Kapitel 8.2b
Speicherhierarchie

UNI FREIBURG

Albert-Ludwigs-Universität Freiburg

Dr. Tobias Schubert, Dr. Ralf Wimmer
Professur für Rechnerarchitektur
Institut für Informatik
Technische Fakultät

Folien gehen zurück auf die Vorlesung…

Technische Grundlagen der Informatik II
– Speicherhierarchie –

Dr. Tobias Schubert
Institut für Informatik
Lehrstuhl für Technische Informatik

1. Einführung

- Ein Speichersystem muss Speicher zur Verfügung stellen
  - „Soviel Speicher wie möglich“
  - Speicher sollte so schnell wie möglich sein
  - Speicher sollte so günstig wie möglich sein

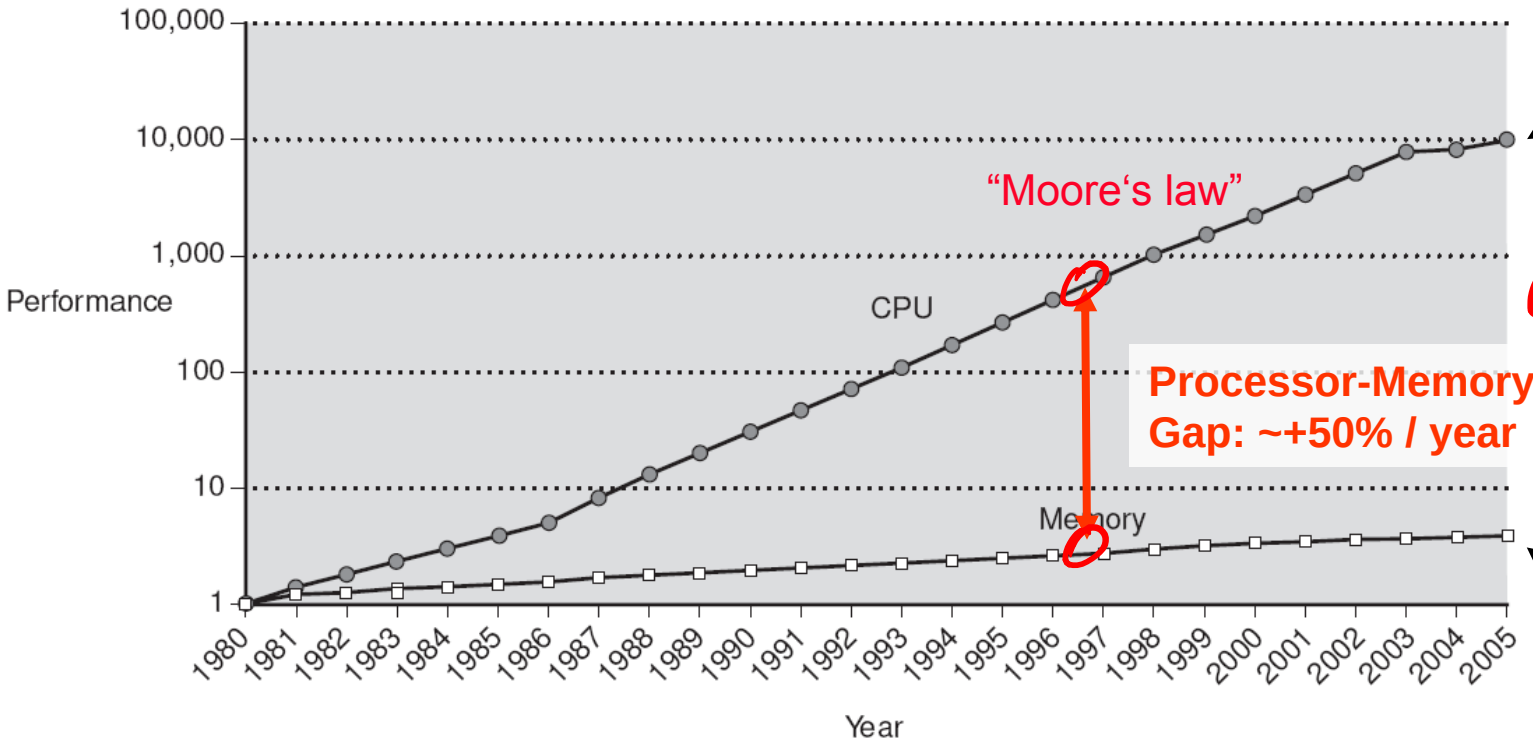
- Typische Werte im Jahr 2004

Speichertechnologie	Zugriffszeit	Kosten pro GByte
SRAM	0.5 – 5 ns	$4'000 – $10'000
DRAM	50 – 70 ns	$100 – $200
Festplatte	5 – 20 ms	$0.5 – $2

- Ziel: „Unbegrenzte“ Menge von sehr schnellem Speicher kostengünstig zur Verfügung zu stellen

1. Technologische Trends

- Entwicklung der Performance von CPUs und DRAM



2. Lokalitätsprinzip

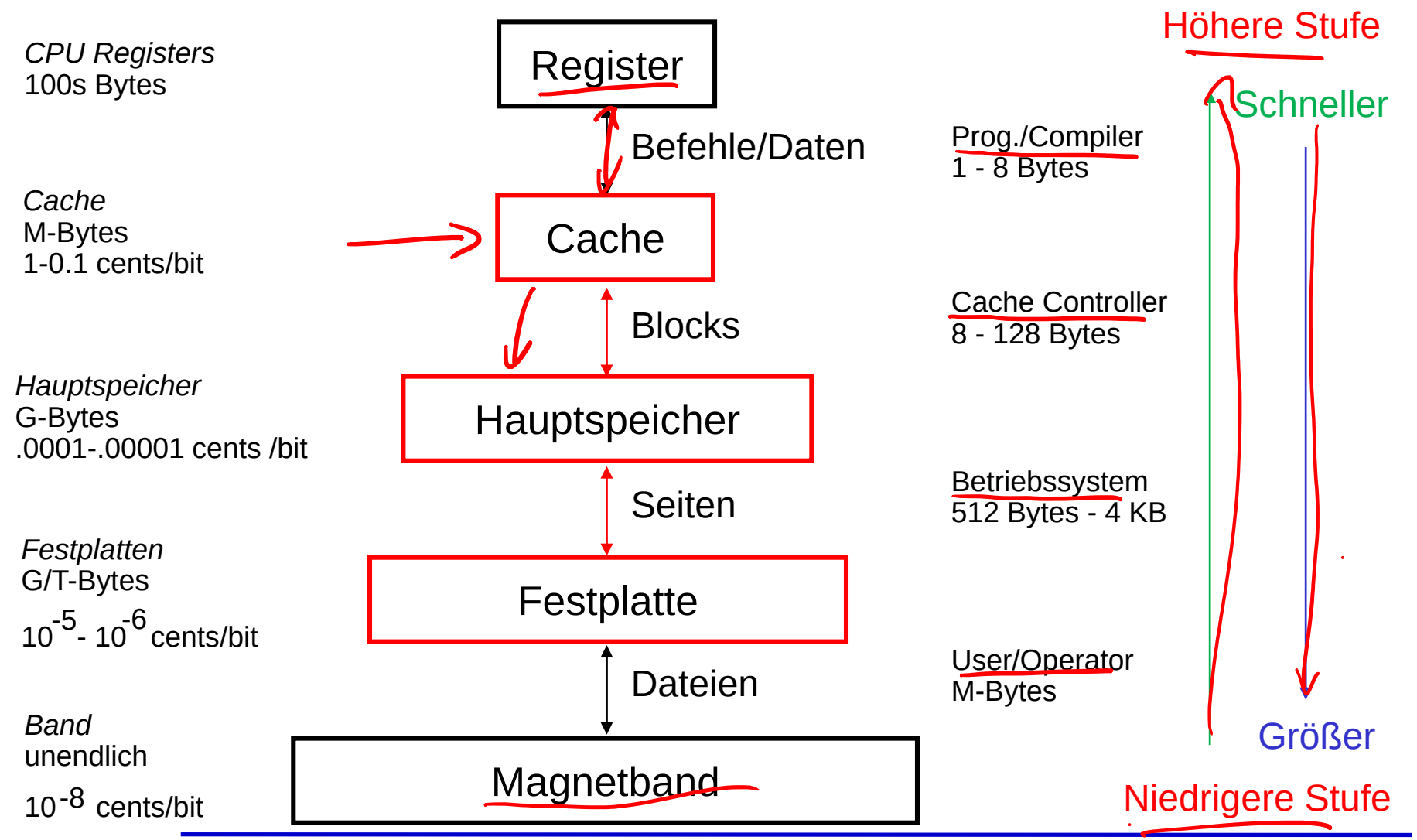
- Lokalitätsprinzip: Programme greifen in einem kleinen Zeitintervall auf einen relativ kleinen Teil des Adressraums zu

- Temporale Lokalität (Lokalität in der Zeit)
  - Wenn ein Zugriff auf eine Adresse erfolgt, wird auf diese Adresse mit „großer“ Wahrscheinlichkeit bald wieder zugegriffen
    - Abarbeitung von Schleifen

- Räumliche Lokalität (Lokalität im Raum)
  - Wenn ein Zugriff auf eine Adresse erfolgt, werden mit „großer“ Wahrscheinlichkeit bald Zugriffe auf in der Nähe liegende Adressen erfolgen
    - Verarbeitung von Array-Daten

- Aufgrund der Lokalität kann man Speichersysteme hierarchisch aufbauen
  - Obere Stufe der Hierarchie: schneller und teurer Speicher (wenig)
  - Untere Stufe der Hierarchie: langsamer und billiger Speicher (viel)

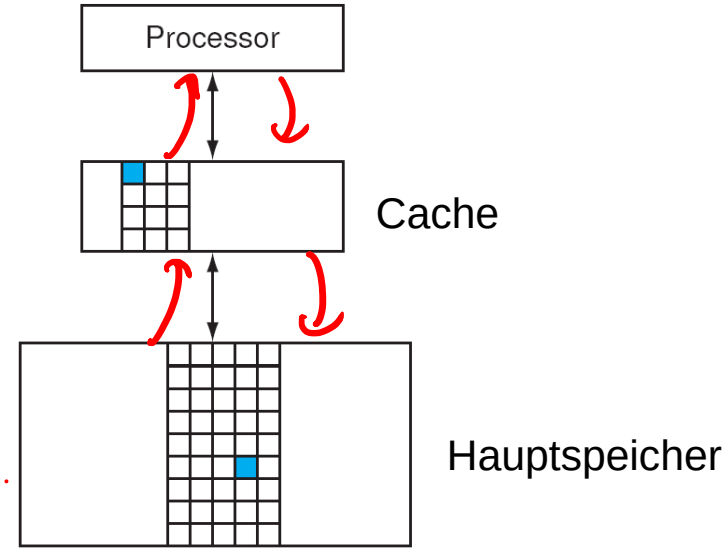
2. Übersicht – Speicherhierarchie



2. Speicherhierarchie

Funktionsprinzip

- Daten werden in bestimmten Einheiten zwischen den Ebenen der Speicherhierarchie übertragen
- Daten werden nur zwischen benachbarten Ebenen übertragen
- Ist eine Dateneinheit auf einer Ebene vorhanden, muss sie auch auf den tieferen Ebenen vorhanden sein

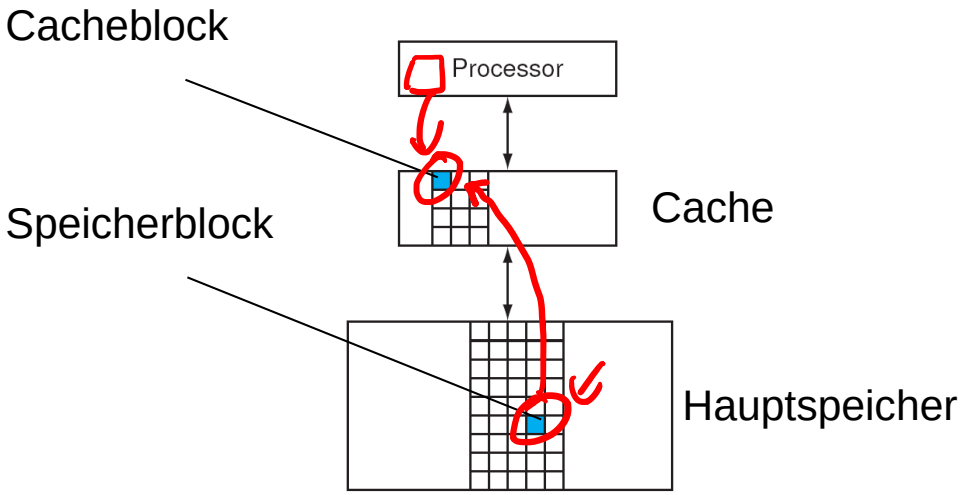


2. Terminologie

- Hit (Treffer): Daten befinden sich im Speicher der Ebene, auf die aktuell zugegriffen wird
  - Hit-Rate: Anteil der erfolgreichen Speicherzugriffe
    - So hoch, dass man immer von Miss-Rate spricht
  - Hit-Time: Zeit für einen erfolgreichen Zugriff
    - Zeit, um Hit festzustellen & Zeit für den eigentlichen Datenzugriff
- Miss (Fehlzugriff): Daten befinden sich nicht im Speicher der Ebene, sondern müssen erst aus unterhalb gelegener Ebene geholt werden
  - Miss-Rate: Anteil der Fehlzugriffe auf einer Ebene
    - Miss-Rate = 1 – Hit-Rate
  - Miss-Penalty (Miss-Strafe): Zeit für das Bereitstellen der Daten aus der unteren Ebene
    - *Zugriffszeit zur niedrigeren Speicherebene*
    - *Übertragungszeit zur höheren Speicherebene*
- Mittlere Speicherzugriffszeit = Hit-Time + Miss-Rate * Miss-Penalty
  (gemessen in bspw. ns or clocks)

2. Caches

Speicherhierarchie: Cache ↔ Hauptspeicher



Wichtige Fragen

- Auf welchen Cacheblock wird ein Speicherblock abgebildet?
- Wie stellt man fest, ob ein Datum im Cache ist, und falls ja, in welchem Cacheblock?
- Welcher Cacheblock wird bei einem Cache-Miss ersetzt?
- Was geschieht beim Schreiben eines Datums?

2. Cache mit direkter Abbildung

- Beim Cache mit direkter Abbildung (direct mapping) kann jeder Speicherblock nur an einer Stelle im Cache stehen

  Wo steht das Datum im Cache?

  - Einfachste Abbildung

    Cacheblockadresse = Speicherblockadresse *mod* (Anzahl Cacheblöcke)

    ↳ Wo im DRAM?

  - Die Cacheblockadresse wird auch als Index bezeichnet

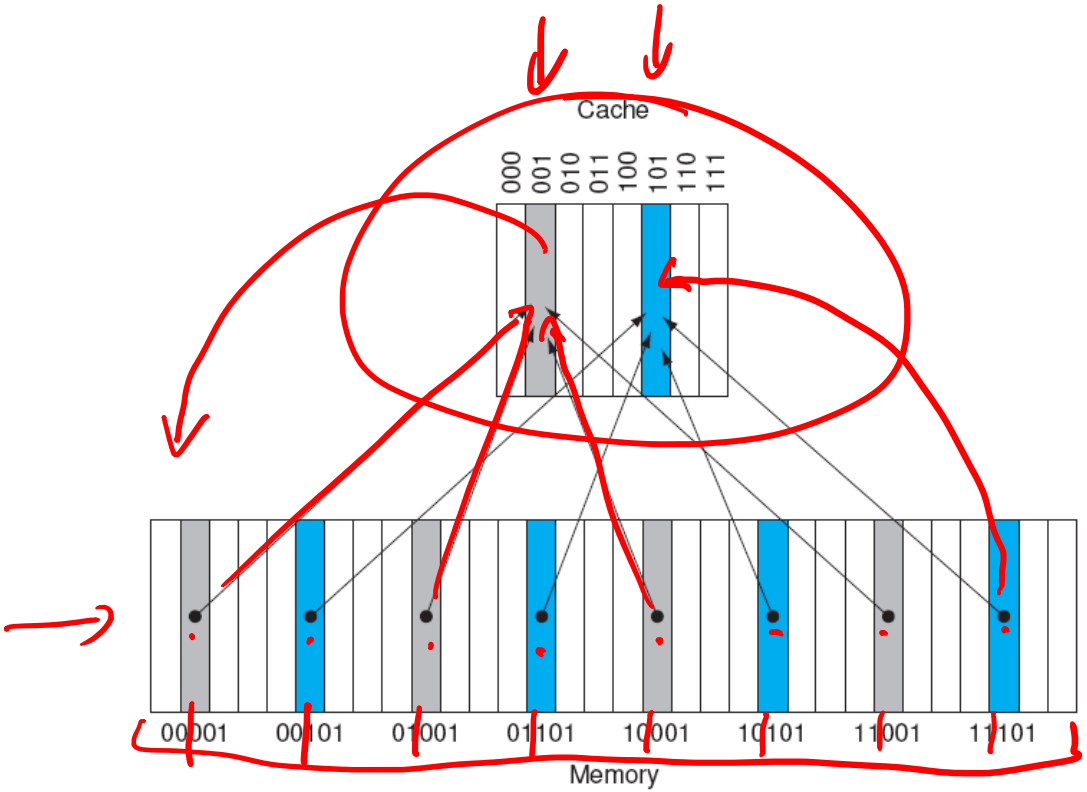
  - I.A. wählt man für die Anzahl der Cacheblöcke eine 2er-Potenz
    - Dadurch lässt sich die *mod* Operation besonders einfach realisieren
    - Für *mod* (2^n) muss man nur die n niederwertigsten Bits der Adresse nehmen

z. B. $n=3$ → 8 Cacheblöcke bzw. 8 Cacheeinträge

→ 3 niederwertigsten Bits d. Speicheradresse

2. Cache mit direkter Abbildung

- Beispiel
  - 8 Cacheblöcke; ein Block besteht aus einem Wort
    - Cacheblockadresse = Speicherblockadresse *mod* 8 = Speicherblockadresse[2:0]



2. Cache-Struktur

- Cache Tag
  - Der Index (Cacheblockadresse) kann nicht eindeutig einer Speicherblockadresse zugeordnet werden
  - Inhalt des Caches an dieser Position somit nicht eindeutig
  - Deshalb verwendet man die restlichen Bits der Speicherblockadresse als so genannten Tag
    - Zu jedem Block im Cache auch den entsprechenden Tag speichern
  - Auffinden von Daten durch Abgleich der Tags des gesuchten Blocks und des Cacheeintrages

- Valid Bit
  - Gibt an, ob ein Cacheeintrag gültig ist oder nicht
    - Am Anfang – und nach jedem Leeren des Caches – enthält der Cache keine gültigen Daten (Valid Bit wird auf 0 gesetzt)
    - Wenn der Cacheeintrag gültig ist, wird das Valid Bit auf 1 gesetzt

2. Cache-Zugriffe

- Beispiel
  - 5-Bit Adressen, Cache mit direkter Abbildung, 8 Cacheblöcke, ein Block besteht aus einem Wort → 32 Speicherblöcke

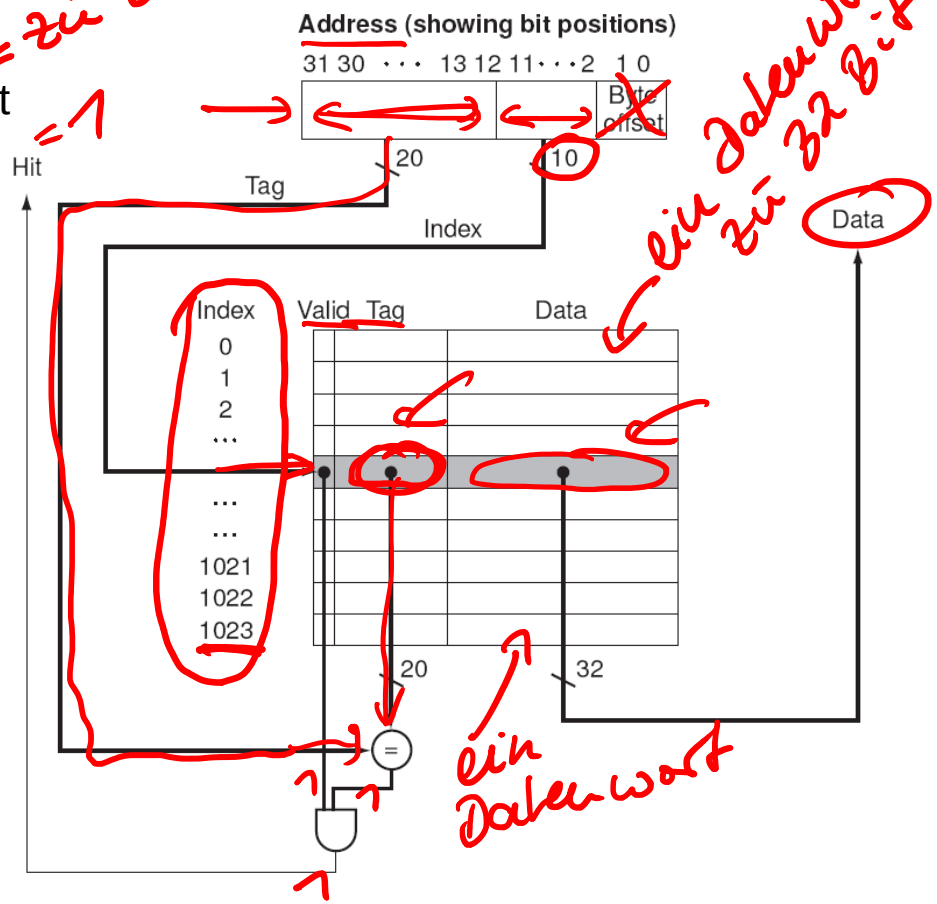
Zeit ↓

	Speicheradresse	Hit / Miss	Cacheblock
(1)	$22_{10} = 10110$	miss	10110 *mod* 8 = 110
(2)	$26_{10} = 11010$	miss	11010 *mod* 8 = 010
(3)	$22_{10} = 10110$	hit	10110 *mod* 8 = 110
(4)	$26_{10} = 11010$	hit	11010 *mod* 8 = 010
(5)	$16_{10} = 10000$	miss	10000 *mod* 8 = 000
(6)	$3_{10} = 00011$	miss	00011 *mod* 8 = 011
(7)	$16_{10} = 10000$	hit	10000 *mod* 8 = 000
(8)	$18_{10} = 10010$	miss	10010 *mod* 8 = 010

2. Cache-Struktur

Beispiel

4 KByte Cache mit direkter Abbildung
32-Bit Adressen,
1024 Cacheblöcke,
ein Block besteht aus einem Datenwort



Wie viele Speicherbits braucht man für die Implementierung eines solchen 4 KByte (=32 KBit) Cache?

→ $2^{10}*(32+20+1) = 53$ KBit !

data

tag

valid bit

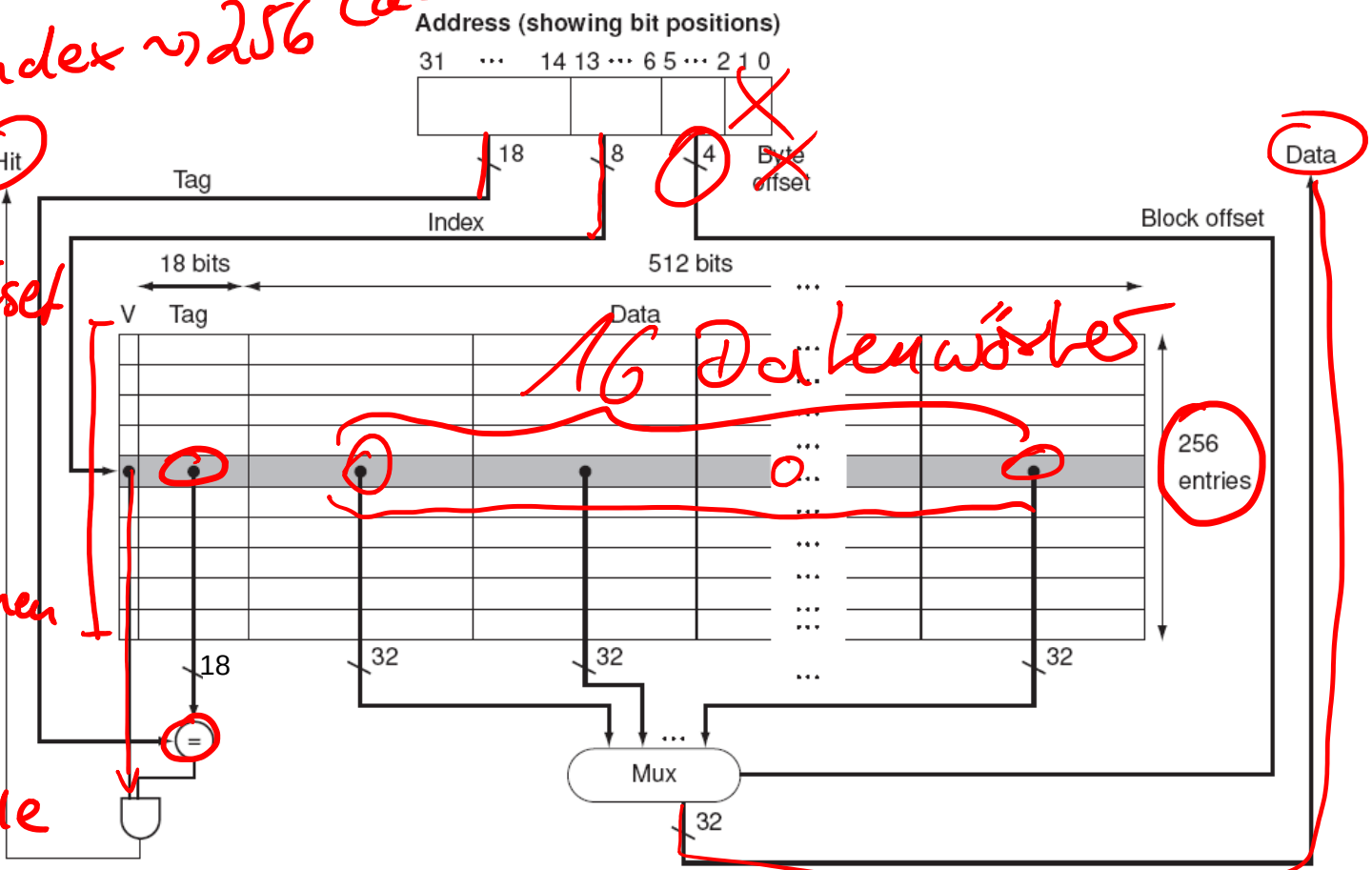
2. Cache mit direkter Abbildung

- Daten werden im Cache in größere Blöcke abgelegt
  - Cache-Eintrag besteht aus mehreren Datenwörtern
  - Nutzung der räumlichen Lokalität
  - Effizientere Datenübertragung

  - Jeder Block hat einen Tag
  - Die einzelnen Worte im Block haben aufsteigende Adressen
  - Die l-Bit Adresse wird aufgeteilt in
    - einen Tag,
    - einen n-Bit Index und
    - einen m-Bit Blockoffset
      - Der Blockoffset wählt unter den Worten eines Blocks aus

2. Cache mit direkter Abbildung

- Beispiel
  - $l = 32$ = 32 Bit-Adresse, 32 Bit Daten
  - $n = 8$ = Index → 256 Cachezeilen
  - $m = 4$



Blockoffset

16 Datenwörter

$16 = 2^4$ vers. Positionen innerhalb einer Cachezeile identifizieren

2. Cache mit direkter Abbildung

- Mehrere Worte pro Block (Multiword Cache Block)
  - Vorteile
    - Durch die Nutzung räumlicher Lokalität sinkt die Miss-Rate
    - Effizientere Speicherung, d.h. weniger Overhead durch Tags und Valid Bits
    - Effizienterer Datentransfer

1 Tag pro Cachezeile, aber >1 Wort pro Cachezeile

  - Nachteile
    - Wachsende Blockgröße → weniger Cacheblöcke (bei konstanter Cachegröße)
      - Blöcke werden immer öfter ersetzt, so dass die Lokalität nur schlecht ausgenutzt werden kann → Miss-Rate steigt wieder
    - Größere Miss-Penalty
      - Bei einem Cache-Miss müssen mehr Daten aus dem Speicher geladen werden

- Weitere Maßnahmen zur Reduktion der Miss-Rate
  - Größerer Cache (teuer!)
  - Assoziativer Cache

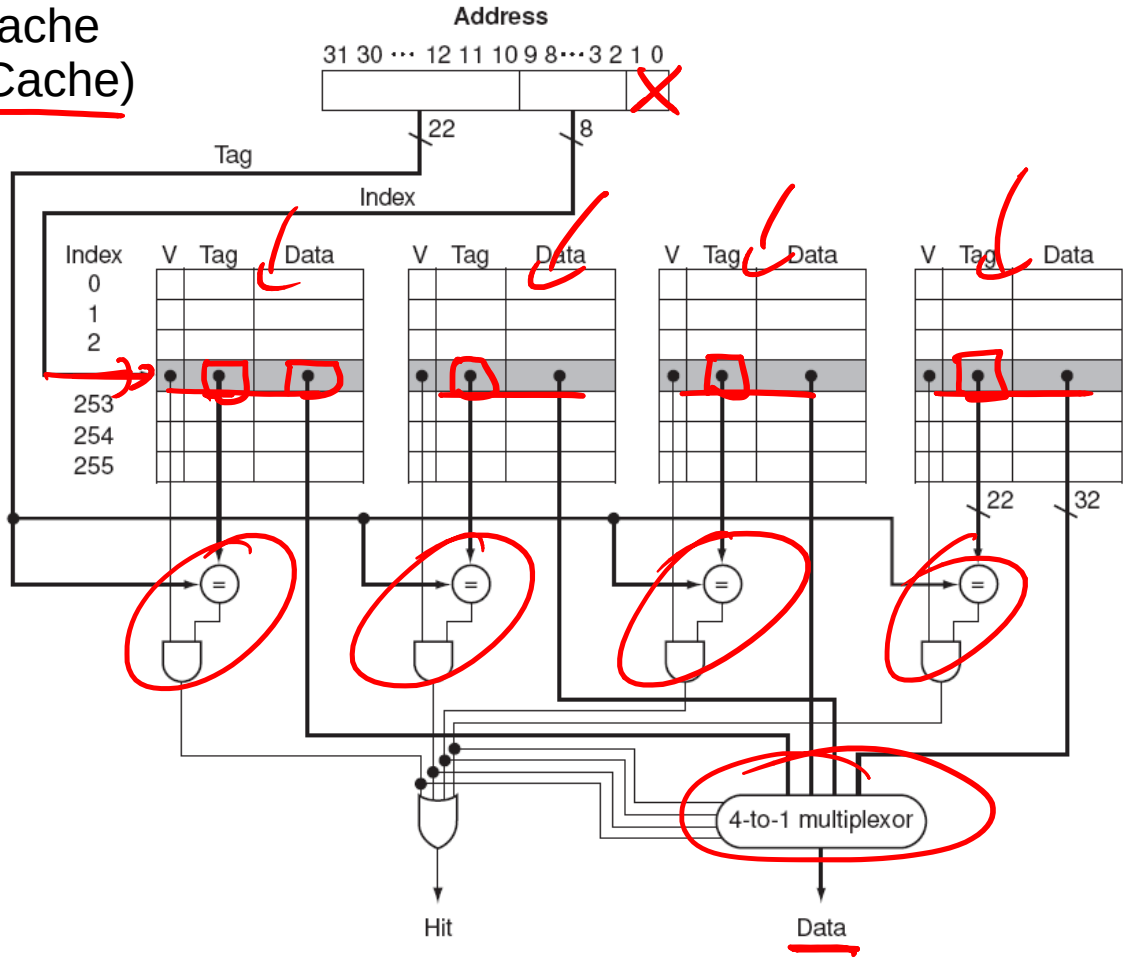
2. Assoziativer Cache

- Assoziativer Cache hat für jeden Cacheindex k Einträge (Blöcke)
  - Satz: Menge von Blöcken, die mit einem Indexwert referenziert werden
- Cache-Zugriff
  - Auswahl des Satzes mit dem Index der Adresse
  - Parallel dazu Vergleich des Tag der Adresse mit allen k Tags der Cacheblöcke in diesem Satz
  - bei 2^n Cacheblöcken:
    - $k = 1$: Cache mit direkter Abbildung
    - $k > 1$: k-fach satzassoziativer Cache (k-Way Set Associative Cache), teilassoziativer Cache
    - $k = 2^n$: voll assoziativer Cache (Full Associative Cache)
  - Vorteil
    - Miss Rate wird reduziert
  - Nachteile
    - Hit Time wird etwas vergrößert
    - Bei großem k wird der Hardwareaufwand beträchtlich

2. Assoziativer Cache

- Beispiel : $k = 4$

4-fach satzassoziativer Cache
mit m=0 (kein Multiword Cache)



eher im Vgl. zu Folie 14 zu sehen, nicht zu Folie 16

2. Assoziativer Cache

- Beispiel: $n = 3$, $m = 0$

One-way set associative (direct mapped)

Block	Tag	Data
0		
1		
2		
3		
4		
5		
6		
7		

Two-way set associative

Set	Tag	Data	Tag	Data
0				
1				
2				
3				

Four-way set associative

Set	Tag	Data	Tag	Data	Tag	Data	Tag	Data
0								
1								

Eight-way set associative (fully associative)

Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data

2. Verdrängungsstrategien

- Welcher Block wird verdrängt, wenn ein neuer Block geladen werden muss und keine freien Cacheblöcke mehr vorhanden sind?

- Cache mit direkter Abbildung
  - Keine Auswahl, da jeder Speicherblock auf genau einen Cacheblock abgebildet wird

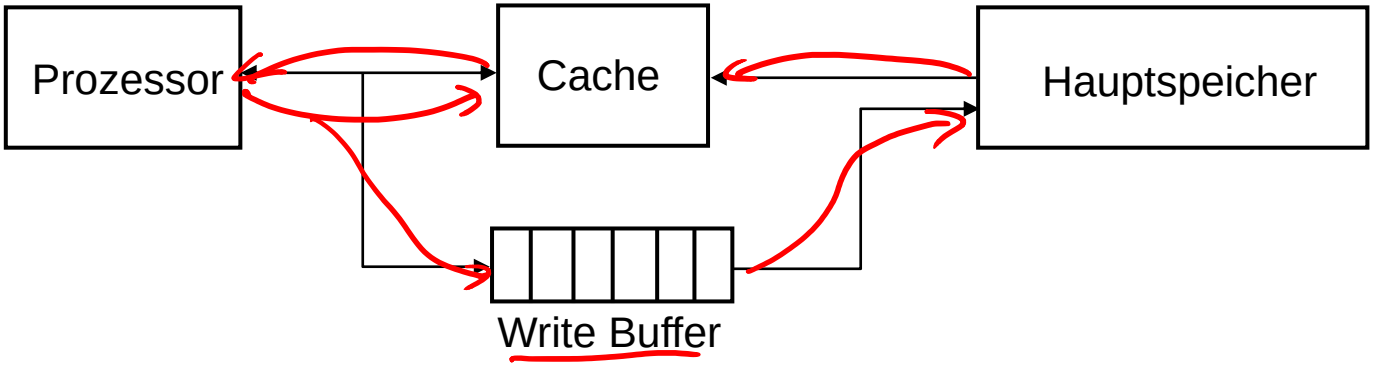
- Assoziativer Cache
  - Zufällige Auswahl eines Blocks
    - Einfach in Hardware zu implementieren
  - Least Recently Used: der am längsten nicht benutzte Block wird ersetzt
    - Zugriffszeiten auf die Blöcke müssen gespeichert werden
    - Aufwendig in Hardware zu realisieren, wird typischerweise bis $k = 4$ gemacht

2. Schreibtechniken

- Durchgängiges Schreiben (write-through)
  - Worte werden in den Cache und gleichzeitig in den Hauptspeicher geschrieben
    - Hauptspeicher und Cache immer konsistent, aber jede Schreib-Operation greift immer auch auf den (langsamen) Hauptspeicher zu
  - Modifikation der Daten erfolgt immer nur im Cache
    - Bei einem Write-Miss wird zuerst der Block in den Cache geladen und dann das Wort in den Cache und gleichzeitig in den Hauptspeicher geschrieben
  - Entlastung des Prozessor durch die Verwendung eines Pufferspeichers (write buffer)
    - Schreiben der Daten in Cache und (schnellen) Pufferspeicher, danach kann der Prozessor sofort weiterarbeiten
    - Ist die Rate, mit der der Prozessor Schreibinstruktionen ausführt, kleiner als die mittlere DRAM-Schreibzykluszeit, funktioniert write-through mit write buffer gut
    - Wenn nicht, füllt sich der Pufferspeicher und der Prozessor muss anhalten und warten, bis im Pufferspeicher wieder Platz ist

2. Schreibtechniken

- Durchgängiges Schreiben (write-through & write buffer)



2. Schreibtechniken

Zurückkopieren (write-back)

- Beim Schreiben wird das Wort nur in den Cache geschrieben
  - Hauptspeicher und Cache sind inkonsistent
- Erst wenn der Cacheblock ersetzt werden muss, wird er in den Hauptspeicher zurückgeschrieben
  - Ein Dirty Bit gibt an, ob der Block verändert wurde oder nicht
    - Dirty Bit wird bei jedem Schreibzugriff auf den Block gesetzt
    - Beim Ersetzen des Blocks wird dieser nur dann in den Hauptspeicher zurückgeschrieben, wenn das Dirty Bit gesetzt ist
- Vorteil
  - Wenn die Schreibrate hoch ist, höhere Performance als write-through
- Nachteil
  - Aufwendigere Steuerung als write-through

2. Weitere Maßnahmen zur Cache-Optimierung

- Reduktion der Miss-Rate
  - Cacheorganisation
    - Größere Blöcke
    - Größere Caches
    - Höhere Assoziativität
  - Compileroptimierungen
    - Instruktionen: Umordnen der Instruktionen, um Instruction Misses zu reduzieren
    - Daten: array merging, loop interchange, loop fusion, blocking, ...
- Reduktion der Miss-Penalty
  - Multi-level Cache
    - Kleiner 1st Level Cache optimiert auf Hit-Time (geringe Assoziativität, kleine Blöcke)
    - Großer 2nd Level Cache optimiert auf Miss-Rate (höhere Assoziativität, große Blöcke)
  - Victim Cache
    - Zusätzlicher "Cache", der kürzlich ersetzte Cacheblöcke ("victims") speichert

2. Weitere Maßnahmen zur Cache-Optimierung

- Reduktion der Miss-Penalty und der Miss-Rate
  - Blöcke werden auf Verdacht in Register, Caches oder so genannten Prefetch-Puffer geladen
    - Macht nur Sinn, wenn das Speichersystem freie Bandbreite zur Verfügung hat
  - Hardware Prefetching
    - Prefetching von Instruktionen

      Bsp.: Alpha 21064 lädt zwei Blöcke bei einem Instruction Miss, der zweite Block kommt in einen separaten Puffer („Stream Buffer“); beim nächsten Instruction Miss wird zuerst im „Stream Buffer“ nach benötigtem Block gesucht

    - Prefetching von Daten

      Bsp.: Stream Buffer, die dauernd mit den Blöcken der nächsten Adressen nachgeladen werden
  - Software Prefetching
    - Compiler fügt spezielle prefetch-Instruktionen ein

3. Zusammenfassung – 4 wesentliche Fragen

1. Wo wird ein Block platziert?
   - direkt abgebildet
   - teilassoziativ
   - vollassoziativ
2. Wie wird ein Block gefunden?
   - direkt abgebildet: Index
   - teilassoziativ: Index & Suche unter den Blöcken (paralleler Vergleich)
   - vollassoziativ: Suche unter allen Blöcken bzw. Tabelle mit Abbildungen
3. Welcher Block wird bei einem Miss ersetzt?
   - Zufällig bestimmter Block
   - „Least Recently Used“-Strategie
4. Was geschieht beim Schreiben?
   - Durchgängiges Schreiben
   - Zurückkopieren