Red Hat
Enterprise Linux

Sant'Anna
School of Advanced Studies – Pisa

UNIVERSIDADE FEDERAL
DE SANTA CATARINA

Demystifying the Real-Time Linux Scheduling Latency

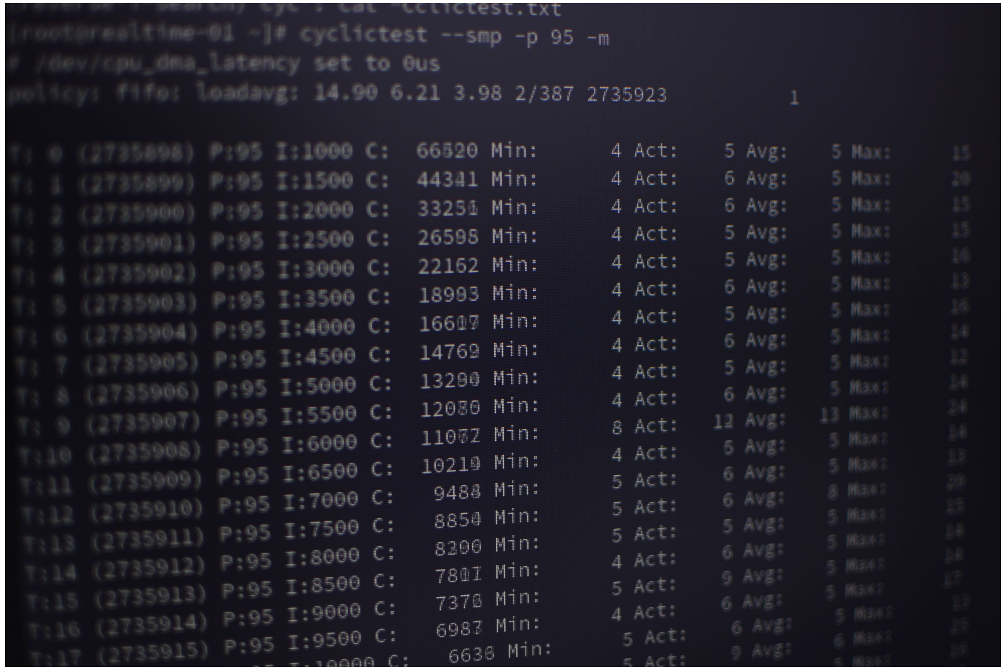
Daniel Bristot de Oliveira, Daniel Casini, Rômulo Silva de Oliveira and Tommaso Cucinotta
Principal Software Engineer

Real-Time Linux

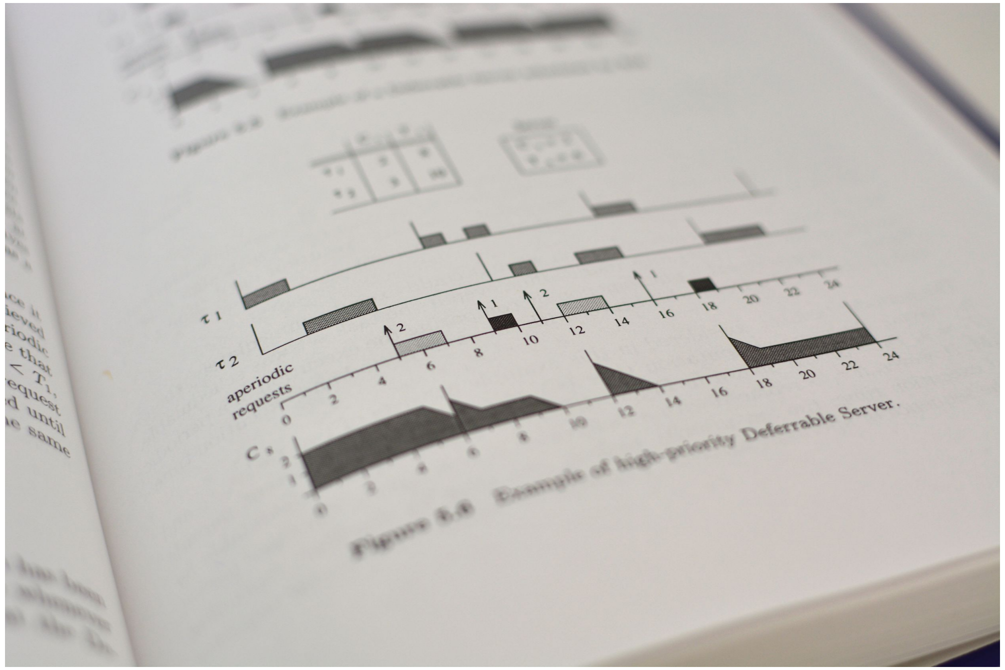
“Real-Time” Linux

Real-Time Linux vs Real-Time theory

Experimental vs Analytical

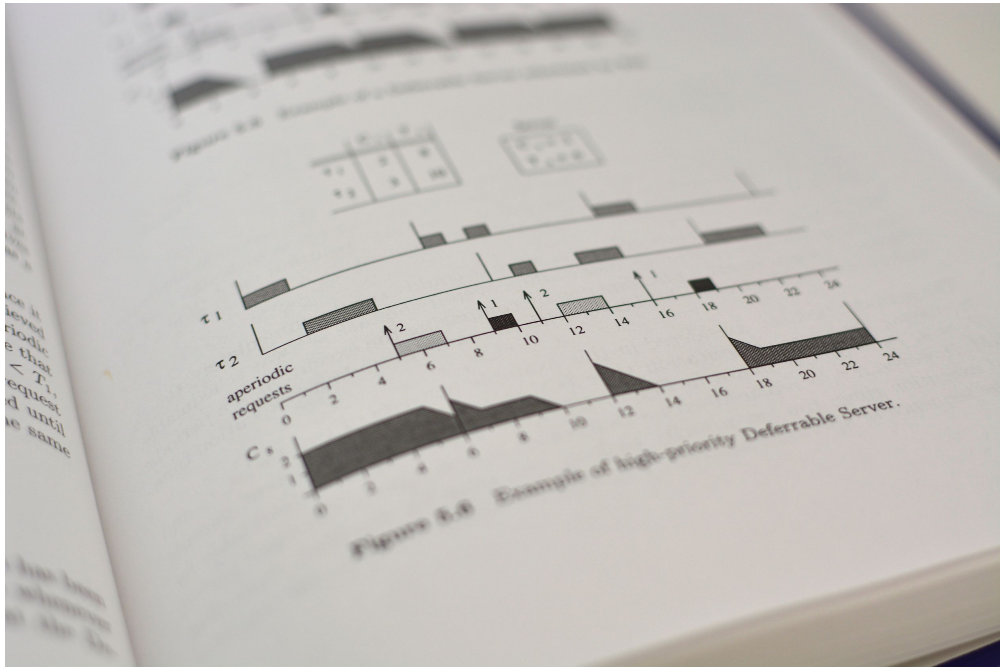


```
[root@realtime-01 ~]# cyclictest --smp -p 95 -m
# /dev/cpu_dma_latency set to 0us
policy: fifo: loadavg: 14.90 6.21 3.98 2/387 2735923          1

T: 0 (2735898) P:95 I:1000 C:  66520 Min:      4 Act:    5 Avg:    5 Max:      15
T: 1 (2735899) P:95 I:1500 C:  44341 Min:      4 Act:    6 Avg:    5 Max:      20
T: 2 (2735900) P:95 I:2000 C:  33251 Min:      4 Act:    6 Avg:    5 Max:      15
T: 3 (2735901) P:95 I:2500 C:  26598 Min:      4 Act:    5 Avg:    5 Max:      15
T: 4 (2735902) P:95 I:3000 C:  22162 Min:      4 Act:    5 Avg:    5 Max:      16
T: 5 (2735903) P:95 I:3500 C:  18993 Min:      4 Act:    6 Avg:    5 Max:      13
T: 6 (2735904) P:95 I:4000 C:  16617 Min:      4 Act:    5 Avg:    5 Max:      16
T: 7 (2735905) P:95 I:4500 C:  14769 Min:      4 Act:    6 Avg:    5 Max:      14
T: 8 (2735906) P:95 I:5000 C:  13290 Min:      4 Act:    5 Avg:    5 Max:      12
T: 9 (2735907) P:95 I:5500 C:  12080 Min:      4 Act:    6 Avg:    5 Max:      14
T:10 (2735908) P:95 I:6000 C:  11072 Min:      8 Act:   12 Avg:   13 Max:      24
T:11 (2735909) P:95 I:6500 C:  10219 Min:      4 Act:    5 Avg:    5 Max:      14
T:12 (2735910) P:95 I:7000 C:   9488 Min:      5 Act:    6 Avg:    5 Max:      13
T:13 (2735911) P:95 I:7500 C:   8854 Min:      5 Act:    6 Avg:    8 Max:      20
T:14 (2735912) P:95 I:8000 C:   8300 Min:      5 Act:    5 Avg:    5 Max:      15
T:15 (2735913) P:95 I:8500 C:   7801 Min:      4 Act:    6 Avg:    5 Max:      14
T:16 (2735914) P:95 I:9000 C:   7378 Min:      5 Act:    9 Avg:    5 Max:      14
T:17 (2735915) P:95 I:9500 C:   6983 Min:      4 Act:    6 Avg:    5 Max:      17
```



Real-Time Linux vs Real-Time theory

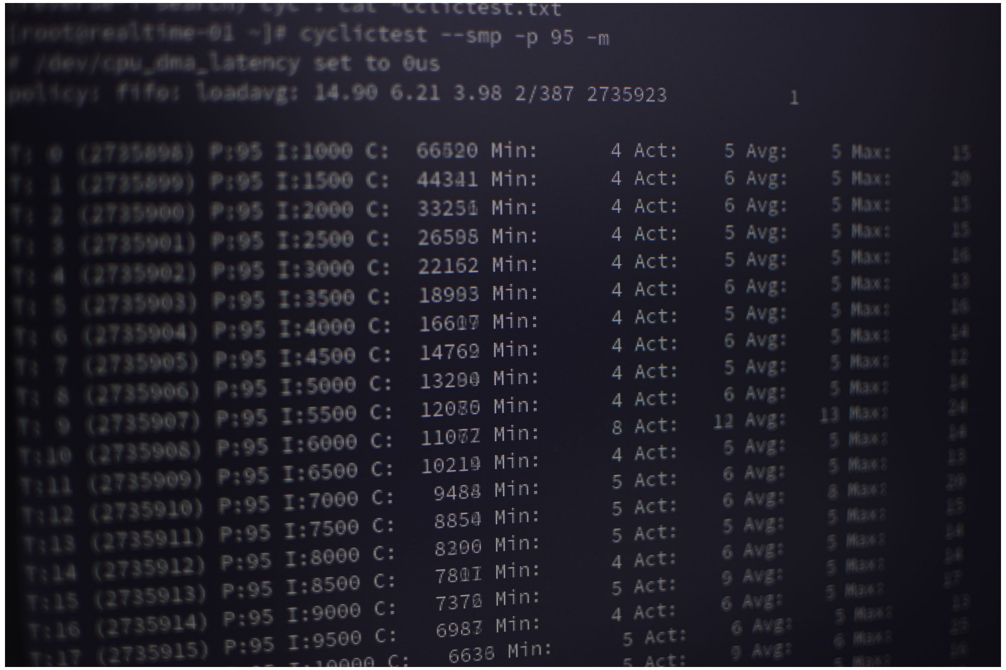
Real-time analysis



- Based on the timing description of the system
- Capture all behaviors
- Precisely define the worst cases
- But depends on a precise definition of the system

Real-Time Linux vs Real-Time theory

Linux approach



```
[root@realtime-01 ~]# cyclictest --smp -p 95 -m
# /dev/cpu_dma_latency set to 0us
policy: fifo: loadavg: 14.90 6.21 3.98 2/387 2735923          1

T: 0 (2735898) P:95 I:1000 C:  66520 Min:      4 Act:    5 Avg:    5 Max:      15
T: 1 (2735899) P:95 I:1500 C:  44341 Min:      4 Act:    6 Avg:    5 Max:      20
T: 2 (2735900) P:95 I:2000 C:  33251 Min:      4 Act:    6 Avg:    5 Max:      15
T: 3 (2735901) P:95 I:2500 C:  26598 Min:      4 Act:    5 Avg:    5 Max:      15
T: 4 (2735902) P:95 I:3000 C:  22162 Min:      4 Act:    5 Avg:    5 Max:      16
T: 5 (2735903) P:95 I:3500 C:  18993 Min:      4 Act:    6 Avg:    5 Max:      13
T: 6 (2735904) P:95 I:4000 C:  16617 Min:      4 Act:    5 Avg:    5 Max:      16
T: 7 (2735905) P:95 I:4500 C:  14769 Min:      4 Act:    6 Avg:    5 Max:      14
T: 8 (2735906) P:95 I:5000 C:  13280 Min:      4 Act:    5 Avg:    5 Max:      12
T: 9 (2735907) P:95 I:5500 C:  12080 Min:      4 Act:    6 Avg:    5 Max:      14
T:10 (2735908) P:95 I:6000 C:  11072 Min:      8 Act:   12 Avg:   13 Max:      24
T:11 (2735909) P:95 I:6500 C:  10219 Min:      4 Act:    5 Avg:    5 Max:      14
T:12 (2735910) P:95 I:7000 C:   9488 Min:      5 Act:    6 Avg:    5 Max:      13
T:13 (2735911) P:95 I:7500 C:   8850 Min:      5 Act:    6 Avg:    8 Max:      20
T:14 (2735912) P:95 I:8000 C:   8200 Min:      5 Act:    5 Avg:    5 Max:      15
T:15 (2735913) P:95 I:8500 C:   7801 Min:      4 Act:    6 Avg:    5 Max:      14
T:16 (2735914) P:95 I:9000 C:   7370 Min:      5 Act:    9 Avg:    5 Max:      14
T:17 (2735915) P:95 I:9500 C:   6983 Min:      4 Act:    6 Avg:    5 Max:      17
                    I:10000 C:   6638 Min:      5 Act:    6 Avg:    5 Max:      13
```

- Linux was adapted to become a RTOS
- PREEMPT_RT: *De facto* standard
- Evaluated (mainly) with cyclictest
- Cyclictest:
  - Practical: lightweight and out-of-the-box
  - It is a “black-box” test
  - No demonstration
  - Does not provide evidence of “root-cause”

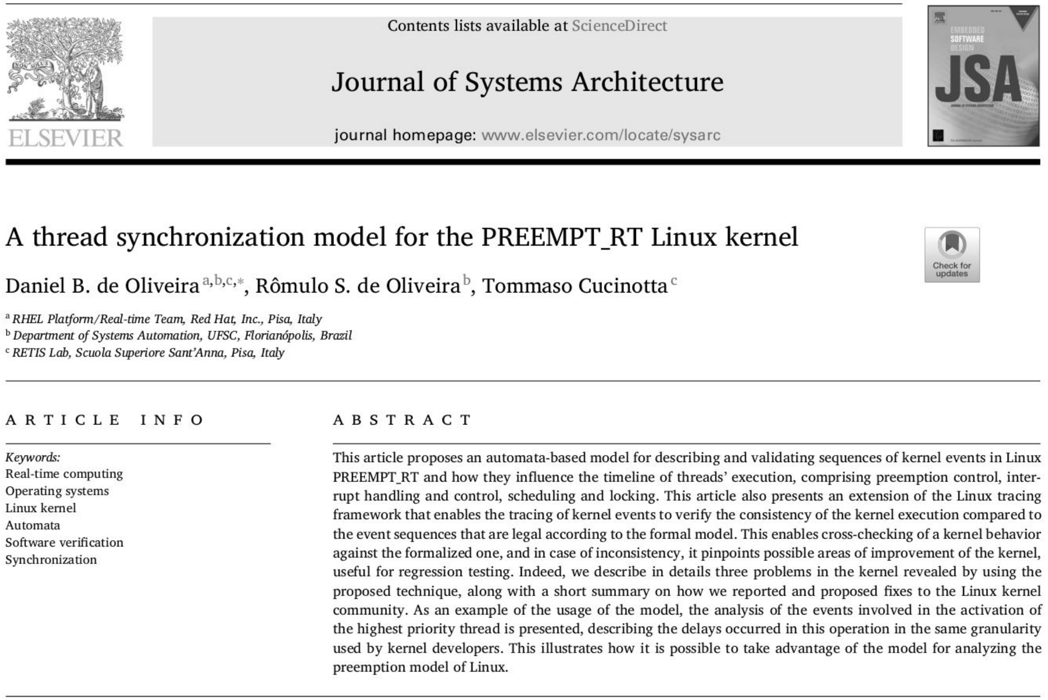
Why don't we apply RT analysis on Linux?

Linux is complex

```
bristot@x1:~/src/git/linux-rt-devel
*/
tatic void __sched notrace __schedule(bool preempt)

        struct task_struct *prev, *next;
        unsigned long *switch_count;
        struct rq_flags rf;
        struct rq *rq;
        int cpu;

        cpu = smp_processor_id();
        rq = cpu_rq(cpu);
        prev = rq->curr;

        schedule_debug(prev, preempt);

        if (sched_feat(HRTICK))
                hrtick_clear(rq);

        local_irq_disable();
        rcu_note_context_switch(preempt);

        /*
         * Make sure that signal_pending_state()
         * can't be reordered with __set_current_state
```

- Lots of contexts
- Lots of hacks
- Lots of information
- Fast pacing
- …

The PREEMPT_RT thread model



Contents lists available at ScienceDirect

Journal of Systems Architecture

journal homepage: www.elsevier.com/locate/sysarc

A thread synchronization model for the PREEMPT_RT Linux kernel

Daniel B. de Oliveira [a,b,c,*], Rômulo S. de Oliveira [b], Tommaso Cucinotta [c]

[a] RHEL Platform/Real-time Team, Red Hat, Inc., Pisa, Italy
[b] Department of Systems Automation, UFSC, Florianópolis, Brazil
[c] RETIS Lab, Scuola Superiore Sant'Anna, Pisa, Italy

ARTICLE INFO

Keywords:
Real-time computing
Operating systems
Linux kernel
Automata
Software verification
Synchronization

ABSTRACT

This article proposes an automata-based model for describing and validating sequences of kernel events in Linux PREEMPT_RT and how they influence the timeline of threads' execution, comprising preemption control, interrupt handling and control, scheduling and locking. This article also presents an extension of the Linux tracing framework that enables the tracing of kernel events to verify the consistency of the kernel execution compared to the event sequences that are legal according to the formal model. This enables cross-checking of a kernel behavior against the formalized one, and in case of inconsistency, it pinpoints possible areas of improvement of the kernel, useful for regression testing. Indeed, we describe in details three problems in the kernel revealed by using the proposed technique, along with a short summary on how we reported and proposed fixes to the Linux kernel community. As an example of the usage of the model, the analysis of the events involved in the activation of the highest priority thread is presented, describing the delays occurred in this operation in the same granularity used by kernel developers. This illustrates how it is possible to take advantage of the model for analyzing the preemption model of Linux.

It defines the *specifications* of threads synchronization:

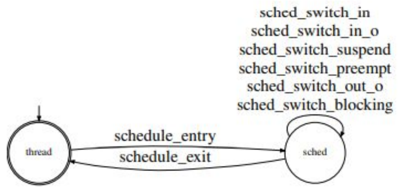


Figure 22: *S08* Switch while scheduling.

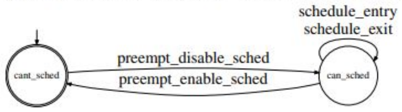


Figure 23: *S03* Scheduler called with preemption disabled.

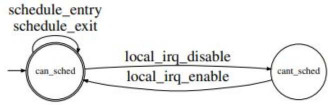


Figure 20: *S05* Scheduler called with interrupts enabled.

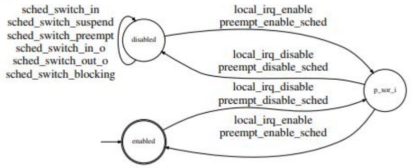
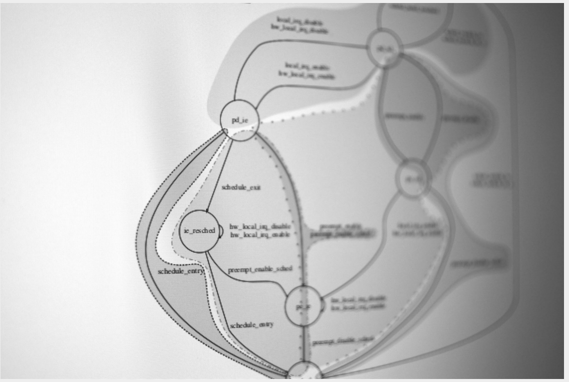


Figure 21: *S07* Switch with interrupts and preempt disabled.

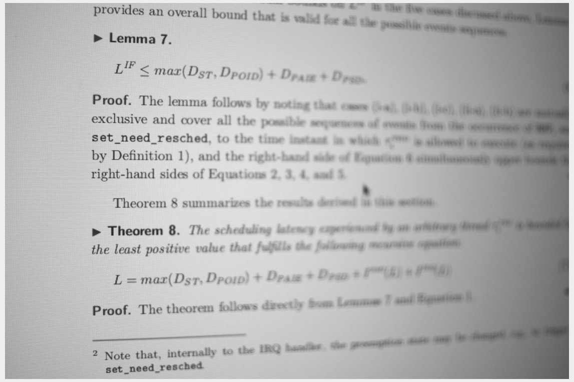
Demystifying the Real-Time Linux Scheduling Latency

Approach

Formal specification



Scheduling latency bound



provides an overall bound that is

▶ Lemma 7.

$$L^{IF} \leq max(D_{ST}, D_{POID}) + D_{PAIE} + D_{PSD}.$$

Proof. The lemma follows by noting that
exclusive and cover all the
`set_need_resched`, to the time
by Definition 1), and the right-hand
right-hand sides of Equations 2, 3, 4, and 5.

Theorem 8 summarizes the

▶ Theorem 8. *The scheduling latency*
the least positive value that

$$L = max(D_{ST}, D_{POID}) + D_{PAIE}$$

Proof. The theorem follows directly

---

[2] Note that, internally to the IRQ
`set_need_resched`.

Measurement and analysis

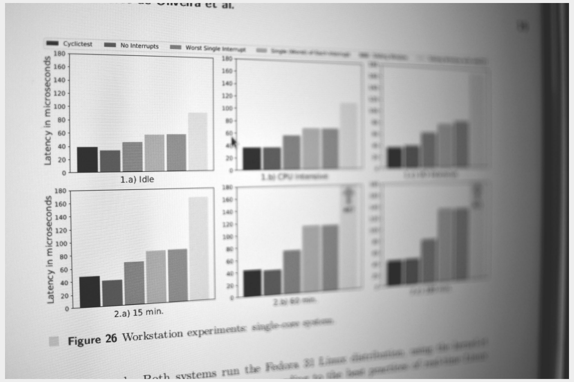
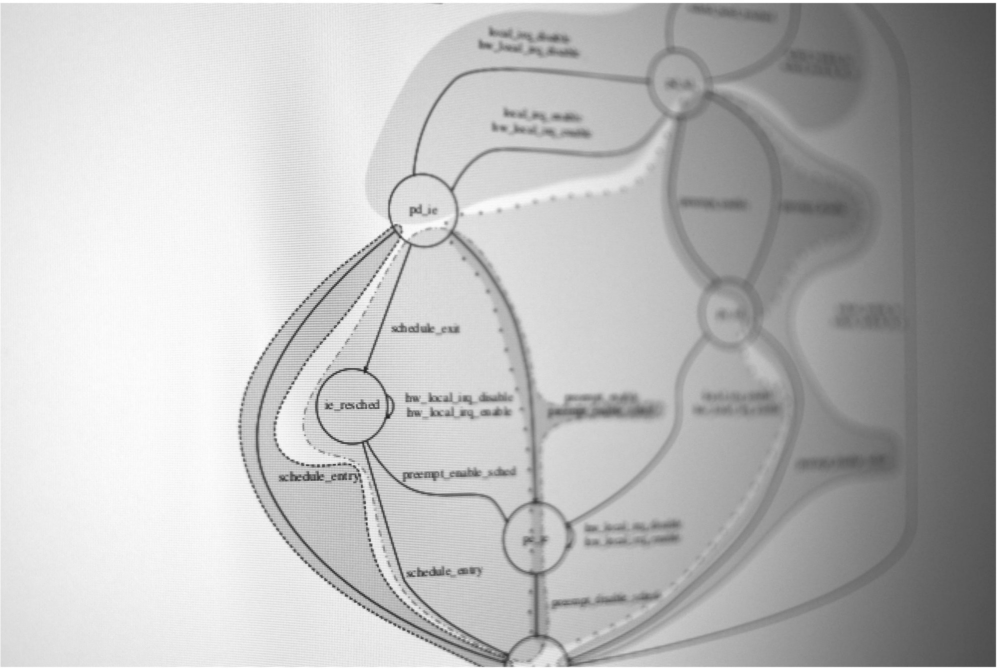


Figure 26 Workstation experiments: single-core system.

From formal specification to synchronization rules

Formally backed natural language arguments



- Generators
  - Basic/Independent behavior
  - e.g., irq_disable/enable, scheduler call
- Translated into a set of operations

- Specifications
  - Relations among generators
  - e.g., necessary conditions to call the scheduler
- Translated into a set of synchronization rules

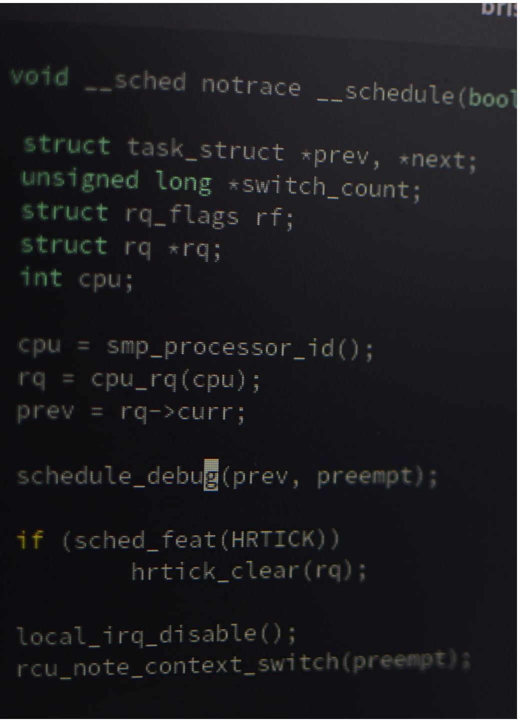
Scheduling latency definition

The **scheduling latency** experienced by an arbitrary thread τ is:

- the **longest time** elapsed **between** the *time A* in which any job of τ becomes **ready and with the highest priority**,
- and the *time F* in which the scheduler returns and allows τ to **execute its code**.

From the first necessary condition to *set need resched*, to the the last action after the scheduling, which is *enabling preemption* after the return from __schedule().

Interference and blocking



```
void __sched notrace __schedule(bool

 struct task_struct *prev, *next;
 unsigned long *switch_count;
 struct rq_flags rf;
 struct rq *rq;
 int cpu;

 cpu = smp_processor_id();
 rq = cpu_rq(cpu);
 prev = rq->curr;

 schedule_debug(prev, preempt);

 if (sched_feat(HRTICK))
         hrtick_clear(rq);

 local_irq_disable();
 rcu_note_context_switch(preempt);
```

The **scheduling latency** is caused by:

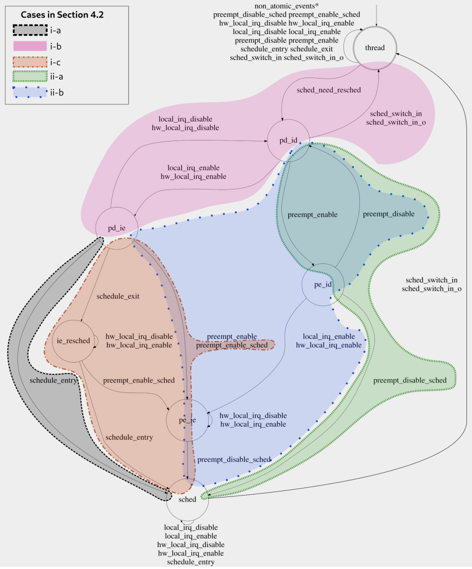
- ● **Blocking** from the current (and so lower) priority thread;
  - ● Including scheduling.
- ● **Interference** from IRQs and NMI.

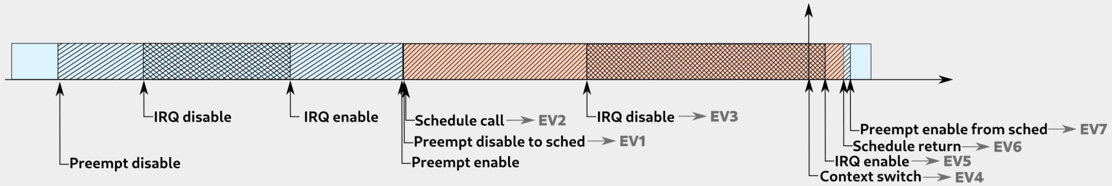
The scheduling latency in this paper refers to the delay between the notification of a new highest priority thread, to point in which this thread starts running its own code.

The highest priority thread can belong to any scheduler: the analysis is scheduler independent.

Blocking bound

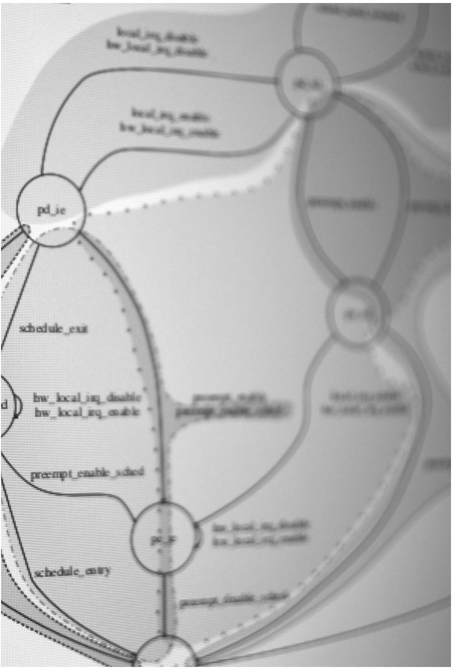
From the specification that bounds the block to a timeline

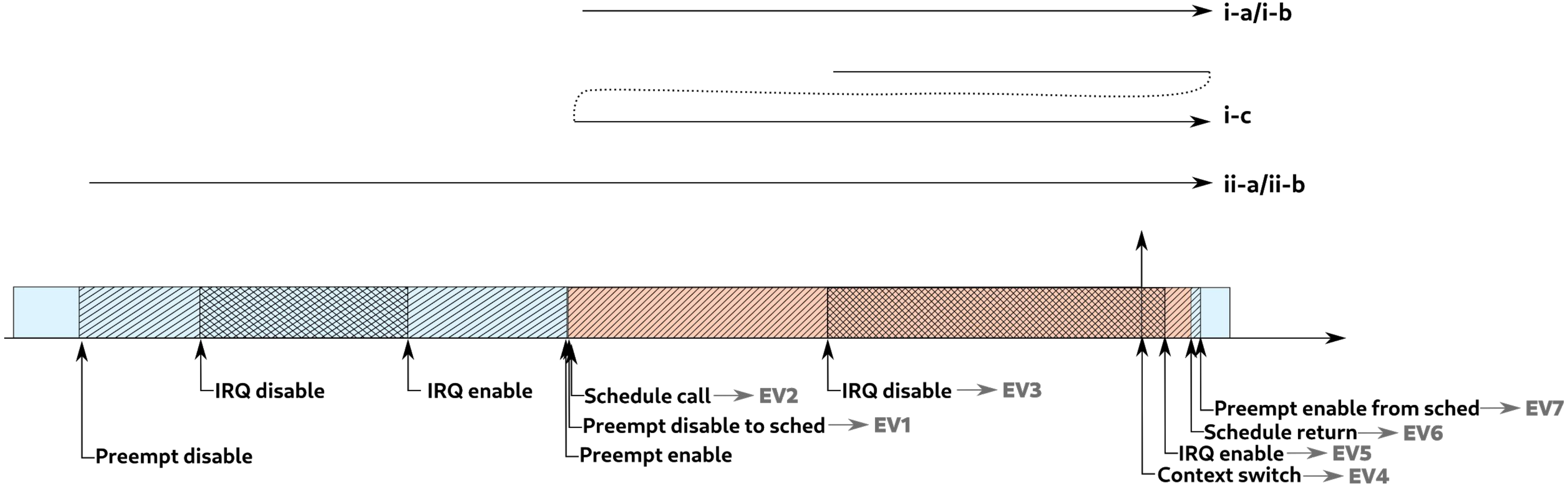




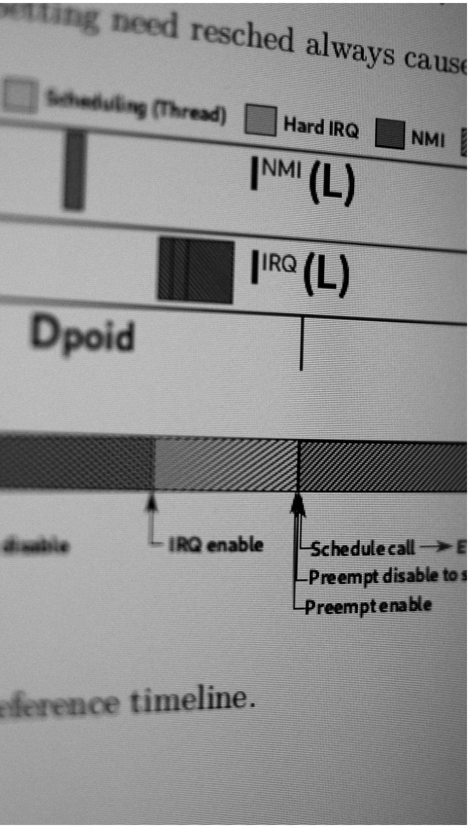
Timeline and cases

All possible cases





Blocking variables

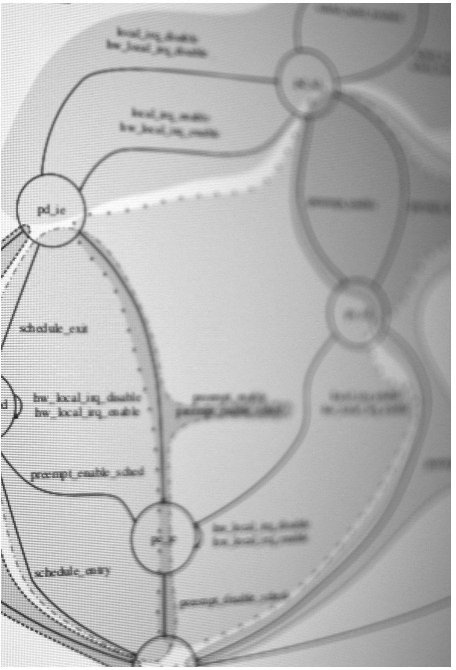


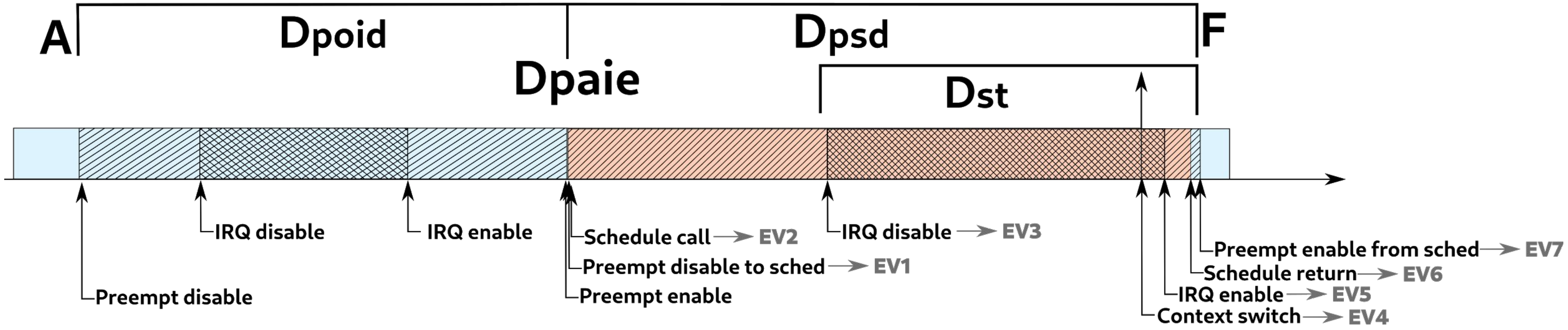
- **Dpoid**: preemption or interrupts disabled to postpone the scheduler;
- **Dpaie**: preemption and interrupts enabled, as a transient state from **poid** to **psd**; when scheduling a new highest priority thread.
- **Dpsd**: preemption disable to schedule;
- **Dst**: delay caused by the scheduling tail; the "non return" point in which a new arrived task will have to wait for the current scheduling operation to finish before scheduling.

In the model, the preemption control is specialized into two different operations: to *postpone the scheduler* (the most known behavior) or to *protect the execution of the __schedule()* function from recursion.

Timeline and cases

Variables in the the timeline





A

Dpoid

Dpaie

Dpsd

Dst

F

Preempt disable

IRQ disable

IRQ enable

Schedule call → EV2

Preempt disable to sched → EV1

Preempt enable

IRQ disable → EV3

Preempt enable from sched → EV7

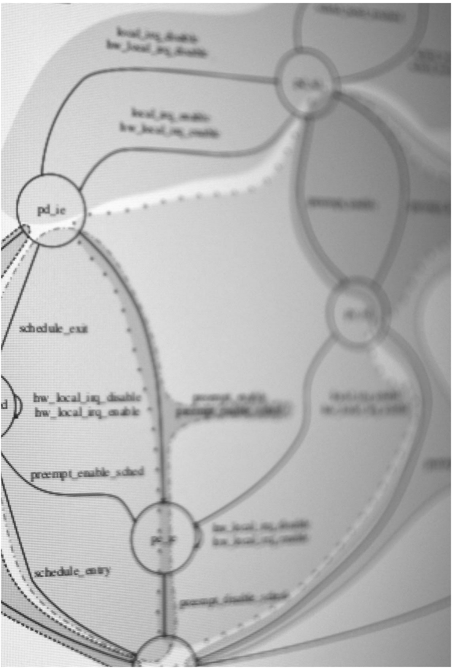
Schedule return → EV6

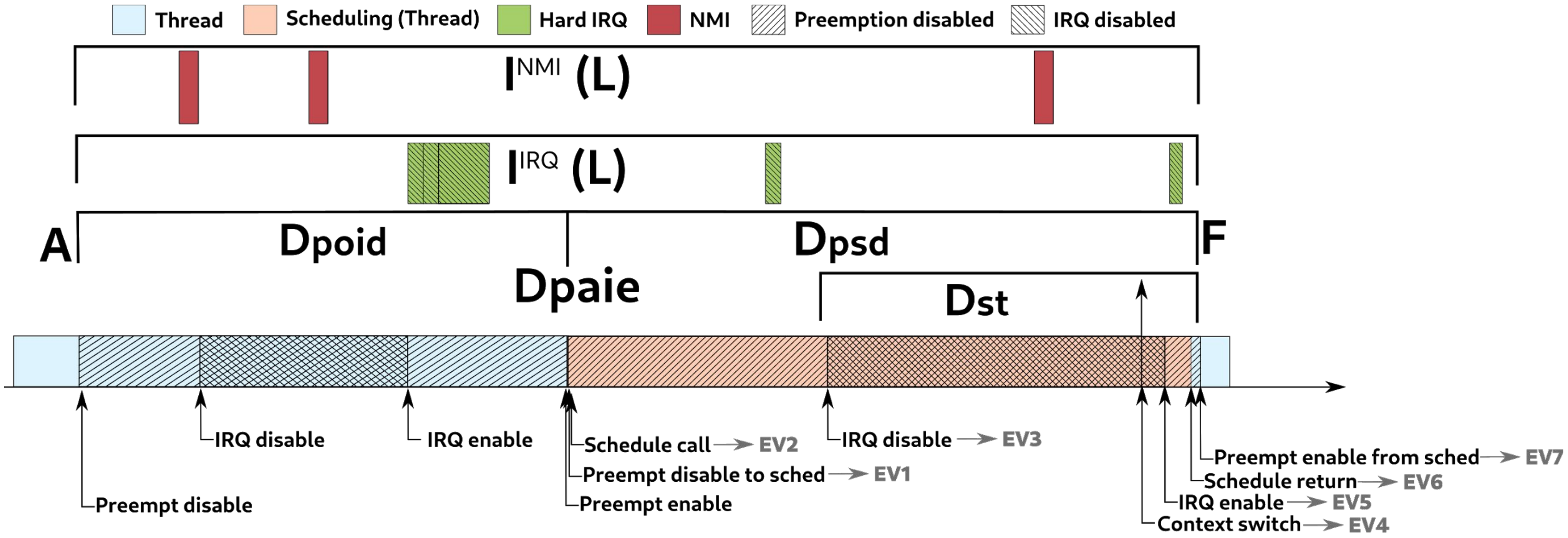
IRQ enable → EV5

Context switch → EV4

Timeline and cases

IRQ and NMI interference





Thread   Scheduling (Thread)   Hard IRQ   NMI   Preemption disabled   IRQ disabled

I^{NMI} (L)

I^{IRQ} (L)

A   Dpoid   Dpaie   Dpsd   F

Dst

Preempt disable

IRQ disable

IRQ enable

Schedule call → EV2

Preempt disable to sched → EV1

Preempt enable

IRQ disable → EV3

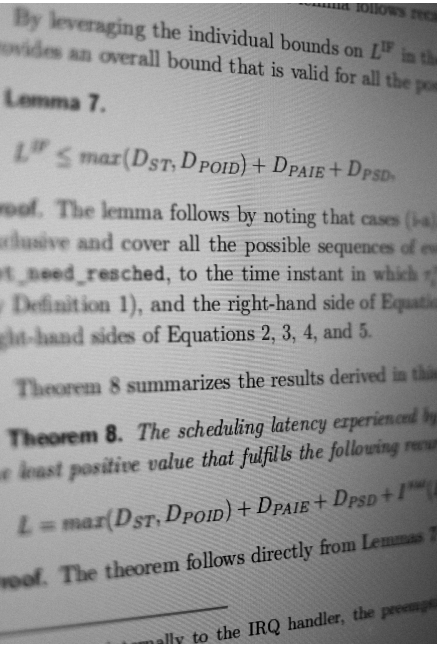
Preempt enable from sched → EV7

Schedule return → EV6

IRQ enable → EV5

Context switch → EV4

And the *scheduling latency* bounds to:



follows rec

By leveraging the individual bounds on L^{IF} in th
ovides an overall bound that is valid for all the po

Lemma 7.

$L^{IF} \leq max(D_{ST}, D_{POID}) + D_{PAIE} + D_{PSD}$.

oof. The lemma follows by noting that cases (i-a)
clusive and cover all the possible sequences of ev
t_need_resched, to the time instant in which
Definition 1), and the right-hand side of Equatio
ght-hand sides of Equations 2, 3, 4, and 5.

Theorem 8 summarizes the results derived in thi

Theorem 8. *The scheduling latency experienced by*
e least positive value that fulfills the following recu

$L = max(D_{ST}, D_{POID}) + D_{PAIE} + D_{PSD} + I$

oof. The theorem follows directly from Lemmas 7

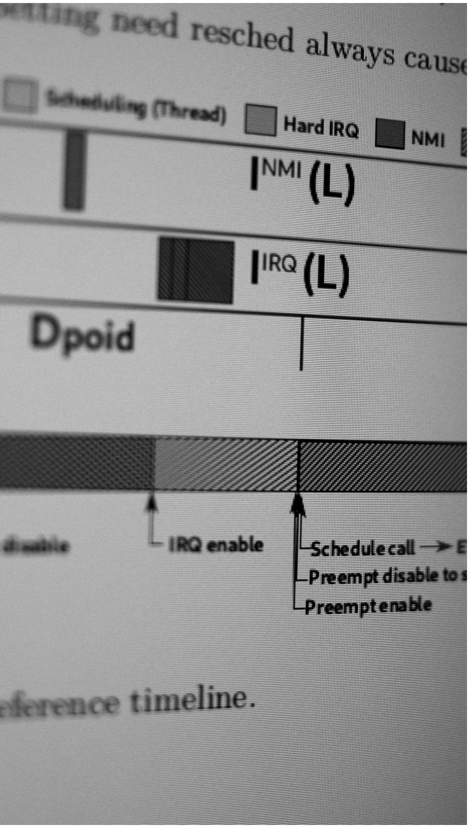
ally to the IRQ handler, the preemp

$$L = \max(D_{ST}, D_{POID}) + D_{PAIE} + D_{PSD} + I^{NMI}(L) + I^{IRQ}(L)$$

The bound considers all possible cases. Note that the Latency *L* is present in both sides of the equation.

So, L is bounded by the least positive value fulfilling the equation (like on RTA).

Interrupts are workload dependent



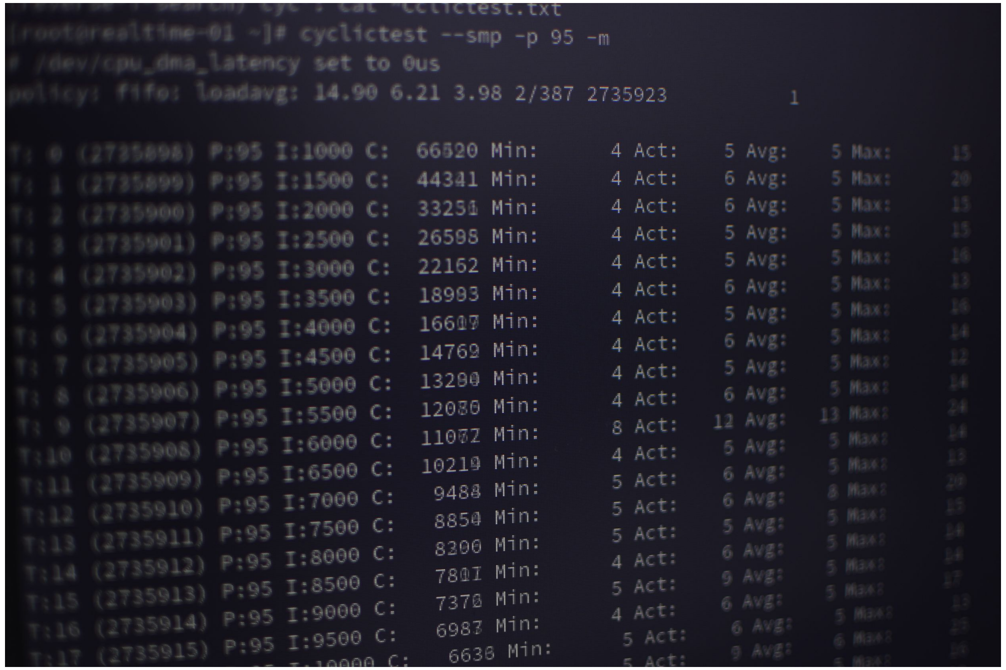
- Instead of proposing “the best” interrupt characterization, the rtsl reports the scheduling latency based on some well-known characterizations:
    - No interrupt
    - Worst single interrupt
    - Single occurence of all interrupts
    - Sporadic
    - Sliding window (Author’s preferred)
    - Sliding window with oWCET

This topic was heavily discussed at the Real-time Micro Conference (inside Linux Plumbers) in 2019, more info here:

Red Hat

A practical scheduling latency estimation tool

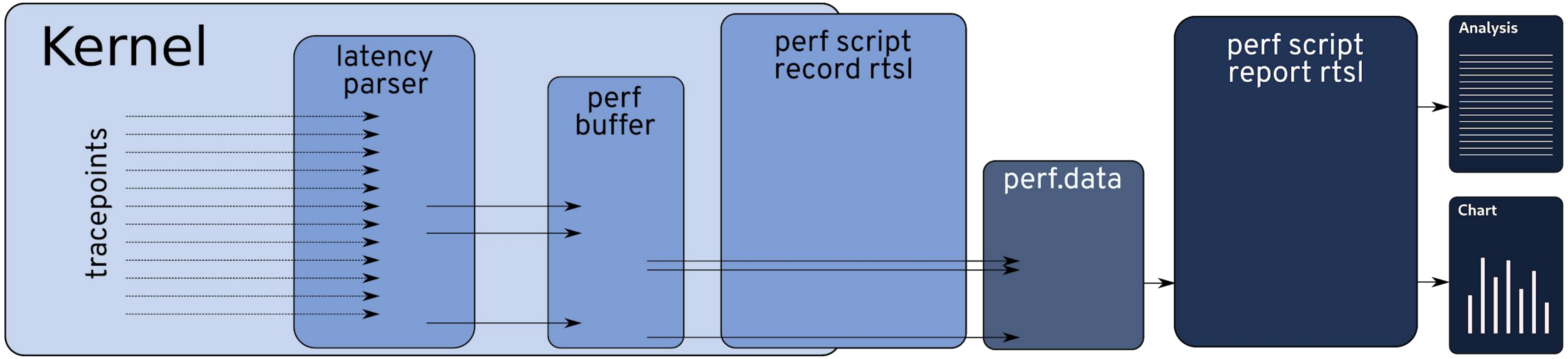
Method and challenges



```
[root@realtime-01 ~]# cyclictest --smp -p 95 -m
# /dev/cpu_dma_latency set to 0us
policy: fifo: loadavg: 14.90 6.21 3.98 2/387 2735923          1

T: 0 (2735898) P:95 I:1000 C:  66520 Min:      4 Act:    5 Avg:    5 Max:      15
T: 1 (2735899) P:95 I:1500 C:  44341 Min:      4 Act:    6 Avg:    5 Max:      20
T: 2 (2735900) P:95 I:2000 C:  33251 Min:      4 Act:    6 Avg:    5 Max:      15
T: 3 (2735901) P:95 I:2500 C:  26588 Min:      4 Act:    5 Avg:    5 Max:      15
T: 4 (2735902) P:95 I:3000 C:  22162 Min:      4 Act:    5 Avg:    5 Max:      16
T: 5 (2735903) P:95 I:3500 C:  18983 Min:      4 Act:    6 Avg:    5 Max:      13
T: 6 (2735904) P:95 I:4000 C:  16617 Min:      4 Act:    5 Avg:    5 Max:      16
T: 7 (2735905) P:95 I:4500 C:  14769 Min:      4 Act:    6 Avg:    5 Max:      14
T: 8 (2735906) P:95 I:5000 C:  13280 Min:      4 Act:    5 Avg:    5 Max:      12
T: 9 (2735907) P:95 I:5500 C:  12080 Min:      4 Act:    6 Avg:    5 Max:      14
T:10 (2735908) P:95 I:6000 C:  11072 Min:      8 Act:   12 Avg:   13 Max:      24
T:11 (2735909) P:95 I:6500 C:  10219 Min:      4 Act:    5 Avg:    5 Max:      14
T:12 (2735910) P:95 I:7000 C:   9488 Min:      5 Act:    6 Avg:    5 Max:      13
T:13 (2735911) P:95 I:7500 C:   8854 Min:      5 Act:    6 Avg:    8 Max:      20
T:14 (2735912) P:95 I:8000 C:   8206 Min:      5 Act:    5 Avg:    5 Max:      15
T:15 (2735913) P:95 I:8500 C:   7801 Min:      4 Act:    6 Avg:    5 Max:      14
T:16 (2735914) P:95 I:9000 C:   7378 Min:      5 Act:    9 Avg:    5 Max:      14
T:17 (2735915) P:95 I:9500 C:   6983 Min:      4 Act:    6 Avg:    5 Max:      17
T:18 (2735916) P:95 I:10000 C:  6638 Min:      5 Act:    9 Avg:    6 Max:
```

- Based on the latency bound
- The latency bound is based on the model
- The *model* is based on tracing of events
    - but high frequency events
        - hundreds MB/sec/CPU
- Challenges:
    - To minimize the (runtime) overhead
    - Work out-of-the-box

rt_sched_latency (rtsl)



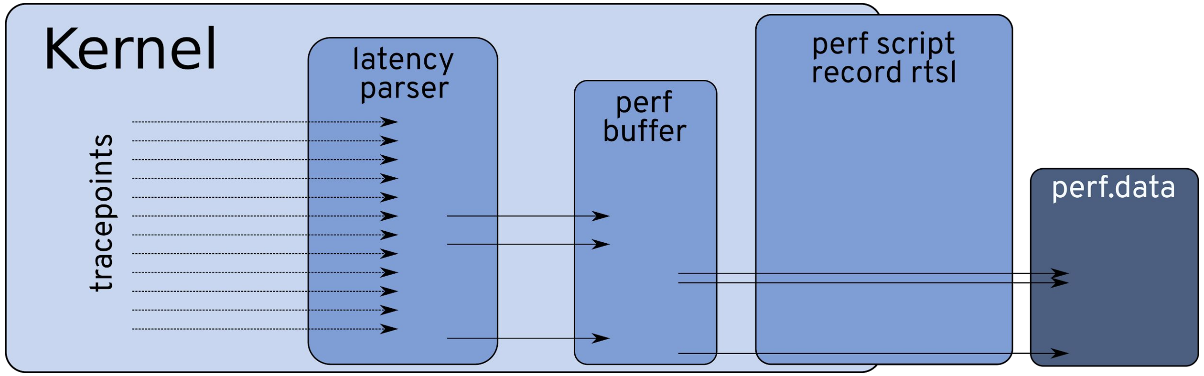
Based on **perf**

Works in two phases:

- The **record** mode saves the trace data;
- The **report** mode process the trace and does the analysis.

record phase

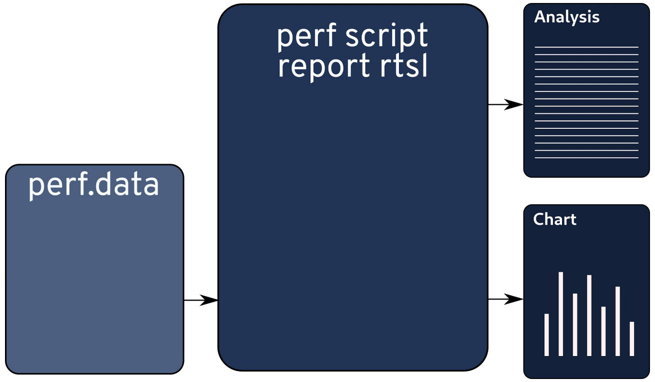
Low overhead trace recording



- Filters the high frequency trace
    - Doing in-kernel processing
- For blocking variables:
    - Reports only the discover of new max values
- For IRQ and NMI:
    - Reports one event for each occurrence
- Discounts the interference:
    - e.g., IRQ interference on a **poid**

report phase

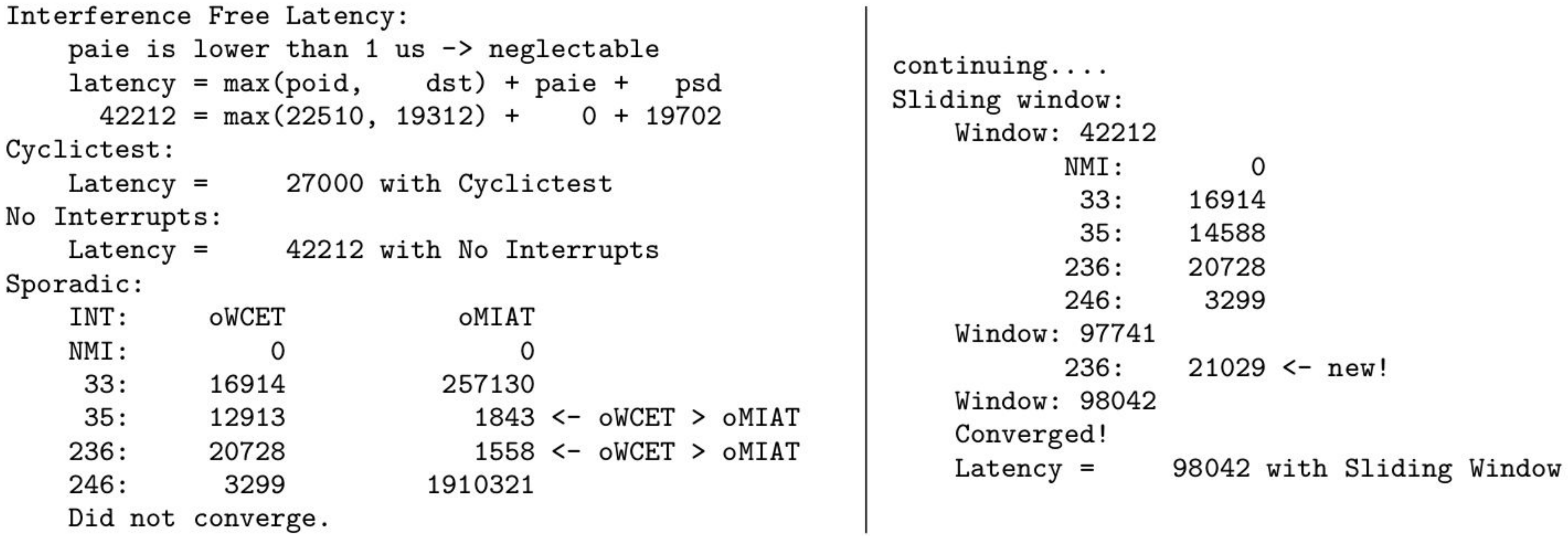
Low overhead trace recording



- After the capture, analyzes the trace.
  - All in user-space.
- Most of the analysis is done in python
  - Easy to extend
- Two outputs:
  - Textual: good for debug
  - Chart: good comparisons (and papers :-))
- Does a per-cpu scheduling latency analysis
  - Using different IRQ/NMI characterization...

rtsl report output

Textual output

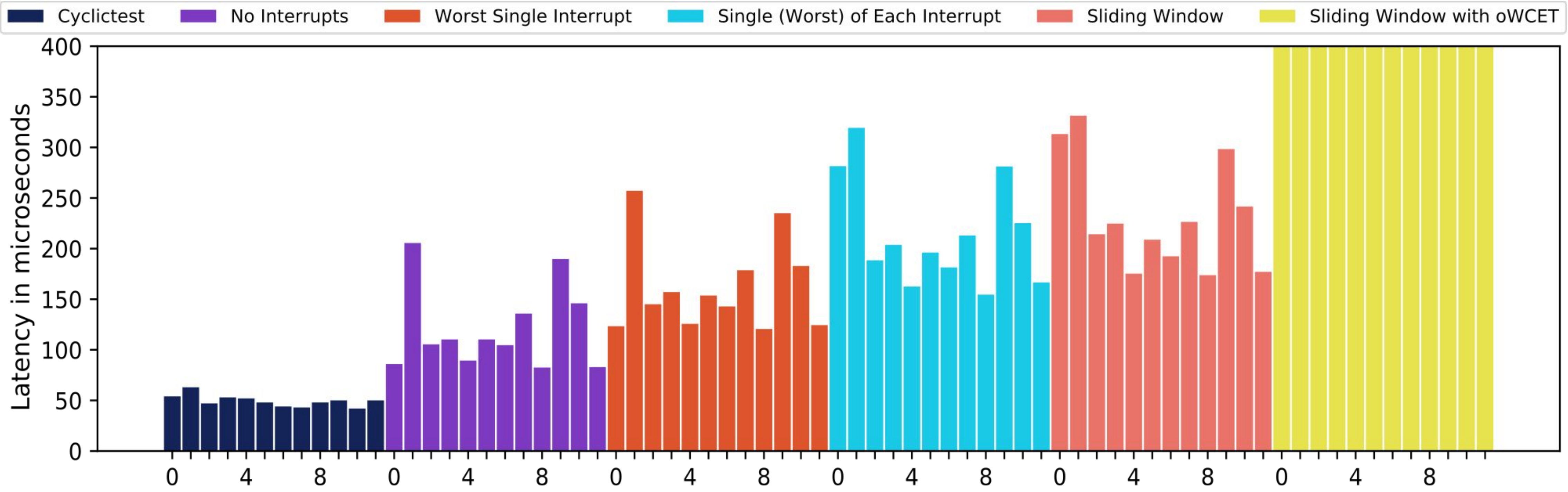


```
Interference Free Latency:
    paie is lower than 1 us -> neglectable
    latency = max(poid,    dst) + paie +   psd
      42212 = max(22510, 19312) +    0 + 19702
Cyclictest:
    Latency =     27000 with Cyclictest
No Interrupts:
    Latency =     42212 with No Interrupts
Sporadic:
    INT:     oWCET           oMIAT
    NMI:         0               0
     33:     16914          257130
     35:     12913            1843 <- oWCET > oMIAT
    236:     20728            1558 <- oWCET > oMIAT
    246:      3299         1910321
    Did not converge.
```

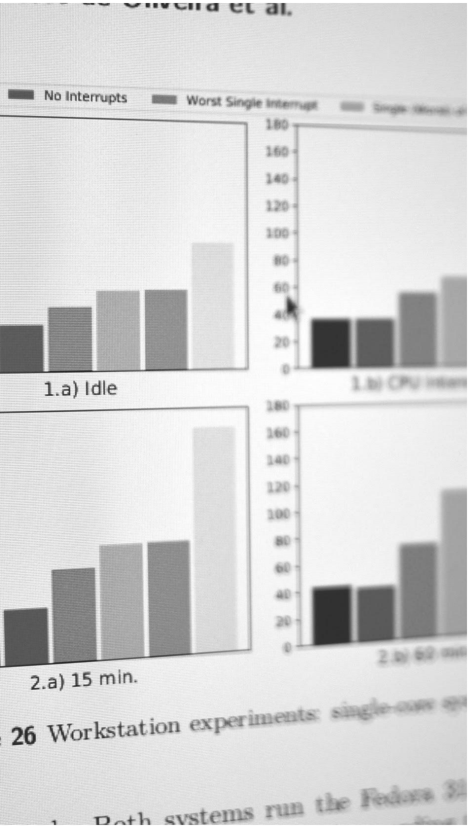
```
continuing....
Sliding window:
    Window: 42212
           NMI:        0
            33:    16914
            35:    14588
           236:    20728
           246:     3299
    Window: 97741
           236:    21029 <- new!
    Window: 98042
    Converged!
    Latency =     98042 with Sliding Window
```

rtsl report output

Chart output



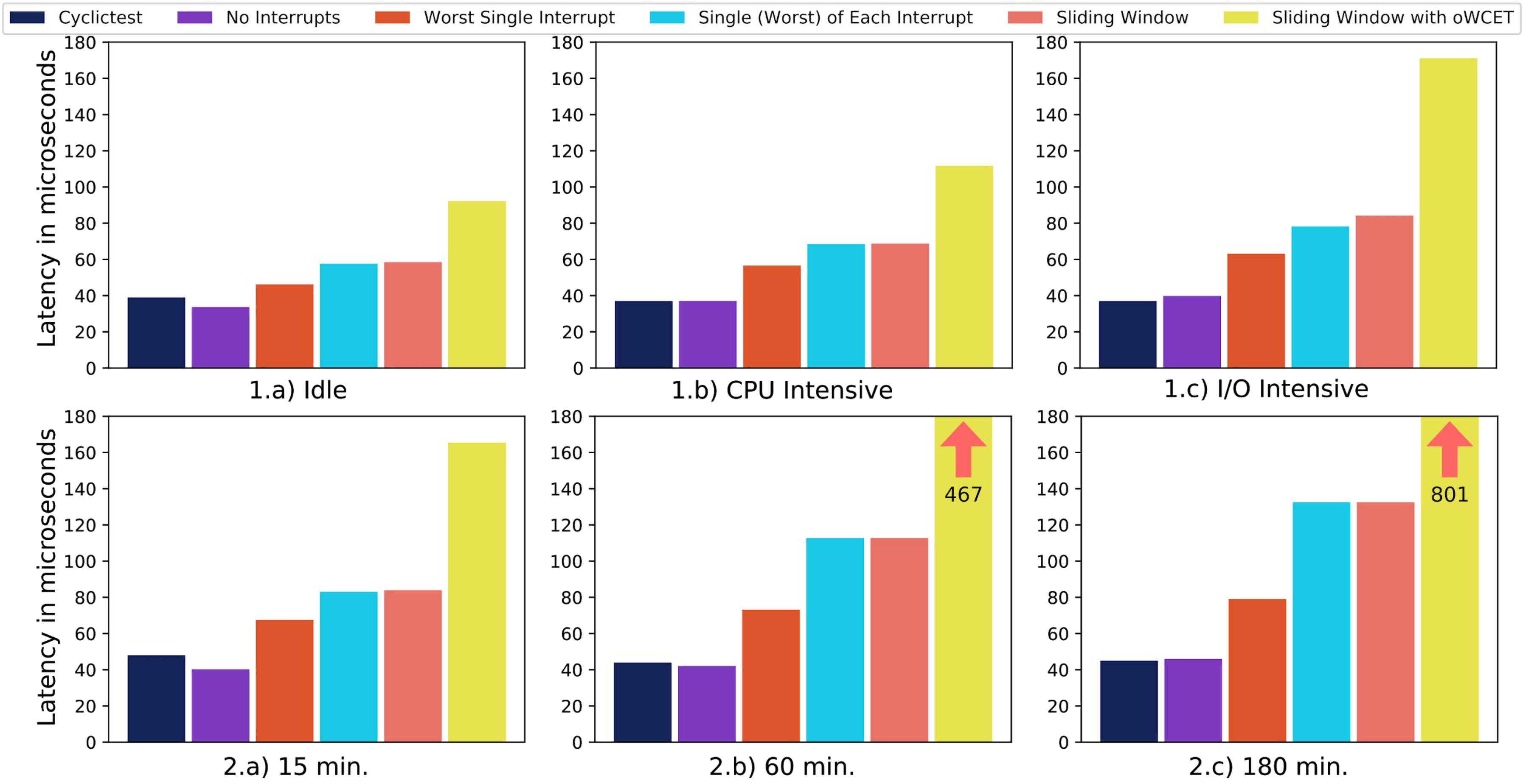
Experiments



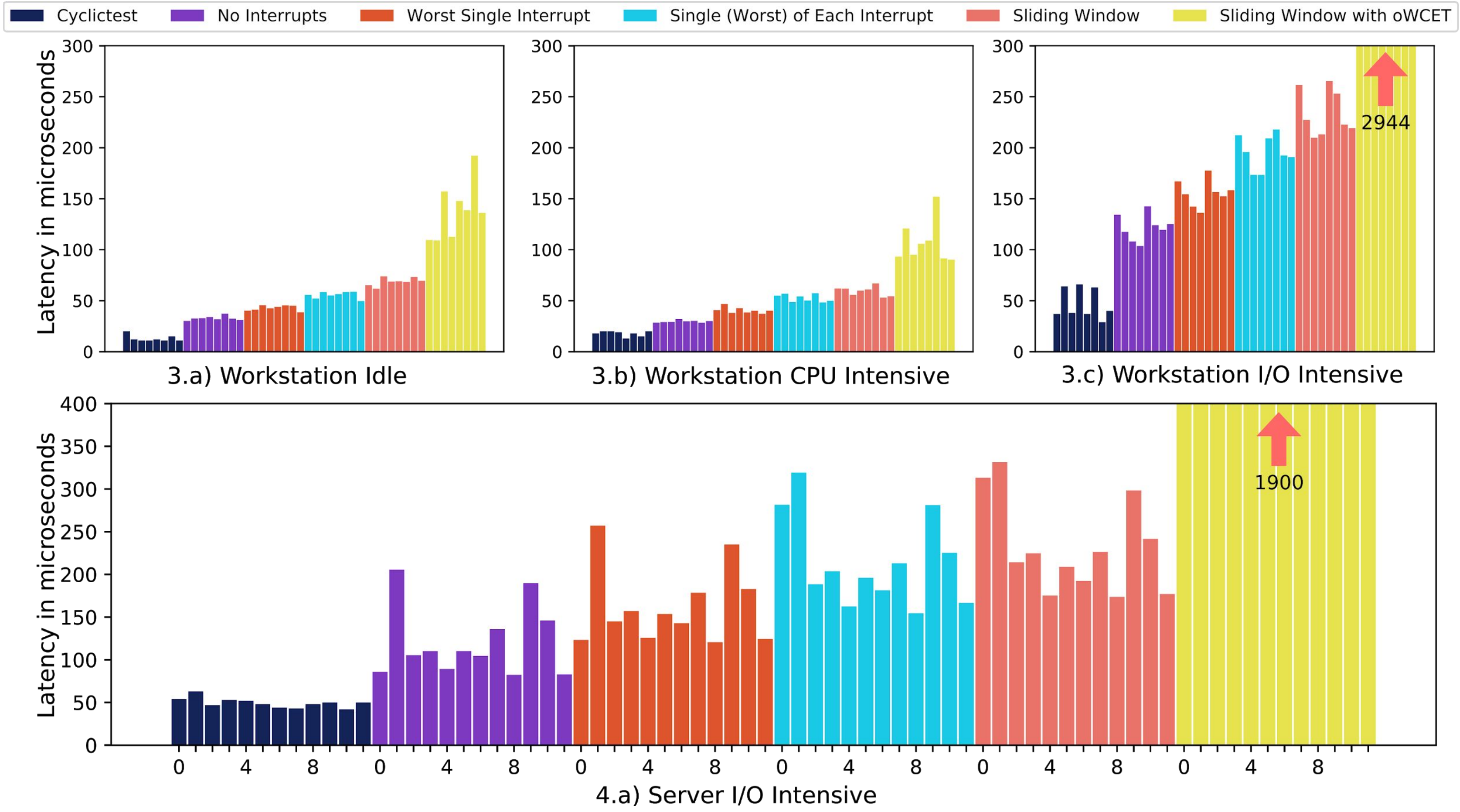
- Scheduling latency measurements on two systems:
  - workstation: eighth CPUs
  - server: twelve CPUs server
- Experiments:
  - Single-core
    - Different duration
    - Different workload
  - Multi-core
- Running in parallel with cyclictest
- Note: The goal of the experiments is to demonstrate the tool, not to define worst values.

The experiments passed by the artifact evaluation!

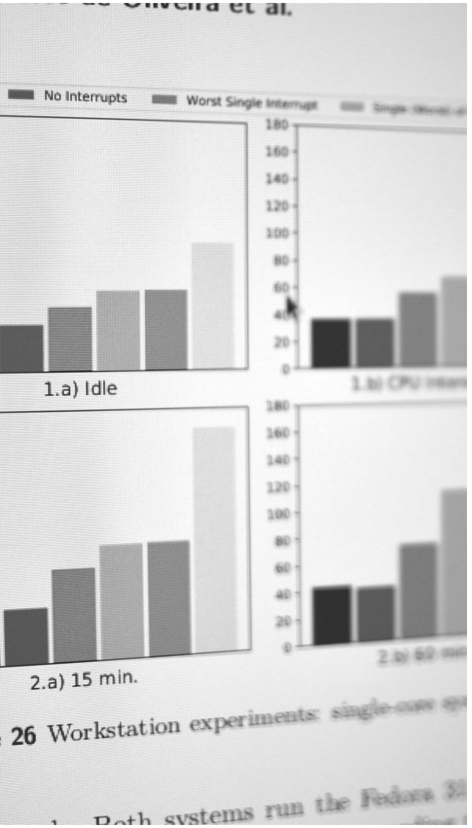
Single-core experiments



Multicore experiments



Conclusions



- The PREEMPT_RT preemption model is deterministic, and the scheduling latency is bounded.
- The approach presented in this paper opens the door for a new set of real-time analysis for Linux;
  - The analytical interpretation of Linux thread model developed in this paper untight the Linux complexity, enabling the reasoning at a more sophisticated level.
- Even though rtsl finds higher scheduling latency values, they are still low enough to justify Linux as RTOS on the current scenarios.
- rtsl is practical, and resolves many problems of cyclictest.
  - E.g., it can be used to point to the root causes of the latency;
  - But still can, and should, be improved:
    - Both with code, and other analysis.

For more information about this paper, like source code, other comments, Q&A, check its companion page!

Thank you

Red Hat is the world's leading provider of enterprise open source software solutions. Award-winning support, training, and consulting services make Red Hat a trusted adviser to the Fortune 500.

linkedin.com/company/red-hat

youtube.com/user/RedHatVideos

facebook.com/redhatinc

twitter.com/RedHat