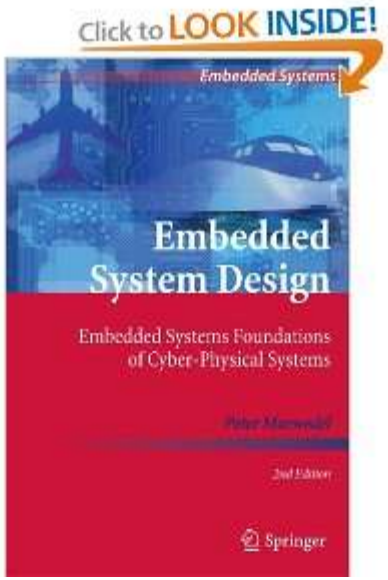
Módulo I - LINUX EMBARCADO
SOs Embarcados

Meios Eletrônicos Interativos II

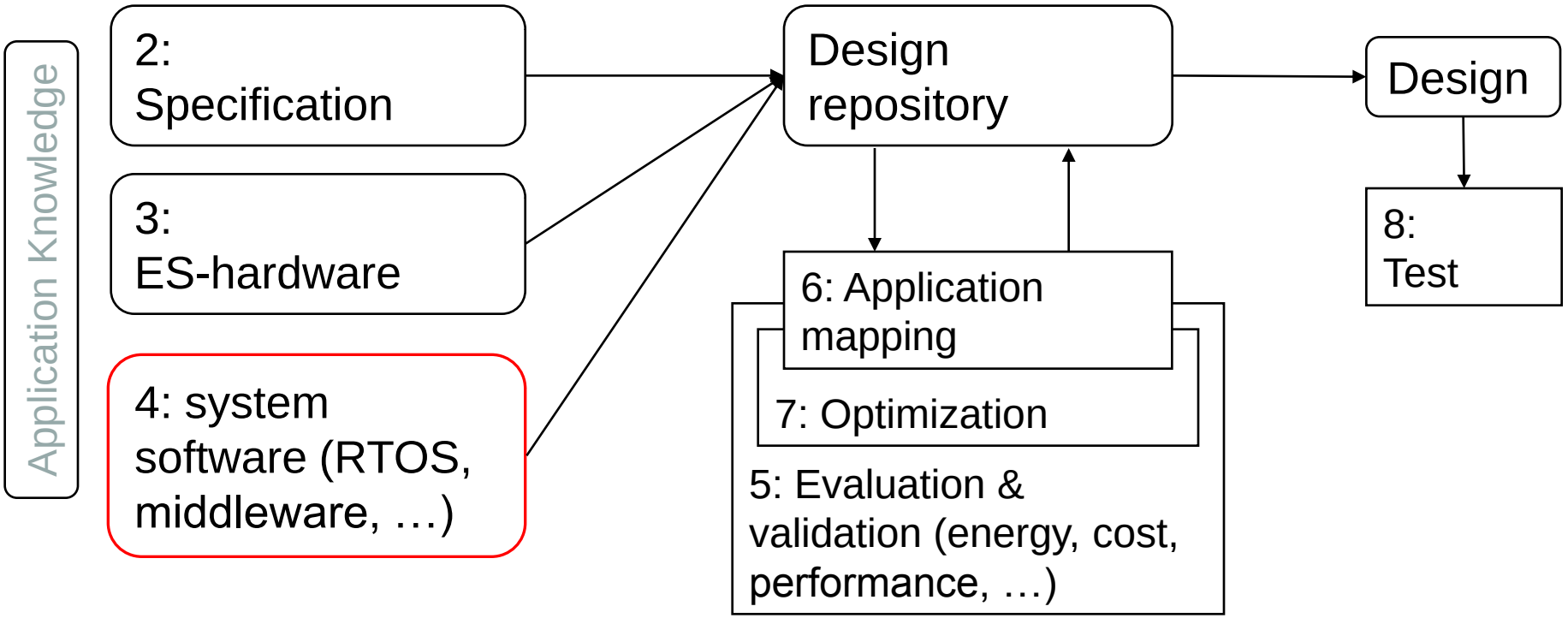
2o. Semestre de 2011

Material Base desta Aula

- Peter Marwedel
- TU Dortmund, Informatik 12
- Germany



Tópico no Livro



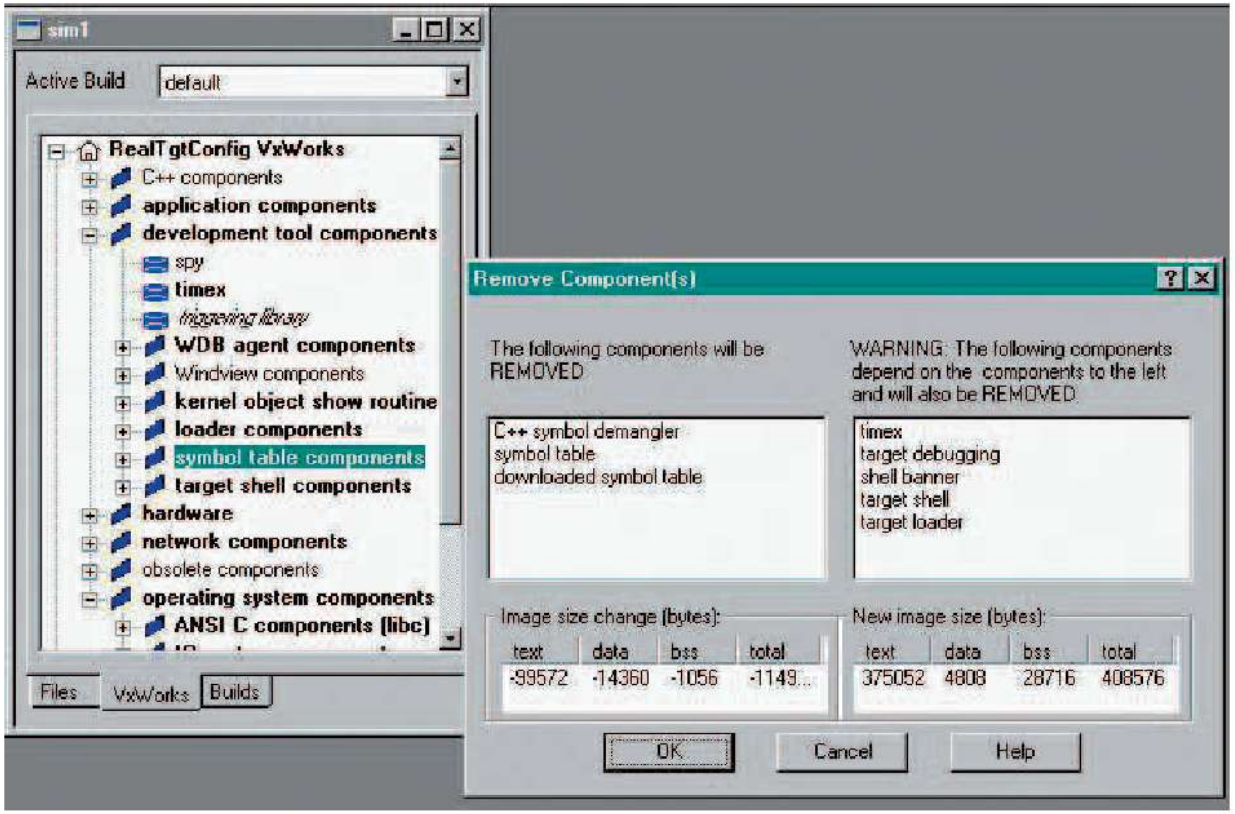
Reuso de componentes de software padrão

- Operating systems
- Middleware
- ….

Sistemas Operacionais Embarcados: Configurabilidade

- Configurabilidade: nenhum SO vai atender todas as necessidades, não deve aver sobrecarga (overhead) devido às funções não utilizadas
  - configurabilidade.
  - Foma mais simples: remover as funções não usadas (pelo linker ?).
  - Compilação condicional (usando comandos #if e #ifdef).
  - Dados dinâmicos devem ser substituidos por dados estáticos.
  - Avaliação avançada em tempo-de-compilação é útil.
  - Orientação a objeto pode levar a derivação de subclasses.

Exemple: Configuração do 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

Verificação do SO derivado?

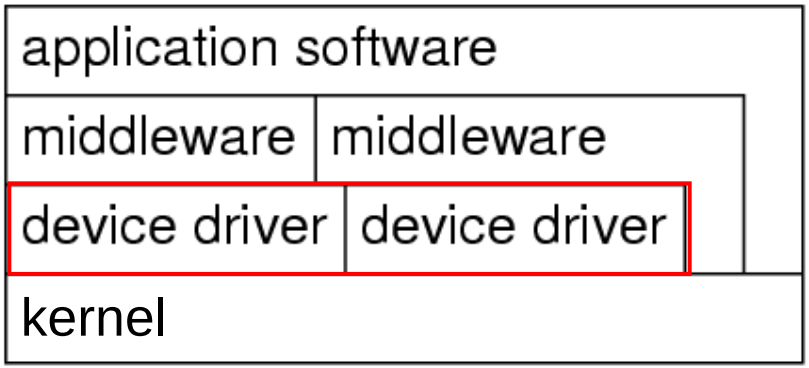
- Verificação é um problema potencial de sistemas com um grande número de Sos derivados:
  - Cada SO deve ser testado cuidadosamente;
  - Exemplo do eCos (RTOS open source RTOS da Red Hat), que inclui 100 a 200 pontos de configuração [Takada, 01].

Sistemas Operacionais Embarcados

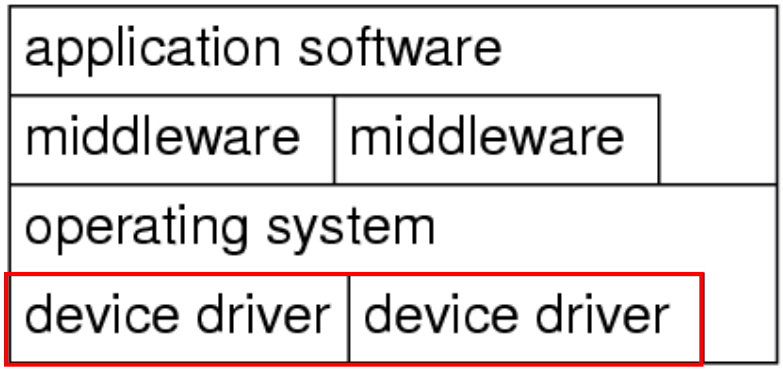
- Disco e rede manipulados por tarefas -

- Efetivamente não há nenhum dispositivo que que necessite ser suportado por todas as variações do SO, com exceção talvez do *system timer*.
- Muitos SE são sem disco, teclado, tela or mouse.
- Disco & rede manipulados por tarefas em vez de drivers integrados.

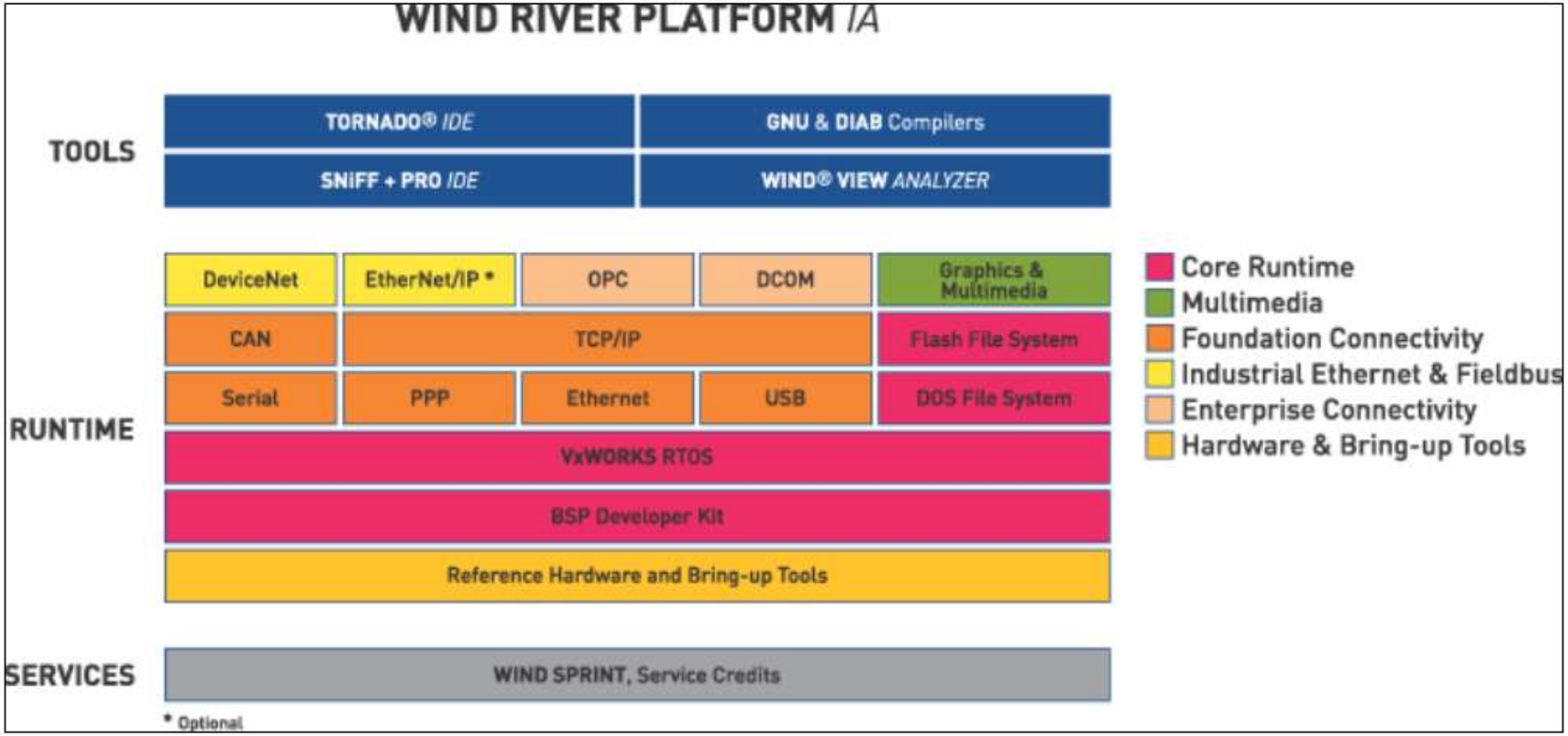
Embedded OS



Standard OS



Exemple: WindRiver Platform Industrial Automation



Sistemas Operacionais Embarcados - Proteção é opcional-

- Mecanismos de proteção não são sempre necessários:

  SE tipicamente são projetados para um propósito simples, programas não testados são raramente carregados, SW considerado confiável**.**

Instruções *de* I/O **privilegiadas** não são necessárias e tarefas podem fazer diretamente a E/S

Exemplo: Let **`switch`** be the address of some switch
Simply use

```
load register,switch
```

instead of OS call.

Porem, mecanismos de proteção podem ser necessários por motivos de seguraça

Sistemas Operacionais Embarcados - Interrupções não restritas ao SO-

- **Interrupções podem ser empregadas por qq processo** (Em SO padrões: fonte de não confiabilidade).

  - Programas embarcados podem ser considerados testados,
  - Uma vez que proteção não é necessária e
    - Uma vez que requer-se controle eficiente sobre diversos dispositivos,
    - É possível permitir que interrupções iniciem ou parem tarefas diretamente
      (armazenando-se o endereço de início da tarefa em uma tabela de interrupção).
  - Mais eficiente do que usar serviços do SO.
  - **Reduz a "composability": se uma tarefa é conectada a uma interrupção, pode ser difícil adicionar outra tarefa que também necessite ser disparada pelo evento.**

SO embarcados: Tempo-Real

- Muitos sistemas emabarcados são tempo-real (RT)
  - O SO deve ser SO tempo-real **(RTOSs).**

SO Tempo-Real: Definição e requisito 1- predictibilidade

- **Def**.: *(A) UM SO Tempo-Real é um SO que suporta a construção de sistemas tempo-real.*
- Tres (3) requisitos chave:

1. **O comportamento temporal do SO deve ser preditível.**
   ∀ serviços do SO: Limite superior no tempo de execução!
   RTOSs devem ser ***timing-predictable***:
   - Períodos em que as interrupções são desabilitadas são CURTOS,
   - (para hard disks:) arquivos contiguos para evitar movimentos não preditíveis da cabeça.

[Takada, 2001]

RTOS: requisito 2- gerenciando temporização

2. **SO deve gerenciar a temporização e escalonamento**
   - O SO possivelmente tem de conhecer os *deadlines* das tarefas;
     (a menos que o escalonamento seja feito *off-line*).
   - Frequentemente, o SO deve prover serviços de tempo precisos com alta resolução.

[Takada, 2001]

Serviços de Tempo (Time services)

- Tempo desempenha um papel central em sistemas tempo-real.
- O tempo verdadeiro é descrito por numeros reais.
- Padrões usados em equipamentos de tempo-real:
  - **International atomic time TAI**
    (frances: *temps atomic internationale*)
    Livre de qq aetefato artificial
  - **Universal Time Coordinated (UTC)**
    UTC é definido por padrões astronomicos
- UTC e TAI foram sincronizados em 01/01/1958.
- 30 segundos foram adicionados desde então.
- Problema: O Ano Novo pode começar duas vezes em uma noite.

Tipos de sincronização

- Interna
- Externa

Sincronização Interna

- Sincronização com um relógio mestre
  - Tipicamente usado em ***startup-phases***
- Sincronização Distribuída:

1. Coletar informação dos vizinhos
2. Computar o valor de correção
3. Ajustar o valor de correção.

Precisão do passo 1 depende de como a informação é coletada:

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

Sincronização externa

- Sincronização externa garante consistência com o tempo físivo real.
- GPS para sincronização externa.
- GPS: provê informação de tempo TAI e UTC.
- Resolução é de 100 ns.

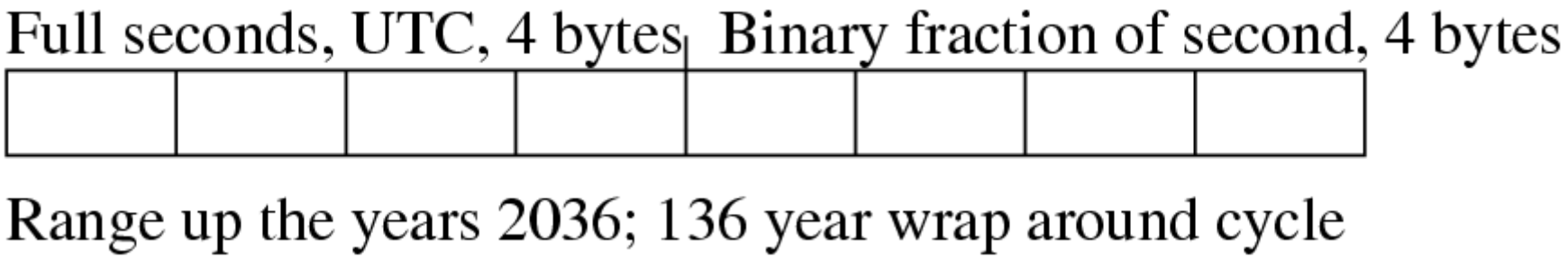
GPS mouse

Problemas com sincronização externa

- Problemática a partir da perspectiva de *fault-tolerance*:

  Valores erroneos são copiados para todas as estações.
  Consequencia: Aceitar apenas pequenas mudanças no tempo local.

- Muitos formatos de tempo são restritos;
  e.g.: NTP suporta até o ano de 2036



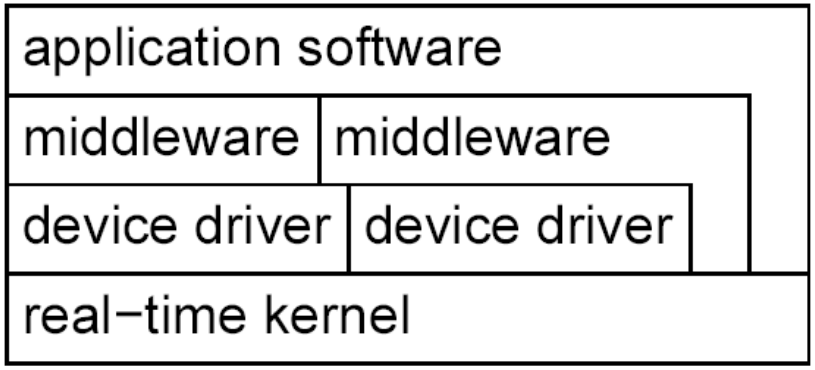
RTOS: Requisito 3 - Velocidade

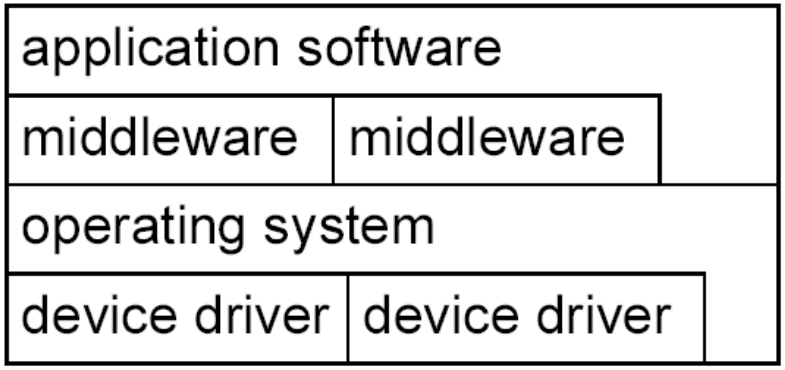
3. **O SO deve ser rápido**

[Takada, 2001]

RTOS-Kernels

- **Distinção entre**
  - real-time kernels e kernels modicados de SO's padrões.





Distinção entre

- RTOSs para aplicações gerais e RTOSs para aplicações específicas,
- APIs padrões (e.g. POSIX RT-Extension of Unix, ITRON, OSEK) ou APIs proprietários.

Funcionalidade de RTOS-Kernels

- **Inclui gerenciamento**
  - processador,
  - memoria,
  - timer;

  } resource management

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

Classes de RTOSes (R. Gupta): 1. Fast proprietary kernels

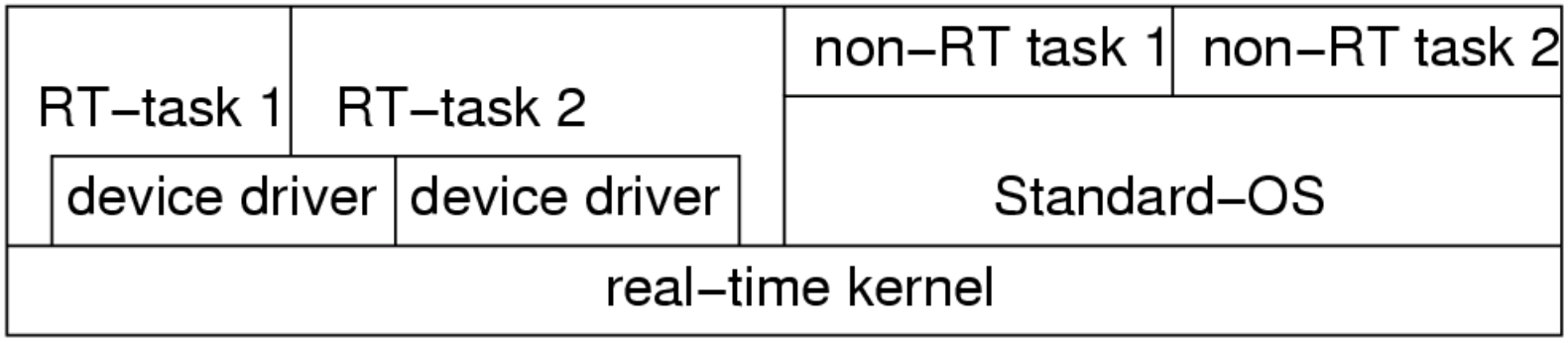
- *Para sistemas complexos, estes kernels são inadequados, pois eles são projetados para serem rápidos (velozes), em vez de serem predictiveis em cada aspecto*
  - [R. Gupta, UCI/UCSD]

Exemplos

QNX, PDOS, VCOS, VTRX32, VxWORKS.

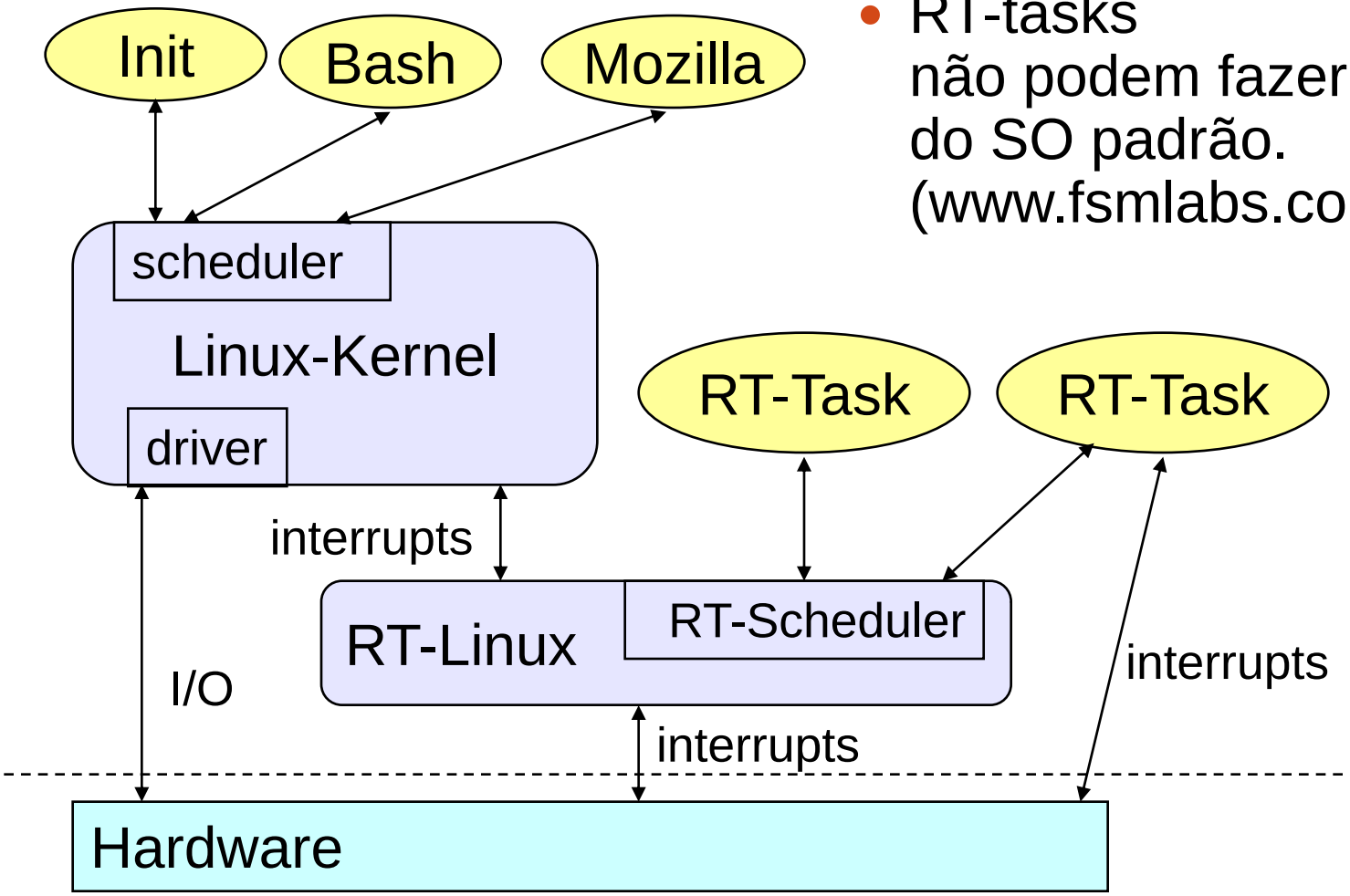
Classes de RTOSs (R. Gupta): 2. RT extensions to std. OSs

RT-kernel roda todas as RT-tasks.
Standard-OS executado como uma tarefa.



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

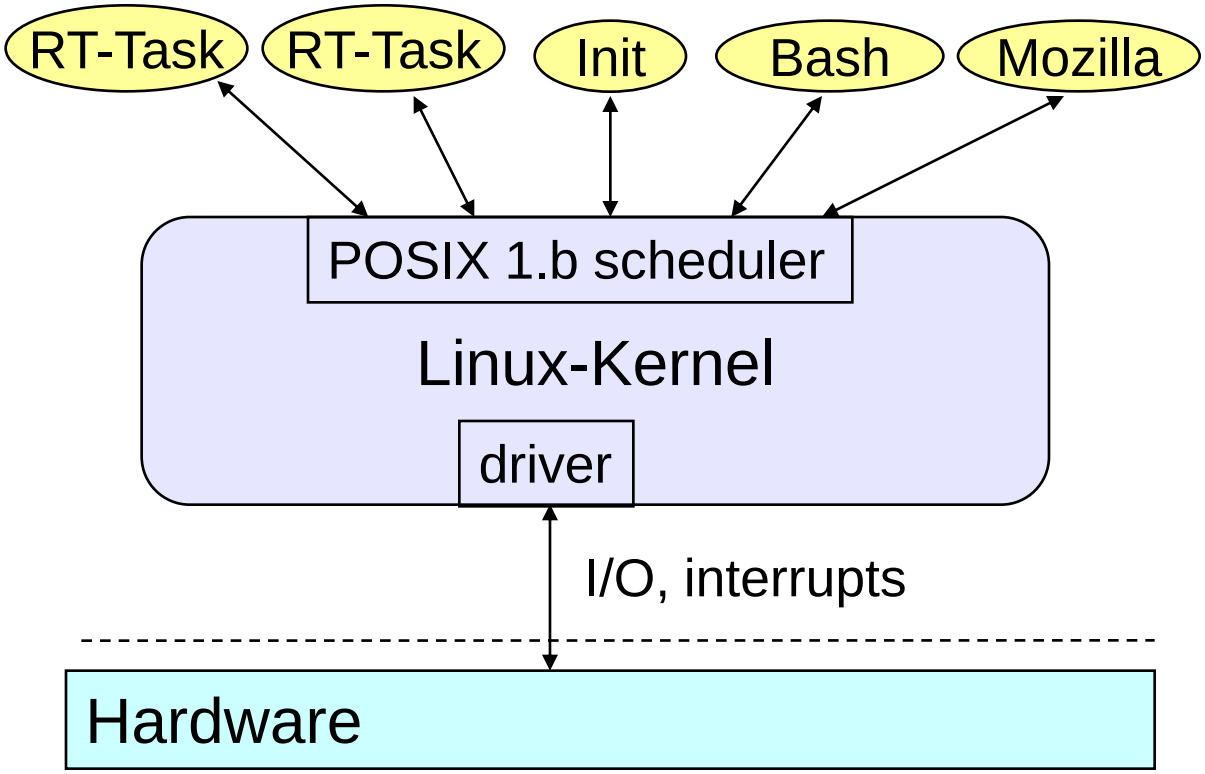
Exemplo: RT-Linux



- RT-tasks
  não podem fazer chamadas do SO padrão.
  (www.fsmlabs.com)

Exemplo:
Posix 1.b RT-extensions to Linux

- Standard scheduler can be replaced by POSIX scheduler implementing priorities for RT tasks



Special RT-calls and standard OS calls available.
Easy programming, no guarantee for meeting deadline

Avaliação (Gupta)

- De acordo com Gupta, tentar usar uma versão de um SO standard:
  - *Não é a abordagem mais correta:*
    - *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*.
  - Dependencias entre tarefas não frequente para a maioria das aplicações de SO's padrões e portanto geralmente ignorados.
  - SE: dependencias entre tarefas são comuns.

Classes de RTOSs (R. Gupta): 3. Tópicos de Pesquisa

- **Research issues** [Takada, 2001]:
  - Proteção de memória com baixo *overhead*
  - Proteção temporal de recursos de computação
  - RTOS's para "*on-chip multiprocessors*"
  - suporte para mídia contínua
  - Controle de qualidade de serviço (QoS).

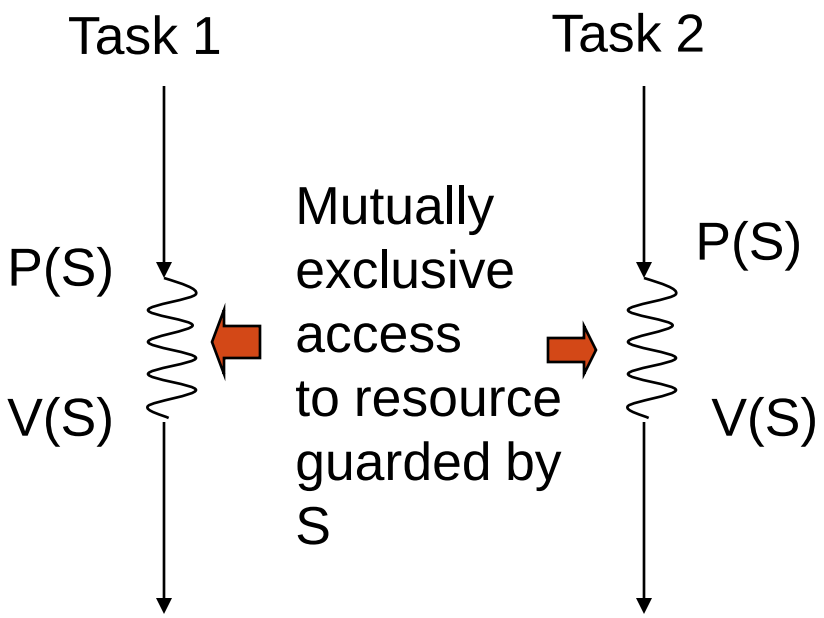
Máquinas Virtuais

- Emula diversos processadores em um único processador real, rodando
  - Como processos simples (Java virtual machine)
  - Sobre o HW
    - Permite que diversos SO’s possam ser executados sob o HW
    - Boa blindagem entre as aplicações
- Comportamento Temporal ???

Protocolos de Acesso a Recursos

Protocolos de Acesso a Recursos

- **Seções Críticas**
- Pode ser garantido com semáforos S ou “mutexes”.

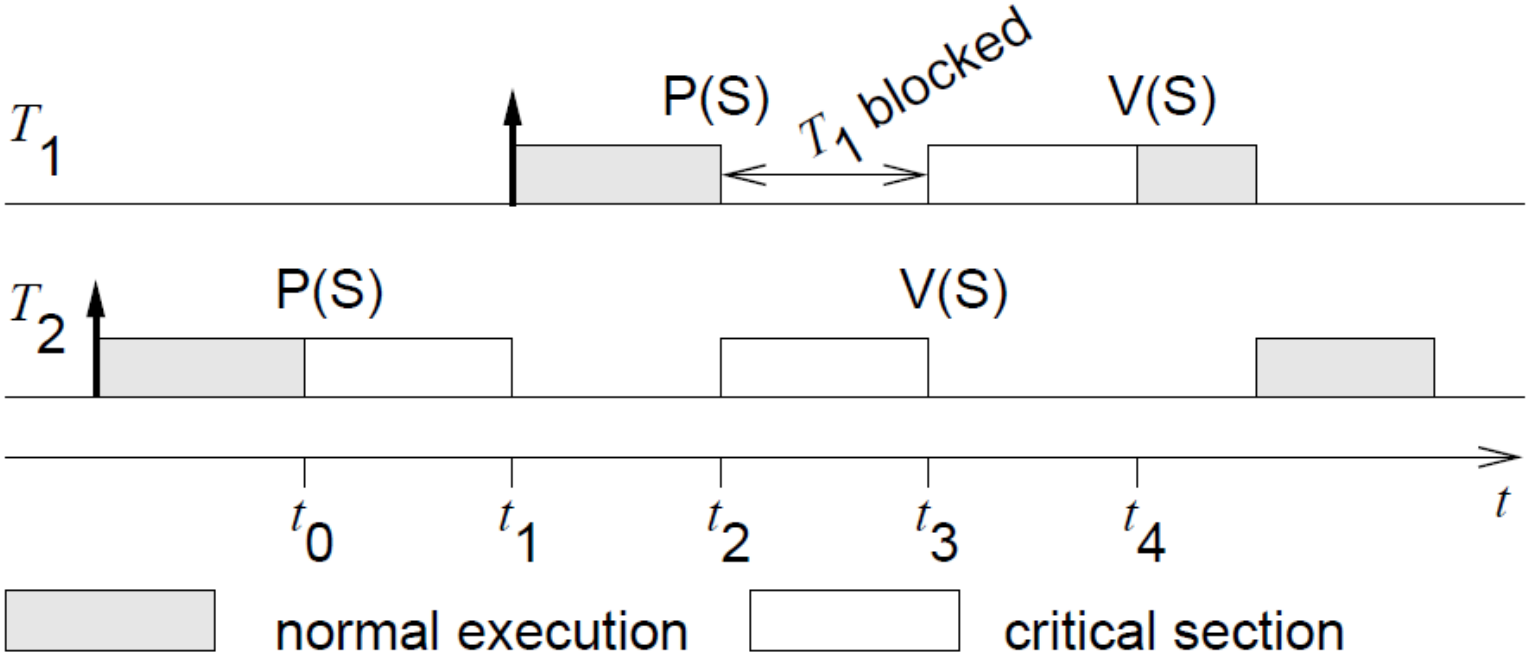


P(S) checks semaphore to see if resource is available and if yes, sets S to “used“. Uninterruptible operations! If no, calling task has to wait.

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

Bloqueio devido a exclusão mútua

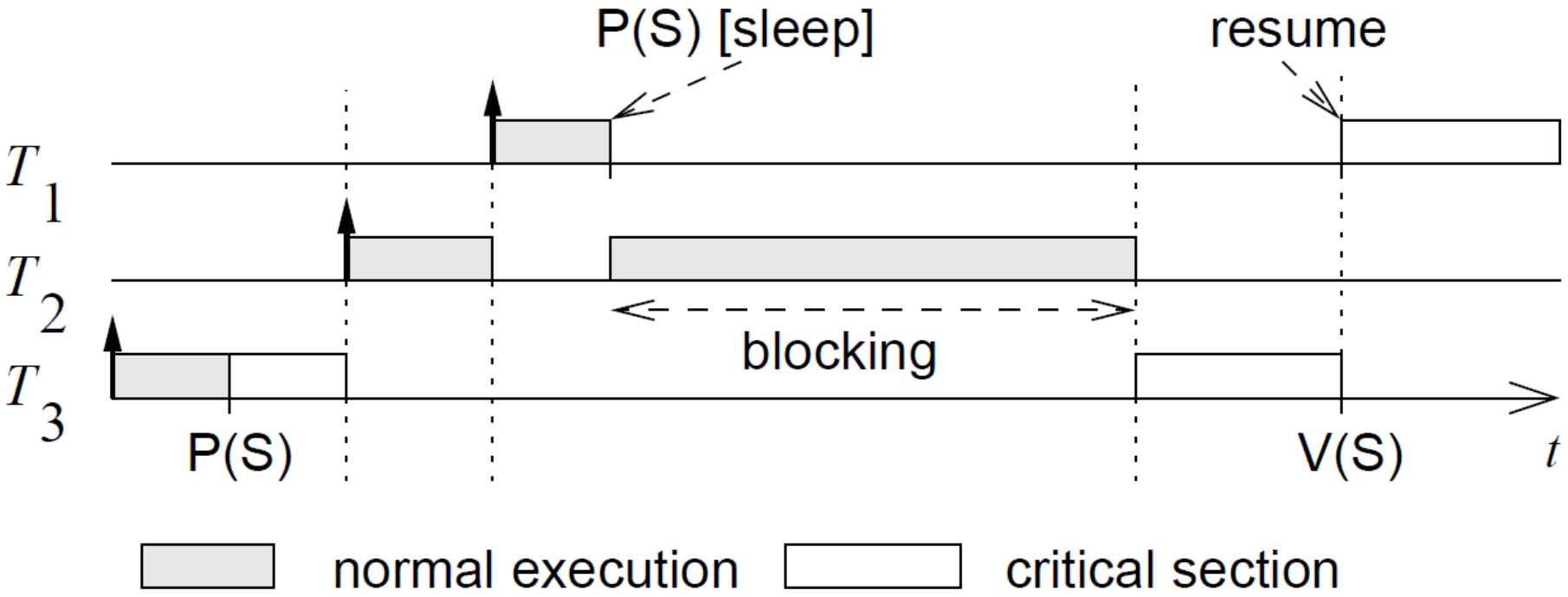
- Prioridade T_1 assumido ser>que prioridade de T_2.
- Se T_2 solicita acesso exclusivo primeiro (emt_0), T_1 deve esperar até T_2 libere o recurso (tempo t_3), então invertendo a prioridade:



Neste exemplo:
bloqueio é "*bounded*" pelo comprimento da seção crítica de T_2.

Bloqueio com >2 tasks pode exceder o comprimento de qq seção crítica

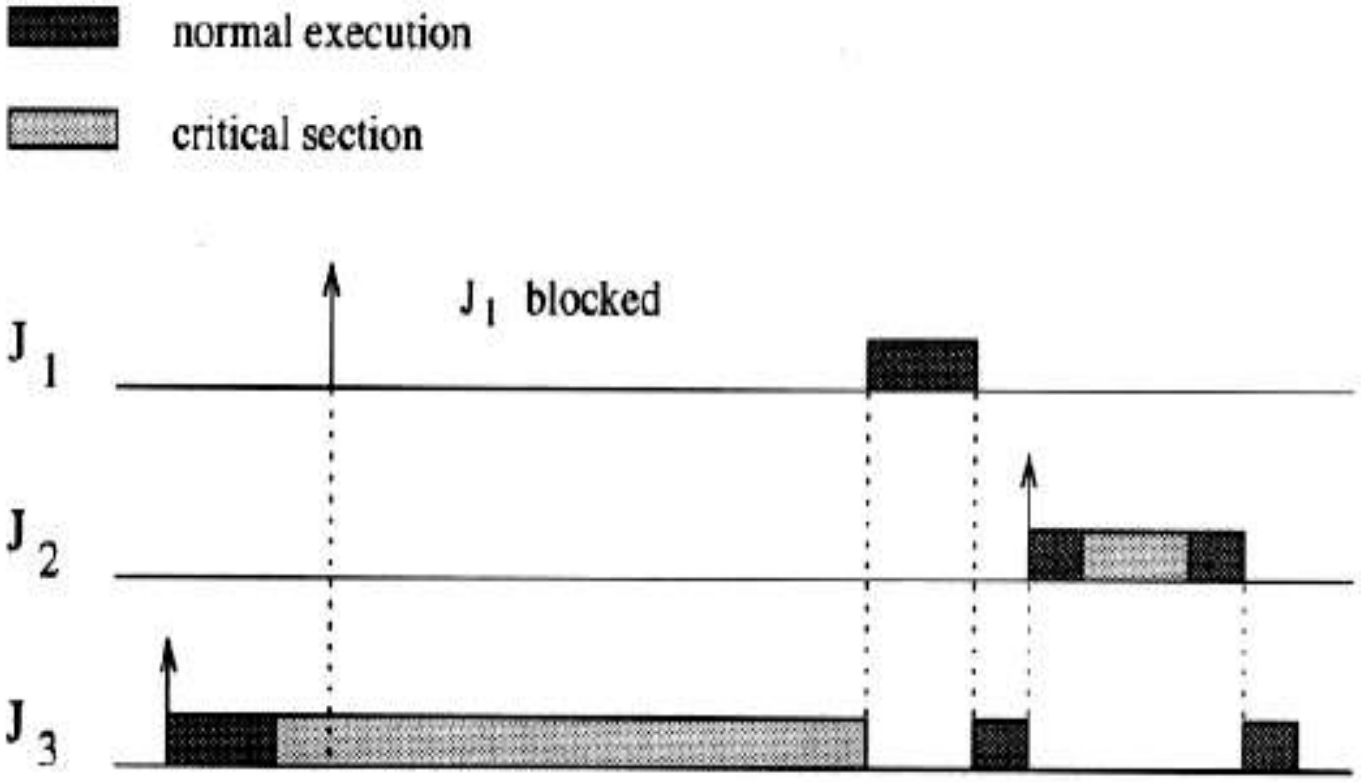
- Prioridade de T_1 > prioridade de T_2 > prioridade de T_3.
- T_2 "preempta" T_3:
- T_2 pode impedir T_3 de liberar o recurso.



Inversão de Prioridade!

Solutions

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



O problema do MARS Pathfinder (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".” …

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

MARS Pathfinder (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)

- “Most of the time this combination worked fine. 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. This scenario is a classic case of priority inversion.”

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

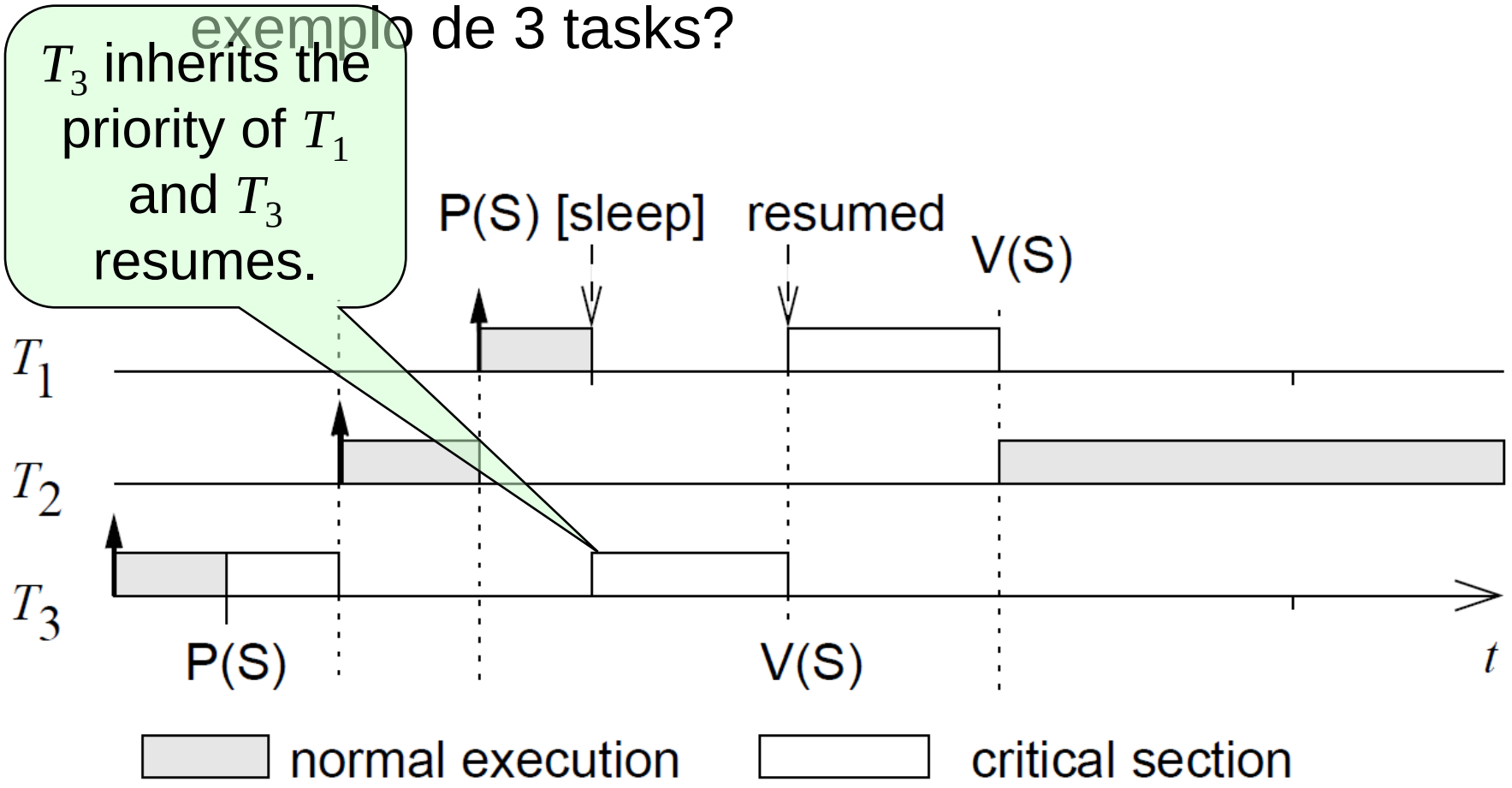
Inversão de Prioridade

Lidando com inversão de prioridade: o protocolo de herança de prioridade

- Tasks são escalonadas de acordo com suas prioridades ativas. Tasks com a mesma prioridade são escalonadas FCFS.
- Se task T_1 executa P(S) & acesso exclusivo é dado a T_3:
  T_1 deve se tornar bloqueado.
  Se a prioridade(T_3) < prioridade(T_1): T_3 herda a prioridade de T_1.
  ☞ T_3 continua execução.
  Regra: tasks herdam a maior prioridade das tasks bloqueadas por ele.
- Qdo T_3 executa V(S), sua prioridade é reduzida à maior prioridade das tarefas nloqueadas por ele.
  Se nenhuma outra task é bloqueada por T_3: prioridade(T_3):= valor original.
  A task bloqueada em S de maior prioridade é continuada.
- Transitivo: se T_3 bloqueia T_2 e T_2 bloqueia T_1, então T_3 herda a prioridade de T_1.

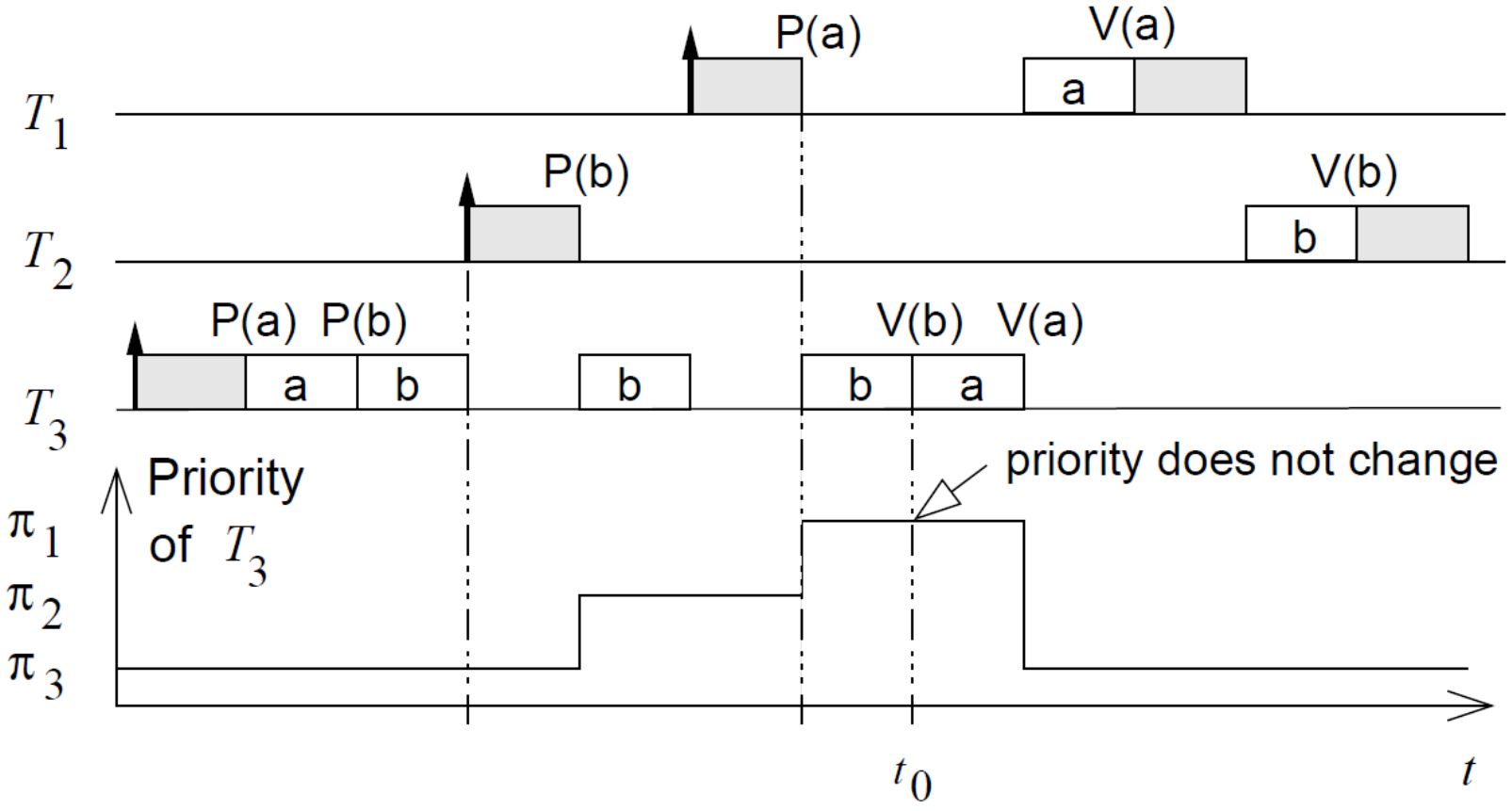
Exemplo

- Como a herança de prioridade afeta nosso exemplo de 3 tasks?

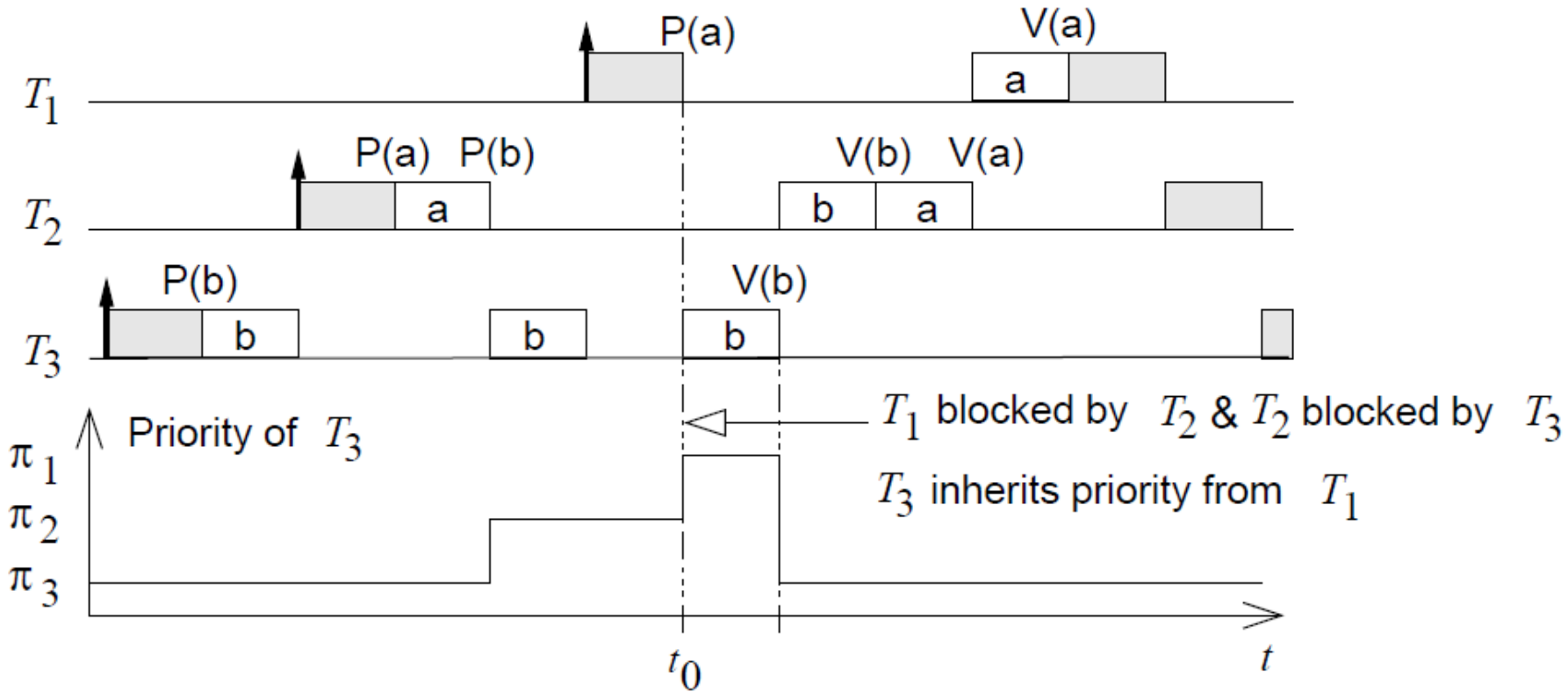


leviRTS animation

Seções Críticas Aninhadas (Nested)



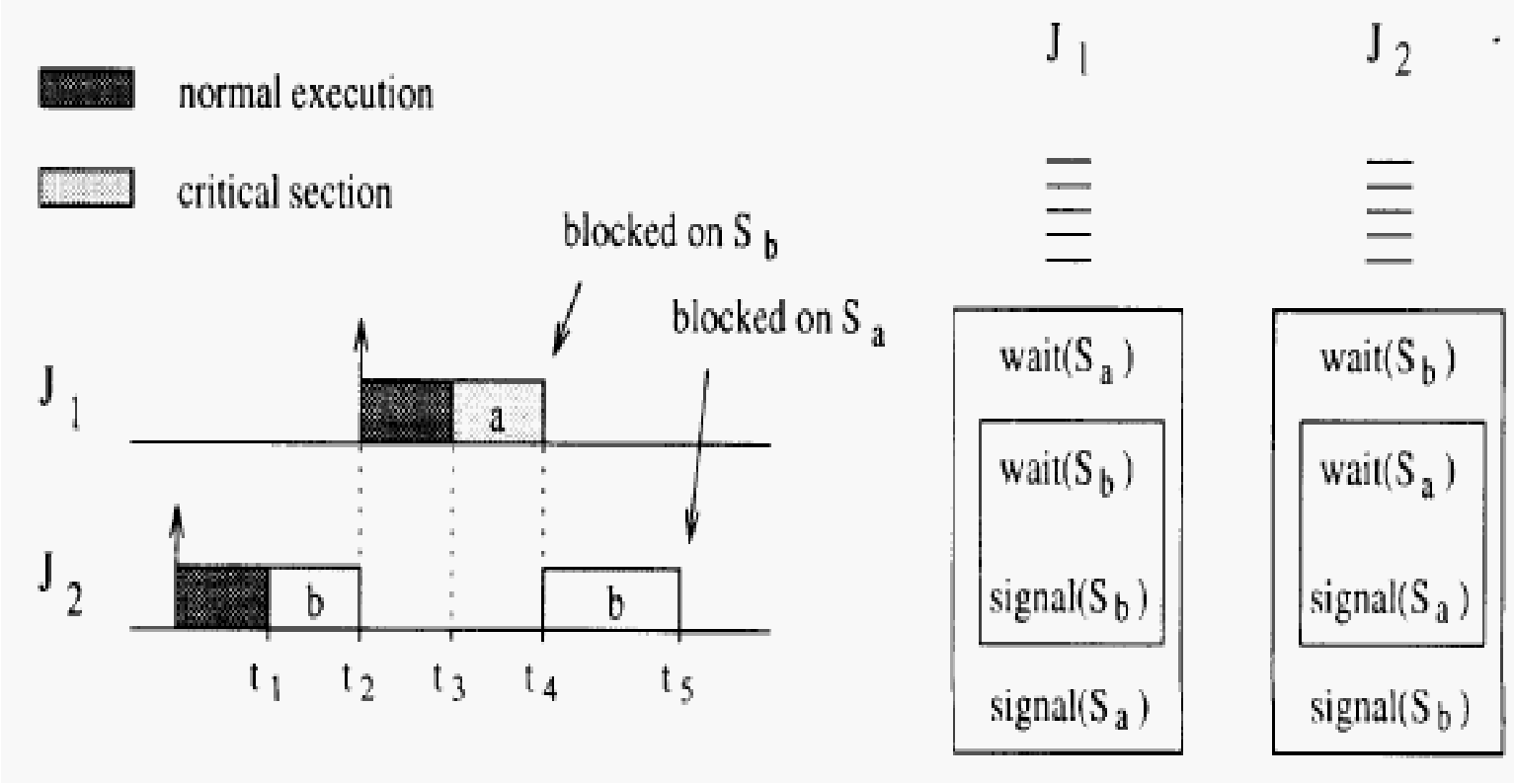
Transitividade da herança de prioridade



[P/V added@unido]

Priority Inheritance Protocol (PIP)

- Problem: ***Deadlock***



[But97, S. 200]

Inversão de Prioridade no 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].

Considerações sobre o protocolo de herança de prioridade

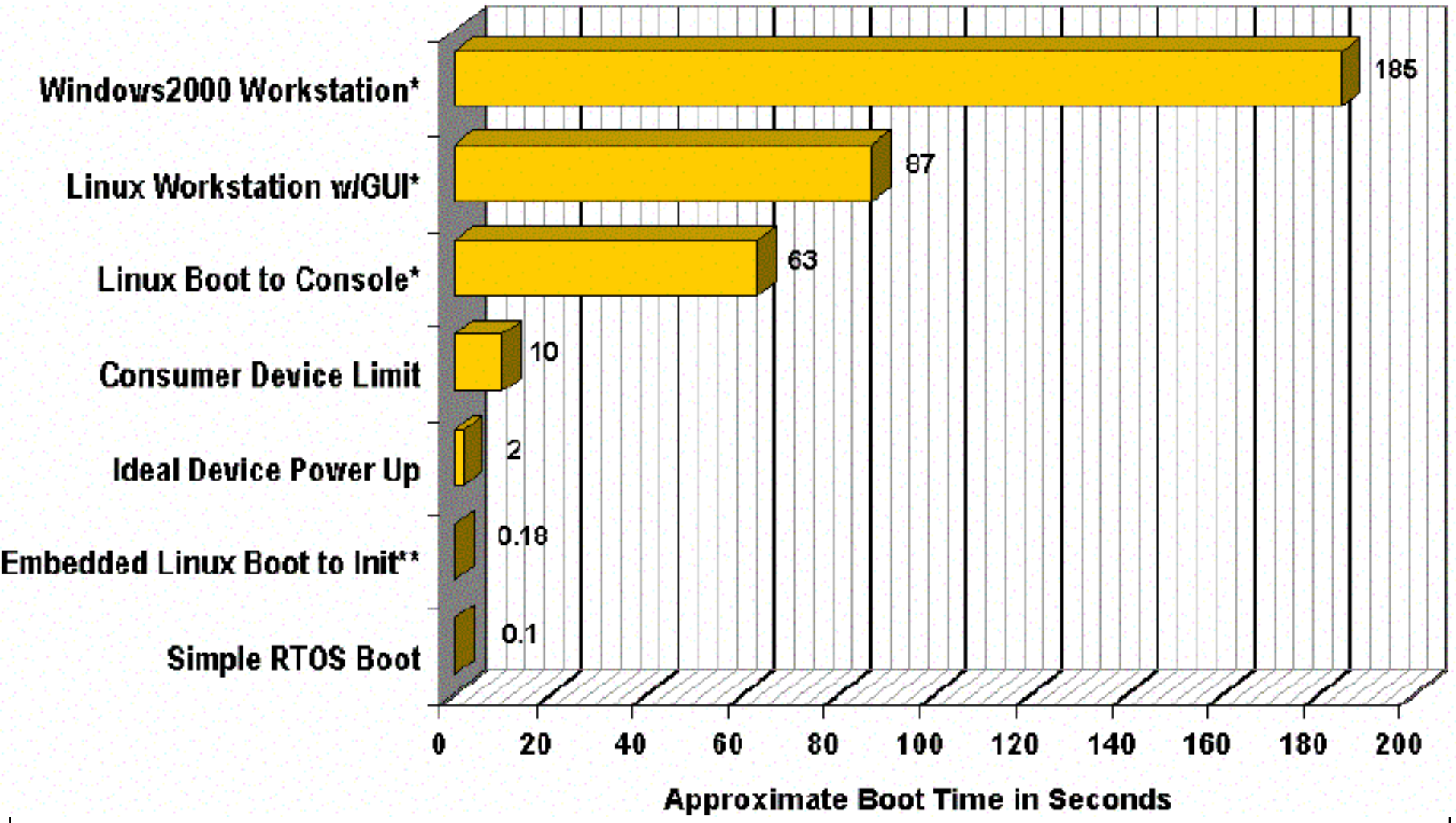
- Número maior de tasks com prioridade alta.
- Mas:
  - *É possível a acorrência de deadlocks (qdo?)*.
  - É possível ocorrer Cadeias de bloqueio
- Protocolo para um conjunto fixo de tasks: *priority ceiling protocol*.

Transparências adicionais

Tempo de boot

- E’um aspecto importante em SE?

SW: Tempo de Boot



FIM!