technische universität
dortmund

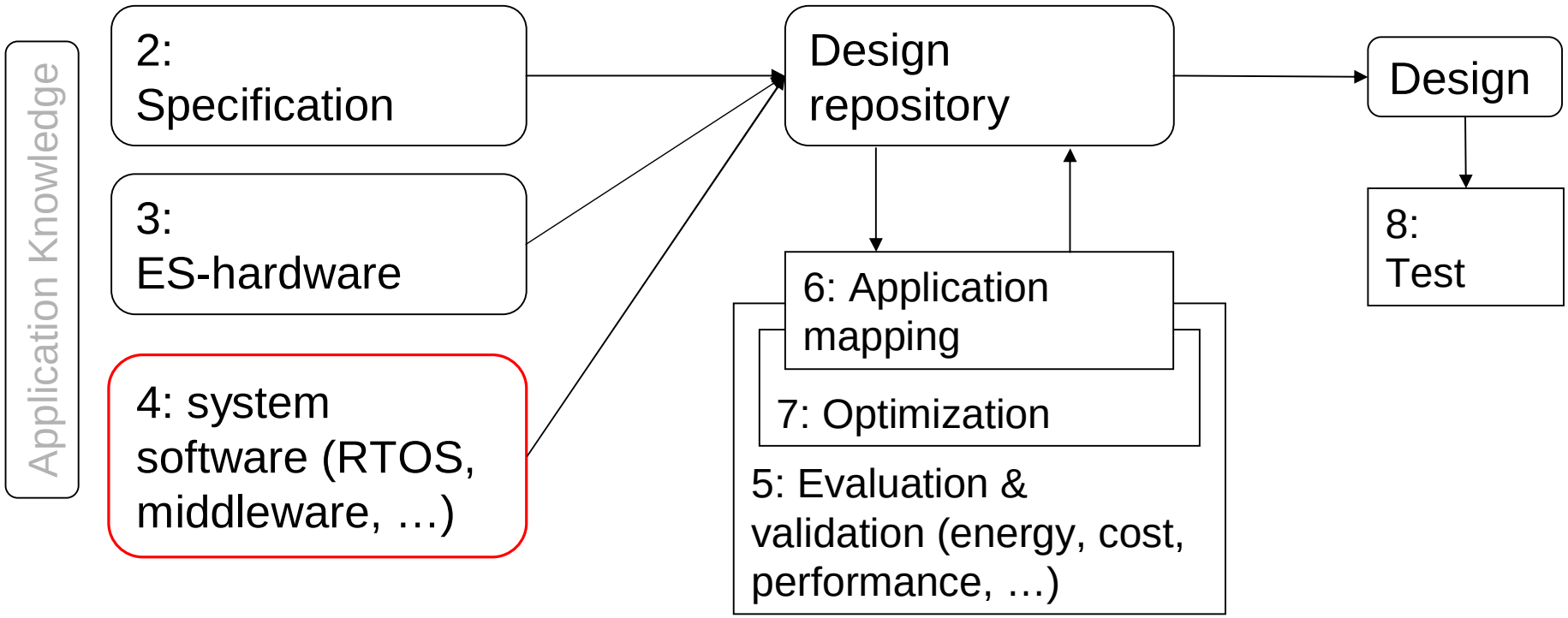
fakultät für
informatik

Embedded & Real-time Operating Systems

Peter Marwedel
TU Dortmund, Informatik 12
Germany

2012年 11 月 27 日

Structure of this course



Numbers denote sequence of chapters

Increasing design complexity + Stringent time-to-market requirements ☞Reuse of components

Reuse requires knowledge from previous designs to be made available in the form of **intellectual property** (IP, for **SW & HW**).

- HW
- Operating systems
- Middleware (Communication, data bases, …)
- ….

Embedded operating systems
- Characteristics: Configurability -

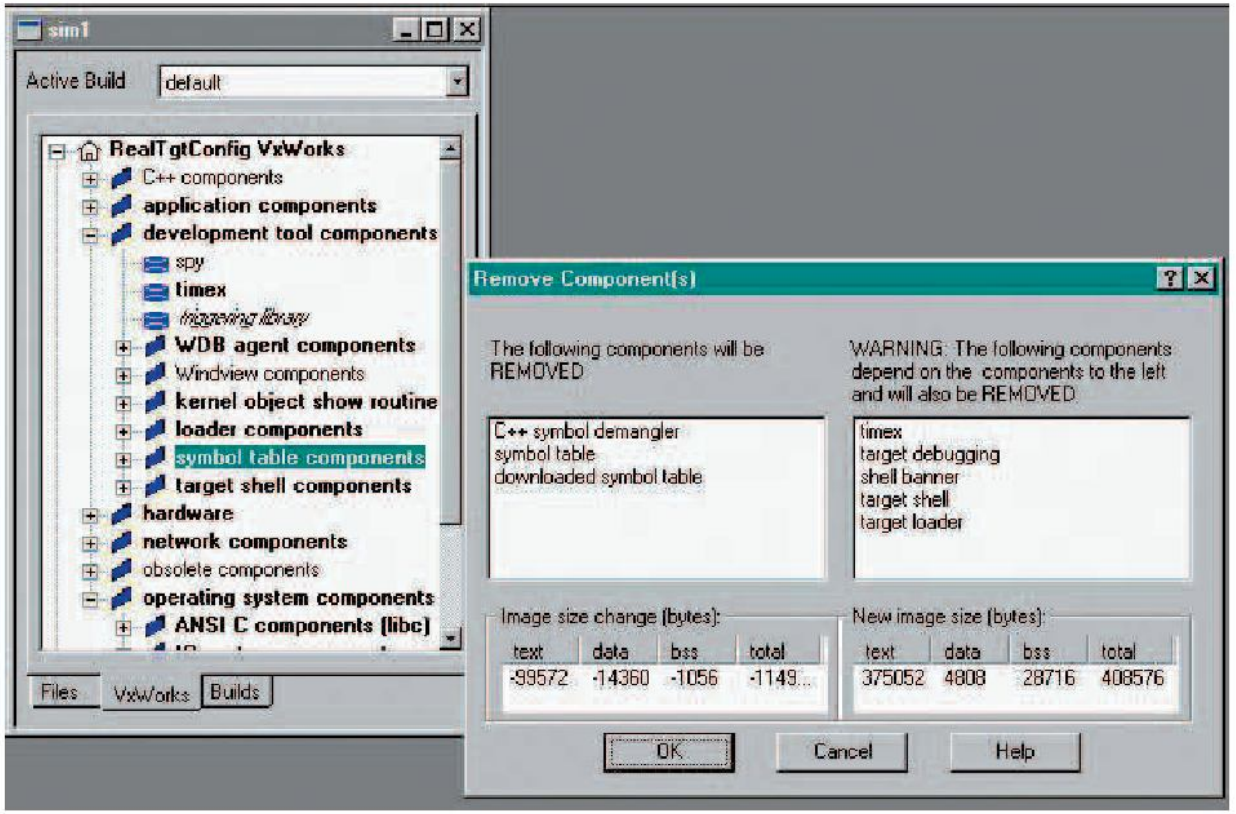
Configurability

No overhead for unused functions tolerated,
no single OS fits all needs, ☞ configurability needed.

- Object-orientation could lead to a of derivation subclasses.
- Aspect-oriented programming
- Conditional compilation (using #if and #ifdef commands).
- Advanced compile-time evaluation useful.
- Linker-time optimization (removal of unused functions)

Dynamic data might be replaced by static data.

Example: Configuration of VxWorks



Automatic dependency analysis and size calculations allow users to quickly custom-tailor the VxWORKS operating system.

http://www.windriver.com/products/development_tools/ide/tornado2/tornado_2_ds.pdf

Verification of derived OS?

Verification a potential problem of systems with a large number of derived OSs:

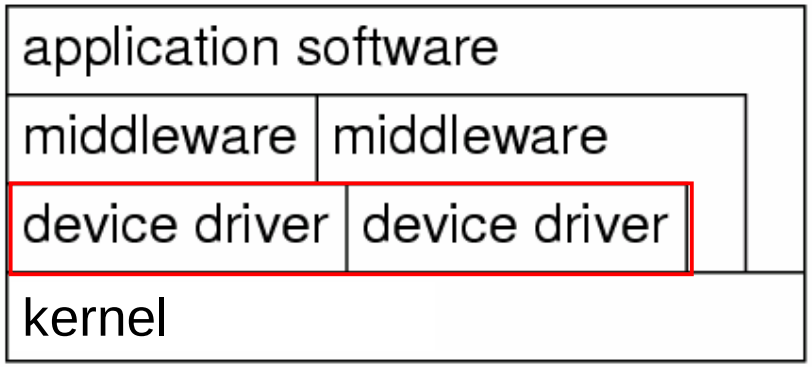
- Each derived OS must be tested thoroughly;
- Potential problem for eCos (open source RTOS from Red Hat), including 100 to 200 configuration points [Takada, 2001].

Embedded operating systems

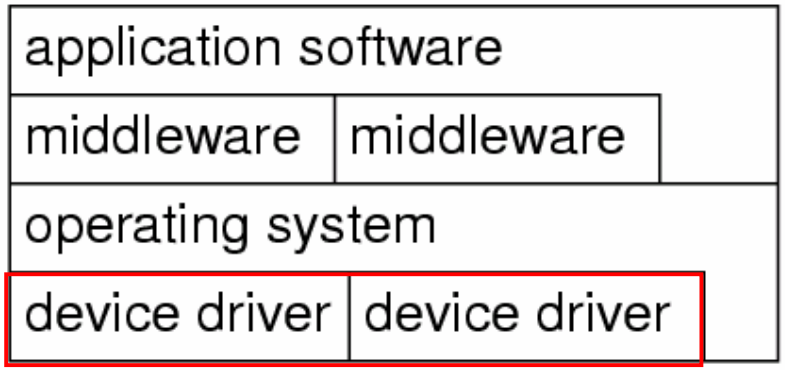
- Characteristics: Disk and network handled by tasks -

- **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.

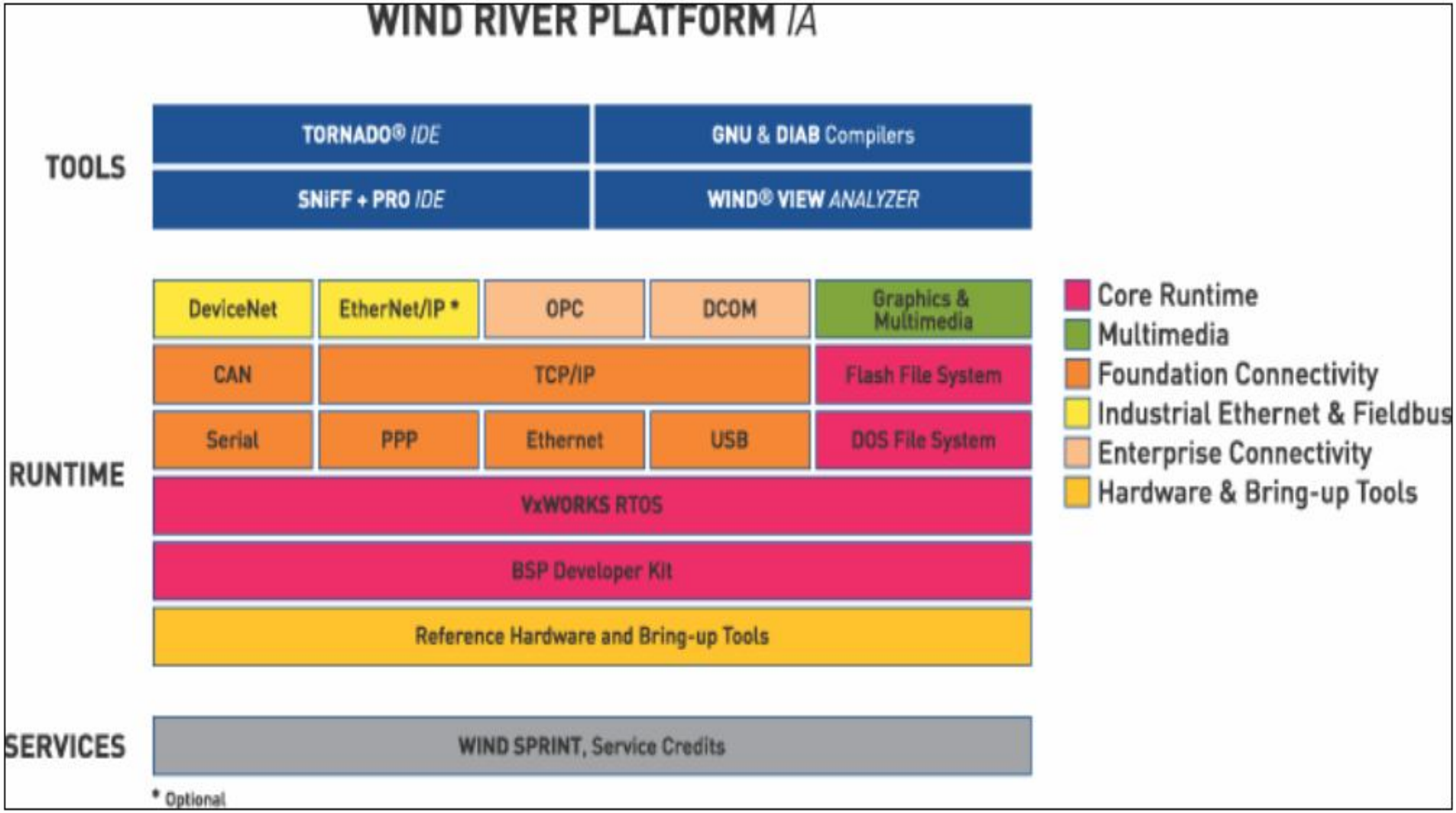
Embedded OS



Standard OS



Example: WindRiver Platform Industrial Automation



Embedded operating systems
- Characteristics: Protection is optional-

Protection mechanisms not always necessary:

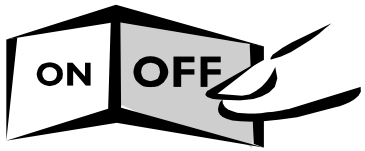
ES typically designed for a single purpose,
untested programs rarely loaded, SW considered reliable.

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

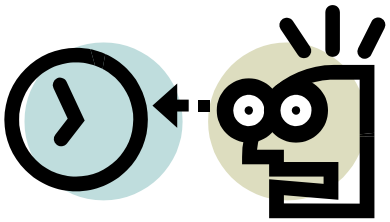
For standard OS: serious source of unreliability.

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 - Definition and requirement 1: predictability -

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

The following are the three key requirements

1. **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]

Real-time operating systems requirement 2: Managing timing

2. OS should manage the timing and scheduling

- OS possibly has to be aware of task deadlines; (unless scheduling is done off-line).
- Frequently, the OS should provide precise time services with high resolution.

[Takada, 2001]

Time

Time plays a central role in “real-time” systems

Physical time: real numbers

Computers: mostly discrete time

- Relative time: clock ticks in some resolution
- Absolute time: wall clock time
  - **International atomic time TAI**
    (french: *temps atomic internationale*)
    Free of any artifacts.
  - **Universal Time Coordinated (UTC)**
    UTC is defined by astronomical standards

  TAI and UTC identical on Jan. 1st, 1958.

  30 seconds had to be added since then.

  Not without problems: New Year may start twice per night.

Internal synchronization

- Synchronization with one master clock
  - Typically used in startup-phases
- Distributed synchronization:

1. Collect information from neighbors
2. Compute correction value
3. Set correction value.

Precision of step 1 depends on how information is collected:

- Application level: ~500 µs to 5 ms
- Operation system kernel: 10 µs to 100 µs
- Communication hardware: < 10 µs

External synchronization

External synchronization guarantees consistency with actual physical time.

Trend is to use GPS for ext. synchronization

GPS offers TAI and UTC time information.

Resolution is about 100 ns.

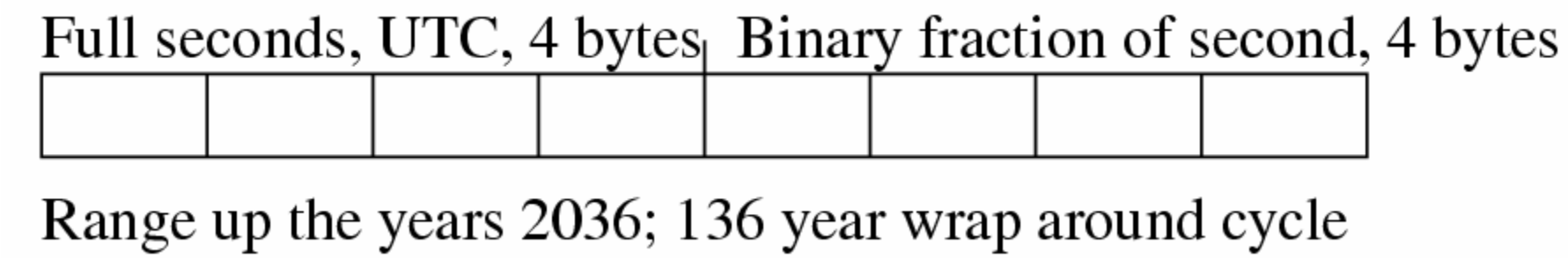
GPS mouse

© Dell

Problems with external synchronization

Problematic from the perspective of fault tolerance:
Erroneous values are copied to all stations.
Consequence: Accepting only small changes to local time.

Many time formats too restricted;
e.g.: NTP protocol includes only years up to 2036



For time services and global synchronization of clocks see Kopetz, 1997.

Real-time operating systems requirement 3: Speed

3. The OS must be fast

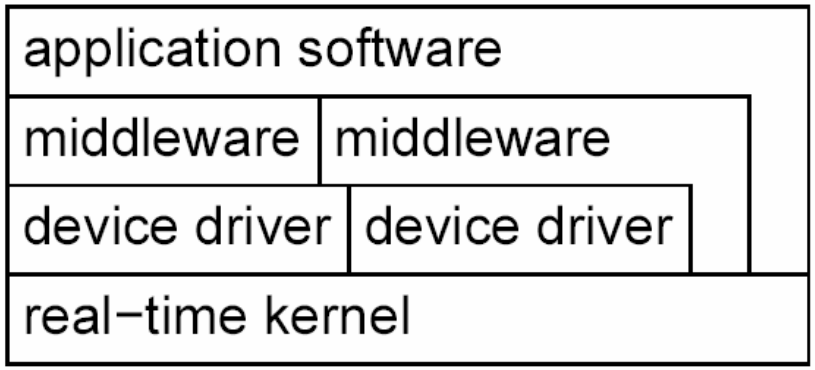
Practically important.

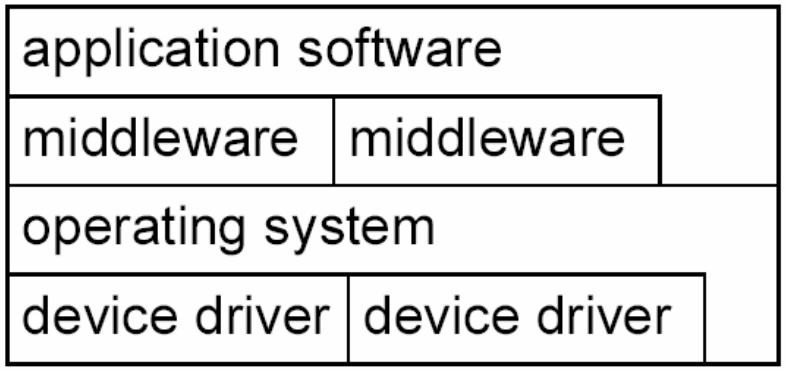
[Takada, 2001]

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, ITRON, 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

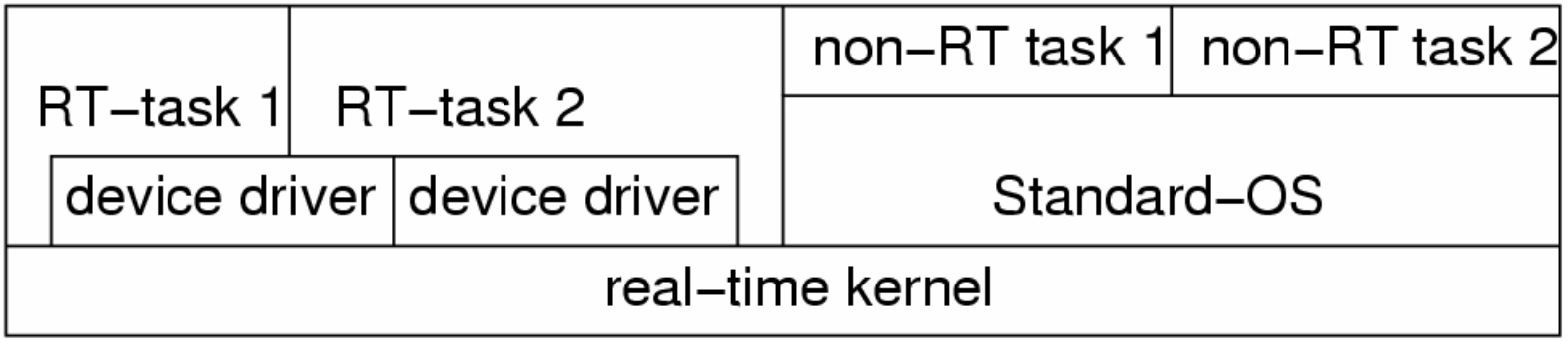
[R. Gupta, UCI/UCSD]

Examples include

QNX, PDOS, VCOS, VTRX32, VxWORKS.

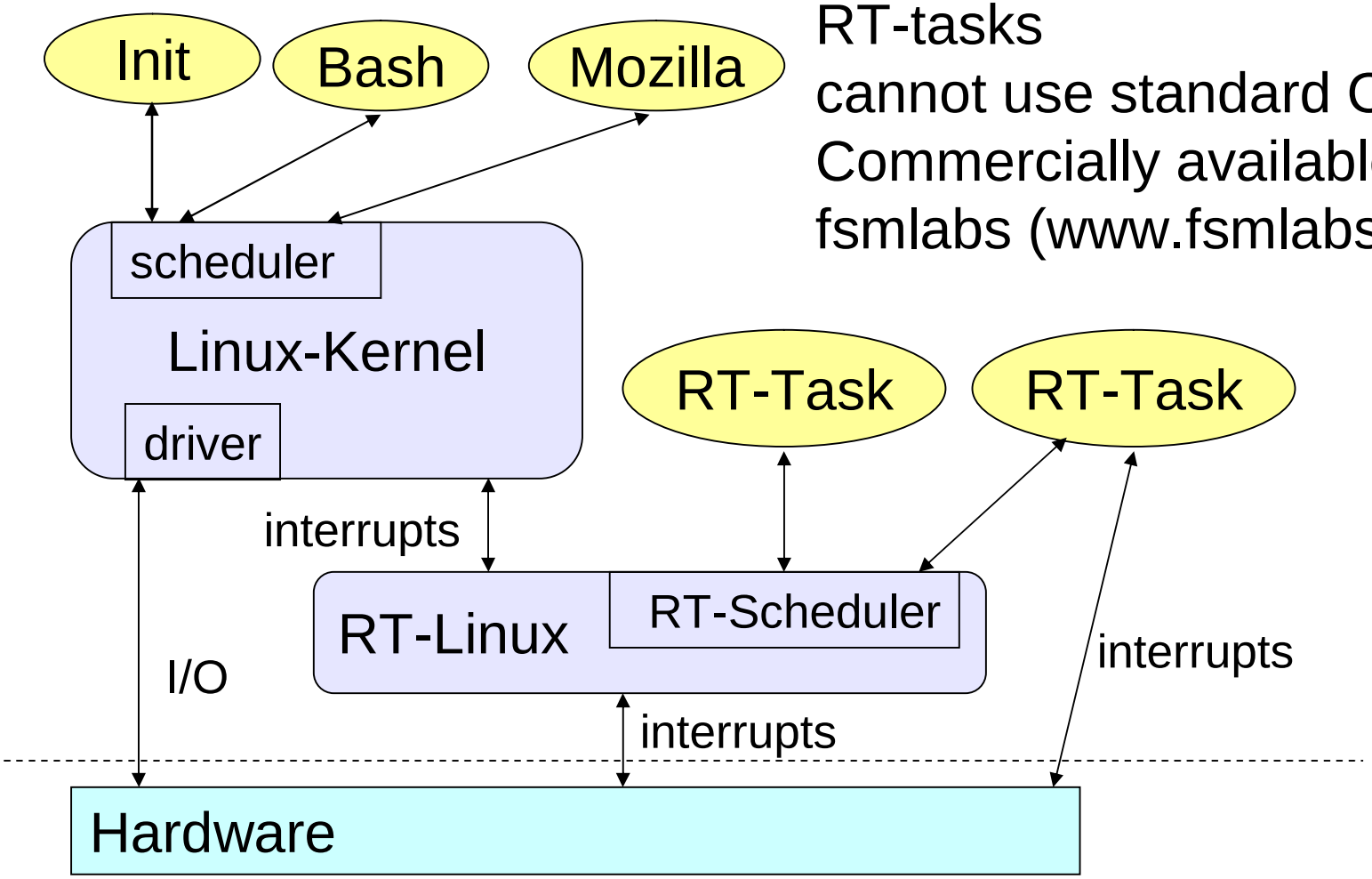
Classes of RTOSs:
2. RT extensions to standard OSs

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

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/

Fixes to many of the sources for unpredictability in Linux

Hardware abstraction layer in between hardware and Linux

Evaluation

According to Gupta, trying to use a version of a standard OS:

not the correct approach because too many basic and inappropriate underlying assumptions still exist such as optimizing for the average case (rather than the worst case), ... ignoring most if not all semantic information, and independent CPU scheduling and resource allocation.

Dependences between tasks not frequent for most applications of std. OSs & therefore frequently ignored.

Situation different for ES since dependences between tasks are quite common.

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
- support for continuous media
- quality of service (QoS) control.

technische universität
dortmund

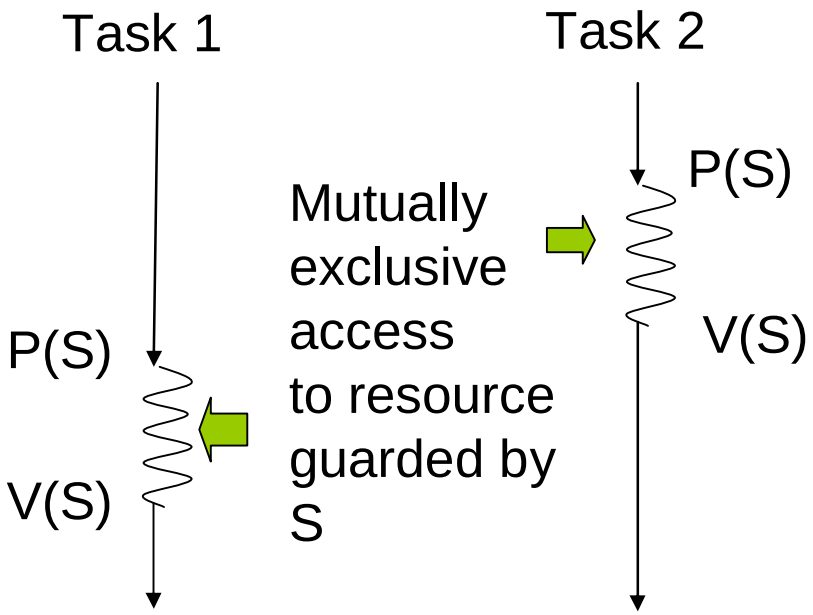
fakultät für
informatik

Resource Access Protocols

Peter Marwedel
Informatik 12
TU Dortmund
Germany

Resource access protocols

Critical sections: sections of code at which exclusive access to some resource must be guaranteed.
Can be guaranteed with semaphores S or “mutexes”.

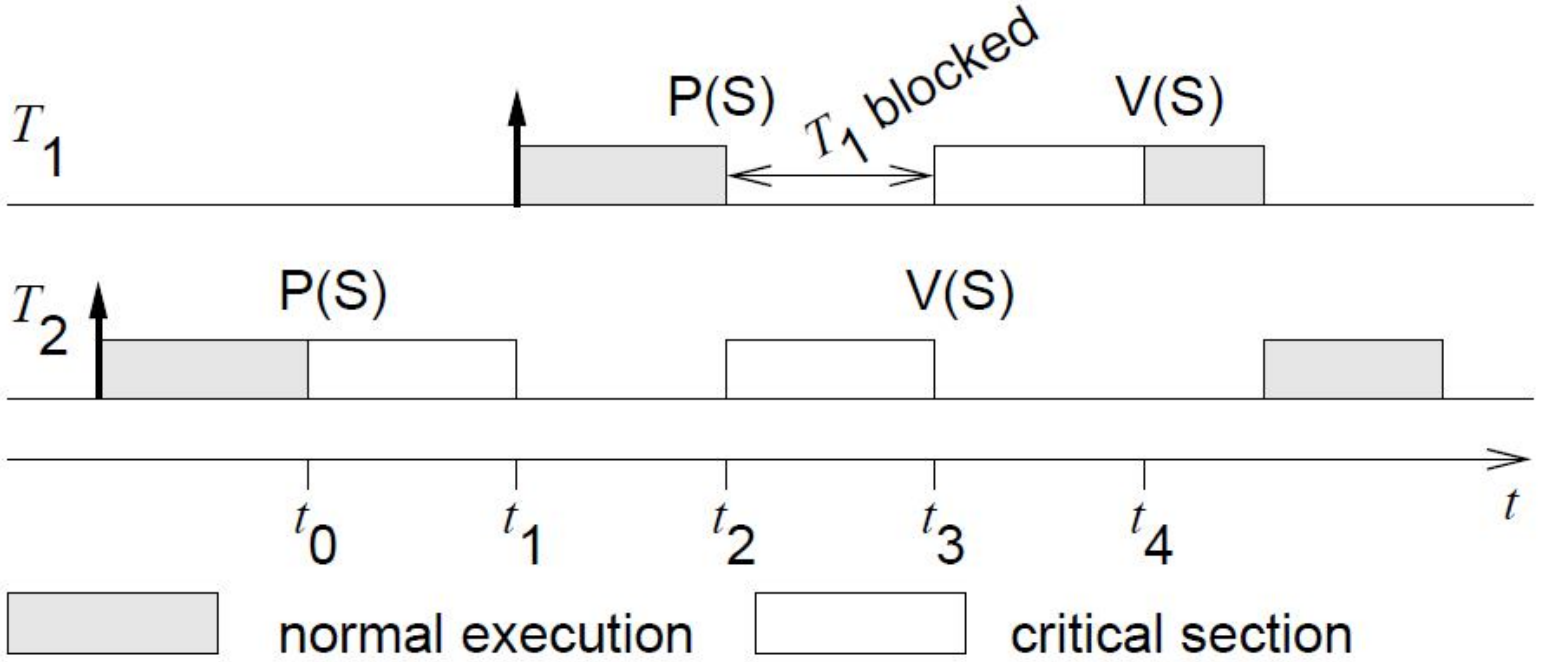


P(S) checks semaphore to see if resource is available
and if yes, sets S to “used“.
Uninterruptible operations!
If no, calling task has to wait.

V(S): sets S to “unused“ and starts sleeping task (if any).

Blocking due to mutual exclusion

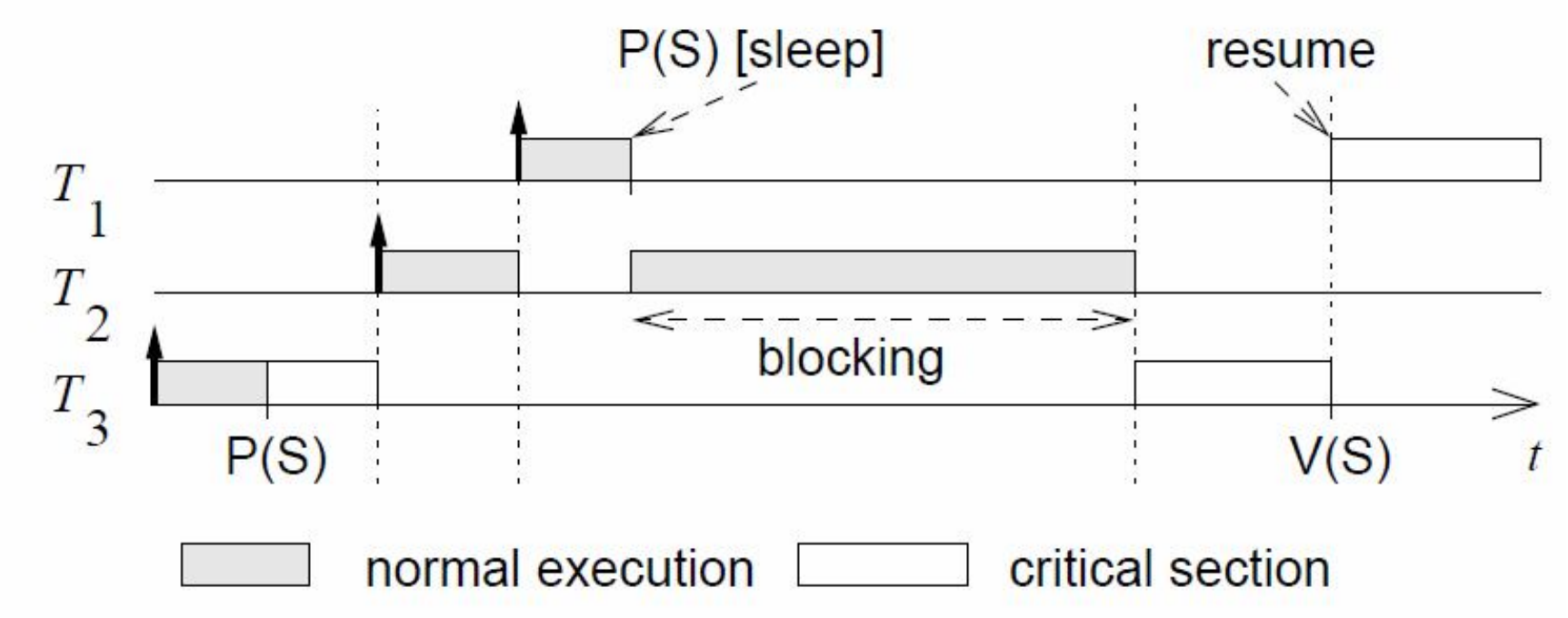
Priority T_1 assumed to be > than priority of T_2.
If T_2 requests exclusive access first (at t_0),
T_1 has to wait until T_2 releases the resource (at time t_3):



For 2 tasks:
blocking is bounded by the length of the critical section

Blocking with >2 tasks can exceed the length of any critical section

Priority of T_1 > priority of T_2 > priority of T_3.
T_2 preempts T_3: T_2 can prevent T_3 from releasing the resource.



Priority inversion!

The MARS Pathfinder problem (1)

“But a few days into the mission, not long after Pathfinder started gathering meteorological data, the spacecraft began experiencing total system resets, each resulting in losses of data. The press reported these failures in terms such as "software glitches" and "the computer was trying to do too many things at once".” …

mars.jpl.nasa.gov

http://research.microsoft.com/~mbj/Mars_Pathfinder/Mars_Pathfinder.html

The MARS Pathfinder problem (2)

“VxWorks provides preemptive priority scheduling of threads. Tasks on the Pathfinder spacecraft were executed as threads with priorities that were assigned in the usual manner reflecting the relative urgency of these tasks.”

“Pathfinder contained an "information bus", which you can think of as a shared memory area used for passing information between different components of the spacecraft.”

- A bus management task ran frequently with high priority to move certain kinds of data in and out of the information bus. Access to the bus was synchronized with mutual exclusion locks (mutexes).”

http://research.microsoft.com/~mbj/Mars_Pathfinder/Mars_Pathfinder.html

The MARS Pathfinder problem (3)

- The meteorological data gathering task ran as an infrequent, low priority thread, … When publishing its data, it would acquire a mutex, do writes to the bus, and release the mutex. ..
- The spacecraft also contained a communications task that ran with medium priority."

☞

High priority: retrieval of data from shared memory
Medium priority: communications task
Low priority: thread collecting meteorological data

http://research.microsoft.com/~mbj/Mars_Pathfinder/Mars_Pathfinder.html

The MARS Pathfinder problem (4)

"… However, very infrequently it was possible for an interrupt to occur that caused the (medium priority) communications task to be scheduled during the short interval while the (high priority) information bus thread was blocked waiting for the (low priority) meteorological data thread.

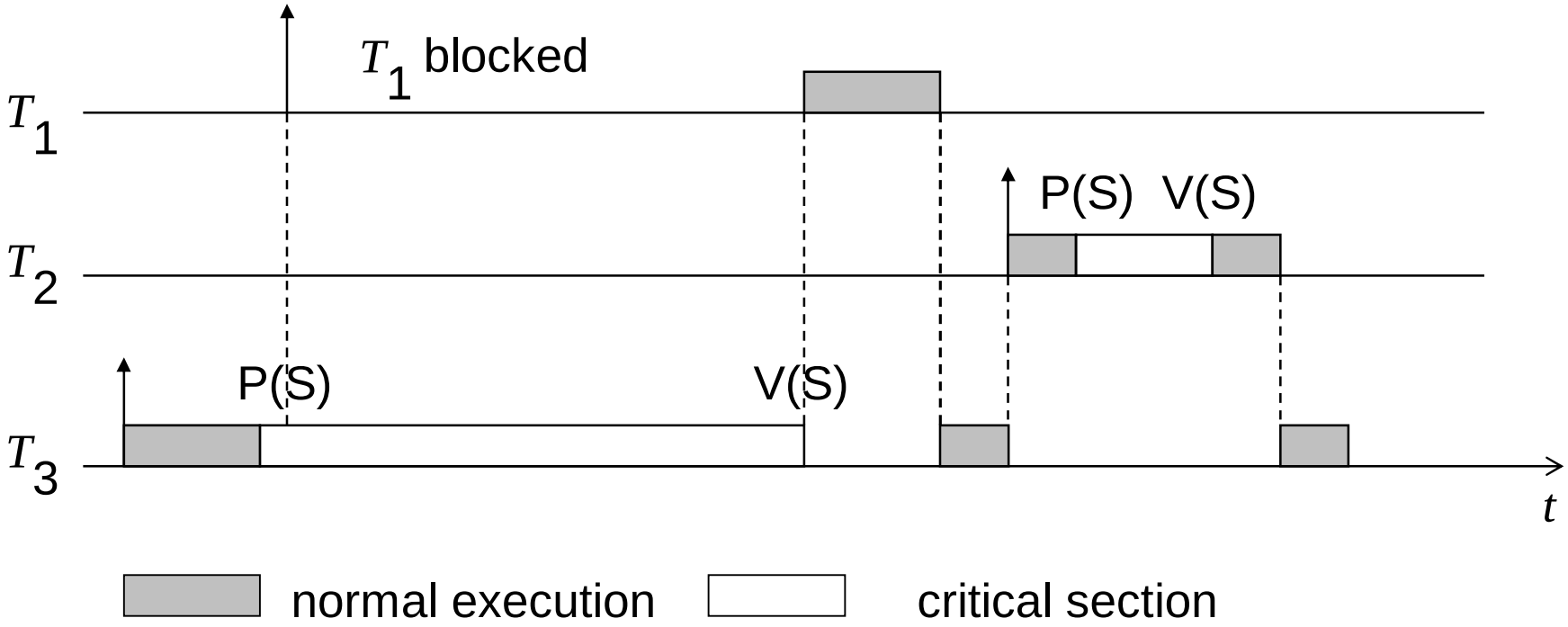
In this case, the long-running communications task, having higher priority than the meteorological task, would prevent it from running, consequently preventing the blocked information bus task from running.

After some time had passed, a watchdog timer would go off, notice that the data bus task had not been executed for some time, conclude that something had gone drastically wrong, and initiate a total system reset."

http://research.microsoft.com/~mbj/Mars_Pathfinder/Mars_Pathfinder.html

Solutions

Disallow preemption during the execution of all critical sections. Simple, but creates unnecessary blocking as unrelated tasks may be blocked.

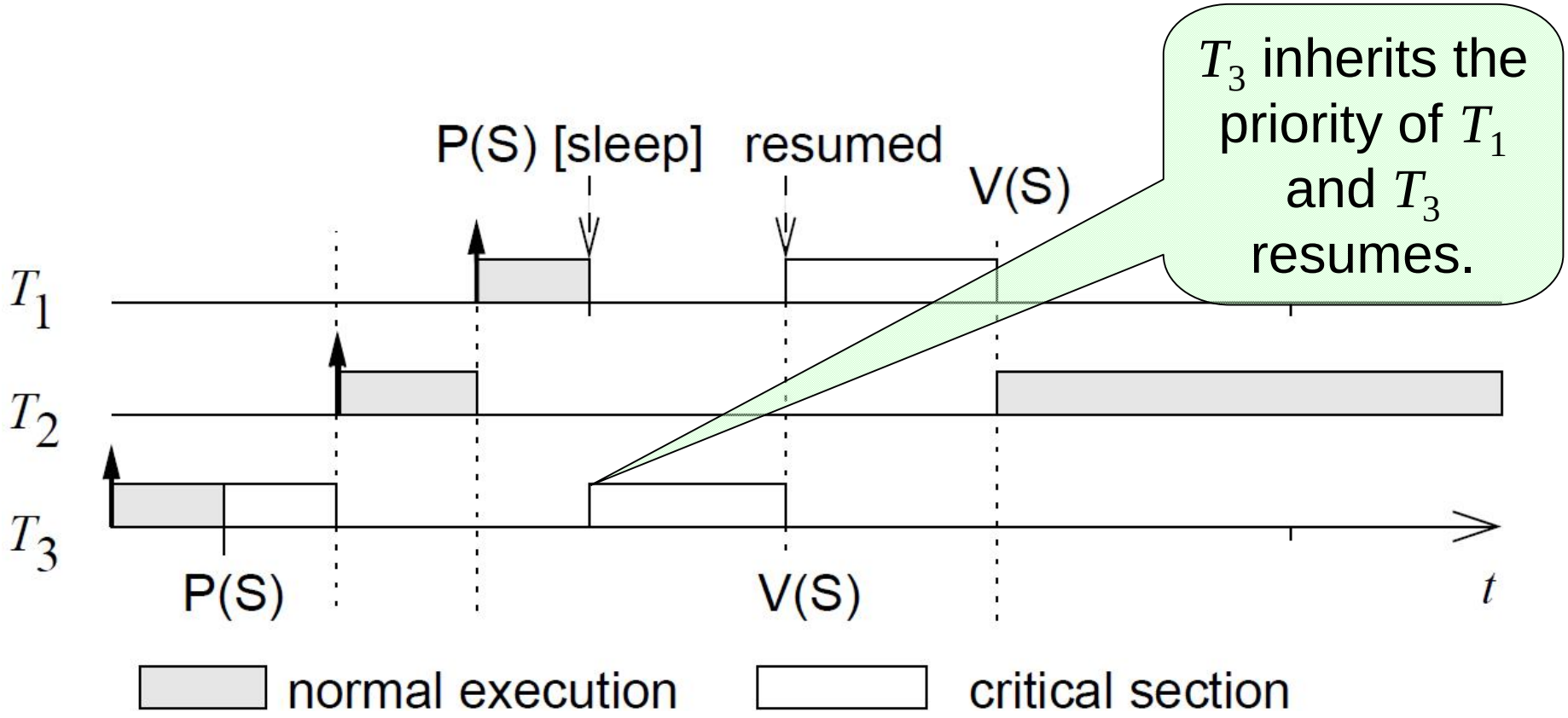


Coping with priority inversion: the priority inheritance protocol

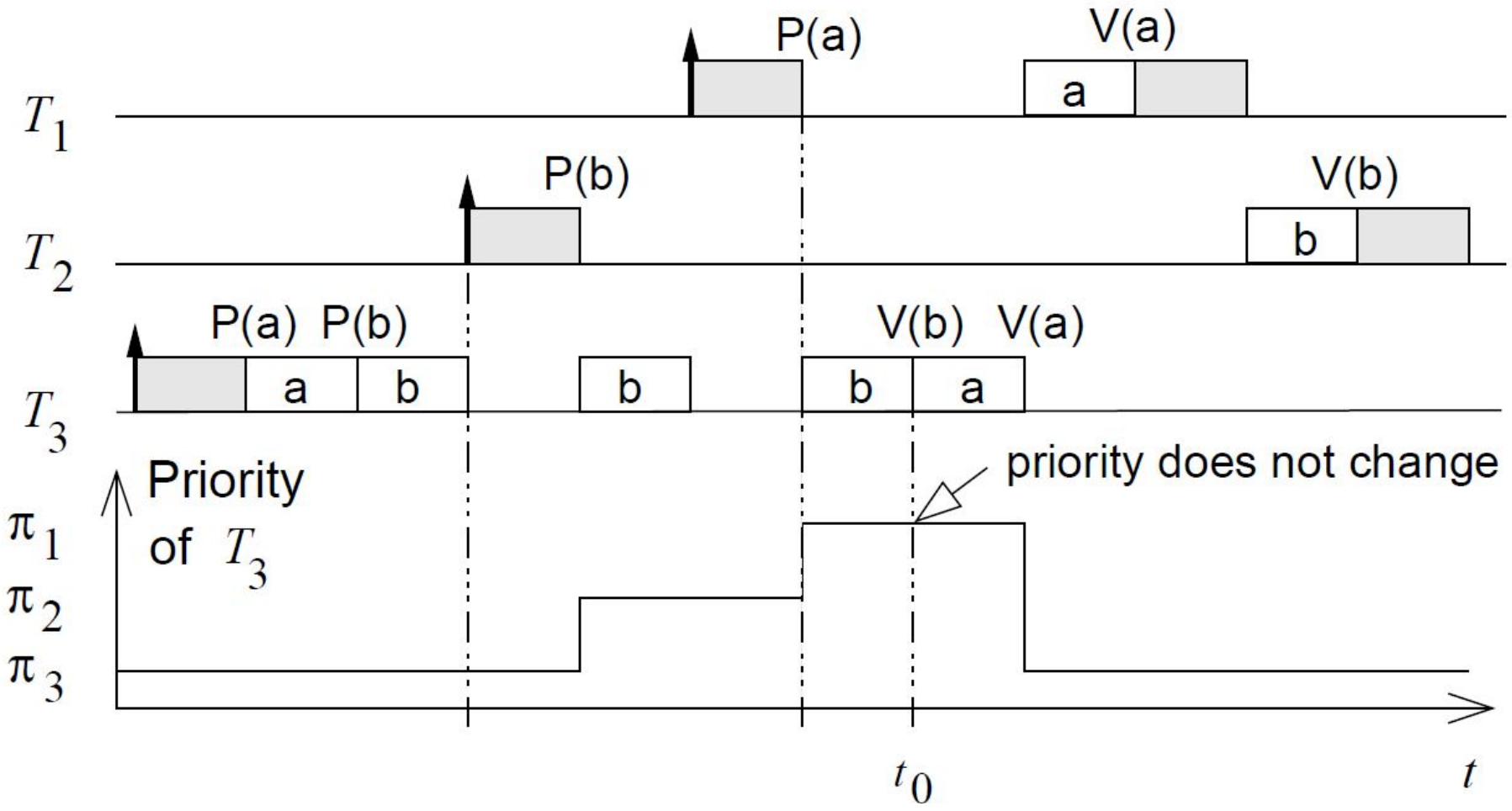
- Tasks are scheduled according to their active priorities. Tasks with the same priorities are scheduled FCFS.
- If task T_1 executes P(S) & exclusive access granted to T_2: T_1 will become blocked.
  If priority(T_2) < priority(T_1): T_2 inherits the priority of T_1.
  ☞ T_2 resumes.
  Rule: tasks inherit the highest priority of tasks blocked by it.
- When T_2 executes V(S), its priority is decreased to the highest priority of the tasks blocked by it.
  If no other task blocked by T_2: priority(T_2):= original value.
  Highest priority task so far blocked on S is resumed.
- Transitive: if T_2 blocks T_1 and T_1 blocks T_0, then T_2 inherits the priority of T_0.

Example

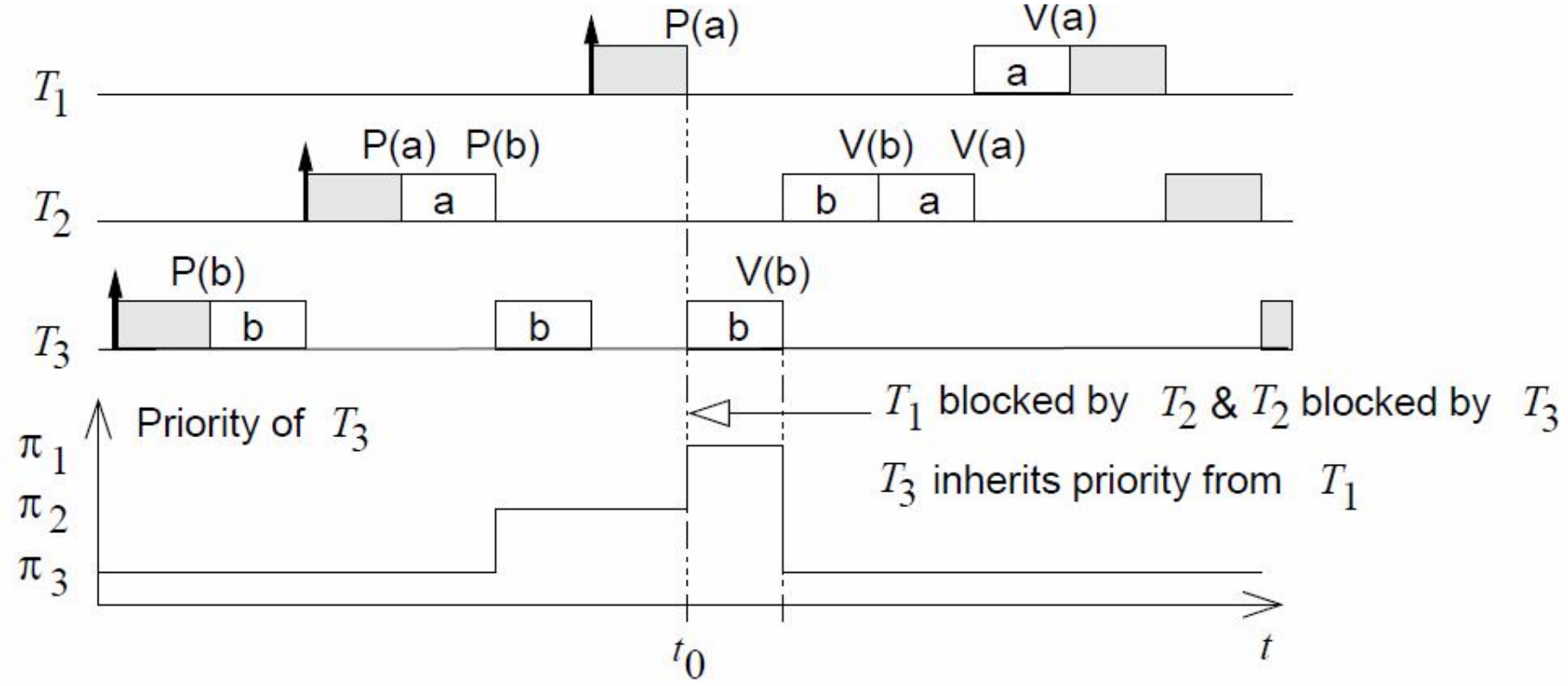
How would priority inheritance affect our example with 3 tasks?



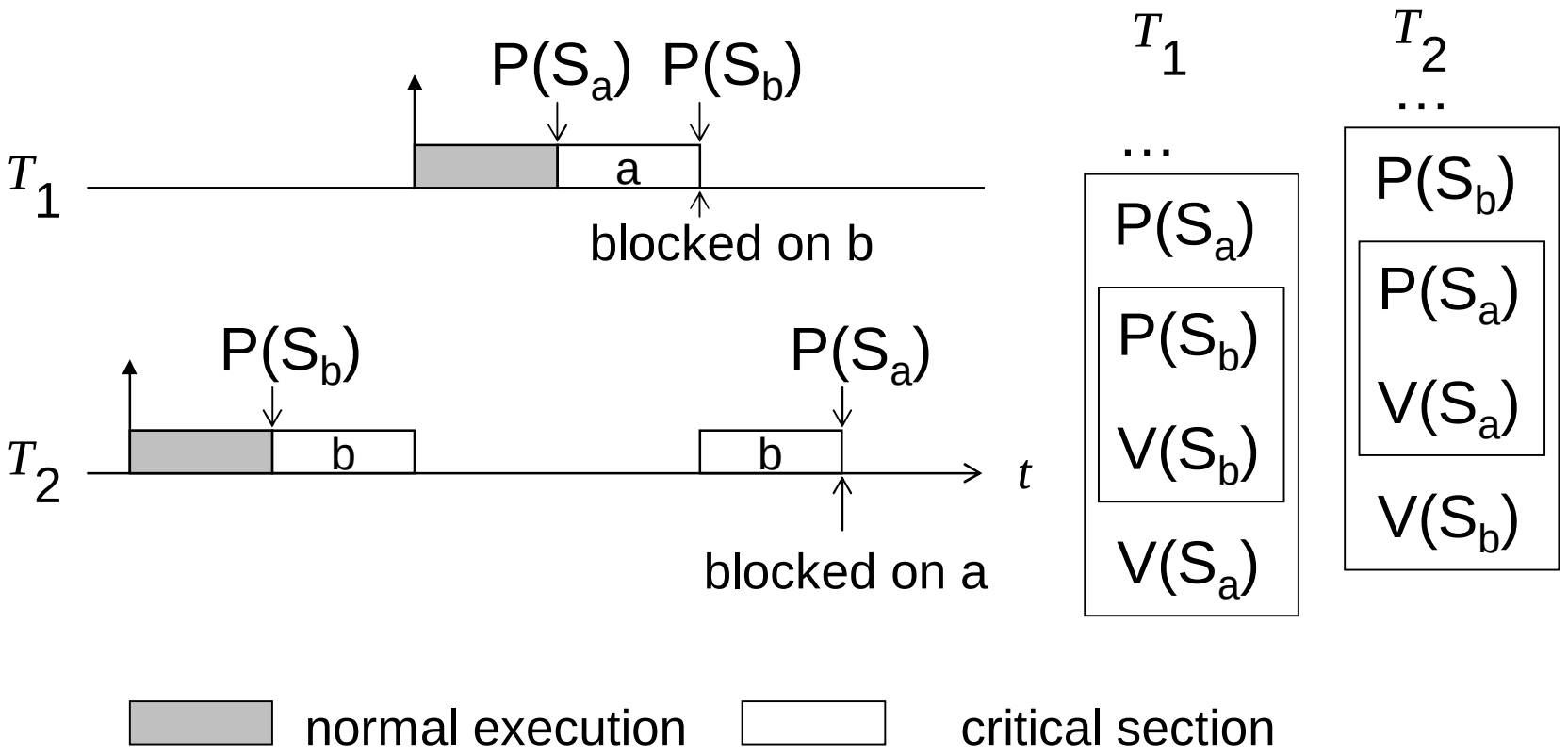
Nested critical sections



Transitiveness of priority inheritance



Deadlock is possible



normal execution critical section

Problem exists also when no priority inheritance is used

Priority inversion on Mars

Priority inheritance also solved the Mars Pathfinder problem: the VxWorks operating system used in the pathfinder implements a flag for the calls to mutex primitives. This flag allows priority inheritance to be set to “on”. When the software was shipped, it was set to “off”.

The problem on Mars was corrected by using the debugging facilities of VxWorks to change the flag to “on”, while the Pathfinder was already on the Mars [Jones, 1997].

mars.jpl.nasa.gov

Remarks on priority inheritance protocol

Possibly large number of tasks with high priority.

Possible deadlocks.

Ongoing debate about problems with the protocol:

> Victor Yodaiken: Against Priority Inheritance, Sept. 2004,
> http://www.fsmlabs.com/resources/white_papers/priority-inheritance/

Finds application in ADA: During *rendez-vous*,
task priority is set to the maximum.

Protocol for fixed set of tasks: priority ceiling protocol.

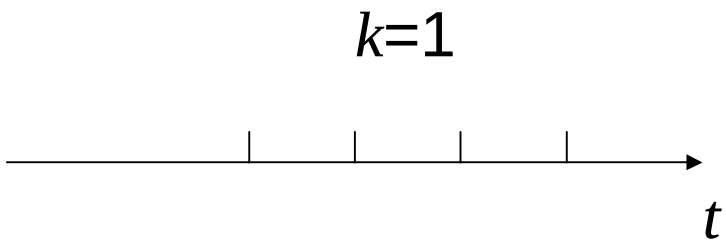
Summary

- General requirements for embedded operating systems
  - Configurability
  - I/O
  - Interrupts
- General properties of real-time operating systems
  - Predictability
  - Time services
  - Synchronization
  - Classes of RTOSs,
  - Device driver embedding
- Priority inversion
  - The problem
  - Priority inheritance

SPARES

Byzantine Error

Erroneous local clocks can have an impact on the computed local time.
Advanced algorithms are fault-tolerant with respect to Byzantine errors. Excluding k erroneous clocks is possible with $3k+1$ clocks (largest and smallest values will be excluded.
Many publications in this area.



Virtual machines

- Emulate several processors on a single real processor
- Running
  - As Single process (Java virtual machine)
  - On bare hardware
    - Allows several operating systems to be executed on top
    - Very good shielding between applications
- Temporal behavior?