Accelerating Pointer Chasing in 3D-Stacked Memory: *Challenges, Mechanisms, Evaluation*

Kevin Hsieh

Samira Khan, Nandita Vijaykumar, Kevin K. Chang,

Amirali Boroumand, Saugata Ghose, Onur Mutlu

Carnegie Mellon University

University of Virginia

ETH zürich

SAFARI

Executive Summary

- **Our Goal:** Accelerating pointer chasing inside main memory
- **Challenges:** Parallelism challenge and Address translation challenge
- **Our Solution:** In-Memory PoInter Chasing Accelerator (IMPICA)
  - Address-access decoupling: enabling parallelism in the accelerator with low cost
  - IMPICA page table: low cost page table structure
- **Key Results**:
  - 1.2X – 1.9X speedup for pointer chasing operations, +16% database throughput
  - 6% - 41% reduction in energy consumption

Linked Data Structures

- Linked data structures are widely used in many important applications

Linked Data Structures

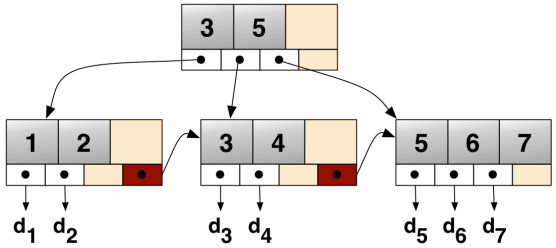
- Linked data structures are widely used in many important applications

Database

Linked Data Structures

- Linked data structures are widely used in many important applications

Database



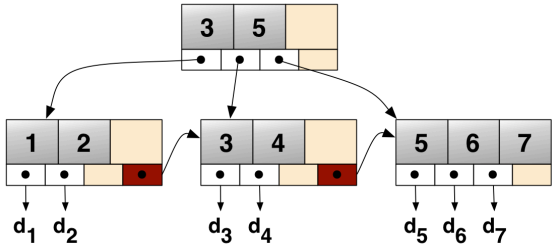
B-Tree

Linked Data Structures

- Linked data structures are widely used in many important applications

Database

Key-value stores



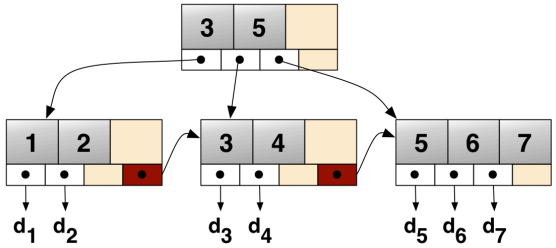
B-Tree

Linked Data Structures

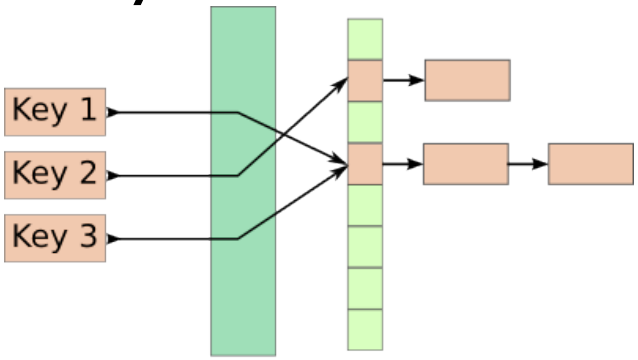
- Linked data structures are widely used in many important applications

Database

Key-value stores



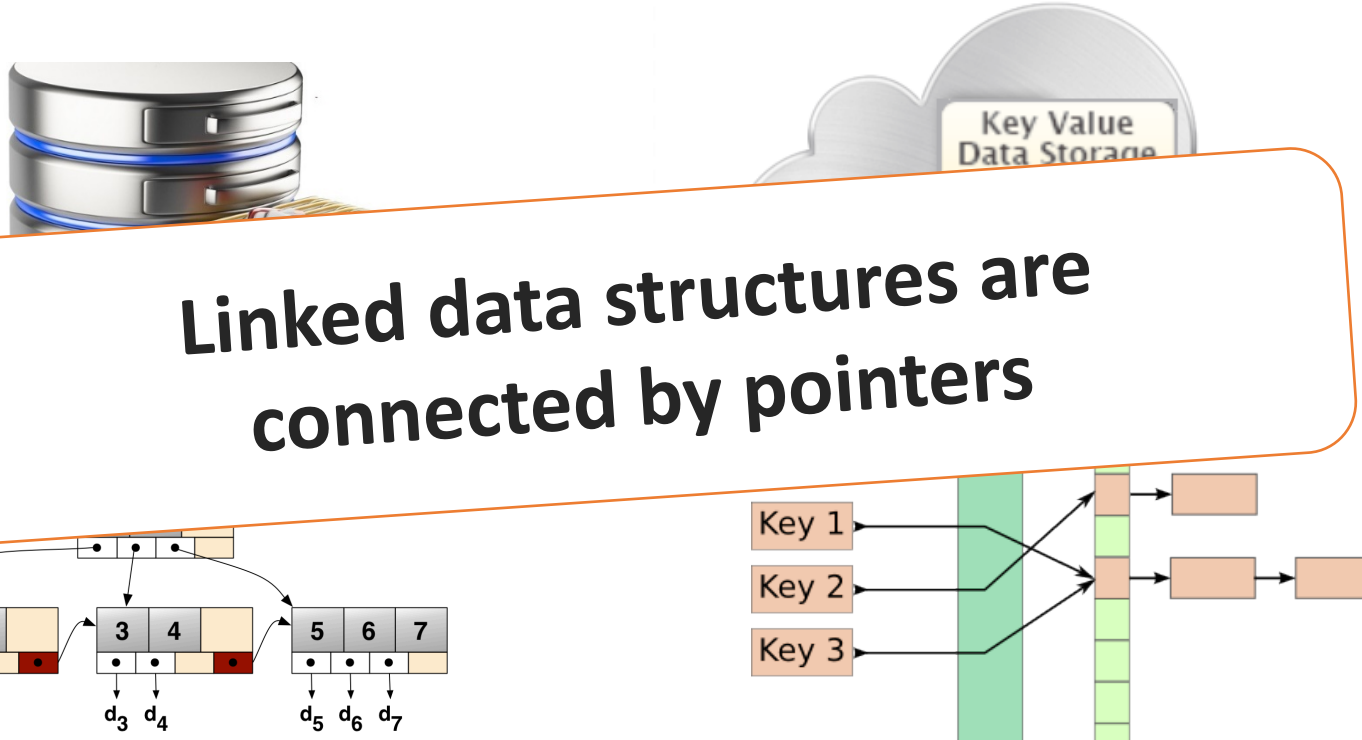
B-Tree



Hash Table

Linked Data Structures

- Linked data structures are widely used in many important applications



B-Tree

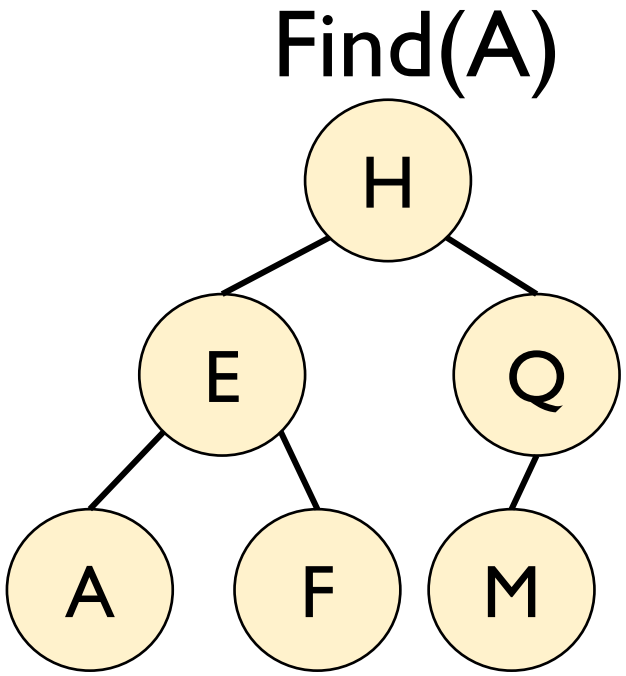
Hash Table

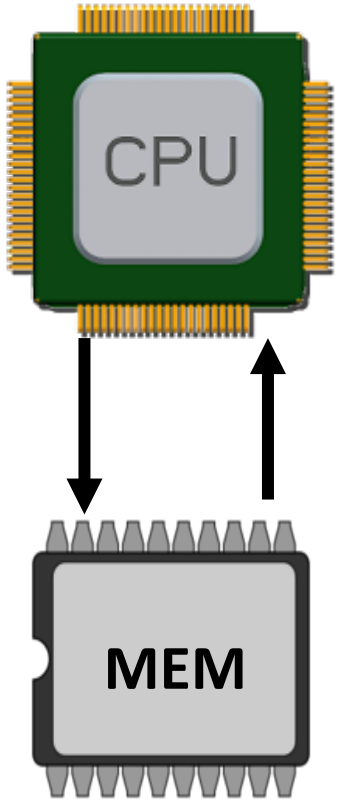
The Problem: Pointer Chasing

- Traversing linked data structures requires chasing pointers

The Problem: Pointer Chasing

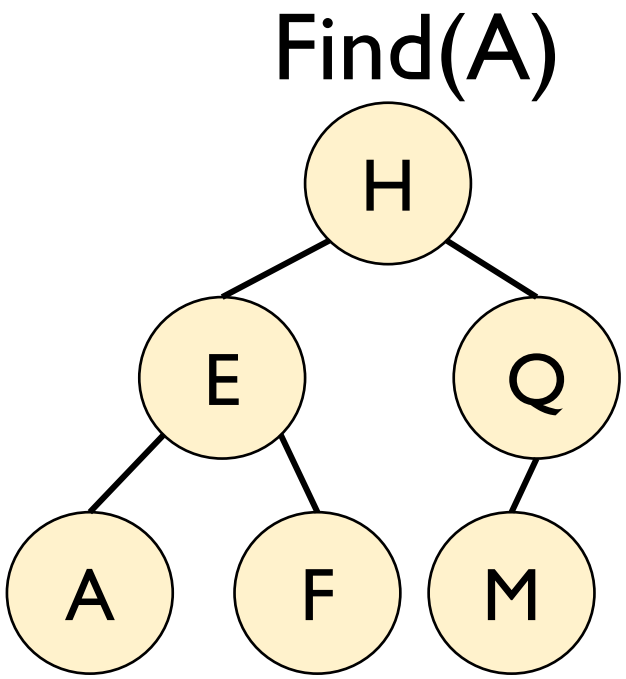
- Traversing linked data structures requires chasing pointers

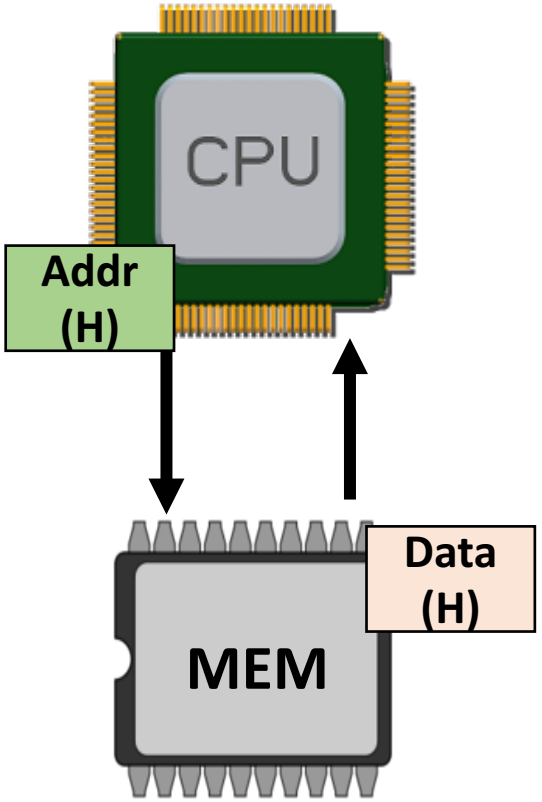




The Problem: Pointer Chasing

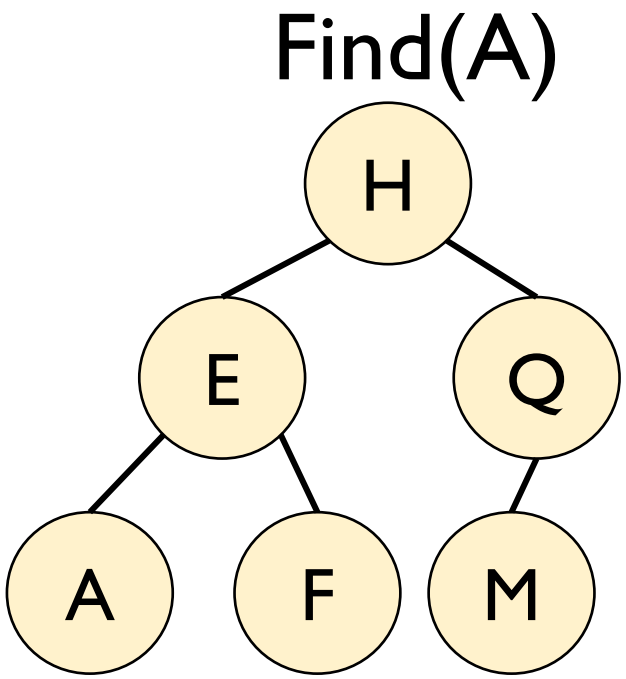
- Traversing linked data structures requires chasing pointers

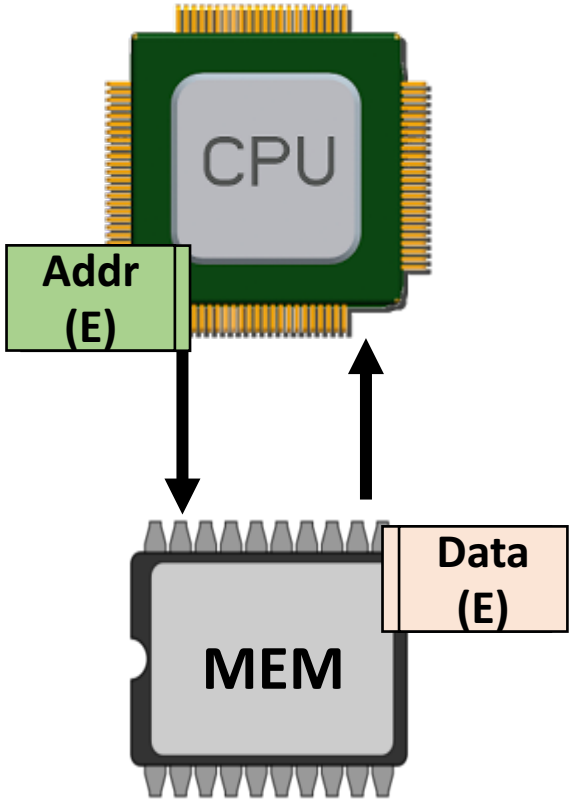




The Problem: Pointer Chasing

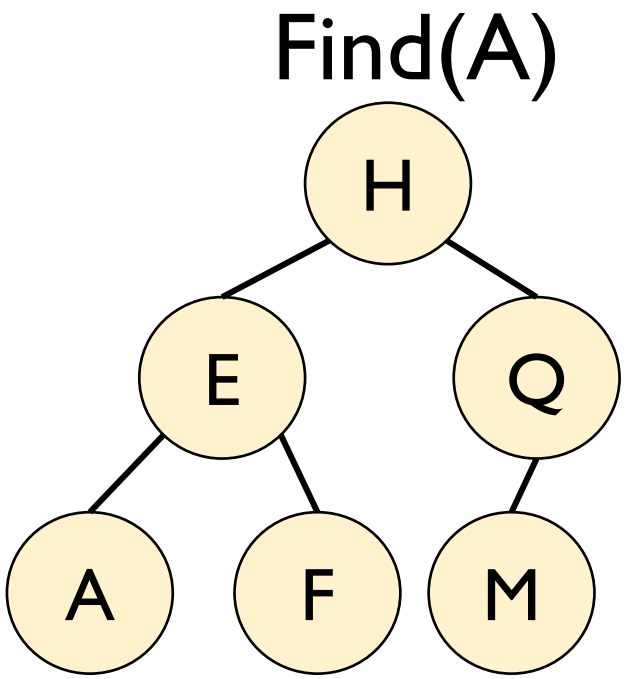
- Traversing linked data structures requires chasing pointers

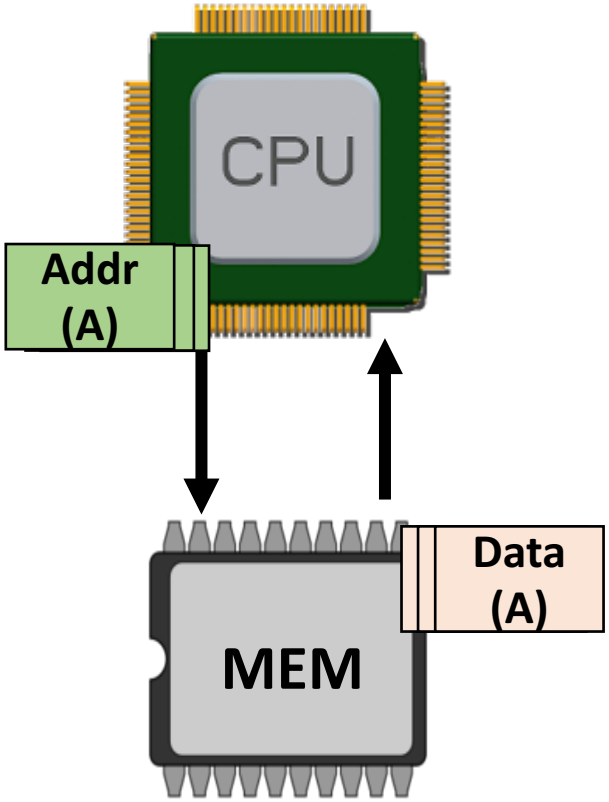




The Problem: Pointer Chasing

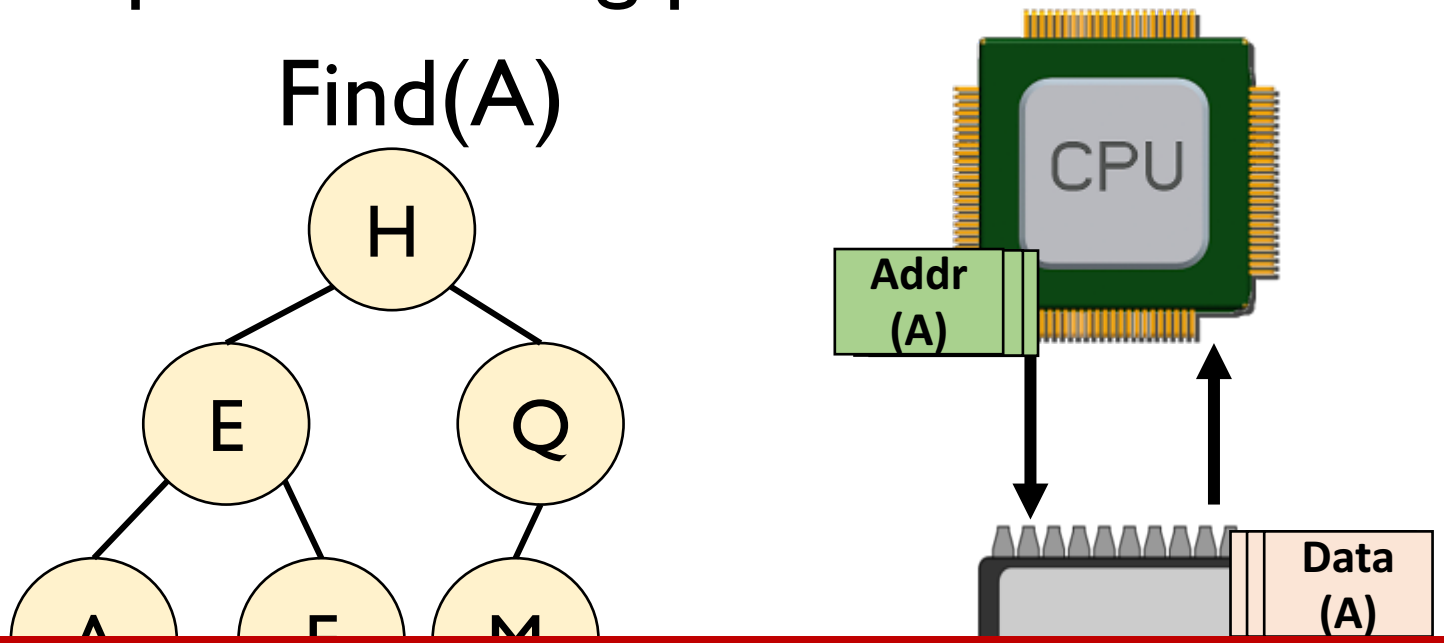
- Traversing linked data structures requires chasing pointers





The Problem: Pointer Chasing

- Traversing linked data structures requires chasing pointers



Serialized and irregular access pattern
6X cycles per instruction in real workloads

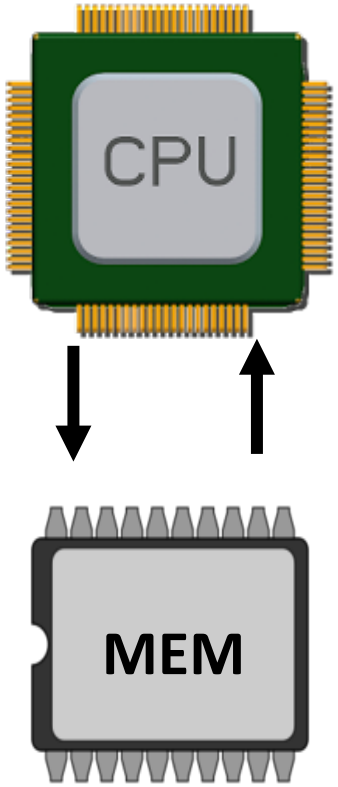
Our Goal

Our Goal

Accelerating pointer chasing inside main memory

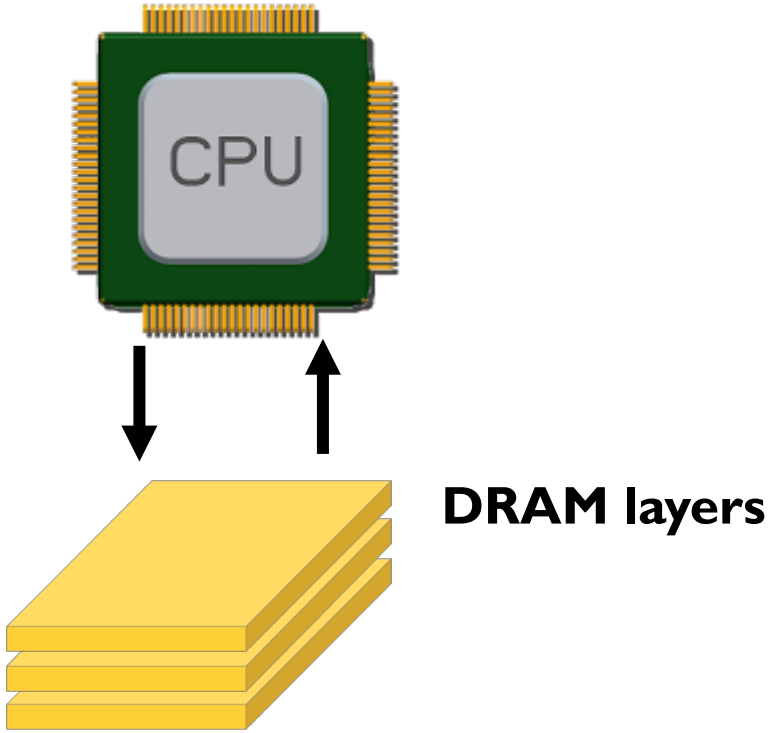
Our Goal

Accelerating pointer chasing inside main memory



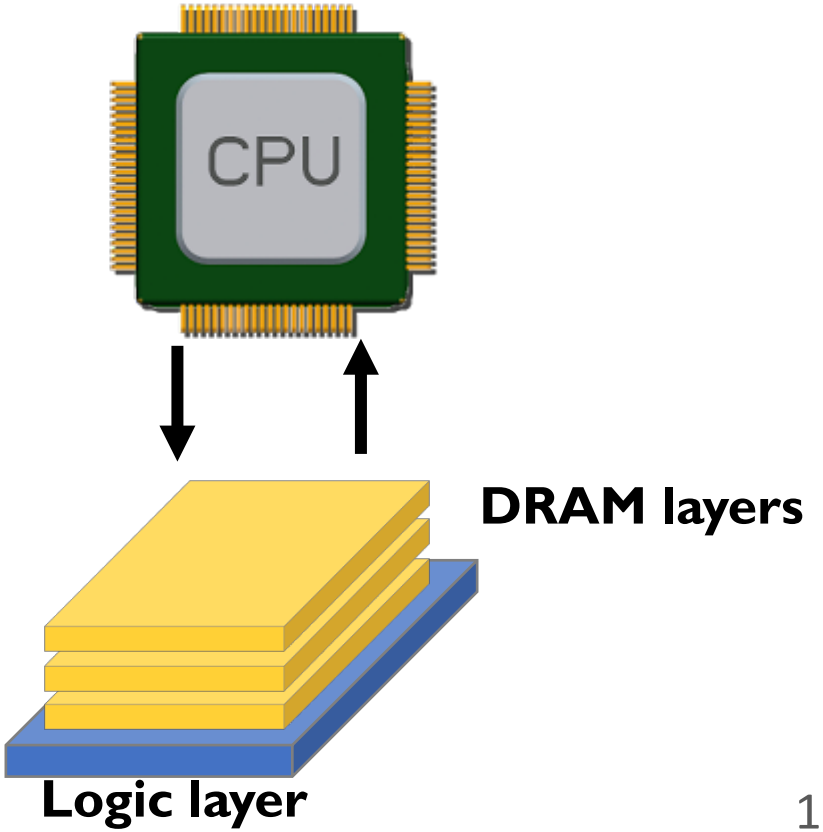
Our Goal

Accelerating pointer chasing inside main memory



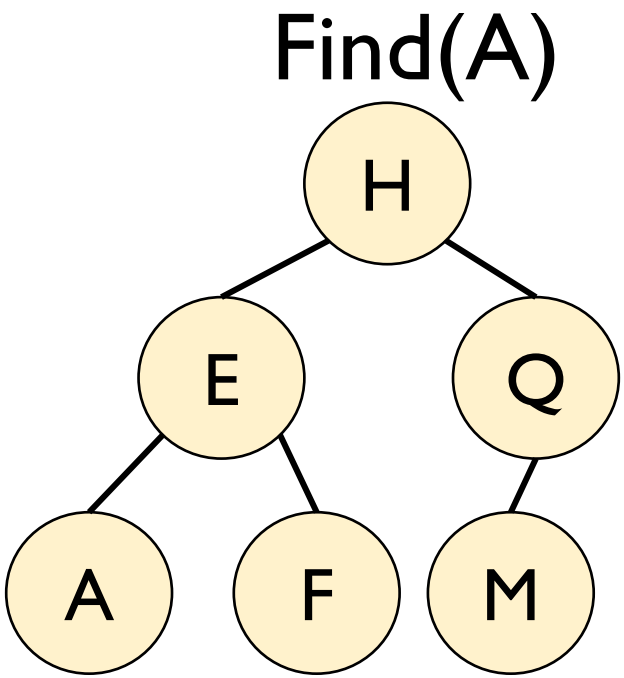
Our Goal

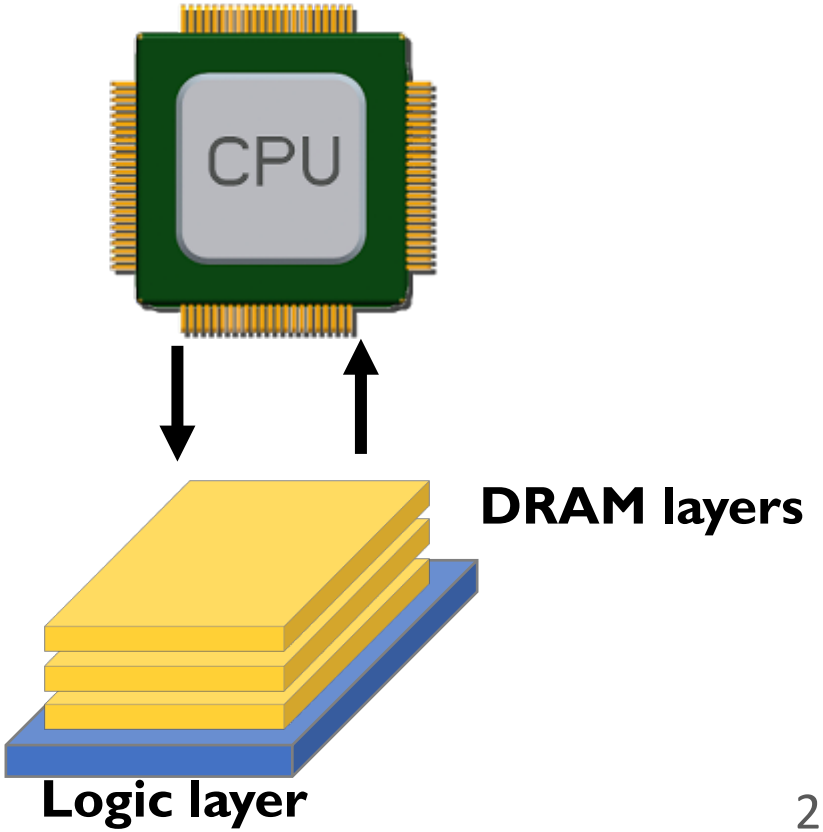
Accelerating pointer chasing inside main memory



Our Goal

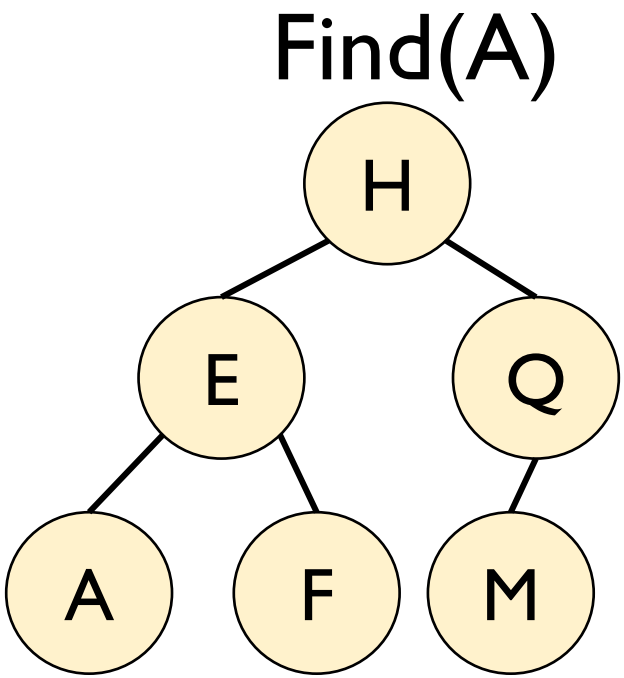
Accelerating pointer chasing inside main memory

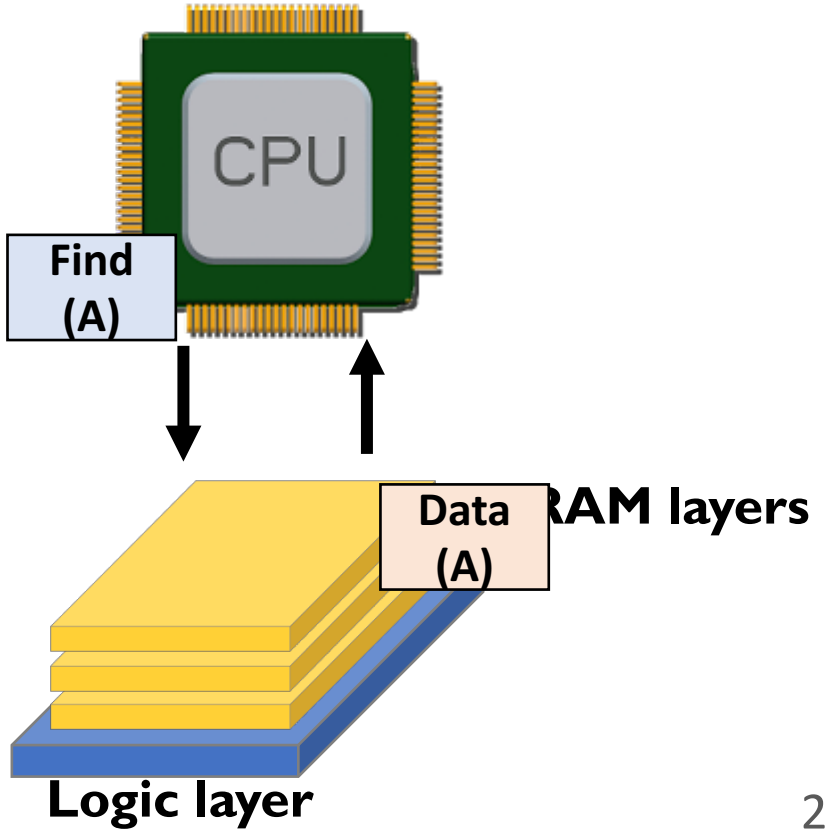




Our Goal

Accelerating pointer chasing inside main memory



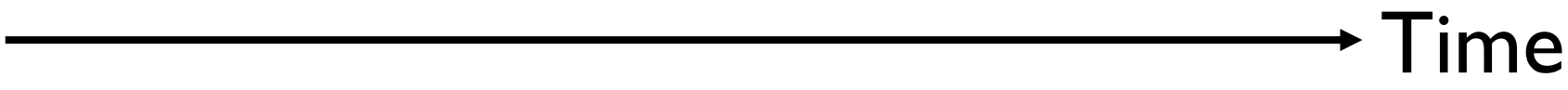


Outline

- Motivation and Our Approach
- Parallelism Challenge
- IMPICA Core Architecture
- Address Translation Challenge
- IMPICA Page Table
- Evaluation
- Conclusion

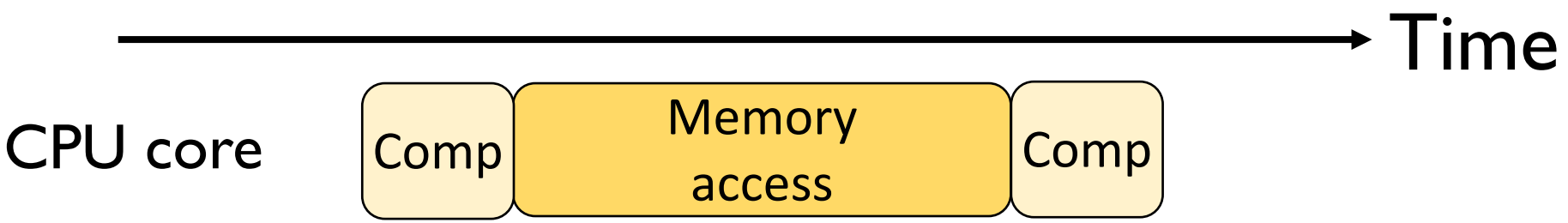
Parallelism Challenge

Parallelism Challenge

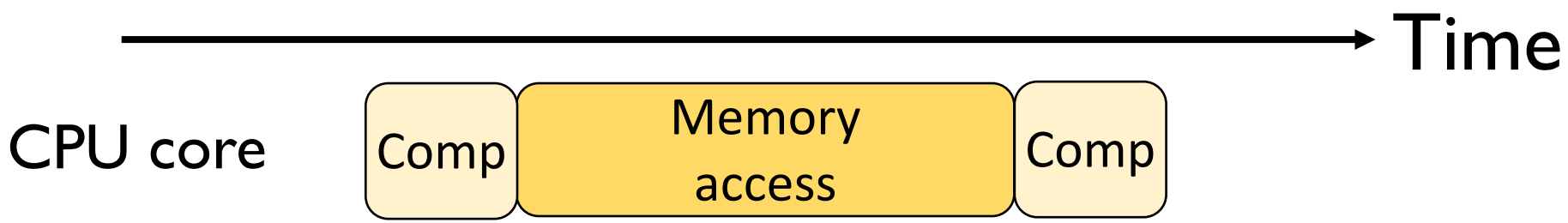


CPU core

Parallelism Challenge

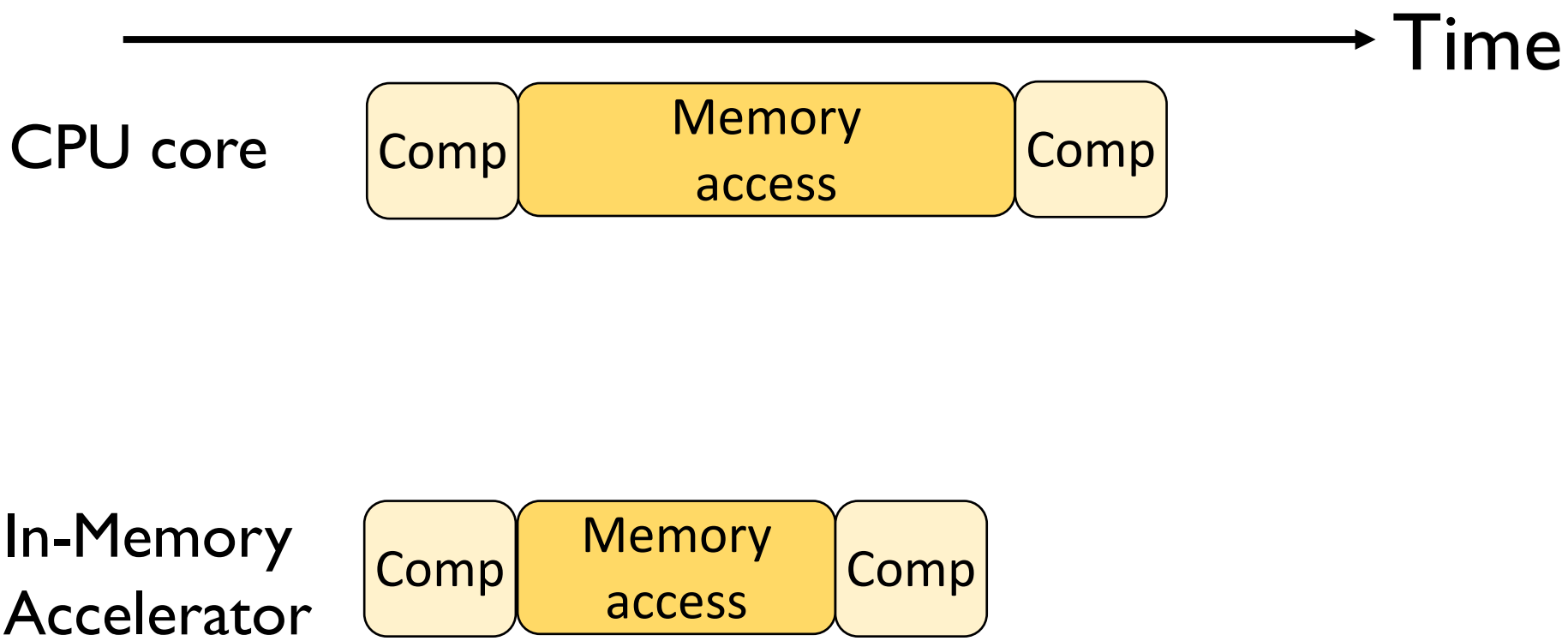


Parallelism Challenge

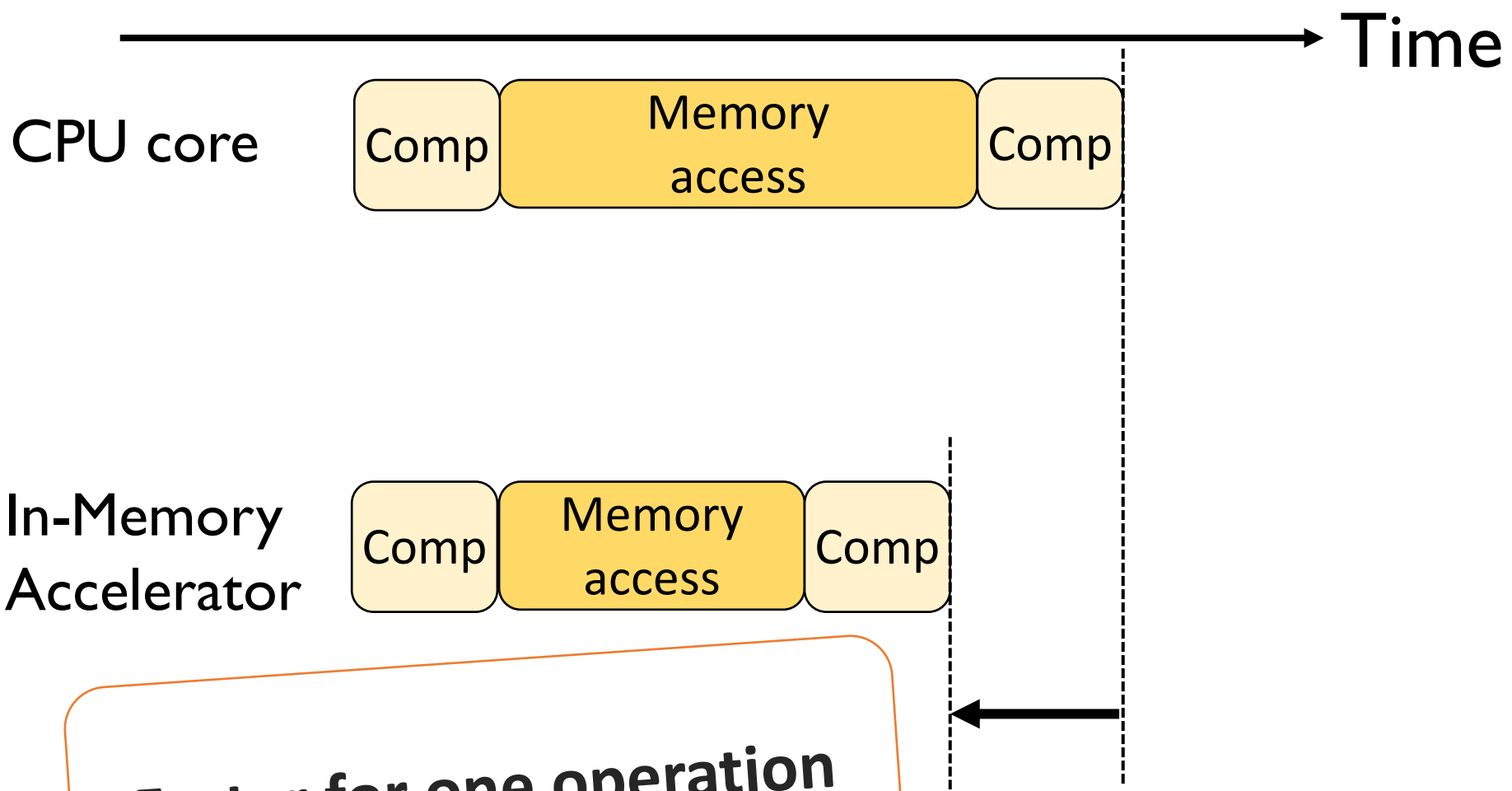


In-Memory Accelerator

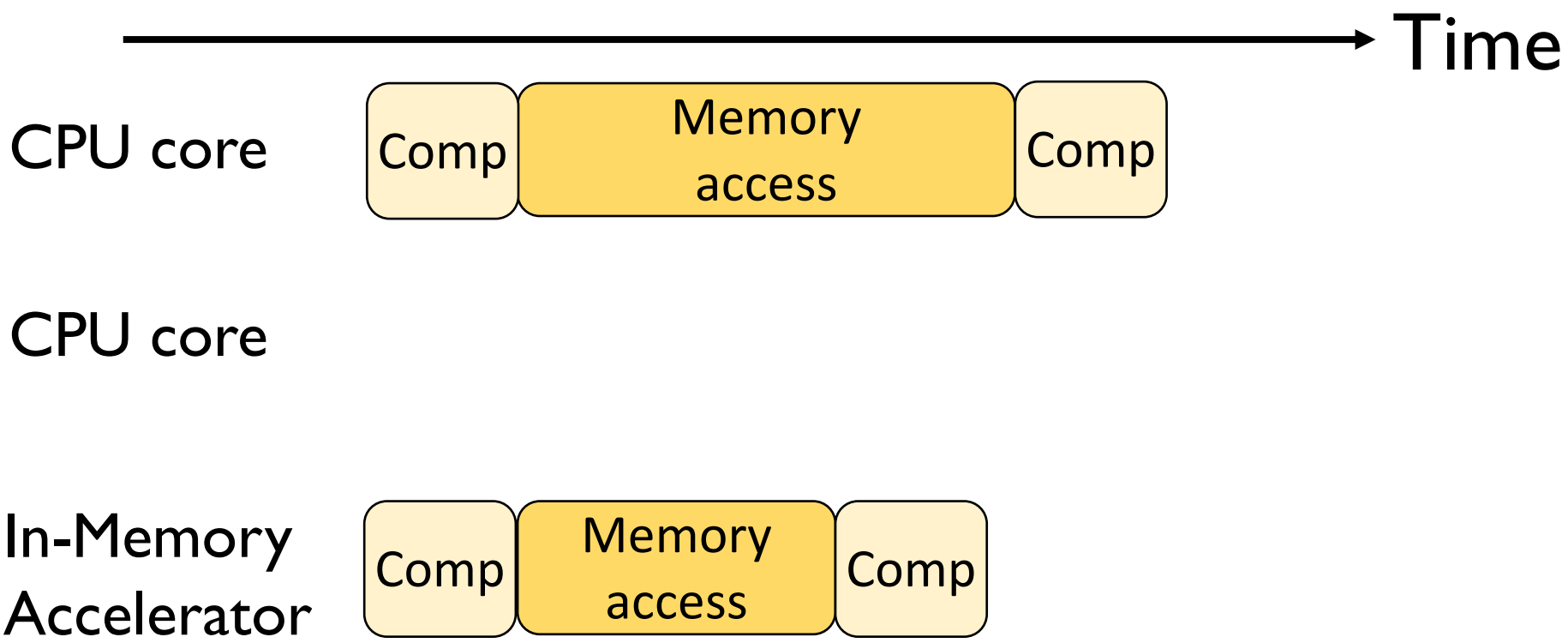
Parallelism Challenge



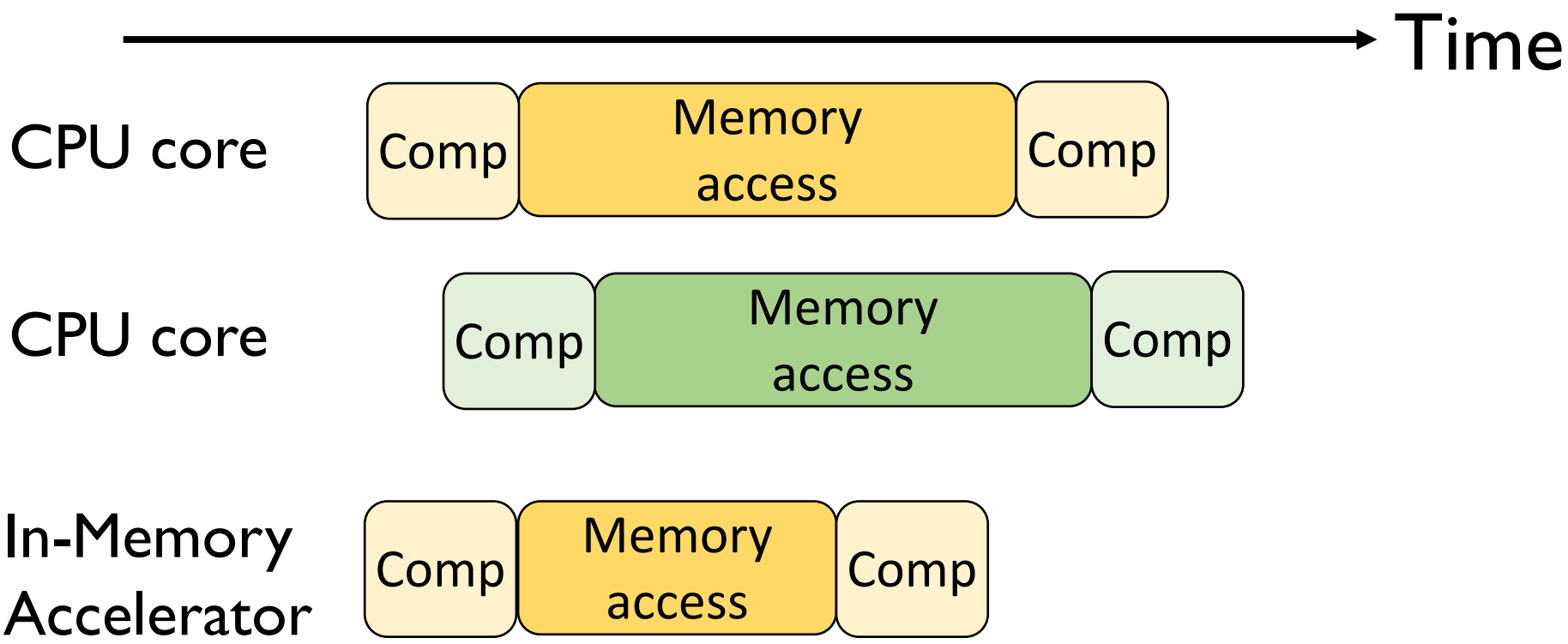
Parallelism Challenge



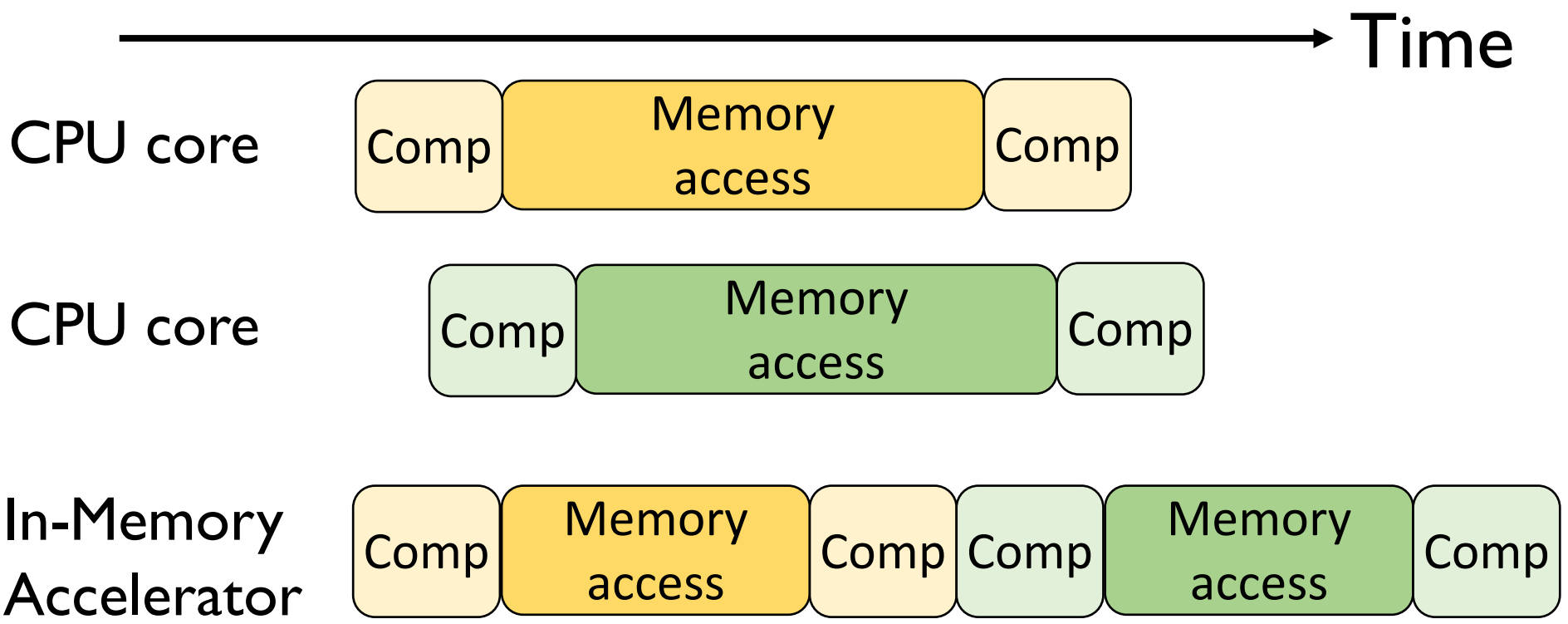
Parallelism Challenge



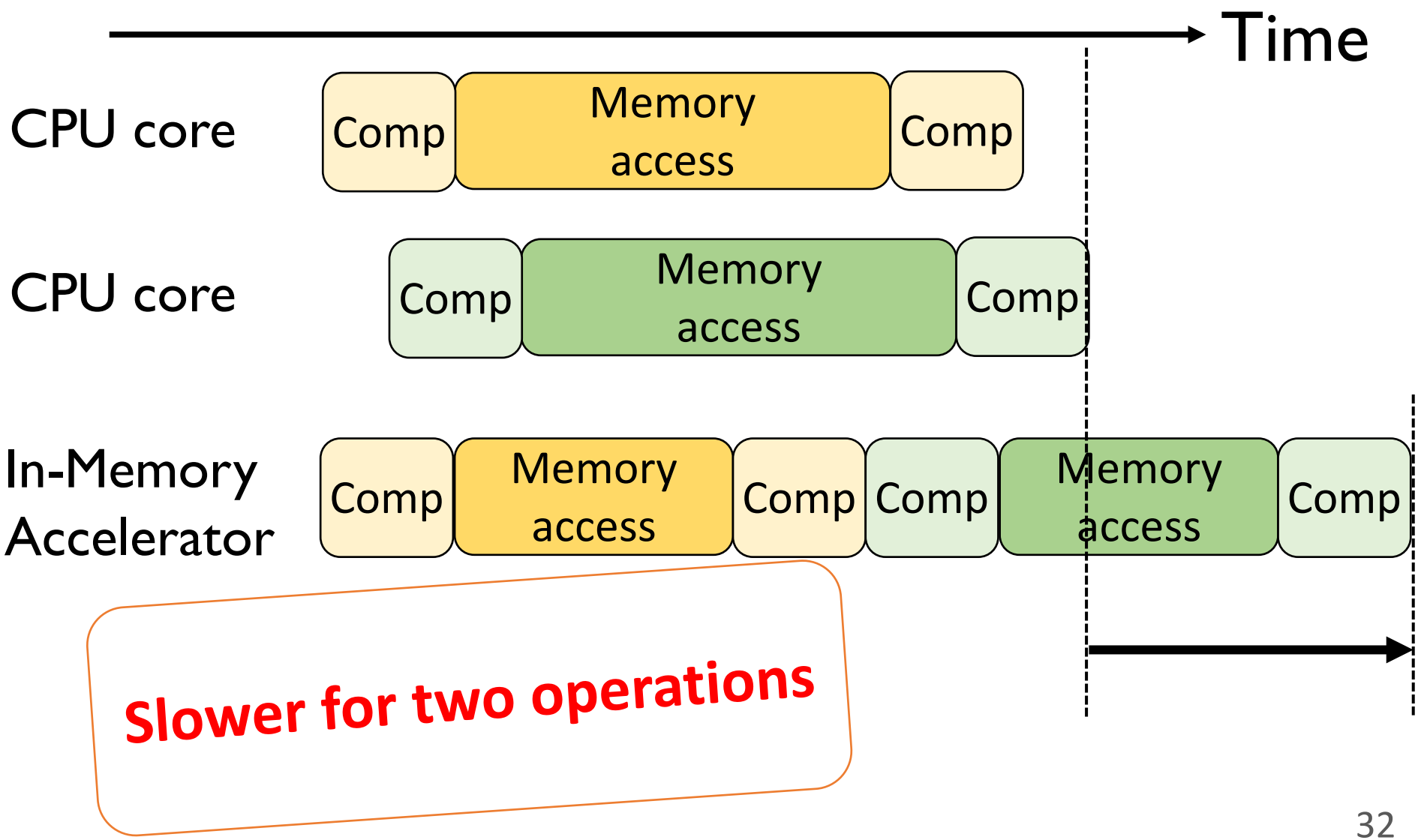
Parallelism Challenge



Parallelism Challenge



Parallelism Challenge

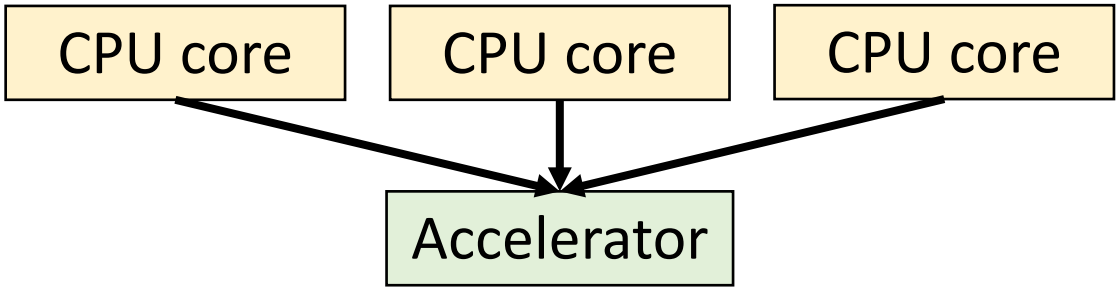


Parallelism Challenge and Opportunity

- A simple in-memory accelerator can still be slower than multiple CPU cores

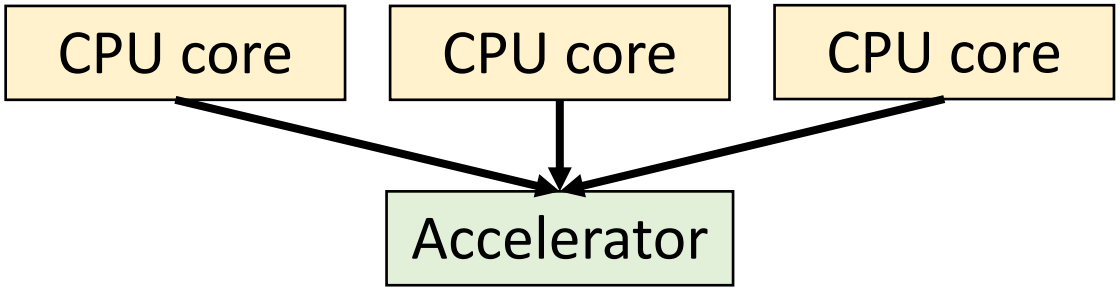
Parallelism Challenge and Opportunity

- A simple in-memory accelerator can still be slower than multiple CPU cores



Parallelism Challenge and Opportunity

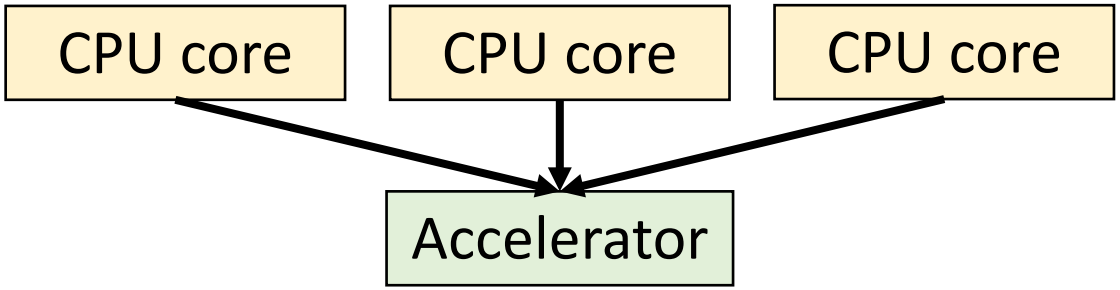
- A simple in-memory accelerator can still be slower than multiple CPU cores



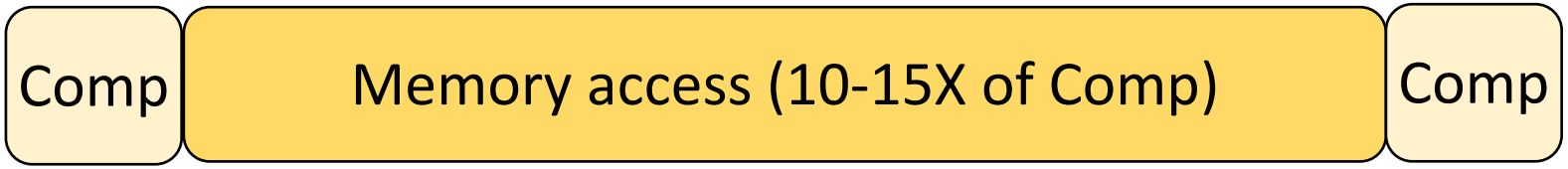
- Opportunity: a pointer-chasing accelerator spends a long time waiting for memory

Parallelism Challenge and Opportunity

- A simple in-memory accelerator can still be slower than multiple CPU cores

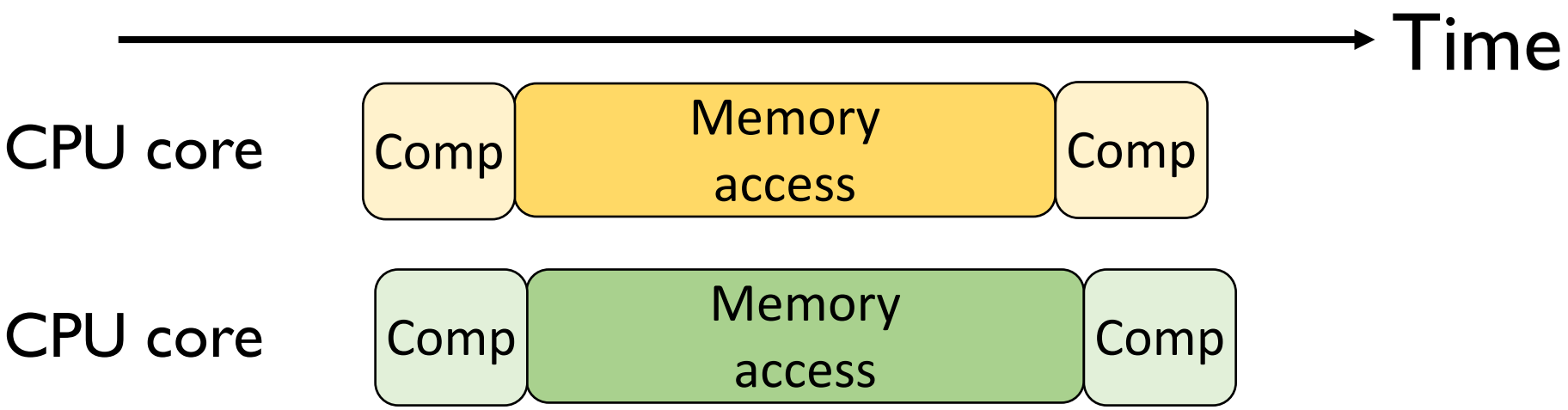


- Opportunity: a pointer-chasing accelerator spends a long time waiting for memory

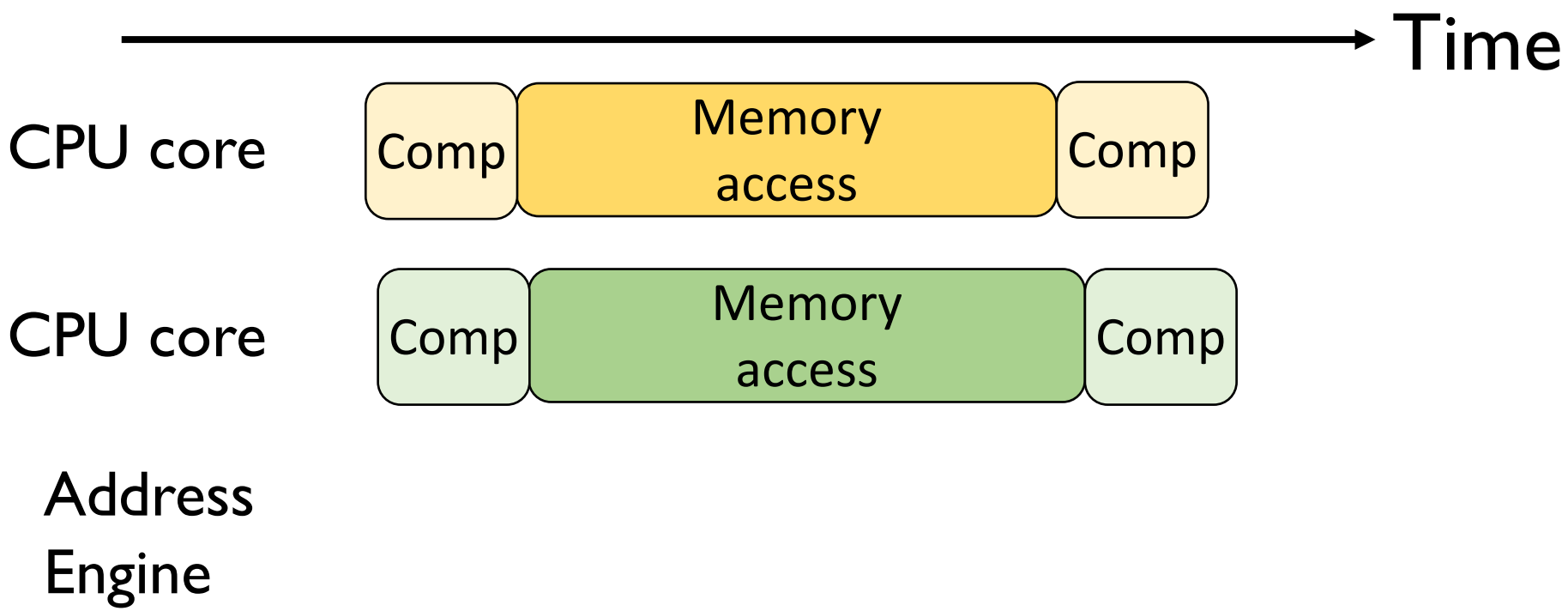


Our Solution: Address-Access Decoupling

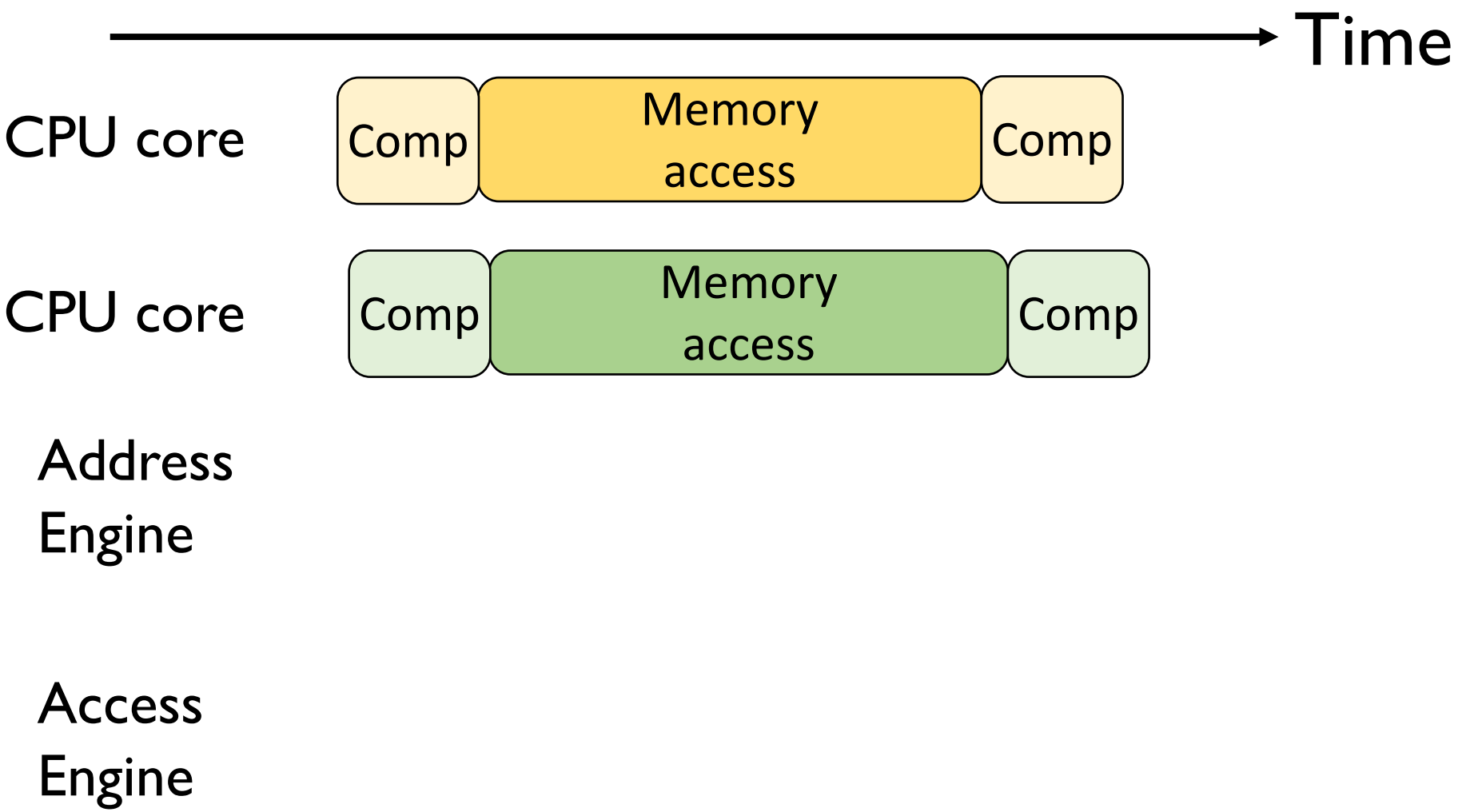
Our Solution: Address-Access Decoupling



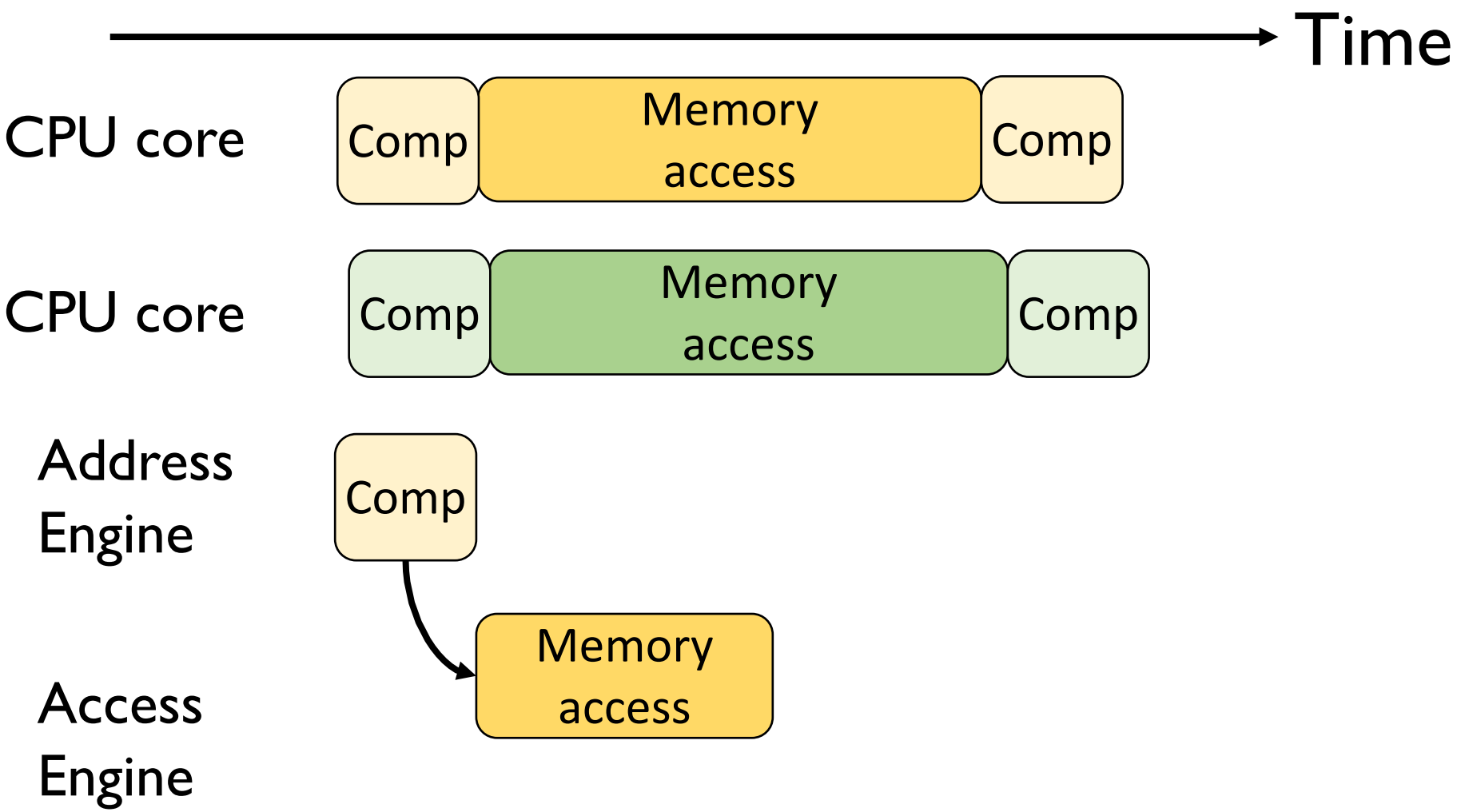
Our Solution: Address-Access Decoupling



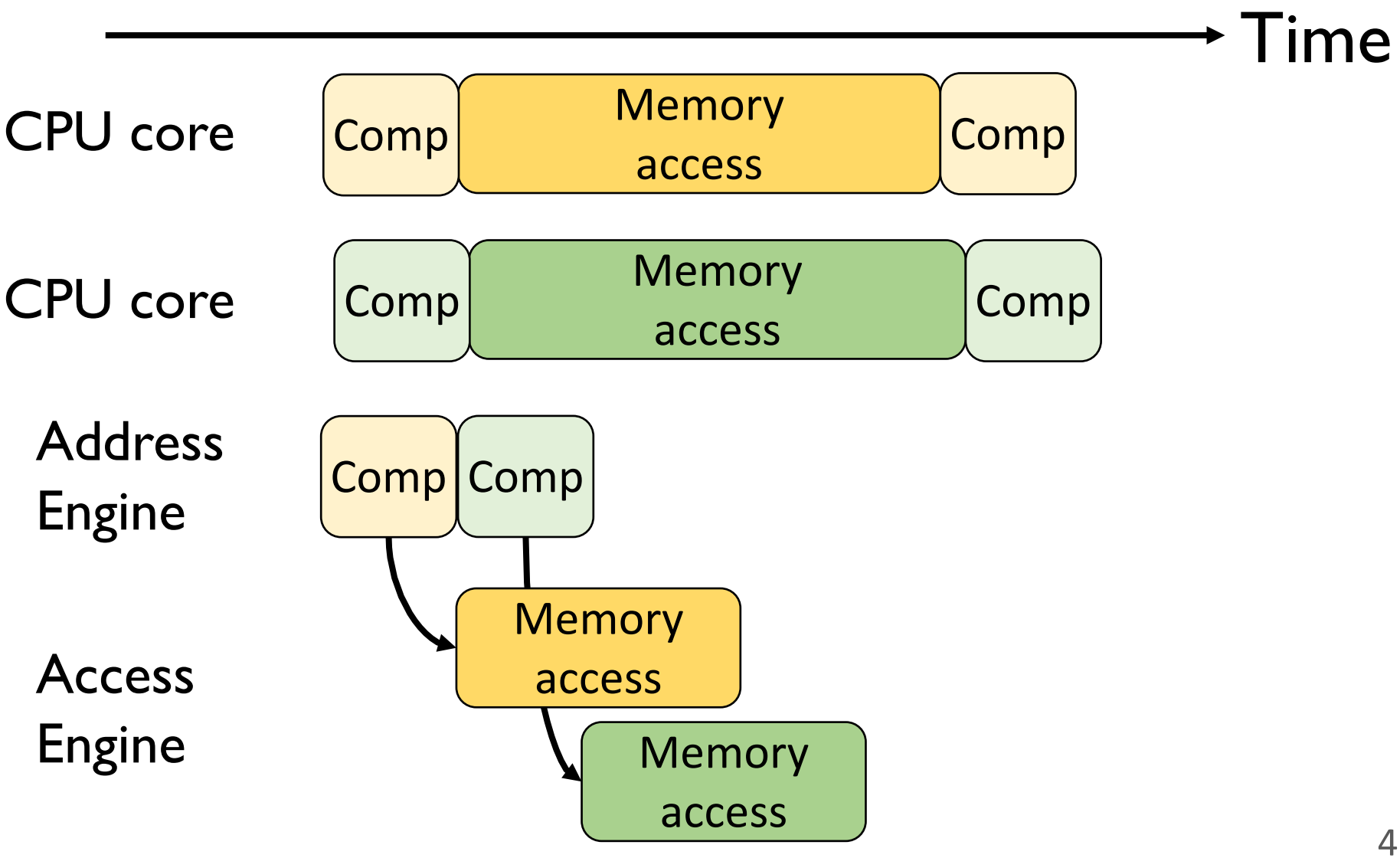
Our Solution: Address-Access Decoupling



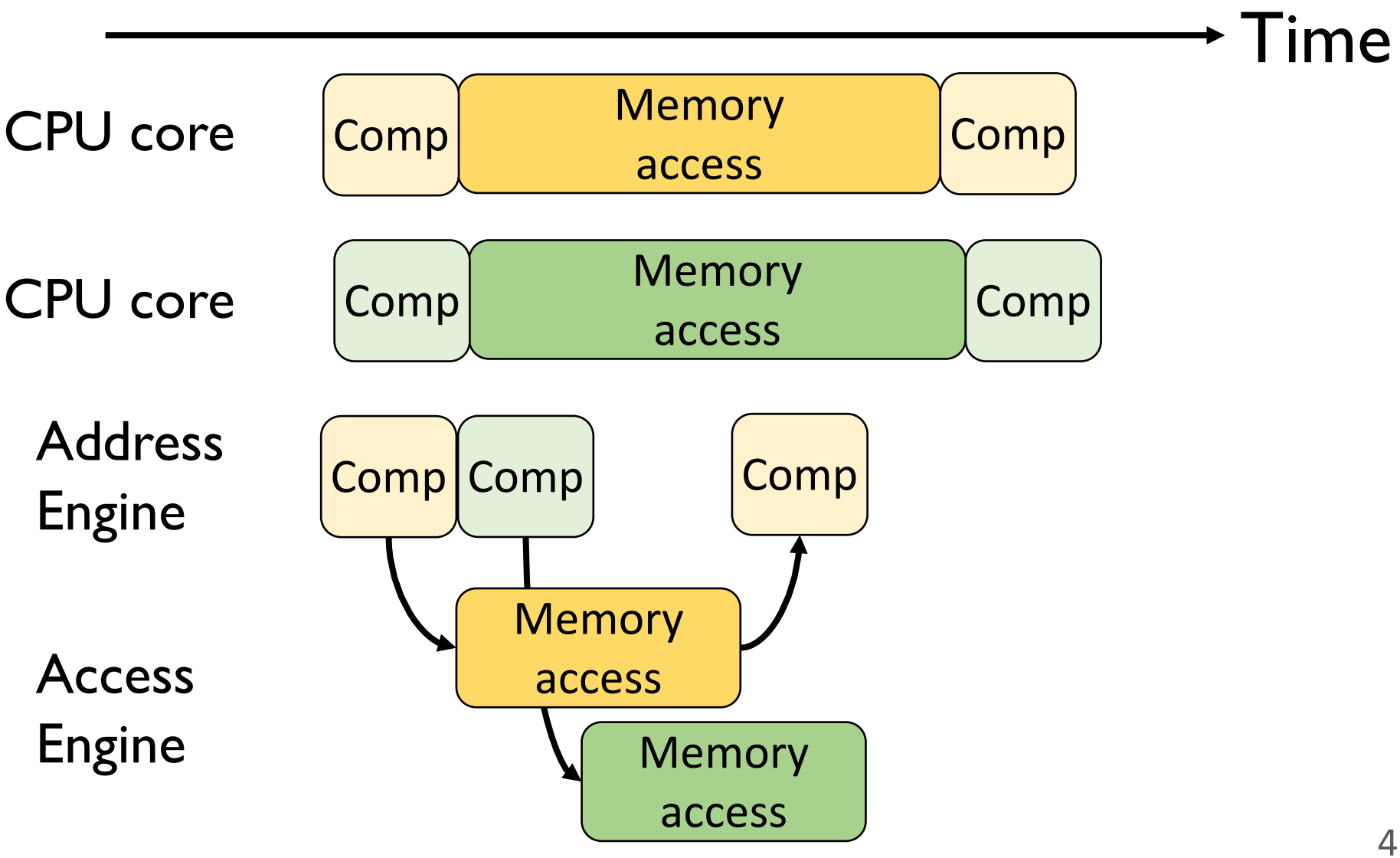
Our Solution: Address-Access Decoupling



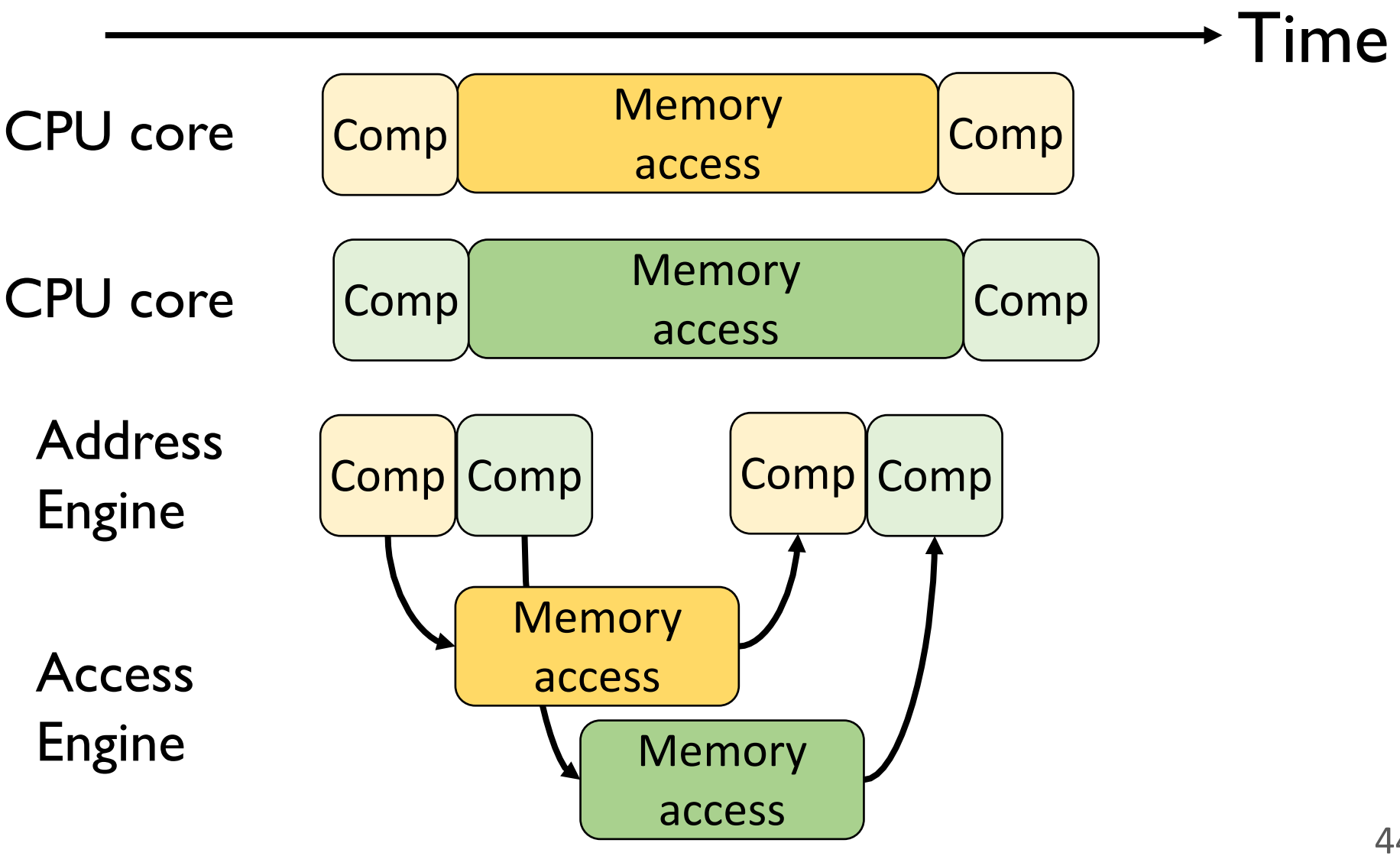
Our Solution: Address-Access Decoupling



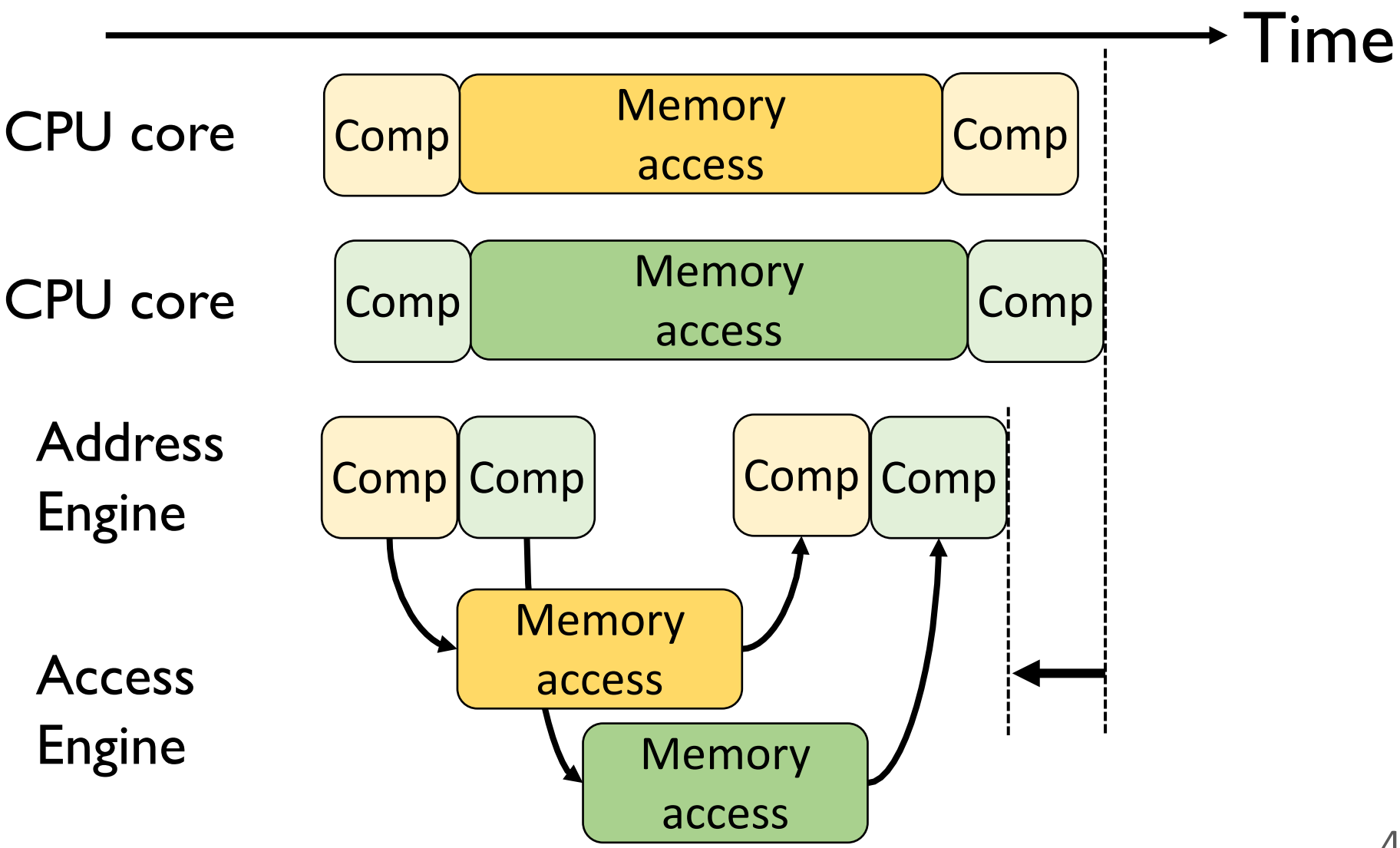
Our Solution: Address-Access Decoupling



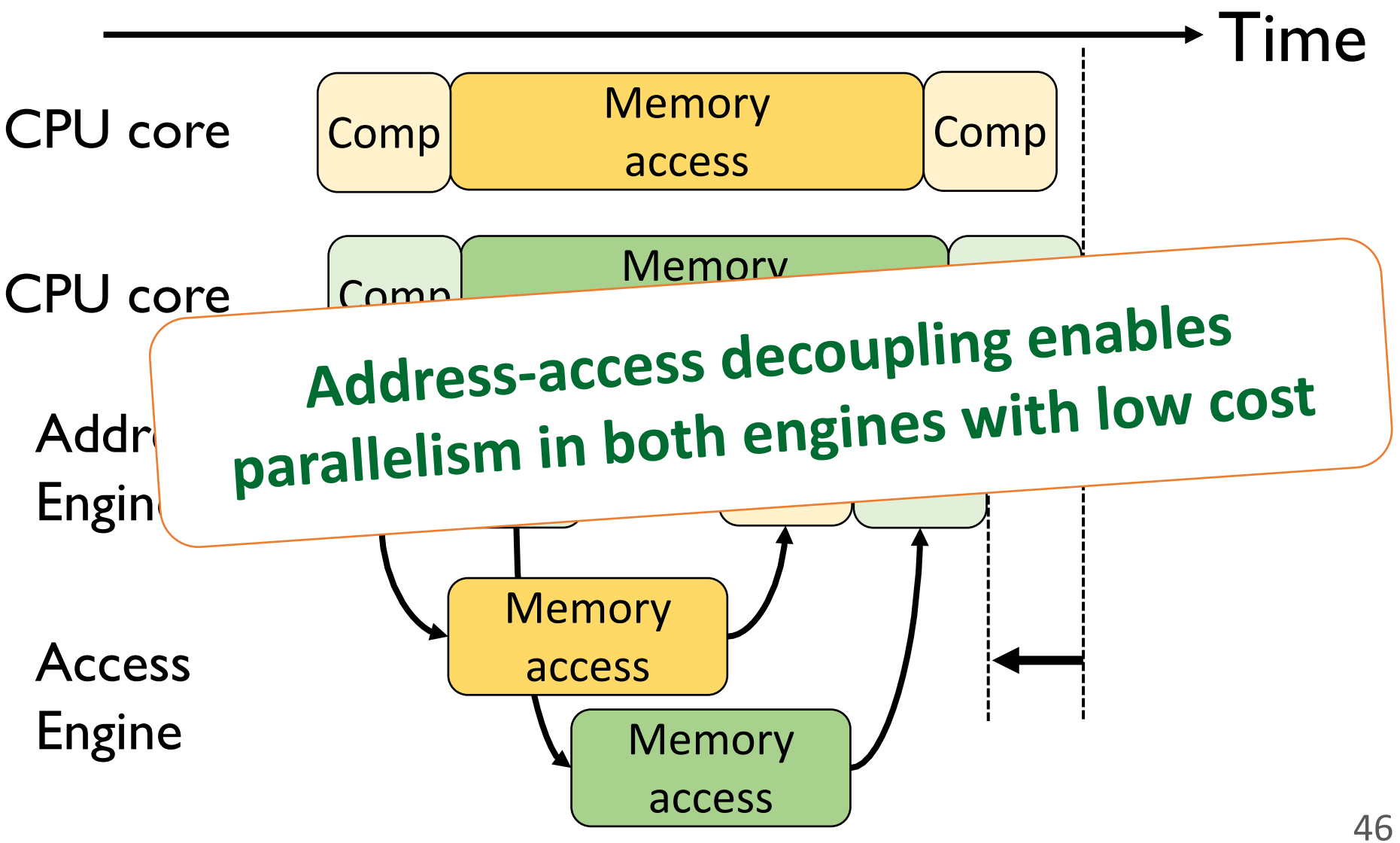
Our Solution: Address-Access Decoupling



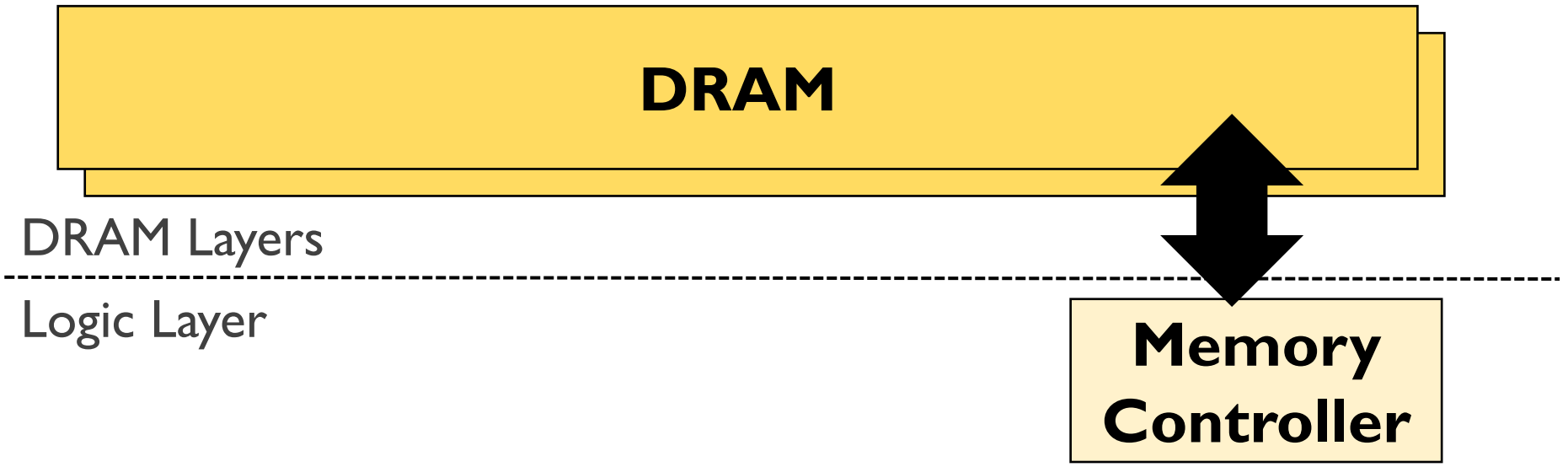
Our Solution: Address-Access Decoupling



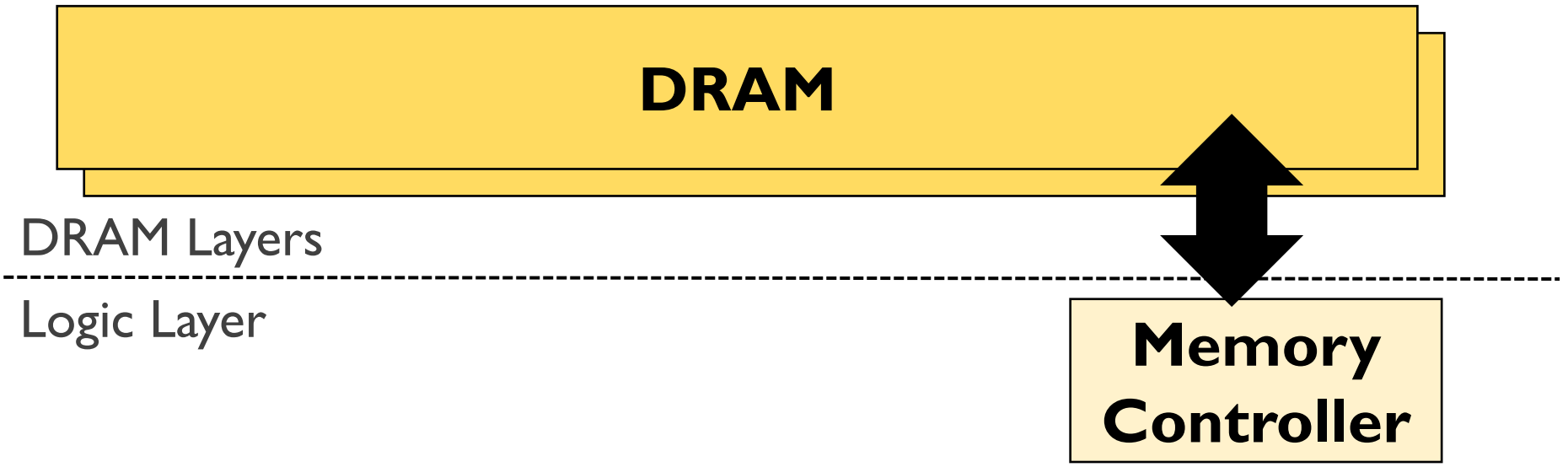
Our Solution: Address-Access Decoupling



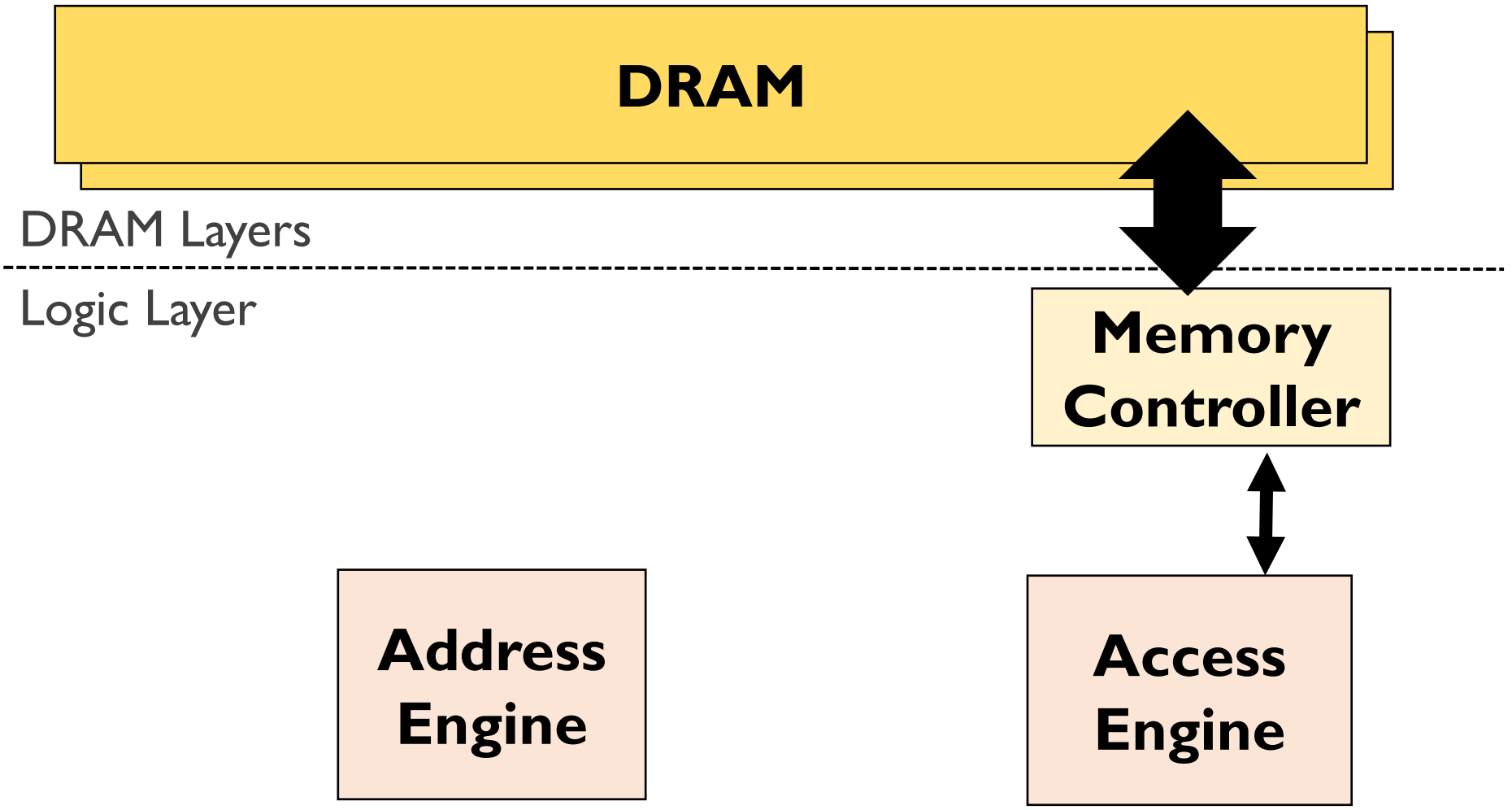
IMPICA Core Architecture



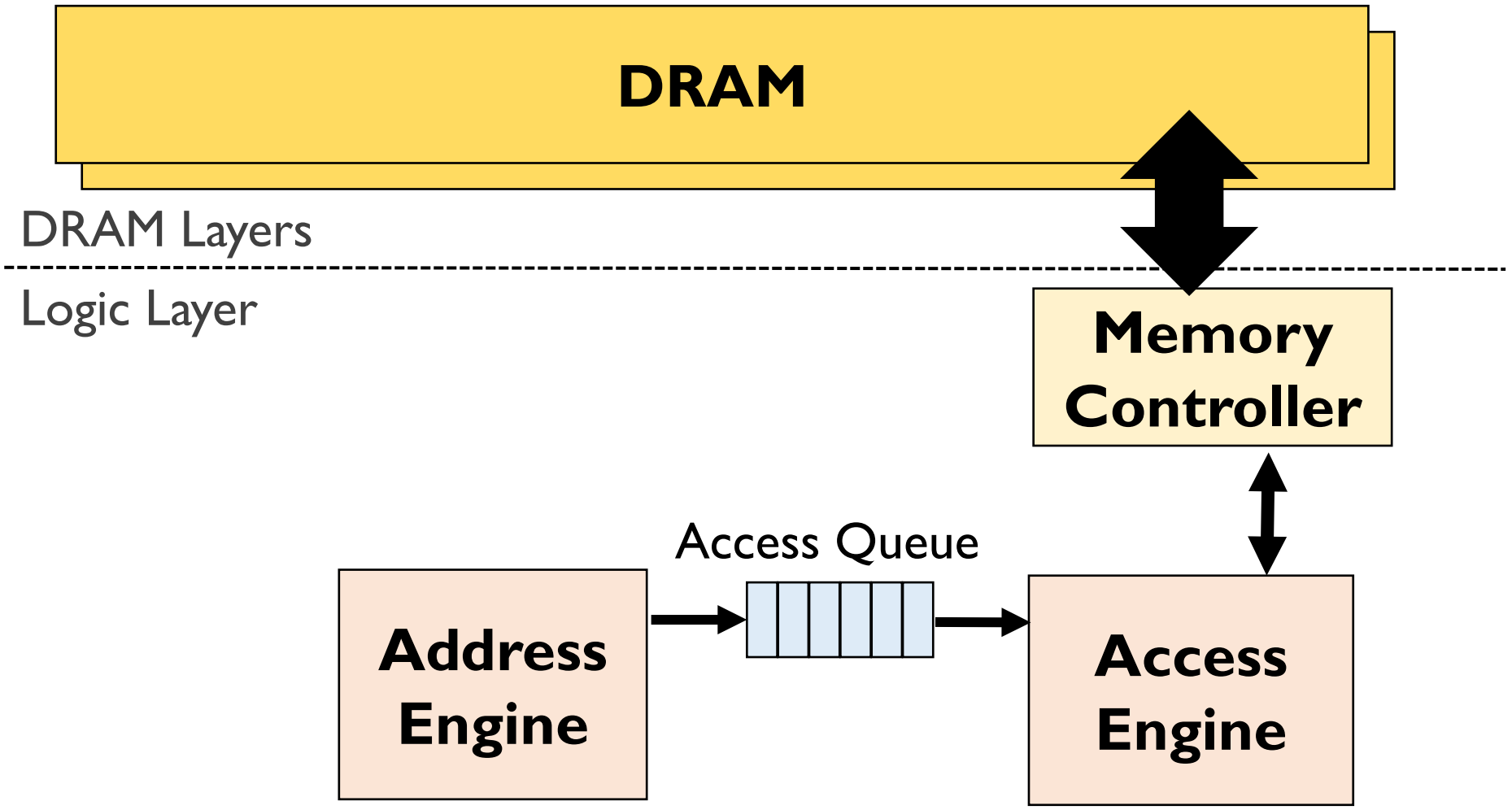
IMPICA Core Architecture



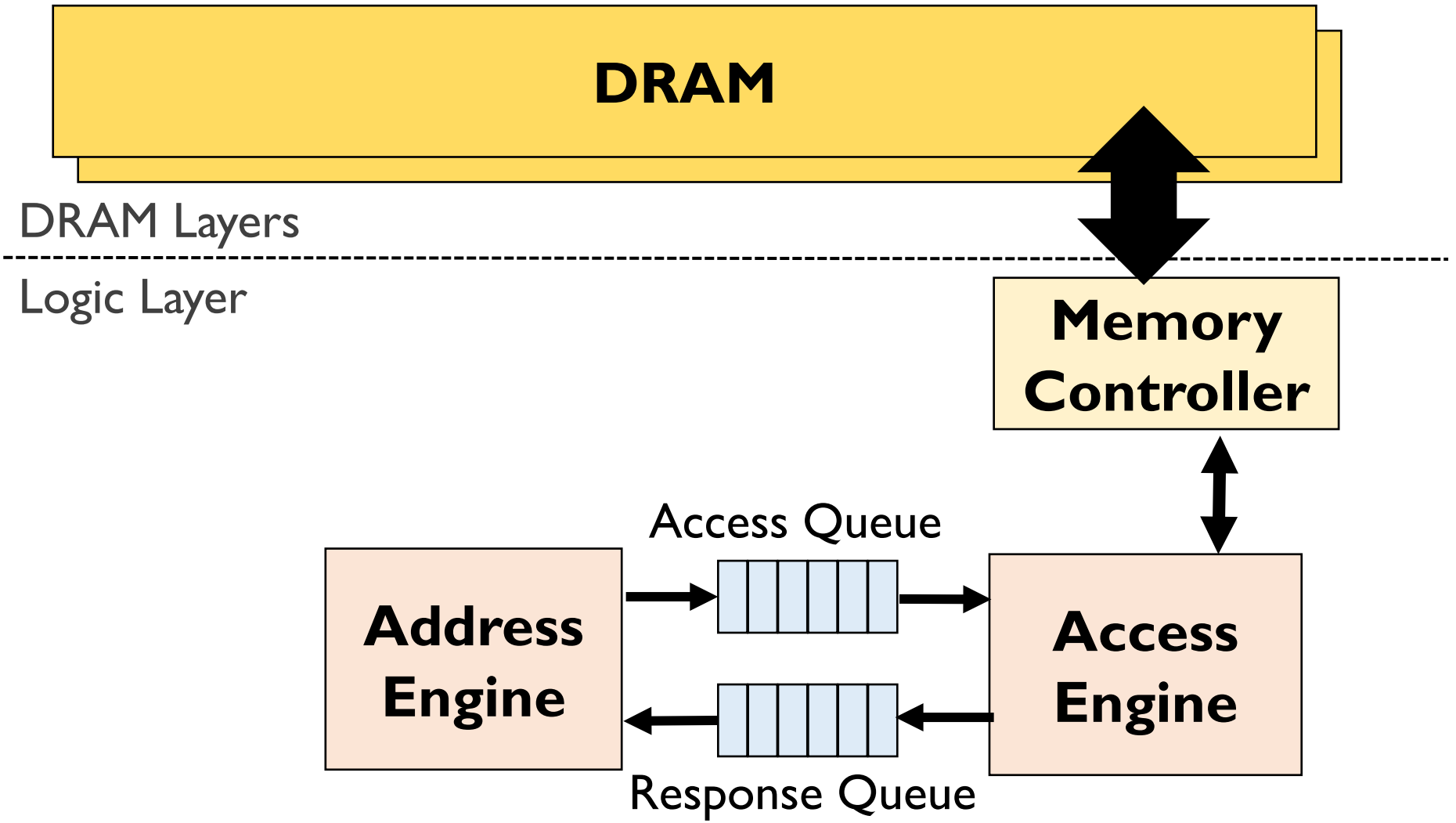
IMPICA Core Architecture



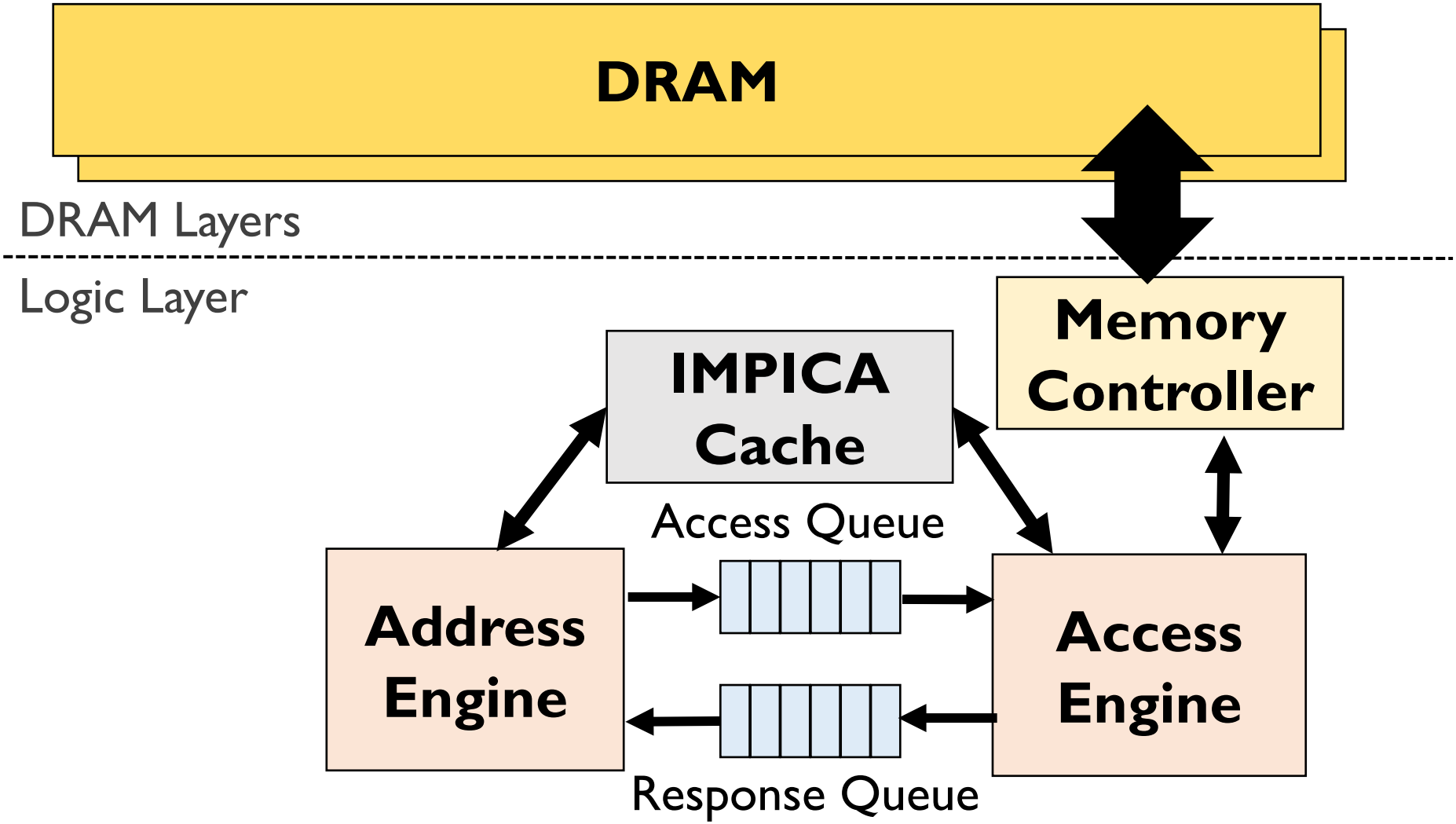
IMPICA Core Architecture



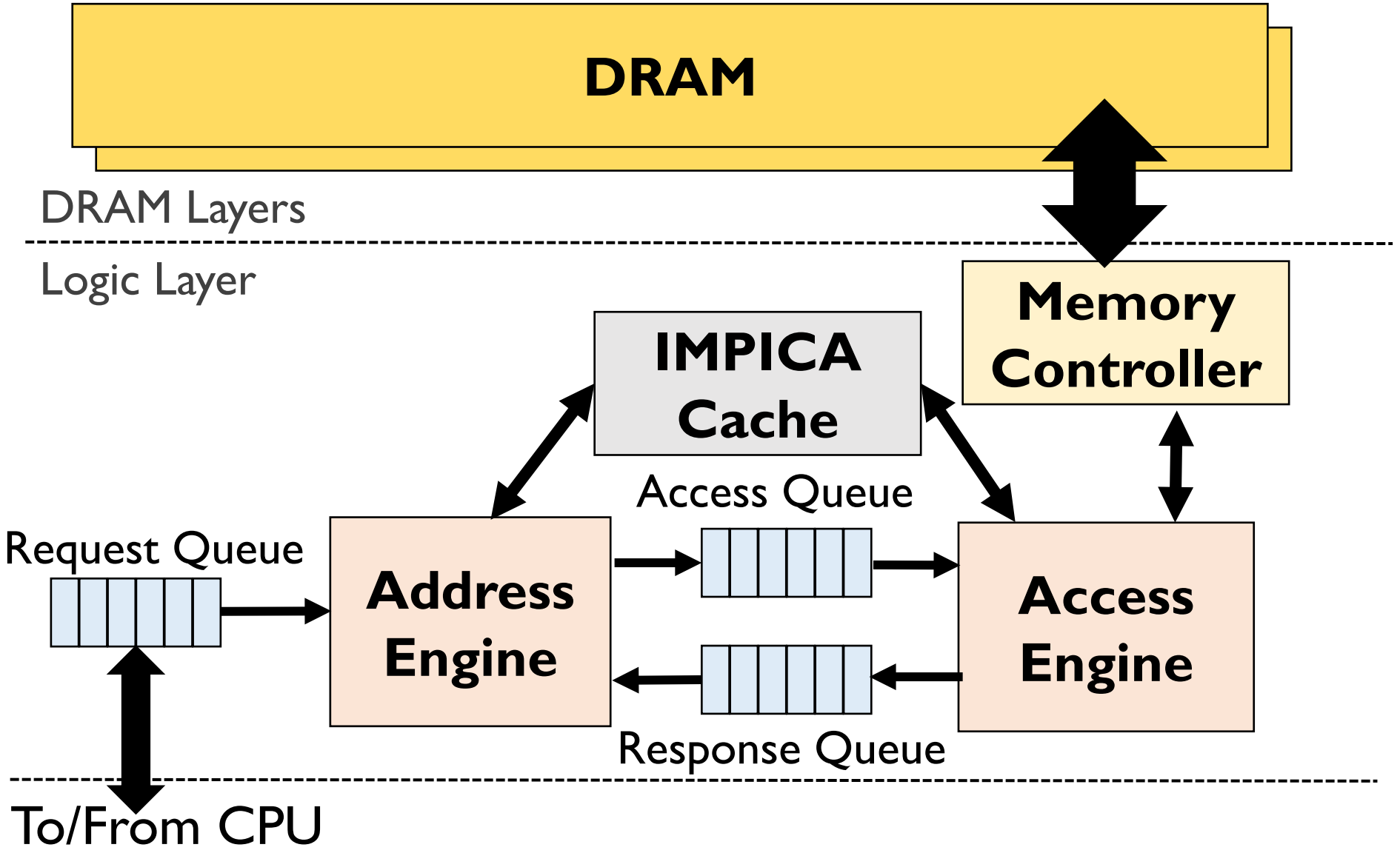
IMPICA Core Architecture



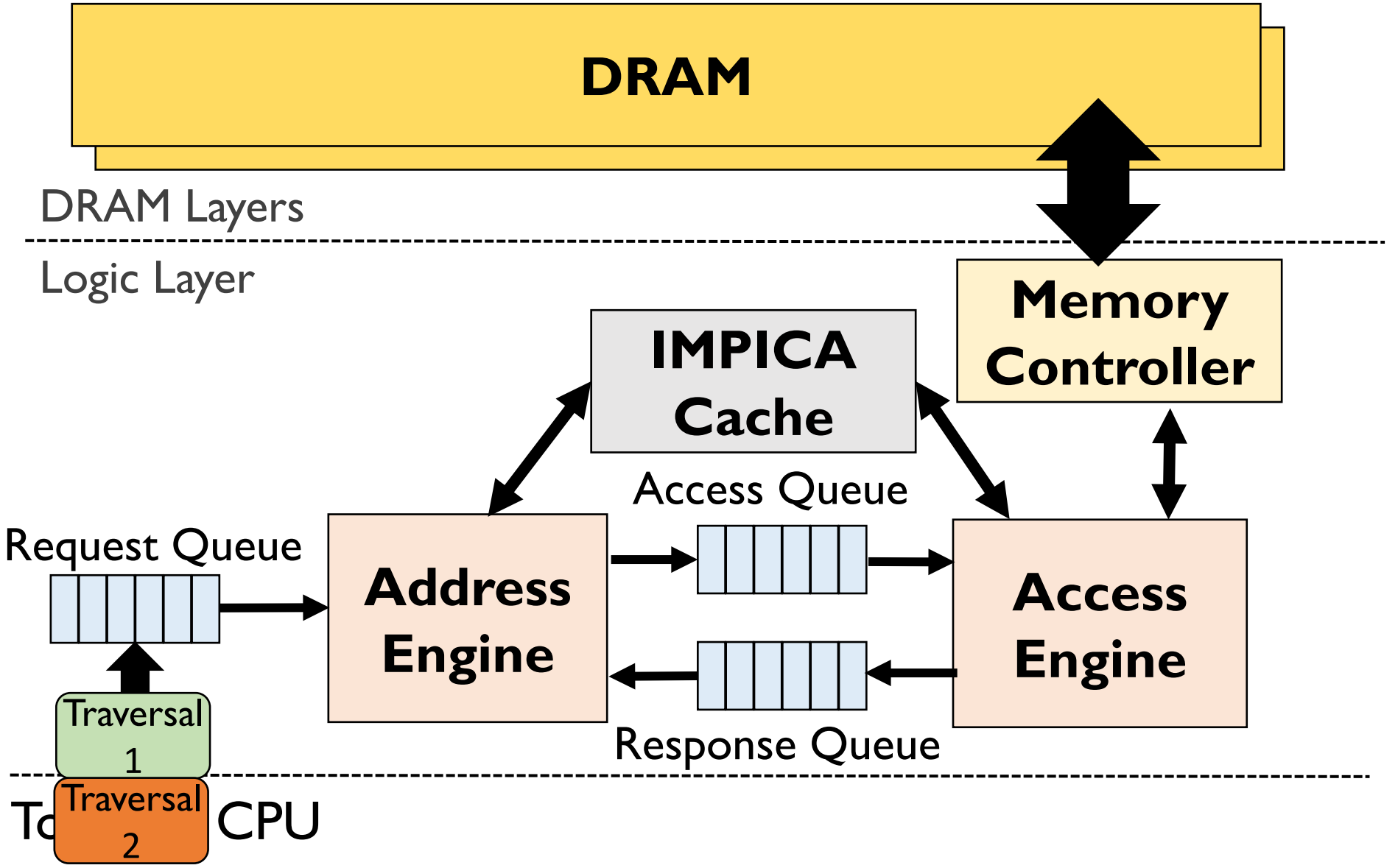
IMPICA Core Architecture



IMPICA Core Architecture



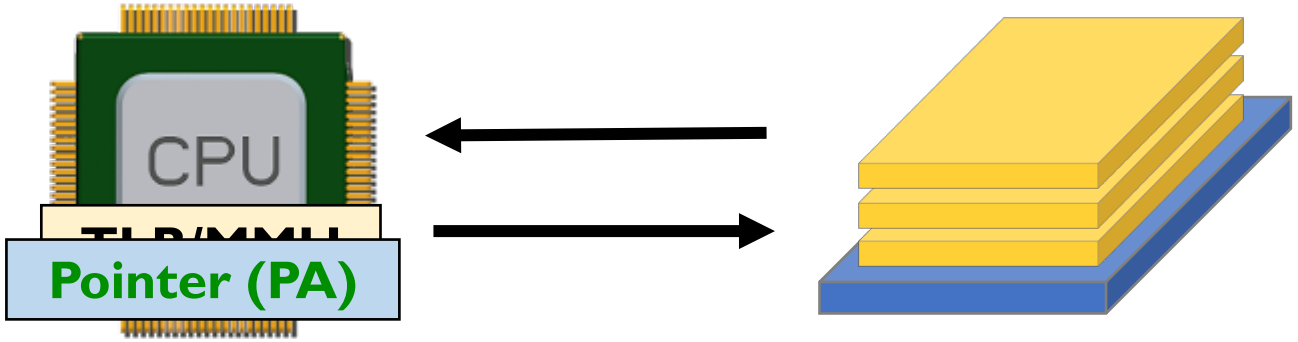
IMPICA Core Architecture

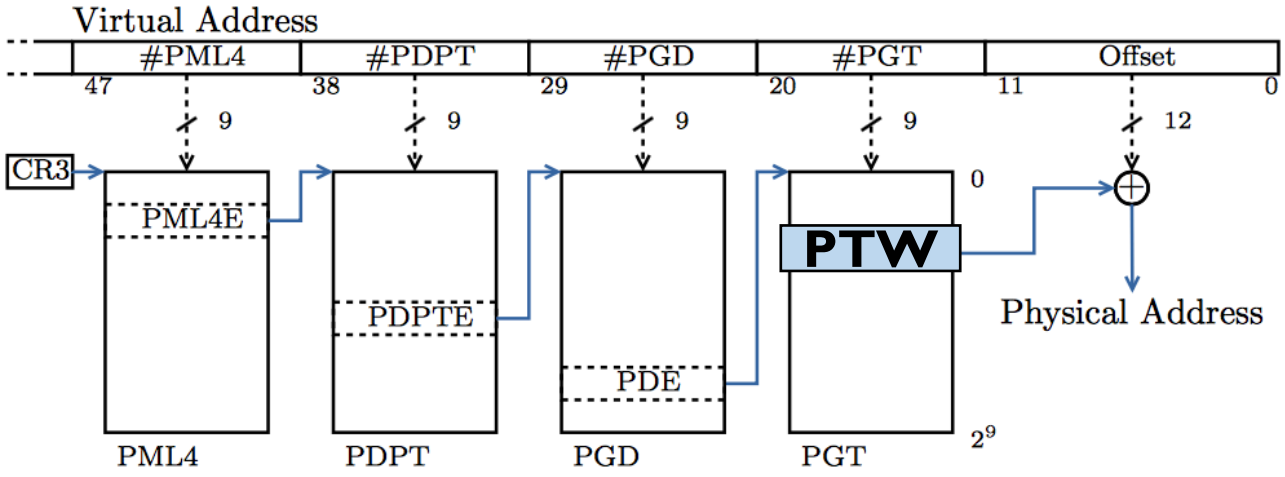


Outline

- Motivation and Our Approach
- Parallelism Challenge
- IMPICA Core Architecture
- Address Translation Challenge
- IMPICA Page Table
- Evaluation
- Conclusion

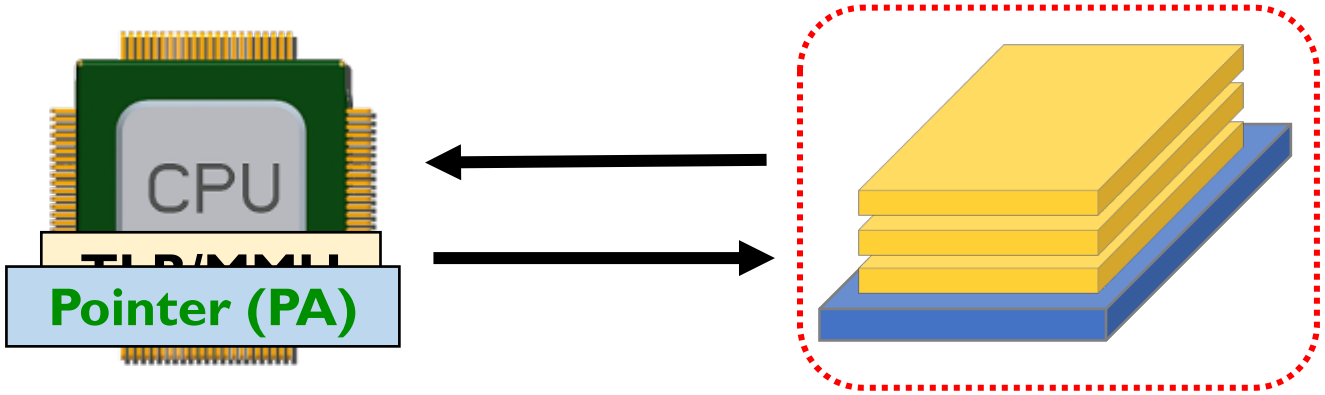
Address Translation Challenge

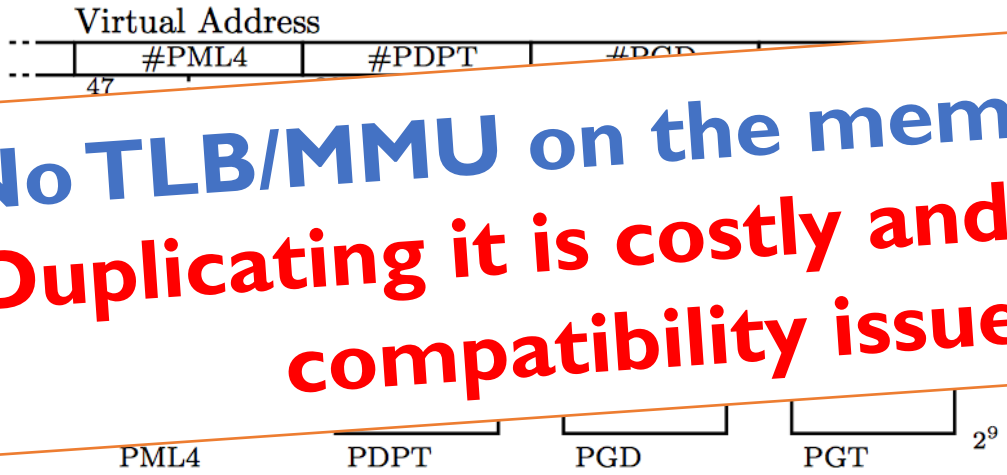




Page table walk

Address Translation Challenge





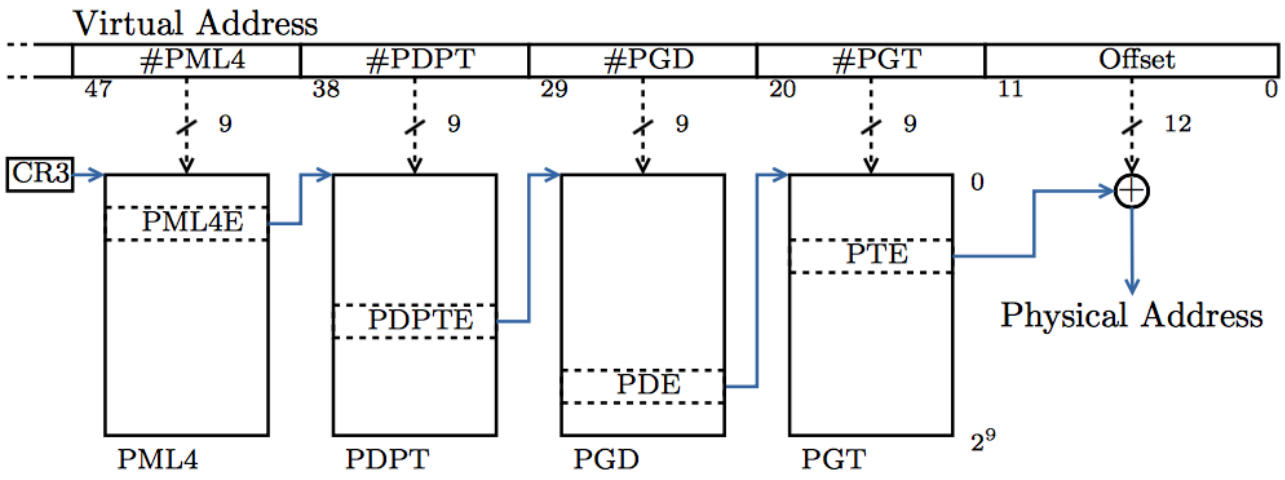
No TLB/MMU on the memory side

Duplicating it is costly and creates compatibility issue

Page table walk

Address Translation Challenge

The page table walk requires multiple memory accesses



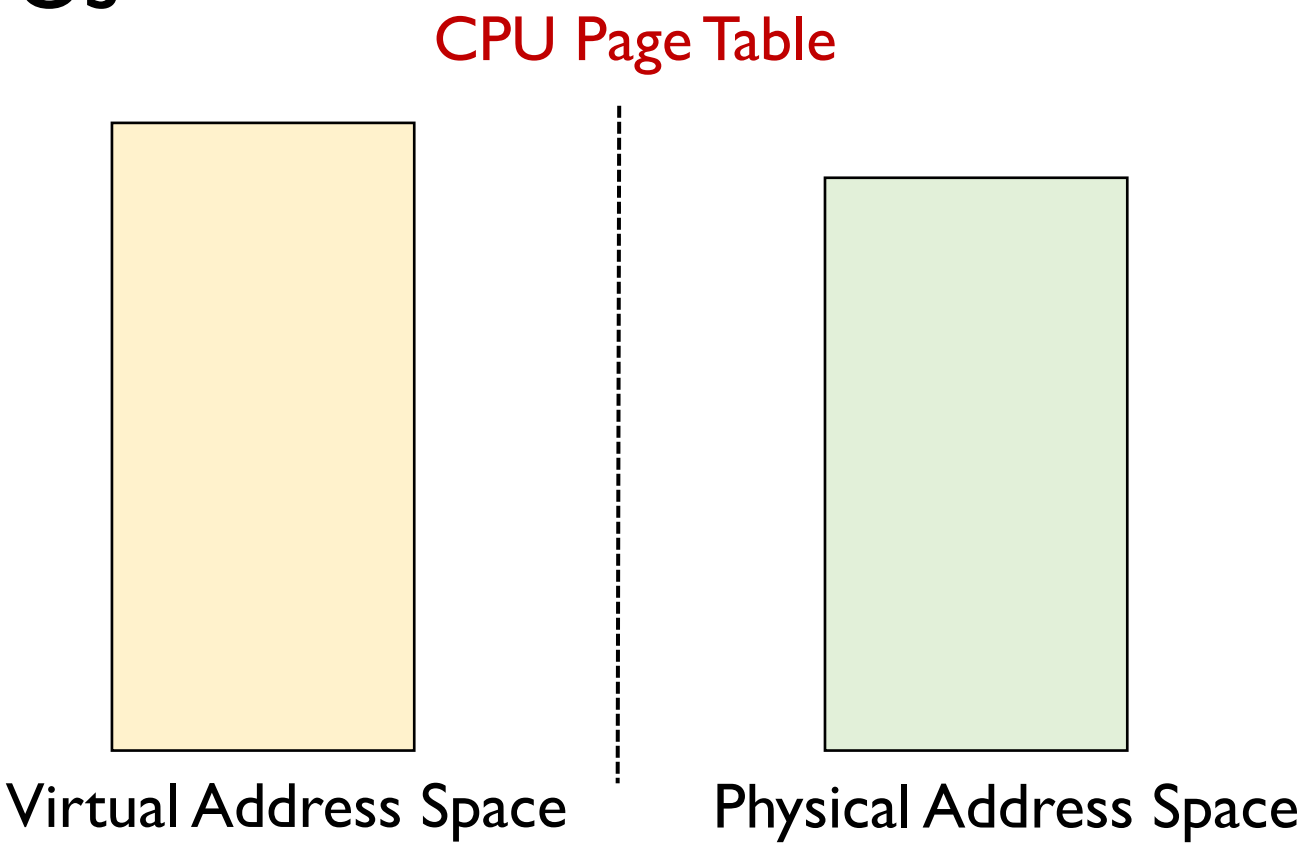
Page table walk

Our Solution: IMPICA Page Table

- Completely decouple the page table of IMPICA from the page table of the CPUs

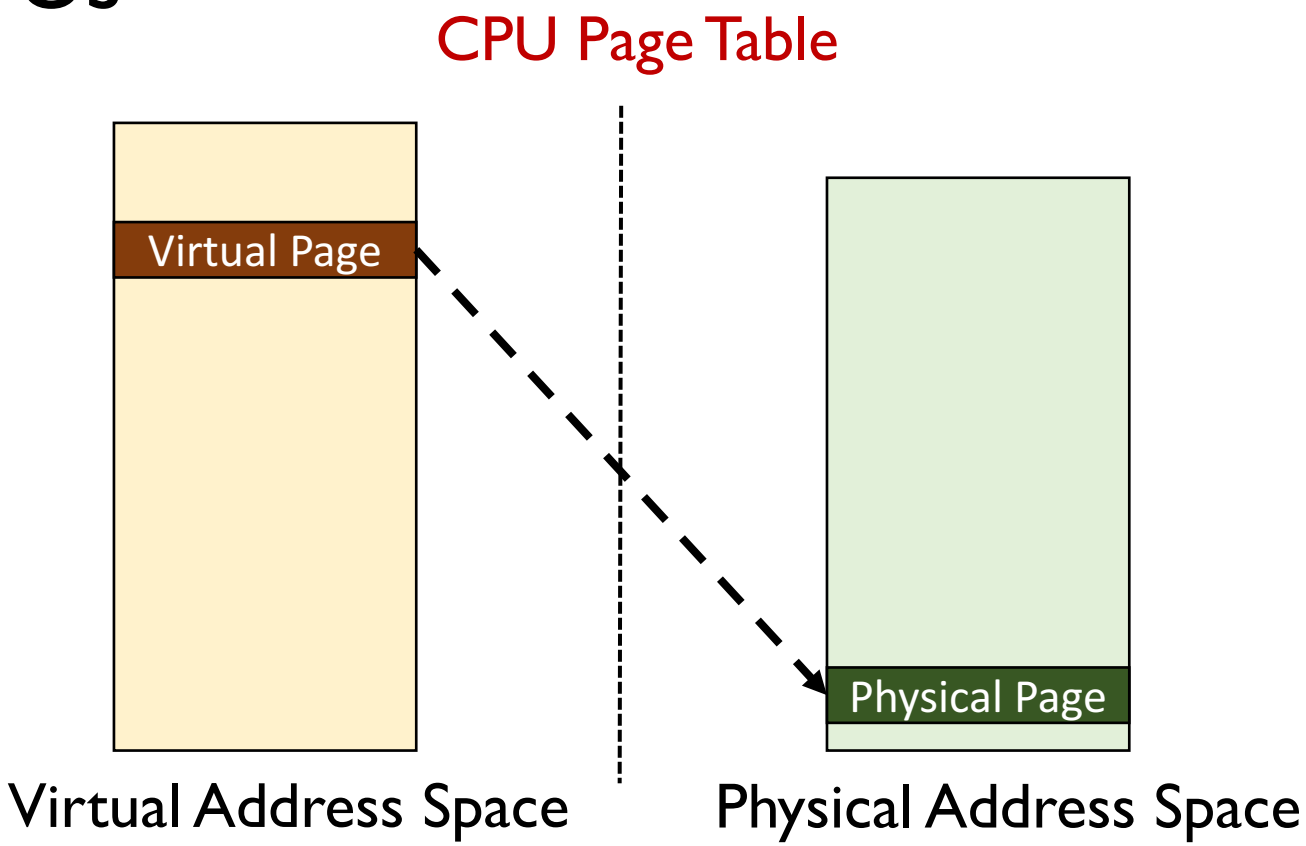
Our Solution: IMPICA Page Table

- Completely decouple the page table of IMPICA from the page table of the CPUs



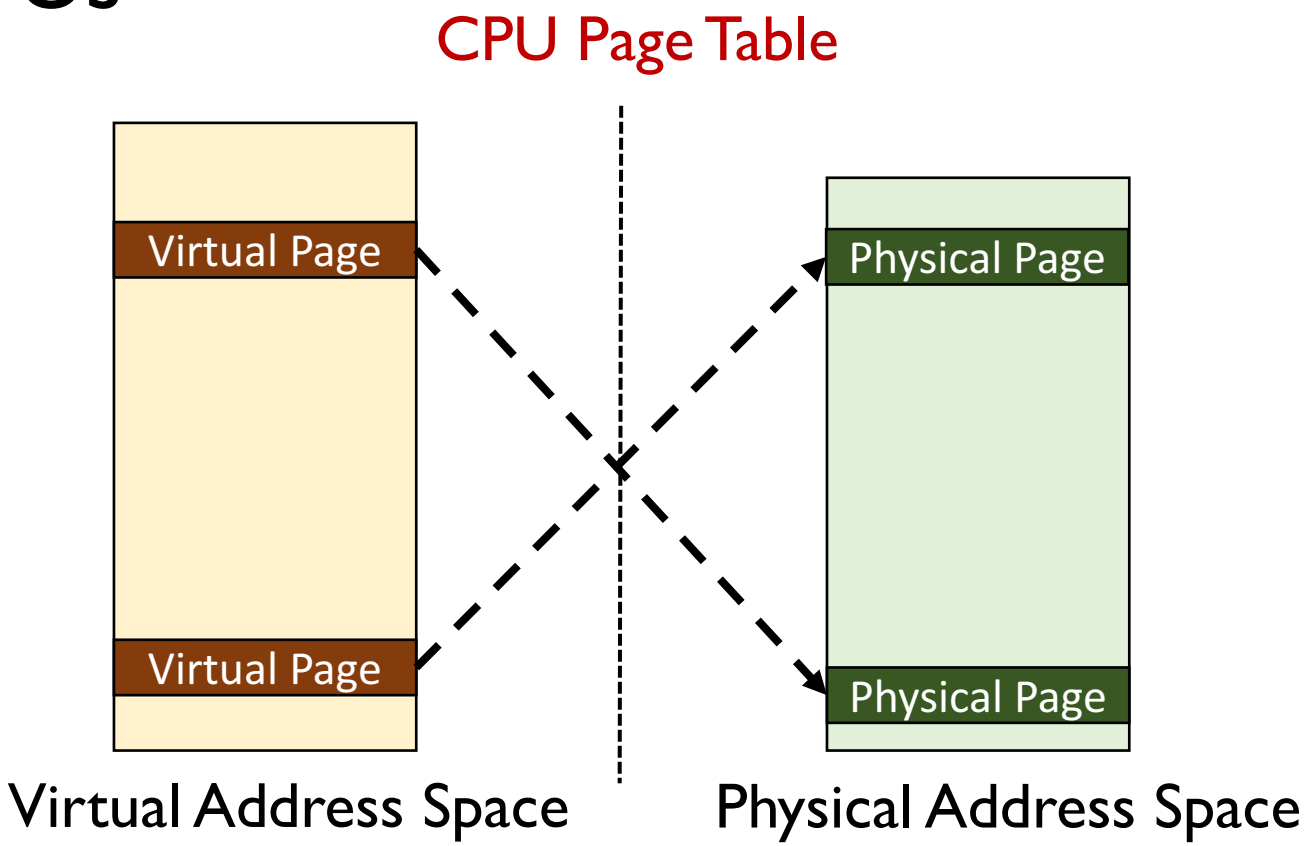
Our Solution: IMPICA Page Table

- Completely decouple the page table of IMPICA from the page table of the CPUs



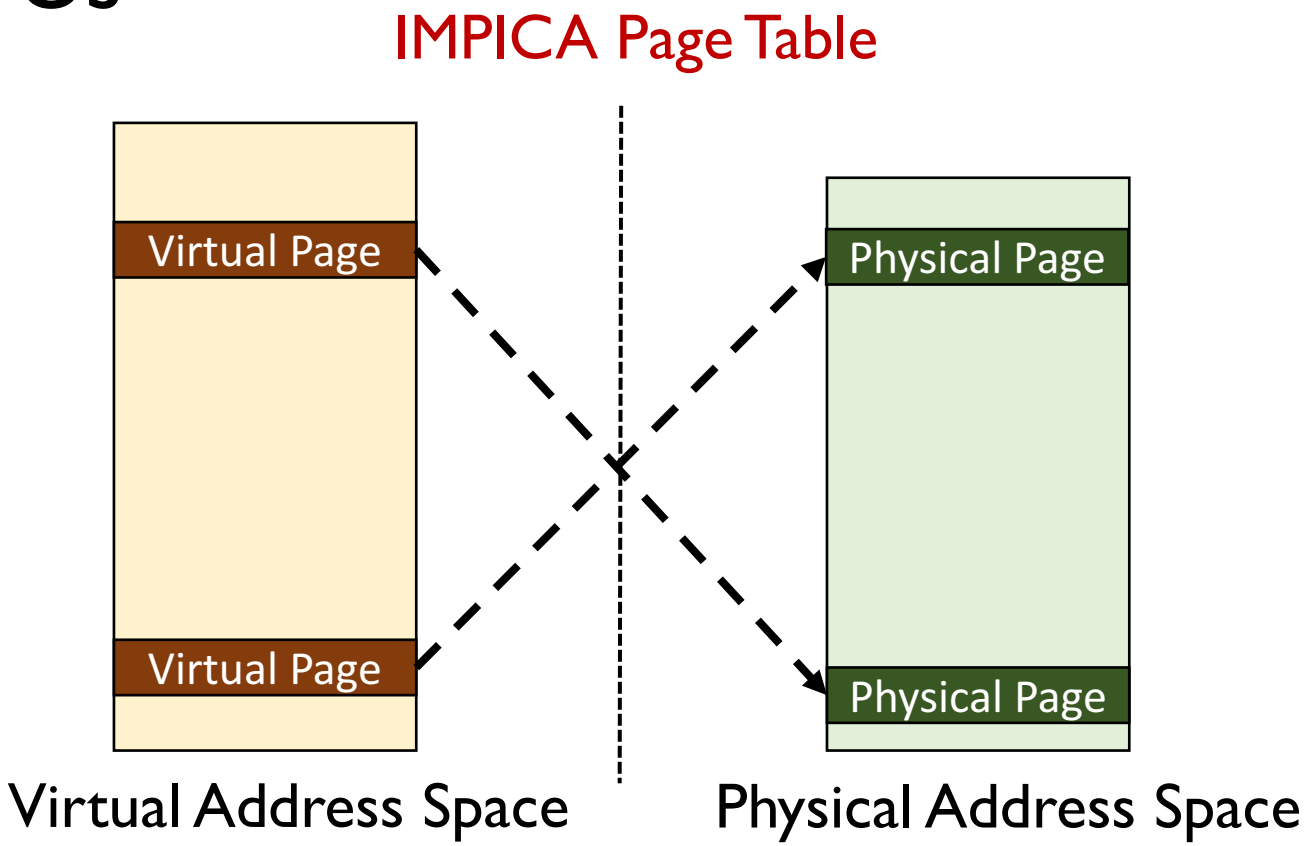
Our Solution: IMPICA Page Table

- Completely decouple the page table of IMPICA from the page table of the CPUs



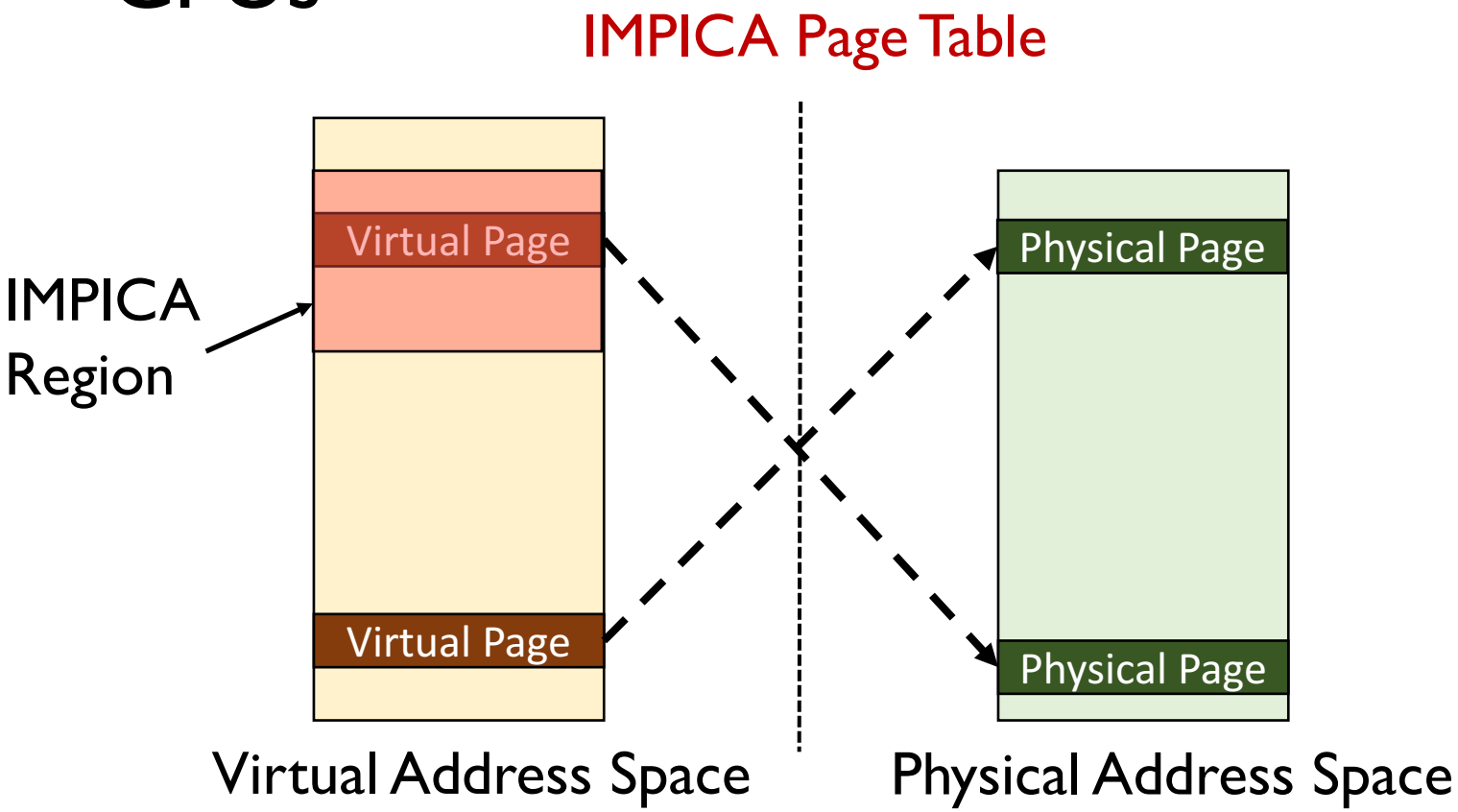
Our Solution: IMPICA Page Table

- Completely decouple the page table of IMPICA from the page table of the CPUs



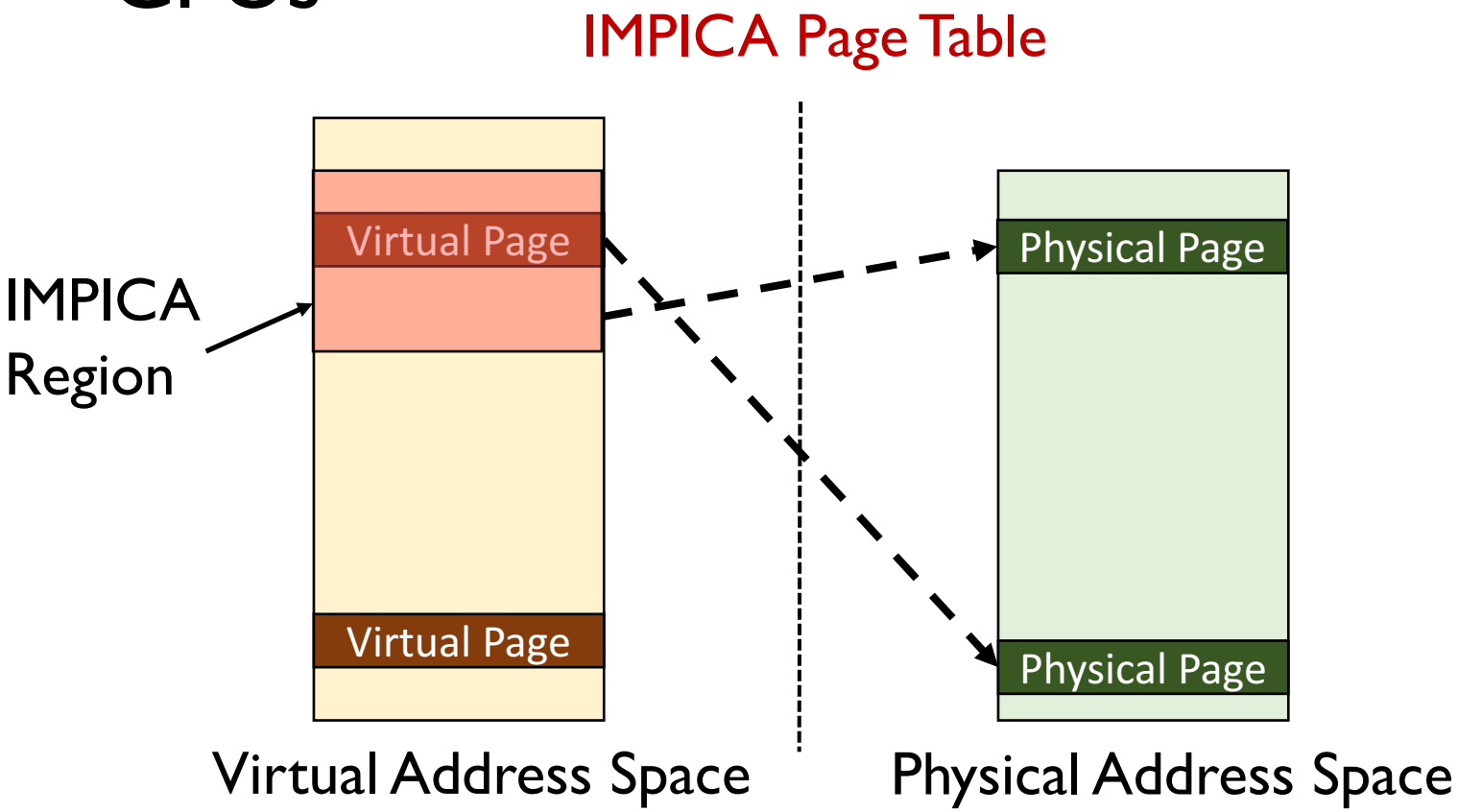
Our Solution: IMPICA Page Table

- Completely decouple the page table of IMPICA from the page table of the CPUs



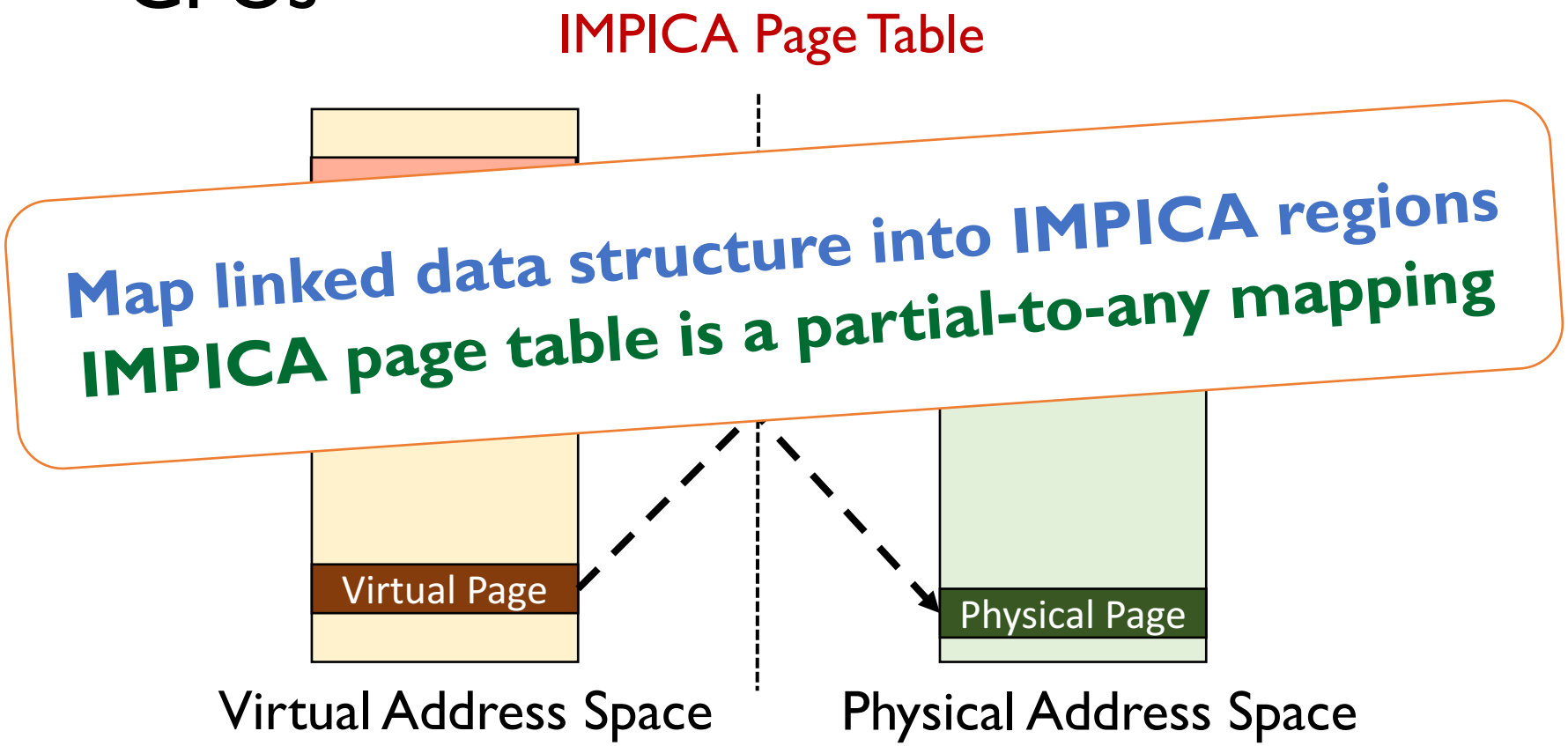
Our Solution: IMPICA Page Table

- Completely decouple the page table of IMPICA from the page table of the CPUs

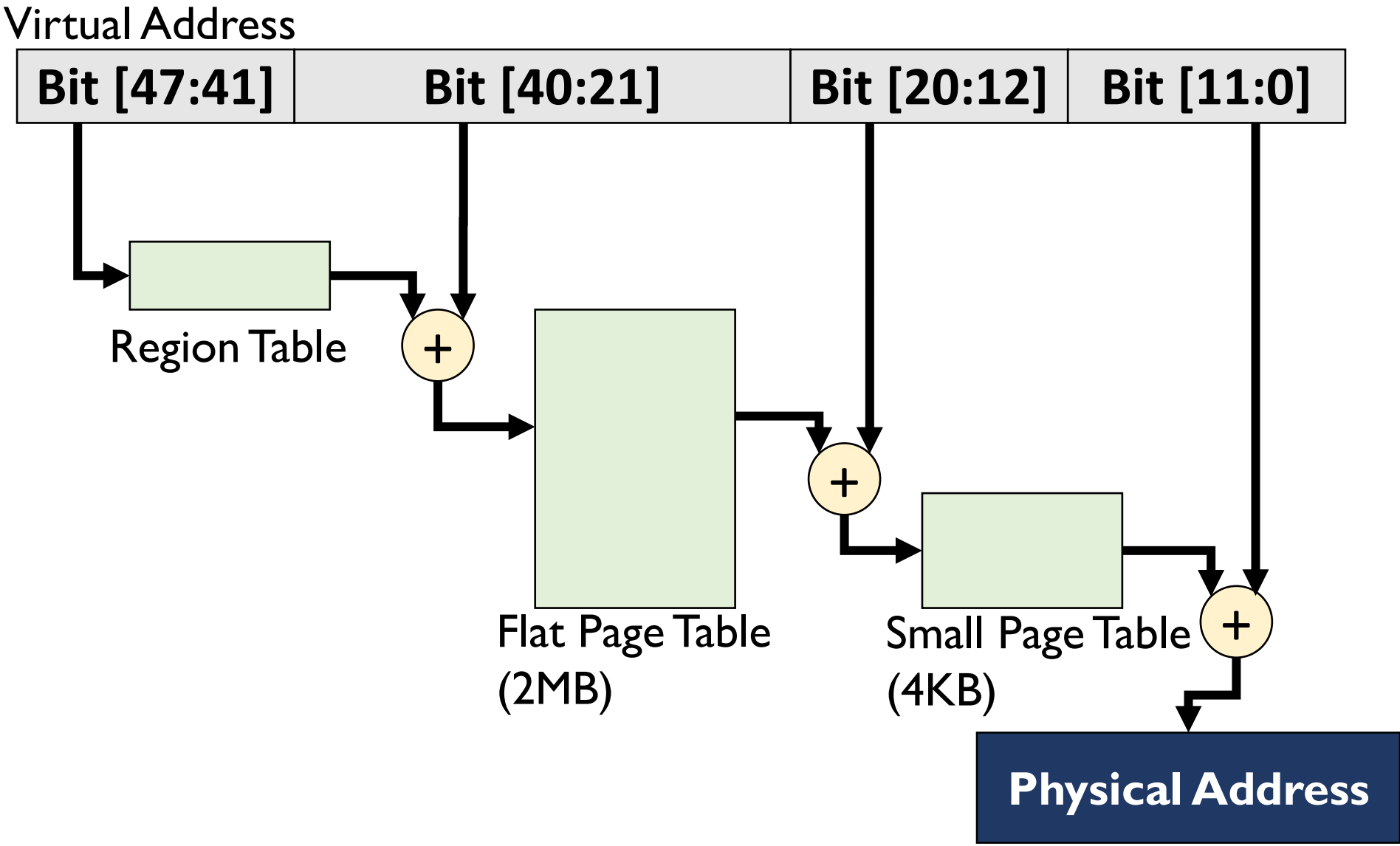


Our Solution: IMPICA Page Table

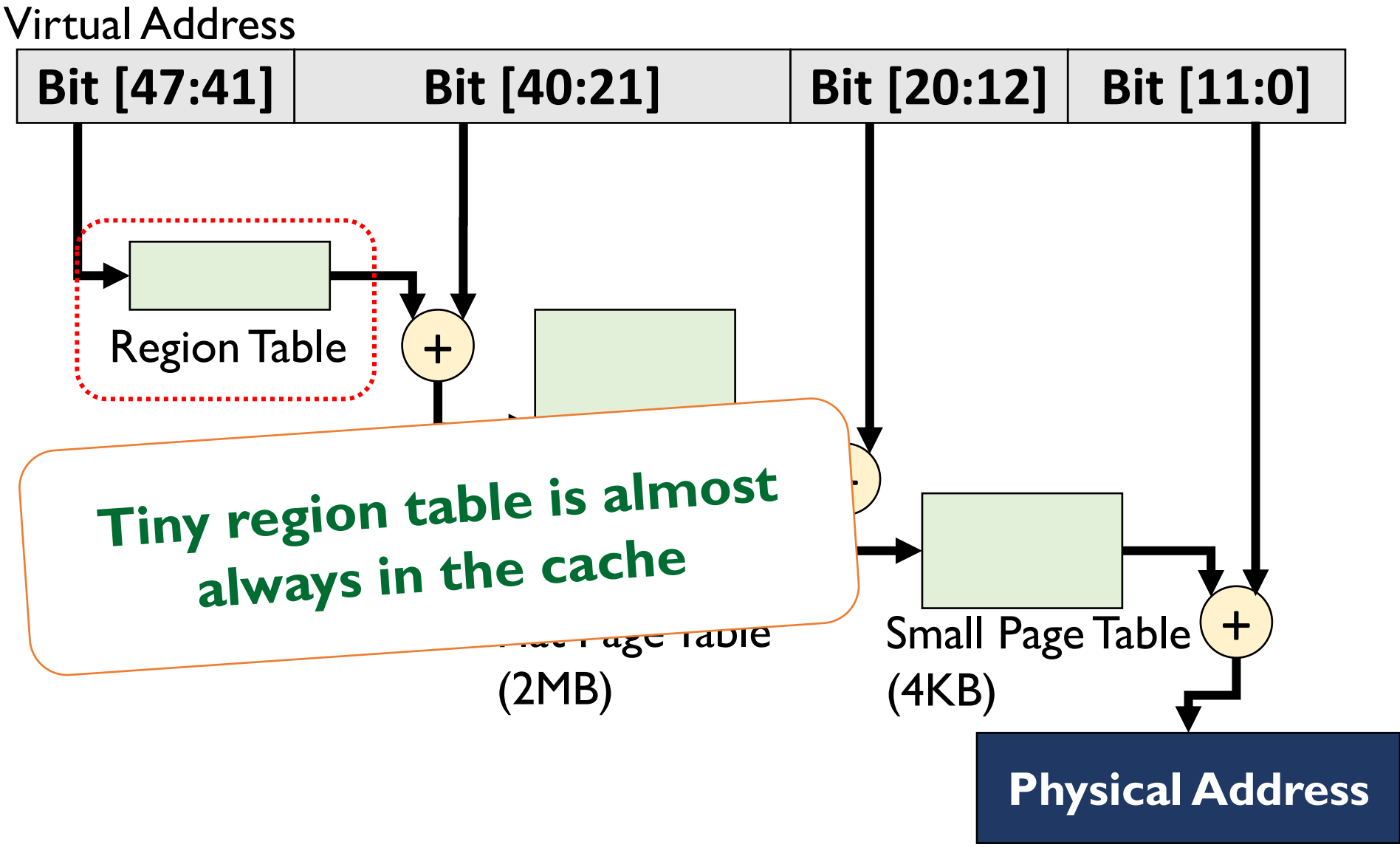
- Completely decouple the page table of IMPICA from the page table of the CPUs



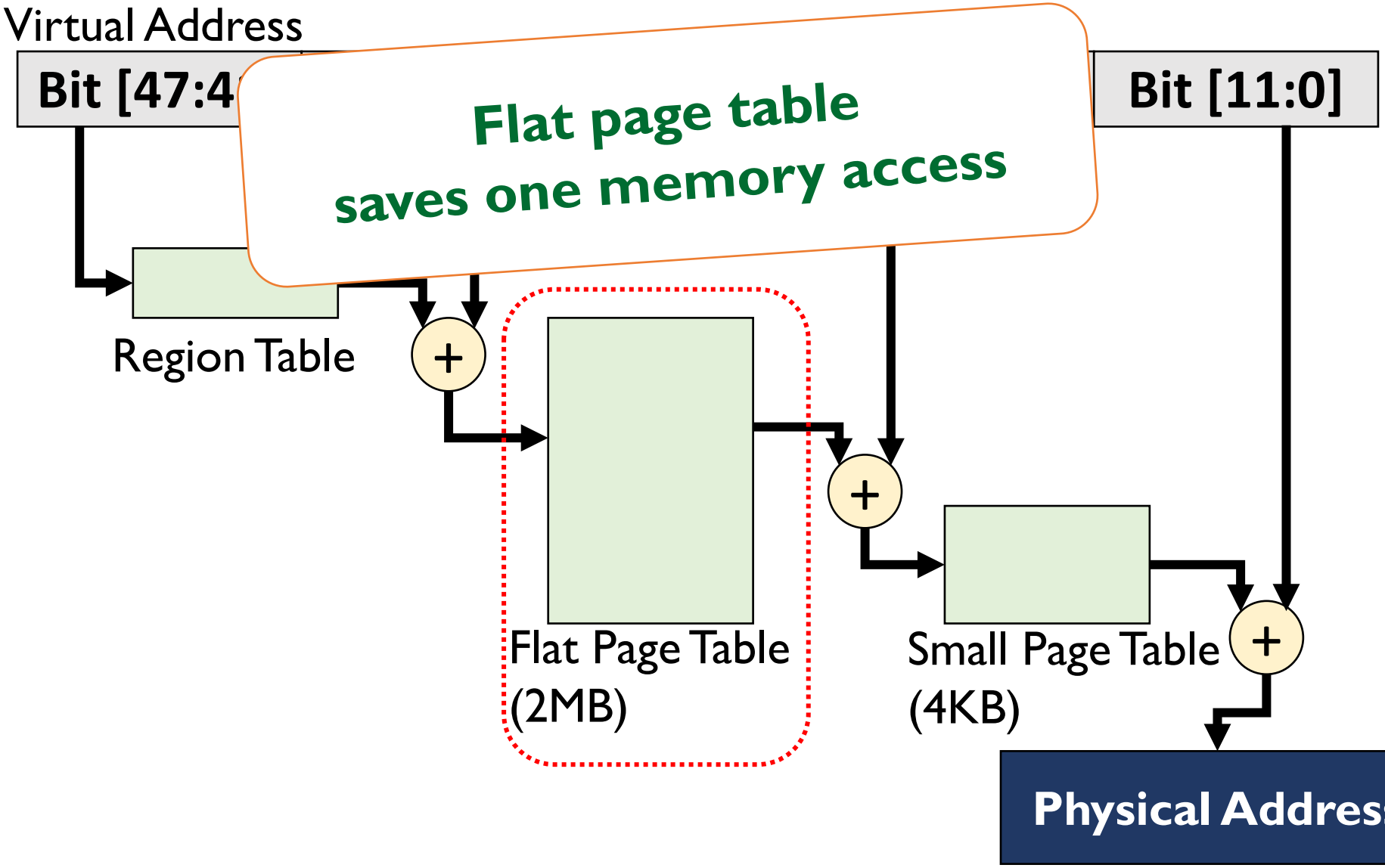
IMPICA Page Table: Mechanism



IMPICA Page Table: Mechanism



IMPICA Page Table: Mechanism



Outline

- Motivation and Our Approach
- Parallelism Challenge
- IMPICA Core Architecture
- Address Translation Challenge
- IMPICA Page Table
- Evaluation
- Conclusion

Evaluated Workloads

- Microbenchmarks
  - Linked list (from Olden benchmark)
  - Hash table (from Memcached)
  - B-tree (from DBx1000)

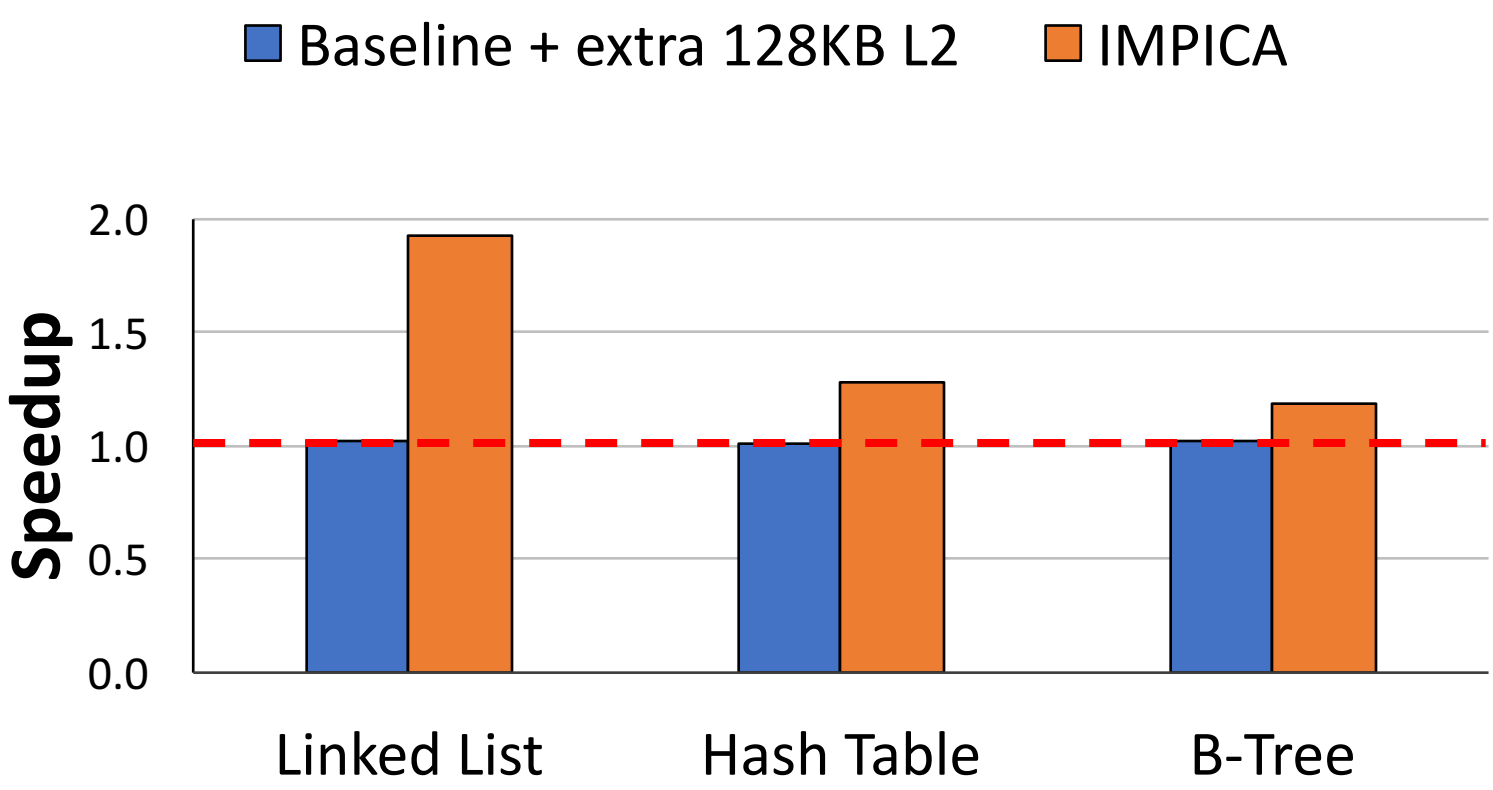
- Application
  - DBx1000 (with TPC-C benchmark)

Evaluation Methodology

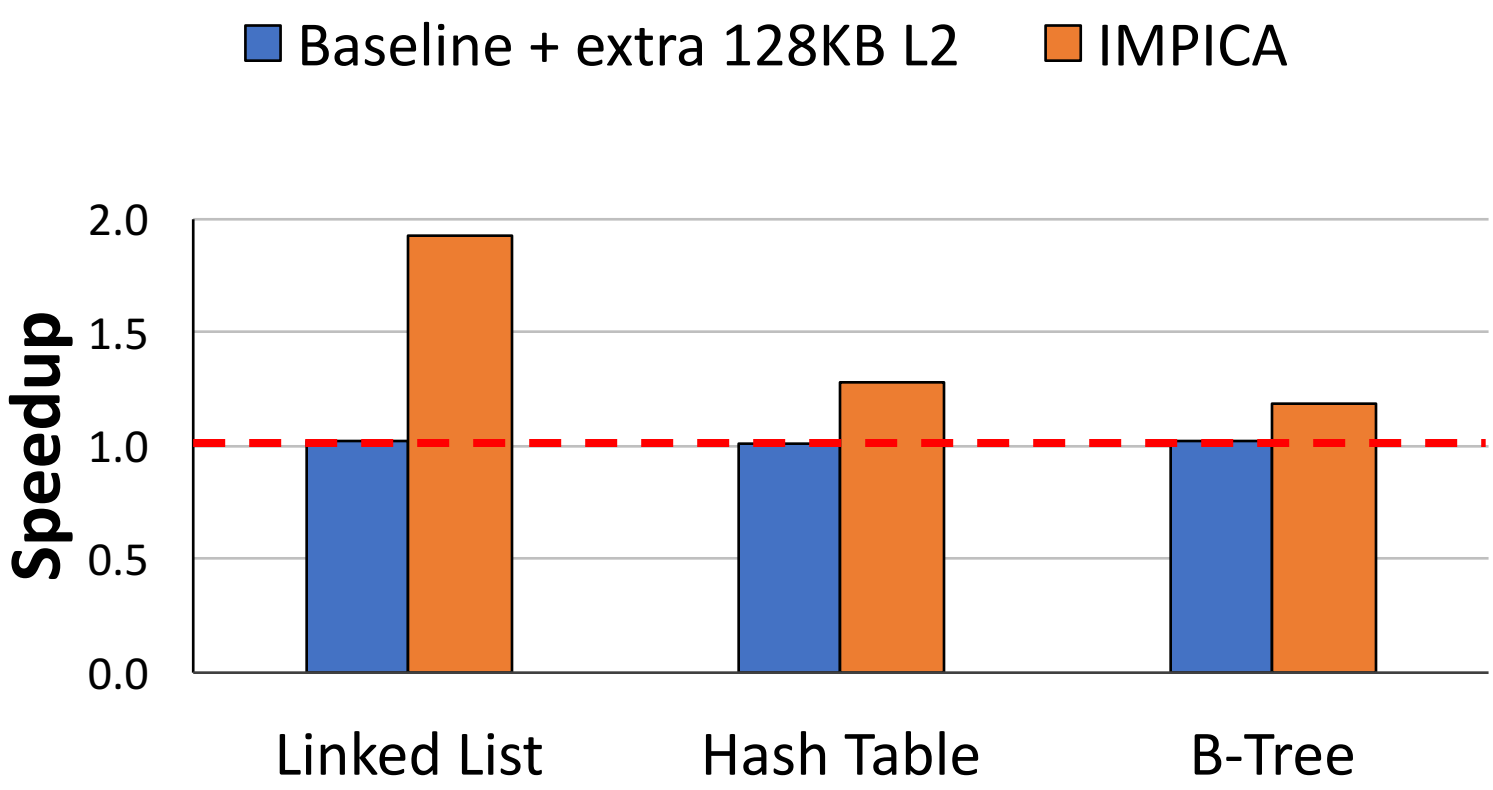
- Simulator: gem5
- System Configuration
  - CPU
    - 4 OoO cores, 2GHz
    - Cache: 32KB L1, 1MB L2
  - IMPICA
    - 1 core, 500MHz, 32KB Cache
  - Memory Bandwidth
    - 12.8 GB/s for CPU, 51.2 GB/s for IMPICA
- Our simulator code will be released in Dec.
  - https://github.com/CMU-SAFARI

Result – Microbenchmark Performance

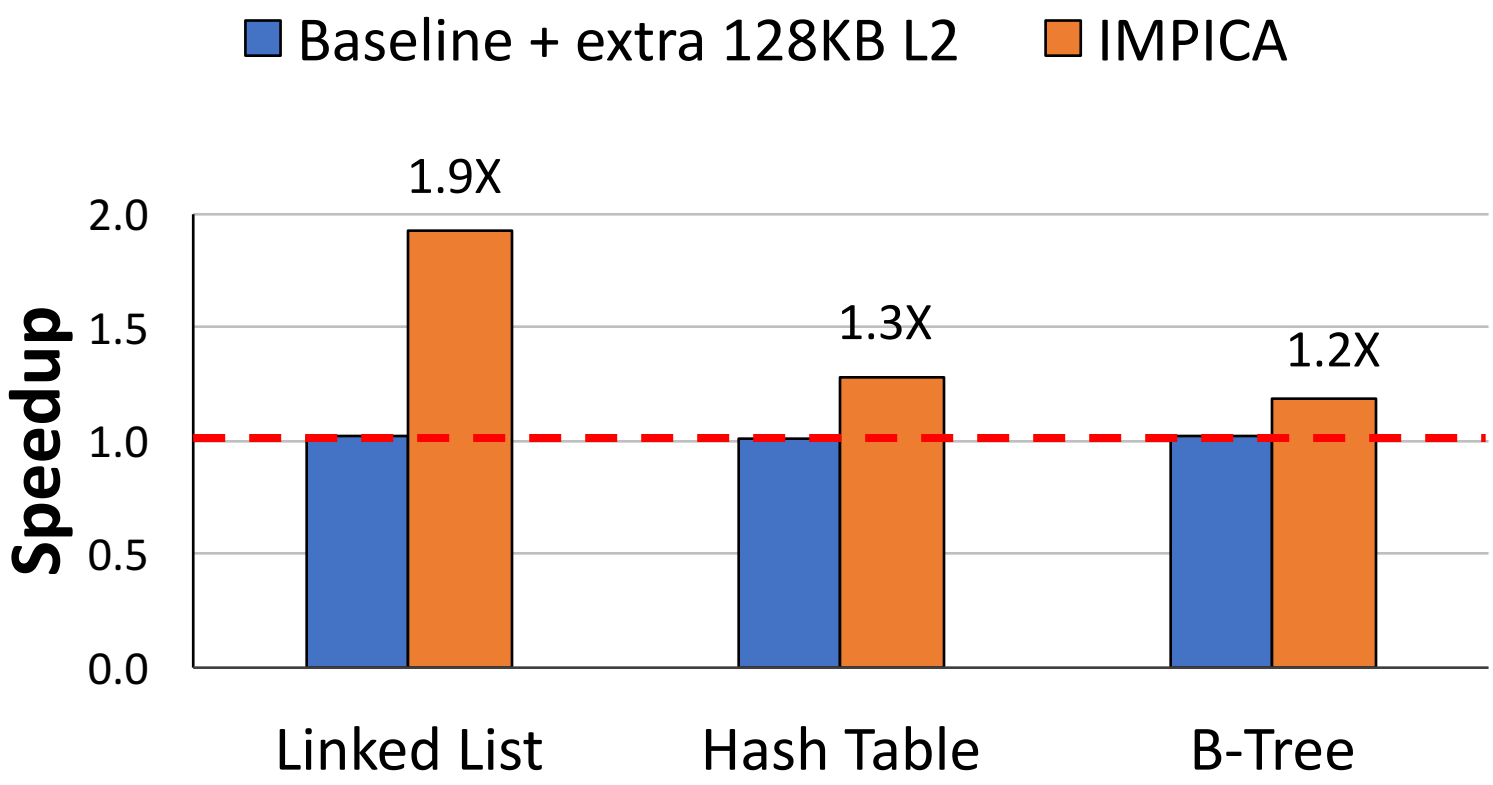
Result – Microbenchmark Performance



Result – Microbenchmark Performance

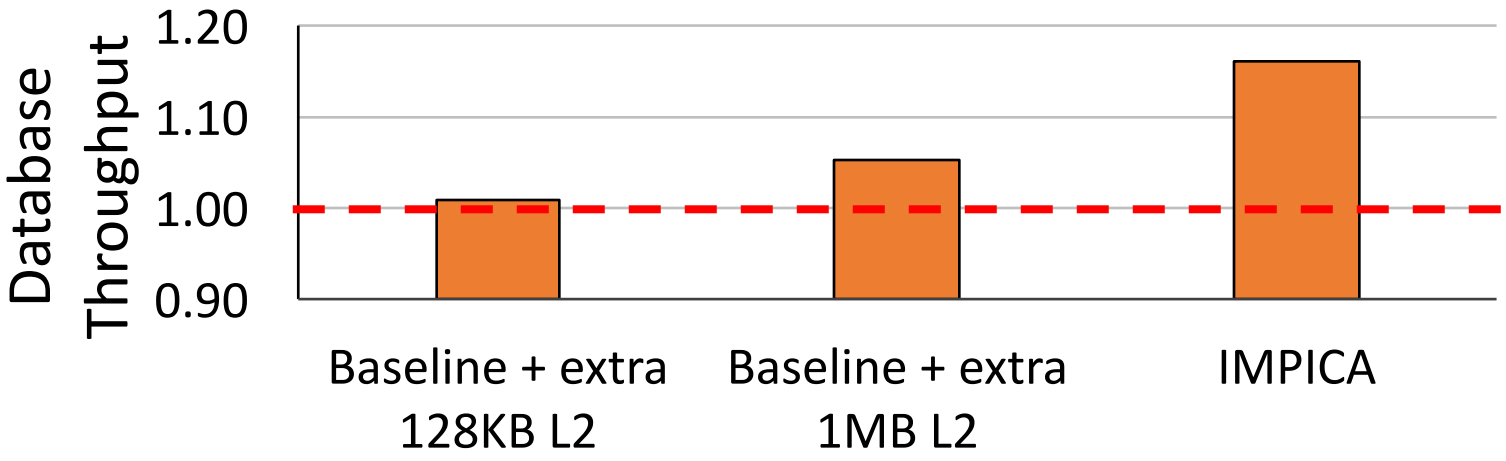


Result – Microbenchmark Performance

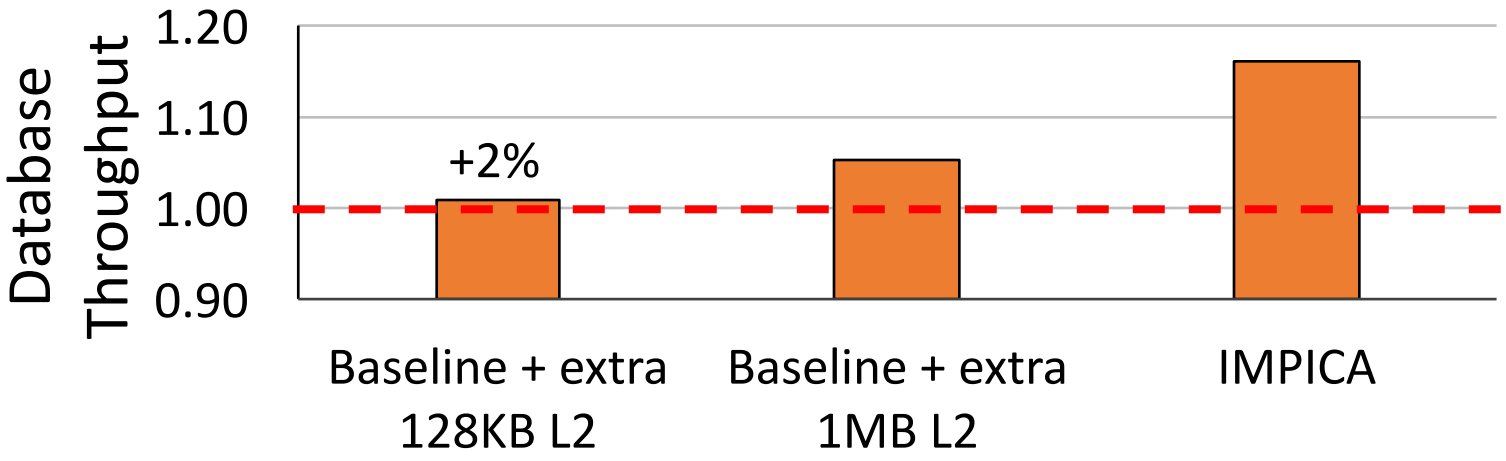


Result – Database Performance

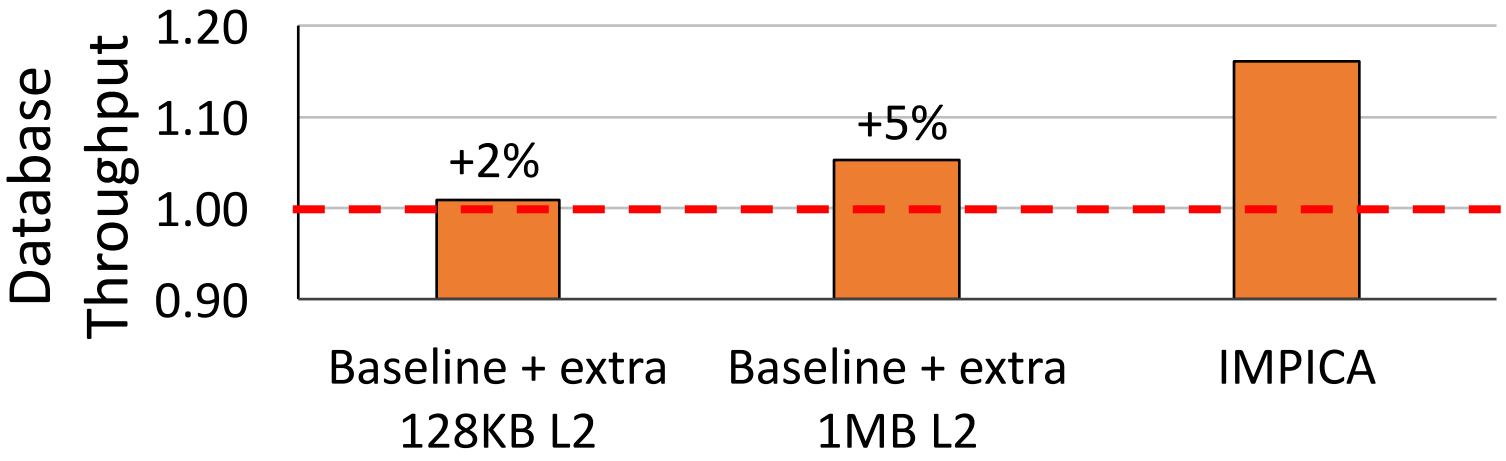
Result – Database Performance



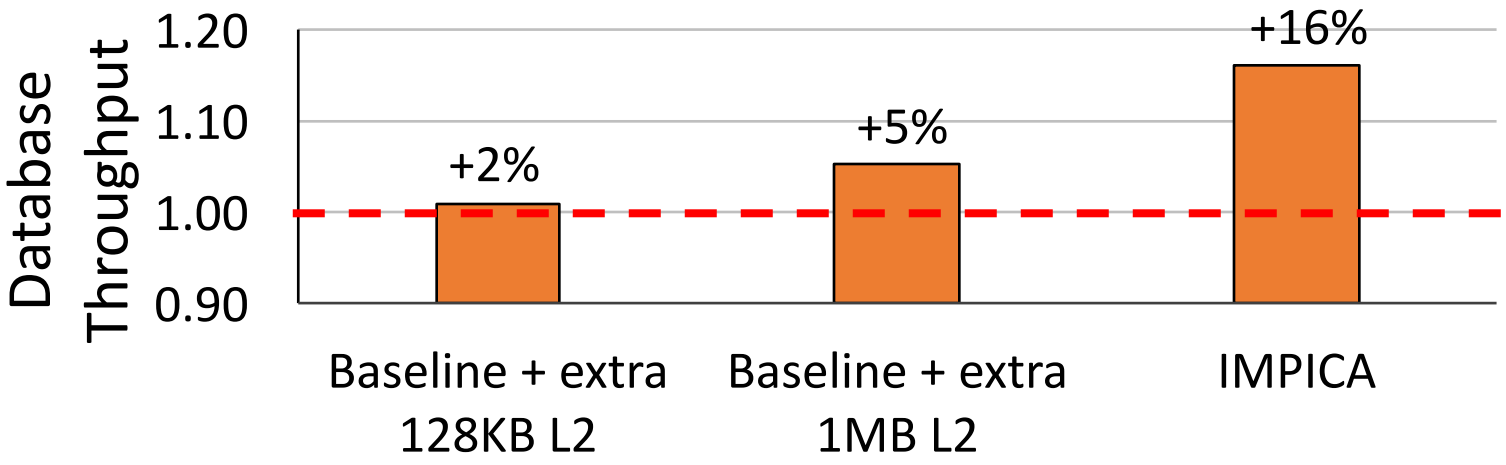
Result – Database Performance



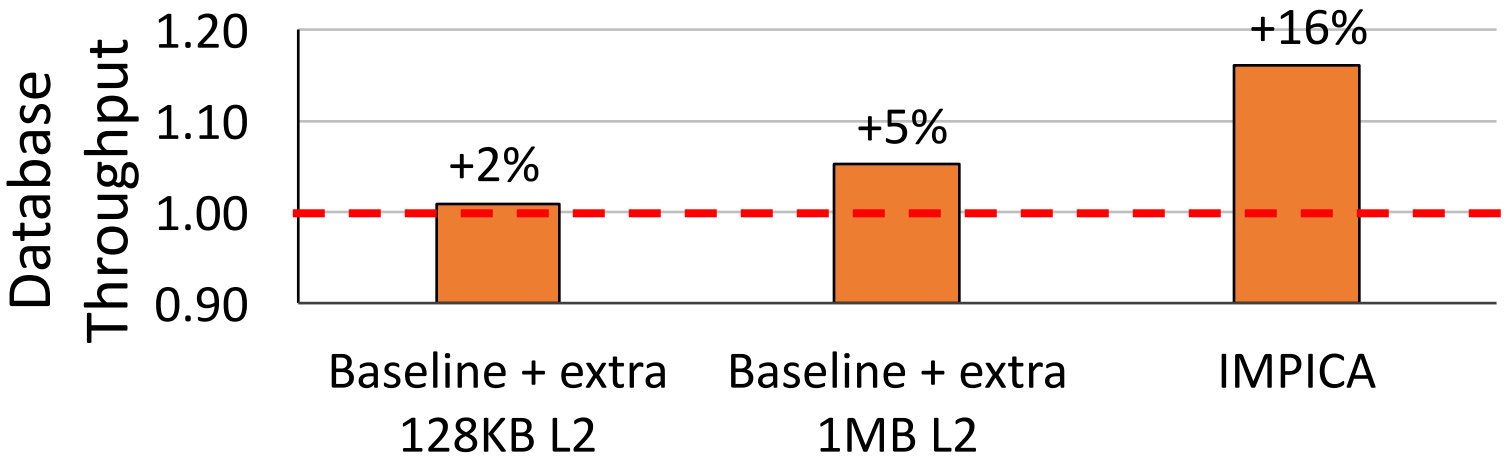
Result – Database Performance

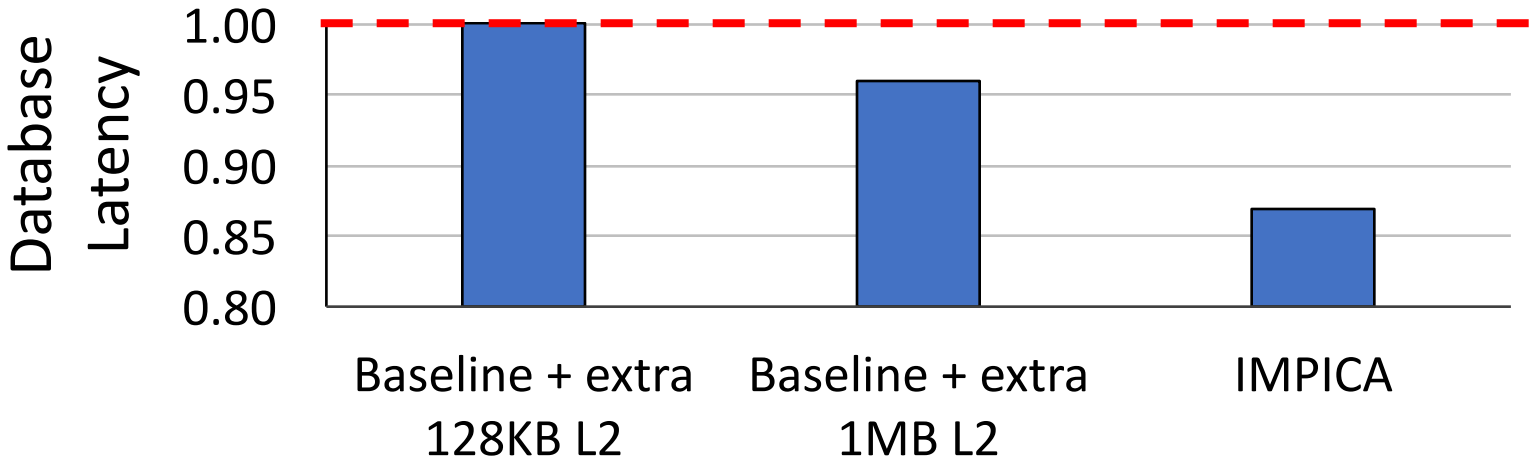


Result – Database Performance

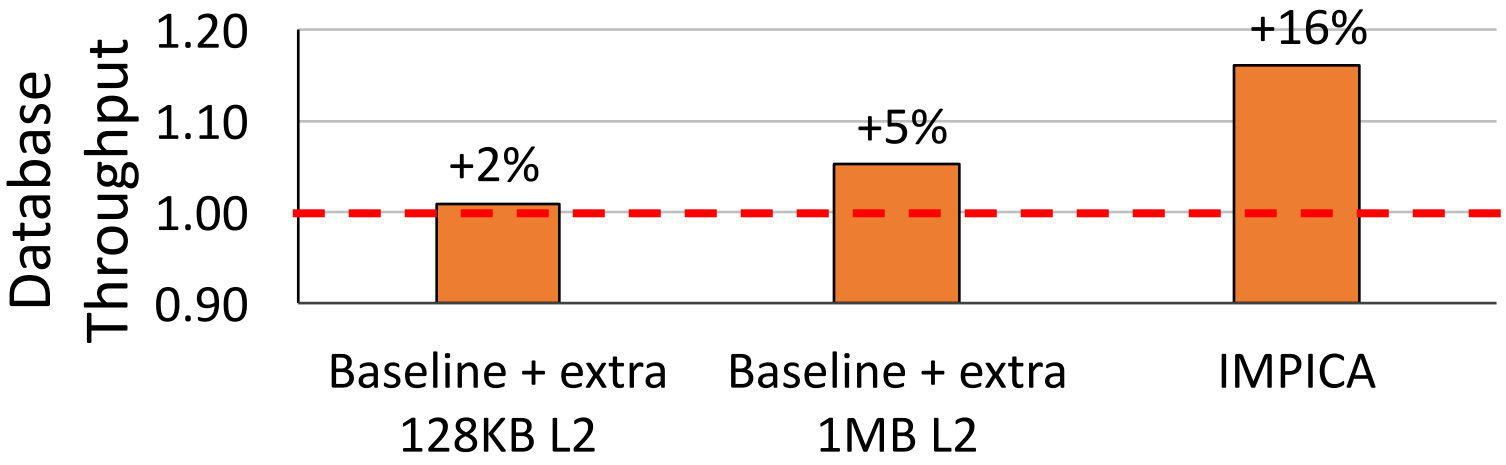


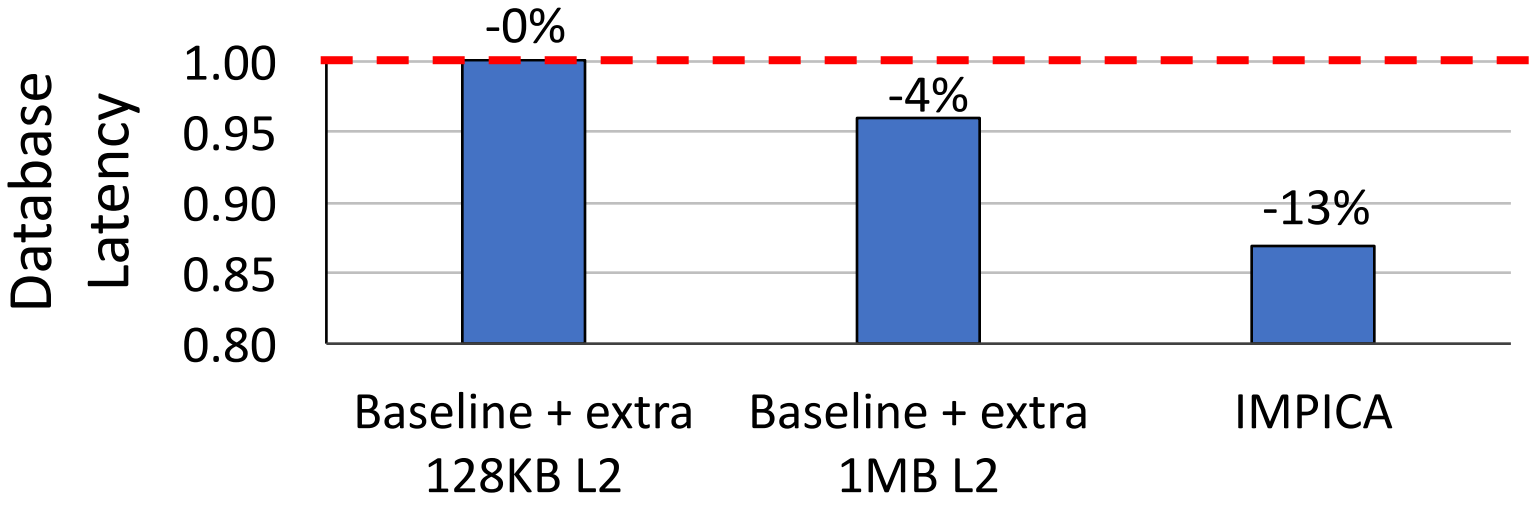
Result – Database Performance





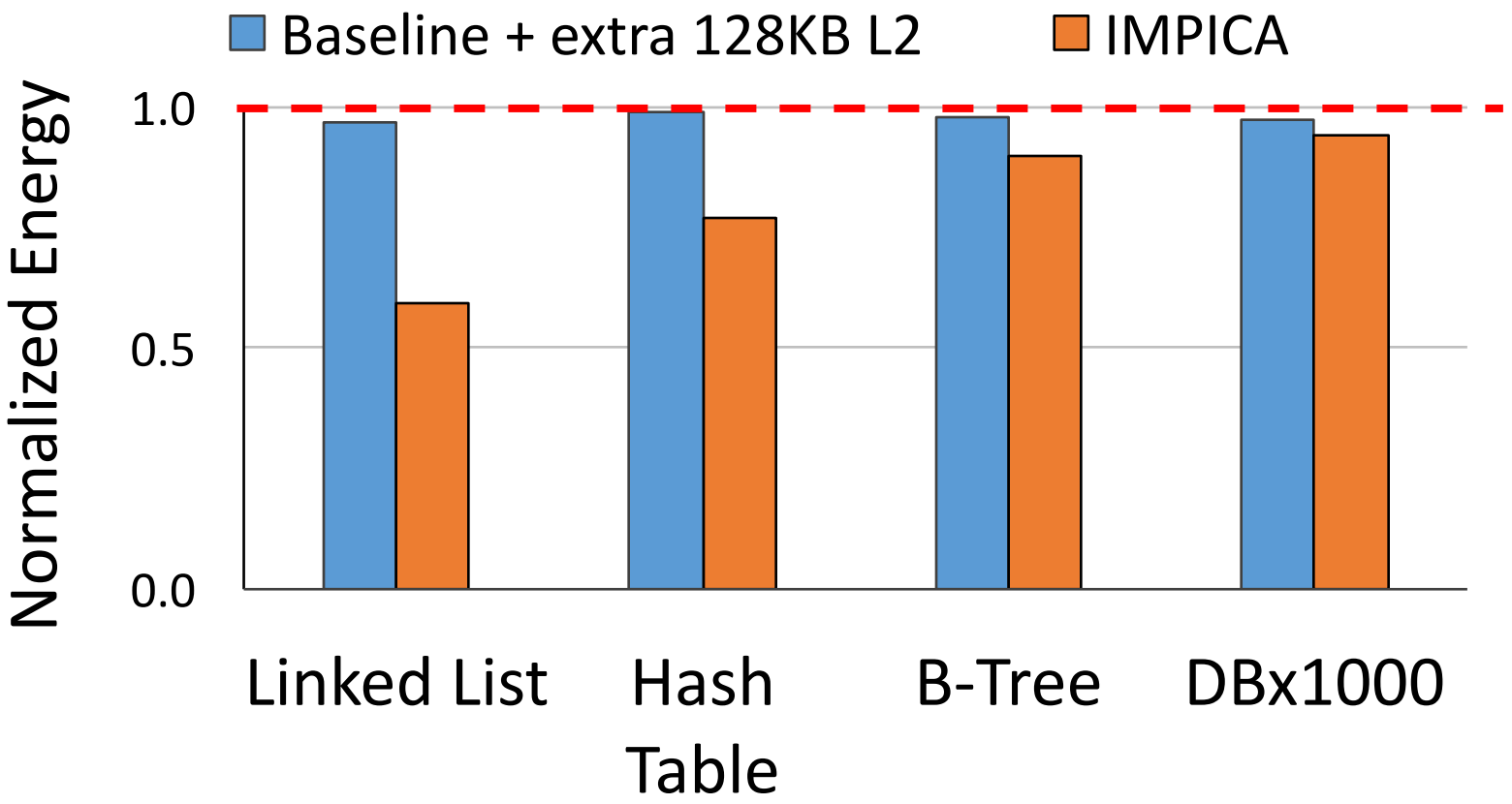
Result – Database Performance



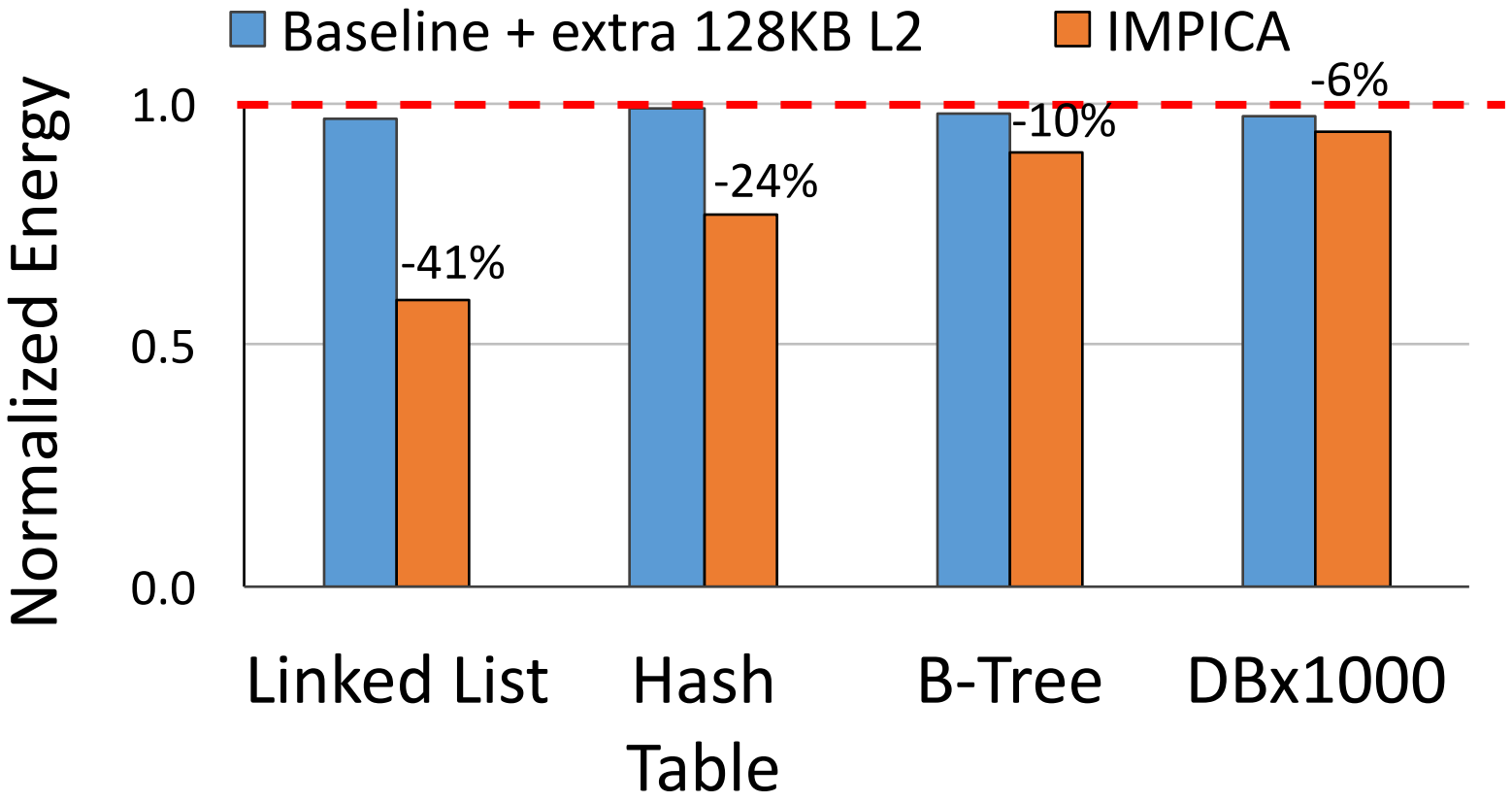


Energy Consumption

Energy Consumption



Energy Consumption



More in the Paper

- Interface and design considerations
  - CPU interface and programming model
  - Page table management
  - Cache coherence
- Area and power overhead analysis
- Sensitivity to IMPICA page table design

Conclusion

- Performing pointer-chasing inside main memory can greatly speed up the traversal of linked data structures
- **Challenges:** Parallelism challenge and Address translation challenge
- **Our Solution:** In-Memory PoInter Chasing Accelerator
  - Address-access decoupling: enabling parallelism with low cost
  - IMPICA page table: low cost page table structure
- **Key Results**:
  - 1.2X – 1.9X speedup for pointer chasing operations, +16% database throughput
  - 6% - 41% reduction in energy consumption
- Our solution can be applied to a broad class of in-memory accelerators

Accelerating Pointer Chasing in 3D-Stacked Memory: *Challenges, Mechanisms, Evaluation*

Kevin Hsieh

Samira Khan, Nandita Vijaykumar, Kevin K. Chang,

Amirali Boroumand, Saugata Ghose, Onur Mutlu

Carnegie Mellon University

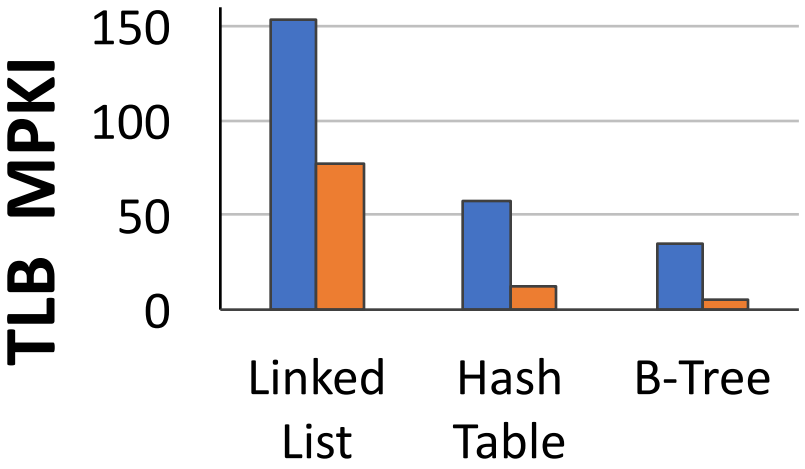
University of Virginia

ETH zürich

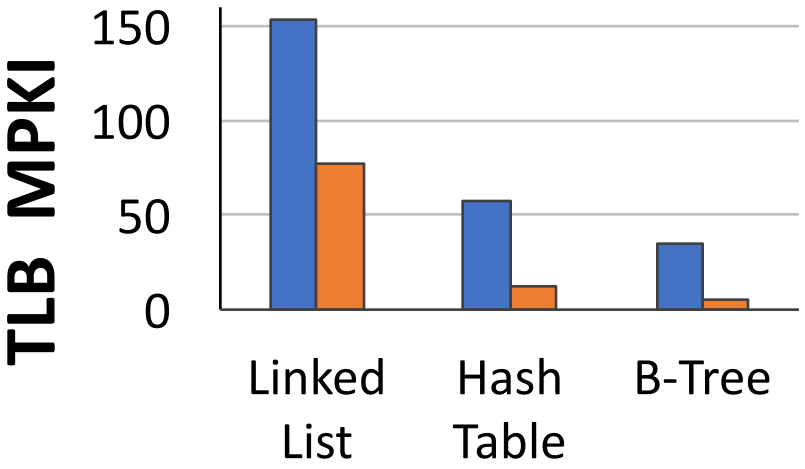
SAFARI

Microarchitecture Metrics

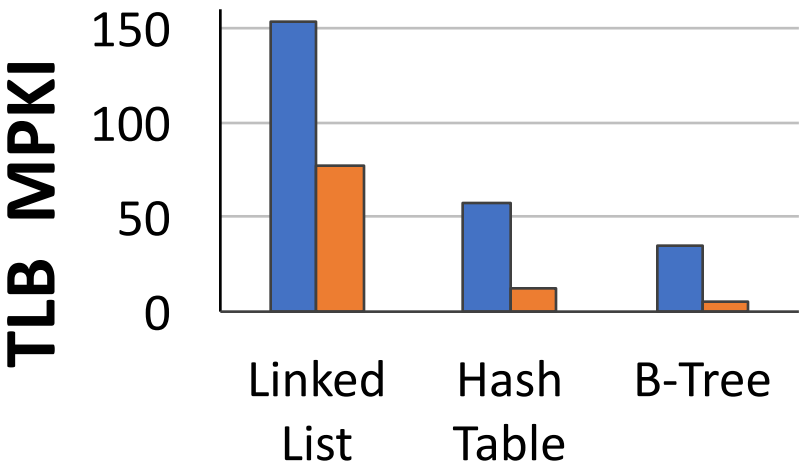
Microarchitecture Metrics

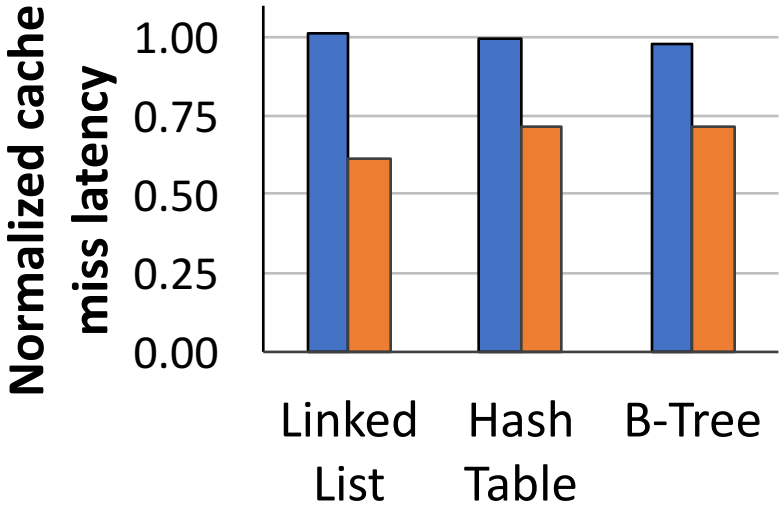


Microarchitecture Metrics

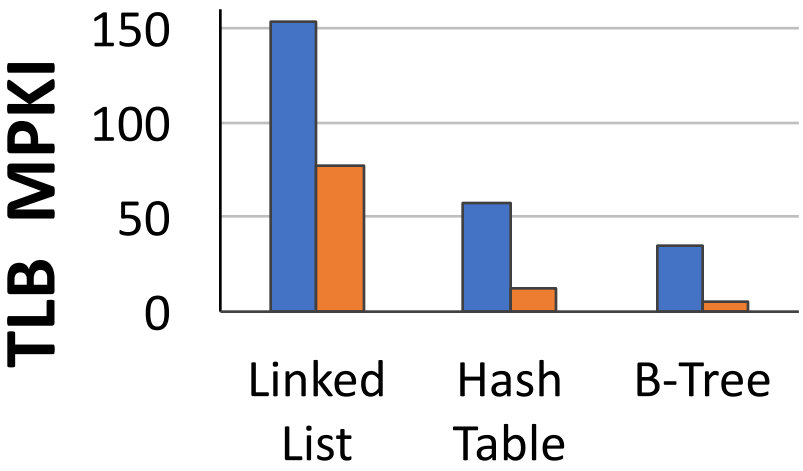


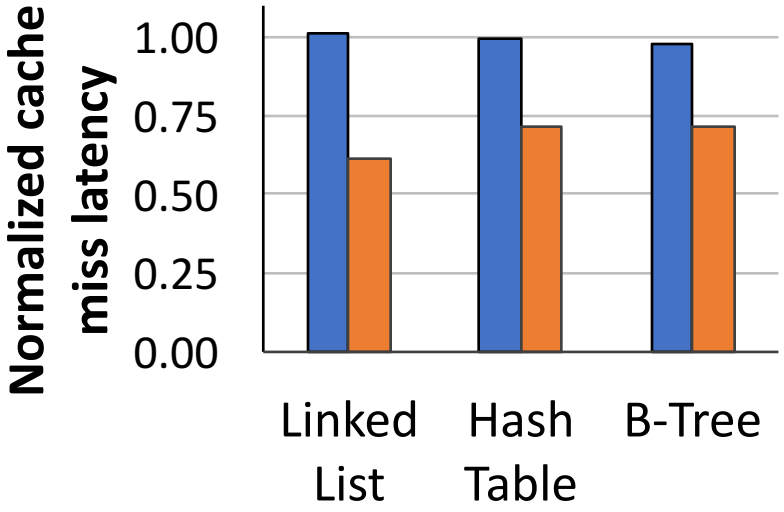
Microarchitecture Metrics



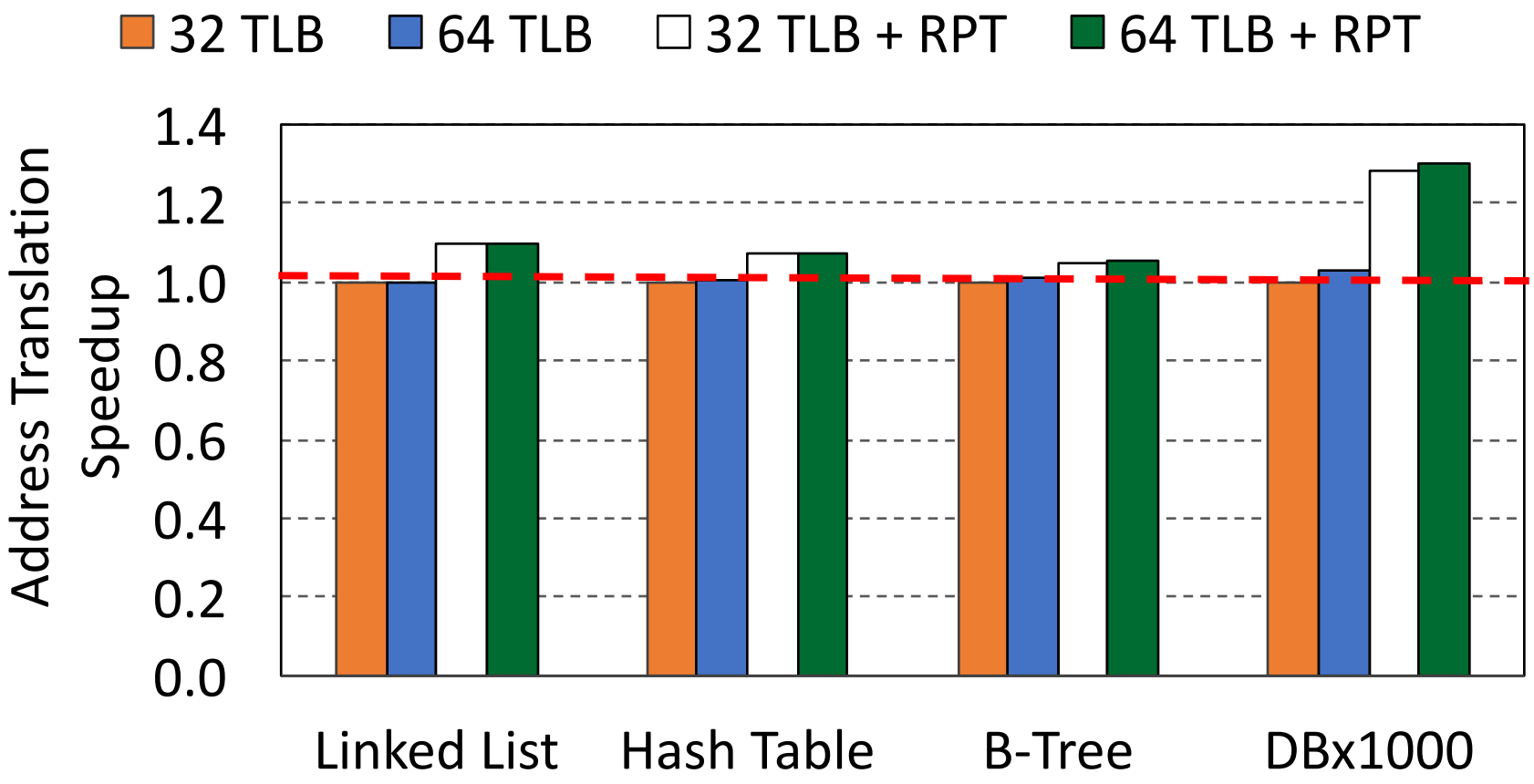


Microarchitecture Metrics

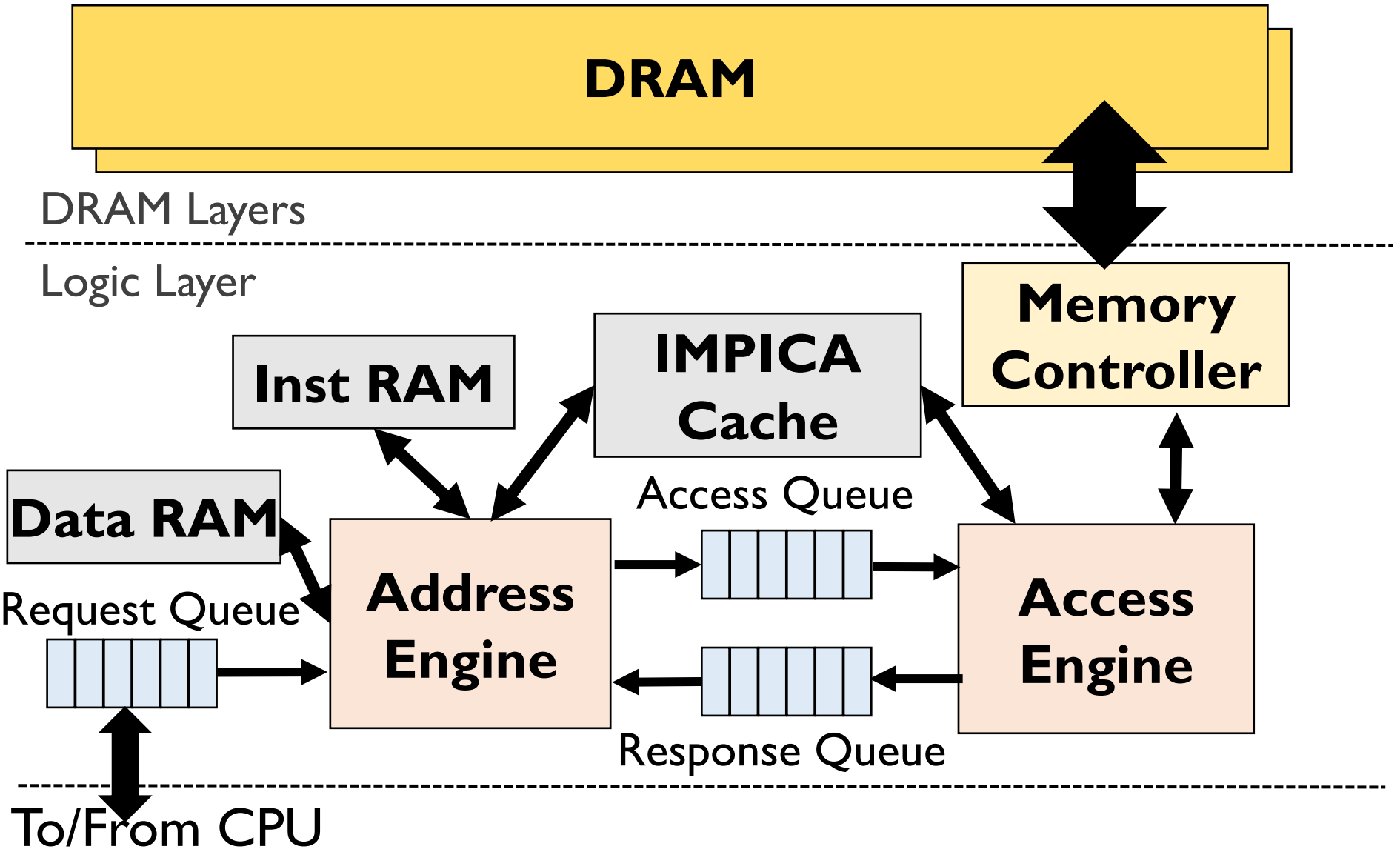




Sensitivity to IMPICA TLB size & Page Table Design



Full IMPICA Core Architecture



CPU Interface

- We use packet-based interface between CPU and IMPICA

- Execution steps
  - CPU sends function call and parameter to IMPICA
  - The packet is written to IMPICA data RAM
  - IMPICA loads the function into inst RAM
  - IMPICA writes results to the data RAM, from which the CPU polls the results.

Programming Model

- An IMPICA program is written as a function in the application code with a compiler directive

- The compiler compiles these functions into IMPICA instructions and wraps the function calls with communication codes

Page Table Management

- The application allocates the memory for its linked data structures with a special API
- The OS reserves a portion of the virtual address space as IMPICA regions
- The OS maintains the coherence between CPU page table and IMPICA page table in the page fault handler

IMPICA Page Table Size

- Region Table
  - 4 entries (covers a 2TB memory range)
  - 68 B
- Flat page table (each)
  - 2^{20} entries
  - 8 MB
- Small page table (each)
  - 2^{9} entries
  - 4 KB

Handling of Multiple Memory Stacks

- The OS knows the IMPICA region because of our page table management

- The OS always maps the IMPICA region of the same application into the same memory stack, including the corresponding IMPICA page table

Cache Coherence

- We execute every function that operates on the IMPICA regions in the accelerator

- It can be extended with more advanced cache coherence mechanism.

Limit of Parallelism

- The parallelism of IMPICA is limited by
  - Data RAM size (for call stacks)
  - Memory access time vs. address computation time
  - The size of the queues

- Each IMPICA core can easily parallelize 10 – 15 pointer chasing requests.

Area and Power Overhead

CPU (Cortex-A57)	5.85 mm^2 per core
L2 Cache	5 mm^2 per MB
Memory Controller	10 mm^2
IMPICA (+32KB cache)	0.45 mm^2

- Power overhead: average power increases by 5.6%