Walk the Line – How RT-safe Application Design Can Be Supported

Jan Kiszka | Linux Plumbers Conference, Real-Time MC, September 19, 2024

Facts?

Real-Time Application developers…

- receive extra trainings on real-time system design
- have deep understanding of the kernel, toolchain, libraries they use
- only write simple programs, few threads, no lock-nesting
- know how to split real-time from non-real-time

What could possibly go wrong? (1/6)

```
void critical_task(void)
{
    /* how deterministic is your console...? */
    printf("trace: doing work\n");
    do_work();
}

void critical_task(void)
{
    /* yeah, this often works, except when not... */
    void *buffer = malloc(SMALL_SIZE);
    do_work_with(buffer);
    free(buffer);
}
```

What could possibly go wrong? (2/6)

```
void critical_task(int fd)
{
    char buffer[MY_DATA_LEN];

    /* Is this /dev/my-rt-device or some file on a Samba share? */
    read(fd, buffer, sizeof(data));
    process_data(buffer);
}
```

What could possibly go wrong? (3/6)

```
void critical_task(int fd)
{
    /* Is fd pointing to shared memory or a regular file? */
    void *data = mmap(NULL, MY_DATA_LEN, PROT_READ | PROT_WRITE,
                      MAP_SHARED, fd, 0);

    while (!stop) {
        read_modify_write(data);
        wait_next_cycle();
    }
}
```

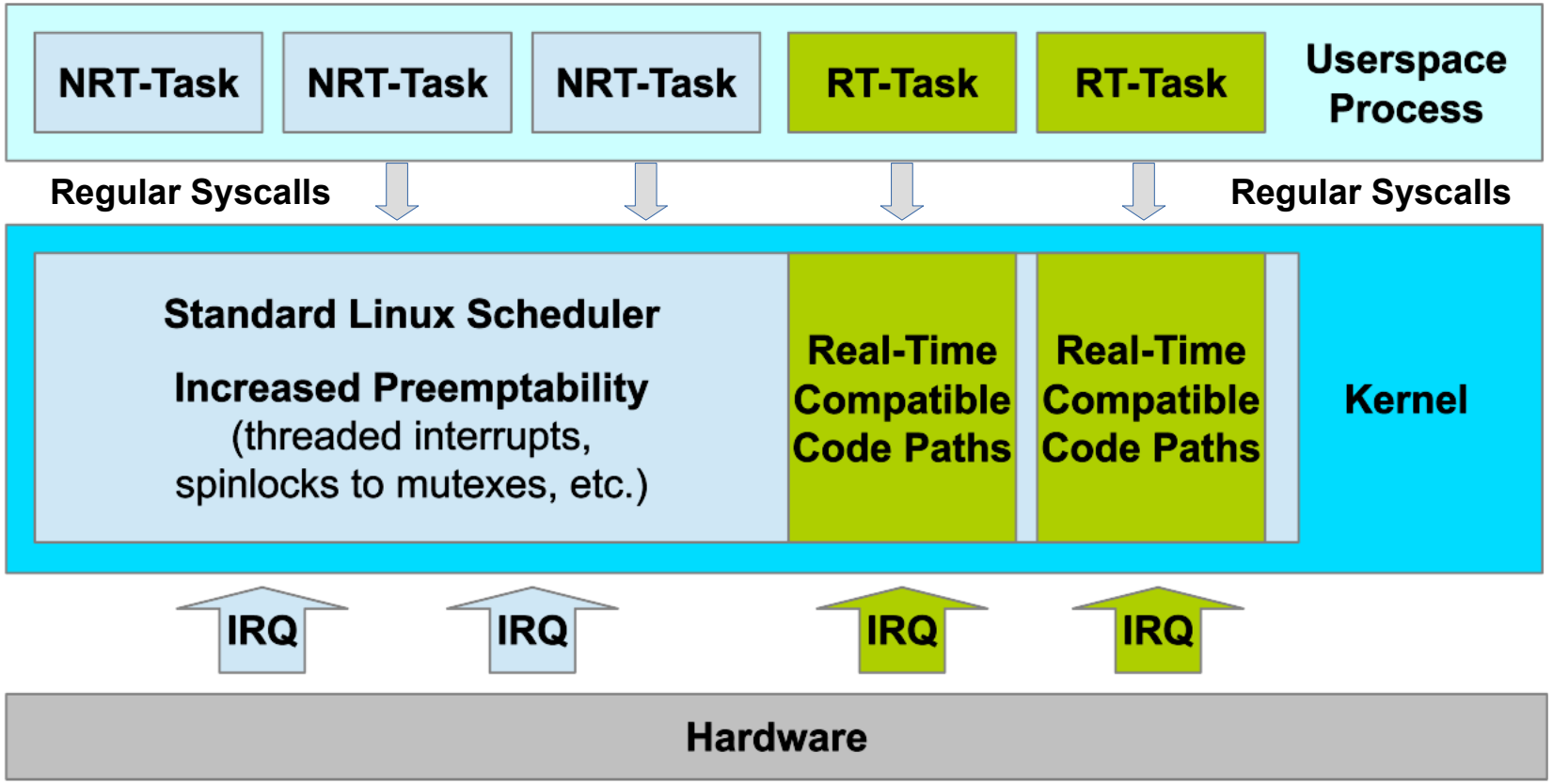
What could possibly go wrong? (5/6)

```
/* Standard lib C++ mutexes are... just standard mutexes */
std::mutex my_mutex;

void high_prio_task(void)
{
    const std::lock_guard<std::mutex> lock(my_mutex);
    do_work();
}

void low_prio_task(void)
{
    const std::lock_guard<std::mutex> lock(my_mutex);
    do_other_work();
}
```

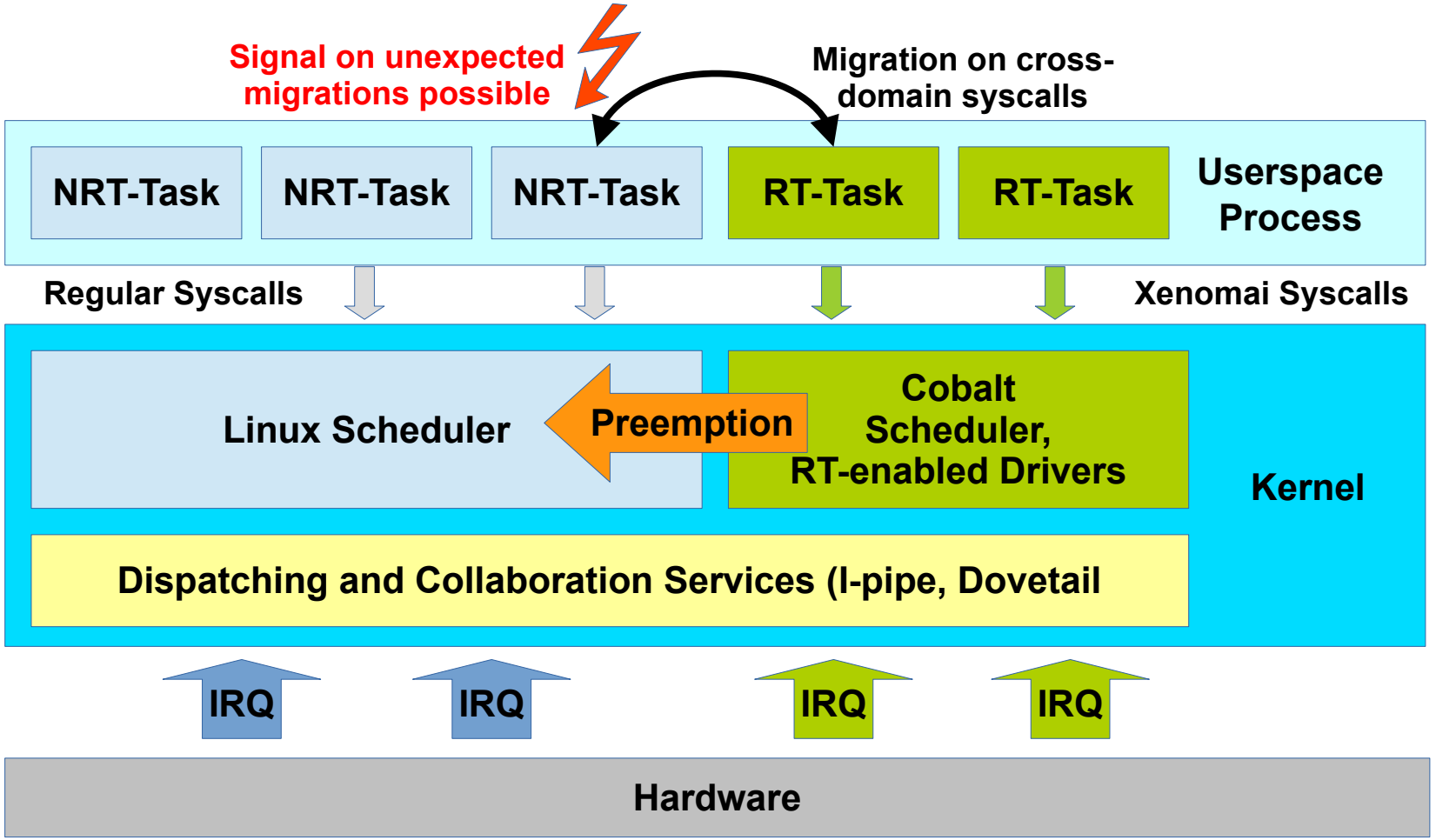
What could possibly go wrong? (6/6)

```
class MySingleton {
public:
    MySingleton() {};
    …
};

void critical_task(void)
{
    /* Ever heard of thread-safe singleton and __cxa_guard_*? */
    static MySingleton s;

    s.do_something();
}
```

Native Real-Time For Linux – PREEMPT-RT



How the Co-Kernel Concept of Xenomai Helps



Mechanism vs. Policy – and Maintenance

Where to detect?

- Xenomai:
  On RT → non-RT migration
- Mainline:
  On syscall entry? Via eBPF?

How to report?

- Xenomai:
  As signal (SIGXCPU) to interested apps
- Mainline:
  tbd

When to trigger?

- Xenomai:
  When Xenomai app declared itself “operational”
- Mainline:
  tbd

…and how to maintain?

- Xenomai:
  Paid via kernel patch maintenance
- Mainline:
  Might be challanging (What is “RT”? Where to annotate? How keep up-to-date? ...)

Some Common Ground: Early Detection in Userspace

Provide tools & instrumenation to find “late-failing” non-RT calls

- malloc & Co.
- Standard locks
- …

Define common interfaces?

- How to declare transitions setup → operational → teardown
- How to receive notifications

Thank you!

Jan Kiszka <jan.kiszka@siemens.com>