FSMLabs RTLinux technology for hard realtime

Victor Yodaiken

`yodaiken@fsmlabs.com`

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

Outline

1. **Hard, Soft, and Imaginary Realtime.**
2. Hard realtime is important and difficult.
3. RTLinux technology.
4. RTLinux Application Programming model.
5. RTLinux applications and users.
6. Technology roadmap.

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").

Hard realtime

1. *Predictable*: Collect data every 100 microseconds. Even if Netscape comes up.

2. *Low latency*: Respond to compact PCI card dead signal within 20 microseconds.

3. *Worst Case not "typical"*

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.

Outline

1. Hard, Soft, and Imaginary Realtime.
2. **Hard realtime is important and difficult.**
3. RTLinux technology.
4. RTLinux Application Programming model.
5. RTLinux applications and users.
6. Technology roadmap.

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.

You don’t need hard realtime.

1. If ”typical” or ”average” performance is all that matters.

2. If you can run slowly.

3. Dropping 10 video frames every now and then is ok.

4. Losing an e-commerce transaction is no problem.

5. Tearing up a metal working machine is fun.

6. Fine to have pops and squeaks ok in voice-over-IP

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.

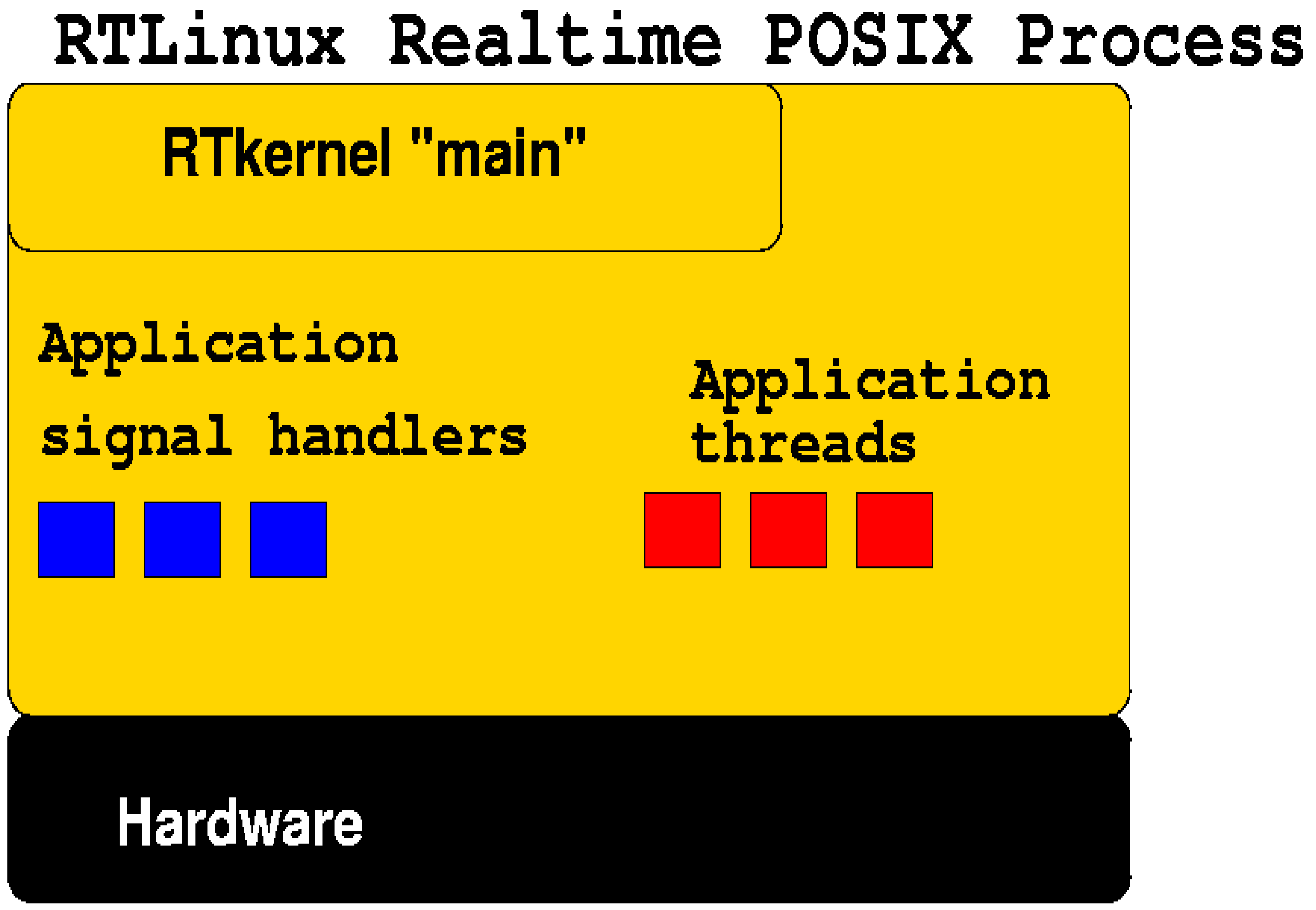
Outline

1. Hard, Soft, and Imaginary Realtime.
2. Hard realtime is important and difficult.
3. **RTLinux technology.**
4. RTLinux Application Programming model.
5. RTLinux applications and users.
6. Technology roadmap.

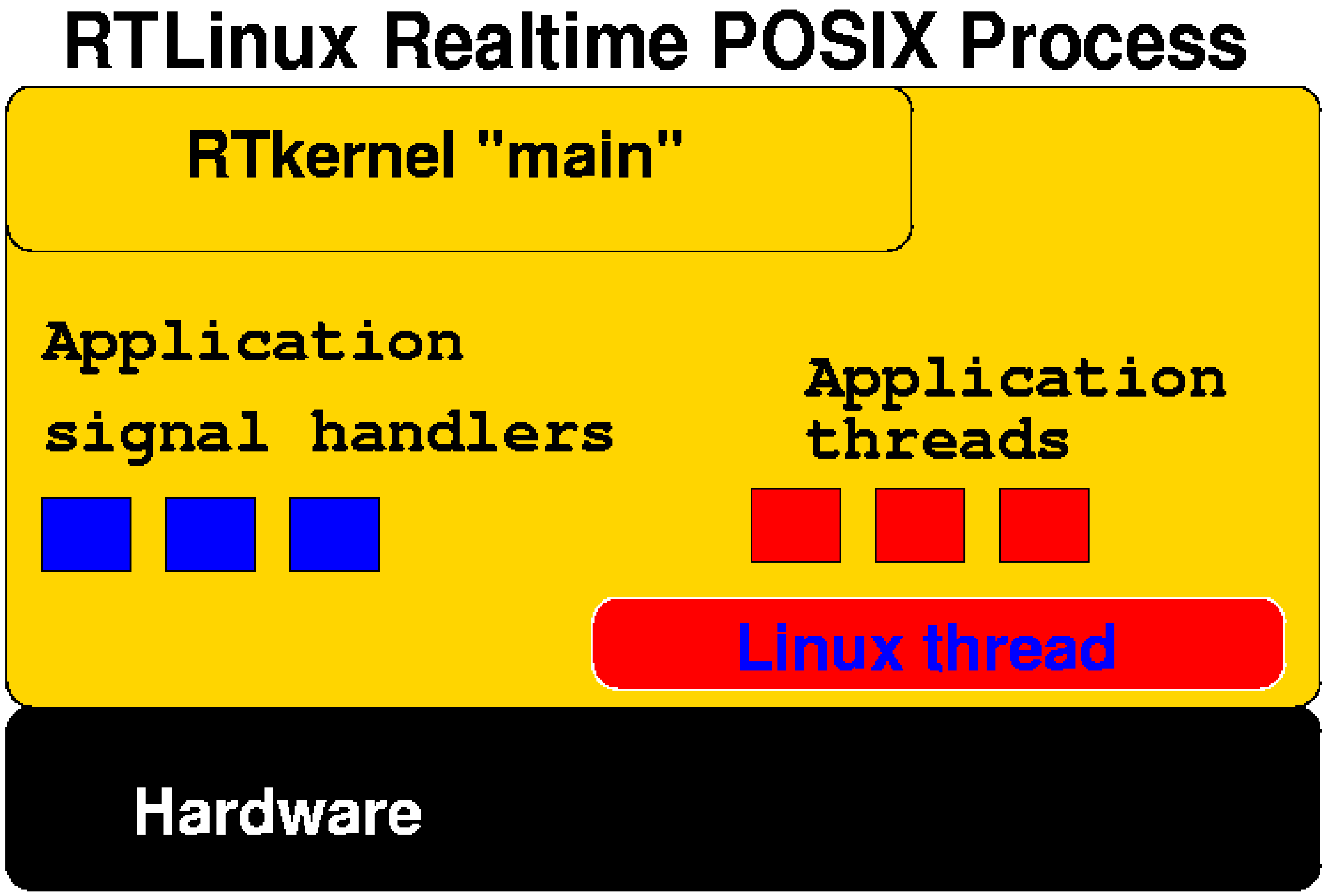
RTLinux is a hard realtime operating system

RTLinux follows the POSIX PE51 1001.13 RT profile.

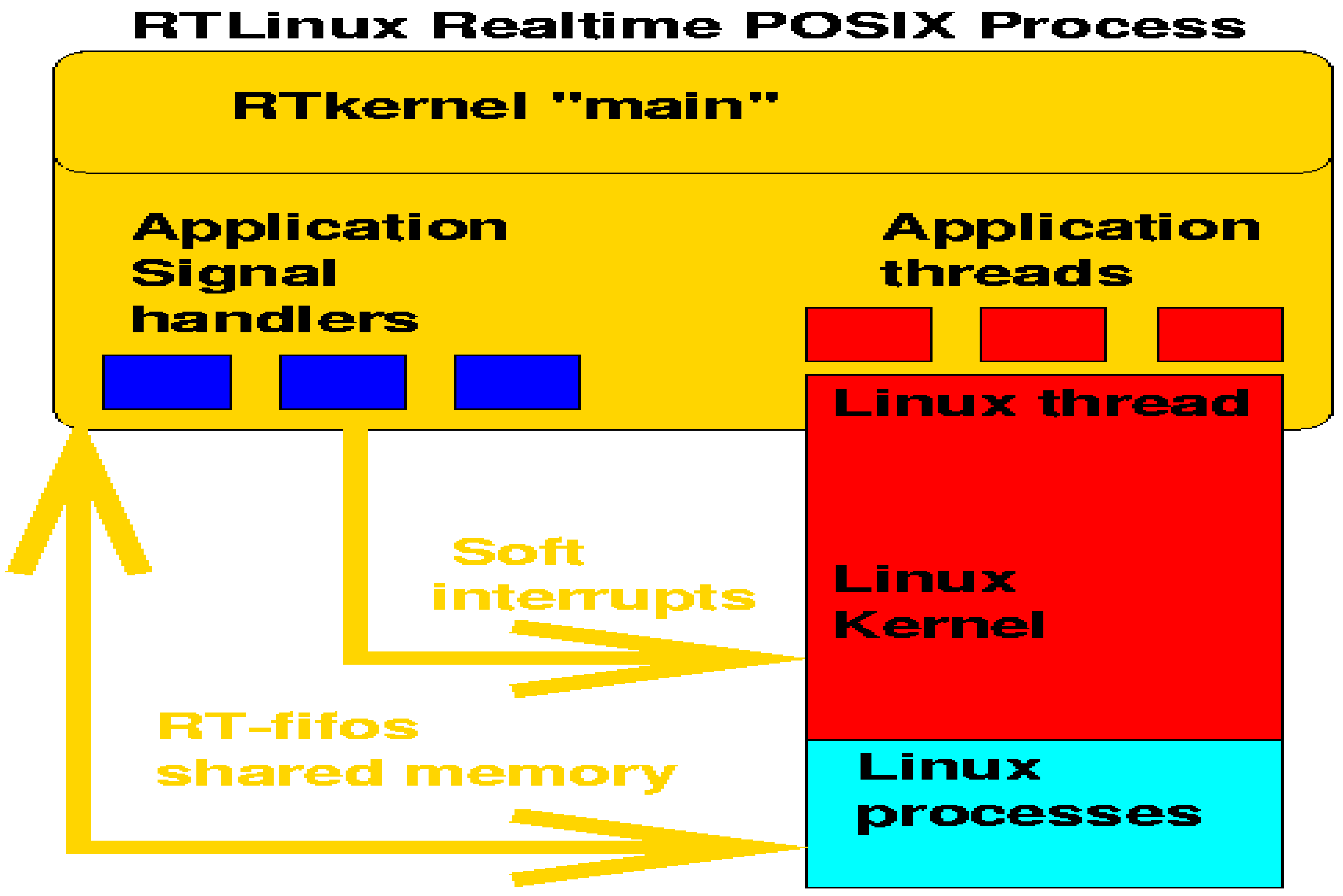
POSIX PE51 Minimal realtime system



Linux is a thread



In detail


RTLinux Realtime POSIX Process
RTkernel "main"
Application Signal handlers
Application threads
Linux thread
Linux Kernel
Linux processes
Soft interrupts
RT-fifos shared memory

RTLinux basic mechanism: US Patent 5,995,745.

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 thread.

Emulation

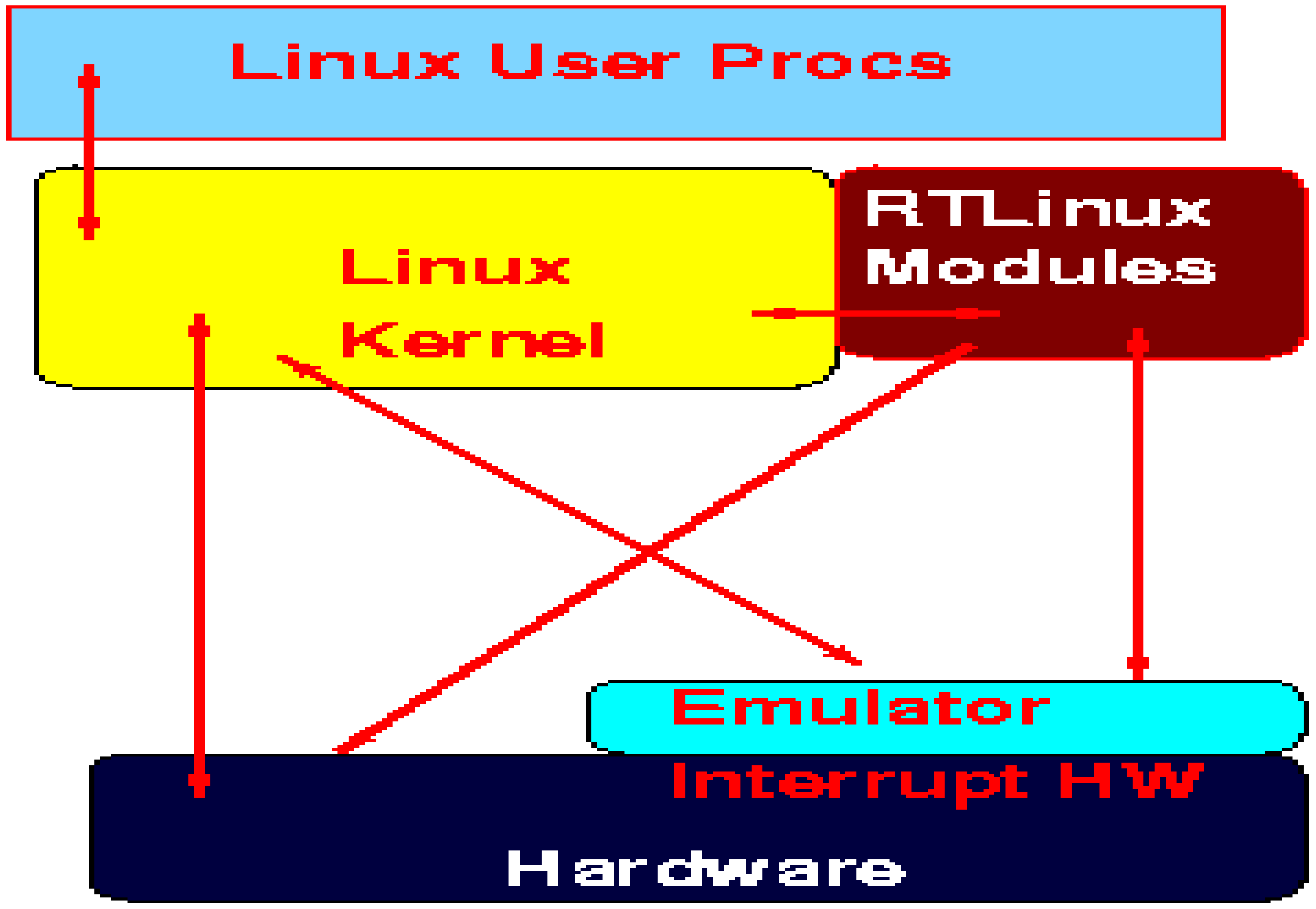
Linux `cli` does “`clear soft enable bit`”.

Linux `sti` does “`set soft enable bit: emulate pending interrupts`”.

On an interrupt:

```
if there is a RT handler: call it.
if there is a linux handler AND RT is idle
                               AND soft enabled
     call the Linux handler
else mark the interrupt pending.
```

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 ...

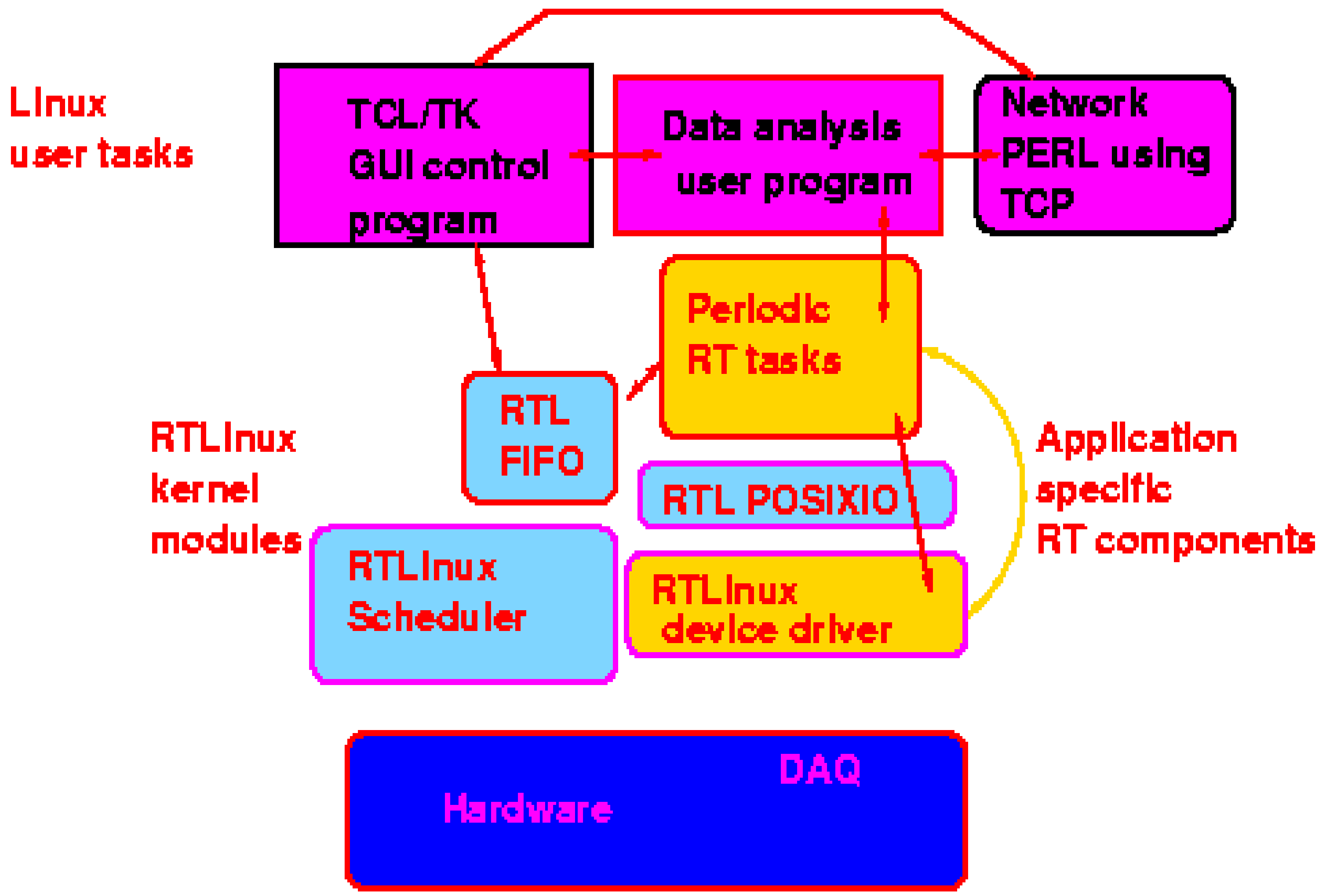
Results: x86, PowerPC, Alpha

- 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.

Outline

1. Hard, Soft, and Imaginary Realtime.
2. Hard realtime is important and difficult.
3. RTLinux technology.
4. **RTLinux Application Programming model.**
5. RTLinux applications and users.
6. Technology roadmap.

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 accomodate: ITRON is a candidate.

Hard realtime at the low microsecond level +full general purpose OS.

1. Microprocessors replace custom hardware.

2. Software layers collapse.

3. Move from asynchronous to synchronous protocols.

Outline

1. Hard, Soft, and Imaginary Realtime.
2. Hard realtime is important and difficult.
3. RTLinux technology.
4. RTLinux Application Programming model.
5. **RTLinux applications and users.**
6. Technology roadmap.

Modest examples

1. Motor control using 200Mhz Pentium and parallel port replaces $15,000 data acquisition board.

2. Machine tool control with PCs instead of VME crates.

3. Puppet control on a laptop using a $200 DAQ card.

Current uses

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

Current users include

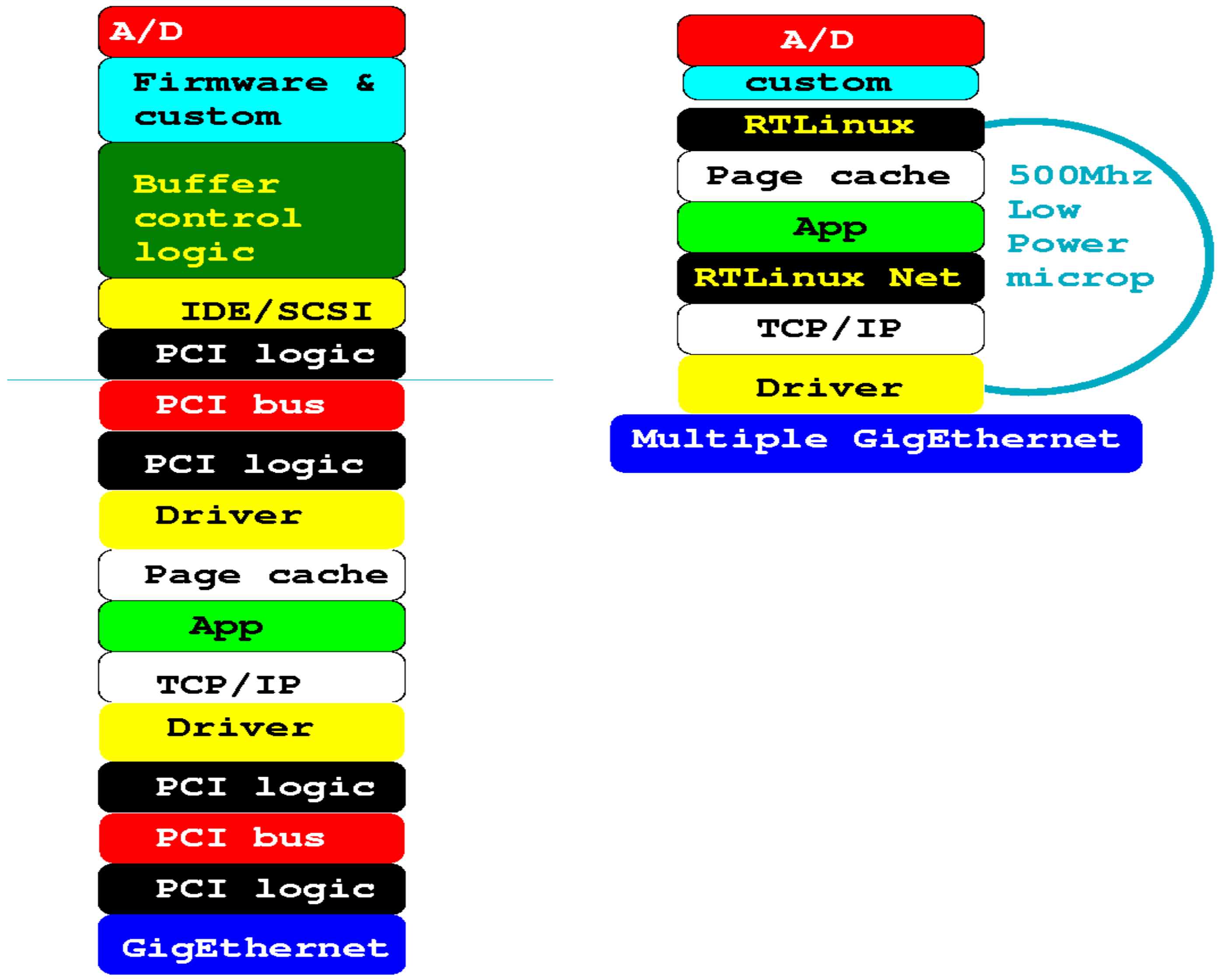
- Kaiser Aluminum, Oregon Cutting: rolling mills, chain saw chains.
- NASA: instrumentation and control.
- Jim Henson Muppet Factory: monsters and other animations/animatronics.
- Lang: 3D metal engraving machines.
- Alcatel: PBXs.

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.



Legacy vs. RTLinux disk drives

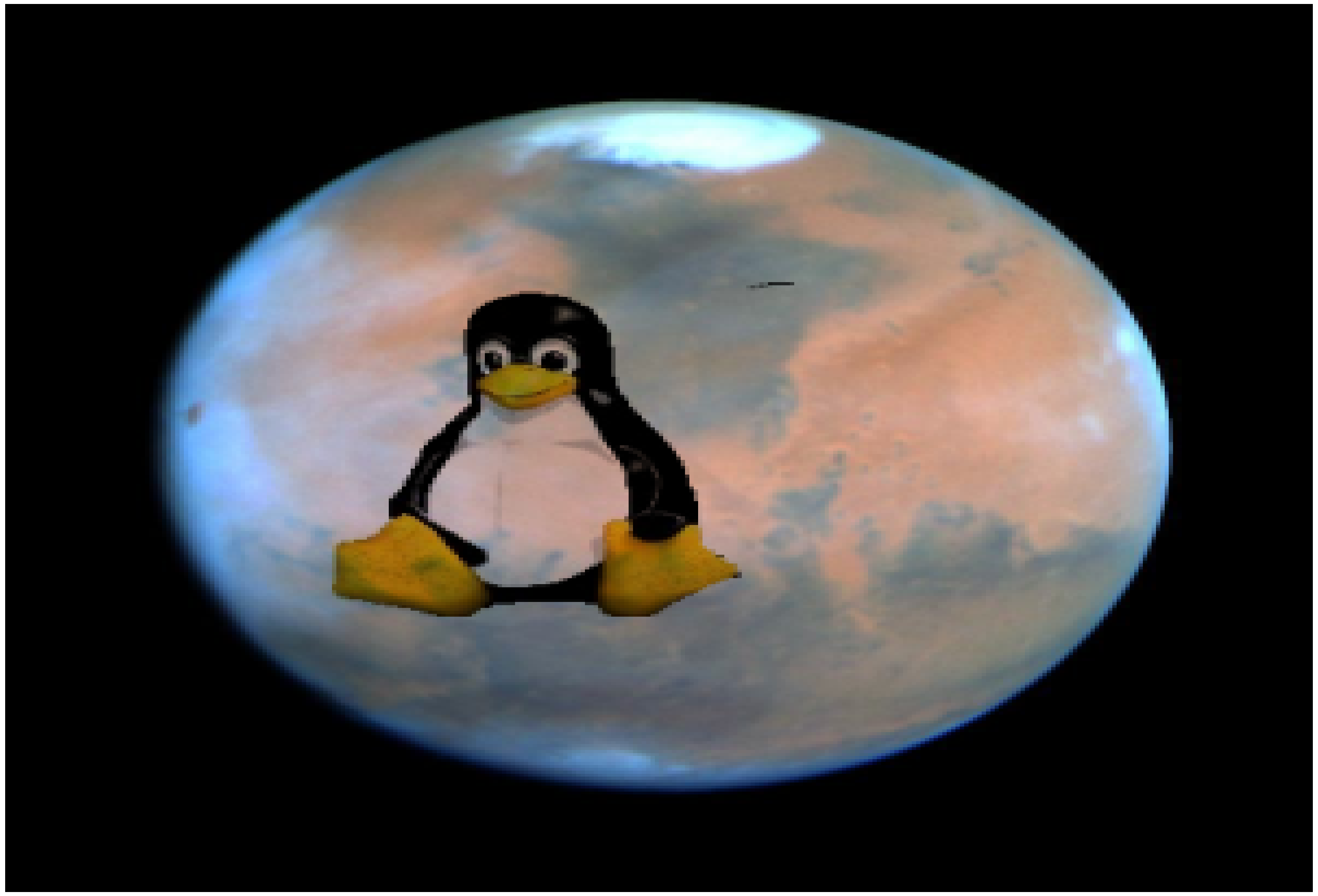
Serving data to a network

Outline

1. Hard, Soft, and Imaginary Realtime.
2. Hard realtime is important and difficult.
3. RTLinux technology.
4. RTLinux Application Programming model.
5. RTLinux applications and users.
6. **Technology roadmap.**

Overall Strategic Objective

- To make RTLinux the standard RTOS for everything from factory floor to cell phones.



Base functionality: partially there

1. RPMs for the most popular distributions.
2. A supported CD distribution.

SMP enhancements: done

1. Interrupt focus.
2. Linux eviction.

Networking: started

1. RT ethernet/etc packet system and software isochronous channels.
2. RT “light” IP layer. QOS hooks for voice over IP and secure IP.
3. RT scheduling of TCP stack.

Data and telecommunications apps

1. Compact PCI fault recovery.
2. Telephony control messaging.
3. Router channel switch applications.
4. Fault tolerance and rommable RTkernel.

Cluster applications

1. Fast failure detect infrastructure.
2. Synchronous data transfer protocols.

And in general:

1. Embedded devices: disks to PDAs.
2. Embedded boards.
3. Routers and switches.
4. Telephony equipment.
5. Clusters.

Open source

1. Core technology is GPL. Fully functional system is free on the net.

2. Royalty free.

3. FSMLabs sells non-GPL licenses.

FSMLabs

1. Core kernel development.

2. Services for GPL software.

3. Related non-GPL software.

www.fsmlabs.com

www.rtlinux.com