School of Computation, Information and Technology
Technische Universität München

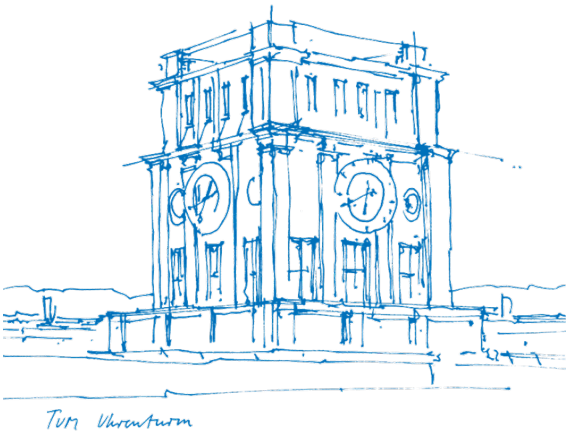
# Übung 05: Caches

## Einführung in die Rechnerarchitektur

**Niklas Ladurner**

School of Computation, Information and Technology
Technische Universität München

14. November 2025



Keine Garantie für die Richtigkeit der Tutorfolien.

Bei Unklarheiten/Unstimmigkeiten haben VL/ZÜ-Folien recht!

# Motivation

- Zugriffe auf Hauptspeicher ($\equiv$ RAM) sind **extrem** langsam. Lösung: Caches
- „Zwischenstation“ zwischen Registern (sehr schnell, sehr klein) und Hauptspeicher (sehr langsam, sehr groß)
- Idee: Häufig genutzte Daten im Cache zwischenspeichern, der Rest wird bei Bedarf aus dem Hauptspeicher geholt
- Die Position eines Datums im Cache wird durch die Adresse des Zugriffs bestimmt
- Heutzutage meist L1/L2/L3-Caches: Caches aufsteigender Größe, aber absteigender Zugriffszeit

# Terminologie

- **Hit**: Datum liegt im Cache
- **Miss**: Datum nicht im Cache, muss erst aus Hauptspeicher geholt werden
  $\rightarrow$ Ziel: möglichst hohe **Hitrate** (Hits/Anfragen), d.h. häufig genutzte Daten im Cache

# Terminologie

- **Hit**: Datum liegt im Cache
- **Miss**: Datum nicht im Cache, muss erst aus Hauptspeicher geholt werden
  $\rightarrow$ Ziel: möglichst hohe **Hitrate** (Hits/Anfragen), d.h. häufig genutzte Daten im Cache

- **zeitliche** Lokalität[1]: Zugriff auf $x \rightarrow$ Zugriff auf $x$ in Zukunft wahrscheinlicher
- **räumliche** Lokalität: Zugriff auf $x \rightarrow$ Zugriff auf Daten in der Nähe wahrscheinlicher
  $\rightarrow$ Konsequenz: Cachen von ganzer Cachezeile

[1] Definition der Lokalitätsprinzipien ist eine beliebte Klausuraufgabe!

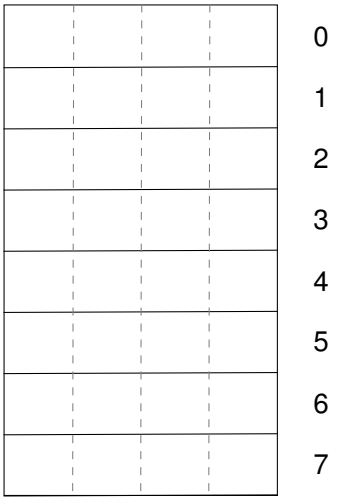
# Direktabbildender Cache[2]

- direkte (surjektive) Abbildung von Hauptspeicheradresse $\mapsto$ Cacheadresse
- jedes Datum aus dem Hauptspeicher kann nur in *eine* bestimmte Cachezeile gecached werden

[2] Englisch: Direct Mapped Cache

# Direktabbildender Cache[2]

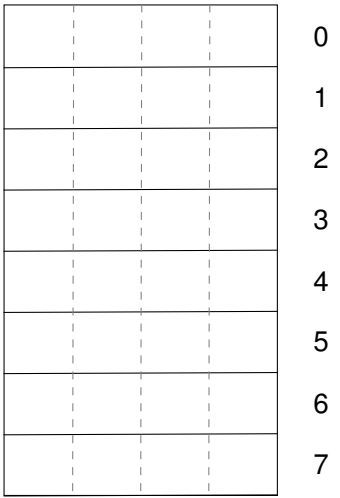
- direkte (surjektive) Abbildung von Hauptspeicheradresse $\mapsto$ Cacheadresse
- jedes Datum aus dem Hauptspeicher kann nur in *eine* bestimmte Cachezeile gecached werden
- Beispiel: 16-Bit-Architektur, 8 Cachezeilen zu je 4 Byte
  1. Anzahl Offset-Bits:
  2. Anzahl Index-Bits:
  3. Anzahl Tag-Bits:




[2] Englisch: Direct Mapped Cache

# Direktabbildender Cache[2]

- direkte (surjektive) Abbildung von Hauptspeicheradresse $\mapsto$ Cacheadresse
- jedes Datum aus dem Hauptspeicher kann nur in *eine* bestimmte Cachezeile gecached werden
- Beispiel: 16-Bit-Architektur, 8 Cachezeilen zu je 4 Byte
  1. Anzahl Offset-Bits: $\log_2(4) = 2$
  2. Anzahl Index-Bits: $\log_2(8) = 3$
  3. Anzahl Tag-Bits: $16 - 3 - 2 = 11$

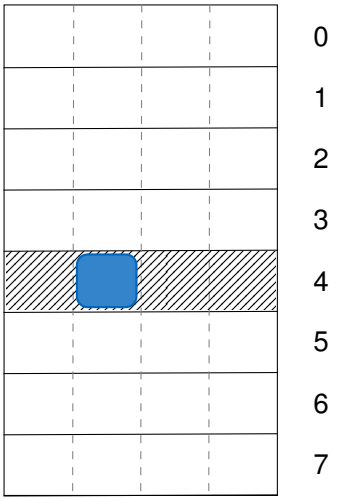



[2] Englisch: Direct Mapped Cache

# Direktabbildender Cache[2]

- 0x6291 = 01100010100 | 100 | 01

| Tag | Index | Offset |
|---|---|---|
| 11 bit | 3 bit | 2 bit |



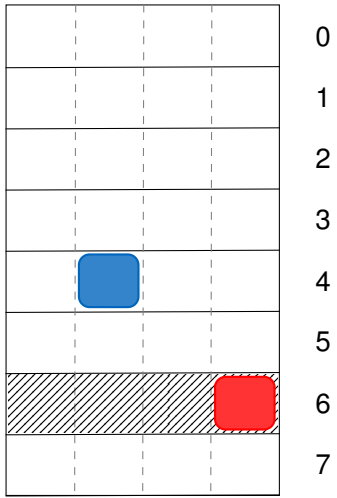

Es wird immer die gesamte Cachezeile ausgetauscht!

[2] Englisch: Direct Mapped Cache

# Direktabbildender Cache[2]

- $0x6291 =$ | Tag: 01100010100 (11 bit) | Index: 100 (3 bit) | Offset: 01 (2 bit)
- $0xccdb =$ | Tag: 11001100110 (11 bit) | Index: 110 (3 bit) | Offset: 11 (2 bit)

Es wird immer die gesamte Cachezeile ausgetauscht!




[2] Englisch: Direct Mapped Cache

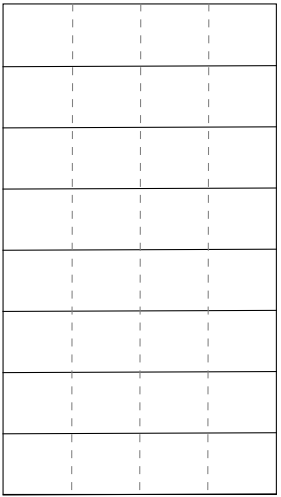
# Vollassoziativer Cache[3]

- Daten werden „wo gerade Platz ist“ gecached
- jedes Datum aus dem Hauptspeicher kann in eine *beliebige* Cachezeile gecached werden

[3] Englisch: Fully Associative Cache

# Vollassoziativer Cache[3]

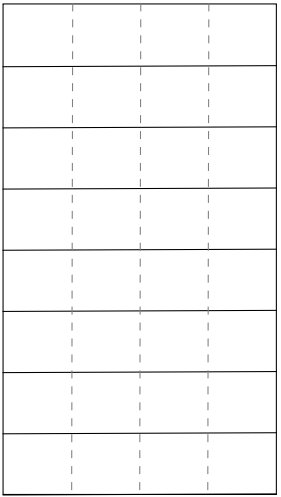
- Daten werden „wo gerade Platz ist“ gecached
- jedes Datum aus dem Hauptspeicher kann in eine *beliebige* Cachezeile gecached werden
- Beispiel: 16-Bit-Architektur, 8 Cachezeilen zu je 4 Byte
  1. Anzahl Offset-Bits:
  2. Anzahl Index-Bits:
  3. Anzahl Tag-Bits:



[3] Englisch: Fully Associative Cache

# Vollassoziativer Cache[3]

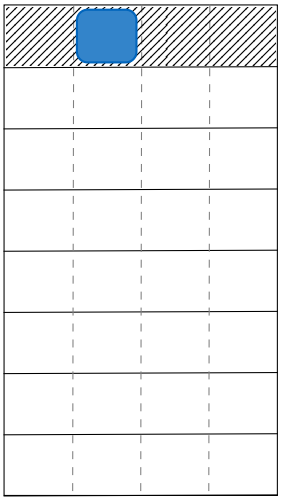
- Daten werden „wo gerade Platz ist“ gecached
- jedes Datum aus dem Hauptspeicher kann in eine *beliebige* Cachezeile gecached werden
- Beispiel: 16-Bit-Architektur, 8 Cachezeilen zu je 4 Byte
  1. Anzahl Offset-Bits: $\log_2(4) = 2$
  2. Anzahl Index-Bits: $0$
  3. Anzahl Tag-Bits: $16 - 2 - 0 = 14$



[3] Englisch: Fully Associative Cache

# Vollassoziativer Cache[3]

- $0x6291 = \underbrace{\boxed{01100010100100}}_{\text{14 bit}}\ \underbrace{\boxed{01}}_{\text{2 bit}}$ (Tag | Offset)



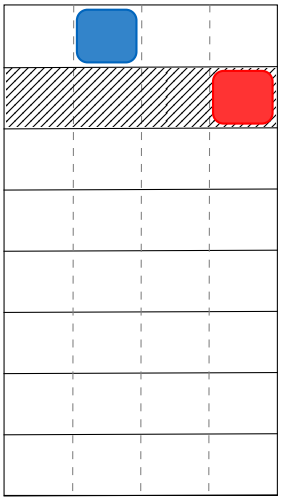
Es wird immer die gesamte Cachezeile ausgetauscht!

[3] Englisch: Fully Associative Cache

# Vollassoziativer Cache[3]

- $0x6291 = \underbrace{\boxed{01100010100100}}_{\text{Tag, 14 bit}} \; \underbrace{\boxed{01}}_{\text{Offset, 2 bit}}$
- $0xccdb = \underbrace{\boxed{11001100110110}}_{\text{Tag, 14 bit}} \; \underbrace{\boxed{11}}_{\text{Offset, 2 bit}}$

Es wird immer die gesamte Cachezeile ausgetauscht!



[3] Englisch: Fully Associative Cache

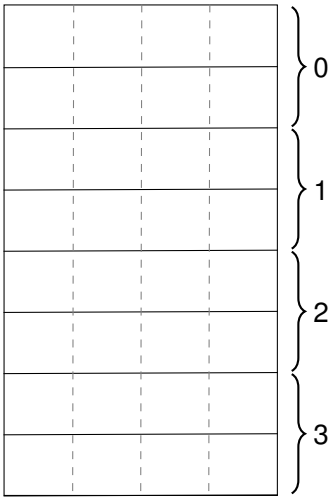
# Mengenassoziativer Cache[4]

- Aufteilung des Cache in Sets zu je $n$ Cachezeilen
- jedes Datum aus dem Hauptspeicher kann nur in *einem* bestimmtem Set gecached werden, im Set aber in einer *beliebigen* Cachezeile stehen

[4] Englisch: Set Associative Cache

# Mengenassoziativer Cache[4]

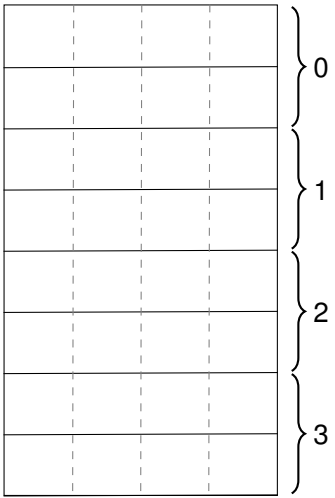
- Aufteilung des Cache in Sets zu je $n$ Cachezeilen
- jedes Datum aus dem Hauptspeicher kann nur in *einem* bestimmtem Set gecached werden, im Set aber in einer *beliebigen* Cachezeile stehen
- Beispiel: 16-Bit-Architektur, 8 Cachezeilen zu je 4 Byte, 2-fach assoziativ
  1. Anzahl Cache-Sets:
  2. Anzahl Offset-Bits:
  3. Anzahl Index-Bits:
  4. Anzahl Tag-Bits:




[4] Englisch: Set Associative Cache

# Mengenassoziativer Cache[4]

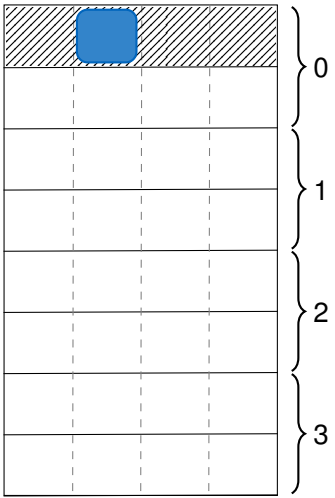
- Aufteilung des Cache in Sets zu je $n$ Cachezeilen
- jedes Datum aus dem Hauptspeicher kann nur in *einem* bestimmtem Set gecached werden, im Set aber in einer *beliebigen* Cachezeile stehen
- Beispiel: 16-Bit-Architektur, 8 Cachezeilen zu je 4 Byte, 2-fach assoziativ
  1. Anzahl Cache-Sets: $\frac{8}{2} = 4$
  2. Anzahl Offset-Bits: $\log_2(4) = 2$
  3. Anzahl Index-Bits: $\log_2(4) = 2$
  4. Anzahl Tag-Bits: $16 - 2 - 2 = 12$




[4] Englisch: Set Associative Cache

# Mengenassoziativer Cache[4]

- $0\text{x}6291 = \underbrace{\boxed{011000101001}}_{\text{Tag: 12 bit}} \underbrace{\boxed{00}}_{\text{Index: 2 bit}} \underbrace{\boxed{01}}_{\text{Offset: 2 bit}}$




Es wird immer die gesamte Cachezeile ausgetauscht!

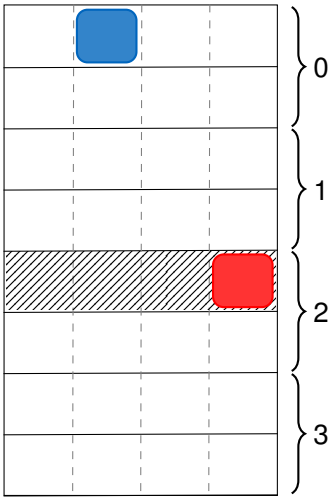
[4] Englisch: Set Associative Cache

# Mengenassoziativer Cache[4]

| | Tag | Index | Offset |
|---|---|---|---|
| ● 0x6291 = | 011000101001 | 00 | 01 |
| | 12 bit | 2 bit | 2 bit |

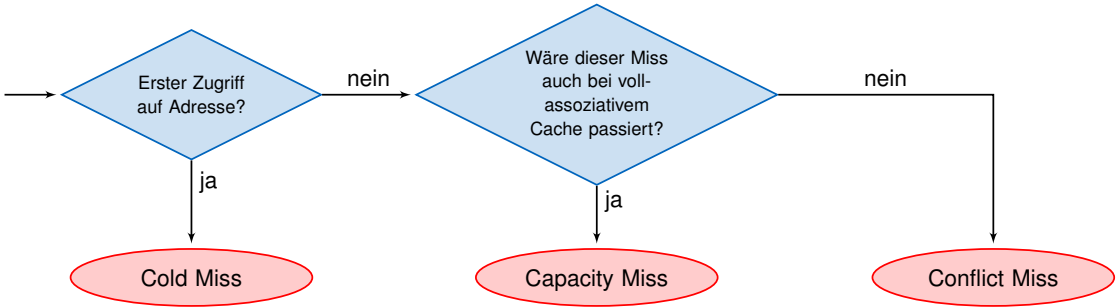
| | Tag | Index | Offset |
|---|---|---|---|
| ● 0xccdb = | 110011001101 | 10 | 11 |
| | 12 bit | 2 bit | 2 bit |

Es wird immer die gesamte Cachezeile ausgetauscht!




[4] Englisch: Set Associative Cache

# Klassifikation von Misses




Nach Gschoßmann et al., 2023

Fragen?

# Links

- Zulip: „ERA Tutorium – Mi-1600-3“ bzw. „ERA Tutorium – Fr-1500-1“
- ERA-Moodle-Kurs
- ERA-Artemis-Kurs
- Wikipedia zu Caches
- Elektronik-Kompendium zu Caches

School of Computation, Information and Technology
Technische Universität München

# Übung 05: Caches

## Einführung in die Rechnerarchitektur

**Niklas Ladurner**

School of Computation, Information and Technology
Technische Universität München

14. November 2025