5.2 Prozessorvergabe für Kern-Fäden

- **Scheduler-Struktur**
  - **Struktur der Warteschlangen**
  - **Parallele Bearbeitung von Warteschlangen**
- **Strategien**
  - **Statische oder dynamische Zuordnung von Prozessor und Faden bei Mehrbenutzerbetrieb;**

Ergebnisse der Untersuchungen von Zahorjan und McCann
(J. Zahorjan and C. McCann. Processor scheduling in shared memory multiprocessors. In *Proceedings of the 1990 ACM SIGMETRICS Conference on Measurement and Modeling of Computer systems* , pages 214-225, May 1990.):

- **Unabhängig von der Einzel- und Gesamtlast ist dynamisches Scheduling am besten, wenn der Zusatzaufwand für Kontextumschaltungen gering ist.**
- **Die Vorteile dynamischer Zuordnung werden um so größer, je häufiger sich der 'Parallelitätsgrad' ändert.**
- **Die Vorteile dyn. Zuordnung werden mit zunehmender Systemlast größer.**
- **Bezüglich der mittleren Antwortzeit ist dyn . Zuordnen fast immer besser.**

- **Gemeinsame Bereitschlange, aus der sich freie Prozessoren selbst bedienen:**
  - **Vorteil:**
    Automatische Lastverteilung
  - **Nachteile:**
    Keine Rücksichtnahme auf häufig interagierende Fäden
    Keine Rücksichtnahme auf Reihenfolgeforderungen (z. B. beschrieben durch Präzedenzgraphen) und damit keine Gesamtoptimierung bezüglich der Aufträge (jobs)

- **Interaktionsorientiertes Scheduling (meist 'coscheduling' oder 'gang scheduling' genannt)**

Ousterhout entwickelte drei Methoden:

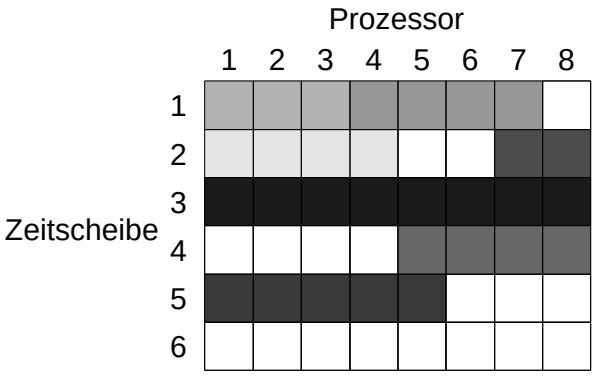
Matrix-Scheduling (matrix)

Fortlaufendes Scheduling (continuous)

Ungeteiltes Scheduling (undivided)

Matrix-Scheduling

- **Alle Fäden eines Auftrags in einer Zeile einordnen**
- **Fäden einer Zeile werden gleichzeitig zugeordnet**
- **Zeilen nach Round Robin zuordnen**



- **Löcher führen zu Effizienzverlusten**

➡ Jeder gemäß ausgewählter Zeile freie Prozessor sucht Faden in seiner Spalte

Fortlaufend

- **Matrix linearisiert durch Aneinanderreihung der Zeilen**
  ➡ je p aufeinanderfolgende Zellen sind verschiedenen Prozessoren zugeordnet
- **Zu jedem Zeitpunkt Fäden eines Fensters der Länge p zugeordnet**
- **Neuankömmling wird in aktives Fenster aufgenommen, wenn dies möglich ist**
- **Andernfalls wird Fenster solange nach rechts geschoben, bis zum erstenmal das linke Feld leer ist. Dies wird wiederholt, bis das Fenster genügend freie (nicht notwendigerweise aufeinanderfolgende) Zellen enthält. Dadurch Verringerung des Verschnitts!**
- **Nach jedem Zeitquant wird Zuordnungsfenster weitergeschoben, bis der linke Eintrag zu einem Auftrag gehört, der im vorherigen Zeitschritt nicht vollständig zugeordnet war.**
- **Nachteil: Unzusammenhängend gespeicherte Aufträge können beim Scheduling benachteiligt sein!**

- **Ungeteilt**
  - **Analog fortlaufend, aber nur zusammenhängende Einordnung**
  - **Der bei fortlaufender Einordnung erwähnte Nachteil verschwindet, dafür größerer Verschnitt**
- **Es handelt sich um zentralisierte Algorithmen**

- **Anderer Ansatz: baumartig angeordnete Controller verwalten Prozessoren ähnlich einem Buddy-Verfahren bei Speicherplatzvergabe.**
- **Präzedenzgraphorientierte Strategien**
  - **Die meisten Untersuchungen betreffen einstufige Präzedenzgraphen, d. h. ein Hauptfaden erzeugt eine Reihe von Unterfäden und beendet sich dann. Meist wird noch angenommen, daß die Unterfäden nicht interagieren.**
  - **Bekannte Strategie für diesen Fall ist RR**

    Erste Version führt Schlange von bereiten Fäden und bearbeitet sie nach RR, d. h. jeder freigewordene Prozessor bearbeitet den nächsten Auftrag für Q Zeiteinheiten und reiht ihn dann am Ende der Bereitschlange wieder ein.

    Zweite Version führt in der WS Aufträge. Falls ein Auftrag mehr Fäden besitzt als es Prozessoren gibt, wird innerhalb des Auftrags wiederum nach RR zugeteilt.
  - **Probleme:**

    Verdrängung während eines Spinlock oder in kritischem Abschnitt

    Häufige Kontextumschaltungen

- **Statische oder dynamische Partitionierung**
  - **Ziel ist die Minimierung der Kontextumschaltungen**
  - **Hypothese: Aufträge erreichen die beste Effizienz, wenn die Zahl der Fäden gleich der Zahl der verwendeten Prozessoren ist.**
  - **In NUMA-Architekturen existiert häufig eine von der Anwendung vorteilhaft nutzbare Kommunikationsstruktur; wegen ihrer guten Skalierbarkeit spielen Gitterarchitekturen eine besondere Rolle.**
  - **Als Strategien bieten sich Weiterentwicklungen der Matrixmethode von Ousterhout an.**

- **Beispiel einer solchen Zuteilungsstrategie für ein torusartiges System, dessen Knoten Multiprozessoren sind (MEMSY):**

Definition

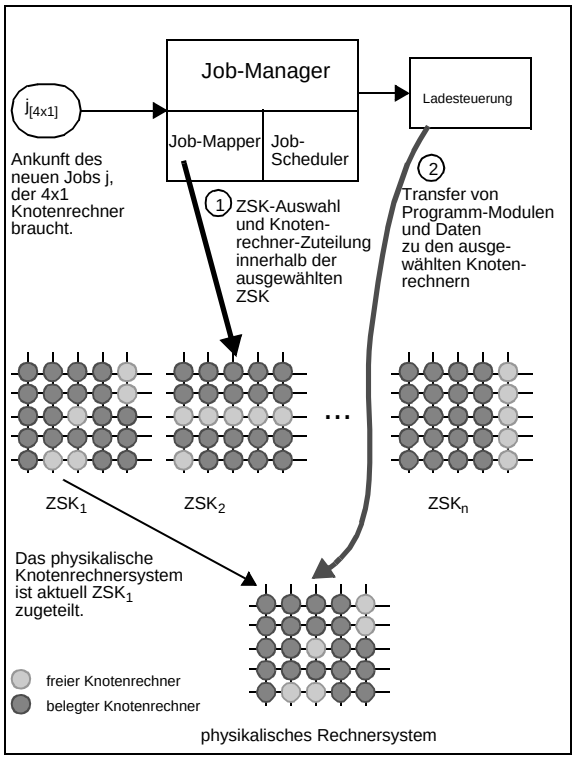
Eine *Zeitscheibenklasse (ZSK)* eines Parallelrechnersystems PS ist ein virtuelles Knotenrechnersystem, das folgende Eigenschaften hat:

(a) Die Anzahl der virtuellen Knotenrechner in ZSK ist gleich der Anzahl der physikalischen Knotenrechner in PS.

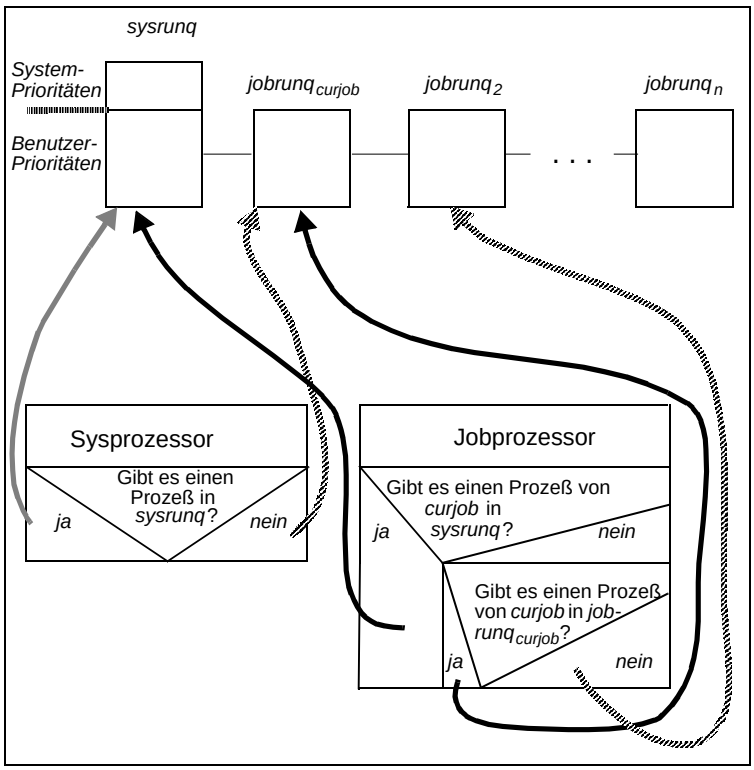
(b) ZSK hat die gleiche Verbindungstopologie wie PS.

(c) Es gibt eine bijektive Abbildung vps_{ZSK} der Knotenrechner von ZSK auf die Knotenrechner von PS. Gilt (für ein $V \in ZSK$ und ein $R \in PS$), so $R = vps_{ZSK}(V)$ sind V und R *topologisch äquivalent*.
Dabei wird unter dem Begriff der *topologischen Äquivalenz* hier folgendes verstanden: Wenn den Systemen PS und ZSK jeweils das gleiche Koordinatensystem zugrundegelegt würde, das die Knotenrechner bijektiv auf Koordinaten abbildet, so wären ein Knotenrechner $K_V \in ZSK$ und ein Knotenrechner $K_R \in PS$ topologisch äquivalent, wenn ihnen die gleichen Koordinaten zugeordnet wären.

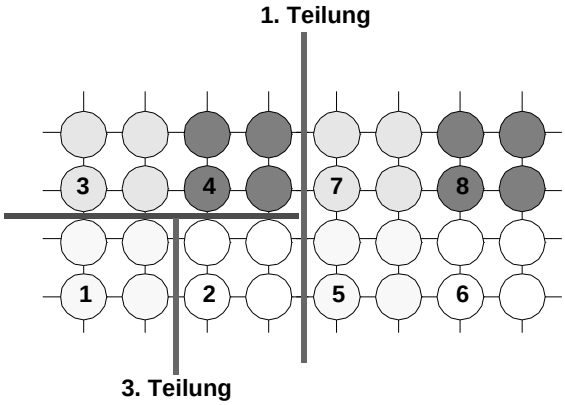
Einordnung neuer Aufträge



Knotenscheduling



Anpassung der eindimensionalen Buddy-Strategie



Ein Torus(8x4)-System, das aus Torus(2x2)-System-Einheiten basierend auf einer zweidimensionalen Buddy-Strategie aufgebaut wird .

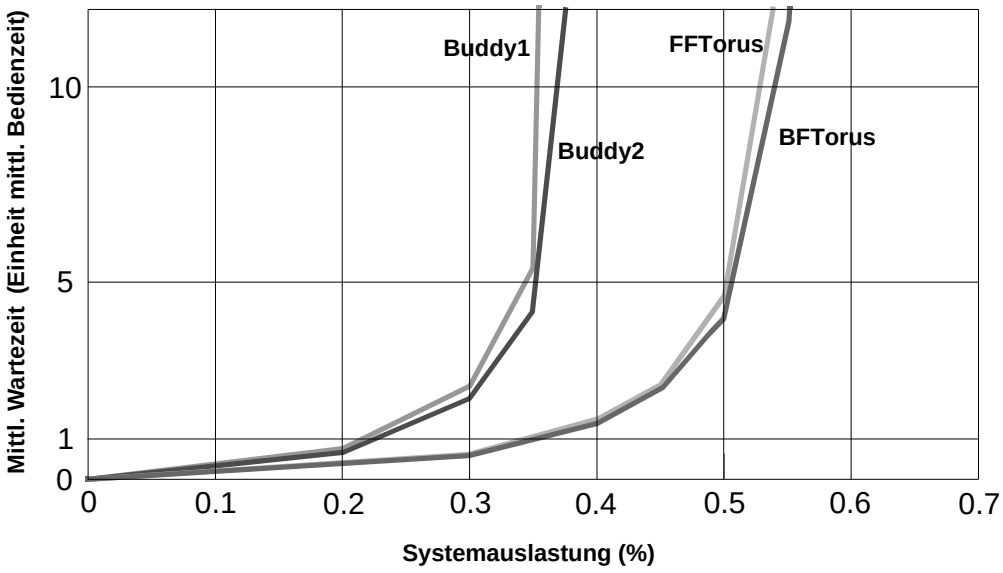
Die Zahlen in den Knoten notieren die Reihenfolge, in der die entsprechenden (2x2)-Systeme hinzugefügt werden.

Vergleich einiger ausgewählter Strategien mittels Simulation

- **FFTorus: First-Fit angepaßt an Oberfläche eines Torus ohne Vorzugsrichtungen bei der Plazierung von Rechtecken**
- **BFTorus: Best-Fit angepaßt an Oberfläche eines Torus ohne Vorzugsrichtungen bei der Plazierung von Rechtecken**
- **Buddy1: Teilung gemäß Buddy-Strategie, vollständige Belegung eines 'Buddy'**
- **Buddy2: Teilung gemäß Buddy-Strategie, aber Belegung nur der wirklich benötigten Knoten**

Simulationsergebnisse für die vorangehenden Plazierungsstrategien

- **exponentielle Verteilung der Zwischenankunftszeiten und Bedienzeiten**
- **Höhen und Längen der angeforderten Teilbereiche unabhängig gleichverteilt**

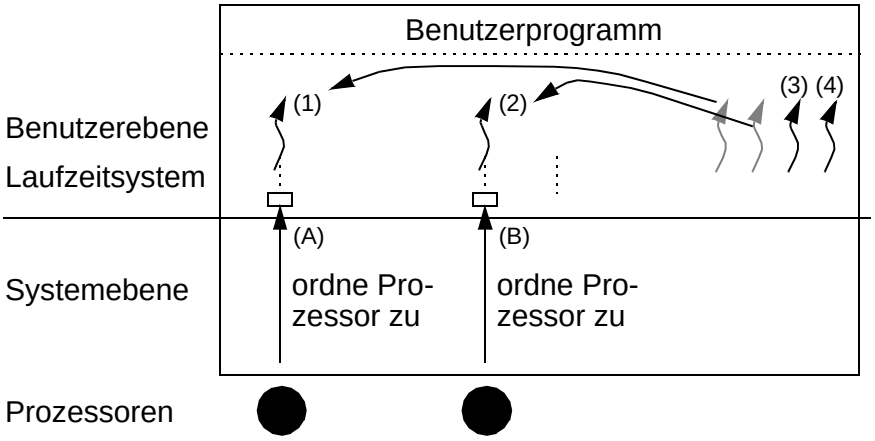


- **Hand-off Scheduling**
  - **Benutzer gibt Hinweise**
    - **Hinweise auf Zweckmäßigkeit des Zuordnens**
    - **Hinweise auf zuzuordnende Prozesse**

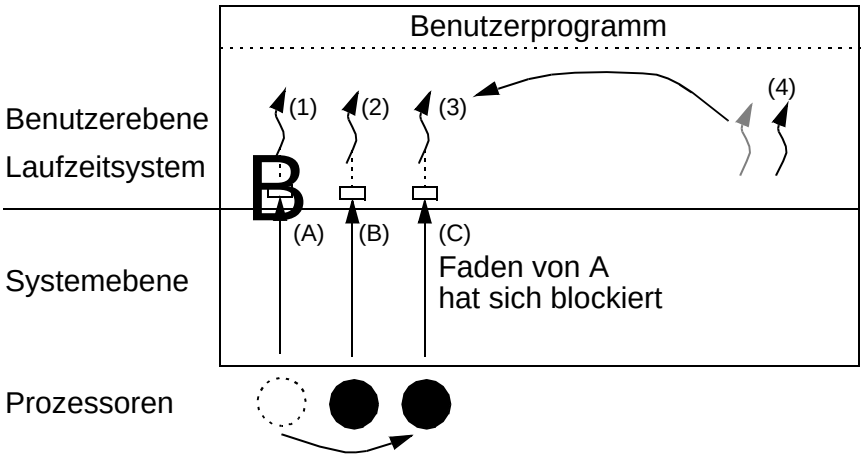
  Letzteres ist insbesondere im Zusammenhang mit Kooperation von Interesse.

❑ **Mischung aus Kern- und Benutzerfäden**

- **Kern ordnet der Anwendung beispielsweise zwei Prozessoren zu**

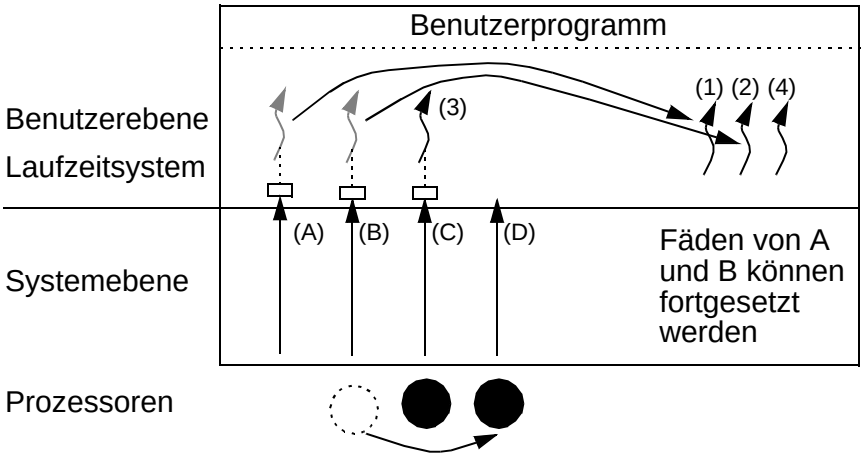


- **Ein Benutzerfaden blockiert sich im Kern wegen E/A**



Prozessorvergabe - Mulitprozessoren: Kern- und Benutzerfäden

- **E/A abgeschlossen**



Prozessorvergabe - Mulitprozessoren: Kern- und Benutzerfäden

- **'upcall' wegen freien Prozessors**