



Kernel Linux 2.6

Felipe W Damasio

<felipewd@sl-linux.org>

SL-LINUX

<http://www.sl-linux.org>

Histórico do kernel Linux:

Data de Release X Linhas de código

- 0.01: 09/1991 7.5 K
- 1.0: 03/1994 158 K
- 1.2: 03/1995 277 K
- 2.0: 07/1996 649 K
- 2.2: 01/1999 1536 K
- 2.4: 01/2001 2888 K
- 2.6: ??/2003 ~6000 K

A faint, stylized illustration of Tux, the Linux mascot, is visible in the background. Tux is a black penguin wearing a white shirt, a red scarf, a brown hat, and a brown belt with a large yellow buckle. He is holding a small object in his right hand.

Kernel Linux 2.6

- Feature Freeze: Novembro de 2002
- Melhor Performance, especialmente em **SMP**
- Melhor Escalabilidade
- Melhor subsistema de I/O
 - Novos Sistemas de Arquivos
- **Vários** drivers novos
- Novas plataformas



Novas *Features*:

Visão Geral

- Kernel Preemptivo
- Escalonador $O(1)$
- Melhor CPU *Affinity*
- Melhoria no suporte a *Threads* POSIX
 - NGPT
 - NPTL
- Melhor balanceamento de IRQ



Novas *Features*:

Visão Geral

- Melhor suporte a ACPI
- USB 2.0
- Novo sub-sistema de SOM (ALSA)
- Suporte a LSM (Linux Security Model)
- Nova Estrutura de Dispositivos
- Driver Hardware Sensor (lm-sensors)
- Melhor suporte a clusters **NuMA**

Novas *Features*:

Visão Geral

- Configuração de voltagem e clock de CPU
- Mapeamento reverso de páginas (rmap)



Novas *Features*:

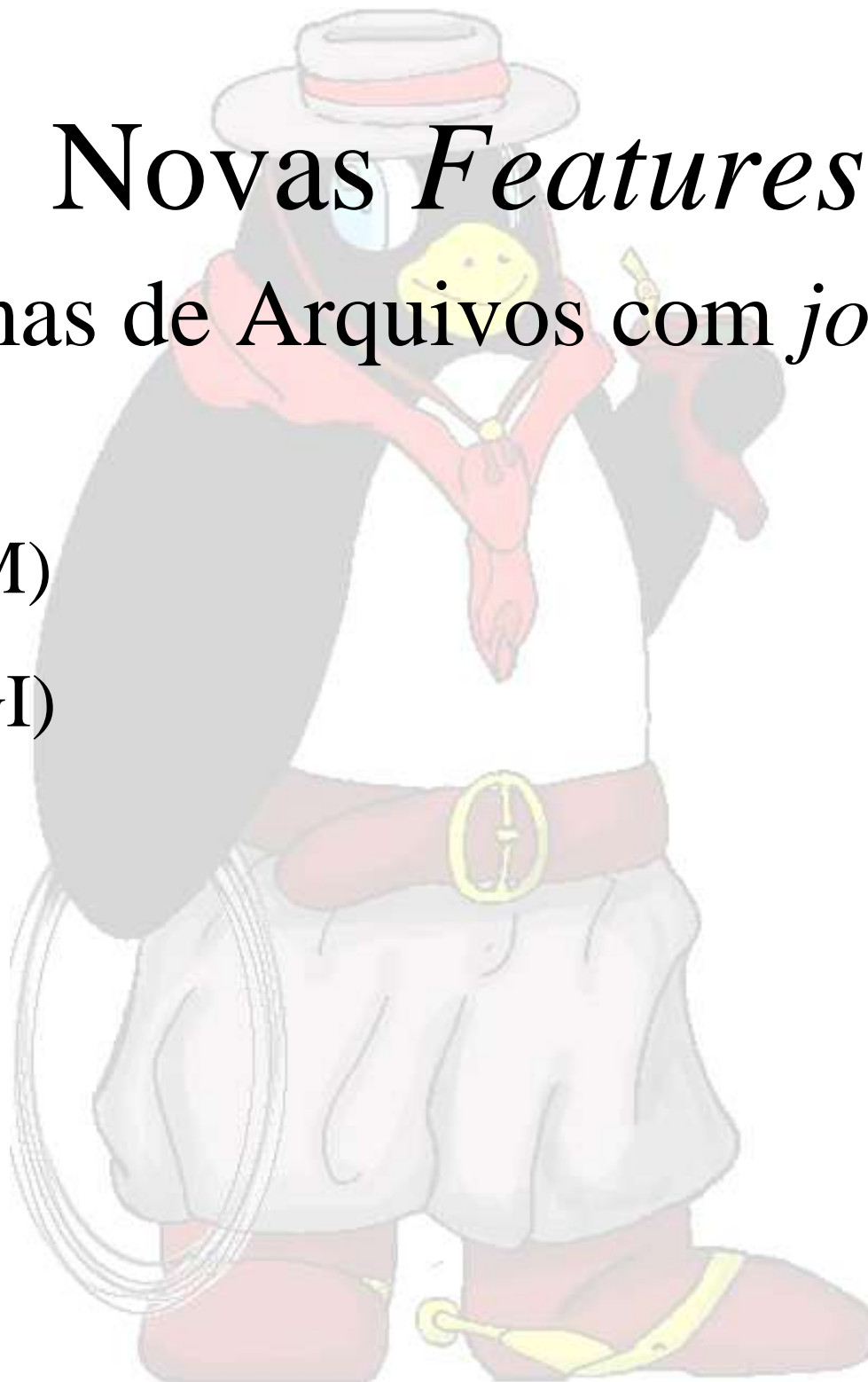
Arquiteturas

- AMD “Hammer” 64-bits (x86-64)
- PowerPC 64-bit (ppc64)
- Sistemas Embedded
 - Motorola 68k (sem MMU)
 - DEC V850
- User Mode Linux (UML)

Novas *Features*:

Sistemas de Arquivos com *journaling*

- Reiser4
- JFS (IBM)
- XFS (SGI)





Novas *Features*:

Camada de I/O

- Reescrita do Block I/O Layer (BIO)
- I/O Assíncrono
- Atualização do sub-sistema de IDE
- Suporte a ACL (Access Control Lists)
- Novo driver NTFS



Novas *Features:*

Redes

- NFSv4
- TCP Segmentation Offload
- Suporte a SCTP
(Stream Control Transmission Protocol)
- Suporte a Bluetooth
- NAPI (Network Interrupt Mitigation)

Novas *Features*:

Novo Mantenedor



Marcelo Tosatti
Mantenedor do Kernel 2.4



Andrew Morton
Mantenedor do Kernel 2.6

Escalonador $O(1)$

Introdução

- O Escalonador decide qual processo vai executar num dado momento
- Num sistema SMP, decide qual processo vai ser executado em qual processador
- Deve ser rápido e **justo!**

Escalonador $O(1)$

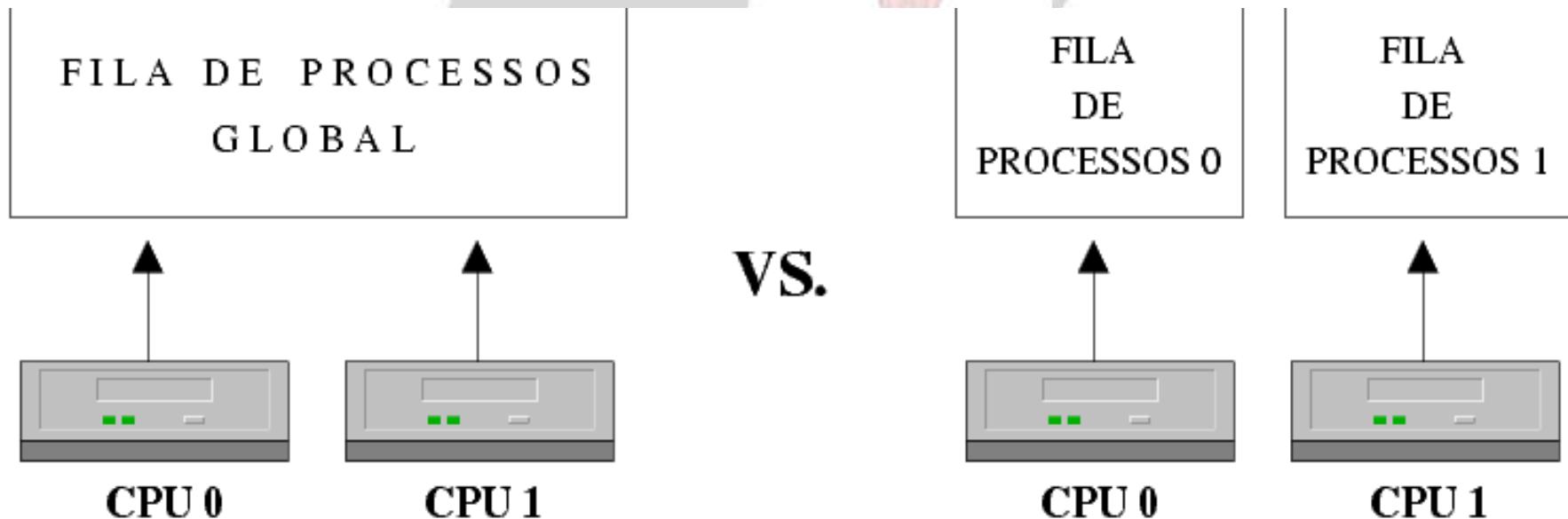
Funcionamento

- Feito por Ingo Molnar:
 - www.kernel.org/pub/linux/kernel/people/mingo
- Multi-queue scheduler:
 - Uma fila de processos pra cada processador
- Todos os algoritmos são $O(1)$
 - Tempo de execução constante
 - Independe do número de processos (ou de qualquer outro fator)

Escalonador O(1)

Multi-queue Scheduler

- Cada processador tem uma fila processos
- Cada fila de processos tem uma *lock*



Kernel 2.4: Apenas 1 fila de processos

Kernel 2.6: Uma fila de processos por processador

Escalonador $O(1)$

$O(1)$ vs. $O(n)$

- Escalonador do kernel 2.4: $O(n)$:

```
Para (cada processo no sistema) {  
    ache o valor de "merecimento" do processo  
    Se (esse é o processo com maior "merecimento") {  
        Lembre-se dele  
    }  
}  
Rode o processo com maior merecimento
```

- Escalonador do kernel 2.6: $O(1)$

```
Valore o primeiro processo por prioridade  
Pegue o primeiro processo nessa fila  
Rode esse processo
```


Escalonador $O(1)$

SMP Affinity

- No antigo escalonador, processos ficavam rodando ora num processador, ora em outro:

Pior-caso do efeito ping-pong

CPU	Tempo 1	Tempo 2	Tempo 3
Processo A	CPU 0	CPU 1	CPU 0
Processo B	CPU 1	CPU 0	CPU 1

Escalonador $O(1)$

SMP Affinity

- No escalonador $O(1)$, esse problema é resolvido pelas Multi-Level queues

Pior-caso do efeito ping-pong

CPU	Tempo 1	Tempo 2	Tempo 3
Processo A	CPU 0	CPU 0	CPU 0
Processo B	CPU 1	CPU 1	CPU 1

Escalonador $O(1)$

$O(1)$ vs. $O(n)$

- Escalonador $O(1)$ é muito mais rápido, principalmente em máquinas SMP

Benchmark ChatServer testa a troca de mensagens entre um grande número de processos. **Resultados em mensagens/segundo**

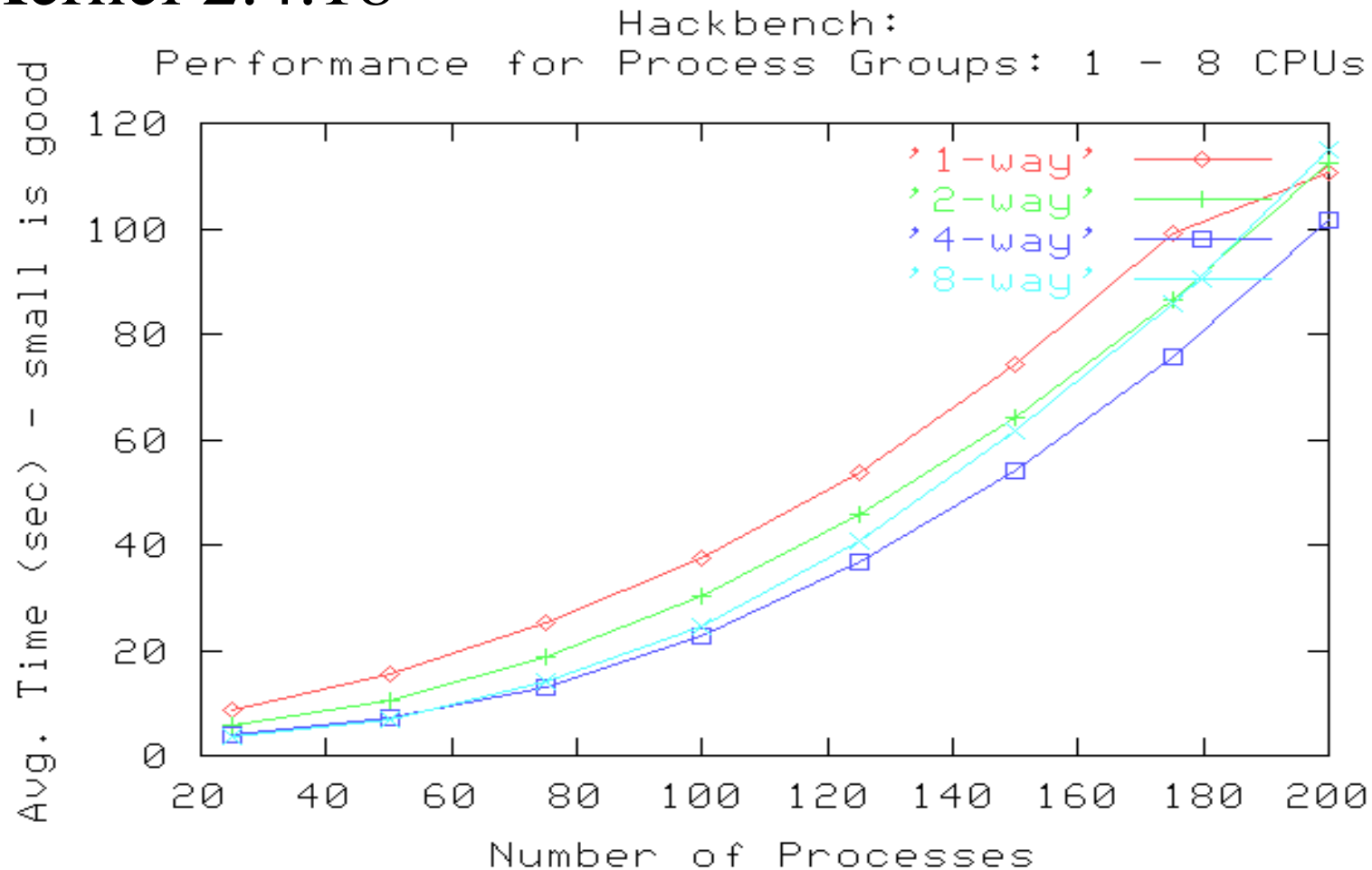
Escalonador	Rodada 1	Rodada 2	Rodada 3
Escalonador 2.4	79723	82210	94803
Escalonador 2.6	612320	620880	609420

Fonte: Linux Journal, Maio de 2003

Escalonador $O(1)$

$O(1)$ vs. $O(n)$

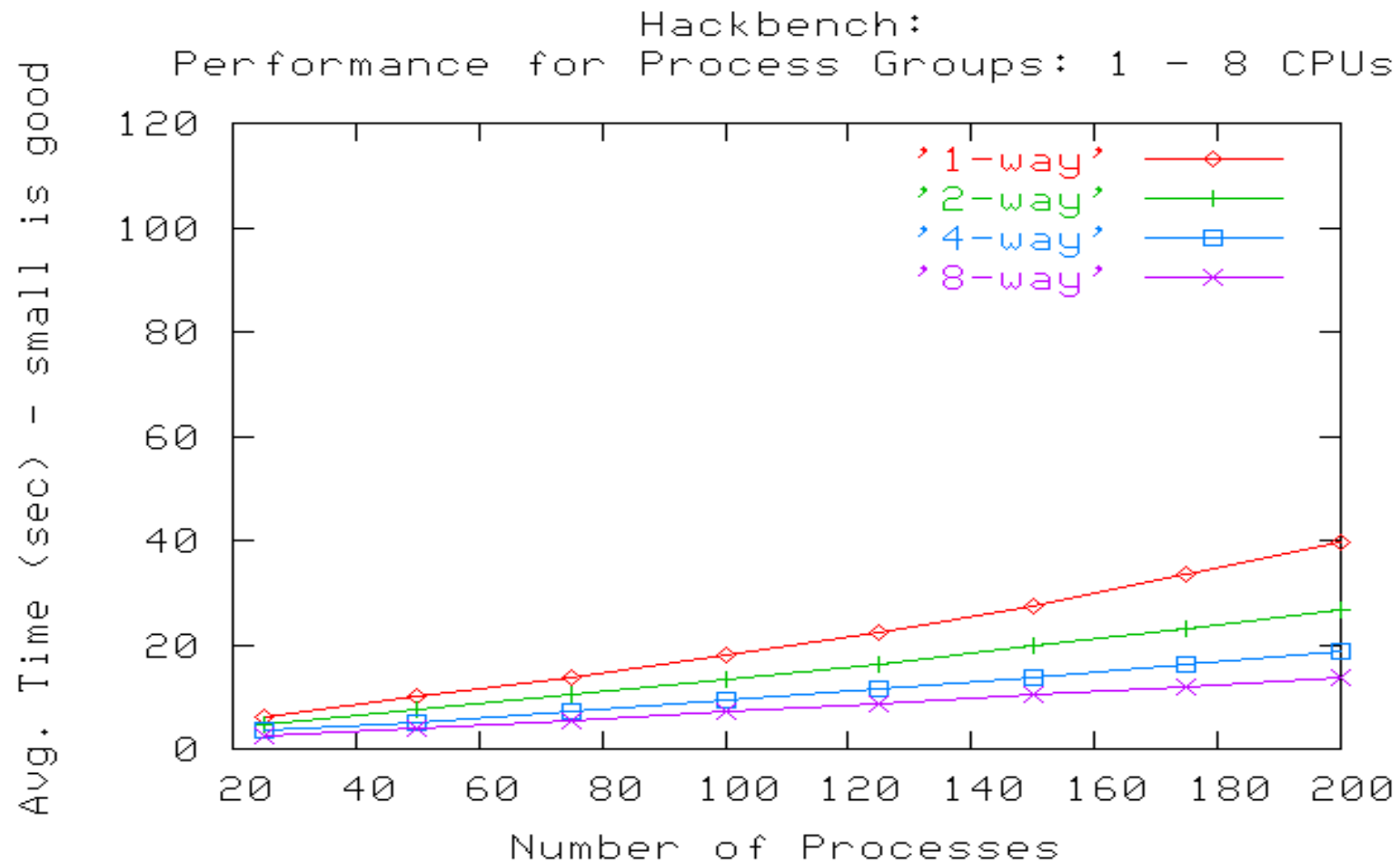
- Kernel 2.4.18



Escalonador $O(1)$

$O(1)$ vs. $O(n)$

- Kernel 2.6.0-test9



Kernel Preemptivo:

Visão Geral

- Feito por Robert Love:
 - www.kernel.org/pub/linux/kernel/people/rml
- Kernel 2.4: Sem preempção em kernel
 - Execução em espaço de kernel só é interrompida por IRQ's, sleep, ou chamando o escalonador
 - **Horrível** tempo de resposta! (Aplicações Realtime)
- Kernel 2.6: O kernel pode ser preemptado
 - Um novo processo pode ser escalonado a “qualquer” momento

Kernel Preemptivo:

Exemplo

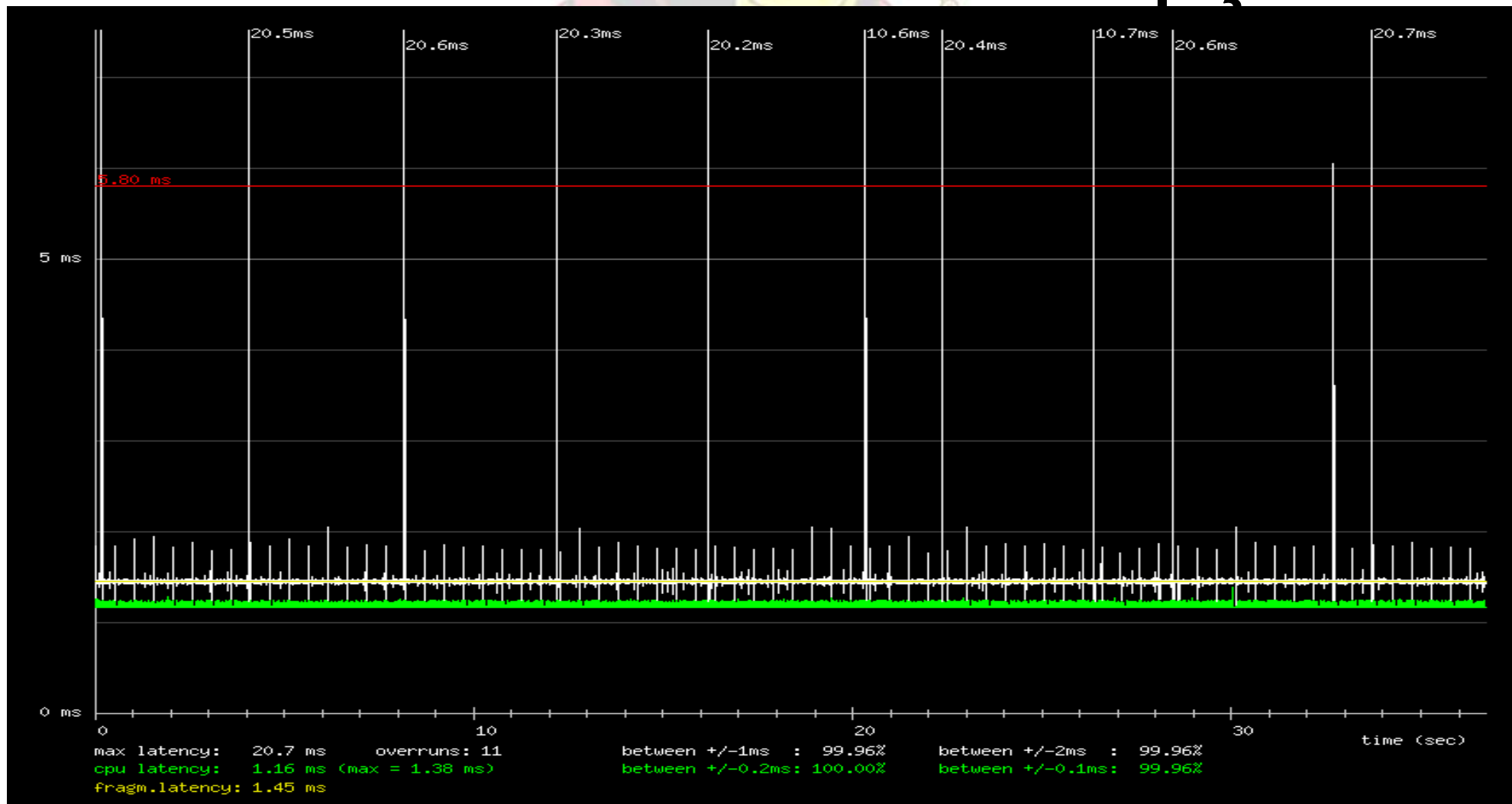
- Kernel antigo:
 - Kernel executa chamada de sistema pro processo A
 - Processo B (de maior prioridade) pode ser executado
 - Kernel termina de executar a chamada para A
- Kernel 2.6:
 - Kernel executa chamada de sistema pro processo A
 - Processo B (de maior prioridade) pode ser escalonado
 - Processo B começa a executar

Kernel Preemptivo:

Ganhos e perdas

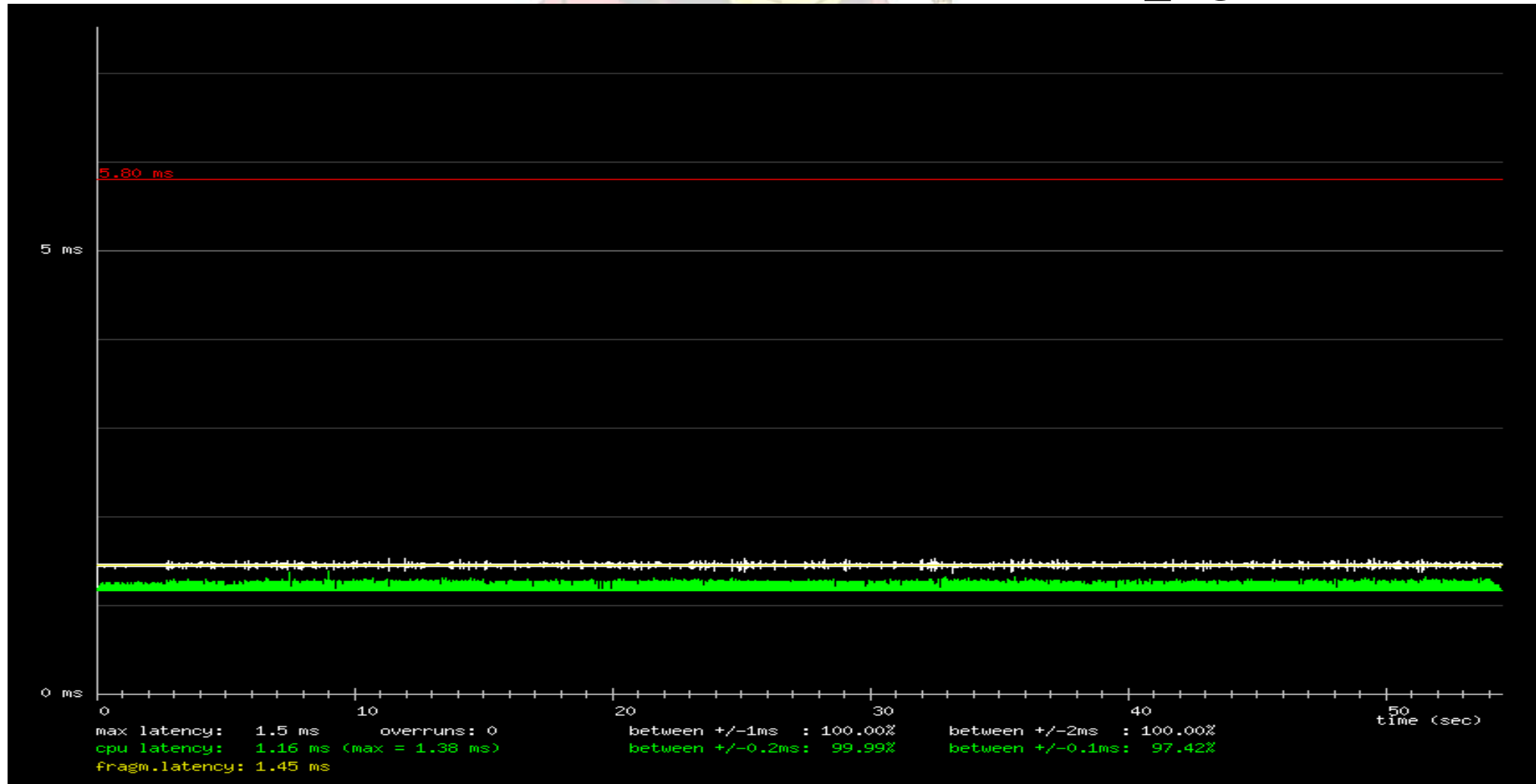
- Latência do sistema diminui consideravelmente
- Ótimo para usuários desktop
 - Ganho de até 200% na performance do sistema!
 - Teste RealFeel mostrou queda de latência de 78,51ms para 0,48ms
- Ruim para servidores
 - Diminui o *throughput* do sistema
 - Entre 0 e 5% de perda

Latência sem Preempção



Fonte: The Linux kernel preemption project

Latência com Preempção



Fonte: The Linux kernel preemption project



Threads POSIX NGPT

- Next Generation POSIX Threads (NGPT)
- Em média 2 vezes mais rápida que a LinuxThreads (usada em sistemas com kernel 2.4)
- Melhor POSIX Compliance
- Modelo M:N (M threads de kernel para N threads de usuário, com $M < N$)
- **[http://www.124.ibm.com/developerworks /
oss/pthreads](http://www.124.ibm.com/developerworks/oss/pthreads)**



Threads POSIX NPTL

- Native POSIX Thread Library (NPTL)
- Feito por Ingo Molnar e Ulrich Drepper - Red Hat
- Modelo 1:1
- **<http://people.redhat.com/drepper/nptl-design.pdf>**



Threads POSIX

NPTL - Descrição

- Mudanças no Kernel:
 - Fast User-space Mutexes” (futexes)
 - Chamada `exit` $O(n)$ foi substituída por `exit` $O(1)$
 - Suporte a threads CPU Affinity

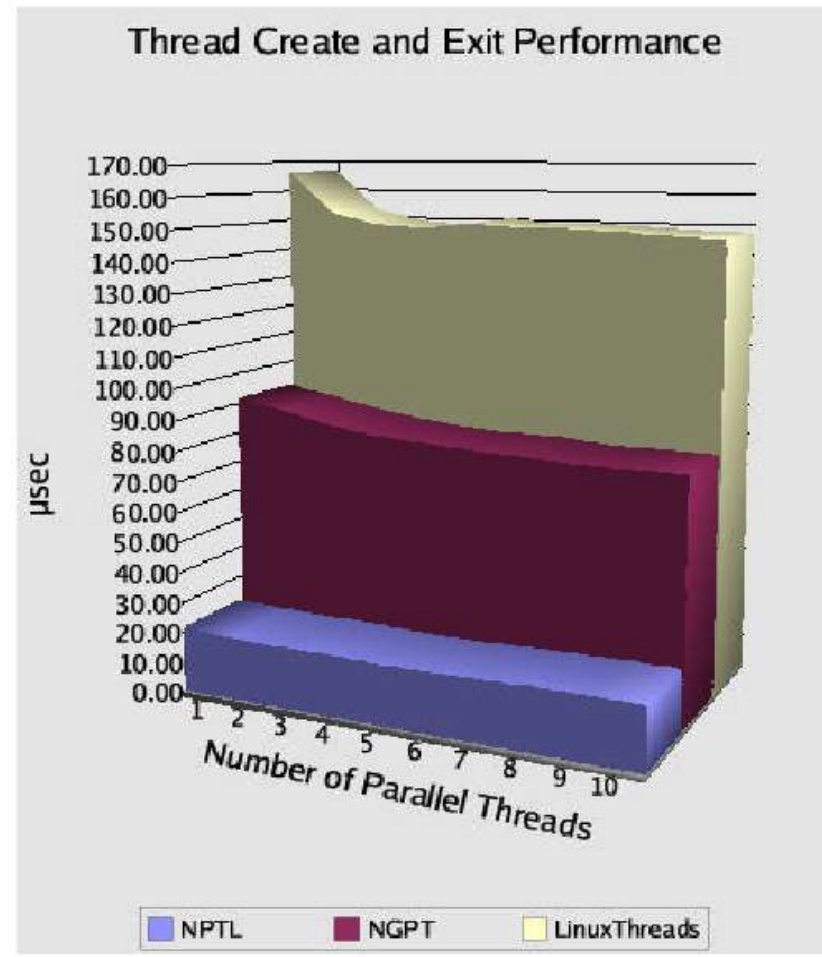
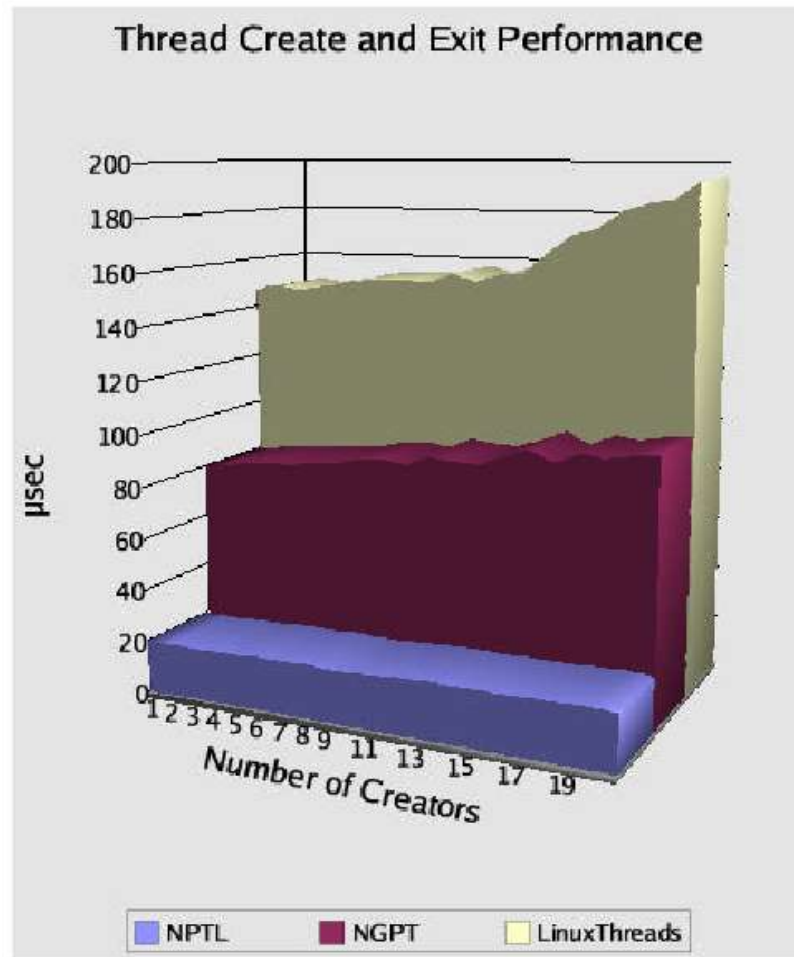


Threads POSIX

NPTL - Performance

- Performance da NPTL no kernel 2.6:
 - 8x mais rápido que a LinuxThreads (kernel 2.4)
 - Desempenho da chamada `exit` melhorou 400% para 4000 Threads!
 - Criação e destruição de 100.000 threads (paralelas) demora menos de **2 segundos**.
 - No kernel 2.4 com LinuxThreads, a mesma operação demora **15 minutos**.
 - Uau!

Threads POSIX Benchmark



<http://people.redhat.com/drepper/perf-s-100000-pro{par}.pdf>



Software suspend

- Salva o estado dos processos da máquina numa partição de *swap*
 - Ativada por *sysrq-d* ou por uma opção no “shutdown”
- Restaura o estado no próximo boot
- Tudo em software. Não é necessário APM!
- **<http://swsusp.sourceforge.net/>**

A cartoon penguin character is the background of the slide. It is wearing a light-colored wide-brimmed hat with a red band, a red scarf, a white shirt with a red tie, and light-colored pants with a red belt and a large gold buckle. It is also wearing red boots with yellow straps. The penguin is holding a small USB drive in its right hand.

USB 2.0

- Substitui USB 1.1
- Divide transações em fases (“início” e “fim”)
- Aumento da velocidade:
 - USB 2.0: 480 Mbit/sec
 - USB 1.1: 12 Mbit/sec
- **<http://www.linux-usb.org/usb2.html>**



ALSA

(Advanced Linux Sound Architecture)

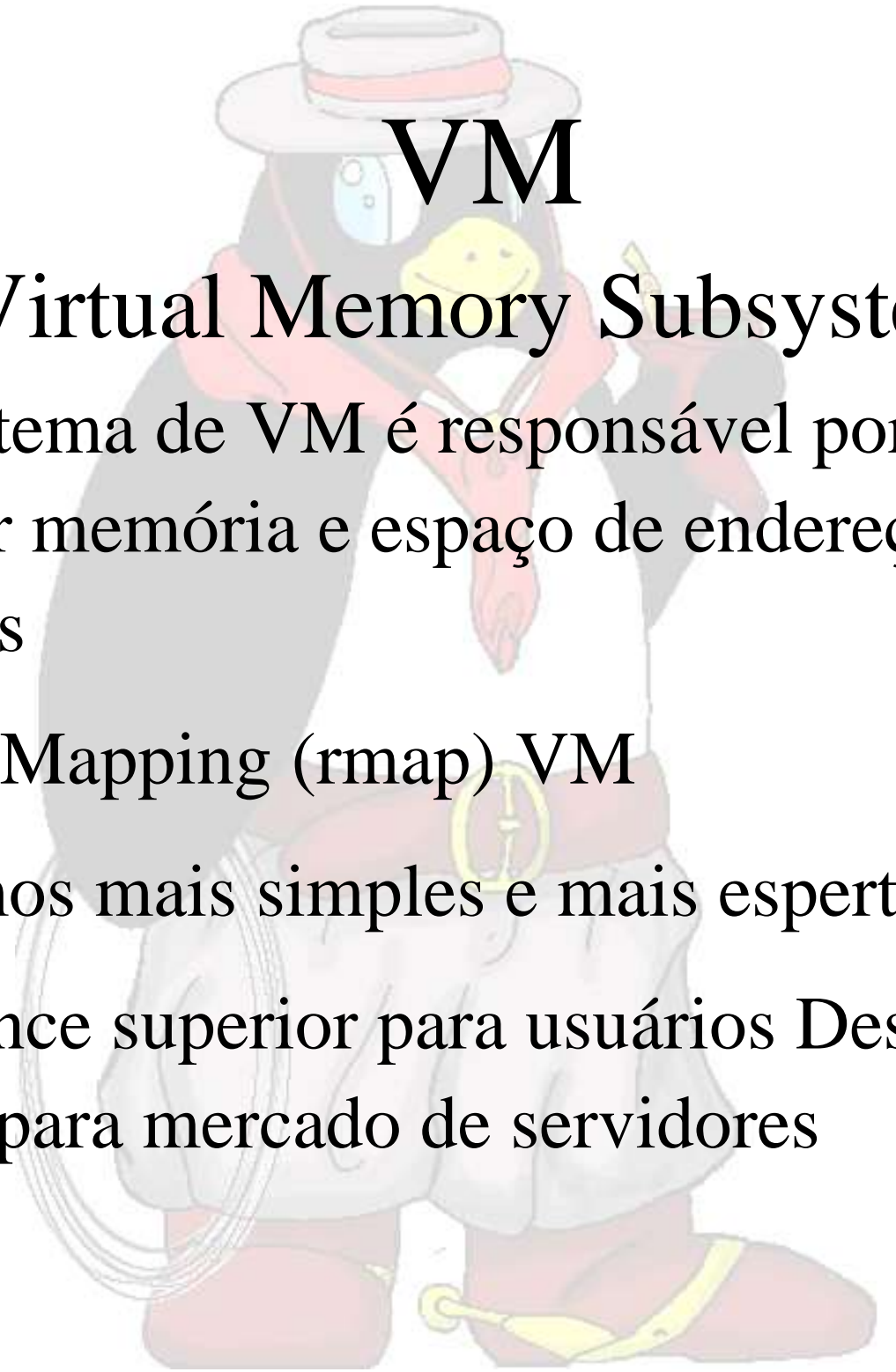
- Áudio e MIDI. Substitui OSS (Open Sound System)
- Tanto SMP quanto thread-safe
- Suporte a muito mais placas.
 - Otimizações específicas ao hardware
 - Também possui drivers genéricos
- Possui biblioteca em espaço de usuário pra aplicativos
- <http://www.alsa-project.org>



LSM

(Linux Security Modules)

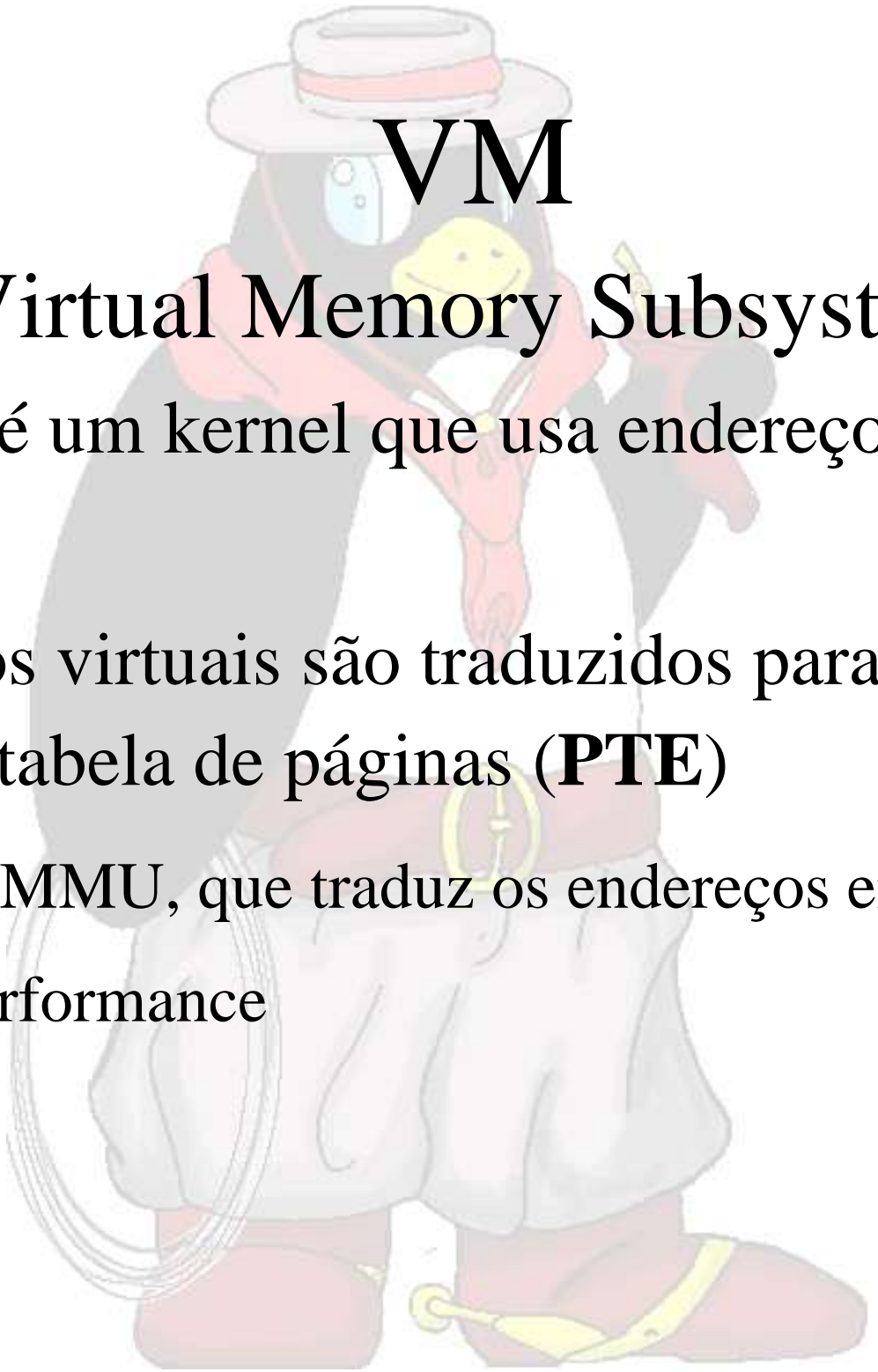
- Framework em kernel pra adicionar módulos de segurança
- Campos de segurança adicionado a estruturas no kernel
- Patches feitos pela NSA
- **<http://lsm.immunix.org>**



VM

(Virtual Memory Subsystem)

- O subsistema de VM é responsável por alocar e gerenciar memória e espaço de endereçamento de processos
- Reverse-Mapping (rmap) VM
- Algoritmos mais simples e mais espertos
- Performance superior para usuários Desktop e também para mercado de servidores

A cartoon illustration of Tux, the Linux mascot, dressed as a cowboy. He is wearing a brown cowboy hat with a red band, a red bandana, a red vest over a white shirt, and a brown belt with a large yellow buckle. He is holding a lasso in his right hand and a yellow object, possibly a banana, in his left hand. The background is a light gray.

VM

(Virtual Memory Subsystem)

- O Linux é um kernel que usa endereços virtuais e reais
- Endereços virtuais são traduzidos para endereços reais via tabela de páginas (**PTE**)
 - Usa-se MMU, que traduz os endereços em hardware
 - Alta performance



VM

kernel 2.4

- Quando for liberar memória, o kernel precisa saber quais **PTE's** referenciam uma certa página física.

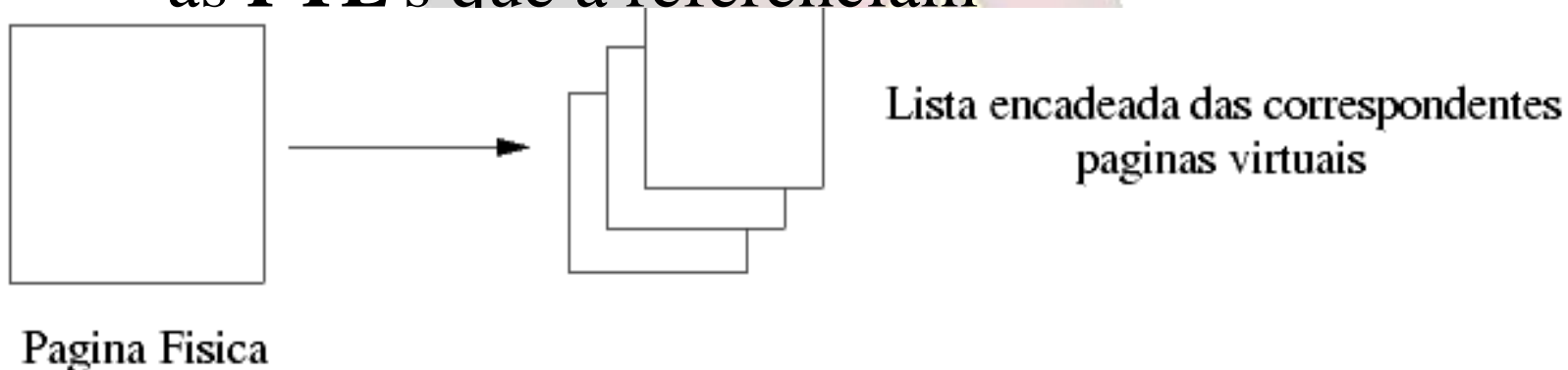
```
Para (cada entrada de tabela de páginas) {  
    Se (esse endereço físico é o que estou procurando){  
        achamos um endereço virtual  
    }  
}  
Libere todos os endereços encontrados
```

- Algoritmo ineficiente: $O(n)$

VM

kernel 2.6: rmap

- Feito por Rik Van Riel:
- Reverse Mapping:
 - Criado uma estrutura pra cada página física que lista as **PTE's** que a referenciam



Kernel 2.6: RMAP – Reverse Mapping uma pagina fisica pra uma ou mais paginas virtuais

A faint, stylized background image of Tux, the Linux mascot penguin, dressed as a cowboy. He is wearing a brown cowboy hat with a red band, a red bandana around his neck, a red vest over a white shirt, and a brown belt with a large gold buckle. He is holding a yellow lasso in his right hand.

VM

kernel 2.6: Huge Page Support

- Na maioria das arquiteturas, o Linux usa páginas de 4KB ou 8KB
- Muitas CPU's suportam páginas maiores:
 - **X86** pode usar 4MB
- Requer menos **PTE's** e aproveita melhor **TLBs**
 - **TLB** faz cache das traduções de endereços virtuais -> reais
- Aumenta 30% a performance do sistema!

Journaling Filesystems

Introdução

- Operações são agrupadas em **transações**
- **Transações** são completadas **atomicamente**
- Arquivo de Log grava cada transação
- Numa falha do sistema, apenas as últimas transações precisam ser checadas
- Resultado: **fsck** se torna mais rápido (alguns segundos)

Journaling Filesystems

Sistemas de arquivos no kernel 2.6

- **EXT3** (2.4) – Extensão do EXT2, mesmo layout de partição, migração simples
 - <http://e2fsprogs.sourceforge.net/ext2.html>
- **ReiserFS** (2.4): <http://www.namesys.com>
- **JFS** (IBM, AIX):
<http://oss.software.ibm.com/developerworks/opensource/jfs>
- **XFS** (SGI, IRIX):
<http://oss.sgi.com/projects/xfs>



Block I/O

Introdução

- Tradicionalmente, UNIX suporta 2 tipos de dispositivos
 - Char devices: Dispositivos que manipulam dados byte a byte
 - Exemplos: Porta serial, teclado, mouse
 - Block devices: Dispositivos que manipulam dados em grupos de dados de tamanho fixo (“blocos” de I/O)
 - Exemplos: Discos Rígidos, CD-ROM, drives de fita



Block I/O

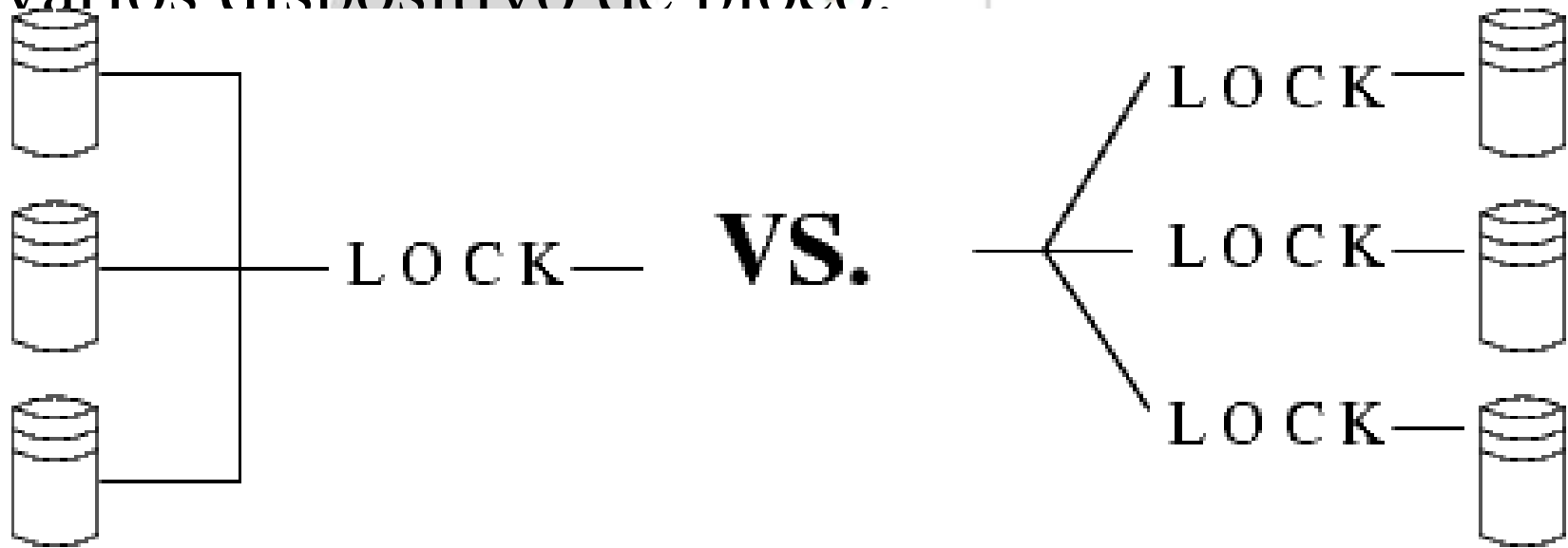
Bounce-buffers

- Uma transferência de I/O de um dispositivo de bloco para high-memory precisa fazer um passo intermediário
 - High-memory em x86 são endereços acima de 1GB (na RAM)
 - Bounce-buffers é o mapeamento de um endereço em low-memory como trampolim

Block I/O

Escalabilidade

- Remoção da `io_request_lock`
- Grande aumento na performance num sistema com vários dispositivos de bloco.



Kernel 2.6: Uma lock por request queue

Block I/O

Anticipatory I/O Scheduler

- Leituras são operações “dependentes”
 - Para a leitura N acontecer, a leitura N-1 deve ter sido completada
- Caso hajam muitas escritas em paralelo a leituras, as leituras demoram muito!



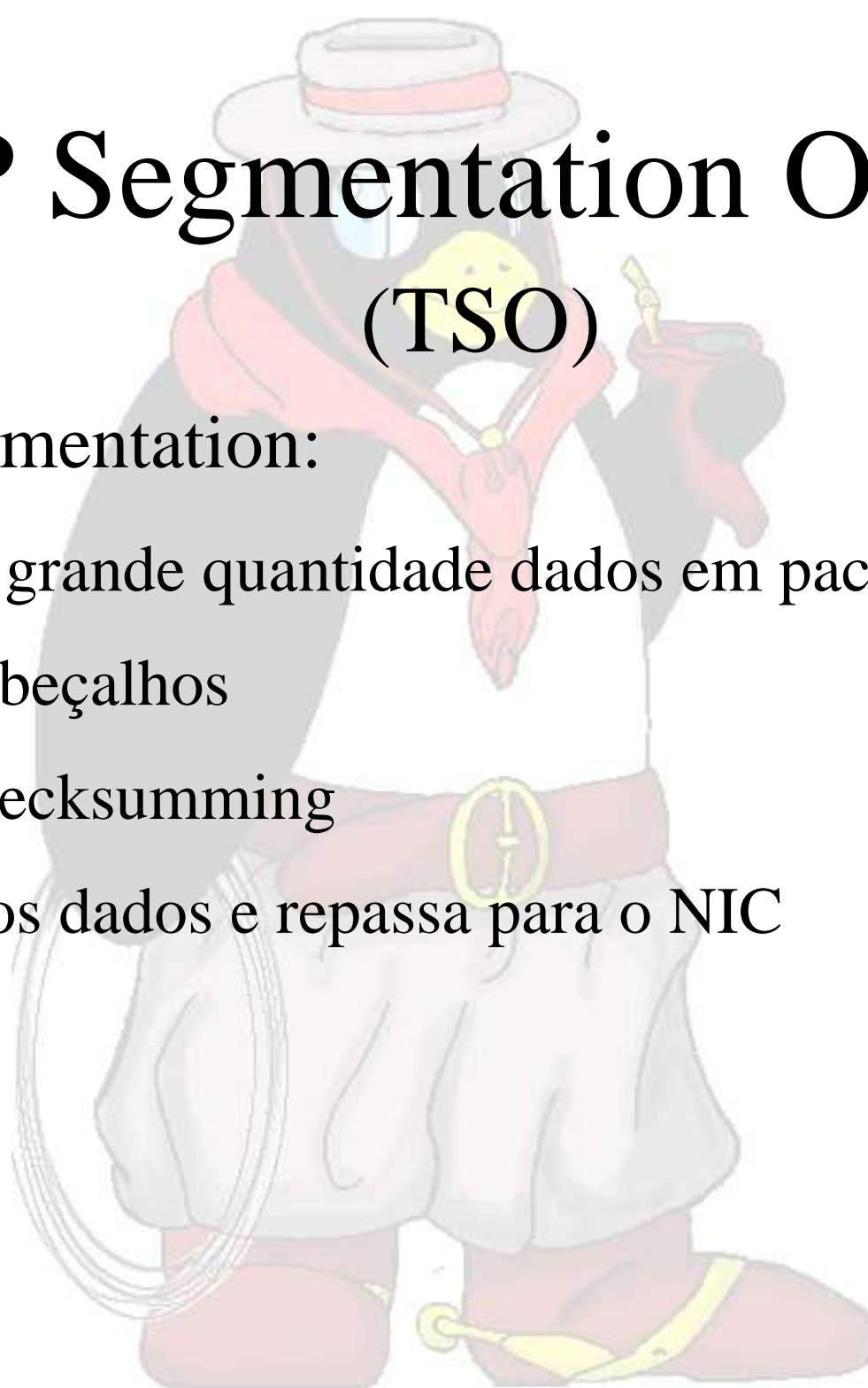
Block I/O

Anticipatory I/O Scheduler

- Aumenta a performance
- Faz menos seek...aumenta a longevidade do disco!
 - Leituras em paralelo (arquivos de 512MB):
 - Kernel 2.4: **31 minutos e 30 segundos**
 - Kernel 2.6: **17 segundos**
 - Aumentou **111x** a performance
 - Infinitas escritas de 1MB e (foram medidas) múltiplas leituras
 - Kernel 2.4: **Várias Horas**

TCP Segmentation Offload (TSO)

- TCP Segmentation:
 - Divide grande quantidade dados em pacotes
 - Cria cabeçalhos
 - Faz Checksumming
 - Copia os dados e repassa para o NIC



TCP Segmentation Offload (TSO)

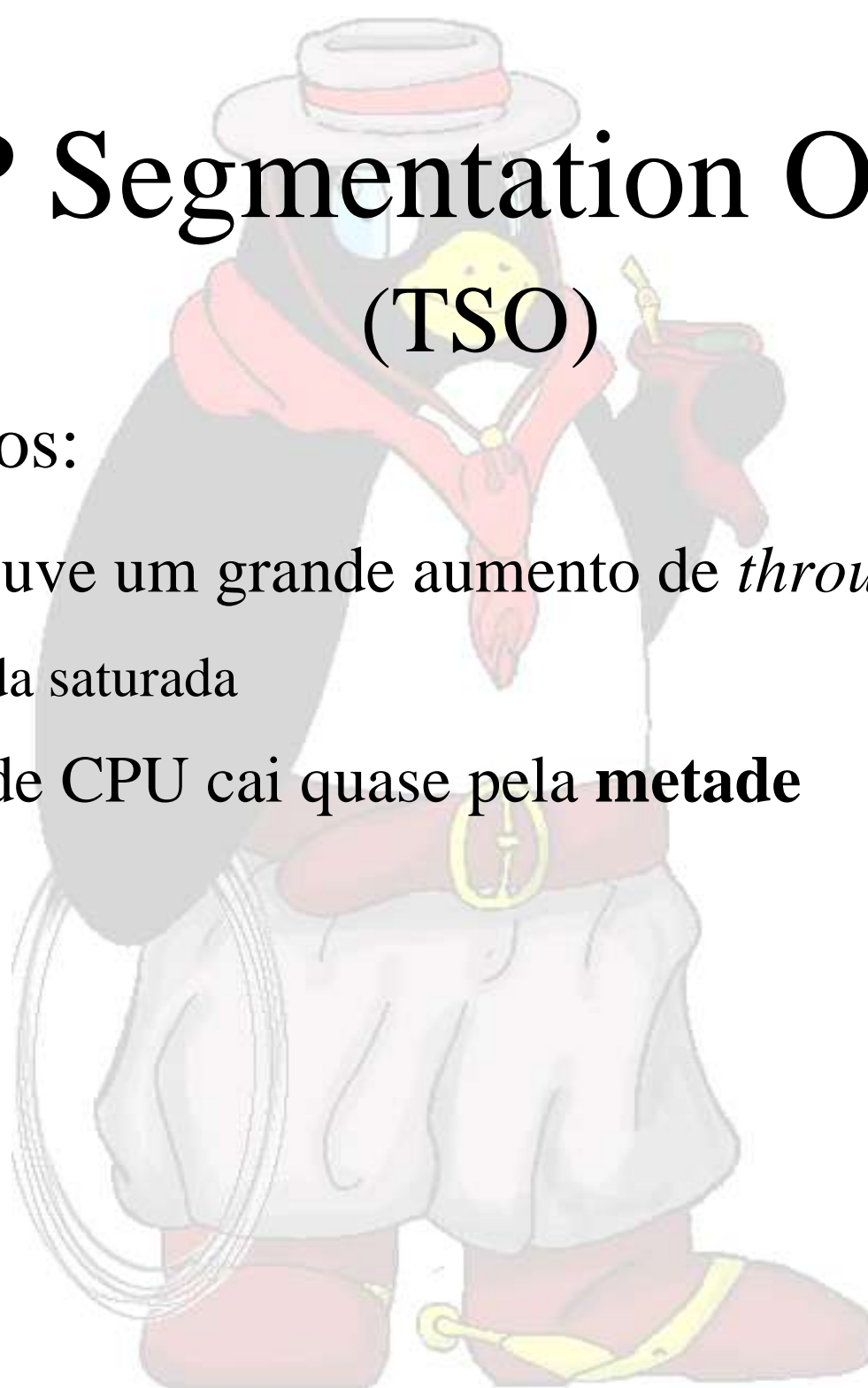
- Offload:
 - Repassa um header como template
 - Repassa um buffer de dados
 - NIC faz segmentação
 - Apenas **uma** grande operação de I/O ao invés de várias pequenas operações
 - Deve possuir suporte no hardware

TCP Segmentation Offload (TSO)

- Resultados pro Intel E1000 (Gigabit Ethernet)
 - Tx/Rx envio de um arquivo (Tx/Rx bidirecional)
 - Sem TSO: 1500Mbps, 82% CPU
 - Com TSO: 1633Mbps, 75% CPU
 - Tx TCP envio de um arquivo (apenas Tx)
 - Sem TSO: 940Mbps, 40% CPU
 - Com TSO: 940Mbps, 19% CPU
- **<http://lwn.net/Articles/9129>** (scott.feldman@intel.org)

TCP Segmentation Offload (TSO)

- Resultados:
 - Não houve um grande aumento de *throughput*
 - Banda saturada
 - O uso de CPU cai quase pela **metade**



Kernel 2.6

Principais culpados



Copyright 2003 LWN.net