ulm university universität ulm

Grundlagen der Rechnerarchitektur

[CS3100.010]

Wintersemester 2014/15

Heiko Falk

Institut für Eingebettete Systeme/Echtzeitsysteme
Ingenieurwissenschaften und Informatik
Universität Ulm

ulm university universität ulm

Kapitel 7

Speicher-Hardware

Inhalte der Vorlesung

1. Einführung
2. Kombinatorische Logik
3. Sequentielle Logik
4. Technologische Grundlagen
5. Rechnerarithmetik
6. Grundlagen der Rechnerarchitektur
7. **Speicher-Hardware**
8. Ein-/Ausgabe

Inhalte des Kapitels (1)

7. Speicher-Hardware

- Einleitung: Speicherhierarchien
  - *„Memory Wall“* Problem
  - Entwurf von Speicherhierarchien
- SRAM
  - Aufbau der SRAM-Zelle
  - Schreib- und Lesevorgänge
  - Aufbau von SRAM-Speichern
  - Verbesserungen
- DRAM
  - Aufbau der DRAM-Zelle
  - Matrixaufbau von DRAM-Speichern
  - Verbesserungen
- ...

Inhalte des Kapitels (2)

7. Speicher-Hardware

- ...
- *Translation Look-Aside Buffer* (TLBs)
  - Motivation: *Paging* & Segmentierung
  - Organisationsformen: *direct mapping*, *set-associative*, *associative mapping*
- *Caches*
  - Organisation von *Caches*
  - Schreibverfahren: *Write-Through*, *Copy-Back*
  - Virtuelle vs. Reale *Caches*
  - Austauschverfahren: *Random*, FIFO, *Clock*, NRU, LRU
  - *Cache*-Kohärenz
    - Gründe für Inkohärenzen
    - *Snooping*- & *Directory*-Protokolle
- ...

Inhalte des Kapitels (3)

7. Speicher-Hardware

- ...
- Massenspeicher
  - RAID-*Arrays*
  - CD-ROMs
  - DVDs
  - Blu-rays
  - Bandlaufwerke: QIC, DAT, LTO

Einführung

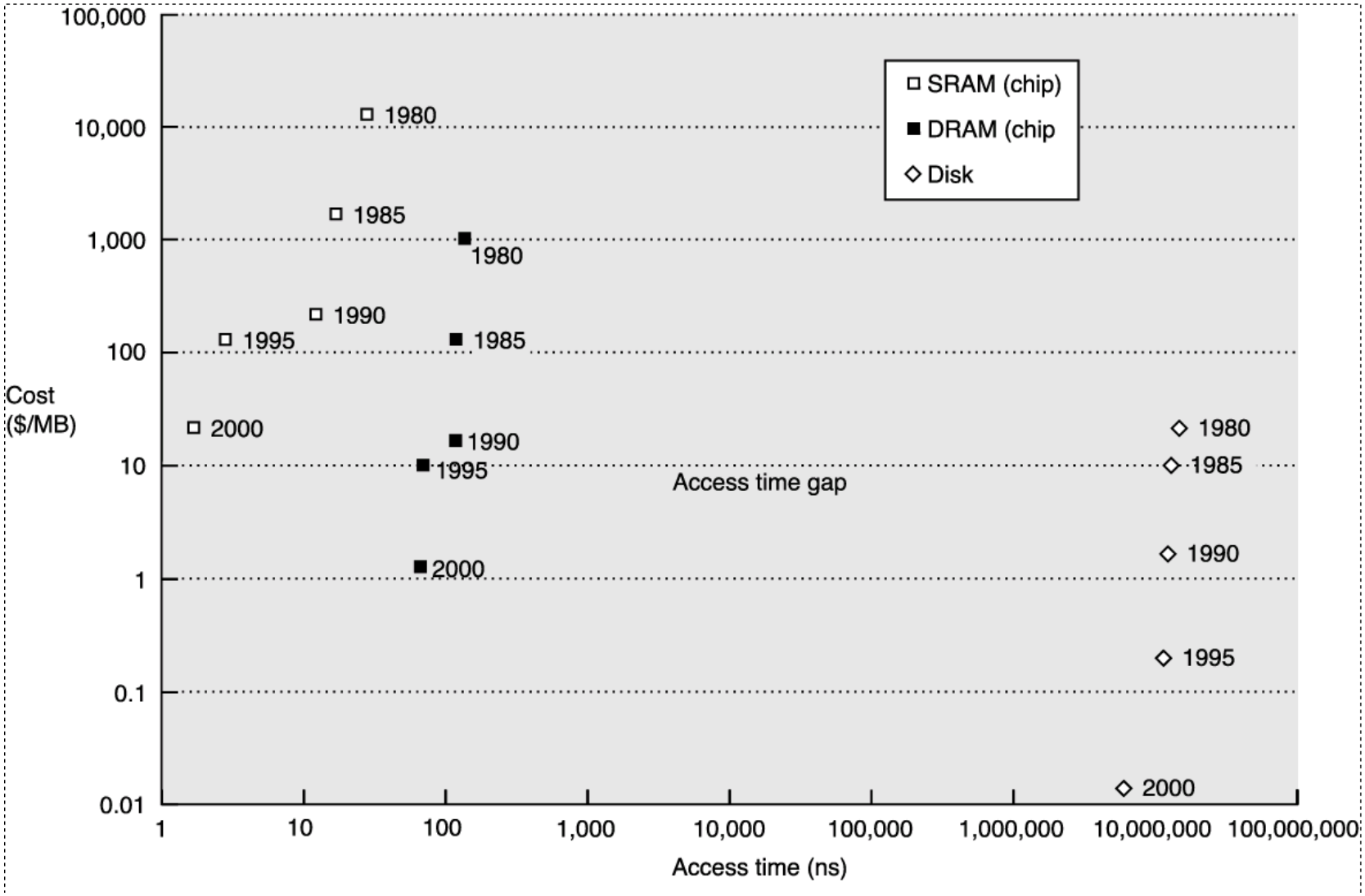
Technologien zum Speichern von Daten

- Modifikation von Strukturen: Lochkarte, Schallplatte
- Rückkopplung: Flip-Flops, SRAM
- Elektrische Ladungen: Kondensator, DRAM
- Magnetismus: Magnetkernspeicher, Magnetband, Diskette, Festplatte
- Optik: Bar-Codes, CD-ROM, DVD

Vergleichskriterien

- Kapazität
- Geschwindigkeit
- Speicherdichte
- Energiebedarf
- Robustheit
- Kosten

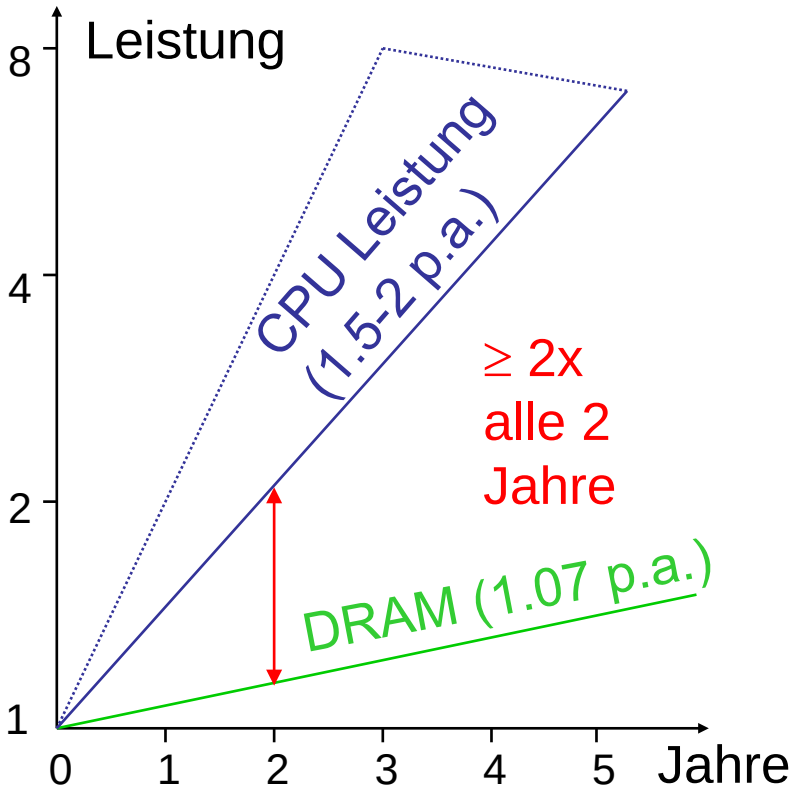
Die Realität: Kosten/Mbyte und Zugriffszeiten



[Hennessy/Patterson, Computer Architecture, 3. Auflage]
© Elsevier Science (USA), 2003. All rights reserved

Trend der Leistungen von DRAM

Die Performanzlücke zwischen Prozessoren und DRAM wächst

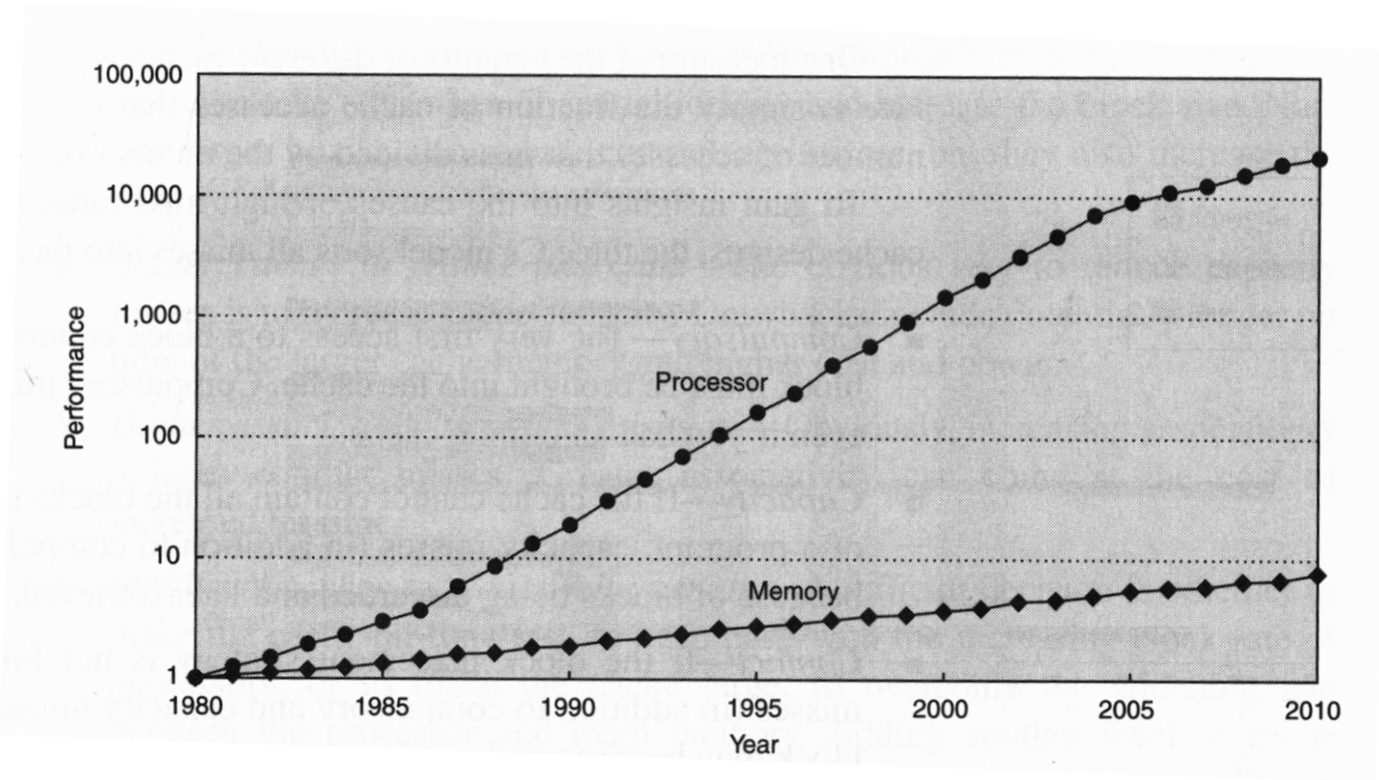


Ähnliche Probleme auch für Eingebettete Systeme

- ☞ In Zukunft: Speicherzugriffszeiten >> Prozessorzykluszeiten
- ☞ *„Memory Wall“* Problem

*[**P. Machanik,** Approaches to Addressing the Memory Wall, TR Univ. Brisbane, Nov. 2002]*

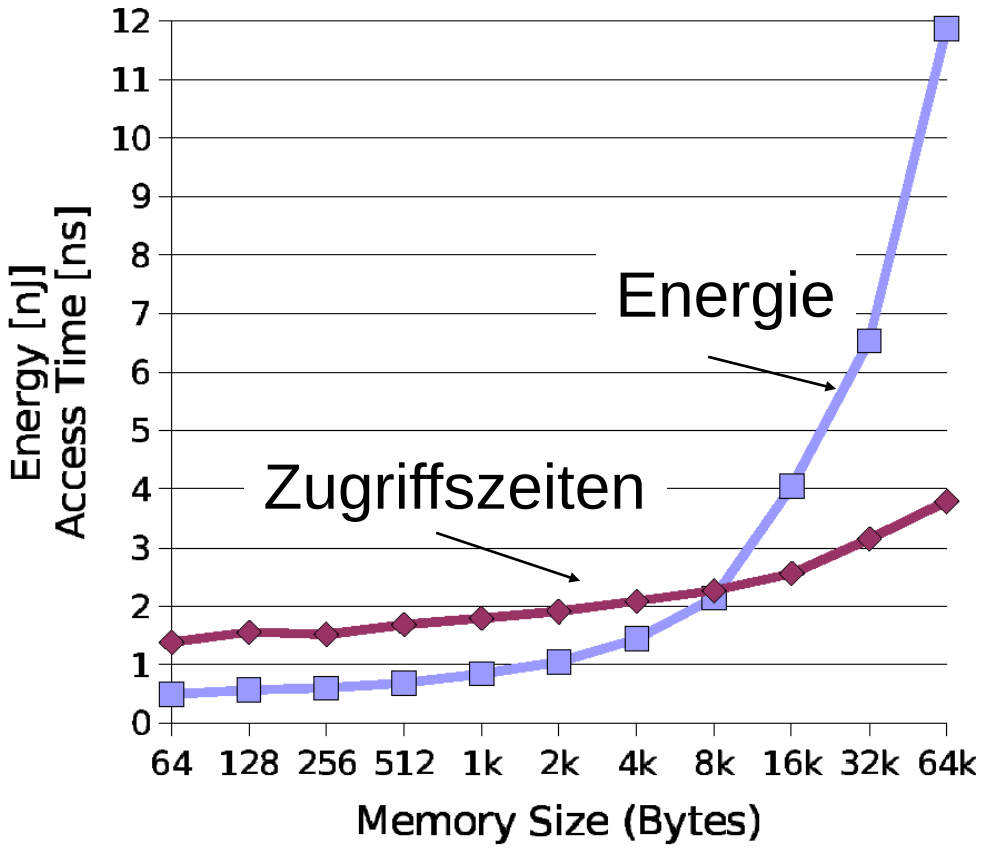
Memory Wall



© Elsevier Science

- Früher (1980er): Prozessoren ohne *Cache* entwickelt
- Seit 2000: mindestens 2 *Cache*-Ebenen
- Problem wird sich weiter verschärfen
- Speicherhierarchien sind Versuch zu kompensieren

Abhängigkeit von der Speichergröße



Anwendungen werden größer und größer…

Speicherhierarchie

– Große Speicher sind langsam
– Anwendung verhalten sich üblicherweise lokal

☞ Häufig benötigte Speicherinhalte in kleinen Speichern, seltener benötigte Inhalte in großen Speichern ablegen!
☞ Einführung einer „Speicherhierarchie“
☞ *Illusion* eines **großen Speichers** mit (durchschnittlich) **kleinen Zugriffszeiten**

– Enge Grenze für die Zugriffszeit wird selten garantiert
☞ Spezielle Überlegungen bei Realzeitsystemen

Entwurf von Speicherhierarchien

„Speicherhunger“ von Applikationen/Entwicklern/…

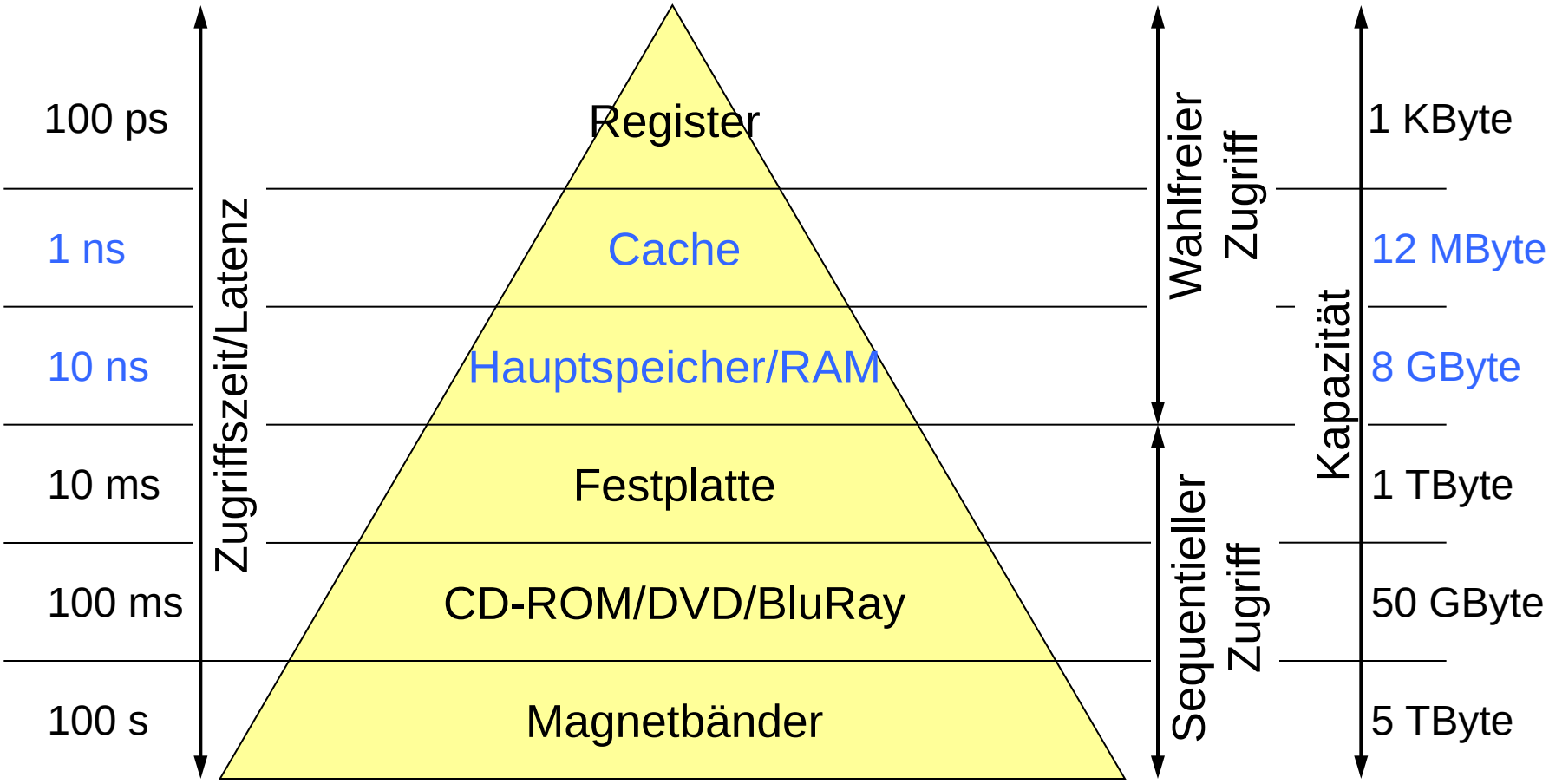
„Ideally one would desire an indefinitely large memory capacity such that any particular … word would be immediately available… We are … forced to recognize the possibility of constructing a hierarchy of memories, each of which has greater capacity than the preceding but which is less quickly accessible“

*[**A. W. Burks, H. H. Goldstine, J. v. Neumann,** Preliminary Discussion of the Logical Design of an Electronic Computing Instrument, 1946]*

Größenordnungen von Kapazitäten und Zugriffszeiten

Speicherhierarchie

– Bis zu sechs Ebenen in modernen Systemen unterscheidbar



Entwurfsziele

- Möglichst großer Speicher
- Möglichst große Geschwindigkeit
  - Geringe Zeit bis zur Verfügbarkeit eines Speicherinhalts (kleine *latency*)
  - Hoher Durchsatz (großer *throughput*)
- Persistente (nicht-flüchtige Speicherung)
- Geringe Energieaufnahme
- Hohe Datensicherheit
- Geringer Preis
- Klein

Maßeinheiten für Speicherkapazitäten

Zweierpotenzen

1 KByte = 1 KB = 1024 Bytes = 2^{10} Bytes

1 MByte = 1 MB = 1024 KB = 2^{20} Bytes = 1.048.576 Bytes

1 GByte = 1 GB = 1024 MB = 2^{30} Bytes = 1.073.741.824 Bytes

1 TByte = 1 TB = 1024 GB = 2^{40} Bytes = 1.099.511.627.776 Bytes

Kilo
Mega
Giga
Tera
Peta
Exa
Zetta
Yotta

elektrisch

magnetisch

im Netz

Begriffe (1)

RAM *(Random Access Memory)*

- Wahlfreier Zugriff
- Zugriffe dauern alle etwa gleich lang
  - Vgl. Plattenzugriffe, Bandzugriffe
- Lesende und schreibende Zugriffe möglich

SRAM *(Static RAM)*

- Statischer RAM-Speicher

DRAM *(Dynamic RAM)*

- Dynamischer RAM-Speicher

Begriffe (2)

ROM *(Read Only Memory)*
- Nur lesender Zugriff möglich

PROM *(Programmable Read only Memory)*
- Programmierbarer ROM-Speicher (einmalig beschreibbar)
- z.B. maskenprogrammiert bei Chip-Herstellung

EPROM *(Erasable PROM)*
- Mit UV-Licht löschbarer PROM-Speicher
- Elektrisch programmierbar (durch zusätzliche Steuerleitungen)

EEPROM *(Electrically Erasable PROM)*
- Elektrisch lösch- und beschreibbarer Speicher

Begriffe (3)

Flash-Speicher
- Eigenname
- Wie EEPROM, jedoch höhere Packungsdichte

NV-RAM *(Non-Volatile RAM)*
- Normalerweise SRAM mit eingebautem EEPROM zur Rettung der Daten bei Stromverlust

Batteriegepuffertes RAM
- Integrierte Batterie sorgt für Erhalt der Daten

Begriffe (4)

Einteilung des Speichers in

- Flüchtigen Speicher: SRAM, DRAM, SDRAM, ...
  - Informationen gehen nach Ausschalten der Versorgungsspannung verloren!
- Nichtflüchtige Speicher: PROM, EPROM, EEPROM, Flash, ...
  - Informationen bleiben auch ohne Versorgungsspannung über längere Zeit (typischerweise einige Jahre) erhalten!

Geschwindigkeit

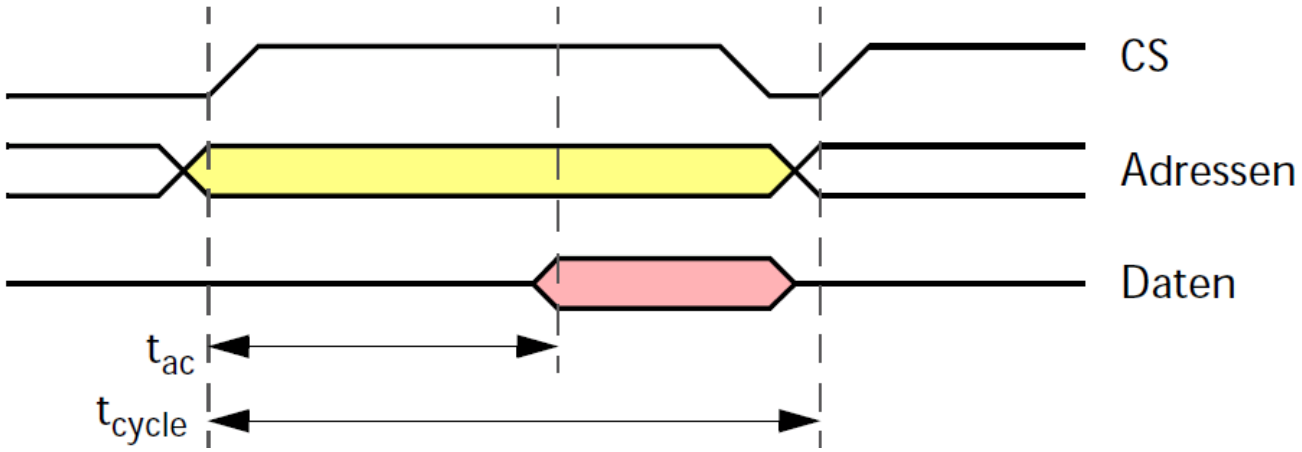
Zugriffszeit t_{ac}

– Zeitspanne vom Anlegen einer Speicheradresse bis zum Vorliegen der Daten (Lesezugriff)

Zykluszeit t_{cycle}

– Zeitspanne vom Anlegen einer Adresse bis zum möglichen Anlegen der nächsten Adresse

Beispiel für Zeitdiagramm



Roter Faden

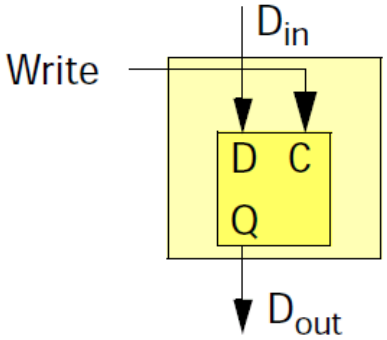
7. Speicher-Hardware

- Einleitung: Speicherhierarchien
  - *„Memory Wall“* Problem
  - Entwurf von Speicherhierarchien
- SRAM
- DRAM
- *Translation Look-Aside Buffer* (TLBs)
- *Caches*
- Massenspeicher

SRAM

Static Random Access Memory (SRAM)

– Auf Flip-Flop Prinzip basierender Speicher



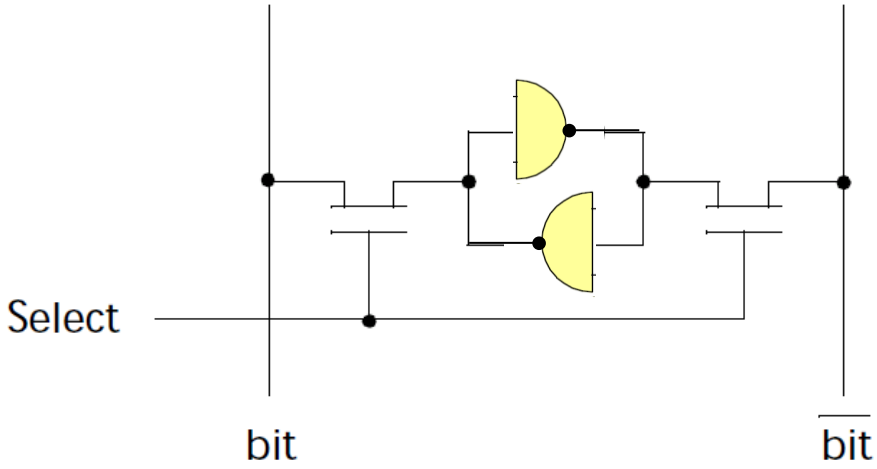
– Bei Write = 1 wird Information von D_{in} gespeichert und steht am Ausgang D_{out} zur Verfügung

– Information bleibt gespeichert, auch wenn sich D_{in} bei Write = 0 wieder ändern sollte

– SRAMs basieren jedoch nicht auf vollständigen D-Flip-Flops, sondern auf einfacheren SRAM-Zellen

SRAM-Zelle (1)

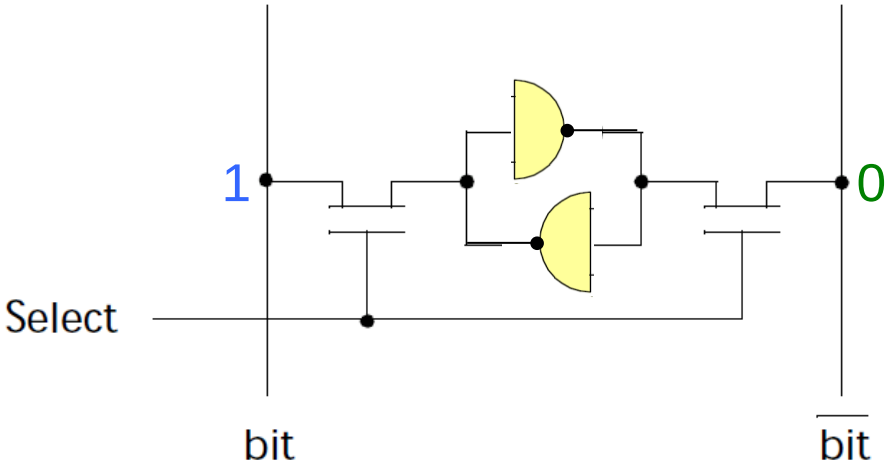
SRAM-Zelle mit zwei Invertern



- Inverterausgänge werden stabil (Flip-Flop, zwei stabile Zustände)
- Mit Select-Leitung kann Flip-Flop mit Leitungen bit und $\overline{bit}$ verbunden werden

SRAM-Zelle (2)

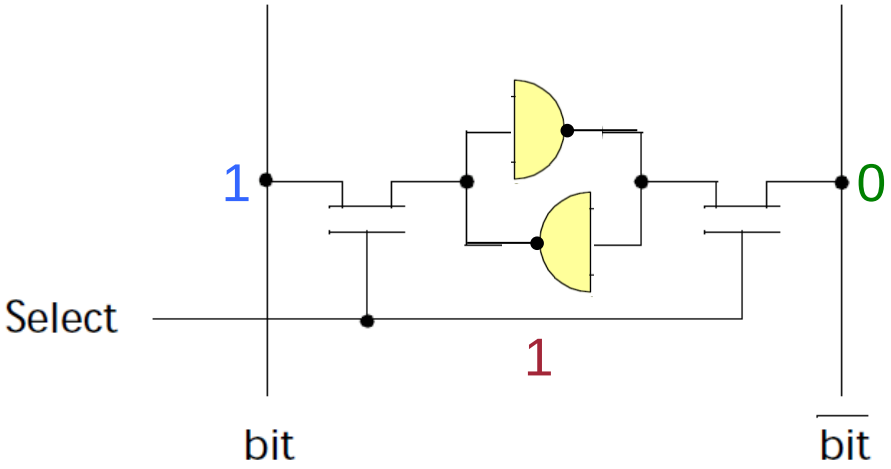
Schreiben in Zelle



– Leitungen bit und $\overline{bit}$ werden verschieden beschaltet: (0,1) oder (1,0)

SRAM-Zelle (3)

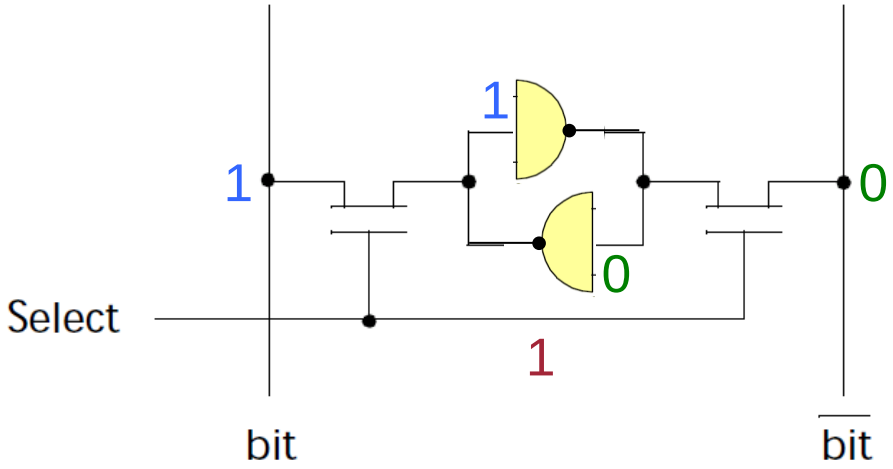
Schreiben in Zelle



- Leitungen bit und $\overline{bit}$ werden verschieden beschaltet: (0,1) oder (1,0)
- Select-Leitung wird auf 1 gesetzt

SRAM-Zelle (4)

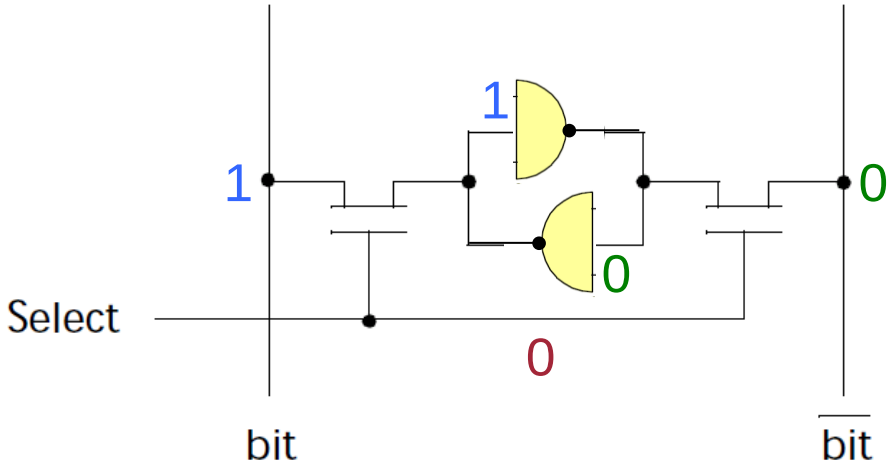
Schreiben in Zelle



- Leitungen bit und $\overline{bit}$ werden verschieden beschaltet: (0,1) oder (1,0)
- Select-Leitung wird auf 1 gesetzt
- Inverter schwingen in den einen bzw. anderen stabilen Zustand ein

SRAM-Zelle (5)

Schreiben in Zelle

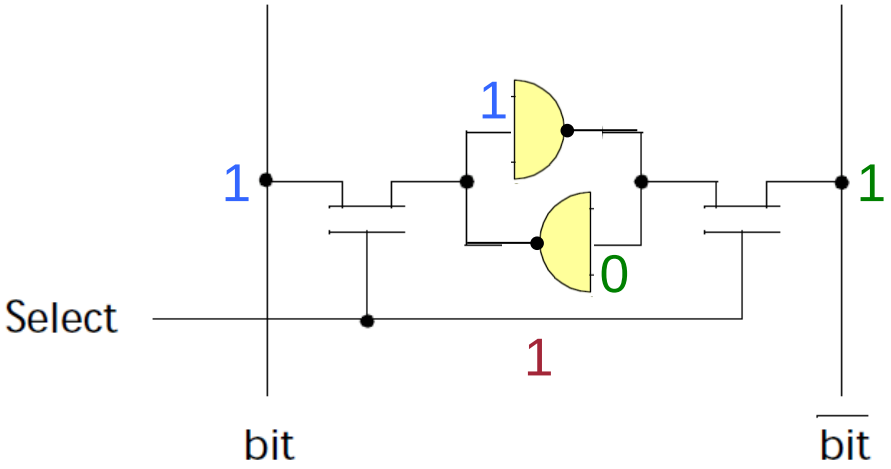


- Leitungen bit und $\overline{bit}$ werden verschieden beschaltet: (0,1) oder (1,0)
- Select-Leitung wird auf 1 gesetzt
- Inverter schwingen in den einen bzw. anderen stabilen Zustand ein
- Rücknahme der Select-Leitung nach Einschwingen

☞ Zustand bleibt erhalten, solange Betriebsspannung anliegt

SRAM-Zelle (6)

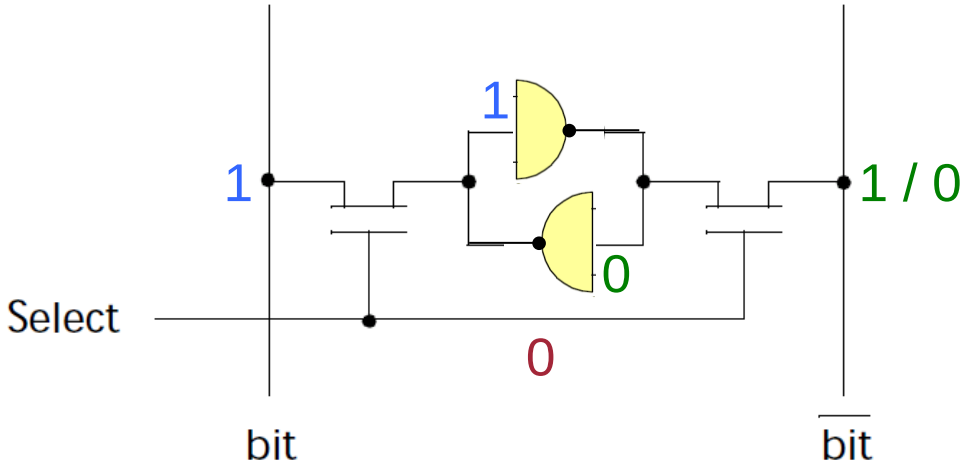
Lesen von Zelle



– Leitungen bit und $\overline{\text{bit}}$ werden mit 1 beschaltet, Select wird kurz mit 1 beschaltet

SRAM-Zelle (7)

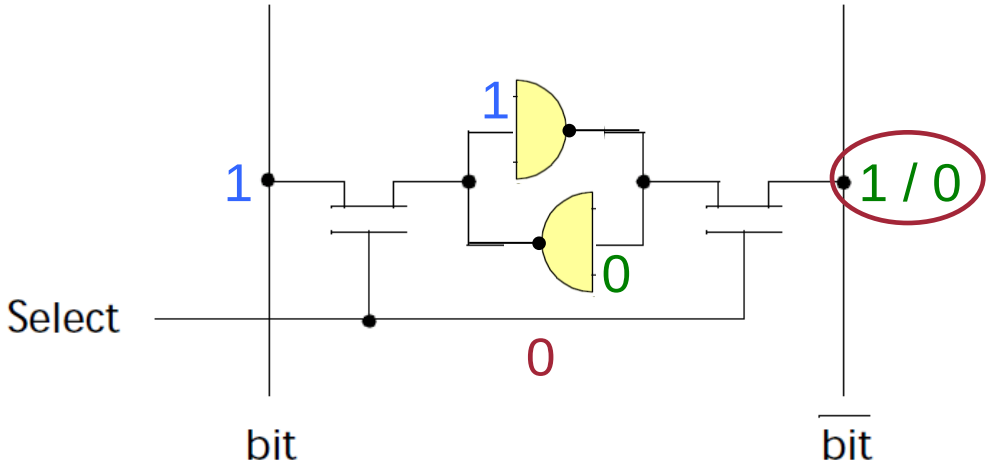
Lesen von Zelle



– Leitungen bit und $\overline{bit}$ werden mit 1 beschaltet, Select wird kurz mit 1 beschaltet
– Inverter versuchen, eine der beiden Leitungen auf Masse zu ziehen

SRAM-Zelle (8)

Lesen von Zelle



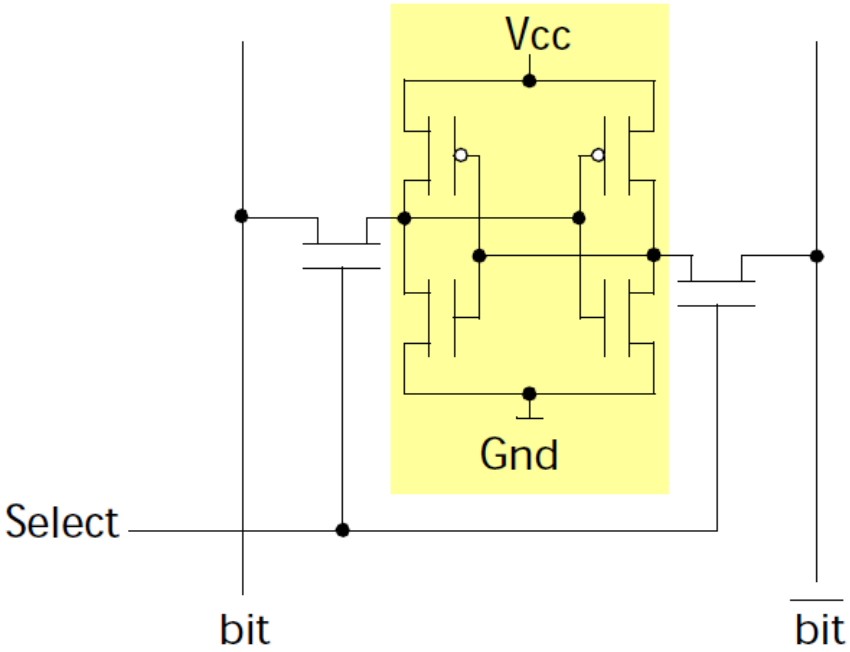
- Leitungen bit und $\overline{bit}$ werden mit 1 beschaltet, Select kurz mit 1
- Inverter versuchen, eine der beiden Leitungen auf Masse zu ziehen
- Geringer Spannungsabfall kann registriert werden

☞ 0 oder 1 wird erkannt

  - Aufwändige Verstärkerschaltung notwendig

SRAM-Zelle (9)

Typischer Aufbau mit 6 Transistoren

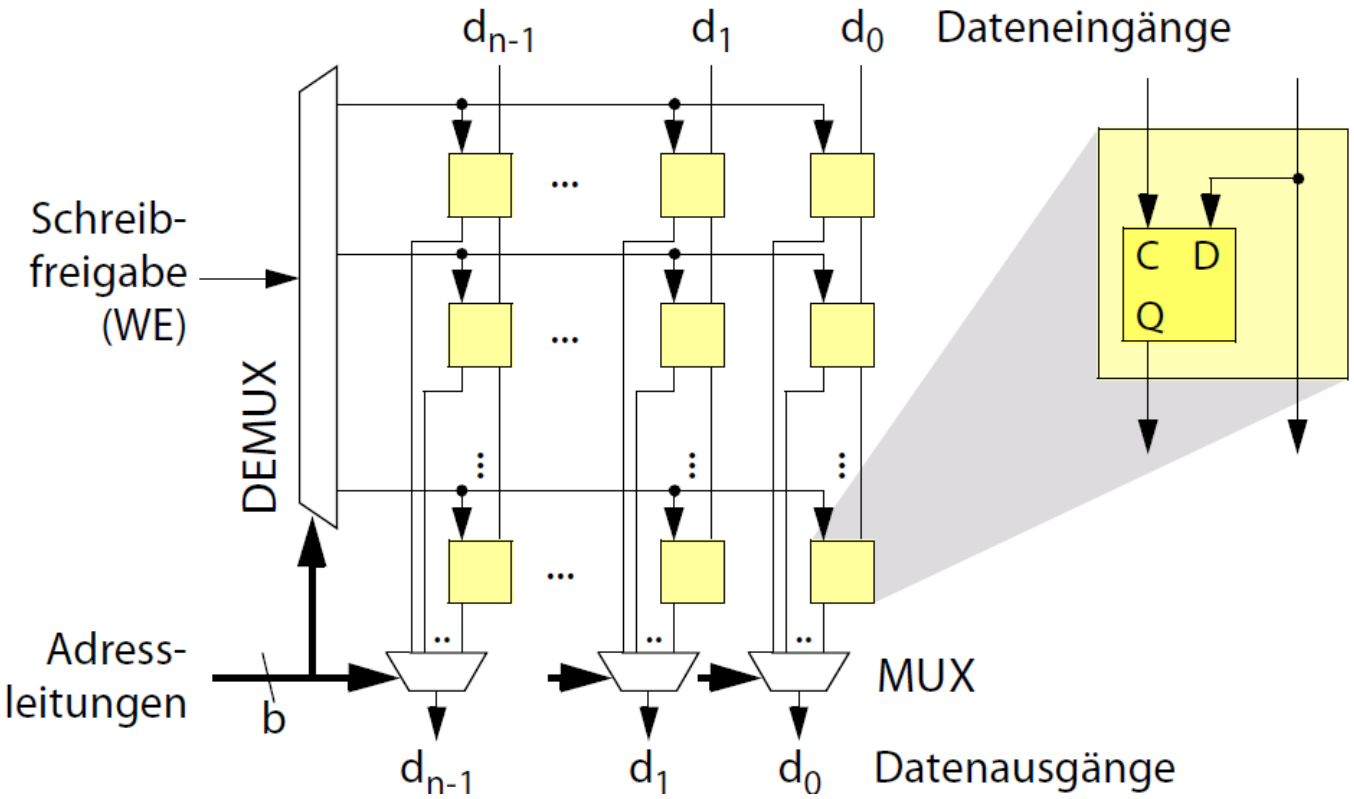


– Realisierung der Inverter mit je zwei Transistoren

Überprüfung der Schaltung sei dem Leser überlassen

SRAM-Speicher (1)

Aufbau eines SRAM-Speichers mit 2^b Zellen



SRAM-Speicher (2)

Chip-Aufbau

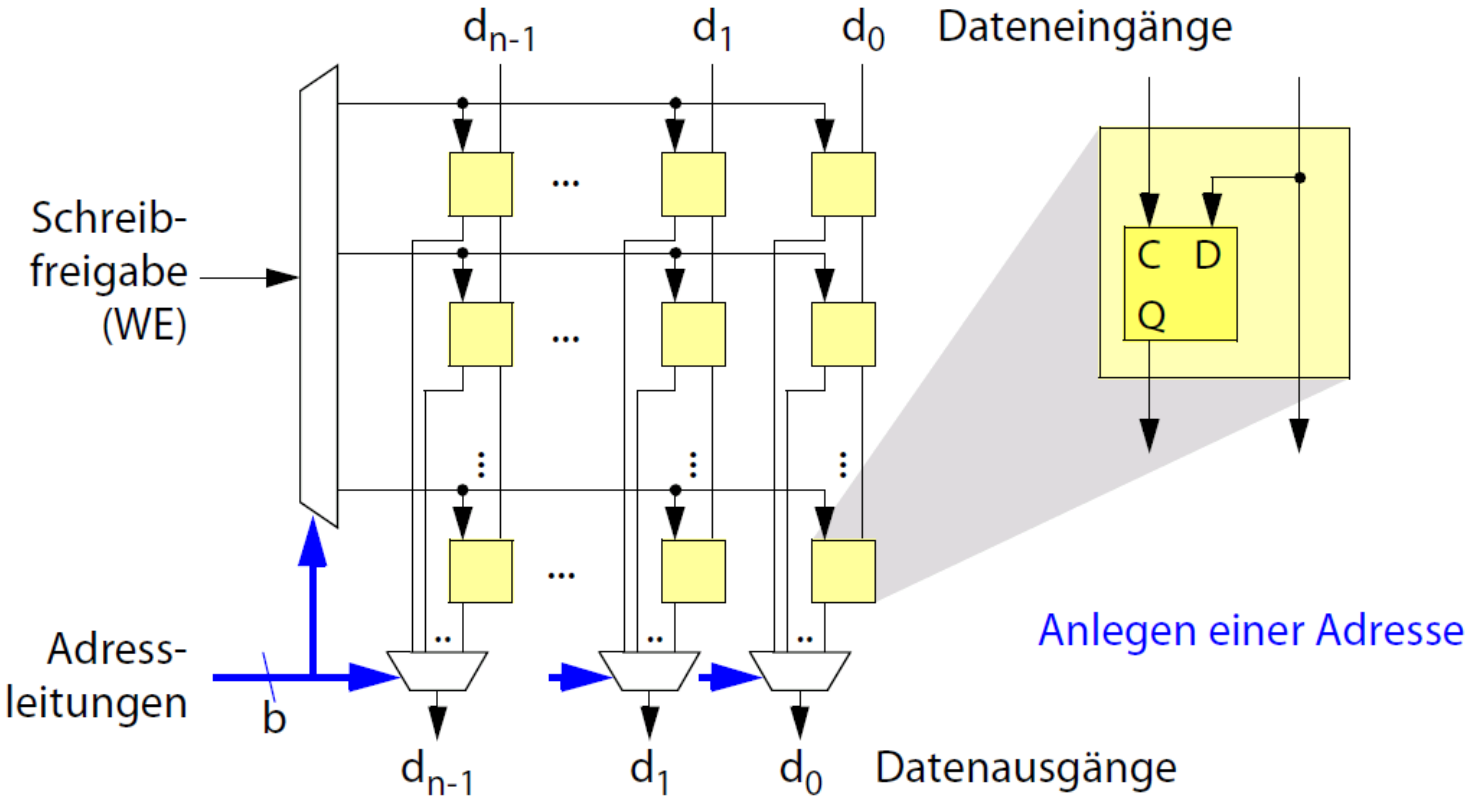
- Zellen sind zeilen- und spaltenweise angeordnet
- Für jede Spalte eine Datenleitung notwendig
- Für 2^n Zeilen sind n Adressleitungen notwendig
- Beispiele für Chip-Aufbau von MBit-Speicherchips
  - 256k x 4 256 x 1024 Worte à 4 Bit (18 Adress-, 4 Datenleitungen)
  - 512k x 2 512 x 1024 Worte à 2 Bit (19 Adress-, 2 Datenleitungen)
  - 1M x 1 1024 x 1024 Worte à 1 Bit (20 Adress-, 1 Datenleitung)

Geschwindigkeit

- Typische Zugriffs- und Zykluszeit: 10ns

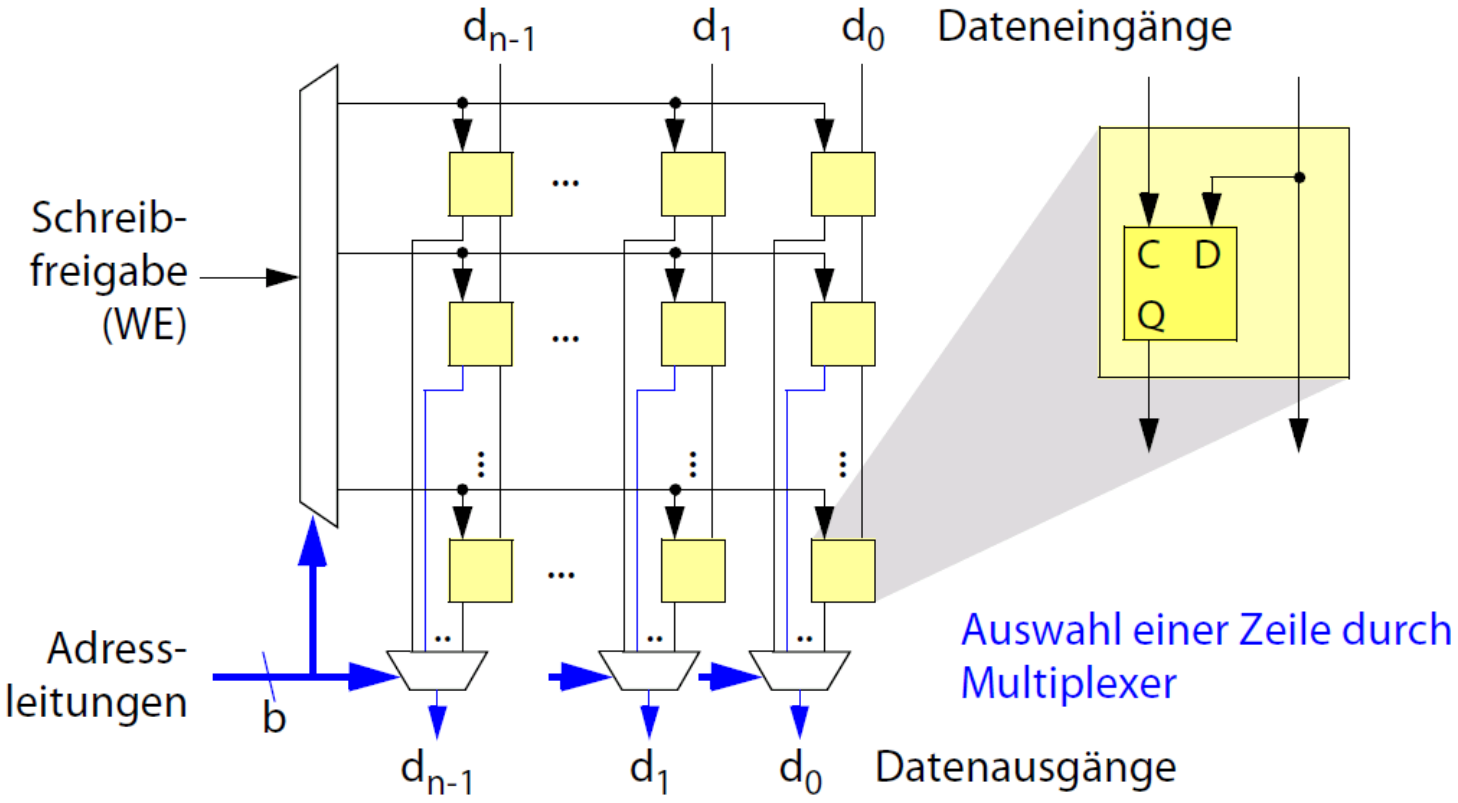
SRAM-Speicher (3)

Aufbau eines SRAM-Speichers mit 2^b Zellen



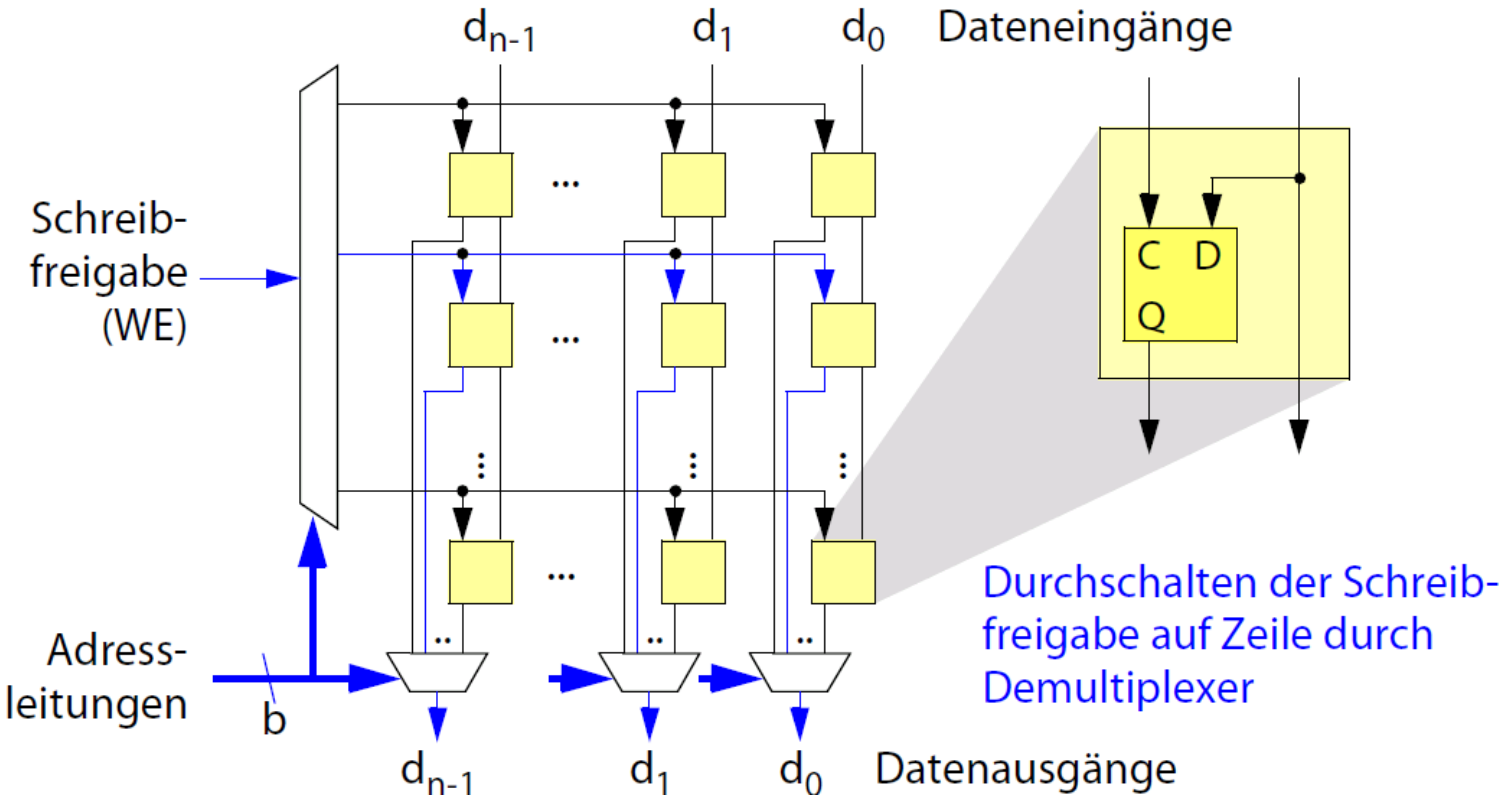
SRAM-Speicher (4)

Aufbau eines SRAM-Speichers mit 2^b Zellen



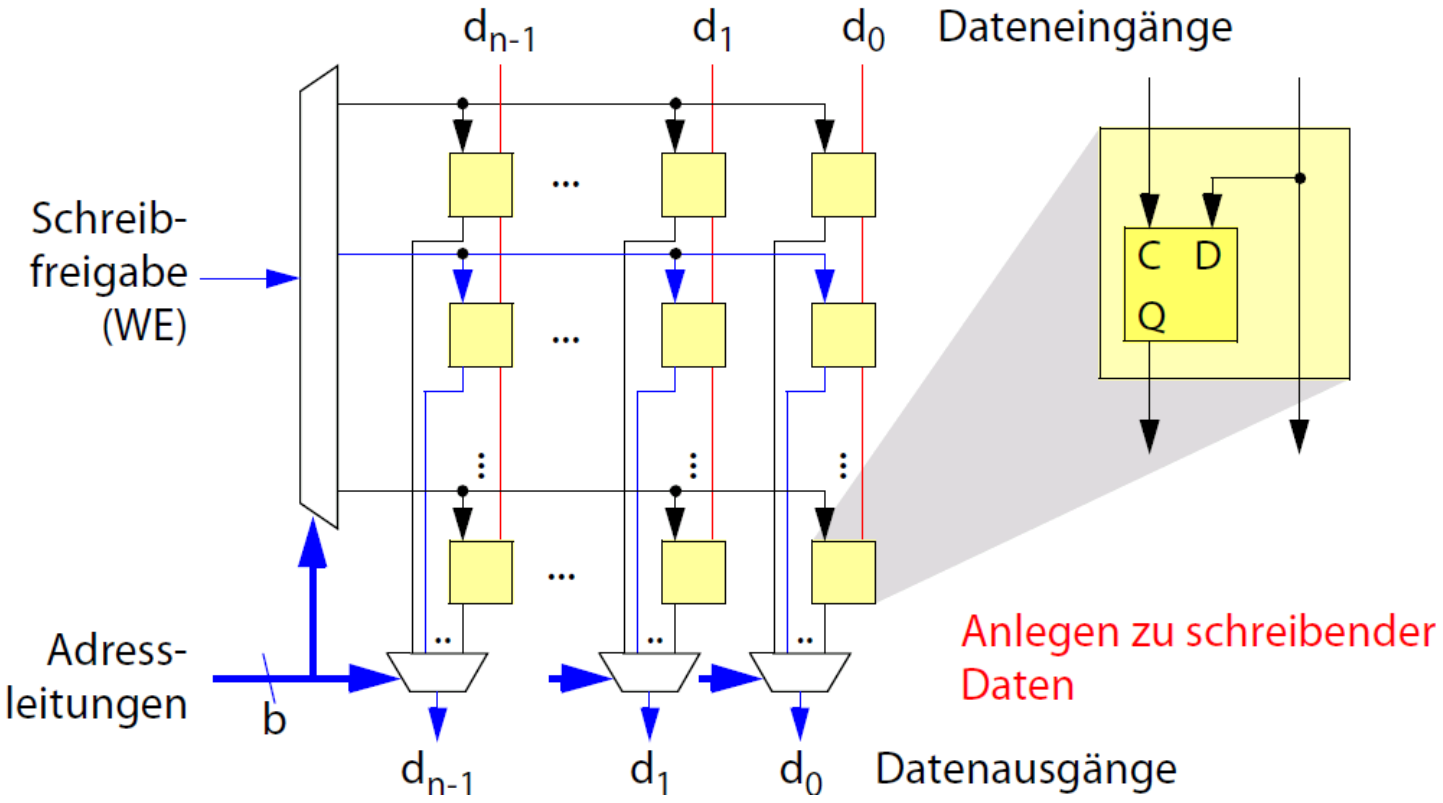
SRAM-Speicher (5)

Aufbau eines SRAM-Speichers mit 2^b Zellen



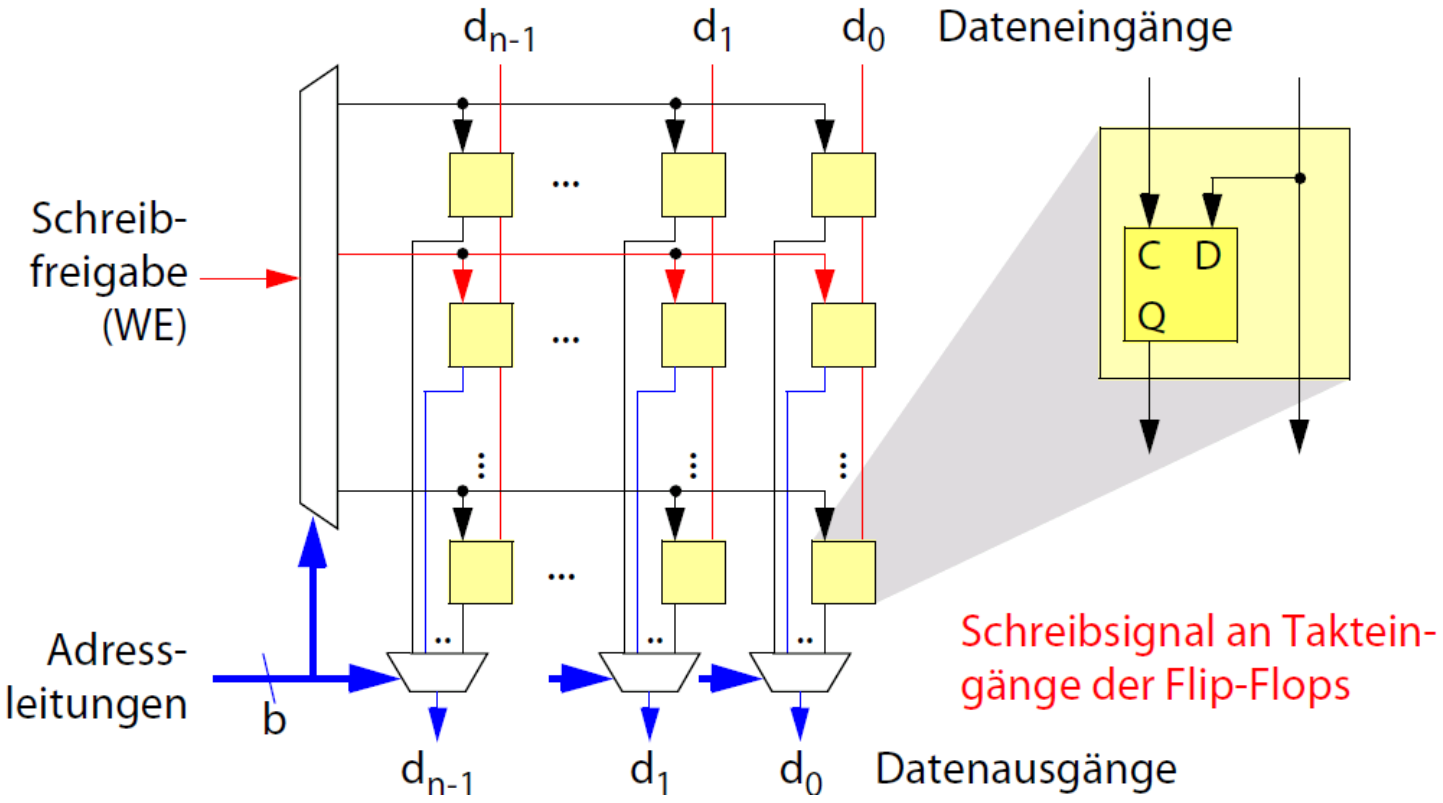
SRAM-Speicher (6)

Aufbau eines SRAM-Speichers mit 2^b Zellen



SRAM-Speicher (7)

Aufbau eines SRAM-Speichers mit 2^b Zellen



Verbesserungen

***Output-Enable*-Steuerleitung (OE)**

– Eigene Steuerleitung zum Schalten des *Tri-State*-Puffers zur Ausgabe
– Überlappung von Adressdekodierung, Speicherzugriff und Ausgabe auf Datenbus möglich

SSRAM *(Synchronous SRAM)*

– Taktleitung
– Synchronität mit Prozessortakt
– Gesteuerte Überlappung möglich

***Burst*-Modus**

– Interner Zähler sorgt für Inkrementierung der Adresse
  – Kein Anlegen einer neuen Adresse erforderlich
– Speicher liefert zwei oder vier Datenworte von aufeinanderfolgenden Adressen (setzt Taktung voraus)

Einsatz

SRAM-Eigenschaften

- Schneller Zugriff
  - Z.B. 10ns auf externem Baustein
  - Im Prozessor integriert *(on-chip)* noch schneller
- Großer Platzbedarf auf dem Chip ☞ Teuer!
- Großer Energiebedarf im aktiven Zustand
- Unempfindlich gegen Strahlung und Einstreuung

Einsatz

- *Cache*-Speicher (Level 1 und 2 *Caches*)
- Sicherheitskritische Systeme (z.B. Raumfahrt, Echtzeitsysteme)
- Mobile Geräte, eingebettete Systeme
- Schneller Hauptspeicher (z.B. Hochleistungsrechner)

Roter Faden

7. Speicher-Hardware

- Einleitung: Speicherhierarchien
- SRAM
  - Aufbau der SRAM-Zelle
  - Schreib- und Lesevorgänge
  - Aufbau von SRAM-Speichern
  - Verbesserungen
- DRAM
- *Translation Look-Aside Buffer* (TLBs)
- *Caches*
- Massenspeicher

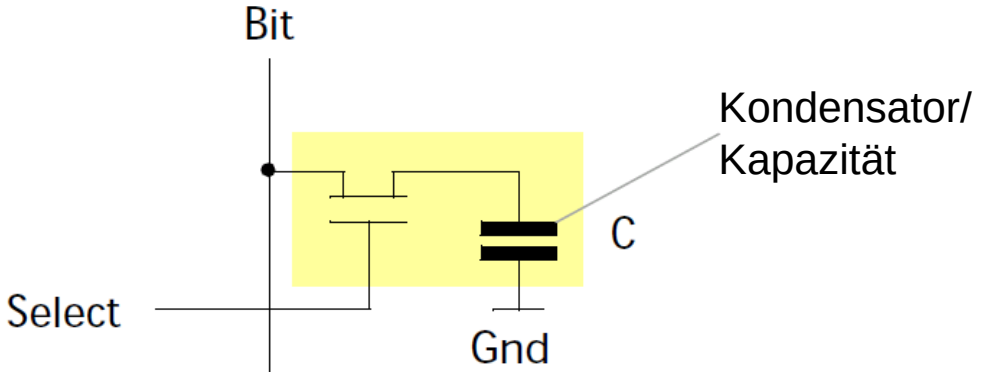
DRAM

Dynamic Random Access RAM (DRAM)

- Speicher basiert auf in einem Kondensator gespeicherter Ladung
- Problem: durch den Kondensator fließt ein kleiner Leckstrom, die Ladung hält sich nur wenige Millisekunden
- Speicher muss periodisch aufgefrischt werden

DRAM-Zelle

- Transistor und Kondensator
- Kondensator zur Speicherung der Information
- Eine Schreib-/Leseleitung pro Spalte
- Eine Auswahlleitung pro Zeile, diese schaltet Transistor



DRAM-Zelle

Schreiben einer DRAM-Zelle

- Bit-Leitung auf 1 oder 0 setzen (mit Vcc oder GND verbinden)
- Select-Leitung ansteuern
  - Kapazität aufladen oder entladen

Lesen einer DRAM-Zelle

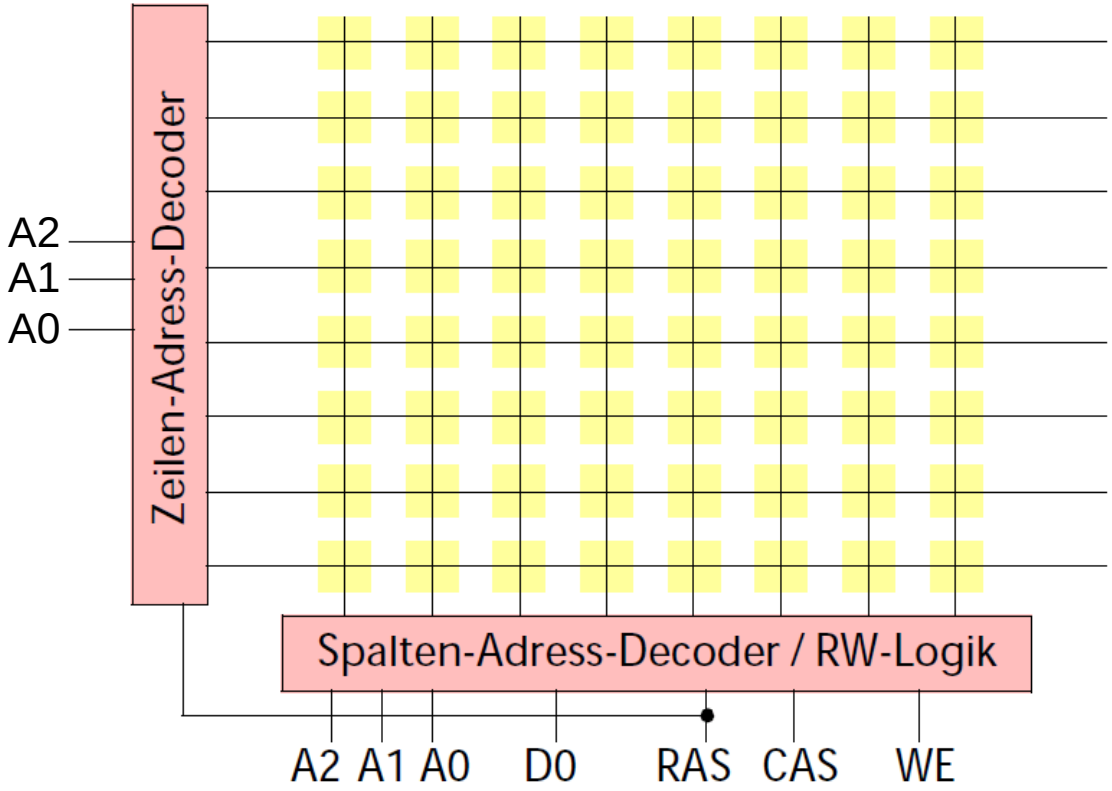
- Select-Leitung ansteuern
- Ladung des Kondensators über Bit-Leitung messen
  - Einfache Verstärkerschaltung
- Lesevorgang zerstört Ladung
  - Anschließendes Rückschreiben des gelesenen Wertes

Auffrischung

- Periodisches Lesen der DRAM-Zelle gleicht Ladungsverlust wieder aus

Typischer DRAM-Speicher (1)

Matrixaufbau eines DRAM-Speichers



Typischer DRAM-Speicher (2)

Matrixaufbau eines DRAM-Speichers

- Adressleitungen werden i.d.R. gemultiplext
  - Die gleichen Adressleitungen werden einmal zur Auswahl der Zeile verwendet, danach zur Auswahl der Spalte

☞ Einsparung von Leitungen, gerade für große Speicher wichtig

- Steuerleitungen RAS/CAS codieren, ob Adressleitungen Zeile oder Spalte auswählen
  - *RAS (Row Address Strobe):* Bei einer fallenden Flanke auf RAS wird die anliegende Adresse als Zeilenadresse interpretiert
  - *CAS (Column Address Strobe):* Bei einer fallenden Flanke auf CAS wird die anliegende Adresse als Spaltenadresse interpretiert

Typischer DRAM-Speicher (3)

Zugriff auf DRAM

- Erster Schritt
  - Zeilenadressdecoder liefert Select-Leitung für eine Zeile
  - Komplette Zeile wird in einen Zwischenpuffer übernommen
    - Und zurückgeschrieben!
- Zweiter Schritt
  - Aus dem Zwischenpuffer wird ein Bit ausgelesen
  - Schritt kann mehrfach wiederholt werden
    (mehrere aufeinanderfolgende Bits können gelesen werden)

Auffrischung

- Heute auf dem DRAM-Speicher integriert
- Früher durch externe Bausteine ausgelöst

Typischer DRAM-Speicher (4)

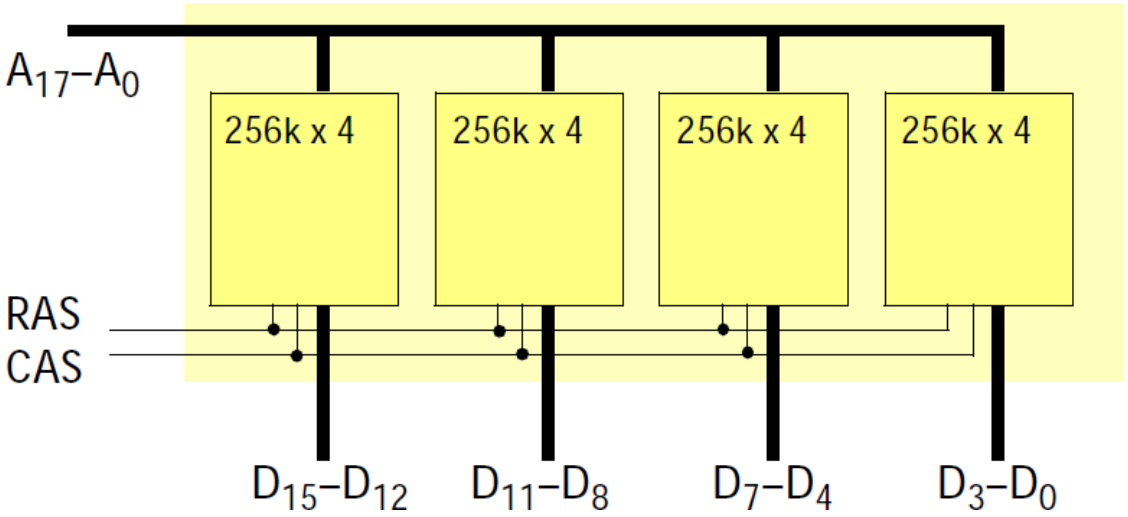
DRAM-Eigenschaften

- Weniger Platzbedarf
  - Nur 1 Transistor und 1 Kondensator pro Speicherzelle, statt 6 Transistoren bei SRAM
  - Integrationsdichte Faktor 4 höher als bei SRAMs
- Langsamerer Zugriff
  - Insbes. Lesezugriff wegen Zwischenspeicherung und Auffrischung
  - Multiplexen der Adressleitungen
  - Auf DRAM-Zeile kann während Auffrischung nicht zugegriffen werden
- Hoher Energieverbrauch sowohl bei Aktivität als auch bei Inaktivität
  - Ausgleich des Ladungsverlusts durch periodische Auffrischung
  - Zwischenpuffer und Logik zur Auffrischung

Zusammenschaltung von Speicherbausteinen (1)

Verbreitern des Datenwortes

– Z.B. vier Bausteine mit 256k Worten à 4 Bit für Speicher mit 256k Worten à 16 Bit

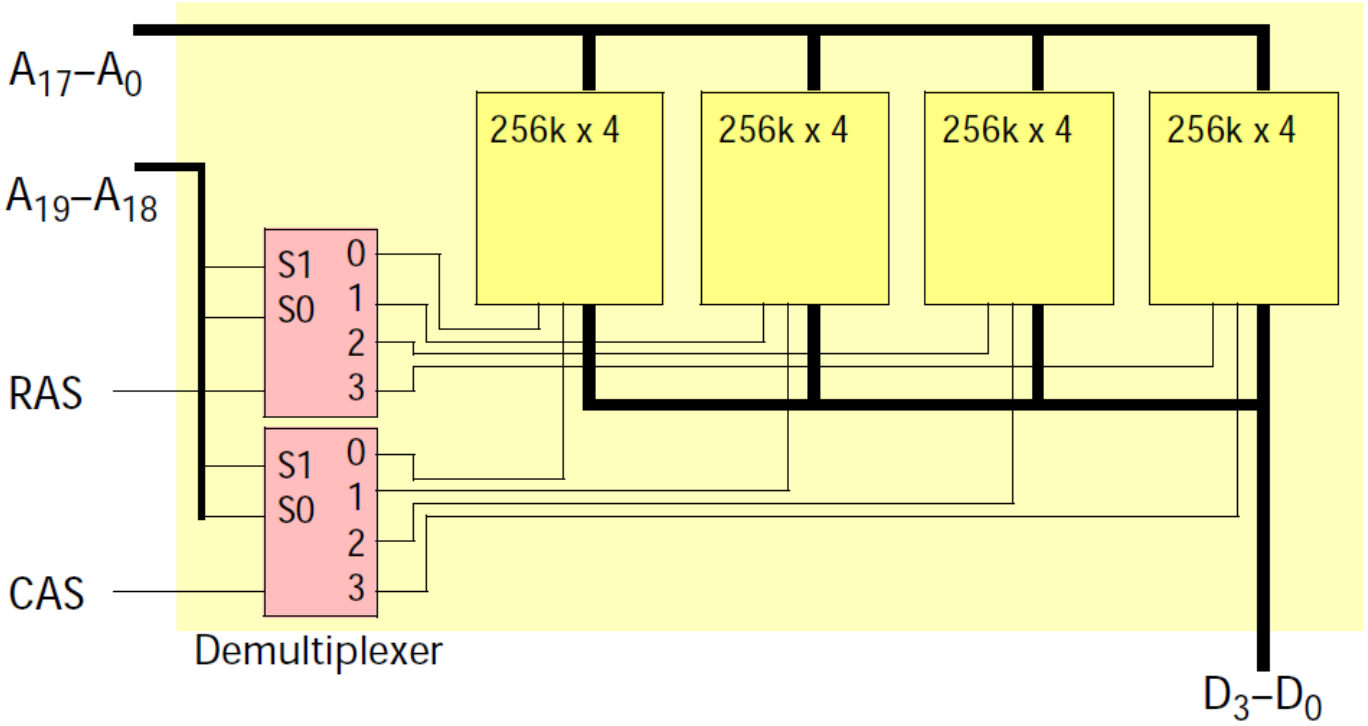


– Alle Bausteine gleich beschaltet
– Zusammenfassen der Datenleitungen

Zusammenschaltung von Speicherbausteinen (2)

Erhöhung der Anzahl der Speicherworte

– Z.B. vier Bausteine mit 256k Worten à 4 Bit für Speicher mit 1M Worten à 4 Bit



Verbesserungen (1)

Mehrfache Anwendung des zweiten Zugriffschrittes

- Halbierung der Zugriffszeit (bei ständigen Zugriffen auf aufeinanderfolgende Adressen)
  - Normal 60-70ns
  - Auf Folgeadressen 30-35ns

Überlappung der Zugriffe

- Daten stehen länger am Datenbus
- CAS-Schritt kann überlappend für nächsten Zugriff erfolgen
- EDO-DRAM *(Extended Data Out DRAM)*

Verbesserungen (2)

SDRAM *(Synchronous DRAM)*

- Getaktetes DRAM
- Alle Steuerleitungen werden getaktet verwendet
- Daten werden bei steigender Taktflanke ausgegeben
- Dauer zwischen CAS-Schaltung und Datenausgabe 2-3 Takte
  - Abhängig von eingestellter CAS-Latenz
- Automatische Ausgabe der Folgedaten durch „Stehenlassen“ des CAS-Signals in weiteren Takten
  - Schneller Zugriff, da Pufferregister in SRAM-Technologie
- Zugriffszeit
  - Erstes Wort: 60ns (16 MWorte/s)
  - Folgewort: 7,5ns (133 MWorte/s)

Verbesserungen (3)

DDR-SDRAM *(Data Double Rate Synchronous DRAM)*

- Gleicher Systemtakt
- Jedoch doppelte Datenrate
  - Auslesen von zwei aufeinanderfolgenden Adressen
  - Datenwort bei steigender und fallender Taktflanke
- Zugriffszeit
  - Erstes Wort: 60ns (16 MWorte/s)
  - Folgewort: 3,75ns (266 MWorte/s)

Achtung

- Versprechen aus der Werbung: 3,75ns Zugriffszeit
- Tatsächlich nur für Folgezugriffe mit aufsteigenden Adressen
- Erster Zugriff kostet nach wie vor 60ns (DRAM-Prinzip unverändert)

Verbesserungen (4)

Hauptsächlich Verbesserungen an der Speicherschnittstelle

– Kern des DRAM ist im Wesentlichen gleich geblieben

Verbesserungen nur bei Folgezugriffen

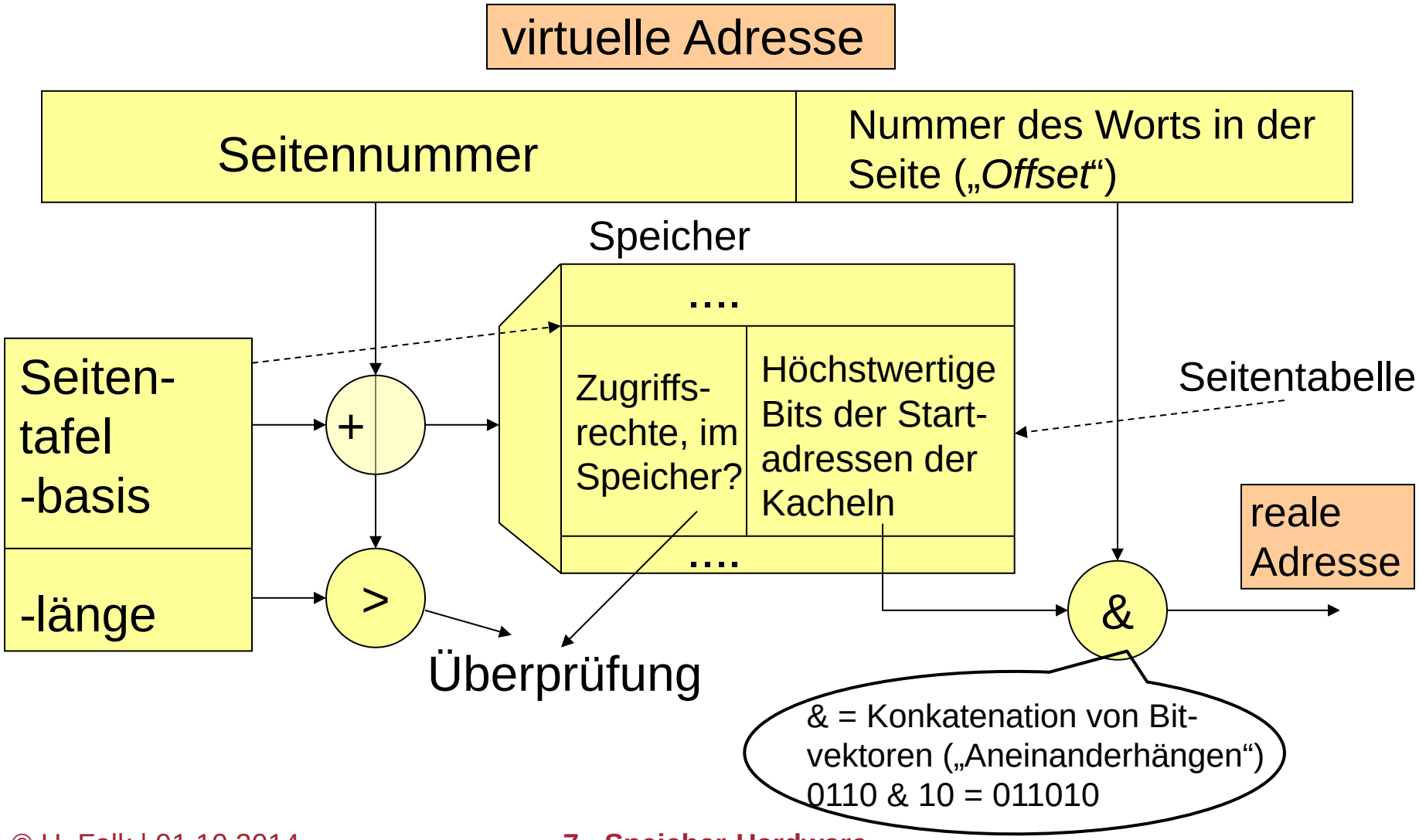
– Wahlfreie Zugriffe brauchen immer noch lange
– Folgezugriffe jedoch häufig durch Einsatz von *Caches*
  – *Caches* speichern nicht individuelle Speicherstellen sondern Zeilen à 16 Bytes

Roter Faden

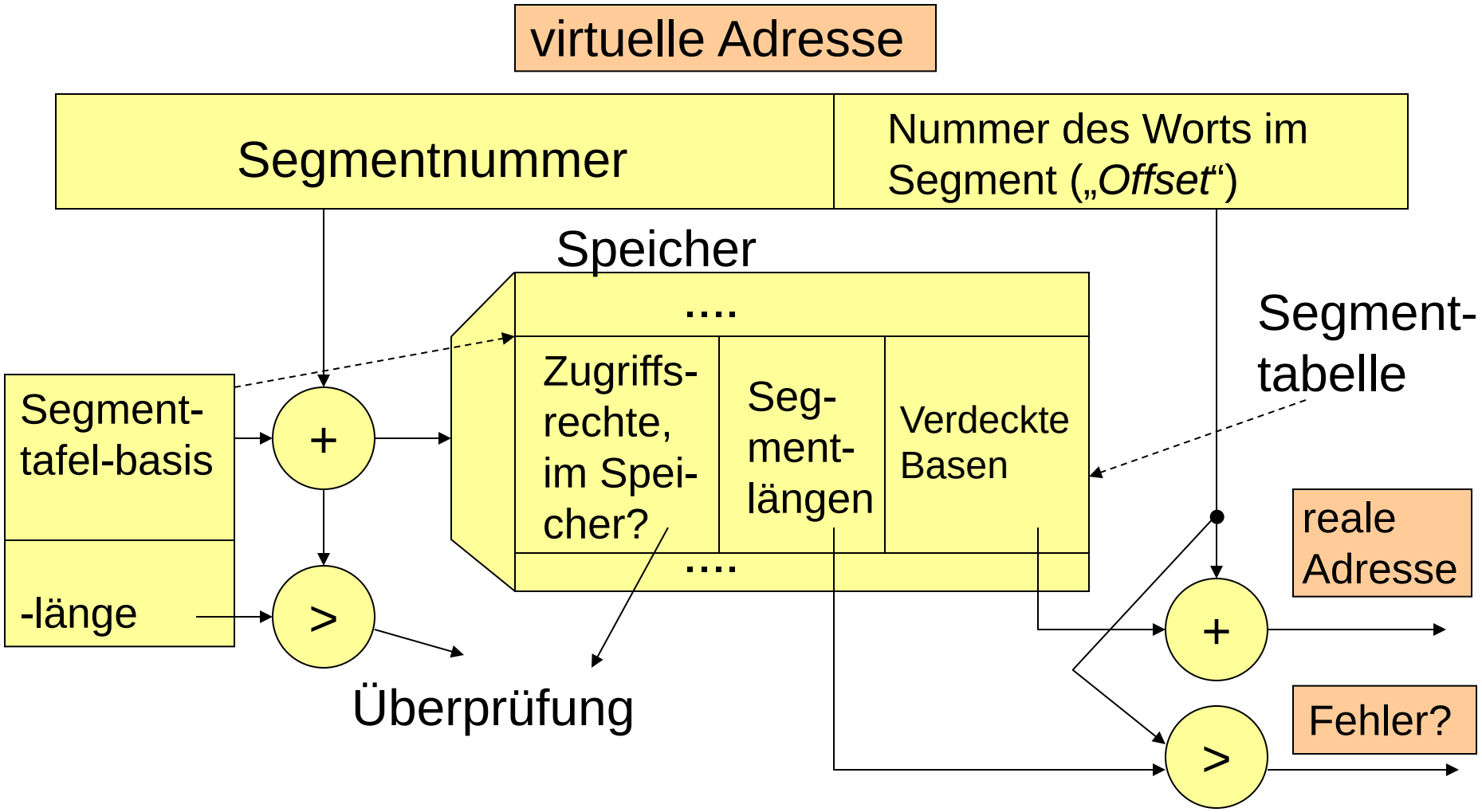
7. Speicher-Hardware

- Einleitung: Speicherhierarchien
- SRAM
- DRAM
  - Aufbau der DRAM-Zelle
  - Matrixaufbau von DRAM-Speichern
  - Verbesserungen
- *Translation Look-Aside Buffer* (TLBs)
- *Caches*
- Massenspeicher

Erinnerung: *Paging*



Erinnerung: Segmentierung



Translation Look-Aside Buffer (TLBs)

Seitentabellen und Segmenttabellen im Hauptspeicher

- Jeder einzelne Zugriff auf einen Befehl oder ein Datum erfordert zusätzliche Speicherzugriffe zur Umrechnung von virtuellen in reale Adressen
- Unakzeptable Verlangsamung der Ausführung

☞ Einführung zusätzlicher, kleiner, schneller Hardwarespeicher, welche häufig die benötigten Tabelleneinträge enthalten

Diese heißen *Translation Look-Aside Buffer (TLB)* oder auch *Address Translation Memory (ATM)*

Drei Organisationsformen von TLBs

In jedem Fall enthalten die TLBs nur Auszüge aus den vollständigen Tabellen, die weiterhin komplett im Hauptspeicher sind.

Ist die gesuchte Seiten- bzw. Segmentnummer nicht im TLB enthalten, so muss in der vollständigen Tabelle im Hauptspeicher nachgesehen werden.

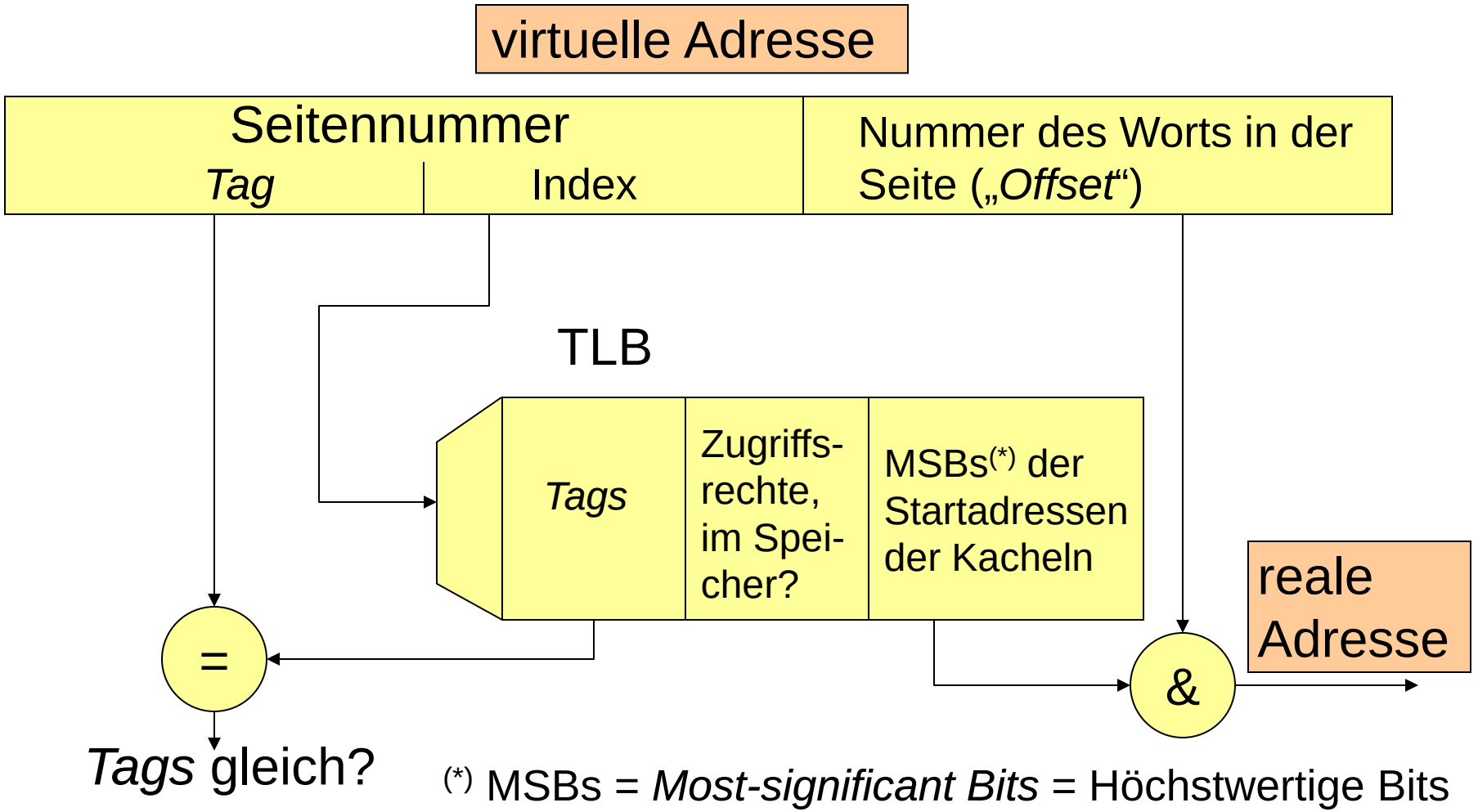
Der Eintrag aus der vollständigen Tabelle verdrängt dann einen Eintrag aus dem TLB nach einem noch zu bestimmenden Algorithmus.

Organisationsformen von TLBs

- *Direct mapping*
- Mengen-assoziative Abbildung *(set-associative mapping)*
- Assoziativspeicher *(associative mapping)*

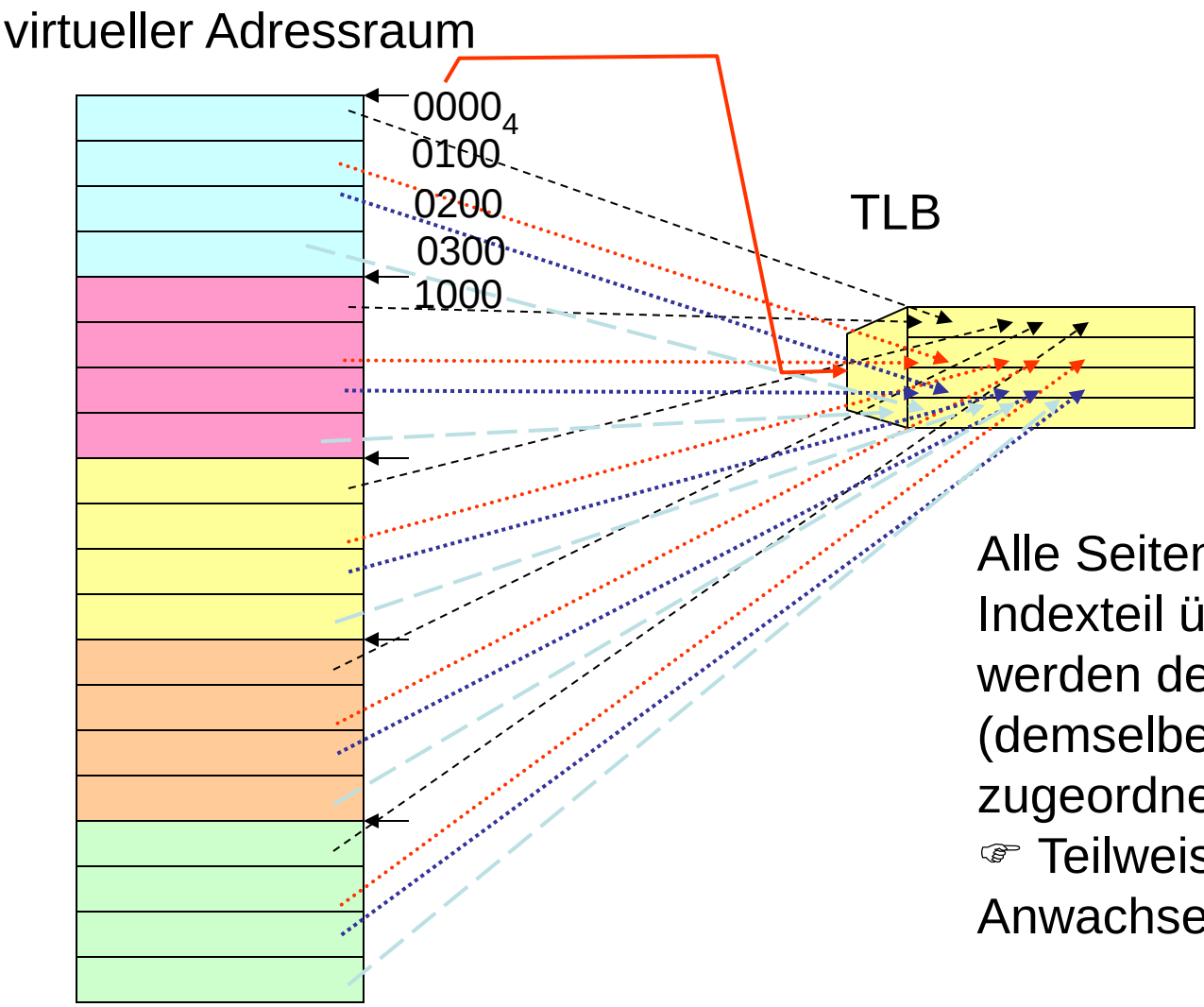
Direct Mapping

Die Seitennummer oder ein Teil davon adressiert den TLB.



(*) MSBs = *Most-significant Bits* = Höchstwertige Bits

Aliasing bei *Direct Mapping*



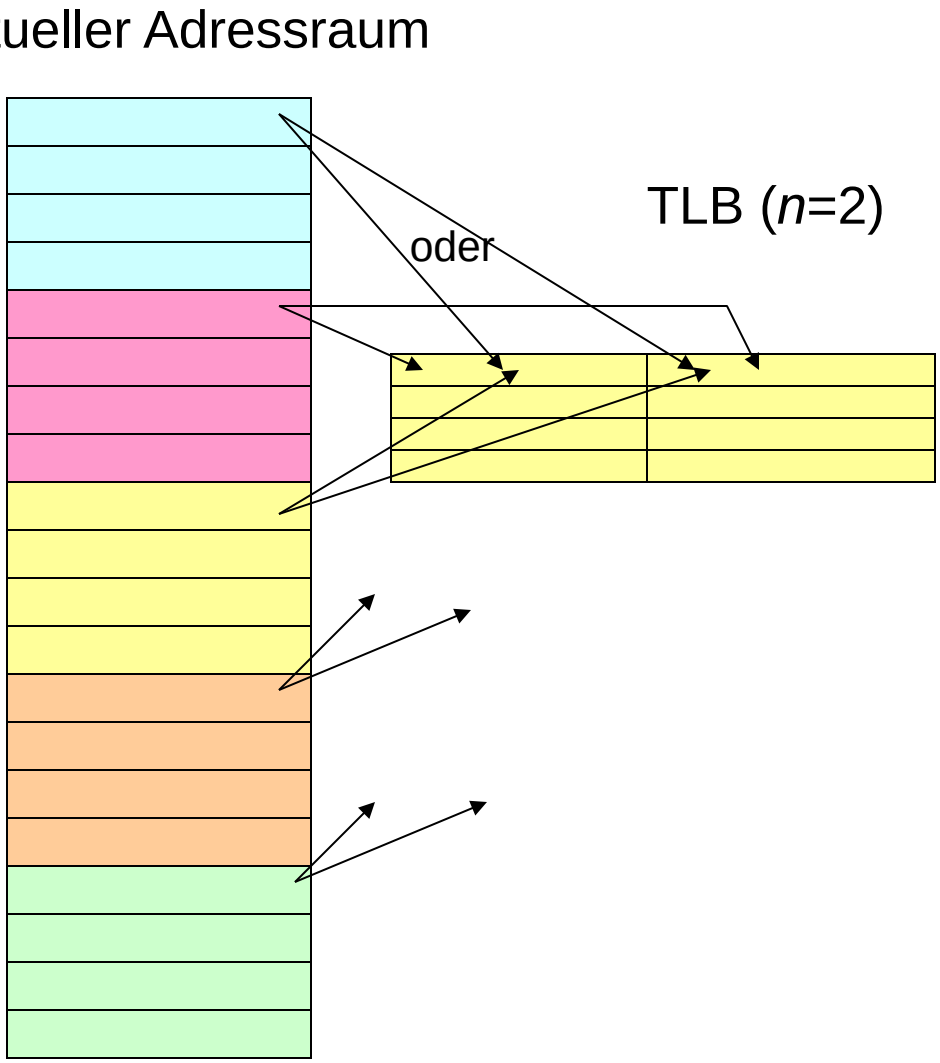
Alle Seiten, deren Adressen im Indexteil übereinstimmen, werden derselben TLB-Zelle (demselben Namen (Alias)) zugeordnet.

☞ Teilweise starkes Anwachsen der Laufzeit

Mengen-assoziative Abbildung – Prinzip

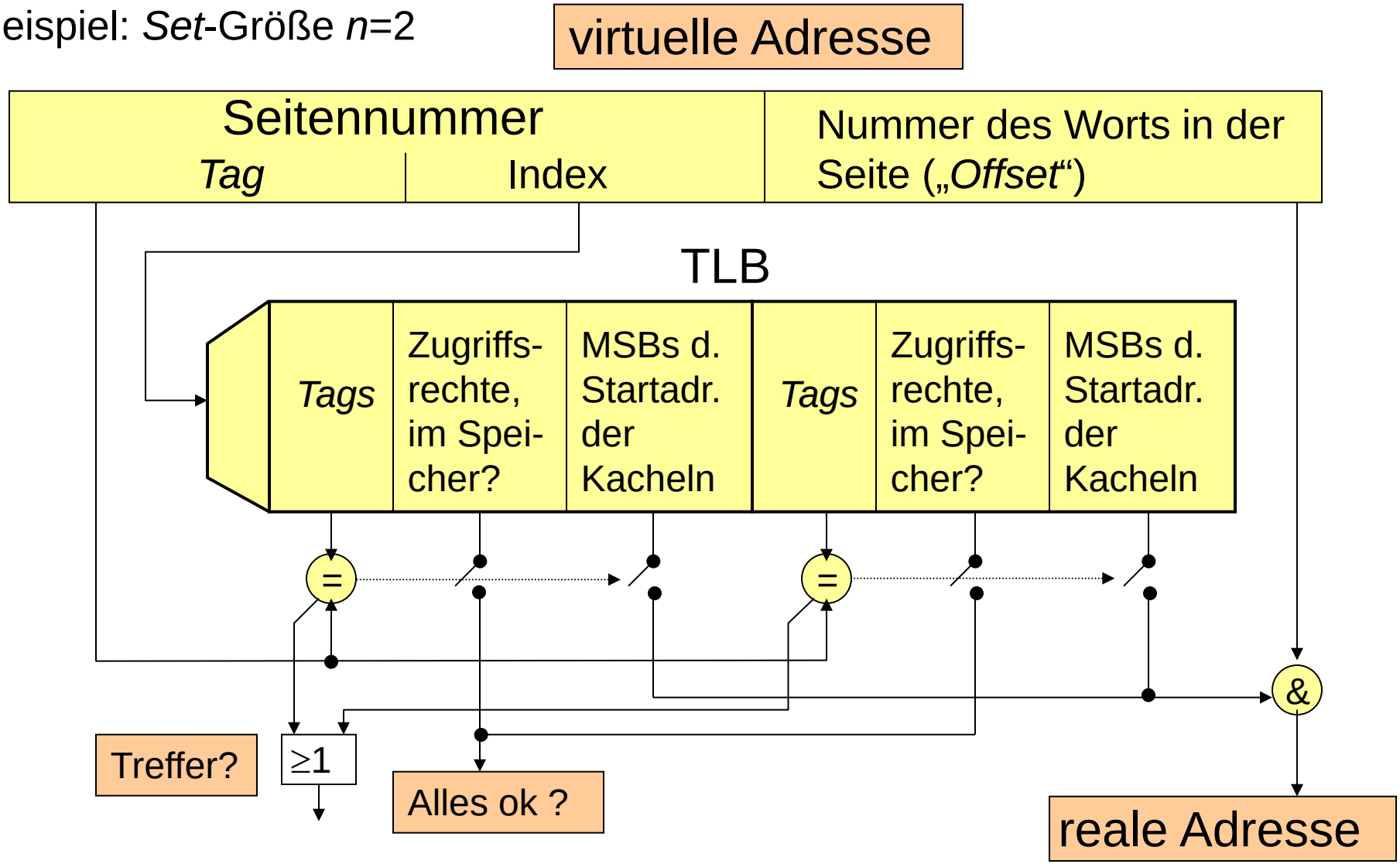
Zu jedem Index-Wert gibt es im TLB *n* zugeordnete Plätze, mit *n*>1

Sind alle *n* Plätze für einen neu einzulagernden Index-Wert belegt, wird ein Platz im TLB überschrieben, z.B. der am längsten nicht benutzte *(least recently used (LRU))*



Mengen-assoziative Abbildung – Realisierung

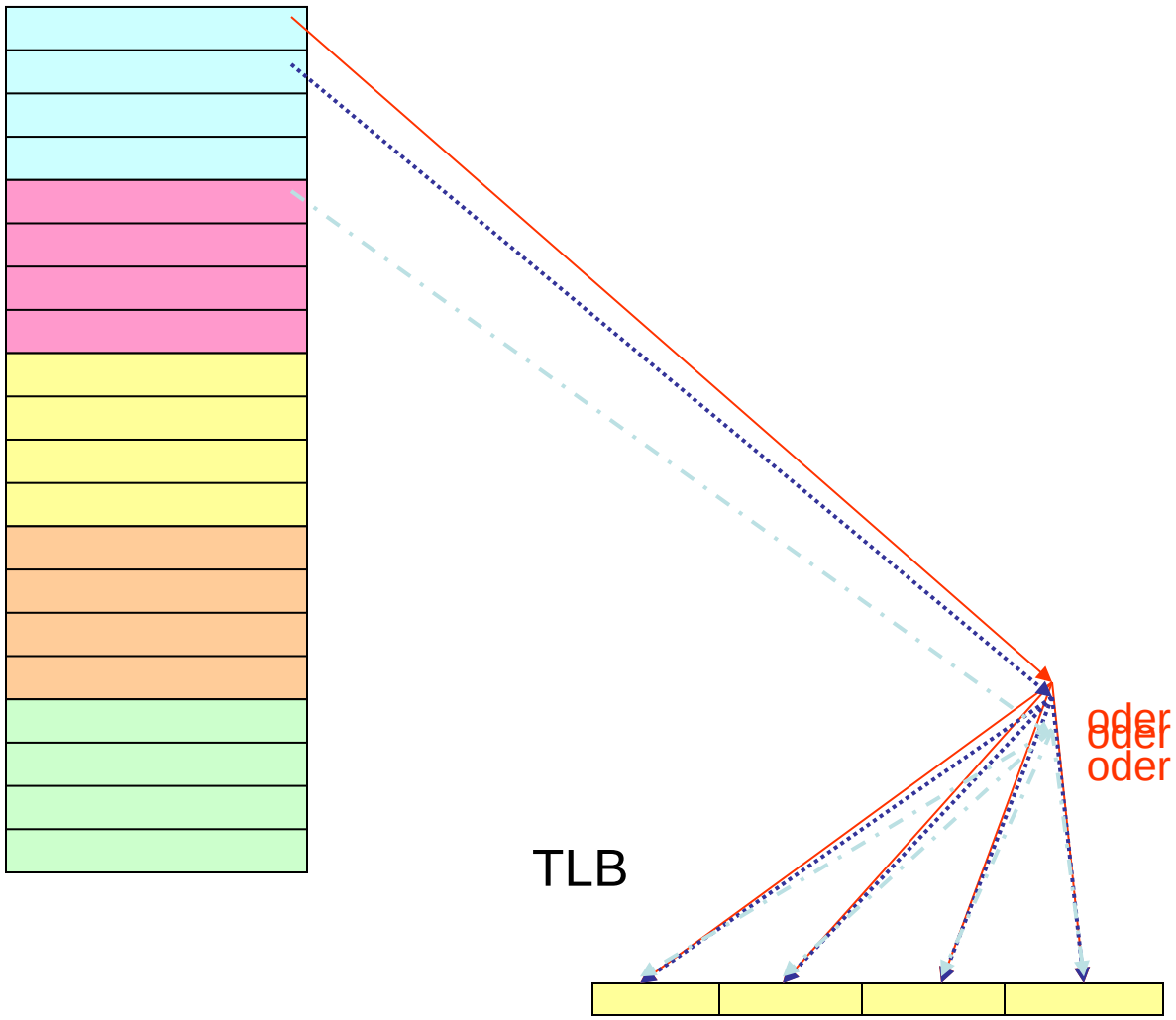
Beispiel: *Set*-Größe *n*=2



Assoziativspeicher, *associative mapping* – Prinzip

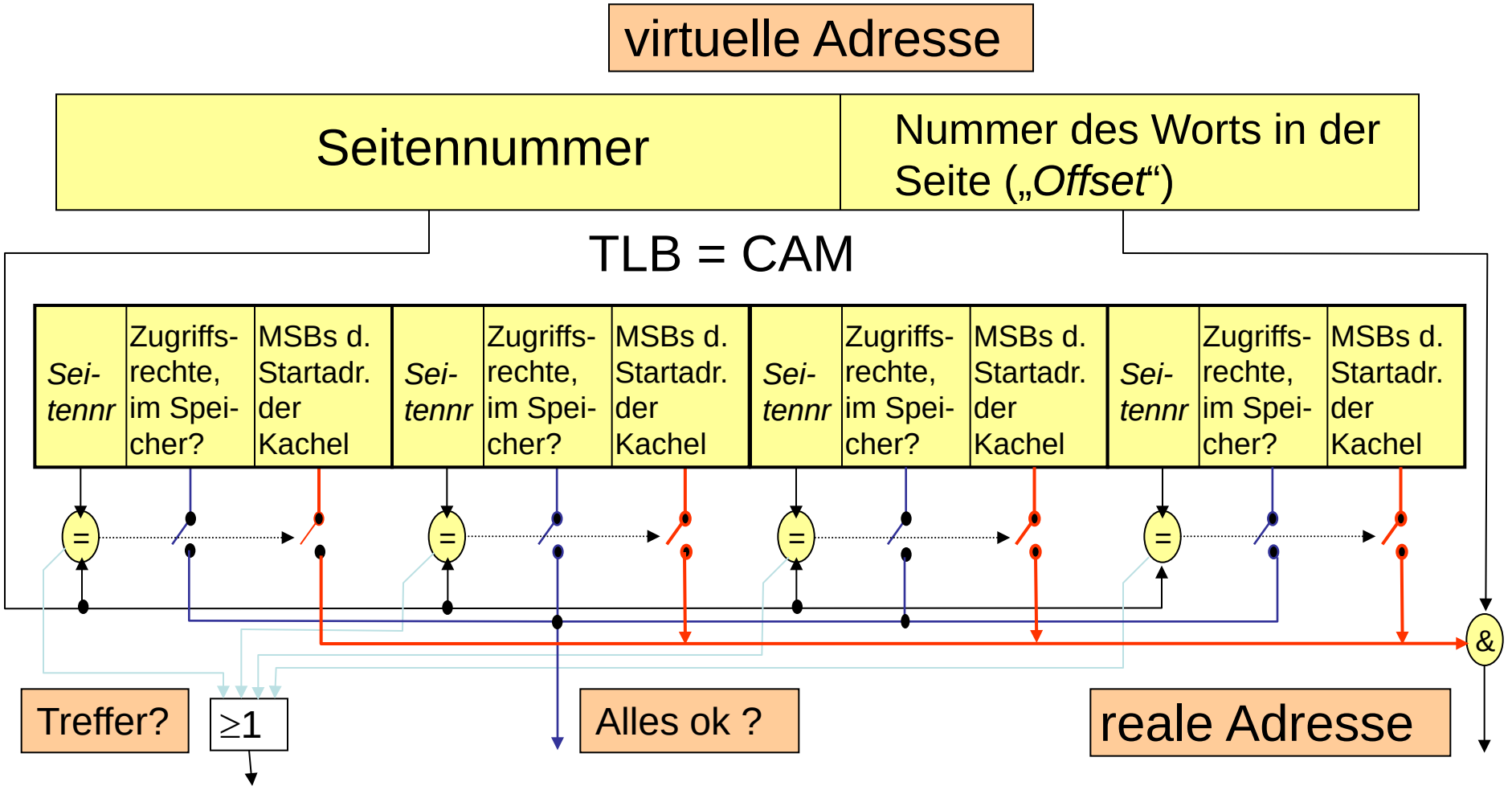
Der Fall *associative mapping* entsteht aus dem *set-associative mapping* für „Anzahl der Bits für den Index-Teil → 0“.

Jeder Platz des TLB kann die Verwaltungsinformation für jede beliebige Seite enthalten (kein *Aliasing*).

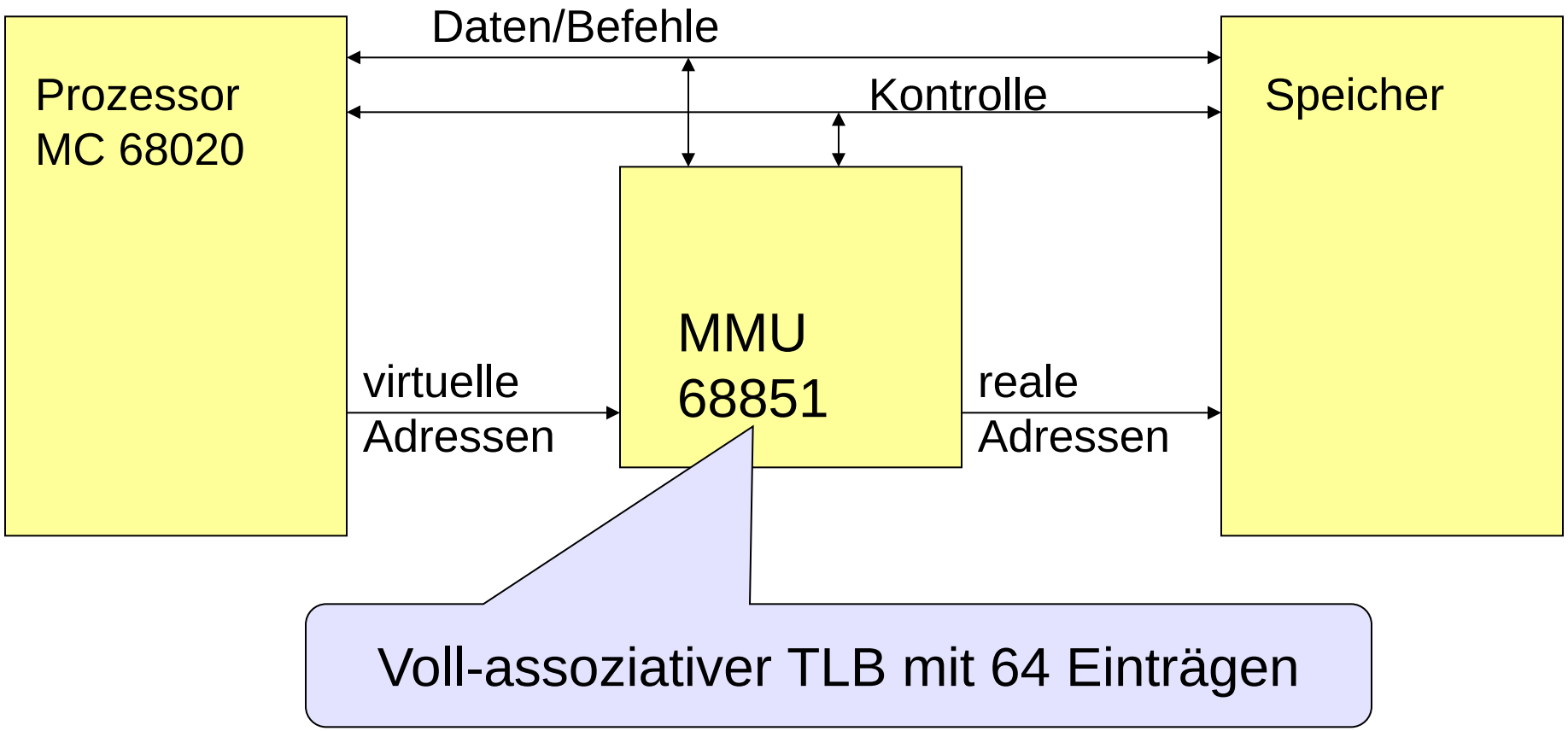


Hardware-mäßige Realisierung mit *content-adressable memory (CAM)*.

Ein CAM vergleicht die Seitennummern aller Zellen gleichzeitig mit dem angelegten Suchwert und liefert im Fall eines Treffers die übrigen Informationen der betreffenden Zelle.



Anwendung: *Memory Management Unit (MMU)* Motorola MC 68851



Zusammenfassung

Translation Look-Aside Buffer (TLBs) dienen dem raschen Zugriff auf häufig benutzte Adressumrechnungs-Informationen

Fehlende Treffer ☞ Umrechnung über Seiten-/Segmenttabellen im Hauptspeicher

Drei Organisationsformen von TLBs

- *Direct mapping*
- Mengen-assoziative Abbildung *(set-associative mapping)*
- Assoziativspeicher *(associative mapping)*

TLBs können Teil der *Memory Management Unit (MMU)* sein

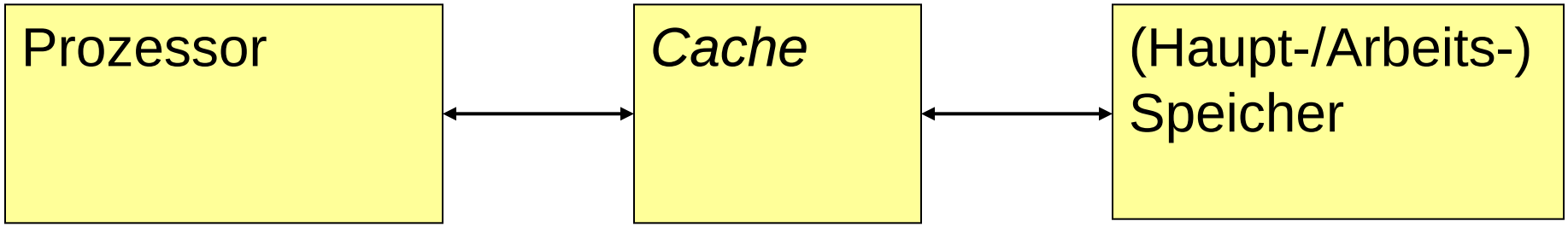
Roter Faden

7. Speicher-Hardware

- Einleitung: Speicherhierarchien
- SRAM
- DRAM
- *Translation Look-Aside Buffer* (TLBs)
  - Motivation: *Paging* & Segmentierung
  - Organisationsformen: *direct mapping*, *set-associative*, *associative mapping*
- *Caches*
- Massenspeicher

Caches

- *Cache* = Speicher, der vor einen größeren, langsamen Speicher geschaltet wird
- Im weiteren Sinn: Puffer zur Aufnahme häufig benötigter Daten
- Im engeren Sinn: Puffer zwischen Hauptspeicher und Prozessor



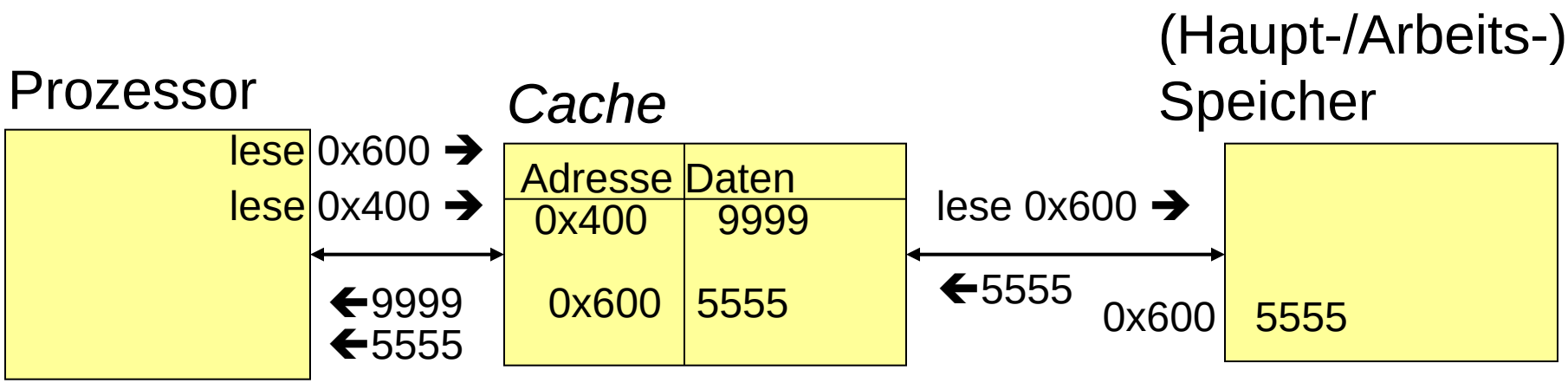
Ursprung: *cacher* (frz.) – verstecken („versteckter Speicher“)

Ablauf

Organisation von *Caches* (im engeren Sinn)

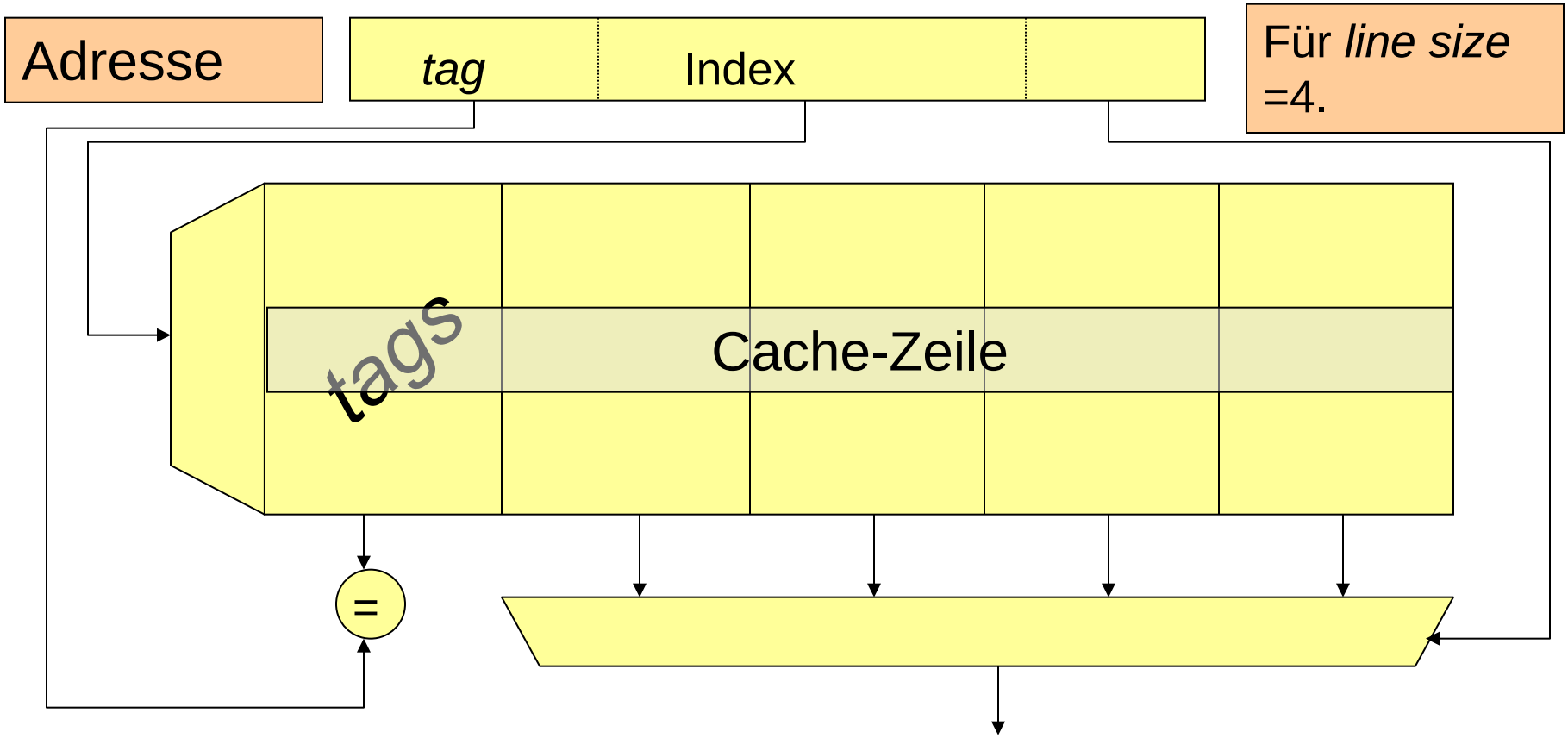
– Prüfung anhand der (virtuellen oder realen) Adresse, ob benötigte Daten im *Cache* vorhanden sind („Treffer“; *cache hit*)

– Falls nicht *(cache miss)*: Zugriff auf den (Haupt-) Speicher, Eintrag in den *Cache*



Prüfung auf *cache hit: Cache*-Zeilen *(cache lines)*

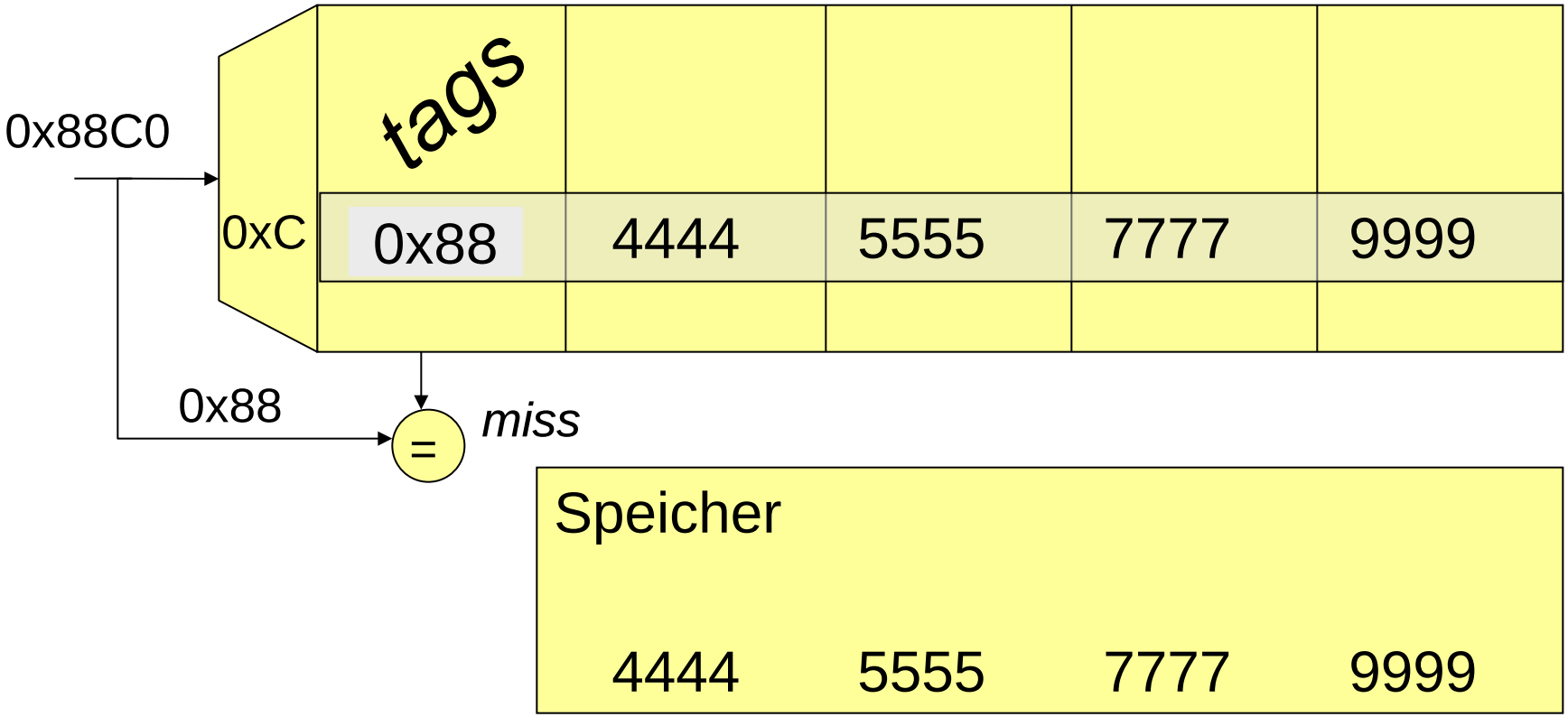
Such-Einheit im Cache: ***Cache*-Zeile** (***cache line***).



Weniger *tag bits*, als wenn man jedem Wort *tag bits* zuordnen würde.

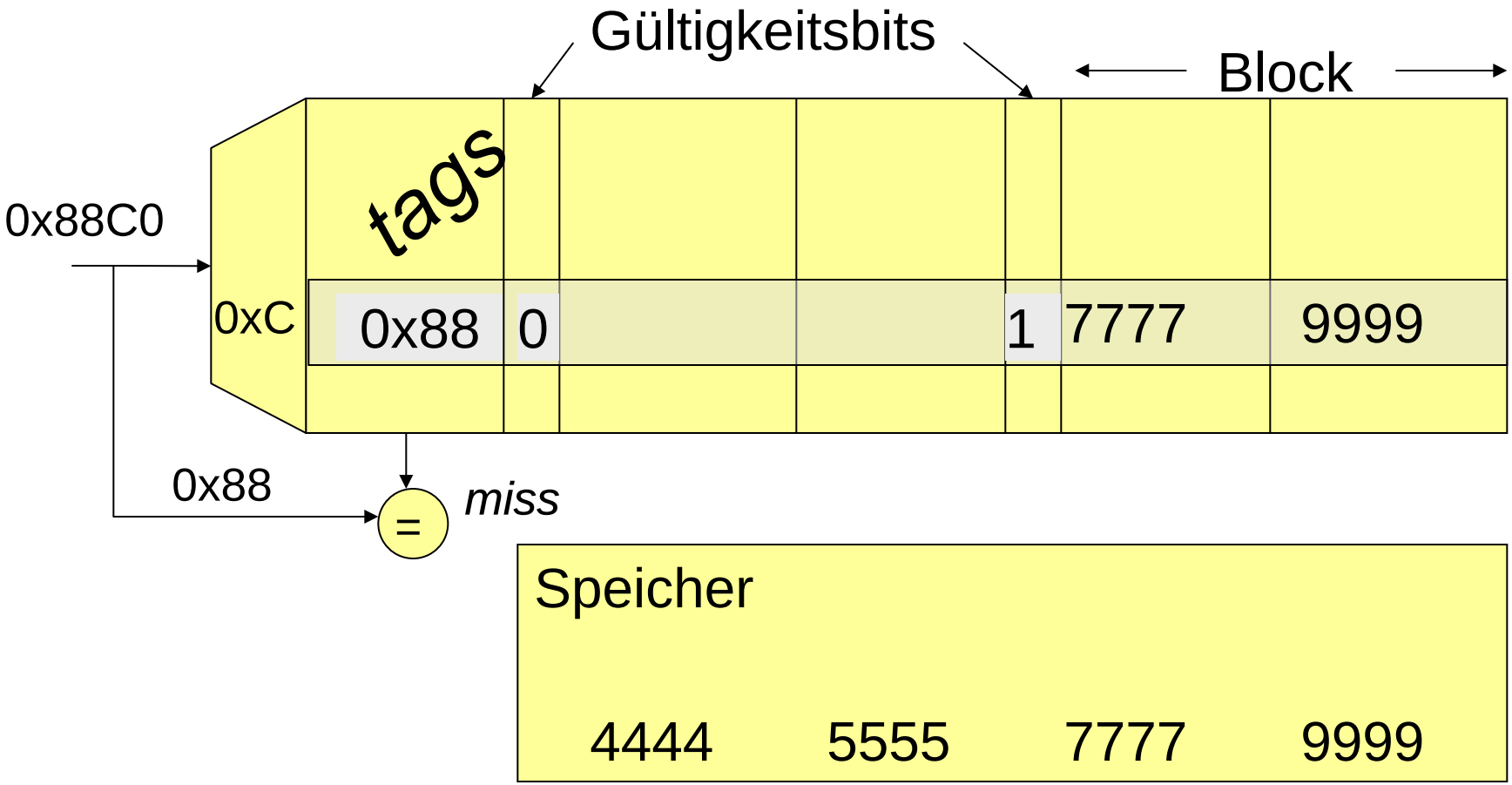
Cache-Blöcke, *cache blocks* (1)

Die **Blockgröße** ist die Anzahl der Worte, die im Fall eines *cache misses* aus dem Speicher nachgeladen werden.
Beispiel: (Blockgröße = *line size*)



Cache-Blöcke, *cache blocks* (2)

Wenn *block size* < *line size*, dann sind zusätzliche Gültigkeitsbits erforderlich. Beispiel: (Blockgröße = *line size* / 2)



Cache-Blöcke, *cache blocks* (3)

Wenn *block size* > *line size*, dann werden bei jedem *miss* mehrere Zeilen nachgeladen.

Stets wird zuerst das gesuchte Wort, dann der Rest des Blocks geladen.

Verbindung Speicher ↔ *Cache* ist so entworfen, dass der Speicher durch das zusätzliche Lesen nicht langsamer wird. Methoden dazu:

1. Schnelles Lesen aufeinanderfolgender Speicherzellen (*Burst*-Modus der Speicher)
2. *Interleaving* (mehrere Speicher ICs mit überlappenden Zugriffen)
3. Fließbandzugriff auf den Speicher (EDO-RAM, SDRAM)
4. Breite Speicher, die mehrere Worte parallel übertragen können

Organisationsformen von *Caches*

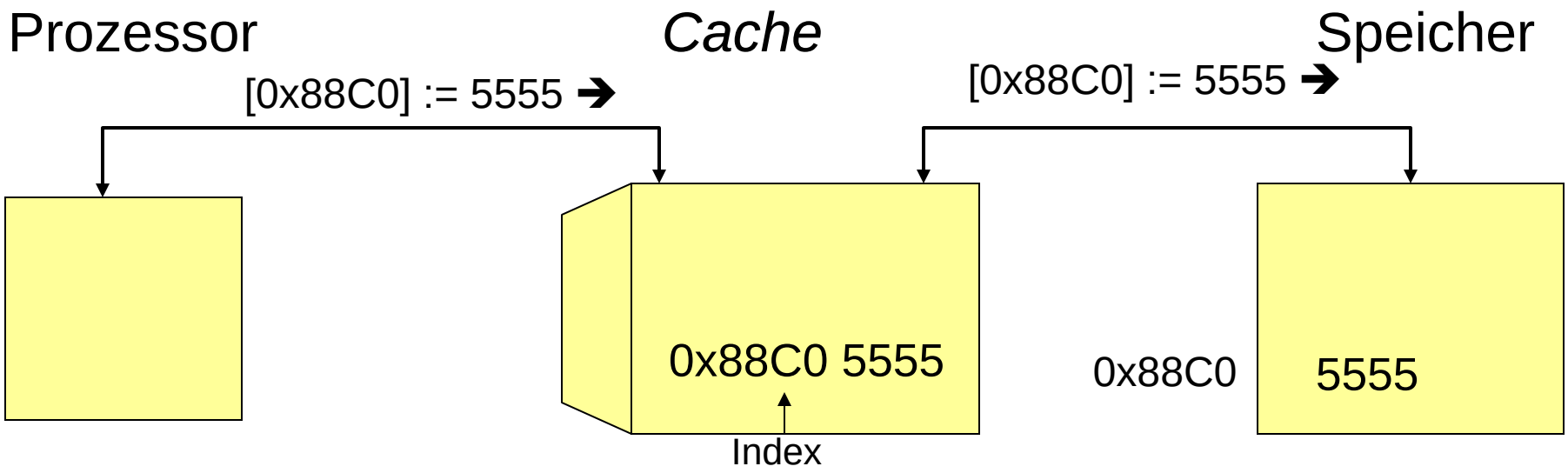
- ***Direct mapping***
  Für *caching* von Befehlen besonders sinnvoll, weil bei Befehlen *Aliasing* sehr unwahrscheinlich ist
- ***Set-associative mapping***
  Sehr häufige Organisationsform, mit *Set*-Größe = 2 oder 4, selten 8
- **Associative mapping**
  Wegen der Größe moderner *Caches* kommt diese Organisationsform kaum in Frage

Schreibverfahren (1)

Strategien zum Rückschreiben *Cache* → (Haupt-) Speicher

1. ***Write-Through*** (Durchschreiben)

   Jeder Schreibvorgang in den *Cache* führt zu einer unmittelbaren Aktualisierung des (Haupt-) Speichers

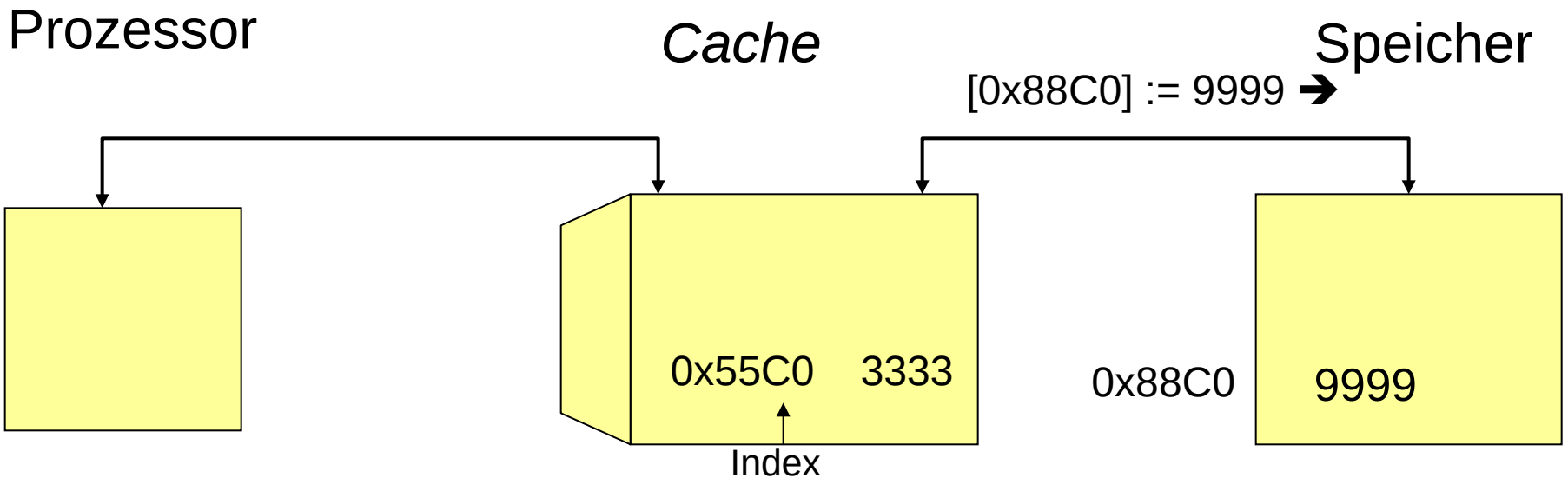


Speicher wird Engpass, es sei denn, der Anteil an Schreiboperationen ist klein oder der (Haupt-) Speicher ist nur wenig langsamer als der *Cache*.

Schreibverfahren (2)

2. Copy-Back, conflicting use write back

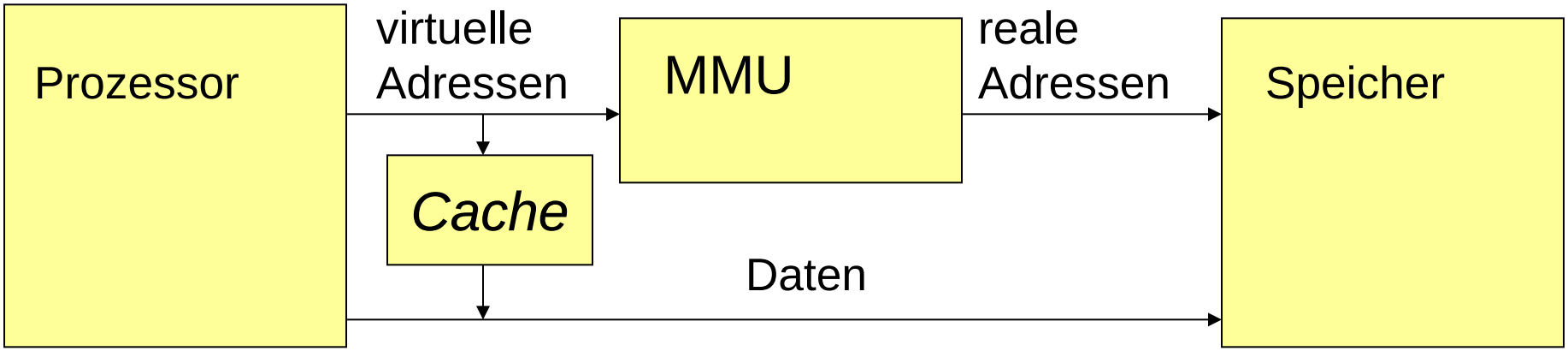
Rückschreiben erfolgt erst, wenn die *Cache*-Zeile überschrieben wird



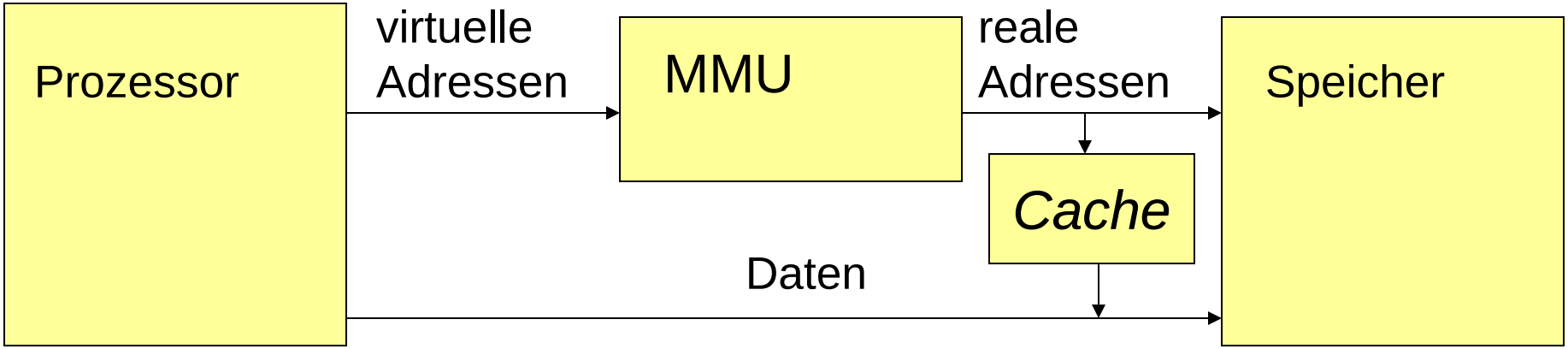
Funktioniert auch bei großen Geschwindigkeitsunterschieden zwischen *Cache* und Speicher. Vorkehrungen erforderlich, damit keine veralteten Werte aus dem Speicher kopiert werden.

Virtuelle und Reale *Caches*

Virtuelle *Caches* (schnell)

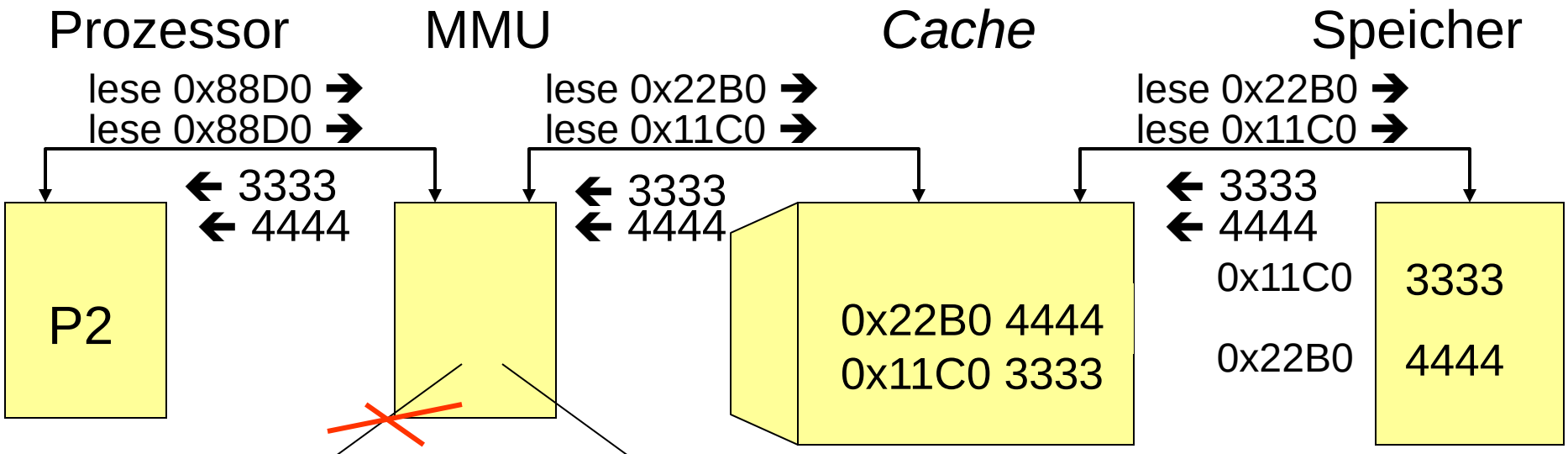


Reale *Caches* (langsamer)



Realer *Cache* bei Prozesswechsel *(context switch)*

Zu einer bestimmten Adresse im *Cache* gehört vor und nach dem Prozesswechsel dieselbe Zelle im Speicher



Prozess P1	
Seite	**Kachel**
0x88D.	0x11C.

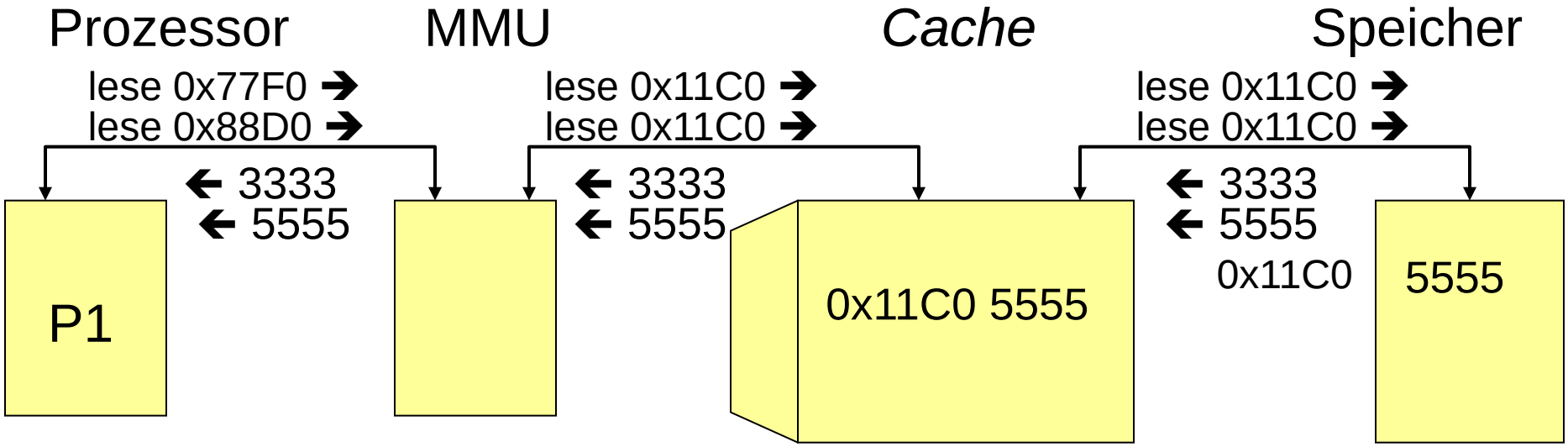
Prozess P2	
Seite	**Kachel**
0x88D.	0x22B.

☞ Bei Prozesswechsel bleiben Informationen im *Cache* korrekt. Keine Aktionen erforderlich.

Eintrag 0x11C0 ist nach Rückschalten auf P1 immer noch korrekt.
Einträge überleben evtl. mehrere Prozesswechsel! ☞ Gut für große *Caches*!

Realer *Cache* bei Seitenfehler (Seite muss per *demand paging* eingelagert werden)

Zu einer bestimmten Adresse im *Cache* gehört vor und nach dem Aus- & Einlagern von Seiten eine andere Information



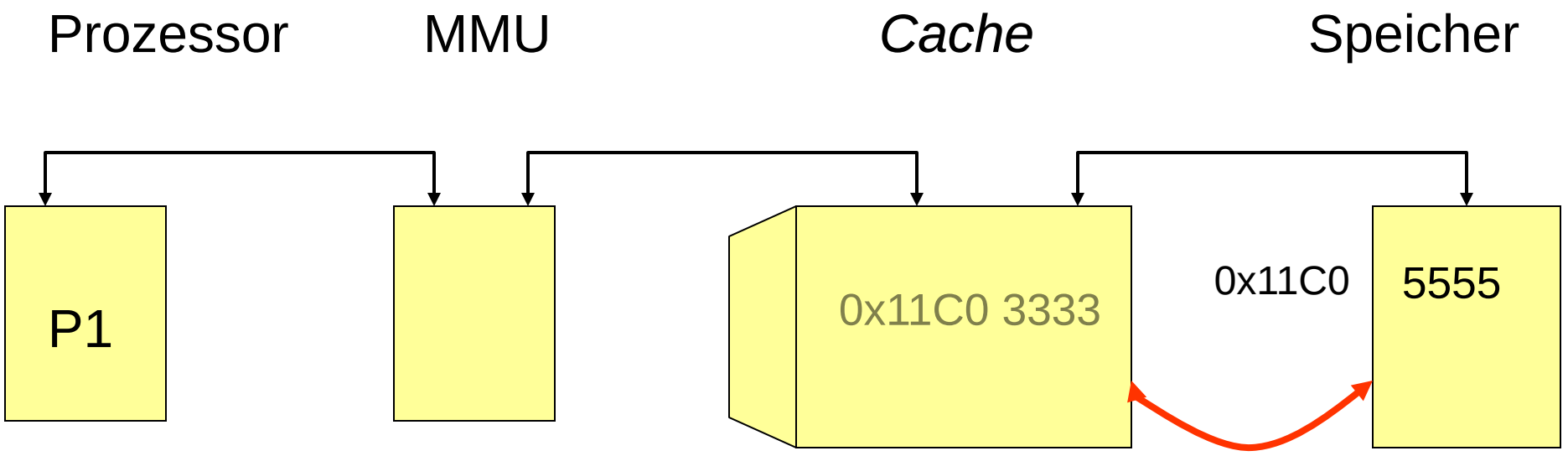
Vor Aus/Einlagern	
Seite	**Kachel**
0x88D.	0x11C.

Nach Aus/Einlagern	
Seite	**Kachel**
0x77F.	0x11C.
0x88D.	☞ Platte

☞ Bei Seitenfehlern müssen alle *Cache*-Zeilen, die Kopien der verdrängten Seite enthalten, ungültig erklärt werden.

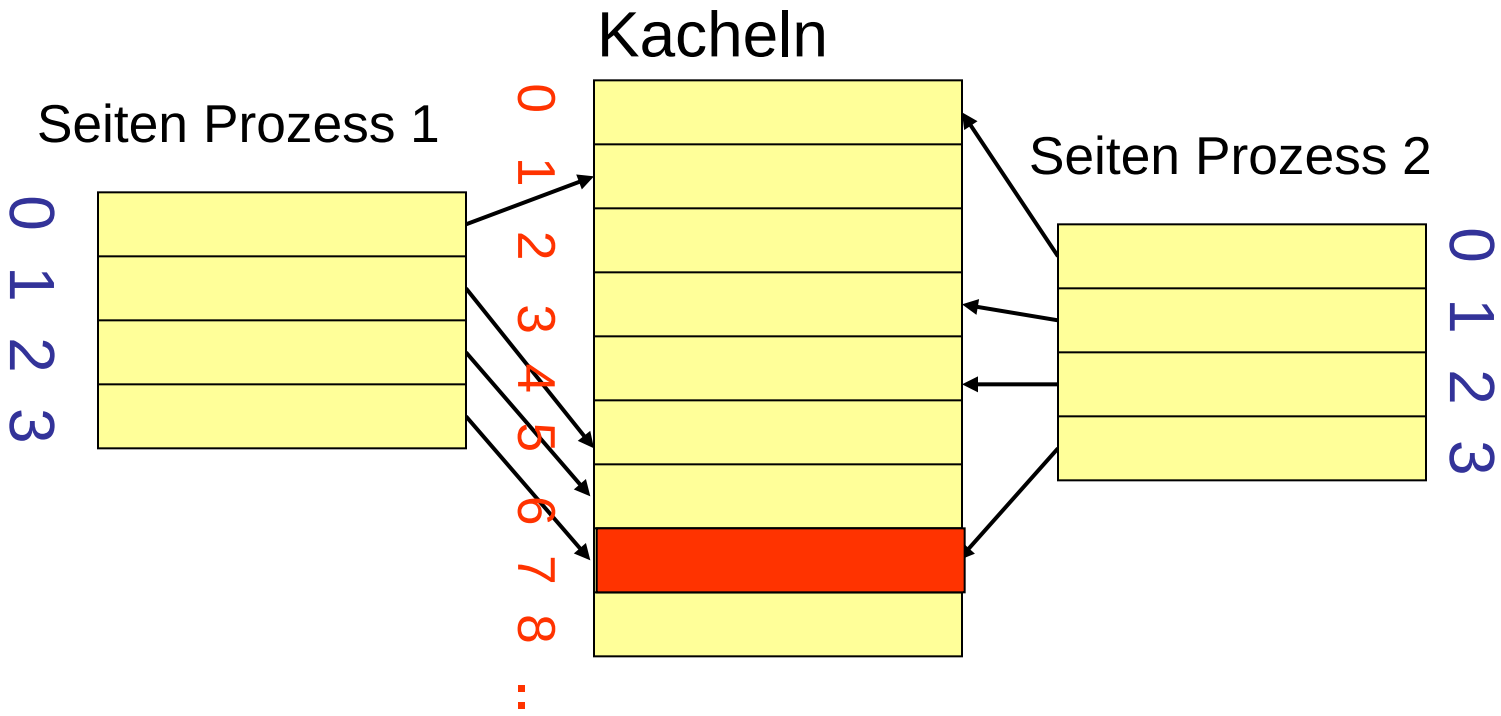
Bus Snooping (Bus-Lauschen)

☞ Bei Seitenfehlern müssen alle *Cache*-Zeilen, die Kopien der verdrängten Seite enthalten, ungültig erklärt werden



Cache „lauscht“, ob ein Schreibvorgang in den Speicher eine Zelle betrifft, von der er eine Kopie hat.
Wenn ja, dann erklärt er diese für ungültig.

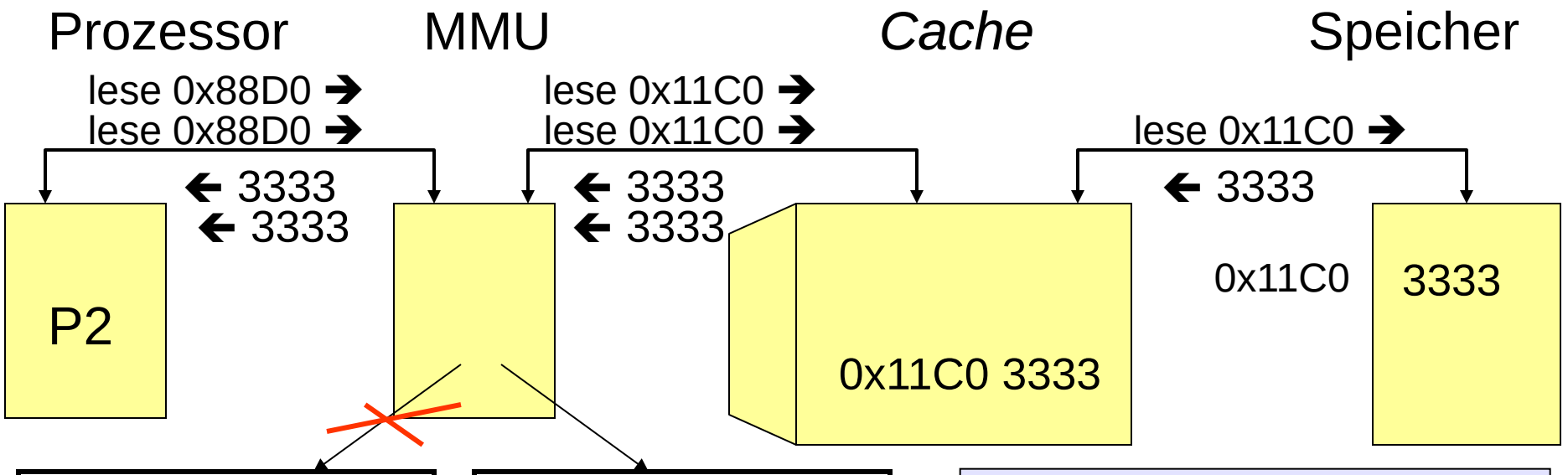
Gemeinsam genutzte Seiten *(Sharing)*



Für gemeinsame Daten oder gemeinsame Befehle:
Eintrag derselben Kachel in mehrere Seitentabellen.

Realer *Cache* bei gemeinsam benutzter Seite

Annahme *: Beide Prozesse benutzen dieselbe virtuelle Adresse für den gemeinsamen Speicherbereich



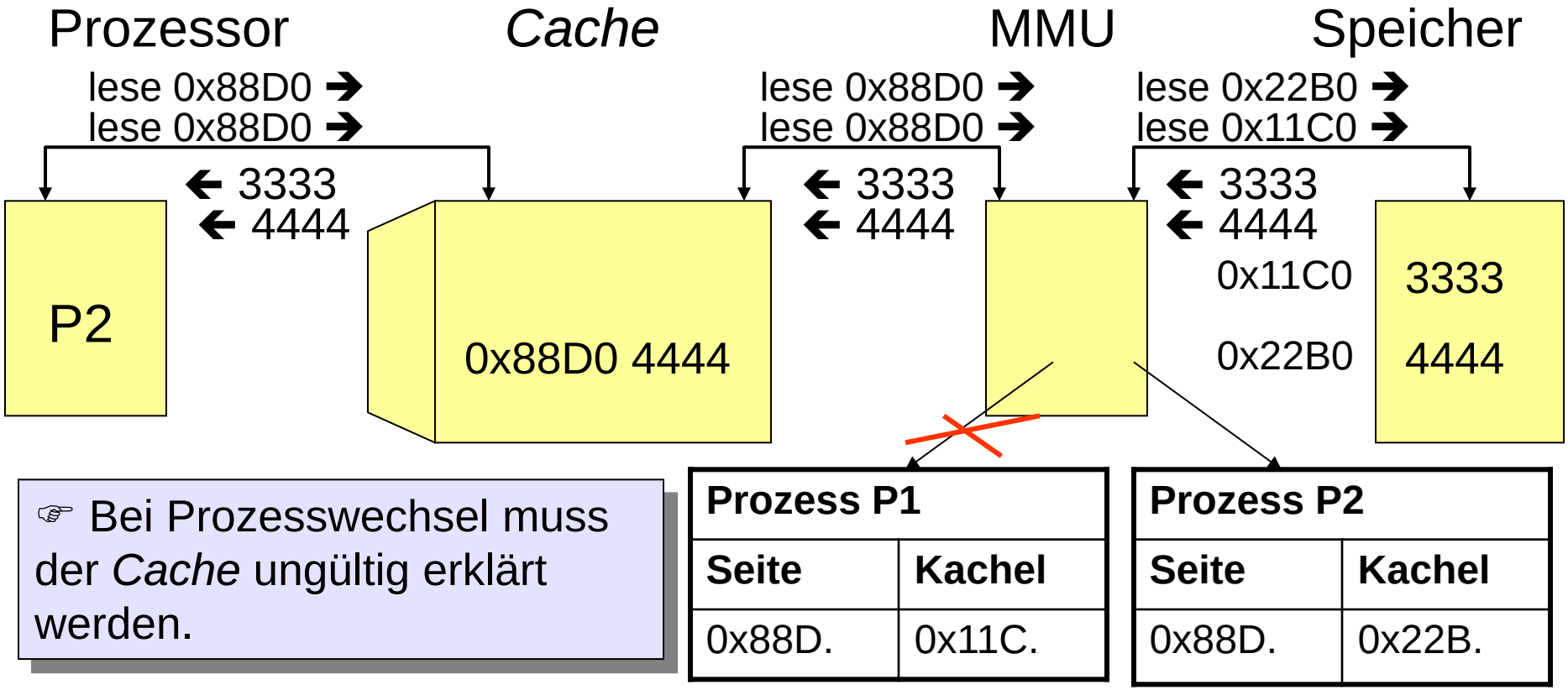
Prozess P1	
Seite	**Kachel**
0x88D.	0x11C.

Prozess P2	
Seite	**Kachel**
0x88D.	0x11C.

☞ Nach Prozesswechsel erhält P2 aus dem *Cache* die richtigen Informationen (unabhängig von *).

Virtueller *Cache* bei Prozesswechsel *(context switch)*

Zu einer bestimmten Adresse im *Cache* gehört vor und nach dem Prozesswechsel eine ***andere*** Zelle im Speicher



☞ Bei Prozesswechsel muss der *Cache* ungültig erklärt werden.

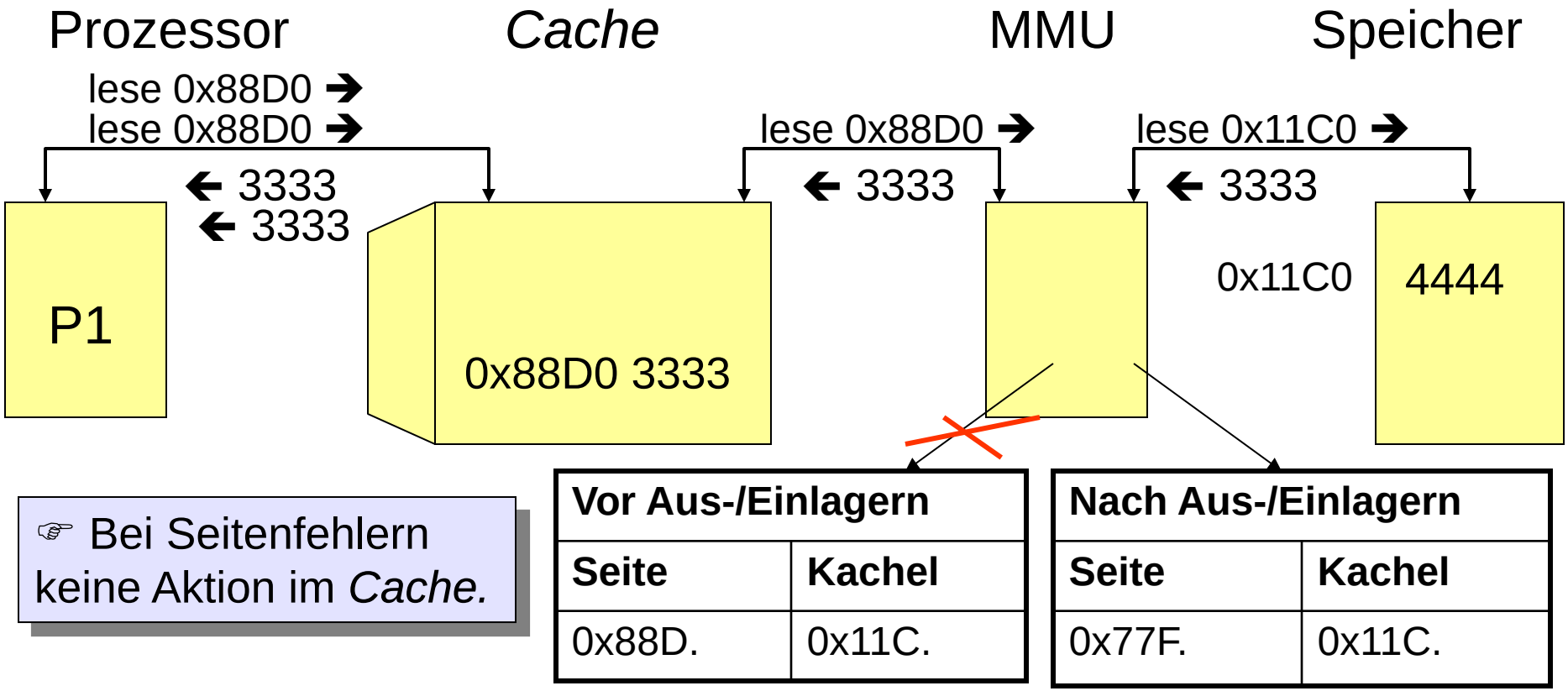
Prozess P1	
Seite	**Kachel**
0x88D.	0x11C.

Prozess P2	
Seite	**Kachel**
0x88D.	0x22B.

Bei jedem Prozesswechsel gehen alle Informationen im *Cache* verloren.
☞ Schlecht für große *Caches*!

Virtueller *Cache* bei Seitenfehler

Zu einer bestimmten Adresse im *Cache* gehört vor und nach dem Seitenfehler dieselbe Information



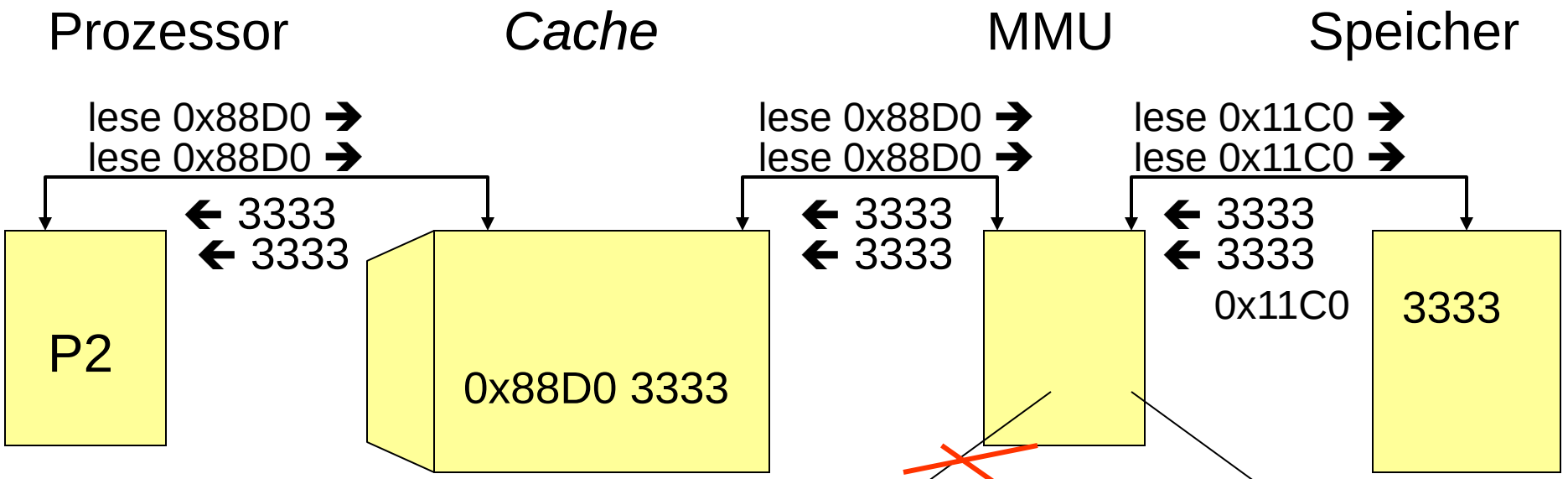
Vor Aus-/Einlagern	
Seite	**Kachel**
0x88D.	0x11C.

Nach Aus-/Einlagern	
Seite	**Kachel**
0x77F.	0x11C.

☞ Bei Seitenfehlern keine Aktion im *Cache*.

Virtueller *Cache* bei gemeinsam benutzter Seite

Annahme: Beide Prozesse benutzen dieselbe virtuelle Adresse für den gemeinsamen Speicherbereich



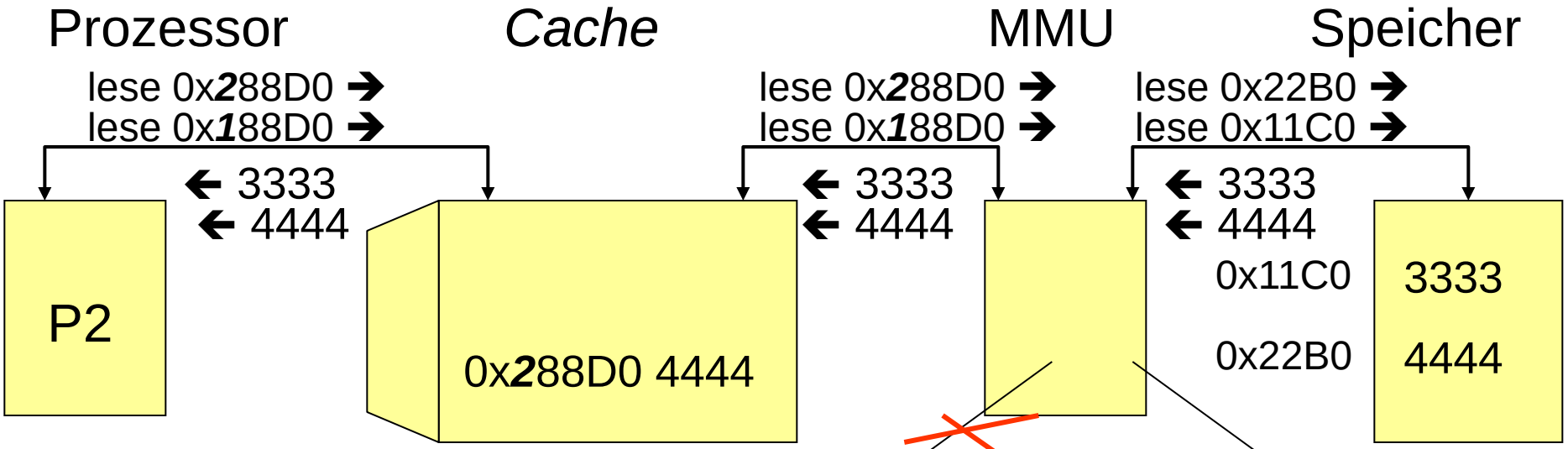
☞ Der zweite Prozess füllt den *Cache* erneut. Langsam, aber korrekt.

Prozess P1	
Seite	**Kachel**
0x88D.	0x11C.

Prozess P2	
Seite	**Kachel**
0x88D.	0x11C.

Virtueller *Cache* mit PIDs bei Prozesswechsel

Virtuelle Adresse wird um Prozessnummer *(process identifier, PID)* erweitert. Zu einer bestimmten Adresse im *Cache* gehört vor und nach dem Prozesswechsel wieder dieselbe Zelle.



☞ Bei Prozesswechsel keine Aktion im *Cache* erforderlich.

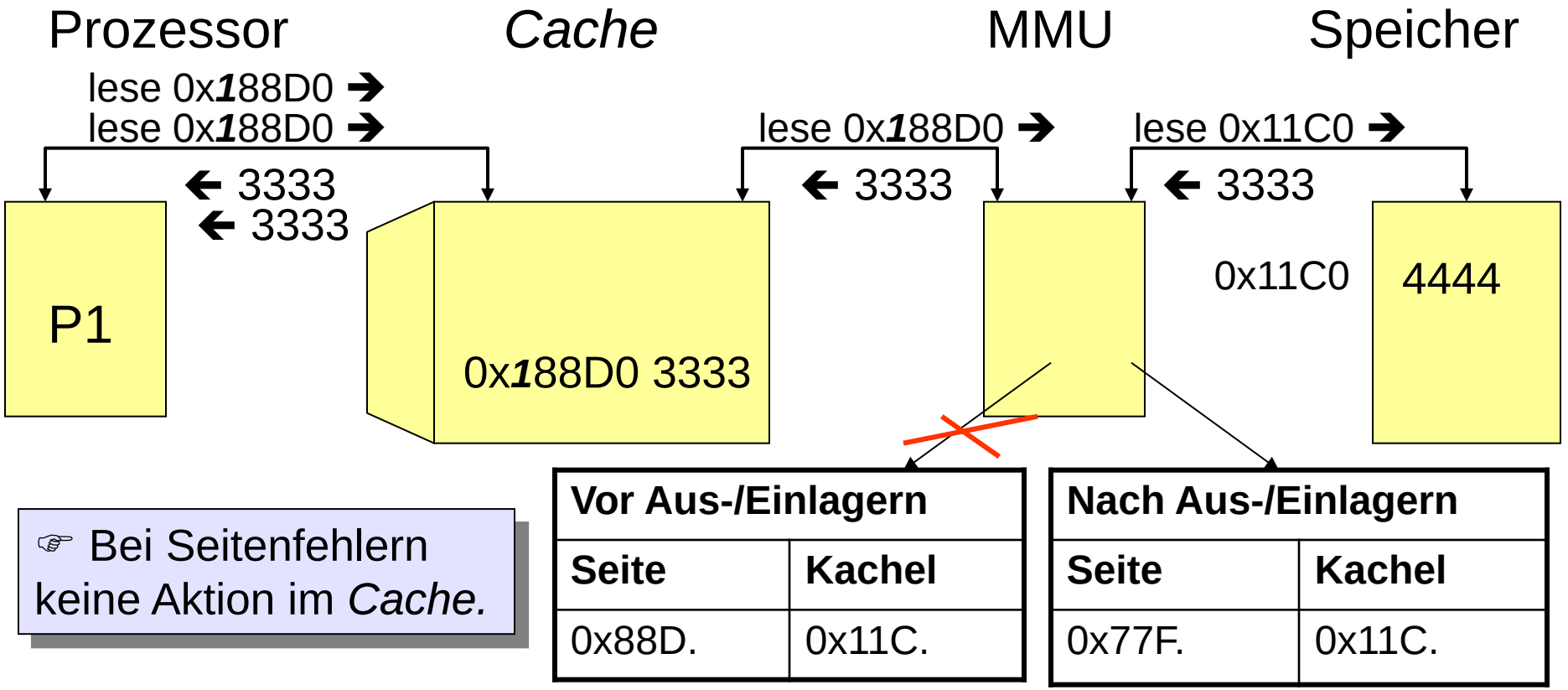
Set associative mapping ☞ Eintrag für P1 würde erhalten.
☞ Gut für große *Caches*!

Prozess P1	
Seite	**Kachel**
0x88D.	0x11C.

Prozess P2	
Seite	**Kachel**
0x88D.	0x22B.

Virtueller *Cache* mit PIDs bei Seitenfehler

Zu einer bestimmten Adresse im *Cache* gehört vor und nach dem Seitenfehler dieselbe Information

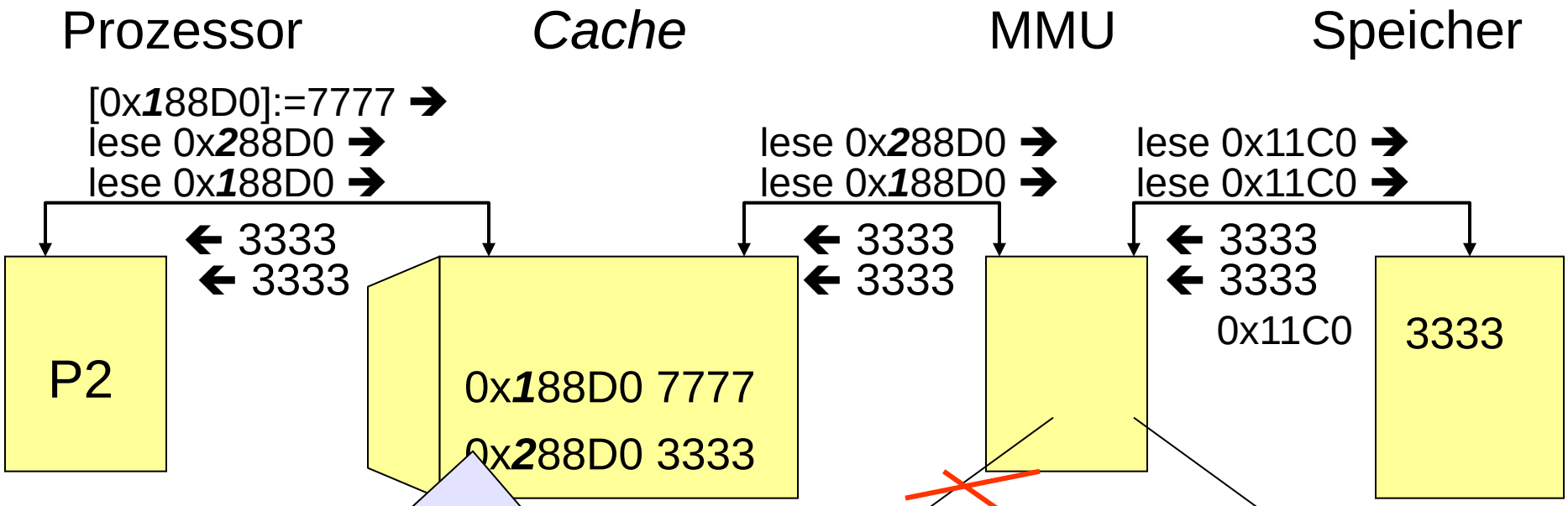


Vor Aus-/Einlagern	
Seite	**Kachel**
0x88D.	0x11C.

Nach Aus-/Einlagern	
Seite	**Kachel**
0x77F.	0x11C.

☞ Bei Seitenfehlern keine Aktion im *Cache*.

Virtueller *Cache* mit PIDs bei gemeinsam benutzter Seite



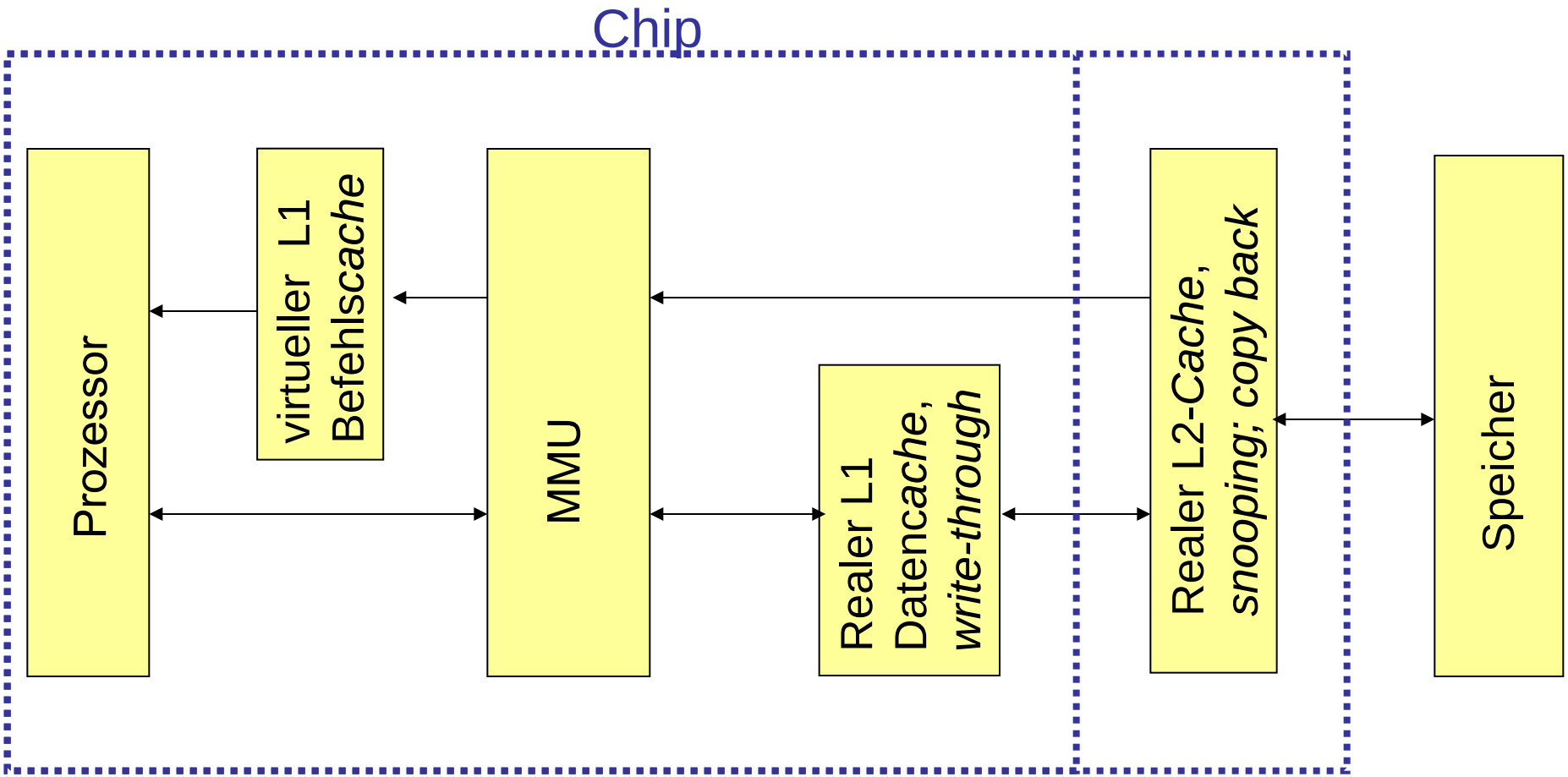
Prozess P1	
Seite	**Kachel**
0x88D.	0x11C.

Prozess P2	
Seite	**Kachel**
0x88D.	0x11C.

Einträge für P1 und P2 können beide im *Cache* existieren (z.B. bei *set-associative mapping)*; sie werden nicht abgeglichen
☞ mögliche **Inkonsistenz** bzw. **Inkohärenz**

Systeme mit mehreren *Caches*

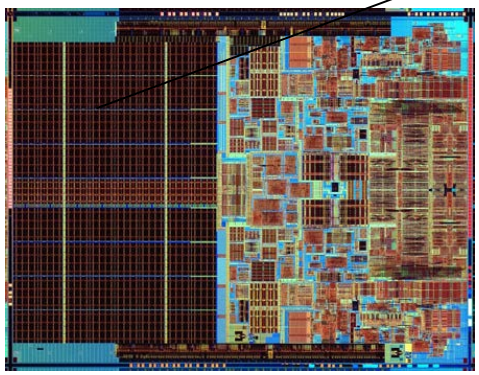
Beispiel

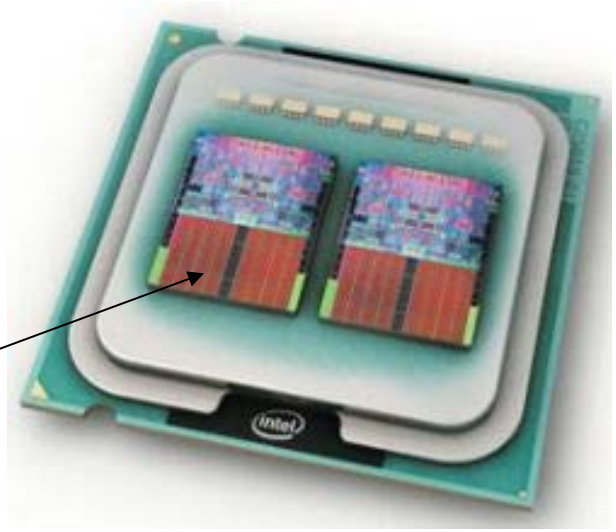


Beispiel: Aktuelle Intel-Prozessoren

Intel© Core™2 Extreme Quad-Core QX6000

- 4 x 32kB L1 *Cache*
- 2 x 4MB L2 *Cache*
- 2 Doppelprozessoren in einem Gehäuse





Austauschverfahren (1)

Überschreiben von Einträgen in TLB, *Caches*, Kacheln:

1. ***Random*- bzw. Zufallsverfahren**
   - Ein zufällig ausgewählter Eintrag wird überschrieben
   - Ziel: Belegung des *Caches* möglichst uniform verteilen
   - Sinnvoll nur dann, wenn ohnehin keine Regularität im Zugriffsverhalten erkennbar ist; sonst ungeeignet
   - Ggf. werden Pseudozufallszahlen verwendet für reproduzierbares Verhalten *(debugging!)*
   - Für Realzeitsysteme eine Katastrophe ☹
2. ***First in, first out (FIFO)***

   Ältester Block wird ersetzt (auch wenn gerade benutzt)

Austauschverfahren (2)

3. Clock

Alle TLB/*Cache*-Einträge sind gedanklich im Kreis auf einem Ziffernblatt angeordnet, ein Zeiger **p** wird „im Uhrzeigersinn“ weiterbewegt und zeigt den zu ersetzenden Eintrag an.

Jeder TLB/*Cache*-Eintrag hat ein *used bit*, das automatisch durch die TLB/*Cache*-Hardware auf ‚1‘ gesetzt wird, wann immer auf den Eintrag zugegriffen wird (bspw. die *access bits* der Pentium MMUs).

Das *Clock*-Verfahren sucht nach Eintrag mit *used bit* = ‚0‘ bei gleichzeitigem Rücksetzen des Bits. Stop nach max. 1 Runde

Austauschverfahren (3)

***3.* *Clock*-Algorithmus**

a) Wenn zu ersetzender TLB/*Cache*-Eintrag gesucht wird:

b) Zeigt Zeiger **p** auf Eintrag mit *used bit* = ‚1‘?

c) Falls ja: Setze *used bit* des Eintrags **p** auf ‚0‘; bewege Zeiger um eine Position im Uhrzeigersinn weiter (**p = p + 1**); gehe zu b)

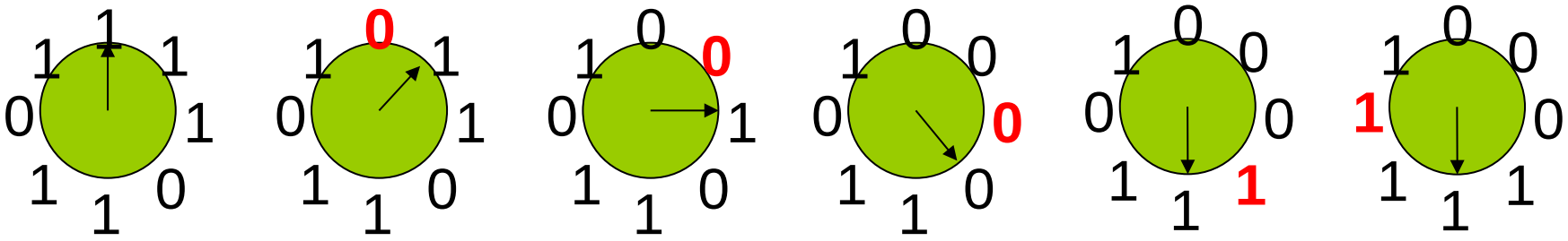
d) Falls nein: Bewege Zeiger um eine Position im Uhrzeigersinn weiter (**p = p + 1**); gebe *alten* Zeigerwert **p-1** als zu ersetzenden Eintrag zurück

Wichtig: Wird auf einen Eintrag zugegriffen, der bereits im TLB/*Cache* enthalten ist *(TLB/Cache Hit)*, wird obiger Algorithmus nicht ausgeführt. Das *used bit* des gefundenen Eintrags wird automatisch durch die Hardware auf ‚1‘ gesetzt, und der („Uhr-“) Zeiger **p** wird *nicht* verändert.

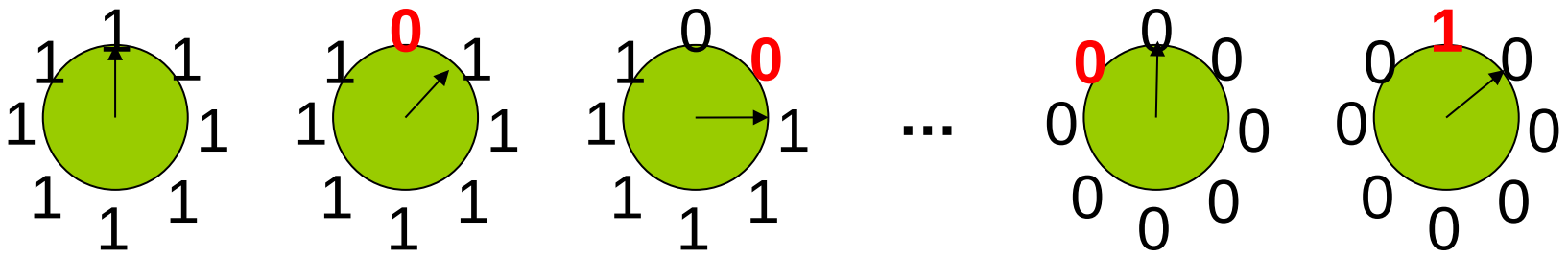
Austauschverfahren (4)

3. Clock

Beispiel 1: Manche Einträge unbenutzt, Platz benötigt



Beispiel 2: Alle Einträge benutzt, Platz benötigt



Austauschverfahren (5)

4. **NRU, *not recently used***

   Einteilung von Einträgen in 4 Klassen mittels *used* und *modified bit*:

   a. Nicht benutzte, nicht modifizierte Einträge
   b. Nicht benutzte, modifizierte Einträge

      Benutzt-Kennzeichnung wird nach einiger Zeit per *Timer* zurückgesetzt
   c. Benutzte, nicht modifizierte Einträge
   d. Benutzte, modifizierte Einträge

   Überschrieben wird ein zufällig ausgewählter Eintrag der niedrigsten Klasse, die nicht leer ist

Austauschverfahren (6)

5. **LRU, *least recently used***
   - Der am längsten nicht benutzte Block wird ersetzt
   - Realisierung erfordert einen gewissen Aufwand
   - Für Realzeitsysteme die beste Lösung ☺

LRU mit Verketteten Listen

Datenstruktur: verkettete Liste von Einträgen, sortiert nach der Reihenfolge des Zugriffs.

Operationen:

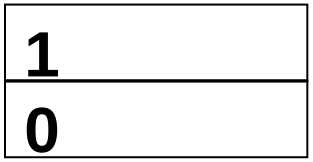
- Zugriff auf Eintrag *i*:
  - *i* wird am Anfang der Liste eingefügt und,
  - falls *i* sonst noch in der Liste vorkommt, dort entfernt.

- Verdrängen eines Eintrags:
  - Der Eintrag, der zum letzten Element der Liste gehört, wird überschrieben.

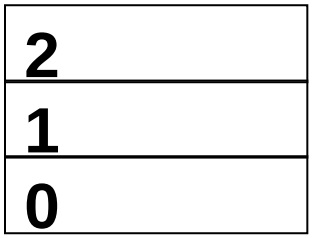
LRU-Beispiel: Dynamischer Ablauf (1)

Zugriff auf *tag*=0

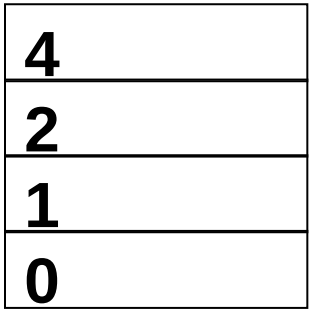
Zugriff auf *tag*=1



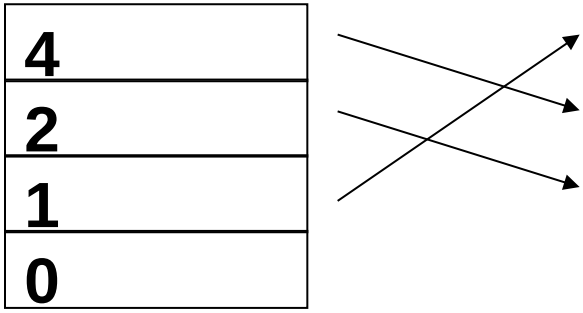
Zugriff auf *tag*=2



Zugriff auf *tag*=4



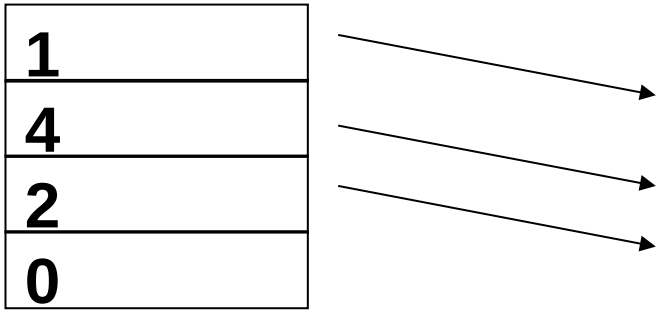
Zugriff auf *tag*=4



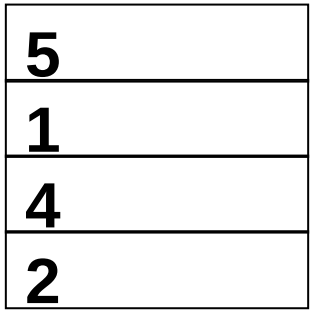
Zugriff auf *tag*=1

LRU-Beispiel: Dynamischer Ablauf (2)

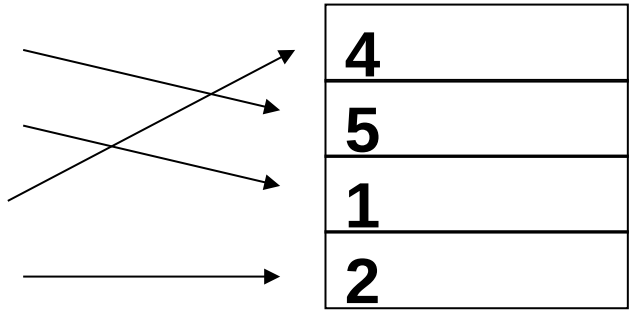
lt. letzter Folie



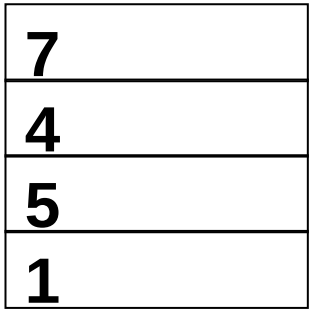
Zugriff auf *tag*=5



Zugriff auf *tag*=4



Zugriff auf *tag*=7



Zusammenfassung

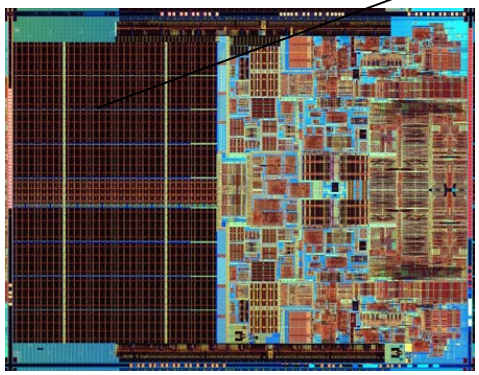
- *Cache*-Zeilen: Einheit der Gültigkeitsprüfung
- *Cache*-Blöcke: Einheit des Nachladens
- Organisationsformen:
  - *Direct mapping*
  - *Set-associative mapping*
- Unterscheidung zwischen virtuellen und realen *Caches*
- Moderne Systeme besitzen mehrere *Cache*-Ebenen

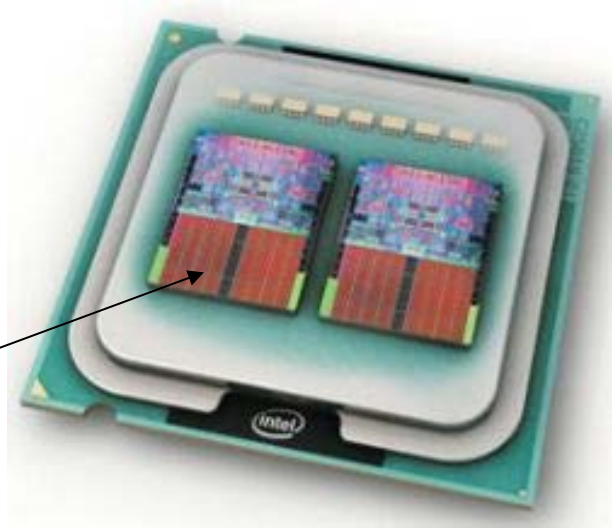
Moment einmal – Gemeinsam benutzte *(shared) Caches*?

Intel© Core™2 Extreme Quad-Core QX6000

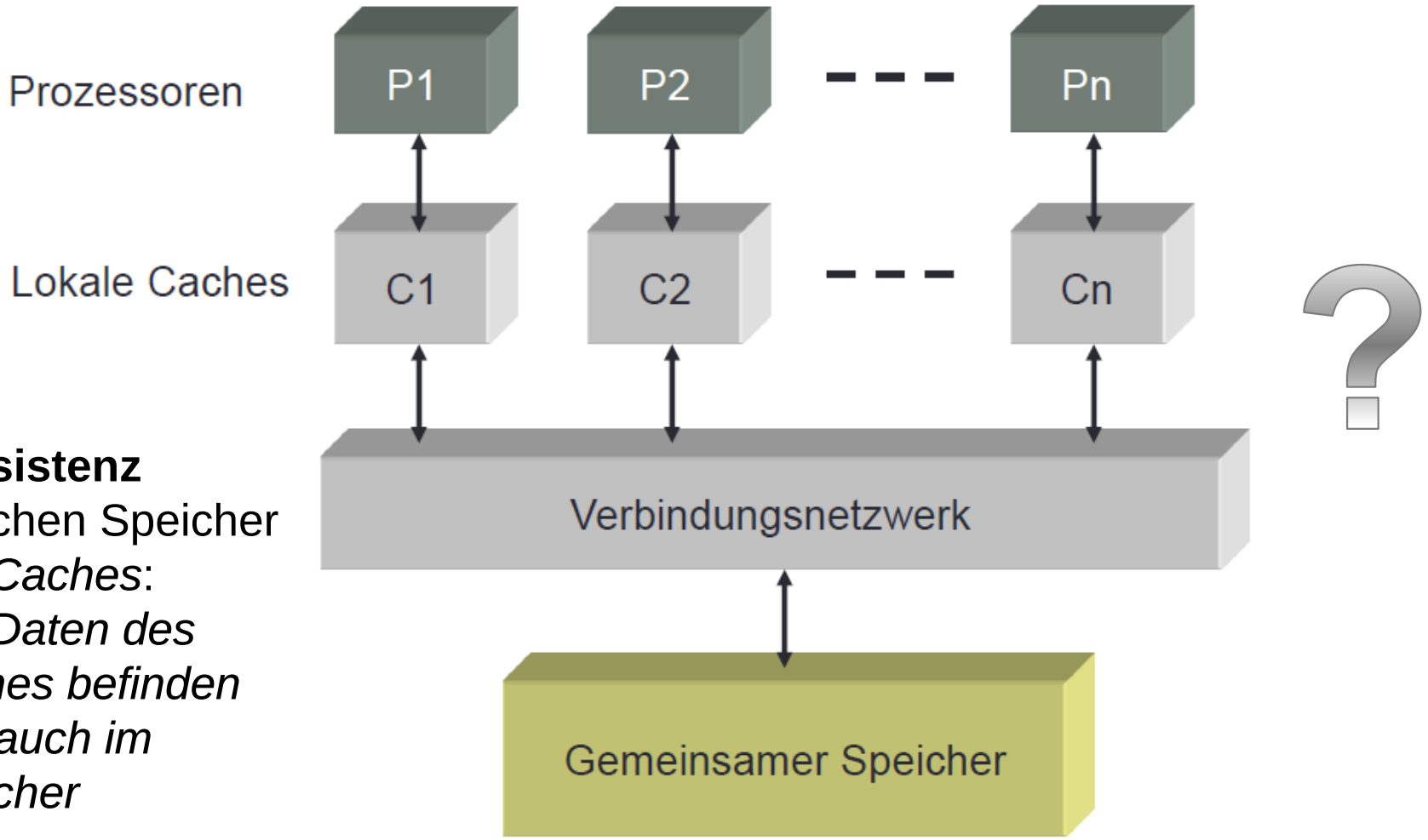
- 4 x 32kB L1 Cache
- 2 x 4MB L2 Cache
- 2 Doppelprozessoren in einem Gehäuse

Es gibt unterschiedliche Anzahlen von L1- und L2-Caches für einen Quadcore-Prozessor.
Wie geht das…?





Reales *Shared Memory System*



Konsistenz zwischen Speicher und *Caches*: *Alle Daten des Caches befinden sich auch im Speicher*

Cache(daten)-Kohärenz (1)

Daten-Kohärenz

- Sagt aus, welcher Wert beim Lesen abgeliefert wird
- Bezug auf Lesen und Schreiben ein- und derselben Speicherzelle

Cache(daten)-Kohärenz (2)

Definition: Ein Speichersystem heißt kohärent, wenn

– bei einem Schreiben einer Zelle *x* durch einen Prozessor, welches von einem Lesen derselben Zelle gefolgt wird, das Lesen immer den geschriebenen Wert abliefert,
sofern zwischen beiden Operationen kein Schreiben eines anderen Prozessors erfolgt;

– Bei einem Schreiben einer Zelle *x* durch einen Prozessor *P*, welches von einem Lesen derselben Zelle durch einen Prozessor *P'* gefolgt wird, das Lesen immer den geschriebenen Wert abliefert,
sofern zwischen beiden Operationen kein Schreiben eines anderen Prozessors erfolgt
und sofern zwischen beiden Operationen hinreichend viel Zeit vergeht;

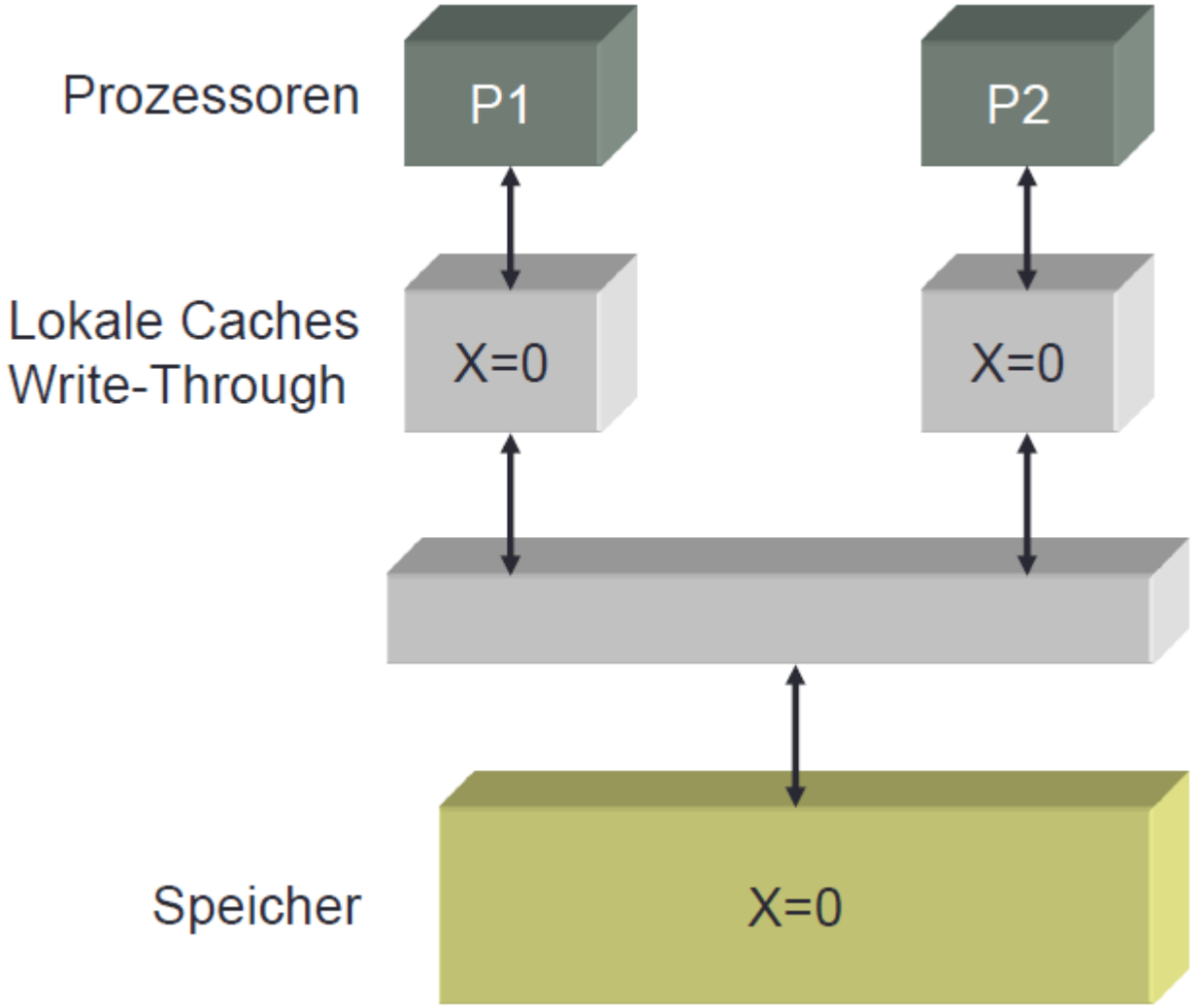
Cache(daten)-Kohärenz (3)

Definition: Ein Speichersystem heißt kohärent, wenn (Forts.)

– Schreibvorgänge in die selbe Zelle serialisiert werden, d.h. zwei Schreibvorgänge durch zwei Prozessoren werden durch die übrigen Prozessoren in derselben Reihenfolge gesehen.

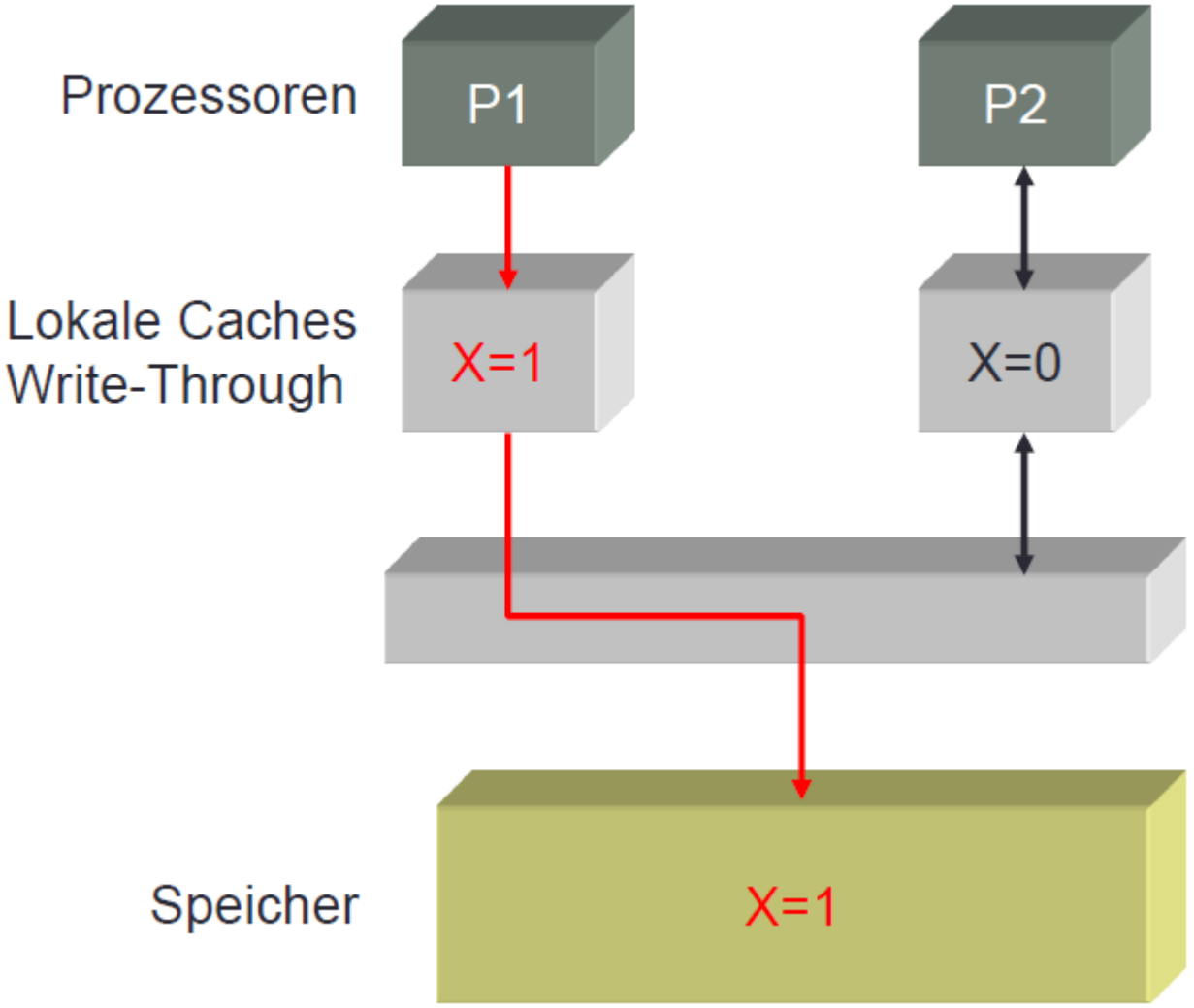
Cache-Kohärenz: Beispiel 1

Variable *X* befindet sich in den *Caches* von *P*1, *P*2 und im Hauptspeicher: kohärente Ausgangssituation



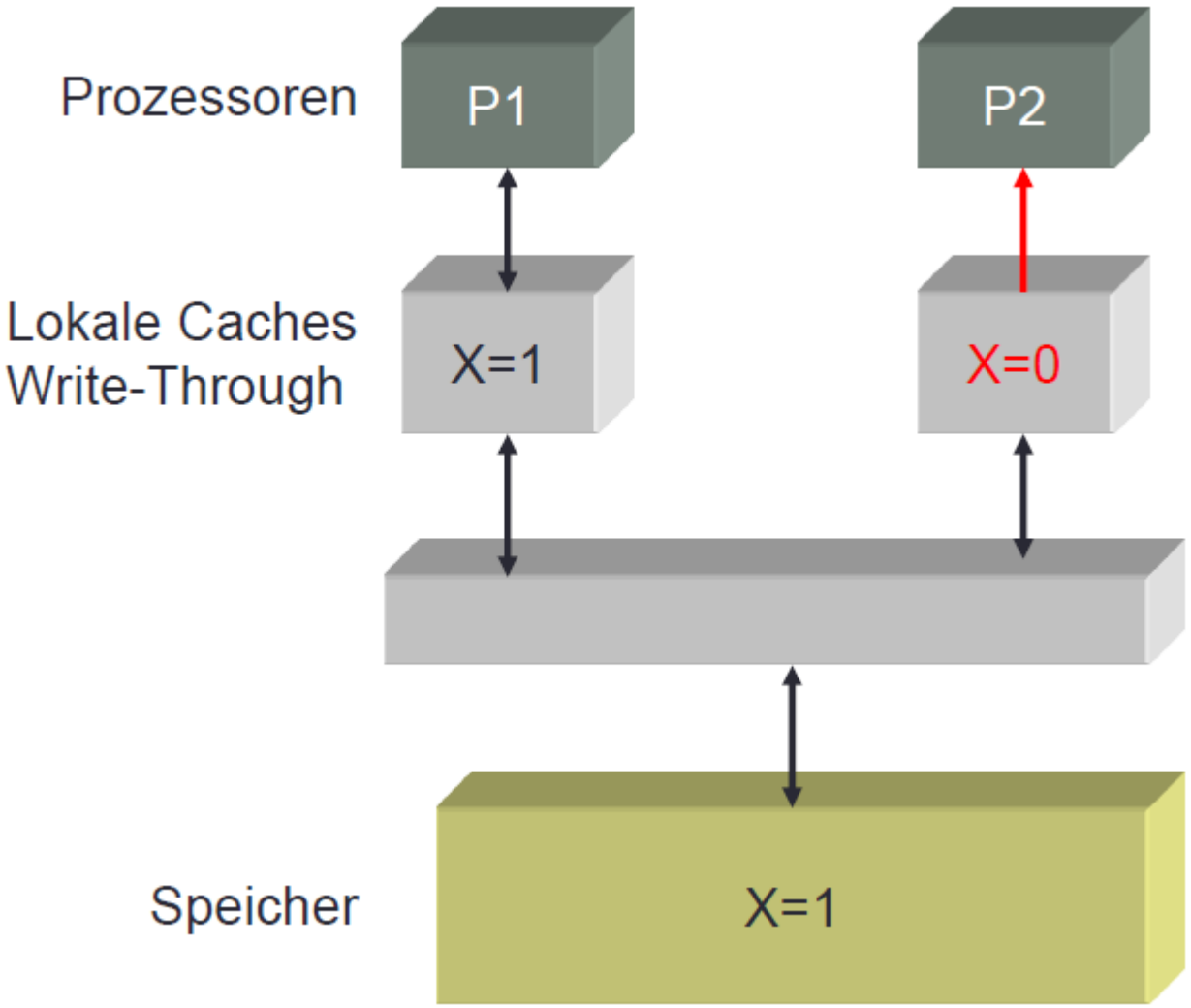
Cache-Kohärenz: Beispiel 1

*P*1 schreibt *X* = 1 in den *Cache* und in den Hauptspeicher



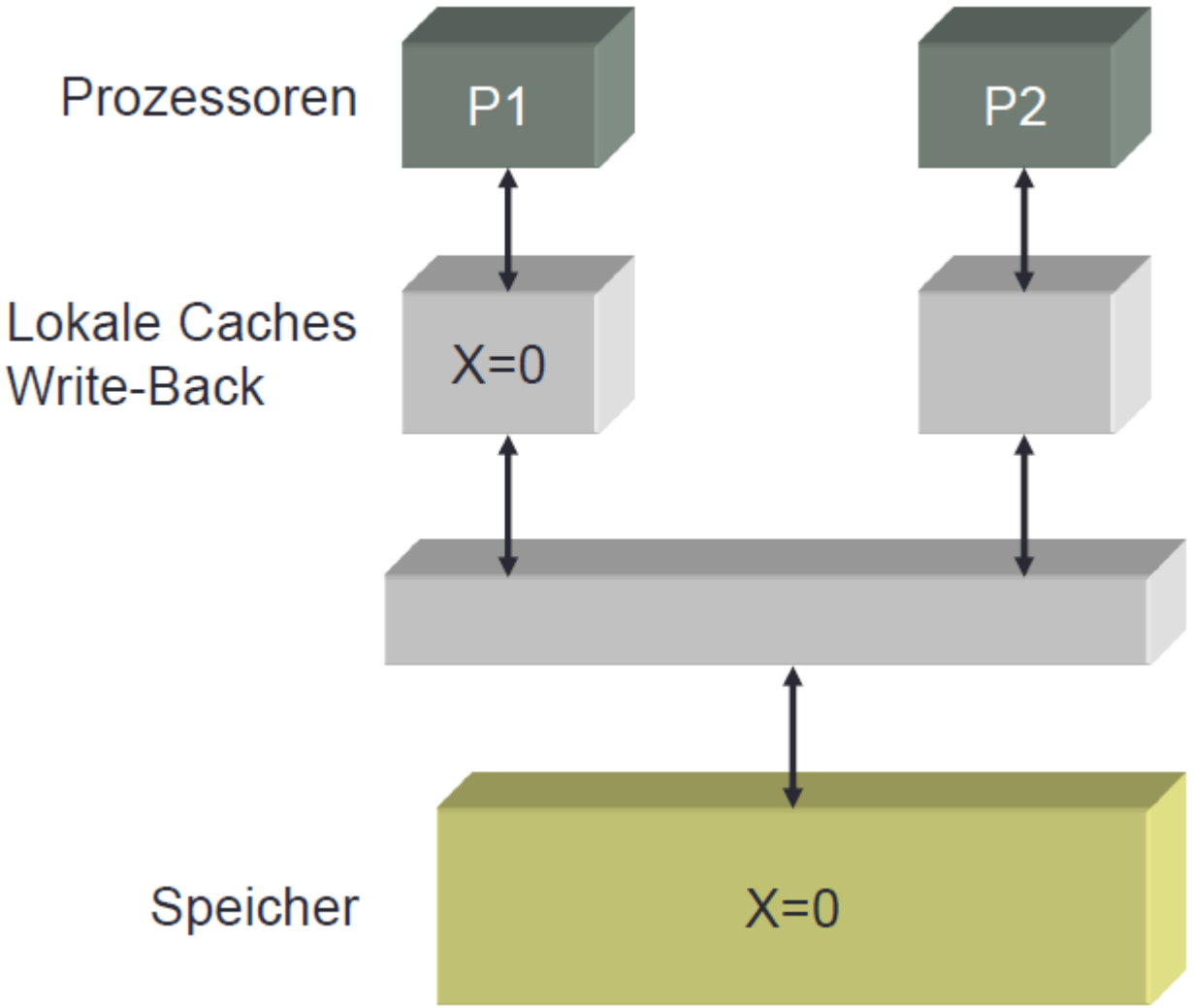
Cache-Kohärenz: Beispiel 1

*P*2 liest alten Wert aus *Cache*: inkohärentes Ergebnis



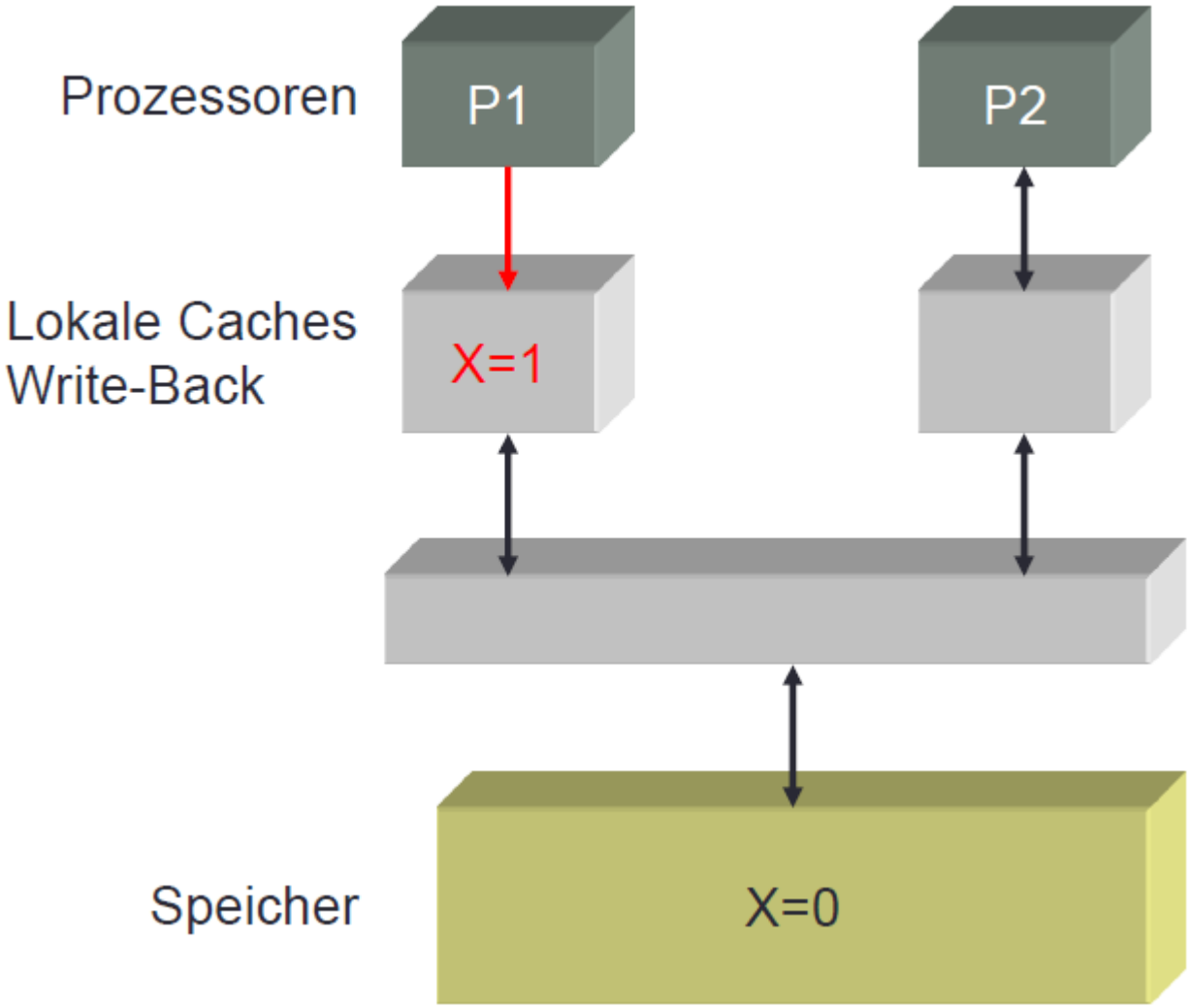
Cache-Kohärenz: Beispiel 2

Variable *X* befindet sich im *Cache* von *P*1 und im Hauptspeicher: kohärente Ausgangssituation



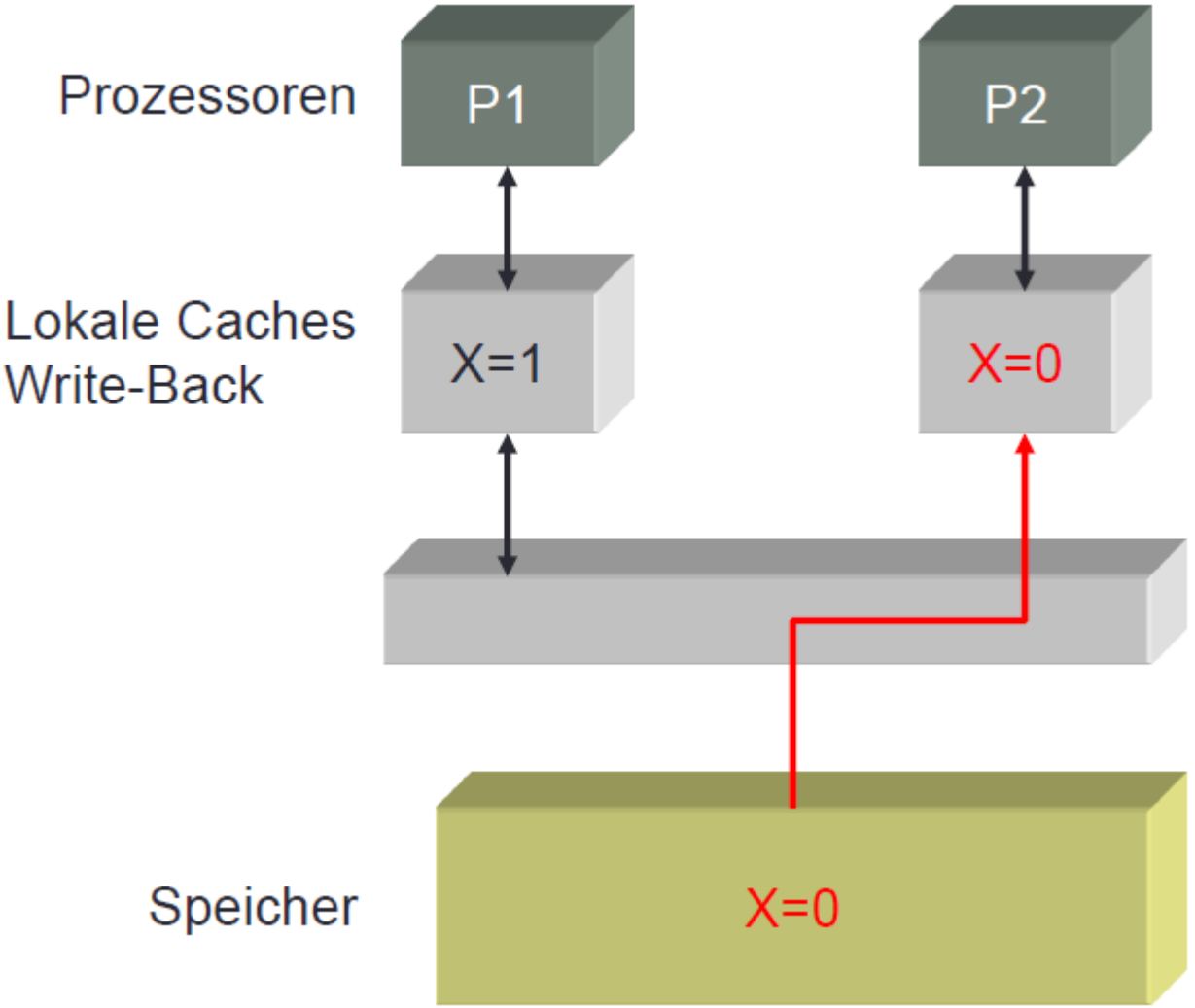
Cache-Kohärenz: Beispiel 2

*P*1 schreibt *X* = 1 nur in den *Cache*



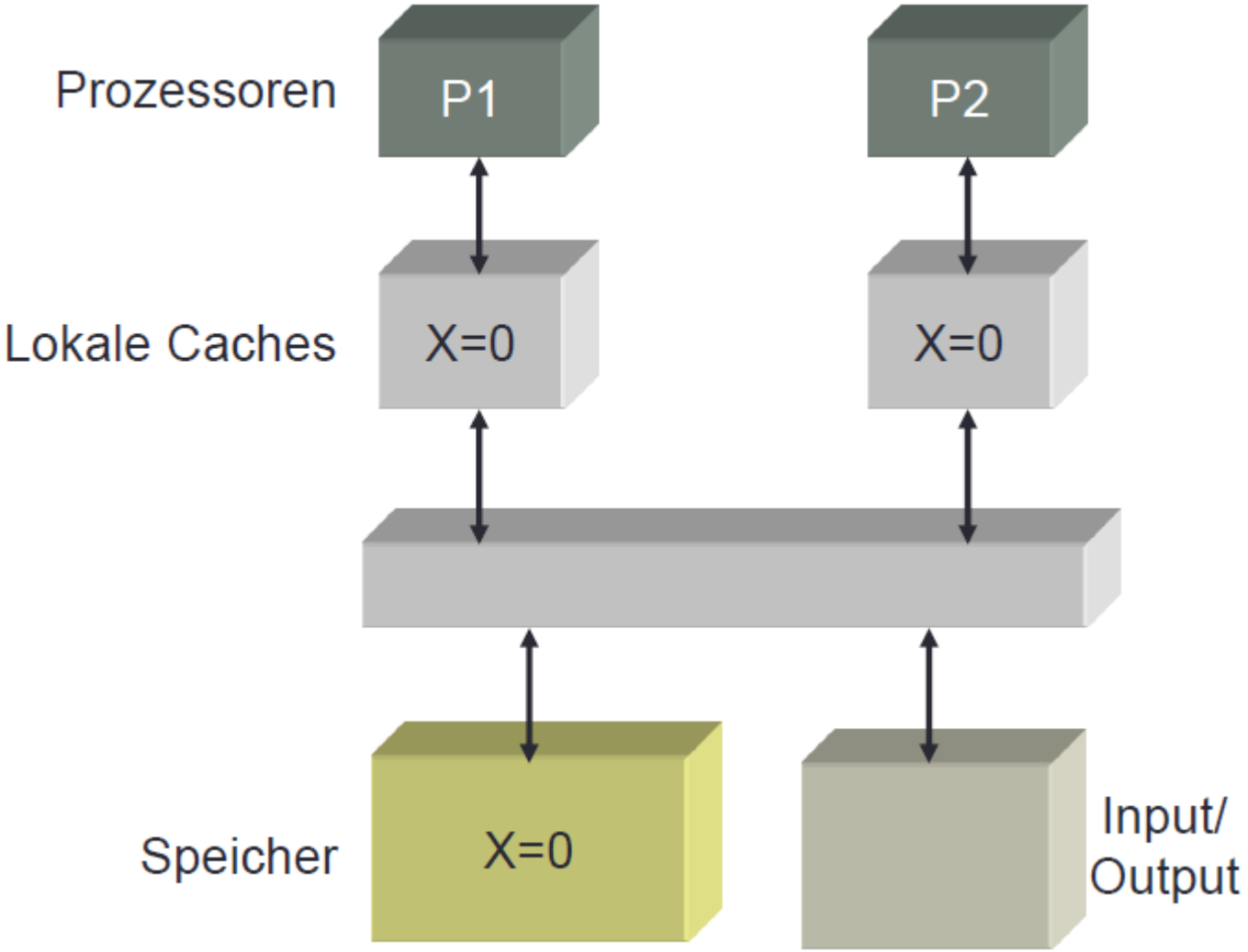
Cache-Kohärenz: Beispiel 2

*P*2 liest alten Wert aus Hauptspeicher: inkohärentes Ergebnis



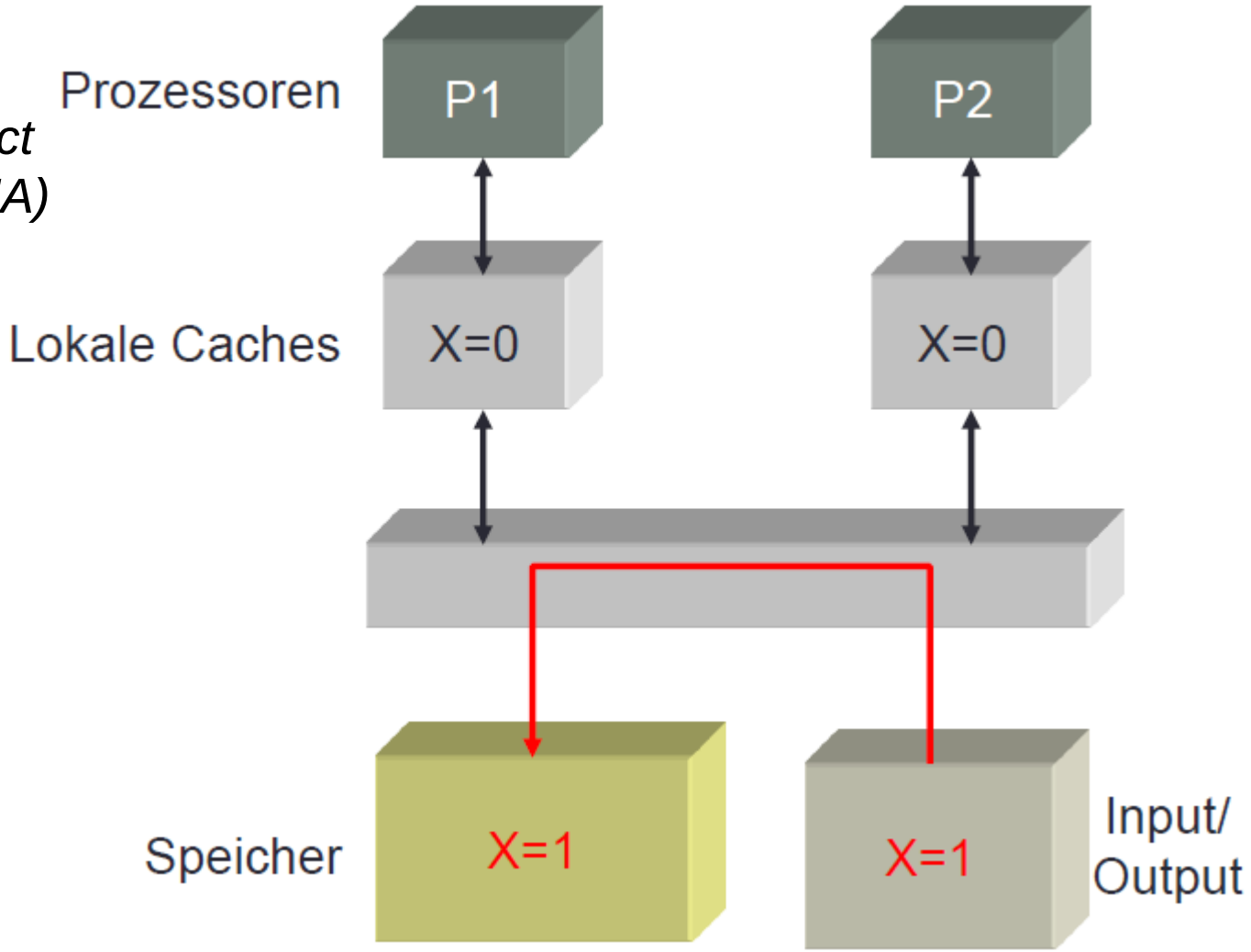
Cache-Kohärenz: Beispiel 3

Kohärente Ausgangssituation



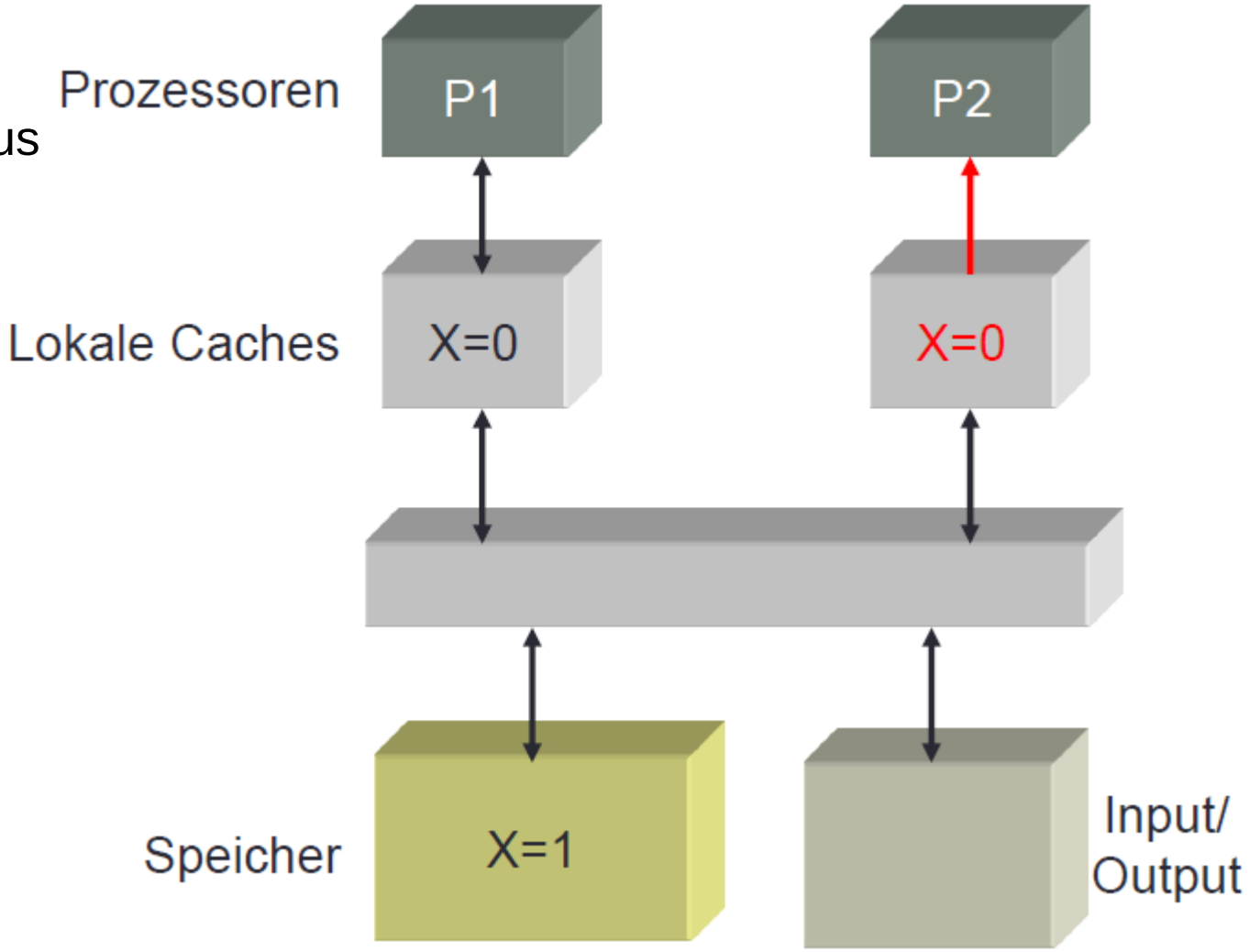
Cache-Kohärenz: Beispiel 3

Einlesen mittels *Direct Memory Access (DMA)*



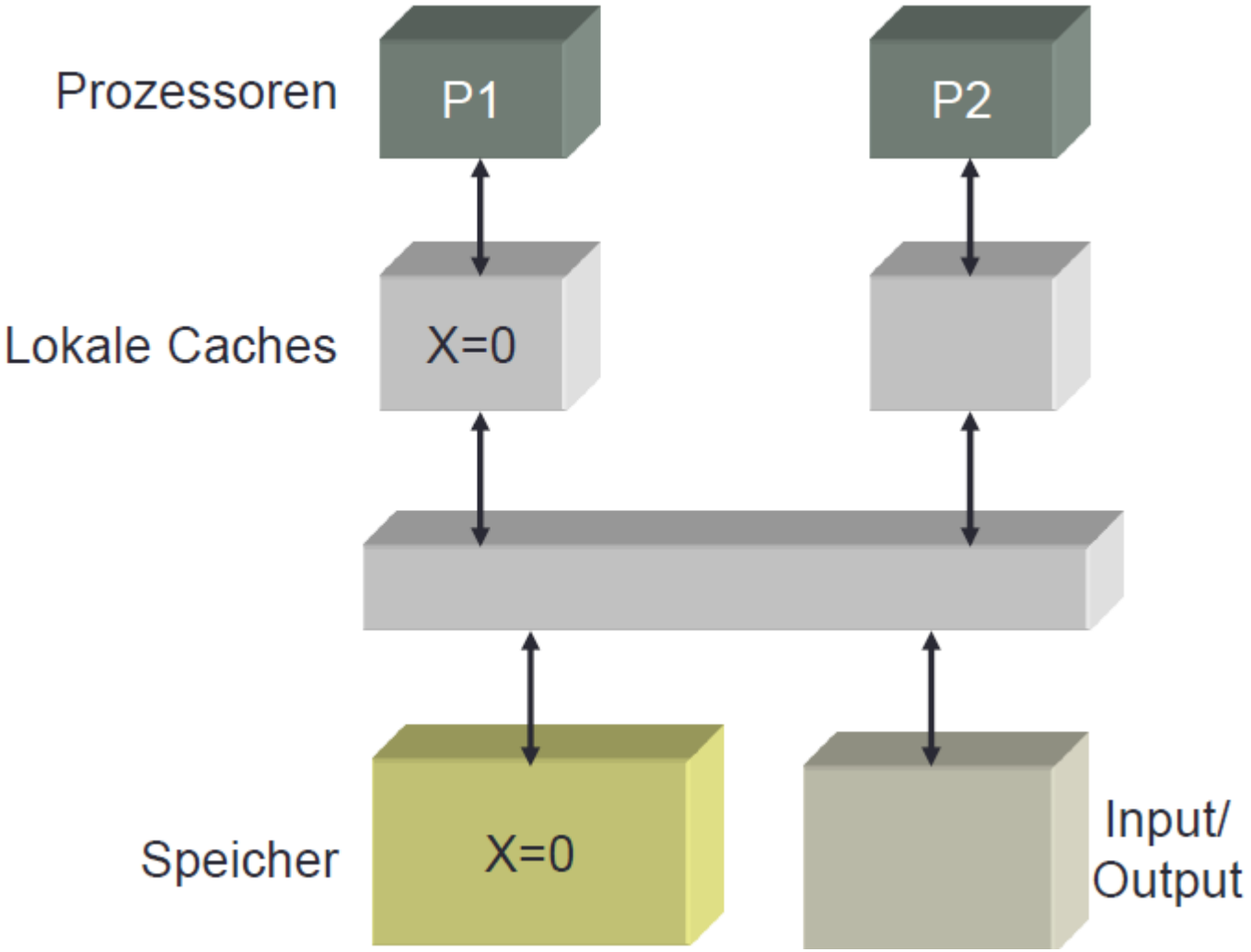
Cache-Kohärenz: Beispiel 3

*P*2 liest alten Wert aus *Cache*: inkohärentes Ergebnis



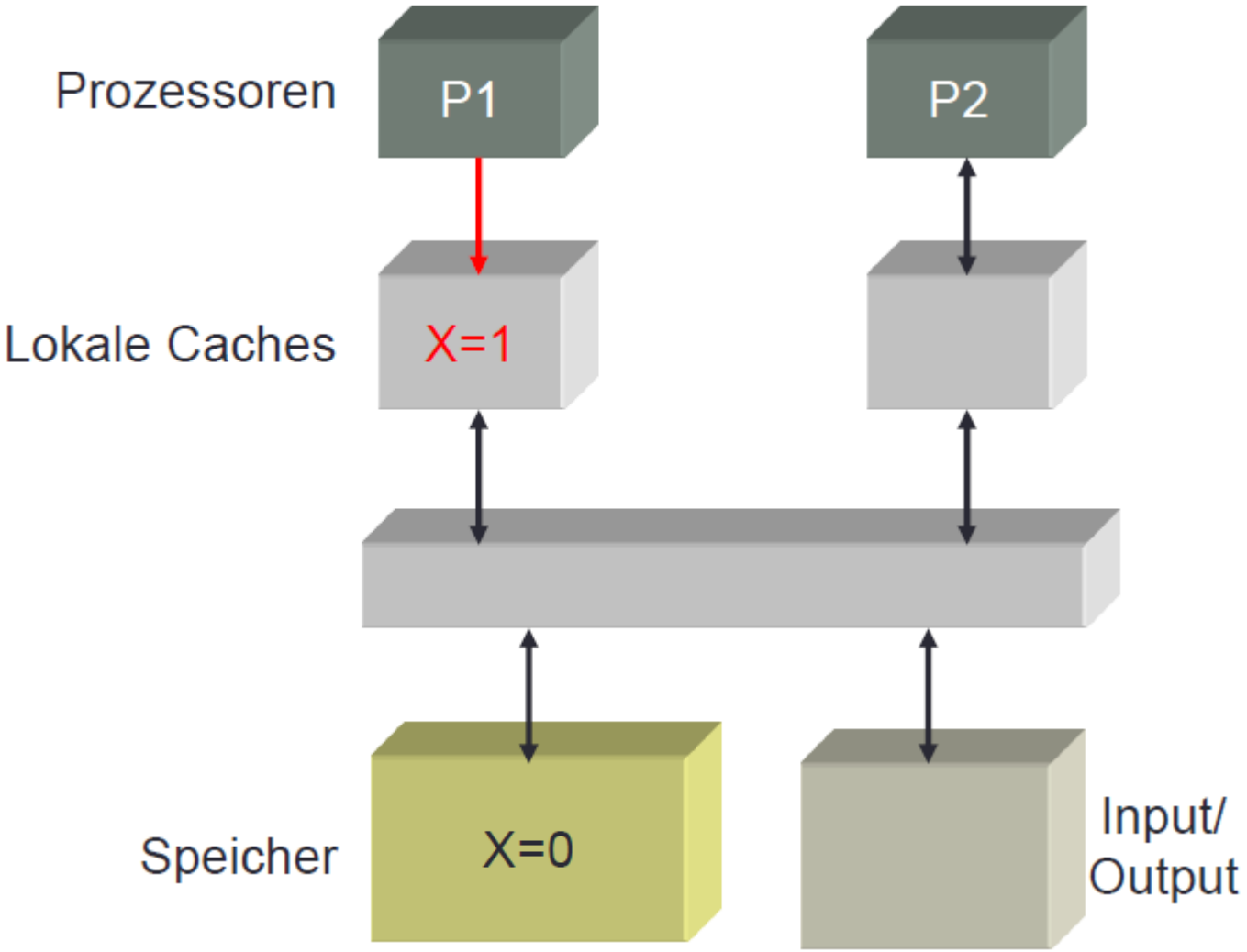
Cache-Kohärenz: Beispiel 4

Kohärente Ausgangssituation



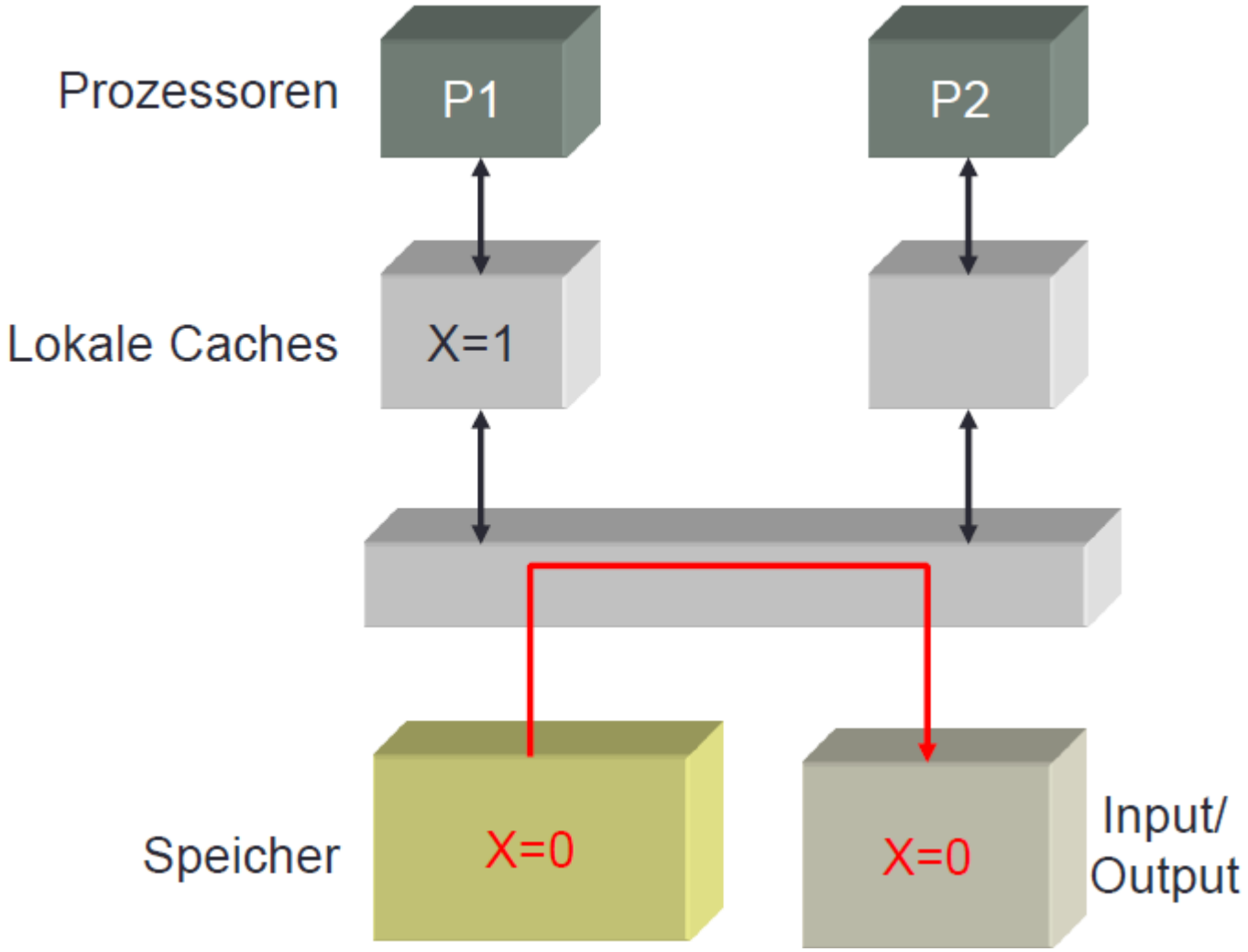
Cache-Kohärenz: Beispiel 4

*P*1 modifiziert *X* im *Write-Back Cache*



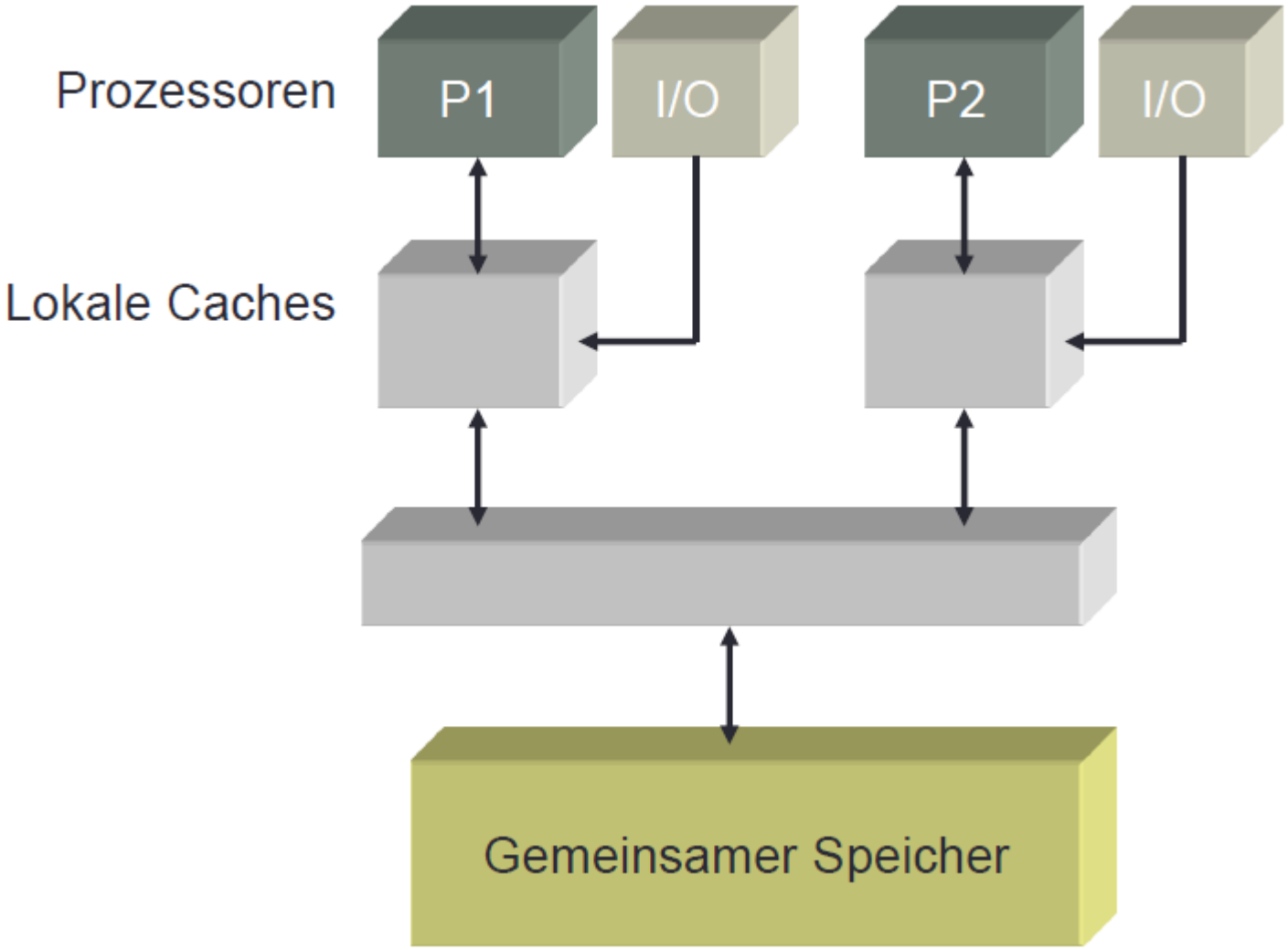
Cache-Kohärenz: Beispiel 4

Inkonsistente Daten werden ausgegeben



Lösung des I/O-Problems

Zuordnung einer I/O-Einheit zu jedem Prozessor



Lösungsmethoden

Hardware-Lösung (I/O-Problem)

– Aufwändig, schlechte Lokalität der Daten

Gemeinsamer *Cache* für alle Prozessoren

– Hoher Hardware-Aufwand, geringe Effizienz

Unterscheidung in *cacheable* und *non-cacheable* Daten

– Hoher Aufwand (Programmierer, Compiler)

***Cache*-Kohärenzprotokolle**

– *Snooping*-Protokolle

– *Directory*-Protokolle

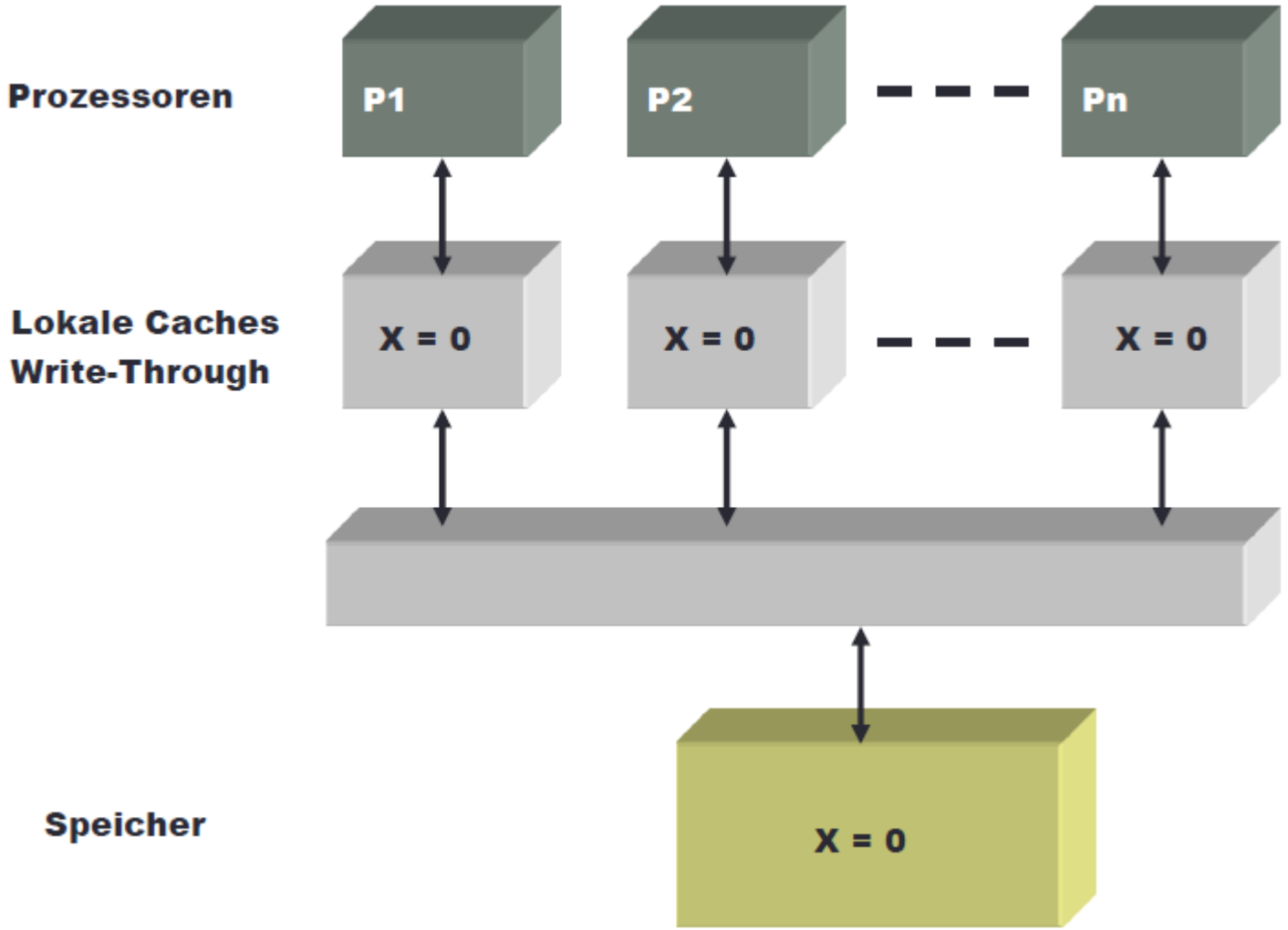
Snooping-Protokolle

Alle Prozessoren beobachten alle Datenübertragungen von jedem *Cache* zum Hauptspeicher.
Voraussetzung: broadcastfähiges Verbindungsnetzwerk

Implementierungen

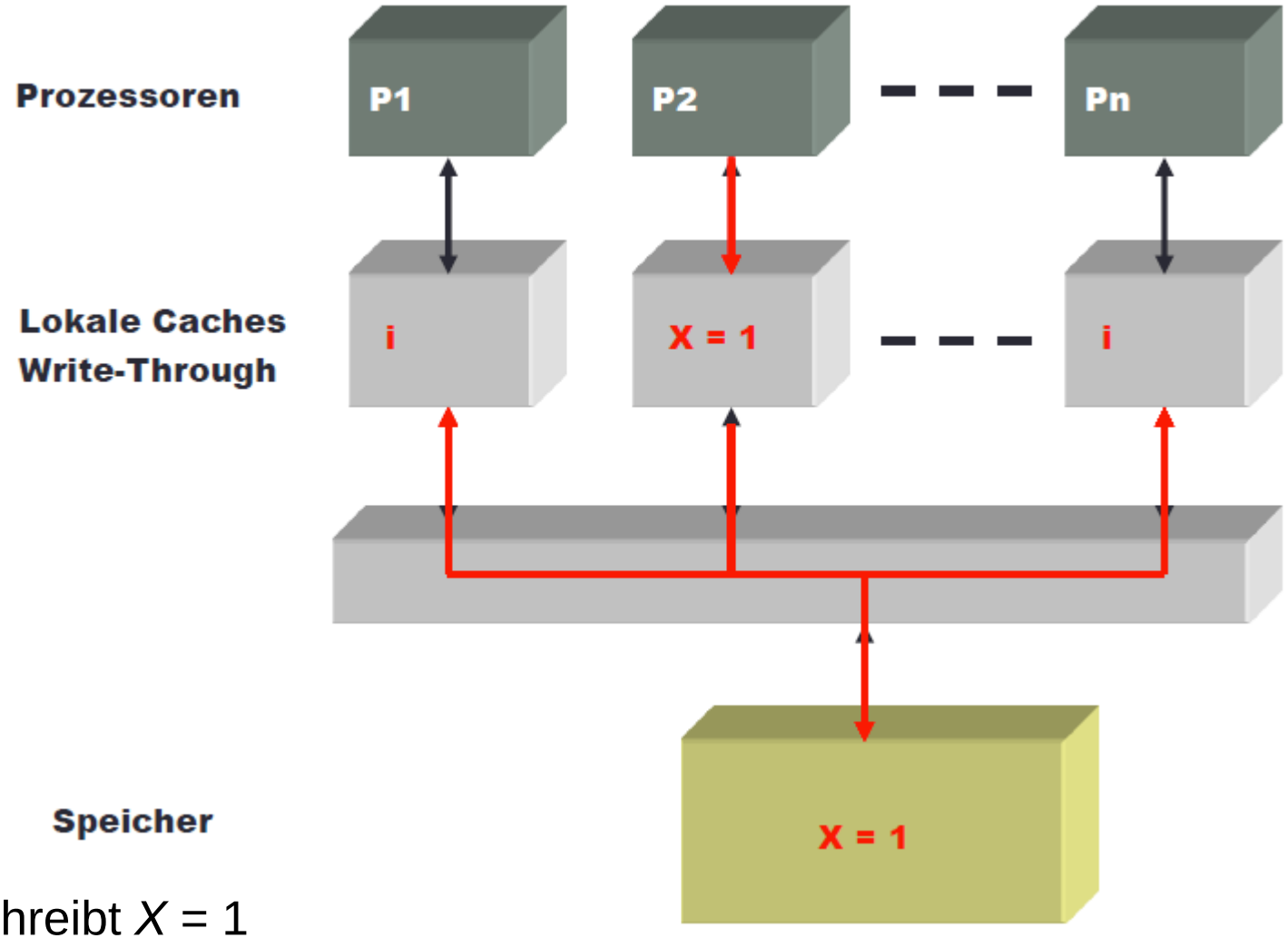
– *Write Invalidate*
  Das Verändern eines Blocks im Speicher führt zur Invalidierung aller *Cache*-Kopien mit der gleichen Adresse
– *Write Update / Write Broadcast*
  Das Verändern eines Blocks im Speicher führt zur Modifikation aller anderen *Cache*-Blöcke mit der gleichen Adresse

Write-Through Cache – Write Invalidate Protokoll



Kohärente Ausgangssituation

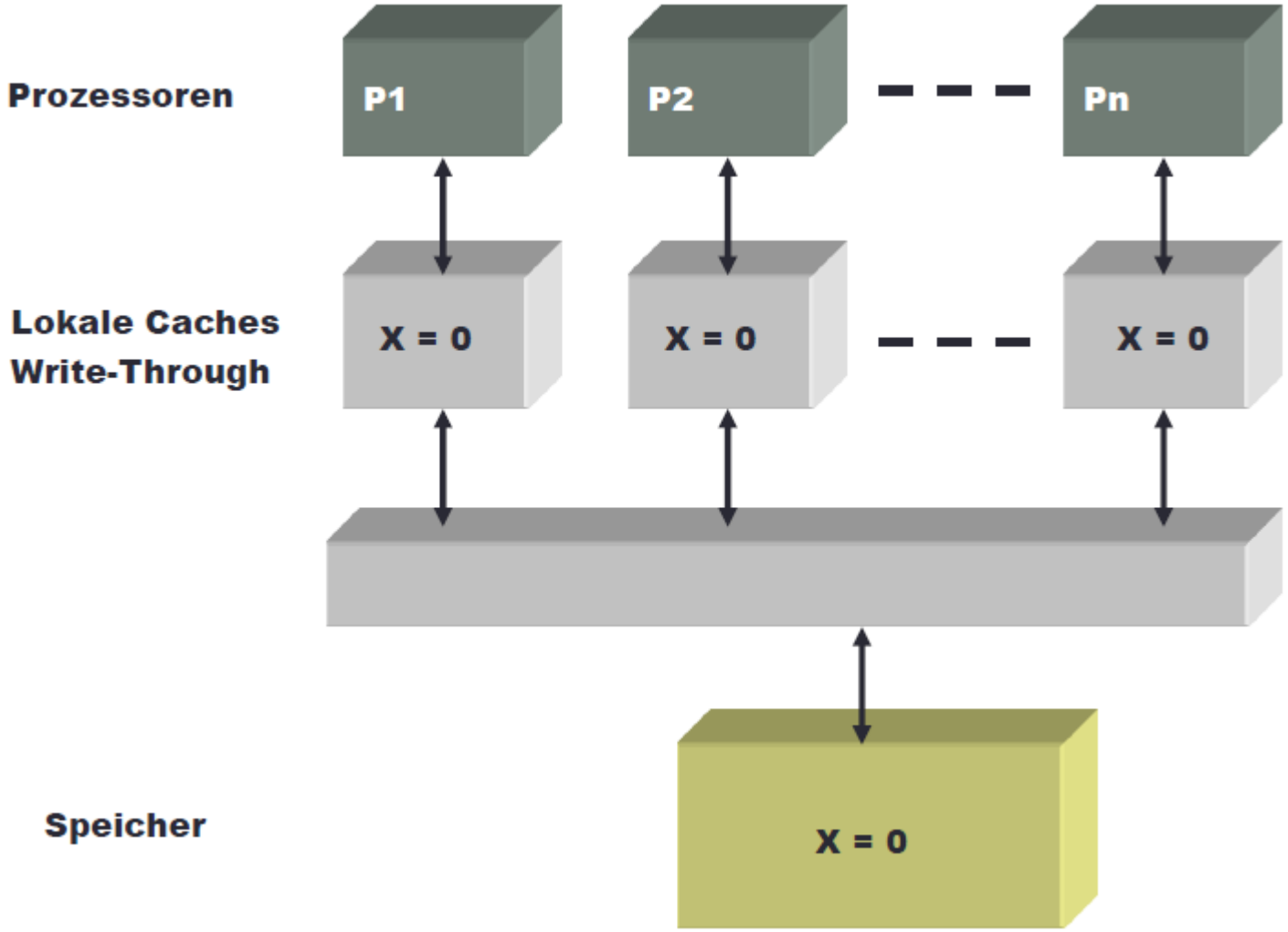
Write-Through Cache – Write Invalidate Protokoll



*P*2 schreibt *X* = 1

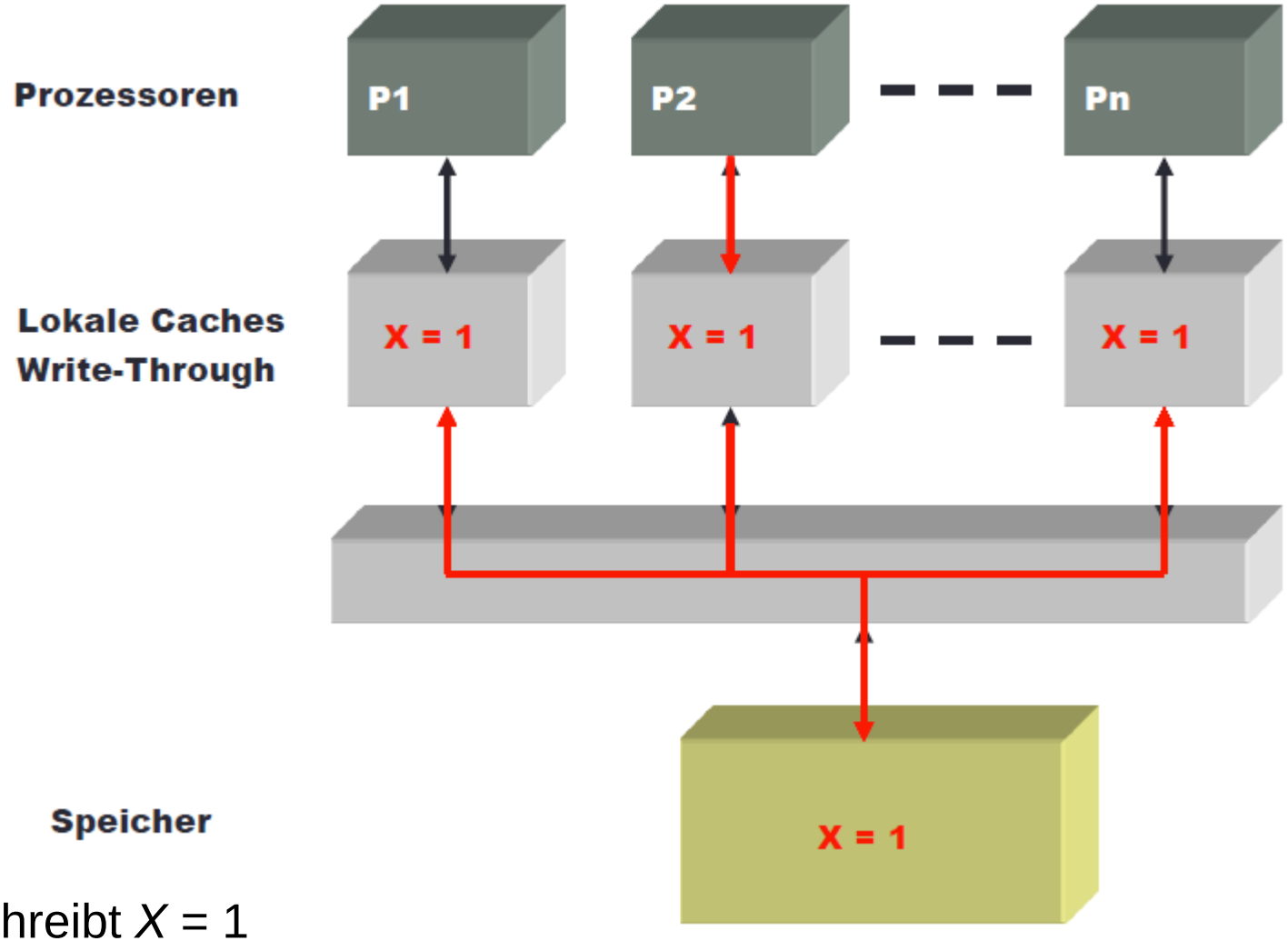
Alle anderen Prozessoren invalidieren den *Cache*-Block

Write-Through Cache – Write Update/Broadcast Protokoll



Kohärente Ausgangssituation

Write-Through Cache – Write Update/Broadcast Protokoll



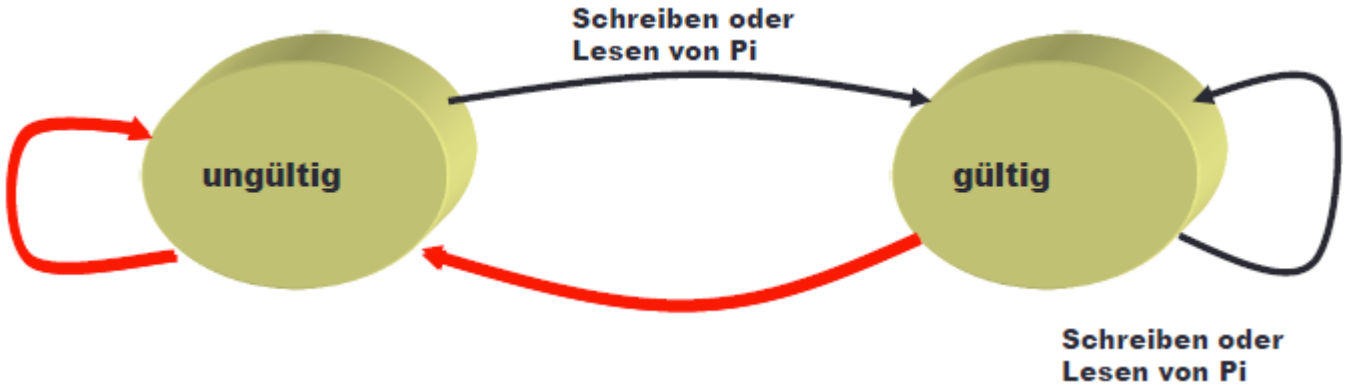
*P*2 schreibt *X* = 1

Alle anderen Prozessoren aktualisieren den *Cache*-Block

Write-Through – Write Invalidate (1)

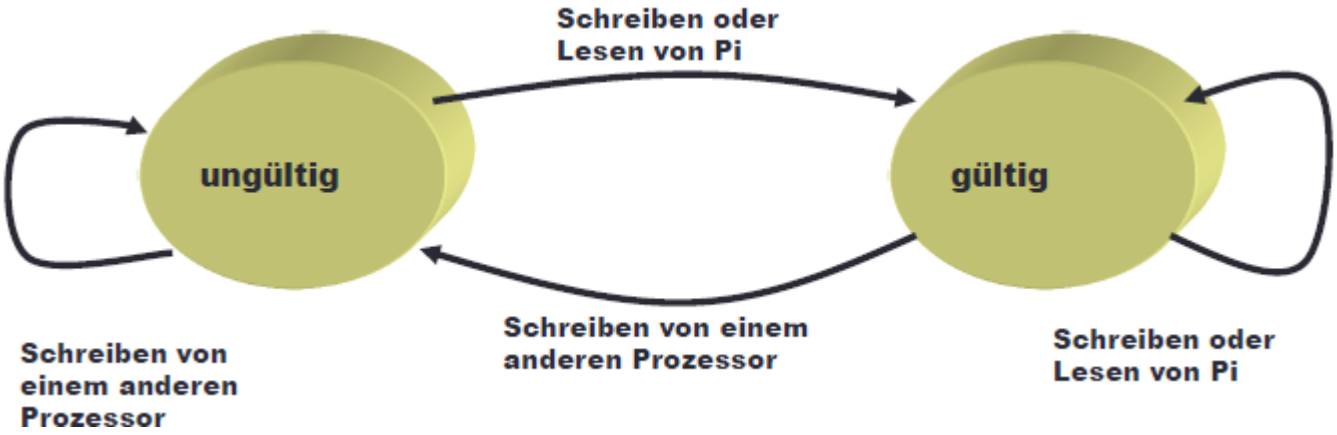
Schreiben oder Lesen von P_i

Write-Through – Write Invalidate (2)



Schreiben eines anderen Prozessors

Write-Through – Write Invalidate (3)



Zustandsdiagramm des *Caches* von P_i

Write-Back

- Problem: *Write Back-Caches* führen zur temporären Inkonsistenz

- Lösung: exklusives Eigentumskonzept durch Zustandsgraph pro *Cache*-Block
  - MSI *(Modified, Shared, Invalid)*
  - MESI *(Modified, Exclusive, Shared, Invalid)*

- Mischung zwischen *Write-Through* und *Write-Back*

MSI (1)

Drei Zustände

– *Modified*

Cache-Block wurde lokal geändert, Kopien im Hauptspeicher oder in *Caches* anderer Prozessoren sind ungültig

– *Shared*

Mehrere *Caches* (mind. 2) enthalten diesen Datenblock; alle *Caches*, die im Zustand *shared* sind, haben eine gültige Kopie

– *Invalid*

Cache-Block ist noch gar nicht geladen bzw. veraltet/ungültig

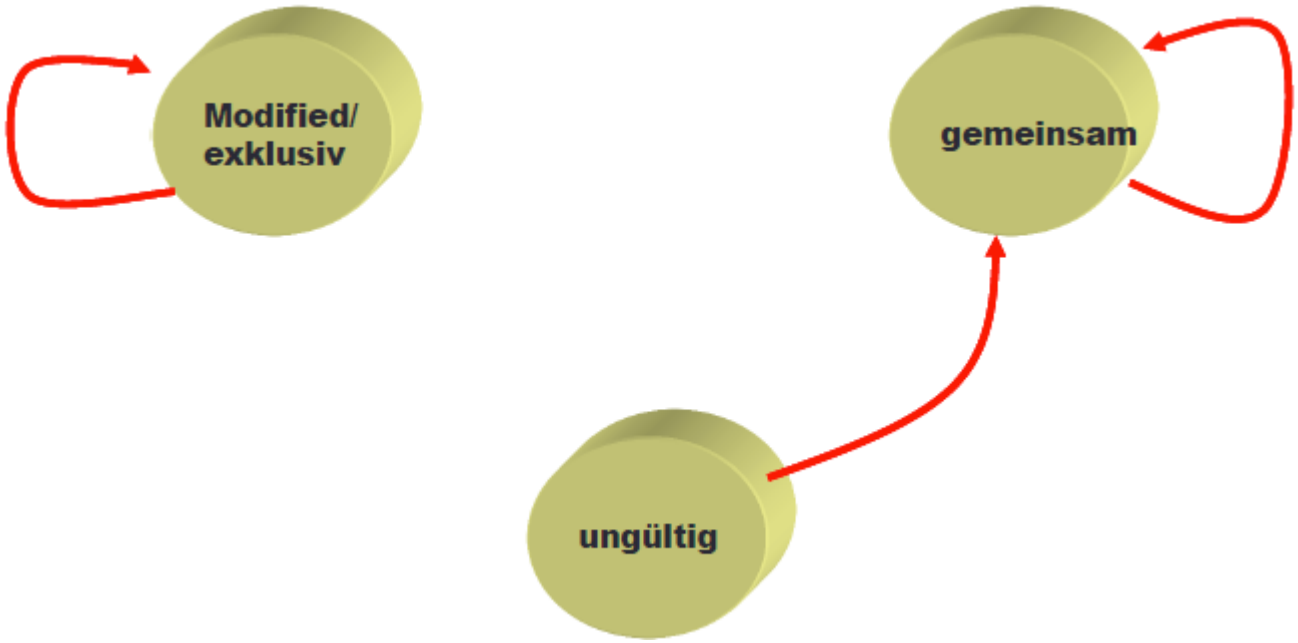
MSI (2)

Prozessoren können auf einen Speicherblock lesend oder schreibend zugreifen. Lese- und Schreiboperationen von Prozessoren lösen Operationen auf dem Bus aus.

Bus-Operationen

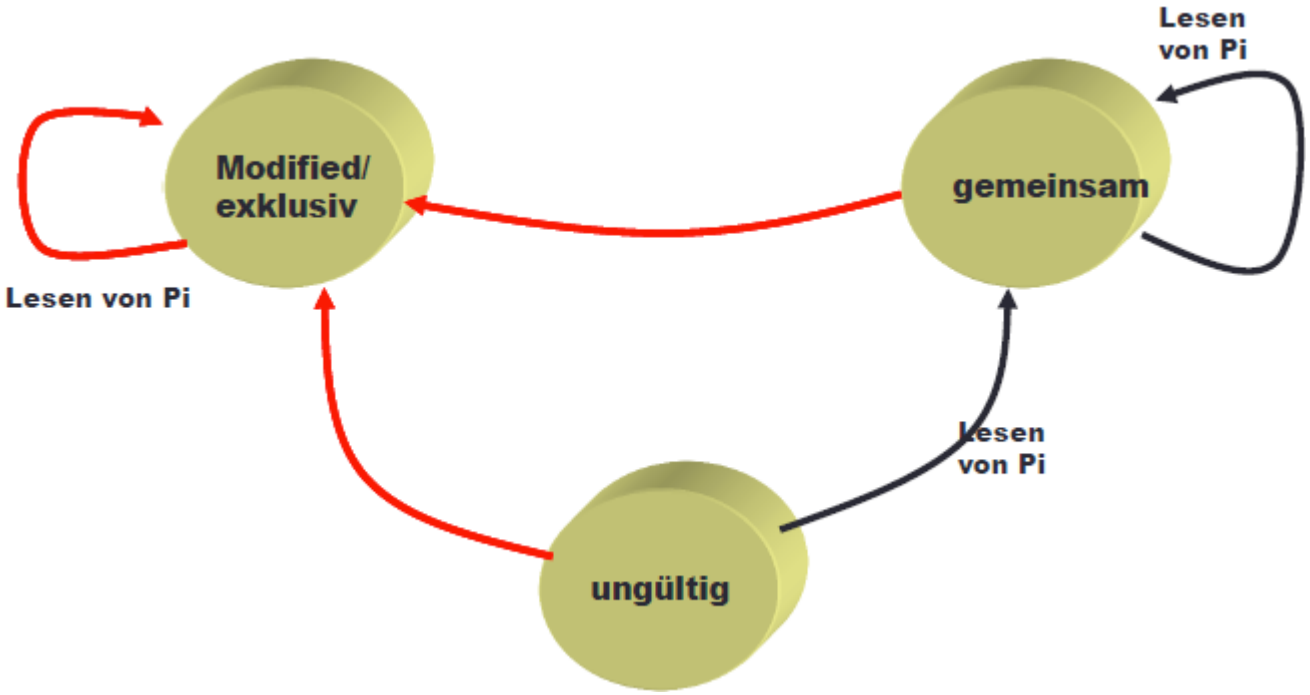
- *Bus Read:* wenn ein Prozessor Wert eines Speicherblocks lesen will
- *Bus Read Exclusive:* wenn ein Prozessor Wert eines Speicherblocks überschreiben will
- *Flush:* wenn ein Prozessor P_i einen Speicherblock alleinig in seinem *Cache* hat, ein anderer Prozessor P_j aber lesend oder schreibend auf diesen Block zugreift. Bei einer *Flush*-Operation legt P_i ebenfalls das Datum des Speicherblocks auf den Bus.

MSI: *Write-Back – Write Invalidate* (1)



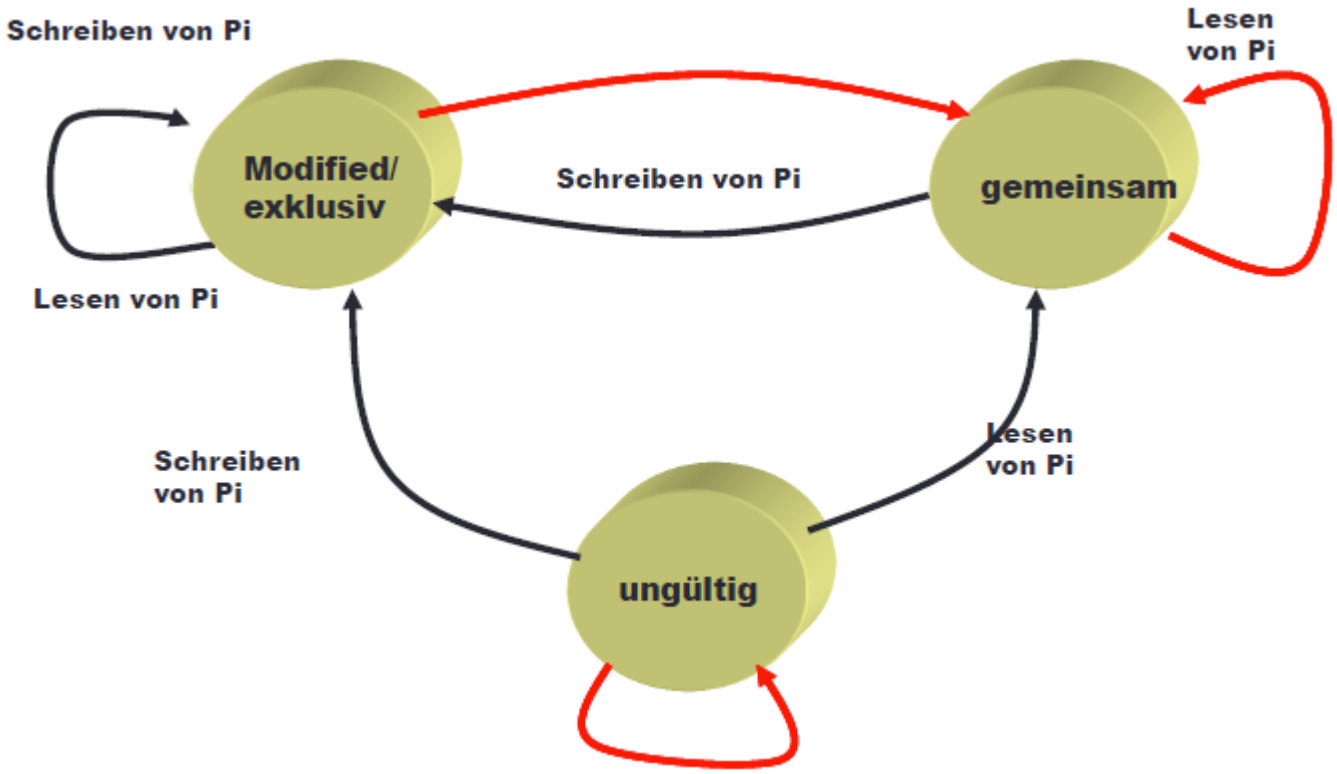
Lesen von P_i

MSI: *Write-Back – Write Invalidate* (2)



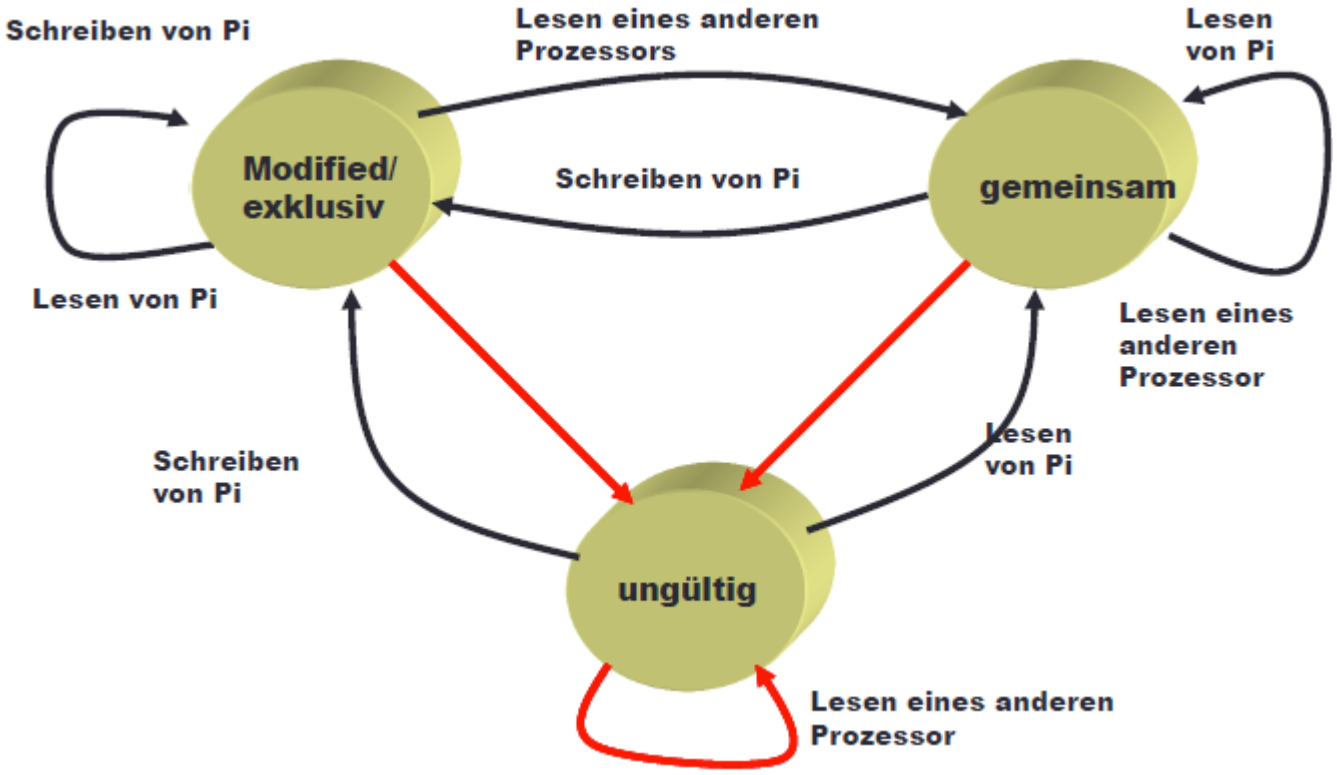
Schreiben von P_i

MSI: *Write-Back – Write Invalidate* (3)



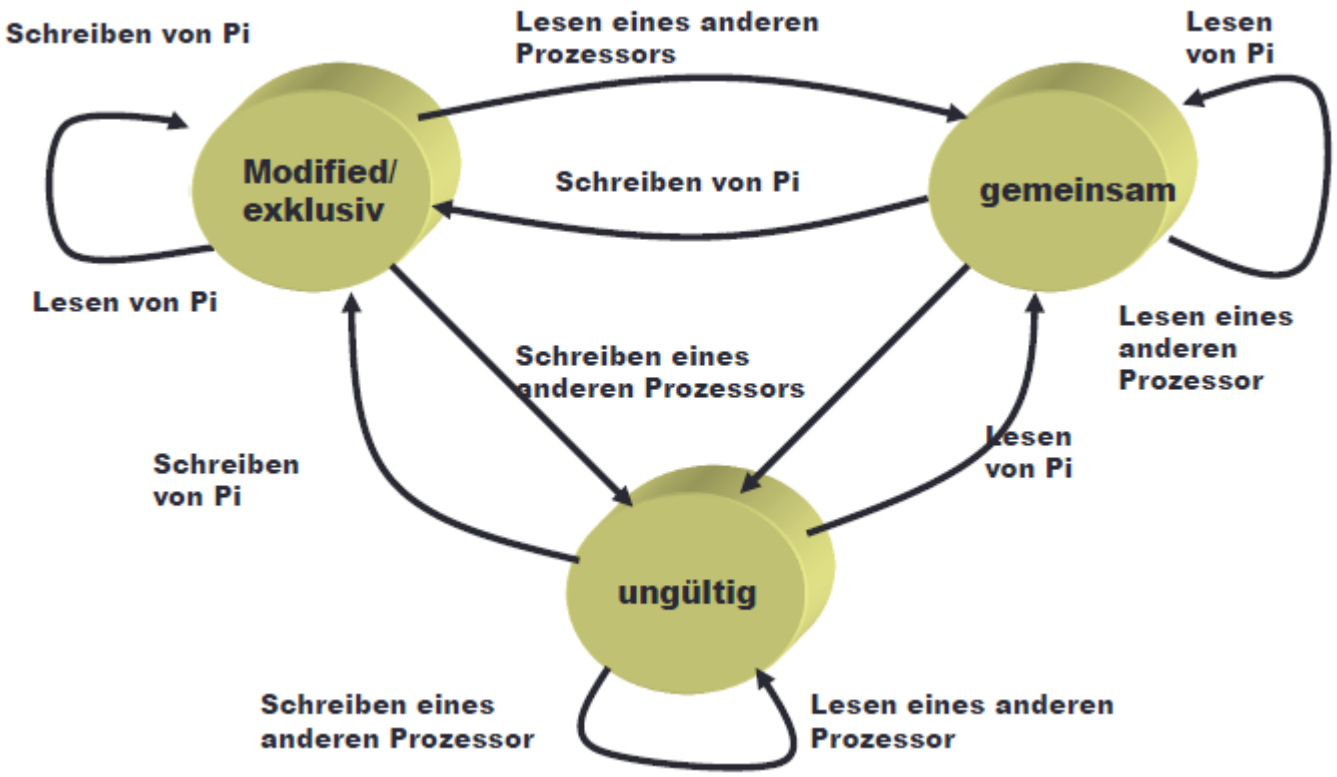
Lesen eines anderen Prozessors

MSI: *Write-Back – Write Invalidate* (4)



Schreiben eines anderen Prozessors

MSI: *Write-Back – Write Invalidate* (5)



Zustandsdiagramm des *Caches* von P_i

MESI (1)

Vier Zustände

– *(exclusive) Modified*
  Cache-Block wurde lokal geändert, die Kopie im Hauptspeicher ist ungültig. Will ein anderer Prozessor dieses Datum im Hauptspeicher lesen, so muss der *Cache*-Block erst in den Hauptspeicher zurückgeschrieben werden.
– *Exclusive (unmodified)*
  Dieser *Cache* ist der einzige, der den *Cache*-Block enthält, Wert im Hauptspeicher ist gültig. Liest ein anderer Prozessor dieses Datum im Hauptspeicher, so muss die Zeile als *shared* markiert werden. Wird das Datum im Hauptspeicher verändert, ist der *Cache*-Block auf *invalid* zu setzen.

MESI (2)

Vier Zustände

– *Shared (unmodified)*

  Mehrere *Caches* (mind. 2) enthalten dieses Datum. Da alle bisher nur gelesen haben, ist das Datum im Hauptspeicher gültig. Schreibzugriffe auf einen *shared Cache*-Block müssen immer zu einer Bus-Operation führen, damit der Hauptspeicher aktualisiert und die *Cache*-Blocks der anderen *Caches* auf *invalid* gesetzt werden können.

– *Invalid*

  Cache-Block ist noch gar nicht geladen bzw. veraltet/ungültig

Bus-Operationen

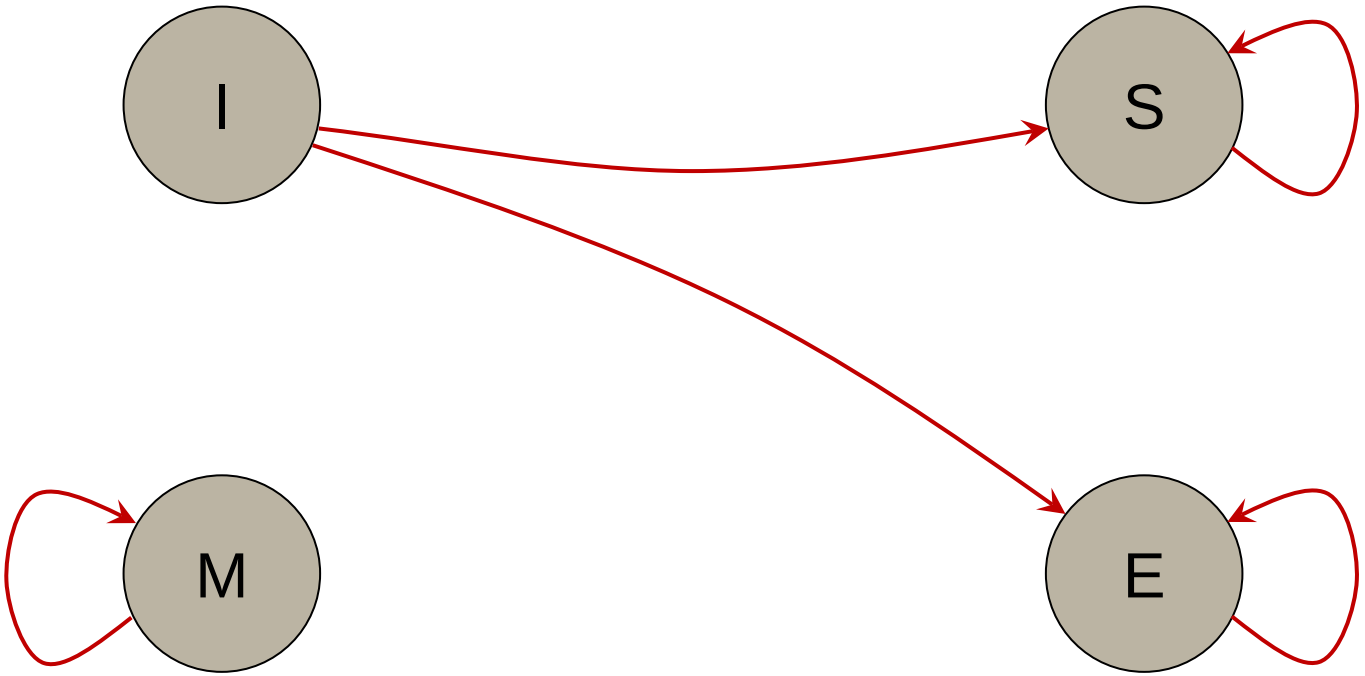
☞ *Siehe MSI*

MESI (3)

Steuersignale

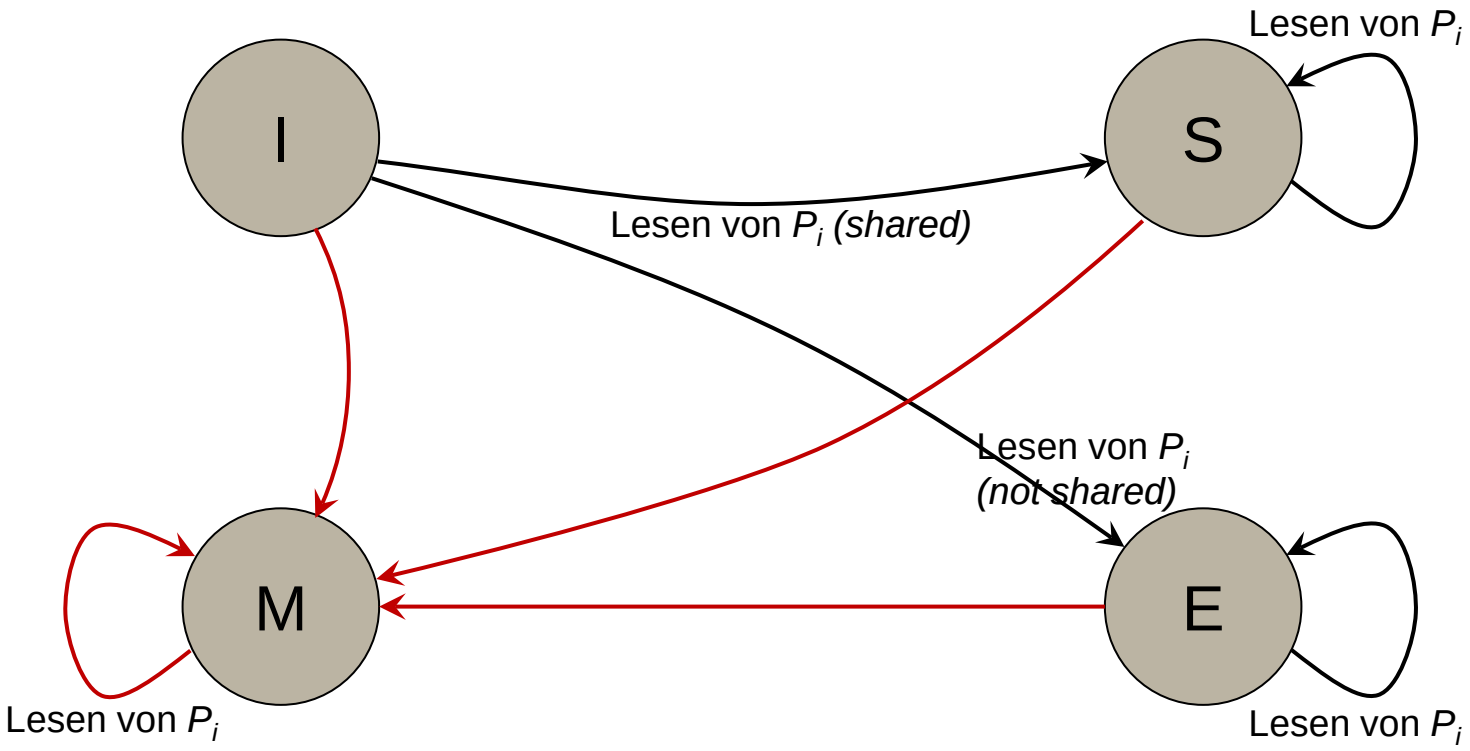
– *Invalidate*-Signal
  Invalidieren des Blocks in den *Caches* anderer Prozessoren
– *Shared*-Signal
  Signalisierung, ob ein zu ladendes Datum bereits als Kopie im *Cache* vorhanden ist
– *Retry*-Signal
  Aufforderung von Prozessor P_i an Prozessor P_j, das Laden eines Datums vom Hauptspeicher abzubrechen, da der Hauptspeicher noch ein altes, ungültiges Datum besitzt und vorher aktualisiert werden muss. Das Laden durch P_j kann danach wiederholt werden.

MESI-Zustandsdiagramm (1)



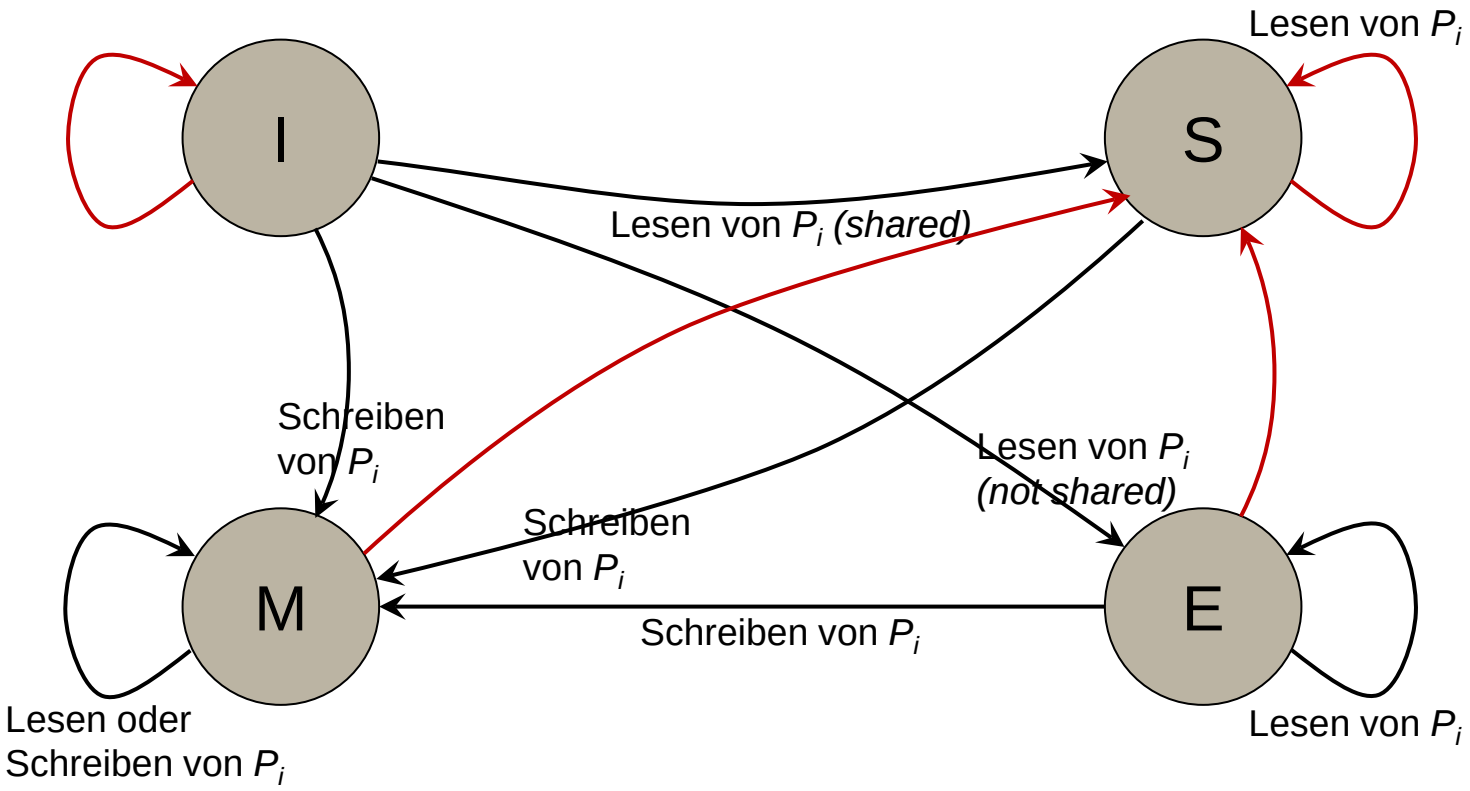
Lesen von P_i

MESI-Zustandsdiagramm (2)



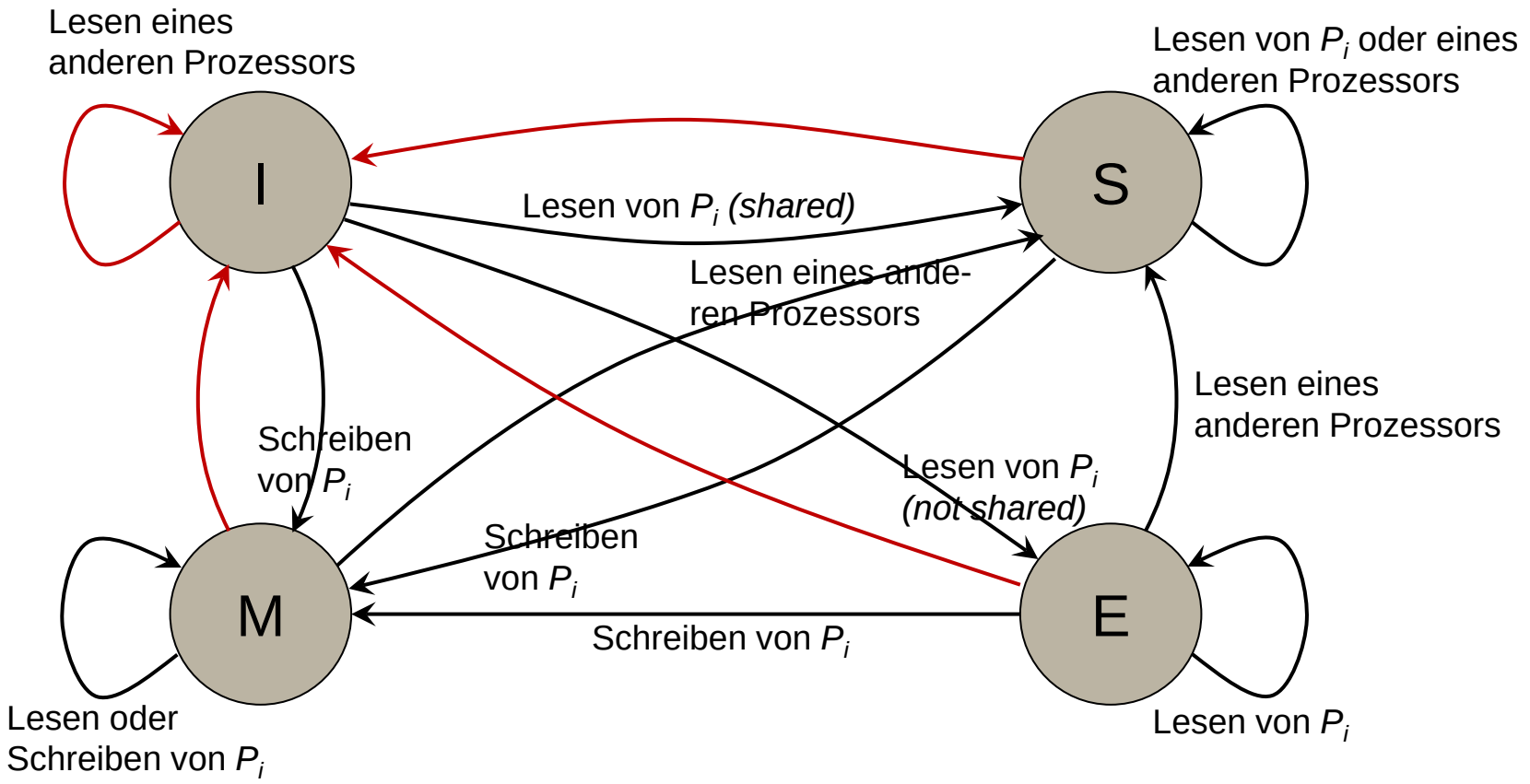
Schreiben von P_i

MESI-Zustandsdiagramm (3)



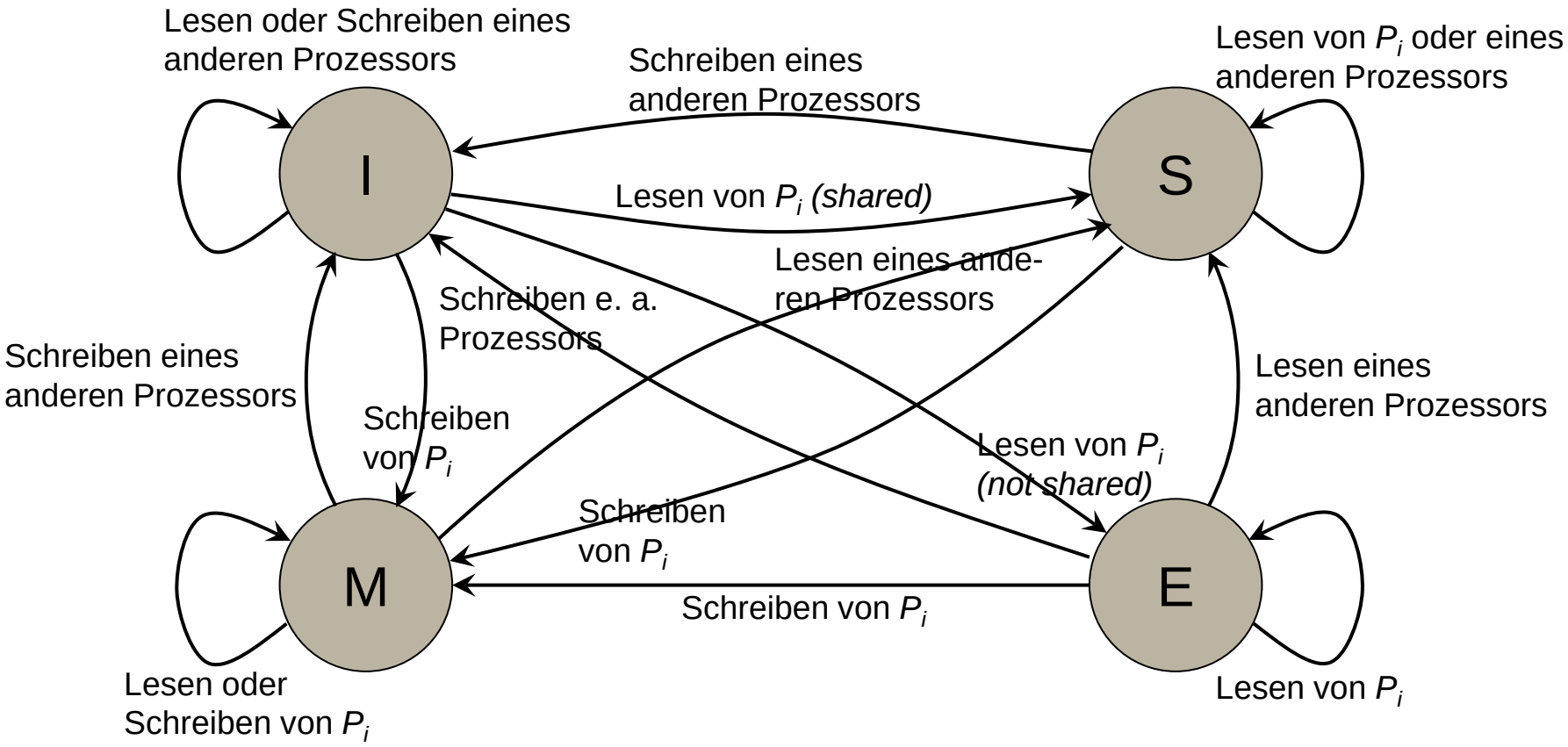
Lesen eines anderen Prozessors

MESI-Zustandsdiagramm (4)



Schreiben eines anderen Prozessors

Letztliches MESI-Zustandsdiagramm



Bewertung von *Snooping*-Protokollen

- Leichte Implementierbarkeit bei Bus-basierten *Shared Memory* Systemen
- *Snooping* skaliert bei Bussen jedoch nicht
  - Bei vielen beteiligten Prozessoren sinkt die effektive Bandbreite des Busses, da überproportional viele Invalidierungsnachrichten per *Broadcast* über den Bus gehen

- Punkt-zu-Punkt Netzwerke sind skalierbar, jedoch ist die Implementierung von *Broadcasts* hier aufwändig
  - Für *Snooping*-Protokolle daher oft ungeeignet

☞ Ist eine *Broadcast*-Invalidierungsnachricht an alle Prozessoren nötig?

Directory-Protokolle (1)

Beobachtung

– Nur wenige Prozessoren teilen sich die gleichen Daten in vielen Anwendungen
– Kenntnis nur dieser Prozessoren ist nötig

***Directory*-Protokolle**

– *Directory*-Protokolle nutzen Lokalitätsinformationen, um die Anzahl an Invalidierungsnachrichten zu minimieren
– Nachrichten gehen nur an Prozessoren, die eine Kopie des *Cache*-Blocks besitzen
– Directory-Protokolle skalieren daher auch für Netze ohne *Broadcast*-Fähigkeit

Directory-Protokolle (2)

Ansatz

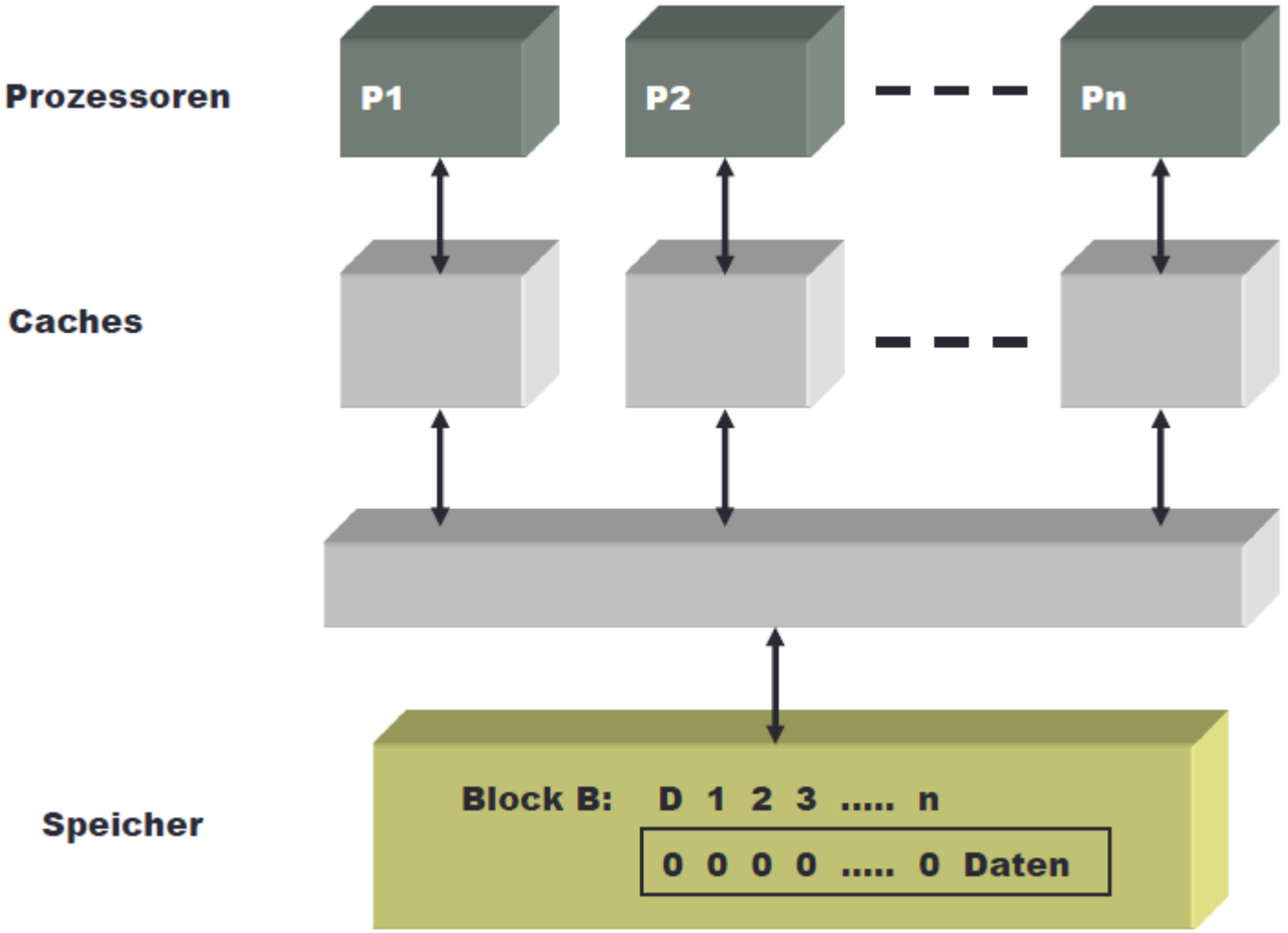
– *Presence Flag Vector*

Im Hauptspeicher abgelegter Bit-Vektor für jeden einzelnen Speicherblock:

1 Bit pro Prozessor/*Cache* + Statusbits (*dirty*, *modified*)

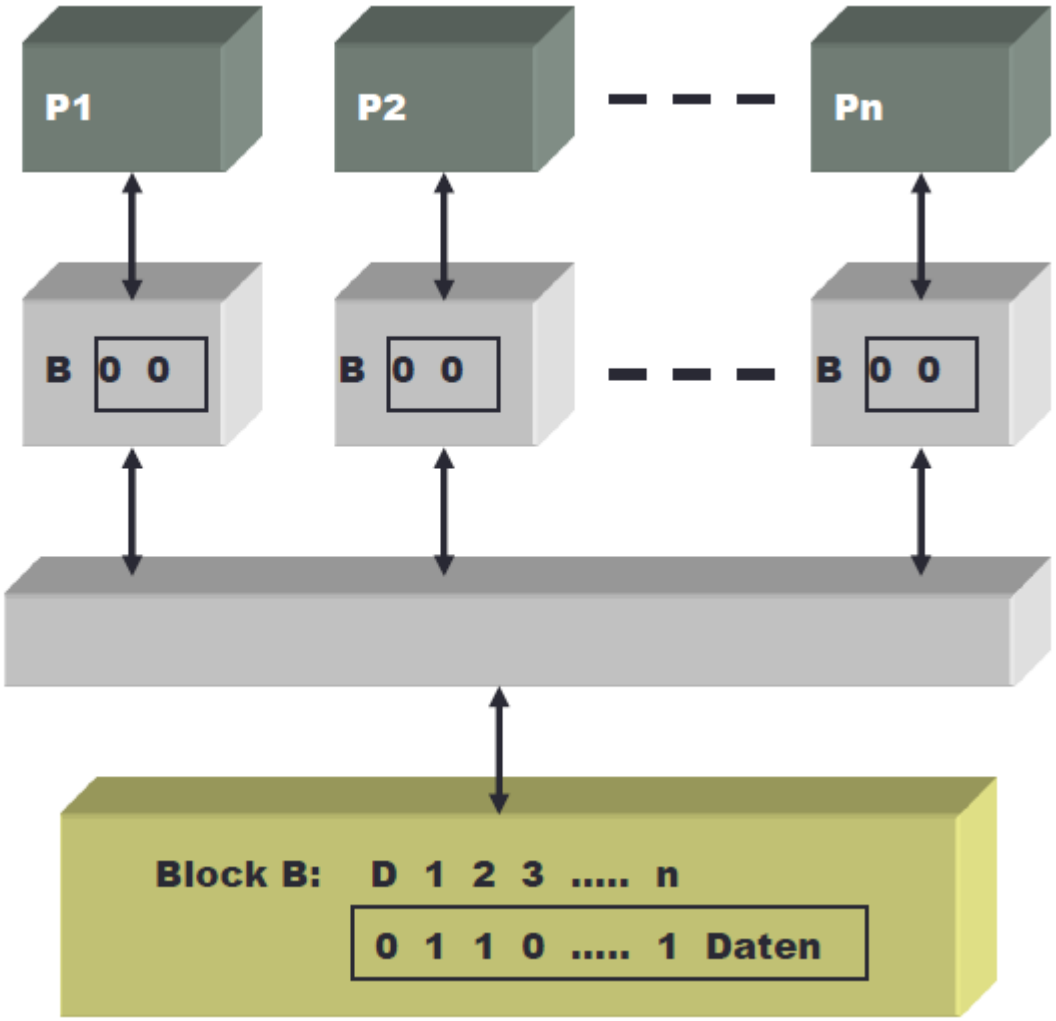
Beispiel *Presence Flag Vector* (1)

Block *B* befindet sich in keinem *Cache*



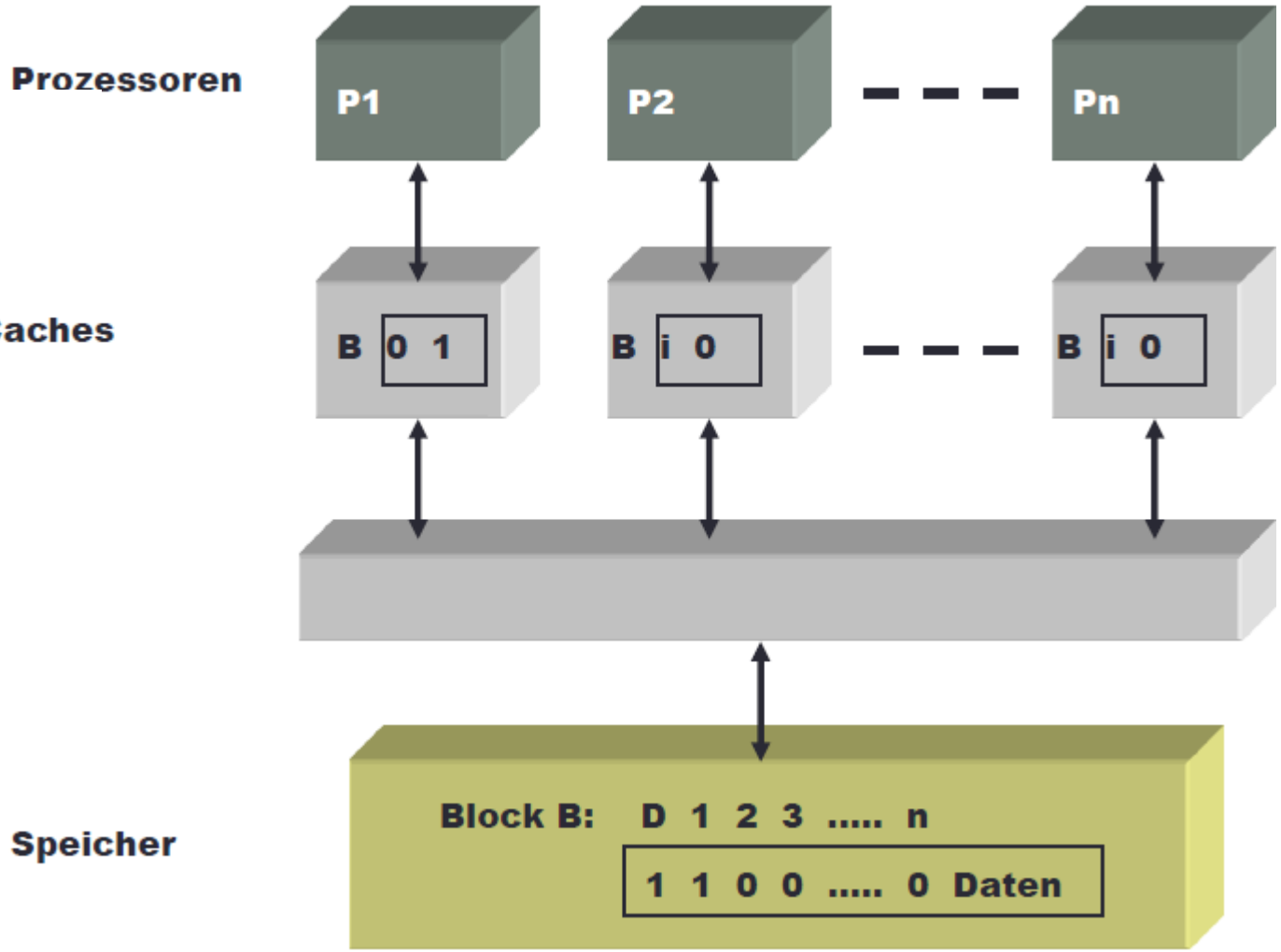
Beispiel *Presence Flag Vector* (2)

Block *B* befindet sich in den *Caches* von *P*1, *P*2 und *Pn*



Beispiel Presence Flag Vector (3)

Block *B* befindet sich in den *Caches* von *P*1 (überschrieben), *P*2 und *Pn* (ungültig)



Bewertung von *Directory*-Protokollen

Problem

– Wachstum des Speicherbedarfs linear mit Anzahl der Prozessoren

Beispiel

– Speicherblöcke von 64 Bytes Größe
  – 64 Prozessoren = Overhead 12,69%
  – 256 Prozessoren = Overhead 50,2%
  – 1024 Prozessoren = Overhead 200,16%

Roter Faden

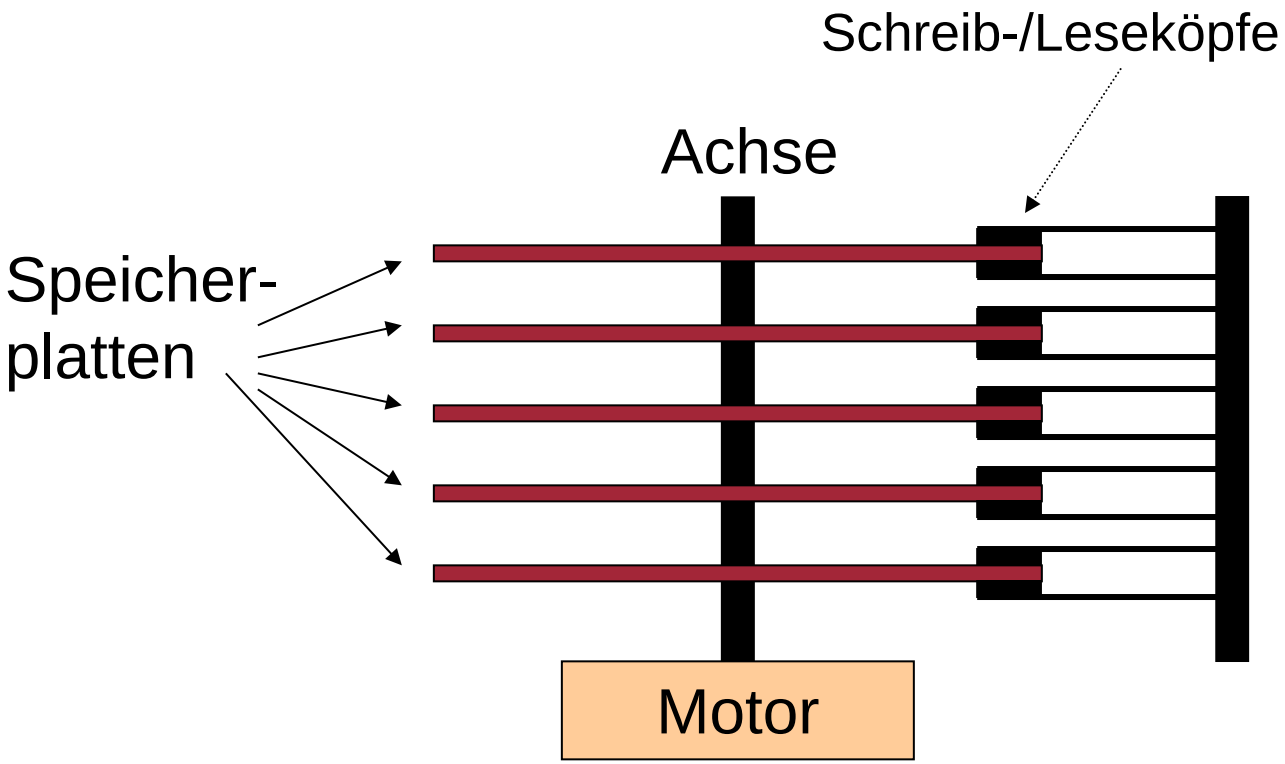
7. Speicher-Hardware

- Einleitung: Speicherhierarchien
- SRAM
- DRAM
- *Translation Look-Aside Buffer* (TLBs)
- *Caches*
  - Organisation von *Caches*
  - Schreibverfahren: *Write-Through*, *Copy-Back*
  - Virtuelle vs. Reale *Caches*
  - Austauschverfahren: *Random*, FIFO, *Clock*, NRU, LRU
  - *Cache*-Kohärenz
    - Gründe für Inkohärenzen
    - *Snooping*- & *Directory*-Protokolle
- Massenspeicher

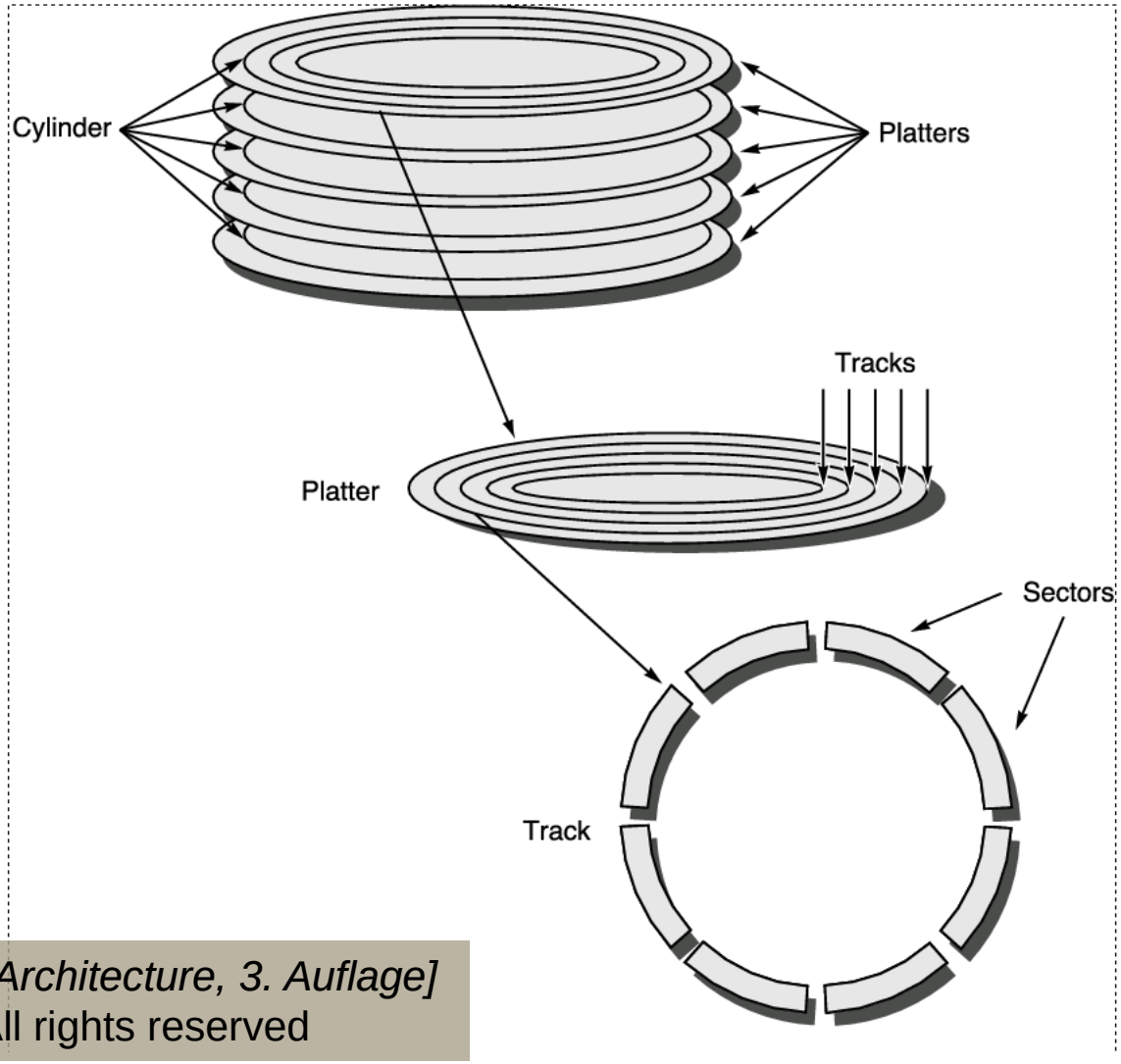
Massenspeicher

- Plattenspeicher
  - *Disc-Arrays*
- CD-ROM
- DVD
- Blu-ray
- Bandlaufwerke

Schematischer Aufbau eines Plattenlaufwerks



Einteilung der Platten in Sektoren, Spuren und Zylinder

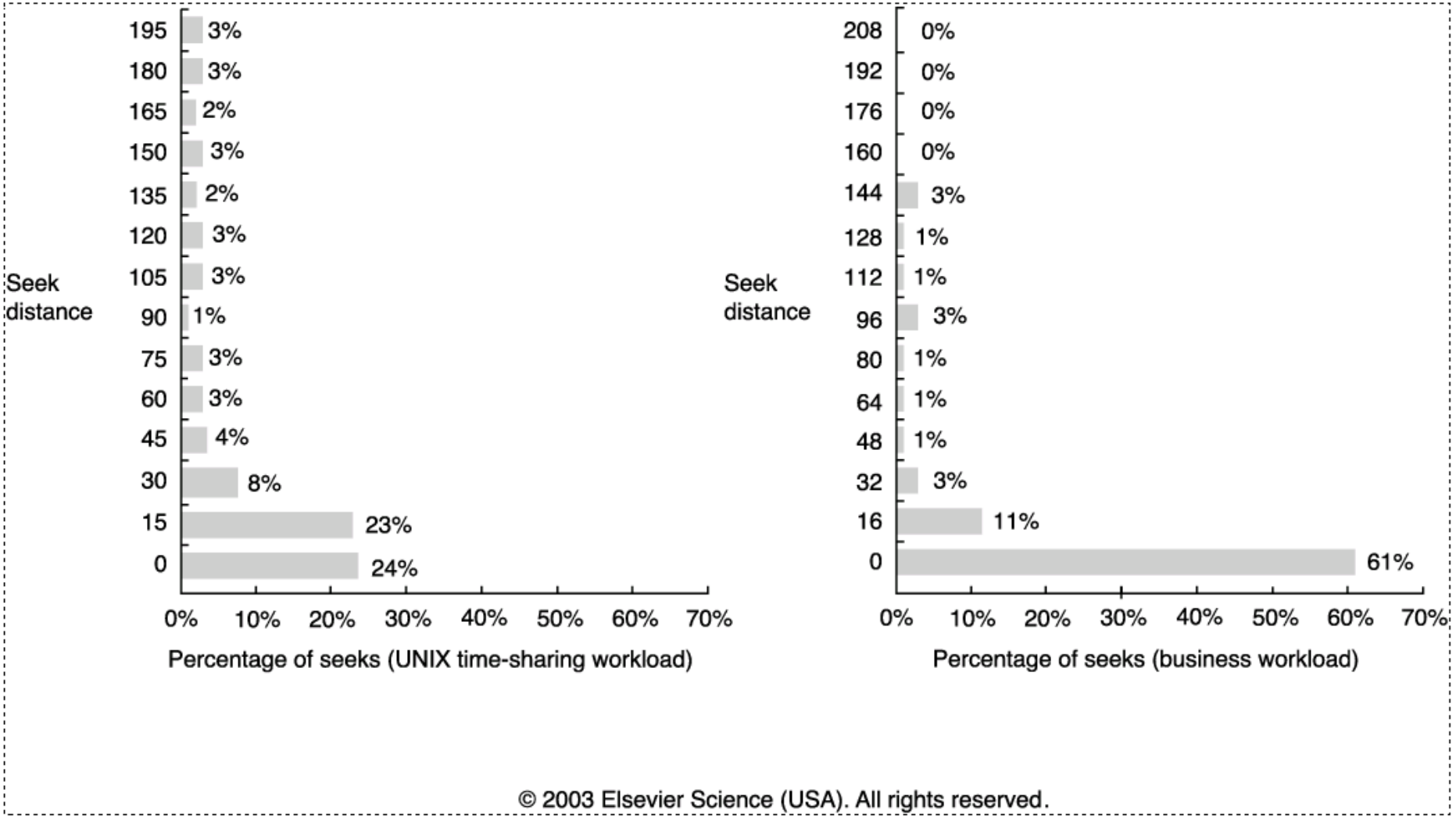


*[**Hennessy/Patterson,** Computer Architecture, 3. Auflage]*
© Elsevier Science (USA), 2003. All rights reserved

Daten einiger Plattenlaufwerke (2011)

	Seagate Barracuda XT.	Seagate Momentus Thin	Toshiba MK1214GAH
Durchmesser ["]	3,5	2,5	1,8
Kapazität [GB]	2000	320	320
Umdrehungen [RPM]	7200	7200	5400
Mittl. Latenz [ms]	4,17	4,17	5,56
Average seek [ms]	8,5 read/9,5 write	11	15
Übertragungsrate	Max. 600 MB/s, 138 MB/s dauerhaft	300 MB/s	300 MB/s
Leistung [W] *passiv/aktiv*	6,39/9,23	0,66/1,6	0,40/1,3
Puffer [MB]	64	16	16
Gb/Quadratzoll	347	425	?
Schock-Toleranz [Betrieb/außer Betrieb]	86 G/300 G	350 G/1000 G	500 G/1500 G
Interface	SATA 6 Gb/s	SATA, 3 Gb/s	SATA, 3 Gb/s

Suchzeit: Verteilung der Spurdifferenzen



© 2003 Elsevier Science (USA). All rights reserved.

Schnittstellen

- IDE/ATA *(Integrated Drive Electronics)*
- SATA *(Serial ATA)*
- SCSI *(Small Computer System Interface)*
- SAS *(Serial attached SCSI)*
  SCSI-Protokoll über serielle, SATA-kompatible Leitung
- *Fibre Channel*
  Hochleitungsanschluss für den Server-Bereich
- USB/FireWire

Redundant Arrays of Independent Discs (RAID)

- Verbesserung der I/O-Geschwindigkeit durch paralleles Lesen/Schreiben auf mehrere Platten
- Einsatz von Redundanz, damit dadurch nicht die Zuverlässigkeit leidet

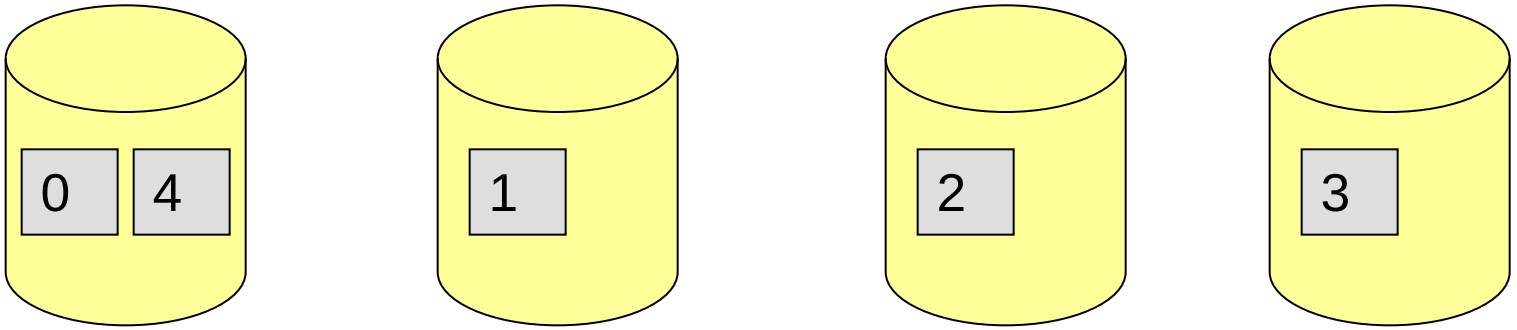
☞ *Disc arrays, drive arrays*

RAID 0 *(Striping)*

Dateien werden unterteilt in so genannte ***Stripes***.

Striping-Parameter = Länge eines *Stripes*.

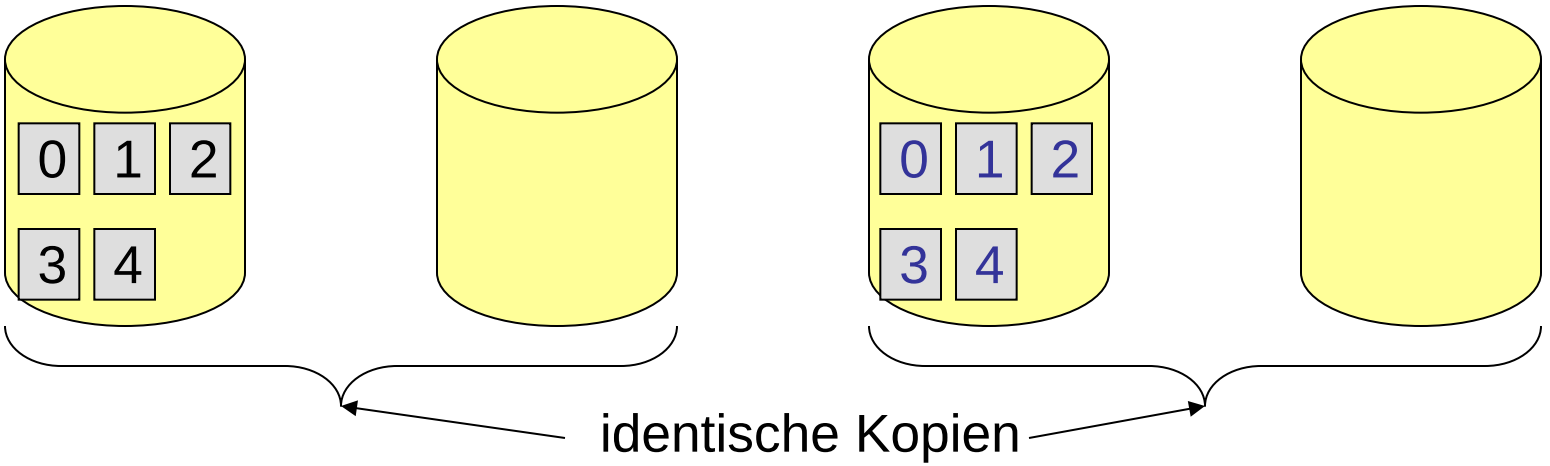
Diese *Stripes* werden auf den Platten verteilt.



– Keine Redundanz
– Nur Verbesserung der Übertragungsrate

RAID 1 *(Mirroring)*

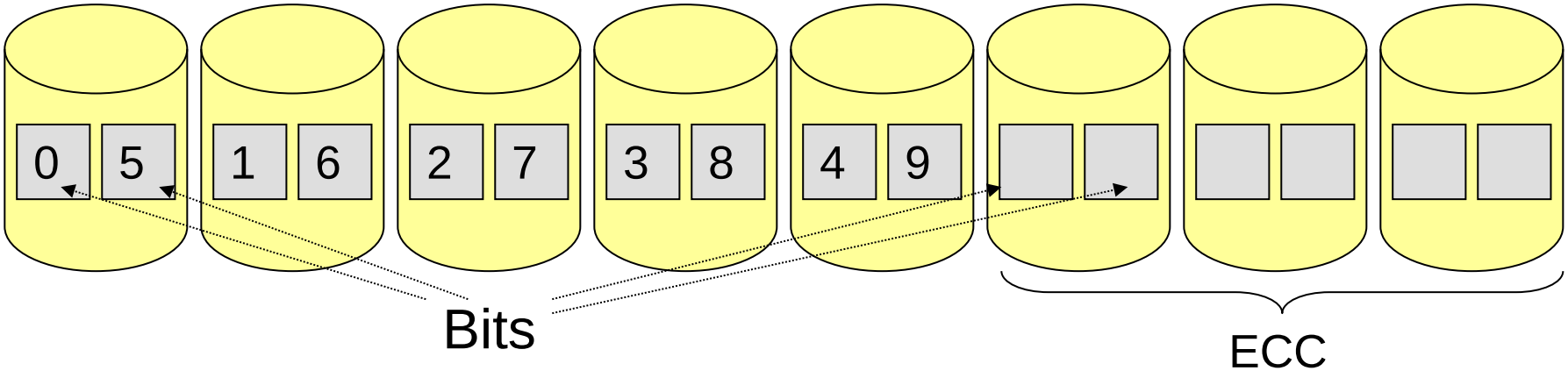
Die selben Informationen werden auf zwei Platten geschrieben (gespiegelte Platten, *mirrored discs*).



- Erhöhte Fehlersicherheit
- Stark erhöhter Aufwand (x 2)
- Es werden eigentlich nur 2 Plattenlaufwerke genutzt

RAID 2 *(ECC)*

Zusätzlich werden Prüfbits auf spezielle Platten geschrieben, Verwendung von fehlerkorrigierenden Codes *(error-correcting code, ECC)*.
Die einzelnen Daten- und Prüfbits werden auf mehrere Platten verteilt.



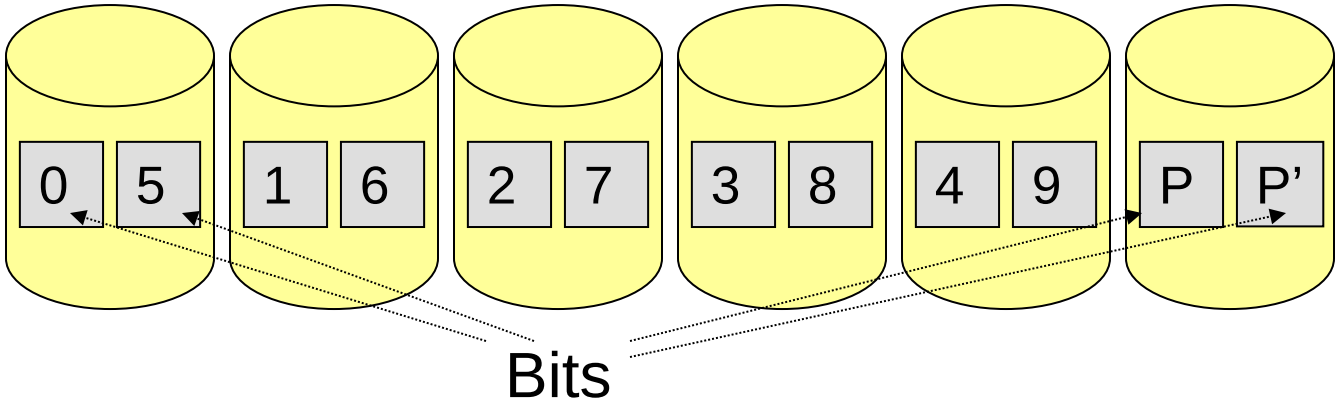
– Geringere Redundanz als RAID 1, aber wegen Prüfbit-Erzeugung und Last auf Prüfbitplatten langsamer
– Erfordert spezielle Platten, kommerziell nicht verfügbar

RAID 3 *(Dedicated parity)*

Es wird nur ein einzelnes Paritäts-*Stripe* auf einer *Parity*-Platte abgelegt.

Häufig wird nur der Fall *Stripe*-Parameter = 1 Bit betrachtet.

Die *Stripes* eines Datenblocks werden auf Platten verteilt.

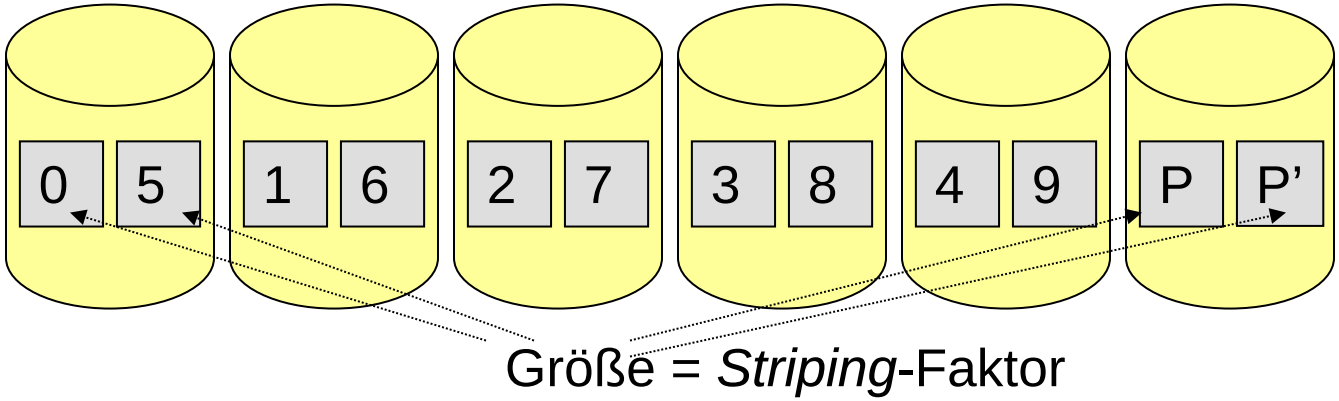


– Selbst für kleinste zu schreibende/lesende Blöcke sind alle Platten aktiv

☞ *Keine gute Performance*

RAID 4

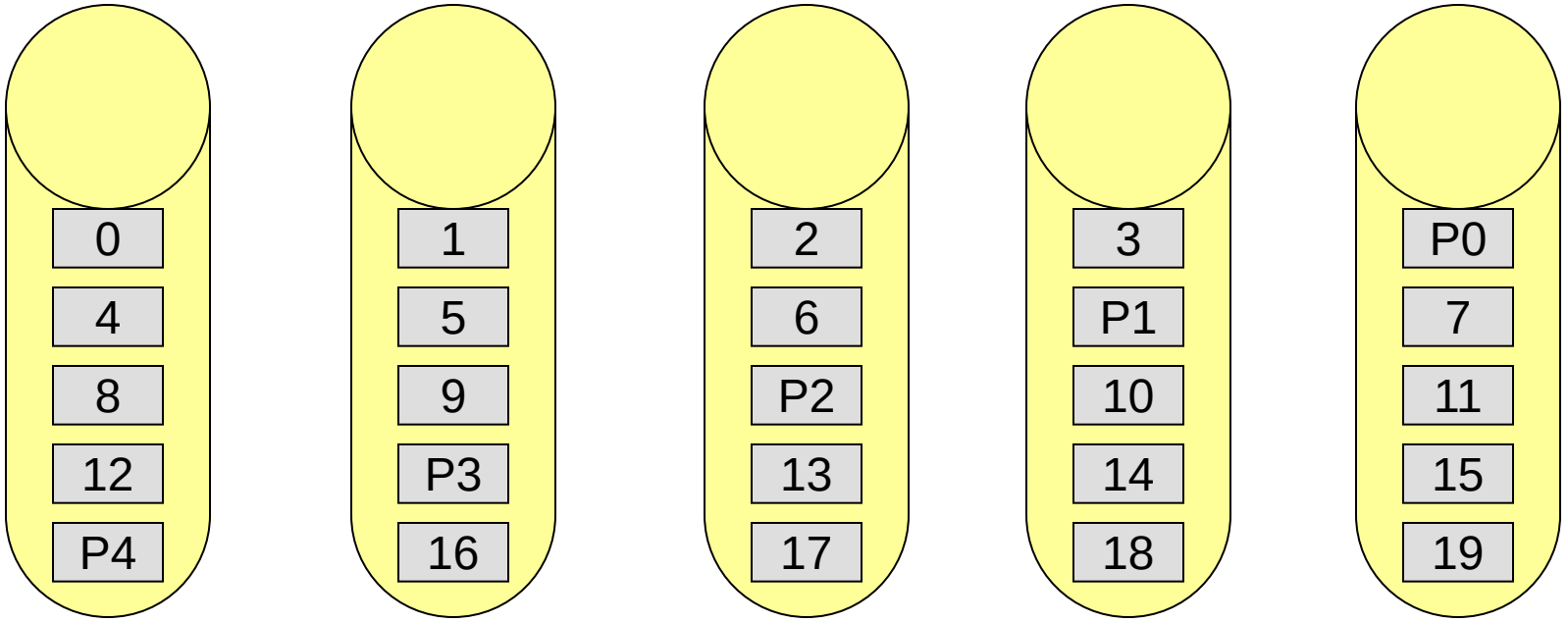
Wie RAID 3, jedoch mit einem *Striping*-Faktor von einem Block und mehr.



- Besserer wahlfreier Zugriff als RAID 3
- Paritätsplatte bleibt ein Engpass

RAID 5 *(Distributed parity)*

Paritätsinformation wird über verschiedene Platten verteilt.

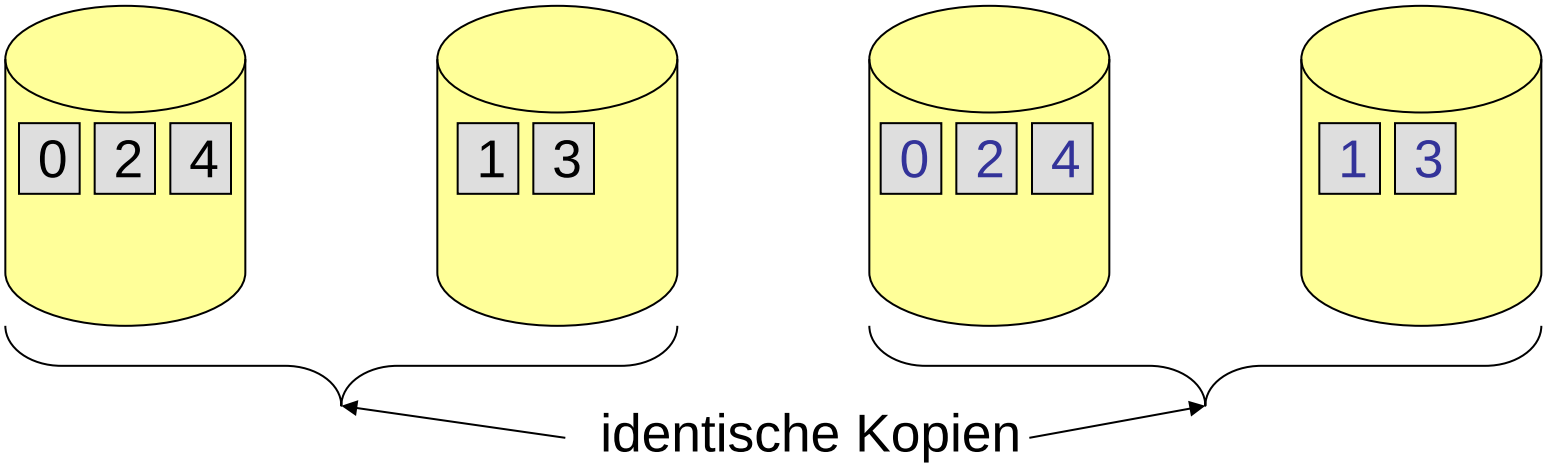


- Paritätsplatte nicht weiter ein Engpass
- Fehlertoleranz: es darf max. 1 komplette Festplatte ausfallen

RAID 0+1, RAID 01 *(Mirrored stripes)*

Gespiegelte Platten (RAID 1) mit verteilten *Stripes* (RAID 0)

☞ *Mirrored stripes*



– *Lese*geschwindigkeit erhöht
– Erhöhte Fehlersicherheit
– Stark erhöhter Aufwand

Übersicht

RAID Variante	Technik	Fehlertoleranz	Daten-Platten	Prüfplatten
0	*non-redundant*	0	8	0
1	*mirrored*	1	8	8
0+1	Komb. v. 0 & 1	1	8	8
2	*memory-style ECC*	1	8	4
3	*bit-interleaved parity*	1	8	1
4	*block-interleaved parity*	1	8	1
5	*dto, distributed parity*	1	8	1
6	*RAID 5 & dop-pelte vert. Parity*	2	8	2

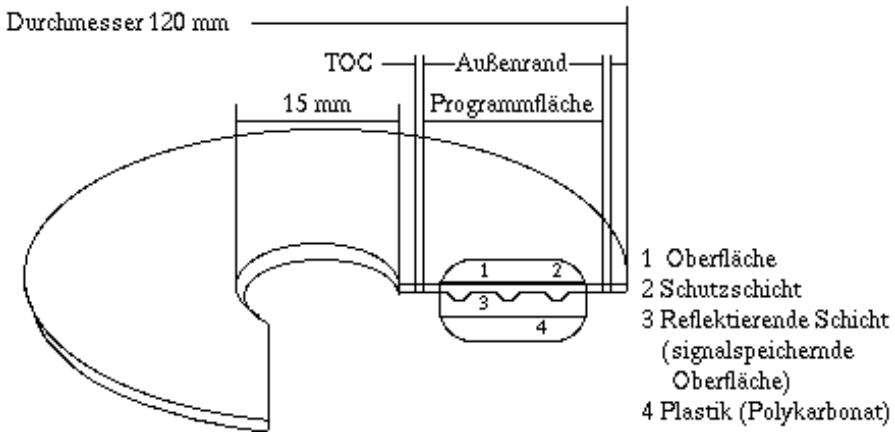
Beispiel

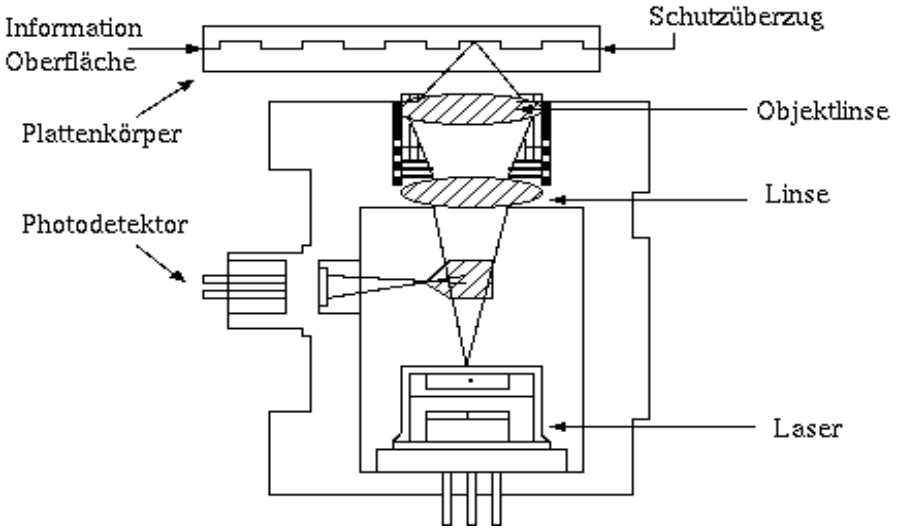
Ein heutiges RAID

- RAID 6
- 16 Hitachi-Festplatten à 300 GB
- Interface HDD ↔ RAID: SAS
- Interface RAID ↔ Server:
  2 SAS-Busse à 6 Gbit/s
- 4 GB interner DDR-*Cache*
- *Hot-pluggable* Festplatten,
  Netzteile und Lüfter

CD-ROM

Datenträger auf der Basis der Technik von Audio-CDs.
Informationen werden als Vertiefungen in einer reflektierenden Oberfläche gespeichert. Abtastung mit 780nm Laser.

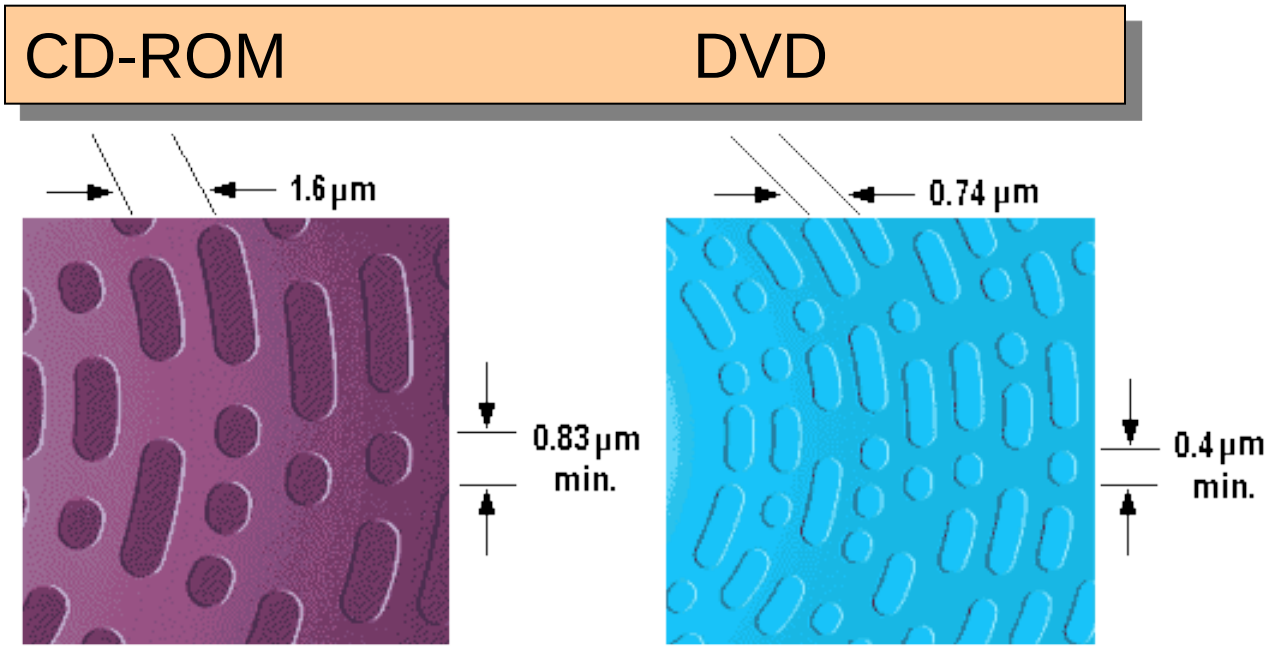




DVDs (1)

Kapazität gegenüber der CD durch zwei Maßnahmen erhöht:

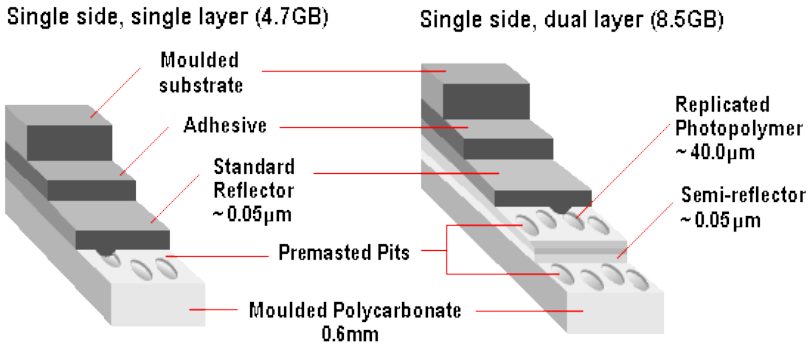
1. Abstände und Größe der Vertiefungen werden reduziert, Abtastung mit 650nm (rotem) Laserstrahl

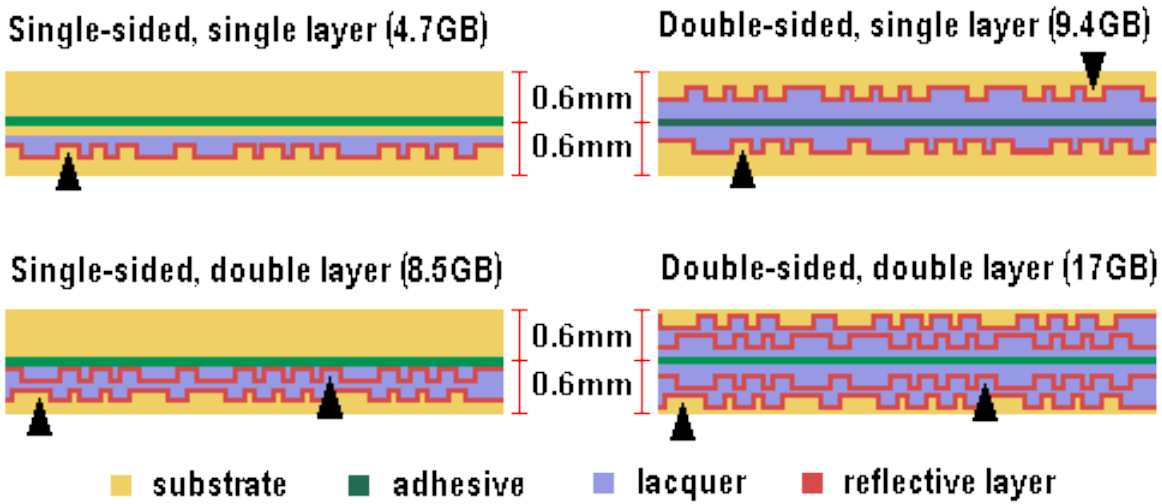


☞ Erhöhung der Kapazität um den Faktor 4.

DVDs (2)

2. Informationen können in zwei Ebenen und auf beiden Seiten gespeichert werden.





Datenrate 1,25 MB/s bei einfacher Geschwindigkeit.
MPEG-2 bei Speicherung von Filmen.

Blu*-ray Disc (BD): Motivation

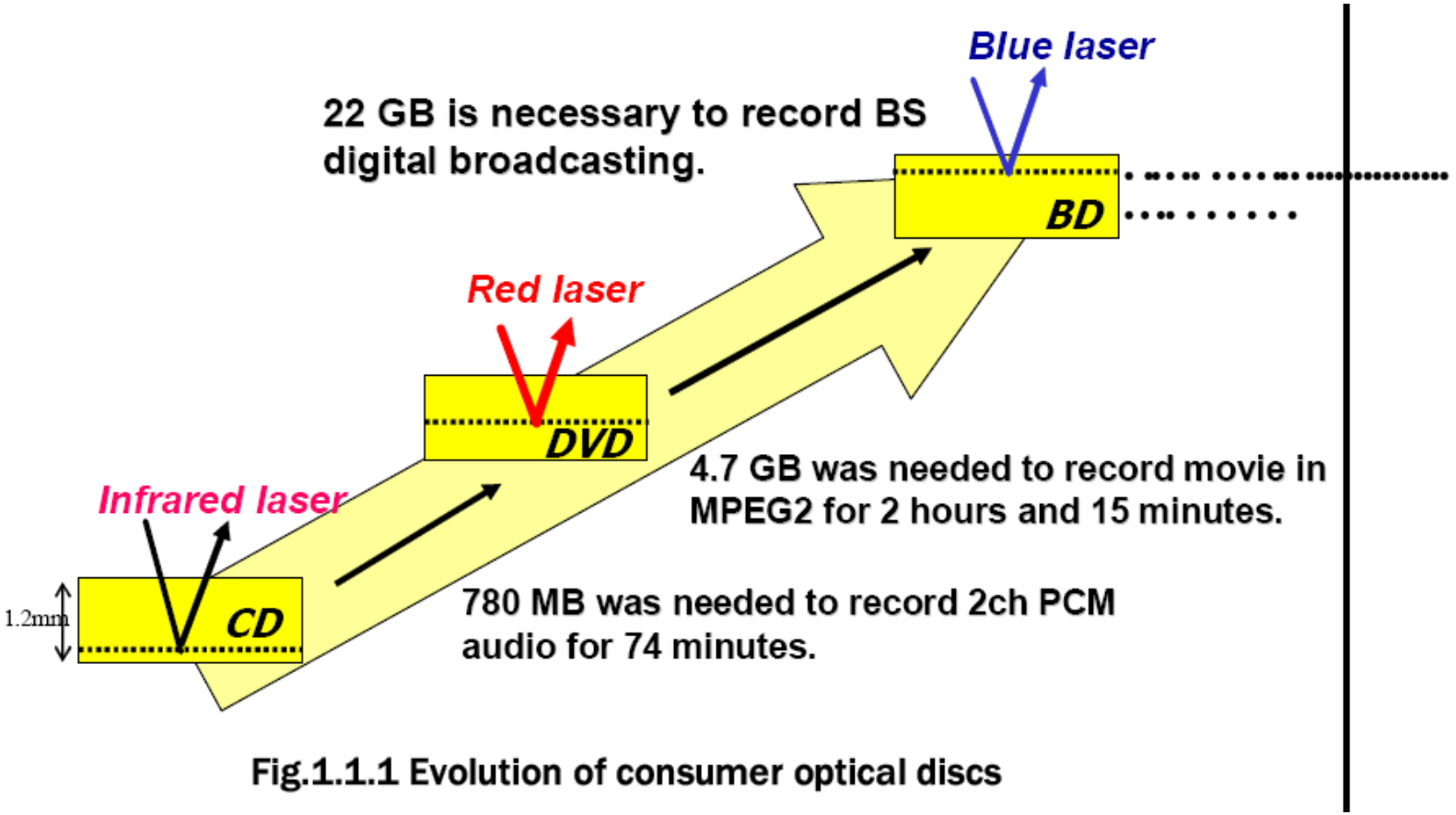


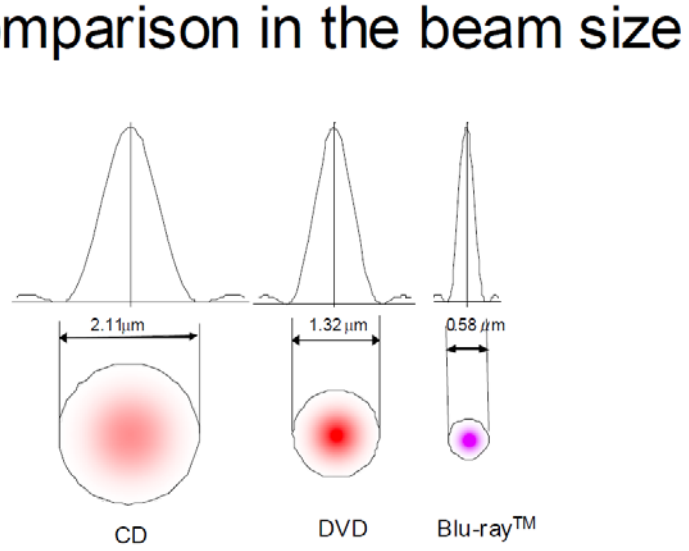
Fig.1.1.1 Evolution of consumer optical discs

* Falsche Schreibweise, um Eintrag als Markenzeichen zu ermöglichen

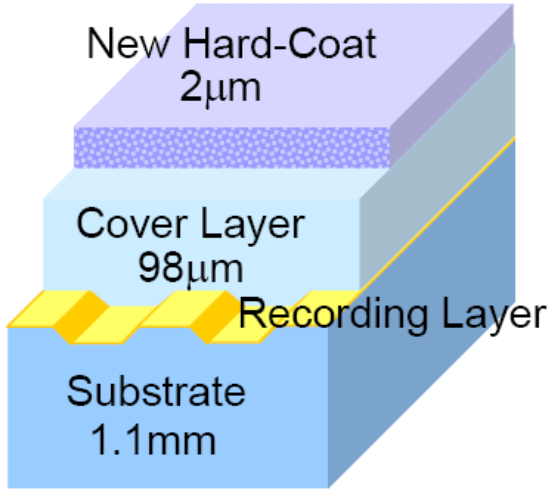
[http://blu-raydisc.com/Assets/Downloadablefile/general_bluraydiscformat-15263.pdf]

Blu-ray: Eigenschaften (1)

Verkürzung der Wellenlänge auf 405 nm (blau).
Durch besser fokussierten Laser Erhöhung der Speicherdichte bei Reduktion des Energiebedarfs möglich.



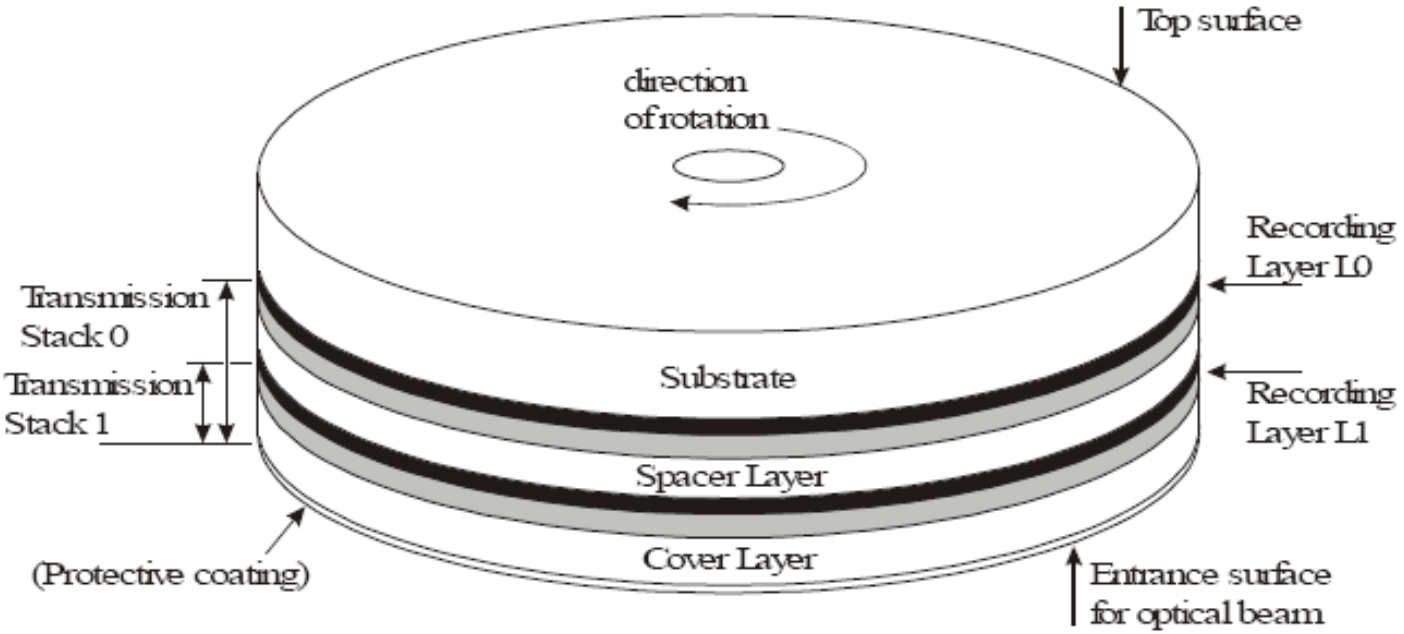
Reduktion der Dicke der transparenten Schicht auf 100 µm, zur besseren Fokussierung des Laserstrahls, bessere Fehlerkorrektur; zusätzliche schmutzresistente Schicht



[http://blu-raydisc.com/Assets/Downloadablefile/general_bluraydiscformat-15263.pdf]

Blu-ray: Eigenschaften (2)

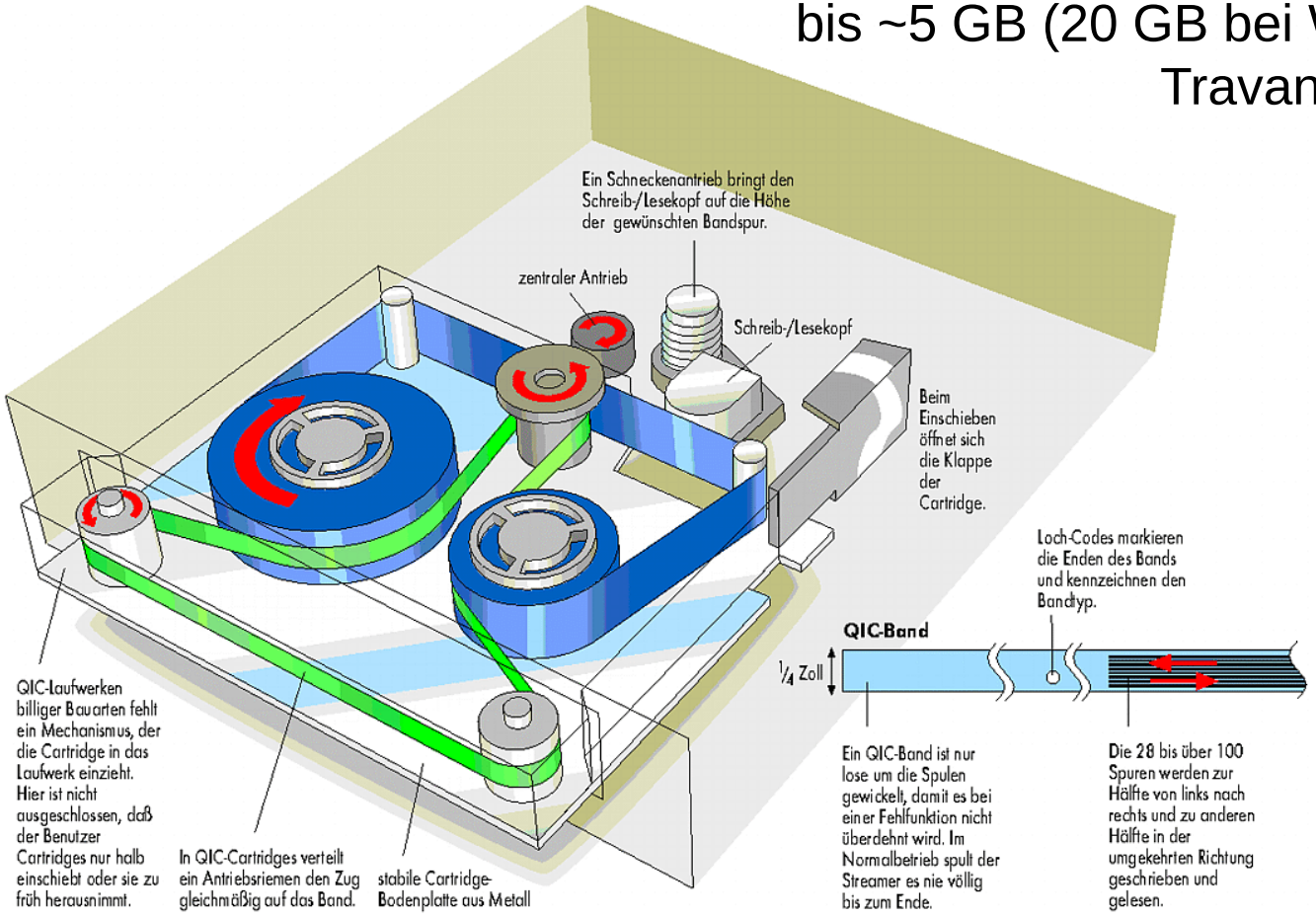
23,3/25/27 GB in einer Speicherebene
46,6/50/54 GB in zwei Speicherebenen
Experimentell 200 GB



[http://blu-raydisc.com/Assets/Downloadablefile/general_bluraydiscformat-15263.pdf]

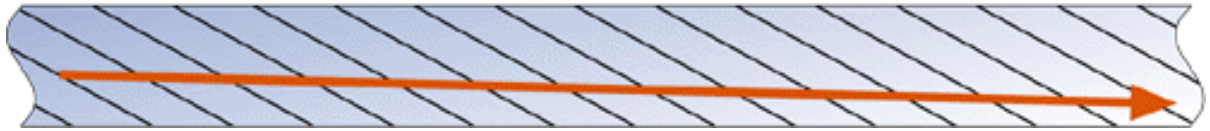
QIC-Laufwerk *(Quarter Inch Cartridge)*

Horizontale Aufzeichnung in mehreren (bis zu 100) Spuren nacheinander, optimiert auf Streaming-Betrieb mit hoher Bandgeschwindigkeit, bis ~5 GB (20 GB bei Weiterentwicklung Travan-5)

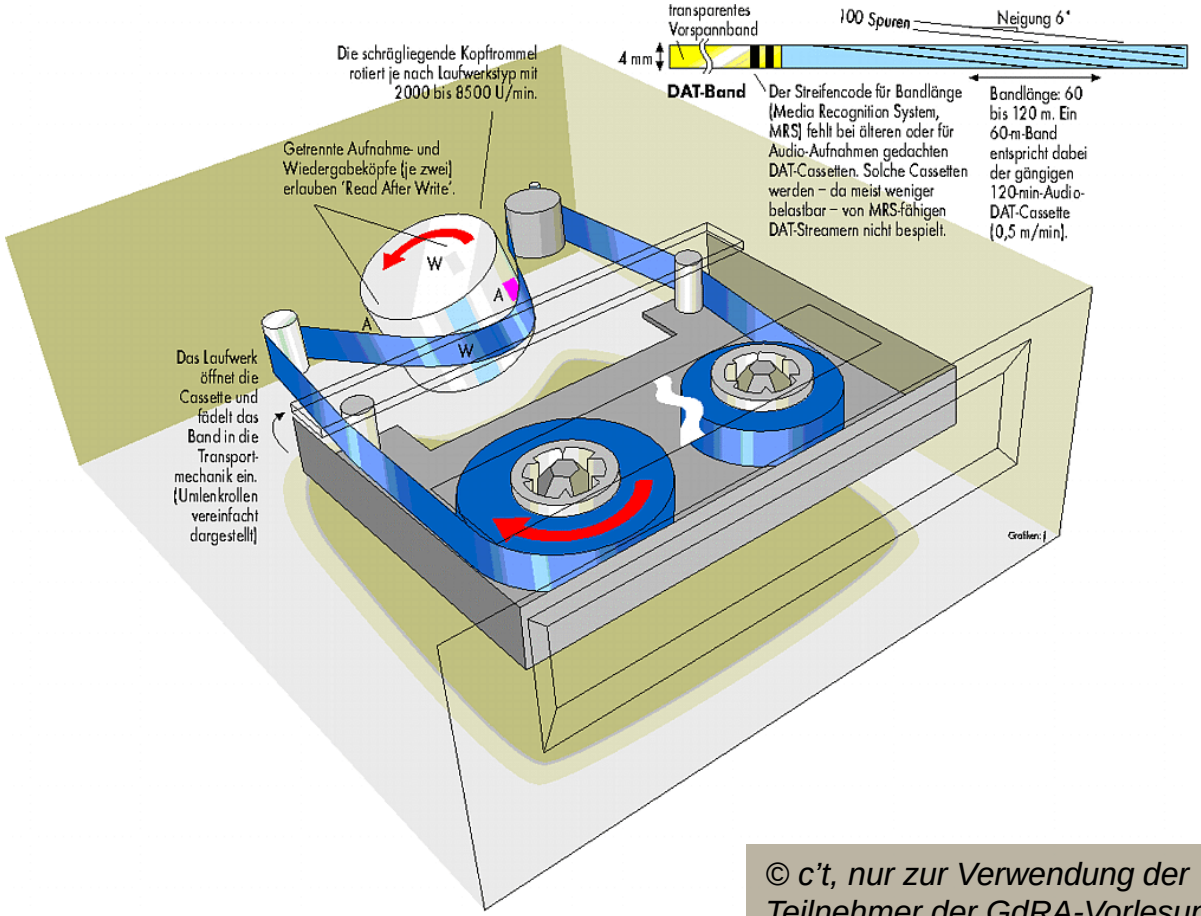


© c't, nur zur Verwendung der Teilnehmer der GdRA-Vorlesung.

DAT-Laufwerk *(Digital Audio Tape)*



Technik der *digital audio tapes*, Schrägspur-aufzeichnung, Weiterentwicklung DDS-4; bis 20 GB (40 GB komprimiert)



© c't, nur zur Verwendung der Teilnehmer der GdRA-Vorlesung.

Linear Tape Open (LTO)

- Offenes Herstellerkonsortium (IBM, HP, Quantum)
- Kontinuierlicher Entwicklungsprozess geplant: alle 2 Jahre neue Generation mit verdoppelter Bandkapazität und mind. 50% erhöhter Datentransferrate
- Laufwerke können Bänder der vorigen Generation verarbeiten, die der vorletzten Generation zumindest lesen
- LTO-4 Bänder können verschlüsselt werden (AES 256 Bit)
- LTO-Bänder enthalten 4KB Speicherchip im Kassettengehäuse (Seriennummer des Bandes, Nutzungslogs der letzten 100 *tape mounts*)

[de.wikipedia.org]

Linear Tape Open (LTO)

Übersicht der Generationen								
Laufwerks-Generation	Bandkapazität bei gleicher Generation		maximale Transferrate		kompatible Bänder		Verfügbarkeit	
	ohne Kompression	mit Kompression	ohne Kompression	mit Kompression	Schreiben	Lesen	Lizenzen seit	Produkte seit
Ultrium 1	100 GB [5]	200 GB [5]	20 MB/s	40 MB/s [5]	Ultrium 1	Ultrium 1	1998 [5]	2000 [5]
Ultrium 2	200 GB [5]	400 GB [5]	40 MB/s	80 MB/s [5]	Ultrium 1 Ultrium 2	Ultrium 1 Ultrium 2	April 2002 [5]	Ende 2002 [5]
Ultrium 3	400 GB [5]	800 GB [5]	80 MB/s	160 MB/s [5]	Ultrium 2 Ultrium 3	Ultrium 1 Ultrium 2 Ultrium 3	Juli 2004 [5]	Ende 2004 [5]
Ultrium 4	800 GB [5]	1.600 GB [5]	120 MB/s	240 MB/s [5]	Ultrium 3 Ultrium 4	Ultrium 2 Ultrium 3 Ultrium 4	Ende 2006 [5]	Frühjahr 2007 [6]
Ultrium 5	1.500 GB	3.000 GB [5]	140 MB/s	280 MB/s [5]	Ultrium 4 Ultrium 5	Ultrium 3 Ultrium 4 Ultrium 5	Sommer 2009 [7]	Frühjahr 2010 [8]
Ultrium 6	3.200 GB	8.000 GB [5]	210 MB/s	525 MB/s [5]	-	-	-	-
Ultrium 7	6.400 GB	16.000 GB [5]	315 MB/s	788 MB/s [5]	-	-	-	-
Ultrium 8	12.800 GB	32.000 GB [5]	472 MB/s	1180 MB/s [5]	-	-	-	-

[de.wikipedia.org]

Bewertung von Band-basierten Medien

- Lange Suchzeiten
- Geringe Transferraten
- Abnutzung
  - Schrägspur: „Hunderte von Durchläufen“
  - Parallelspur: „Tausende bis Millionen von Durchläufen“
- Ursprünglich deutlich kostengünstiger als Platten

- *Hennessy/Patterson: Gegenwärtiger Kostenvorteil?*
  Derzeit (2011): ~22€ für Archivierung von 800 GB über LTO-4
  ☞ Bietet immer noch Kosten- und Archivierungsvorteile

Zusammenfassung

Flüchtige Speicher

- SRAM / DRAM
- *Translation Look-Aside Buffer*, *Caches*
- Organisationsformen von TLBs und *Caches*
  direct mapped, *set-associative*, Assoziativspeicher
- Austauschverfahren
  Random, NRU, LRU
- *Cache*-Kohärenz

Nichtflüchtige Massenspeicher

- Festplatten: Traditionell preisgünstiges Speichermedium
  - RAID Festplattenverbünde bieten benötigte Zuverlässigkeit
- Optische Speicher (CD, DVD, Blu-ray): Kapazität steigt weiter an
- Bandspeicher: Zur Archivierung/Datensicherung günstiges Medium