

Eigenschaften des Speichers

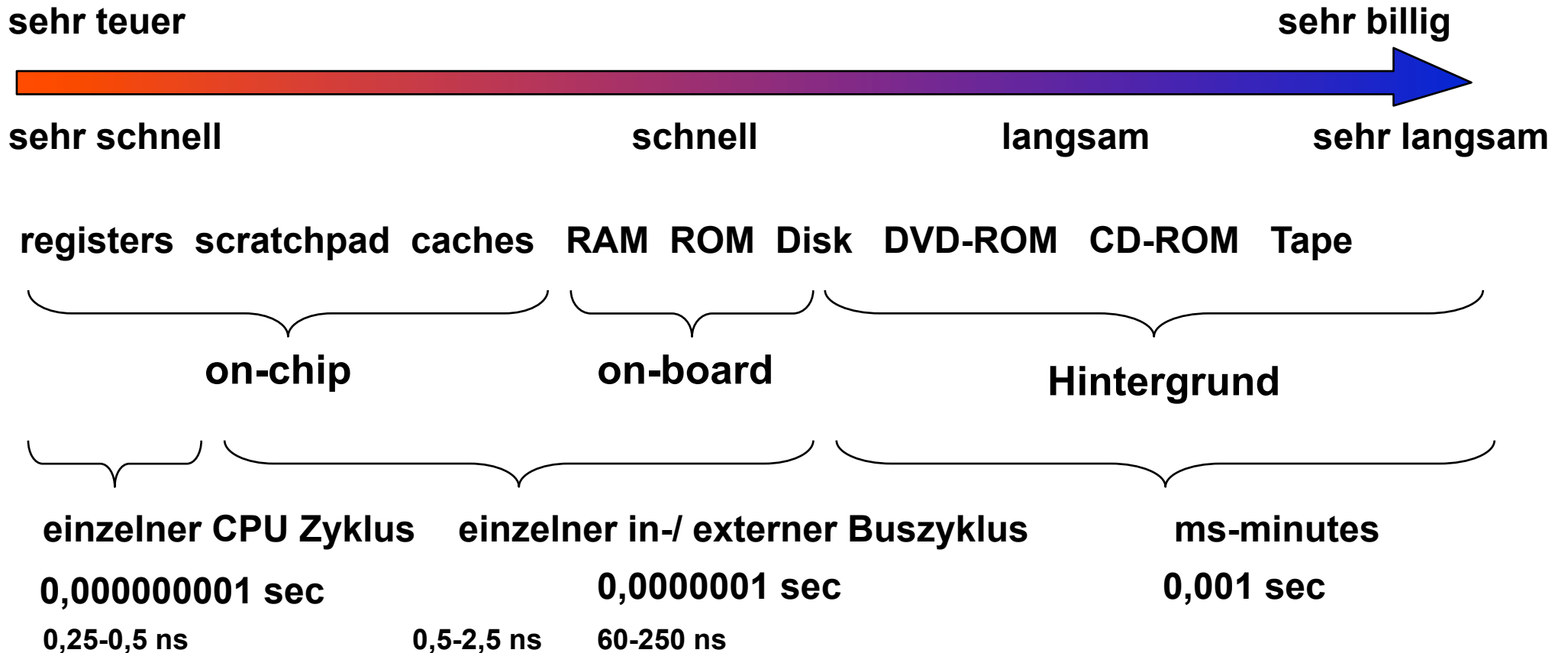
- unendlich groß,
- unendlich schnell,
- unendlich billig,
- nichtflüchtig,
- kein Effekt durch konkurrierende Zugriffe,
- Schutz vor fehlerhaften Zugriffen.

Die Speicherverwaltung hat die Approximierung dieser Eigenschaften zum Ziel !



Speicherverwaltung

Die Kosten-Leistungs-Perspektive



Speicherverwaltung

Register
Scratchpad

wird explizit vom Programmierer/Compiler kontrolliert

Cache

Hardware-Kontrolle

RAM

RAM
Abstraktion

ROM

flüchtig

Disk

DVD-ROM

Dateiabstraktion

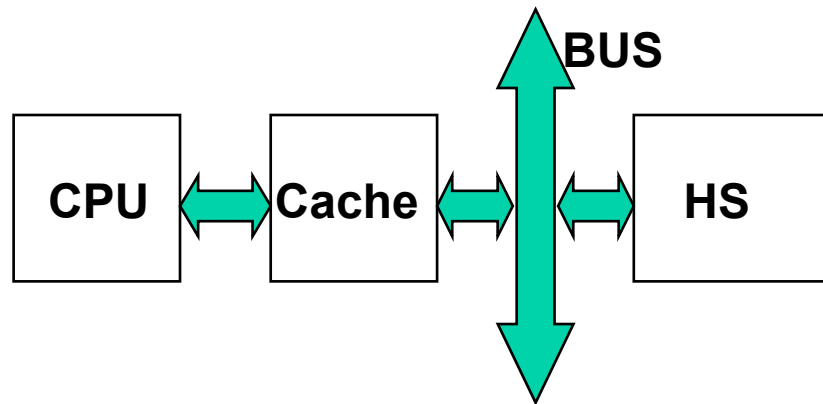
CD-ROM

persistent

Tape

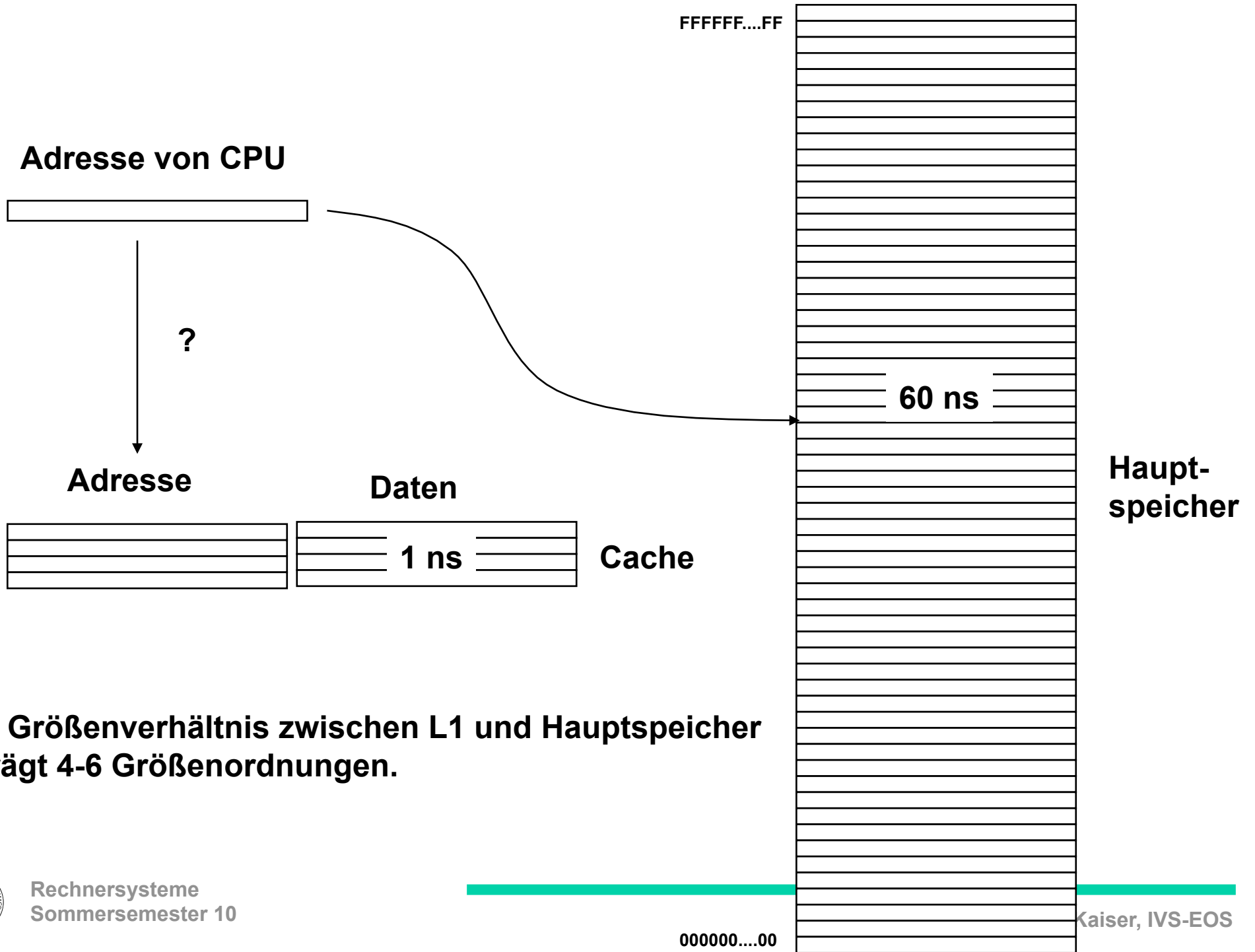
unter Kontrolle des Betriebssystems





- ➔ Die von der CPU gelieferte Adresse wird mit den im Cache gespeicherten Adressen verglichen.
- ➔ Gleichzeitig wird der Hauptspeicher adressiert.
- ➔ Liegt das adressierte Speicherwort im Cache (Treffer, Hit) wird es in die CPU geladen. Der Hauptspeicherzugriff wird abgebrochen.
- ➔ Liegt das adressierte Speicherwort nicht im Cache (Nicht-Treffer, Miss) wird es (zusammen mit anderen Speicherworten) in den Cache und die CPU geladen.
- ➔ Ist der Cache vollständig belegt, müssen Speicherworte ausgelagert werden.





Das Größenverhältnis zwischen L1 und Hauptspeicher beträgt 4-6 Größenordnungen.



Warum funktioniert ein Cache?

Warum findet man ein referenziertes Speicherwort "häufig" im Cache?

Das Prinzip der Lokalität:

zeitliche Lokalität: wenn ein Speicherwort benutzt wurde wird es mit hoher Wahrscheinlichkeit nach einem "kurzen" Zeitintervall wieder benutzt.

räumliche Lokalität: wenn ein Speicherwort benutzt wurde werden Speicherworte der (Adressen-) Umgebung mit hoher Wahrscheinlichkeit auch genutzt.



Quantifizierung der Lokalität - die Trefferrate

$$\text{Trefferrate: } h = \frac{\text{Anzahl der Treffer}}{\text{Anzahl der Gesamtzugriffe}} \cdot 100\%$$

$$\text{Fehl-Rate: } m = \frac{\text{Anzahl der Nicht-Treffer}}{\text{Anzahl der Gesamtzugriffe}} \cdot 100\% \\ \text{oder Miss-Rate}$$

$$\text{Durchschnittliche Zugriffszeit : } T_{AV} = (h \cdot T_{\text{cache}}) + (m \cdot T_{HS})$$

$$h = 90, T_{\text{cache}} = 1 \text{ ns}, T_{HS} = 50 \text{ ns} \Rightarrow T_{AV} = (0,9 \cdot 1) + (0,1 \cdot 50) = 5,9 \text{ ns}$$

$$h = 99, T_{\text{cache}} = 1 \text{ ns}, T_{HS} = 50 \text{ ns} \Rightarrow T_{AV} = (0,99 \cdot 1) + (0,01 \cdot 50) = 1,49 \text{ ns}$$



Beschleunigung durch Caches

Beispiel:

CPU mit CPI=1 wenn alle Daten im Cache sind.

Load und Store machen 50% der Befehle aus.

Der Aufwand bei einem Cache miss beträgt 25 CPU Zyklen

Die Miss-Rate beträgt 2%.

Wie viel schneller ist der Computer, wenn alle Daten im Cache sind?

$$\begin{aligned}CT &= (\text{Taktzyklen} + \text{Speicher-Wartezyklen}) \cdot \text{Zykluszeit} \\ &= (IC \cdot CPI + 0) \cdot \text{Zykluszeit} \quad (IC: \text{Anzahl der Instruktionen}) \\ &= (IC \cdot 1,0 + 0) \cdot \text{Zykluszeit}\end{aligned}$$

$$\begin{aligned}\text{Speicher-Wartezyklen} &= IC \cdot \text{Speicherzugriffe/Instruktion} \cdot \text{Fehl-Rate} \cdot \text{Fehl-Aufwand} \\ &= IC \cdot (1 + 0,5) \cdot 0,02 \cdot 25 = IC \cdot 0,75\end{aligned}$$

1 Zugriff/Instr. in der IF-Phase
und 0,5 Zugriffe/Instr. in der Datenphase

$$CT = IC \cdot (1 + 0,75) \cdot \text{Zykluszeit}$$

$$\frac{\text{Ausführungszeit}_1}{\text{Ausführungszeit}_2} = \frac{IC \cdot 1,75 \cdot \text{Taktzyklus}}{IC \cdot 1,0 \cdot \text{Taktzyklus}} = 1,75 \quad \sim 1,75 \text{ mal schneller bei nur 2\% Fehl-Rate!}$$



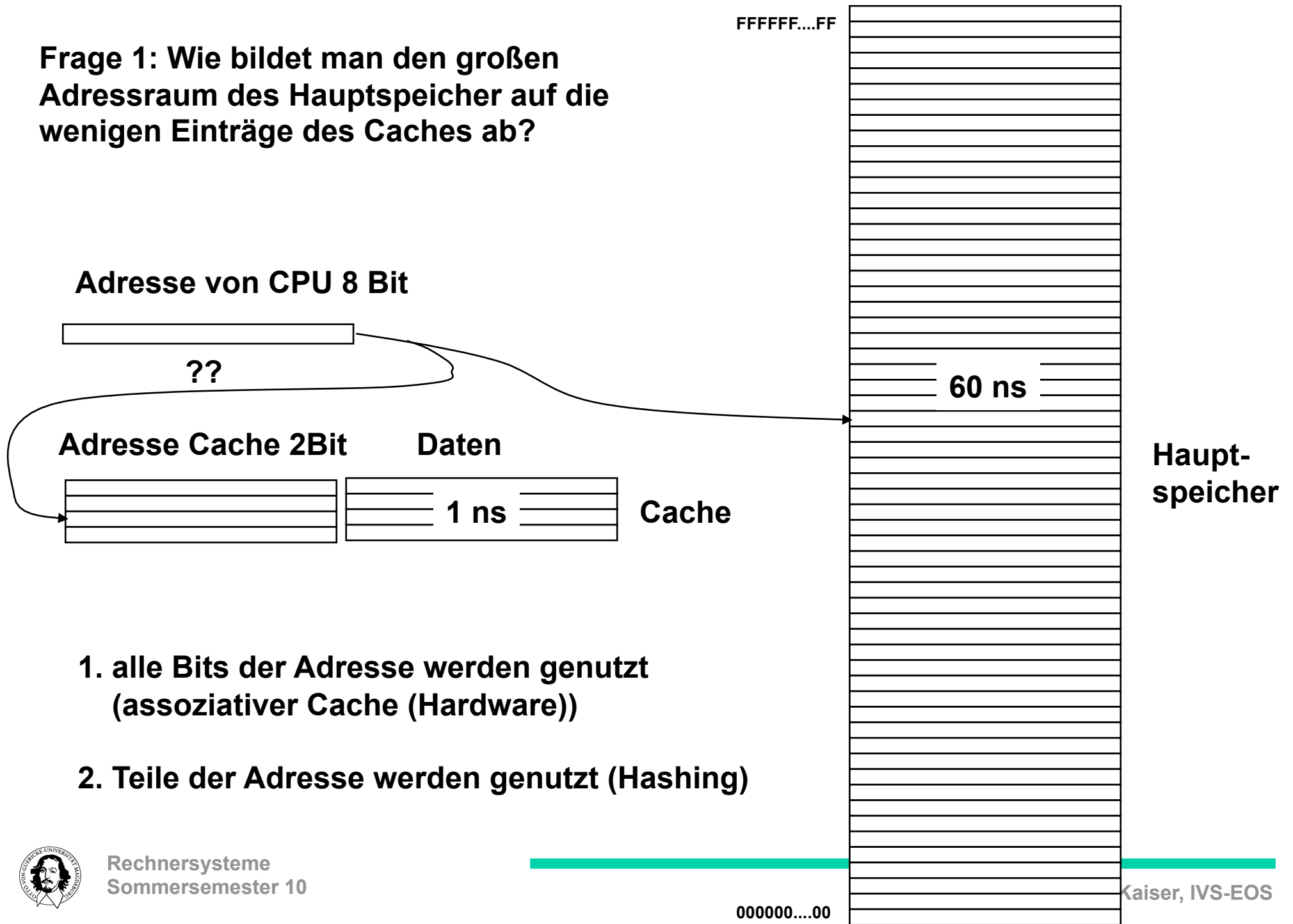
4 Fragen zur Speicherhierarchie*

- 1.) **Wie kann der sehr große Adressraum des Hauptspeichers auf die wenigen Einträge im Cache abgebildet werden?
oder: wohin wird ein Datenblock im Cache eingelagert?**
- 2.) **Wie wird ein Datenblock im Cache gefunden?
oder: ist der adressierte Block auch der gesuchte Block?**
- 3.) **Wann und wie werden Datenblöcke im Cache ersetzt?**
- 4.) **Was passiert bei Schreibzugriffen?
oder: wie wird die Übereinstimmung zwischen Cache und Hauptspeicher sichergestellt?**

* John L. Hennessy, David A. Patterson: Computer-Architecture. A Quantitative Approach



Frage 1: Wie bildet man den großen Adressraum des Hauptspeicher auf die wenigen Einträge des Caches ab?

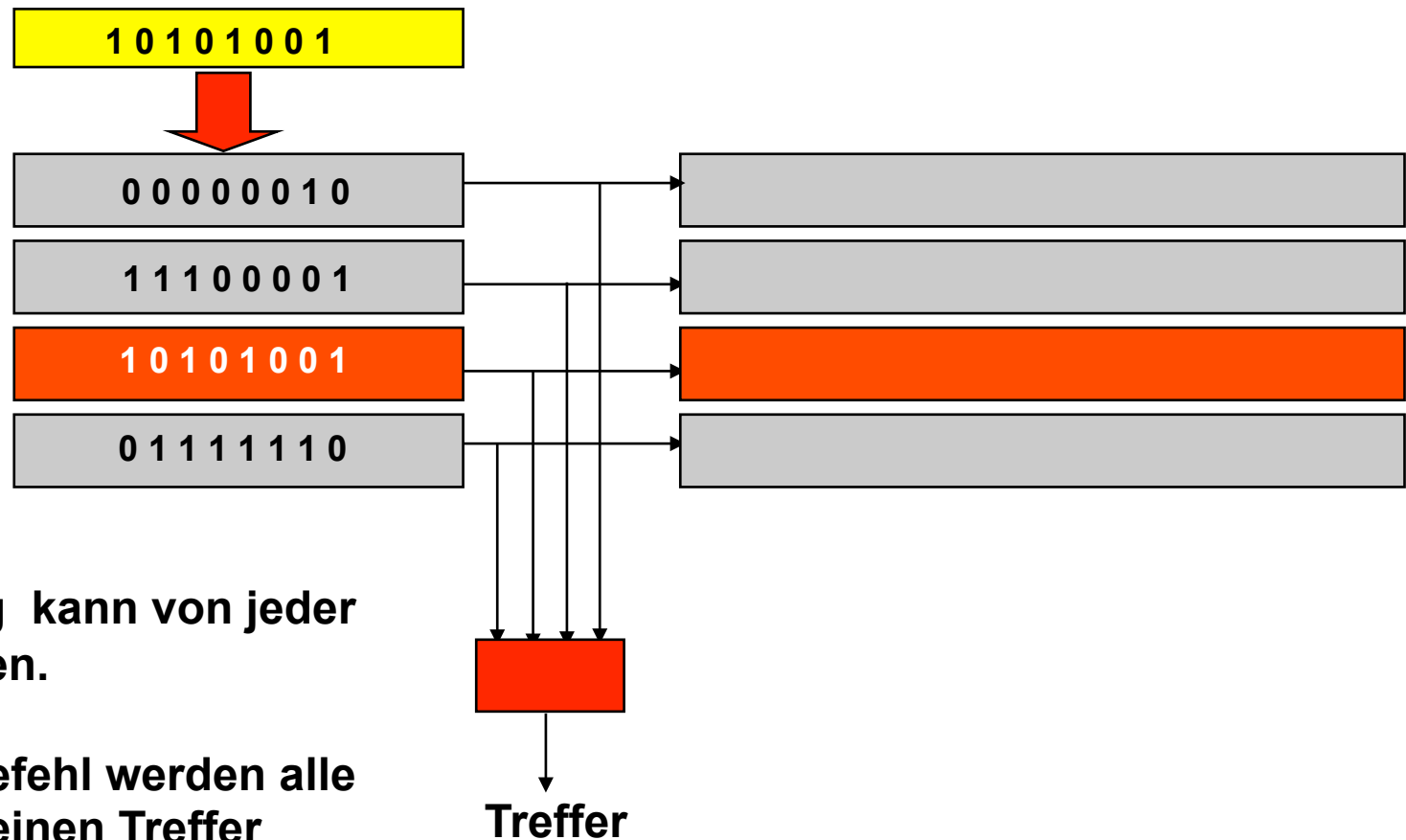


1. alle Bits der Adresse werden genutzt (assoziativer Cache (Hardware))
2. Teile der Adresse werden genutzt (Hashing)



(Voll-) assoziativer Cache

Adresse (Suchschlüssel)

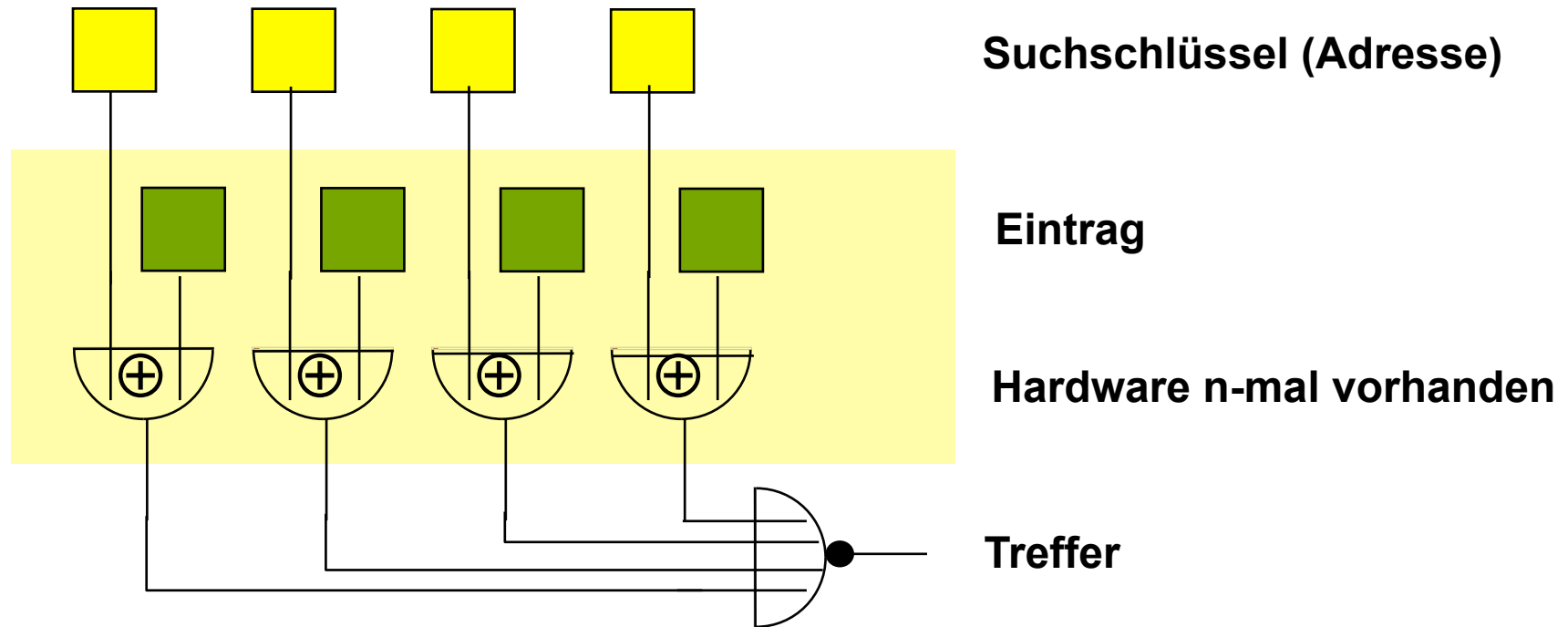


Ein beliebiger Eintrag kann von jeder Adresse belegt werden.

Bei jedem Speicherbefehl werden alle Einträge parallel auf einen Treffer durchsucht



Prinzip des Assoziativspeichers



+ sehr schnell

+ wenig Verwaltungsaufwand, alle Einträge können universell verwendet werden

- sehr aufwändig durch den Komparator für jeden Eintrag

➔ wurde/wird für sehr kleine Caches genutzt, z.B. zur Adressübersetzung



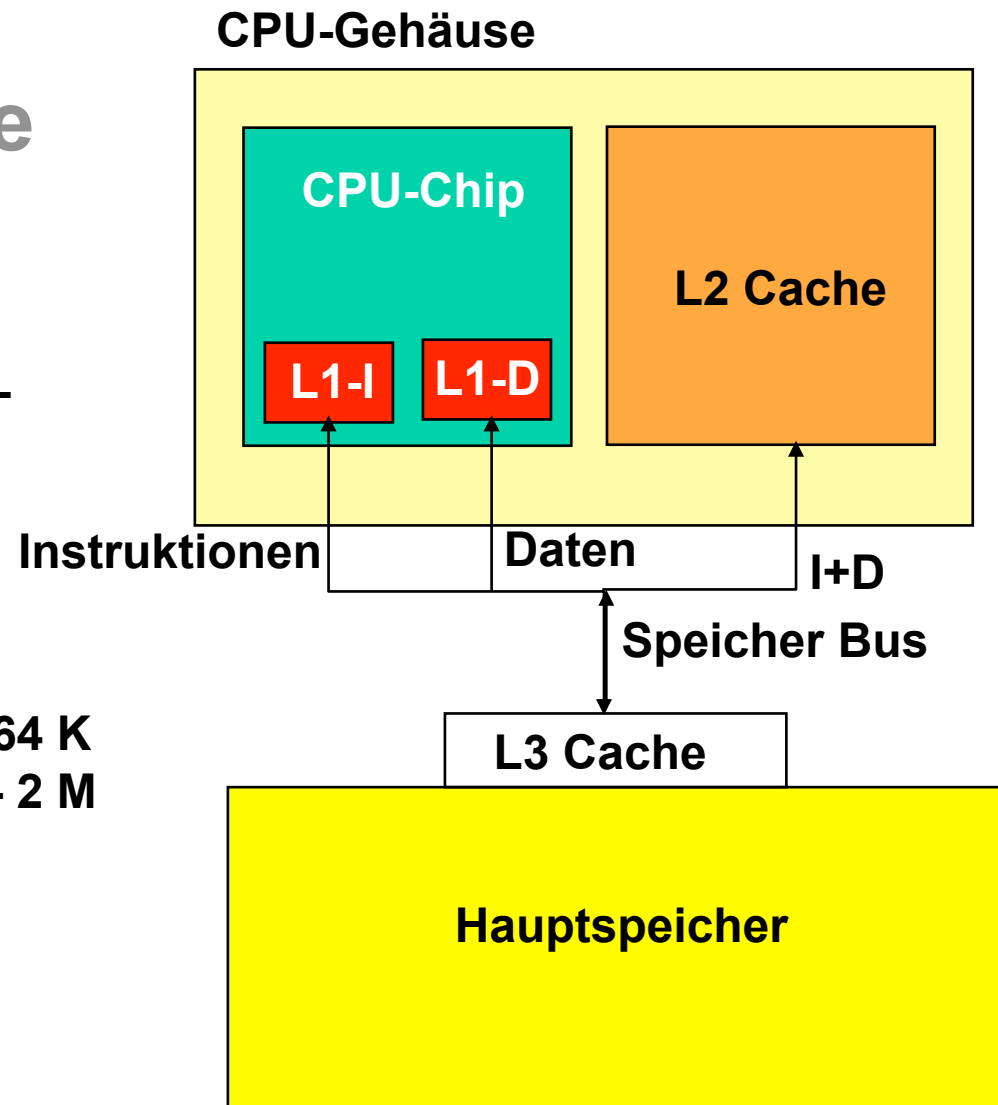
Speicherhierarchie und Caches



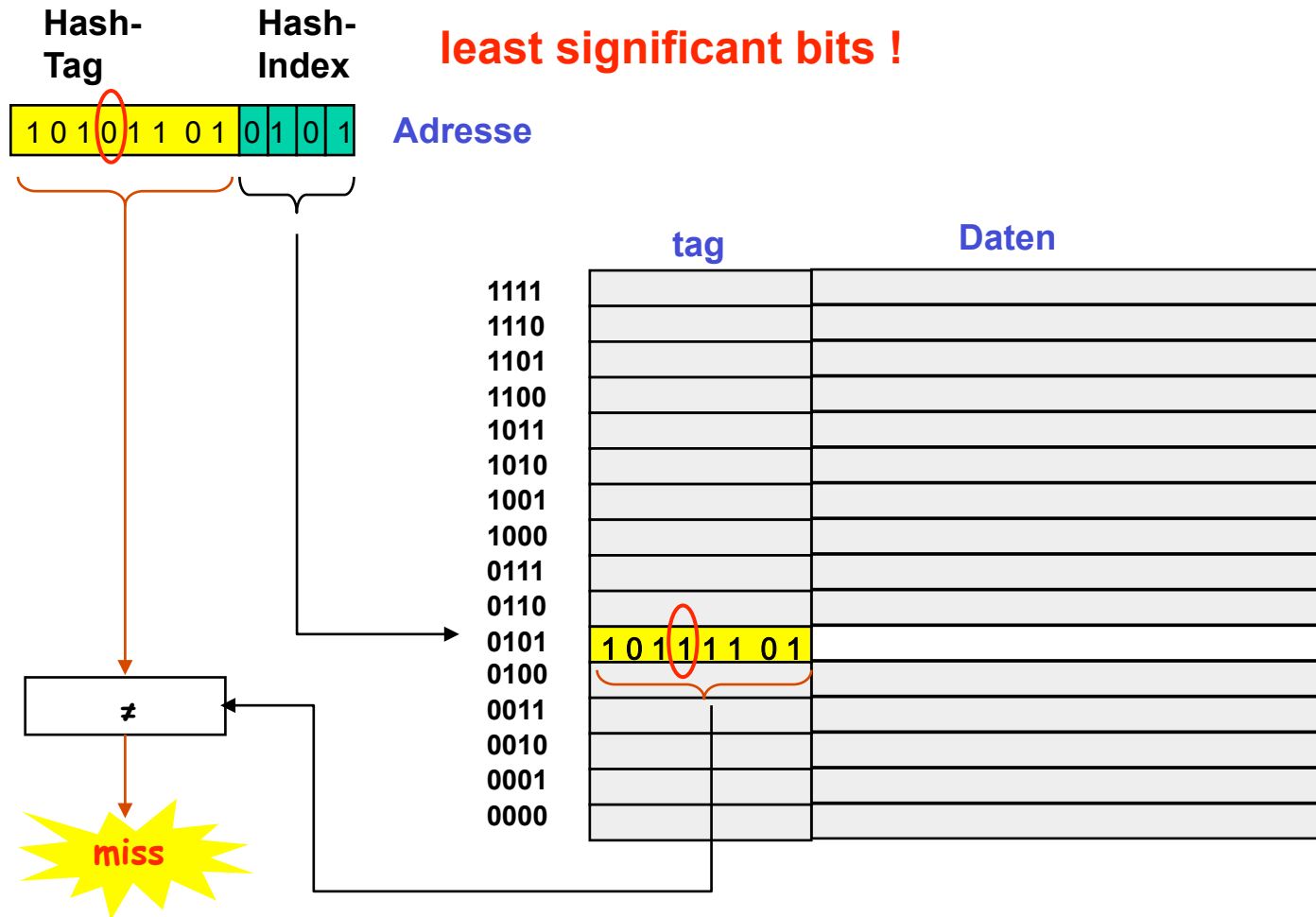
Typische Cache Hierarchie

Der Zugriff zum Cache ist ein transparenter Speicherzugriff mit der effektiven Adresse.

L1 ~ 16 - 64 K
L2 ~ 256 - 2 M



Einfacher direkt abbildender Cache

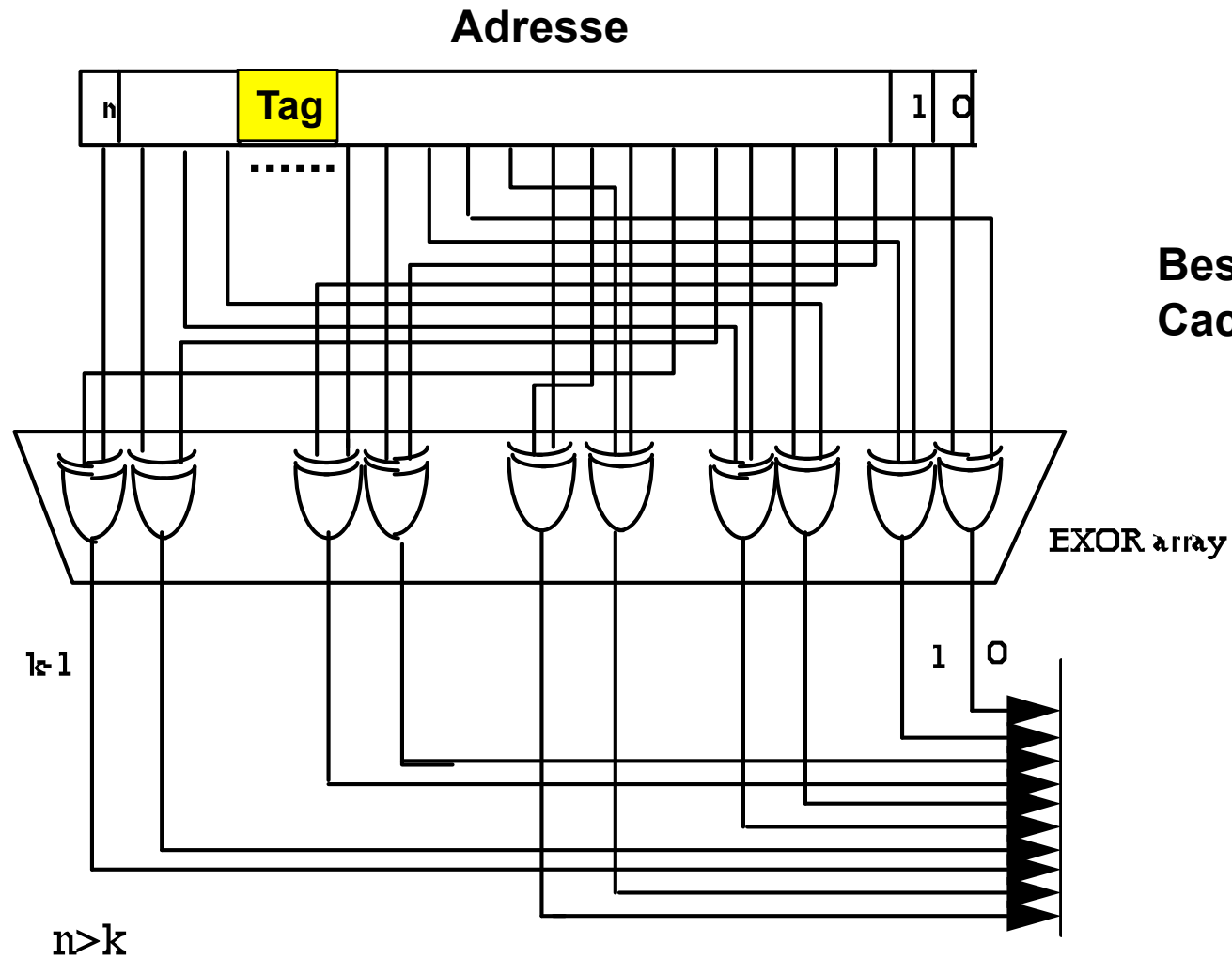


Probleme:

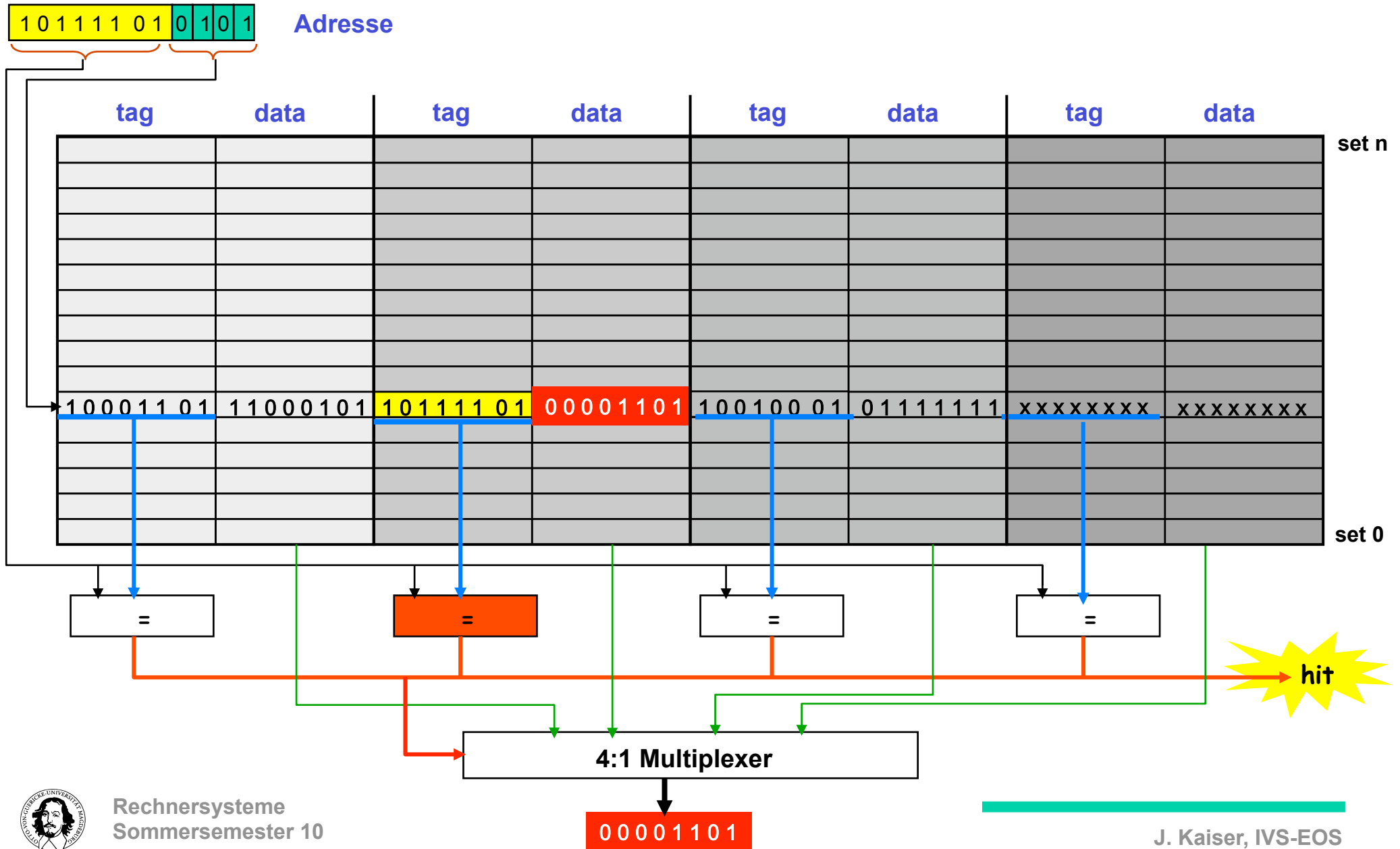
**Schlechte Ausnutzung des Cache Adreßraums
Viele Kollisionen**



Hash-Funktion mit EXOR-Array



4-Wege Mengen-assoziativer Cache

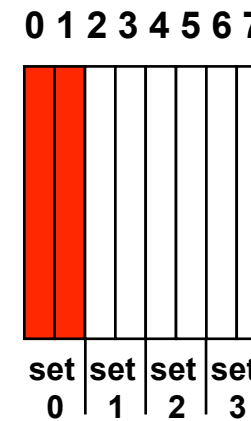
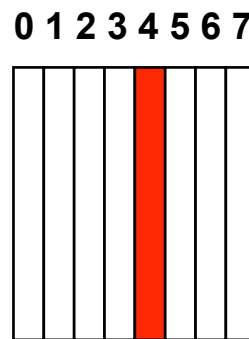
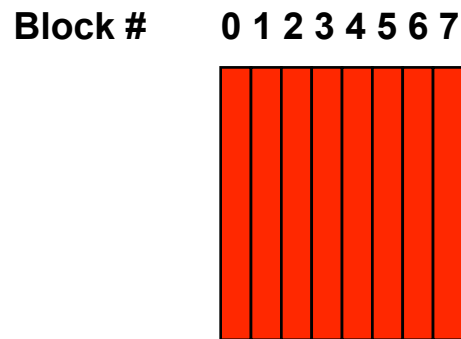


An welche Stelle kann ein Datenblock eingelagert werden?

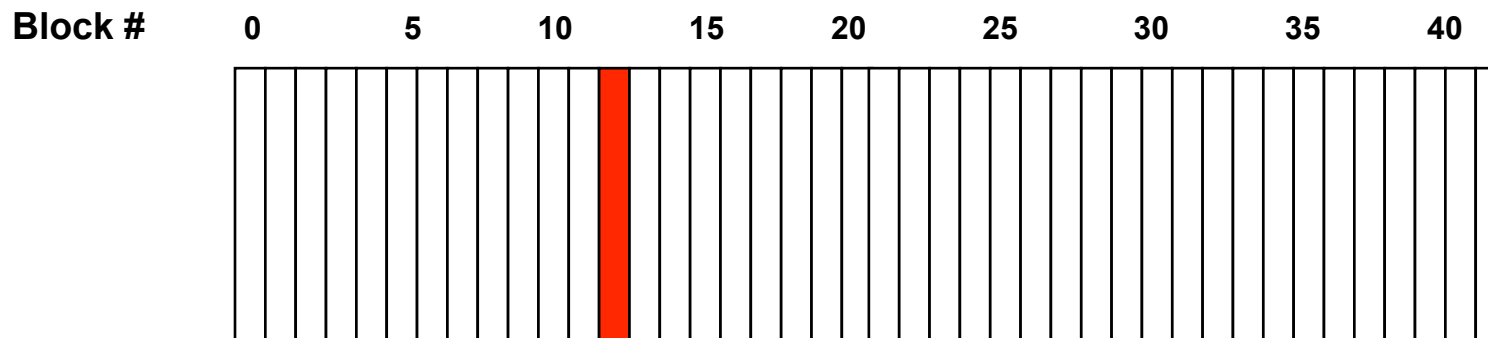
**Voll-assoziativer
Cache: Block 12
kann überall ein-
gelagert werden**

**direkt abbildender
Cache: Block 12
kann nur in Eintrag 4 ein-
gelagert werden.
(12 mod 8)**

**Mengen-assoziativer
Cache: Block 12
kann überall in Set 0 ein-
gelagert werden.
(12 mod 4)**



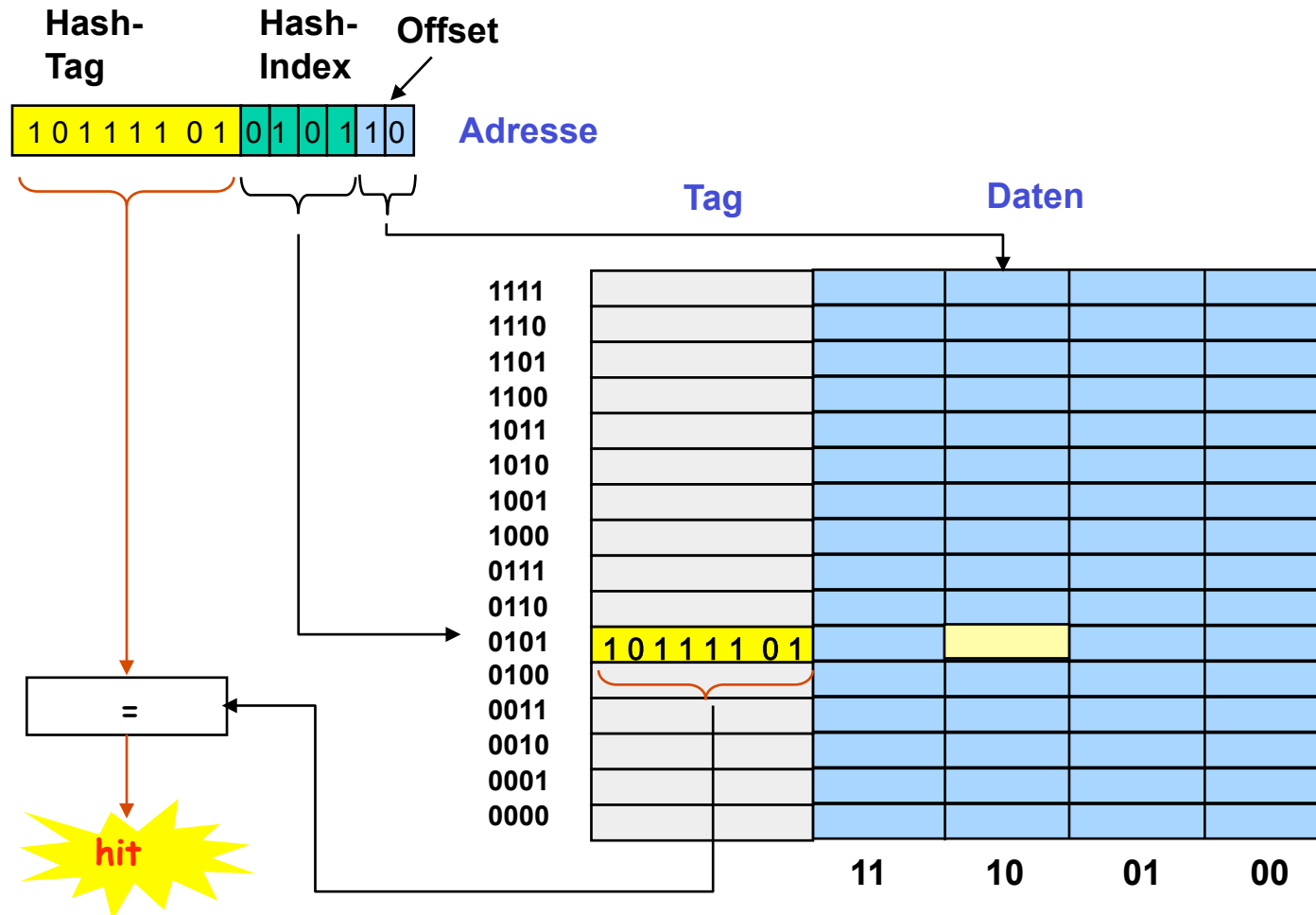
Caches



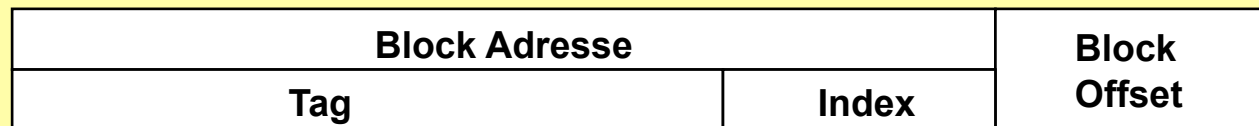
**Haupt-
speicher**



Wie wird ein Speicherwort im Cache identifiziert?



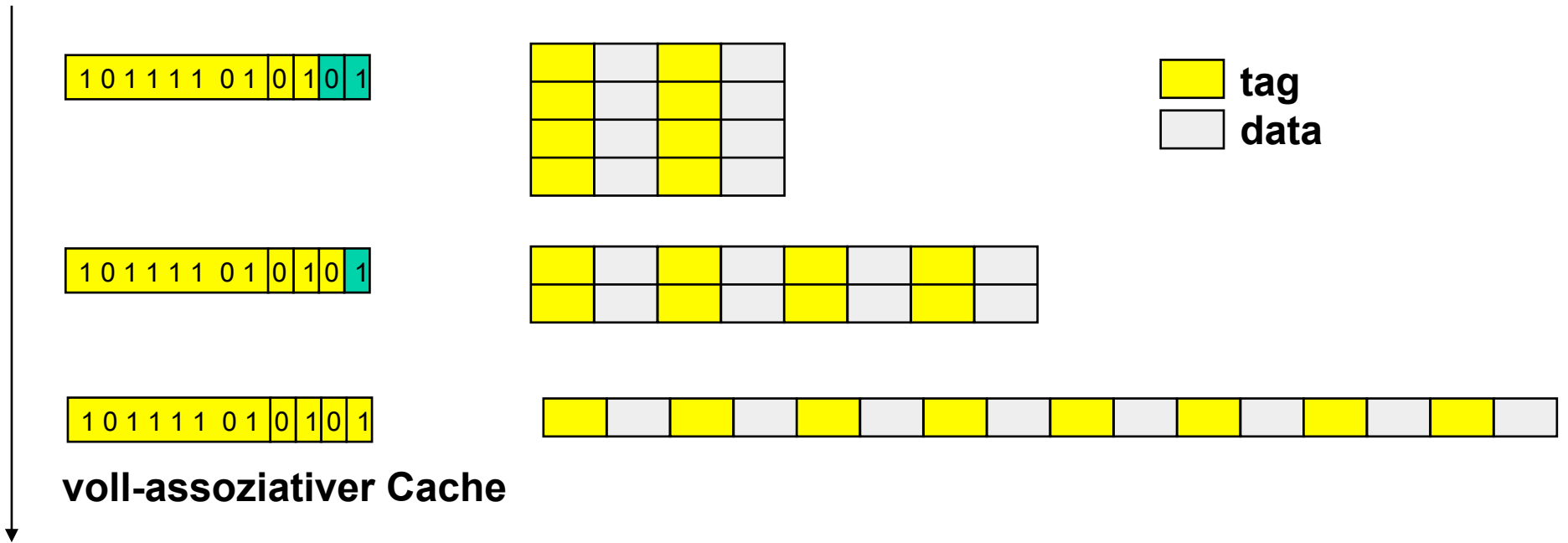
Die Unterteilung in die 3 wesentlichen Felder einer Adresse für direkt abbildende Caches



Diskussion: Erhöhung der Assoziativität unter Beibehaltung der Cachegröße

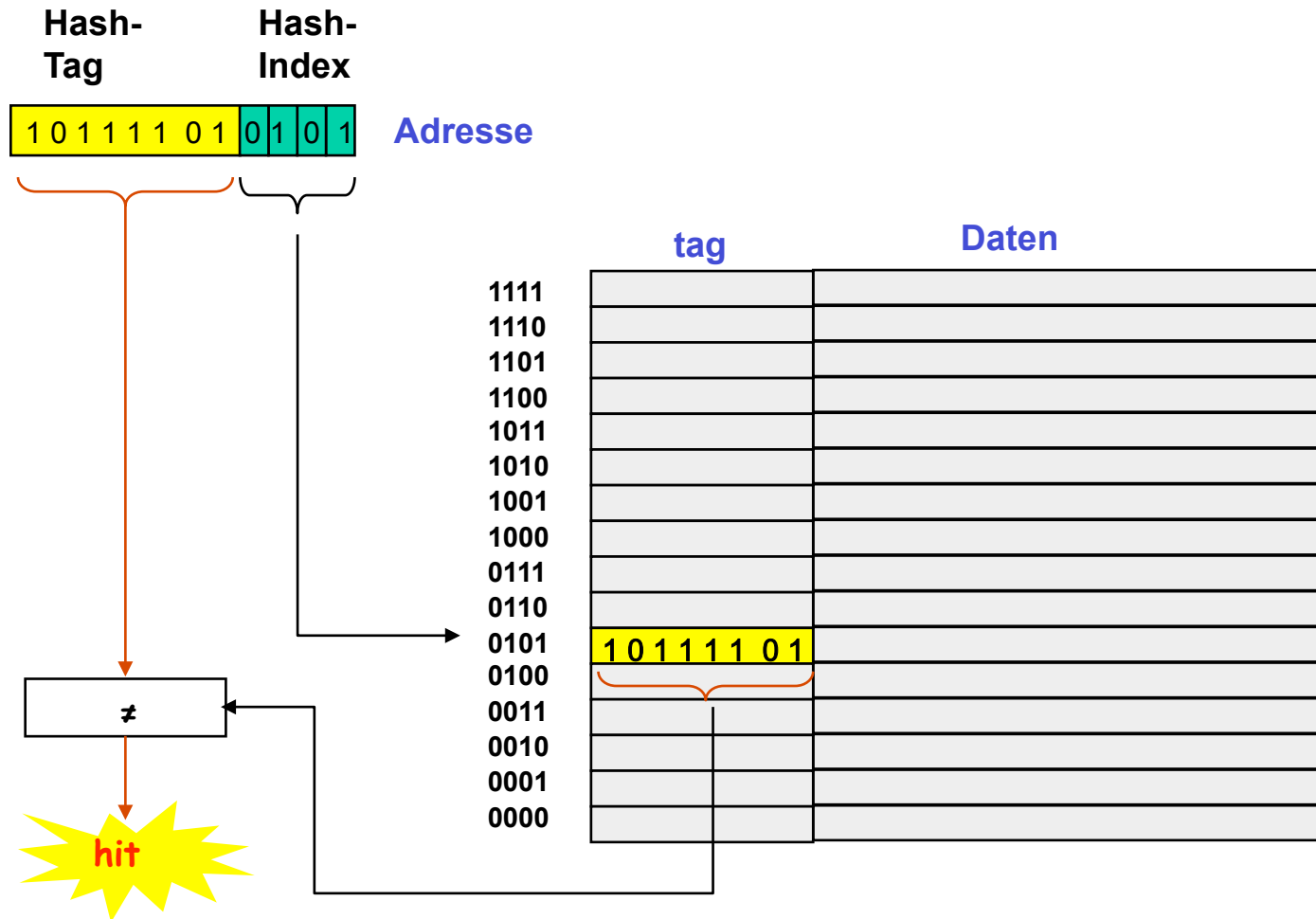
| | | |
|--------------------|-------|--------------|
| Datenblock Adresse | | Block Offset |
| Tag | Index | |

Vergrößern des Tag-Feldes --> mehr Blocks/Set --> Hardware-Aufwand wächst



| Cache (KB) | Degree assoc. | Total missrate | Compulsory | | Capacity | | Conflict | |
|------------|---------------|----------------|------------|------|----------|------|----------|-----|
| 4 | 1-way | 0.098 | 0.0001 | 0.1% | 0.070 | 72% | 0.027 | 28% |
| 4 | 2-way | 0.076 | 0.0001 | 0.1% | 0.070 | 93% | 0.005 | 7% |
| 4 | 4-way | 0.071 | 0.0001 | 0.1% | 0.070 | 99% | 0.001 | 1% |
| 4 | 8-way | 0.071 | 0.0001 | 0.1% | 0.070 | 100% | 0.000 | 0% |
| 8 | 1-way | 0.068 | 0.0001 | 0.1% | 0.044 | 65% | 0.024 | 35% |
| 8 | 2-way | 0.049 | 0.0001 | 0.1% | 0.044 | 90% | 0.005 | 10% |
| 8 | 4-way | 0.044 | 0.0001 | 0.1% | 0.044 | 99% | 0.000 | 1% |
| 8 | 8-way | 0.044 | 0.0001 | 0.1% | 0.044 | 100% | 0.000 | 0% |
| 16 | 1-way | 0.049 | 0.0001 | 0.1% | 0.040 | 82% | 0.009 | 17% |
| 16 | 2-way | 0.041 | 0.0001 | 0.2% | 0.040 | 98% | 0.001 | 2% |
| 16 | 4-way | 0.041 | 0.0001 | 0.2% | 0.040 | 99% | 0.000 | 0% |
| 16 | 8-way | 0.041 | 0.0001 | 0.2% | 0.040 | 100% | 0.000 | 0% |
| 32 | 1-way | 0.042 | 0.0001 | 0.2% | 0.037 | 89% | 0.005 | 11% |
| 32 | 2-way | 0.038 | 0.0001 | 0.2% | 0.037 | 99% | 0.000 | 0% |
| 32 | 4-way | 0.037 | 0.0001 | 0.2% | 0.037 | 100% | 0.000 | 0% |
| 32 | 8-way | 0.037 | 0.0001 | 0.2% | 0.037 | 100% | 0.000 | 0% |
| 64 | 1-way | 0.037 | 0.0001 | 0.2% | 0.028 | 77% | 0.008 | 23% |
| 64 | 2-way | 0.031 | 0.0001 | 0.2% | 0.028 | 91% | 0.003 | 9% |
| 64 | 4-way | 0.030 | 0.0001 | 0.2% | 0.028 | 95% | 0.001 | 4% |
| 64 | 8-way | 0.029 | 0.0001 | 0.2% | 0.028 | 97% | 0.001 | 2% |
| 128 | 1-way | 0.021 | 0.0001 | 0.3% | 0.019 | 91% | 0.002 | 8% |
| 128 | 2-way | 0.019 | 0.0001 | 0.3% | 0.019 | 100% | 0.000 | 0% |
| 128 | 4-way | 0.019 | 0.0001 | 0.3% | 0.019 | 100% | 0.000 | 0% |
| 128 | 8-way | 0.019 | 0.0001 | 0.3% | 0.019 | 100% | 0.000 | 0% |
| 256 | 1-way | 0.013 | 0.0001 | 0.5% | 0.012 | 94% | 0.001 | 6% |
| 256 | 2-way | 0.012 | 0.0001 | 0.5% | 0.012 | 99% | 0.000 | 0% |
| 256 | 4-way | 0.012 | 0.0001 | 0.5% | 0.012 | 99% | 0.000 | 0% |
| 256 | 8-way | 0.012 | 0.0001 | 0.5% | 0.012 | 99% | 0.000 | 0% |
| 512 | 1-way | 0.008 | 0.0001 | 0.8% | 0.005 | 66% | 0.003 | 33% |
| 512 | 2-way | 0.007 | 0.0001 | 0.9% | 0.005 | 71% | 0.002 | 28% |
| 512 | 4-way | 0.006 | 0.0001 | 1.1% | 0.005 | 91% | 0.000 | 8% |
| 512 | 8-way | 0.006 | 0.0001 | 1.1% | 0.005 | 95% | 0.000 | 4% |

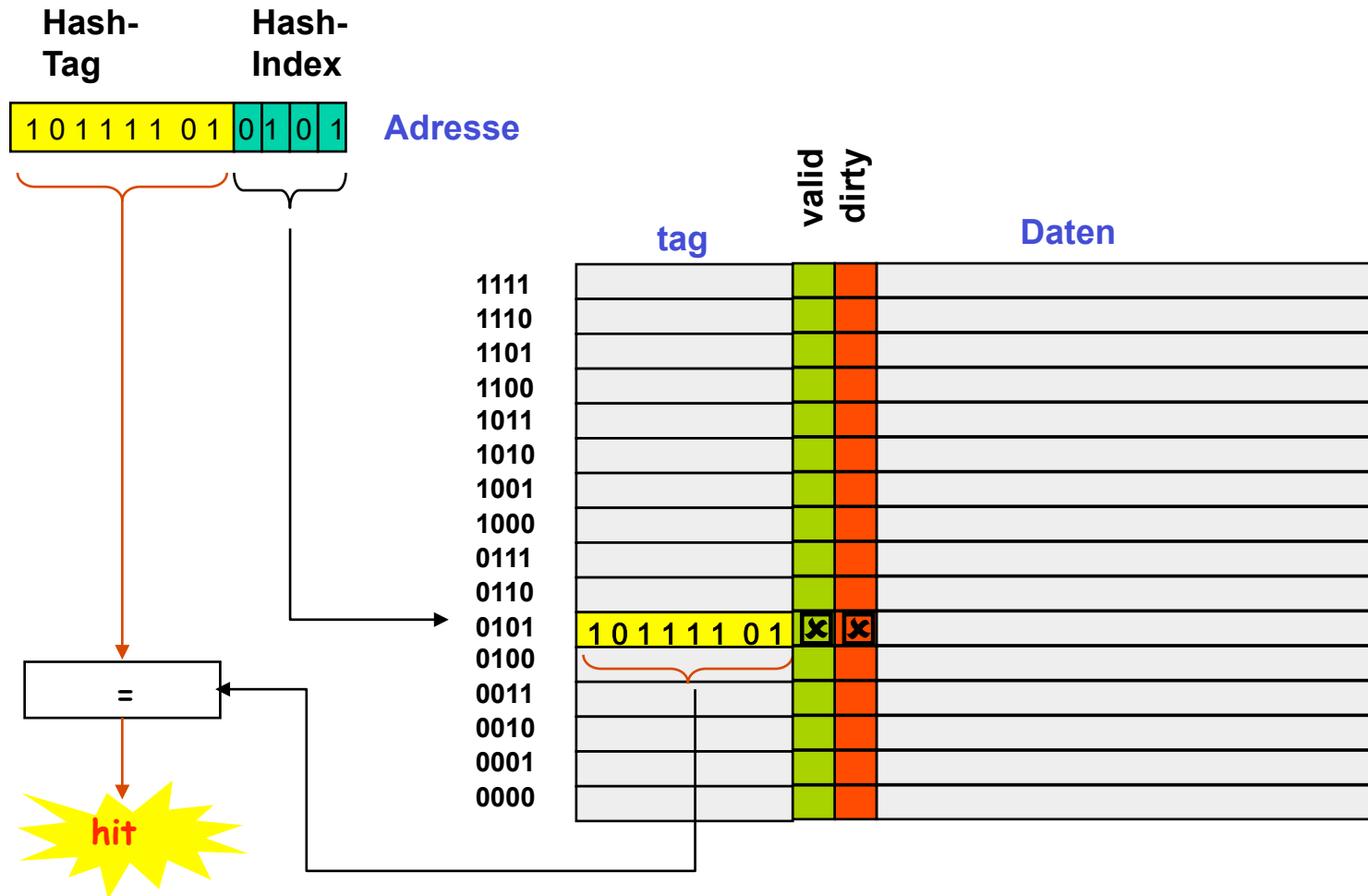
Frage 3: Welcher Block sollte ersetzt werden?



Beim direkt abbildenden Cache ist die Ersetzung eindeutig festgelegt durch den Index.



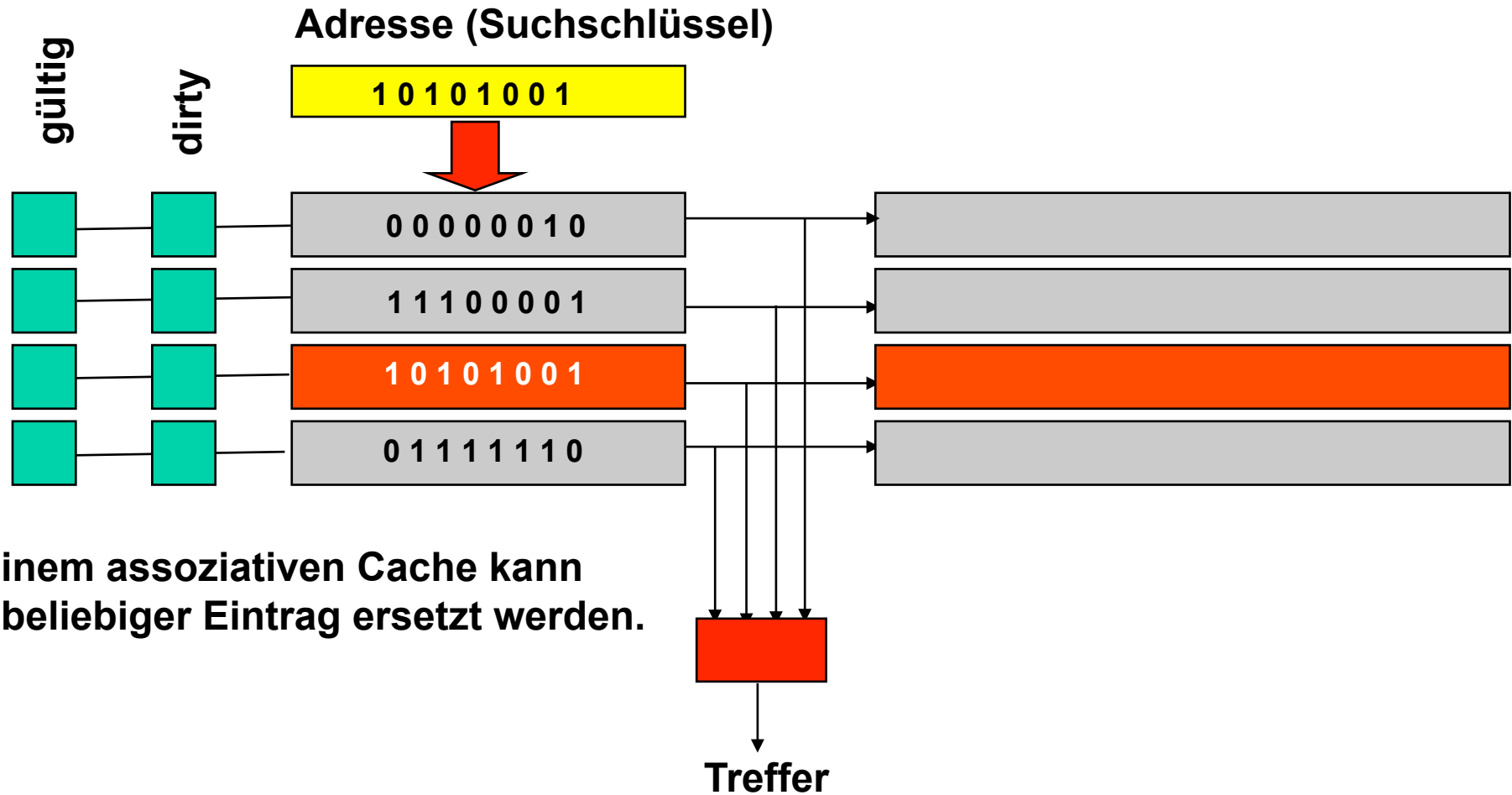
Frage 3: Welcher Block sollte ersetzt werden?



Beim direkt abbildenden Cache ist die Ersetzung eindeutig festgelegt durch den Index.



Frage 3: Welcher Block sollte ersetzt werden?



In einem assoziativen Cache kann ein beliebiger Eintrag ersetzt werden.



Ersetzungsstrategien bei assoziativen Caches

Least recently used (LRU)

First-In-First-Out (FIFO)

Zufall (Random)

Assoziativität

| Cache größe | 2-Wege | | | 4-Wege | | | 8-Wege | | |
|----------------|--------|--------|-------|--------|--------|-------|--------|--------|-------|
| | LRU | Random | Fifo | LRU | Random | Fifo | LRU | Random | Fifo |
| 16 KB | 114,1 | 117,3 | 115,5 | 111,7 | 115,1 | 113,3 | 109,0 | 111,8 | 110,4 |
| 64 KB | 103,4 | 104,3 | 103,9 | 102,4 | 102,3 | 103,1 | 99,7 | 100,5 | 100,3 |
| 256 KB | 92,2 | 92,1 | 92,5 | 92,1 | 92,1 | 92,5 | 92,1 | 92,1 | 92,5 |

**Datencache Misses/10000 Instruktionen, Alpha, 64 Byte Blockgröße,
10 Benchmarkläufe aus der SPEC2000 Menge.**



Lese- und Schreibzugriffe zum Speicher

Lesen:

Cache Eintrag Validierung (vergleichen von Tag) und lesen kann nebenläufig erfolgen.

Bei "hit" wurde der korrekte Wert gelesen.

Bei "miss" wird der gelesene Wert ignoriert.

➔ Lesen kann mit maximaler Cache-Zugriffsrate erfolgen!

Beim Füllen des Caches kann mehr eingelesen werden als notwendig ist.

Schreiben:

Cache Eintrag Validierung (vergleichen von Tag) muss vor dem Ändern eines Cache-Blocks erfolgen.

Cache und Speicher müssen konsistent gehalten werden.

➔ "Write Through": Daten werden bei jeder Änderung auch im Speicher aktualisiert

➔ "Write Back": Daten werden nur im Cache aktualisiert und nur bei Verdrängung aktualisiert.



Vergleich von write-through und write-back

Write-Through:

- + einfach zu implementieren
- + Cache und Hauptspeicher stimmen immer überein
- + Read-misses führen nie zu einem Zurückschreiben des Blocks
- jedes "write" führt zu einem Hauptspeicherzugriff mit entsprechenden Verzögerungen "write stall" (kann teilweise durch write-Puffer aufgefangen werden).

Write-Back:

- ++ mehrere Schreibzugriffe können auf dem Cache durchgeführt werden.
- zusätzliches ("modified" oder "dirty") Bit im Cache
- Cache und Hauptspeicher sind nicht kohärent
- ein "read" kann ein Zurückschreiben auslösen (wenn ein entsprechender Block verdrängt wird)



Was tun bei einem "Write-Miss"?

Write Allocate: Block wird eingelagert und write wird auf dem Cache ausgeführt.
Verhalten wie bei einem "Read Miss".

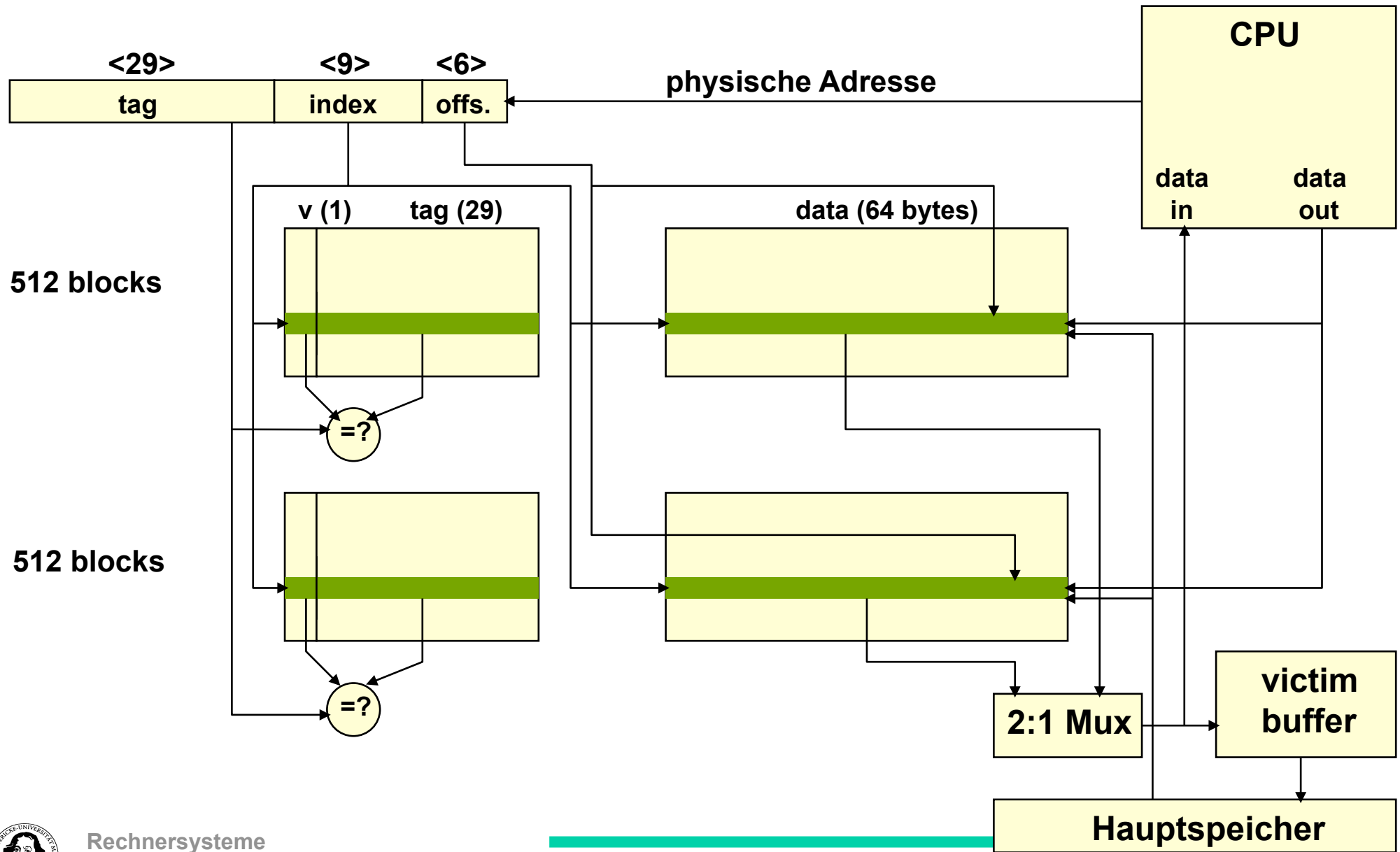
No Write Allocate: Block wird nur im Hauptspeicher beschrieben, nicht eingelagert.

| Aktion: | No Allocate | Allocate |
|------------------|-------------|----------|
| Write mem [100]; | Miss | Miss |
| Write mem [100]; | Miss | Hit |
| Write mem [200]; | Miss | Miss |
| Read mem [200]; | Miss | Hit |
| Write mem [100]; | Miss | Hit |




Wie die Allokations- und die Rückschreibestrategie kombinieren?



Organisation des Alpha 21264 Daten Caches



Metriken zur Characterisierung der Cache Leistung

- **Treffer- und Fehlzugriff (Miss) -Rate** 
- **Mittlere Zugriffszeit zum Speicher** 
- **Ausführungszeit eines Programms** 



Cache Analyse

mittlere Zugriffszeit = Cache Zugriffszeit + Fehl-Rate · Fehl-Aufwand

Bisher haben wir verwendet: $T_{AV} = (h \cdot T_{cache}) + (m \cdot T_{HS})$

Diese Formel berücksichtigt nicht, dass die CPU immer, also auch bei einem Fehlzugriff über den Cache zugreifen muss.

$$(h \cdot T_{cache}) + (m \cdot T_{HS}) \leq T_{cache} + m \cdot T_{HS}$$



Cache Analyse

Frage: Sind bei gleicher Gesamtgröße getrennte Instruktions- und Datencaches besser als ein gemeinsamer Cache?

Grundlage: Fehlzugriff-Rate pro 1000 Instruktionen

| Cachegröße KB | Instruktions- Cache | Daten- Cache | gemeinsamer Cache |
|--------------------------|--------------------------------|-------------------------|------------------------------|
| 8 | 8,16 | 44,0 | 63,0 |
| 16 | 3,82 | 40,9 | 51,0 |
| 32 | 1,36 | 38,4 | 43,3 |
| 64 | 0,61 | 36,9 | 39,4 |
| 128 | 0,30 | 35,3 | 36,2 |
| 256 | 0,02 | 32,6 | 32,9 |

74% der Referenzen sind Instruktionen

Cache: 2-Wege-Assoziativ, 64 Byte Blöcke

Daten wurden gemessen durch: 5 x SPECint2000 (gap,gcc, gzip, mcf, perl)

5 x SPECfp2000 (applu, art, equake, lucas, swim)



Wandlung von Fehlern pro 1000 Instruktionen (FPTI) in eine Fehlerrate

$$\text{MR} = \frac{\frac{\text{FPTI}}{1000}}{\frac{\text{Speicherzugriffe}}{\text{Zahl der Instruktionen}}}$$

36% Datenzugriffe
74% Instruktionszugriffe

MR: Fehlzugriff-Rate
(miss rate)

$$\text{MR}_{16 \text{ KB Instruktionen}} = \frac{3,82/1000}{1,00} = 0,004$$

$$\text{MR}_{16 \text{ KB Daten}} = \frac{40,9/1000}{0,36} = 0,114$$

$$\text{MR}_{32 \text{ KB Gemeinsam}} = \frac{43,3/1000}{1,00 + 0,36} = 0,0318$$

$$\text{MR}_{2 \times 16 \text{ KB split}} = (74\% \times 0,004) + (26\% \times 0,114) = 0,0324$$



Cache Analyse

Analyse der mittlere Zugriffszeiten:




$$T_{av} = \% \text{ Instruktionen} \cdot (t_{\text{treffer}} + m r_{\text{instruktionen}} + t_{\text{Fehleraufwand}}) \\ + \% \text{ Daten} \cdot (t_{\text{treffer}} + m r_{\text{instruktionen}} + t_{\text{Fehleraufwand}})$$

$$T_{av\text{-split}} = 74\% \cdot (1 + 0,004 \cdot 100) + 26\% \cdot (1 + 0,114 \cdot 100) \\ = 74\% \cdot 1,4 + 26\% \cdot 12,4 = 1,036 + 3,224 = 4,26$$

$$T_{av\text{-gemeinsam}} = 74\% \cdot (1 + 0,0318 \cdot 100) + 26\% \cdot (1 + 1 + 0,0318 \cdot 100) \\ = 74\% \cdot 4,18 + 26\% \cdot 5,18 = 3,093 + 1,347 = 4,44$$



Metriken zur Characterisierung der Cache Leistung

- **Treffer- und Fehlzugriff (Miss) -Rate** 
- **Mittlere Zugriffszeit zum Speicher** 
- **Ausführungszeit eines Programms** 



Cache Analyse

Ausführungszeit und Cache Performance

Frage: Welche Auswirkung hat ein Cache auf die Ausführungszeit eines Programms?

unter Benutzung der Cache-Fehlzugriffe/ 1000 Instruktionen:

$$CT = IC \cdot (CPI + SC/I) \cdot ZZ$$

SC: Speicher-Wartezyklen

I: Instruktion

ZZ: Zykluszeit

Beispiel:

$$SC = 100$$

$$CPI = 1$$

(mittlere) Cache-Fehlzugriffe (Misses) = 2%

(mittlere) Speicherzugriffe/Instruktion = 1,5

(mittlere) Cache- Fehlzugriffe/1000 Instruktionen = 30

$$\begin{aligned} CT &= IC \cdot (1,0 + (30/1000 \cdot 100)) \cdot ZZ \\ &= IC \cdot 4,00 \cdot ZZ \end{aligned}$$



Cache Analyse

Ausführungszeit und Cache Performance

unter Benutzung der Cache-Fehlzugriff-Rate:

$$CT = IC \cdot (CPI + MR \cdot Z/I \cdot MP) \cdot ZZ$$

MR: Fehlzugriffs-Rate (miss rate)

MP: Fehlzugriffs-Aufwand (miss penalty)

I: Instruktion

Z: Zugriffszeit zum Speicher

ZZ: Zykluszeit

Beispiel:

$$SC = 100$$

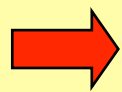
$$CPI = 1$$

(mittlere) Cache-Fehlzugriffe (Misses) = 2%

(mittlere) Speicherzugriffe/Instruktion = 1,5

(mittlere) Cache- Fehlzugriffe/1000 Instruktionen = 30

$$\begin{aligned} CT &= IC \cdot (1,0 + (1,5 \cdot 2\% \cdot 100)) \cdot ZZ \\ &= IC \cdot 4,00 \cdot ZZ \end{aligned}$$



Verbesserung
durch Cache

mit perfektem cache:

$$1 \cdot IC \cdot ZZ$$

mit Cache-Fehlzugriffen:

$$4 \cdot IC \cdot ZZ$$

ohne Cache:

$$1,0 + 100 \cdot 1,5 = 151 \cdot IC \cdot ZZ$$

