Kapitel 8 – Entwurfs- und Architekturkonzepte

1. VLSI-Entwurf
2. **Rechnerarchitektur**

Albert-Ludwigs-Universität Freiburg

UNI FREIBURG

Dr. Tobias Schubert, Dr. Ralf Wimmer

Professur für Rechnerarchitektur

WS 2016/17

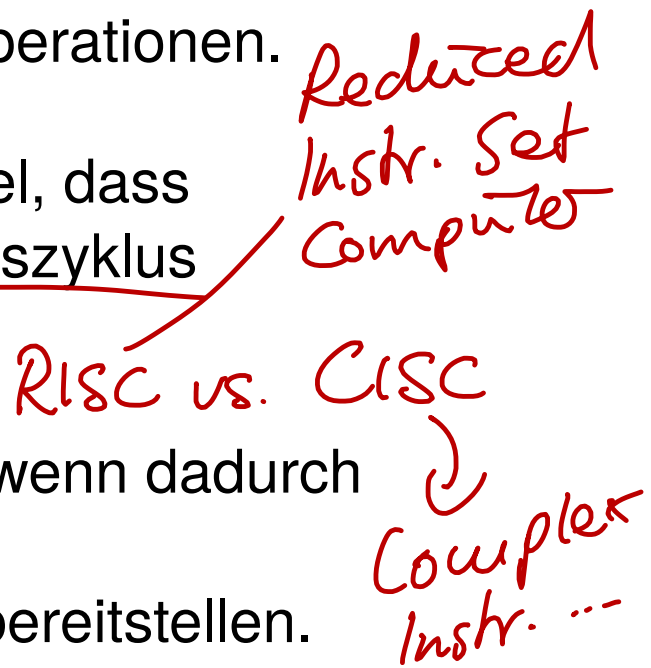
Rechnerarchitektur

- Designprinzipien
- Pipelining, Parallelverarbeitung
- Speicheroganisation

- Details s. Vorlesung Rechnerarchitektur

Designprinzipien für Rechner

- Was sind die Schlüsseloperationen für diesen speziellen Rechnertyp?
- Entwurf des Datenpfades (Rechenwerk mit Verbindungsstruktur) für diese Schlüsseloperationen.
- Entwurf der Maschinenbefehle mit dem Ziel, dass möglichst jeder Befehl in einem Datenpfadszyklus (Fetch-Execute) ausführbar ist.
- Erweiterung des Befehlssatzes nur dann, wenn dadurch die Maschine nicht langsamer wird; ggfs anwendungsspezifische Zusatzhardware bereitstellen.
- Schnittstelle nach außen (Speicher, Input/Output)



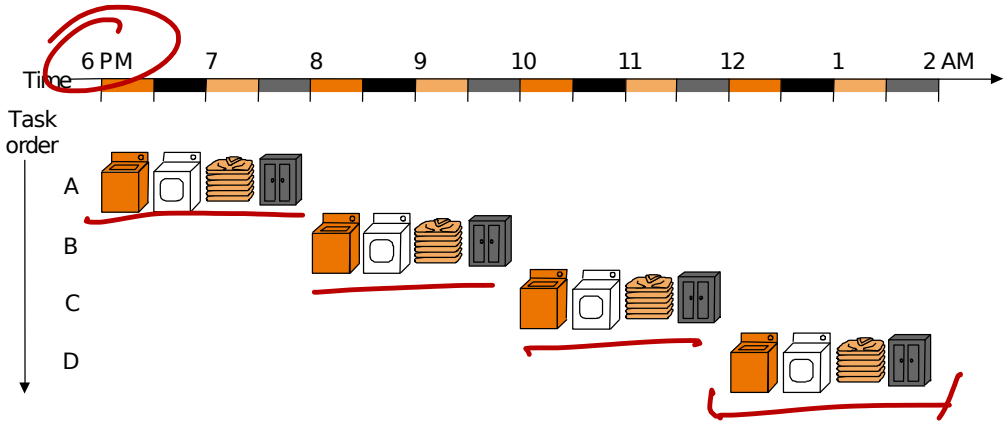
Pipelining

- Performanz von Rechnern lässt sich durch Pipelining steigern.
  - Prinzip der Fließbandverarbeitung
  - Probleme der Fließbandverarbeitung

Das Prinzip an einem alltäglichen Beispiel

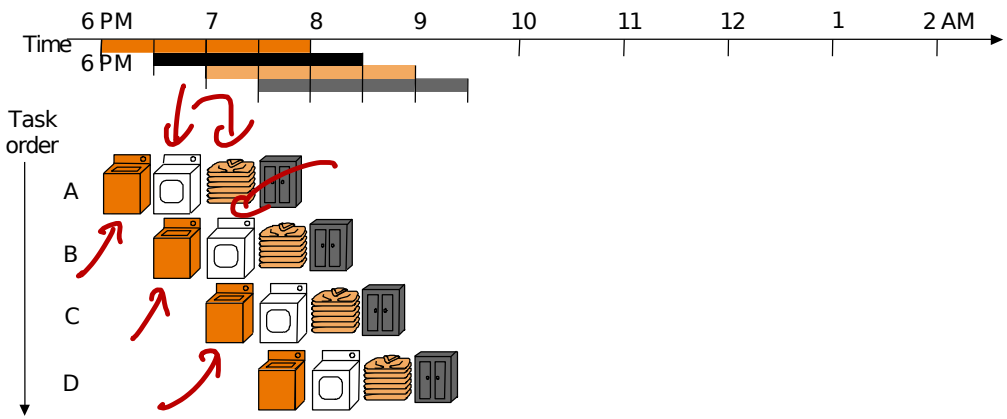
- Personen A, B, C und D kommen aus dem Urlaub; es ist viel schmutzige Wäsche zu waschen!
- Zur Verfügung stehen:
  - eine Waschmaschine (1/2 Stunde Laufzeit)
  - ein Trockner (1/2 Stunde Laufzeit)
  - ein Bügeleisen (1/2 Stunde Arbeit zum Bügeln)
  - ein Wäscheschrank (1/2 Stunde Arbeit zum Einräumen)
- jeder der Personen A, B, C, D aus dem Haushalt wäscht seine Wäsche selbst.
- Es gibt grundsätzlich zwei Möglichkeiten, die vier Waschvorgänge auszuführen!

Das Prinzip an einem Beispiel



Dauer der Arbeiten:
8 Stunden

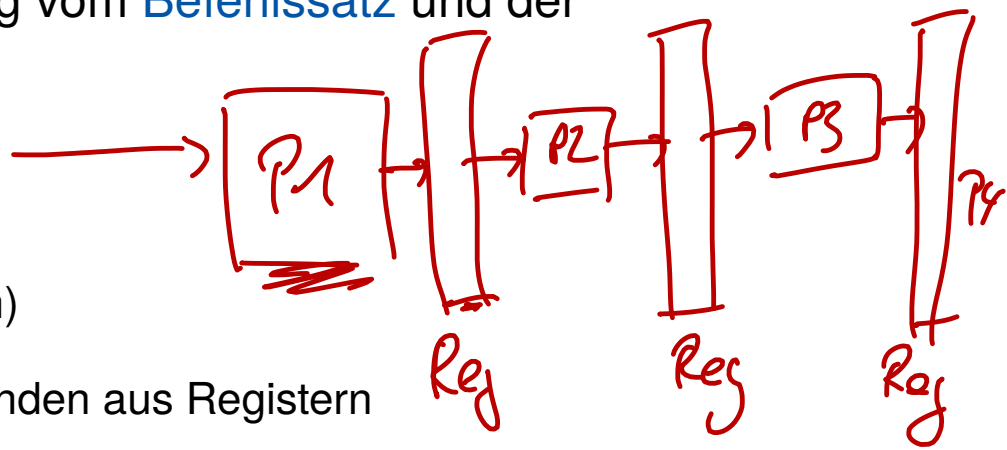
Mit Pipelining:



Dauer der Arbeiten:
$3\frac{1}{2}$ Stunden

Aufteilung der Befehlsabarbeitung in Phasen

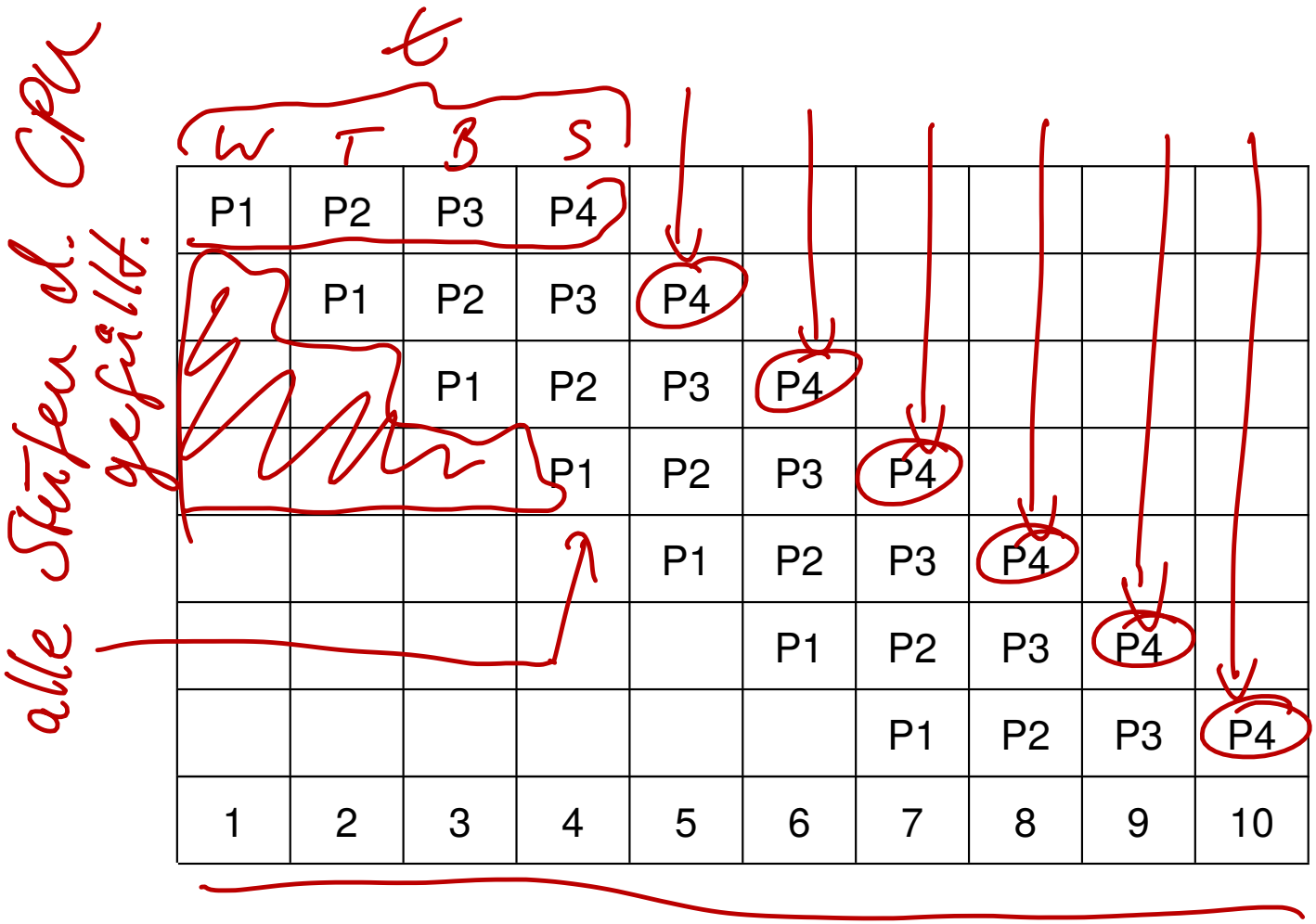
- Um Pipelining im Datenpfad ausnutzen zu können, muss die Abarbeitung eines Maschinenbefehls in mehrere Phasen mit möglichst gleicher Dauer aufgeteilt werden.
- Eine sinnvolle Aufteilung ist abhängig vom Befehlssatz und der verwendeten Hardware.
- Beispiel: Abarbeitung in 4 Schritten:
  - Befehls-Holphase (instruction fetch)
  - Dekodierphase / Lesen von Operanden aus Registern
  - Ausführung / Adressberechnung
  - Abspeicherphase (result write back phase)



- Pipelining ist schwierig, wenn die Dauer der Dekodier- und Ausführungsphase bei denen verschiedenen Maschinenbefehlen sehr unterschiedlich ist.

Pipelining: Illustration

- **Annahme**: Aufteilung der Befehlsabarbeitung in 4 gleichlange Phasen.



Befehl 1:	P1	P2	P3	P4						
Befehl 2:		P1	P2	P3	P4					
Befehl 3:			P1	P2	P3	P4				
Befehl 4:				P1	P2	P3	P4			
Befehl 5:					P1	P2	P3	P4		
Befehl 6:						P1	P2	P3	P4	
Befehl 8:							P1	P2	P3	P4
Zeitschritt:	1	2	3	4	5	6	7	8	9	10

Pipelining: Speedup (1/2)

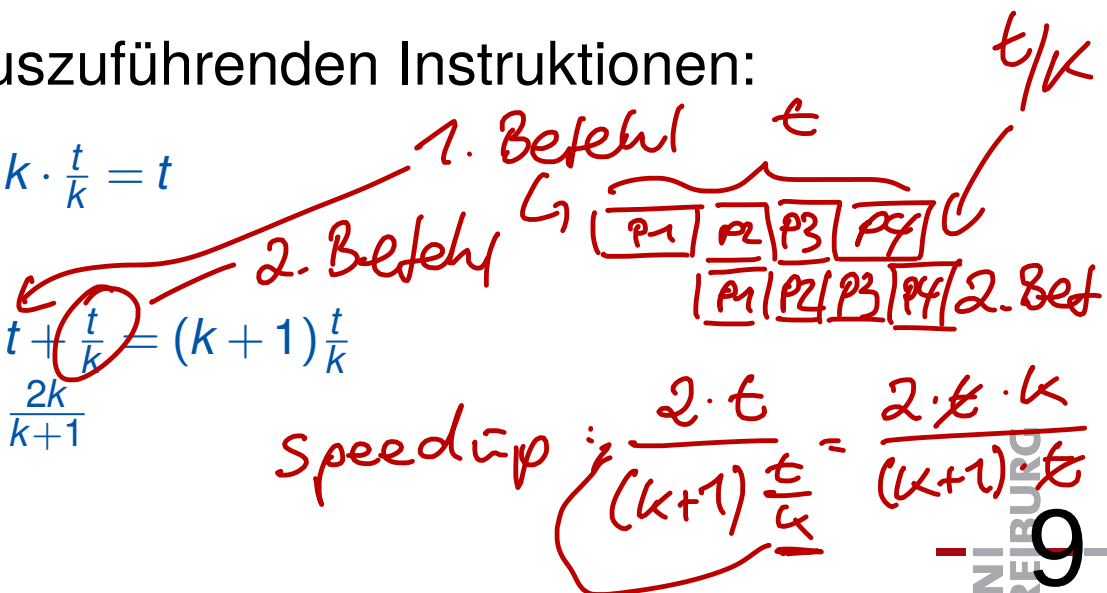
- **Annahmen**:
  - Abarbeitungszeit eines Befehls ohne Pipelining: t
  - k Pipelinestufen, gleiche Laufzeit der Stufen

  $\Rightarrow$ Laufzeit einer Stufe der Pipeline: $\frac{t}{k}$

- Beschleunigung bei m auszuführenden Instruktionen:

  $m = 1$: Laufzeit mit Pipeline $= k \cdot \frac{t}{k} = t$
  $\Rightarrow$ keine Beschleunigung

  $m = 2$: Laufzeit mit Pipeline $= t + \frac{t}{k} = (k+1)\frac{t}{k}$
  $\Rightarrow$ Beschleunigung um Faktor $\frac{2k}{k+1}$



Pipelining: Speedup (2/2)

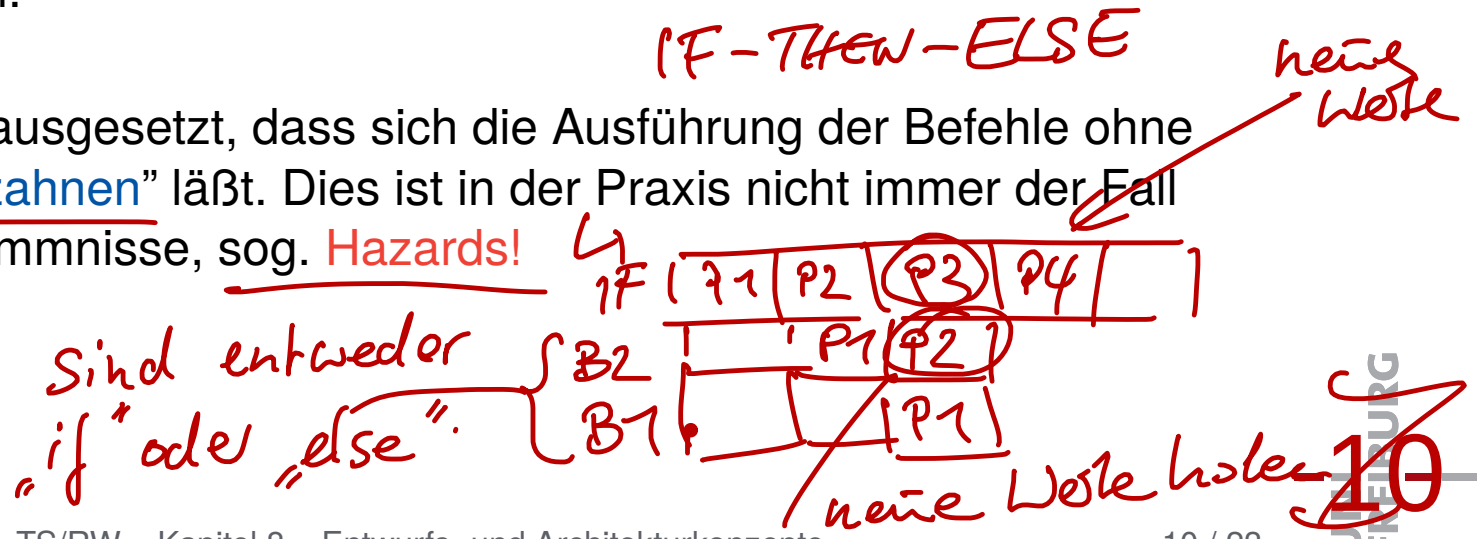
- $m \geq 1$: Laufzeit mit Pipeline $= t + (m-1)\frac{t}{k}$,
  Laufzeit ohne Pipeline $= m \cdot t$.

  $\Rightarrow$ Beschleunigung um $\frac{mt}{t+(m-1)\cdot\frac{t}{k}} = \frac{mk}{m+k-1} = k - \frac{k(k-1)}{m+k-1}$

- **Ergebnis**:

  Für $m >> k$ nähert sich der Speedup also der Anzahl k der Pipelinestufen.

- Es wurde vorausgesetzt, dass sich die Ausführung der Befehle ohne weiteres „verzahnen" läßt. Dies ist in der Praxis nicht immer der Fall
  $\Rightarrow$ Pipelinehemmnisse, sog. Hazards!



Pipelining: Zusammenfassung

- Beschleunigung um bis zu Faktor k durch Einsatz einer Pipeline
  (k = Anzahl der Pipeline-Stufen).
- Hazards verringern die Beschleunigung.
- Viele Möglichkeiten Hazards zu vermeiden, sowohl durch Software- als auch Hardwaremaßnahmen

Übersicht: Formen der Parallelität

- Parallelität auf Bitebene: bis etwa 1985
  - Kombinatorische Addierer und Multiplizierer, etc.
  - wachsende Wortbreite auf 64 Bit
- Parallelität auf Instruktionsebene: 1985 bis heute
  - Pipelining der Instruktionsverarbeitung
  - Mehrere Funktionseinheiten (superskalare Prozessoren) bei mehr als 4 Funktionseinheiten werden Datenabhängigkeiten oft zum Hindernis für eine effiziente Ausnutzung.
  - Vektorprozessoren führen eine Operation parallel auf vielen Daten durch
- Parallelität auf Prozessor-/Rechnerebene
  - Nur so scheinen Beschleunigungen um Faktor 50 und mehr möglich.

Hm?

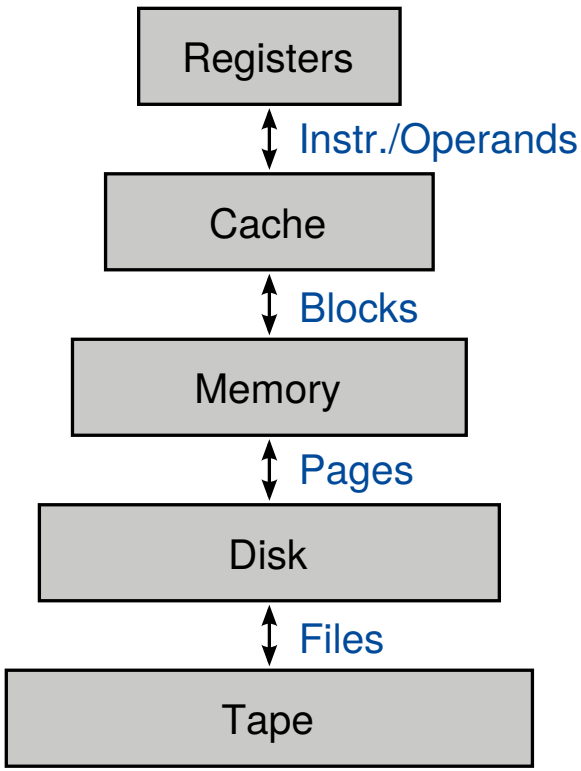
Speicherorganisation im Überblick

- **Fragen**:
  - Wie kann verhindert werden, dass ein Rechner durch Hauptspeicherzugriffe zu sehr verlangsamt wird?
  - Was passiert, wenn nicht alle Daten in den Hauptspeicher passen?
  - Welche sonstigen Speichermedien gibt es?
- **Begriffe**:
  - Speicherhierarchie
  - Cache
  - Virtueller Speicher, Verdrängungsstrategien
  - Festplatte, Flash, CDROM / DVD / Blu-Ray, Magnetband

Gründe für komplexe Speicherorganisation

- Ein Zugriff auf eine Hauptspeicherzelle ist langsamer, als ein Zugriff auf ein Register.
  - Hauptspeicherzellen sind DRAM-Zellen (dynamische Speicherzellen), während Register in der Regel SRAM-Zellen (statische Speicherzellen) sind!
  - Bei einem Registerzugriff kommt man ohne Bus-Operation aus!
- **Idee**: Man stellt dem Prozessor einfach einige Mbyte Register zur Verfügung.
  - Aber: SRAM-Zellen sind wesentlich größ er als DRAM-Zellen (Faktor 3-4).
  - So abwegig ist die Idee nicht!
  - Mit der weiteren Technologieentwicklung (noch kleinere Strukturen) wird die verfügbare Chip-Fläche vorwiegend dazu benutzt werden, um schnellen Speicher zu integrieren.

Speicherorganisation heute

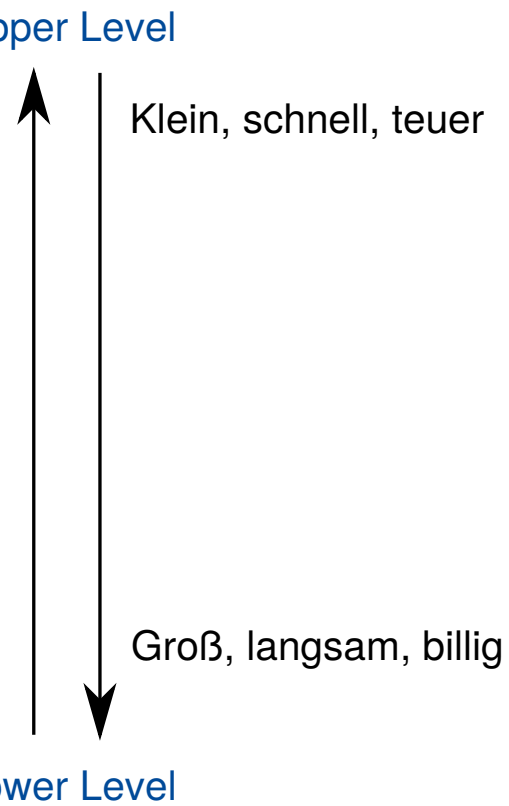


Programm
1-8 bytes

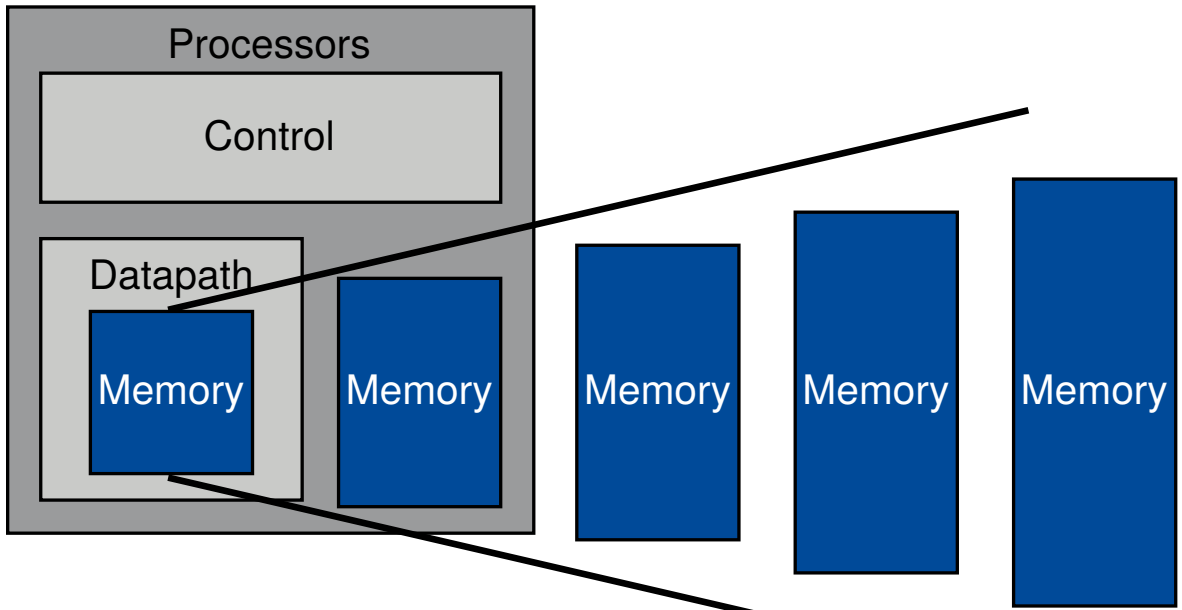
Cache
8-128 bytes

OS
512-4K bytes

Benutzer
MByte, GByte, TByte

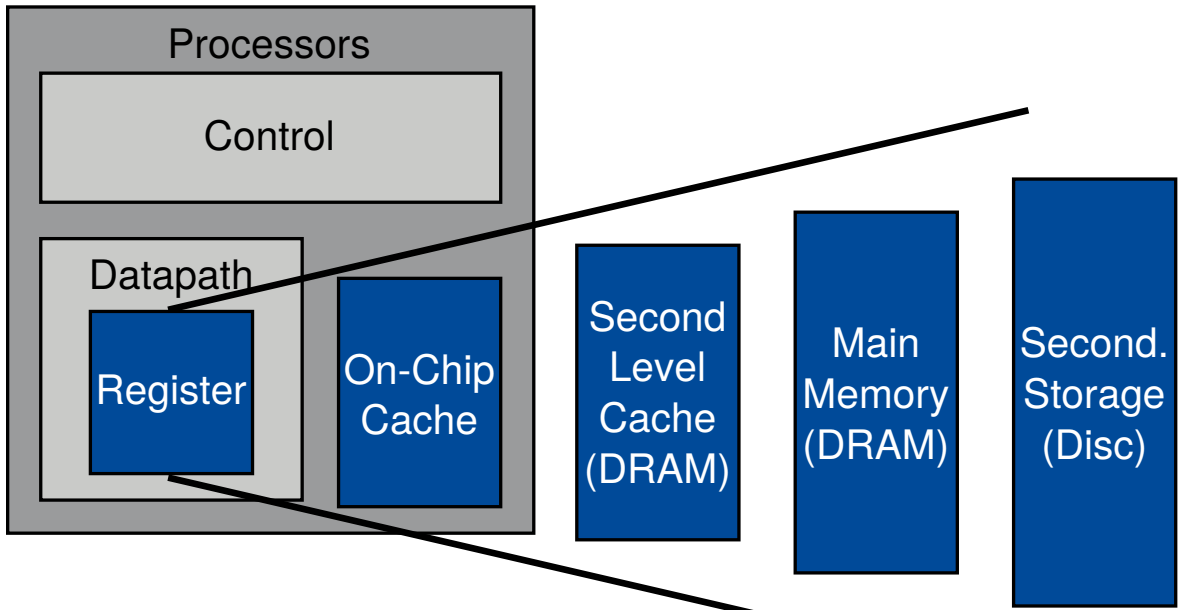


Speicherhierarchie (1/3)



Geschwindigkeit:	Am schnellsten	Am langsamsten
Größe:	Am kleinsten	Am größten
Kosten:	Am teuersten	Am billigsten

Speicherhierarchie (2/3)



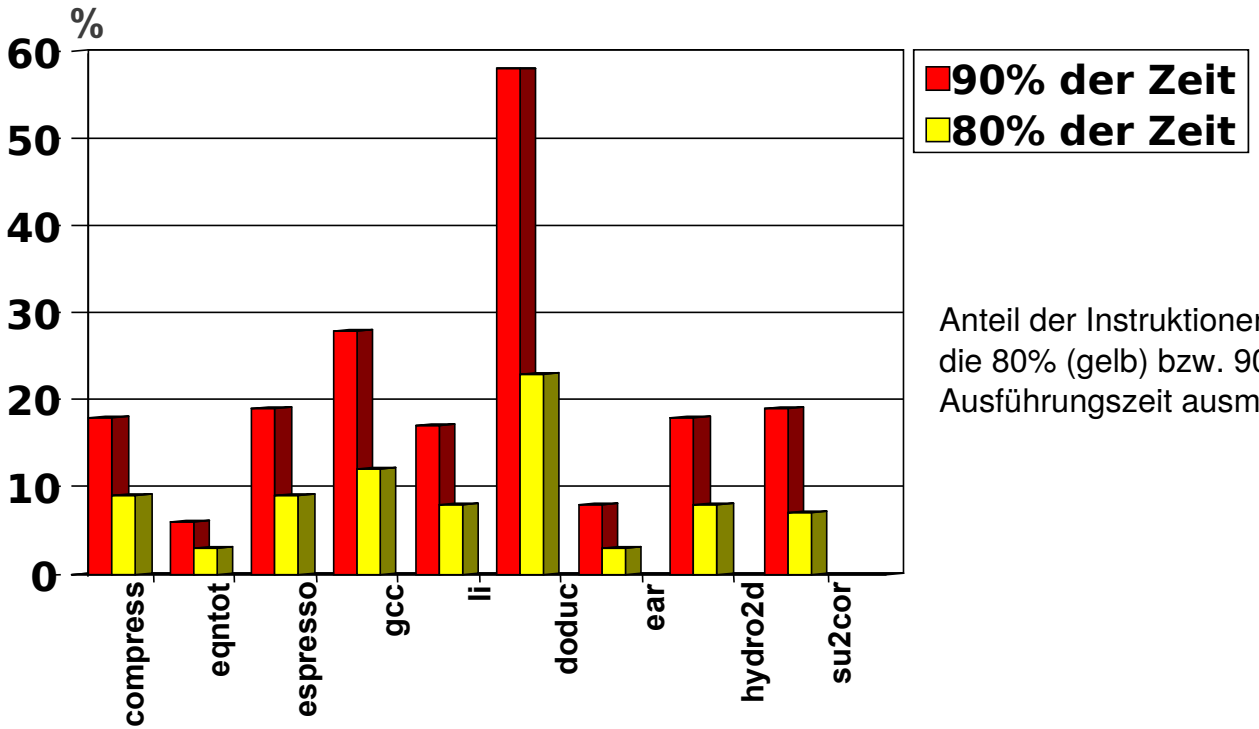
Geschwindigkeit:	Am schnellsten	Am langsamsten
Größe:	Am kleinsten	Am größten
Kosten:	Am teuersten	Am billigsten

Speicherhierarchie (3/3)

- Grundprinzip:
  - Wenig schneller und teurer Speicher
  - Viel langsamer und billiger Speicher
  - Speicherhierarchie
- Wieso funktioniert das?
  - Wegen Prinzip der Lokalität!

Prinzip der Lokalität (1/2)

Typische Programme verbringen 80% der Ausführungszeit in etwa 10% des Gesamtprogramms.



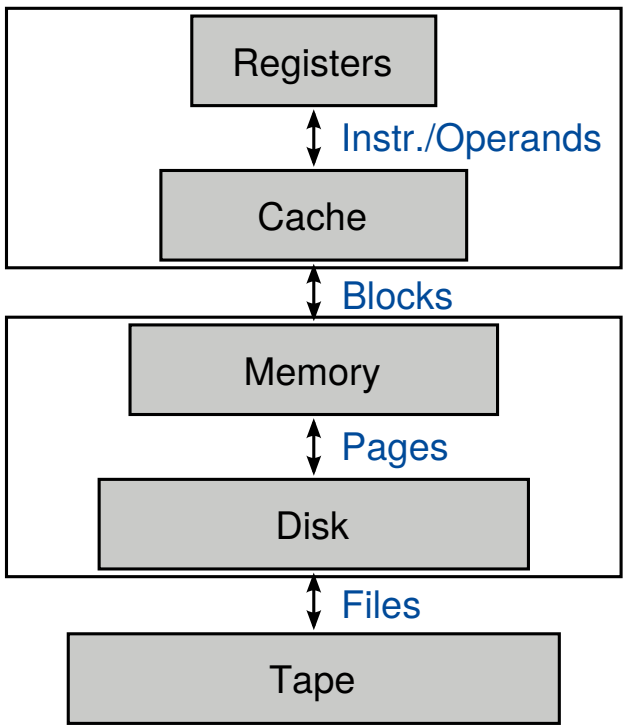
Anteil der Instruktionen des Gesamtcodes, die 80% (gelb) bzw. 90% (rot) der Ausführungszeit ausmachen.

Patterson, Hennessey Computer Architecture a Quantitative Approach

Prinzip der Lokalität (2/2)

- Es gibt zwei Arten von Lokalität:
  - Zeitliche Lokalität (Locality in Time): Wenn ein Datum verwendet wird, wird es auch bald wieder verwendet werden.
  - Räumliche Lokalität (Locality in Space): Wenn ein Datum verwendet wird, werden auch die Adressen in dessen Nähe bald verwendet werden.
  - Siehe z.B. Schleifen ...
- Wegen Lokalität kommt man auf höheren Ebenen der Speicherhierarchie mit weniger Speicher aus.

Speicherorganisation heute

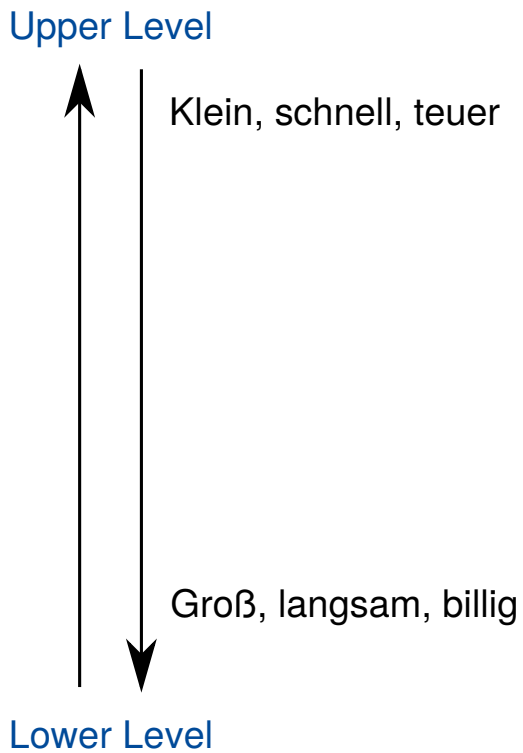


Programm
1-8 bytes

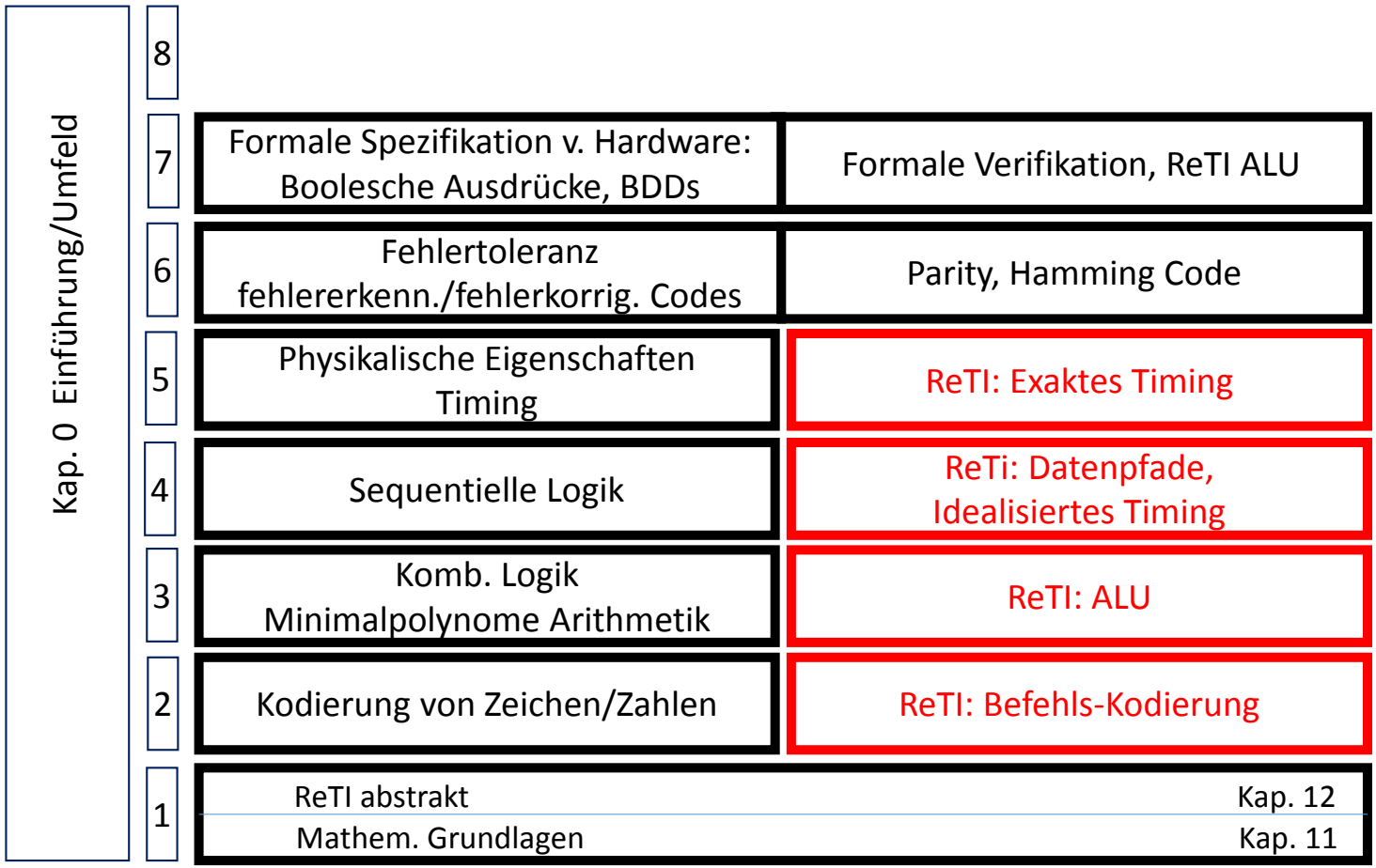
Cache
8-128 bytes

OS
512-4K bytes

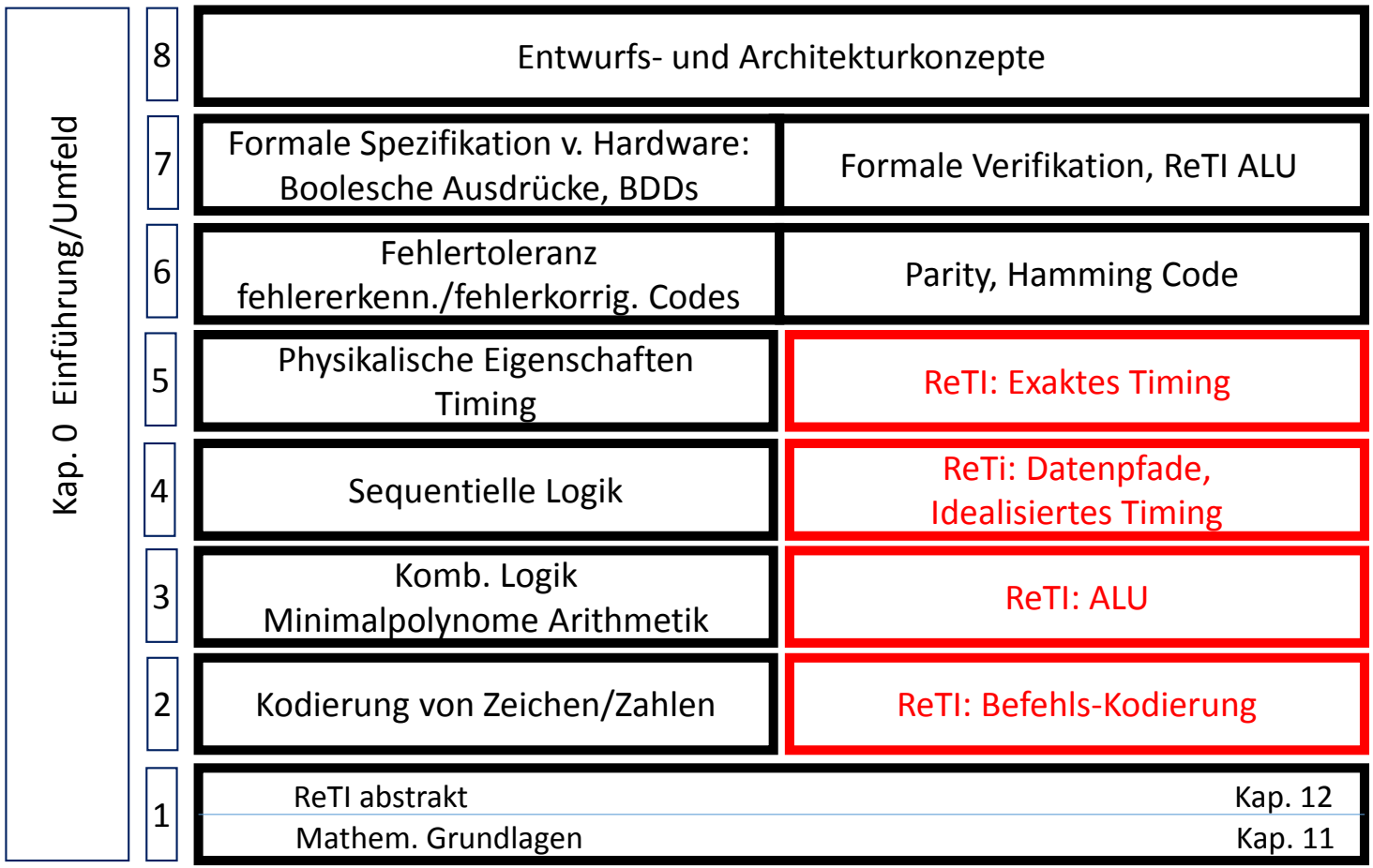
Benutzer
MByte, GByte, TByte



Überblick


Kap. 0 Einführung/Umfeld
8
7
Formale Spezifikation v. Hardware: Boolesche Ausdrücke, BDDs
Formale Verifikation, ReTI ALU
6
Fehlertoleranz fehlererkenn./fehlerkorrig. Codes
Parity, Hamming Code
5
Physikalische Eigenschaften Timing
ReTI: Exaktes Timing
4
Sequentielle Logik
ReTi: Datenpfade, Idealisiertes Timing
3
Komb. Logik Minimalpolynome Arithmetik
ReTI: ALU
2
Kodierung von Zeichen/Zahlen
ReTI: Befehls-Kodierung
1
ReTI abstrakt
Kap. 12
Mathem. Grundlagen
Kap. 11

Überblick


Kap. 0 Einführung/Umfeld
8
Entwurfs- und Architekturkonzepte
7
Formale Spezifikation v. Hardware: Boolesche Ausdrücke, BDDs
Formale Verifikation, ReTI ALU
6
Fehlertoleranz fehlererkenn./fehlerkorrig. Codes
Parity, Hamming Code
5
Physikalische Eigenschaften Timing
ReTI: Exaktes Timing
4
Sequentielle Logik
ReTi: Datenpfade, Idealisiertes Timing
3
Komb. Logik Minimalpolynome Arithmetik
ReTI: ALU
2
Kodierung von Zeichen/Zahlen
ReTI: Befehls-Kodierung
1
ReTI abstrakt
Kap. 12
Mathem. Grundlagen
Kap. 11