Empirische Bestimmung der Echtzeitfähigkeit von Linux-Systemen

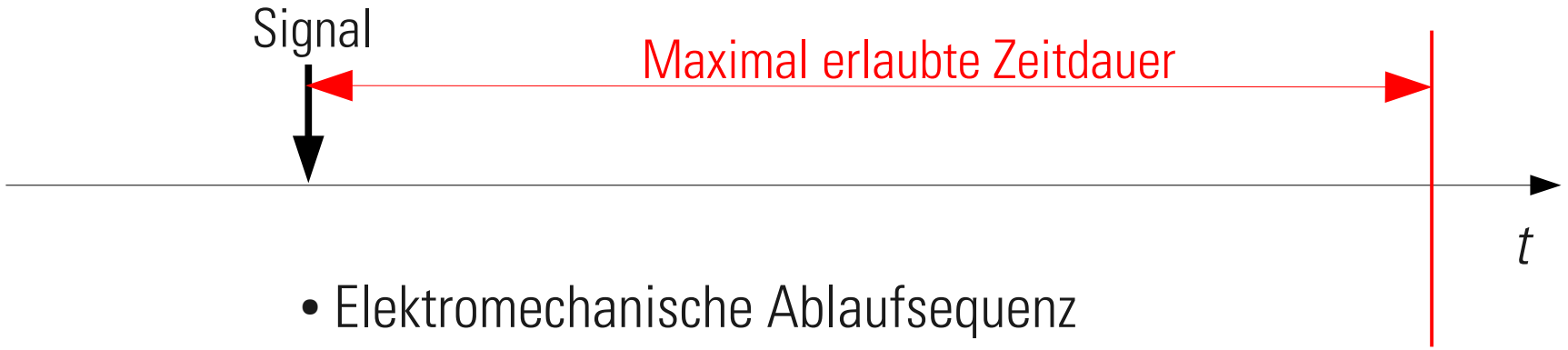
Carsten Emde

Open Source Automation Development Lab

(OSADL) eG

Linux Kernel Development
Free and Open Software Learning Centre (FOSSLC) e.V.
26.1.2011, Ilmenau

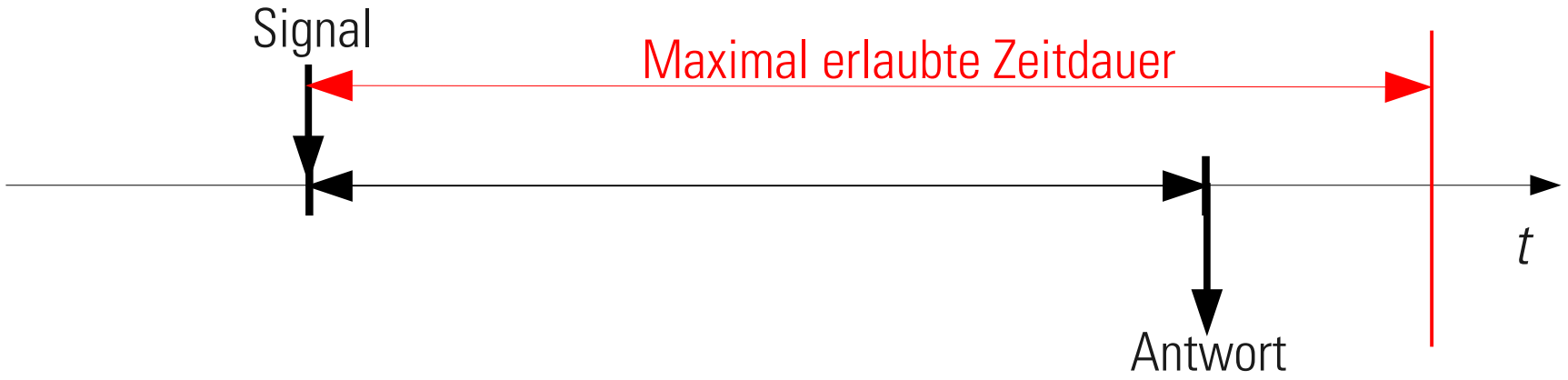
Wann ist ein System ein Echtzeitsystem?



- Elektromechanische Ablaufsequenz
- Zyklusdauer der Hauptschleife
- FIFO-Länge bzw. Datenrate im Kommunikationskontroller

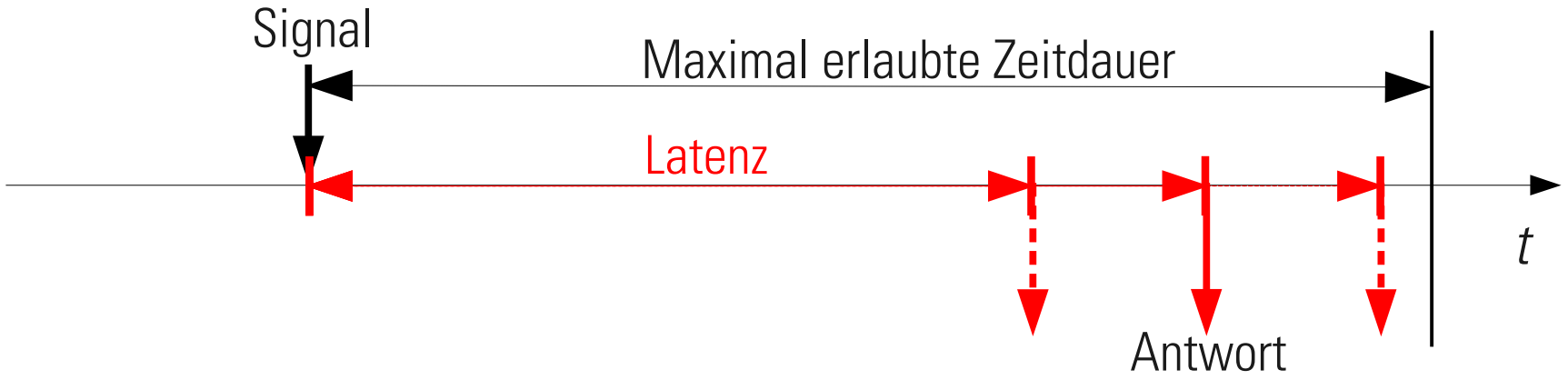
Wann ist ein System ein Echtzeitsystem?

"Ein Echtzeitsystem beantwortet ein asynchron einfallendes externes Signal *immer* in einer vorgegebenen *maximal erlaubten Zeitdauer*."



Wann ist ein System ein Echtzeitsystem?

"Ein Echtzeitsystem beantwortet ein asynchron einfallendes externes Signal *immer* in einer vorgegebenen *maximal erlaubten Zeitdauer*."

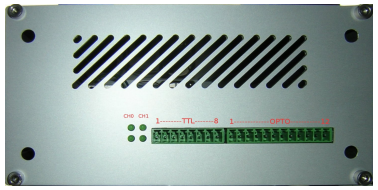


"Die maximale Latenz eines Echtzeitsystems ist *immer* kleiner als die vorgegebene *maximal erlaubte Zeitdauer*."

Wie misst man die Latenz eines Echtzeitsystems?

Externe Messung mit Simulation

OSADLs „Latency-Box“



Interne Latenz-Aufzeichnung

Eingebaute Kernel-Latenz-Histogramme

```
CONFIG_WAKEUP_LATENCY_HIST=y
CONFIG_MISSED_TIMER_OFFSETS_HIST=y
```

Interne Messung mit Simulation

Cyclictest

```
# cyclictest -a -t -n -p99
```

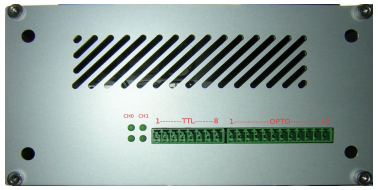
Interne Messung in Programm-Hauptschleife

Applikation

```
# <application>
```

Latenz-Tests in vier Stufen

Externe Messung mit Simulation
OSADLs „Latency-Box“



Interne Latenz-Aufzeichnung
Eingebaute Kernel-Latenz-Histogramme

```
CONFIG_WAKEUP_LATENCY_HIST=y
CONFIG_MISSED_TIMER_OFFSETS_HIST=y
CONFIG_INTERRUPT_OFF_HIST=y
CONFIG_PREEMPT_OFF_HIST=y
```

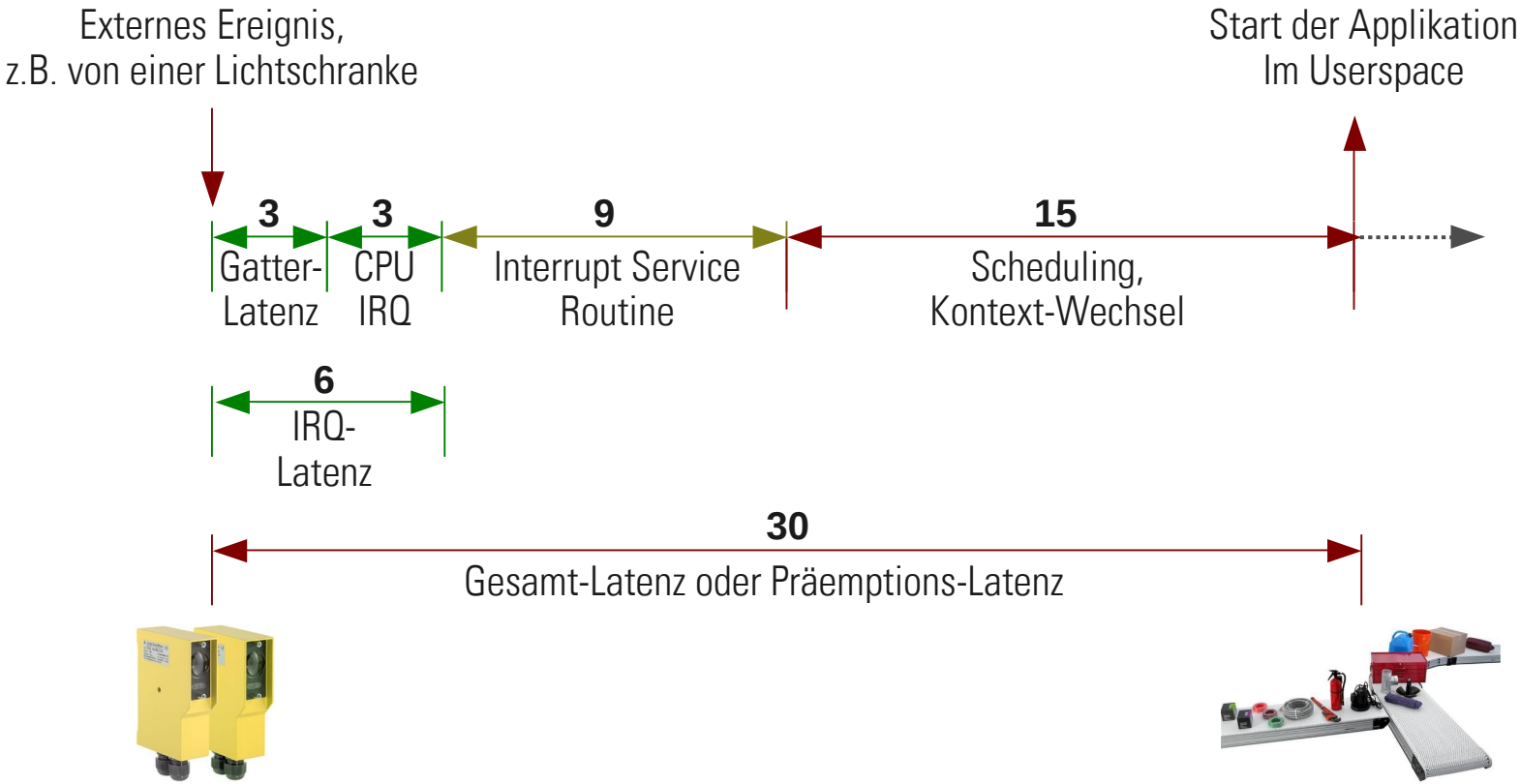
Interne Messung mit Simulation
Cyclictest

```
# cyclictest -a -t -n -p99
```

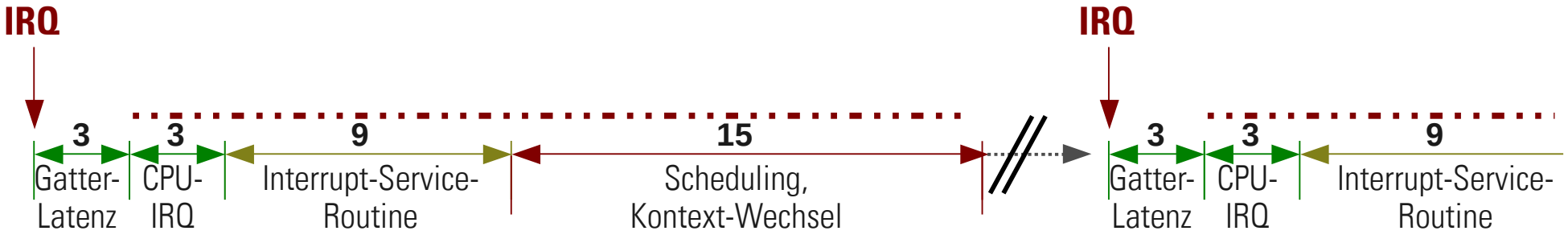
Interne Messung in Programm-Hauptschleife
Applikation

```
# <application>
```

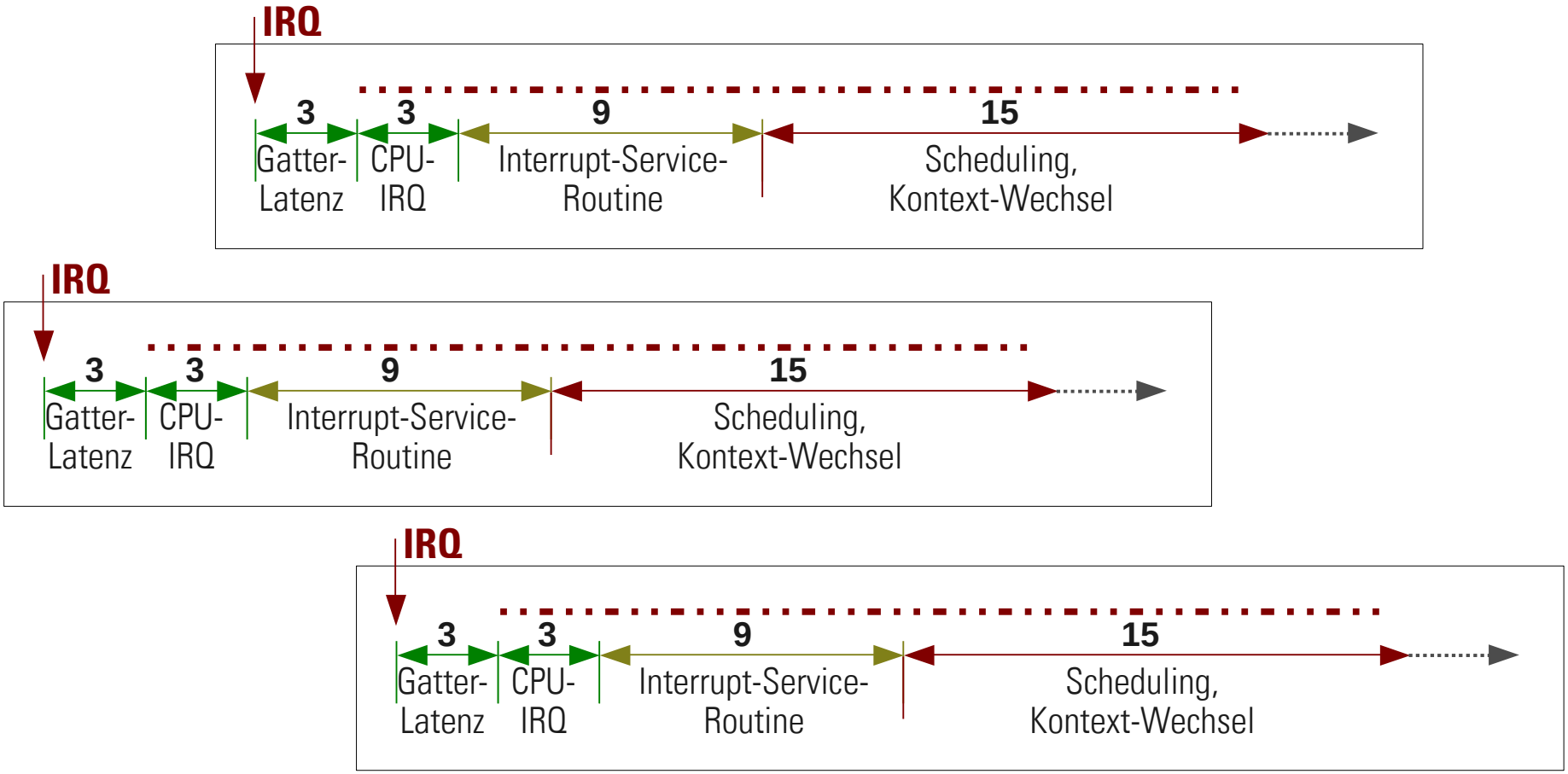
Signalpfad in der Messung



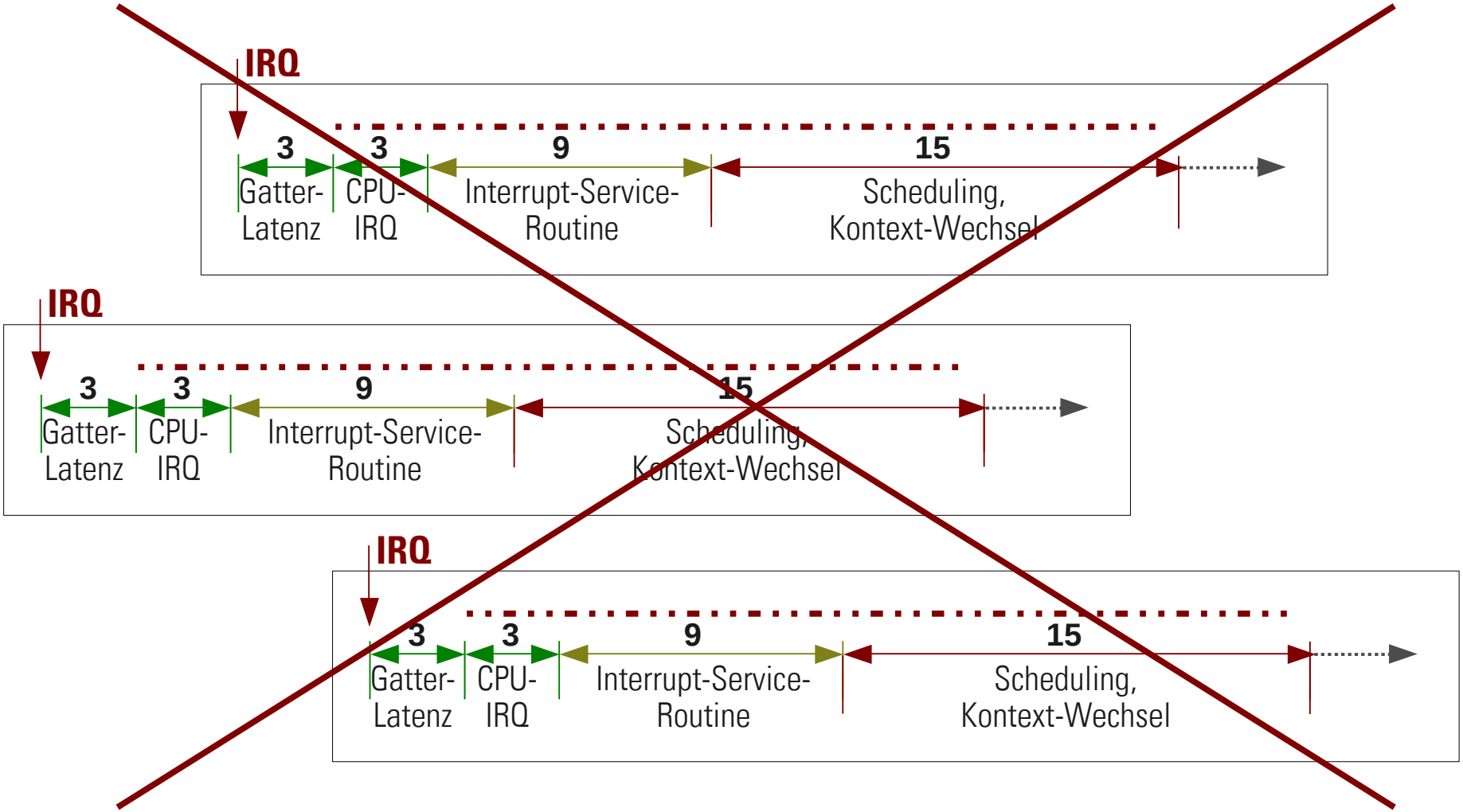
Latenzquellen (Single-Task)



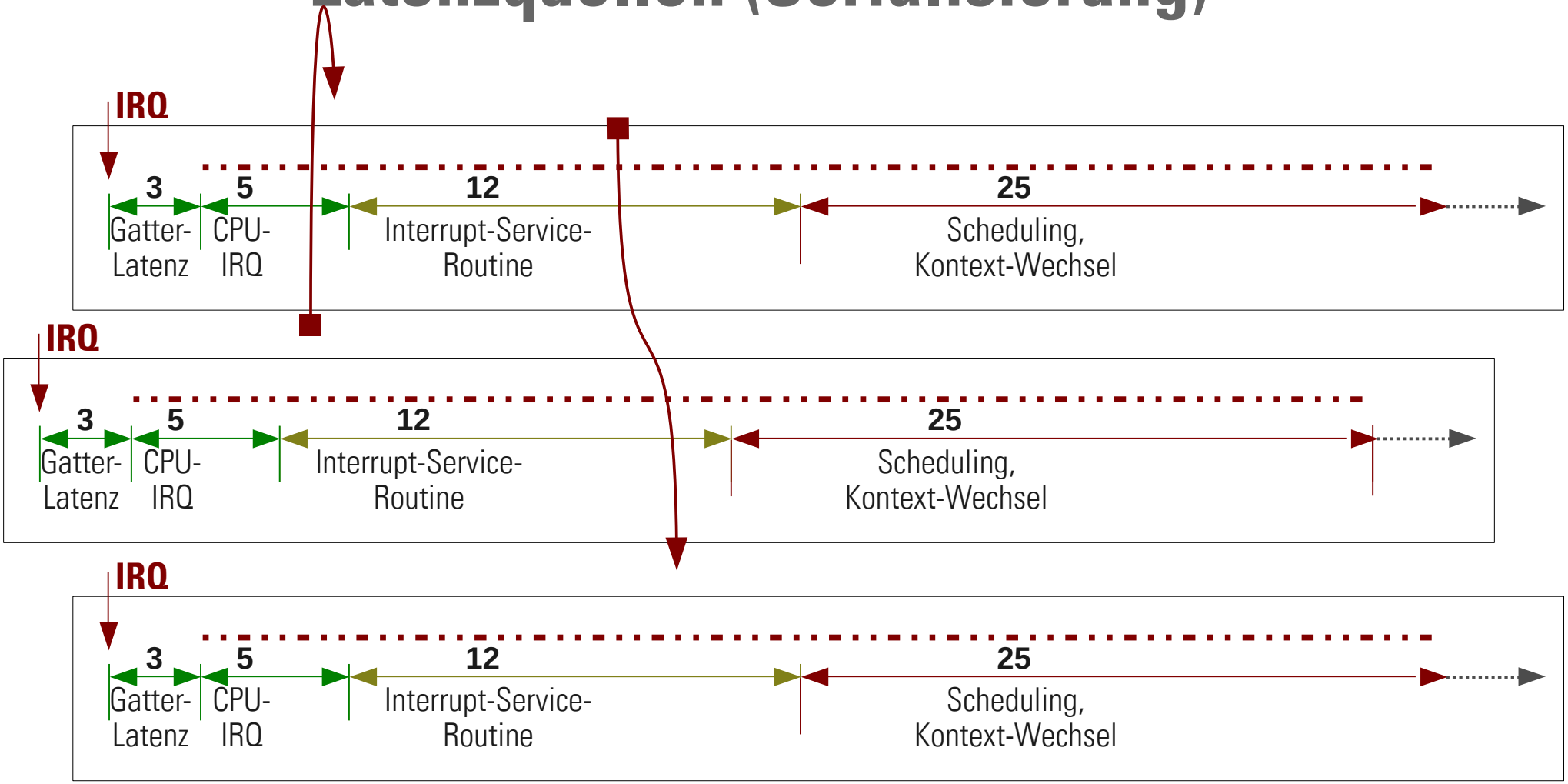
Latenzquellen (Multi-Tasking)



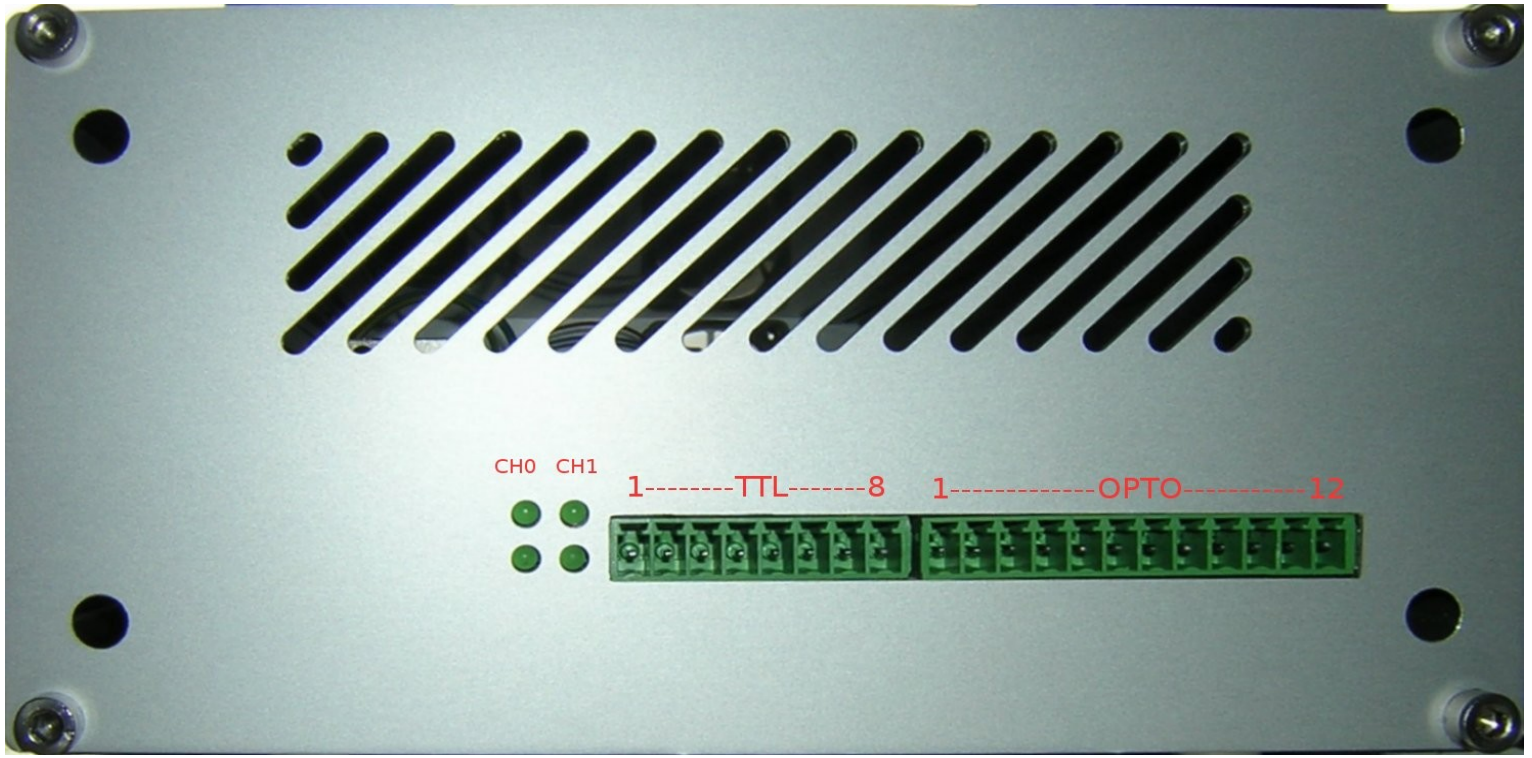
Latenzquellen (Multi-Tasking)



Latenzquellen (Serialisierung)



OSADLs „Latency Box“



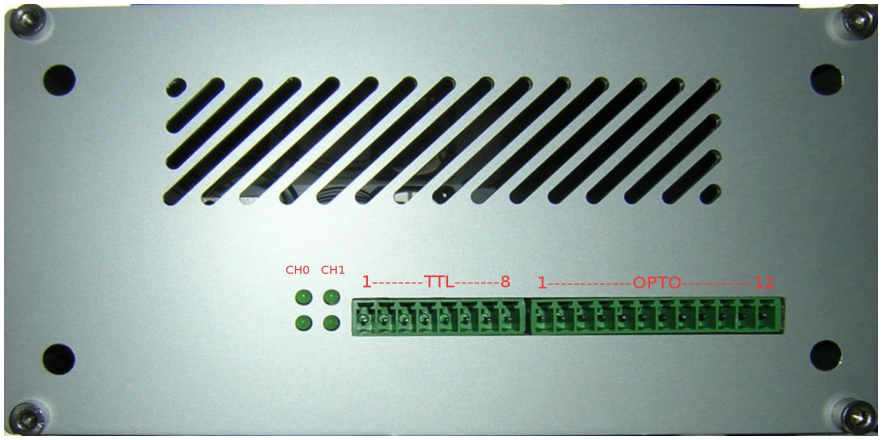
ELTEC systems

Linux Kernel Development
Free and Open Software Learning Centre (FOSSLC) e.V.
26.1.2011, Ilmenau

FOSSLC

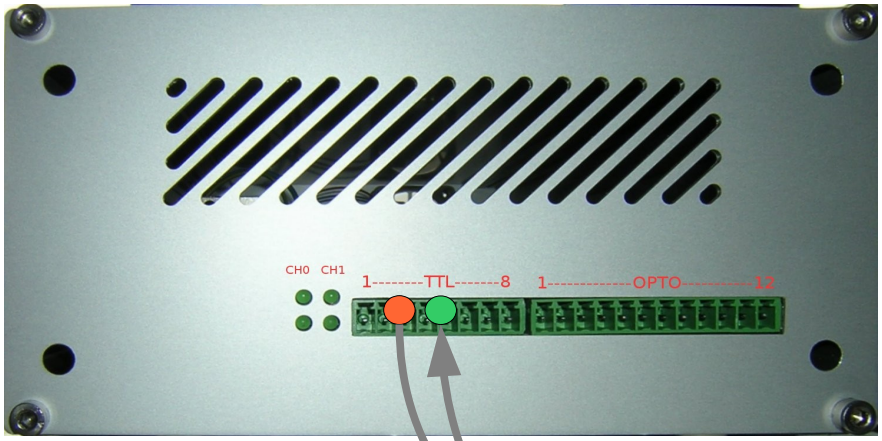
OSADL
Open Source Automation Development Lab eG

OSADLs „Latency Box“ - Spezifikation

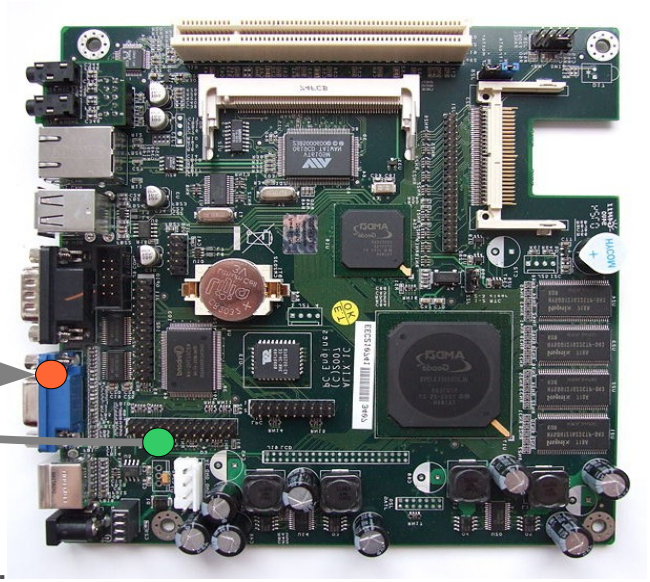


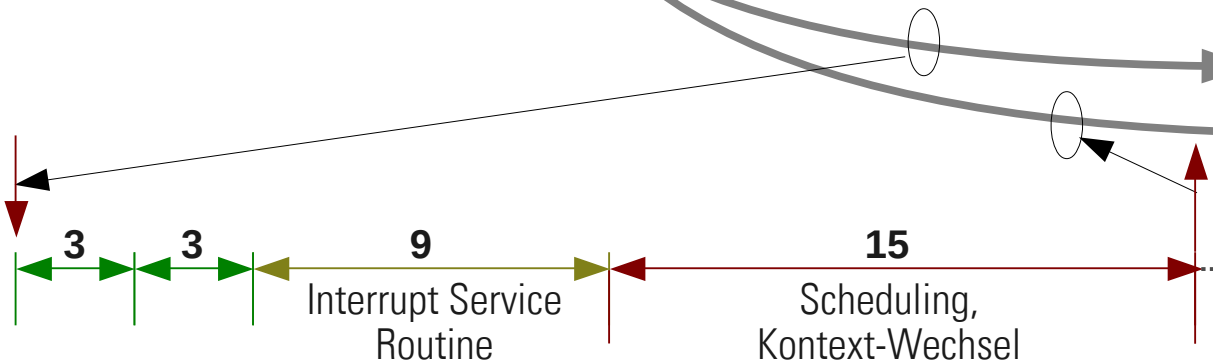
PowerPC 750FX@600MHz
64 MB SDRAM auf SODIMM, 16 MB Flash-EPROM
10/100 Mb/s Netzwerk
2 serielle Kanäle RS232 and RS485
2 TTL Outputs, 4 TTL Inputs
4 Status LEDs
On-board FPGA

OSADLs „Latency Box“ verbunden mit CPU-Board



PowerPC 750FX@600MHz
64 MB SDRAM auf SODIMM, 16 MB Flash-EPROM
10/100 Mb/s Netzwerk
2 serielle Kanäle RS232 and RS485
2 TTL Outputs, 4 TTL Inputs
4 Status LEDs
On-board FPGA



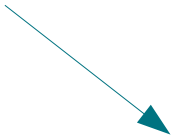


FOSSLC

OSADL
Open Source Automation Development Lab eG

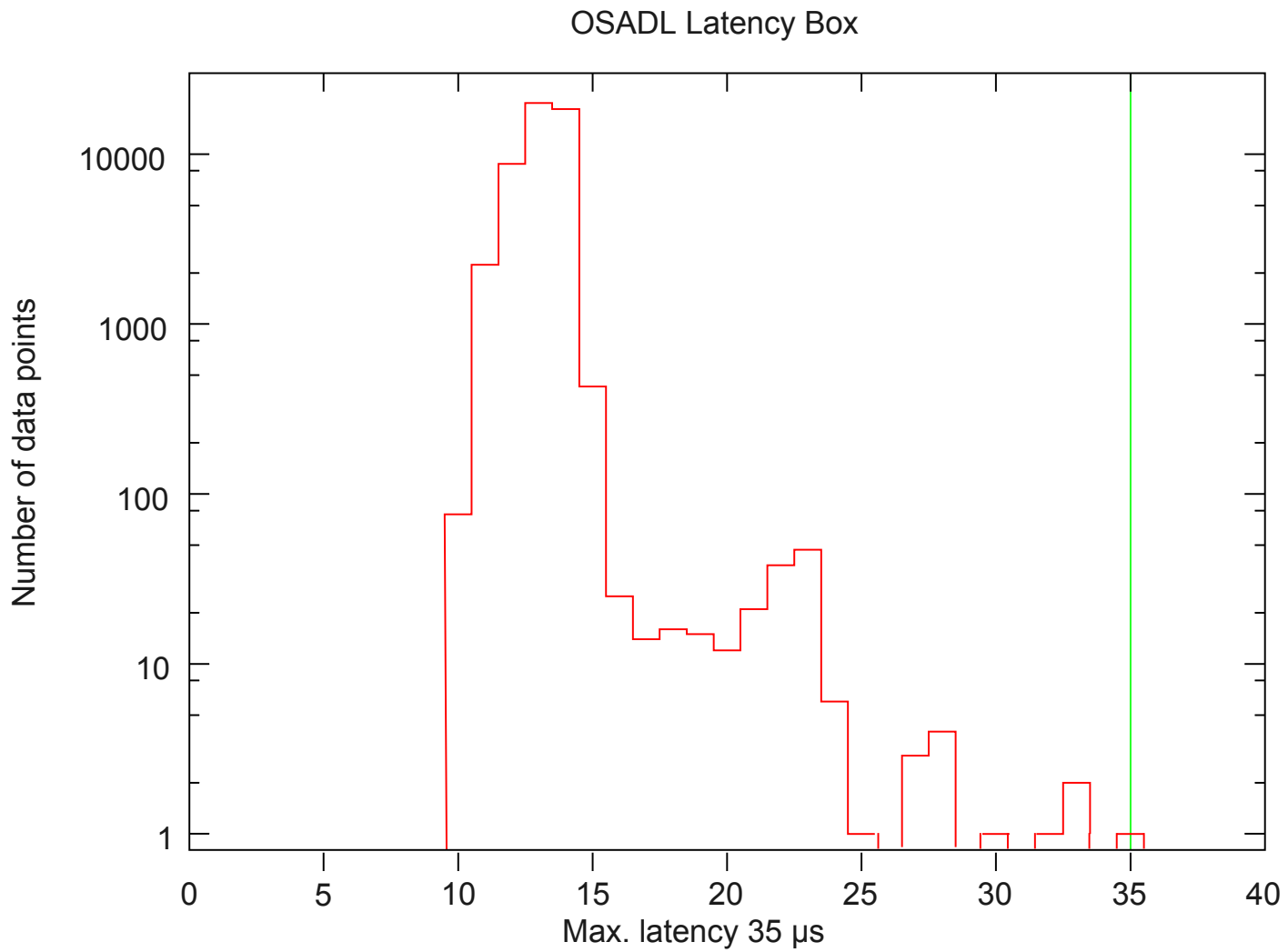
OSADLs „Latency Box“ Daten-Transfer

Histogramm-Daten



```
Linie #1     0      (Keine Latenzwerte zwischen 0 und 1 aufgezeichnet)
             0
             0
             0
             0
             0
             0
             0
             0
             0
Linie #11    76     (Insgesamt 76 Messwerte zwischen 10 und 11 µs)
             2238
             8800
             20027  (Am häufigsten vorkommender Latenzwert zwischen 13 und 14 µs)
             18433
             430
             25
             14
             [..]
Linie #1000  0      (Kein Overflow)
```

OSADLs „Latency Box“ - Latenz-Plot



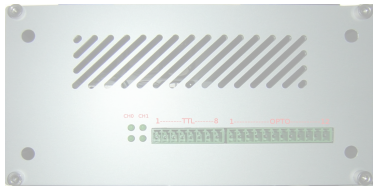
Linux Kernel Development
Free and Open Software Learning Centre (FOSSLC) e.V.
26.1.2011, Ilmenau

FOSSLC

OSADL
Open Source Automation Development Lab eG

Latenz-Tests in vier Stufen

Externe Messung mit Simulation
OSADLs „Latency-Box“



Interne Latenz-Aufzeichnung
Eingebaute Kernel-Latenz-Histogramme

```
CONFIG_WAKEUP_LATENCY_HIST=y
CONFIG_MISSED_TIMER_OFFSETS_HIST=y
```

Interne Messung mit Simulation
Cyclictest

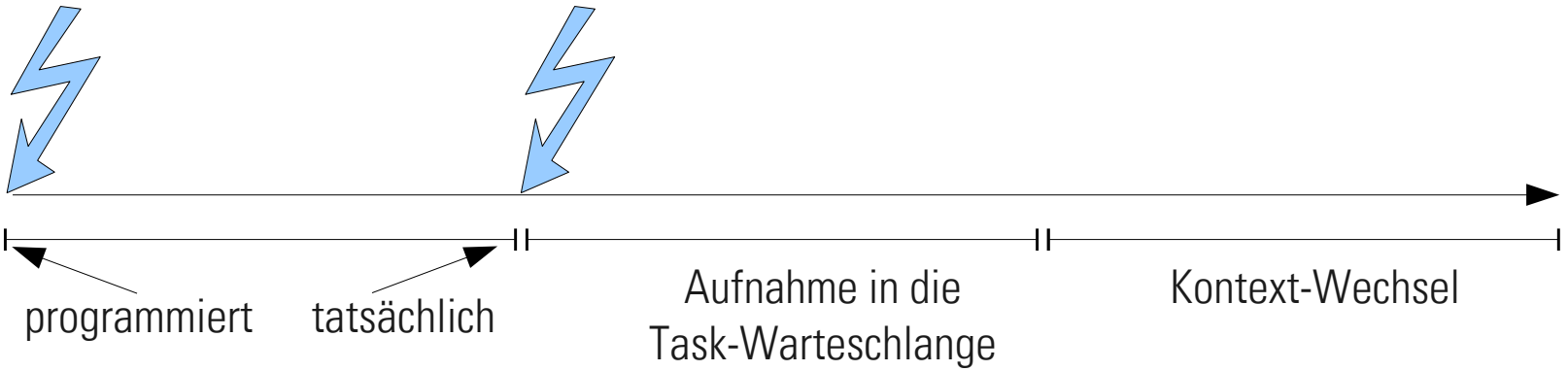
```
# cyclictest -a -t -n -p99
```

Interne Messung in Programm-Hauptschleife
Applikation

```
# <application>
```

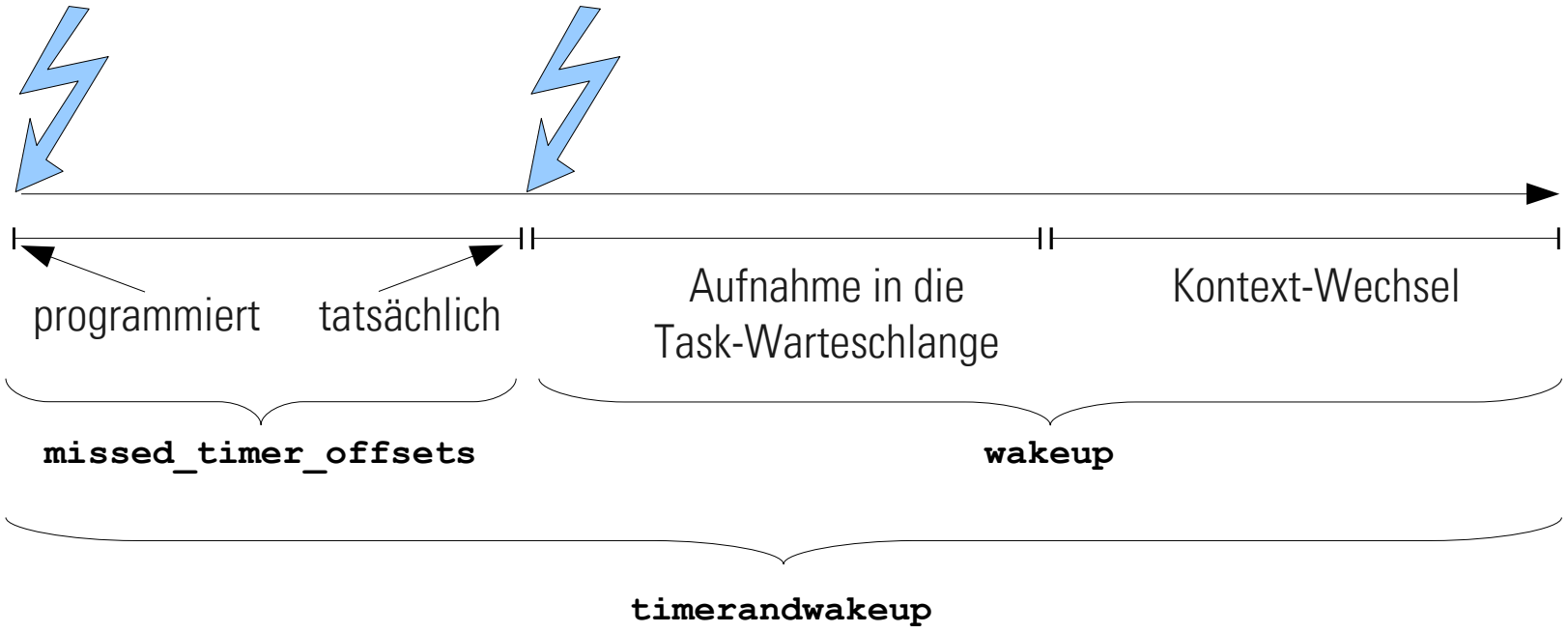
Interne Registrierung von Latenzen

Restart einer wartenden Applikation durch abgelaufenen Timer



Interne Registrierung von Latenzen

Restart einer wartenden Applikation durch abgelaufenen Timer



Interne Latenz-Aufzeichnung

Kernel-Konfiguration

```
CONFIG_WAKEUP_LATENCY_HIST=y
CONFIG_MISSED_TIMER_OFFSET_HIST=y
```

Zugang über Debug-Filesystem

Kommando

```
mount -t debugfs nodev /sys/kernel/debug
```

Eintrag in /etc/fstab

```
nodev /sys/kernel/debug debugfs defaults 0 0
```

Verzeichnisse

```
/sys/kernel/debug/tracing/latency_hist/enable

/sys/kernel/debug/tracing/latency_hist/wakeup

/sys/kernel/debug/tracing/latency_hist/missed_timer_offsets

/sys/kernel/debug/tracing/latency_hist/timerandwakeup
```

Interne Latenz-Aufzeichnung

Dateien

Latenz-Aufzeichnung einschalten

```
echo 1 >/sys/kernel/debug/tracing/latency_hist/enable/preemptirqsoff
echo 1 >/sys/kernel/debug/tracing/latency_hist/enable/wakeup
echo 1 >/sys/kernel/debug/tracing/latency_hist/enable/missed_timer_offsets
```

Latenz-Daten in Histogrammform

```
/sys/kernel/debug/tracing/latency_hist/wakeup/CPU?

/sys/kernel/debug/tracing/latency_hist/missed_timer_offsets/CPU?

/sys/kernel/debug/tracing/latency_hist/timerandwakeup/CPU?
```

Histogramm-Management - Reset

Reset

```
#!/bin/bash

HISTDIR=/sys/kernel/debug/tracing/latency_hist
if test -d $HISTDIR
then
  cd $HISTDIR
  for i in `find . | grep /reset$`
  do
    echo 1 >$i
  done
fi
```

Histogramm-Management - Datenauswertung

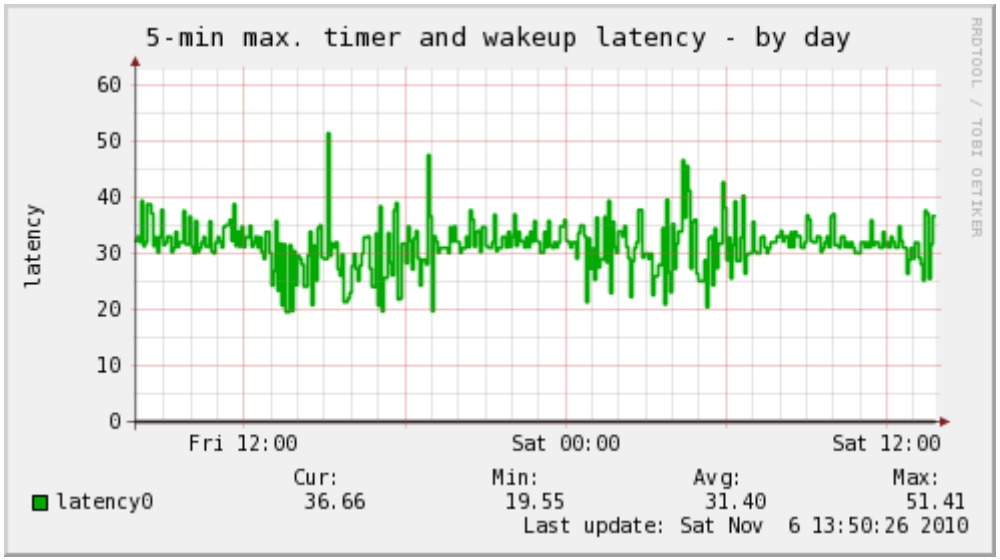
Data

```
# grep -v " 0$" /sys/kernel/debug/tracing/latency_hist/irqsoff/CPU0
#Minimum latency: 0 microseconds.
#Average latency: 0 microseconds.
#Maximum latency: 63 microseconds.
#Total samples: 2622976567
#There are 0 samples greater or equal than 10240 microseconds
#usecs              samples
    0           2174555930
    1            251129896
    2            108221353
    3             22726693
    4             17853433
    5             20486535
    6             13811530
    7              6996682
    8              3464499
    9              2084766
   10               832247
   11               366531
   12               158594
   13                67561
   14                40456
   15                28985
   16                21873
   17                16504
```

Leistungsverluste durch kontinuierliche Latenz-Aufzeichnung

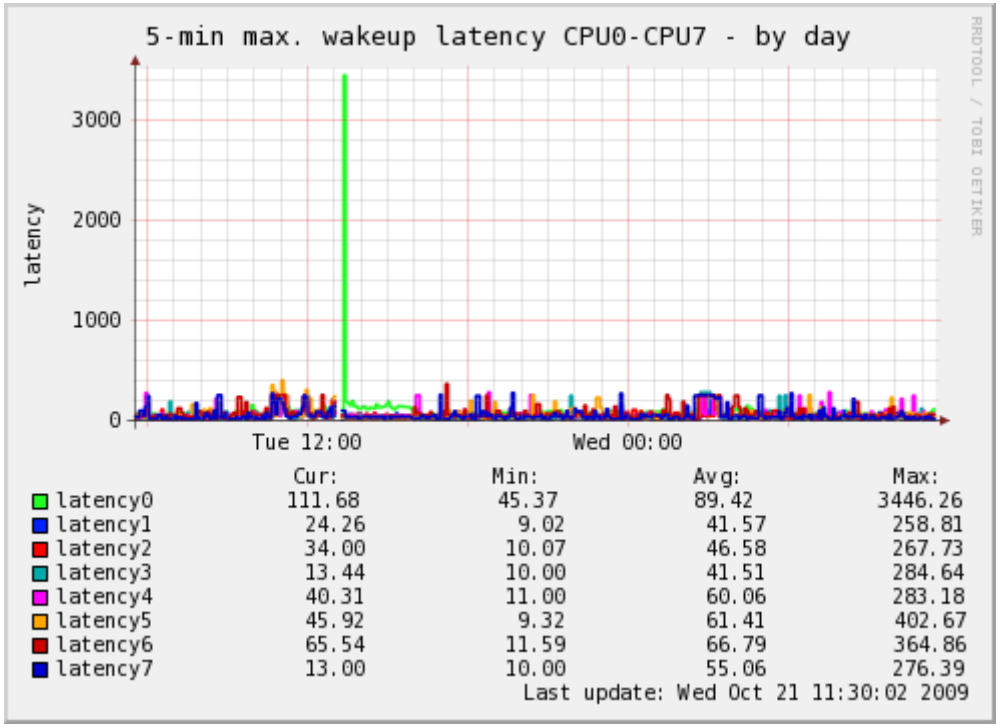
Die interne Aufzeichnung von Timer-Verzögerung und Wakeup-Latenz hat einen vernachlässigbaren Effekt auf die CPU-Leistung von unter 1%. Dadurch ist es möglich, diese Latenzen kontinuierlich unter Produktionsbedingungen zu registrieren (sogar während der gesamten Lebensdauer einer Maschine).

Kontinuierliche Latenz-Aufzeichnung (1)



(Munin Monitoring-Tool)

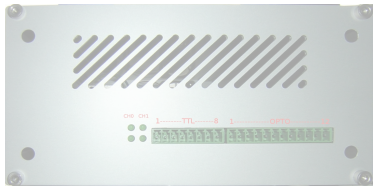
Kontinuierliche Latenz-Aufzeichnung (2)



Einmalige zu Kalibrationszwecken künstlich erzeugte Latenz

Latenz-Tests in vier Stufen

Externe Messung mit Simulation
OSADLs „Latency-Box“



Interne Latenz-Aufzeichnung
Eingebaute Kernel-Latenz-Histogramme

```
CONFIG_WAKEUP_LATENCY_HIST=y
CONFIG_MISSED_TIMER_OFFSETS_HIST=y
```

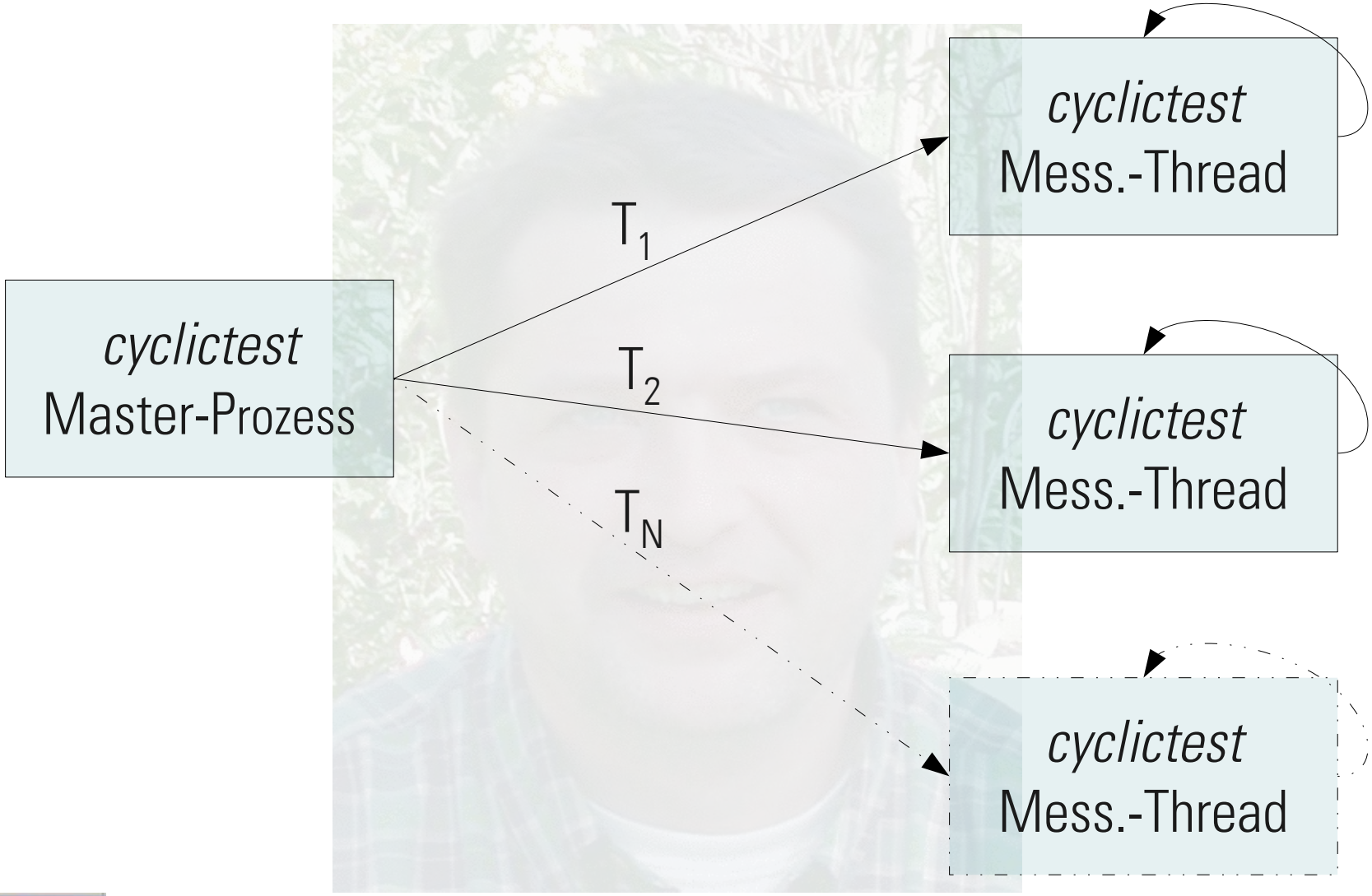
Interne Messung mit Simulation
Cyclictest

```
# cyclictest -a -t -n -p99
```

Interne Messung in Programm-Hauptschleife
Applikation

```
# <application>
```

Cyclictest - Prinzip



Cyclictest: Kommandozeile

```
# cyclictest -a -t -n -p99 -i100 -d50
560.44 586.11 606.12 211/1160 3727
T: 0 (18617) P:99 I:100 C:1,011,846,111 Min:  2 Act:  4 Avg:  5 Max: 39
T: 1 (18618) P:98 I:150 C:  708,641,019 Min:  2 Act:  5 Avg: 11 Max: 57
```

-a PROC *Affinity:* Alle Threads laufen auf Prozessor **PROC**. Wenn **PROC** nicht angegeben wird, läuft Thread #N auf Prozessor #N.

-t NUM *Threads:*.Erzeuge **NUM** Test-Threads (default ist 1). Wenn **NUM** nicht angegeben wird, wird die Anzahl vorhandener Prozessoren für **NUM** verwendet.

-n *Nanosleep*. Benutze **clock_nanosleep()** für alle Zeitabfragen. Dies ist Standard und sollte immer verwendet werden..

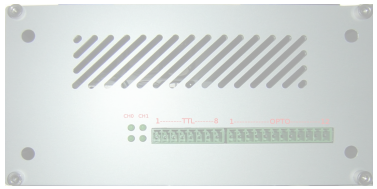
-p99 *Priority*. Setze die Priorität des ersten Threads.Jeder folgende Thread erhält diese Priorität reduziert um die Nummer des jeweilgen Threads.

-i100 *Interval*. Wiederholrate des ersten Threads in µs (default ist 1000 µs).

-d50 *Delay of additional threads*. Setze den Abstand der Thread-Wiederholrate in µs (default ist 500 µs). Wenn Cyclictest mit der **-t** Option aufgerufen und mehr als ein einziger Thread erzeugt wurde, wird dieser Wert zu der jeweiligen Wiederholrate hinzugezählt.

Latenz-Tests in vier Stufen

Externe Messung mit Simulation
OSADLs „Latency-Box“



Interne Latenz-Aufzeichnung
Eingebaute Kernel-Latenz-Histogramme

```
CONFIG_WAKEUP_LATENCY_HIST=y
CONFIG_MISSED_TIMER_OFFSETS_HIST=y
```

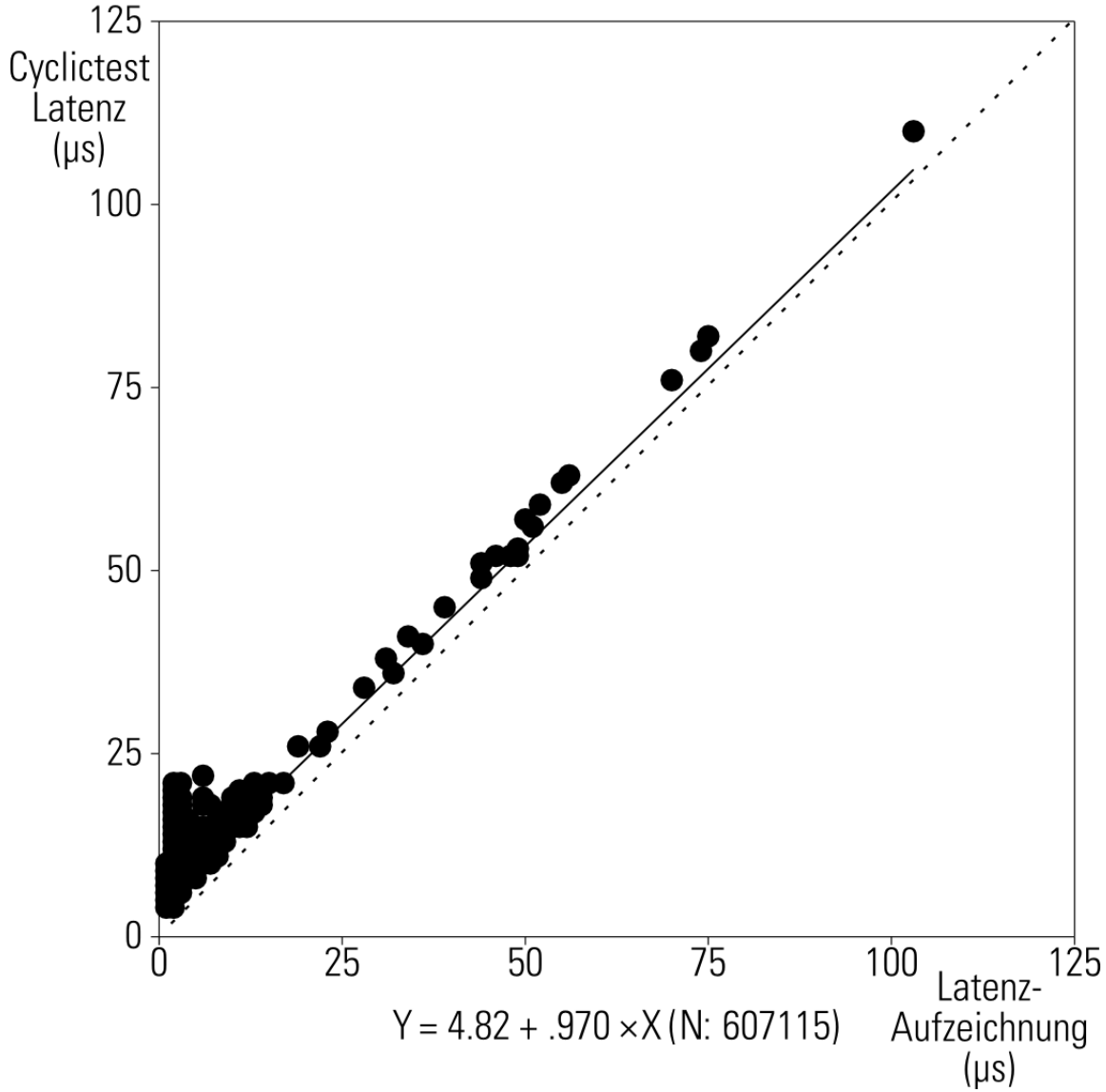
Interne Messung mit Simulation
Cyclictest

```
# cyclictest -a -t -n -p99
```

Interne Messung in Programm-Hauptschleife
Applikation

```
# <application>
```

Cyclictest *vs.* interne Latenz-Aufzeichnung



FOSSLC

OSADL
Open Source Automation Development Lab eG

Uniprozessor *vs.* Multicore-Prozessor (1)

Uniprozessor

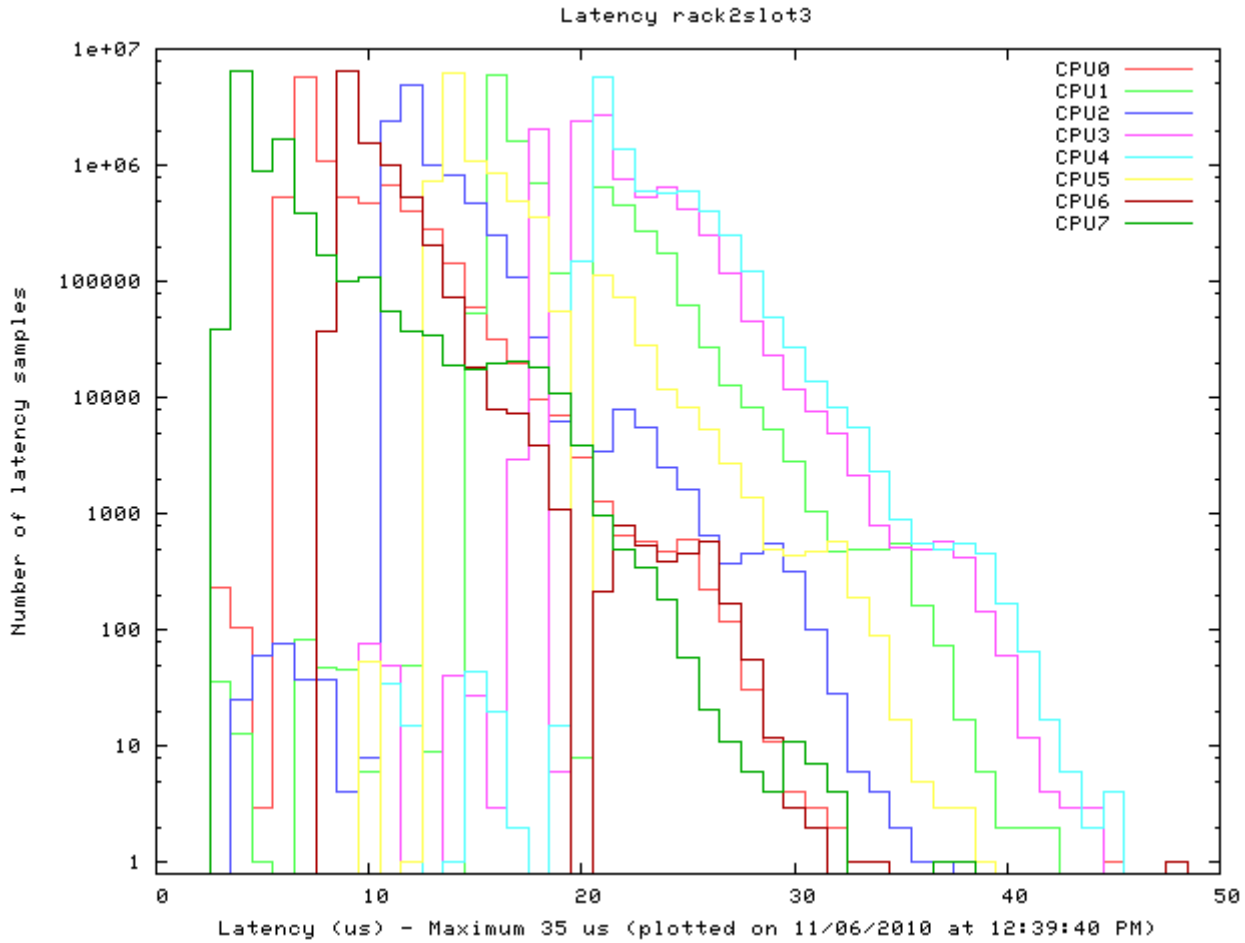
Nur ein einziger Prozess mir der höchsten Priorität des Systems darf sich in der Warteschleife befinden. Ausnahme: Wenn streng sequentielle Ausführung garantiert ist, kann von dieser Regel abgewichen werden.

Befinden sich zwei konkurrierende Prozesse mit der höchsten Priorität des Systems in der Warteschleife, so verlängert sich die Worst-Case-Latenz des Systems (L_{sys}) um die maximale ununterbrochene Laufzeit (T_{exe}) der beiden Prozesse und ergibt die effektive Latenz

$$L_{eff} = L_{sys} + T_{exe}$$

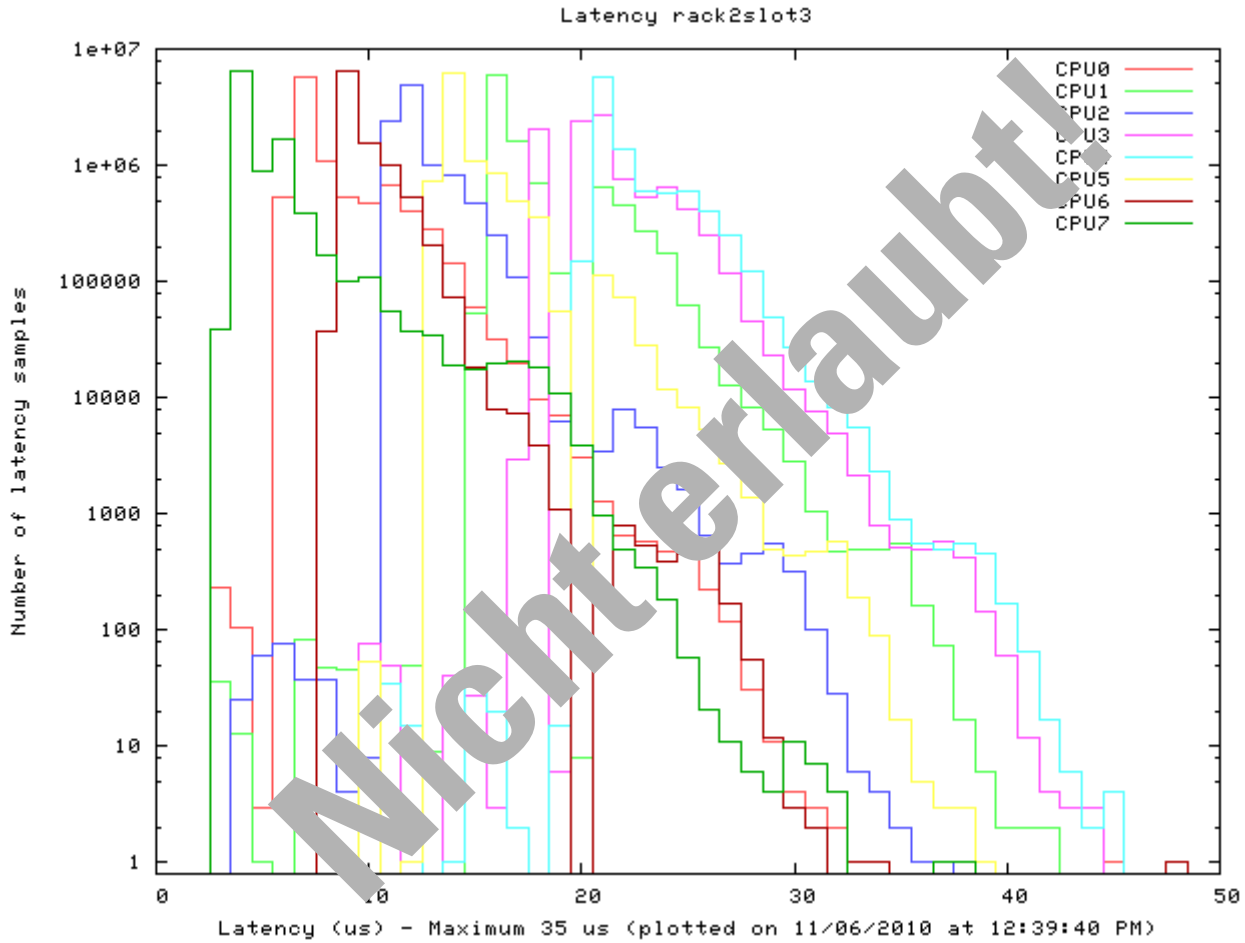
Eine solche Konstellation gilt als Verletzung der Bedingungen eines Echtzeitsystems.

Uniprozessor *vs.* Multicore-Prozessor (2)



Uniprozessor, 8 Prozesse mit Priorität 99

Uniprozessor *vs.* Multicore-Prozessor (3)



Uniprozessor, 8 Prozesse mit Priorität 99

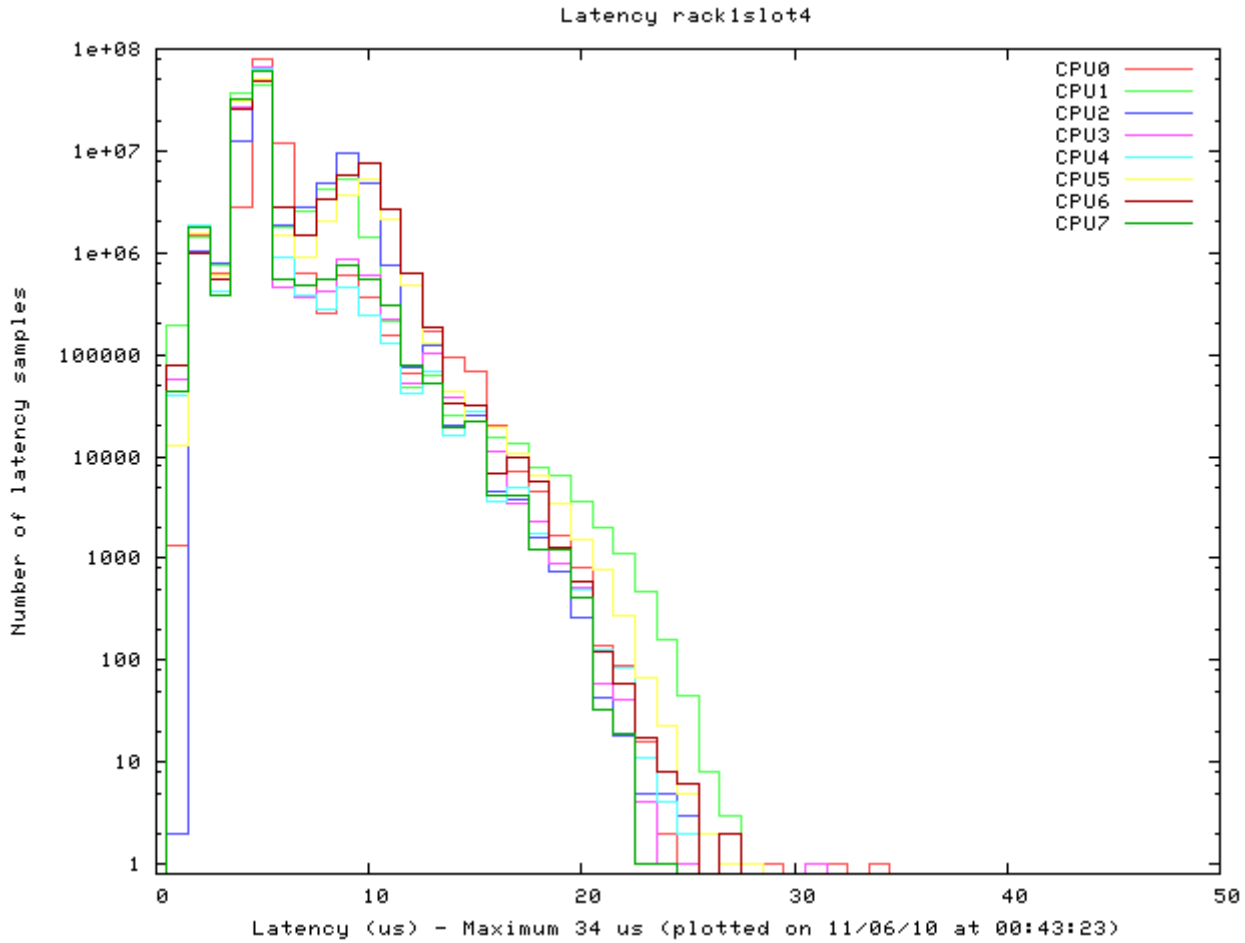
Uniprozessor *vs.* Multicore-Prozessor (4)

Multicore-Prozessor

Es dürfen sich so viele Prozesse mir der höchsten Priorität des Systems in der Warteschleife befinden wie Prozessor-Cores vorhanden sind. Wenn streng sequentielle Ausführung einzelner Prozesse garantiert ist, kann wiederum von dieser Regel abgewichen werden.

Dadurch ist es zum Beispiel möglich, mehrere Echtzeit-Regelprozesse oder mehrere Echtzeit-Kommunikationskanäle unabhängig voneinander auf dem gleichen CPU-Board zu betreiben.

Uniprozessor *vs.* Multicore-Prozessor (5)



Multicore, 8 Prozesse mit Priorität 99

Linux Kernel Development

Free and Open Software Learning Centre (FOSSLC) e.V.

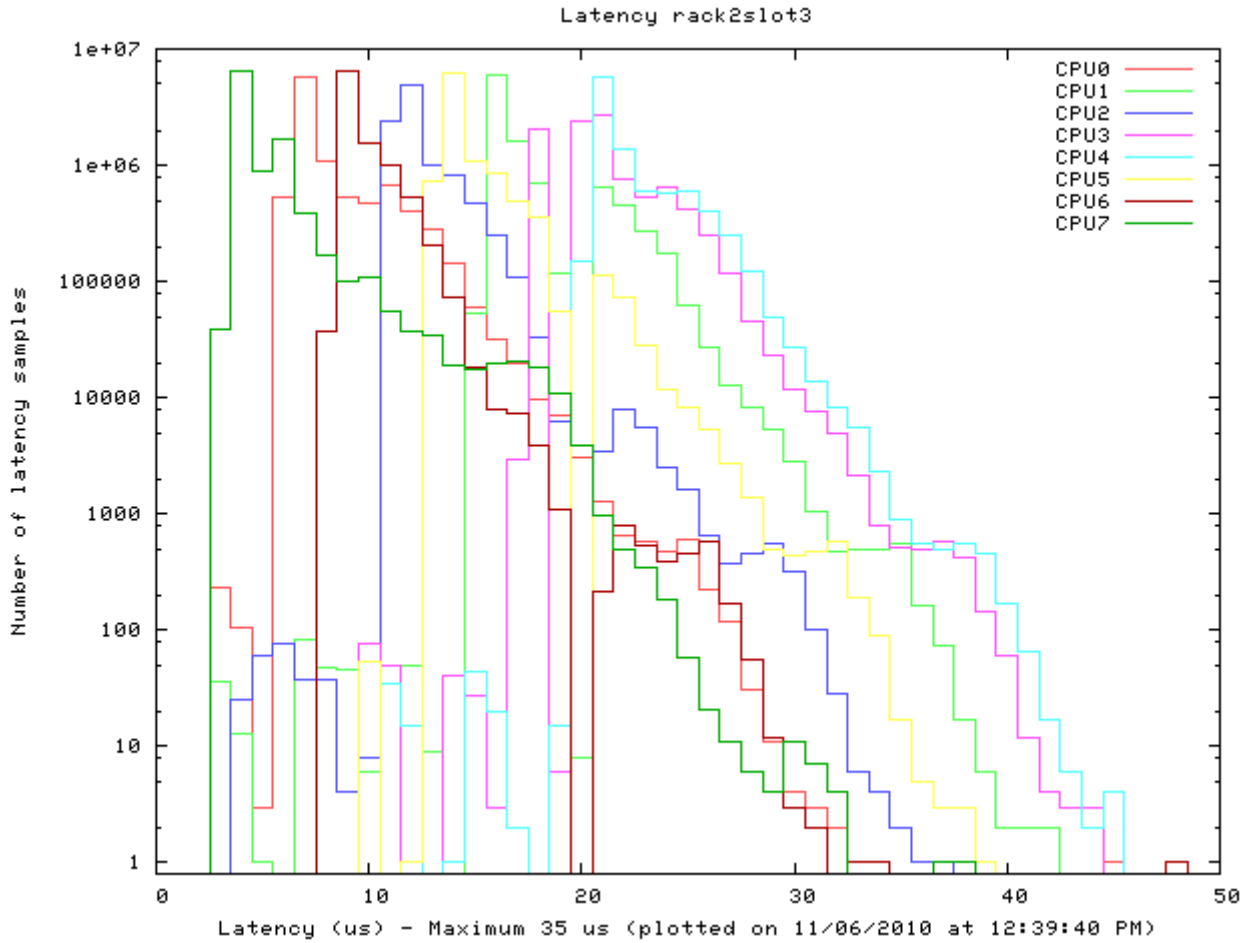
26.1.2011, Ilmenau

FOSSLC

OSADL

Open Source Automation Development Lab eG

Uniprozessor *vs.* Multicore-Prozessor (2)



Uniprozessor, 8 Prozesse mit Priorität 99

“Latency-Fighting“ - Ursachen

Mögliche Ursachen für Latenzen (Auswahl)

Hardware

Lesen/Schreiben von Hardware-Registern dauert zu lange
Umschalten der CPU-Clockfrequenz dauert zu lange

Firmware

System Management Interrupts (SMIs) unterbrechen den Prozessor
Durch Microcode-Patche eingespielte Software-Emulationen dauern zu lange
Power-Management hält den Prozessor zu lange an
Protokoll-Emulationen bzw. -Konversionen (z.B. USB-PS/2) unterbrechen den Prozessor

Software

Interrupts/Preemption zu lange abgeschaltet

“Latency-Fighting“ - Bekämpfung (1)

Kernel-Tracing (ftrace)

- Interface in /sys/kernel/debug/tracing
- Verfügbare Tracer:

```
# cat /sys/kernel/debug/tracing/available_tracers
```

- Aktivieren des Funktions-Tracers:

```
# echo function >/sys/kernel/debug/tracing/current_tracer
```

- Cyclictest -fb<latency> Option:

```
# cyclictest -n -p99 -i1000 -fb500
```

“Latency-Fighting“ - Bekämpfung (2)

Hardware-Latency-Detektor (hwlatdetect)

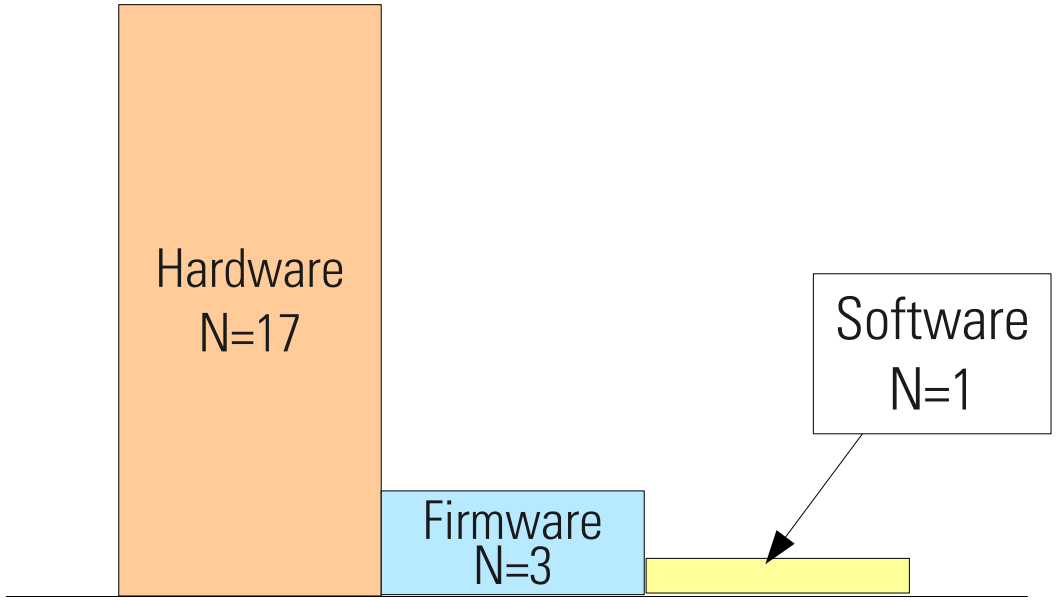
- Zyklischer Aufruf von `stop_machine()`, Auslesen der CPU TSCs und Suche nach Lücken

```
# hwlatdetect
hwlatdetect:  test duration 120 seconds
   parameters:
        Latency threshold: 10us
        Sample window:     1000000us
        Sample width:      500000us
     Non-sampling period:  500000us
        Output File:       None

Starting test
test finished
Max Latency: 835us
Samples recorded: 1
Samples exceeding threshold: 1
1264105805.0765999851  835
```

“Latency-Fighting“ - Bekämpfung (3)

OSADL hilft: Latency Fighters <latency-fighters@osadl.org>



Insgesamt 21
Anfragen

Welche Messmethode wann?

Externe Messung mit Simulation

- Evaluation einer geeigneten Hardware-Plattform
- Nach Hardware-Redesign
- Nach Änderung von BIOS-Parametern
- Bei erstmaliger Verwendung einer neuen echtzeitpflichtigen Hardware-Komponente

Interne Messung mit Simulation (cyclictest)

- Wenn Projekt bzw. Projektbedingungen ***noch nicht bekannt*** sind

Interne Latenz-Aufzeichnung

- Wenn Projekt bzw. Projektbedingungen ***bereits bekannt*** sind

Interne Messung mit Simulation (cyclictest) oder interne Latenz-Aufzeichnung

- Nach Einspielen einer neuen RT-Kernel-Version
- Nach Änderung der Kernel-Konfiguration
- Nach Einspielen einer neuen Treiber-Version
- Bei erstmaliger Verwendung einer neuen nicht-echtzeitpflichtigen Hardware-Komponente

Die OSADL QA-Farm (1)

OSADL Test-Rack

- 8 auswechselbare Tabletts
- Jeweils 220 V Stromversorgung, Ethernet, RS233
- 10/100/1000 Mb/s Switch mit Port-Mirroring
- 8-fach Fernsteuerung der Stromversorgung
- 8-fach Seriell-zu-Netzwerk-Adapter
- 8-fach fernsteuerbarer KVM-Switch (optional)
- Idee und Realisation von Kernel-Entwickler Thomas Gleixner
- Erhältlich bei OSADL-Gründungsmitglied Linutronix

Die OSADL QA-Farm (2)

URL: osadl.org/QA

x86/x86_64

AMD
- LX-800 @500 MHz, 32 bit
- Athlon XP 2000+, 32 bit
- Athlon 64 2800+, 64 bit
- Phenom II X6 @3200 MHz, 64 bit

Intel
- Atom N270 @1600 MHz, 32 bit
- Atom Z530 @1600 MHz, 32 bit
- Celeron M @1500 MHz, 32 bit
- Pentium M @2300 MHz, 32 bit
- Core 2 Duo @2400 MHz, 64 bit
- Core 2 Quad @2400 MHz, 32 bit
- Nehalem 975 @3333 MHz, 32 bit
- Gulftown X980 @3333 MHz, 64 bit

x86

VIA
- C3 Nehemiah @533 MHz, 32 bit
- C7 @1000 MHz, 32 bit

ARM

Marvell
- SheevaPlug @1200 MHz, 32 bit

Texas Instruments
- AM3517 @600 MHz, 32 bit

PowerPC

Freescale
- MPC 5121 @400 MHz, 32 bit

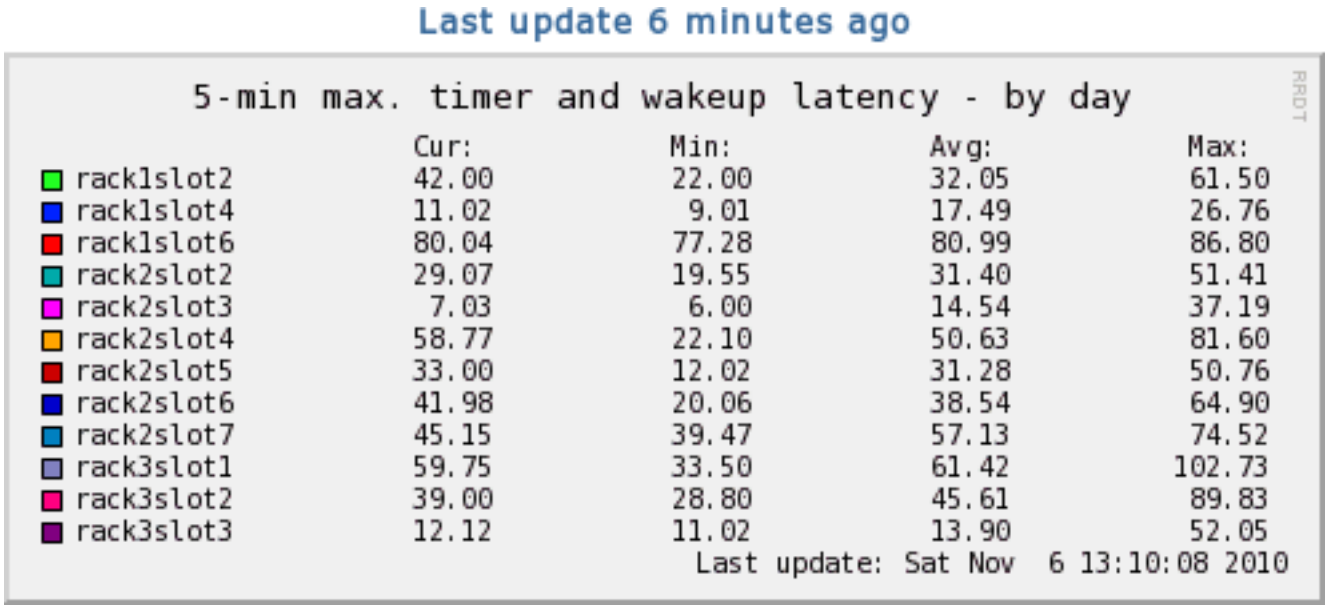
MIPS

ICT
- Loongson 2F @800 MHz, 64 bit

Die OSADL QA-Farm (3)

URL: osadl.org/QA

Kontinuierliche Latenz-Aufzeichnung

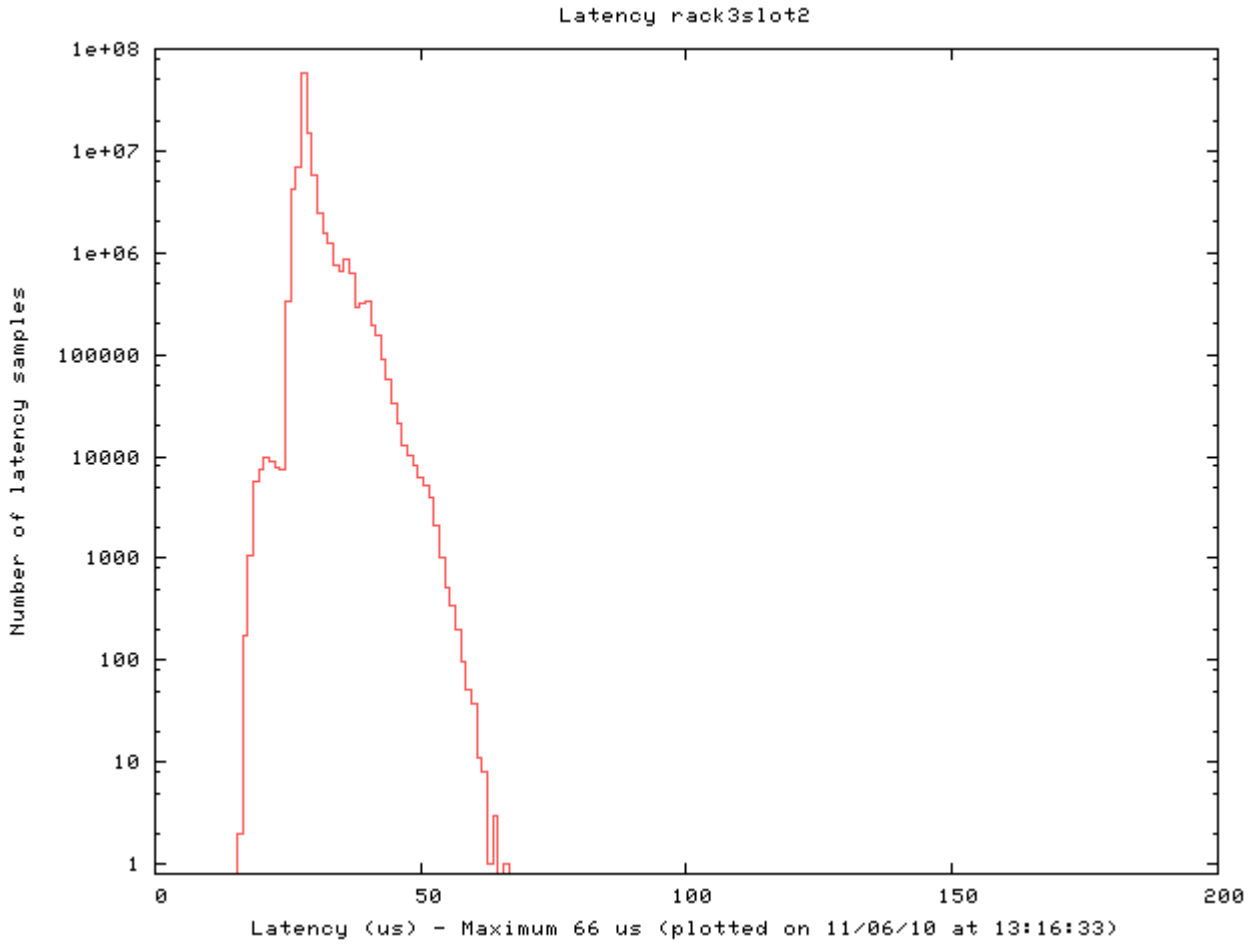


Die OSADL QA-Farm (4)

URL: osadl.org/QA

Individuelle Latenz-Plots mit 10^8 Messwerten unter Idle- und Lastbedingungen

Z.B. VIA C7 @1000 MHz



FOSSLC

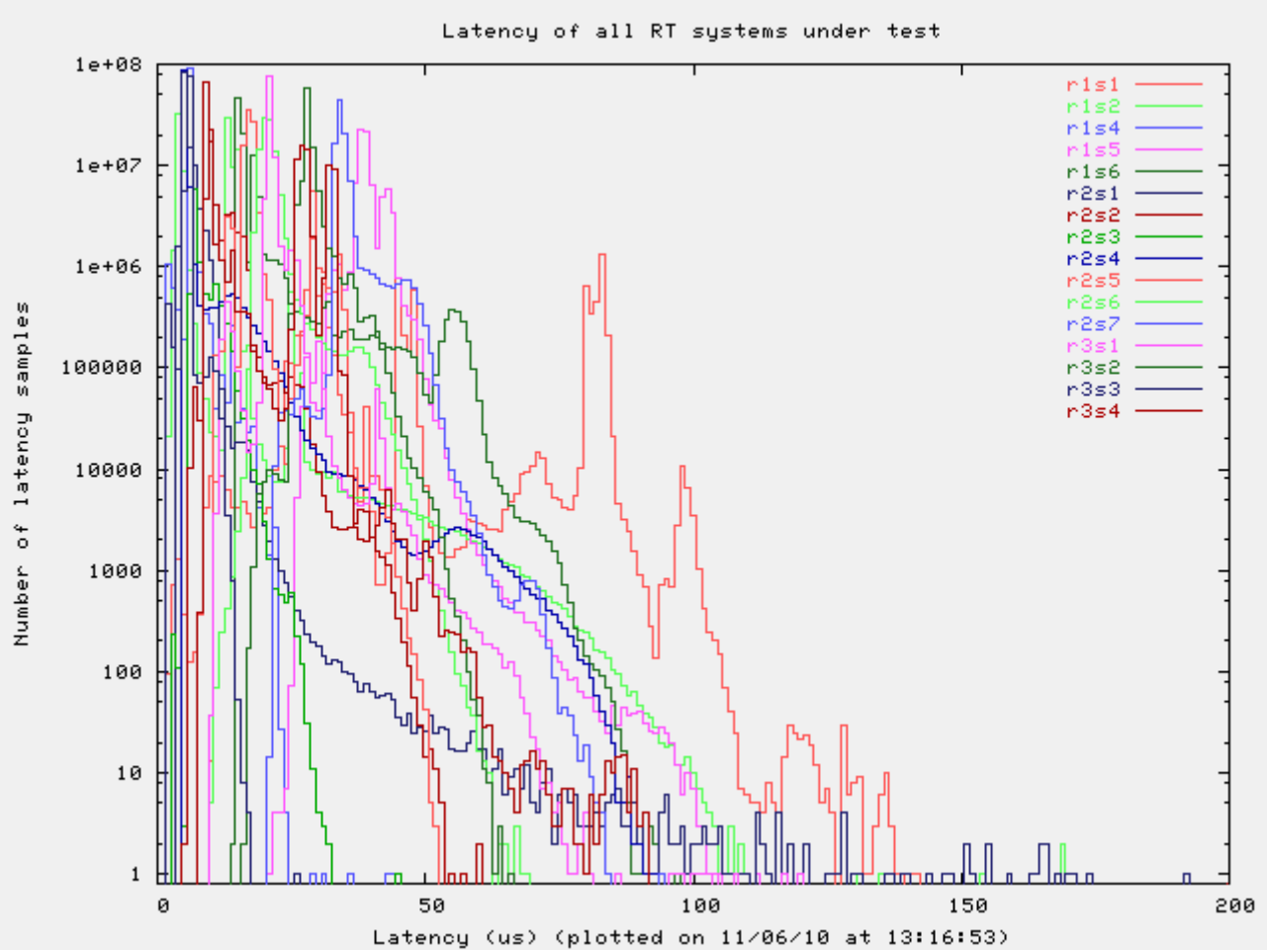
OSADL
Open Source Automation Development Lab eG

Die OSADL QA-Farm (5)

URL: osadl.org/QA

Individuelle Latenz-Plots mit 10^8 Messwerten unter Idle- und Lastbedingungen

Nur RT-Systeme



FOSSLC

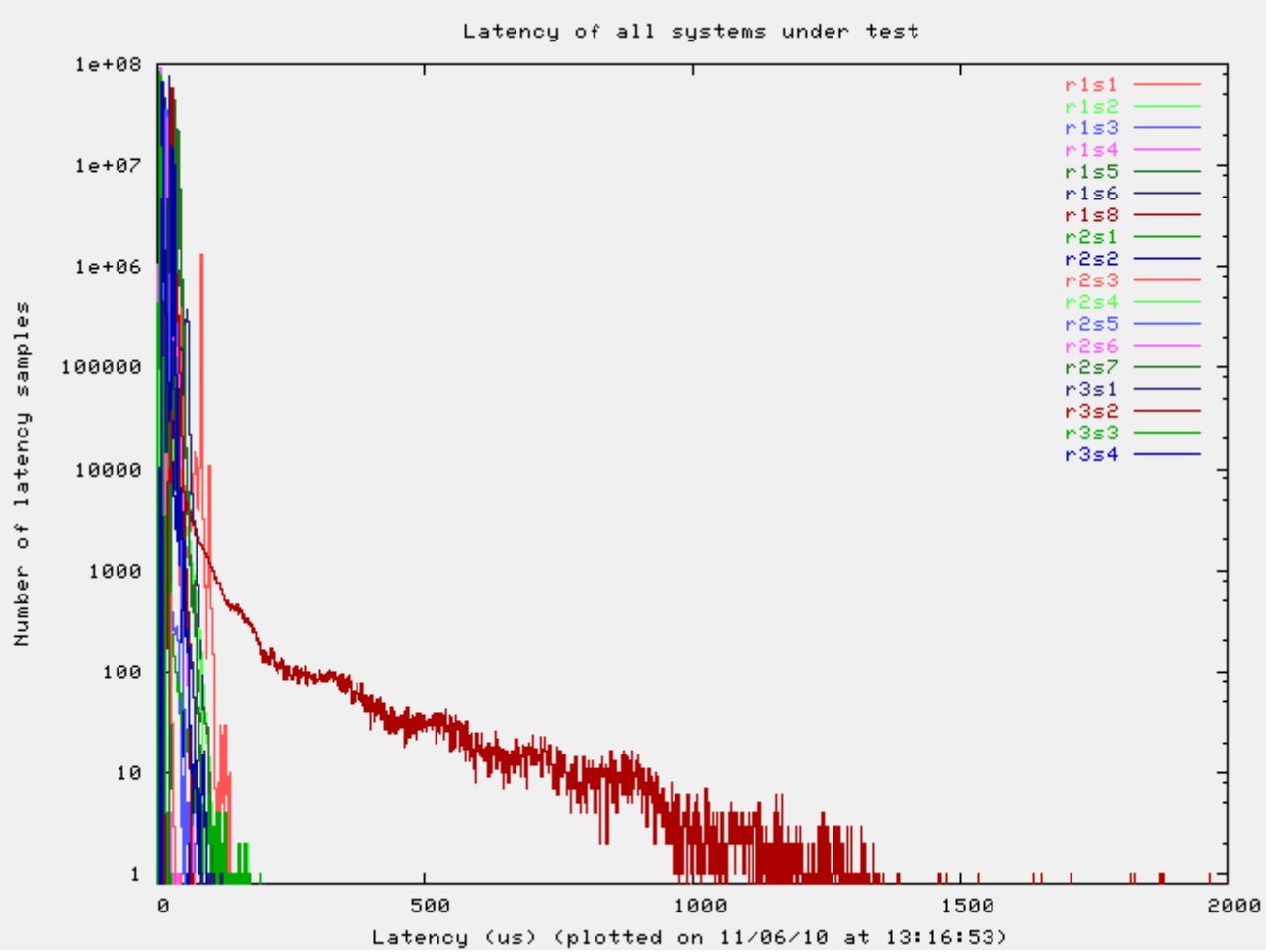
OSADL
Open Source Automation Development Lab eG

Die OSADL QA-Farm (6)

URL: osadl.org/QA

Individuelle Latenz-Plots mit 10^8 Messwerten unter Idle- und Lastbedingungen

Mit Nicht-RT-System als Referenz



FOSSLC

OSADL
Open Source Automation Development Lab eG

Zusammenfassung (1)

- Die Eignung eines Echtzeit-Systems für einen bestimmten Anwendungsfall ergibt sich aus der „Worst-Case-Latenz”.
- Für die jeweils individuellen Messbedingungen stehen externe Latenz-Messung mit Simulation, interne Latenz-Messung mit Simulation und interne Latenz-Aufzeichnung zur Verfügung.
- Damit eine zuverlässige Aussage getroffen werden kann, muss lange genug (z.B. 10^9 Wakeup-Episoden) und unter geeigneter Last gemessen werden. Dafür bietet sich die kontinuierliche kernel-interne Latenzmessung besonders an, da diese unter den realen Bedingungen in Feld durchgeführt werden kann.

Zusammenfassung (2)

- Ein ungeeignetes Echtzeit-Design kann die Echtzeitfähigkeit eines Betriebssystems zunichte machen. Wird parallele Echtzeit mehrerer Prozesse benötigt, ist dies unter bestimmten Umständen mit Multicore-Prozessoren möglich.
- Für das Auffinden von Latenzen stehen eine Reihe von effektiven Debug-Tools zur Verfügung, im Zweifelsfall Email an latency-fighters@osadl.org senden.
- Die OSADL-QA-Farm (osadl.org/QA) bietet Referenz-Daten für eine große Anzahl von Prozessor-Architekturen und Konfigurationen.