CoNDA: Efficient Cache Coherence Support for Near-Data Accelerators

Amirali Boroumand

Saugata Ghose, Minesh Patel, Hasan Hassan,
Brandon Lucia, Rachata Ausavarungnirun, Kevin Hsieh,
Nastaran Hajinazar, Krishna Malladi, Hongzhong Zheng,
Onur Mutlu

SAFARI

Carnegie Mellon

SAMSUNG

SFU

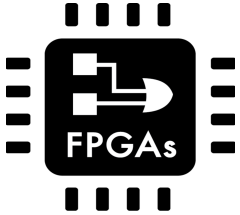
KING MONGKUT'S UNIVERSITY OF TECHNOLOGY NORTH BANGKOK

ETH zürich

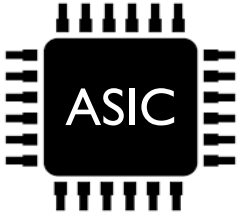
Specialized Accelerators

Specialized accelerators are now everywhere!

GPU

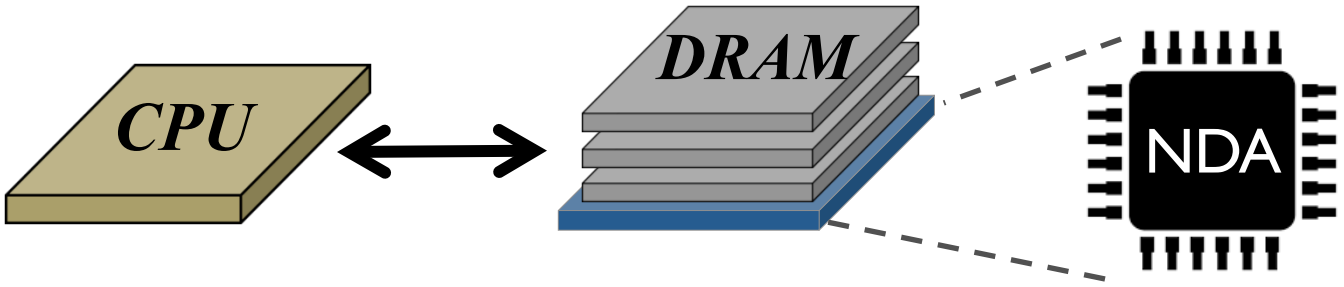


FPGA



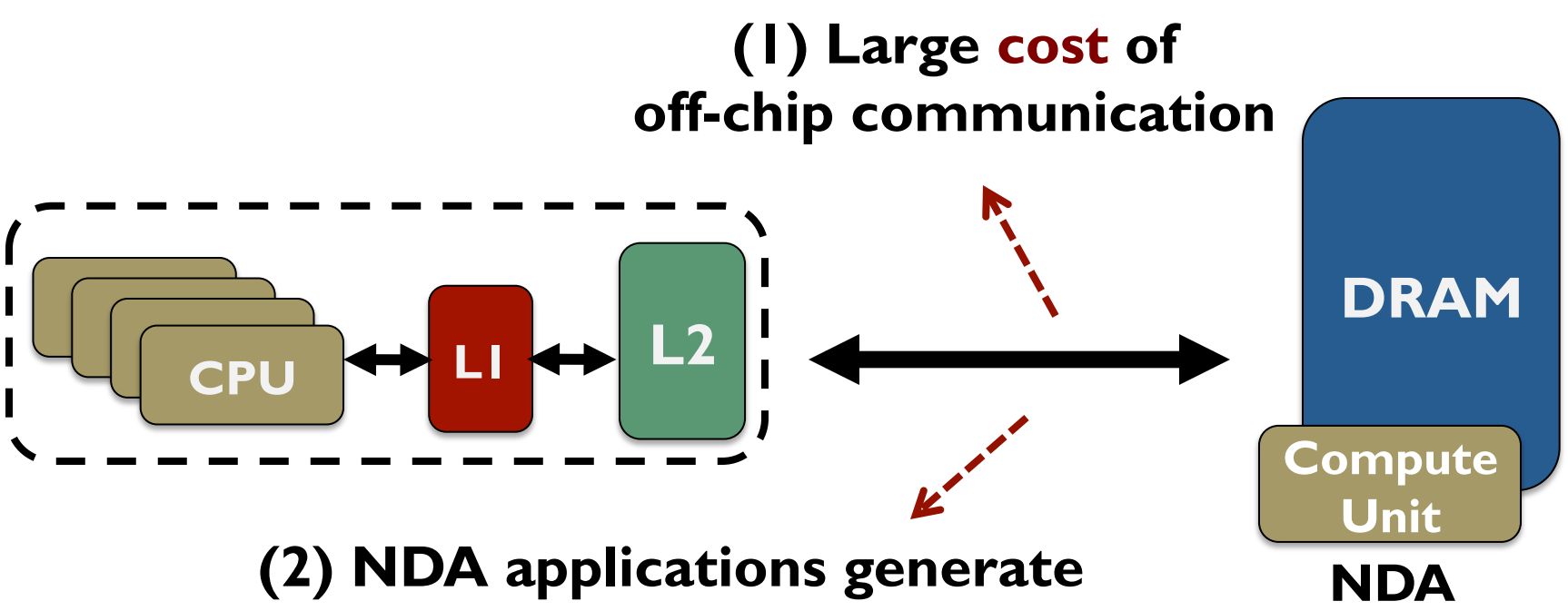
ASIC

Recent advancement in 3D-stacked technology enabled Near-Data Accelerators (NDA)



Coherence For NDAs

Challenge: Coherence between NDAs and CPUs



It is impractical to use traditional coherence protocols

Existing Coherence Mechanisms

We extensively study existing NDA coherence mechanisms and make three key observations:

1. **These mechanisms eliminate a significant portion of NDA's benefits**

2. **The majority of off-chip coherence traffic generated by these mechanisms is unnecessary**

3. **Much of the off-chip traffic can be eliminated if the coherence mechanism has insight into the memory accesses**

An Optimistic Approach

We find that an optimistic approach to coherence can address the challenges related to NDA coherence

1. **Gain insights *before* any coherence checks happen**
2. **Perform only the necessary coherence requests**

We propose CoNDA, a coherence mechanism that lets an NDA optimistically execute an NDA kernel

↓

Optimistic execution enables CoNDA to identify and avoid unnecessary coherence requests

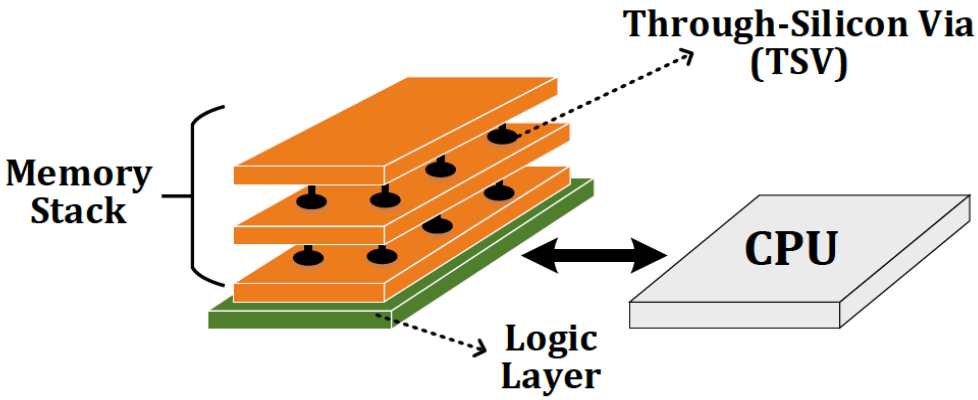
CoNDA comes within 10.4% and 4.4% of performance and energy of an ideal NDA coherence mechanism

Outline

- Introduction
- **Background**
- Motivation
- CoNDA
- Architecture Support
- Evaluation
- Conclusion

Background

- **Near-Data Processing (NDP)**
  - A potential solution to **reduce data movement**
  - **Idea:** move computation close to data
    - ✓ Reduces data movement
    - ✓ Exploits large in-memory bandwidth
    - ✓ Exploits shorter access latency to memory
- **Enabled by recent advances in 3D-stacked memory**



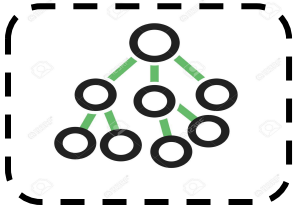
Outline

- Introduction
- Background
- **Motivation**
- CoNDA
- Architecture Support
- Evaluation
- Conclusion

Application Analysis

Sharing Data between NDAs and CPUs

Hybrid Databases (HTAP)



Graph Processing

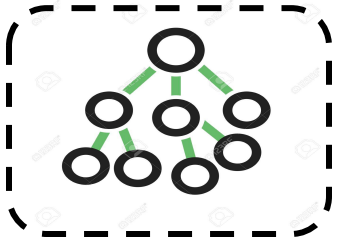
We find not all portions of applications benefit from NDA

1. **Memory-intensive portions benefit from NDA**
2. **Compute-intensive or cache friendly portions should remain on the CPU**

1st key observation: CPU threads often concurrently access the same region of data that NDA kernels access which leads to significant data sharing

Shared Data Access Patterns

2nd key observation: CPU threads and NDA kernels typically *do not* concurrently access the same cache lines



For Connected Components application, only 5.1% of the CPU accesses collide with NDA accesses

CPU threads rarely update the same data that an NDA is actively working on

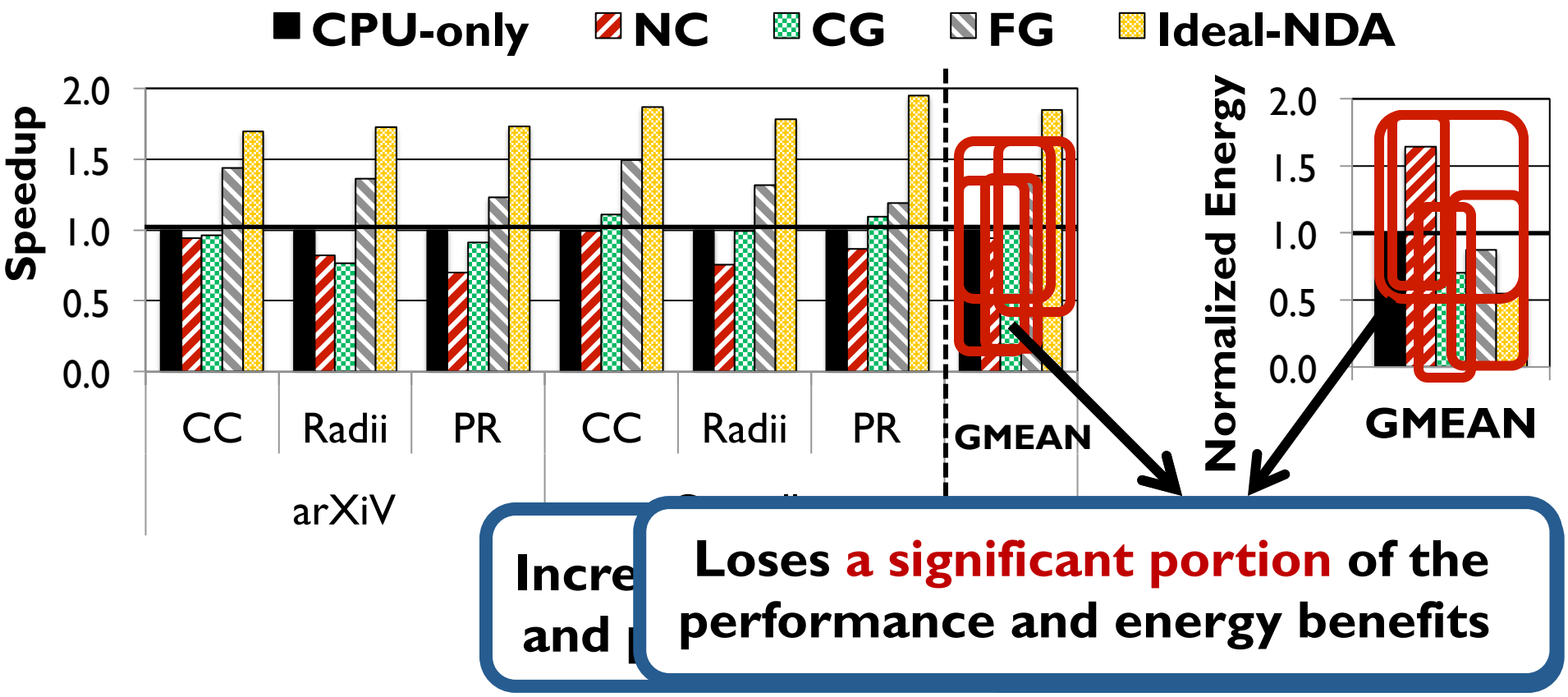
Analysis of NDA Coherence Mechanisms

Analysis of Existing Coherence Mechanism

We analyze three existing coherence mechanisms:

1. **Non-cacheable (NC)**
   - Mark the NDA data as non-cacheable
2. **Coarse-Grained Coherence (CG)**
   - Get coherence permission for the entire NDA region
3. **Fine-Grained Coherence (FG)**
   - Traditional coherence protocols

Analysis of Existing Coherence Mechanisms



Poor handling of coherence eliminates much of an NDA's performance and energy benefits

Motivation and Goal

1 **Poor handling of coherence eliminates much of an NDA's benefits**

2 **The majority of off-chip coherence traffic is unnecessary**

Our goal is to design a coherence mechanism that:

1. **Retains benefits of Ideal NDA**
2. **Enforces coherence with only the necessary data movement**

Outline

- Introduction
- Background
- Motivation
- **CoNDA**
- Architecture Support
- Evaluation
- Conclusion

Optimistic NDA Execution

We leverage two key observations:

1. **Having insight enables us to eliminate much of unnecessary coherence traffic**
2. **Low rate of collision for CPU threads and NDA kernels**

We propose to use optimistic execution for NDAs

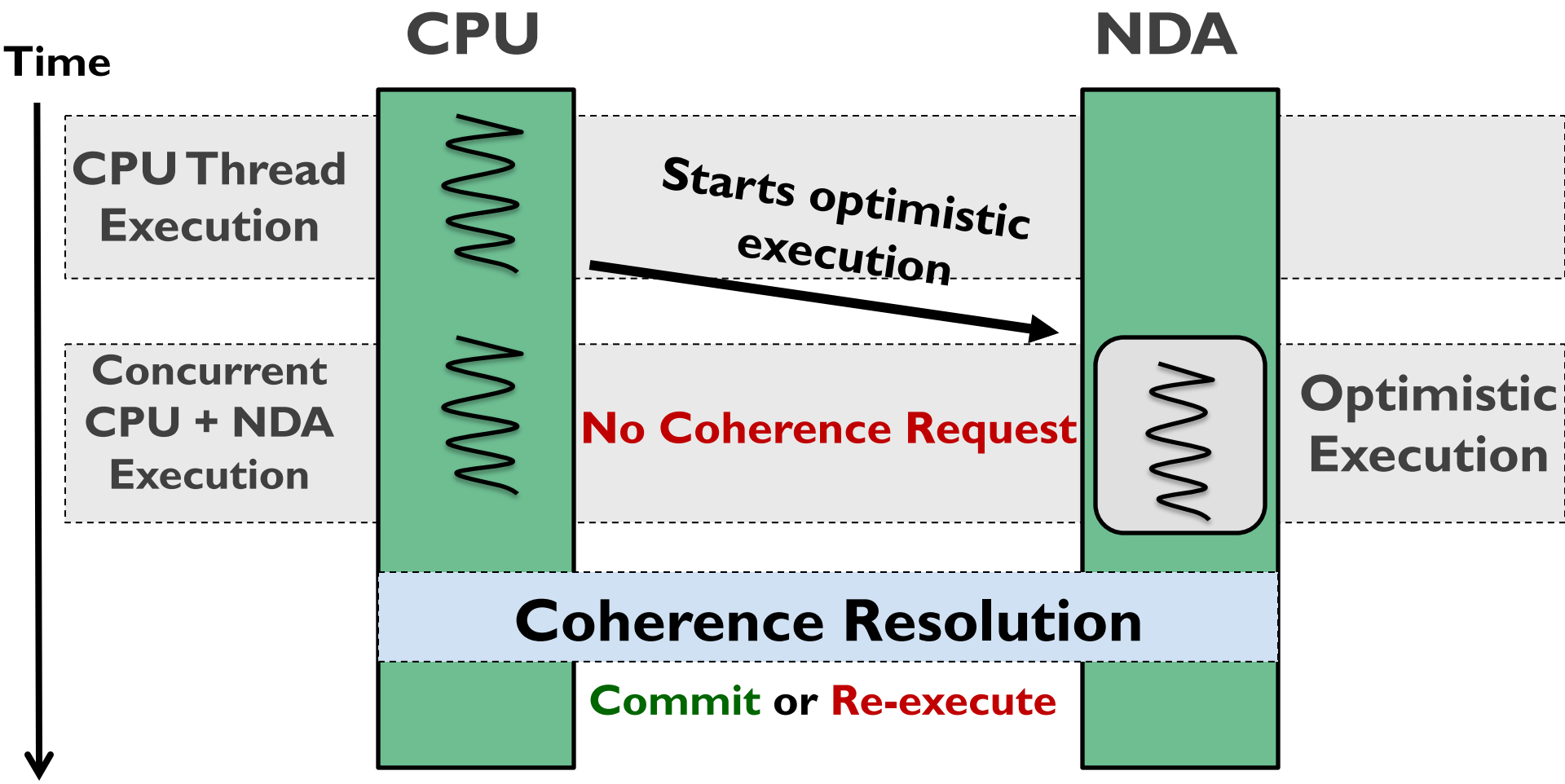
NDA executes the kernel:

1. **Assumes it has coherence permission**
2. **Gains insights into memory accesses**

When execution is done:

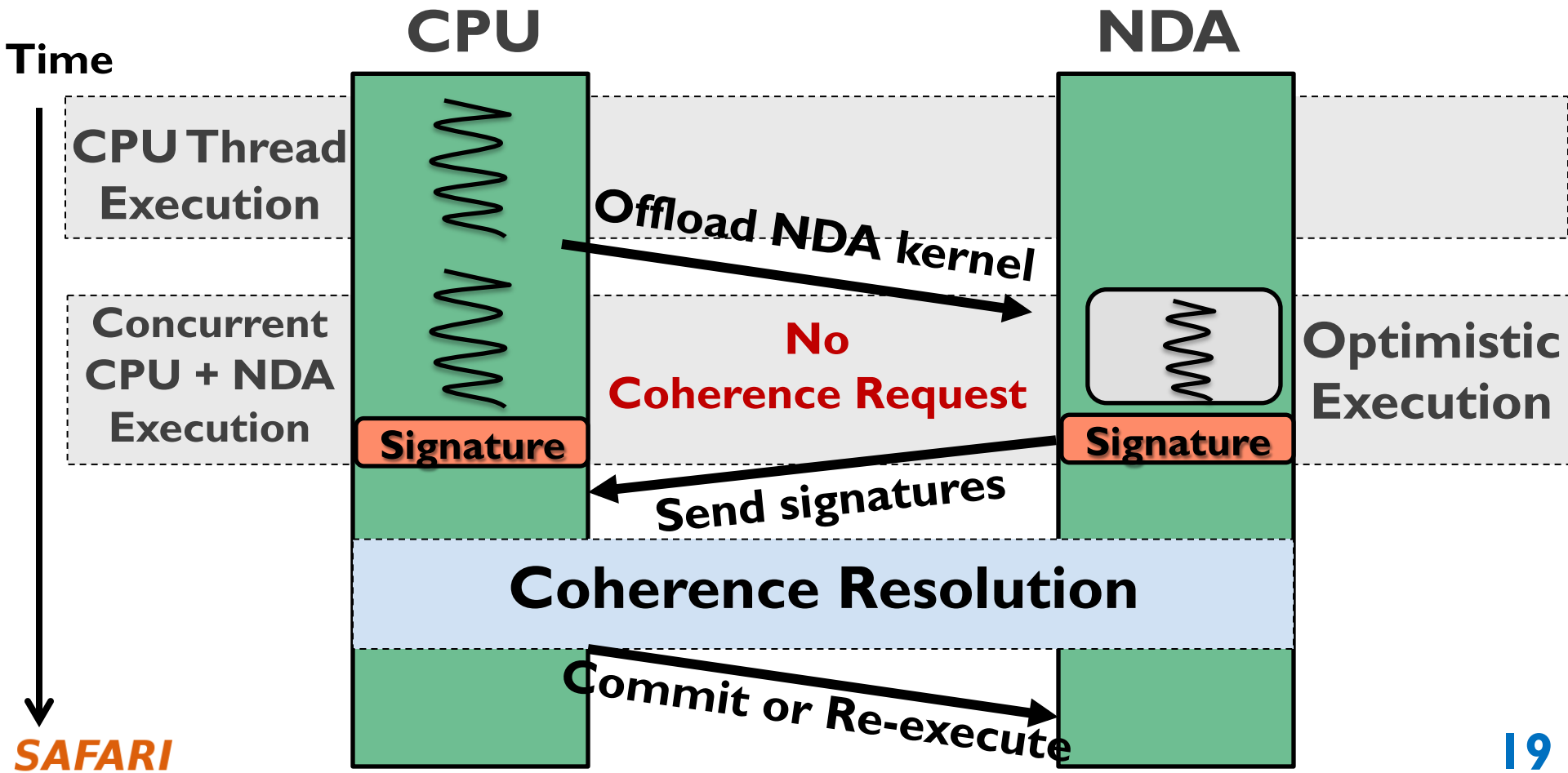
Performs only the necessary coherence requests

High-Level Overview of Optimistic Execution Model



High-Level Overview of CoNDA

We propose CoNDA, a mechanism that uses optimistic NDA execution to avoid unnecessary coherence traffic



How do we identify coherence violations?

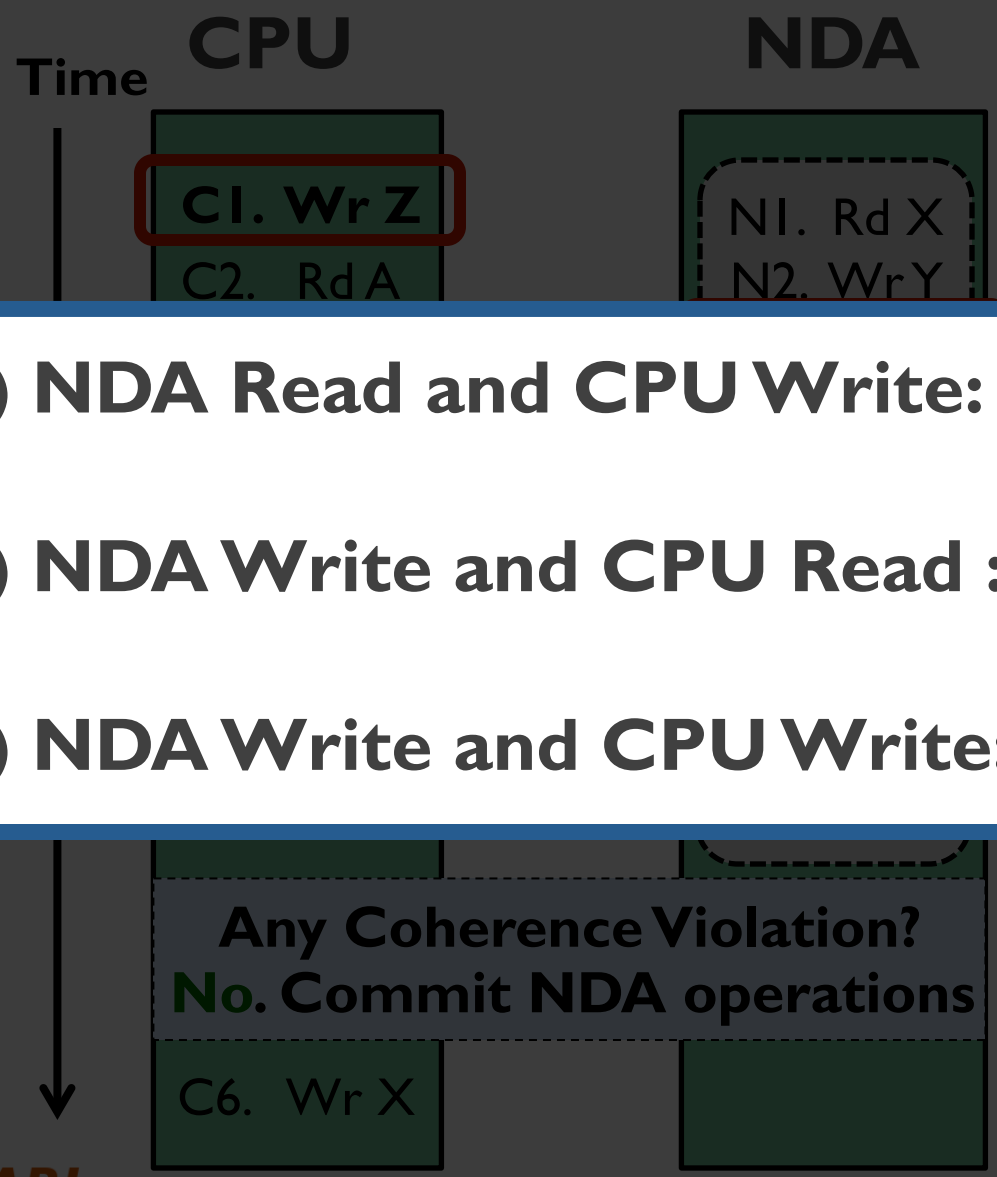
Necessary Coherence Requests

- **Coherence requests are only necessary if:**
  - **Both NDA and CPU access a cache line**
  - **At least one of them updates it**

We discuss three possible interleaving of accesses to the same cache line:

1. **NDA Read and CPU Write (coherence violation)**
2. **NDA Write and CPU Read (no violation)**
3. **NDA Write and CPU Write (no violation)**

Identifying Coherence Violations



No coherence checks during NDA execution

1) NDA Read and CPU Write: **violation**

2) NDA Write and CPU Read : **no violation**

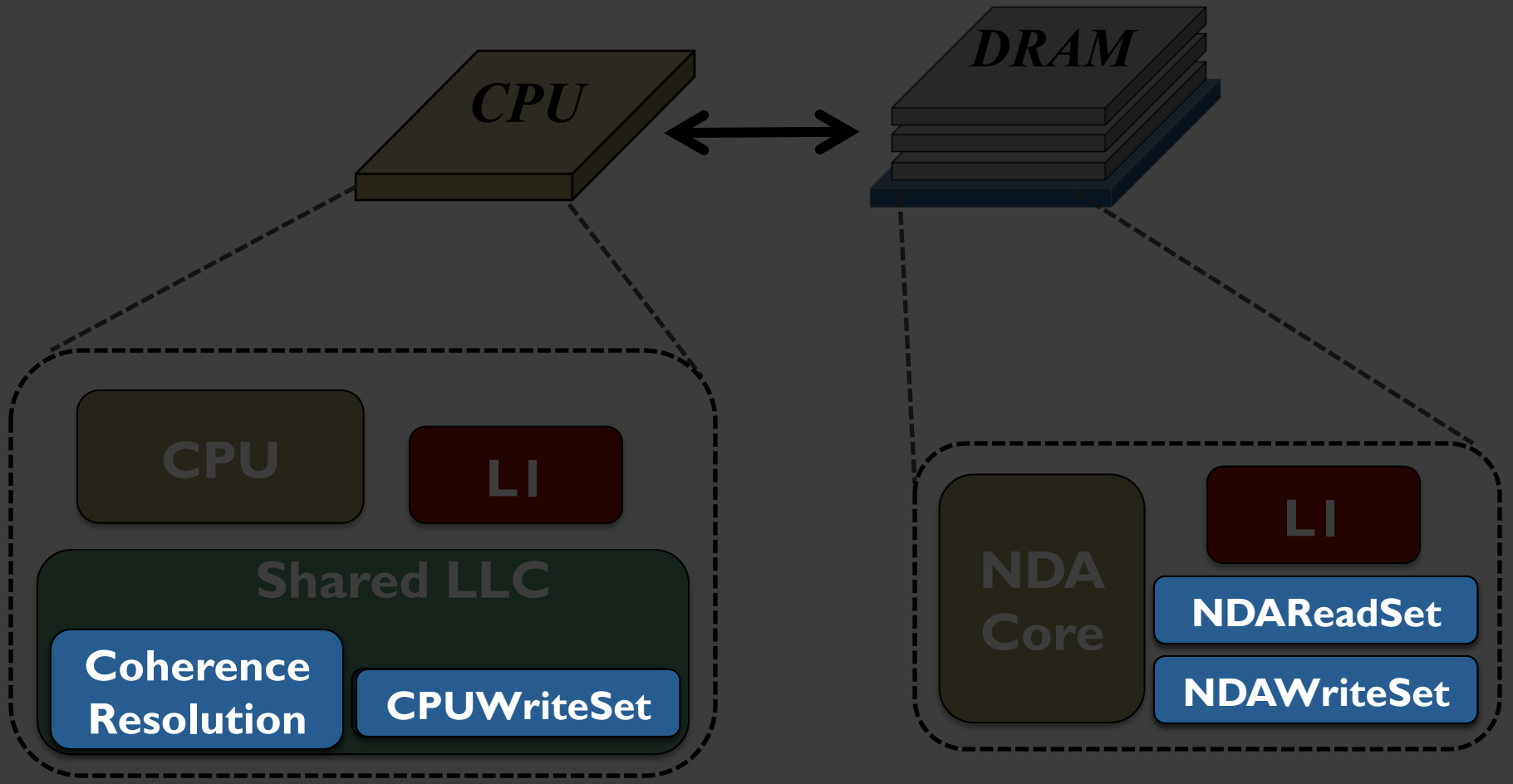
3) NDA Write and CPU Write: **no violation**

"C4" and "C5" are **ordered before** "N5"

Outline

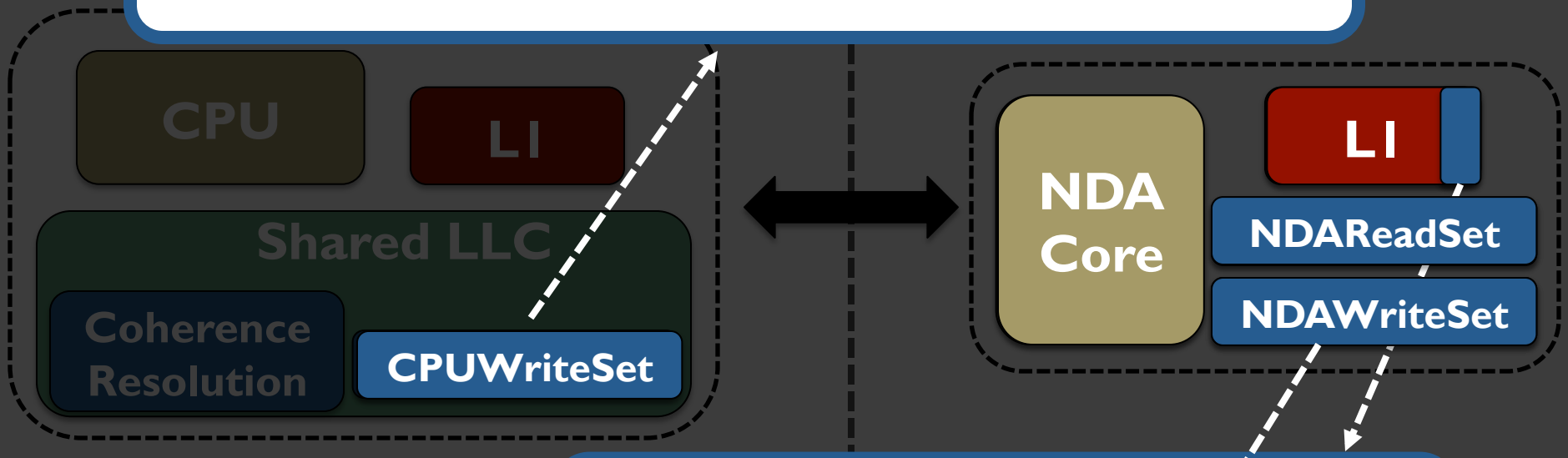
- Introduction
- Background
- Motivation
- CoNDA
- **Architecture Support**
- Evaluation
- Conclusion

CoNDA: Architecture Support



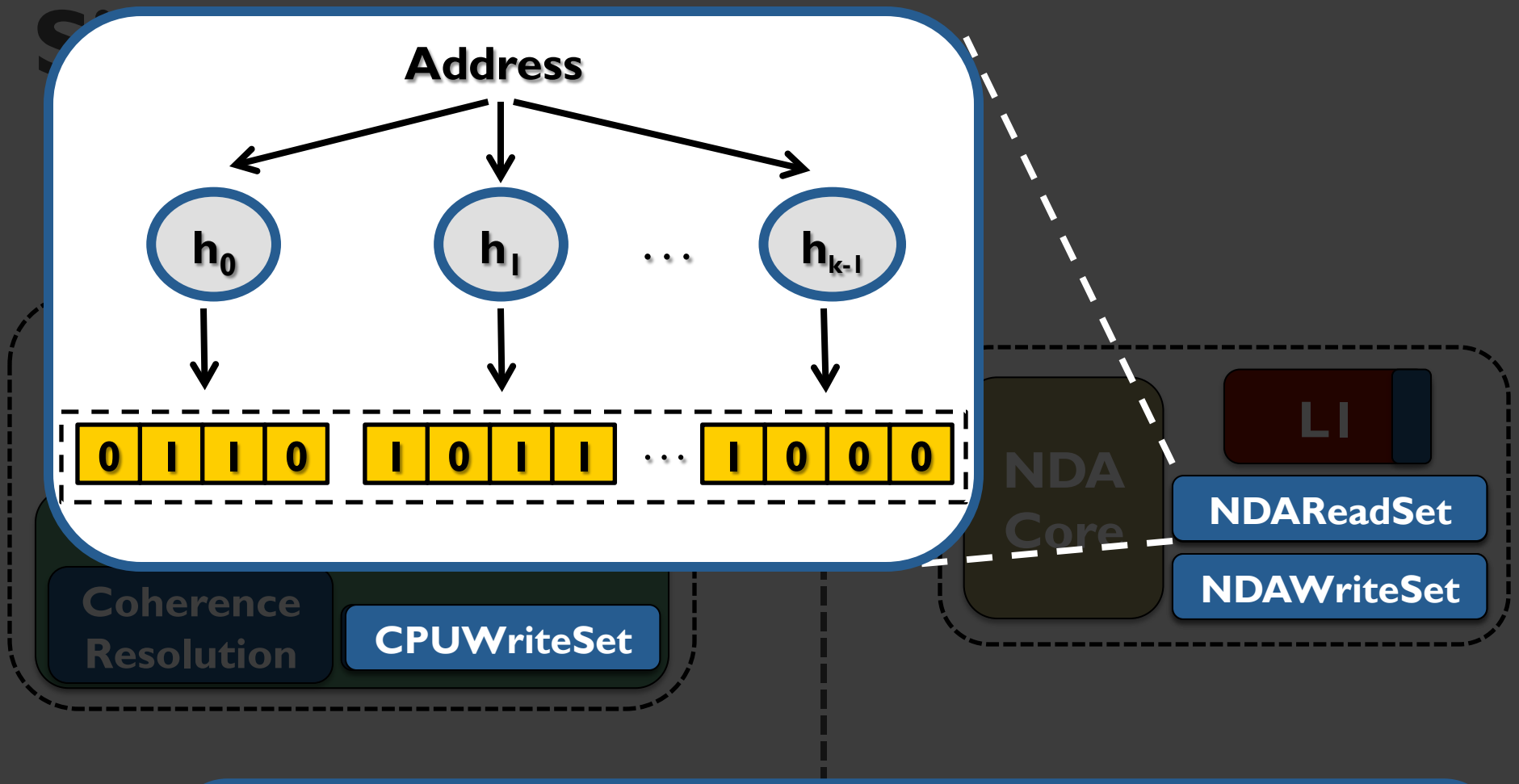
Optimistic Mode Execution

The CPU records all writes to the NDA data region in the CPUWriteSet



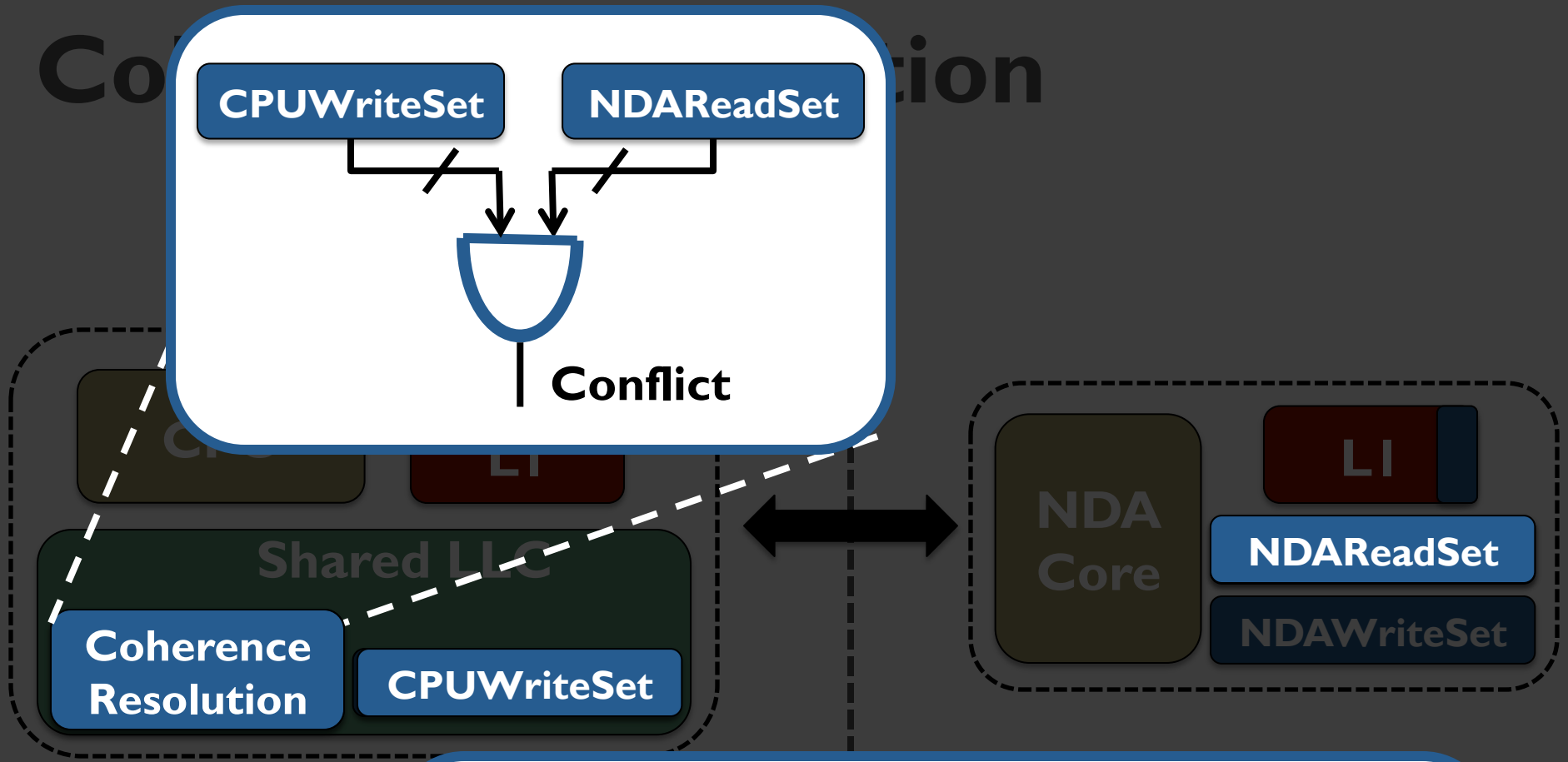
Per-word dirty bit mask to mark all uncommitted data updates

The NDAReadSet and NDAWriteSet are used to track memory accesses from NDA



Bloom filter based signature has two major benefits:

- Allows us to easily perform coherence resolution
- Allows for a large number of addresses to be stored within a fixed-length register



If conflicts happens:

If no conflicts:

- Any clean cache lines in the CPU that match an address in the NDAWriteSet are invalidated
- NDA commits data updates

Outline

- Introduction
- Background
- Motivation
- CoNDA
- Architecture Support
- **Evaluation**
- Conclusion

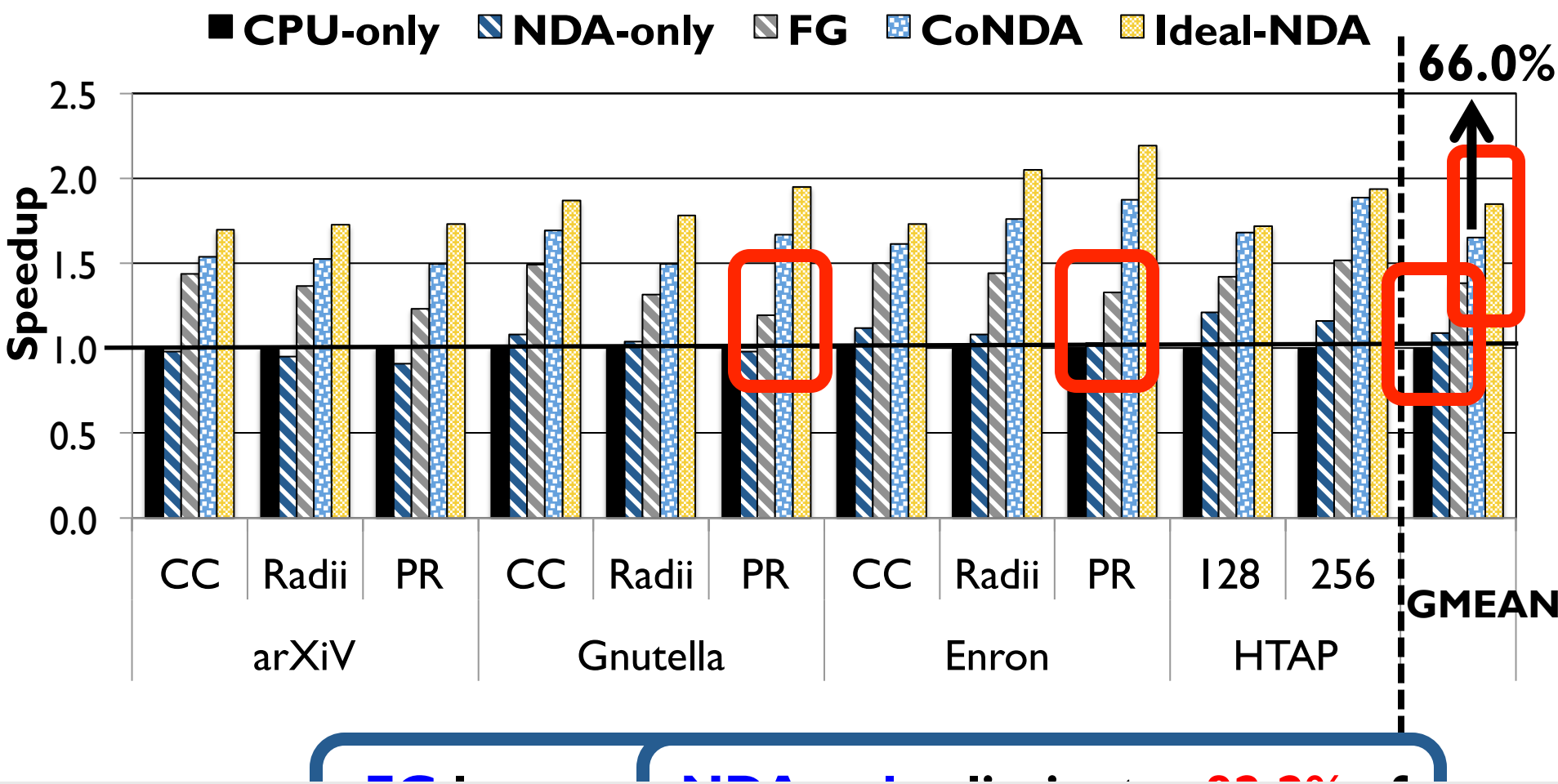
Evaluation Methodology

- **Simulator**
  - Gem5 full system simulator
- **System Configuration:**
  - **CPU**
    - 16 cores, 8-wide, 2GHz frequency
    - L1 I/D cache: 64 kB private, 4-way associative, 64 B block
    - L2 cache: 2 MB shared, 8-way associative, 64 B blocks
    - Cache Coherence Protocol: MESI
  - **NDA**
    - 16 cores, 1-wide, 2GHz frequency
    - L1 I/D cache: 64 kB private, 4-way associative, 64 B Block
    - Cache coherence protocol: MESI
  - **3D-stacked Memory**
    - One 4GB Cube, 16 Vaults per cube

Applications

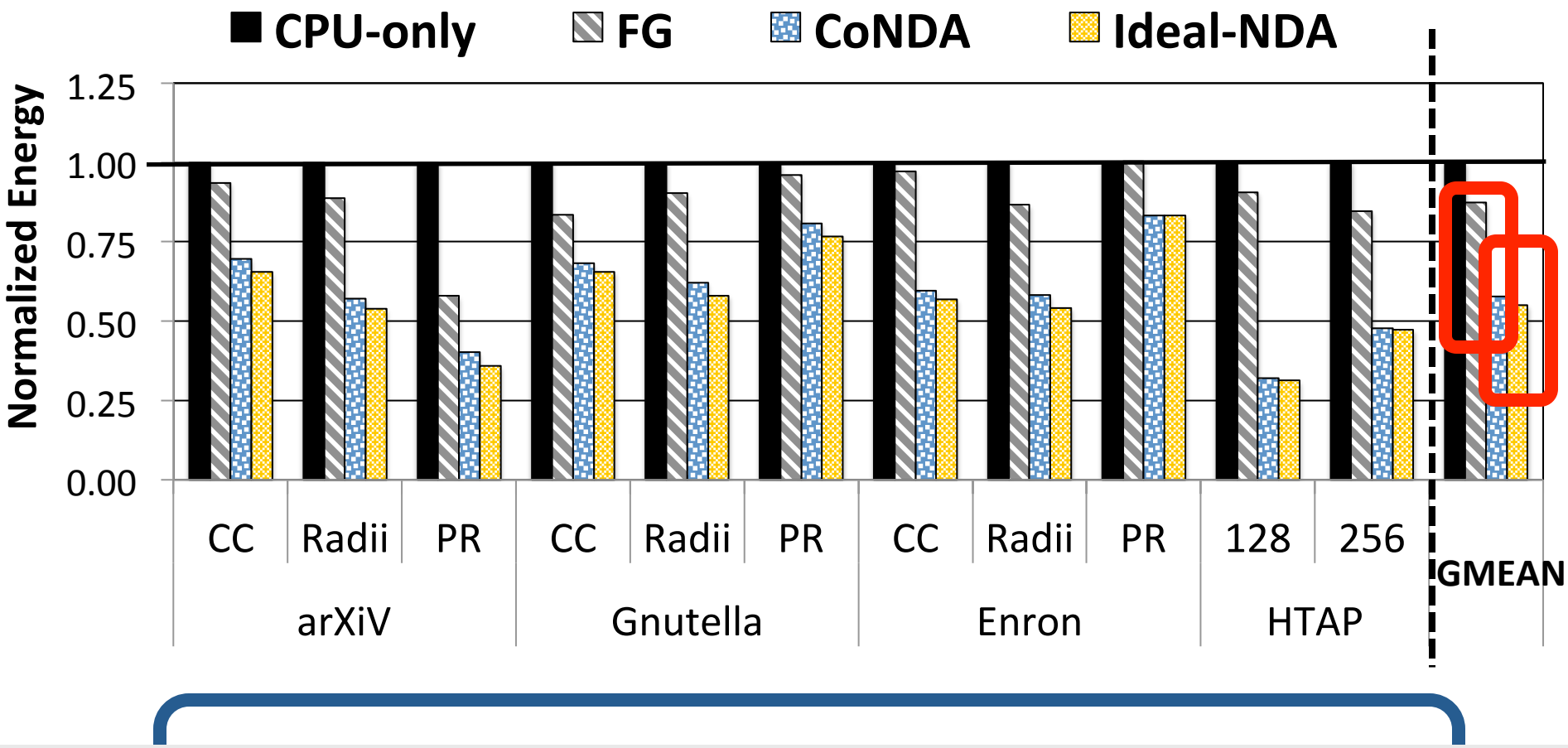
- **Ligra**
  - Lightweight multithreaded graph processing
  - We used three **Ligra** graph applications
    - **PageRank (PR)**
    - **Radii**
    - **Connected Components (CC)**
  - Real-world Input graphs:
    - Enron
    - arXiV
    - Gnutella25
- **Hybrid Database (HTAP)**
  - In-house prototype of an in-memory database
  - Capable of running both **transactional** and **analytical** queries on the **same** database (**HTAP workload**)
  - 32K transactions, 128/256 analytical queries

Speedup



CoNDA consistently retains most of Ideal-NDA's benefits, coming within 10.4% of the Ideal-NDA performance

Memory System Energy



CoNDA significantly reduces energy consumption and comes within 4.4% of Ideal-NDA

Other Results in the Paper

- **Results for larger data sets**
  - **8.4x** over CPU-only
  - **7.7x** over NDA-only
  - **38.3%** over the best prior coherence mechanism

- **Sensitivity analysis**
  - Multiple memory stacks
  - Effect of optimistic execution duration
  - Effect of signature size
  - Effect of data sharing characteristics

- **Hardware overhead analysis**
  - **512 B** NDA signature, **2 kB** CPU signature, **1 bit** per page table, **1 bit** per TLB entry, **1.6%** increase in NDA L1 cache

Outline

- Introduction
- Background
- Motivation
- CoNDA
- Architecture Support
- Evaluation
- **Conclusion**

Conclusion

- Coherence is a major system challenge for NDA
  - Efficient handling of coherence is critical to retain NDA benefits
- We extensively analyze NDA applications and existing coherence mechanisms. Major Observations:
  - There is a significant amount of data sharing between CPU threads and NDAs
  - A majority of off-chip coherence traffic is unnecessary
  - A significant portion of off-chip traffic can be eliminated if the mechanism has insight into NDA memory accesses
- We propose CoNDA, a mechanism that uses optimistic NDA execution to avoid unnecessary coherence traffic
- CoNDA comes within 10.4% and 4.4% of performance and energy of an ideal NDA coherence mechanism

CoNDA: Efficient Cache Coherence Support for Near-Data Accelerators

Amirali Boroumand

Saugata Ghose, Minesh Patel, Hasan Hassan,
Brandon Lucia, Rachata Ausavarungnirun, Kevin Hsieh,
Nastaran Hajinazar, Krishna Malladi, Hongzhong Zheng,
Onur Mutlu

SAFARI

Carnegie Mellon

SAMSUNG

SFU

KING MONGKUT'S UNIVERSITY OF TECHNOLOGY NORTH BANGKOK

ETH zürich

Backup

Breakdown of Performance Overhead

- **CoNDA's execution time consist of three major parts:**
  - (1) NDA kernel execution
  - (2) Coherence resolution overhead (**3.3%** of execution time)
  - (3) Re-execution overhead (**8.4%** of execution time)

- **Coherence resolution overhead is low**
  - CPU-threads **do not stall** during resolution
  - NDAWriteSet contains only **a small number** of addresses (**6**)
  - Resolution mainly involves **sending signatures** and **checking necessary coherence**

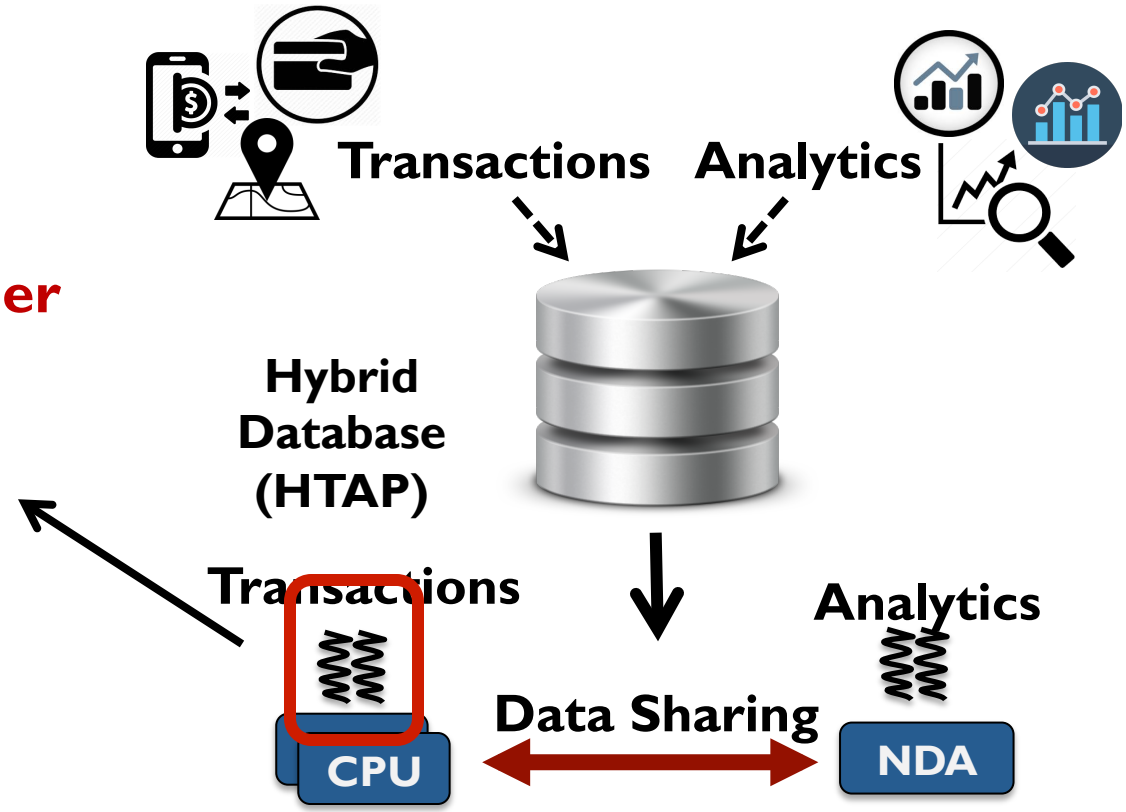
- **Overhead of re-execution is low**
  - The **collision rate** is **low** for our applications → **13.4%**
  - Re-execution is significantly faster than original execution

Non-Cacheable (NC) Approach

Mark the NDA data as non-cacheable

(1) Generates a large number of off-chip accesses

(2) Significantly hurts CPU threads performance

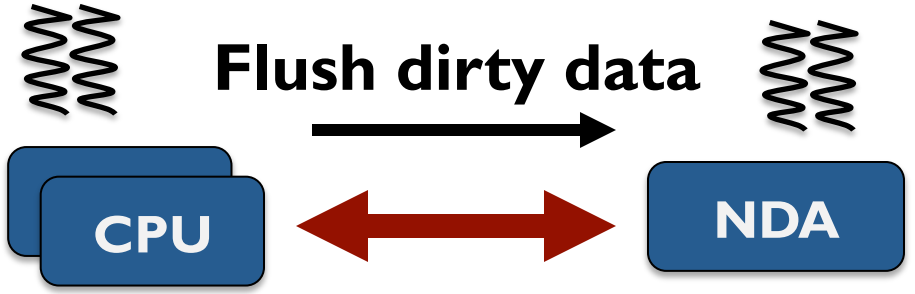


NC fails to provide any energy saving and perform 6.0% worse than CPU-only

Coarse-Grained (CG) Coherence

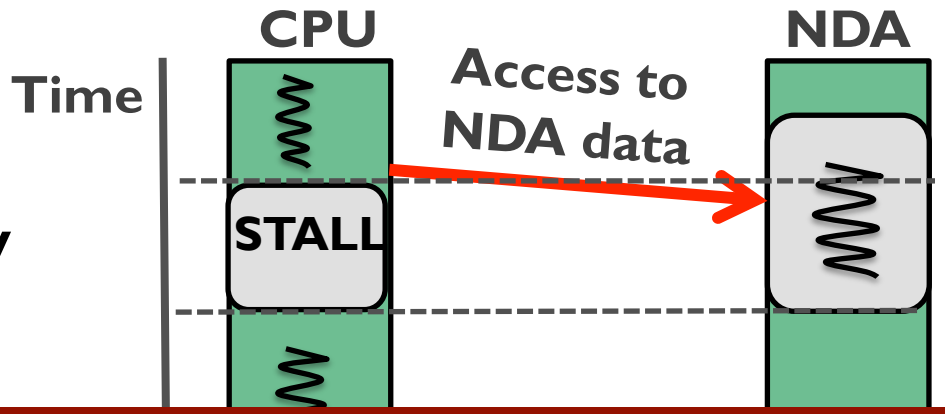
Get coherence permission for the entire NDA region

Unnecessarily flushes a large amount of dirty data, especially in pointer-chasing applications



Use coarse-grained locks to provide exclusive access

Blocks CPU threads when they access NDA data regions

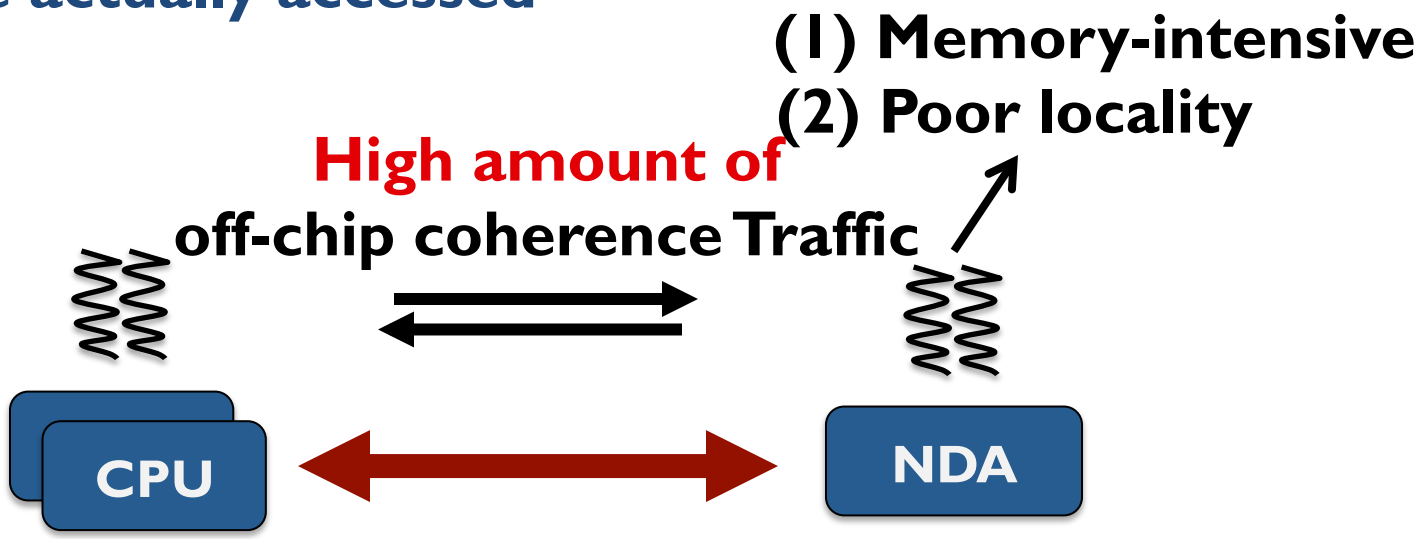


CG fails to provide any performance benefit of NDA

Fine-Grained (FG) Coherence

Using fine-grained coherence has two benefits:

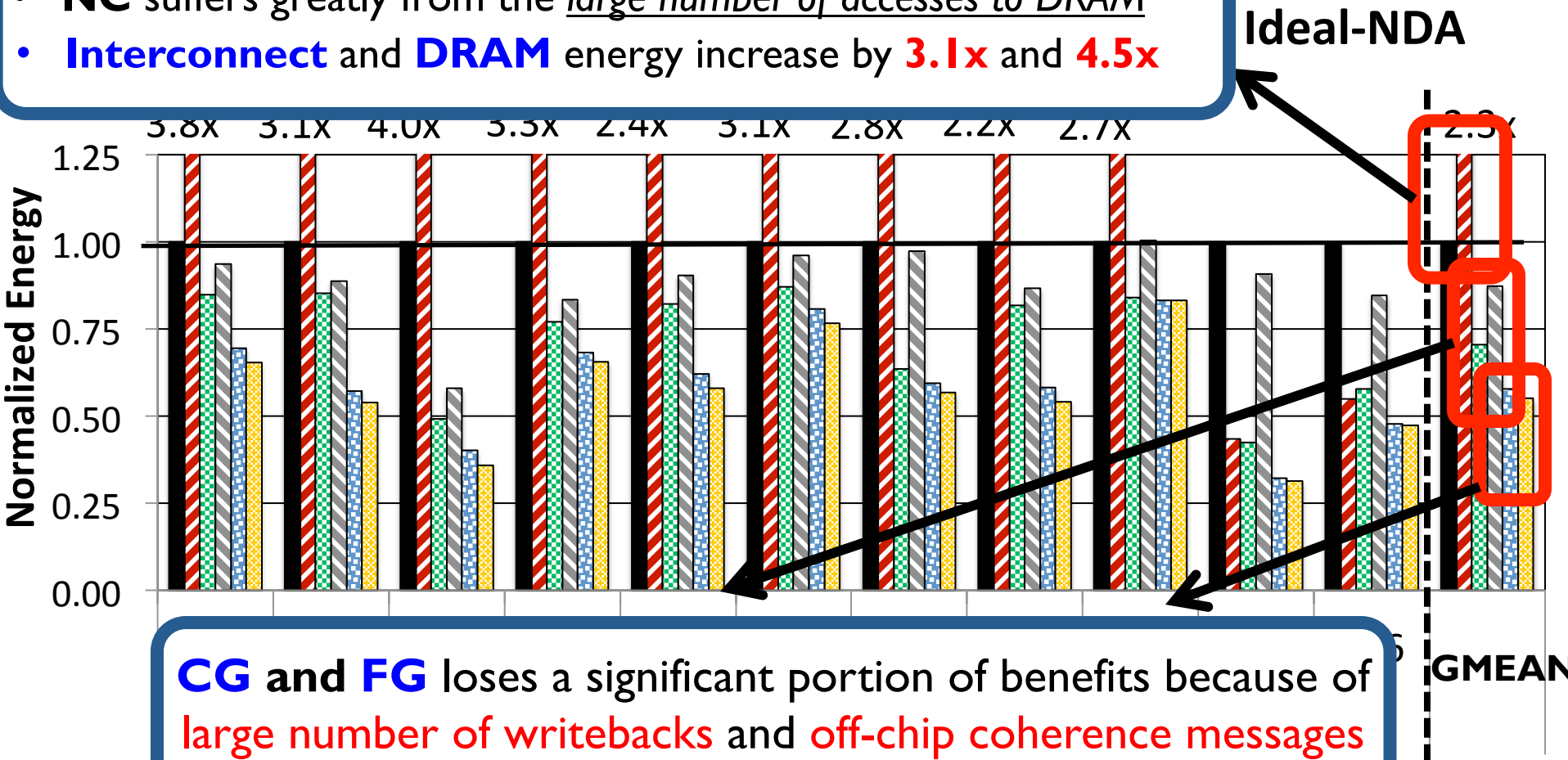
1. **Simplifies NDA programming model**
2. **Allows us to get permissions for only the pieces of data that are actually accessed**



FG eliminates 71.8% of the energy benefits of an ideal NDA mechanism

Memory System Energy

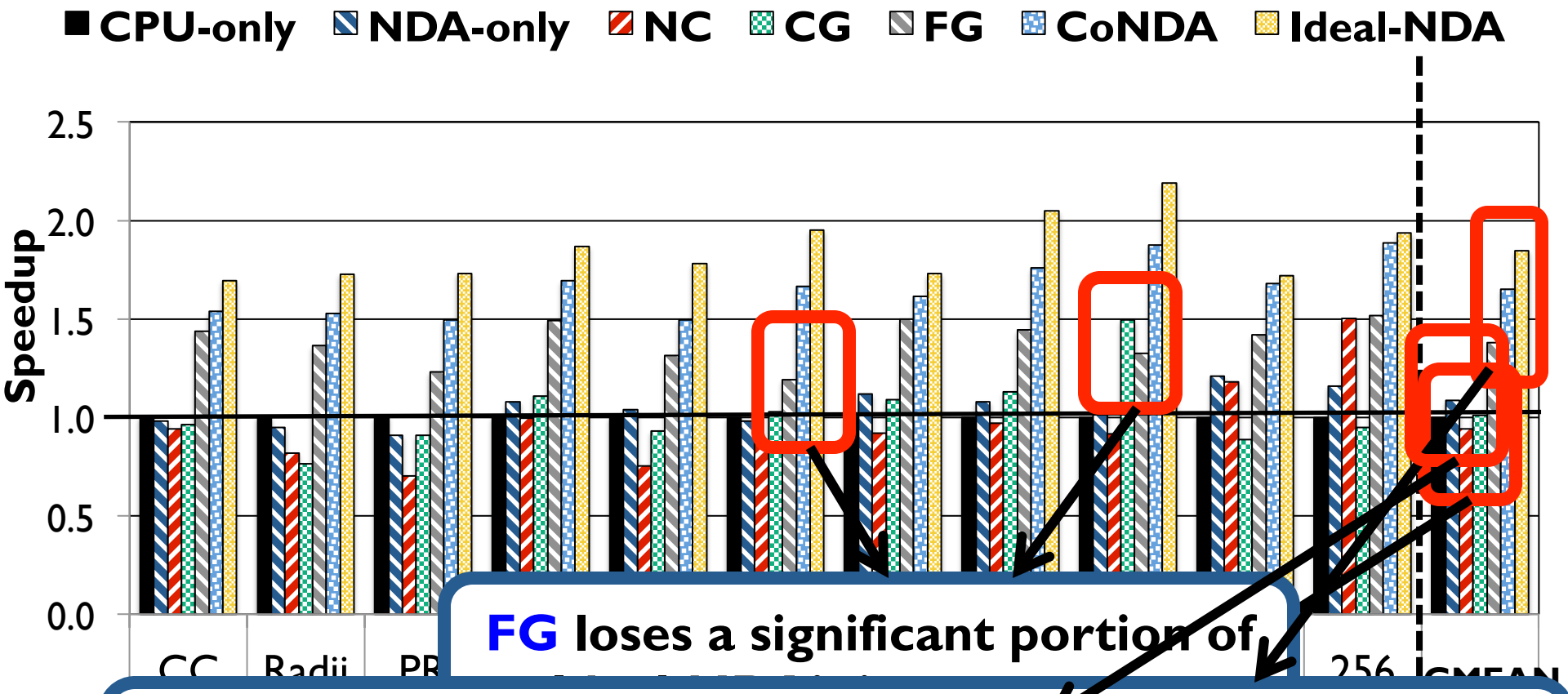
- **NC** suffers greatly from the *large number of accesses to DRAM*
- **Interconnect** and **DRAM** energy increase by **3.1x** and **4.5x**



CG and FG loses a significant portion of benefits because of large number of writebacks and off-chip coherence messages

CoNDA significantly reduces energy consumption

Speedup



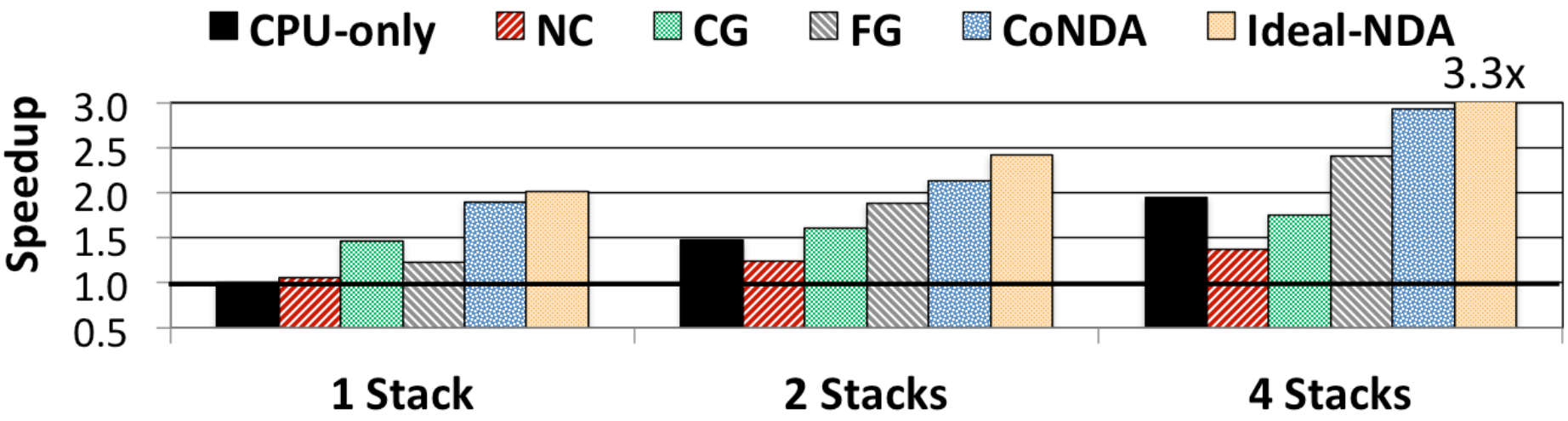
FG loses a significant portion of

CoNDA consi benefits, coming wi mance

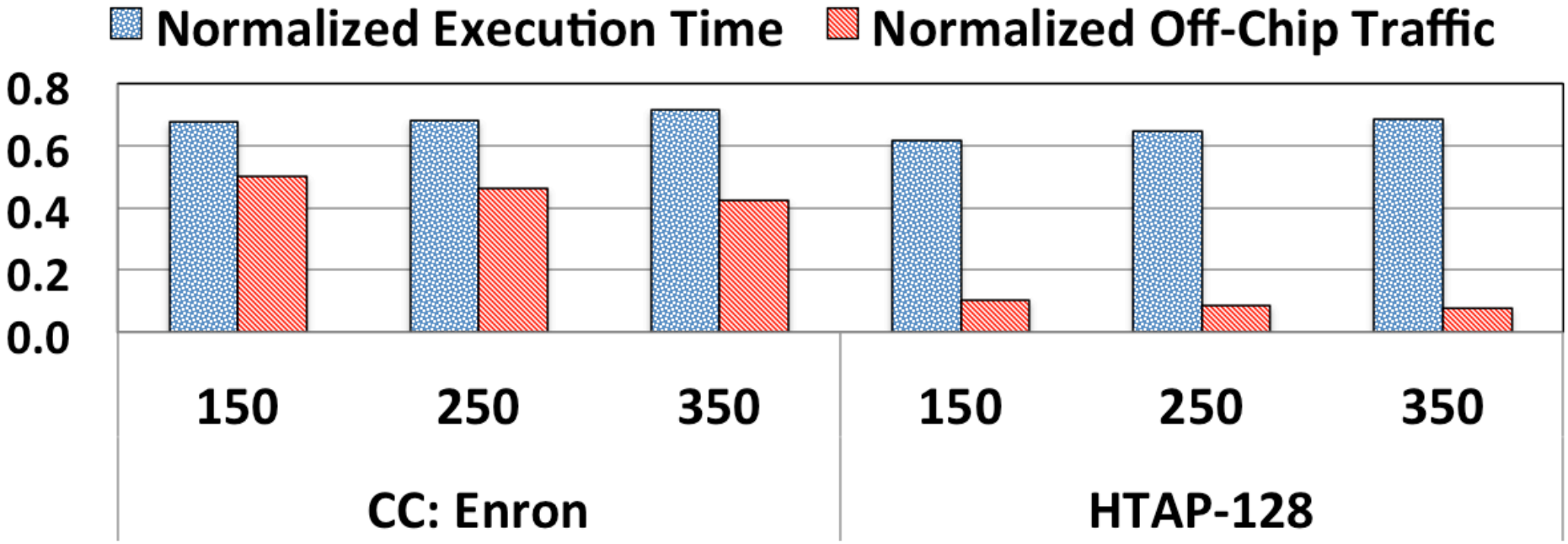
NDA-only eliminates 82.2% of Ideal-NDA's improvement

ion

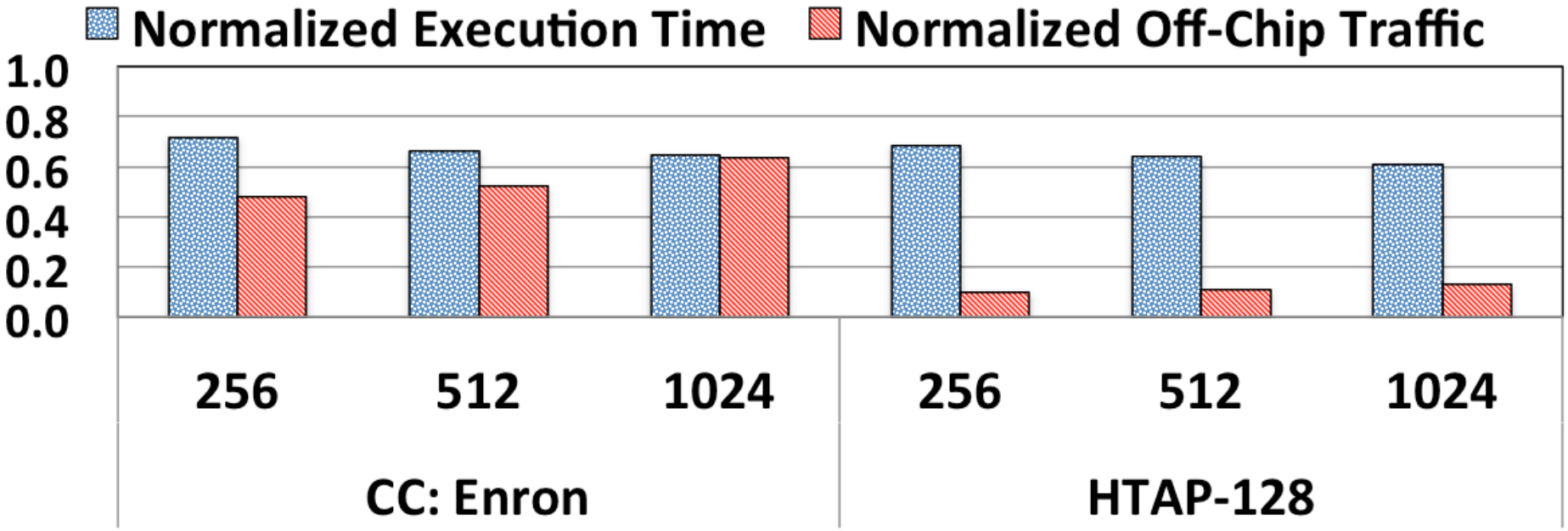
Effect of Multiple Memory Stacks



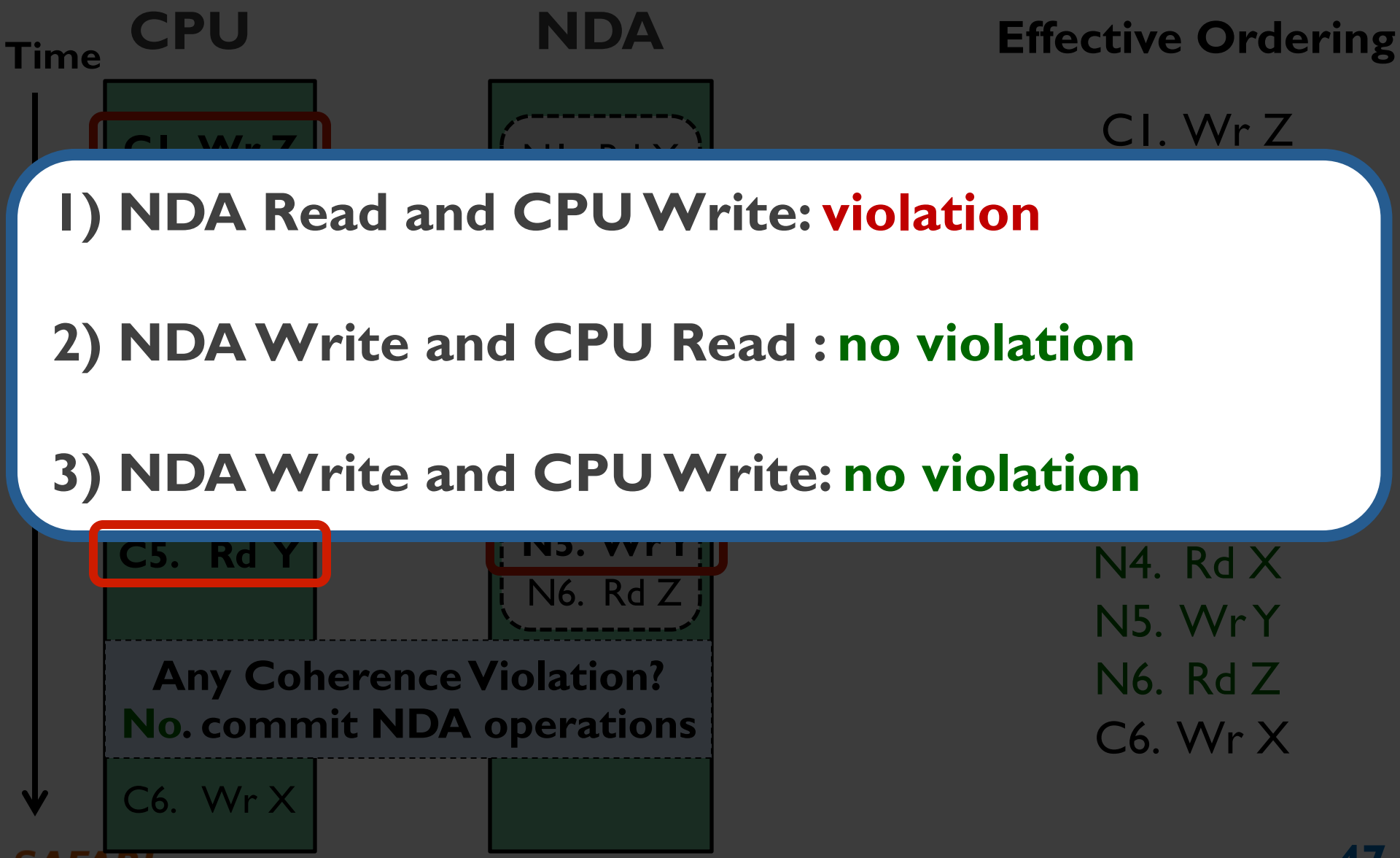
Effect of Optimistic Execution Duration



Effect of Signature Size



Identifying Coherence Violations



1) NDA Read and CPU Write: **violation**

2) NDA Write and CPU Read : **no violation**

3) NDA Write and CPU Write: **no violation**

Optimistic NDA Execution

We leverage two key observations

1. **Majority of coherence**
2. **Enforce coherence with only the necessary data movement**

We propose to use optimistic execution for NDAs

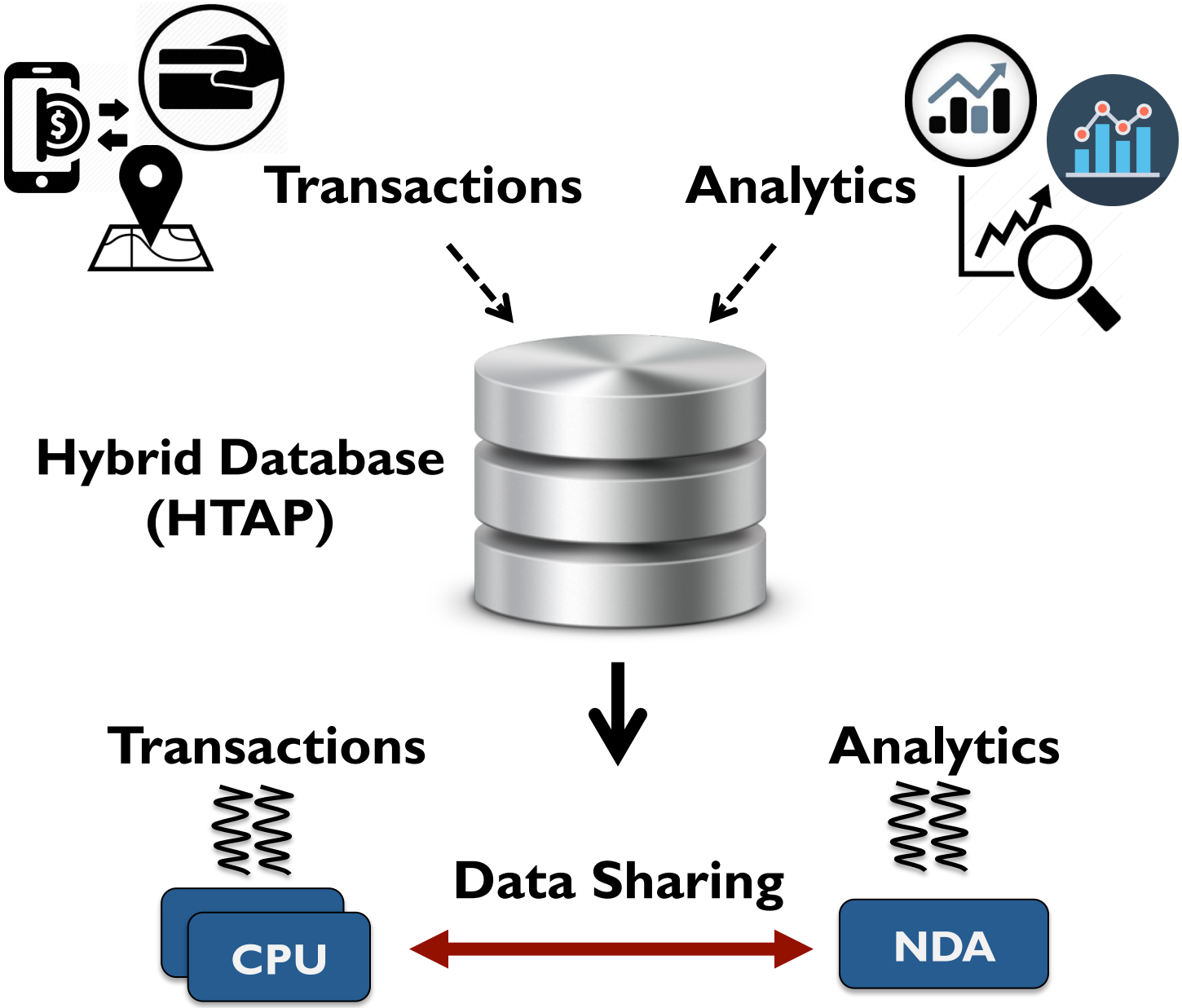
When executing in optimistic mode:

- **An NDA gains insight into its memory accesses without issuing any coherence requests**

When optimistic mode is done:

- **The NDA uses the tracking information to perform necessary coherence requests**

Example: Hybrid Database (HTAP)



Application Analysis Wrap up

1. There is a significant amount of data sharing between CPU threads and NDAs

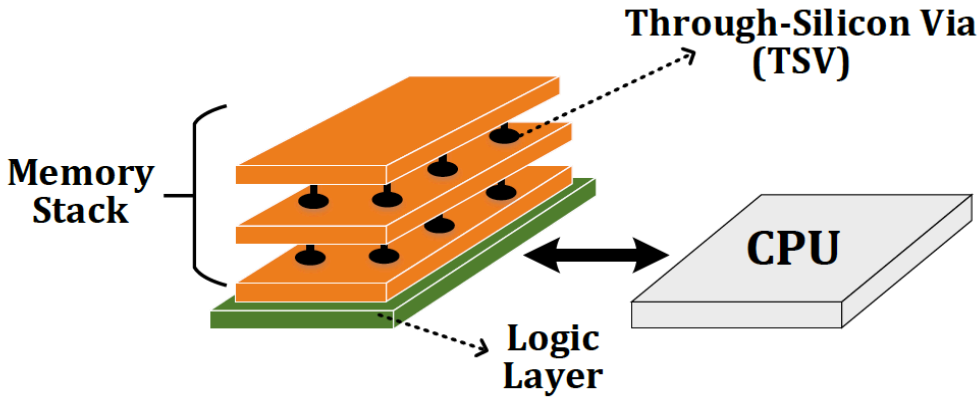
2. CPU threads and NDAs often do not access the same cache lines concurrently

3. CPU threads rarely update the same data that NDAs are actively working on

Background

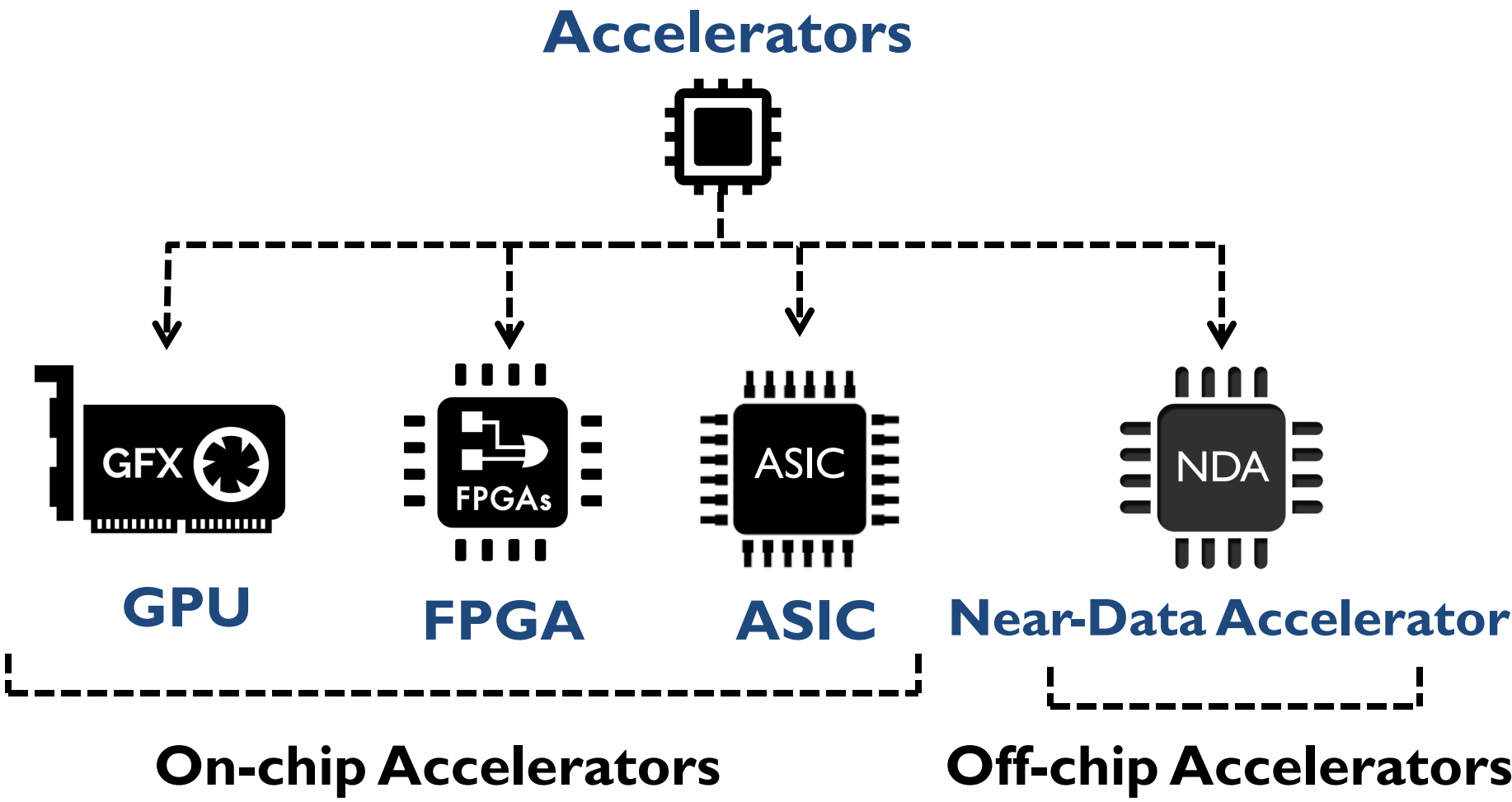
- **Near-Data Processing (NDP)**
  - A potential solution to **reduce data movement**
  - **Idea:** move computation close to data

- **Enabled by recent advances in 3D-stacked memory**



Specialized Accelerators

Specialized accelerators are now everywhere!



Applications

- **Ligra**
  - Lightweight multithreaded graph processing for shared memory system
  - We used three Ligra graph applications
    - PageRank (PR)
    - Radii
    - Connected Components (CC)
  - Input graphs constructed from real-world network datasets:
    - Enron email communication network (36K nodes, 183K edges)
    - arXiV General Relativity (5K nodes, 14K edges)
    - peer-to- peer Gnutella25 (22K nodes, 54K edges).
- **IMDB**
  - In-house prototype of an in-memory database (IMDB)
  - Capable of running both **transactional** queries and **analytical** queries on the **same** database tables (**HTAP workload**)
  - 32K transactions, 128/256 analytical queries

Optimistic NDA Execution

We leverage two key observations:

1. Eliminate much of unnecessary coherence traffic by having insight into memory accesses
2. CPU threads and NDA kernels typically do not concurrently access the same cache lines

We propose to use optimistic execution for NDAs

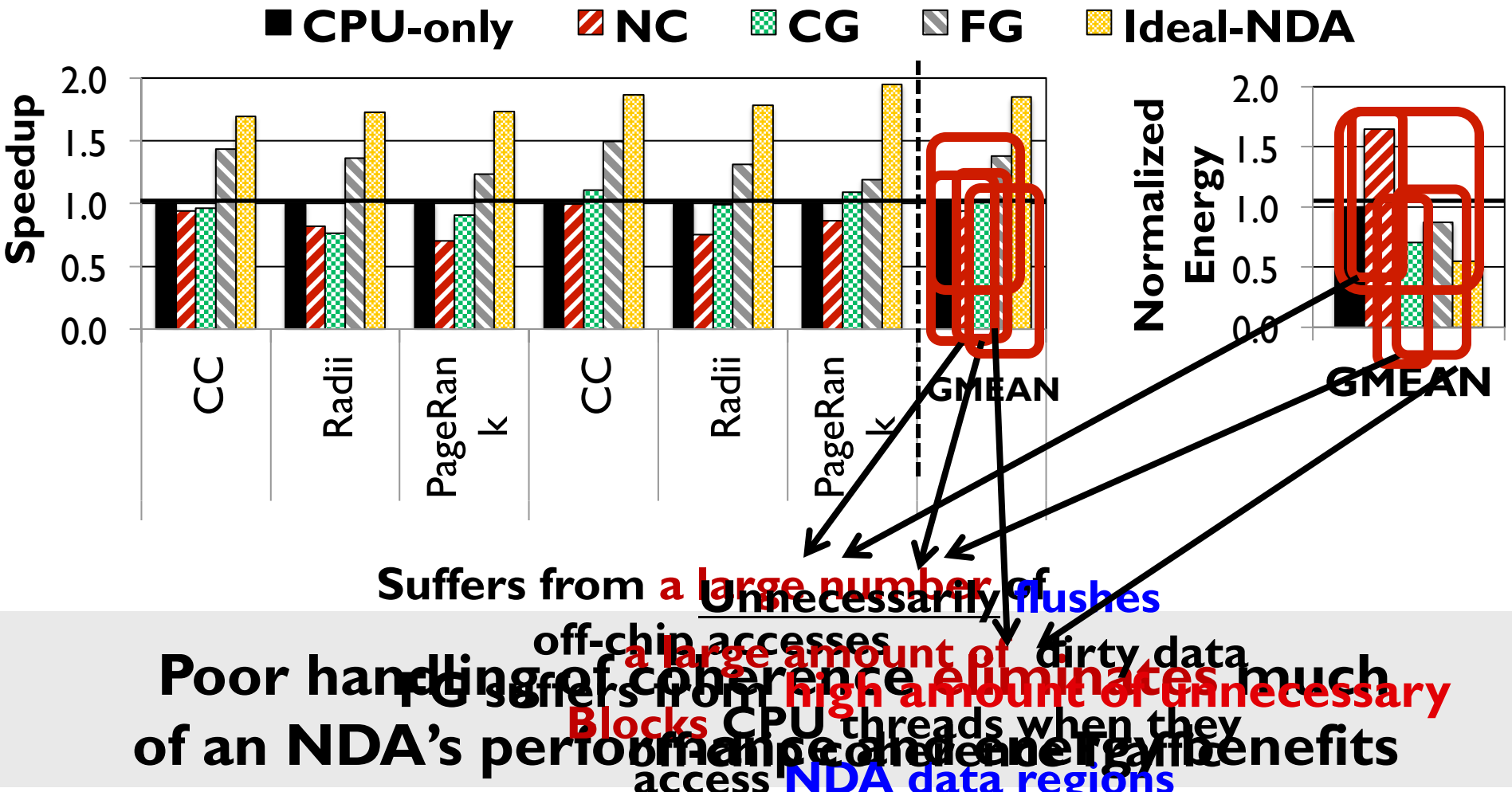
NDA executes the kernel:

1. Assumes it has coherence permission
2. Gains insights into memory accesses

When execution is done:

Performs only the necessary coherence requests

Analysis of Existing Coherence Mechanisms



Poor handling of coherence can significantly reduce the performance of an NDA's off-chip memory bandwidth and energy benefits