Selfish-LRU:

Preemption-Aware Caching for *Predictability* and *Performance*

Jan Reineke	Saarland University, Germany
Sebastian Altmeyer	University of Amsterdam, Netherlands
Daniel Grund	Thales Germany
Sebastian Hahn	Saarland University, Germany
Claire Maiza	INP Grenoble, Verimag, France

20th IEEE Real-Time and Embedded Technology and Applications Symposium
April 15-17, 2014
Berlin, Germany

Context: Preemptive Scheduling

Non-preemptive Execution:

Task 1

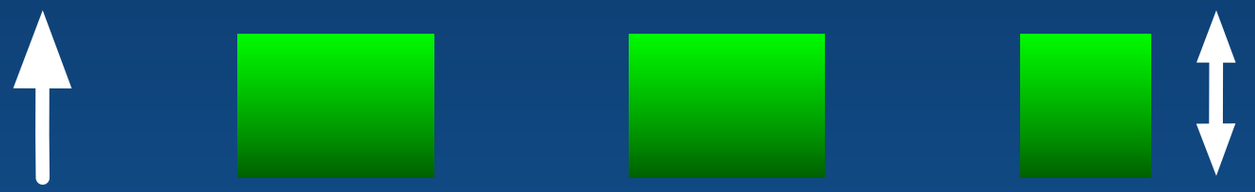
Task 2

Context: Preemptive Scheduling

Preemptive Execution:

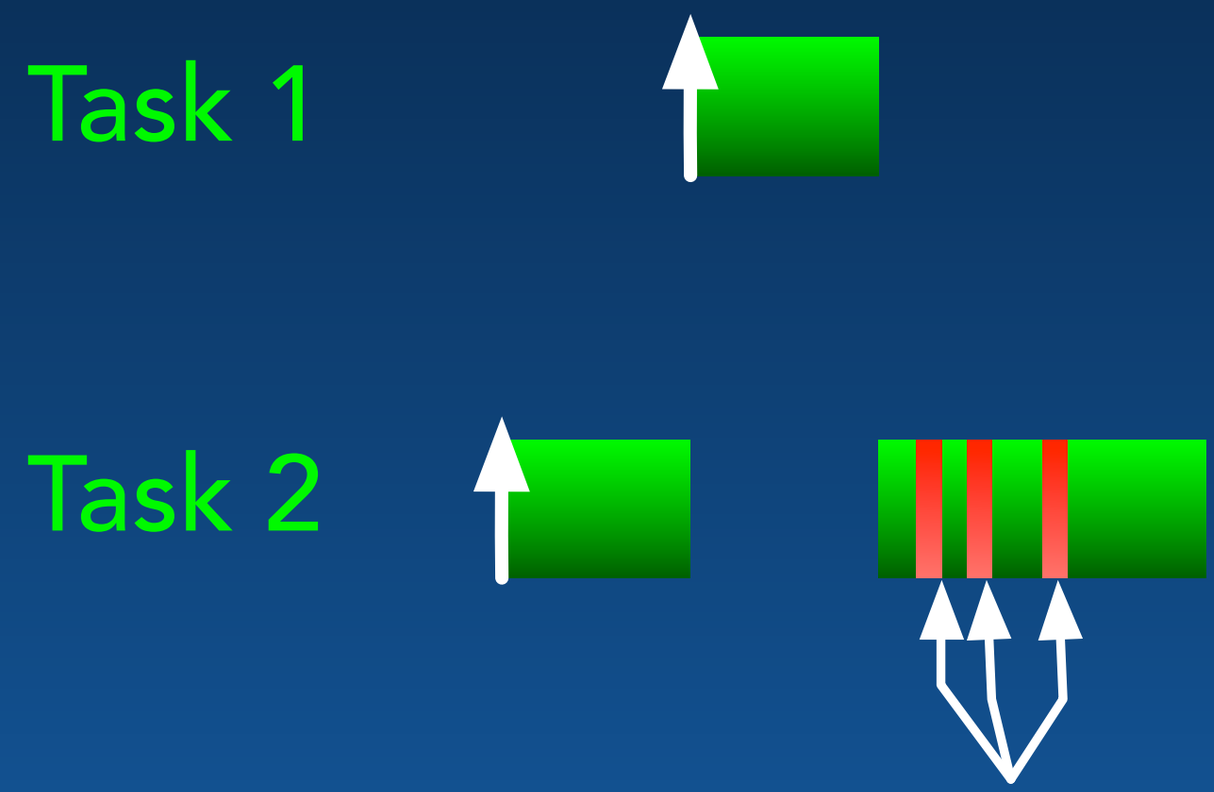
Task 1

Task 2



Caveat: Preemptions are **not** free!

Preemptive Execution:



Cache-Related Preemption Delay (CRPD)

Contribution of this paper

Selfish-LRU: a new cache replacement policy, that

➡ Increases performance by *reducing* the CRPD

➡ *Simplifies* static analysis of the CRPD

Contribution of this paper

Selfish-LRU: a new cache replacement policy, that

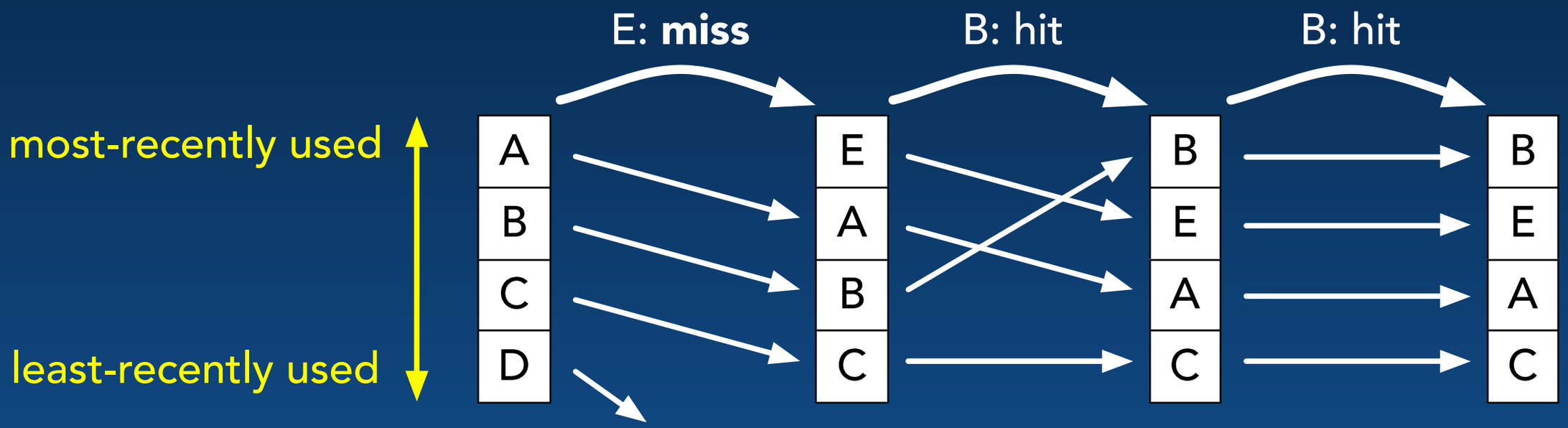
➡ Increases performance by *reducing* the CRPD

➡ *Simplifies* static analysis of the CRPD

Selfish-LRU is a *preemption-aware* variant of *least-recently used* (LRU)

Least-Recently Used (LRU)

"Replace data that has not been used for the longest time"



➡ Usually works well due to temporal locality

CRPD Example under LRU Replacement

Assume simple **preempted** task:

```
for i in [1,10]:
     do something()
```

```
for i in [1,10]:
     access A
     access B
     access C
     access D
```

CRPD Example under LRU Replacement

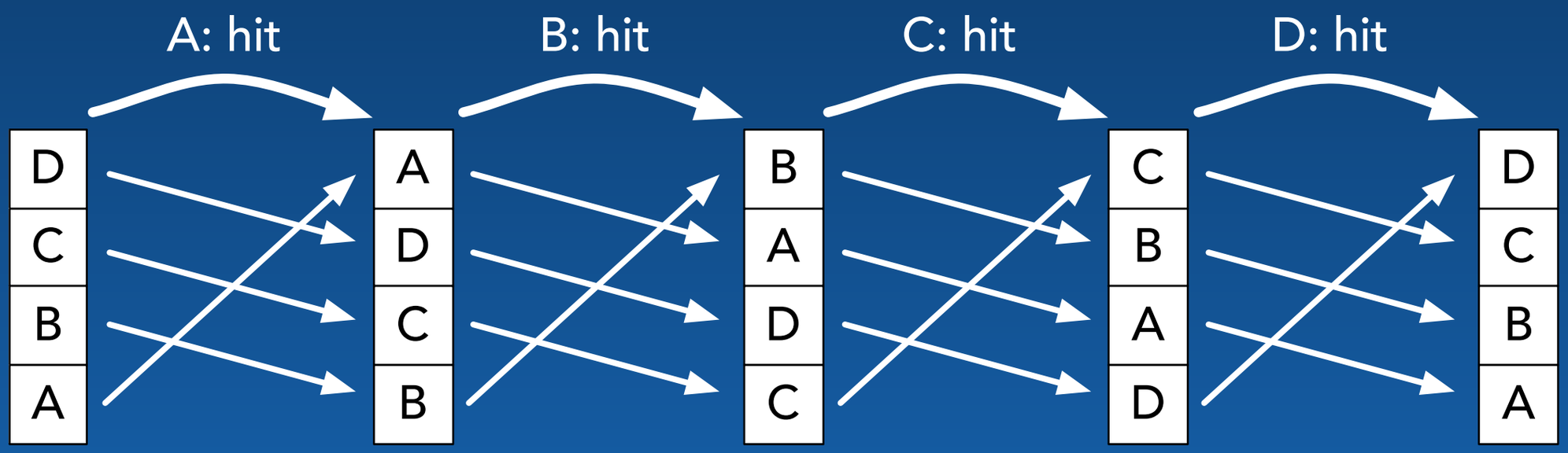
Assume simple **preempted** task:

```
for i in [1,10]:
     do something()
```

→

```
for i in [1,10]:
     access A
     access B
     access C
     access D
```

Without preemption (after warmup): **0 misses**

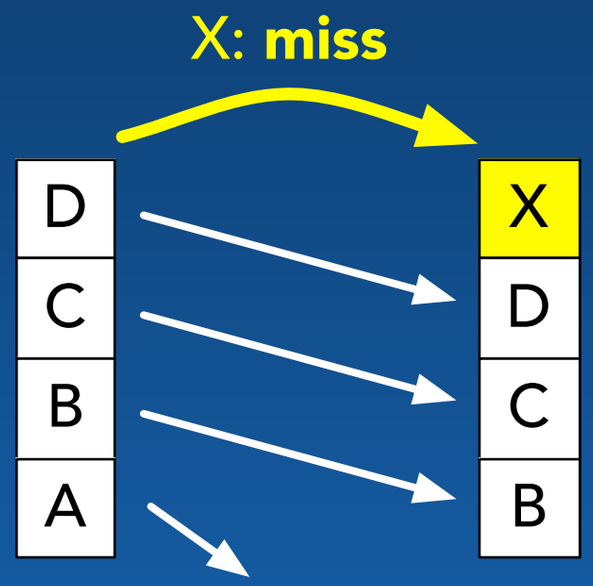


CRPD Example under LRU Replacement

Assume simple **preempting** task:

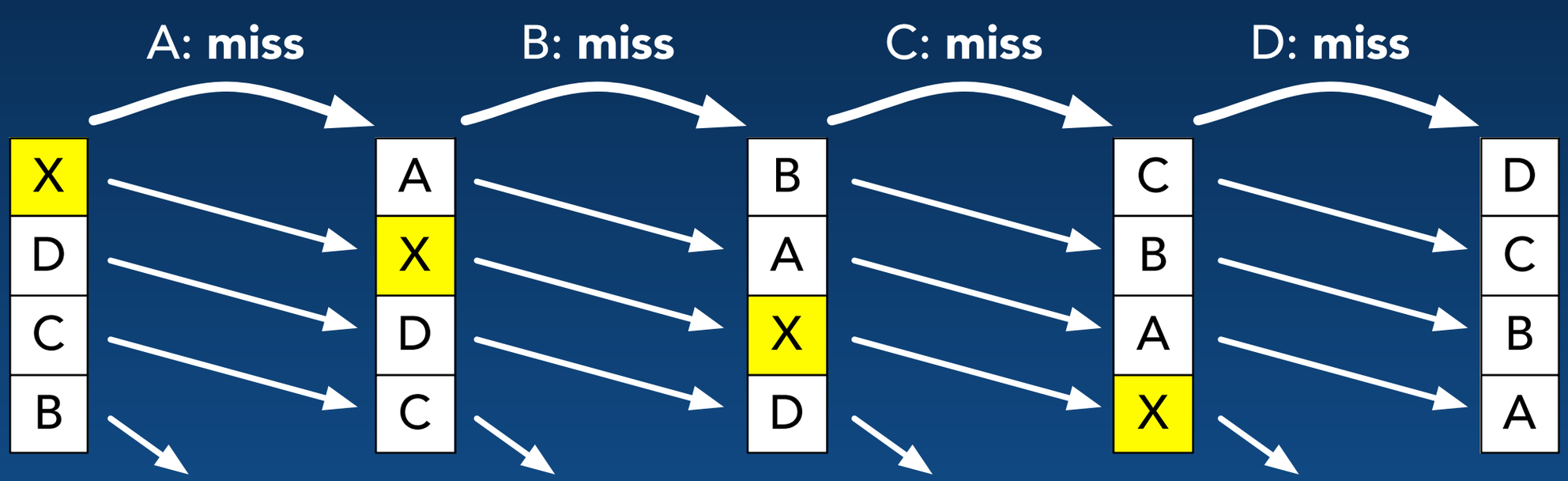
```
do something_else()  →  access X
```

Preemption between loop iterations: 1 access



CRPD Example under LRU Replacement

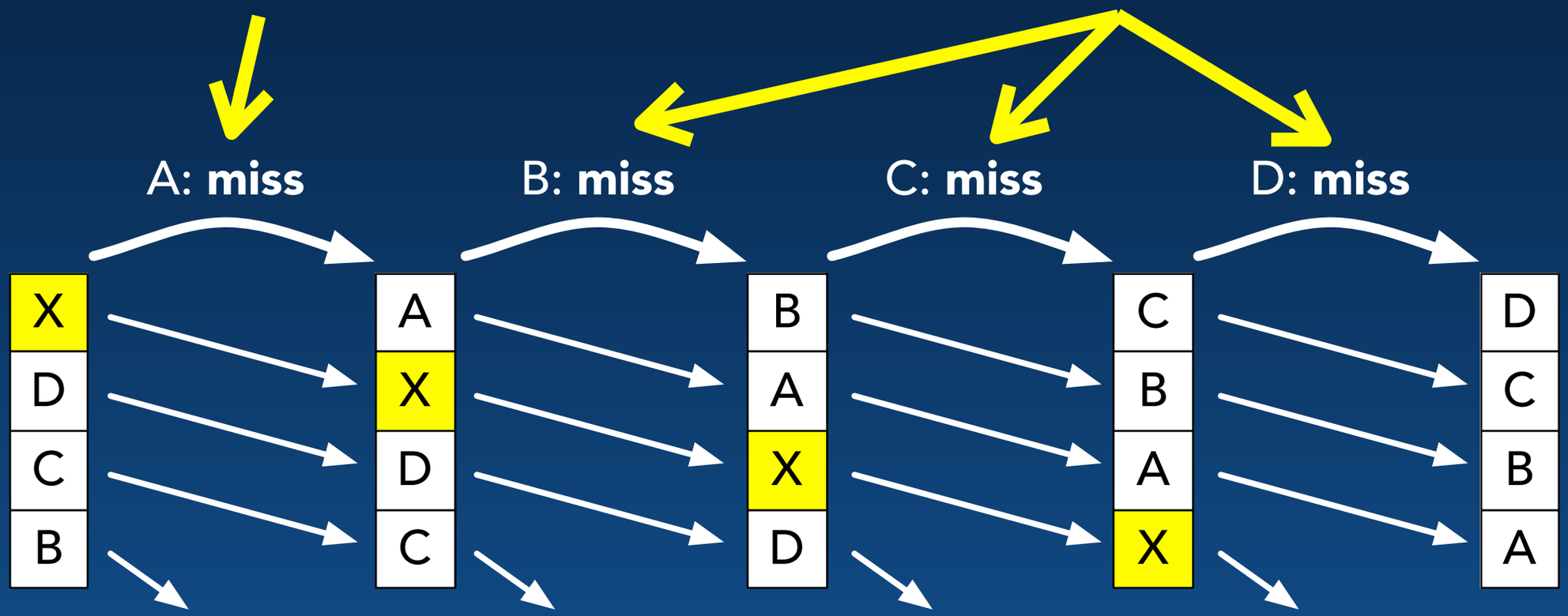
First loop iteration after preemption: **4 misses**



CRPD Example under LRU Replacement:
Two types of misses related to preemption

1. Replaced Misses

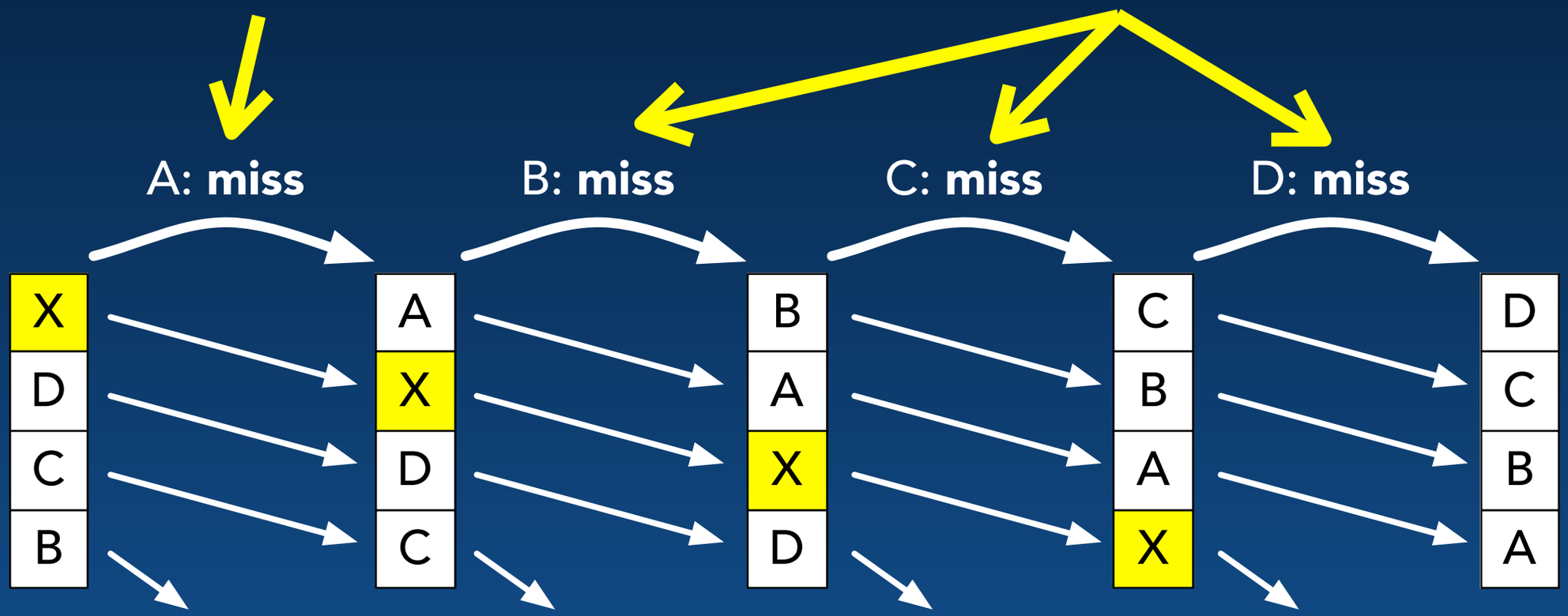
2. Reordered Misses



CRPD Example under LRU Replacement:
Two types of misses related to preemption

1. Replaced Misses

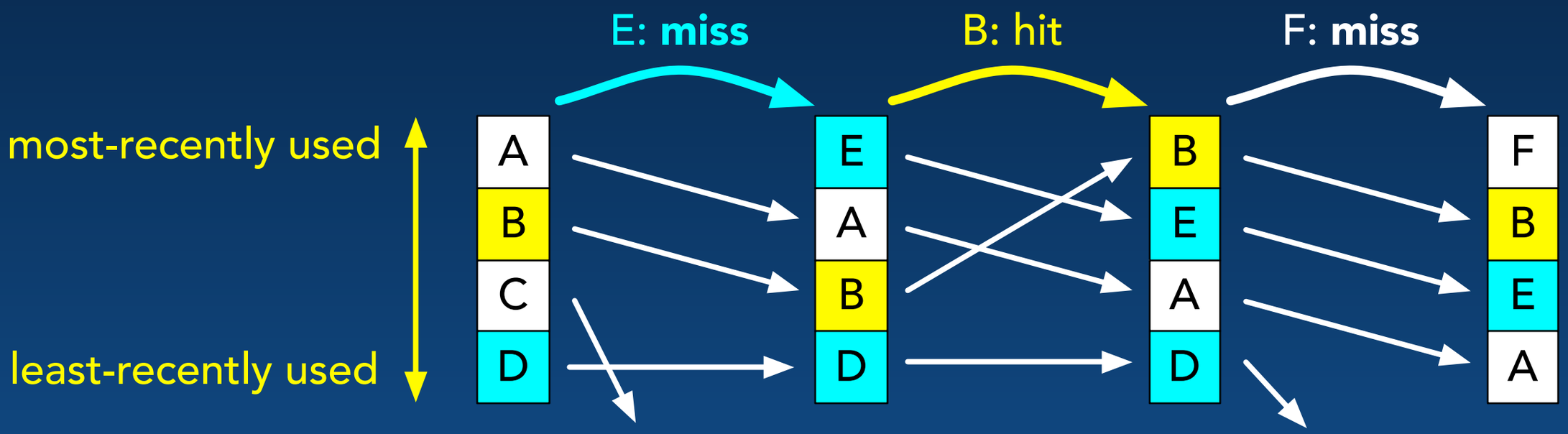
2. Reordered Misses



Liu et al., PACT 2008: reordered misses account for **10%** to **28%** of all preemption-related misses

Selfish-LRU: Idea

Prioritize blocks of currently running task:



Intuition:
"Memory blocks of currently running task more likely to be accessed again soon."

Selfish-LRU: CRPD Example Revisited

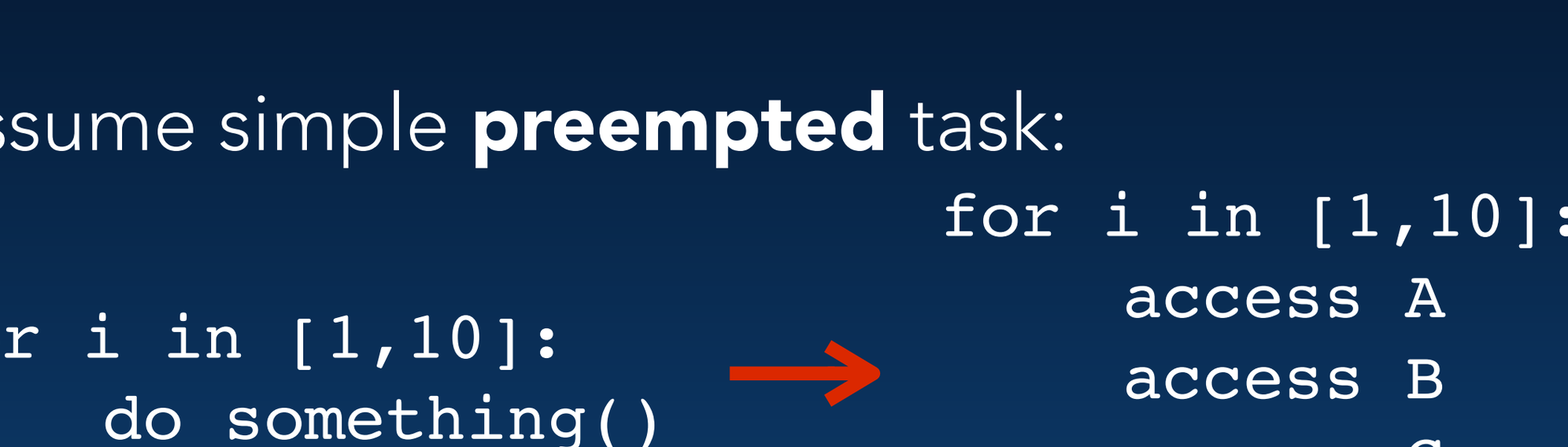
Assume simple **preempted** task:

```
for i in [1,10]:
     do something()
```

→

```
for i in [1,10]:
     access A
     access B
     access C
     access D
```

Without preemption (after warmup): **0 misses**



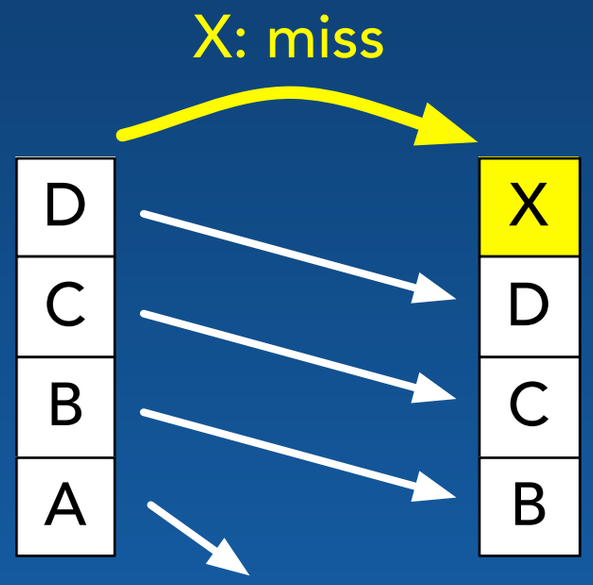
➡ Same behavior as LRU

Selfish-LRU: CRPD Example Revisited

Assume simple **preempting** task:

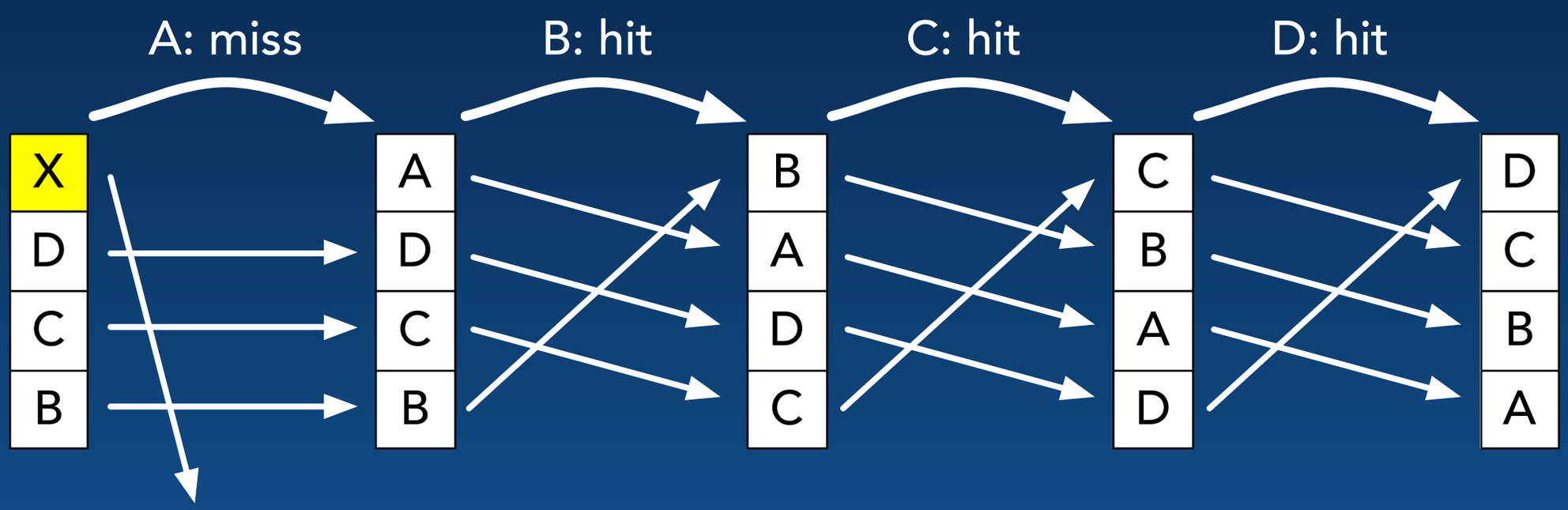
```
do something_else()  →  access X
```

Preemption between loop iterations: 1 access



Selfish-LRU: CRPD Example Revisited

First loop iteration after preemption: **1 miss**



➡ No reordering misses

Selfish-LRU: Properties

Selfish-LRU: Properties

Property 1:

Selfish-LRU does not exhibit reordering misses.

➡ Often: *smaller* CRPD

➡ *Simplifies* static analysis of the CRPD

Selfish-LRU: Properties

Property 1:

Selfish-LRU does not exhibit reordering misses.

➡ Often: *smaller* CRPD

➡ *Simplifies* static analysis of the CRPD

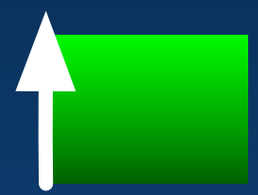
Property 2:

In non-preempted execution, Selfish-LRU = LRU.

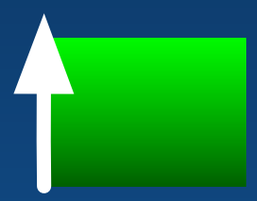
➡ No change in "regular" WCET analysis

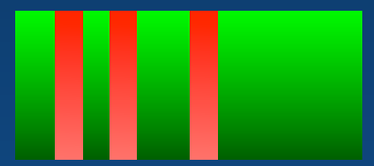
Selfish-LRU: CRPD Analysis

Preempting



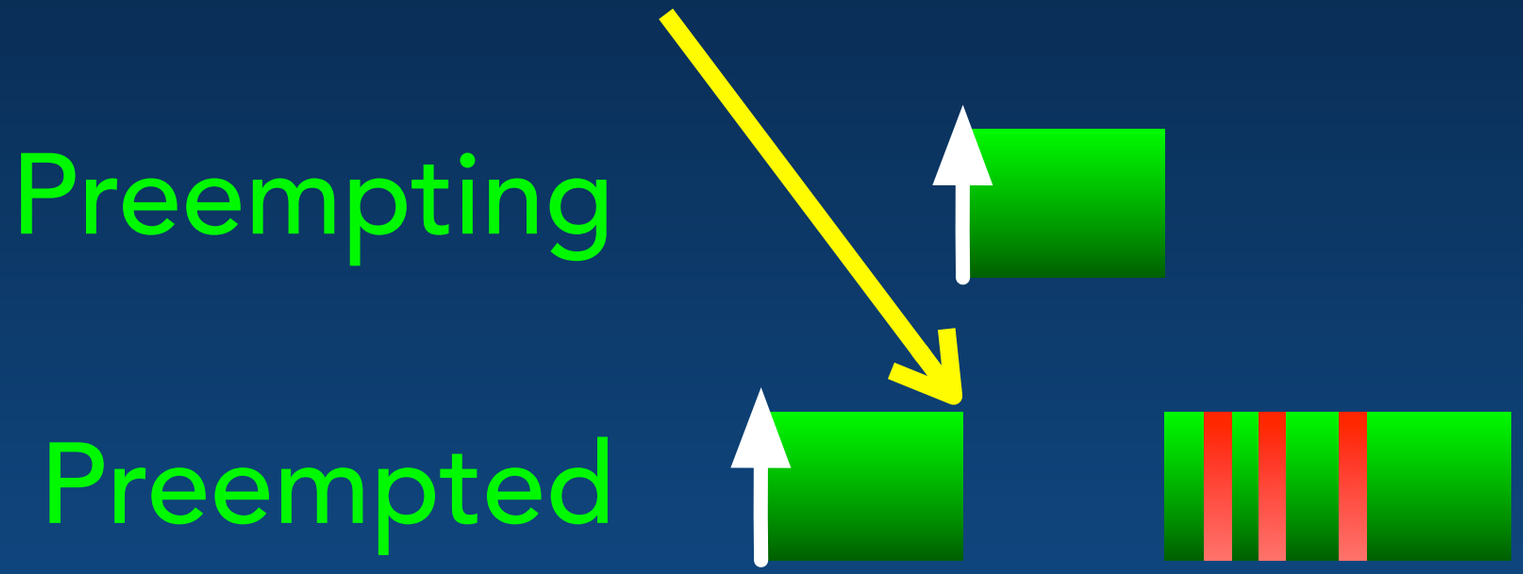
Preempted





Selfish-LRU: CRPD Analysis

1. Number of **useful cache blocks (UCBs)**?



Selfish-LRU: CRPD Analysis

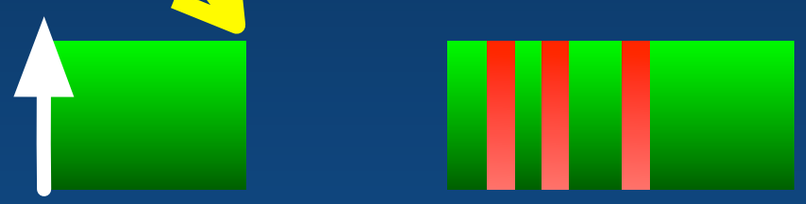
1. Number of **useful cache blocks (UCBs)**?

2. Number of **evicting cache blocks (ECBs)**?

➡ Smaller Bound

Preempting

Preempted

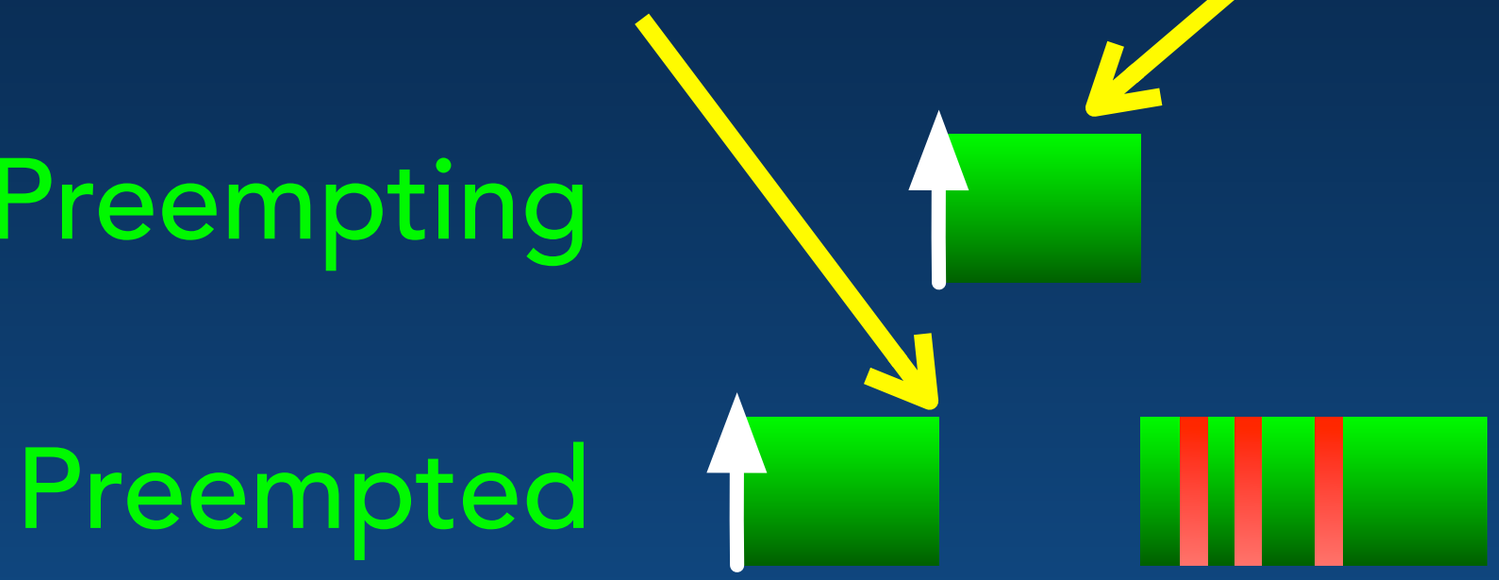


Selfish-LRU: CRPD Analysis

1. Number of **useful cache blocks (UCBs)**?

2. Number of **evicting cache blocks (ECBs)**?

➡ Smaller Bound



3. Combination of ECBs and UCBs based on **Resilience**

➡ Simplified and Smaller Bound

Selfish-LRU: Implementation

Required modifications:

- Manage **task ids (TID)** in operating system
- Make TID available to cache in **TID register**
- **Augment cache lines** with TID of "owner" task
  - Conservative estimate: < 3% space overhead
- Modified replacement logic

Similar to virtually-addressed caches

Experimental Evaluation

Experimental Evaluation

Main goal:

Compare Selfish-LRU with LRU in terms of **performance** and **predictability**

Experimental Evaluation

Main goal:

Compare Selfish-LRU with LRU in terms of **performance** and **predictability**

➡ Modified MPARM simulator

➡ CRPD analyses implemented in AbsInt's aiT

Experimental Evaluation

Main goal:
Compare Selfish-LRU with LRU in terms of **performance** and **predictability**

➡ Modified MPARM simulator
➡ CRPD analyses implemented in AbsInt's aiT

Secondary goal: (see paper for details)
Compare CRPD approach with cache partitioning

Experimental Evaluation: Benchmarks and Cache Configuration

Benchmarks:

- Four of the largest Mälardalen benchmarks
- Four models from the SCADE distribution
- Two SCADE models from an embedded systems course

Cache configuration:

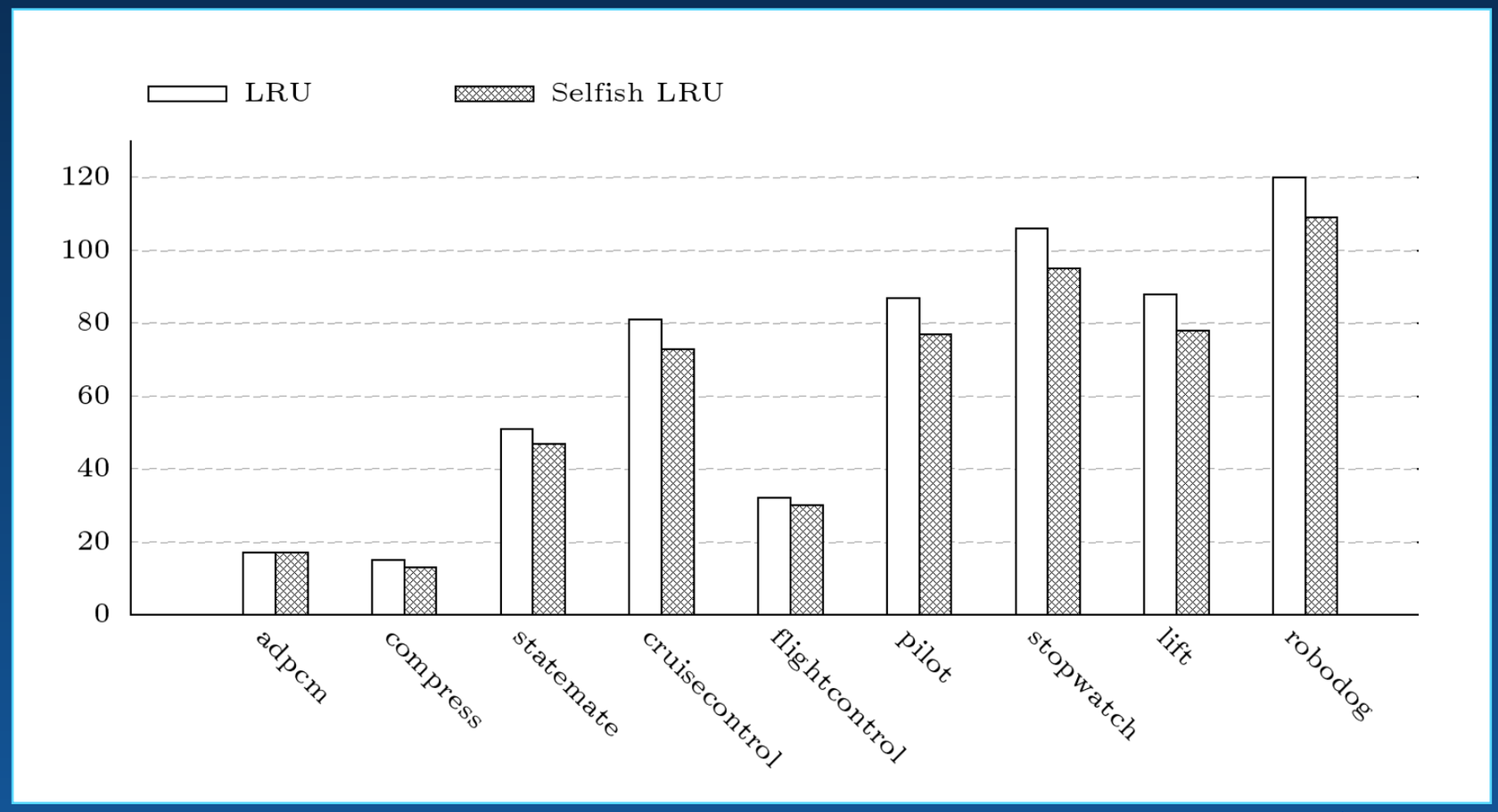
Capacity:	2 KB, 4 KB, 8 KB
Associativity:	4, 8
Number of sets:	32, 64, 128

Experimental Evaluation: Simulation Results, "Large" Preempting Task

Cache configuration:

Capacity: 2 KiB, Associativity 4, Number of sets: 32

Measured number of additional misses



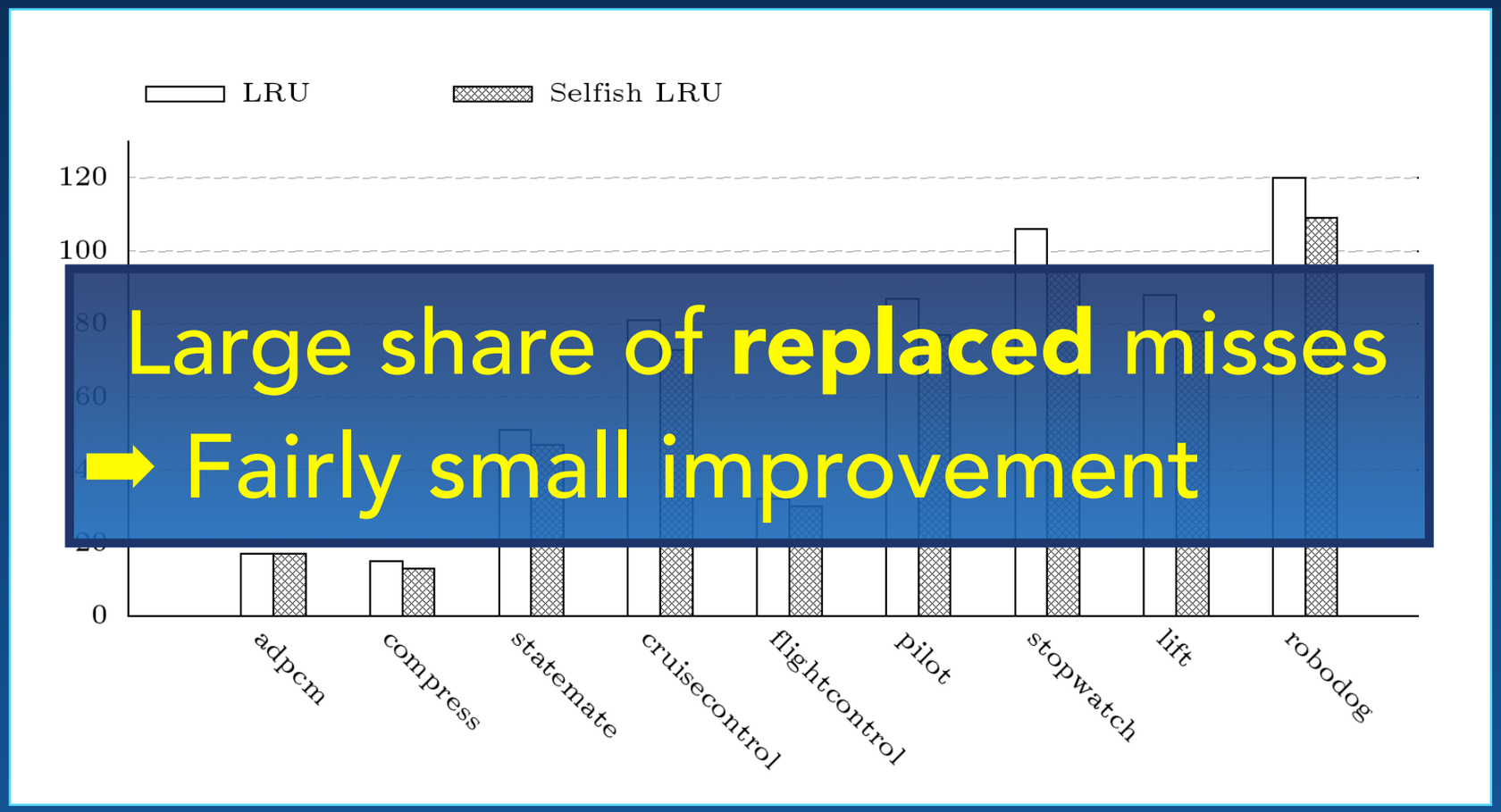
Preempted Tasks

Experimental Evaluation: Simulation Results, "Large" Preempting Task

Cache configuration:

Capacity: 2 KiB, Associativity 4, Number of sets: 32

Measured number of additional misses



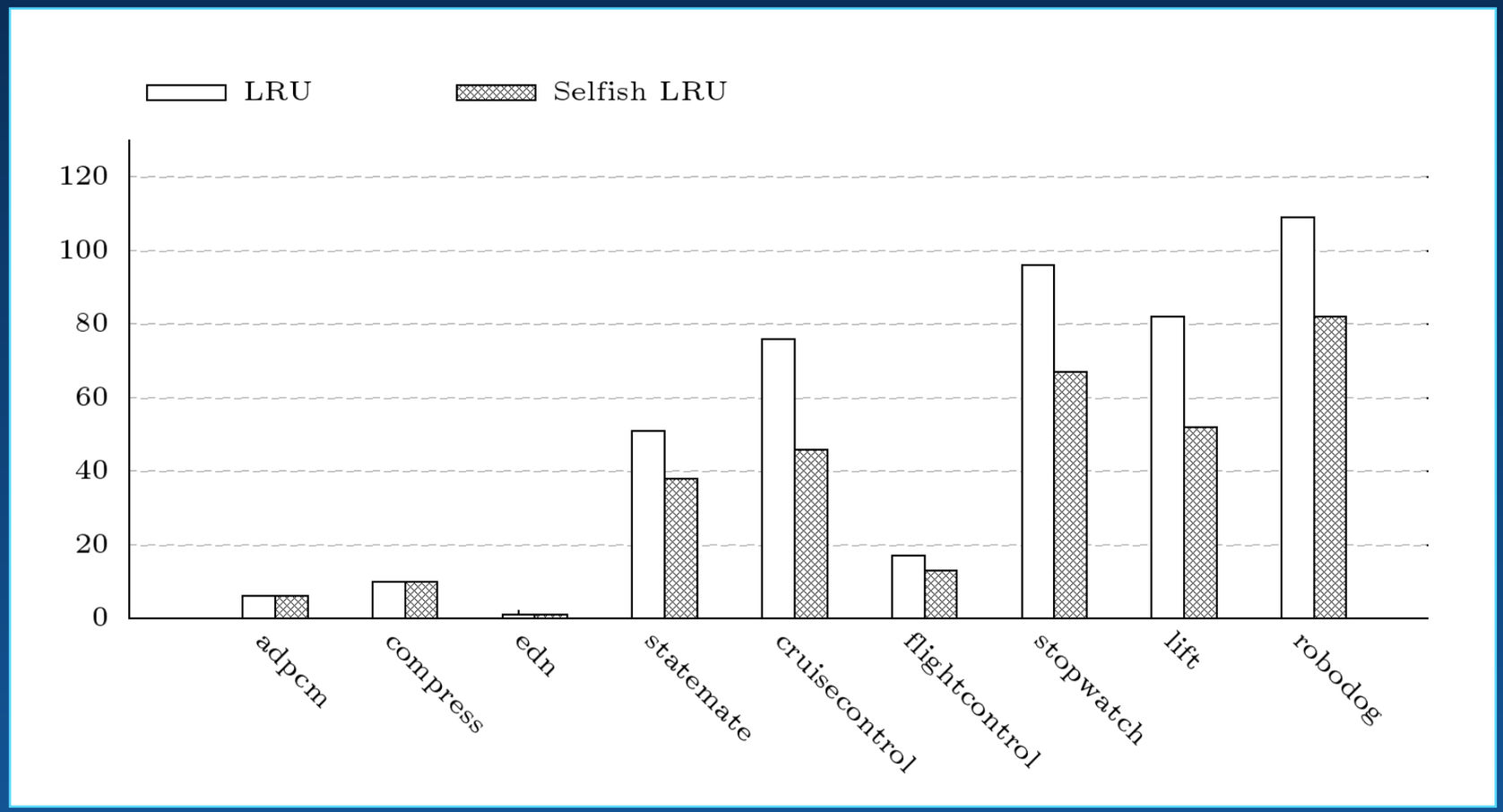
Preempted Tasks

Experimental Evaluation: Simulation Results, "Small" Preempting Task

Cache configuration:

Capacity: 2 KiB, Associativity 4, Number of sets: 32

Measured number of additional misses



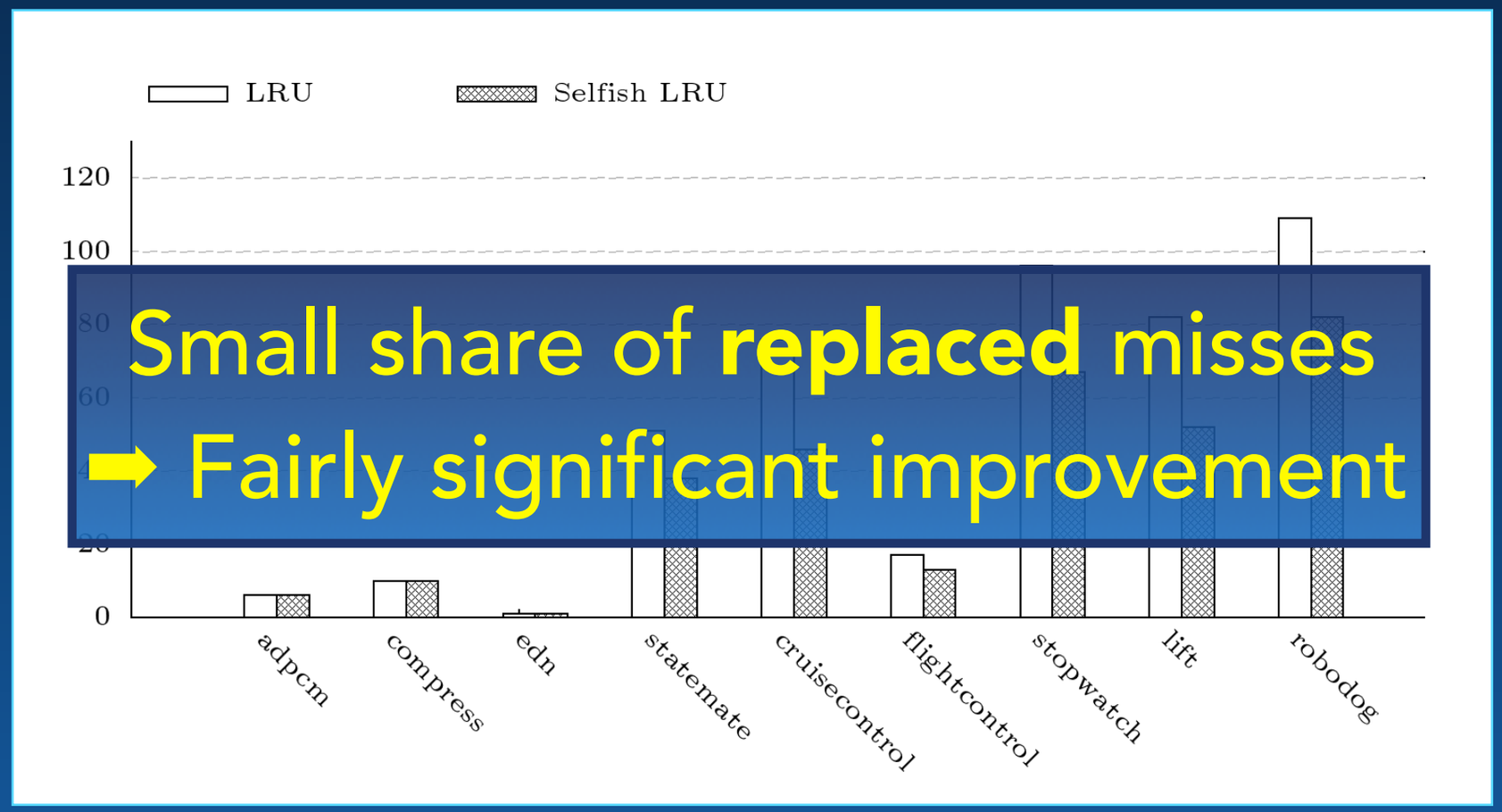
Preempted Tasks

Experimental Evaluation: Simulation Results, "Small" Preempting Task

Cache configuration:

Capacity: 2 KiB, Associativity 4, Number of sets: 32

Measured number of additional misses



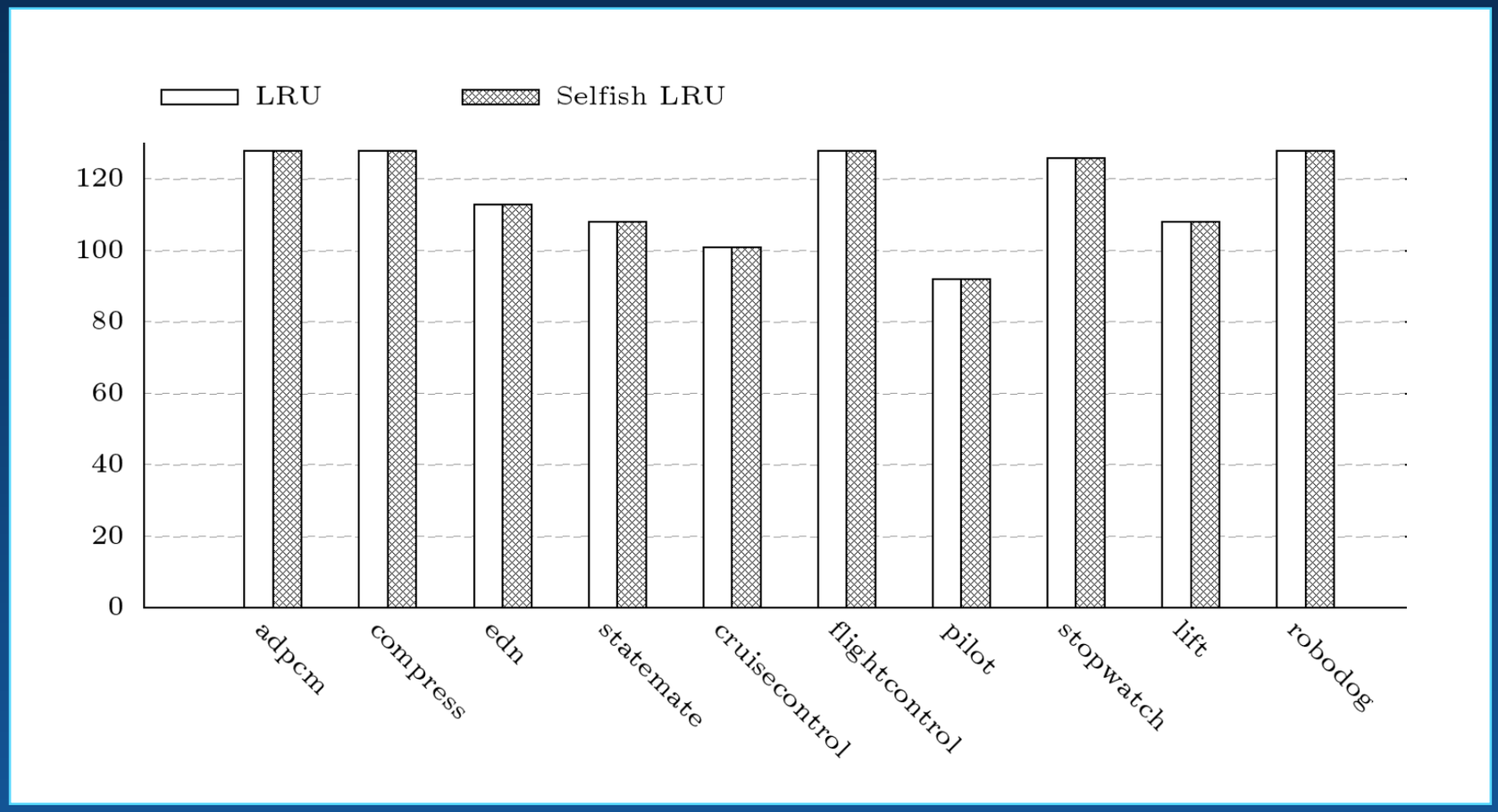
Preempted Tasks

Experimental Evaluation: Analysis Results, "Large" Preempting Task

Cache configuration:

Capacity: 2 KiB, Associativity 4, Number of sets: 32

Bound on number of additional misses



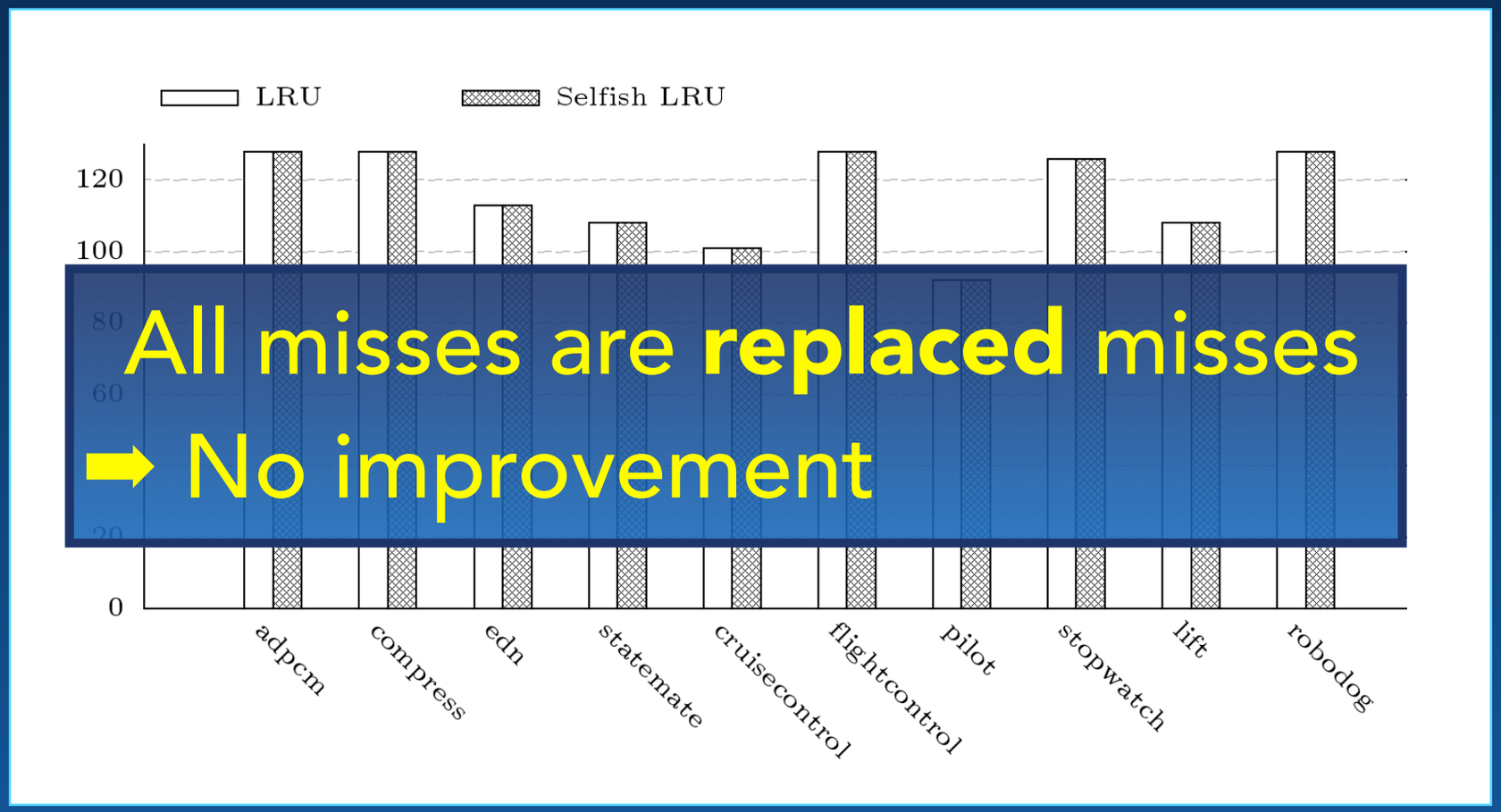
Preempted Tasks

Experimental Evaluation: Analysis Results, "Large" Preempting Task

Cache configuration:

Capacity: 2 KiB, Associativity 4, Number of sets: 32

Bound on number of additional misses



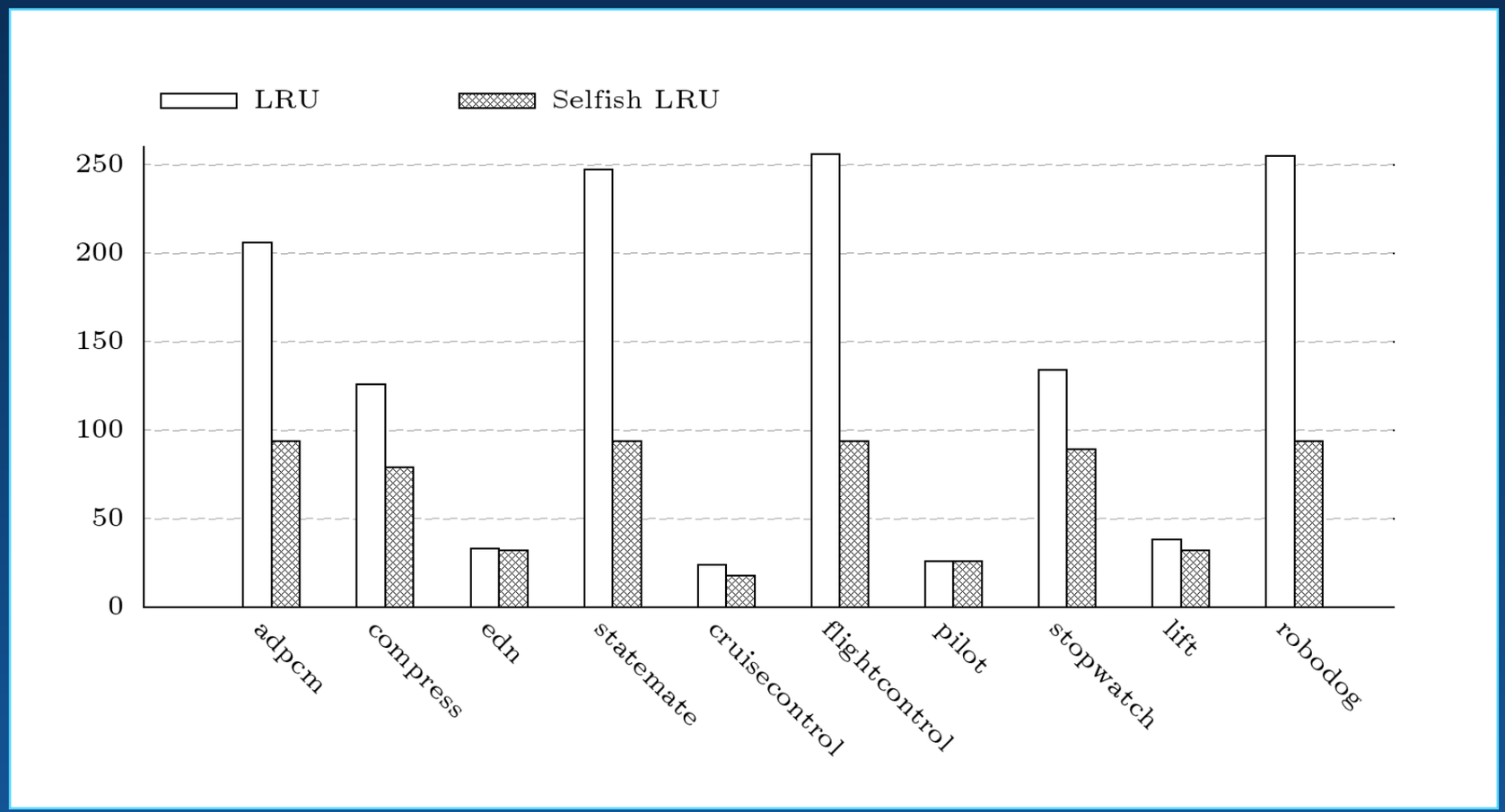
Preempted Tasks

Experimental Evaluation: Analysis Results, "Small" Preempting Task

Cache configuration:

Capacity: 4 KiB, Associativity 8, Number of sets: 32

Bound on number of additional misses



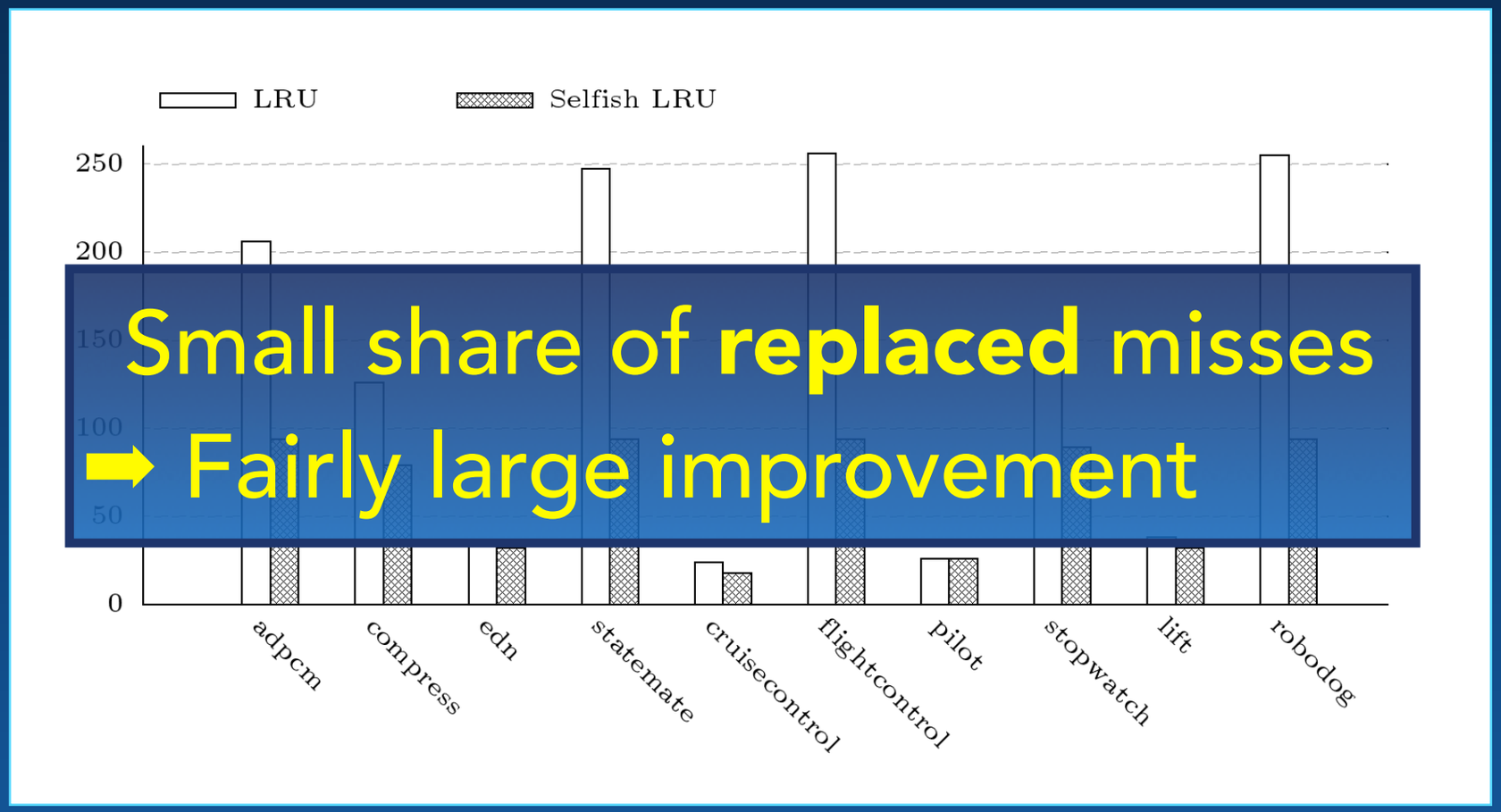
Preempted Tasks

Experimental Evaluation: Analysis Results, "Small" Preempting Task

Cache configuration:

Capacity: 4 KiB, Associativity 8, Number of sets: 32

Bound on number of additional misses



Preempted Tasks

Summary and Future Work

Selfish-LRU eliminates reordered misses:

- ➡ Increases performance by *reducing* the CRPD
- ➡ *Simplifies* static analysis of the CRPD
- ➡ Large improvements for small preempting tasks like interrupt handlers

Summary and Future Work

Selfish-LRU eliminates reordered misses:

➡ Increases performance by *reducing* the CRPD

➡ *Simplifies* static analysis of the CRPD

➡ Large improvements for small preempting tasks like interrupt handlers

Apply same idea in **shared caches** in multi-cores?