technische universität dortmund

fakultät für informatik

# *Embedded & Real-time Operating Systems*

Jian-Jia Chen
TU Dortmund, Informatik 12
Germany

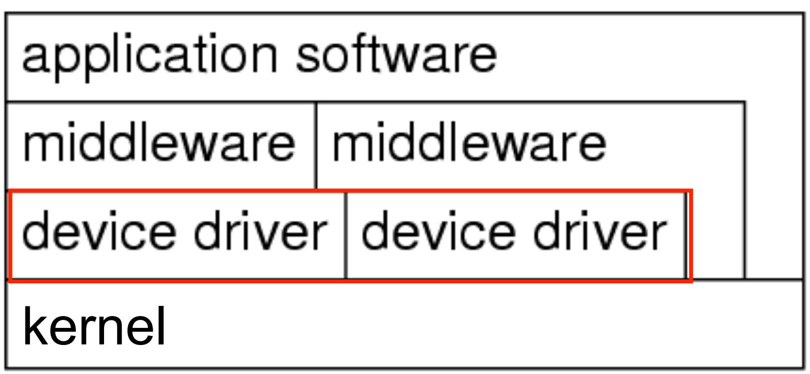
*2015年 06 月 16日*

# *Embedded operating systems*

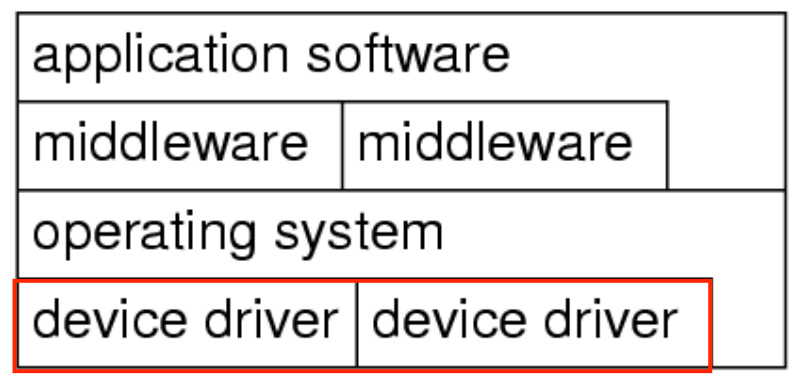
## *- Characteristics: Disk and network handled by tasks -*

- *Effectively no device needs to be supported by all variants of the OS*, except maybe the system timer.
- Many ES without disk, a keyboard, a screen or a mouse.
- Disk & network handled by tasks instead of integrated drivers.

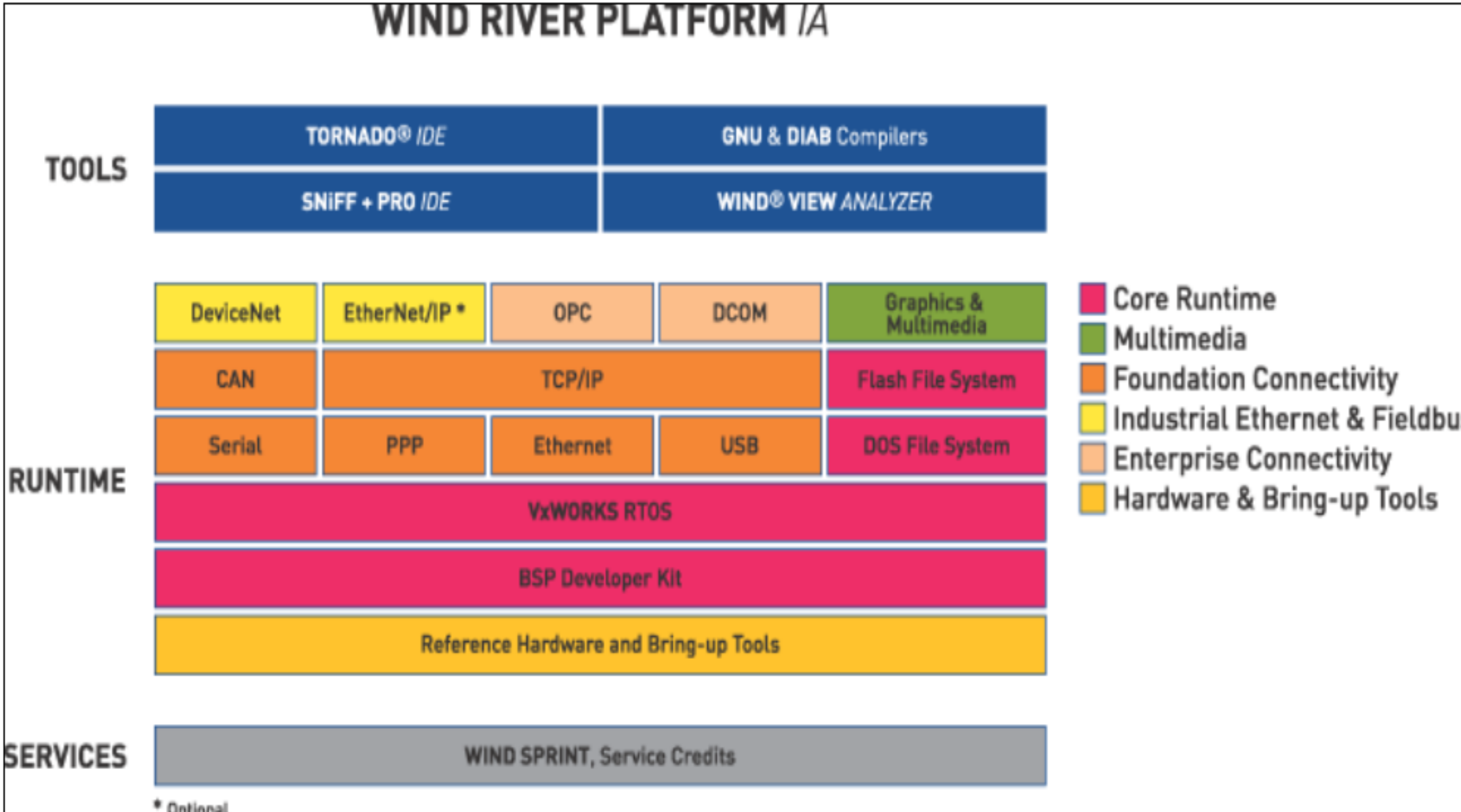
Embedded OS




Standard OS



# *Example: WindRiver Platform Industrial Automation*



# *Embedded operating systems - Characteristics: Protection is optional-*

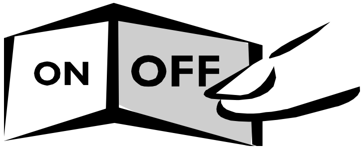
*Protection mechanisms (user mode and privilege mode) not always necessary:* especially for single-purpose ES untested programs rarely loaded, SW considered reliable.

*Privileged* I/O instructions not necessary and tasks can do their own I/O.

Example: Let switch be the address of some switch
Simply use

```
load register,switch
```

instead of OS call.




However, protection mechanisms may be needed for safety and security reasons.

# *Embedded operating systems - Characteristics: Interrupts not restricted to OS -*

*Interrupts can be employed by any process*

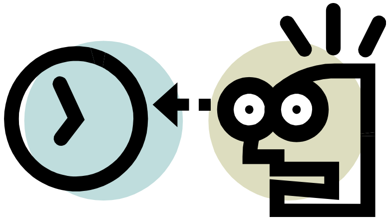
For standard OS: serious source of unreliability.

Since

- embedded programs can be considered to be tested,
- since protection is not always necessary and
- since efficient control over a variety of devices is required,
- it is possible to let interrupts directly start or stop SW (by storing the start address in the interrupt table).
- More efficient than going through OS services.
- Reduced composability: if SW is connected to an interrupt, it may be difficult to add more SW which also needs to be started by an event.

# *Embedded operating systems - Characteristics: Real-time capability-*

Many embedded systems are real-time (RT) systems and, hence, the OSs used in these systems must be *real-time operating systems (RTOSs).*



# *RT operating systems - predictability -*

*Def.: (A) real-time operating system is an operating system that supports the construction of real-time systems.*

*The timing behavior of the OS must be predictable.*

∀ services of the OS: Upper bound on the execution time!

RTOSs must be timing-predictable:

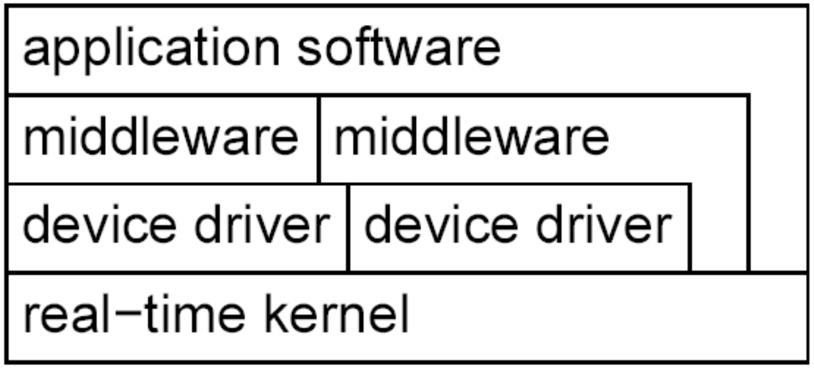
- short times during which interrupts are disabled,
- (for hard disks:) contiguous files to avoid unpredictable head movements.

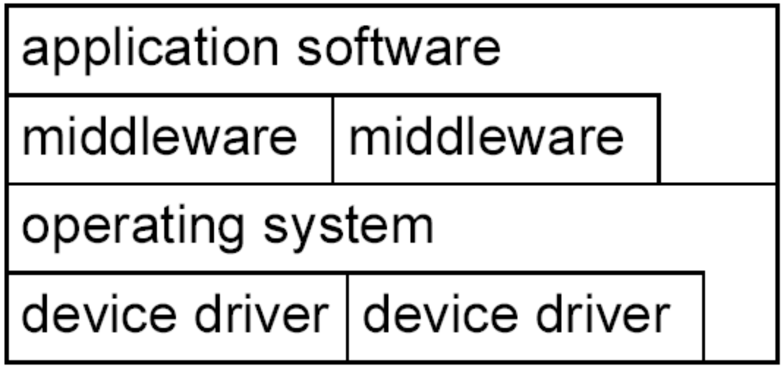
[Takada, 2001]

# *RTOS-Kernels*

*Distinction between*

- real-time kernels and modified kernels of standard OSes.




*Distinction between*

- general RTOSs and RTOSs for specific domains,
- standard APIs (e.g. POSIX RT-Extension of Unix, OSEK) or proprietary APIs.

# *Functionality of RTOS-Kernels*

*Includes*

- processor management,
- memory management,
- and timer management;

} resource management

- task management (resume, wait etc),
- inter-task communication and synchronization.

# *Classes of RTOSes: 1. Fast proprietary kernels*

*For complex systems, these kernels are inadequate, because they are designed to be fast, rather than to be predictable in every respect*

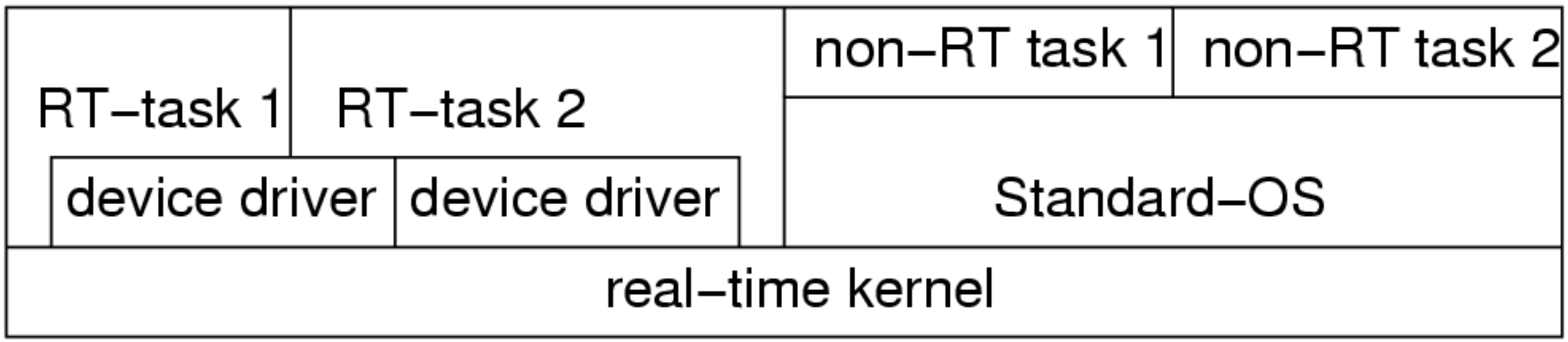
[R. Gupta, UCI/UCSD]

Examples include

QNX, PDOS, VCOS, VTRX32, VxWORKS, FreeRTOS.

# Classes of RTOSs:
# 2. RT extensions to standard OSs

Attempt to exploit comfortable main stream OS.
RT-kernel running all RT-tasks.
Standard-OS executed as one task.

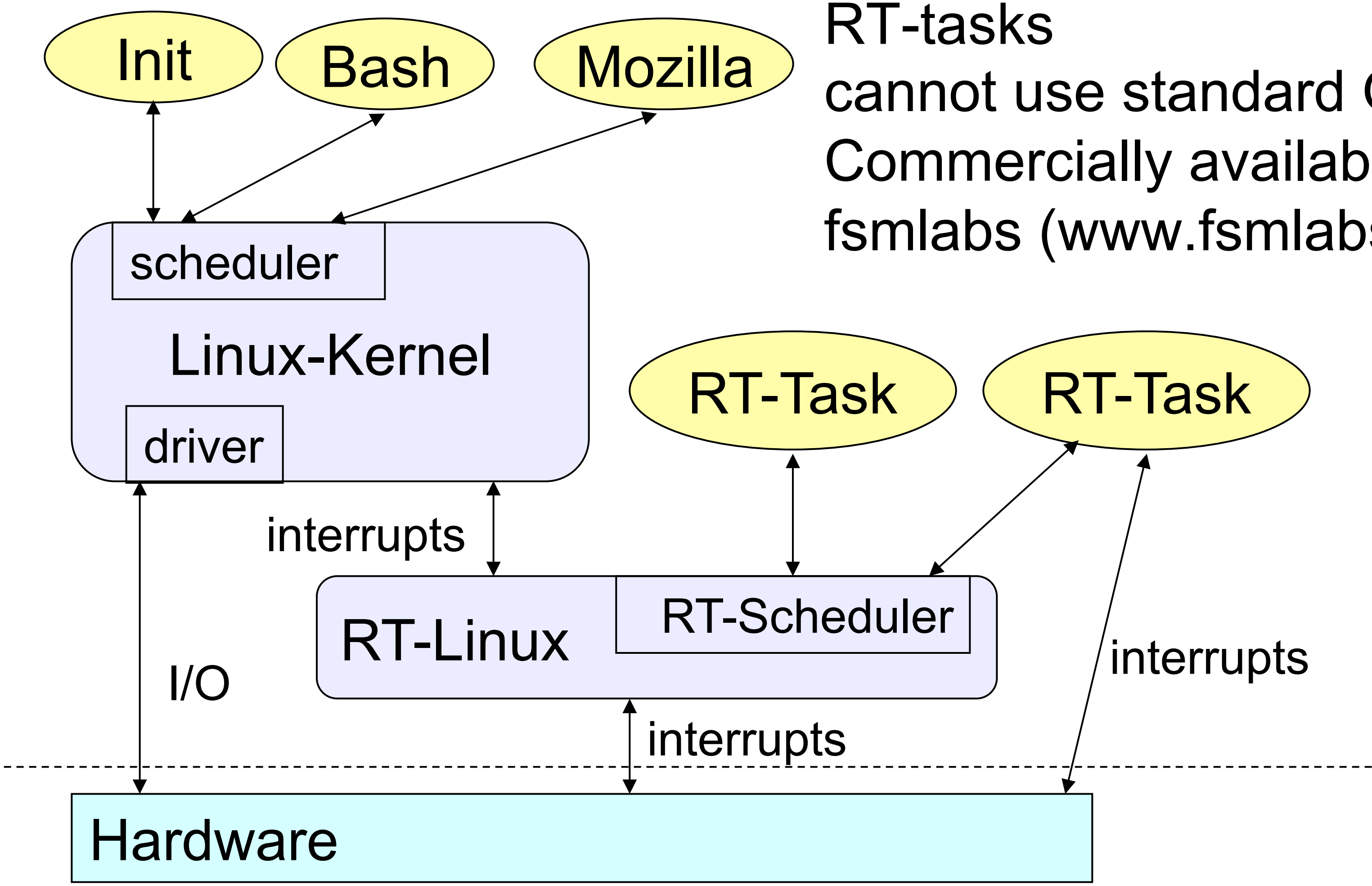



+ Crash of standard-OS does not affect RT-tasks;
- RT-tasks cannot use Standard-OS services; less comfortable than expected

# *RT extensions to standard OSs*

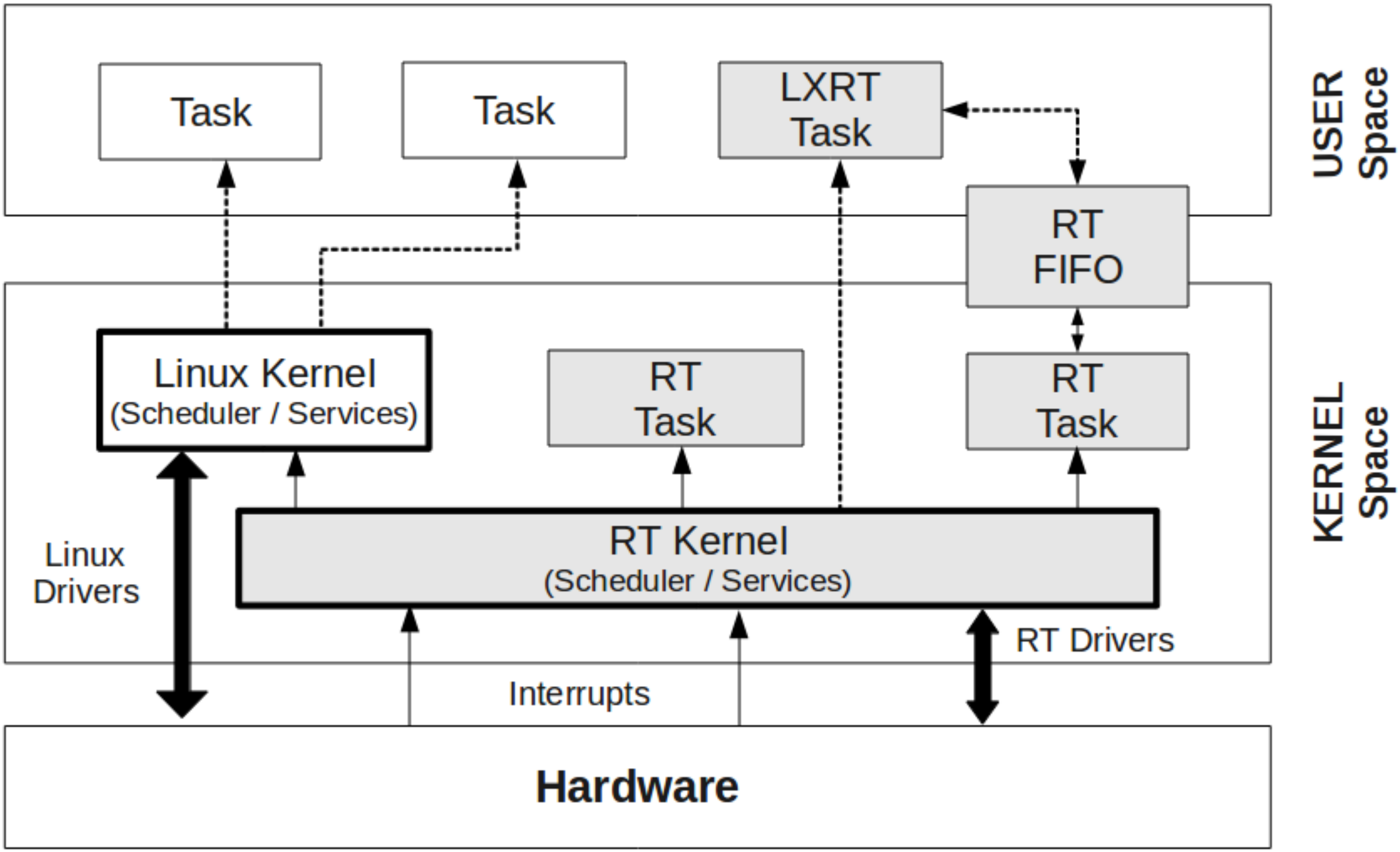
- A common approach is to extend Unix
  - Linux: RT-Linux, RTLinuxPro, RTAI, etc.
  - Posix: RT-POSIX
- Also done for Windows based on virtualization, e.g. RTOSWin, RT-Xen

# Example: RT-Linux




RT-tasks
cannot use standard OS calls.
Commercially available from fsmlabs (www.fsmlabs.com)

# Example (2):
# RTAI – Real Time Application Interface






https://www.rtai.org/

# *Classes of RTOSs:*
# *3. Research trying to avoid limitations*

*Research systems trying to avoid limitations.*
Include MARS, Spring, MARUTI, Arts, Hartos, DARK, and Melody

*Research issues* [Takada, 2001]:

- low overhead memory protection,
- temporal protection of computing resources
- RTOSes for on-chip multiprocessors
- quality of service (QoS) control.



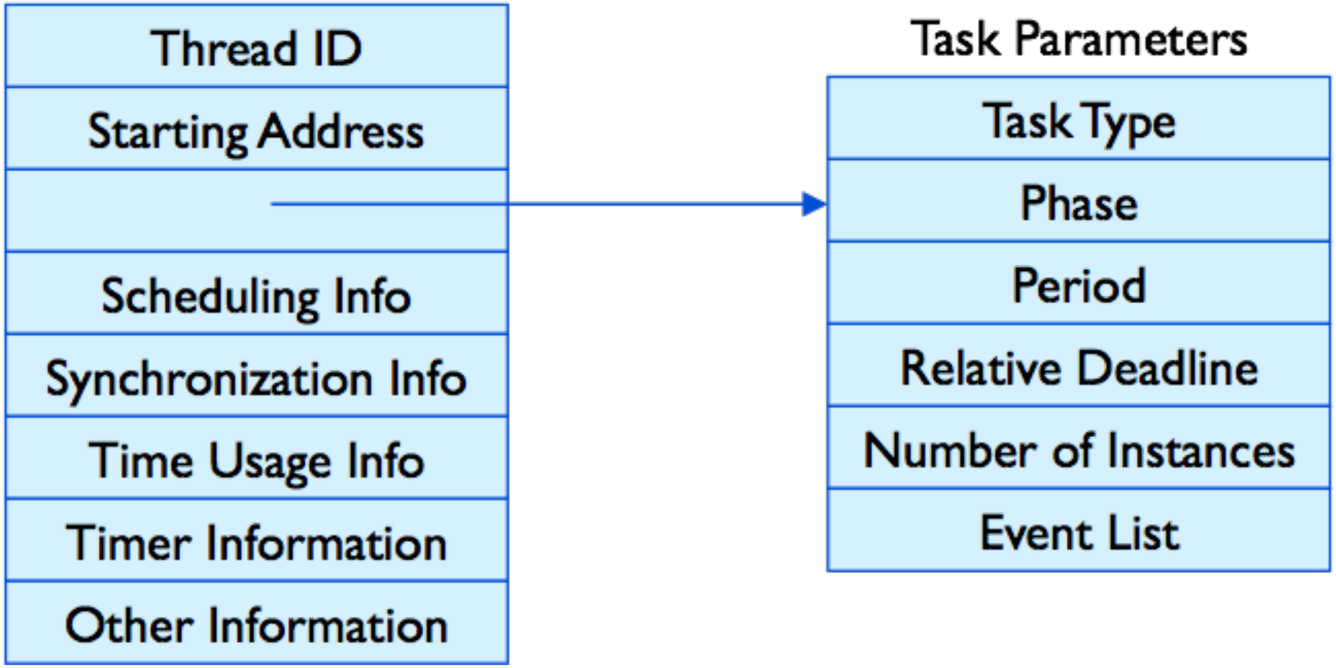
Source: R. Gupta, UCSD

# Task, Thread, Job

- Thread (Job): A basic unit of work handled by the scheduler
- Task: Threads implement the jobs of a task. Usually the same thread is re-used for each job of a task
- Thread Context: The values of registers and other volatile data that define the state and environment of the thread

# *Task, Thread, Job (Continued)*

- TCB: The thread control block (TCB) is the data structure created when the kernel creates a thread
  - The TCB stores the context of the thread when it is not executing



# *Periodic Tasks and Threads*

- *Periodic thread*: Reinitialized by the kernel and put to sleep (i.e., suspends) when the thread completes. Released by the kernel at the beginning of the next period (i.e., becomes ready)
- The task parameters (e.g., phase and period) are stored in a separate manner
- *Most commercial (RT or non-RT) OSs do not support periodic threads*
  - Instead, the thread itself sleeps (i.e., suspends itself via some system call) until the start of the next period after it finishes executing

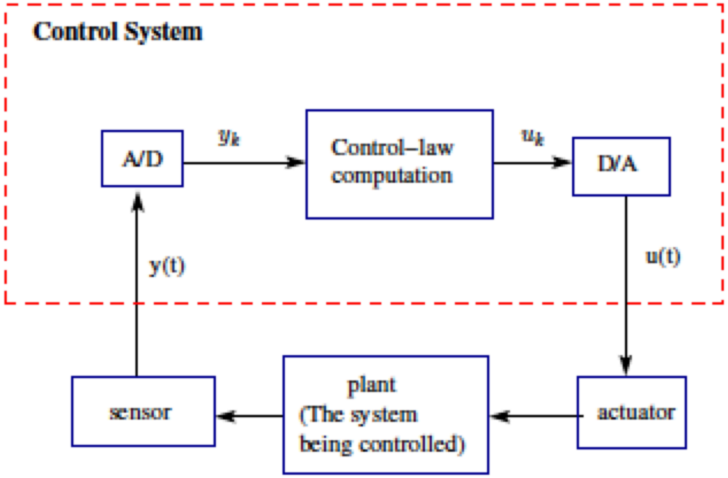
# Example: Control System

**Pseudo-code for this system**

**while (true)**

- start := get the system tick;
- perform analog-to-digital conversion to get *y*;
- compute control output *u*;
- output *u* and do digital-to-analog conversion;
- end := get the system tick;
- *timeToSleep* := $T - (end - start)$;
- sleep *timeToSleep*;

**end while**



# Example: Periodic Control System

**Pseudo-code for this system**

set timer to interrupt periodically with period $T$;

at each timer interrupt

**do**

- perform analog-to-digital conversion to get $y$;
- compute control output $u$;
- output $u$ and do digital-to-analog conversion;

**od**