Analyzable and Practical Real-Time Gang Scheduling on Multicore Using **RT-Gang**

Waqar Ali, Michael Bechtel, Heechul Yun
University of Kansas

KU THE UNIVERSITY OF KANSAS

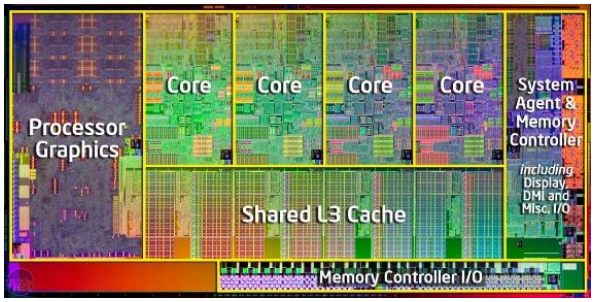
KU-CSL COMPUTER SYSTEMS LAB

Outline

- RT-Gang
- Tutorial
- DeepPicar Case Study

Multicore Processors

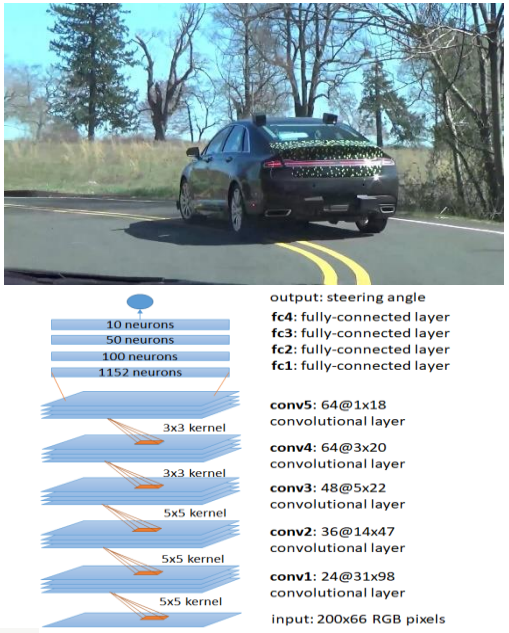
- Provide high computing performance
- Needed for intelligent safety-critical real-time systems



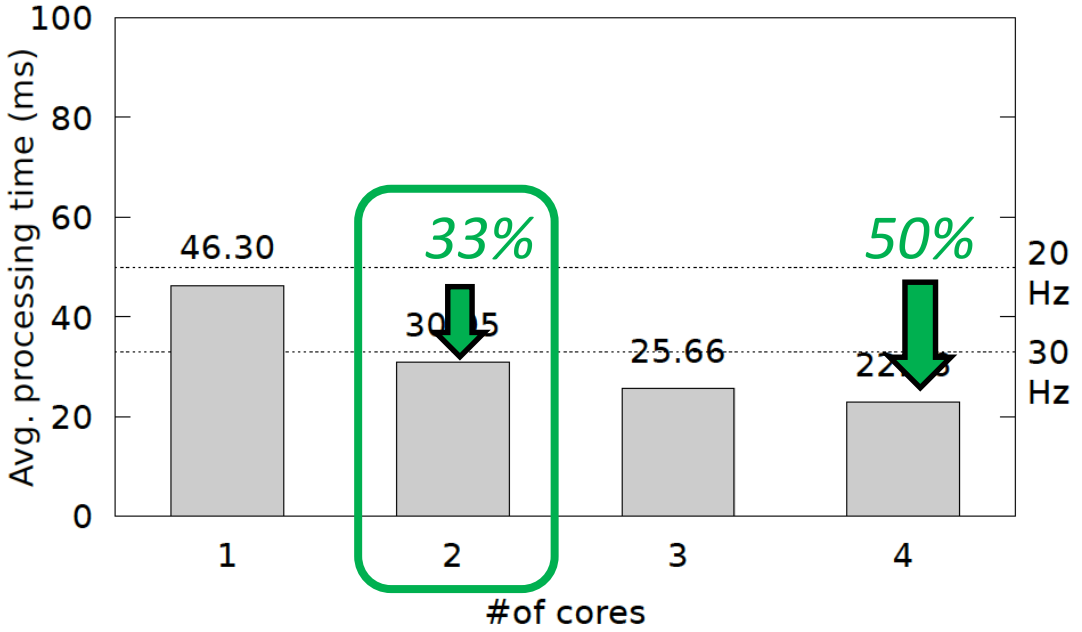
Parallel Real-Time Tasks

- Many emerging workloads in AI, vision, robotics are parallel real-time tasks

*DNN based real-time control **

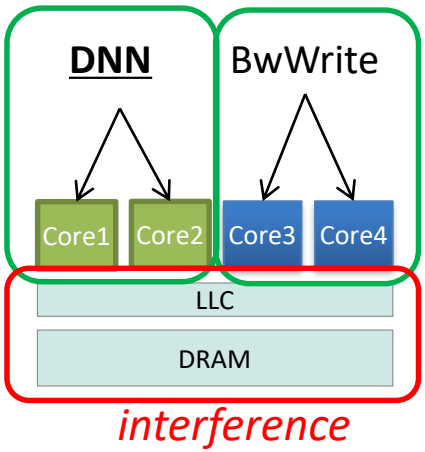


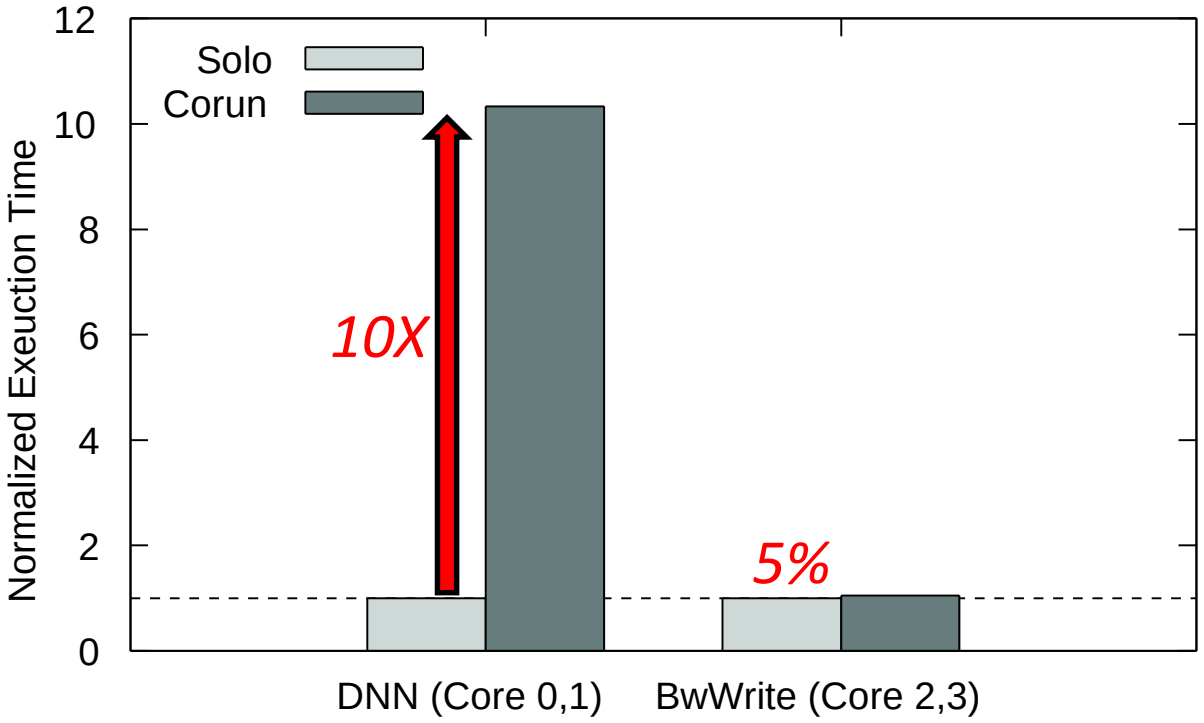
Effect of parallelization on DNN control task



* M. Bojarski, "End to End Learning for Self-Driving Cars." arXiv:1604.07316, 2016

Effect of Co-Scheduling





- DNN control task suffers **>10X slowdown**
  - Due t

It can be worse! (> 300X slowdown)*

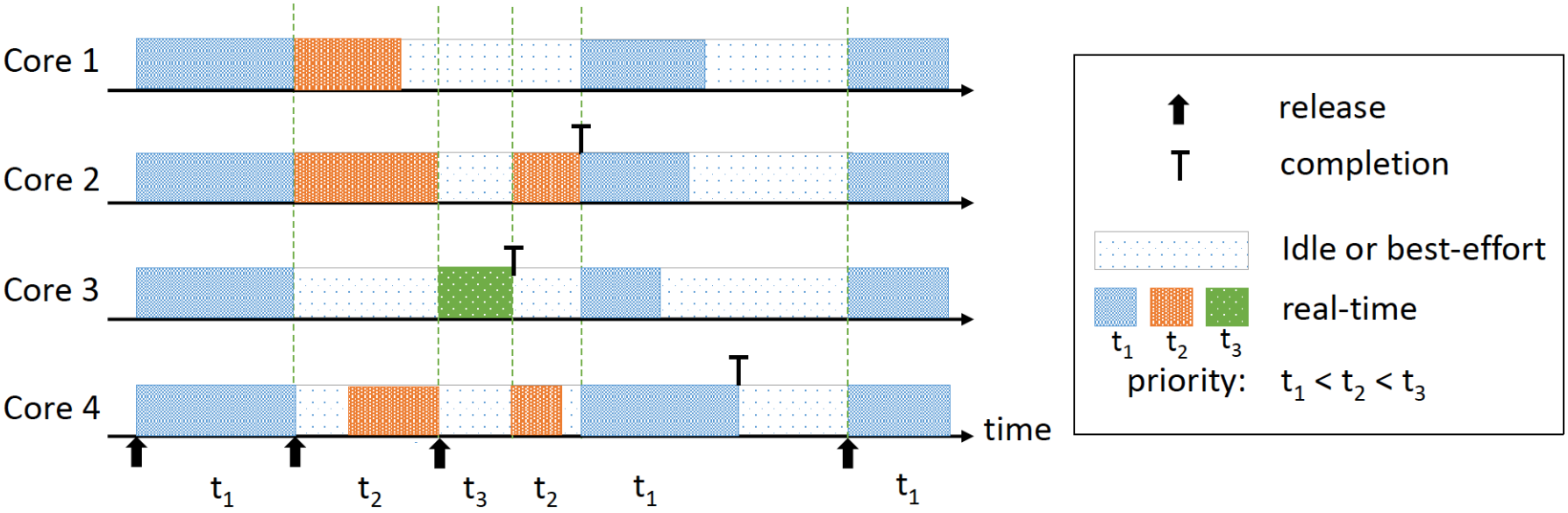
* Michael G. Bechtel and Heechul Yun. "Denial-of-Service Attacks on Shared Cache in Multicore: Analysis and Prevention." In *RTAS*, 2019

Observations

- Interference in shared memory hierarchy
  - Can be very high and unpredictable
  - Depends on the hardware (black box)
- Constructive sharing (Good)
  - Between threads of a single parallel task
- Destructive sharing (Bad)
  - Between threads of different tasks

- **Goal: analyzable and efficient parallel real-time task scheduling framework for multicore**
  - By avoiding destructive sharing

RT-Gang

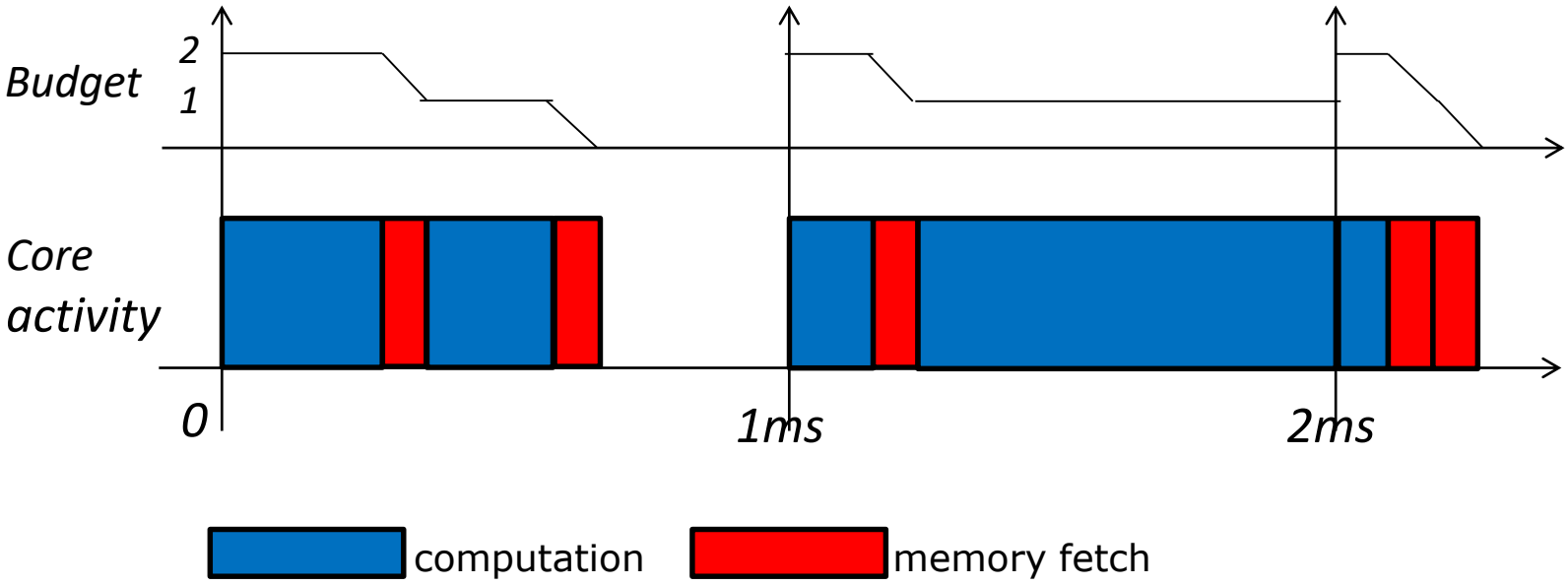


- **One (parallel) real-time task---a gang---at a time**
  - Eliminate inter-task interference by construction
- Schedule best-effort tasks during slacks **w/ throttling**
  - Improve utilization with bounded impacts on the RT tasks

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

Safe Best-Effort Task Throttling

- Throttle the best-effort core(s) if it exceeds a given bandwidth budget **set by the RT task**



Basic throttling mechanism *

* Yun et al., "MemGuard: Memory Bandwidth Reservation System for Efficient Performance Isolation in Multi-core Platforms." In *RTAS*, 2013

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

Implementation

- Modified Linux's RT scheduler
  - Implemented as a "feature" of SCHED_FIFO (sched/rt.c)

- Best-effort task throttling
  - A separate kernel module based on BWLOCK++ *

KU THE UNIVERSITY OF KANSAS

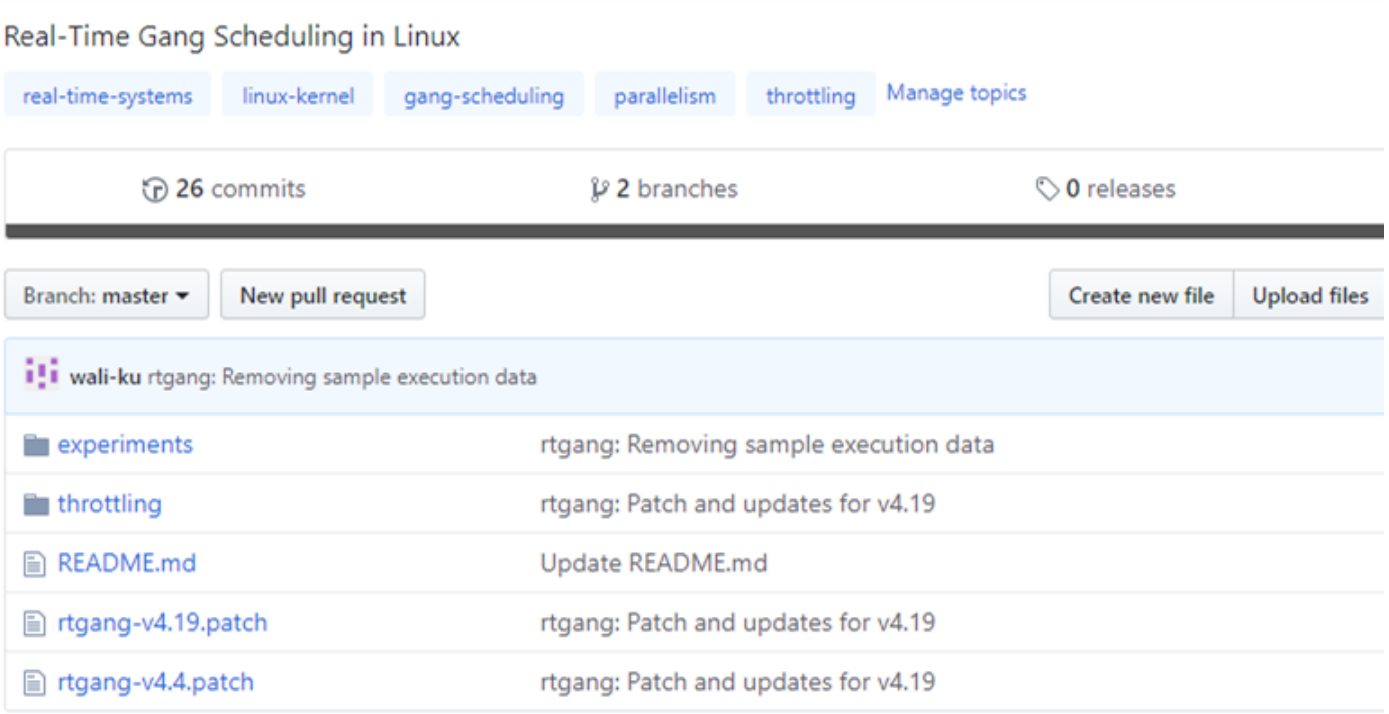
* W. Ali and H. Yun., "Protecting Real-Time GPU Kernels on Integrated CPU-GPU SoC Platforms." In *ECRTS*, 2018

Outline

- RT-Gang
- **Tutorial**
- DeepPicar Case Study

Source Code Repository

- git clone https://github.com/CSL-KU/RT-Gang



Installation

- From the Linux kernel directory:
  - patch -p1 < ../RT-Gang/rtgang-v4.19.patch
  - Compile & install & restart
- To check if installed correctly:
  - sudo cat /sys/kernel/debug/sched_features | grep RT_GANG_LOCK

```
pi@raspberrypi:~ $ sudo cat /sys/kernel/debug/sched_features | grep RT_GANG_LOCK
GENTLE_FAIR_SLEEPERS NO_RT_GANG_LOCK START_DEBIT NO_NEXT_BUDDY LAST_BUDDY CACHE_HOT
_BUDDY WAKEUP_PREEMPTION NO_HRTICK NO_DOUBLE_TICK LB_BIAS NONTASK_CAPACITY TTWU_QUE
UE NO_SIS_AVG_CPU SIS_PROP NO_WARN_DOUBLE_CLOCK RT_PUSH_IPI RT_RUNTIME_SHARE NO_LB_
MIN ATTACH_AGE_LOAD WA_IDLE WA_WEIGHT WA_BIAS UTIL_EST
```

Enable/Disable RT-Gang

- RT-Gang is enabled/disabled through the kernel's scheduling feature

```
# Enable RT-Gang
echo RT_GANG_LOCK >> /sys/kernel/debug/sched_features

# Disable RT-Gang
echo NO_RT_GANG_LOCK >> /sys/kernel/debug/sched_features
```

Best-Effort Task Throttling

- Throttling is enabled through a kernel module
  - cd RT-Gang/throttling/kernel_module
  - make
  - sudo insmod exe/bwlockmod.ko

Best-Effort Task Throttling

- Only occurs when a real-time task is running
  - W/o real-time task

```
pi@raspberrypi:~ $ bandwidth
memsize=4096 KB, type=read, cpuid=0
stop at 5
g_nread(bytes read) = 11228151808
elapsed = 5.00 sec ( 5000001 usec )
CPU0: B/W = 2141.60 MB/s | CPU0: average = 28.50 ns
```

  - W/ real-time task

```
pi@raspberrypi:~ $ sudo chrt -f 1 bandwidth -t 0 &> /dev/null &
[1] 1222
pi@raspberrypi:~ $ bandwidth
memsize=4096 KB, type=read, cpuid=0
stop at 5
g_nread(bytes read) = 524288000
elapsed = 5.00 sec ( 5000394 usec )
CPU0: B/W = 99.99 MB/s | CPU0: average = 610.40 ns
```

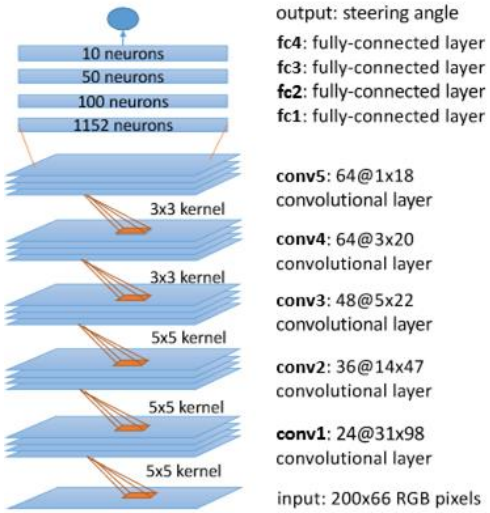
Outline

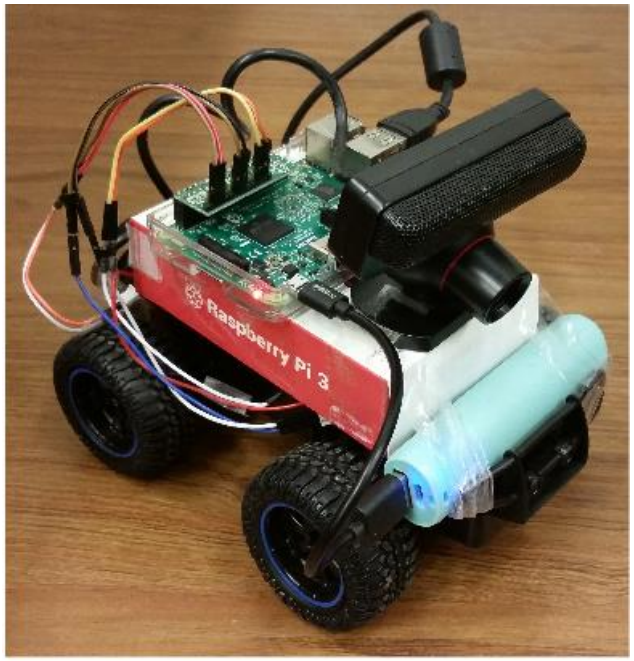
- RT-Gang
- Tutorial
- **DeepPicar Case Study**

DeepPicar

- A **low cost**, small scale replication of NVIDIA's DAVE-2
- **Uses the exact same DNN**
- Runs on a Raspberry Pi 3 in **real-time**

Item	Cost ($)
Raspberry Pi 3 Model B	35
New Bright 1:24 scale RC car	10
Playstation Eye camera	7
Pololu DRV8835 motor hat	8
External battery pack & misc.	10
Total	70





KU THE UNIVERSITY OF KANSAS

* Bechtel et al. DeepPicar: A Low-cost Deep Neural Network-based Autonomous Car. In *RTCSA*, 2018

https://github.com/mbechtel2/DeepPicar-v2

DNN based Real-Time Control

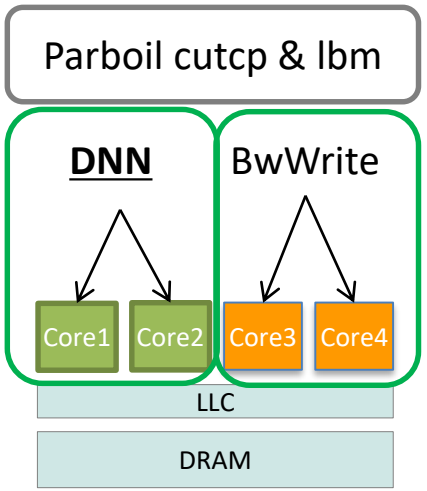
```
while True:
  # 1. read from the forward camera
  frame = camera.read()
  # 2. convert to 200x66 rgb pixels
  frame = preprocess(frame)
  # 3. perform inferencing operation
  angle = DNN_inferencing(frame)
  # 4. motor control
  steering_motor_control(angle)
  # 5. wait till next period begins
  wait_till_next_period()
```

- DNN Inferencing is the most compute intensive part.
- Parallelized by TensorFlow to utilize multiple cores.

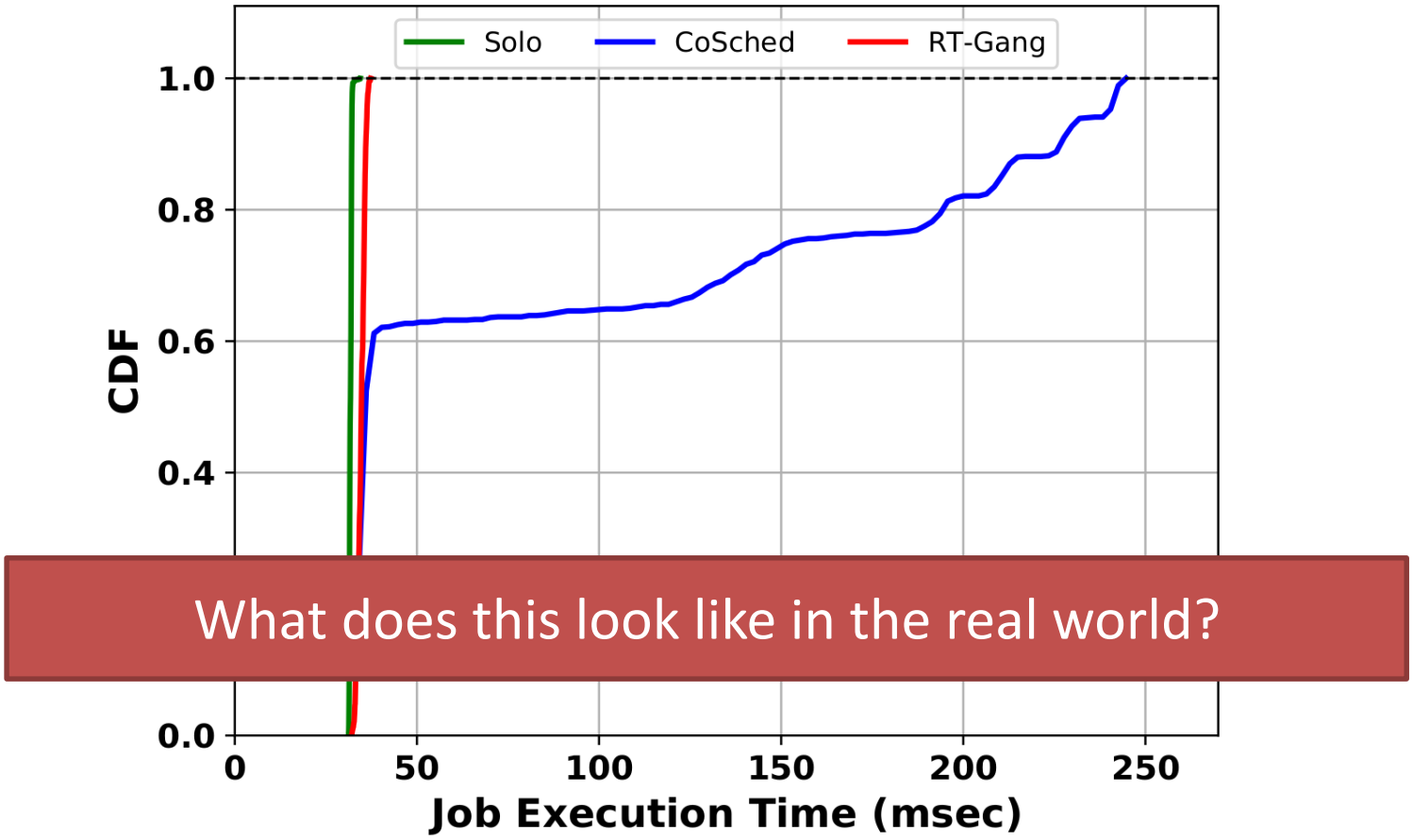
Experiment Setup

- 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



Execution Time Distribution

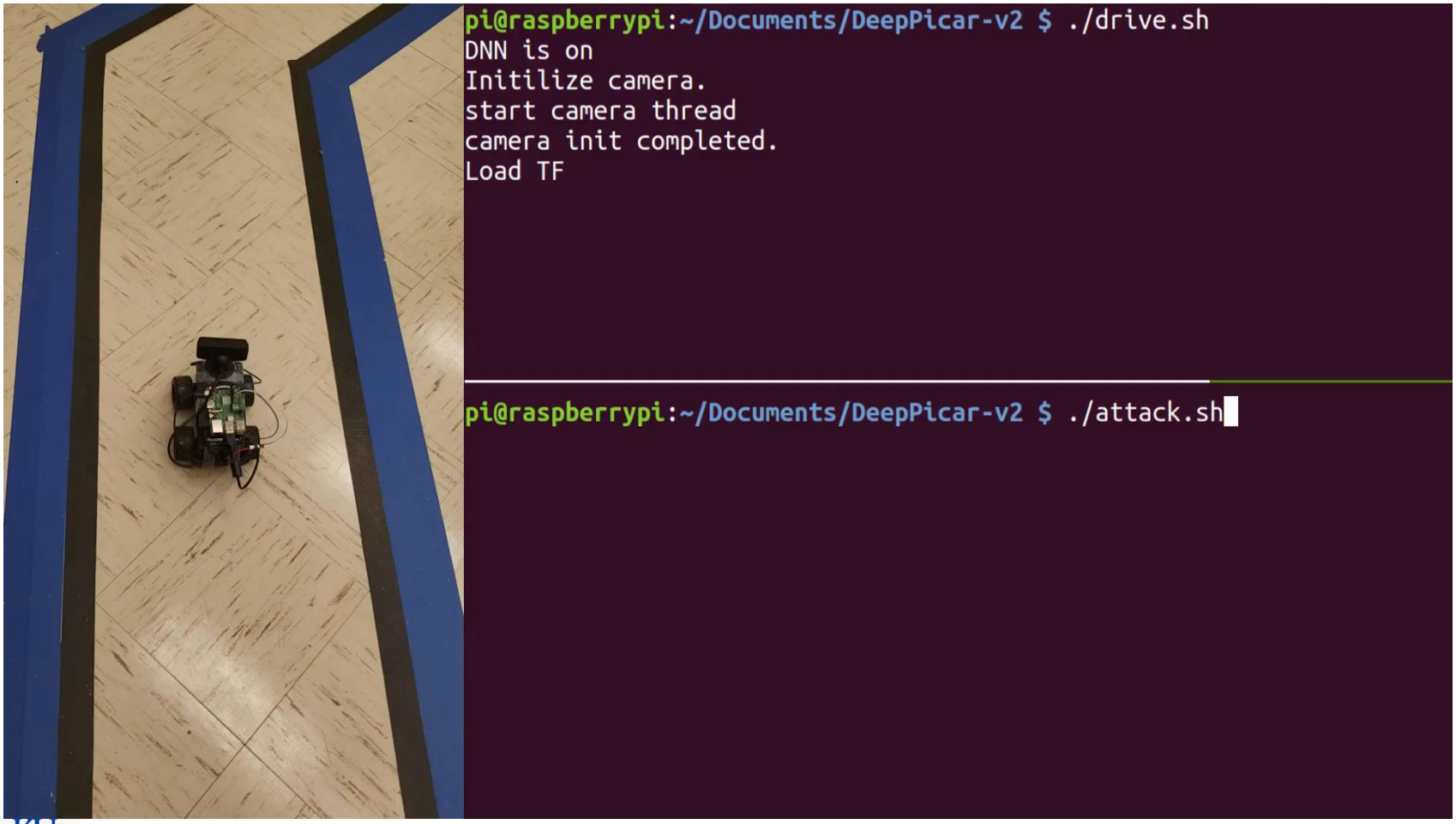


What does this look like in the real world?

- RT-Gang achieves deterministic timing

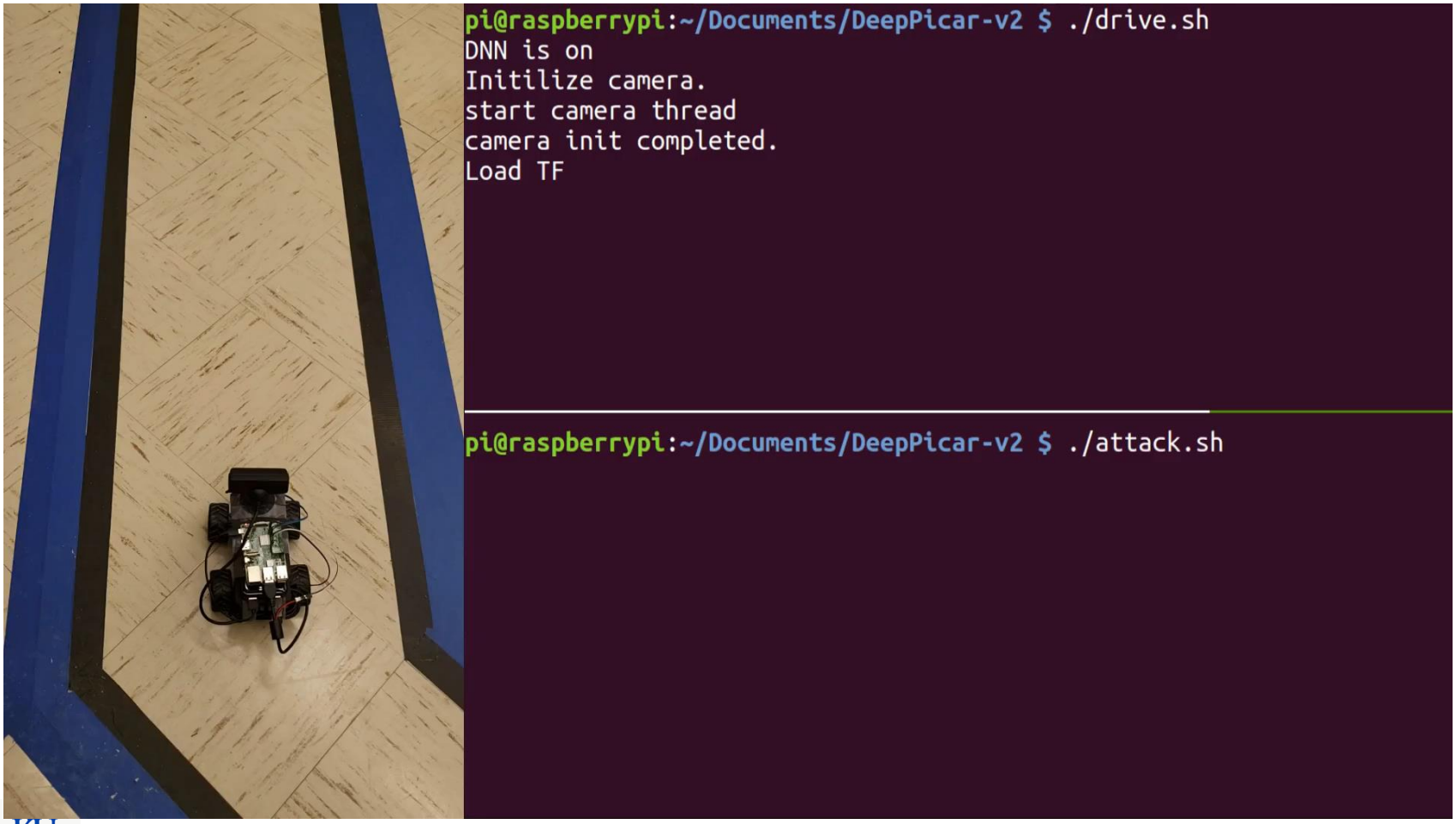
KU THE UNIVERSITY OF KANSAS

CoSched (w/o RT-Gang)



https://youtu.be/Jm6KSDqlqiU

RT-Gang



https://youtu.be/pk0j063cUAs

Conclusion

- Parallel real-time task scheduling
  - Hard to analyze on COTS multicore
  - Due to interference in shared memory hierarchy
- RT-Gang
  - **Analyzable** and **efficient** parallel real-time gang scheduling framework, implemented in Linux
  - Avoid interference by construction
    - Can protect critical real-time tasks

https://github.com/CSL-KU/rt-gang

KU THE UNIVERSITY OF KANSAS

Thank You!

Disclaimer:

This research is supported by NSF CNS 1718880, CNS 1815959, and NSA Science of Security initiative contract #H98230-18-D-0009.

KU THE UNIVERSITY OF KANSAS