Optimierung von Cache Performance und Vorhersagbarkeit

Verbesserung der internen Struktur und Management des Cache Verhaltens in Echtzeitsystemen mittels verschiedener Strategien

Vanessa Kern

Friedrich-Alexander Universität Erlangen-Nürnberg (FAU)

CS 4 - FAU

Echtzeitsysteme

Echtzeitsysteme

Voraussetzungen an Echtzeitsysteme

- Einhalten von Terminen
- Vorhersagbarkeit

Einleitung

Cache Management Strategien
- Cache Partitioning
- Cache Locking

Optimierung durch Scheduling
- Kontrollieren von Events
- Berechnung der Working Set Size (WSS)

Agenda

Cache Partitioning

- Idee: Aufteilung des Caches in Teile und Verteilung auf Prozessoren

Cache Partitioning

- Idee: Aufteilung des Caches in Teile und Verteilung auf Prozessoren

Operating System Components (OSC)

Cache Partitioning

- Idee: Aufteilung des Caches in Teile und Verteilung auf Prozessoren

Operating System Components (OSC)

- Aufteilung von Daten im Hauptspeicher in Komponenten

Cache Partitioning

- Idee: Aufteilung des Caches in Teile und Verteilung auf Prozessoren

Operating System Components (OSC)

- Aufteilung von Daten im Hauptspeicher in Komponenten
- Gemeinsame Daten zur Ausführung von Anwendungen

Cache Partitioning

- Idee: Aufteilung des Caches in Teile und Verteilung auf Prozessoren

Operating System Components (OSC)

- Aufteilung von Daten im Hauptspeicher in Komponenten
- Gemeinsame Daten zur Ausführung von Anwendungen
- **Granularität:** Vielfaches einer Cachezeile

Statisch vs. Dynamisch

- **Statisch**

Statisch vs. Dynamisch

Statisch

Statische Verteilung

Statisch vs. Dynamisch

Statisch

Statische Verteilung

\+ Bei geringer Anzahl an Prozessoren

Statisch vs. Dynamisch

Statisch

Statische Verteilung

+ Bei geringer Anzahl an Prozessoren
+ Mehrere Cachezeilen pro Prozessor

Statisch vs. Dynamisch

- **Statisch**

Statische Verteilung

+ Bei geringer Anzahl an Prozessoren
+ Mehrere Cachezeilen pro Prozessor

- **Dynamisch**

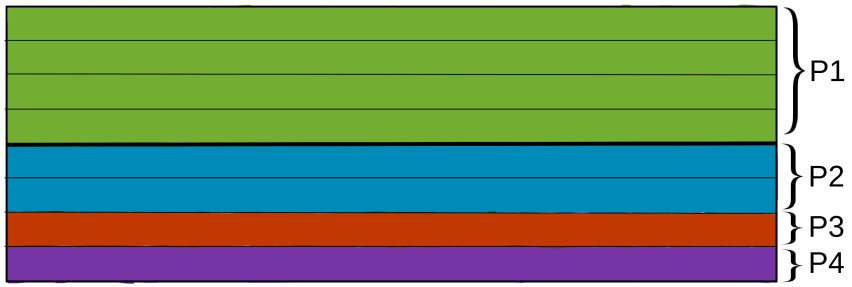
Statisch vs. Dynamisch

- **Statisch**

Statische Verteilung

+ Bei geringer Anzahl an Prozessoren
+ Mehrere Cachezeilen pro Prozessor

- **Dynamisch**



Dynamische Verteilung

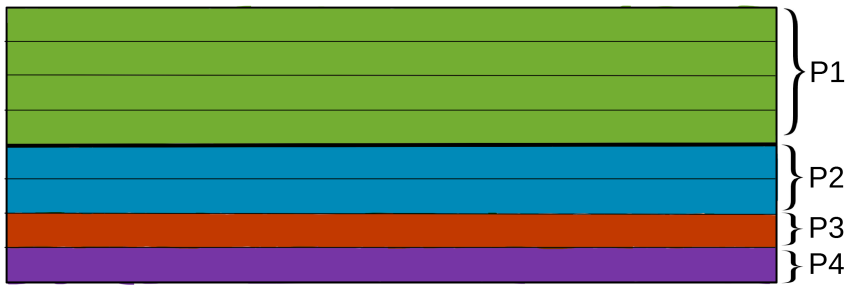
Statisch vs. Dynamisch

Statisch

Statische Verteilung

+ Bei geringer Anzahl an Prozessoren
+ Mehrere Cachezeilen pro Prozessor

Dynamisch



Dynamische Verteilung

+ Fair bei größerer Anzahl an Prozessoren

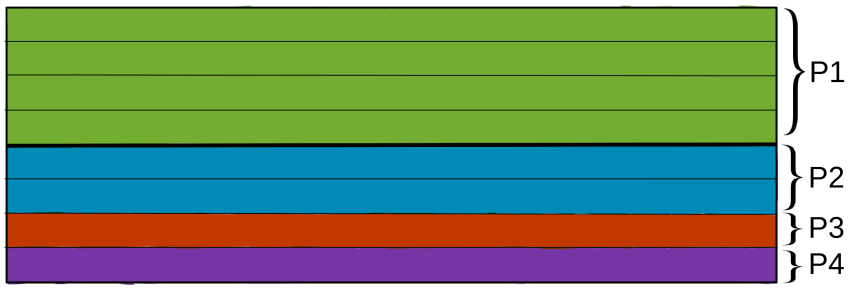
Statisch vs. Dynamisch

Statisch

Statische Verteilung

+ Bei geringer Anzahl an Prozessoren
+ Mehrere Cachezeilen pro Prozessor

Dynamisch



Dynamische Verteilung

+ Fair bei größerer Anzahl an Prozessoren
− Overhead für zusätzliche Berechnungen

Agenda

Einleitung

Cache Management Strategien
- Cache Partitioning
- Cache Locking

Optimierung durch Scheduling
- Kontrollieren von Events
- Berechnung der Working Set Size (WSS)

Cache Locking

Problem

Hohe Cache Miss Rate

Cache Locking

Problem

Hohe Cache Miss Rate

- **Idee:**
  - Blockieren einzelner Zeilen im Cache

Cache Locking

Problem

Hohe Cache Miss Rate

- **Idee:**
  - Blockieren einzelner Zeilen im Cache
  - Auswahl der Zeilen mit der höchsten Miss Rate

Cache Locking

Problem

Hohe Cache Miss Rate

- **Idee:**
  - Blockieren einzelner Zeilen im Cache
  - Auswahl der Zeilen mit der höchsten Miss Rate
  - Granularität:

Cache Locking

Problem

Hohe Cache Miss Rate

- **Idee:**
  - Blockieren einzelner Zeilen im Cache
  - Auswahl der Zeilen mit der höchsten Miss Rate
  - Granularität:
    - Speicherplatz kompletter Funktionen

Cache Locking

Problem

Hohe Cache Miss Rate

- **Idee:**
  - Blockieren einzelner Zeilen im Cache
  - Auswahl der Zeilen mit der höchsten Miss Rate
  - Granularität:
    - Speicherplatz kompletter Funktionen
    - Einzelne Speicher Blöcke

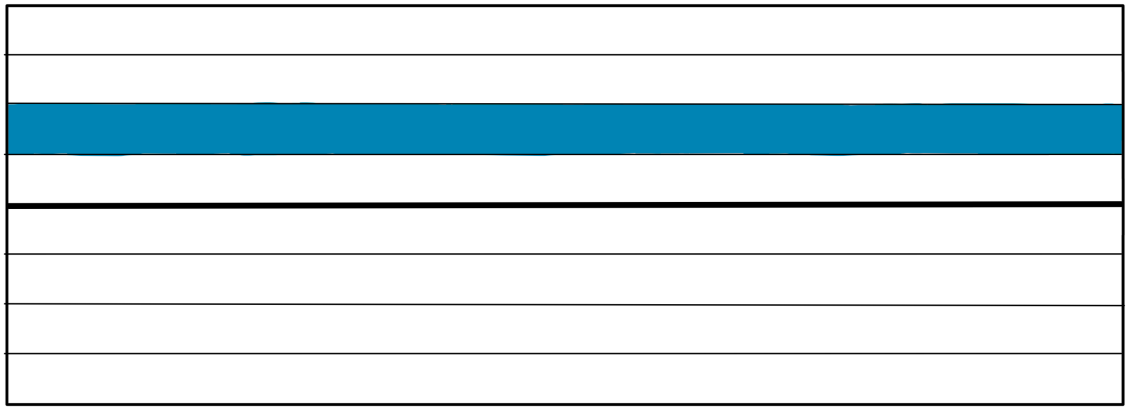
Cache Locking

Problem

Hohe Cache Miss Rate

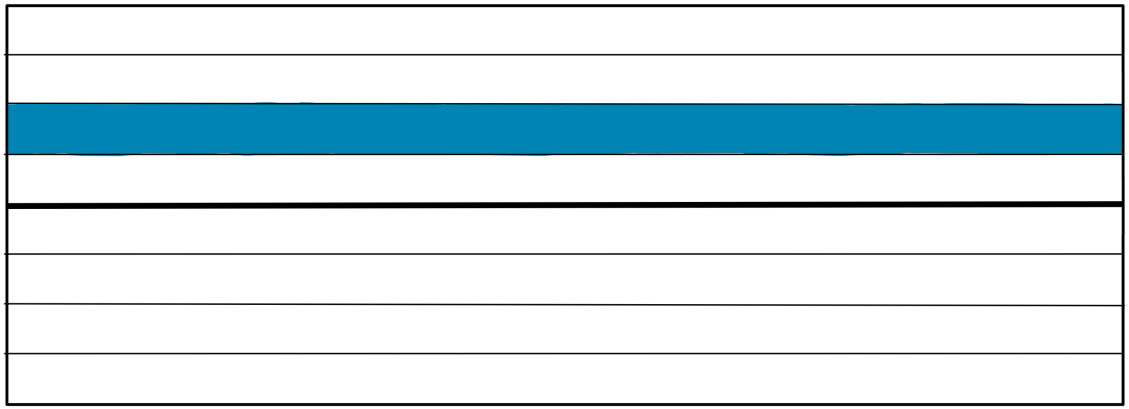
- **Idee:**
  - Blockieren einzelner Zeilen im Cache
  - Auswahl der Zeilen mit der höchsten Miss Rate
  - Granularität:
    - Speicherplatz kompletter Funktionen
    - Einzelne Speicher Blöcke
    - „Lockdown by way“

Cache Locking



Blockierte Cachezeile („lockdown by way“)

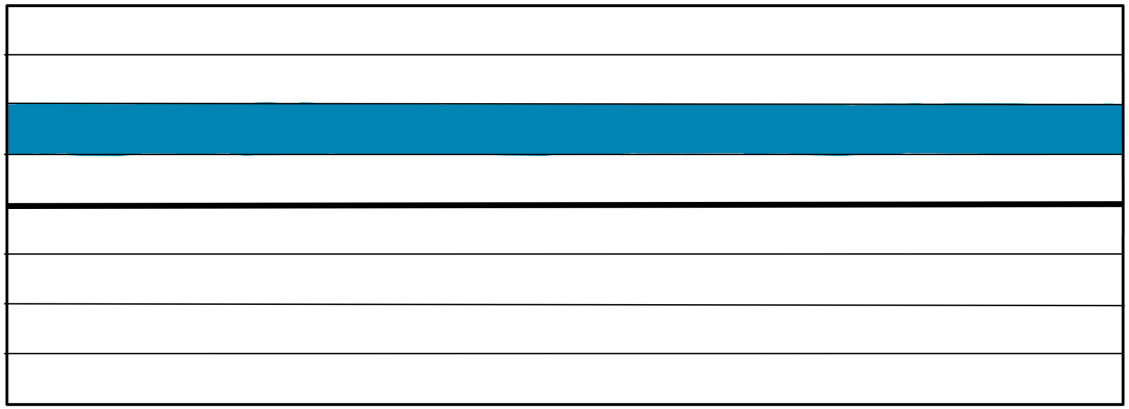
Cache Locking



Blockierte Cachezeile („lockdown by way“)

- Vorteile
  - Verringerung von Cache Misses

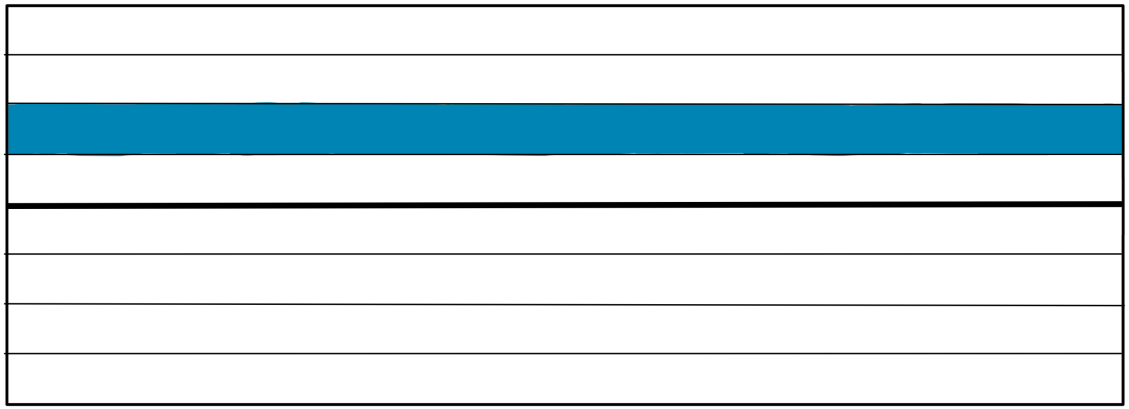
Cache Locking



Blockierte Cachezeile („lockdown by way“)

- Vorteile
  - Verringerung von Cache Misses
  - Energieeffizienz

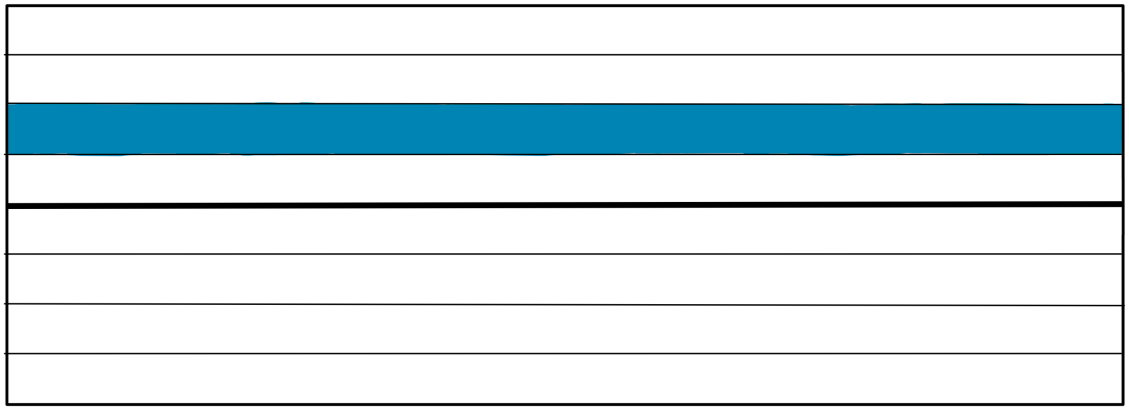
Cache Locking



Blockierte Cachezeile („lockdown by way“)

- Vorteile
  - Verringerung von Cache Misses
  - Energieeffizienz
  - Gesteigerte Vorhersagbarkeit

Cache Locking



Blockierte Cachezeile („lockdown by way“)

- Vorteile
  - Verringerung von Cache Misses
  - Energieeffizienz
  - Gesteigerte Vorhersagbarkeit
- Nachteil
  - Overhead bei komplett blockiertem Cache

Agenda

Kontrollieren von Events

- **Idee:**

Kontrollieren von Events

- **Idee:**
  - Sammlung von Daten in **Operating System Components (OSC)**

Kontrollieren von Events

- **Idee:**
  - Sammlung von Daten in **Operating System Components (OSC)**
  - Keine direkten Funktionsaufrufe zwischen OSCs

Kontrollieren von Events

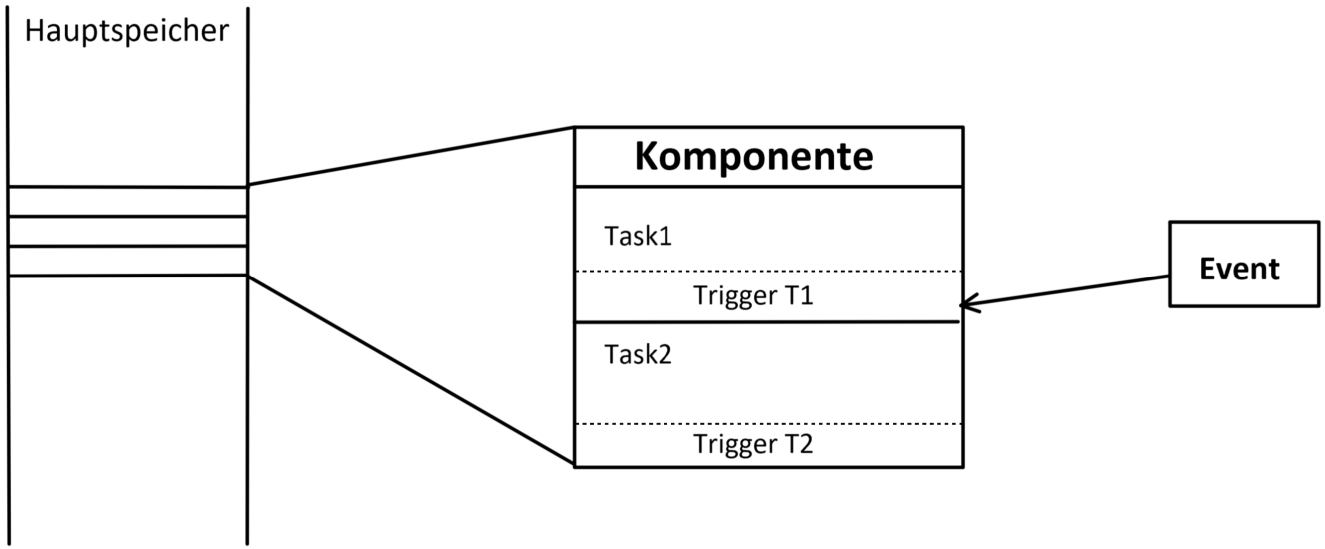
- **Idee:**
  - Sammlung von Daten in **Operating System Components (OSC)**
  - Keine direkten Funktionsaufrufe zwischen OSCs
  - Kontrolle der Kommunikation durch Events

Kontrollieren von Events

- **Idee:**
  - Sammlung von Daten in **Operating System Components (OSC)**
  - Keine direkten Funktionsaufrufe zwischen OSCs
  - Kontrolle der Kommunikation durch Events
  - Aufruf eines Triggers, der die Komponente ausführt

Kontrollieren von Events

- **Idee:**
  - Sammlung von Daten in **Operating System Components (OSC)**
  - Keine direkten Funktionsaufrufe zwischen OSCs
  - Kontrolle der Kommunikation durch Events
  - Aufruf eines Triggers, der die Komponente ausführt



Aufruf eines Triggers durch ein Event

Vor- und Nachteile

Wichtig

Vor- und Nachteile

Wichtig

- Ausführung nur eines Triggers pro Komponente

Vor- und Nachteile

Wichtig

- Ausführung nur eines Triggers pro Komponente
- Verwendung von Cache Locking und Cache Partitioning

Vor- und Nachteile

Wichtig

- Ausführung nur eines Triggers pro Komponente
- Verwendung von Cache Locking und Cache Partitioning

- Vorteile
  - Kontrolle über Inhalt des Caches

Vor- und Nachteile

Wichtig

- Ausführung nur eines Triggers pro Komponente
- Verwendung von Cache Locking und Cache Partitioning

- Vorteile
  - Kontrolle über Inhalt des Caches
  - Energieeffizienz

Vor- und Nachteile

Wichtig

- Ausführung nur eines Triggers pro Komponente
- Verwendung von Cache Locking und Cache Partitioning

- Vorteile
  - Kontrolle über Inhalt des Caches
  - Energieeffizienz
  - Vorhersagbarkeit

Vor- und Nachteile

Wichtig

- Ausführung nur eines Triggers pro Komponente
- Verwendung von Cache Locking und Cache Partitioning

- Vorteile
  - Kontrolle über Inhalt des Caches
  - Energieeffizienz
  - Vorhersagbarkeit
- Nachteil
  - Overhead

Multi-threaded Tasks (MTT)

- Annahmen:

Multi-threaded Tasks (MTT)

- Annahmen:
  - Mehrere Jobs pro MTT

Multi-threaded Tasks (MTT)

- Annahmen:
  - Mehrere Jobs pro MTT
  - Existenz einer Deadline für jeden Job

Multi-threaded Tasks (MTT)

- Annahmen:
  - Mehrere Jobs pro MTT
  - Existenz einer Deadline für jeden Job

Thrashing

Abbildung mehrere Speicher Blöcke auf die gleiche Cachezeile
$\rightarrow$ Kontinuierliches Ein- und Auslagern von Blöcken

Multi-threaded Tasks (MTT)

- Annahmen:
  - Mehrere Jobs pro MTT
  - Existenz einer Deadline für jeden Job

Thrashing

Abbildung mehrere Speicher Blöcke auf die gleiche Cachezeile
$\rightarrow$ Kontinuierliches Ein- und Auslagern von Blöcken

Tardy Jobs

Ausführung eines Jobs erst nach seiner Deadline

Algorithmus

Working Set Size (WSS)

Größe der Daten einer Task

Algorithmus

Working Set Size (WSS)

Größe der Daten einer Task

- **Idee:**
  Kontrolle der Summe der WSSs im Cache

Algorithmus

Working Set Size (WSS)

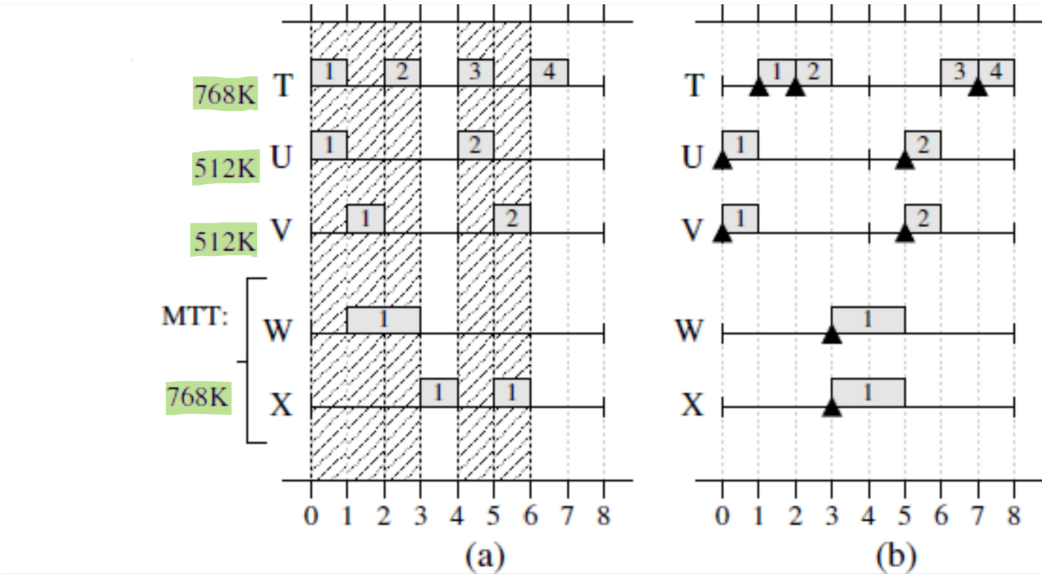
Größe der Daten einer Task

- **Idee:**
  Kontrolle der Summe der WSSs im Cache

```
struct queue *promoted; *edf;
int curCacheSize; maxCacheSize;
// Mark if tardy
if edf->first->deadline < currentTime() then
    edf->first->tardy = true;
end
// Promotion
while edf->next != NULL do
    struct element *smallest = edf->first;
    if edf->next->wss < smallest->wss then
        smallest = edf->next;
    end
    push(smallest, promoted);
end
// Scheduling
if edf->first->tardy then
    if (curCacheSize + edf->first->wss) > maxCacheSize then
        idleCore;
    end
    schedule edf->first;
    curCacheSize += edf->first->wss;
else
    if (curCacheSize + promoted->first->wss) > maxCacheSize
     then
        idleCore;
    else
        schedule promoted->first;
        curCacheSize += promoted->first->wss;
    end
end
```

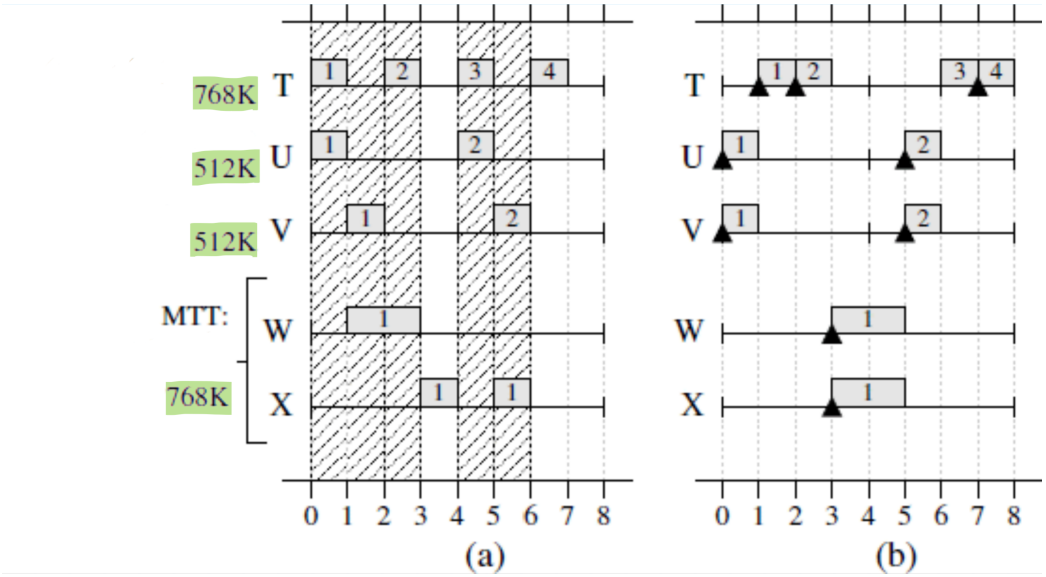
Algorithm 1: Order of scheduling.

Vor- und Nachteile



Scheduling Tasks nach [1]

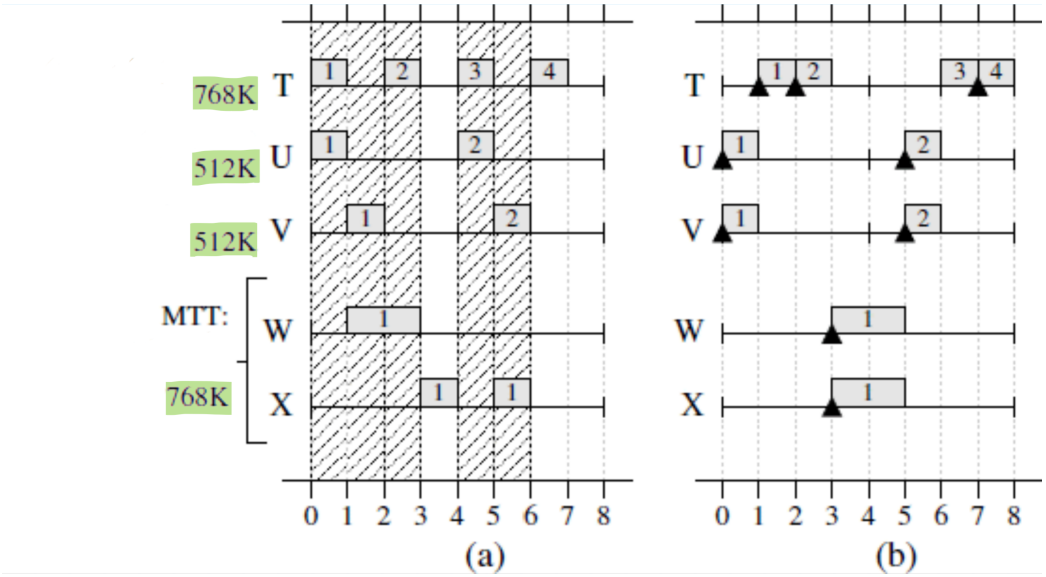
Vor- und Nachteile



Scheduling Tasks nach [1]

- Vorteile
  - Verbesserte Cache Performance

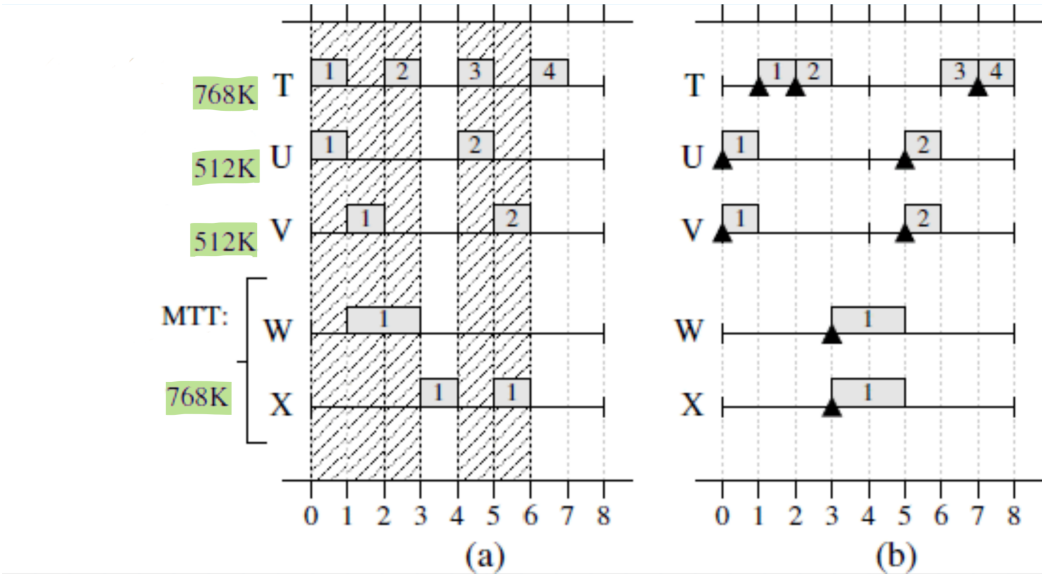
Vor- und Nachteile



Scheduling Tasks nach [1]

- Vorteile
  - Verbesserte Cache Performance
  - Weniger Energie Kosten für Task Ausführung

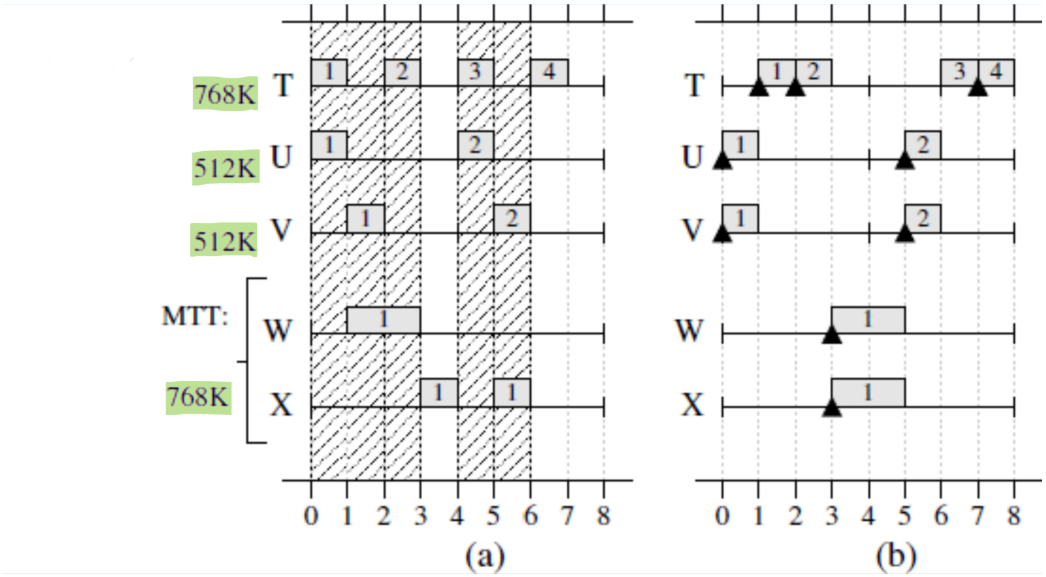
Vor- und Nachteile



Scheduling Tasks nach [1]

- Vorteile
  - Verbesserte Cache Performance
  - Weniger Energie Kosten für Task Ausführung
  - Gesteigerte Vorhersagbarkeit

Vor- und Nachteile



Scheduling Tasks nach [1]

- Vorteile
  - Verbesserte Cache Performance
  - Weniger Energie Kosten für Task Ausführung
  - Gesteigerte Vorhersagbarkeit
- Nachteil
  - Keine Vermeidung von tardy Jobs

Agenda

Cache Profiler

- Zählen von Cache Misses durch Performance Counter

Cache Profiler

- Zählen von Cache Misses durch Performance Counter

Formel

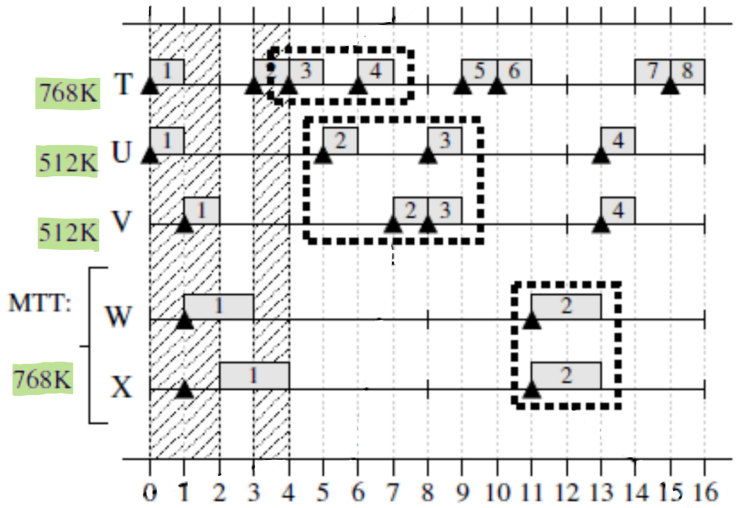
$$WSS = \frac{AnzahlCacheMisses}{AnzahlAnJobsDesMTT} * GrösseDerCachezeile$$

Cache Profiler

- Zählen von Cache Misses durch Performance Counter

Formel

$$WSS = \frac{AnzahlCacheMisses}{AnzahlAnJobsDesMTT} * GrösseDerCachezeile$$



Scheduling Tasks mit Profiler nach [1]

Zusammenfassung

Voraussetzungen an Echtzeitsysteme

Zusammenfassung

Voraussetzungen an Echtzeitsysteme

- Einhalten von Terminen

Zusammenfassung

Voraussetzungen an Echtzeitsysteme

- Einhalten von Terminen
- Vorhersagbarkeit

Zusammenfassung

Voraussetzungen an Echtzeitsysteme

- Einhalten von Terminen
- Vorhersagbarkeit

- Einhalten von Terminen durch **Scheduling**

  $\rightarrow$ Kontrolle von Events

  $\rightarrow$ Weniger Cache Misses

Zusammenfassung

Voraussetzungen an Echtzeitsysteme

- Einhalten von Terminen
- Vorhersagbarkeit

- Einhalten von Terminen durch **Scheduling**

  → Kontrolle von Events

  → Weniger Cache Misses

- Vorhersagbarkeit durch **Management Strategien**

  → Wissen darüber, wo sich Daten befinden

Zusammenfassung

Voraussetzungen an Echtzeitsysteme

- Einhalten von Terminen
- Vorhersagbarkeit

- Einhalten von Terminen durch **Scheduling**

  → Kontrolle von Events

  → Weniger Cache Misses

- Vorhersagbarkeit durch **Management Strategien**

  → Wissen darüber, wo sich Daten befinden

Vielen Dank für die Aufmerksamkeit!

Bibliographie I

John M. Calandrino and James H. Anderson. „On the Design and Implementation of a Cache-Aware Multicore Real-Time Scheduler“. In: 2009, Seiten 194–204.