

Linux Automation Konferenz 2005 am 31.03.05

Konzepte zur Steigerung der Preemptivität des Linux Kernels 2.6

Arnd Heursch, Witold Jaworski, Romesch Düe
und Helmut Rzehak

Institut für Informationstechnische Systeme (IIS)
Fakultät für Informatik
Universität der Bundeswehr München



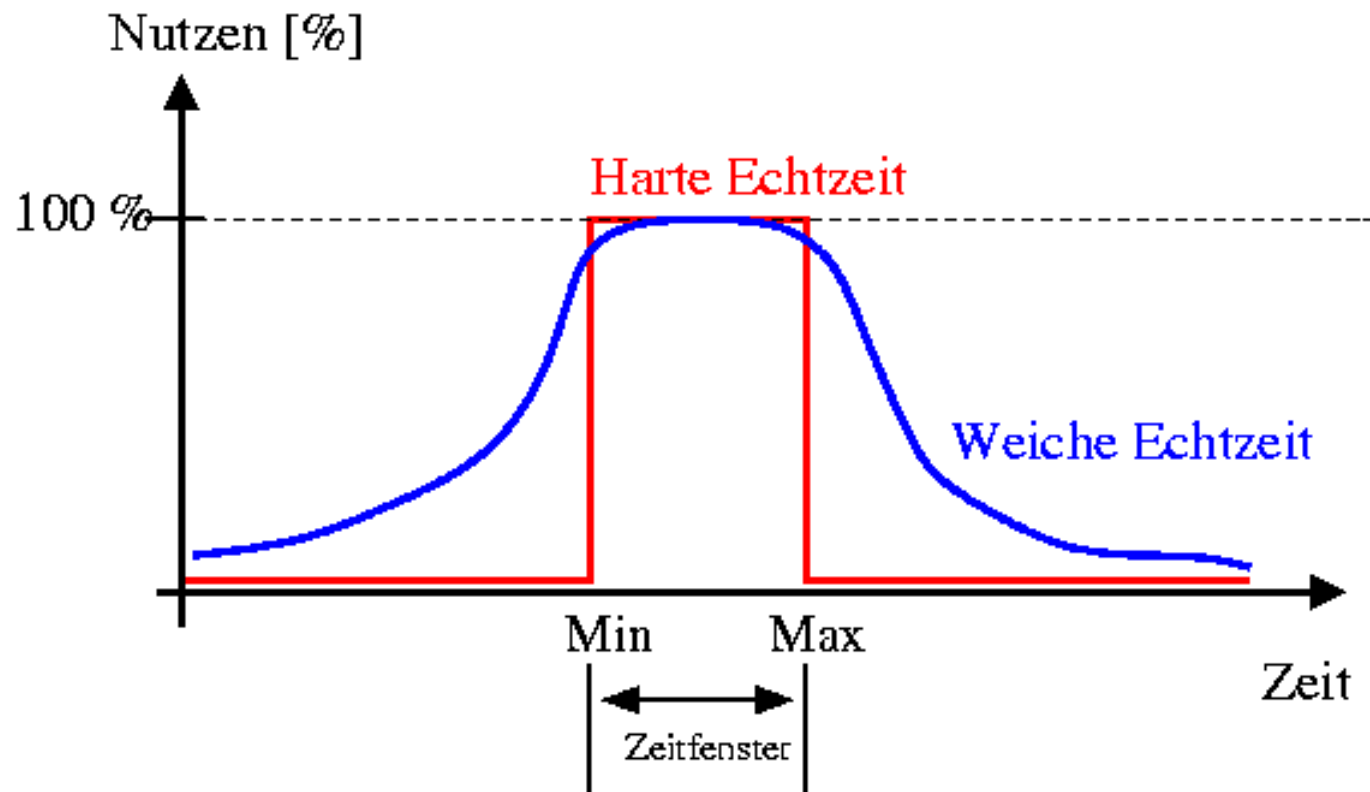
Gliederung

- Definition von Latenzzeiten: IRT und PDLT
- Messungen des Linux 2.6 O(1) Schedulers
- MontaVista Open Source Realtime Linux Project
- Konzept zur Verbesserung der durchschnittlichen Preemptivität des Kernels:
Ersatz von Preemptionsperren & Read/Write Locks durch Mutexe
- Messungen zur IRT des IRQ-Thread Patch
- Zusammenfassung

Nutzenfunktionen

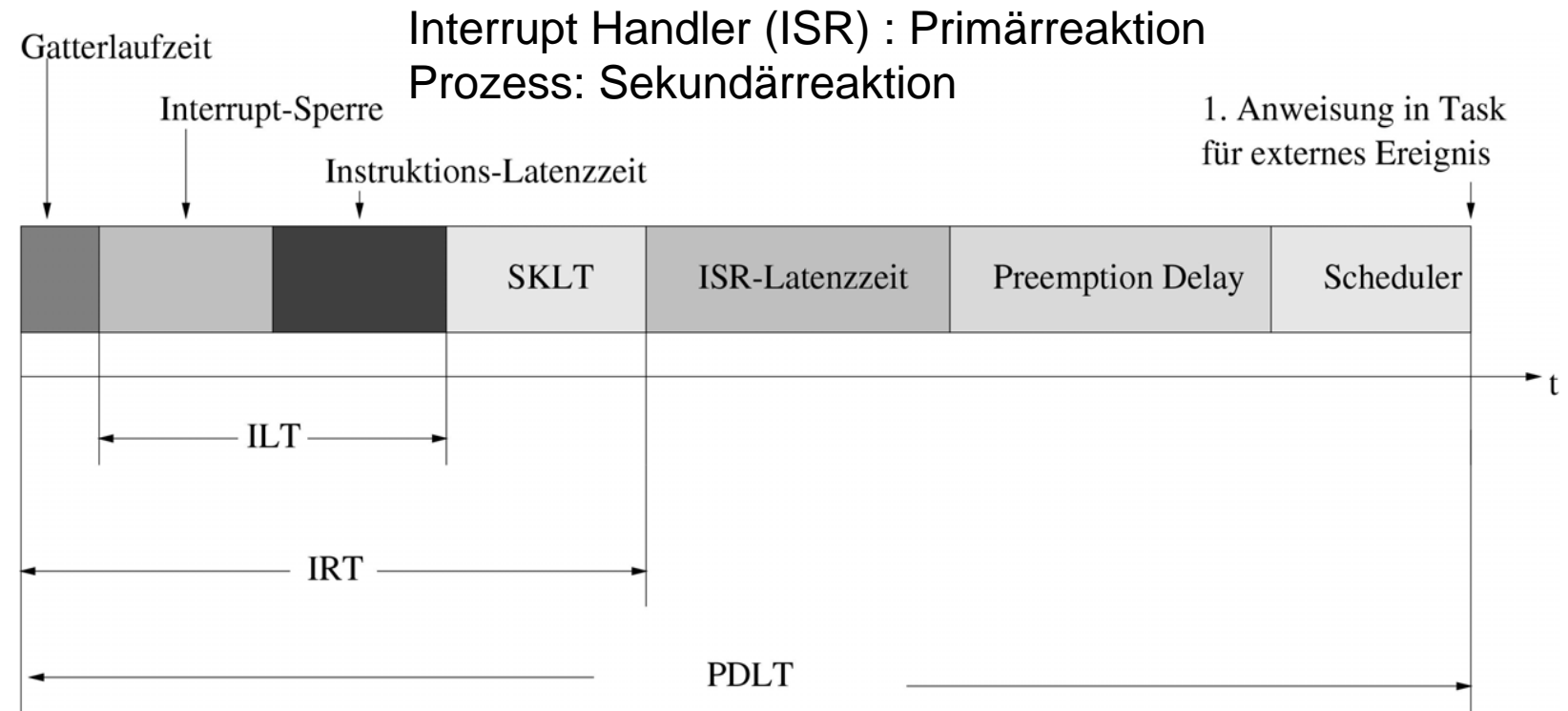
harter & weicher Echtzeitapplikationen

- Harte Echtzeit: Kraftwerke, Fahrzeuge steuern
- Weiche Echtzeit: Audio-, Video-Applikationen



Interrupt Handler (ISR) weckt Prozess

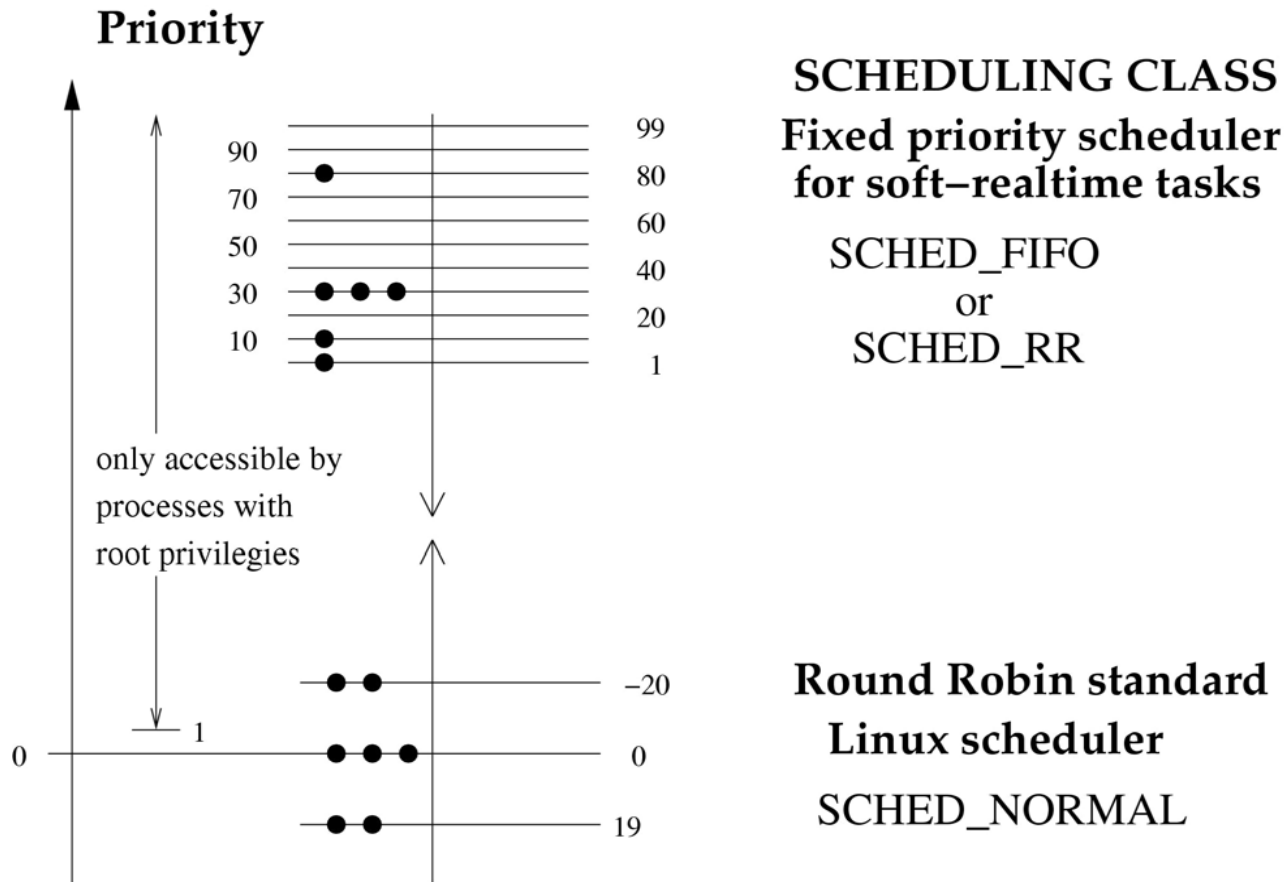
Metriken: IRT & PDLT



$$\text{PDLT}_{\text{soft-realtime}} = \text{IRT} + T_{\text{ISR}} + T_{\text{PREEMPTION-DELAY}} + T_{\text{SCHEDULER}}$$

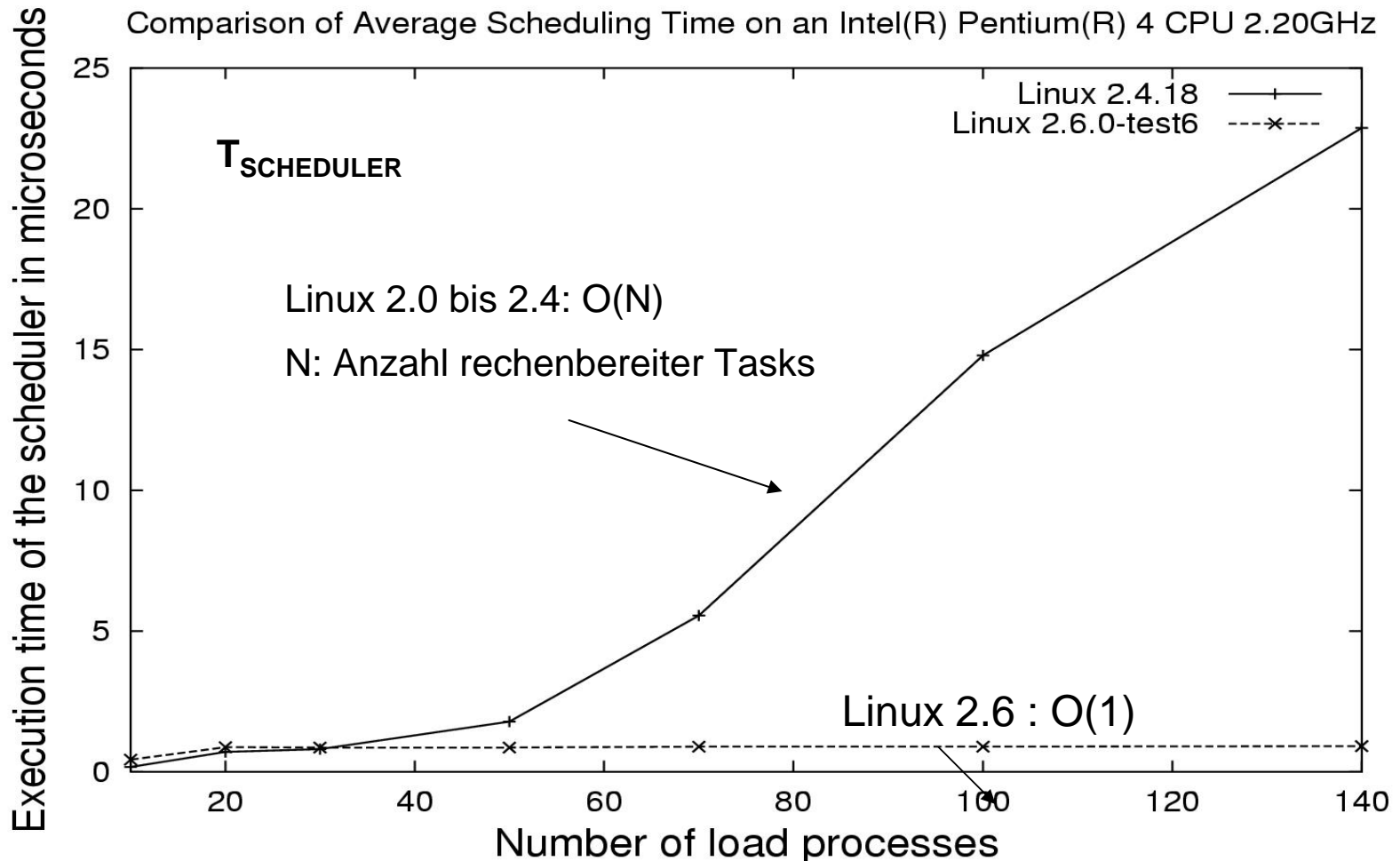


Standard Linux Scheduler



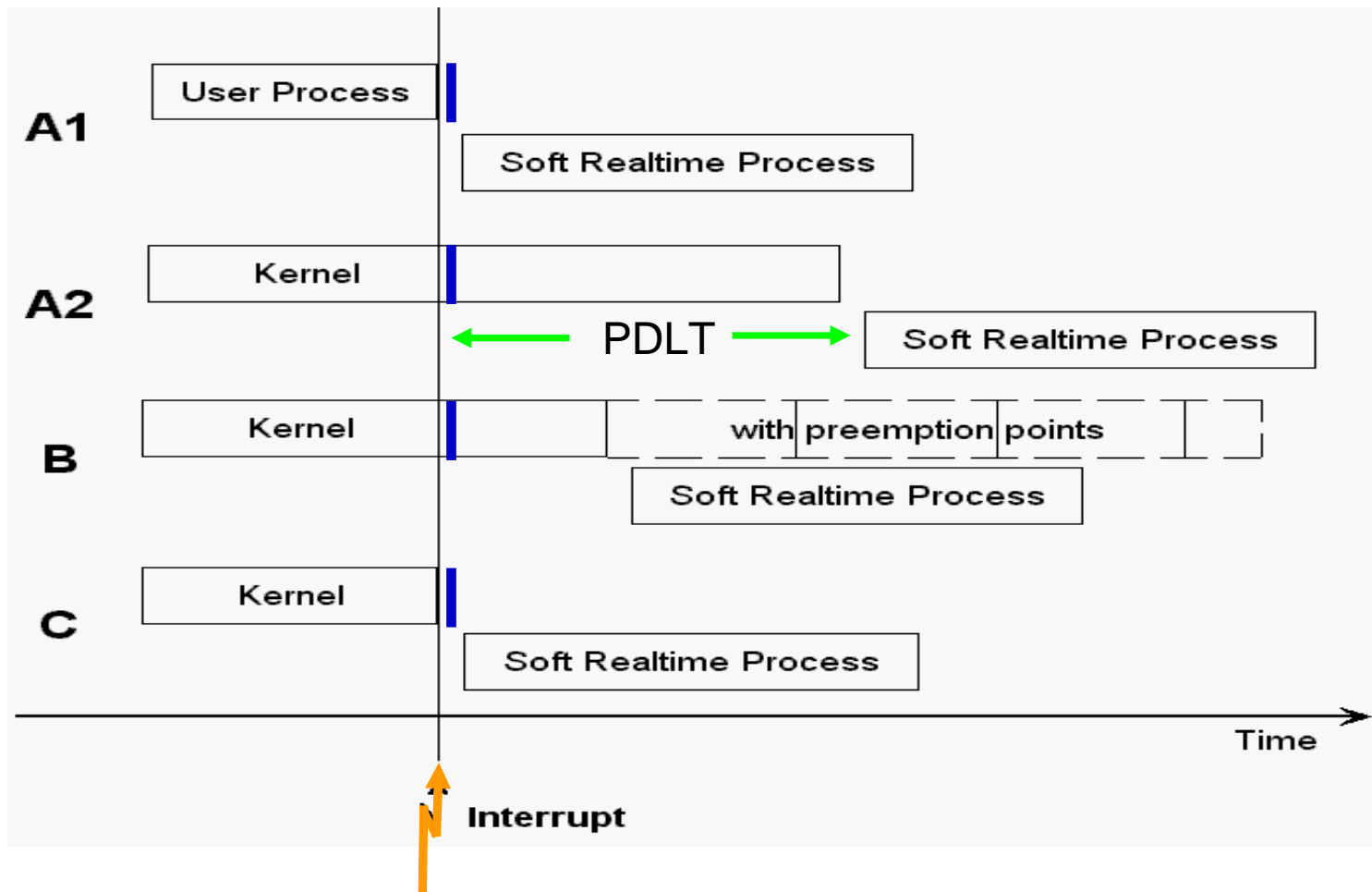
- The circle means a task executing at the priority level where the circle is placed

Scheduler-Laufzeiten: Linux 2.0-2.4 & 2.6

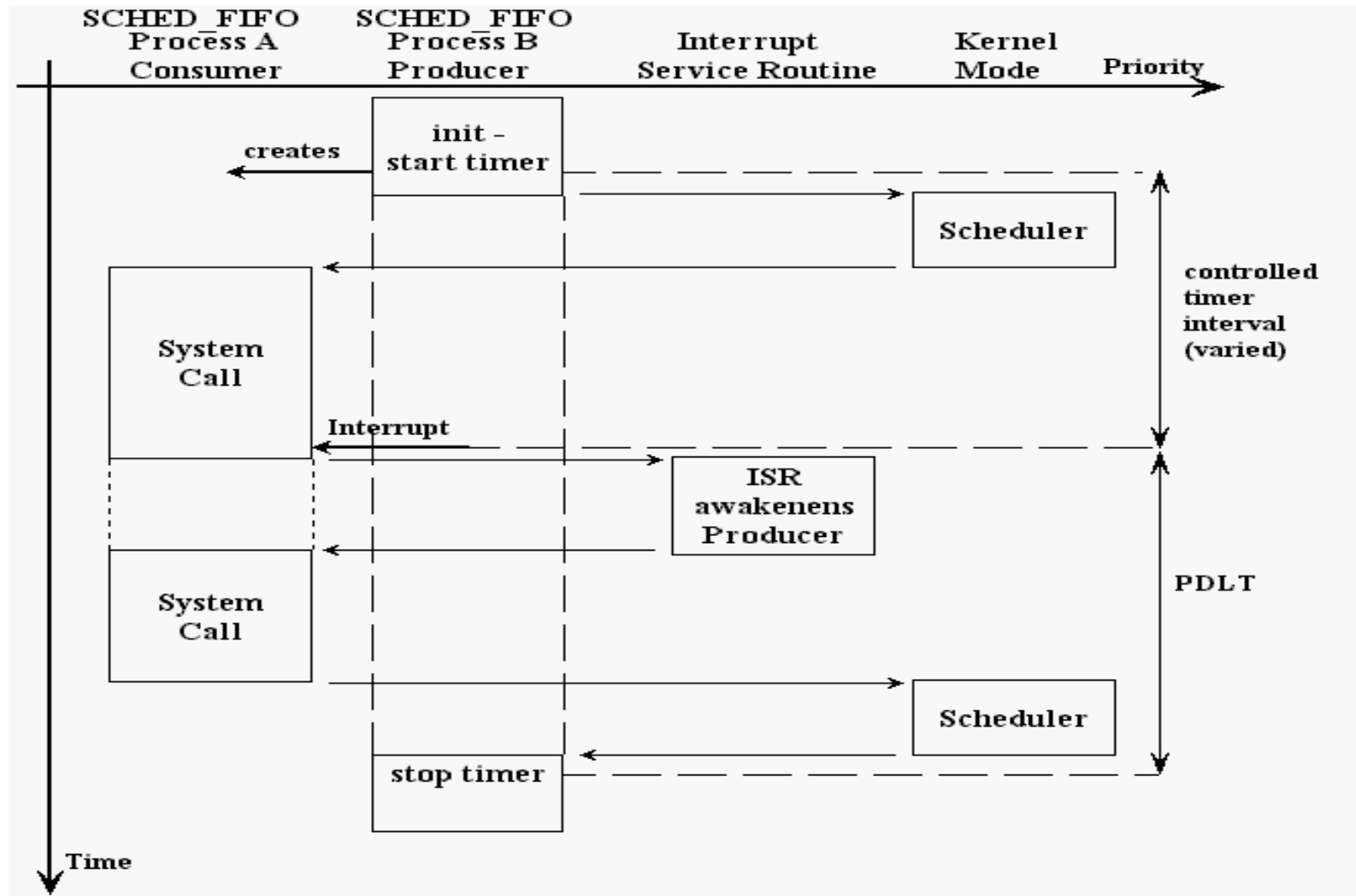


Steigerung der Preemptivität des Linux Kernels

$T_{\text{PREEMPTION-DELAY}}$

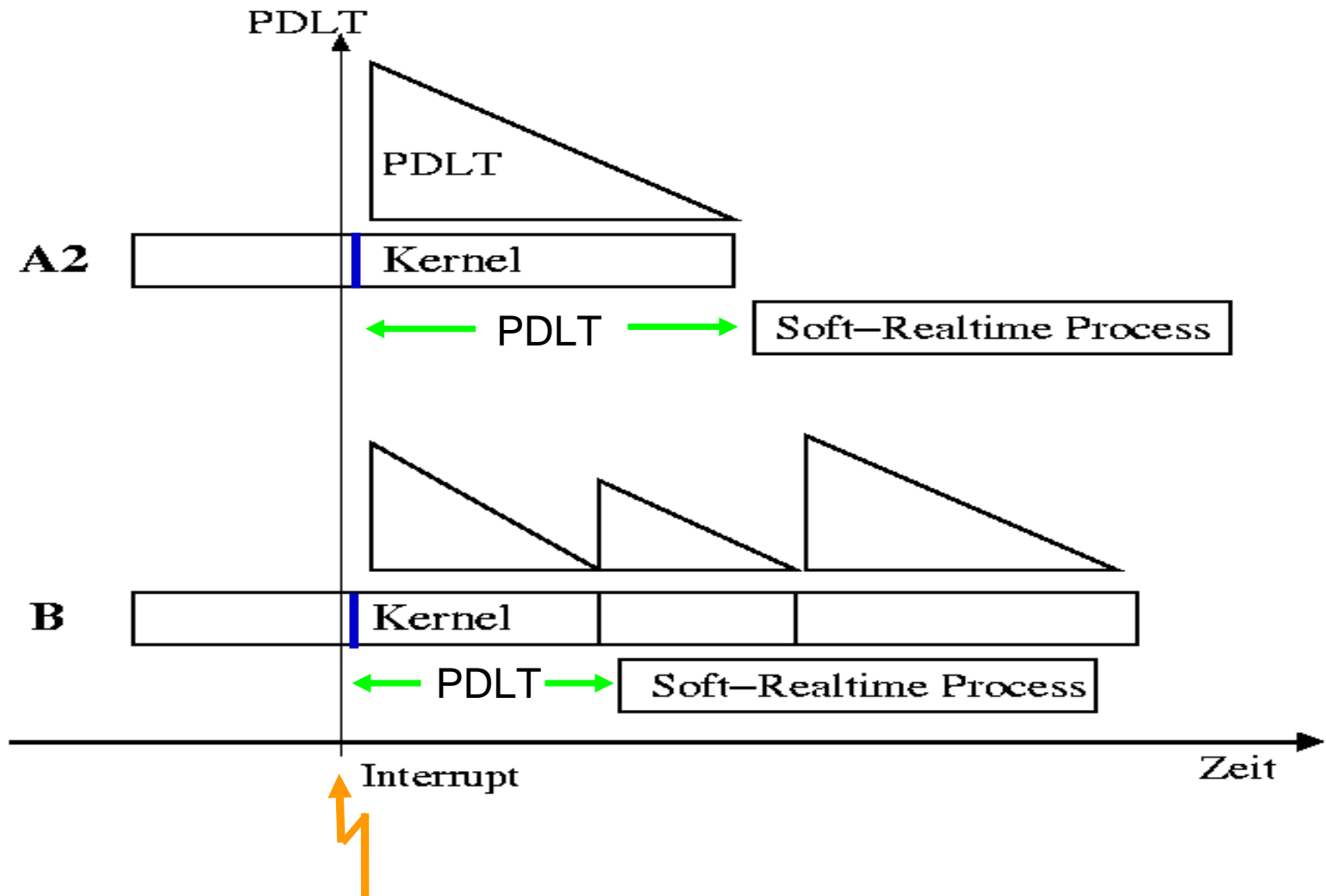


Der Software-Monitor von Prof. M. Mächtel



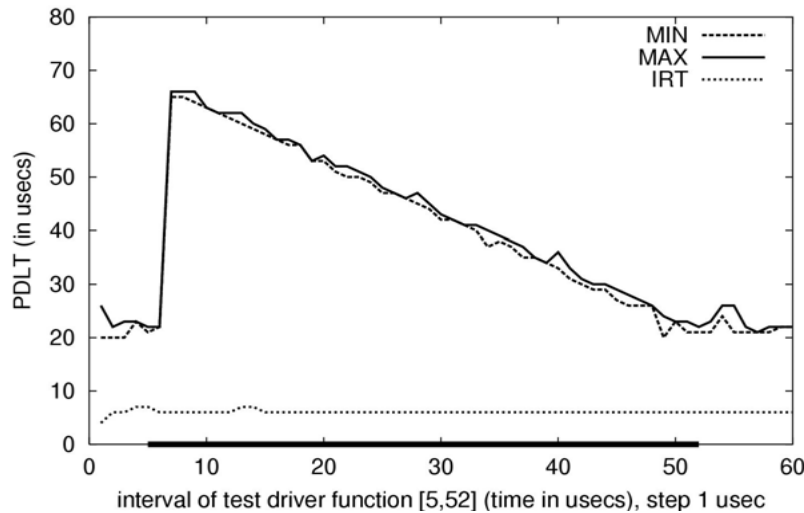
Der Software-Monitor

zur Messung und Visualisierung von Latenzzeiten

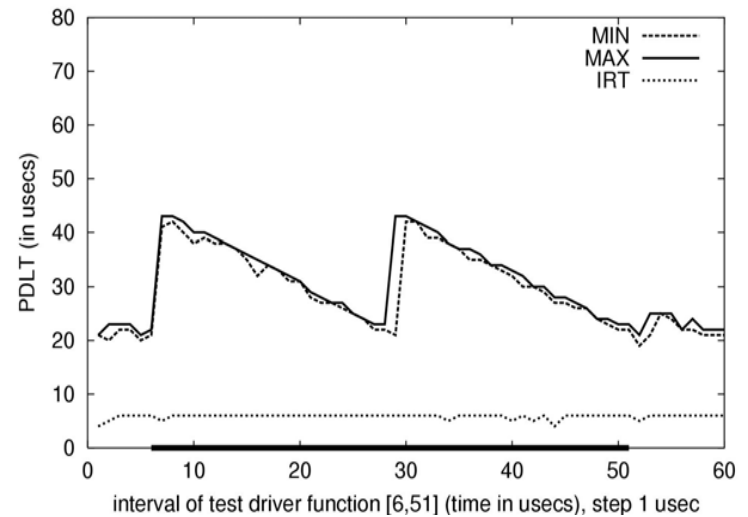


B: Einfügen von Preemption-Points

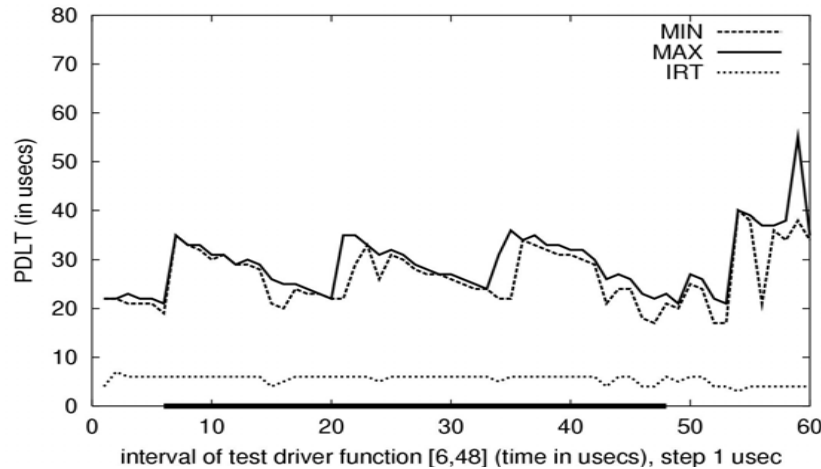
Test driver function read(), no Preemption Points (PP), Kernel 2.4.2



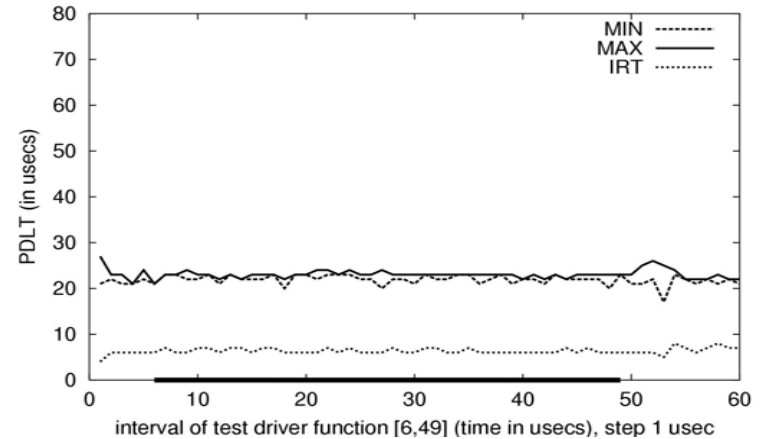
Test driver function read(), with 1 Preemption Point (PP), Kernel 2.4.2



Test driver function read(), with 2 Preemption Points (PP), Kernel 2.4.2

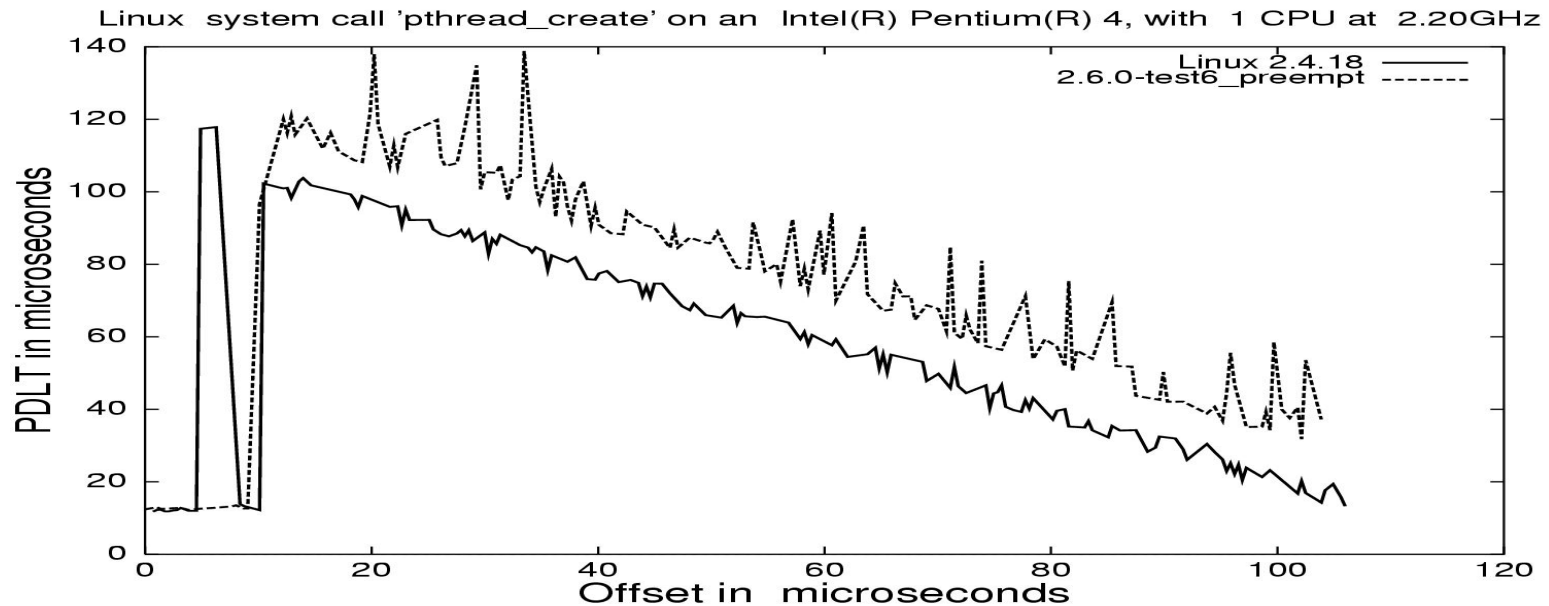


Test driver function read(), 23 Preemption Points (PP), Kernel 2.4.2



Linux 2.6 mit Preemptible Kernel

Schutz kritischer Abschnitte



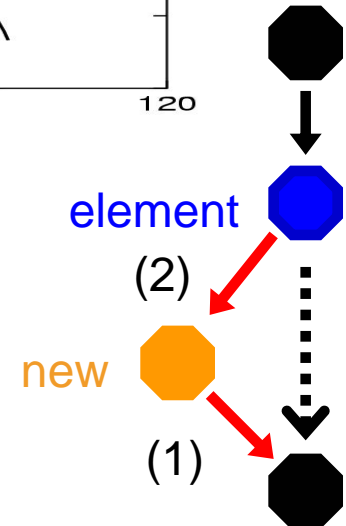
BEGIN_kritischer_Abschnitt:

//UP: Preemptionsperre, SMP Spinlock + IRQ-Lock

(1) `new->next = element->next;`

(2) `element->next = new;`

END_kritischer_Abschnitt



MontaVista Open Source Linux Realtime Project

- **Entwicklerkernel Linux-2.6.9-rc4**

- **+IRQ-Thread Patch** von Andrew Morton

Linux-2.6.9-rc4-mm1

<http://www.kernel.org/pub/linux/kernel/people/akpm/patches/2.6/2.6.9-rc4/2.6.9-rc4-mm1/>

- **+Voluntary Preemption Patch** von Ingo Molnar
realtime-preempt-2.6.9-rc4-mm1-U8.1

<http://people.redhat.com/mingo/realtime-preempt/older/>

- **+MontaVista Patch mit PMutexen der UniBwM:**
603 Preemptionsperren durch Mutexe ersetzt, 30 nicht

Linux-2.6.9_rc4-mm1-U8.1-mv2.patch

<ftp://source.mvista.com/pub/realtime>

Preemptionsperren in Mutexe wandeln

■ Vorteil:

Kernel preemptiver => im Durchschnitt PDLT ↓

■ Nachteile:

I) selten: Mutex belegt => Wartezeit KLT

II) Prioritätsinversion vermeiden

III) Deadlocks vermeiden

IV) In Interrupt-Handlern keine Mutexe möglich

V) Worst Case immer noch:

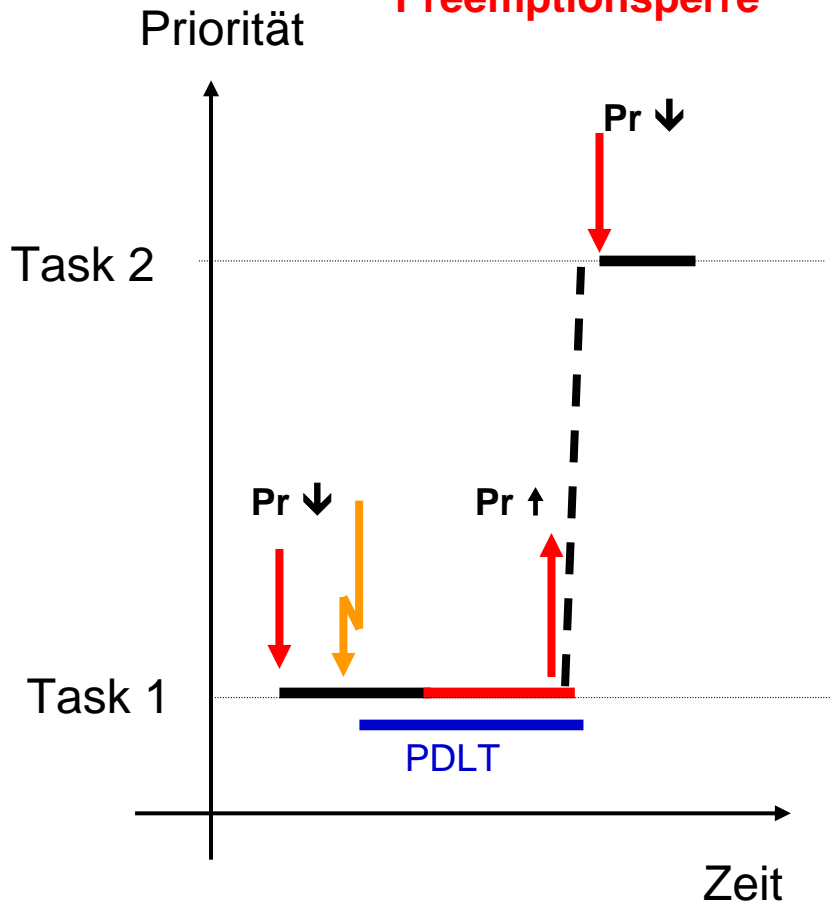
PDLT = Länge des längsten kritischen Abschnittes,
d.h.

Konzept verbessert Soft- aber nicht Hard-Realtime
Fähigkeit von Linux 2.6

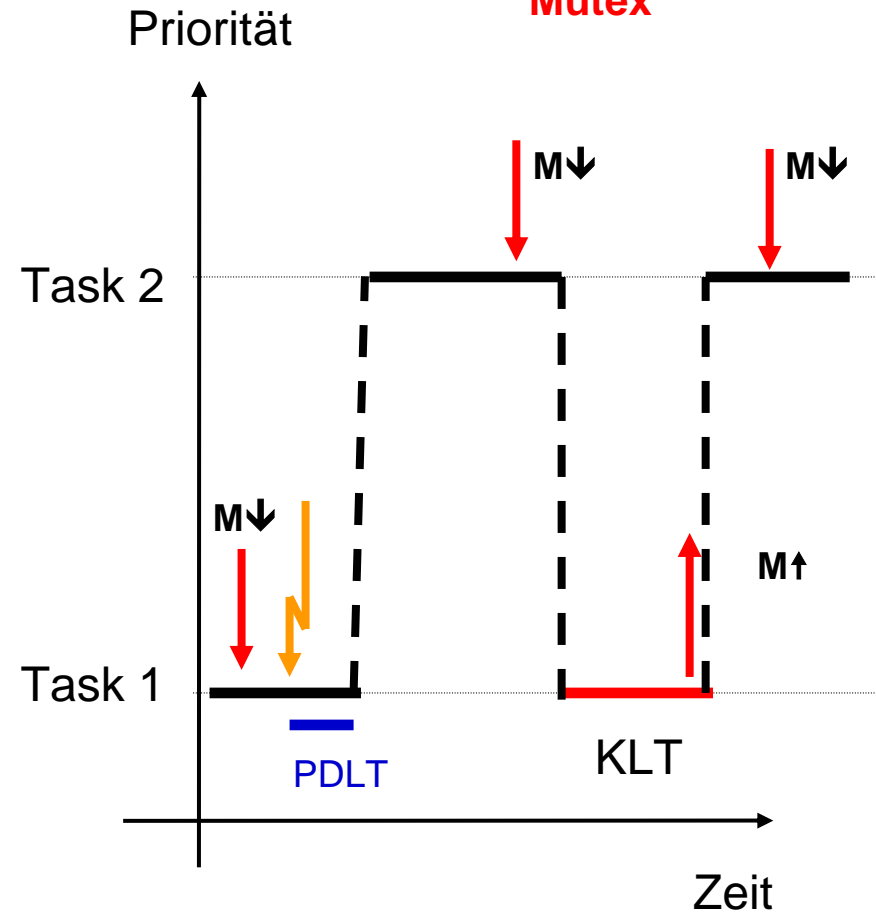
I) Mutexe – Latenzzeiten: PDLT & KLT

Kritischer Bereich geschützt durch

Preemptionsperre

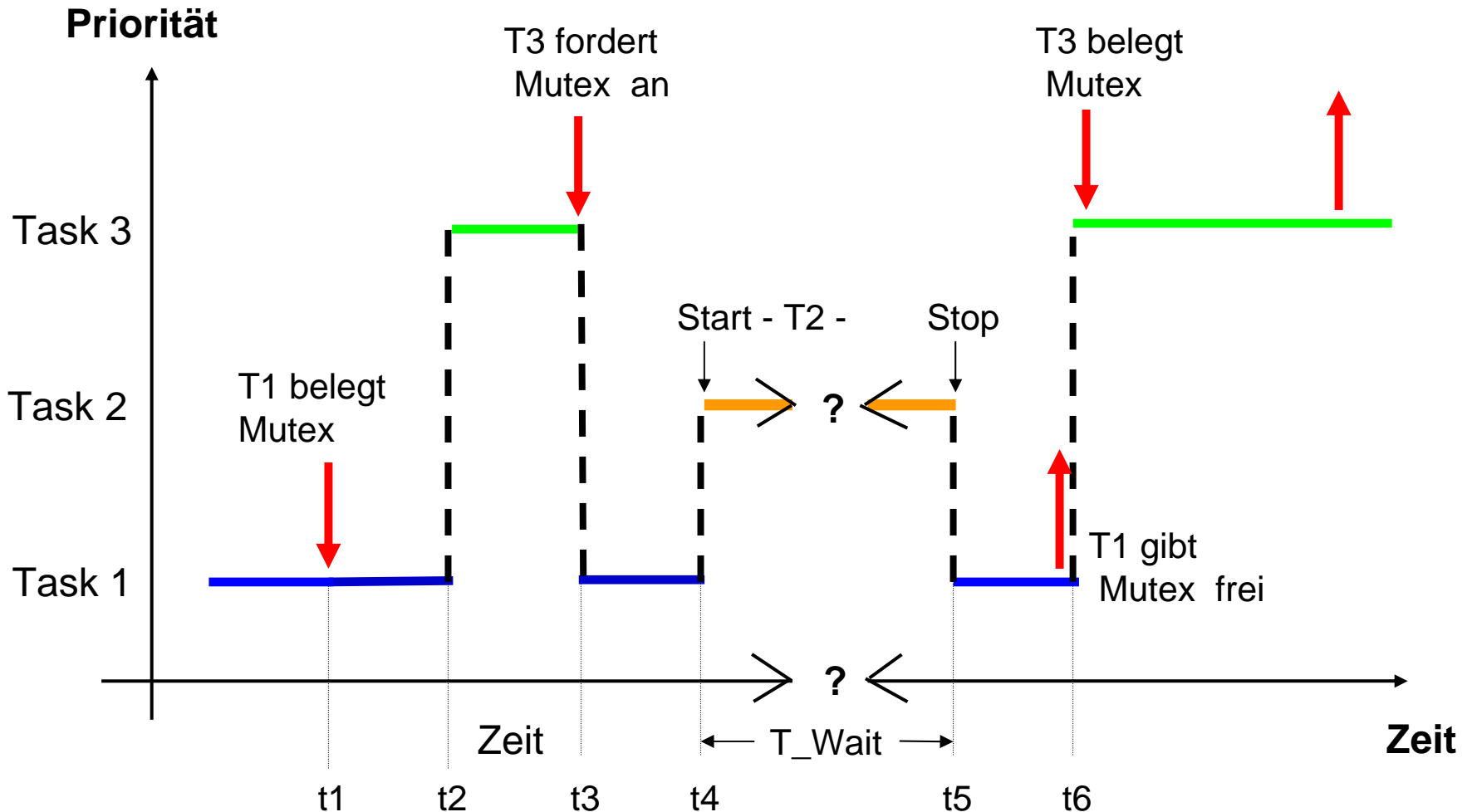


Mutex



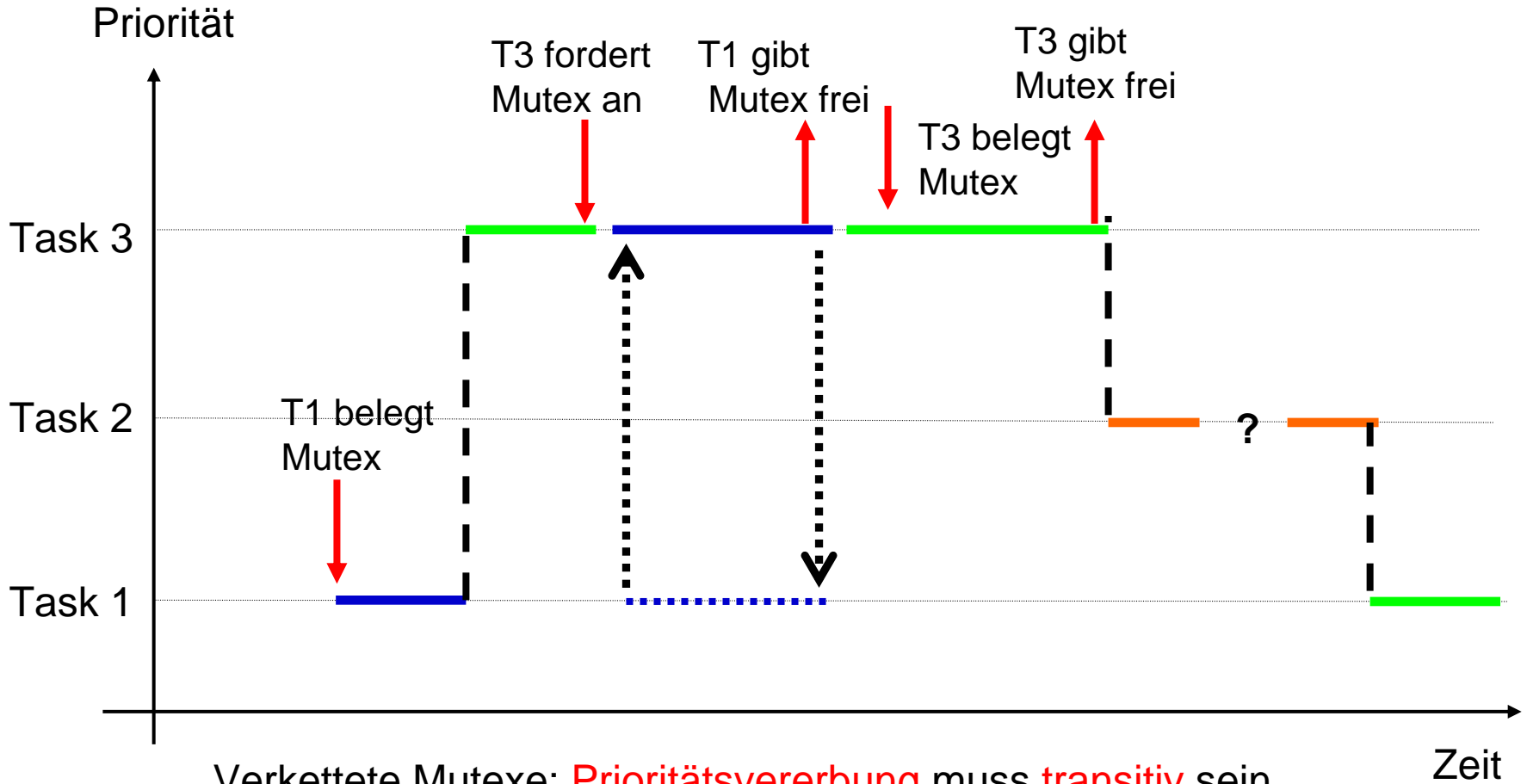
II) Mutexe - Prioritätsinversion

Prozessorbelegung



Lösung: Prioritätsvererbung an Mutexe

Einfaches Prioritäts-Vererbungs-Protokoll (EVP2@UniBwM)



Verkettete Mutexe: **Prioritätsvererbung** muss **transitiv** sein,
Protokoll nach Prof. Victor Yodaiken, Univ. New Mexico

III) Mutexe : Deadlocks vermeiden

- Verklemmungen nur bei geschachtelten Mutexen:
- Task 1:
 - Request Mutex 1**
 - Request Mutex 2**
- Task 2:
 - Request Mutex 2**
 - Request Mutex 1**
- Lösung:
Verklemmungen vermeiden:
Mutexe müssen immer in der gleichen Reihenfolge angefordert werden, d.h. totale bzw. partielle Ordnung auf Mutexen

Ersetzen auch von Read-Write (RW) Locks durch Mutexe, sog. PRW-Locks

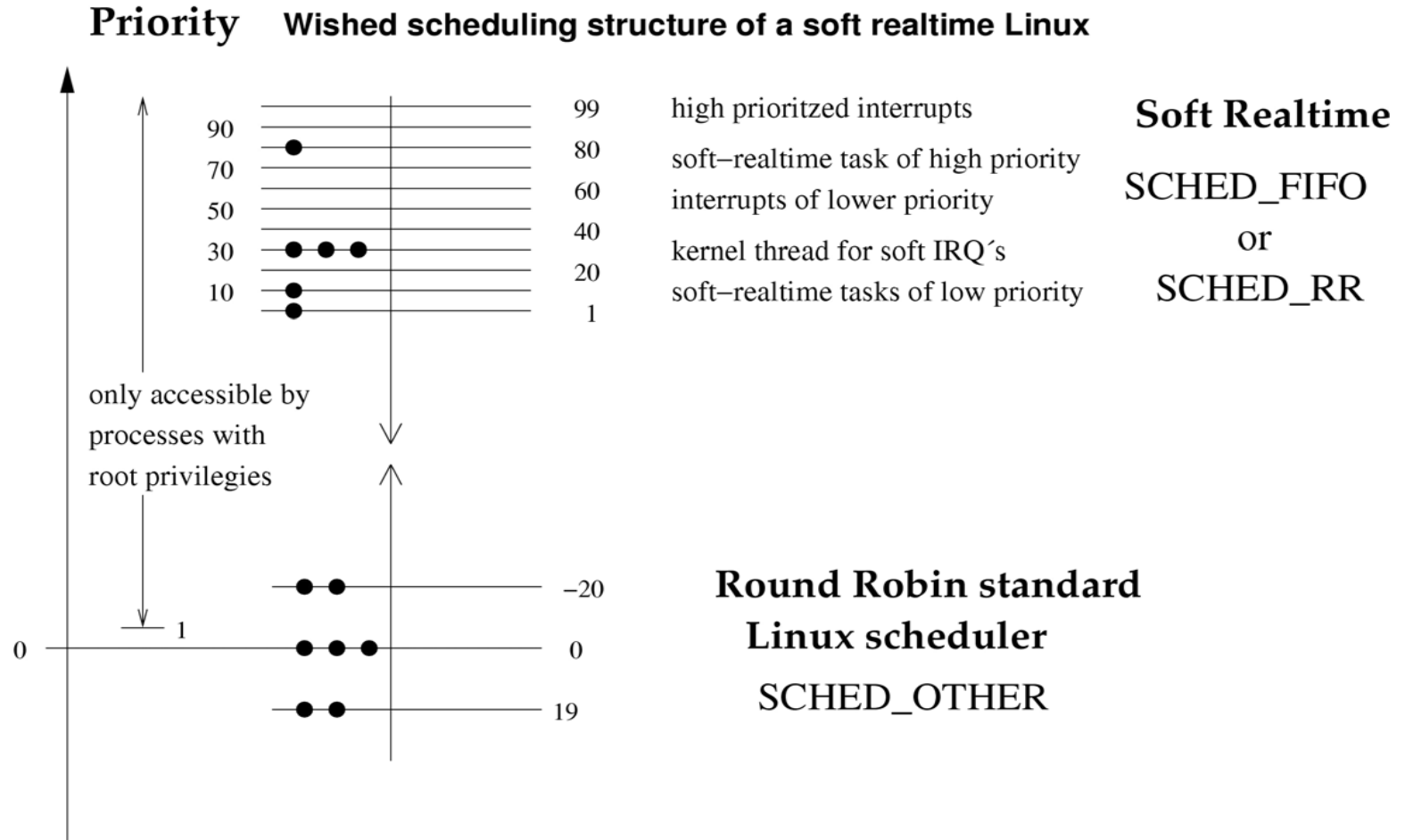
- Read Write Locks (im Linux Kernel nur auf SMP):
 - mehrere Reader-Tasks XOR
 - nur 1 exklusive Writer-Task im kritischen Bereich
 - Umwandlung der RW-Locks in Mutexe:
 - Anzahl gleichzeitige Reader-Tasks auf 4 begrenzen, um Writer nicht zu lange auszusperren
 - Prioritätsvererbung wartender Tasks an das RW-Lock haltenden Task (an den Writer oder an 1 der Reader)
 - Implementierung auch für SingleProzessor-Systeme

Implementierung der PRW-Locks

```
typedef struct prw {
    int mrw_reader;    # Reader
    int mrw_wwriter;  # wartende Writer
    int mrw_status;   Lock lesend/schreibend
    spinlock_t mrw_spin_lock;
    struct list_head mrw_owner;  Besitzer des Locks
    struct list_head mrw_waiter; Wartende Tasks
} prw_t;
typedef struct prw_sleeper_list {
    struct list_head list;
    int prw_type;      lesend/schreibend
    unsigned long initial_prio;  Prioritätsvererbng
    struct task_struct *task;    Pointer auf Task
} prw_sleeper_list_t;
```

IV) Mutexe – in Interrupt-Handlern

Lösung: Kernel Threads führen Interrupt-Handler aus



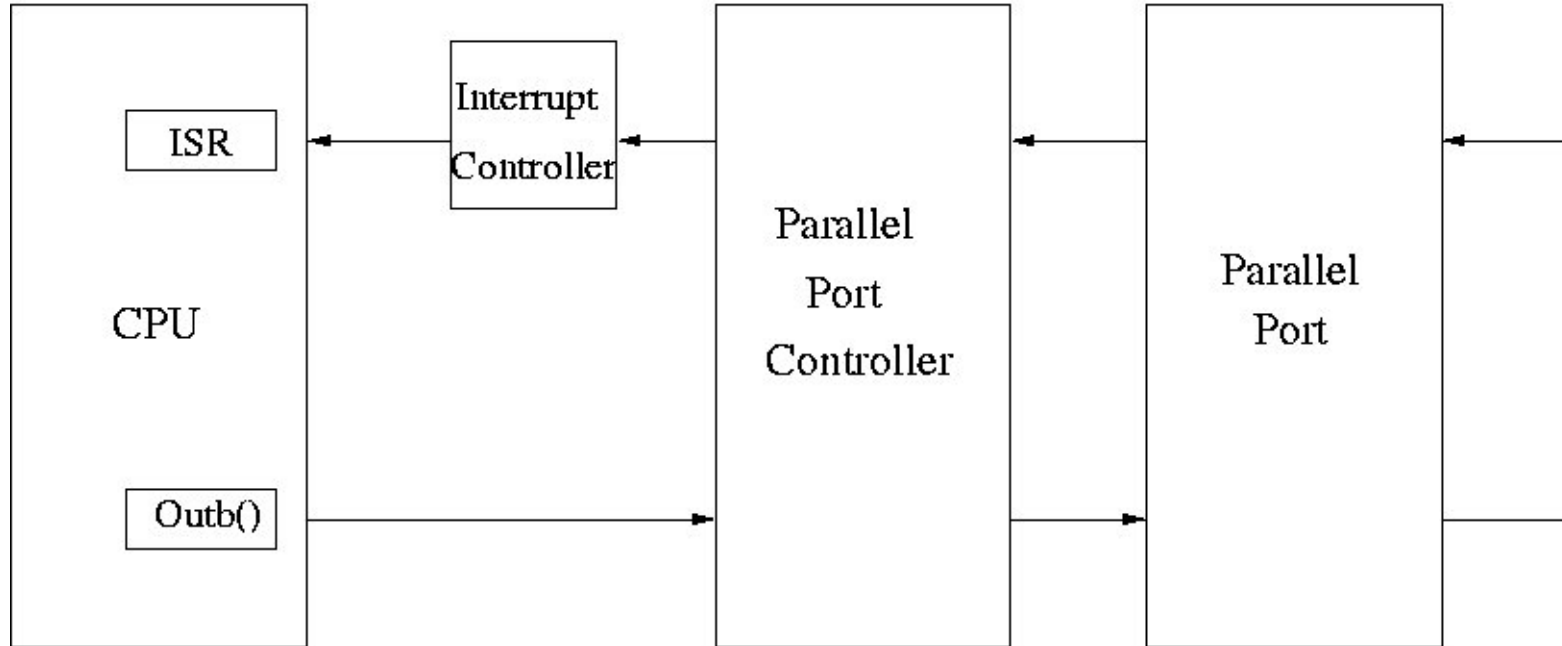
IRQ-Handler durch Kernel Threads

ausgeföhrt: ps ax zeigt Tasks:

PID	TTY	STAT	TIME	COMMAND
1	?	S	0:00	init [5]
2	?	SW<	0:00	[ksoftirqd/0]
3	?	SW<	0:00	[events/0]
4	?	SW<	0:00	[khelper]
9	?	SW<	0:00	[kthread]
20	?	SW<	0:00	[kblockd/0]
664	?	SW<	0:00	[IRQ 12]
679	?	SW<	0:00	[IRQ 6]
658	?	SW	0:00	[kseriod]
713	?	SW<	0:00	[IRQ 14]
715	?	SW<	0:00	[IRQ 15]
746	?	SW<	0:00	[IRQ 9]
750	?	SW<	0:00	[IRQ 11]
....				
4467	?	S	0:00	/opt/kde3/bin/kdm
4475	?	S	0:01	/usr/X11R6/bin/X vt7 -auth

Meßprinzip zur Messung der Interrupt Response Time (IRT) am Parallelen Port

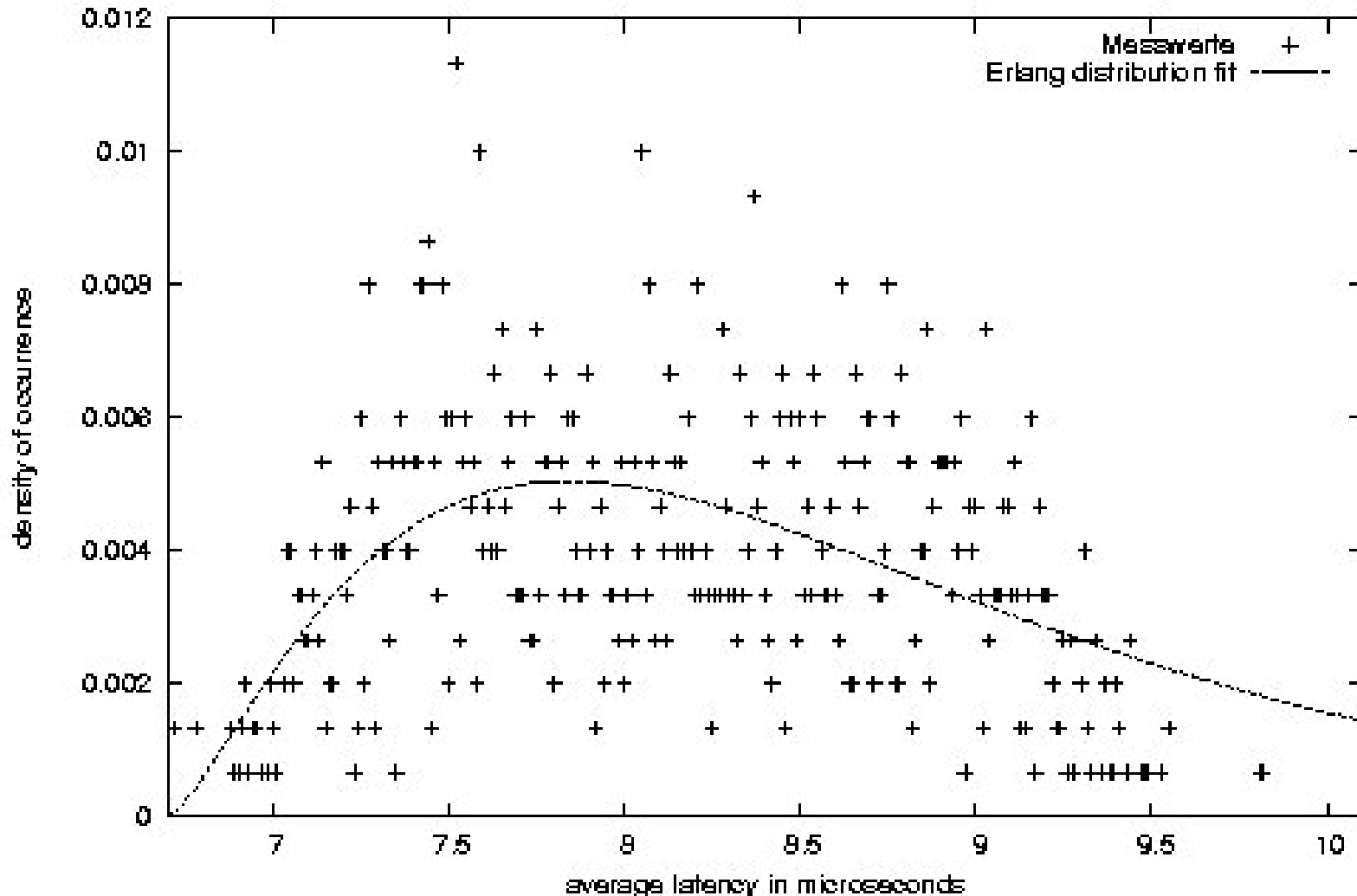
Alle Messungen durchgeführt an einem Pentium 4 mit 2,2 GHz
mit XT-PIC Interrupt-Controller, ohne Verwendung des Local APIC



IRT-Messung Linux 2.6.10 Kernel

6,7 < IRT < 12,6 Mikrosekunden ; IRT-Ø= 8,1 Mikrosekunden

IRT measurements on a Intel(R) Pentium(R) 4 CPU 2.20GHz SMP, using linux release 2.6.10

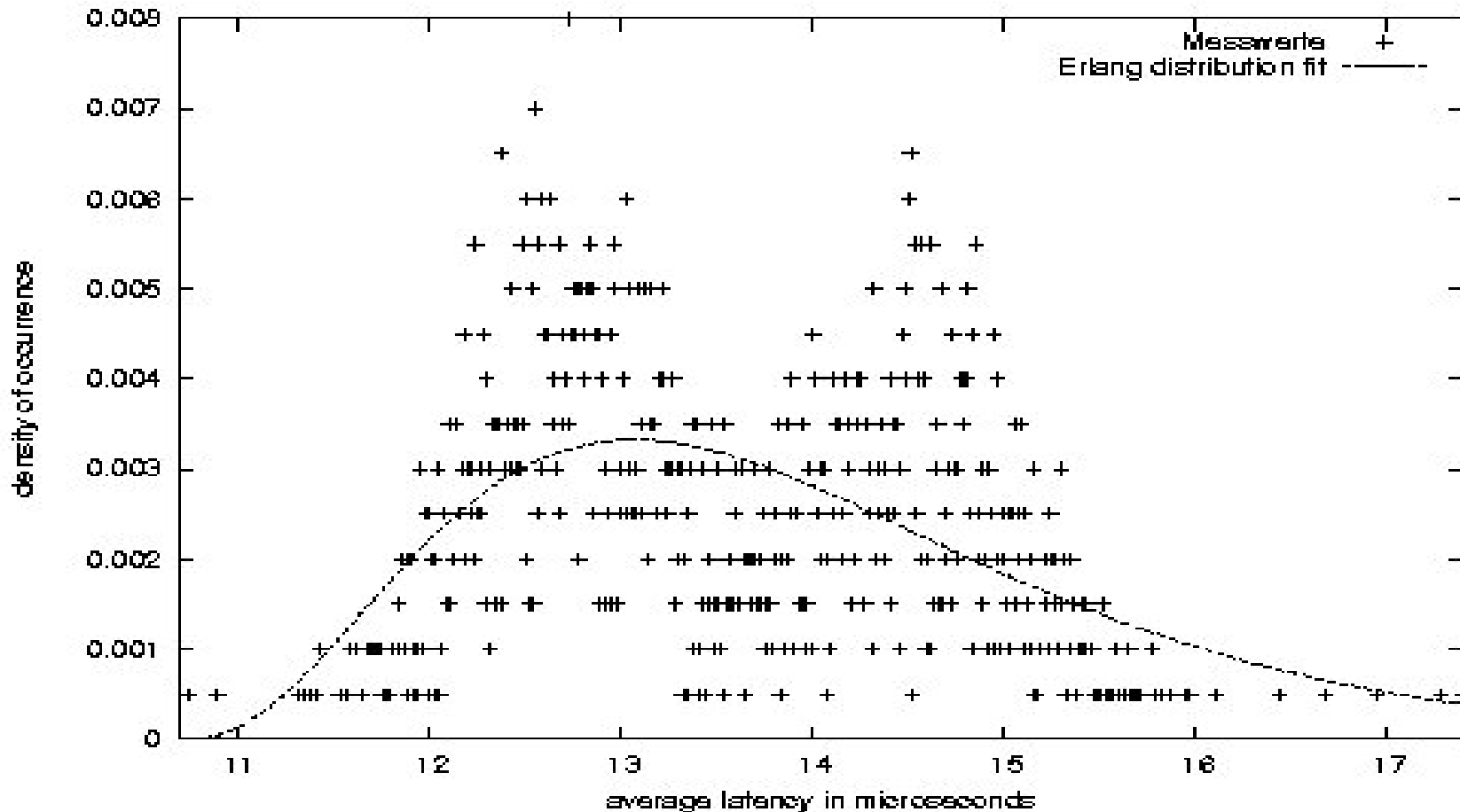


IRT-Messung MontaVista Kernel Linux 2.6.9-rc4-mm1-RT-U8.1

Entwicklerkernel mit IRQ-Threads + PMutex-Patch

10,7 < IRT < 26,8 Mikrosekunden ; IRT-Ø= 13,7 Mikrosekunden

IRT measurements on a Intel(R) Pentium(R) 4 CPU 2.20GHz SMP, using linux 2.6.9-rc4-mm1-RT-U8.1



Preemptivität des Linux Kernels

E
N
T
W
I
C
K
L
U
N
G



Linux 2.0, 2.2, 2.4	Kernel ist nicht preemptiv Zeitliche Auflösung f.Scheduling: 10 ms
Linux 2.6 + Preemptible Kernel	Kernel ist preemptiv, bis auf kritische Abschnitte, die durch Preemptionsperren geschützt sind Zeitliche Auflösung f. Scheduling:1 ms
Kernelvarianten für Linux 2.6: - PMutex@UniBwM Kooperation mit MontaVista Inc. - TimeSys(kommerziell) - RobustMutexes@Intel OSDL,Telekommunikation	Kernel ist preemptiv & kritische Abschnitte sind durch Mutexe geschützt Implementierung PMutex@UniBwM publiziert im Mai 2003 für Linux 2.4 & in Kooperation mit MontaVista Inc. im Open Source Real Time Linux Project am 8.10.2004 für Linux 2.6

Zusammenfassung

- MontaVista Open Source Realtime Linux Project für Linux 2.6:
Ersetzen von Preemptionsperren durch Mutexe &
Ersatz von Read/Write-Locks durch Mutexe/PRW-Locks
 - erhöht die durchschnittliche Preemptivität des Linux Kernels für Soft-Realtime Aufgaben, nicht jedoch die Hard-RT-Fähigkeiten

Ausführung der Interrupt-Handler durch Kernel Threads
im MontaVista Open Source Realtime Project für Linux 2.6

- erlaubt Priorisierung der Interrupthandler
- ermöglicht Umwandlung von Preemptionsperren in Mutexe auch in Interrupthandlern
- erhöht die durchschnittliche Interrupt Response Time um 5,6 Mikrosekunden (davon Schedulerlaufzeit > 1 Mikrosekunde)
- Seltene IRT von 30 Mikrosekunden und mehr möglich, da bei diesem Konzept Interrupt-Handler verbliebene Preemptionsperren abwarten müssen

Danke für Ihre Aufmerksamkeit

- **Kernel Patches unter GPL Lizenz Version 2:**

[http:// iis.unibw-muenchen.de/research/linux/mutex.html](http://iis.unibw-muenchen.de/research/linux/mutex.html)

- PMutexe mit Prioritätsvererbung
- Read/Write Locks mit Prioritätsvererbung

- Kooperation mit MontaVista Inc. im **Open Source Real-Time Linux Project:**

http://source.mvista.com/linux_2_6_RT.html