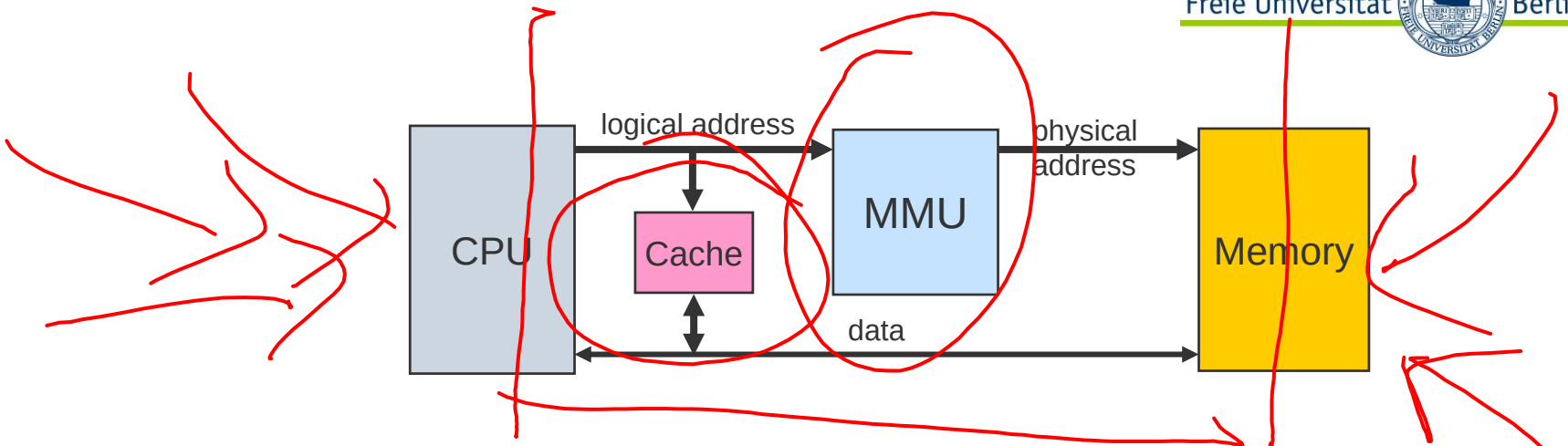
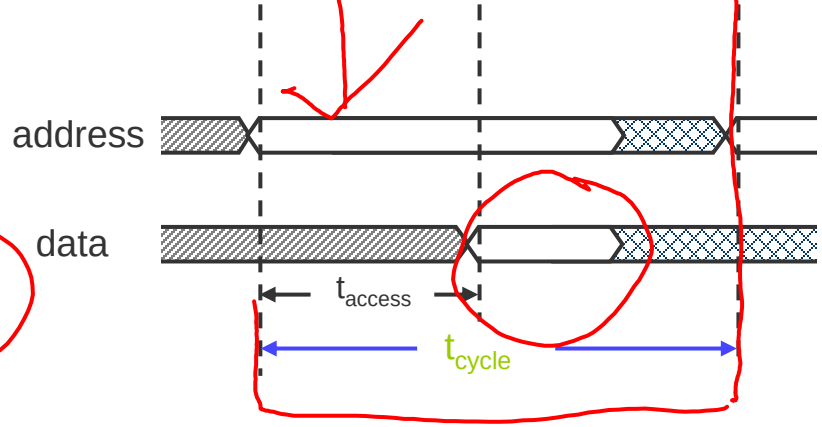
Prof. Dr.-Ing Jochen H. Schiller
Inst. of Computer Science
Freie Universität Berlin
Germany



TI II: Computer Architecture
Memories

- Hierarchy
- Types
- Physical & Virtual Memory
- Segmentation & Paging
- Caches



SPEICHERHIERARCHIE

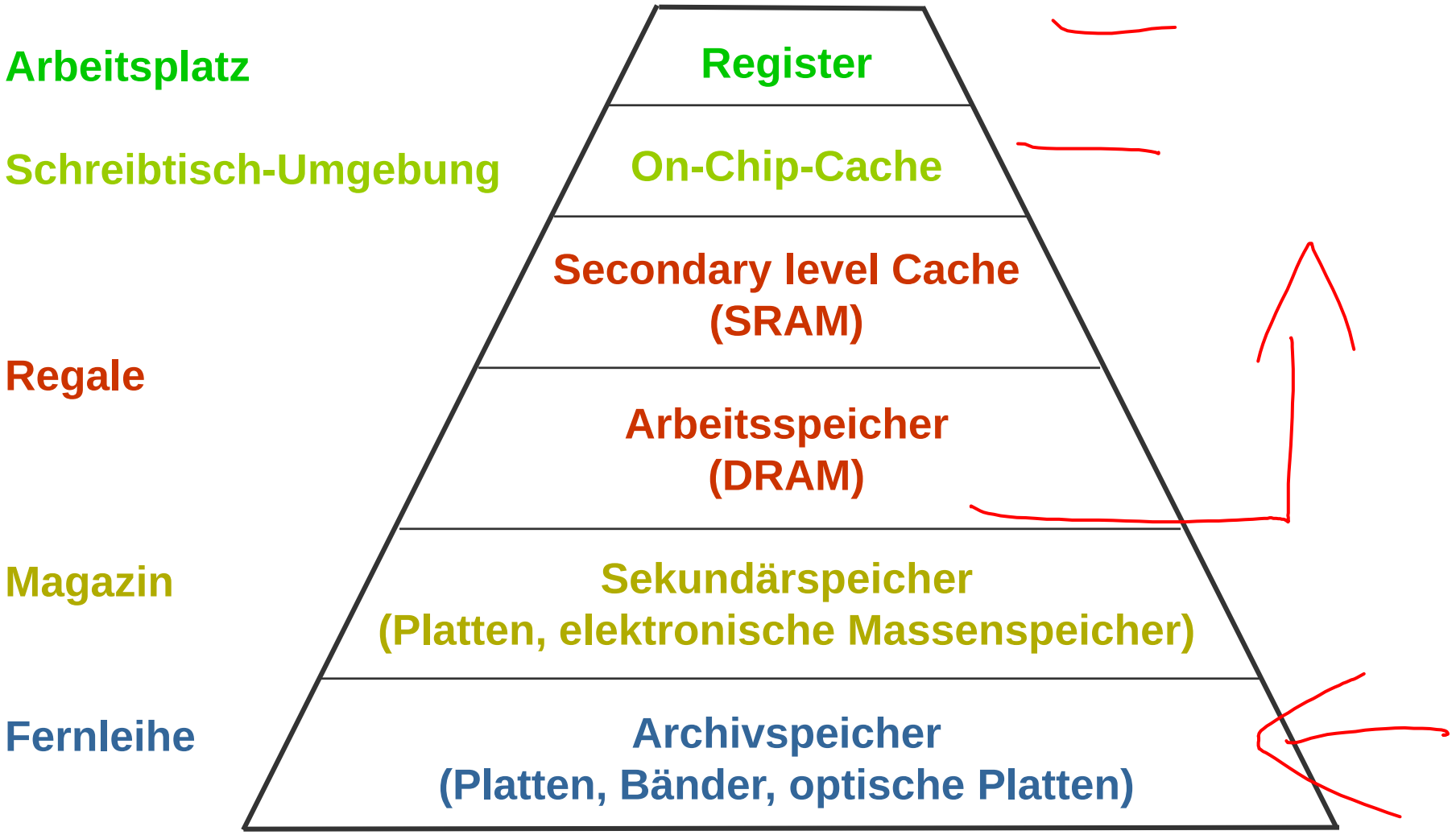
Speicherhierarchie

Ein technologisch einheitlicher Speicher mit kurzer Zugriffszeit und großer Kapazität ist aus Kostengründen i.Allg. nicht realisierbar

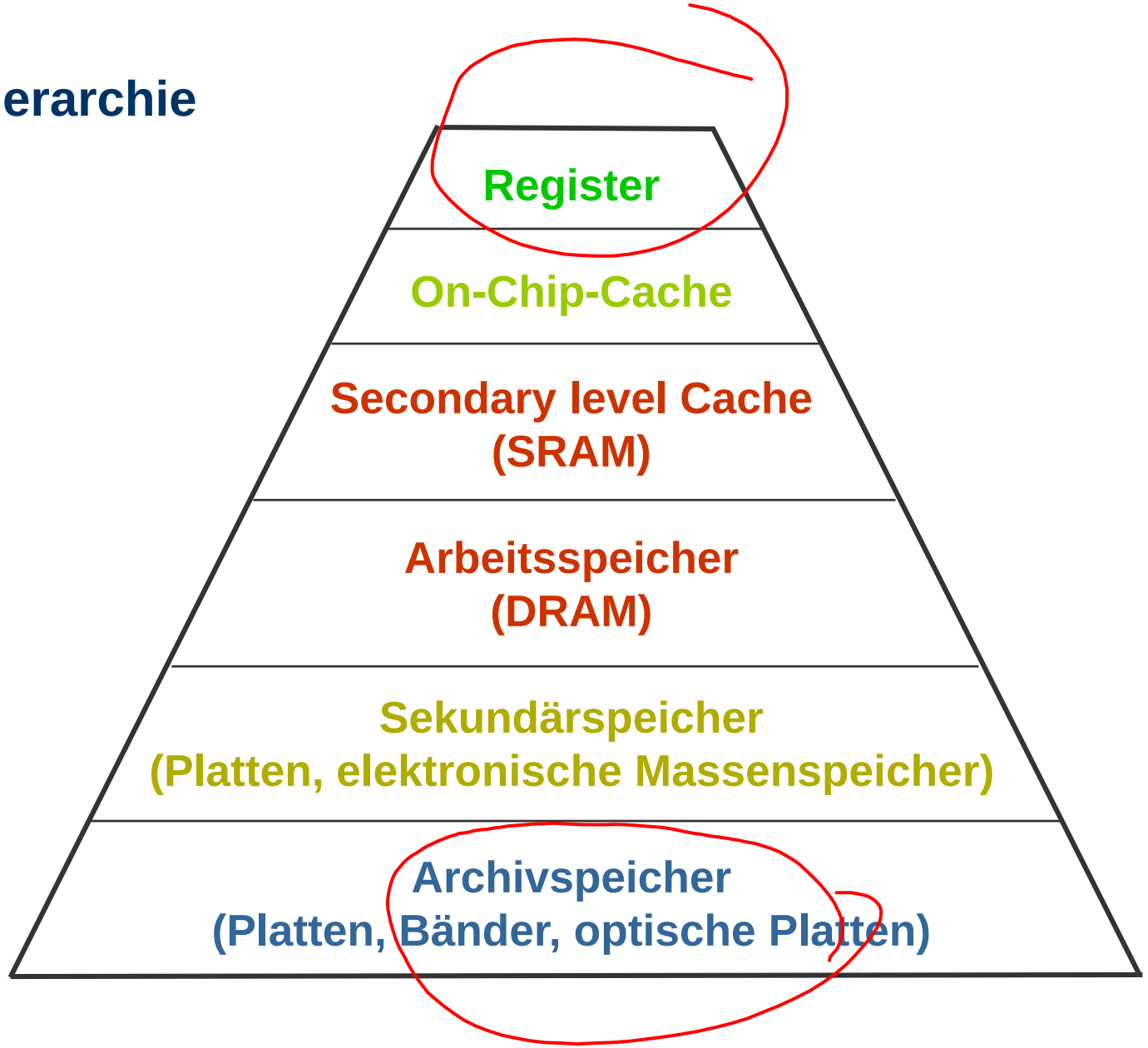
Lösung

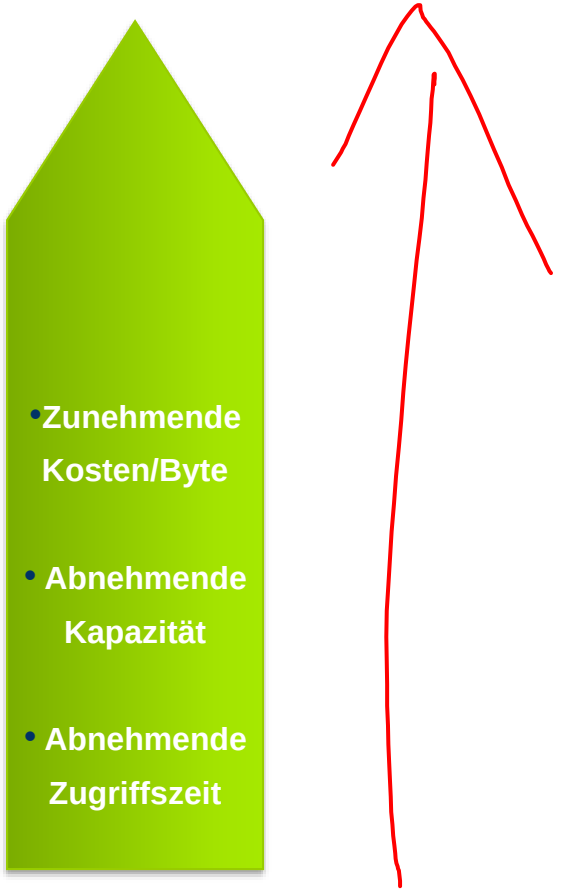
- Schichtenweise Anordnung verschiedener Speicher und Verschiebung der Information zwischen den Schichten (Speicherhierarchie)
- **Cache-Speicher**: Kurze Zugriffszeiten ➔ Beschleunigung des Prozessorzugriffs
- **Virtueller Speicher**: Vergrößerung des tatsächlich vorhandenen Hauptspeichers (z.B. bei gleichzeitiger Bearbeitung mehrerer Prozesse)

Speicherhierarchie



Speicherhierarchie





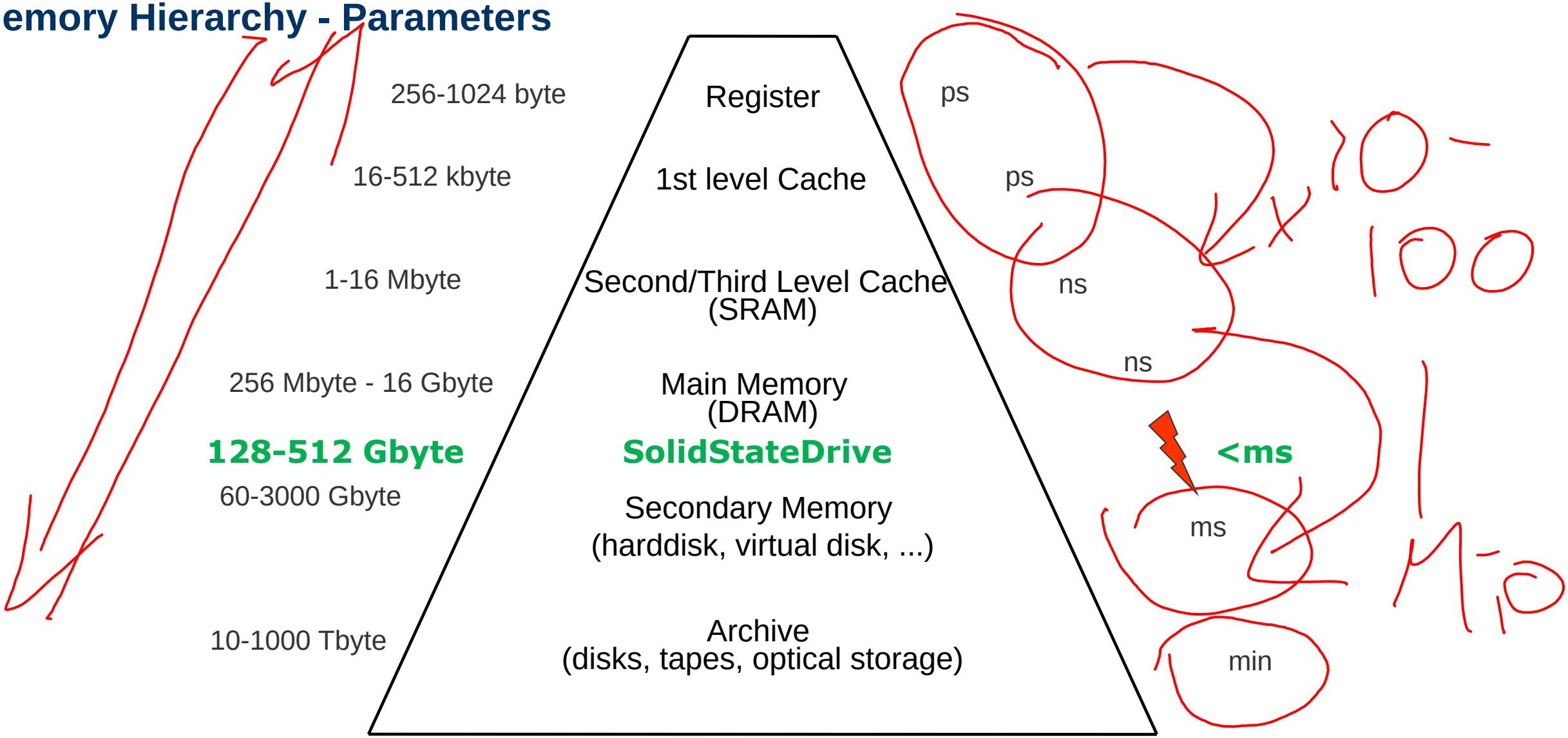
Speicherhierarchie

Wirkung: wie ein großer und schneller Speicher, wenn

- **Lokalitätsverhalten** der Programmverarbeitung
- Umlagerung der Information rechtzeitig (Umlagerungsstrategien)
- Inhomogenität des Speichersystems für Benutzer nicht sichtbar ist (Virtueller Speicher)

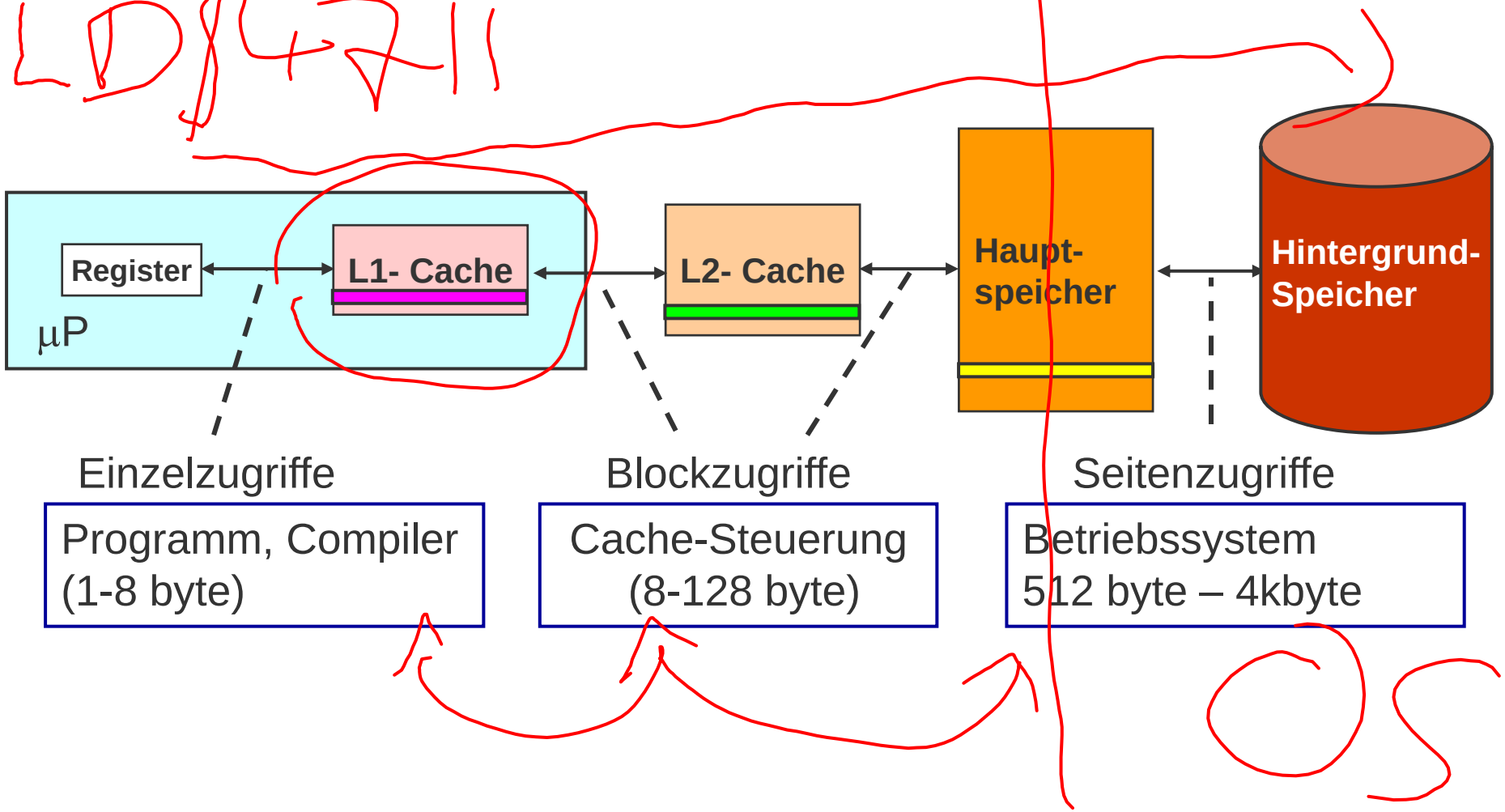
Leistungsfähigkeit der Hierarchie ist bestimmt durch die Eigenschaften der Speichertechnologien (Zugriffsart, Zugriffszeiten, ...), Adressierung der Speicherplätze und Organisation des Betriebs

Memory Hierarchy - Parameters



Speicherhierarchie

Daten werden nur zwischen aufeinander folgenden Ebenen der Speicherhierarchie kopiert.

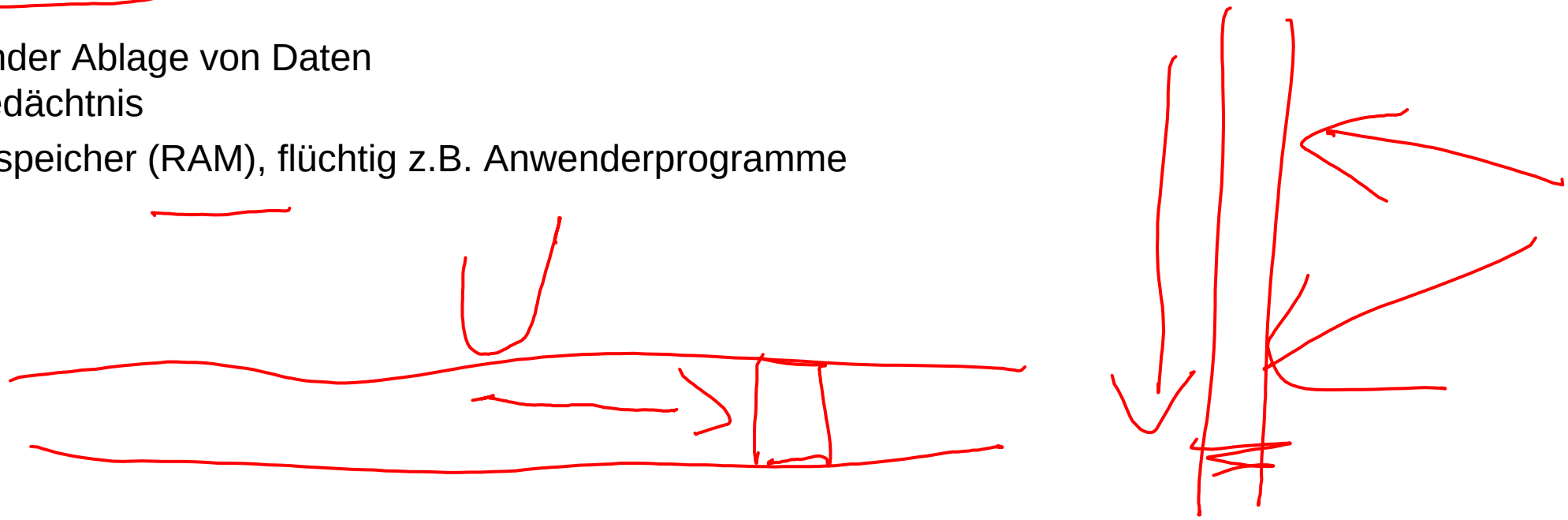


HAUPTSPEICHER

Einleitung

Man unterscheidet zwischen:

- permanenter Ablage von Daten
  ➔ Langzeitgedächtnis
- Festwertspeicher (ROM), nicht flüchtig z.B. Betriebssystemkern, Firmware, Systemtabellen
- vorübergehender Ablage von Daten
  ➔ Kurzzeitgedächtnis
- Schreib/Lesespeicher (RAM), flüchtig z.B. Anwenderprogramme



Begriffe

Speicherelement
- 1 Bit Speicher

Speicherzelle (-platz, -stelle)
- Feste Anzahl von Speicherelementen, die durch eine einzige Adresse ausgewählt werden, z.B. 8, 16, 32 bit

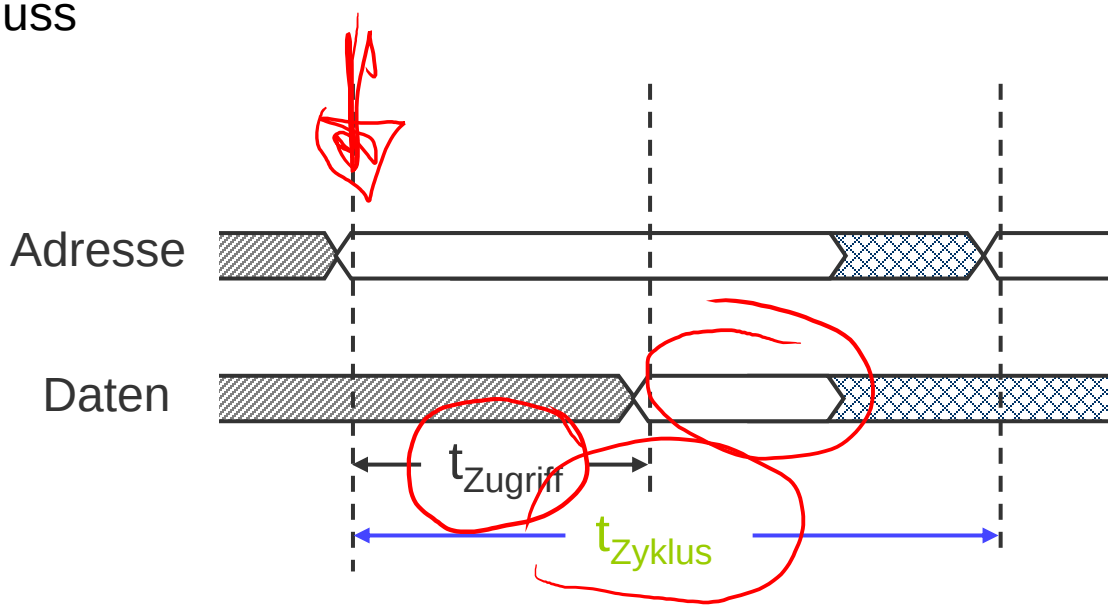
Speicherwort
- Maximale Anzahl von Speicherelementen, die in einem Buszyklus zwischen µP und Speicher übertragen werden können ➔ Speicherwortbreite = Datenbusbreite

Wahlfreier Zugriff
- Jede Speicherzelle kann direkt angesprochen werden (ohne vorher andere Zellen ansprechen zu müssen)
- Die Selektion erfolgt über einen Adressdekoder. Die Adresse wird in einen 1-aus-n-Code umgeformt.

Begriffe

Größen zur Charakterisierung der Arbeitsgeschwindigkeit
eines Speicherbausteins

- Zugriffszeit (access time)
  - maximale Zeitdauer, die vom Anlegen einer Adresse an den Speicher bis zur Ausgabe der gewünschten Daten vergeht
- Zykluszeit (cycle time)
  - minimale Zeitdauer, die zwischen zwei hintereinander folgenden Aufschaltungen von Adressen an den Speicher vergehen muss



Zugriffszeit / Zykluszeit

Die Zykluszeit kann erheblich länger als die Zugriffszeit sein!

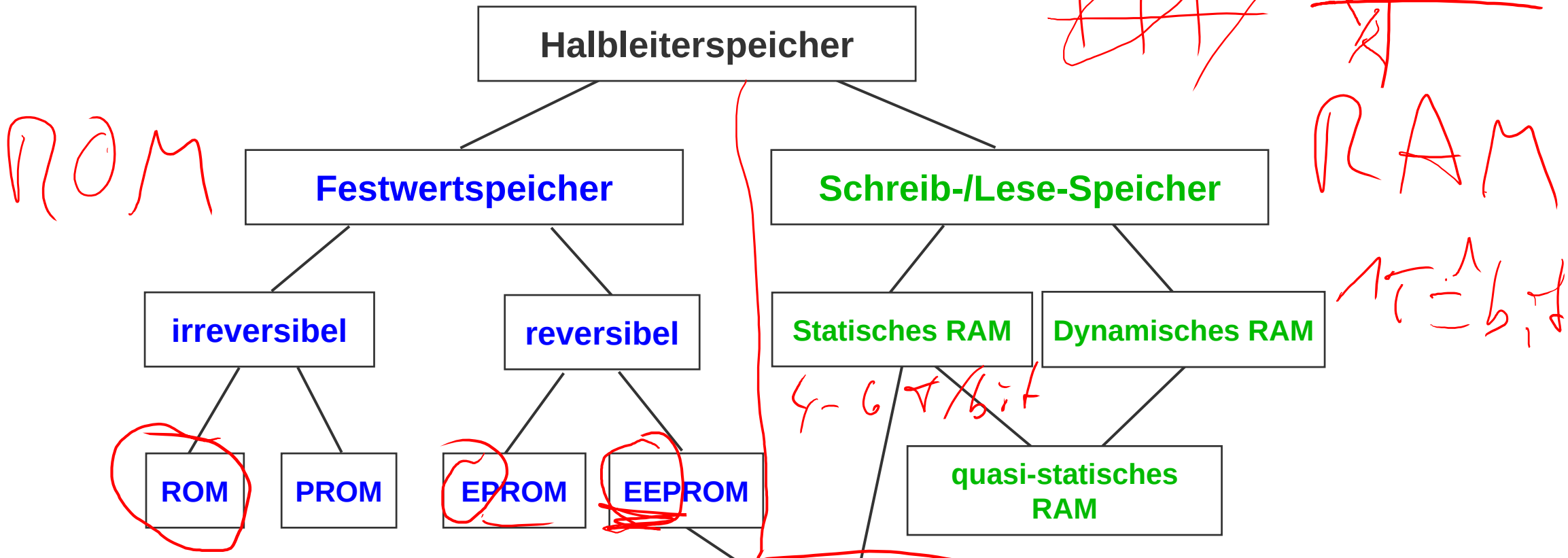
Gründe

- Speicherzelle muss sich nach einem Zugriff “erholen”
- Bei einigen Speicherarten wird die Information durch das Auslesen zerstört und muss erst wieder eingeschrieben werden (refresh)

Idealfall: Zykluszeit = Zugriffszeit

Realität: meist Zykluszeit > Zugriffszeit (bis zu 80%)

Klassifizierung von Halbleiterspeichern



RAM	Random Access Memory
ROM	Read-Only Memory
PROM	Programmable ROM
EPROM	Erasable PROM
EEPROM	Electrically EPROM

DRAM-Speichertypen

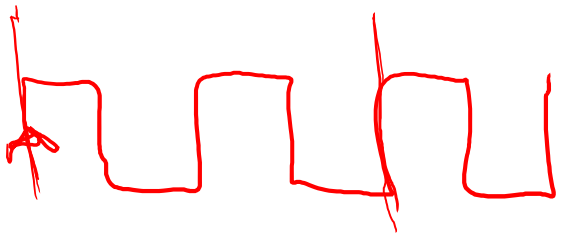
Revolutionäre Schritte im Speicherdesign bleiben aus. Durch ein bewährtes und über die Jahre verfeinertes Prinzip behauptet sich DRAM als Arbeitsspeicher Nr. 1

12. August 1981

- Erster IBM PC (Model 5150 mit Intel 8086 Prozessor),
  16 KByte Arbeitsspeicher auf 8 einzelnen Chips, die jeweils eine Kapazität von 16 KBit hatten.

Heute

- Diese RAM-Menge von damals verschwindet mehrfach im L1-Cache des Prozessors



SDRAM

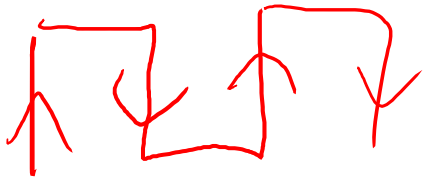
SDRAM-Technologie hat sich (durch intensive Unterstützung von Intel) schnell durchgesetzt und beherrscht heute den Speichermarkt.

- Alle Ein- und Ausgangssignale sind synchron zum Systemtakt.
- Prozessor, Chipsatz und Speicher kommunizieren über ein Bussystem, das synchron mit der gleichen Frequenz getaktet ist.

Intern sind SDRAMs aus zwei unabhängigen Speicherbänken aufgebaut (auch bis zu 4 Speicherbänke).

Nach dem Anlegen der Zeilen- und Spaltenadresse, generiert die Speichersteuerung die nachfolgenden Adressen und führt einen alternierenden und überlappenden Zugriff auf die beiden Speicherbänke selbstständig aus

DDR-DRAM



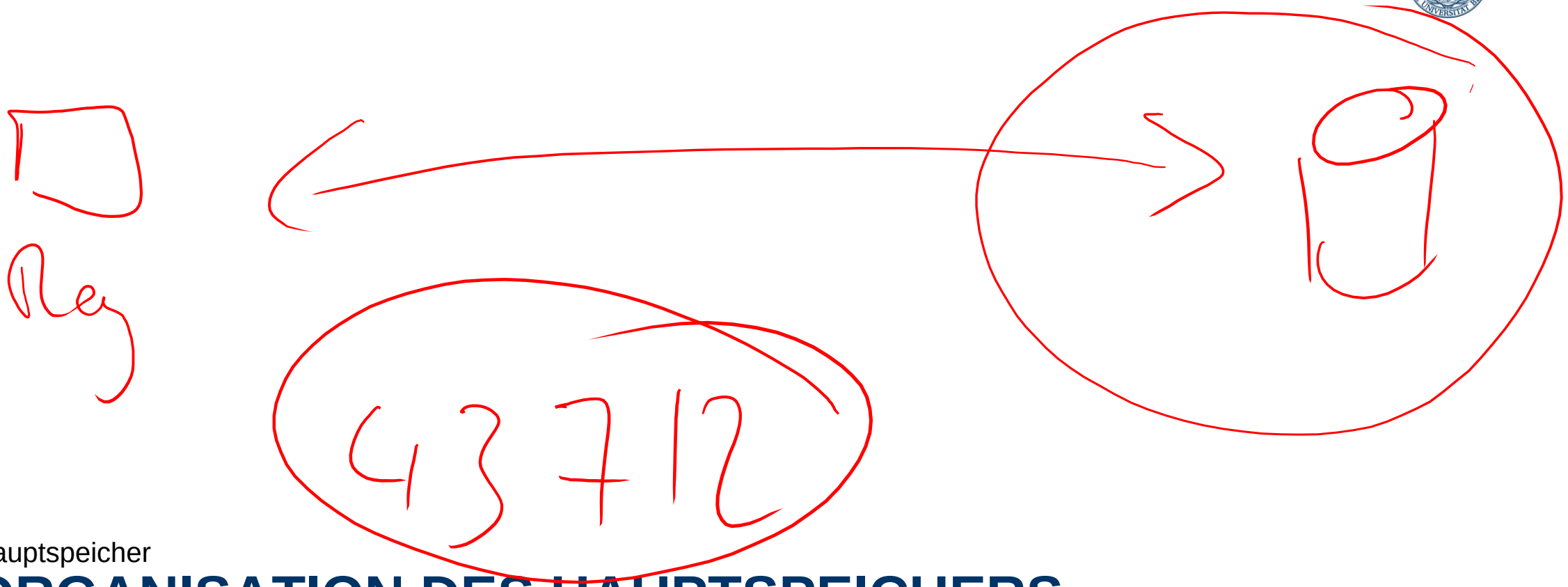
Nächste Stufe der SDRAM-Entwicklung (SDRAM II)

Bestehen intern aus vier unabhängigen Speicherbänken, die parallel „Instruktionen“ bearbeiten können

Prinzip der DDR-DRAMs

- Erweiterung der Bandbreite durch Nutzung beider Taktflanken. Daten werden bei steigender und fallender Taktflanke übertragen
  ➔ doppelter Datendurchsatz

Laufzeitverzögerungen sind sehr kritisch, deshalb wird zur Synchronisation nicht nur der Systemtakt, sondern auch ein bidirektionales Strobe-Signal (DQS) benutzt.



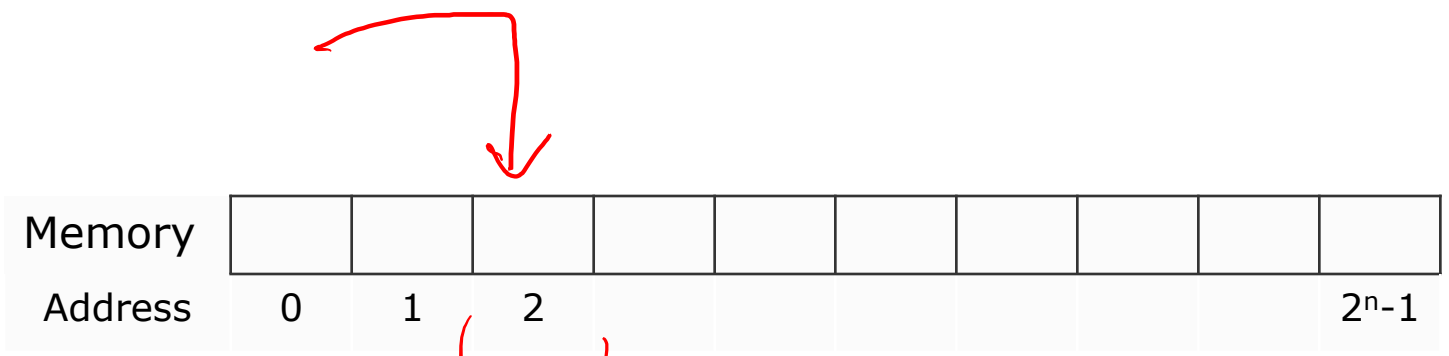
Hauptspeicher

ORGANISATION DES HAUPTSPEICHERS

Organisation des Arbeitsspeichers

Arbeitsspeicher

- Lineare Liste von Speicherworten
- Aufgebaut aus Speicherbausteinen
- Zugriffszeit hängt allein von der Art der verwendeten Speicherbausteine ab
- Die Breite des Arbeitsspeichers entspricht i.A. der Breite des Datenbus (8, 16, 32, 64 Bit). Dies entspricht der maximalen Informationsmenge, auf die in einem Buszyklus zugegriffen werden kann.

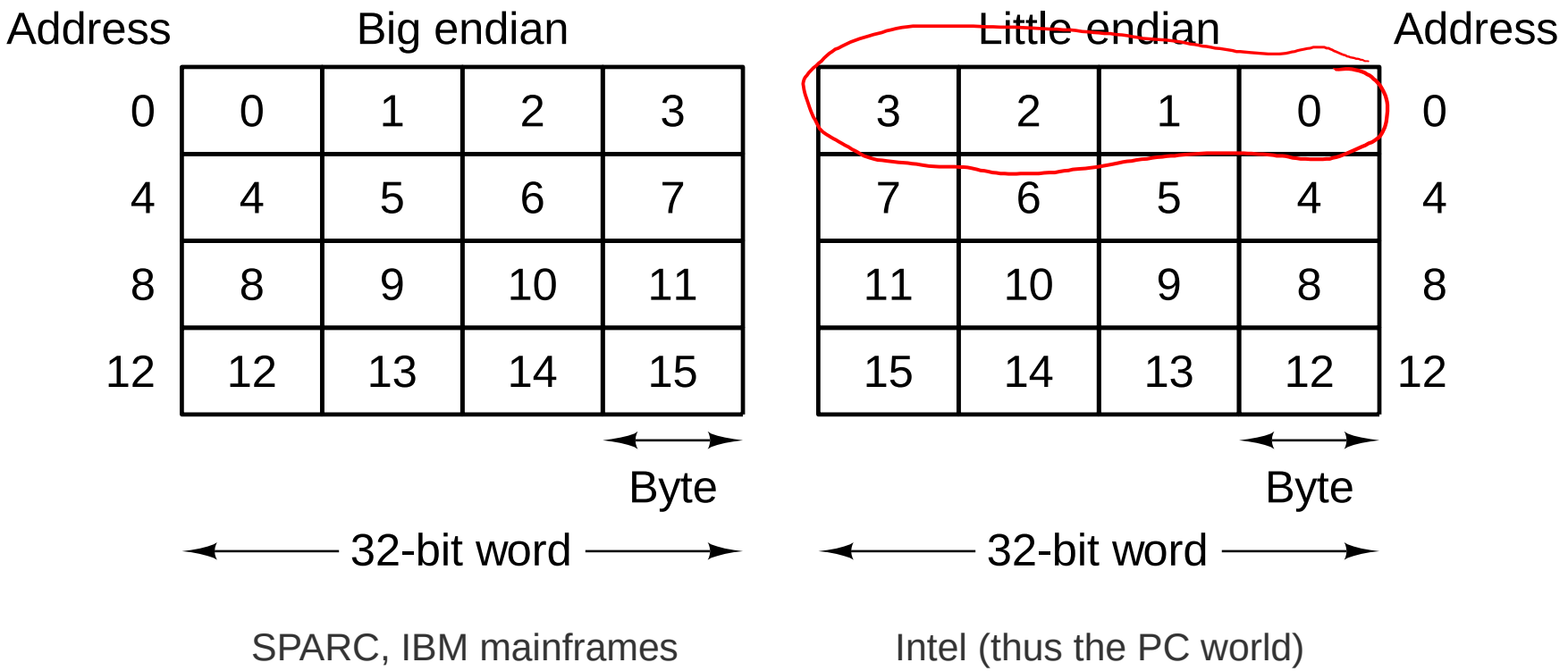


History: Bits per storage cell

… or why a byte/smallest addressable unit has not always equaled 8 bits

Computer	bit/cell
Burroughs B1700	1
IBM PC	8
DEC PDP-8	12
IBM 1130	16
DEC PDP-15	18
XDS 940	24
Electrologica X8	27
XDS Sigma 9	32
Honeywell 6180	36
CDC 3600	48
CDC Cyber	60

Byte Ordering
Big endian vs. little endian



Organisation des Arbeitsspeichers

Bei Prozessoren mit einer Datenbusbreite > 8 bit kann meist immer noch auf einzelne Bytes zugegriffen werden

➡Speicherkapazität wird auch bei breiteren Organisationsformen in Bytes angegeben

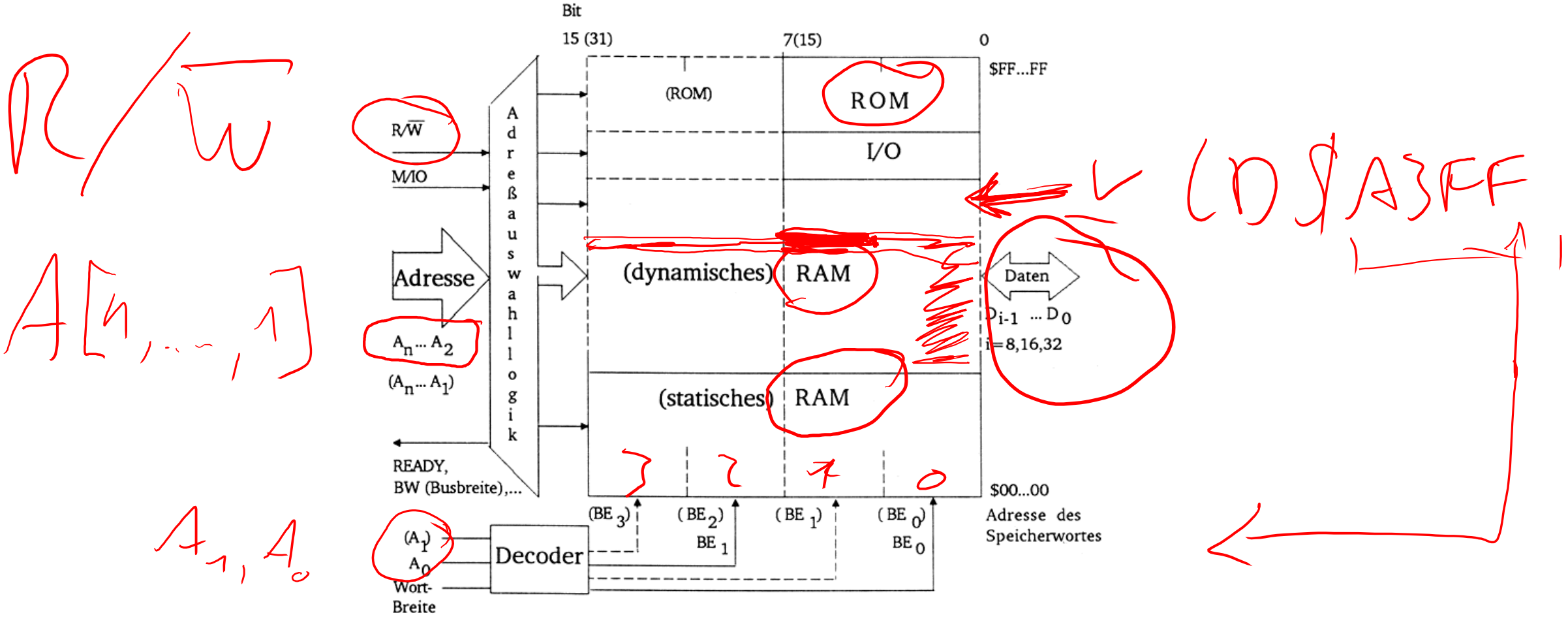
Die maximale Kapazität des Arbeitsspeichers ist durch die Breite des Adressbusses gegeben

Typische maximale Speicherkapazitäten:

- 8-bit-Prozessoren mit 16-bit-Adressbus: 64 kbyte
- 16-bit-Prozessoren mit 24-bit-Adressbus: 16 MByte
- 32-bit-Prozessoren mit 32-bit-Adressbus: 4 GByte
- 64-bit-Prozessoren mit 64-bit-Adressbus: 16 EByte

Speicher-Belegungsplan (memory map)

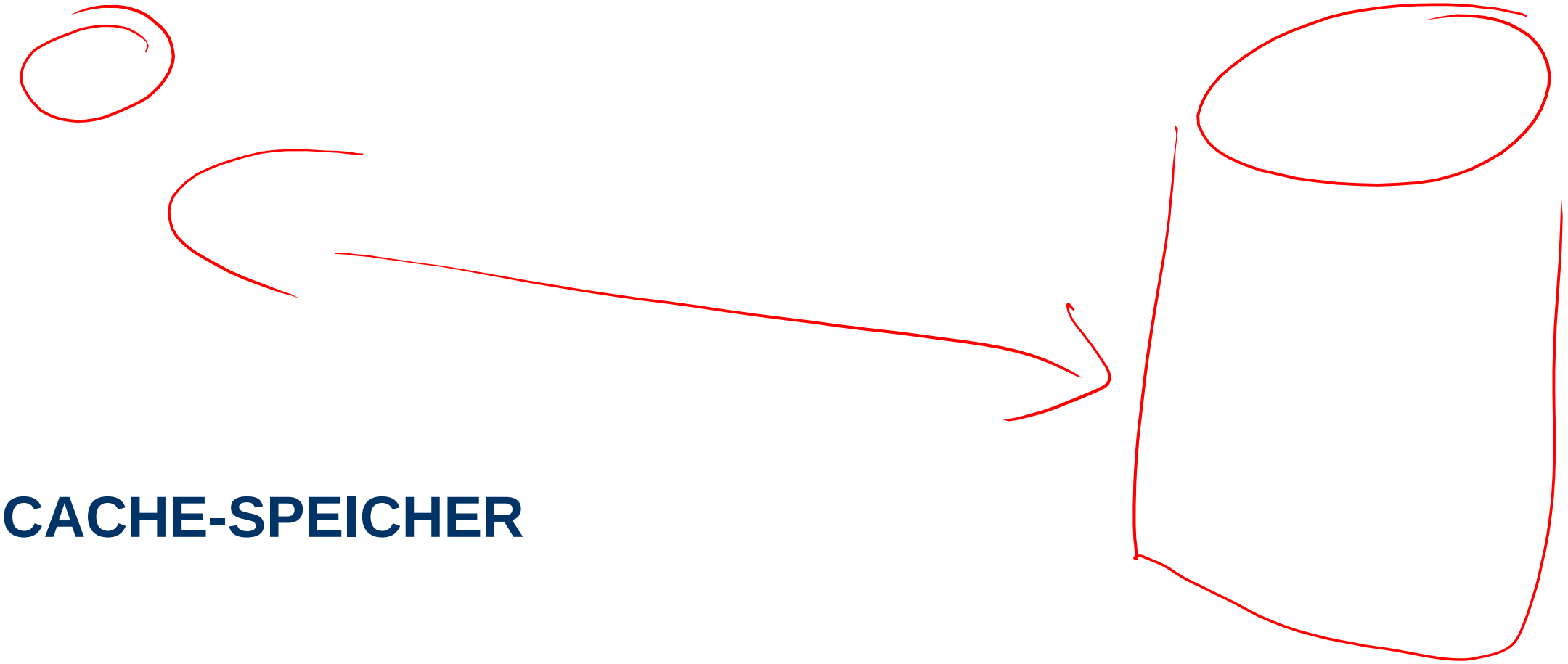
Gibt an, welche Speicherbausteine für welche Bereiche des Arbeitsspeichers verwendet wurden



Speicher-Belegungsplan (memory map)

Im Beispiel

- obere Adressen
  - ROM, z.B. für nicht-flüchtige Teile des Betriebsystems (Bootstrap, BIOS)
- dann
  - I/O-Bereich (Prozessor mit memory-mapped I/O)
- Rest
  - RAM, meist dynamische RAM-Bausteine, da diese große Kapazität besitzen und billig sind
  - Nachteil: Sie sind auch langsam
  - Aus diesem Grund werden manchmal in kleinen Speicherbereichen auch statische RAMs eingesetzt, auf die ohne Wartezyklen zugegriffen werden kann



CACHE-SPEICHER

Cache-Speicher

Problem

- Die Buszykluszeit moderner Prozessoren ist erheblich kürzer als die Zykluszeit preiswerter, großer DRAM-Bausteine
  - dies zwingt zum Einfügen von Wartezyklen.
- SRAM-Bausteine hingegen, die ohne Wartezyklen betrieben werden können, sind jedoch klein, teuer und besitzen eine höhere Verlustleistung.
  - nur relativ kleine Speicher können so aufgebaut werden.

Lösung des Problems

- zwischen den Prozessor und den relativ langsamen, aber billigen Arbeitsspeicher aus DRAM-Bausteinen legt man einen kleinen, schnellen Speicher aus SRAM-Bausteinen, den sogenannten Cache-Speicher.

Unter einem Cache-Speicher versteht man allgemein einen kleinen, schnellen Pufferspeicher, der vor einen langsamen, größeren Speicher geschaltet wird, um dessen Zugriffszeit zu verbessern.

Cache-Speicher

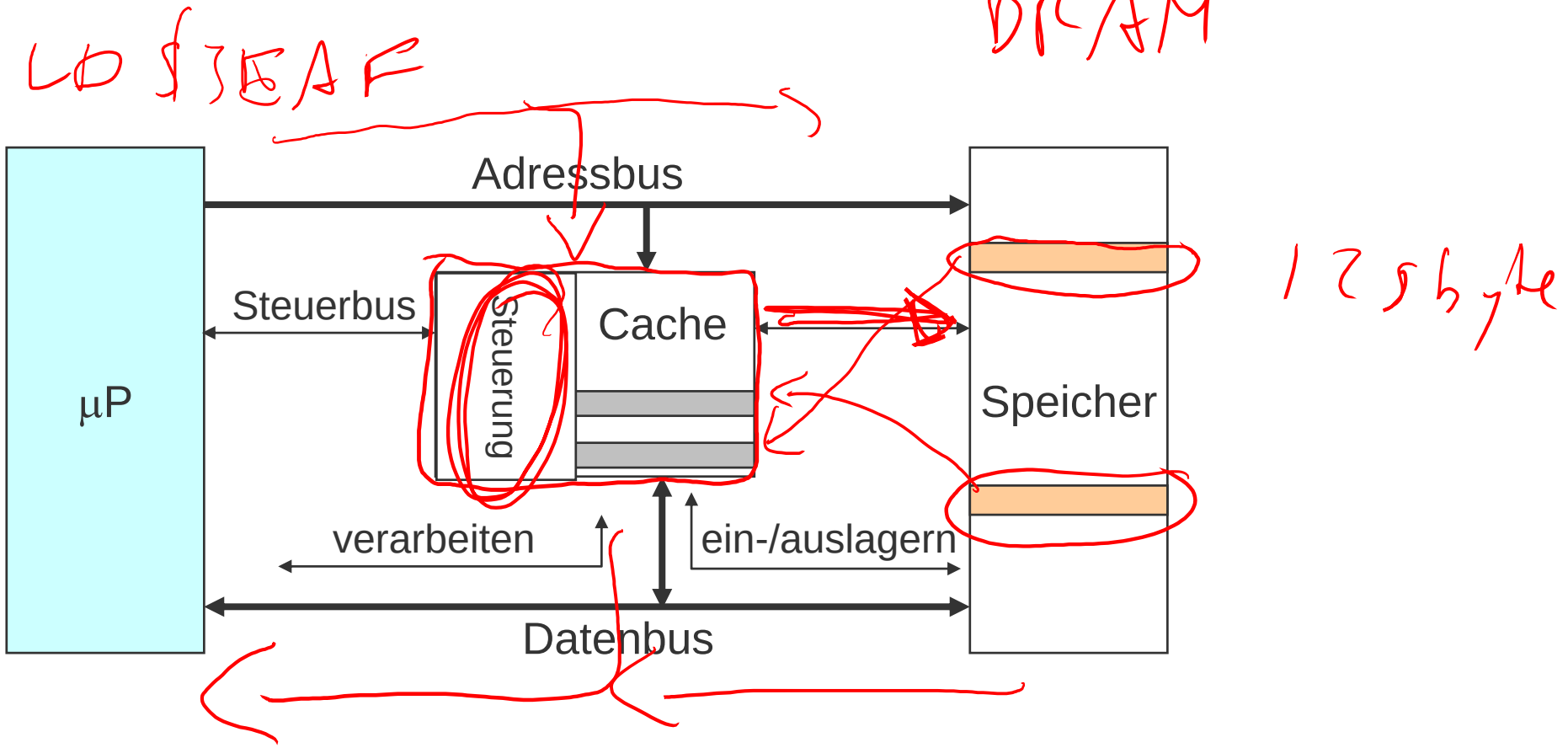
Anwendungsbeispiele

- Verbesserung der Zugriffszeit des Hauptspeichers eines Prozessors durch einen Cache zur Vermeidung von Wartezyklen (CPU-Cache, Befehls- und Daten-Cache)
- Verbesserung der Zugriffszeit von Plattenspeichern durch einen Cache (Plattencache)

Hier soll im Wesentlichen der erste Fall näher betrachtet werden

Cache-Speicher

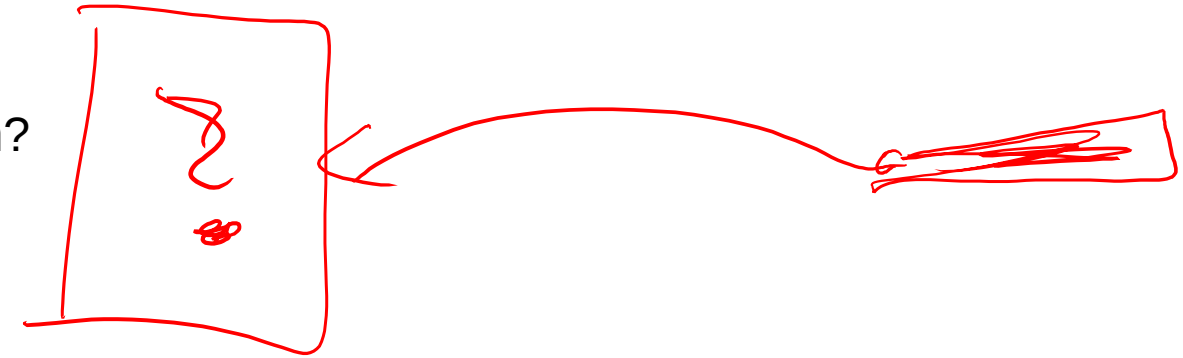
Unter einem **CPU-Cache-Speicher** versteht man einen kleinen, schnellen Pufferspeicher, in dem Kopien derjenigen Teile des Arbeitsspeichers bereitgehalten werden, auf die aller Wahrscheinlichkeit nach von der CPU als nächstes zugegriffen wird.



Fragen für den Cache-Entwurf

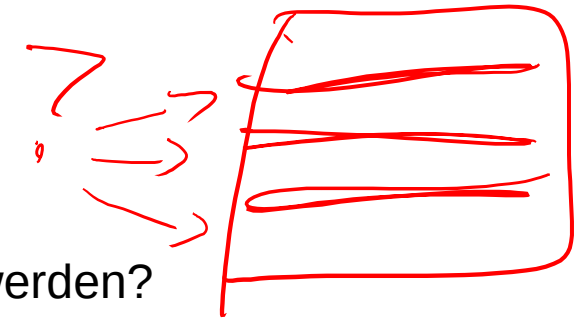
Wohin kann ein Block (cache line) abgebildet werden?

- Block-Abbildungsstrategie
- Vollassoziativ, Satz-Assoziativ, Direct-Mapped



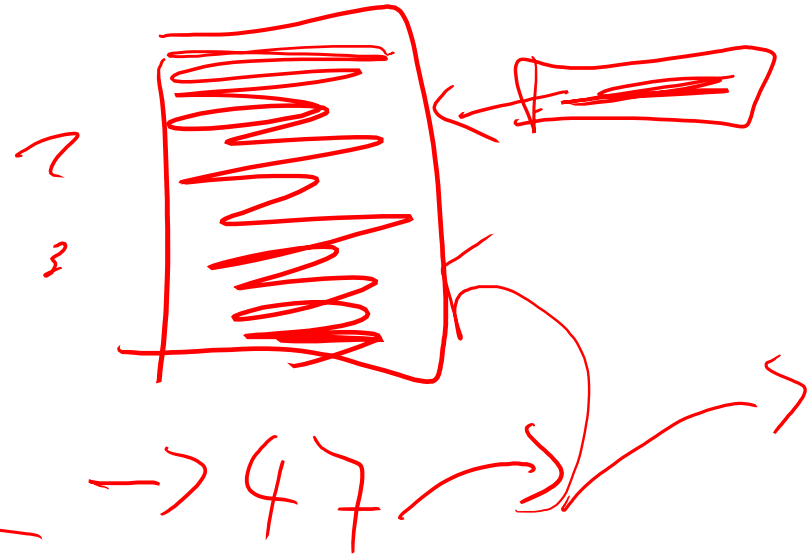
Wie kann ein Block gefunden werden?

- Block-Identifikation
- Tag/Block



Welcher Block soll beim einem Miss ersetzt werden?

- Block-Ersetzungsstrategie
- Random, FIFO, LRU



Was passiert bei einem Schreibzugriff?

- Schreib-Strategie
- Write Back oder Write Through (mit Write Buffer)

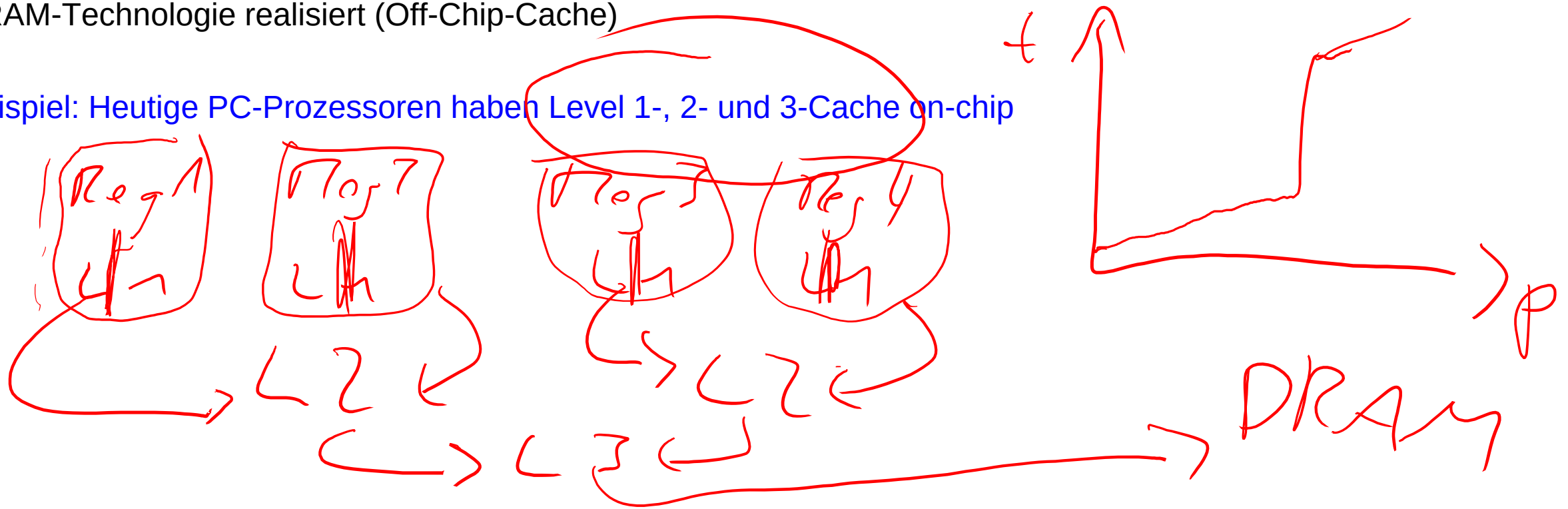
Cache-Speicher

Dieser Cache-Speicher ist zwischen CPU und Arbeitsspeicher plaziert.

Auf den Cache-Speicher soll der Prozessor fast so schnell wie auf seine Register zugreifen können.

Er ist entweder direkt auf dem Prozessorchip integriert (on-chip-Cache) oder in der schnellsten und teuersten SRAM-Technologie realisiert (Off-Chip-Cache)

Beispiel: Heutige PC-Prozessoren haben Level 1-, 2- und 3-Cache on-chip

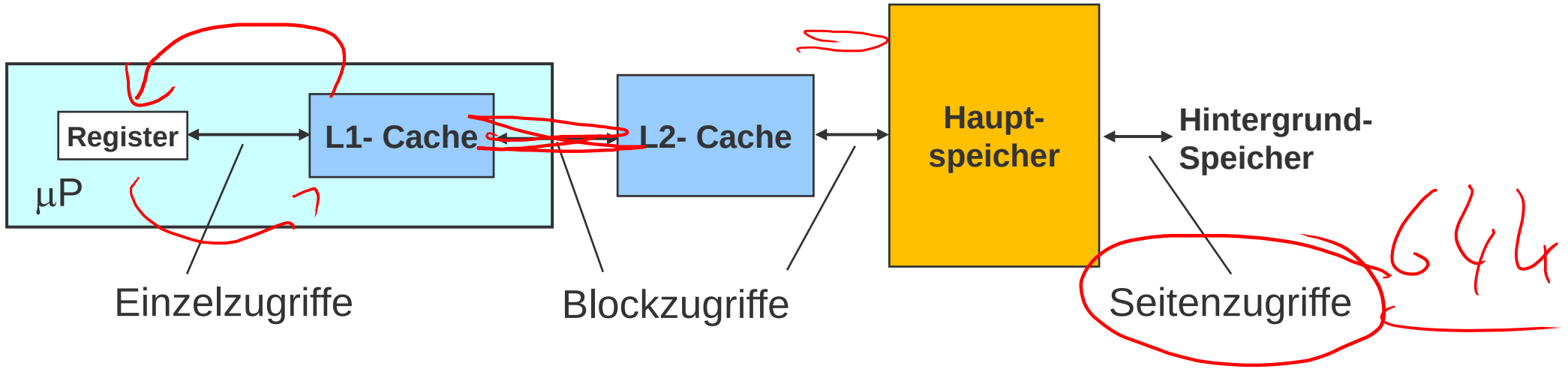


Cache-Speicher

On-Chip-Cache
- Integriert auf dem Prozessorchip
- Sehr kurze Zugriffszeiten (wie die der prozessorinternen Register)
- Aus technologischen Gründen begrenzte Kapazität

Off-Chip-Cache
- prozessorextern

Beispiel (Achtung: auch L2, L3, … können on-chip sein!)



Cache-Speicher

Die Cachespeicherverwaltung (Steuerung) sorgt dafür, dass die am häufigsten benötigten Daten sich im Cache befinden

➡ Die CPU kann mit hoher Wahrscheinlichkeit das nächste Datum aus dem schnellen Cache und muss nicht aus dem langsamen Arbeitsspeicher holen.

Dies wird erreicht, indem automatisch durch eine Hardware-Steuerung (Cache-Controller) alle Daten in den Cache kopiert werden, auf die der Prozessor zugreift.

Weniger häufig benötigte Daten werden nach verschiedenen Strategien aus dem Cache verdrängt.

Cache-Speicher

Ein CPU-Cache-Speicher bezieht seine Effizienz im Wesentlichen aus der **Lokalitätseigenschaft** von Programmen (locality of reference), d.h. es werden bestimmte Speicherzellen bevorzugt und wiederholt angesprochen (z.B. Programmschleifen)

Zeitliche Lokalität

- Die Information, die in naher Zukunft angesprochen wird, ist mit großer Wahrscheinlichkeit schon früher einmal angesprochen worden (Schleifen).

Örtliche Lokalität

- Ein zukunftiger Zugriff wird mit großer Wahrscheinlichkeit in der Nähe des bisherigen Zugriffs liegen.

Funktionsweise eines Caches

Lesezugriffe

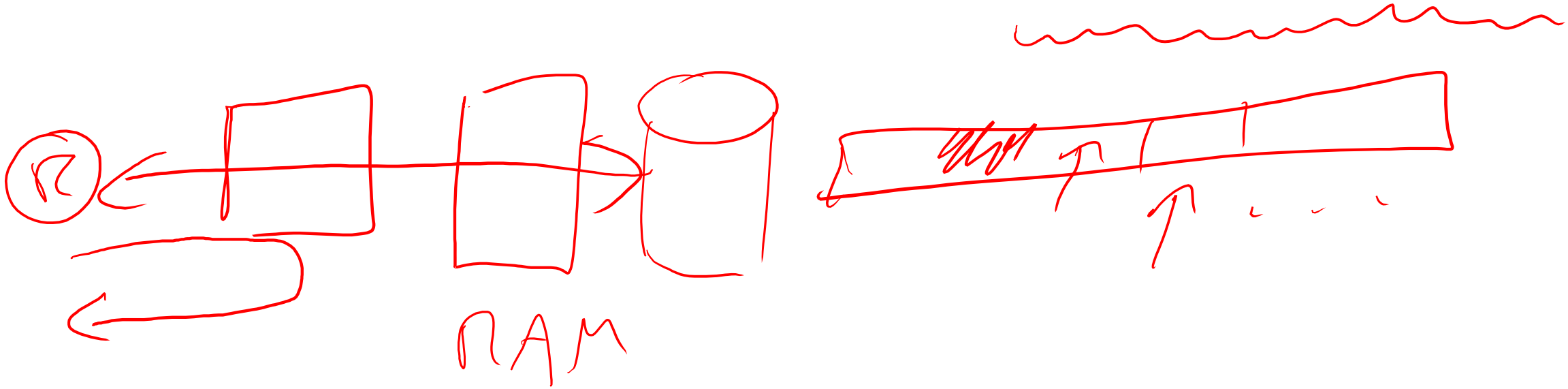
- Vor jedem Lesezugriff prüft der µP, ob das Datum im Cache steht.

Wenn ja: Treffer (**Cache Hit**)

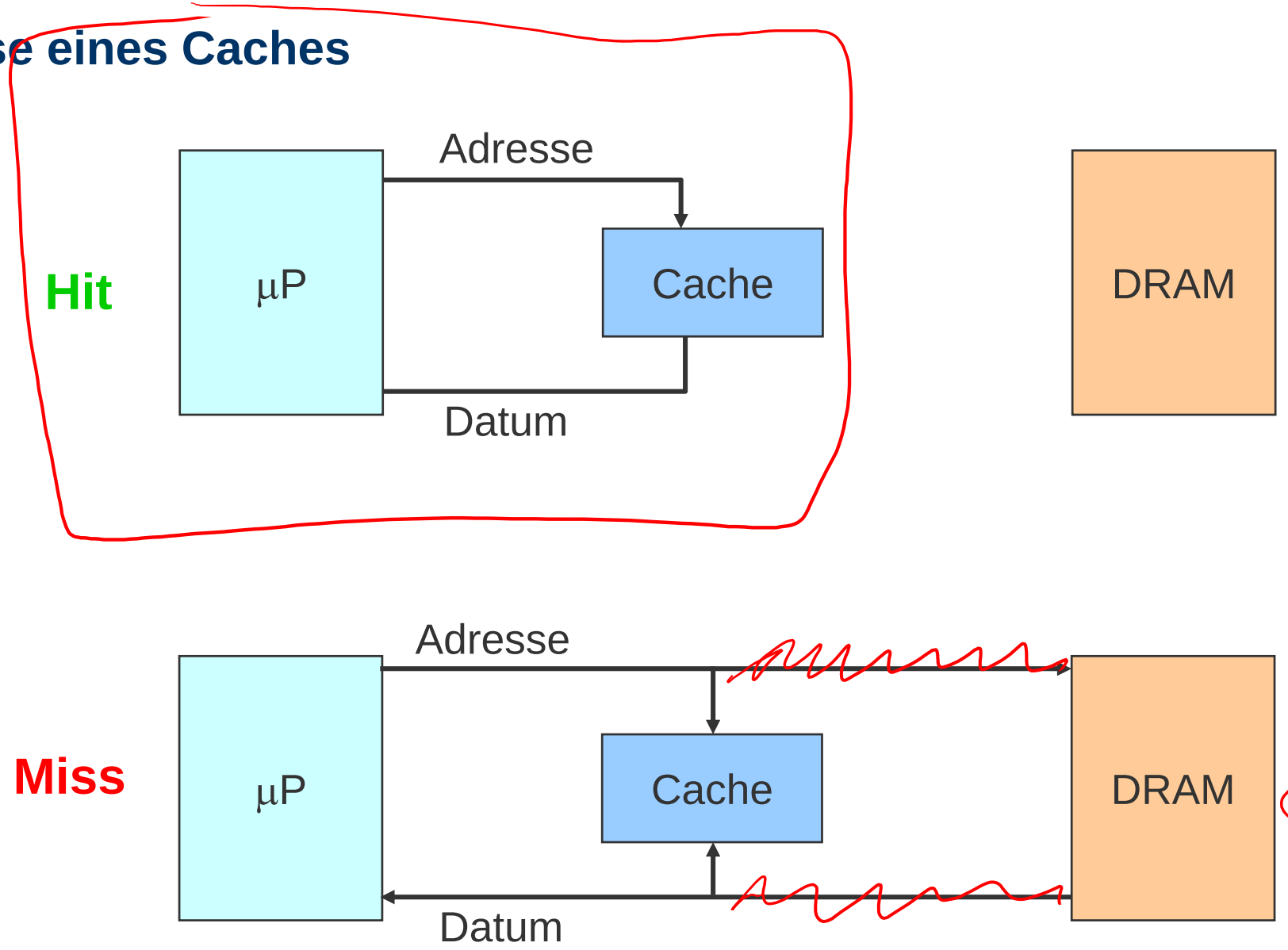
- Das Datum kann ohne Wartezyklen dem Cache entnommen werden.

Wenn nein: kein Treffer (**Cache Miss**)

- Das Datum wird mit Wartezyklen aus dem Arbeitsspeicher gelesen und gleichzeitig in den Cache eingefügt.



Funktionsweise eines Caches



Begriffe

Die **Hit-Rate** bezeichnet die **Trefferquote** im Cache:

HitRate = Anzahl Treffer / Anzahl Zugriffe

Die **mittlere Zugriffszeit** berechnet sich annähernd wie folgt

$$t_{Access} = (HitRate) \times t_{Hit} + (1 - HitRate) \times t_{Miss}$$

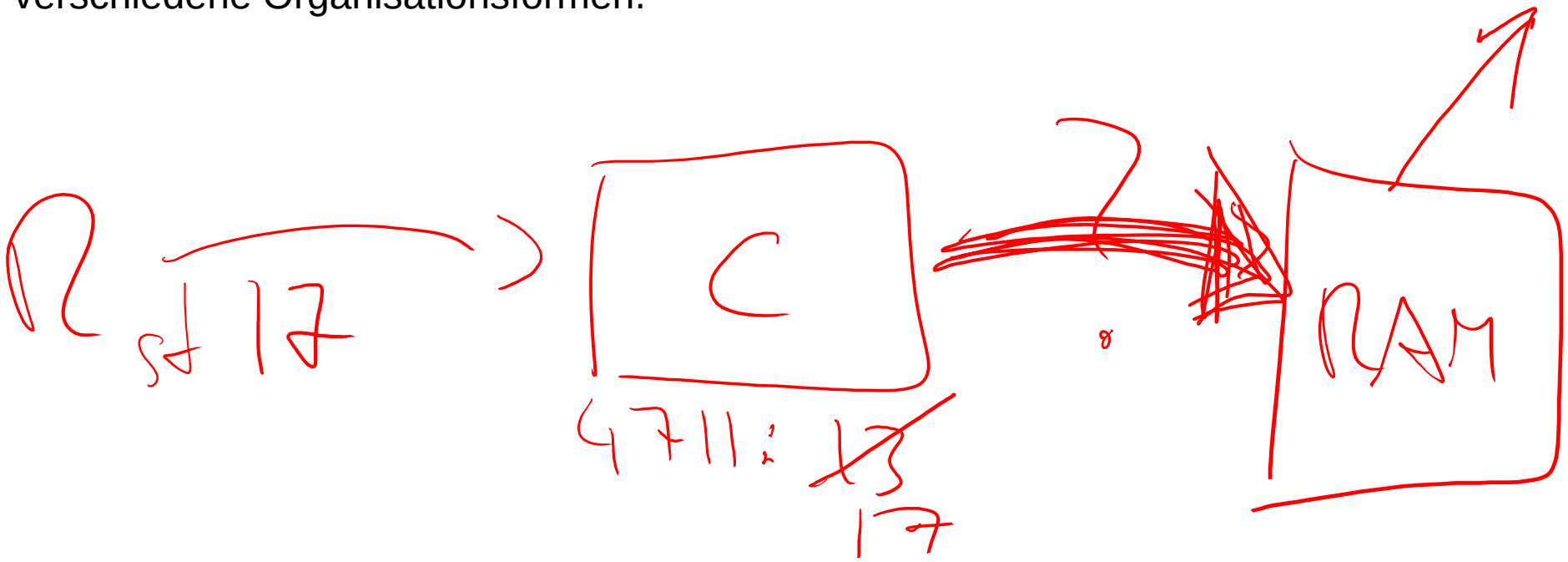
- t_{Hit} : Zugriffszeit des Caches
- t_{Miss}: Zugriffszeit ohne den Cache

Funktionsweise eines Caches

Schreibzugriffe

- Liegt beim Schreiben ein Cache-Miss vor, wird das Datum sowohl in den Arbeitsspeicher wie in den Cache geschrieben.

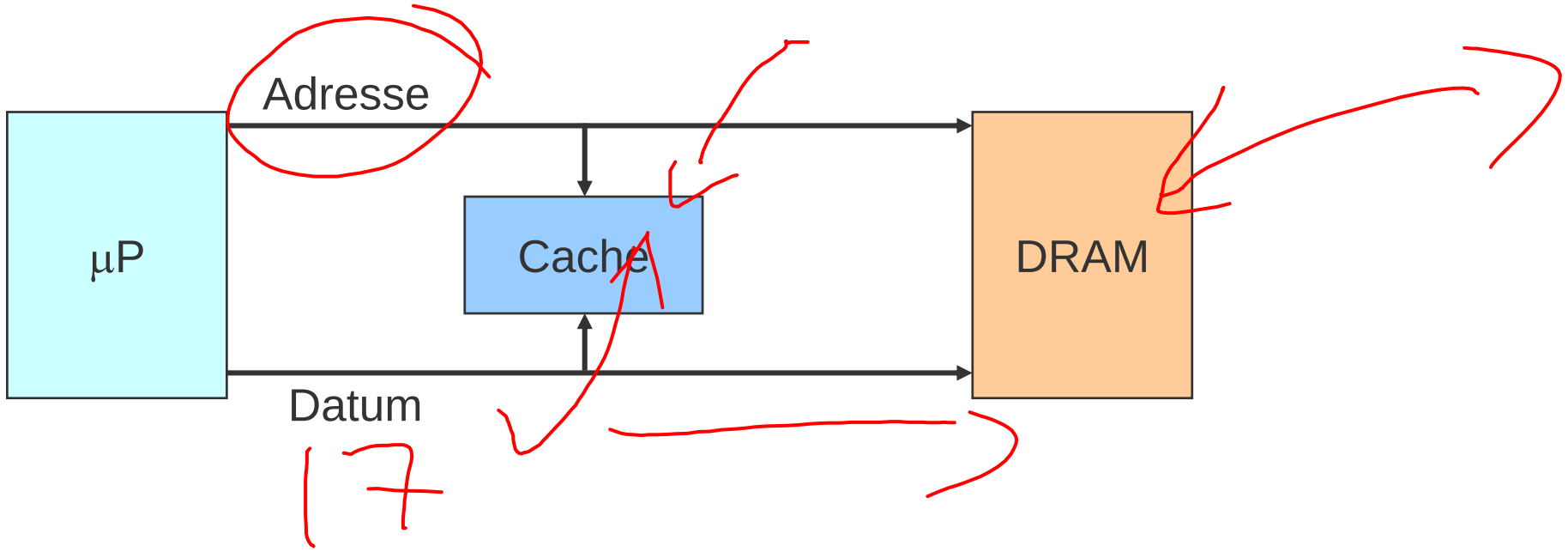
- Liegt beim Schreiben jedoch ein Cache-Hit vor, d.h. ein im Cache stehendes Datum wird durch den Prozessor verändert, so existieren verschiedene Organisationsformen.



Schreibzugriffe: write through

Durchschreibverfahren (write through policy)

- Ein Datum wird von der CPU immer gleichzeitig in den Cache- und in den Arbeitsspeicher geschrieben.



Vorteil

- Garantierte Konsistenz zwischen Cache- und Arbeitsspeicher.

Nachteil

- Schreibzugriffe benötigen immer die langsame Zykluszeit des Hauptspeichers und belasten den Systembus.

Schreibzugriffe: buffered write through

Gepuffertes Durchschreibverfahren (buffered write through policy)

- Variante des Durchschreibverfahrens
- Zur Milderung des Nachteils beim Durchschreibverfahren
  wird ein kleiner Schreib-Puffer verwendet, der die zu schreibenden Daten temporär aufnimmt.
- Diese Daten werden dann automatisch vom Cache-Controller in den Hauptspeicher übertragen, während der Prozessor parallel dazu mit weiteren Operationen fortfährt.

Schreibzugriffe: write back

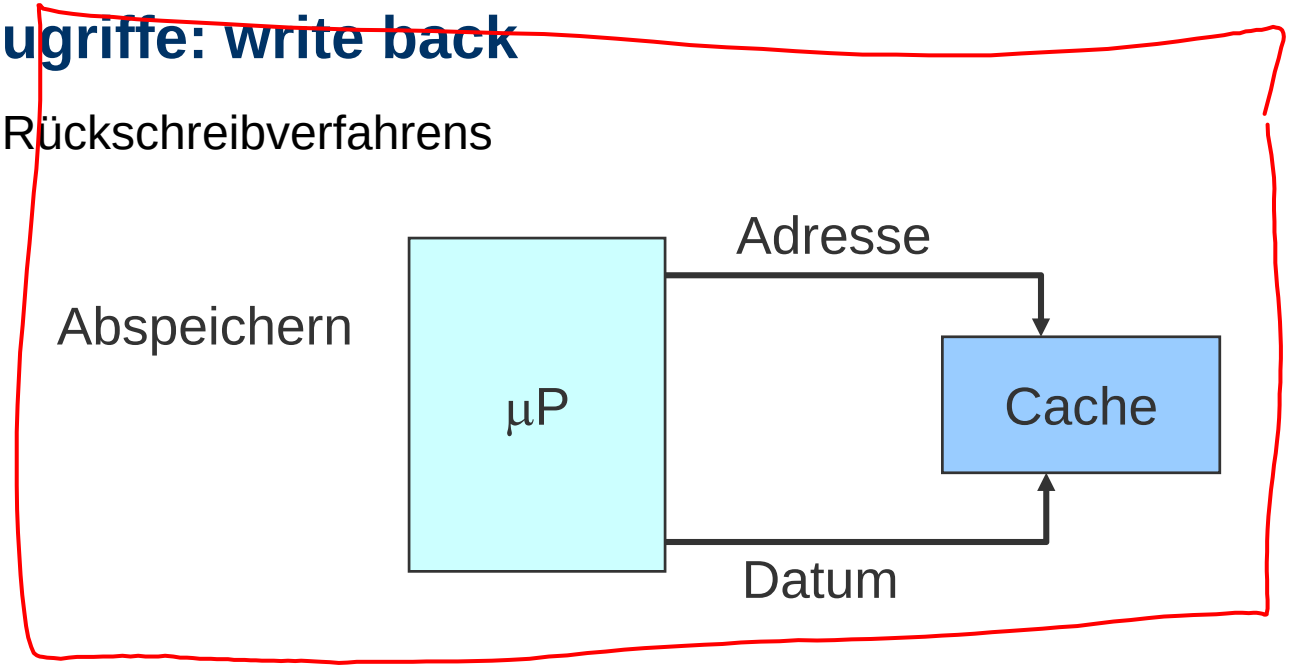
Rückschreib-Verfahren (write back policy)

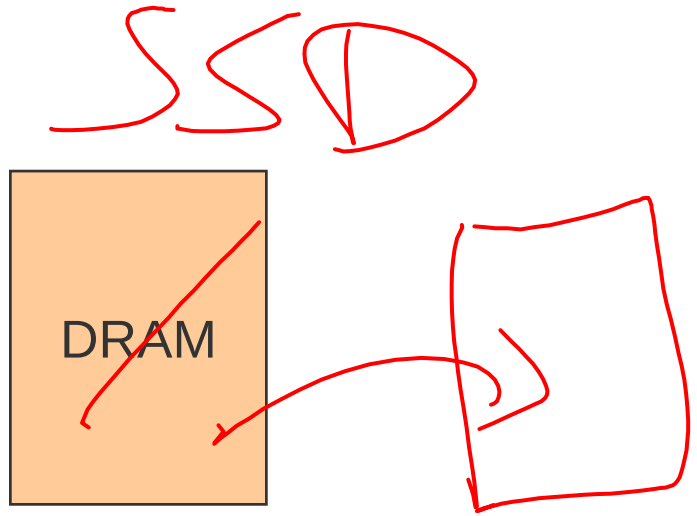
- Ein Datum wird von der CPU nur in den Cachespeicher geschrieben und durch ein spezielles Bit (altered bit, modified bit, dirty bit) gekennzeichnet.

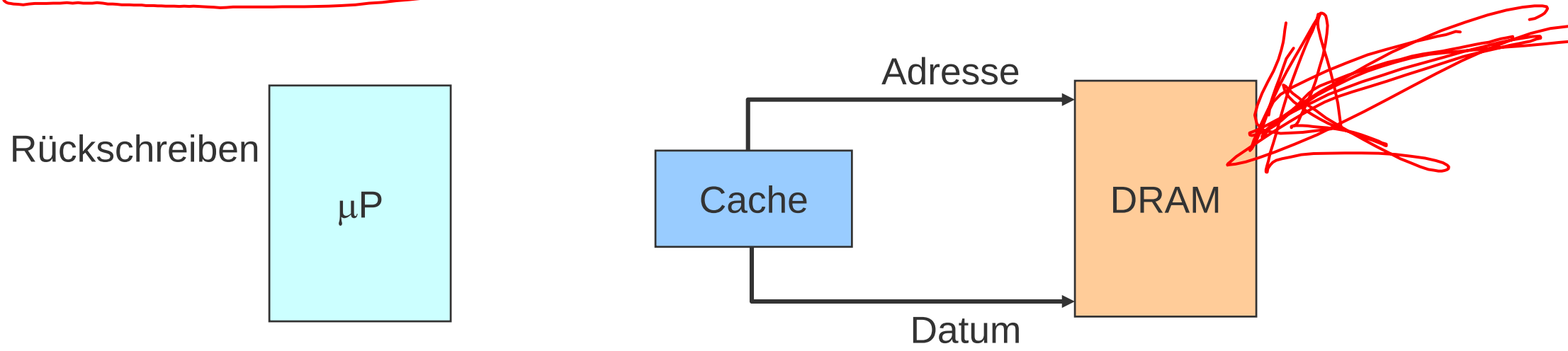
- Der Arbeitsspeicher wird nur geändert, wenn ein so gekennzeichnetes Datum aus dem Cache verdrängt wird.

Schreibzugriffe: write back

Prinzip des Rückschreibverfahrens







Schreibzugriffe: write back

Vorteil

- Auch Schreibzugriffe können mit der schnellen Cache-Zykluszeit abgewickelt werden

Nachteil

- Konsistenzprobleme zwischen Cache- und Arbeitsspeicher

Beispiele für Konsistenzprobleme

Andere Systemkomponenten, z.B. DMA-Controller, finden nun unter Umständen „veraltete Daten“ im Arbeitsspeicher vor, die von der CPU längst geändert, jedoch noch nicht in den Arbeitsspeicher übertragen wurden.

Ebenfalls können andere Systemkomponenten Daten im Hauptspeicher ändern, während die CPU noch mit den alten Daten im Cachespeicher arbeitet.

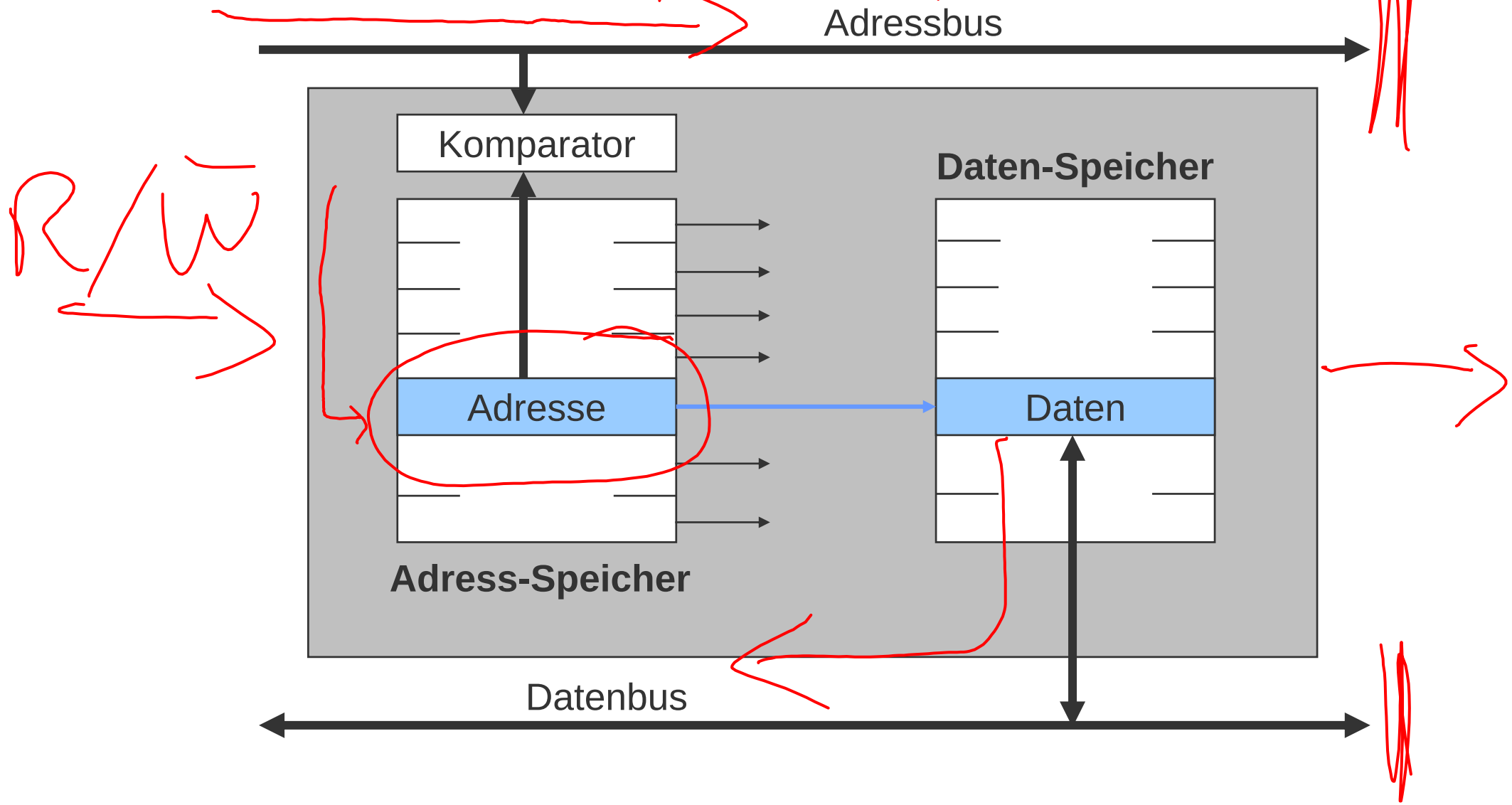
- Aufwändige Verfahren bei der Cache-Steuerung zur Verhinderung solcher Inkonsistenzen sind erforderlich (z.B. muss die Cache-Steuerung über jede Datenänderung im Hauptspeicher informiert werden).

Später mehr hierzu!

Cache-Speicher

AUFBAU EINES CACHE-SPEICHERS

Aufbau eines Cache-Speichers



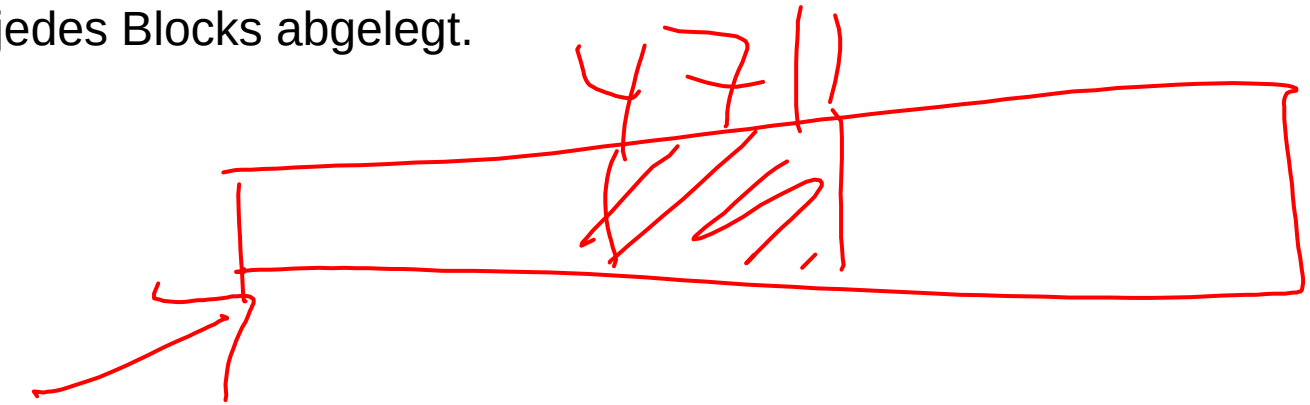
Aufbau eines Cache-Speichers

Ein Cache-Speicher besteht aus zwei Speicher-Einheiten

- Datenspeicher
  - enthält die im Cache abgelegten Daten
- Adressspeicher
  - enthält die Adressen dieser Daten im Arbeitsspeicher

Typischerweise

- Jeder Dateneintrag besteht aus einem ganzen Datenblock (cache line, z.B. 64 byte).
- Mit **jedem Datum**, auf das der Prozessor zugreift, wird die **Umgebung** miteingelagert (Hoffnung auf Lokalität von Programmen).
- Im Adressspeicher wird die **Basisadresse** jedes Blocks abgelegt.



Aufbau eines Cache-Speichers

Ein Komparator ermittelt, ob das zu einer auf dem Adressbus liegende Adresse gehörende Datum auch im Cache abgelegt worden ist

➡Adressvergleich mit den Adressen im Adressspeicher

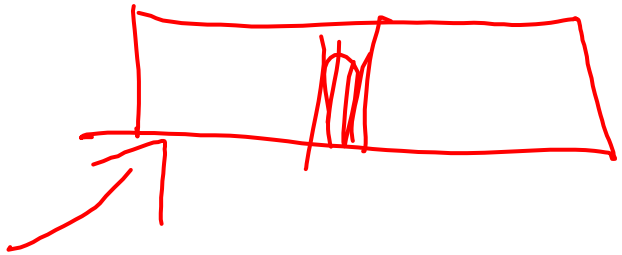
Dieser Adressvergleich muss sehr schnell gehen (in einem Taktzyklus), da sonst der Cachespeicher effektiv langsamer wäre als der Arbeitsspeicher.

Oft werden sog. **Assoziativspeicher** in Caches eingesetzt.

Blöcke und Sätze

Blockrahmen (Block Frame)

- Eine **Reihe von Speicherplätzen** im Cachespeicher, dazu ein **Adressetikett** (Address Tag oder Cache-Tag) und **Statusbits**.
- Das Adressetikett enthält die Speicheradresse (entweder die physikalische Hauptspeicheradresse oder die virtuelle Adresse) der aktuell im Blockrahmen gespeicherten Datenkopie.
- Die Statusbits sagen aus, ob die Datenkopie gültig ist.



Blocklänge (Block Size, Line Size)

- Die Anzahl der Speicherplätze in einem Blockrahmen.

Block (synonym Cache-Line)

- Datenportion, die in einen Blockrahmen passt. Typische Blocklängen sind 16 oder 32 Bytes.

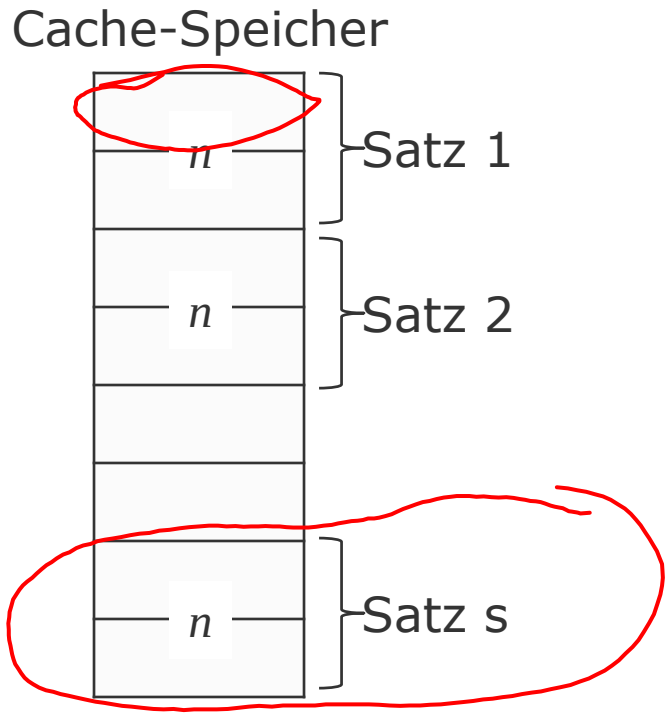
Die Menge der Blockrahmen ist in Sätze (Sets) unterteilt.

Weitere Begriffe

Assoziativität
- Anzahl der Blockrahmen in einem Satz

Gesamtzahl c der Blockrahmen in einem Cachespeicher ist das Produkt aus der Anzahl s der Sätze und der Assoziativität n, also c = s ×n.

Ein Cachespeicher heißt:
- *voll-assoziativ*, falls er nur aus einem einzigen Satz besteht (s=1, n=c),
- *direkt-abgebildet* (Direct-Mapped), falls jeder Satz nur einen einzigen Blockrahmen enthält (n=1, s=c) und
- *n-fach satzassoziativ* (n-Way Set-Associative) sonst (s= c/n).



Verdrängungsstrategie

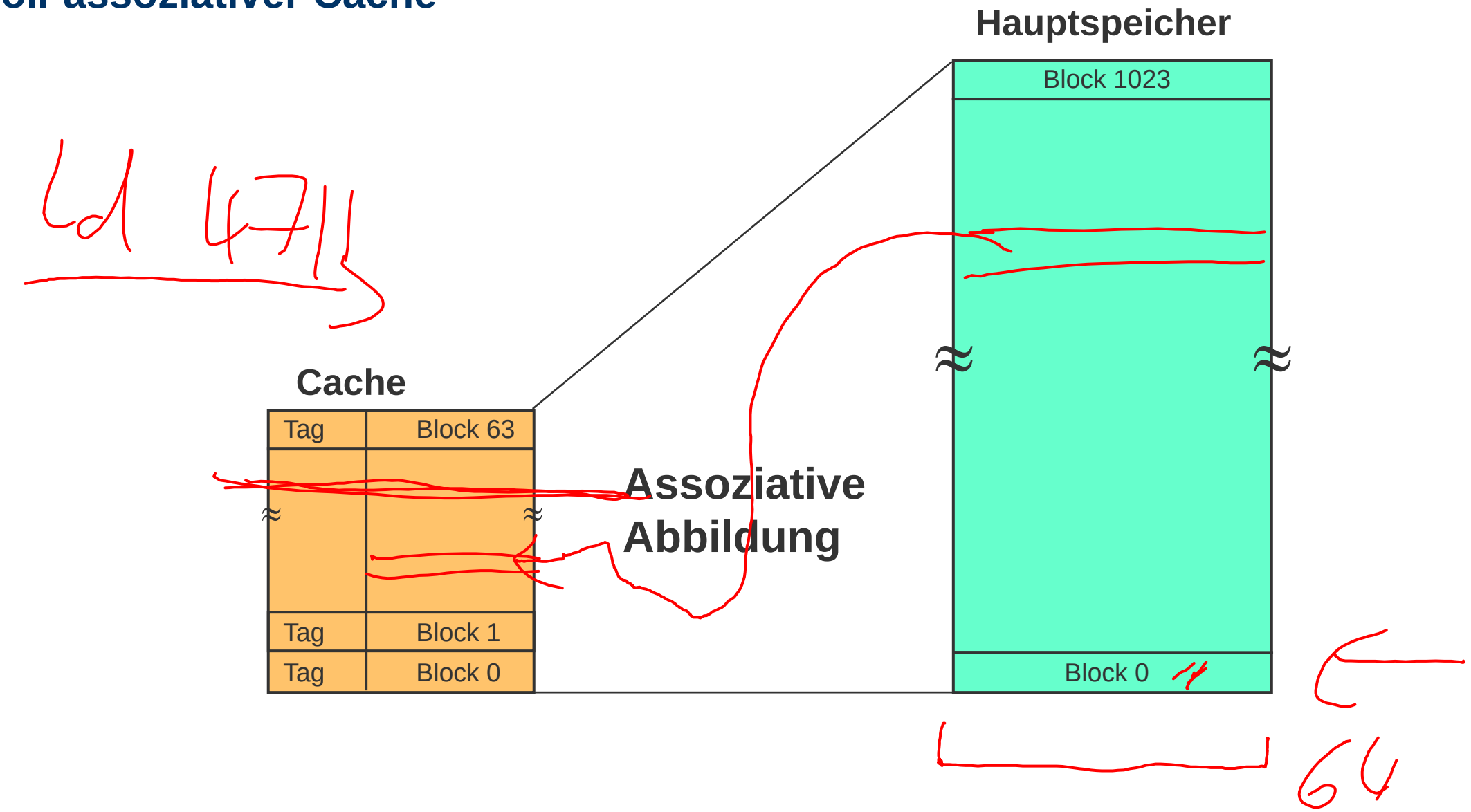
Verdrängungsstrategie gibt an, welcher Teil des Cachespeichers nach einem Cache-Miss durch eine neu geladene Speicherportion überschrieben wird.

Verdrängungsstrategie wird nur bei voll- oder n-fach satzassoziativer Cachespeicherorganisation angewandt

Meist wird eine einfache Strategie gewählt

- Least Recently Used (LRU)
- Die am längsten nicht benutzte Speicherportion ersetzen

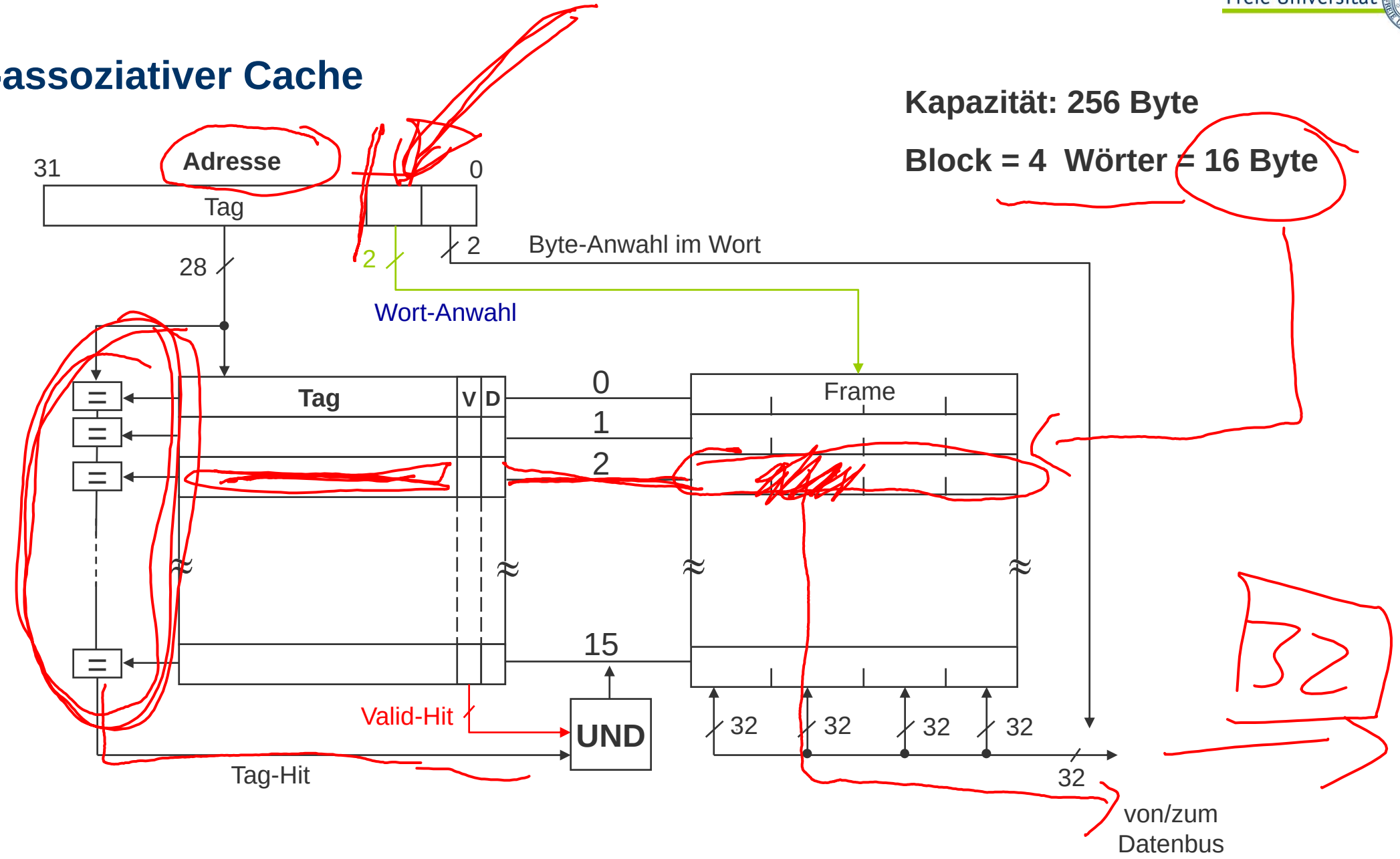
Voll-assoziativer Cache



Voll-assoziativer Cache

Kapazität: 256 Byte

Block = 4 Wörter = 16 Byte



Voll-assoziativer Cache

Vollparalleler Vergleich aller Adressen im Adressspeicher in einem einzigen Taktzyklus

Vorteil

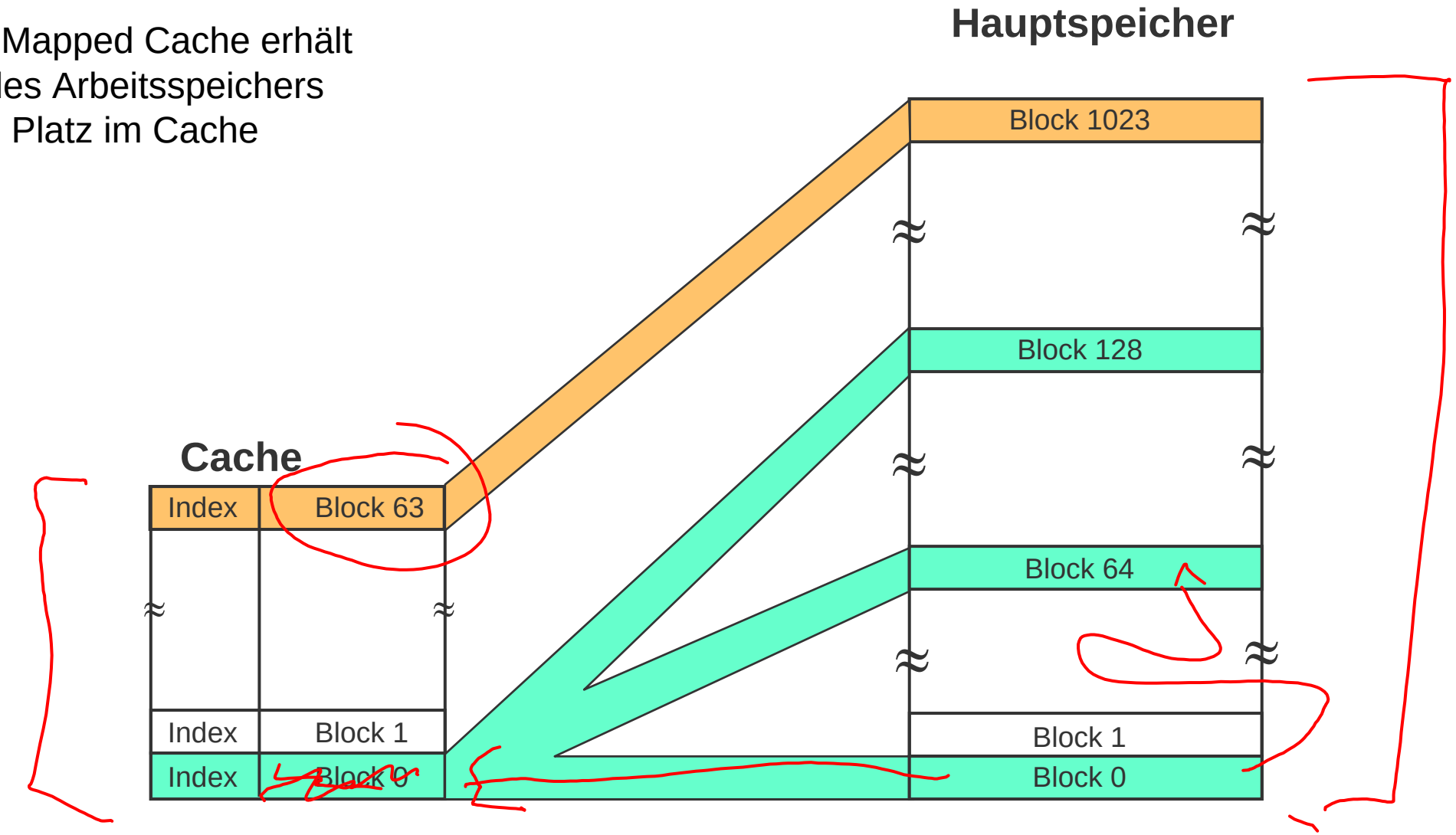
- ein Datum kann an beliebiger Stelle im Cache abgelegt werden
- Optimale Cache-Ausnutzung, völlig freie Wahl der Strategie bei Verdrängungen

Nachteil

- Hoher Hardwareaufwand (für jede Cache-Zeile ein Vergleicher)
  ➔ nur für sehr kleine Cachespeicher realisierbar
- Die große Flexibilität der Abbildungsvorschrift erfordert eine weitere Hardware, welche die Ersetzungsstrategie (welcher Block soll überschrieben werden, wenn der Cache voll ist) realisiert.

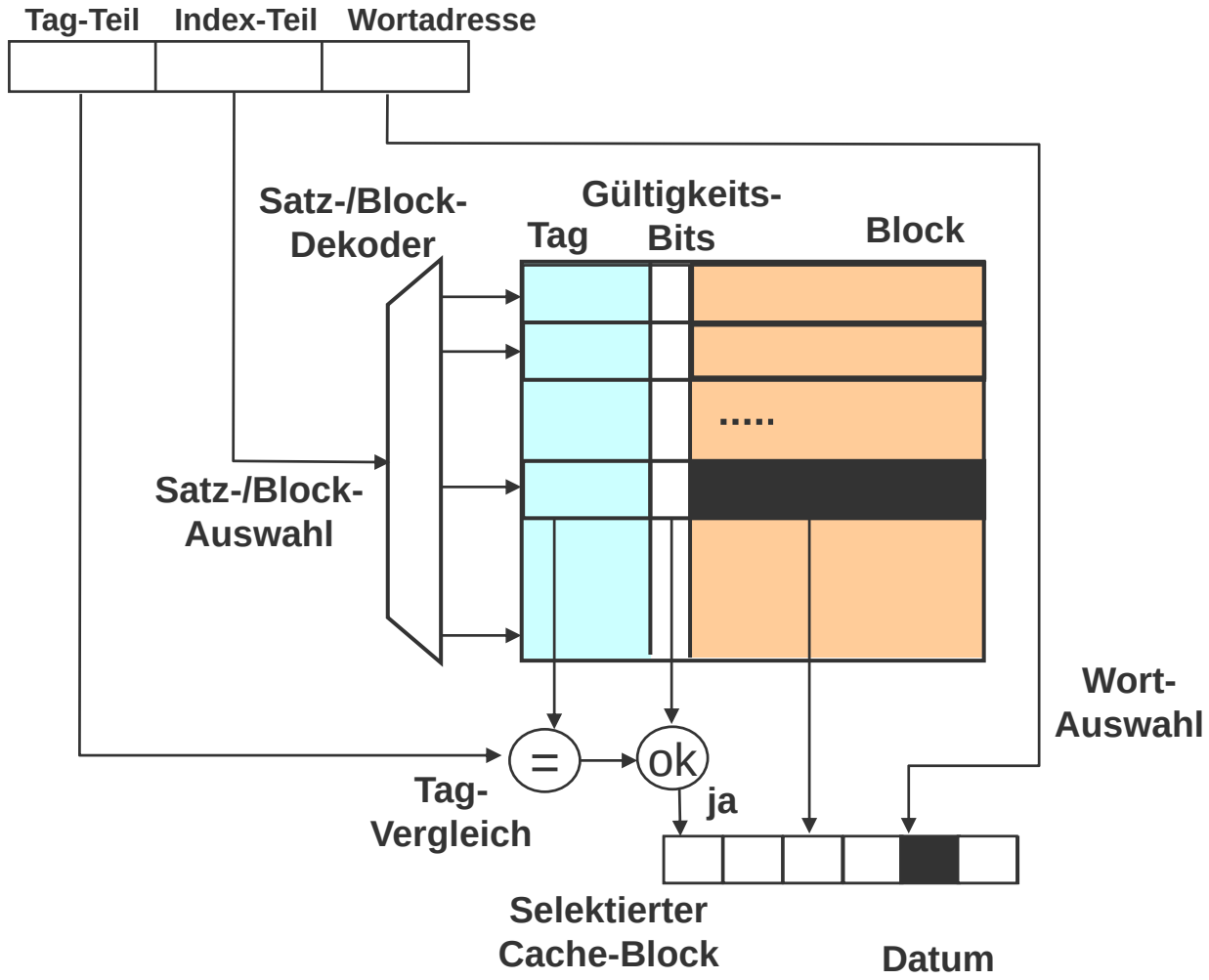
Direct-mapped-Cache

Beim Direct Mapped Cache erhält jede Stelle des Arbeitsspeichers einen festen Platz im Cache



Direct Mapped Cache

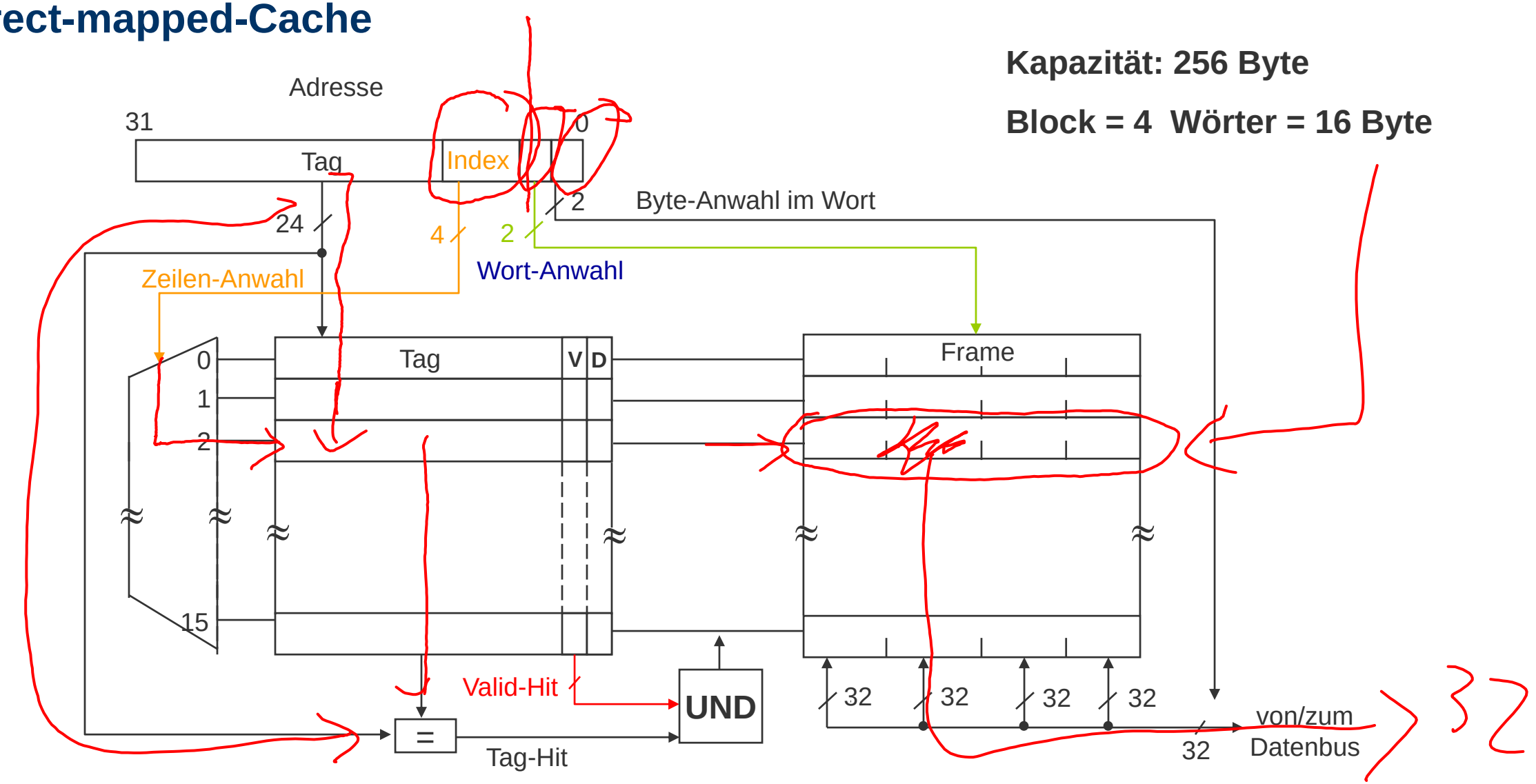
Adresse des gesuchten Datums



Direct-mapped-Cache

Kapazität: 256 Byte

Block = 4 Wörter = 16 Byte



Merkmale des direkt-abgebildeten Cachespeichers

Die Hardware-Realisierung ist einfach
- nur ein Vergleicher und ein Tag-Speicher

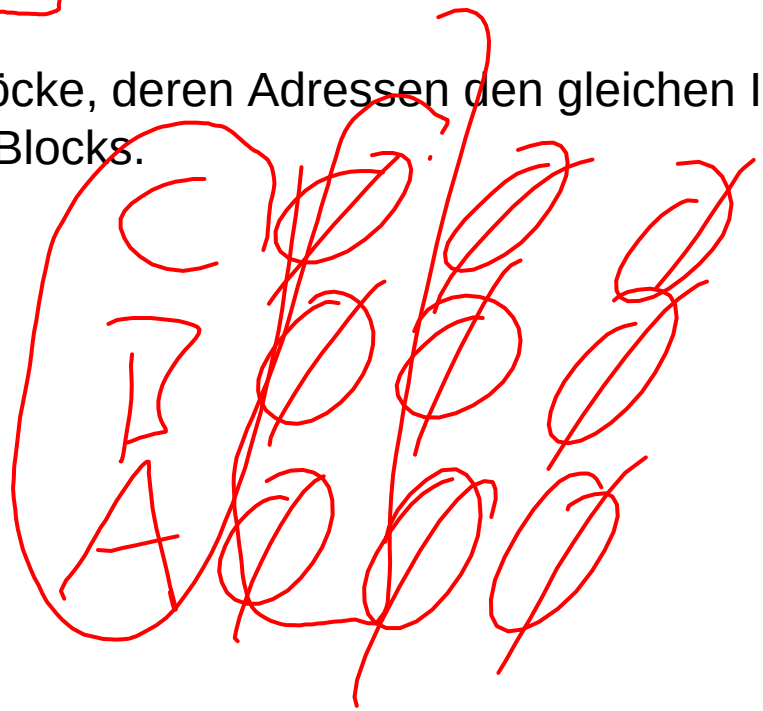
Der Zugriff erfolgt schnell, weil das Tag-Feld parallel mit dem Block gelesen werden kann.

Es ist kein Verdrängungsalgorithmus erforderlich, weil die direkte Zuordnung keine Alternativen zulässt.

Auch wenn an anderer Stelle im Cache noch Platz ist, erfolgt wegen der direkten Zuordnung eine Ersetzung.

Bei einem abwechselnden Zugriff auf Speicherblöcke, deren Adressen den gleichen Index-Teil haben, erfolgt laufendes Überschreiben des gerade geladenen Blocks.

Es kommt zum „Flattern“ (Thrashing).



Direct-mapped-Cache

Nachteile

- Ständige Konkurrenz der Blöcke (z.B. 0, 64, 128,...), obwohl andere Blöcke im Cache frei sein können.

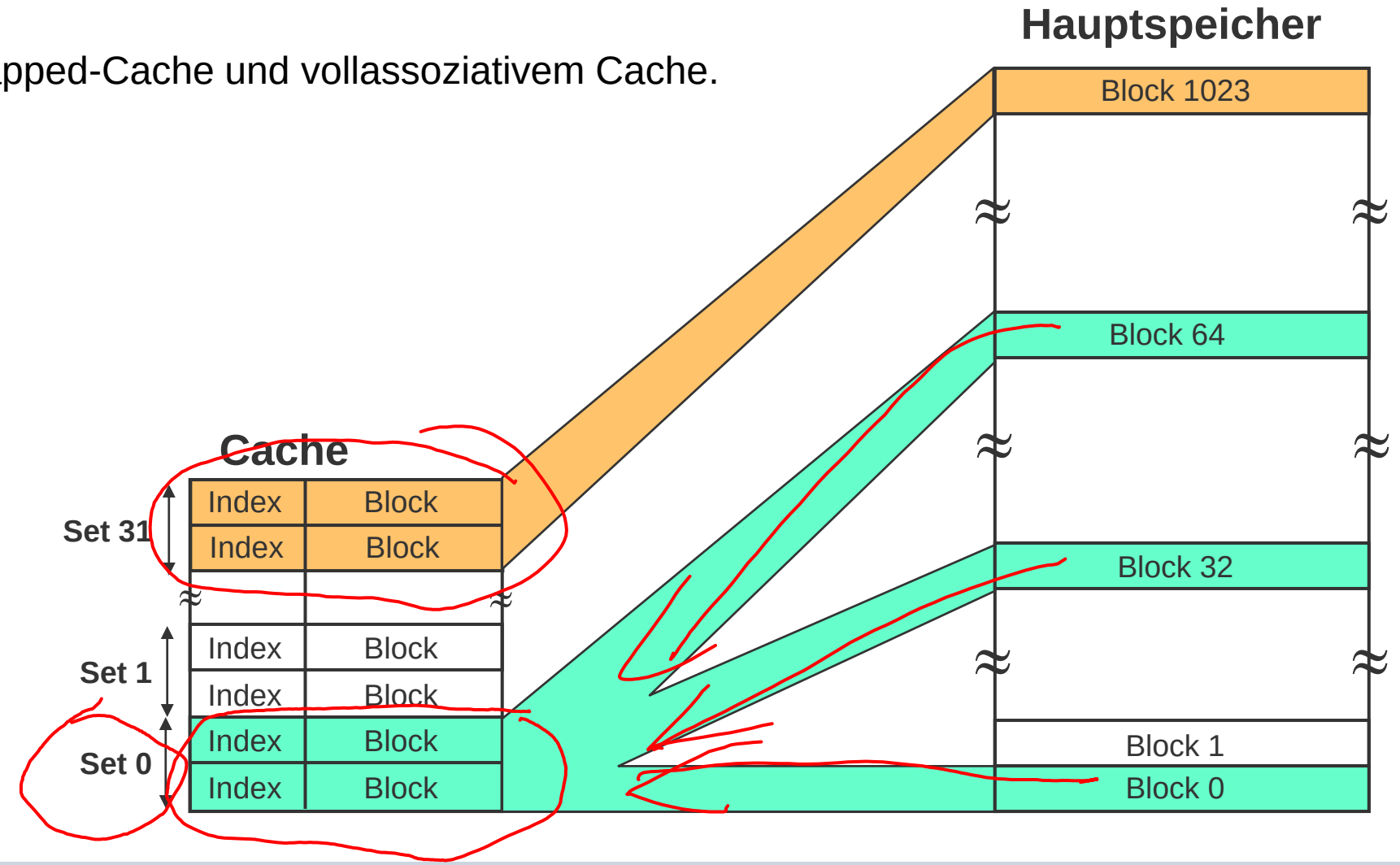
Vorteile

- Geringer Hardwareaufwand für die Adressierung, da nur ein Vergleicher für alle Tags benötigt wird.

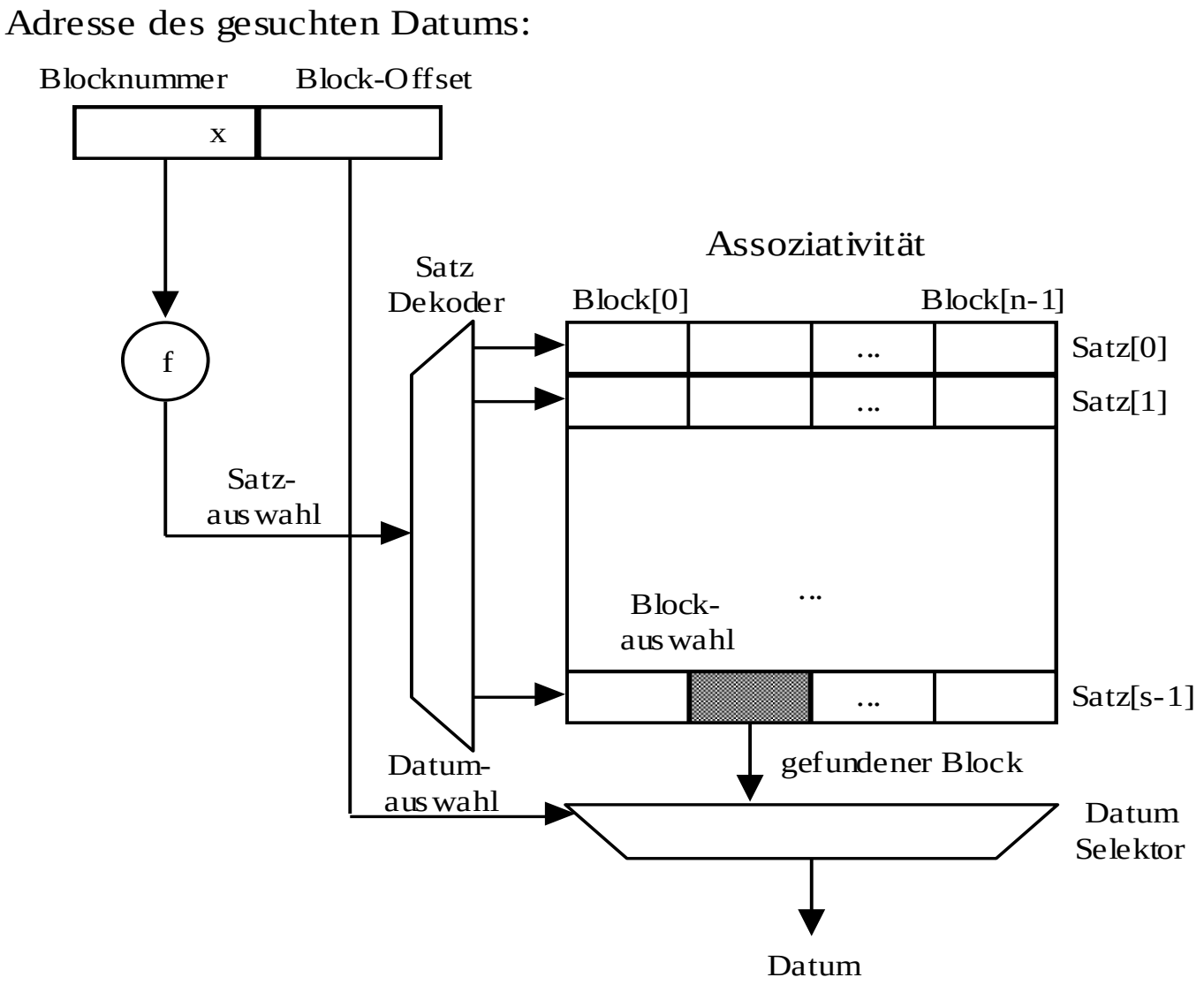
n-way-set-assoziativer Cache

Mehrere Cache-Zeilen werden zu Sätzen (Sets) zusammengefasst.

Kompromiss zwischen direct-mapped-Cache und vollassoziativem Cache.



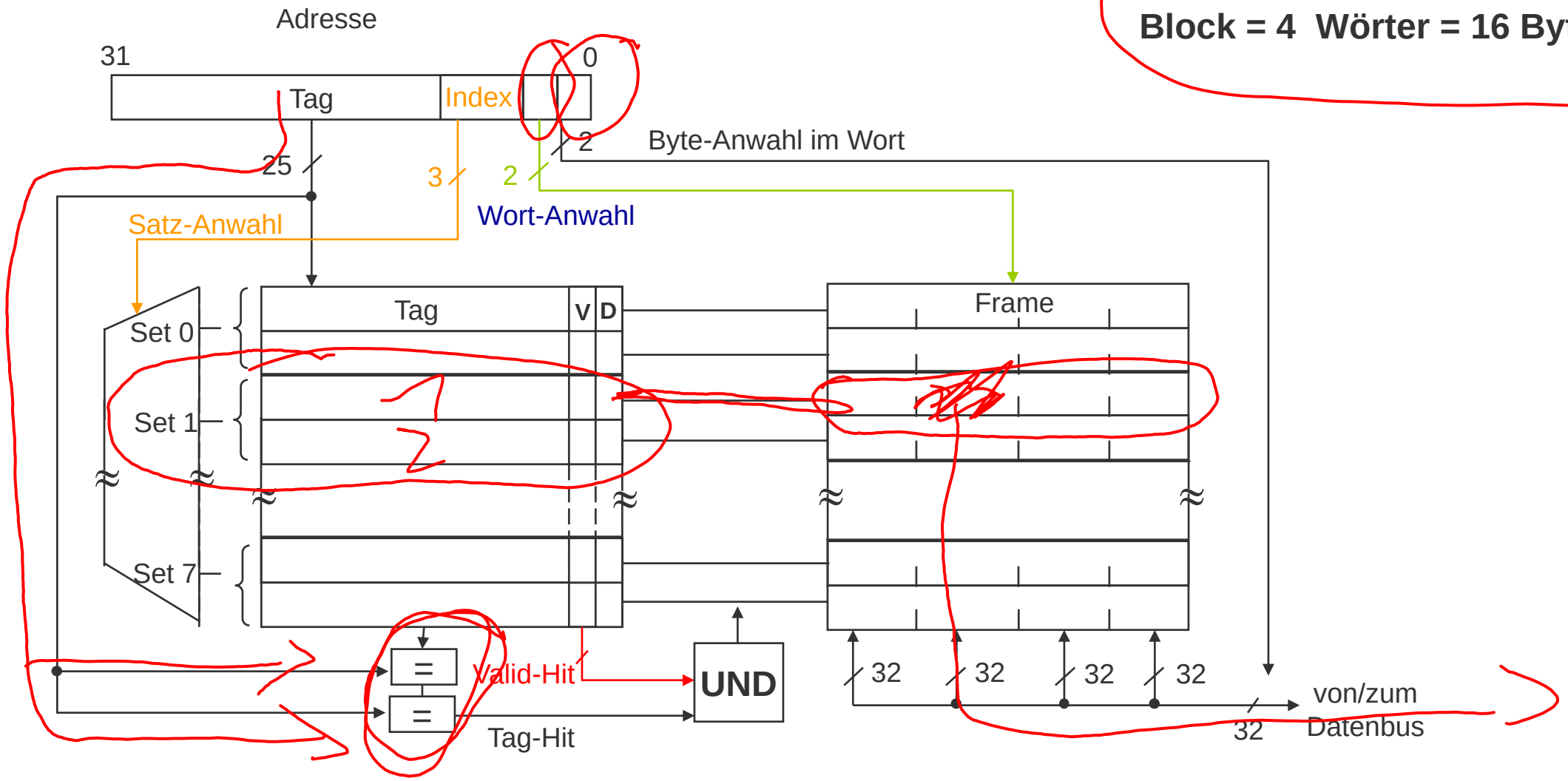
n-fach satzassoziative Cachespeicherverwaltung



2-way-set-assoziativer Cache

Kapazität: 256 Byte

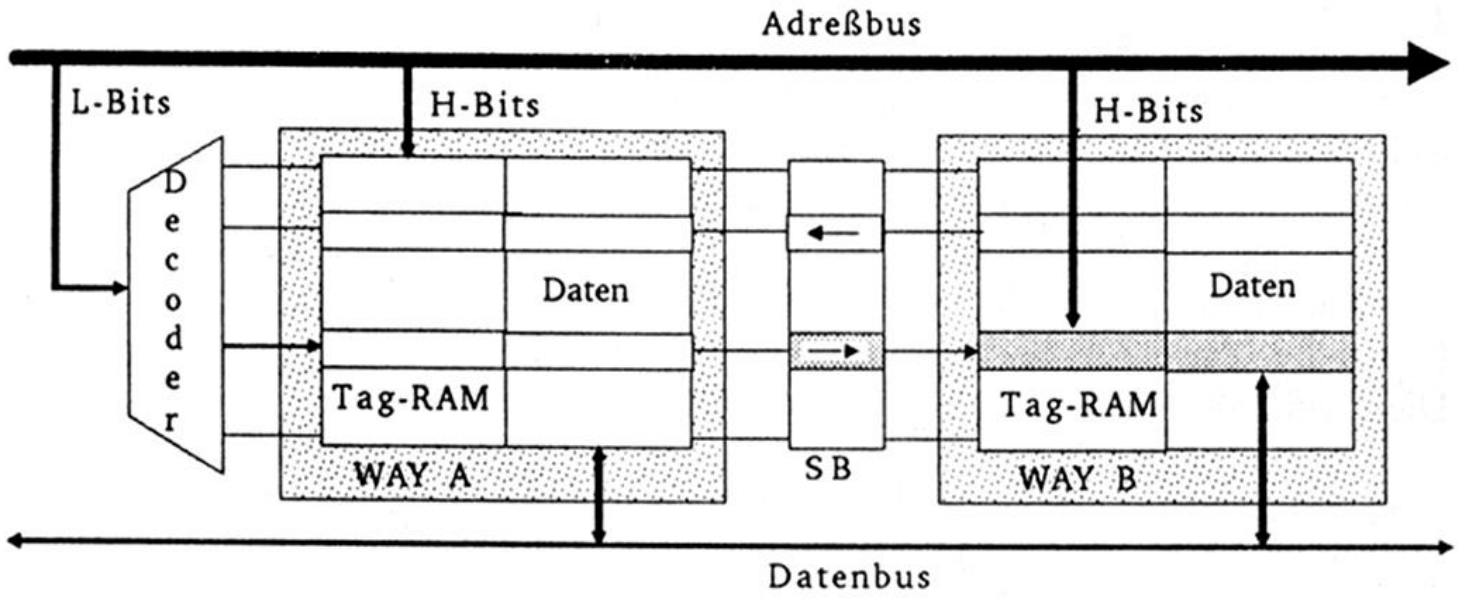
Block = 4 Wörter = 16 Byte



n-Way Set Associative Cache

n parallel-geschaltete Direct Mapped Caches

Eine Auswahllogik entscheidet, in welchen Cache (Set) ein Datum eingetragen wird



n-Way Set Associative Cache

Verbesserte Trefferrate, da hier eine Auswahl möglich ist

- der zu verdrängende Eintrag kann unter n ausgewählt werden

Auswahlstrategien (Verdrängungsstrategien)

- Zyklisch: der zuerst eingelagerte Eintrag wird auch wieder verdrängt, FIFO-Strategie
- Zufällig: durch Zufallsgenerator
- LRU-Strategie (least recently used): der am längsten nicht mehr benutzte Eintrag wird entfernt.

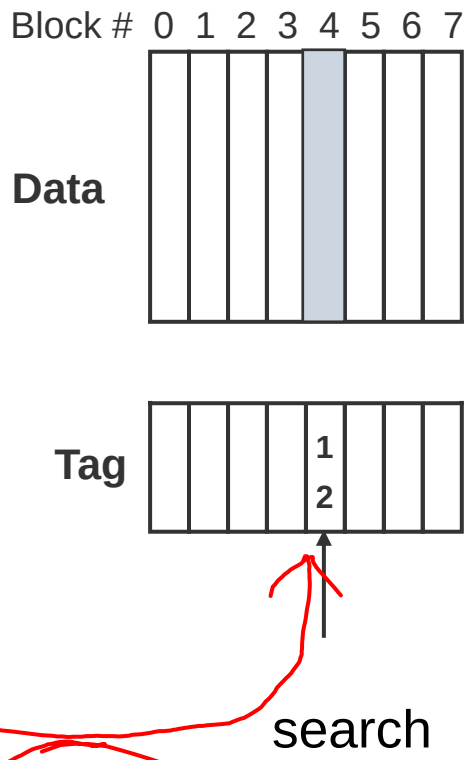
n-Way Set Associative Cache

Zum Auffinden eines Datums müssen alle n Tags mit demselben Index parallel verglichen werden

- ➡ der Aufwand steigt mit der Zahl n, für große n nähert sich der Aufwand den voll-assoziativen Caches
- ➡ Kompromiss zwischen Direct Mapped Cache und voll-assoziativem Cache

Example: Organization of a cache with 8 cache lines capacity

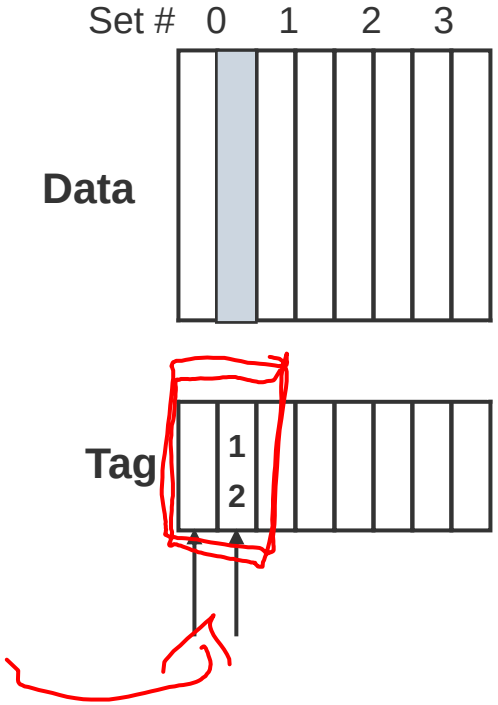
Direct-mapped



search

only one place for cache line 12

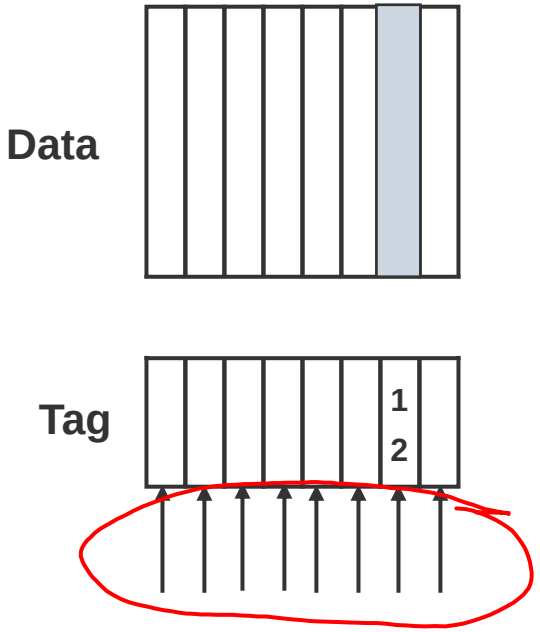
Set-associative



search

two possible places for cache line 12

Fully associative

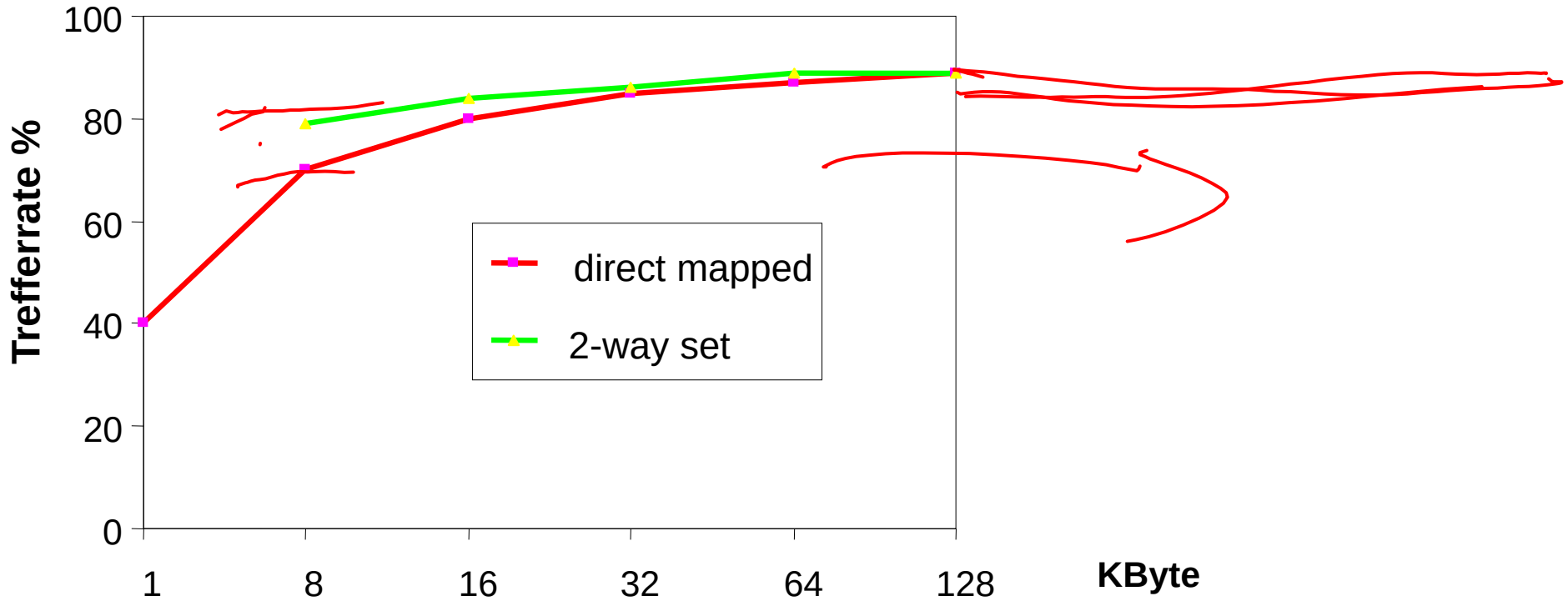


search

cache line 12 can be anywhere

Erzielbare Cache-Trefferquoten

Nach Unterlagen der Firma Intel:



Cache-Größen < 64 kbyte: 2-Way Set Associative Cache besser als Direct Mapped Cache

Cache-Größen ≥ 64 kbyte: kaum noch Unterschiede

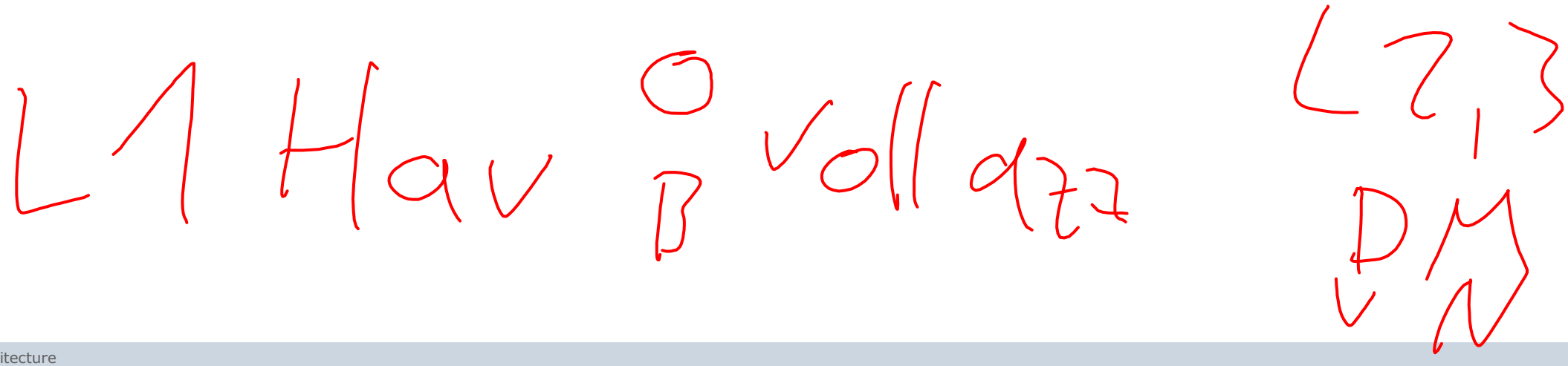
Erzielbare Cache-Trefferquoten

Nach Untersuchungen von Agarwal, Hennessy und Horowitz:

- Eine Cache-Trefferquote von circa 94% kann bei einem 64 kByte großen Cachespeicher erreicht werden (selbstverständlich gilt: je größer der Cachespeicher, desto größer die Trefferquote)

- Getrennte Daten- und Befehls-Cachespeicher sind bei sehr kleinen Cachespeichergrößen vorteilhaft, fallen jedoch bei Cachespeichergrößen ab ca. 8 KByte nicht mehr ins Gewicht

- Bei Cachespeichergrößen ab 64 KByte sind Direct Mapped Cachespeicher mit ihrer Trefferquote nur wenig schlechter als Cachespeicher mit Assoziativität 2 oder 4



Erzielbare Cache-Trefferquoten

➡ Voll-assoziative Cachespeicher werden heute nur für sehr kleine auf dem Chip integrierte Caches mit 32 bis 128 Einträgen verwendet.

Bei größeren Cachespeichern findet sich zur Zeit ein Trend zur Direct Mapped Organisation oder 2 - 4 fach assoziativer Organisation.

VIRTUELLER SPEICHER

Virtueller Speicher - Motivation

Anwendungen benötigen meist mehr Speicher als physikalisch im RAM vorhanden

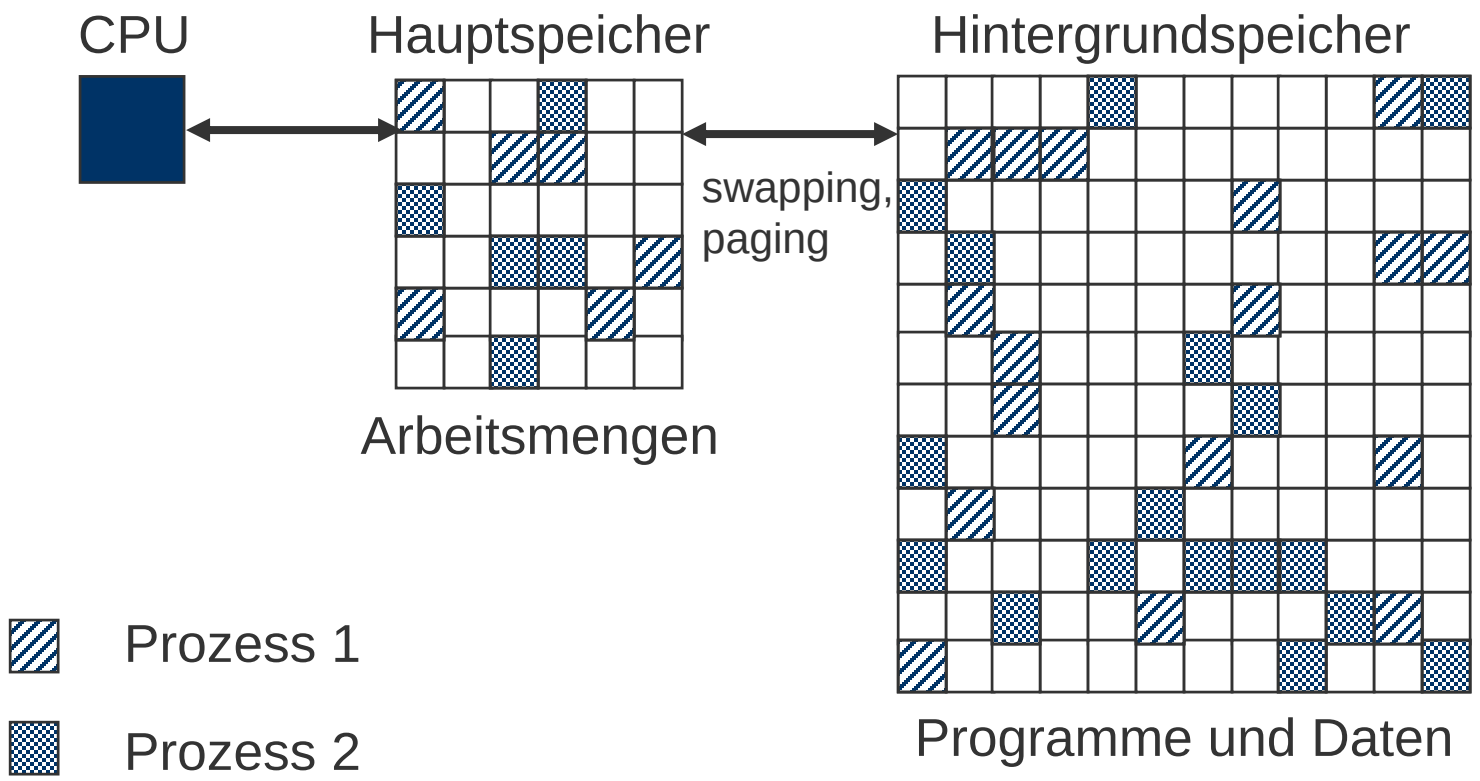
- Insbesondere: Mehrbenutzer-, Multitasking- und mehrfädige (multi-threaded) Betriebssysteme

Programme, die in einer Mehrprozessumgebung lauffähig sein sollen, müssen verschiebbar sein, d.h. nicht an festgelegte, physikalische Speicheradressen gebunden sein.

Speicherreferenzen in den Maschinenbefehlen des Objektccodes erfolgen daher über sogenannte **virtuelle** oder **logische Adressen**, die erst bei der Programmausführung in physikalische Speicheradressen transformiert werden.

- Ein großer einheitlicher Adressraum für die einzelnen Prozesse.
- Geeignete Schutzmechanismen zwischen den Prozessen erforderlich.

Grundstruktur virtueller Speicherverwaltung



Virtuelle Speicherverwaltung 1

Betriebssystem

- Verwaltet freie Speicherbereiche und lagert bei nicht ausreichendem Freiplatz im Hauptspeicher diejenigen Speicherinhalte auf den Hintergrundspeicher aus, die gegenüber ihrem Originalen auf dem Hintergrundspeicher verändert worden sind.

Übersetzungstabellen

- Erforderliche Abbildungsinformation

Vorgang bleibt dem Anwender völlig verborgen

- d.h. der Arbeitsspeicher erscheint dem Anwender wesentlich größer, als er in Wahrheit ist – daher virtueller Speicher

MMU (Memory Management Unit)

- Unterstützung der Verwaltung des virtuellen Speichers durch Hardware
- Schnelle Umsetzung virtueller (logischer) Adressen in physikalische Adressen

Virtuelle Speicherverwaltung 2

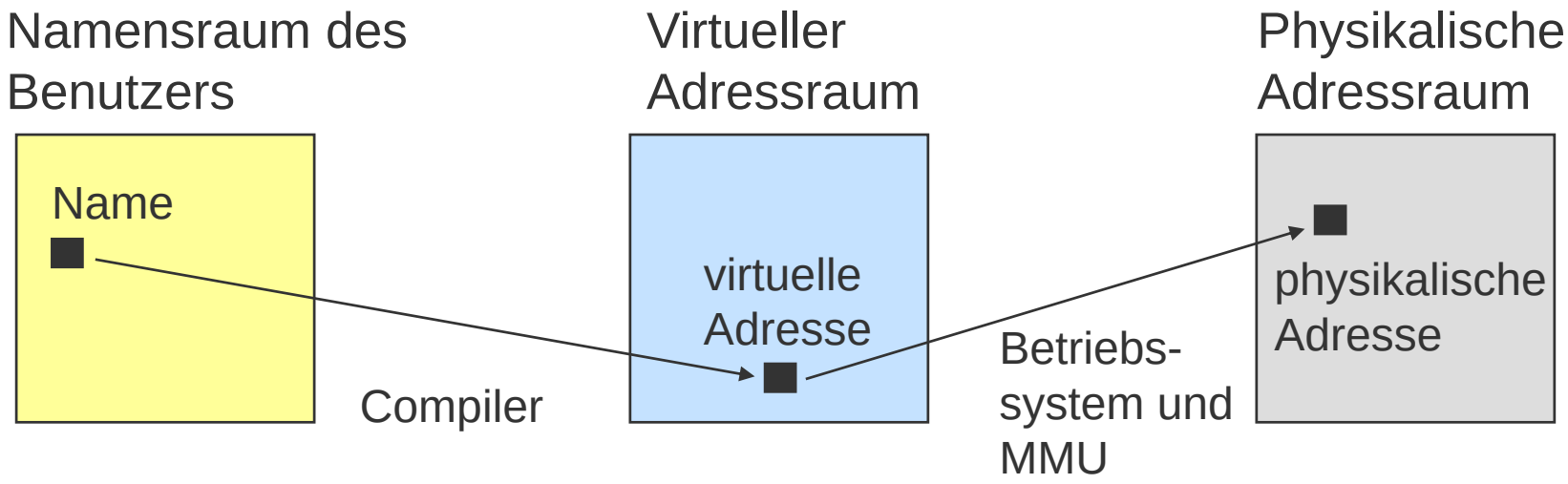
Lokalitätseigenschaften von Programmen und Daten

- Programme greifen in einem kleinen Zeitintervall auf einen relativ kleinen Teil des Adressraumes zu.
- Die Lokalitätseigenschaften gewährleisten eine hohe Wahrscheinlichkeit, dass die Daten, welche die CPU anfordert, im physikalischen Hauptspeicher zu finden sind.

Zwei Arten der Lokalität

- Zeitliche Lokalität
  - Falls ein Datum oder ein Befehl referenziert wird, so werden sie bald wieder referenziert.
- Örtliche Lokalität
  - Falls ein Datum oder ein Befehl referenziert wird, werden bald Daten oder Befehle mit benachbarten Adressen referenziert.

Virtuelle Speicherverwaltung 3

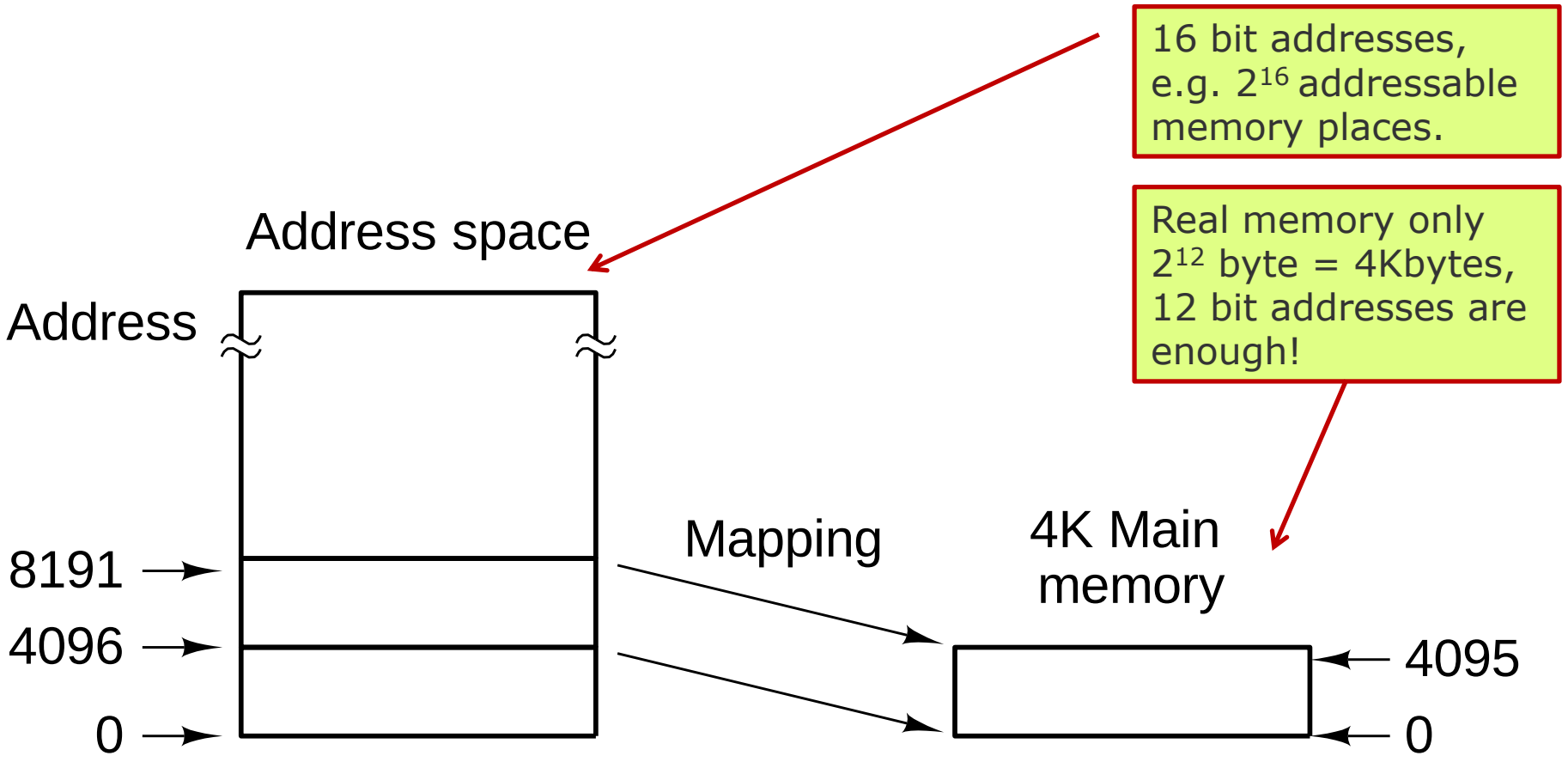


Benutzer: kennzeichnet seine Objekte (Programme, Unterprogramme, Variablen, ...) durch Namen

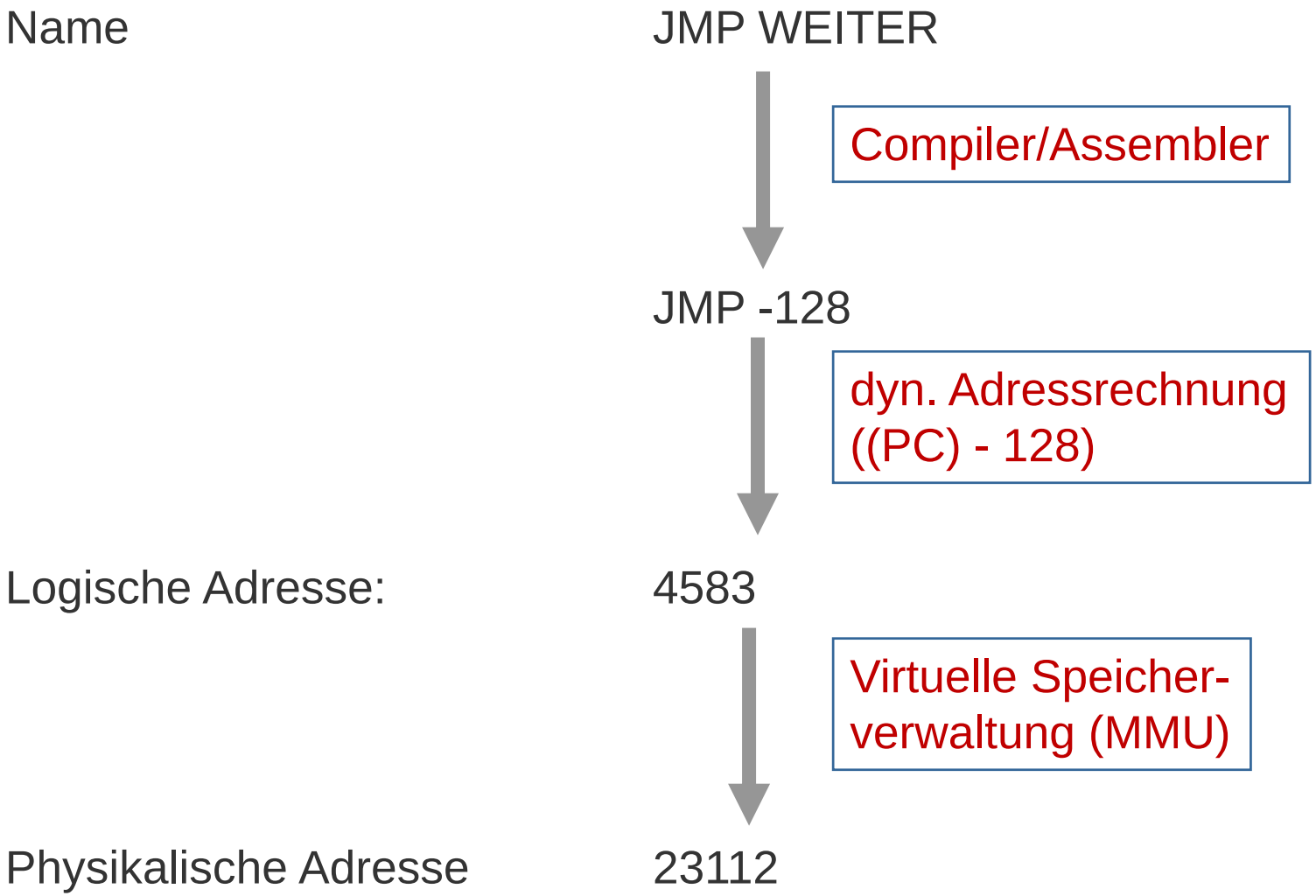
Compiler: übersetzt diese Namen in virtuelle (logische) Adressen.

Virtuelle Speicherverwaltung: wandelt diese Adressen zur Laufzeit je nach gerade gegebener Speicherbelegung in die physikalische Adresse (wirklicher Ort des Objekts im Hauptspeicher) um

Mapping of virtual addresses



Beispiel


Name
JMP WEITER
Compiler/Assembler
JMP -128
dyn. Adressrechnung
((PC) - 128)
Logische Adresse:
4583
Virtuelle Speicher-
verwaltung (MMU)
Physikalische Adresse
23112

Virtueller Speicher

SEITENWECHEL (PAGING)

Segmentierungs- und Seitenwechselverfahren

Es existieren zwei grundlegende Verfahren zur virtuellen Speicherverwaltung:

- **Seitenwechsel (Paging)**
- Segmentierung (Segmentation)

Aufteilung in Seiten

- Hierbei wird der logische und der physikalische Adressraum in "Segmente fester Länge", die so genannten Seiten (pages) unterteilt.
- Die Seiten sind relativ klein (256 Byte - 4 kByte)
- Ein Prozess wird auf viele dieser Seiten verteilt (keine logischen Zusammenhänge wie bei der Segmentierung)

Pages in virtual and real memory

Page	Virtual addresses
≈	≈
15	61440 - 65535
14	57344 - 61439
13	53248 - 57343
12	49152 - 53247
11	45056 - 49151
10	40960 - 45055
9	36864 - 40959
8	32768 - 36863
7	28672 - 32767
6	24576 - 28671
5	20480 - 24575
4	16384 - 20479
3	12288 - 16383
2	8192 - 12287
1	4096 - 8191
0	0 - 4095

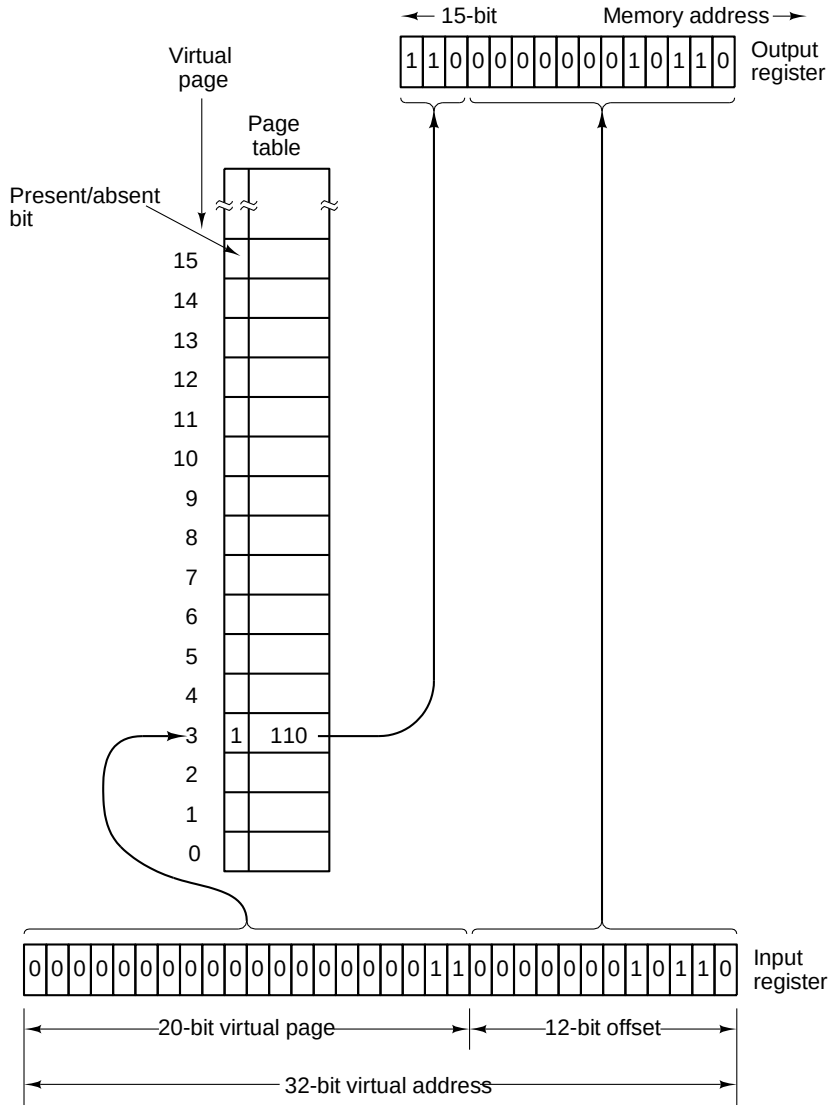
(a)

Mapping of the virtual addresses onto physical addresses!

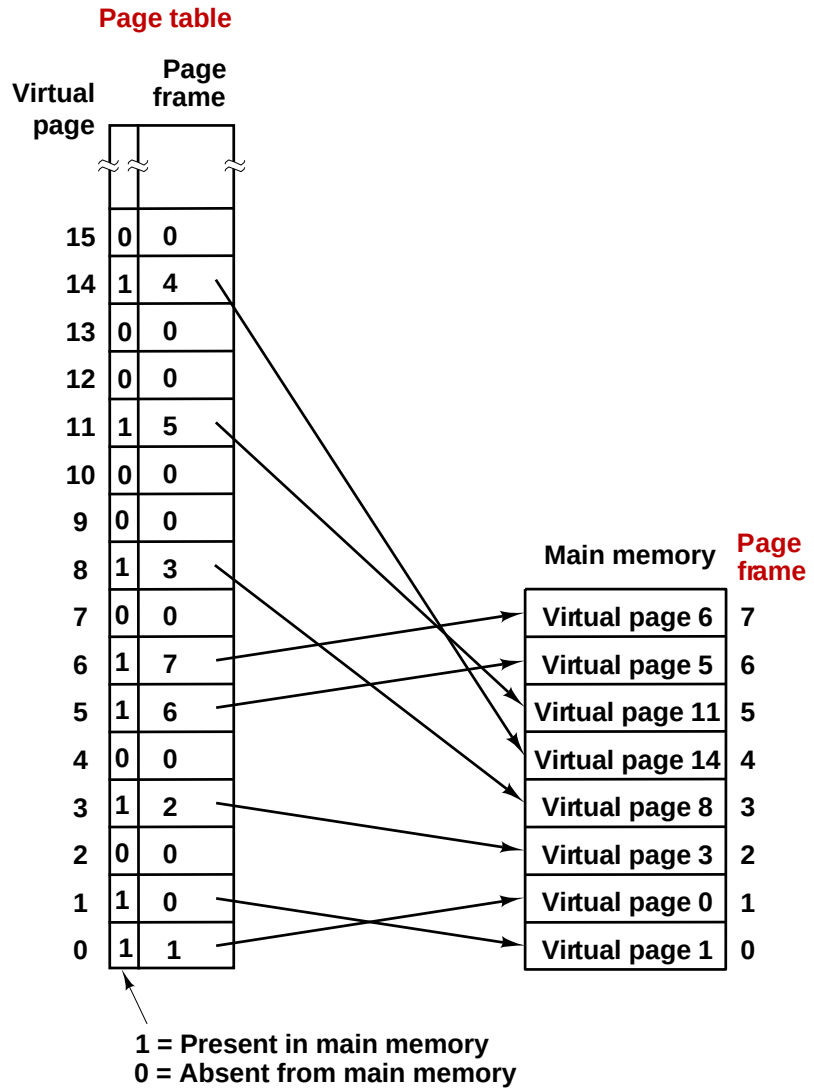
Page frame	Bottom 32K of main memory Physical addresses
7	28672 - 32767
6	24576 - 28671
5	20480 - 24575
4	16384 - 20479
3	12288 - 16383
2	8192 - 12287
1	4096 - 8191
0	0 - 4095

(b)

Translation of virtual to real addresses



Possible mapping of virtual pages



Seitenaufteilung

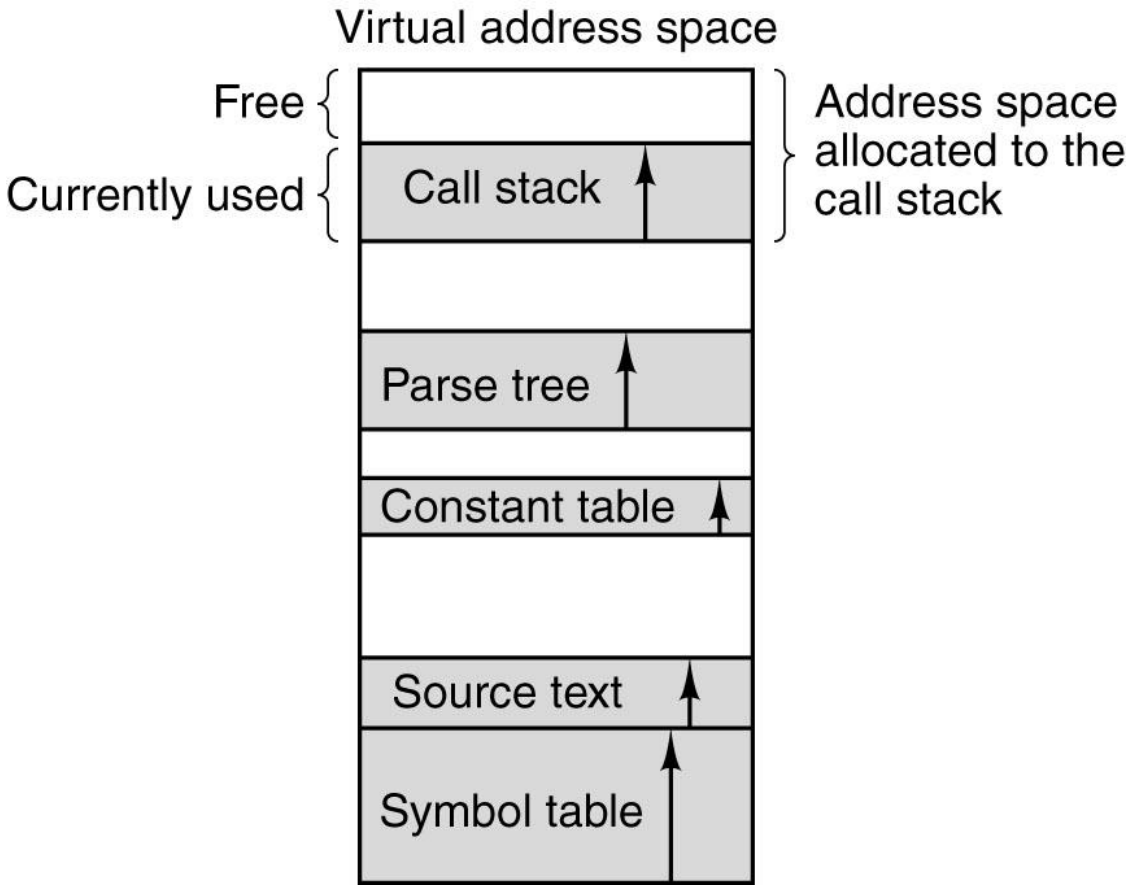
Vorteile

- durch kleine Seiten wird nur der wirklich benötigte Teil eines Programms eingelagert.
- geringerer Verwaltungsaufwand als Segmentierung

Nachteil

- häufiger Datentransfer

Problems of a one-dimensional address space



Growing tables may bump into another

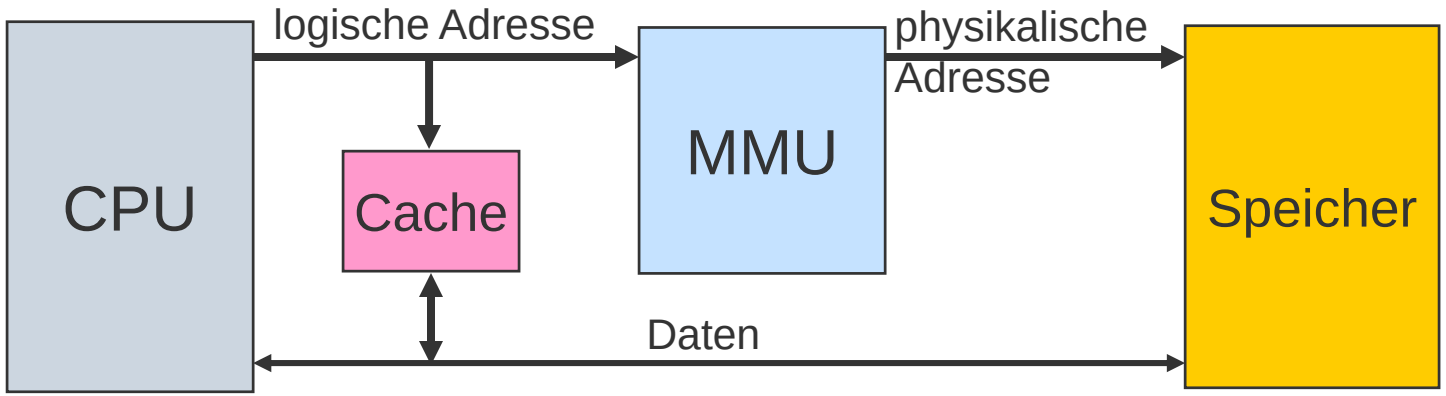
VIRTUELLER SPEICHER UND DER CACHE

Cache und Speicherverwaltungseinheit

Zwei Möglichkeiten der Cache-Einbindung bei virtueller Speicherverwaltung:

1. Virtueller Cache

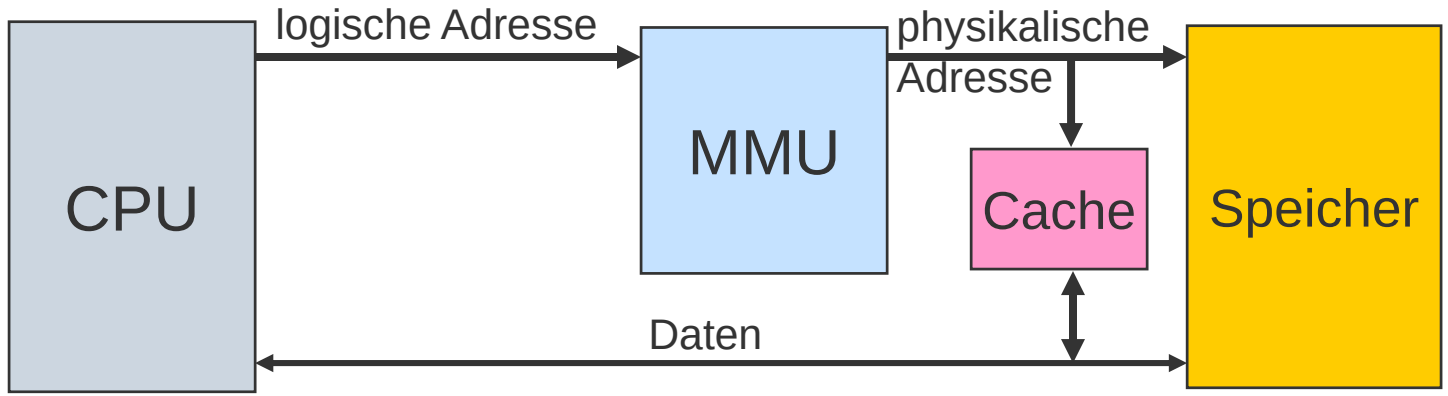
- wird zwischen CPU und MMU gelegt. Die höherwertigen Bits der logischen Adressen werden als Tags abgelegt.



Cache und Speicherverwaltungseinheit

2. Physikalischer Cache

- wird zwischen MMU und Speicher gelegt. Die höherwertigen Bits der physikalischen Adressen werden als Tags abgelegt.



Virtueller und physikalischer Cache

Vorteile des virtuellen Caches

- bei Treffern wird die MMU nicht benötigt

Vorteile des physikalischen Caches

- physikalische Adresse ist i. Allg. viel kleiner als die logische Adresse
  ➔ weniger Bits müssen als Tag gespeichert werden.
- befindet sich die MMU auf dem Prozessorchip, so kann nur der physikalische Cache außerhalb erweitert werden.

Schutzmechanismen

Moderne Mikroprozessoren bieten Schutzmechanismen an, um während der Laufzeit von Programmen unerlaubte Speicherzugriffe zu verhindern.

Dies geschieht im Wesentlichen durch:

- Trennung der Systemsoftware, z.B. des Betriebssystems, insbesondere des Ein-/Ausgabe-Subsystem (BIOS, basic I/O system), von den Anwendungsprozessen.
- Trennung der Anwendungsprozesse voneinander. Ist dies nicht gewährleistet, könnte ein fehlerhaftes Anwenderprogramm andere, fehlerfreie Programme beeinflussen (Schutzebenen und Zugriffsrechte).

SPEICHER IN MULTIPROZESSORSYSTEMEN

Allgemeine Grundlagen

Multiprozessorsysteme oder Multiprozessor Rechner, die in die MIMD-Klasse (multiple instructions multiple data) des (historischen) Flynnschen Klassifikationsschemas fallen.

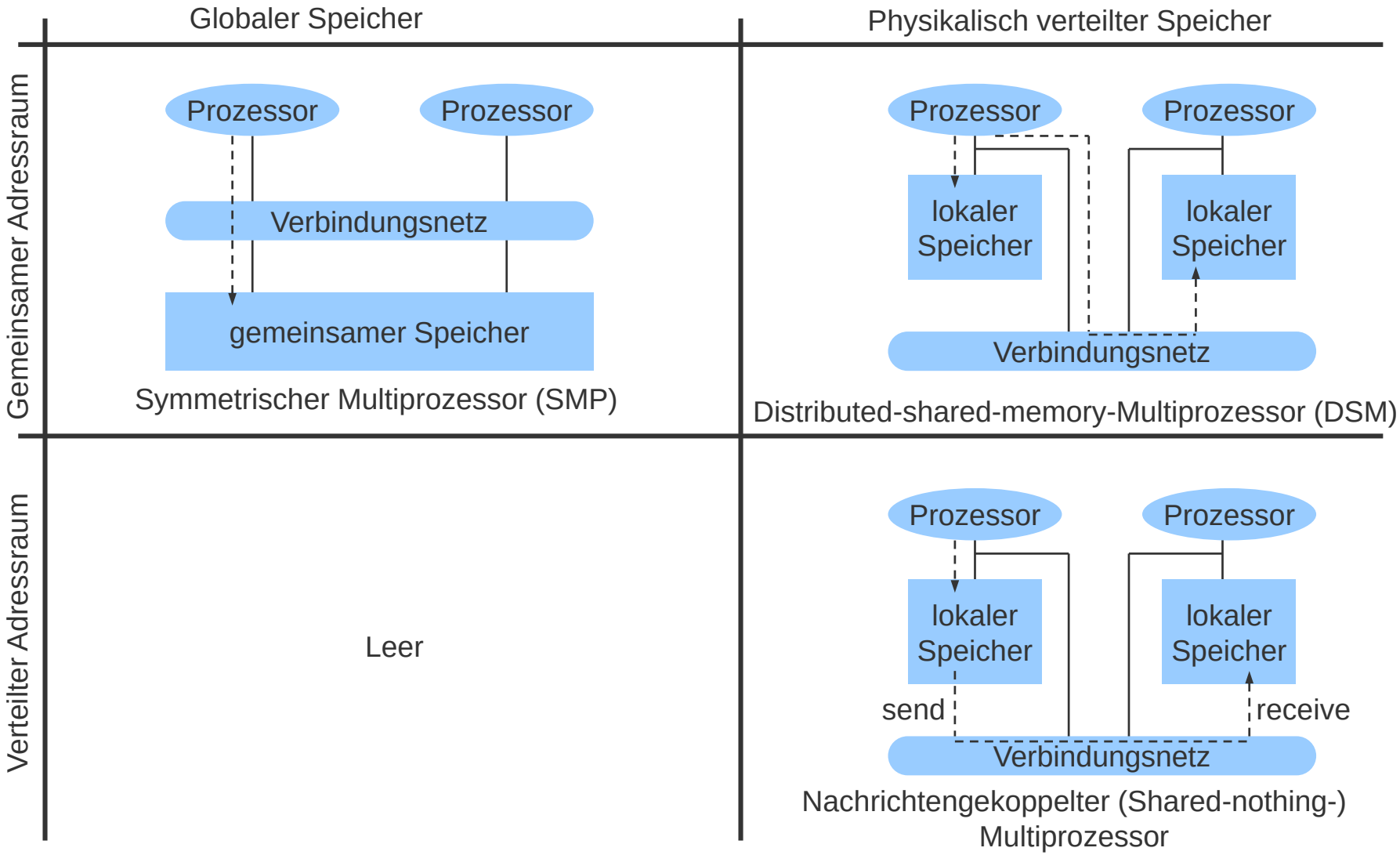
Speichergekoppelte Multiprozessoren

- Gemeinsamer Adressraum für alle Prozessoren
- Kommunikation und Synchronisation über gemeinsame Variable
- Symmetrischer Multiprozessor (SMP)
  - gemeinsamer Adressraum, ein globaler Speicher
- Distributed-shared-memory-System (DSM)
  - gemeinsamer Adressraum trotz physikalisch verteilter Speichermodule

Nachrichtengekoppelte Multiprozessoren

- nur physikalisch verteilte Speicher und prozessorlokale Adressräume für alle Prozessoren
- Kommunikation durch Austauschen von Nachrichten

Konfigurationen



Speichergekoppelte Multiprozessoren

Uniform-memory-access-Modell (UMA)

- Alle Prozessoren greifen gleichermaßen auf den gemeinsamen Speicher zu.
- Insbesondere ist die **Zugriffszeit** aller Prozessoren auf den gemeinsamen Speicher gleich.
- Jeder Prozessor kann zusätzlich einen lokalen Cache-Speicher besitzen.
- Typische Beispiele: symmetrische Multiprozessoren

Nonuniform-memory-access-Modell (NUMA)

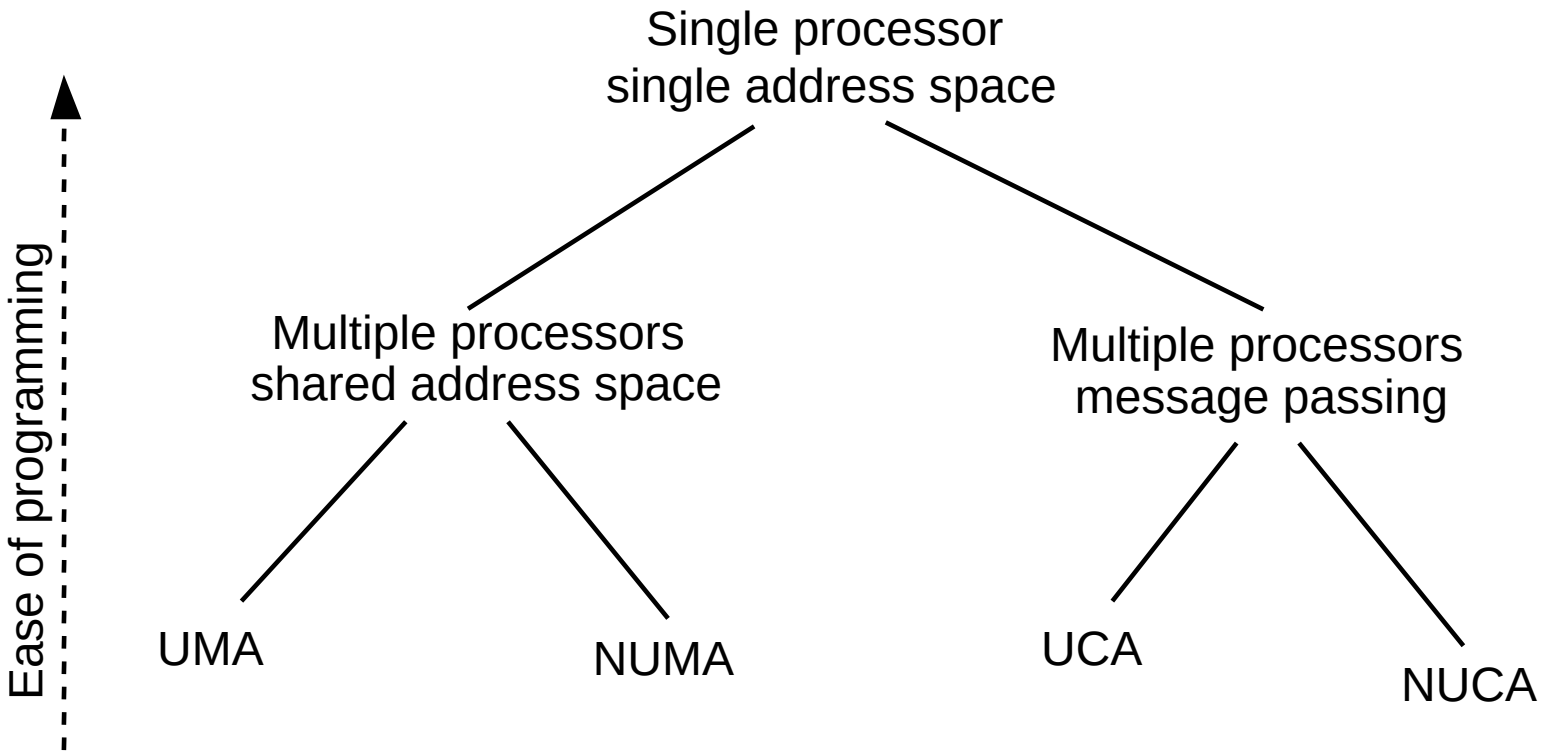
- Die **Zugriffszeiten** auf Speicherzellen des gemeinsamen Speichers variieren je nach dem Ort, an dem sich die Speicherzelle befindet.
- Die Speichermodule des gemeinsamen Speichers sind physikalisch auf die Prozessoren aufgeteilt.
- Typische Beispiele: Distributed-Shared-Memory-Systeme

Nachrichtengekoppelte Multiprozessoren

Unterscheidung

- Uniform-communication-architecture-Modell (UCA)
  - Zwischen allen Prozessoren können gleich lange Nachrichten mit einheitlicher Übertragungszeit geschickt werden
- Non-uniform-communication-architecture-Modell (NUCA)
  - Die Übertragungszeit des Nachrichtentransfers zwischen den Prozessoren ist je nach Sender- und Empfänger-Prozessor verschieden lang

Zugriffszeit-/Übertragungszeit-Modelle



UMA	Uniform Memory Access
NUMA	Nonuniform Memory Access
UCA	Uniform Communication Architecture
NUCA	Nonuniform Communication Architecture

Quantitative Maßzahlen für parallele Systeme

Die (Gesamt-)Ausführungszeit T (Execution Time) eines parallelen Programms ist die Zeit zwischen dem Starten der Programmausführung auf einem der Prozessoren bis zu dem Zeitpunkt, an dem der letzte Prozessor die Arbeit an dem Programm beendet hat.

Während der Programmausführung sind alle Prozessoren in einem der drei Zustände „rechnend“, „kommunizierend“ oder „untätig“.

Ausführungszeit T

Ausführungszeit T eines parallelen Programms auf einem dezidiert zugeordneten Parallelrechner setzt sich zusammen aus

- Berechnungszeit T_{comp} (Computation Time)
  - Zeit für Rechenoperationen
- Kommunikationszeit T_{comm} (Communication Time)
  - Zeit für Sende- und Empfangsoperationen
- Untätigkeitszeit T_{idle} (Idle Time)
  - Zeit für Warten (auf zu empfangende Nachrichten oder auf das Versenden)

Es gilt: $T = T_{comp} + T_{comm} + T_{idle}$

Übertragungszeit einer Nachricht T_{msg}

Die Zeit, die für das Verschicken einer Nachricht einer bestimmten Länge zwischen zwei Prozessen (oder Prozessoren) benötigt wird.

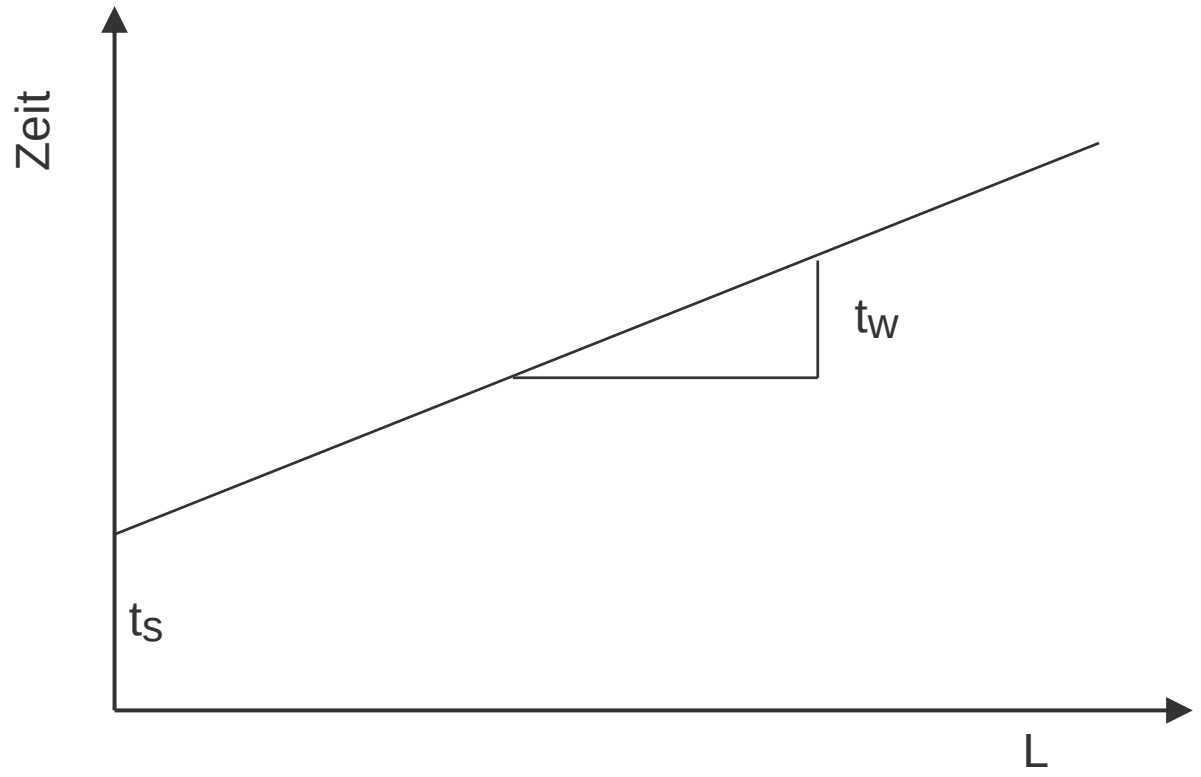
Die Übertragungszeit setzt sich zusammen aus

- Startzeit t_s (Message Startup Time),
  - Initiierung der Kommunikation

- und Transferzeit t_w pro übertragenem Datenwort
  - Abhängig von der physikalischen Bandbreite des Kommunikationsmediums

Übertragungszeit einer Nachricht

Die Übertragungszeit einer Nachricht mit L Datenwörtern beträgt $T_{msg} = t_s + t_w L$



Definitionen

P(1): Anzahl der auszuführenden (Einheits-)Operationen des Programms auf einem Einprozessorsystem.

P(n): Anzahl der auszuführenden (Einheits-)Operationen des Programms auf einem Multiprozessorsystem mit n Prozessoren.

T(1): Ausführungszeit auf einem Einprozessorsystem in Schritten (oder Takten).

T(n): Ausführungszeit auf einem Multiprozessorsystem mit n Prozessoren in Schritten (oder Takten).

Es gelte

$T(1) = P(1)$

- da in einem Einprozessorsystem (Annahme: einfacher Prozessor) jede (Einheits-)Operation in genau einem Schritt ausgeführt werden kann.

$T(n) \leq P(n)$

- da in einem Multiprozessorsystem mit n Prozessoren ($n \geq 2$) in einem Schritt mehr als eine (Einheits-)Operation ausgeführt werden kann.

Beschleunigung und Effizienz

Beschleunigung (Leistungssteigerung, Speed-up)

$$S(n) = \frac{T(1)}{T(n)}$$

Normal: $1 \leq S(n) \leq n$
Leistungsabfall: $S(n) < 1$
Synergie: $S(n) > n$

Effizienz

$$E(n) = \frac{S(n)}{n}$$

Typisch: $1/n \leq E(n) \leq 1$

T(1) Ausführungszeit auf einem Einprozessorsystem in Schritten (oder Takten)

T(n) Ausführungszeit auf einem Multiprozessorsystem mit n Prozessoren in Schritten (oder Takten)

Abhängigkeit der Beschleunigung

Die Begriffe *Beschleunigung* und *Effizienz* lassen sich „algorithmenunabhängig“ oder „algorithmenabhängig“ definieren.

absolute Beschleunigung (absolute Effizienz)

- Man setzt den besten bekannten sequentiellen Algorithmus für das Einprozessorsystem in Beziehung zum vergleichbaren parallelen Algorithmus für das Multiprozessorsystem.

relative Beschleunigung (relative Effizienz)

- Man benutzt den parallelen Algorithmus so, als sei er sequentiell, und misst dessen Laufzeit auf einem Einprozessorsystem.
- Der für die Parallelisierung erforderliche Zusatzaufwand an Kommunikation und Synchronisation kommt „ungerechterweise“ auch für den sequentiellen Algorithmus zum Tragen.

Skalierbarkeit eines Parallelrechners

Hinzufügen von weiteren Verarbeitungselementen führt zu einer kürzeren Gesamtausführungszeit, ohne dass das Programm geändert werden muss.

Insbesondere meint man damit eine lineare Steigerung der Beschleunigung mit einer Effizienz nahe bei Eins.

Wichtig für die Skalierbarkeit ist eine angemessene Problemgröße.

Bei fester Problemgröße und steigender Prozessorzahl wird ab einer bestimmten Prozessorzahl eine Sättigung eintreten. Die Skalierbarkeit ist in jedem Fall beschränkt.

Skaliert man mit der Anzahl der Prozessoren auch die Problemgröße (*scaled problem analysis*), so tritt dieser Effekt bei gut skalierenden Hardware- oder Software-Systemen nicht auf.

Definitionen

Mehraufwand für die Parallelisierung:

$$R(n) = \frac{P(n)}{P(1)}$$

Dabei gilt:
(immer erhöhter Aufwand bei paralleler Programmierung)

$$1 \leq R(n)$$

Parallelindex I(n) (mittlerer Grad an Parallelität)

$$I(n) = \frac{P(n)}{T(n)}$$

Definitionen

Auslastung (utilization):

$$U(n) = \frac{I(n)}{n}$$

$$= R(n) \cdot E(n) = \frac{P(n)}{n \cdot T(n)}$$

- entspricht normiertem Parallelindex.

- gibt an, wie viele Operationen jeder Prozessor im Durchschnitt pro Zeiteinheit ausgeführt hat.

R(n): Mehraufwand für Parallelisierung
E(n): Effizienz

Folgerungen

Alle definierten Ausdrücke haben für n = 1 den Wert 1.

Der Parallelindex gibt eine obere Schranke für die Leistungssteigerung:

$$1 \leq S(n) \leq I(n) \leq n$$

Die Auslastung ist eine obere Schranke für die Effizienz:

$$\frac{1}{n} \leq E(n) \leq U(n) \leq 1$$

Zahlenbeispiel I

Ein Einprozessorsystem benötige für die Ausführung von 1000 Operationen 1000 Schritte.

Ein Multiprozessorsystem mit 4 Prozessoren benötige dafür 1200 Operationen, die aber in 400 Schritten ausgeführt werden können.

Damit gilt

- P(1) = T (1) = 1000, P(4) = 1200 und T(4) = 400

Daraus ergibt sich

- S(4) = 2.5 und E(4) = 0.625

Die Leistungssteigerung verteilt sich also zu 62,5% auf jeden Prozessor

Zahlenbeispiel II

I(4) = 3 und U(4) = 0.75

- Es sind im Mittel drei Prozessoren gleichzeitig tätig, d.h., jeder Prozessor ist nur zu 75% der Zeit aktiv.

R(4) = 1.2

- Bei Ausführung auf einem Multiprozessorsystem sind 20% mehr Operationen als bei Ausführung auf einem Einprozessorsystem notwendig.

Amdahls Gesetz (1967, Gene Amdahl)

$$T(n) = T(1) \cdot \frac{1-a}{n} + T(1) \cdot a$$

a = Anteil des nur sequentiell ausführbaren Programmteils

$$S(n) = \frac{T(1)}{T(n)} = \frac{T(1)}{T(1) \cdot \frac{1-a}{n} + T(1) \cdot a}$$

$$= \frac{1}{\frac{1-a}{n} + a} = \frac{n}{(1-a) + n \cdot a}$$

$$S(n) \leq \frac{1}{a}$$

Bem.: Synchronisation und Kommunikation vernachlässigt!

Diskussion

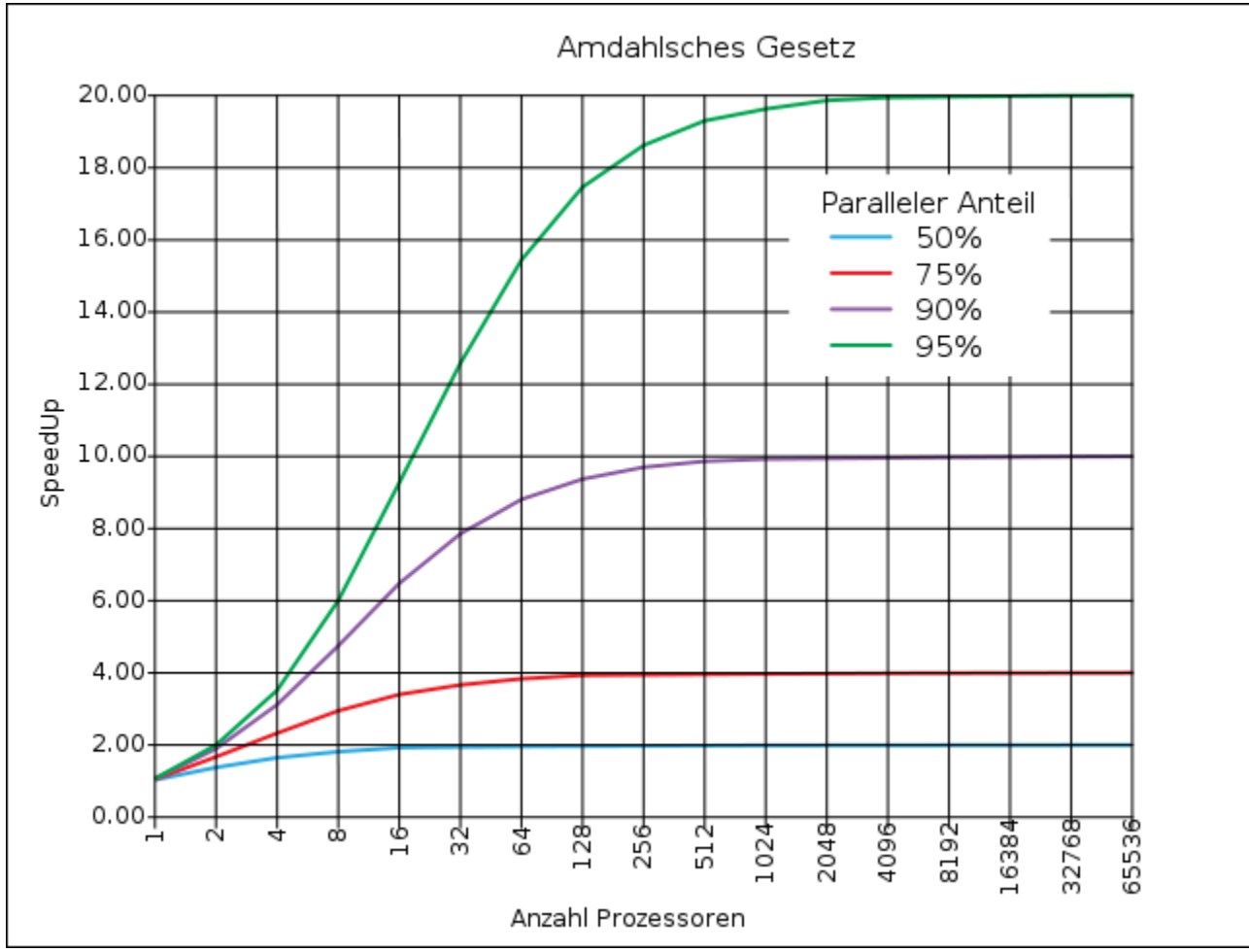
Amdahls Gesetz zufolge kann eine kleine Anzahl von sequenziellen Operationen die mit einem Parallelrechner erreichbare Beschleunigung signifikant begrenzen.

Beispiel: a = 1/10 des parallelen Programms kann nur sequenziell ausgeführt werden,
➔ das gesamte Programm kann maximal zehnmal schneller als ein vergleichbares, rein sequenzielles Programm sein.

... ein großes Argument gegen Multiprozessorsysteme!

Jedoch:

- viele parallele Programme haben einen sehr geringen sequenziellen Anteil (a«1)
- Cache nicht berücksichtigt



„AmdahlsLaw german“ von Schwammerl-Bob - Derived from http://en.wikipedia.org/wiki/File:AmdahlsLaw.svg. Lizenziert unter CC BY-SA 3.0 über Wikimedia Commons – https://commons.wikimedia.org/wiki/File:AmdahlsLaw_german.svg#/media/File:AmdahlsLaw_german.svg

Synergetischer Effekt oder superlinearer Speed-up (S>n, E>1)

Theorie: einen superlinearen Speed-up kann es nicht geben

- Jeder parallele Algorithmus lässt sich auf einem Einprozessorsystem simulieren, indem in einer Schleife jeweils der nächste Schritt jedes Prozessors der parallelen Maschine emuliert wird.

Ein „superlinearer Speed-up“ kann real beobachtet werden z.B. bei

- Beim Programmlauf auf einem Rechner passen die Daten nicht in den Hauptspeicher des Rechners (häufiger Seitenwechsel), aber:
  bei Verteilung auf die Knoten des Multiprozessors können die parallelen Programme vollständig in den Cache- und Hauptspeichern der einzelnen Knoten ablaufen.

Folgerung

Einprozessor- und Mehrprozessorrechner sind nicht direkt vergleichbar, da bei n Prozessoren häufig die n-fache Speichergröße vorhanden ist.

Durch eine größere Anzahl von Prozessoren kann ein Problem schneller ausgeführt werden und durch eine Vergrößerung der Speicherkapazität können größere Probleme gerechnet werden können.

Die Theorie geht von unendlichem Speicher aus – die Praxis kann Superlinearität aufweisen!

Potentielle Probleme bei Multiprozessoren

Der Verwaltungsaufwand (overhead), der mit der Zahl der zu verwaltenden Prozessoren ansteigt, die Möglichkeit von Systemverklemmungen (deadlocks) und die Möglichkeit von Sättigungserscheinungen, die durch Systemengpässe (bottlenecks) verursacht werden.

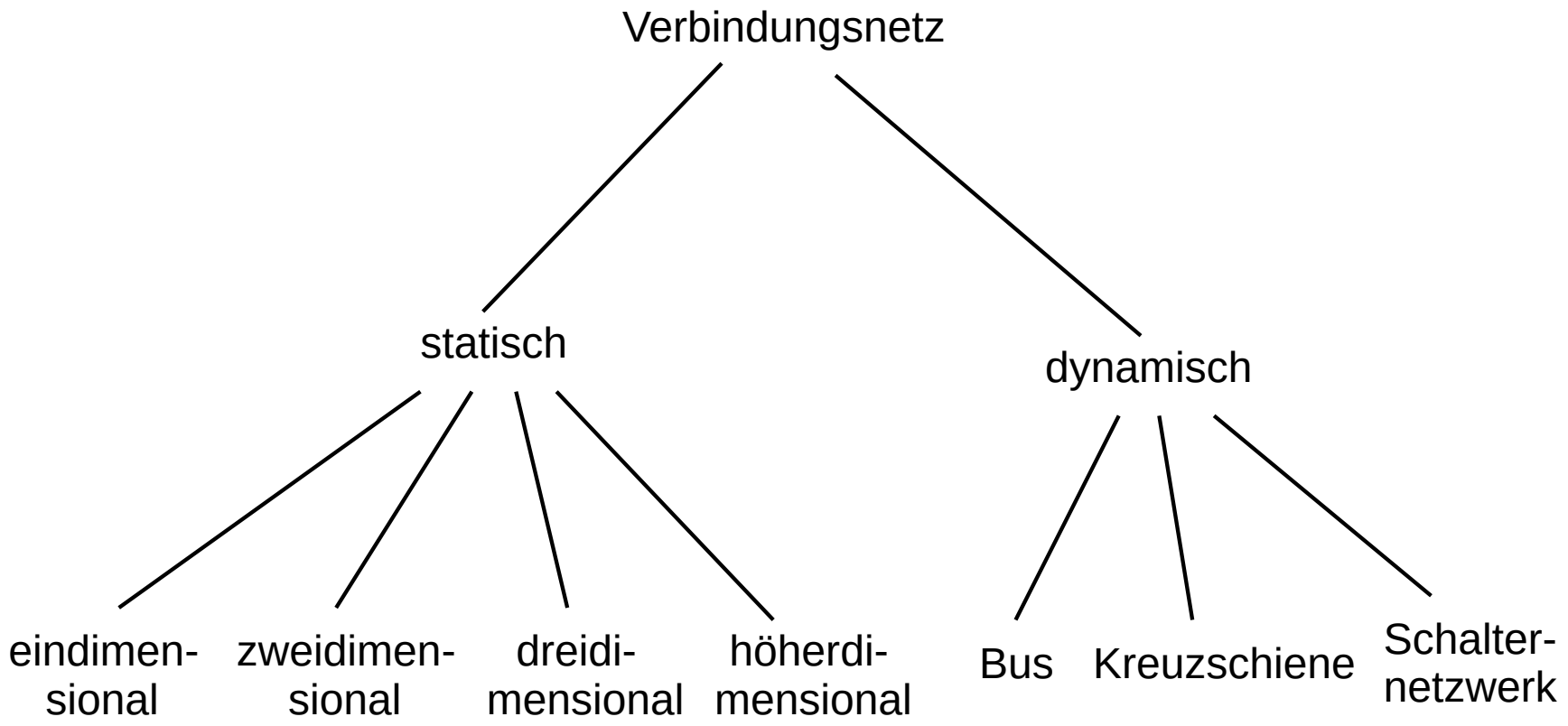
Verbindungsnetze

Verbindung zwischen Prozessoren und/oder Speicher

Statische Netze
- Fest installierte Verbindungen zwischen Paaren von Netzknoten

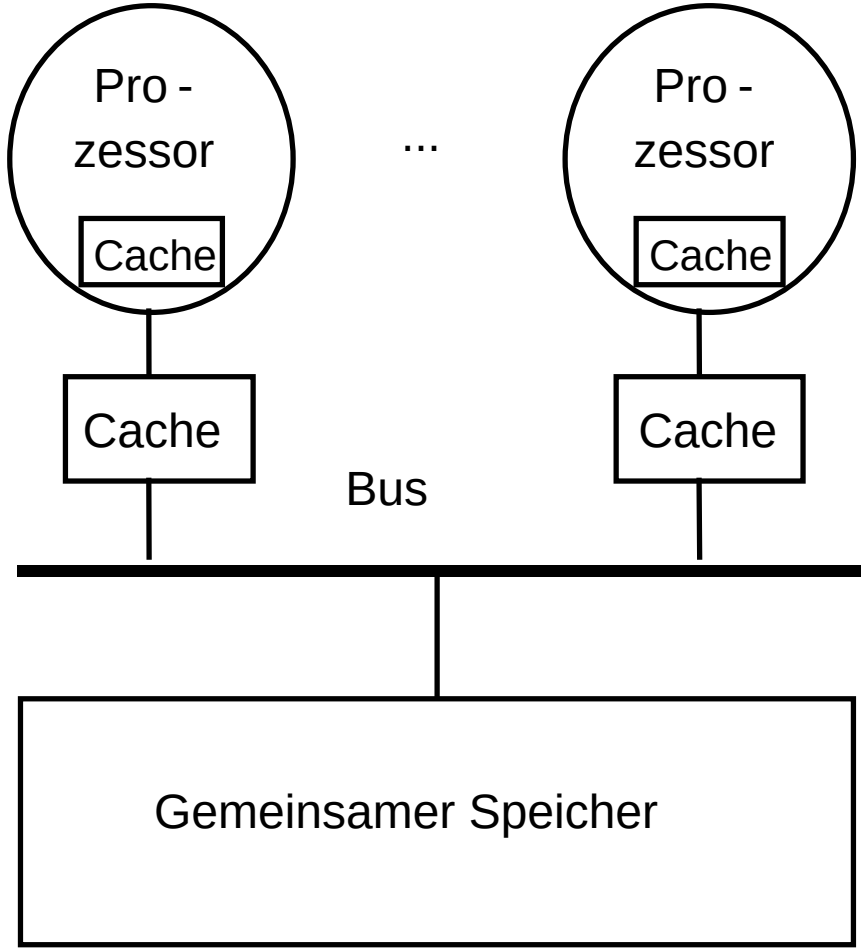
Dynamische Netze
- Enthalten eine Komponente „Schaltnetz“, an die alle Knoten über Ein- und Ausgänge angeschlossen sind
- Direkte fest installierte Verbindungen zwischen den Knoten existieren nicht.

Klassifizierung der Verbindungsnetze



Dynamische Verbindungsnetze: Bus

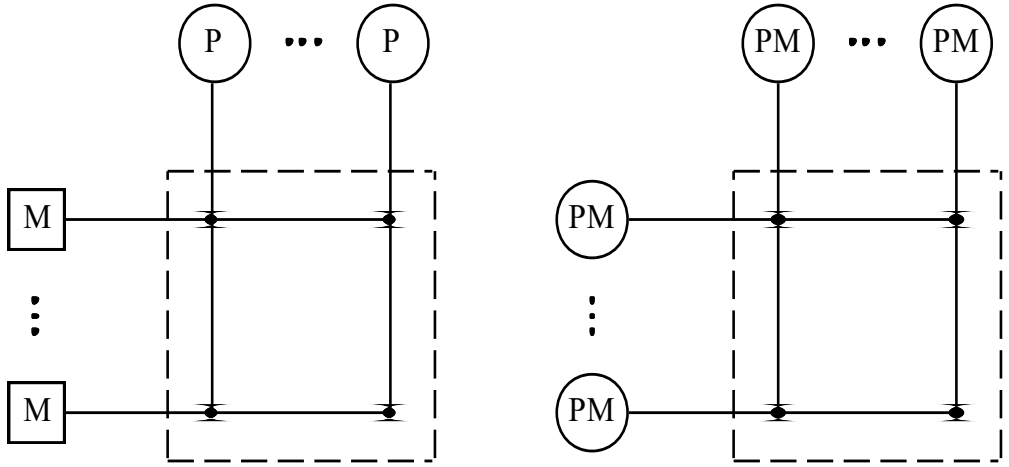
Speichergekoppelter Multiprozessor mit einem Einfachbus und Cache-Speichern



Dynamische Verbindungsnetze: Kreuzschiene

Kreuzschiene oder Kreuzschienenverteiler (crossbar switch)

- Hardware-Einrichtung, die so geschaltet werden kann, dass in einer Menge von Prozessoren alle möglichen disjunkten Paare von Prozessoren gleichzeitig und blockierungsfrei miteinander kommunizieren können.
- In Abhängigkeit vom Zustand der Schaltelemente im Kreuzschienenverteiler können dann je zwei beliebige Elemente aus den verschiedenen Mengen miteinander kommunizieren.



Cache-Kohärenzproblem (Cache Coherency Problem)

Gültigkeitsproblem, das beim Zugriff mehrerer Verarbeitungselemente auf Speicherworte des Hauptspeichers entsteht.

Kohärenz

- Korrektes Voranschreiten des Systemzustands durch ein abgestimmtes Zusammenwirken der Einzelzustände

Im Zusammenhang mit dem Cache muss das System dafür sorgen, dass immer die aktuellen Daten und nicht die veralteten Daten gelesen werden.

Ein System ist konsistent, wenn immer alle Kopien eines Datums im Hauptspeicher und den verschiedenen Cachespeichern identisch sind.

- Dadurch ist auch die Kohärenz sichergestellt, jedoch entsteht ein hoher Aufwand.

Warum Unterscheidung?

Eine Inkonsistenz zwischen Cache-Speicher und Hauptspeicher entsteht dann, wenn ein Speicherwort nur im Cache-Speicher und nicht gleichzeitig im Hauptspeicher verändert wird.

Dieses Verfahren nennt man Rückschreibeverfahren (copy-back oder write-back cache policy) im Gegensatz zum Durchschreibeverfahren (write-through cache policy).

Um alle Kopien eines Speicherworts immer konsistent zu halten, müsste ein hoher Aufwand getrieben werden.

Trick

- In begrenzten Umfang die Inkonsistenz der Daten zulassen.
- Das Cache-Kohärenzprotokoll sorgt dafür, dass die Cache-Kohärenz gewährleistet ist.
- Das Protokoll muss sicherstellen, dass immer die aktuellen und nicht die veralteten Daten gelesen werden.

Ansätze für Cache-Kohärenzprotokolle

Write-update-Protokoll

- Beim Verändern einer Kopie in einem Cache-Speicher müssen alle Kopien in anderen Cache-Speichern ebenfalls verändert werden, wobei die Aktualisierung auch verzögert (spätestens beim Zugriff) erfolgen kann

Write-invalidate-Protokoll

- Vor dem Verändern einer Kopie in einem Cache-Speicher müssen alle Kopien in anderen Cache-Speichern für „ungültig“ erklärt werden

Üblicherweise wird bei symmetrischen Multiprozessoren ein Write-invalidate-Cache-Kohärenzprotokoll mit Rückschreibeverfahren angewandt.

Das MESI-Protokoll

Bus-Schnüffeln

- Mithören der Lese-/Schreibzugriffe anderer Prozessoren am gemeinsamen Bus

Schnüffel-Logik jedes Prozessors hört die Adressen mit, die andere Prozessoren auf den Bus legen

Bei Übereinstimmung der Adressen mit Adressen der Cache-Blöcke im Speicher

- Erschnüffelter Schreibzugriff, bisher nur gelesen
  - Cache-Block wird für ungültig erklärt
- Erschnüffelter Lese-/Schreibzugriff, Kopie im Cache wurde verändert
  - Schnüffel-Logik übernimmt Bus-Transaktion, schreibt Wert aus Cache in Hauptspeicher zurück, lässt dann erst ursprünglichen Zugriff zu

MESI - das am meisten verwendete Write-invalidate-Protokoll (in verschiedenen Versionen)

Das MESI-Protokoll

Die Zustände der Cache-Lines beim MESI-Protokoll

- Exclusive modified
  - Die Cache-Line wurde durch einen Schreibzugriff geändert und befindet sich ausschließlich in diesem Cache.

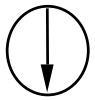
- Exclusive unmodified
  - Die Cache-Line wurde für einen Lesezugriff übertragen und befindet sich nur in diesem Cache.

- Shared unmodified
  - Kopien der Cache-Line befinden sich für Lesezugriffe in mehr als einem Cache.

- Invalid
  - Die Cache-Line ist ungültig.

Das MESI-Protokoll

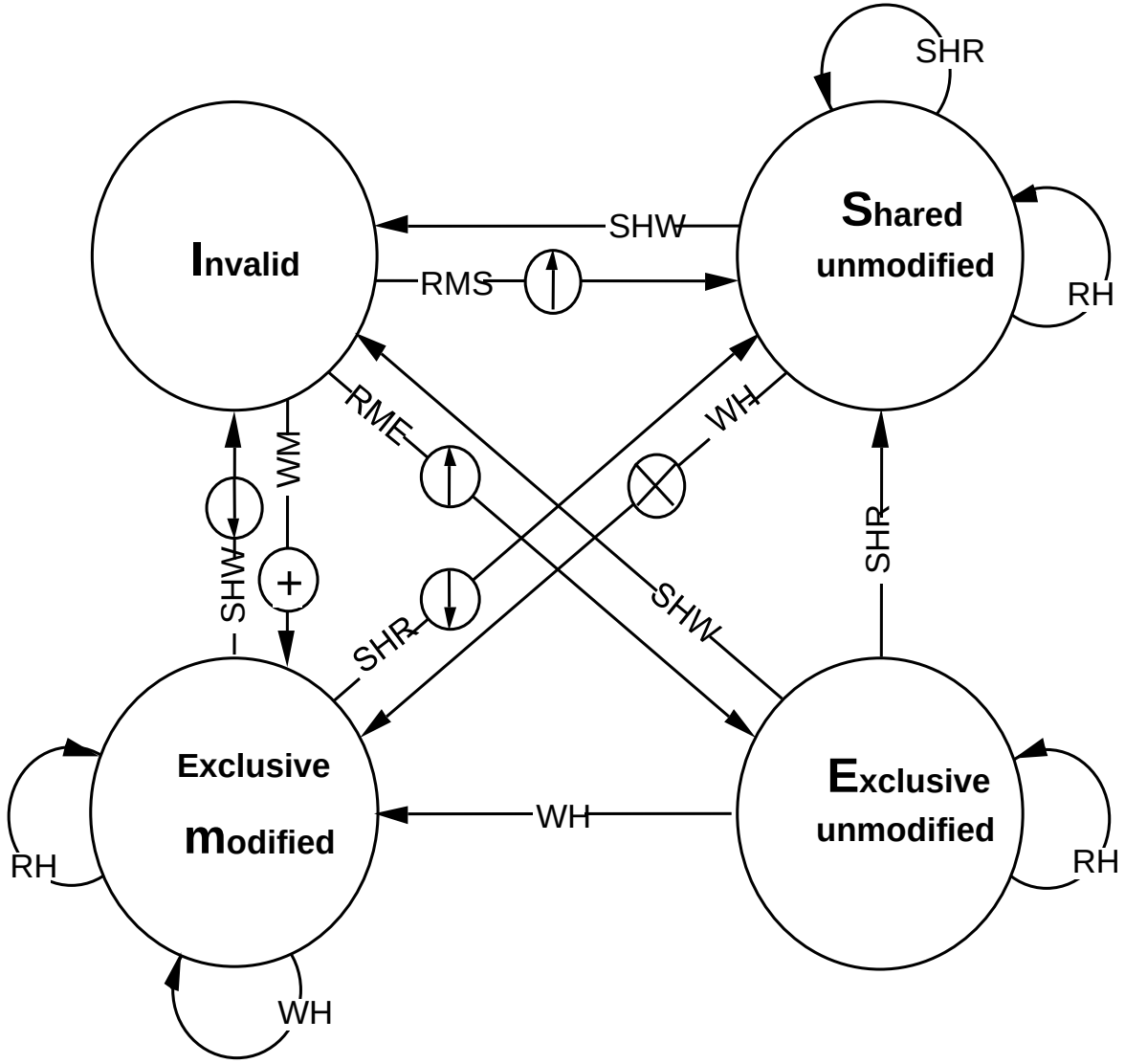
RH	Read hit
RMS	Read miss, shared
RME	Read miss, exclusive
WH	Write hit
WM	Write miss
SHR	Snoop hit on a read
SHW	Snoop hit on a write or read-with-intent-to modify


Dirty line copyback


Invalidate transaction


Read-with-intent-to-modify

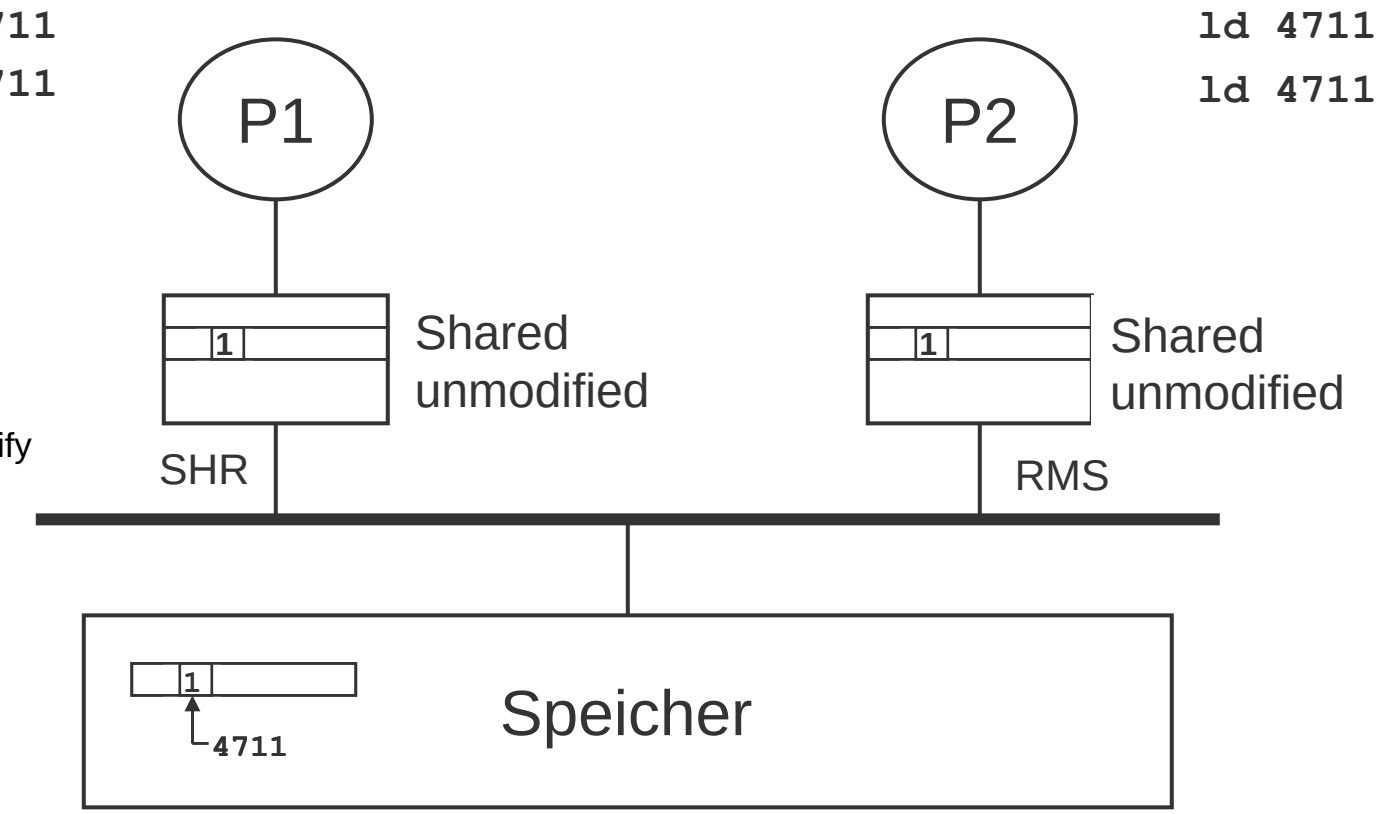

Cache line fill



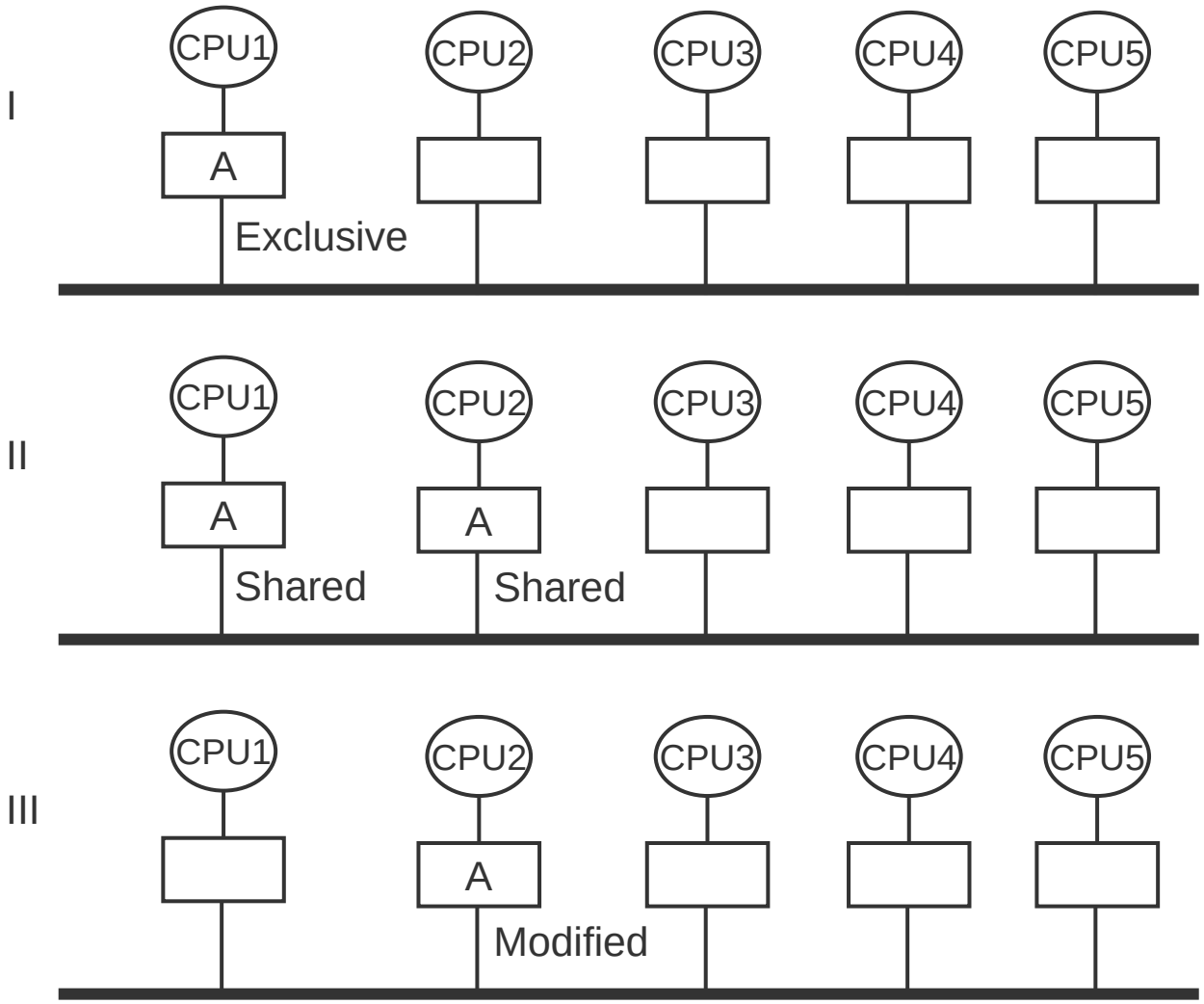
Beispiel zum MESI-Protokoll

```
ld 4711
st 4711
```

RH	Read hit
RMS	Read miss, shared
RME	Read miss, exclusive
WH	Write hit
WM	Write miss
SHR	Snoop hit on a read
SHW	Snoop hit on a write or read-with-intent-to modify



Weiteres Beispiel (Tanenbaum) I

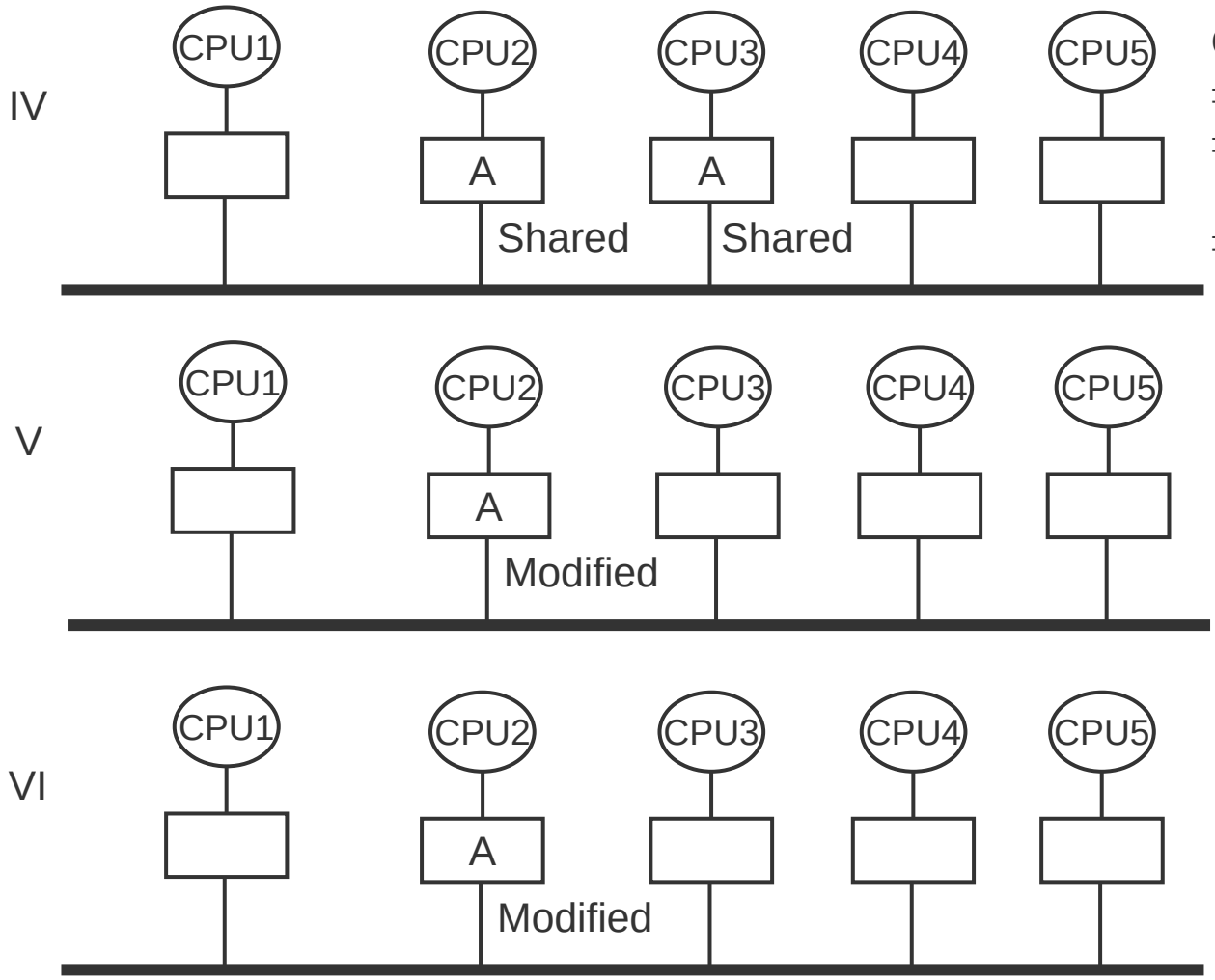


CPU 1 **liest** Block A in Cache ein

CPU 2 **liest** Block A
⇒ CPU 1 kündigt auf Bus an, dass sie auch Kopie hat

CPU 2 **schreibt** Block A (nur in Cache, nicht in HS!)
⇒ CPU 2 setzt invalid-Signal auf Bus

Weiteres Beispiel (Tanenbaum) II



CPU 3 **liest** Block A
⇒ CPU 2 erschnüffelt Bus-Zugriff
⇒ CPU2 lässt CPU3 warten (Bus-signal)
⇒ CPU2 schreibt, CPU3 liest

CPU 2 **schreibt** Block A

CPU 1 **schreibt** Block A
⇒ CPU2 lässt CPU2 warten
⇒ CPU2 schreibt Daten zurück
⇒ CPU1 kann Daten in Cache übernehmen (modified)

Distributed-shared-memory-Multiprozessoren

DSM (Distributed-shared-memory)-Multiprozessoren

- ein allen Prozessoren gemeinsamer Adressraum
- einzelne Speichermodule jedoch auf die einzelnen Prozessoren verteilt

Voraussetzung

- Der Zugriff eines Prozessors auf ein Datenwort in seinem lokalen Speicher geschieht schneller als der Zugriff auf ein Datenwort, das in einem der lokalen Speicher eines anderen Prozessors steht.
  $\Rightarrow$ Einordnung als NUMA

In der Regel werden zusätzliche Cache-Speicher (prozessorinterne Primär-Cache-Speicher und optionale Sekundär-Cache-Speicher) verwendet.

Diese können die Cache-Kohärenz über sämtliche Cache-Speicher des gesamten Systems sichern (CC-NUMA und COMA), oder sie puffern Cache-Blöcke nur im Falle eines prozessorlokalen Speicherzugriffs (NCC-NUMA).

CC-NUMA, COMA, NCC-NUMA (I)

CC-NUMA (Cache-Coherent Non-Uniform Memory Access):

- Caches über das gesamte System hinweg kohärent
- Bei Zugriff auf entfernte Daten (d.h. lokaler Cache-Miss) Übertragung eines entfernten Cache-Blocks in den lokalen Cache
- Cache-Kohärenzprotokoll benötigt (z.B. verzeichnisbasiert)
- Bsp.: SGI Origin, HP/Convex

COMA (Cache-Only Memory Architecture):

- Spezialfall von CC-NUMA: kein zentraler Hauptspeicher mehr
- Daten wandern durch das System, von Cache zu Cache (anfängliche Aufteilung verändert sich, i. Ggs. zu CC-NUMA)
- Bsp.: Data Diffusion Machine, KSR-2 (Experimentelle Architekturen)

CC-NUMA, COMA, NCC-NUMA (II)

NCC-NUMA (Non-Cache-Coherent Non-Uniform Memory Access)

- Keine globale Cache-Kohärenz, kein Cache-Kohärenz-Protokoll!
  - genauer: kein *globales* Cache-Kohärenz-Protokoll, wohl aber lokale Kohärenz zwischen lokalem Cache und Hauptspeicher

- Unterschiedliche Befehle für Zugriffe auf lokale Daten (im Cache) und entfernte Daten (auf anderen Caches oder im Hauptspeicher)

- Anfragen auf entfernte Daten am Cache vorbei

- Kohärenz des Programmablaufs muss durch Software (Barrieren, Mutexes) gesichert werden!

Zwei Arten von Distributed-shared-memory-Multiprozessoren

Der Zugriff geschieht in einer für das Maschinenprogramm transparenten Weise oder durch explizite Befehle, die nur dem entfernten Speicherzugriff dienen.

Die erste Variante ist bei CC-NUMAs üblich, die zweite Variante bei NCC-NUMA-Systemen.

Im ersten Fall muss Spezial-Hardware anhand der Adresse zwischen einem prozessorlokalen und einem entfernten Speicherzugriff unterscheiden.

Im zweiten Fall wird der Maschinenbefehlssatz des Prozessors erweitert.

Nachrichtengekoppelte Multiprozessoren

Bei den nachrichtengekoppelten Systemen gibt es keine gemeinsamen Speicher- oder Adressbereiche.

Die Kommunikation geschieht durch Austauschen von Nachrichten über ein Verbindungsnetz.

Alle Prozessoren besitzen nur lokale Speicher.

Prozessorknoten sind üblicherweise durch serielle Punkt-zu-Punkt-Verbindungen gekoppelt.

Die Skalierbarkeit ist bei nachrichtengekoppelten Multiprozessoren im Prinzip unbegrenzt.

Parallelität lässt sich in effizienter Weise auf Programm- oder Taskebene nutzen.

Block- oder Anweisungsebenenparallelität sind mit heutiger Übertragungstechnologie nicht effizient nutzbar.

Alternative: Verteilte Systeme

Unter einem verteilten System versteht man eine Anzahl von vernetzten Rechnern, die zur Lösung einer gemeinsamen Aufgabe kooperieren können.

Vorteile verteilter Systeme:

- Rechner sind schon vorhanden und nur selten ausgelastet
  $\Rightarrow$ großes Rechenpotential.
- Vorhandene Ressourcen können durch verteiltes Rechnen besser ausgelastet werden.
- Keine Hardwareerweiterungen notwendig $\rightarrow$ kostengünstig.
- Die Verwendung aller Speicher der verwendeten Rechner hilft, lokalen Platzmangel zu verhindern.

Vorteile verteilter Systeme

Ein aus vernetzten Arbeitsplatzrechnern zusammengesetzter **virtueller Parallelrechner** kann problemlos um weitere Rechner erweitert werden.

Heterogene, vernetzte Arbeitsplatzrechner nutzbar.

Die Programmentwicklung kann auf gewohnten Rechnern durchgeführt werden.

PVM und MPI sind standardisierte nachrichtengekoppelte Programmierschnittstellen für verteilte Systeme und für Parallelrechner $\rightarrow$ Quellcode-Kompatibilität

Bestimmte Unterprogramme können auf speziell darauf zugeschnittenen Rechnern ausgeführt werden. Spezialrechner für Graphik, Datenbanken, etc.

Vorteile hinsichtlich der Fehlertoleranz

PVM: parallel virtual machine
MPI: message-passing interface

Nachteile

Erschwerte Gewährleistung der Sicherheit.

Heterogene Netze, gerätespezifische Anforderungen an Hard- und Software.

Aufsetzen auf einer hohen UNIX-Schnittstelle (UDP, TCP)

Relativ langsame Nachrichtenübermittlung über das lokale Netz $\Rightarrow$ nur grobkörnig parallele Programme mit wenig Kommunikation sind effizient einsetzbar.

- Hardware-gestützte Kommunikation: Parastation, Myrinet, SCI-Cluster…

Metacomputing

Metacomputer

- Integration eigenständiger und über eine Hochgeschwindigkeitsverbindung gekoppelter Rechner zu einem virtuellen System
- „Virtuelles Rechenzentrum“, „verteilter Maschinenraum“, der sich nach außen als ein homogener Rechner darstellt

Hypercomputing

- Kopplung von Workstations aus verschiedenen lokalen Netzen auch über weite Entfernungen hinweg

Grid Computing

- Konzept der koordinierten gemeinsamen Nutzung von Rechnern, Software und Daten sowie Lösung von Problemen in dynamischen, mehrere Institutionen umfassenden virtuellen Organisationen.

Other Applications of Computer Systems

Embedded Systems

- Hardware and Software are components of a bigger system, e.g. plant and process control, robotics, “anthropomatics”, ...

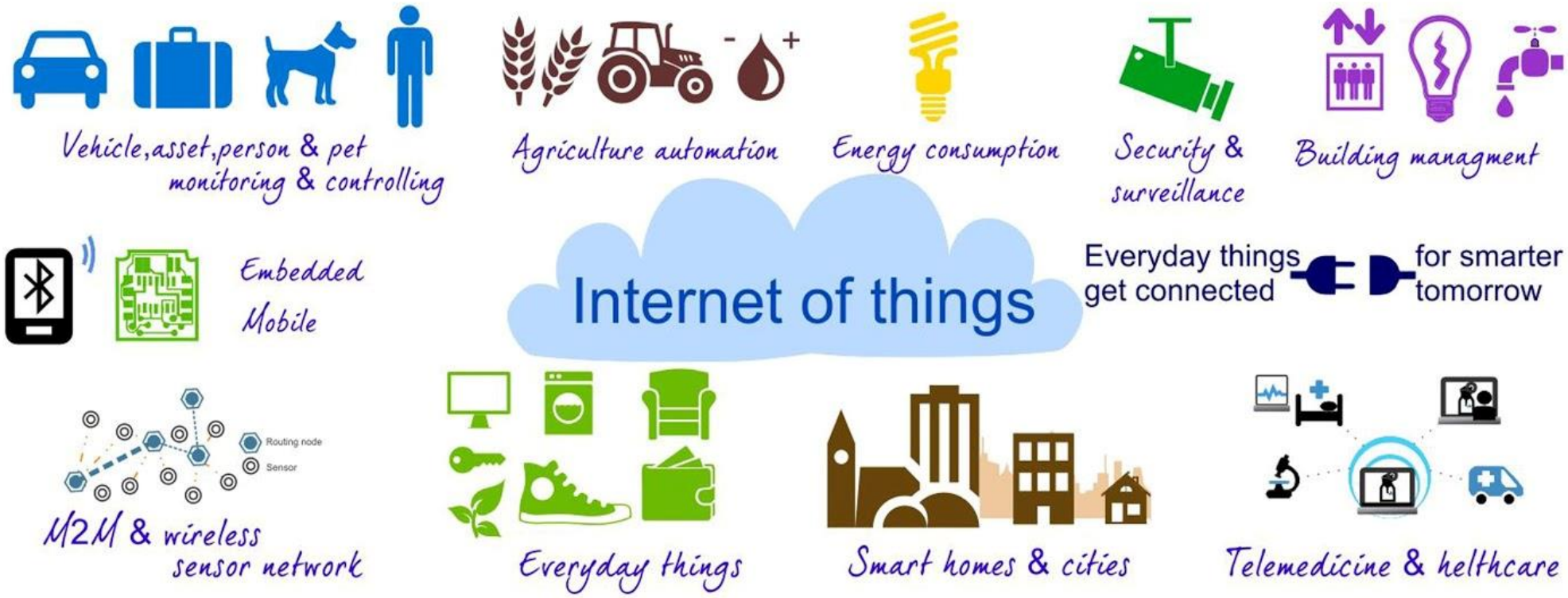
ICT (Information Communication Technology)

- Hardware and Software are main components of communication networks, multimedia, mobile phones, ...

Tools

- Use of Hardware and Software as support tools for technical/scientific applications: analysis, simulation and design of complex systems, gathering and processing of sensor data

Das Internet der Dinge

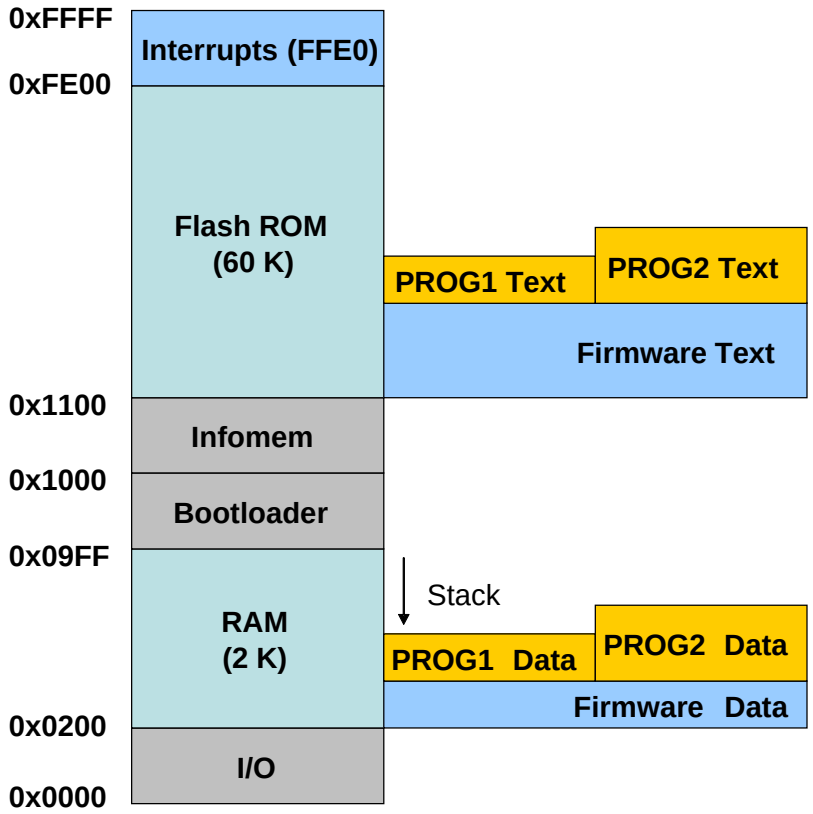


Quelle: The Telecare Blog, thetelecareblog.blogspot.de, 24.10.14

Example for a tight SW/HW integration

Separation into firmware and task
- Stable “core”
- SW updates stored in EEPROM first
- Flashing as second step
  - Synchronized (time or command)
  - Checksum

Task
- Linked against firmware
- Can use all functions
- Can register callback functions



Summary

Speicherhierarchie

Hauptspeicher

Cache-Speicher

Virtueller Speicher
- Paging
- Probleme des virtuellen Speichers

Speicher in Multiprozessorsystemen