Hardware Virtualisierungs Support für PikeOS

Design eines Virtual Machine Monitors auf Basis eines Mikrokernels

Tobias Stumpf

SYSGO AG, Am Pfaffenstein 14, 55270 Klein-Winternheim

HS Furtwangen, Fakultät Computer and Electrical Engineering,
Robert-Gerwig-Platz 1, 78120 Furtwangen

18. November 2010

Motivation

Servermarkt

- Zusammenfassen verschiedener Serverdienste auf einer physikalischen Maschine
- Lastverteilung

Desktopbereich

- Einsatz verschiedener (ähnlicher) Betriebssysteme
- Entwicklung

eingebettete Systeme

- Abschottung von Teilkomponenten
- Anwendung mit unterschiedlichen Sicherheits- und Echtzeitanforderungen
- Einsatz unterschiedlicher Betriebssysteme

Motivation

Servermarkt

- Zusammenfassen verschiedener Serverdienste auf einer physikalischen Maschine
- Lastverteilung

Desktopbereich

- Einsatz verschiedener (ähnlicher) Betriebssysteme
- Entwicklung

eingebettete Systeme

- Abschottung von Teilkomponenten
- Anwendung mit unterschiedlichen Sicherheits- und Echtzeitanforderungen
- Einsatz unterschiedlicher Betriebssysteme

Motivation

Servermarkt

- Zusammenfassen verschiedener Serverdienste auf einer physikalischen Maschine
- Lastverteilung

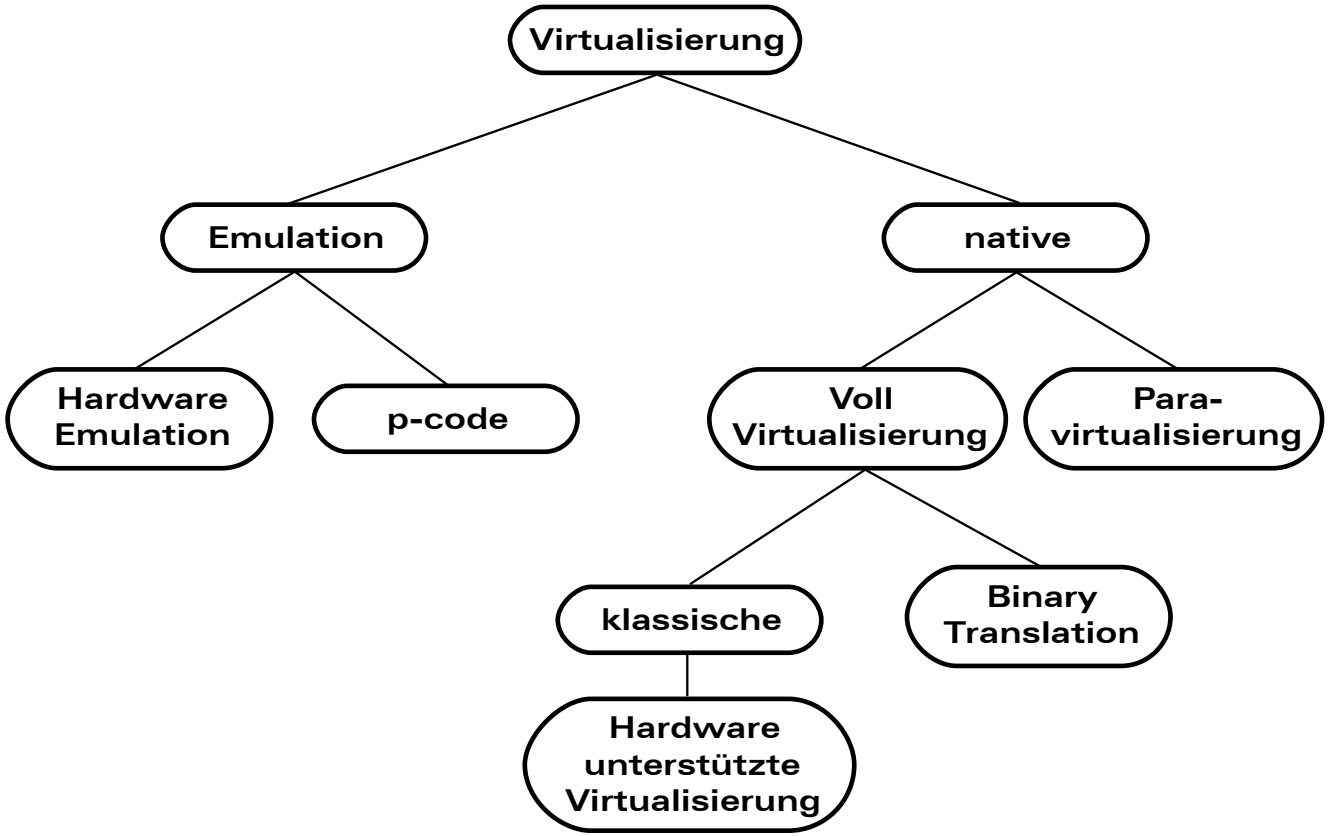
Desktopbereich

- Einsatz verschiedener (ähnlicher) Betriebssysteme
- Entwicklung

eingebettete Systeme

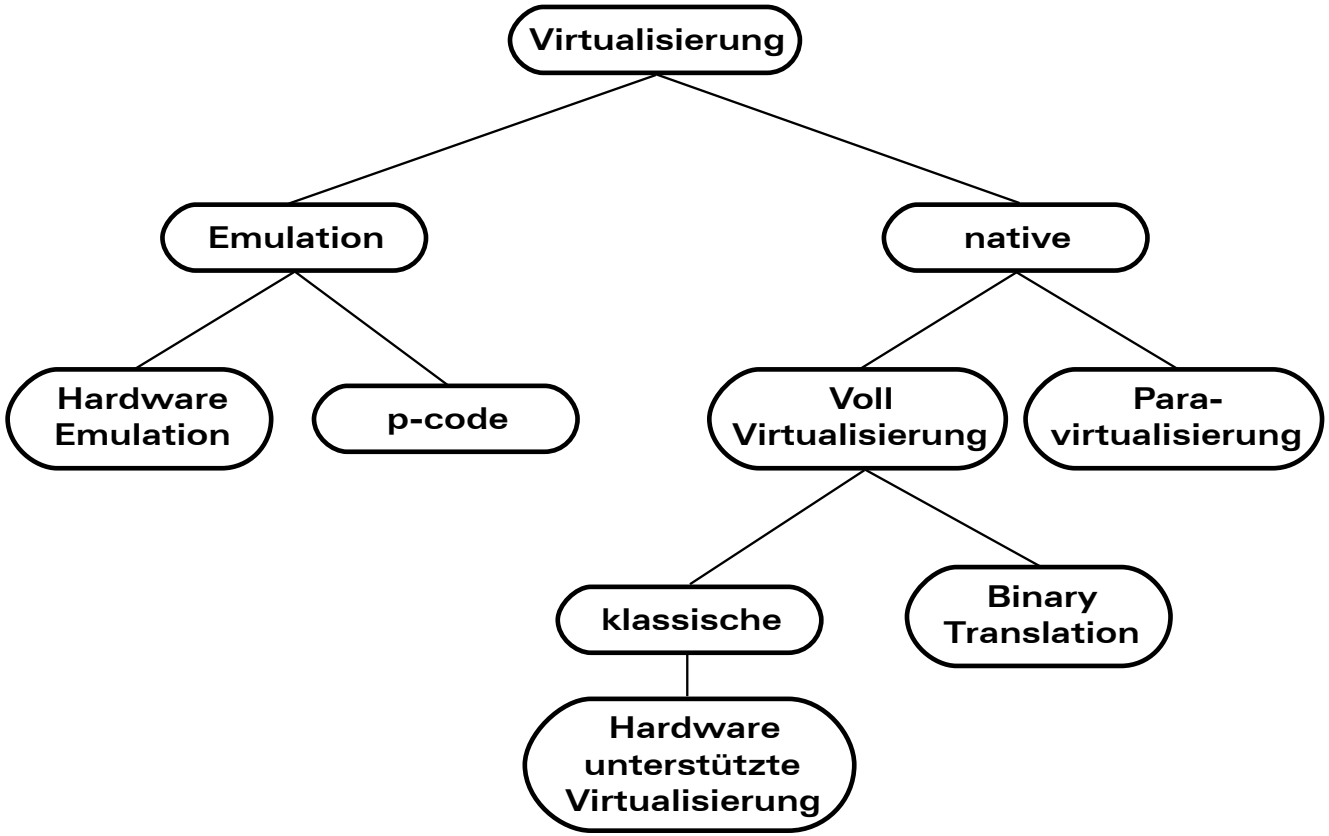
- Abschottung von Teilkomponenten
- Anwendung mit unterschiedlichen Sicherheits- und Echtzeitanforderungen
- Einsatz unterschiedlicher Betriebssysteme

Überblick



Wir betrachten nur Systeme mit gleicher Architektur für Host- (Wirt-) und Gastsystem.

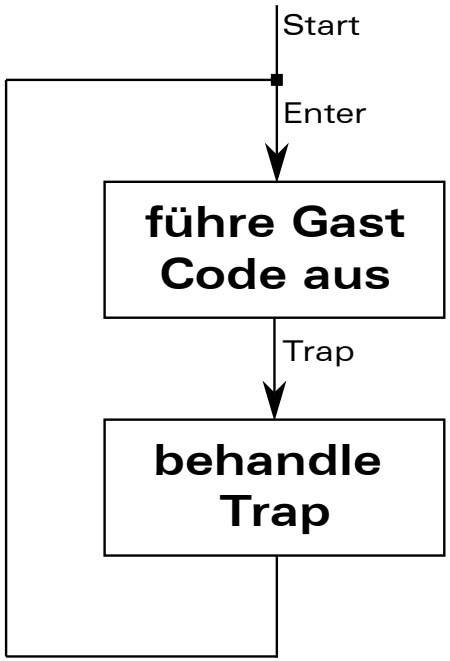
Überblick



Wir betrachten nur Systeme mit gleicher Architektur für Host- (Wirt-) und Gastsystem.

Hardware Anforderungen

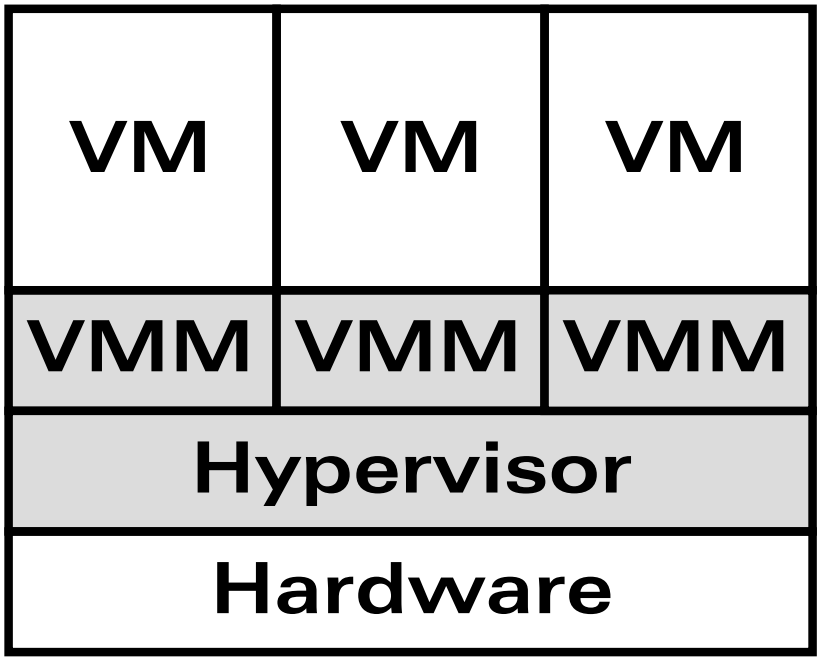
- 2 Betriebsmodi (Kernel-/Usermodus)
- Alle sensitiven Instruktionen müssen privilegiert sein
- Zugriff auf nicht zugewiesene Ressourcen und Ausführen privilegierter Instruktionen im Usermodus müssen einen Trap hervorrufen.



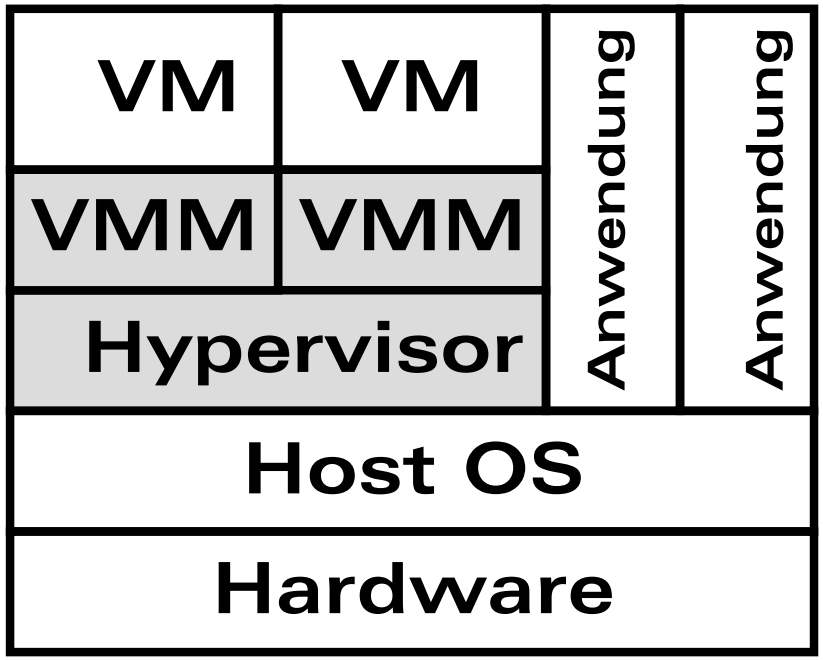
Hardware-unterstütze Virtualisierung

- Architektur muss vorherige Anforderungen erfüllen.
- Neuer Betriebsmodus (Hypervisor)
- neue Instruktionen zum Verwalten einer VM
- In-Memory Datenstrukturen
- Vollständige Kontrolle des Gastsystems
  - Gerätezugriff
  - Interruptbehandlung

Aufbau



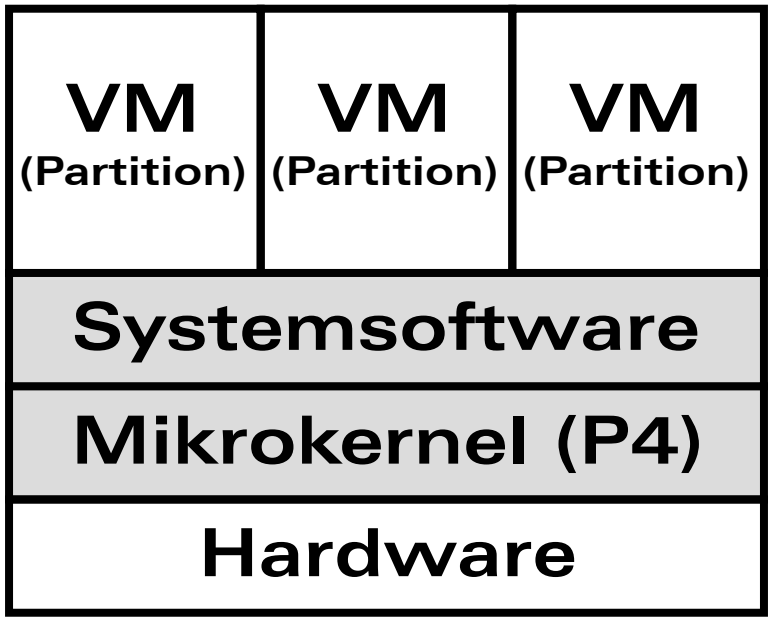
Type I



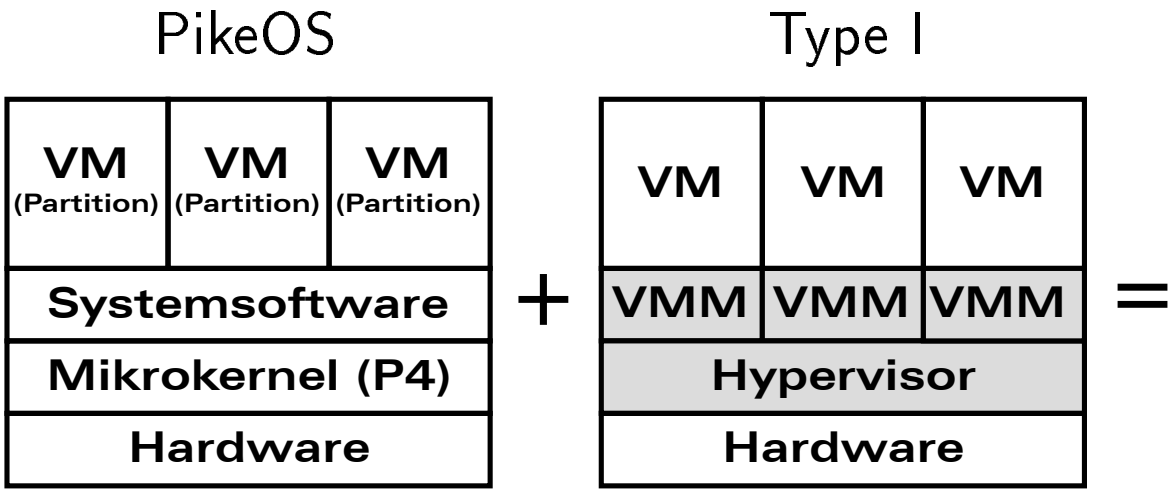
Type II

PikeOS

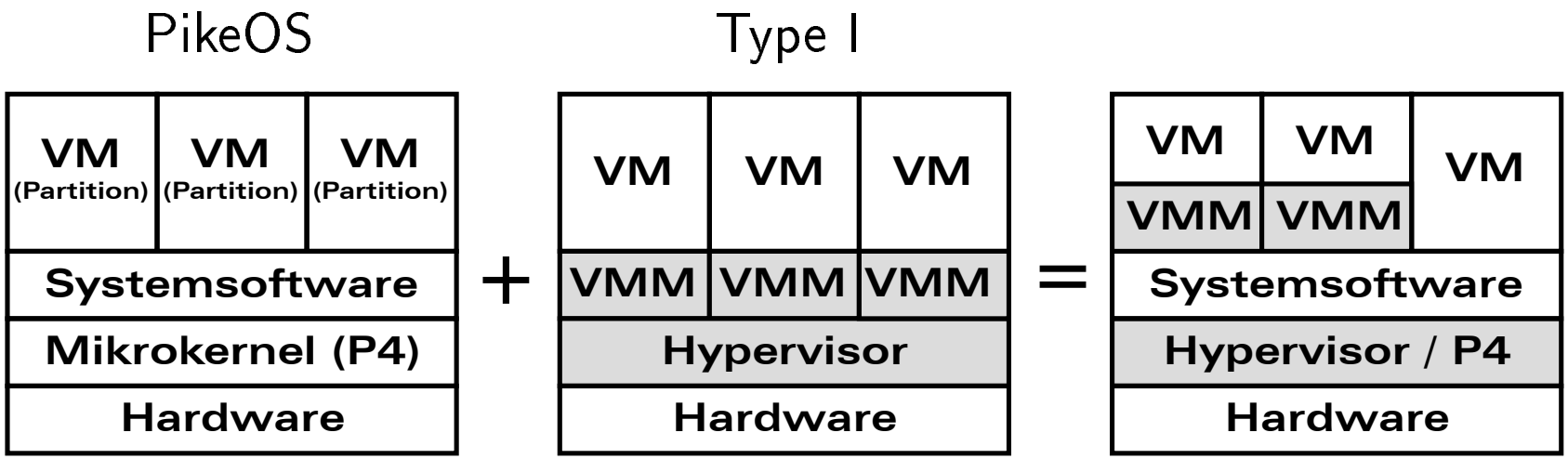
- Mikrokernel basiertes Echtzeitbetriebssystem
- Anwendungen laufen in isolierten Partitionen
- Bietet Support für Paravirtualisierung (Linux)



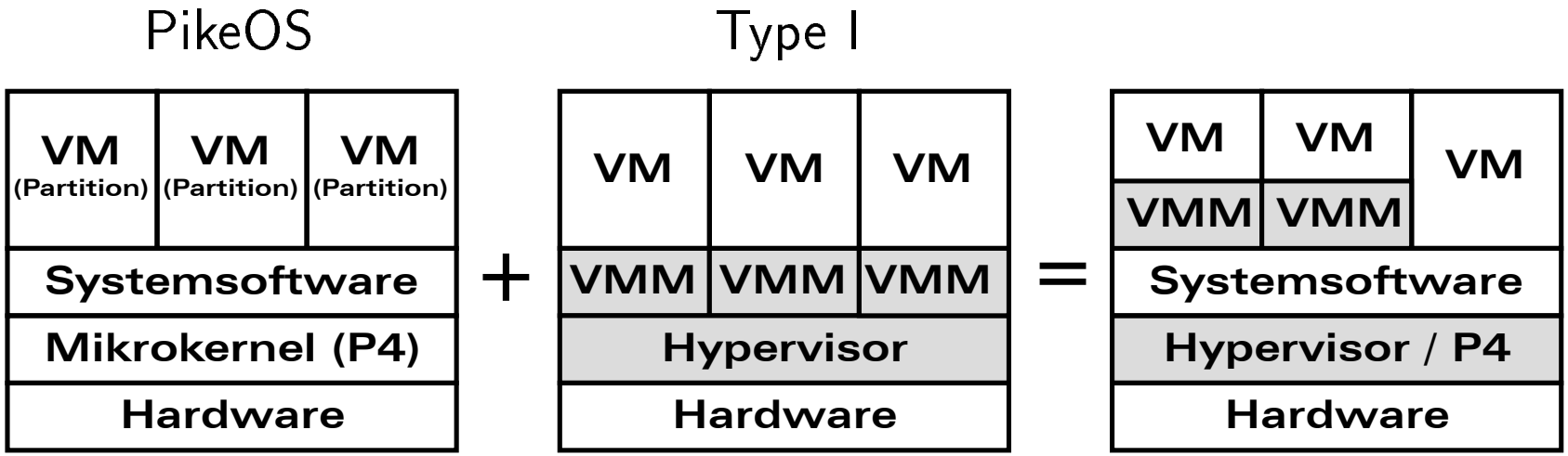
Zielsetzung



Zielsetzung



Zielsetzung



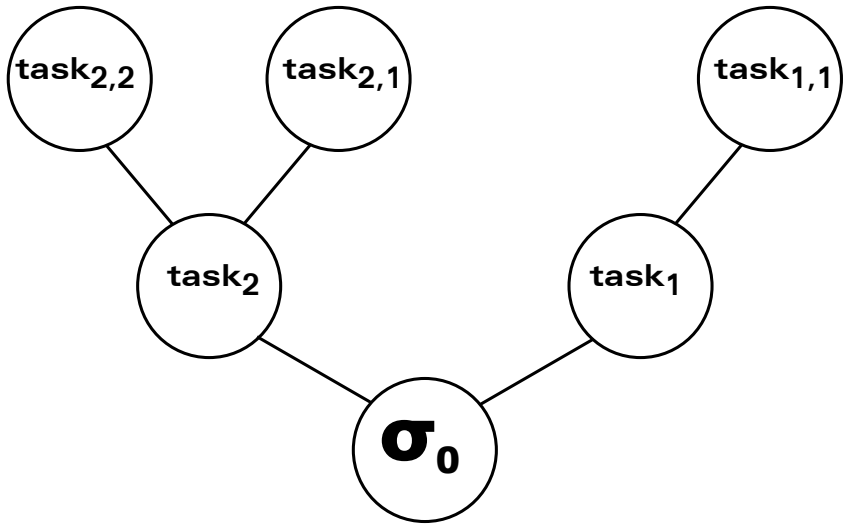
- Vollvirtualisierung aufbauend auf HuV
- Erweiterung von PikeOS zum Type I Hypervisor
- Minimale Änderungen am bestehenden Code
- Übernahme von bestehenden Konzepten
- Hardwareunabhängiges Design
- Nach Möglichkeit Hardwarefunktionen verwenden

Resultat

- eine VMM Implementation für jede Architektur
- Kernel bietet HW Abstraktion (verschiedene HuV Implementierungen innerhalb einer Architektur sind für den VMM transparent)
- Speichervirtualisierung im Kernel
- externe Interrupts werden ausschließlich vom Hostsystem behandelt.

Speicherrepräsentation

- Adressraum als P4 Task
- P4 Taskbaum (hierarchische Weitergabe von Ressourcen)
- **Einschränkung:** P4 Kernel im oberen Adressraum
  $\Rightarrow$ Modifikation oder neuer Ansatz
  - Modifikation greift in gut optimierten und getesteten Code ein
  - neuer Ansatz kann Hardwarefunktionen besser nutzen

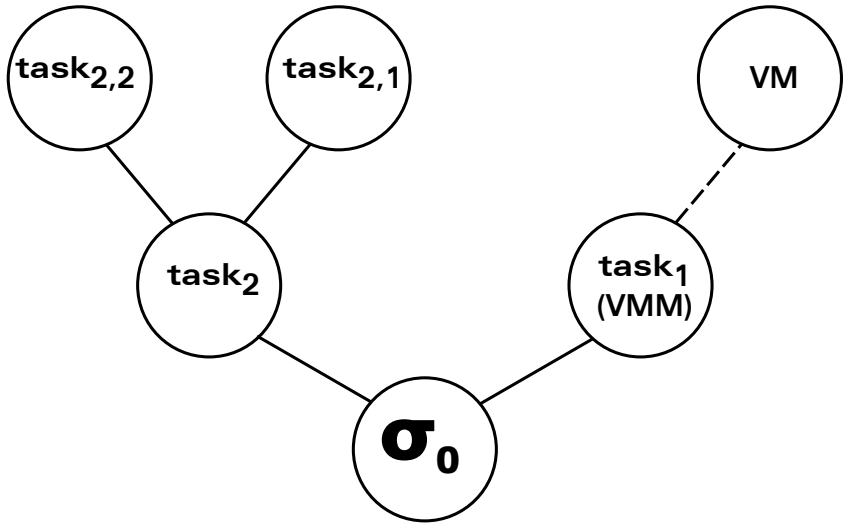


neuer Ansatz

- Kernel verwaltet virtuellen Adressraum der VM
- VMM konfiguriert Adressraum der VM
- VMM kann nur eigene Ressourcen weitergeben

$\Rightarrow$ P4 Ansatz bleibt erhalten!

Kernel kann wahlweise HW-/SW Funktionalität für die Speicheradressübersetzung nutzen



virtuelle CPU

Relation: $Thread \leftrightarrow Task \Leftrightarrow vCPU \leftrightarrow VM$
Bedingung: Ein Thread gehört zu einem Task und darf nur auf dessen Betriebsmittel zugreifen.

- vCPU wird durch Thread dargestellt
- vCPU Thread gehört zur VMM Task, weil VM nicht als eigenständiger Task implementiert ist
- vCPU Thread greift nur auf Betriebsmittel der VM zu
- Die Menge der Betriebsmittel der VM ist eine Teilmenge der Betriebsmittel des VMM

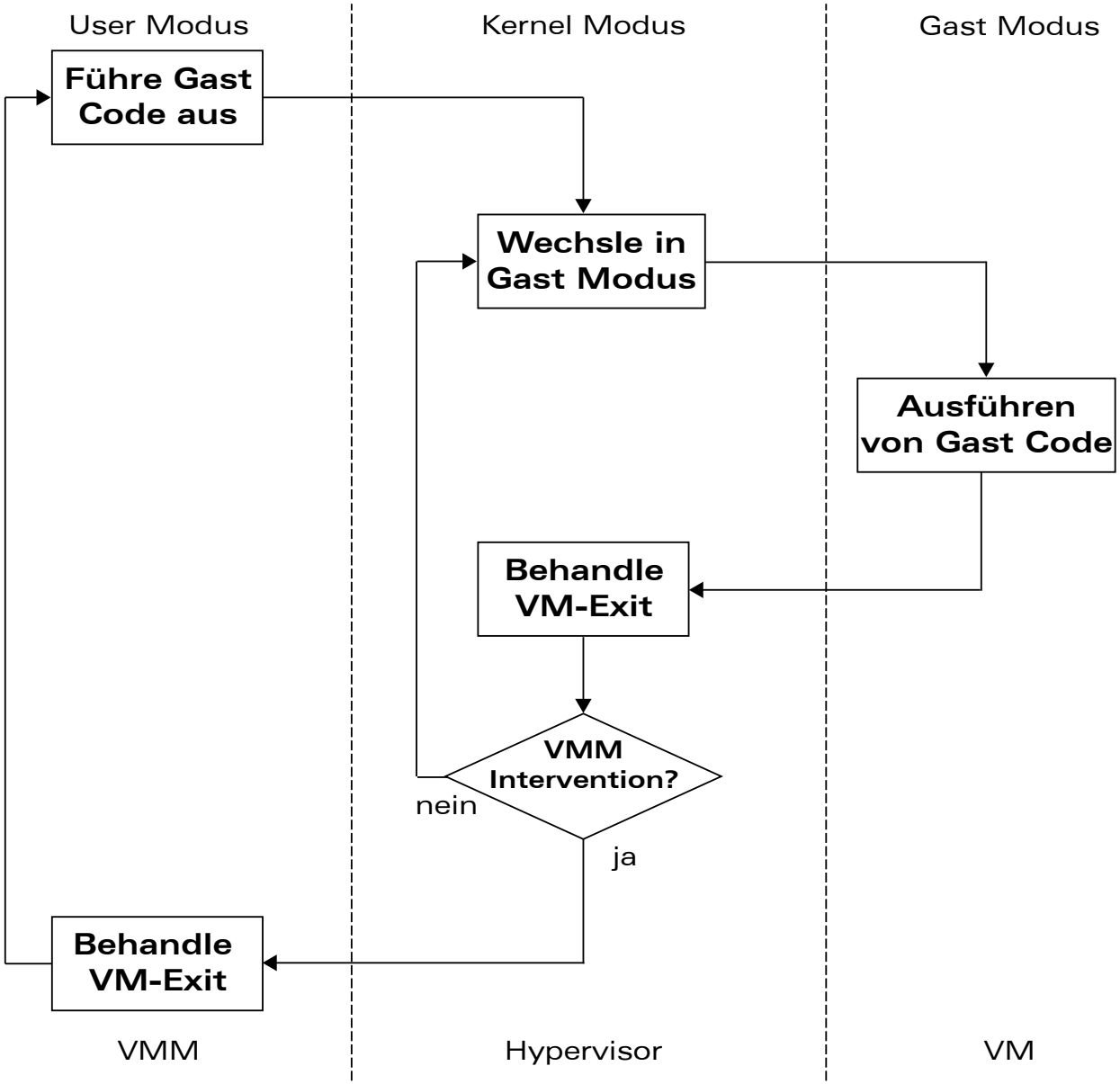
Interface

Anforderung

- gleiches Interface für alle Architekturen
- Zugriff auf Register der virtuellen CPU
- Verwalten der Ressourcen
- Eventinjektion

Datenaustausch über Shared Memory

- Informationsfluss vom Kernel in den User Space
- Ausnahme: VMM kann Register der vCPU ändern
- Es werden nur Daten abgelegt, die
  - nicht von der HW gesichert werden
  - auf die häufig zugegriffen wird


User Modus
Kernel Modus
Gast Modus
Führe Gast Code aus
Wechsle in Gast Modus
Ausführen von Gast Code
Behandle VM-Exit
VMM Intervention?
nein
ja
Behandle VM-Exit
VMM
Hypervisor
VM

Änderungen am bestehenden Code

Erweiterung:

- des Interfaces
- der HW-Initialisierung
- der Kernel Daten Strukturen
  - Globale Daten - systemweit gültig für alle VMs
  - VM bezogen - Setup der VM (z.B. Betriebsmittel)
  - vCPU bezogen - Register, etc.
- des Task / Thread Kontextes um jeweils einen Pointer zu den VM / vCPU bezogenen Daten

Details

- Verschiedene Implementierungen innerhalb einer Architektur
  - Traditionell: verschieden PSPs
  - Besonderheit der x86 Architektur: nur ein PSP nötig
  - keine unterschiedlichen PSPs für Virtualisierung gewünscht
  - Funktionstabelle, die je nach Hardwarehersteller initialisiert wird
- vCPUs werden durch ThreadID identifiziert, VMs durch TaskID
- Speichervirtualisierung als virtueller TLB

Aktueller Stand

- Design für den Hypervisor
- Implementierung für die grundlegenden CPU Funktionen
- einfachen VMM, incl.
  - Bootloader
  - Timer
  - Interrupt Controller
  - serielle Schnittstelle
- Linux ist lauffähig
  - 16-bit Startcode durch statisches Setup ersetzt
  - Zugriff auf Low Level Devices ausgeschaltet

Aktueller Stand

- Design für den Hypervisor
- Implementierung für die grundlegenden CPU Funktionen
- einfachen VMM, incl.
  - Bootloader
  - Timer
  - Interrupt Controller
  - serielle Schnittstelle
- Linux ist lauffähig
  - 16-bit Startcode durch statisches Setup ersetzt
  - Zugriff auf Low Level Devices ausgeschaltet

Aktueller Stand

- Design für den Hypervisor
- Implementierung für die grundlegenden CPU Funktionen
- einfachen VMM, incl.
  - Bootloader
  - Timer
  - Interrupt Controller
  - serielle Schnittstelle
- Linux ist lauffähig
  - 16-bit Startcode durch statisches Setup ersetzt
  - Zugriff auf Low Level Devices ausgeschaltet

Ausblick

- Vervollständigen der Hypervisorimplementierung
- Ausführen von 16-bit Code
- VMM Evaluation
  - Eigenentwicklung
  - bestehende Software (z.B. Qemu)
- Speichervirtualisierung
- Echtzeitanforderungen
- Evaluation des Interfaces

Zusammenfassung

- Erweiterung von PikeOS für die HuV
- Design für den Hypervisor
- Implementation