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

Brandon Lucia, Luis Ceze, Karin Strauss, Shaz Qadeer and Hans-J. Boehm

Microsoft Research

Data-Races are Trouble

Complicated language specifications

Usually incorrect, and difficult to debug

Negative impact on system reliability

What If...

Fail-Stop Semantics for Data-Races

Semantics are clear and simple

Better data-race debugging

Safety: races can't cause problems

When a data-race occurs, throw an exception

Requirements

- High-Performance - Always-on detection
- Precise detection - No false positives

Prior Work

	Happens-Before [Elmas'07, Flanagan'09]
Performance	✗
Precision	✓

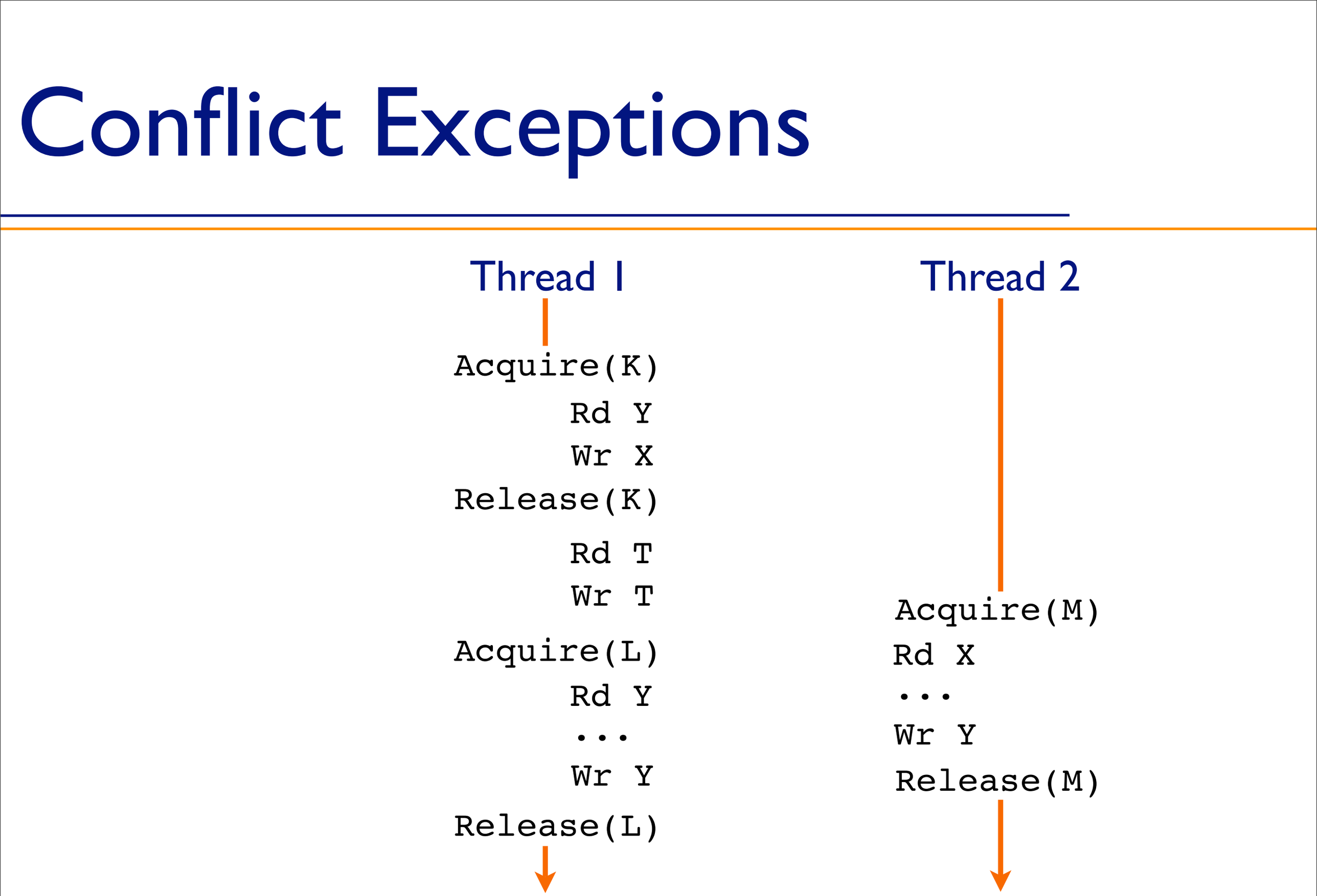
Prior Work

	Happens-Before [Elmas'07, Flanagan'09]	Approx. Methods [Savage'97, Zhou'07, Yu'05]
Performance	✗	✓
Precision	✓	✗

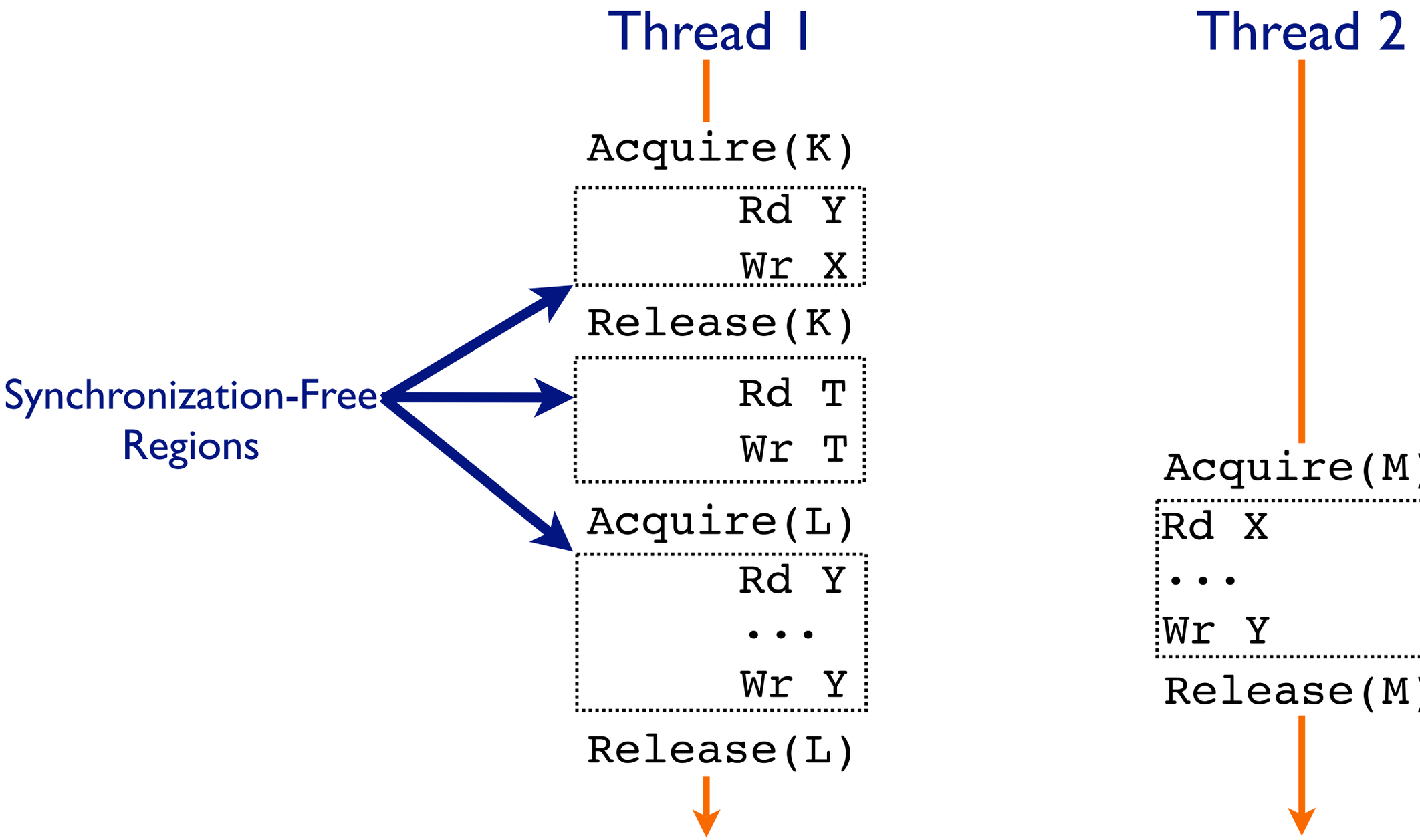
Prior Work

	Happens-Before [Elmas'07, Flanagan'09]	Approx. Methods [Savage'97, Zhou'07, Yu'05]	Conflict Exceptions [ISCA '10]
Performance	✗	✓	✓
Precision	✓	✗	✓

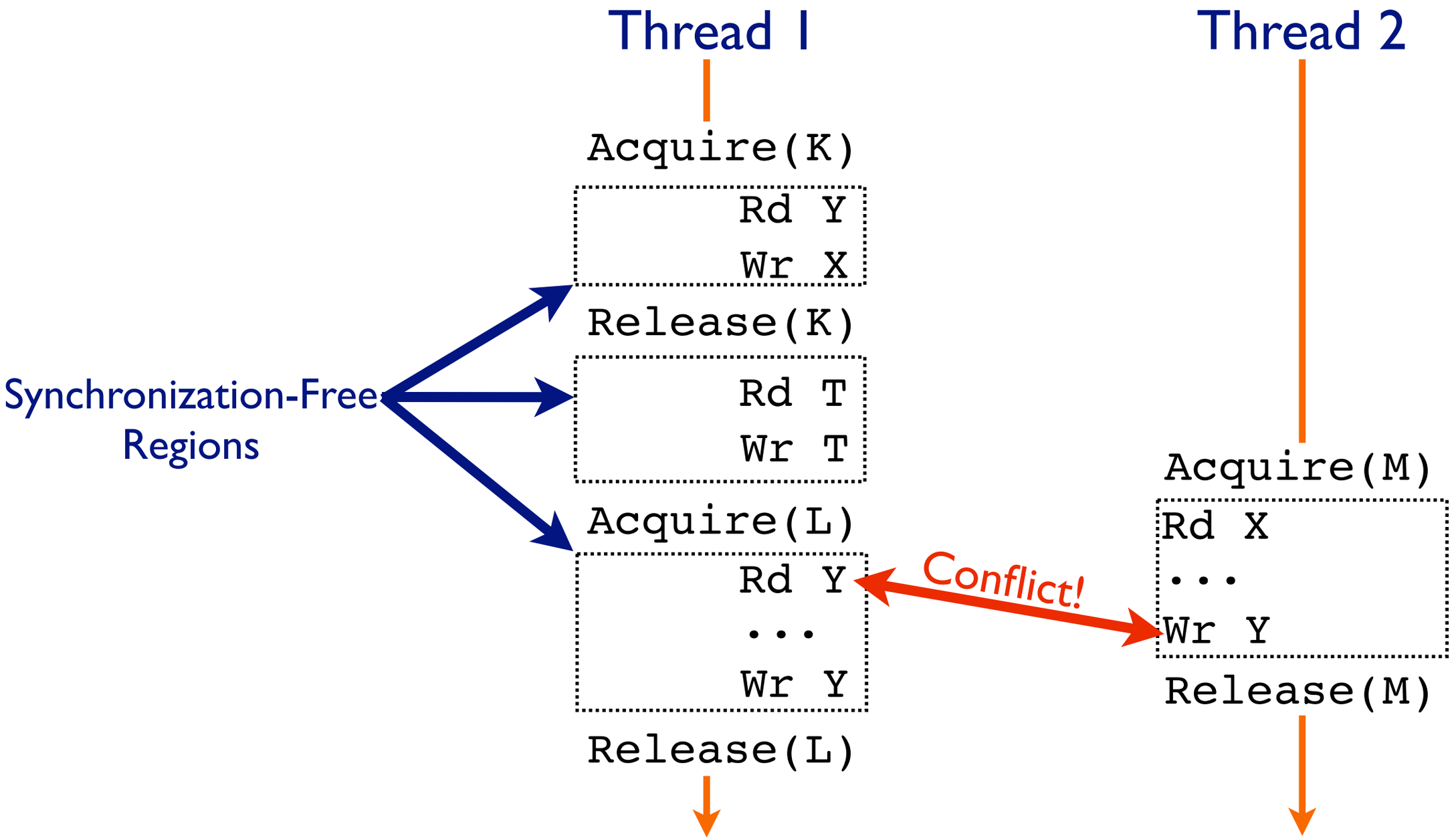
Conflict Exceptions



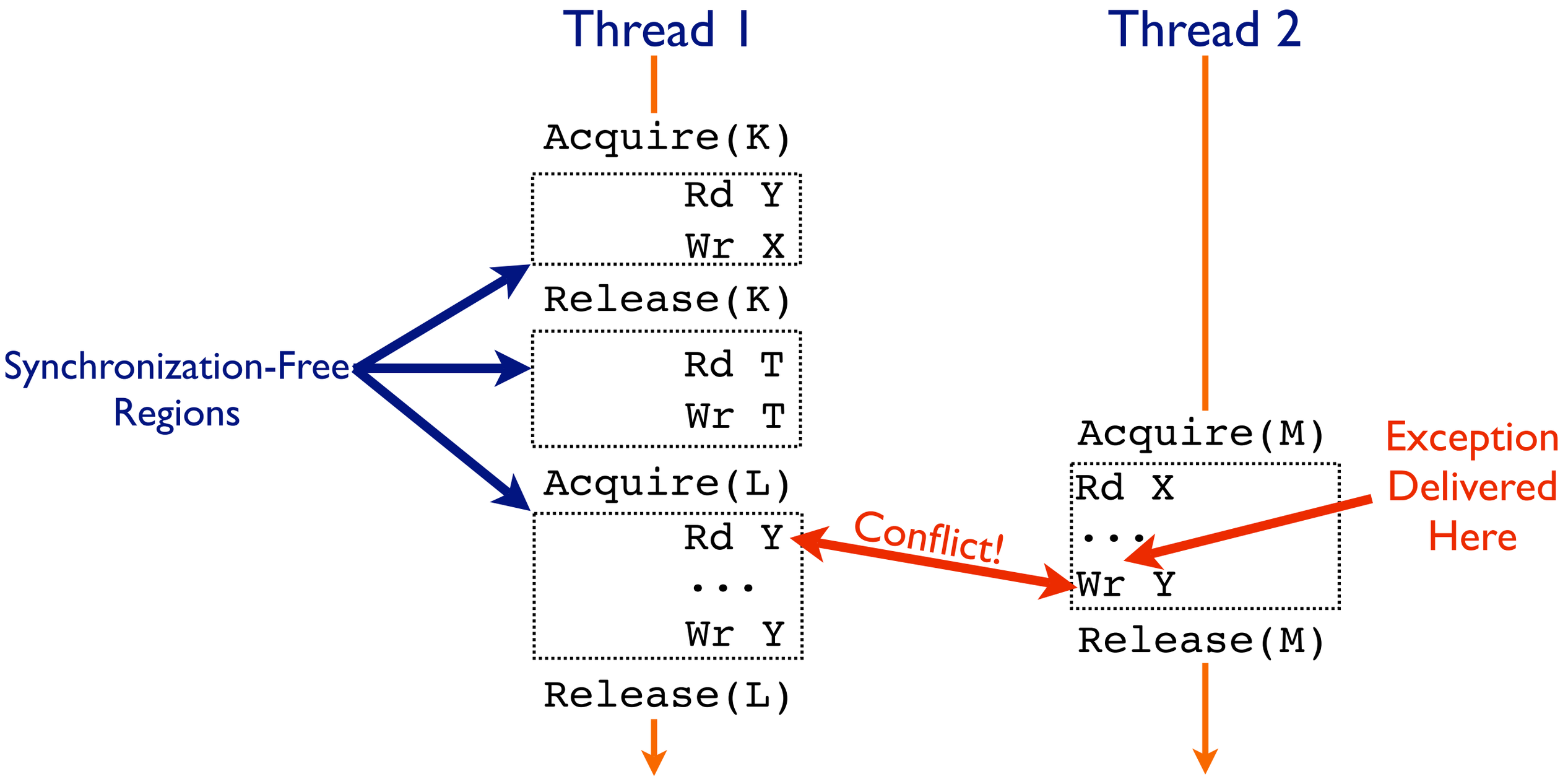
Conflict Exceptions



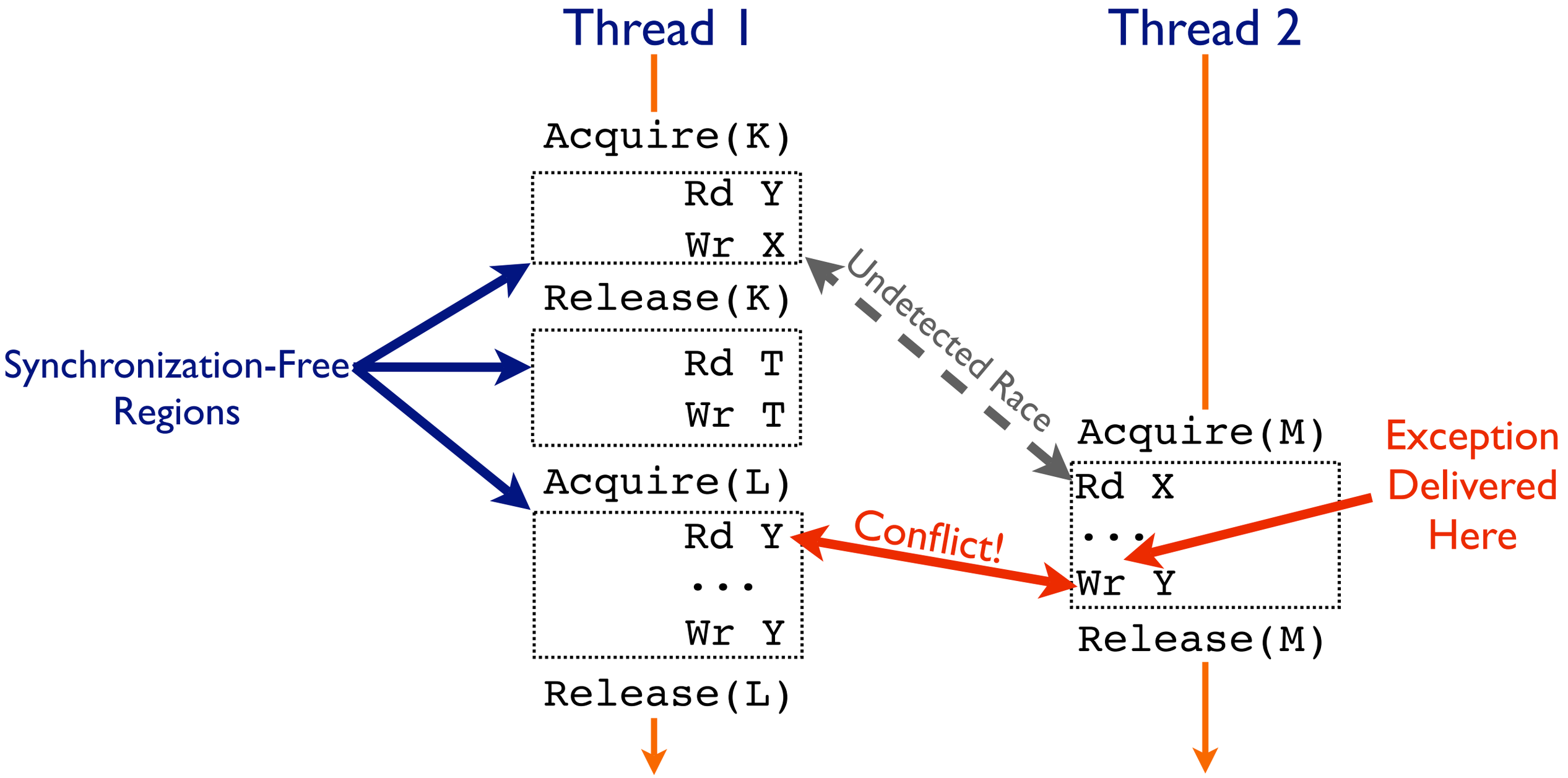
Conflict Exceptions



Conflict Exceptions



Conflict Exceptions



Conflict Exceptions

Precisely detect only races that can effect consistency

Ignoring unimportant races is key to performance

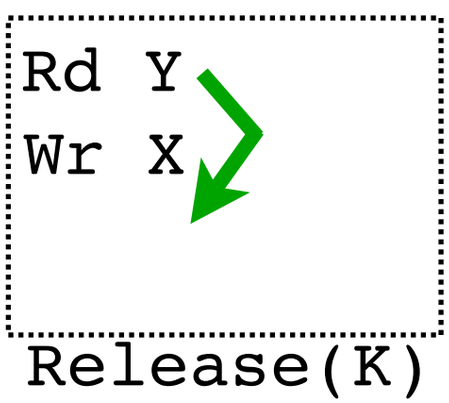
The **Guarantee**:

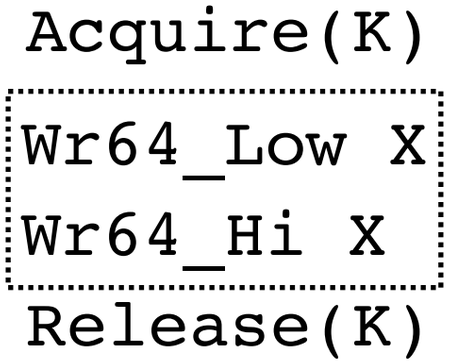
Exception-Thrown? There was a data-race.

Exception-Free? Sequential Consistency.

Language Level Benefits

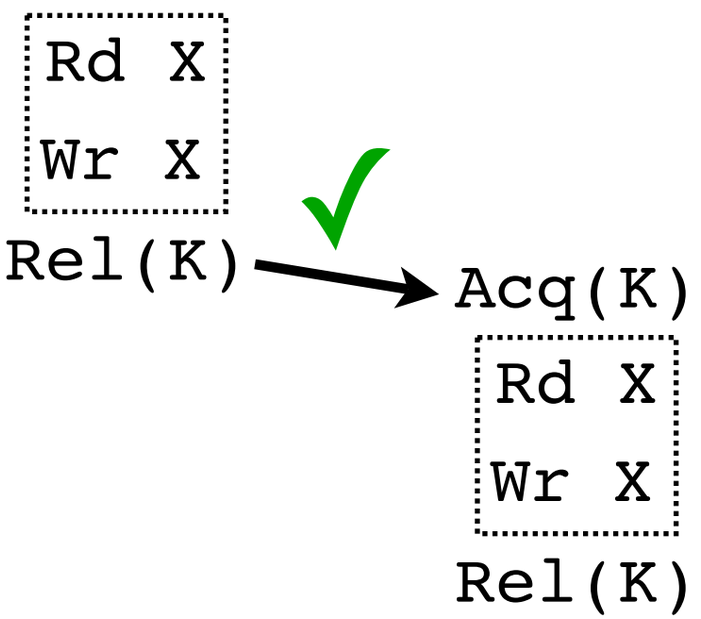
Reordering in SFRs is legal





Granularity independence

Exception-Free executions are SC



Language Level Benefits

Programming is the same

```
pthread_lock(K)
   Rd  Y
   Wr  X
   Wr  Q
   Wr  Z
pthread_unlock(K)
```

```
Acq(K)
 Rd  X
 Wr  X
                 !
                      Acq(L)
                       Rd  X
```

Racy programs are well-behaved

Race semantics are simpler

w such that $w.v = r.v$ and $W(r) \xrightarrow{hb} w \xrightarrow{hb} r$.

5.4 Causality Requirements for Executions

A well-formed execution

$$E = \langle P, A, \xrightarrow{po}, \xrightarrow{so}, W, V, \xrightarrow{sw}, \xrightarrow{hb} \rangle$$

is validated by *committing* actions from A. If all of the actions in A can be committed, then the execution satisfies the causality requirements of the Java memory model.

Starting with the empty set as C_0, we perform a sequence of steps where we take actions from the set of actions A and add them to a set of committed actions C_i to get a new set of committed actions C_{i+1}. To demonstrate that this is reasonable, for each C_i we need to demonstrate an execution E_i containing C_i that meets certain conditions.

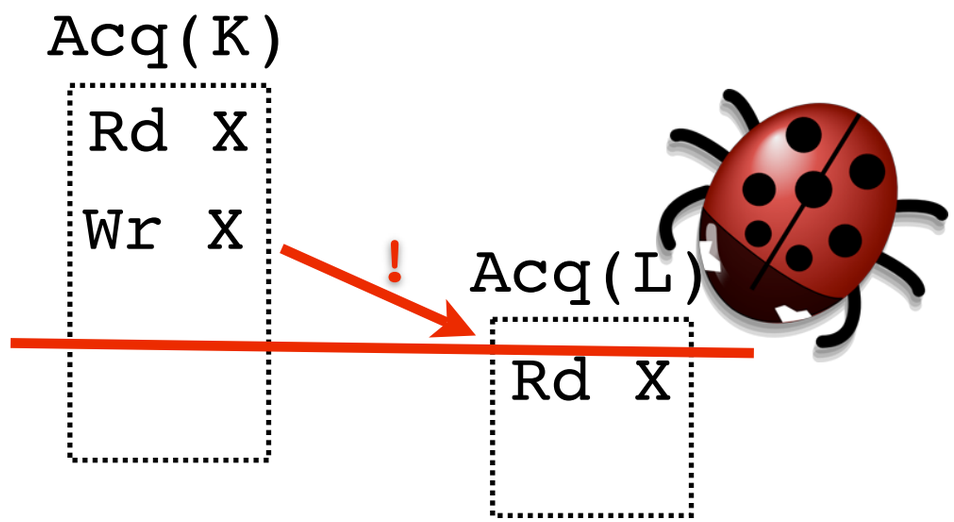
Formally, an execution E satisfies the causality requirements of the Java memory model if and only if there exist

- Sets of actions $C_0, C_1, \ldots$ such that
  - $C_0 = \emptyset$
  - $C_i \subset C_{i+1}$

Debugging and Reliability

Concurrent, conflicting SFRs *throw exceptions*

All races have some exceptional schedule



Exception Handling: Log + Recover

Damage Control: Shut down buggy module

System Support for Conflict Exceptions

Hardware/Software Interface

New Instructions:
`BeginRegion` and `EndRegion`

Synchronization Operations
are Singleton Regions

Exceptions Thrown Precisely
Before Conflicting Instruction

Hardware/Software Interface

New Instructions:
`BeginRegion` and `EndRegion`

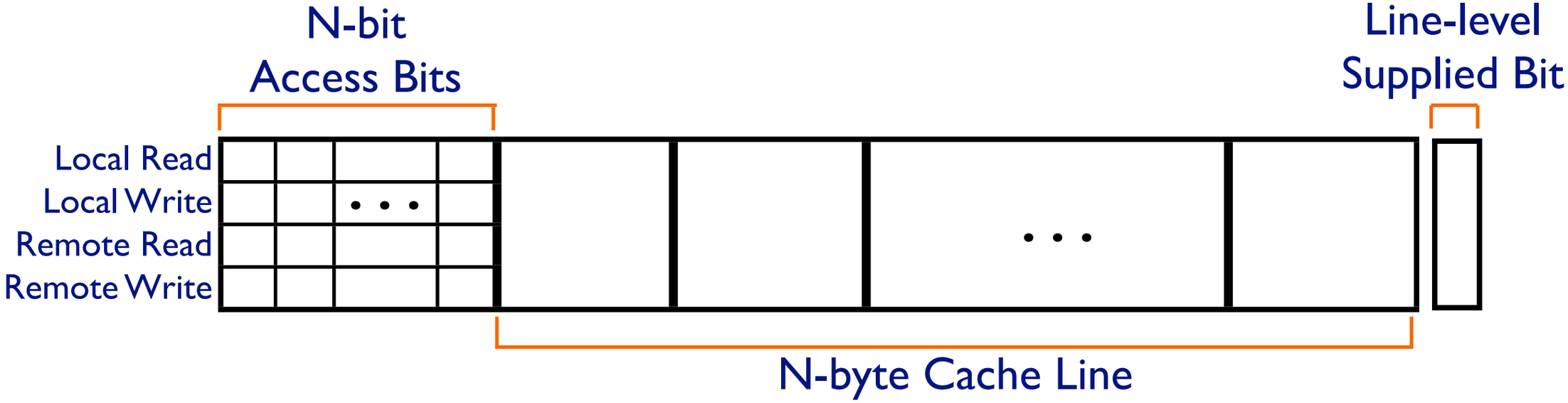
Synchronization Operations
are Singleton Regions

Exceptions Thrown Precisely
Before Conflicting Instruction

```
Acquire(K)

BeginRegion
        Rd Y
        Wr X
EndRegion

Release(K)

BeginRegion
        Rd T
        Wr T
EndRegion
```

Access Monitoring

Byte-granular access information is required

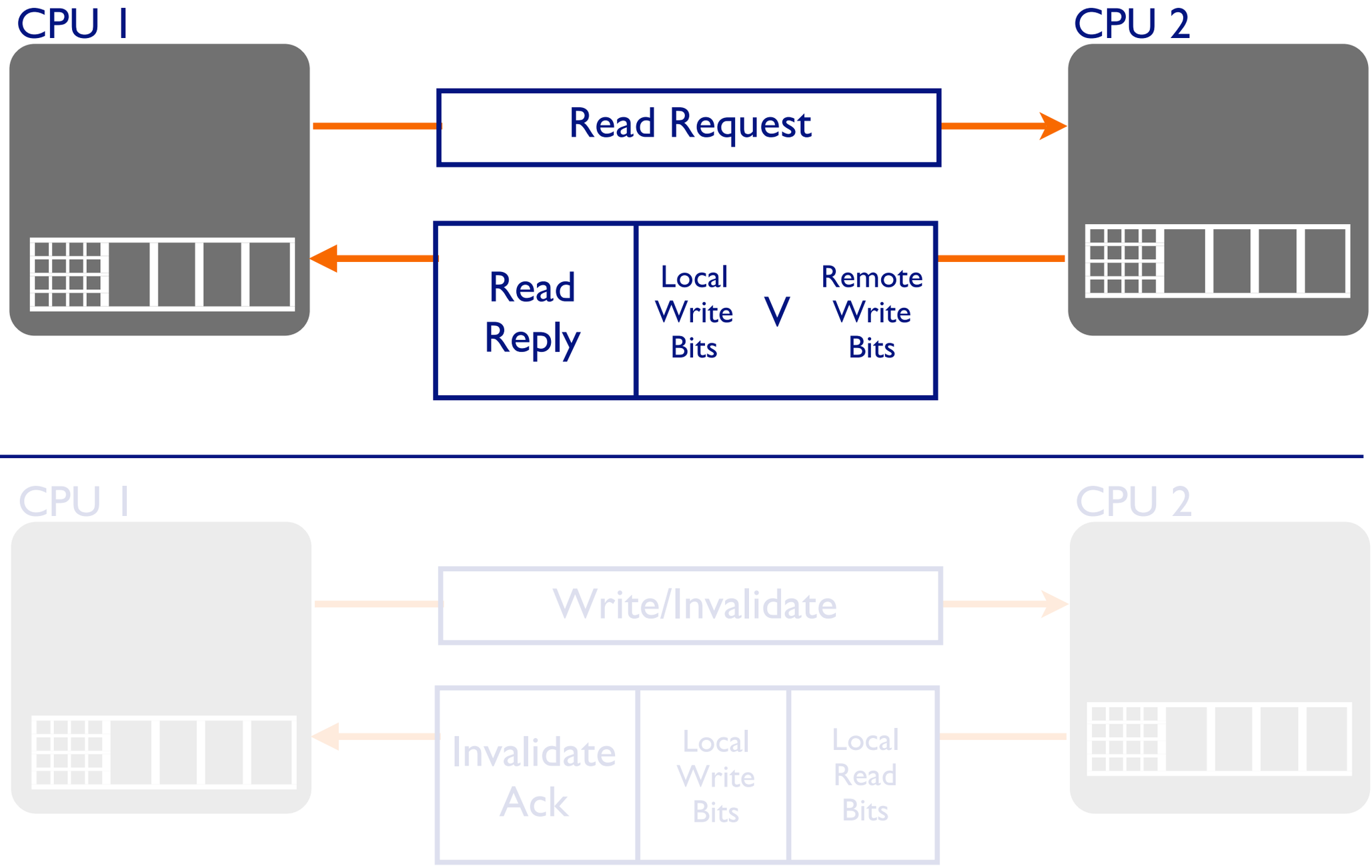


Exception Test: compare appropriate local and remote bits

sampa

Coherence Support

Read Coherence Actions

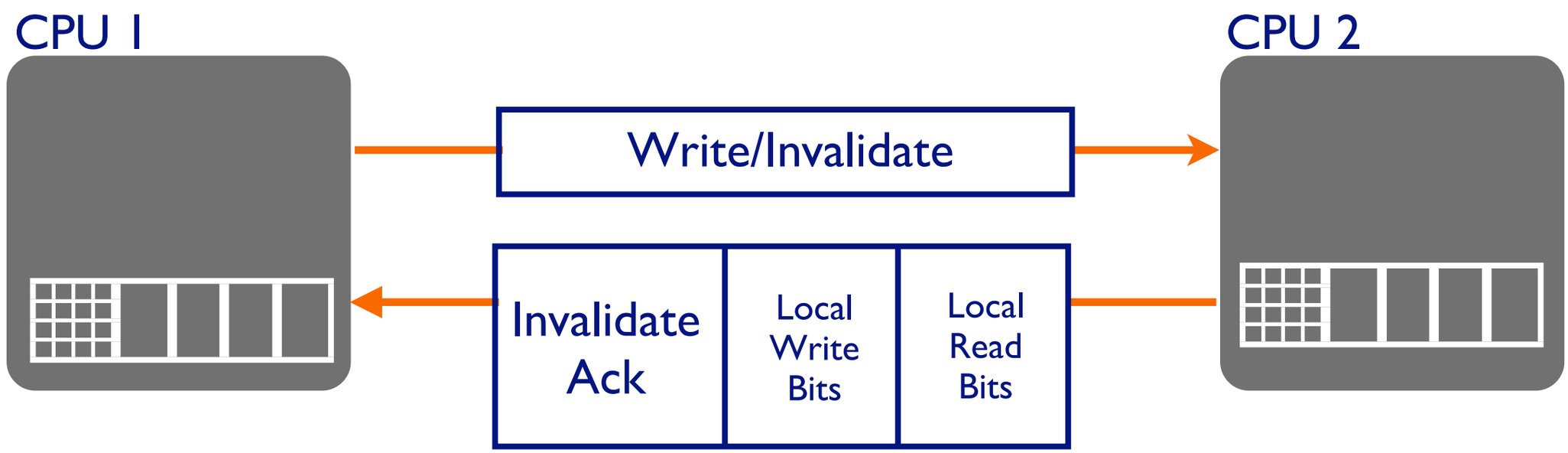


Write Coherence Actions

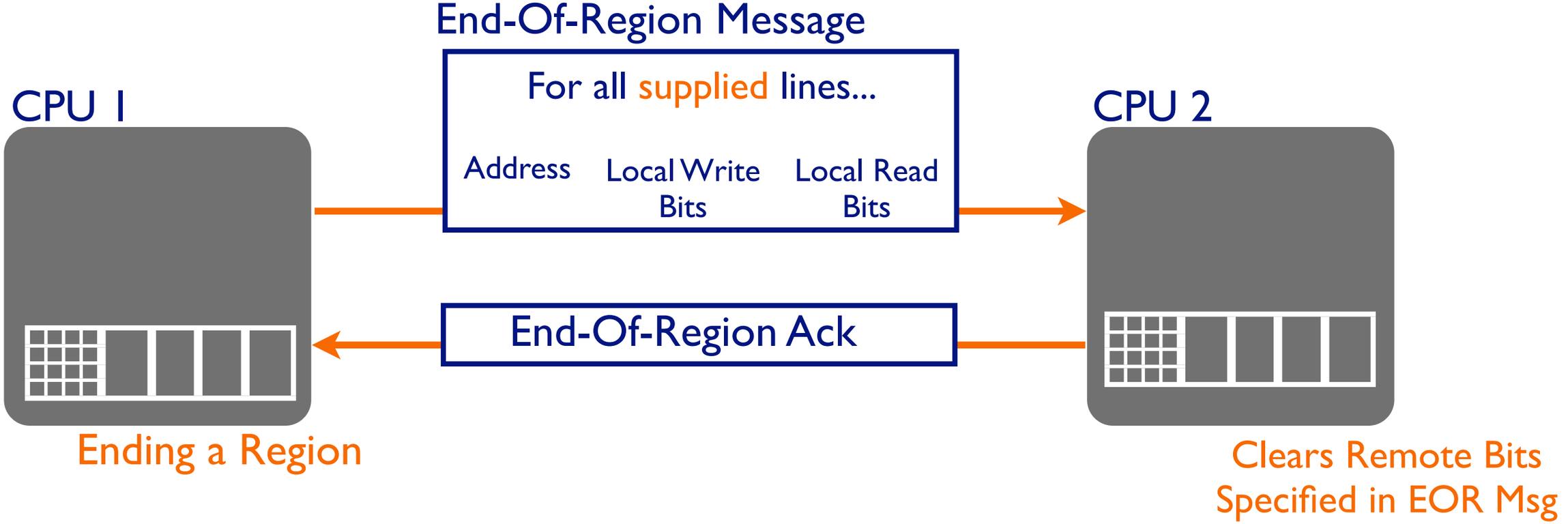
Coherence Support

Read Coherence Actions

Write Coherence Actions

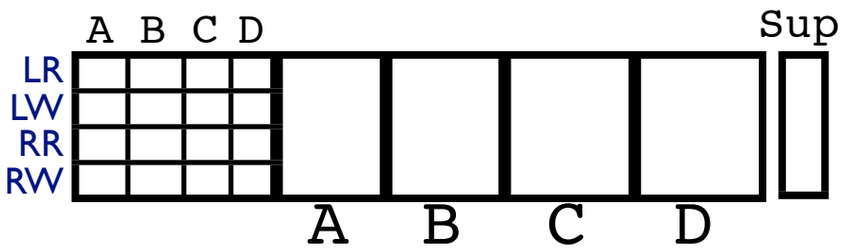


Ending a Region

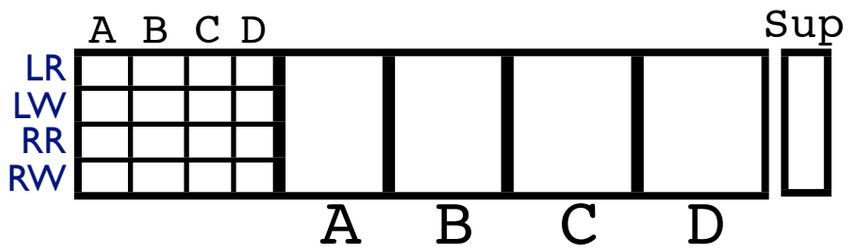


Putting It Together

CPU 1's Cache



CPU 2's Cache



CPU 1's Code

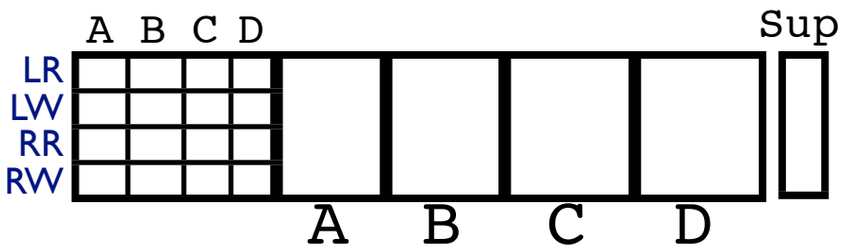
```
BeginRegion
Wr A
EndRegion
BeginRegion
Wr C
```

CPU 2's Code

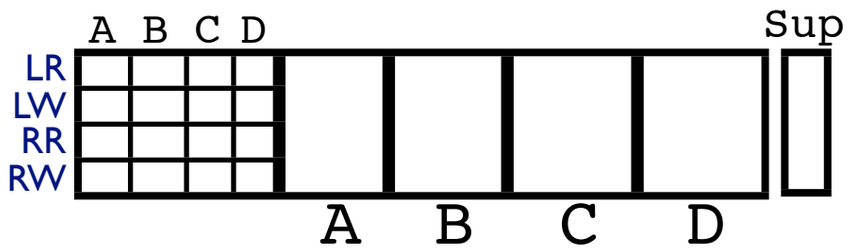
```
BeginRegion
Rd C
```

Putting It Together

CPU 1's Cache



CPU 2's Cache



CPU 1's Code

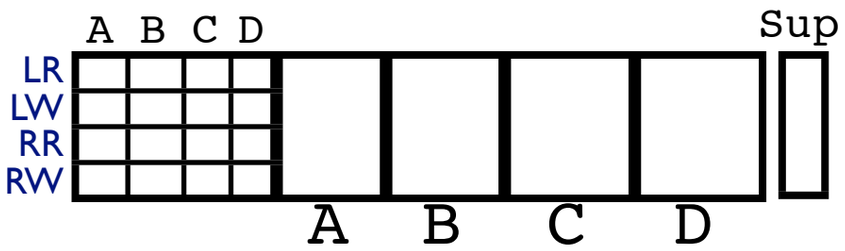
```
BeginRegion
Wr A
EndRegion
BeginRegion
Wr C
```

CPU 2's Code

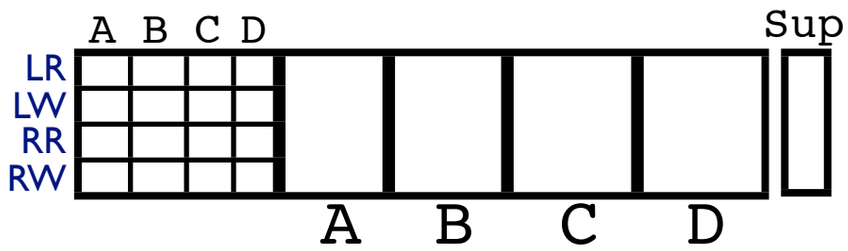
```
BeginRegion
Rd C
```

Putting It Together

CPU 1's Cache



CPU 2's Cache



CPU 1's Code

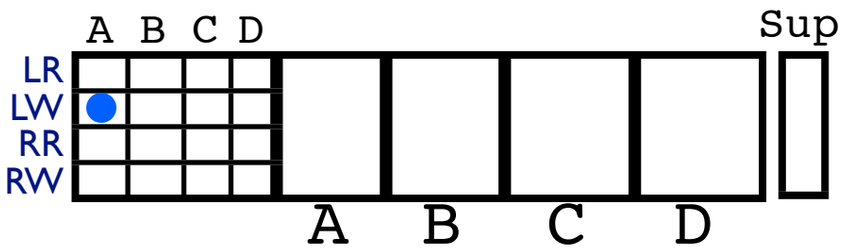
```
BeginRegion
Wr A
EndRegion
BeginRegion
Wr C
```

CPU 2's Code

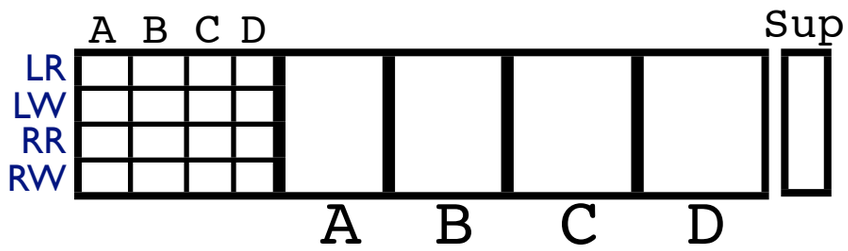
```
BeginRegion
Rd C
```

Putting It Together

CPU 1's Cache



CPU 2's Cache



CPU 1's Code

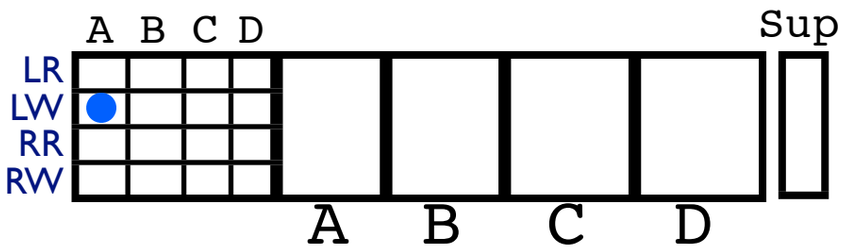
```
BeginRegion
Wr A
EndRegion
BeginRegion
Wr C
```

CPU 2's Code

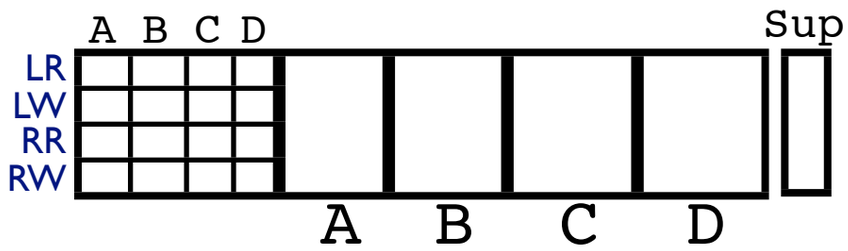
```
BeginRegion
Rd C
```

Putting It Together

CPU 1's Cache



CPU 2's Cache



CPU 1's Code

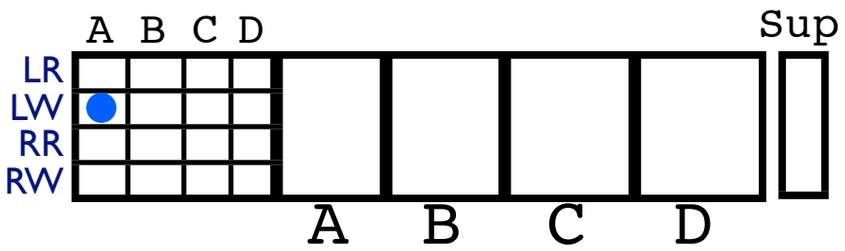
```
BeginRegion
Wr A
EndRegion
BeginRegion
Wr C
```

CPU 2's Code

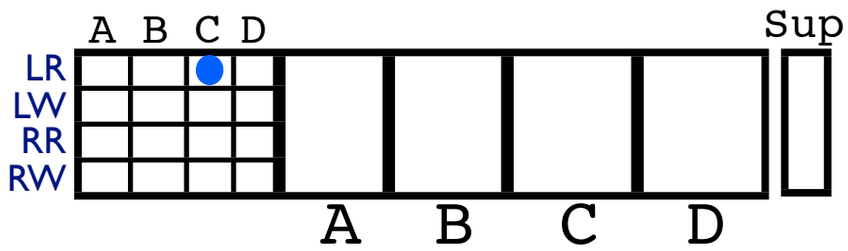
```
BeginRegion
Rd C
```

Putting It Together

CPU 1's Cache



CPU 2's Cache



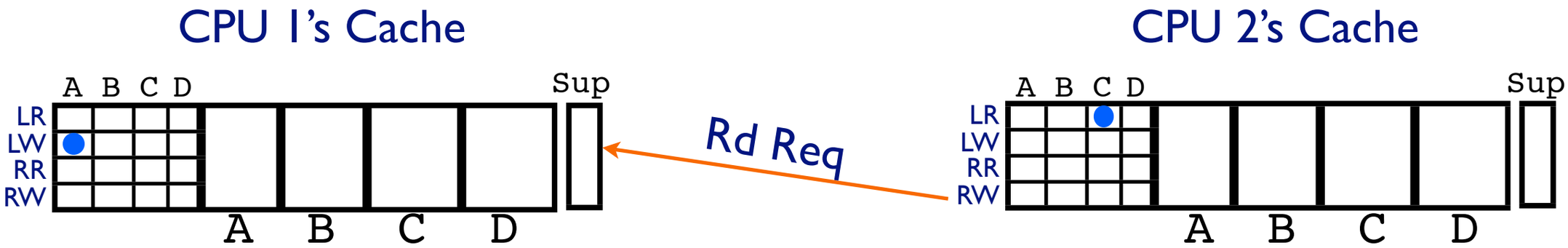
CPU 1's Code

```
BeginRegion
Wr A
EndRegion
BeginRegion
Wr C
```

CPU 2's Code

```
BeginRegion
Rd C
```

Putting It Together



CPU 1's Code

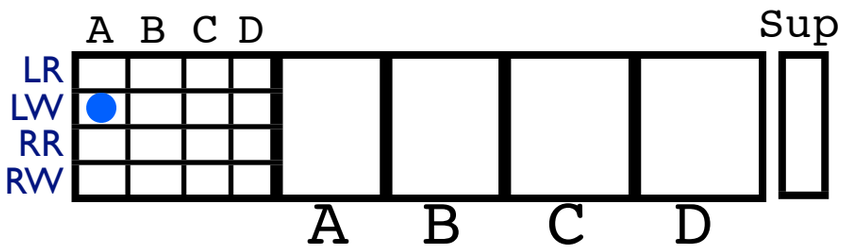
```
BeginRegion
Wr A
EndRegion
BeginRegion
Wr C
```

CPU 2's Code

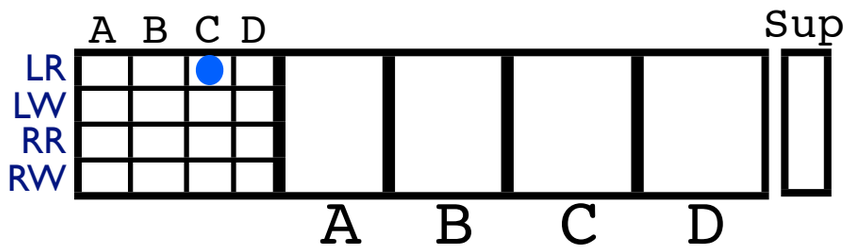
```
BeginRegion
Rd C
```

Putting It Together

CPU 1's Cache



CPU 2's Cache



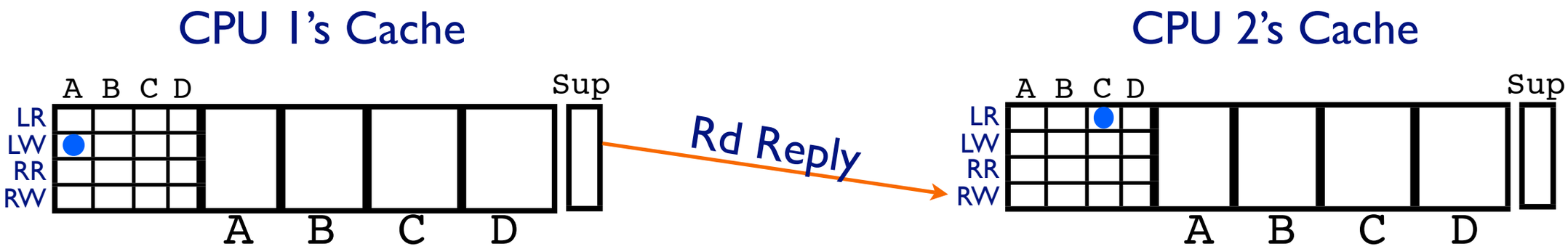
CPU 1's Code

```
BeginRegion
Wr A
EndRegion
BeginRegion
Wr C
```

CPU 2's Code

```
BeginRegion
Rd C
```

Putting It Together



CPU 1's Code

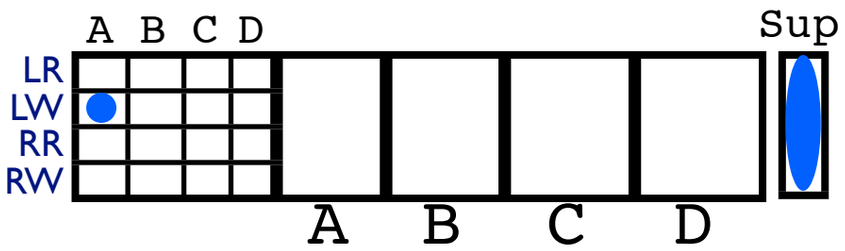
```
BeginRegion
Wr A
EndRegion
BeginRegion
Wr C
```

CPU 2's Code

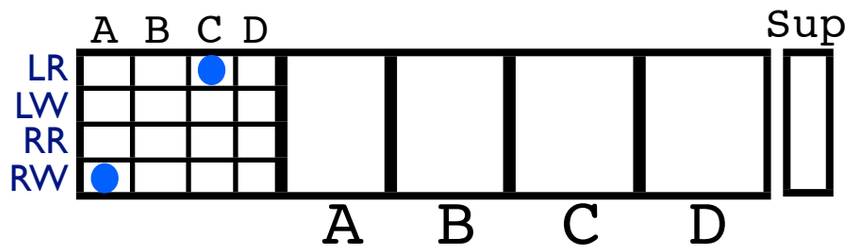
```
BeginRegion
Rd C
```

Putting It Together

CPU 1's Cache



CPU 2's Cache



CPU 1's Code

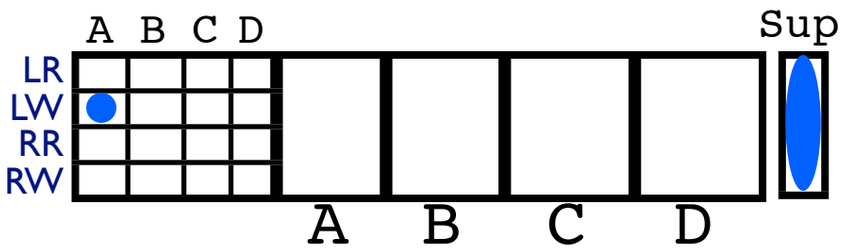
```
BeginRegion
Wr A
EndRegion
BeginRegion
Wr C
```

CPU 2's Code

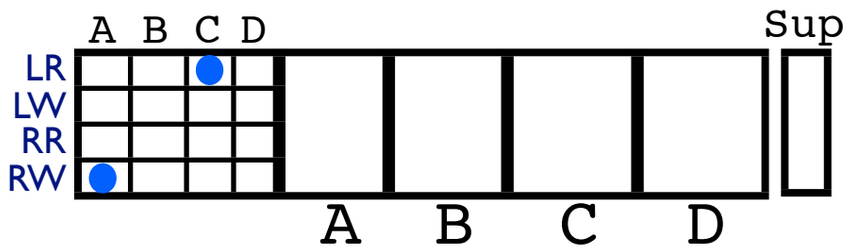
```
BeginRegion
Rd C
```

Putting It Together

CPU 1's Cache



CPU 2's Cache



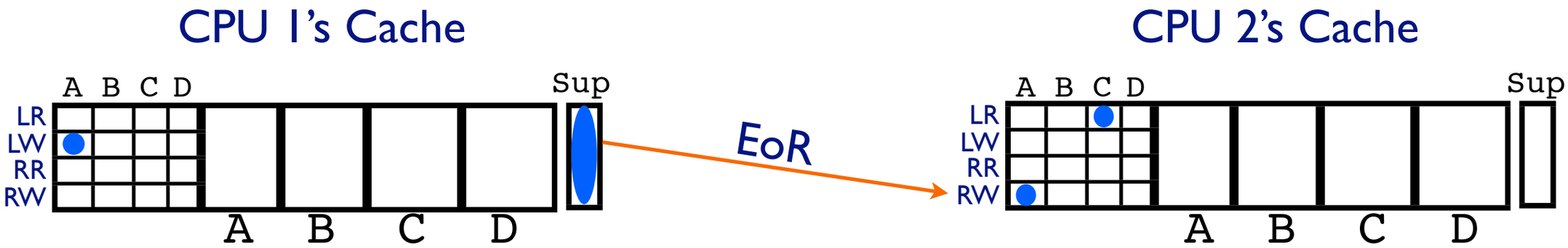
CPU 1's Code

```
BeginRegion
Wr A
EndRegion
BeginRegion
Wr C
```

CPU 2's Code

```
BeginRegion
Rd C
```

Putting It Together



CPU 1's Code

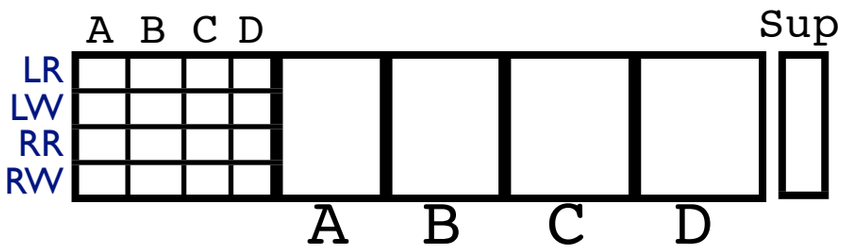
```
BeginRegion
Wr A
EndRegion
BeginRegion
Wr C
```

CPU 2's Code

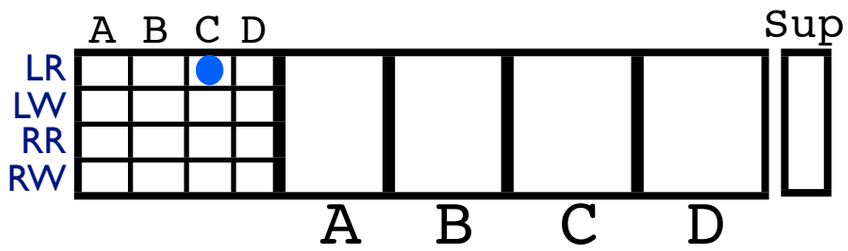
```
BeginRegion
Rd C
```

Putting It Together

CPU 1's Cache



CPU 2's Cache



CPU 1's Code

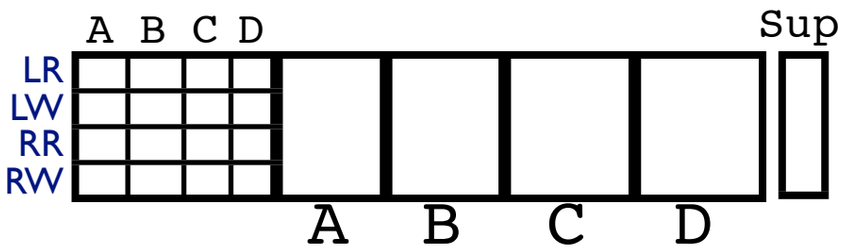
```
BeginRegion
Wr A
EndRegion
BeginRegion
Wr C
```

CPU 2's Code

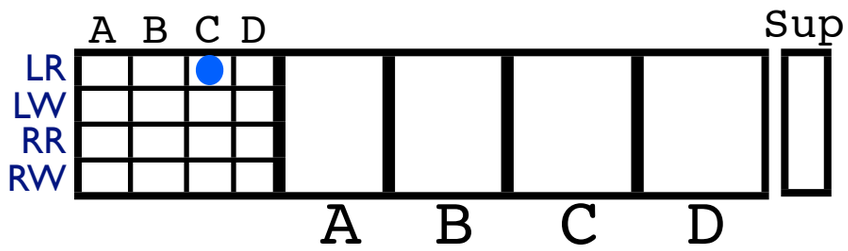
```
BeginRegion
Rd C
```

Putting It Together

CPU 1's Cache



CPU 2's Cache



CPU 1's Code

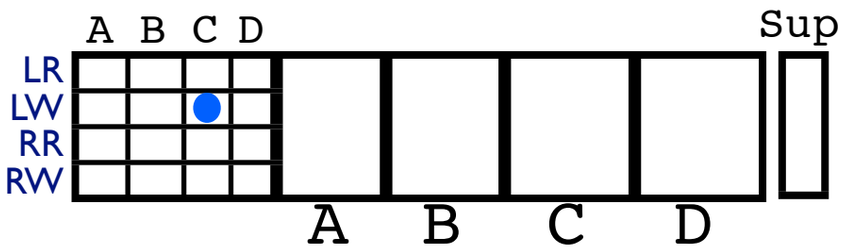
```
BeginRegion
Wr A
EndRegion
BeginRegion
Wr C
```

CPU 2's Code

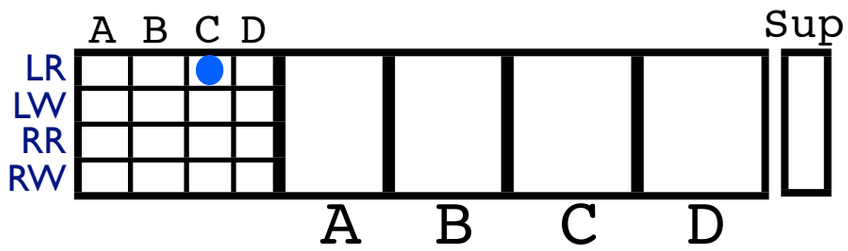
```
BeginRegion
Rd C
```

Putting It Together

CPU 1's Cache



CPU 2's Cache



CPU 1's Code

```
BeginRegion
Wr A
EndRegion
BeginRegion
Wr C
```

CPU 2's Code

```
BeginRegion
Rd C
```

Putting It Together

CPU 1's Code

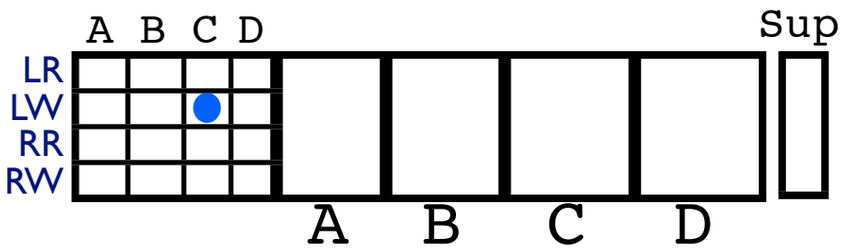
```
BeginRegion
Wr A
EndRegion
BeginRegion
Wr C
```

CPU 2's Code

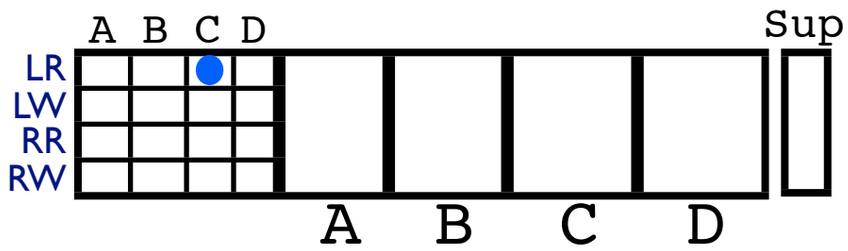
```
BeginRegion
Rd C
```

Putting It Together

CPU 1's Cache



CPU 2's Cache



CPU 1's Code

```
BeginRegion
Wr A
EndRegion
BeginRegion
Wr C
```

CPU 2's Code

```
BeginRegion
Rd C
```

Putting It Together

CPU 1's Code

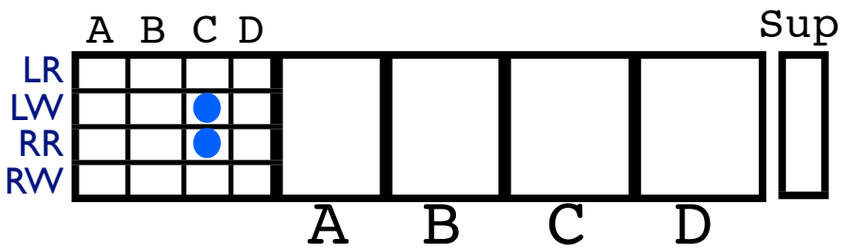
```
BeginRegion
Wr A
EndRegion
BeginRegion
Wr C
```

CPU 2's Code

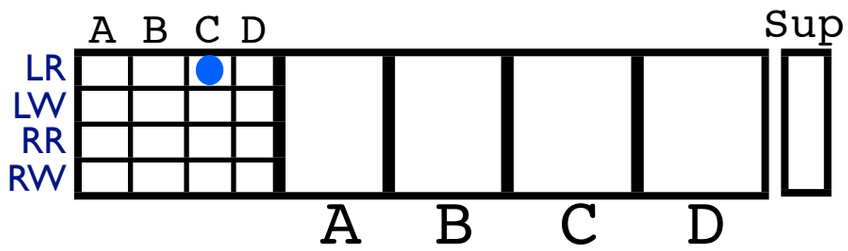
```
BeginRegion
Rd C
```

Putting It Together

CPU 1's Cache



CPU 2's Cache



CPU 1's Code

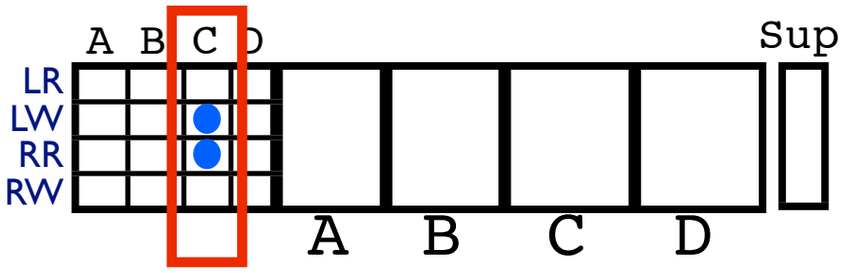
```
BeginRegion
Wr A
EndRegion
BeginRegion
Wr C
```

CPU 2's Code

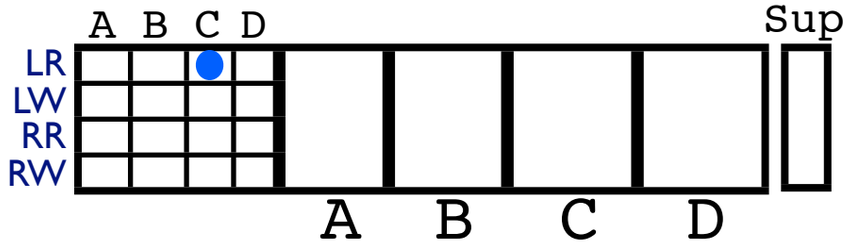
```
BeginRegion
Rd C
```

Putting It Together

CPU 1's Cache



CPU 2's Cache



CPU 1's Code

```
BeginRegion
Wr A
EndRegion
BeginRegion
Wr C
```

Exception!

CPU 2's Code

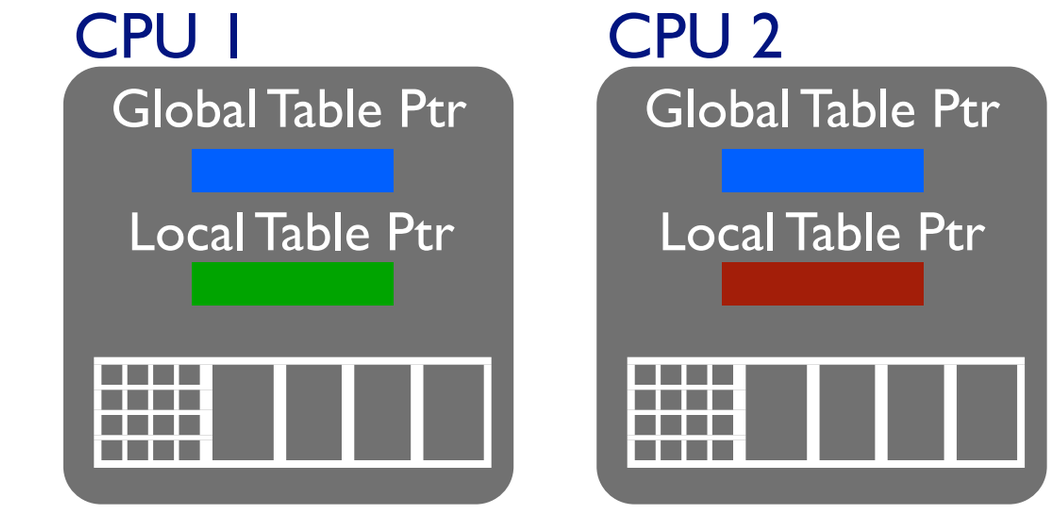
```
BeginRegion
Rd C
```

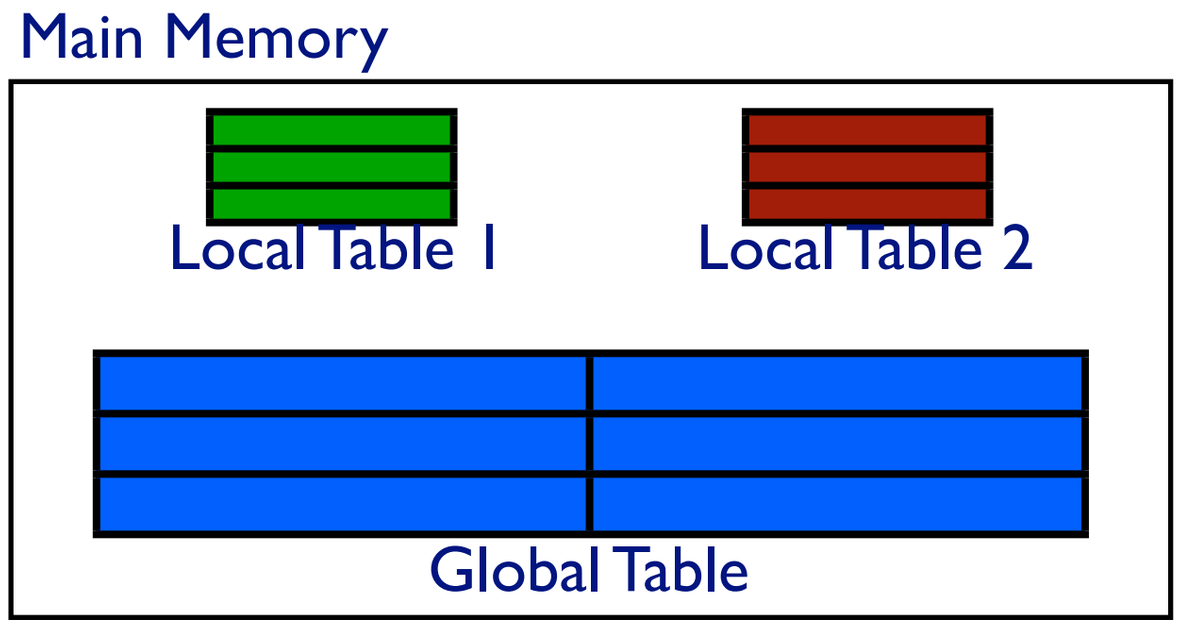
Out-Of-Cache Operation

Per-thread local table tracks evicted accessed addresses

Per-process global table stores evicted lines' access bits

EoR messages for regions with evictions are expensive





sampa

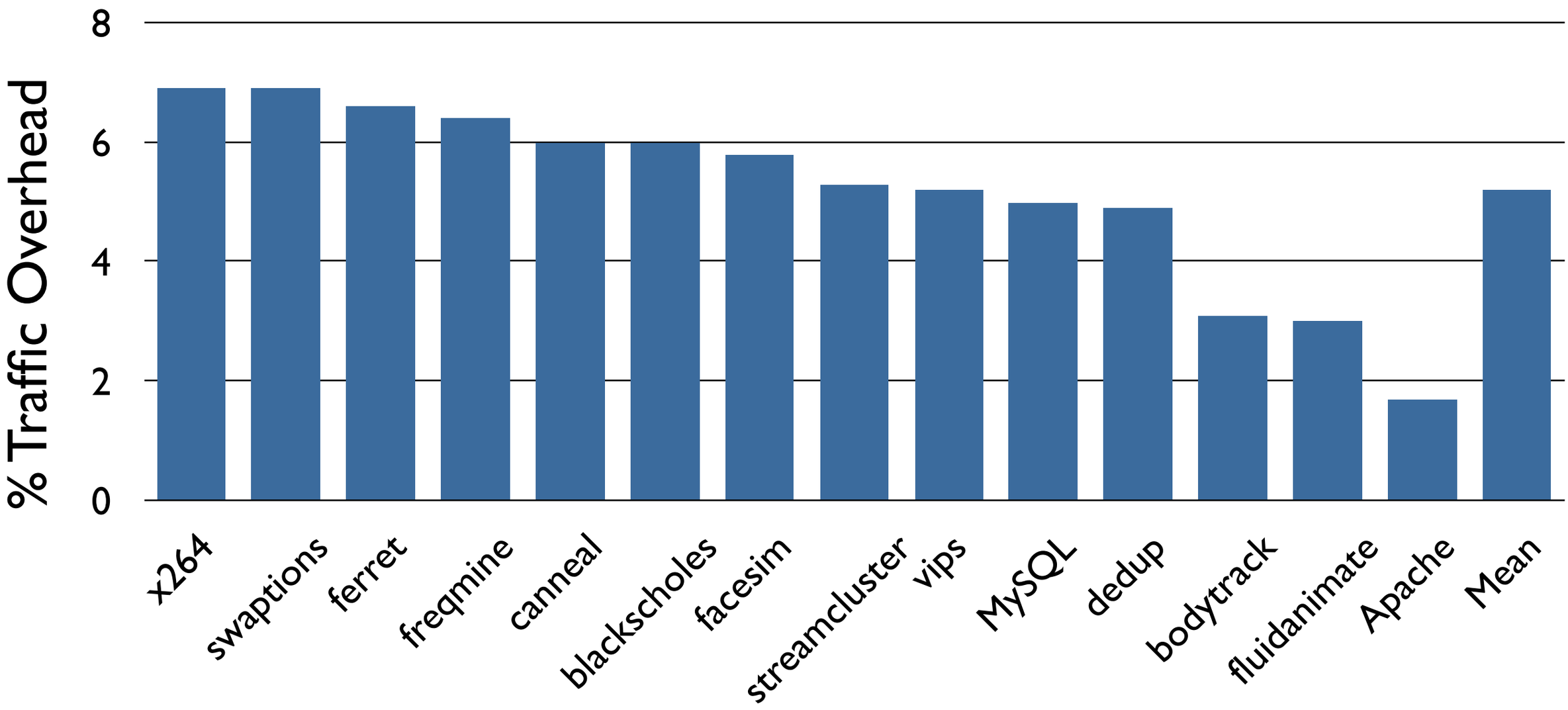
Evaluation

Protocol verified with Zing model checker

Simulator built using SESC and Pin

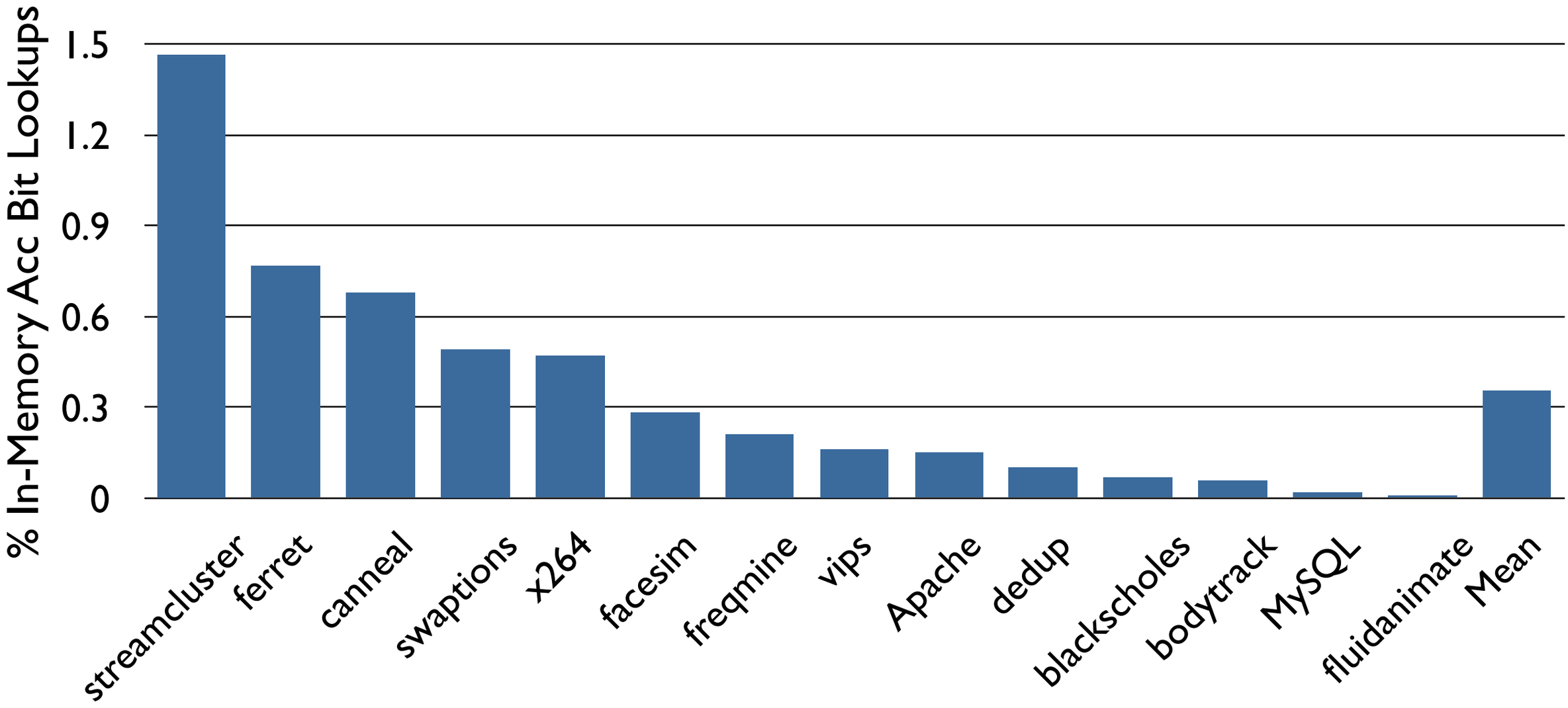
Evaluated using PARSEC, MySQL and Apache

Overheads



~5% traffic overhead on average

Performance Impact



Costly access bit lookups are very infrequent - 1.5% in the worst case

Conflict Exceptions

When a data-race occurs, throw an exception

Simplified language specifications

Easier to debug data races

Limit damage caused by race bugs

Also In The Paper!

Programming Model suitability analysis

Formal proof that exception free executions are SC

More in depth performance characterization

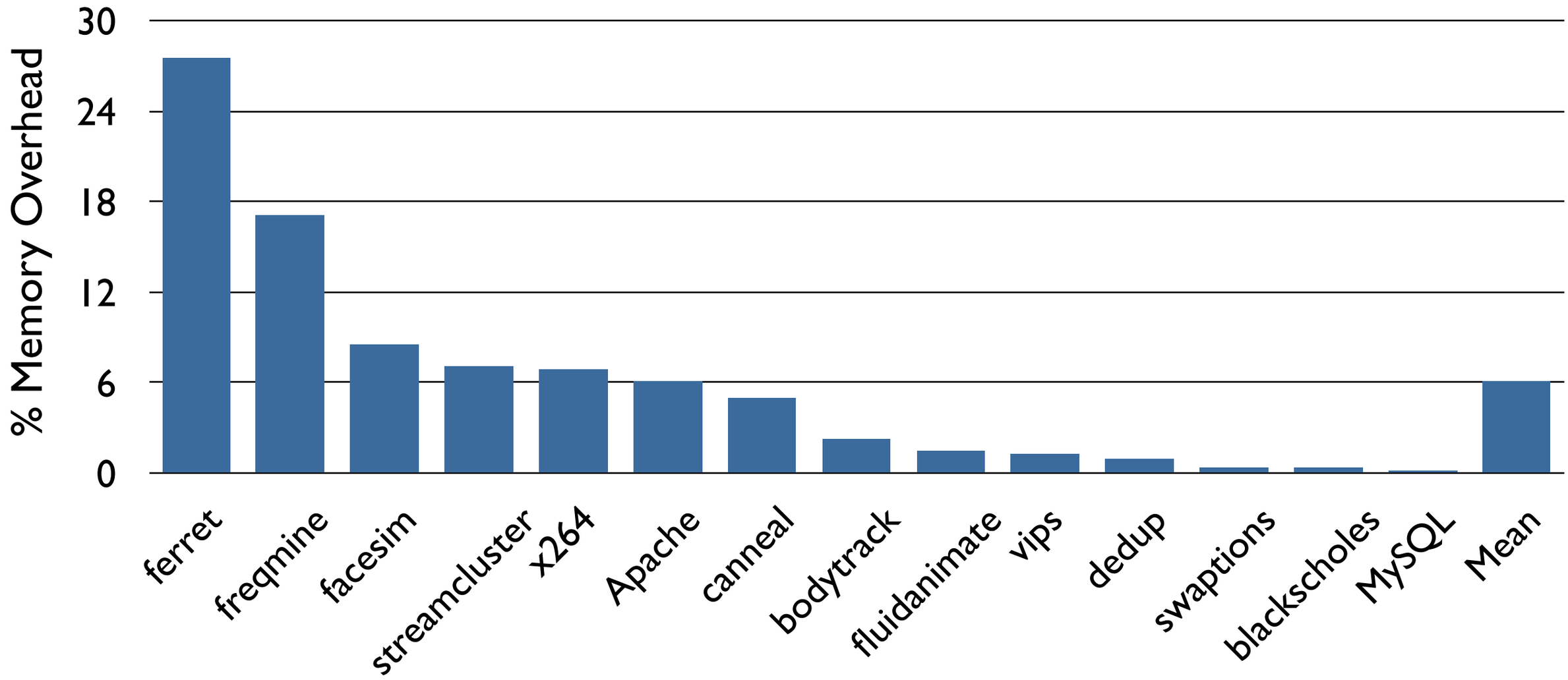
Further protocol implementation details

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

Brandon Lucia, Luis Ceze, Karin Strauss, Shaz Qadeer and Hans-J. Boehm

Microsoft Research

Memory Overhead



Suitability