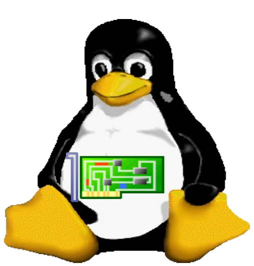
SMP/Linux Real-time Analysis & Enhancements

Jim Huang (黃敬群) <jserv.tw@gmail.com>

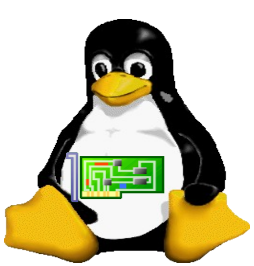


Real-Time System

- Deadline

- Hard Real-time
  - Meet deadline Deterministically
  - Guaranteed worst case

- Soft Real-time
  - Best Effort

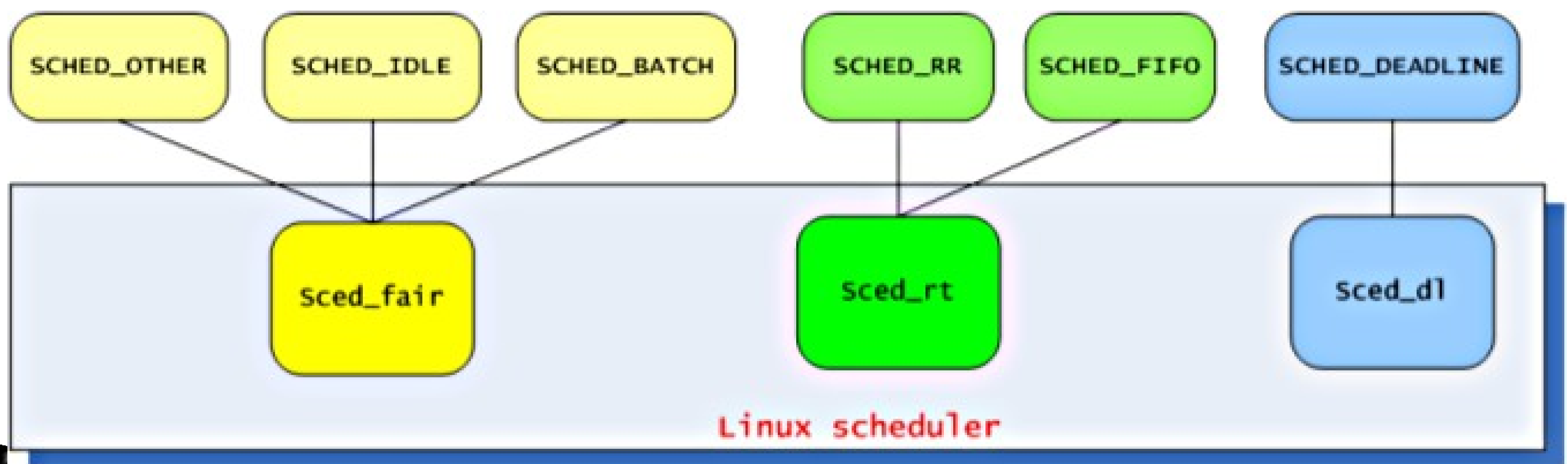


Real-time System

- Critical Task
  - Type
    - Periodic Task
    - Non Periodic Task
  - Latency
    - Preemption
    - Interrupt
    - Critical Section
    - Others

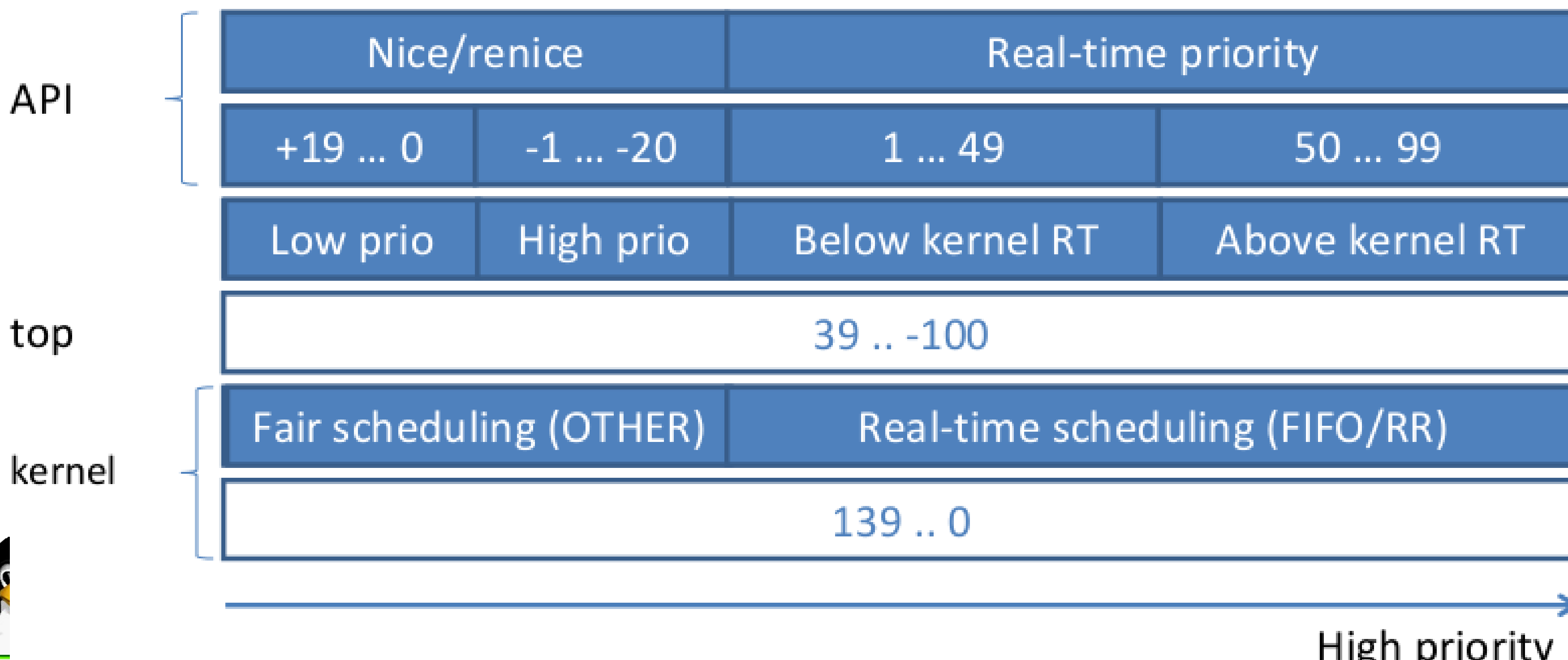
Real-time Task in Linux

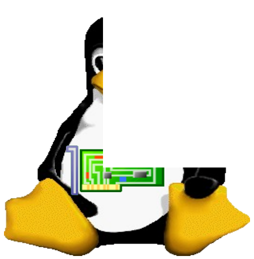
- Schedule Policy
  - SCHED_FIFO
  - SCHED_RR



Real-time Priority of Linux

- Nice value: -20 ~ +19 (19 is lowest)
- Real-time Priority: 0 ~ 99
  - higher value with higher priority





Real-time Task

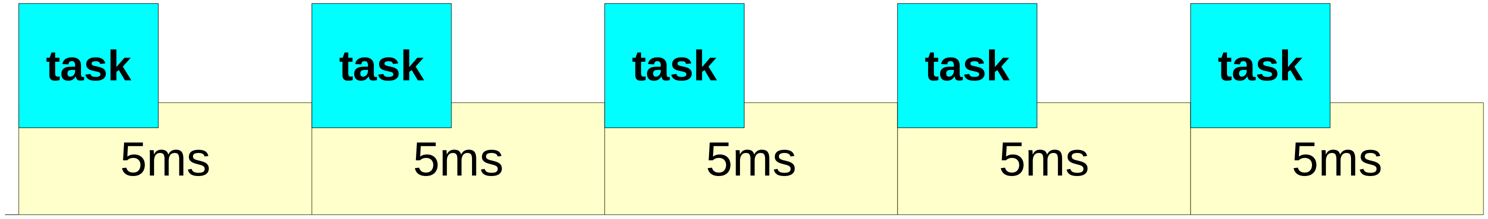
- Periodic Real-time Task
  - Precise timer is needed

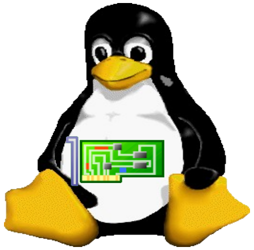
- Non periodic Real-time task
  - Triggered by Interrupt

Periodic real-time task

- Periodic Real-time task

e.g.

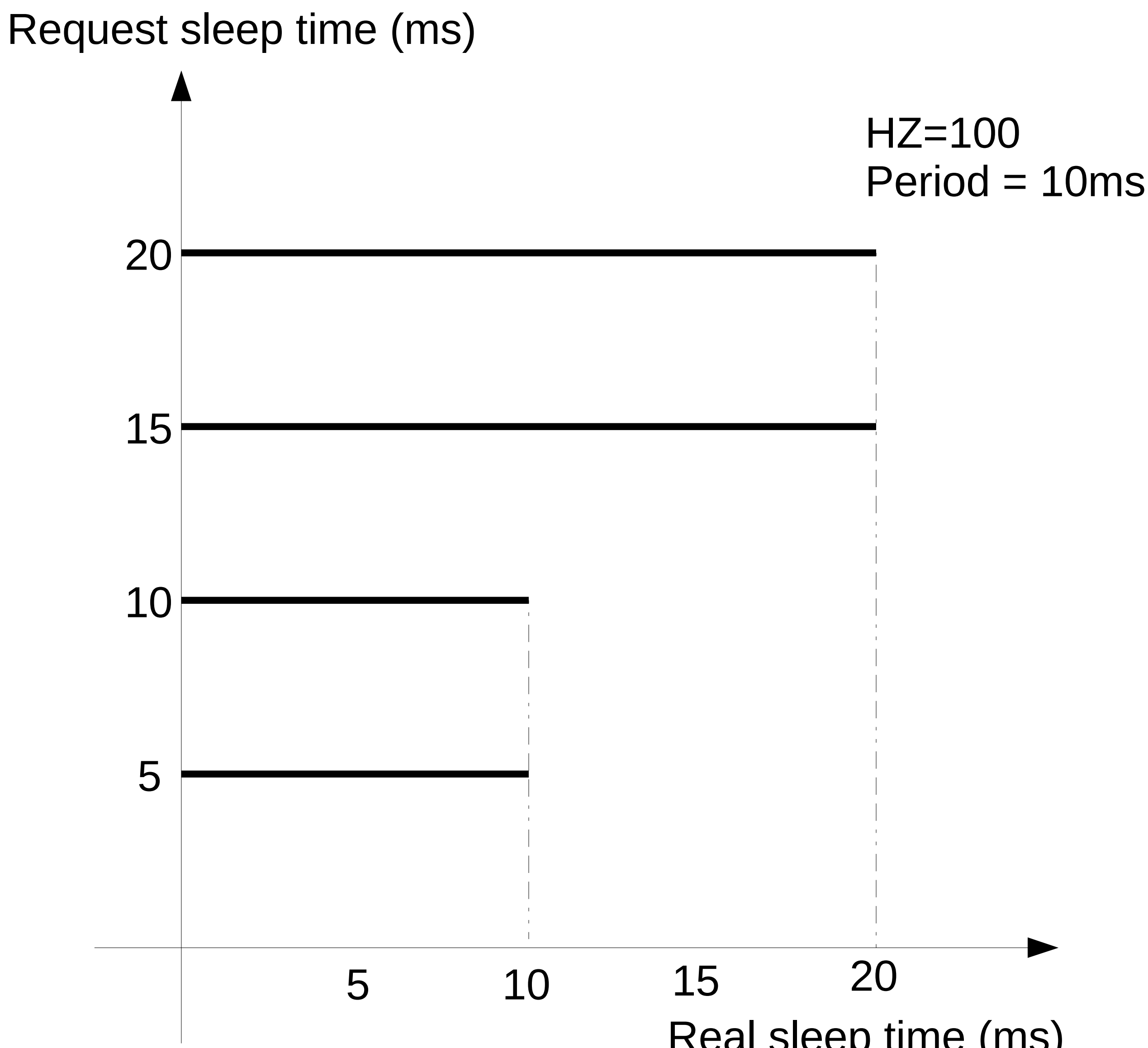


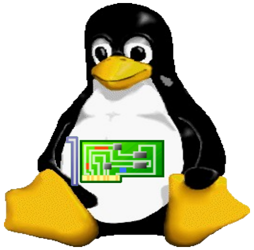


Linux Periodic Timer

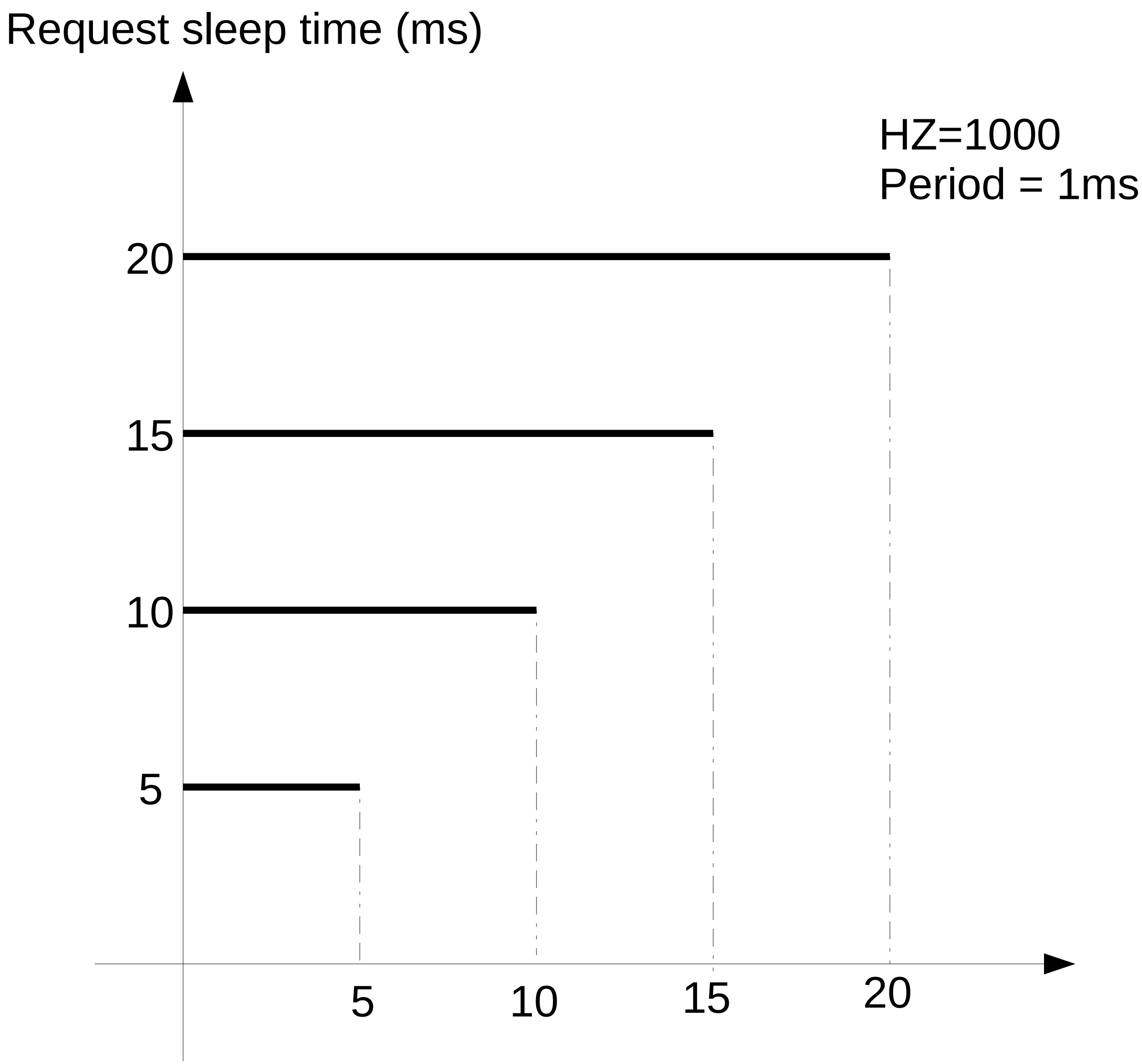
- Tick in Linux
  - jiffies
  - HZ
    - Number of tick of one second
    - Configurable
  - Timer Resolution

Periodic Timer Resolution





Periodic Timer Resolution



Periodic Timer Issue

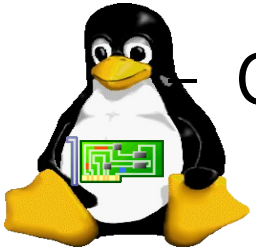
- High resolution
  - High overhead
  - Max HZ value : 1000
    - Min time granularity: 1ms

- Bound to jiffies

High Resolution Timer

- hrtimer
- Time Source
  - Hardware clock event
  - All timer events are One shot events
  - Recent expired timer will be triggered

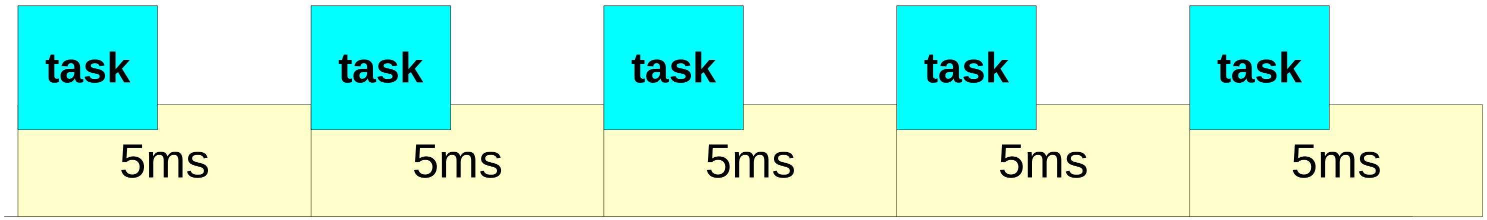
- e.g. Tick with hrtimer support
  - tick_sched_timer
  - Periodic timer event by one shot hrtimer event
  - Current tick event will register next tick event



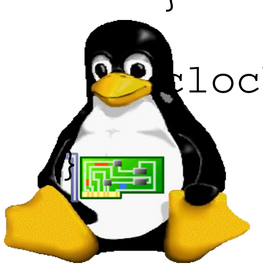
Periodic Real-time Task

- Periodic Real-time Task with hrtimer support
  - User Space
    - clock_nanosleep
    - clock source: CLOCK_MONOTONIC
  - Kernel Space
    - hrtimer_init
    - Triggered by Clock Event
    - One shot event

Periodic Real-time Task




```
const int NSEC_IN_SEC = 10000000001, INTERVAL = 50000001;
clock_gettime(CLOCK_MONOTONIC, &timeout);
While (1) {
    task_work(&some_data);
    if (timeout.tv_nsec >= NSEC_IN_SEC) {
        timeout.tv_nsec -= NSEC_IN_SEC;
        timeout.tv_sec++;
    }
    clock_nanosleep(CLOCK_MONOTONIC, TIMER_ABSTIME, &timeout, NULL);
```



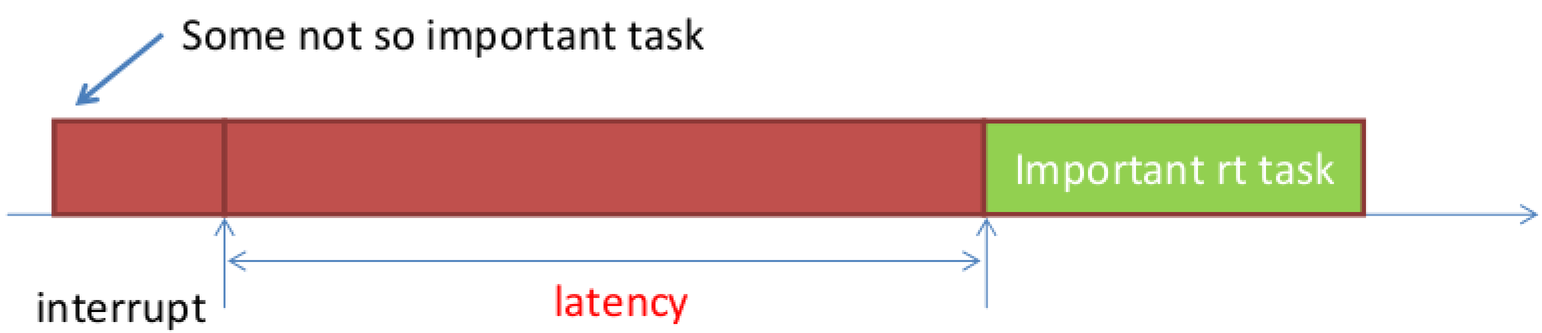
Real-time Task with hrtimer

- All tasks are triggered by Interrupt

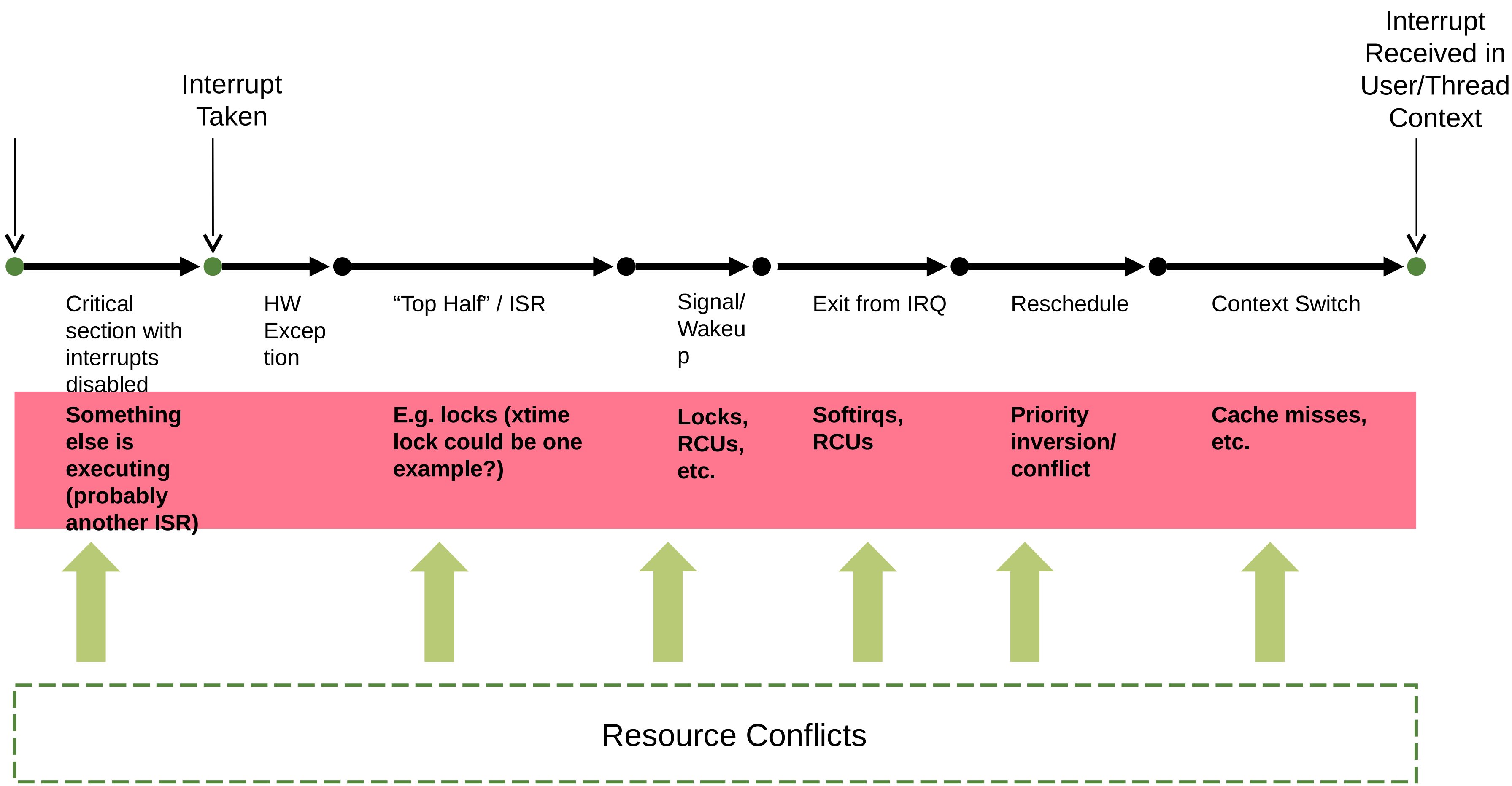
Latency in Linux

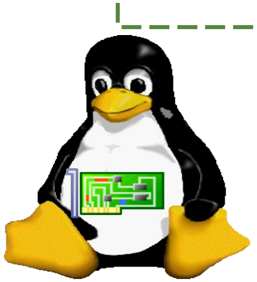
Latency

- Time interval from event trigger till handler task react this event
  - Event: Interrupt
  - Handler: Real-time Task



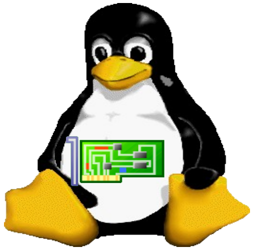
From Interrupt to Received





Latency results from...

- **Preemption**
- Critical Section
- Interrupt

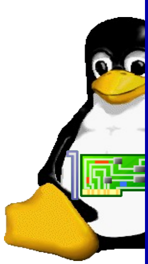


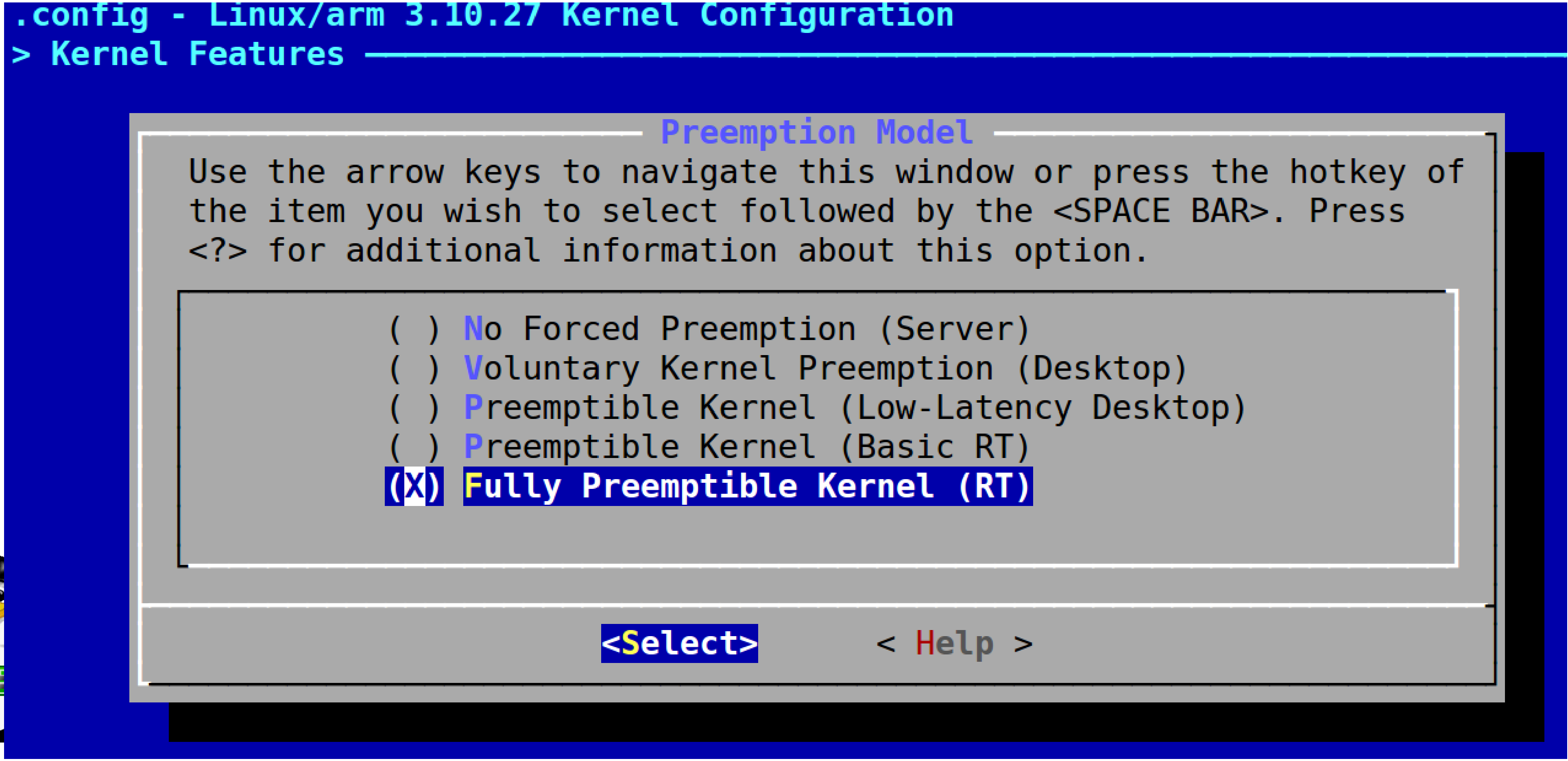
Preemption

- Re-schedule when high priority task is ready
- Increase responsibility
- Decrease throughput

Preemption in Linux

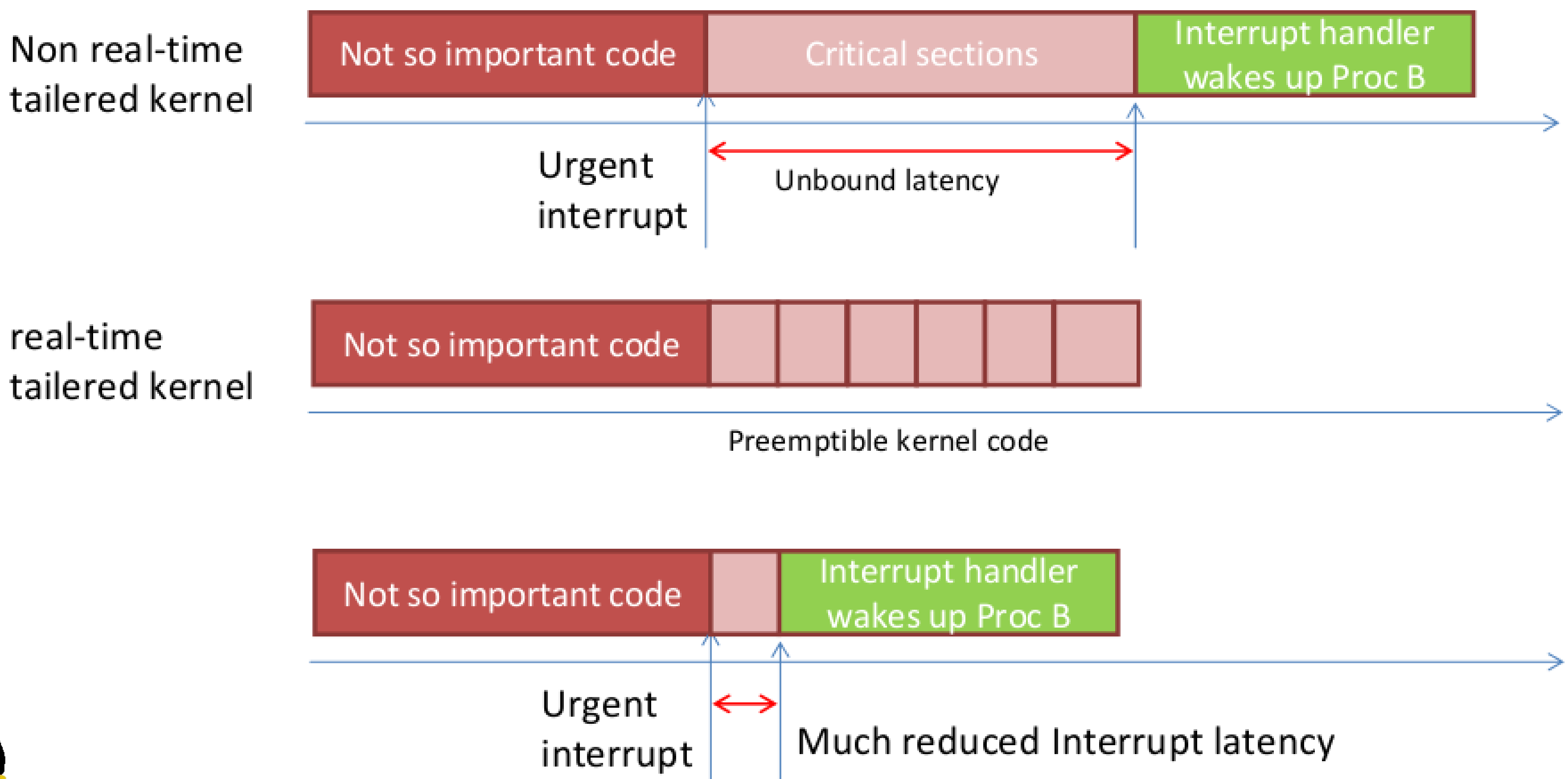
- Preempt configurations
  - NONE, Voluntary, Basic RT, RT_FULL

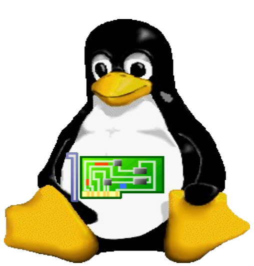




Toward complete preemption

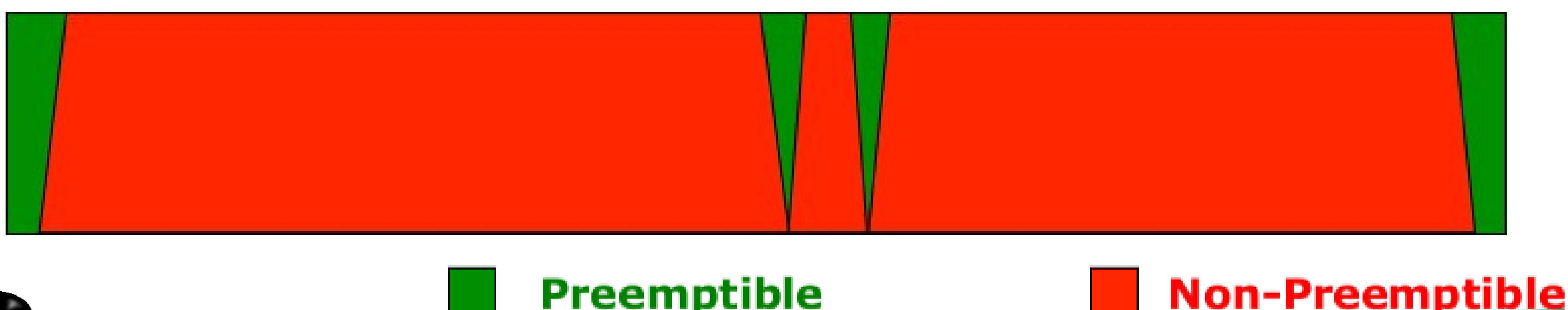
- Most important aspects of Real-time
  - Controlling latency by allowing kernel to be preemptible everywhere





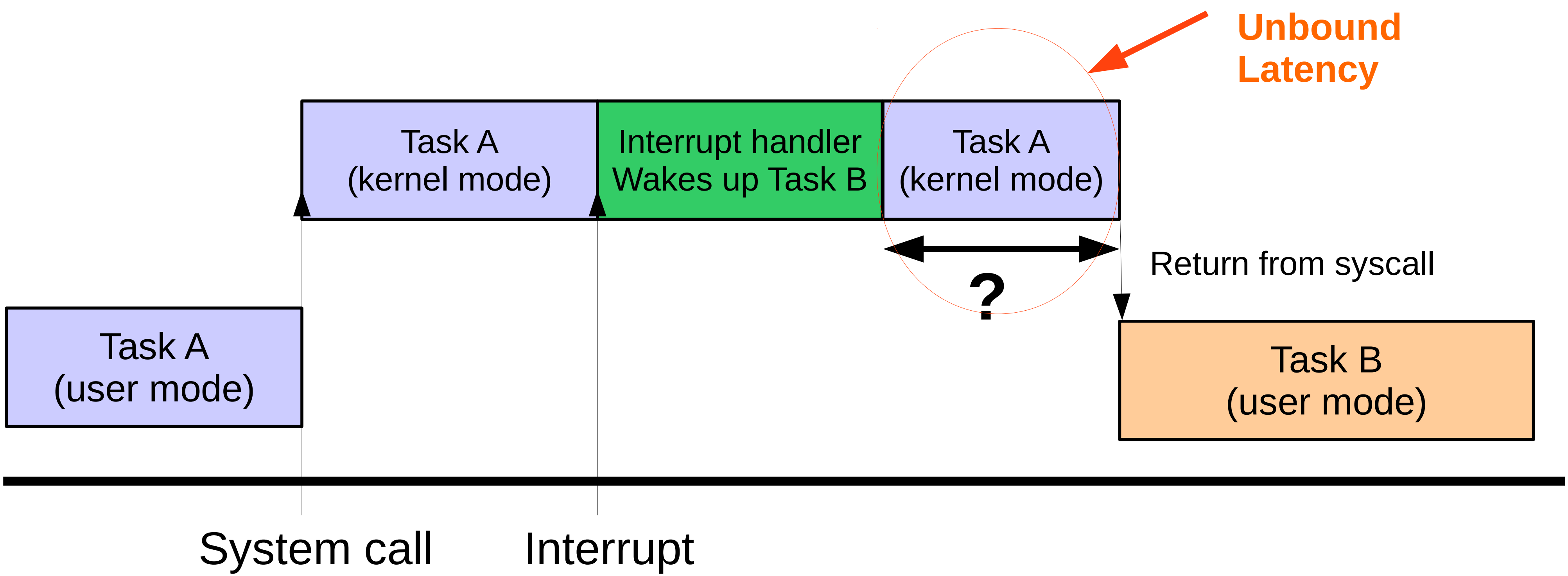
Non-Preemptive

- CONFIG_PREEMPT_NONE
- Preemption is not allowed in **Kernel Mode**
- Preemption could happen upon returning to user space



Non-Preemptive Issue

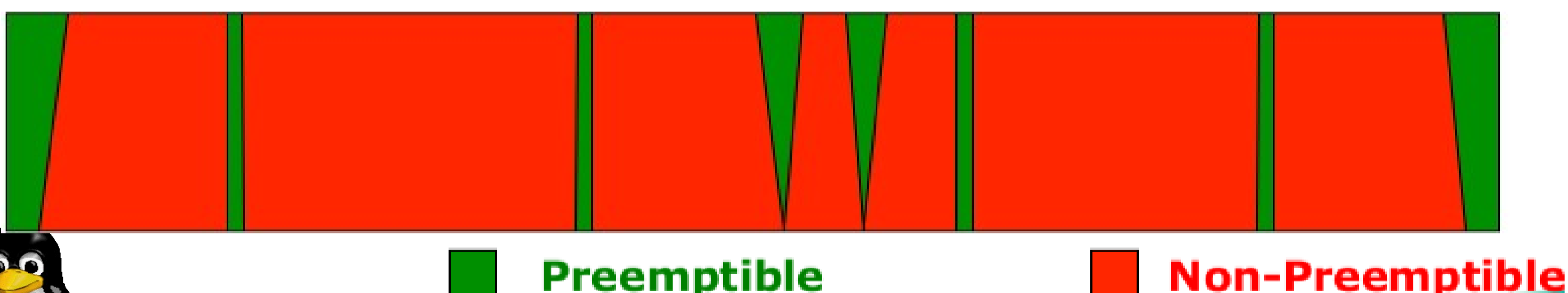
- Latency of Non-Preemptive configuration



Preemption Point

- CONFIG_PREEMPT_VOLUNTARY
- Insert ***explicit*** preemption point in Kernel
  - might_sleep

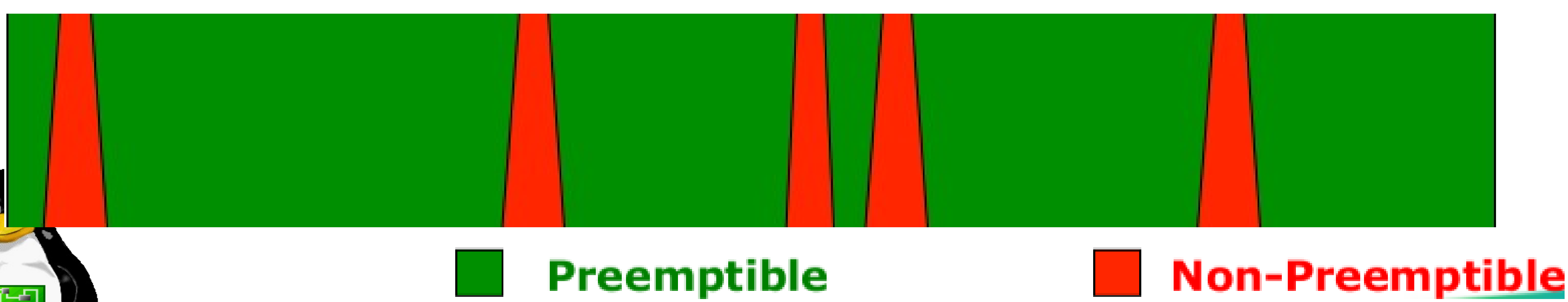
- Kernel can be preempted only at preemption point



Preemptible Kernel

- CONFIG_PREEMPT
- ***Implicit*** preemption in Kernel

- preempt_count
  - Member of thread_info
  - Preemption could happen when preempt_count == 0

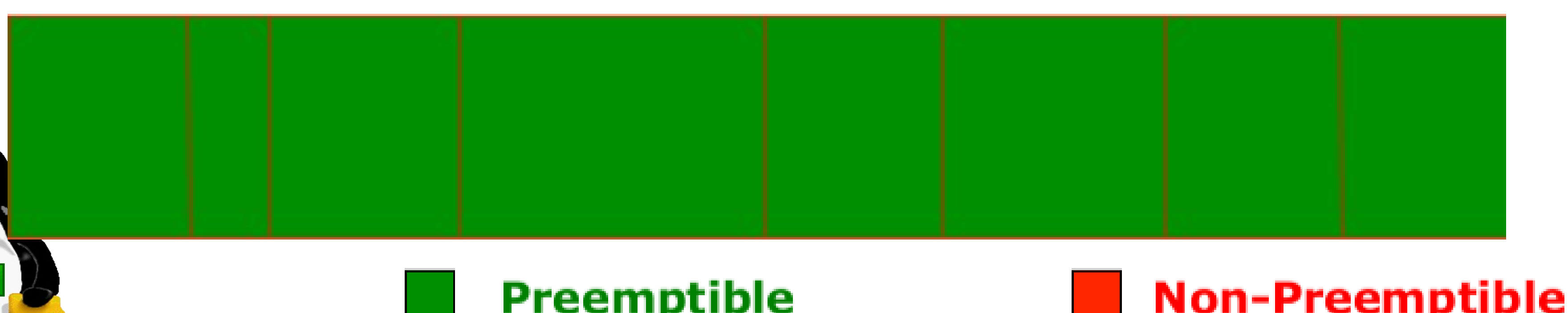


Preemptible Kernel

- Preemption could happen when
  - return to user mode
  - return from irq handler
    - Kernel is preemptible with timer interrupt (RR)

Full Preemptive

- CONFIG_PREEMPT_RT_BASE / CONFIG_PREEMPT_RT_FULL
  - Difference appears in the interrupt context
- Goal: Preempt Everywhere except
  - Preempt disable
  - Interrupt disable
- Reduce non-preemptible cases in kernel
  - spin_lock
  - Interrupt



Latency Issue

- Preemption
- **Critical Section**
- Interrupt

Spinlock

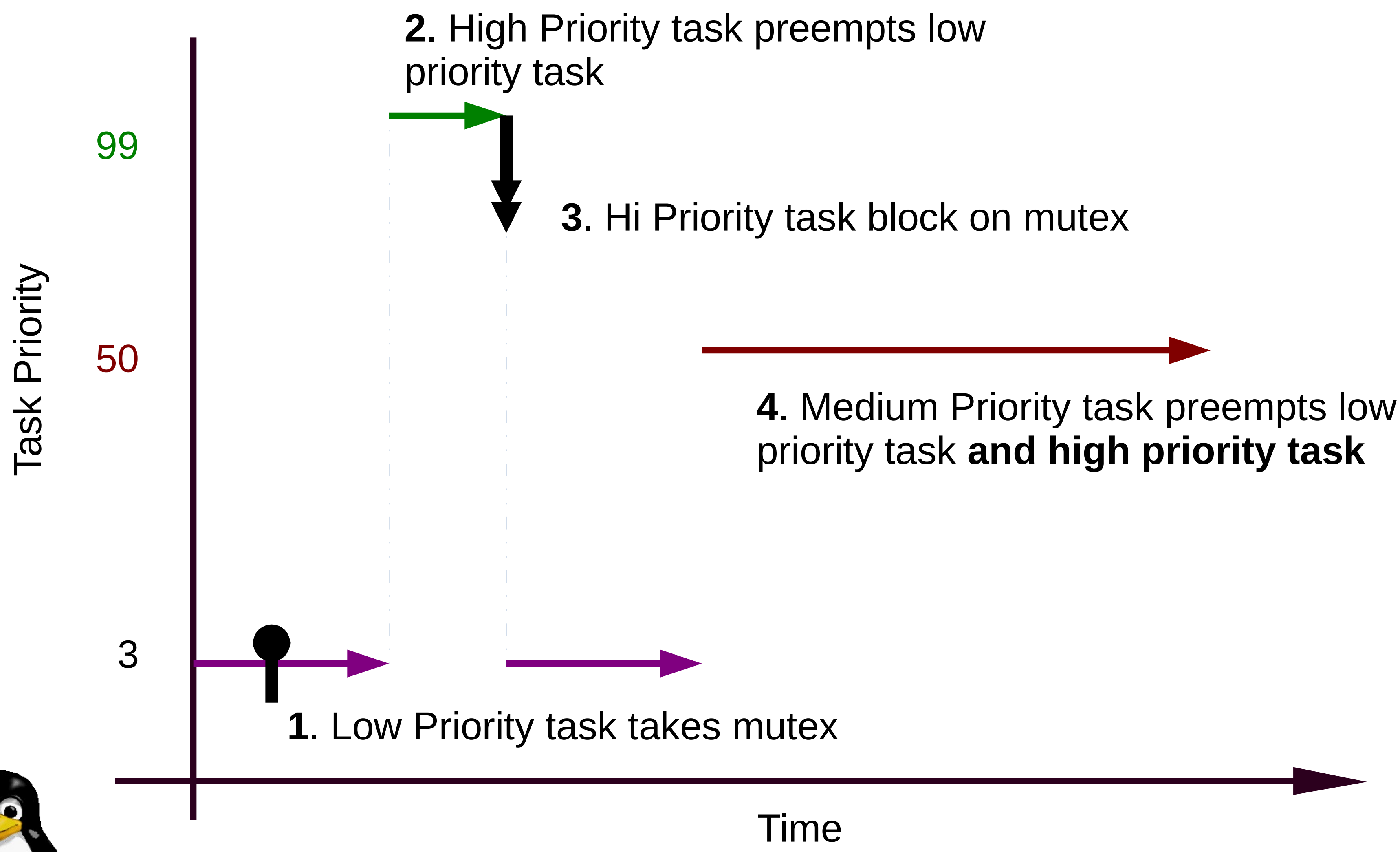
- Task will be busy waiting until it acquires the lock

- Spinlock in Preemptible Linux (CONFIG_PREEMPT)
  - Uniprocessor
    - preempt_disable
  - SMP
    - preempt_disable
    - Lock acquire, busy waiting

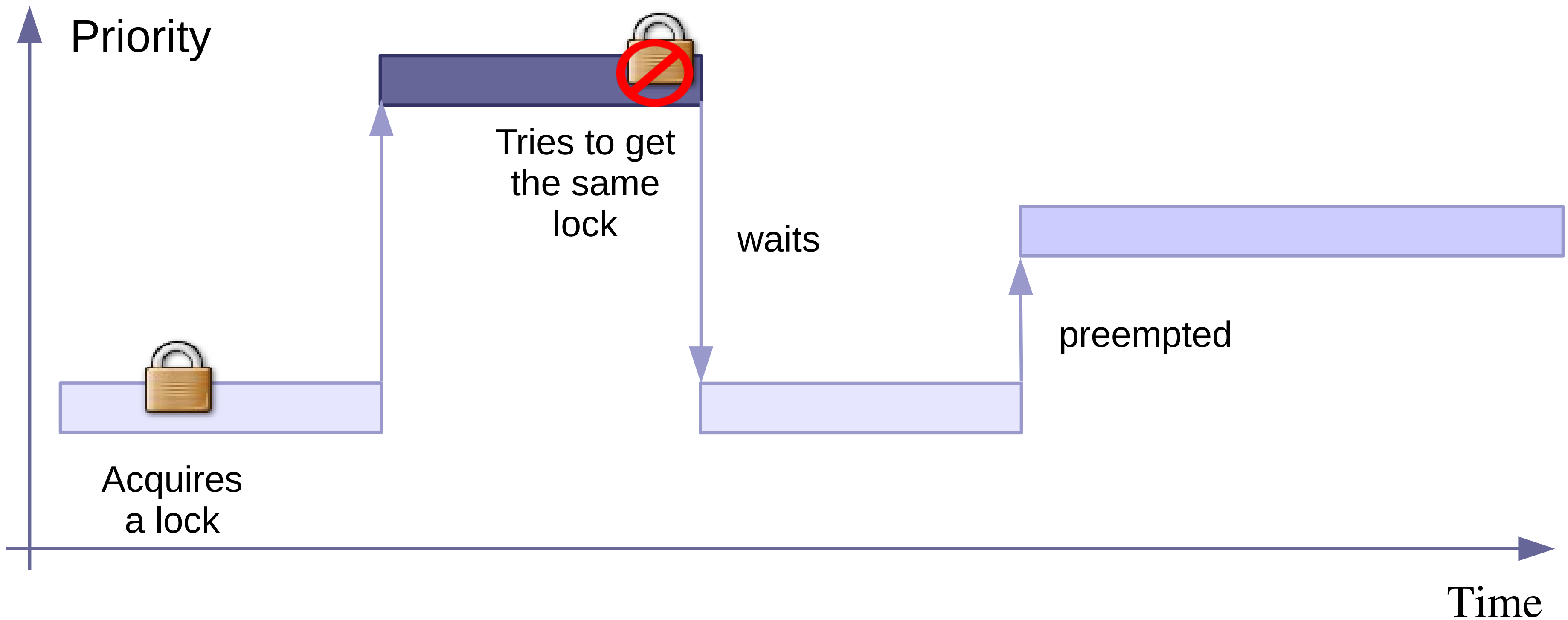
Spin lock in Full Preemptive

- Preemptible splin_lock
  - rtmutex
  - Avoid priority inversion
    - Priority Inheritance (PI) protocol

Priority Inversion

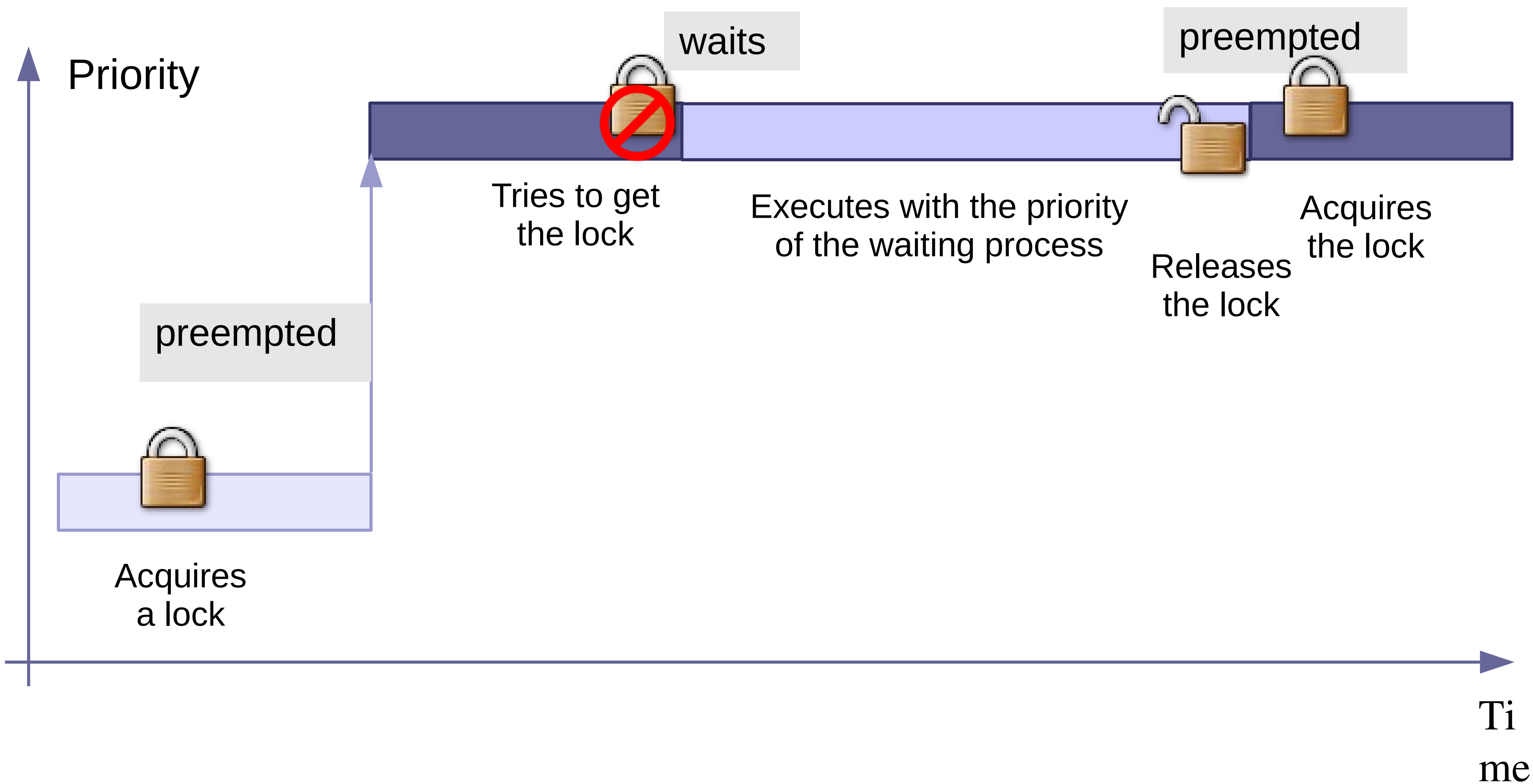


Priority Inversion



- High priority task is preempted by medium priority task
- Unbound waiting for high priority task

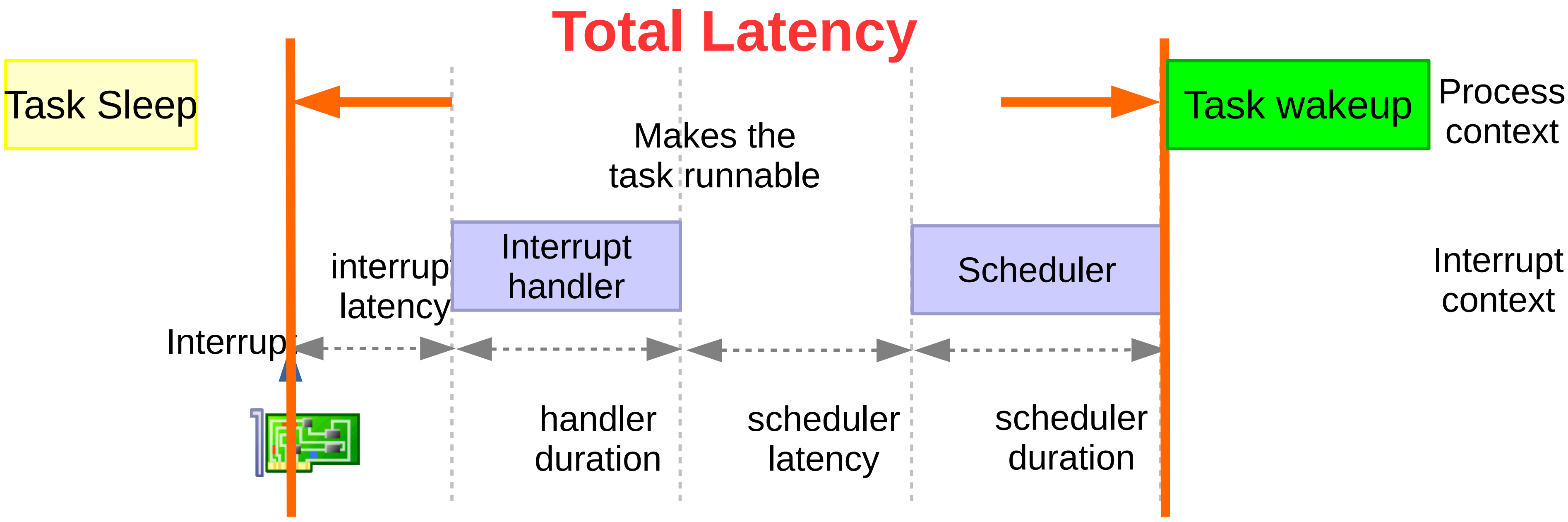
Priority Inheritance

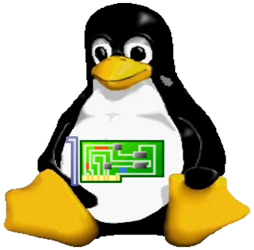


Latency Issue

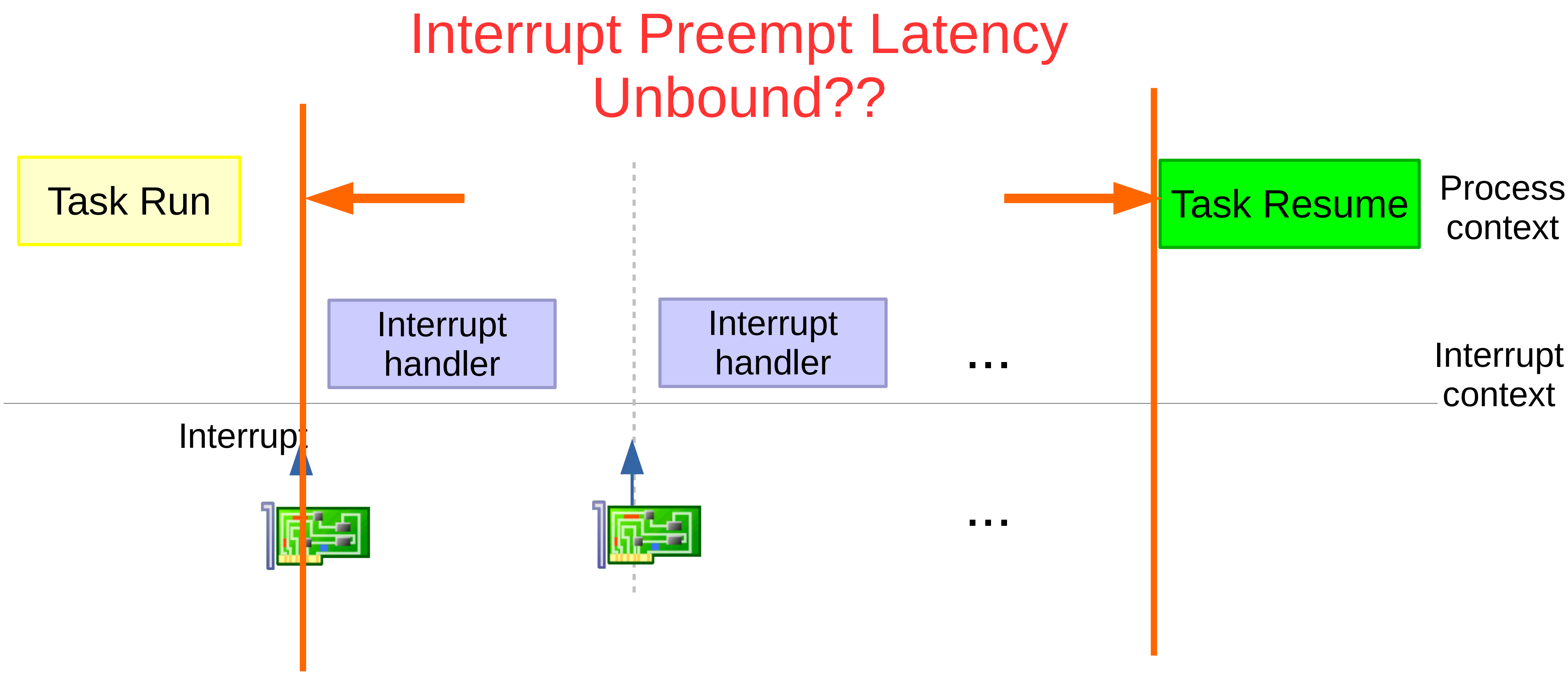
- Preemption
- Critical Section
- **Interrupt**

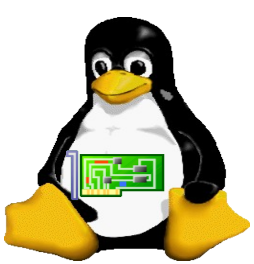
Task triggered by IRQ





Task interrupted by IRQ

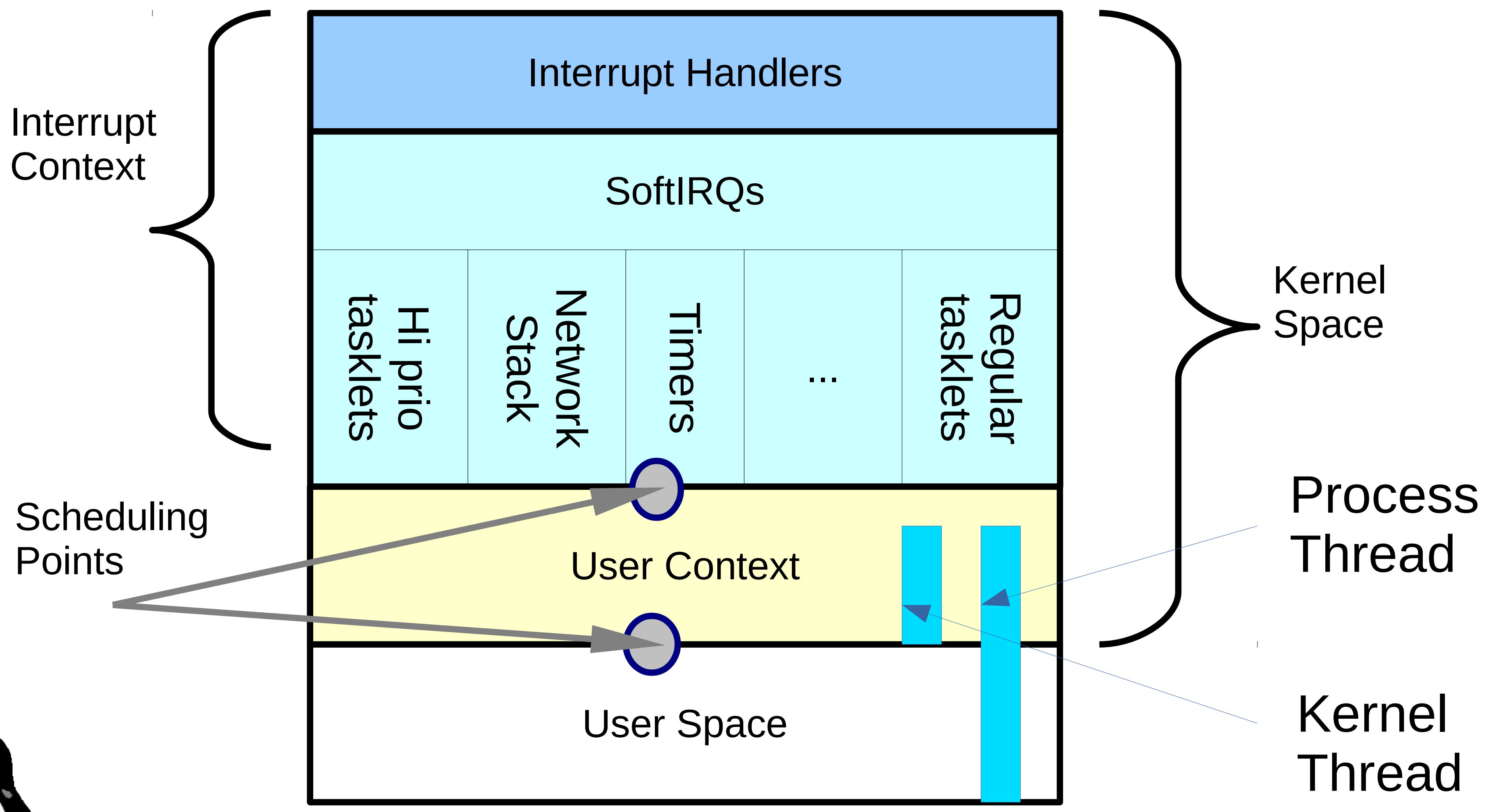


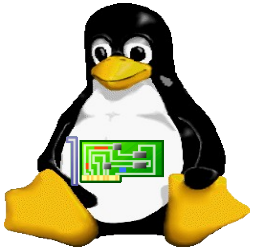


Interrupt Context

- In original Kernel
  - HardIRQ
  - SoftIRQ
  - Highest priority in system

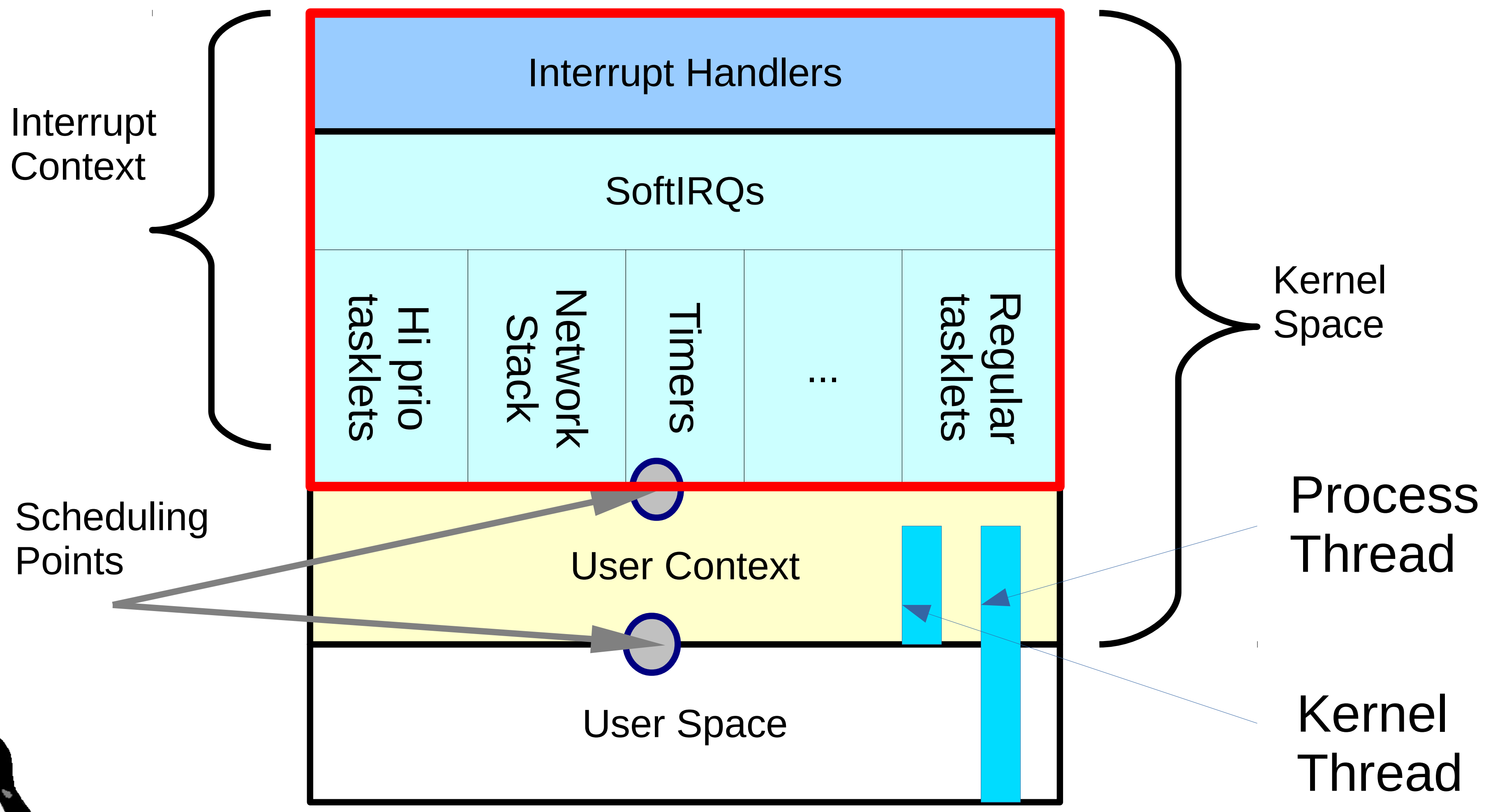
Original Linux Kernel

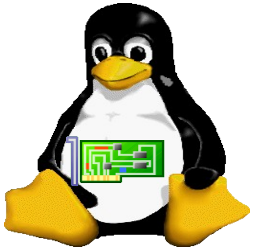




Original Linux Kernel

Priority of interrupt context is always higher than others



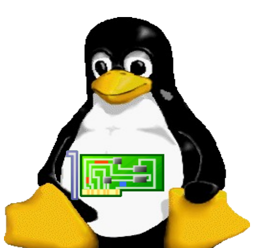


Non-determined Issue

- Interrupt context can always preempt others

- Interrupt as an external event
  - Interrupt number of a time interval is non-determinated
  - Nature of interrupt, can not be avoided

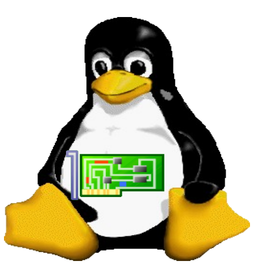
- Behavior of interrupt handler is not well defined
  - Non-determined interrupt handler
  - Threaded IRQ



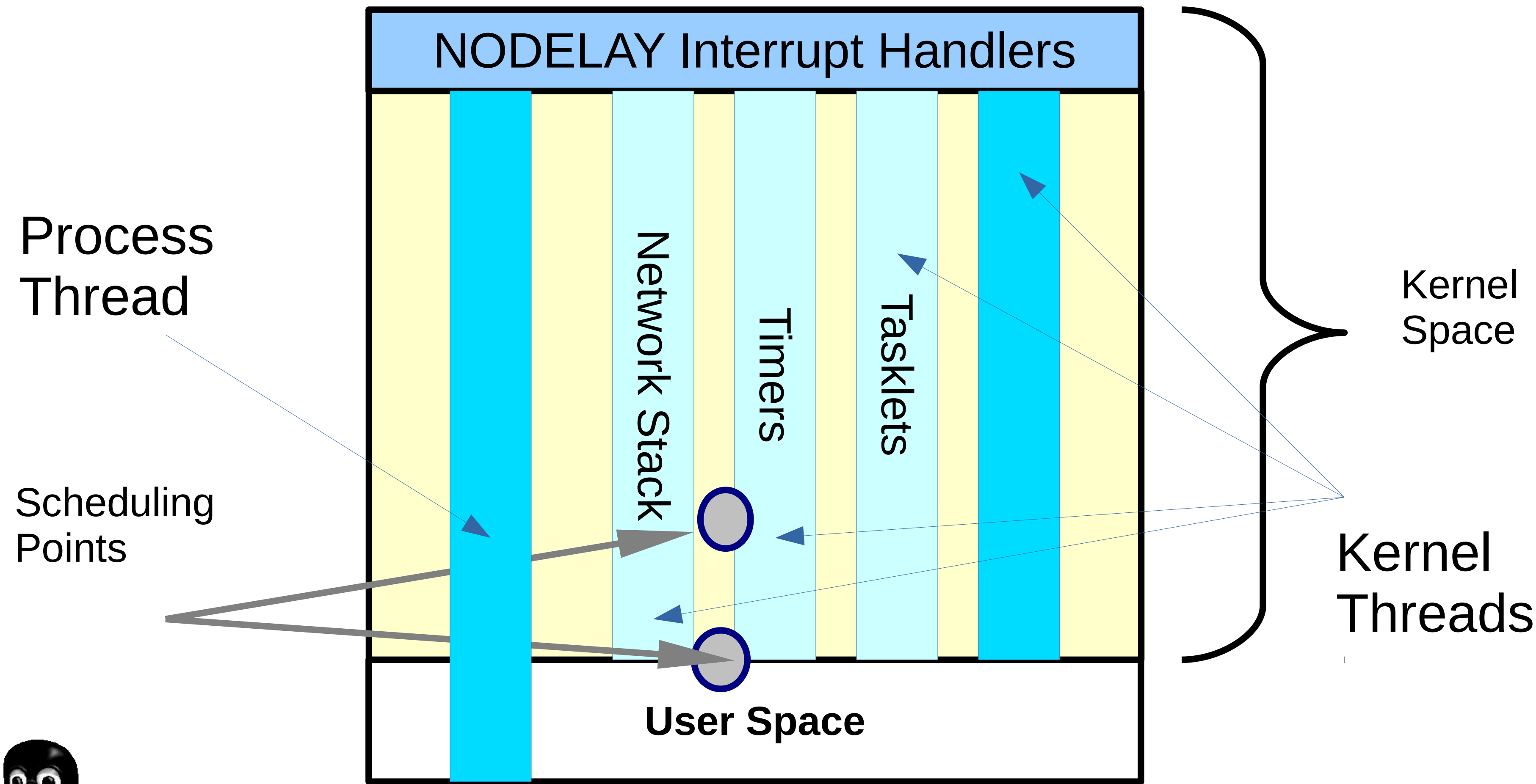
IRQ in PREEMPT_RT

- Threaded IRQ
  - IRQ handler is actually a kernel thread by default
  - Hard IRQ handler only wakes up IRQ handler thread
    - Behavior of hard IRQ handler is well defined
  - Original IRQ handler (No Delayed) is reserved by IRQF_NO_THREAD flag.

- Remove softirq
  - ksoftirqd as a normal kernel thread, handles all softirqs



PREEMPT_RT

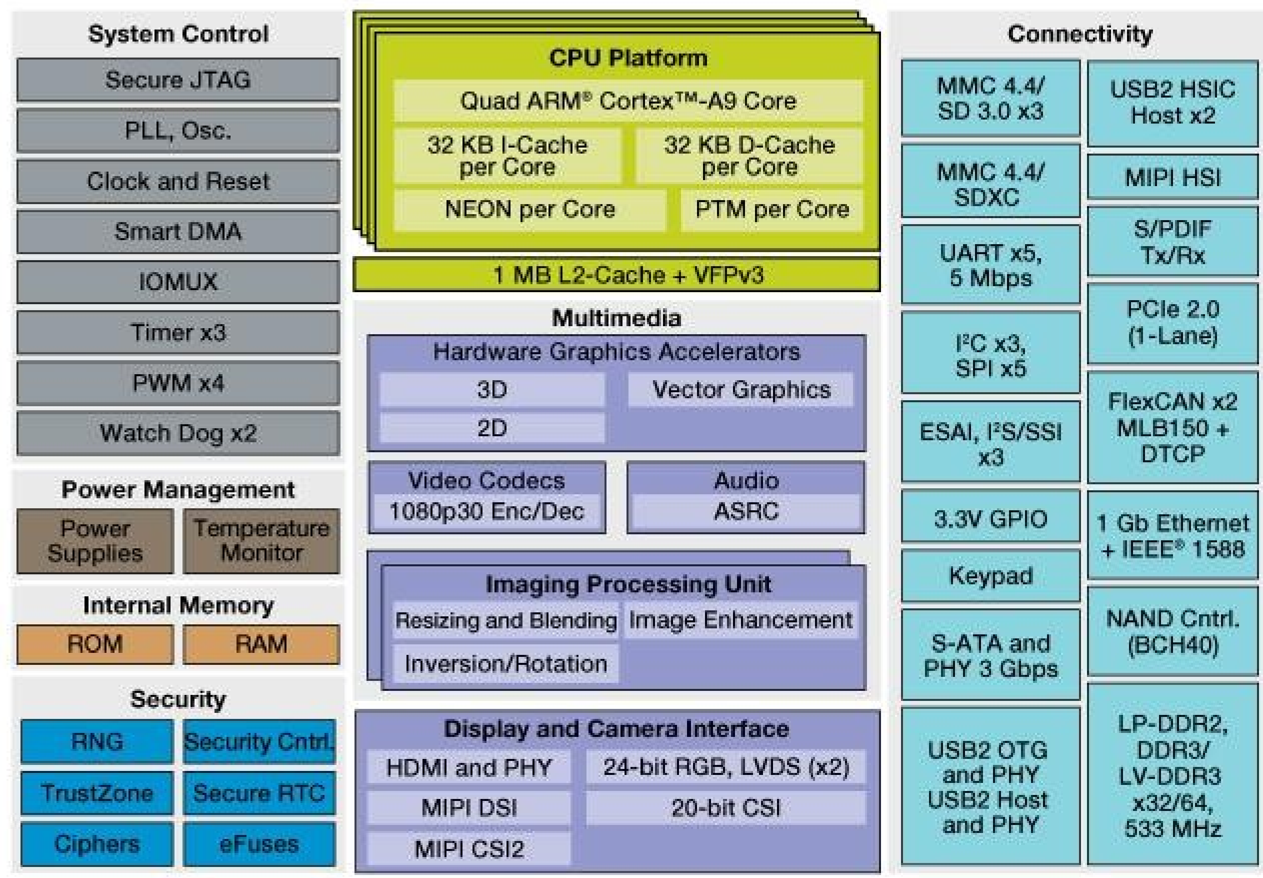


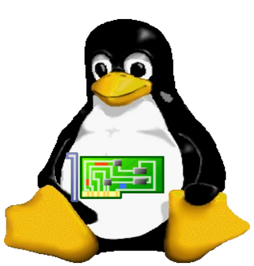
NODELAY Interrupt Handlers
Process Thread
Network Stack
Timers
Tasklets
Scheduling Points
User Space
Kernel Space
Kernel Threads

Experiments on ARM

- NXP i.MX6Q SABRE Board
  - Cortex-A9 x 4; 1 GHz
- Linux kernel 4.1+

i.MX 6Quad Applications Processor Block Diagram



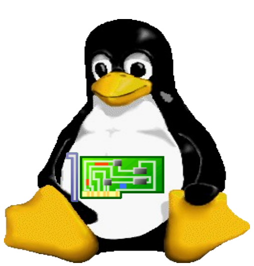


實驗方式

- 在 user space 中使用高精準度的沉睡系統呼叫進行沉睡，並在不同的工作情境以及不同的 Kernel Timer 測量實際沉睡的時間 .
- 測量方式 : 使用 clock_gettime 獲取沉睡前以及沉睡後的時間

```
clock_gettime(CLOCK_MONOTONIC, &time1);
next_time = time1;
next_time.tv_nsec += (sleep_us * 1000);
tsnorm(&next_time);
clock_nanosleep(CLOCK_MONOTONIC, TIMER_ABSTIME, &next_time, NULL);
clock_gettime(CLOCK_MONOTONIC, &time2);
```

- 沉睡方式 : 以 clock_nanosleep 沉睡
- 工作情境 : 分為無週邊通訊負載以及有大量的週邊負載兩種情境，負載產生方式為
  - 每秒印出 /proc/interrupts 的內容來產生 serial 負載
  - 週期為 1ms 的 ping 網路通訊來產生網路負載



Practical Issues in PREEMPT_RT

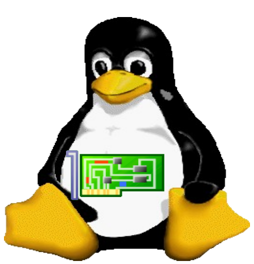
Tool for Observation

- Linux Kernel Tracing Tool
- Event tracing
  - Tracing kernel events

```
mount -t debugfs debugfs /sys/kernel/debug
```

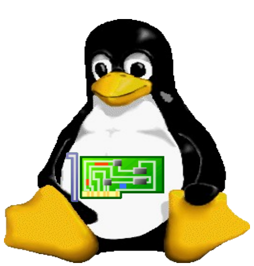
  - Event: irq_handler_entry, irq_handler_exit, sched:* (/sys/kernel/debug/tracing/events/)

```
trace-cmd record -e irq_handler_entry \
                 -e irq_handler_exit \
                 -e sched:*
```



Trace Log

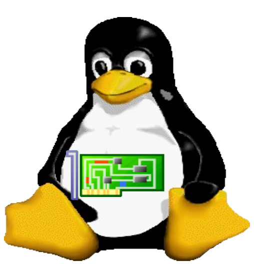
```
trace-cmd-1061  [000]   142.334403: irq_handler_entry:    irq=29 name=critical_irq
trace-cmd-1061  [000]   142.334437: sched_wakeup: critical_task:1056 [9] success=1 CPU:000
trace-cmd-1061  [000]   142.334456: irq_handler_exit:    irq=29 ret=handled
trace-cmd-1061  [000]   142.334480: sched_wakeup: ksoftirqd/0:3 [98] success=1 CPU:000
trace-cmd-1061  [000]   142.334505: sched_stat_runtime: comm=trace-cmd pid=1061 runtime=201500 [ns] vruntime=4720432487 [ns]
trace-cmd-1061  [000]   142.334526: sched_switch:    trace-cmd:1061 [120] R ==> critical_task:1056 [9]
```



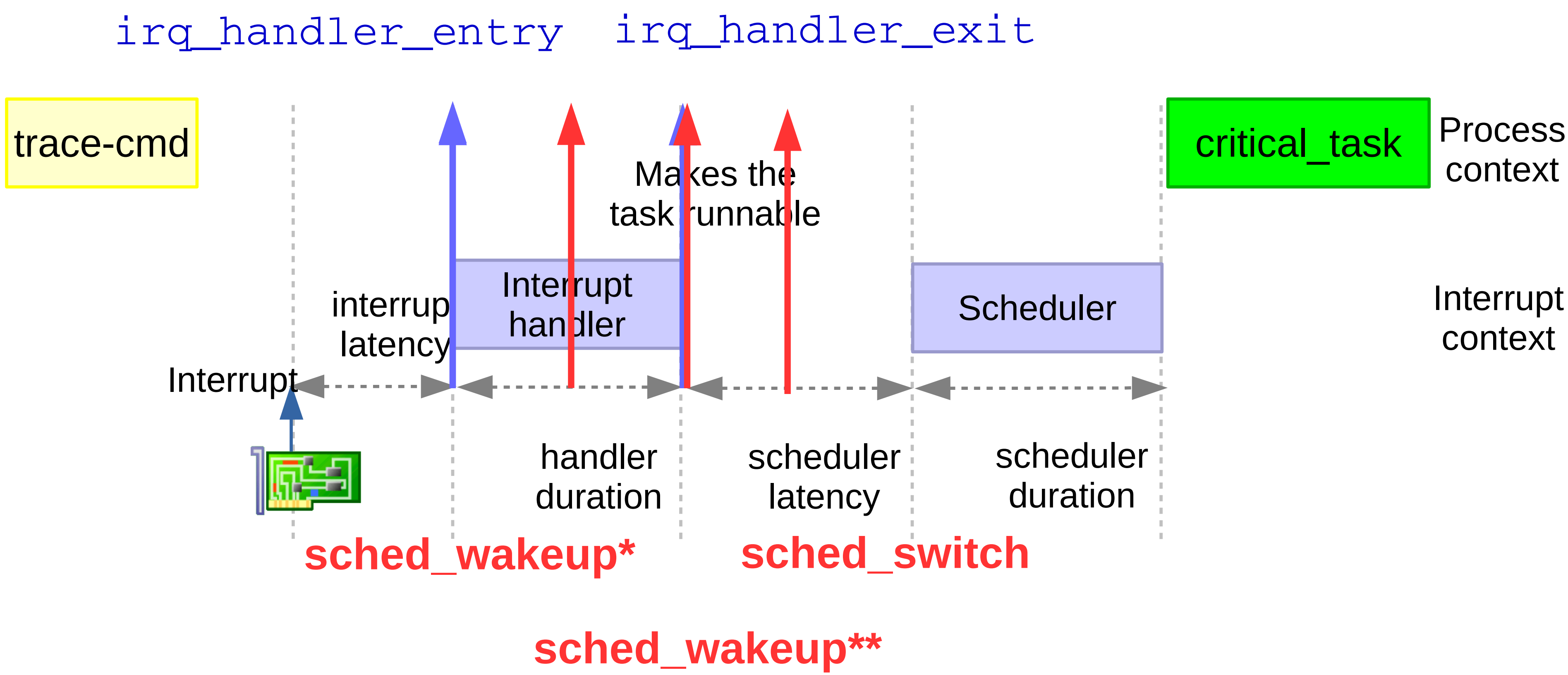
Trace Log Format

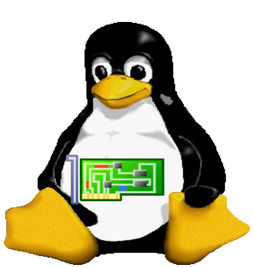
- `trace-cmd-1061  [000]   142.334403: irq_handler_entry: irq=21 name=critical_irq`

Current Task	CPU#	Time Stamp	Event Name	Message
trace-cmd-1061	[000]	142.334403	irq_handler_entry	Irq=21 name=critical_irq



Trace Log

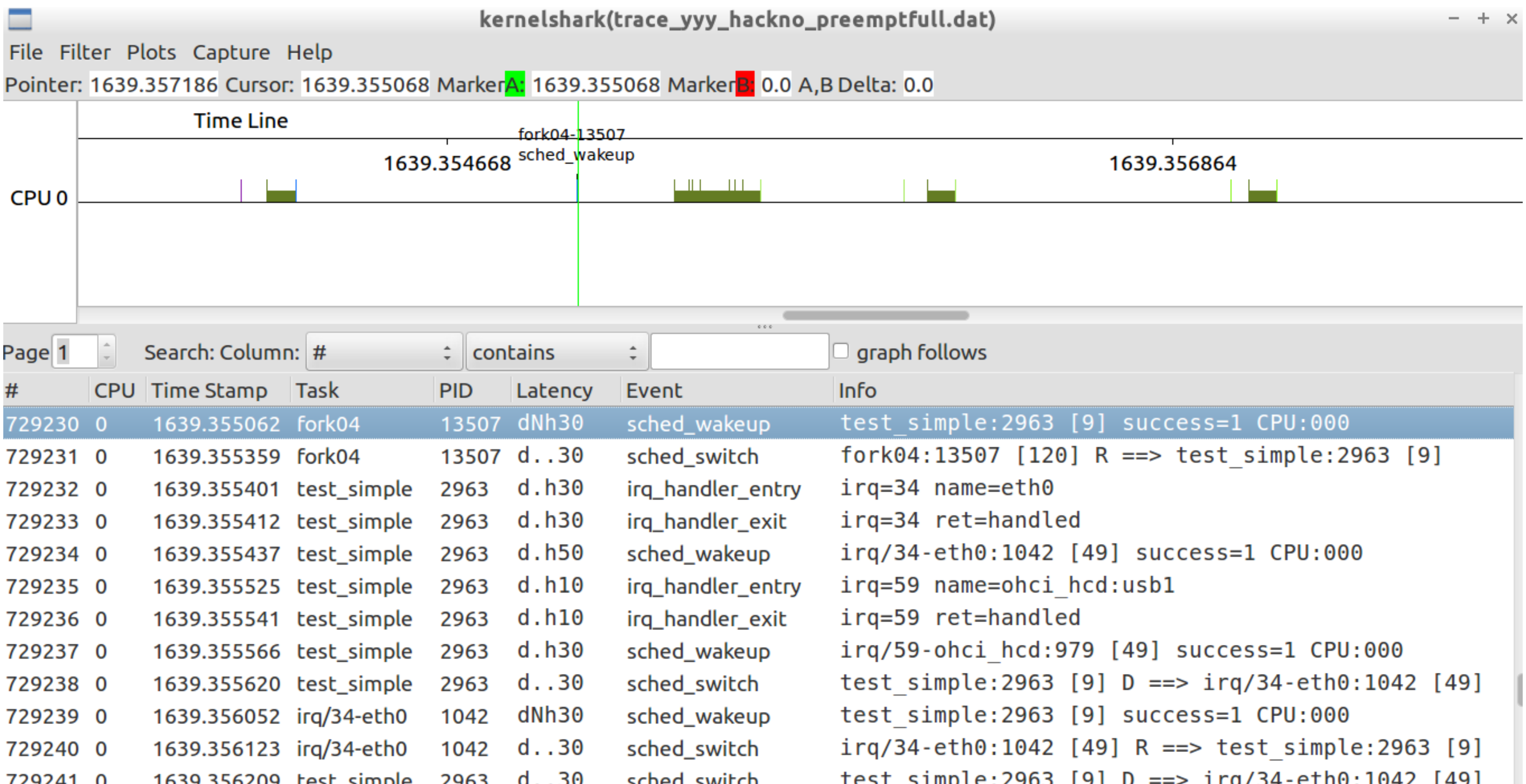




sched_wakeup*: wakeup critical_task
sched_wakeup**: wakeup ksoftirqd

Trace Log Visualization

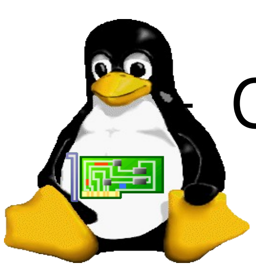
- KernelShark



HRT Impact

- High Resolution Timer
  - Non threaded Interrupt
    - Non-determined IRQ handler
  - processor affinity overhead
  - Important and complex component of Linux Kernel
    - Tick timer
    - Timer for RR scheduler
    - nanosleep, clock_nanosleep
    - Futex, rtmux
    - Others

Many Unpredicted Timer Interrupts !!



  - Can not be disabled