Supporting Time-sensitive Applications on a Commodity OS

Ashvin Goel, Luca Abeni, Charles Krasic, Jim Snow, Jonathan Walpole
Department of Computer Science and Engineering
Oregon Graduate Institute, Portland

ACM SIGOPS Operating Systems Review - OSDI '02: Proceedings of the 5th symposium on Operating systems design and implementation

(Presented by Youngho Choi)

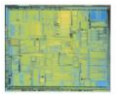
Agenda

- Background
- Introduction
  - Related Work
- Time-Sensitive Requirements
- Implementing Time-Sensitive Linux
  - Firm Timers
  - Fine-Grained Kernel Preemptibility
  - CPU Scheduling
- Evaluation
- Conclusions
- Technical Trends

Background - Evaluation environments

1994 1996 1998 1999 2000 **2002** 2003 2004 2005 2008

Intel®
Pentium® II
300 MHz



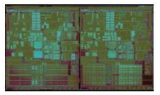
Intel®
Pentium® III
500 MHz

Intel®
Pentium® 4
1.5 GHz
(with 512MB
memory)

ARM
ARM 926
100MHz

Paper submitted

ARM
ARM 926
200MHz



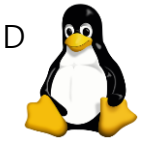
Intel®
Pentium® D
3.2 GHz

Linux-1.2.0

Linux-2.2.0

Linux-2.4.16
Epoch Scheduler
(a.k.a. O(n) scheduler)

Linux-2.6.8.1
O(1) Scheduler



Linux-2.6.23
CFS Scheduler

Source: - https://news.samsung.com
- https://www.hotchips.org
- http://download.intel.com/pressroom/kits/IntelProcessorHistory.pdf
- https://en.wikipedia.org/wiki/Linux_kernel

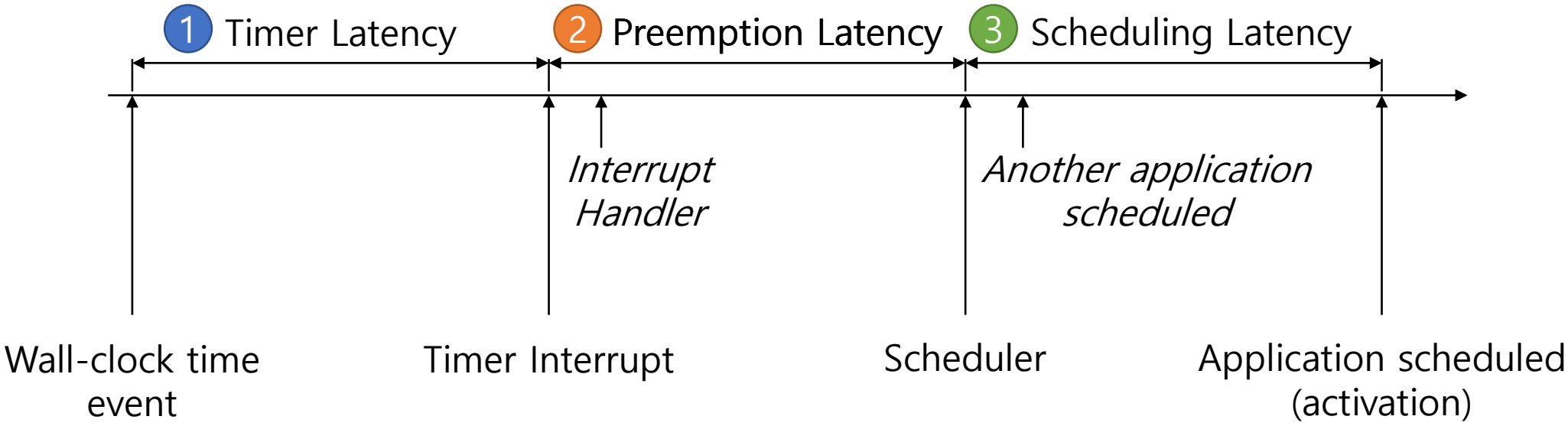
Introduction

- Time-sensitive applications
  - Require low-latency response from the kernel and from other system-level services (e.g. multimedia applications)
- Current commodity systems approaches
  - *Coarse-gained resource allocation* - require more precise allocation
  - *Improve the timing response* - Do not evaluate performance of non-real time applications
- Suggested Solutions
  - Firm timers
  - Fine-grained kernel preemptibility
  - Priority and reservation-based CPU scheduling

Introduction - Related Work

- Real-time community [10, 19]
  - Ignore practical system issues such as kernel non-preemptibility and interrupt processing overhead
- Linux/RK [17], RED Linux [22], MontaVista Linux [12]
  - Performance overhead on throughput-oriented application is not clear
- SMaRT [16]
  - Implemented on SVR4 Unix and Solaris
- Linux-SRT [6]
  - Do not incorporate kernel preemption
  - Do not discuss the issue of time-sensitive performance applications
- RTLinux [4]
  - Do not provide real-time performance to Linux applications
- RT-Mach [18]
  - The overhead of high resolution timers can affect performance of throughput-oriented applications
- Nemesis operating system [9]
  - Structure and API is very diffident from the standard programming environment of operating systems

Time-Sensitive Requirements



1 *Timer Latency* - Need an accurate timing mechanism

2 *Preemption Latency* - Need a responsive kernel

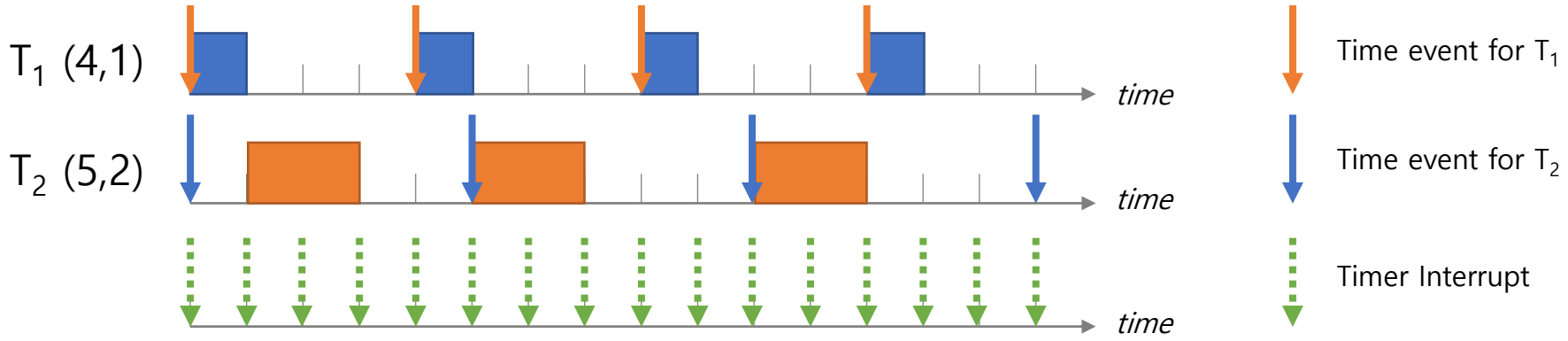
3 *Scheduling Latency* - Need an appropriate CPU scheduling

Implementing Time-Sensitive Linux

- Timing Mechanism
  - Firm timers = One-shot timers + Soft timers
- Responsive Kernel
  - Reduce non-preemptible sections
- CPU Scheduling Algorithm
  - Proportion-period scheduler + Priority-based scheduler

Timing Mechanism (1/4)

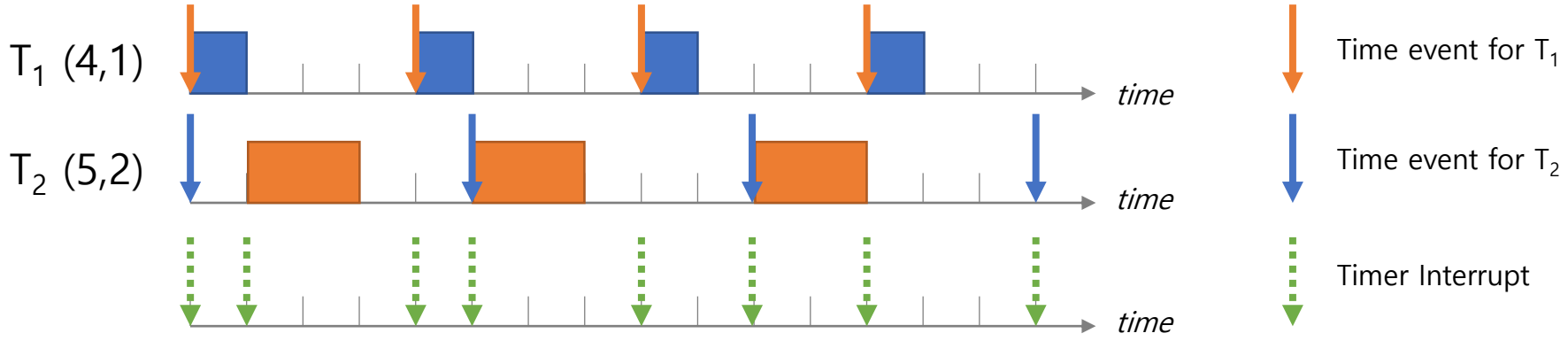
- *Periodic Timers*
  - Timer interrupts repeat regularly
    - e.g. Period of interrupts = Maximum timer latency = 10ms
  - Problem: reducing period of timer interrupt increases system overhead



Solution: Move from a periodic timer interrupt model to a one-shot timer interrupt model

Timing Mechanism (2/4)

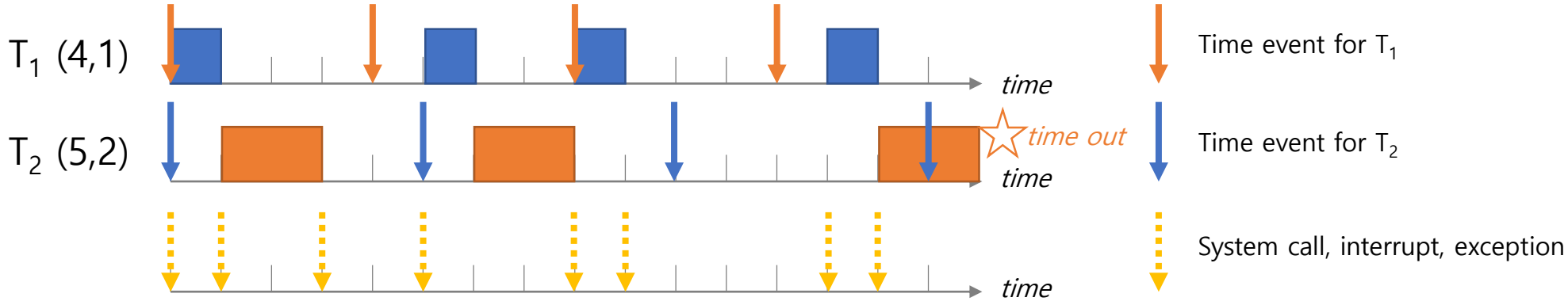
- *One-shot Timers*
  - Interrupts are generated only when needed
  - Problem: Expensive timer reprogramming cost
  - Problem: cache pollution



Solution: Uses the APIC and Soft timers

Timing Mechanism (3/4)

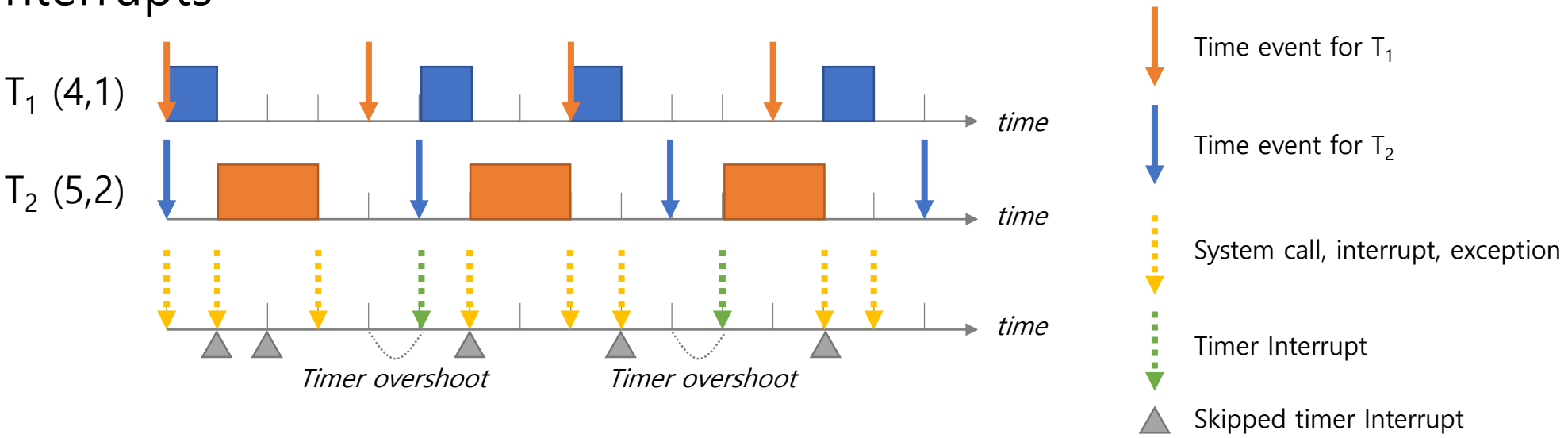
- *Soft Timers*
  - Poll for expired timers at strategic points
    - E.g. system call, interrupt, and exception return paths
  - Problem: Introduce timer latency



Solution: combining one-shot timers with soft timers

Timing Mechanism (4/4)

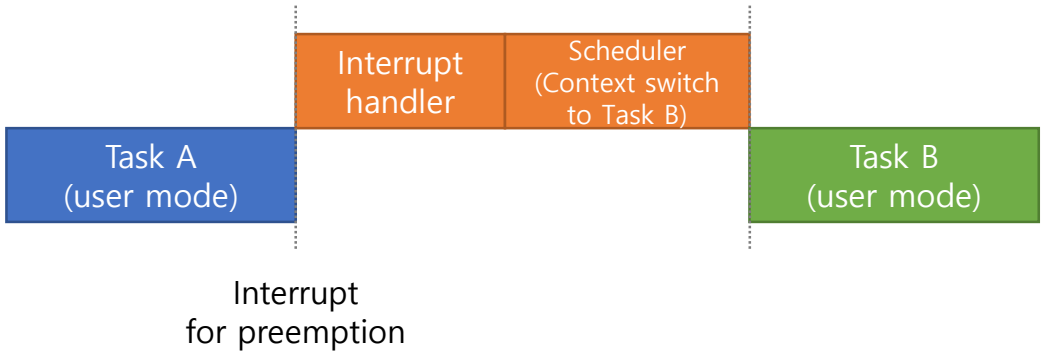
- *Firm Timers*
  - Combination of one shot timer and soft timer
  - Use APIC one shot timers in Intel Pentium
  - *Timer overshoot* : fire an overshoot amount of time after the next timer expiry to overcome overhead associated with fielding interrupts



Kernel Preemptibility (1/3)

- Why?
  - Interrupt might be disabled
  - Non-preemptible critical section (e.g. 30ms)

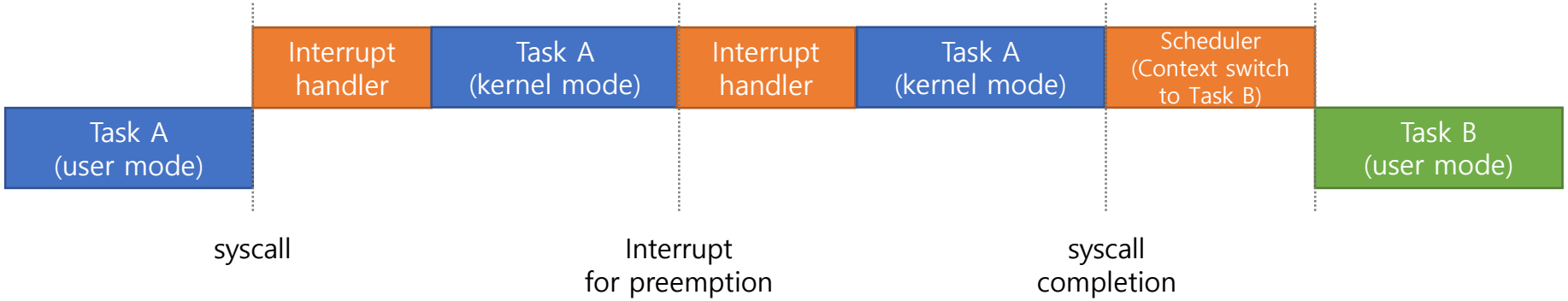
  - Three Preemption Types
    A. user preemption in user mode



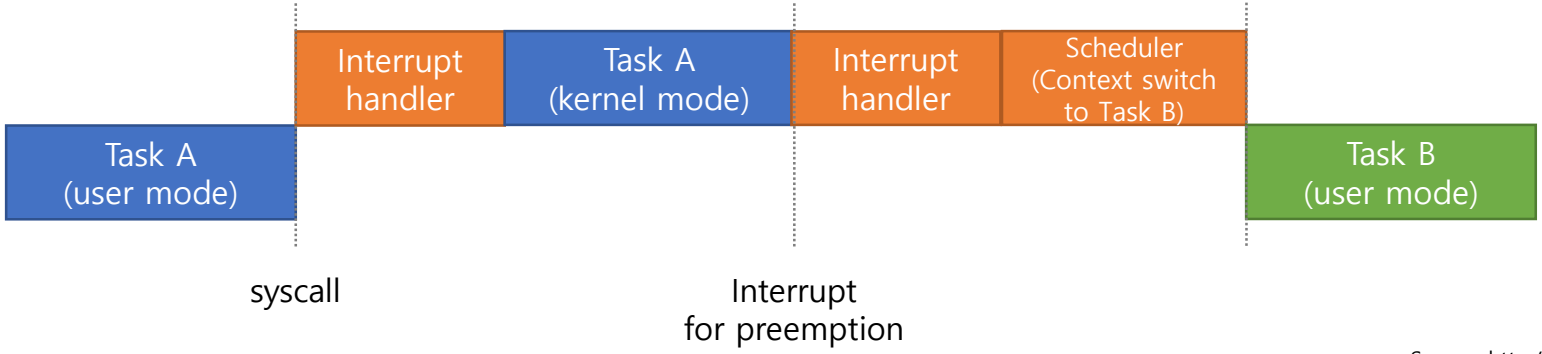
Source: http://jake.dothome.co.kr/preemption/

Kernel Preemptibility (2/3)

B. User preemption in kernel mode



C. Kernel preemption in kernel mode



Source: http://jake.dothome.co.kr/preemption/

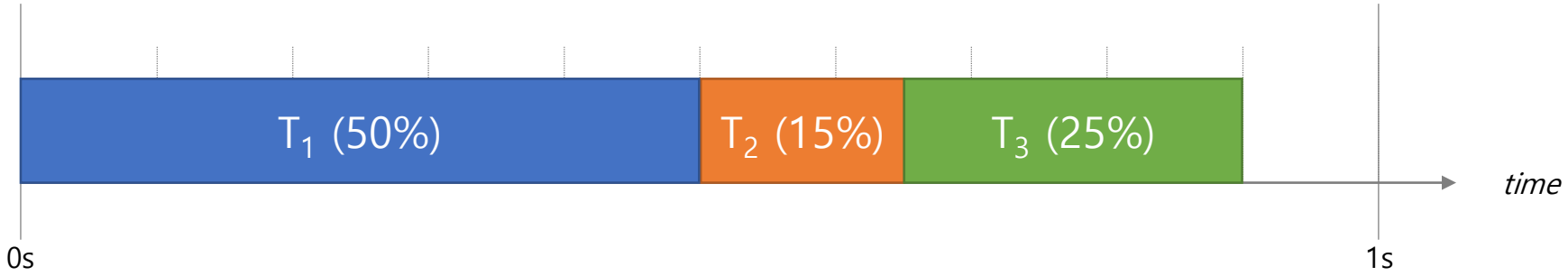
Kernel Preemptibility (3/3)

- Solution
  - *Explicit preemption*
    - Explicit insertion of preemption points at strategic points inside the kernel
    - Problem: Manually placed on system call paths
  - *Preemptible kernel*
    - Allow preemption anytime the kernel is not accessing shared data structures
    - Problem: Have high preemption latency when spinlocks are held for a long time
  - *Robert Love's lock-breaking preemptible kernel patch*
    - Combine explicit preemption with the preemptible kernel approach
    - Release (and reacquiring) spin-locks at strategic points in long code sections

CPU Scheduling (1/3)

- *Proportion-Period CPU Scheduling*
  - Each task is allocated a fixed proportion of the CPU at each task period
  - automatically provides temporal protection
  - proportion period is decided by "feedback mechanism"

e.g. a total of T = 100 shares, T_1 is assigned 50 shares,
T2 is assigned 15 shares and T3 is assigned 35 shared



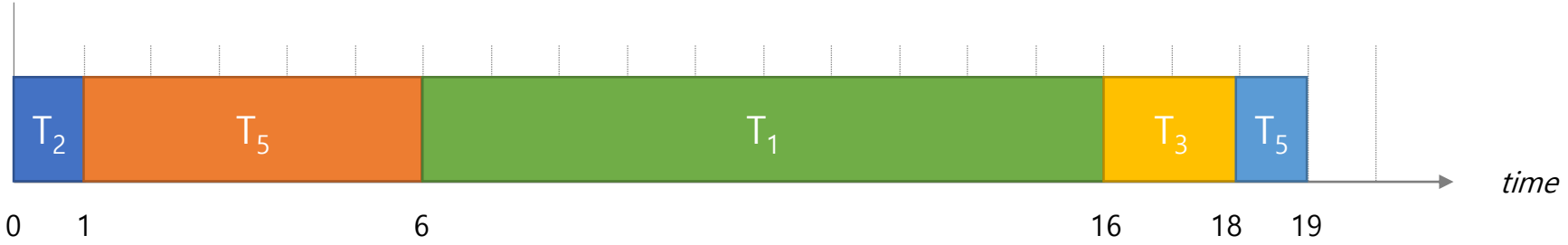
Source: Operating System Concepts 9th Edition

CPU Scheduling (2/3)

- *Priority CPU Scheduling*
  - Real-time priority are assigned to time-sensitive tasks
  - Shared server can cause priority inversion

e.g.

Process	Burst Time	Priority
T1	10	3
T2	1	1
T3	2	4
T4	1	5
T5	5	2



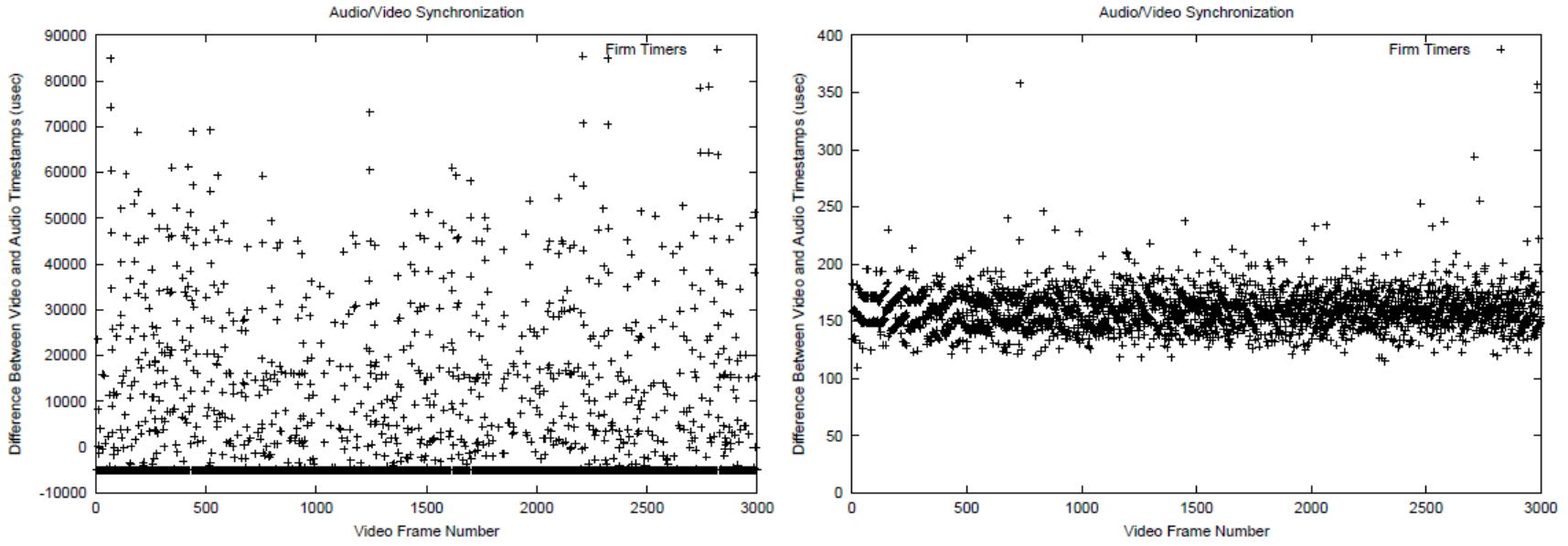
Source: Operating System Concepts 9th Edition

CPU Scheduling (3/3)

- *TLS Scheduling Model*
  - Use the highest locking priority (HLP) protocol
    - When a task acquires a resource, it automatically gets the highest priority of any task that can acquire this resource
  - Combine proportion-period cpu scheduling with priority cpu scheduling
    - Proportion-period tasks can be scheduled with the high priority when access high priority task.

Evaluation (1/2)

- Latency in Real Applications - Mplayer

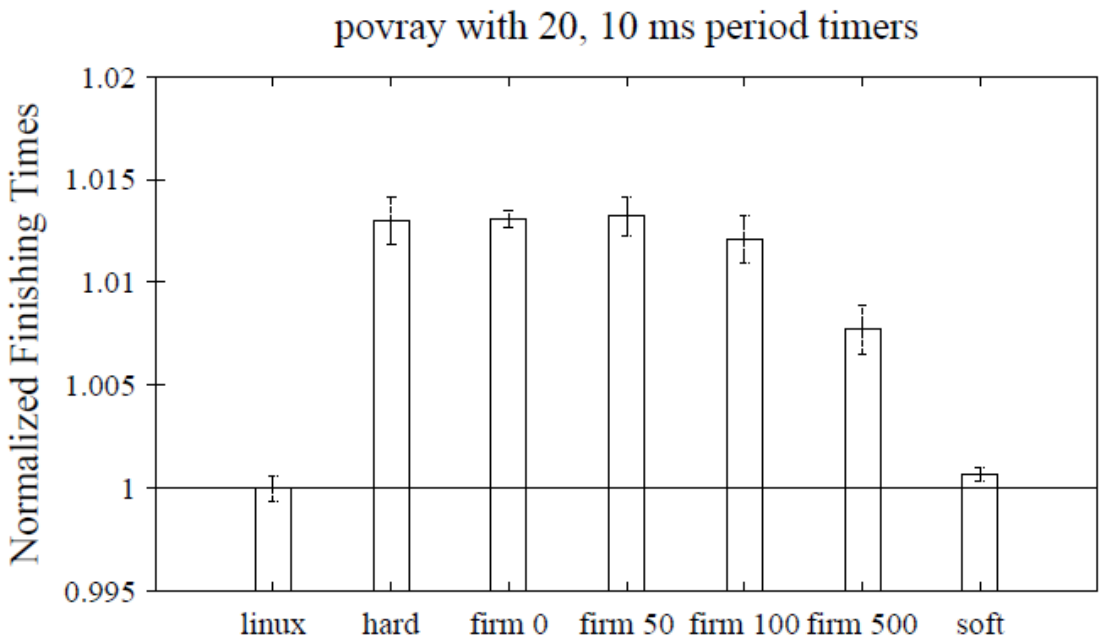


(a) Linux (b) TSL

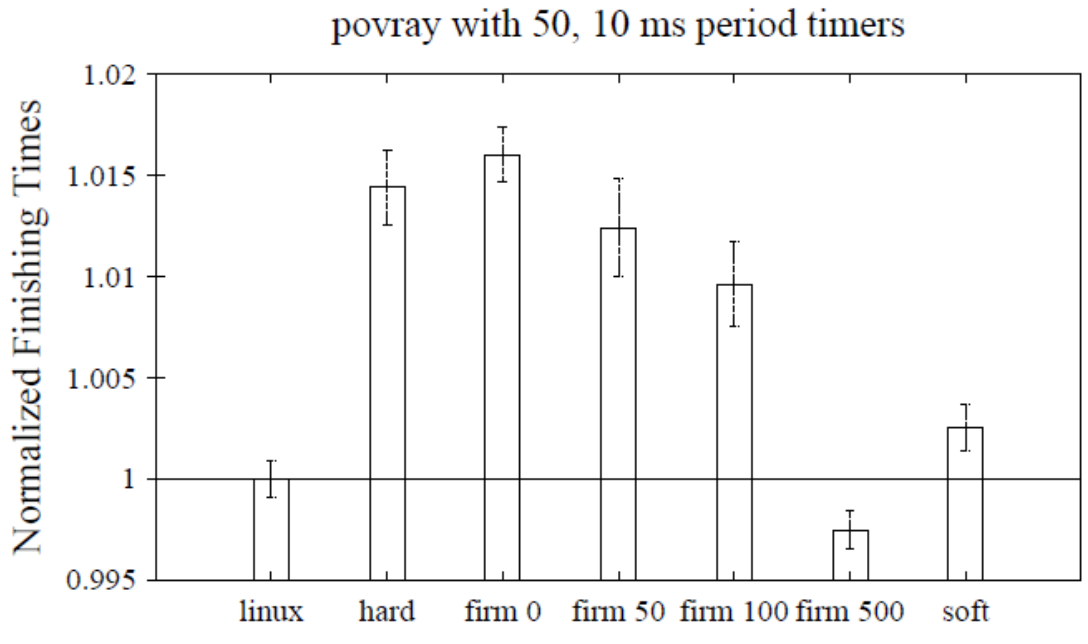
Audio/Video Skew on Linux and on TSL with kernel CPU load.

Evaluation (2/2)

- System Overhead - Firm Timers



Overhead of firm timers in TSL as compared to standard Linux with 20 timer processes



Overhead of firm timers in TSL as compared to standard Linux with 50 timer processes

Conclusions

- Time-Sensitive Linux (TSL)
  - Support applications requiring fine-grained resource allocation and low latency response
  - Three key techniques
    - Firm timers – support accurate timing
    - Fine-grained kernel preemptibility - improve kernel responsiveness
    - Porportion-period scheduling – provide precise allocations to tasks

Further Technical Trends (1/2)

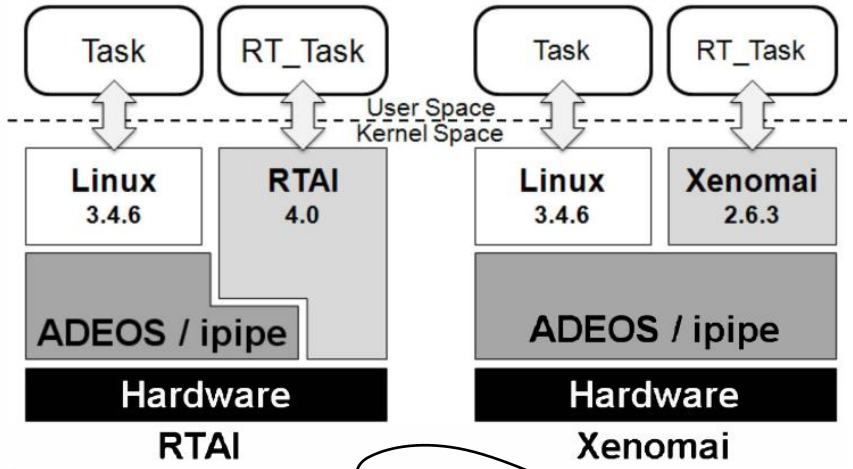
- Improve preemption capabilities of Linux kernel

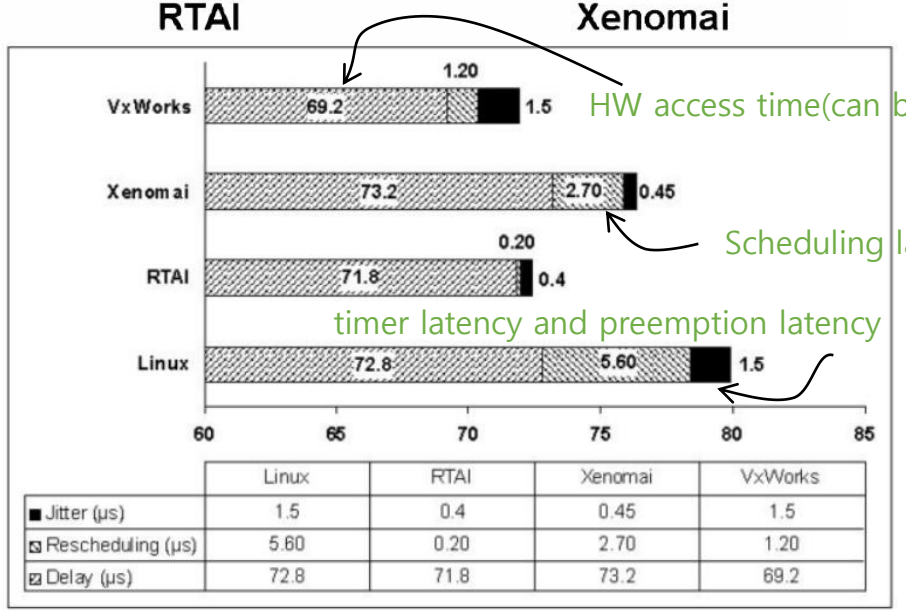
Kernel version	Preemption Capability
Early Linux Kernel 1.x	No In-Kernel preemption
SMP Linux Kernel 2.x	No In-Kernel preemption BLK SMP Lock
SMP Linux Kernel 2.2 2.4	No In-Kernel preemption Spin-locked Critical Sections
Preemptible Linux Kernel 2.4	Preemptible Non-Preemptible Spinlock Sections
Linux Kernel 2.6.x	Preemptible Non-Preemptible Spinlock Sections Preemptible BKL (since 2.6.11)
RT-Preempt Linux Kernel 2.6.x	Preemptible Kernel Critical Sections Preemptible IRQ Subsystem Mutex Locks with Priority Inheritance

Source: Dietrich, Sven-Thorsten, and Daniel Walker. "The evolution of real-time linux." *7th RTL Workshop*. 2005.

Further Technical Trends (2/2)

- Dual-Kernel Approaches
  - RTLinux
    - Microkernel architecture to handle interrupt controller only
  - RTAI
    - ADEOS nanokernel
    - Interrupt handling on both ADEOS and RTAI
  - Xenomai
    - ADEOS nanokernel
    - Interrupt handing on ADEOS only
    - Support domain migration of real-time tasks





	Linux	RTAI	Xenomai	VxWorks
Jitter (μs)	1.5	0.4	0.45	1.5
Rescheduling (μs)	5.60	0.20	2.70	1.20
Delay (μs)	72.8	71.8	73.2	69.2

Source: Y. Lee, M. E. Jang, J. Kim, and Y. W. Park, "Performance analysis of software architectures with real-time kernel patches for the Rescue Robot," in International Conference on Control, Automation and Systems, 2014, pp. 460-465.
A. Barbalace, A. Luchetta, G. Manduchi, M. Moro, A. Soppelsa, and C. Taliercio, "Performance comparison of VxWorks, Linux, RTAI, and Xenomai in a hard real-time application," IEEE Transactions on Nuclear Science, vol. 55, pp. 435-439, 2008.

Q & A

(Thank you!)