RACER: TSO CONSISTENCY VIA RACE DETECTION

Alberto Ros[1] Stefanos Kaxiras[2]

[1]Universidad de Murcia
aros@ditec.um.es

[2]Uppsala University
stefanos.kaxiras@it.uu.se

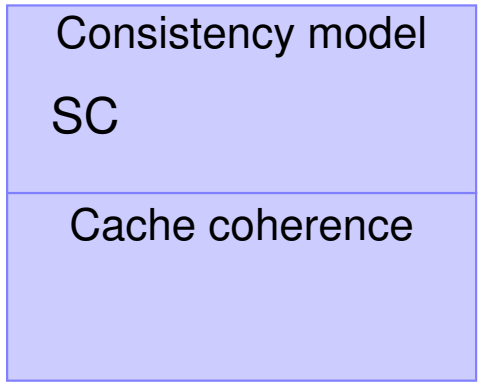
Nov 28th, 2016

Outline

Outline

Cache coherence (SC)

- Cache coherence problem studied for several decades
- Cache coherence serves as a black box to support strict consistency models: e.g., Sequential Consistency (SC)



Cache coherence (SC)

- Cache coherence problem studied for several decades
- Cache coherence serves as a black box to support strict consistency models: e.g., Sequential Consistency (SC)
  - Single-writer-multiple-readers (SWMR) invariant
  - Invalidation/update of the copies on every write
  - Large amount of traffic $\Rightarrow$ increases energy consumption

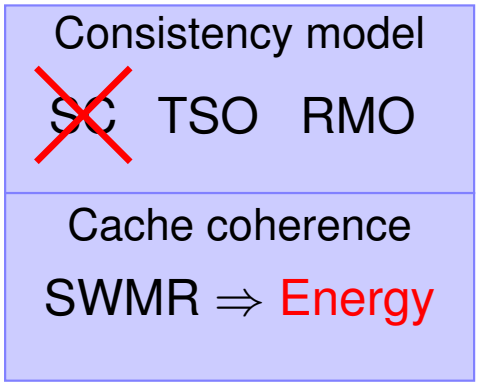
Consistency model
SC
Cache coherence
SWMR $\Rightarrow$ Energy

Cache coherence (SC)

- Cache coherence problem studied for several decades
- Cache coherence serves as a black box to support strict consistency models: e.g., Sequential Consistency (SC)
  - Single-writer-multiple-readers (SWMR) invariant
  - Invalidation/update of the copies on every write
  - Large amount of traffic $\Rightarrow$ increases energy consumption

Observation 1

Most processors offer consistency models weaker than SC



Cache coherence (SC)

- Cache coherence problem studied for several decades
- Cache coherence serves as a black box to support strict consistency models: e.g., Sequential Consistency (SC)
  - Single-writer-multiple-readers (SWMR) invariant
  - Invalidation/update of the copies on every write
  - Large amount of traffic $\Rightarrow$ increases energy consumption

Observation 1

Most processors offer consistency models weaker than SC

- Why implement protocols that provide more functionality than necessary?

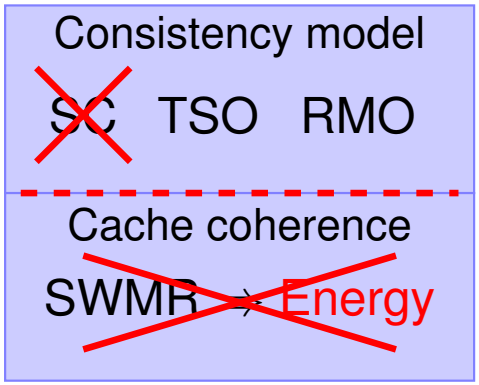
Cache coherence (SC)

- Cache coherence problem studied for several decades
- Cache coherence serves as a black box to support strict consistency models: e.g., Sequential Consistency (SC)
  - Single-writer-multiple-readers (SWMR) invariant
  - Invalidation/update of the copies on every write
  - Large amount of traffic $\Rightarrow$ increases energy consumption

Observation 1

Most processors offer consistency models weaker than SC

- Why implement protocols that provide more functionality than necessary?
- Protocol as a black box?
  - Break the layer between the consistency model and the coherence protocol!



CACHE COHERENCE (SC-FOR-DRF)

- Simple cache coherence: VIPS-M *[Ros & Kaxiras, PACT'12]*
  - Strictly request-response ⇒ Simple
  - Allows virtual caches without reverse translation ⇒ Efficient
  - Coherence distributed across cores ⇒ Scalable
  - No directory ⇒ Simple and scalable

Cache coherence (SC-for-DRF)

- Simple cache coherence: VIPS-M *[Ros & Kaxiras, PACT'12]*
  - Strictly request-response ⇒ Simple
  - Allows virtual caches without reverse translation ⇒ Efficient
  - Coherence distributed across cores ⇒ Scalable
  - No directory ⇒ Simple and scalable
- How? Synchronization exposed to the protocol

Example of DRF code

`X = 1;`	`WAIT(cond);`
`SIGNAL(cond);`	`$r1 = X;`

CACHE COHERENCE (SC-FOR-DRF)

- Simple cache coherence: VIPS-M *[Ros & Kaxiras, PACT'12]*
  - Strictly request-response ⇒ Simple
  - Allows virtual caches without reverse translation ⇒ Efficient
  - Coherence distributed across cores ⇒ Scalable
  - No directory ⇒ Simple and scalable
- How? Synchronization exposed to the protocol
- Release: SELF-DOWNGRADE (SD)
  - ⇒ Write-through dirty blocks

EXAMPLE OF DRF CODE

SD

```
X = 1;          | WAIT(cond);
SIGNAL(cond);   | $r1 = X;
```

CACHE COHERENCE (SC-FOR-DRF)

- Simple cache coherence: VIPS-M *[Ros & Kaxiras, PACT'12]*
  - Strictly request-response ⇒ Simple
  - Allows virtual caches without reverse translation ⇒ Efficient
  - Coherence distributed across cores ⇒ Scalable
  - No directory ⇒ Simple and scalable
- How? Synchronization exposed to the protocol
- Release: SELF-DOWNGRADE (SD)
  ⇒ Write-through dirty blocks
- Acquire: SELF-INVALIDATION (SI)
  ⇒ Empty the cache

EXAMPLE OF DRF CODE

X = 1;	WAIT(cond);
SD SIGNAL(cond);	$r1 = X; SI

CACHE COHERENCE (SC-FOR-DRF)

- Simple cache coherence: VIPS-M *[Ros & Kaxiras, PACT'12]*
  - Strictly request-response ⇒ Simple
  - Allows virtual caches without reverse translation ⇒ Efficient
  - Coherence distributed across cores ⇒ Scalable
  - No directory ⇒ Simple and scalable
- How? Synchronization exposed to the protocol
- Release: SELF-DOWNGRADE (SD)
  ⇒ Write-through dirty blocks
- Acquire: SELF-INVALIDATION (SI)
  ⇒ Empty the cache

EXAMPLE OF DRF CODE

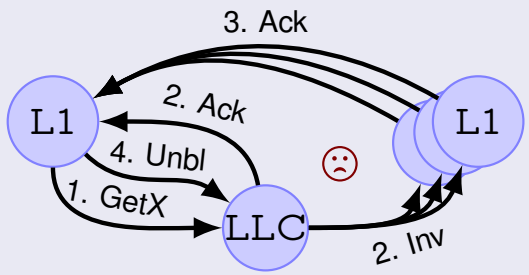
```
SD   X = 1;              |  WAIT(cond);   SI
     SIGNAL(cond);       |  $r1 = X;
```

OBSERVATION 2

SI & SD are conservatively performed because of static synchronization even if there is no actual value propagation between cores

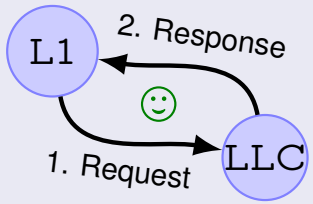
SC VERSUS SC-FOR-DRF COHERENCE

SC



☺ Works for all codes

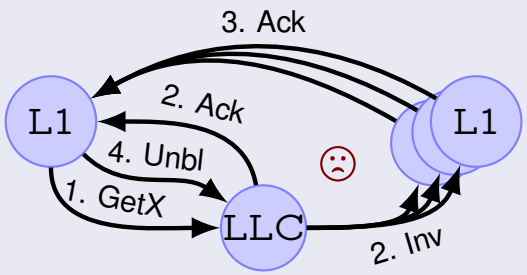
SC-FOR-DRF



☹ Only works for DRF codes

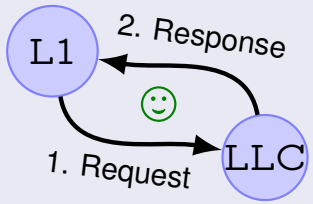
SC versus SC-for-DRF coherence

SC



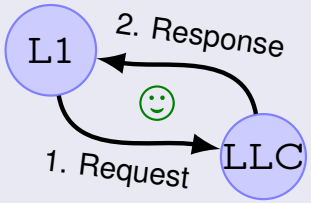
☺ Works for all codes

SC-for-DRF



☹ Only works for DRF codes

Racer (Total-Store-Order – TSO)



☺ Works for all (TSO) codes

First efficient, request-response protocol for all codes

TSO MEMORY CONSISTENCY

TSO RULES

- load→load
- store→store
- load→store

TSO MEMORY CONSISTENCY

TSO RULES

- load→load
- store→store
- load→store

- Is it always necessary to ensure these rules?

TSO MEMORY CONSISTENCY

TSO RULES

- load→load
- store→store
- load→store

- Is it always necessary to ensure these rules?

CODE EXAMPLE

```
/* Initially X, Y = 0 */

X = 1;          |  $r1 = Y;
Y = 1;          |  $r2 = X;

/* $r1 == 1 and $r2 == 0 not allowed */
```

TSO MEMORY CONSISTENCY

TSO RULES

- load→load
- store→store
- load→store

- Is it always necessary to ensure these rules?

CODE EXAMPLE

```
/* Initially X, Y = 0 */

X = 1;          |  $r1 = Y;
Y = 1;          |  $r2 = X;

/* $r1 == 1 and $r2 == 0 not allowed */
```

POSSIBLE EXECUTION

```
Y = 1;          |
X = 1;          |
                |  $r1 = Y;
                |  $r2 = X;

/* (1, 1) allowed */
```

TSO MEMORY CONSISTENCY

TSO RULES

- load→load
- store→store
- load→store

- Is it always necessary to ensure these rules?

CODE EXAMPLE

```
/* Initially X, Y = 0 */

X = 1;          |  $r1 = Y;
Y = 1;          |  $r2 = X;

/* $r1 == 1 and $r2 == 0 not allowed */
```

POSSIBLE EXECUTION

```
Y = 1;          |
                |  $r1 = Y;
                |  $r2 = X;
X = 1;          |

/* (1, 0) not allowed */
```

TSO MEMORY CONSISTENCY

TSO RULES

- load→load
- store→store
- load→store

OBSERVATION 3

Memory operations can be safely reordered as long as they are not observed by other cores

- Is it always necessary to ensure these rules?

CODE EXAMPLE

```
/* Initially X, Y = 0 */

X = 1;          |  $r1 = Y;
Y = 1;          |  $r2 = X;

/* $r1 == 1 and $r2 == 0 not allowed */
```

POSSIBLE EXECUTION

```
Y = 1;          |
                |  $r1 = Y;
                |  $r2 = X;
X = 1;          |

/* (1, 0) not allowed */
```

Outline

RACER AT A GLANCE

- A novel way of supporting TSO consistency (Obs.1)
  - $\Rightarrow$ At the cache coherence protocol level

RACER AT A GLANCE

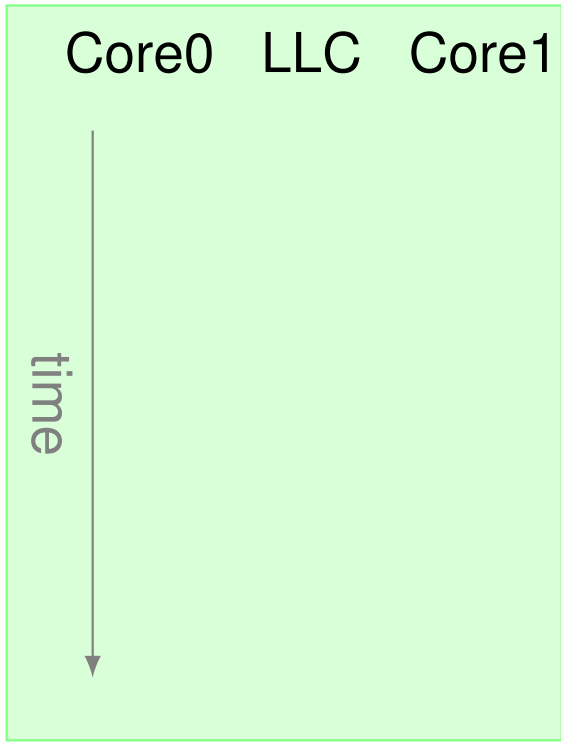
- A novel way of supporting TSO consistency (Obs.1)
  - $\Rightarrow$ At the cache coherence protocol level
- We start with a very simple request-response protocol
  - $\Rightarrow$ Order enforced on SI & SD

RACER AT A GLANCE

- A novel way of supporting TSO consistency (Obs.1)
  - $\Rightarrow$ At the cache coherence protocol level
- We start with a very simple request-response protocol
  - $\Rightarrow$ Order enforced on SI & SD
- When it is necessary to enforce order?
  - $\Rightarrow$ In SC-for-DRF conservatively on synchronization (Obs.2)

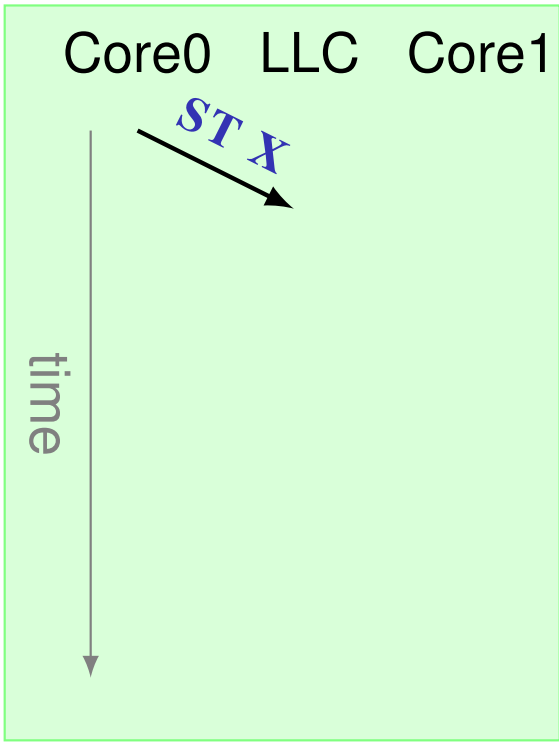
RACER AT A GLANCE

- A novel way of supporting TSO consistency (Obs.1)
  - ⇒ At the cache coherence protocol level
- We start with a very simple request-response protocol
  - ⇒ Order enforced on SI & SD
- When it is necessary to enforce order?
  - ⇒ In SC-for-DRF conservatively on synchronization (Obs.2)
  - ⇒ In RACER only when it is possible to see a reordering (Obs.3)
  - ⇒ On actual (read-after-write) RAW races



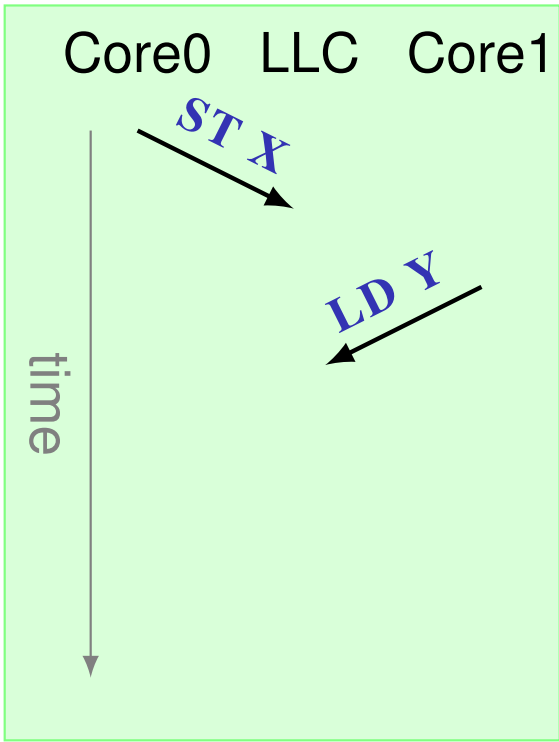
RACER AT A GLANCE

- A novel way of supporting TSO consistency (Obs.1)
  - ⇒ At the cache coherence protocol level
- We start with a very simple request-response protocol
  - ⇒ Order enforced on SI & SD
- When it is necessary to enforce order?
  - ⇒ In SC-for-DRF conservatively on synchronization (Obs.2)
  - ⇒ In RACER only when it is possible to see a reordering (Obs.3)
  - ⇒ On actual (read-after-write) RAW races



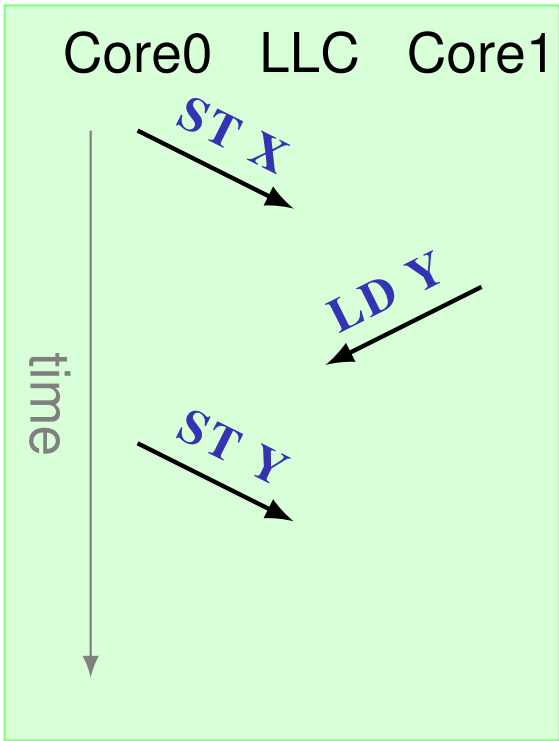
RACER AT A GLANCE

- A novel way of supporting TSO consistency (Obs.1)
  - ⇒ At the cache coherence protocol level
- We start with a very simple request-response protocol
  - ⇒ Order enforced on SI & SD
- When it is necessary to enforce order?
  - ⇒ In SC-for-DRF conservatively on synchronization (Obs.2)
  - ⇒ In RACER only when it is possible to see a reordering (Obs.3)
  - ⇒ On actual (read-after-write) RAW races



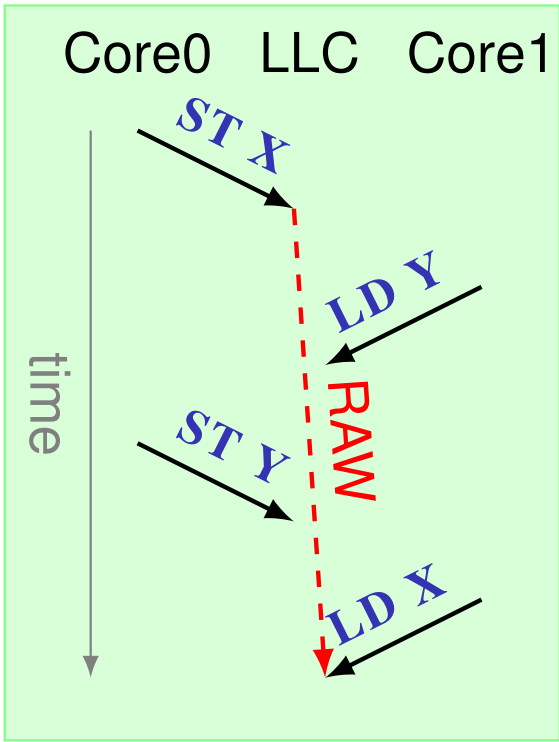
RACER AT A GLANCE

- A novel way of supporting TSO consistency (Obs.1)
  - ⇒ At the cache coherence protocol level
- We start with a very simple request-response protocol
  - ⇒ Order enforced on SI & SD
- When it is necessary to enforce order?
  - ⇒ In SC-for-DRF conservatively on synchronization (Obs.2)
  - ⇒ In RACER only when it is possible to see a reordering (Obs.3)
  - ⇒ On actual (read-after-write) RAW races



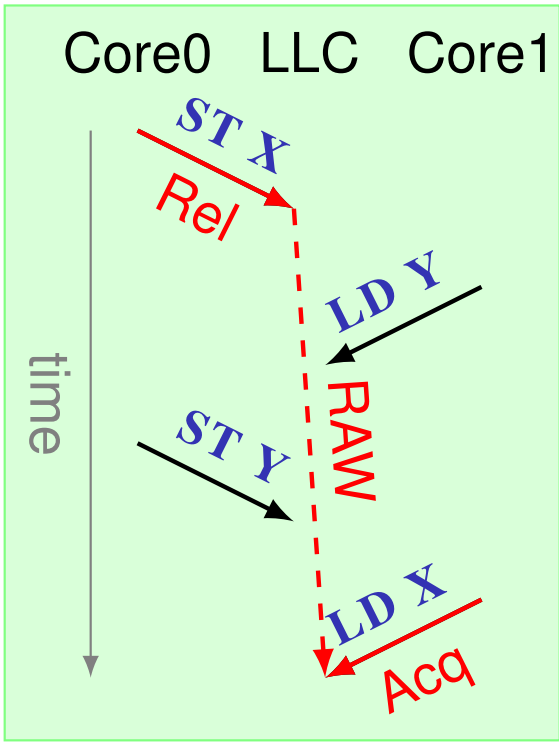
RACER AT A GLANCE

- A novel way of supporting TSO consistency (Obs.1)
  - ⇒ At the cache coherence protocol level
- We start with a very simple request-response protocol
  - ⇒ Order enforced on SI & SD
- When it is necessary to enforce order?
  - ⇒ In SC-for-DRF conservatively on synchronization (Obs.2)
  - ⇒ In RACER only when it is possible to see a reordering (Obs.3)
  - ⇒ On actual (read-after-write) RAW races



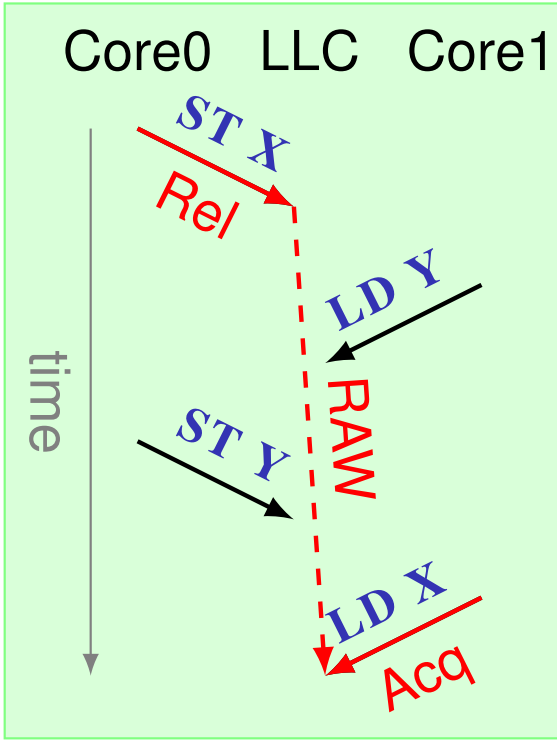
RACER AT A GLANCE

- A novel way of supporting TSO consistency (Obs.1)
  - ⇒ At the cache coherence protocol level
- We start with a very simple request-response protocol
  - ⇒ Order enforced on SI & SD
- When it is necessary to enforce order?
  - ⇒ In SC-for-DRF conservatively on synchronization (Obs.2)
  - ⇒ In RACER only when it is possible to see a reordering (Obs.3)
  - ⇒ On actual (read-after-write) RAW races

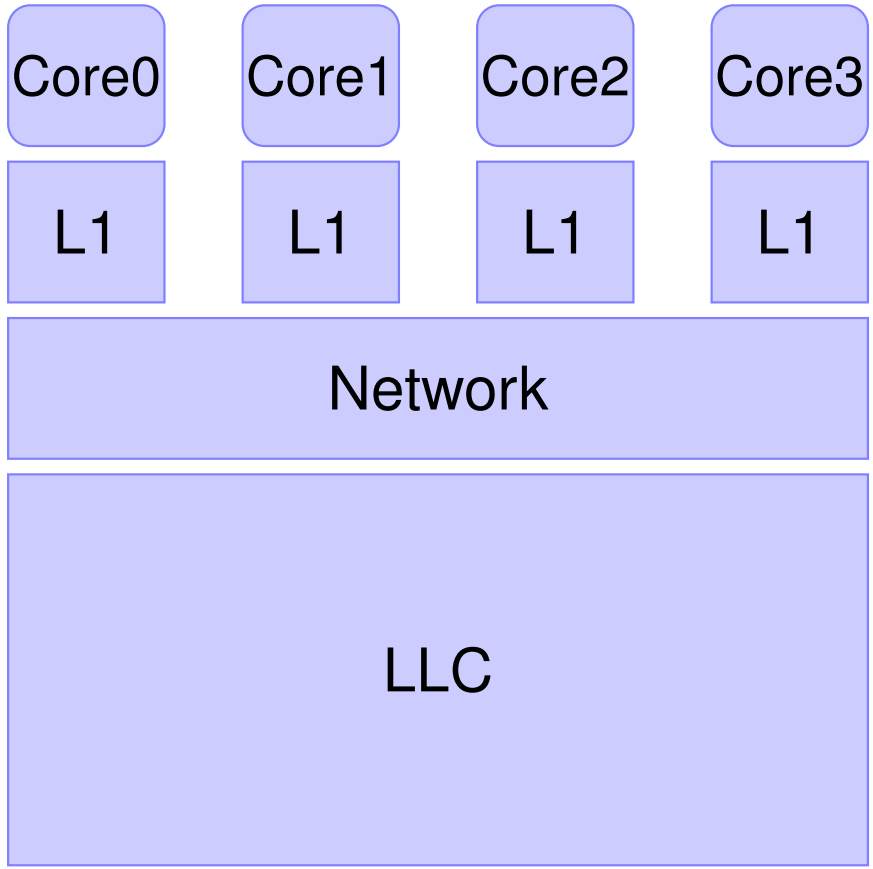


RACER AT A GLANCE

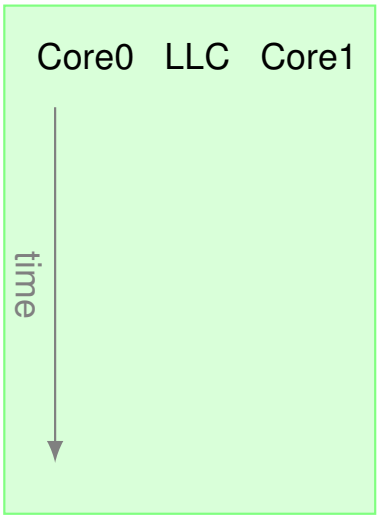
- A novel way of supporting TSO consistency (Obs.1)
  - ⇒ At the cache coherence protocol level
- We start with a very simple request-response protocol
  - ⇒ Order enforced on SI & SD
- When it is necessary to enforce order?
  - ⇒ In SC-for-DRF conservatively on synchronization (Obs.2)
  - ⇒ In RACER only when it is possible to see a reordering (Obs.3)
  - ⇒ On actual (read-after-write) RAW races
- Consistency only enforced for shared data *[Singh et al. ISCA'12]*



BASIC OPERATION



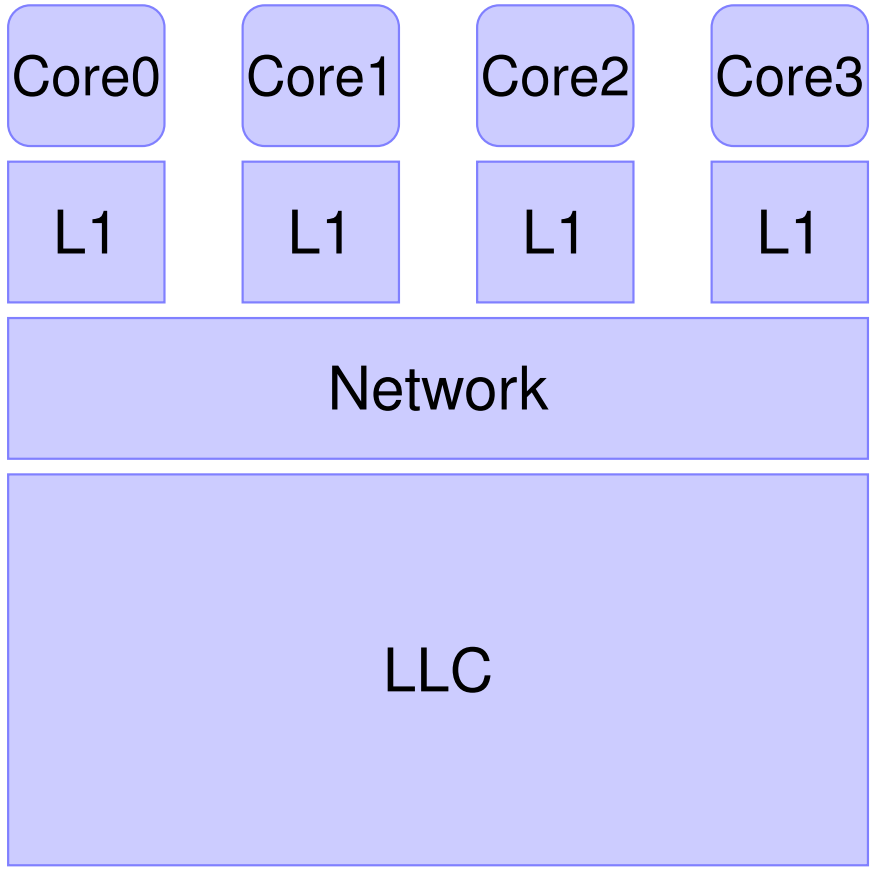
BASIC OPERATION



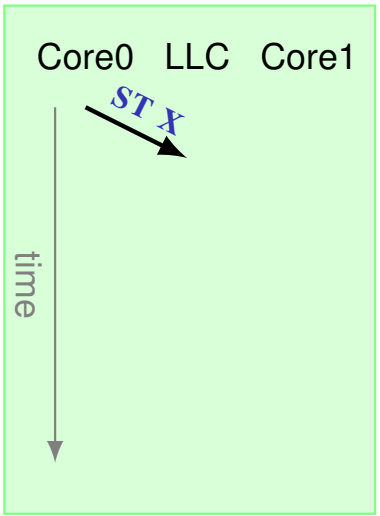
One bloom filter per core

RAWR DETECTOR

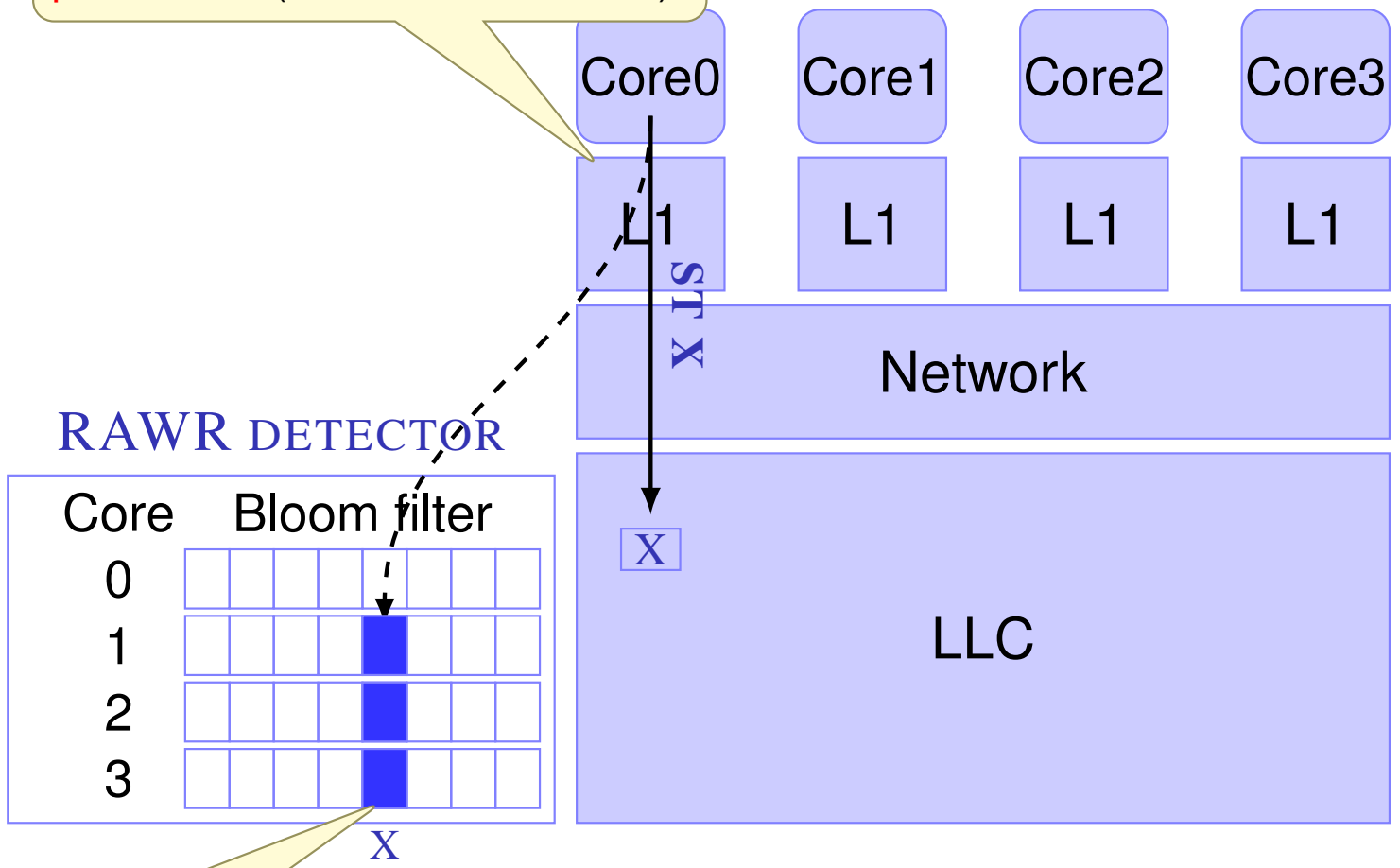
Core	Bloom filter
0	
1	
2	
3	



BASIC OPERATION

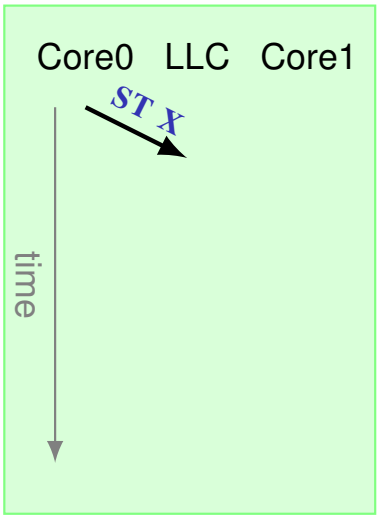


Stores do not require write permission (no SWMR invariant)

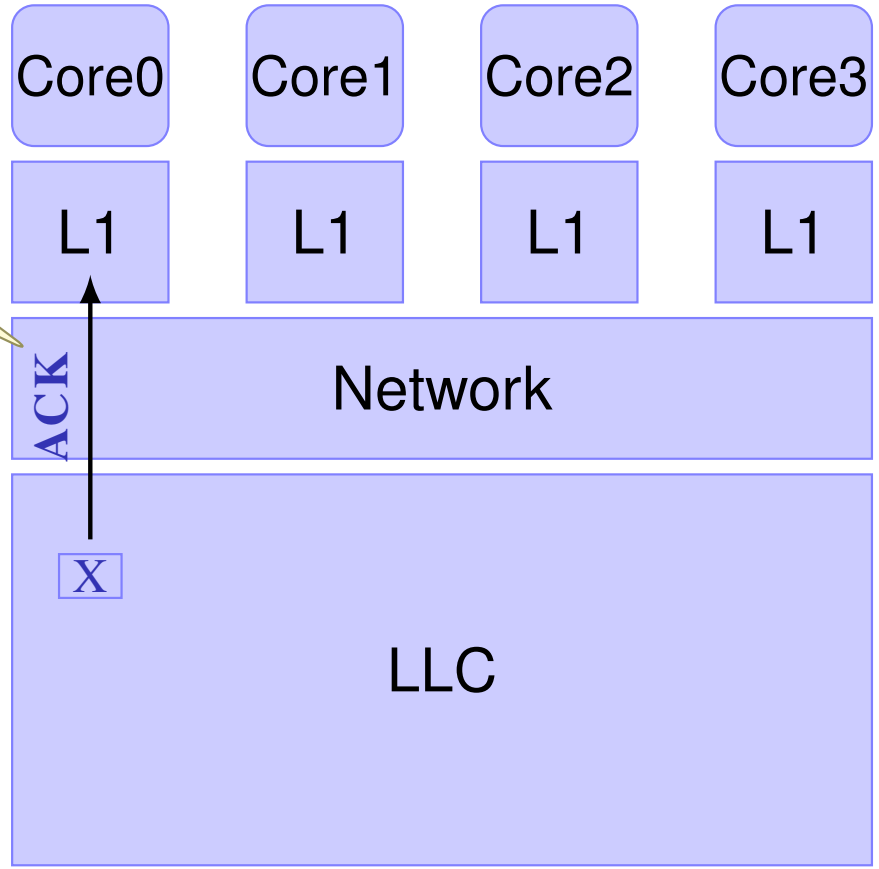


Cores 1, 2, and 3 have not seen the write to x

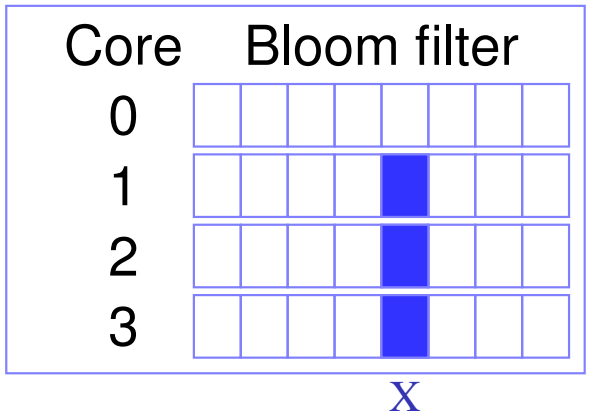
BASIC OPERATION



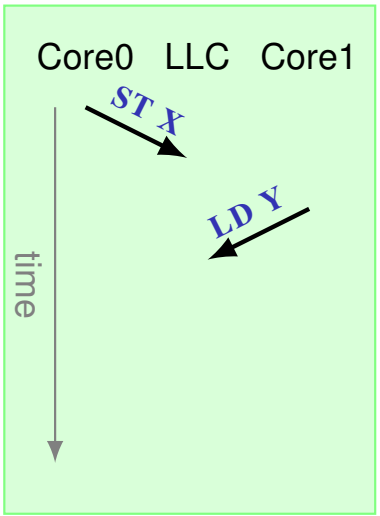
ACK to guarantee order of store operations

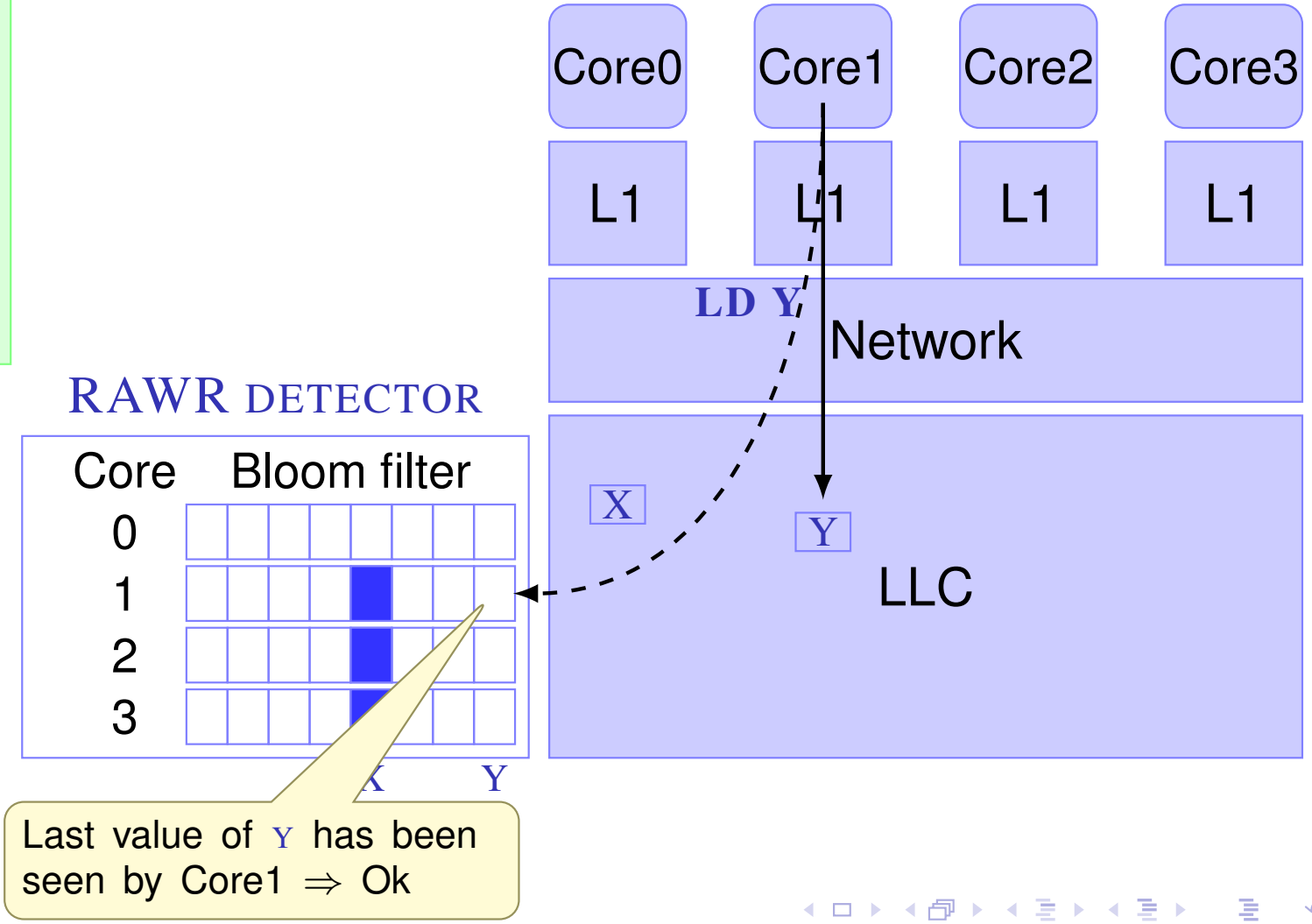


RAWR DETECTOR

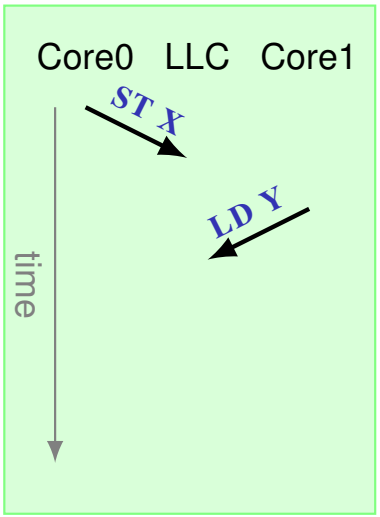


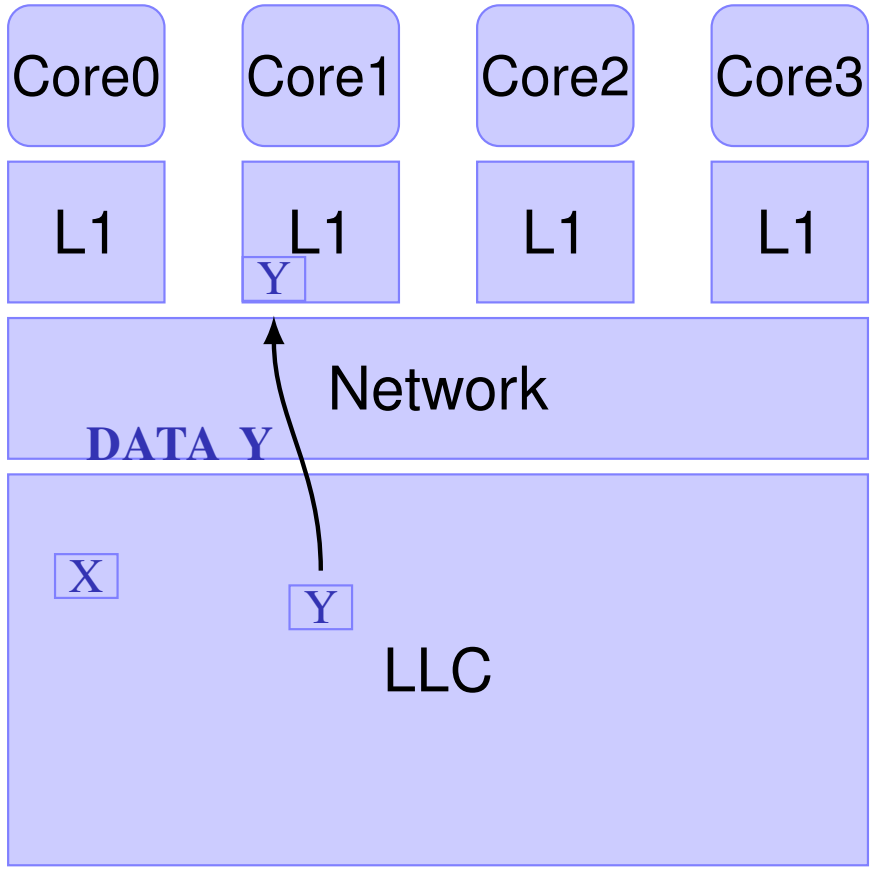
BASIC OPERATION



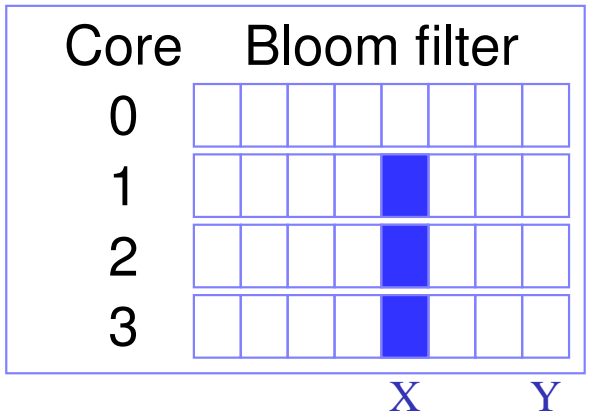


BASIC OPERATION

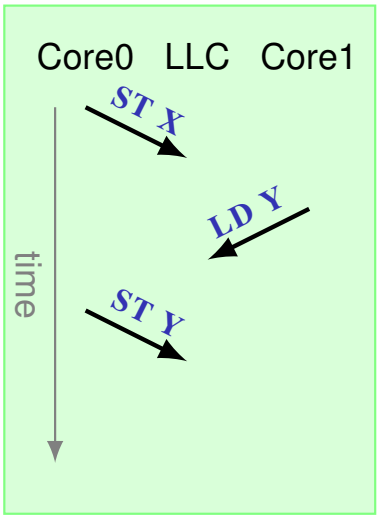


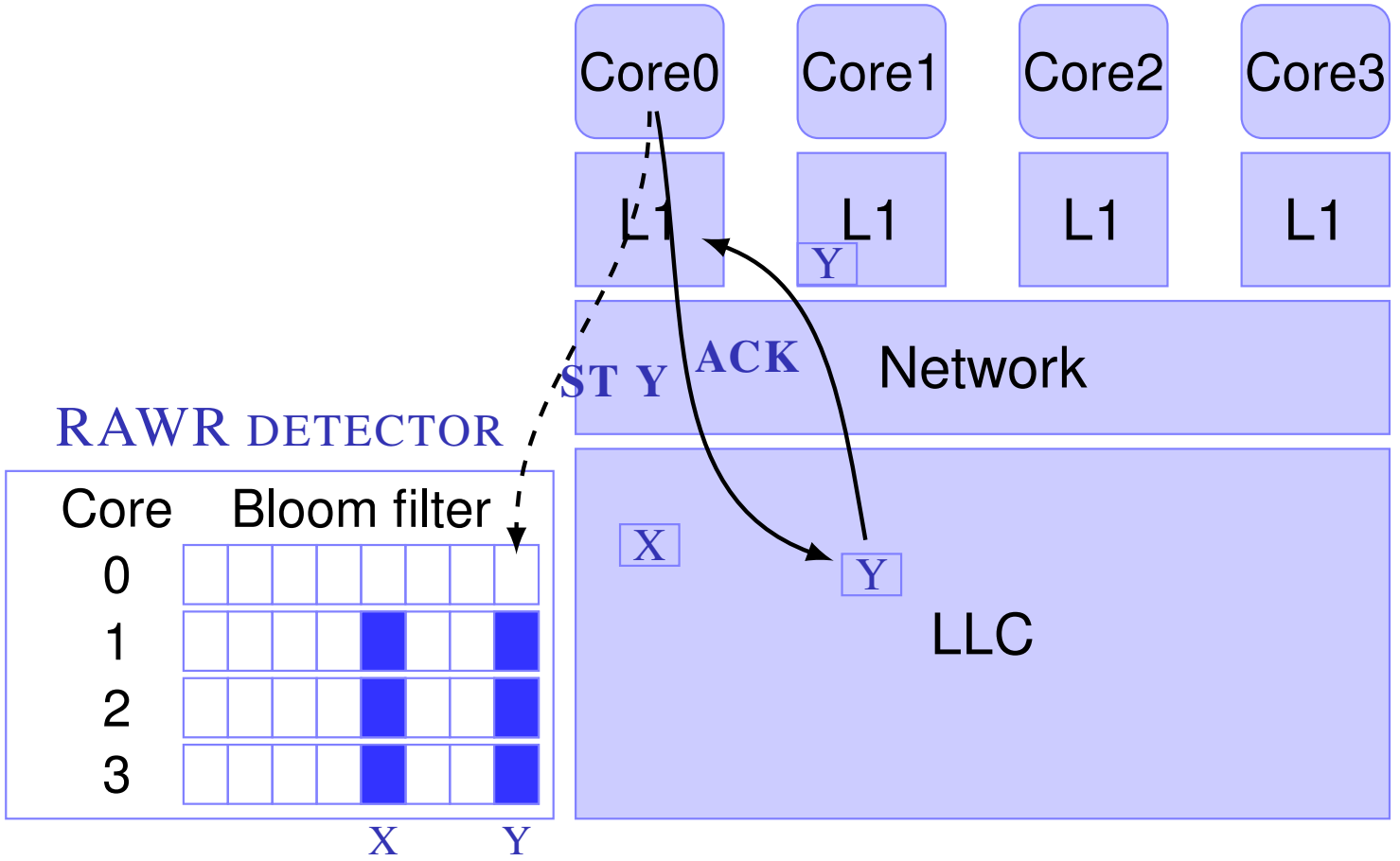


RAWR DETECTOR

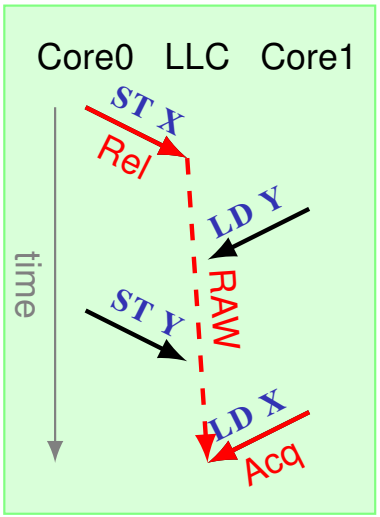


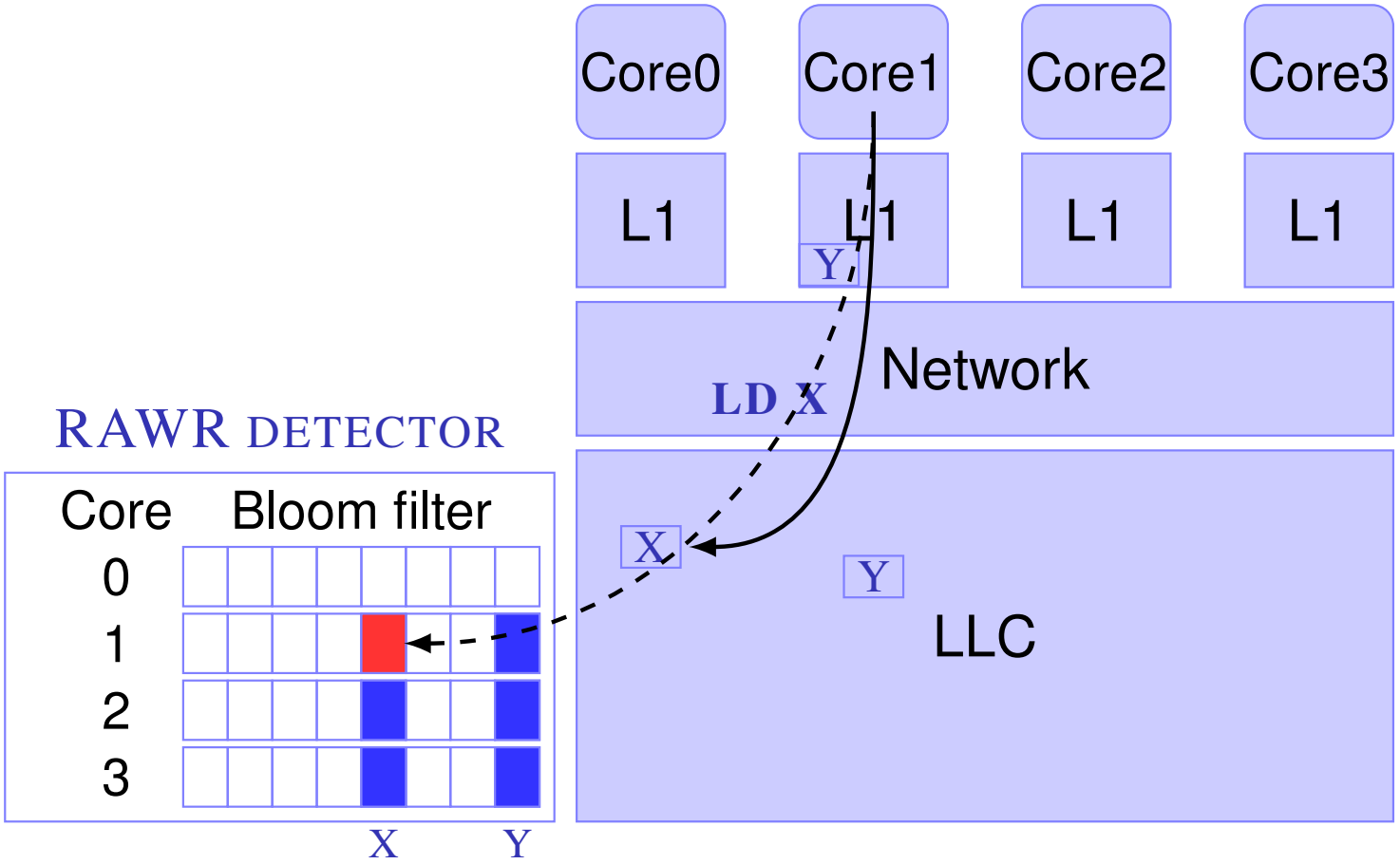
BASIC OPERATION



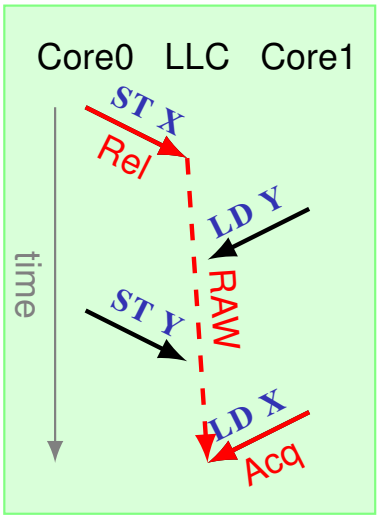


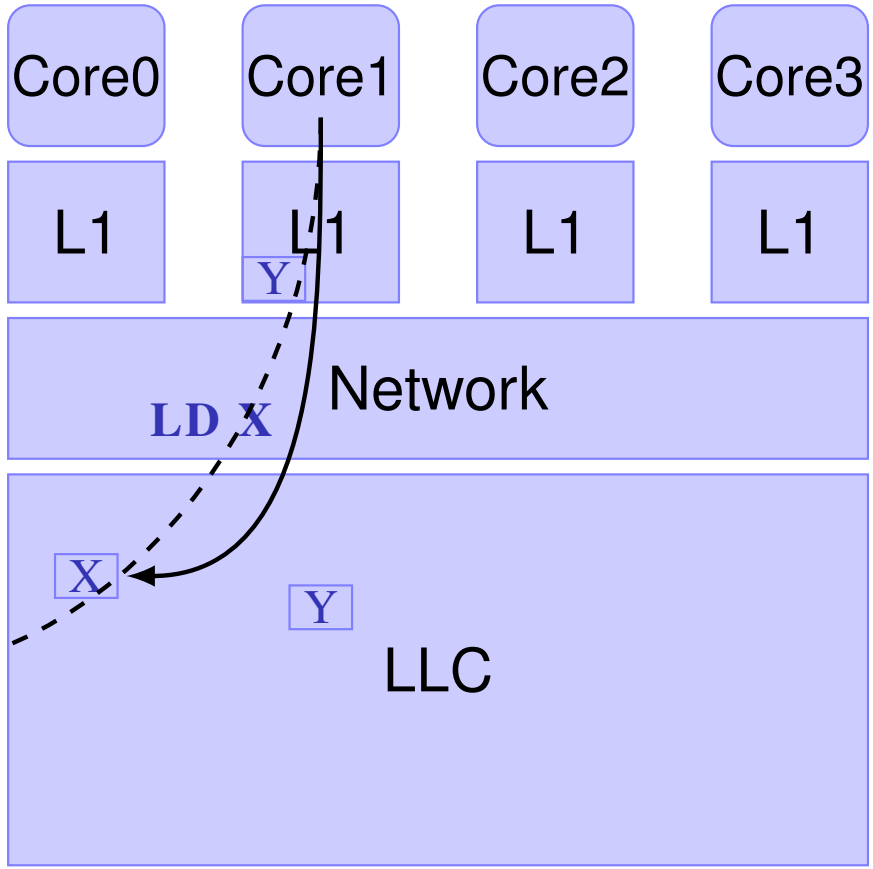
BASIC OPERATION



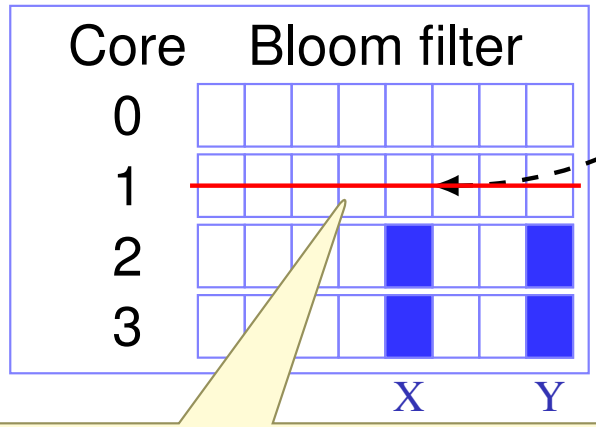


BASIC OPERATION



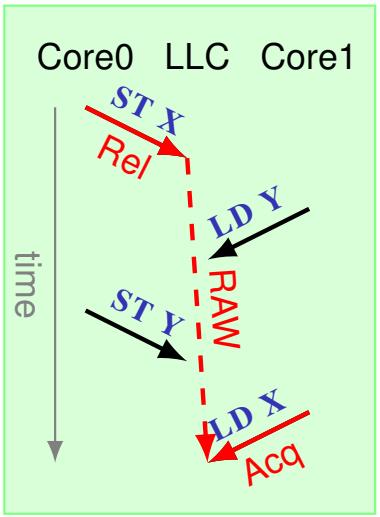


RAWR DETECTOR



Clean Core1 bloom filter.
Filters are naturally cleared when detecting races!

BASIC OPERATION



SELF-INVALIDATE (shared)
Core1 cache content

Core0 Core1 Core2 Core3

L1 L1 L1 L1

X X

Network

DATA X + SI

X

Y

LLC

RAWR DETECTOR

Core	Bloom filter
0	
1	
2	X Y
3	X Y

Efficiency issues of basic operation

- ☺ Basic operation ⇒ Request-response protocol for TSO
- ☹ But it has two efficiency problems

Efficiency issues of basic operation

- ☺ Basic operation ⇒ Request-response protocol for TSO
- ☹ But it has two efficiency problems

1. Write-through ⇒ Traffic, energy
   - Solution: Coalesce, but keep TSO order

Efficiency issues of basic operation

- ☺ Basic operation ⇒ Request-response protocol for TSO
- ☹ But it has two efficiency problems

1. Write-through ⇒ Traffic, energy
   - Solution: Coalesce, but keep TSO order
2. L1 hits cannot detect races ⇒ Starvation
   - Solution: Check-for-race, but efficient

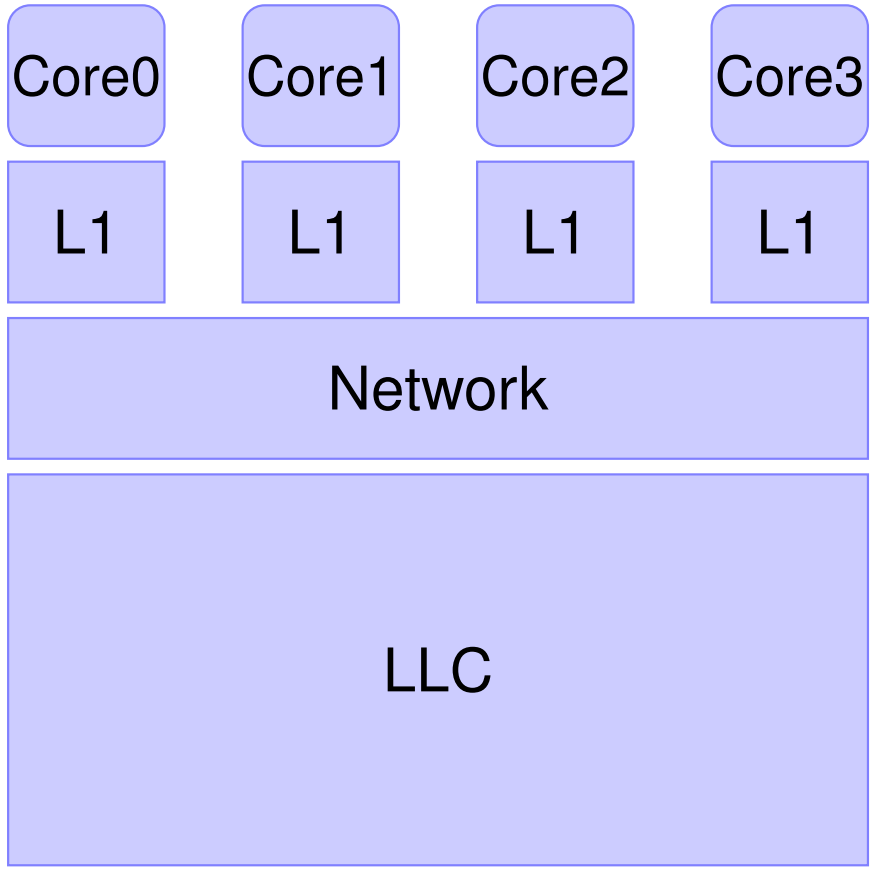
Coalescing and TSO order

- How to coalesce without violating TSO order?
  - Write coalescing violates the `store`→`store` order

COALESCING AND TSO ORDER

- How to coalesce without violating TSO order?
  - Write coalescing violates the `store`→`store` order
- Only a problem if someone sees the reordering (Obs.3)
- Solution: COLLAPSED ORDER
  - ⇒ Allows to coalesce non-consecutive stores
  - ⇒ By not allowing observing reorderings

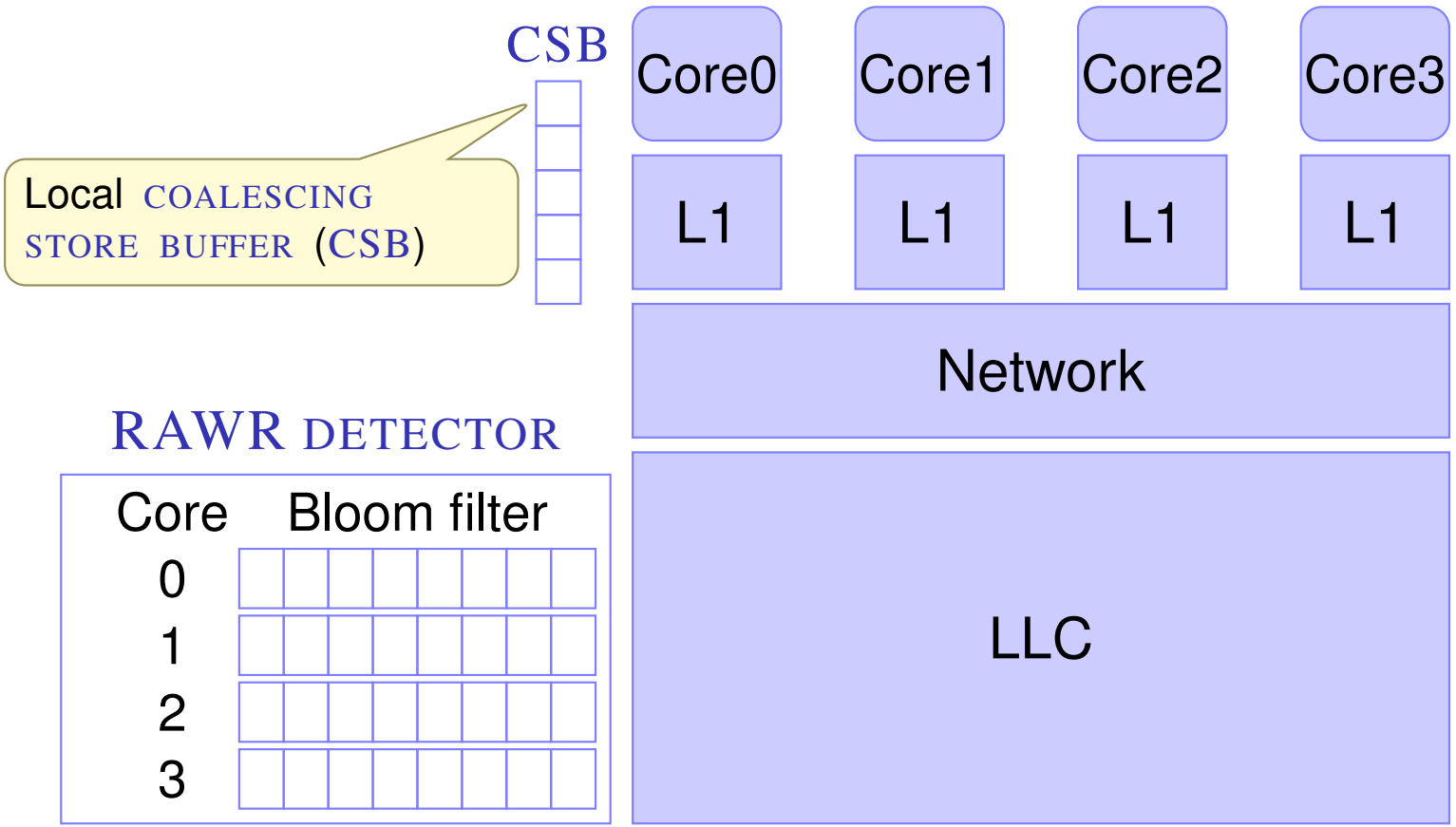
Coalescing and collapsed order



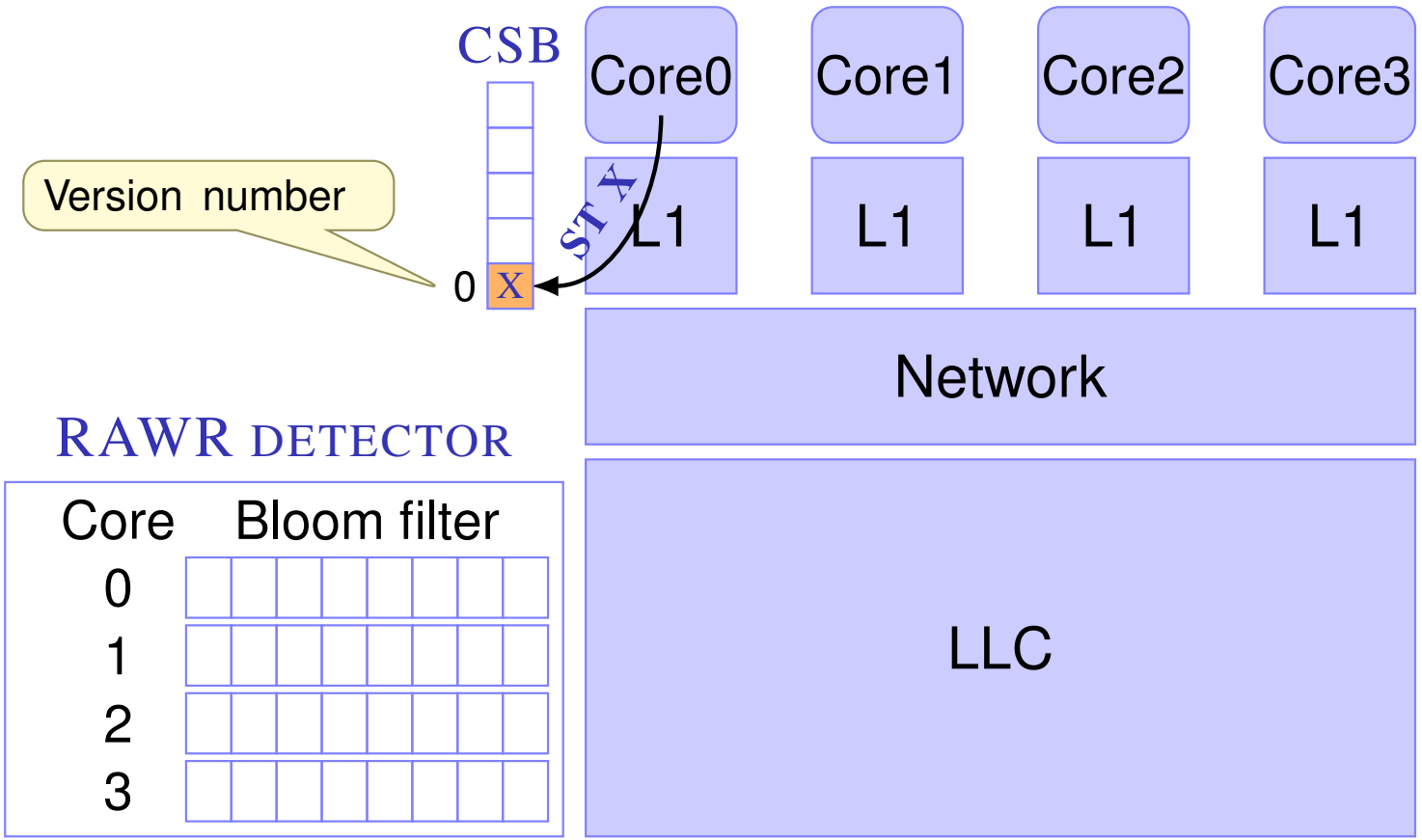
RAWR detector

Core	Bloom filter
0	
1	
2	
3	

COALESCING AND COLLAPSED ORDER

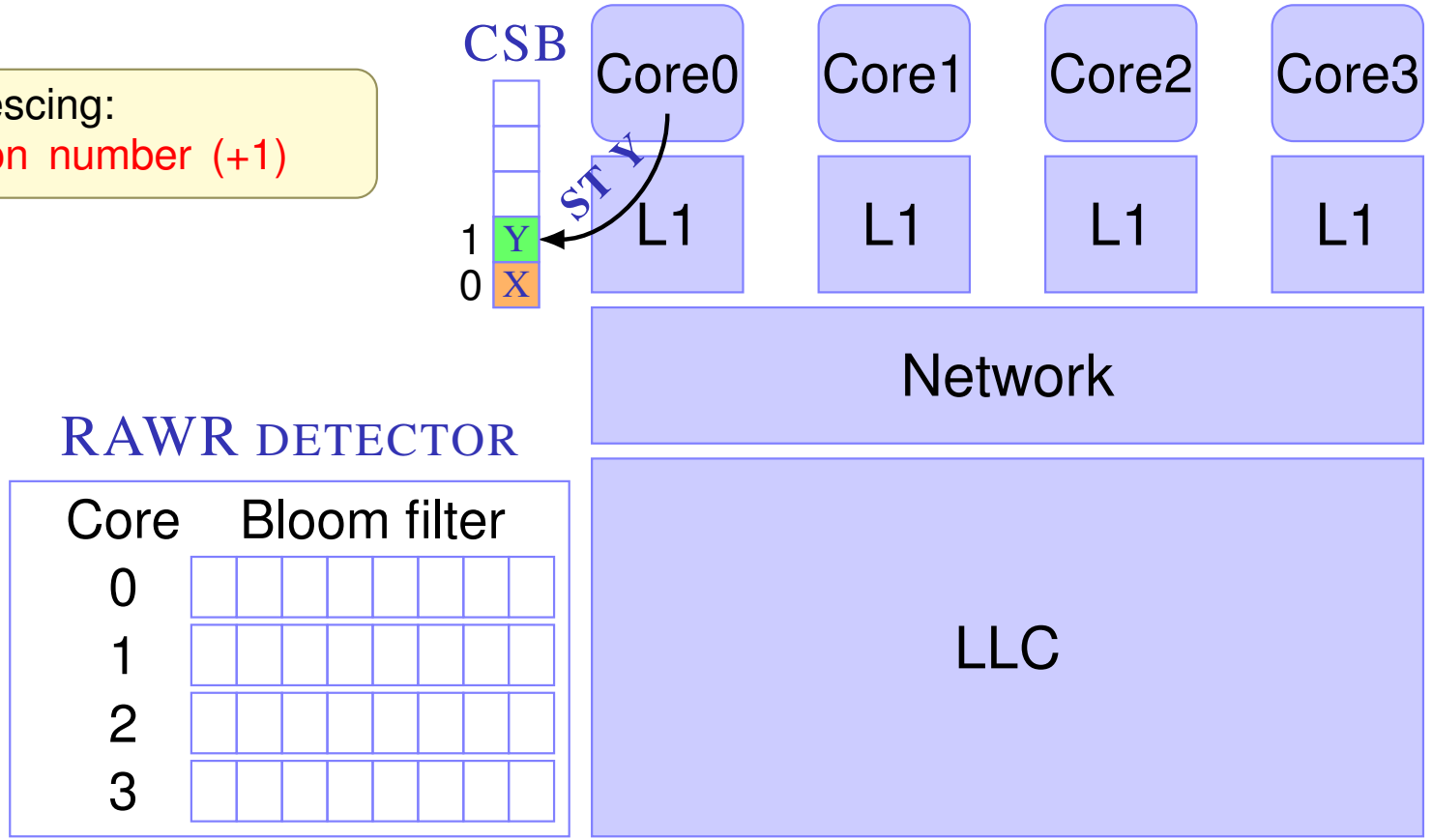


COALESCING AND COLLAPSED ORDER



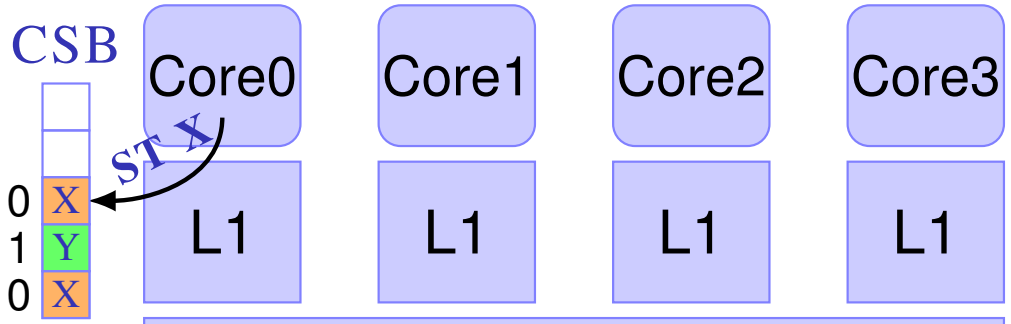
COALESCING AND COLLAPSED ORDER

No coalescing:
⇒ version number (+1)



COALESCING AND COLLAPSED ORDER

Coalescing:
$\Rightarrow$ all version numbers between coalescing writes the same

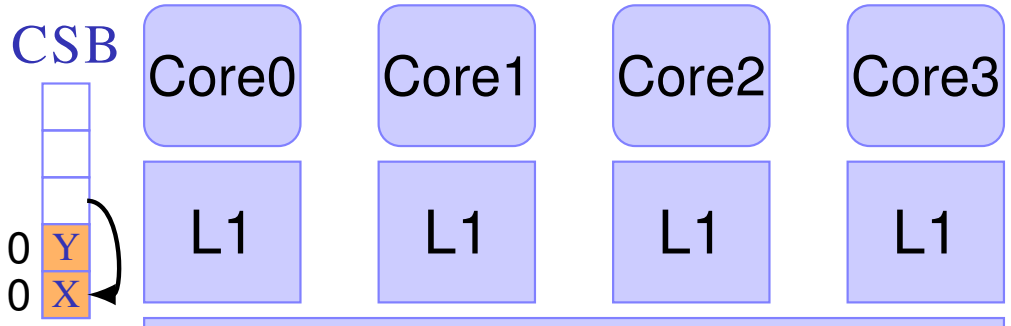


RAWR DETECTOR

Core	Bloom filter
0	
1	
2	
3	

COALESCING AND COLLAPSED ORDER

Coalescing:
$\Rightarrow$ all version numbers between coalescing writes the same



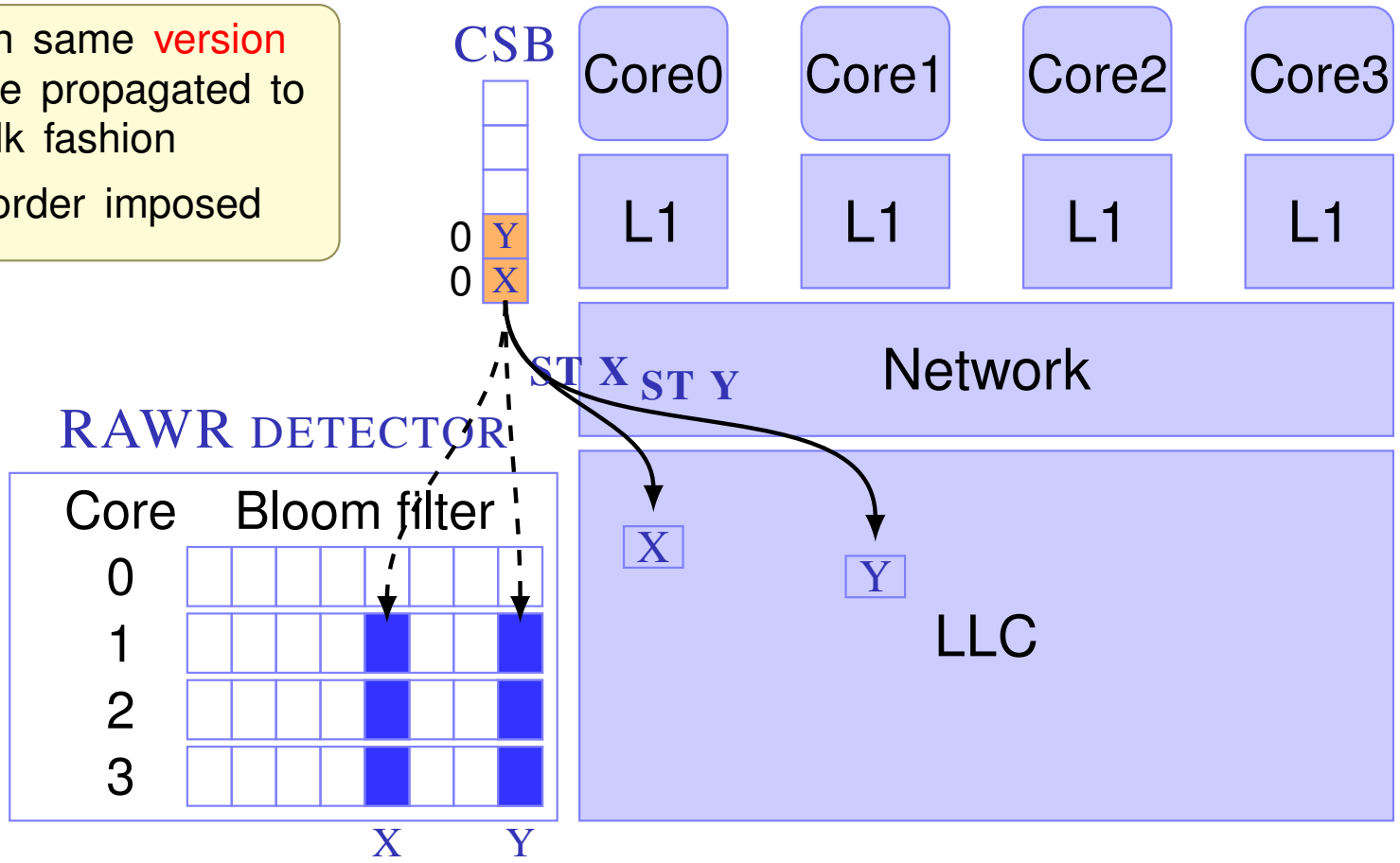
RAWR DETECTOR

Core	Bloom filter
0	
1	
2	
3	

COALESCING AND COLLAPSED ORDER

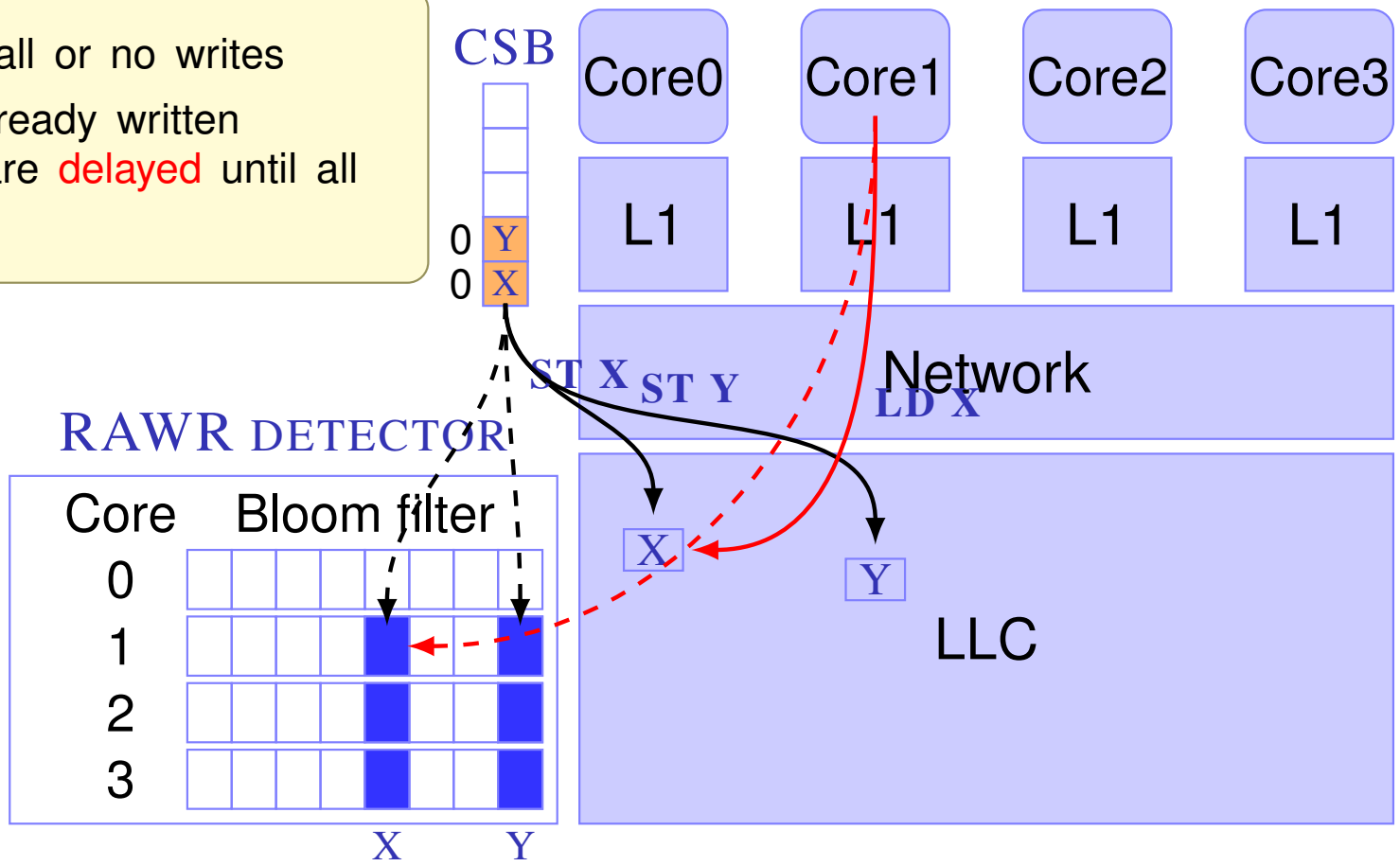
Writes with same version number are propagated to LLC in bulk fashion

$\Rightarrow$ No order imposed



Coalescing and collapsed order

- Reads see all or no writes
- Reads to already written addresses are delayed until all writes finish



CHECK RACE AND RACE PREDICTION

- In RACER, SELF-INVALIDATION is performed on RAW races
  - Cache hits prevent the detection of races $\Rightarrow$ Starvation

CHECK RACE AND RACE PREDICTION

- In RACER, SELF-INVALIDATION is performed on RAW races
  - Cache hits prevent the detection of races $\Rightarrow$ Starvation
- To guarantee progress, it is necessary to check for races even in the case of hits
  - RACER issues a CHECK-RACE request after a timeout

CHECK RACE AND RACE PREDICTION

- In RACER, SELF-INVALIDATION is performed on RAW races
  - Cache hits prevent the detection of races ⇒ Starvation
- To guarantee progress, it is necessary to check for races even in the case of hits
  - RACER issues a CHECK-RACE request after a timeout
- Large timeouts delay observing new values
  - ☹ Slow write propagation

CHECK RACE AND RACE PREDICTION

- In RACER, SELF-INVALIDATION is performed on RAW races
  - Cache hits prevent the detection of races ⇒ Starvation
- To guarantee progress, it is necessary to check for races even in the case of hits
  - RACER issues a CHECK-RACE request after a timeout
- Large timeouts delay observing new values
  - ☹ Slow write propagation
- RACE PREDICTOR to check more frequently racy operations
  - ☺ Fast propagation of writes

TSO GUARANTEES

- Load→Load
  - SI of (shared) cached copies on races
  - Only when the race actually happens

TSO GUARANTEES

- Load→Load
  - SI of (shared) cached copies on races
  - Only when the race actually happens
- Store→Store
  - Writes are constantly SD in COLLAPSED ORDER
  - Store coalescing is allowed

TSO GUARANTEES

- Load→Load
  - SI of (shared) cached copies on races
  - Only when the race actually happens
- Store→Store
  - Writes are constantly SD in COLLAPSED ORDER
  - Store coalescing is allowed
- Load→Store
  - SD performed after all previous loads are resolved

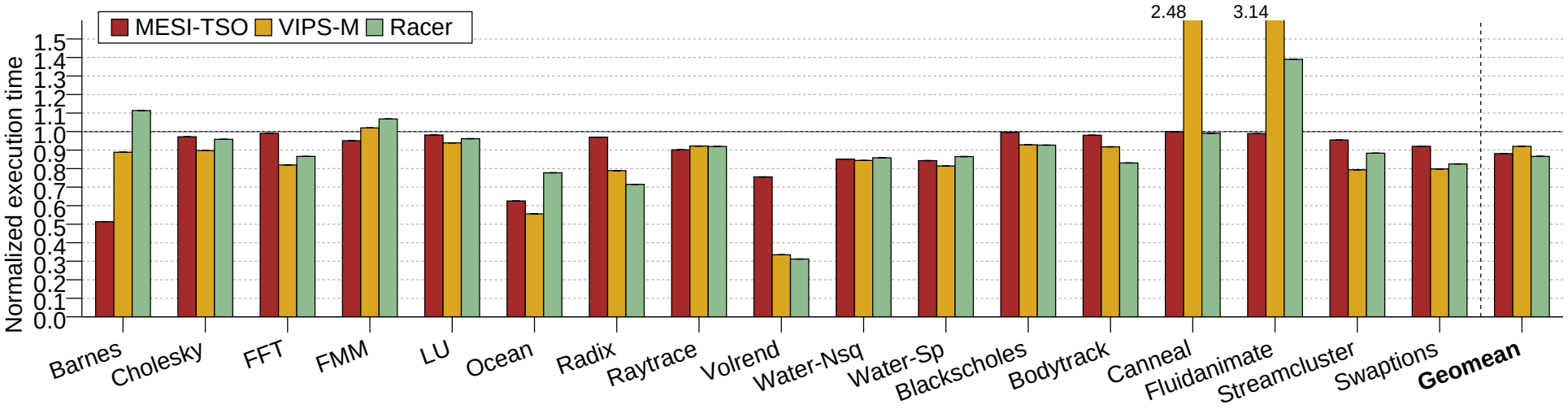
Outline

SIMULATION ENVIRONMENT

- 64-core tiled-CMP (GEMS simulator)
  - L1 (private): 32KB 4-way
  - LLC (shared): 256KB 16-way (per tile)
  - RAWR DETECTOR: 256-byte bloom filter
  - RACER overhead: ≈18KB per tile
- Benchmarks: Splash-3 and Parsec-2.1
- Protocols evaluated:
  - MESI: Directory-based SC protocol
  - MESI-TSO: Directory-based TSO protocol
  - VIPS-M: SC-for-DRF protocol
  - RACER: TSO protocol

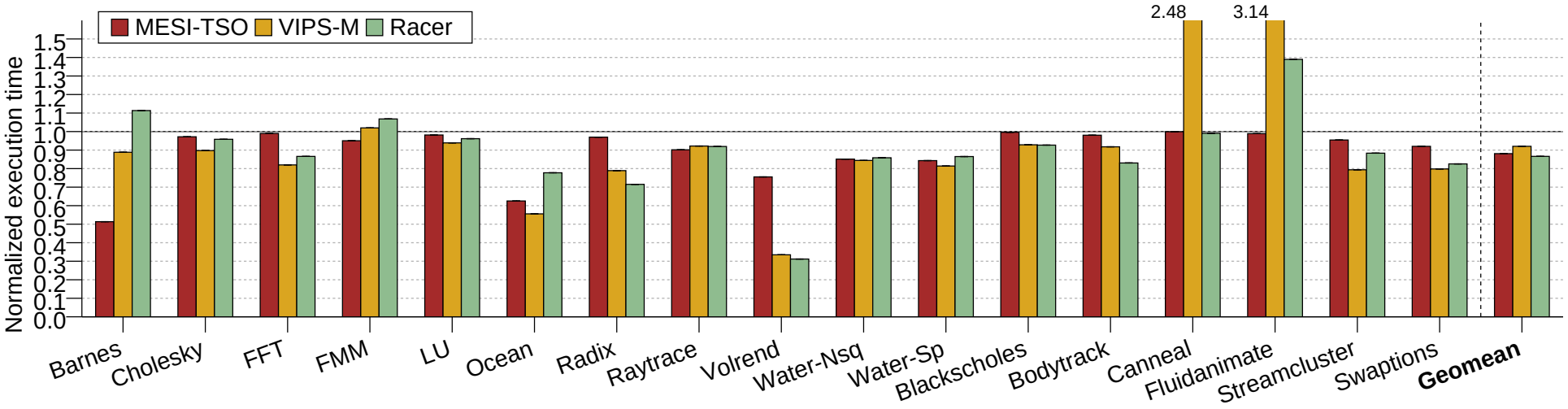
EXECUTION TIME

- Normalized to MESI



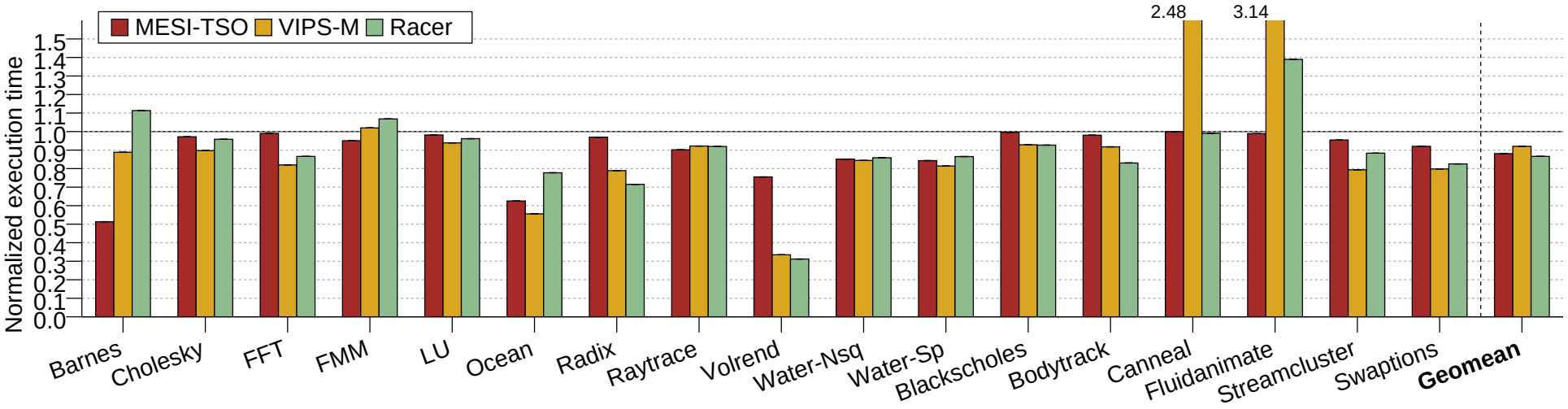
EXECUTION TIME

- Normalized to MESI
- VIPS-M: Conservative SI & SD results in dramatic slow-downs for Fluidanimate and Canneal (Obs.2)



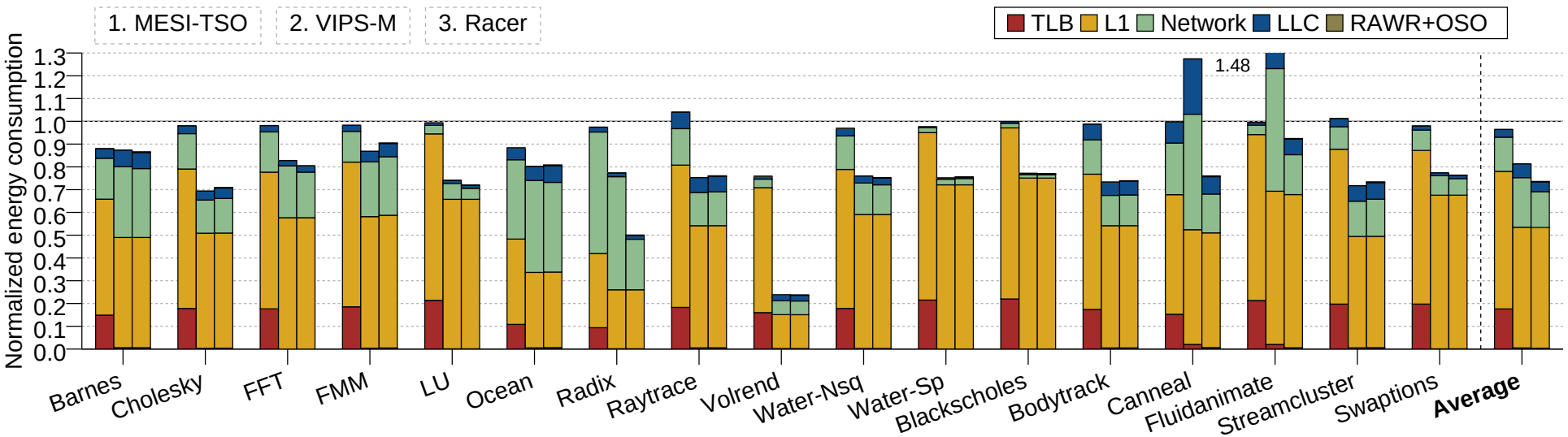
EXECUTION TIME

- Normalized to MESI
- VIPS-M: Conservative SI & SD results in dramatic slow-downs for Fluidanimate and Canneal (Obs.2)
- RACER ≈ non-scalable MESI-TSO
- RACER: better performance than VIPS-M, while providing stronger consistency, but only when needed at run time



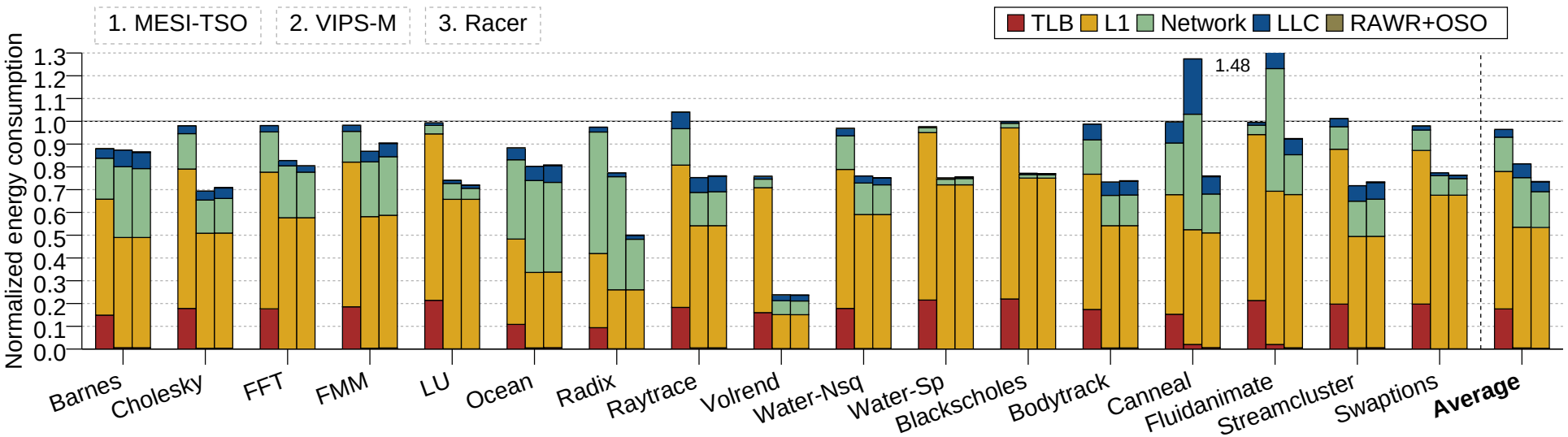
ENERGY CONSUMPTION

- Energy of TLBs, L1 caches, network, LLC, and RAWR
- Normalized to MESI



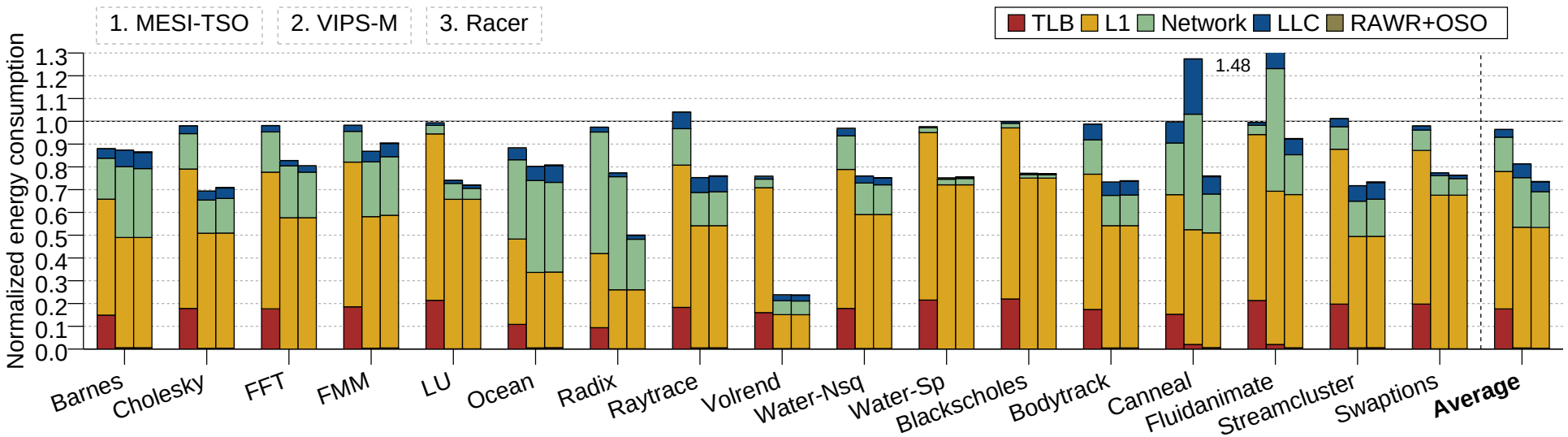
ENERGY CONSUMPTION

- Energy of TLBs, L1 caches, network, LLC, and RAWR
- Normalized to MESI
- RACER gets the best from MESI-TSO and VIPS-M



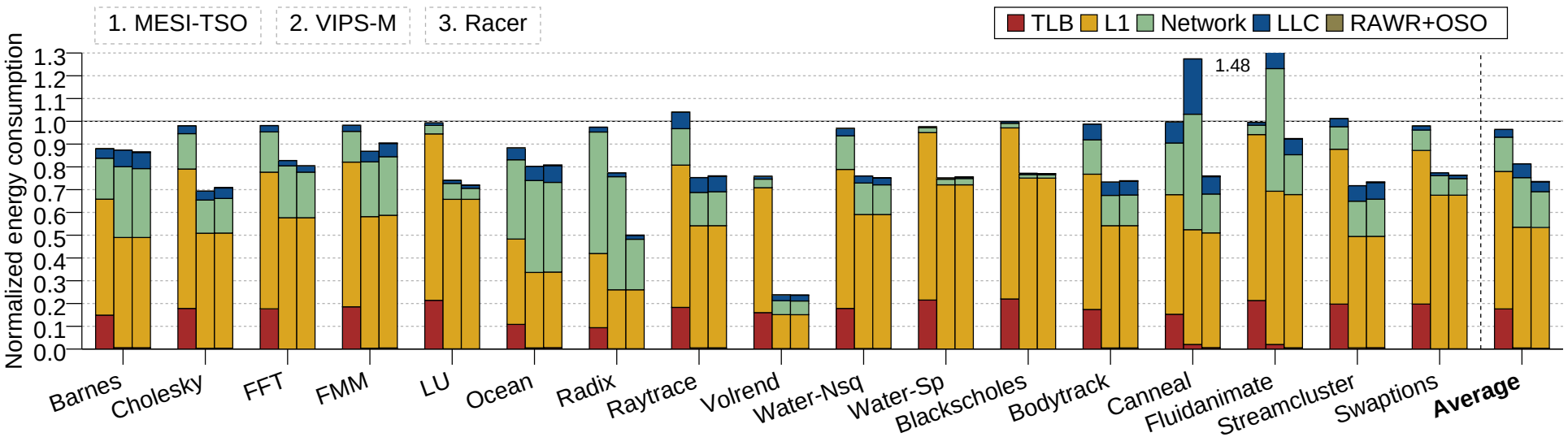
ENERGY CONSUMPTION

- Energy of TLBs, L1 caches, network, LLC, and RAWR
- Normalized to MESI
- RACER gets the best from MESI-TSO and VIPS-M
  - TLB consumption mitigated by using virtual caches (as VIPS-M)



ENERGY CONSUMPTION

- Energy of TLBs, L1 caches, network, LLC, and RAWR
- Normalized to MESI
- RACER gets the best from MESI-TSO and VIPS-M
  - TLB consumption mitigated by using virtual caches (as VIPS-M)
  - LLC and network consumption of MESI-TSO (runtime synchronization)



Outline

CONCLUSIONS

- RACER is a novel way of providing TSO consistency
  - $\Rightarrow$ First efficient, request-response protocol for TSO

CONCLUSIONS

- RACER is a novel way of providing TSO consistency
  - ⇒ First efficient, request-response protocol for TSO
- Main benefits of RACER
  - ⇒ No indirection: supports low-cost virtual caches
  - ⇒ No timestamps: collapsed order
  - ⇒ Fast write propagation thanks to race prediction
  - ⇒ Low area overhead

CONCLUSIONS

- RACER is a novel way of providing TSO consistency
  - ⇒ First efficient, request-response protocol for TSO
- Main benefits of RACER
  - ⇒ No indirection: supports low-cost virtual caches
  - ⇒ No timestamps: collapsed order
  - ⇒ Fast write propagation thanks to race prediction
  - ⇒ Low area overhead
- More in the paper
  - ⇒ Implementation of a distributed RAWR
  - ⇒ Implementation for OoO cores with speculation

RACER: TSO CONSISTENCY VIA RACE DETECTION

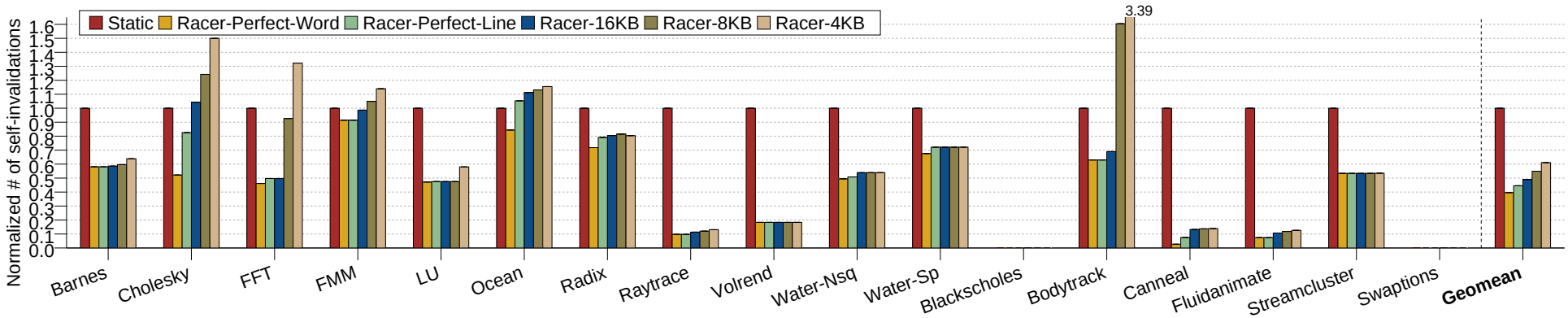
Alberto Ros[1] Stefanos Kaxiras[2]

[1]Universidad de Murcia
aros@ditec.um.es

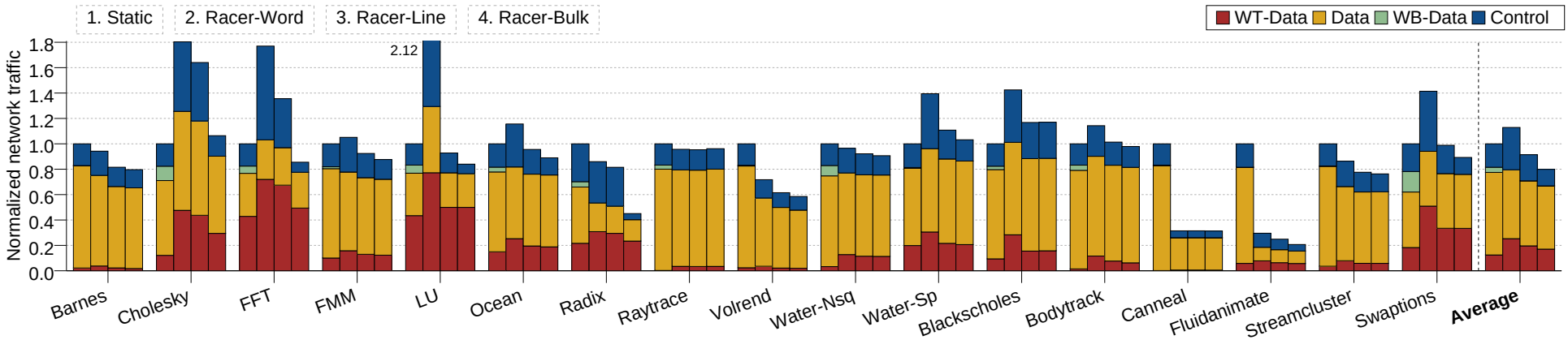
[2]Uppsala University
stefanos.kaxiras@it.uu.se

Nov 28th, 2016

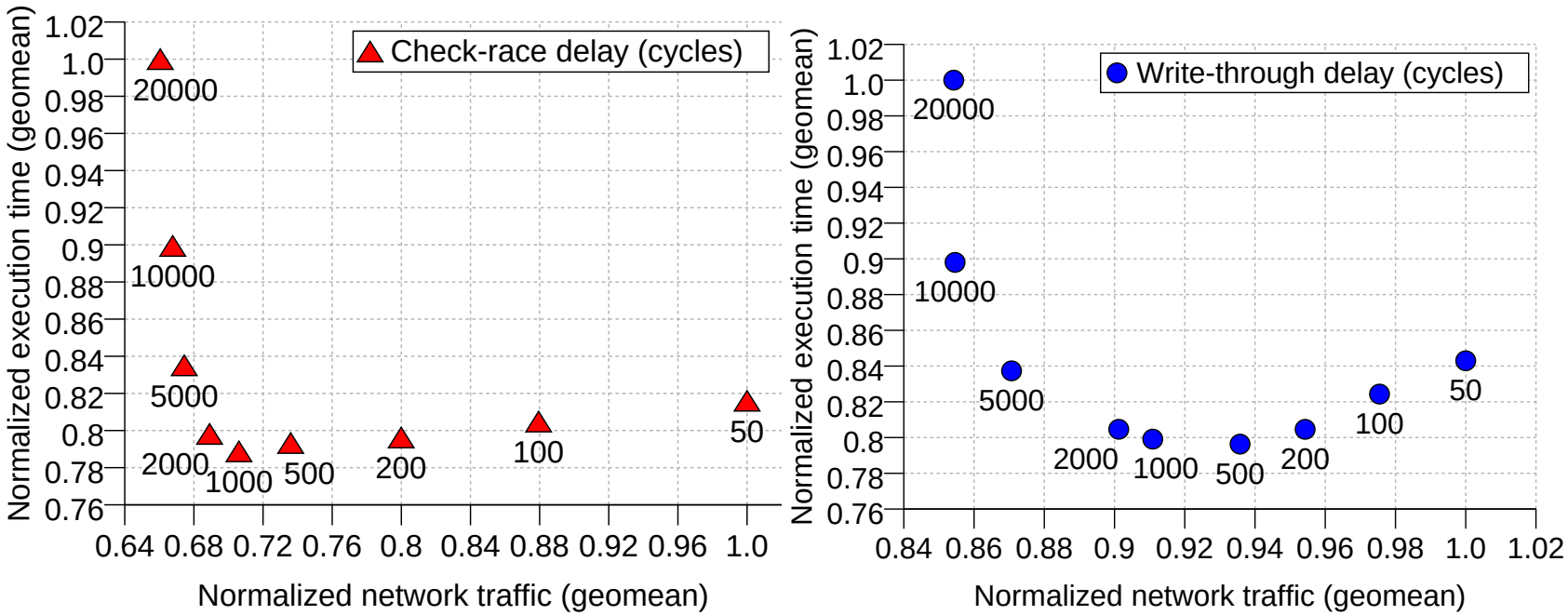
SELF-INVALIDATION



NETWORK TRAFFIC



SENSITIVITY



Protocol weaknesses and related work

Protocol	Strength	DRF	Dir	Time-stamps	Broad-cast	Indirection	Slow release

PROTOCOL WEAKNESSES AND RELATED WORK

Protocol	Strength	DRF	Dir	Time-stamps	Broad-cast	Indirection	Slow release
DIRECTORY	SC	—	✓	—	—	✓	—

- DIR: Non-scalable directory memory overhead
- INDIRECTION: Latency, complexity (protocol states, virtual caches require reverse translation)

PROTOCOL WEAKNESSES AND RELATED WORK

Protocol	Strength	DRF	Dir	Time-stamps	Broad-cast	Indirection	Slow release
DIRECTORY	SC	—	✓	—	—	✓	—
SNOOPING	SC	—	—	—	✓	✓	—

- DIR: Non-scalable directory memory overhead
- BROADCAST: Increases traffic and energy consumption
- INDIRECTION: Latency, complexity (protocol states, virtual caches require reverse translation)

PROTOCOL WEAKNESSES AND RELATED WORK

Protocol	Strength	DRF	Dir	Time-stamps	Broad-cast	Indirection	Slow release
DIRECTORY	SC	—	✓	—	—	✓	—
SNOOPING	SC	—	—	—	✓	✓	—
DENOVO [ASPLOS'15]	SC-for-DRF	✓	—	—	—	✓	—
VIPS/CALLBACKS [ISCA'15]	SC-for-DRF	✓	—	—	—	—	—

- DRF: Not widely supported, software cooperation, conservative
- DIR: Non-scalable directory memory overhead

- BROADCAST: Increases traffic and energy consumption
- INDIRECTION: Latency, complexity (protocol states, virtual caches require reverse translation)

PROTOCOL WEAKNESSES AND RELATED WORK

Protocol	Strength	DRF	Dir	Time-stamps	Broad-cast	Indirection	Slow release
DIRECTORY	SC	—	✓	—	—	✓	—
SNOOPING	SC	—	—	—	✓	✓	—
DENOVO [ASPLOS'15]	SC-for-DRF	✓	—	—	—	✓	—
VIPS/CALLBACKS [ISCA'15]	SC-for-DRF	✓	—	—	—	—	—
TSO-CC [HPCA'14]	TSO	—	—	✓	✓	✓	✓
TARDIS 2.0 [PACT'16]	TSO	—	—	✓	—	✓	✓

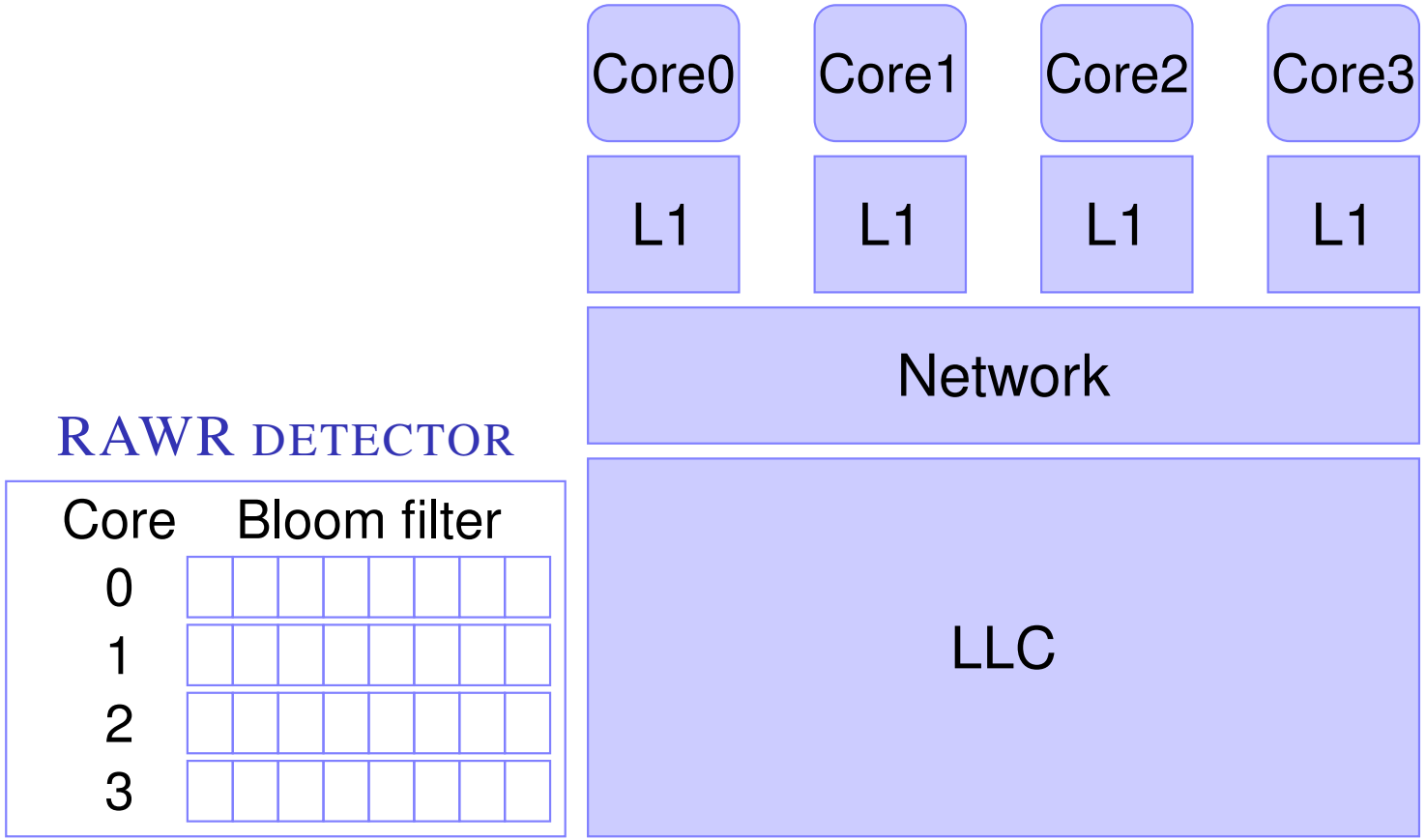
- DRF: Not widely supported, software cooperation, conservative
- DIR: Non-scalable directory memory overhead
- TIMESTAMPS: Size (L1), rollback, evictions
- BROADCAST: Increases traffic and energy consumption
- INDIRECTION: Latency, complexity (protocol states, virtual caches require reverse translation)
- SLOW RELEASE: Slowdown application progress (performance)

PROTOCOL WEAKNESSES AND RELATED WORK

Protocol	Strength	DRF	Dir	Time-stamps	Broad-cast	Indirection	Slow release
DIRECTORY	SC	—	✓	—	—	✓	—
SNOOPING	SC	—	—	—	✓	✓	—
DENOVO [ASPLOS'15]	SC-for-DRF	✓	—	—	—	✓	—
VIPS/CALLBACKS [ISCA'15]	SC-for-DRF	✓	—	—	—	—	—
TSO-CC [HPCA'14]	TSO	—	—	✓	✓	✓	✓
TARDIS 2.0 [PACT'16]	TSO	—	—	✓	—	✓	✓
RACER	TSO	—	—	—	—	—	—

- DRF: Not widely supported, software cooperation, conservative
- DIR: Non-scalable directory memory overhead
- TIMESTAMPS: Size (L1), rollback, evictions
- BROADCAST: Increases traffic and energy consumption
- INDIRECTION: Latency, complexity (protocol states, virtual caches require reverse translation)
- SLOW RELEASE: Slowdown application progress (performance)

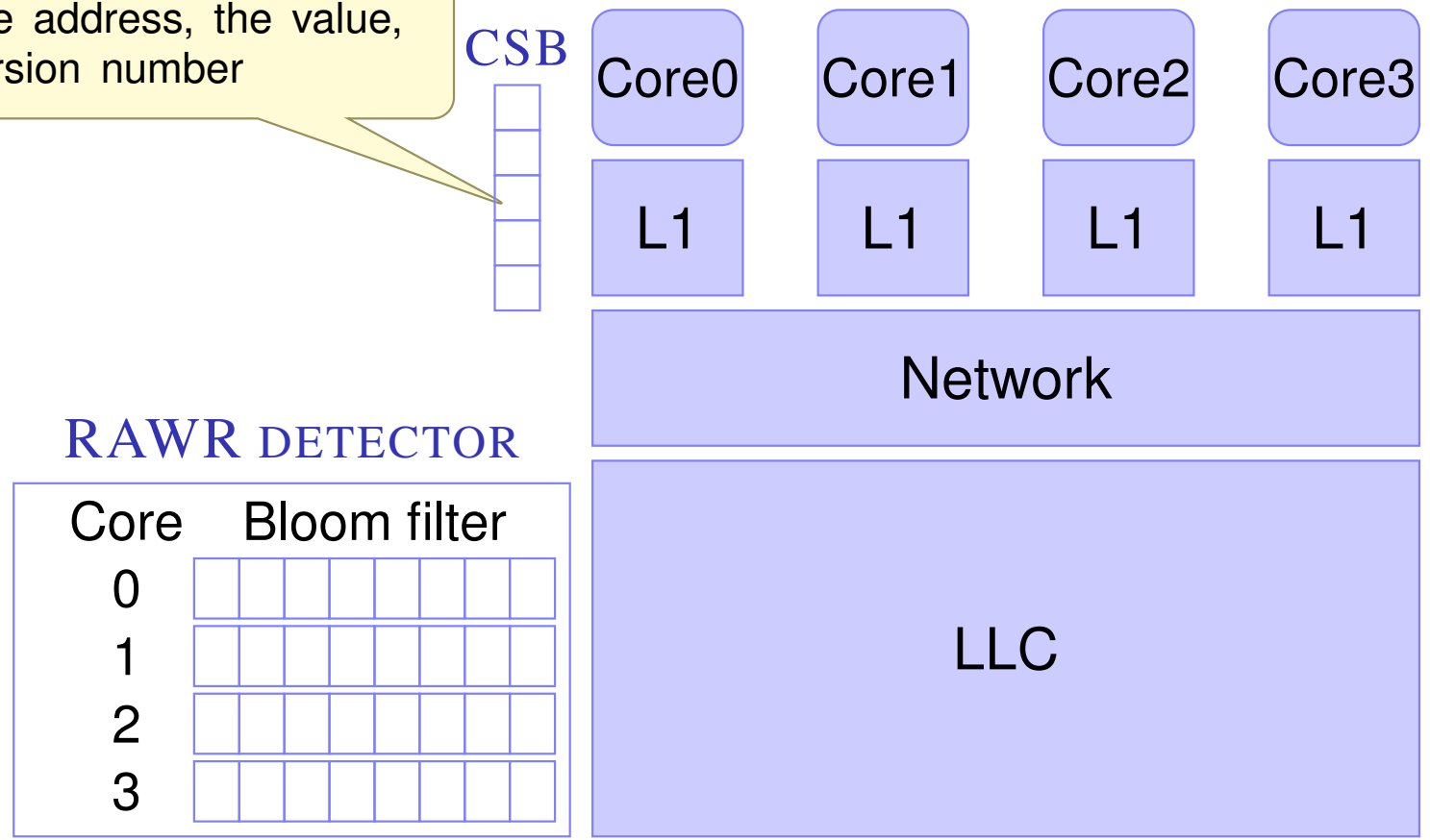
Optimizing write-back traffic



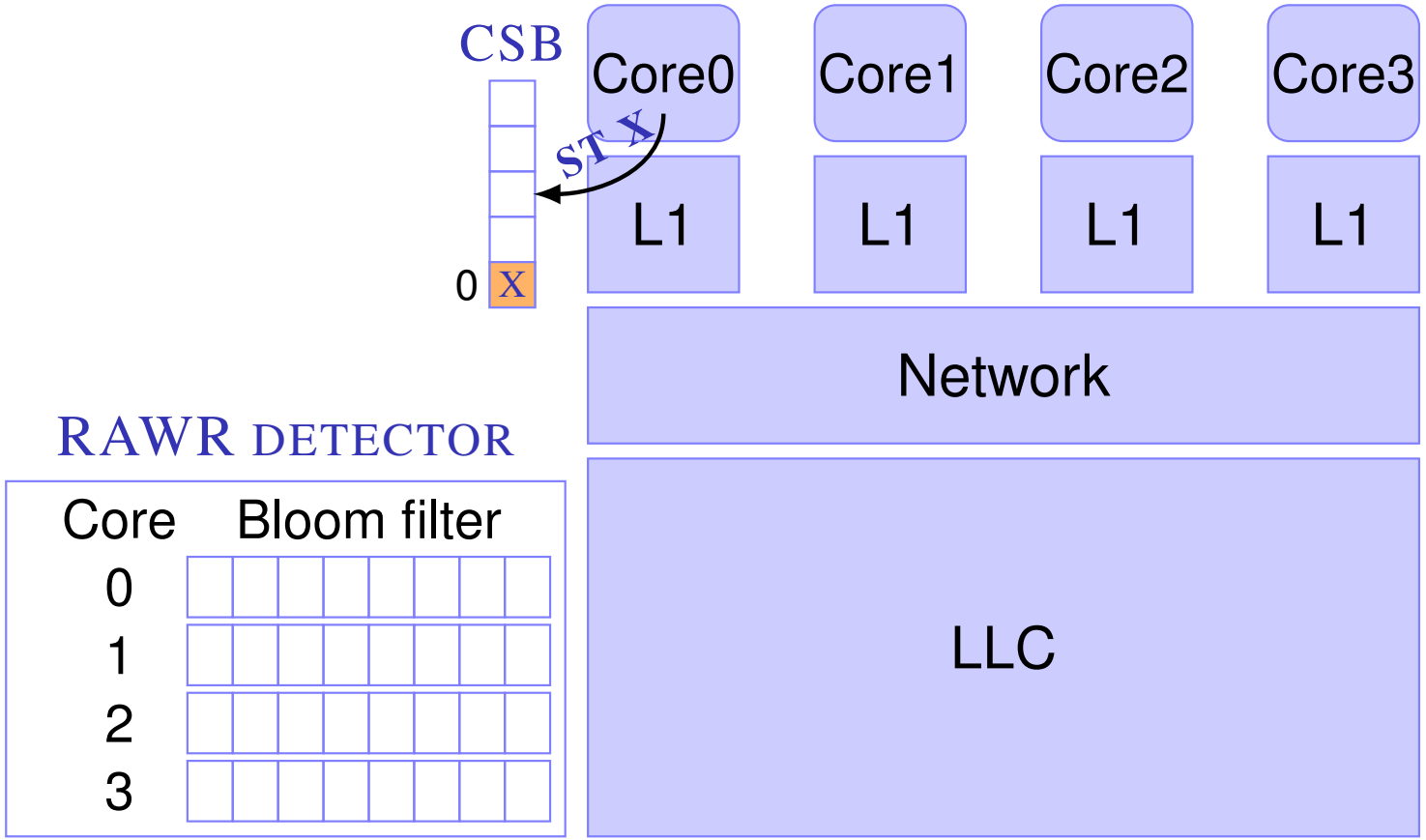
Optimizing write-back traffic

Local coalescing store buffer (CSB)

- Stores the address, the value, and a version number

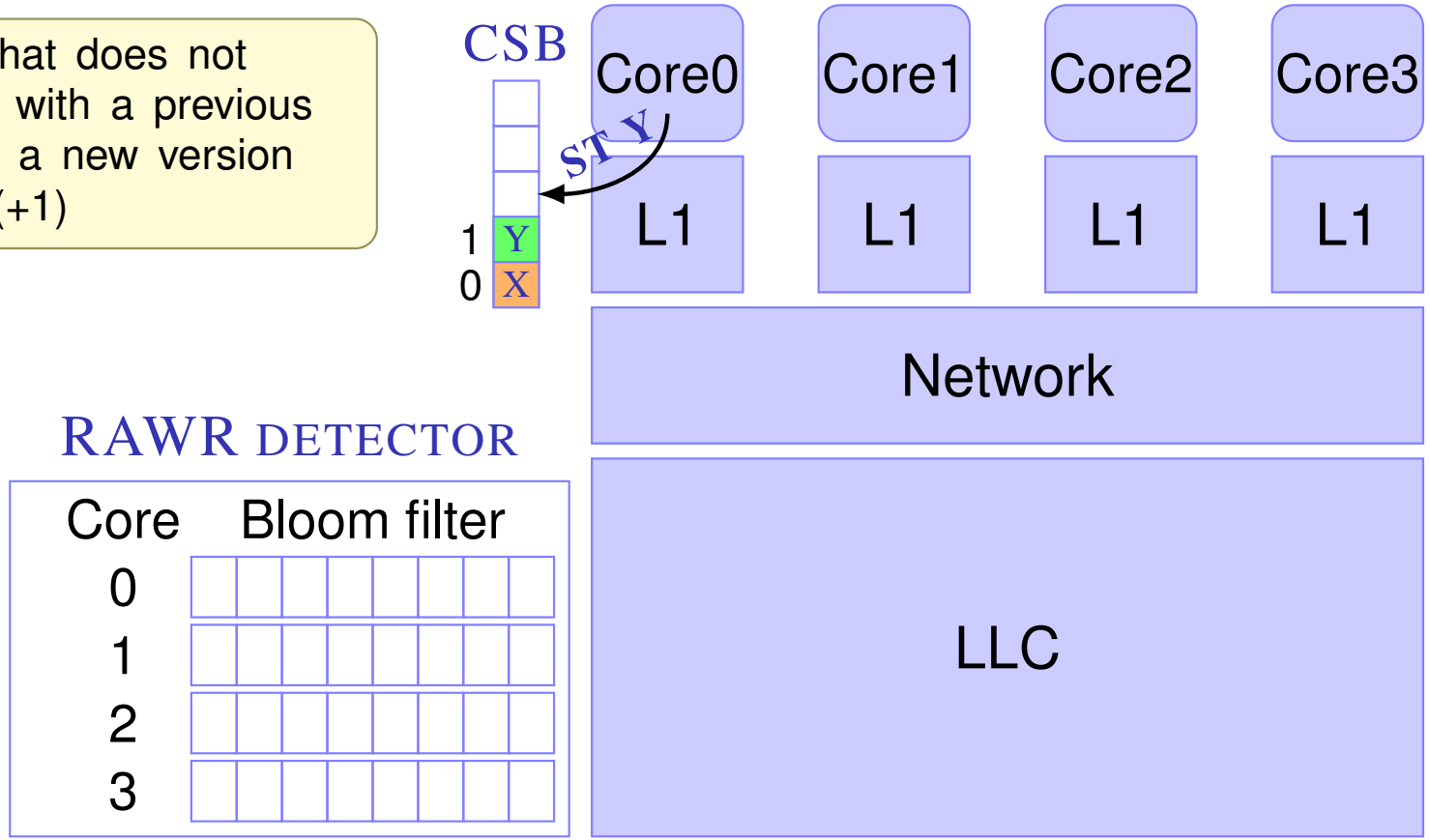


Optimizing write-back traffic

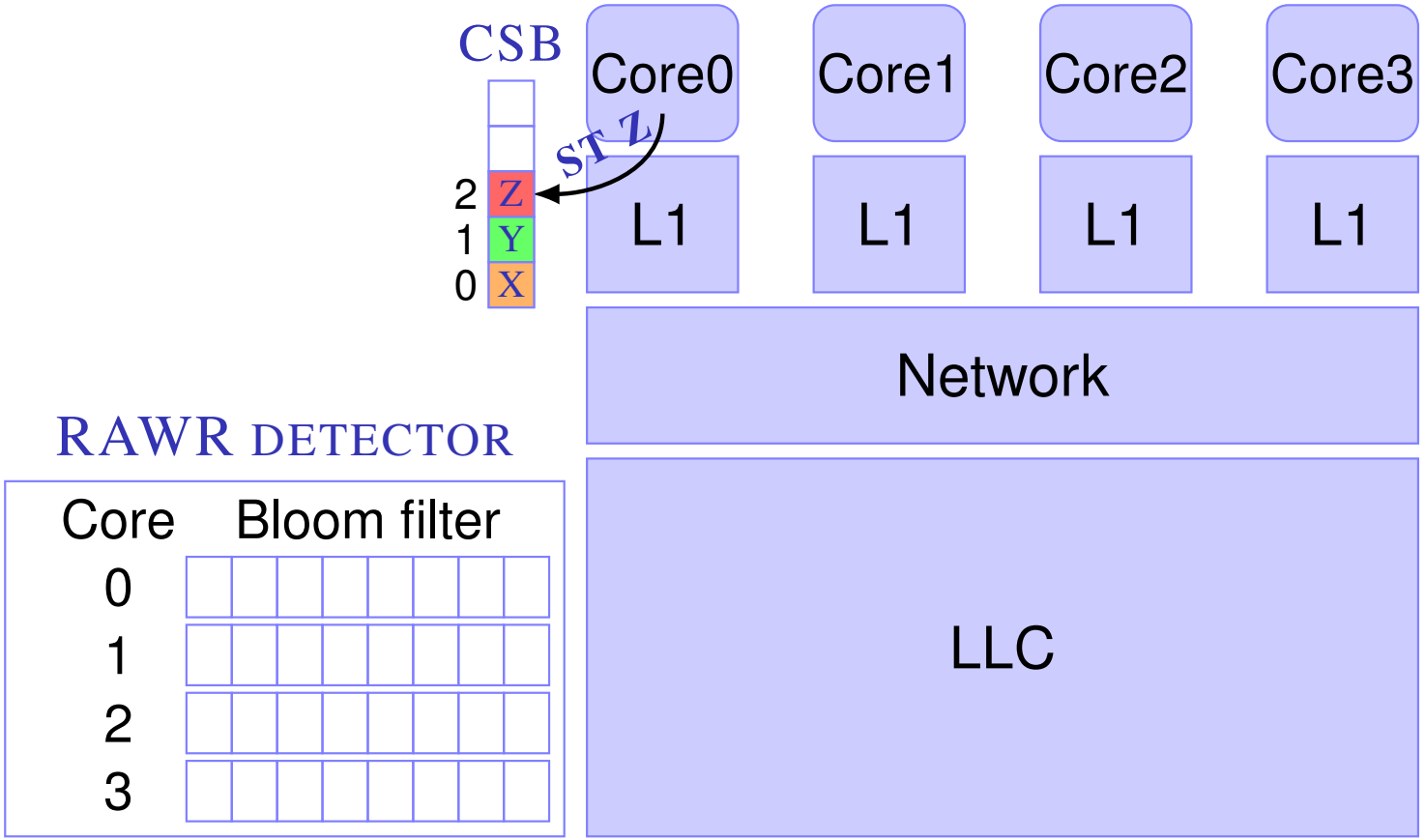


Optimizing write-back traffic

A write that does not coalesce with a previous one gets a new version number (+1)

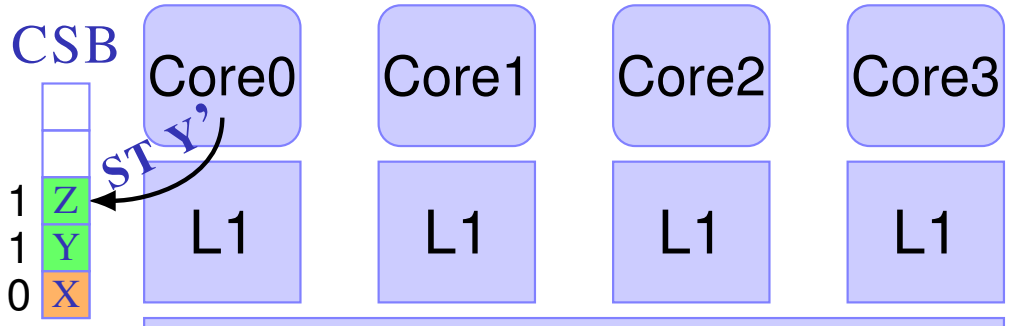


Optimizing write-back traffic



Optimizing write-back traffic

A write that coalesces forces all version numbers between the last one and the coalesced one to be same (e.g., coalesced order 1)

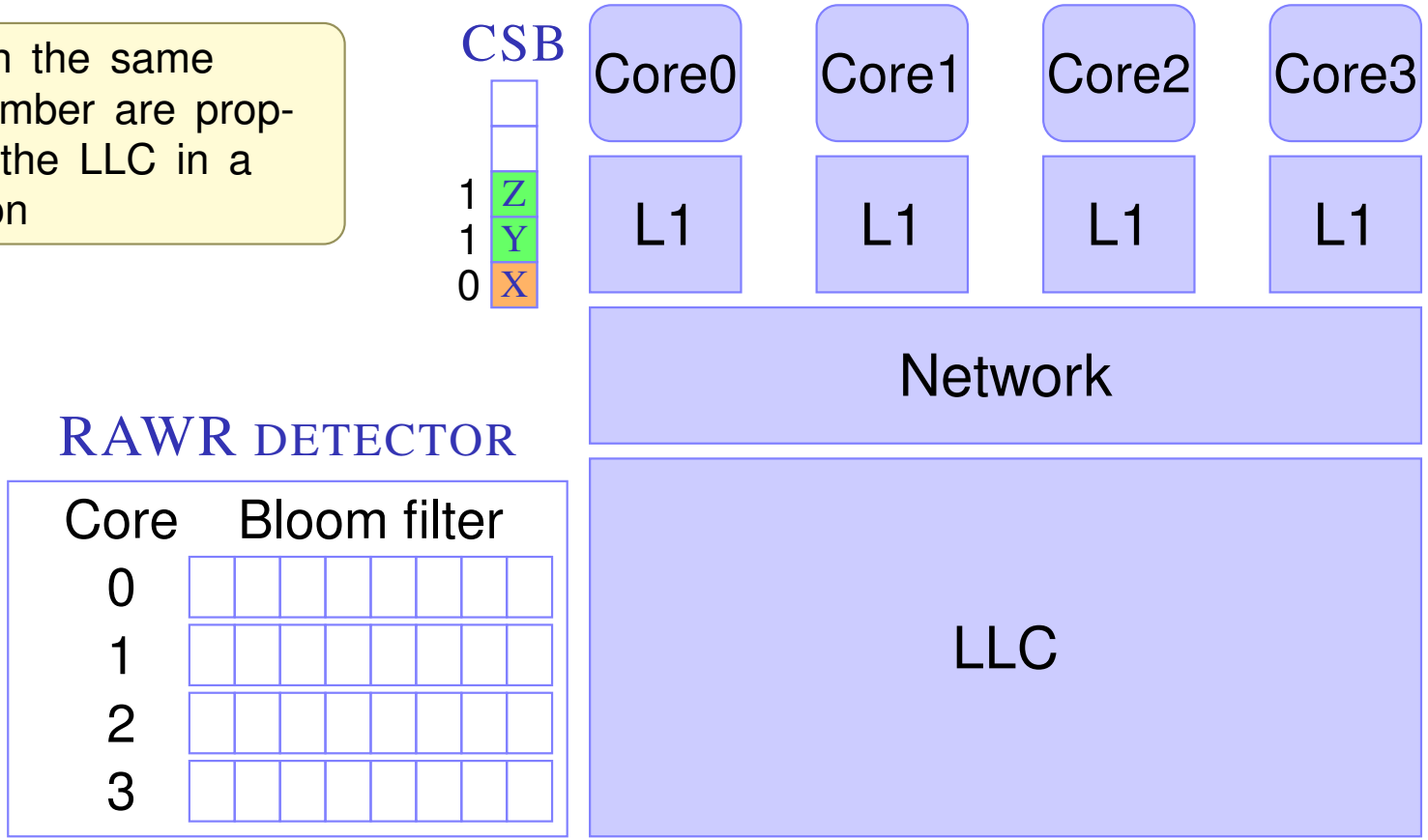


RAWR detector

Core	Bloom filter
0	
1	
2	
3	

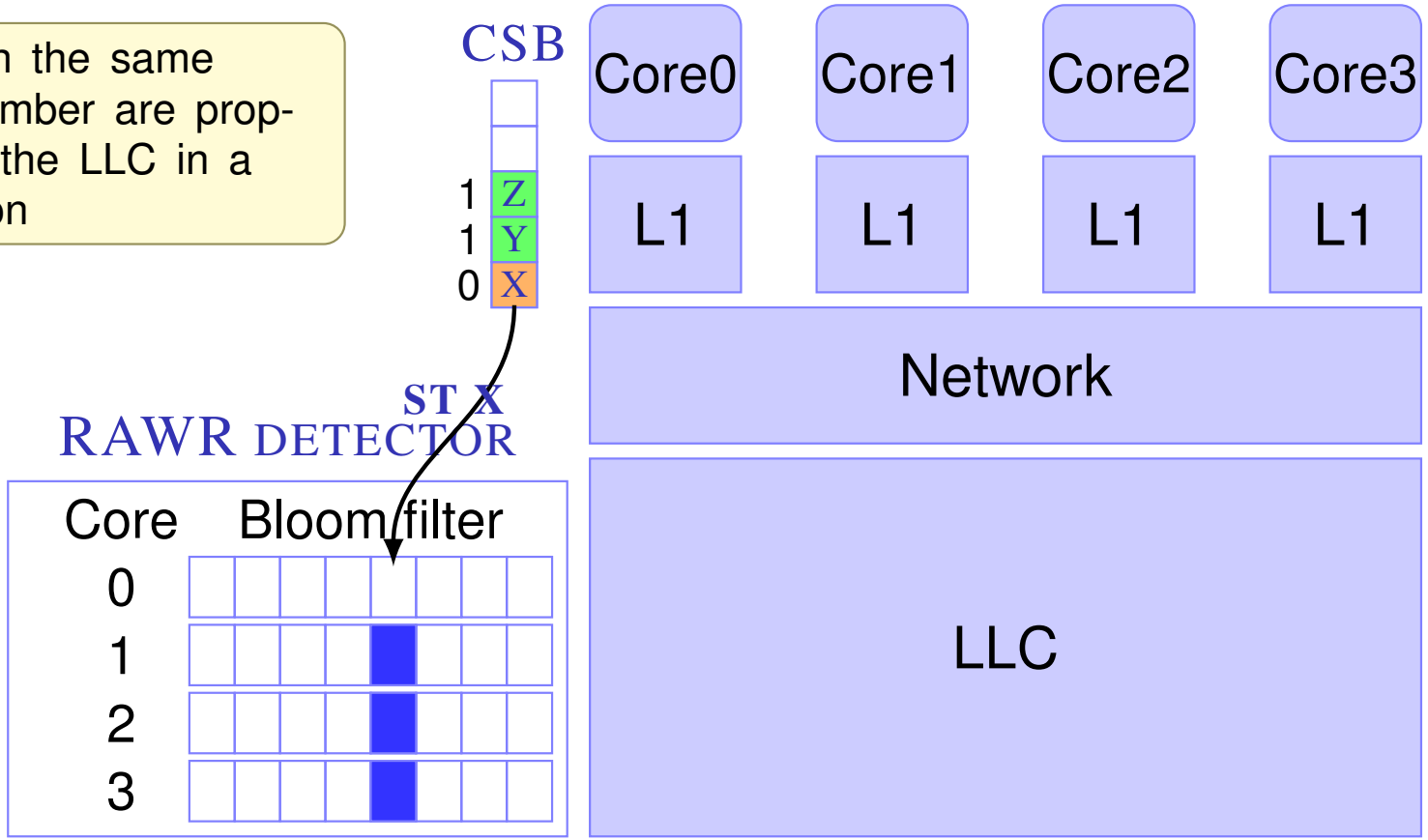
Optimizing write-back traffic

Writes with the same version number are propagated to the LLC in a bulk fashion



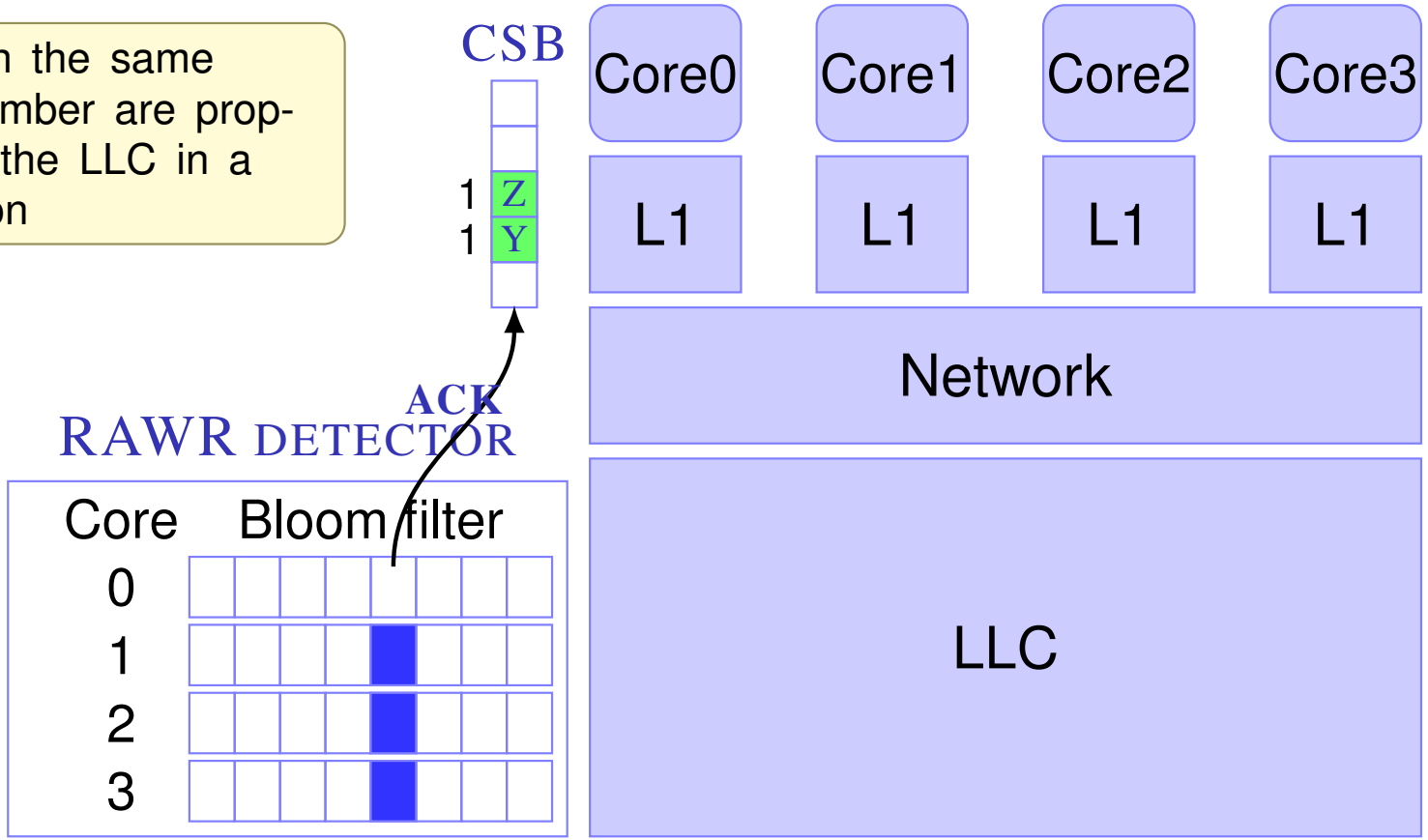
Optimizing write-back traffic

Writes with the same version number are propagated to the LLC in a bulk fashion



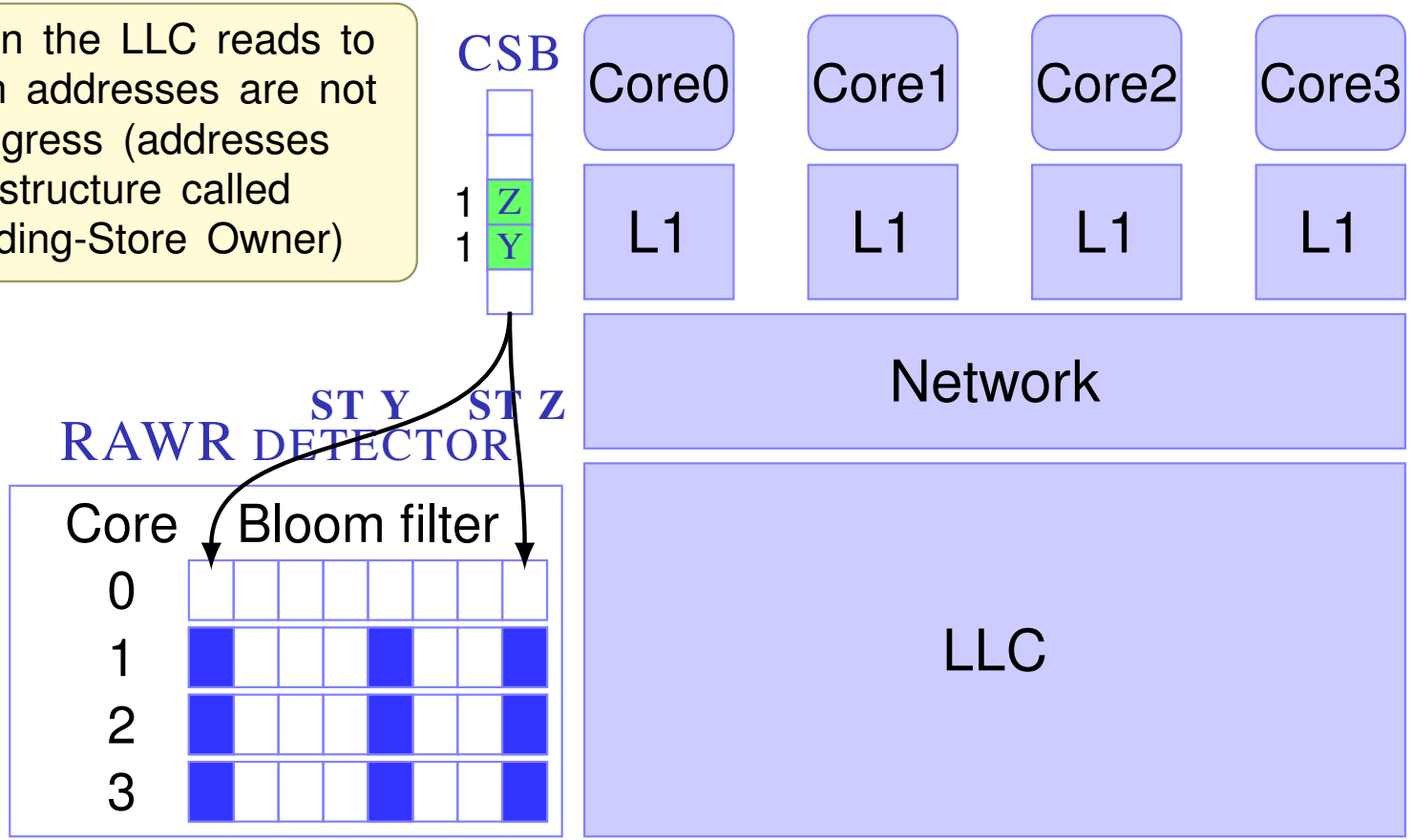
Optimizing write-back traffic

Writes with the same version number are propagated to the LLC in a bulk fashion



Optimizing write-back traffic

While writing in the LLC reads to already written addresses are not allowed to progress (addresses tracked in an structure called OSO, Outstanding-Store Owner)



Optimizing write-back traffic