# Proseminar 2003 - AlphaSort

Benedikt Meurer

Fachbereich Elektrotechnik und Informatik
Universität Siegen

# Übersicht

# Thema

- Vorstellung des Sortiverfahrens AlphaSort
- Messverfahren in der Praxis (Benchmarks)
- Einführung in die Speicherhierarchie
- Überblick über weitere kommerzielle Sortierverfahren

## Was ist AlphaSort?

- 1995 veröffentlichtes, kommerzielles, externes Sortierverfahren für handelsübliche Hardware
- Sortiert 100 Megabyte Daten in 7 Sekunden

# AlphaSort

## Was ist AlphaSort?

- 1995 veröffentlichtes, kommerzielles, externes Sortierverfahren für handelsübliche Hardware
- Sortiert 100 Megabyte Daten in 7 Sekunden

## Frage: Ein weiteres Sortierverfahren?

- Das Sortierproblem gilt doch als theoretisch gelöst?
- Warum nicht einfach QuickSort?

# AlphaSort

## Was ist AlphaSort?

- 1995 veröffentlichtes, kommerzielles, externes Sortierverfahren für handelsübliche Hardware
- Sortiert 100 Megabyte Daten in 7 Sekunden

## Frage: Ein weiteres Sortierverfahren?

- Das Sortierproblem gilt doch als theoretisch gelöst?
- Warum nicht einfach QuickSort?

## Antwort

- Aus EI1 bekannte Sortierverfahren nicht extern
- Praxis hat andere Maßstäbe als Komplexität (Zeit und Kosten)

## Sortieren bis 1992

- Durchschnittliches System brauchte 1985 ca. 15 Minuten, um 100 Megabyte Daten zu sortieren
- Beste Ergebnisse verbuchten Grossrechner (z.B. inoffizieller Rekord von 1986 mit 26 Sekunden auf einem Cray Supercomputer)

# Übersicht

1. Einleitung
2. Benchmarks
3. Die Speicherhierarchie
4. Das AlphaSort Verfahren
5. Weitere Verfahren

## Was ist eine Benchmark?

- Messen abhängiger, absoluter Werte (z.B. Laufzeit und Kosten) anstatt Messen im Sinne von Komplexität.
- Das bedeutet wiederum, dass bei der Veröffentlichung von derartigen Messwerten auch immer die genauen Umstände des Messens angegeben werden müssen.

## Beispiele für Benchmarks

- 3D Benchmarks messen Bildwiederholrate, d.h. Leistung von Graphikchip, Bus und OpenGL Implementierung.

## Die Datamation Benchmark

Der Datamation Artikel legt drei grundlegende Benchmarks fest:

- DebitCredit misst Datenverarbeitungsgeschwindigkeit eines Systems durch einfache Lese-/Schreibeoperationen.
- Scan misst I/O-Geschwindigkeit des Dateisystems durch Lesen und Schreiben von 100 KB Daten von Festplatte.
- Sort misst Geschwindigkeit von CPU, I/O Subsystems und Betriebssystems, durch Sortieren vom 1 Million 100-byte Datensätzen.

## Die Sort Benchmark–I

### Regeln für Sort Benchmark

- Eingabedatei auf Festplatte mit 1 Million 100-byte Datensätzen in zufälliger Reihenfolge
- Datensatz mit 10-byte Schlüssel und nicht komprimierbar
- Ausgabedatei auf Festplatte mit Datensätzen in aufsteigend sortierter Reihenfolge

### Implementation mit üblichen "Tricks"

- Low-Level Funktionalität
- undokumentierte Funktionen
- soviel CPUs, Platten, Speicher wie man will/braucht
- das Dateisystem darf nicht umgangen werden!

## Die Sort Benchmark–II

### Zeitmessung

- Start des Sortierprogramms
- Öffnen der Eingabedatei und Anlegen der Ausgabedatei
- Lesen der Eingabedatei
- Sortieren der Datensätze
- Schreiben der Ausgabedatei
- Schliessen der Dateien
- Beenden des Sortierprogramms

### Kostenberechnung

- $\texttt{Kosten} = \frac{\frac{\texttt{5 Jahres Kosten}}{\texttt{5 Jahre}}}{\texttt{Sortierzeit}}$
- Beispiel: 1 Minute sortieren bei 1 Million Dollar 5-Jahres Kosten ergibt $0,38$ Cent pro Sortierdurchgang

## Probleme der Datamation Benchmark

- Geht von Sortierverfahren aus, die 10 Minuten bis 1 Stunde dauern
- Faktor Startup- und Shutdown-Zeit gewinnt zu stark an Bedeutung (von den 7 Sekunden bei AlphaSort sind das schon 25%), Ziel ist es aber die I/O-Performance des Systems zu messen (unter der Last des Sortierens).

## Zwei neue Benchmarks

- MinuteSort Welches Datenvolumen kann ich in einer Minute sortieren?
- PennySort Welches Datenvolumen kann ich für einen Penny sortieren?

## Regeln für MinuteSort

- Sortiere soviel du kannst in einer Minute.
- Die Eingabe liegt extern vor.
- Datensätze sind 100 Byte groß.
- Die ersten 10 Byte bilden den Schlüssel.
- Die Ausgabedatei ist eine sortierte Folge der Eingabedaten.
- Die Ein- und Ausgabedateien müssen mit Hilfe konventioneller Werkzeugen lesbar sein.

## Regeln für PennySort

- Sortiere soviel du kannst für weniger als einen Penny.
- Ansonsten gelten diesselben Regeln wie für MinuteSort.

## Problem

- Wie soll mit ”Nur”-Gewinnern verfahren werden?

## Kategorien

### Problem

- Wie soll mit "Nur"-Gewinnern verfahren werden?

### Lösung

- Aufteilen in zwei Kategorien
- Indy ist offen für jegliche Sortier Hard- und Software
- Daytona ist nur für kommerziell verfügbare Sortiersoftware

## Kategorien

### Problem

- Wie soll mit "Nur"-Gewinnern verfahren werden?

### Lösung

- Aufteilen in zwei Kategorien
- Indy ist offen für jegliche Sortier Hard- und Software
- Daytona ist nur für kommerziell verfügbare Sortiersoftware

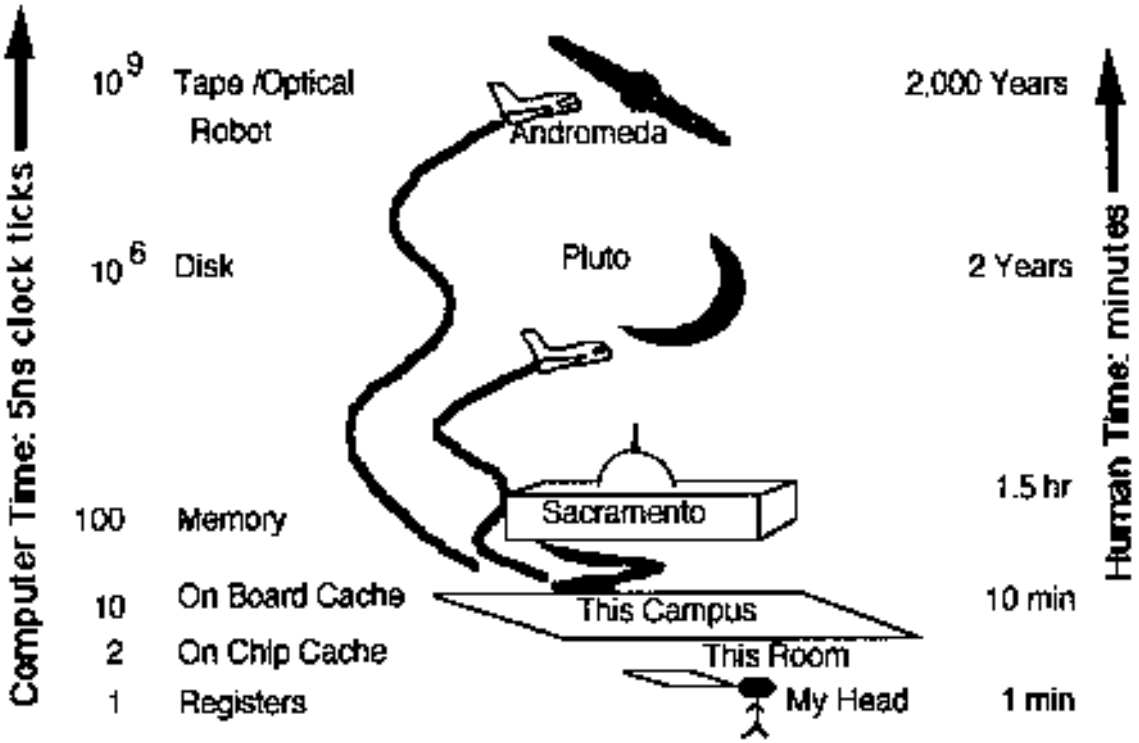
### Benchmarks

- PennySort/Indy
- PennySort/Daytona
- MinuteSort/Indy
- MinuteSort/Daytona

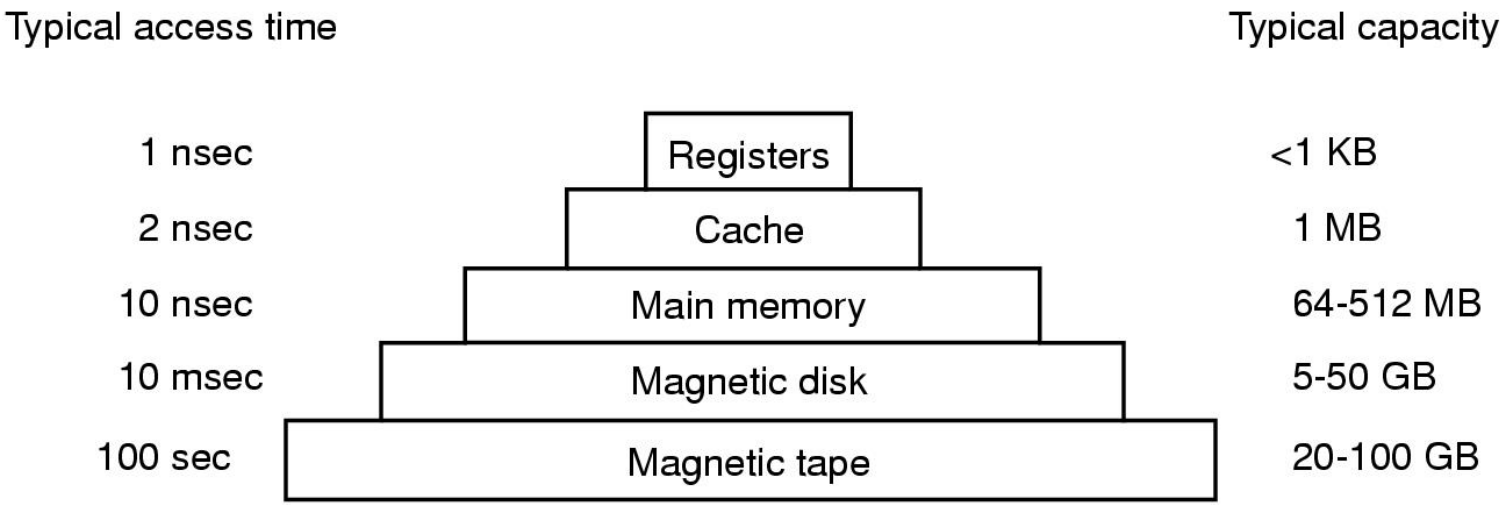
# Übersicht

1. Einleitung
2. Benchmarks
3. Die Speicherhierarchie
4. Das AlphaSort Verfahren
5. Weitere Verfahren

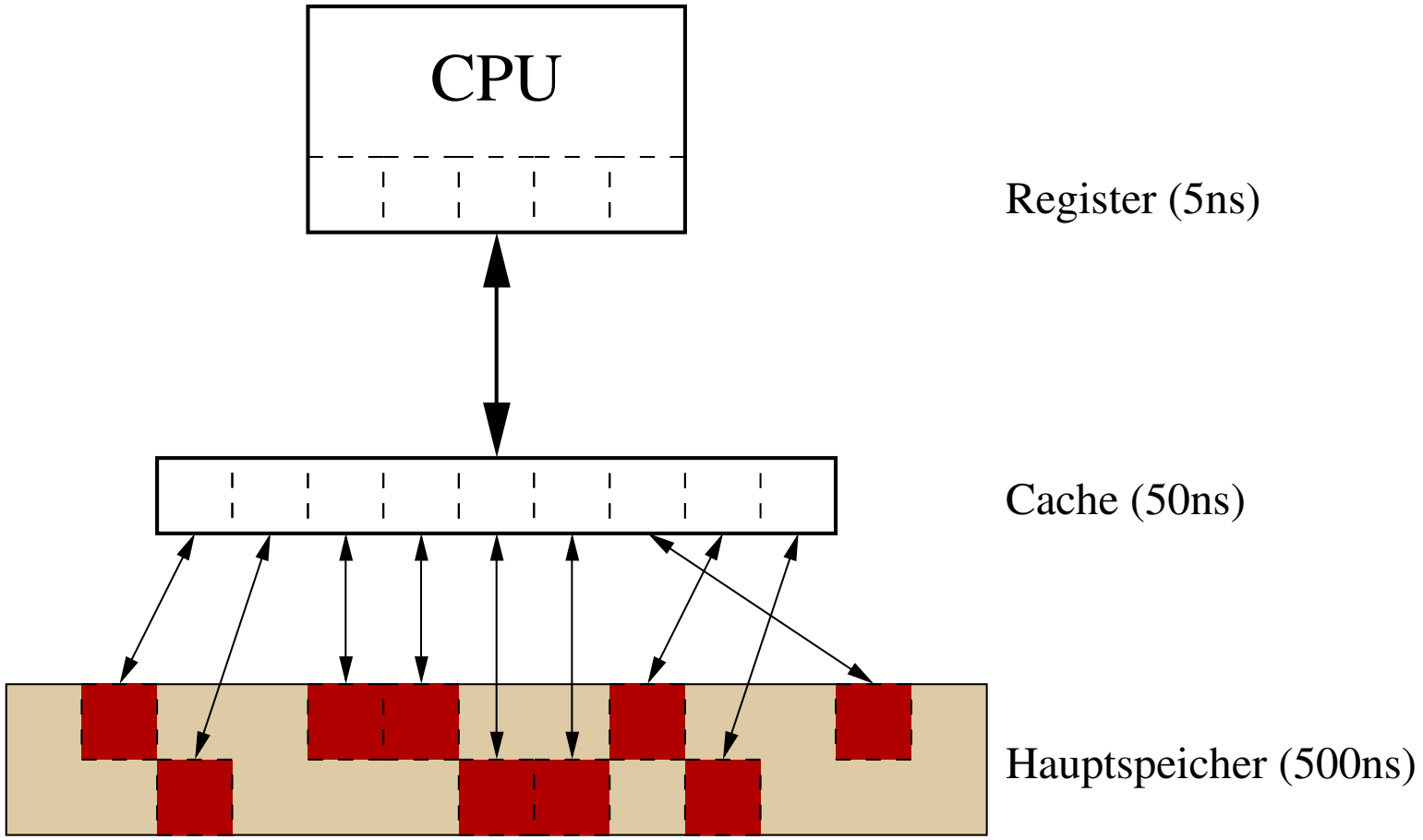
# 1994 - C. Nyberg, "AlphaSort"



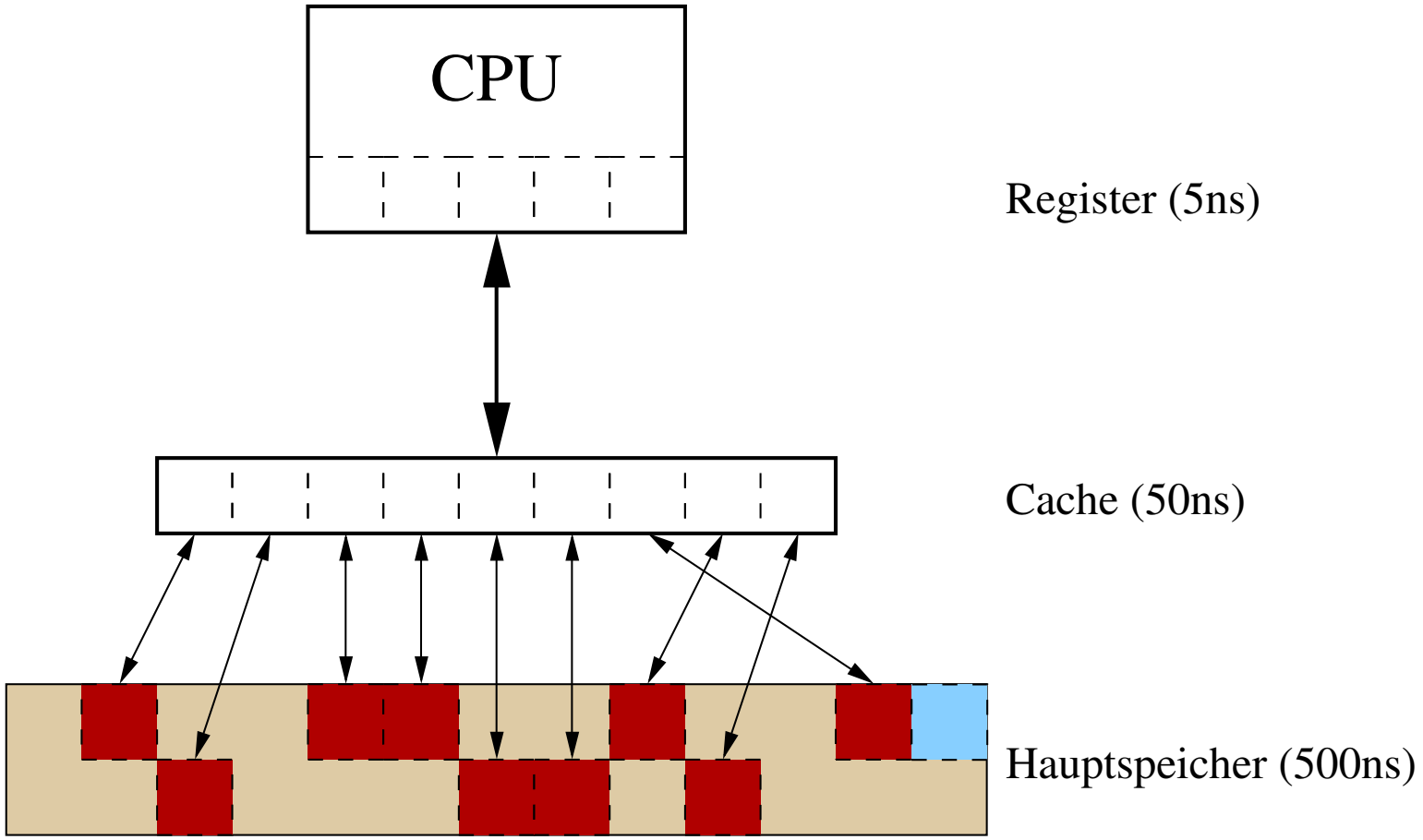
# 2001 - A. Tanenbaum, "Modern Operating Systems"



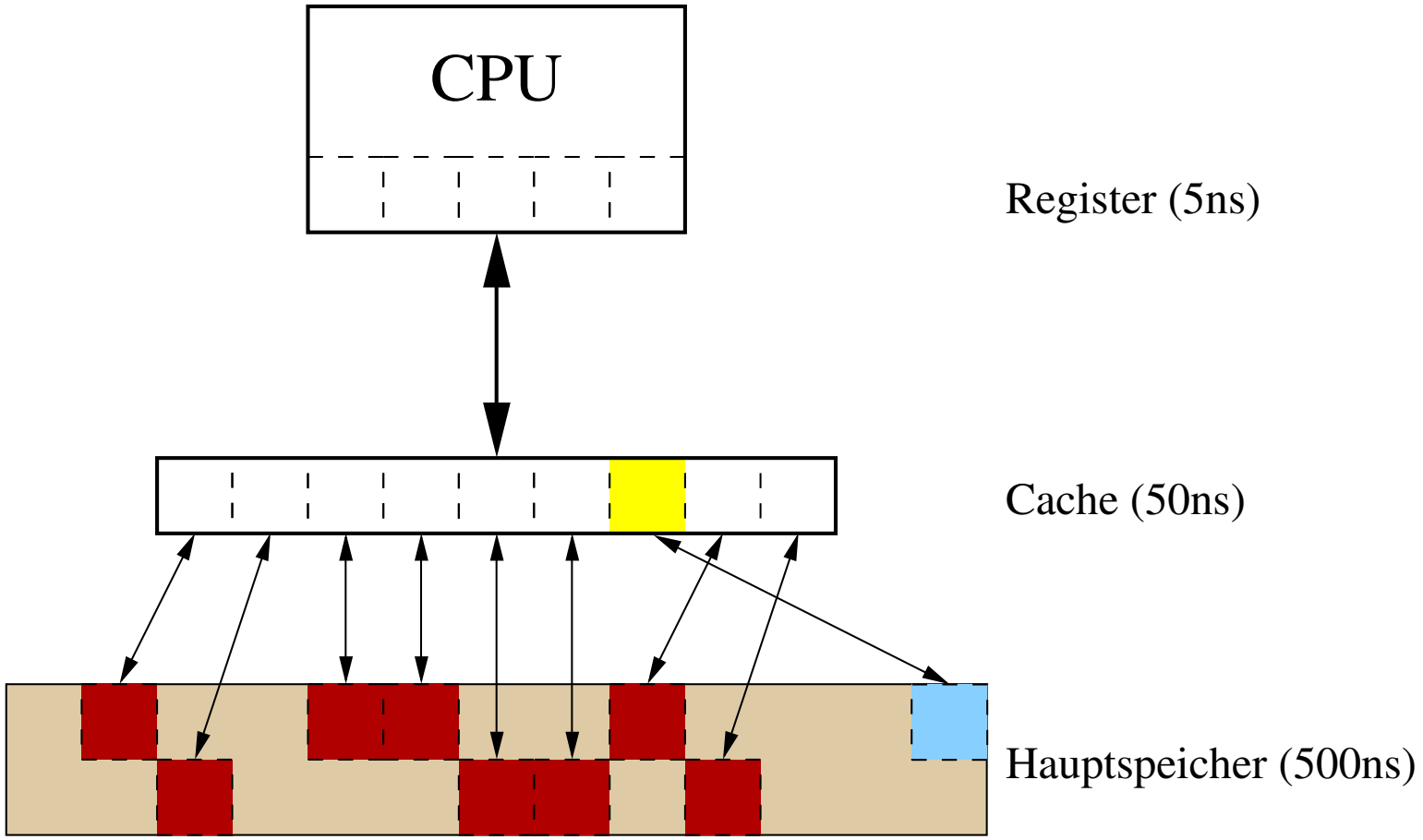
# Lokalität von Daten



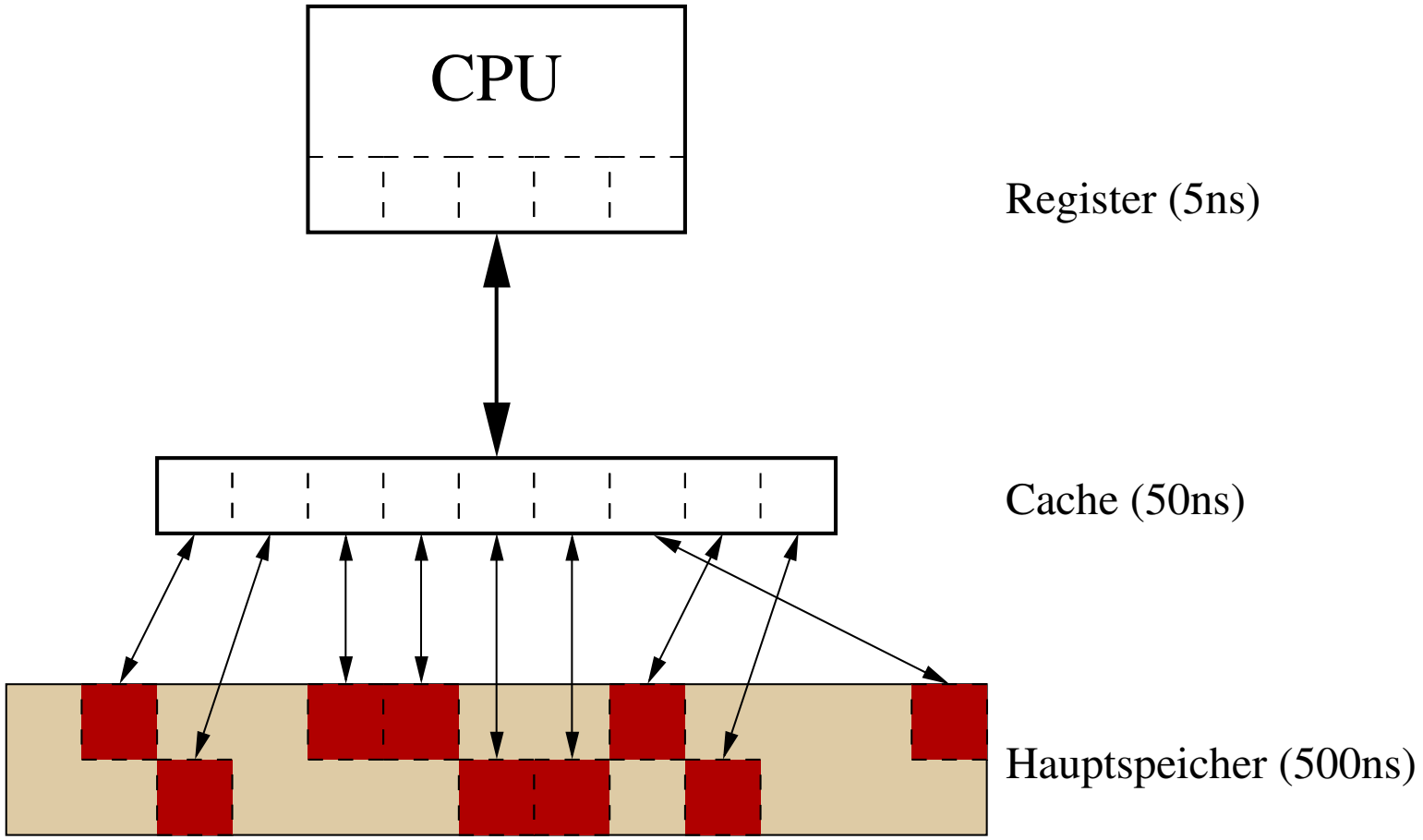
# Lokalität von Daten



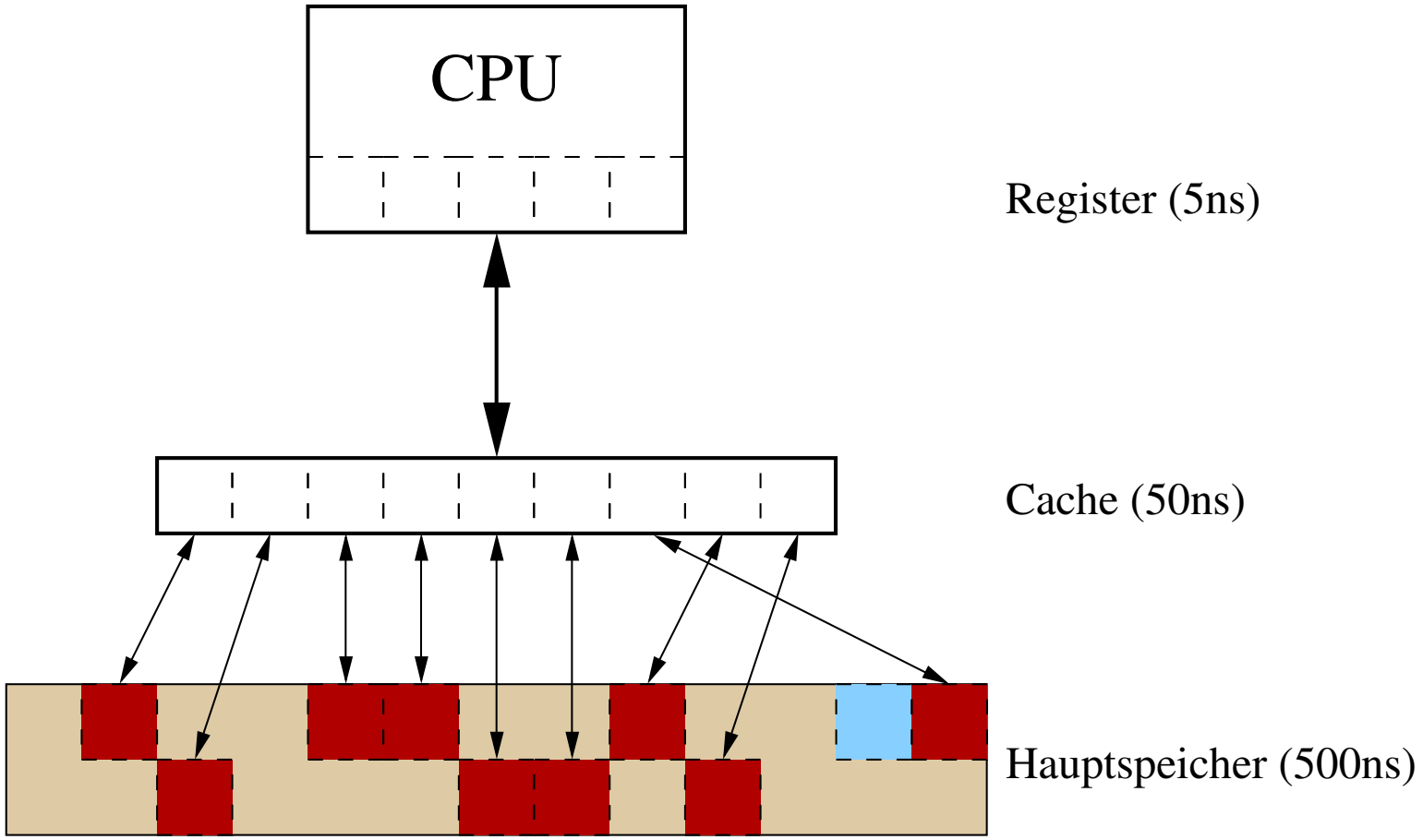
# Lokalität von Daten



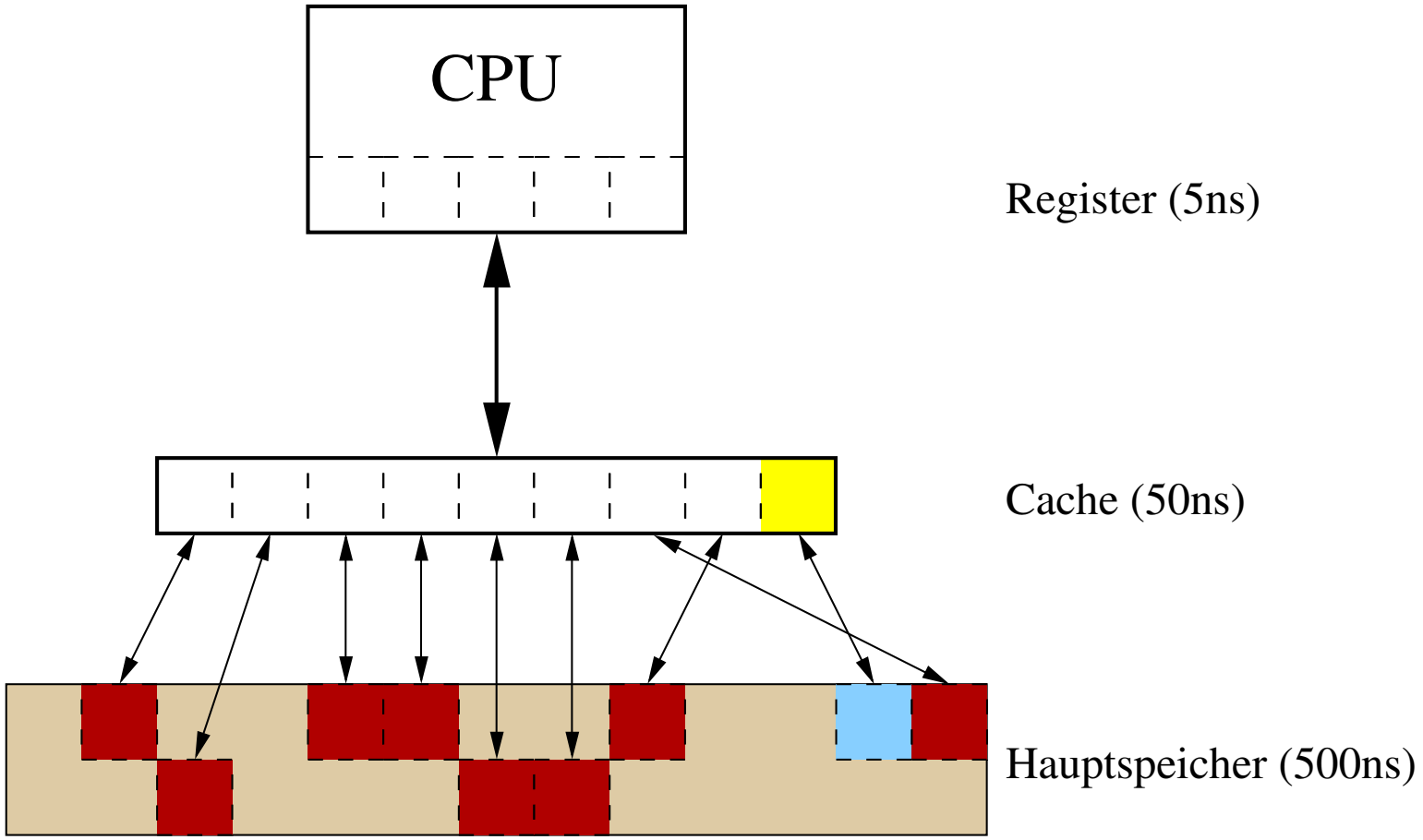
# Lokalität von Daten



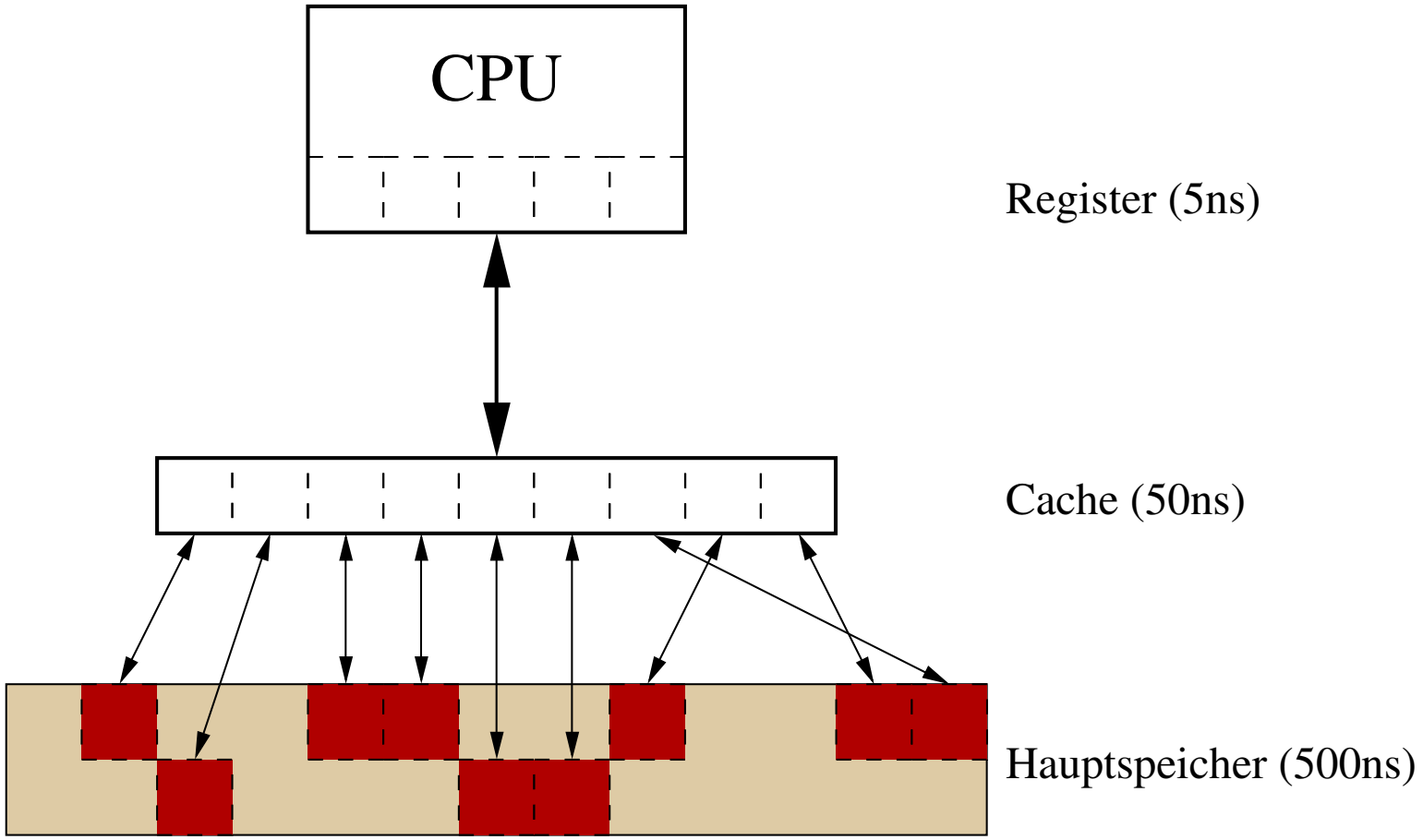
# Lokalität von Daten



# Lokalität von Daten



# Lokalität von Daten



# Der "Festplatten"-Flaschenhals

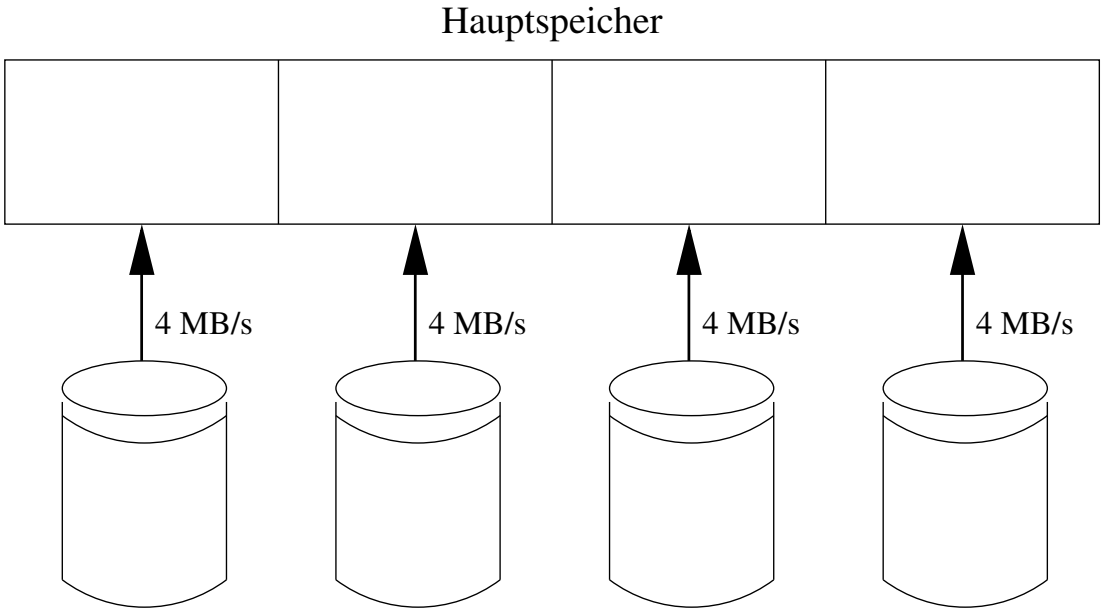
## Problem

- Festplatten haben geringen Datendurchsatz
- SCSI Festplatten 1994 max. 4 MB/s
- Lesen/Schreiben von 100 MB/s in min. 25 s

## Lösung

- Paralleler Datentransfer
- Gleichzeitig mehrere Festplatten lesen/schreiben
- (Möglicher Ansatz: RAID)

# Paralleler Datentransfer



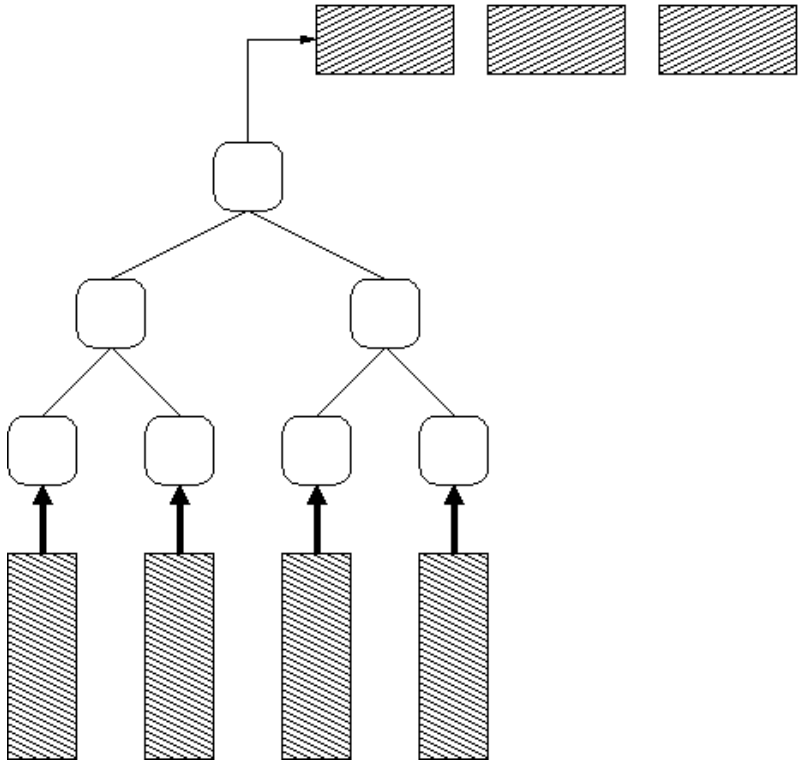

- Gesamtdatendurchsatz: $4 \cdot 4$ MB/s $= 16$ MB/s
- Festplatten-Controller erledigt Arbeit (DMA), CPU frei
- AlphaSort: 36 Festplatten mit je $1,8$ MB/s, d.h. ca. 64 MB/s Lesen und 49 MB/s Schreiben

# Übersicht

1. Einleitung
2. Benchmarks
3. Die Speicherhierarchie
4. Das AlphaSort Verfahren
5. Weitere Verfahren

# AlphaSort–I

- Verfahren zweistufig
- QuickSort
- Replacement-Selection
  (→ Vorlesung Algorithmen)



## Warum nicht z.B. HeapSort?

- HeapSort braucht alle Daten gleichzeitig im Speicher

## Warum nicht z.B. HeapSort?

- HeapSort braucht alle Daten gleichzeitig im Speicher
- CPU würde auf Festplatte warten, da zunächst 100 MB gelesen werden müssten (→ Speicherhierarchie)

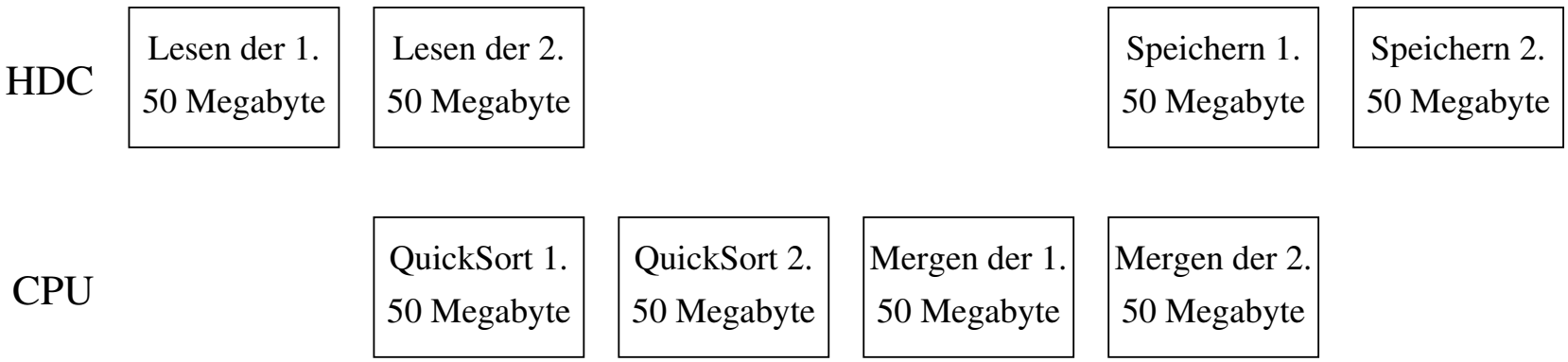
**Warum nicht z.B. HeapSort?**

- HeapSort braucht alle Daten gleichzeitig im Speicher
- CPU würde auf Festplatte warten, da zunächst 100 MB gelesen werden müssten (→ Speicherhierarchie)

**Stattdessen wünschenswert...**

- während Daten von Festplatten laden, die CPU mit Sortieren beauftragen

# Zeitlicher Ablauf




- Zerlegen der Eingabedaten in Blöcke
- Sortieren eines Blockes während der nächste geladen wird
- Zusammenführen von Blöcken (ähnlich MergeSort)
- Sobald erster Block fertig, schreiben
- 10 bis 30 Blöcke vorteilhaft

# QuickSort–I

## Frage

- Warum QuickSort?

# QuickSort–I

## Frage

- Warum QuickSort?

## Antwort

- QuickSort hat eine bessere Cache-Lokalität

## QuickSort–I

**Frage**

- Warum QuickSort?

**Antwort**

- QuickSort hat eine bessere Cache-Lokalität

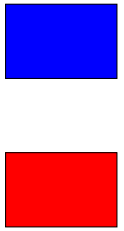
**Das heißt...**

- Daten werden bevorzugt im Cache vorgefunden
- Sortieren läuft (annähernd) mit Cache-Geschwindigkeit statt mit Hauptspeicher-Geschwindigkeit

# QuickSort–II

## Naiver Ansatz

- QuickSort eines Arrays mit Elementen zu je 100 Byte

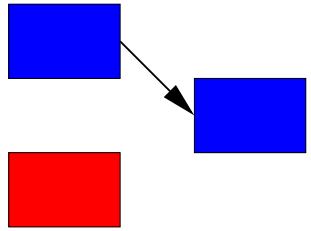


## Problem

- Vertauschen aufwendig

## Naiver Ansatz

- QuickSort eines Arrays mit Elementen zu je 100 Byte

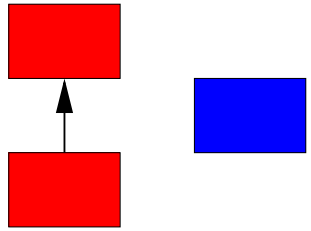


## Problem

- Vertauschen aufwendig

## Naiver Ansatz

- QuickSort eines Arrays mit Elementen zu je 100 Byte



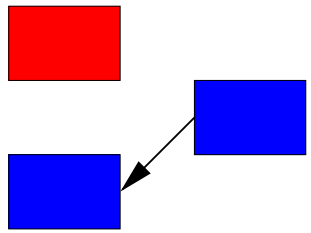
## Problem

- Vertauschen aufwendig

# QuickSort–II

## Naiver Ansatz

- QuickSort eines Arrays mit Elementen zu je 100 Byte



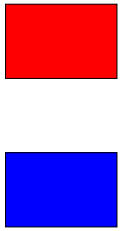
## Problem

- Vertauschen aufwendig

# QuickSort–II

## Naiver Ansatz

- QuickSort eines Arrays mit Elementen zu je 100 Byte

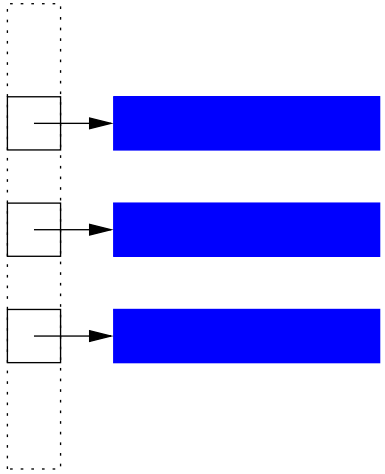


## Problem

- Vertauschen aufwendig

## Lösung

- Statt kompletter Datensätze lieber Referenzen (Pointer) sortieren
- Hilfs-Array mit Referenzen auf Datensätze erstellen



## Problem

- Sortieren mit ”Referenzen” schlecht im Sinne von Lokalität

## QuickSort–III

### Problem

- Sortieren mit "Referenzen" schlecht im Sinne von Lokalität

### Feststellung

- Zum Sortieren reicht Schlüssel

## QuickSort–III

### Problem

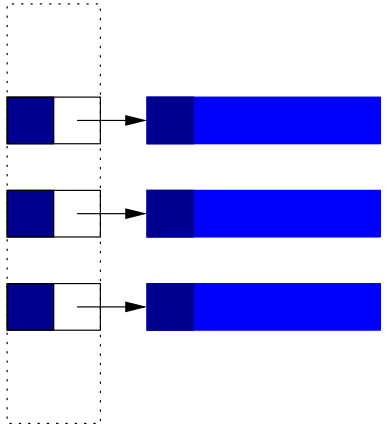
- Sortieren mit "Referenzen" schlecht im Sinne von Lokalität

### Feststellung

- Zum Sortieren reicht Schlüssel

### Lösung

- Schlüssel/Referenz-Paare in Hilfs-Array speichern ("Detached key sort" Lorin, 1974)
- Nur noch Schlüssel/Referenz-Paare vertauschen
- Sortieren fast ausschliesslich in Hilfs-Array ($\rightarrow$ gute Lokalität, Stichwort: Cachelines)



## QuickSort–III

### Problem

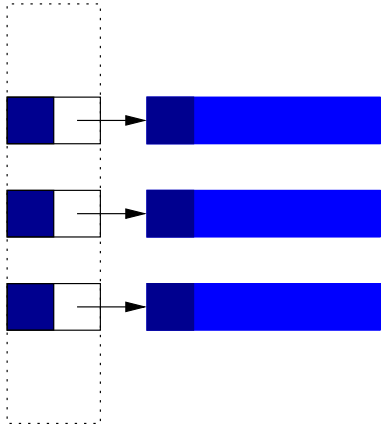
- Sortieren mit "Referenzen" schlecht im Sinne von Lokalität

### Feststellung

- Zum Sortieren reicht Schlüssel

### Lösung

- Schlüssel/Referenz-Paare in Hilfs-Array speichern ("Detached key sort" Lorin, 1974)
- Nur noch Schlüssel/Referenz-Paare vertauschen
- Sortieren fast ausschliesslich in Hilfs-Array ($\rightarrow$ gute Lokalität, Stichwort: Cachelines)



- Weitere Optimierung: Key-Prefix/Pointer Paare

# Zusammenfassung

- Daten werden blockweise gelesen
- Blöcke mit QuickSort vorsortieren
- Sobald alle Blöcke vorsortiert, blockweises Zusammenführen mit Replacement-Selection
- Sobald nächster Block fertig, anstossen des Schreibvorgangs

# Übersicht

1. Einleitung
2. Benchmarks
3. Die Speicherhierarchie
4. Das AlphaSort Verfahren
5. Weitere Verfahren

## Technische Daten

- Sieger Datamation 2001 in 0,32 Sekunden
- Linux/i386 Cluster mit 32 Knoten, je Pentium III, 896MB RAM und 5 x 8,5GB Festplattenkapazität
- Verbunden durch 100MBit und GigaBit Ethernet

## Gewonnene Erkenntnisse

- `rexec` und `ssh` ungeeignet für Lastverteilung (→ Overhead)
- TCP Handshake zu kostspielig im Bezug auf Startup-Zeit (→ Datamation Problematik), stattdessen UDP (→ Vorlesung Rechnernetze)

## Technische Daten

- Sieger TeraByte Indy Benchmark 2000-2003 in 1057 Sekunden
- IBM RS/6000 SP mit 488 Knoten und 2168 Festplatten