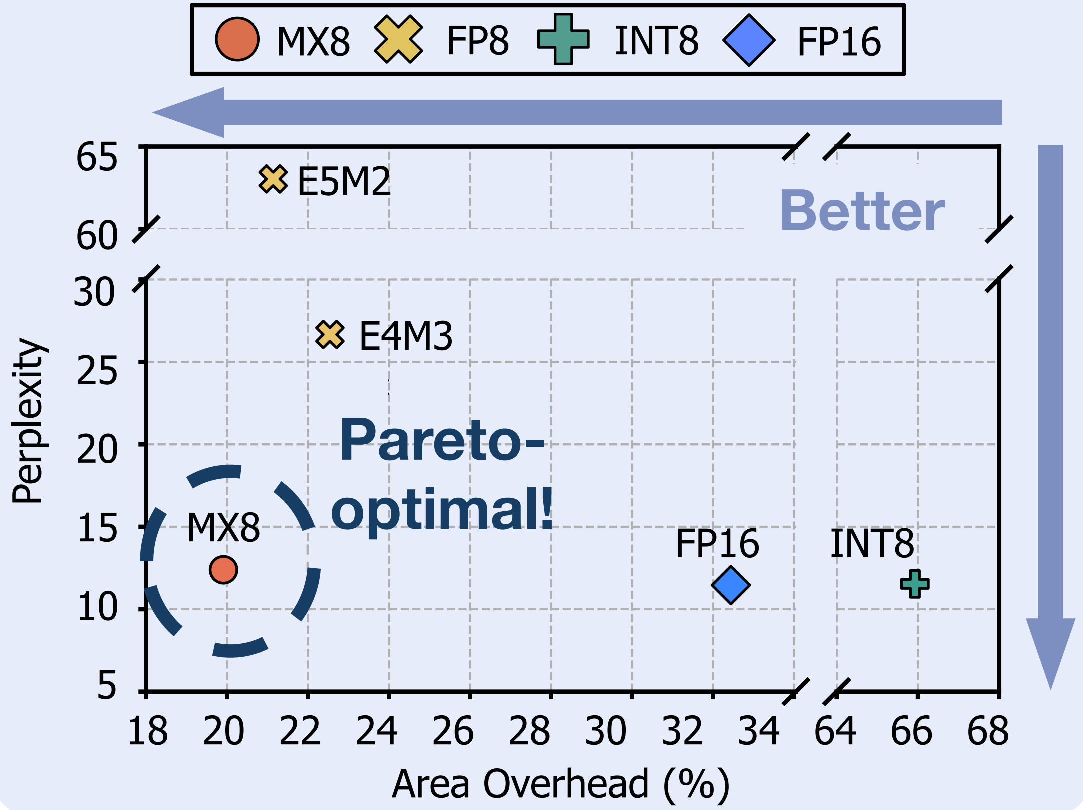
Access Interleaving



The processing units are now fully utilized, which in turn reduces the area overhead by half while maintaining the same throughput

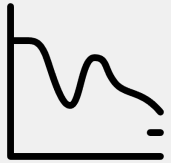
Quantization Analysis

Area-Accuracy Tradeoff



Floating Point Formats

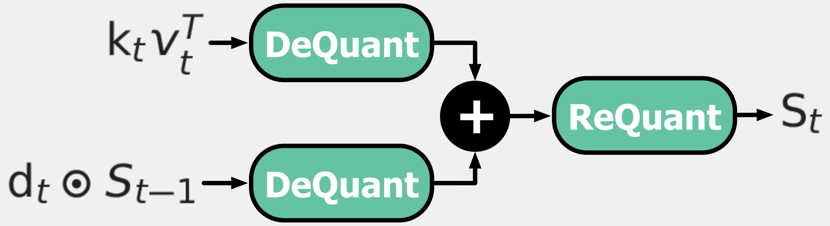
high accuracy drop



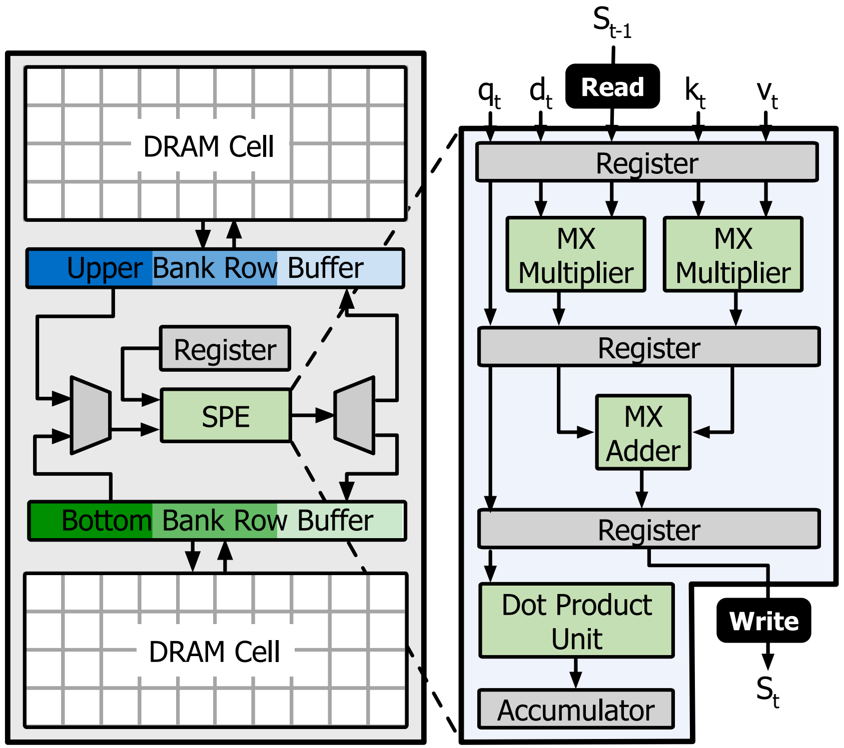
numerical fidelity issue

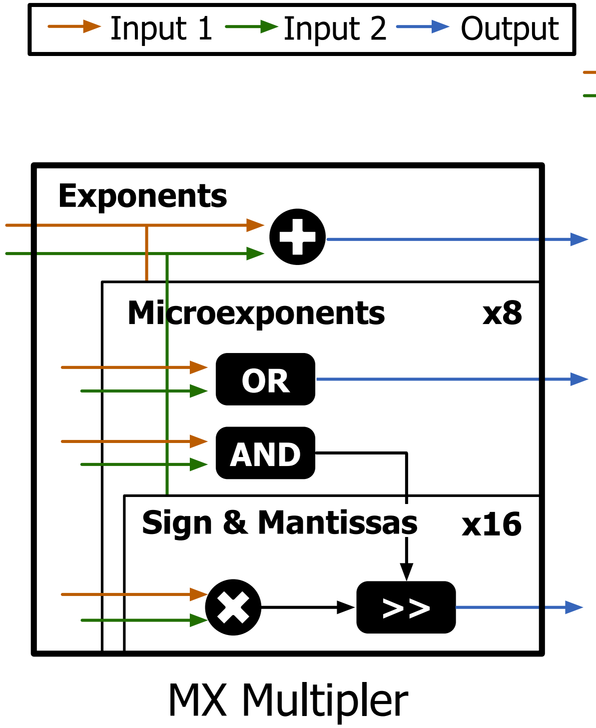
Integer Formats

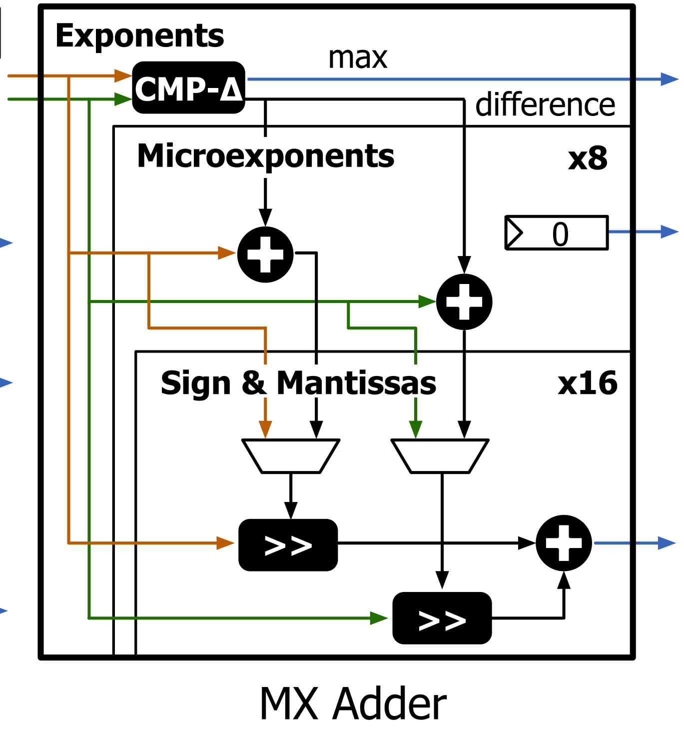
high area overhead



MX-based Processing Element







- MX operations require only simple addition, multiplication, and logic operations

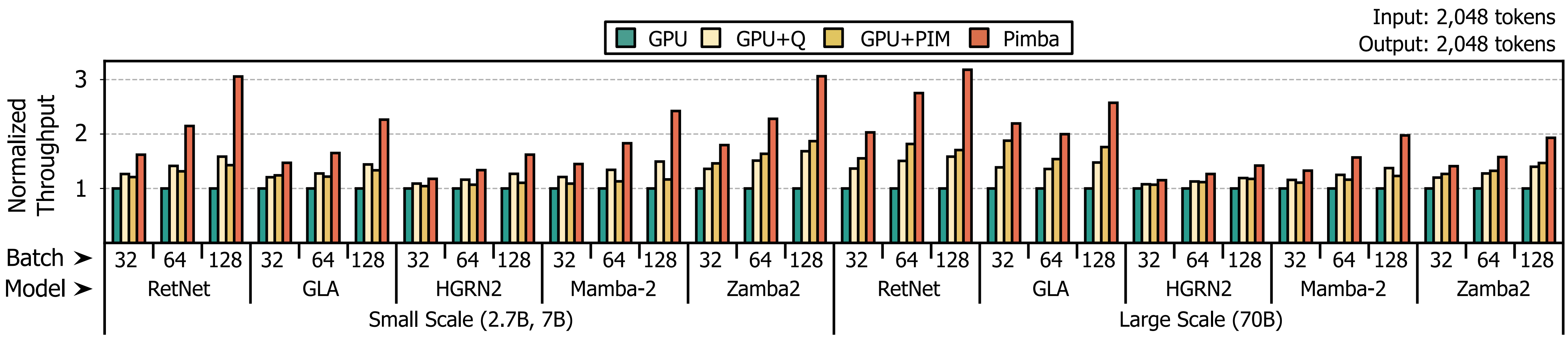
Experimental Methodology

- **Baselines**
  - **GPU**: NVIDIA A100 GPUs
  - GPU w/ Quantization (**GPU+Q**): A100 + 8-bit quantized state
  - GPU w/ HBM-PIM (**GPU+PIM**): A100 + Samsung HBM-PIM

- **Models**
  - RetNet, GLA, HGRN2, Mamba2, Zamba2
  - small-scale (2.7B, 7B)
  - large-scale (70B)

- **Simulation**
  - GPU: extends AttAcc system simulator
  - PIM: extends cycle-accurate Ramulator2

Throughput Results



- **PIMBA achieves** **14.6×** **faster state update operations compared to GPU**
- **PIMBA delivers** **up to 4.1×** **higher decoding throughput compared to GPU**

More Results on Paper

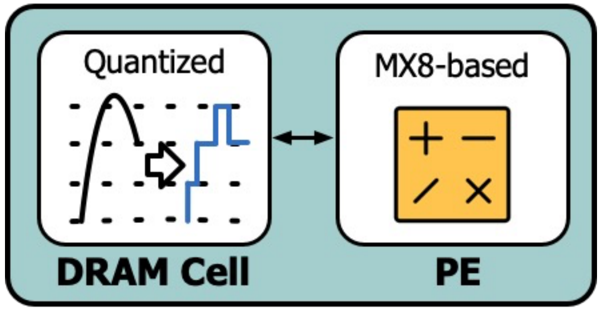
- Accuracy evaluation
- Performance improvements on attention-based transformers
- Decode phase latency breakdown
- Energy consumption
- RTL area and power overhead
- Comparison with existing PIM-based LLM serving system
- General adoption of PIMBA

Conclusion

- **PIMBA**
  - An efficient PIM-based post-transformer acceleration solution
- **Contributions**
  - We conduct first comprehensive study of post-transformer LLMs
  - We analyze unique characterizations of post-transformer LLMs
  - We propose novel access interleaving strategy and quantization-based PIM

Access Interleaving

Quantization



Code