

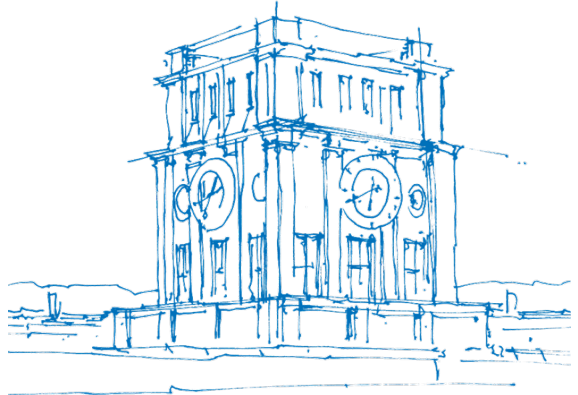
# Übung 05: Caches

## Einführung in die Rechnerarchitektur

**Niklas Ladurner**

School of Computation, Information and Technology  
Technische Universität München

14. November 2025



*TUM Uhrenturm*

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

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

- **Hit**: Datum liegt im Cache
- **Miss**: Datum nicht im Cache, muss erst aus Hauptspeicher geholt werden  
→ Ziel: möglichst hohe **Hitrate** (Hits/Anfragen), d.h. häufig genutzte Daten im Cache
  
- **zeitliche** Lokalität<sup>1</sup>: Zugriff auf  $x \rightarrow$  Zugriff auf  $x$  in Zukunft wahrscheinlicher
- **räumliche** Lokalität: Zugriff auf  $x \rightarrow$  Zugriff auf Daten in der Nähe wahrscheinlicher  
→ Konsequenz: Cachen von ganzer Cacheline

---

<sup>1</sup> Definition der Lokalitätsprinzipien ist eine beliebte Klausuraufgabe!

## Direktabbildender Cache<sup>2</sup>

- direkte (surjektive) Abbildung von Hauptspeicheradresse  $\mapsto$  Cacheadresse
- jedes Datum aus dem Hauptspeicher kann nur an *einer* bestimmten Stelle im Cache gecached werden

---

<sup>2</sup>Englisch: Direct Mapped Cache

# Direktabbildender Cache<sup>2</sup>

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

Index	
0	
1	
2	
3	
4	
5	
6	
7	

---

<sup>2</sup>Englisch: Direct Mapped Cache

## Direktabbildender Cache<sup>2</sup>

- direkte (surjektive) Abbildung von Hauptspeicheradresse  $\mapsto$  Cacheadresse
- jedes Datum aus dem Hauptspeicher kann nur an *einer* bestimmten Stelle im Cache 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:  $16 - 3 - 2 = 11$

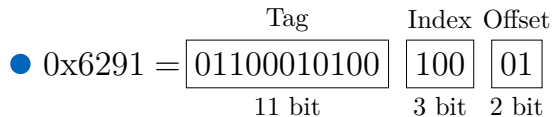
				Index
				0
				1
				2
				3
				4
				5
				6
				7

---

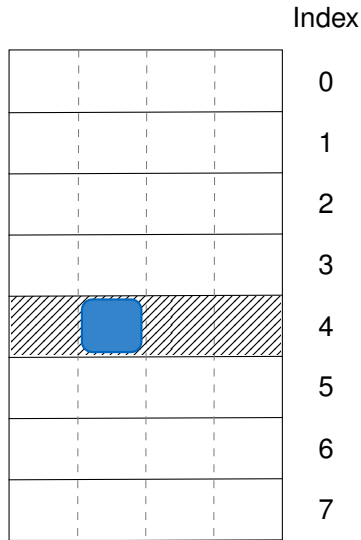
<sup>2</sup>Englisch: Direct Mapped Cache



# Direktabbildender Cache<sup>2</sup>



Es wird immer die gesamte Cacheline ausgetauscht!



<sup>2</sup>Englisch: Direct Mapped Cache

# Direktabbildender Cache<sup>2</sup>

● 0x6291 = 

Tag
01100010100

Index
100

Offset
01

  
11 bit      3 bit    2 bit

● 0xccdb = 

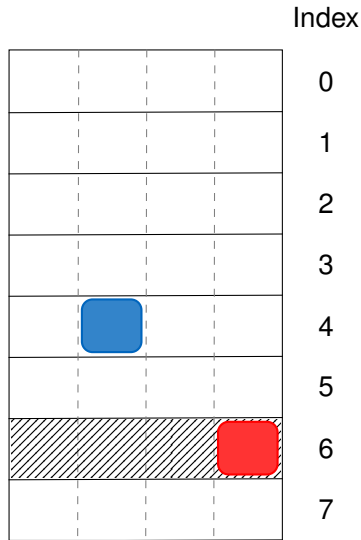
Tag
11001100110

Index
110

Offset
11

  
11 bit      3 bit    2 bit

Es wird immer die gesamte Cacheline ausgetauscht!



<sup>2</sup>Englisch: Direct Mapped Cache

# Vollassoziativer Cache<sup>3</sup>

- Daten werden „wo gerade Platz ist“ gecached
- jedes Datum aus dem Hauptspeicher kann an einer *beliebigen* Stelle im Cache gecached werden

---

<sup>3</sup>Englisch: Fully Associative Cache

# Vollassoziativer Cache<sup>3</sup>

- Daten werden „wo gerade Platz ist“ gecached
- jedes Datum aus dem Hauptspeicher kann an einer *beliebigen* Stelle im Cache gecached werden
- Beispiel: 16-Bit-Architektur, 8 Cachezeilen zu je 4 Byte
  1. Anzahl Offset-Bits:
  2. Anzahl Index-Bits:
  3. Anzahl Tag-Bits:


---

<sup>3</sup>Englisch: Fully Associative Cache

# Vollassoziativer Cache<sup>3</sup>

- Daten werden „wo gerade Platz ist“ gecached
- jedes Datum aus dem Hauptspeicher kann an einer *beliebigen* Stelle im Cache 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:  $16 - 3 - 0 = 13$


---

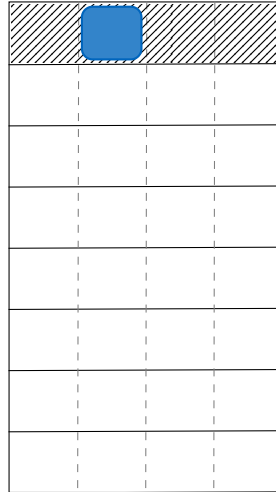
<sup>3</sup>Englisch: Fully Associative Cache

# Vollassoziativer Cache<sup>3</sup>

●  $0x6291 =$ 

Tag	Offset
01100010100100	01
13 bit	2 bit

Es wird immer die gesamte Cacheline ausgetauscht!



<sup>3</sup>Englisch: Fully Associative Cache

# Vollassoziativer Cache<sup>3</sup>

●  $0x6291 =$ 

Tag
01100010100100
13 bit

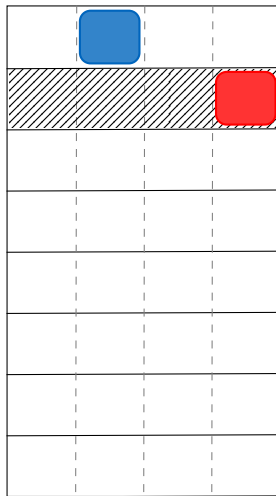
Offset
01
2 bit

●  $0xccdb =$ 

Tag
11001100110110
13 bit

Offset
11
2 bit

Es wird immer die gesamte Cacheline ausgetauscht!



<sup>3</sup>Englisch: Fully Associative Cache

## Mengenassoziativer Cache<sup>4</sup>

- Aufteilung des Cache in Sets zu je  $n$  Cachezeilen
- jedes Datum aus dem Hauptspeicher kann nur in *einem* bestimmtem Set gecached werden, innerhalb des Sets aber an *beliebiger* Stelle stehen

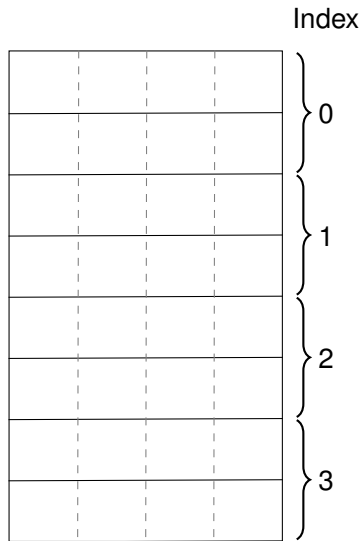
---

<sup>4</sup>Englisch: Set Associative Cache



# Mengenassoziativer Cache<sup>4</sup>

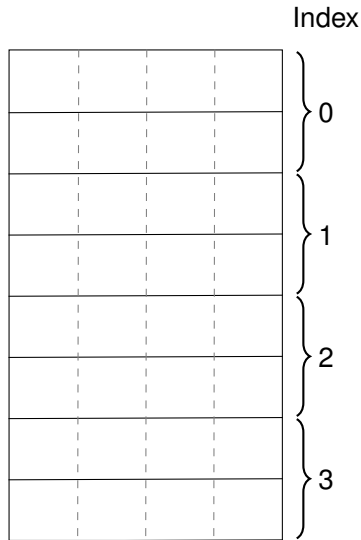
- Aufteilung des Cache in Sets zu je  $n$  Cachezeilen
- jedes Datum aus dem Hauptspeicher kann nur in *einem* bestimmtem Set gecached werden, innerhalb des Sets aber an *beliebiger* Stelle stehen
- Beispiel: 16-Bit-Architektur, 8 Cachezeilen zu je 4 Byte, 2-fach assoziativ
  1. Anzahl Cache-Sets:
 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



<sup>4</sup>Englisch: Set Associative Cache

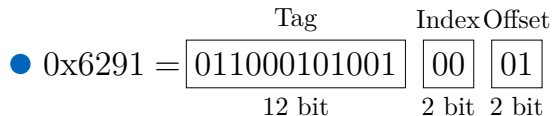
## Mengenassoziativer Cache<sup>4</sup>

- Aufteilung des Cache in Sets zu je  $n$  Cachezeilen
- jedes Datum aus dem Hauptspeicher kann nur in *einem* bestimmtem Set gecached werden, innerhalb des Sets aber an *beliebiger* Stelle 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:  $16 - 2 - 2 = 12$

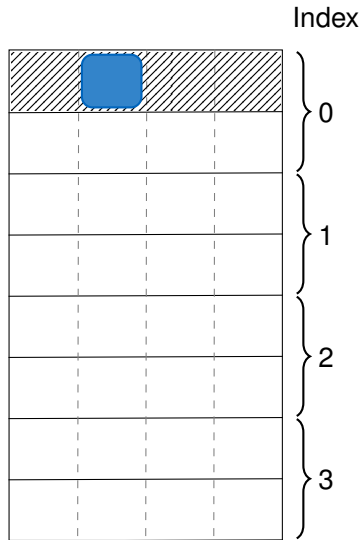


<sup>4</sup>Englisch: Set Associative Cache

# Mengenassoziativer Cache<sup>4</sup>



Es wird immer die gesamte Cacheline ausgetauscht!



<sup>4</sup>Englisch: Set Associative Cache

# Mengenassoziativer Cache<sup>4</sup>

● 0x6291 = 

Tag
011000101001

Index
00

Offset
01

  
12 bit      2 bit   2 bit

● 0xccdb = 

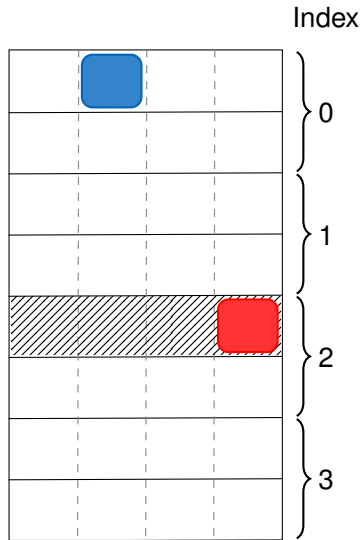
Tag
110011001101

Index
10

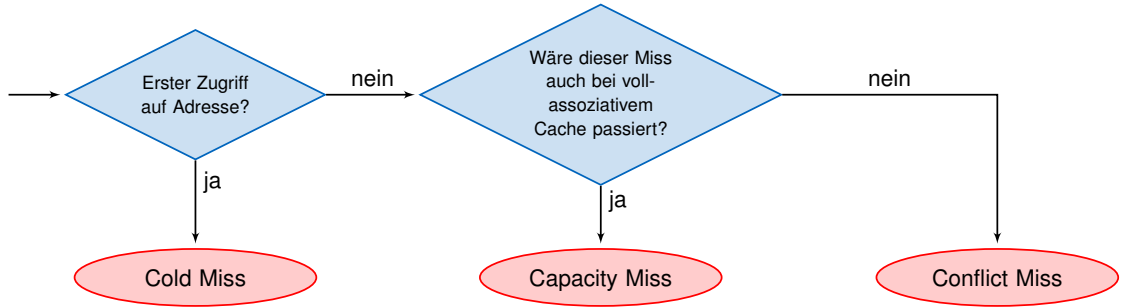
Offset
11

  
12 bit      2 bit   2 bit

Es wird immer die gesamte Cacheline ausgetauscht!



<sup>4</sup>Englisch: Set Associative Cache



Nach Gschoßmann et al., 2023

Fragen?

- 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

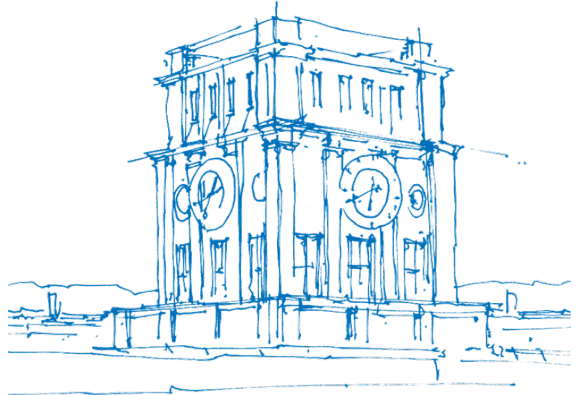
# Übung 05: Caches

## Einführung in die Rechnerarchitektur

**Niklas Ladurner**

School of Computation, Information and Technology  
Technische Universität München

14. November 2025



*TUM Uhrenturm*