Integrated Timing Analysis of Application and Operating Systems Code

Lee Kee CHONG
Clement BALLABRIGA
Sudipta CHATTOPADHYAY
Van-Thuan PHAM
Abhik ROYCHOUDHURY

Example of a real-time system

- Hazardous environment - in need of an autonomous robot

- Uneven, tight terrain - self balancing on two wheels

Example:
A robot used in Fukushima disaster cleanup

Task	Description
balance	Needs to keep upright all the time. *Must consistently run at 50Hz*
navigation	Auto navigation and collision detection. *Must consistently run at 20Hz*
remote	Receives remote commands. *Must finish processing within 100 ms*

Requires precise timing, high reliability

Real-time Operating System (RTOS)

Goal: We want to verify that all real-time constraints are met!

- Worst Case Response Time (WCRT) of an application *running on top of an RTOS*

- Need to consider timing effects of RTOS:
  - System call
  - Interrupt

Application timing analysis

Heptane [ECRTS '01]

SWEET [WCET '05]

Otawa [ERTS '06]

Chronos [SCP '07]

Real-time operating system (RTOS) timing analysis

RTEMS [ECRTS '01]

OSE [RT-TOOLS '02] [ISoLA '04]

µC/OS-II [CSE '09]

seL4 [RTSS '11]

Timing analysis of application + RTOS?

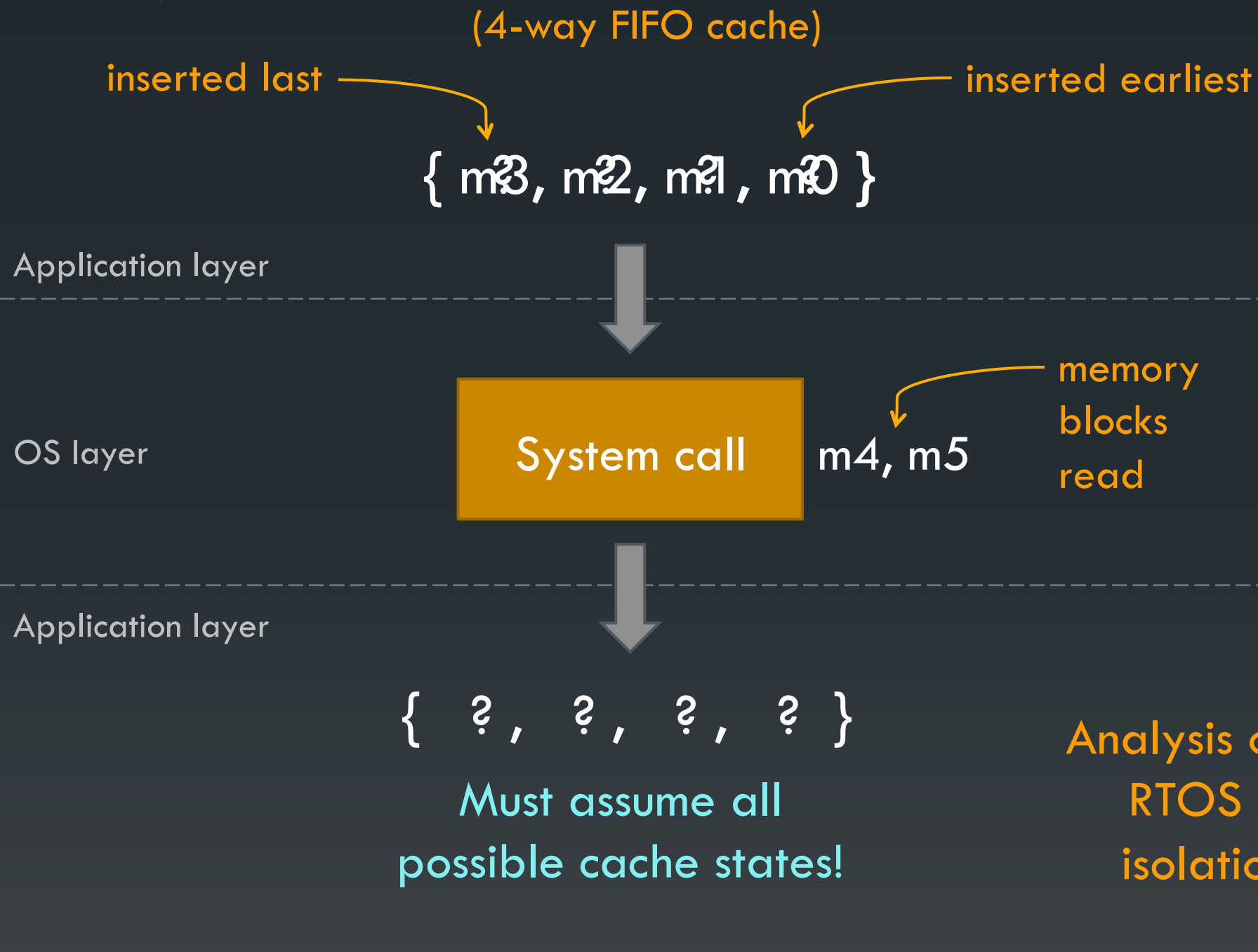
Can we analyze app and RTOS in isolation and combine the result?

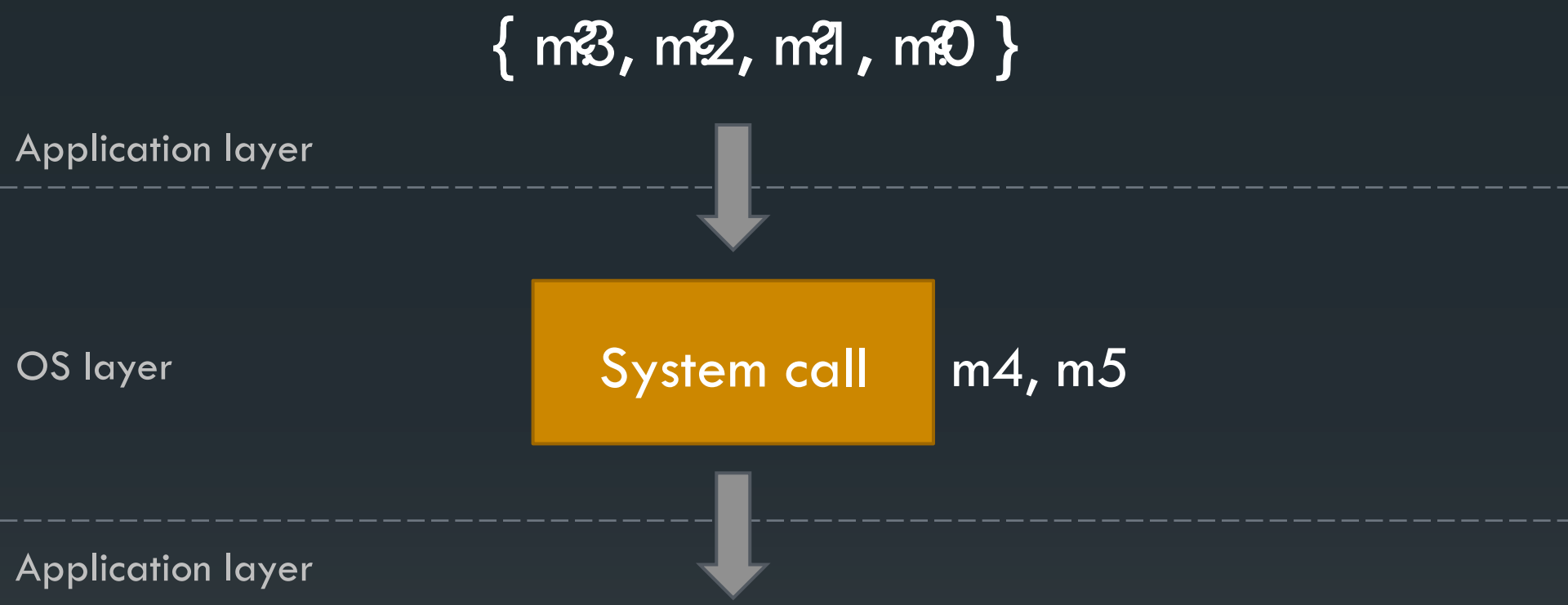
Yes…

Add fixed delay to account for all mode/ context switches

But… gross *overestimation* may occur

Due to loss of application context


(4-way FIFO cache)
inserted last
inserted earliest
{ m3, m2, m1, m0 }
Application layer
OS layer
System call
m4, m5
memory blocks read
Application layer
{ ?, ?, ?, ? }
Must assume all possible cache states!
Analysis of RTOS in isolation


{ m3, m2, m1 , m0 }
Application layer
OS layer
System call
m4, m5
Application layer
{ m5, m4, m3, m2 }
Only one feasible cache state
Takes into account application context

Application timing analysis

More integration

RTOS timing analysis

We propose…

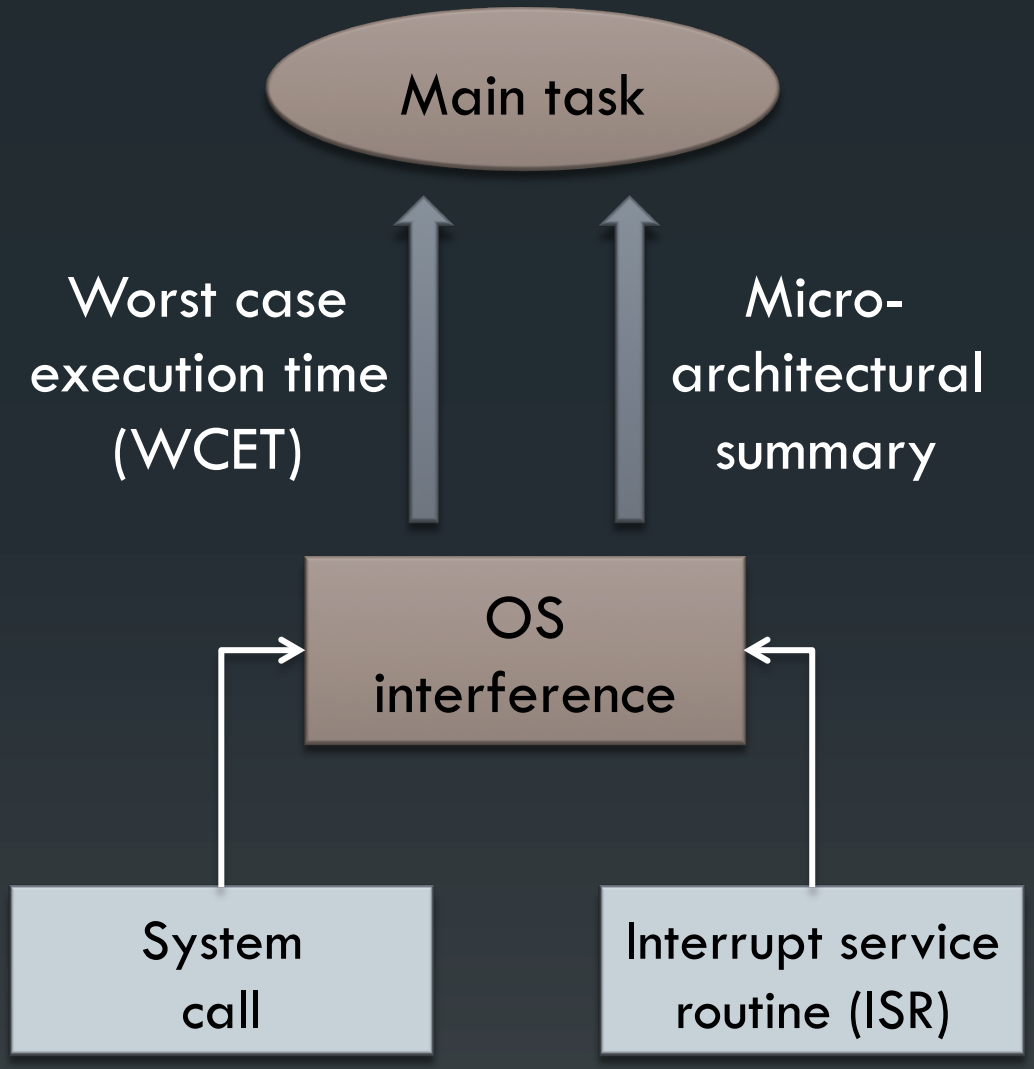
- A timing analysis framework for app + OS
- Consider application context when analyzing OS
- Take into account timing effects of all interferences from OS

Assumptions

- Fixed priority preemptive scheduling
- Each interrupt is serviced by an *interrupt service routine (ISR)*
- A task/ISR can be *periodic* or *sporadic*
- For a sporadic task/ISR, assume release/arrival at its *minimum inter-arrival time*

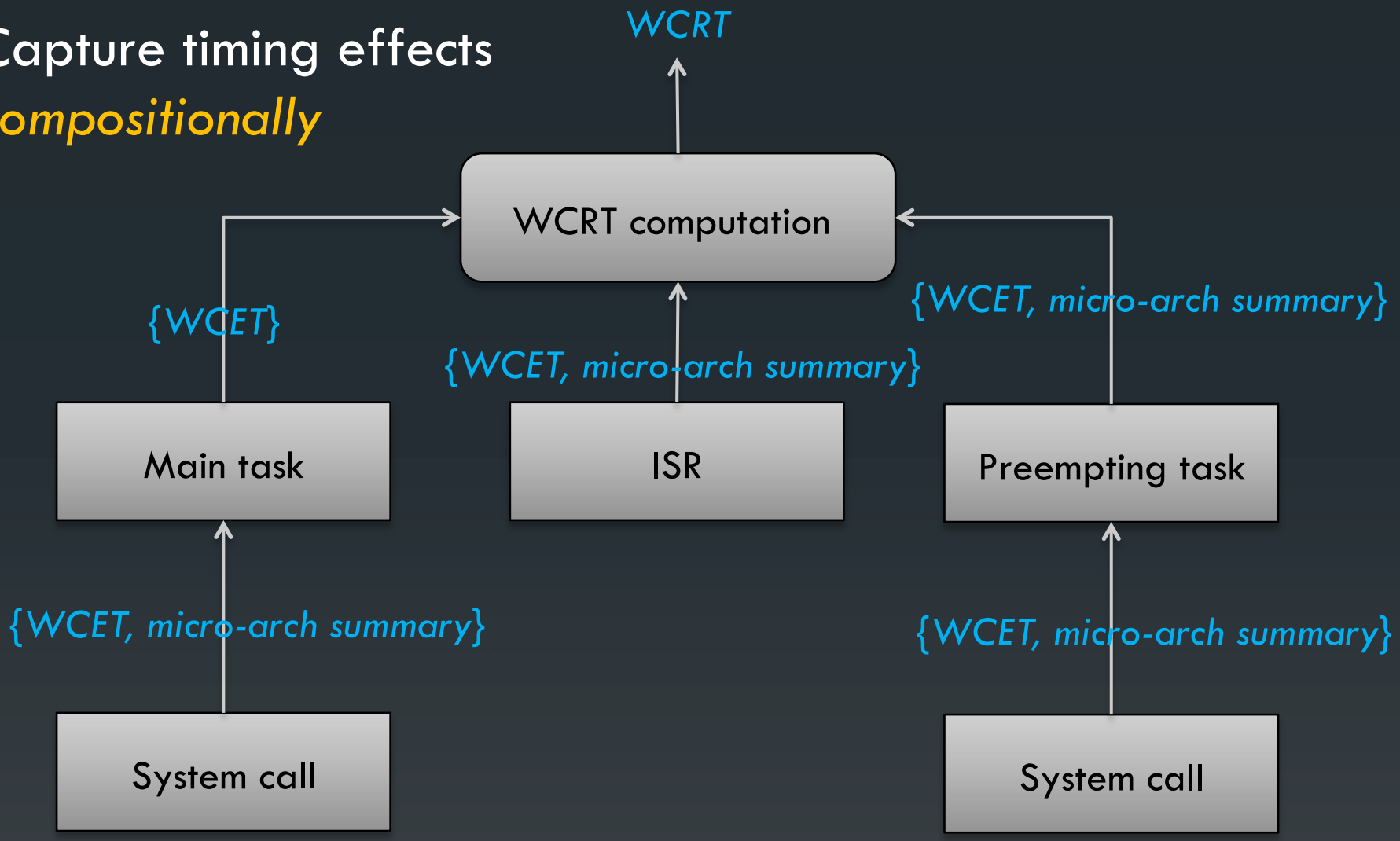
Key idea

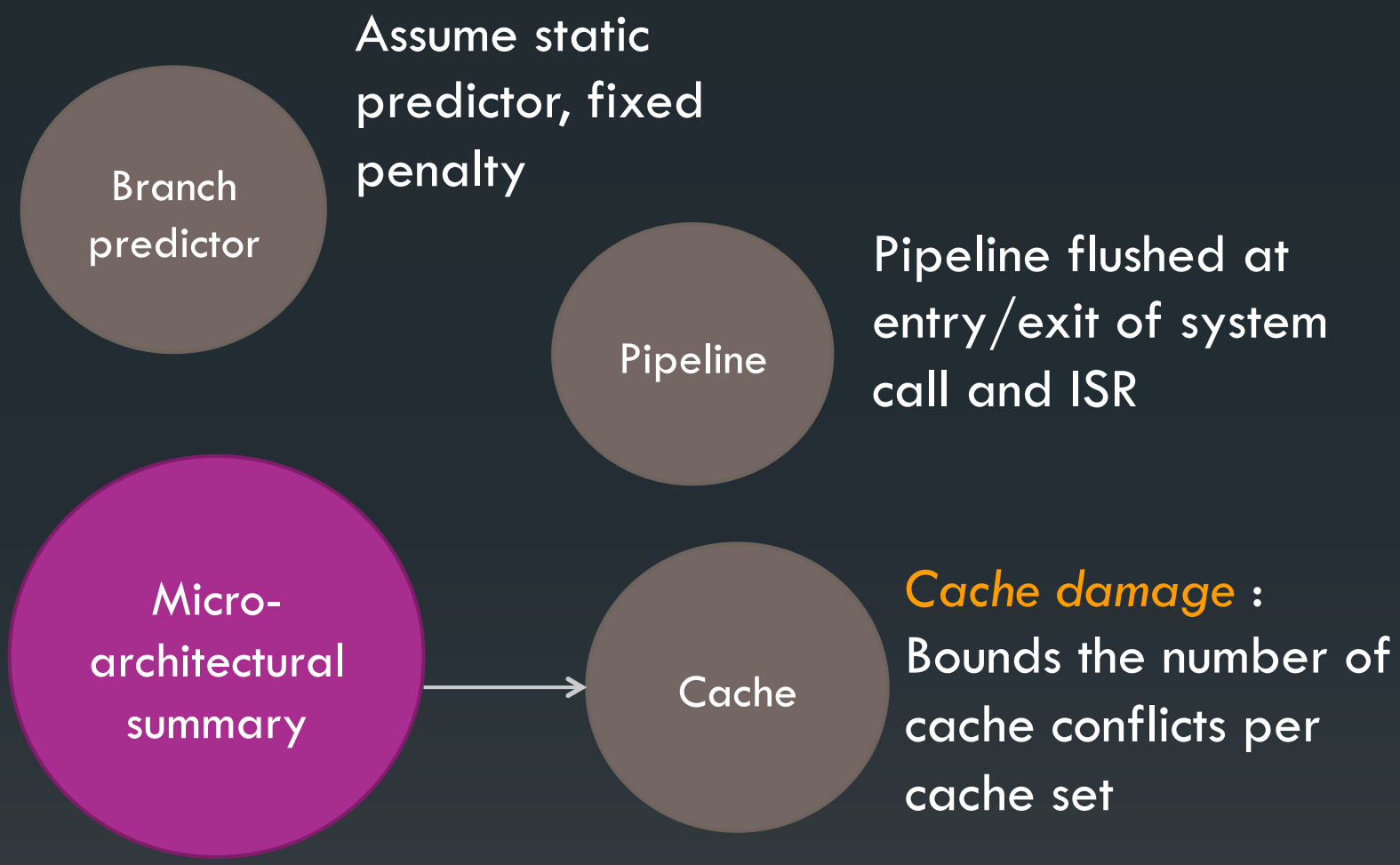
Capture *changes to underlying micro-architectural states* at application layer



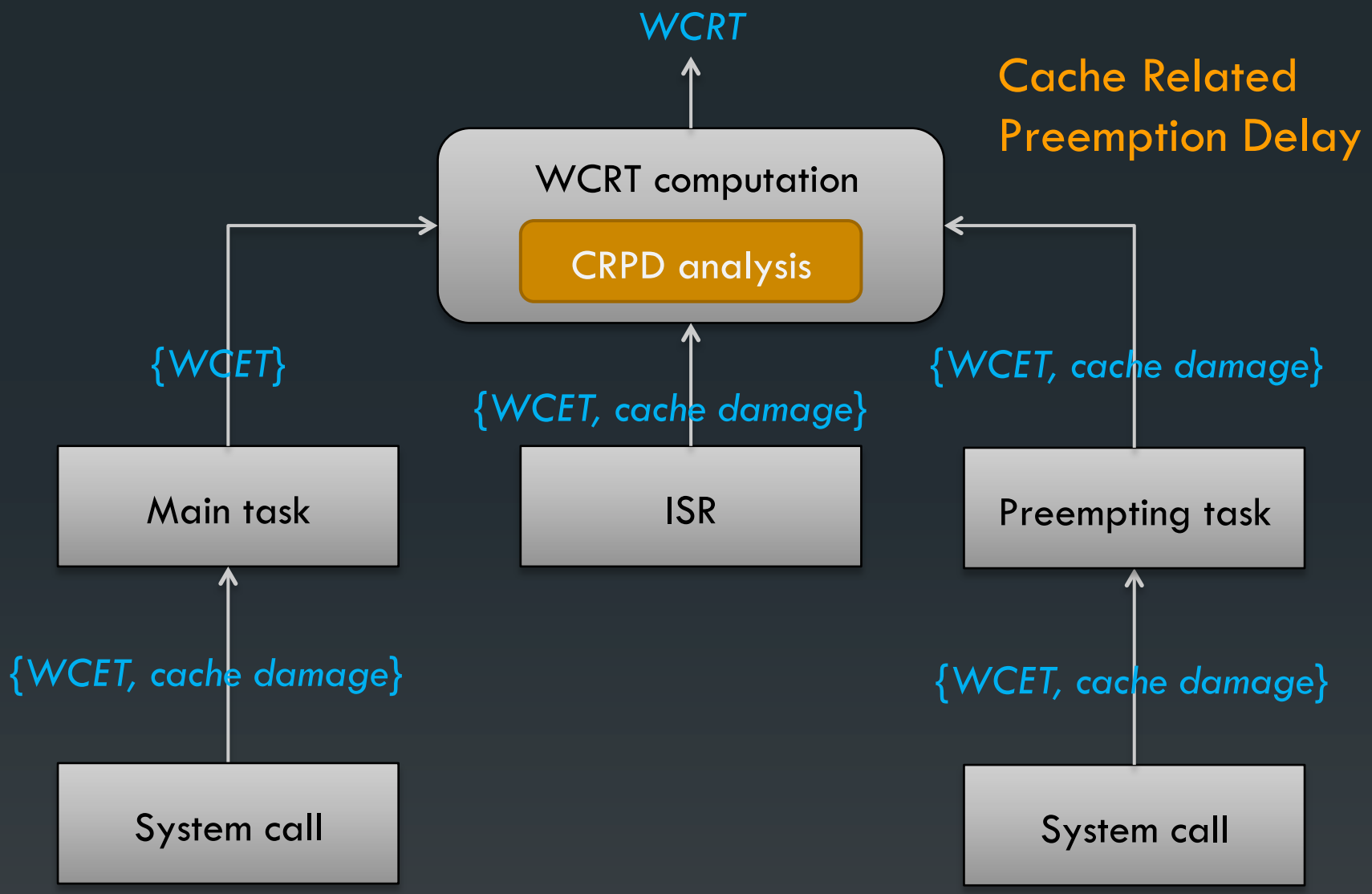
Key idea

Capture timing effects *compositionally*





In this work, we concentrate on the effect of caches.


WCRT
Cache Related Preemption Delay
WCRT computation
CRPD analysis
{WCET}
{WCET, cache damage}
{WCET, cache damage}
Main task
ISR
Preempting task
{WCET, cache damage}
{WCET, cache damage}
System call
System call

Our experiment uses FIFO caches for instruction and data

Problem 1: Existing CRPD analyses are not safe for FIFO caches

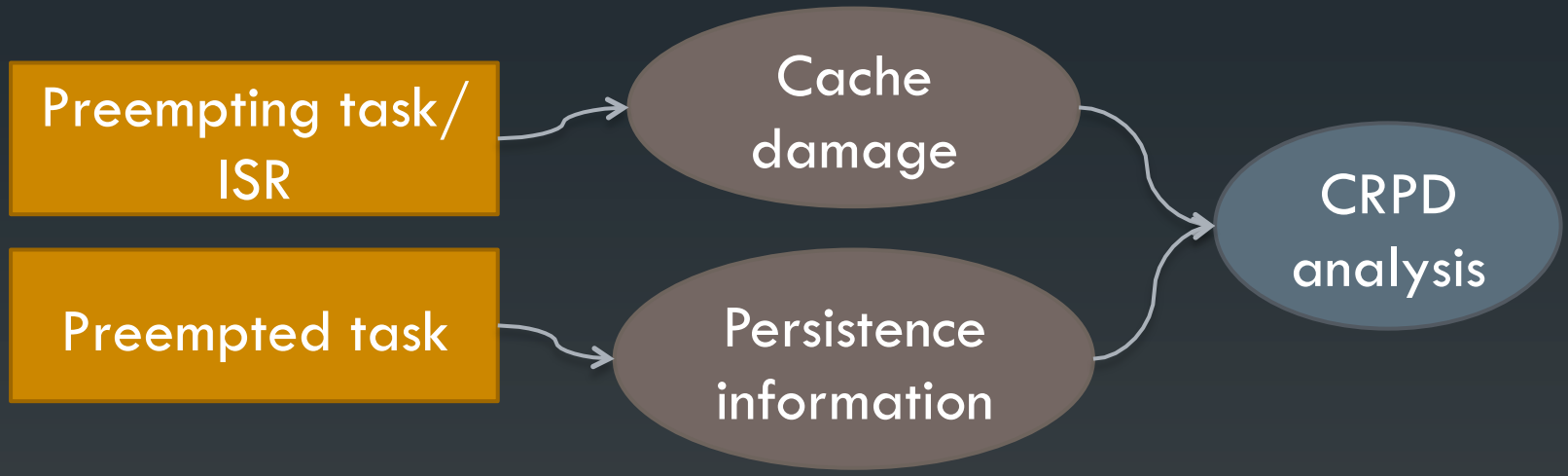
Not bounded by ECB (Evicting cache blocks) and/or UCB (Useful cache blocks)

Problem 2: Also not suitable for data caches

Data reference access pattern

Our approach

Optimizes "allocation" of preemptions to maximize CRPD



A memory block *m* is said to be *persistent* if it can never be evicted from cache

Cache damage due to preemption can *disrupt* persistence of *m*

Needed to bound the damage on preempted task

Persistence information

A slight complication… data cache

```
char x, y, A[32], B[16];
void func() {
  for (int i = 0; i < 8; i++) {
    if (i<4) {
      x = A[i];
    }
    y = B[i];
  }
}
```

A[0..15]	m0
A[16..31]	m1
B[0..15]	m2

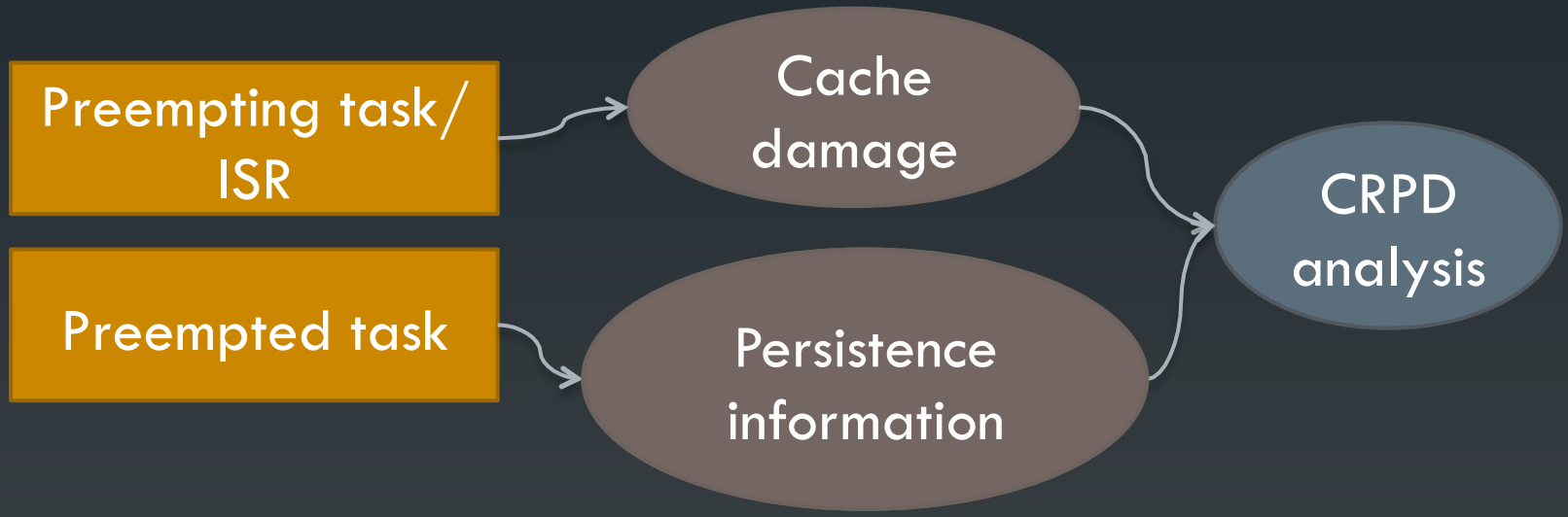
m0, m2 conflicts in a direct-mapped cache

Loop context	Persistent blocks
Iteration 0 .. 3	none
Iteration 4 .. 7	m2

Our approach

Optimizes "allocation" of preemptions *to loop context* that maximizes CRPD

(by maximizing disruption to persistent blocks)



Our experiment uses FIFO caches for instruction and data

Problem 1: Existing CRPD analyses are not safe for FIFO caches

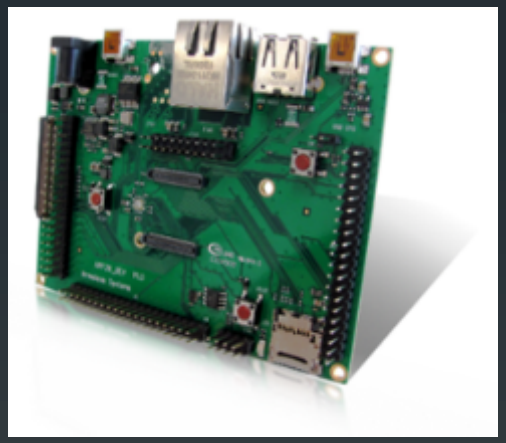
Solution: Bounds disruption to persistent blocks

Problem 2: Not suitable for data caches

Solution: Compute persistent blocks for each loop context

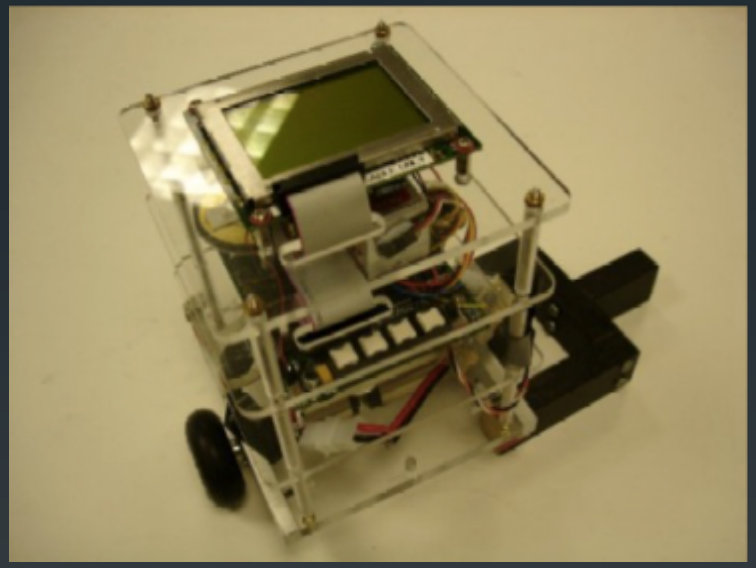
Experimental setup

- Implemented in Chronos [SCP '07]
- µC/OS-II RTOS kernel
- ARM9 processor
  - Single core
  - In-order pipeline
  - Static branch predictor
  - FIFO cache (data and instruction)



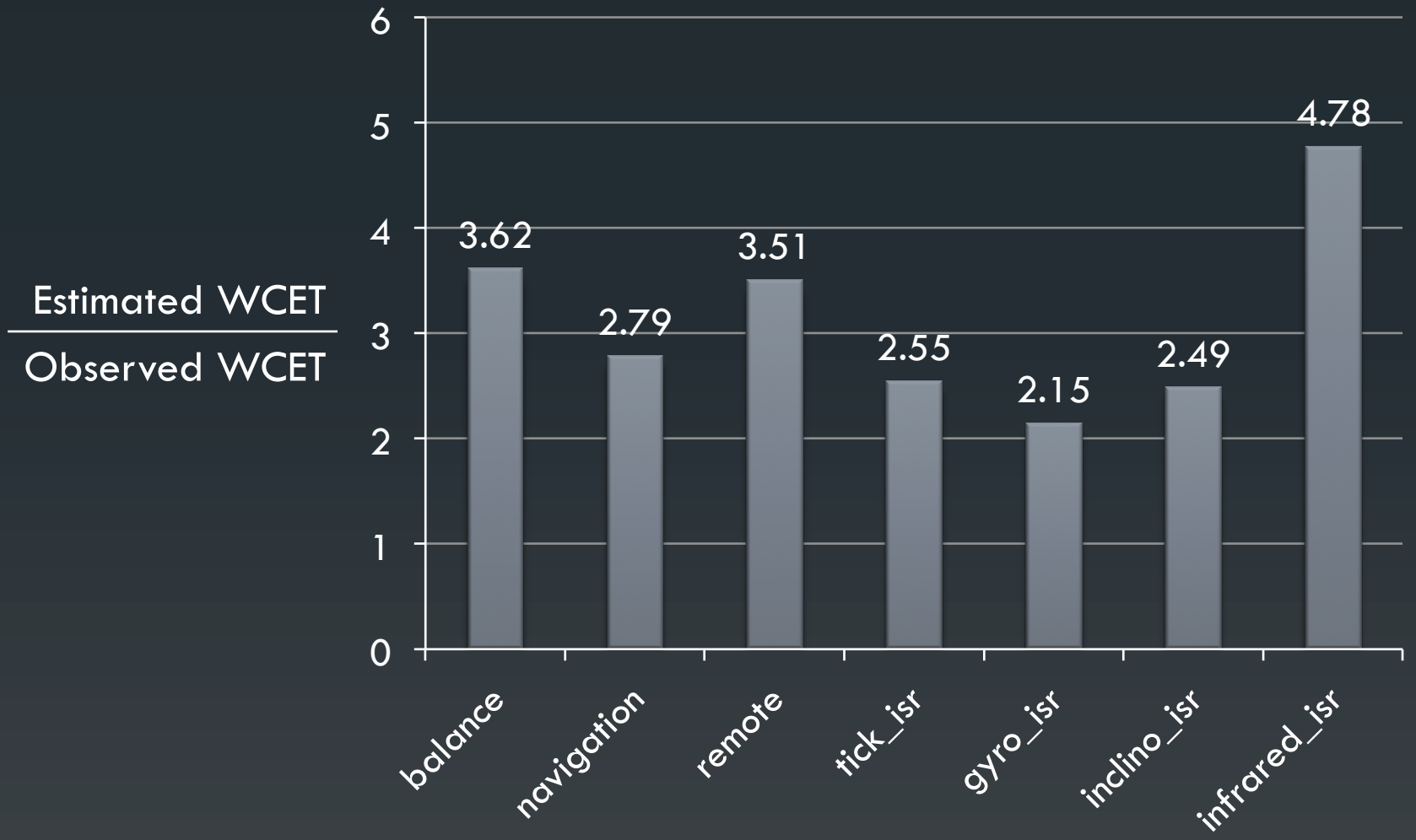
Implementation

- Based on Bally2 robot controller[1]
- Ported to uC/OS-II
- 3 main tasks:
  balance, navigation, remote
- *4 interrupt service routines*

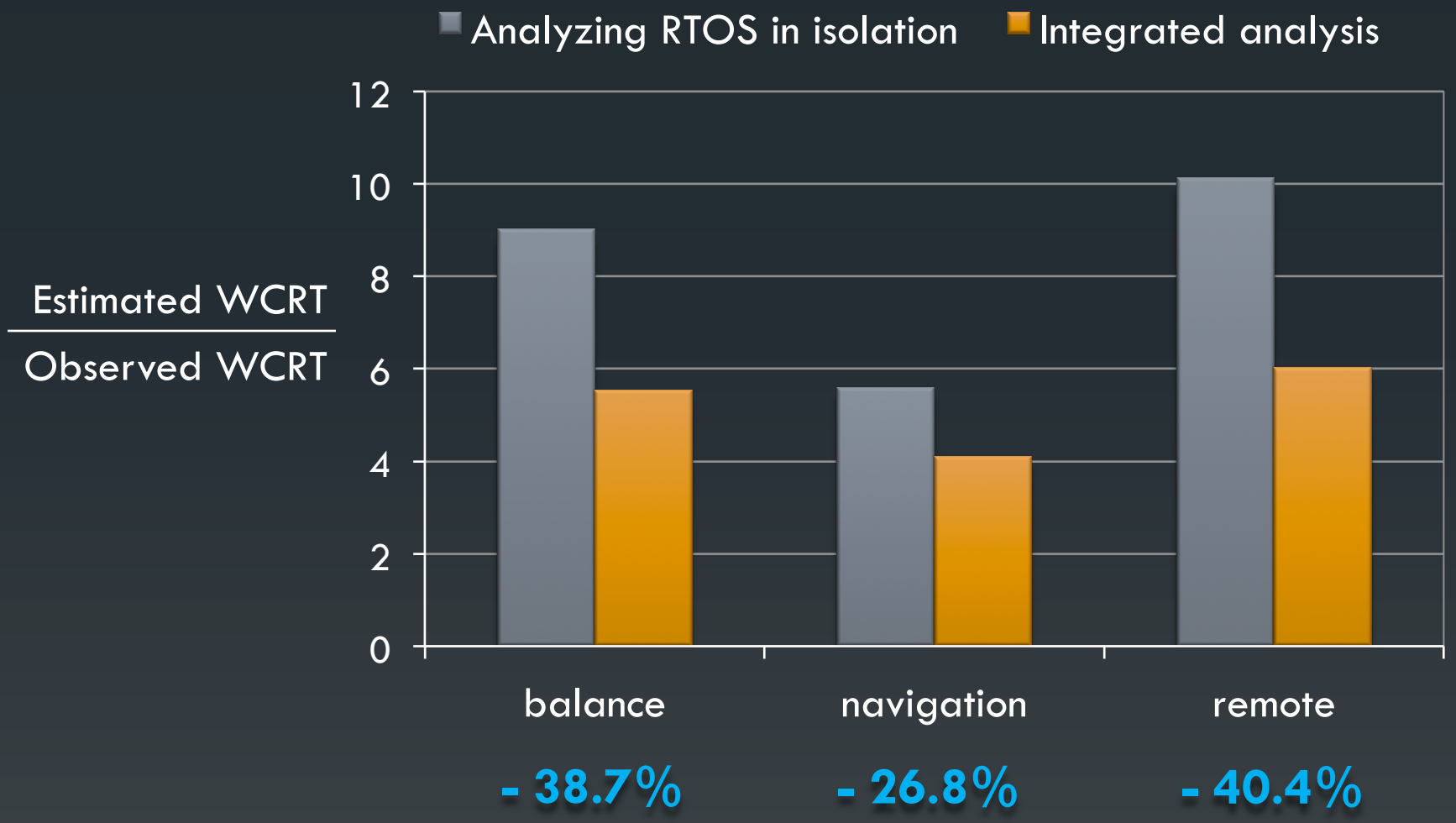


[1] Ballybot balancing robots. http://robotics.ee.uwa.edu.au/eyebot/doc/robots/ballybot.html

Results – WCET overestimation



Results – WCRT overestimation



Verification of real-time constraints

Task	Real-time constraints
balance	Must consistently run at 50Hz
navigation	Must consistently run at 20Hz
remote	Must finish processing within 100ms

Task	Deadline (ms)	Observed WCRT (ms)	Estimated WCRT (ms)
balance	20	0.240	1.331
navigation	50	9.013	36.936
remote	100	0.245	1.478

All tasks meet their deadlines

Summary

- More *integrated* timing analysis of real-time application in the presence of supervisory software (RTOS)
- Capture timing effects of OS through *compositional* analysis
- Evaluated on *real hardware* with realistic robotic application scenario
- *Verification* of real-time constraints for each task in the evaluated robot controller

Our robot control infrastructure will be made available.