TECHNISCHE UNIVERSITÄT DRESDEN

VLSI-EDA

Fakultät Informatik, Institut für Technische Informatik, Professur VLSI-Entwurfssysteme, Diagnostik und Architektur

Rechnerstrukturen und -organisation

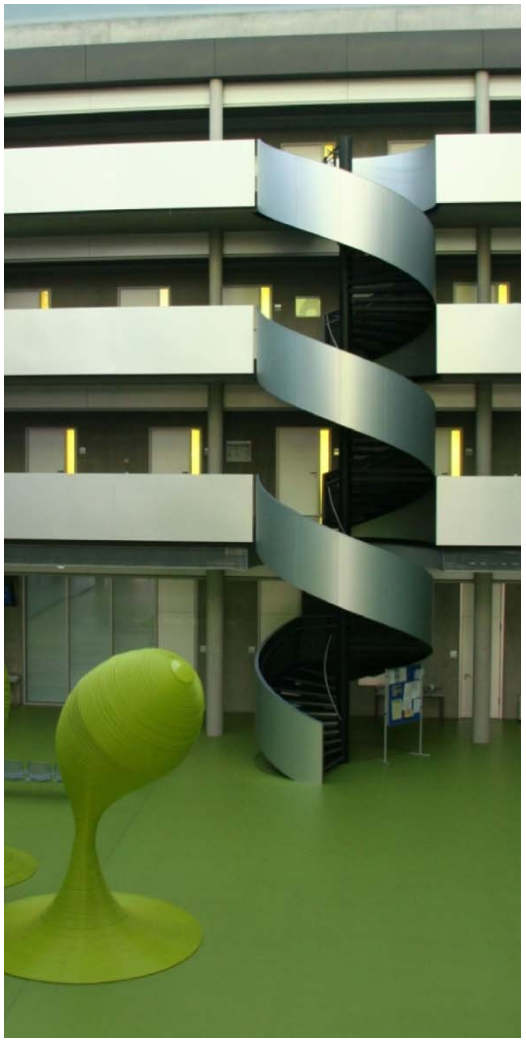
Speicherarchitektur

Rainer G. Spallek

TU Dresden, 13.01.2022

Rainer.Spallek@tu-dresden.de http://vlsi-eda.inf.tu-dresden.de

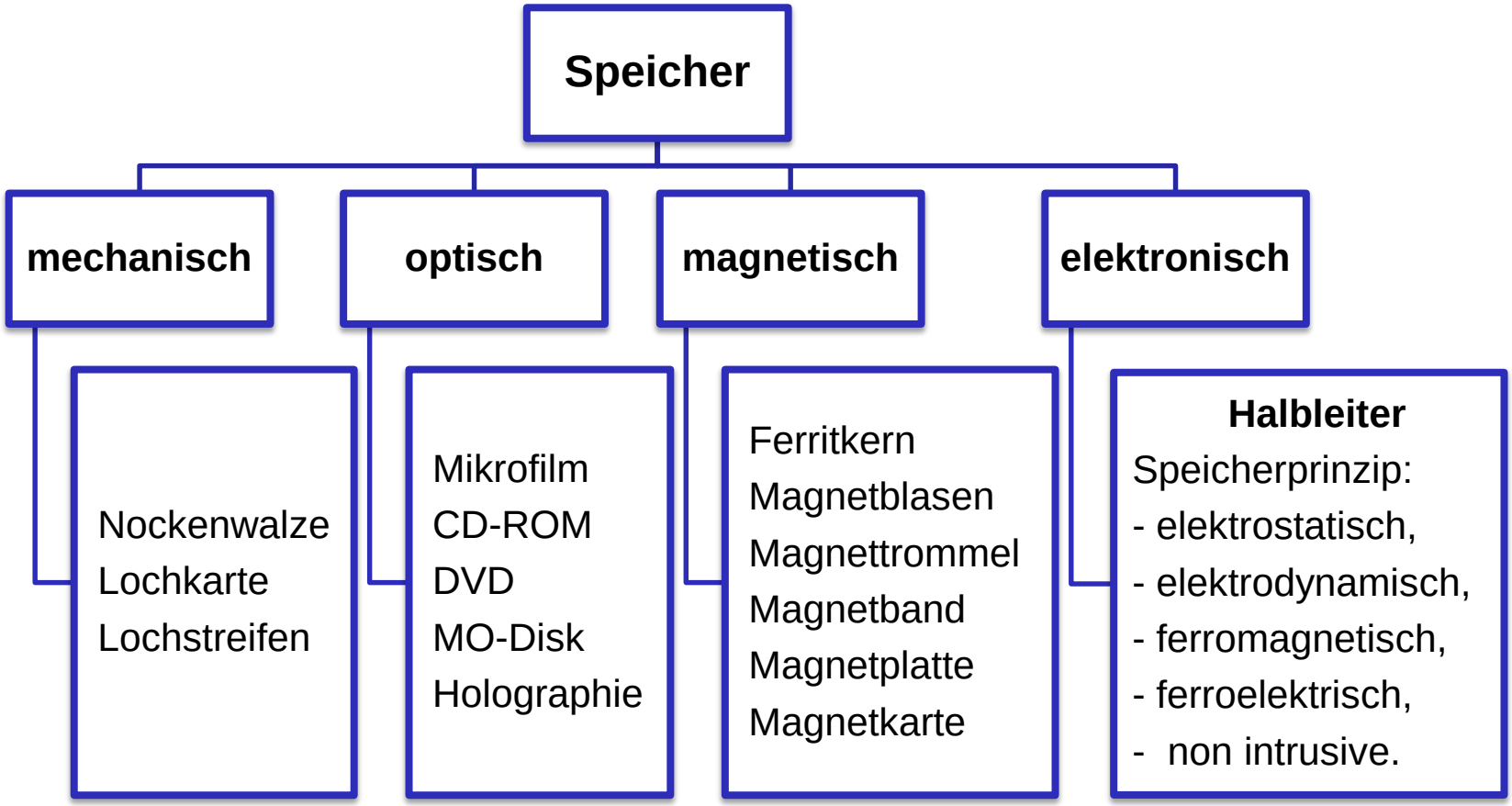
Gliederung



1 Zielstellung

- Der Hauptspeicher als zentraler Teil des von Neumann Rechners.
- Eigenschaften von Speichern, Anforderungen im Rechner.
- Übersicht über Speicher, speziell Halbleiterspeicher.
- Speicher innerhalb der CPU, Register, Stack.
- Wie kann der Gegensatz von Speicherkapazität und Zugriffszeit überwunden werden, neue Architekturkonzepte.
- Aufbau einer Rechnerarchitektur mit Speicherhierarchie, Speicherebenen.
- Struktur und Organisation von Cache-Architekturen, Assoziativität.
- Analyse und Lösung der Probleme bei Cache-Architekturen.
- Realisierung der Speicherkonsistenz trotz Caches.
- Realisierung effizienter Hauptspeicherkonzepte.

2 Speicherübersicht



Unterteilung nach physischer Haupteigenschaft (speichern, lesen, schreiben)

Anforderungen an Speicher

Persistente Speicherung:

Die Speicherzustände sind zeitlich stabil und reproduzierbar.
Die Speicherzustände bleiben auch ohne Energieversorgung erhalten.

Veränderbarkeit und Lesbarkeit:

Der Speicherzustand ist zu jedem beliebigen Zeitpunkt lesbar und je nach Anwendung ein- oder mehrmalig oder beliebig oft veränderbar.

Auswählbarkeit:

Die Speicherzustände sind einzeln auswählbar, ansprechbar und eindeutig unterscheidbar.

Bündelung:

Mehrere Speicherzustände sind zu einer Einheit, einem Block zusammengefasst und nur gemeinsam ansprechbar, lesbar oder änderbar.

Eigenschaften von Speichern

Zugriffsart:

- sequentiell, strukturiert — SAM (Serial Addressable Memory)
- inhaltsbezogen, assoziativ — CAM (Content Addressable Memory)
- wahlfrei adressierbar — RAM (Random Access Memory)

Änderbarkeit:

- fester Inhalt, nur lesbar — ROM (Read Only Memory)
- einmalig programmierbar, nur lesbar — PROM (Programmable ROM)
- meist lesbar, mehrmalig schreibbar — RMM (Read Mostly Memory)
- mehrfach programmierbar, löschbar, nur lesbar — EPROM (Erasable PROM)
- beliebig oft les- und schreibbar — RWM (Read Write Memory)

Persistenz, Zeitverhalten:

- statisch, flüchtig — SRAM (Static RAM)
- dynamisch, flüchtig — DRAM (Dynamic RAM)
- auch bei Spannungswegfall nicht flüchtig — NVRAM (Non-volatile RAM)

Komponenten von Speicher

Speicherelement:

Realisiert (physikalisch) die Speicherzustände (meist binär).

Speicherzelle:

Zusammenfassung von Speicherelementen, kleinste ansprechbare Einheit.

Zugriffsbreite:

Paralleler Zugriff auf mehrere Speicherzellen gleichzeitig (Bit, Byte, Wort, …).

Speicherwort:

Feste Anzahl von Speicherzellen, die gleichzeitig als Block aus dem Speicher gelesen bzw. in ihn geschrieben werden können (typisch 1 Wort).

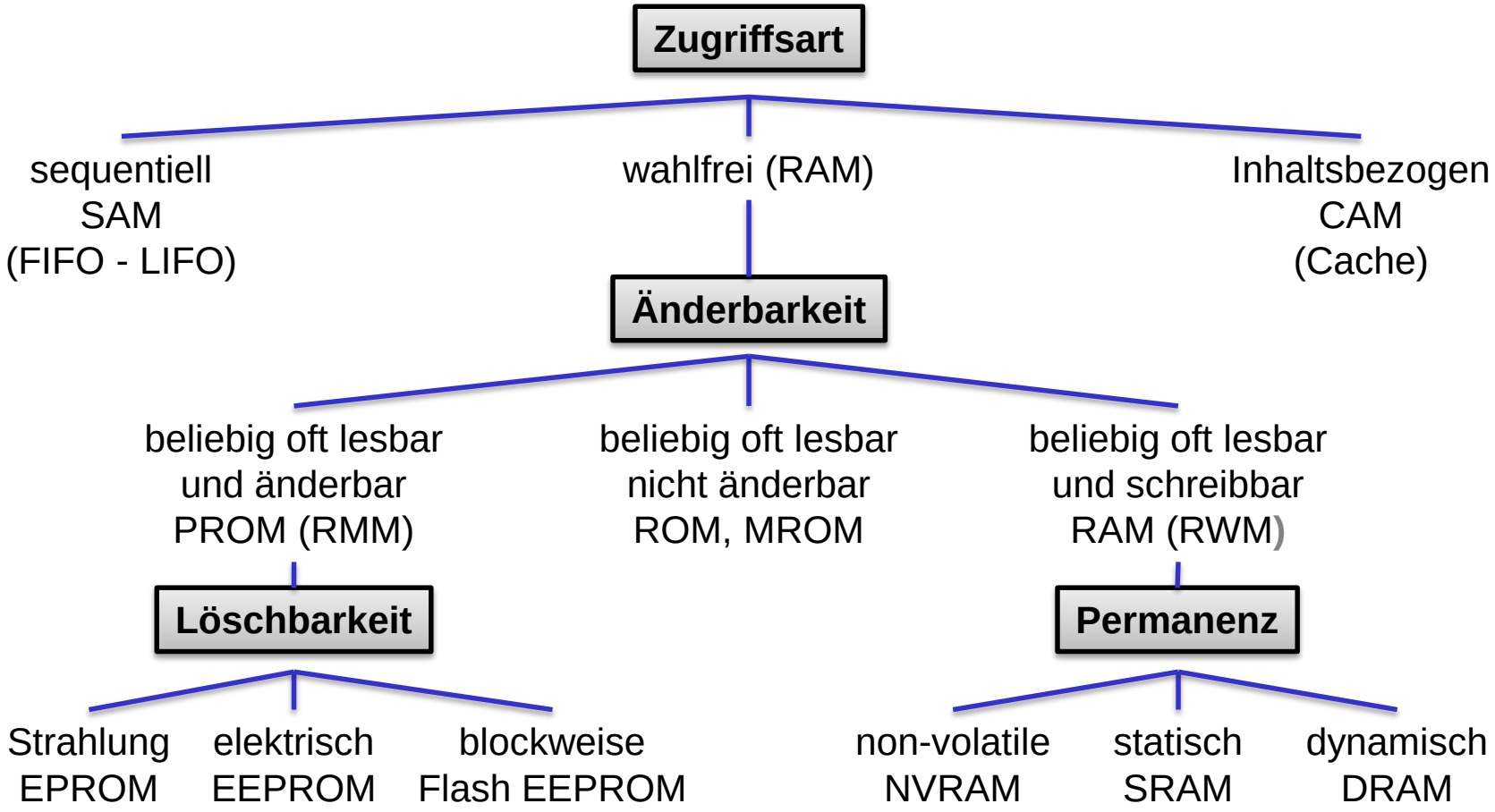
Speicherkapazität:

Anzahl der effektiv nutzbaren Speicherzellen.

Ansteuerung:

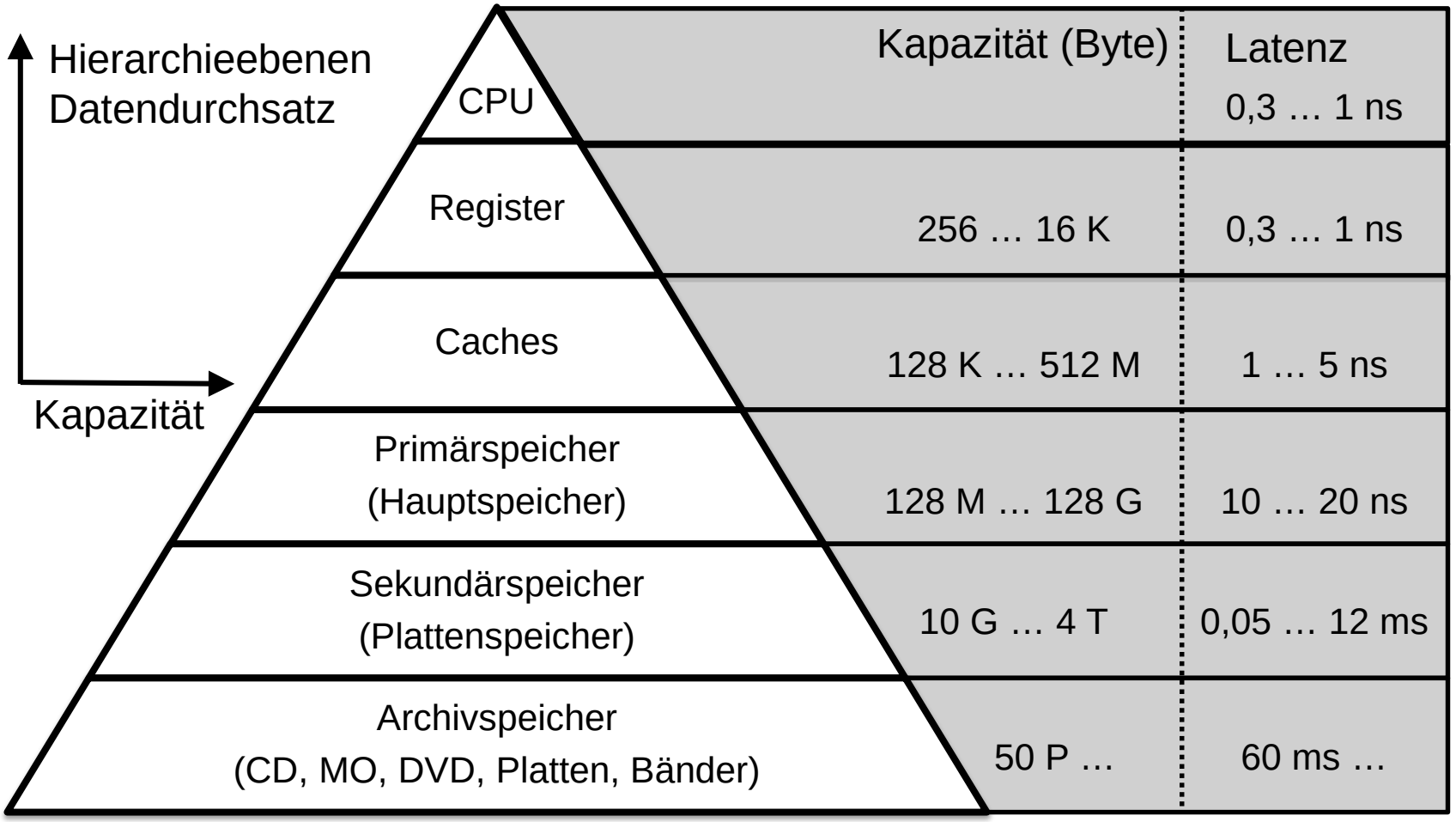
Taktsynchrone oder asynchrone Ansteuerung des Speichers.

Halbleiterspeicher



Alle ROM-Typen sind non-volatile

3 Speicherhierarchie im Computer



Technisch/Technologisches Problem Speicherkapazität ↔ Speicherzugriffszeit

Speicher mit kurzen Zugriffszeiten lassen sich nur mit relativ geringer Kapazität und mit hohem Kostenaufwand realisieren.

Dagegen lassen sich Speicher mit hoher Kapazität nur mit relativ langen Zugriffszeiten kostengünstig realisieren.

Mit der Entfernung von der CPU wachsen die Zugriffszeit und die Kapazität der Speicher gleichermaßen, die Realisierungskosten pro Byte sinken dagegen.

Zielstellung:

→ Erhöhung der Speicherbandbreite und Verringerung der Zugriffszeit bei gleichgroßer oder größerer Speicherkapazität (Realisierungsfrage).

→ Minimierung der Speicherkosten/Byte Speicherkapazität (Kostenfrage).

Lokalitätsprinzip des Speichers

Zeitliche Lokalität:

Nach einem Speicherzugriff ist die Wahrscheinlichkeit hoch, dass in einem der nächsten Befehle ein erneuter Zugriff auf denselben Speicherplatz erfolgt.

Örtliche Lokalität:

Nach einem Speicherzugriff ist die Wahrscheinlichkeit hoch, dass in einem der nächsten Befehle ein Zugriff auf einen der benachbarten Speicherplatz erfolgt.

Ursachen zum Lokalitätsprinzip:

Befehle: Sequenzieller Befehlsstrom, Programmschleifen, Unterprogramme, Abspeicherung in Bibliotheken, Programmierung, Compiler …

Daten: Zusammenhängende Datenstrukturen, Speicherzuteilung, Variablenanordnung durch Compiler, Felder, Zeichenketten …

90/10-Regel beim Speicher
90 % aller Speicherzugriffe erfolgen auf nur 10 % der Speicherplätze.

Konfliktlösung: Speicherkapazität ↔ Zugriffszeit

Lösung unter Ausnutzung des Lokalitätsprinzips:

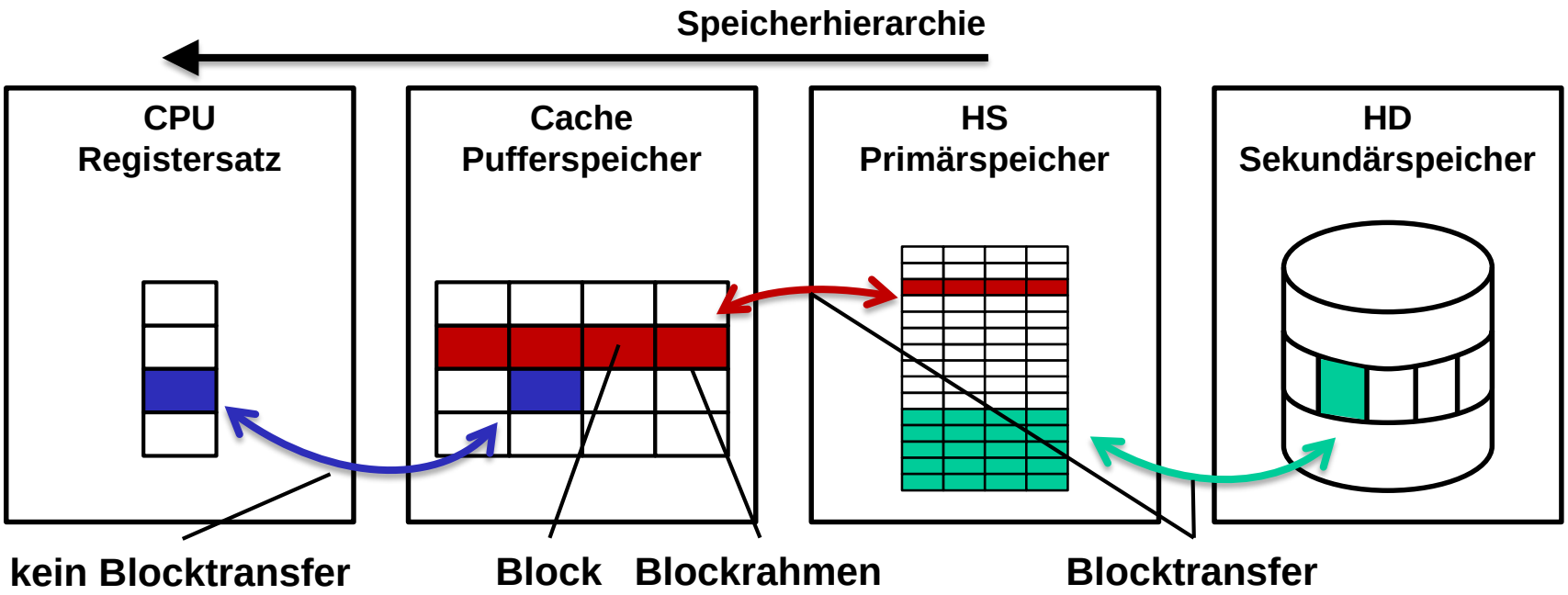
- Häufig benötigte Daten werden im nahen, kleinen und schnellen Speicher gehalten (als Kopien der Originaldaten).
- Seltener benötigte Daten werden im fernen, großen, jedoch langsamen Speicher gehalten (Originaldaten).
- Reicht die Kapazität des kleinen, schnellen Speichers nicht mehr aus, so werden nicht mehr benötigte Daten in den großen, langsamen Speicher ausgelagert und die neu benötigten Daten aus diesem nachgeladen.

→ **Aufbau einer Speicherhierarchie, Bildung von Hierarchieebenen**

Blockprinzip:

Der Austausch der Daten zwischen den einzelnen Hierarchieebenen erfolgt nicht einzeln, sondern zusammengefasst zu Blöcken (effektiverer Datentransfer). Blöcke werden in Blockrahmen gleicher Größe transportiert.

Speicherhierarchie – Blockprinzip



Kapazität/Zugriffszeit

Speicherbandbreite

Unterschiedliche Blockgrößen zwischen den Hierarchieebenen sind möglich.

Grundbegriffe der Speicherhierarchie

Block: Kleinster zusammenhängender Bereich, der zwischen den Hierarchieebenen transferiert, ausgetauscht wird (nicht notwendig gleich groß).

Blockrahmen (Frame): Bereich der höheren Hierarchieebene, in den ein Block eingelagert, transferiert werden kann.

Blocktransfer: Blockaustausch zwischen zwei Hierarchieebenen (Burst, …).

Treffer (Hit): Die gesuchten Daten befinden sich in einem Blockrahmen der höheren Hierarchieebene und sind aktuell.

Transferrate (Hit Rate): Relative Trefferanzahl bezogen auf die Gesamtzugriffe.

Fehlzugriff (Miss): Die gesuchten Daten wurden nicht in einem Blockrahmen der höheren Hierarchieebene gefunden bzw. sind nicht aktuell, kein Treffer.

Fehlerrate (Miss Rate): Relative Fehleranzahl (= 1 - Trefferrate).

Fehlerzuschlag (Miss Penalty): Zeit, die bei einem Fehlzugriff zusätzlich bis zur Erlangung der Daten benötigt wird.

Probleme der Speicherhierarchie

Abbildungsproblem:
Wie erfolgt die Abbildung der Blöcke einer Ebene auf die der nächst höheren?

Identifikationsproblem:
Wie werden die gesuchten Daten (Blöcke) lokalisiert und identifiziert?

Ersetzungsproblem:
Welcher Block wird beim Nachladen eines neuen ersetzt und wie?

Aktualisierungsproblem:
Wann und wie erfolgt die Aktualisierung der Blöcke in den einzelnen Ebenen bei einer Veränderung von Daten in einem Block?

Konsistenzproblem (Kohärenzproblem):
Die Daten der Blöcke einer Hierarchieebene sind konsistent in den Blöcken aller niederen Ebenen enthalten (Datenkonsistenz über alle Ebenen – mit Ausnahme des Registersatzes) → Aktualisierungsproblem.

4 Registersatz, Stack

Register:

- SRAM-Speicher (Flipflop-Kette) innerhalb der CPU (mit CPU-Takt getaktet).
- Nutzung als Universalregister oder als Spezialregister (Registertypen, …).
- Sonderfunktionen, auch verteilt innerhalb der CPU (Hilfsregister, …).
- Realisierung in verschiedenen Datenformaten (Halbwort, Word, …).
- Nutzung zur Rechnersteuerung (Befehlszähler, Statusregister, …).

Registersatz:

- Fest organisierter Satz von Registern innerhalb der CPU (8, 16, 32, …).
- Operandenspeicherung, Registerspeicher der ALU (Daten, Adressen, …).
- Register werden nicht adressiert, sondern direkt ausgewählt (Multiplexer).
- Ausführung als Multiport-RAM (z.B. 2 Leseports + 1 Schreibport, …).
- Aufwendige Hardware-Realisierung (Registerfenster, Schattenregister, …).

Übersicht Registersatz

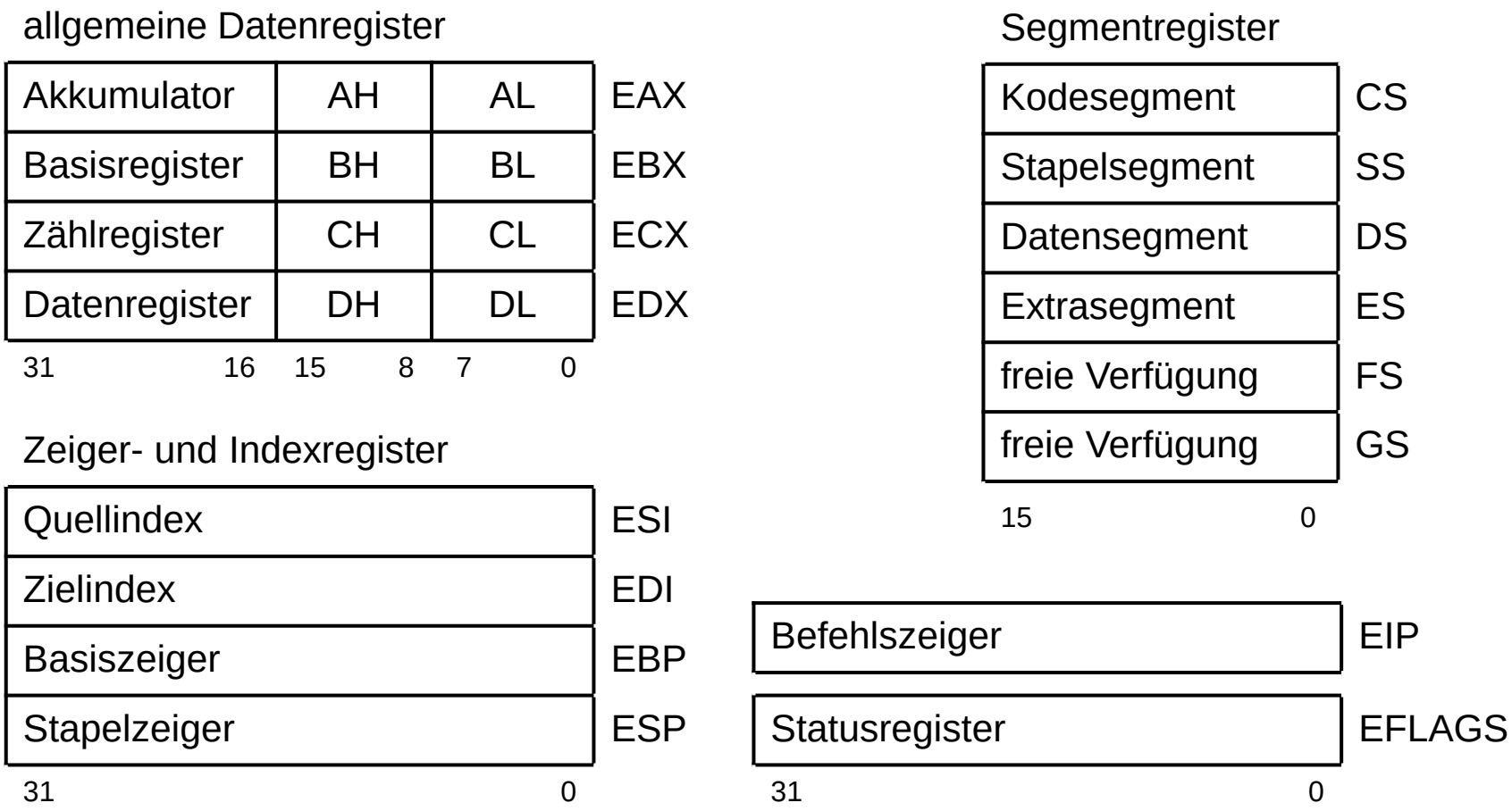
Registersatz

- Datenregister (data register)
  - Ganzzahlenregister (integer register)
  - Gleitkommazahlenregister (floating point register)
- Adressregister (address register)
  - Basisregister (pointer register)
  - Indexregister (displacement register)
  - Segmentregister (segment register)
- Spezialregister (special purpose register)
  - Programmzähler (program counter, PC)
  - Folgebefehl (instruction pointer, IP)
  - Statusregister (status register)
  - Basisregister (vector base register)
  - Stackregister (stack pointer)
  - Hilfsregister (auxiliary register)

Bei RISC zusammengefasst zum Universal-Registersatz (general purpose register)

→ **GPR-Architektur**

Der primäre Register des Pentium II (CISC)



Die allgemeinen Register der UltraSPARC II V9 (RISC)

Register	Andere Bezeichnung	Funktion
R0	G0	Fest auf 0 verdrahtet; Speicheroperationen darauf werden ignoriert.
R1 – R7	G1 – G7	Hält globale Variablen.
R8 – R13	O0 – O5	Hält Parameter für die aufzurufende Prozedur.
R14	SP	Stapelzeiger.
R15	O7	Arbeitsregister.
R16 – R23	L0 – L7	Hält lokale Variablen für die laufende Prozedur.
R24 – R29	I0 – I5	Hält ankommende Parameter.
R30	FP	Zeiger auf die Basis des laufenden Stapelrahmens.
R31	I7	Hält die Rückgabeadresse für die laufende Prozedur.

Erweiterungen des Registersatzes

Globale, lokale Register (global, local register):
Unterteilung des Registersatzes in globale und lokale Register (z.B. Variablenübergabe bei Unterprogrammtechnik).

Registerfenster (register window):
Unterteilung eines großen Registersatzes durch Registerfenster (konstante, variable Fenstergröße - überlappende, disjunkte Fenster). Lokalisierung des aktuellen Fensters durch Fensterzeiger (Current Window Pointer, *CWP*), der im Statusregister gespeichert wird (Reduktion des Registeradressraumes).

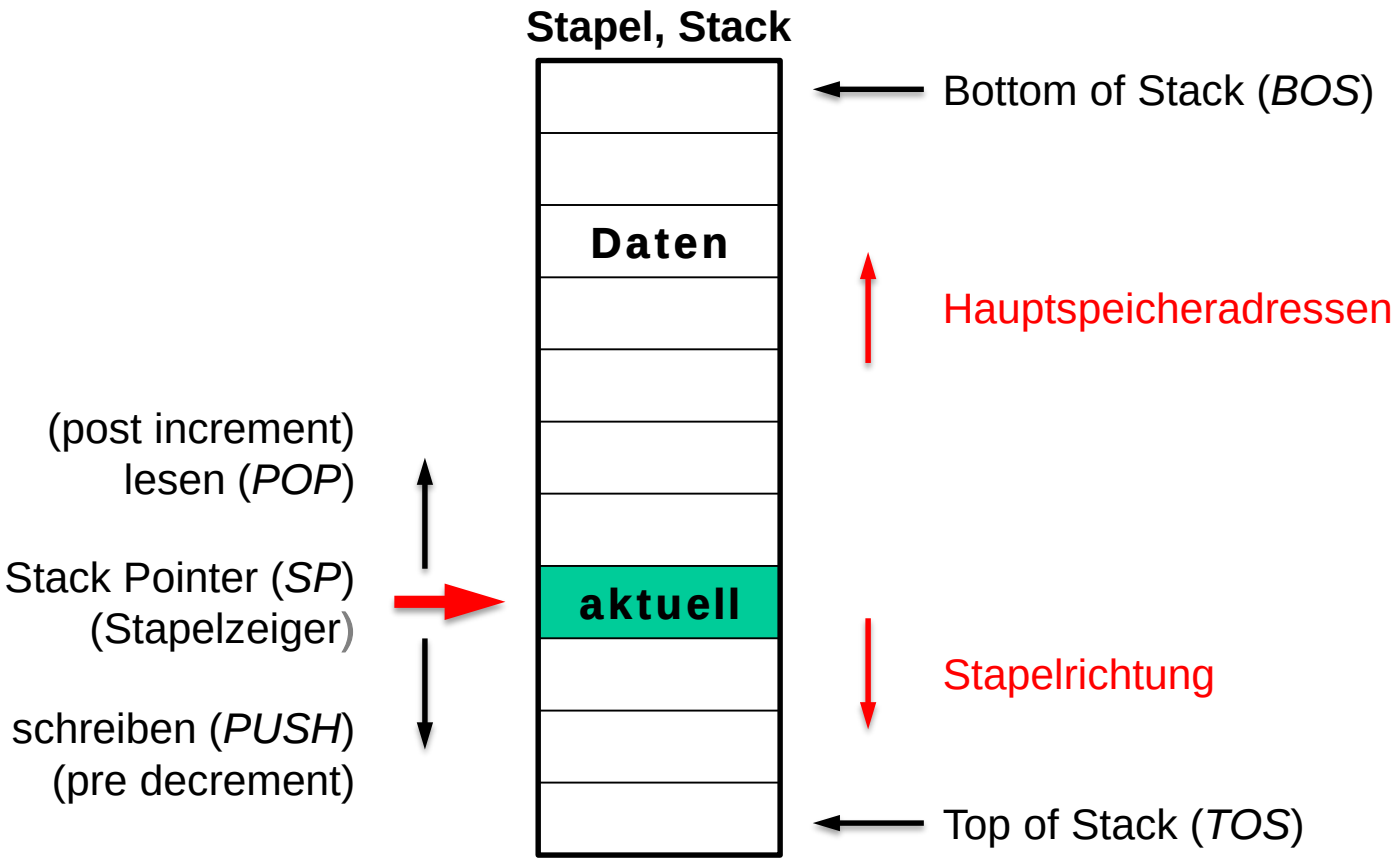
Schattenregister (shadow register):
Nutzung eines weiteren Registersatzes im Hintergrund (z.B. für spekulative Befehlsabarbeitung von Programmverzweigungen, Programmunterbrechungen).

Registerumbenennung (register renaming):
Umbenennung von Registern ohne den Inhalt zu transportieren (z.B. bei Ausführungs- oder Datenkonflikten in der Verarbeitungseinheit, out of order execution).

Stapelspeicher, Stack, Kellerspeicher

- SAM-Speicher (Serial Addressable Memory) nach dem LIFO-Prinzip (Last In First Out). Als Hardware-Stack innerhalb der CPU oder als Software-Stack im Hauptspeicher realisierbar.
- Der Stapelzeiger (Stack Pointer, *SP*) zeigt immer auf das Stapelelement, die letzten aktuellen Daten. Die Daten im Stack können nicht direkt adressiert werden.
- *PUSH* – Dekrementieren (erniedrigen) des Stack Pointers und Speichern der Daten auf den Stack (pre decrement – Stapelrichtung beachten).
- *POP* – Lesen der Daten vom Stack und anschließendes Inkrementieren (erhöhen) des Stack Pointers (post increment – Stapelrichtung beachten).
- Nutzung des Stacks zur effektiven Abspeicherung bzw. Zwischenspeicherung von Daten: Prozessorstatus, Unterprogrammparameter, rekursive Programme, Unterbrechungsroutinen,
- Stack-Typen: system stack, data stack, user stack, program stack, …

Verwaltung des Stapelspeichers



5 Cache-Speicher

- CAM-Speicher (Content Addressable Memory) als Pufferspeicher zur Überbrückung bzw. Anpassung stark unterschiedlicher Zugriffszeiten (z.B. Prozessorregister – Hauptspeicher – Festplatte).
- Der Cache arbeitet inhaltsorientiert, ist nicht direkt adressierbar. Der Vergleich mit dem Inhalt kann auch maskiert erfolgen (Ausblenden einzelner Bits). Ein Cache-Zugriff ist nicht immer eindeutig.
- Im Cache werden nur Kopien der aktuellen Speicherinhalte der darunter liegenden Hierarchieebene abgebildet. Die aktuellen Daten einer Hierarchieebene befinden sich auch immer in allen darunter liegenden.
- Zur Überbrückung sehr großer Differenzen in der Zugriffszeit bzw. im Datendurchsatz können auch mehrere Caches hintereinander geschaltet werden (Primary Cache, Secondary Cache, …).
- Der Cache-Speicher kann sich mit auf dem CPU-Chip befinden (On-Chip Cache) oder extern, außerhalb der CPU (Off-Chip Cache).

Cache-Begriffe

Adressspeicher (Tag-RAM):
Hauptspeicheradresse bzw. Adressteil des Datenblockes (Cache-Line).
Valid-Bit: Cache-Inhalt bzgl. der Adresse ist aktuell, gültig.
Dirty-Bit: Cache- und Hauptspeicherinhalt der Adresse sind nicht konsistent.

Datenspeicher (Data-RAM):
Datenblöcke (Cache-Line): Blöcke des Cache-Speichers, → Blockprinzip.
Datenblockgröße (Cache-Line-Size): Blockgröße in Worten (Byte).
Blockrahmen (Data-Frame): Nimmt den Datenblock auf

Cache-Treffer (Cache-Hit):
Daten/Befehle aktuell im Cache gefunden.
Trefferrate (Hit-Rate): Maß für die Effizienz des Caches.

Cache-Fehlzugriff (Cache-Miss):
Daten/Befehle nicht im Cache gefunden bzw. nicht aktuell.
Fehlerrate (Miss-Rate): 1 - Trefferrate.
Fehlerzuschlag (Miss-Penalty): Zugriffszeit nach Cache-Fehlzugriff.

Cache-Eigenschaften

Struktur:
Ein Cache-Eintrag besteht aus dem Datenblock (Cache-Line, → Blockprinzip) und dessen Adresse, Adressteil als Tag (Etikett) sowie Statusinformationen.

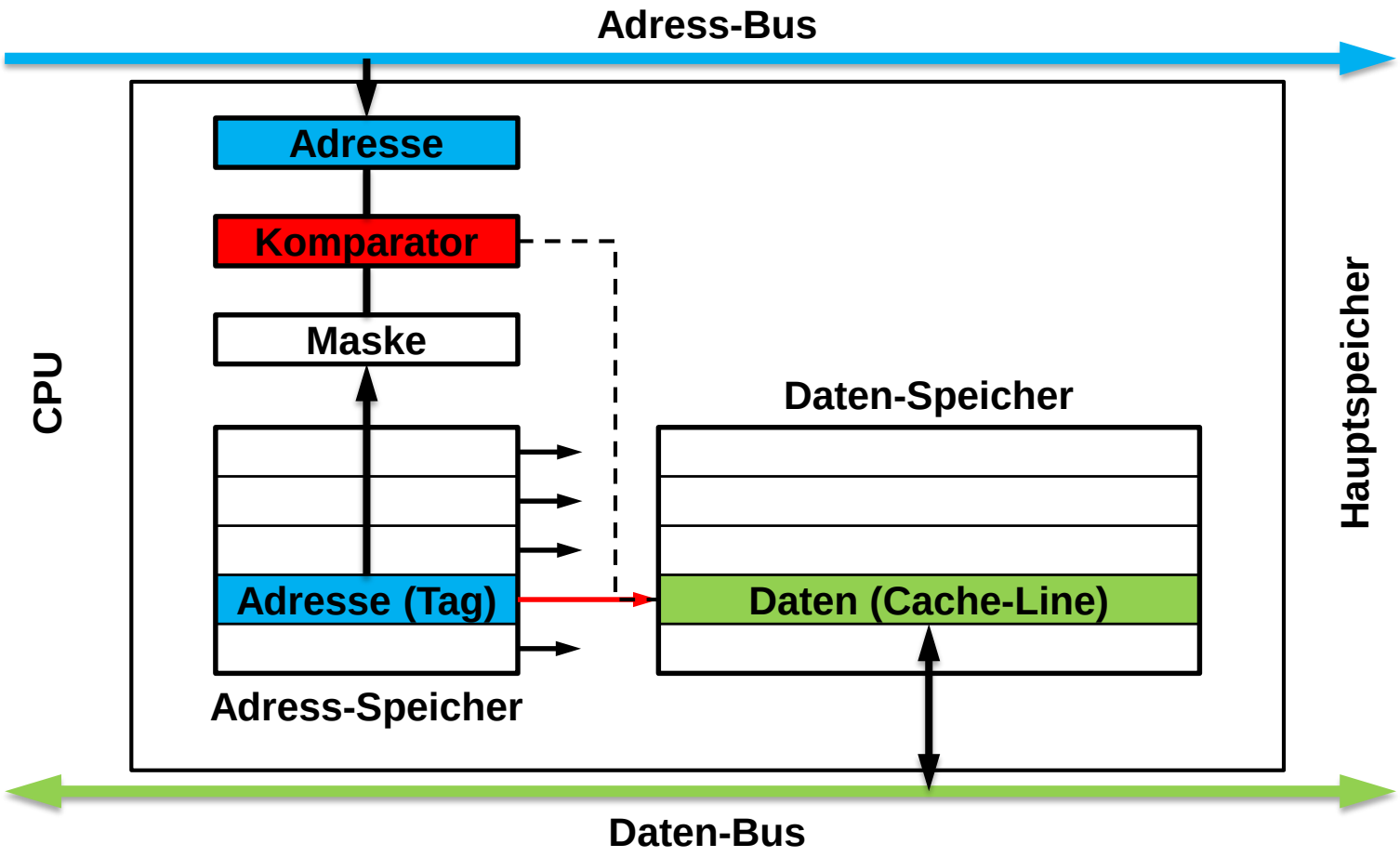
Sichtbarkeit:
Transparent (nicht sichtbar im Befehlssatz) oder nicht transparent (sichtbar, wird durch den Befehlssatz gesteuert, z.B. Laden, Invalidieren, …).

Adressraum:
Realer Adressraum (realer Cache) oder virtueller Adressraum (virtueller Cache).

Architektur:
Befehle und Daten gemeinsam (Unified Cache) oder getrennt (Split Cache).

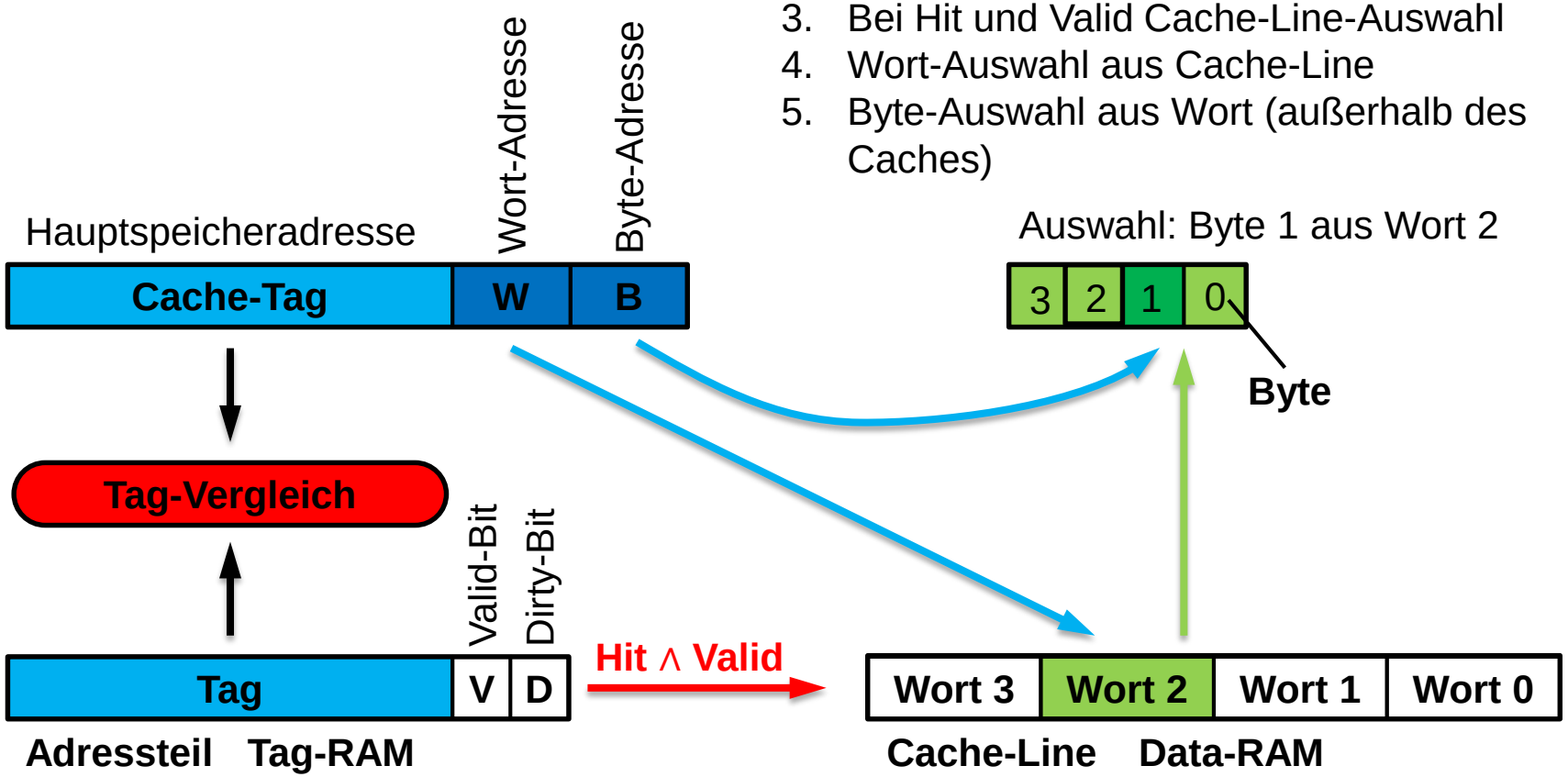
Organisation:
Lösung der Probleme der Speicherhierarchie: Abbildung, Identifikation, Ersetzung, Aktualisierung, Konsistenz, Kohärenz.

Struktur des Cache-Speichers (Übersicht)

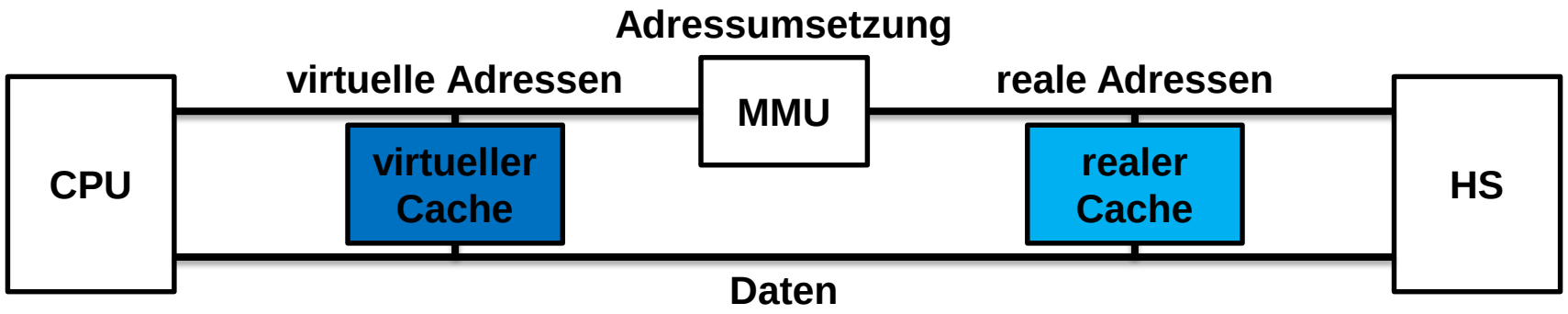


Cache-Struktur

1. Adresszerlegung: Tag, Wort, Byte
2. Tag-Vergleich und Valid-Auswertung
3. Bei Hit und Valid Cache-Line-Auswahl
4. Wort-Auswahl aus Cache-Line
5. Byte-Auswahl aus Wort (außerhalb des Caches)

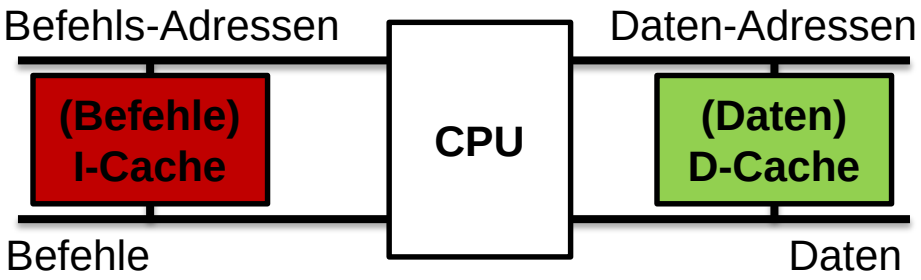


Cache-Adressraum



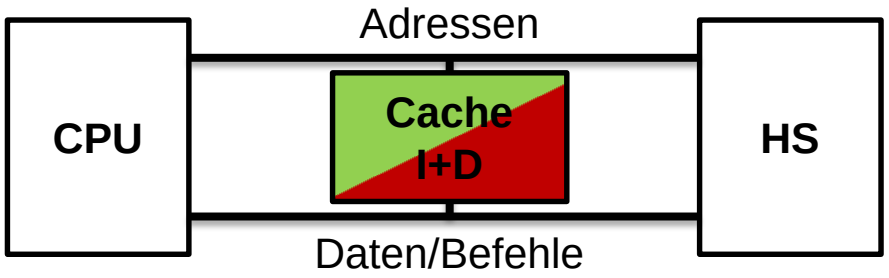
Virtueller Cache	Realer Cache
• schnell, Cache und MMU nebenläufig	• langsam, Adressen durch MMU
• Invalidierung bei Prozesswechsel, notwendig, da gleiche Adressen möglich	• direkter I/O-Cache-Transfer möglich, da reale Adressen
• besondere Eignung als Befehls-Cache, da Rückschreiben entfällt	• für Snooping geeignet, da 1:1 Adress-Daten-Abbildung

Cache-Architektur



Harvard-Architektur

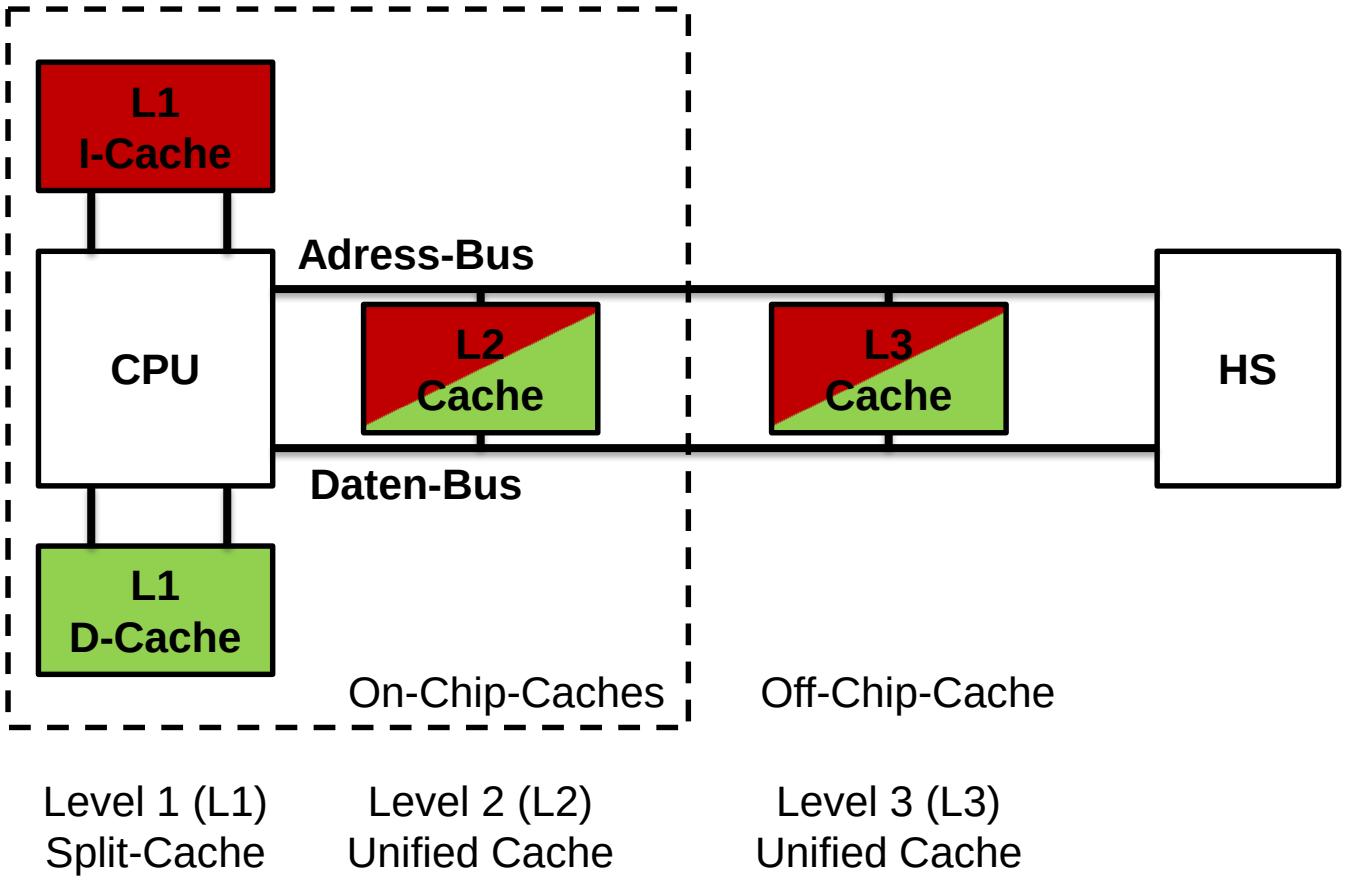
Split-Cache, getrennt für Daten und Befehle (separate Lokalitäten)



Princeton-Architektur

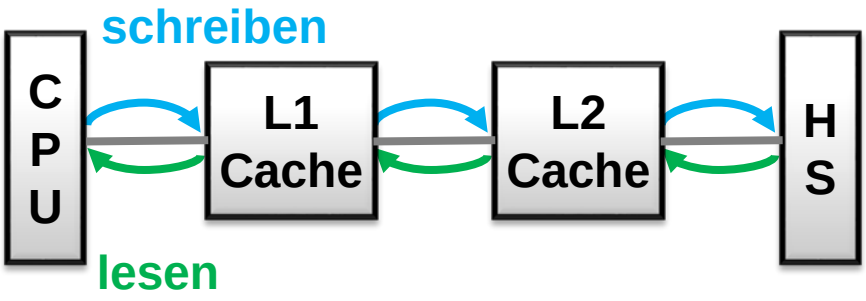
Unified Cache, gemeinsam für Daten und Befehle (überlagerte Lokalitäten)

Cache-Architektur mit drei Ebenen



Cache-Anordnung, linear - parallel

Lineare Anordnung

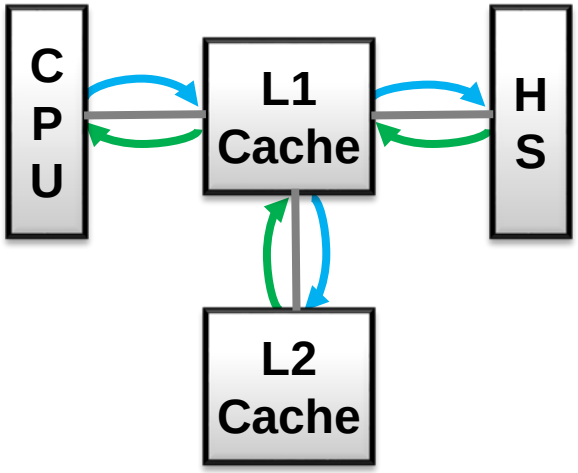


L2-Cache wird durch HS geladen.
L1-Cache wird durch L2 geladen

Victim-Cache

Aus L1 verdrängte Cache-Zeilen werden in den L2 geschrieben.
Werden Daten im L1 nicht gefunden werden sie aus dem L2 geladen. Werden sie dort nicht gefunden, werden sie aus dem HS direkt in den L1 geladen.
Aus L2 verdrängte Cache-Zeilen werden über den L1 in den HS propagiert.

Parallele Anordnung Victim-Cache

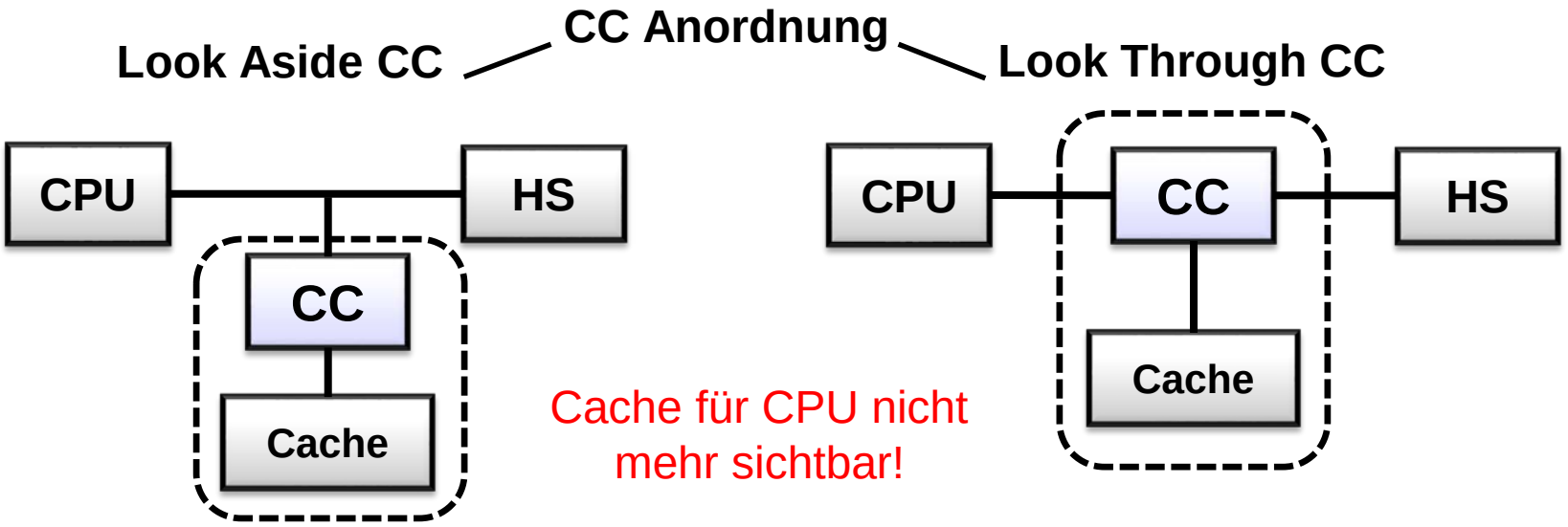


L1-Cache wird durch HS geladen.
L2-Cache wird durch L1 geladen

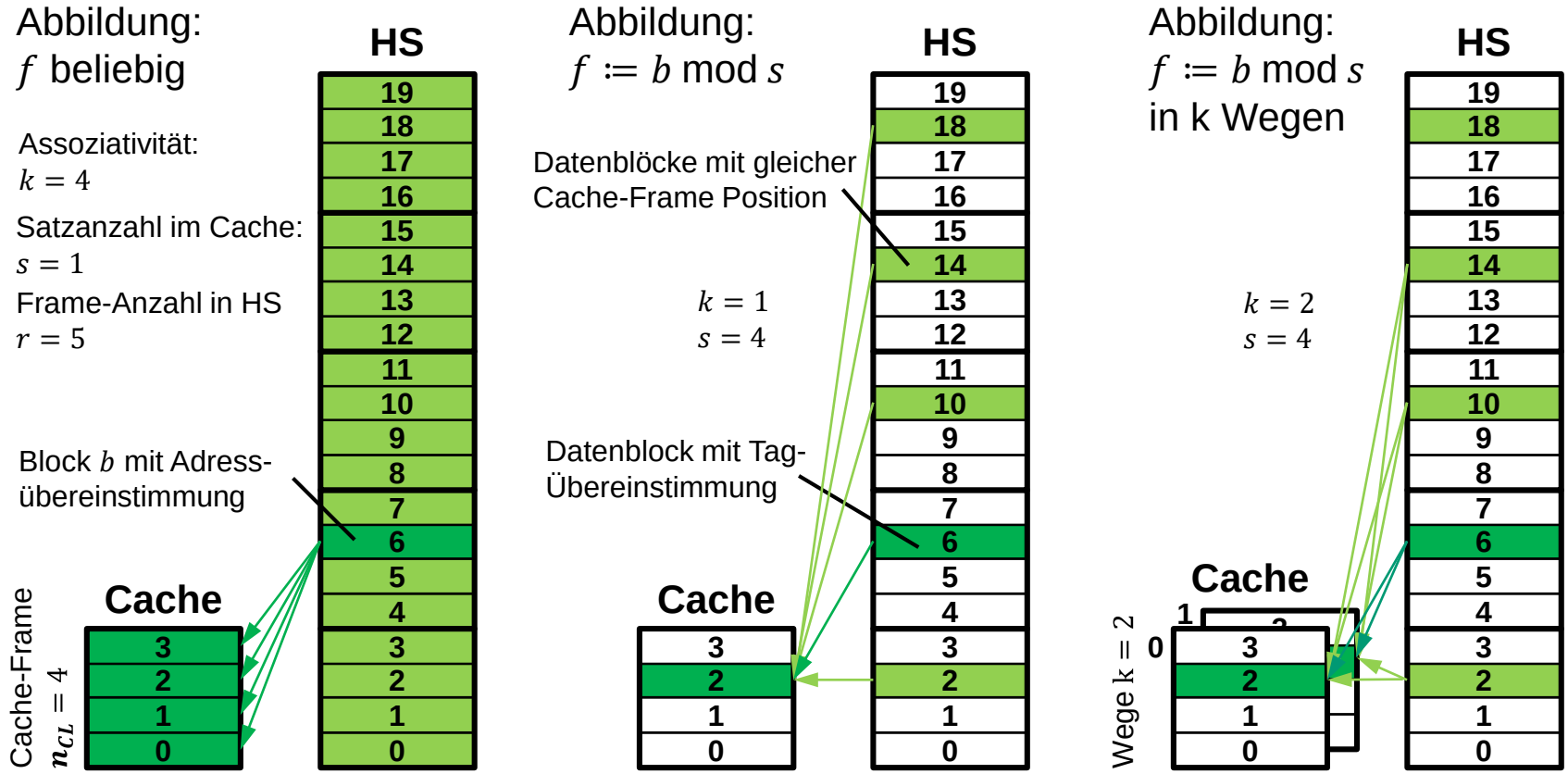
Cache-Controller

Aufgaben des Cache-Controllers (CC):

- Entlastung der CPU bei der Organisation des Speicherzugriffs.
- Überwachen des gemeinsamen Mediums (z. B. Bus oder Switch).
- Sicherstellen der Datenkonsistenz bzw. –Kohärenz.
- Beantworten bzw. bearbeiten von Prozessor-Anfragen.



Cache-Organisation – Abbildungsproblem



voll assoziativ
n_{cl}-fach assoziativ

direkt abgebildet
1-fach assoziativ

k-Wege Satz-assoziativ
k-fach assoziativ

Cache - Assoziativität

Vollassoziativer Cache (fully-associative)

Ein Speicherblock des HS kann sich in einer beliebigen Cache-Zeile des Caches befinden. Der Cache besteht aus einem Satz ($s=1$) mit $k=n_{cl}$ Zeilen

→ k-fach assoziativ (k-Wege).

Direkt abgebildeter Cache (direct-mapped)

Ein Speicherblock des HS kann sich nur in einem bestimmten Satz, Zeile des Caches befinden. Die Zeile, der Satz wird durch den Index in der HS-Adresse festgelegt. Der Cache besteht aus $s=n_{cl}$ Sätzen mit je einer Cache-Zeile ($k=1$)

→ 1-fach assoziativ (1-Weg).

K-Wege satzassoziativer Cache (k-way set-associativ)

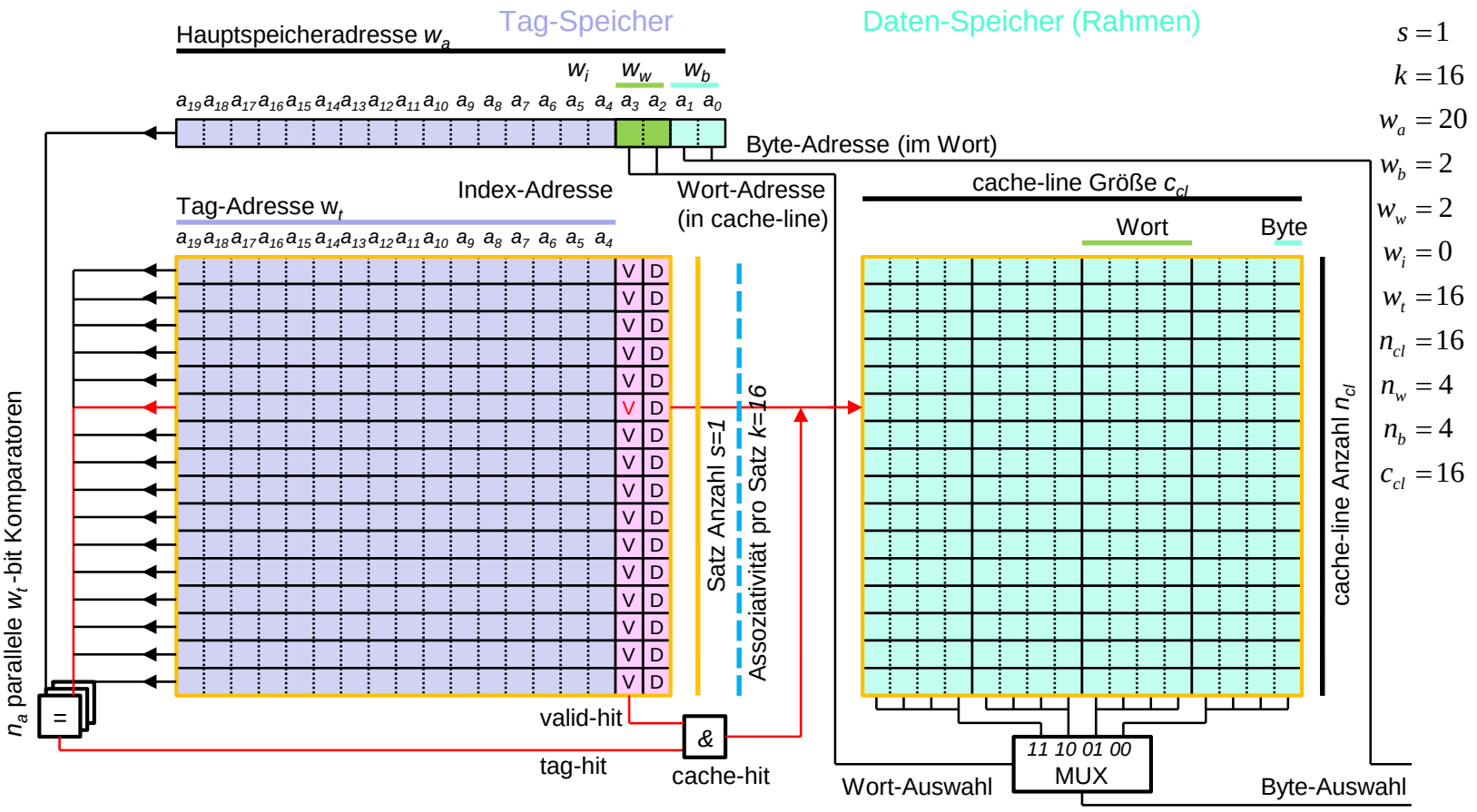
Ein Speicherblock des HS kann sich nur in einem bestimmten Satz des Caches befinden. Der Satz wird durch den Index in der HS-Adresse festgelegt. Der Cache besteht aus $s=n_{cl}/k$ Sätzen mit je k Cache-Zeilen

→ k-fach satzassoziativ (k-Wege) (assoziativ pro Satz)

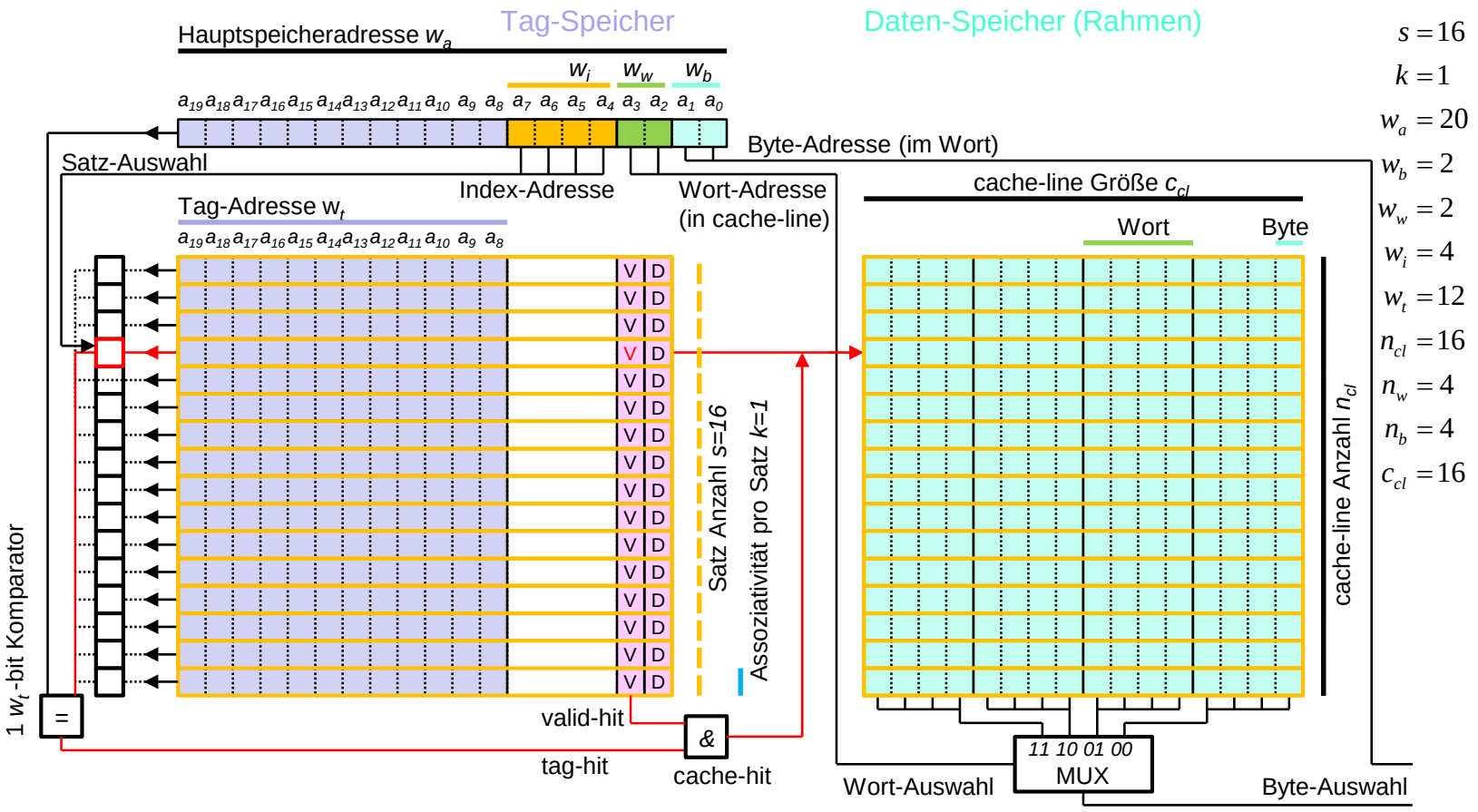
Cache - Assoziativität – Vorteile / Nachteile

1. **Vollassoziativer Cache (fully-associative)**
   - Sehr gute Trefferrate und universell einsetzbar.
   - Aufwändige HW-Realisierung, n_{cl} voll-parallele Komparatoren für die gesamte Tag-Breite, relativ langsam.
2. **Direkt abgebildeter Cache (direct-mapped)**
   - Einfache HW-Realisierung, direkte Zeilen-Adressierung mit Index, nur ein Vergleicher mit einem um den Index reduzierten Tag, sehr schnell.
   - Schlechte Trefferrate, Ping-Pong Effekt (ständiges Ein- und Auslagern der Daten, cache trashing).
3. **K-Wege satzassoziativer Cache (k-way set-associativ)**
   - Kompromiss aus vollassoziativem und direkt abgebildetem Cache. 8-fach satzassoziativer Cache ist nahezu so gut wie ein vollassoziativer.
   - Gute Trefferrate bei hoher Geschwindigkeit – Kompromiss aus 1. und 2.

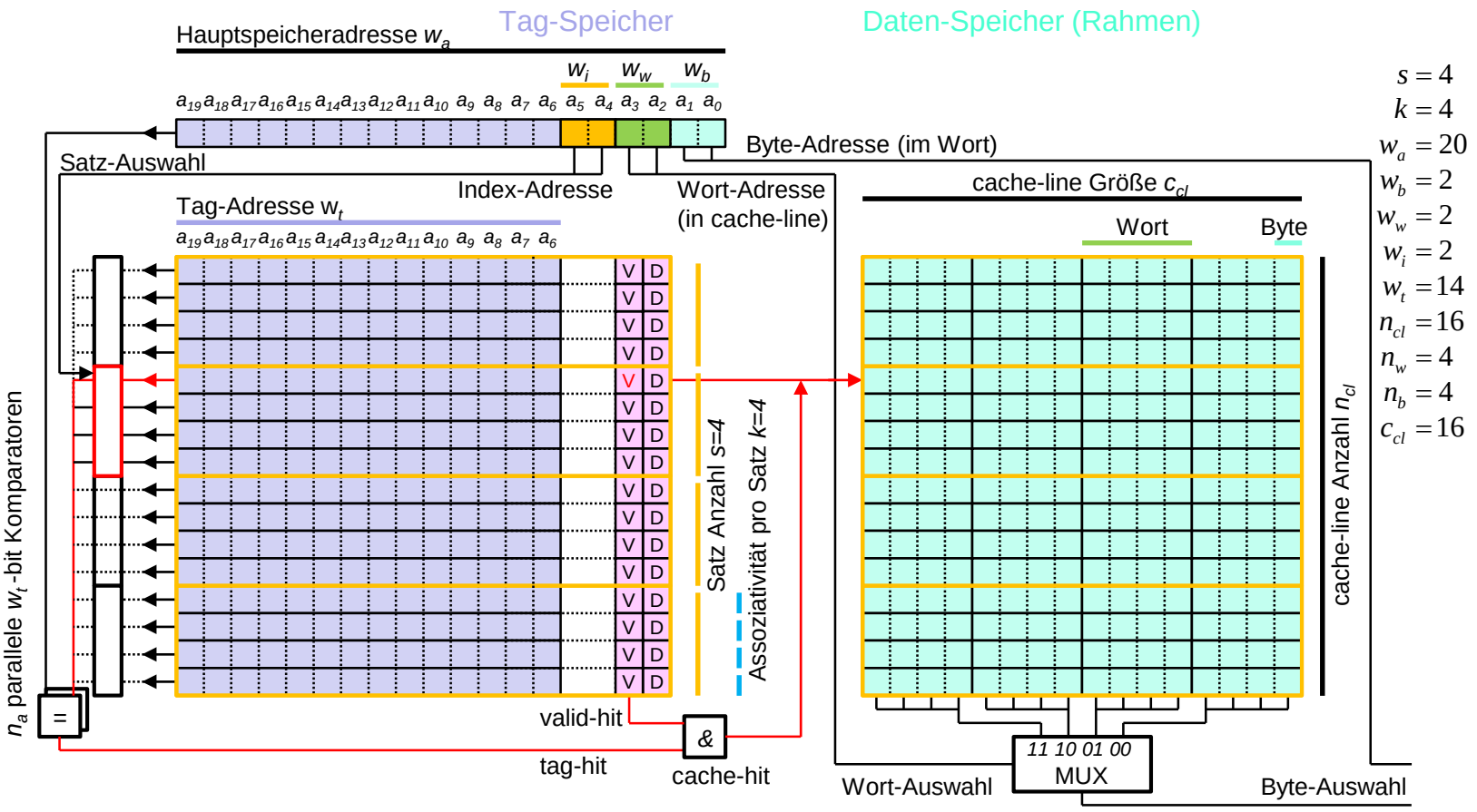
Vollassoziativer Cache (*k*-fach assoziativ)



Direkt abgebildeter Cache (1-fach assoziativ)

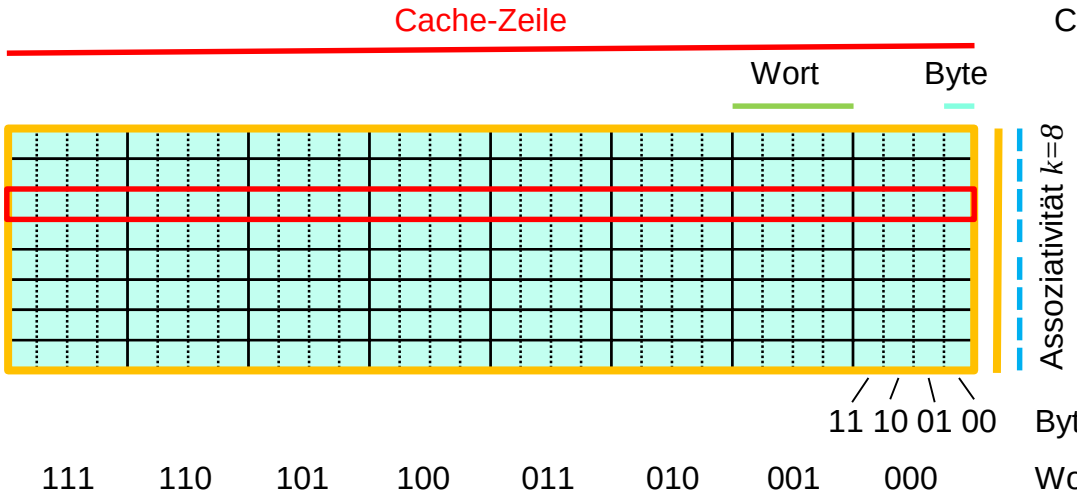


K-Wege satzassoziativer Cache (*k*-fach assoziativ)



Aufbau eines Cache-Satzes, Cache-Zeile

Satz eines Cache-Daten-Speichers
(Cache-Rahmen, durch Index-Adresse festgelegt)



Cache-Zeilenkapazität c_{cl} Byte

$$c_{cl} = 2^{w_w} \cdot 2^{w_b} = n_w \cdot n_b$$

Satzkapazität c_s Byte

$$c_s = k \cdot c_{cl}$$

$k = 8$
$c_{cl} = 32$
$w_b = 2$
$w_w = 3$
$c_s = 256$

$n_w = 8$
$n_b = 4$

Byteadresse w_b bit

Wortadresse w_w bit

vollassoziativ: $s = 1, \quad k = n_{cl}, \quad c_s = n_{cl} \cdot c_{cl}$

direkt abgebildet: $s = n_{cl}, \quad k = 1, \quad c_s = c_{cl}$

k-Wege satzassoziativ: $s = {}^{n_{cl}}\!/_{k}, \quad c_s = k \cdot c_{cl}$

Daten-Speicher: $c_d = n_{cl} \cdot c_{cl}$

Cache - Identifikationsproblem

Aufteilung der Hauptspeicheradresse zur Lokalisation der Daten im Cache

$$w_a = w_t + w_i + w_w + w_b$$

$$w_o = w_w + w_b$$

Tag:	Adressteil zum assoziativen Vergleich mit dem Cache-Tag-RAM
Index:	Satzauswahl (Set) im Cache → vollassoziativer Cache hat keine Satzauswahl
Offset:	Adresse in der Cache-Zeile
Wort:	Wort-Adresse in der Cache-Zeile
Byte:	Byte-Adresse im Wort

Cache - Berechnungen

w_a	Bitanzahl der Hauptspeicheradresse	$w_a = w_t + w_i + w_w + w_b$
w_t	Bitanzahl der Tag-Adresse	
w_i	Bitanzahl der Index-Adresse	
w_w	Bitanzahl der Wort-Adresse	
w_b	Bitanzahl der Byte-Adresse	
w_o	Bitanzahl des Zeilen-Offset	$w_o = w_w + w_b$
n_w	Anzahl der Wörter je Cache-Zeile	$n_w = 2^{w_w}$
n_b	Anzahl der Byte je Wort	$n_b = 2^{w_b}$

Cache - Berechnungen

n_{cl}	Anzahl der Cache-Zeilen	$n_{cl} = c_d / c_{cl}$
s	Anzahl der Sätze (Mengen)	$s = 2^{w_i}$
k	Assoziativität im Satz	
c_{cl}	Kapazität der Cache-Zeile (Byte)	$c_{cl} = n_w \cdot n_b$
c_s	Kapazität eines Satzes (Byte)	$c_s = k \cdot c_{cl}$
c_d	Kapazität des Daten-Speichers (Byte)	$c_d = n_{cl} \cdot c_{cl}$
c_t	Kapazität des Tag-Speichers, ohne V,D	$c_t = n_{cl} \cdot w_t$

Cache - Berechnungen

Hauptspeicheradresse (Aufteilung)

$w_a = w_t + w_i + w_w + w_b$ (Summe der Bitanzahlen)

Cache-Zeilengröße (cache line size)

$c_{cl} = 2^{w_w} \cdot 2^{w_b}$ Byte

$= n_w \cdot n_b$ Byte (Wort-Anzahl je Cache-Zeile * Byte-Anzahl je Wort)

Cache-Zeilenanzahl

$n_{cl} = 2^{w_i} \cdot k = s \cdot k$ (Satz-Anzahl je Cache * Assoziativität je Satz)

Daten-Speichergröße

$c_d = n_{cl} \cdot c_{cl}$ (Cache-Zeilenanzahl * Cache-Zeilengröße)

Tag-Speichergröße (ohne V,D)

$c_t = w_t \cdot n_{cl}$ bit (Bitanzahl der Tagadresse* Cache-Zeilenanzahl)

Beispiel Cache - Analyse

Beispielaufgabe: Ein 4-fach satzassoziativer Cache hat eine Daten-Speichergröße von 4 MiByte. Die Cache-Zeile enthält 16 Worte zu je 4 Byte. Bestimmen Sie die Aufteilung der Hauptspeicheradresse (32 bit) sowie die Größe des Tag-Speichers, ohne V- und D-Bit!

1. Bitanzahl der Wort-, Byteadresse

$$w_w = ld16 = 4, \quad w_b = ld\,4 \;\; = 2$$

2. Cache-Zeilengröße

$$c_{cl} = n_w \cdot n_b = 64 \text{ Byte}$$

3. Daten-Speichergröße

$$c_d = n_{cl} \cdot c_{cl}, \quad n_{cl} = \frac{c_d}{c_{cl}} = \frac{4 \cdot 2^{20}}{64} = 2^{16}$$

4. Cache-Zeilenanzahl

$$n_{cl} = 2^{w_i} \cdot k, \quad w_i = ld\,\frac{n_{cl}}{k} = ld\,\frac{2^{16}}{4} = 14$$

5. Bitanzahl der Tag-Adresse

$$w_a = w_t + w_i + w_w + w_b$$

$$w_t = w_a - \left(w_i + w_w + w_b\right) = 12$$

6. Tag-Speichergröße

$$c_t = w_t \cdot n_{cl} = 3 \cdot 2^{18} \text{ bit}$$

$$w_a = 32, \;\; w_t = 12, \;\; w_i = 14$$
$$w_w = 4, \;\; w_b = 2, \;\; c_t = 3 \cdot 2^{18} \text{ bit}$$

Cache - Ersetzungsproblem

Welche Cache-Zeile wird beim Nachladen eines mehrfach assoziativen Caches (innerhalb eines Satzes) verdrängt, ersetzt?

Beim direkt abgebildeten Cache (direct mapped, einfach assoziativ) kein Ersetzungsproblem, da nur eine mögliche Position.

Bein vollassoziativen Cache (nur ein Satz) steht jede Cache-Zeile zur Disposition.

Ist innerhalb des Satzes eine Cache-Zeile frei ($P=0$), so wird diese verwendet, eine Verdrängung ist nicht erforderlich.

Die Verdrängung einer Cache-Zeile aus einem Satz kann nach verschiedenen Strategien erfolgen.

Beachte!

Ist bei der zu verdrängenden Cache-Zeile das Dirty-Bit gesetzt ($D=1$), so ist diese Zeile zuerst in den Hauptspeicher zu propagieren
→ Aktualisierungsproblem.

Cache – Ersetzungsproblem - Strategien

PERFECT (Optimal)

Ersetze in Zukunft nicht mehr benötigte Cache-Zeile.
Auswahl ist nicht möglich, da der Programmverlauf nicht im Voraus bekannt ist. Nicht realisierbar!

LRU (Least Recently Used)

Ersetze am längsten (zeitlich gesehen) nicht mehr benutze Cache-Zeile.
Auswahl erfolgt mit Hilfe einer Warteschlange.
→ mittelmäßige Trefferrate (Anzahl der Benutzungen nicht berücksichtigt)

LFU (Least Frequently Used)

Ersetze am wenigsten (Häufigkeit) benutze Cache-Zeile.
Auswahl erfolgt mit Hilfe einer Benutzungsstatistik (Utilization Counter).
→ gute Trefferrate (oft benutze Zeile kann Cache blockieren!)

Cache – Ersetzungsproblem - Strategien

FIFO (First In First Out)

Ersetze älteste Cache-Zeile (zeitlich gesehen).
Auswahl erfolgt mit Hilfe einer Ladestatistik.
→ mittelmäßige Trefferrate (Anzahl der Benutzungen nicht berücksichtigt)

RANDOM (Zufallsprinzip)

Ersetze eine Cache-Zeile nach dem Zufallsprinzip.
Auswahl mit Hilfe von Pseudo-Zufallszahlen.
→ gute Trefferrate

ROUND ROBBIN

Ersetze Cache-Zeile in vorher festgelegter Reihenfolge.
Auswahl mit Hilfe von Markierungen bereits verwendeter Cache-Zeilen.
→ gute Trefferrate

Cache-Organisation – Aktualisierungsproblem

Datenkonsistenz und Datenkohärenz

Wann werden welche Daten im Cache oder im Hauptspeicher aktualisiert?

Konsistenz:

Sowohl der Cache als auch der Hauptspeicher enthalten stets die identischen aktuellen Daten. Alle untergeordneten Hierarchieebenen enthalten stets auch die Daten aller übergeordneten Ebenen in identischer Form.

Kohärenz:

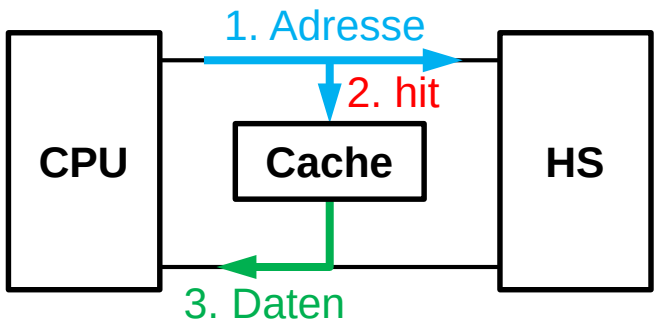
Der CPU werden immer die aktuellen Daten bereitgestellt, unabhängig davon, ob sie im Cache oder im Hauptspeicher stehen. Auf veraltete Daten (Stale Data) darf nicht zugegriffen werden (→ Konsistenzabschwächung).

Problemfälle:

- Auslagern, Rückschreiben in untergeordnete Hierarchieebenen (Castoff).
- Mehrere Master im System (CPU und DMAC, Direct Memory Access).

Cache Lese-Strategie (Read Strategy) Read-Hit

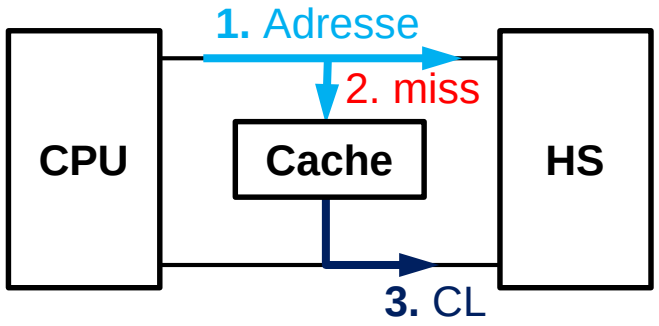
Treffer: (Read-Hit)

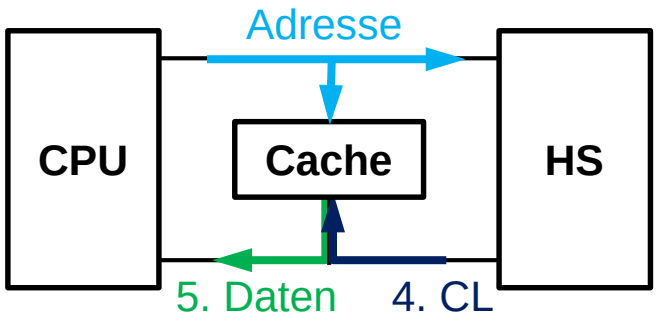


1. Cache und HS adressieren
2. Cache-Hit, HS-Zugriff abbrechen
3. Lesen der Daten aus dem Cache

Cache Lese-Strategie (Read Strategy) Read-Miss

Fehlzugriff: (Read-Miss)





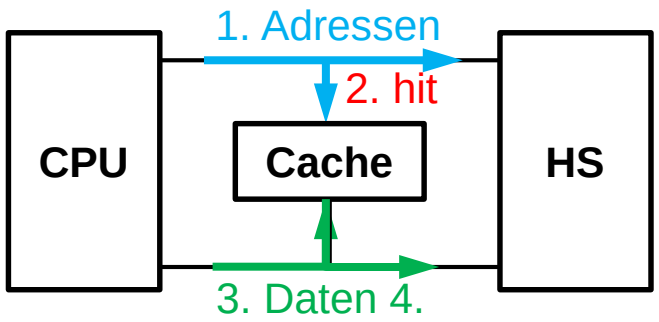
1. Cache und HS adressieren
2. Cache-Miss, HS-Zugriff fortsetzen
3. Verdrängen einer Cache-Line, nicht erforderlich bei freier CL (Ersetzungsproblem, Castoff)

4. **Cache-Line aus HS in Cache laden**
5. **Lesen der Daten aus dem Cache**

Ein Schreibpuffer (Castoff-Buffer) zu Verkürzung der Schreibzeit in den Hauptspeicher ist sinnvoll.

Cache Schreib-Strategie (Write Strategy) Write-Hit

Treffer: Write-Through (WT)

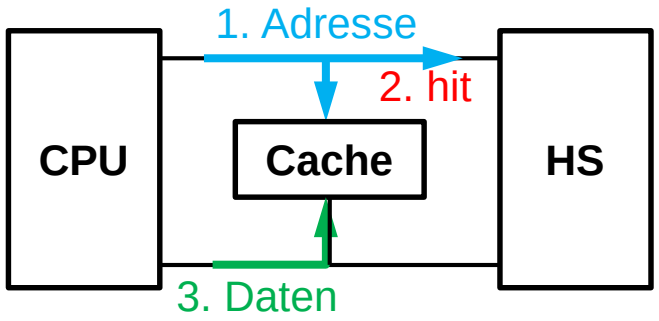


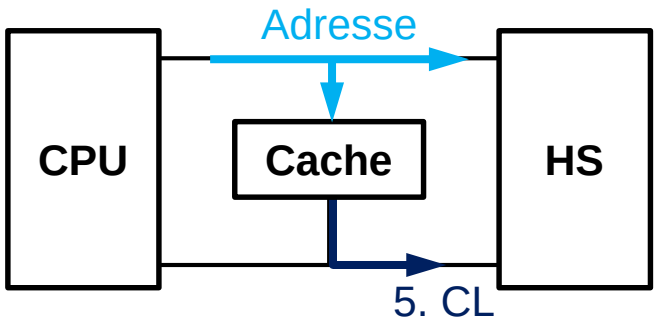
1. Cache und HS adressieren
2. Cache-Hit
3. Schreiben der Daten in den Cache
4. HS unverzüglich aktualisieren

→ Zeitaufwendig, aber konsistent !
Ein Schreibpuffer (Write-Buffer) zur Verkürzung der Schreibzeit in den Hauptspeicher ist sinnvoll.

Cache Schreib-Strategie (Write Strategy) Write-Hit

Treffer: Write-Back, Copy-Back (WB)





1. Cache und HS adressieren
2. Cache-Hit
3. Schreiben der Daten in den Cache und Setzen des Dirty-Bit im Cache

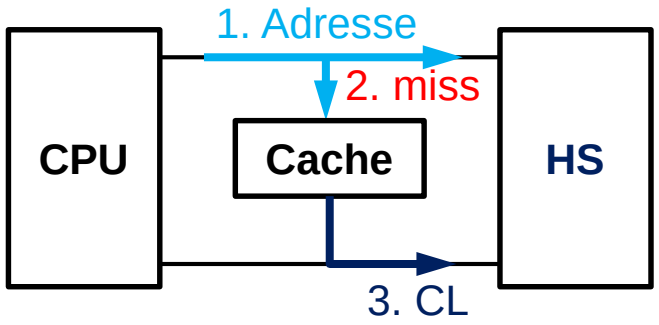
→ zeitversetzt →

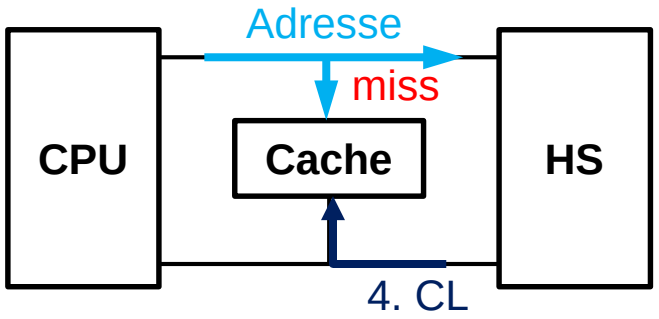
4. HS bei Verdrängung der Cache-Line oder Cache Flush aktualisieren
   → Dirty-Bit Rücksetzen

Daten-Kohärenz, keine Konsistenz
→ Problem mit Daten-Konsistenz

Cache Schreib-Strategie (Write Strategy) Write-Miss

Fehlzugriff: Write-Allocate



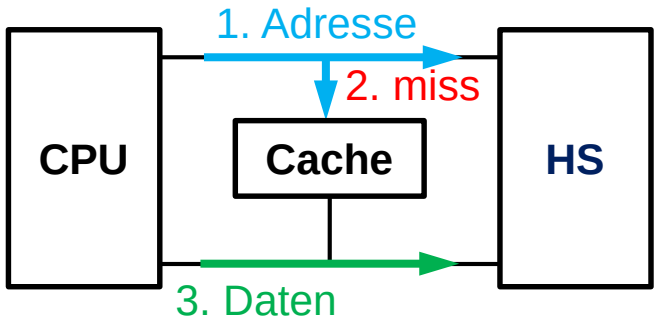


1. Cache und HS adressieren
2. Cache-Miss, HS-Zugriff fortsetzen
3. Verdrängen einer Cache-Line, nicht erforderlich bei freier CL (Ersetzungsproblem, Castoff)
4. Cache-Line aus HS in Cache laden
5. → Weiter analog Cache Schreib-Strategie Write-Hit

Gesuchte Cach-Line wird erst in den Cache geladen → Cache hit →
Write-Allocate mit Write-Back
Write-Allocate mit Write-Through

Cache Schreib-Strategie (Write Strategy) Write-Miss

Fehlzugriff: Write-Around (No-Write-Allocate)



1. Cache und HS adressieren
2. Cache-Miss, HS-Zugriff fortsetzen
3. Schreiben der Daten direkt in den HS (Cache wird nicht geladen)

Der Cache wird ignoriert !

→ **Daten stehen nach Schreibvorgang nicht im Cache**

Ursachen für Cache-Fehlzugriffe (cache-misses, 3C/4C)

Capacity-Miss:

Cache ist nicht groß genug. Blöcke werden ständig ein- und ausgelagert (thrasching) → Vergrößerung des Caches.

Conflict-Miss (Collision-Miss):

Die Assoziativität des Caches reicht nicht aus, zu große Blöcke,

(Ping-Pong-Effekt) → Erhöhung der Assoziativität.

Compulsory-Miss:

Gesuchte Daten sind nicht im Cache, Cache am Anfang leer, Erstzugriff → Verringerung der Cache-Größe.

Coherence-Miss:

Blöcke sind ungültig aufgrund von Zugriffen anderer Prozessoren.

Cache-Dimensionierung → Optimierungsproblem

6 Hauptspeicher

- RAM (Random Acces Memory), wahlfreier Lese- und Schreibzugriff, beliebig oft schreibbar, lesbar.
- Technologisch allgemein als dynamischer RAM, DRAM realisiert.
- DRAM-Speicher besitzen gegenüber SRAM-Speicher eine höhere Speicherkapazität und sind pro Speicherbit preislich deutlich günstiger.
- Als Speicherelement wird eine Ladung in einem Kondensator gespeichert.
- Die Speicherung, Ladung ist flüchtig → Auffrischung des Speicherinhaltes, der Ladung (refresh) erforderlich → zusätzliche Zeiten → Zykluszeiten.
- Der Hauptspeicher wird über den Speicherbus mit der CPU verbunden.
- Aktuelle DRAM-Module sind z.B.:
  DIMMs mit z. B. SD-, DDR-SD-, DDR2-, DDR3- oder DDR4-SDRAMs
  SDRAM: Synchronous Dynamic Random Access Memory
  DDR-SDRAM (Double Data Rate Synchronous Dynamic Random Access Memory

Hauptspeicher-Kenngrößen

Kenngrößen

- Speichertyp, Modulausführung, nutzbare Speicherkapazität, ECC
- Zugriffsbreite, Speicher-Timingwerte, Burst Unterstützung
- Lese-, Schreibgeschwindigkeiten, Datenübertragungsrate
- Speicherbandbreite, Speichertakt, Zugriffslatenzen, Zykluszeiten, Transferrate
- Parallelität, n-fach prefetch, Speichercontroller, DRAM-Controller, Businterface

Erhöhung der Transferraten, Speicherbandbreite durch Paralelität

- Speichercontroller mit Dual Channel, Triple Channel oder Quad Channel
- Mehrere Speicherkanäle, mehrere Speichermodule parallel
- Vergrößerung der Zugriffsbreite, Erhöhung des Speichertaktes

7 Zusammenfassung

- Die Speicherhierarchie dient der Überbrückung der Differenzen bzgl. Speicherkapazität und Zugriffszeit zwischen CPU und Hauptspeicher.
- Die Nutzung des Lokalitätsprinzipes örtlich und zeitlich ermöglicht eine hohe Effizienz in der Speicherhierarchie.
- Nutzung spezieller Registerarchitekturen.
- Nutzung der Harvard-Architektur auf Cache-Ebene.
- Lösung der Probleme in der Cache-Architektur: Abbildungsproblem, Identifikationsproblem, Ersetzungsproblem, Aktualisierungsproblem, Konsistenzproblem (Kohärenzproblem).
- Die Nutzung von Parallelität ermöglicht in allen Ebenen der Speicherhierarchie weitere Reduzierungen der Zugriffszeit und Erhöhungen des Datendurchsatzes.
- Weiterentwicklung der Speicherarchitektur → virtueller Speicher.