

Rethinking Support for Region Conflict Exceptions

Swarnendu Biswas, Rui Zhang, Michael D. Bond, and Brandon Lucia

IPDPS 2019

C++ Program with Data Race

```
X* x = NULL;  
bool done= false;
```

Thread T1

```
x = new X();  
done = true;
```

Thread T2

```
if (done) {  
    x->func();  
}
```



Catch-Fire Semantics in C++

C++ treats data races as errors

```
X* x = NULL;  
bool done= false;
```

Thread T1

```
x = new X();  
done = true;
```

Thread T2

```
if (done) {  
    x->func();  
}
```



Catch-Fire Semantics in C++



ANYTHING



BAD CAN



HAPPEN!

```
X *x = NULL;  
bool done= false;
```

Thread T1

```
x = new X();  
done = true;
```

Thread T2

```
while (!done) {}  
x->func();
```

```
X *x = NULL;  
bool done= false;
```

Thread T1

```
x = new X();  
done = true;
```

Thread T2

```
while (!done) {}  
x->func();
```



Thread T1

```
x = new X();  
  
done = true;
```

Thread T2

```
temp = done;  
while (!temp) {}
```

LICM

Infinite
loop

```
X *x = NULL;  
bool done= false;
```

Thread T1

```
x = new X();  
done = true;
```

Thread T2

```
while (!done) {}  
x->func();
```



Thread T1

```
done = true;
```

```
x = new X();
```

Thread T2

```
while (!done) {}  
x->func();
```

NPE



KILLED BY A MACHINE: THE THERAC-25

by: Adve



BUSINESS SOFTWARE
business

BUSINESS
READY



Nasdaq's Facebook Glitch Came From Race Conditions

Joab Jackson
@Joab_Jackson

May 21, 2012 12:30 PM |

The Nasdaq computer system that delayed trade notices of the Facebook IPO on Friday was plagued by race conditions, the stock exchange announced Monday. As a result of



research highlights

Technical Perspective Data Races are Evil with No Exceptions

By Sarita Adve

EXPLOITING PARALLELISM HAS become the primary means to higher performance. racy code. Java's safety requirements preclude the use of "undefined" beh

How to miscompile programs with "benign" data races

Hans-J. Boehm
HP Laboratories

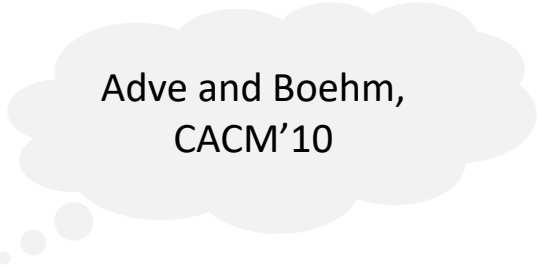
SUBSCRIBE

Enter Email Address

SUBSCRIBE

The Therac-25 was not a device anyone was happy to see. It was a radiation therapy machine. In layman's terms it was a "cancer zapper"; a linear accelerator with a human as its target. Using X-rays or a beam of electrons,

Need for Stronger Semantics for Programs with Data Races



Adve and Boehm,
CACM'10

“The **inability to define reasonable semantics for programs with data races** is not just a theoretical shortcoming, but a **fundamental hole** in the foundation of our languages and systems.”

“We call upon software and hardware communities to develop languages and systems that enforce data-race-freedom, ...”

What Do We Mean by Strong Semantics?

End-to-end guarantees even for programs with data races

Outline




Impact of Data Races on Language Models



Strong Semantics with Region Conflict Exceptions



Providing Region Conflict Exceptions



ARC: Practical Architecture Support for Region Conflict Exceptions



Comparison of ARC with Related Approaches

Strong Execution Semantics with Region Conflict Exceptions

C++ Program with Data Race

```
X* x = NULL;  
bool done= false;
```

Thread T1

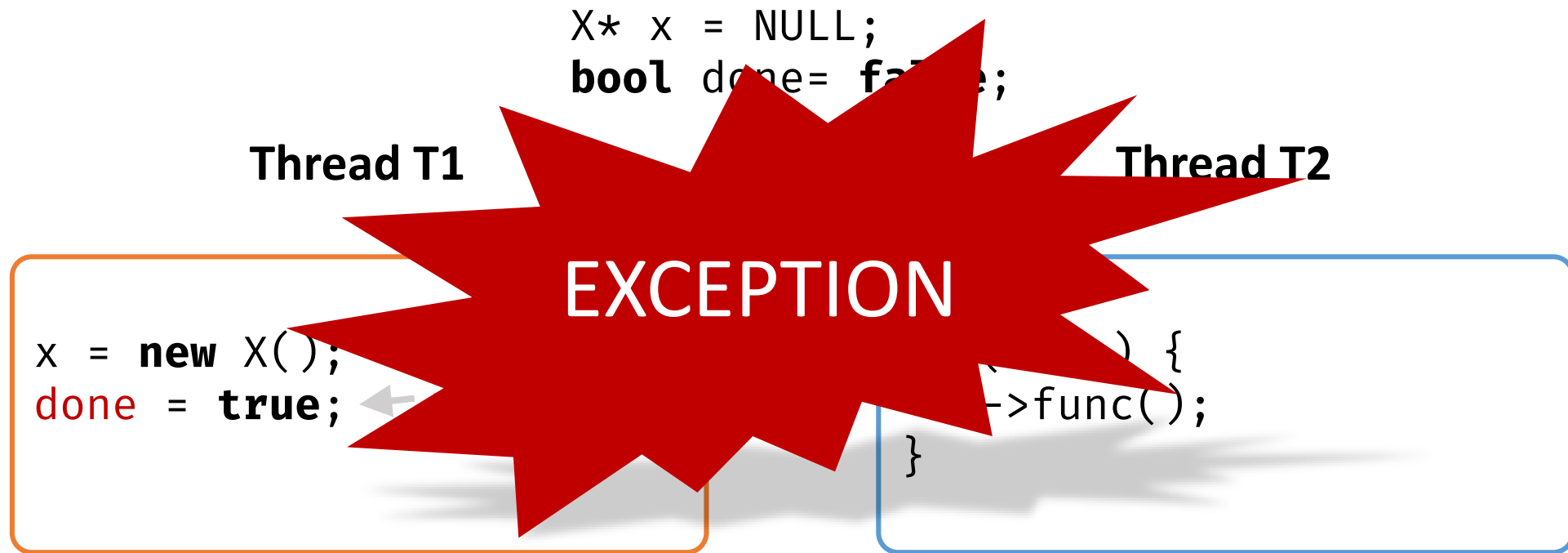
```
x = new X();  
done = true;
```

Thread T2

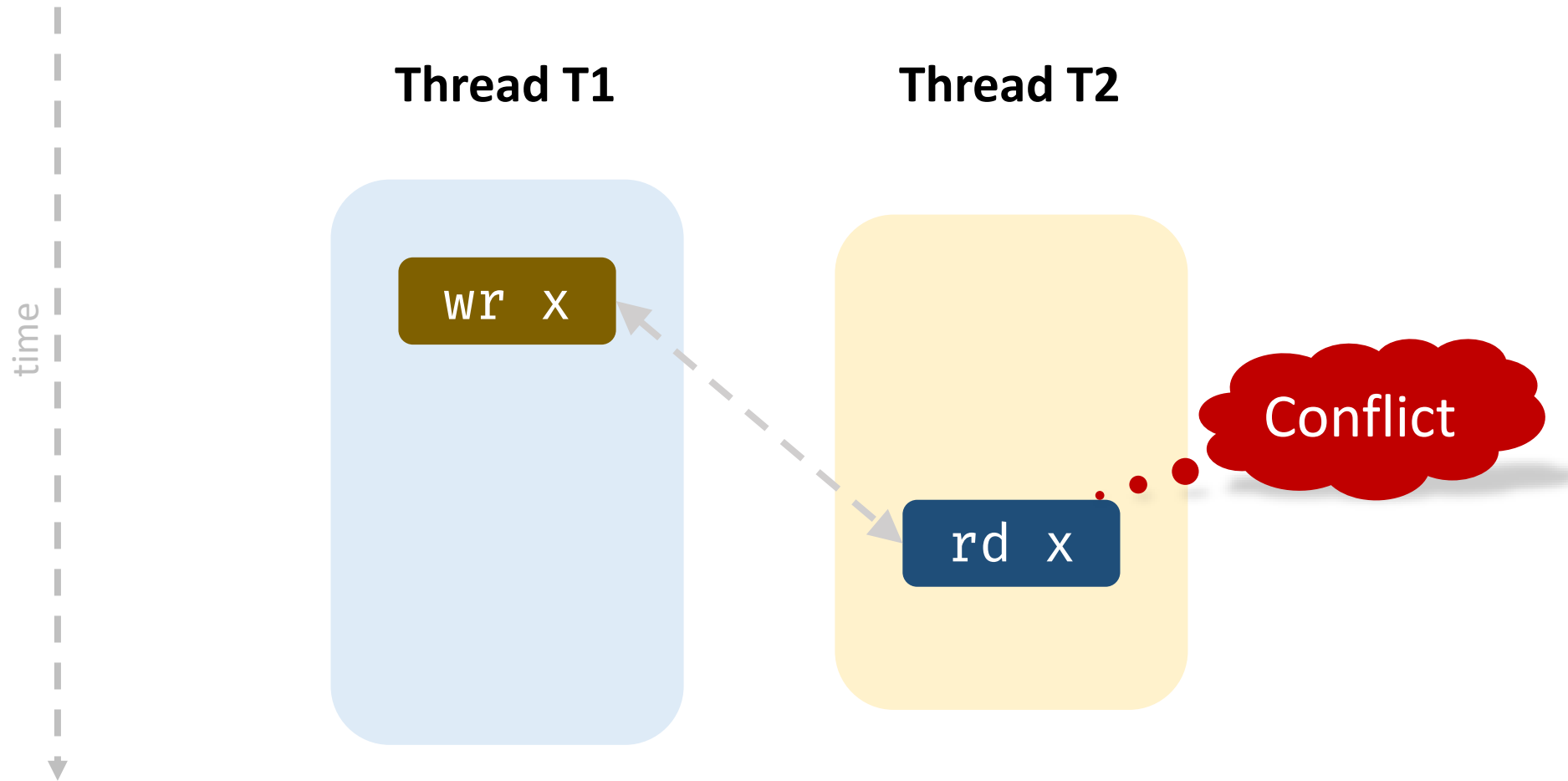
```
if (done) {  
    x->func();  
}
```



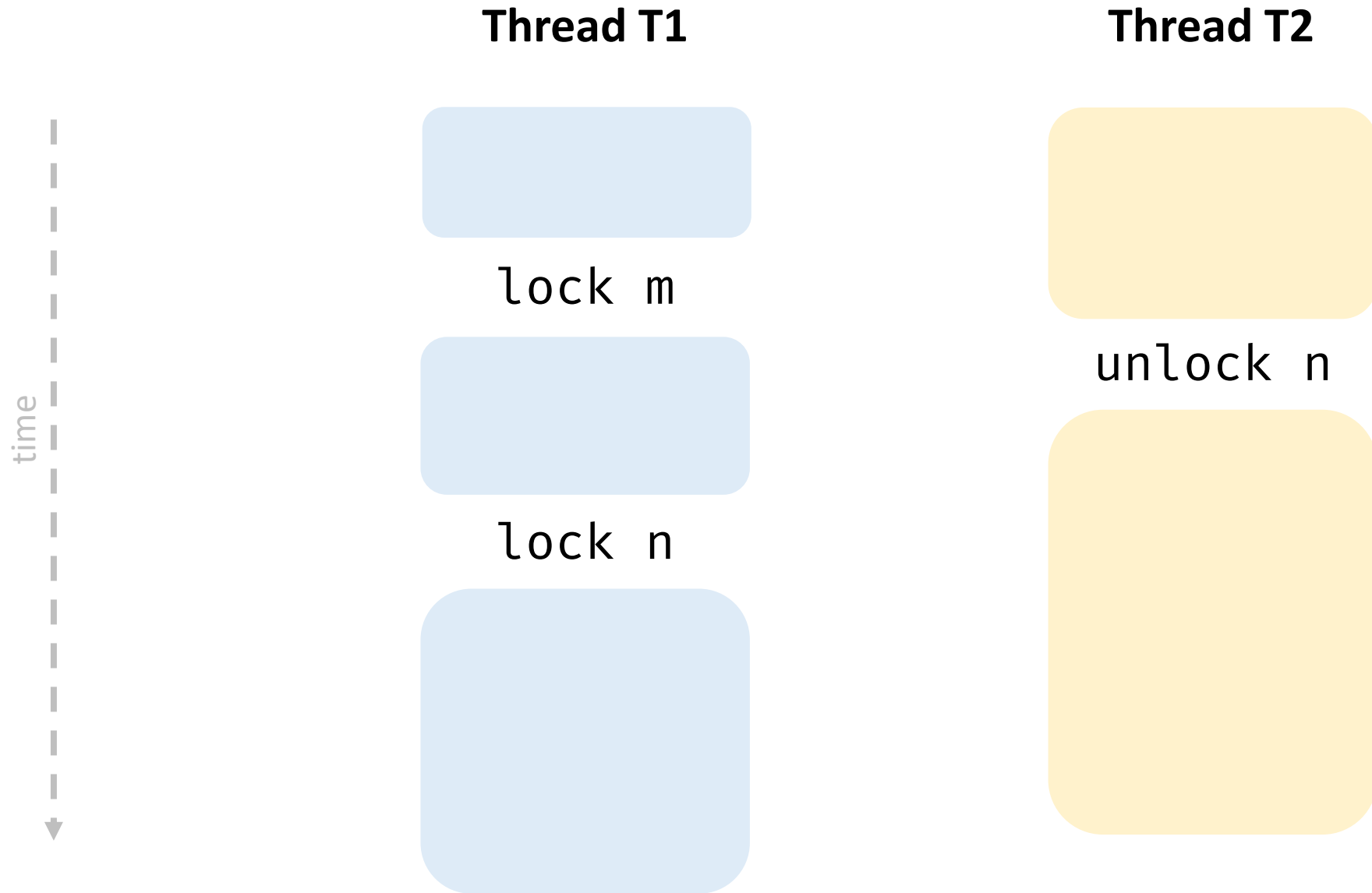
Data Race Exceptions



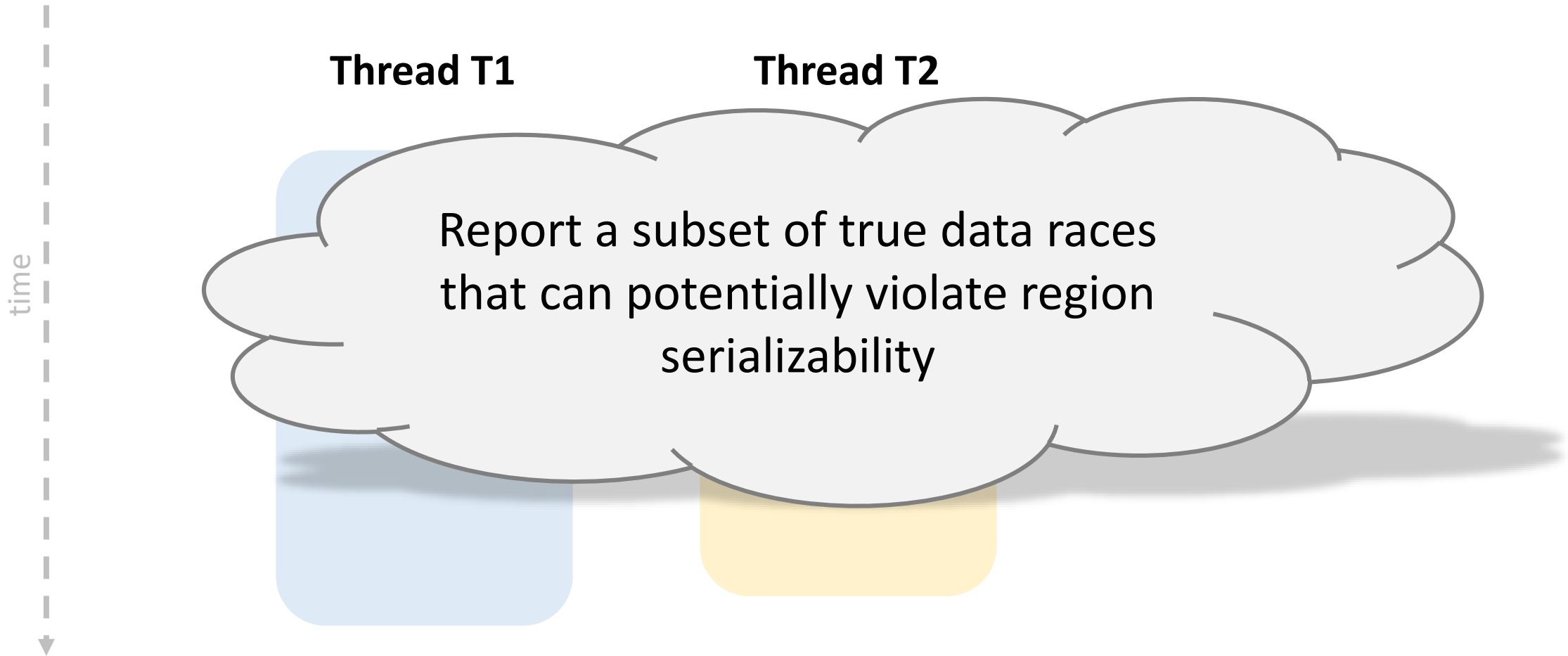
Region Conflicts



Synchronization Free Regions (SFRs)

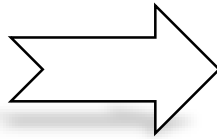


Region Conflicts



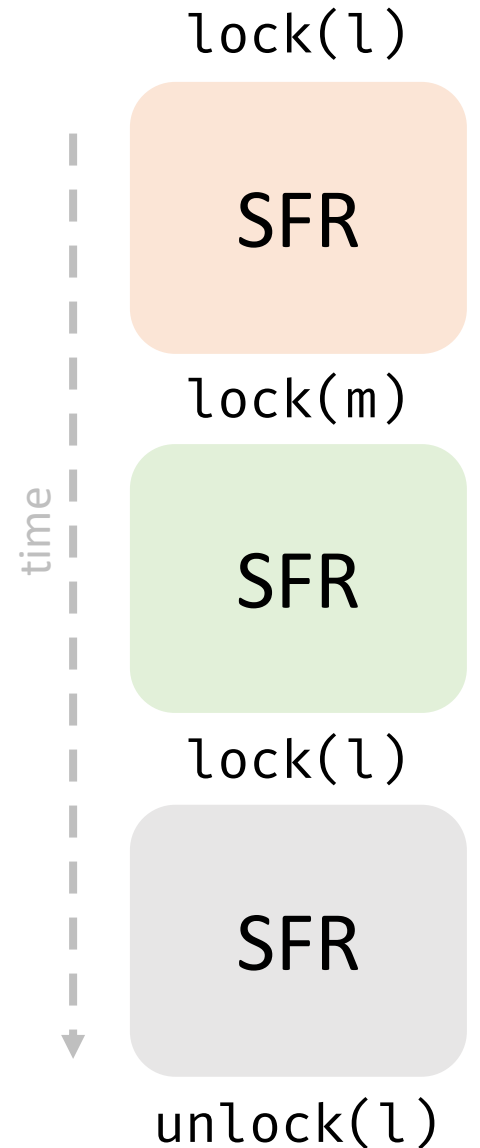
Semantics with Region Conflict Exceptions

Conflict-free
execution



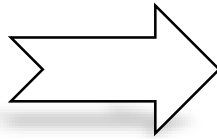
SFR
serializability

Synchronization-free regions (SFRs) execute
atomically



Semantics with Region Conflict Exceptions

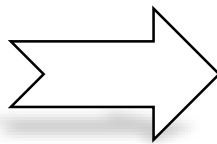
Conflict-free
execution



SFR
serializability

Region
conflict

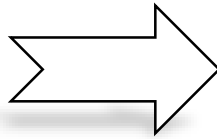
true data
races



Potential
serializability
violation

Semantics with Region Conflict Exceptions

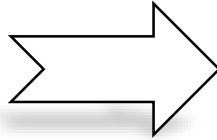
Conflict-free
execution



SFR
serializability

Region
conflict

true data
races



Exception

Providing Region Conflict Exceptions

Providing Region Conflict Exceptions

Valor: Efficient, Software-Only Region Conflict Exceptions *

Conflict Exceptions: Simplifying Concurrent Language Semantics with Precise Hardware Exceptions for Data-Races

Biswas et al. Valor: Efficient, Software-Only Region Conflict Exceptions. OOPSLA 2015.

Lucia et al. Conflict Exceptions: Simplifying Concurrent Language Semantics With Precise Hardware Exceptions for Data-Races. ISCA 2010.

Drawbacks with Conflict Exceptions

Builds on top of M(O)ESI-style cache coherence

- Introduces hardware on top existing structures
- Increases complexity

Inter-core communication at region boundaries

- Metadata in private cache lines are forwarded to other cores
- Increases on-chip interconnect bandwidth requirement

Private line evictions communicate with memory

- Relies on in-memory backup for evicted metadata
- Increases off-chip memory bandwidth requirement

ARC: Practical Architecture Support for Region Conflict Exceptions

Design Overview

Architectural Modifications

Example Executions with ARC

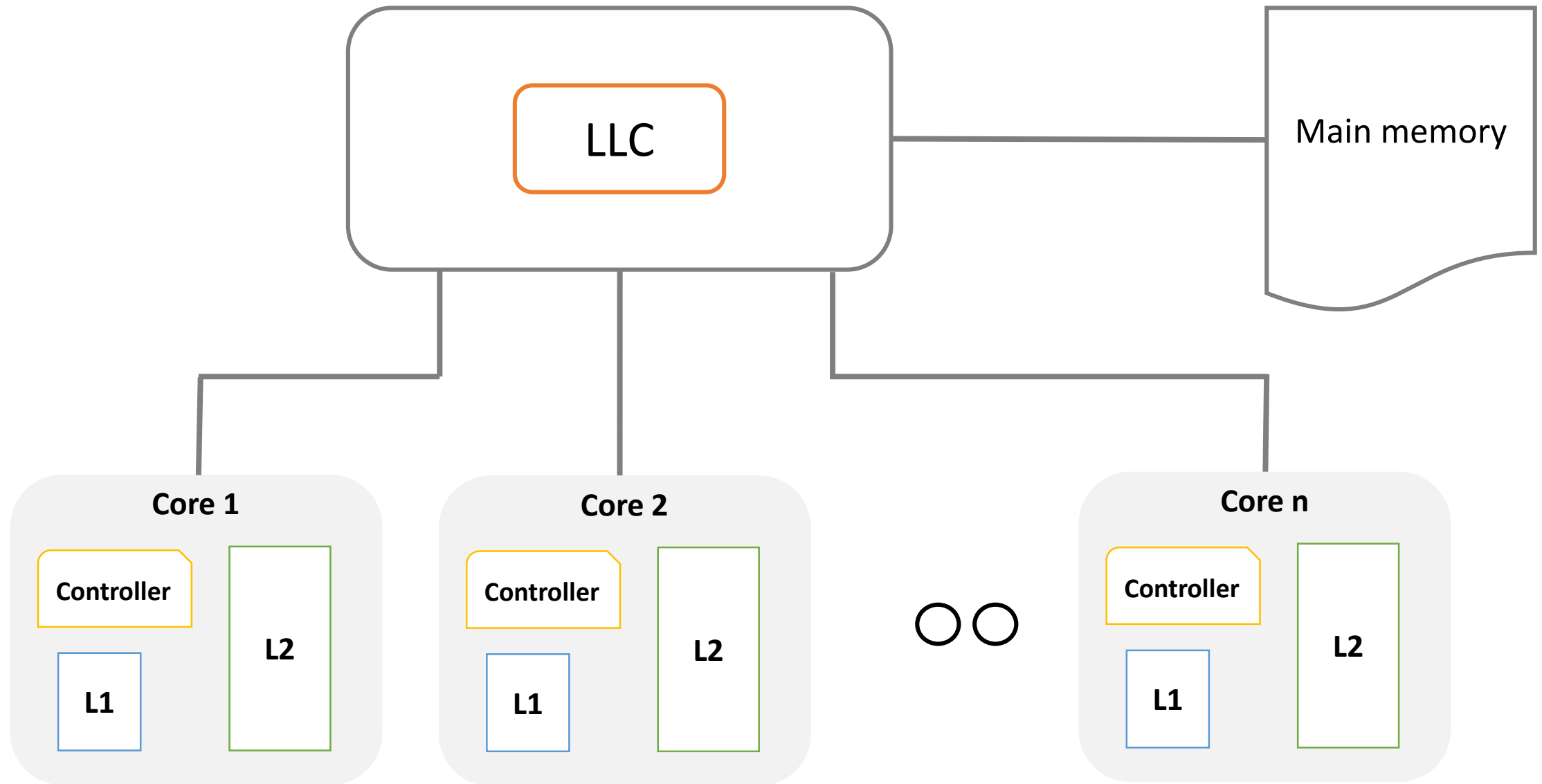
ARC: Practical Architecture Support for Region Conflict Exceptions

- Design Overview

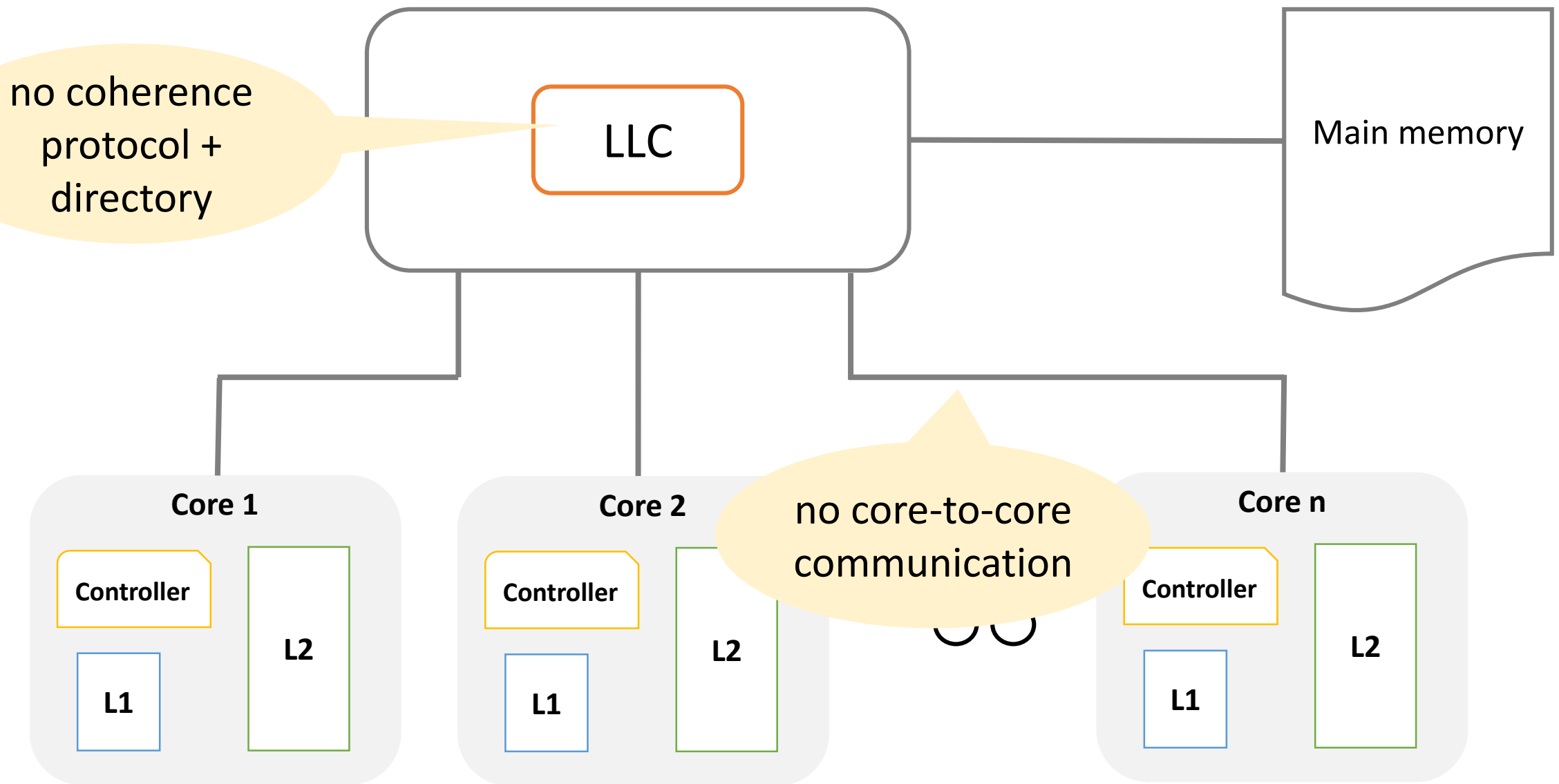
Architectural Modifications

Example Executions with ARC

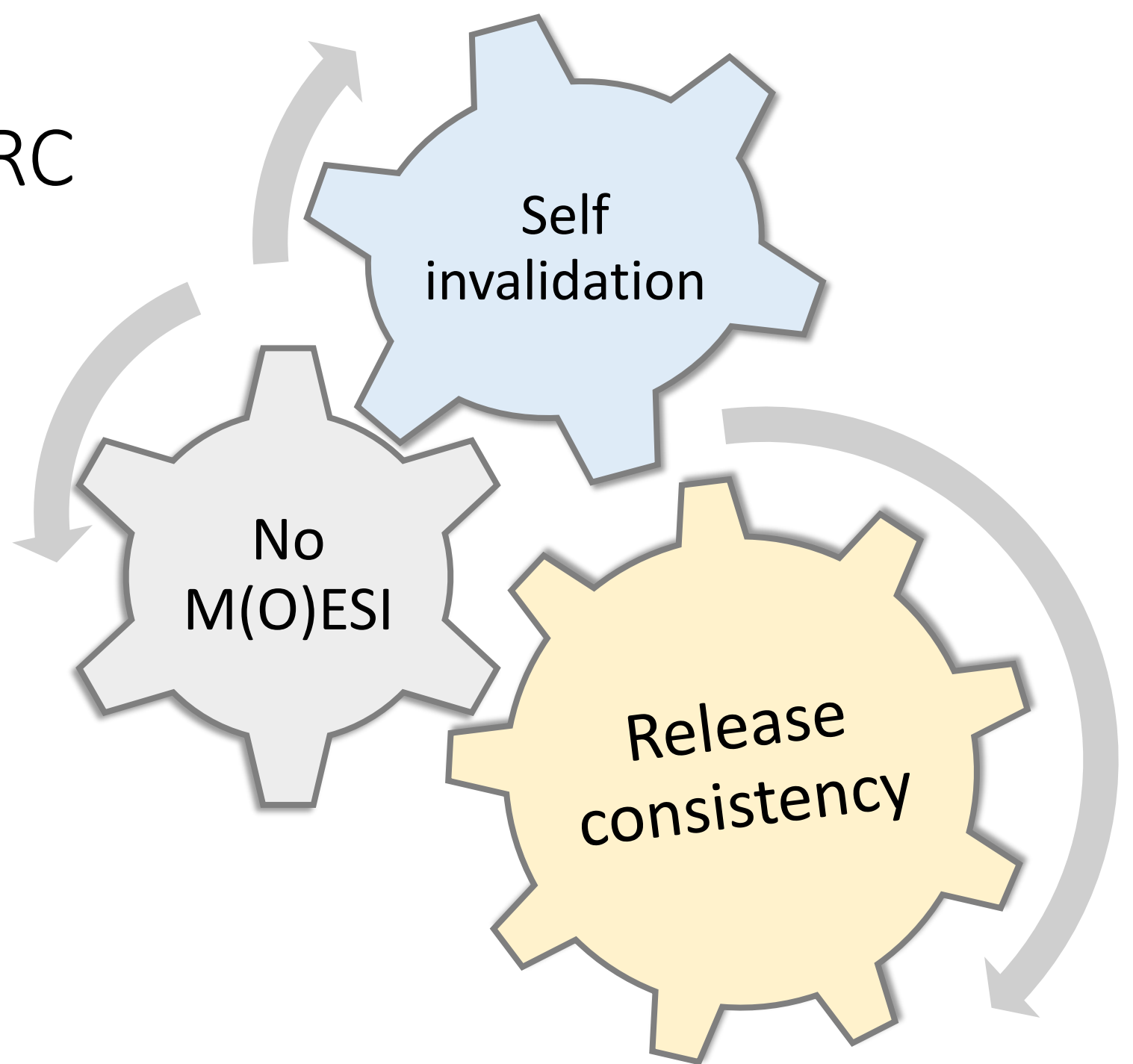
Baseline Architecture in ARC



Baseline Architecture in ARC



Key Insights in ARC



Release Consistency

core's private cache waits to write back its dirty data until a synchronization release operation

time
↓

```
X = new Object();  
done = true;  
unlock(m);
```

```
lock(m);  
while (!done) {}  
X.compute();
```

Self-Invalidation

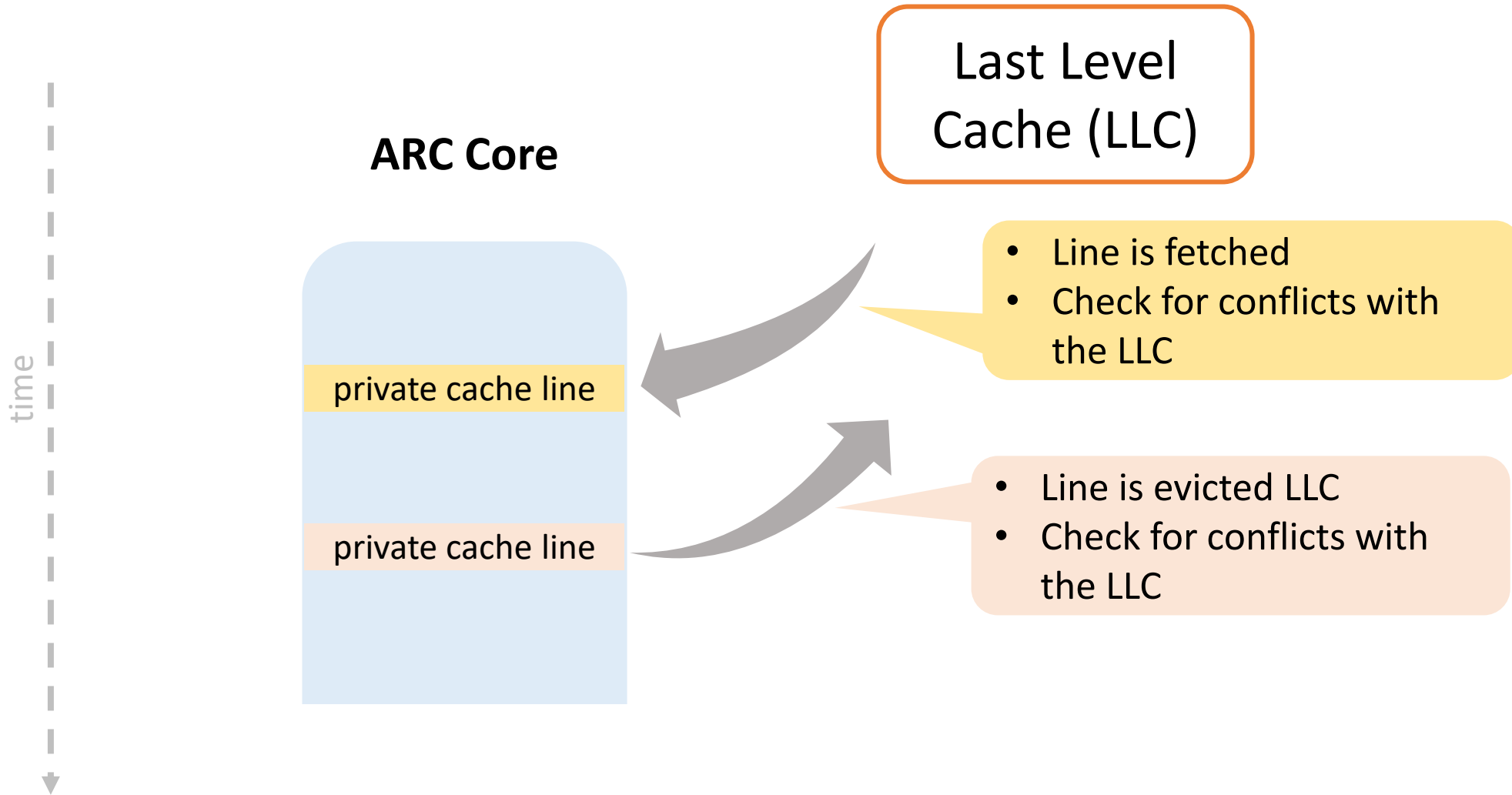
core invalidates private cache lines that may be **out-of-date** at synchronization acquire operations

ARC: Our Proposed Technique for Region Conflict Detection

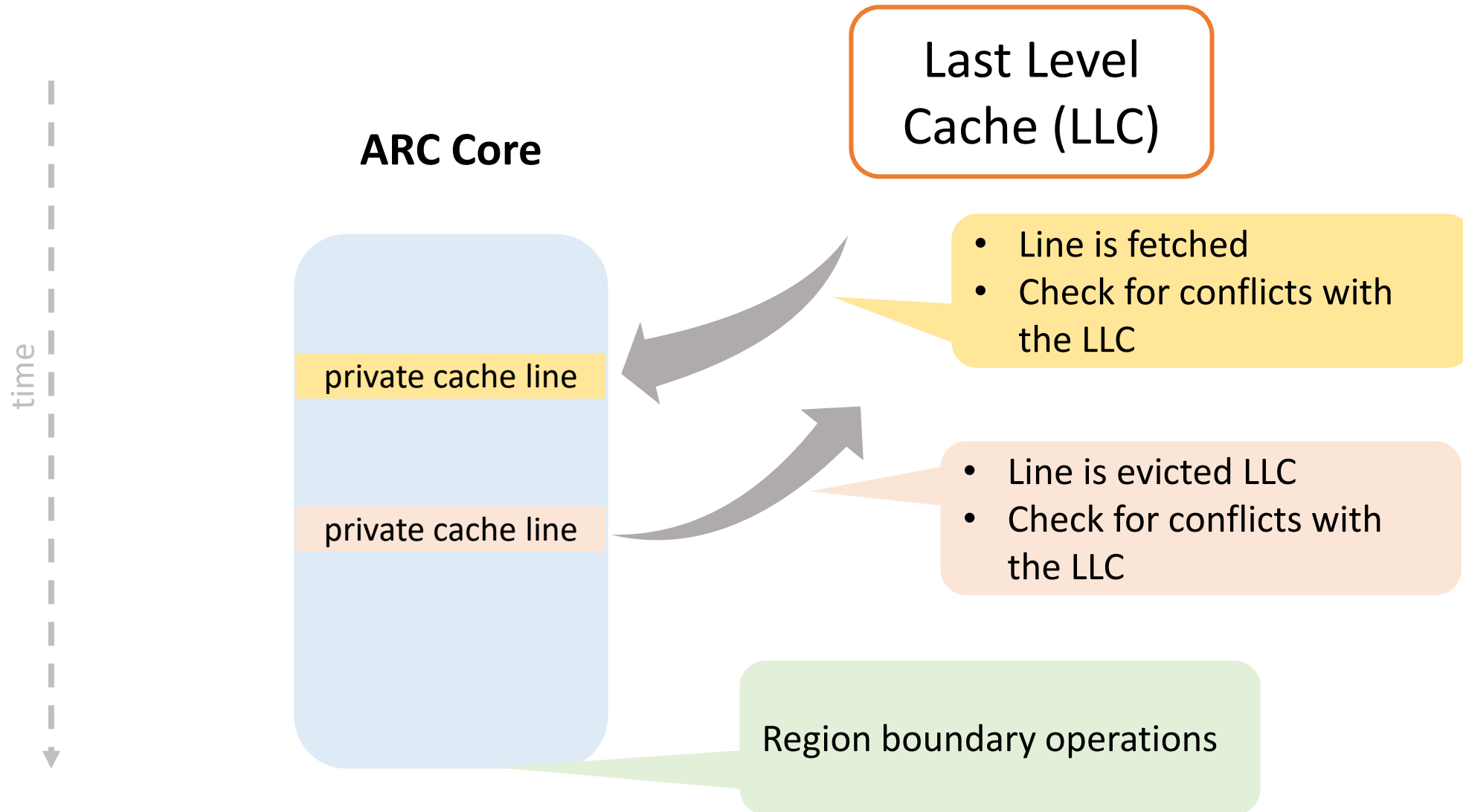
Explore whether synergistic use of release consistency and self-invalidation can be competitive

Provide consistency and coherence at SFR boundaries and on private cache line evictions

Understanding how ARC Works



Understanding how ARC Works



Serializability of Regions

A region appears serializable
if:

☐ There were no conflicts

☐ Writes appear atomic

☐ Values read are consistent

Region Boundary Operations in ARC

A region appears serializable if:

provide
coherence

There were no conflicts

Writes appear atomic

Values read are consistent

At a region boundary, an ARC core executes:

Pre-commit – Write back dirty lines to the LLC

Region Boundary Operations in ARC

A region appears serializable if:

■ There were no conflicts

■ Writes appear to be sequential

■ Values read are consistent

ensure consistency

At a region boundary, an ARC core executes:

Pre-commit – Write back dirty lines to the LLC

Read validation – Validate reads using version and value validation



Region Boundary Operations in ARC

A region appears serializable if:

- There were no conflicts
- Writes appear atomic
- Values read are consistent

provide
coherence

At a region boundary, an ARC core executes:

Pre-commit – Write back dirty lines to the LLC

Read validation – Validate reads using version and value validation

Post-commit – Clear per-core metadata, self-invalidate private lines

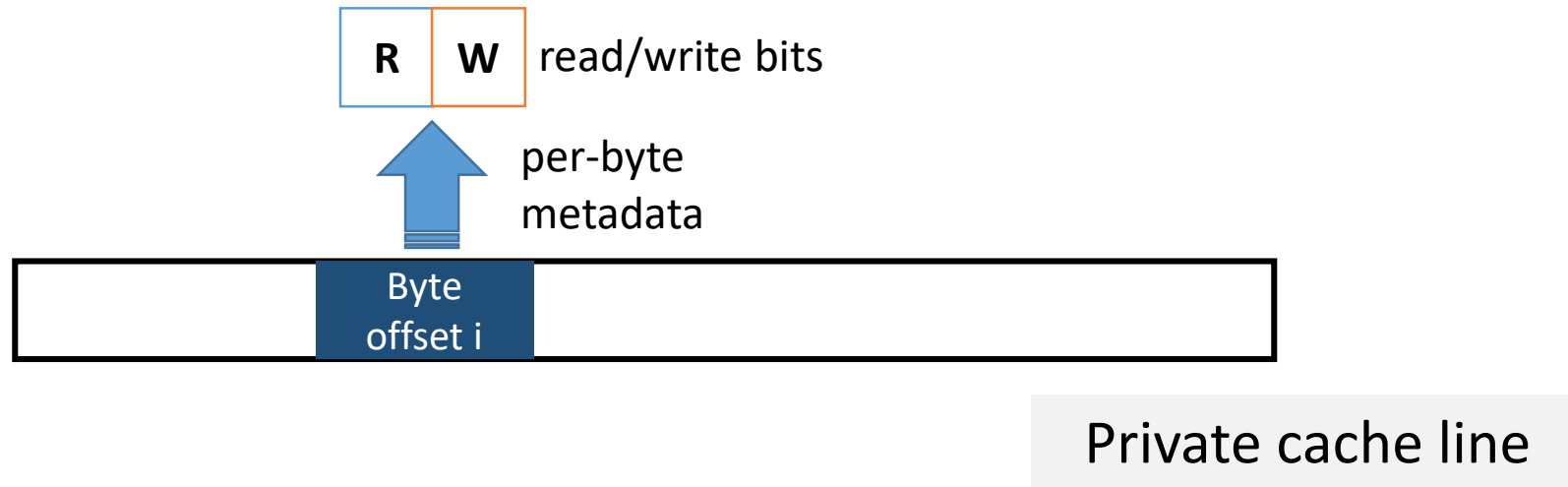
ARC: Practical Architecture Support for Region Conflict Exceptions

Design Overview

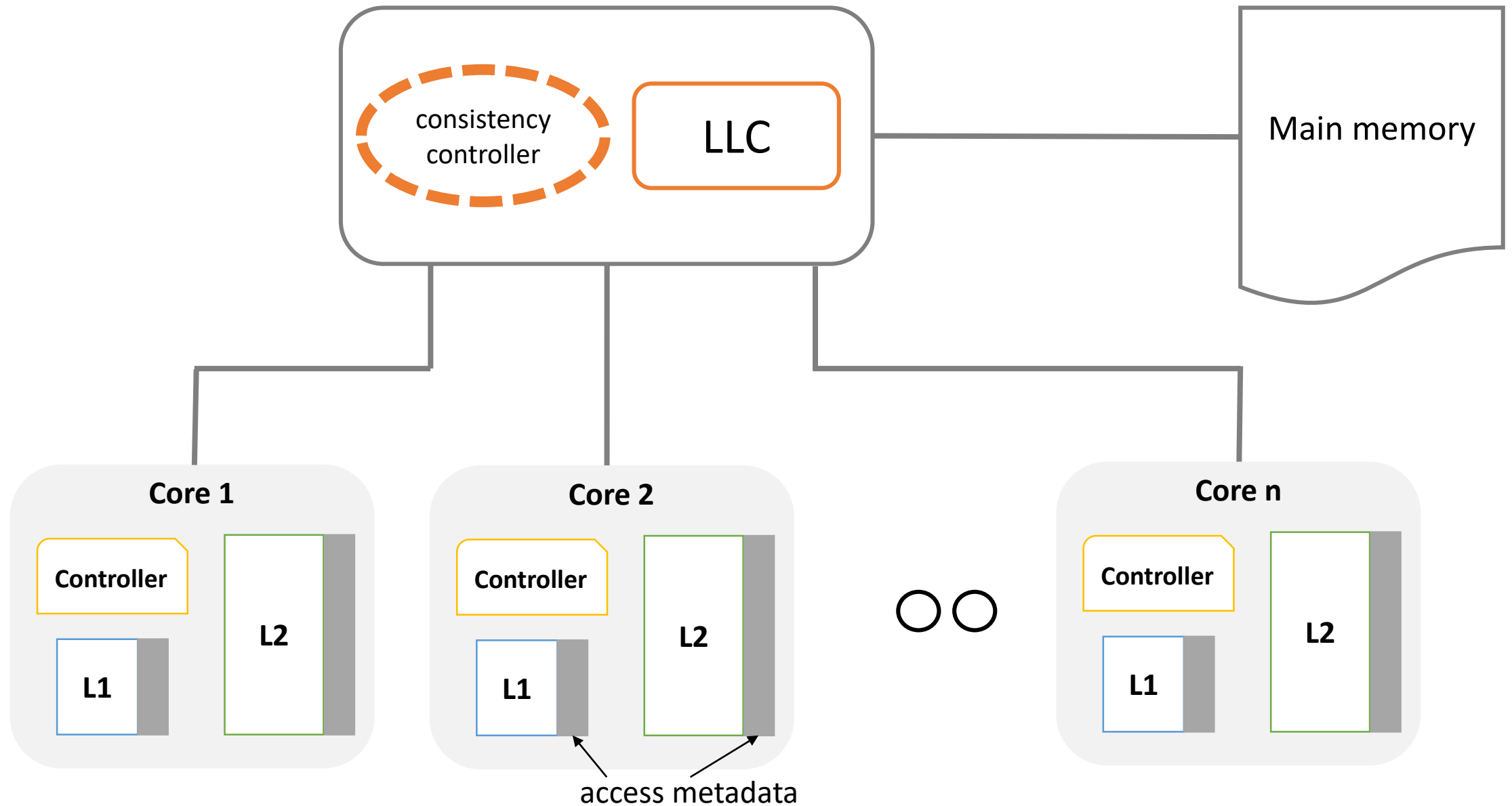
- Architectural Modifications

Example Executions with ARC

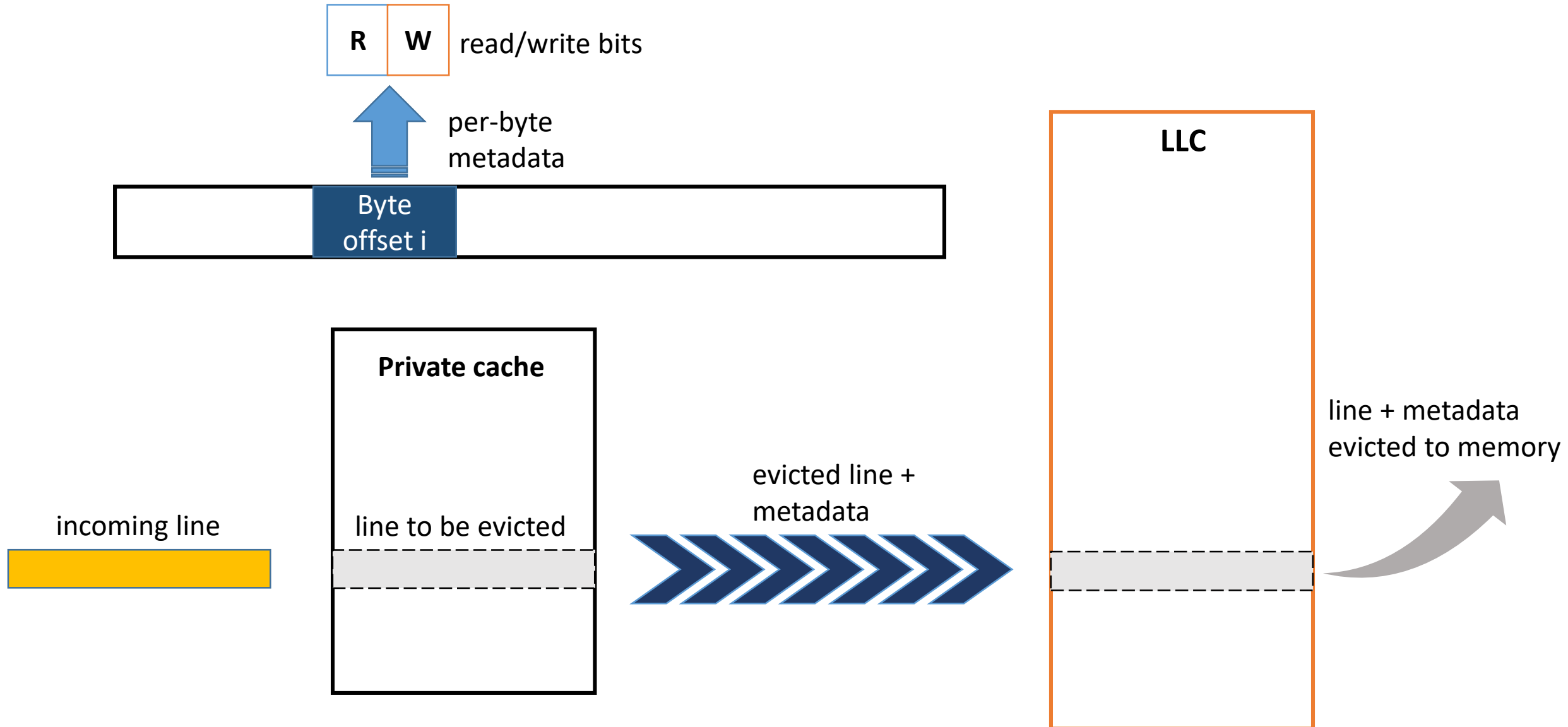
Detecting Sound and Precise Conflicts



Modifications Introduced by ARC



Metadata Management



Access Information Memory (AIM)

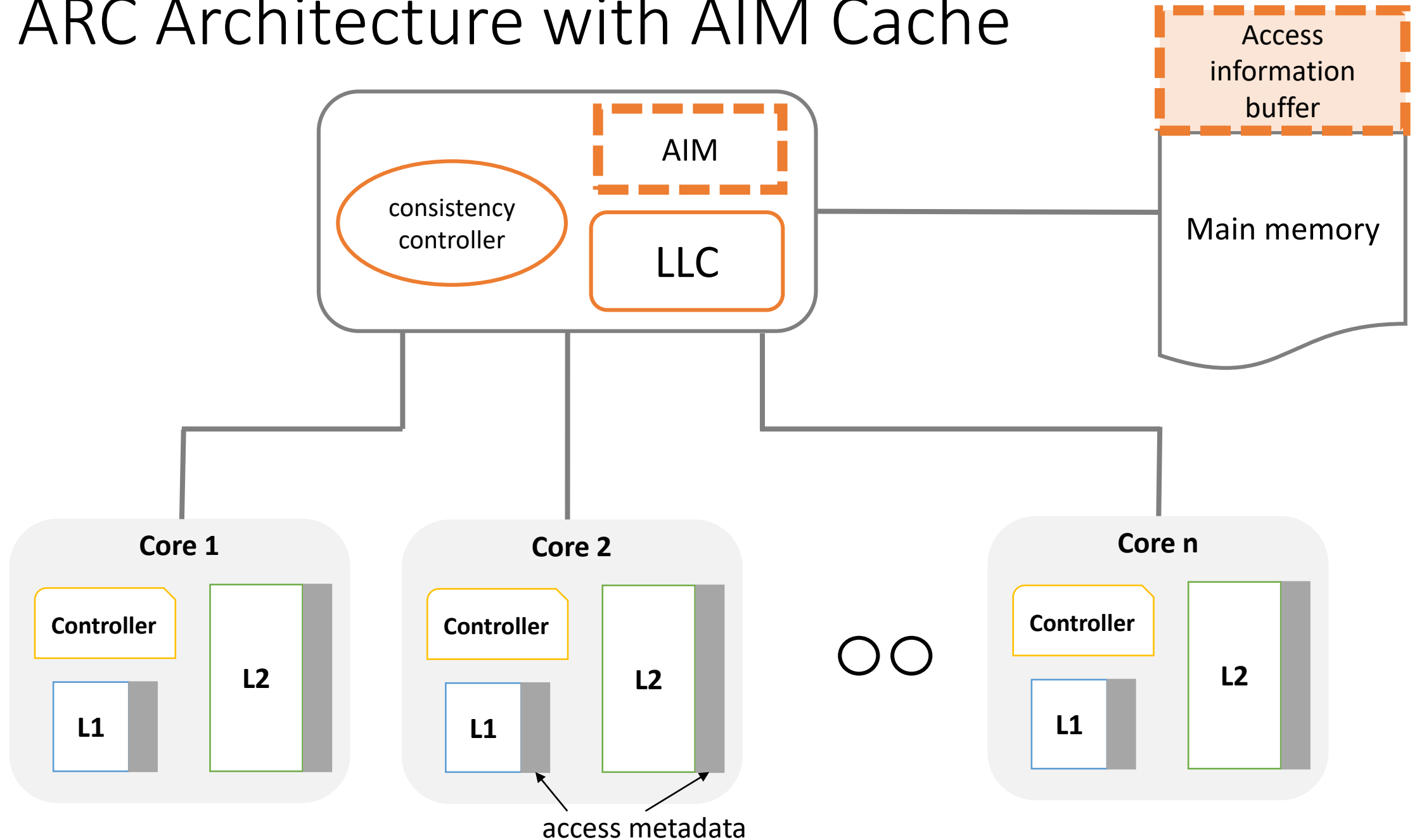
AIM is a dedicated metadata cache adjacent to the LLC

AIM lines in ARC can be large

- 100 bytes for 8 cores, 178 bytes for 16 cores, and 308 bytes for 32 cores
- Impractical to have large AIM cache structures

ARC assumes a realistic AIM design with 32K entries

ARC Architecture with AIM Cache



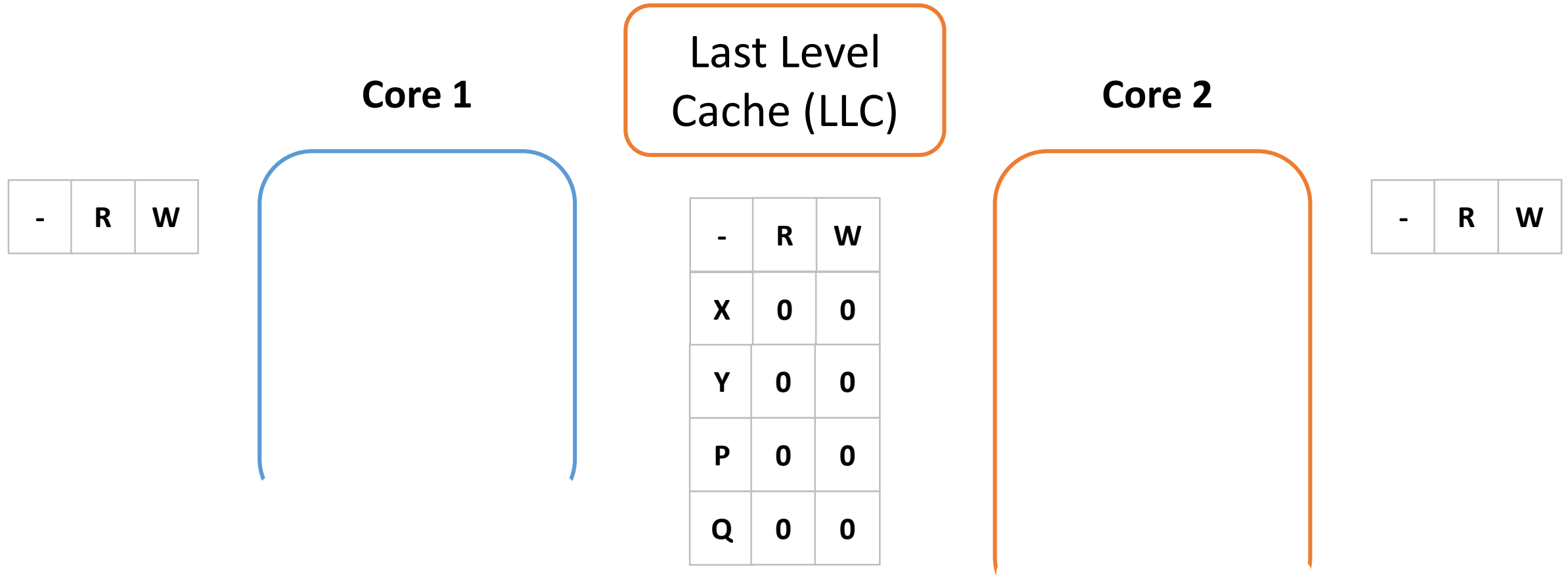
ARC: Practical Architecture Support for Region Conflict Exceptions

Design Overview

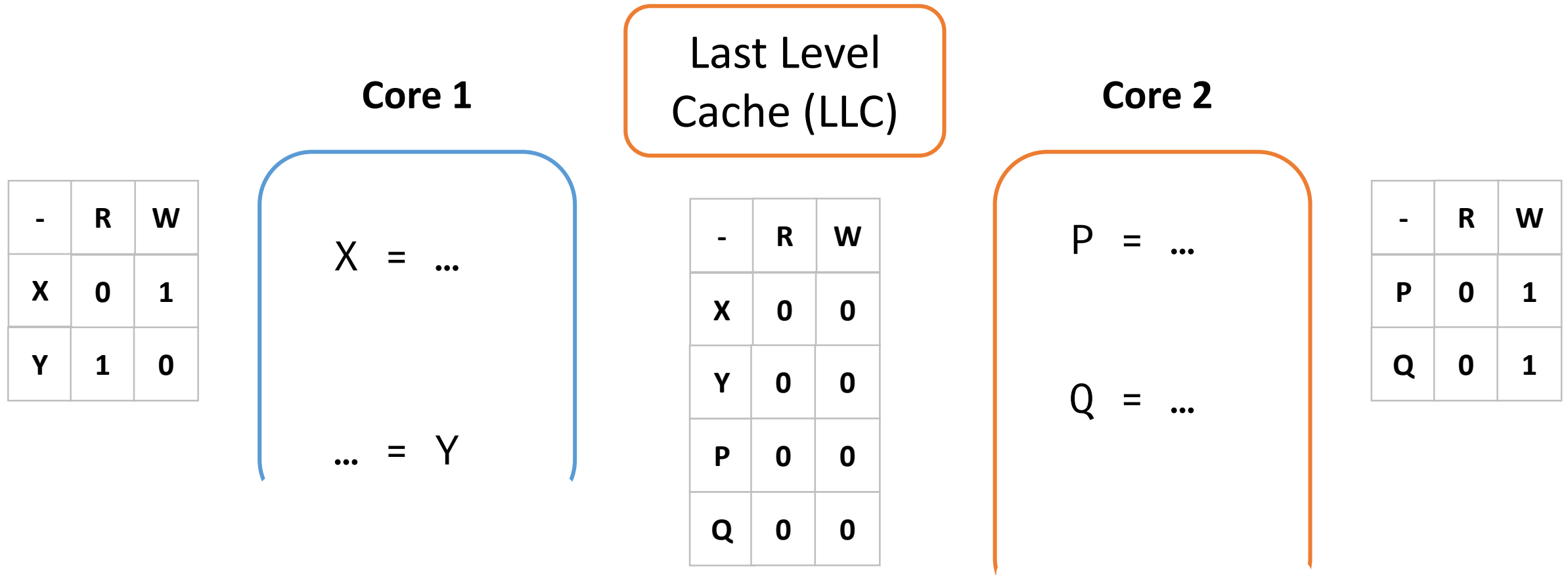
Architectural Modifications

- Example Executions with ARC

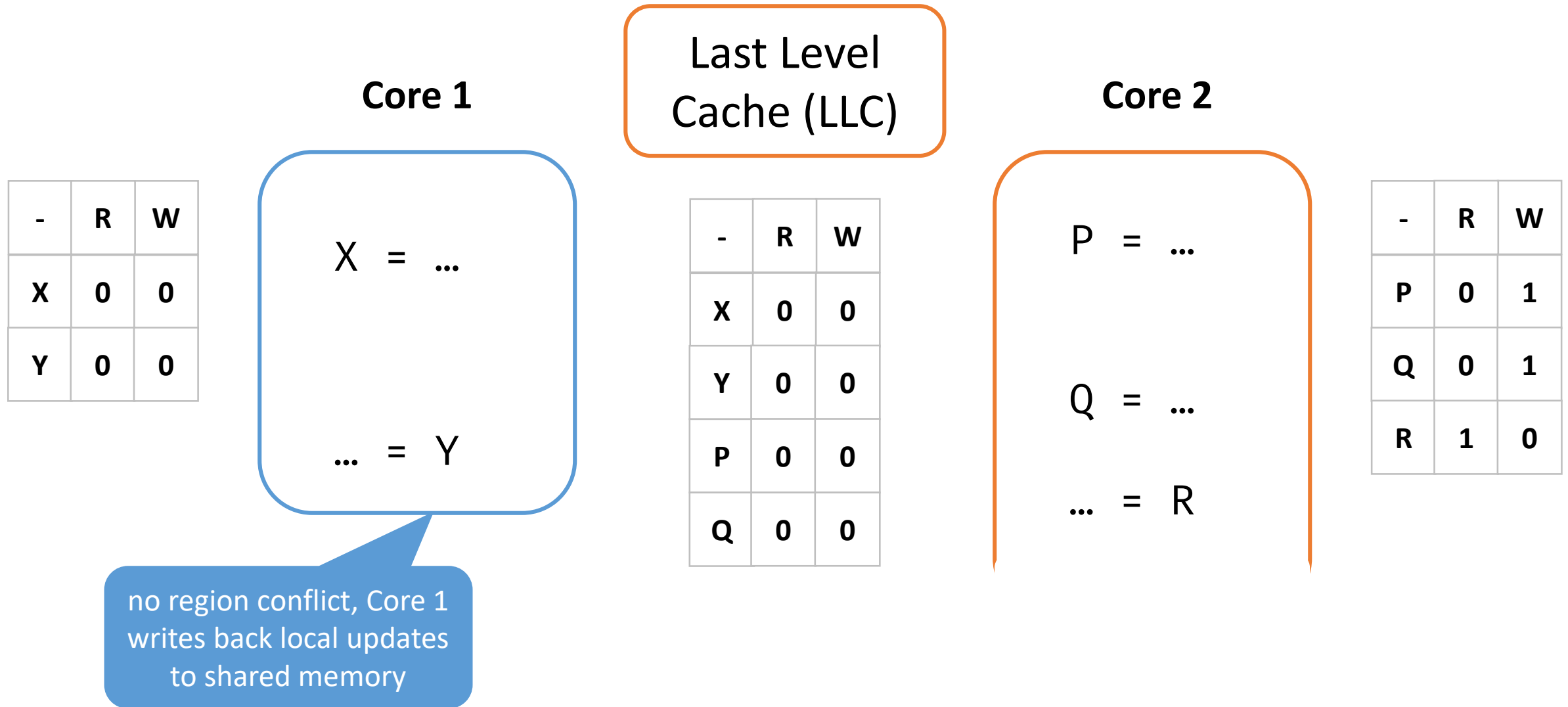
Example Execution with ARC: No Conflict



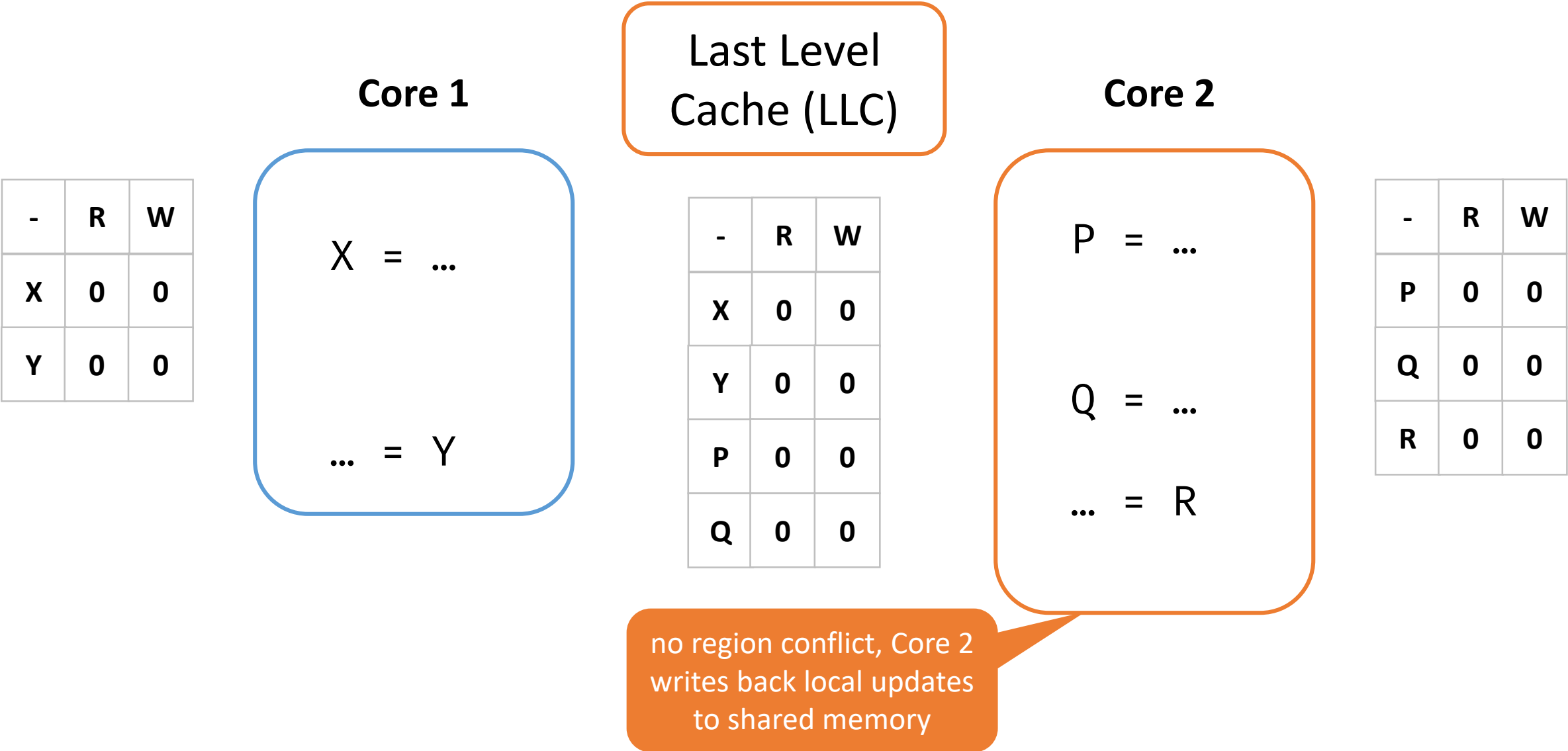
Example Execution with ARC: No Conflict



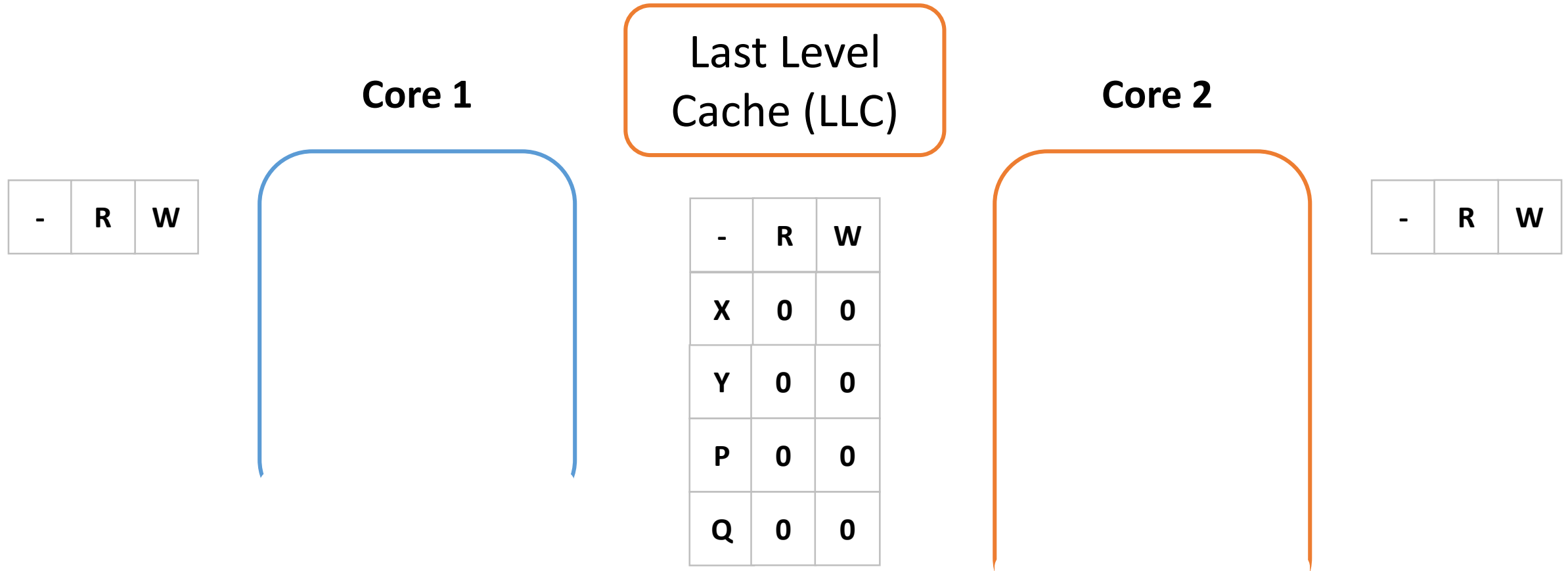
Example Execution with ARC: No Conflict



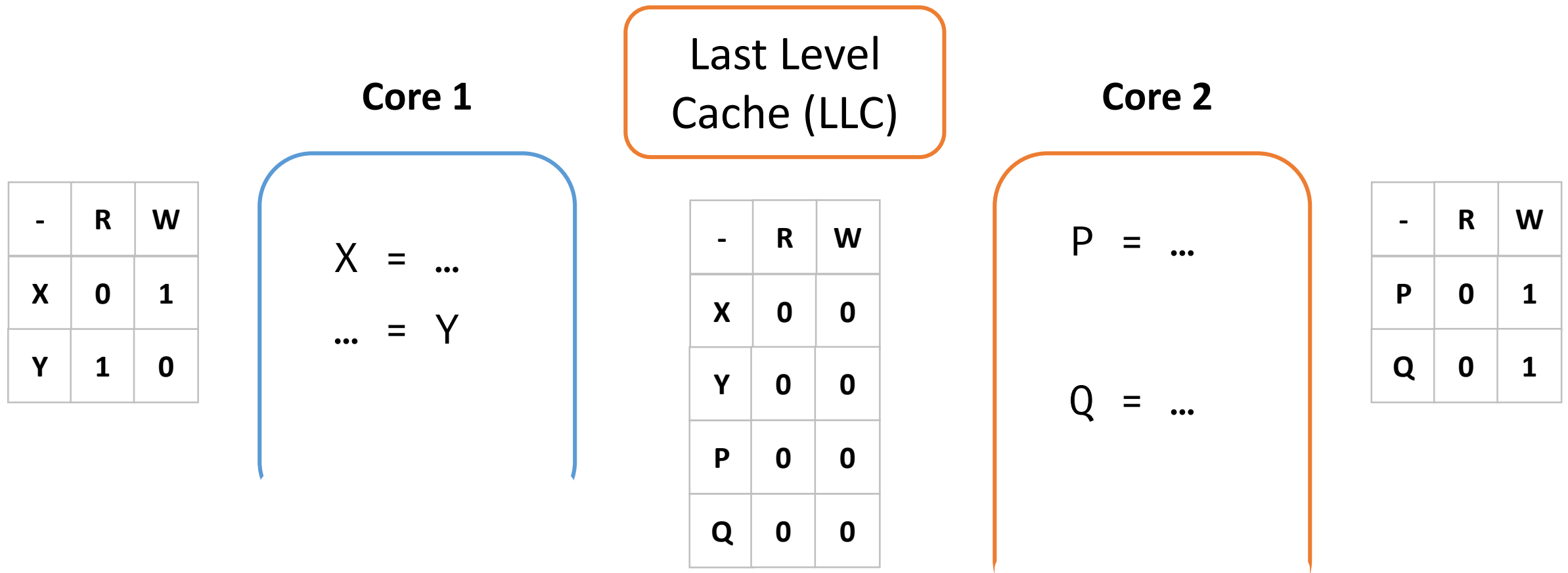
Example Execution with ARC: No Conflict



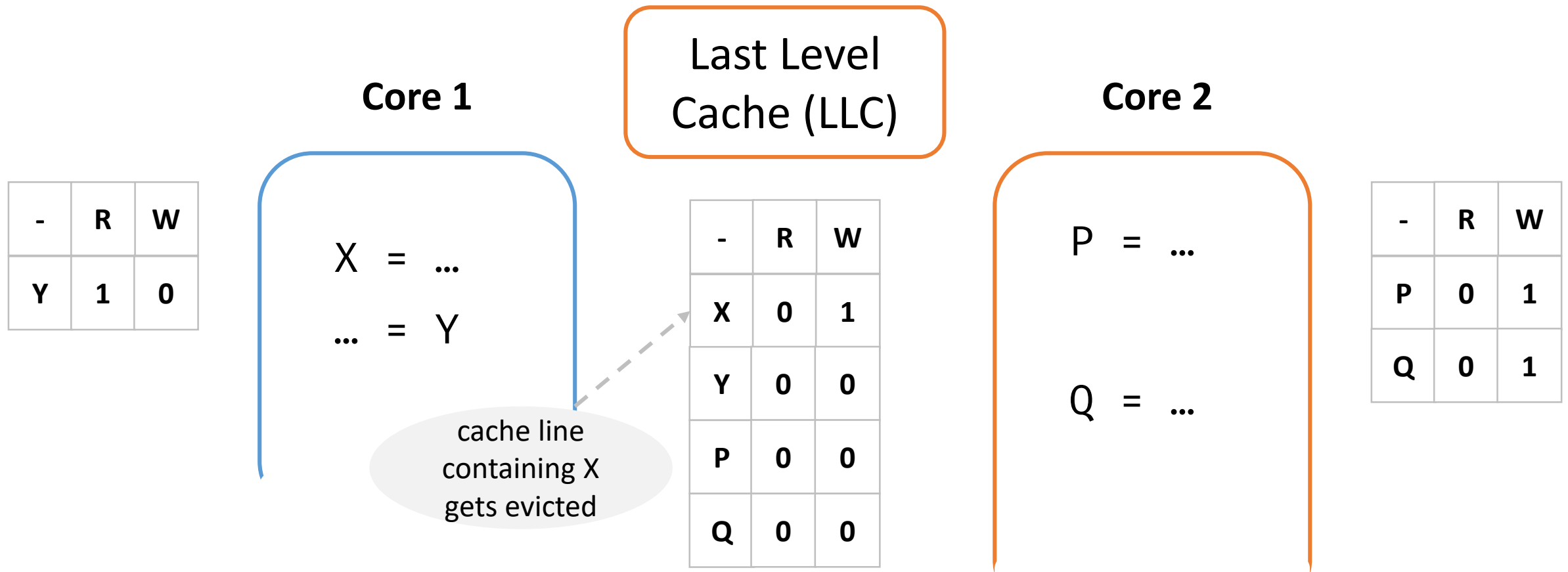
Eager Conflict Detection with ARC: Conflict on Evicted Line



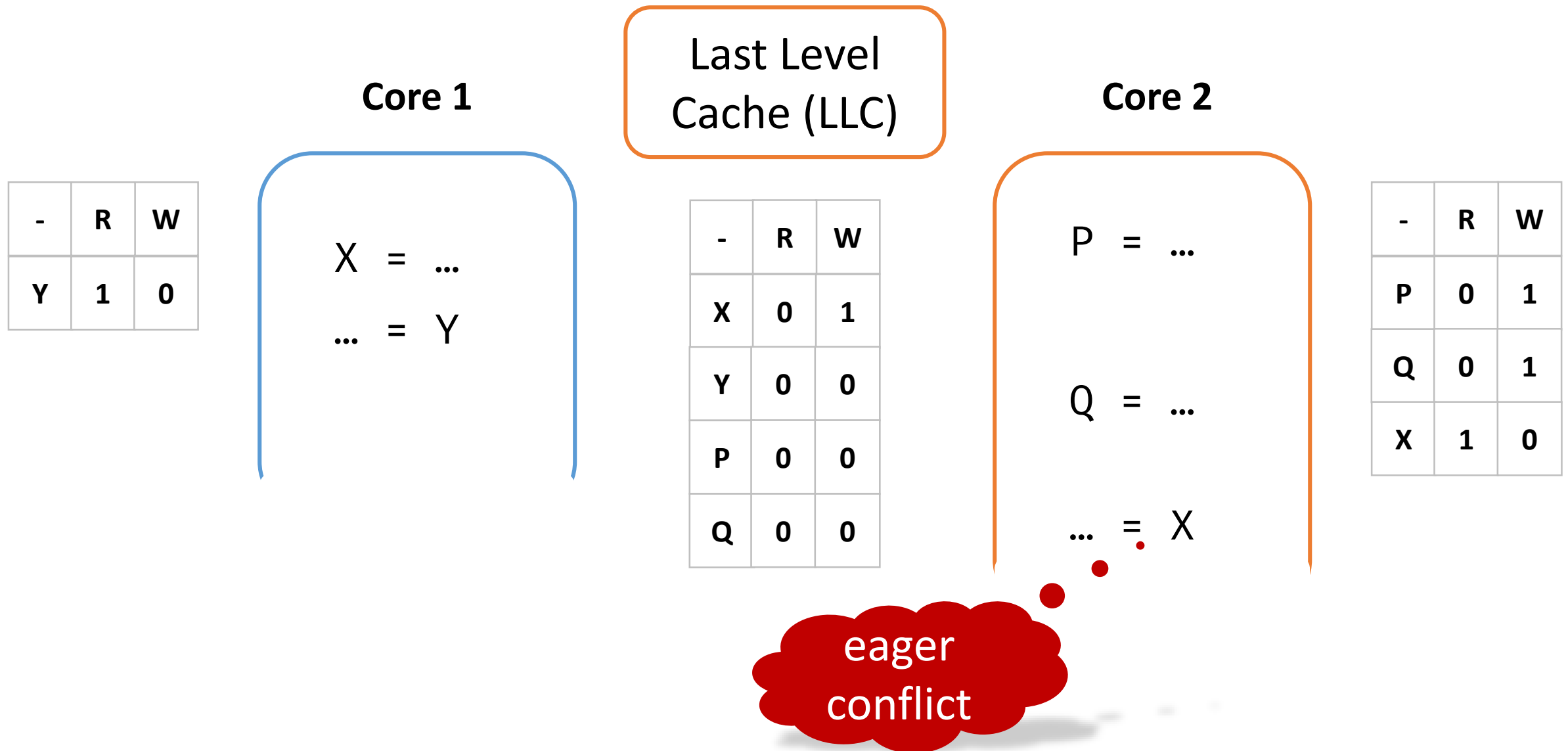
Eager Conflict Detection with ARC: Conflict on Evicted Line



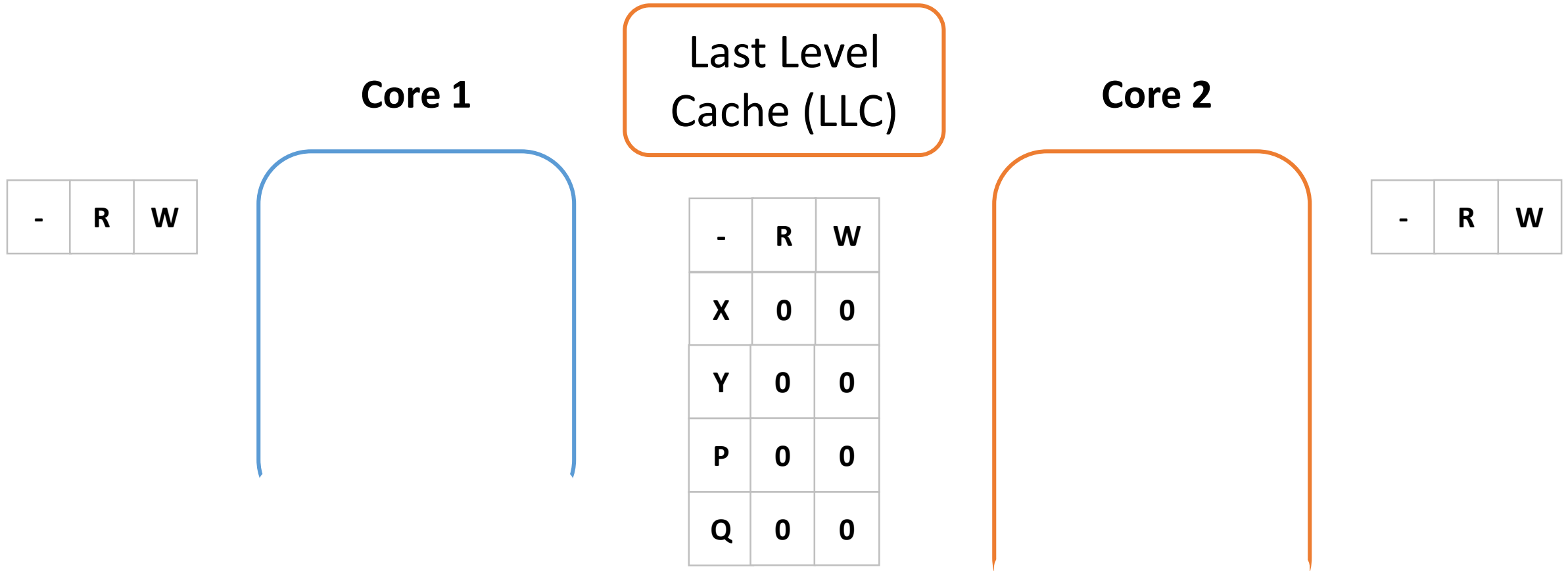
Eager Conflict Detection with ARC: Conflict on Evicted Line



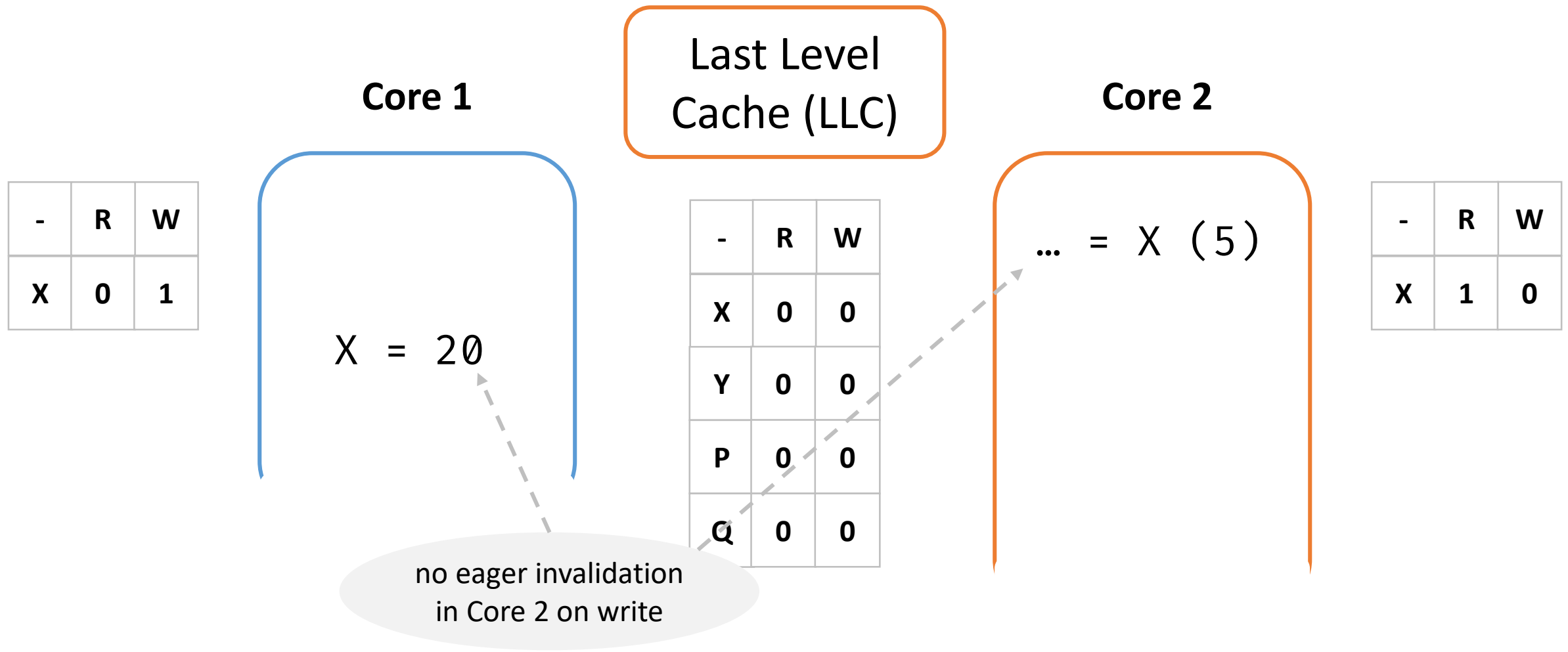
Eager Conflict Detection with ARC: Conflict on Evicted Line



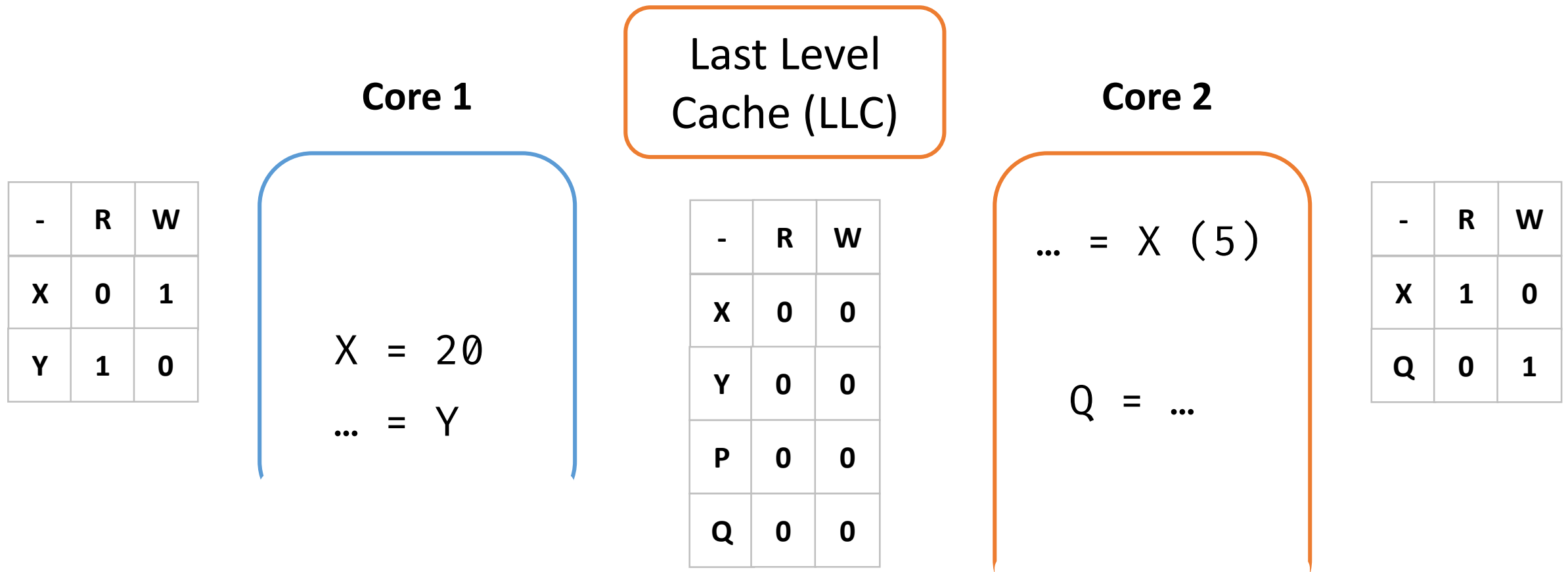
Lazy Conflict Detection with ARC: Conflict on Private Lines



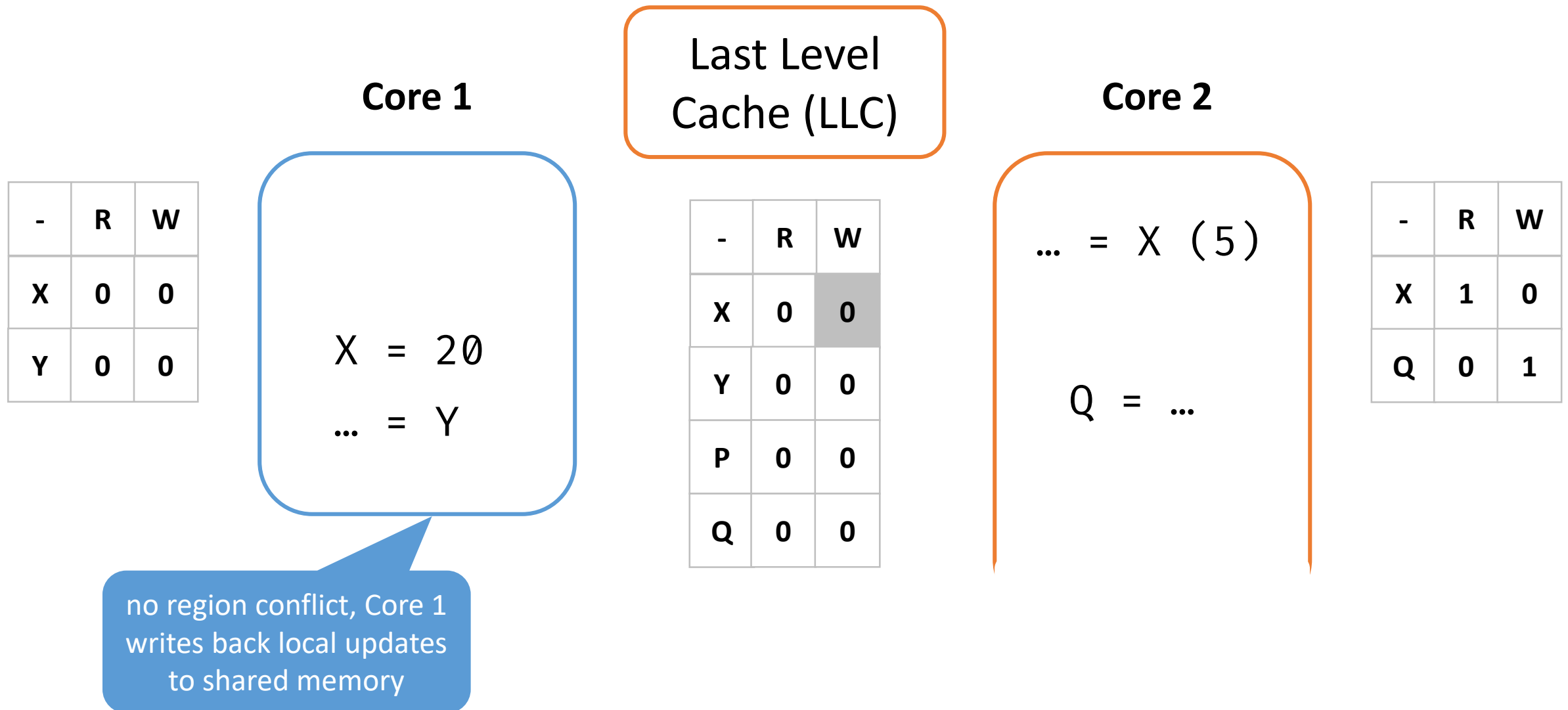
Lazy Conflict Detection with ARC: Conflict on Private Lines



Lazy Conflict Detection with ARC: Conflict on Private Lines



Lazy Conflict Detection with ARC: Conflict on Private Lines



Lazy Conflict Detection with ARC: Conflict on Private Lines

Core 1

-	R	W
X	0	0
Y	0	0

$X = 20$
 $\dots = Y$

no region conflict, Core 1
writes back local updates
to shared memory

**Last Level
Cache (LLC)**

-	R	W
X	0	0
Y	0	0
P	0	0
Q	0	0

fails to ensure
consistency of reads

Core 2

$\dots = X (5)$

$Q = \dots$

-	R	W
X	1	0
Q	0	1

lazy conflict
on X

Comparison of ARC with Related Approaches

Implementation and Evaluation

Comparing Conflict Exceptions and ARC

Conflict Exceptions

- Builds on M(O)ESI
 - Coherence at granularity of memory accesses
- Requires support for a Directory and point-to-point communication
- Detects conflicts eagerly

ARC

- Adapts release consistency and self-invalidation schemes
 - Coherence at region granularity
- Requires a AIM cache, write signatures, and consistency controllers
- Uses a mix of eager and lazy conflict detection

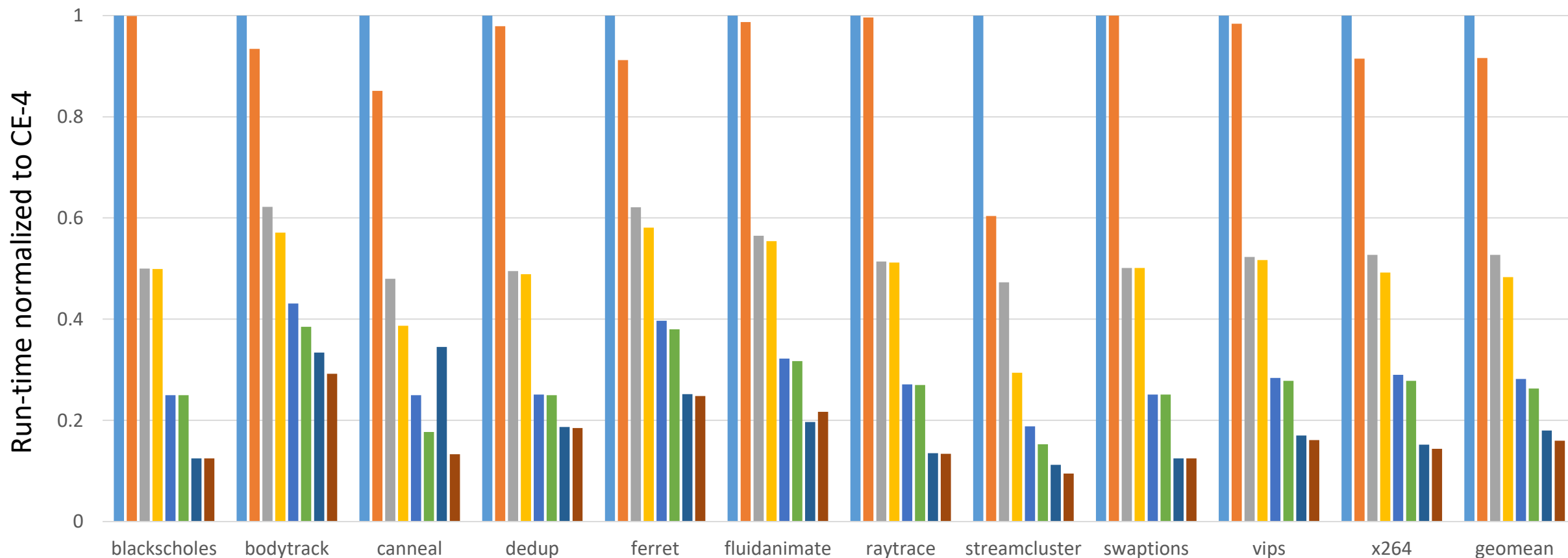
Implementation and Evaluation

- Simulation
 - A Pintool generates a stream of memory and synchronization events
 - Events are processed by model implementations of Conflict Exceptions (CE) and ARC
 - Use McPAT to estimate energy usage

Run-time Performance

normalized
to CE-4

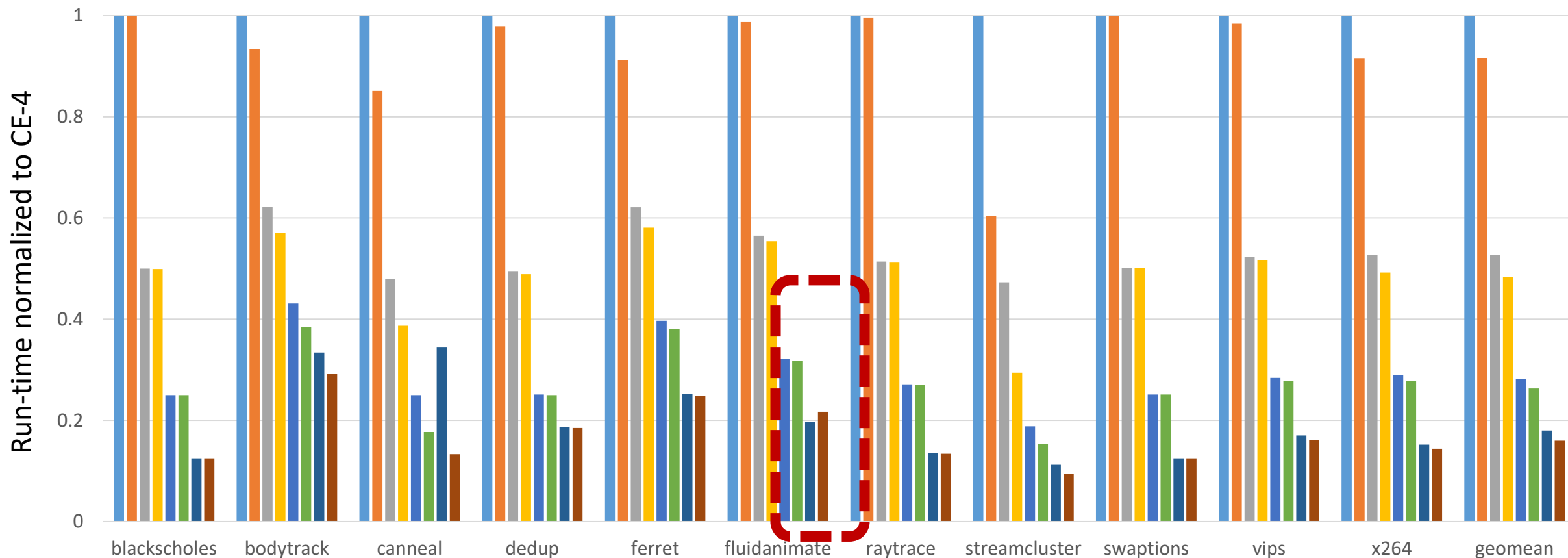
CE-4 ARC-4 CE-8 ARC-8 CE-16 ARC-16 CE-32 ARC-32



Run-time Performance

normalized
to CE-4

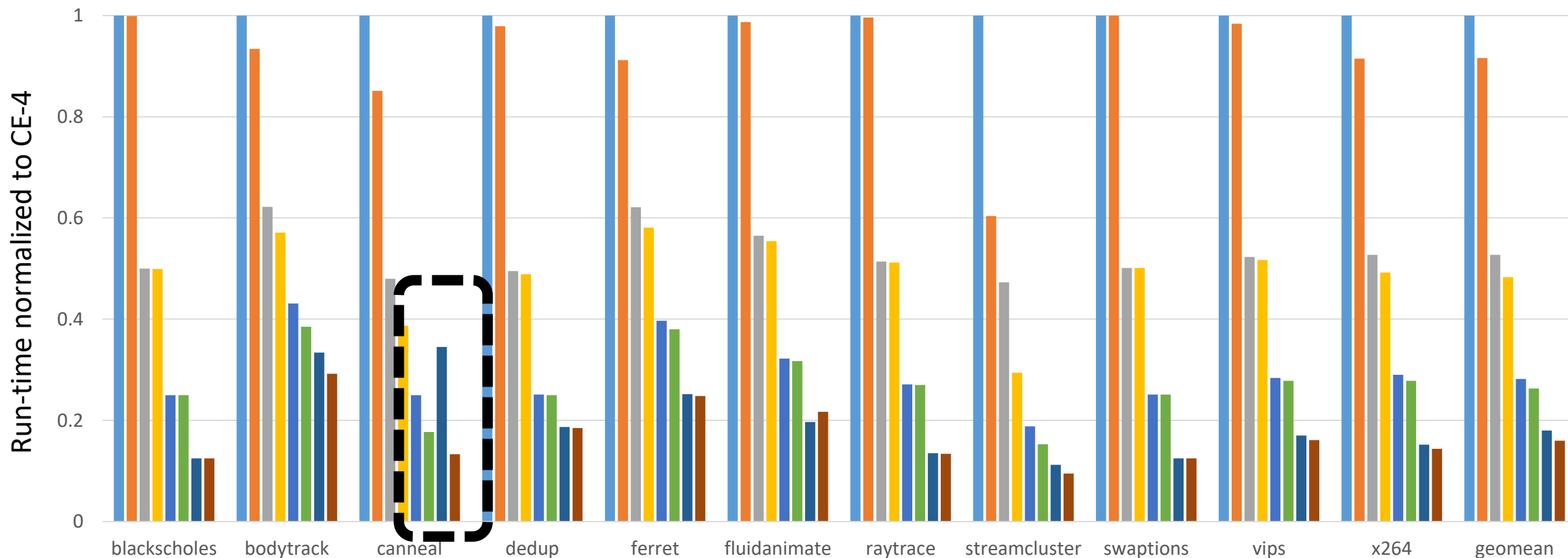
CE-4 ARC-4 CE-8 ARC-8 CE-16 ARC-16 CE-32 ARC-32



Run-time Performance

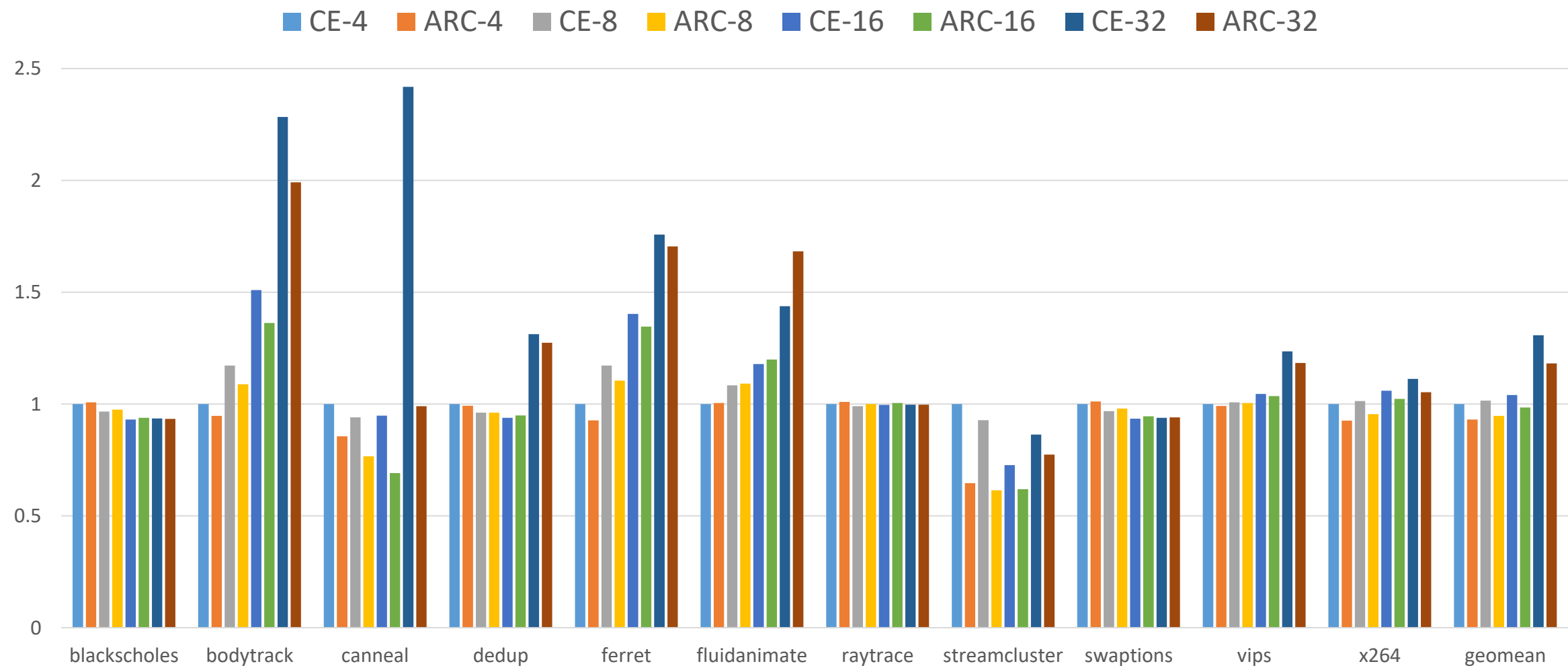
normalized
to CE-4

CE-4 ARC-4 CE-8 ARC-8 CE-16 ARC-16 CE-32 ARC-32



Energy Usage

normalized
to CE-4



Overhead of Providing Region Conflict Detection

- Current shared-memory systems provide undefined semantics for racy programs

Overhead comparison at 32 cores		
Approaches	Run-time performance (%)	Energy usage (%)
CE	26.7	41.4
ARC	12.5	27.8

**Key
Takeaways!**

Release consistency and self-invalidation techniques can be a good fit for detecting region conflicts

Small metadata cache provides reasonable tradeoffs between performance and complexity

Compared to state-of-art, ARC shows promise in making region conflict detection practical

Rethinking Support for Region Conflict Exceptions

Swarnendu Biswas, Rui Zhang, Michael D. Bond, and Brandon Lucia

IPDPS 2019