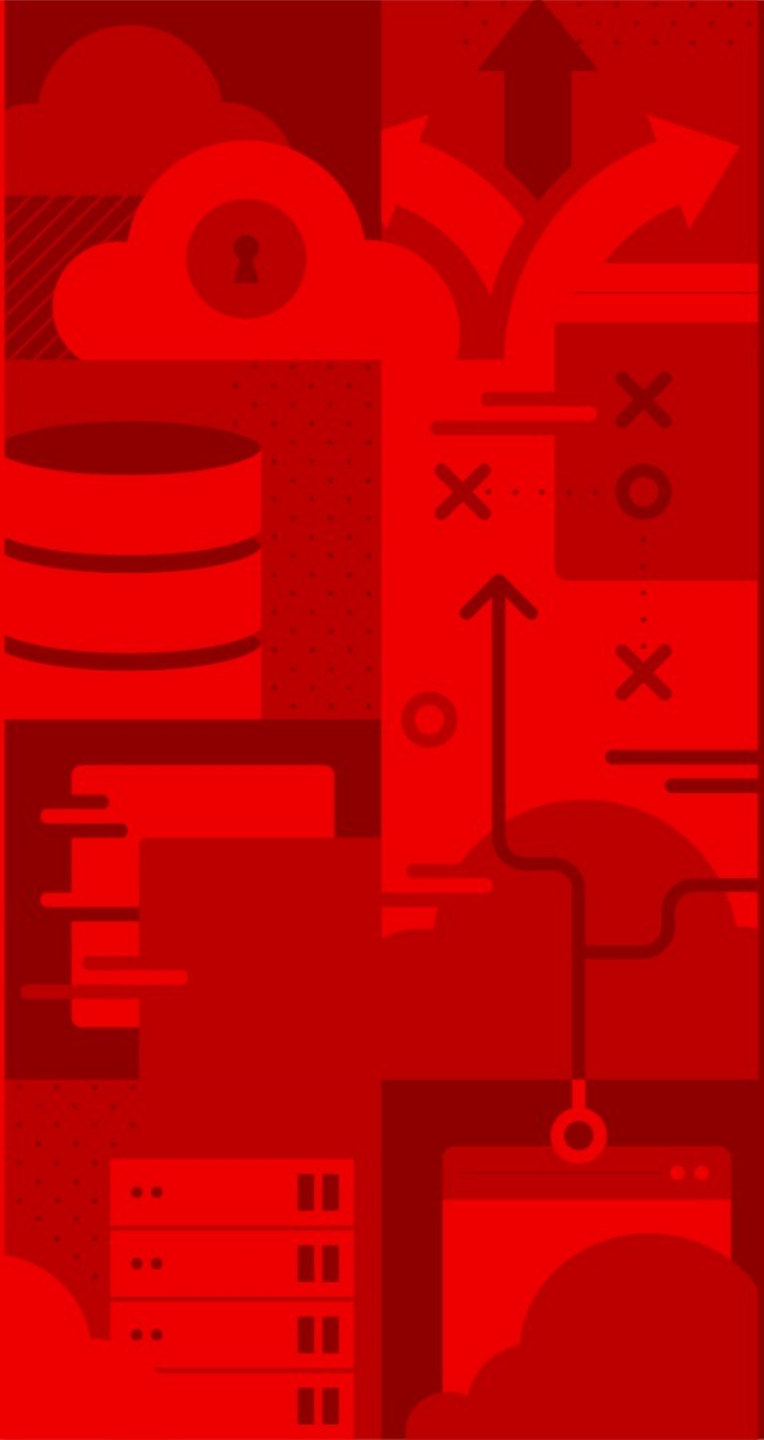
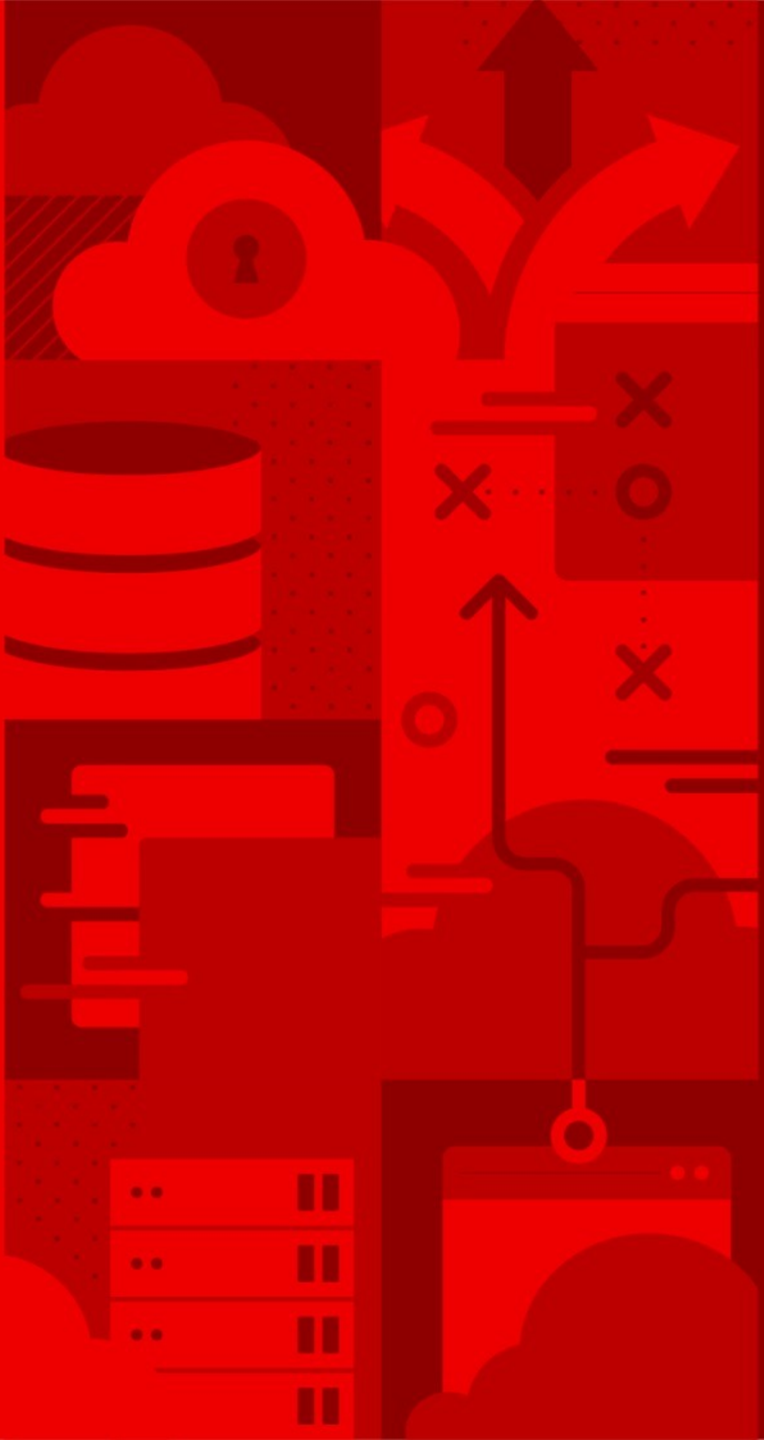
Red Hat
Enterprise Linux

Real-time Linux: What is, what is not and what is next.

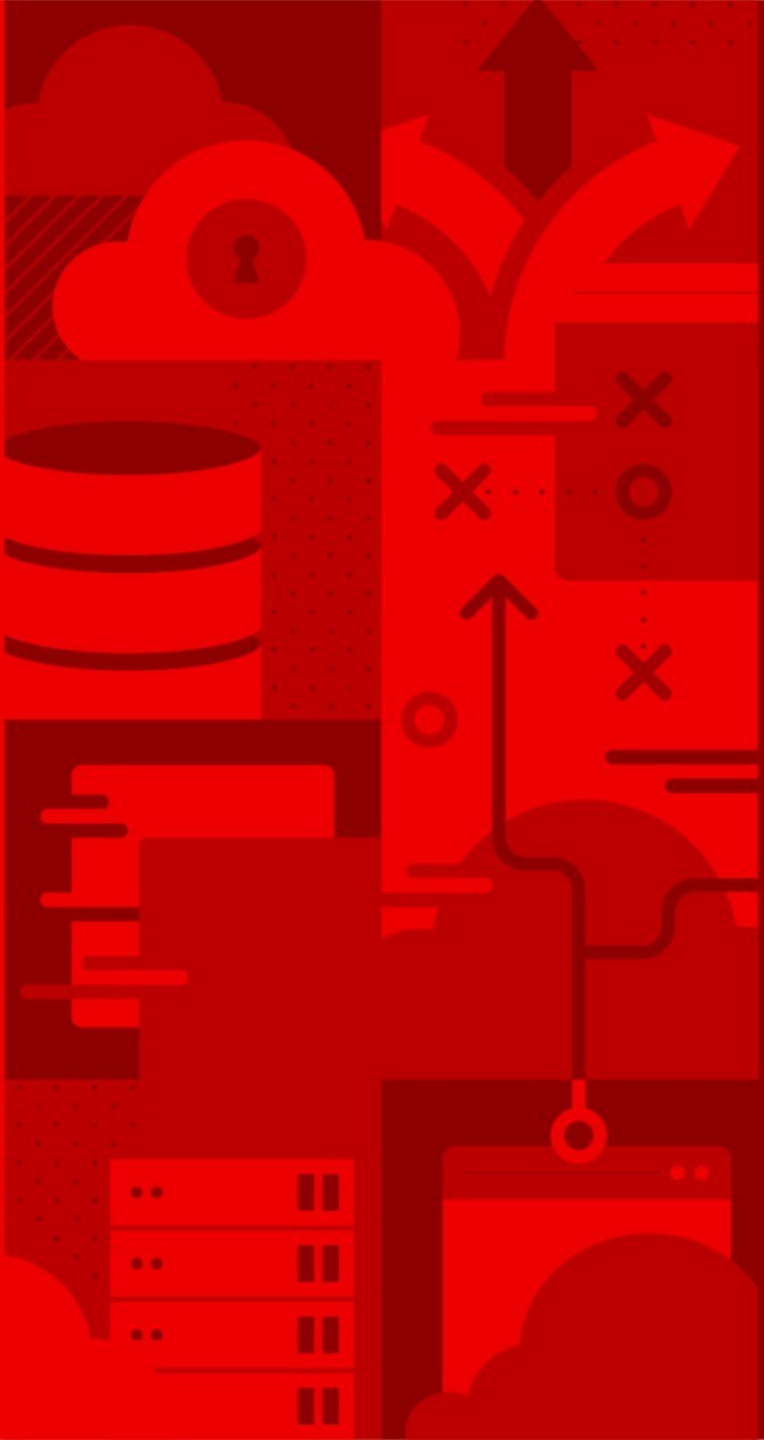
Daniel Bristot de Oliveira
Principal Software Engineer, Red Hat
Researcher, Retis Lab – Scuola Superiore Sant'Anna



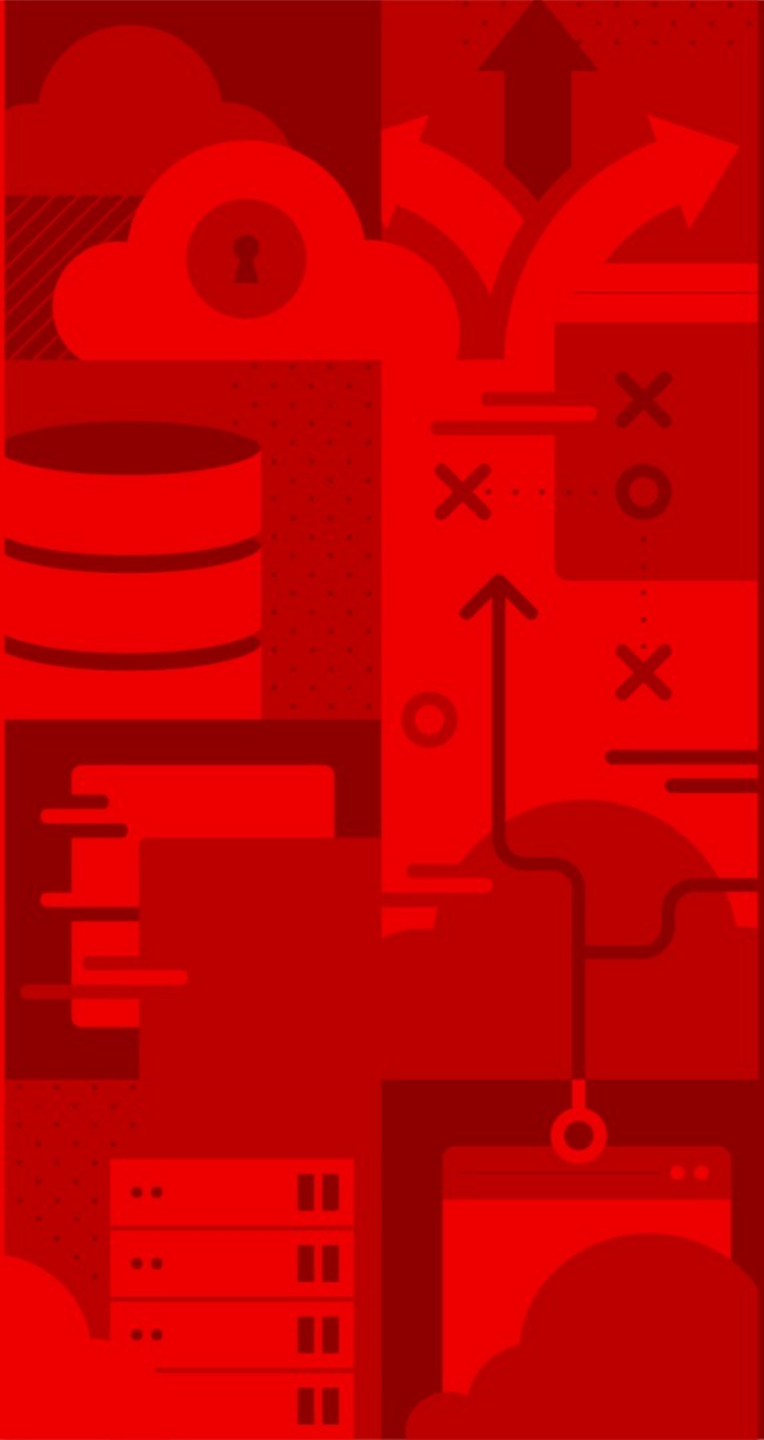
PREEMPT_RT is mainline!



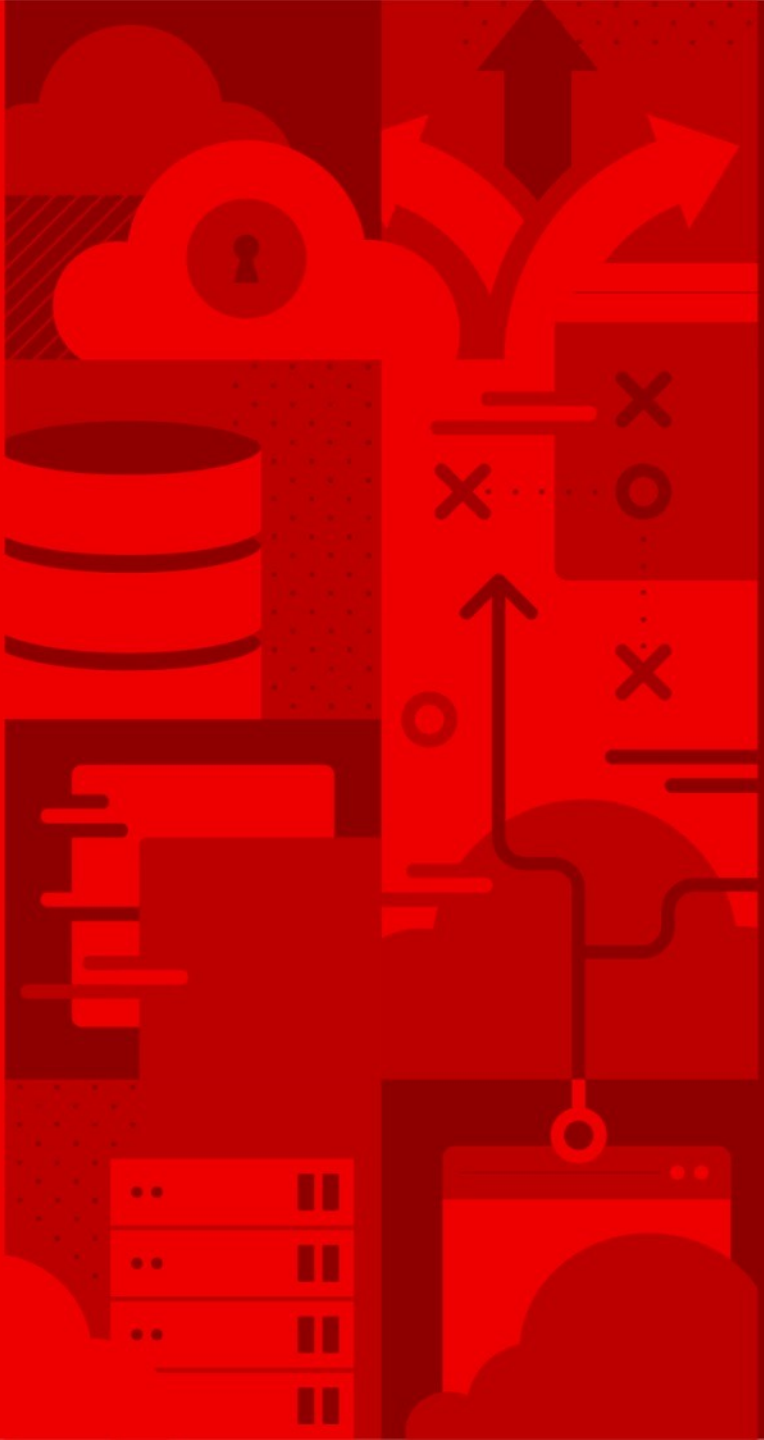
So… Real-time Linux is *done*?



Let’s discuss…

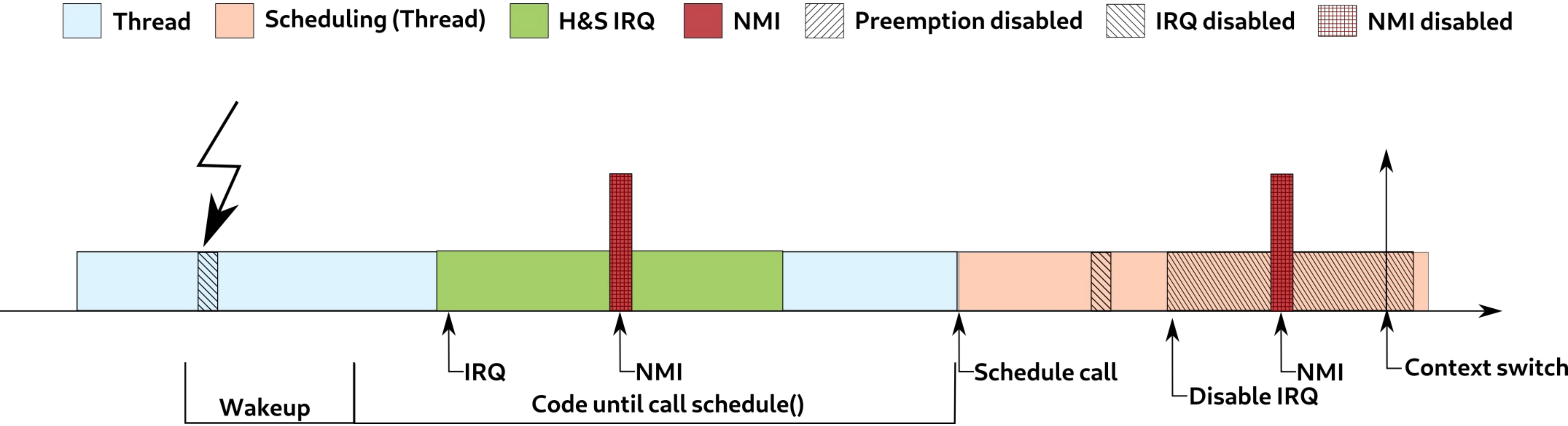


But before start...



What is?

Non-rt Timeline



Thread

Scheduling (Thread)

H&S IRQ

NMI

Preemption disabled

IRQ disabled

NMI disabled

Wakeup

IRQ

NMI

Code until call schedule()

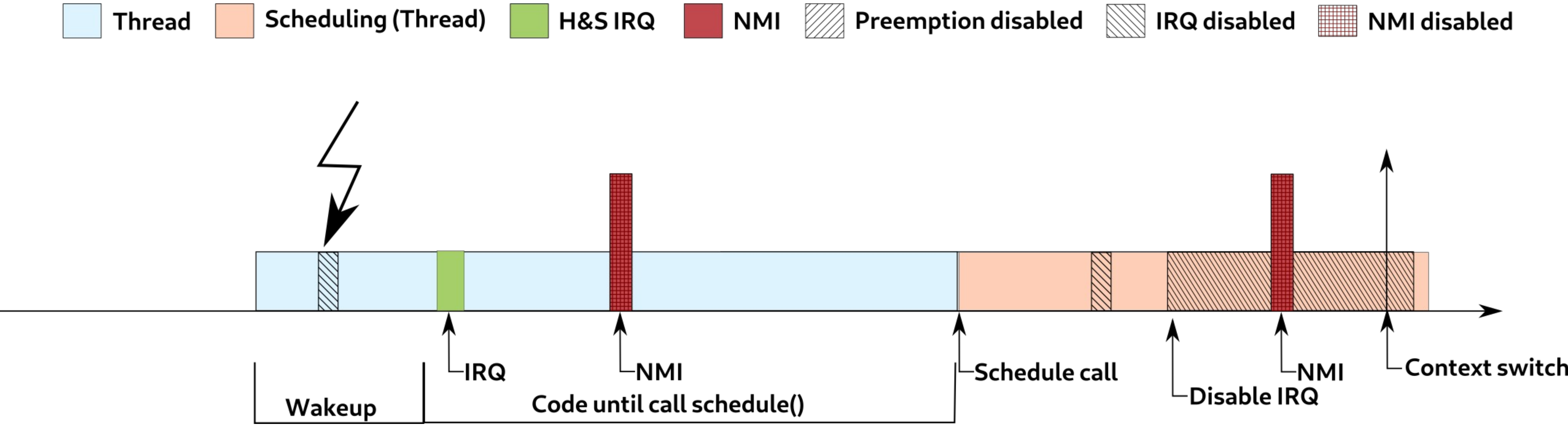
Schedule call

Disable IRQ

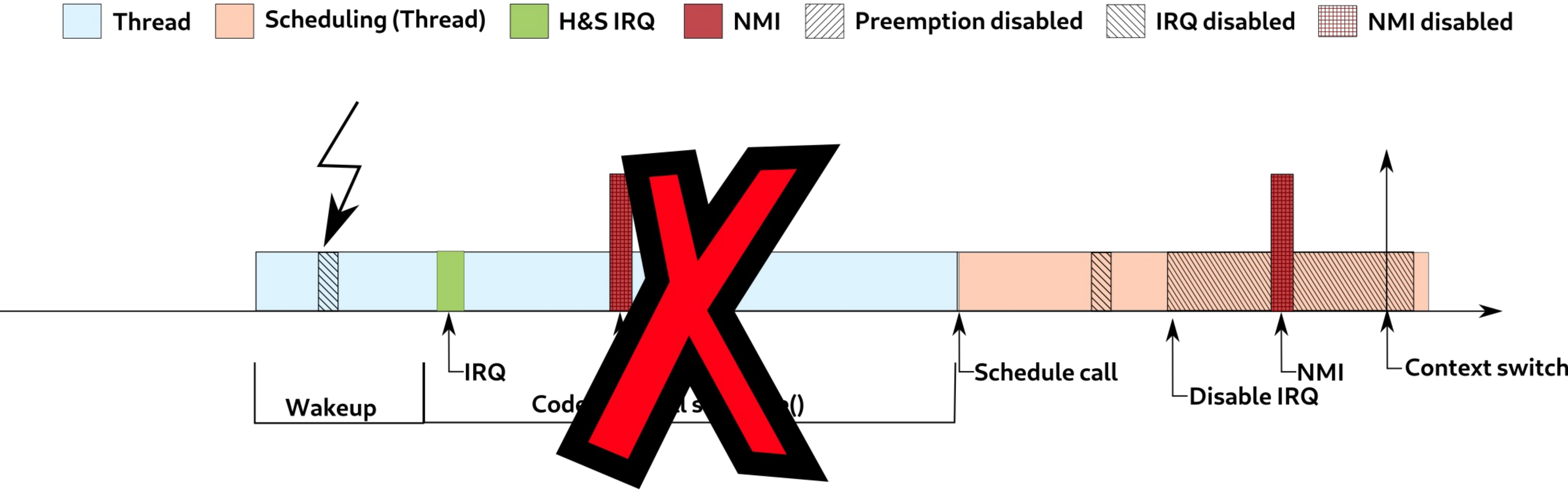
NMI

Context switch

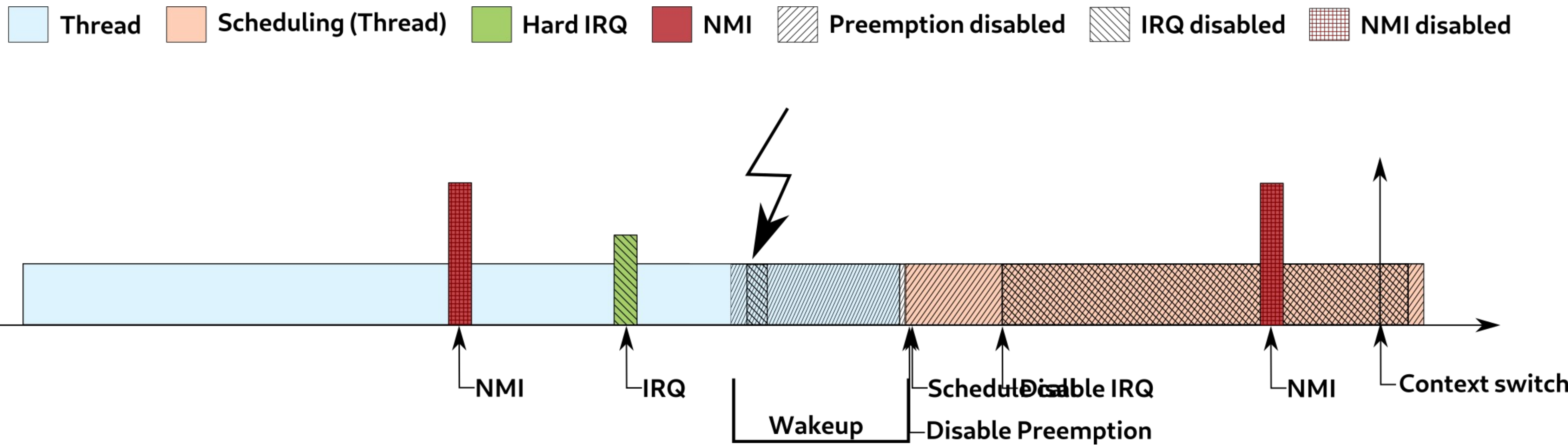
Non-rt Timeline + thread IRQ



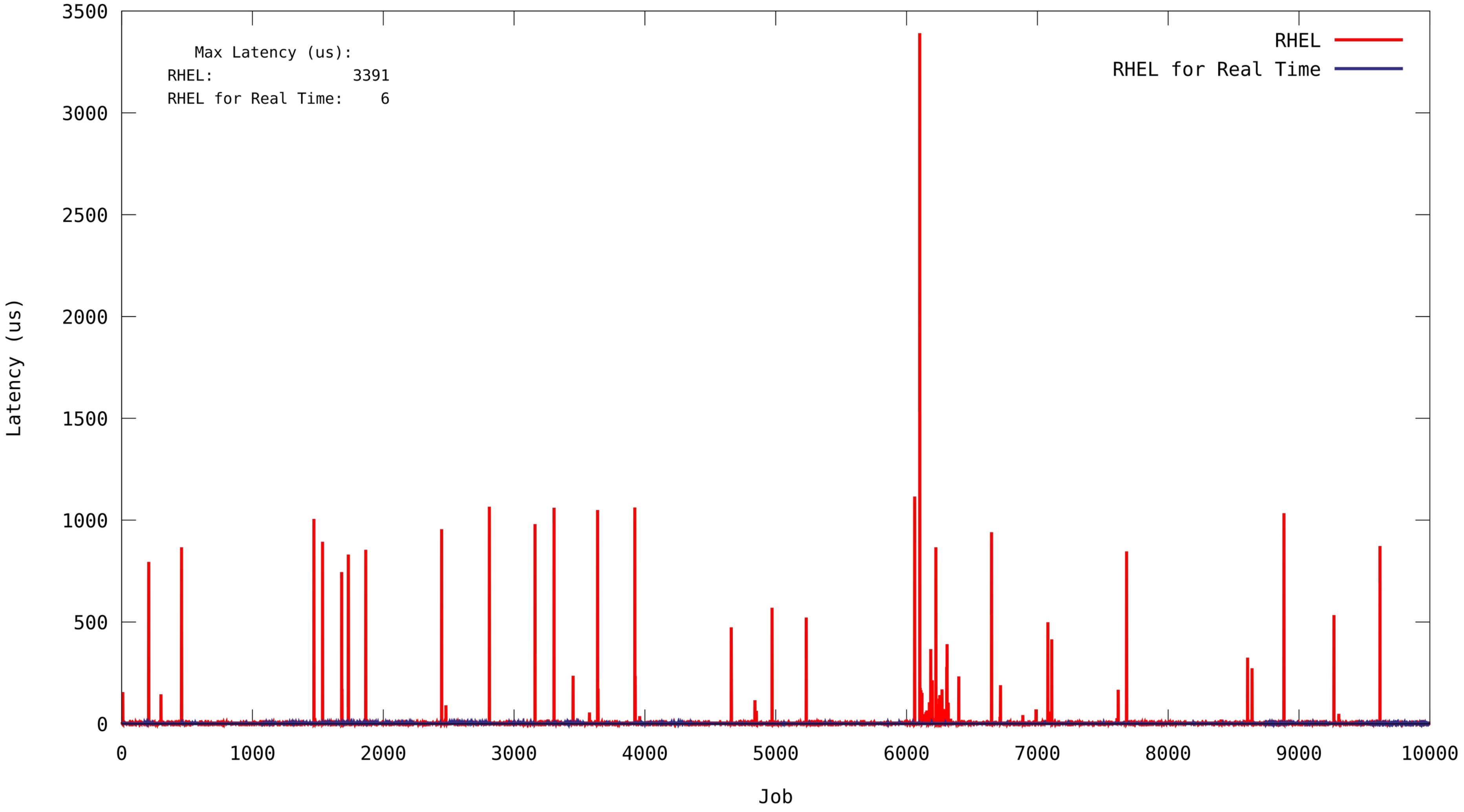
Non-preempt -> preempt

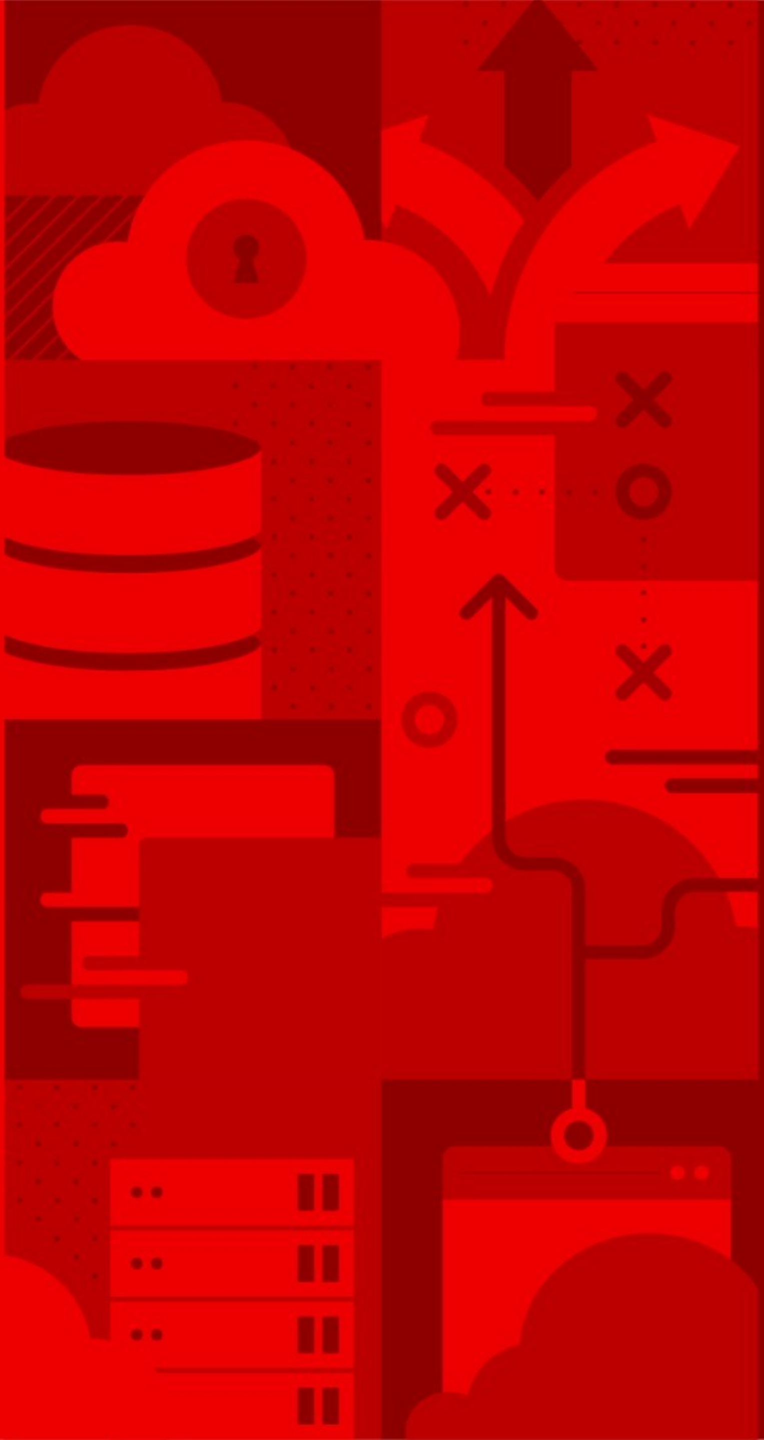


Non-preempt -> preempt



Latency is the output!

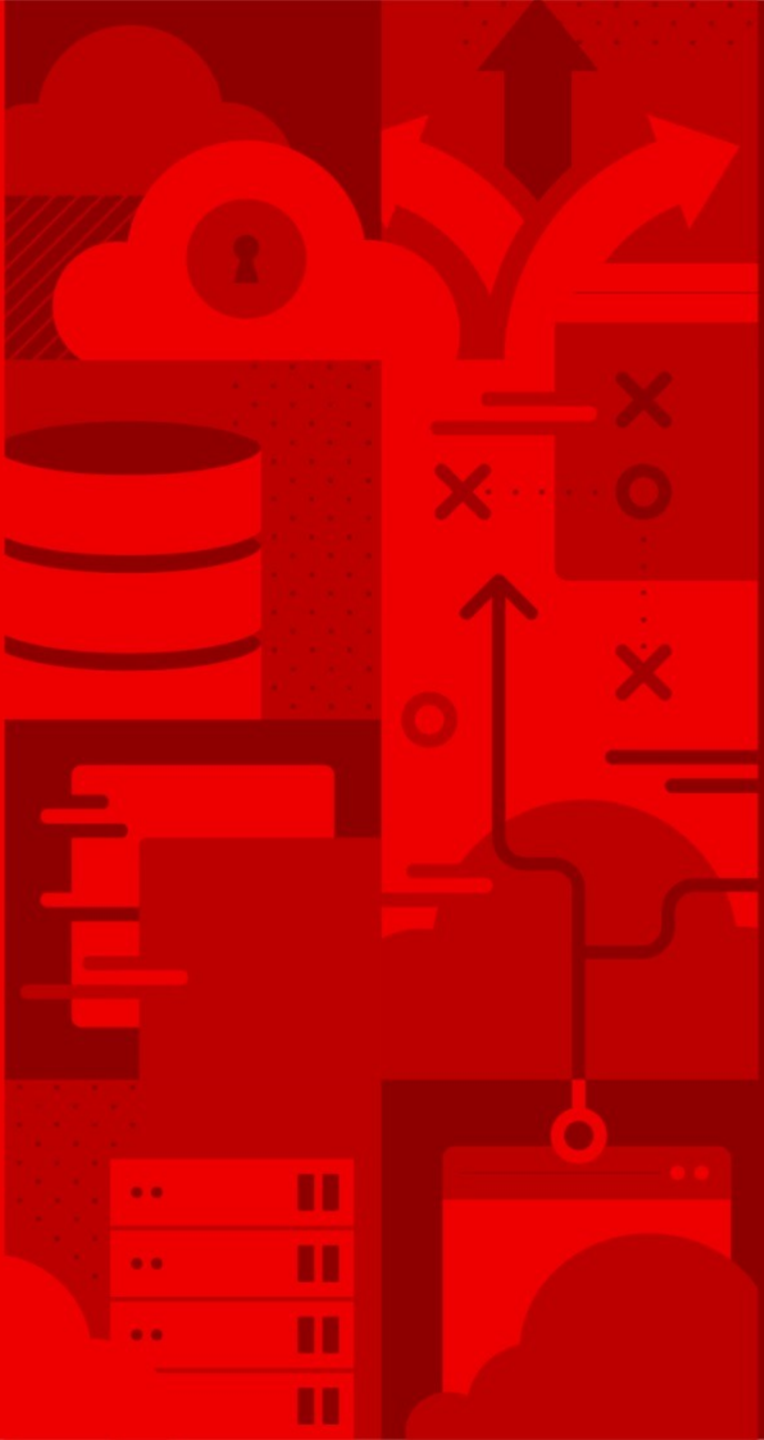




That is good!

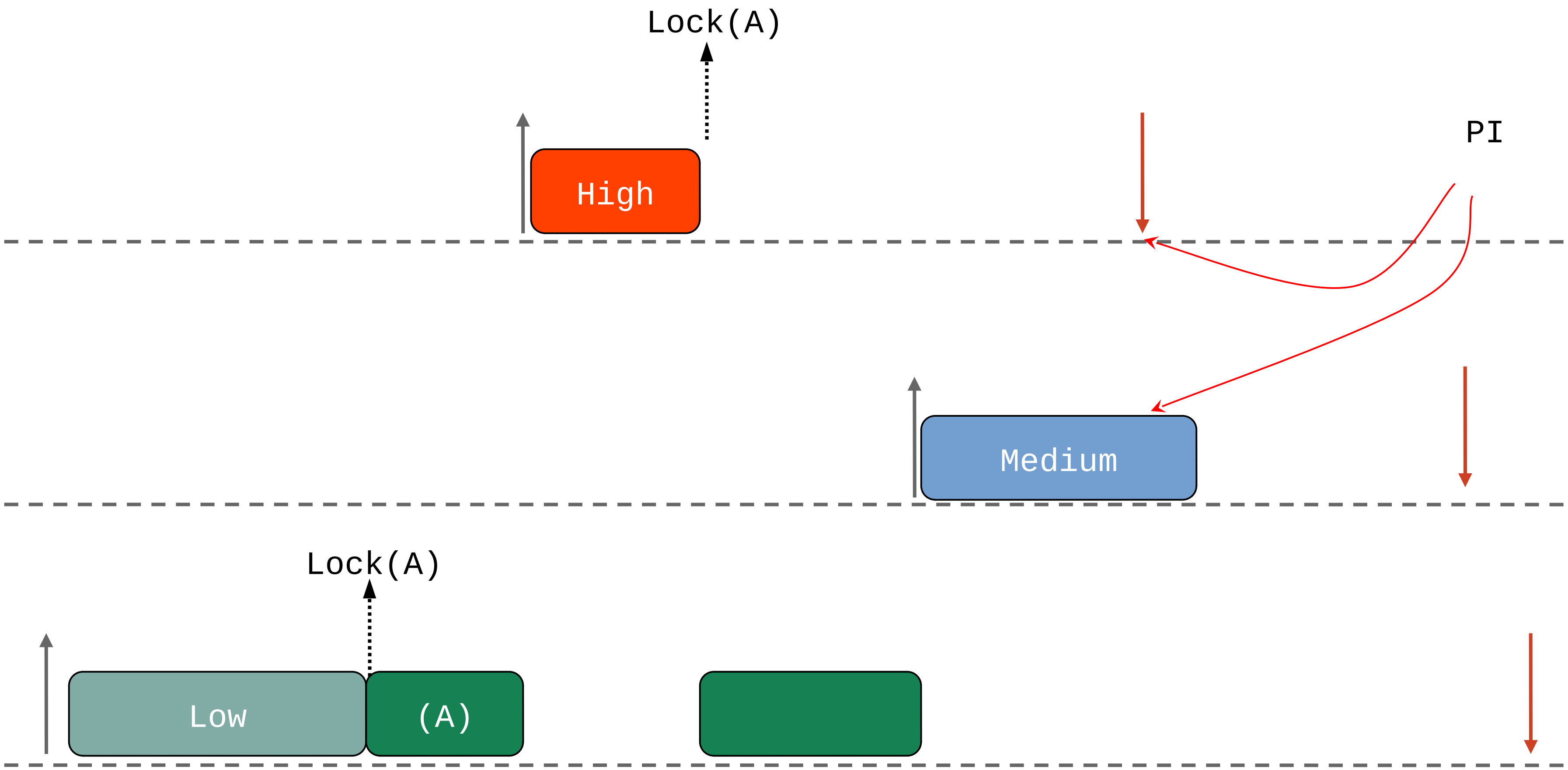
In words

- IRQs become a scheduling problem
- The Preemptive mode:
  - Other than reducing the latency
  - Brings Linux closer to the theoretical models used on schedulers

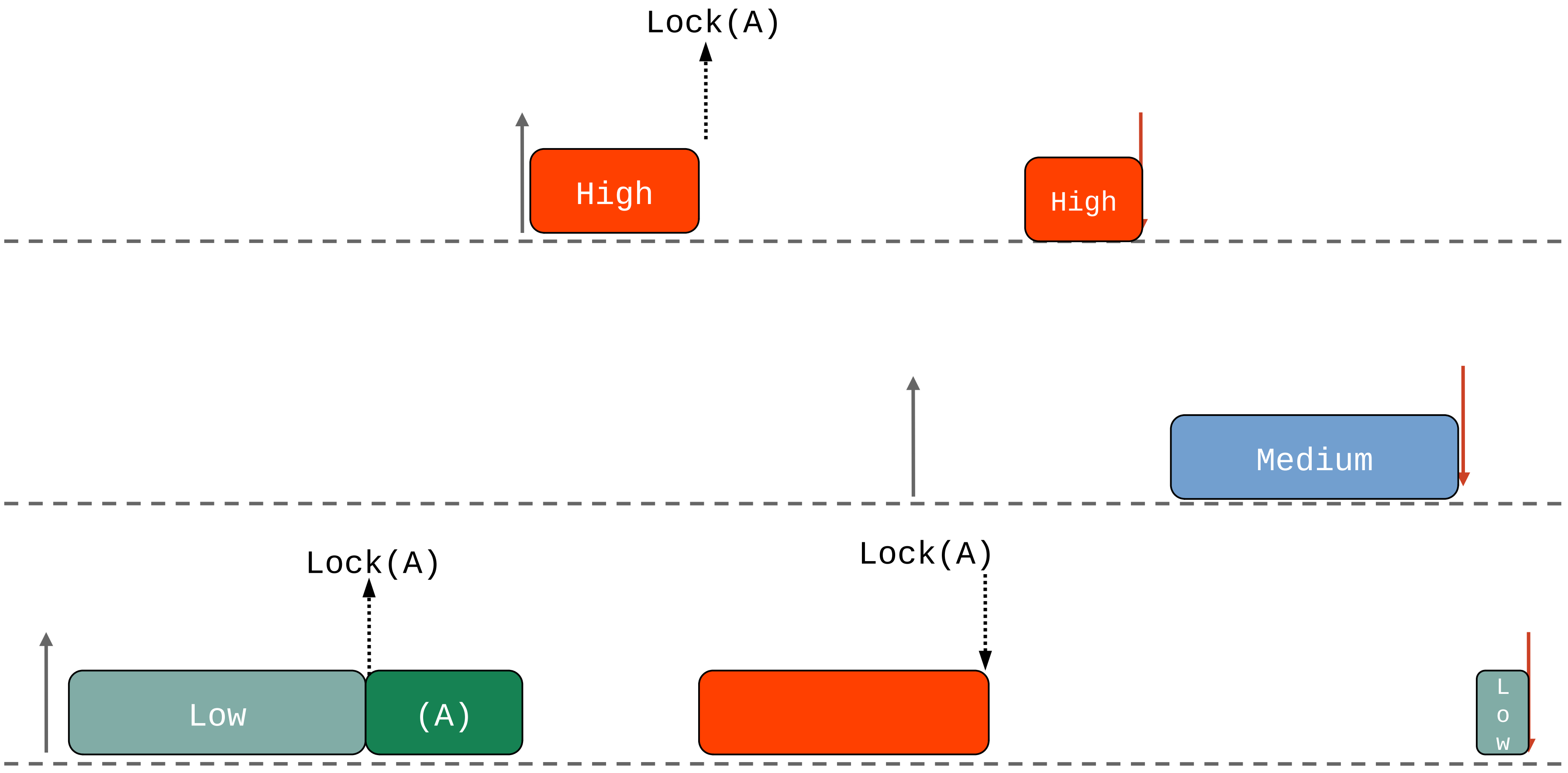


Another good thing:

RM | DL | Locking

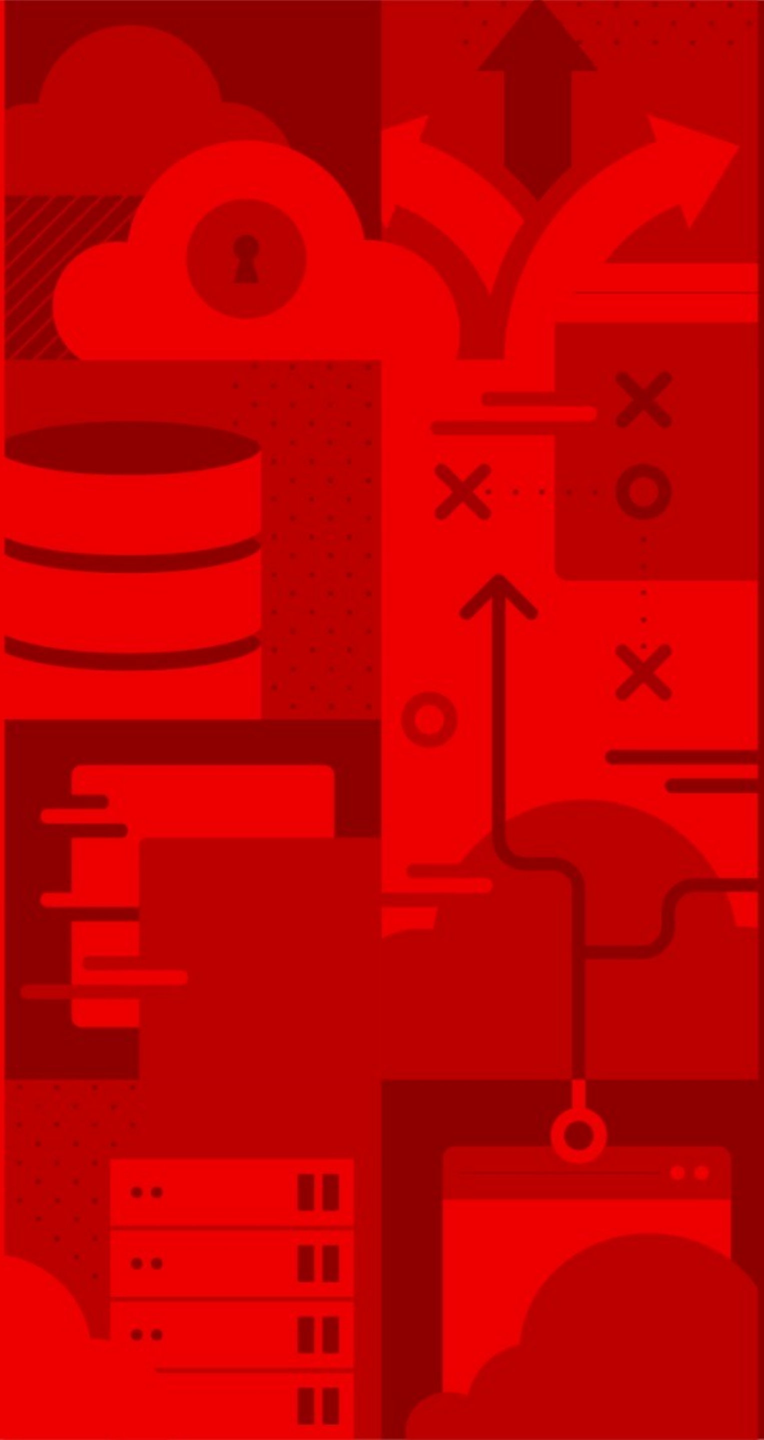


RM | DL | Locking

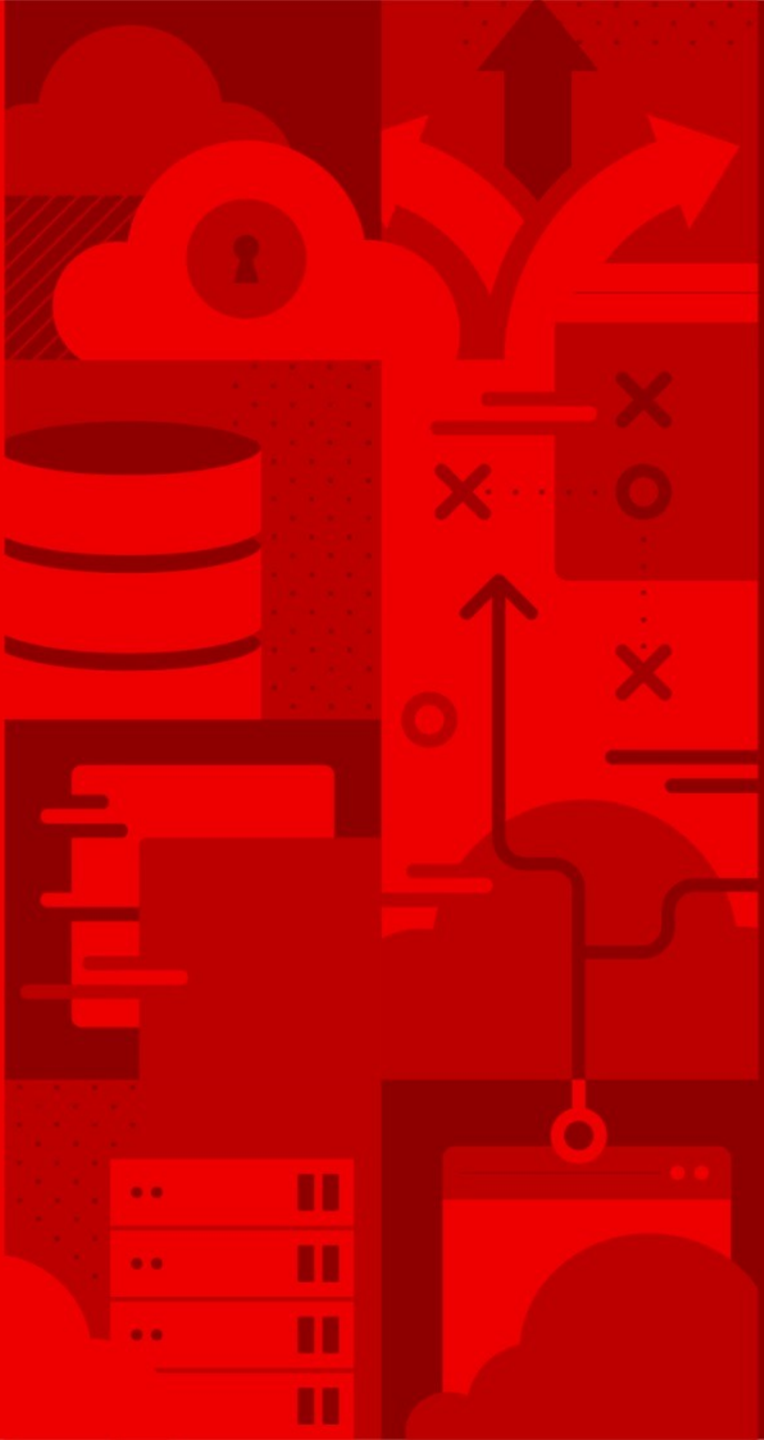


In words

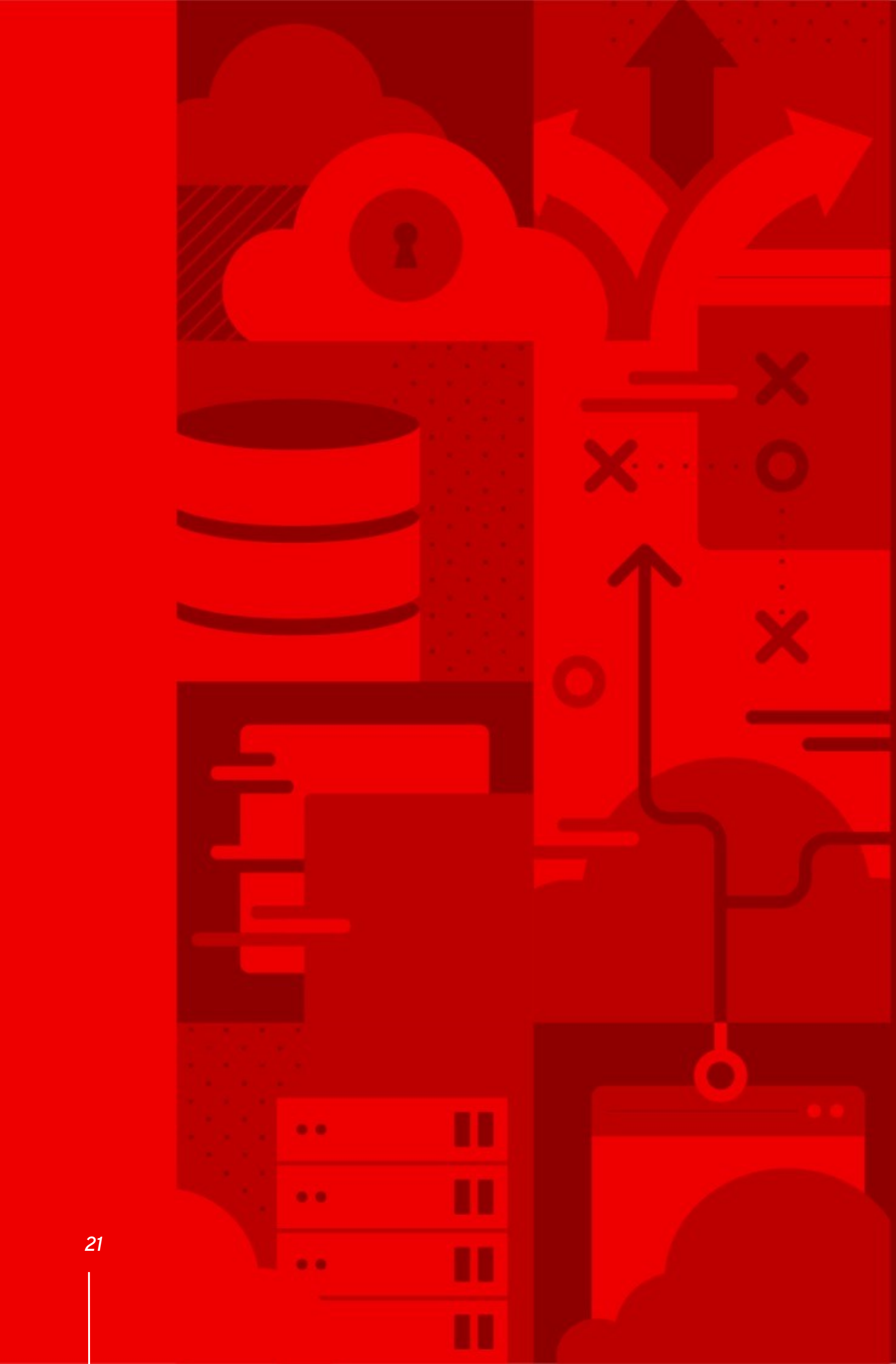
- Informally, the **response time** of a task depends:
  - A little on the IRQ
  - A little on the Latency/scheduling
  - On the locks the thread depends on
  - On it’s own execution time
  - On the interference of higher priority threads (which is OK)



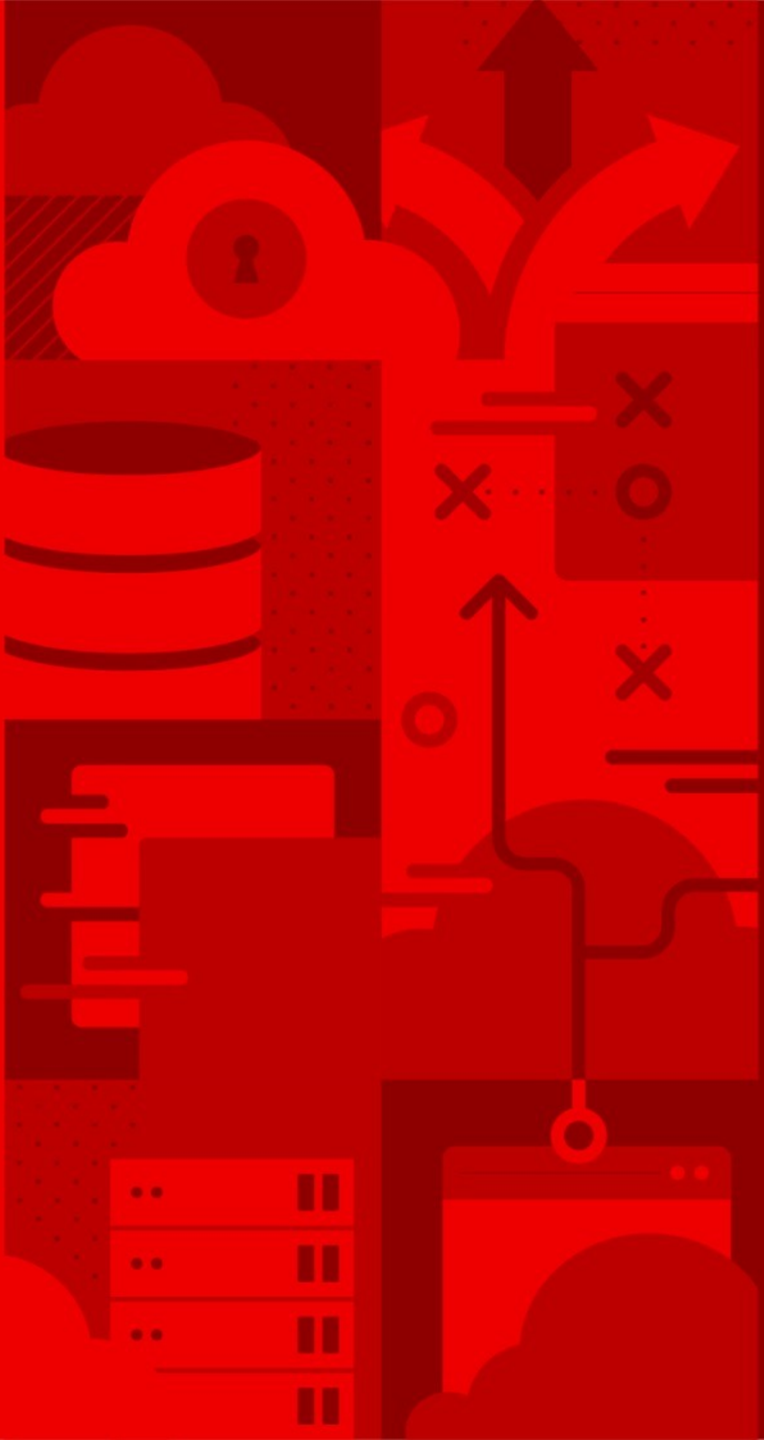
Empirically, turning Linux a possible RTOS!



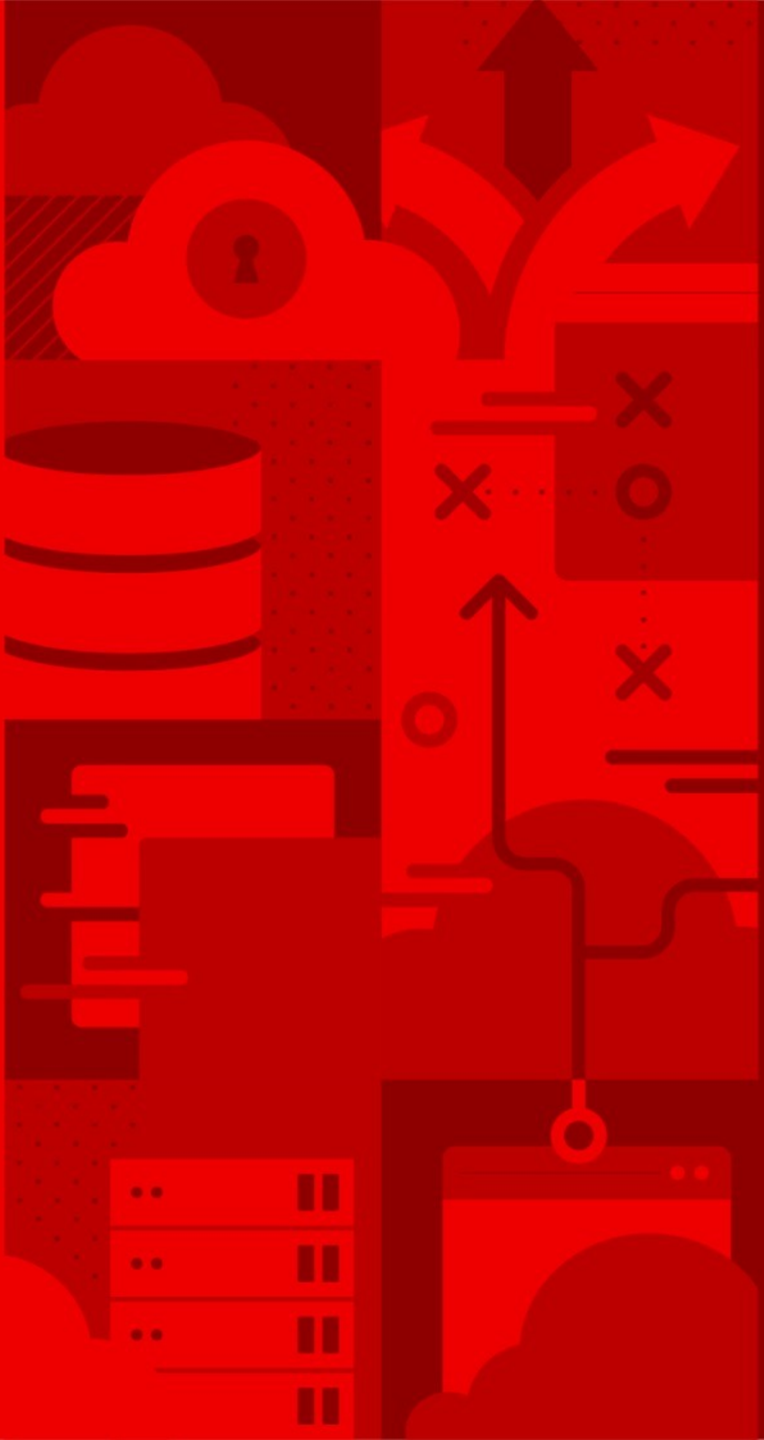
But can we say we are a Hard RTOS?



Let’s do an exercise!



No deadline miss

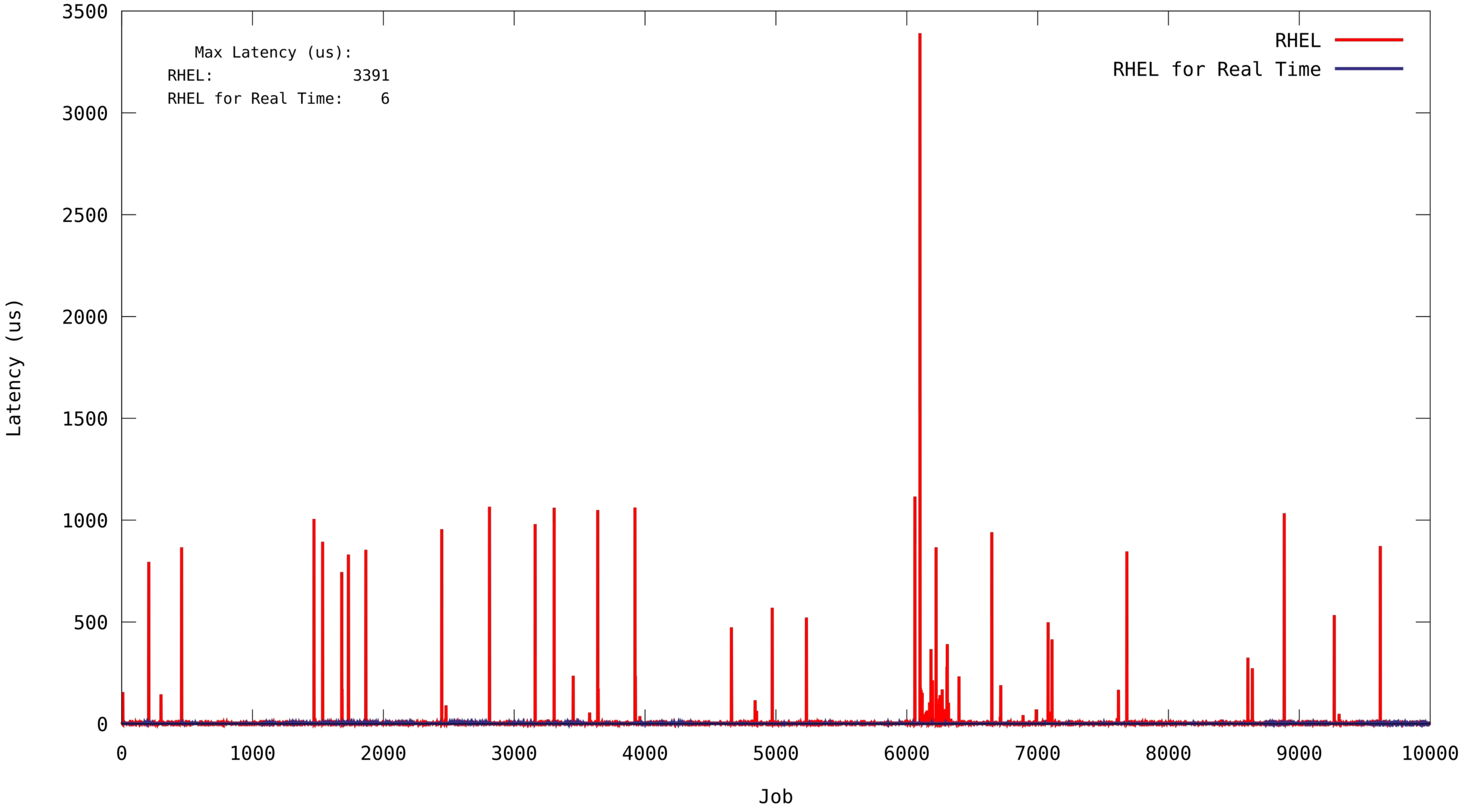


Mean knowing worst case scenarios

Worst Case Execution Time

- There are two classical methods
  - Static methods
    - The code is not executed
    - The control flows are analyzed
      - Execution time is “computed” based on model of the hardware.
  - Measurement methods:
    - Measures the code running in the real hardware.

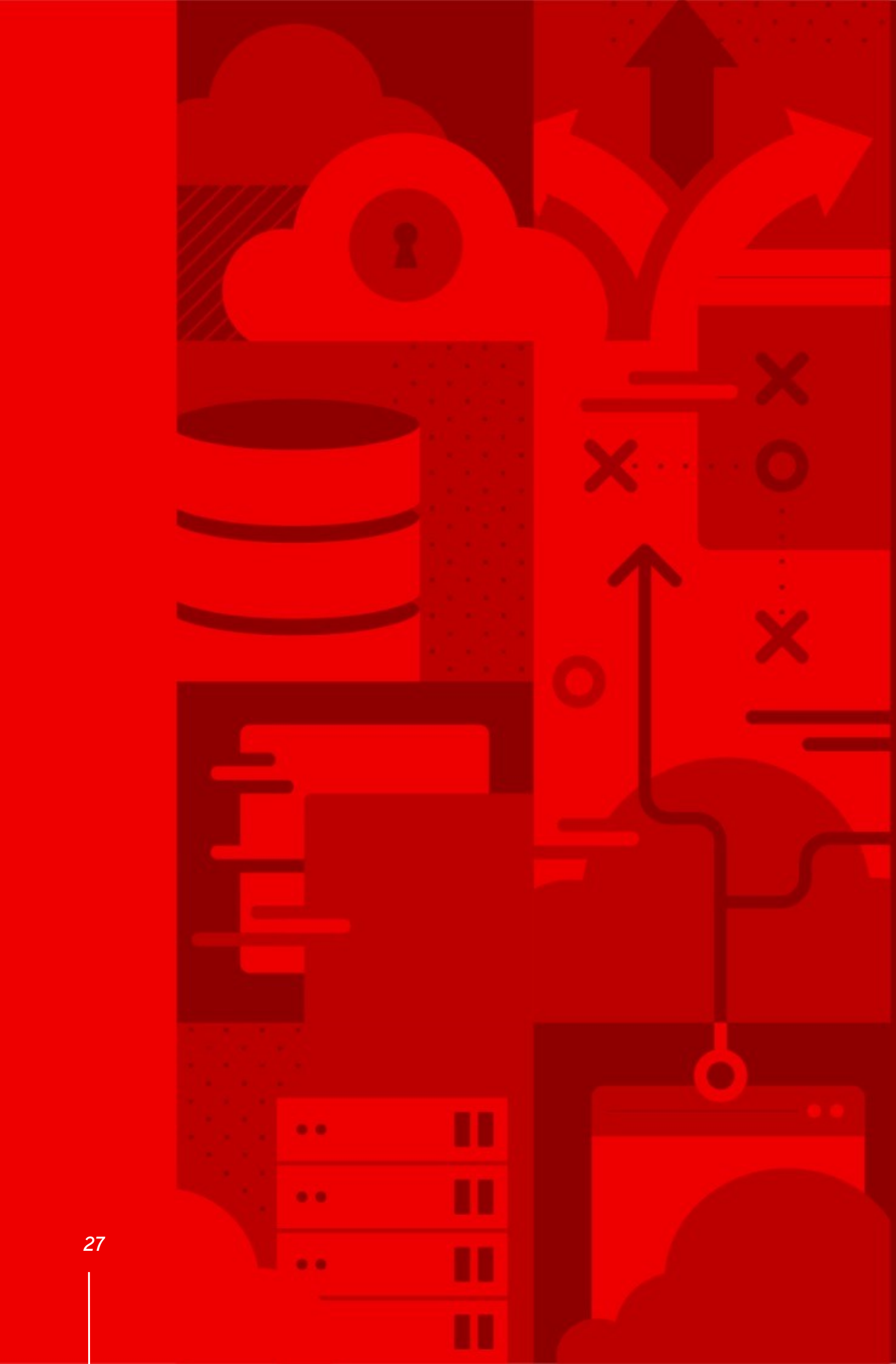
We do measurements!



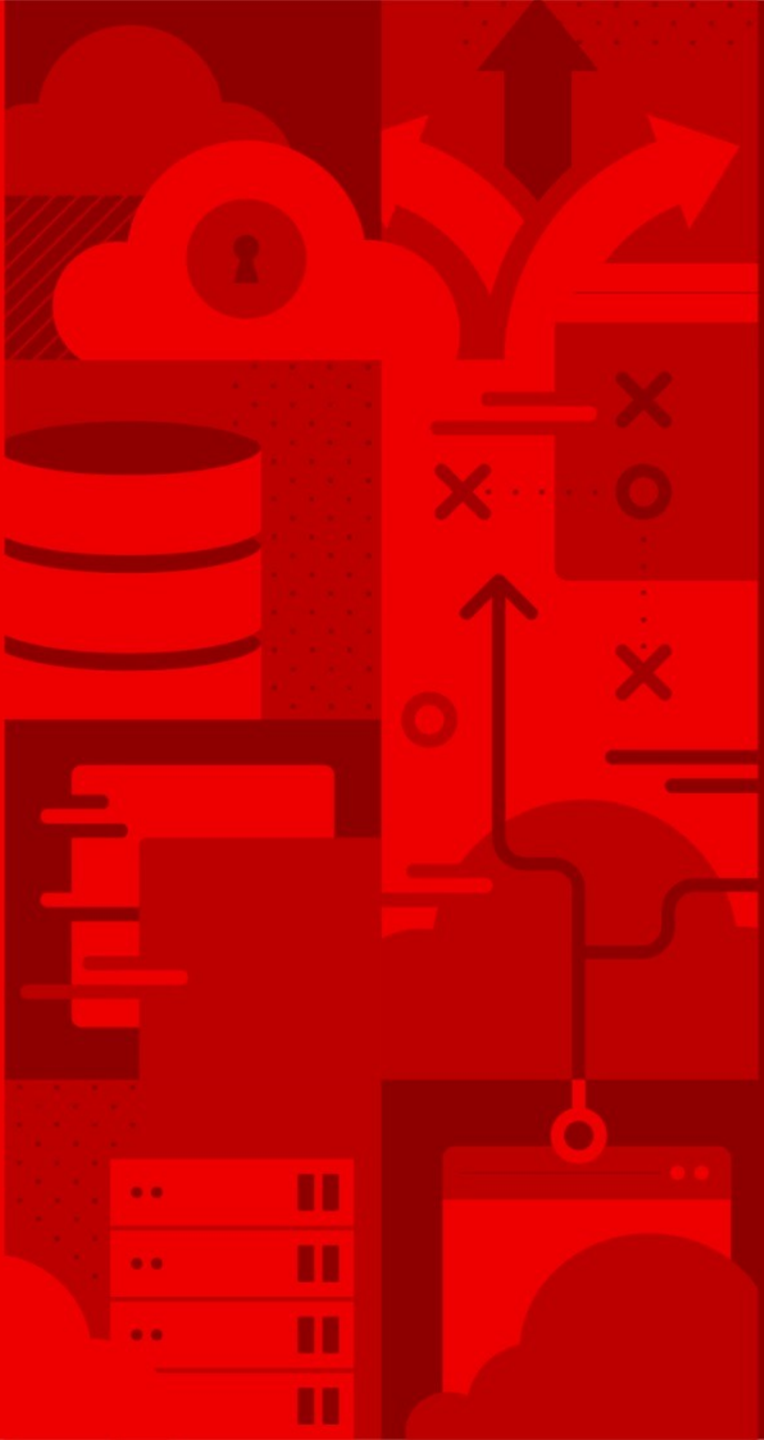
For example:

- DO-178C, **Software Considerations in Airborne Systems and Equipment Certification:**
  - *“Timing measurements by themselves cannot be used without an analysis demonstrating that the worst-case timing would be achieved*. In other words, testing alone is not adequate for demonstrating worst case execution times – some form of analysis is also required.”

  - From: Automating WCET analysis for DO-178B/C, Rapita Systems.



We need more analsys!



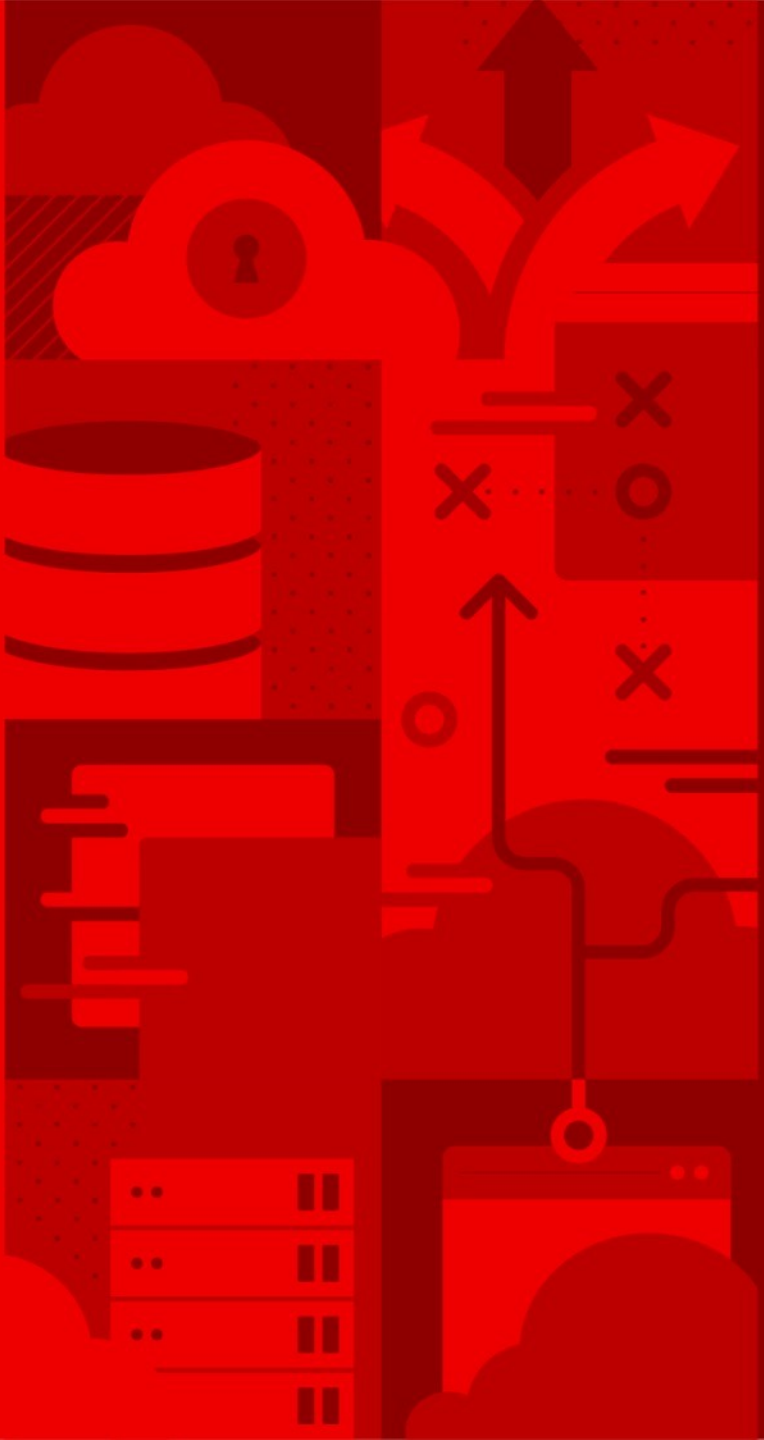
But we have some!

For sched deadline:

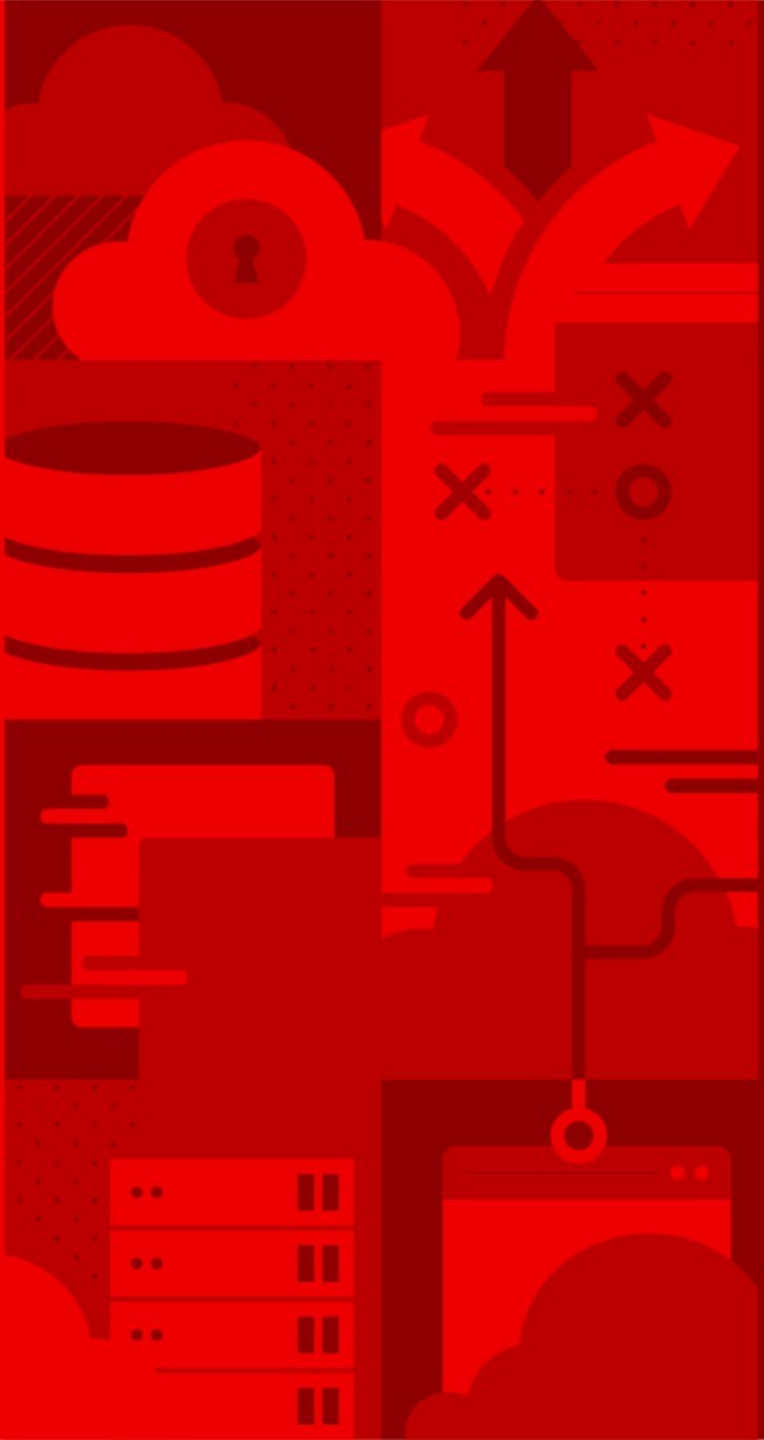
$$U = \sum U_i$$

$$U_i = \frac{C_i}{T_i}$$

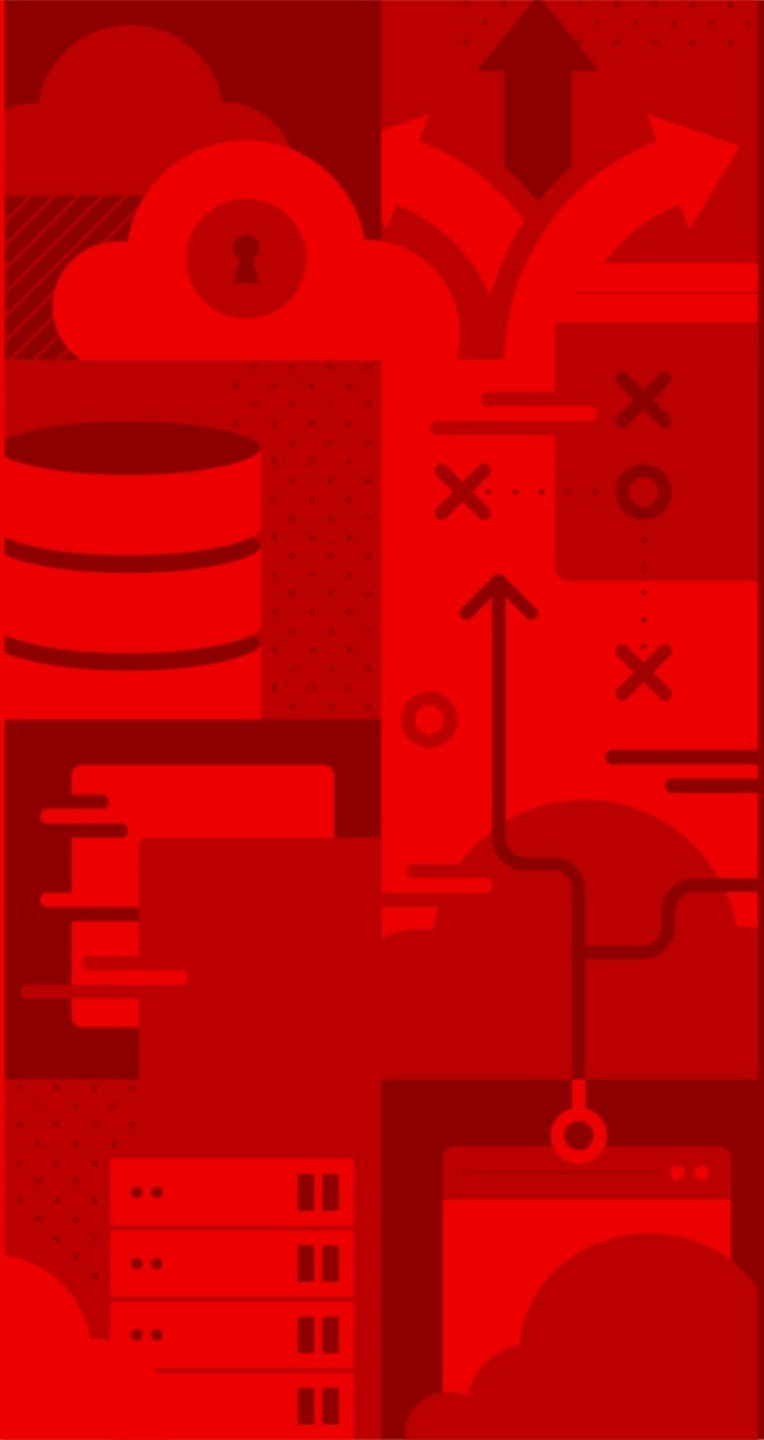
$$is\,schedulable \Leftrightarrow U < 1$$



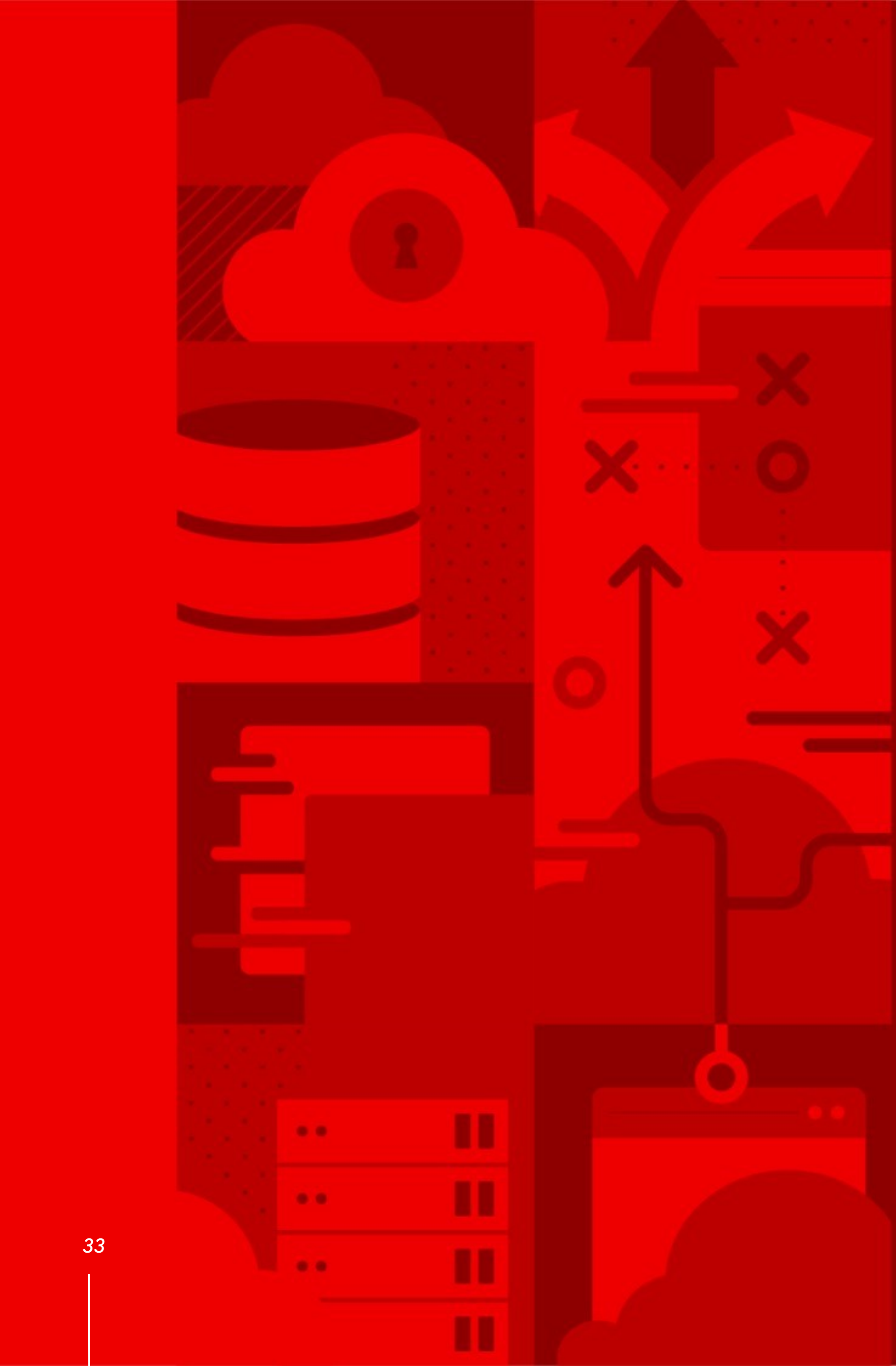
This is good!



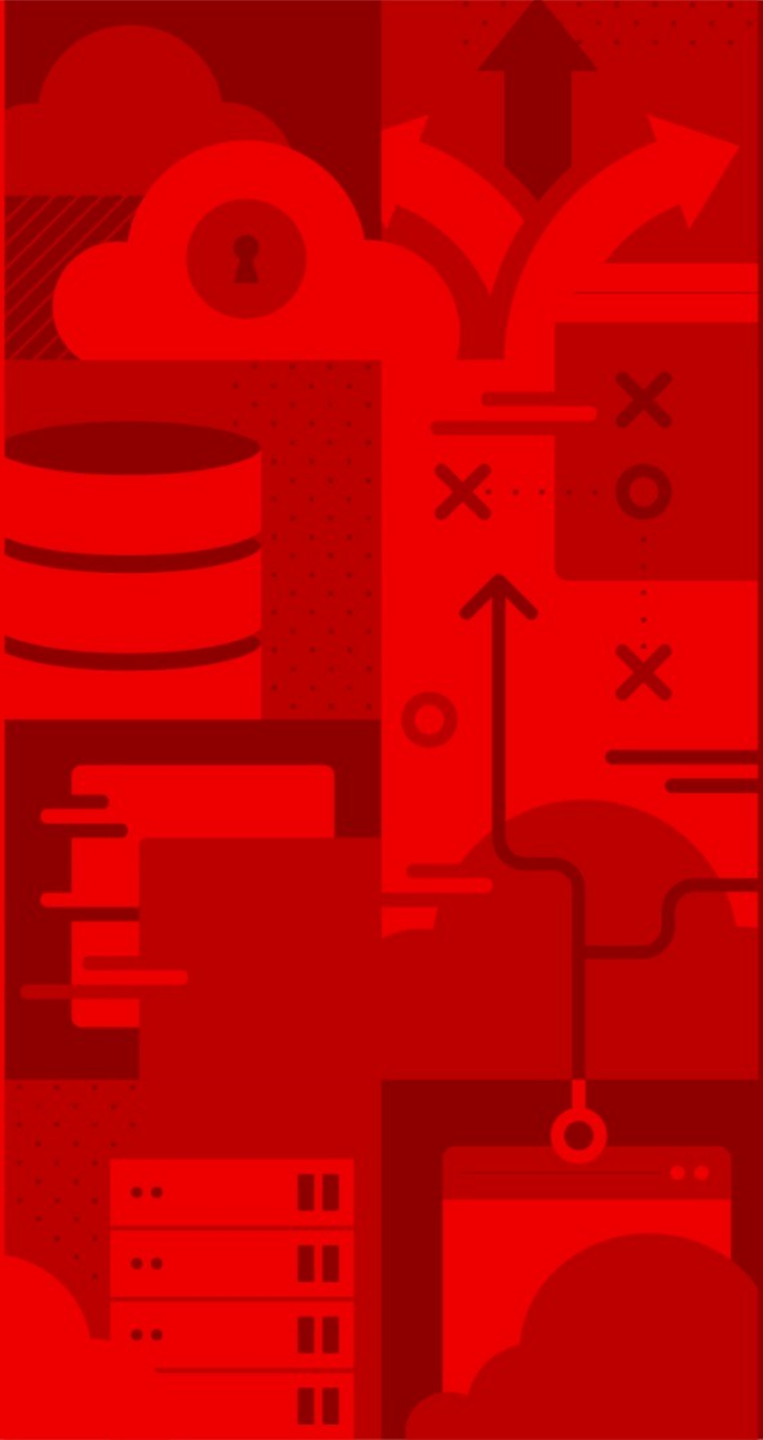
But this is an oversimplification



We need more fined gained analysis



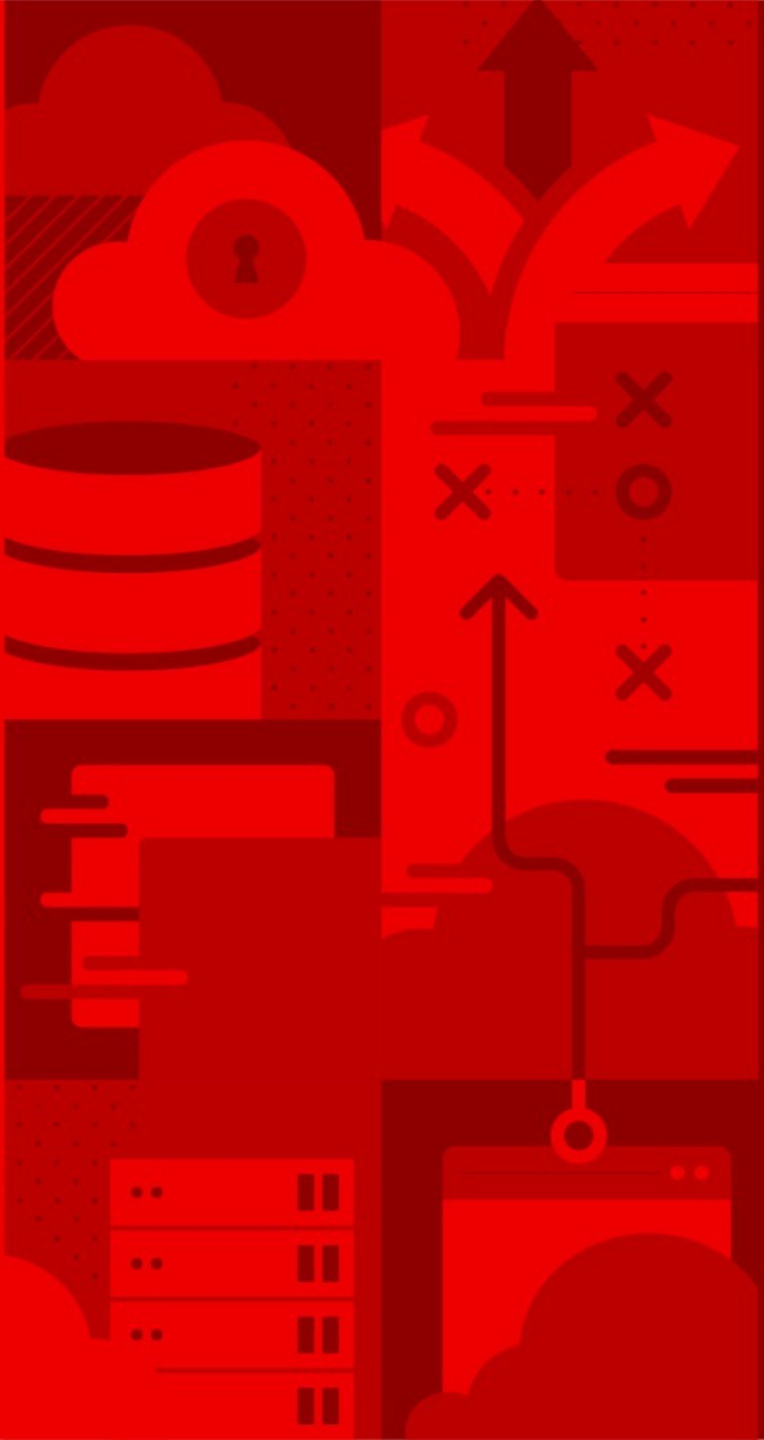
From the sched until the code!



From the scheduler side

Sched

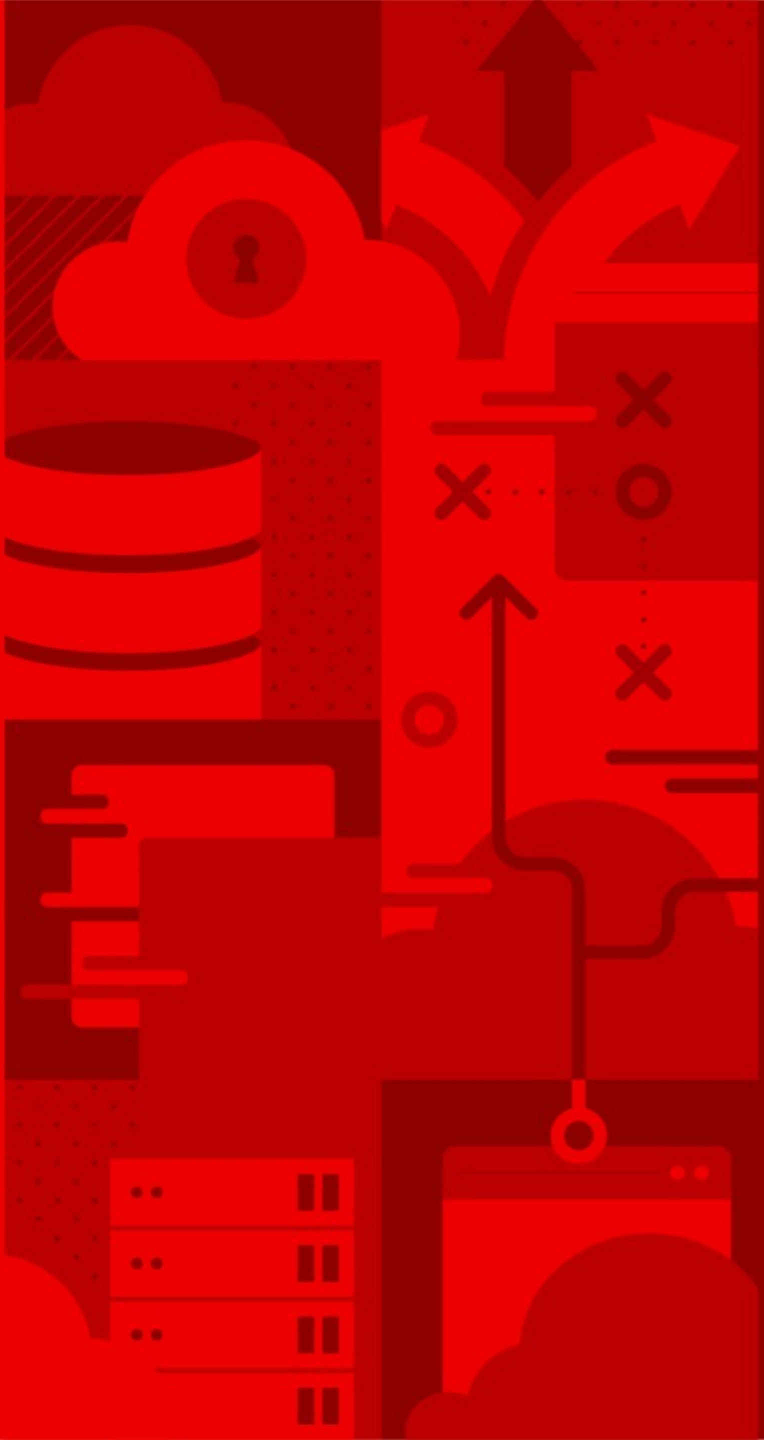
- Being aware of the delays caused by synchronization
- More features
  - Hierarchical
  - Arbitrary affinities
  - Better schedulability
  - We will see more this afternoon



From the locking side

Locking

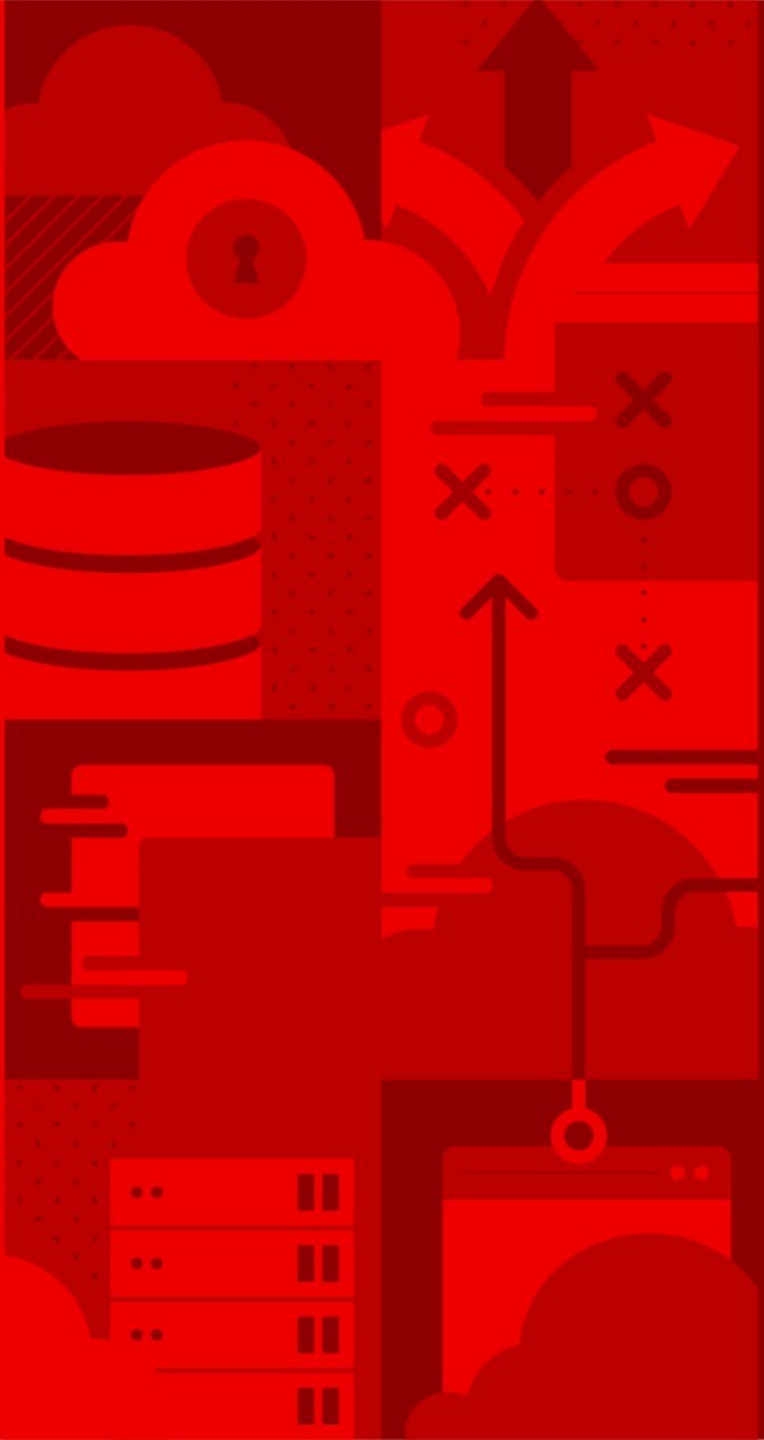
- We have problems with sched deadline
  - Proxy execution
- Other methods that do not have PI support
- Analysis from the locking (nested locking is a problem even in theory)



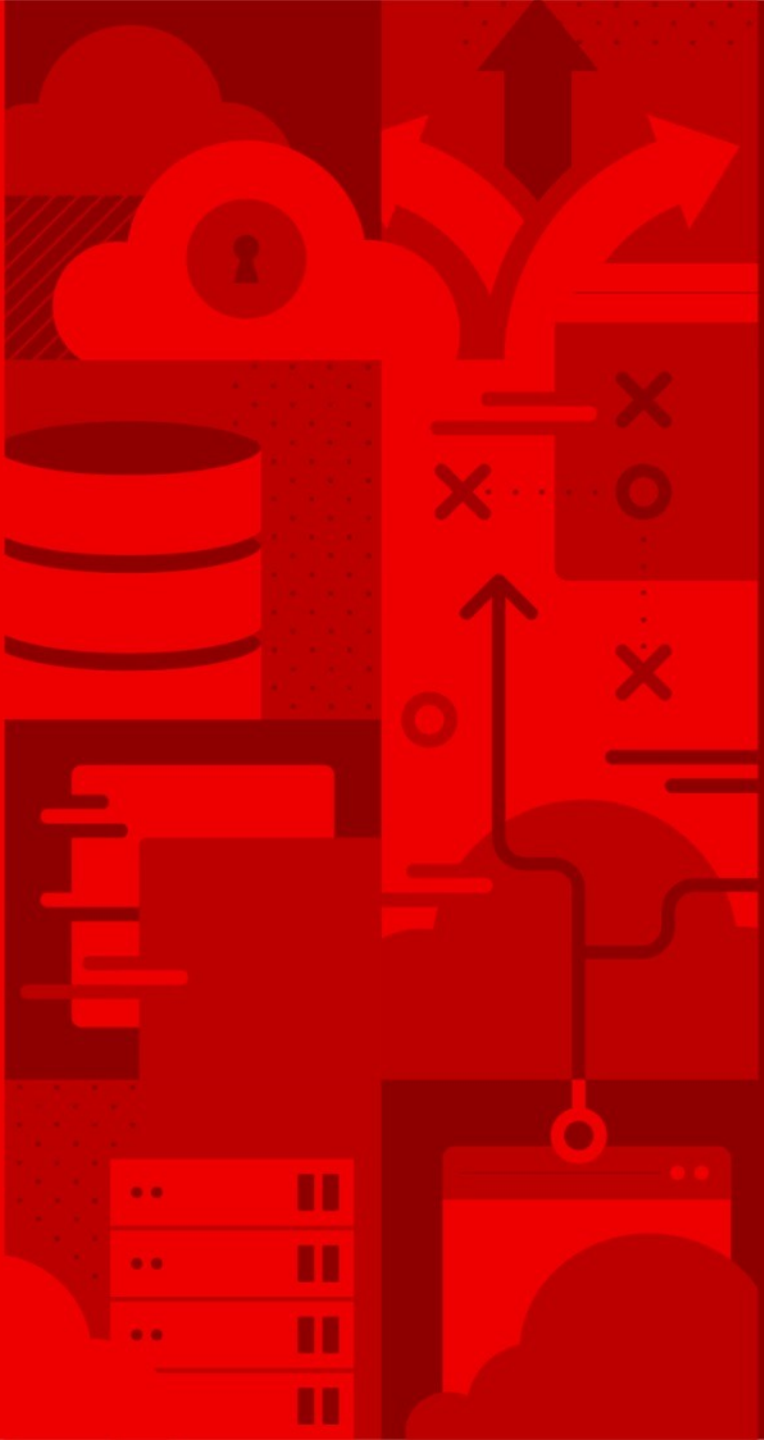
From the code side

WCET

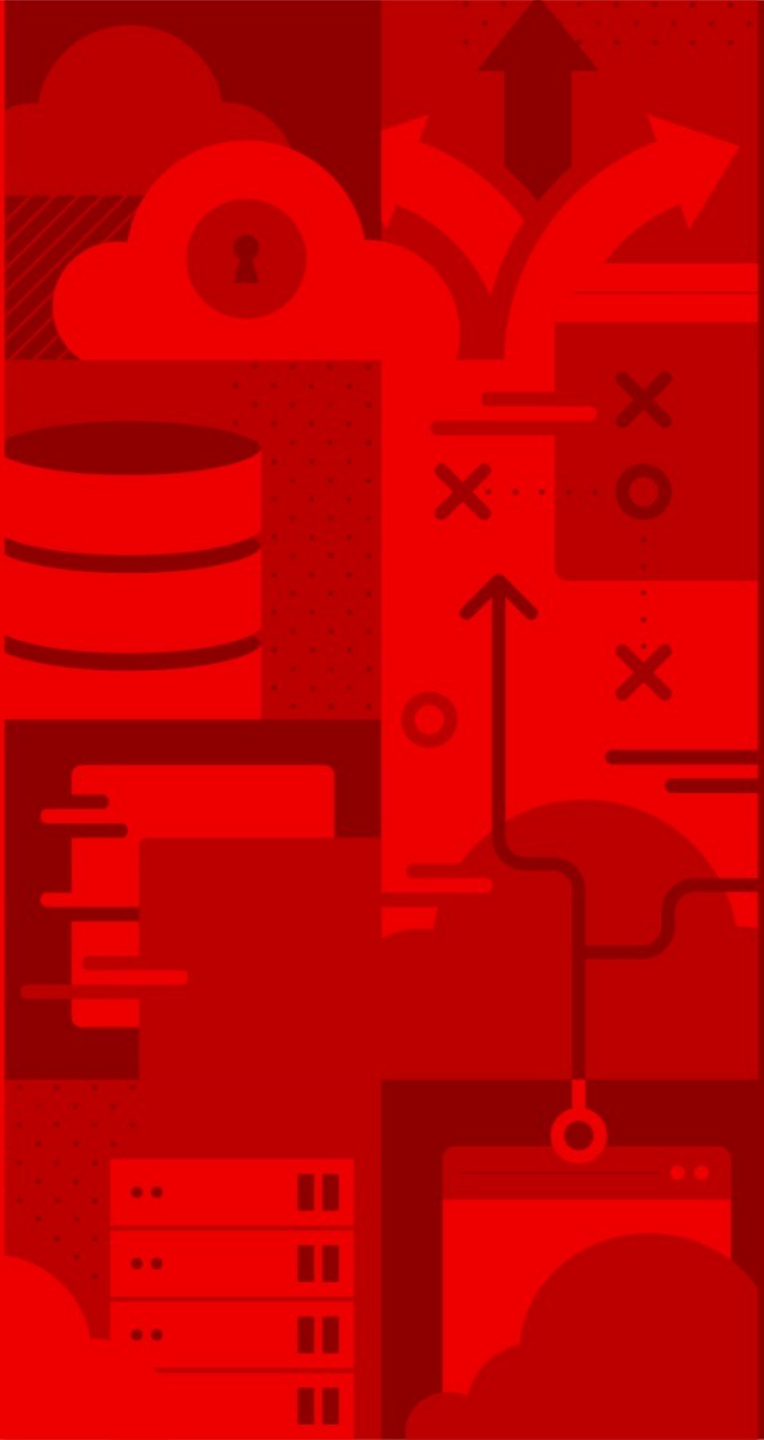
- We currently just observe
- There are many people talking about pWCET
  - But there is a long way to go
  - The WCET companies in the field have pWCET but not for complex system like Linux



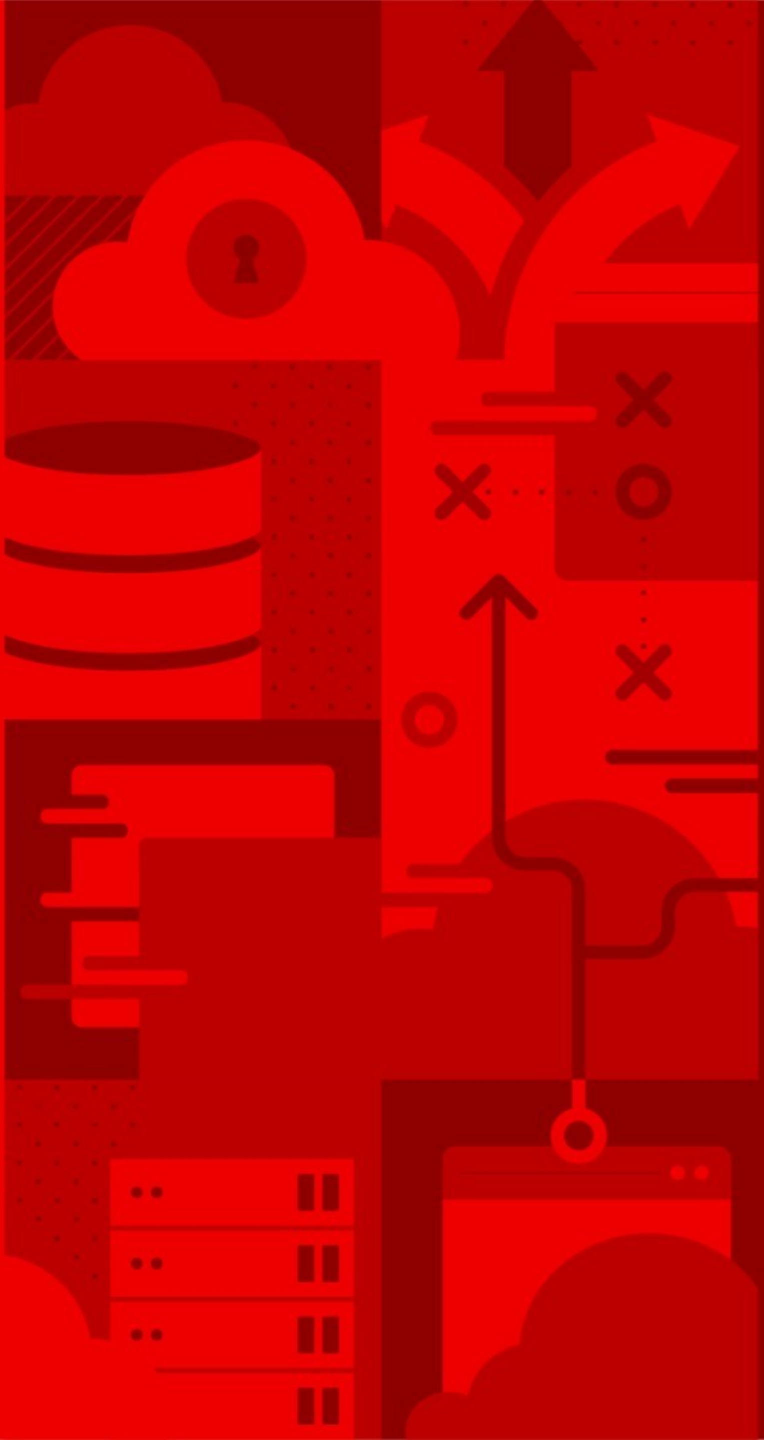
There is a lot of work to do!



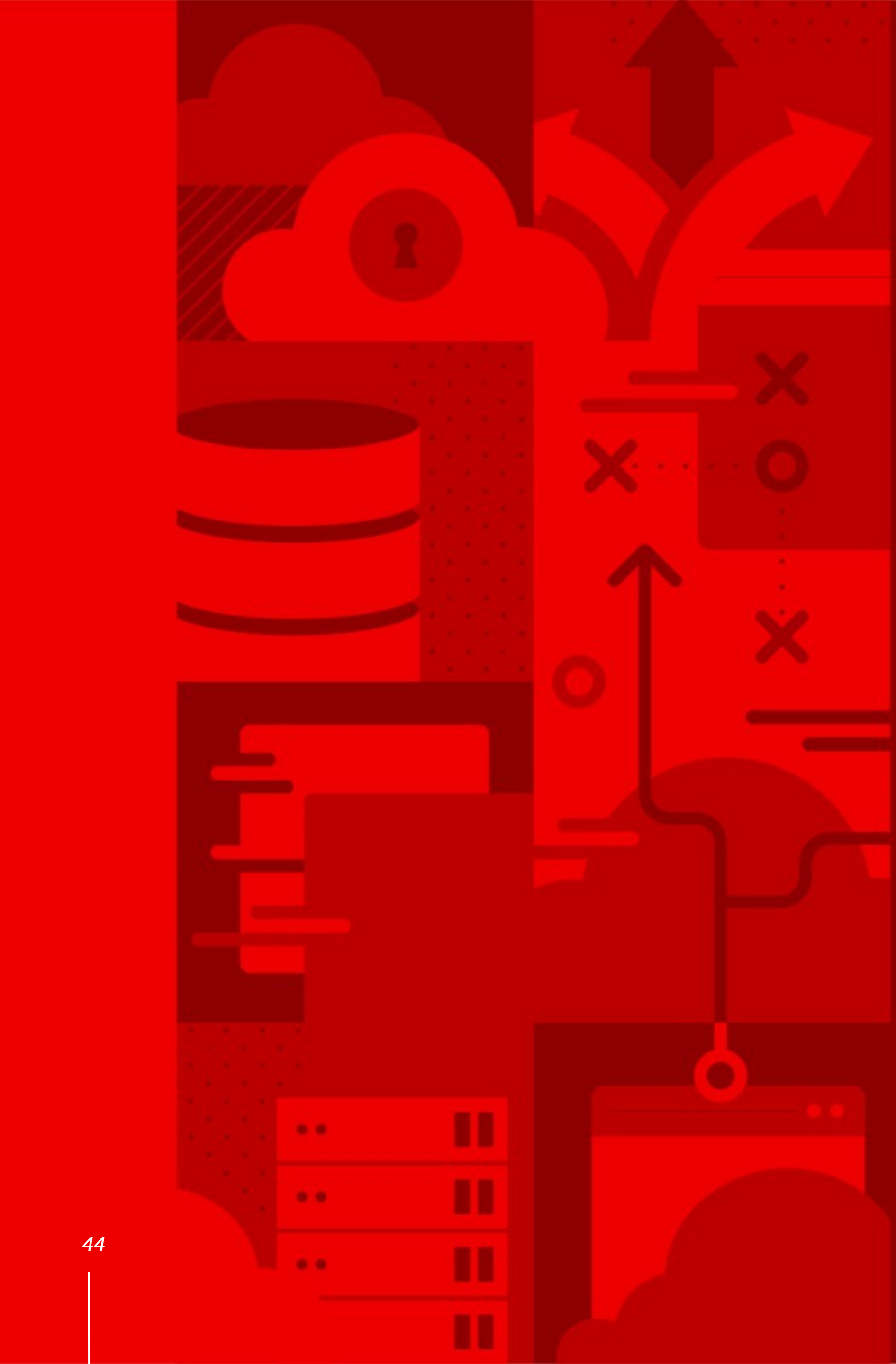
So Linux is not a Hard RTOS



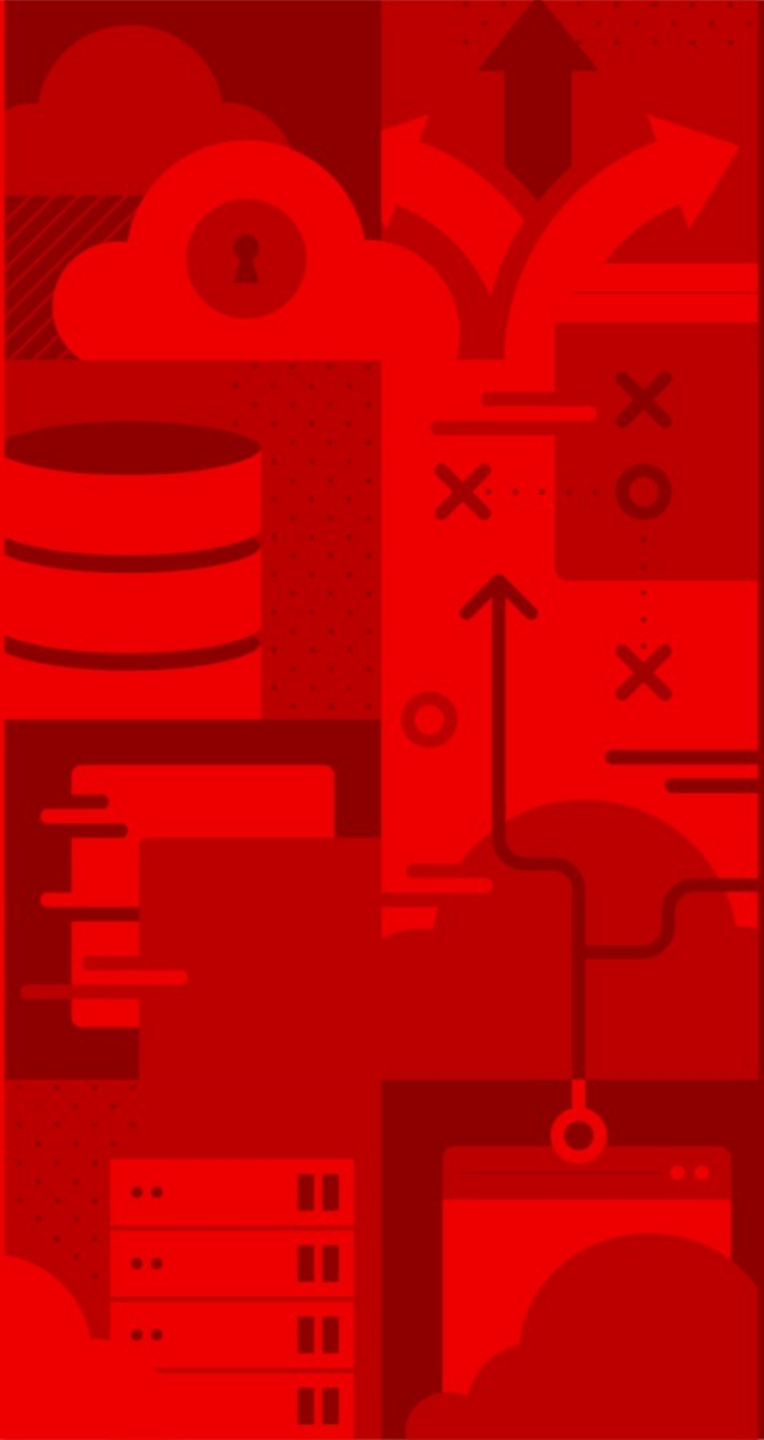
But we are touching the edges of RT theory



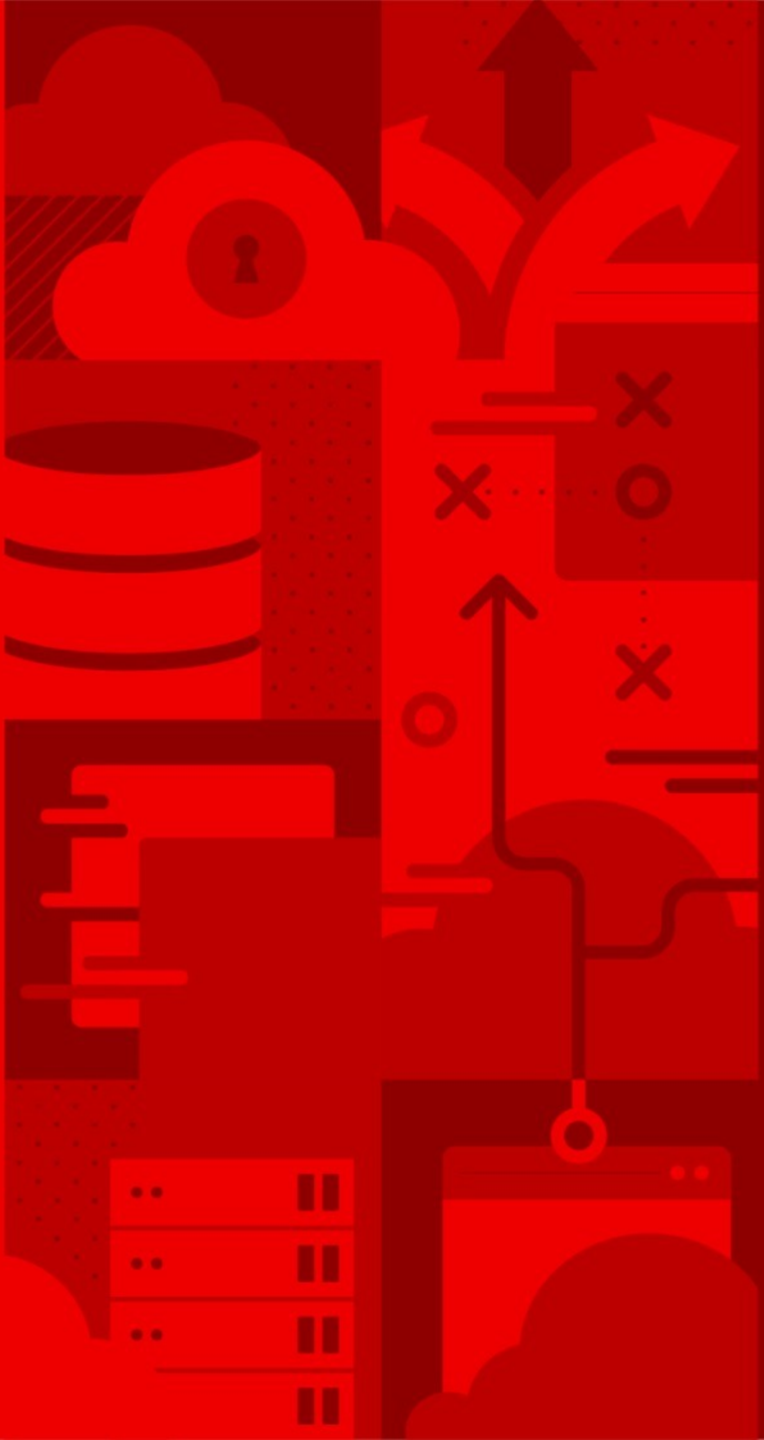
And the PREEMPT RT
Enable that!



Job Security



That is it



Thoughts?