EECS 388: Embedded Systems

10. Timing Analysis

Heechul Yun

Agenda

- Execution time analysis
- WCET, BCET, ACET
- Static analysis methods
- Measurement based methods
- Modern computer architecture

Execution Time Analysis

- Will my brake-by-wire system actuate the brakes within one millisecond?
- Will my camera based steer-by-wire system identify a bicycler crossing within 100ms (10Hz)?
- Will my drone be able to finish computing control commands within 10ms (100Hz)?

Execution Time

- **Worst-Case Execution Time (WCET)**
- Best-Case Execution Time (BCET)
- Average-Case Execution Time (ACET)

Execution Time

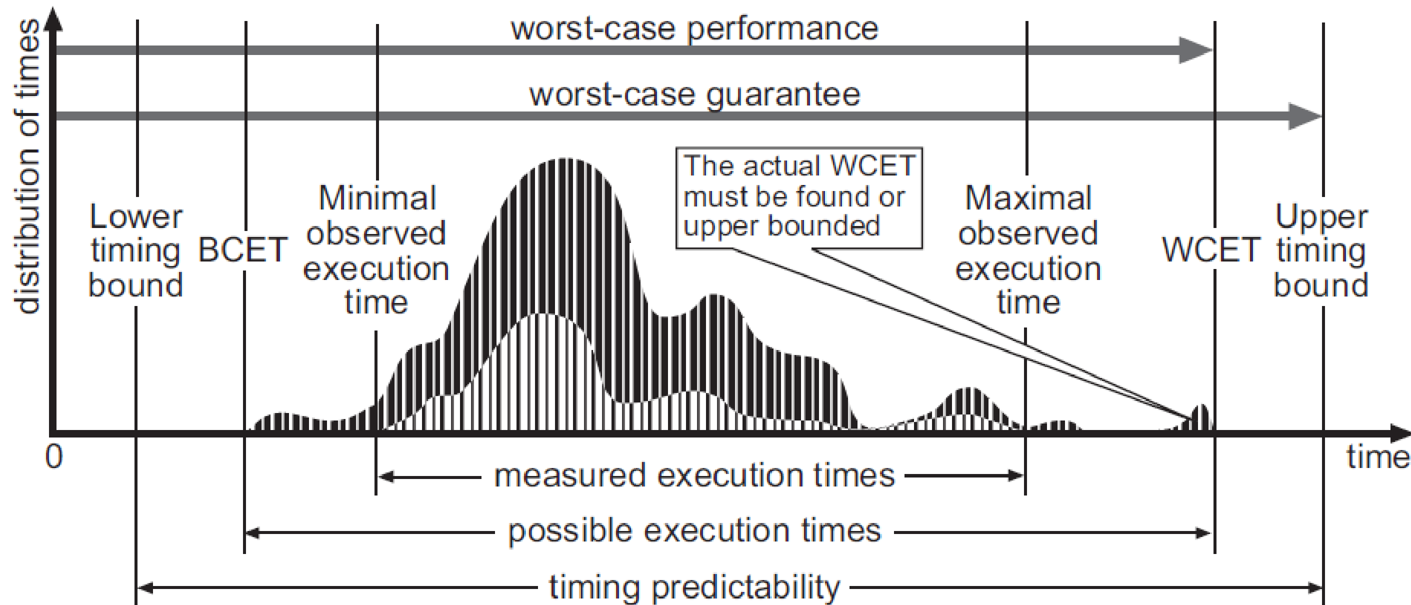


Image source: [Wilhelm et al., 2008]

- Real-time scheduling theory is based on the assumption of known WCETs of real-time tasks

The WCET Problem

- For a given code of a task and the platform (OS & hardware), determine the WCET of the task.

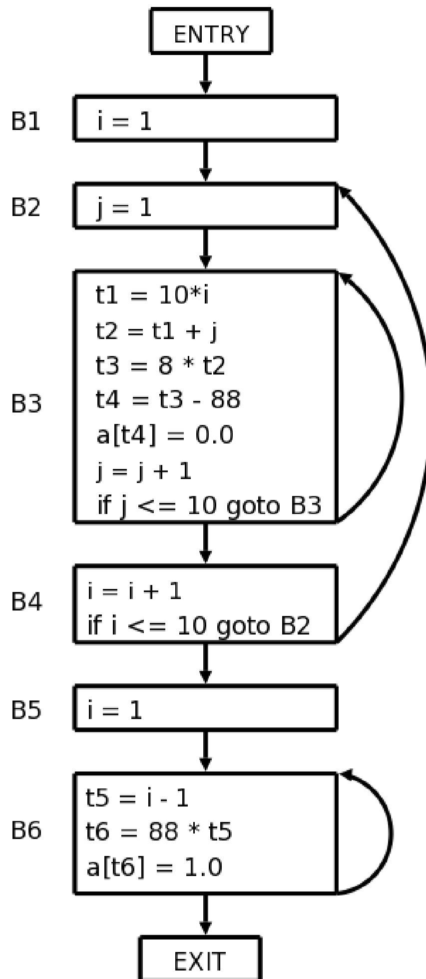
```
while(1) {
    read_from_sensors();
    compute();
    write_to_actuators();
    wait_till_next_period();
}
```

Loops w/ finite bounds
No recursion
Run uninterrupted

Computing WCET

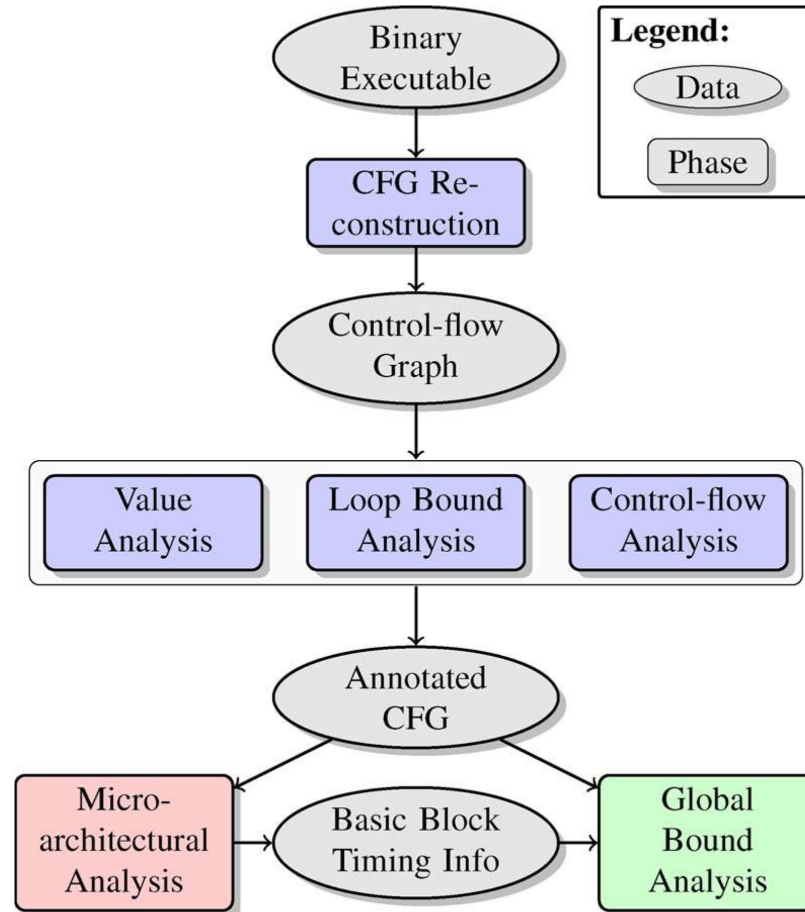
- Static analysis
  - Input: program code, architecture model
  - output: WCET
  - Problem: architecture model is hard and pessimistic
- Measurement
  - No guarantee on true worst-case
  - But, widely used in practice

Static Timing Analysis



- Analyze code
- Split basic blocks
- Find longest path
  - consider loop bounds
- Compute per-block WCET
  - use abstract CPU model
- Compute task WCET
  - by summing up the WCETs of the longest path

Static Timing Analysis



THE UNIVERSITY OF KANSAS

Timing Compositionality

- Consider a task T with two parts A and B composed in sequence: T = A; B
- Is WCET(T) = WCET(A) + WCET(B) ?
- Not always

KU THE UNIVERSITY OF KANSAS

WCET and Caches

- How to determine the WCET of a task?
- The longest execution path of the task?
  - Problem: the *longest path* can take *less time* to finish than shorter paths *if your system has a cache(s)!*
- Example
  - Path1: 1000 instructions, 0 cache misses
  - Path2: 500 instructions, 100 cache misses
  - Cache hit: 1 cycle, Cache miss: 100 cycles
  - Path 2 takes much longer

Processor Behavior Analysis: Cache Effects

```
1  float dot_product(float *x, float *y, int n) {
2    float result = 0.0;
3    int i;
4    for(i=0; i < n; i++) {
5      result += x[i] * y[i];
6    }
7    return result;
8  }
```

Suppose:

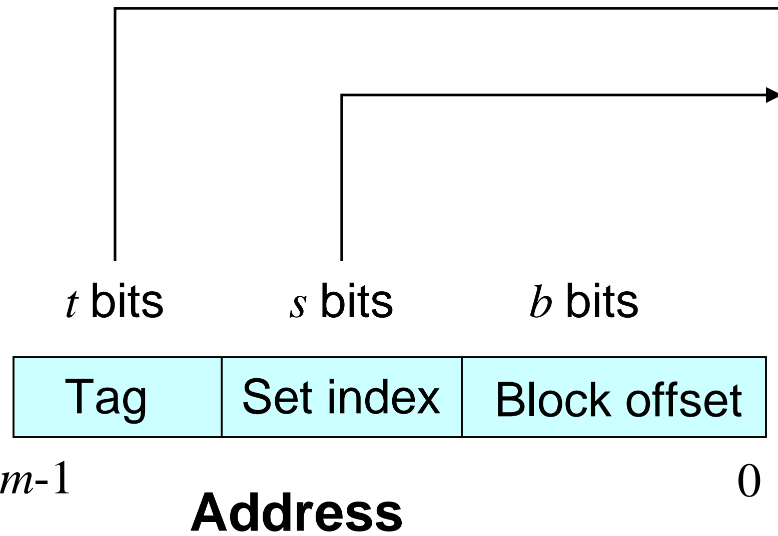
1. 32-bit processor
2. Direct-mapped cache holds two sets
   - 4 floats per set
   - x and y stored contiguously starting at address 0x0

What happens when **n=2**?

Slide source: Edward A. Lee and Prabal Dutta (UCB)

Direct-Mapped Cache

A “set” consists of one “line”



If the tag of the address matches the tag of the line, then we have a “cache hit.” Otherwise, the fetch goes to main memory, updating the line.

1 valid bit | t tag bits | $B = 2^b$ bytes per block

Set 0: Valid | Tag | Block

Set 1: Valid | Tag | Block

⋮

Set S: Valid | Tag | Block

CACHE

Slide source: Edward A. Lee and Prabal Dutta (UCB)

This Particular Direct-Mapped Cache

1 valid bit t tag bits $B = 2^b$ bytes per block

Set 0: Valid | Tag | Block

Set 1: Valid | Tag | Block

Four floats per block, four bytes per float, means 16 bytes, so $b = 4$

$t = 27$ bits $s = 1$ bits $b = 4$ bits

Tag	Set index	Block offset

m-1 … 0

Address = 32 bits

CACHE

KU THE UNIVERSITY OF KANSAS

Slide source: Edward A. Lee and Prabal Dutta (UCB)

Processor Behavior Analysis: Cache Effects

```
1 float dot_product(float *x, float *y, int n) {
2   float result = 0.0;
3   int i;
4   for(i=0; i < n; i++) {
5     result += x[i] * y[i];
6   }
7   return result;
8 }
```

What happens when **n=2**?

Suppose:

1. 32-bit processor
2. Direct-mapped cache holds two sets
   - 4 floats per set
   - x and y stored contiguously starting at address 0x0

x[0] will miss, pulling x[0], x[1], y[0] and y[1] into the set 0. All but one access will be a cache hit.

Slide source: Edward A. Lee and Prabal Dutta (UCB)

Processor Behavior Analysis: Cache Effects

```
1  float dot_product(float *x, float *y, int n) {
2    float result = 0.0;
3    int i;
4    for(i=0; i < n; i++) {
5      result += x[i] * y[i];
6    }
7    return result;
8  }
```

Suppose:

1. 32-bit processor
2. Direct-mapped cache holds two sets
   - 4 floats per set
   - x and y stored contiguously starting at address 0x0

What happens when **n=8**?

x[0] will miss, pulling x[0-3] into the set 0. Then y[0] will miss, pulling y[0-3] *into the same set,* evicting x[0-3]. Every access will be a miss!

Slide source: Edward A. Lee and Prabal Dutta (UCB)

Timing Anomalies

- Locally faster != globally faster

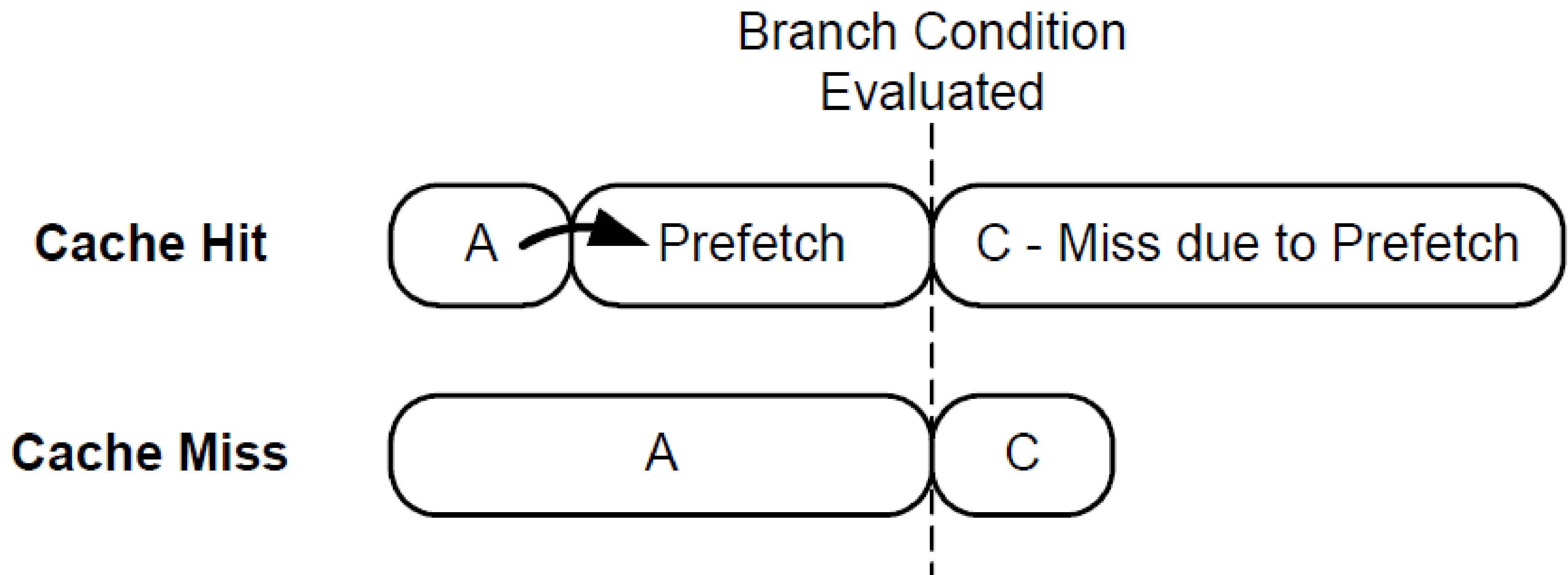


Image source: [Wilhelm et al., 2008]

Timing Anomalies

- Locally faster != globally faster

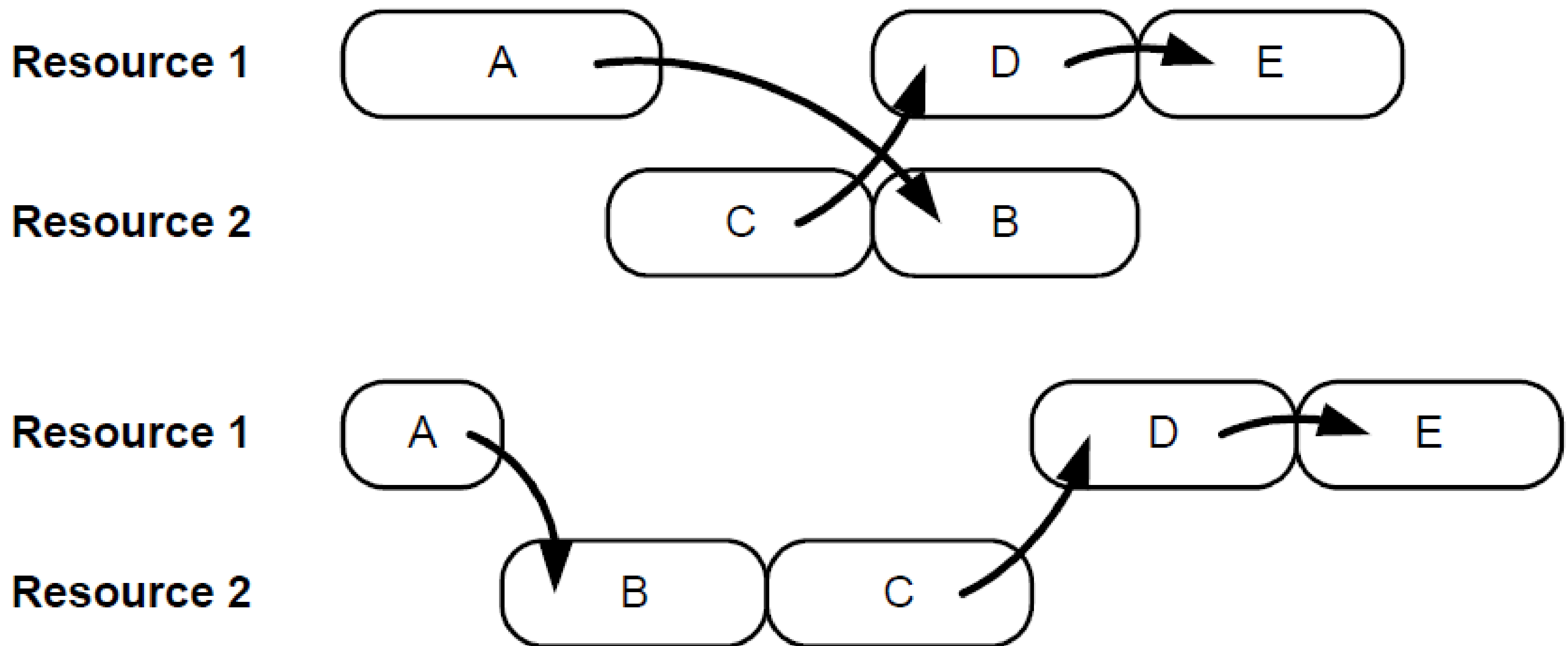


Image source: [Wilhelm et al., 2008]

"Problematic" CPU Features

- Architectures are optimized to reduce **average** performance
- WCET estimation is hard because of
  - Pipelining
  - TLBs/Caches
  - Super-scalar
  - Out-of-order scheduling
  - Branch predictors
  - Hardware prefetchers
  - Basically anything that affect processor state

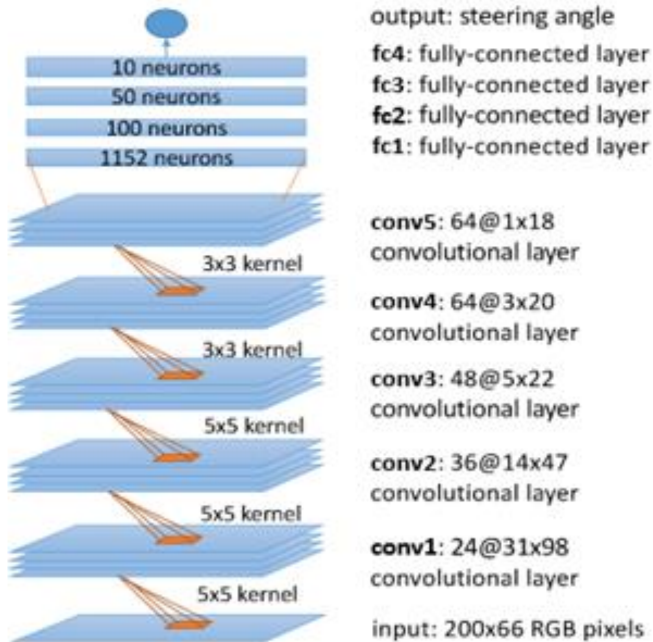
KU THE UNIVERSITY OF KANSAS

Measurement

- Measurement Based Timing Analysis (MBTA)
- Do a lots of measurement under worst-case scenarios (e.g., heavy load)
- Take the maximum + safety margin as WCET
- Commonly practiced in industry

KU THE UNIVERSITY OF KANSAS

Real-Time DNN Control



- ~27M floating point multiplication and additions
  - Per image frame (deadline: 50ms)

M. Bechtel. E. McEllhiney, M Kim, H. Yun. “DeepPicar: A Low-cost Deep Neural Network-based Autonomous Car.” In *RTCSA*, 2018

KU THE UNIVERSITY OF KANSAS

First Attempt

- 1000 samples (minus the first sample. Why?)

	CFS (nice=0)
Mean	23.8
Max	47.9
99pct	47.4
Min	20.7
Median	20.9
Stdev.	7.7

Why?

KU THE UNIVERSITY OF KANSAS

DVFS

- Dynamic voltage and frequency scaling (DVFS)
- Lower frequency/voltage saves power
- Vary clock speed depending on the load
- Cause timing variations
- Disabling DVFS

```
# echo performance > /sys/devices/system/cpu/cpu0/cpufreq/scaling_governor
# echo performance > /sys/devices/system/cpu/cpu1/cpufreq/scaling_governor
# echo performance > /sys/devices/system/cpu/cpu2/cpufreq/scaling_governor
# echo performance > /sys/devices/system/cpu/cpu3/cpufreq/scaling_governor
```

Second Attempt (No DVFS)

	CFS (nice=0)
Mean	21.0
Max	22.4
99pct	21.8
Min	20.7
Median	20.9
Stdev.	0.3

- What if there are other tasks in the system?

KU THE UNIVERSITY OF KANSAS

Third Attempt (Under Load)

	CFS (nice=0)
Mean	31.1
Max	47.7
99pct	41.6
Min	21.6
Median	31.7
Stdev.	3.1

- 4x cpuhog compete the cpu time with the DNN

KU THE UNIVERSITY OF KANSAS

Recall: kernel/sched/fair.c (CFS)

- Priority to CFS weight conversion table
  - Priority (Nice value): -20 (highest) ~ +19 (lowest)
  - kernel/sched/core.c

```
const int sched_prio_to_weight[40] = {
 /* -20 */     88761,     71755,     56483,     46273,     36291,
 /* -15 */     29154,     23254,     18705,     14949,     11916,
 /* -10 */      9548,      7620,      6100,      4904,      3906,
 /*  -5 */      3121,      2501,      1991,      1586,      1277,
 /*   0 */      1024,       820,       655,       526,       423,
 /*   5 */       335,       272,       215,       172,       137,
 /*  10 */       110,        87,        70,        56,        45,
 /*  15 */        36,        29,        23,        18,        15,
};
```

KU THE UNIVERSITY OF KANSAS

Fourth Attempt (Use Priority)

	CFS (nice=0)	CFS (nice=-2)	CFS (nice=-5)
Mean	31.1	27.2	21.4
Max	47.7	44.9	31.3
99pct	41.6	40.8	22.4
Min	21.6	21.6	21.1
Median	31.7	22.1	21.3
Stdev.	3.1	5.8	0.4

- Effect may vary depending on the workloads

KU THE UNIVERSITY OF KANSAS

Fifth Attempt (Use RT Scheduler)

	CFS (nice=0)	CFS (nice=-2)	CFS (nice=-5)	FIFO
Mean	31.1	27.2	21.4	21.4
Max	47.7	44.9	31.3	22.0
99pct	41.6	40.8	22.4	21.8
Min	21.6	21.6	21.1	21.1
Median	31.7	22.1	21.3	21.4
Stdev.	3.1	5.8	0.4	0.1

- Are we done?

KU THE UNIVERSITY OF KANSAS

BwRead

```
#define MEM_SIZE (4*1024*1024)
char ptr[MEM_SIZE];
while(1)
{
    for(int i = 0; i < MEM_SIZE; i += 64) {
        sum += ptr[i];
    }
}
```

- Use this instead of the ‘cpuhog’ as background tasks
- Everything else is the same.
- Will there be any differences? If so, why?

KU THE UNIVERSITY OF KANSAS

Sixth Attempt (Use BwRead)

	Solo	w/ BwRead		
	CFS (nice=0)	CFS (nice=0)	CFS (nice=-5)	FIFO
Mean	21.0	75.8	52.3	50.2
Max	22.4	123.0	80.1	51.7
99pct	21.8	107.8	72.4	51.3
Min	20.7	40.6	40.9	38.3
Median	20.9	81.0	50.1	50.6
Stdev.	0.3	17.7	6.1	1.9

- ~2.5X (fifo) WCET increase! Why?

KU THE UNIVERSITY OF KANSAS

BwWrite

```
#define MEM_SIZE (4*1024*1024)
char ptr[MEM_SIZE];
while(1)
{
    for(int i = 0; i < MEM_SIZE; i += 64) {
        ptr[i] = 0xff;
    }
}
```

- Use this background tasks instead
- Everything else is the same.
- Will there be any differences? If so, why?

KU THE UNIVERSITY OF KANSAS

Seventh Attempt (Use BwWrite)

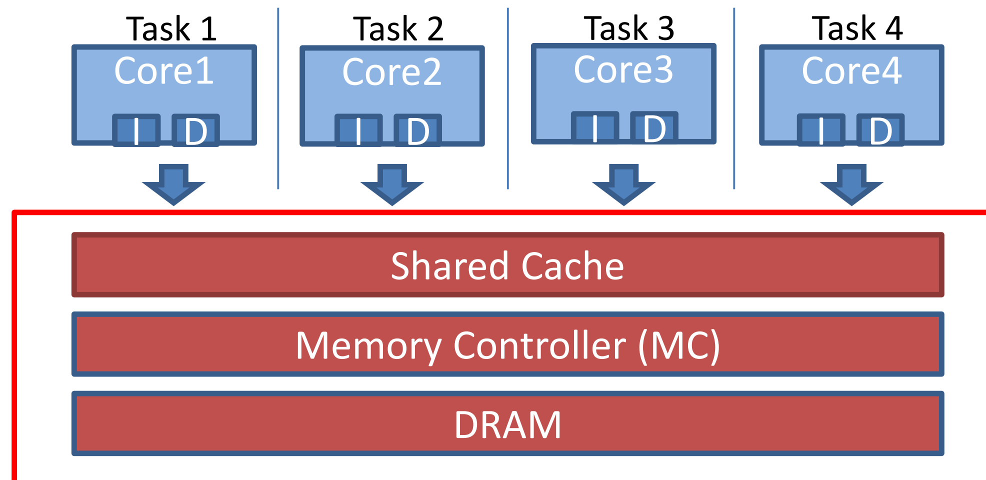
	Solo	w/ BwWrite		
	CFS (nice=0)	CFS (nice=0)	CFS (nice=-5)	FIFO
Mean	21.0	101.2	89.7	92.6
Max	22.4	194.0	137.2	99.7
99pct	21.8	172.4	119.8	97.1
Min	20.7	89.0	71.8	78.7
Median	20.9	93.0	87.5	92.5
Stdev.	0.3	22.8	7.7	1.0

- ~4.7X (fifo) WCET increase! Why?

KU THE UNIVERSITY OF KANSAS

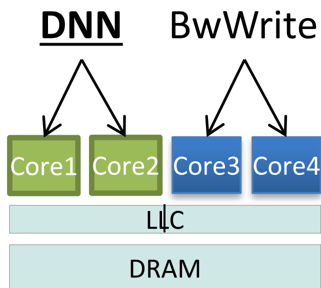
Shared Memory Hierarchy

- **Memory performance** varies widely due to **interference**
- Task WCET can be **extremely pessimistic**



KU THE UNIVERSITY OF KANSAS

Effect of Memory Interference



12
10
8
6
4
2
0

Normalized Exeution Time

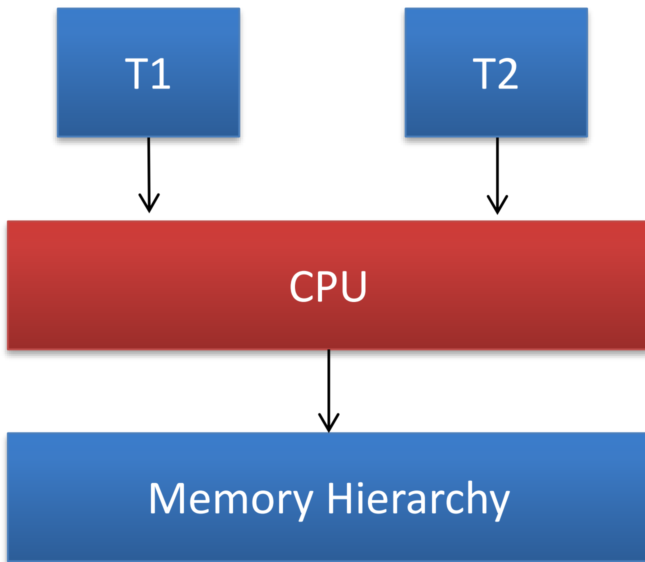
Solo
Corun

DNN (Core 0,1)
BwWrite (Core 2,3)

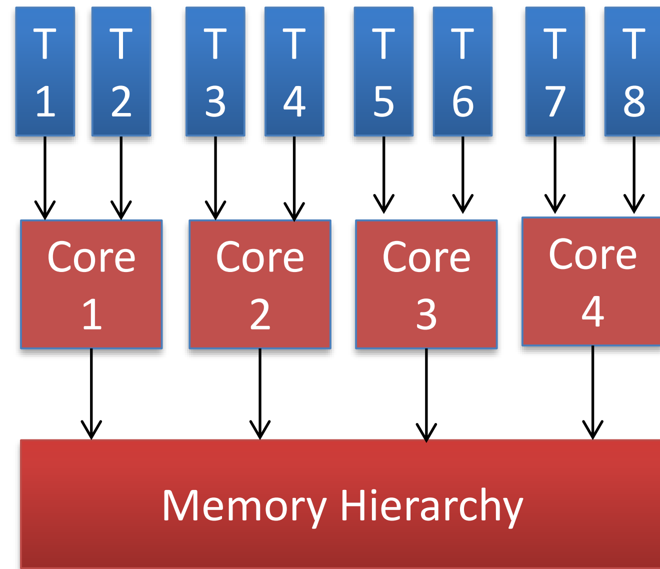
- DNN control task suffers **>10X slowdown**
  - When co-scheduling different tasks on on idle cores.

Waqar Ali and Heechul Yun. “RT-Gang: Real-Time Gang Scheduling Framework for Safety-Critical Systems.” *RTAS*, 2019

Challenges: **Shared Memory Hierarchy**



Unicore

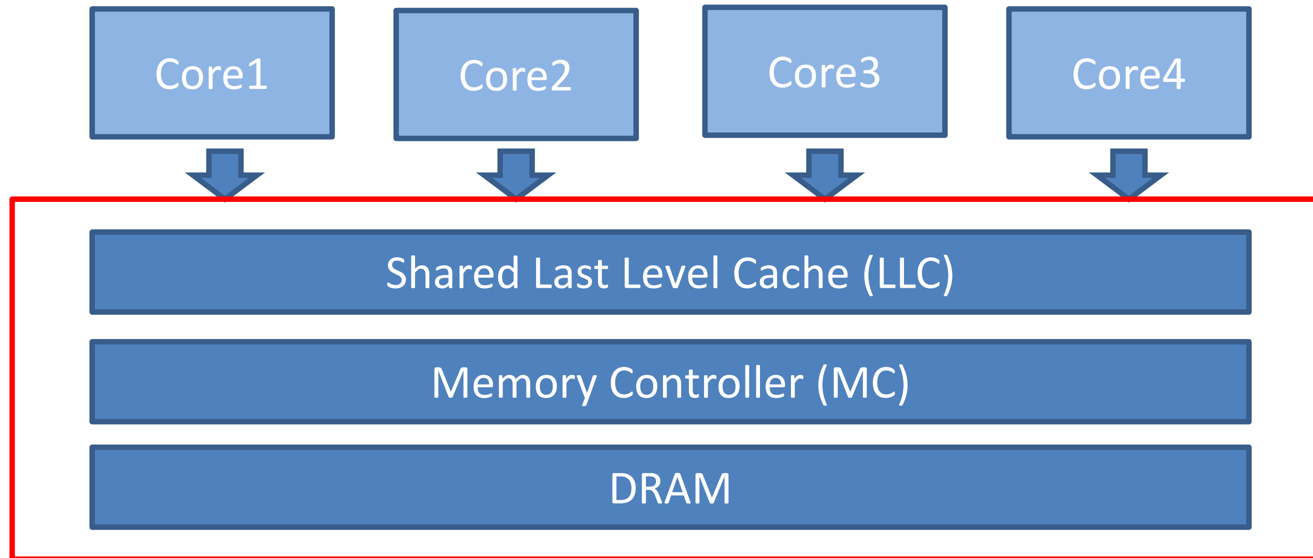


Multicore

Performance Impact

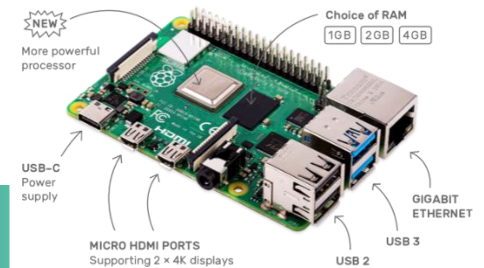
KU THE UNIVERSITY OF KANSAS

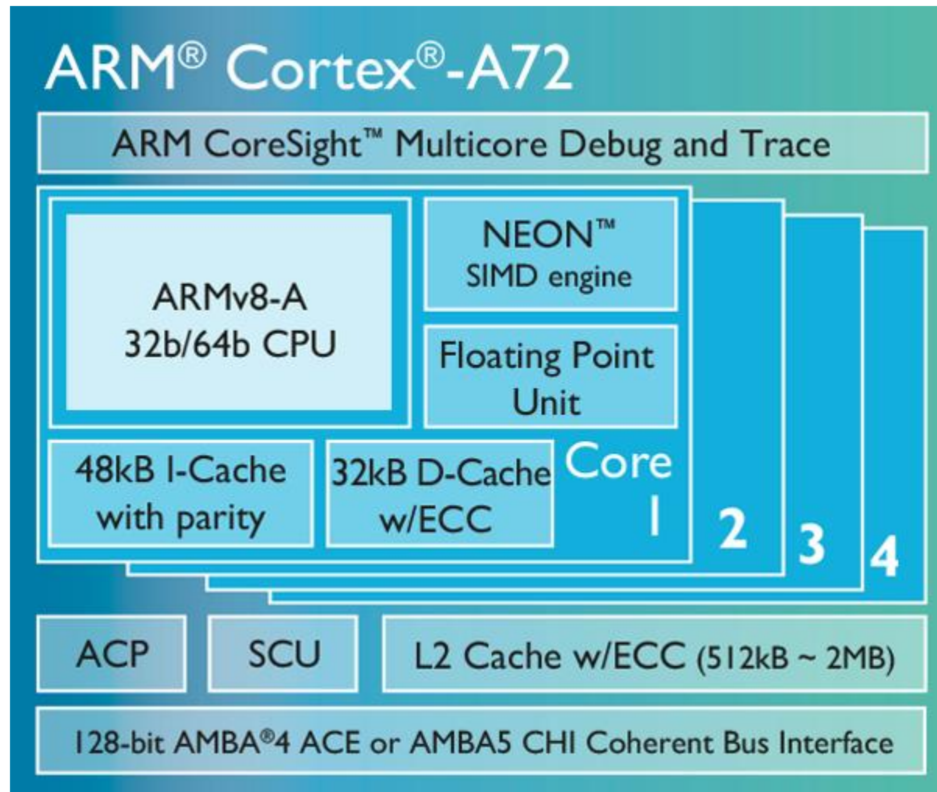
Shared Memory Hierarchy



- Cache space
- Memory bus bandwidth
- Memory controller queues
- …

ARM Cotex-A72





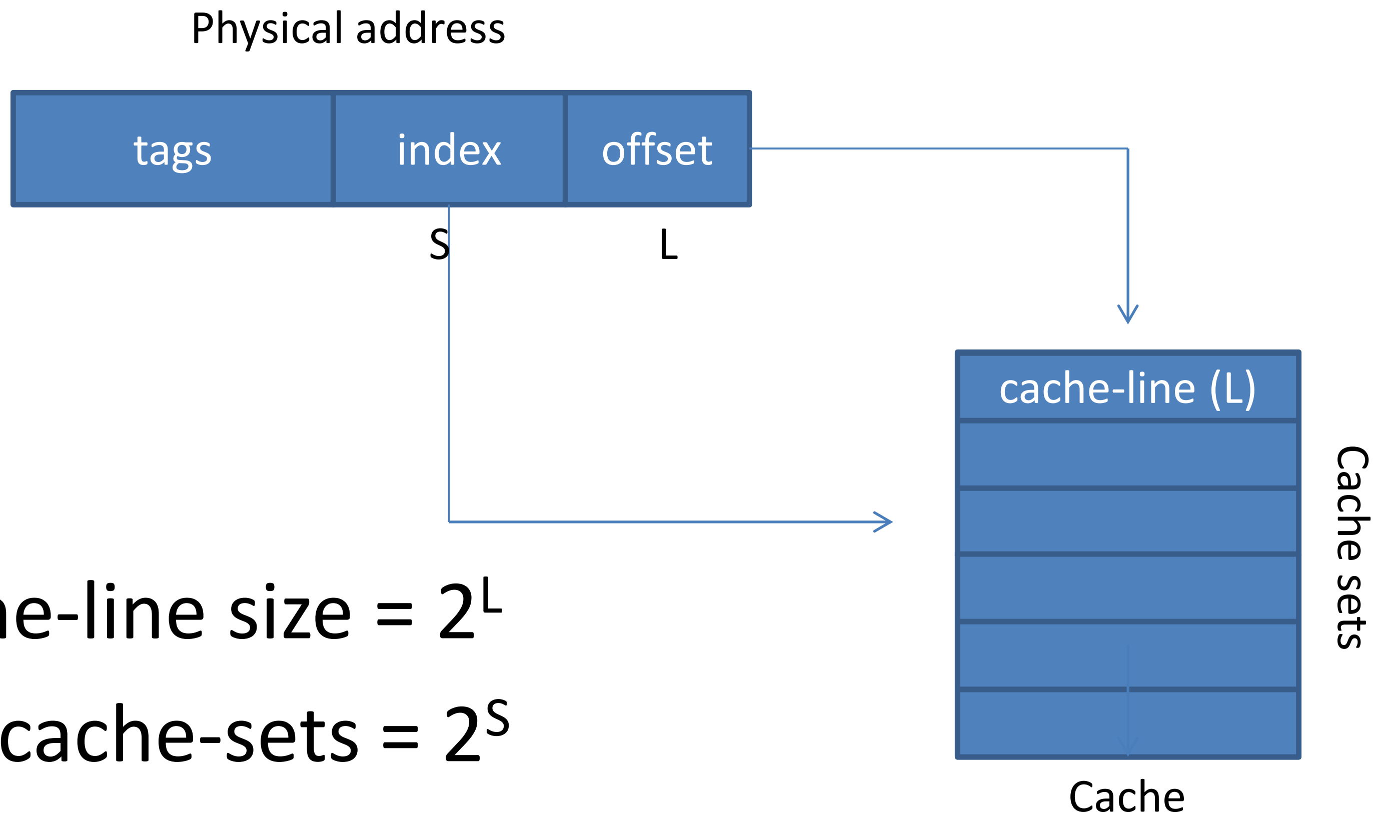
- Your Pi 4: 1 MB shared L2 cache, 2GB DRAM

Cache Architecture

- Some terminologies
  - Cache-line
  - Cache tag, index, and offset
  - Direct map cache
  - Set-associative cache, cache ways

KU THE UNIVERSITY OF KANSAS

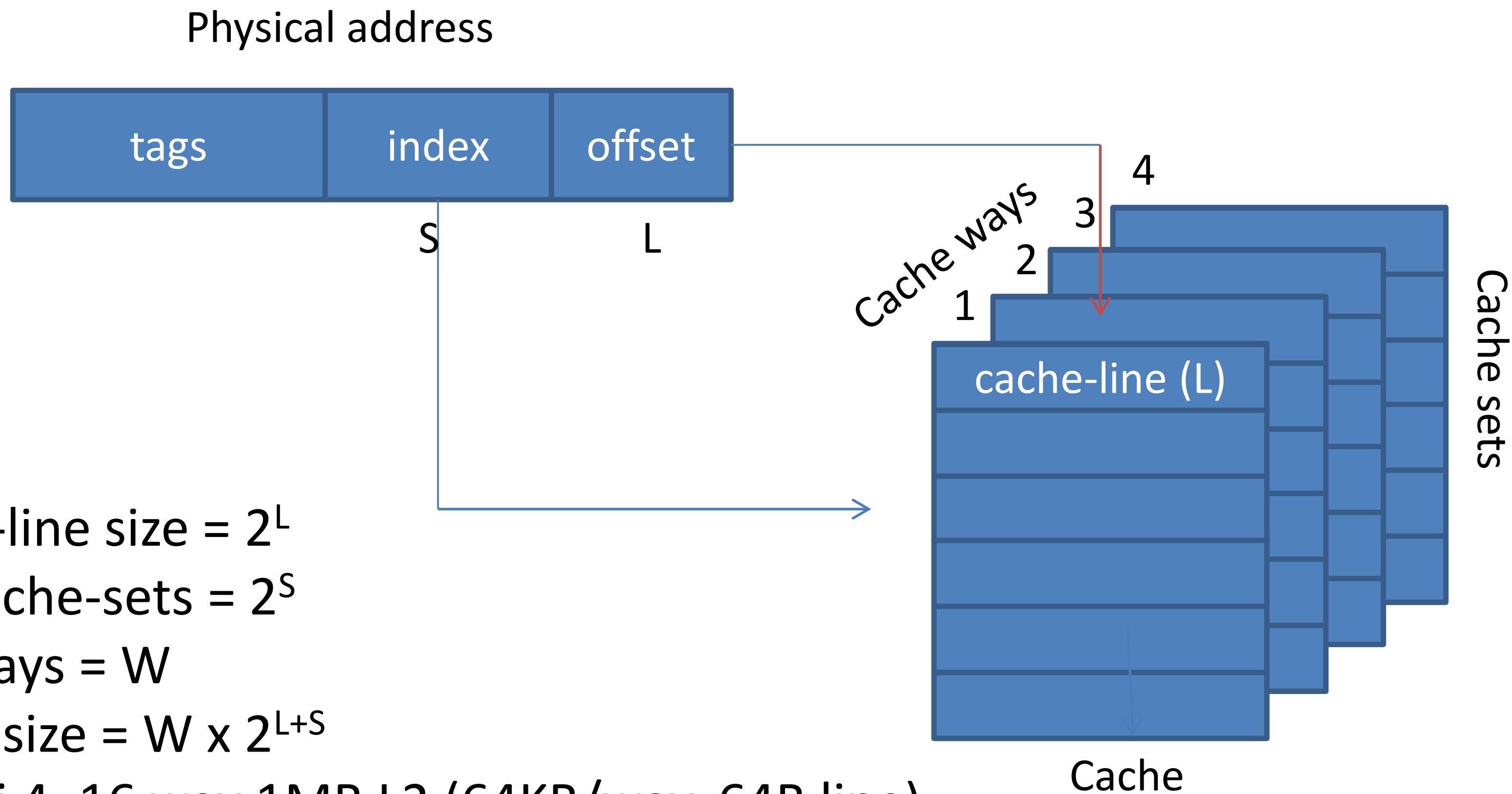
Direct Map Cache



- Cache-line size = 2^L
- # of cache-sets = 2^S
- Cache size = 2^{L+S}

KU THE UNIVERSITY OF KANSAS

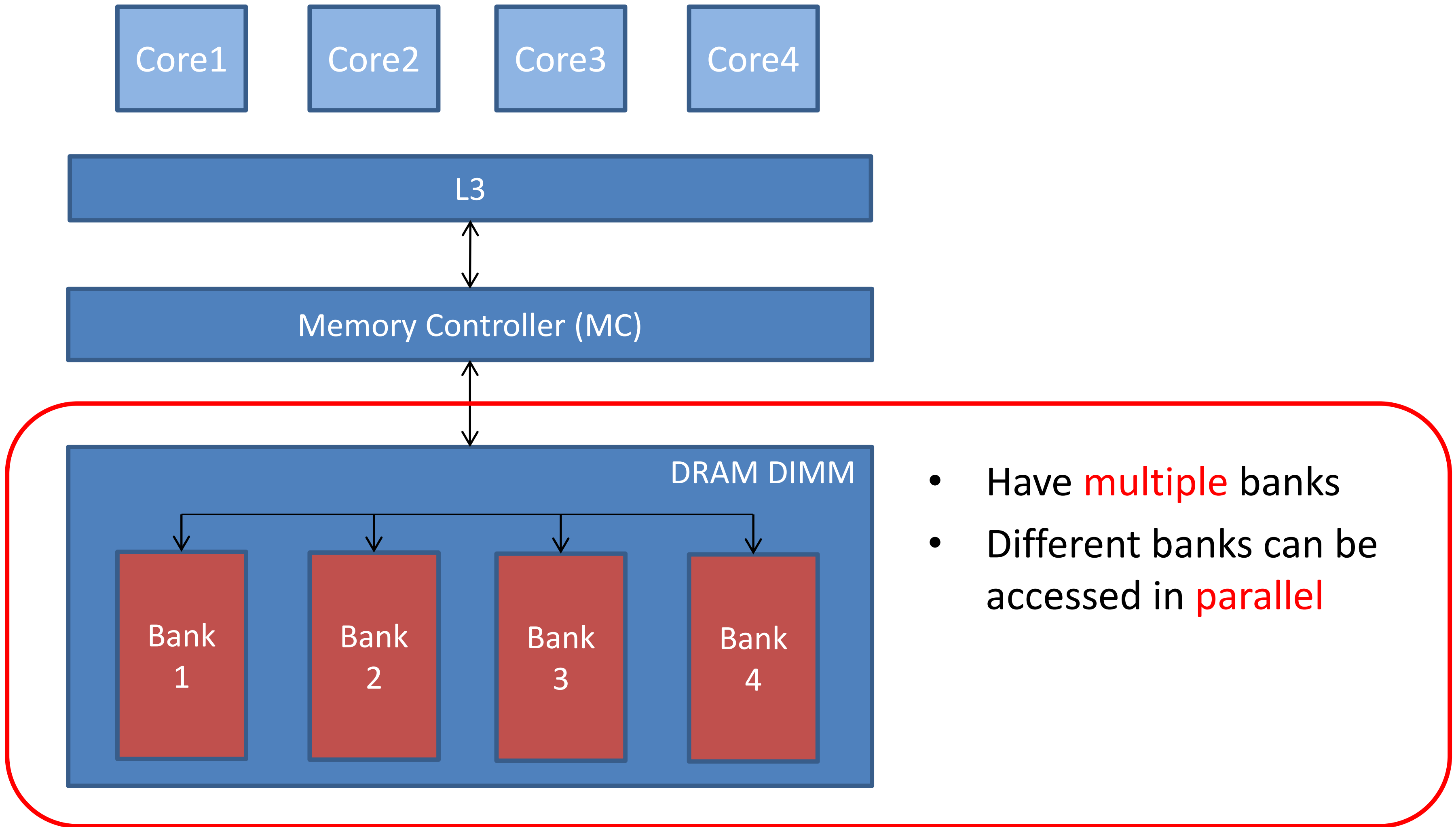
Set-associative Cache



- Cache-line size = 2^L
- # of cache-sets = 2^S
- # of ways = W
- Cache size = W x 2^{L+S}
- Your Pi 4: 16 way 1MB L2 (64KB/way, 64B line)
  - W = 16, L = 6, S = 10

KU THE UNIVERSITY OF KANSAS

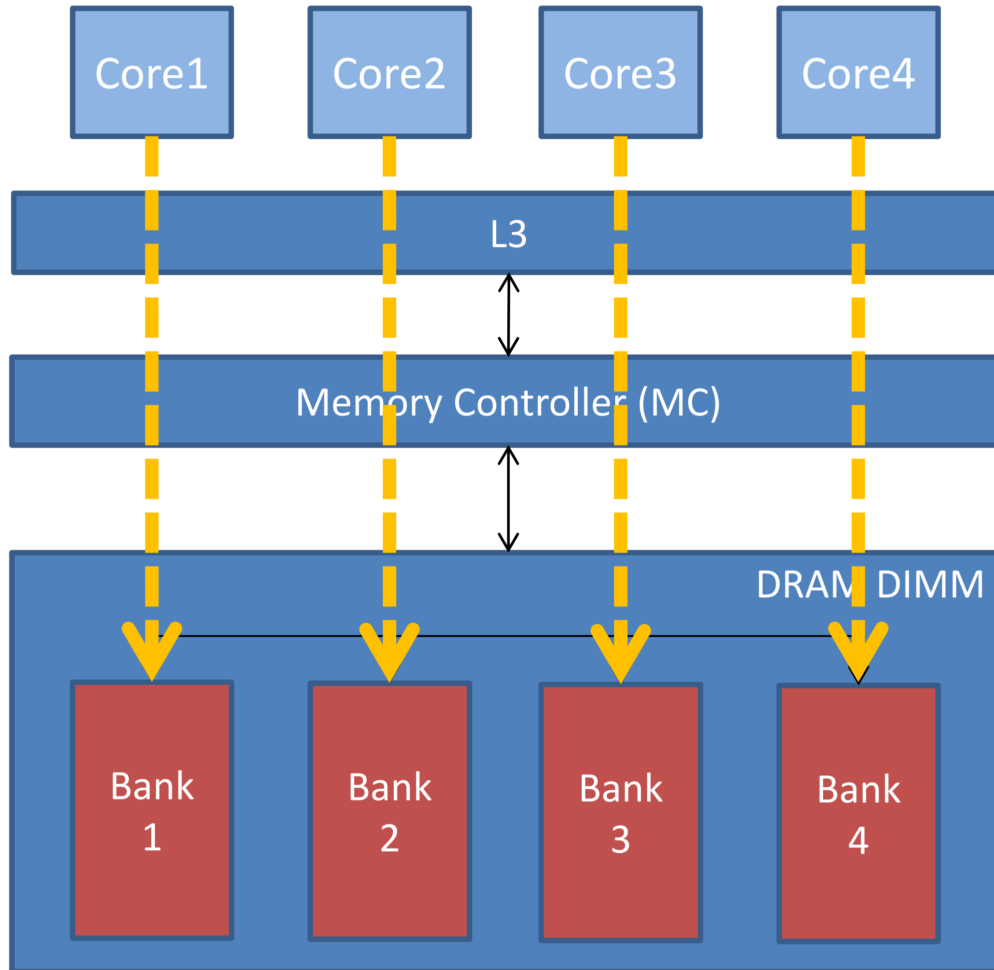
DRAM Organization



- Have multiple banks
- Different banks can be accessed in parallel

KU THE UNIVERSITY OF KANSAS

Best-case

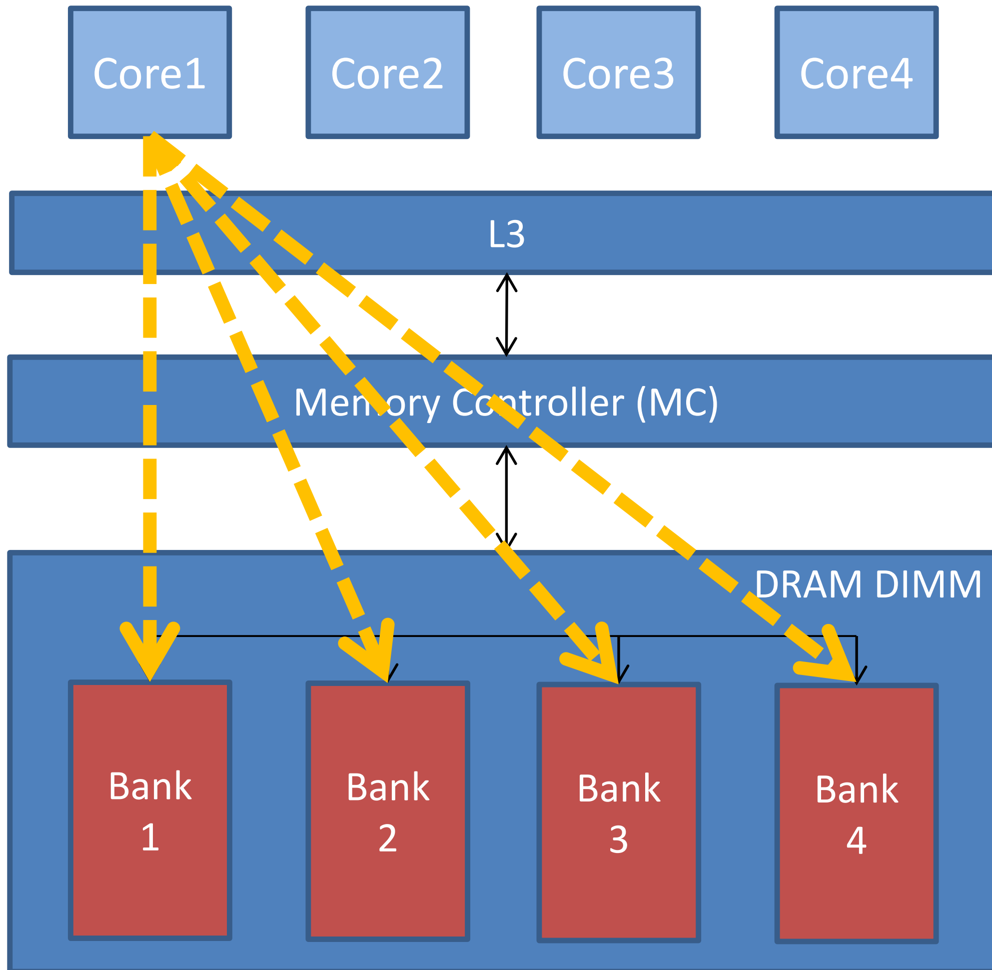


Fast

- Peak = 10.6 GB/s
  - DDR3 1333Mhz

KU THE UNIVERSITY OF KANSAS

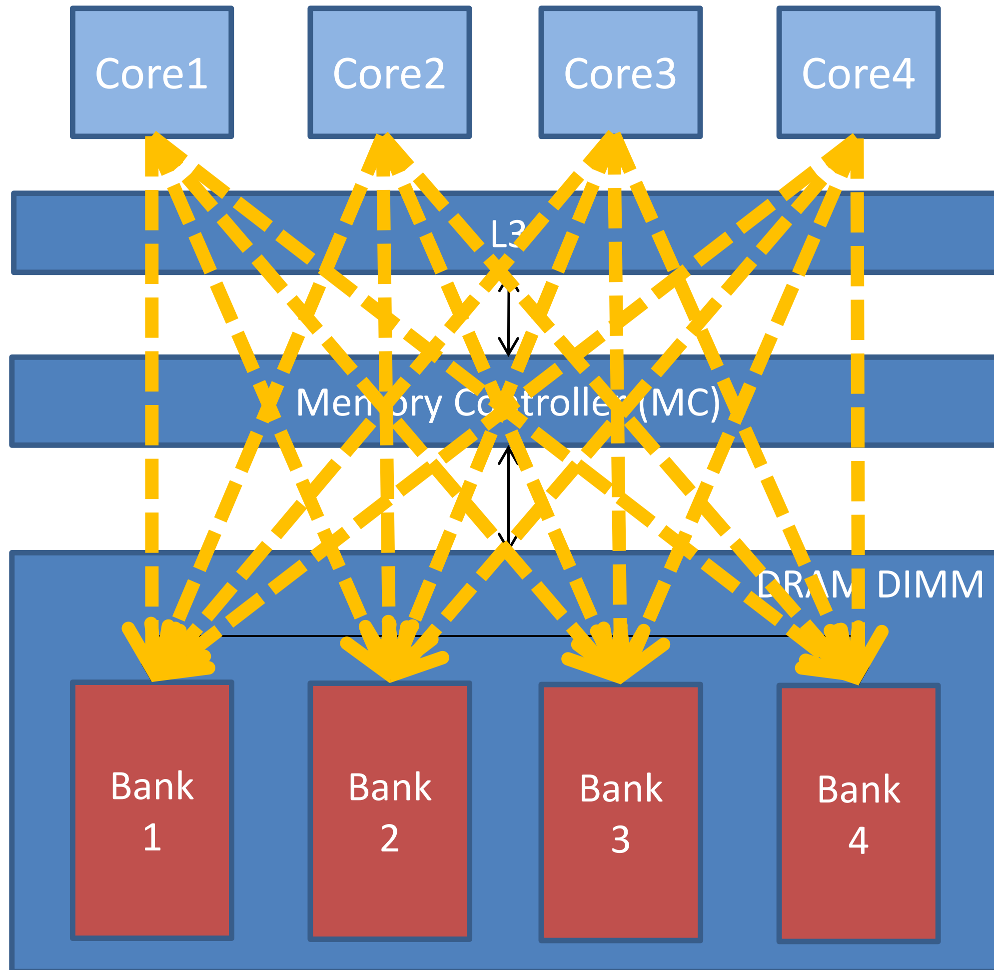
Best-case



Fast

- Peak = 10.6 GB/s
  - DDR3 1333Mhz
- Out-of-order processors

Most-cases

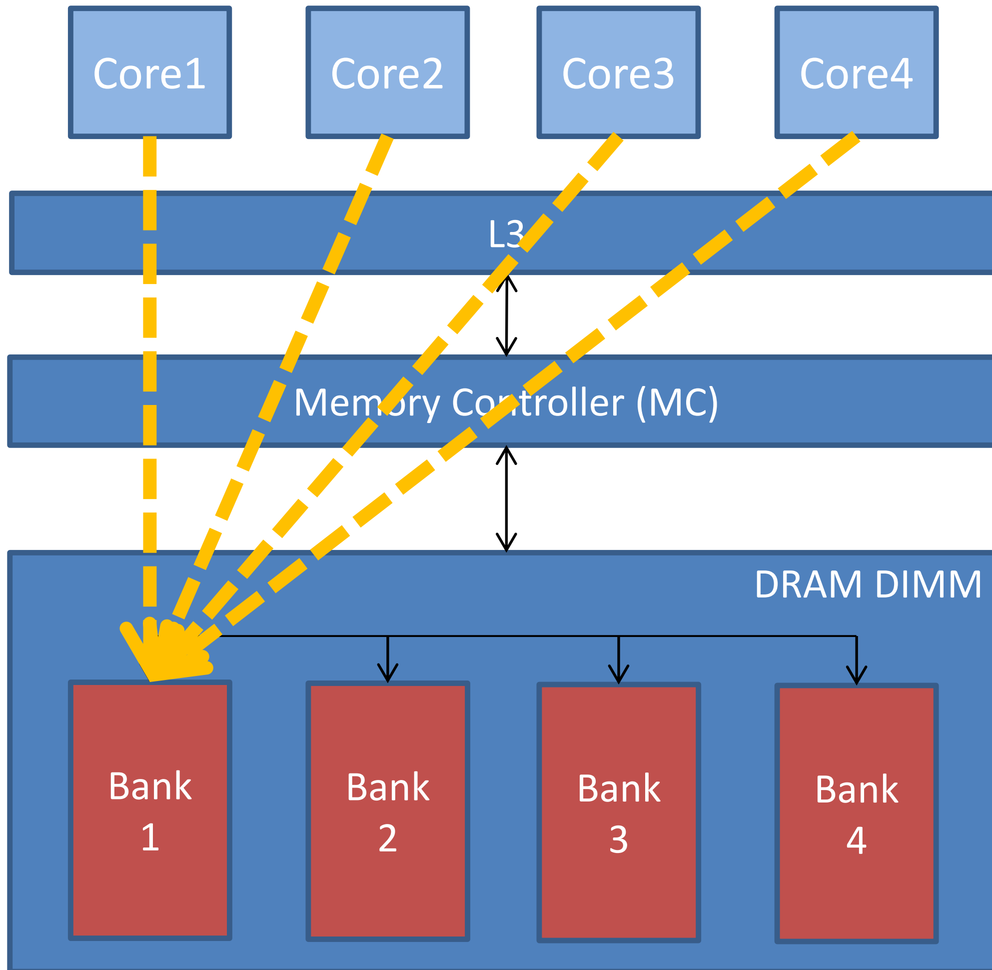


Mess

- Performance = ??

KU THE UNIVERSITY OF KANSAS

Worst-case



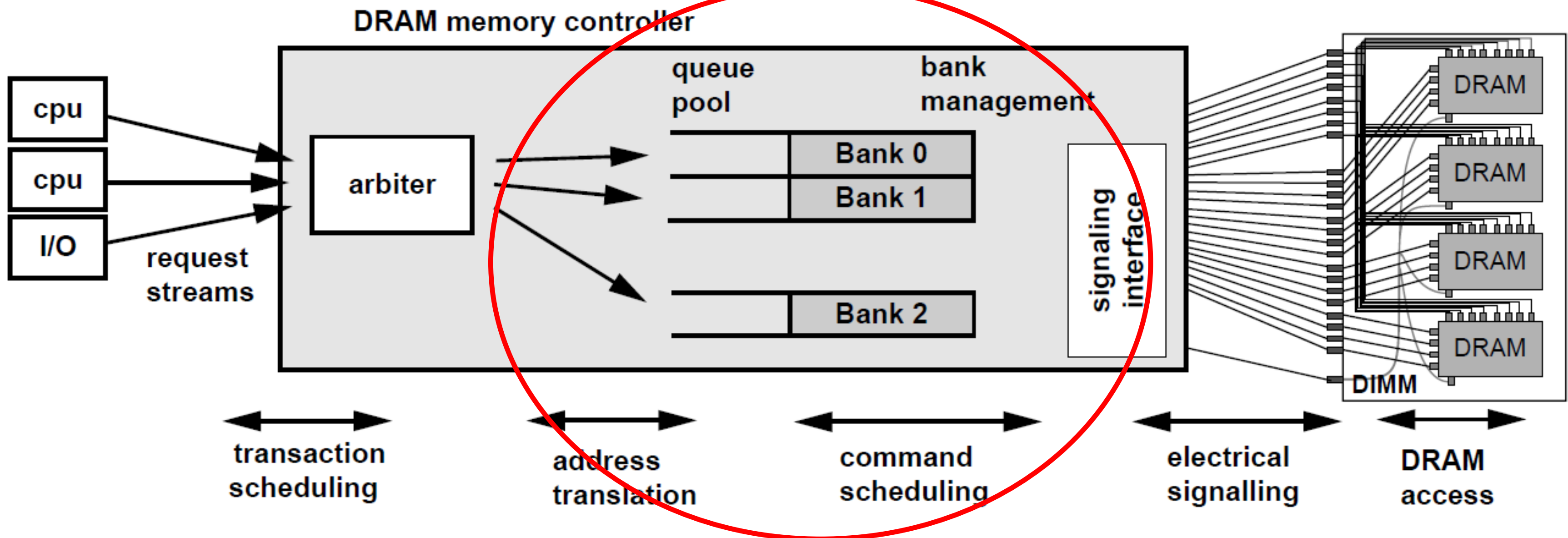
Slow

- 1bank b/w
  - Less than peak b/w
  - How much?

KU THE UNIVERSITY OF KANSAS

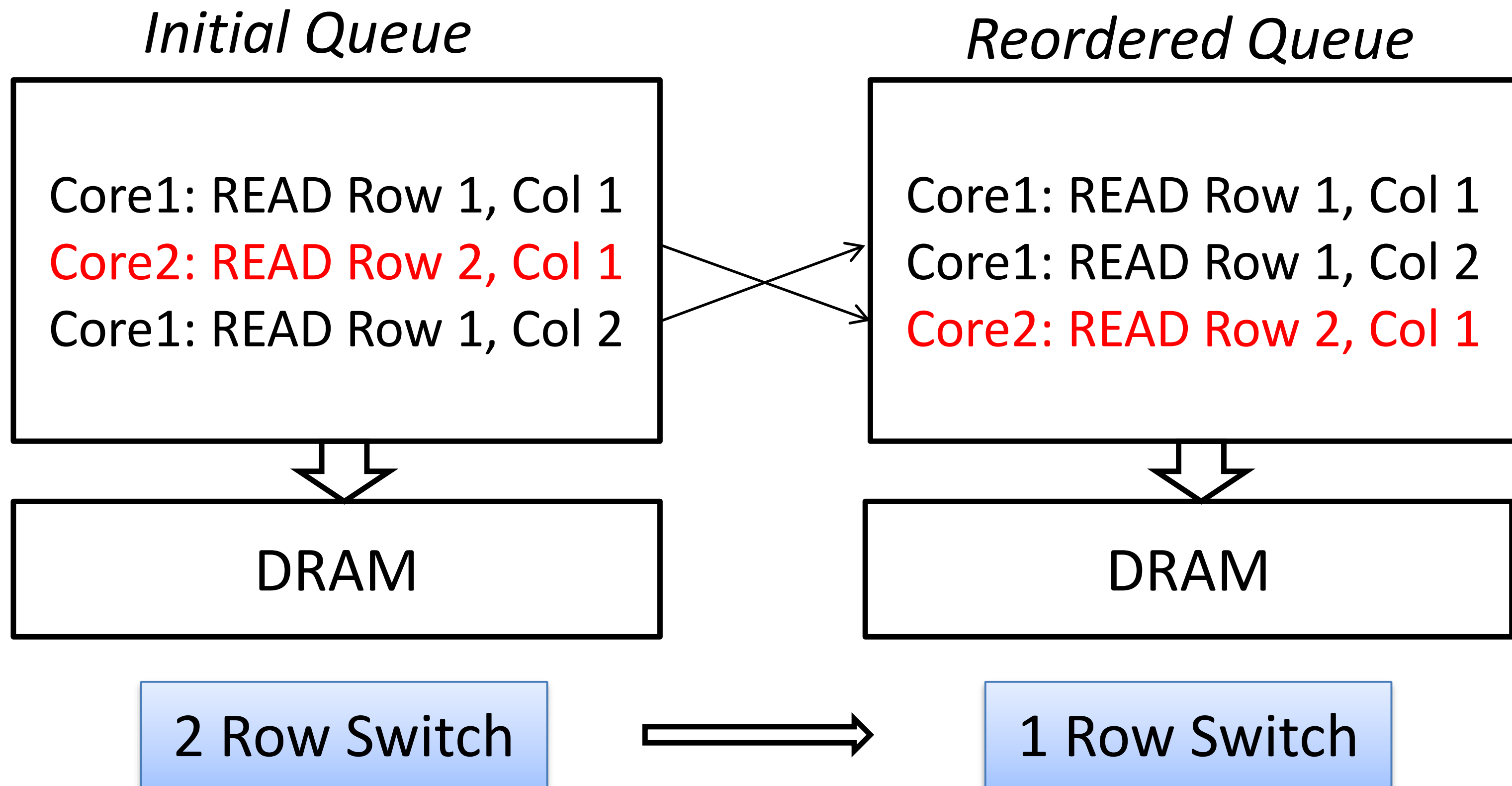
DRAM Controller

Bruce Jacob et al, "Memory Systems: Cache, DRAM, Disk" Fig 13.1.



- Request queue
  - Buffer read/write requests from CPU cores
  - Unpredictable queuing delay due to reordering

Request Reordering



- Improve row hit ratio and throughput
- Unpredictable queuing delay

KU THE UNIVERSITY OF KANSAS

How to Improve Predictability?

- Partitioning
  - Reserve resources (cache space, bank) to tasks
- Throttling
  - Limit access rates to the shared resources
- Scheduling
  - Schedule tasks in ways to avoid contention

Cache Partitioning

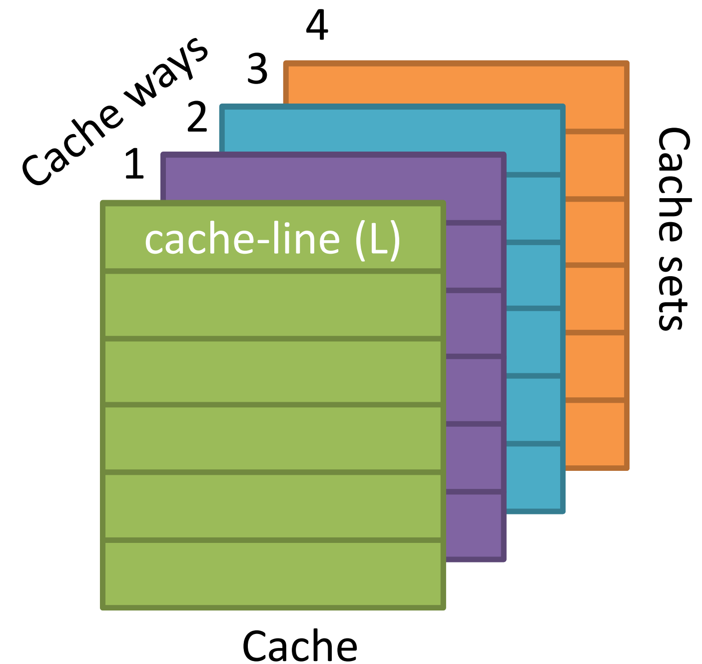
- Divide cache space among cores/tasks
- To improve **throughput** and **isolation**
  - Protect "useful" cache-lines from being evicted can improve throughput
  - Prevent "unwanted" evictions to improve isolation

Cache Partitioning

- Way-partitioning
  - Requires h/w support
- Set-partitioning
  - Can be done in s/w as long as there's MMU.
    - MMU: virtual -> physical address translation h/w
    - Page table: translation table managed by the OS
    - Most (but not all) processors support MMU
  - Page-coloring

Way Partitioning

- H/W support is needed
  - E.g., Freescale P4080, Intel

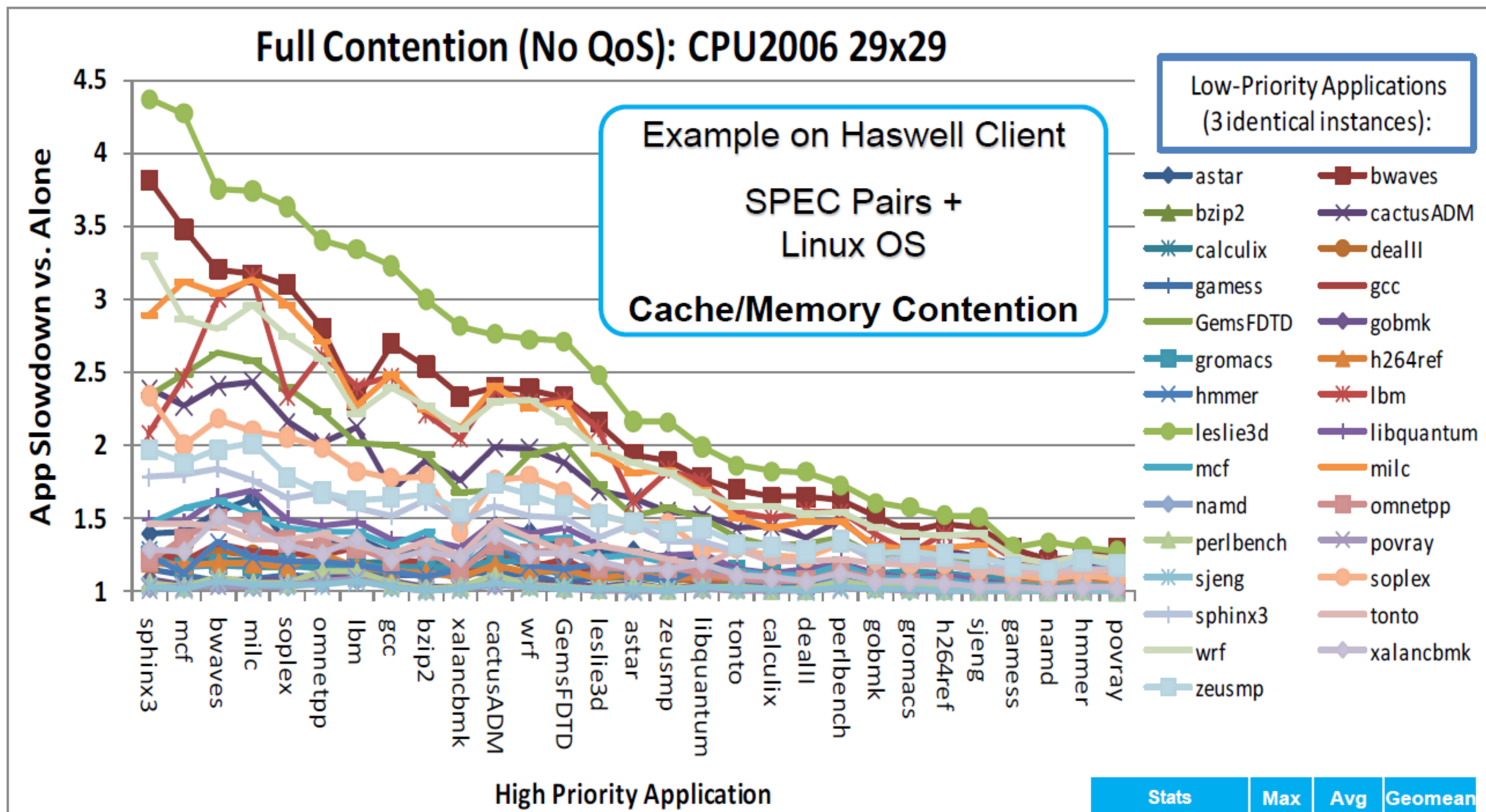


Intel CAT

- Cache Allocation Technology (CAT)
  - Intel’s **way partitioning** mechanism
  - Thread/VM → logical id → resource (cache) partition
- Part of intel’s platform QoS techniques
  - CAT: cache allocation technology
  - CMT: cache monitoring technology
  - MBM: memory bandwidth monitoring
  - CDP: code/data prioritization

KU THE UNIVERSITY OF KANSAS

No QoS: Thread Contention



Stats	Max	Avg	Geomean
Full Contention	4.37x	1.37x	1.31x

Data on Haswell Client (3GHz, 4 cores, 8MB cache, DDR3-1333, SPEC* CPU2006)

Resource contention causes up to 4X slowdown in performance
(Need ability to monitor and enforce cache/memory resource usage)

KU THE UNIVERSITY OF KANSAS

Slide source: [C. Peng, “Achieving QoS in Server Virtualization,” 2016]

With CAT applied: Reduced Thread Contention

CAT Applied: 6MB Dedicated to HP App, LP Apps share 2MB

App Slowdown vs. Alone: 4.5, 4, 3.5, 3, 2.5, 2, 1.5, 1

High Priority Application: sphinx3, mcf, bwaves, milc, soplex, omnetpp, lbm, gcc, bzip2, xalancbmk, cactusADM, wrf, GemsFDTD, leslie3d, astar, zeusmp, libquantum, tonto, calculix, dealII, perlbench, gobmk, gromacs, h264ref, sjeng, gamess, namd, hmmer, povray

CAT Significantly reduces contention

Remaining contention due to memory bandwidth

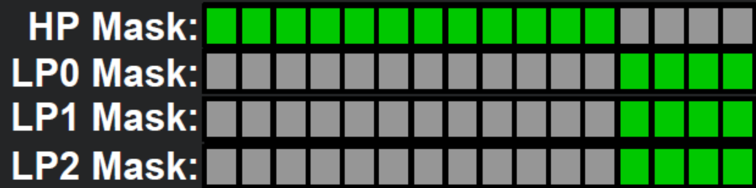
Low-Priority Applications (3 identical instances):

astar, bwaves, bzip2, cactusADM, calculix, dealII, gamess, gcc, GemsFDTD, gobmk, gromacs, h264ref, hmmer, lbm, leslie3d, libquantum, mcf, milc, namd, omnetpp, perlbench, povray, sjeng, soplex, sphinx3, tonto, wrf, xalancbmk, zeusmp

Stats	Max	Avg	Geomean
Full Contention	4.37x	1.37x	1.31x
CAT Applied	3.44x	1.25x	1.22x

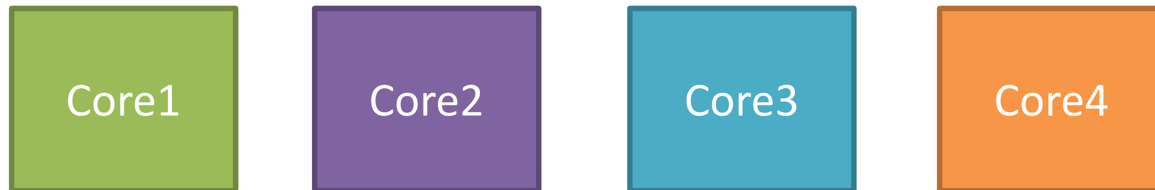
Previous Contention Reduced Substantially!

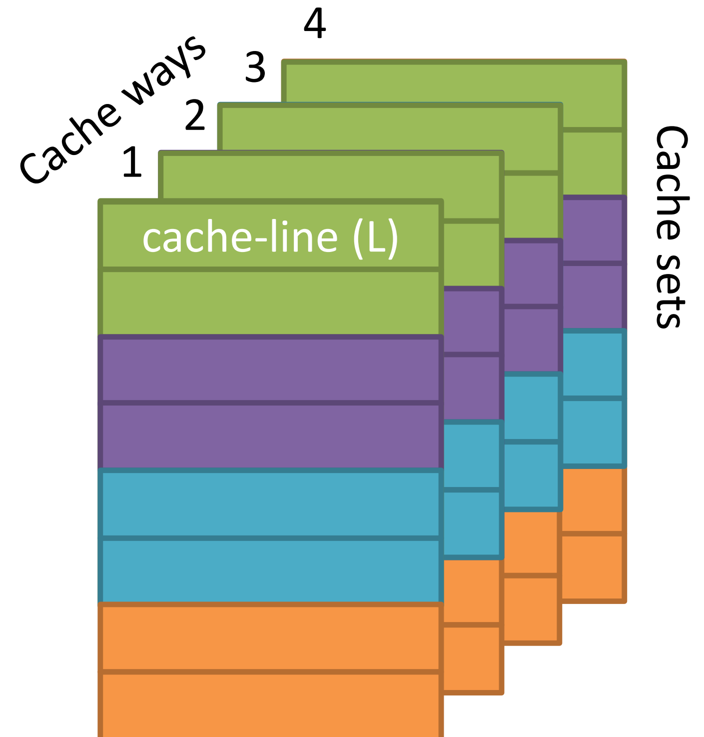
Important Thread=6MB isolated,
3 Low-Priority threads share 2MB



Slide source: C. Peng, “Achieving QoS in Server Virtualization,” 2016

Set Partitioning

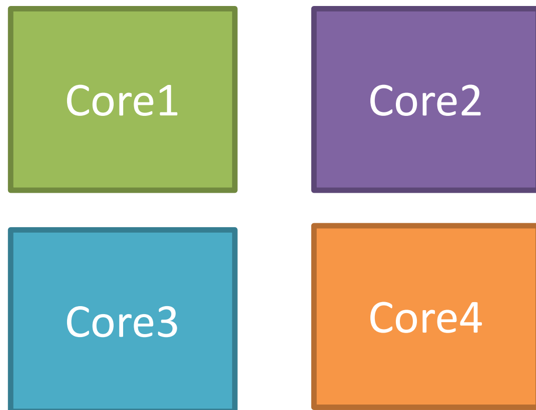


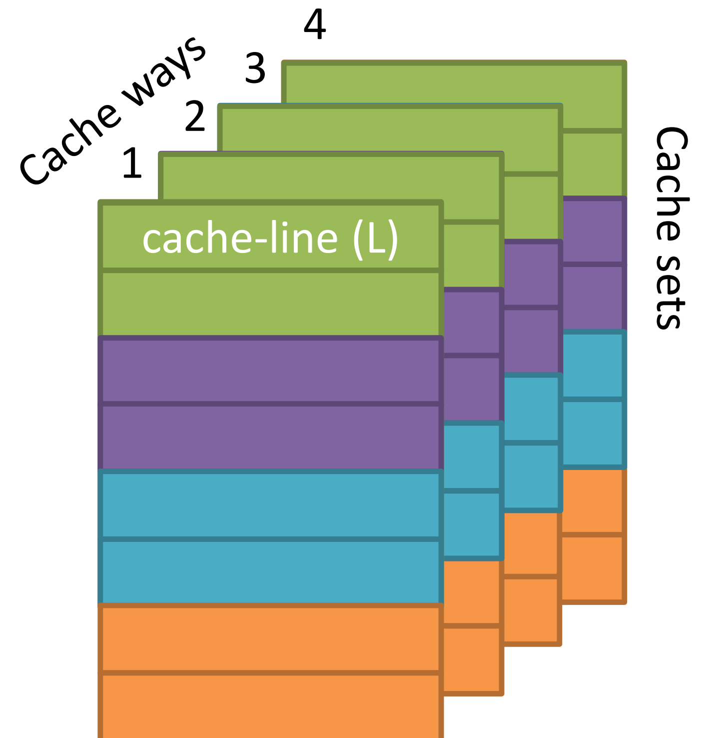


- Can be done in S/W
  - Page coloring: control physical address (cache index) of pages

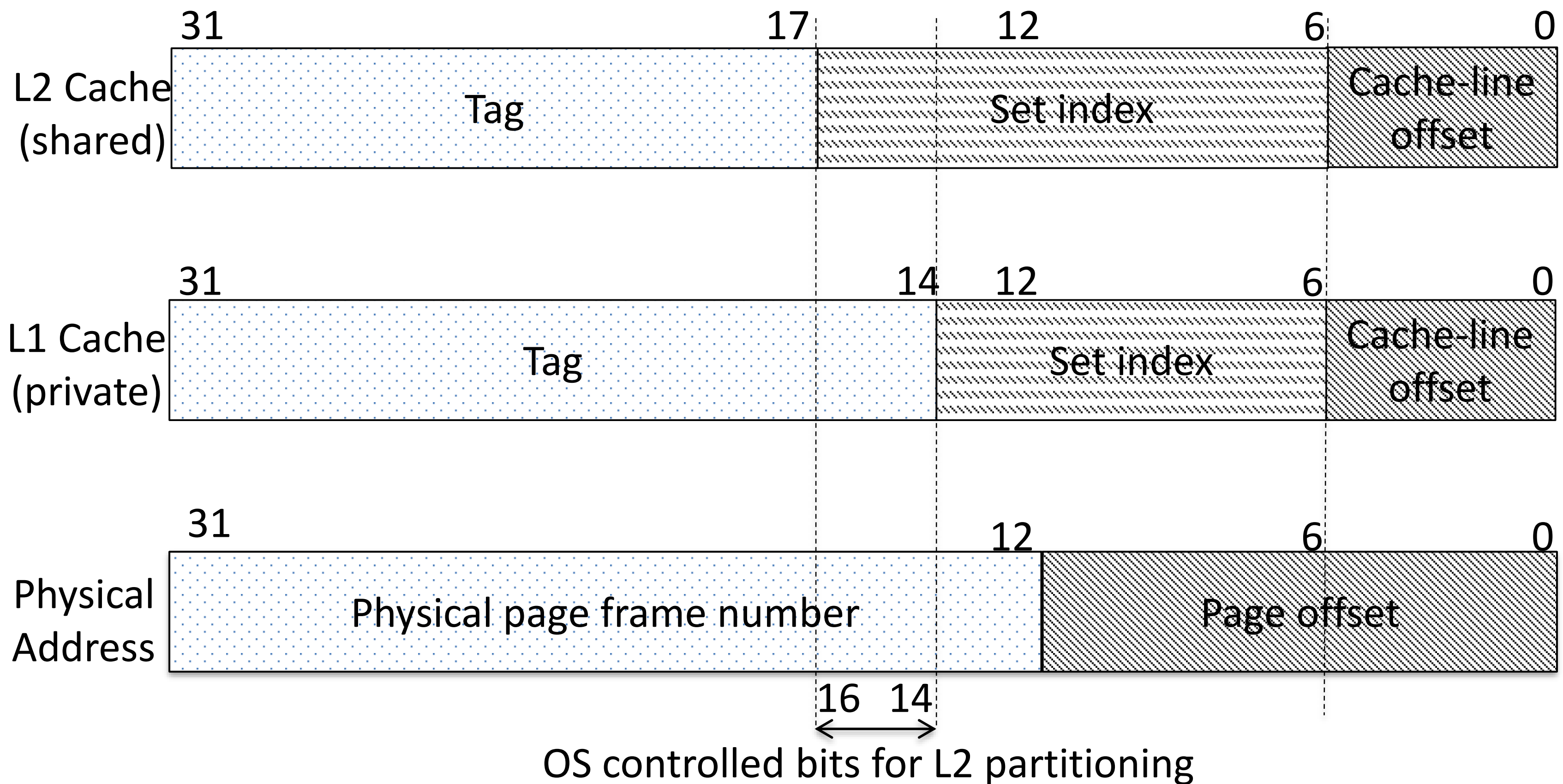
Page Coloring

- Cache can be divided into page colors
- Assign certain colors to certain CPU cores





Page Coloring on Cortex-A15



- OS controls the **color** (bit 14, 15, 16) of allocated memory block to partition the cache

PALLOC: DRAM Bank-Aware Memory Allocator for Performance Isolation on Multicore Platforms

Heechul Yun*, Renato Mancuso+, Zheng-Pei Wu#, Rodolfo Pellizzoni#

*University of Kansas, +University of Illinois , #University of Waterloo

IEEE Real-Time and Embedded Technology and Applications Symposium (***RTAS***), 2014

Problem

SMP OS

Core1 Core2 Core3 Core4

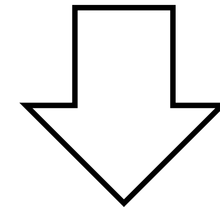
L3

????

DRAM DIMM

Bank 1 | Bank 2 | Bank 3 | Bank 4

- OS does NOT know DRAM banks
- OS memory pages are spread all over multiple banks



Unpredictable memory performance

KU THE UNIVERSITY OF KANSAS

PALLOC

Core1 Core2 Core3 Core4

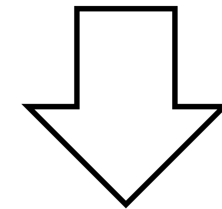
L3

Memory Controller (MC)

DRAM DIMM

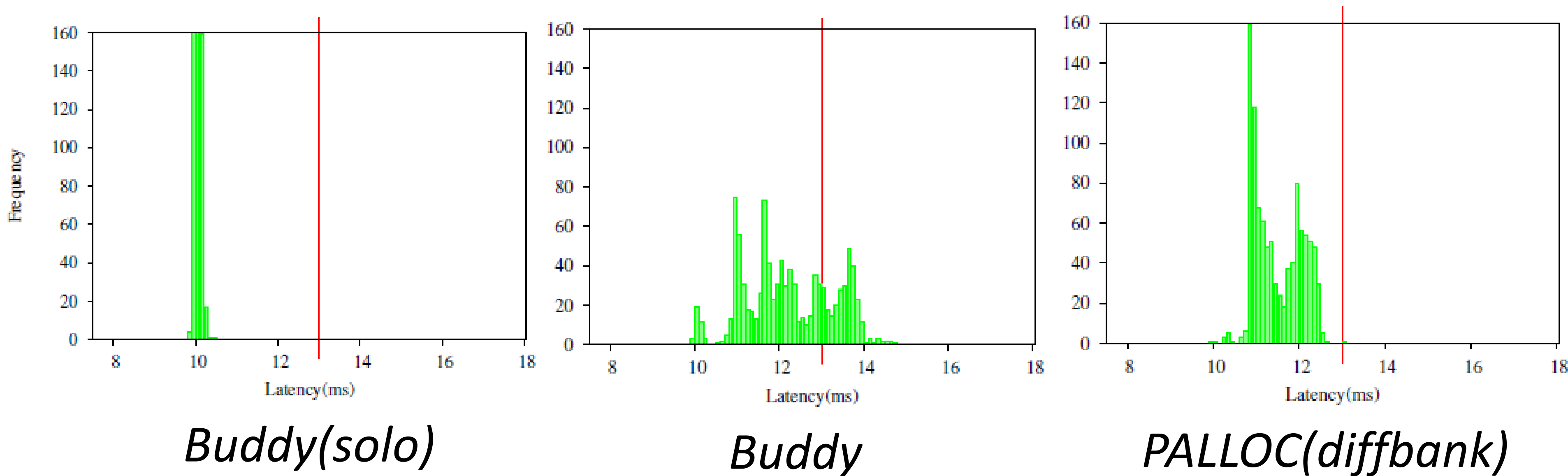
Bank 1 Bank 2 Bank 3 Bank 4

- Private banking
  - *Allocate* pages on certain *exclusively* assigned banks



Eliminate Inter-core bank conflicts

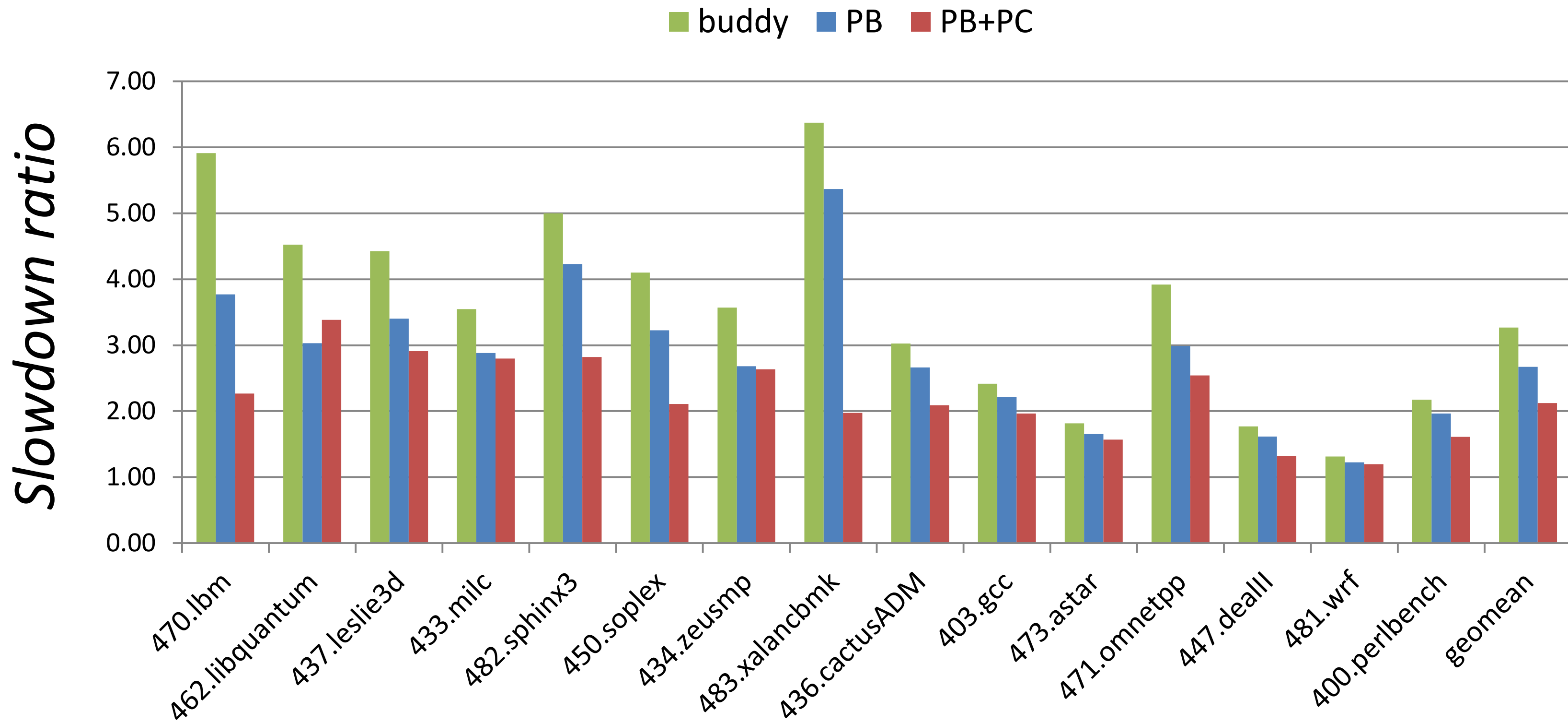
Real-Time Performance



- Setup: HRT → Core0, X-server → Core1
- Buddy: no bank control (use all Bank 0-15)
- Diffbank: Core0 → Bank0-7, Core1 → Bank8-15

KU THE UNIVERSITY OF KANSAS

Performance Isolation on 4 Cores



- Setup: Core0: X-axis, Core1-3: 470.lbm x 3 (interference)
- PB: DRAM bank partitioning only;
- PB+PC: DRAM bank and Cache partitioning
- Finding: bank (and cache) partitioning improves isolation, but far from ideal

KU THE UNIVERSITY OF KANSAS

Taming Non-blocking Caches to Improve Isolation in Multicore Real-Time Systems

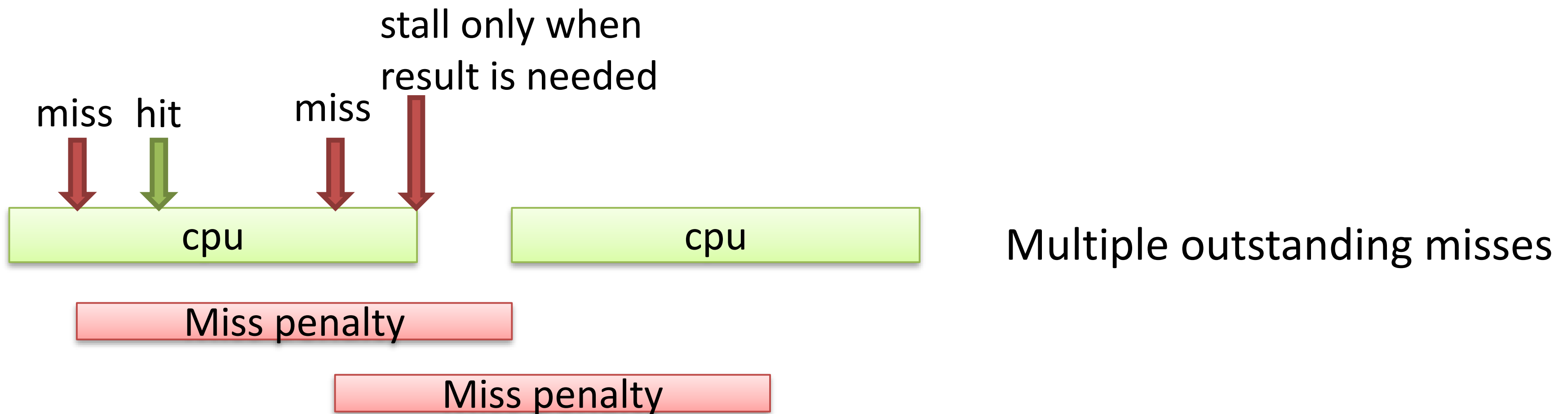
Prathap Kumar Valsan, Heechul Yun, Farzad Farshchi

University of Kansas

*IEEE Real-Time and Embedded Technology and Applications Symposium (**RTAS**)*, 2016

Best Paper Award

Non-blocking Cache



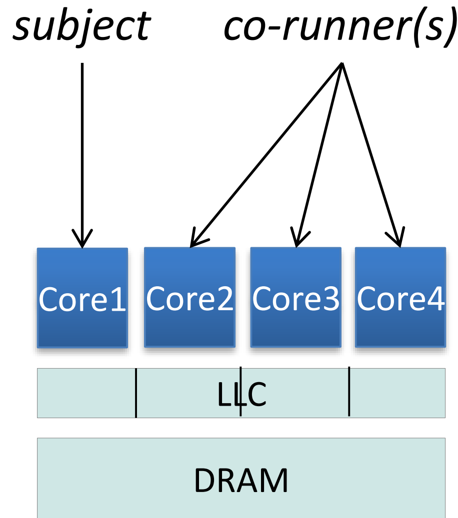
- Can serve cache hits under multiple cache misses
  - Essential for an out-of-order core and any multicore

- Miss-Status-Holding Registers (MSHRs)
  - On a miss, allocate a MSHR entry to track the req.
  - On receiving the data, clear the MSHR entry

KU THE UNIVERSITY OF KANSAS

(*) D. Kroft. "Lockup-free instruction fetch/prefetch cache organization," ISCA'81

Cache Interference Experiments



- Measure the performance of the 'subject'
  - (1) alone, (2) with co-runners
  - LLC is partitioned (equal partition) using PALLOC (*)
- Q: Does cache partitioning provide isolation?

KU THE UNIVERSITY OF KANSAS

(*) Heechul Yun, Renato Mancuso, Zheng-Pei Wu, Rodolfo Pellizzoni. "PALLOC: DRAM Bank-Aware Memory Allocator for Performance Isolation on Multicore Platforms." RTAS'14

IsolBench: Synthetic Workloads

Experiment	Subject	Co-runner(s)
Exp. 1	Latency(LLC)	BwRead(DRAM)
Exp. 2	BwRead(LLC)	BwRead(DRAM)
Exp. 3	BwRead(LLC)	BwRead(LLC)
Exp. 4	Latency(LLC)	BwWrite(DRAM)
Exp. 5	BwRead(LLC)	BwWrite(DRAM)
Exp. 6	BwRead(LLC)	BwWrite(LLC)

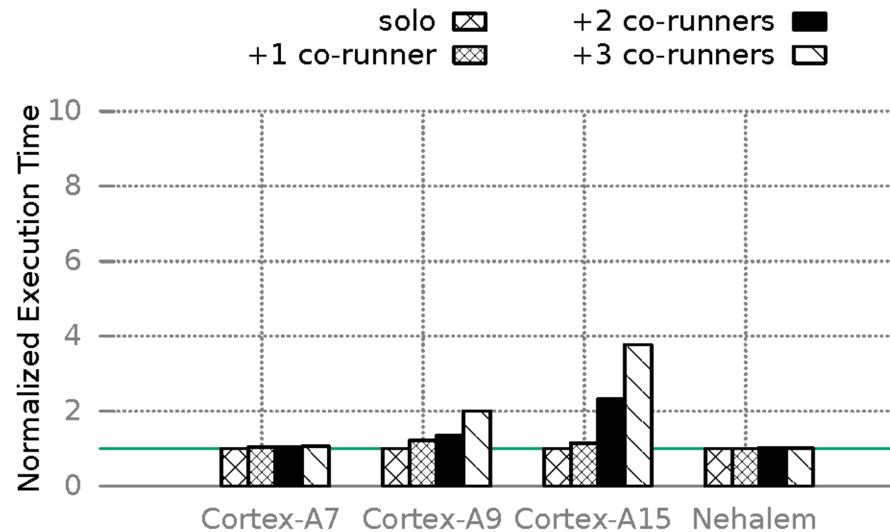
Working-set size: (LLC) < ¼ LLC → cache-hits, (DRAM) > 2X LLC → cache misses

- Latency
  - A linked-list traversal, data dependency, one outstanding miss
- Bandwidth
  - An array reads or writes, no data dependency, multiple misses

- **Subject** benchmarks: **LLC partition fitting**

KU THE UNIVERSITY OF KANSAS

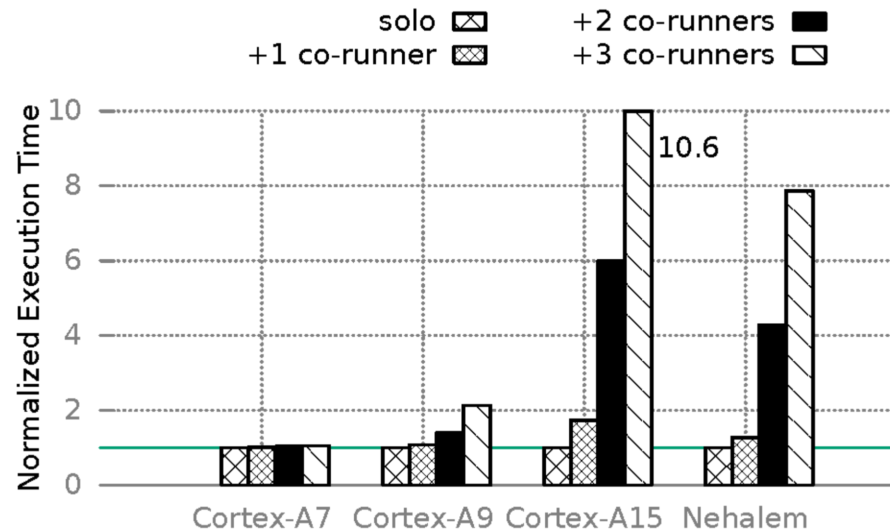
Latency(LLC) vs. BwRead(DRAM)



- No interference on Cortex-A7 and Nehalem
- On Cortex-A15, Latency(LLC) suffers 3.8X slowdown
  - despite partitioned LLC

KU THE UNIVERSITY OF KANSAS

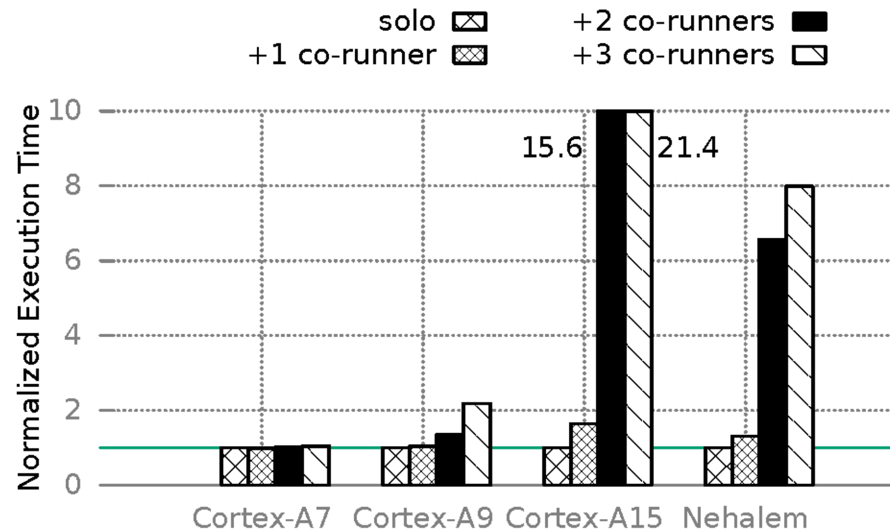
BwRead(LLC) vs. BwRead(DRAM)



- Up to 10.6X slowdown on Cortex-A15
- Cache partitioning != performance isolation
  - On all tested out-of-order cores (A9, A15, Nehalem)

KU THE UNIVERSITY OF KANSAS

BwRead(LLC) vs. BwWrite(DRAM)



- Up to 21X slowdown on Cortex-A15
- Writes generally cause more slowdowns
  - Due to write-backs

KU THE UNIVERSITY OF KANSAS

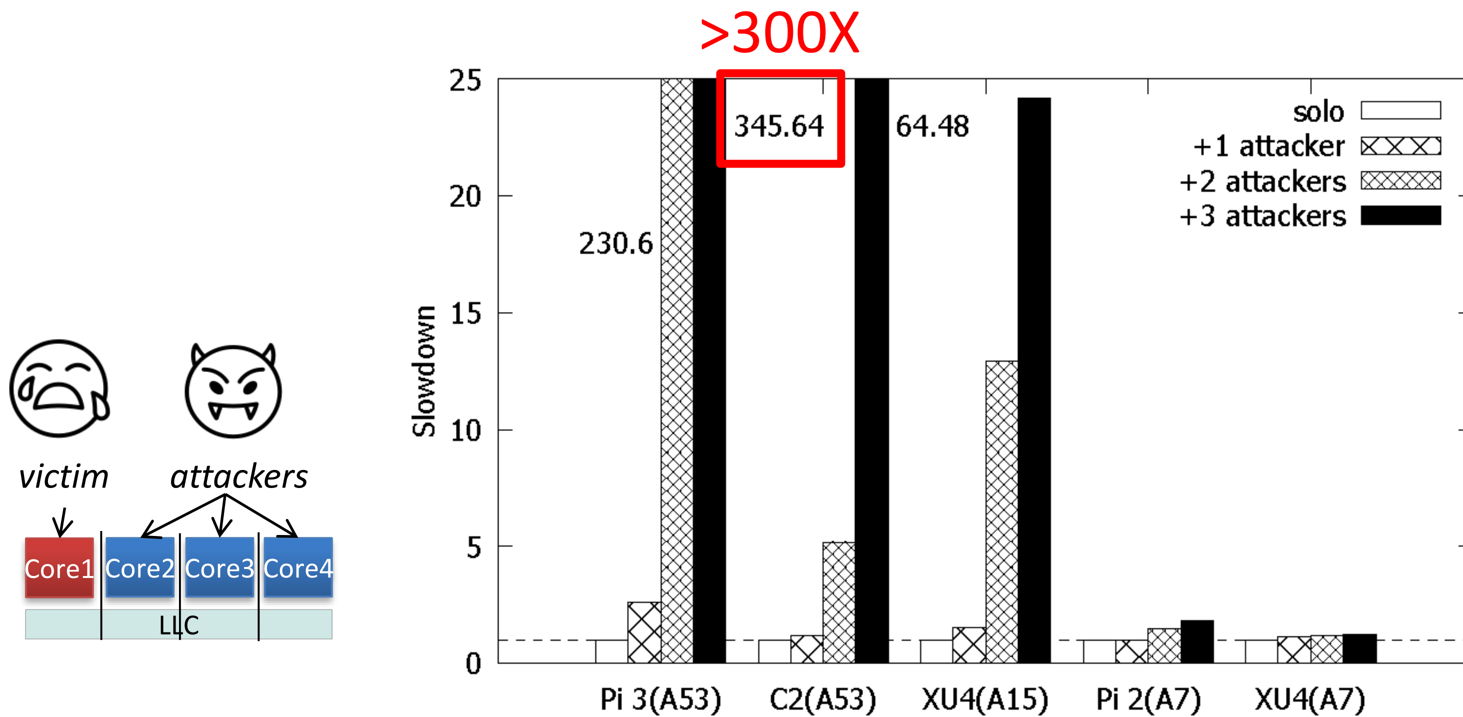
Denial-of-Service Attacks on Shared Cache in Multicore: Analysis and Prevention

Michael Garrett Bechtel and Heechul Yun
University of Kansas

*IEEE Real-Time and Embedded Technology and Applications Symposium (**RTAS**),* 2019

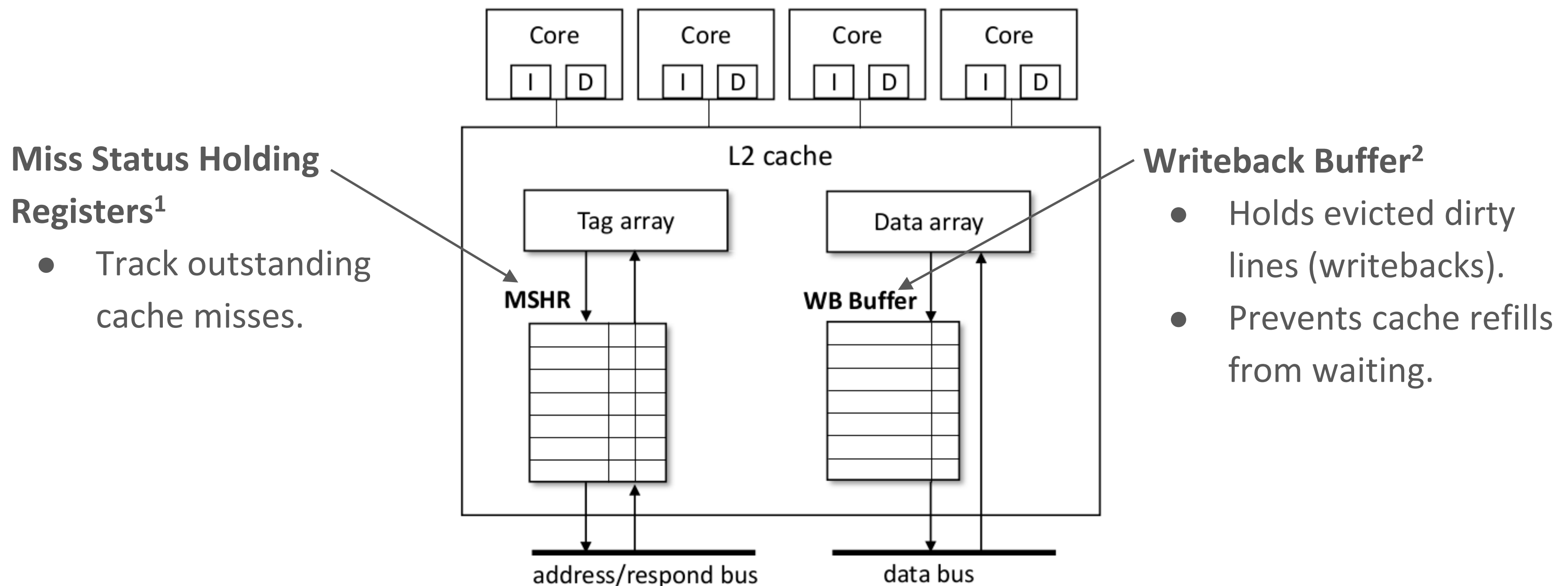
Outstanding Paper Award

Effects of Cache DoS Attacks



- Observed worst-case: >300X (times) slowdown
  - On popular in-order multicore processors
  - Due to contention in cache write-back buffer

Non-Blocking Cache



- We identified cache internal structures that are potential DoS attack vectors

[1] P. K. Valsan, H. Yun, F. Farshchi. "Taming Non-blocking Caches to Improve Isolation in Multicore Real-Time Systems." In *RTAS*, 2016
[2] M. G. Bechtel and H. Yun. "Denial-of-Service Attacks on Shared Cache in Multicore: Analysis and Prevention." In *RTAS*, 2019

How to Improve Predictability?

- Partitioning
  - Reserve resources (cache space, bank) to tasks
- **Throttling**
  - **Limit access rates to the shared resources**
- Scheduling
  - Schedule tasks in ways to avoid contention

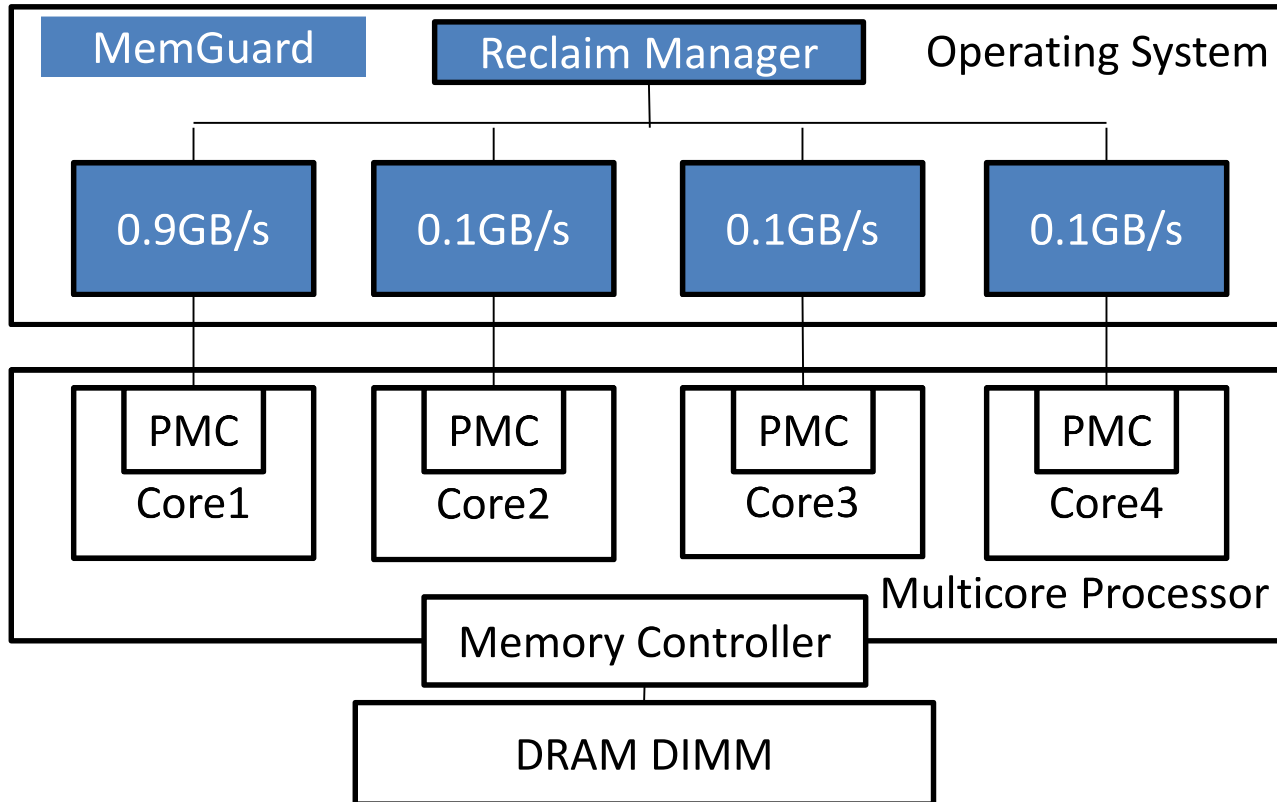
MemGuard: Memory Bandwidth Reservation System for Efficient Performance Isolation in Multi-core Platforms

Heechul Yun[+], Gang Yao[+], Rodolfo Pellizzoni[*], Marco Caccamo[+], Lui Sha[+]

[+]University of Illinois, [*]University of Waterloo

*IEEE Real-Time and Embedded Technology and Applications Symposium (**RTAS**), 2013*

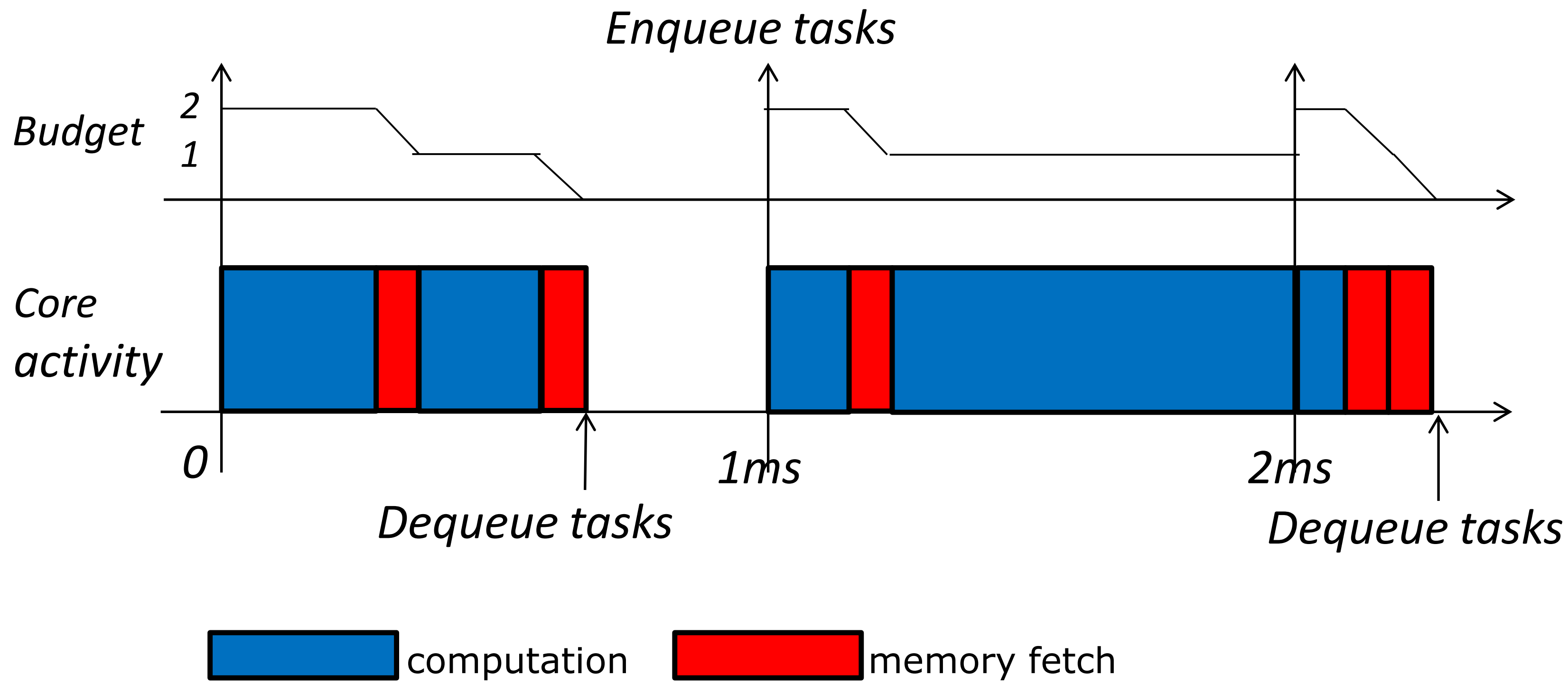
MemGuard



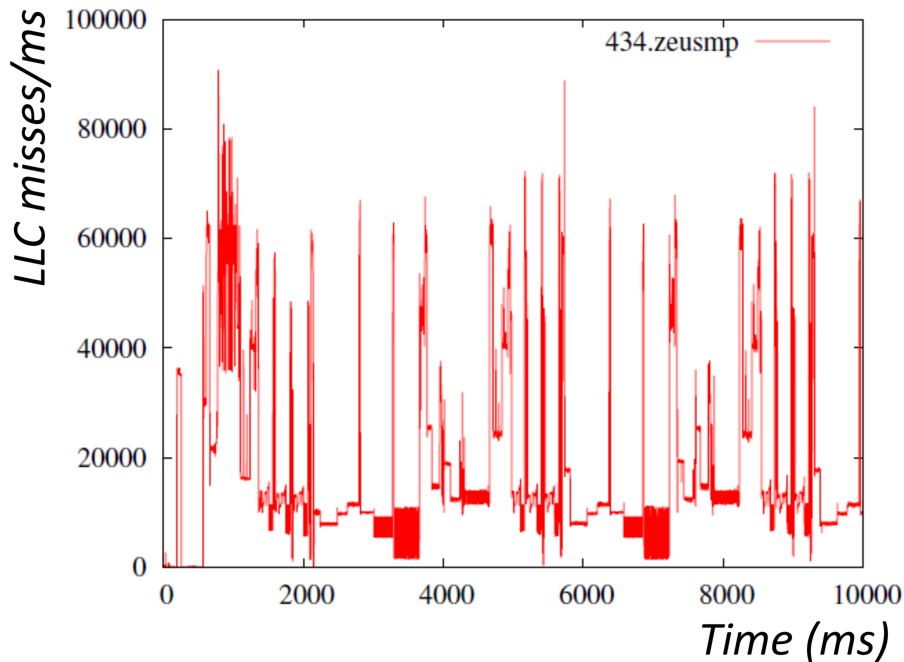
- Memory bandwidth management system

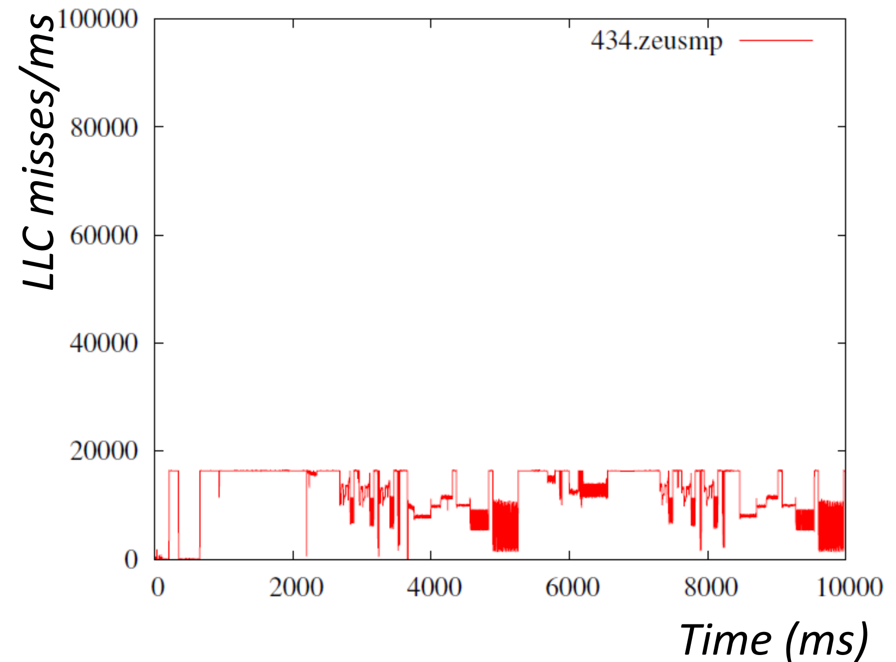
Memory Bandwidth Throttling

- Idea
  - OS **monitor** and **enforce** each core's memory bandwidth usage



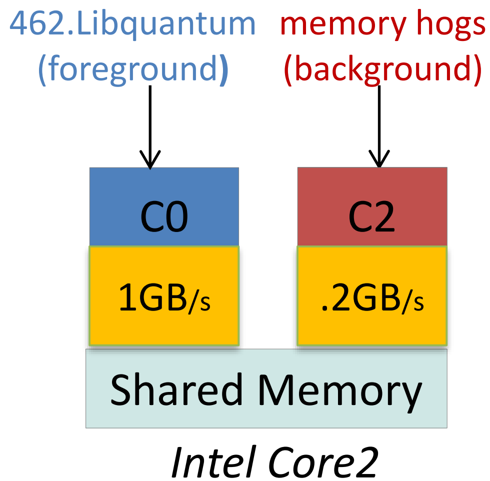
Impact of Throttling

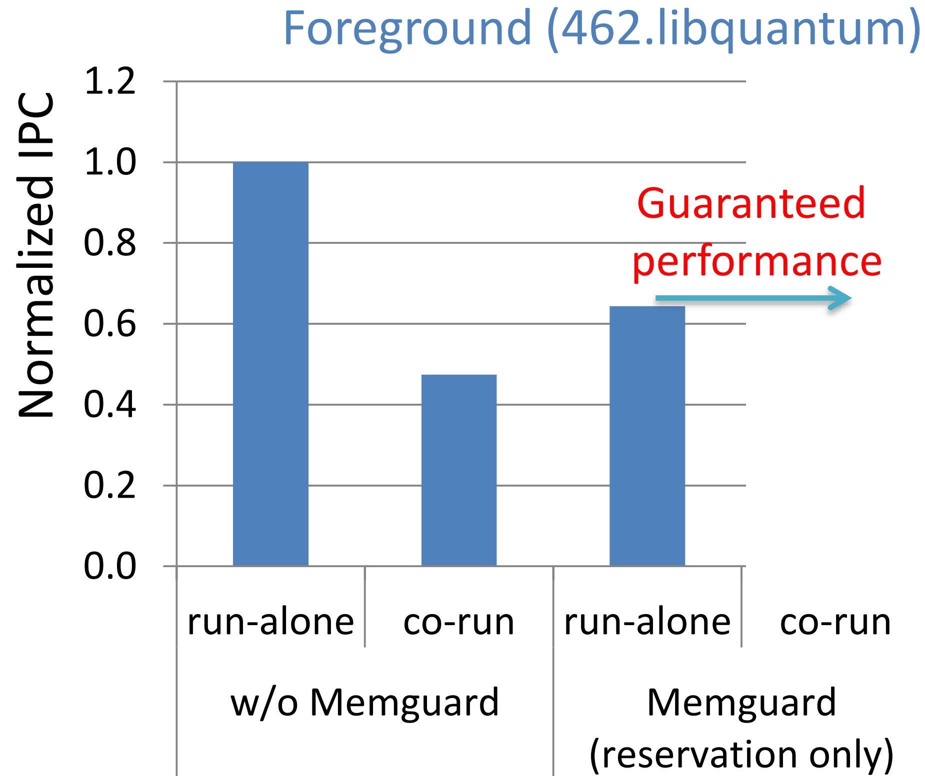




KU THE UNIVERSITY OF KANSAS

Evaluation Results





Reservation provides performance isolation

KU THE UNIVERSITY OF KANSAS

How to Improve Predictability?

- Partitioning
  - Reserve resources (cache space, bank) to tasks
- **Throttling**
  - **Limit access rates to the shared resources**
- **Scheduling**
  - **Schedule tasks in ways to avoid contention**

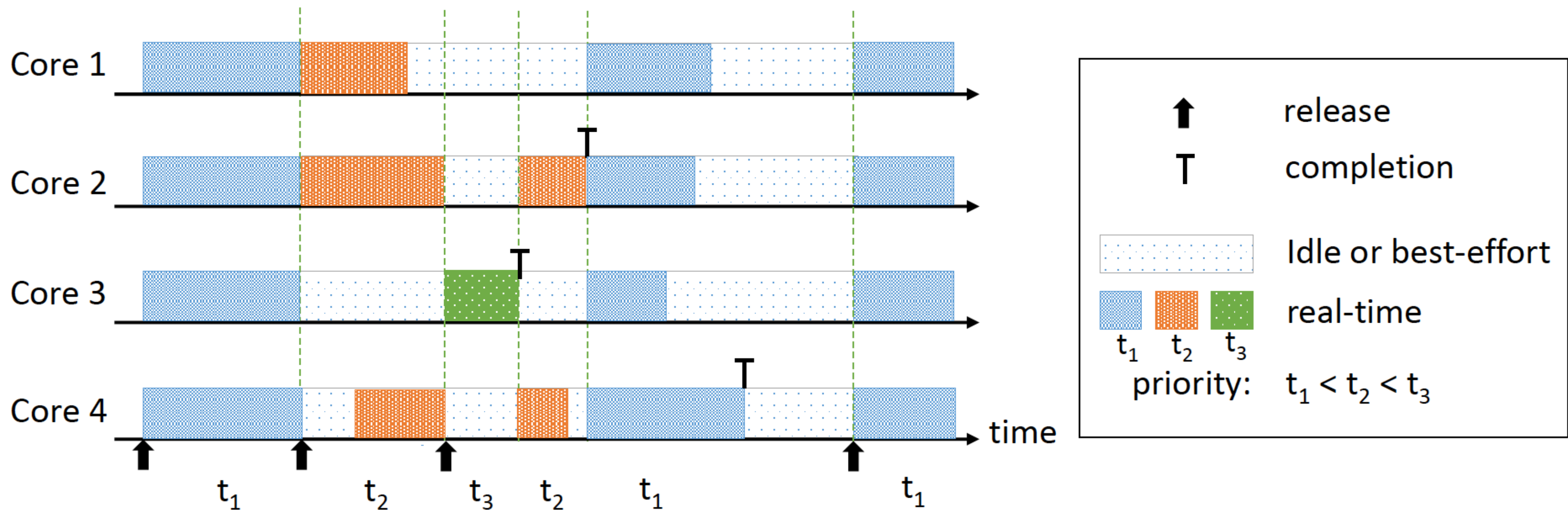
RT-Gang: Real-Time Gang Scheduling Framework for Safety-Critical Systems

Waqar Ali and Heechul Yun

University of Kansas

*IEEE Real-Time and Embedded Technology and Applications Symposium (**RTAS**)*, 2019

RT-Gang



- **One (parallel) real-time task---a gang---at a time**
  - Eliminate inter-task interference by construction
- Schedule best-effort tasks during slacks **w/ throttling**
  - Improve utilization with bounded impacts on the RT tasks

KU THE UNIVERSITY OF KANSAS

Implementation

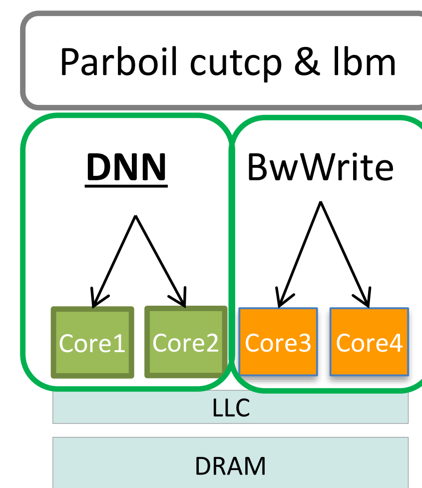
- Modified Linux's RT scheduler
  - Implemented as a "feature" of SCHED_FIFO (sched/rt.c)
- Best-effort task throttling
  - Based on BWLOCK++[*]

[*] W. Ali and H. Yun., "Protecting Real-Time GPU Kernels on Integrated CPU-GPU SoC Platforms." In *ECRTS*, 2018

Experiment Setup

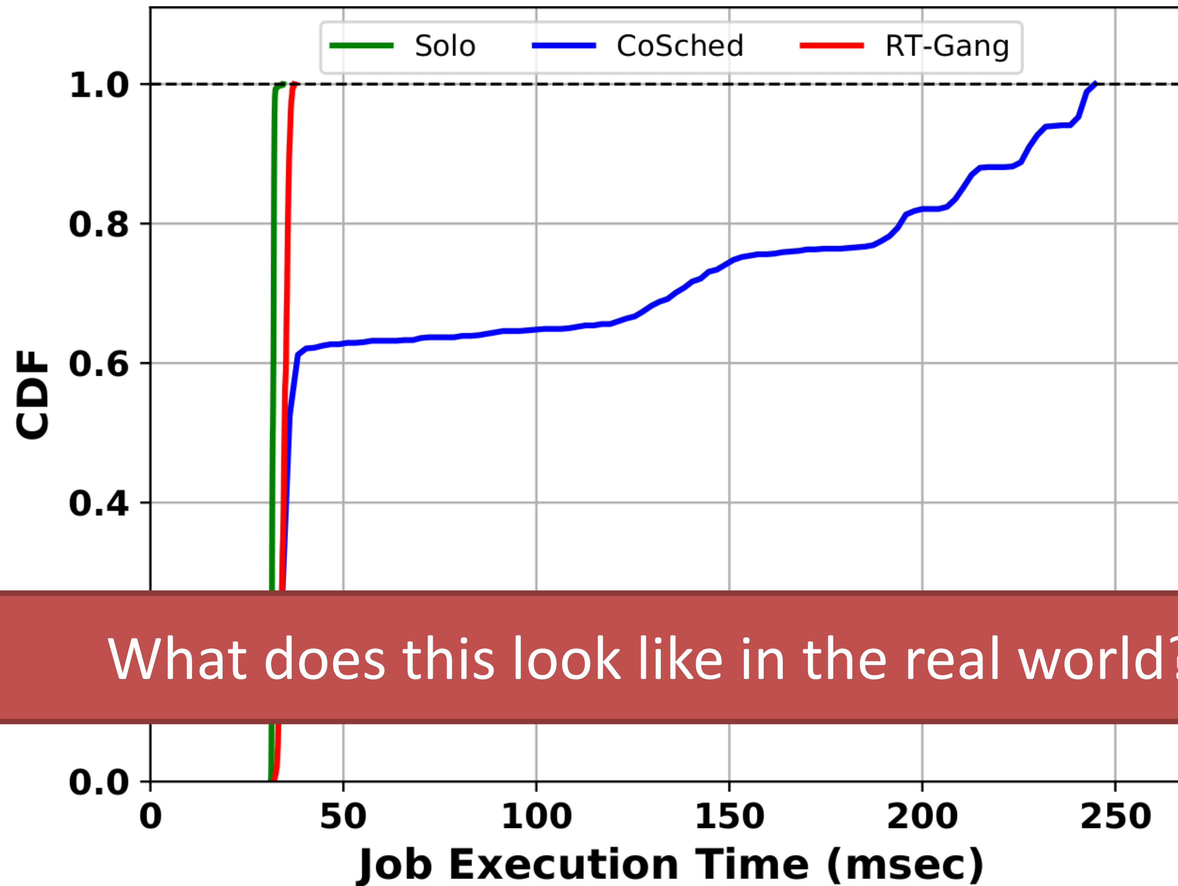
- DNN control task of DeepPicar (real-world RT)
- IsolBench BwWrite benchmark (synthetic RT)
- Parboil benchmarks (real-world BE)

	Task	WCET (C ms)	Period (P ms)	# Threads
RT	t^{rt}_{dnn}	**34**	**100**	**2**
	t^{rt}_{bww}	**220**	**340**	**2**
BE	t^{be}_{cutcp}	∞	N/A	4
	t^{be}_{lbm}	∞	N/A	4



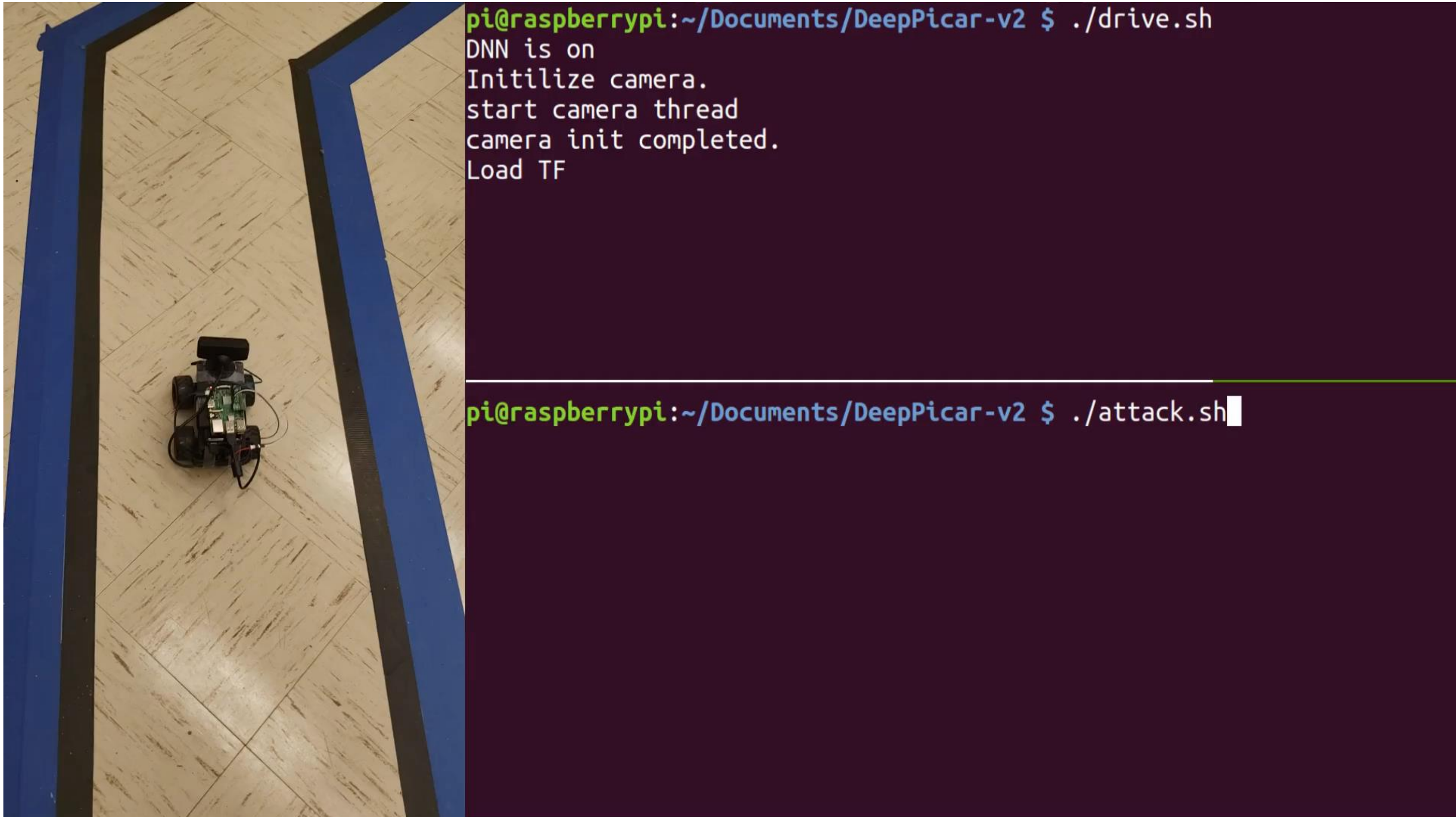
KU THE UNIVERSITY OF KANSAS

Execution Time Distribution



- RT-Gang achieves deterministic timing

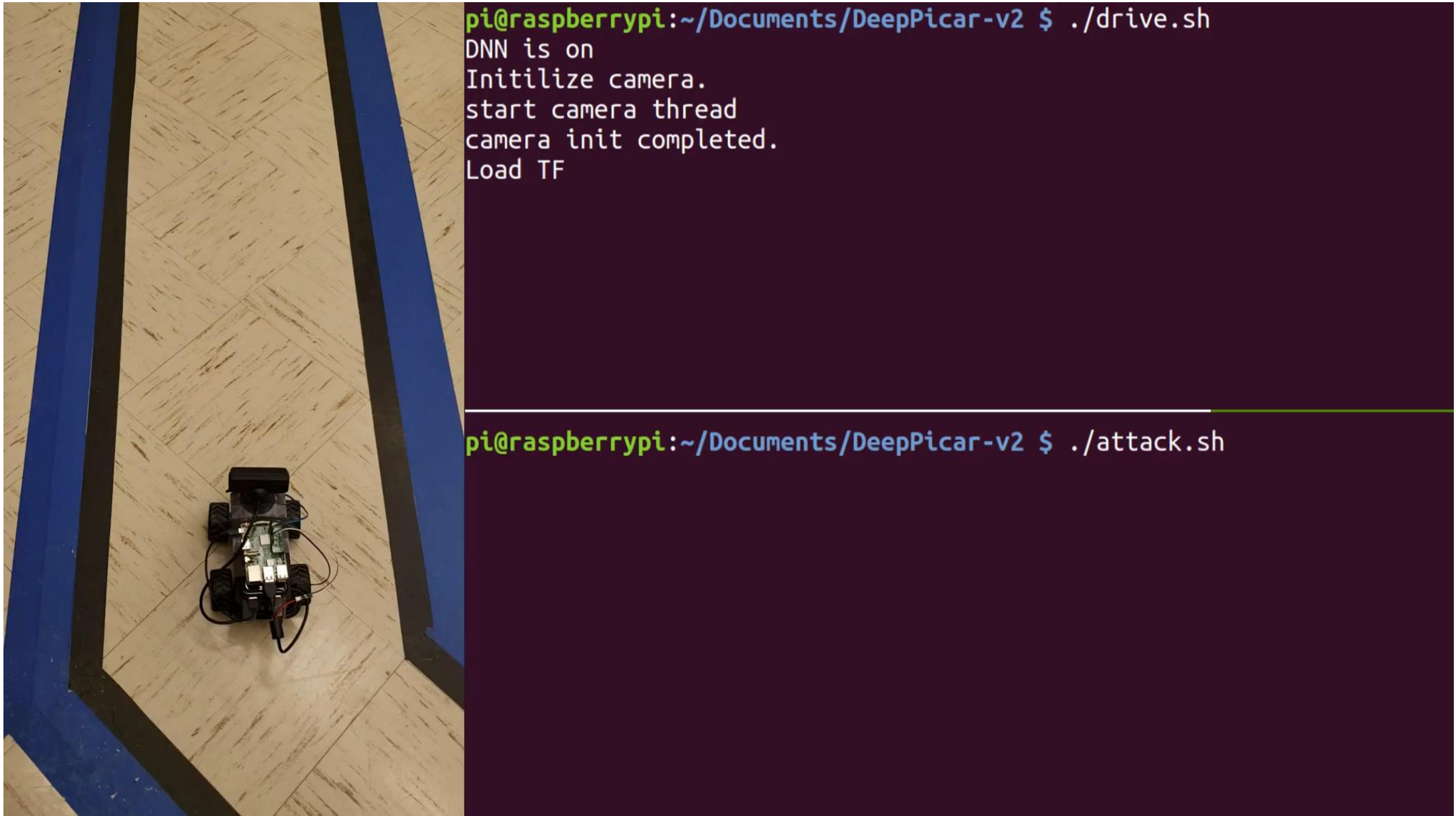
CoSched (w/o RT-Gang)



https://youtu.be/Jm6KSDqlqiU

KU THE UNIVERSITY OF KANSAS

RT-Gang



https://youtu.be/pk0j063cUAs

KU THE UNIVERSITY OF KANSAS

Summary

- Real-time != Real-fast
  - Real-time: about predictability
  - Real-fast: about average performance
- Real-fast chips are often bad for real-time
- Because timing is highly unpredictable on most real-fast chips.
- Traditional real-time systems use simple micro-controllers (like HiFive1), which are predictable
- But, they cannot run complex stuff (e.g., AI)
- Increasingly, we need both: real-time & real-fast

KU THE UNIVERSITY OF KANSAS

Acknowledgements

- Some slides draw on materials developed by
  - Edward A. Lee and Prabal Dutta (UCB) for EECS149/249A

KU
THE UNIVERSITY OF KANSAS