technische universität
dortmund

Aspektorientierte Entwicklung einer konfigurierbaren x86-Variante der Betriebssystemfamilie CiAO

Abschlussvortrag

Björn Bosselmann

Bjoern.Bosselmann@cs.uni-dortmund.de

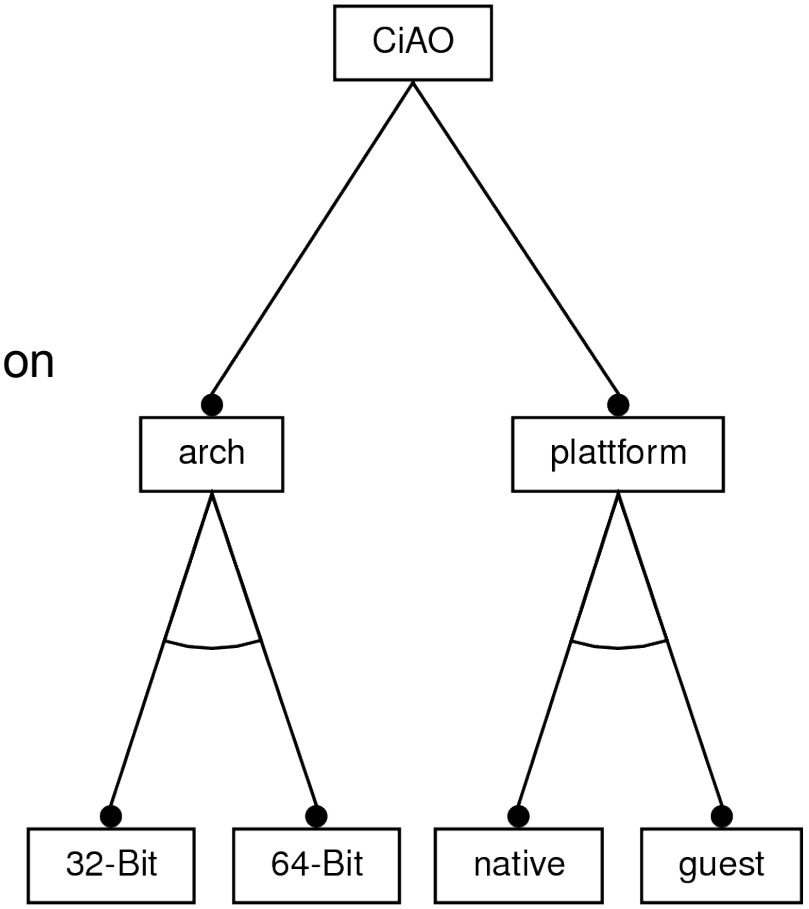
30. August 2010

Inhalt

- **Einleitung**
- Systemarchitektur
- Plattformspezifische Treiber
- Evaluation
- Zusammenfassung
- Ausblick

Einleitung

- CiAO als konfigurierbares Betriebssystem für *x86*
  - natives System
    - direkt auf einem PC bootbar
  - Linux-Gast-System
    - als „normale“ Linux-Applikation
  - 32-Bit-System
    - *i386*-Architektur
  - 64-Bit-System
    - *x86_64*-Architektur



- Ziel:
  - Aspektorientierte Lösungsansätze evaluieren

Inhalt

- Einleitung
- **Systemarchitektur**
- Plattformspezifische Treiber
- Evaluation
- Zusammenfassung
- Ausblick

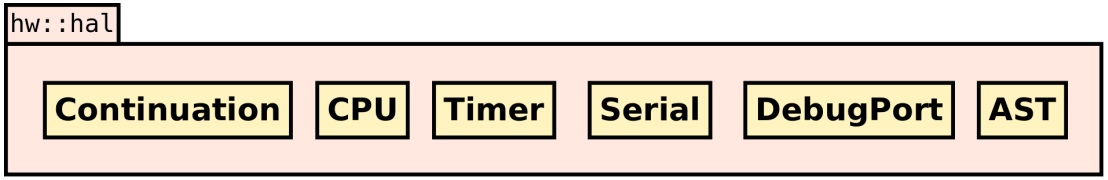
Systemarchitektur

- Vorgehen beim Portieren von CiAO auf den *x86*:

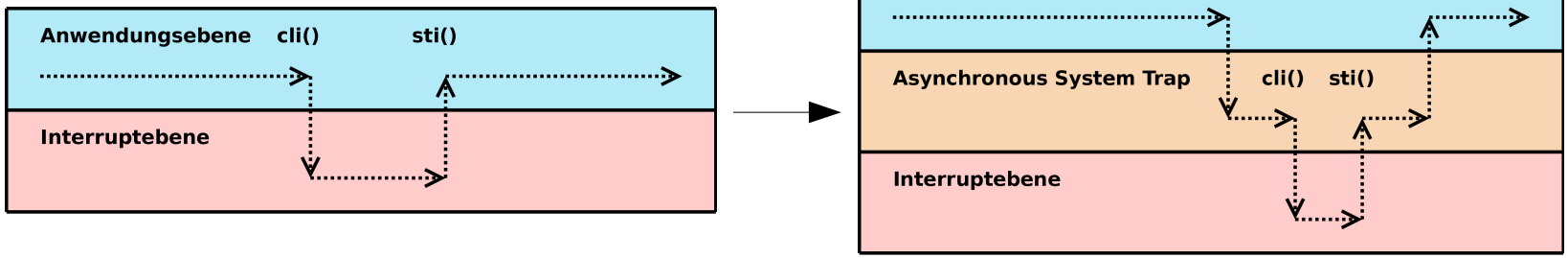
1) Erstellen eines neunen p::v-Konfigurationsraumes
   - *Featuremodel* und *Familymodel*
2) Anpassungen an die verwendete Toolchain
   - *Makefile*, *intrinsics.h*, *compiler.h*
3) Schreiben eines plattformspezifischen Startupcodes + Linkerskripts
   - Betriebsmodus (32-Bit, 64-Bit)
4) Implementieren der grundlegenden HAL-Schnittstellen und Treiber
   - *CPU*, *Continuation*, *Timer*, *DebugPort*
   - *PIT*, *PIC*, *CGA*, *AT-Keyboard*
5) Bereitstellen plattformspezifischer HAL-Schnittstellen
   - *Paging*, *Ethernet*, *Blockdevice*
6) Implementieren von beispielhaften Treibern
   - *PRO/1000*, *VirtioNet*, *IDE*

Systemarchitektur

- Grundlegende Anpassungen



  - Systeminitialisierung
    - Bootloader
  - Kontextwechsel
    - klassisch oder *TSS*?
  - Unterbrechungsbehandlung
  - *Asynchronous System Trap* (*AST*)



Systemarchitektur

- Linux-Gast
  - Initialisierung
    - ggf. verwendete Bibliotheken initialisieren
  - Unterbrechungsbehandlung
    - Signalbasierte Unterbrechungen
  - Treiber
    - Linux-Syscall-Wrapper

  - wiederverwendbar:
    - Kontextwechsel
      - außer *TSS*-Taskwechsel
    - keine Unterscheidung zwischen 32-Bit und 64-Bit notwendig
      - Unterschiede werden vom Compiler behandelt

Systemarchitektur

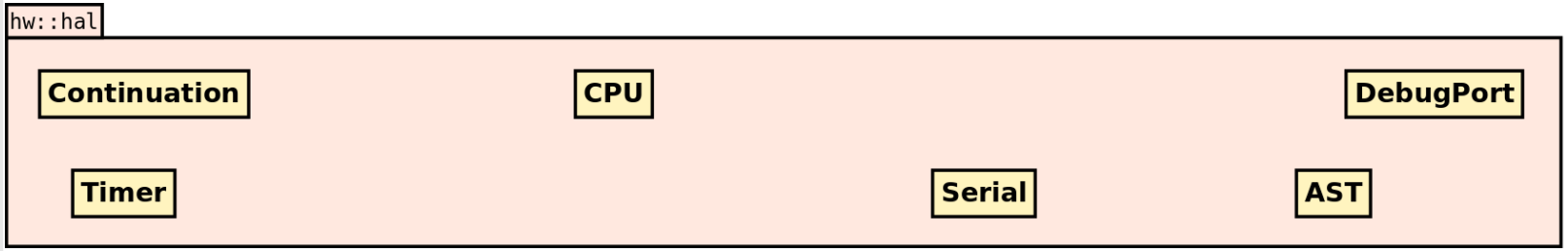
- 64-Bit-Unterstützung
  - Systeminitialisierung
    - Umschaltung in den *Long-Mode*
  - Kontextwechsel
    - 64-Bit Register
  - Paging
    - im *Long-Mode* zwingend notwendig
    - anderes Seitentabellenformat
  - wiederverwendbar:
    - Unterbrechungsbehandlung
    - alle Treiber

Inhalt

- Einleitung
- Systemarchitektur
- **plattformspezifische Treiber**
- Evaluation
- Zusammenfassung
- Ausblick

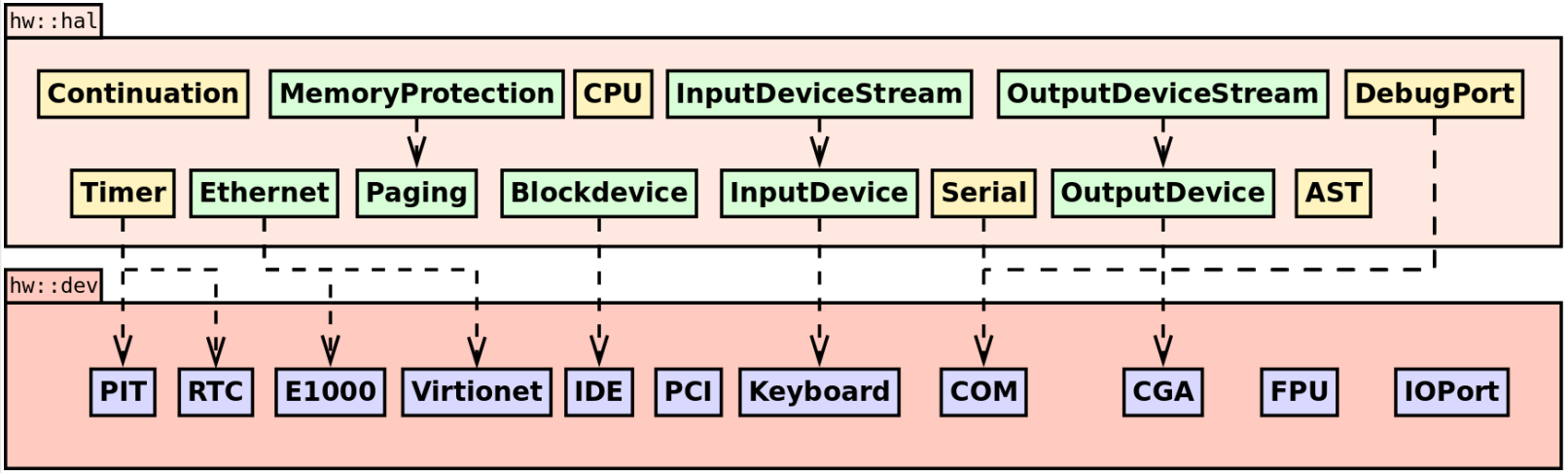
Plattformspezifische Treiber

- Was war bereits vorher vorhanden?
  - Vorhandene Schnittstellen müssen implementiert werden
  - hauptsächlich grundlegende Komponenten, die fast immer vorhanden sind
  - Ausnahme: *AST – Asynchronous System Trap*
    - muss in Software emuliert werden



Plattformspezifische Treiber

- Was musste ergänzt werden?
  - Ethernet
  - Blockdevice
  - Paging
  - *x87*-Gleitkommazahlunterstützung
  - gepufferte, formatierte Ein- und Ausgabe



Plattformspezifische Treiber

zur Erinnerung:

- Aspektorientierte Entwurfsprinzipien
  - **Lose Kopplung** ↔ *Binding-Aspekte*
    - Selbstintegration von Komponenten
  - **Sichtbare Übergänge** ↔ *Policy-Aspekte*
    - Explizite Joinpoints
  - **Minimale Erweiterungen** ↔ *Extensions-Slices*
    - Erweiterung von Klassen mittels Slices

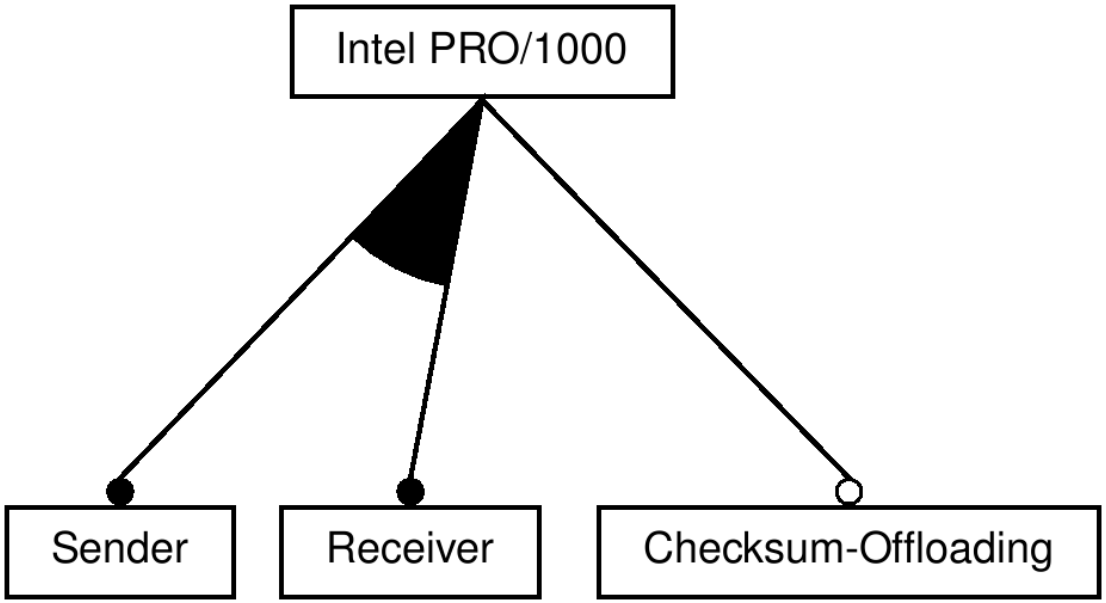
Plattformspezifische Treiber

- Beispiel *x87*-Gleitkommazahlunterstützung:
  - FPU-Kontext muss bei jedem Taskwechsel gesichert werden
    - Nutzung von Policy-Aspekten an `before_CPURelease` und `after_CPUReceive`

```
aspect hw_hal_x87context_ah {

  advice slice class x87Context_FPU {
    UInt8 fpu[108];
  };

  advice execution("% hw::hal::Continuation::before_CPURelease(...)")
  && that(from) : before(hw::hal::Continuation& from) {
    hw::dev::x87_FPU::Inst().save(from.context_.fpu);
  }

  advice execution("% hw::hal::Continuation::after_CPUReceive()")
  && that(to) : before(hw::hal::Continuation& to) {
    hw::dev::x87_FPU::Inst().load(to.context_.fpu);
  }

};
```

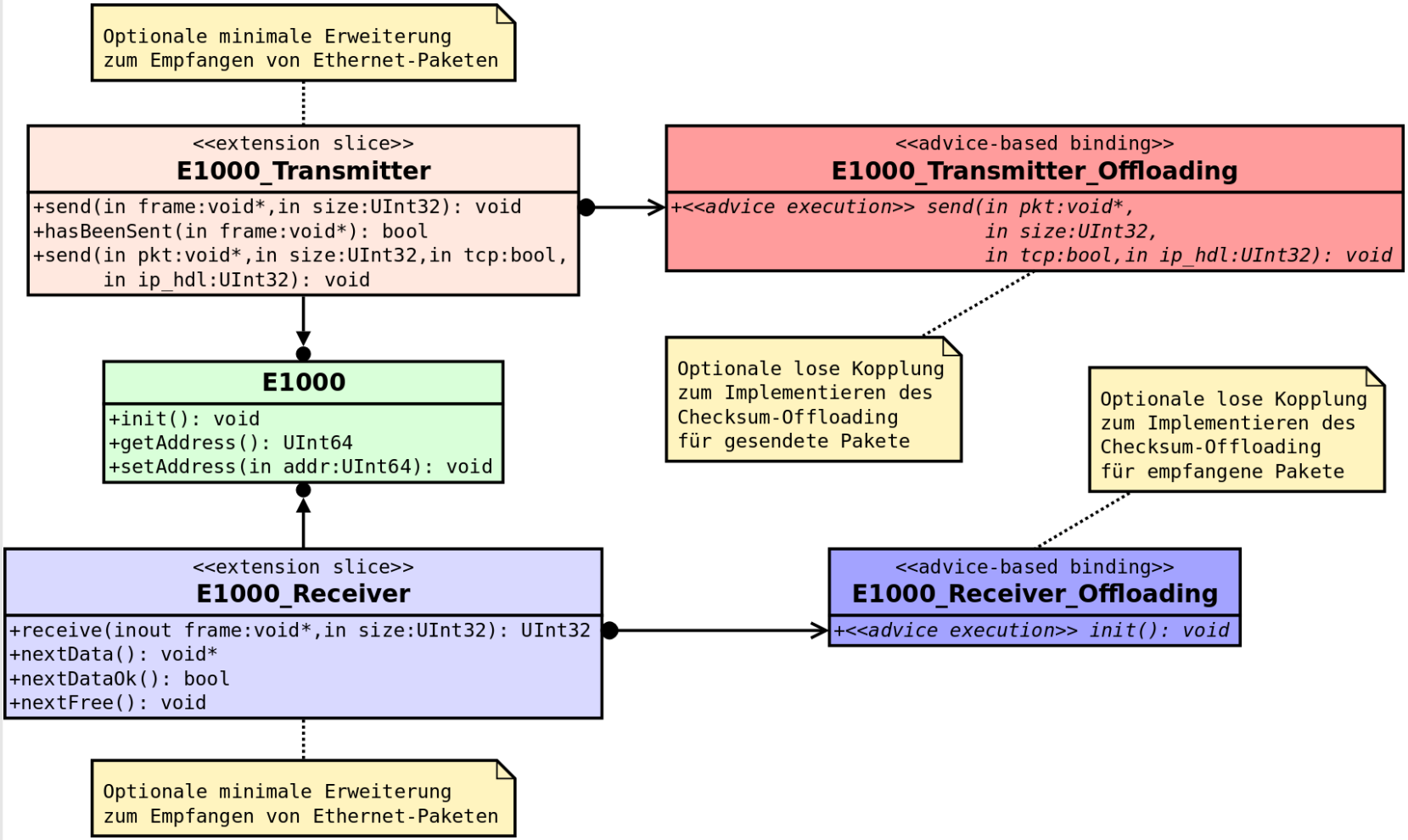
Plattformspezifische Treiber

- Beispiel *Ethernet* mit *Intel PRO/1000*:
  - optionale Sende- und Empfangsfunktionen für Ethernetframes
  - optionale *Offloading*-Funktionalitäten
    - Berechnung der IP-, UDP- und TCP-Prüfsummen in Hardware



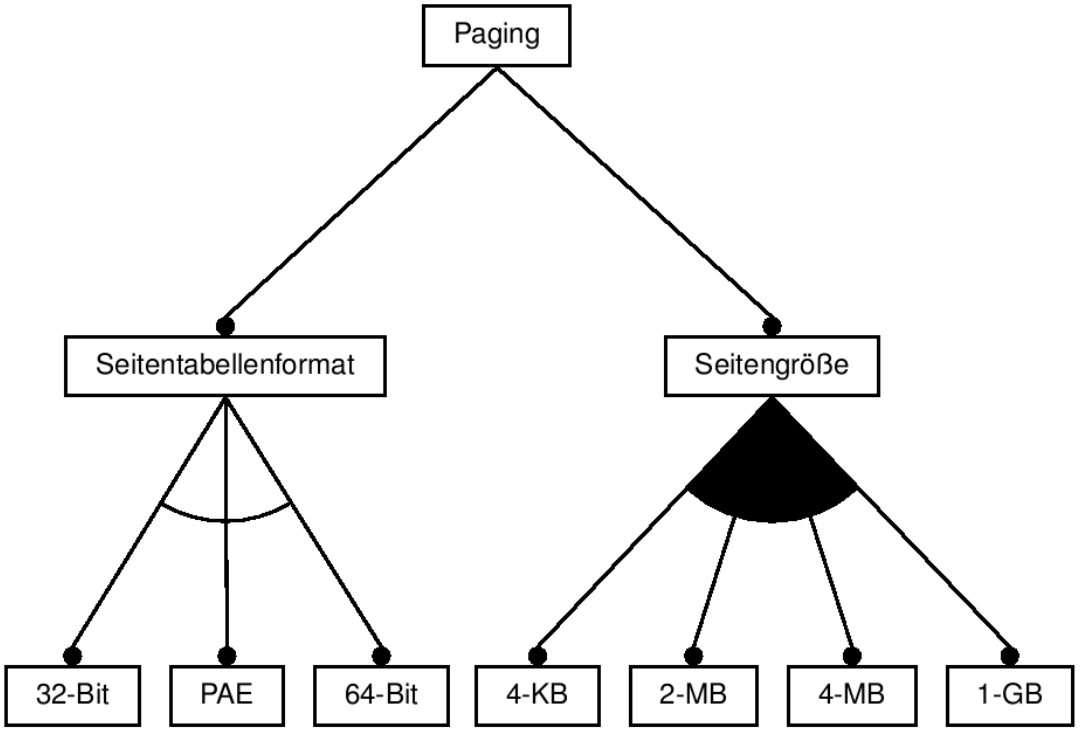
Plattformspezifische Treiber

- Beispiel *Ethernet* mit *Intel PRO/1000*:



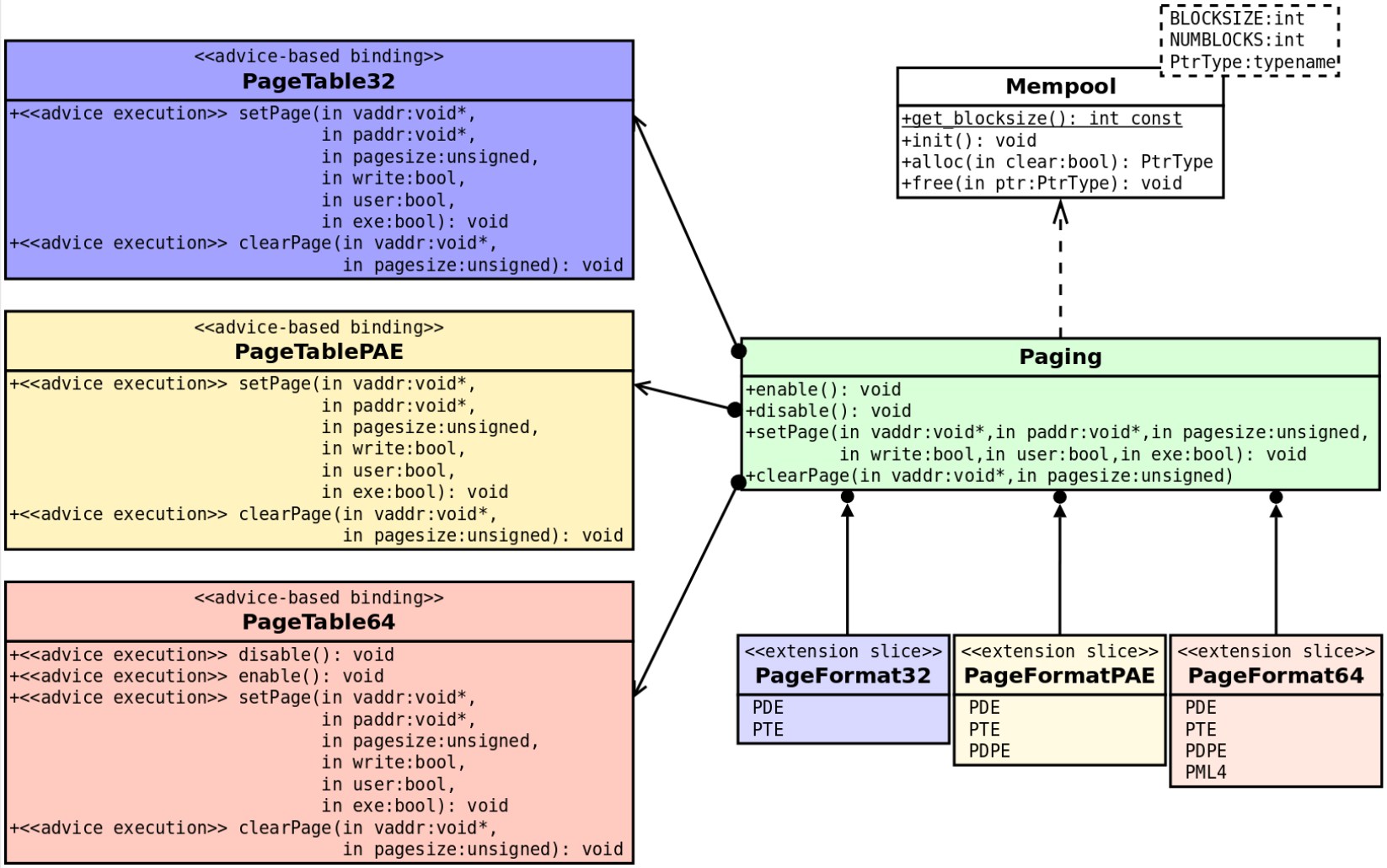
Plattformspezifische Treiber

- Beispiel *Paging*:
  - Alternative Seitentabellenformate
    - 32-Bit, PAE, 64-Bit
  - wählbare Seitengrößen
    - 4-KB, 2-MB, 4-MB, 1-GB



Plattformspezifische Treiber

- Beispiel *Paging*:



Inhalt

- Einleitung
- Systemarchitektur
- Plattformspezifische Treiber
- **Evaluation**
- Zusammenfassung
- Ausblick

Evaluation

- Qualitative Evaluation
  - Wie ließen sich die Belange umsetzen?
    - aspektorientiert
      - ⚭ Binding-Aspekt
      - § Policy-Aspekt
      - \+ Extension-Slice
    - nicht aspektorientiert
      - x Austausch der Implementierung auf Quellcode-Ebene

Belang	Initialisierung	Kontextwechsel	Paging	Systemaufrufe	Unterbrechungen	Treiberkonfiguration
Gastsystem	x				x	
64-Bit-Modus	x	x	⚭ +		x	
x87	⚭	§ +				
TSS-Taskwechsel	⚭	x				
Paging			⚭ +	§		
Speicherschutz	⚭	+		⚭	⚭	
Tastaturtreiber	⚭					+
PRO/1000-Treiber	⚭		⚭			⚭ +
IDE-Treiber	⚭					+

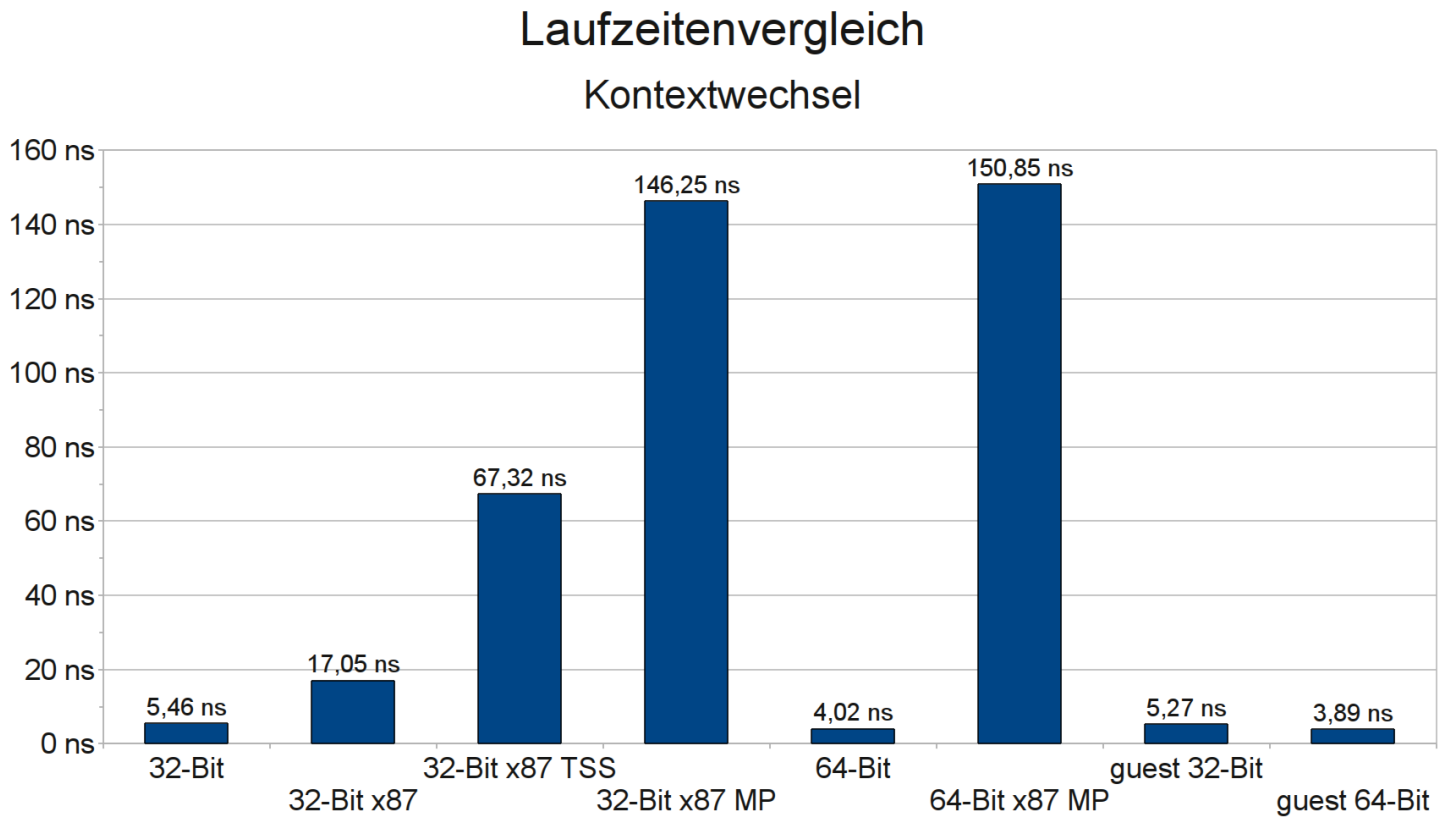
Evaluation

- Quantitative Evaluation
  - Code- und Datengrößen

Konfiguration	.text	.data	.bss
32-Bit „Minimalsystem“	7.718	2.127	73.038
32-Bit PCI	+ 628	+ 0	+ 640
32-Bit PCI + PRO/1000	+ 2.061	+ 0	+ 67.783
32-Bit PCI + IDE	+ 1.371	+ 0	+ 172
32-Bit Paging	+ 626	+ 0	+ 4.202.502
32-Bit „Maximalsystem“	19.290	2.127	4.578.121
64-Bit „Minimalsystem“	11.744	4.245	4.280.888
64-Bit PCI	+ 622	+ 0	+ 1.568
64-Bit PCI + PRO/1000	+ 2.861	+ 0	+ 67.896
64-Bit PCI + IDE	+ 2.005	+ 0	+ 1,744
64-Bit „Maximalsystem“	26.053	4.245	4.586.461

Evaluation

- Quantitative Evaluation
  - Laufzeitenvergleich
    - 2 Tasks fordern jeweils 100 Mio. Taskwechsel an
    - Time-Stamp-Counter der CPU misst Gesamtzeit



Inhalt

- Einleitung
- Systemarchitektur
- Plattformspezifische Treiber
- Evaluation
- **Zusammenfassung**
- Ausblick

Zusammenfassung

- Was wurde gemacht?
  - CiAO-*HAL* für *x86* angepasst
    - natives und Gast-System für 32-Bit und 64-Bit
  - Neue *HAL*-Schnittstellen für
    - Paging
    - Ethernet
    - Blockdevice
- Wie wurde es umgesetzt?
  - Architekturelle Unterschiede nur schwer aspektorientiert umsetzbar
    - Startupcode
    - Kontextwechselcode
  - Neue Features lassen sich gut aspektorientiert umsetzen
    - *x87* Gleitkommazahlunterstützung
    - Paging
    - Treiber

Inhalt

- Einleitung
- Systemarchitektur
- Plattformspezifische Treiber
- Evaluation
- Zusammenfassung
- **Ausblick**

Ausblick

- „Low-Level-Ebene“
  - *APIC*
    - Unterstützung für Multicore- / Multiprozessorsysteme
  - Paravirtualisierung
    - Virtio bietet nur paravirtualisierte Peripheriegeräte
    - *Xen Hypercalls* und *VMware-VMI*
      - Paravirtualisierung der Unterbrechungsbehandlung und Speicherverwaltung
  - Linux-Gastsystem
    - austauschbares Treibermodell
      - Beispiel: Unterbrechungsbehandlung

Ausblick

- „High-Level-Ebene“
  - *POSIX*-Personality
    - Quellcode-Kompatibilität zu *POSIX*-Applikationen
  - Netzwerkprotokolle für Ethernet
    - konfigurierbarer TCP/IP-Stack
  - Dateisystem für ATA-Blockgeräte

Schluss

Zeit für Fragen...