Pinned Loads: Taming Speculative Loads in Secure Processors

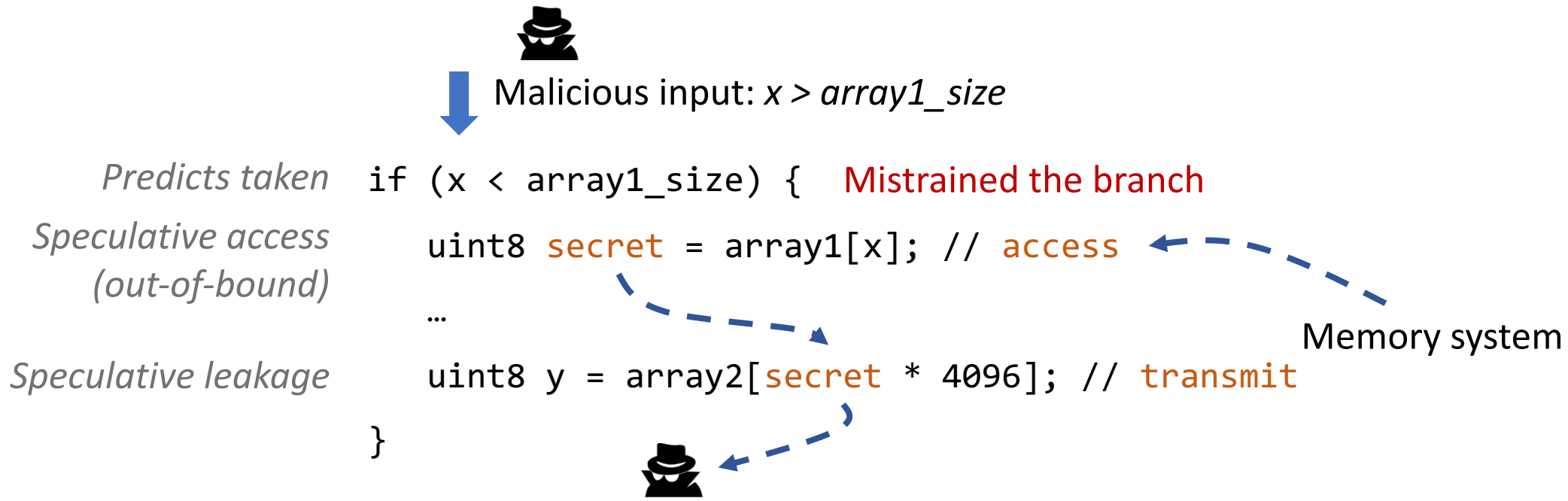
Zirui Neil Zhao, Houxiang Ji, Adam Morrison, Darko Marinov, Josep Torrellas

University of Illinois Tel Aviv University

ASPLOS'22

Speculative Execution Attacks

Modern microprocessors are threatened by speculative execution side-channel attacks



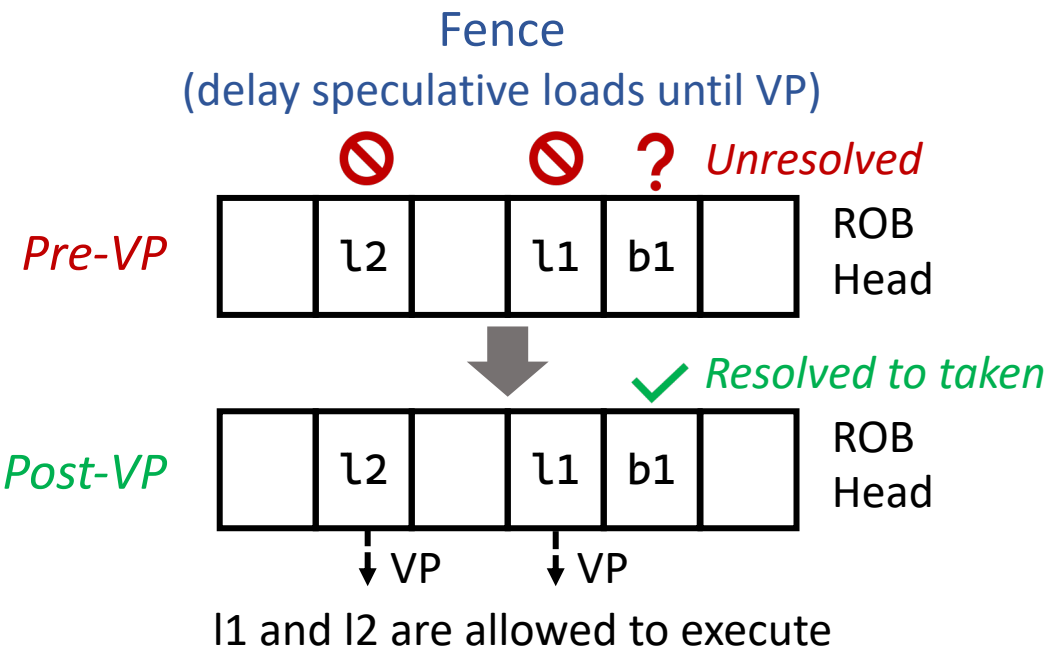
Visibility Point (VP)

Hardware Defenses: protect the execution of vulnerable instructions until they reach their ***visibility point (VP)*** --- i.e., they are no longer vulnerable to pipeline squashes that are relevant to the threat model considered

Reach VP sooner ⇒ Better performance

```
if (x < array1_size) { // b1
   secret = array1[x]; // l1
   …
   load secret; // l2
}
```

Assume: x is in-bound, b1 resolves to taken

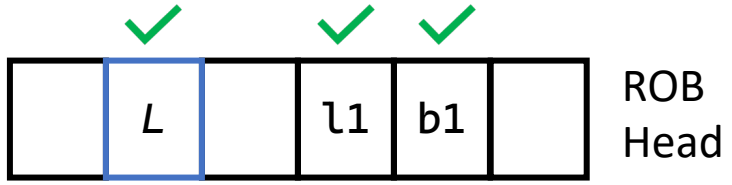


VP Under the Comprehensive Model

Consider: vulnerable instruction=> Load (*L*)

Comprehensive Model

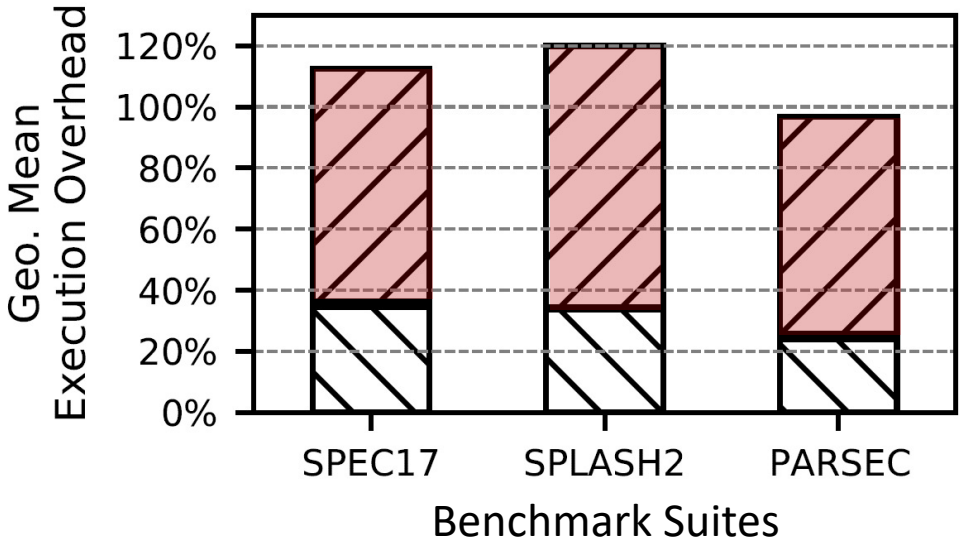
- All control-flow instructions older than *L* are resolved
- *L* plus any older instruction cannot suffer exceptions
- *L* plus any older load cannot alias with other loads/stores
- *L* plus any older load cannot cause a memory consistency violation (MCV)



L needs to reach the ROB head or become the oldest load to be invulnerable to MCV

Focus on Memory Consistency Violations

Execution overhead breakdown of **Fence**
(by each reason that delays reaching the VP)



Conditions of Reaching VP
(Comprehensive Model)

- No MCVs
- No exceptions
- No aliasing
- No control-flow mispredictions

Ensuring no MCVs causes the most overhead

Pinned Loads

Goal: a **general** hardware mechanism that makes loads invulnerable to MCVs as early as possible

Intuition: When certain conditions are satisfied, *Pinned Loads* will **pin** a load *L* *and guarantee:*

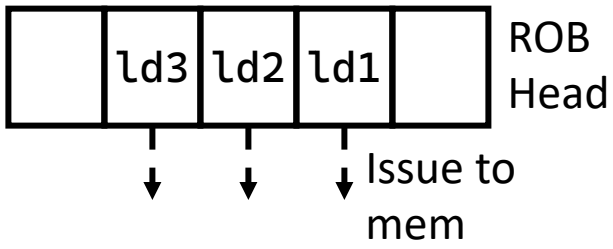
- No squashes of *L* due to invalidations
- No squashes of *L* due to cache evictions

$\Rightarrow$ after pinning, **L is invulnerable to MCVs**

L reaches VP sooner (before reaching the ROB head) $\Rightarrow$ higher performance

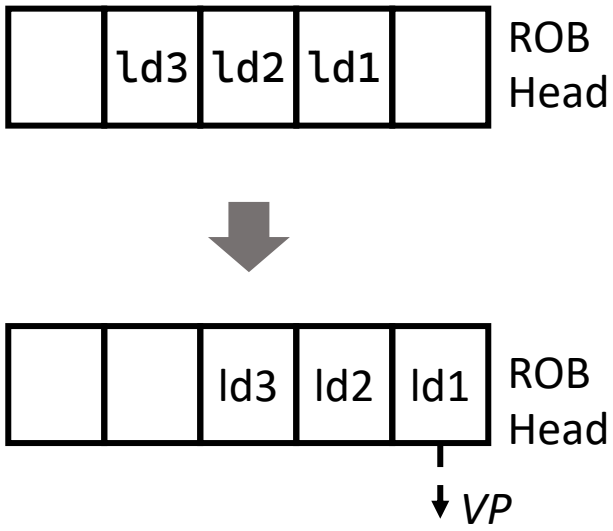
Potential Performance Gain

Conventional Unsafe



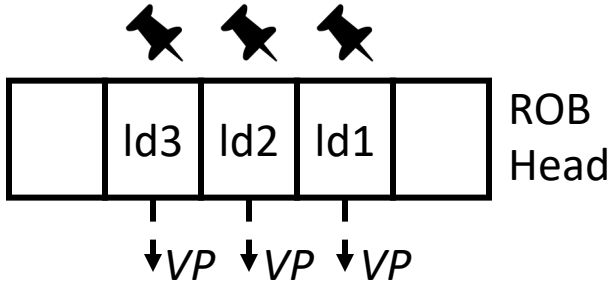
All independent loads in parallel

Fence



Serialized!

Fence + *Pinned Loads*



Quickly “pass” the VP downstream, issue all independent loads in parallel

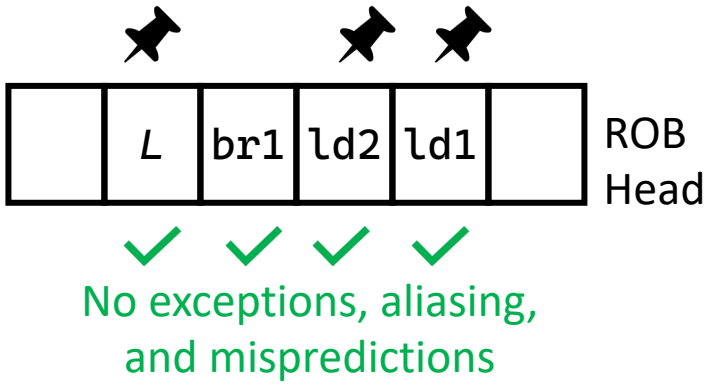
Threat Model

Assume the Comprehensive threat model:

- No control-flow mispredictions
- No aliasing
- No exceptions
- No MCVs

Preserve the speculative execution security properties of the baseline defense schemes:

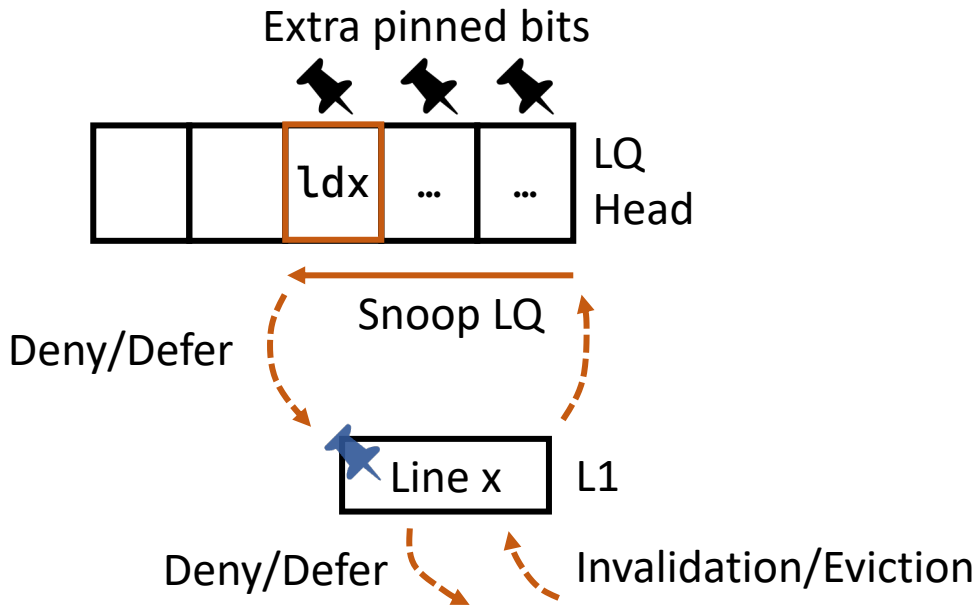
A load *L* can be pinned only if *L* has met all the conditions to reach VP, except for guaranteeing *L* itself will not cause an MCV



Implication: Loads are pinned in the program order

Pinned Loads Overview

Intuition: defer invalidations and prevent evictions to lines that are read by a pinned load

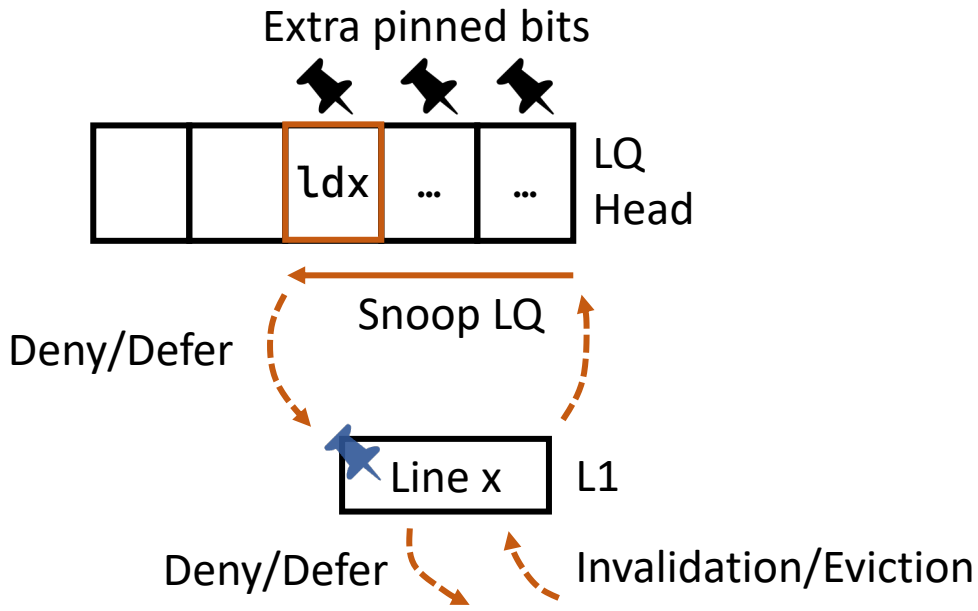


Conceptually, line x is “pinned” (no actual pinned bit)

Line x is automatically “un-pinned” when ldx retires

Pinned Loads Overview

Intuition: defer invalidations and prevent evictions to lines that are read by a pinned load



Conceptually, line x is “pinned” (no actual pinned bit)

Line x is automatically “un-pinned” when ldx retires

Design Overview:

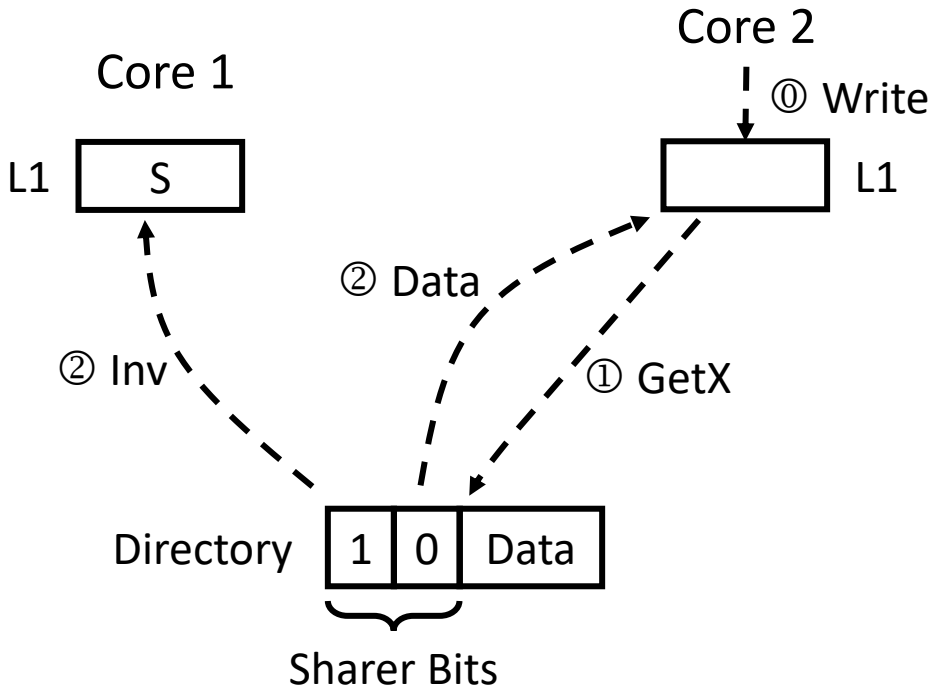
- *Defer Invalidations:* introduce new coherence messages to defer remote writes

  Safety guarantees:
  + No starvation
  + No deadlock (*detailed in the paper*)

- *Prevent Cache Evictions:* guarantee space in cache & directory:
  + Late Pinning (LP)
  + Early Pinning (EP)

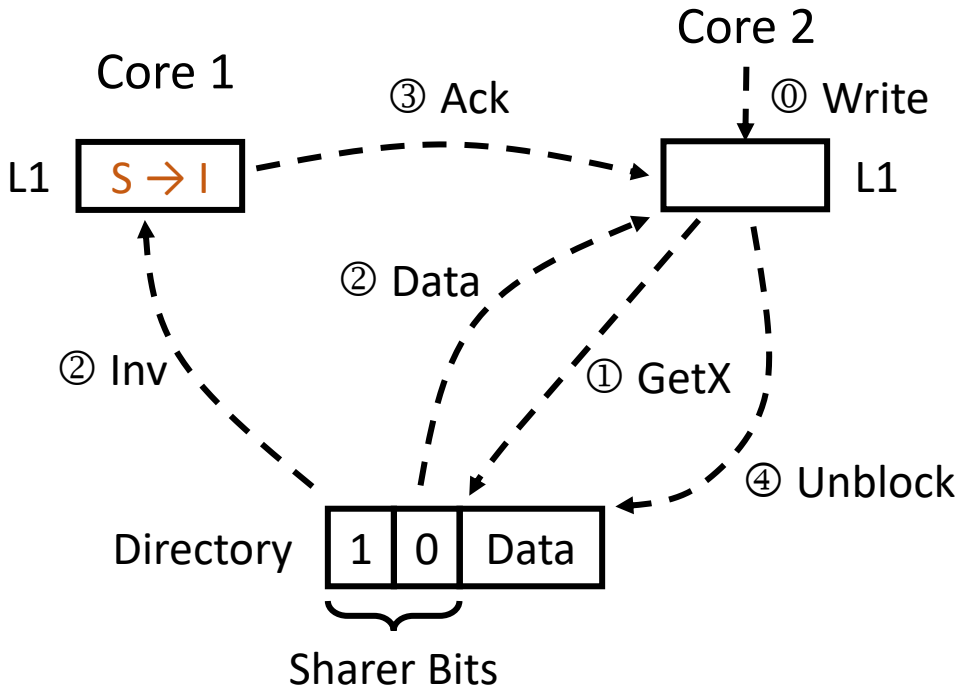
Defer Invalidations to Pinned Lines

Conventional Protocol

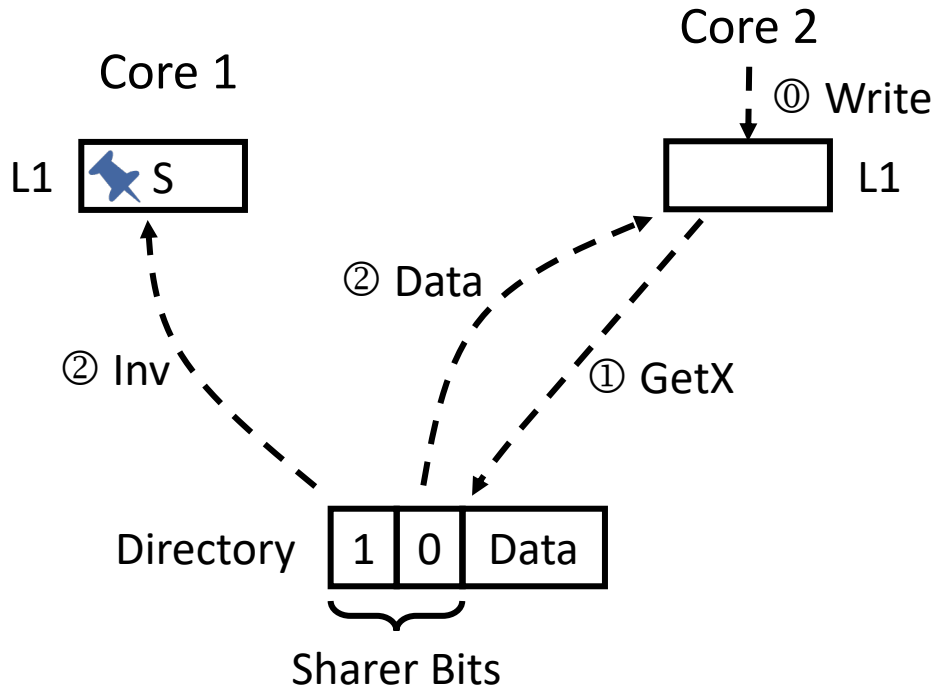


Defer Invalidations to Pinned Lines

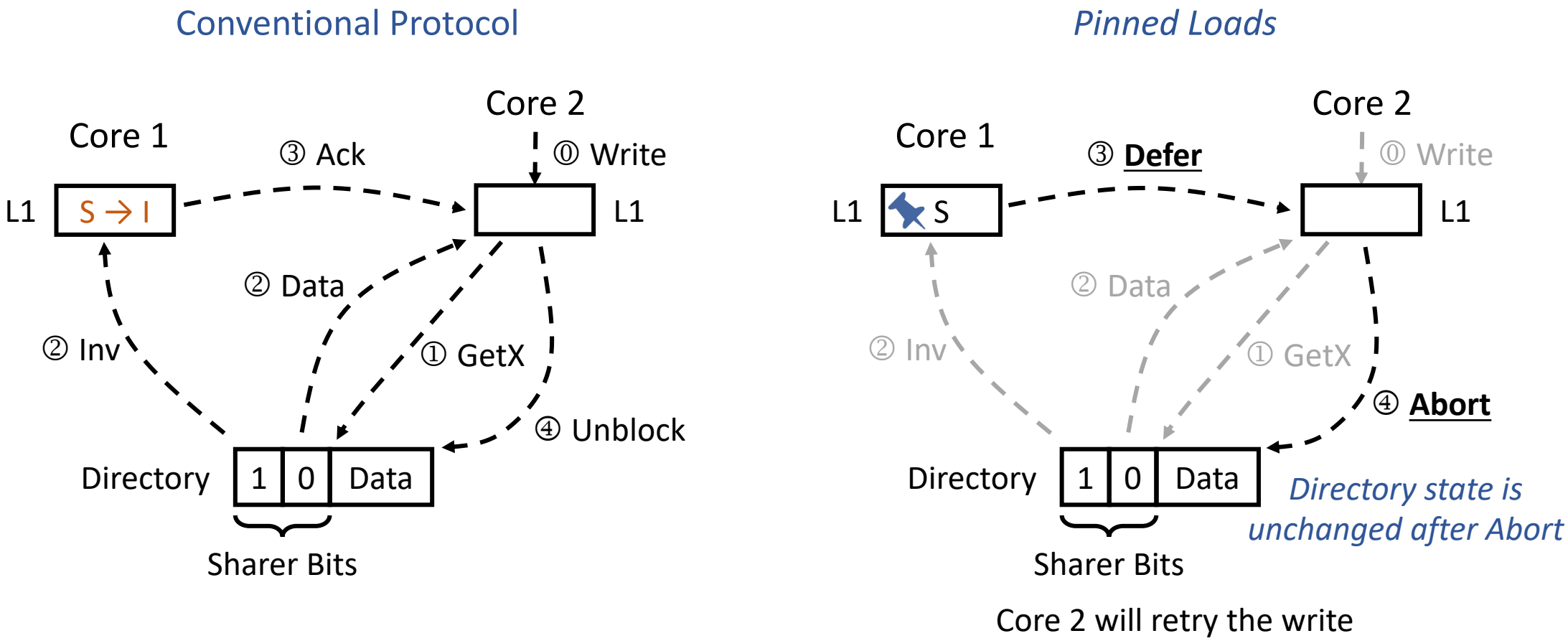
Conventional Protocol



Pinned Loads



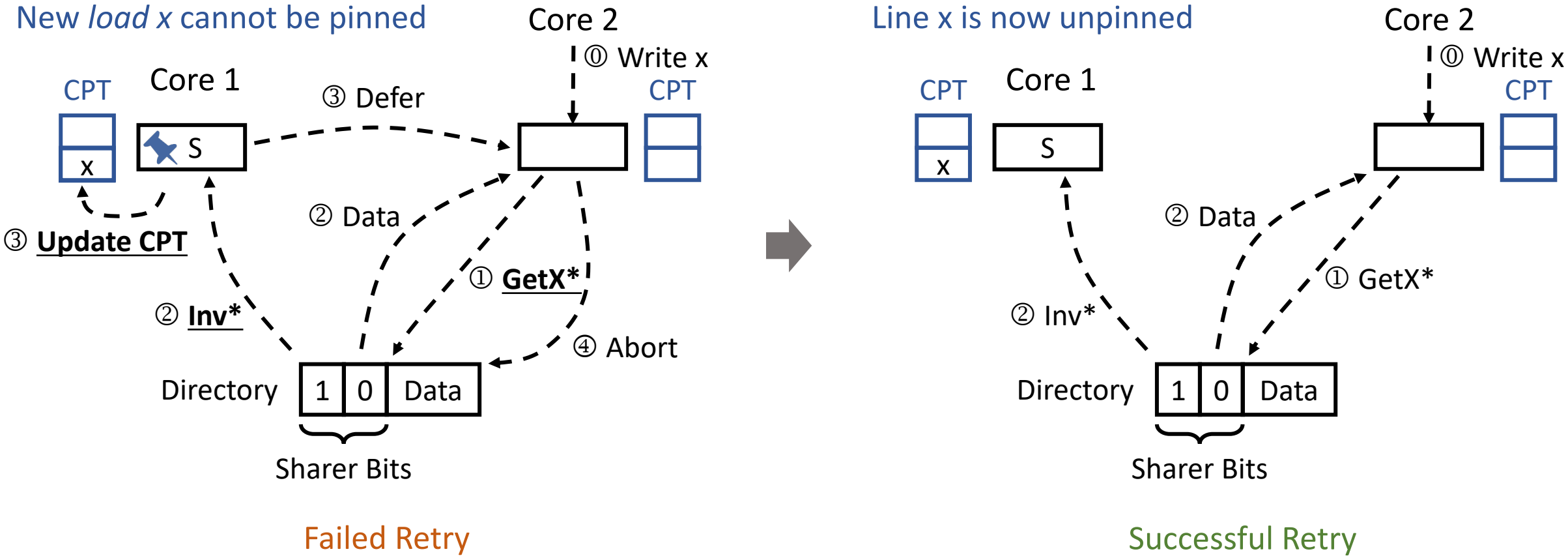
Defer Invalidations to Pinned Lines



Changes from the Conventional: new coherence messages (and the logic of handling them)

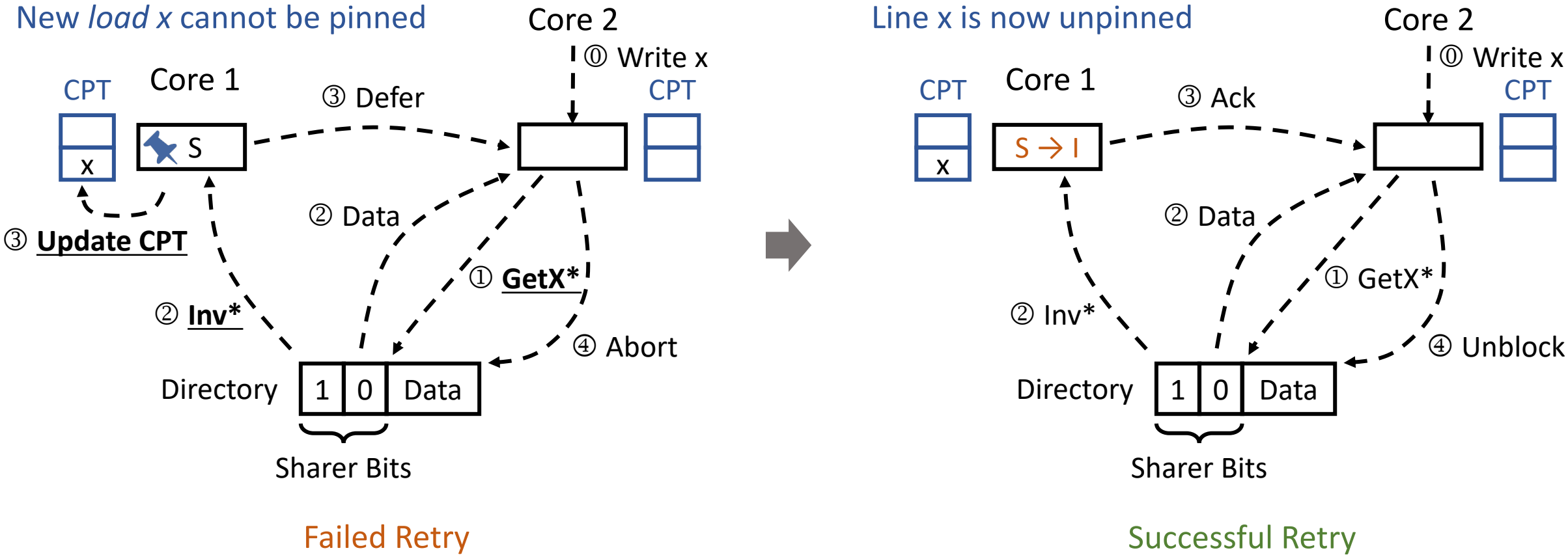
Prevent Store Starvation

Cannot-Pin Table (CPT): a per-core hardware structure that records the addresses of lines that the core is not allowed to pin *at the moment*



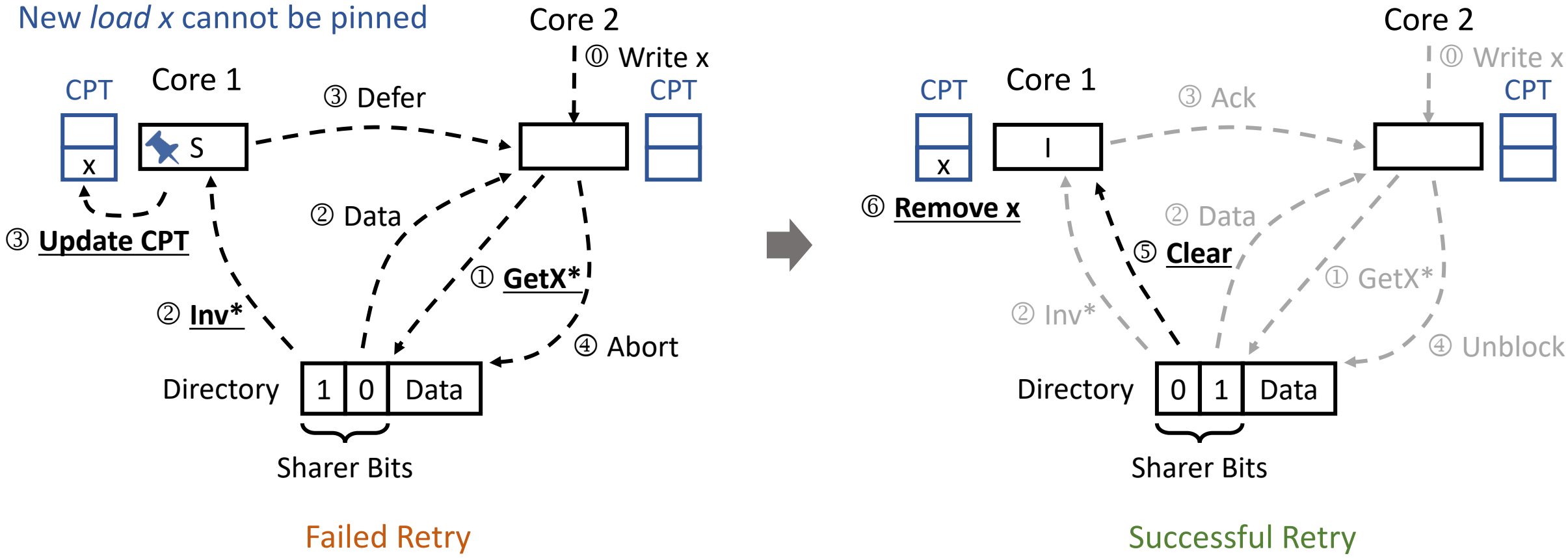
Prevent Store Starvation

Cannot-Pin Table (CPT): a per-core hardware structure that records the addresses of lines that the core is not allowed to pin *at the moment*



Prevent Store Starvation

Cannot-Pin Table (CPT): a per-core hardware structure that records the addresses of lines that the core is not allowed to pin *at the moment*

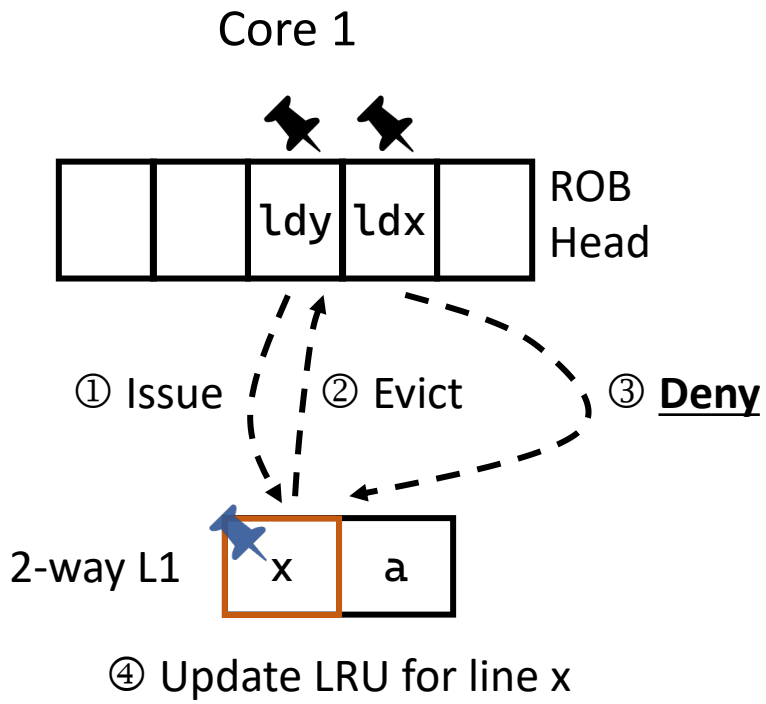


Prevent Evictions of Pinned Lines

Intuition: Pinned Loads denies evictions to pinned lines

```
1: ld x // L1 Hit
2: ld y // L1 Miss
```

Line x and y are mapped to the same L1 set

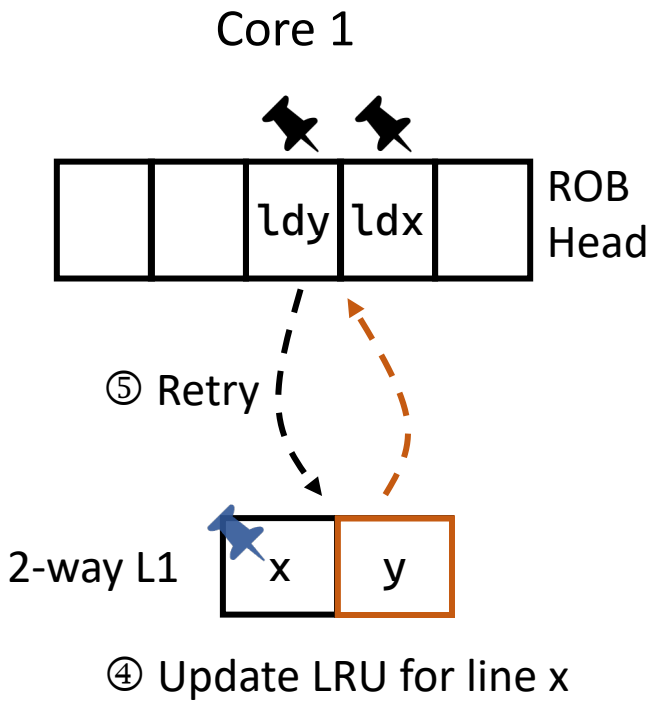


Prevent Evictions of Pinned Lines

Intuition: Pinned Loads denies evictions to pinned lines

```
1: ld x // L1 Hit
2: ld y // L1 Miss
```

Line x and y are mapped to the same L1 set



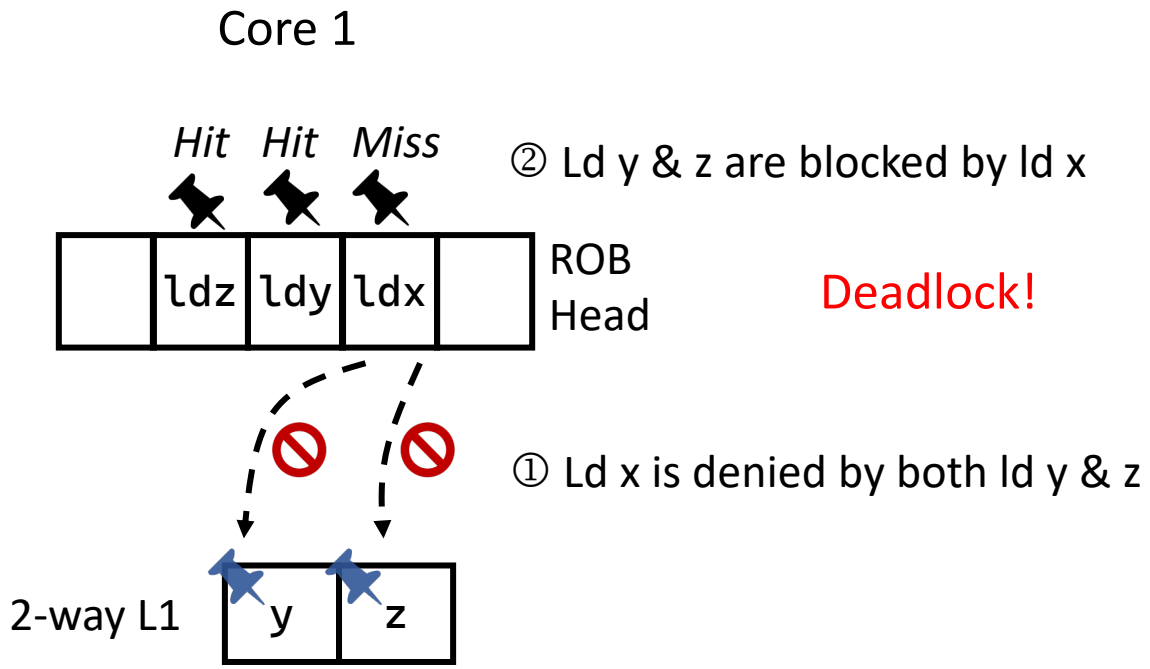
Guarantee Space in Cache & Directory

Insight: a core cannot pin more lines than a set can hold, otherwise, deadlocks may occur

```
1: ld x // L1 Miss
2: ld y // L1 Hit
3: ld z // L1 Hit
```

Line x, y, and z are mapped to the same L1 set

Ld x, y, and z are pinned before issuing



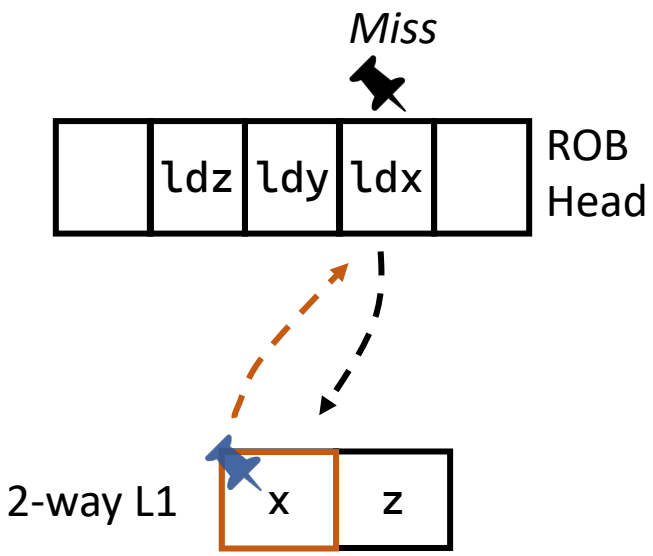
Two possible designs to avoid deadlock

Design 1: Late Pinning (LP)

Intuition: receive the data first (meaning it can find space in cache and directory sets), then pin the load

```
1: ld x // L1 Miss
2: ld y
3: ld z
```

Line x, y, and z are mapped to the same L1 set

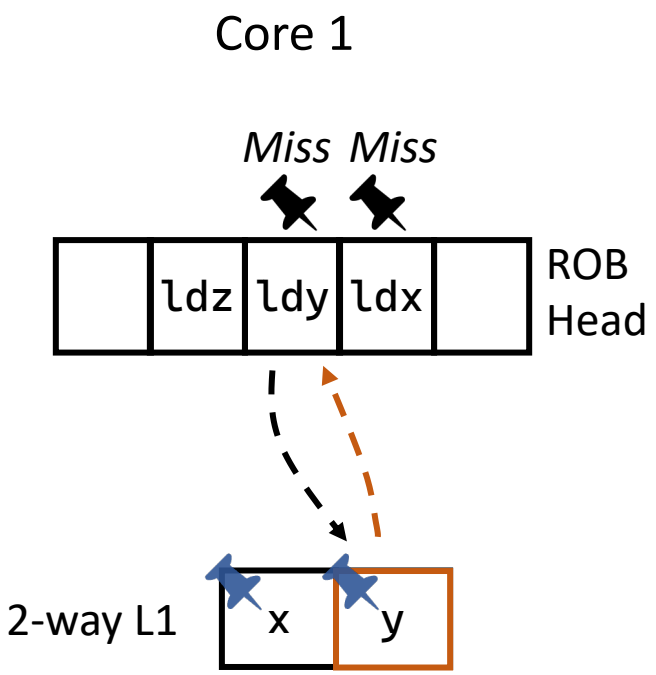


Design 1: Late Pinning (LP)

Intuition: receive the data first (meaning it can find space in cache and directory sets), then pin the load

```
1: ld x // L1 Miss
2: ld y
3: ld z
```

Line x, y, and z are mapped to the same L1 set

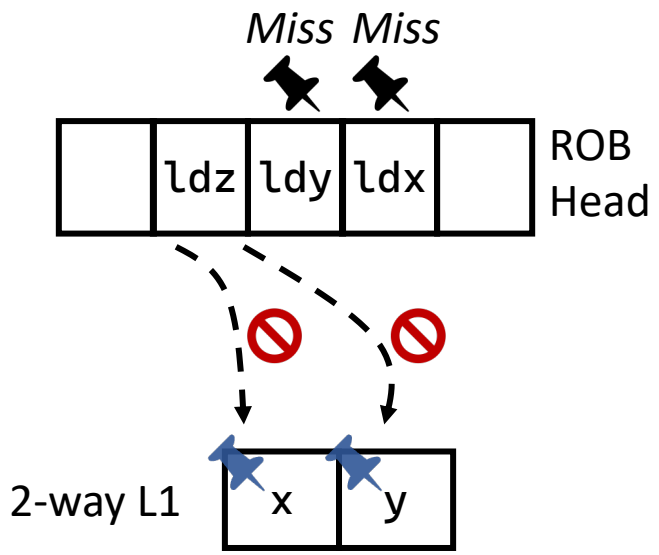


Design 1: Late Pinning (LP)

Intuition: receive the data first (meaning it can find space in cache and directory sets), then pin the load

```
1: ld x // L1 Miss
2: ld y
3: ld z
```

Line x, y, and z are mapped to the same L1 set



Serialized memory access, but it issues loads much earlier than it would in Fence

+ Simple hardware

- Low performance for programs with high L1 miss rates

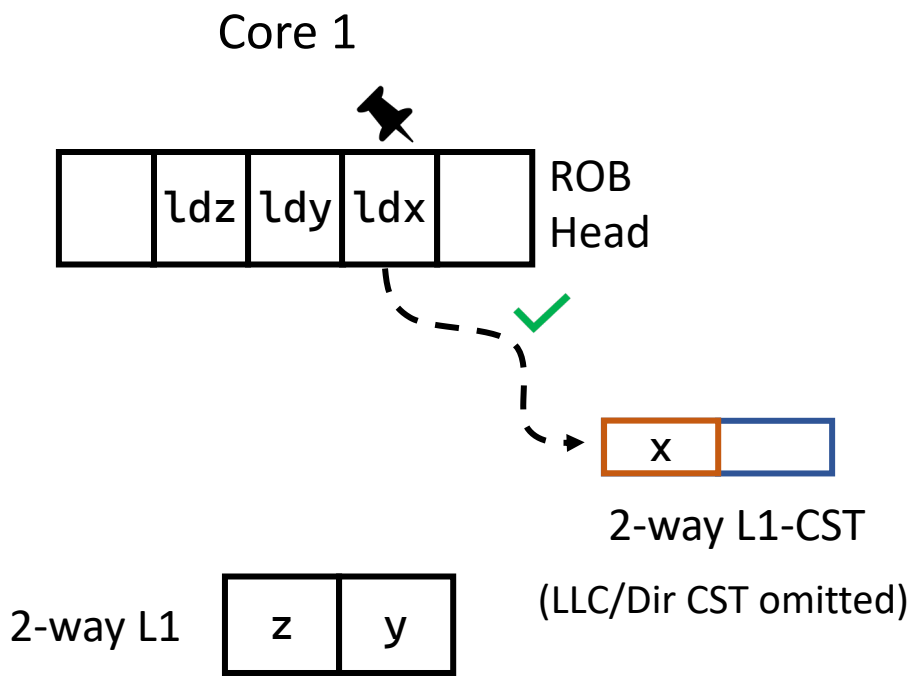
Ld z will stall until ld x retires and unpins x
(slow but safe from deadlock)

Design 2: Early Pinning (EP)

Intuition: add a small local hardware table called **Cache Shadow Table (CST)** in each core.
CST tracks, for **each set** in L1 and LLC/Dir, how many lines are pinned by **in-flight loads**

```
1: ld x
2: ld y
3: ld z
```

Line x, y, and z are mapped to the same L1 set (and the **same CST set**)



Checks before pinning a load:

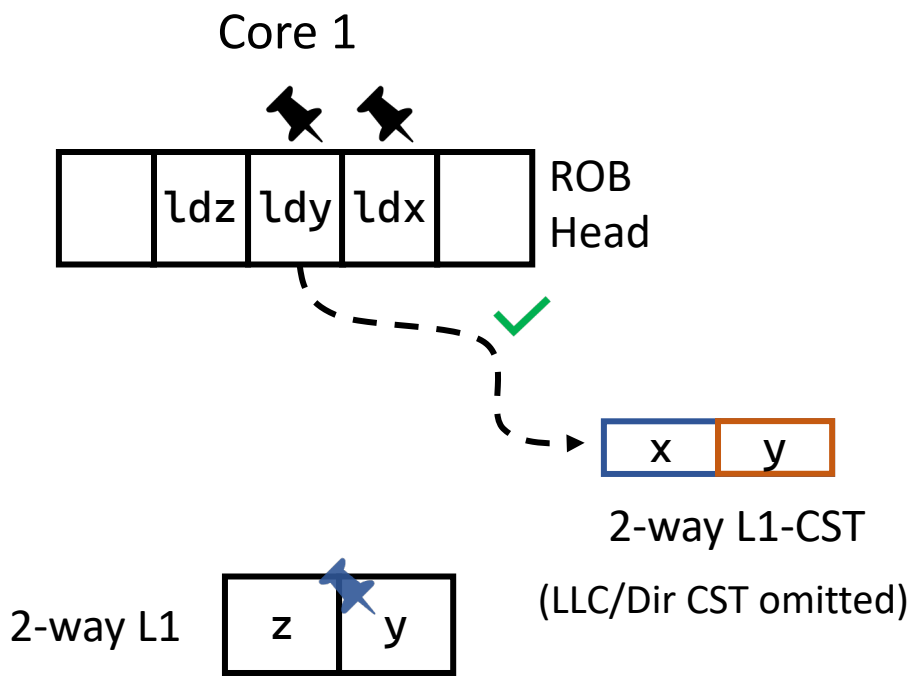
1) Hardware determines the L1 set and the LLC/Dir set where the line maps
2) Access CST sets and check if such sets can hold the **additional** pinned line
3) Pin the load if find space in each cache level and directory

Design 2: Early Pinning (EP)

Intuition: add a small local hardware table called **Cache Shadow Table (CST)** in each core.
CST tracks, for **each set** in L1 and LLC/Dir, how many lines are pinned by **in-flight loads**

```
1: ld x
2: ld y
3: ld z
```

Line x, y, and z are mapped to the same L1 set (and the **same CST set**)



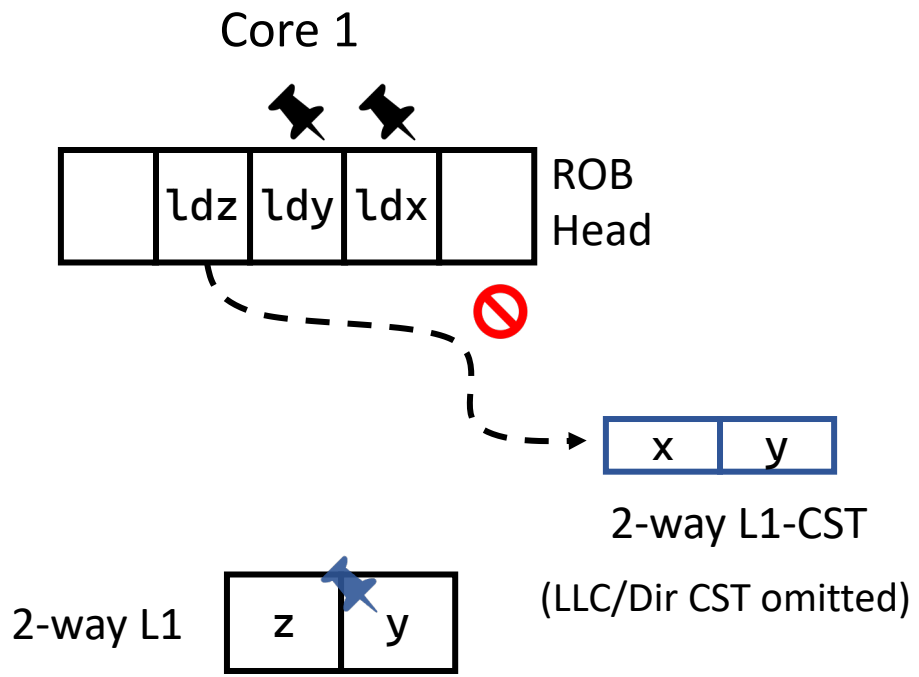
Checks before pinning a load:

1) Hardware determines the L1 set and the LLC/Dir set where the line maps
2) Access CST sets and check if such sets can hold the **additional** pinned line
3) Pin the load if find space in each cache level and directory

Design 2: Early Pinning (EP)

Intuition: add a small local hardware table called **Cache Shadow Table (CST)** in each core.
CST tracks, for **each set** in L1 and LLC/Dir, how many lines are pinned by **in-flight loads**

```
1: ld x
2: ld y
3: ld z
```



Line x, y, and z are mapped to the same L1 set (and the **same CST set**)

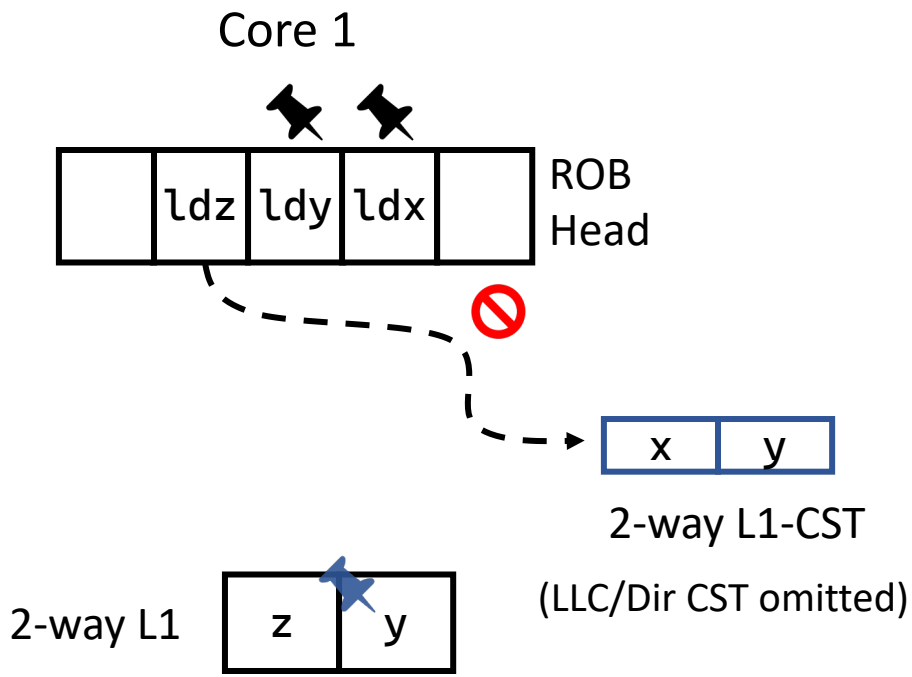
The L1-CST set is full, ldz will remain unpinned
(slow but safe from deadlock)

CST for a shared cache requires small changes in its geometry (*detailed in the paper*)

Design 2: Early Pinning (EP)

Intuition: add a small local hardware table called **Cache Shadow Table (CST)** in each core.
CST tracks, for **each set** in L1 and LLC/Dir, how many lines are pinned by **in-flight loads**

```
1: ld x
2: ld y
3: ld z
```



Line x, y, and z are mapped to the same L1 set (and the **same CST set**)

All independent loads in parallel (assume enough space)

+ Parallelized access

- Require CSTs

Pinned Loads Summary

Mechanism: defer invalidations and prevent evictions to lines that are read by a pinned load

For a load *L*, it can be pinned if:

Security

- *L* has met all the conditions to reach the VP **except for guaranteeing L itself will not cause an MCV**

Avoid Starvation

- The line that *L* tries to pin is not in **Cannot-Pin Table (CPT)**

Avoid Deadlock

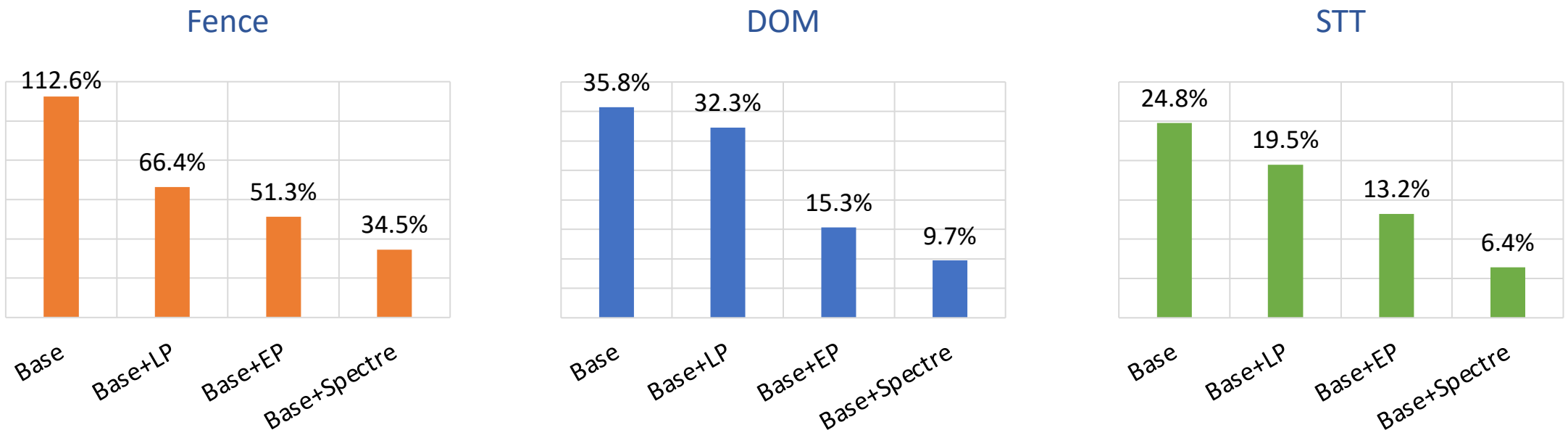
- Write buffer has enough entries for all the yet-to-complete stores older than *L* *(detailed in the paper)*
- Guaranteeing space in cache & directory
  - *L* has received the data, or ⇒ **Late Pinning (LP)**
  - CSTs report enough space in L1 cache and LLC/Dir ⇒ **Early Pinning (EP)**

Performance Evaluation

Workloads: single-threaded (SPEC17) and parallel (SPLASH2 + PARSEC)

Defenses: Fence, DOM, and STT

Geo. Mean Execution Overhead over a Conventional Unsafe Core (SPEC17)



≈50% overhead reduction (with EP)

Conclusions

- Under the Comprehensive model, most execution overhead is caused by ensuring no memory consistency violations (MCVs)

- *Pinned Loads* is a general technique to reduce the execution overhead of speculative-execution defense schemes by making loads invulnerable to MCVs as early as possible

- *Pinned Loads* can substantially reduce execution overhead of many existing defense schemes by ≈50%

Open Source: https://github.com/zzrcxb/PinnedLoads

Pinned Loads: Taming Speculative Loads in Secure Processors

Zirui Neil Zhao, Houxiang Ji, Adam Morrison, Darko Marinov, Josep Torrellas

University of Illinois Tel Aviv University

ziruiz6@illinois.edu

ASPLOS'22 – **Session 3B**

ILLINOIS

TEL AVIV UNIVERSITY