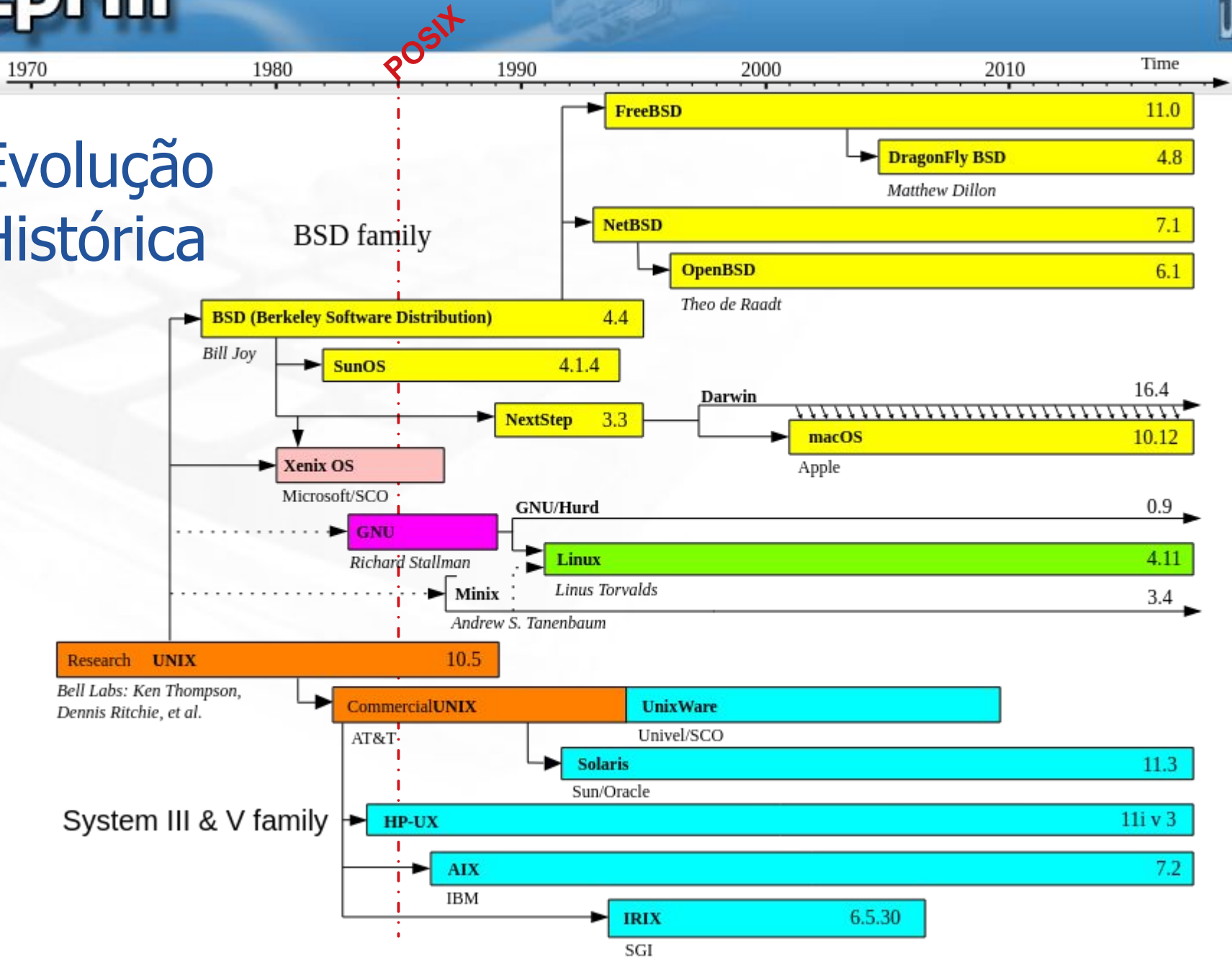
Laboratório de Pesquisa em Redes e Multimídia

Sistemas Operacionais

Processos - Escalonamento no UNIX (RESUMIDO)

Universidade Federal do Espírito Santo
Departamento de Informática

Evolução Histórica



Como projetar um Escalonador

- O projeto de um escalonador deve focar em dois aspectos:
  - ***Política de escalonamento*** – estabelece os princípios gerais e as regras usadas para decidir para qual processo ceder a CPU e quando chaveá-la para um outro processo.
  - ***Implementação*** – define os algoritmos e as estruturas de dados que irão viabilizar a execução destas políticas.
- A política de escalonamento deve buscar:
  - **Tempos de resposta** rápidos para aplicações interativas.
  - Alto ***throughput*** (vazão) para aplicações em *background.*
  - Evitar ***starvation***

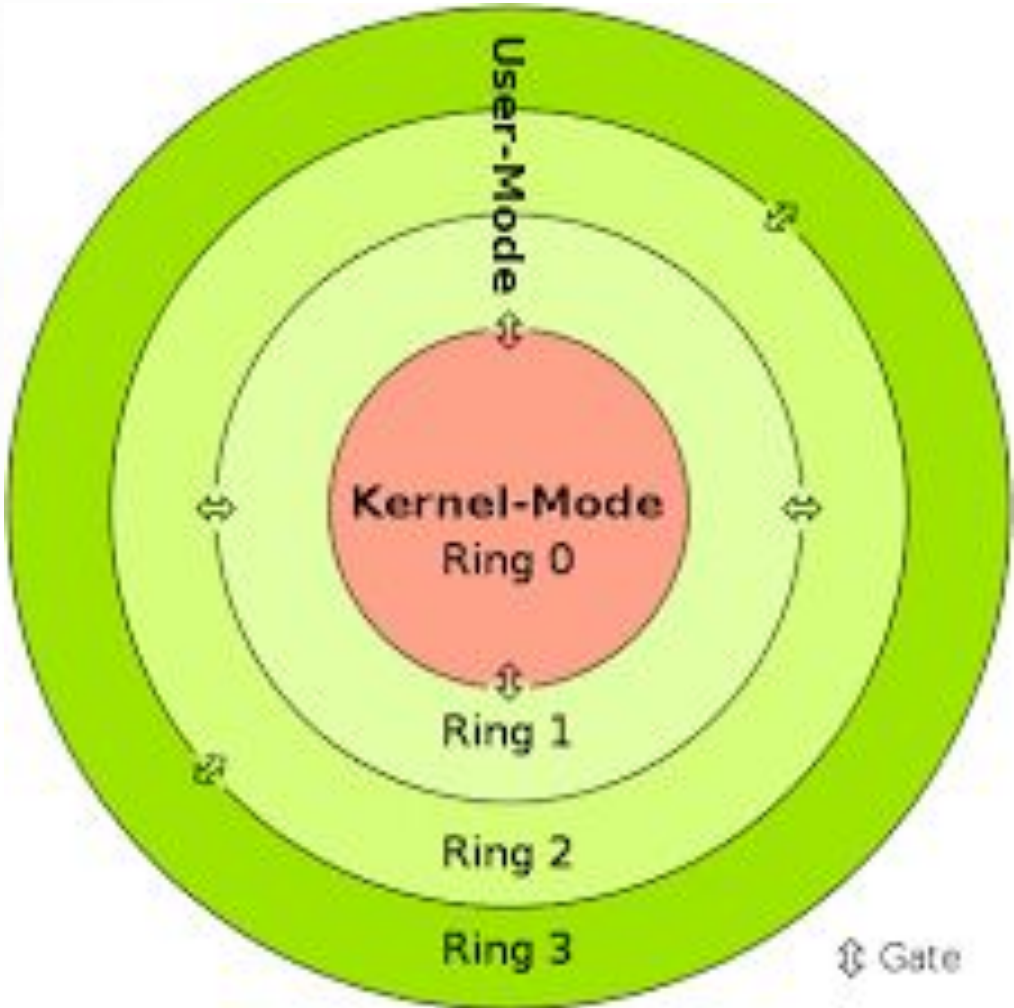
  ... Os objetivos acima podem ser **conflitantes**!

Requisitos do Escalonador para cada tipo de Processo

- Processos **interativos** (*shells*, editores, interfaces gráficas, etc.)
  - Interações devem ser processadas rapidamente
  - Requisito: reduzir os **tempos de resposta** médios (e a variância), de forma que o usuário não detecte/perceba este atraso (50-150 ms)
- Processos ***batch*** (que rodam em *background*, sem interação com o usuário)
  - Requisito: o tempo para completar uma tarefa na presença de outras atividades, comparado ao tempo para completá-la em um sistema "inativo", sem outras atividades paralelas (***turnarround***) deve ser razoável.
- Processos de **tempo real (soft)**
  - Requisito: como as aplicações são "*time-critical*", o escalonador deve ter um comportamento previsível, com limites garantidos nos **tempos de resposta**.
  - Ex: aplicações de vídeo.
- As **funções do kernel**
  - Devem ser executadas prontamente (gerência de memória, tratamento de interrupções e gerência de processos), i.e., bons t**empos de resposta**

Lembrando ...
Modos de Operação da CPU

- Em geral, CPUs oferecem 4 modos de operação:



Antes de mais nada...
Modos de Operação da CPU

- ... mas o Unix requer do hardware a implementação de apenas 2 modos:
  - ***user mode*** - menos privilegiado. Execução de certas instruções e acesso a certos endereços é proibido.
  - ***kernel mode*** - mais privilegiado. Todas as instruções podem ser executadas, acesso à memória é irrestrito.

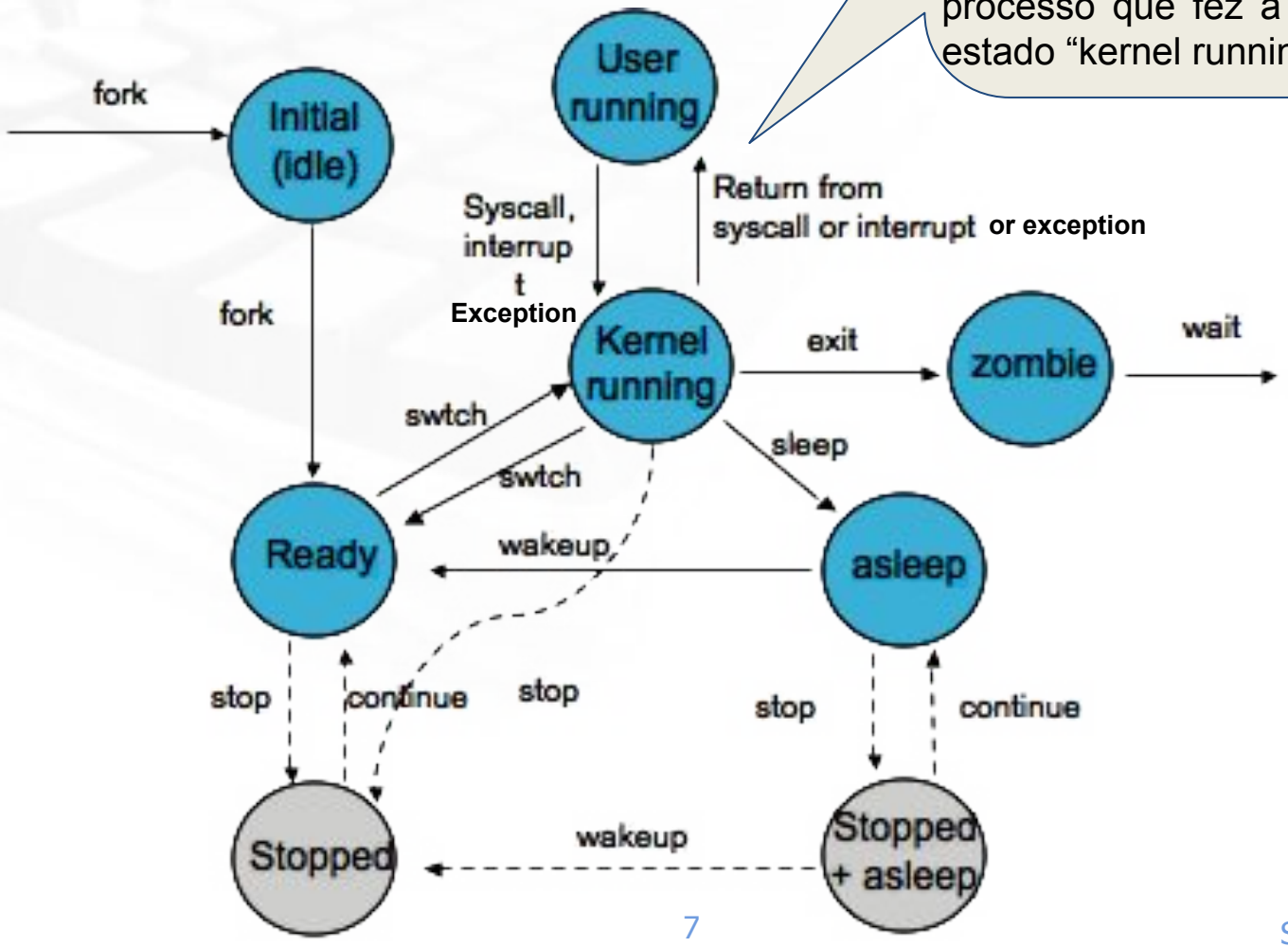
- "**Código de usuário**" é executado em ***user mode***; logo, não podem – acidental ou maliciosamente –, corromper outro processo ou mesmo o kernel.
- "**Código de kernel**" roda em ***kernel mode***

Máquina de Estados do Unix

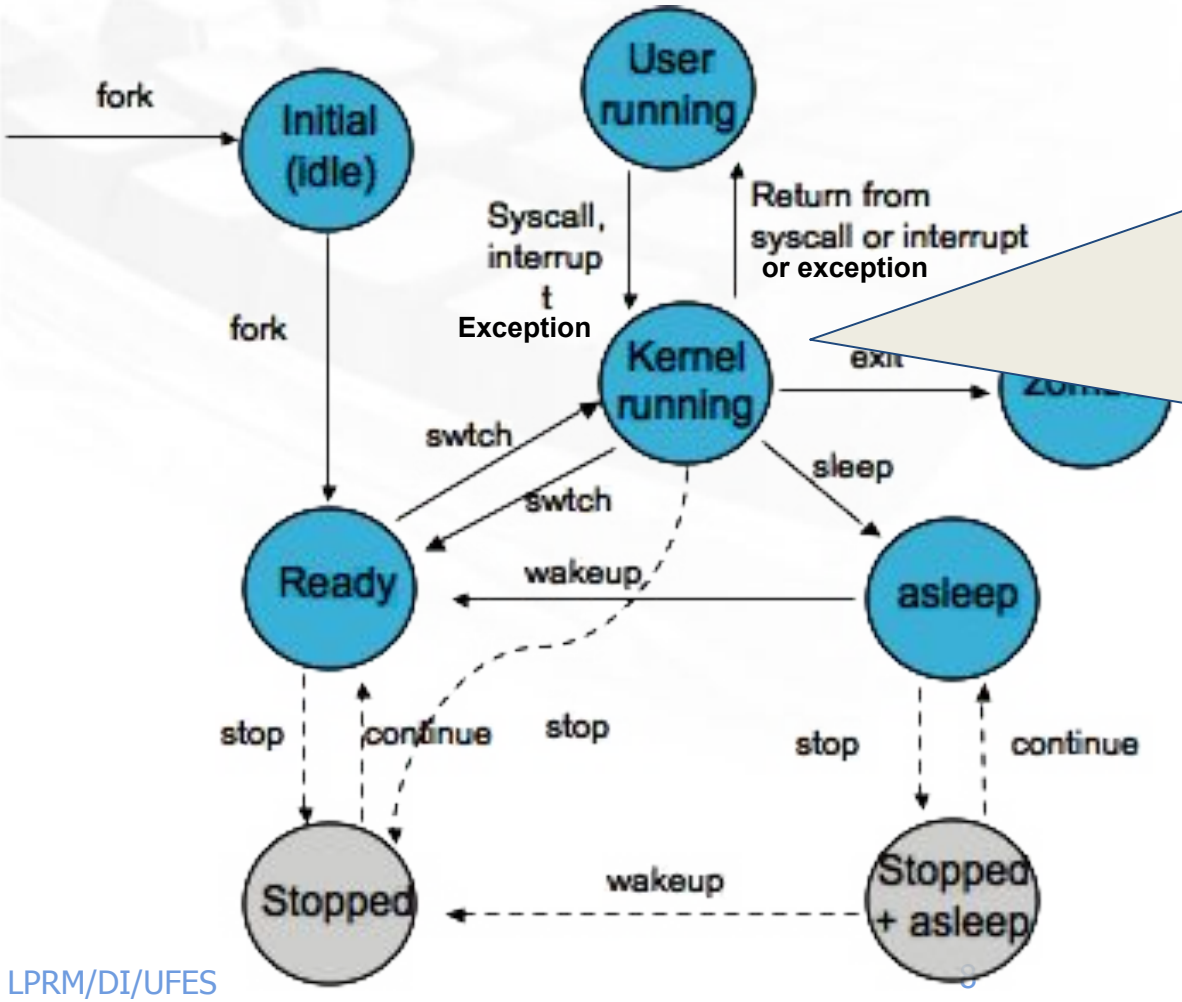
2 estados running!!

- Enquanto o processo executa código de usuário, ele está no “User running.

- Quando um processo, por exemplo, faz uma chamada de sistema, a CPU muda para “kernel mode” passa a executar código de kernel (o processo que fez a SVC vai para o estado “kernel running”)



Máquina de Estados do Unix



2 estados running!!

- Quando um processo "ganha a posse" da CPU pela primeira vez, ele começa rodando no estado "Kernel running", e em seguida passa para "User running" para executar código de usuário.
- Para "perder a posse da CPU", e outro processo ser escalonado, o primeiro processo estará no estado "Kernel Running", porque...
  - ou houve uma interrupção de HW
  - ou ele fez uma SVC
  - ou houve uma exceção

Sleep Priority

- Quando o processo está bloqueado, ele é associado a uma *sleep priority*
  - Relacionada com o dispositivo pelo qual ele está esperando
  - É uma prioridade de kernel !!!
- Quando ele é acordado, ELE PASSA A TER ESSA PRIORIDADE (mas isso é temporário!!)
  - O processo será escalonado **na frente de outros processos** de usuário e continuará a sua execução em modo kernel a partir do ponto de bloqueio.
- Quando a SVC é finalmente completada, imediatamente antes de retornar ao modo usuário, o kernel recupera a prioridade normal do processo

Escalonamento Tradicional

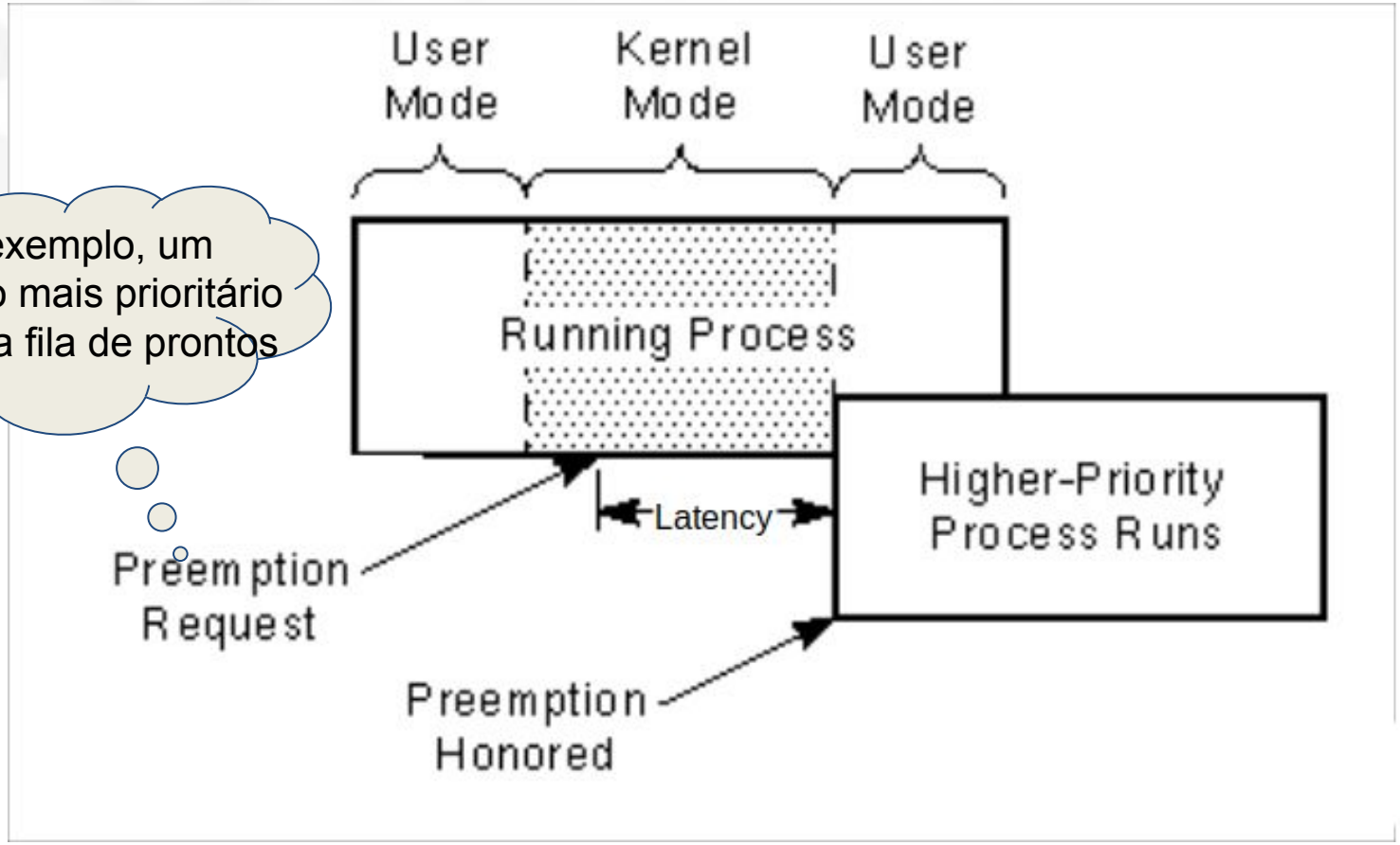
- O BSD implementava o "Escalonamento tradicional!
- O kernel do Unix tradicional é **não-preemptivo**
  - A chegada de um processo de mais alta prioridade na fila de prontos força a preempção do processo em execução **SOMENTE SE** ele está executando em *user mode*
  - Um processo executando em *kernel mode* nunca é preemptado (nunca perde a posse da CPU para um outro processo naquele momento):
    - Quando o sistema **for retornar** para *user mode*, ou seja, quando o que estava sendo executado em kernel mode (por exemplo, uma chamada de sistema) foi finalizado, aí sim ele pode ser preemptado!

Escalonamento Tradicional (cont.)

- O kernel do Unix tradicional é **não-preemptivo**
  - Portanto, a chegada de um processo de mais alta prioridade na fila de prontos **NÃO força a preempção imediata** do processo que está rodando
  - Se o processo está executando em *kernel mode* , código de kernel está sendo executado ... ele deve terminar de executar para então o processo poder ser preemptado:
    - Isso traz mais segurança ao SO já que com isso, a execução de uma rotina de kernel (por exemplo, uma SVC) não é interrompida... a rotina é executada até o final
  - Quando o sistema for retornar para *user mode*, é verificado se há um processo mais prioritário na fila de prontos. Se sim, ocorre a preempção (troca de contexto).

Escalonamento Tradicional (cont.)

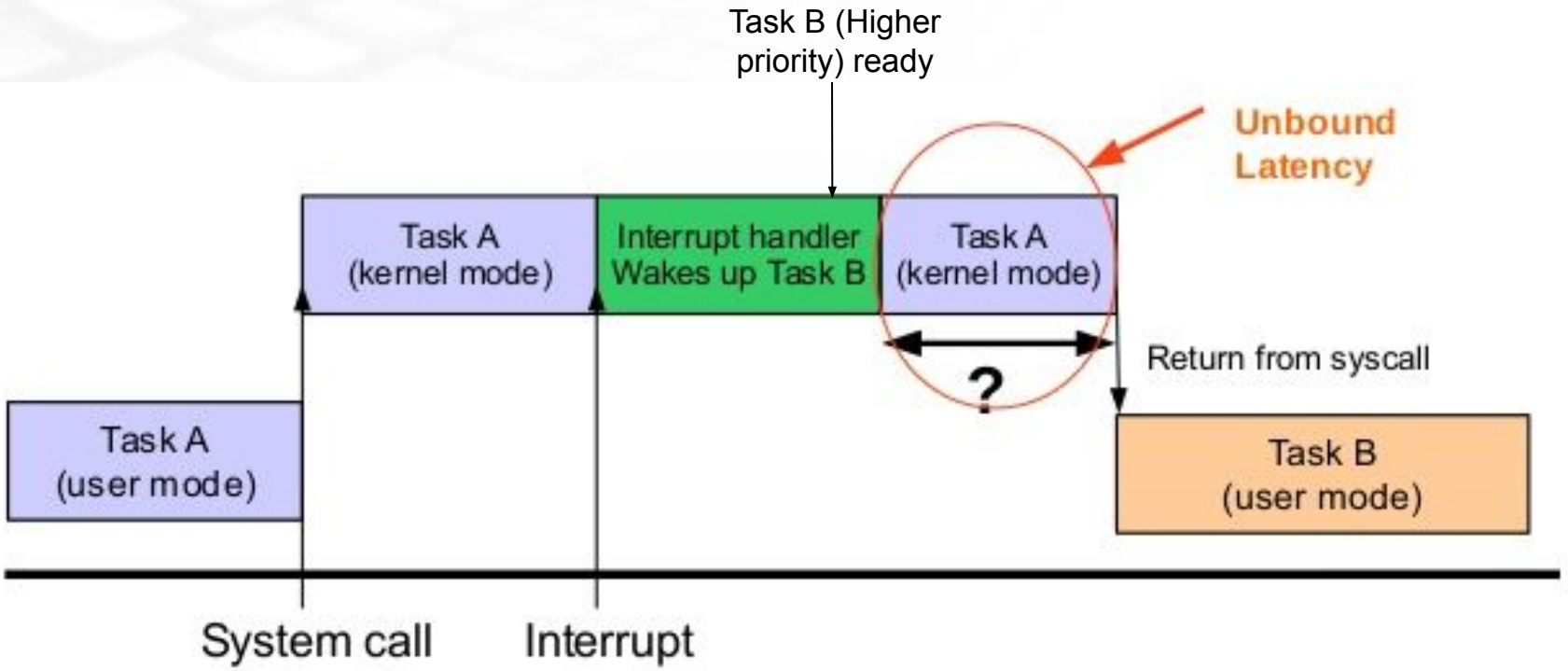
- Kernel não-preemptivo



por exemplo, um processo mais prioritário chega na fila de prontos

Escalonamento Tradicional (cont.)

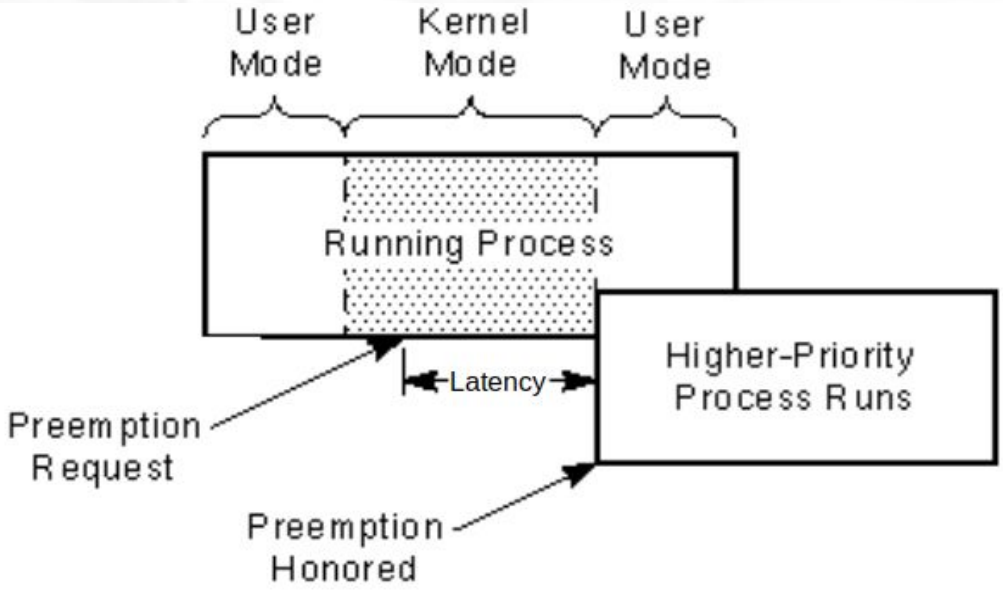
- Kernel não-preemptivo

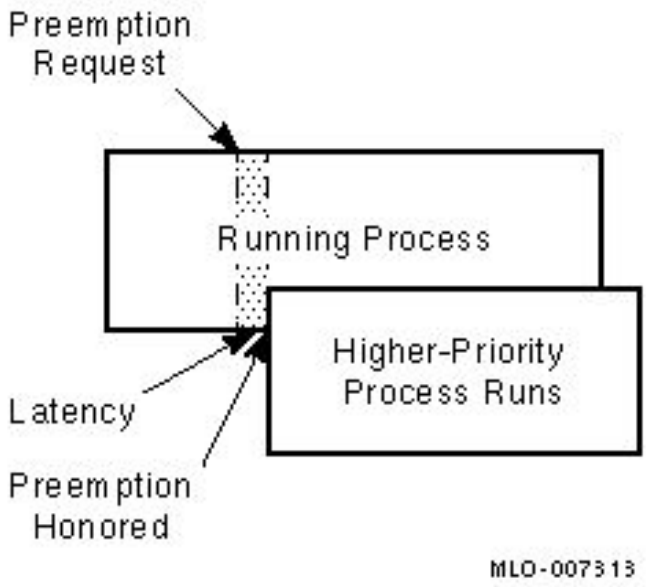


Principal Problema com o Escalonador tradicional

- Não existem garantias de tempos de resposta para aplicações com característica de **tempo-real**.
  - Como o kernel é **não-preemptivo**, processos de maior prioridade podem ter que esperar muito tempo para ganhar a posse da CPU
- **A partir do System V Release 4 (SVR4) o kernel passa a ser <u>PREEMPTIVO</u>**

Kernel preemptivo x Kernel não-preemptivo



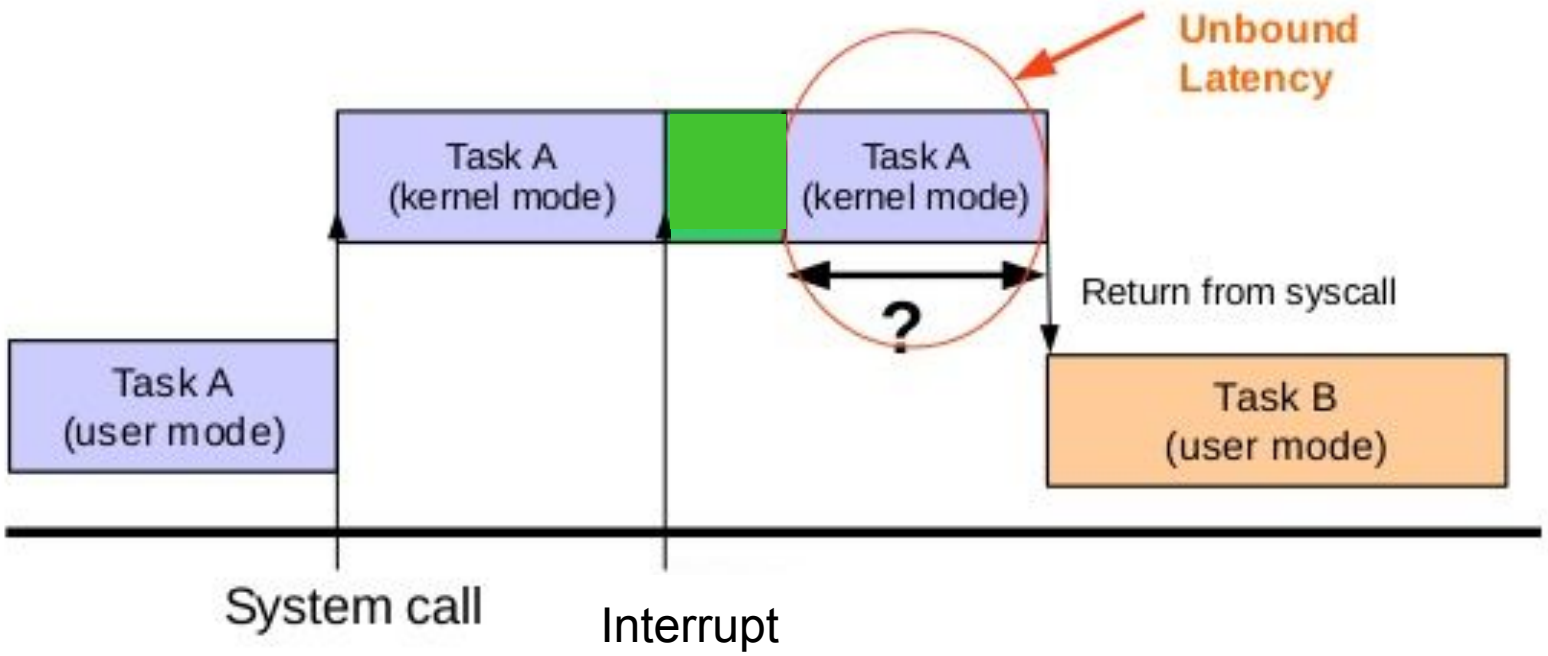


Kernel preemptivo - Implementação no SVR4

- Processos de tempo real exigem tempos de resposta limitados
- **Preemption points** são definidos em diferentes pontos do código do kernel
  - Onde todas as estruturas de dados do kernel **encontram-se estáveis**
  - Ou onde o kernel esteja prestes a iniciar alguma computação longa
- Em cada preemption point
  - O kernel verifica se um processo de tempo-real tornou-se pronto e precisa ser executado
    - O processo corrente é então preemptado
- Os limites nos tempos máximos que um processo de tempo-real precisa esperar são definidos pelo **maior intervalo entre dois preemption points** consecutivos

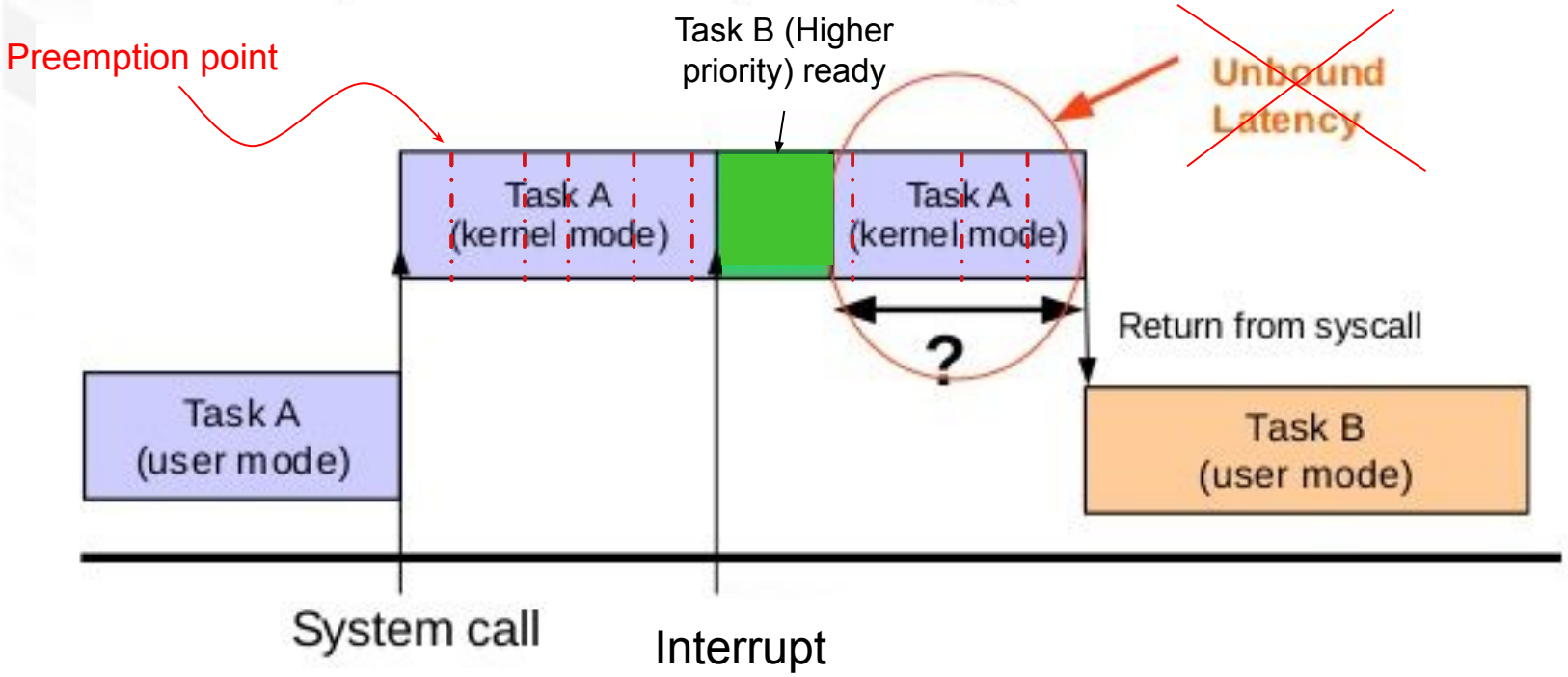
Implementação - SVR4 (5)

- Latency of Non-Preemptive configuration



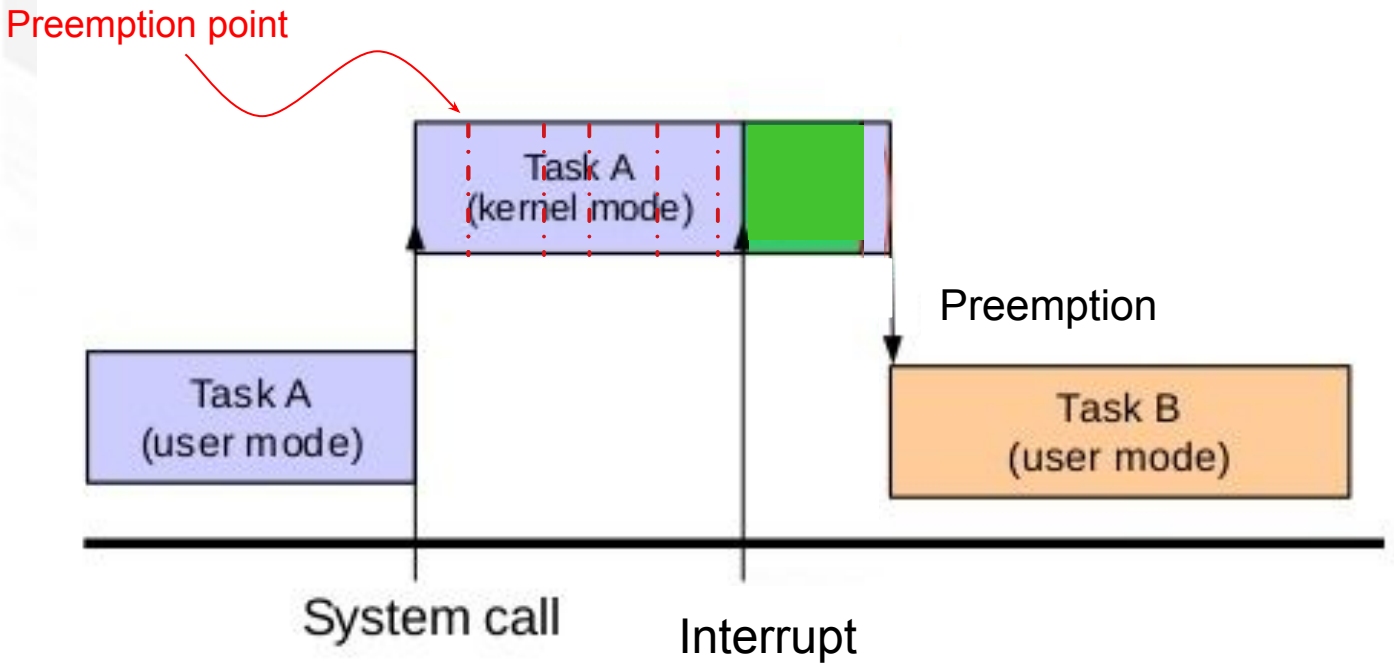
Implementação - SVR4 (5)

- Latency of Non-Preemptive configuration



Implementação - SVR4 (5)

- Latency of Non-Preemptive configuration



Referências

- **VAHALIA, U. Unix Internals: the new frontiers. Prentice-Hall, 1996.**
  - **Capítulo 5 (até seção 5.5)**