

FSMLabs RTLinux hard realtime OS and POSIX 1003.13

Victor Yodaiken

yodaiken@fsmlabs.com

www.fsmlabs.com *and* www.rtlinux.com

This talk is copyright FSMLabs Inc.

Outline

1. **Hard, Soft, and Imaginary Realtime.**
2. RTLinux technology and users.
3. POSIX: good and bad.

What is realtime?

1. *Hard Realtime* : Strict timing rules. Example: stop cutter on machine tool in 2milliseconds, abort rocket firing sequence in 1microsecond, catch the next video frame, ...
2. *Soft Realtime* : " Guideline, not a rule" . Examples: display video. (*but* soft realtime often needs hard-realtime).
3. *Handwaving Realtime* : Whatever you want (e.g. "typical" interrupt response latency").

Soft realtime

- Typically the chip is placed on the board in the right place.
- We almost never drop purchase orders.
- Believe me, the rocket abort sequence *usually* runs before it's too late.

Do you really need hard realtime?

1. Time between packets on gigabit ethernet: 1.5microseconds.
2. Time between frames on 70hz video display: 14milliseconds. (think about serving 10 streams).
3. Time between transactions on a e-commerce server: 1 millisecond or less.
4. Time for 1 degree error on manipulator: 10 microseconds.

The RTOS designers dilemma.

1. On the one hand: only a small simple OS can be predictable and fast.
2. On the other hand: users want TCP/IP and GUIs and ...

The RTOS designers dilemma Part II.

1. Adding RT to a general purpose OS is costly and hard to maintain.
2. Adding complex services to a small RTOS is costly and often self-defeating.

RTLinux basic mechanism

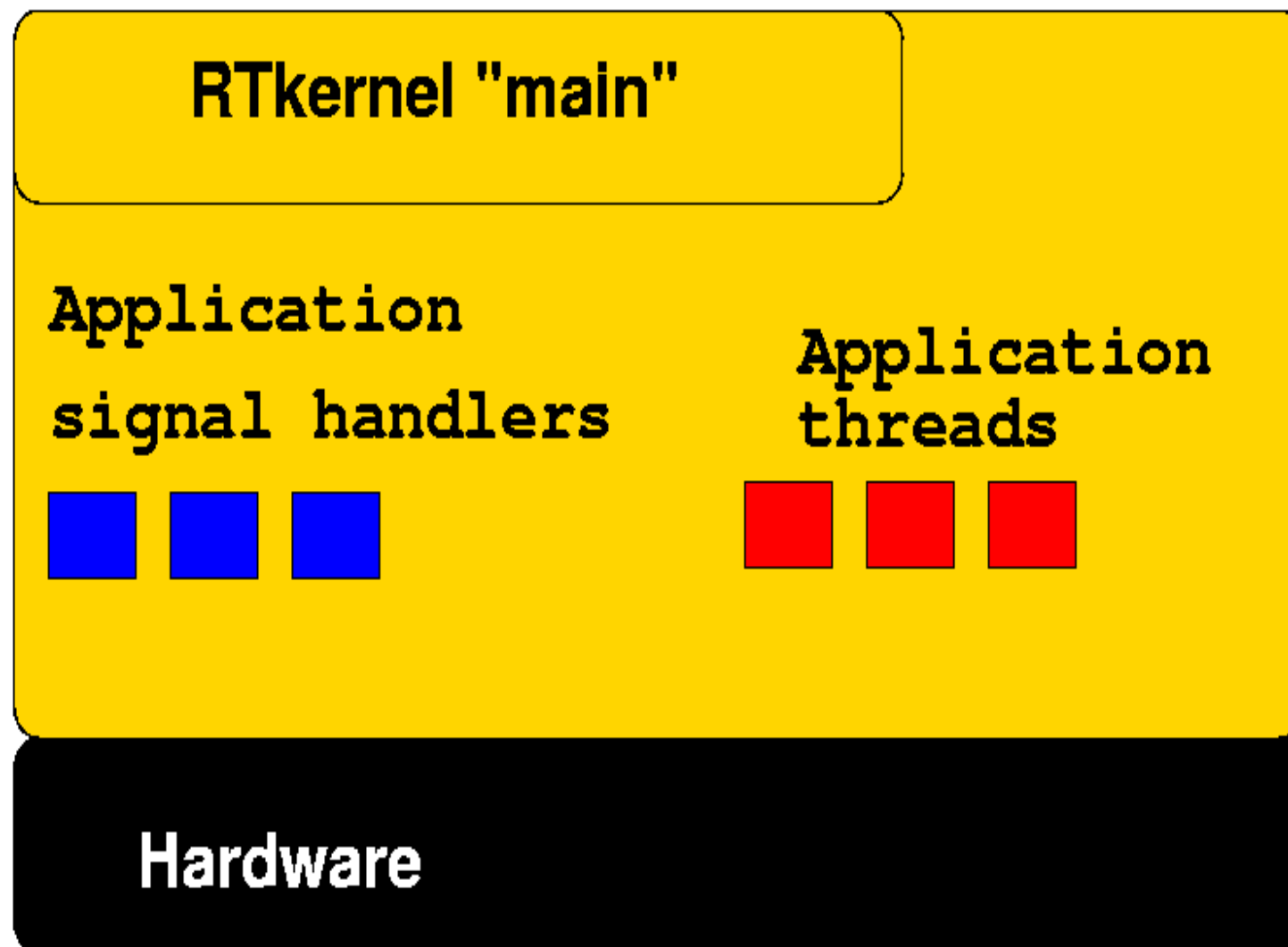
1. Put an emulated interrupt controller in a general purpose OS.
2. The general purpose OS (Linux) cannot disable interrupts.
3. Run the general purpose OS as the low priority subtask.

RTLinux is a hard realtime operating system

RTLinux follows the POSIX PSE51 1001.13 Minimal RT Profile.

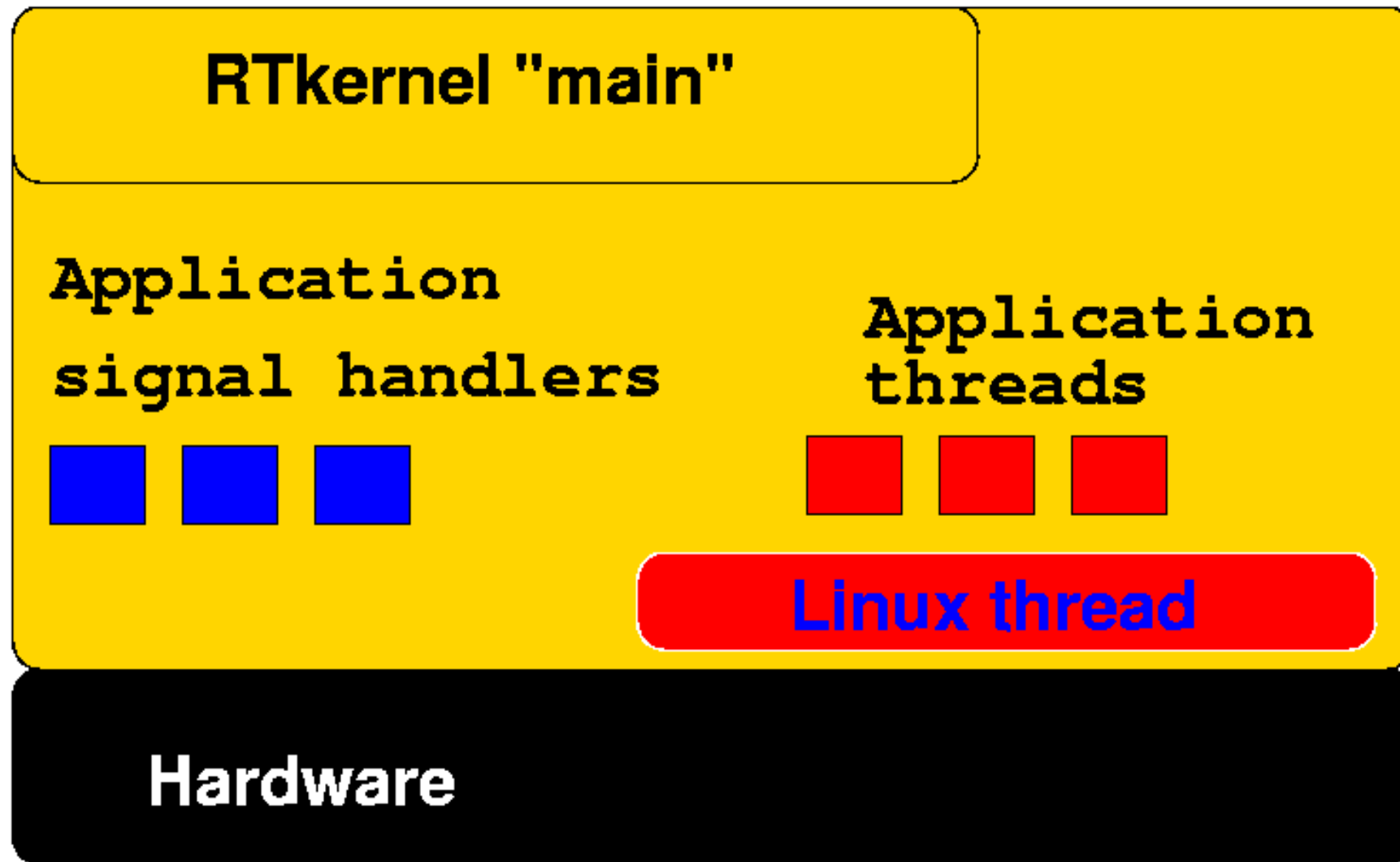
POSIX PSE51 Minimal realtime system

RTLinux Realtime POSIX Process

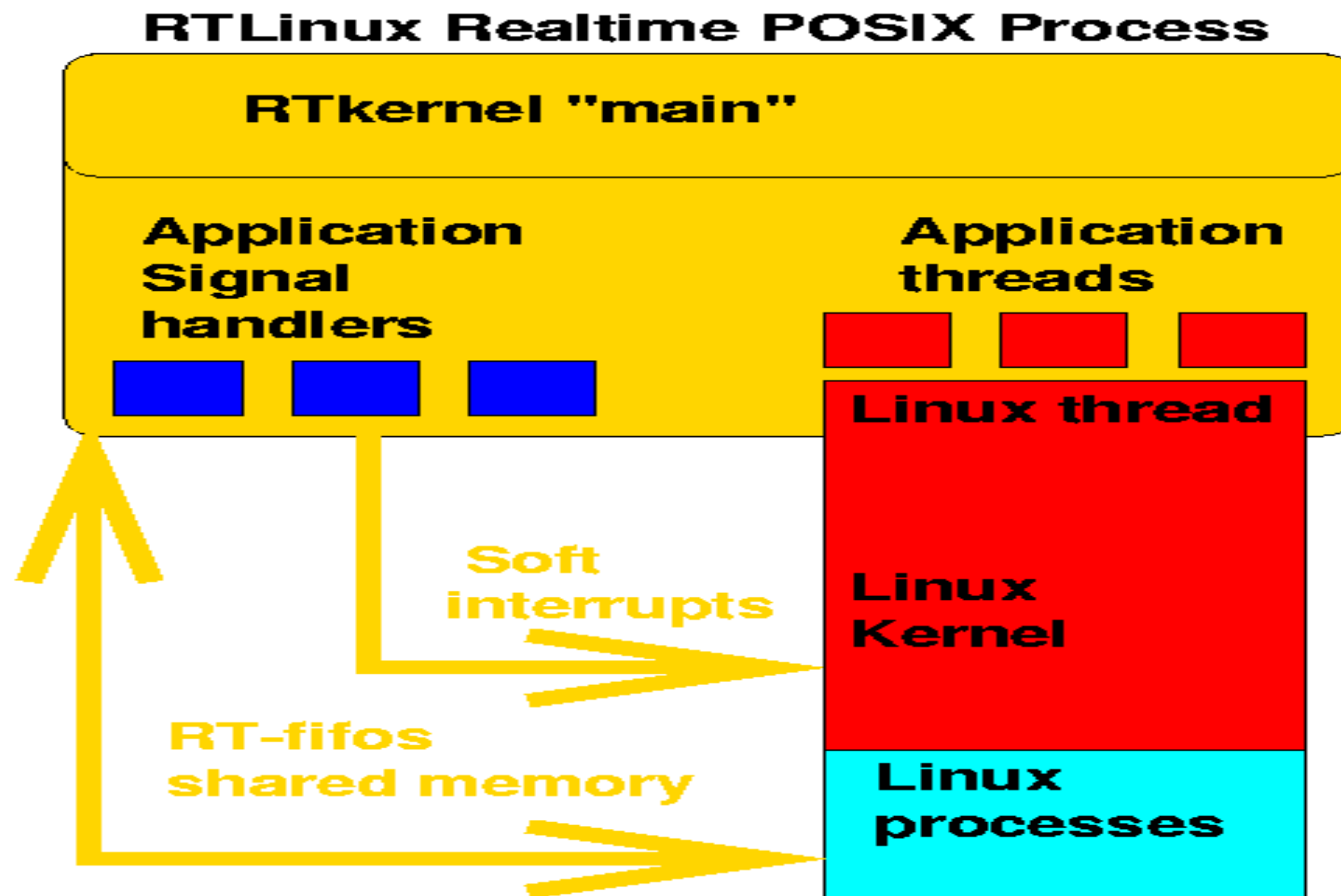


Linux is a thread

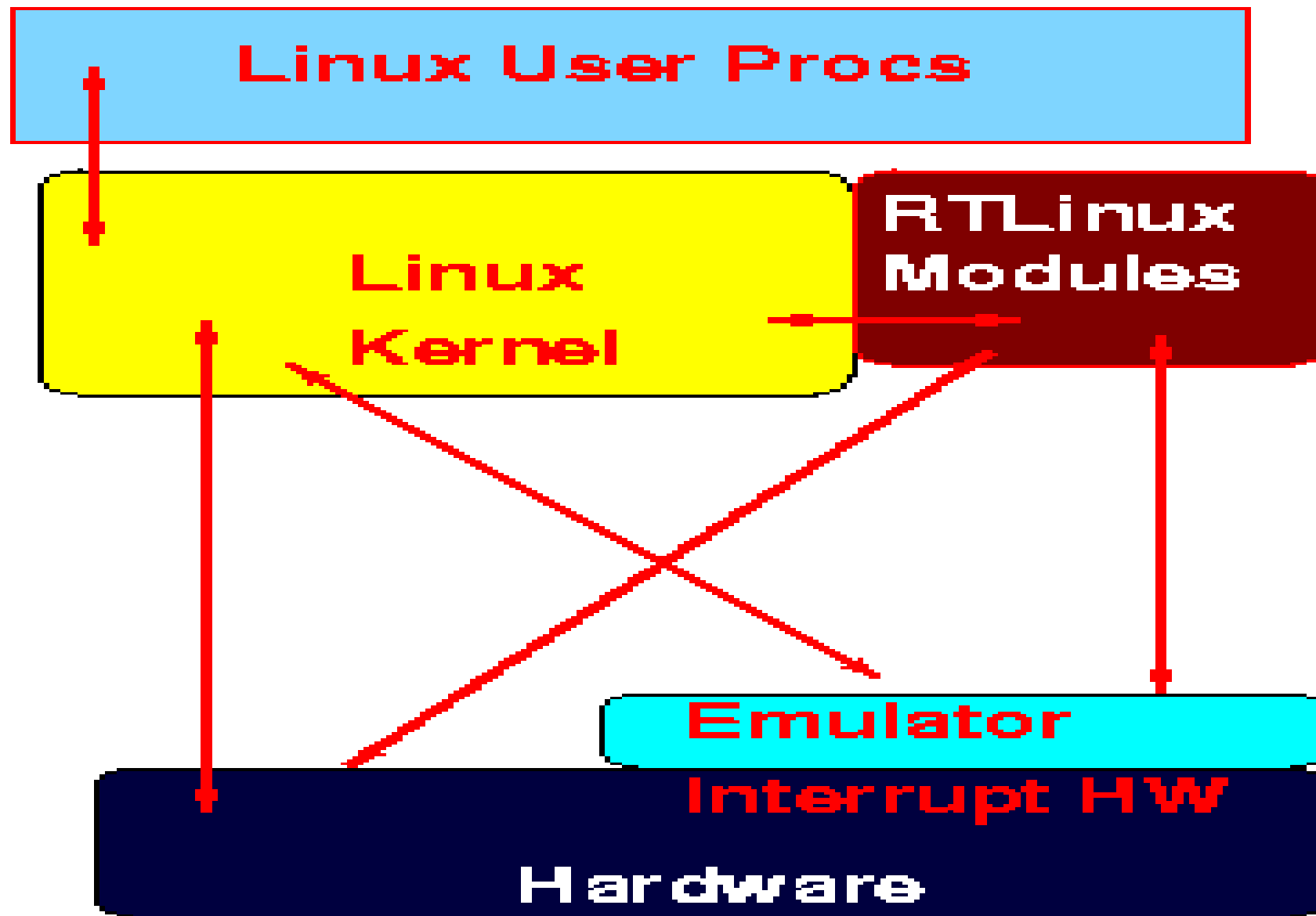
RTLinux Realtime POSIX Process



In detail



RTLinux block diagram



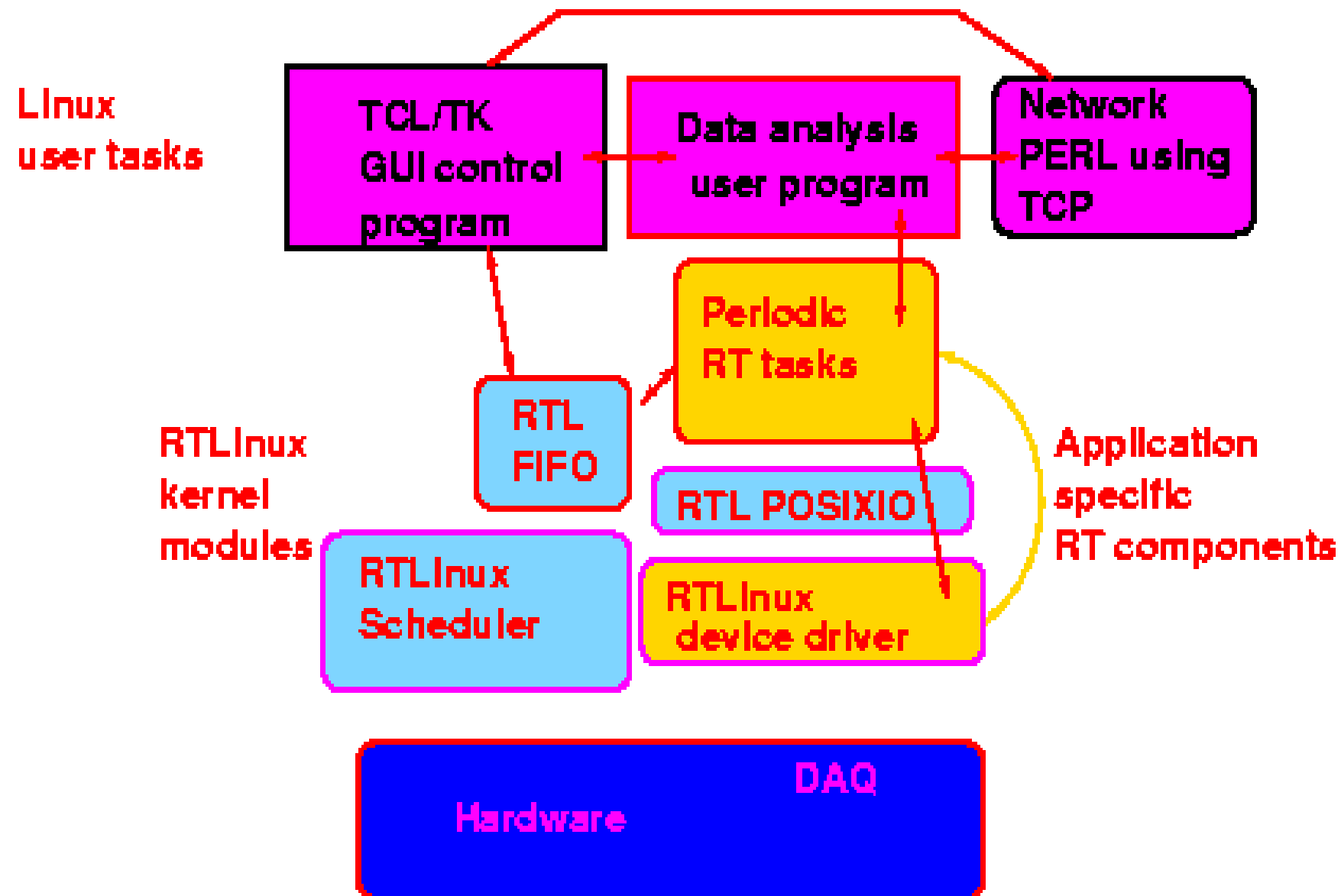
Design principles

1. The RT part must be simple and fast.
 - Decouple RT and general purpose operating systems.
 - Decouple RT and general purpose parts of applications.
2. Make Linux do the hard stuff
 - Device initialization and any non-rt drivers
 - Networking, gui's, file systems, databases ...

Results: x86, PowerPC, Alpha

- Under 15microsecond interrupt latency – HW bounded.
- Under 30microsecond jitter on periodic tasks – HW bounded.
- Embeds like Linux: MiniRTL comes on a floppy and runs in 4Meg.

A typical simple application



The simplest user side data logging program

```
cat < /dev/rtf0 > logfile
```

API

- POSIX-threads “lite”. Familiar, yet efficient.
- RT tasks are threads in a single RT process per processor.
- Follow POSIX 1001.13 “Minimal RT Application Environment Profile” plus some simplifications
- Alternative APIs are easy to accommodate: ITRON is a candidate.

API

- Linux itself follows the PSE54 (mostly).
- So: there are processes and subthreads in the non-RT thread.

Current uses

- Aircraft simulators, PBXs, video editors, robots, instrumentation, telecom switches, telescopes, ...

Current users include

- Kaiser Aluminum, Oregon Cutting: rolling mills, chain saw chains.
- NASA: instrumentation and control.
- Jim Henson Muppet Factory: monsters and other animations/animatronics.
- Lang: 3D metal engraving machines.
- Alcatel: PBXs.

Possibilities

1. High speed networking/industrial control using SMP boards and ethernet.
2. RTLinux on a disk drive: replacing custom firmware and controlling storage/network data movement.
3. E-commerce failover with no transaction loss.

Good POSIX

- Basic model is perfect.
- Extends to smp easily: process per CPU.
- Coherent "story" for programmers.

Additions

- `pthread_setfp_np(thread)` *enable FP.*
- Added a thread "cpu" attribute for SMP.
- `pthread_make_periodic_np(thread, start, period)`
- `pthread_linux()`

Thinking about it

- `sigaction` to insert a interrupt handler?
- `sa_focus` for interrupt focus.

Flexibility

- `pthread_kill(pthread_linux(), IRQBASE+i)`
- `pthread_join(pthread_linux())`
Wait for Linux to crash! (Alan Cox).

Flexibility 2

- `sigqueue(process,n,v)`
Send Linux on `cpu==process` signal "n".
- `pthread_suspend_np(pthread_linux())`
So we can run RT in L2.

Subtractions

- Much config variable use to make slow parts optional.
- Example: `CONFIG_RTL_SLOW_POSIX_SIGNALS`
- Example: `CONFIG_RTL_SLOW_POSIX_SCHED`

Subtractions 2

- No priority inheritance: not compatible with hard RT.
- Ignore scheduling parameters unless asked.

Subtractions 3

- Clocks are baroque.
- Signals cannot be fully implemented efficiently.
- “SHARED” cross cpu operations: *mutex_unlock* are inherently slow.

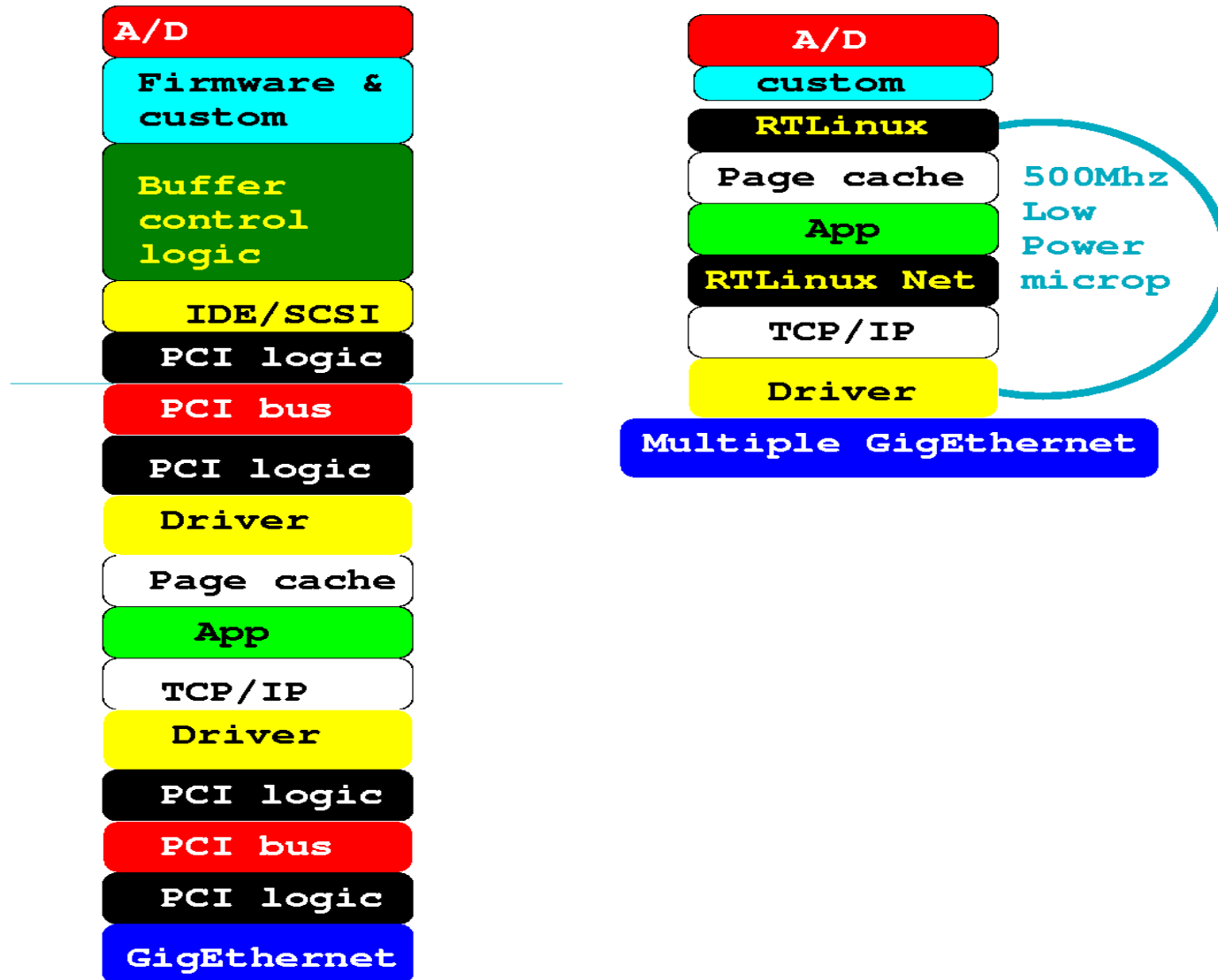
Open source

1. Core technology is GPL. Fully functional system is free on the net.
2. Royalty free.
3. FSM Labs sells non-GPL licenses, customizes, and supports.

www.fsmlabs.com

www.rtlinux.com

Serving data to a network



Legacy vs. RTLinux disk drives