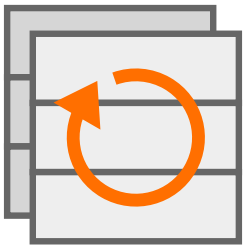


TUHH
Technische Universität Hamburg

Betriebssystembau (BSB)

VL 12 – Zusammenfassung und Ausblick

Christian Dietrich

Operating System Group

SS 22 – 12.7.2022

Interaktion während der Veranstaltung

TUHH
Technische Universität Hamburg

- BSB ist vom "Stil" her eine **interaktive Präsenzveranstaltung**
  - Wir wollen versuchen, dieses soweit wie möglich "online" zu retten

⇝ **Synchrones** Format – Fragen und Beteiligung ist erwünscht!

- Interaktion **während** der Veranstaltung
  - 1. „Melden“
  - 2. „Drankommen“
  - 3. Profit
- Interaktion **außerhalb** der Veranstaltung
  - Über das Stud.IP-Forum
  - **NEU**: EIM Mattermost Team: `https://communicating.tuhh.de/eim`

Aufzeichnung der Vorlesung

You are on mic!

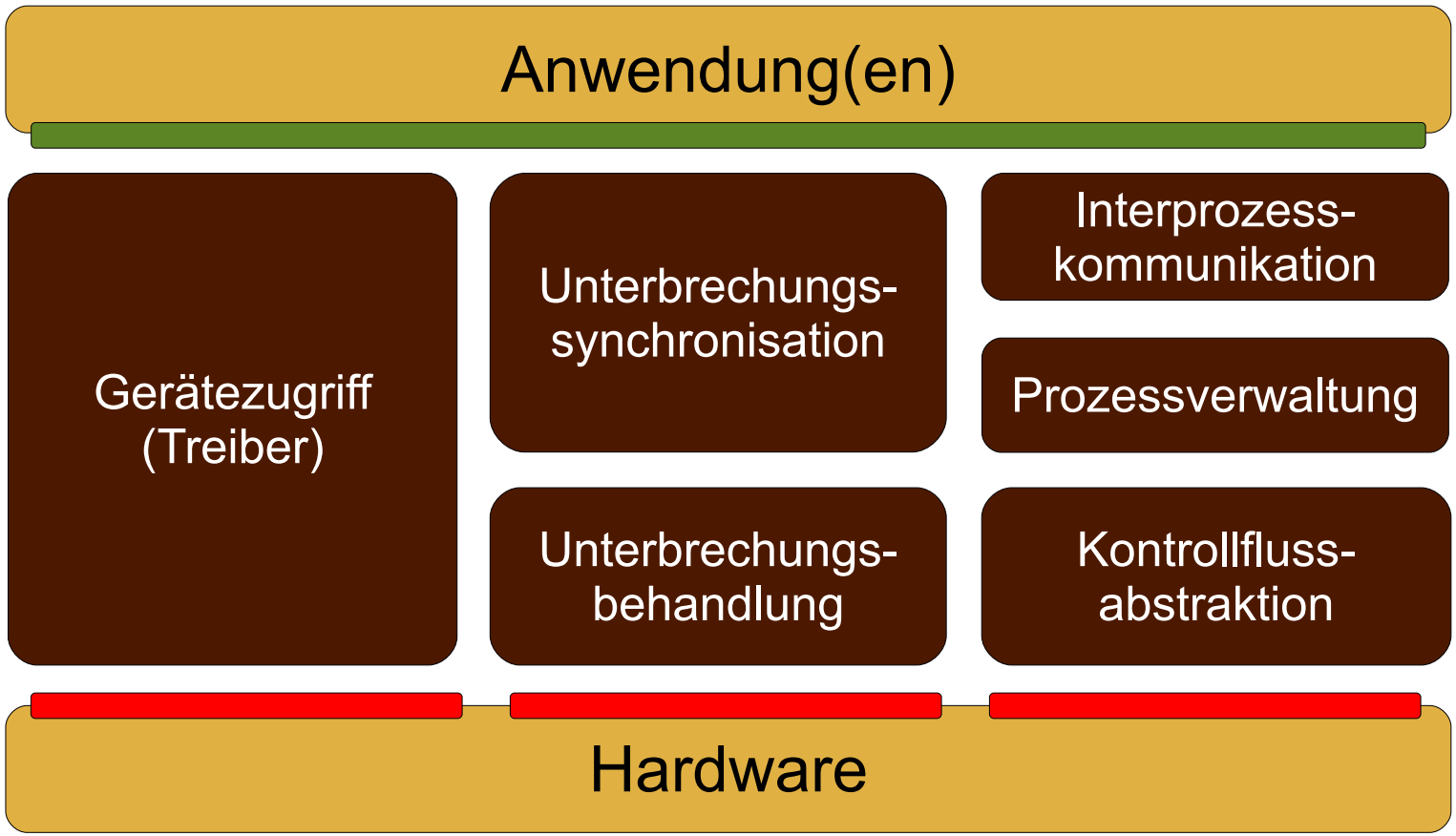
- Auf vielfachen Studierendenwunsch:
  Veranstaltung wird aufgezeichnet
  - Wird im Anschluss über Stud.IP verfügbar gemacht
  - ⇝ Geschlossene Nutzergruppe
- Aufgezeichnet wird
  - Screencast der BBB-Session **ohne den Chat (Klarnamen)**
  - Ihre Stimme bei Fragen und Anmerkungen
  - **Durch Aktivierung Ihres Mikrofons willigen Sie dazu ein!**
- Fragen können über direkte Nachricht an mich auch anonym gestellt werden

Lernziele

⇝VL 1

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

Überblick Vorlesungen



Was wir gemacht haben

Drei inhaltliche Schwerpunkte!

- VL_1 *Einführung*
- VL_2 *BS-Entwicklung*
- VL_3 *IRQs (Hardware)*
- VL_4 *IRQs (Software)*
- VL_5 *IRQs (Synchronisation)*
- VL_6 *Intel IA-32*
- VL_7 *Koroutinen und Fäden*
- VL_8 *Scheduling*
- VL_9 *Architekturen*
- VL_{10} *Fadensynchronisation*
- VL_{11} *Gerätetreiber*
- VL_{12} *Ausblick*

Was wir gemacht haben

Drei inhaltliche Schwerpunkte!

1. Ein Streifzug durch die PC-Architektur

- VL_1 *Einführung*
- VL_2 *BS-Entwicklung*
- **VL_3 *IRQs (Hardware)***
- VL_4 *IRQs (Software)*
- VL_5 *IRQs (Synchronisation)*
- **VL_6 *Intel IA-32***
- VL_7 *Koroutinen und Fäden*
- VL_8 *Scheduling*
- VL_9 *Architekturen*
- VL_{10} *Fadensynchronisation*
- VL_{11} *Gerätetreiber*
- VL_{12} *Ausblick*

Was wir gemacht haben

Drei inhaltliche Schwerpunkte!

1. Ein Streifzug durch die PC-Architektur

APIC Architetkur - Zusammenfassung

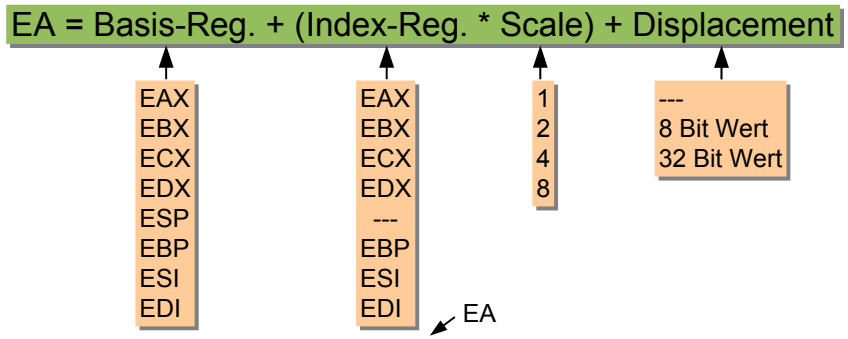
TUHH Technische Universität Hamburg

- flexible Verteilung an CPUs im x86 Multiprozessorsystem
  - fest, Gruppen, an die CPU mit der geringsten Priorität
  - Liegen mehrere IRQs an, so wird nach Vektornummer priorisiert
- Vektornummer 16-254 können frei zugeordnet werden
  - sollte (an sich) reichen, um „sharing“ zu vermeiden
- *Local* APIC erwartet explizites EOI
  - dafür muss die Software sorgen
- Mit APIC unterstützt x86 prinzipiell auch Priori
  - Systemsoftware muss jedoch entsprechend agieren (Unterbrechungen freigeben, evtl. *Task-Priority-Regis*

IA-32: Adressierungsarten

TUHH Technische Universität Hamburg

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



- Beispiel: MOV EAX, **Feld[ESI * 4]**
  - Lesen aus Feld mit 4 Byte großen Elementen und ESI als Index

Was wir gemacht haben

Drei inhaltliche Schwerpunkte!

2. Kontrollflüsse und ihre Interaktionen

- VL_1 *Einführung*
- VL_2 *BS-Entwicklung*
- **VL_3 *IRQs (Hardware)***
- **VL_4 *IRQs (Software)***
- **VL_5 *IRQs (Synchronisation)***
- VL_6 *Intel IA-32*
- **VL_7 *Koroutinen und Fäden***
- **VL_8 *Scheduling***
- VL_9 *Architekturen*
- **VL_{10} *Fadensynchronisation***
- VL_{11} *Gerätetreiber*
- VL_{12} *Ausblick*

Was wir gemacht haben

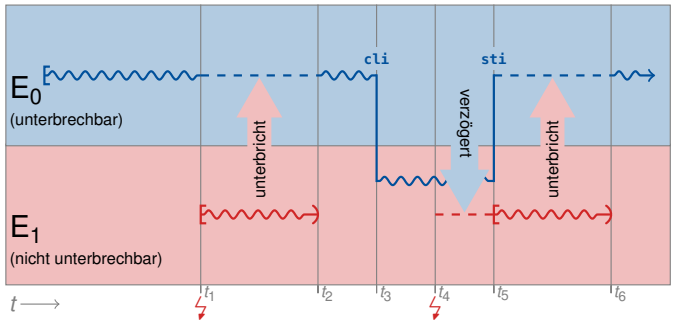
Drei inhaltliche Schwerpunkte!

2. Kontrollflüsse und ihre Interaktionen

Prioritätsebenenmodell

TUHH Technische Universität Hamburg

- Kontrollflüsse können die Ebene wechseln
  - Mit `cli` wechselt ein E_0-Kontrollfluss explizit auf E_1
    - er ist ab dann nicht mehr unterbrechbar
    - andere E_1-Kontrollflüsse werden verzögert (⟷ Sequentialisierung)
  - Mit `sti` wechselt ein E_1-Kontrollfluss explizit auf E_0
    - er ist ab dann (wieder) unterbrechbar
    - anhängige E_1-Kontrollflüsse „schlagen durch“ (⟷ S

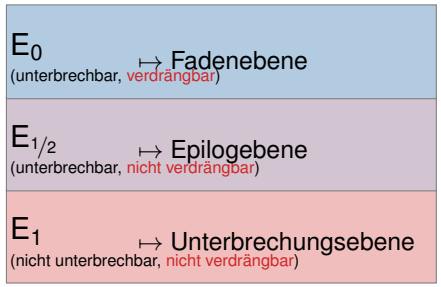


cd Betriebssystembau – 5 Unterbrechungen, Synchronisation

Erweitertes Prioritätsebenenmodell

TUHH Technische Universität Hamburg

- Kontrollflüsse auf E_l werden
  1. jederzeit unterbrochen durch Kontrollflüsse von E_m (für $m > l$)
  2. nie unterbrochen durch Kontrollflüsse von E_k (für $k \leq l$)
  3. sequentialisiert mit weiteren Kontrollflüssen von E_l (für $l > 0$)
  4. jederzeit verdrängt durch Kontrollflüsse von E_l (für $l = 0$)



Kontrollflüsse der E_0 (Fadenebene) sind **verdrängbar**.

Für die Konsistenzssicherung auf dieser Ebene brauchen wir zusätzliche **Mechanismen** zur **Fadensynchronisation**.

cd Betriebssystembau – 10 Fadensynchronisation 10 – 12

Was wir gemacht haben

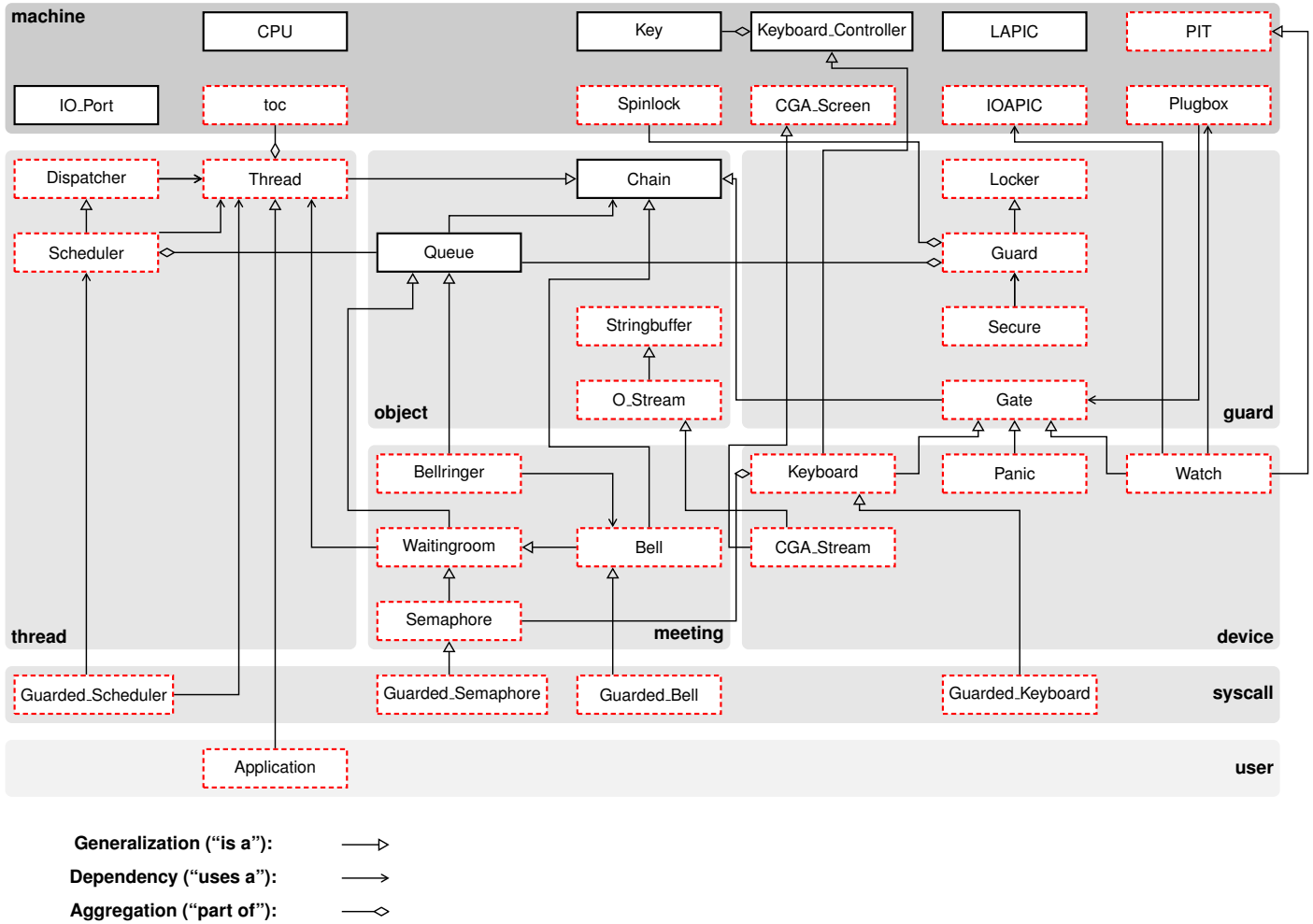
Drei inhaltliche Schwerpunkte!

2. Kontrollflüsse und ihre Interaktionen

Was wir gemacht haben

Drei inhaltliche Schwerpunkte!

2. Kontrollflüsse und ihre Interaktionen



Was wir gemacht haben

Drei inhaltliche Schwerpunkte!

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

- VL_1 *Einführung*
- **VL_2 *BS-Entwicklung***
- VL_3 *IRQs (Hardware)*
- VL_4 *IRQs (Software)*
- VL_5 *IRQs (Synchronisation)*
- VL_6 *Intel IA-32*
- VL_7 *Koroutinen und Fäden*
- **VL_8 *Scheduling***
- **VL_9 *Architekturen***
- **VL_{10} *Fadensynchronisation***
- **VL_{11} *Gerätetreiber***
- **VL_{12} *Ausblick***

Was wir gemacht haben

Drei inhaltliche Schwerpunkte!

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

Zusammenfassung: Scheduling

TUHH Technische Universität Hamburg

- *Threads* sind Koroutinen des Betriebssystems
  - BS hat Entzugsmechanismen
  - Strategie der Ablaufplanung wird als *Scheduling* bezeichnet
- *Scheduling* hat großen Einfluss auf die Performanz des Gesamtsystems, es legt fest, ...
  - welche Prozesse warten und welche voranschreiten
  - welche Betriebsmittel wie ausgelastet sind
- Es gibt verschiedenste Varianten des Schedul
  - nur wenige Unterschiede bei gängigen PC/Workstation-Bestriebsystemen
  - eventuell aber starke Unterschiede in andere Anwendungsdomänen

Windows – Treiberstruktur

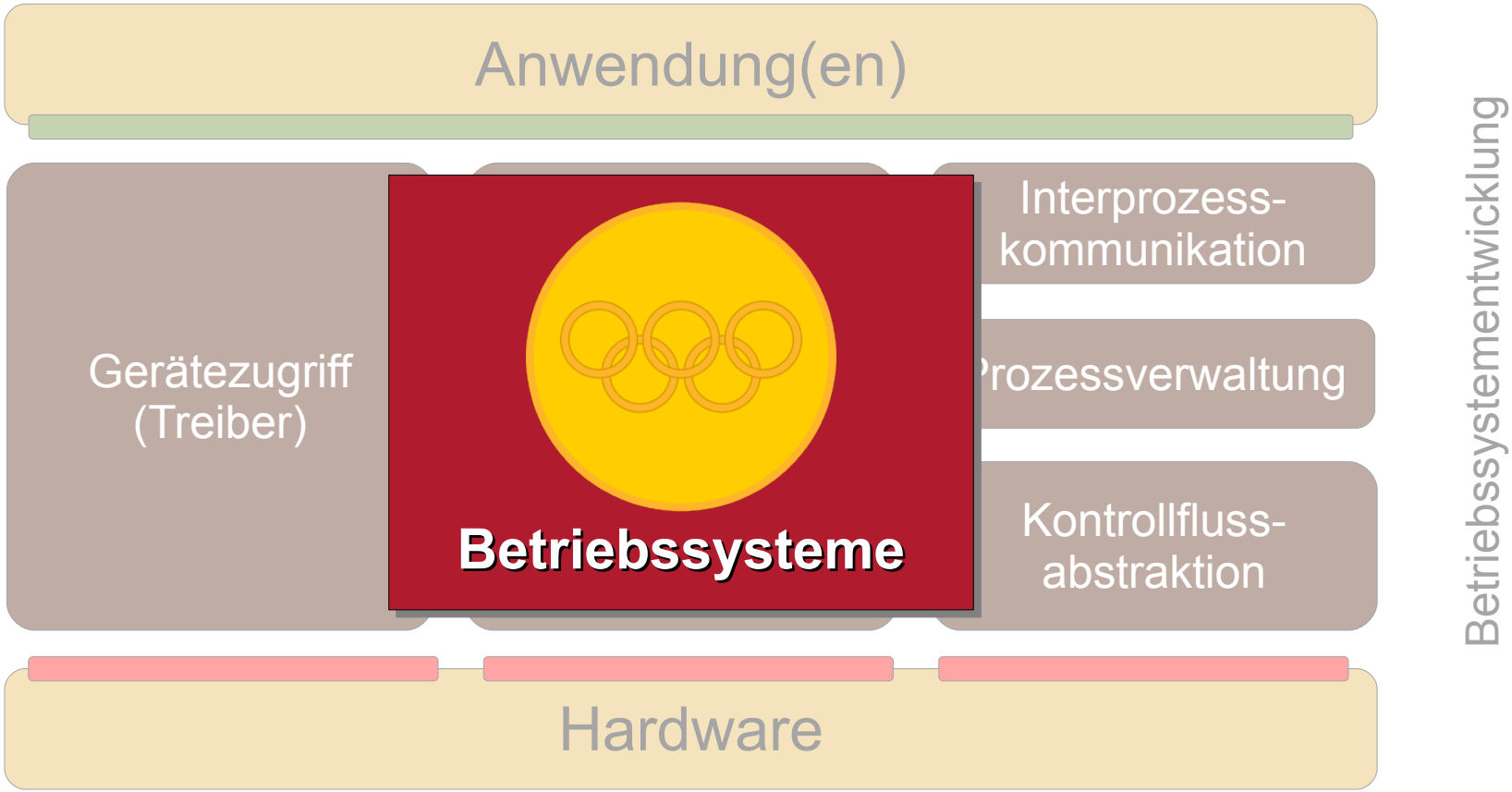
TUHH Technische Universität Hamburg

Das E/A-System steuert den Treiber mit Hilfe der ...

- Initialisierungsroutine/Entladeroutine
  - wird nach/vor dem Laden/Entladen des Treibers ausgeführt
- Routine zum Hinzufügen von Geräten
  - PnP Manager hat ein neues Gerät für den Treiber
- "Verteilerroutinen"
  - Öffnen, Schließen, Lesen, Schreiben und gerätespezifische Oper.
- Interrupt Service Routine
  - wird von der zentralen Interrupt-Verteilungsroutine aufgerufen
- DPC-Routine
  - "Epilog" der Unterbrechungsbehandlung
- E/A-Komplettierungs- und -Abbruchroutine
  - Informationen über den Ausgang weitergeleiteter E/A-Aufträge

...

Zusammen eine ganze Menge!



Realitätscheck: MPStuBS $\leftrightarrow$ „richtiges BS“

Es fehlt noch eine ganze Menge!

- Adressraumverwaltung und Prozesskonzept $\rightsquigarrow$ BST im WS!
- Dateisystem und Programmlader
- Netzwerk und TCP/IP
- . . .

Realitätscheck: MPStuBS $\leftrightarrow$ „richtiges BS“

Es fehlt noch eine ganze Menge!

- Adressraumverwaltung und Prozesskonzept $\rightsquigarrow$ BST im WS!
- Dateisystem und Programmlader
- Netzwerk und TCP/IP
- . . .

Beispiel Linux [10]

Aug 91 Linux 0.01: bash, Dateisystem

Jan 92 Linux 0.12: Virtueller Speicher (Paging)

Realitätscheck: MPStuBS $\leftrightarrow$ „richtiges BS“

Es fehlt noch eine ganze Menge!

- Adressraumverwaltung und Prozesskonzept $\rightsquigarrow$ BST im WS!
- Dateisystem und Programmlader
- Netzwerk und TCP/IP
- . . .

Beispiel Linux [10]

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!)

Realitätscheck: MPStuBS $\leftrightarrow$ „richtiges BS“

Es fehlt noch eine ganze Menge!

- Adressraumverwaltung und Prozesskonzept $\rightsquigarrow$ BST im WS!
- Dateisystem und Programmlader
- Netzwerk und TCP/IP
- . . .

Beispiel Linux [10]

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**

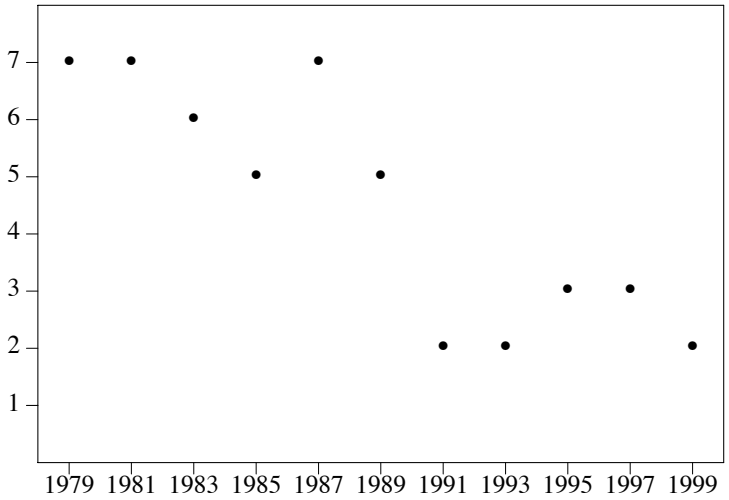
Betriebssysteme ↦ ausgeforscht?

TUHH
Technische Universität Hamburg

„*Systems Software Research is Irrelevant*” [6]

Urgestein Robert Pike (2000), einer der Entwickler von UNIX, Inferno [5], Plan 9 [7] 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
- . . .



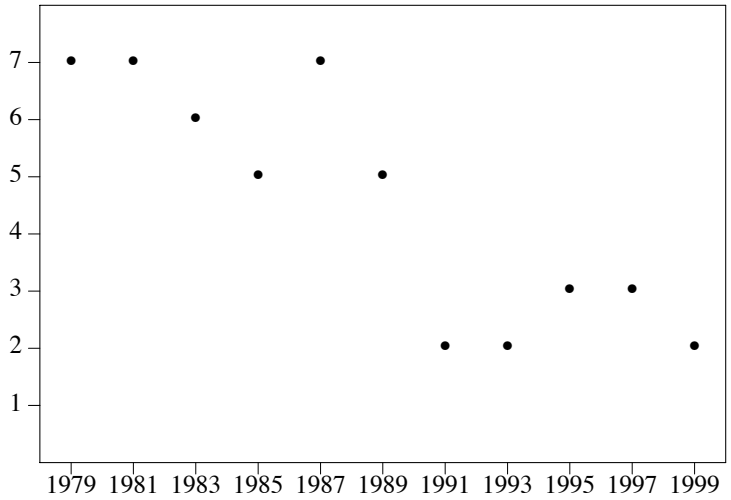
New Operating Systems at SOSP [6]

Betriebssysteme ↦ ausgeforscht?

„Systems Software Research is Irrelevant” [6]

Urgestein Robert Pike (2000), einer der Entwickler von UNIX, Inferno [5], Plan 9 [7] und UTF-8 (zur Zeit bei Google beschäftigt):

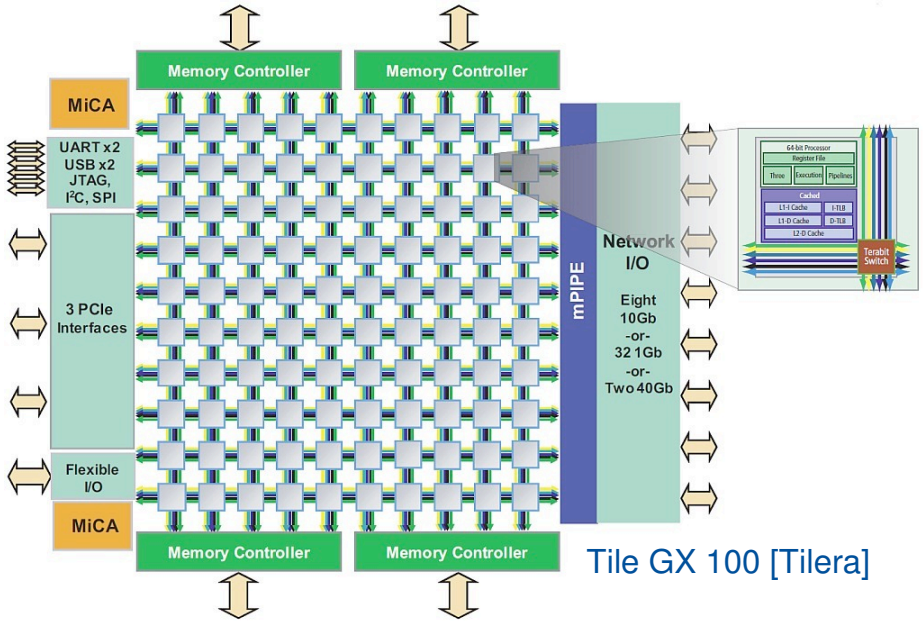
- Where is the innovation? ⇝ Microsoft, mostly
- Every other „new” OS ends up being UNIX
- Linux? ⇝ Just another copy of the same old stuff
- . . .



New Operating Systems at SOSP [6]

Aber dann. . .

The Multicore Challenge!

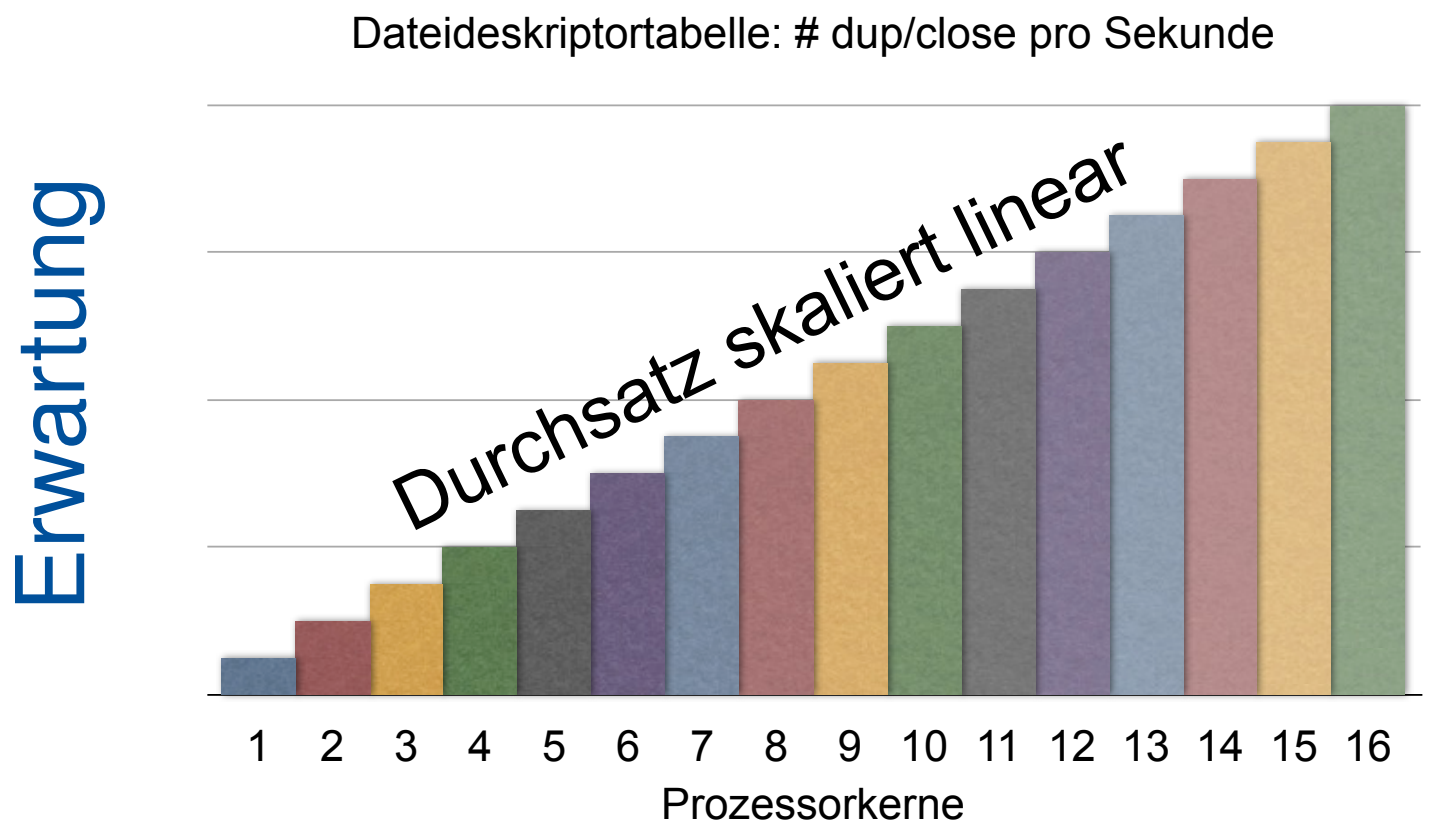


Tile GX 100 [Tilera]

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:

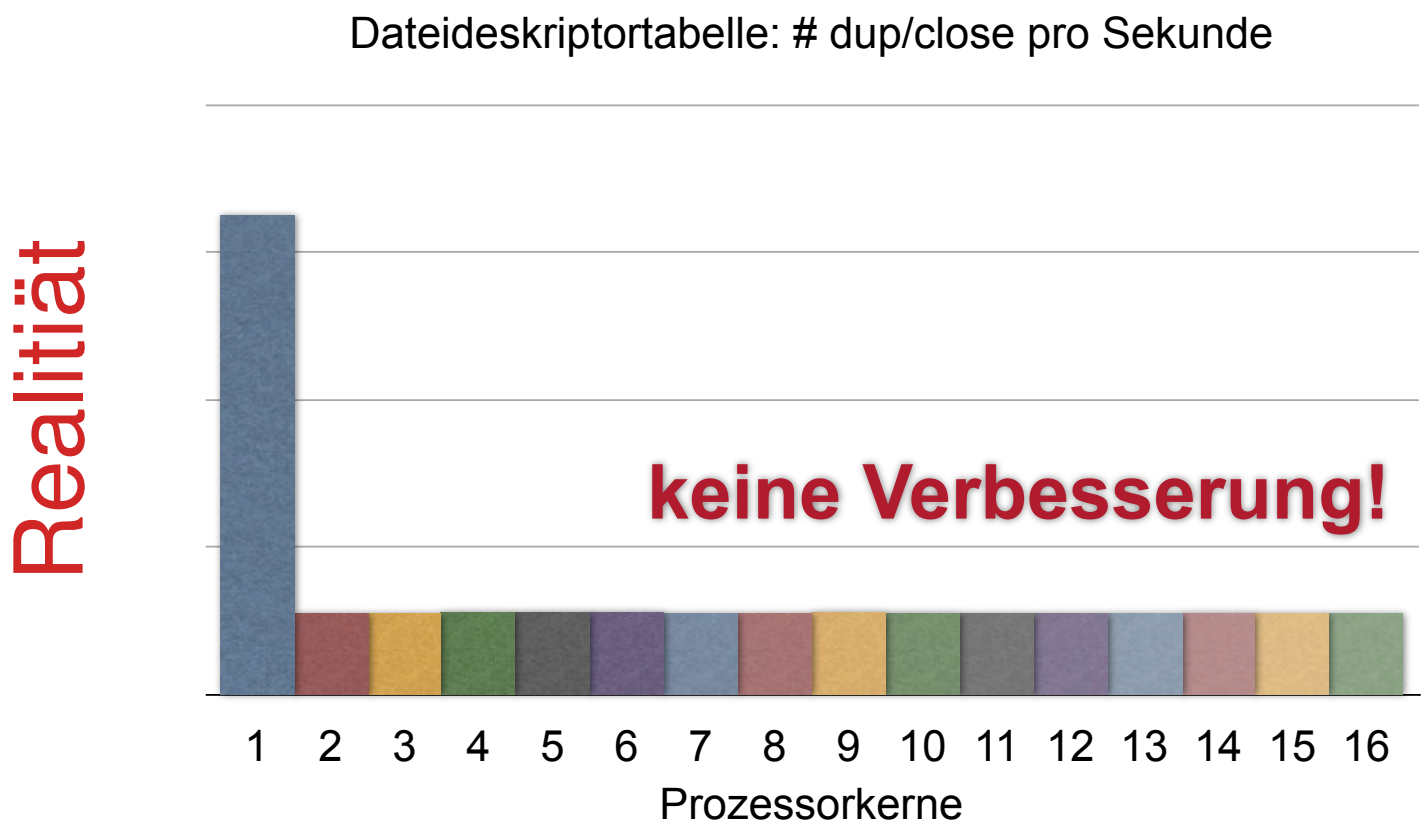
```
int f = open(...); while(1){ close( dup( f ) ); }
```



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

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
  1. Grobgranulares *Locking* $\rightsquigarrow$ *false sharing* $\rightsquigarrow$ keine Skalierbarkeit

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

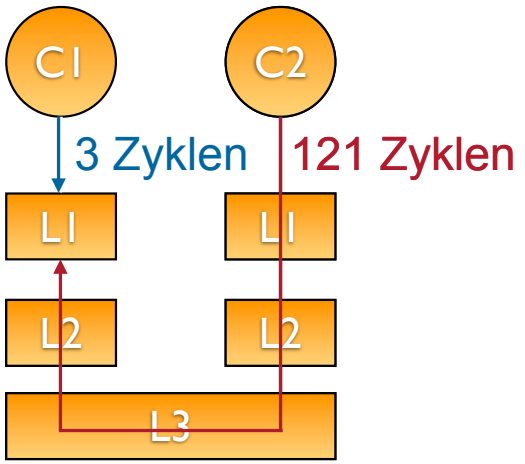
1. false sharing

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
  1. Grobgranulares *Locking* ⇝ *false sharing* ⇝ keine Skalierbarkeit
  2. Geteilte Datenstruktur ⇝ *cache trashing* ⇝ 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
  1. Grobgranulares *Locking* $\rightsquigarrow$ *false sharing* $\rightsquigarrow$ keine Skalierbarkeit
  2. Geteilte Datenstruktur $\rightsquigarrow$ *cache trashing* $\rightsquigarrow$ Durchsatzabfall

Multicore: POSIX ($\mapsto$ 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: [11]
  - BS für 100 bis 1000 Kerne $\leadsto$ *time sharing* wird zu *space sharing*
  - Letztlich ähnlicher Ansatz wie Barrelfish
- TxOS — UT, SOSP 2009, Monolith (Linux): [8]
  - Konkurrenz zulassen durch *transactional syscalls* (statt *Locks*)
  - Anwendungen müssen angepasst werden

. . . oder doch nicht?

TUHH
Technische Universität Hamburg

- 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

. . . oder doch nicht?

TUHH
Technische Universität Hamburg

- Clements u. a. (SOSP 2013) [4]
  - „The Scalable Commutativity Rule: Designing Scalable Software for Multicore Processors“
  - Skalierbarkeit von *Schnittstellen* theoretisch und praktisch untersucht anhand Kommutativität der (möglichen) Implementierung.

- Idee: Wenn Operationen kommutativ sind, können sie (im Prinzip) auch skalierbar implementiert werden.

. . . oder doch nicht?

TUHH
Technische Universität Hamburg

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]

„Finally, using sv6, we showed that it ***is practical to achieve a broadly scalable implementation of POSIX*** *by applying the rule, and that commutativity is essential to achieving scalability and performance on real hardware. ”* [4]

Fazit

Parallelität ist ein Dauerbrenner

The Next Frontier: Speicher. . . (again)

TUHH
Technische Universität Hamburg

- Neue und verbesserte Speichertechnologien sind disruptiv
  - DRAM ist flüchtig und klein $\Rightarrow$ NVRAM ist größer und persistent
  - Festplatten sind langsam $\Rightarrow$ SSDs sind (unfassbar) schnell

  $\Rightarrow$ BS orchestriert die Speicherhierarchie

Performance

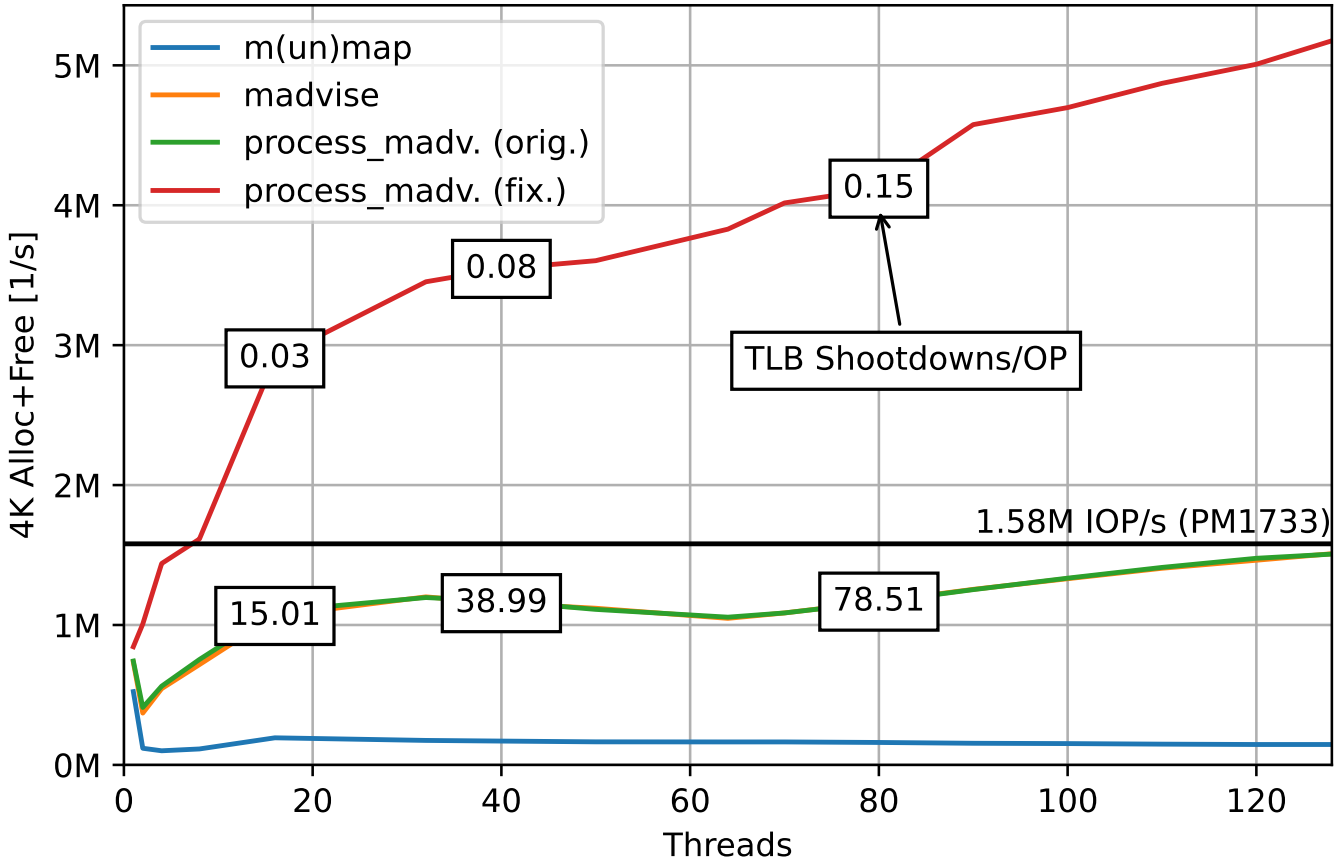
- PM1733/35 NVMe: 8 GiB/s
- Das BS kann nicht mithalten
- By-Pass Technologien haben Hochkonjunktur (SPDK)

Abstraktionen

- Traditionelle Trennung zwischen Dateien und Speicher
- `mmap()` als Brücke
- *Neu*: Heterogeniät (CPU, GPU)

4K Page Allocation

Linux 5.16, AMD EPYC 7713 processor (64 cores, 128 hardware threads), 512 GB RAM, 3.8 TB Samsung PM1733 SSD.



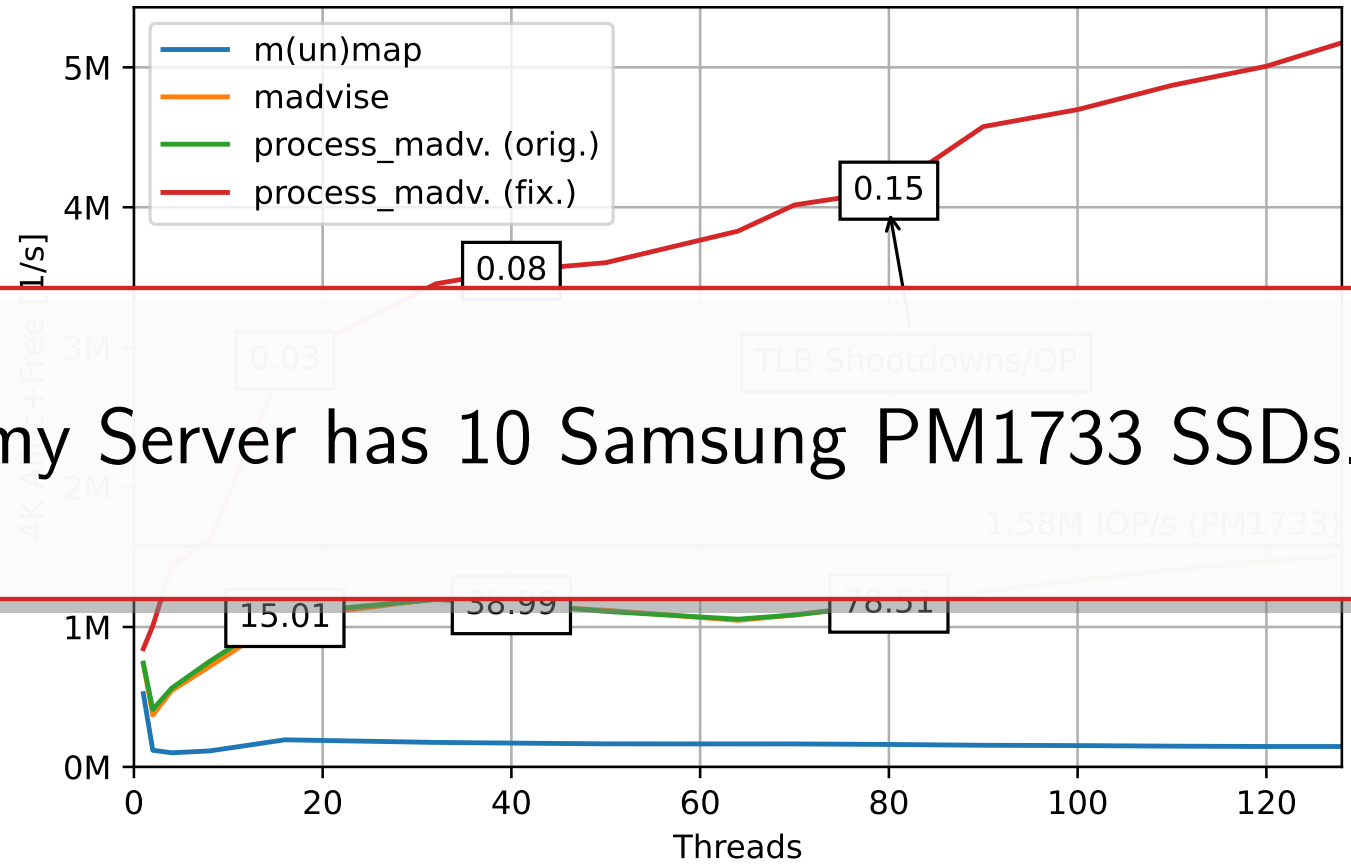
- **Task**: Install 4K Pages in VM
- **Goal**: 1.58M IOP/s (PM1733)

\+ Batch TLB Shootdowns

– Lock-Based Synchronization

4K Page Allocation

Linux 5.16, AMD EPYC 7713 processor (64 cores, 128 hardware threads), 512 GB RAM, 3.8 TB Samsung PM1733 SSD.



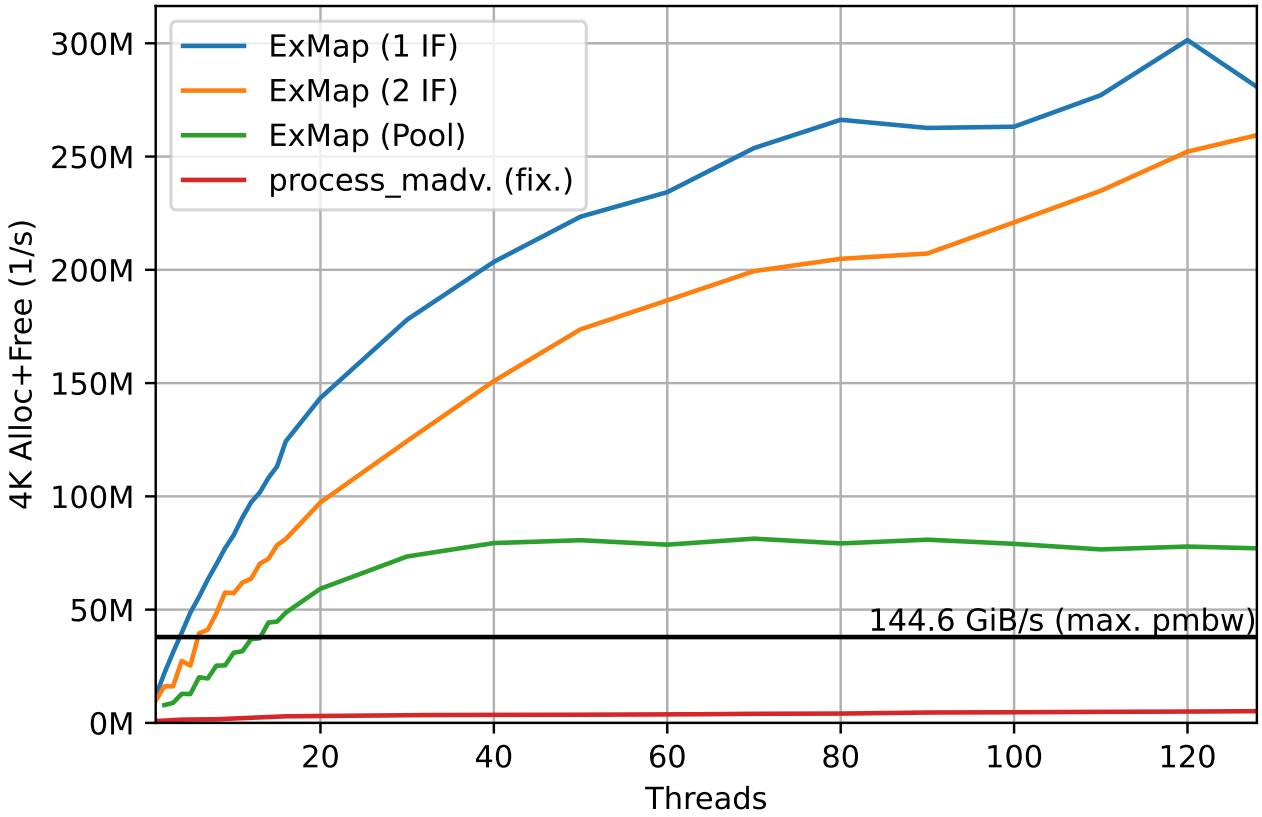
. . . my Server has 10 Samsung PM1733 SSDs. . .

- **Task**: Install 4K Pages in VM
- **Goal**: 1.58M IOP/s (PM1733)

\+ Batch TLB Shootdowns

– Lock-Based Synchronization

ExMap: Fast and Explicit File Mappings



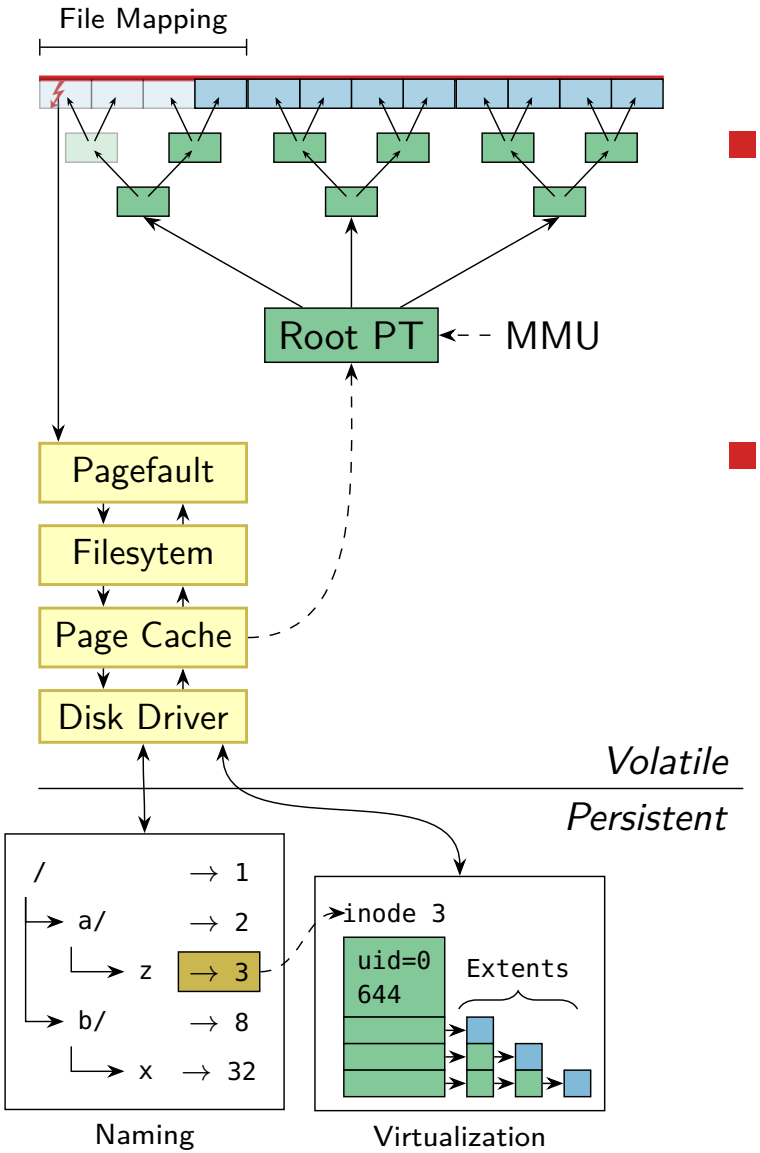
User Interface

- Explicit Alloc, Free, Read
- No Implicit Write-Back
- Batched Operations

Techniques

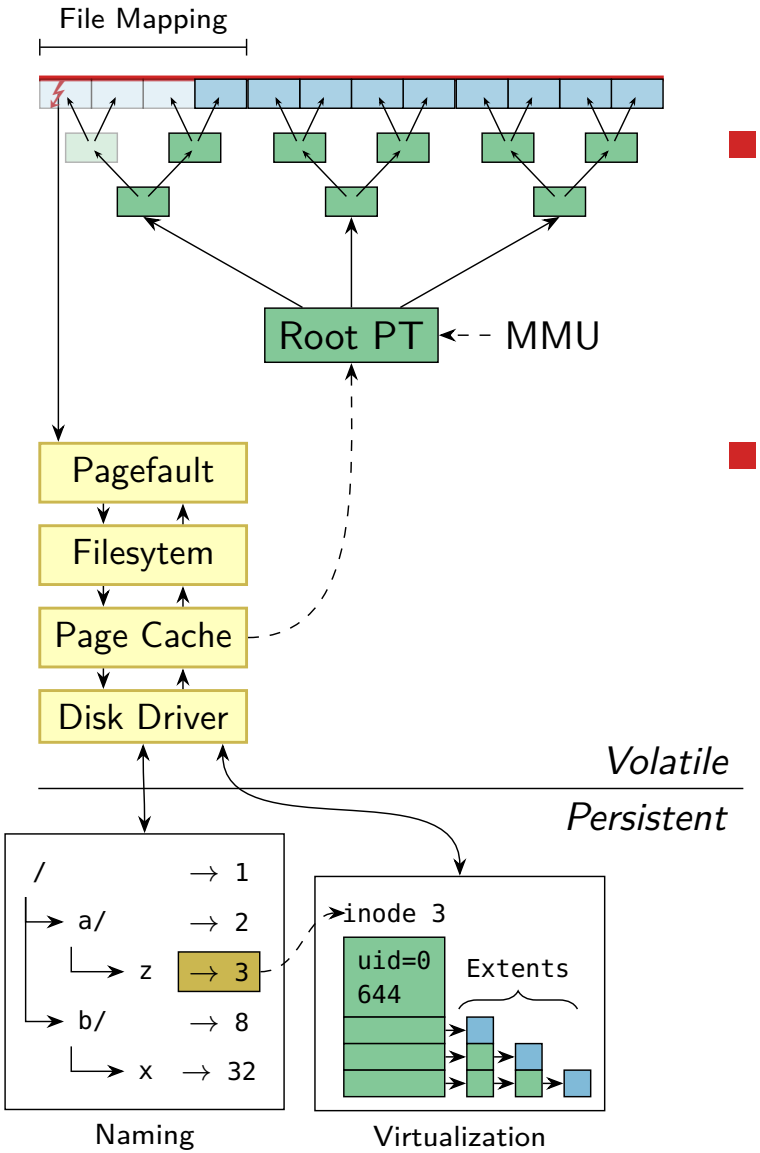
- Pre-Allocated Memory Pool
- Multiple Free-Lists and Stealing
- Lock-Free Algorithms

Morsels: New OS Abstraction for Persistency



- Files: Naming and Virtualization
  - Resolve Filename to inode
  - Translate from File Offset to Disk Offset
- Hot Path: Access File Contents
  1. Traverse FS, Caches, and Driver Layers
  2. Load Data into DRAM
  3. Update MMU Config
  4. MMU translates Virtual Addresses

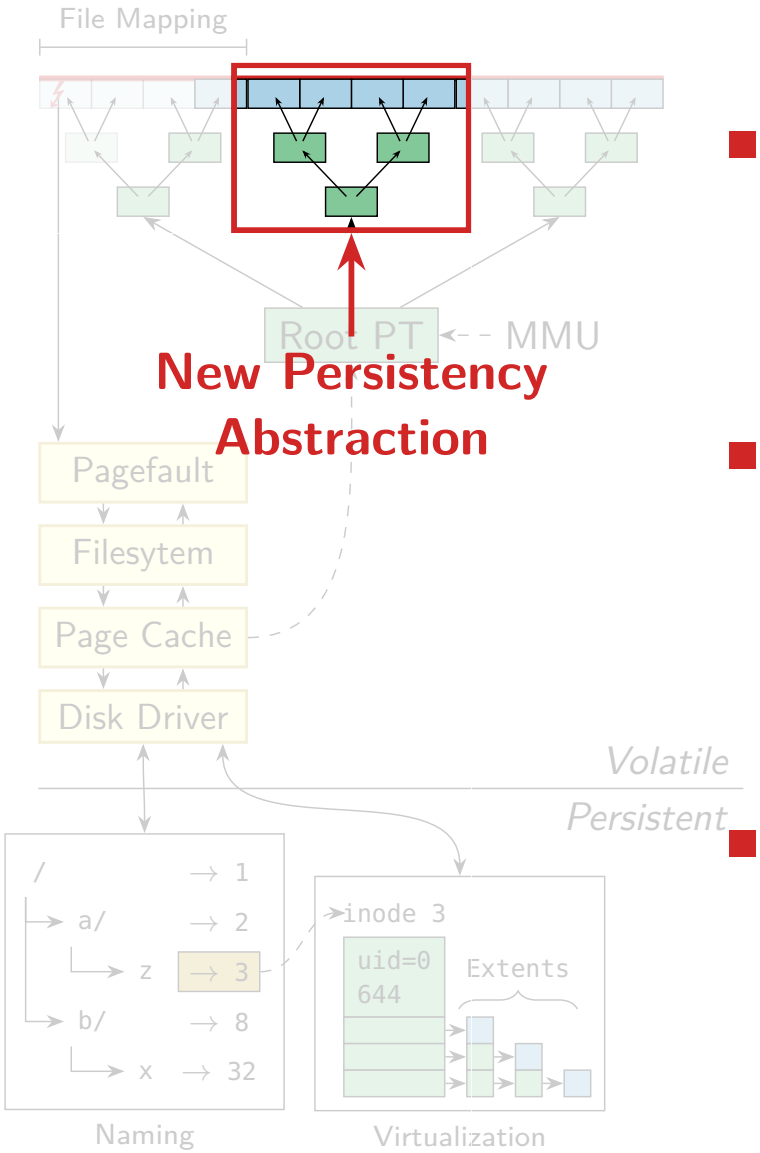
Morsels: New OS Abstraction for Persistency



- **Files**: Naming and Virtualization
  - Resolve Filename to inode
  - Translate from File Offset to Disk Offset
- **Hot Path**: Access File Contents
  1. Traverse FS, Caches, and Driver Layers
  2. ~~Load Data into DRAM~~
  3. Update MMU Config
  4. MMU translates Virtual Addresses

MMU/CPU can address NV-RAM directly!

Morsels: New OS Abstraction for Persistency



- Files: Naming and Virtualization
  - *(Resolve Filename to inode)*
  - ~~Translate from File Offset to Disk Offset~~
- Hot Path: Access File Contents
  1. *(Traverse FS, Caches, and Driver Layers)*
  2. ~~Load Data into DRAM~~
  3. ~~Update MMU Config~~
  4. MMU translates Virtual Addresses
- **New approach** to provide memory
  - Unified (non-)volatile RAM Objects
  - Page-Tables Subtrees in (NV-)RAM
  - Optional: Naming and Caching

Wie geht es weiter?

- Vorlesung/Übung: Betriebssystemtechnik (**BST**)
  - https://osg.tuhh.de/Lehre/WS22/L_BST/
  - OOStuBS um Virtuellen Speicher und Schutz erweitern.
  - „Natürliche“ Fortsetzung von BSB
- Examensarbeiten
  - https://osg.tuhh.de/Theses/
  - Bachelorarbeiten
  - Masterarbeiten
  - Doktorarbeiten :-)

... oder persönlich nachfragen!

Das war’s :-)

Ich wünsche euch
erfolgreiche und erholsame
„Semesterferien“

. . . und ein Wiedersehen
im Wintersemester 2022!

Referenzen

TUHH
Technische Universität Hamburg

[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] Austin T. Clements, M. Frans Kaashoek, Nickolai Zeldovich u. a. „The Scalable Commutativity Rule: Designing Scalable Software for Multicore Processors“. In: *Proceedings of the 24th ACM Symposium on Operating Systems Principles (SOSP '13)* (Farmington, PA, USA). New York, NY, USA: ACM Press, 2013, S. 1–17. ISBN: 978-1-4503-2388-8. DOI: 10.1145/2517349.2522712.

[5] 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.

Referenzen (Forts.)

TUHH
Technische Universität Hamburg

[6] Rob Pike. *Systems Software Research is Irrelevant*. Talk. CS Colloquium, Columbia University, 2000. URL: http://herpolhode.com/rob/utah2000.pdf (besucht am 09. 12. 2010).

[7] Rob Pike, Dave Presotto, Sean Dorward u. a. „Plan 9 from Bell Labs“. In: *Computing Systems* 8.3 (1995), S. 221–254.

[8] 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.

[9] *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.

[10] Linus Torvalds und David Diamond. *Just for Fun: The Story of an Accidental Revolutionary*. HarperCollins, 2001. ISBN: 978-0066620725.

[11] 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.