technische universität
dortmund

fakultät für informatik
informatik 12

2.4 Speicherarchitektur

Peter Marwedel

Informatik 12

Otto-Hahn-Str. 16

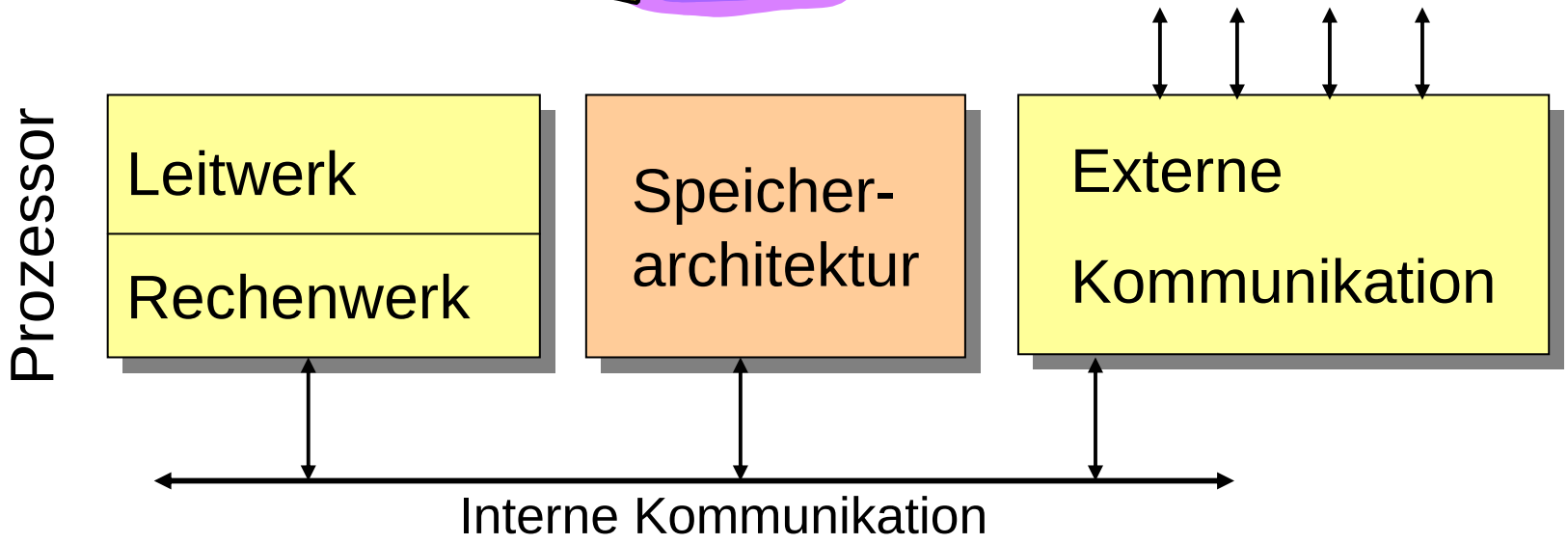
Tel. 755 6111

E-mail: peter.marwedel@tu-dortmund.de

Sprechstunde: Mo 13:00-14:00

2010/09/01

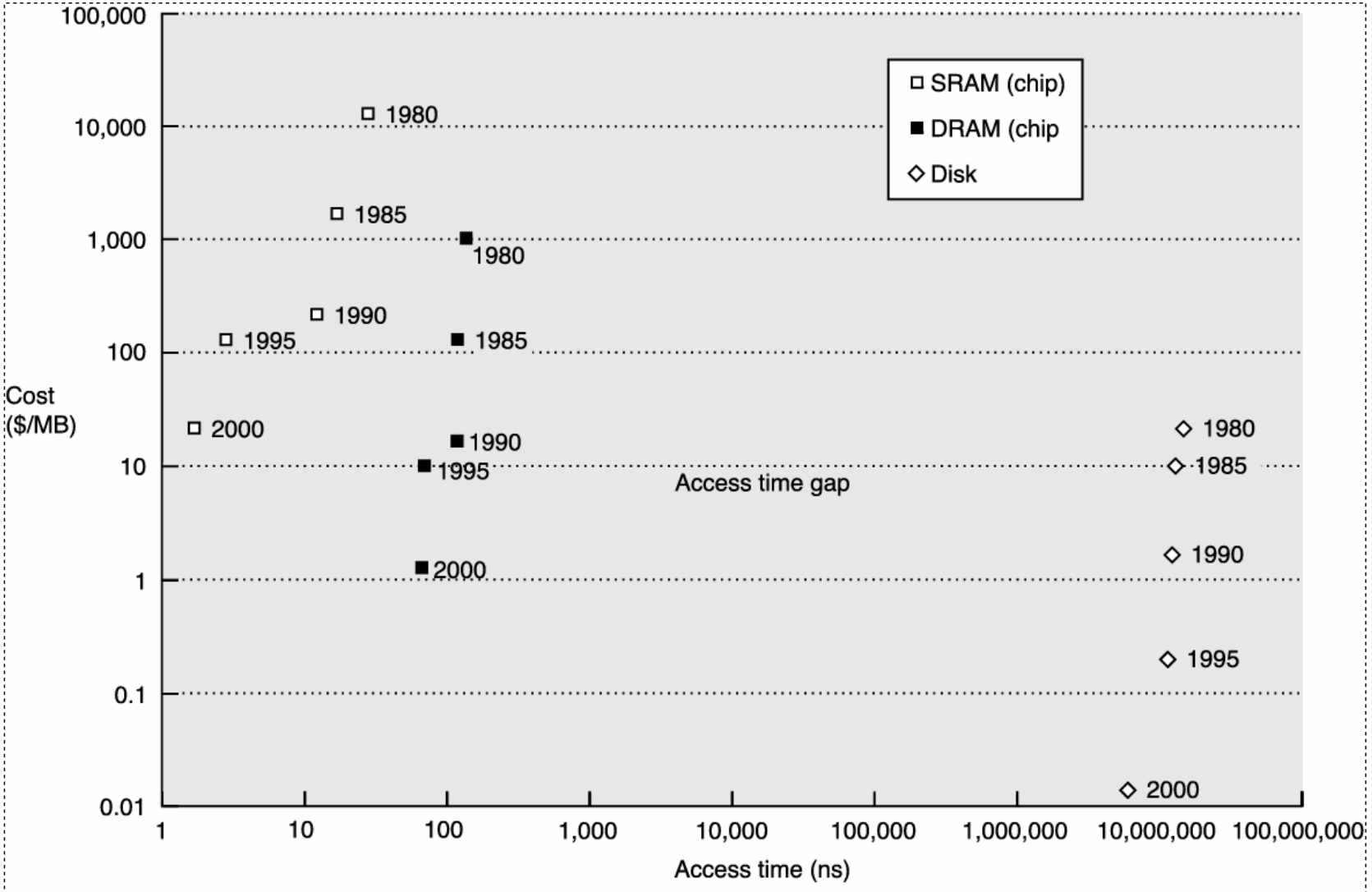
Kontext


Prozessor
Leitwerk
Rechenwerk
Speicher-
architektur
Externe
Kommunikation
Interne Kommunikation

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

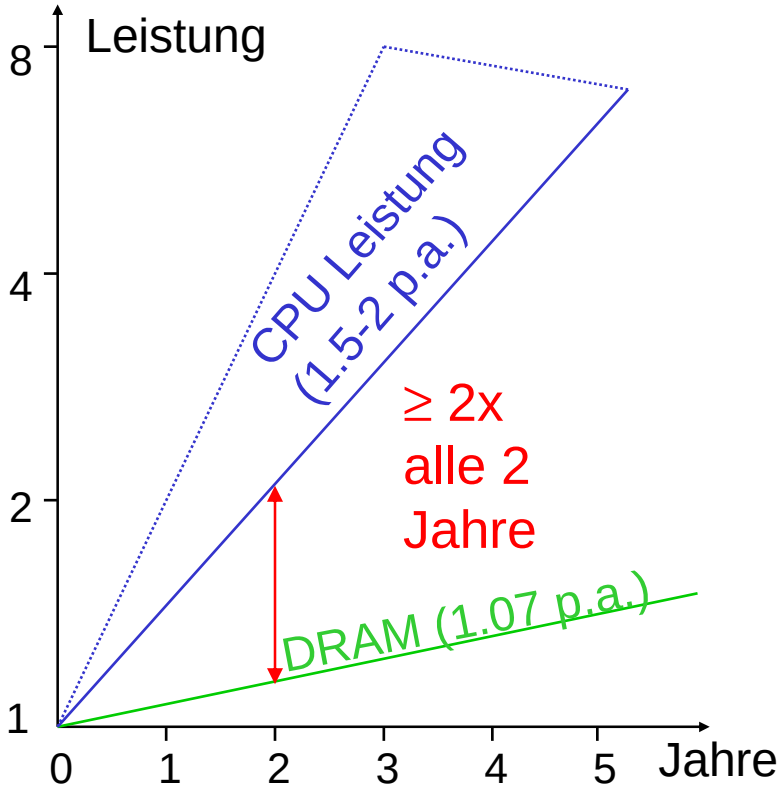
Die Realität: Kosten/Mbyte und Zugriffszeiten für verschiedene Speichermedien



[Hennessy/Patterson, *Computer Architecture*, 3. Aufl.]

Trend der Leistungen von DRAM

Die Leistungsslücke zwischen Prozessoren und DRAM wächst



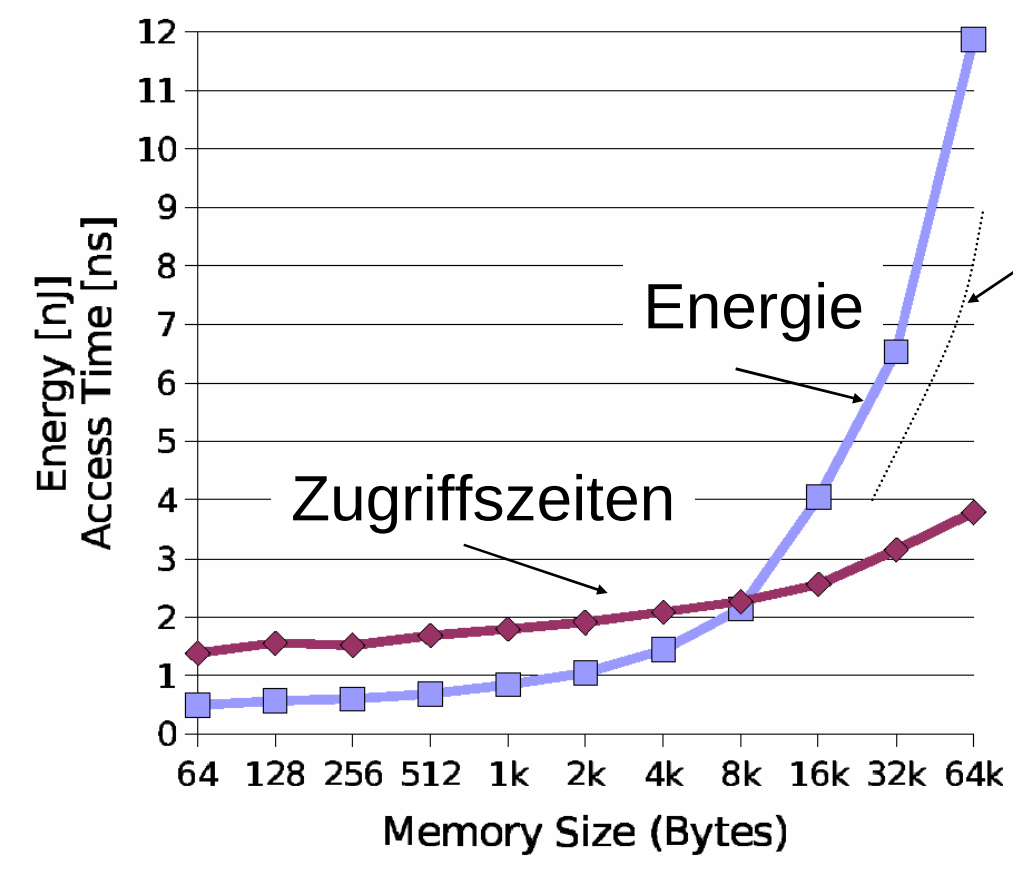
Ähnliche Probleme auch für Eingebettete Systeme

☞ In Zukunft:
Speicherzugriffszeiten >>
Prozessorzykluszeiten

☞ „*Memory wall*"
Problem

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

Abhängigkeit von der Speichergröße

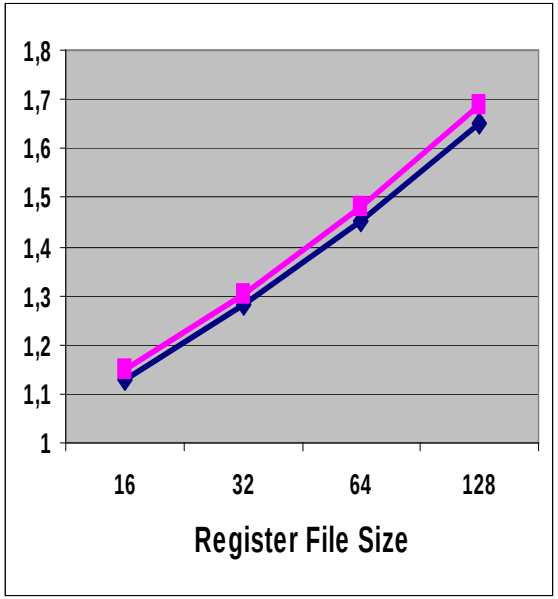


Anwendungen werden größer und größer …

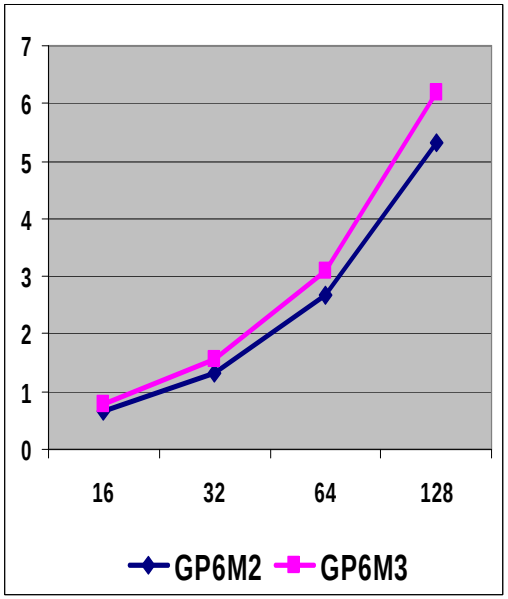
Quelle: CACTI

„Alles“ ist groß für große Speicher

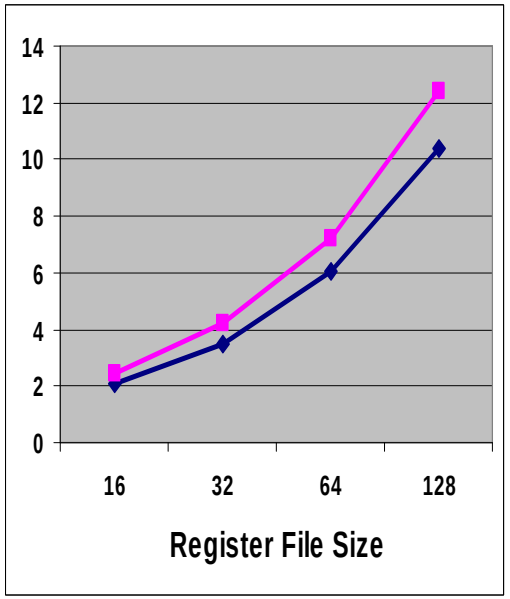
Zykluszeit (ns)*



Fläche (λ^2x10^6)



El. Leistung (W)

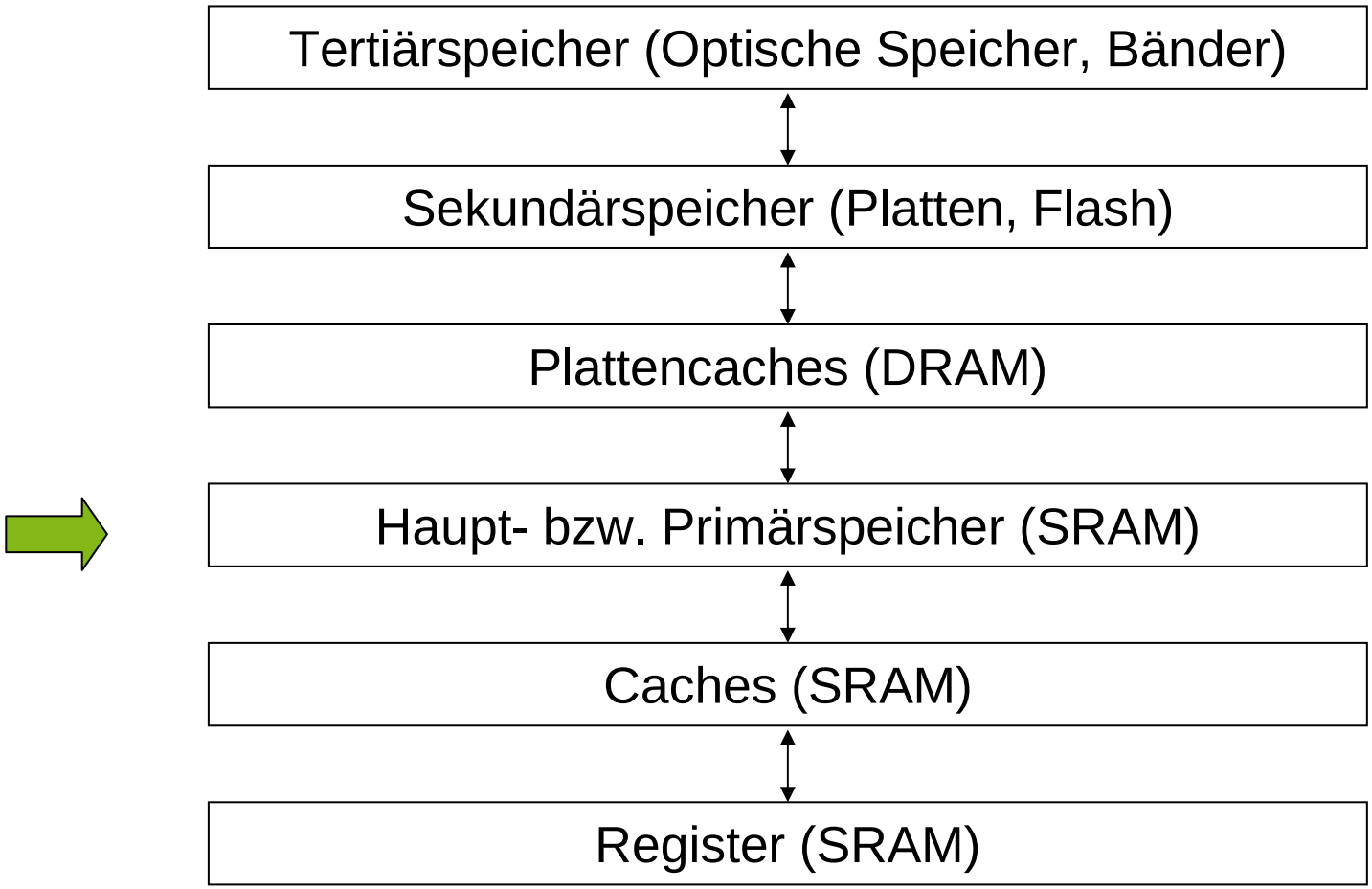


* Monolithic register file; Rixner’s et al. model [HPCA’00], Technology of 0.18 mm; VLIW configurations for a certain number of ports („GP*x*M*y*REG*z* where: x={6}, y={2, 3} and z={16, 32, 64, 128“}; Based on slide by and ©: Harry Valero, U. Barcelona, 2001

Speicherhierarchie

- Große Speicher sind langsam
- Anwendungen verhalten sich üblicherweise lokal
- ☞ Wir versuchen, häufig benötigte Speicherinhalte in kleinen Speichern, seltener benötigte Inhalte in großen Speichern abzulegen
- ☞ Einführung einer „Speicherhierarchie“
- Man möchte durch diese Kombination die Illusion eines großen Speichers mit kleinen Zugriffszeiten erzielen, zumindest im Mittel
- Eine enge Grenze für die Zugriffszeit ist vielfach kaum zu garantieren
- ☞ Bei Realzeitsystemen ergeben sich spezielle Überlegungen

Mögliche Stufen der Speicherhierarchie
und derzeit eingesetzte Technologien

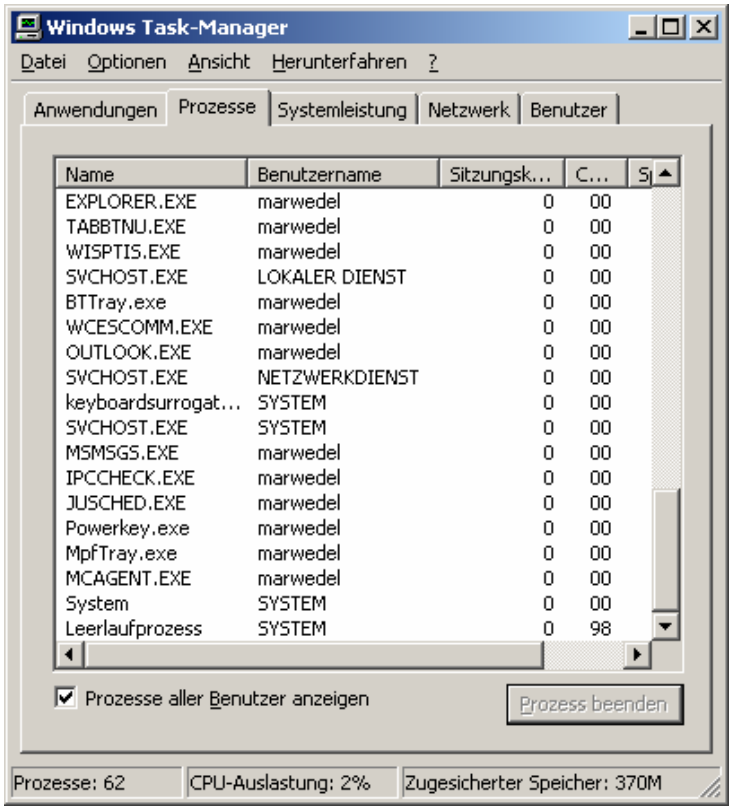


2.4.2 Mehrprozess-System (*Multitasking, Multiprocessing*)

In realen Rechnern meist mehrere „Programme“ gespeichert.

Ausführbare Programme heißen **Prozesse** oder ***tasks*.**

Prozessen wird durch einen ***dispatcher*** (z.TL. ***scheduler*** genannt) Rechenzeit auf dem Prozessor zugeteilt.



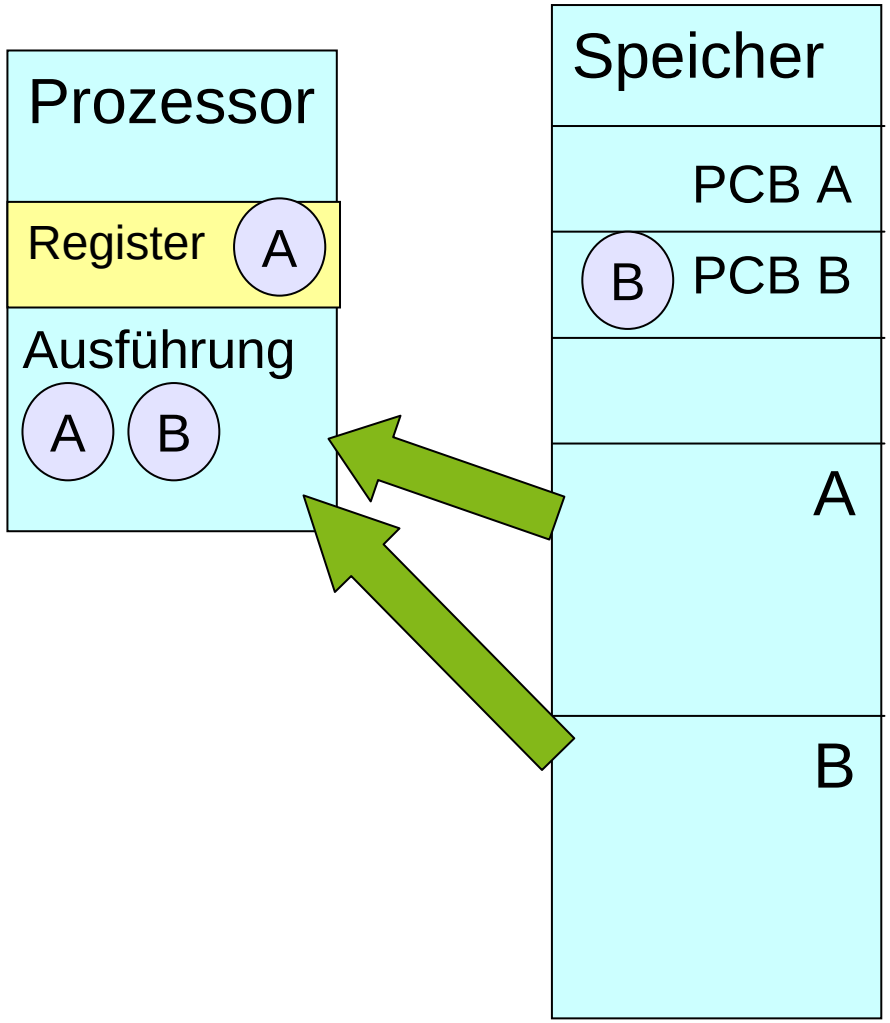
Dispatcher

Dispatcher schaltet zwischen Ausführung von Prozessen um (***context switch***). Bei *context switch* werden alle Registerinhalte

- des anzuhaltenden Prozesses in einen ihm zugeordneten Datenblock (***process control block, PCB***) gerettet &
- die des zu fortzuführenden Prozesses aus seinem PCB geladen.

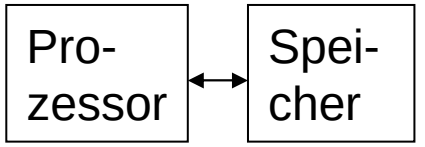
Prozessen wird suggeriert, Prozessor gehöre ihnen allein (bis auf Zeitverhalten).

Context switch



Speicherverwaltung

Unterscheidung zwischen den Adressen, welche der Prozessor generiert und jenen, mit denen der physikalische Speicher adressiert wird.



Def.: Prozessadressen bzw**. virtuelle Adressen** sind die effektiven Adressen nach Ausführung aller dem Assemblerprogrammierer sichtbaren Adressmodifikationen, d.h. die von einem Prozess dem Rechner angebotenen Adressen.

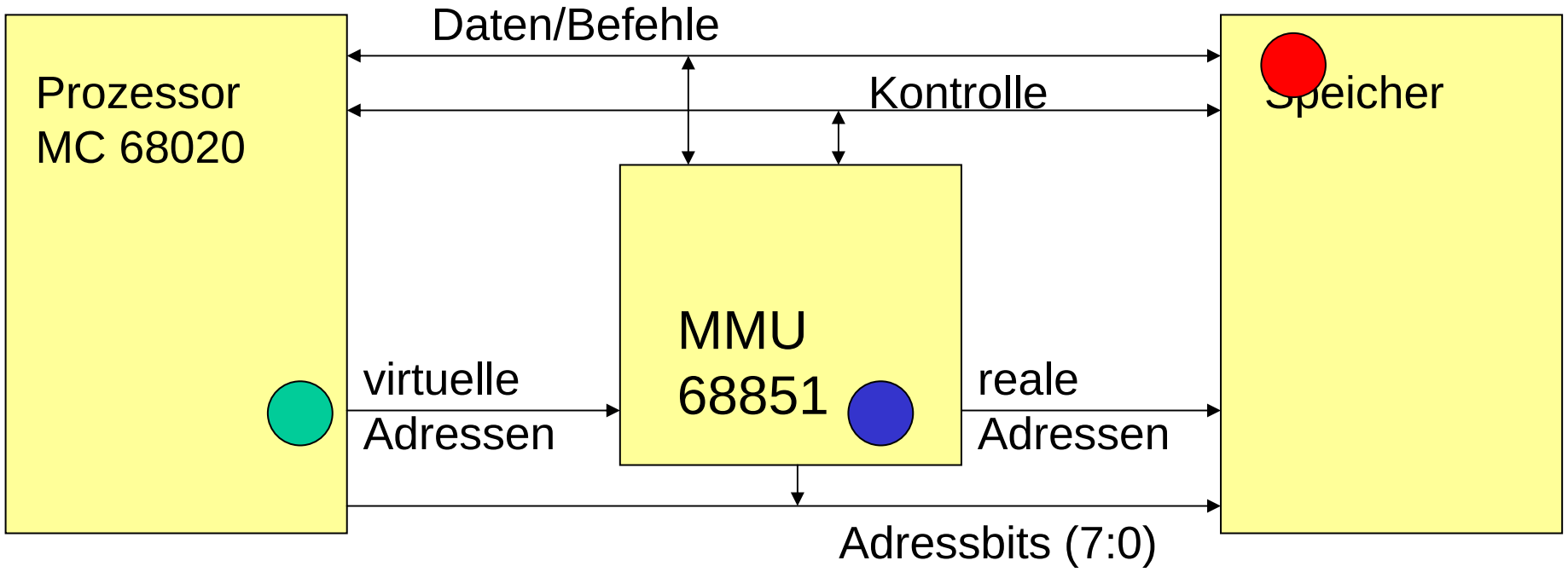
Def.: Maschinenadressen, reale Adressen bzw**. physikalische Adressen** sind die zur Adressierung der realen Speicher verwendeten Adressen.

* http://www.acquacetosaricerca.it/Immagini2/VirtualReality.jpg

Memory Management Unit (MMU)

MMU ist für das Umrechnen der Adressen verantwortlich.

Beispiel Motorola



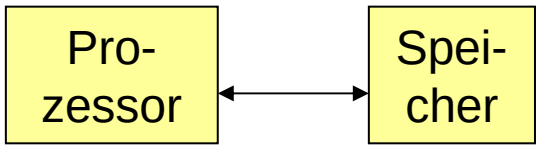
Identität

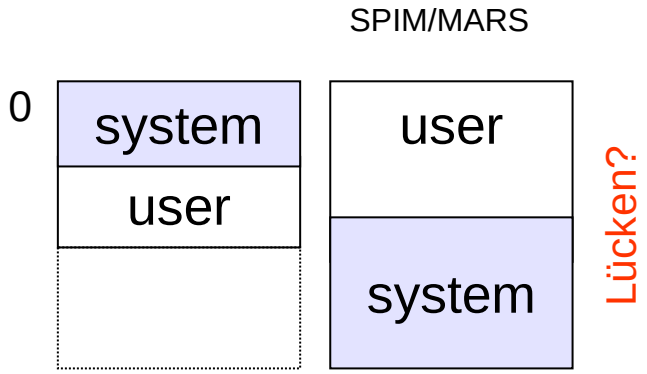
Bei einfachen Systemen sind reale und virtuelle Adressen identisch *(„memory management without swapping or paging“).*

Adressraum durch real vorhandenen Speicher beschränkt.

Kommt v.a. bei eingebetteten Systemen (Fahrzeugelektronik, Handys usw.) vor, weniger bei PCs und Workstations.

Sehr einfache Systeme:
- Aufteilung in Systembereich und Benutzerprogramm (☞ SPIM/MARS)



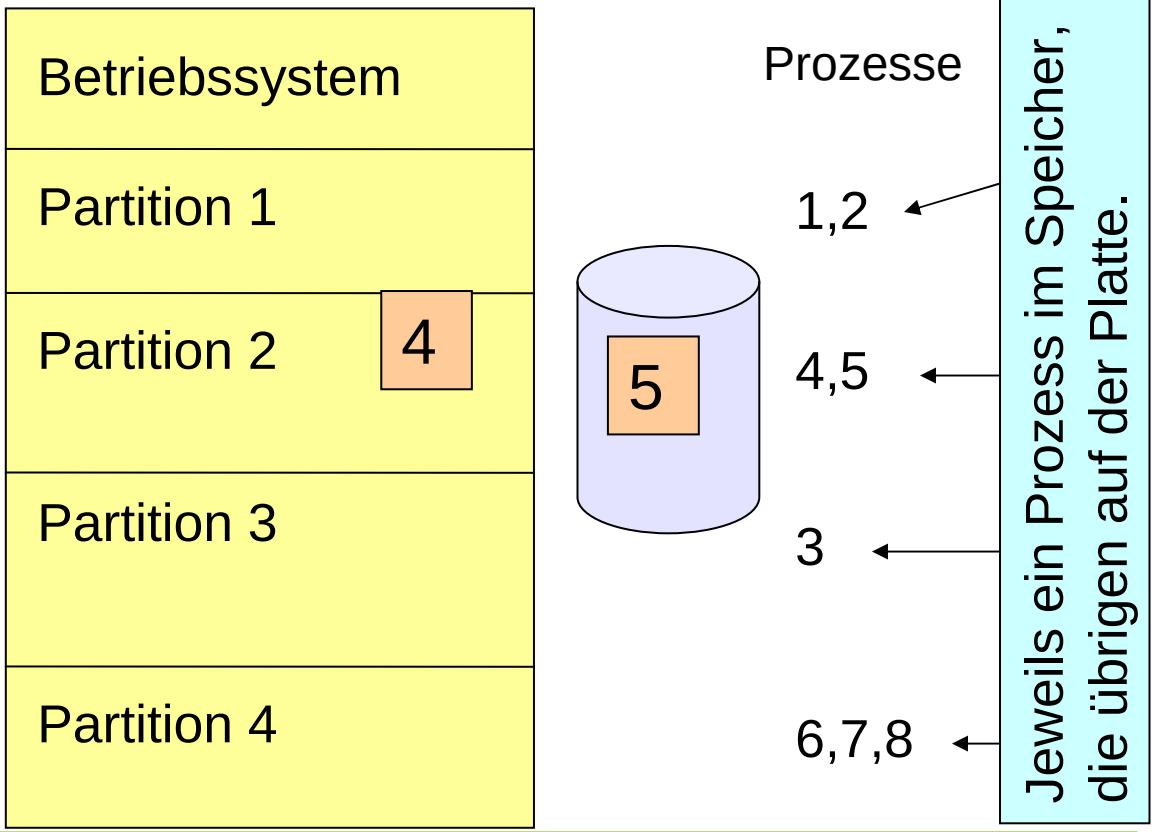


Einteilung in Laufbereiche (*core partitions*)

Aus Aufwandsgründen statt flexibler Speicherzuteilung an Prozesse feste Partitionierung des Speichers. *

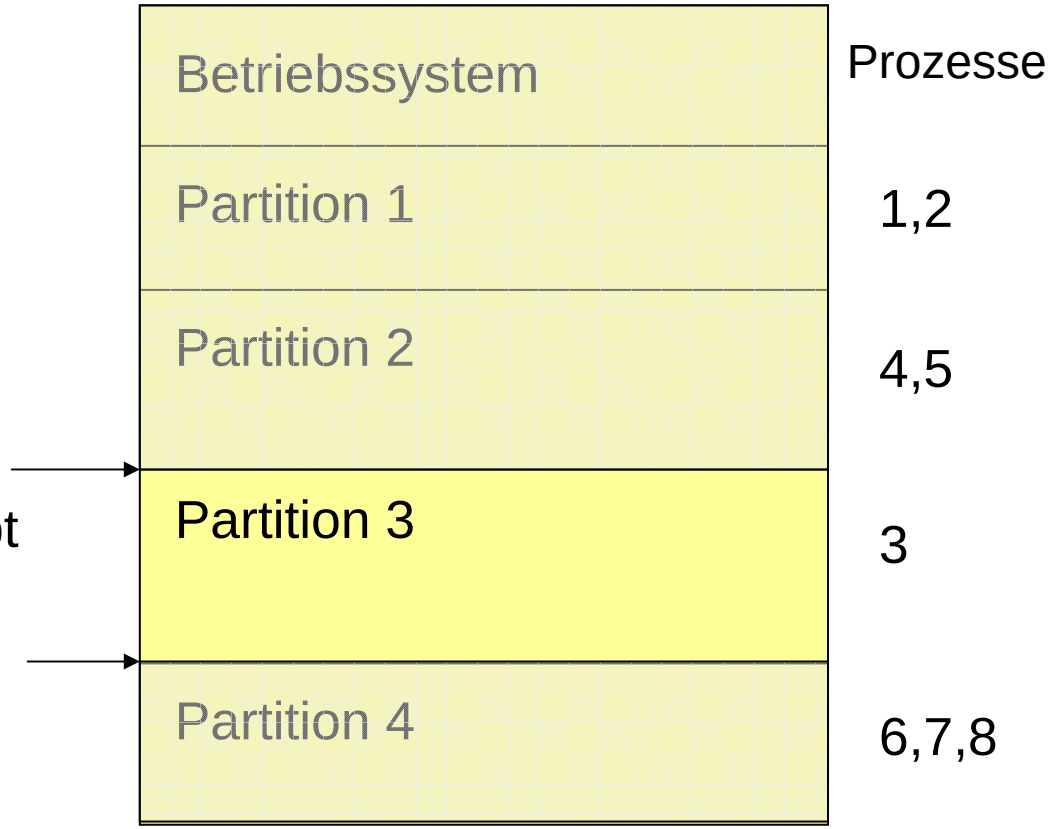
Adressbereich der Prozesse zusammen-hängend im virtuellen und im realen Adressraum.

Compiler adressieren Befehle ab Adresse 0. Beim Laden muss auf alle Adressen Anfangsadresse der Partition addiert werden. ☞ Identifikation von Adressen im übersetzten Programm!



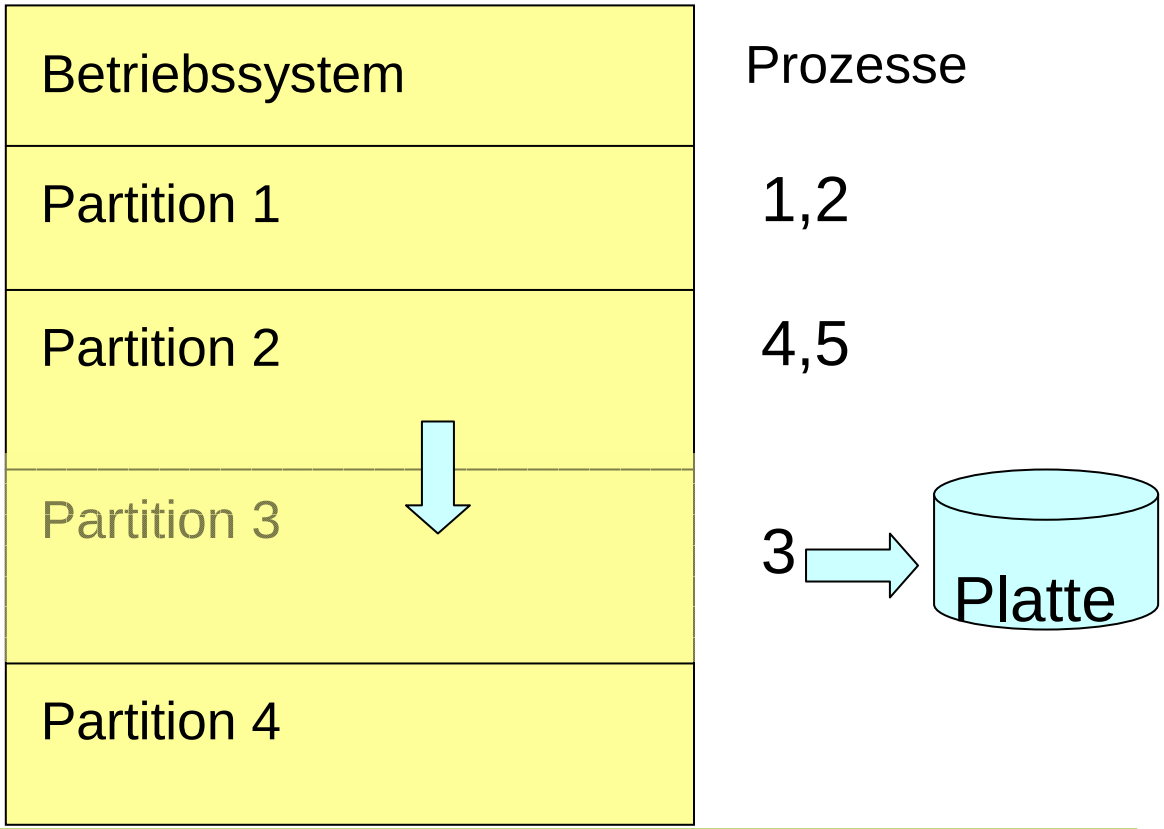
Speicherschutz durch Grenzregister möglich

Z.Tl. nur Zugriff auf durch Grenzregister begrenzten Speicherbereich erlaubt (☞ Speicherschutz); Grenzregister werden bei *context switch* umgeladen.



Was tun, wenn Prozesse mehr Speicher benötigen? (1)

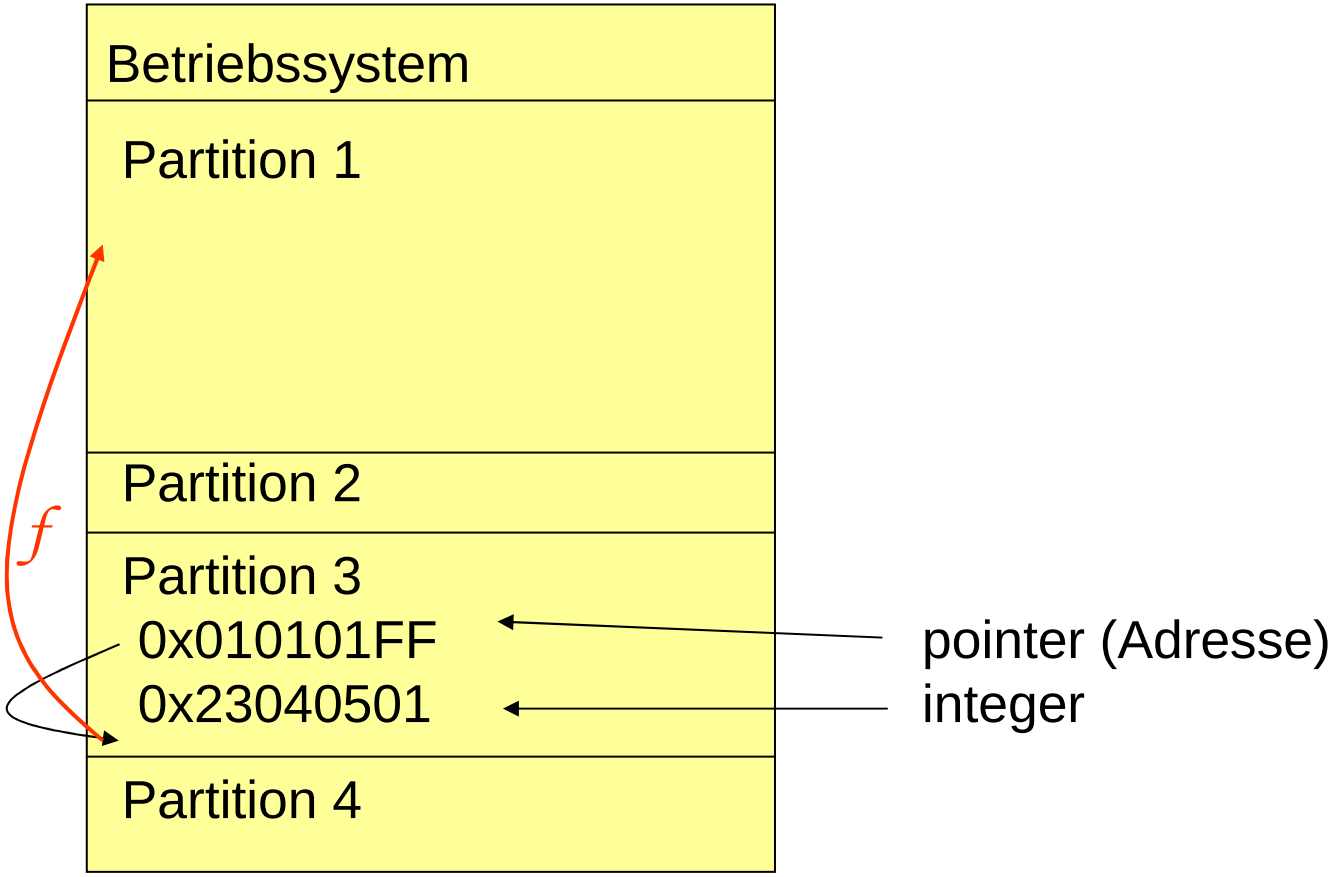
Man könnte Partitionen verschmelzen, wenn Prozesse mit dem Speicher der zugeteilten Partition nicht auskommen. Aus Aufwandsgründen selten realisiert.



Was tun, wenn Prozesse mehr Speicher benötigen? (2)

Verschieben von Prozessen im Speicher nicht möglich, da Adressen angepasst werden müssten, andere Informationen nicht.
Im Speicher sind Adressen und andere Informationen ununterscheidbar

Aus demselben Grund kann nur der gesamte Inhalt einer Partition auf die Platte ausgelagert werden (***swapping***) und an dieselbe Stelle zurückgeschrieben werden.



Einsatzbereiche

Sehr einfache Rechner, eingebettete Systeme (Handys, Fahrzeugelektronik).
Systeme, bei denen die Programme von vornherein alle bekannt sind.

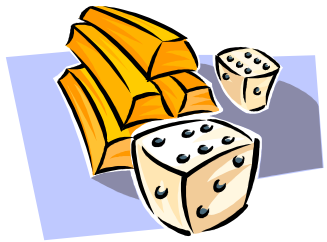
Beurteilung

Nachteile:

Sehr starke „externe Fragmentierung“
(Speicherbereiche bleiben ungenutzt).

Adressraum durch real vorhandenen Speicher beschränkt.

Man muss den Speicherbedarf von Applikationen recht genau kennen, um eine möglichst passende Speichergröße zuzuweisen.



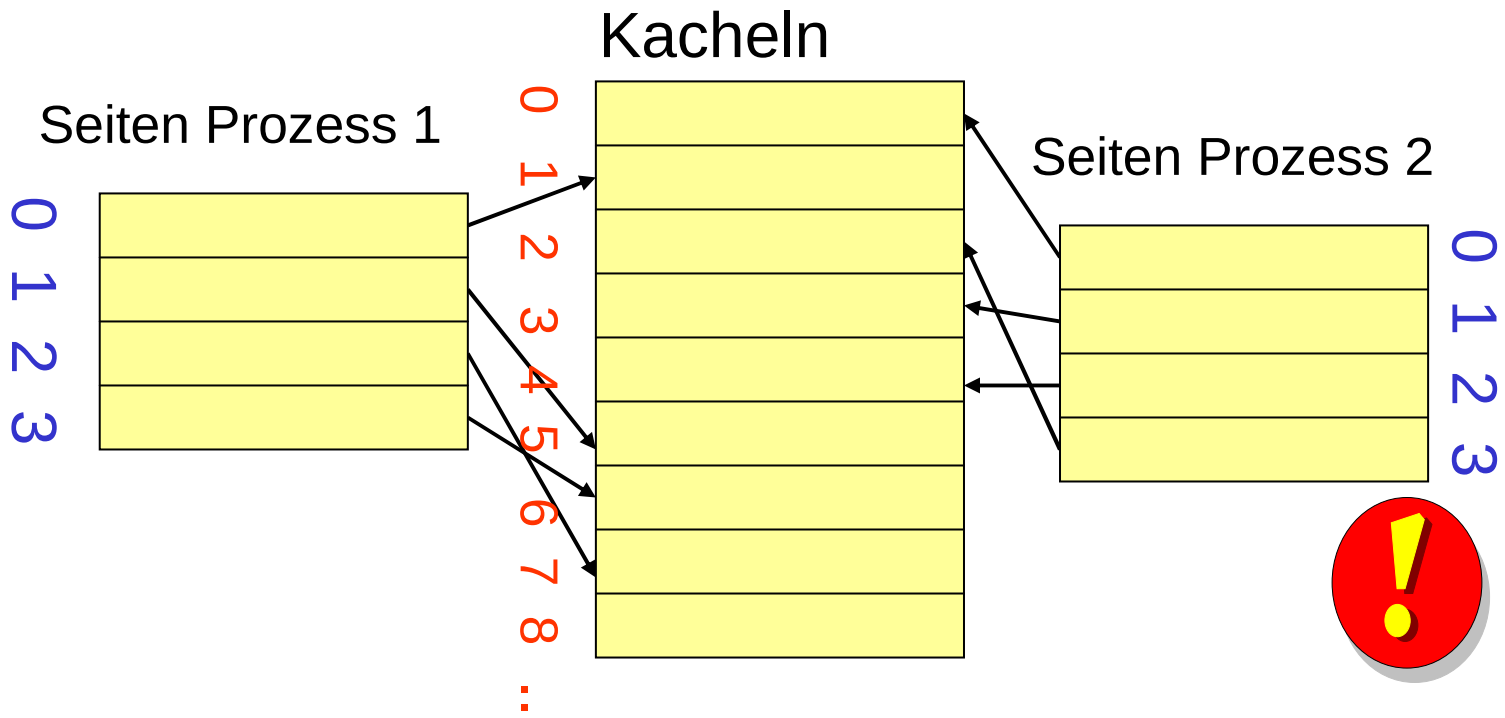
Risiko des Prozessabbruchs, wenn Speicher nicht ausreicht.

☞ Gründe für bessere Verfahren.

Seiten-Adressierung (*paging*)

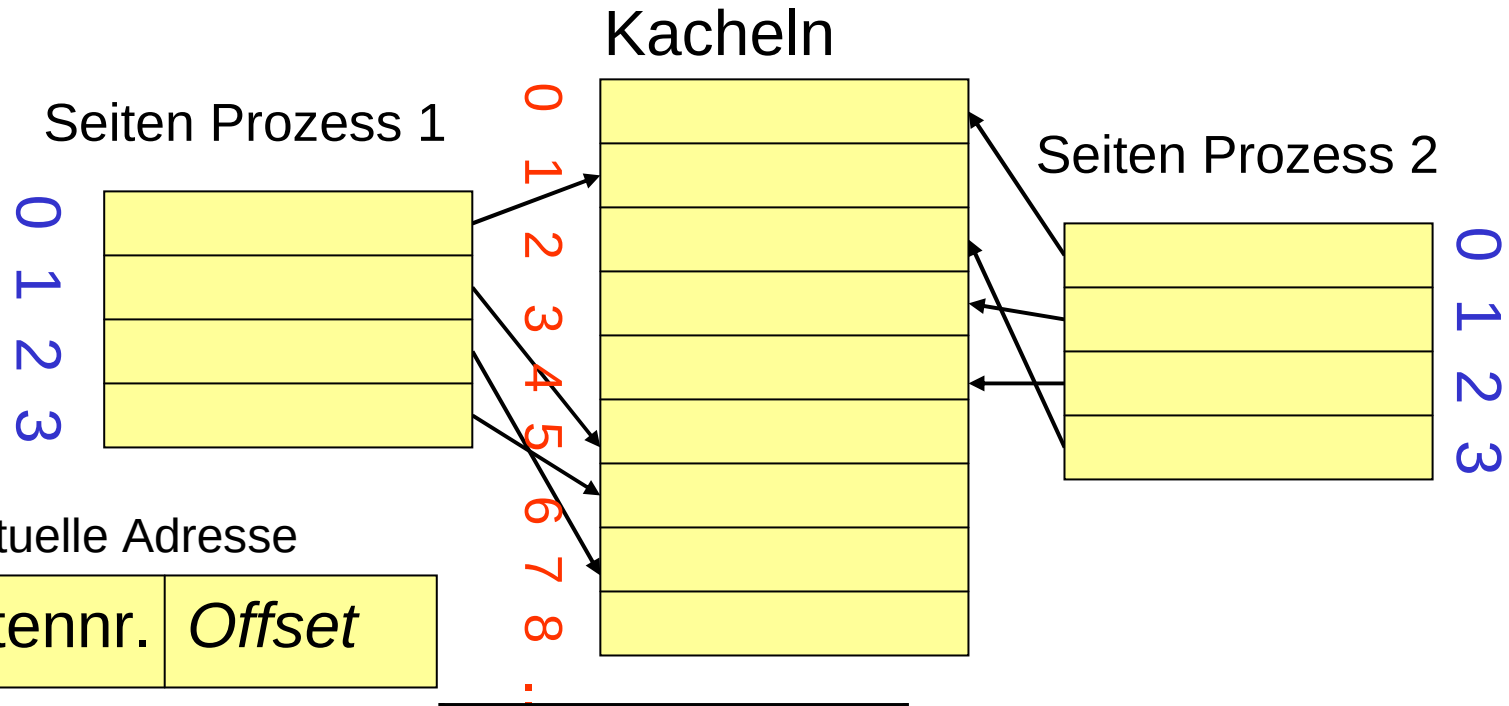
- Einteilung der realen Adressen in Bereiche gleicher Länge (**Seitenrahmen, Kacheln, *page frames***).

Jede Seite kann auf jeder Kachel gespei-chert werden.



- Einteilung der virtuellen Adressen in Bereiche konstanter Länge (**Seiten, *pages***)

Seiten-Adressierung (*paging*)



P1:Seite	Kachel
0	1
1	5
2	7
3	6

Zu jedem Prozess existiert eine Tabelle, die **Seitentabelle.**

reale Adresse = Anfangsadresse der Kachel + *Offset*

Vorteile

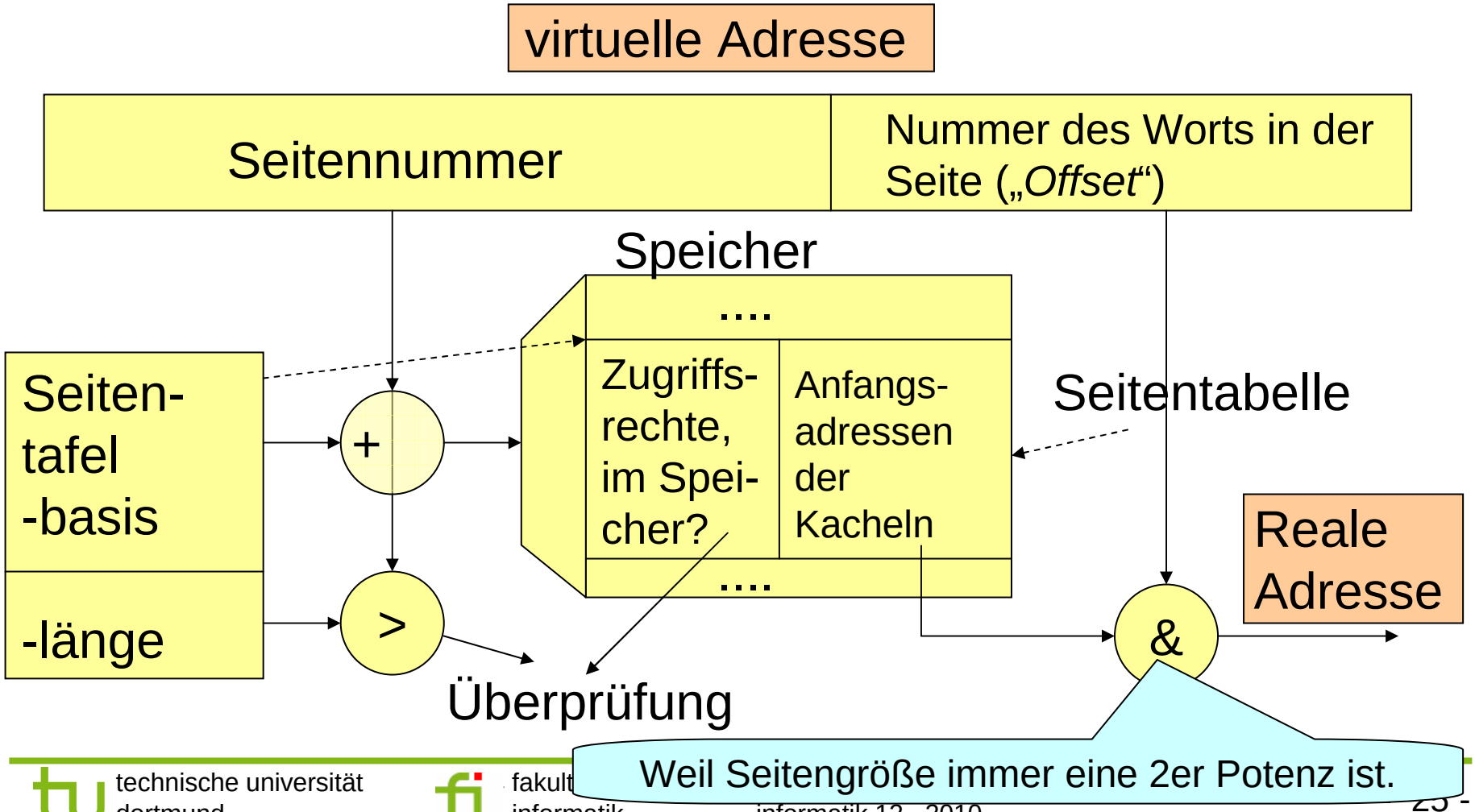
- Man muss sich nicht vorab überlegen, welchem Speicherbereich (*text, stack ,heap*) man wie viel Speicher zuweist.
- Jede Kachel des Speichers kann genutzt werden (keine „**externe Fragmentierung**“).
- Lücken im virtuellen Adressraum (wie beim MARS/SPIM) sind kein Problem.
- Virtuelle Adressräume aller Prozesse können bei 0 beginnen, nur bei getrennt übersetzten Prozeduren sind Adressanpassungen erforderlich.

Nachteil (☹):

- „**interne Fragmentierung**“
  (Seiten nicht unbedingt gefüllt)

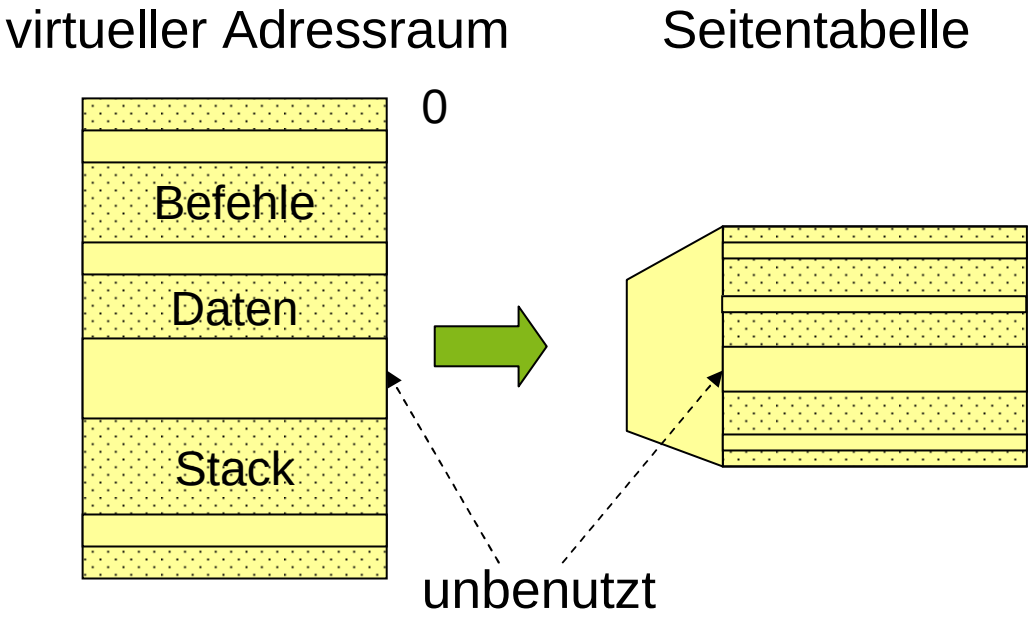
Organisation der Adressabbildung

Schnelle Umrechnung von virtuellen auf reale Adressen!
Bei zusammenhängenden Adressen und Seitenadressierung:



Lücken im virtuellen Adressraum

Bei MIPS-Konvention: Adressbereiche zwischen Stack, Befehls- und Datenadressen werden meist nicht benutzt.



Ungenutzte Seiten- bzw. Segmentnummern: bei der bisherigen Methode werden Tafeleinträge nicht benutzt!

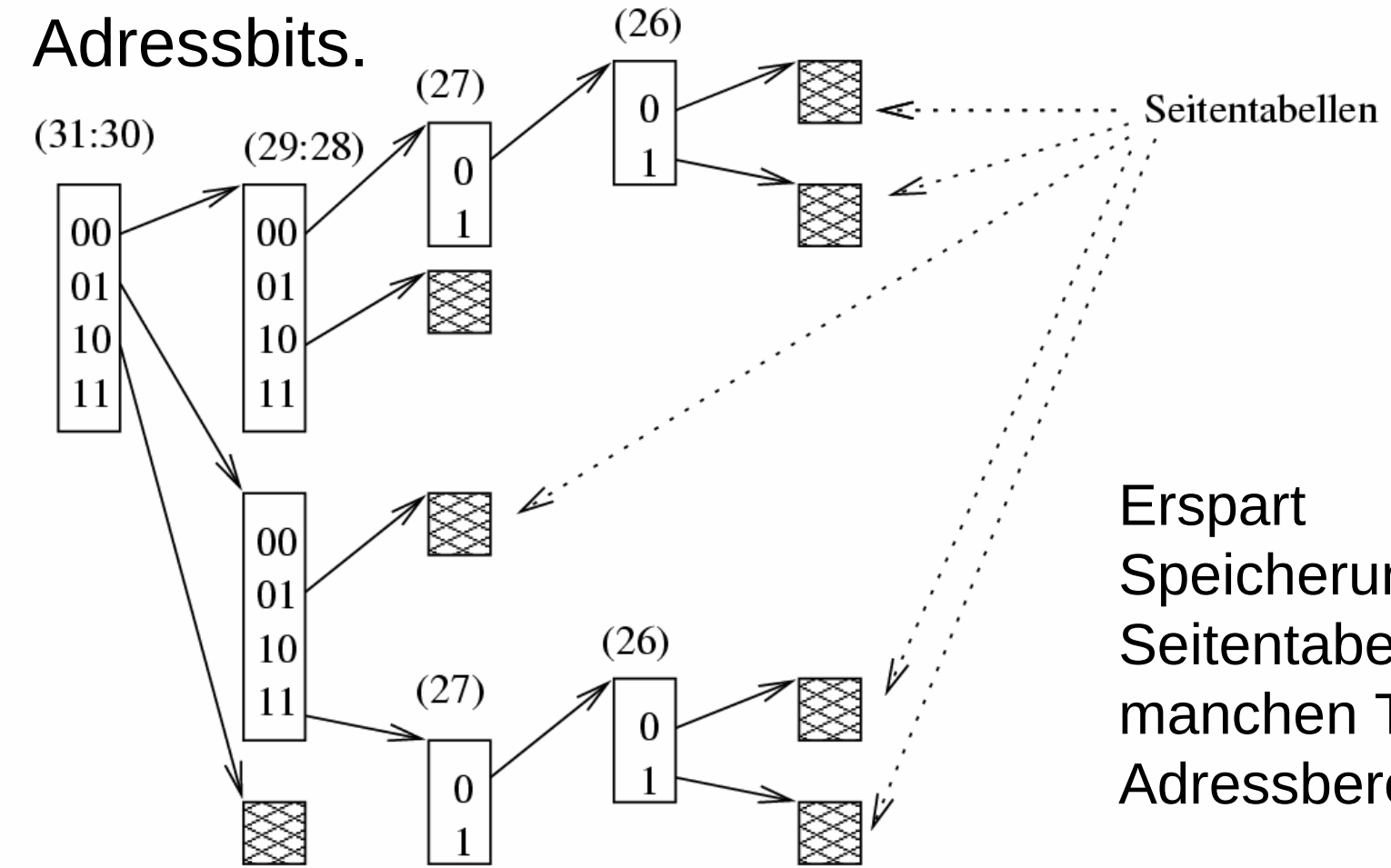
Seitentafel benötigt großen zusammenhäng. Speicherbereich.

☞ Berücksichtigung von Lücken im virtuellen Adressraum:

nicht für jede **mögliche** Seite auch Einträge in der Seitentafel!

Hierarchische Seitentabellen

Mehrfache Verzweigung anhand einzelner Adressbits.

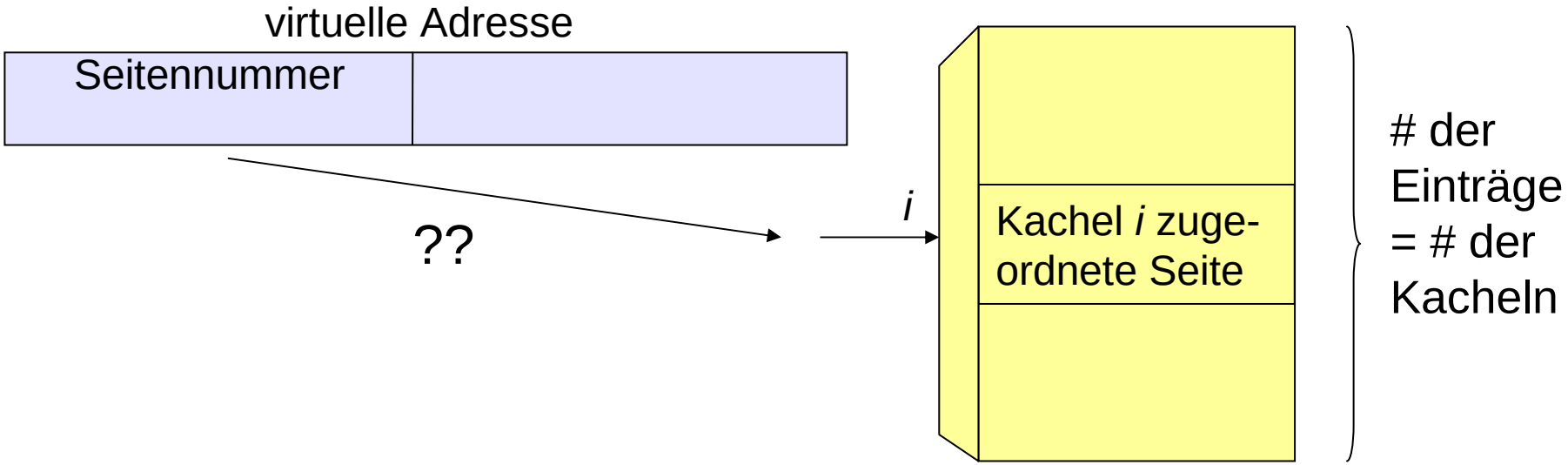


Erspart Speicherung von Seitentabellen in manchen Teilen von Adressbereichen

Invertierte Seitentabelle

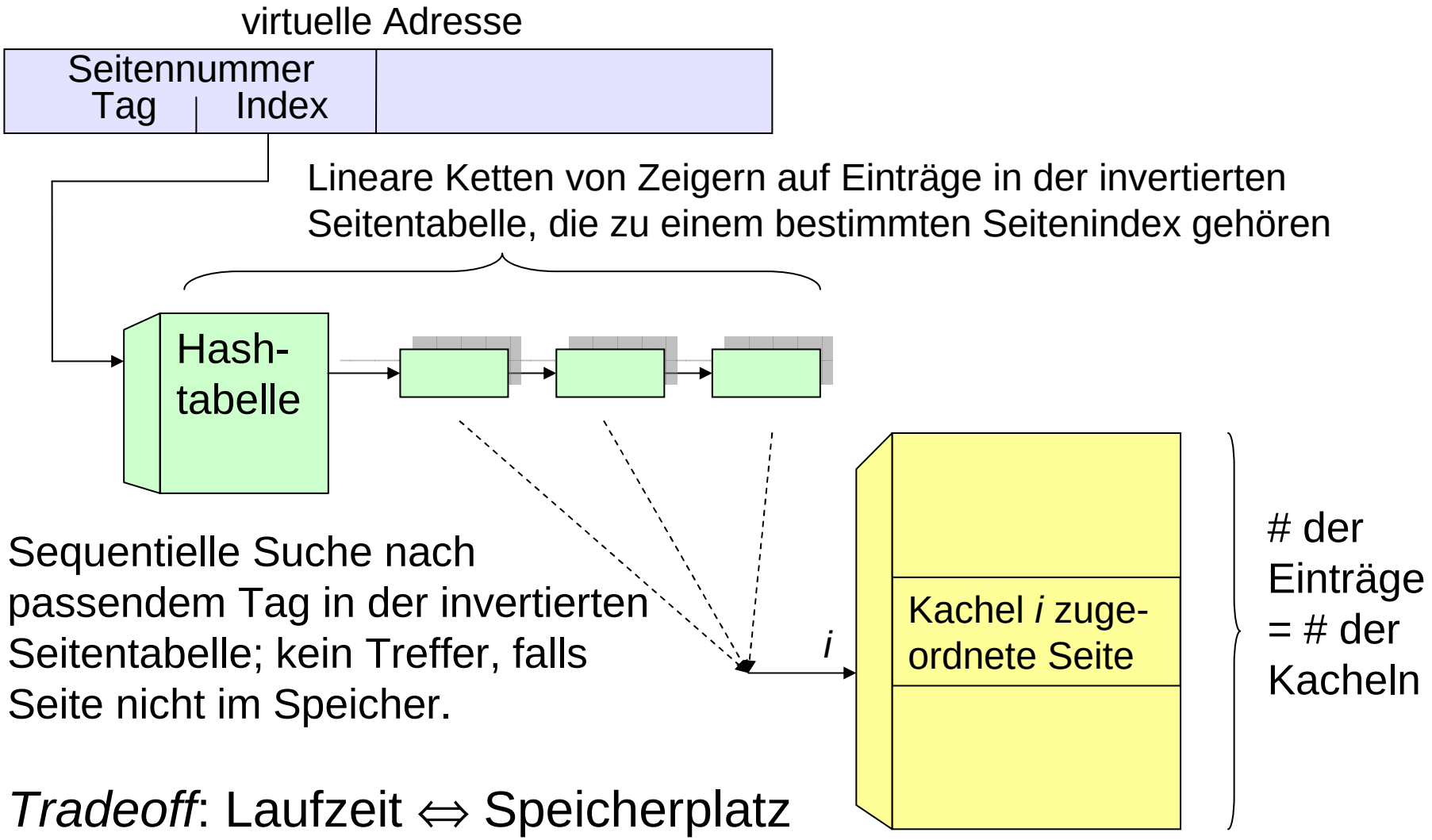
Selbst bei hierarchischen Seitentabellen kann ein großer Speicherbedarf entstehen, v.a. bei großen Adressräumen

☞ „invertierte Seitentabellen“



Suche durch die gesamte Kacheltabelle wäre zu langsam

Beschleunigung mittels Hashtabellen

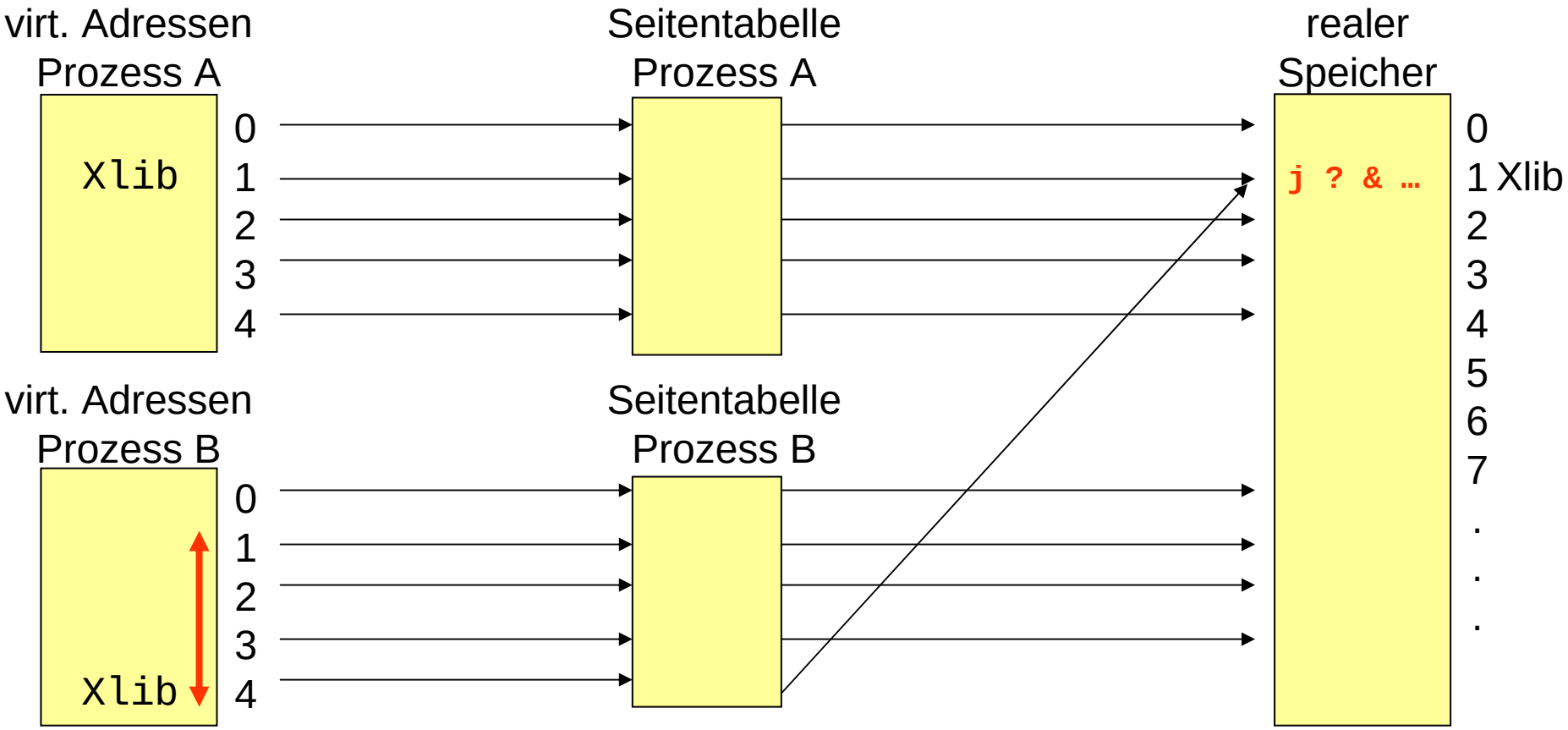


Zusammenfassung

- Viele Rechensysteme unterstützen *Multitasking*
- Reale und virtuelle Adressen, *memory management*
- Formen der Speicherverwaltung
  - Identität
  - Seitenadressierung
    - Sharing
  - Segmentadressierung
  - Segmentadressierung mit Seitenadressierung

Sharing (*shared libraries*)

Einfache Idee: Seiten-Abbildungen mehrerer Prozesse führen zur selben Kachel



Idee ist so nicht realisierbar

- Im gemeinsam genutzten Code kommen (virtuelle) Befehlsadressen vor, die sich z.B. auf Adressen in diesem Code beziehen.
- Adressen müssen Seitennummern verwenden, die nach Abbildung auf die realen Adressen über die jeweiligen Seitentafeln wieder auf den gemeinsam genutzten Bereich verweisen.
- Wenn diese Seitennummern ≠ sind: ☞ nicht im Code eintragbar (in 1 Speicherzelle nur 1 Seitennummer).
- **Nutzen Prozesse gemeinsamen Code, so muss dieser bei der Seitenadressierung bei allen beteiligten Prozessen auf dieselben virtuellen Adressen abgebildet werden.**

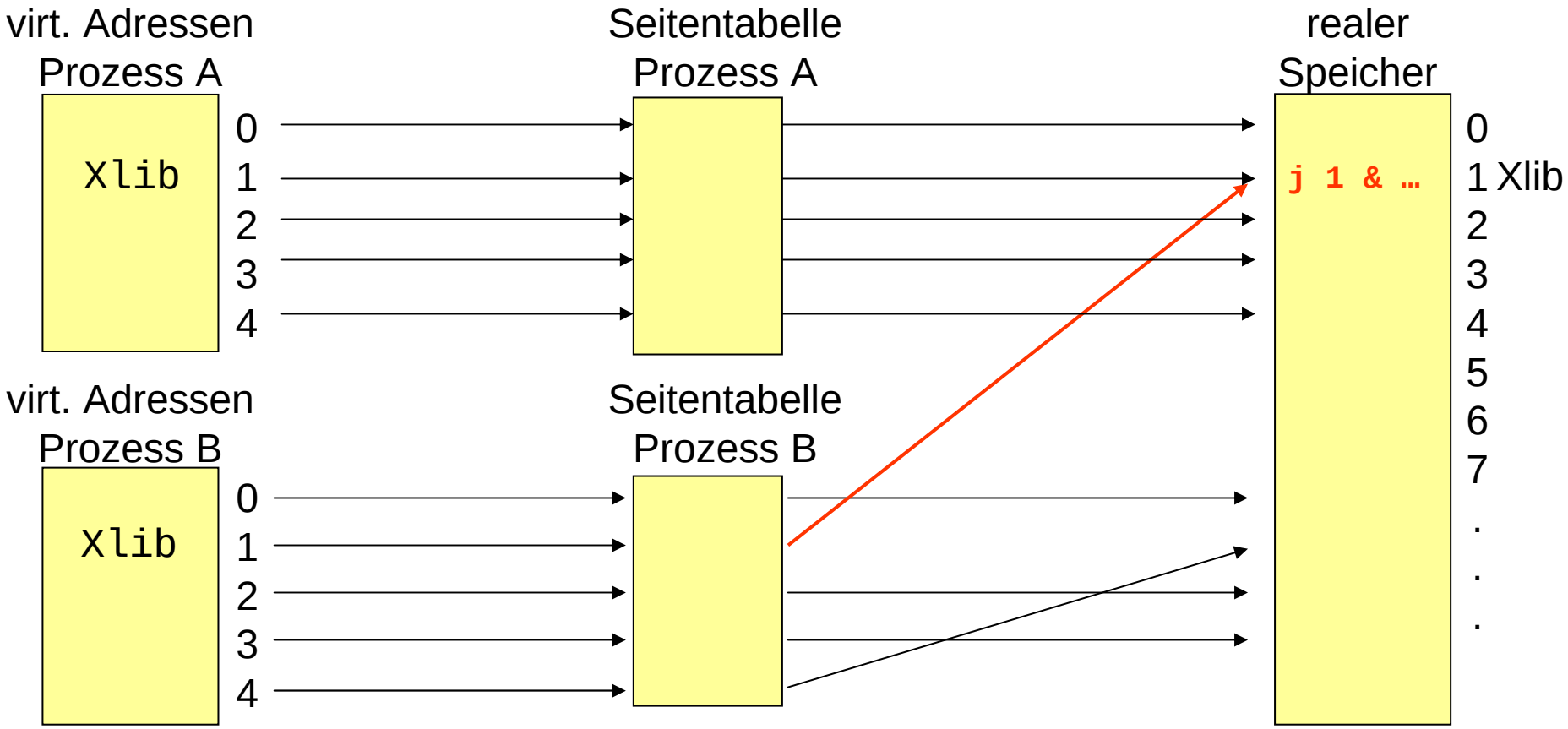
Idee ist so nicht realisierbar (2)

- **Nutzen Prozesse gemeinsamen Code, so muss dieser bei der Seitenadressierung bei allen beteiligten Prozessen auf dieselben virtuellen Adressen abgebildet werden.**
- Windows nutzt für gemeinsamen Code die obere Hälfte des Adressbereichs und benutzt Absprachen zur Adressvergabe.

?

Realisierbares *Sharing*

Shared libraries befinden sich an derselben Stelle des virtuellen Adressraums

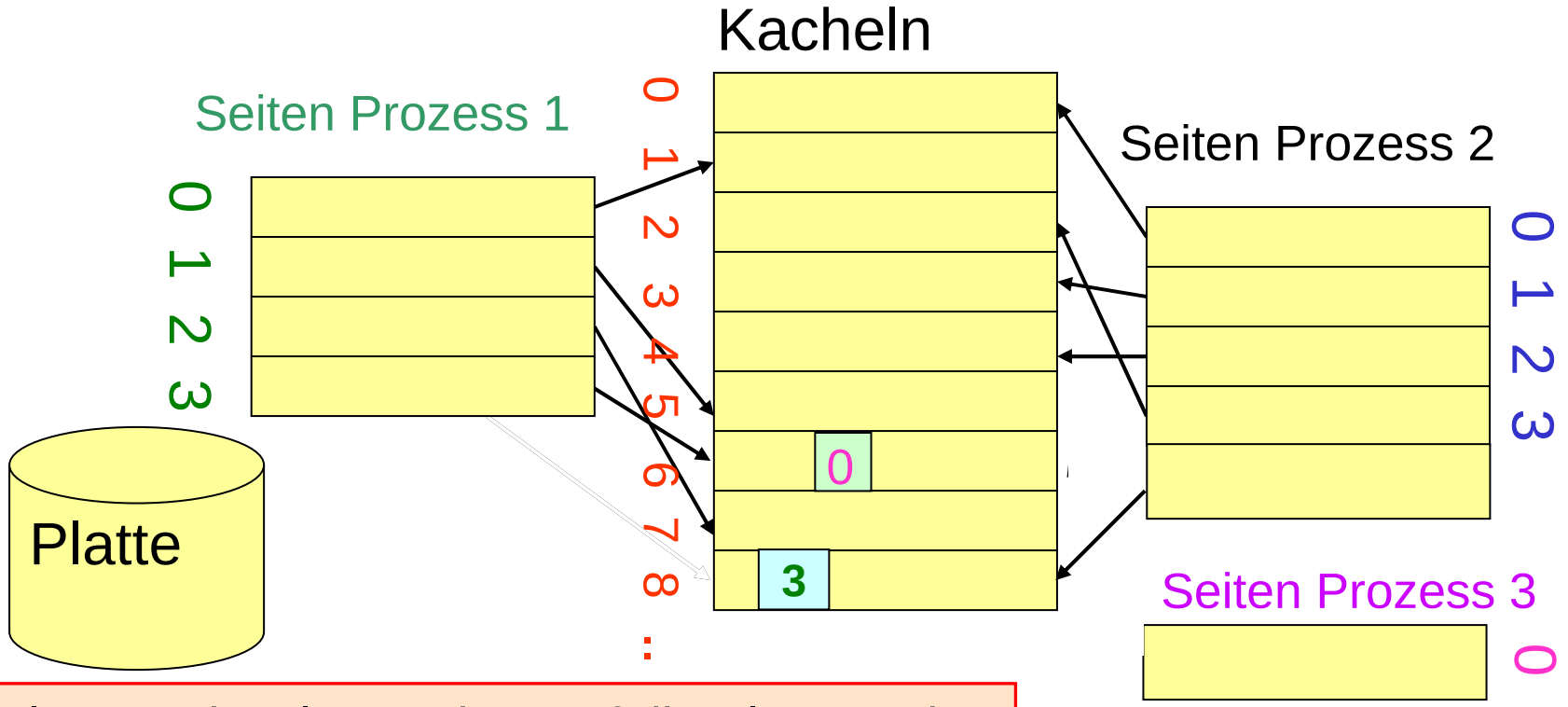


Beobachtungen

- *Paging* in der beschriebenen Form funktioniert ohne größeren Speicher (Plattenlaufwerk, *hard disc* (HD)) als weiteren Speicher.
- Wird bei Systemen ohne HD eingesetzt (in vielen eingebetteten Systemen).
- *Paging* bietet Form des **Speicherschutzes**: Prozesse können unterschiedliche Rechte zur Nutzung von Speicherbereichen haben.
- Speicherschutz mittels *paging* ist grob: Wunsch, durch verschiedene Prozesse auf Speicheradressen abgebildete Peripheriegräte mit unterschiedlichen Rechten zu nutzen. Häufig werden viele Peripheriegeräte auf dieselbe Seite abgebildet.

Demand paging

Wenn größerer und langsamerer Speicher (HD) verfügbar:
Einteilung des Speichers in Bereiche konstanter Länge + automatisches Auslagern selten benutzter Seiten sowie Einlagern von ausgelagerten Seiten, wenn auf die entsprechenden Adressen zugegriffen wird.

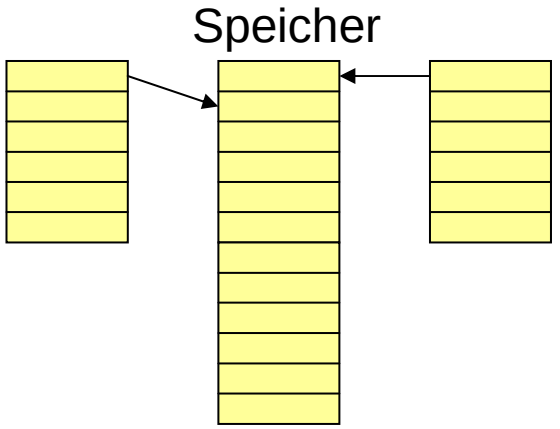


Auslagern einzelner Seiten auf die Platte und Rückschreiben auf beliebige Kachel möglich.

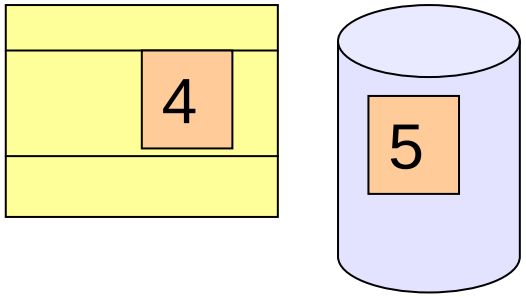
Paging

Paging ist mehrdeutig:

1. Einteilung des Speichers in Bereiche konstanter Länge, evtl. ohne Ein- und Auslagern (z.B., weil man Realzeit-Verhalten erzielen will).



2. Benutzung im Sinne von *demand paging.*

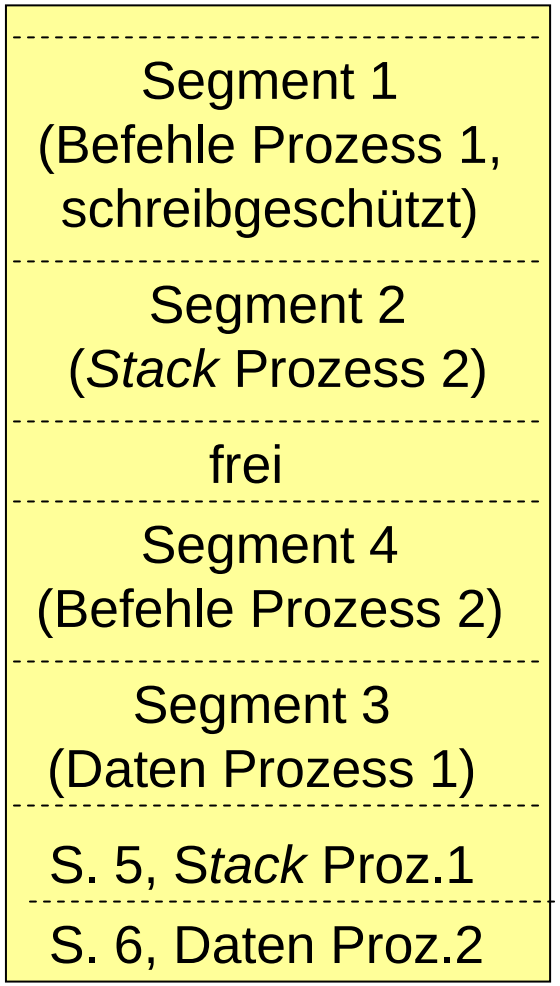


Definition 2 ist die Gebräuchlichere.

Segmentadressierung (1)

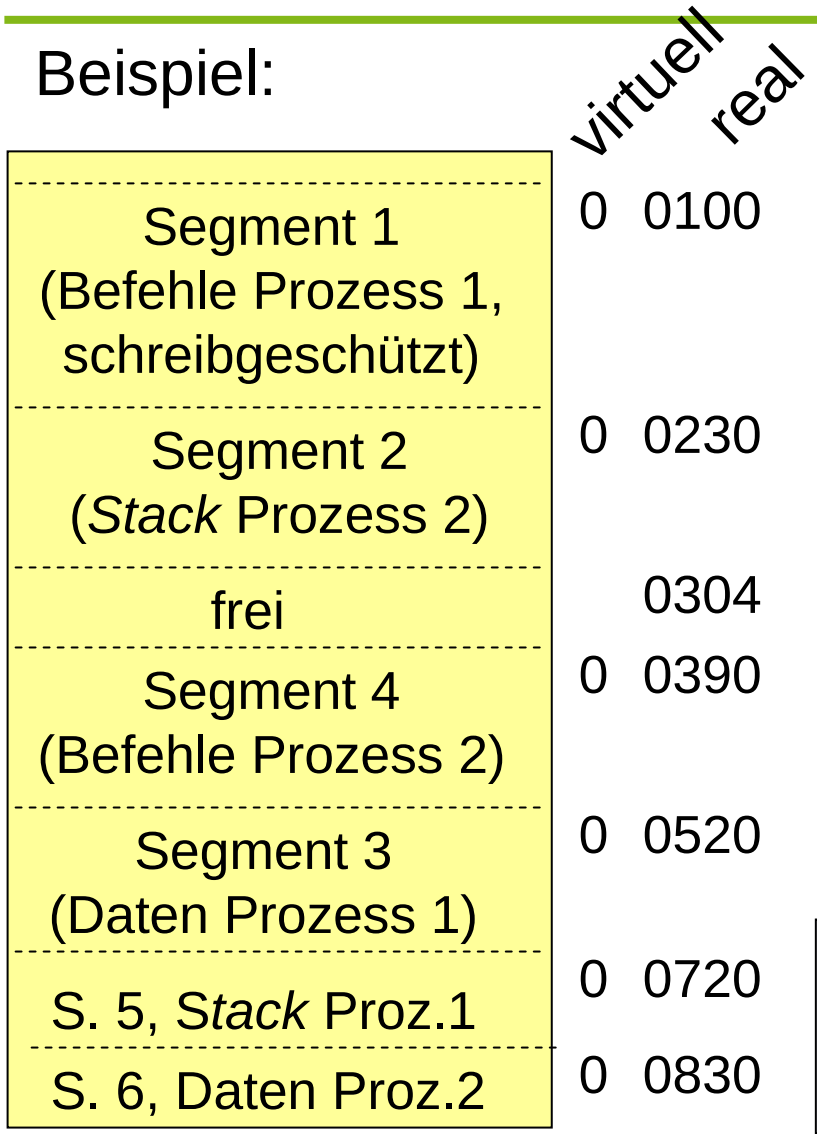
Zusammenhängenden Speicherbereichen (für Befehle, Daten, und den *stack*) im Prozessadressenraum werden **zusammenhängende Bereiche im realen Adressraum** zugeordnet.

Beispiel:



Segmentadressierung (2)

Beispiel:



Prozesse adressieren in jedem Segment ab Adresse 0;

Zu jedem Prozess gehört eine Tabelle der für ihre Segmente zu addierenden Adressen (**verdeckte Basen**).

Prozess 2		
Befehle	0390	
Daten	0830	
Stack	0230	

Länge

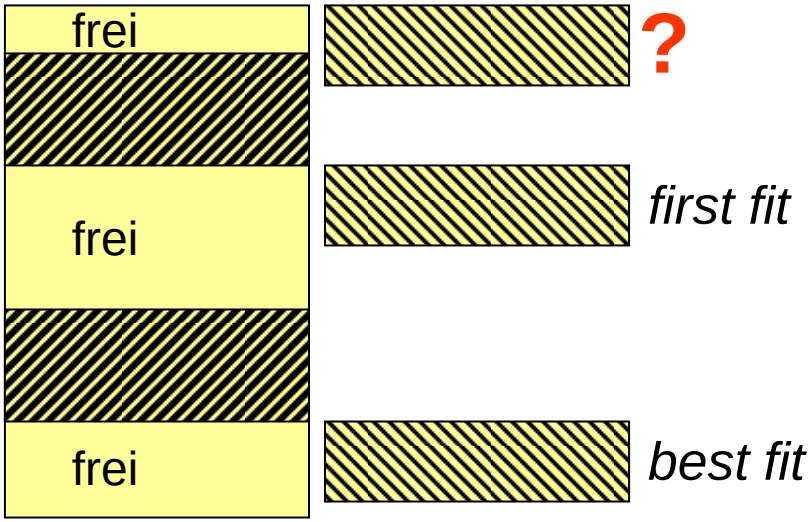
Erfordert Befehlssatz mit separaten Befehlen für *stack*-Zugriffe

Bei jedem Speicherzugriff wird die verdeckte Basis des jeweiligen Segments zur virtuellen Adresse addiert.

Eigenschaften der Segmentadressierung (1)

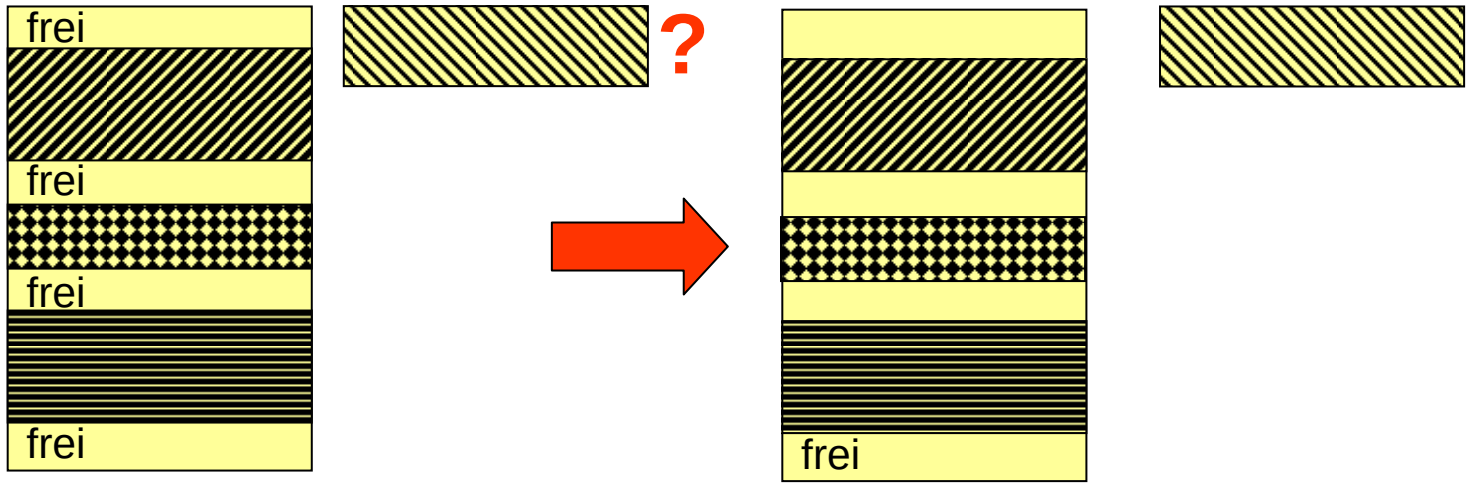
- Speicherschutz: Abfrage der Segmentlänge, R/W/E-Modi
- Beim Laden müssen keine Adressen angepasst werden, dies passiert stets zur Laufzeit.
- Verschieben von Segmenten ist möglich, da alle Adressen mittels Änderung der verdeckten Basis automatisch angepasst werden.
- Auslagern ganzer Segmente und Rückschreiben an beliebige, ausreichend große Lücke ist möglich.

Strategien zur Suche einer ausreichend großen, freien Lücke:

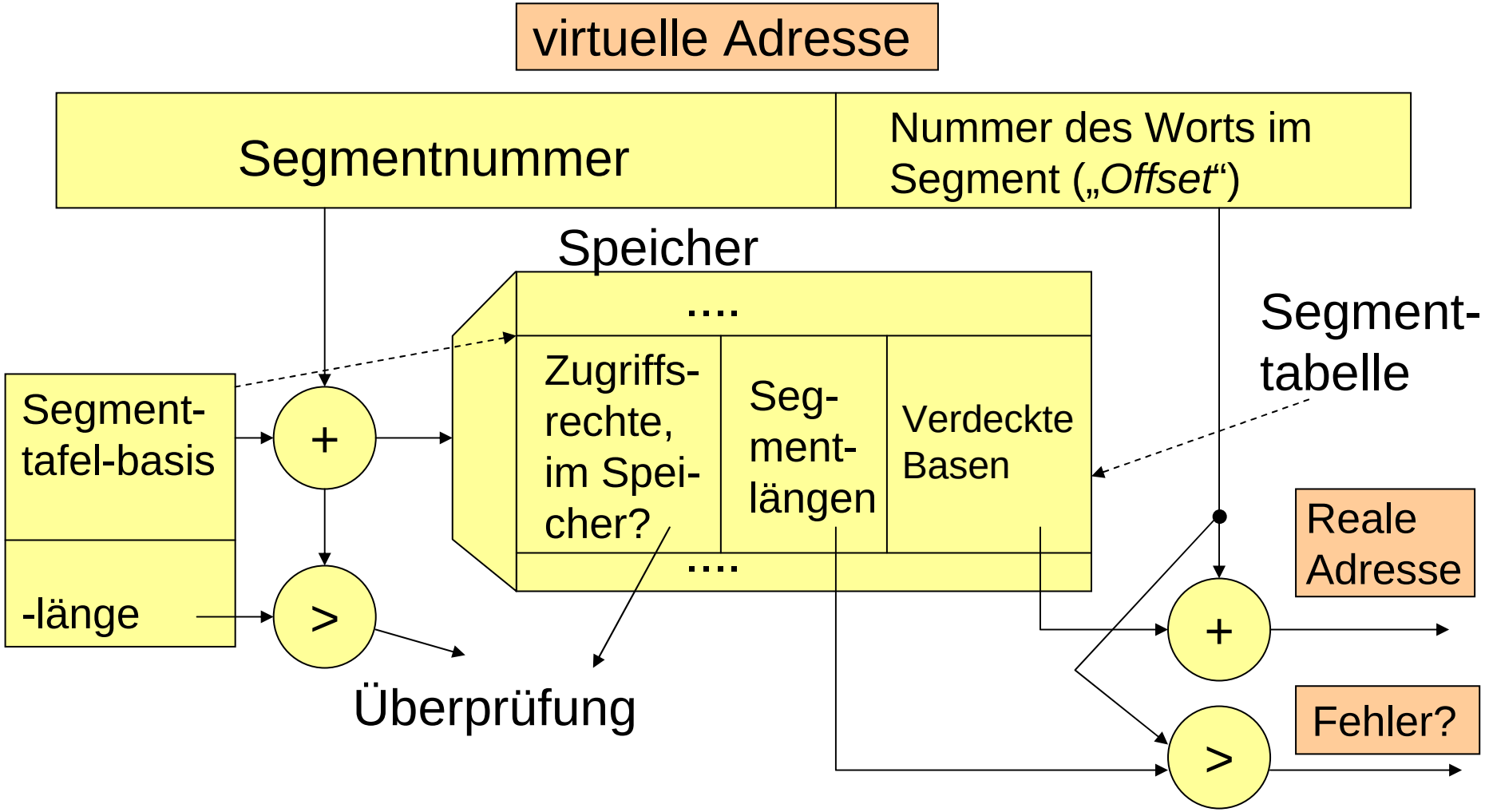


Eigenschaften der Segmentadressierung (2)

- Segmente können sich ausdehnen. Ist der anschließende Speicher belegt, so müssen sie umkopiert werden.
- Ein Teil des Speichers außerhalb der Segmente bleibt unbenutzt (externe Fragmentierung).
- Ggf. müssen Segmente verschoben werden, um ausreichend große freie Bereiche herzustellen.



Adressabbildung bei Segmentadressierung



Bei zusammenhängenden Segmentnummern effizient.

2 Formen der Segmentadressierung

- Segmentnummern in separaten Registern
  Beispiel: Intel Pentium
  Kein Problem mit dem *sharing:* Segmentregister müssen nur auf den gemeinsam genutzten Speicher „zeigen“.
  Wird von Microsoft nicht genutzt.
- Segmentnummern sind Teil der Adressen im Speicher
  Beispiel: PowerPC, IBM-Großrechner
  Beim *sharing* dasselbe Problem wie bei Seitenadressierung

Kombination von Segment- und Seitenadressierung

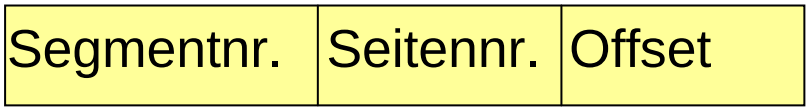
Segmentadressierung besitzt:

- Vorteile beim *Sharing*, wenn Segmentnummern in Registern gespeichert werden.
- Nachteile durch externe Fragmentierung

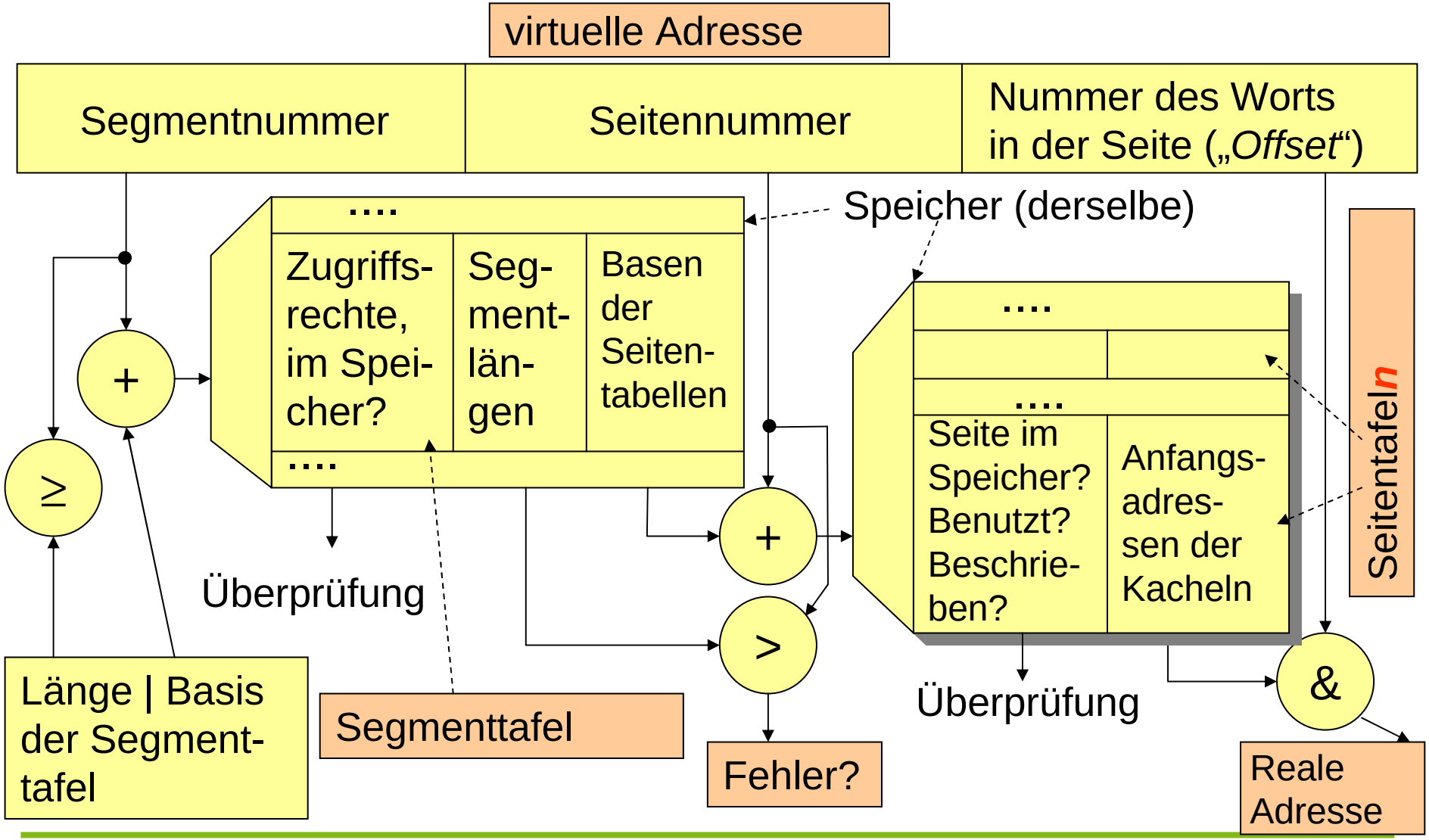
☞ **Kombination von Segment- und Seitenadressierung**:

- Jedes Segment enthält Seiten, diese wiederum Worte
- Fragmentierung reduziert sich auf interne Fragmentierung
- Jede Kachel kann jede Seite aufnehmen (kein Kopieren)
- Vorteile bei *shared libraries*

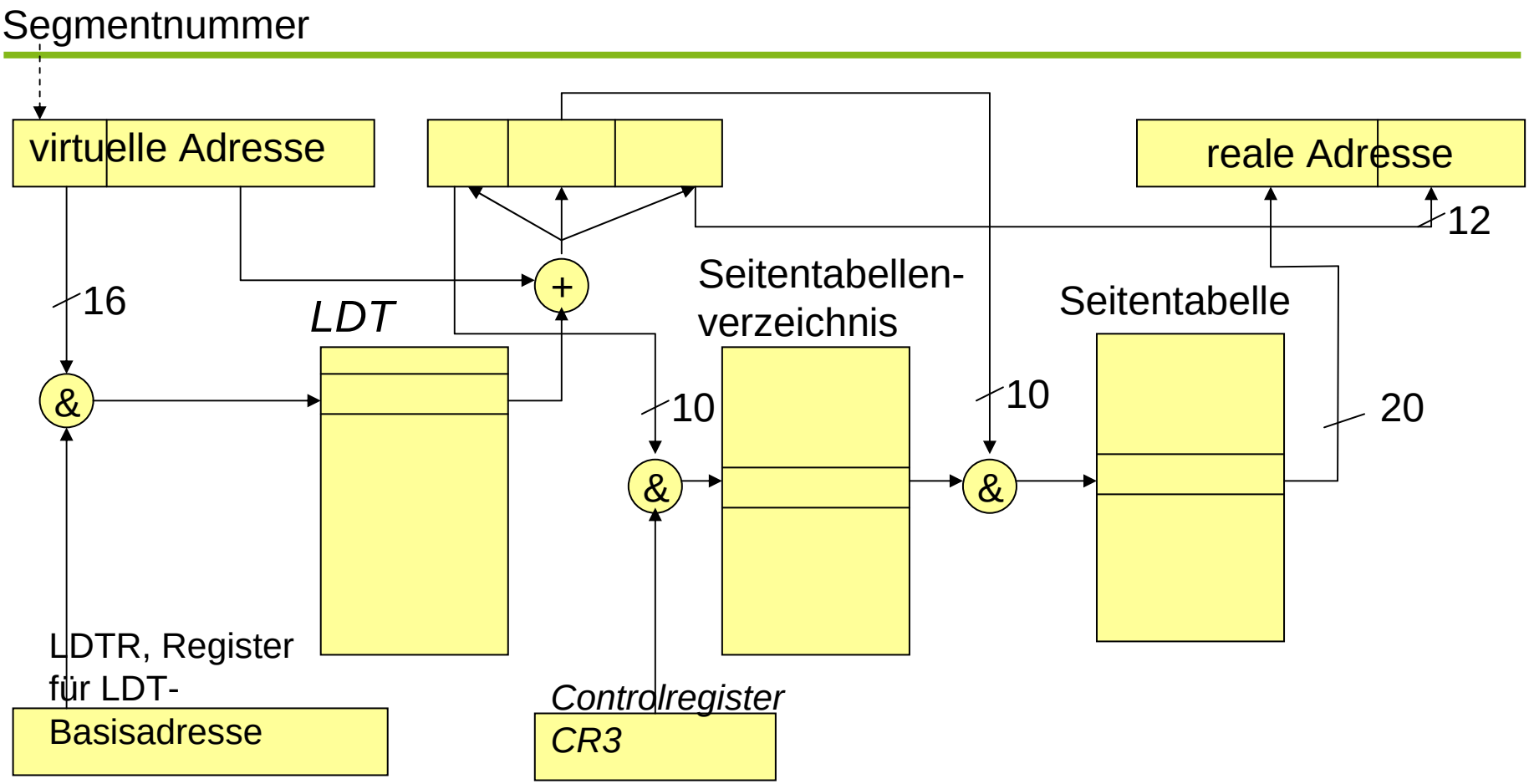
virtuelle Adressen



Segmentadressierung mit Seitenadressierung



Adressabbildung beim Intel 386



Die *Local Descriptor Table LDT* benötigt Einträge für den Bereich möglicher Segmentnummern; Tabellen für unbenutzte Seiten sind nicht erforderlich. Diese haben selbst wieder die Größe einer Seite. Adressübersetzungstabellen können teilweise auf die Platte ausgelagert werden

Zusammenfassung

- Viele Rechensysteme unterstützen *Multitasking*
- Reale und virtuelle Adressen, *memory management*
- Formen der Speicherverwaltung
  - Identität
  - Seitenadressierung
    - Sharing
  - Segmentadressierung
  - Segmentadressierung mit Seitenadressierung

2.4.3 *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** Hardware-speicher, welche häufig die benötigten Tabelleneinträge enthalten.

Diese heißen ***translation look aside buffer (TLB)*** oder auch ***address translation memory (ATM).***

Die 3 Organisationsformen von TLBs

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

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

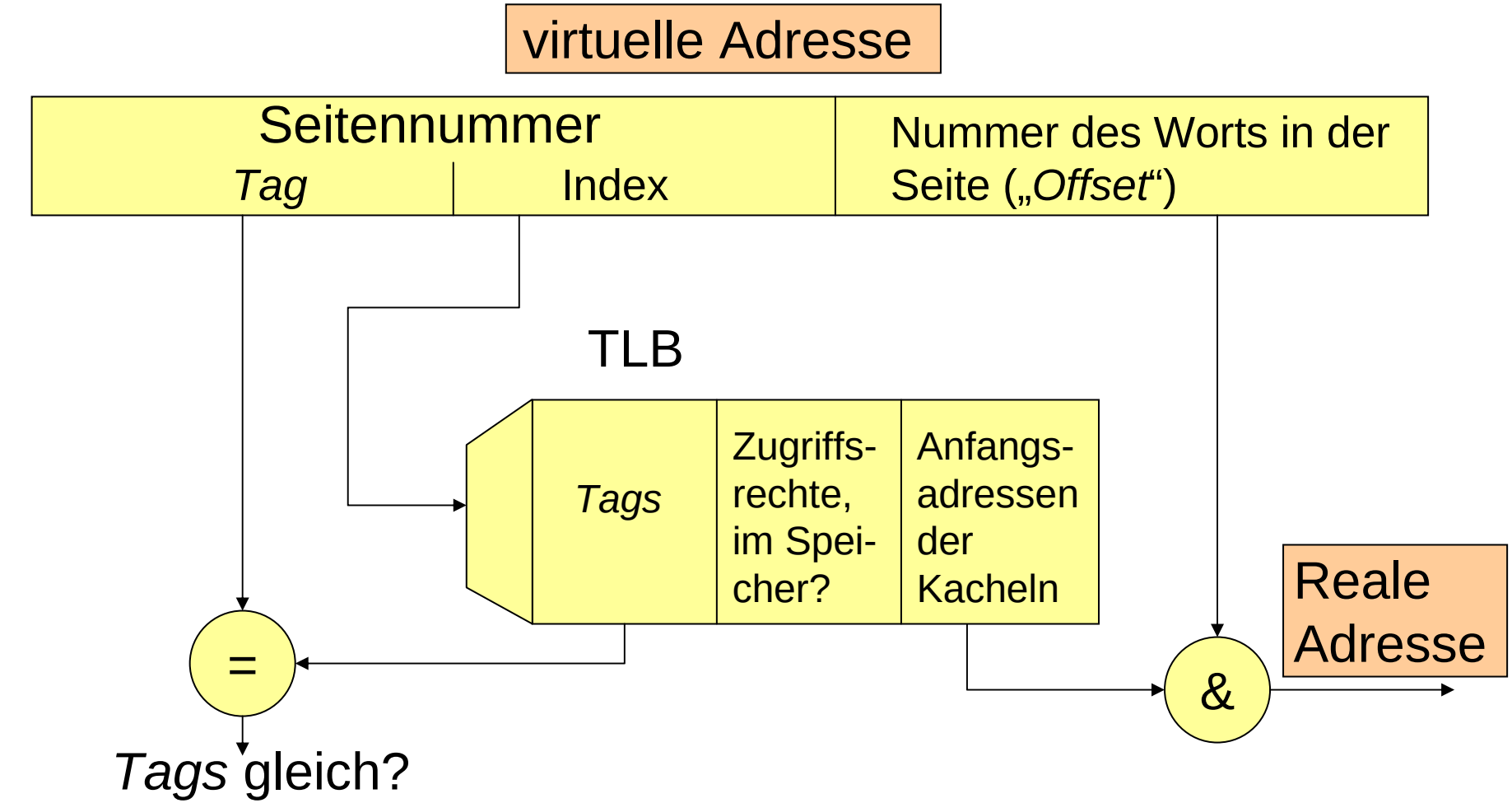
Der Eintrag aus der vollständigen Tabelle verdrängt dann einen Eintrag aus dem TLB nach einem noch zu bestimmenden Algorithmus.

Drei Organisationsformen von TLBs:

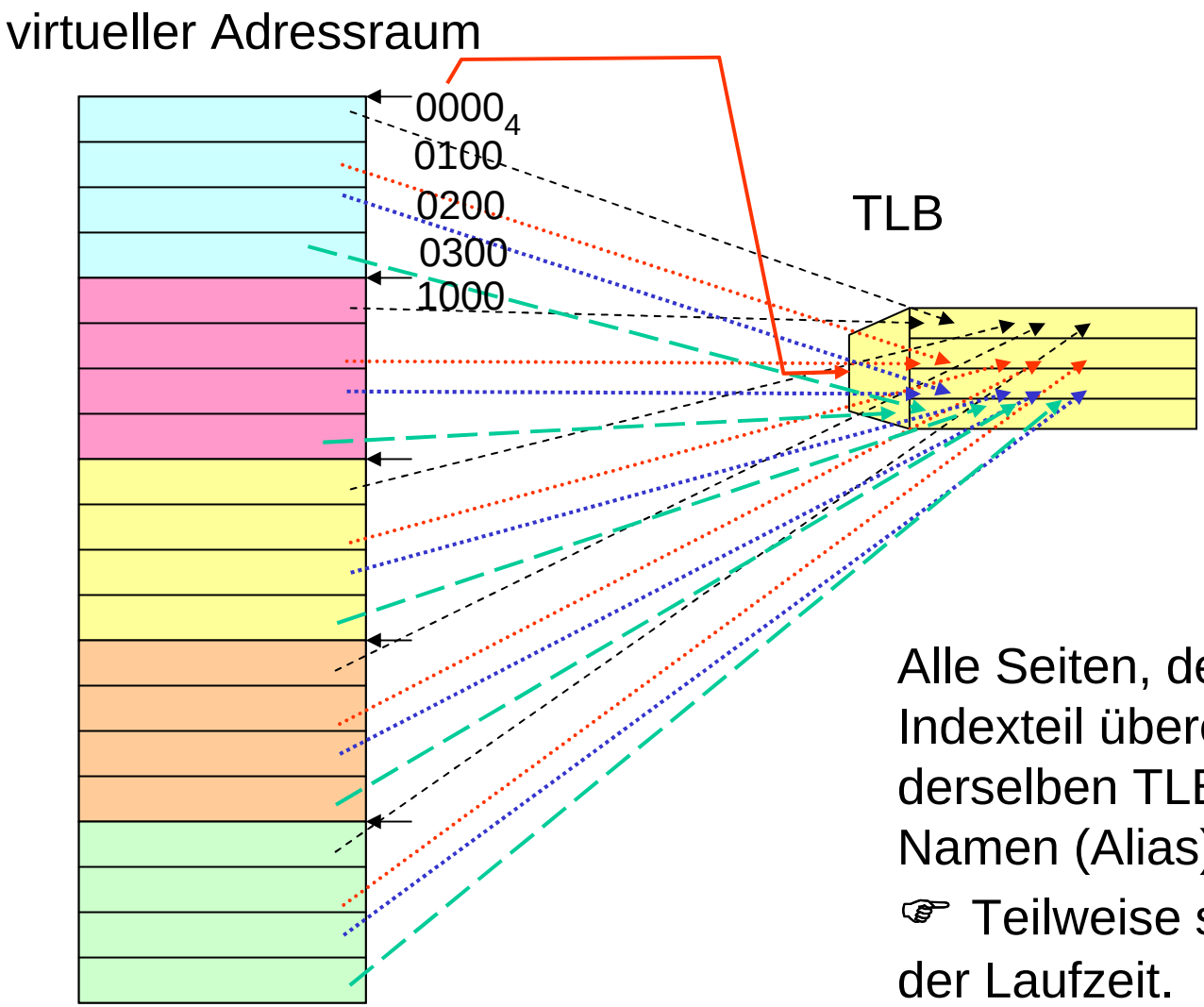
1. ***direct mapping***,
2. **Mengen-assoziative Abbildung** *(set associative mapping)*,
3. und **Assoziativspeicher *(associative mapping)***.

Direct Mapping

Die Seitennummer oder ein Teil davon adressiert den TLB.



Aliasing bei *direct mapping*



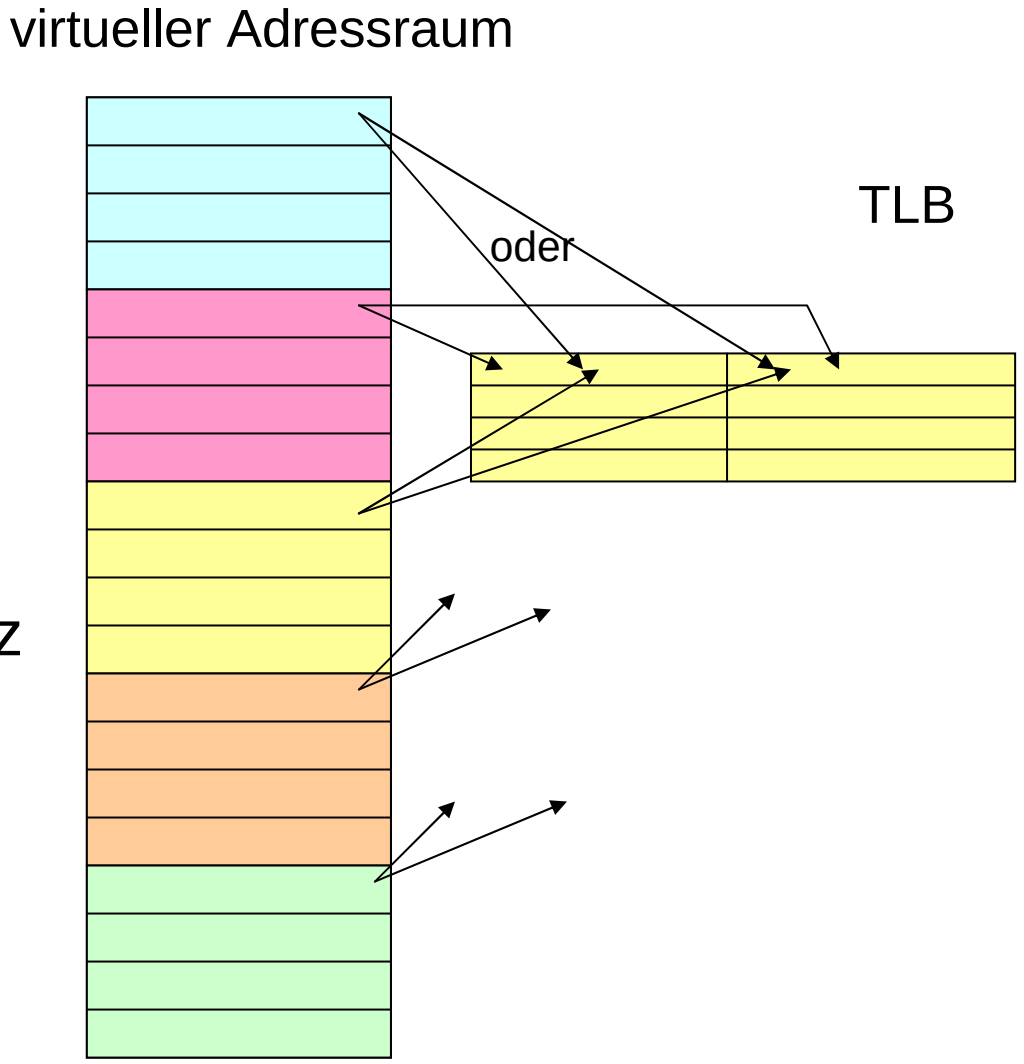
Alle Seiten, deren Adressen im Indexteil übereinstimmen, werden derselben TLB-Zelle (demselben Namen (Alias)) zugeordnet.

☞ Teilweise starkes Anwachsen der Laufzeit.

Mengen-assoziative Abbildung
- Prinzip -

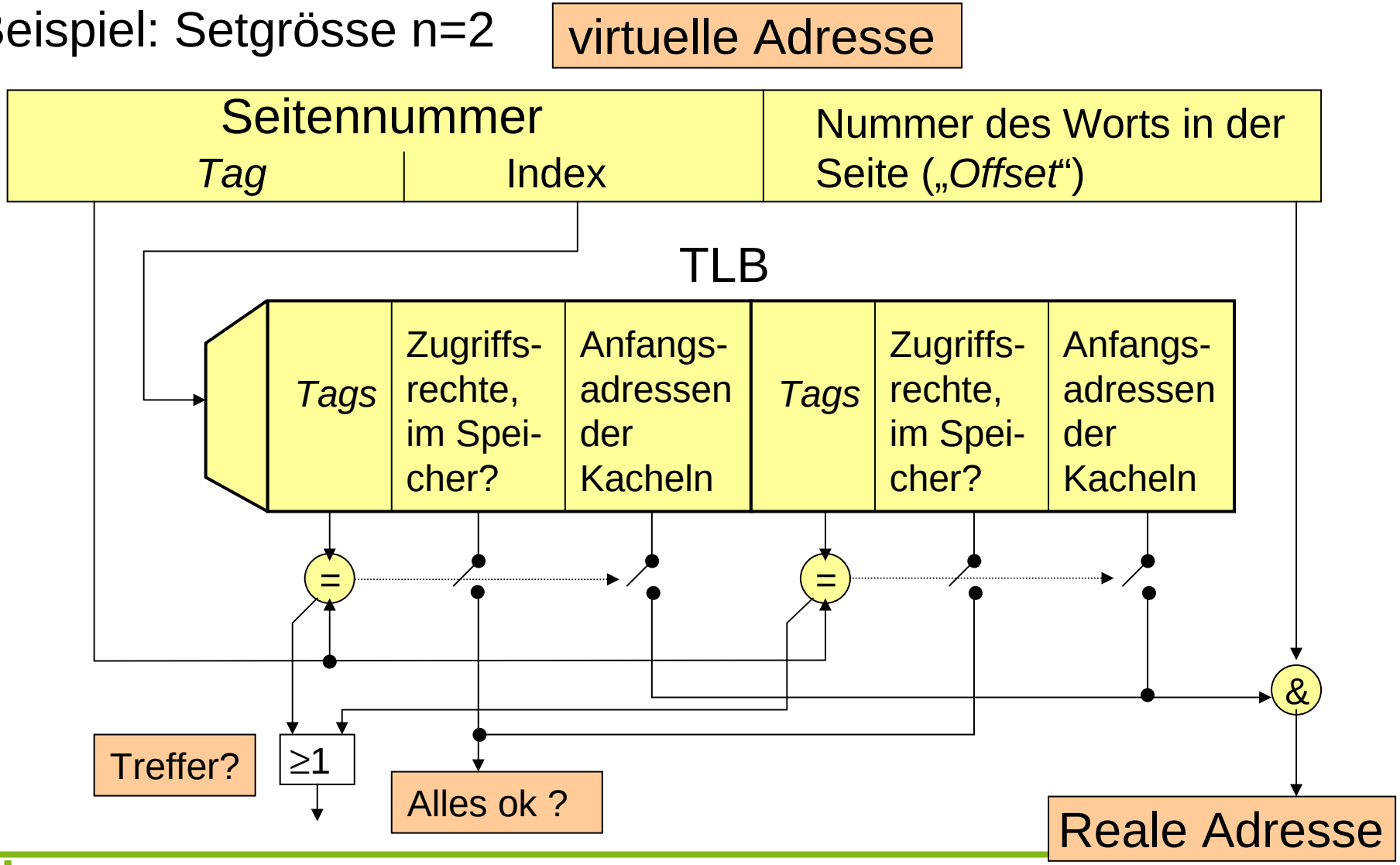
Zu jedem Index-Wert gibt es *n* zugeordnete Plätze, mit *n*>1.

Sofern über die vollständigen Tabellen im Hauptspeicher abgebildet werden muss, wird ein Platz im TLB überschrieben, z.B. der am Längsten nicht benutzte (*least recently used* (LRU)-Strategie)



Mengen-assoziative Abbildung
- Realisierung -

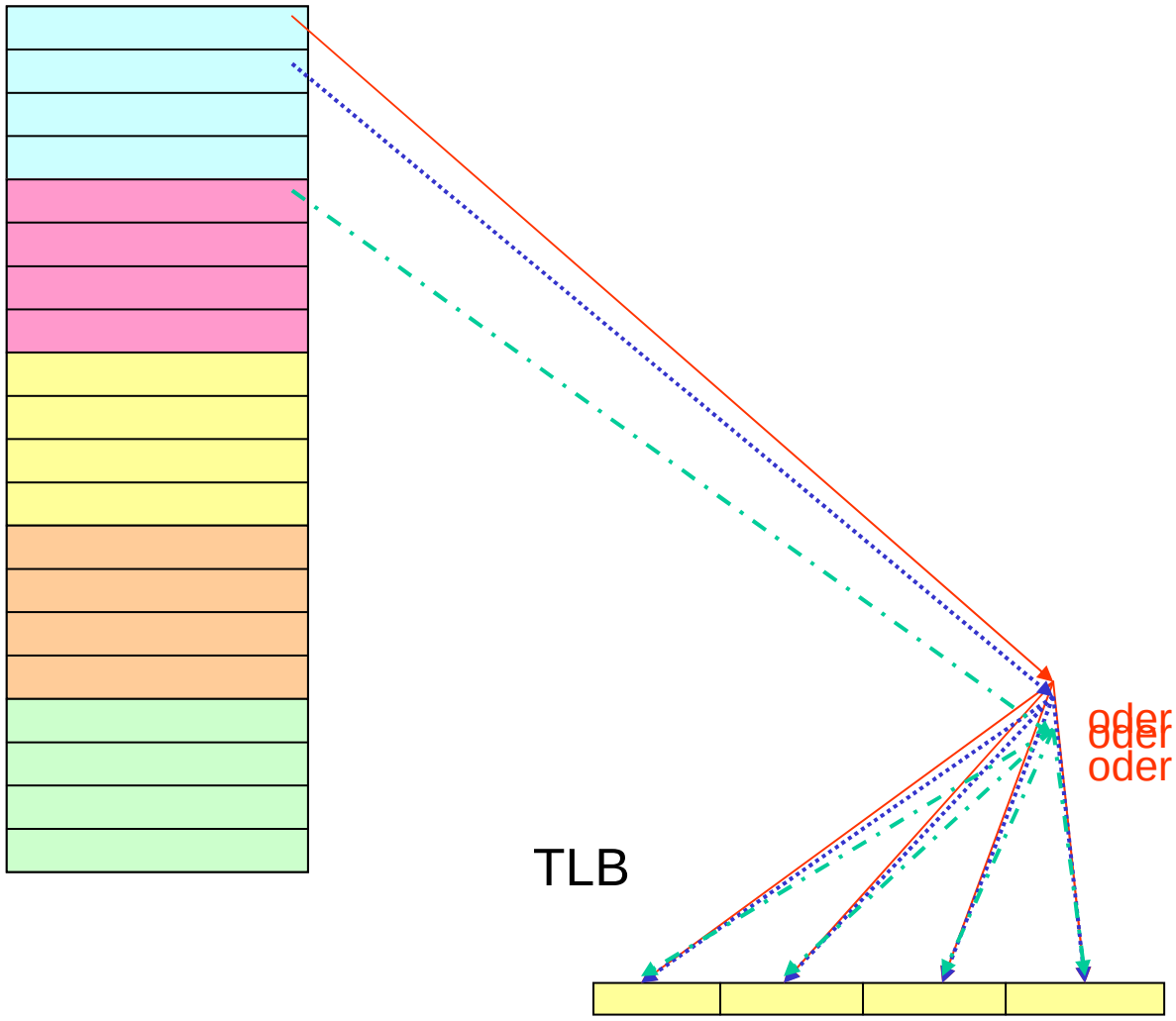
Beispiel: Setgrösse n=2



Assoziativspeicher, *associative mapping* - Prinzip -

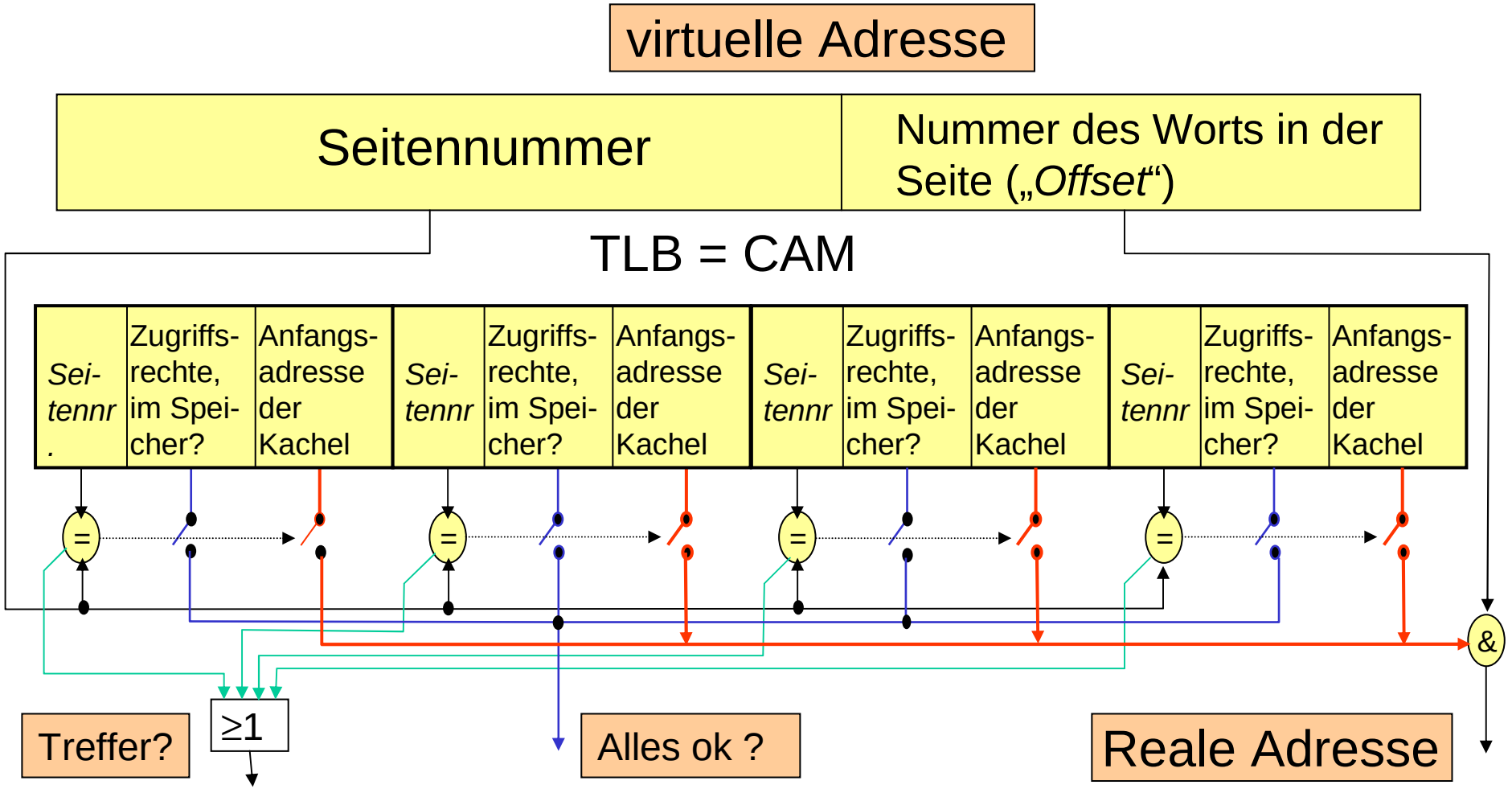
Der Fall *associative mapping* entsteht aus dem *set associative mapping* für „Anzahl der Bits für den Index-Teil $\rightarrow$ 0“.

Jeder Platz des TLB kann die Verwaltungs-Information für jede beliebe Seite enthalten (kein *aliasing*).

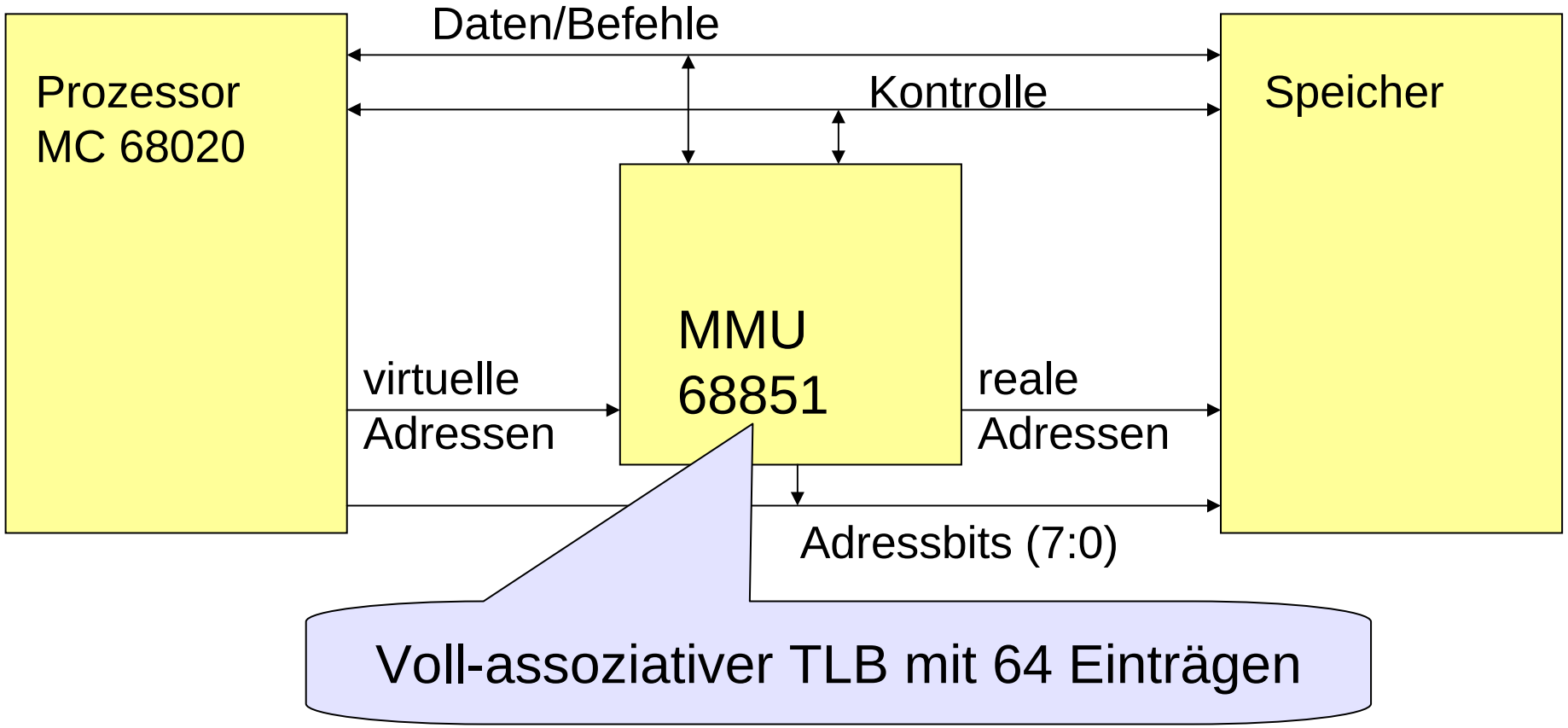


Hardware-mäßige Realisierung mit ***content-adressable memory (CAM).***

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 den Hauptspeicher.

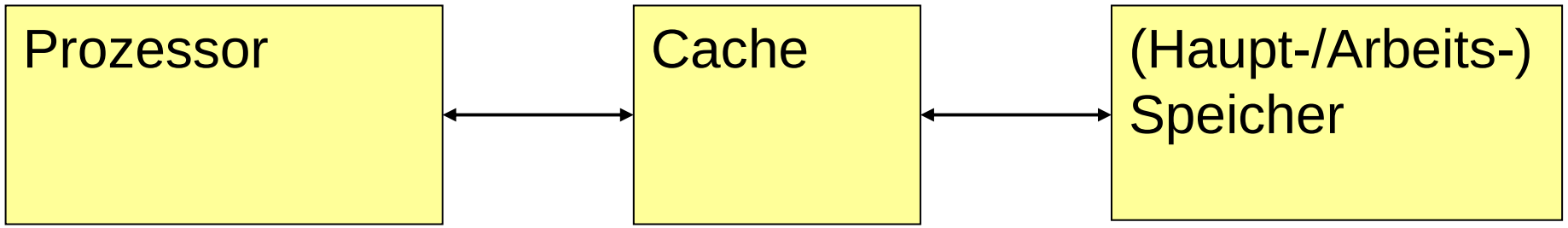
3 Organisationsformen:

1. ***Direct mapping***
2. **Mengen-assoziative Abbildung** *(set associative mapping), und*
3. **Assoziativspeicher *(associative mapping)*.**

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

2.4.4 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.



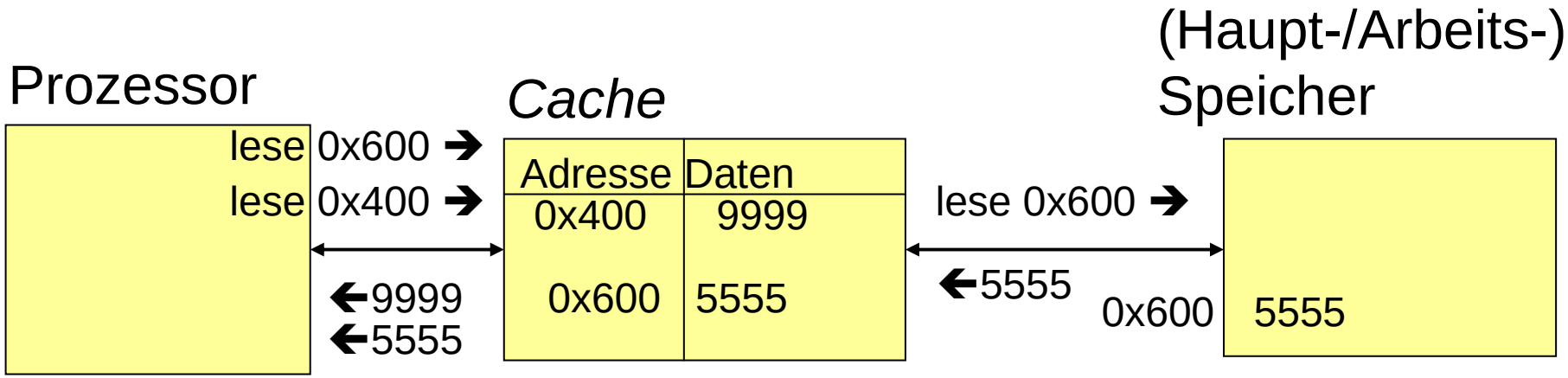
Ursprung*: cacher* (frz.): verstecken („versteckter Speicher“).

Ablauf

Organisation von Caches (im engeren Sinn):

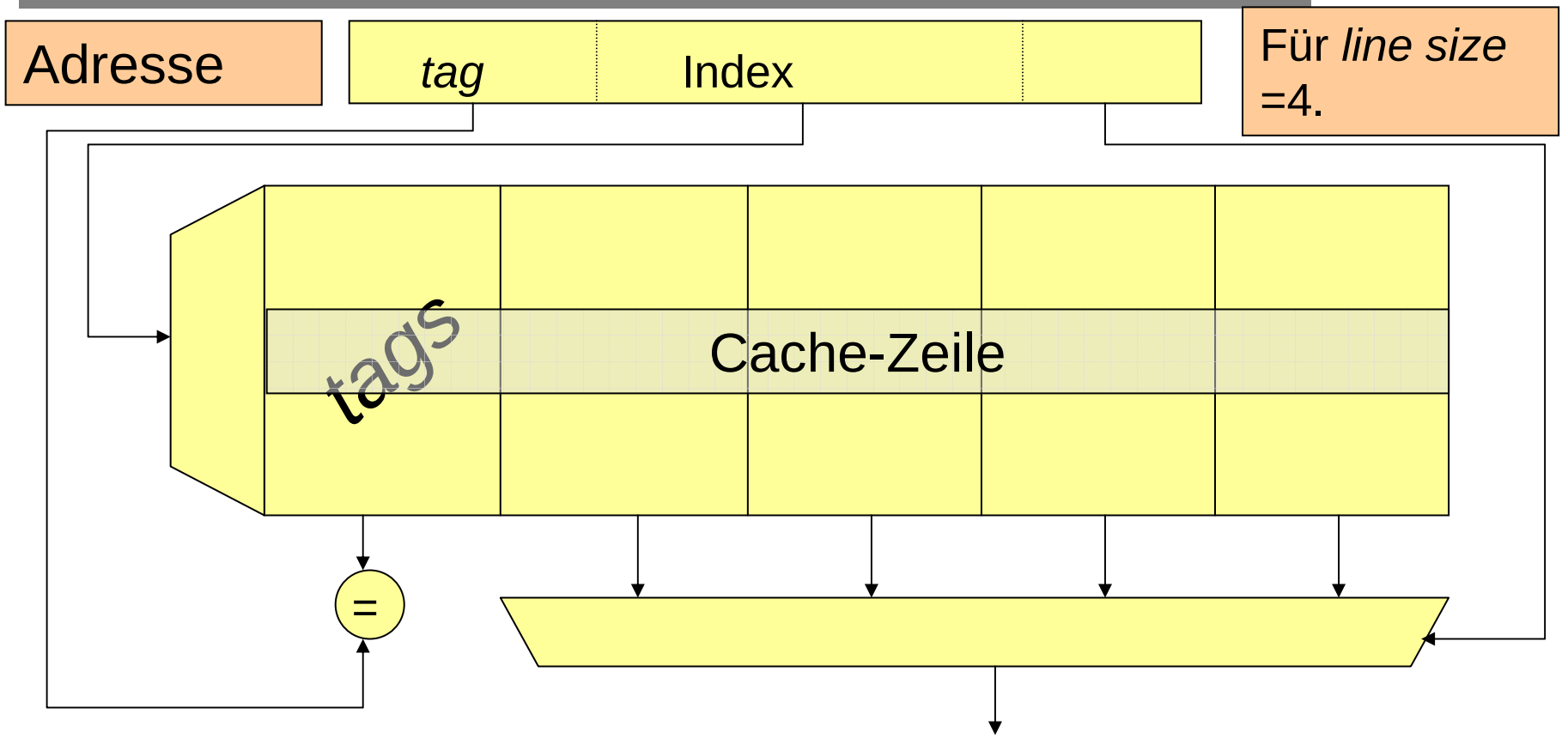
Prüfung anhand der (virtuellen oder realen) Adresse, ob benötigte Daten im *Cache* vorhanden sind („Treffer“; *cache hit*).

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



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

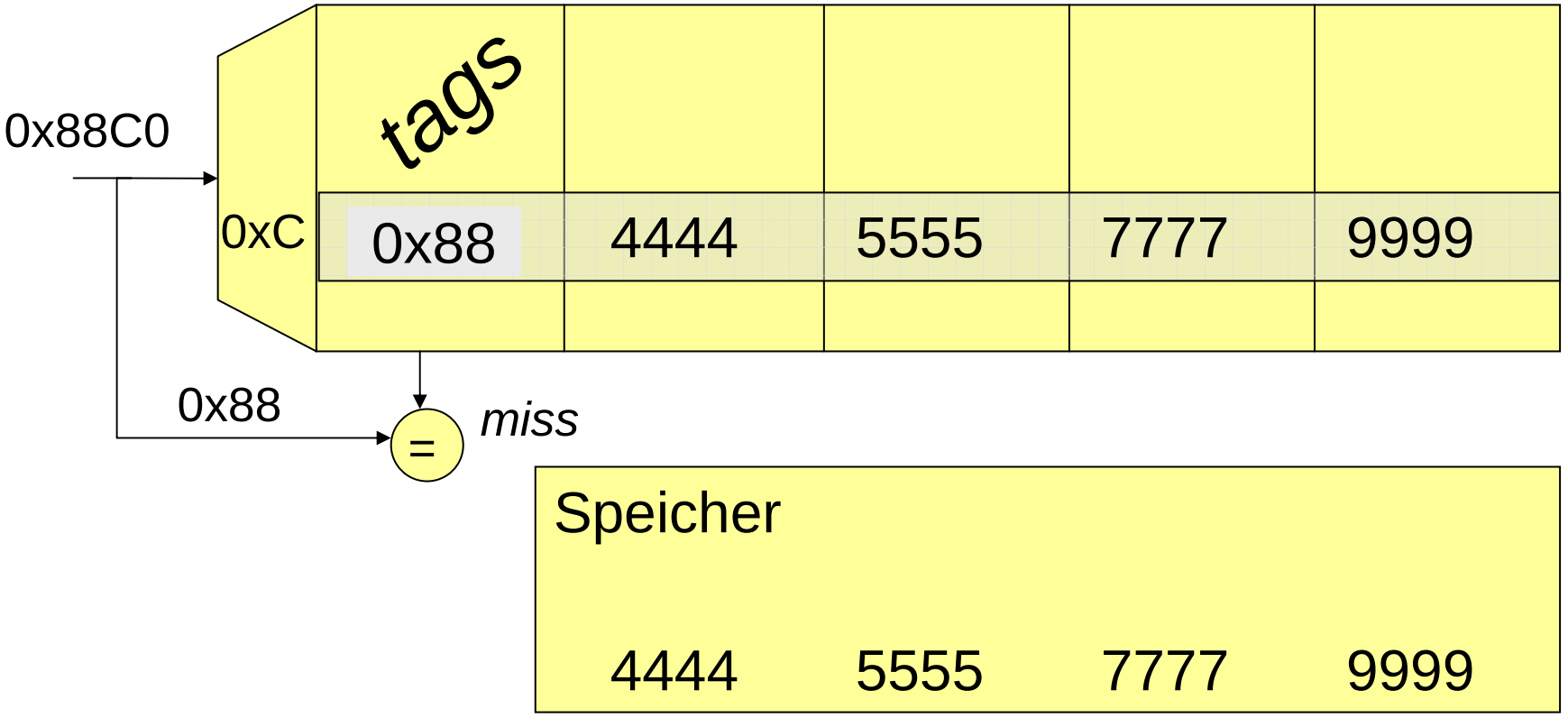
Such-Einheit im Cache: ***Cache*-Zeile** (***cache line***).



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

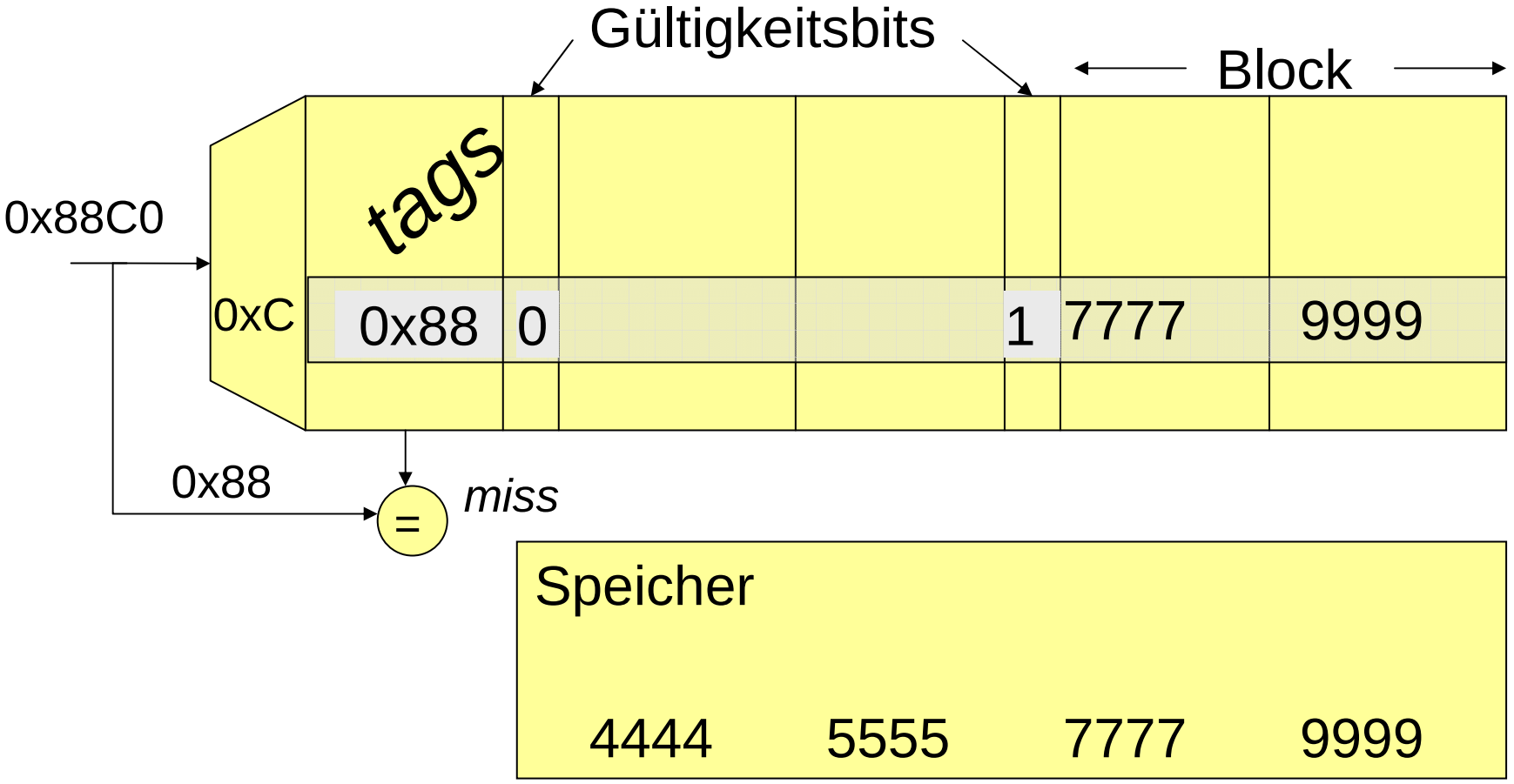
Cache-Blöcke (*cache blocks*) - 1 -

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 so entworfen, dass der Speicher durch das zusätzliche Lesen nicht langsamer wird:
Methoden dazu:

1. Schnelles Lesen aufeinanderfolgender Speicherzellen (*nibble mode, block/page mode* der Speicher)
2. *Interleaving* (mehrere Speicher-ICs mit überlappenden Zugriffen)
3. Block-Mode beim Buszugriff, Page-Mode beim Speicherzugriff
4. Fließbandzugriff auf den Speicher (EDO-RAM, SDRAM)
5. breite Speicher, die mehrere Worte parallel übertragen können.

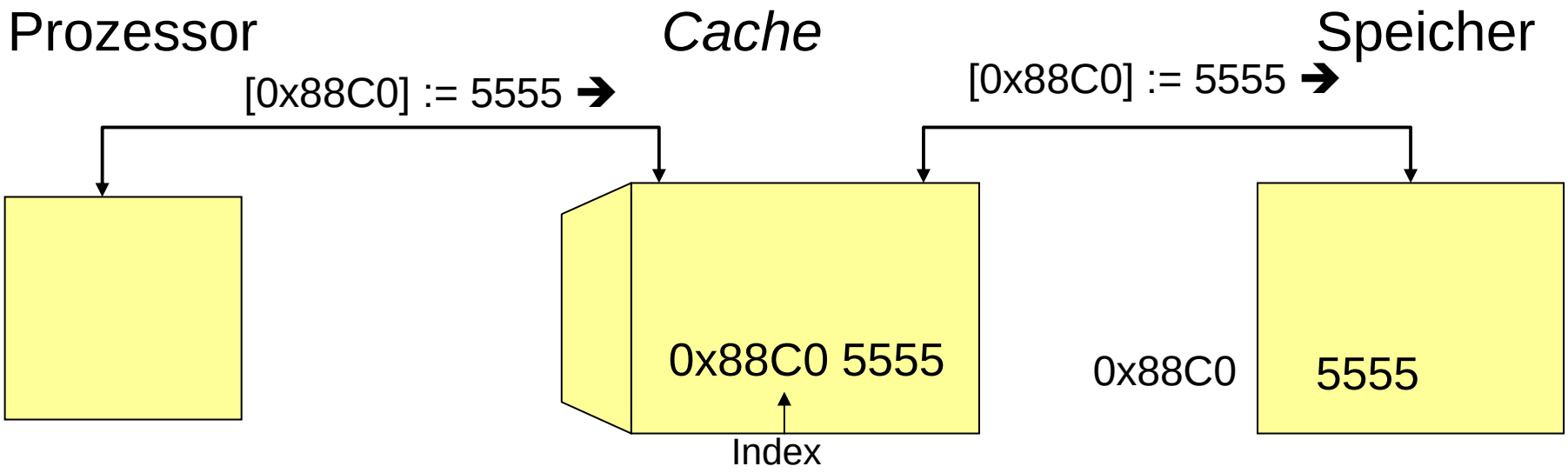
Organisationsformen von Caches

- ***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 eines Caches kommt diese Organisationsform kaum in Frage.

Schreibverfahren

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

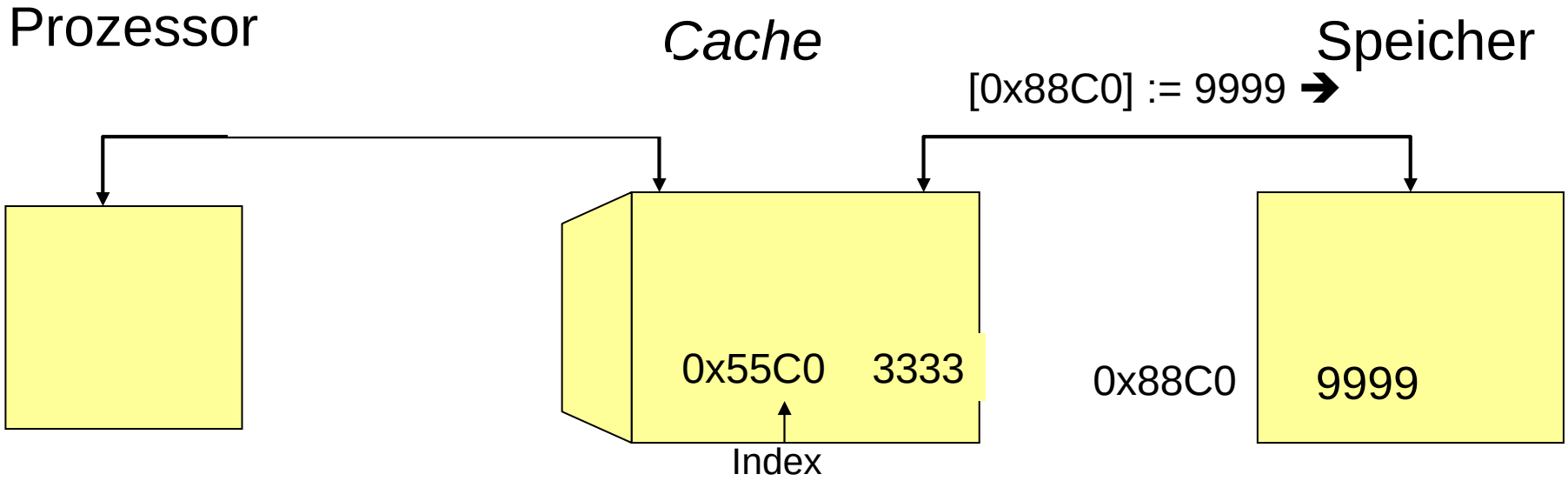
1. ***Write-Through*** (durchschreiben)
   Jeder Schreibvorgang in den *Cache* führt zu einer unmittelbaren Aktualisierung des (Haupt-) Speichers



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

Schreibverfahren

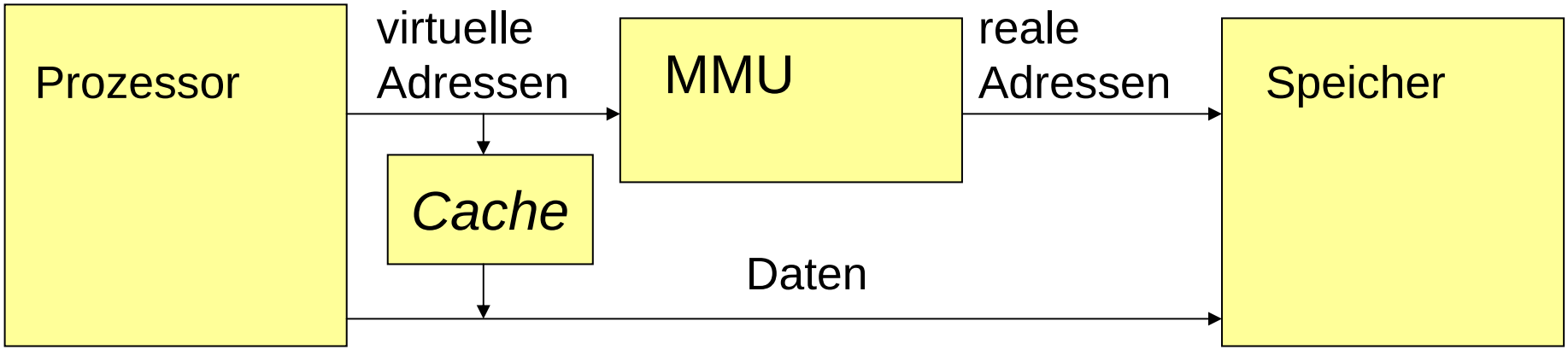
2. *Copy-Back, conflicting use write back:*
Rückschreiben erfolgt erst, wenn die Cache-Zeile überschrieben wird.



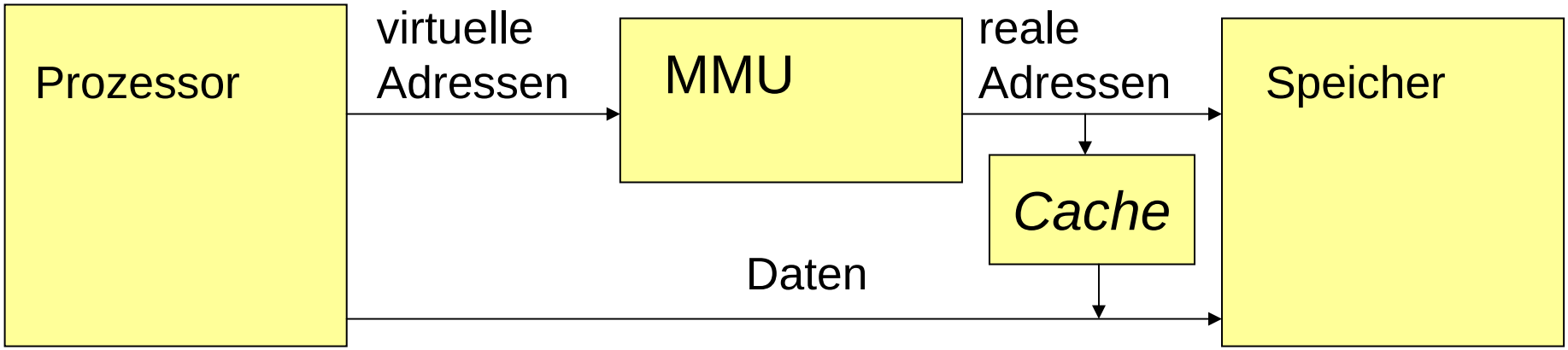
Funktioniert auch bei großen Geschwindigkeitsunterschieden zwischen *Cache* und Speicher. Vorkehrungen erforderlich, damit keine veralteten Werte aus dem Speicher kopiert werden.

Virtuelle und reale *Caches*

Virtuelle *Caches* (schnell)

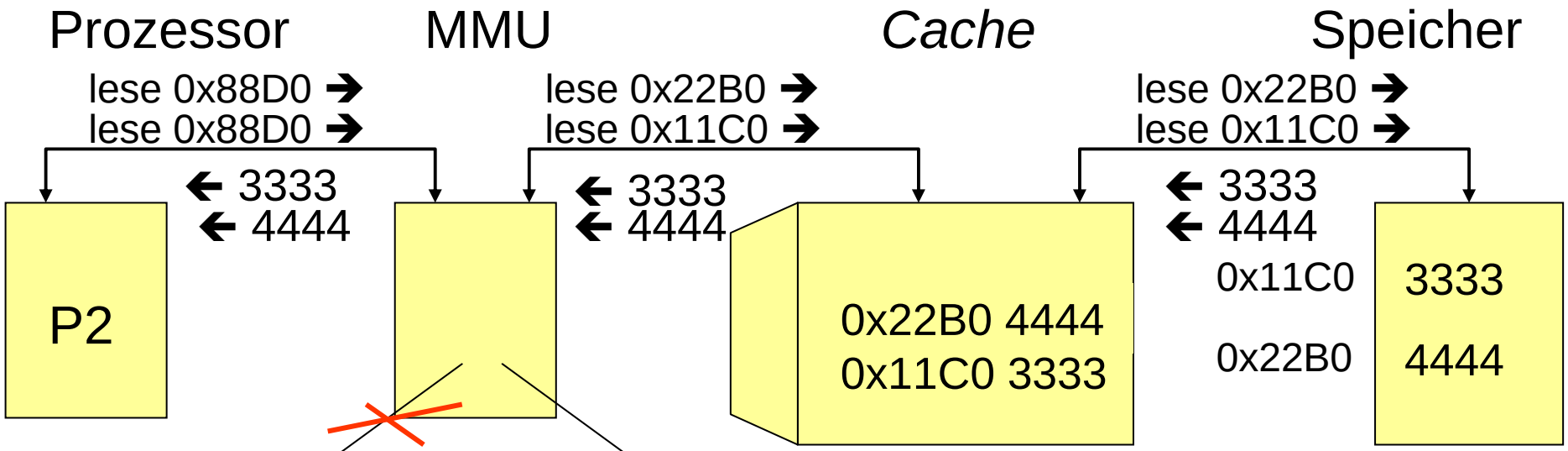


Reale *Caches* (langsamer):



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.

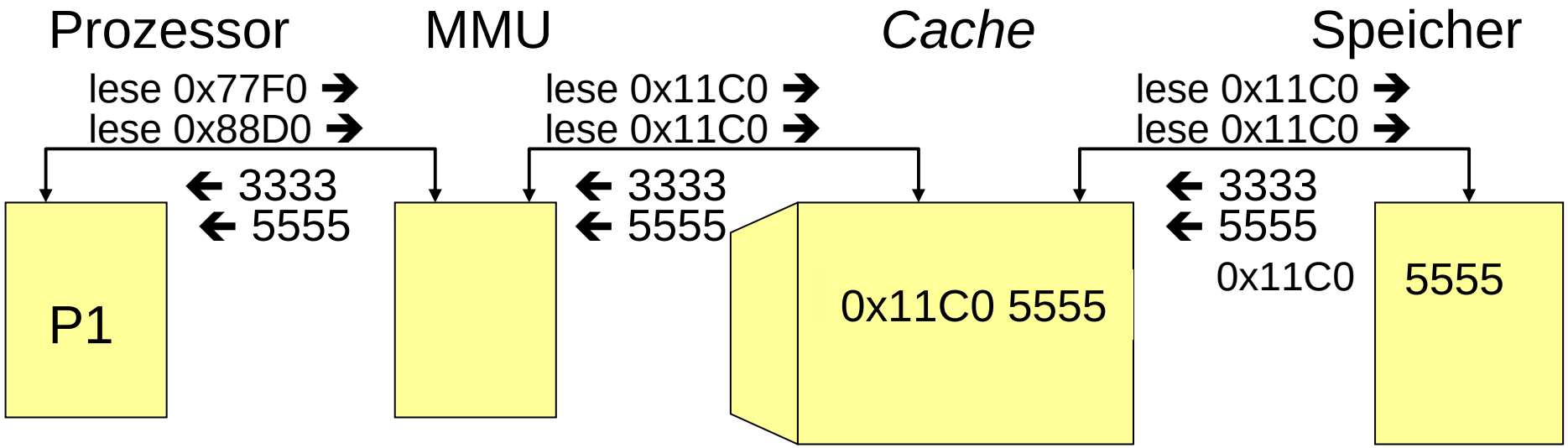
Prozess P2	
Seite	**Kachel**
0x88D.	0x22B.

☞ 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.



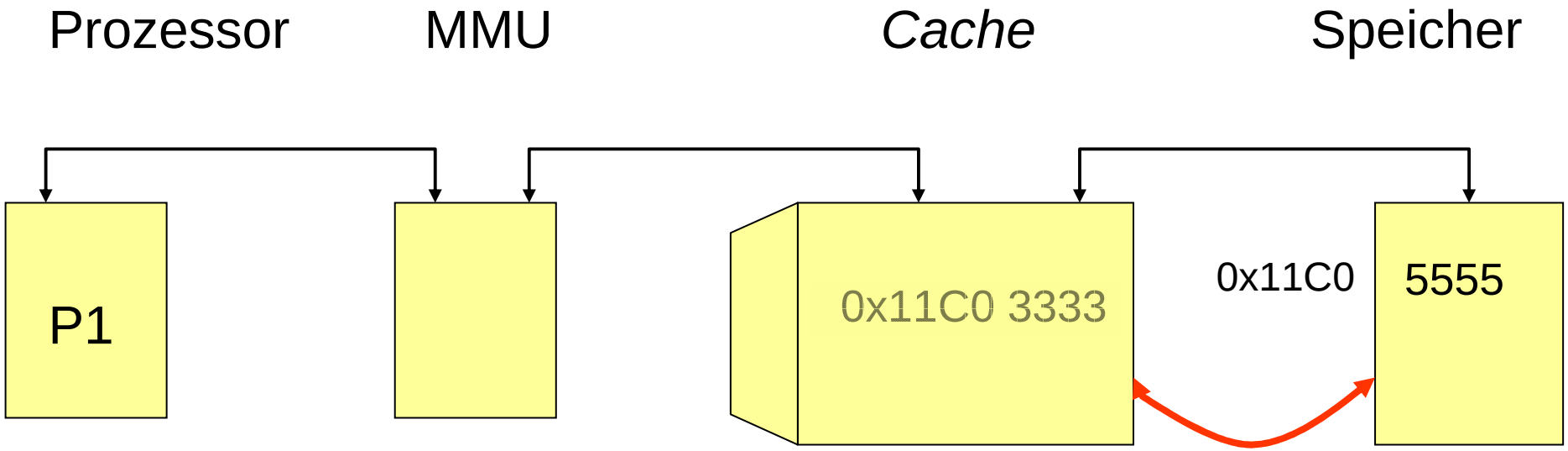
Vor Aus/Einlagern	
Seite	**Kachel**
0x88D.	0x11C.

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

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

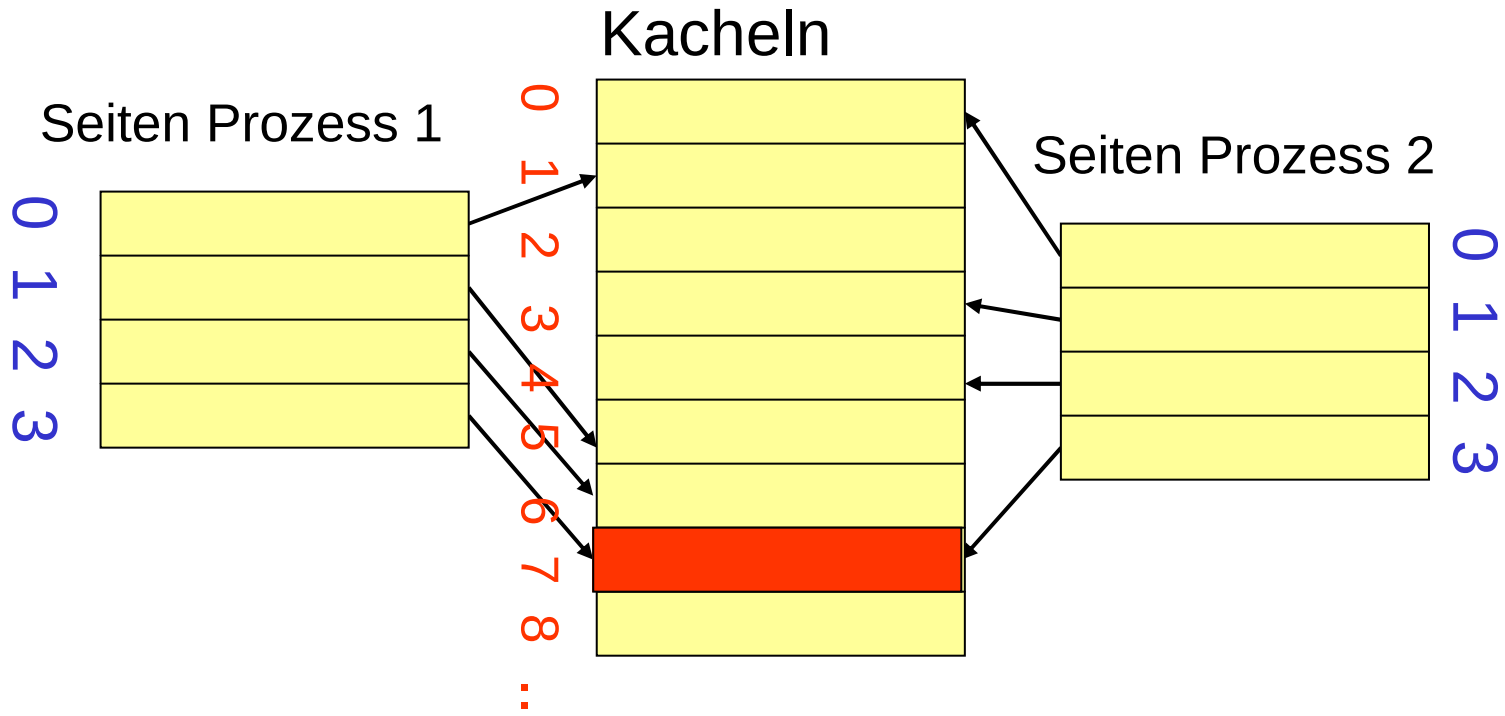
Bus snooping (Bus-Lauschen)

☞ 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.

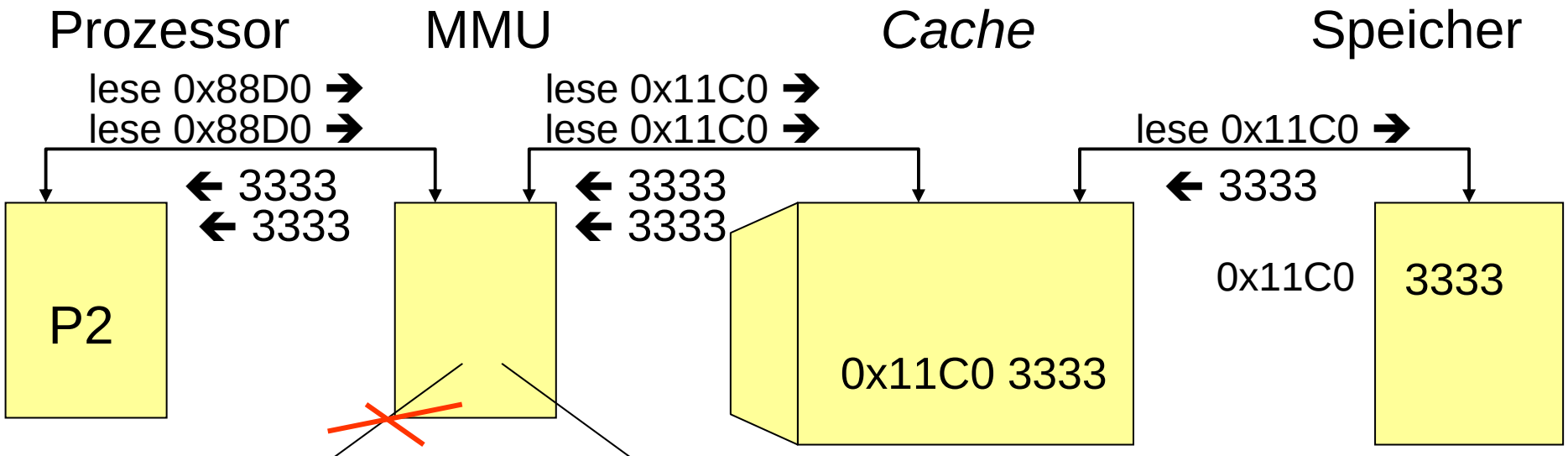
Gemeinsam genutzte Seiten (*sharing*)



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

Realer *Cache* bei gemeinsam genutzter Seite

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



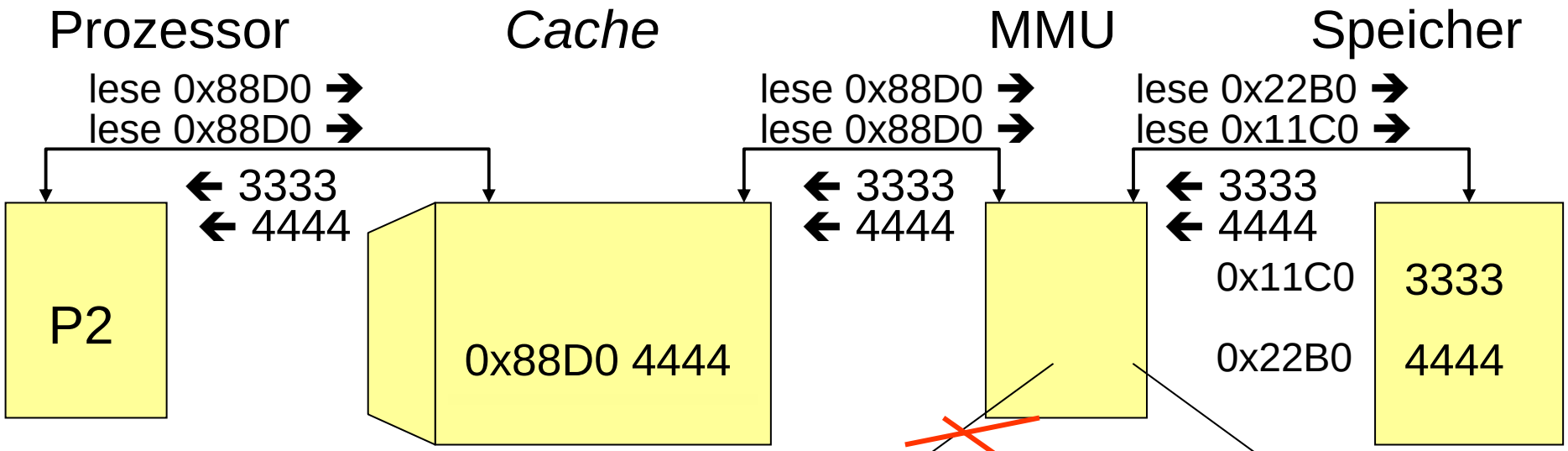
Prozess P1	
Seite	**Kachel**
0x88D.	0x11C.

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.

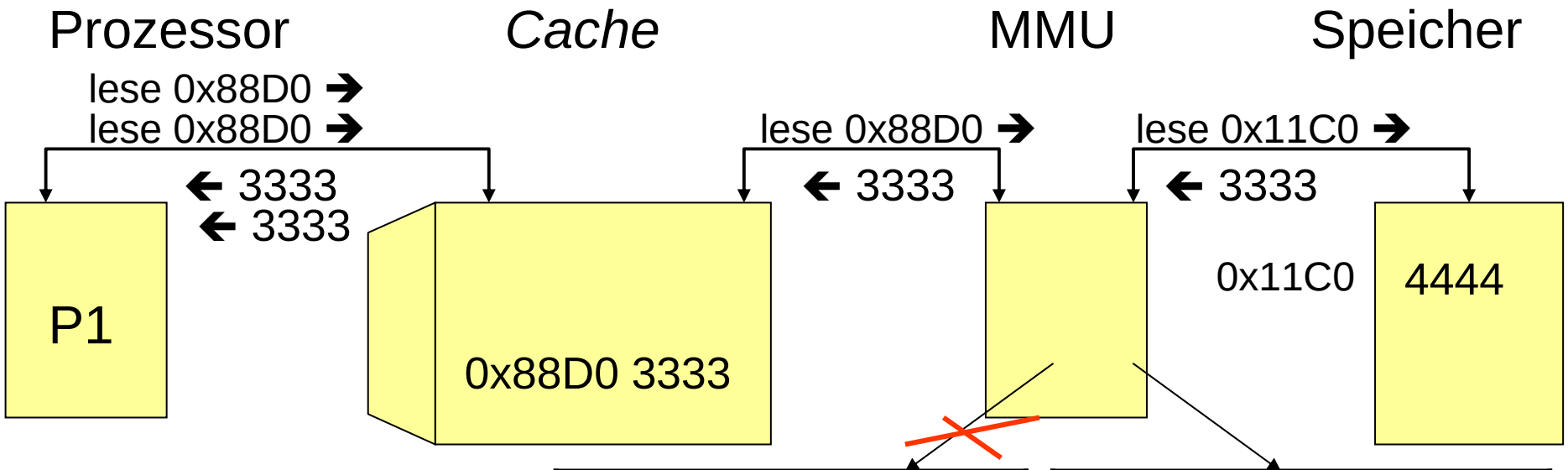
Prozess P1	
Seite	**Kachel**
0x88D.	0x11C.

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.



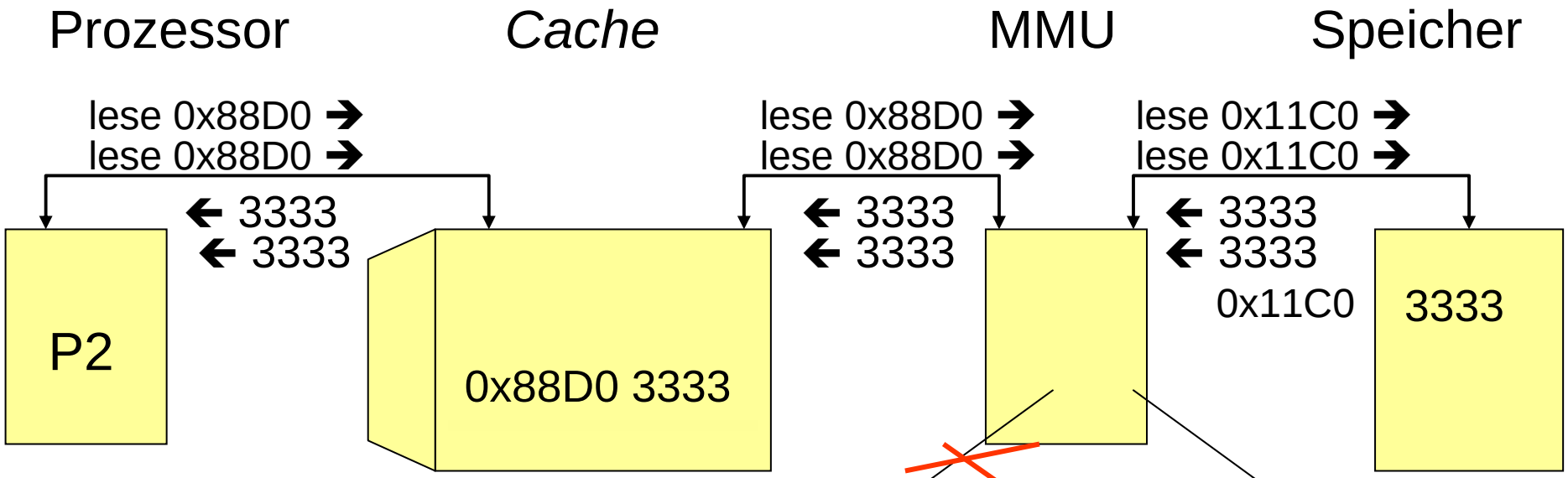
☞ Bei Seitenfehlern keine Aktion im *Cache.*

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

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

Virtueller *Cache* bei gemeinsam genutzter Seite (*sharing*)

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



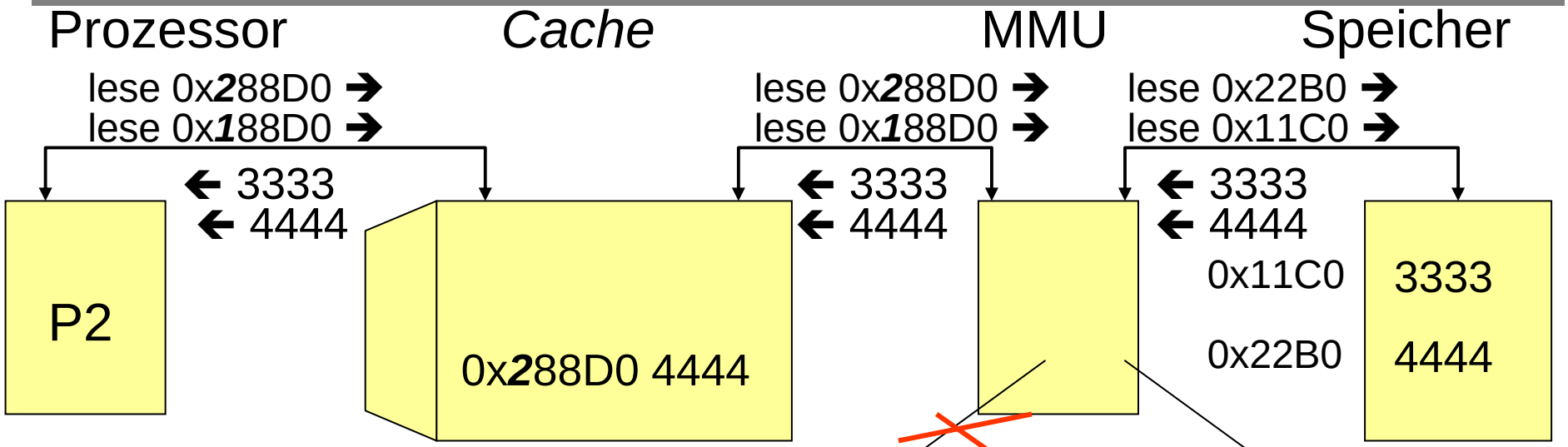
☞ Der zweite Prozess füllt den Cache erneut. Langsam, aber korrekt.

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

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

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

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!

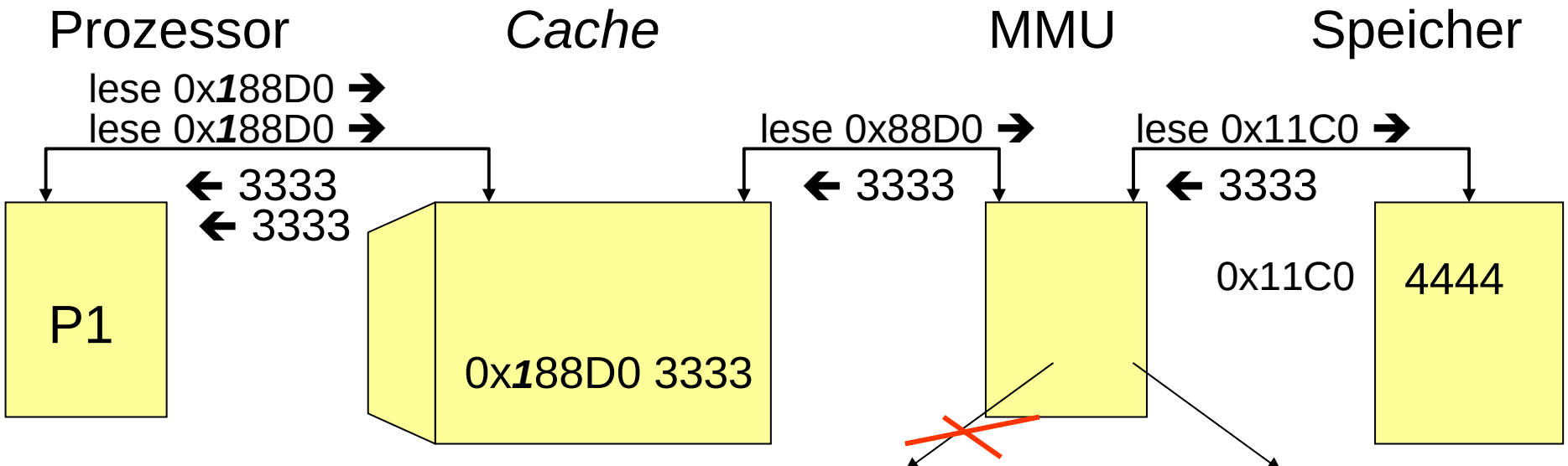
Prozess P1	
Seite	**Kachel**
0x88D.	0x11C.

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

TLBs in der MMU könnten PIDs nutzen.

Virtueller *Cache* mit PIDs bei Seitenfehler

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

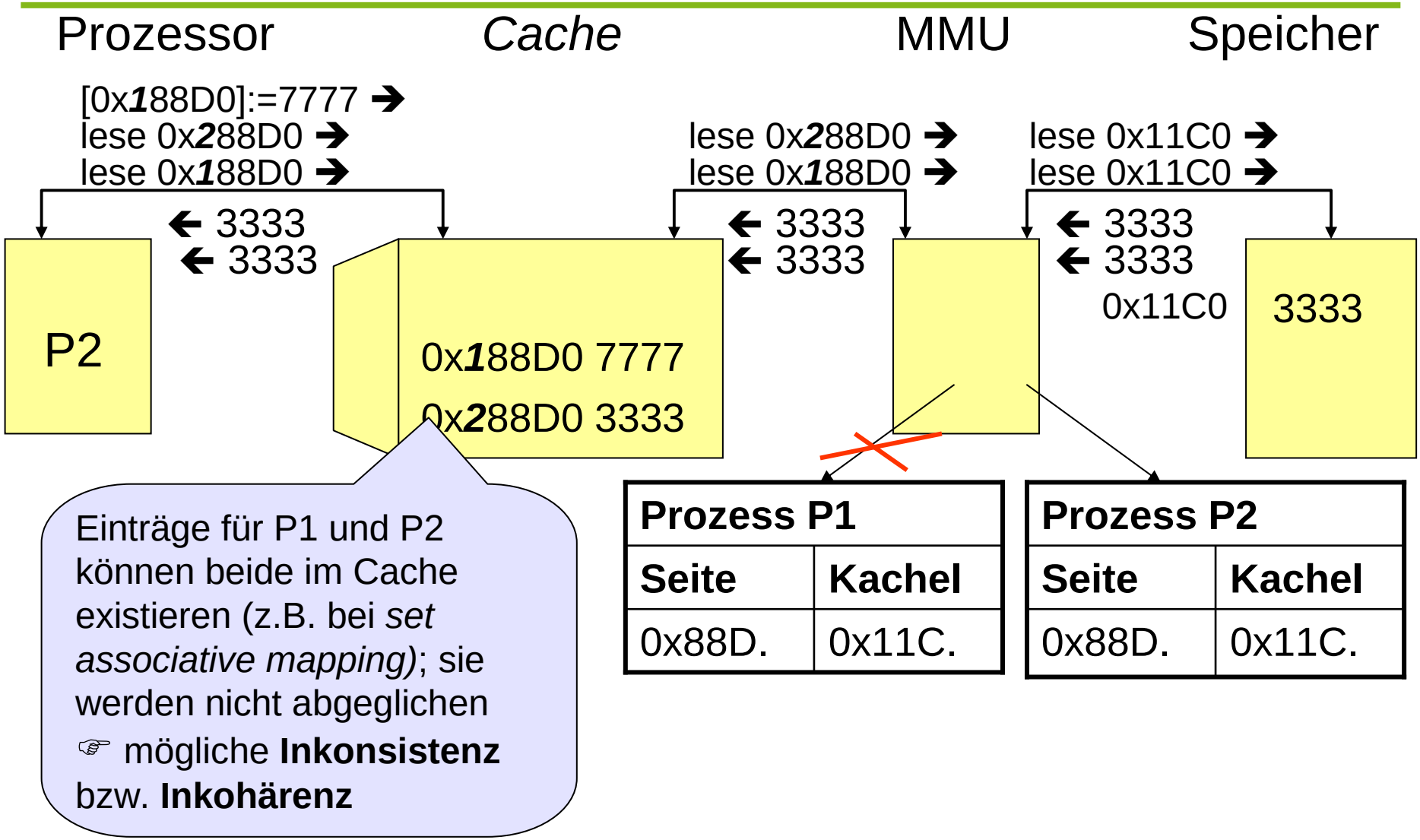


☞ Bei Seitenfehlern keine Aktion im *Cache.*

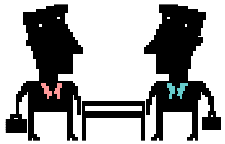
Vor Aus-/Einlagern	
Seite	**Kachel**
0x88D.	0x11C.

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

Virtueller *Cache* mit PIDs bei gemeinsam genutzter Seite (*sharing)*



Cache-Kohärenz

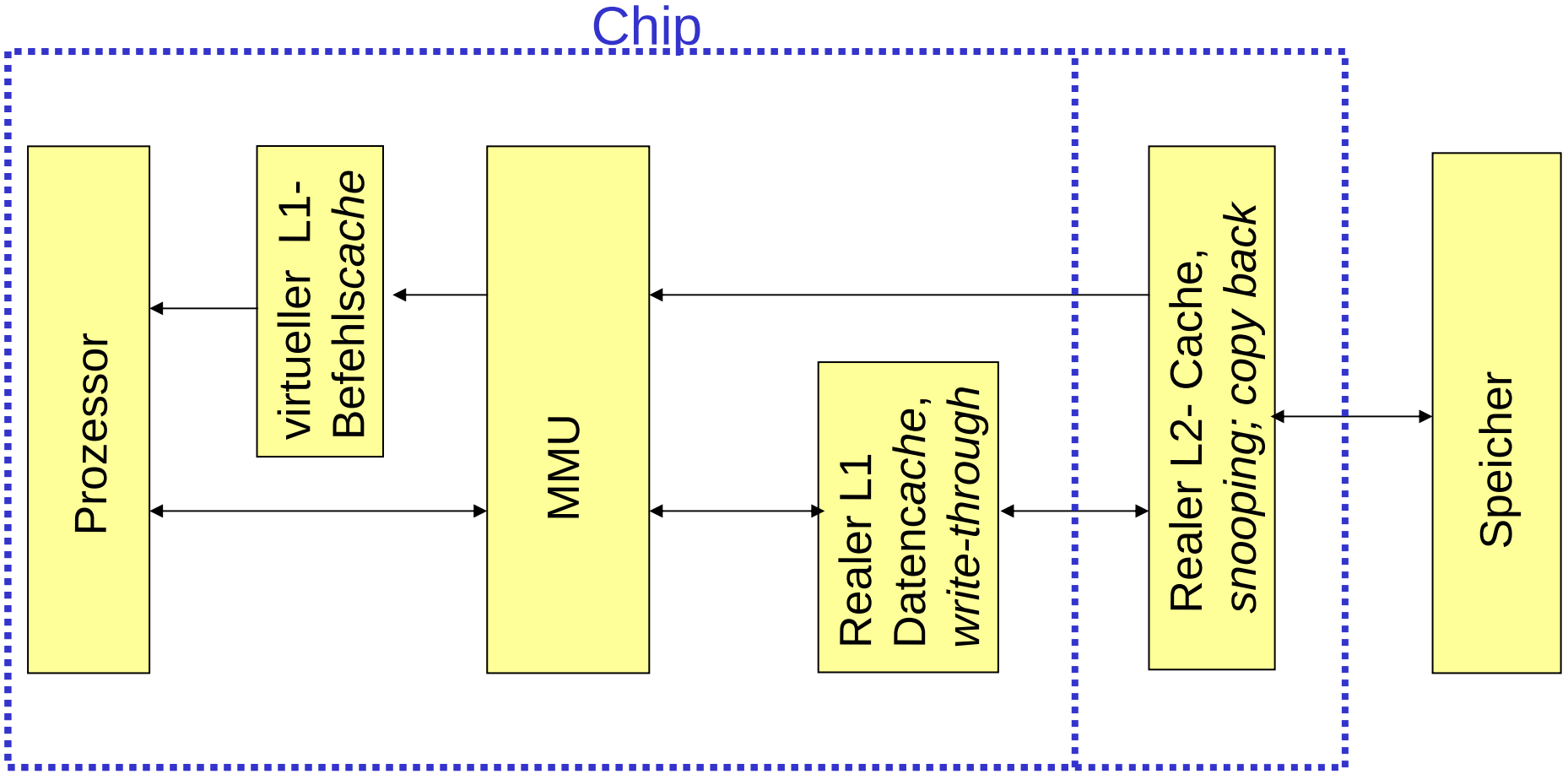


	Realer *Cache*	Virtueller *Cache* mit PIDs	Virtueller *Cache*
Inhalt nach Prozesswechsel	gültig	gültig, falls es ausreichend viele PIDs gibt	ungültig
Inhalt nach Seitenfehler	modifizierte Kachel ungültig	gültig	gültig*
Geschwindigkeit	langsam	schnell	schnell
Automatische Kohärenz bei *Sharing*	ja	nein	ja, aber Code muss neu geladen werden.
primäre Anwendung	große *Caches*	kleine *Caches*; Befehls*caches*, falls dynamisches Überschreiben von Befehlen verboten ist.	

* Potenzielle Schwierigkeiten, wenn Cacheinhalte $\not\subset$ Hauptspeicherinhalte.

Systeme mit mehreren Caches

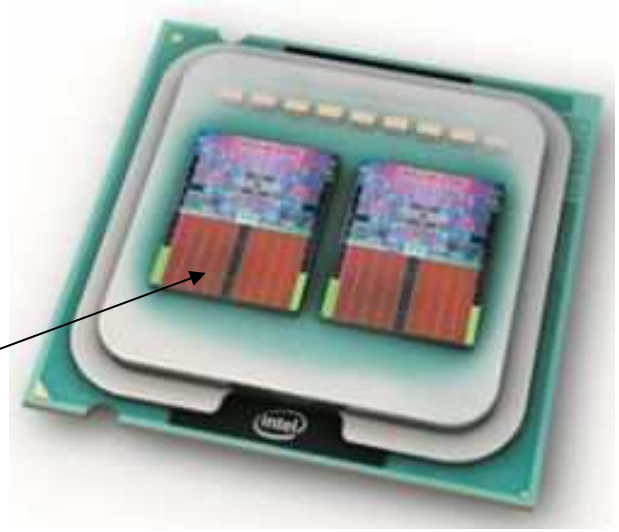
Beispiel:

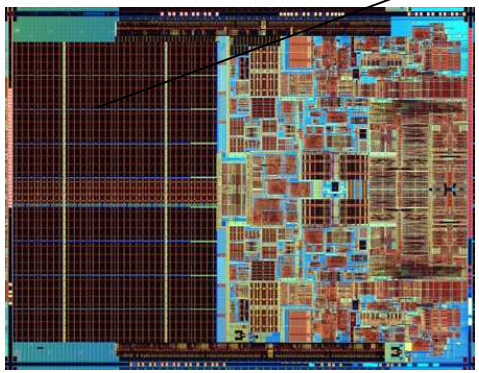


Beispiel: aktuelle Intel Prozessoren

Beispiel: Intel® Core™ 2 Extreme Quad-Core QX6000:

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





Austauschverfahren

Überschreiben von Einträgen in TLB, Caches, Kacheln:

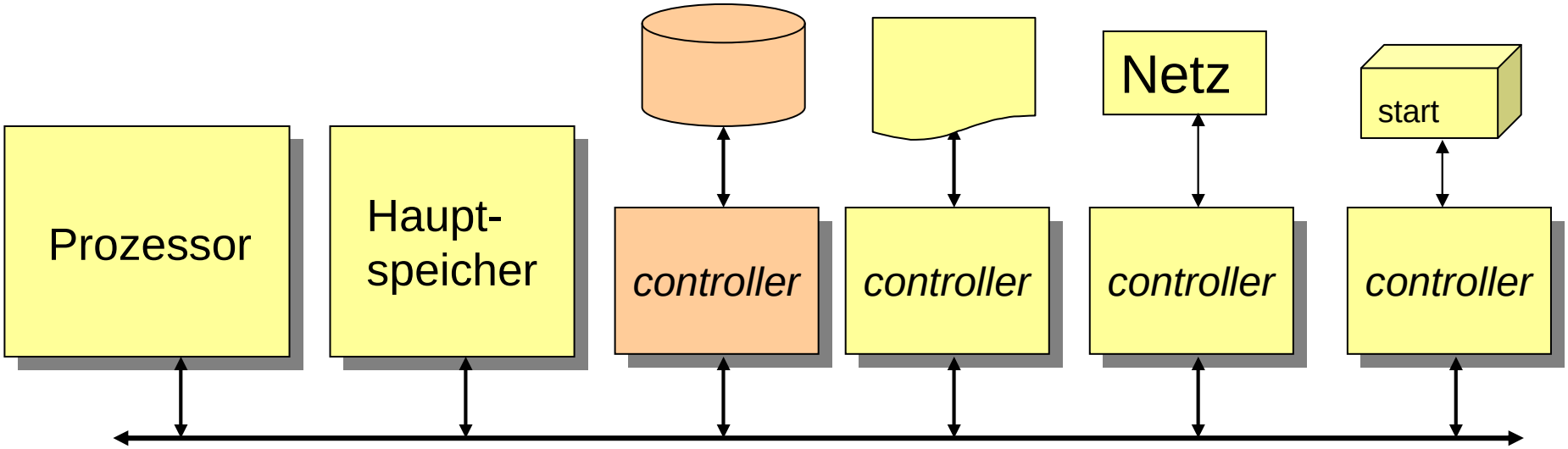
1. **Random- bzw. Zufallsverfahren**
   Ein zufällig ausgewählter Eintrag wird überschrieben.
2. **NRU, *not recently used***
   Einteilung von Einträgen in 4 Klassen:
   a. **Nicht benutzte, nicht modifizierte Einträge**
   b. **Nicht benutzte, modifizierte Seiten**
      Benutzt-Kennzeichnung wird nach einiger Zeit zurückgesetzt.
   c. **Benutzte, nicht modifizierte Seiten**
   d. **Benutzte, modifizierte Seiten**

   Überschrieben wird ein zufällig ausgewählter Eintrag der niedrigsten Klasse, die nicht-leer ist.
3. **LRU=*least recently used***
   Überschreiben des Eintrags, der am längsten nicht benutzt wurde.

Zusammenfassung

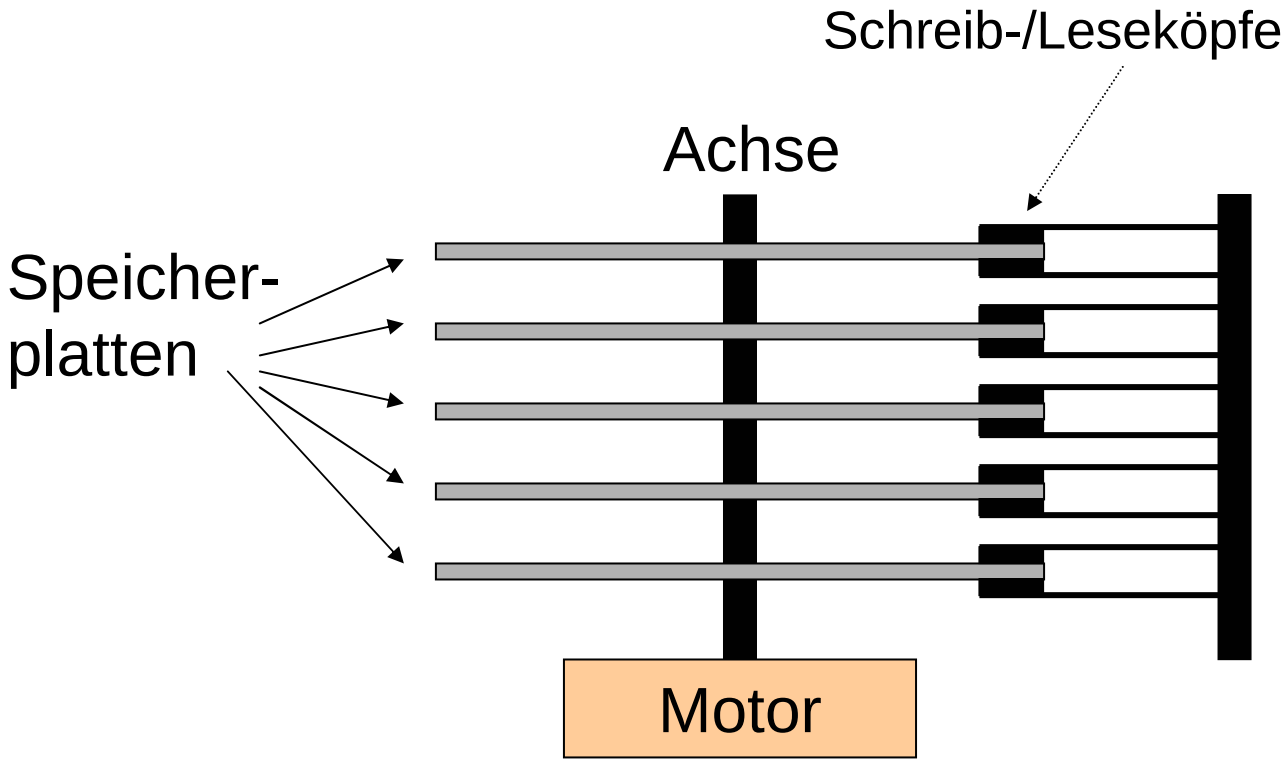
- Cache-Zeilen: Einheit der Gültigkeitsprüfung
- Cache-Blöcke: Einheit des Nachladens
- Organisationsformen:
  - *direct mapping*
  - *set associative mapping*
- Unterscheidung zwischen virtuellen und realen Caches.
- Moderne Systeme besitzen mehrere Cache-Ebenen

Massenspeicher

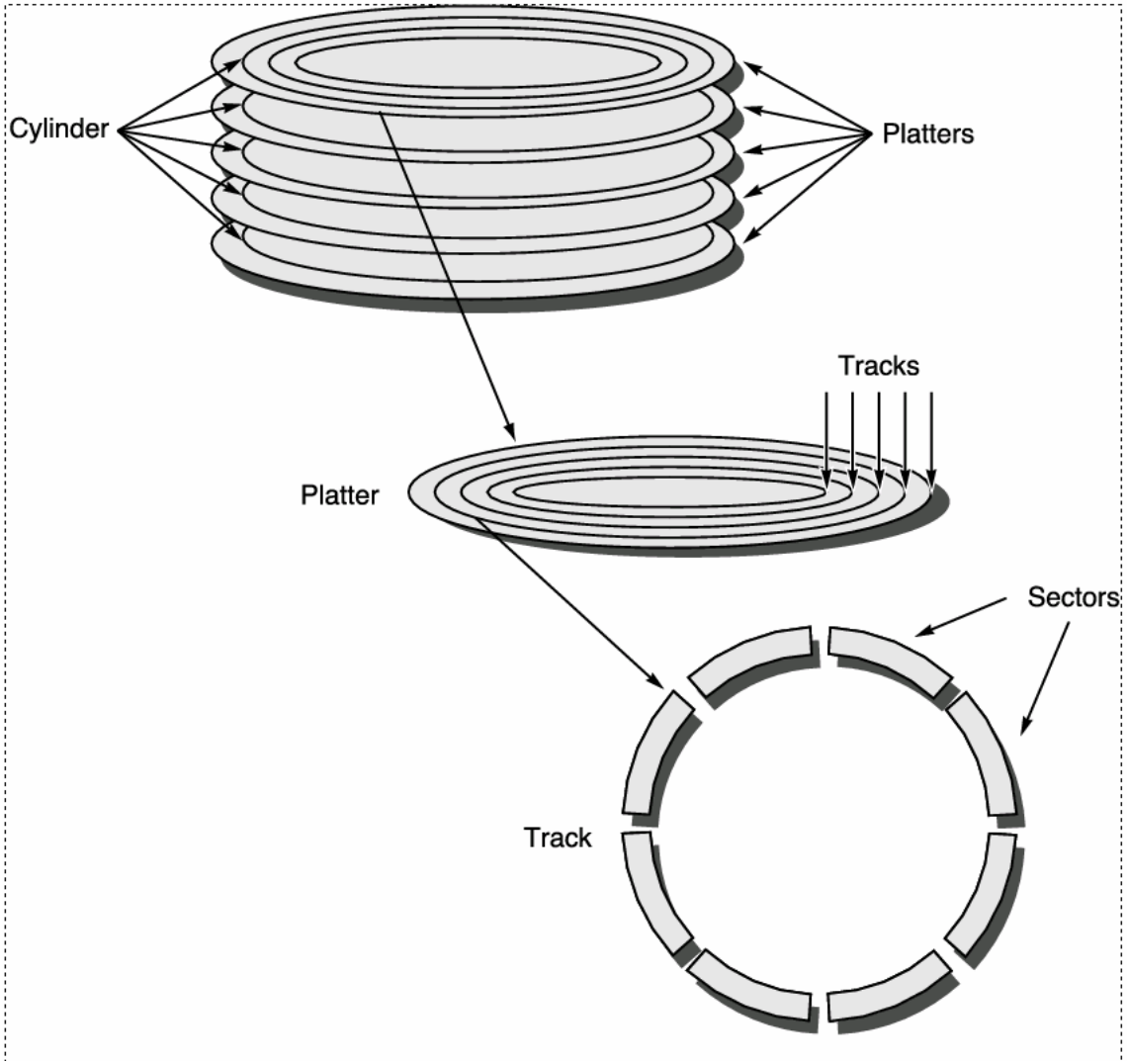


- Plattenspeicher
  - Disc-Arrays
- CD-ROM
- DVD
- Bandlaufwerke
- Weitere

Schematischer Aufbau eines Plattenlaufwerks



Einteilung der Platten in Sektoren, Spuren und Zylinder



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

Daten einiger Plattenlaufwerke (2000)

	Seagate Ultra 160 SCSI	IBM Travelstar ATA-4	IBM 1 GB Microdrive
Durchmesser ["]	3,5	2,5	1
Kapazität [GB]	73,4	32	1
Zylinder	14100	21664	7167
Platten	12	4	1
Gb/sq.in	6	14	15,2
Average seek [ms]	5,6 read/6,2 write	12	12
Übertragungsrate	27-40 MB/s	11-21 MB/s	2,6-4,2 MB/s
Leistung [W] *passiv/aktiv*	16,4/23,5	2,0/2,6	0,5/0,8
Puffer [MB]	4	2	0,125
Schock-Toleranz [Betrieb/außer Betrieb]	10 G/ 175 G	150 G/700 G	175 G/1500 G

[Hennessy/Patterson, *Computer Architecture*, 3. Aufl.]

Trends

- Schwergewicht bei Steigerung der Kapazität:
  1988: 29%/Jahr;
  1996: 60%/Jahr;
  2001: 100%/Jahr
- Kosten pro GB zwischen 1983 und 2000 um Faktor 10000 reduziert.
- Zugriffszeit um ca. 10% pro Jahr verbessert.
- Versuche, Lücke in Zugriffszeit zu füllen, über Jahrzehnte fehlgeschlagen.
  Wird es mit Flash-Speicher gelingen?

[Hennessy/Patterson, *Computer Architecture*, 3. Aufl.]

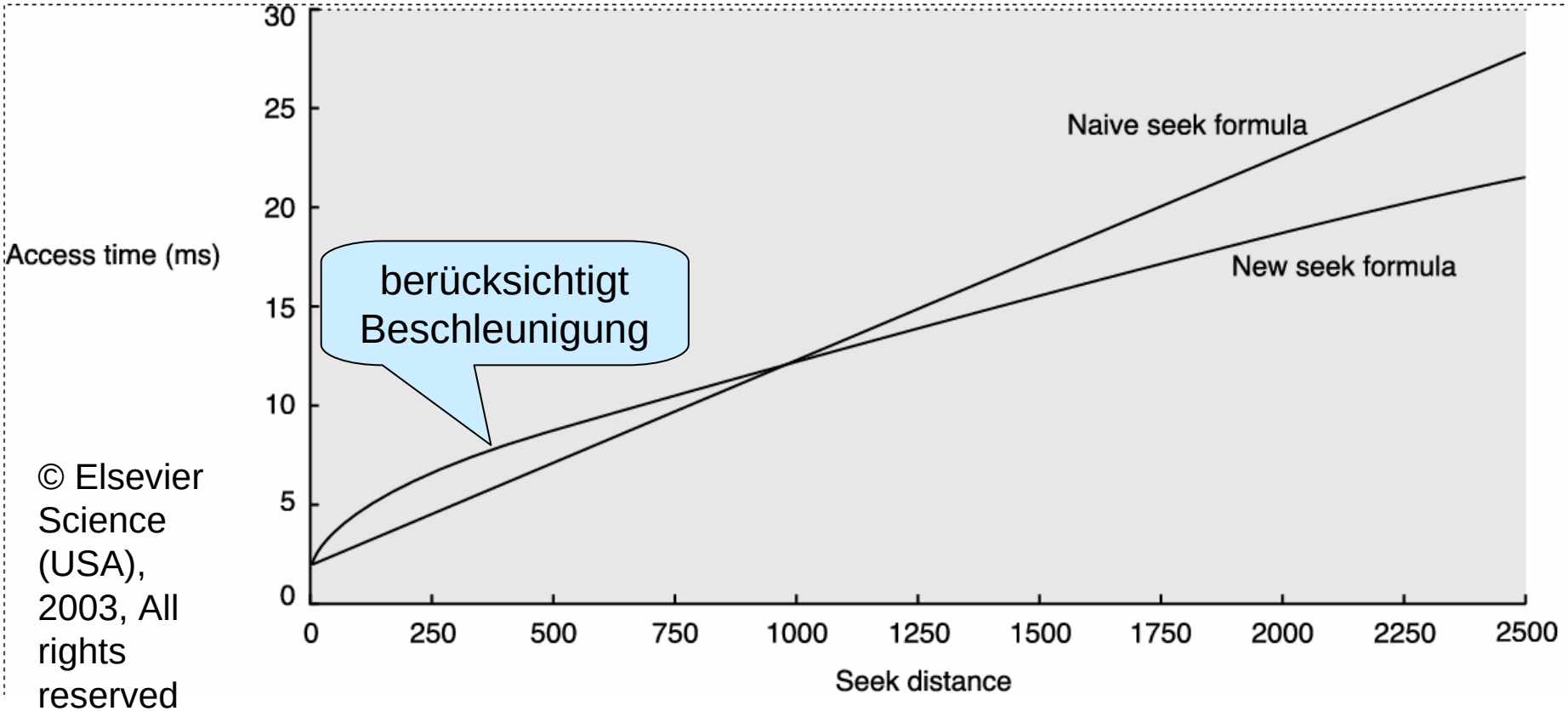
Vergleich *Harddisc/Flash*-Speicher

	Sandisk Type I Flash	Sandisk Type II Flash	IBM Microdrive DSCM-10340
Kapazität [MB]	64	300	340
Leistungsaufnahme [W] (standby/operating)	0,15/0.66	0,15/0,66	0,07/0.83
Mögl. Schreibzyklen	300.000	300.000	keine Einschränkung
Mean-time between failures [h]	>1.000.000	>1.000.000	service-life=min(5J, 8800 h Betrieb)
Fehlerraten, unkorrigb.	< 1 per 10^{14}	<1 per 10^{14}	<1 per 10^{13}
Ein/Ausschaltvorgänge	beliebig	beliebig	300.000
Schock-Toleranz	2000 G; 2000 G	2000 G;	175 G; 1500 G

[Hennessy/Patterson, *Computer Architecture*, 3. Aufl.]

Flash: - weniger Energie als Platten;
Lesezeiten ähnlich DRAMs (20-70 ns);
Löschzeiten von 1-2 s/64kB NOR-Flash, 5-6ms/4-8kB NAND-Flash
interessant für kleinere Speicher

Abhängigkeit der Suchzeit von der Spurdifferenz

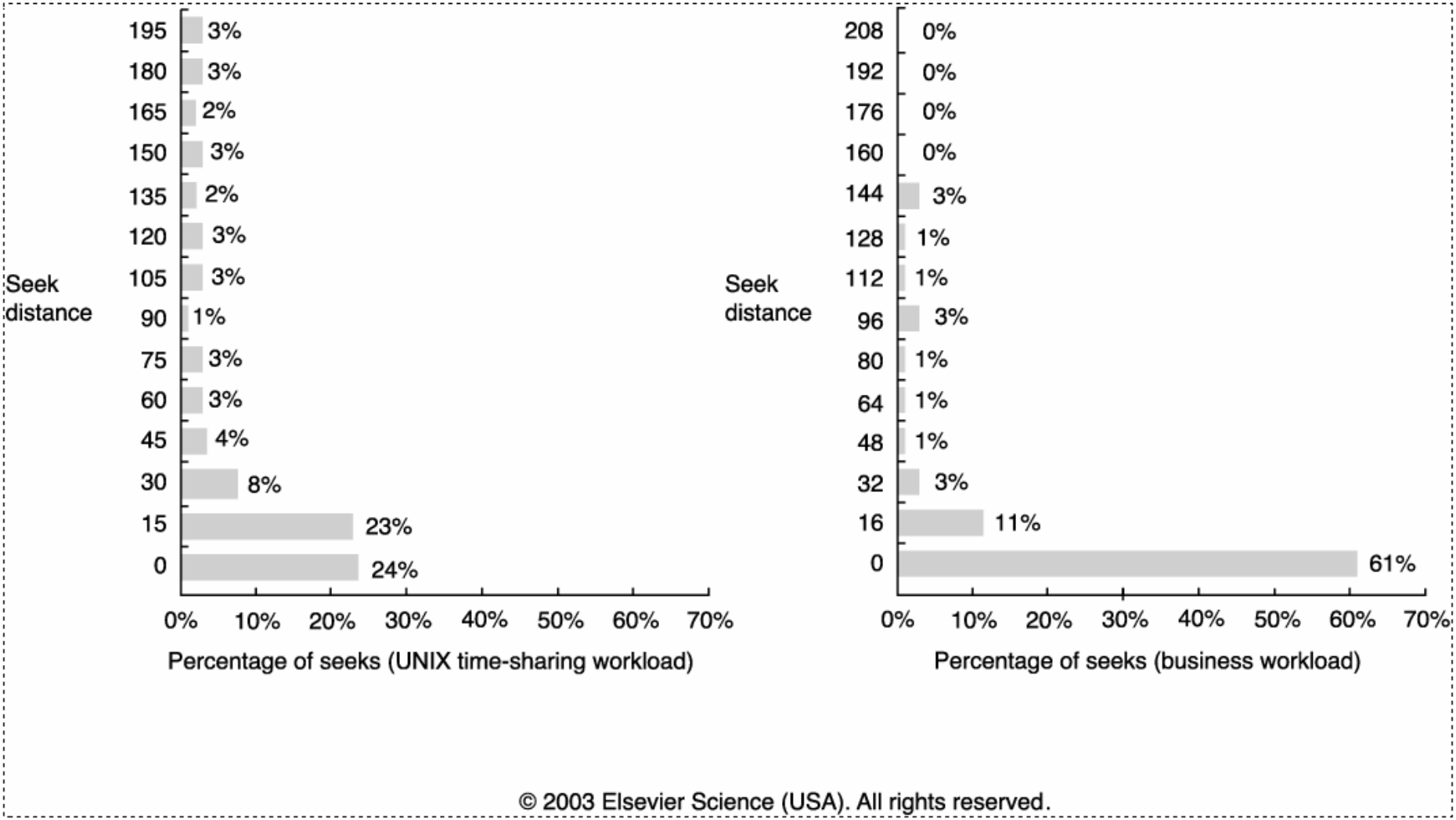


$$\text{"Naive formula": Accesstime(distance)} = Time_{\min} + \frac{\text{distance}}{\text{distance}_{ave}} \times \left(Time_{ave} - Time_{\min}\right)$$

$$\text{"New formula": Access time(distance)} = a \times \sqrt{\text{distance} - 1} + b \times (\text{distance} - 1) + c$$

[Hennessy/Patterson, *Computer Architecture*, 3. Aufl.]

Verteilung der Spurdifferenzen



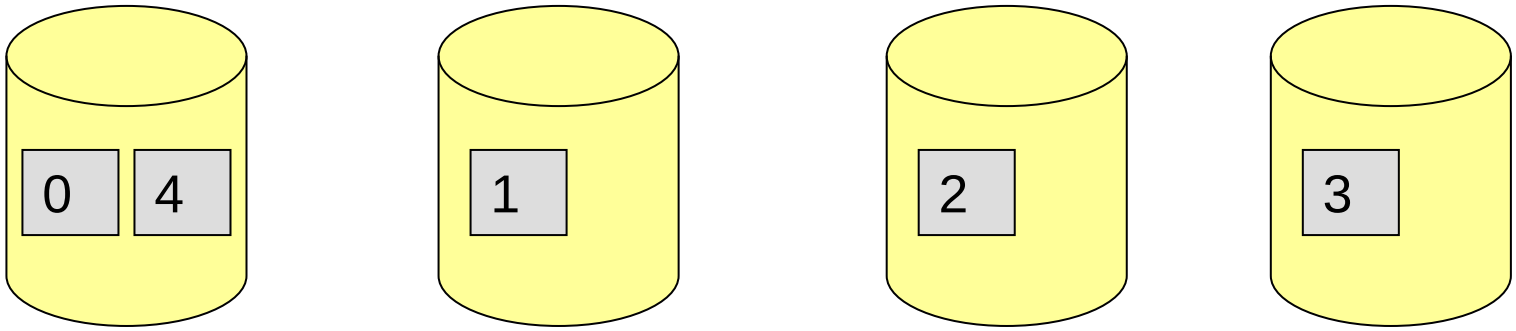
Redundant arrays of inexpensive discs (RAID)

- Verbesserung der E/A-Geschwindigkeit durch paralleles Lesen/Schreiben auf mehrere Platten;
- Einsatz von Redundanz, damit dadurch nicht die Zuverlässigkeit leidet.

☞ *Disc arrays, drive arrays*

RAID 0 (*Striping*)

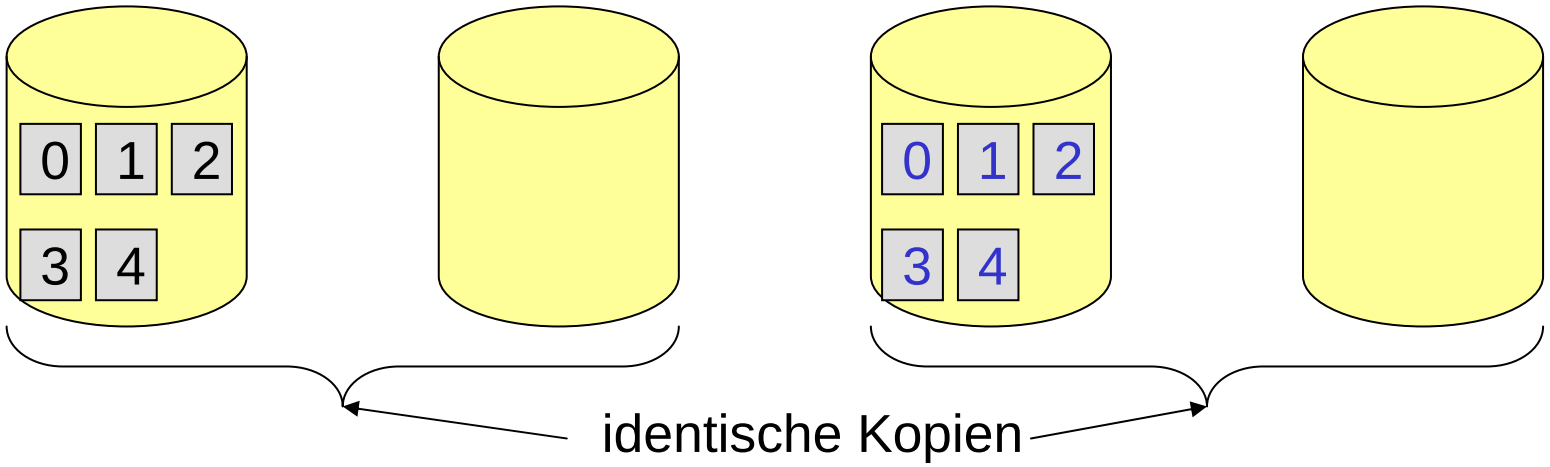
Dateien werden unterteilt in so genannte ***Stripes***.
Striping-Parameter = Länge eines *Stripes.*
Diese *Stripes* werden auf den Platten verteilt.



- Keine Redundanz,
- nur Verbesserung der Übertragungsrate

RAID 1 (*Mirroring*)

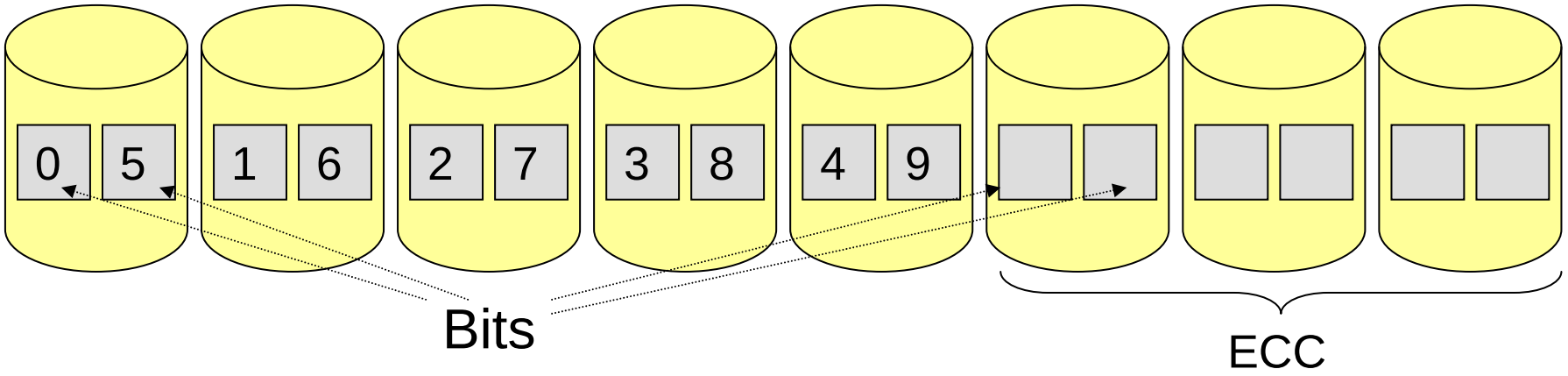
Dieseleben Informationen werden auf zwei Platten geschrieben (gespiegelte Platten, *mirrored discs*)



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

RAID 2 (ECC)

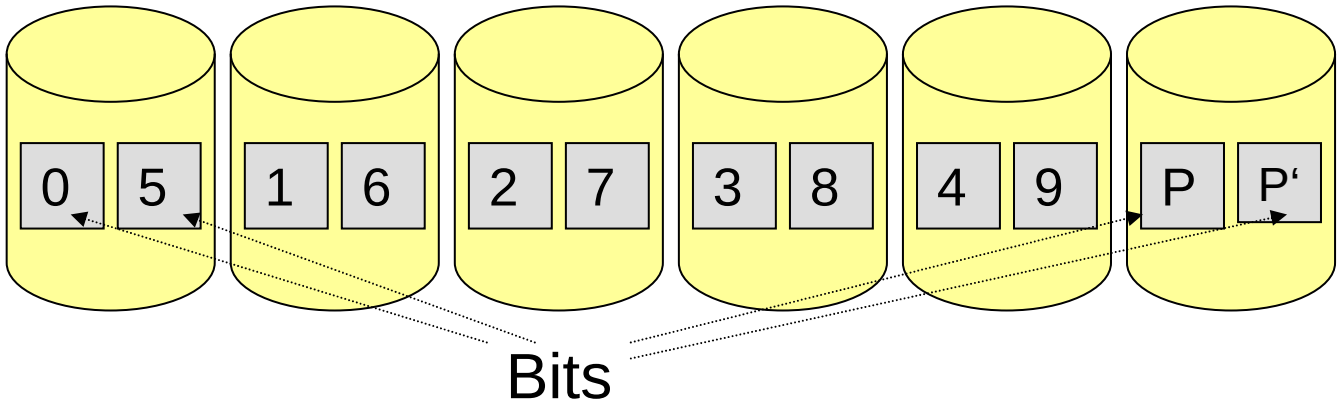
Zusätzlich werden Prüfbits auf spezielle Platten geschrieben, Verwendung von fehlerkorrigierenden Codes (*error correcting code*, ECC).
Die **Bits** werden auf mehreren Platten verteilt.



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

RAID 3 (*dedicated parity*)

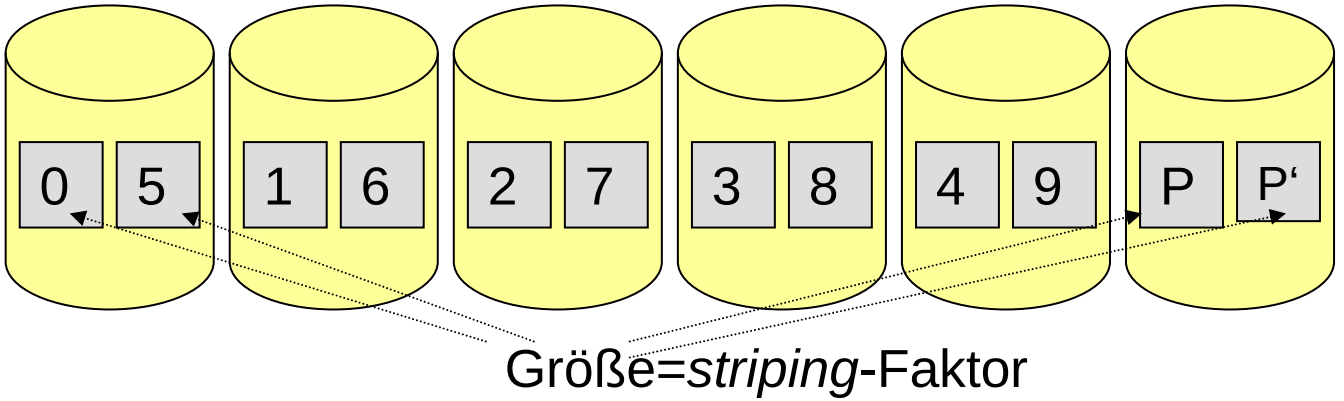
Es nur ein einzelnes Paritäts-*Stripe* auf einer *Parity*-Platte abgelegt.
Häufig nur der Fall *Stripe*-Parameter=1 Bit betrachtet.
Die ***Stripes*** eines Datenblocks werden auf Platten verteilt.



Selbst für kleinste zu schreibende/lesende Blöcke alle Platten aktiv. Keine gute *Performance.*

RAID 4

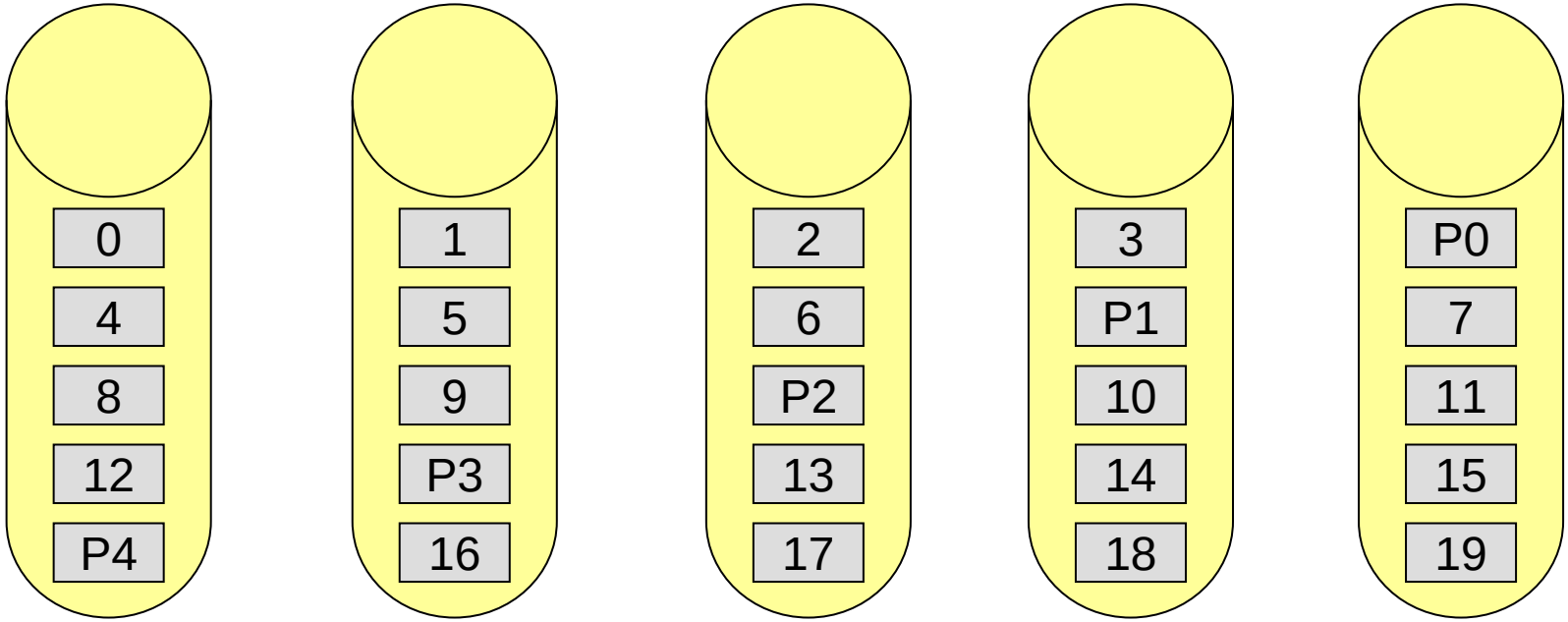
Wie RAID-3, jedoch mit einem *Striping*-Faktor von einem Block und mehr.



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

RAID 5 (*distributed parity*)

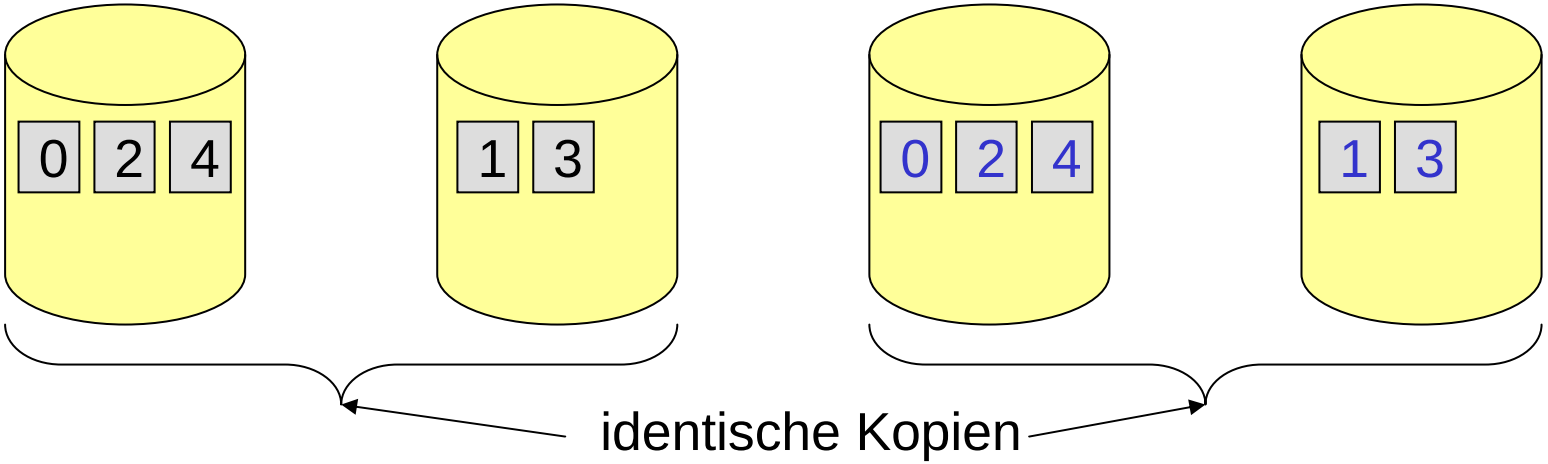
Paritätsinformation wird über verschiedene Platten verteilt



Paritätsplatte nicht weiter ein Engpass

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

Gespiegelte Platten (RAID 1), mit verteilten Stripes (RAID 0) ☞ *mirrored stripes*.



- Lesegeschwindigkeit erhöht,
- erhöhte Fehlersicherheit,
- stark erhöhter Aufwand.

RAID 1+0, RAID 10 (*striped mirrors*)

Striped mirrors: *striping* von gespiegelten Platten

- Lesegeschwindigkeit erhöht,
- erhöhte Fehlersicherheit,
- stark erhöhter Aufwand.

Informationen zu neuen Varianten:
http://www.pcguide.com/ref/hdd/perf/raid/levels/multLevel01-c.html

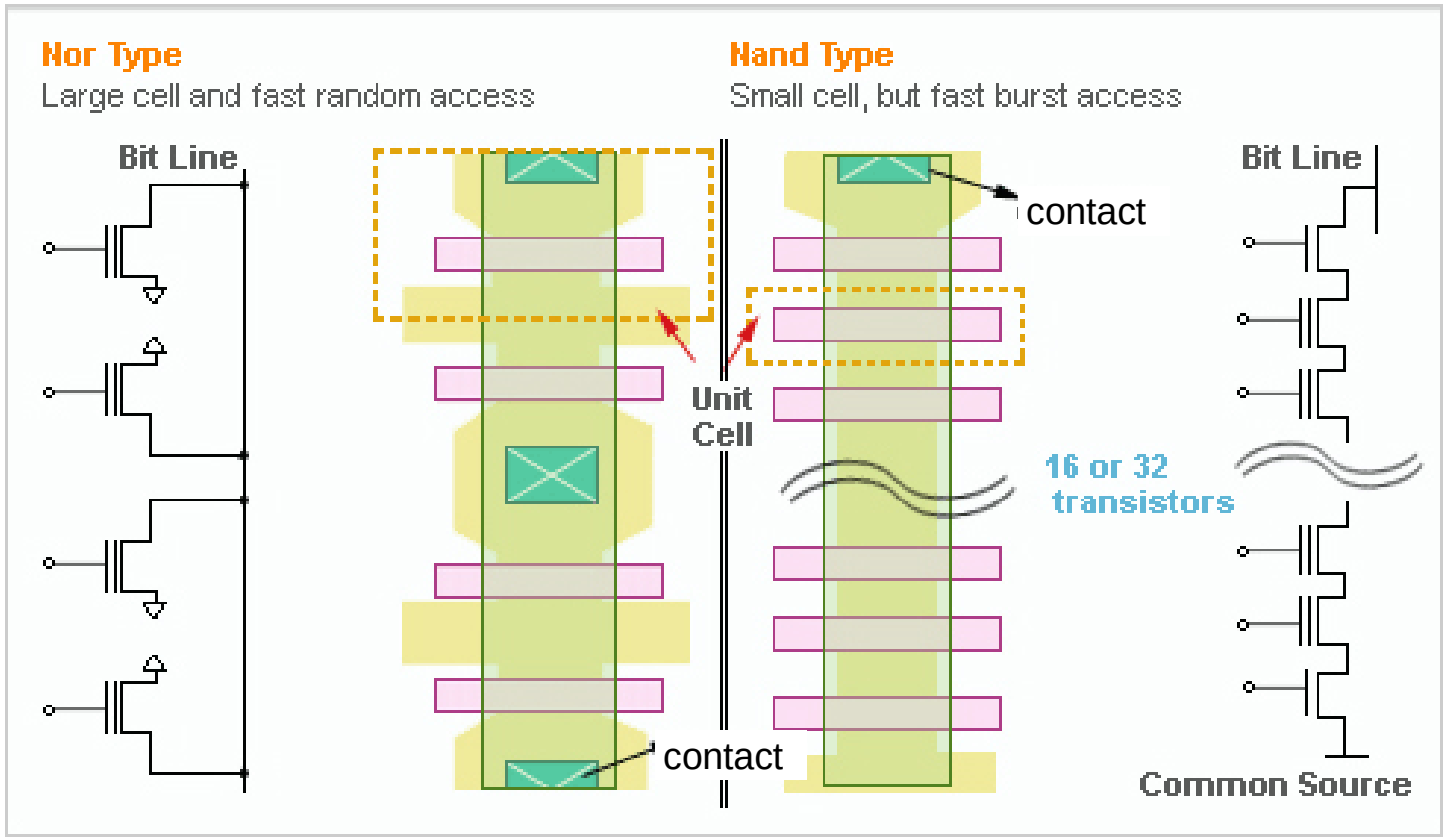
Übersicht

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	*P+Q redundancy*	2	8	2

bevorzugt

NOR- and NAND-Flash

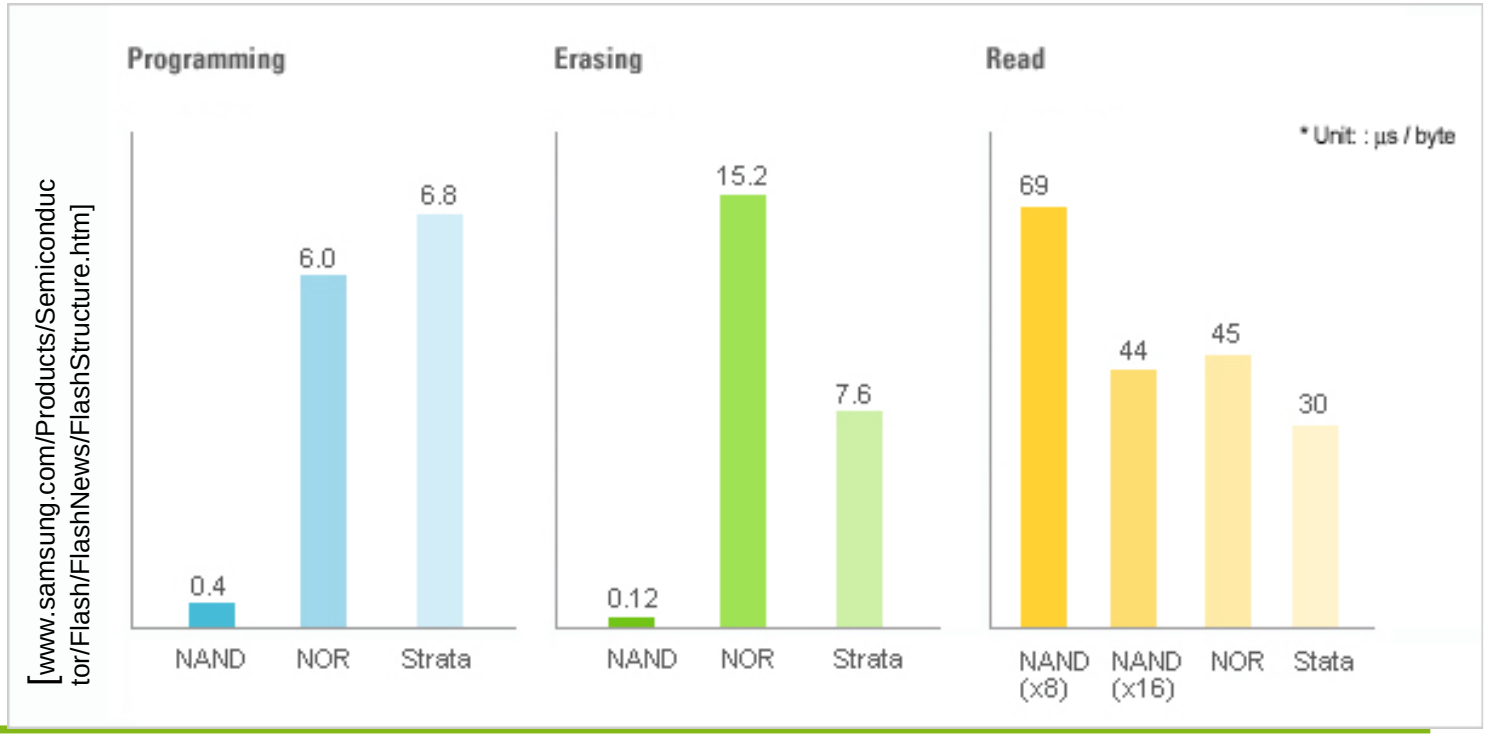
NOR: Transistor between bit line and ground
NAND: Several transistor between bit line and ground



was at [www.samsung.com/Products/ Semicon-ductor/ Flash/FlashNews/ FlashStructure. htm] (2007)

Properties of NOR- and NAND-Flash memories

Type/Property	NOR	NAND
Random access	Yes ☺	No ☹
Erase block	Slow ☹	Fast ☺
Size of cell	Larger ☺	Small ☺
Reliability	Larger ☺	Smaller ☹
Execute in place	Yes ☺	No ☹
Applications	Code storage, boot flash, set top box	Data storage, USB sticks, memory cards



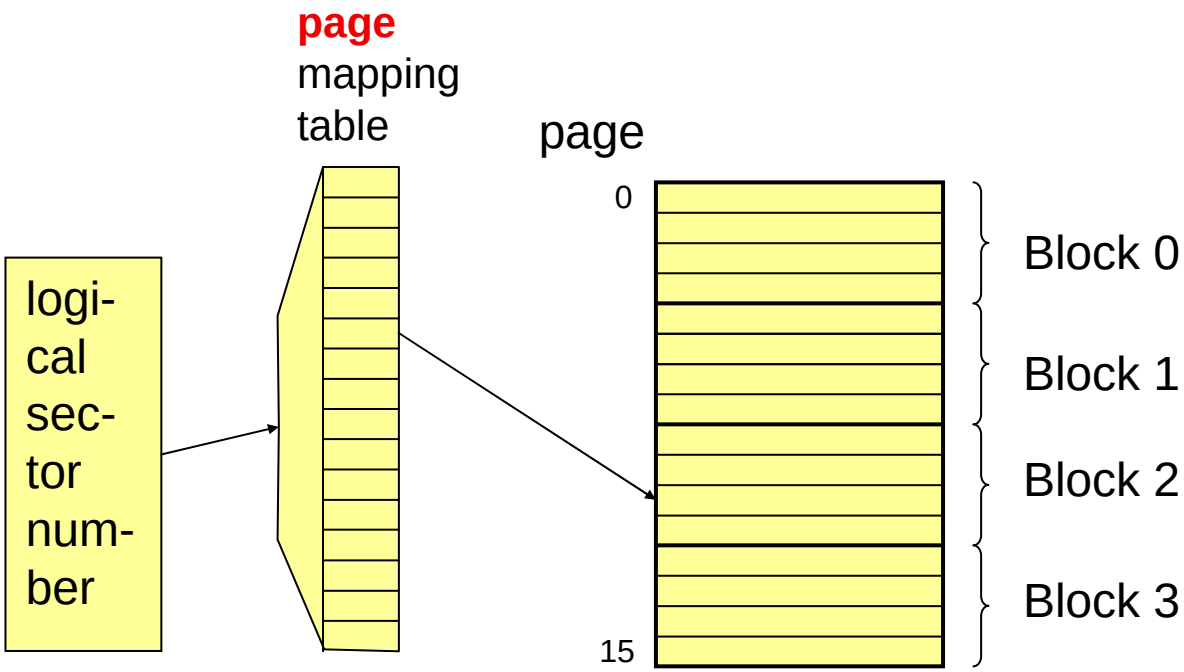
Characteristics of NAND Flash memory

Memory partitioned into blocks (typ. 16-256 KB),
blocks partitioned into pages (typ. 0.5-5 KB).
Read/write operations performed in page units.

	Single Level Cell (SLC)	Multi Level Cell (MLC)
Read (page)	25 µs	≫ 25 µs
Write (page)	300 µs	≫ 300 µs
Erase (block)	2 ms	1.5 ms

J. Lee, S. Kim, H. Kwin, C. Hyun, S, Ahn, J. Choi, D. Lee, S.Noh: Block Recycling Schemes and Their Cost-based Optimization in NAND Flash Memory Based Storage System, EMSOFT'07, Sept. 2007

Page/sector mapping flash transaction layer (FTL)



Inverted page table stored in flash memory (extra bits);
"normal page" table constructed during initialization.
Page table may become large
Used in low capacity NOR Flash memories

sector ≈ page
+ extra bits

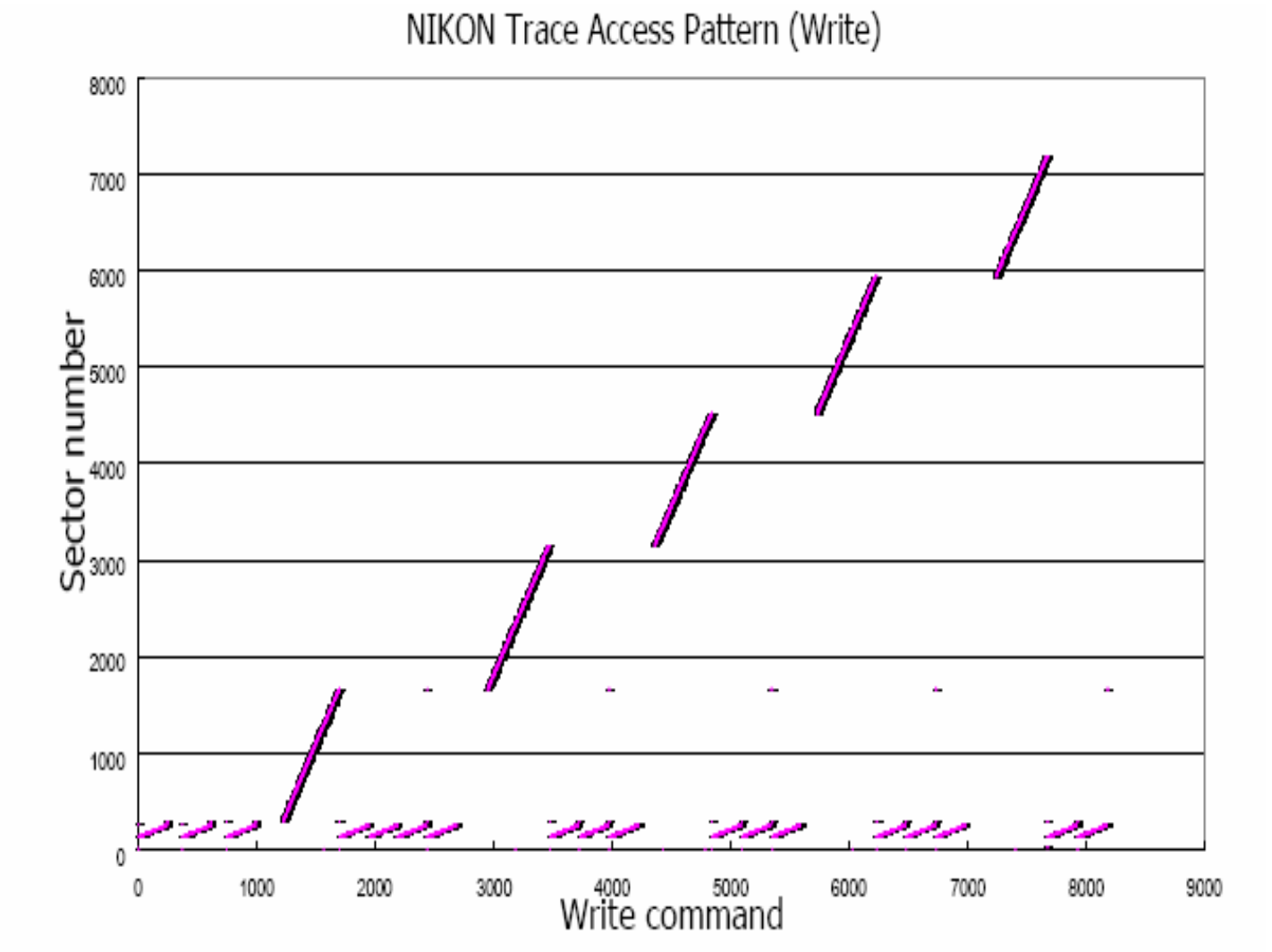
Comparison Flash/Microdrive

	Sandisk Type I Flash	Sandisk Type II Flash	IBM Microdrive DSCM-10340
Capacity [MB]	64	300	340
Power [W] (standby/operating)	0,15/0.66	0,15/0,66	0,07/0.83
Write cycles	300.000	300.000	unlimited
Mean-time between failures [h]	>1.000.000	>1.000.000	service-life=min(5J, 8800 h operating)
Error rates, uncorrectable	< 1 per 10^{14}	<1 per 10^{14}	<1 per 10^{13}
Max. power ons	unlimited	unlimited	300.000
Shock tolerance	2000 G; 2000 G	2000 G;	175 G; 1500 G

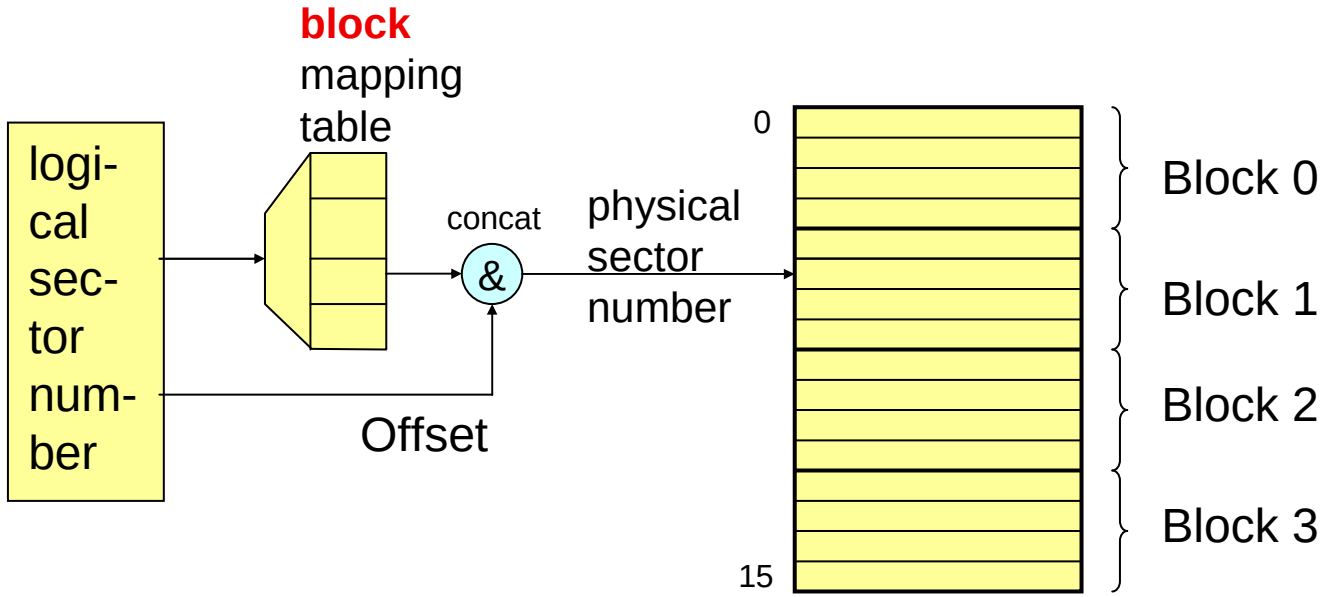
Source: Hennessy/Patterson, Computer Architecture, 2002

Exploiting regularity

Usually, long sequence of sequential writes



Block mapping flash transaction layer (FTL)



- Mapping tables smaller than for page-based FTLs.

☞ used in high capacity NAND Flash memories

- Overall operation is simple,
- but successive writes require copying into a new block
- Degraded performance for random and repeated writes.

☞ Hybrid schemes

Wear-leveling

- Example (Lofgren et al., 2000, 2003):
  - Each erase unit carries erase counter
  - One erase unit set aside as a spare
  - When one of the most worn out units is reclaimed, its counter is compared to least-worn out unit. If Δ is large:
    - content of least-worn-out ($\approx$ constants) $\rightarrow$ spare
    - content of most worn-out $\rightarrow$ least worn-out
    - most worn-out unit becomes the new spare

  Counter increment may be lost if power is lost between erase and counter update

☞ Attempts to avoid erase counter in the same erase unit

Source: Gal, Toledo, *ACM Computing Surveys*, June 2005

Flash-specific file systems

- Two-layer approach can be inefficient:
  - FTL emulates flash as a magnetic disc
  - Standard file system assumes magnetic disc

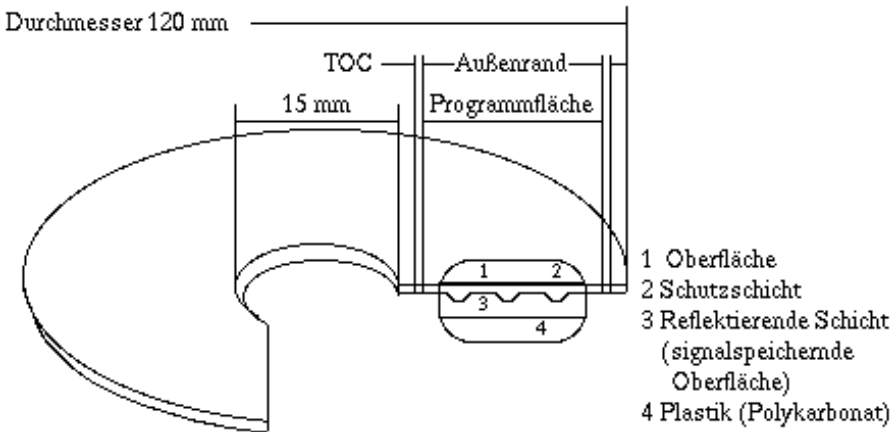
  Example: deleted sectors not marked ☞ not reclaimed
- Log-structured file systems just append new information
  - For disc-based file system:
    - Fast writes
    - Slow reads (head movement for gather operations)
  - Ideal for flash-based file system:
    - Writes done in new sectors
    - Reads not slow: no head movement

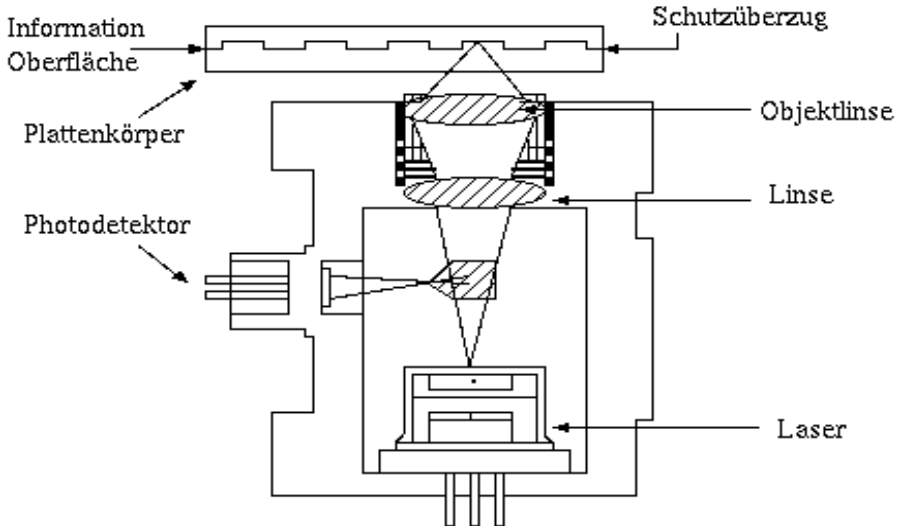
☞ Specific log-based flash file systems
  - JFFS2 (NOR)
  - YAFFS (NAND)

Source: Gal, Toledo, *ACM Computing Surveys*, June 2005

CD-ROM

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

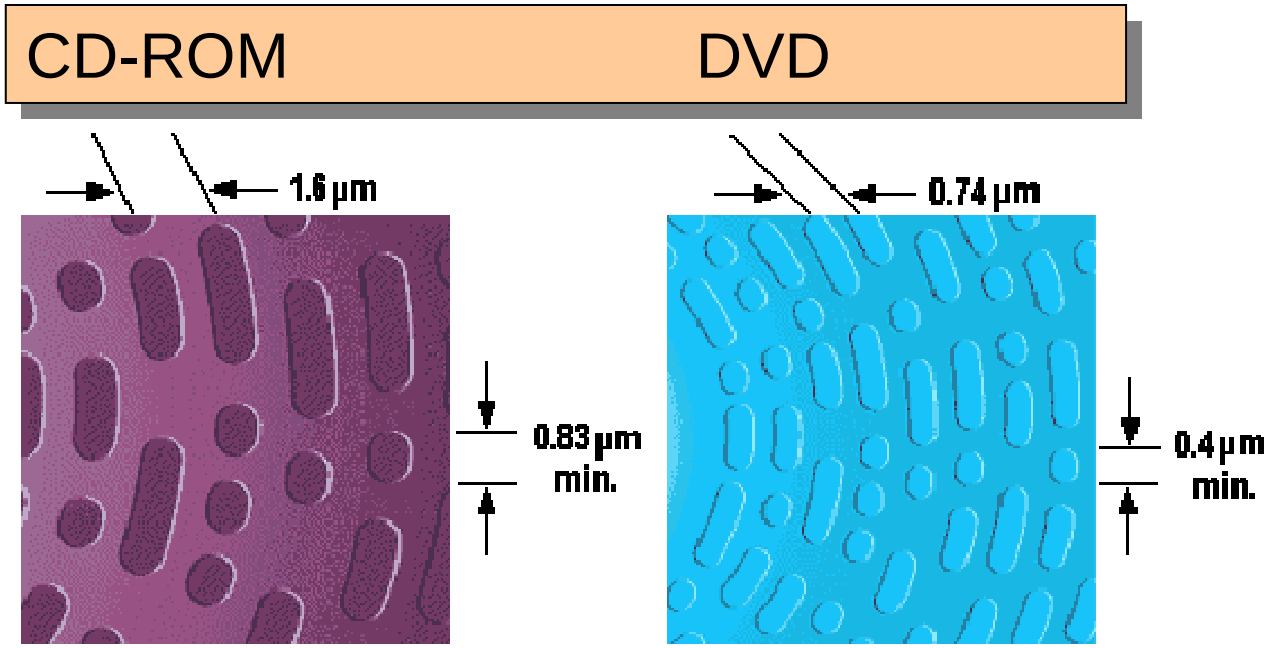




DVD-Laufwerke (1)

Kapazität gegenüber CDs erhöht:

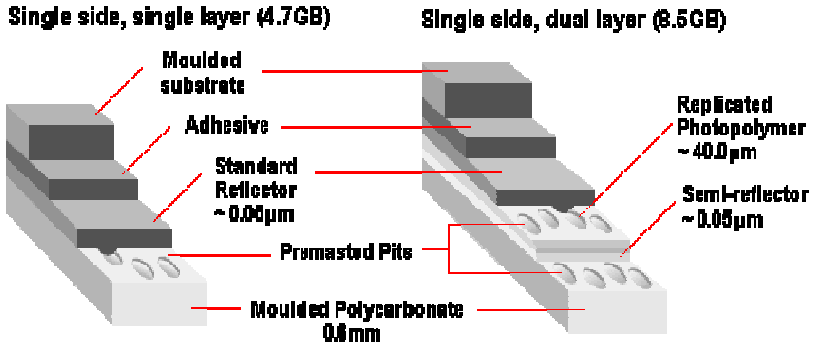
1. Abstände und Größe der Vertiefungen werden reduziert,
   Abtastung mit 650 nm (rotem) Laserstrahl:

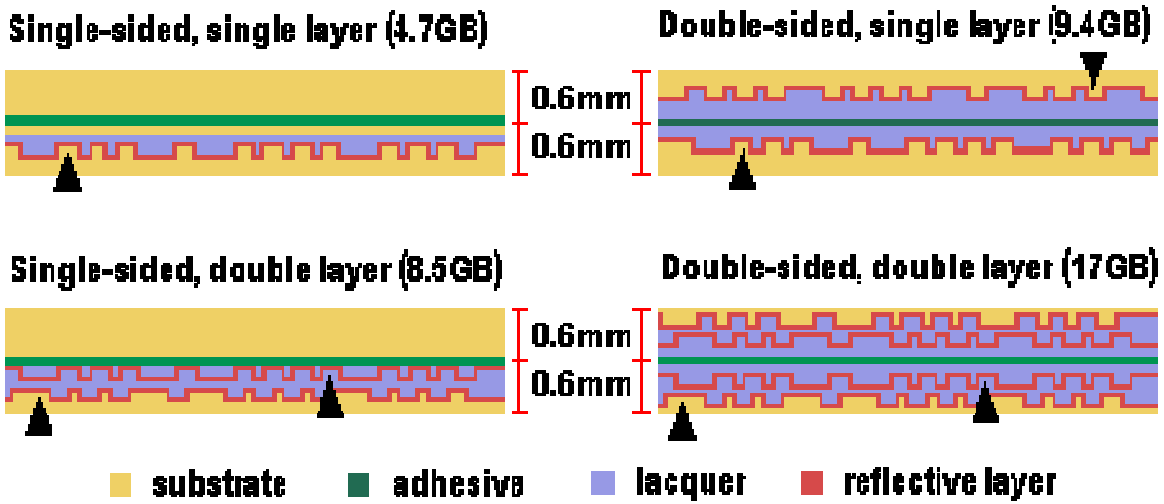


☞ Erhöhung der Kapazität um den Faktor 4.

DVD-Laufwerke (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.

Vergleich Übertragungsgeschwindigkeiten

Annahme: Transport einer DVD, einseitig in einer Ebene beschrieben (Kapazität=4,7GB) auf der Strecke Paris-Dakar (Strecke=6180km) mit einer Fahrrad-Staffel mit 36km/Std im Mittel. Gleichzeitig ISDN-Übertragung mit Übertragungsrate 64 kBit/s. Wie viel Prozent der maximalen Übertragungsrate muss die ISDN-Strecke erreichen, damit nicht die Radfahrer gewinnen ? (Vernachlässigung von *bit stuffing*)

$x \times$ Übertragungsrate $\times$ Zeit des Radfahrers = Kapazität

x = Kapazität $\times$ Geschwindigkeit / (Übertragungsrate $\times$ Strecke)

$x = 4{,}7 \times \overset{16}{\cancel{1024}} \times \cancel{1024} \times 1024 \times 8 \times 36 / (\cancel{64} \times \cancel{1024} \times 60 \times 60 \times 6180)$

$x = 4{,}7 \times 16 \times 1024 \times 8 \times 36 / (60 \times 60 \times 6180)$

$x = 0{,}9968$

Die ISDN-Strecke müsste 99,68 % der max. Übertragungsrate erreichen.

Blu*-ray disc (BD) : Motivation

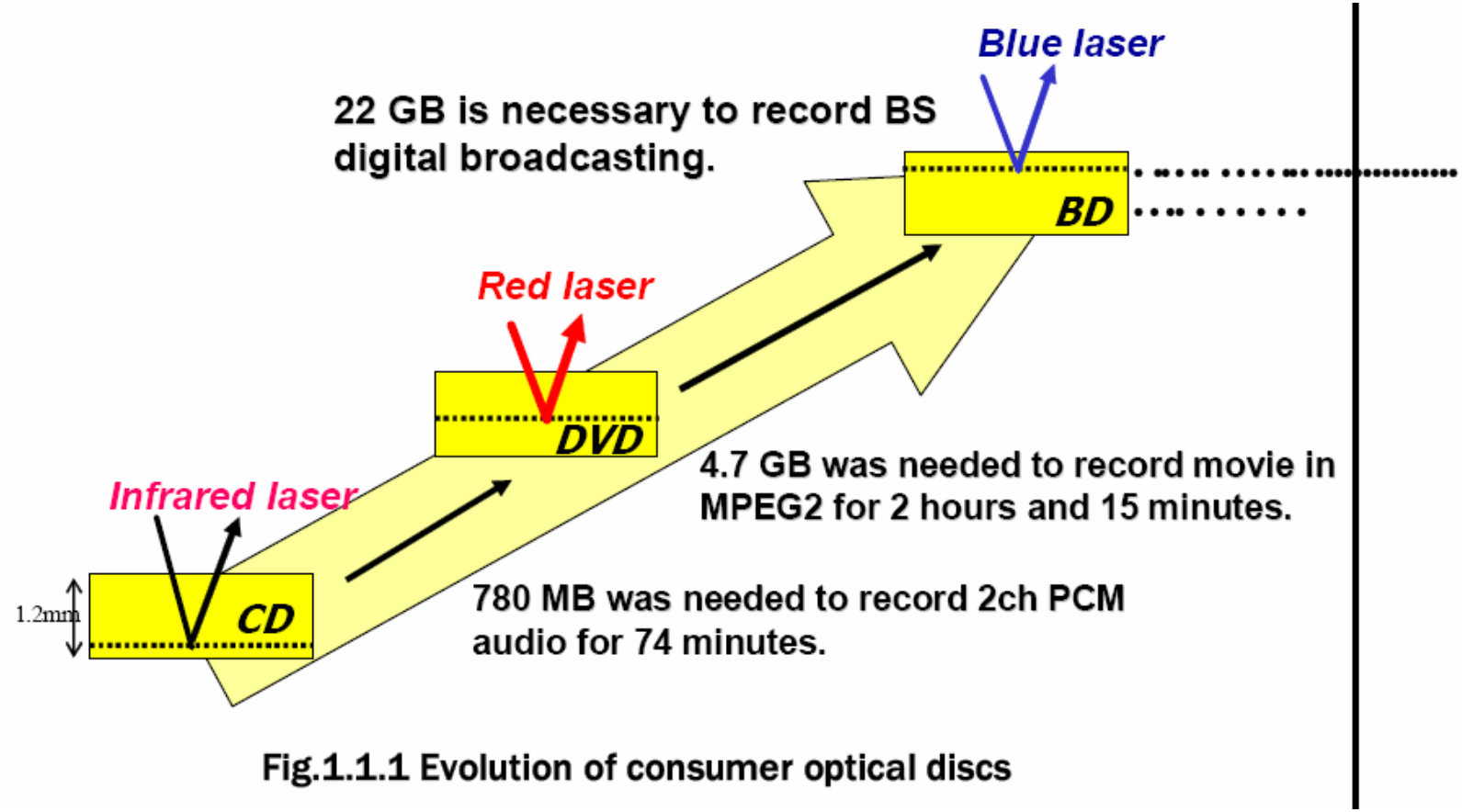


Fig.1.1.1 Evolution of consumer optical discs

*Falsche Schreibweise, damit Eintrag als Markenzeichen möglich

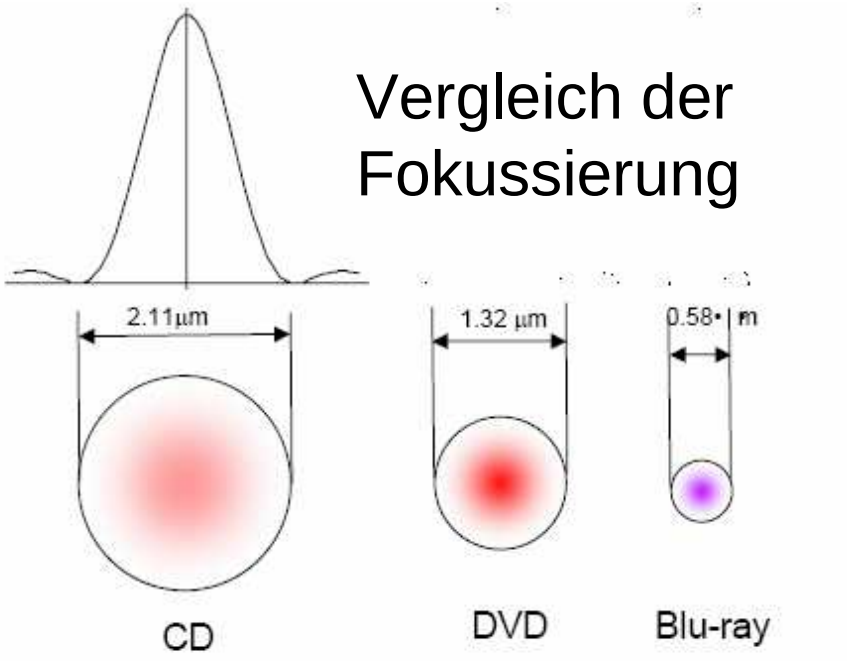
http://www.blu-raydisc.com/assets/downloadablefile/general_bluraydiscformat-12834.pdf

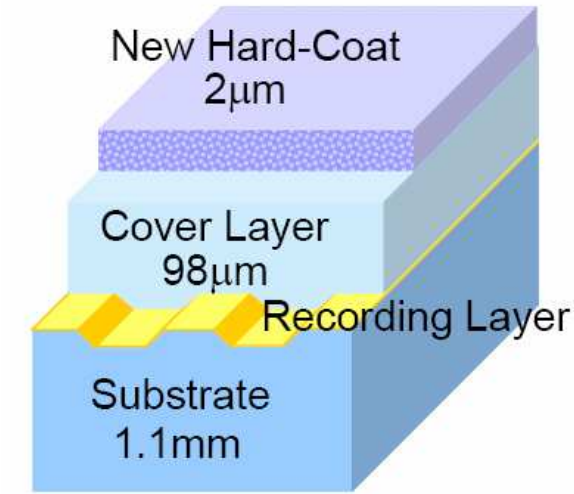
Blu-ray: Eigenschaften

Verkürzung der Wellenlänge auf
405 nm (blau)

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

Vergleich der Fokussierung

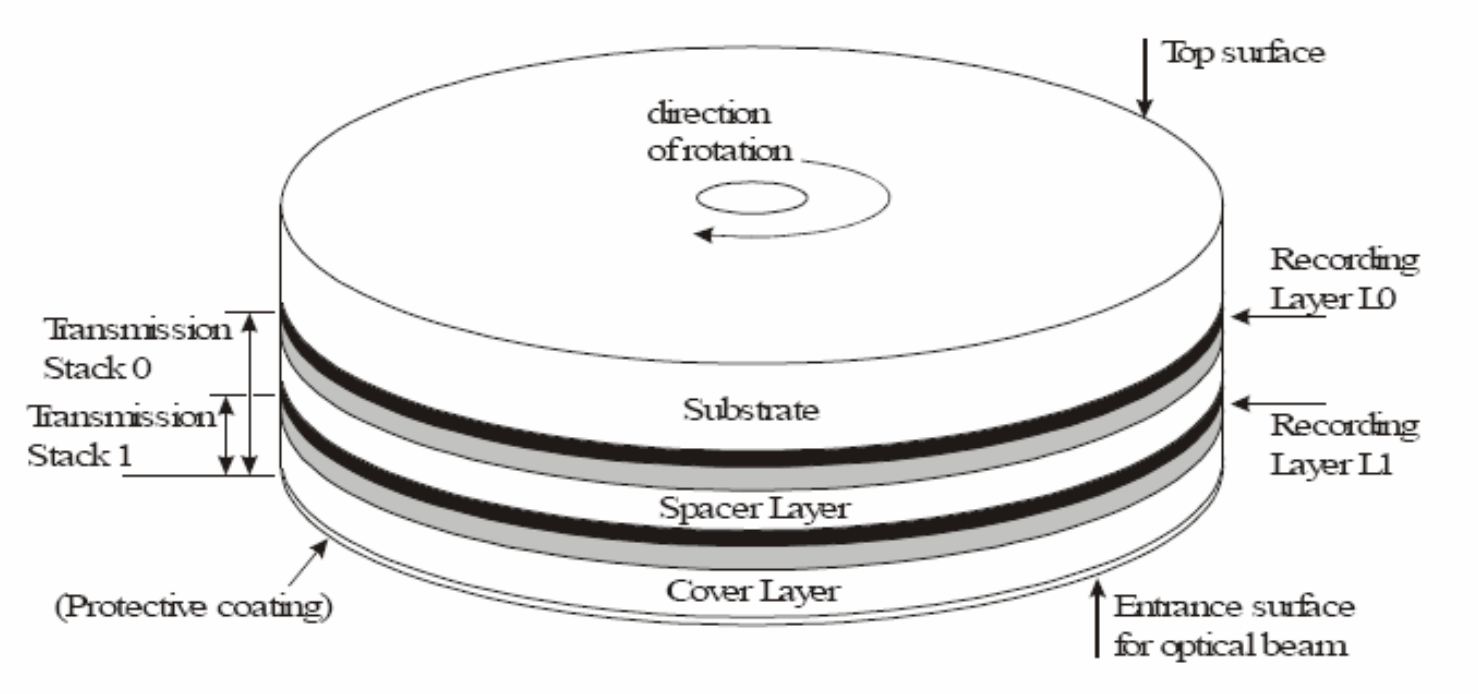




http://www.blu-raydisc.com/assets/downloadablefile/general_bluraydiscformat-12834.pdf

Blu-ray: Eigenschaften (2)

23.3/25/27 GB in einer Speicherebene
46.6/50/54 GB in zwei Speicherebenen
Experimentell 200 GB



http://www.blu-raydisc.com/assets/downloadablefile/general_bluraydiscformat-12834.pdf

HD-DVD

HD-DVD konkurriert mit BD,

HD-DVD hat größere Ähnlichkeiten mit der DVD:

- Schichtdicke bleibt bei 0,6 mm
- Wellenlänge aber 405 nm wie bei BD

Speicherkapazität geringer als bei BD:
15 GB in einer, 30 GB in zwei Lagen

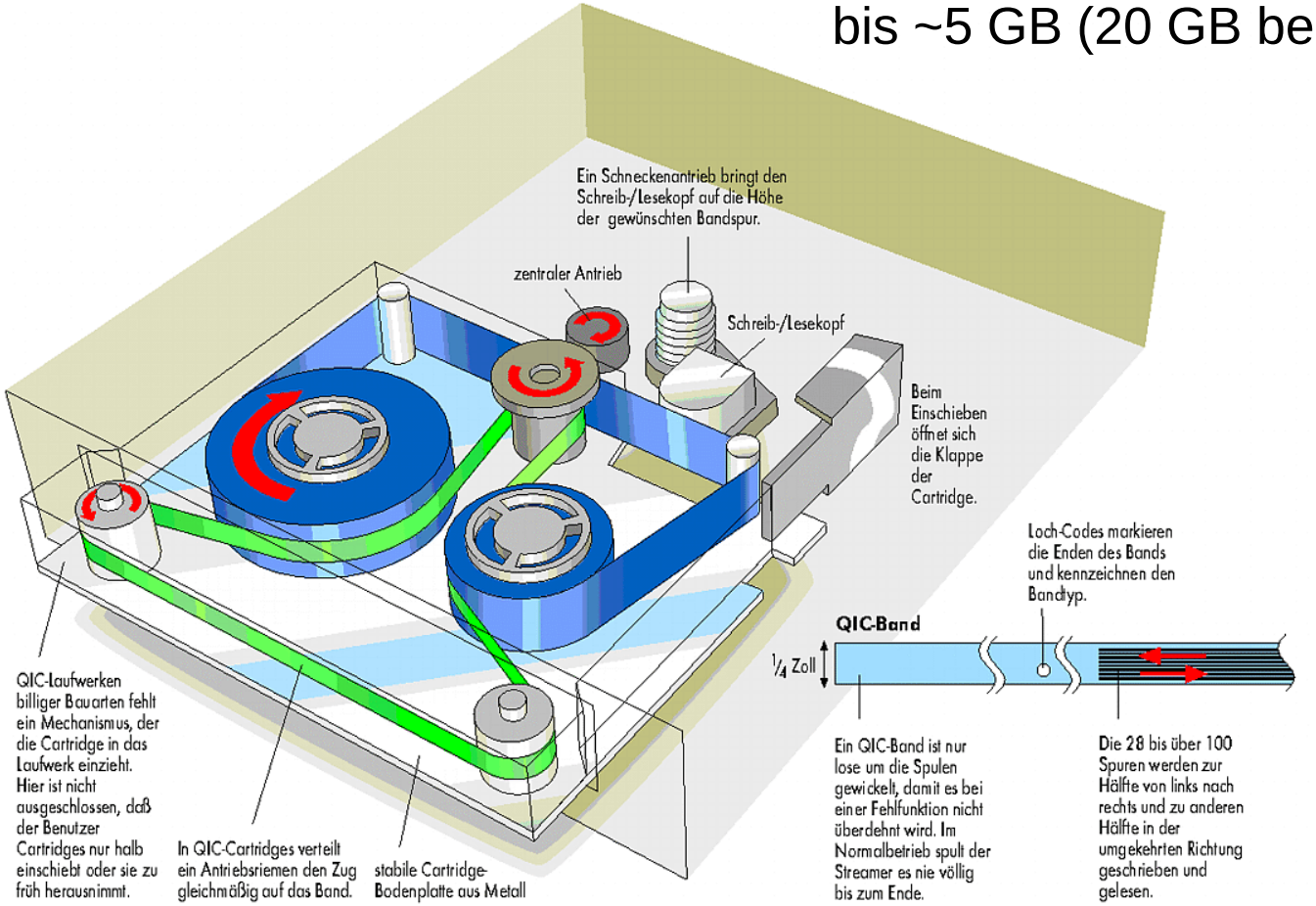
Evtl. preisgünstiger herzustellen als BD.

Unterstützt u.a. von Microsoft und Intel.

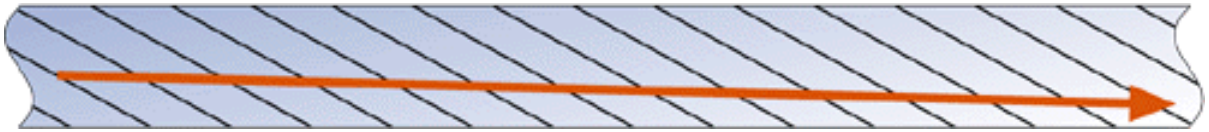
Erste Abspielgeräte in den USA ab April 2006 angeboten

QIC-Laufwerk (*Quarter inch tape*)

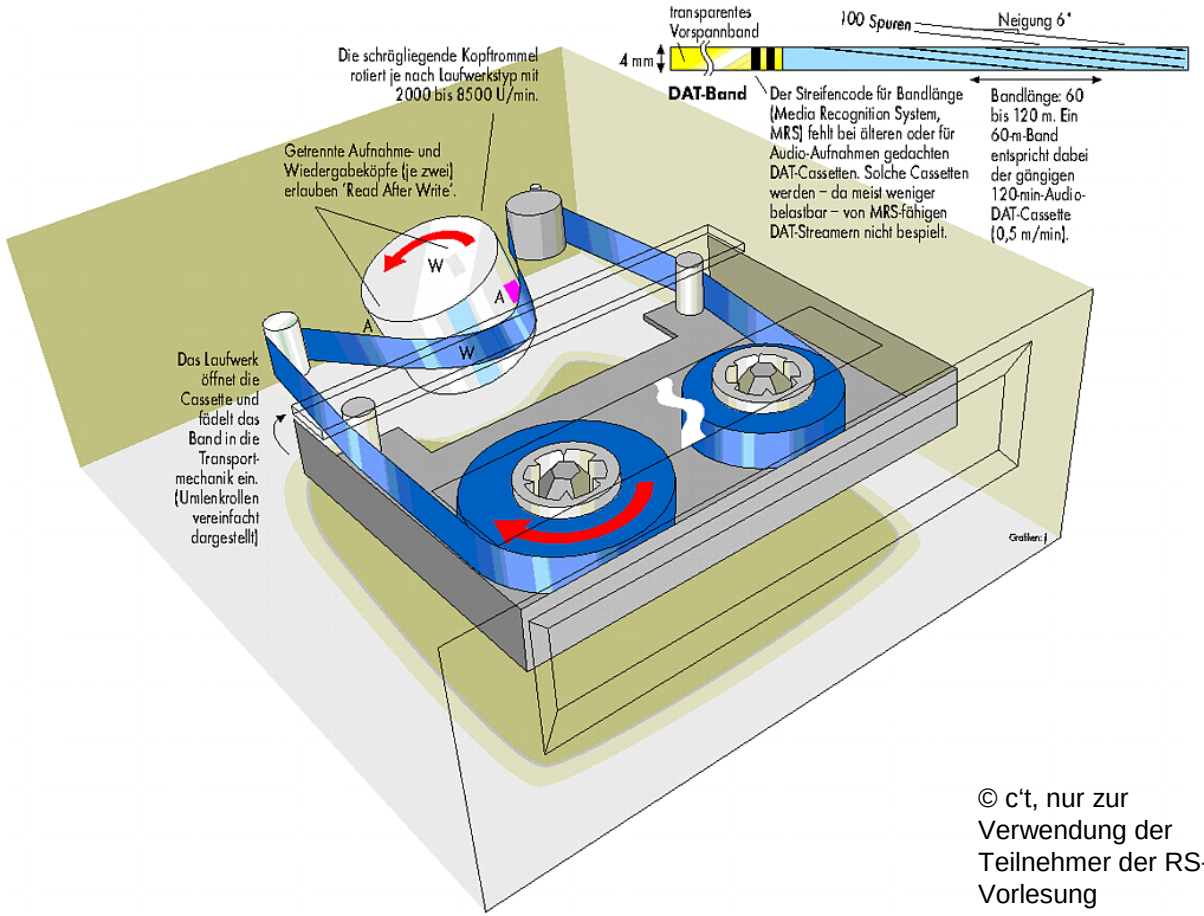
Horizontale Aufzeichnung in mehreren (bis zu 100) Spuren nacheinander, optimiert auf *streaming*-Betrieb mit hoher Bandgeschwindigkeit, bis ~5 GB (20 GB bei Weiterentwicklung Travan-5)



DAT-Laufwerk (*digital audio tape*)



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



Linear Tape Open (LTO) - 4

[www.wikipedia.de]

Linear Tape Open (LTO) - 4

Übersicht der Generationen								
Generation	Bandkapazität		maximale Transferrate		kompatible Laufwerke		Verfügbarkeit	
	ohne Kompression	mit Kompression	ohne Kompression	mit Kompression	Schreiben	Lesen	Lizenzen seit	Produkte seit
Ultrium 1	100 GB [3]	200 GB [3]	20 MB/s	40 MB/s [3]	Ultrium 2	Ultrium 2 Ultrium 3	1998 [3]	2000 [3]
Ultrium 2	200 GB [3]	400 GB [3]	40 MB/s	80 MB/s [3]	Ultrium 3	Ultrium 3 Ultrium 4	April 2002 [3]	Ende 2002 [3]
Ultrium 3	400 GB [3]	800 GB [3]	80 MB/s	160 MB/s [3]	Ultrium 4	Ultrium 4 Ultrium 5	Juli 2004 [3]	Ende 2004 [3]
Ultrium 4	800 GB [3]	1.600 GB [3]	120 MB/s	240 MB/s [3]	Ultrium 5	Ultrium 5 Ultrium 6	Ende 2006 [3]	Frühjahr 2007 [4]
Ultrium 5	1.600 GB	3.200 GB [3]	180 MB/s	360 MB/s [3]	Ultrium 6	Ultrium 6	Sommer 2009 [5]	-
Ultrium 6	3.200 GB	6.400 GB [3]	270 MB/s	540 MB/s [3]	-	-	-	-

[www.wikipedia.de]

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,
da nur das Speichermedium repliziert wird.

Hennessy/Patterson: Gegenwärtiger Kostenvorteil?

Schulte (IRB): Derzeit (2009): 45 € für Archivierung von 800 GB über LTO-4; bietet immer noch Kosten- und Archivierungsvorteile

Zusammenfassung

Nicht-flüchtige Speicher:

- Plattenspeicher sind traditionell preisgünstiges Speichermedium
- Flash-Speicher bieten neue Speicheroptionen, trotz begrenzter Anzahl von Schreiboperationen
- RAID-Speicher bieten benötigte Zuverlässigkeit
- Kapazität optischer Speicher steigt weiter
- Unterschiedliche Beurteilung der Preisvorteile von Bandspeichern