Systemstruktur und Echtzeitverhalten - Analyse und Vergleich von Windows CE 5.00 und 6.00

von Erik Merkel

Gliederung

1. Motivation
2. Allgemeines zu Windows CE
3. Systemstruktur
4. Unterschiede zwischen CE 5 und CE 6
5. Leistungsvergleich
6. Fazit

Motivation

- Kernel Quellcode frei verfügbar, aber nur Schnittstellen sind dokumentiert
- Es gibt kaum Erkenntnisse über interne Arbeitsweise von CE
- Mit CE 6 wurde neues Prozess- und Speichermodell eingeführt
- Keine Informationen über Einfluss auf Leistung und Echtzeitfähigkeit des neuen Prozessmodells
- Ziel der Arbeit:
  - Dokumentation der Kernelstruktur
  - Analyse der Änderungen zwischen CE 5 und 6
  - Evaluation beider Versionen auf unterschiedlichen Hardware-Plattformen

1. Motivation
2. **Allgemeines zu Windows CE**
3. Systemstruktur
4. Unterschiede zwischen CE 5 und CE 6
5. Leistungsvergleich
6. Fazit

Historisches zu CE

- 1996 auf den Markt gekommen
- Ursprüngliche Zielgruppe: Kleinstcomputer mit graphischem Interface (PDAs, Mobiltelefone, Pocket PCs)
- Designziele:
  - Optimierung für sparsamen Umgang mit Betriebsmitteln
  - Flexibilität durch Modularisierung und Konfigurierbarkeit
  - Gleiches Look & Feel der GUI wie Desktop Windows
  - Kompatibilität mit Win32-API
- Optimierung für Echtzeit seit Version 3
  - Anpassung des Schedulers zur Behandlung von Prioritätsinversionen
  - Entfernung von Schleifen mit variabler Laufzeit auf allen kritischen Pfaden im Kern
  - Erweiterung der Prioritätsstufen

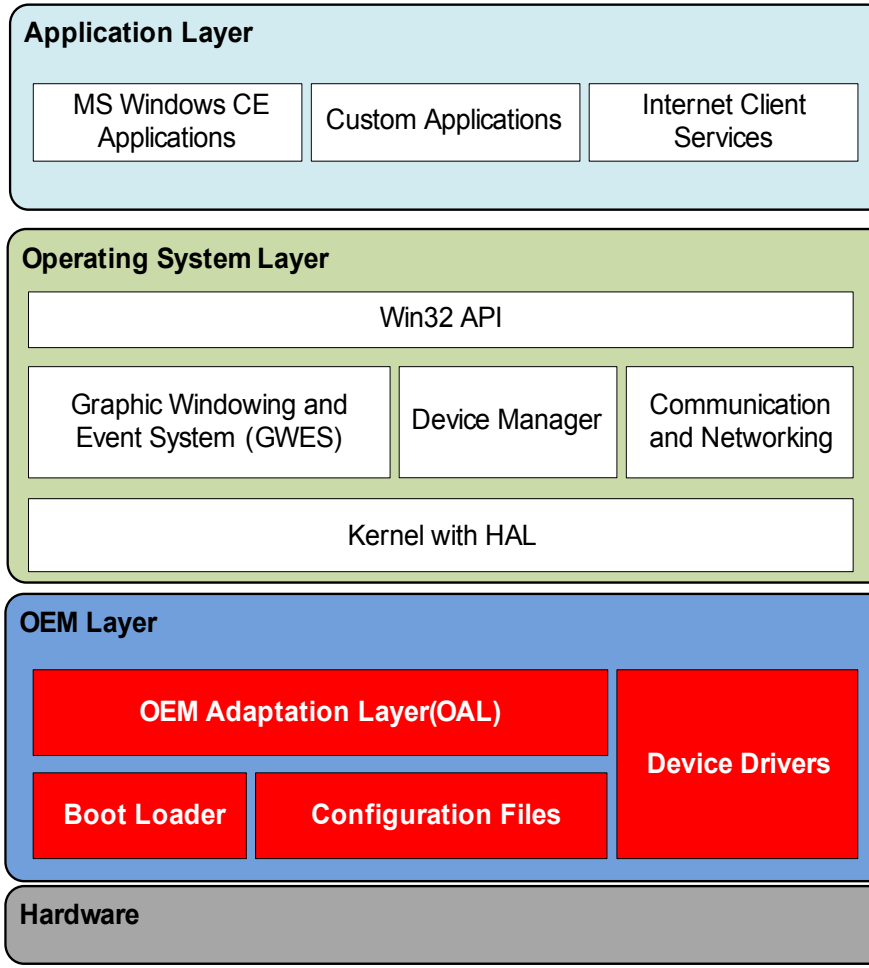
Unterstützte Hardware

- Nur für 32-Bit CPUs
  - Unterstützt werden ARM, MIPS, SHx und x86 CPUs
  - Keine Multiprozessorunterstützung
  - CPU-Unterstützung nicht von OEMs erweiterbar
- MMU zwingend erforderlich für Speicherverwaltung
  - Page-Größe 4 KB
  - Kein Swapping, aber memory mapped files
  - Maximal 512 MB physikalischer Speicher können von CE verwaltet werden
- Vielzahl von Gerätetreibern mitgeliefert, teilweise im Quellcode
- On-Target Debugging über Ethernet, USB oder RS232

1. Motivation
2. Allgemeines zu Windows CE
3. **Systemstruktur**
4. Unterschiede zwischen CE 5 und CE 6
5. Leistungsvergleich
6. Fazit

Systemstruktur

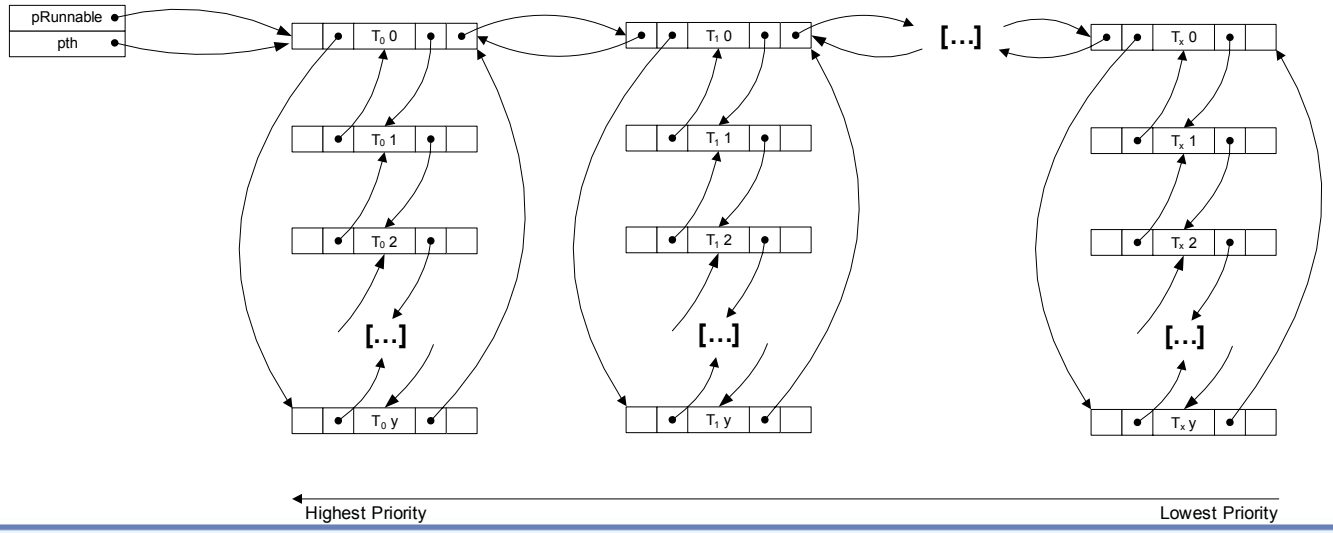
- Betriebssystemschicht:
  - Hardware-Abstraktionsschicht
  - Generischen Kernel für alle CPUs
  - Win32 API
- OEM Schicht:
  - Von OEMs bereitgestellt
  - Enthält plattformspezifische Anpassungen
  - Treiber von MS oder OEM
- OS- und OEM-Schicht ergeben Board Support Package (BSP)



Quelle: Microsoft [1

Scheduling

- Round-Robin-Algorithmus auf Basis von Thread-Prioritäten (256 Stufen)
- Thread-Datenstrukturen in doppelt verketteter Liste
- Hash-Tabelle mit 32 Einträgen zum schnellen Zugriff
- System-Tick variabel oder statisch (wenn statisch 1 ms)
- Thread-Quantum 100 ms (änderbar)
- 1-stufige Prioritätsvererbung bei Inversion
- Echtzeit-Klassifizierung nur über Priorität



Treiber in CE

Teilung der Interrupt-Behandlung in zwei Stufen:

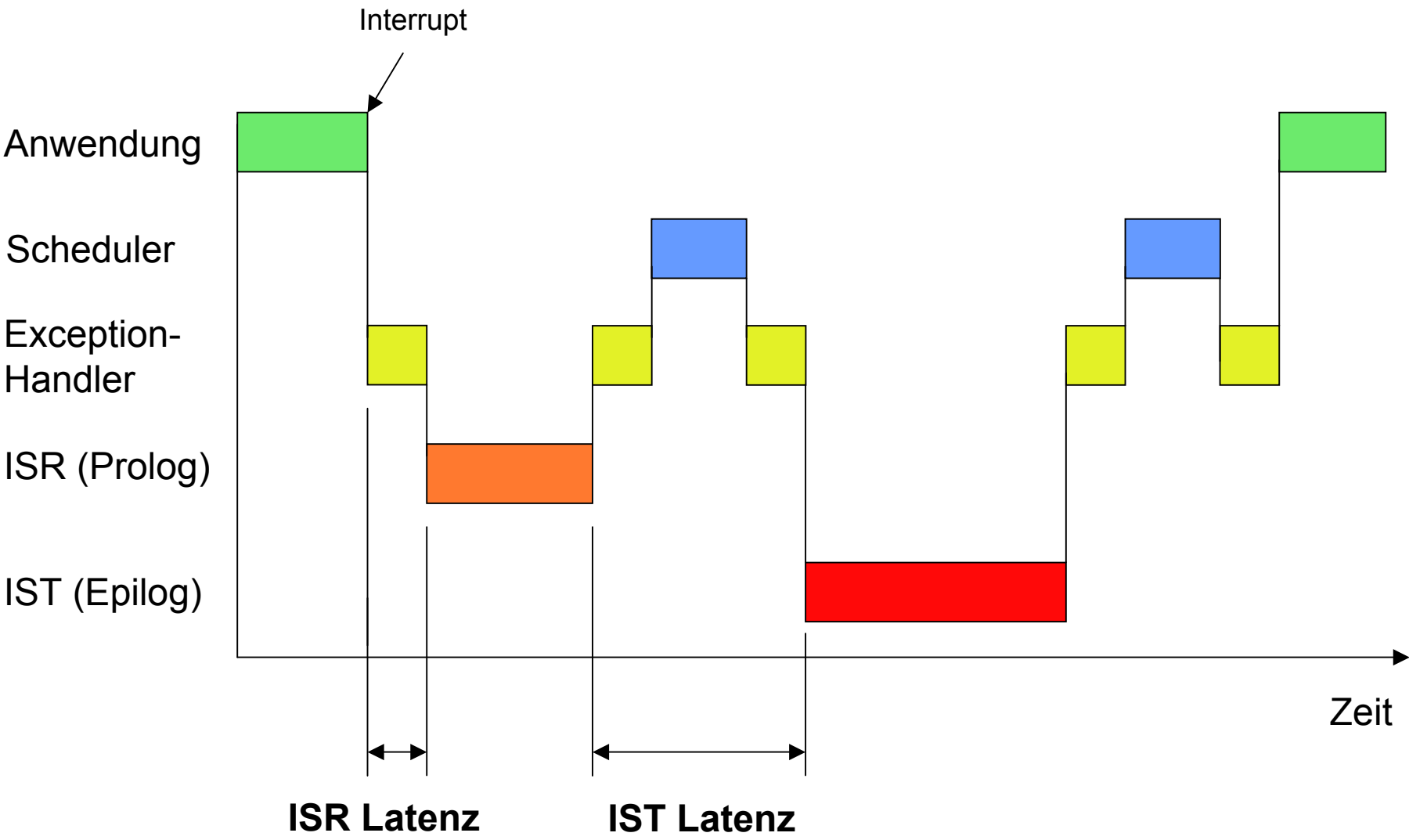
1. Stufe: Interrupt Service Routine (Prolog)

- Ausführung im Kontext des Exception-Handlers
- Unterbrechbarkeit architekturabhängig
- Ziel: Interrupt quittieren und Ereignis dem Scheduler mitteilen
- Gibt logische Interrupt Nummer (SYSINTR) zurück mit welcher der passende Interrupt Service Thread ausgewählt wird

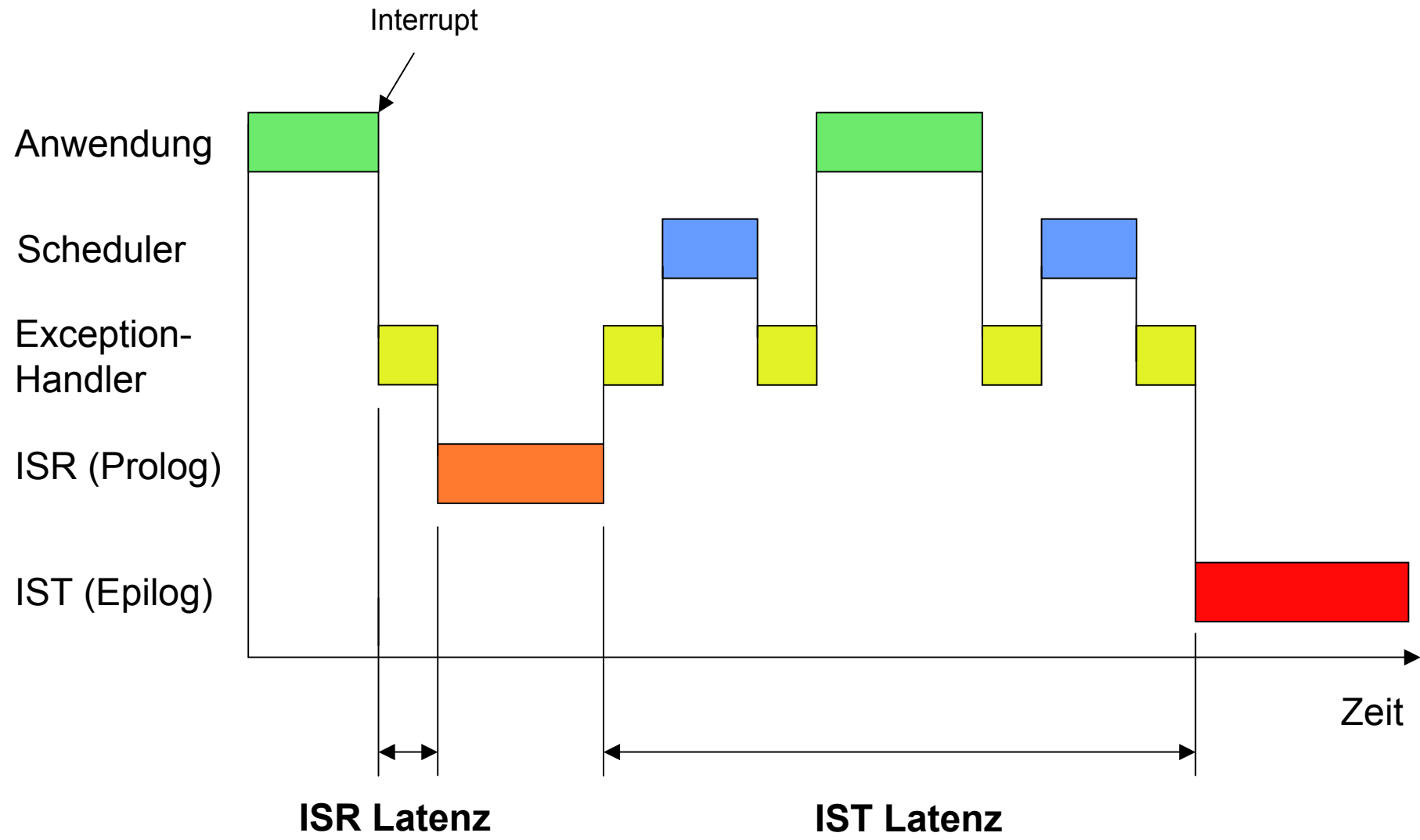
2. Stufe: Interrupt Service Thread (Epilog)

- Eigentliche Ereignisbehandlung
- Ausführung im Kontext des *Device Managers*
- Gewöhnlicher Thread mit Priorität

Interrupt-Behandlung (1)

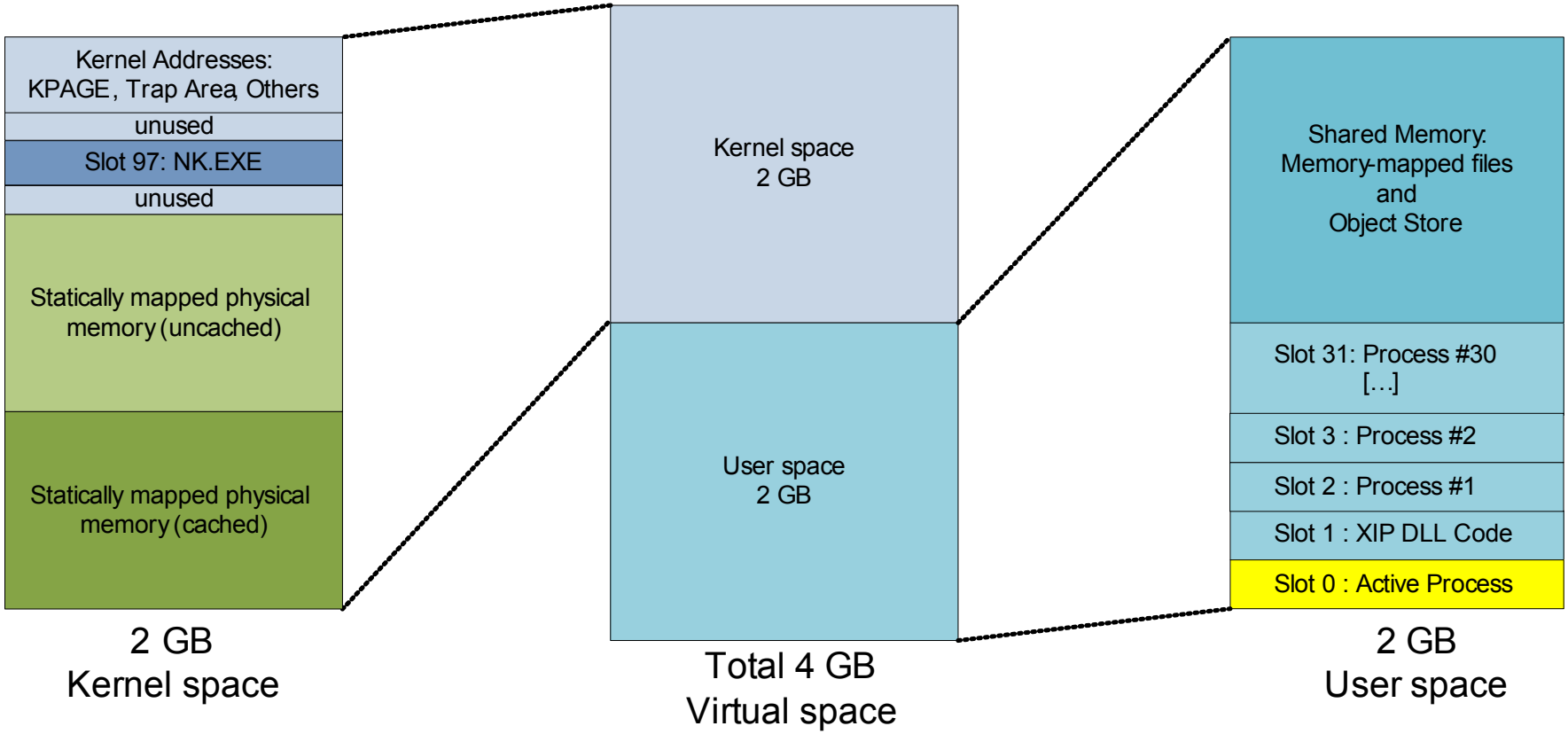


Interrupt-Behandlung (2)

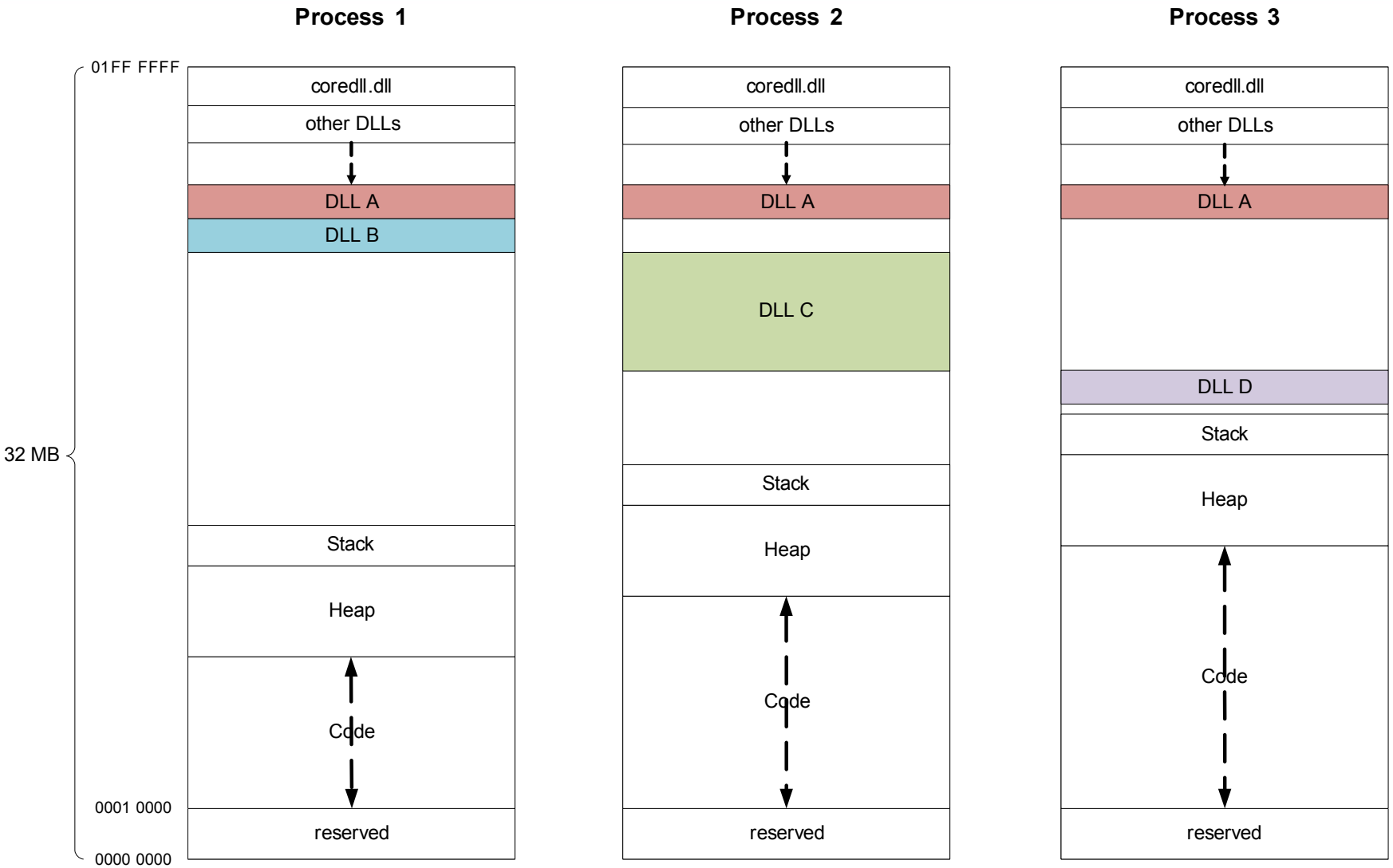


1. Motivation
2. Allgemeines zu Windows CE
3. Systemstruktur
4. **Unterschiede zwischen CE 5 und CE 6**
5. Leistungsvergleich
6. Fazit

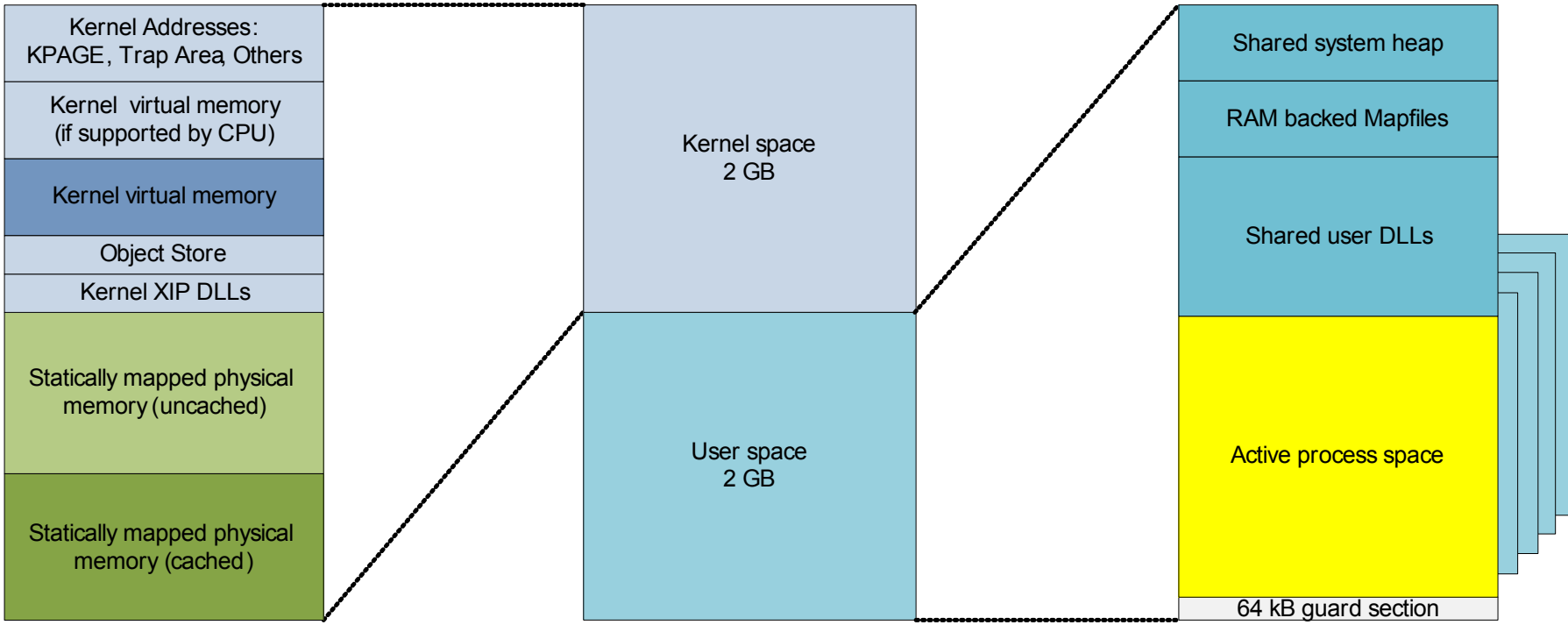
Virtueller Adressraum in CE 5



Das DLL Problem in CE 5



Virtueller Adressraum in CE 6



Änderungen bei Treibern

- *Device Manager* ist in CE 5 ein eigenständiger Prozess (Usermode Treiber)
  - Folge: Kontextwechsel zum *Device Manager* Prozess bei jedem Ereignis
- *Device Manager* ist Bestandteil des Kernels in CE 6 (Kernelmode Treiber)
  - Folge: kein Kontextwechsel mehr nötig
- Usermode Treiber mit eigenem Prozess optional möglich in CE 6
  - Kontextwechsel wieder nötig, aber viel teurer als in CE
  - Nur eingeschränkter Zugriff auf Betriebsmittel

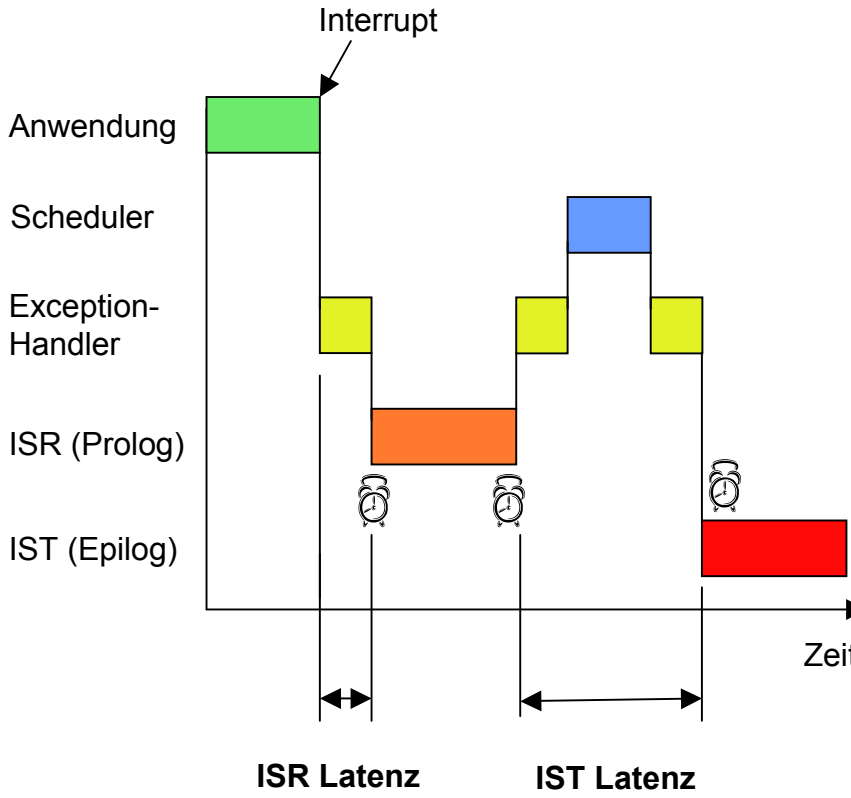
1. Motivation
2. Allgemeines zu Windows CE
3. Systemstruktur
4. Unterschiede zwischen CE 5 und CE 6
5. **Leistungsvergleich**
6. Fazit

Testkonzept

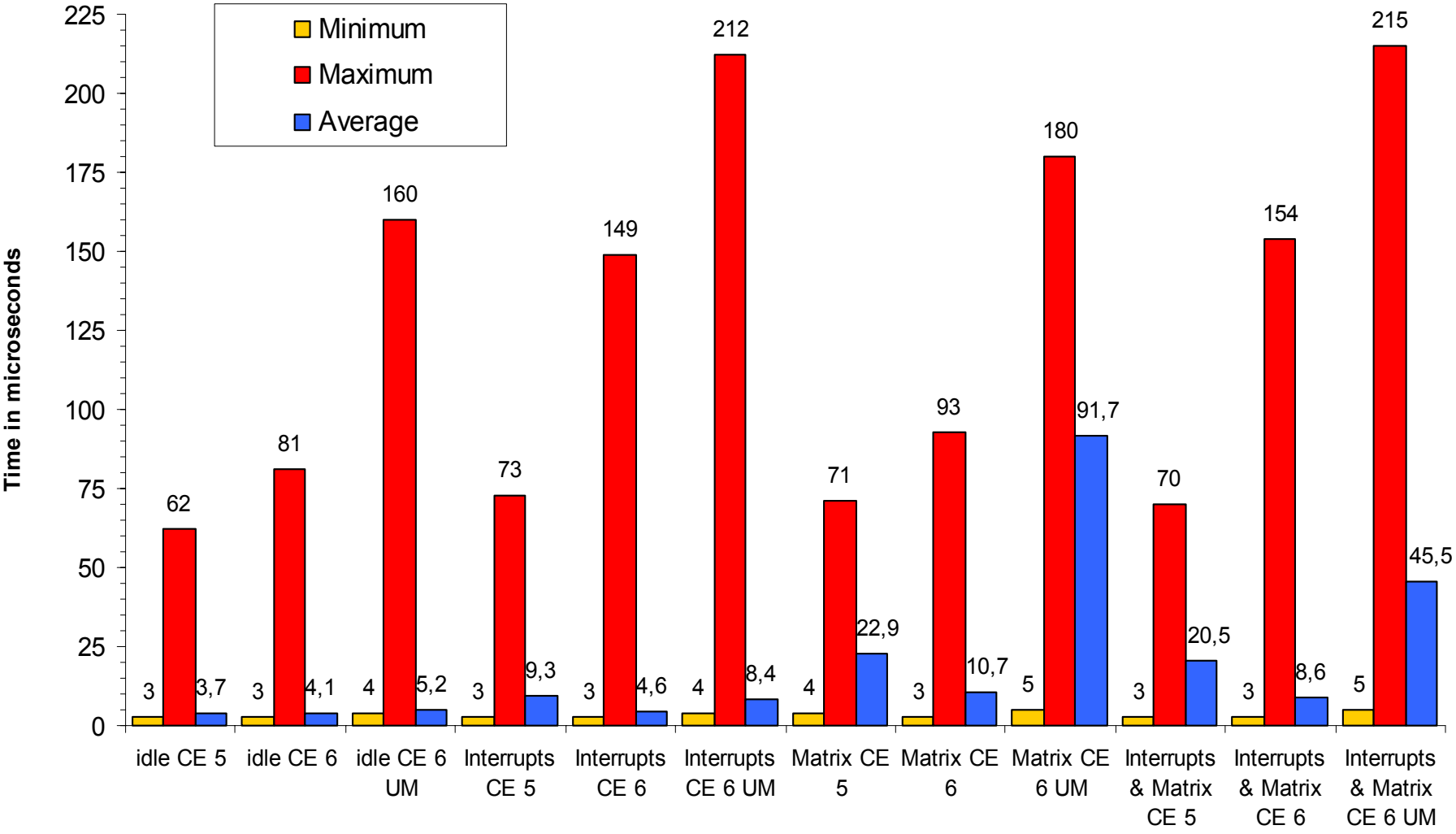
- **Was soll getestet werden:**
  - Messung der Behandlungslatenzen in den einzelnen Treiberstufen
  - Analyse der Datenaustausch-Primitive zwischen Treiber und Anwendung
  - Zeit für Kontextwechsel bei Signalisierung von Ereignissen
- **Wie wird gemessen:**
  - Laufzeiten werden mit Hilfe eines CPU-Timers bestimmt
  - Interrupts werden vom CPU-Timer ausgelöst
  - Messungen auf einer ARM- und einer MIPS-Plattform

Latenzmessung

- CPU-Timer löst Interrupt aus
- Timer so programmiert, dass Zähler bei Interrupt auf Null gesetzt wird
- Auslesen der Zähler in ISR und IST
- Abfragen des ISR-Zählerstands über Kernel IOControl
- Speichern der Zählerwerte in IST
- Offlineauswertung der Messergebnisse
- Wiederholung der Messreihe mit verschiedenen Lastprofilen

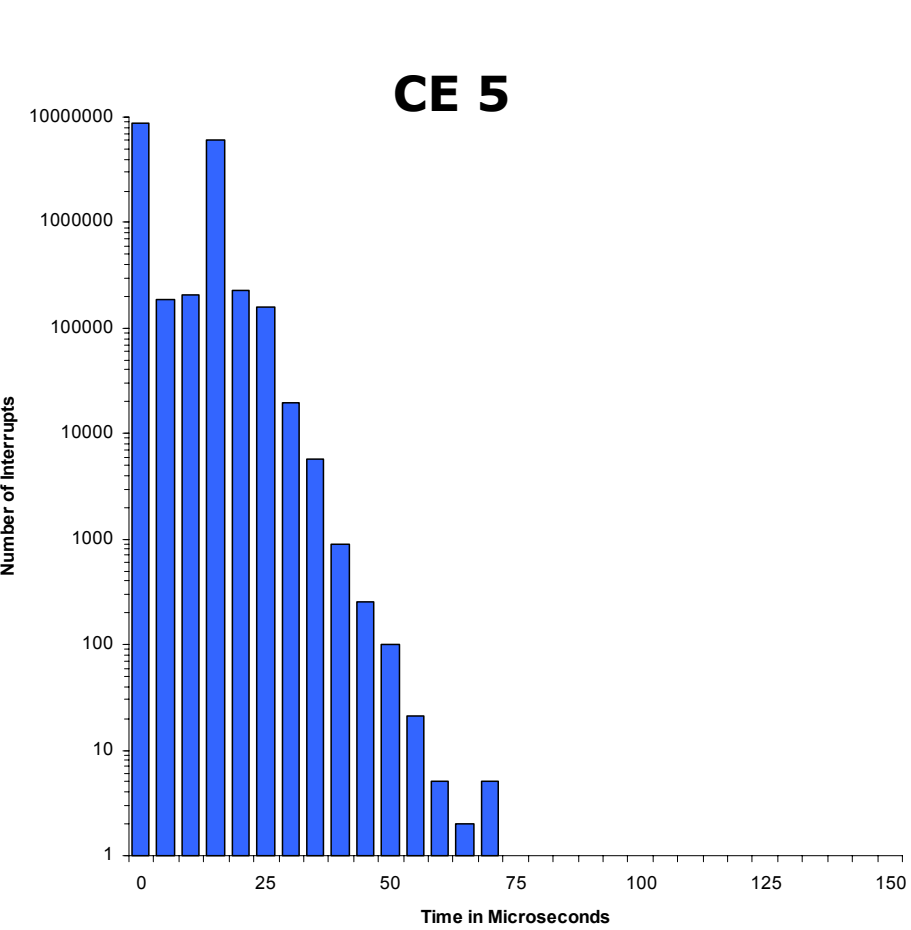


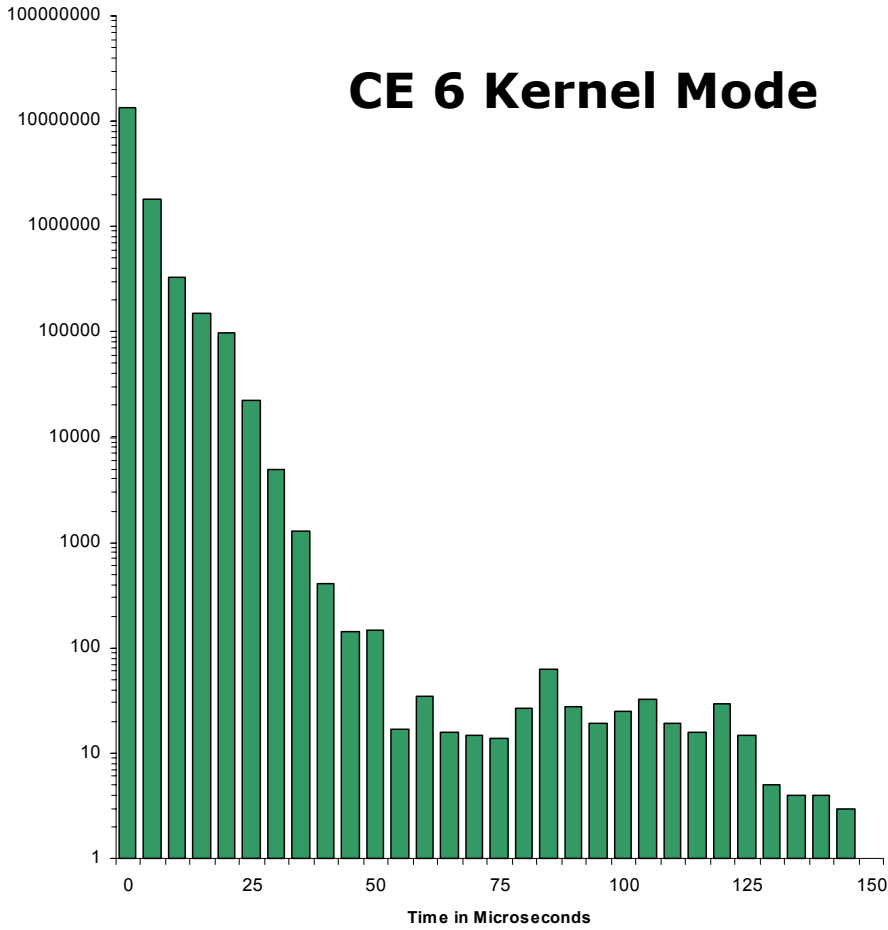
IST Latenzen (ARM)



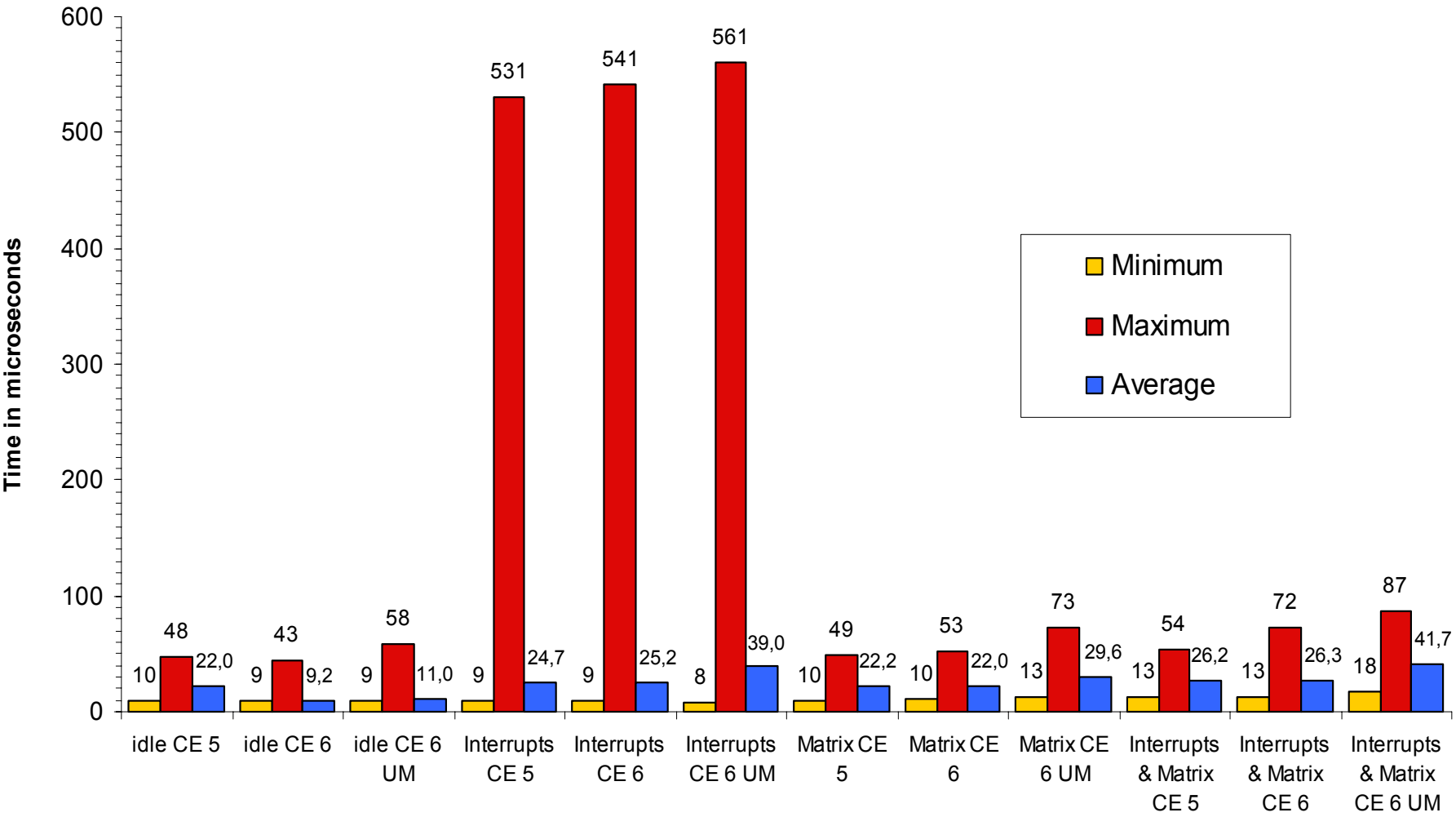
Verteilung IST Latenzen (ARM)

- Histogramm der IST-Verteilung bei zusätzlicher Interruptquelle





IST Latenzen MIPS



Datentransfer-Strategie

- **CE 5:**
  - Pointer zeigen immer auf Strukturen in Slot 0
  - Verdrängte Prozesse befinden sich in Slots 2-31
  - Pointer werden auf den Prozess Slot umgeleitet zu dem sie gehören und Zugriffsrechte angepasst
- **CE 6:**
  - Kernel-Mode Treiber können übergebene Zeiger direkt benutzen (synchroner Zugriff)
  - Kernel blendet automatisch Speicher für User-Mode Treiber ein (page mapping)
  - Eingebettete Zeiger müssen durch System-Calls im User-Mode Treiber entpackt werden (Speicher wird kopiert)

Kontextwechsel

- **Messung erzwungener Kontextwechsel durch Event-Signalisierung:**
  - Event-Mechanismus wird auch verwendet, um Treiber über neue Interrupts zu informieren
  - Die Signalisierung erzeugt Overhead zusätzlich zum Kontextwechsel
  - Kontextwechsel können zwischen Threads eines Prozesses oder zwischen mehreren Prozessen stattfinden
  - Overhead von Prozesswechseln ist stark architekturabhängig
- **Messung:**
  - Ein Thread (Prozess) wartet auf ein Event
  - Ein zweiter Thread (Prozess) signalisiert das Event
  - Der Signalisierer wird schlafen gelegt (Verzicht auf restliches Thread-Quantum)
  - Der wartende Thread (Prozess) wird geweckt
  - In beiden Akteuren wird jeweils der CPU-Timer ausgelesen um die Dauer des Wechsels zu bestimmen
  - Messung mit und ohne CPU-Cache-Invalidierung

Kontextwechsel

Windows CE Version	Test Case	CPU Instruction Cache State	Latency on ARM	Latency on MIPS
CE 5	process switch	cached	19.9 µs	20.8 µs
		cache discarded	79.9 µs	42.2 µs
	thread switch	cached	5.0 µs	20.9 µs
		cache discarded	55.9 µs	42.0 µs
CE 6	process switch	cached	135.1 µs	30.0 µs
		cache discarded	150.1 µs	52.8 µs
	thread switch	cached	6.4 µs	30.0 µs
		cache discarded	70.1 µs	52.5 µs

1. Motivation
2. Allgemeines zu Windows CE
3. Systemstruktur
4. Unterschiede zwischen CE 5 und CE 6
5. Leistungsvergleich
6. **Fazit**

Fazit

- Laufzeitverhalten:
  - Deutliche Verschlechterung der Worst-Case Interrupt Latenzen
  - Verbesserung der durchschnittlichen Interrupt Latenz
  - Prozess-Kontextwechsel sind je nach Plattform um Größenordnungen teurer
- Speicherverwaltung:
  - Keine Beschränkungen mehr auf sehr kleine virtuelle Adressräume pro Prozess
  - DLL Problem entschärft
- Usermode Treiber
  - Nur sinnvoll bei E/A-armen Aufgaben
  - Wenn keine privilegierten CPU-Instruktionen benötigt werden

Fragen oder

Anmerkungen?

Quellen

- Grafiken:

[1]: Microsoft Corp. Windows CE Architecture. http://msdn2.microsoft.com/en-us/library/ms905093.aspx, 2007.