Presented at MECO 2020 and CPSIoT 2020, Budva, Montenegro

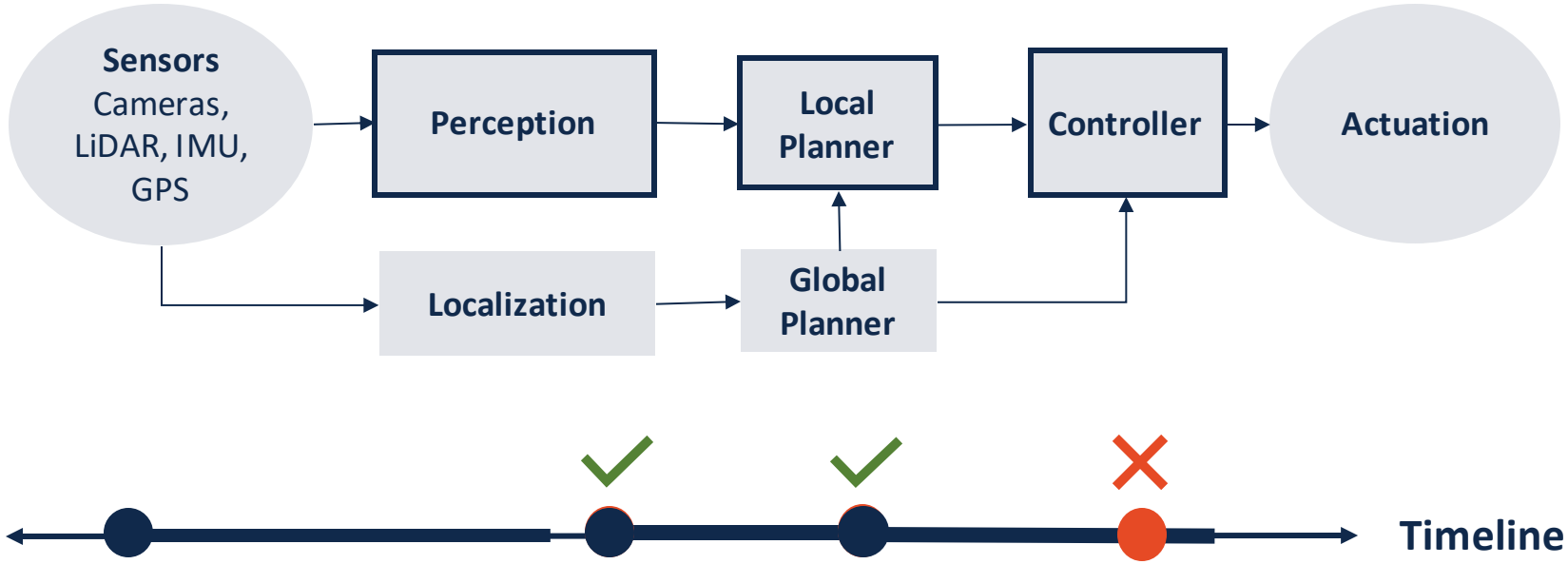
Reconciling Predictability and Coherent Caching

Ayoosh Bansal, **Jayati Singh**, Yifan Hao, Jen-Yang Wen, Renato Mancuso, Marco Caccamo

ILLINOIS

BOSTON UNIVERSITY

HARD REAL-TIME SYSTEMS



Tasks in a sample autonomous pipeline. Adapted from Yurtsever, Ekim, et al. "A survey of autonomous driving: Common practices and emerging technologies."

Key requirement of real-time (RT) systems. All tasks should finish execution before the deadline.

WCET < deadline.

HARD REAL-TIME SYSTEMS AND MULTI-CORE PROCESSORS

Porting legacy single-threaded RT apps

Solution I. Deactivate all but one core.

✓ Easy to port.

✗ Expensive.

Solution II. N-core processor ≡ N single core processors. *Mancuso, Renato, et al. "WCET (m) estimation in multi-core systems using single core equivalence."*

Supporting new multi-threaded RT apps

Challenges.

Multi-threaded applications rely on **cache coherence** to ensure correctness.

How does cache coherence complicate WCET estimation for tasks?

Contribution 1.

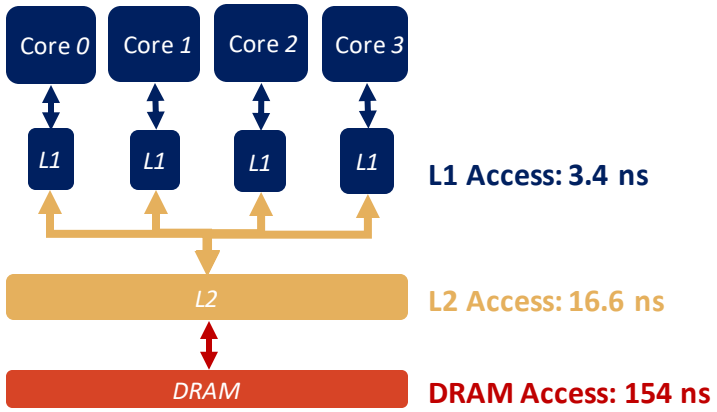
Solutions.

How do we solve the complications introduced by coherence for RT tasks?

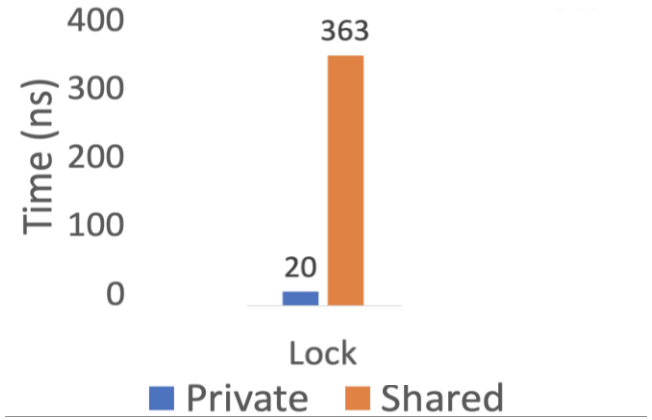
Contribution 2.

A MOTIVATING MICROBENCHMARK

Hardware Setup. ARMv8-A Cortex-A53*



Results*



Software Setup. Average latency to acquire-release spinlock

```
while(__sync_lock_test_and_set(&lck,1)){};
__sync_lock_release (&lck)
```

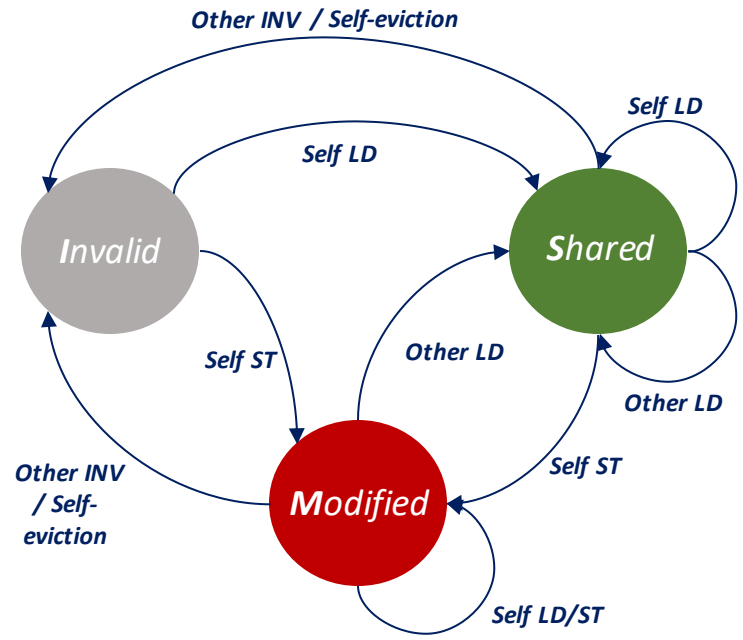
Why? Dirty Miss

Observation. The coherence overhead leads to a delay even **greater than DRAM latency.**

*Refer to our paper for other results on a 14-core Intel Xeon E5-2658. Code available on https://gitlab.engr.illinois.edu/rtesl/inc-oc

DIRTY MISS IN CACHE COHERENCE

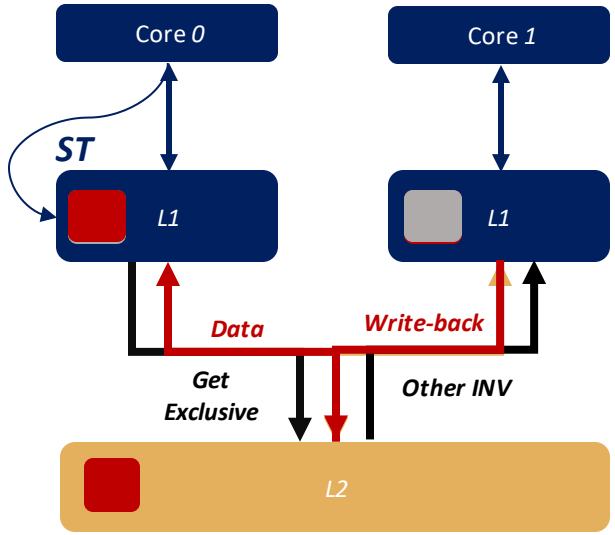
States in MSI Protocol



LD Load
ST Store

INV Invalidate
Self (Other) Request from same (other) core

Transitions in a dirty miss



1. *Longer than L2 access latency.*
2. Sometimes, *longer than DRAM access latency.*

POSSIBLE SOLUTIONS

Solution I. Mark all accesses where dirty misses are a possibility as *uncacheable.*

✓ No changes Linux kernel and ARM ISA
✓ Supported by ARM processor hardware.
✗ Significantly worsens average execution time.

Solution II. Software coherence only.

✓ No hardware changes.
✗ Software-programmer to maintain coherence.

Solution III. New coherence protocol. Hassan, Mohamed, et al. "Predictable cache coherence for multi-core real-time systems." (PMSI)

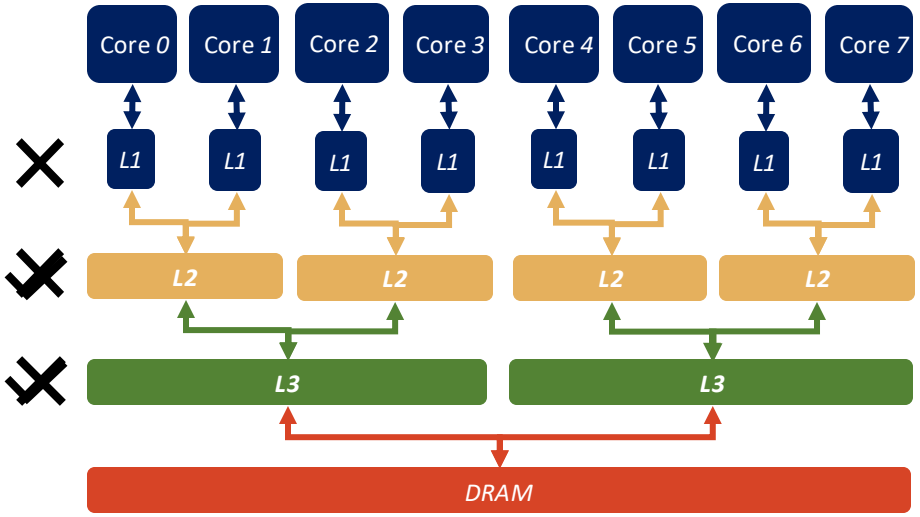
✓ Transparent to legacy software.
✗ Worsens average execution time by 1.5x.
✗ Significant hardware extensions.

Solution IV. Mark all accesses where dirty misses are a possibility as *uncacheable* up to the closest shared cache level of the sharers.

✓ < 50 lines **diff** in the Linux kernel
✓ Supported by ARM ISA.
✓ 1% increase change in average execution time.
✓ Small hardware changes.

PROPOSED SOLUTION

Intuition. With generalized processor.



Example I. Core 0 and Core 1 share data.
Example II. Core 0 and Core 2 share data.
Example III. Core 0 and Core 4 share data.
Example IV. Core 0 and Core 6 share data.

Implementation

RT Application

Identify all shared variables. Mark them with ***cacheability type flag** (`CACHE_FLAG`) in `mmap()`.*

Linux Kernel

Modify `mmap()` to include all ***cacheability type flags.***

ARM ISA

ARM v8-A supports the ***cacheability types.***

Hardware

Replicate **ARM Cortex A53** with hardware changes (TLB, cache controller) required in **gem5 full-system mode.**

**Command:* `buf = mmap(0, size, PROT_READ | PROT_WRITE, MAP_SHARED | CACHE_FLAG, fd, offset);`

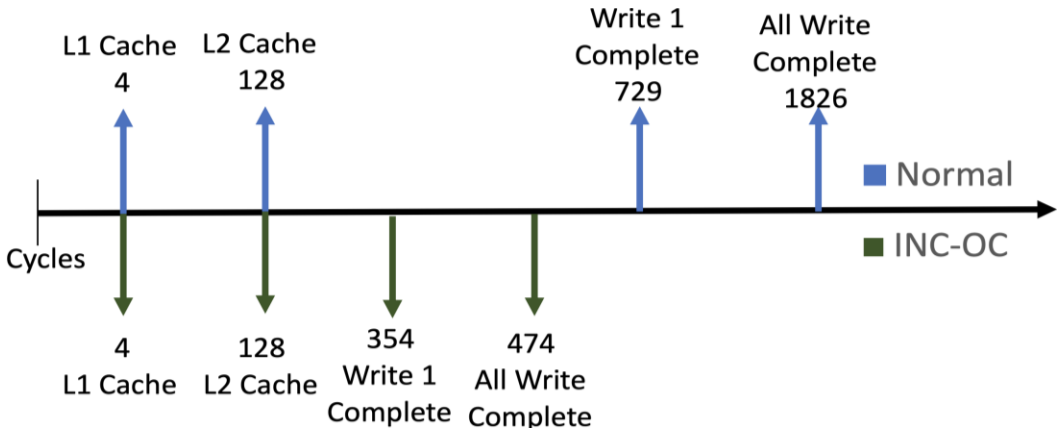
RESULTS: WORST-CASE ANALYSIS USING GEM5

Experiment I. Concurrent write-requests from 4 cores to the same address. **INC-OC** is cached in L2 and not L1. The cache line is in **M** state in one core.

Observations.

I. **Process dirty miss** 52% faster.

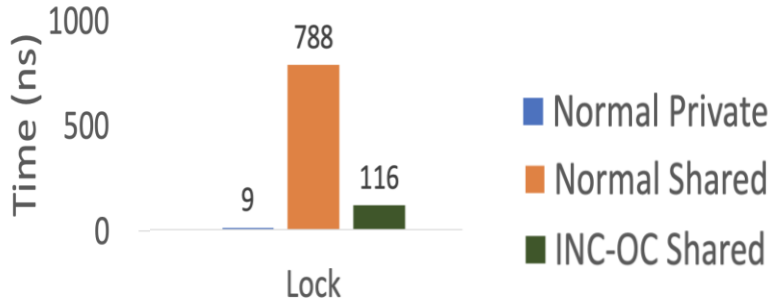
II. **Complete 4 write requests** 74% faster.



Experiment II. Measure the lock acquire-release latency.

Observations.

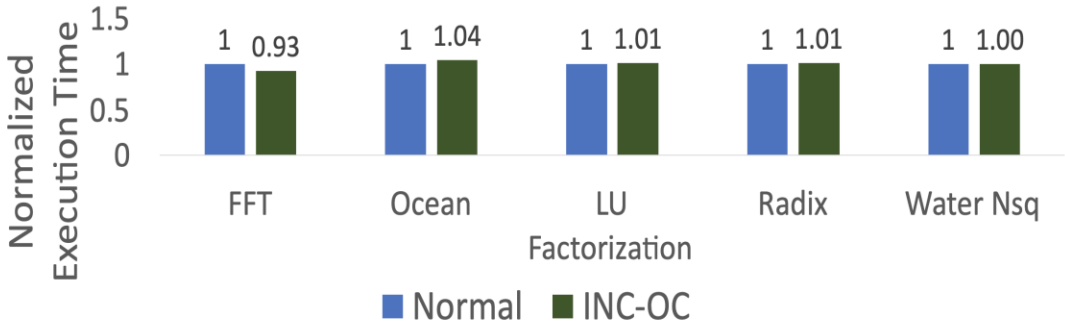
I. **Lock acquire-release** 85% faster.



RESULTS: AVERAGE-CASE ANALYSIS USING GEM5

Experiment.

Analysis of SPLASH2 Benchmark Suite. **INC-OC** is cached in L2 and not L1. All variables incurring possible dirty misses are identified and marked as **INC-OC.**



Observations.

I. Worse the average execution time a by maximum of 1%.
II. This is significantly better than PMSI[1]

[1] Hassan, Mohamed, et al. "Predictable cache coherence for multi-core real-time systems."

CONCLUSION AND FUTURE WORK

Contributions

I. **Identify** major causes of memory-access time variability due to cache coherence.

II. **Introduce mechanism** to improve worst-case memory-request latency by avoiding coherence overheads with selective cache-level bypassing.

Future work

Introduce **compiler support**.

I. **Identify** variables with possibilities of dirty misses through **static analysis**.

II. **Mark** these variables with the necessary cacheability type through static **code-transformation.**

THANK YOU!
Questions?

ILLINOIS

BOSTON UNIVERSITY