7. Rechenleistung

Leistungsmaß für Rechner ist die MIPS-Rate μ (Millionen von Instruktionen pro Sekunde). Wenn

- n: die mittlere Anzahl Zyklen (Takte) pro Befehl;
- T: Zykluszeit in Mikrosekunden [μs];
- n·T : Latenzzeit, Zeit zum Abarbeiten eines Befehls
- 1/T ist die Taktrate.

Dann lautet für eine einstufige CPU die Definition für μ [MIPS]:

$\mu = 1 / n \cdot T$

Bei mehrstufiger Pipeline bleibt zwar die Latenzzeit gleich, jedoch wird pro Takt etwa eine Instruktion abgearbeitet:

μ **= 1 / T**

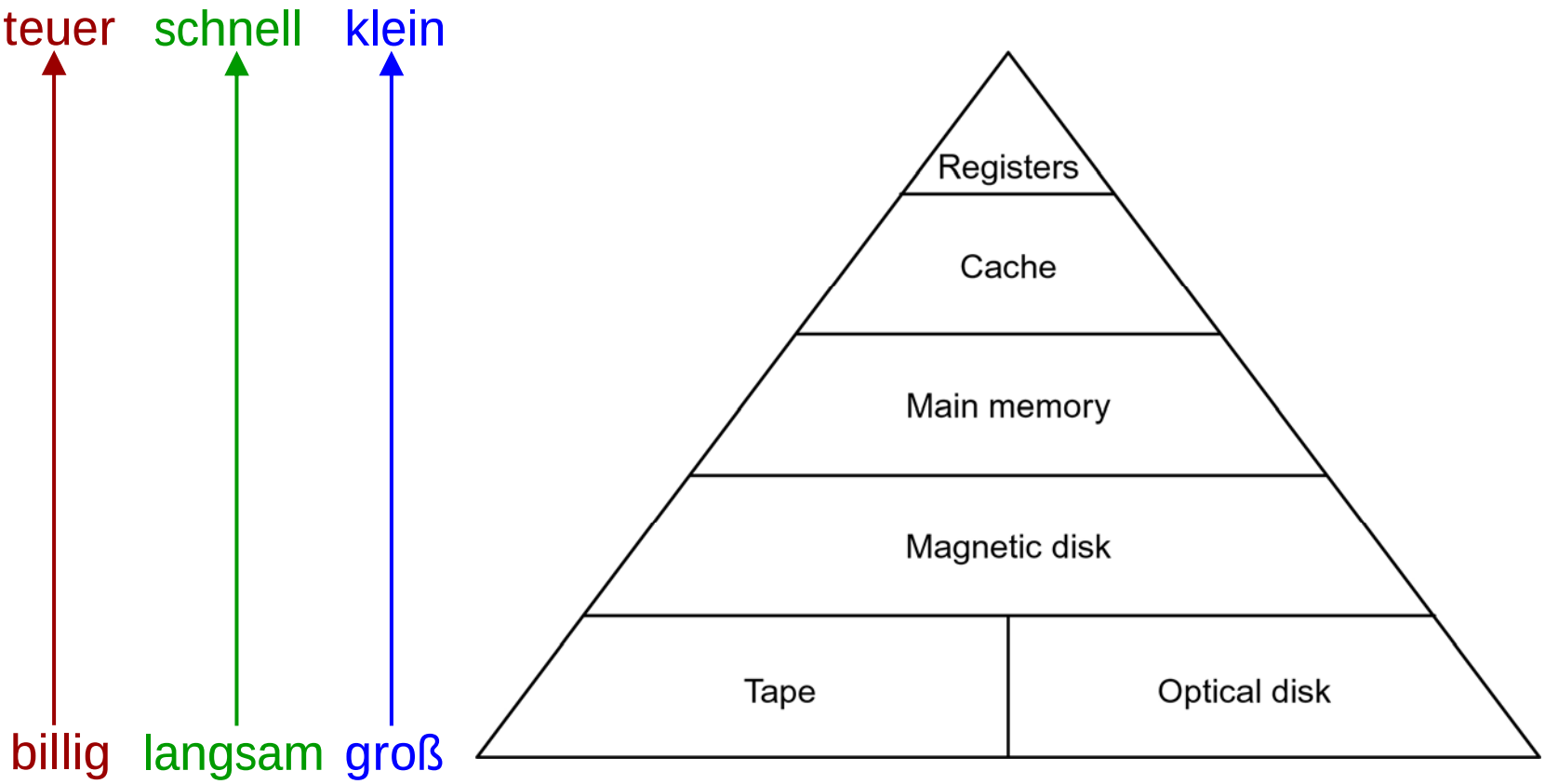
Beachte: MIPS ist eine einseitige Bewertung

- Die relative Performanz hängt noch von Programmlänge für CISC und RISC ab.
- hängt ab vom Befehlsvorrat und vom Compiler
- variiert zwischen Programmen auf derselben Maschine, kann invers zur Leistung sein (!). Beispiel: optimierter Code ist schneller bei weniger MIPS.

8. Speicherhierarchie

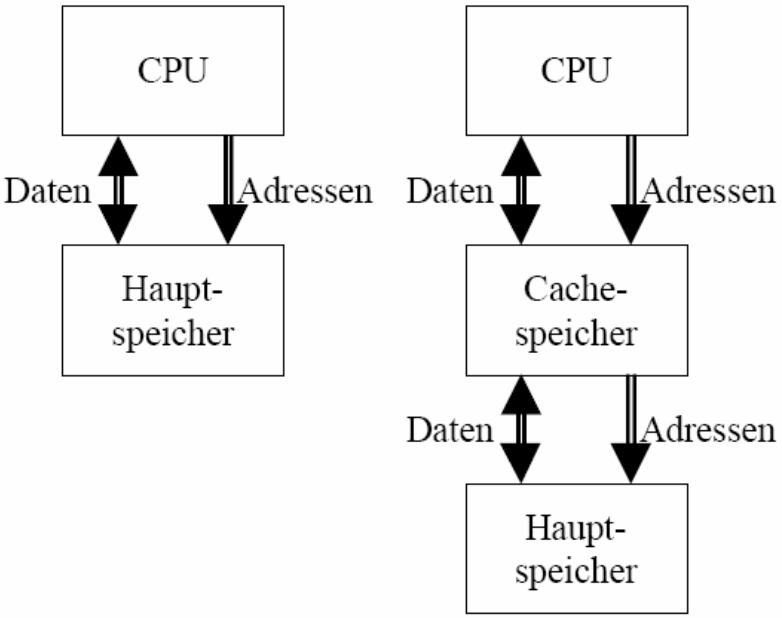
1. Cache-Speicher, Grundprinzipien
2. Cache-Organisation
3. Kenngrößen
4. Schreiben von Daten
5. Lesen von Daten

8. Speicherhierarchie



In Rechnern werden sehr unterschiedliche Arten von Speichern eingesetzt, die sich stark in Preis und Leistung unterscheiden. → **Speicherhierarchie**

8.1 Cache-Speicher, Grundprinzipien



1. Zugriff auf Cache deutlich schneller als auf Hauptspeicher
2. CPU kann auf Daten im Cache sehr schnell zugreifen
3. Cache viel kleiner als Hauptspeicher

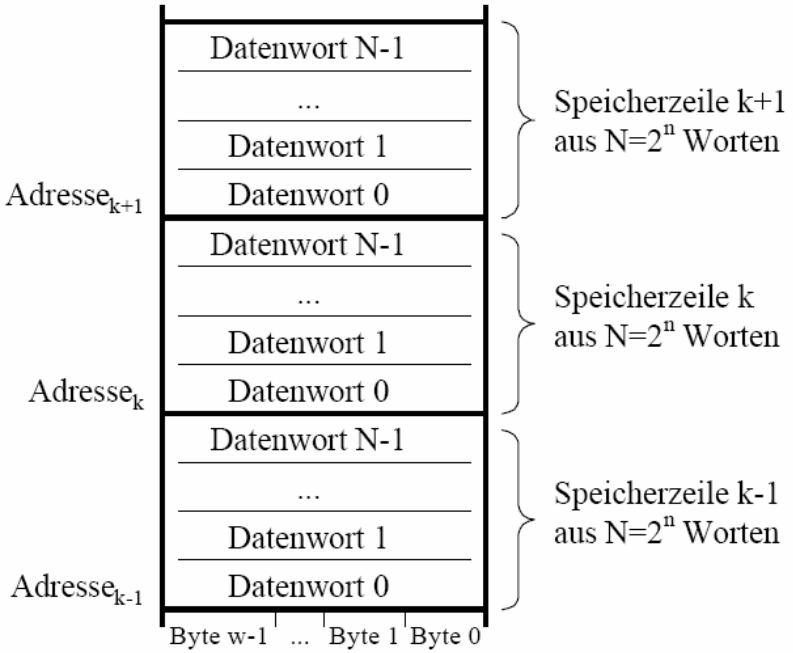
$$\text{Beschl.faktor}_{\text{gesamt}} = \frac{1}{(1 - \text{Anteil}_{\text{beschl.}}) + \frac{\text{Anteil}_{\text{beschl.}}}{\text{Beschl.faktor}}}$$

Amdahl's Gesetz:

Beispiel: Cache 10x schneller, Zugriff auf Cache in 90% d. Fälle

$$\text{Beschl.faktor}_{\text{gesamt}} = \frac{1}{(1 - 0{,}9) + \frac{0{,}9}{10}} = \frac{1}{0{,}19} = 5{,}3$$

8.1 Cache-Speicher, Speicheradressierung



Speicheradressierung:

32 Bit Speicheradresse

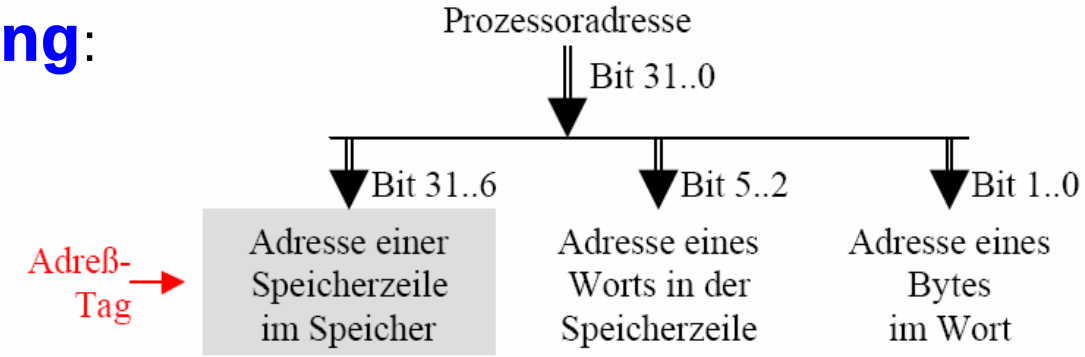
(2^{32}=4 294 967 296 Adressen)

- Speicherwort: 4 Byte (2^w)
- Speicherzeile: 16 Worte (2^n)
- Speicherzeilen: 2^{32-n-w} = 67 108 864 Zeilen

Speicheradressierung:

Cache speichert:

- Adress-Tag
- Kopie der Speicherzeile



8.1 Cache-Speicher, Speicheradressierung

Cache-Speicher

Adreß-Tag	Datenwort N-1	...	Datenwort N-2	Datenwort 0
0x0100	0x4210	...	0x3344	0x3449
0x2000	0xFE23	...	0xAA33	0xB255
0x1C00	0x5E93	...	0xD1E4	0xCC59
...	...	...	...	...
0x4600	0x5001	...	0xE4E4	0xD453

Cache-Zeile (Line)

Adressen der Zeilen im Hauptspeicher

Kopien von Speicherzeilen des Hauptspeichers

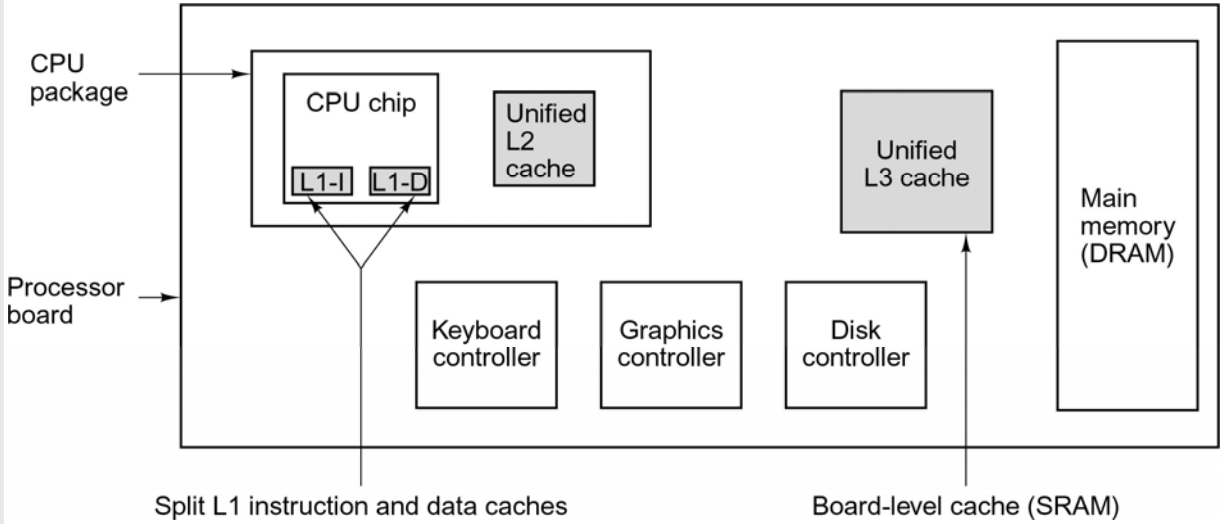
Wenn Wort aus Hauptspeicher gelesen werden soll, prüft Cache-Logik, ob Wort bereits im Cache ist → **cache hit**.

Wenn nicht → **cache miss**, dann muss Wort aus dem HS gelesen werden. Wird gleichzeitig im Cache gespeichert.

Soll ein (verändertes) Wort geschrieben werden, muss dies sowohl im Cache als auch im HS geschehen → **write through**.

Andere Strategien schreiben nur in Cache und markieren Eintrag (**dirty**).

8.2 Cache-Organisation



- Level-1: 16 ... 64 kB
- Level-2: 512 kB ... 1 MB
- Level-3: ~ MB SRAM

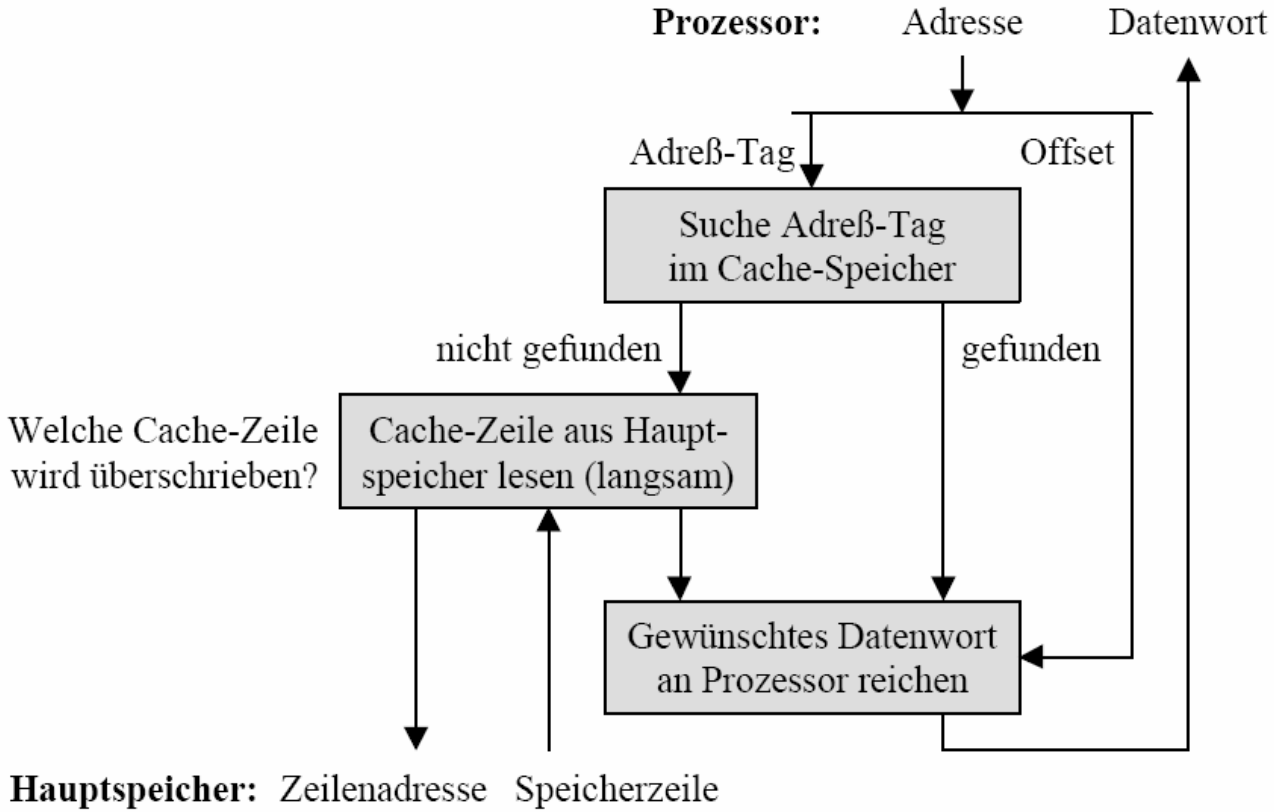
Um Rechner weiter zu optimieren werden mehrere Cache Ebenen eingefügt.

kleiner, schneller, teurer → größer, langsamer, billiger

→ **Split-Cache**, verbessert **Bandbreite** und **Latenz**:

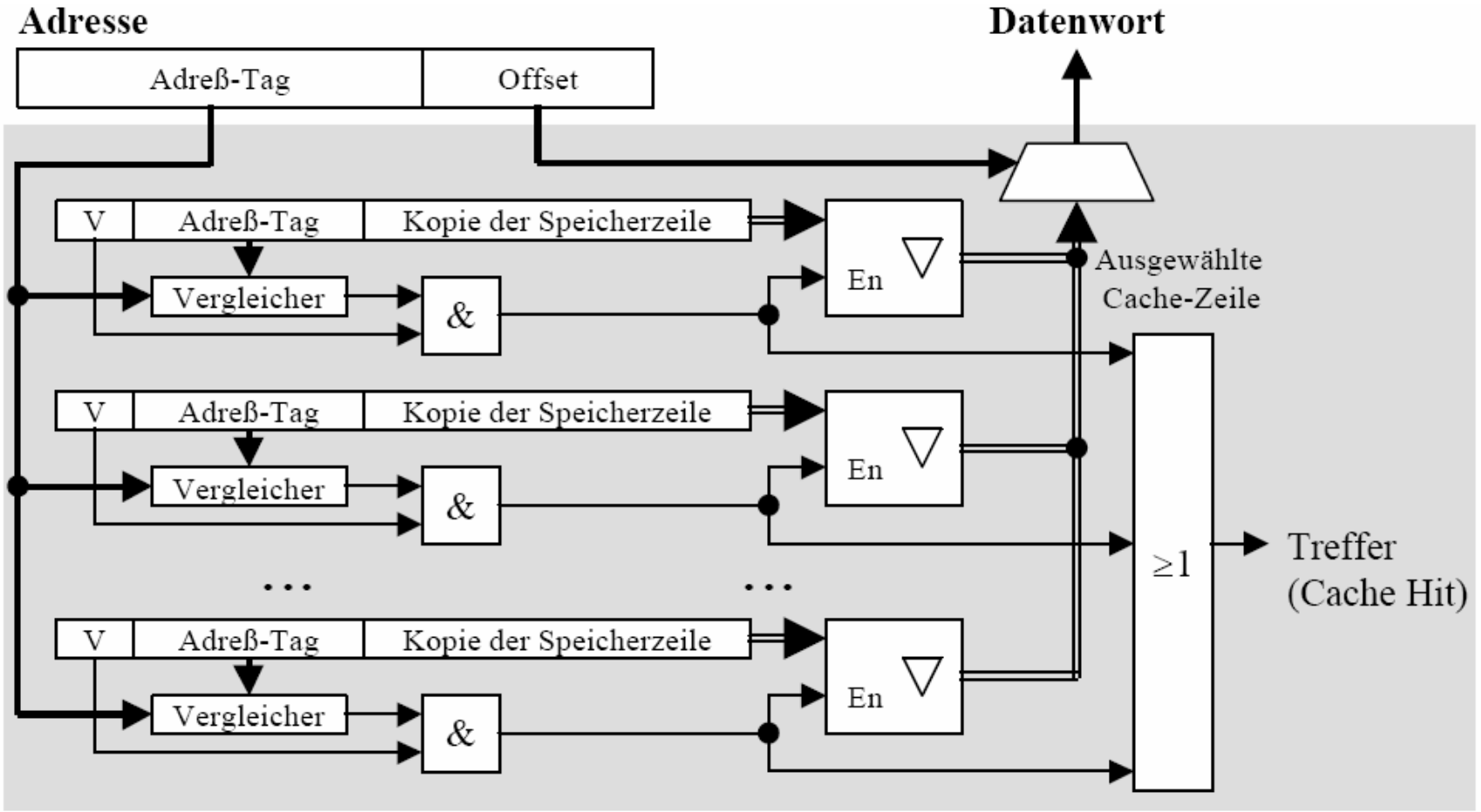
- Level-1: Cache für Instruktionen und Daten
- Level-2: vereint (unified) für Instruktionen und Daten
- Level-3: auf Karten-Ebene (**S**tatic **RAM**)

8.2 Cache-Organisation: Lesen

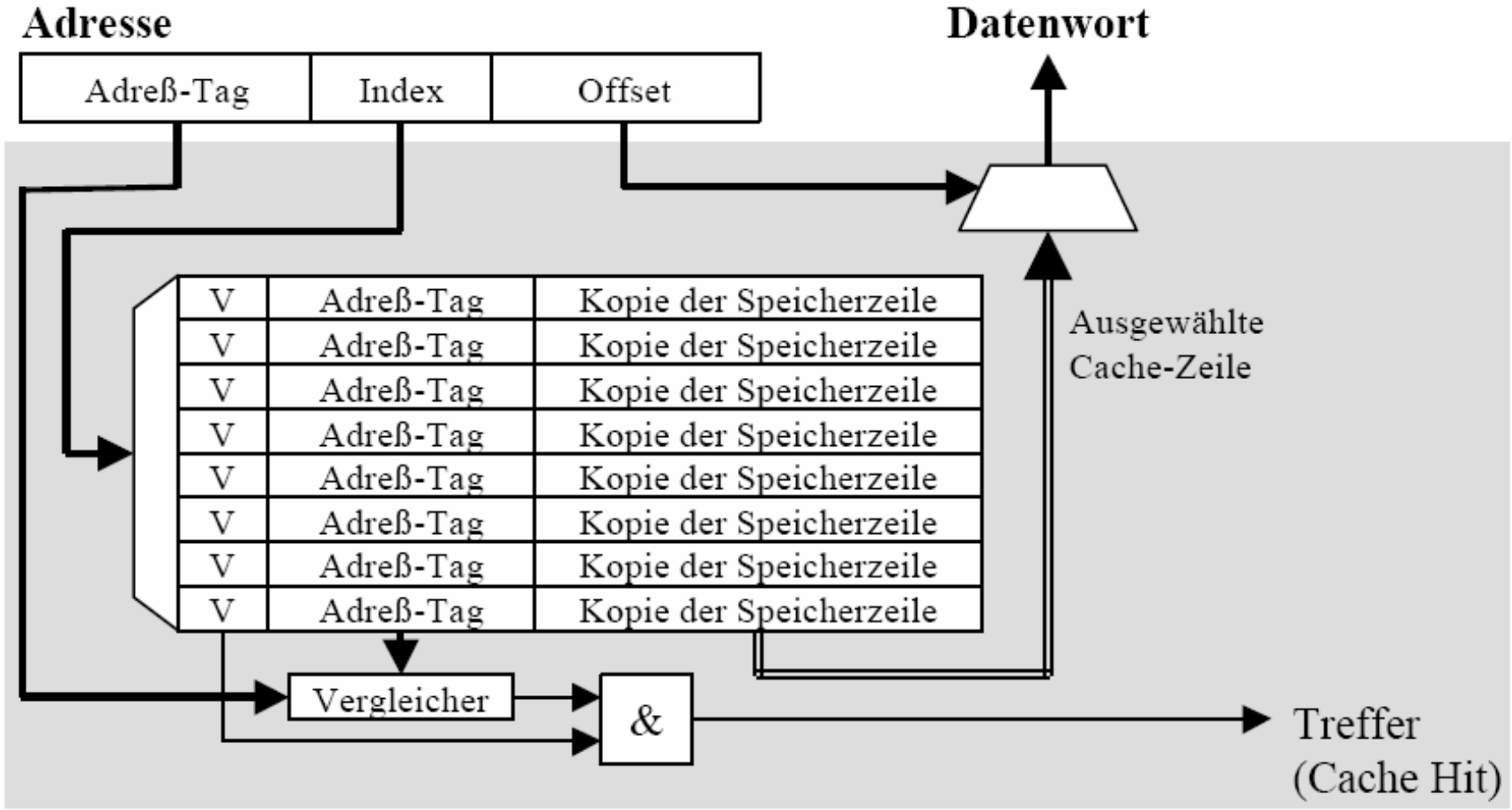


- **Voll assoziativ** (Fully associative): Ein Datenwort kann in einem beliebigen Cache-Eintrag abgelegt sein.
- **Direkt abgebildet** (Direct mapped): Ein Datenwort kann genau in einem Eintrag abgelegt sein.
- **Mengen-assoziativ** (Set associative): Ein Datenwort kann in wenigen Cache-Einträgen (typischerweise 2 bis 8) abgelegt sein.

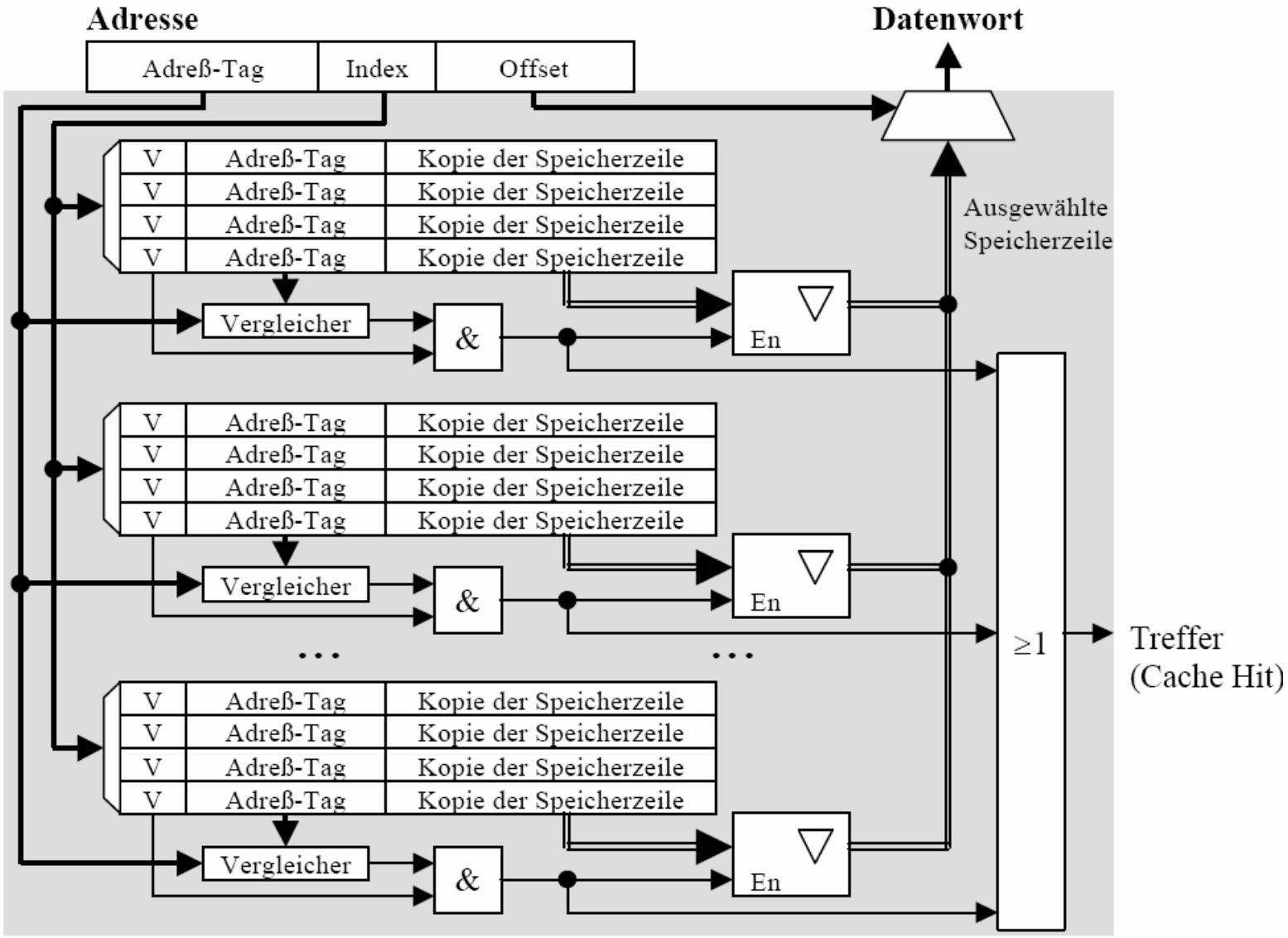
8.2 Voll assoziativer Cache



8.2 Direkt abbildender Cache



8.2 N-Wege mengenassoziativer Cache



8.3 Kenngrößen

- Liegt ein gesuchtes Wort im Cache, erhält man Treffer (hit).
  Trefferrate a in Prozent (hit rate).
- Liegt gesuchtes Wort nicht im Cache (miss), muss es aus dem Speicher gelesen werden:
  Fehlrate (1-a) in Prozent (miss rate).
- Mit Zugriffszeit t_c eines Wortes im Cache und der Zugriffszeit t_m eines Wortes im Hauptspeicher ist die
  mittlere Zugriffszeit t_a auf ein Wort:
  $t_a = a \cdot t_c + (1-a) \cdot t_m$
- Ist ein Wort nicht im Cache vorhanden, muss aus Speicher in den Cache geladen werden.
- Bei **Direkt abgebildetendem Cache** ist Zuordnung der Adresse des Wortes zu einer Cache-Zeile eindeutig.
- Beim **Assoziativen Cache** sind mehrere Strategien möglich:
  1. Wähle älteste Zeile aus
  2. Wähle Zeile, auf die am längsten nicht mehr zugegriffen wurde.
  3. Wähle zufällig eine Zeile aus.

  Alle drei Strategien haben weitgehend gleichen Trefferraten.

8.4 Schreiben von Daten

Beim Schreiben von Daten muss sichergestellt sein, dass Daten in Cache und Hauptspeicher konsistent bleiben.

Mehrer Strategien:

- **No-Write**: Wort nur in Hauptspeicher schreiben, nicht in Cache. Zugehörige Zeile im Cache als ungültig markieren.
  → langsam, Schreiben in HS, ggf. Lesen aus HS
- **Write-Through**: Cache- und Hauptspeicher beide schreiben. Cache-Zeile bleibt gültig. Übliches Verfahren.
  → langsame Schreibzugriffe.
- **Write-Back**: Nur Cache-Speicher schreiben, aber Cache-Zeile als modifiziert („Dirty“) markieren. Vor Überschreiben der Zeile muss der Inhalt in Hauptspeicher gesichert werden. Dirty-Bit ist der Cache-Zeile angefügt, wird beim Laden einer Zeile aus dem HS gelöscht, beim Schreiben eines Wortes in der Cache-Zeile gesetzt.

8.5 Lesen von Daten

Zwei prinzipielle Lese- und Füllstrategien

- **on demand, demand fetching**
  Beim ersten Lesen einer Information aus dem Hauptspeicher wird die gesamte Zeile auch in den Cache übertragen. Benötigte Blöcke, die noch nicht im Cache liegen, werden nachgeladen. Diese Strategie wird von jedem Cache verwendet
- **Prädiktiv, prefetch**
  Es wird versucht, vorherzusagen, welche Speicherzeilen zukünftig gebraucht werden und diese während Leerlaufphasen schon in den Cache vorgeladen.
  Ein zweiter Programmzähler (remote PC) im Cache wird dazu verwendet, den künftigen Programmverlauf vorherzubestimmen und entsprechende Blöcke vorzuladen.

9. Speichermanagement

1. Kopplung von CPU- und Speicheradresse
2. Virtuelle und physikalische Adressen

9.1 Kopplung CPU- und Speicheradresse

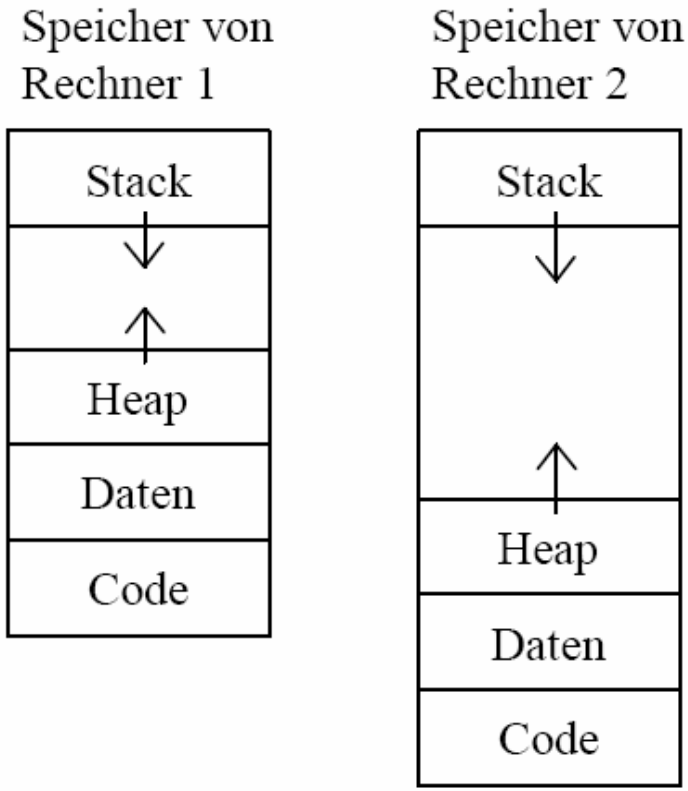
Informatik

Bisher sind wir davon ausgegangen, dass die intern vom Prozessor verwendete Adresse identisch mit der externen Speicheradresse ist.

Das aber hat **Nachteile**:

- Anpassung der Speicherbelegung eines Programms an die Rechnerkonfiguration notwendig.
- Bei Multitasking Zuweisung eines speziellen Speicherbereichs für jeden Task nötig.

9.1 Stack-, Heap-, Daten- und Code-Bereich

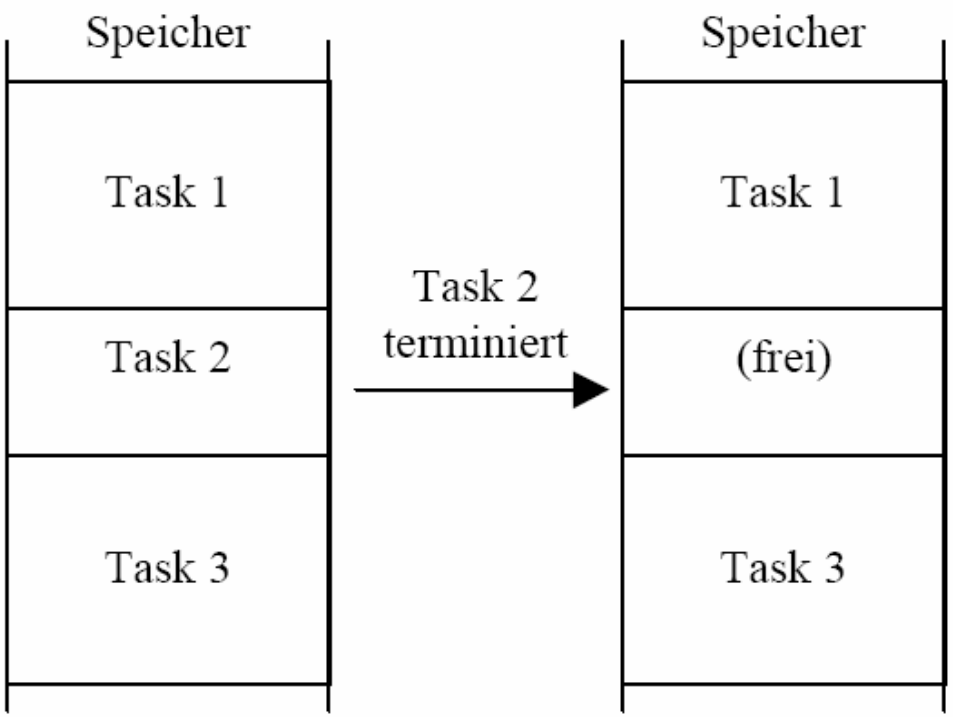


- Größe von Daten- und Code-Bereich wird beim Übersetzen vom Compiler ermittelt (feste Größe für ein Programm).
- Stack und Heap besitzen dynamische Größen, die vom Programmablauf abhängen.

(Stack: z.B. Lokale Variable, Heap: dynamische Objekte)

- Der freie Bereich zwischen Stack und Heap sollte immer möglichst groß gewählt werden um Speicherunterlauf zu vermeiden.
- Mögliche Größe hängt aber vom Rechner (Ressourcen) ab.

9.1 Speicherzuweisung bei Multitasking



Bei Multitasking muss beim Start einer Task ein freier Speicher-bereich gesucht und die Task an diesen Speicherbereich durch Linken angepasst werden.

Jede Task benötigt einen eigenen Ausschnitt aus dem gesamten Adressraum, der nicht von anderen Tasks verwendet werden darf.

Wenn ein Task terminiert, entsteht eine Lücke im Speicher.

Passt ein neuer Task nicht genau in diese Lücke, entstehen nach und nach ungenutzte Speicherfragmente.

9.1 Kopplung CPU- und Speicheradresse

Informatik

Eine feste Kopplung von CPU-interner Adresse und externer Speicheradresse **verhindert flexible und dynamische Nutzung des Hauptspeichers**.

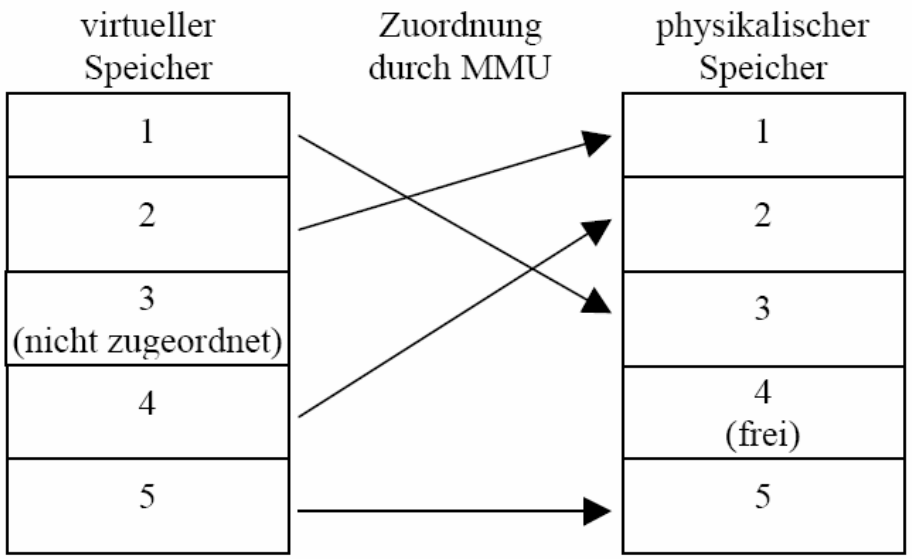
Dies ist nur bei Embedded Systemen akzeptabel.

In allen anderen Systemen, welche Flexibilität erfordern, ist die Entkopplung interner und externer Adressen notwendig, z.B. Mainframe, Workstations, PC's etc.

→ **Entkopplung** der CPU-internen (**virtuellen**) Adresse und der Speicheradresse (**physikalische** Adresse) schafft die notwendige Flexibilität.

Die Umsetzung erfolgt durch eine Hardwareeinheit, diese nennt man Memory Management Unit (MMU) oder Paging Unit

9.2 virtuelle und physikalische Adresse



Der virtuelle und der physikalische Speicherbereich wird in Speicherseiten fester Größe aufgeteilt → pages (Kacheln)

Übliche Seitengrößen sind 4 kBytes oder 8 kBytes (2er Potenz), aber auch deutlich größere Seiten werden verwendet.

Virtuelle Speicherseiten werden durch die MMU physikalischen Seiten zugeordnet.

Ist bei Zugriff auf eine virtuelle Seite keine physikalische zugeordnet, liegt ein **Seitenfehler** (page fault) vor. Die MMU erzeugt dann in der CPU einen Interrupt, die Ausnahmebehandlung lädt die fehlende Seite.