

# Betriebssysteme (BS)


## VL 14 – Zusammenfassung und Ausblick

Daniel Lohmann

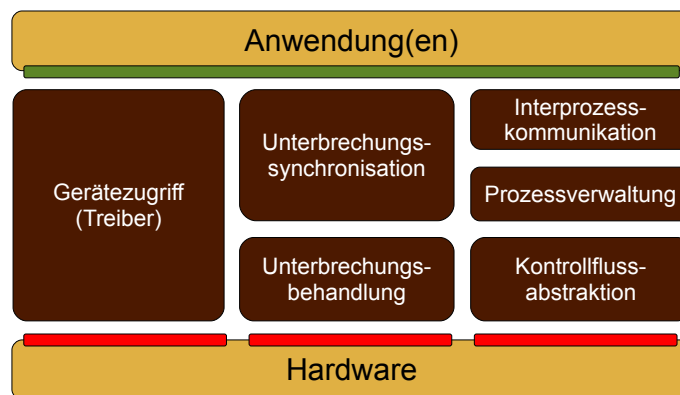
Lehrstuhl für Informatik 4  
Verteilte Systeme und Betriebssysteme


Friedrich-Alexander-Universität  
Erlangen Nürnberg

WS 14 – 27. Januar 2014

 [https://www4.cs.fau.de/Lehre/WS14/V\\_BS](https://www4.cs.fau.de/Lehre/WS14/V_BS)

## Überblick Vorlesungen




 dl Betriebssysteme (VL 14 | WS 14) 14 Zusammenfassung und Ausblick – Ziele und Zielerreichung 14–3

## Lernziele

~ VL 1


- **Vertiefen** des Wissens über die interne Funktionsweise von Betriebssystemen
  - Ausgangspunkt: Systemprogrammierung
  - Schwerpunkt: Nebenläufigkeit und Synchronisation
- **Entwickeln** eines Betriebssystems *von der Pike auf*
  - OOSTuBS / MPStuBS (neu!) Lehrbetriebssysteme
  - **Praktische** Erfahrungen im Betriebssystembau machen
- **Verstehen** der technologischen Hardware-Grundlagen
  - PC-Technologie verstehen und einschätzen können
  - Schwerpunkt: Intel x86 / IA-32

 dl Betriebssysteme (VL 14 | WS 14) 14 Zusammenfassung und Ausblick – Ziele und Zielerreichung 14–2

## Was wir gemacht haben

Drei inhaltliche Schwerpunkte!

- VL<sub>1</sub> **Einführung**
- VL<sub>2</sub> **BS-Entwicklung**
- VL<sub>3</sub> **IRQs (Hardware)**
- VL<sub>4</sub> **IRQs (Software)**
- VL<sub>5</sub> **IRQs (Synchronisation)**
- VL<sub>6</sub> **Intel IA-32**
- VL<sub>7</sub> **Koroutinen und Fäden**
- VL<sub>8</sub> **Scheduling**
- VL<sub>9</sub> **BS-Architekturen I**
- VL<sub>10</sub> **Fadensynchronisation**
- VL<sub>11</sub> **BS-Architekturen II**
- VL<sub>12</sub> **Gerätetreiber**
- VL<sub>13</sub> **IPC**

 dl Betriebssysteme (VL 14 | WS 14) 14 Zusammenfassung und Ausblick – Ziele und Zielerreichung 14–4

# Was wir gemacht haben

Drei inhaltliche Schwerpunkte!

## 1. Ein Streifzug durch die PC-Architektur

- VL<sub>1</sub> Einführung
- VL<sub>2</sub> BS-Entwicklung
- VL<sub>3</sub> IRQs (Hardware)
- VL<sub>4</sub> IRQs (Software)
- VL<sub>5</sub> IRQs (Synchronisation)
- VL<sub>6</sub> Intel IA-32
- VL<sub>7</sub> Koroutinen und Fäden
- VL<sub>8</sub> Scheduling
- VL<sub>9</sub> BS-Architekturen I
- VL<sub>10</sub> Fadensynchronisation
- VL<sub>11</sub> BS-Architekturen II
- VL<sub>12</sub> Gerätetreiber
- VL<sub>13</sub> IPC



# Was wir gemacht haben

Drei inhaltliche Schwerpunkte!

## 1. Ein Streifzug durch die PC-Architektur

### Local APICs - Register

### IA-32: Adressierungsarten

- Effektive Adressen (EA) werden nach einem einfachen Schema gebildet
- alle Vielseitregister können dabei gleichzeitig verwendet werden

$$EA = \text{Basis-Reg.} + (\text{Index-Reg.} * \text{Scale}) + \text{Displacement}$$

Beispiel: MOV EAX, Feld[ESI \* 4]

- Lesen aus Feld mit 4 Byte großen Elementen und ESI als Index

dl Betriebssysteme (VL 6 | WS 14) 6 IA-32 - Die 32-Bit Intel-Architektur 6-19



# Was wir gemacht haben

Drei inhaltliche Schwerpunkte!

## 2. Kontrollflüsse und ihre Interaktionen

- VL<sub>1</sub> Einführung
- VL<sub>2</sub> BS-Entwicklung
- VL<sub>3</sub> IRQs (Hardware)
- VL<sub>4</sub> IRQs (Software)
- VL<sub>5</sub> IRQs (Synchronisation)
- VL<sub>6</sub> Intel IA-32
- VL<sub>7</sub> Koroutinen und Fäden
- VL<sub>8</sub> Scheduling
- VL<sub>9</sub> BS-Architekturen I
- VL<sub>10</sub> Fadensynchronisation
- VL<sub>11</sub> BS-Architekturen II
- VL<sub>12</sub> Gerätetreiber
- VL<sub>13</sub> IPC



# Was wir gemacht haben

Drei inhaltliche Schwerpunkte!

## 2. Kontrollflüsse und ihre Interaktionen

### Prioritätsebenenmodell

- Kontrollflüsse können die Ebene wechseln
  - Mit c<sub>li</sub> wechselt ein E<sub>0</sub>-Kontrollfluss explizit auf E<sub>1</sub>
    - er ist ab dann nicht mehr unterbrechbar
    - andere E<sub>1</sub>-Kontrollflüsse werden verzögert (← Sequentialisierung)
  - Mit s<sub>ti</sub> wechselt ein E<sub>1</sub>-Kontrollfluss explizit auf E<sub>0</sub>
    - er ist ab dann (wieder) unterbrechbar
    - anhängige E<sub>1</sub>-Kontrollflüsse „schlagen durch“ (← S)

### Erweitertes Prioritätsebenenmodell

- Kontrollflüsse auf E<sub>l</sub> werden
  - jederzeit unterbrochen durch Kontrollflüsse von E<sub>m</sub> (für m > l)
  - nie unterbrochen durch Kontrollflüsse von E<sub>k</sub> (für k ≤ l)
  - jederzeit verdrängt durch Kontrollflüsse von E<sub>l</sub> (für l = 0)

Kontrollflüsse der E<sub>0</sub> (Faden-ebene) sind **verdrängbar**.  
Für die Konsistenzsicherung auf dieser Ebene brauchen wir zusätzliche **Mechanismen** zur **Fadensynchronisation**.

dl Betriebssysteme (VL 5 | WS 14) 5 Unterbrechungen, Synchronisation – Prioritäts- 10-10



# Was wir gemacht haben

Drei inhaltliche Schwerpunkte!

## 2. Kontrollflüsse und ihre Interaktionen

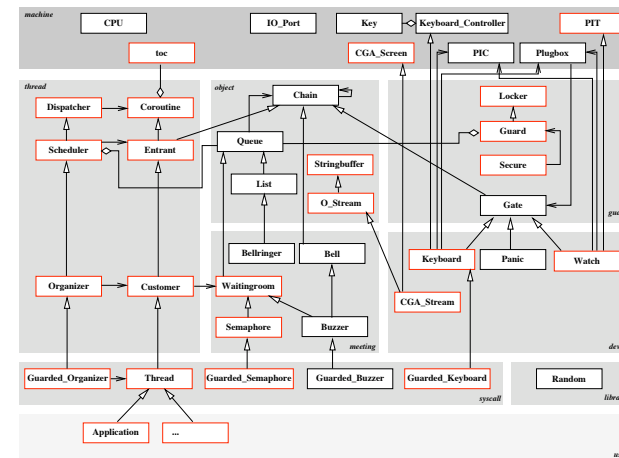
- VL<sub>1</sub> Einführung
- VL<sub>2</sub> BS-Entwicklung → Ü<sub>0</sub> Einführung
- VL<sub>3</sub> IRQs (Hardware) → Ü<sub>1</sub> Ein-/Ausgabe
- VL<sub>4</sub> IRQs (Software) → Ü<sub>2</sub> IRQ-Behandlung
- VL<sub>5</sub> IRQs (Synchronisation) → Ü<sub>3</sub> IRQ-Synchronisation
- VL<sub>6</sub> Intel IA-32
- VL<sub>7</sub> Koroutinen und Fäden → Ü<sub>4</sub> Fadenumschaltung
- VL<sub>8</sub> Scheduling → Ü<sub>5</sub> Zeitscheiben-Scheduling
- VL<sub>9</sub> BS-Architekturen I
- VL<sub>10</sub> Fadensynchronisation → Ü<sub>6</sub> Fadensynchronisation
- VL<sub>11</sub> BS-Architekturen II
- VL<sub>12</sub> Gerätetreiber
- VL<sub>13</sub> IPC → Ü<sub>7</sub> "Eine Anwendung" (opt.)



# Was wir gemacht haben

Drei inhaltliche Schwerpunkte!

## 2. Kontrollflüsse und ihre Interaktionen



# Was wir gemacht haben

Drei inhaltliche Schwerpunkte!

## 3. BS-Konzept allgemein und am Beispiel (Windows/Linux)

- VL<sub>1</sub> Einführung
- VL<sub>2</sub> BS-Entwicklung
- VL<sub>3</sub> IRQs (Hardware)
- VL<sub>4</sub> IRQs (Software)
- VL<sub>5</sub> IRQs (Synchronisation)
- VL<sub>6</sub> Intel IA-32
- VL<sub>7</sub> Koroutinen und Fäden
- VL<sub>8</sub> Scheduling
- VL<sub>9</sub> BS-Architekturen I
- VL<sub>10</sub> Fadensynchronisation
- VL<sub>11</sub> BS-Architekturen II
- VL<sub>12</sub> Gerätetreiber
- VL<sub>13</sub> IPC



# Was wir gemacht haben

Drei inhaltliche Schwerpunkte!

## 3. BS-Konzept allgemein und am Beispiel (Windows/Linux)

### Completely Fair Scheduler (CFS)

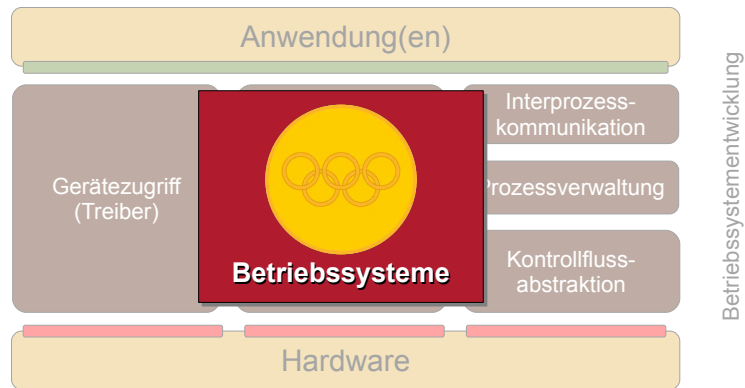
### Windows – typischer E/A-Ablauf

- über das Dateisystem wird das Dateisystem und der Treiber gefunden
- Daten an bestimmten Byte-Offset in Datei schreiben
- Position auf Datenträger berechnen
- E/A Auftrag weitergeben
- Daten an best. Byte Offset auf Datenträger schreiben
- Position in Plattenr. und Offset umrechnen
- E/A Auftrag weitergeben
- Daten an best. Byte Offset auf Platte 2 schreiben
- Phys. Block berechnen und Operation initiieren

○ Rückkehr zum Anwendungsprozess!



## Zusammen eine ganze Menge!



## Realitätscheck: MPStuBS ↔ „richtiges BS“

### Es fehlt noch eine ganze Menge!

- Adressraumverwaltung und Prozesskonzept ~ [BST]
- Dateisystem und Programmloader
- Netzwerk und TCP/IP
- ...



## Realitätscheck: MPStuBS ↔ „richtiges BS“

### Es fehlt noch eine ganze Menge!

- Adressraumverwaltung und Prozesskonzept ~ [BST]
- Dateisystem und Programmloader
- Netzwerk und TCP/IP
- ...

### Beispiel Linux [13]

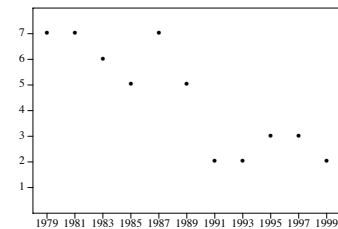
- Aug 91 Linux 0.01: bash, Dateisystem
- Jan 92 Linux 0.12: Virtueller Speicher (Paging)
- Mär 92 Linux 0.95: X-Windows, Unix Domain Sockets  
(jetzt fehlte nur noch Netzwerk!)
- Mär 94 Linux 1.00: **Netzwerk und TCP/IP**



## Betriebssysteme → ausgeforscht?

### „Systems Software Research is Irrelevant“ [8]

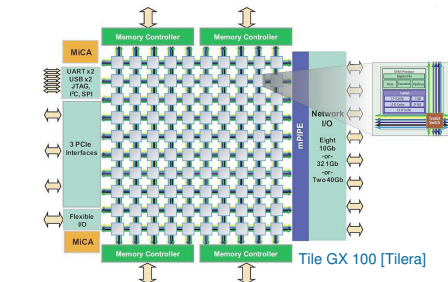
- Urgestein Robert Pike (2000), einer der Entwickler von UNIX, Inferno [4], Plan 9 [9] und UTF-8 (zur Zeit bei Google beschäftigt):
- Where is the innovation? ~ Microsoft, mostly
  - Every other „new“ OS ends up being UNIX
  - Linux? ~ Just another copy of the same old stuff
  - ...



New Operating Systems at SOS [8]

Aber dann...

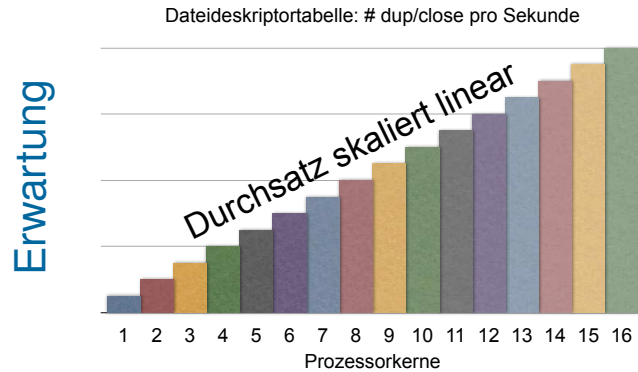
*The Multicore Challenge!*



## Fallstudie: Dateideskriptortabelle in Linux

- Boyd-Wickizer u. a. (OSDI 2008) [2]
  - Linux 2.6.25 auf 16-Kern AMD Opteron, 1–16 Kerne in Gebrauch
  - Pro Kern ein Faden, der Dateideskriptoren anfordert und freigibt:
 

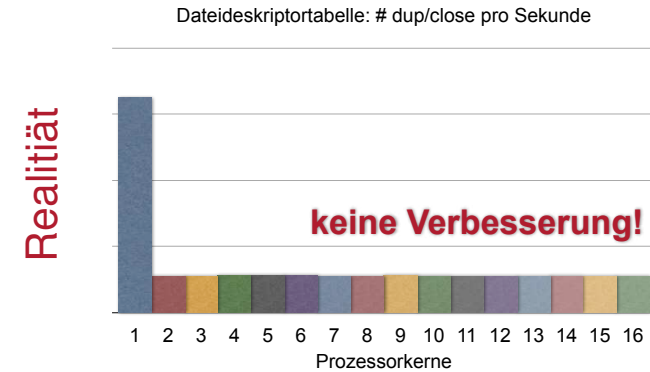
```
int f = open(...); while(1){ close( dup( f ) ); }
```



## Fallstudie: Dateideskriptortabelle in Linux

- Boyd-Wickizer u. a. (OSDI 2008) [2]
  - Linux 2.6.25 auf 16-Kern AMD Opteron, 1–16 Kerne in Gebrauch
  - Pro Kern ein Faden, der Dateideskriptoren anfordert und freigibt:
 

```
int f = open(...); while(1){ close( dup( f ) ); }
```



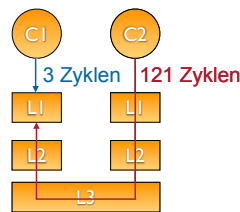
## Fallstudie: Dateideskriptortabelle in Linux

- Boyd-Wickizer u. a. (OSDI 2008) [2]
  - Linux 2.6.25 auf 16-Kern AMD Opteron, 1–16 Kerne in Gebrauch
  - Pro Kern ein Faden, der Dateideskriptoren anfordert und freigibt:
 

```
int f = open(...); while(1){ close( dup( f ) ); }
```
- Ergebnis: Schon ab **2 Kernen sinkt** der Gesamtdurchsatz
  - Grobgranulares *Locking*  $\rightsquigarrow$  *false sharing*  $\rightsquigarrow$  keine Skalierbarkeit
  - Geteilte Datenstruktur  $\rightsquigarrow$  *cache trashing*  $\rightsquigarrow$  Durchsatzabfall

```
fd_alloc () {
    lock(fd_table);
    fd = get_free_fd();
    set_fd_used(fd);
    fix_smallest_fd();
    unlock(fd_table);
}
```

1. *false sharing*



2. *cache trashing*



## Fallstudie: Dateideskriptortabelle in Linux

- Boyd-Wickizer u. a. (OSDI 2008) [2]
  - Linux 2.6.25 auf 16-Kern AMD Opteron, 1–16 Kerne in Gebrauch
  - Pro Kern ein Faden, der Dateideskriptoren anfordert und freigibt:
 

```
int f = open(...); while(1){ close( dup( f ) ); }
```
- Ergebnis: Schon ab **2 Kernen sinkt** der Gesamtdurchsatz
  - Grobgranulares *Locking*  $\rightsquigarrow$  *false sharing*  $\rightsquigarrow$  keine Skalierbarkeit
  - Geteilte Datenstruktur  $\rightsquigarrow$  *cache trashing*  $\rightsquigarrow$  Durchsatzabfall

**Multicore: POSIX ( $\rightarrow$  UNIX) considered harmful!**

„This problem is not specific to Linux, but is **due to POSIX semantics**, which require that a new file descriptor be visible to all of a process's threads even if only one thread uses it.” [2]



## Folgerung: Wir brauchen neue Entwurfsansätze!

- **Corey** MIT, OSDI 2008, Exokern-artig: [2]
  - *Sharing* unter die Kontrolle der Applikation stellen
  - Datenstrukturen (im Normalfall) nur von einem Kern aus bearbeiten
  - Anwendungen müssen angepasst werden
- **Barrelfish** ETH/MSR, SOSP 2009, Mikrokern-artig: [1]
  - BS als verteiltes System von Kernen verstehen und organisieren
  - kein implizites *Sharing*, Kommunikation nur über Nachrichten
- **Factored OS (fos)** MIT, 2009, Mikrokern-artig: [14]
  - BS für 100 bis 1000 Kerne  $\leadsto$  *time sharing* wird zu *space sharing*
  - Letztlich ähnlicher Ansatz wie Barrelfish
- **TxOS** UT, SOSP 2009, Monolith (Linux): [10]
  - Konkurrenz zulassen durch *transactional syscalls* (statt *Locks*)
  - Anwendungen müssen angepasst werden



## ... oder doch nicht?

- Boyd-Wickizer u. a. (OSDI 2010) [3]
  - „An Analysis of Linux Scalability to Many Cores“
  - Skalierbarkeit von Linux 2.6.35-rc5 auf 48-Kern AMD Opteron
- Ansatz: *run – analyze – fix*
  - *run*: sieben „systemintensive“ Anwendungen
    - Exim, memcached, Apache, PostgreSQL, gmake, Psearchy, MapReduce
  - *analyze*: gezielte Identifizierung von Flaschenhälsen
    - im Linux-Kern selber (16)
    - im Entwurf der Anwendung
    - durch die ungeschickte Verwendung der Systemschnittstelle
  - *fix*: Verbesserung, überwiegend durch Standardtechniken der parallelen Programmierung ( $\leadsto$  [PFP])



## ... oder doch nicht?

**Ergebnis:** Alles nicht so schlimm. . .

„We find that we can remove most kernel bottlenecks that the applications stress by modifying the applications or kernel slightly. [ . . . ] the results suggest that **traditional kernel designs may be compatible with achieving scalability** on multicore computers.“ [3]

Fazit

# Es bleibt spannend!

Systementwurf für Skalierbarkeit  $\leadsto$  [CC] (WS 2015).



## Evaluation



## Evaluationsergebnisse

Hauptfragen zu Lehrveranstaltung und Dozent (= LQI)



Weitere Fragen zu Lehrveranstaltung und Dozent



### Vergleich mit den Vorjahren

■ WS 14:	n=18	(53%)	mw=1.17
■ WS 13:	n=30	(52%)	mw=1.17
■ WS 12:	n=18	(51%)	mw=1.11
■ WS 11:	n=17	(53%)	mw=1.30
■ WS 10:	n=9	(29%)	mw=1.42
■ WS 09:	n=19	(100%)	mw=1.34
■ WS 08:	n=7	(27%)	mw=1.41
■ WS 07:	n=16	(50%)	mw=1.39

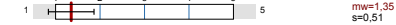


## Evaluationsergebnisse: Übung

Hauptfragen zu Lehrveranstaltung und Übungsleiter (= LQI)



Weitere Fragen zu Lehrveranstaltung und Übungsleiter



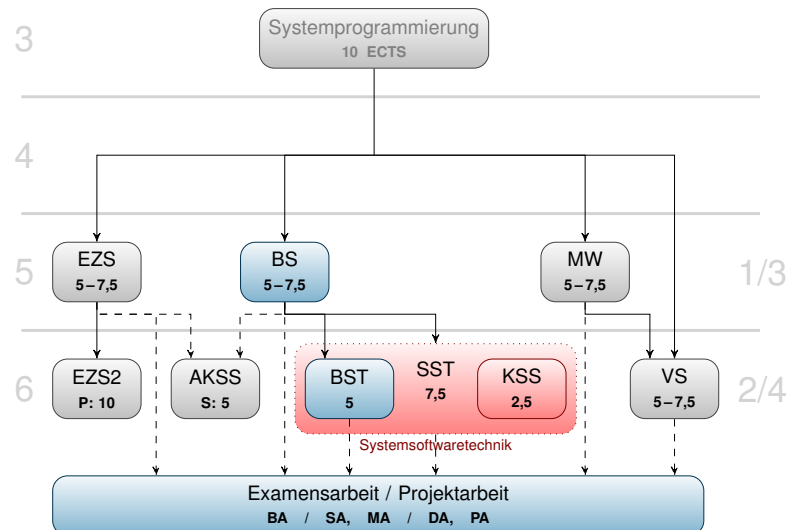
### Vergleich mit den Vorjahren

■ WS 14:	n=320	(62%)	mw=1.24
■ WS 13:	n=30	(52%)	mw=1.23
■ WS 12:	n=23	(52%)	mw=1.55
■ WS 11:	n=18	(40%)	mw=1.56
■ WS 10:	n=9	(29%)	mw=1.59
■ WS 09:	n=9	(41%)	mw=1.49
■ WS 08:	n=7	(30%)	mw=1.41
■ WS 07:	n=23	(52%)	mw=1.37



## Wie geht es weiter?

(Bachelor/Master)



## Ausblick: Betriebssystemtechnik (BST)

### Lernziele

#### Vorlesung

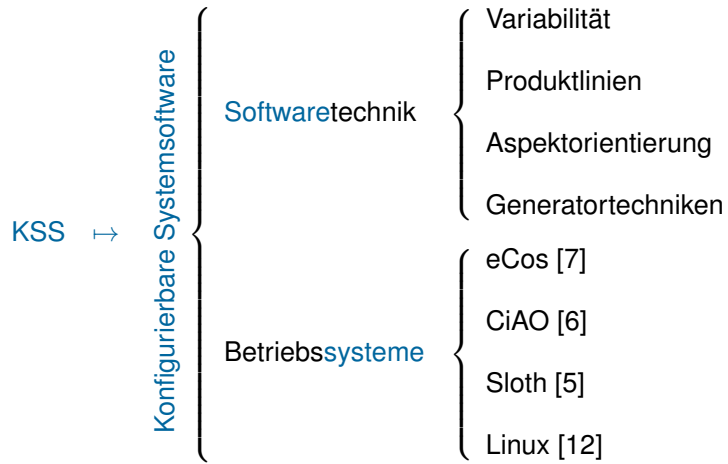
- Wissen zu Adressraumkonzepten von Betriebssystemen vertiefen
- Verstehen über (logische) Adressräume festigen
  - inhaltliches Begreifen verschiedener Facetten von Adressräumen
  - intellektuelle Erfassung des Zusammenhangs, in dem Adressräume stehen

#### Übung ~> mikrokern-ähnliches Betriebssystem

- Anwenden ausgewählter Vorlesungsinhalte für OOSTuBS
- Analyse der Anforderungen an und Gegebenheiten von OOSTuBS
- Synthese von Adressraumabstraktionen und OOSTuBS
- Evaluation des erweiterten OOSTuBS: Vorher-nachher-Vergleich



# Hinter der Kulisse: KSS in aller Kürze...



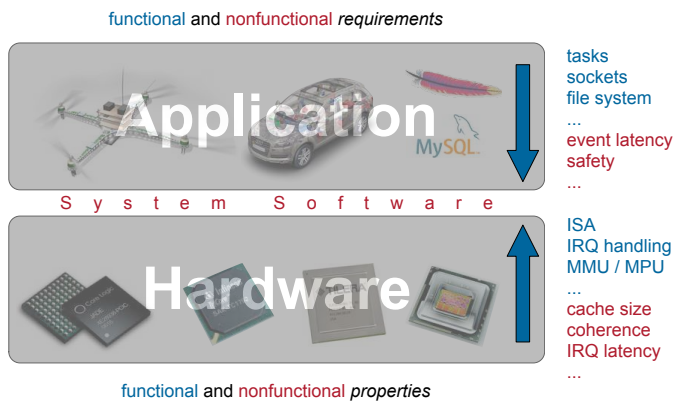
# Ausblick: Konfigurierbare Systemsoftware (KSS)

## Motivation: Special-Purpose Systems



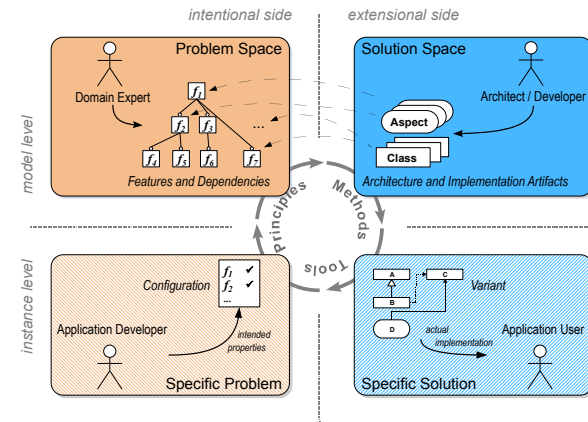
# Ausblick: Konfigurierbare Systemsoftware (KSS)

## “Between a Rock and a Hard Place”



# Ausblick: Konfigurierbare Systemsoftware (KSS)

## Configurable Software → Product Line





## The State of the Art: eCos

### The embedded Configurable OS

- operating system for embedded applications
- open source, maintained by eCosCentric
- broadly accepted real-world system

### More than 750 configuration options

- feature-based selection
- preprocessor-based implementation

➔ This has a **severe impact on the code!**



## eCos – Implementation of Configurability

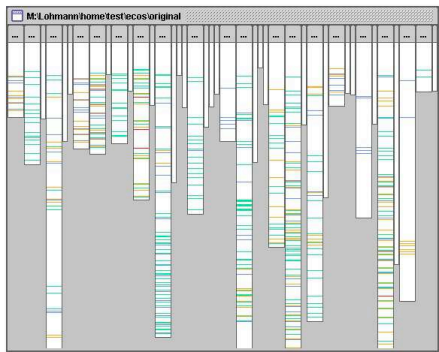
```

Cyg_Mutex::Cyg_Mutex() {
    CYG_REPORT_FUNCTION();
    locked = false;
    owner = NULL;
    #if defined(CYSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_DEFAULT) && \
    defined(CYSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_DYNAMIC)
    #ifndef CYSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_DEFAULT_INHERIT
    protocol = INHERIT;
    #endif
    #endif
    #ifdef CYSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_DEFAULT_CEILING
    protocol = CEILING;
    ceiling = CYSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_DEFAULT_PRIORITY;
    #endif
    #ifndef CYSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_DEFAULT_NONE
    protocol = NONE;
    #endif
    #else // not (DYNAMIC and CEILING)
    #ifdef CYSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_DEFAULT_PRIORITY
    // if there is a default priority ceiling defined, use that to initialize
    // the ceiling.
    ceiling = CYSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_DEFAULT_PRIORITY;
    #else
    // Otherwise set it to zero.
    ceiling = 0;
    #endif
    #endif
    #endif // DYNAMIC and DEFAULT defined
    CYG_REPORT_RETURN();
}
    
```

- Mutex options:
- PROTOCOL
  - CEILING
  - INHERIT
  - DYNAMIC

Kernel policies: Tracing Instrumentation Synchronization

## Issue: Crosscutting Concerns

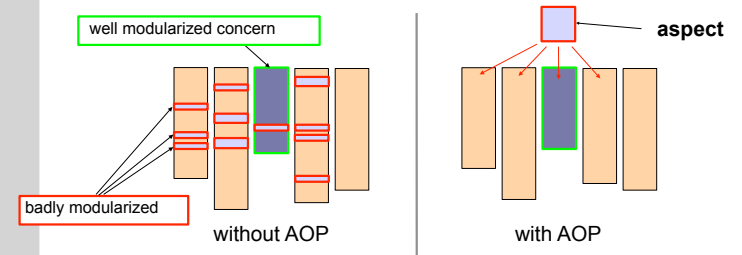


- Mutex options:
- PROTOCOL
  - CEILING
  - INHERIT
  - DYNAMIC

Kernel policies: Tracing Instrumentation Synchronization

## Solution Idea: Aspect-Oriented Programming

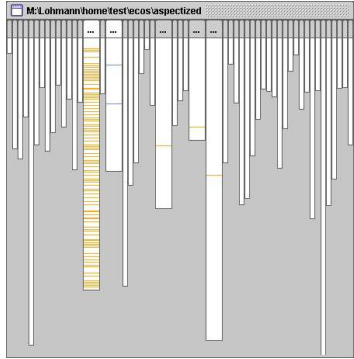
AOP provides language means to encapsulate crosscutting and scattered concerns



Kernel policies: Tracing Instrumentation Synchronization

### Qualitative Results: eCos → AspeCos

(EuroSys '06)



Kernel policies: Tracing Instrumentation Synchronization

14

### Example: Synchronization in AspeCos

```

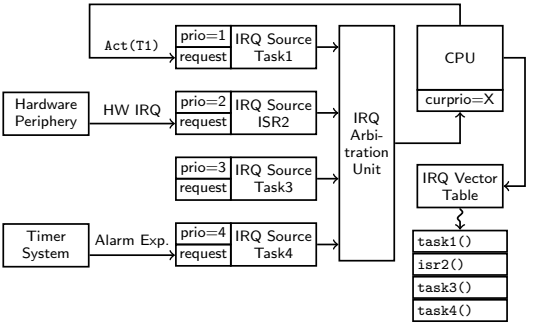
aspect int_sync {
  pointcut sync() = execution(...) // kernel calls to sync
    || construction(...)
    || destruction(...);
  where

  // advise kernel code to invoke lock() and unlock()
  advice sync() : before() {
    Cyg_Scheduler::lock();
  }
  advice sync() : after() {
    Cyg_Scheduler::unlock();
  }
  what

  // In eCos, a new thread always starts with a lock value of 0
  advice execution(
    "%Cyg_HardwareThread::thread_entry(...)" ) : before() {
    Cyg_Scheduler::zero_sched_lock();
  }
  ...
};
    
```

18

### SLOTH Design

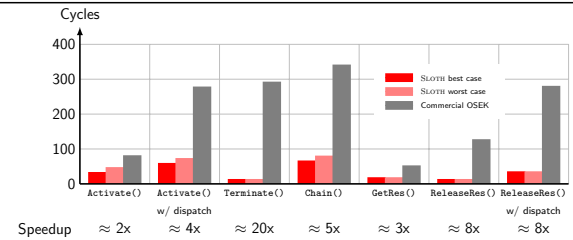


■ Platform must support IR priorities and software IR triggering

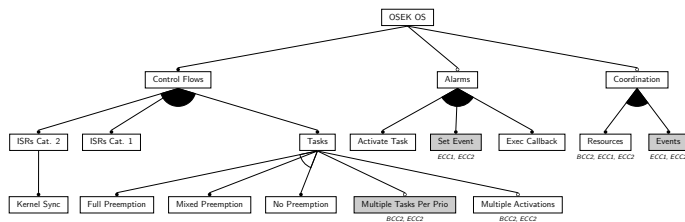
Wanja Hofer SLOTH: Threads as Interrupts (RTSS 2009)

6

### SLOTH [RTSS '09]



Speedup ≈ 2x ≈ 4x ≈ 20x ≈ 5x ≈ 3x ≈ 8x ≈ 8x

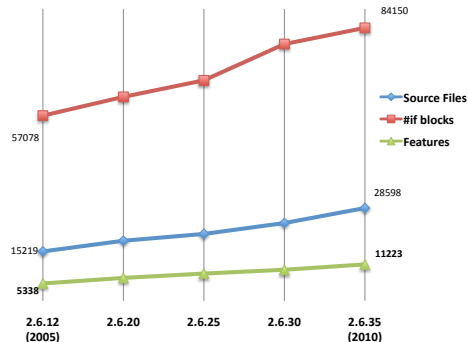


19

## Configurability in the Large: Linux

More than **11,000** configuration options!

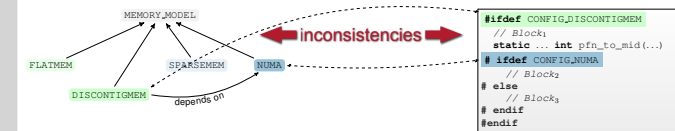
- 85,000 **#ifdef blocks**, sprinkled over 29,000 **source files**
- numbers have **doubled** within the last five years!



## Configurability in the Large: Linux

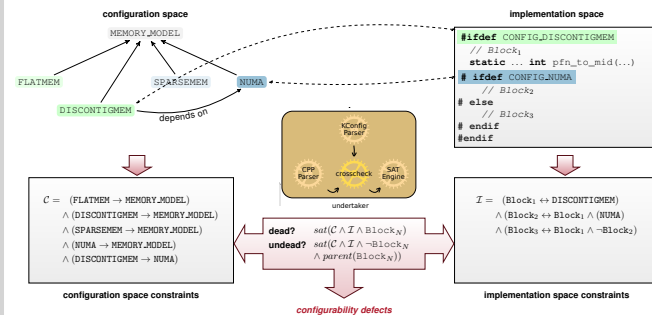
More than **11,000** configuration options!

- 85,000 **#ifdef blocks**, sprinkled over 29,000 **source files**
- numbers have **doubled** within the last five years!



## The Undertaker

[EuroSys '11]



- found **1,776** defects (and that is just a lower bound!)
  - proposed fix for 364 (including 20 new bugs)
  - 123 patches submitted (49 merged into Linux-Tree)
  - removed 5,129 lines of *unnecessary* #ifdef-code
- tool suite now published as open-source project

## AKSS: Rekonfigurierbare Systemsoftware

- Masterseminar *Ausgewählte Kapitel der Systemsoftware*
  - 4 SWS bzw. 5 ECTS
- Das Seminar beleuchtet das breite Spektrum **dynamisch rekonfigurierbarer** Systemsoftware:
  - Dynamisches Nachladen von Funktionalität, z. B. Kernel-Module
  - „Hot-Patching“: Code-Änderungen zur Laufzeit, z. B. Ksplice
  - Austauschen funktionaler Einheiten im laufenden Betrieb, „Operation am offenen Herzen“
- Aktuelle Forschungsarbeiten aus dem *Systems*-Bereich
- Themenliste demnächst™ online: [http://www4.cs.fau.de/Lehre/SS14/MS\\_AKSS](http://www4.cs.fau.de/Lehre/SS14/MS_AKSS)

## Examensarbeiten am LS4

### Zur Zeit im Angebot:

- Bachelorarbeiten
- Masterarbeiten
- Projektarbeiten

<http://www4.informatik.uni-erlangen.de/DE/Theses/>

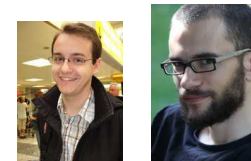


## Das war's :-)

Das Lehrstuhl 4 BS-Team  
wünscht **erfolgreiche** und **erhol-**  
**same** "Semesterferien"



... und ein Wiedersehen  
im Sommersemester 2015!



## Referenzen

- [1] Andrew Baumann, Paul Barham, Pierre-Evariste Dagand u. a. „The multikernel: a new OS architecture for scalable multicore systems“. In: *Proceedings of the 22nd ACM Symposium on Operating Systems Principles (SOSP '09)*. (Big Sky, Montana, USA). New York, NY, USA: ACM Press, 2009, S. 29–44. ISBN: 978-1-60558-752-3. DOI: 10.1145/1629575.1629579.
- [2] Silas Boyd-Wickizer, Haibo Chen, Rong Chen u. a. „Corey: An Operating System for Many Cores“. In: *8th Symposium on Operating System Design and Implementation (OSDI '08)*. (San Diego, CA, USA). Berkeley, CA, USA: USENIX Association, 2008, S. 43–57.
- [3] Silas Boyd-Wickizer, Austin T. Clements, Yandong Mao u. a. „An Analysis of Linux Scalability to Many Cores“. In: *9th Symposium on Operating System Design and Implementation (OSDI '10)*. (Vancouver, Canada). Berkeley, CA, USA, 2010.
- [4] Sean Dorward, Rob Pike, Dave Presotto u. a. „The Inferno Operating System“. In: *Bell Labs Technical Journal 2.1* (1997). URL: <http://www.vitanuova.com/inferno/papers/bltj.html>.
- [5] Wanja Hofer, Daniel Lohmann, Fabian Scheler u. a. „Sloth: Threads as Interrupts“. In: *Proceedings of the 30th IEEE International Symposium on Real-Time Systems (RTSS '09)*. (Washington, D.C., USA, 1.–4. Dez. 2009). IEEE Computer Society Press, Dez. 2009, S. 204–213. ISBN: 978-0-7695-3875-4. DOI: 10.1109/RTSS.2009.18.



## Referenzen (Forts.)

- [6] Daniel Lohmann, Wanja Hofer, Wolfgang Schröder-Preikschat u. a. „CiAO: An Aspect-Oriented Operating-System Family for Resource-Constrained Embedded Systems“. In: *Proceedings of the 2009 USENIX Annual Technical Conference*. Berkeley, CA, USA: USENIX Association, Juni 2009, S. 215–228. ISBN: 978-1-931971-68-3. URL: [http://www.usenix.org/event/usenix09/tech/full\\_papers/lohmann/lohmann.pdf](http://www.usenix.org/event/usenix09/tech/full_papers/lohmann/lohmann.pdf).
- [7] Daniel Lohmann, Fabian Scheler, Reinhard Tartler u. a. „A Quantitative Analysis of Aspects in the eCos Kernel“. In: *Proceedings of the ACM SIGOPS/EuroSys European Conference on Computer Systems 2006 (EuroSys '06)*. (Leuven, Belgium). Hrsg. von Yolande Berbers und Willy Zwaenepoel. New York, NY, USA: ACM Press, Apr. 2006, S. 191–204. ISBN: 1-59593-322-0. DOI: 10.1145/1218063.1217954.
- [PFP] Michael Philippsen und Norbert Oster. *Parallele und Funktionale Programmierung*. Vorlesung mit Übung. Friedrich-Alexander-Universität Erlangen-Nürnberg, Lehrstuhl für Informatik 2, 2009 (jährlich). URL: <http://www2.informatik.uni-erlangen.de/teaching/WS2009/PFP/index.html>.
- [8] Rob Pike. *Systems Software Research is Irrelevant*. Talk. CS Colloquium, Columbia University. URL: <http://herpolhode.com/rob/utah2000.pdf> (besucht am 09. 12. 2010).



## Referenzen (Forts.)

- [9] Rob Pike, Dave Presotto, Sean Dorward u. a. „Plan 9 from Bell Labs“. In: *Computing Systems* 8.3 (1995), S. 221–254.
- [10] Donald E. Porter, Owen S. Hofmann, Christopher J. Rossbach u. a. „Operating System Transactions“. In: *Proceedings of the 22nd ACM Symposium on Operating Systems Principles (SOSP '09)*. (Big Sky, Montana, USA). New York, NY, USA: ACM Press, 2009, S. 161–176. ISBN: 978-1-60558-752-3. DOI: 10.1145/1629575.1629591.
- [11] *Proceedings of the 22nd ACM Symposium on Operating Systems Principles (SOSP '09)*. (Big Sky, Montana, USA). New York, NY, USA: ACM Press, 2009. ISBN: 978-1-60558-752-3.
- [BST] Wolfgang Schröder-Preikschat. *Betriebssystemtechnik*. Vorlesung mit Übung. Friedrich-Alexander-Universität Erlangen-Nürnberg, Lehrstuhl für Informatik 4, 2013 (jährlich). URL: [http://www4.informatik.uni-erlangen.de/Lehre/SS13/V\\_BST](http://www4.informatik.uni-erlangen.de/Lehre/SS13/V_BST).
- [CC] Wolfgang Schröder-Preikschat. *Concurrent Systems*. Vorlesung mit Übung. Friedrich-Alexander-Universität Erlangen-Nürnberg, Lehrstuhl für Informatik 4, 2014 (jährlich). URL: [http://www4.informatik.uni-erlangen.de/Lehre/WS14/V\\_CC](http://www4.informatik.uni-erlangen.de/Lehre/WS14/V_CC).



## Referenzen (Forts.)

- [12] Reinhard Tartler, Daniel Lohmann, Julio Sincero u. a. „Feature Consistency in Compile-Time-Configurable System Software: Facing the Linux 10,000 Feature Problem“. In: *Proceedings of the ACM SIGOPS/EuroSys European Conference on Computer Systems 2011 (EuroSys '11)*. (Salzburg, Austria). Hrsg. von Christoph M. Kirsch und Gernot Heiser. New York, NY, USA: ACM Press, Apr. 2011, S. 47–60. ISBN: 978-1-4503-0634-8. DOI: 10.1145/1966445.1966451.
- [13] Linus Torvalds und David Diamond. *Just for Fun: The Story of an Accidental Revolutionary*. HarperCollins, 2001. ISBN: 978-0066620725.
- [14] David Wentzlaff und Anant Agarwal. „Factored operating systems (fos): the case for a scalable operating system for multicores“. In: *ACM SIGOPS Operating Systems Review* 43 (2 Apr. 2009), S. 76–85. ISSN: 0163-5980. DOI: 10.1145/1531793.1531805.

