# Linux Real Time Patch

Understanding a Real-Time System by Steven Rostedt:
https://www.youtube.com/watch?v=wAX3jOHHhn0

Building Embedded Linux Systems, Chapter 14: The RT Patch

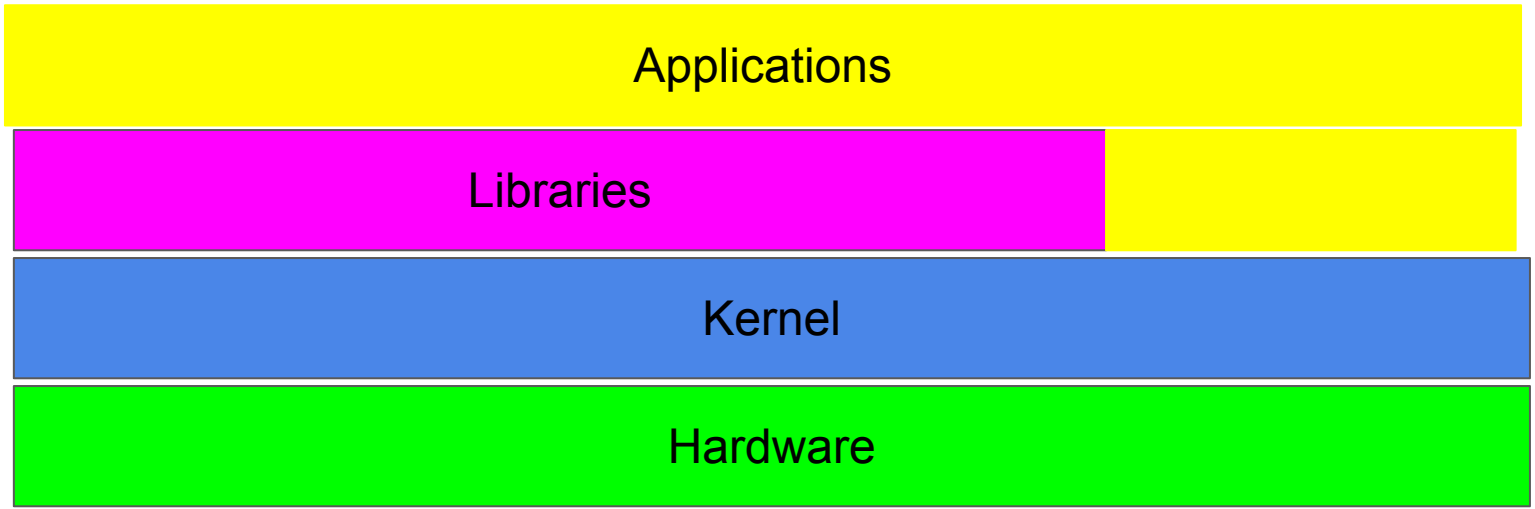
# What is real time?

- Deterministic results
- Repeatable results
- Predictable: Doing what you expect when you expect it
- No unbounded latency
- Can calculate worst case scenarios

# Different from real fast. What is real fast?

- Hot cache look ahead features
- Translation Lookaside Buffer (TLB) for paging
- Least interruptions
- Transactional memory

# System architecture



# The hardware

If this isn’t deterministic, forget the rest. Example causes of non-determinism:

- Memory cache
- Branch prediction
- NUMA
- Hyper-threading
- TLB
- Transactional memory
- SMI
- CPU frequency scaling

# Memory cache

- Try to run tests with cold cache
- Try to find the worst case scenario
- If your system works without cache, it should work with cache
    - Non cache is more deterministic

# Branch Prediction

- CPU recognizes branch patterns
- Optimizes the pipeline
- What happens when logic changes?

# Branch Prediction

```
Good:
 Performance counter stats for './deadline_test -c 0,3':

      15309.175906      task-clock (msec)         #    1.272 CPUs utilized            (100.00%)
            16,693      context-switches          #    0.001 M/sec                    (100.00%)
                6      cpu-migrations            #    0.000 K/sec                    (100.00%)
              220      page-faults               #    0.014 K/sec                    (100.00%)
   45,336,201,800      cycles                    #    2.961 GHz                      (100.00%)
   27,839,671,679      stalled-cycles-frontend   #   61.41% frontend cycles idle     (100.00%)
   <not supported>     stalled-cycles-backend
   24,654,001,731      instructions              #    0.54  insns per cycle
                                                 #    1.13  stalled cycles per insn  (100.00%)
    5,846,443,551      branches                  #  381.891 M/sec                    (100.00%)
          798,866      branch-misses             #    0.01% of all branches          (100.00%)
   15,143,395,012      L1-dcache-loads           #  989.171 M/sec                    (100.00%)
        6,830,685      L1-dcache-load-misses     #    0.05% of all L1-dcache hits    (100.00%)
        5,646,962      LLC-loads                 #    0.369 M/sec                    (100.00%)
   <not supported>     LLC-load-misses

     12.037594790 seconds time elapse
```

# Branch Prediction

```
Bad:
 Performance counter stats for './deadline_test -c 0,3':

       9191.898036      task-clock (msec)         #    0.763 CPUs utilized            (100.00%)
            16,693      context-switches          #    0.002 M/sec                    (100.00%)
                 9      cpu-migrations            #    0.001 K/sec                    (100.00%)
               219      page-faults               #    0.024 K/sec                    (100.00%)
    22,043,401,852      cycles                    #    2.398 GHz                      (100.00%)
    13,531,252,221      stalled-cycles-frontend   #   61.38% frontend cycles idle     (100.00%)
   <not supported>      stalled-cycles-backend
    12,012,005,499      instructions              #    0.54  insns per cycle
                                                  #    1.13  stalled cycles per insn  (100.00%)
     2,841,672,774      branches                  #  309.150 M/sec                    (100.00%)
         4,689,983      branch-misses             #    0.17% of all branches          (100.00%)
     7,339,066,676      L1-dcache-loads           #  798.428 M/sec                    (100.00%)
         6,443,901      L1-dcache-load-misses     #    0.09% of all L1-dcache hits    (100.00%)
         5,131,751      LLC-loads                 #    0.558 M/sec                    (100.00%)
   <not supported>      LLC-load-misses

      12.040237863 seconds time elapsed
```

# NUMA

- Memory speeds dependent on CPU
- Need to organize the tasks
- Make sure RT tasks always have their memory in one place

# Hyper-threading on Intel processor

- One execution unit, one cache, one system bus
- Two sets of registers, two sets of CPU pipelines
- Execution unit switches between them on stalls
- Recommended to disable for RT

# Transactional memory

- Allows for parallel actions in the same critical section
- Backs out when the same memory is touched
- Restart the transaction or take another part

# System Management Interrupt (SMI)

- Puts processor into System Management Mode (SMM)
- Check CPU temperature change frequency
- Perform memory scans
- Causes the system to stop what it was doing

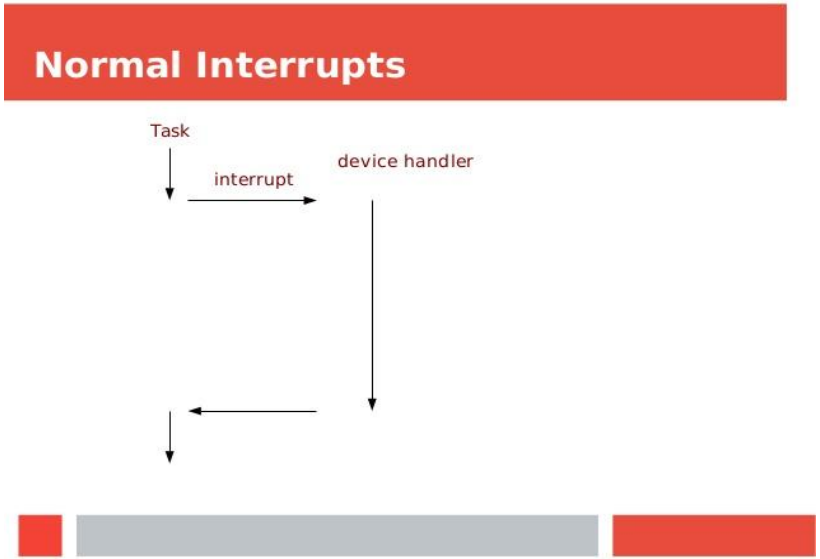
# CPU frequency scaling

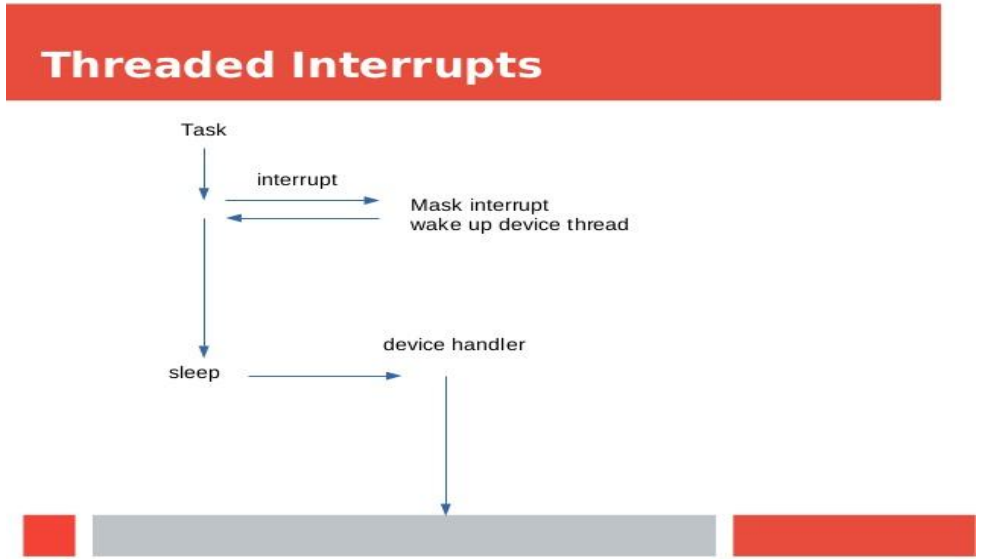
- Compute times change based on CPU frequency
- CPU wakeup times are different based on idle deep sleep vs. higher frequency

# RT Linux kernel Preempt_RT patch

- Sched_deadline
  - Earliest deadline first scheduling, in addition to FCFS/FIFO and RR
- CPU isolation
- Threaded interrupts
- Complete preemption
- High resolution timers

# Threaded interrupts




- User tasks can run higher priority than interrupts
- Set required interrupts higher than your task

# Complete preemption

The Complete Preemption converts spin locks from busy loops into mutexes. Instead of spinning in a busy loop on a CPU, a process that tries to get a spin lock just goes to sleep and lets the kernel schedule other processes on the CPU. When the spin lock is released, the blocked process is awakened. If it is currently the highest-priority running process, it will preempt the current thread and take over the CPU. By eliminating the disabling of preemption at the acquisition of a spin lock, the Linux kernel becomes tremendously more responsive and latencies are reduced to a few microseconds.

# Linux Jiffies

One thing that is crucial to a real-time system is the ability to trigger an event at a specific time (otherwise there's no sense in calling the system real-time). In the early 2.6 versions of Linux and earlier, the smallest unit of time was called a jiffy. A jiffy started out as one 100th of a second. A global variable called HZ represented the hertz of jiffies, and with the one 100th frequency, the HZ was defined as 100. This was all very simple and suited the needs of Linux at the time, but as machines became faster and people ran more applications on Linux, the 100 HZ setting started to show its age. Under a heavy load, applications were not smooth with the low HZ frequency. To get better reaction times, the HZ value became 1,000. With the 1,000 HZ value for jiffies, the best reaction time that could be set was 1 millisecond. This is still very poor for any serious application that needs the slightest bit of real-time. But for every jiffy, a timer interrupt was needed to update the jiffy variable. So a balance was needed where one could choose to suffer increased overhead (more timer interrupts) in order to gain a finer timer resolution.

# Two different timers

The RT kernel maintains two types of timers with two distinct use cases.

- The high-resolution timer ("hrtimer") mechanism provides accurate timers for work that needs to be done in the near future; hrtimer uses RB trees and almost always run to completion.
- “Timeouts”, instead, are normally used to alert the kernel to an expected event that has failed to arrive — a missing network packet or I/O completion interrupt, for example. The accuracy requirements for these timers are less stringent (it doesn't matter if an I/O timeout comes a few milliseconds late), and, importantly, these timers are usually canceled before they expire. The timer wheel is used for the latter variety of timers. (https://lwn.net/Articles/152436/)

# RT Tasks: priority inheritence

A thread can turn on priority inheritance for a mutex by specifying PTHREAD_PRIO_INHERIT as a mutex attribute.

The following sample code snippet implements a pthread mutex that uses priority inheritance:

```
extern pthread_mutex_t mutex;
pthread_mutexattr_t attr;
if (pthread_mutexattr_init(&attr))
        perr("pthread_mutexattr_init");
if (pthread_mutexattr_setprotocol(&attr, PTHREAD_PRIO_INHERIT))
        perr("pthread_mutexattr_setprotocol");
```

# RT Tasks: memory locking

## memory locking

**mlockall()**

- **Lock in memory to prevent page faults**

**MCL_CURRENT**

- **Lock in all current pages**

**MCL_FUTURE**

- **Lock in pages for heap and stack and shared memory**

**Careful about how much you lock in!**

# Preempt_RT patch on Raspberry PI 3, Model B+