Echtzeitsysteme

Mehrkern-Echtzeitsysteme

Peter Wägemann

Lehrstuhl für Verteilte Systeme und Betriebssysteme

Friedrich-Alexander-Universität Erlangen-Nürnberg

`https://sys.cs.fau.de/lehre/ss22/ezs/`

12. Juli 2022

Fragestellungen

Wie lassen sich komplexe Echtzeitsysteme handhaben?

- Rechenzeitbedarf ist durch einfache Rechensysteme nicht zu erfüllen
- Beispiel: moderne Fahrerassistenzsysteme
- Die Vielfalt der abzuarbeitenden Aufgaben ist enorm

- Herausforderungen von Mehrkern-Echtzeitsystemen?
  - Welche Anomalien entstehen durch Ausführungsparallelität?
  - Welche Konsequenzen hat dies für die Ablaufplanung?

Wie sehen Verfahren & Mechanismen für Mehrkern-Echtzeitsystem aus?

- Ablaufplanung?
- WCET-Analyse?
- Zugriffssteuerung?

Gliederung

Rückblick Einkernsysteme

Was wir zu wissen glaubten. . .

- Liu-Layland Planbarkeitskriterium für statische Prioritäten[1]
  - RMA ist optimal für periodische Aufgabensysteme (vgl. IV-2/5 ff)
  - Planbare Auslastung $u_{RMA} \leq ln(2) \rightsquigarrow 69,3\%$

[1] Bezogen auf die Priorität der Aufgaben (task priority).

Rückblick Einkernsysteme

Was wir zu wissen glaubten. . .

- Liu-Layland Planbarkeitskriterium für statische Prioritäten[1]
  - RMA ist optimal für periodische Aufgabensysteme (vgl. IV-2/5 ff)
  - Planbare Auslastung $u_{RMA} \leq ln(2) \rightsquigarrow 69,3\%$
- Planungsalgorithmen für dynamische Prioritäten sind optimal[1]
  - Beispielsweise EDF für beliebige Aufgabensysteme (vgl. IV-2/13 ff)

[1] Bezogen auf die Priorität der Aufgaben (task priority).

Rückblick Einkernsysteme

Was wir zu wissen glaubten...

- Liu-Layland Planbarkeitskriterium für statische Prioritäten[1]
  - RMA ist optimal für periodische Aufgabensysteme (vgl. IV-2/5 ff)
  - Planbare Auslastung $u_{RMA} \leq ln(2) \rightsquigarrow 69,3\%$
- Planungsalgorithmen für dynamische Prioritäten sind optimal[1]
  - Beispielsweise EDF für beliebige Aufgabensysteme (vgl. IV-2/13 ff)
- Ablaufplanung behält auch im positiven Fall ihre Zulässigkeit
  - Wenn sich das System besser verhält als angenommen
  - Antwortzeiten vergrößern sich nicht bei abnehmender Ausführungszeit
  - Auslösezeiten und Termine verschieben sich ausführungsbedingt nicht

[1] Bezogen auf die Priorität der Aufgaben (task priority).

Rückblick Einkernsysteme

Was wir zu wissen glaubten...

- Liu-Layland Planbarkeitskriterium für statische Prioritäten[1]
  - RMA ist optimal für periodische Aufgabensysteme (vgl. IV-2/5 ff)
  - Planbare Auslastung $u_{RMA} \leq ln(2) \rightsquigarrow 69,3\%$
- Planungsalgorithmen für dynamische Prioritäten sind optimal[1]
  - Beispielsweise EDF für beliebige Aufgabensysteme (vgl. IV-2/13 ff)
- Ablaufplanung behält auch im positiven Fall ihre Zulässigkeit
  - Wenn sich das System besser verhält als angenommen
  - Antwortzeiten vergrößern sich nicht bei abnehmender Ausführungszeit
  - Auslösezeiten und Termine verschieben sich ausführungsbedingt nicht
- Gleichzeitige Auslösung repräsentiert den kritischen Zeitpunkt
  - Maximale Antwortzeit hängt von der Menge der Aufgaben ab (vgl. IV-2/38)

[1] Bezogen auf die Priorität der Aufgaben (task priority).

Rückblick Einkernsysteme

Was wir zu wissen glaubten...

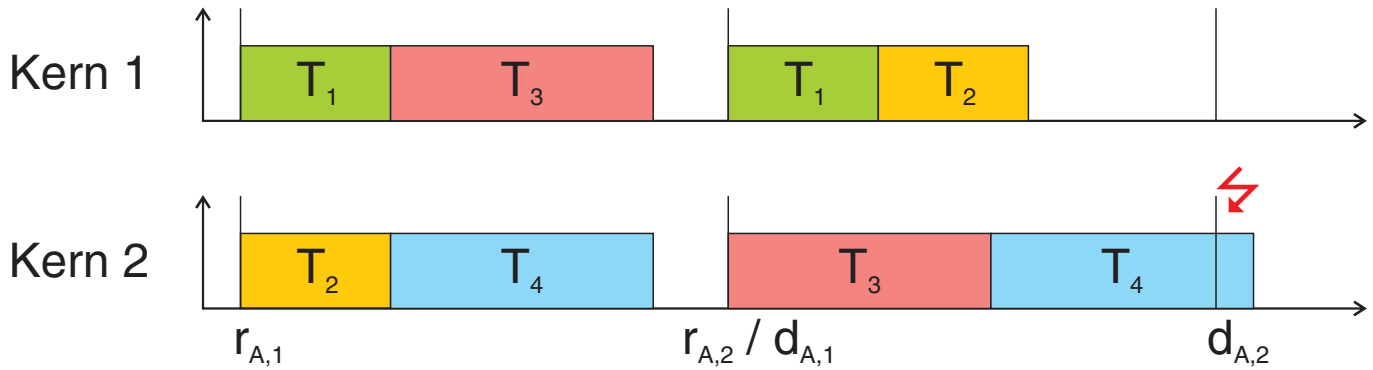
- Liu-Layland Planbarkeitskriterium für statische Prioritäten[1]
  - RMA ist optimal für periodische Aufgabensysteme (vgl. IV-2/5 ff)
  - Planbare Auslastung $u_{RMA} \leq ln(2) \rightsquigarrow 69,3\%$
- Planungsalgorithmen für dynamische Prioritäten sind optimal[1]
  - Beispielsweise EDF für beliebige Aufgabensysteme (vgl. IV-2/13 ff)
- Ablaufplanung behält auch im positiven Fall ihre Zulässigkeit
  - Wenn sich das System besser verhält als angenommen
  - Antwortzeiten vergrößern sich nicht bei abnehmender Ausführungszeit
  - Auslösezeiten und Termine verschieben sich ausführungsbedingt nicht
- Gleichzeitige Auslösung repräsentiert den kritischen Zeitpunkt
  - Maximale Antwortzeit hängt von der Menge der Aufgaben ab (vgl. IV-2/38)

Viele in Einkernsystemen valide Annahmen verlieren in Mehrkernsystemen ihre Gültigkeit!

[1] Bezogen auf die Priorität der Aufgaben (task priority).

Anomalie: Kritischer Zeitpunkt

Antwortzeit in Abhängigkeit von der Ausführungsreihenfolge

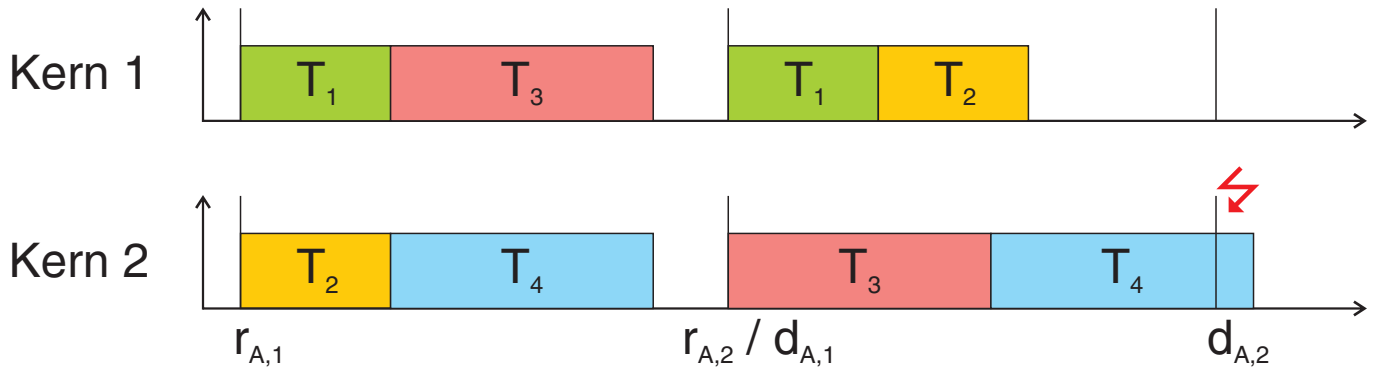


⚠ Gleichzeitige Auslösung repräsentiert nicht mehr zwingend den kritischen Zeitpunkt

- Antwortzeit der zweiten Periode vergrößert
- Terminverletzung durch Wahl des Kerns

Anomalie: Kritischer Zeitpunkt

Antwortzeit in Abhängigkeit von der Ausführungsreihenfolge



Gleichzeitige Auslösung repräsentiert nicht mehr zwingend den kritischen Zeitpunkt

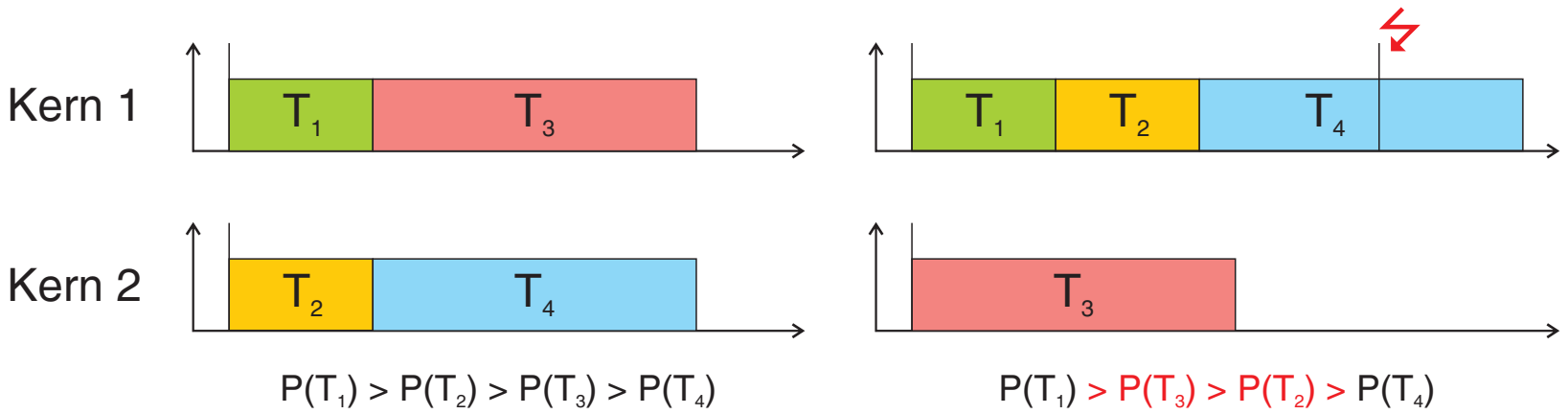
- Antwortzeit der zweiten Periode vergrößert
- Terminverletzung durch Wahl des Kerns

Dhall Effekt [3]

- Garantierte Auslastung wird beliebig schlecht
- Konvergiert im schlimmsten Fall gegen $u = 1$ (Einkernsystem)

Anomalie: Relative Prioritätsordnung

Antwortzeit in Abhängigkeit von der Ausführungsreihenfolge

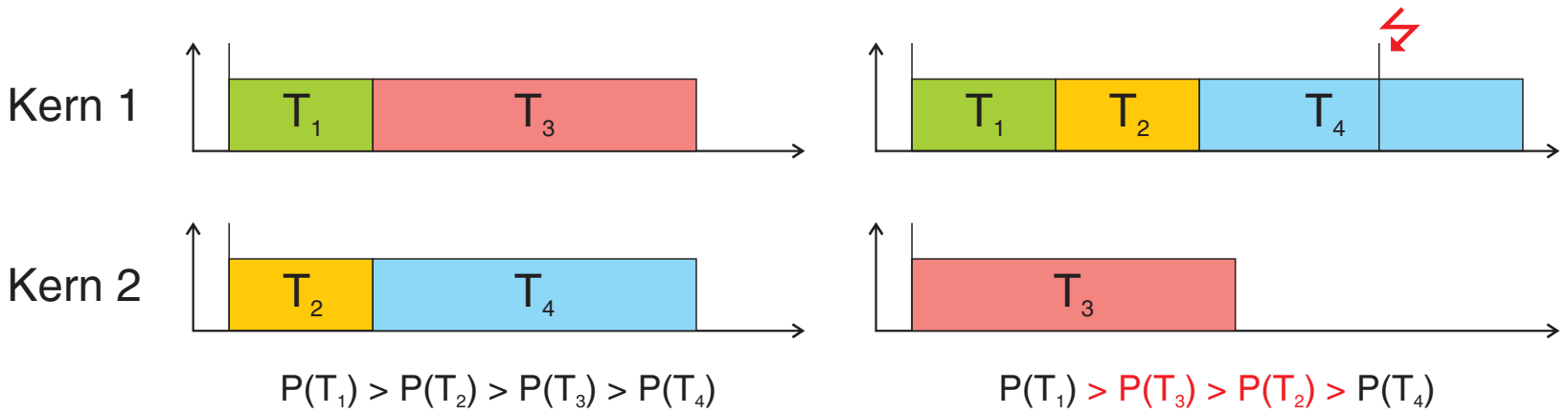


Antwortzeit abhängig vom relativen Prioritätsgefüge

- Höherpriorer Arbeitsaufträge

Anomalie: Relative Prioritätsordnung

Antwortzeit in Abhängigkeit von der Ausführungsreihenfolge



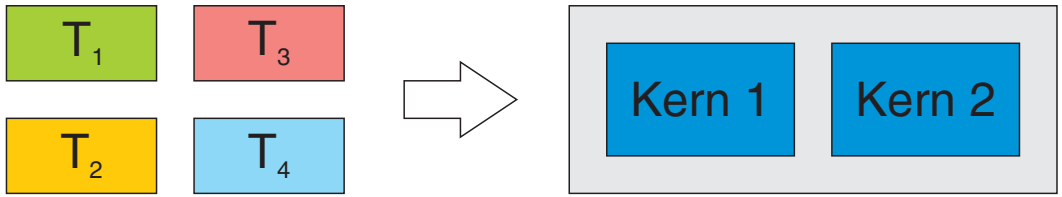
Antwortzeit abhängig vom relativen Prioritätsgefüge

- Höherpriorer Arbeitsaufträge

- Beispiel: T_4 verpasst seinen Termin
  - Vorgezogene Ausführung von T_3 auf Kern 2
  - T_4 wird auf Kern 1 eingeplant und verpasst seinen Termin

→ Erschwert signifikant die Antwortzeitanalyse

Herausforderung Mehrkernechtzeitsystem

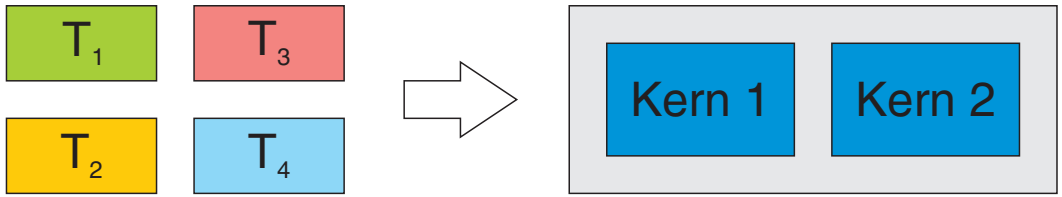


1 Prioritätsproblem (engl. *priority problem*)

→ Wann und in welcher Reihenfolge laufen Arbeitsaufträge?

- Statische Prioritäten für Aufgaben (z.B. RMA)
- Dynamische Prioritäten für Aufgaben (z.B. EDF)
- Dynamische Prioritäten für Aufträge (z.B. PFAIR)

Herausforderung Mehrkernechtzeitsystem



1 Prioritätsproblem (engl. *priority problem*)

→ Wann und in welcher Reihenfolge laufen Arbeitsaufträge?

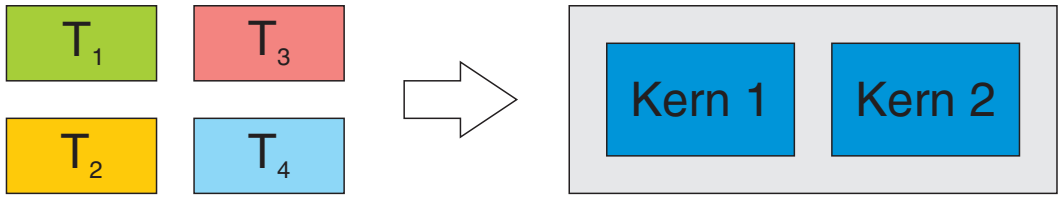
- Statische Prioritäten für Aufgaben (z.B. RMA)
- Dynamische Prioritäten für Aufgaben (z.B. EDF)
- Dynamische Prioritäten für Aufträge (z.B. PFAIR)

2 Allokationsproblem (engl. *allocation problem*)

→ Auf welchem Kern laufen Arbeitsaufträge?

- Keine Migration (engl. *no migration*)
- Migration von Aufgaben (engl. *task-level migration*)
- Migration von Arbeitsaufträgen (engl. *job-level migration*)

Herausforderung Mehrkernechtzeitsystem



1 Prioritätsproblem (engl. *priority problem*)

→ Wann und in welcher Reihenfolge laufen Arbeitsaufträge?

- Statische Prioritäten für Aufgaben (z.B. RMA)
- Dynamische Prioritäten für Aufgaben (z.B. EDF)
- Dynamische Prioritäten für Aufträge (z.B. PFAIR)

2 Allokationsproblem (engl. *allocation problem*)

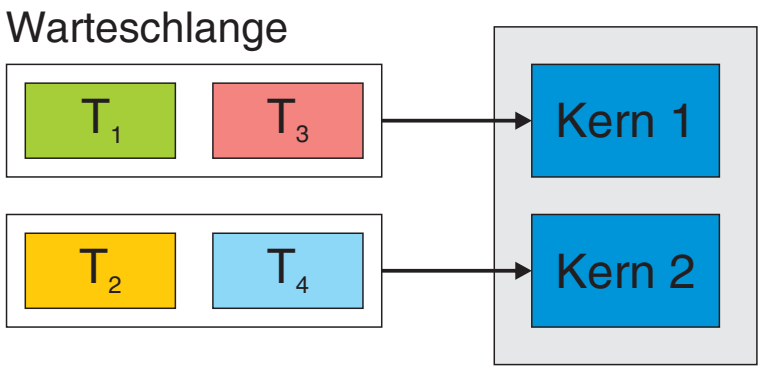
→ Auf welchem Kern laufen Arbeitsaufträge?

- Keine Migration (engl. *no migration*)
- Migration von Aufgaben (engl. *task-level migration*)
- Migration von Arbeitsaufträgen (engl. *job-level migration*)

☞ Partitionierte, globale und hybride Ablaufplanungsverfahren

Gliederung

Partitionierte Ablaufplanung (vgl. [2])



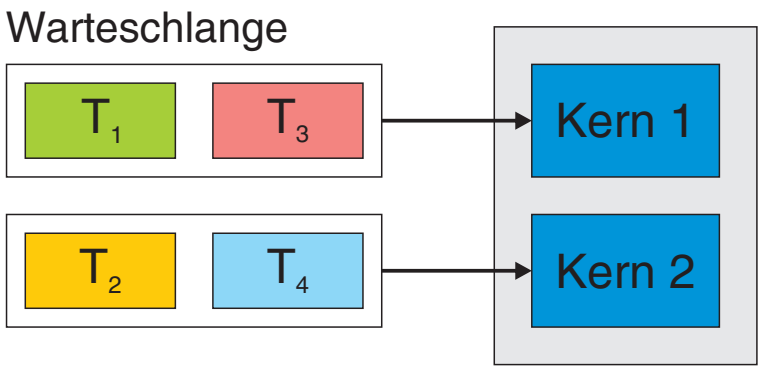
☞ Partitionierte Ablaufplanung (engl. *partitioned scheduling*)

- Verteilung der Aufgaben auf Kerne vor der Laufzeit

  → Alle Aufträge einer Aufgabe werden auf demselben Kern ausgeführt

- Anwendung klassischer EK-Verfahren auf die lokale Aufgabenmenge

Partitionierte Ablaufplanung (vgl. [2])



☞ Partitionierte Ablaufplanung (engl. *partitioned scheduling*)

- Verteilung der Aufgaben auf Kerne vor der Laufzeit

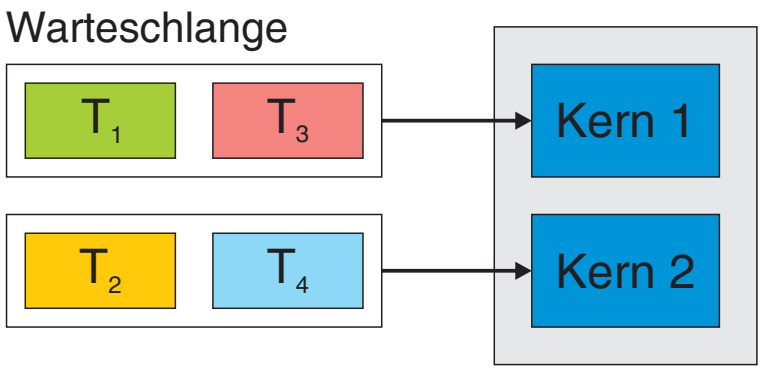
  → Alle Aufträge einer Aufgabe werden auf demselben Kern ausgeführt

- Anwendung klassischer EK-Verfahren auf die lokale Aufgabenmenge

■ Verteilung ↦ Behälterproblem (engl. *bin packing problem*)

- Verteile n Aufgaben der Dichte Δ_i auf m Kerne der Kapazität $\Delta_{max} = 1$
- Zahlreiche Verfahren verfügbar: First-Fit, Next-Fit, Best-Fit, ...

Partitionierte Ablaufplanung (vgl. [2])



☞ Partitionierte Ablaufplanung (engl. *partitioned scheduling*)

- Verteilung der Aufgaben auf Kerne vor der Laufzeit
  → Alle Aufträge einer Aufgabe werden auf demselben Kern ausgeführt
- Anwendung klassischer EK-Verfahren auf die lokale Aufgabenmenge

■ Verteilung ↦ Behälterproblem (engl. *bin packing problem*)

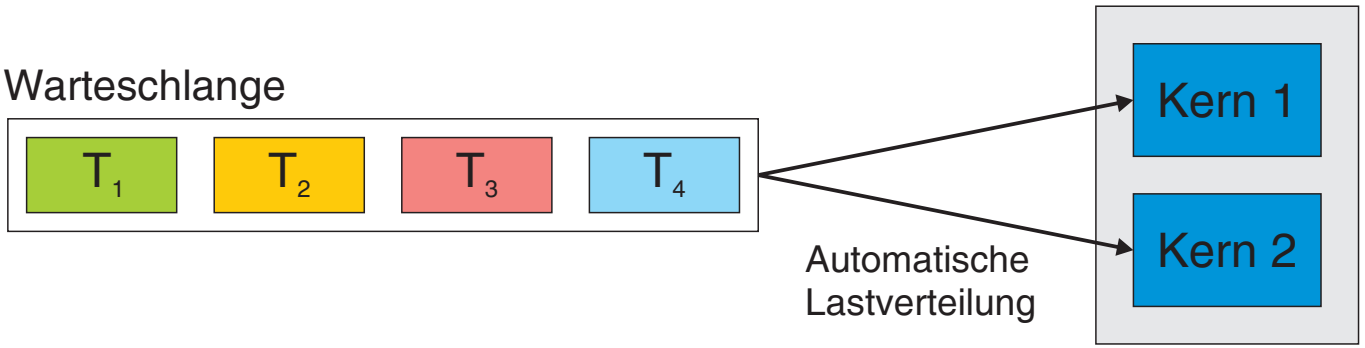
- Verteile n Aufgaben der Dichte Δ_i auf m Kerne der Kapazität $\Delta_{max} = 1$
- Zahlreiche Verfahren verfügbar: First-Fit, Next-Fit, Best-Fit, ...

⚠ Planbare Auslastung im schlimmsten Fall $u_{wc} \approx 0,5$ (50 %)

→ Alle Aufgaben haben eine Auslastung wenig größer als $u_i > 0,5$

Globale Ablaufplanung (vgl. [2])

Migration löst das Auslastungsproblem



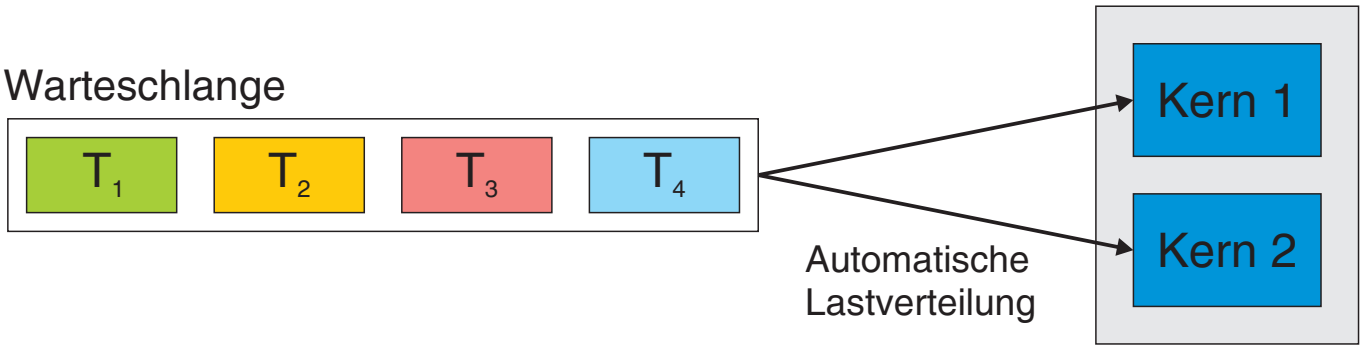
☞ Globale Ablaufplanung (engl. *global scheduling*)

- Verteilung der Arbeitsaufträge auf Kerne zur Laufzeit

→ Aufgaben bzw. Aufträge können zwischen Kernen migrieren

Globale Ablaufplanung (vgl. [2])

Migration löst das Auslastungsproblem



☞ Globale Ablaufplanung (engl. *global scheduling*)

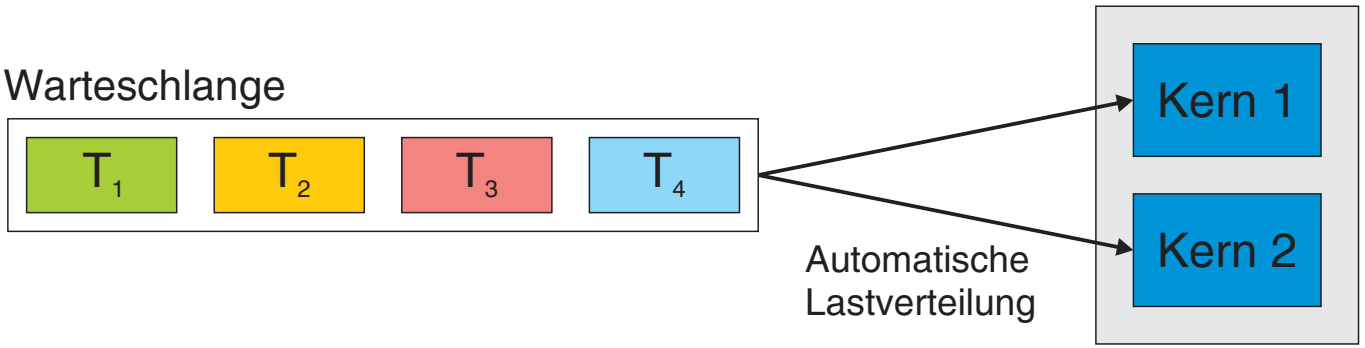
- Verteilung der Arbeitsaufträge auf Kerne zur Laufzeit

→ Aufgaben bzw. Aufträge können zwischen Kernen migrieren

- Mehrkern-Planungsverfahren auf der globalen Warteschlange
  - Statische und dynamische Prioritäten: G-EDF, D-RMA, PFAIR, ...
  - Verfahren mit dynamic-job-level Prioritäten (z.B. PFAIR) dominieren alle anderen Planungsverfahren $\rightsquigarrow$ Auslastung $u_{opt} = 1$

Globale Ablaufplanung (vgl. [2])

Migration löst das Auslastungsproblem



☞ Globale Ablaufplanung (engl. *global scheduling*)

- Verteilung der Arbeitsaufträge auf Kerne zur Laufzeit

→ Aufgaben bzw. Aufträge können zwischen Kernen migrieren

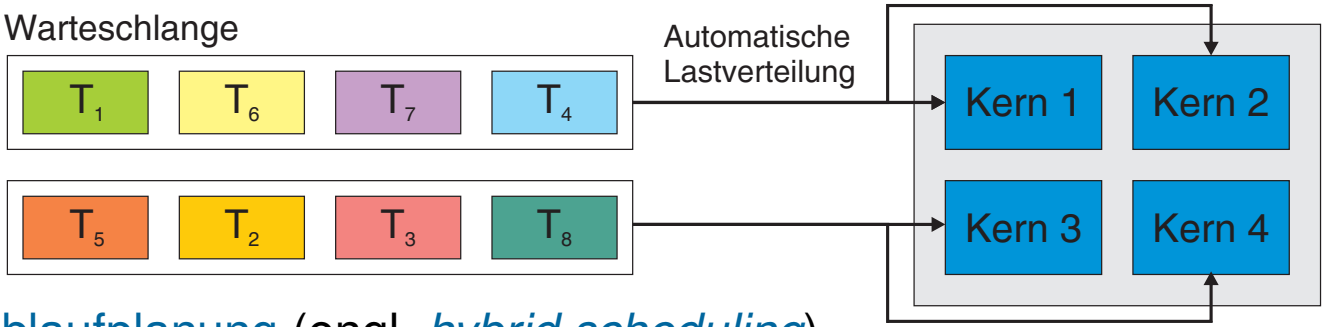
- Mehrkern-Planungsverfahren auf der globalen Warteschlange
  - Statische und dynamische Prioritäten: G-EDF, D-RMA, PFAIR, ...
  - Verfahren mit dynamic-job-level Prioritäten (z.B. PFAIR) dominieren alle anderen Planungsverfahren $\rightsquigarrow$ Auslastung $u_{opt} = 1$

⚠ Globale Verfahren können mit erheblichen Kosten und Unwägbarkeiten behaftet sein

- Hohe Migrationskosten, insbesondere bei Migration von Aufträgen
- Analysierbarkeit der Laufzeit nicht mehr praktikabel

Hybride Ansätze (vgl. [2])

Das Beste aus beiden Welten



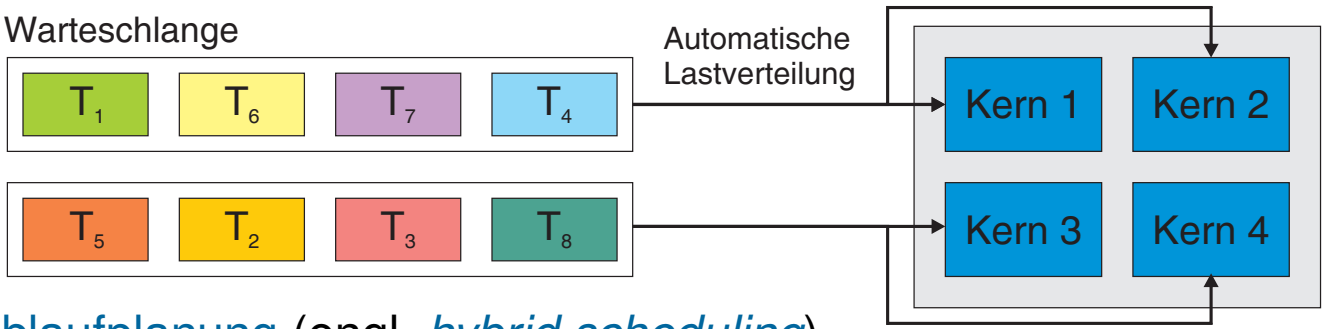
☞ Hybride Ablaufplanung (engl. *hybrid scheduling*)

- Kombination globaler und partitionierter Ablaufplanung

→ Migration minimieren und Auslastung maximieren

Hybride Ansätze (vgl. [2])

Das Beste aus beiden Welten



☞ Hybride Ablaufplanung (engl. *hybrid scheduling*)

- Kombination globaler und partitionierter Ablaufplanung

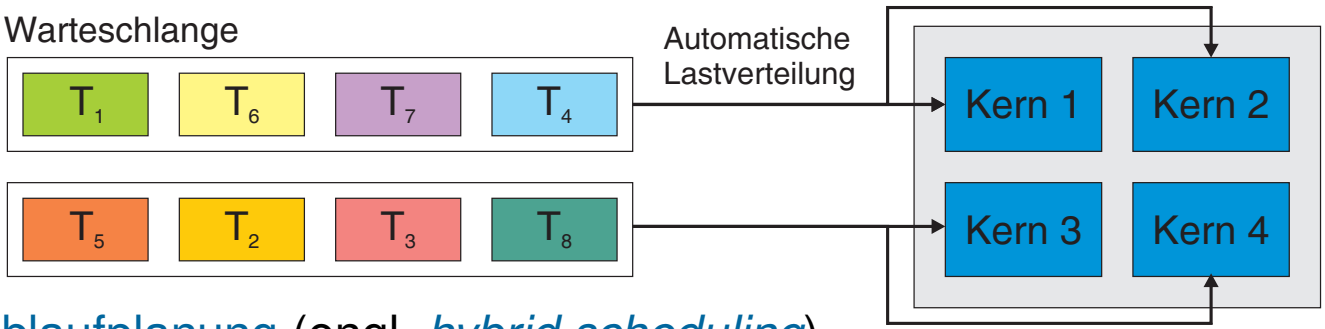
→ Migration minimieren und Auslastung maximieren

- Gruppierende Ablaufplanung (engl. *clustered scheduling*)
  - Zusammenfassen von Kernen zu Gruppen (engl. *cluster*) und Partitionierung
  - Pro Gruppe eine globale Warteschlange

→ Hierarchische Ablaufplanung (vgl. Zusteller V-2/27)

Hybride Ansätze (vgl. [2])

Das Beste aus beiden Welten



☞ Hybride Ablaufplanung (engl. *hybrid scheduling*)

- Kombination globaler und partitionierter Ablaufplanung

→ Migration minimieren und Auslastung maximieren

- Gruppierende Ablaufplanung (engl. *clustered scheduling*)
  - Zusammenfassen von Kernen zu Gruppen (engl. *cluster*) und Partitionierung
  - Pro Gruppe eine globale Warteschlange

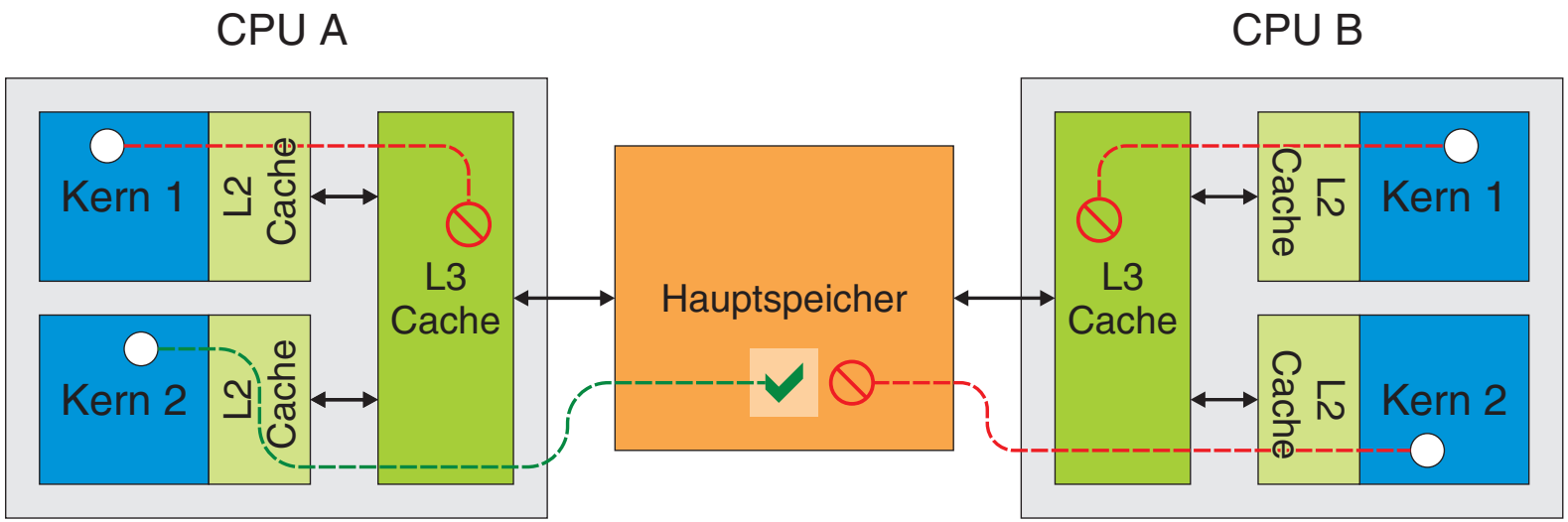
→ Hierarchische Ablaufplanung (vgl. Zusteller V-2/27)

- Teilpartitionierende Ablaufplanung (engl. *semi-partitioned scheduling*)
  - Einschränkung der Migration durch (komplexes) Regelwerk
  - Bsp: Aufgabenmigration nur zwischen Kernen; begrenze Zahl migrierbarer Aufgaben
- Auch Partitionierung nach Wichtigkeit/Kritikalität (vgl. IV-2/16)

Gliederung

Herausforderung: Echte Ausführungsparallelität

Interferenz auf der physikalischeben Ebene



☞ Echte Ausführungsparallelität ist das Problem

- Einkernsysteme ⇝ virtuelle Parallelisierung, Sequentialisierung
- Mehrkehrsysteme ⇝ gemeinsames Betriebsmittel

■ Beispiel: simultaner Speicherzugriff

- impliziter Wettstreit auf physikalischer Ebene

⚠ Auswirkungen auf Laufzeitverhalten

- Blockadezeit durch Speicherzugriff und Invalidierung des Cache-Inhalts
- Studien zeigen dramatische Zunahme der WCET [1]

→ Abhängig von den Eigenschaften der Hardware!

Erweiterung der WCET-Betrachtung

1 Kosten durch Verdrängung (engl. *preemption costs*)

→ Die Problem sind uns bereits bekannt (vgl. III-3), die Komplexität der Analyse bzw. Abstraktion steigt jedoch im Mehrkernfall dramatisch an

Erweiterung der WCET-Betrachtung

1 Kosten durch Verdrängung (engl. *preemption costs*)

→ Die Problem sind uns bereits bekannt (vgl. III-3), die Komplexität der Analyse bzw. Abstraktion steigt jedoch im Mehrkernfall dramatisch an

- Direkte Kosten
  - Unterbrechung, Sicherung und - wiederherstellung des Hardwarekontexts

  ⚠ Einplanung ⇝ Synchronisation im Mehrkernsystem notwendig

Erweiterung der WCET-Betrachtung

1 Kosten durch Verdrängung (engl. *preemption costs*)

→ Die Problem sind uns bereits bekannt (vgl. III-3), die Komplexität der Analyse bzw. Abstraktion steigt jedoch im Mehrkernfall dramatisch an

- Direkte Kosten
  - Unterbrechung, Sicherung und - wiederherstellung des Hardwarekontexts
  - ⚠ Einplanung ⇝ Synchronisation im Mehrkernsystem notwendig
- Indirekte Kosten
  - Wiederaufsetzen/-füllen der Seitentabelle
  - Dislokation von Cacheinhalten (engl. *cache evictions*) zwischen Ausführungsrunden
  - Vergleiche *Cache-Related Preemption Delays*
  - ⚠ Ggf. Herstellen der Cache-Kohärenz (engl. *cache coherency*): einheitliche Sicht auf L3-Cache
  - ⚠ Kommunikation zwischen Kernen/Sockeln

Erweiterung der WCET-Betrachtung (Forts.)

2 Kosten durch Migration (engl. *migration costs*)

⚠ Diese Kosten entstehen nur in Mehrkernsystemen

Erweiterung der WCET-Betrachtung (Forts.)

2 Kosten durch Migration (engl. *migration costs*)

⚠ Diese Kosten entstehen nur in Mehrkernsystemen

- Direkte Kosten
  - Manipulation der Bereitliste (engl. *ready queue*) $\rightsquigarrow$ Synchronisation
  - Sicherung und -wiederherstellung des Hardwarekontexts

Erweiterung der WCET-Betrachtung (Forts.)

2 Kosten durch Migration (engl. *migration costs*)

⚠ Diese Kosten entstehen nur in Mehrkernsystemen

- Direkte Kosten
  - Manipulation der Bereitliste (engl. *ready queue*) $\rightsquigarrow$ Synchronisation
  - Sicherung und -wiederherstellung des Hardwarekontexts
- Indirekte Kosten
  - Laden des aktiven Cachekontexts (engl. *cache working set*) vom Quellkern $\rightsquigarrow$ Dislokation von Cacheinhalten
  - Laden des restlichen Prozesskontexts (transitive Hülle) $\rightsquigarrow$ Cache-Kohärenz, Kommunikation

Erweiterung der WCET-Betrachtung (Forts.)

2 Kosten durch Migration (engl. *migration costs*)

⚠ Diese Kosten entstehen nur in Mehrkernsystemen

- Direkte Kosten
  - Manipulation der Bereitliste (engl. *ready queue*) $\rightsquigarrow$ Synchronisation
  - Sicherung und -wiederherstellung des Hardwarekontexts
- Indirekte Kosten
  - Laden des aktiven Cachekontexts (engl. *cache working set*) vom Quellkern $\rightsquigarrow$ Dislokation von Cacheinhalten
  - Laden des restlichen Prozesskontexts (transitive Hülle) $\rightsquigarrow$ Cache-Kohärenz, Kommunikation

⚠ Zusätzlich Kosten durch Verdrängung bei Migration von (laufenden) Arbeitsaufträgen

Gliederung

Synchronisation in Mehrkernsystemen

Probleme der (unkontrollierten) Prioritätsumkehr (siehe VII/13) verschärft sich in Mehrkernsystemen

- Synchronisation erzeugt kernübergreifende Abhängigkeiten $\rightsquigarrow$ entfernte Blockierung (engl. *remote blocking*)
- Relative Prioritätsordnung (Folie 6) betrifft auch Betriebsmittelnutzung
  - Potentielles Problem: lokal $\neq$ global höchste Priorität

$\rightarrow$ Wovon hängt die Blockierungszeit (s. VII/43) ab?

- Längster kritischer Abschnitt aller Betriebsmittel?
- Maximale Verzögerung durch höherpriore Aufgaben auf einem Kern?

Synchronisation in Mehrkernsystemen

Probleme der (unkontrollierten) Prioritätsumkehr (siehe VII/13) verschärft sich in Mehrkernsystemen

- Synchronisation erzeugt kernübergreifende Abhängigkeiten $\rightsquigarrow$ entfernte Blockierung (engl. *remote blocking*)
- Relative Prioritätsordnung (Folie 6) betrifft auch Betriebsmittelnutzung
  - Potentielles Problem: lokal $\neq$ global höchste Priorität

$\rightarrow$ Wovon hängt die Blockierungszeit (s. VII/43) ab?

- Längster kritischer Abschnitt aller Betriebsmittel?
- Maximale Verzögerung durch höherpriore Aufgaben auf einem Kern?

Gesucht sind Verfahren, welche:

- Kerne nicht unnötig ungenutzt lassen ($\neq$ NPCS VII/22)
- Sich ausschließlich auf die Ausführungszeit der kritischen Abschnitte beziehen ($\neq$ Prioritätsvererbung VII/28)

Prioritätsobergrenzen für Mehrkernsysteme

☞ Prioritätsobergrenzen für Mehrkernsysteme (engl. *Multiprocessor Priority Ceiling Protocol*) (MPCP) [4, S. 352],[5]

- Eine direkte Erweiterung des PCP (s. VII/28) für Einkernsysteme

→ Entfernte Blockierung als Funktion der Ausführungszeit kritischer Abschnitte

Prioritätsobergrenzen für Mehrkernsysteme

☞ Prioritätsobergrenzen für Mehrkernsysteme (engl. *Multiprocessor Priority Ceiling Protocol*) (MPCP) [4, S. 352],[5]

- Eine direkte Erweiterung des PCP (s. VII/28) für Einkernsysteme

→ Entfernte Blockierung als Funktion der Ausführungszeit kritischer Abschnitte

- Vorgehen und Annahmen

1. Einteilung in lokale und globale Betriebsmittel
   - Lokale Betriebsmittel werden nur von lokalen Aufgaben genutzt ⇝ Sychronisation durch klassisches PCP
   - Globale Betriebsmittel werden von Aufgabe auf verschiedenen Kernen genutzt

Prioritätsobergrenzen für Mehrkernsysteme

☞ Prioritätsobergrenzen für Mehrkernsysteme (engl. *Multiprocessor Priority Ceiling Protocol*) (MPCP) [4, S. 352],[5]

- Eine direkte Erweiterung des PCP (s. VII/28) für Einkernsysteme

→ Entfernte Blockierung als Funktion der Ausführungszeit kritischer Abschnitte

- Vorgehen und Annahmen

1. Einteilung in lokale und globale Betriebsmittel
   - Lokale Betriebsmittel werden nur von lokalen Aufgaben genutzt ⇝ Sychronisation durch klassisches PCP
   - Globale Betriebsmittel werden von Aufgabe auf verschiedenen Kernen genutzt
2. Globale Betriebsmittel erhalten eigenen Prioritätenraum
   - Prioritätsobergrenze $\hat{\Pi}_i$ eines globalen Betriebsmittels R_i ist höher als die höchstpriore unabhängige Aufgabe T_H

   → $\hat{\Pi}_i = P_{T_H} + 1 + max(P_{T_i} | T_i \text{ nutzt } R_i)$

Prioritätsobergrenzen für Mehrkernsysteme

☞ Prioritätsobergrenzen für Mehrkernsysteme (engl. *Multiprocessor Priority Ceiling Protocol*) (MPCP) [4, S. 352],[5]

- Eine direkte Erweiterung des PCP (s. VII/28) für Einkernsysteme

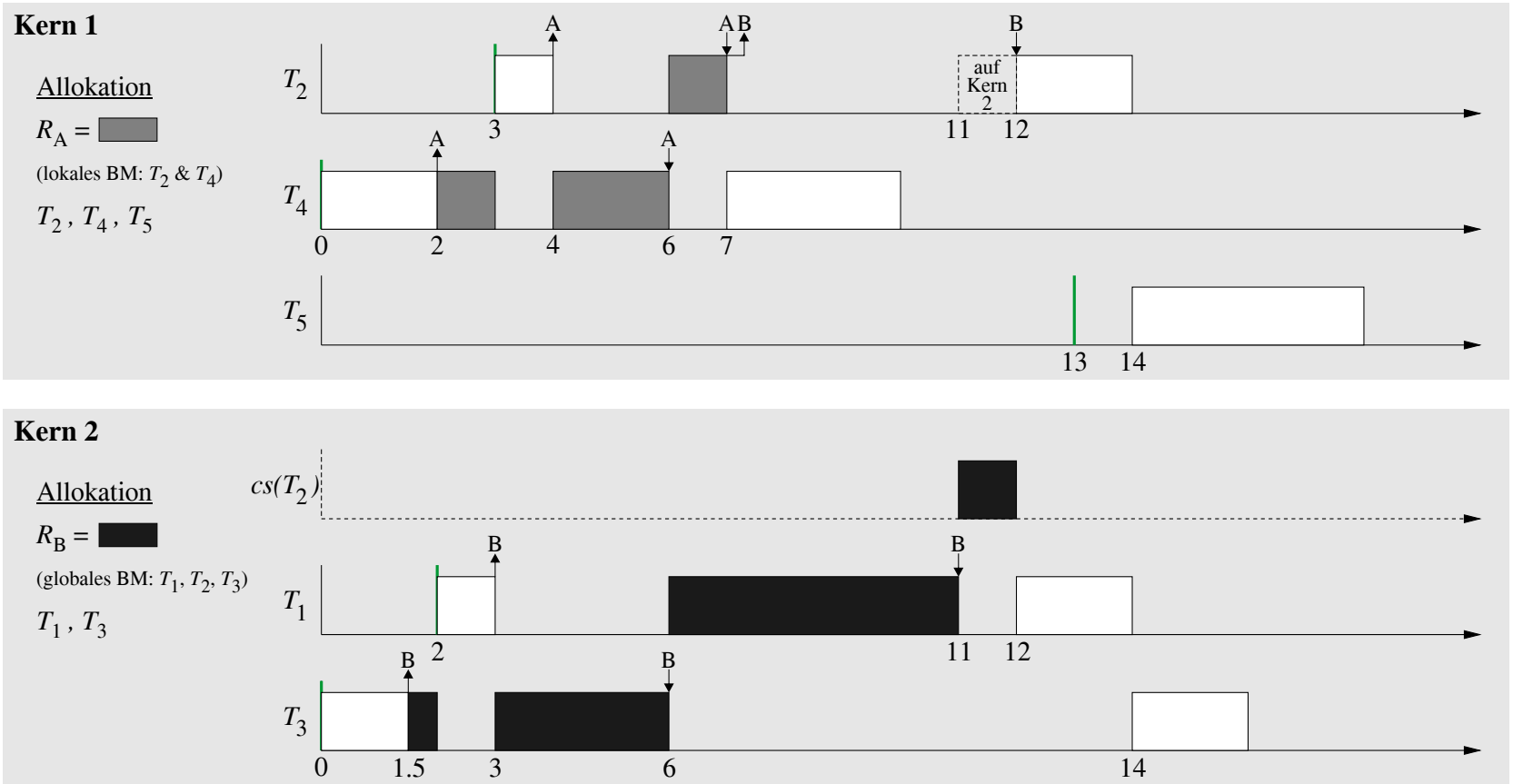
→ Entfernte Blockierung als Funktion der Ausführungszeit kritischer Abschnitte

- Vorgehen und Annahmen

1. Einteilung in lokale und globale Betriebsmittel
   - Lokale Betriebsmittel werden nur von lokalen Aufgaben genutzt $\rightsquigarrow$ Sychronisation durch klassisches PCP
   - Globale Betriebsmittel werden von Aufgabe auf verschiedenen Kernen genutzt
2. Globale Betriebsmittel erhalten eigenen Prioritätenraum
   - Prioritätsobergrenze $\hat{\Pi}_i$ eines globalen Betriebsmittels R_i ist höher als die höchstpriore unabhängige Aufgabe T_H

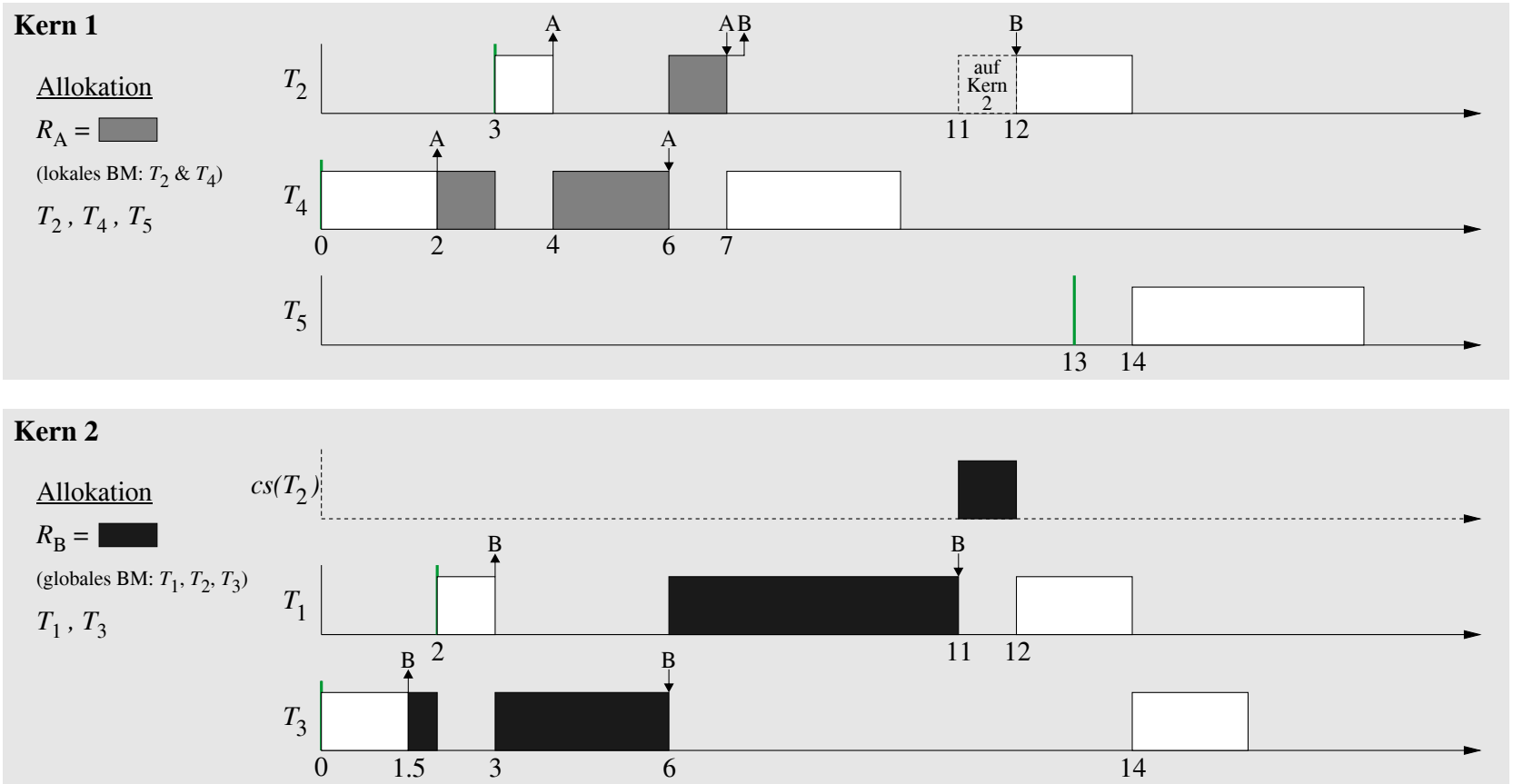
   → $\hat{\Pi}_i = P_{T_H} + 1 + max(P_{T_i} | T_i \text{ nutzt } R_i)$
3. Betriebsmittel werden Kernen zugewiesen und nur dort ausgeführt
   - Ausführung entfernter cs_i auf Ebene der Prioritätsobergrenze $\hat{\Pi}_i$ ($\mapsto$ Vererbung)

Beispiel: Prioritätsobergrenzen für Mehrkernsysteme



bis t_6 Normaler Ablauf mit lokalen Betriebsmitteln auf Kern 1 & 2

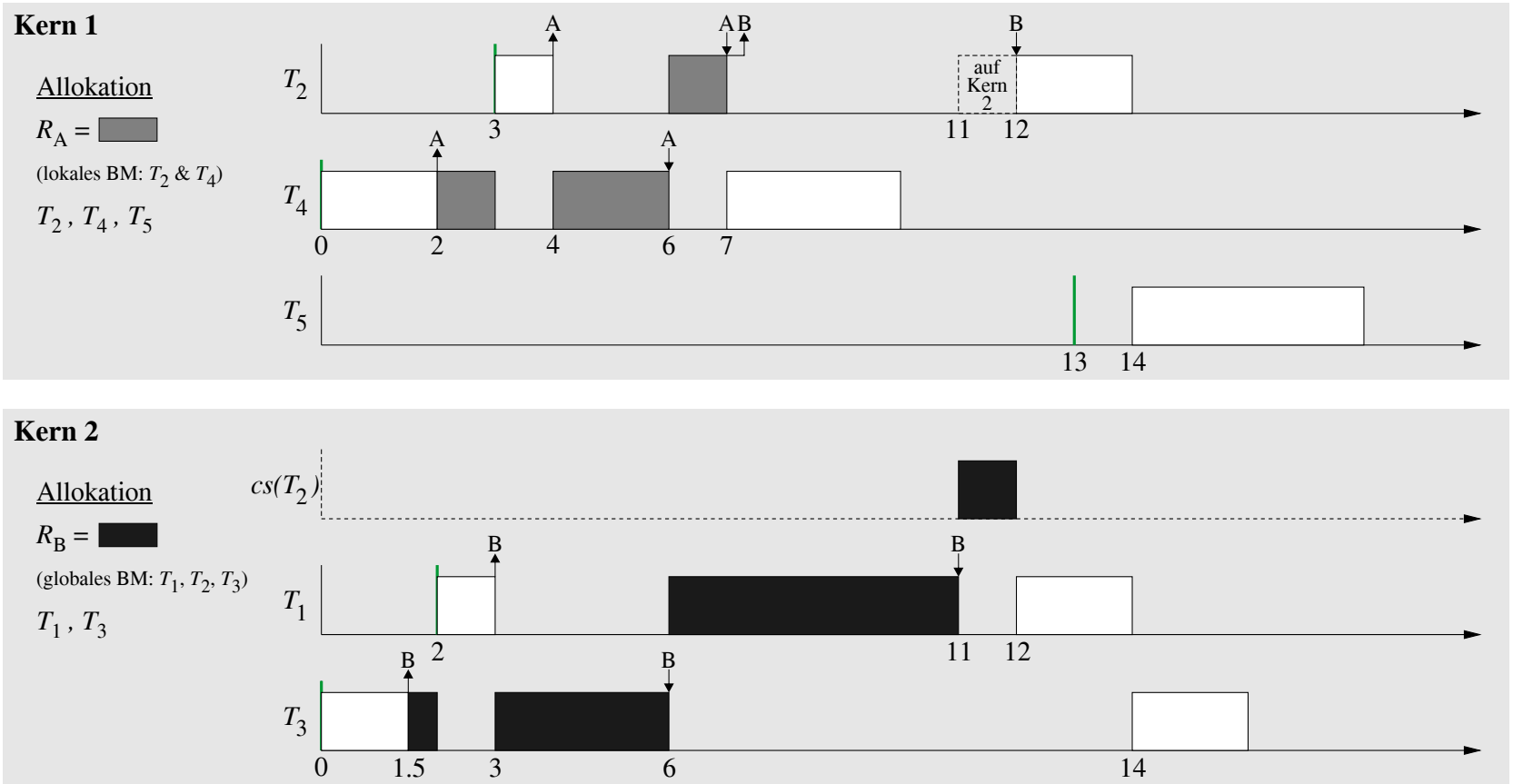
Beispiel: Prioritätsobergrenzen für Mehrkernsysteme



bis t_6 Normaler Ablauf mit lokalen Betriebsmitteln auf Kern 1 & 2

t_7 T_2 fordert R_B (globales BM) an $\rightsquigarrow$ Suspendierung von T_2 auf Kern 1

Beispiel: Prioritätsobergrenzen für Mehrkernsysteme

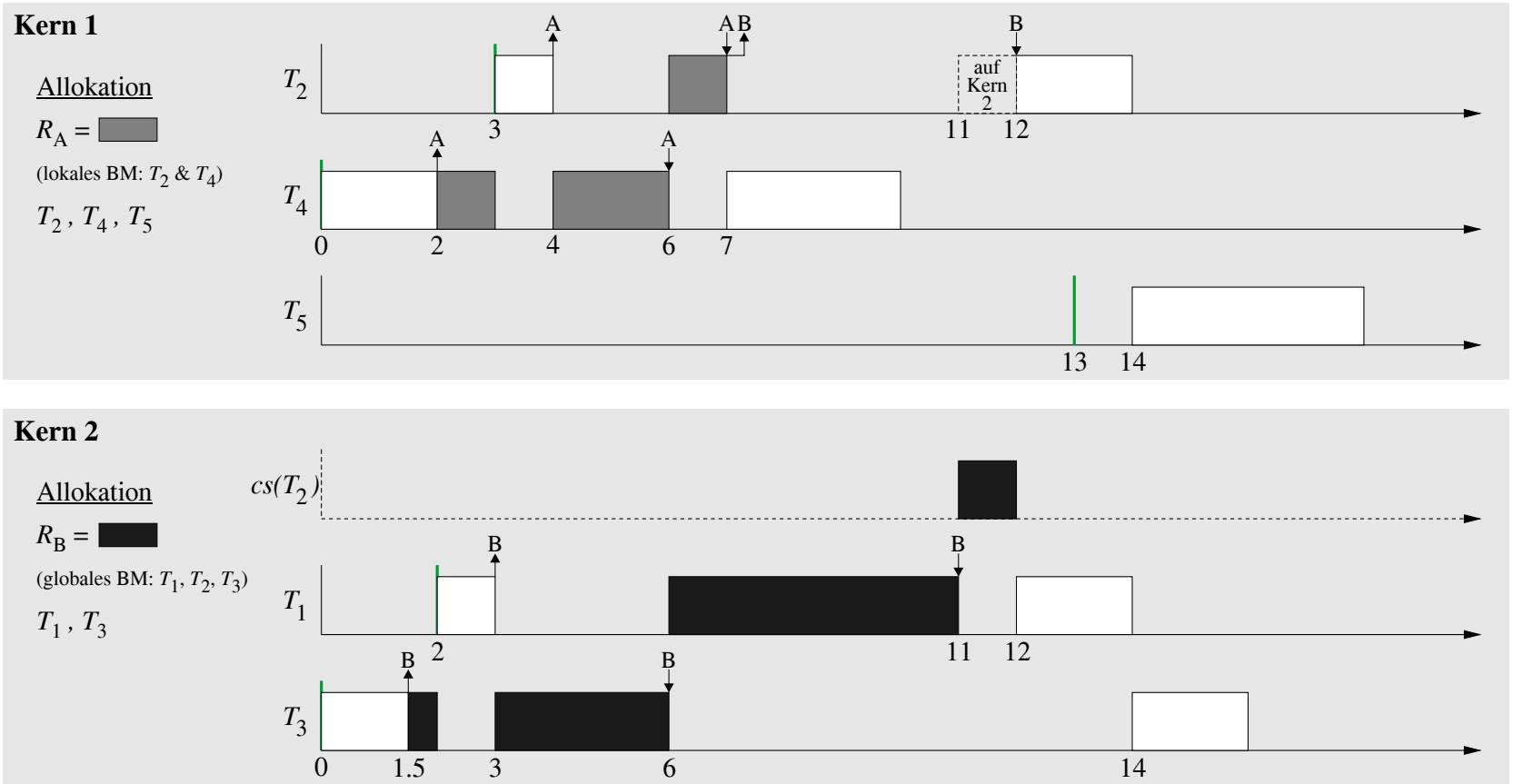


bis t_6 Normaler Ablauf mit lokalen Betriebsmitteln auf Kern 1 & 2

t_7 T_2 fordert R_B (globales BM) an $\rightsquigarrow$ Suspendierung von T_2 auf Kern 1

t_{11} R_B wird von T_1 freigegeben $\rightsquigarrow$ T_2 erbt als entfernter Aufrufer $\hat{\Pi}_B$ auf Kern 2

Beispiel: Prioritätsobergrenzen für Mehrkernsysteme



bis t_6 Normaler Ablauf mit lokalen Betriebsmitteln auf Kern 1 & 2

t_7 T_2 fordert R_B (globales BM) an $\rightsquigarrow$ Suspendierung von T_2 auf Kern 1

t_{11} R_B wird von T_1 freigegeben $\rightsquigarrow$ T_2 erbt als entfernter Aufrufer $\hat{\Pi}_B$ auf Kern 2

t_{11} *cs* von T_2 wird auf Kern 2 mit $P_{\hat{\Pi}_B}(t)$ ausgeführt

Gliederung

Zusammenfassung

- Mehrkernechtzeitsysteme sind die Zukunft
  - Leistungssteigerung durch Parallelisierung
- Ablaufplanung ist eine Herausforderung
  - Wissen aus Einkernsystemen im Allgemeinen nicht übertragbar
  - Zeitliche Anomalien $\rightsquigarrow$ Kritischer Zeitpunkt, Prioritätsordnung
  - Prioritätsproblem und Allokationsproblem
- Partitionierte Ablaufplanung
  - Verteilen der Aufgaben auf Kerne zum Entwurfszeitpunkt
  - Transformation in mehrere Einkernsysteme

  → Bekannte Techniken und Algorithmen sind wieder anwendbar

  → Garantiert planbare Auslastung sehr schlecht
- Globale Ablaufplanung
  - Findet zur Laufzeit statt und erfordert Migration
  - Verfahren mit dynamischen Prioritäten auf Auftragsebene erlauben vollständige Auslastung

  → In der Praxis mit hohen Kosten und Unwägbarkeiten behaftet

Zusammenfassung (Forts.)

- Hybride Ablaufplanung
  - Verbinden die Vor- und Nachteile der anderen Verfahren
  - Teilpartitionierte und gruppierende Ablaufplanung
- WCET-Analyse
  - Komplexität nimmt stark zu
  - Zusätzliche Kosten durch Migration und Synchronisation

Literaturverzeichnis

[1] Baruah, S. ; Bertogna, M. ; Buttazzo, G. :
Multiprocessor Scheduling for Real-Time Systems.
Springer, 2015. –
ISBN 978–3319086958

[2] Davis, R. I. ; Burns, A. :
A Survey of Hard Real-Time Scheduling for Multiprocessor Systems.
In: *ACM Computing Surveys* 43 (2011), Okt., Nr. 4.
http://dx.doi.org/10.1145/1978802.1978814. –
DOI 10.1145/1978802.1978814

[3] Dhall, S. K. ; Liu, C. :
On a real-time scheduling problem.
In: *Operations research* 26 (1978), Nr. 1, S. 127–140

[4] Liu, J. W. S.:
Real-Time Systems.
Englewood Cliffs, NJ, USA : Prentice Hall PTR, 2000. –
ISBN 0–13–099651–3

[5] Rajkumar, R. ; Sha, L. ; Lehoczky, J. P.:
Real-time synchronization protocols for multiprocessors.
In: *Proceedings of the 9th IEEE Real-Time Systems Symposium (RTSS '88)*.
Washington, DC, USA : IEEE Computer Society Press, 1988, S. 259–269