EECS 388: Embedded Systems

10. Timing Analysis

Heechul Yun

Agenda

- Execution time analysis
- Static timing analysis
- Measurement based timing analysis

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)?

KU THE UNIVERSITY OF KANSAS

Execution Time

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

Execution Time

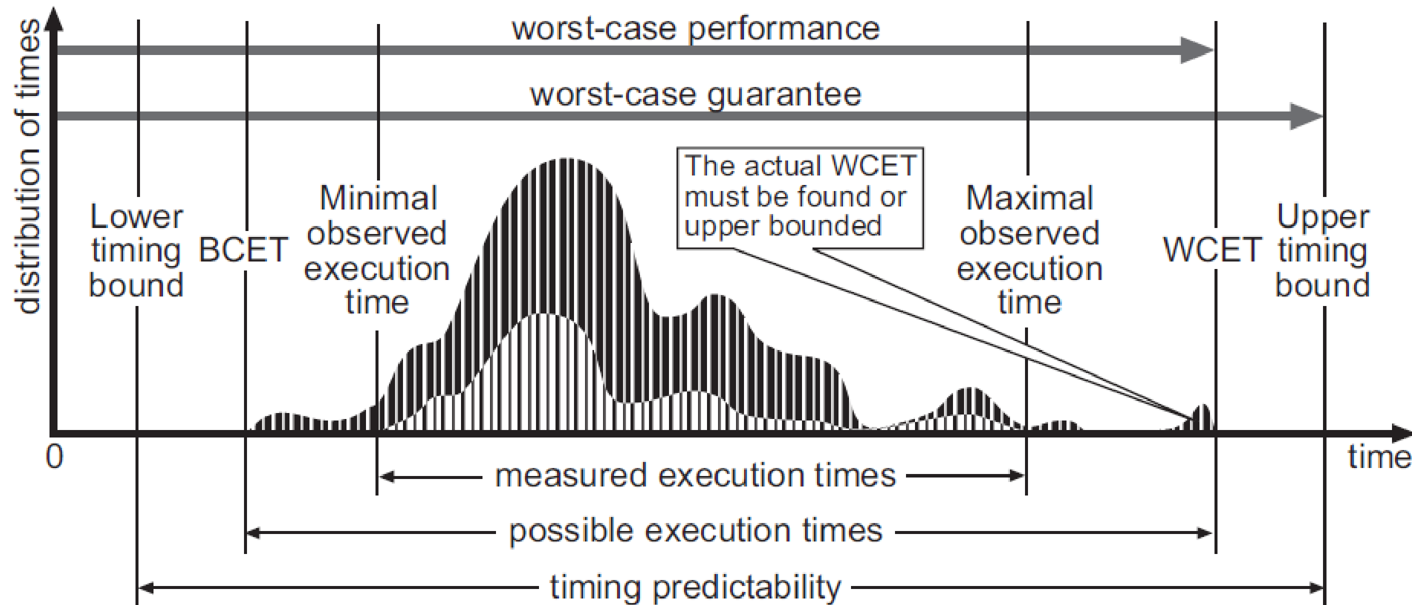


Image source: [Wilhelm et al., 2008]

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

KU THE UNIVERSITY OF KANSAS

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();
}
```

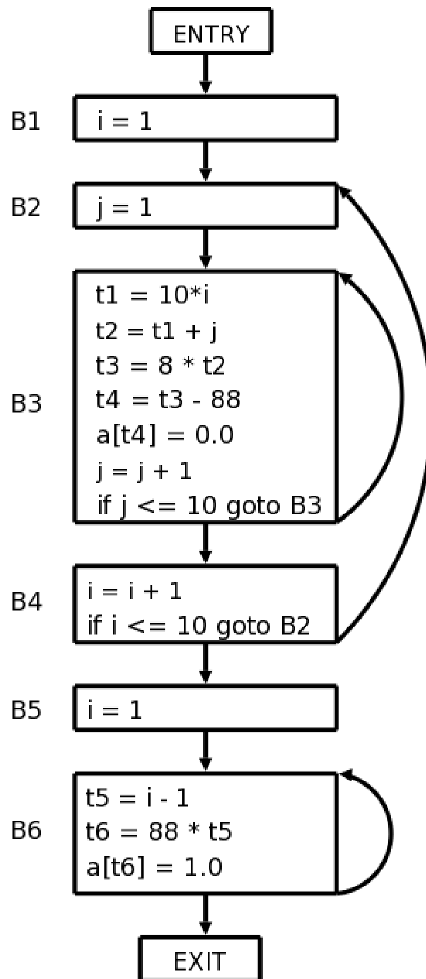
Loops w/ finite bounds
No recursion
Run uninterrupted

Timing Analysis

- Static timing analysis
  - Input: code, arch. model; output: WCET
- Measurement based timing analysis
  - Based on lots of measurements. Statistical.

KU THE UNIVERSITY OF KANSAS

Static Timing Analysis

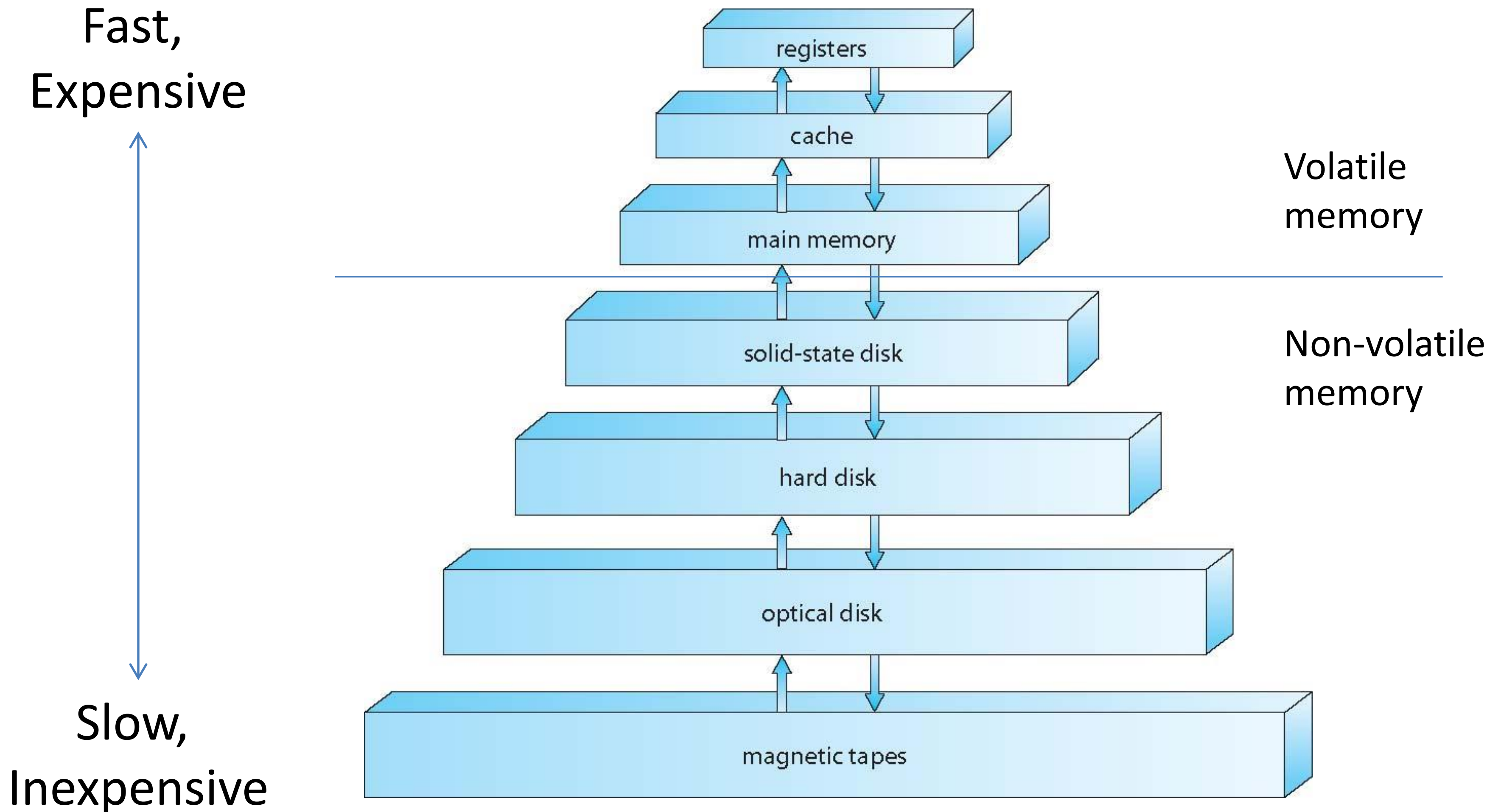


- Analyze code
- Split basic blocks
- Find longest path
  - consider loop bounds
- Compute per-block WCET
  - use abstract CPU model
- Compute task WCET
  - by summing up the WCETs of the longest path

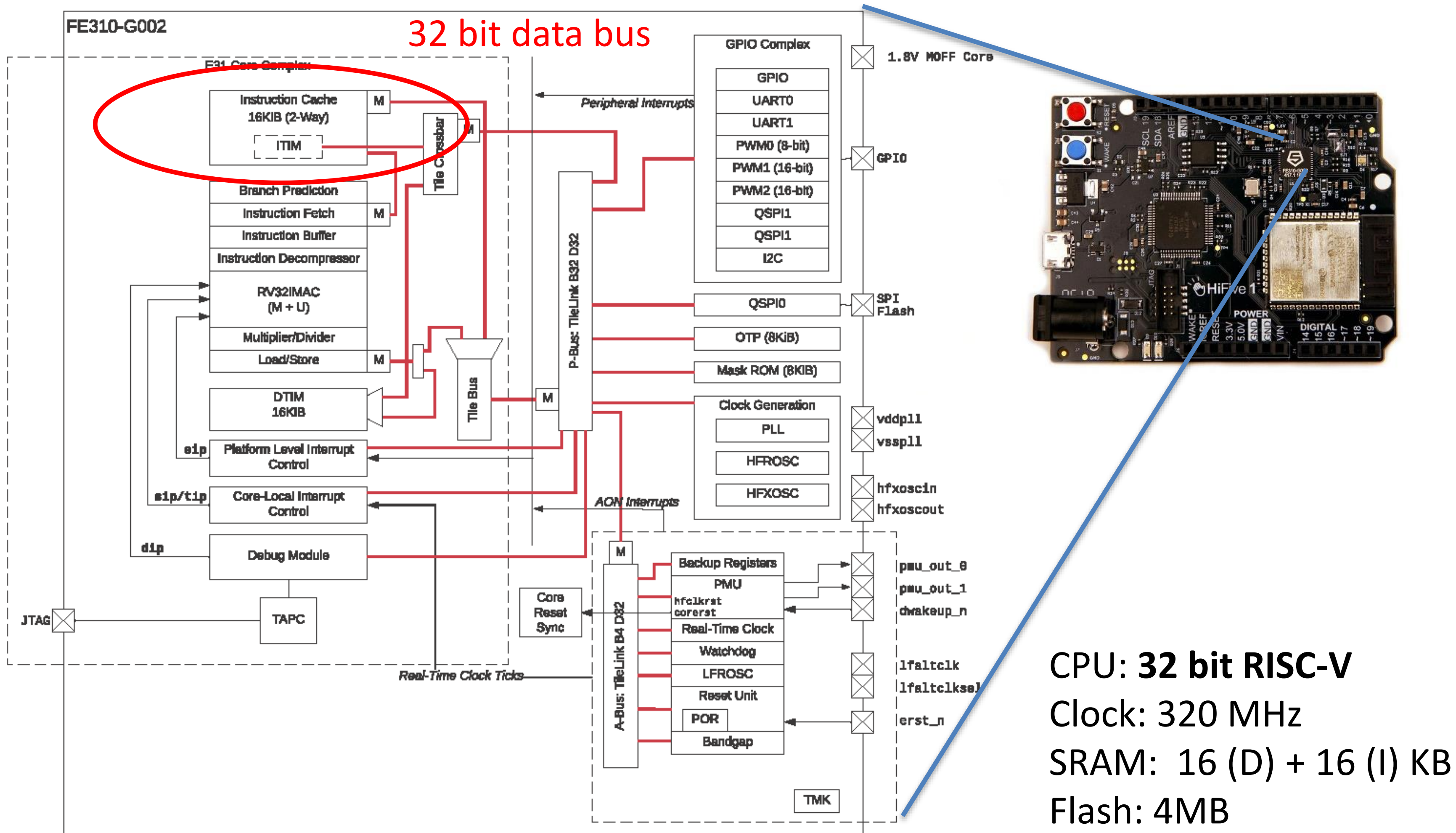
WCET and Caches

- 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

Recall: Memory Hierarchy



SiFive FE310

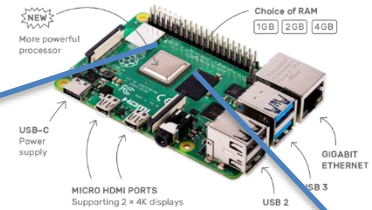


CPU: **32 bit RISC-V**
Clock: 320 MHz
SRAM: 16 (D) + 16 (I) KB
Flash: 4MB

Figure 1: FE310-G002 top-level block diagram.

KU THE UNIVERSITY OF KANSAS

Raspberry Pi 4: Broadcom BCM2711



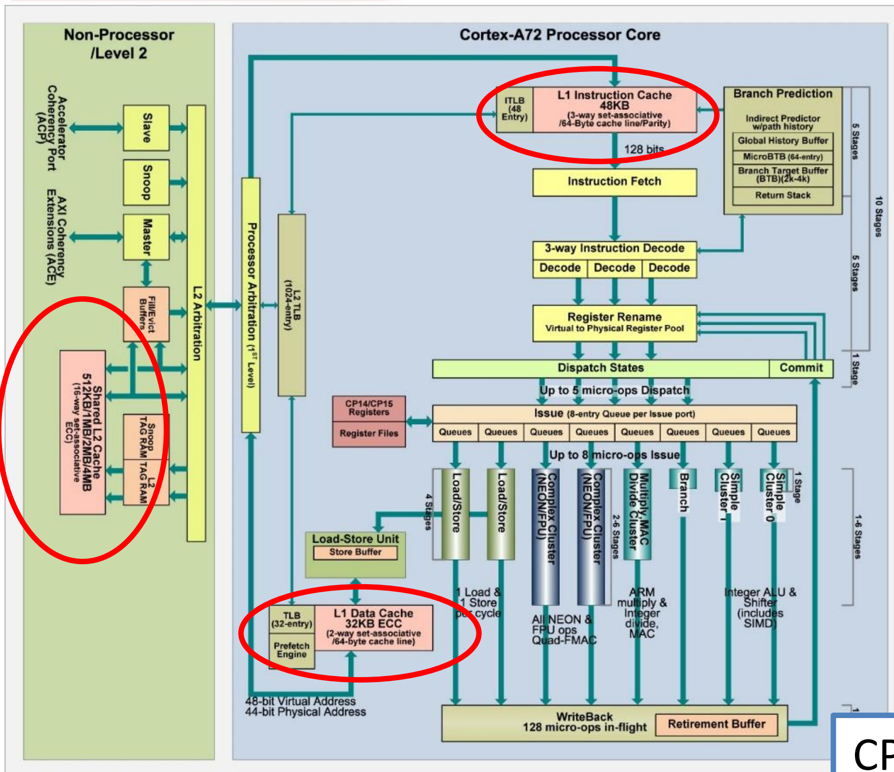
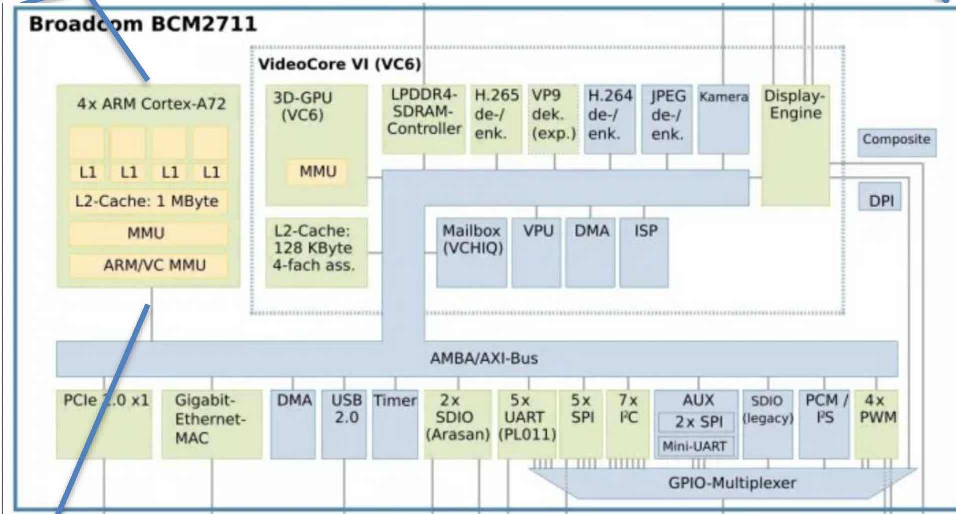


Image source: PC Watch.



(Bild: ct.de/Maik Merten (CC BY SA 4.0))

CPU: 4x **Cortex-A72@**1.5GHz
L2 cache (shared): 1MB
GPU: VideoCore IV@500Mhz
DRAM: 1/2/4 GB LPDDR4-3200
Storage: micro-SD

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:

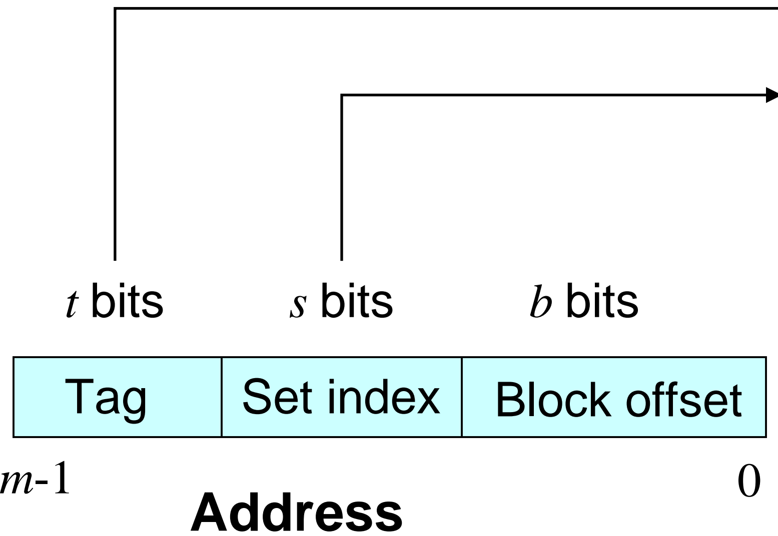
1. 32-bit processor
2. Direct-mapped cache holds two sets
   - 4 floats per set
   - x and y stored contiguously starting at address 0x0

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

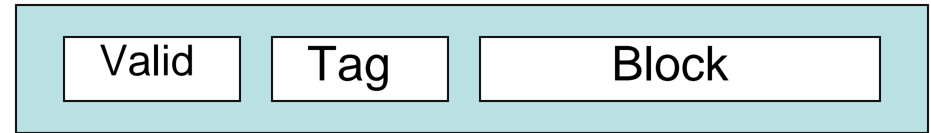
CACHE

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

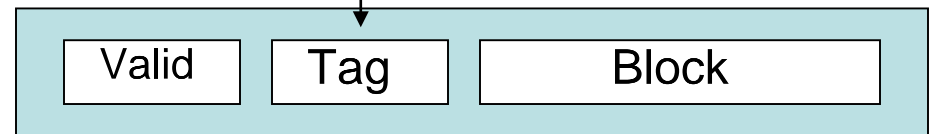
This Particular Direct-Mapped Cache

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

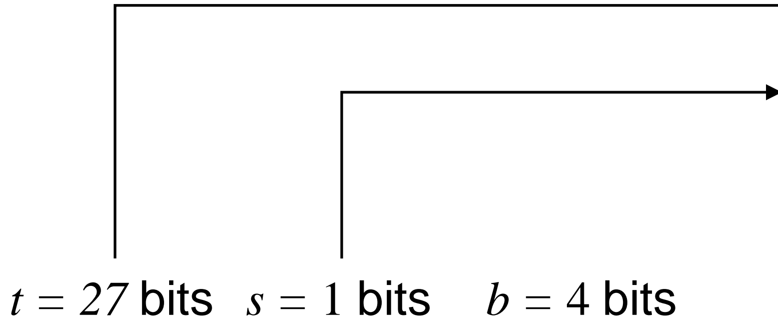
Set 0



Set 1



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



m-1 $\quad$ 0

Address = 32 bits

CACHE

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  }
```

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!

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

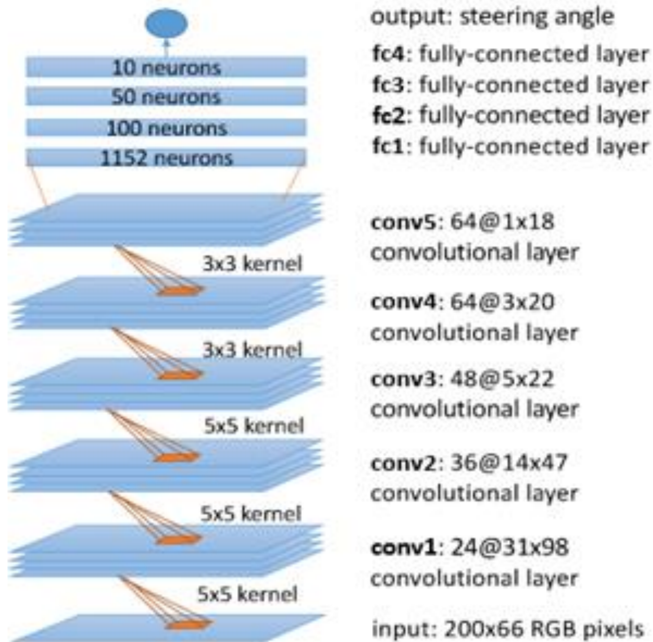
Slide source: Edward A. Lee and Prabal Dutta (UCB)

Measurement Based Timing Analysis

- 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
- No need for detailed architecture models
- 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

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

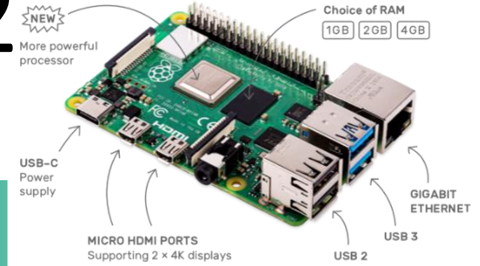
Seventh Attempt (Use BwWrite)

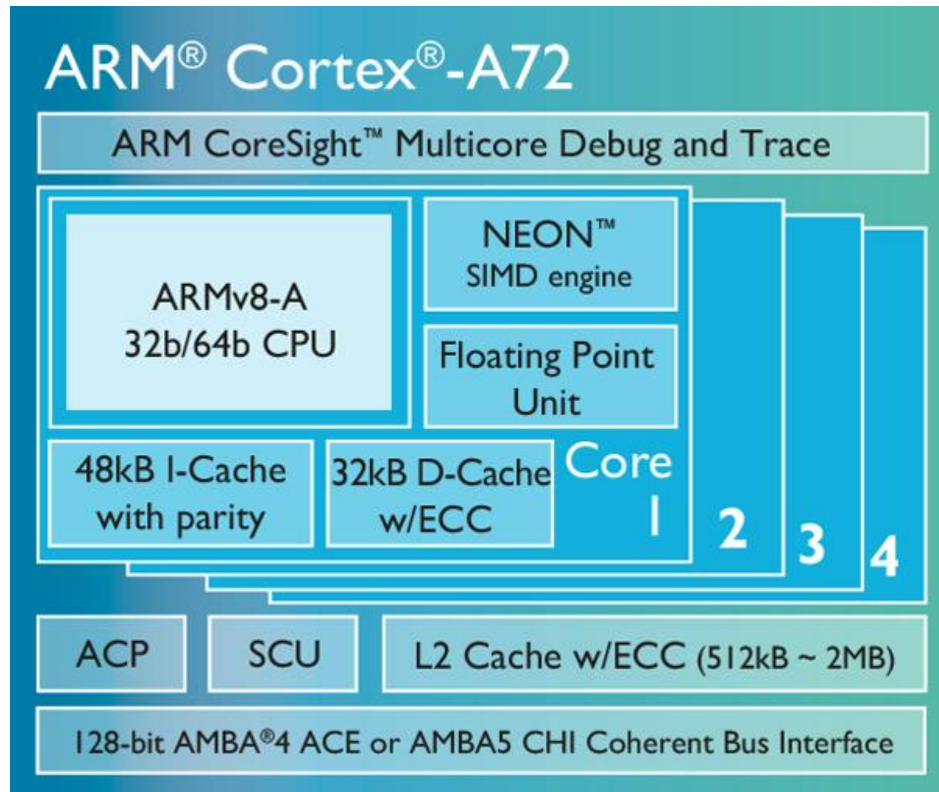
	Solo	w/ BwWrite		
	CFS (nice=0)	CFS (nice=0)	CFS (nice=-5)	FIFO
Mean	21.0	101.2	89.7	92.6
Max	22.4	194.0	137.2	99.7
99pct	21.8	172.4	119.8	97.1
Min	20.7	89.0	71.8	78.7
Median	20.9	93.0	87.5	92.5
Stdev.	0.3	22.8	7.7	1.0

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

KU THE UNIVERSITY OF KANSAS

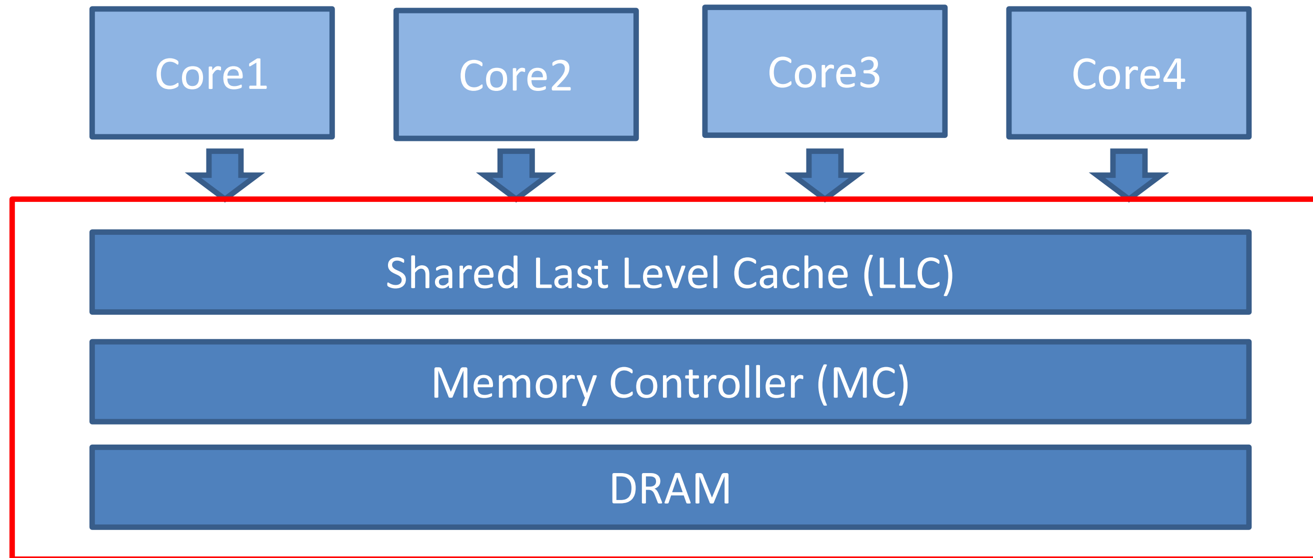
4xARM Cotex-A72





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

Shared Memory Hierarchy

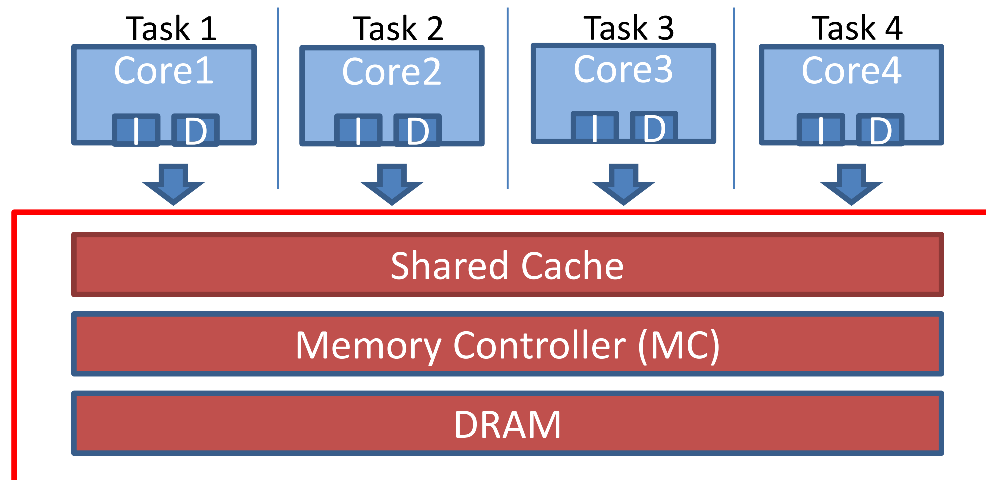


- Cache space
- Memory bus bandwidth
- Memory controller queues
- …

KU THE UNIVERSITY OF KANSAS

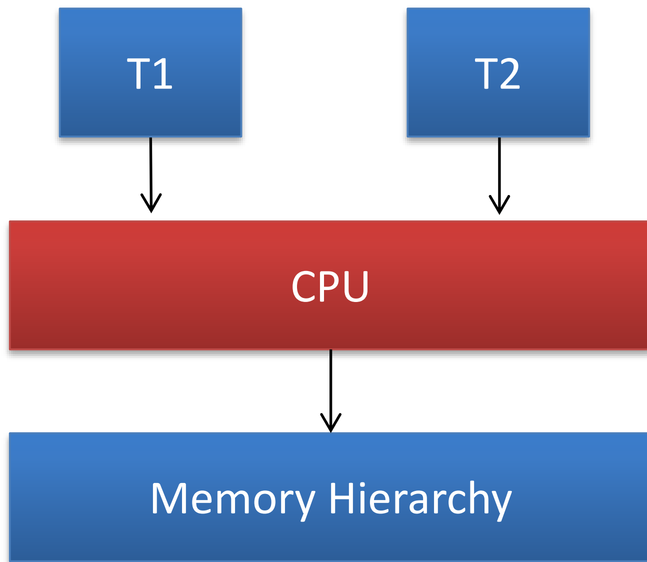
Shared Memory Hierarchy

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

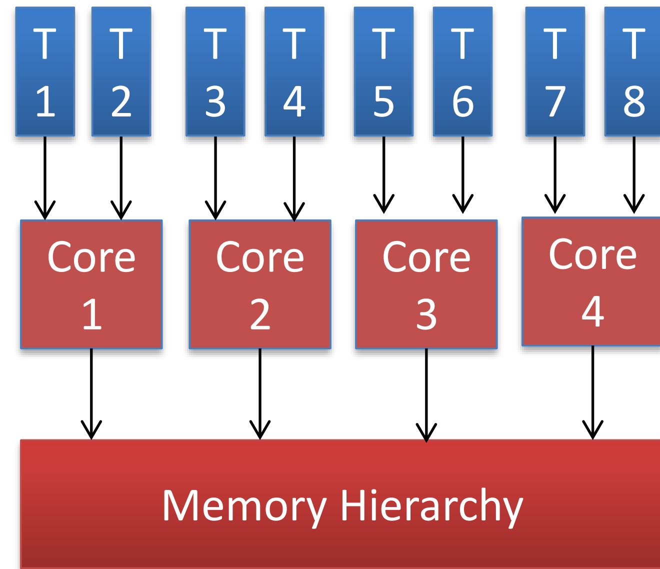


KU THE UNIVERSITY OF KANSAS

Multicore and Memory Hierarchy



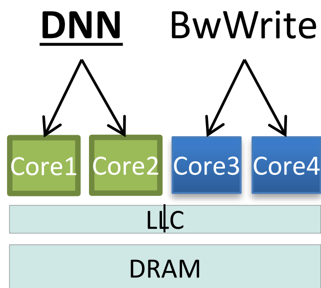
Unicore



Multicore

Performance Impact

Effect of Memory Interference



Normalized Exeuction Time

12
10
8
6
4
2
0

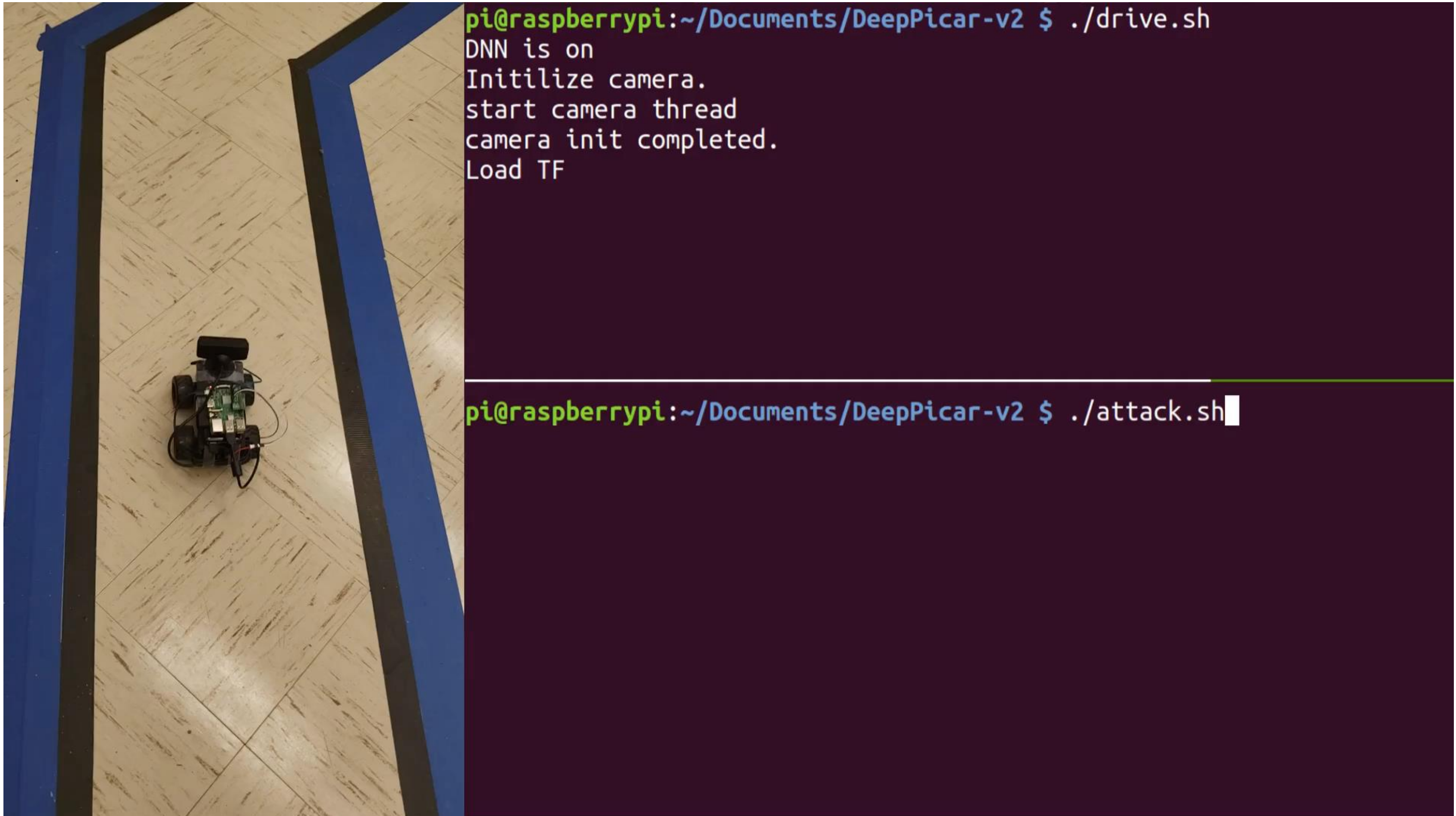
Solo
Corun

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

Effect of Memory Interference



https://youtu.be/Jm6KSDqlqiU

KU THE UNIVERSITY OF KANSAS

Summary

- Timing analysis is important for time sensitive, safety-critical real-time applications
- Static timing analysis

  ++ Strong analytic guarantee

  --- Architecture model is hard and pessimistic
- Measurement based timing analysis

  ++ Practical, no need for architecture model

  --- No guarantee on true worst-case
- Multicore is difficult to handle

KU THE UNIVERSITY OF KANSAS