Grundlagen der Rechnerarchitektur und Betriebssysteme

Speicher

Florian Knoch · Lehrstuhl für Praktische Informatik,
insbes. Systemnahe Programmierung, Universität Bamberg

Wie läuft es bisher?

Gebt Feedback bis zum 10. Mai.

ZUM FORMULAR

Speicher-Grundlagen

Gigabyte vs. Gibibyte

SI- vs. IEC-Präfixe

Dezimalpräfixe

1 Byte	8 Bit
1 Kilobyte (KB)	10^3 Byte
1 Megabyte (MB)	10^6 Byte
1 Gigabyte (GB)	10^9 Byte
1 Terabyte (TB)	10^{12} Byte
1 Petabyte (PB)	10^{15} Byte

Binärpräfixe[1]

1 Byte	8 Bit
1 Kibibyte (KiB)	2^{10} Byte
1 Mebibyte (MiB)	2^{20} Byte
1 Gibibyte (GiB)	2^{30} Byte
1 Tebibyte (TiB)	2^{40} Byte
1 Pebibyte (PiB)	2^{50} Byte

[1] Das "bi" in Kibi-/Mebi-/...-byte steht für "binary".

Addressierung

Eine Sachaufgabe

AUFGABE 1A

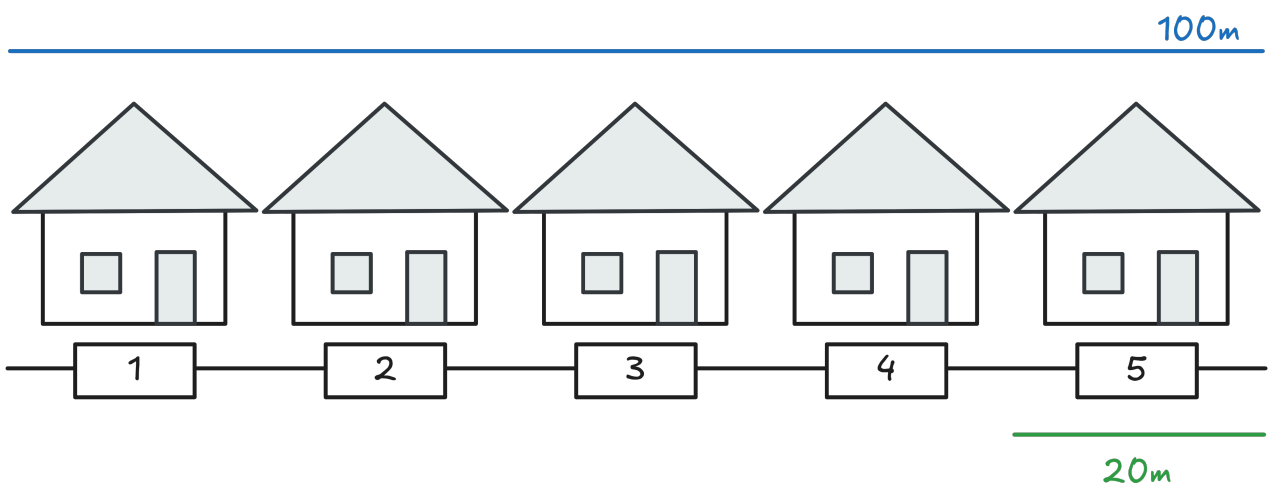
- **Straßenlänge**:
  100 m
- **Hausbreite**:
  20 m
- **Hausnummern**:
  …

Addressierung

Eine Sachaufgabe

AUFGABE 1A

- **Straßenlänge**:
  100 m
- **Hausbreite**:
  20 m
- **Hausnummern**:
  … 5 (1 Stelle breit)



Addressierung

Eine Sachaufgabe

AUFGABE 1B

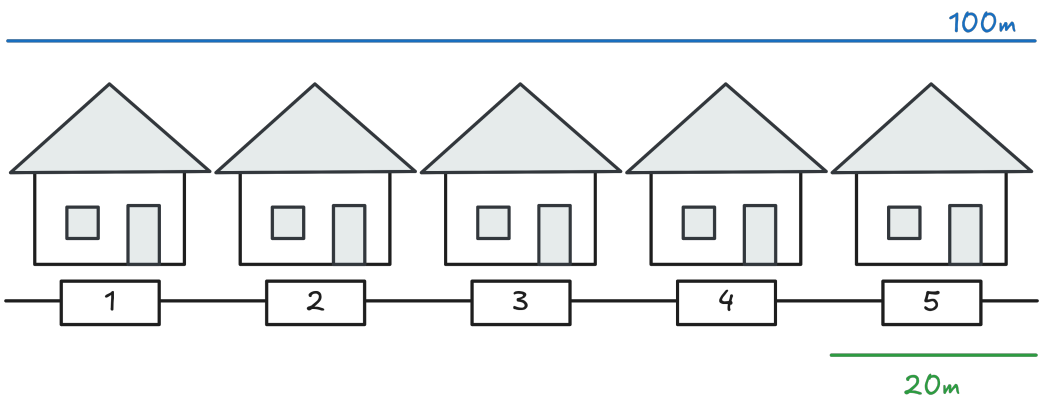
- **Straßenlänge**:
  100 m
- **Hausbreite**:
  25 m
- **Hausnummern**:
  …

Addressierung

Eine Sachaufgabe

AUFGABE 1B

- **Straßenlänge**:
  100 m
- **Hausbreite**:
  25 m
- **Hausnummern**:
  … 4 (1 Stelle breit)



Addressierung

Eine Sachaufgabe

AUFGABE 1C

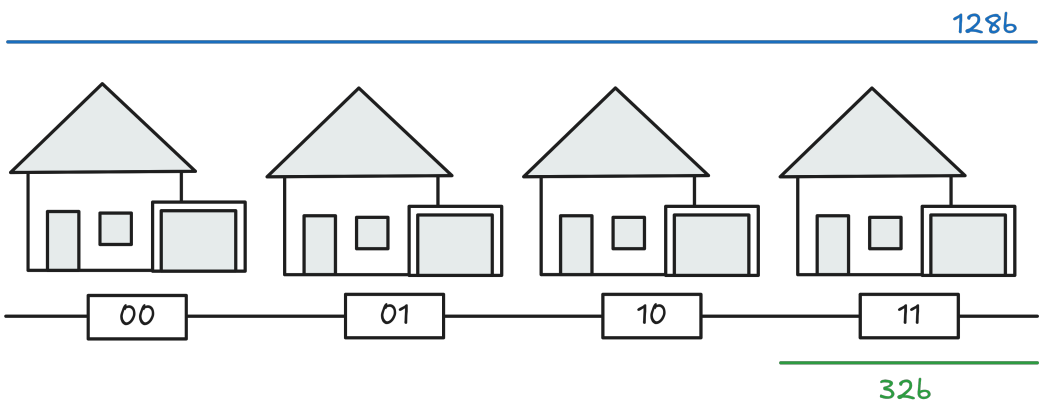
- **Speicher**:
  128 Bit
- **Wortgröße**:
  32 Bit
- **Adressen**:
  …

Addressierung

Eine Sachaufgabe

AUFGABE 1C

- **Speicher**:
  128 Bit
- **Wortgröße**:
  32 Bit
- **Adressen**:
  … 4 (2 Byte breit)

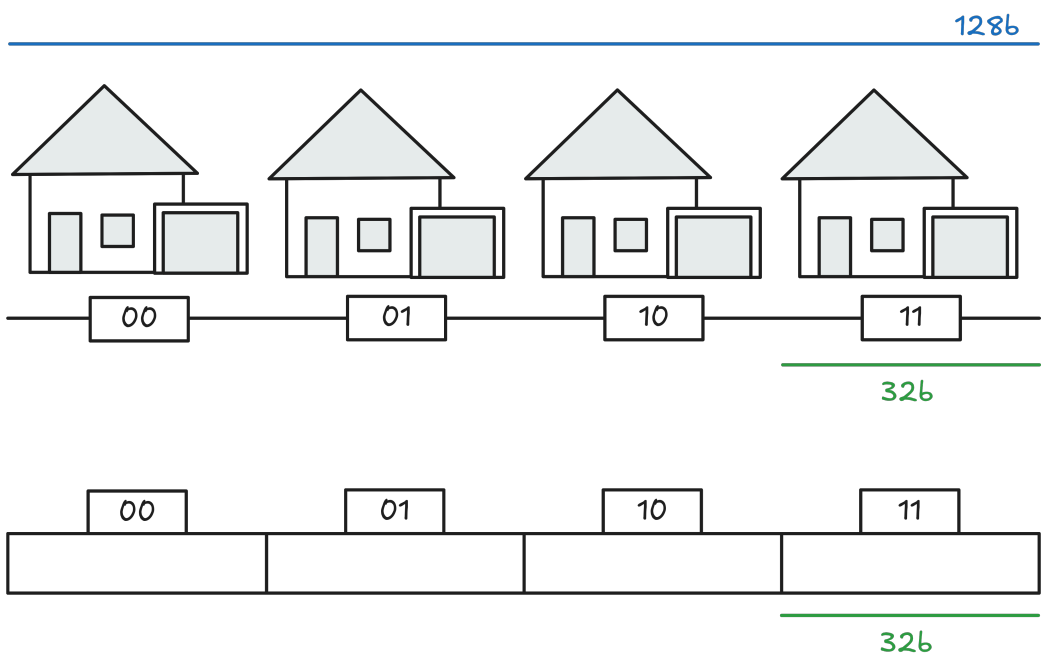


Addressierung

Eine Sachaufgabe

AUFGABE 1C

- **Speicher**:
  128 Bit
- **Wortgröße**:
  32 Bit
- **Adressen**:
  … 4 (2 Byte breit)



Geschicktere Konzepte zur Strukturierung und Adressierung von Speicher behandeln wir später in der Vorlesung zu **Virtuellem Speicher**.

WEITERFÜHREND

SYSNAP-OSE-B, SYSNAP-PMAP-M

Lokalität

AUFGABE 2A

Welche Rolle spielt Lokalität in diesem Codeschnipsel? [1, S. 959]

```
// C-Code
for (int i=0; i<8; i++) {
  for (int j=0; j<8000; j++) {
    A[i][j]=B[i][0]+A[j][i];
  }
}
```

Zeitliche Lokalität:

Räumliche Lokalität:

Lokalität

AUFGABE 2A

Welche Rolle spielt Lokalität in diesem Codeschnipsel? [1, S. 959]

```
// C-Code
for (int i=0; i<8; i++) {
  for (int j=0; j<8000; j++) {
    A[i][j]=B[i][0]+A[j][i];
  }
}
```

Zeitliche Lokalität: Was gerade verwendet wurde, wird bald wieder verwendet.

Räumliche Lokalität:

Lokalität

AUFGABE 2A

Welche Rolle spielt Lokalität in diesem Codeschnipsel? [1, S. 959]

```
// C-Code
for (int i=0; i<8; i++) {
  for (int j=0; j<8000; j++) {
    A[i][j]=B[i][0]+A[j][i];
  }
}
```

Zeitliche Lokalität: Was gerade verwendet wurde, wird bald wieder verwendet. Im Beispiel: `i`, `j` und `B[i][0]`

Räumliche Lokalität:

Lokalität

AUFGABE 2A

Welche Rolle spielt Lokalität in diesem Codeschnipsel? [1, S. 959]

```
// C-Code
for (int i=0; i<8; i++) {
  for (int j=0; j<8000; j++) {
    A[i][j]=B[i][0]+A[j][i];
  }
}
```

Zeitliche Lokalität: Was gerade verwendet wurde, wird bald wieder verwendet. Im Beispiel: `i`, `j` und `B[i][0]`

Räumliche Lokalität: Der nächste Zugriff erfolgt wahrscheinlich in der Nähe des aktuellen.

Lokalität

AUFGABE 2A

Welche Rolle spielt Lokalität in diesem Codeschnipsel? [1, S. 959]

```
// C-Code
for (int i=0; i<8; i++) {
  for (int j=0; j<8000; j++) {
    A[i][j]=B[i][0]+A[j][i];
  }
}
```

Zeitliche Lokalität: Was gerade verwendet wurde, wird bald wieder verwendet. Im Beispiel: `i`, `j` und `B[i][0]`

Räumliche Lokalität: Der nächste Zugriff erfolgt wahrscheinlich in der Nähe des aktuellen. Im Beispiel: `A[i][j]`

Lokalität

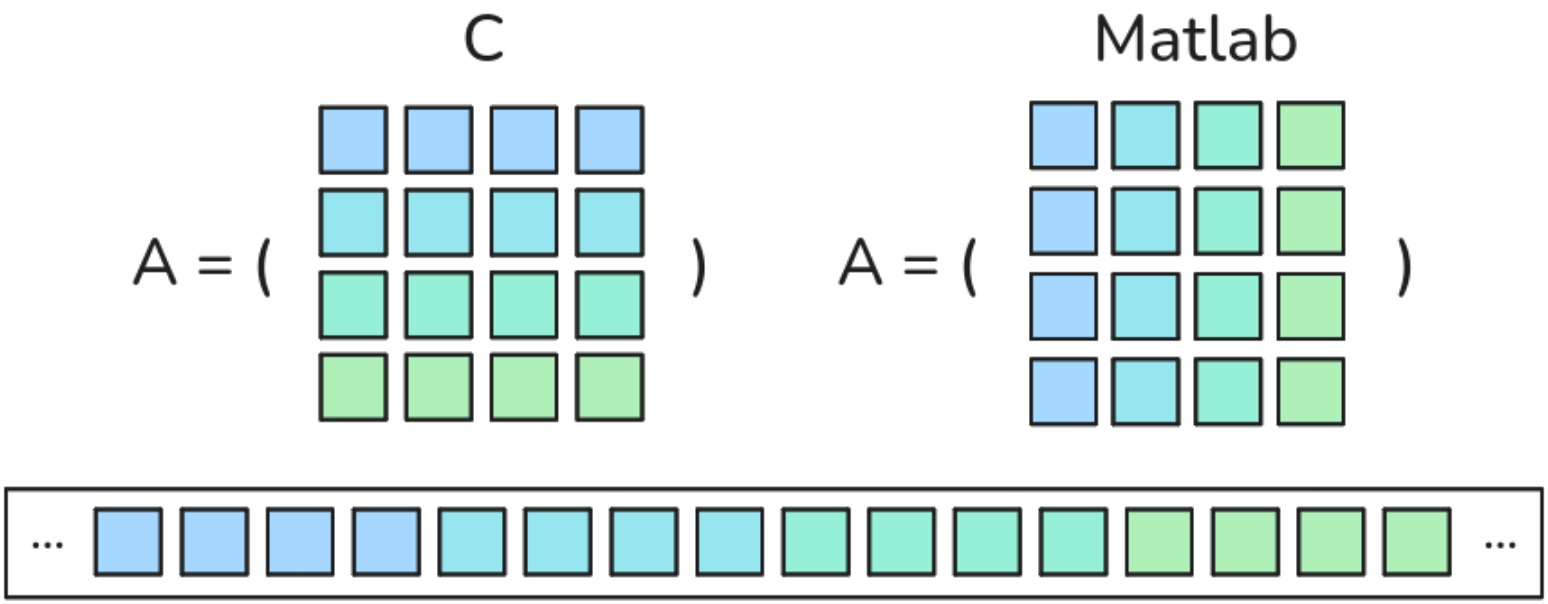


Abbildung 1: Speicherlayout für mehrdimensionale Arrays in C und Matlab

Lokalität

AUFGABE 2B

Was ändert sich, wenn stattdessen Matlab zum Einsatz kommt? [1, S. 959]

```
% Matlab-Code
for I=1:8
  for J=1:8000
    A(I,J)=B(I,0)+A(J,I);
  end
end
```

Zeitliche Lokalität: Was gerade verwendet wurde, wird bald wieder verwendet.

Räumliche Lokalität: Der nächste Zugriff erfolgt wahrscheinlich in der Nähe des aktuellen.

Lokalität

AUFGABE 2B

Was ändert sich, wenn stattdessen Matlab zum Einsatz kommt? [1, S. 959]

```
% Matlab-Code
for I=1:8
  for J=1:8000
    A(I,J)=B(I,0)+A(J,I);
  end
end
```

Zeitliche Lokalität: Was gerade verwendet wurde, wird bald wieder verwendet. Im Beispiel: `I`, `J` und `B[I][0]`

Räumliche Lokalität: Der nächste Zugriff erfolgt wahrscheinlich in der Nähe des aktuellen.

Lokalität

AUFGABE 2B

Was ändert sich, wenn stattdessen Matlab zum Einsatz kommt? [1, S. 959]

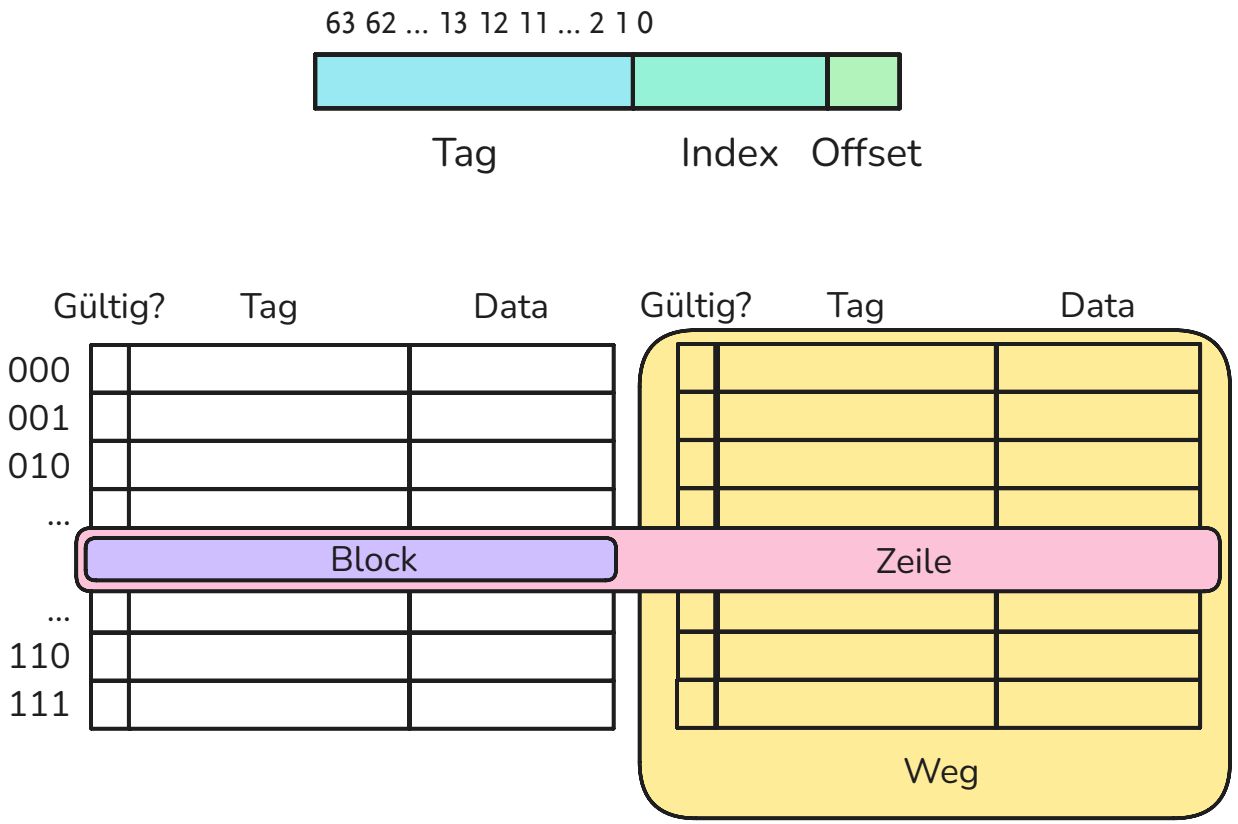
```
% Matlab-Code
for I=1:8
  for J=1:8000
    A(I,J)=B(I,0)+A(J,I);
  end
end
```

Zeitliche Lokalität: Was gerade verwendet wurde, wird bald wieder verwendet. Im Beispiel: `I`, `J` und `B[I][0]`

Räumliche Lokalität: Der nächste Zugriff erfolgt wahrscheinlich in der Nähe des aktuellen. Im Beispiel: `A[J][I]` und `B[I][0]`

Caches

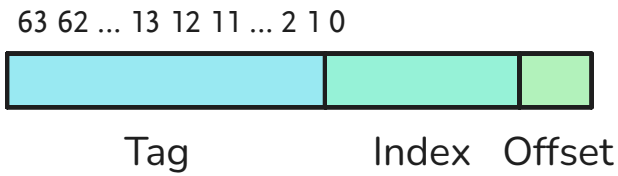
Grundbegriffe

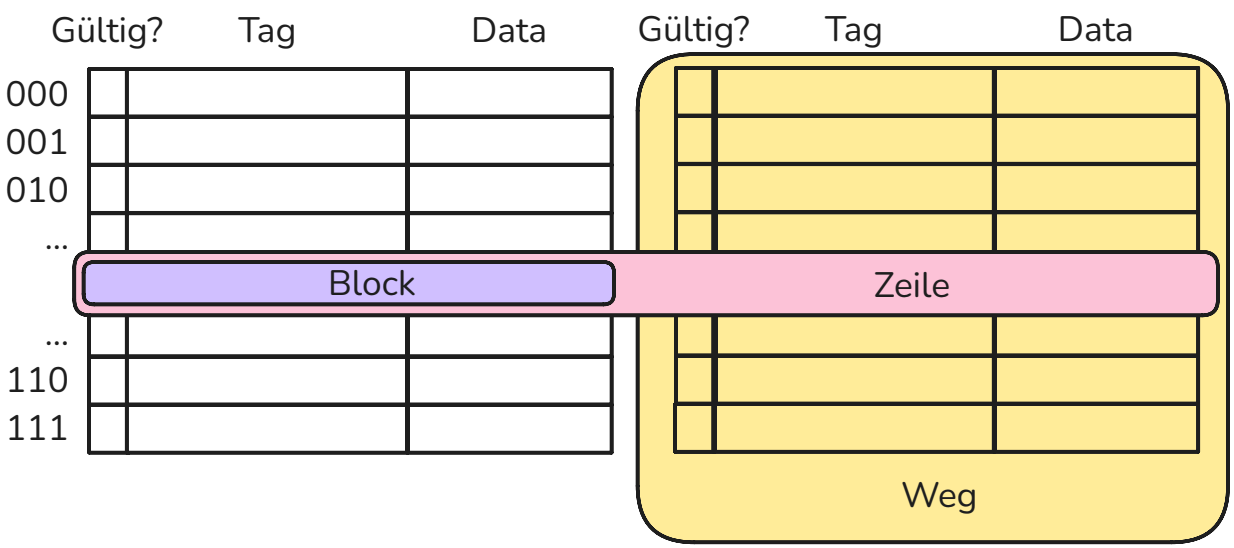


Cache-Anatomie

- **Zeile**: Sucheinheit im Cache, kann mehrere Blöcke beinhalten
- **Block**: Ergebniseinheit im Cache, Menge der Cache-Inhalte, die gleichzeitig zurückgegeben (*Hit*) bzw. ausgetauscht (*Miss*) werden
- **Weg**: „Spalte“ im Cache

Grundbegriffe





Adressen

- **Tag**: einzigartiger Bezeichner eines Blocks
- **Index**: Adresse einer Cache-Zeile
- **Offset**: Adresse eines Wortes innerhalb der Daten eines Blocks

Organisationsformen von Caches

Abbildung von Adressräumen

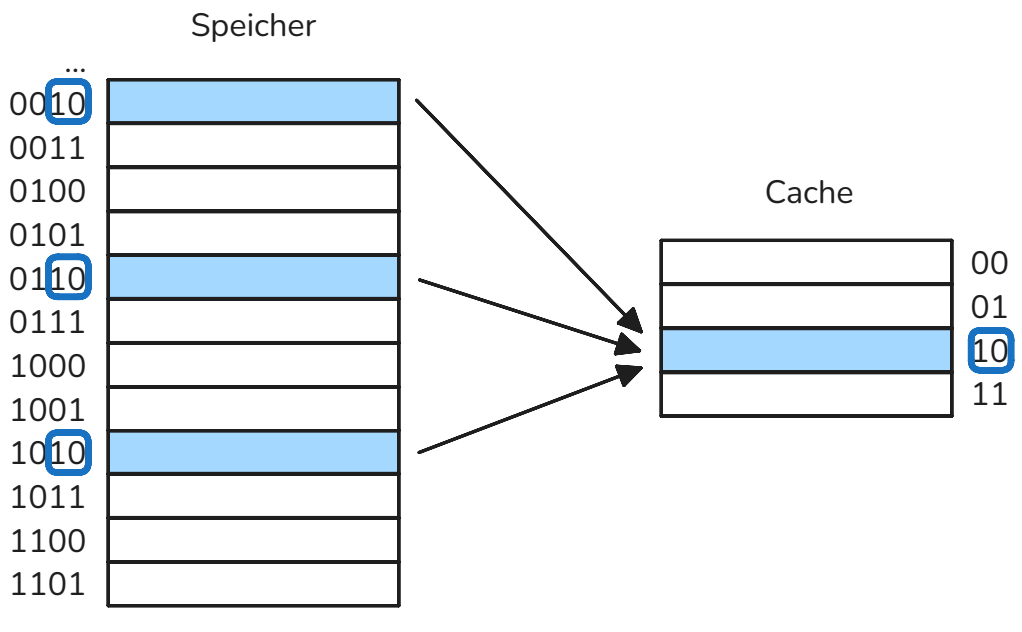


Abbildung 2: Einfache Abbildung von Adressen auf einen vier Zeilen großen Cache: `index = address % num_cache_lines`, angelehnt an Hennesy und Patterson [1, S. 762]

Direkte Abbildung

en. *Direct-Mapped Cache*

- zu jeder Speicheradresse gibt es nur genau **einen Eintrag** im Cache
- Vergleich eines Tags[2] ergibt ***Hit*** bzw. ***Miss***
- einfache, kostengünstige Schaltung
- führt zu häufigen **Verdrängungen** und niedriger ***Hit Rate***
- potenziell ungleichmäßige Auslastung des Caches

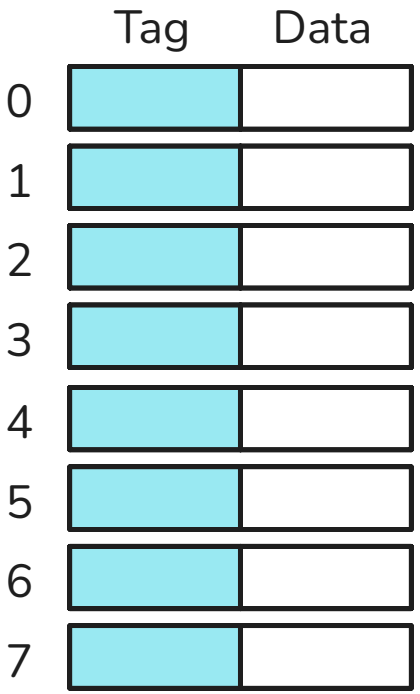


Abbildung 3: Direkte Abbildung mit 8 Blöcken, angelehnt an Hennesy und Patterson [1, S. 762]

[2] unter Zuhilfenahme des Gültigkeits-Bits

Mengenassoziative Abbildung

en. *Set-Associative Cache*

- eine Speicheradresse führt zu einer **Menge von Einträgen**
- Vergleich aller Tags[3] ergibt ***Hit*** bzw. ***Miss***

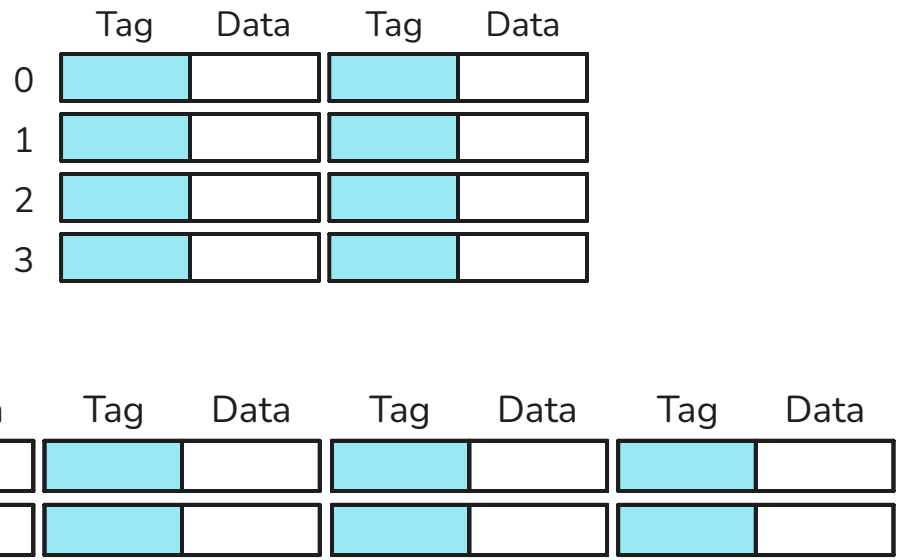


Abbildung 4: Mengenassoziative Abbildung mit acht Blöcken und zwei bzw. vier Wegen, angelehnt an Hennesy und Patterson [1, S. 762]

[3] unter Zuhilfenahme des Gültigkeits-Bits

Vollassoziative Abbildung

en. *Fully Associative Cache*

- jede Adresse wird auf **dieselbe Zeile** im Cache abgebildet
- benötigt entsprechend viel Platz und viele Bauteile zum Vergleich der Tags
- kommt in der Praxis kaum zum Einsatz

Abbildung 5: Vollassoziative Abbildung mit acht Blöcken und acht Wegen (entspricht einer mengenassoziativen Abbildung), angelehnt an Hennesy und Patterson [1, S. 762]

Fragen?

Literatur

[1] J. L. Hennesy und D. A. Patterson, *Computer Architecture: A Quantitative Approach*, 6. Aufl. Morgan Kaufmann Publishers, 2019.

Lizenzhinweise

Kursmaterialien · **Quelltext**

Diese Präsentation wurde zuletzt am 05.05.2026 bearbeitet. Sie basiert auf den Folien der zugehörigen Vorlesung von Prof. Dr. Michael Engel. Sofern nicht anders angegeben, sind die Inhalte unter der **Lizenz CC BY-SA 4.0** verfügbar. Das Universitätslogo sowie die Schriftart UB Scala Sans sind Eigentum der Otto-Friedrich-Universität Bamberg.

Folgende **Open-Source-Komponenten** kommen in der Präsentation zum Einsatz: **circuiteria** (Apache 2.0), **finite** (MIT), **fontawesome** (SIL), **pgf-tikz** (GPL 2.0), **typst** (Apache 2.0), **typst-ccicons** (MIT), **typst-polylux** (MIT).