

# Kapitel 7

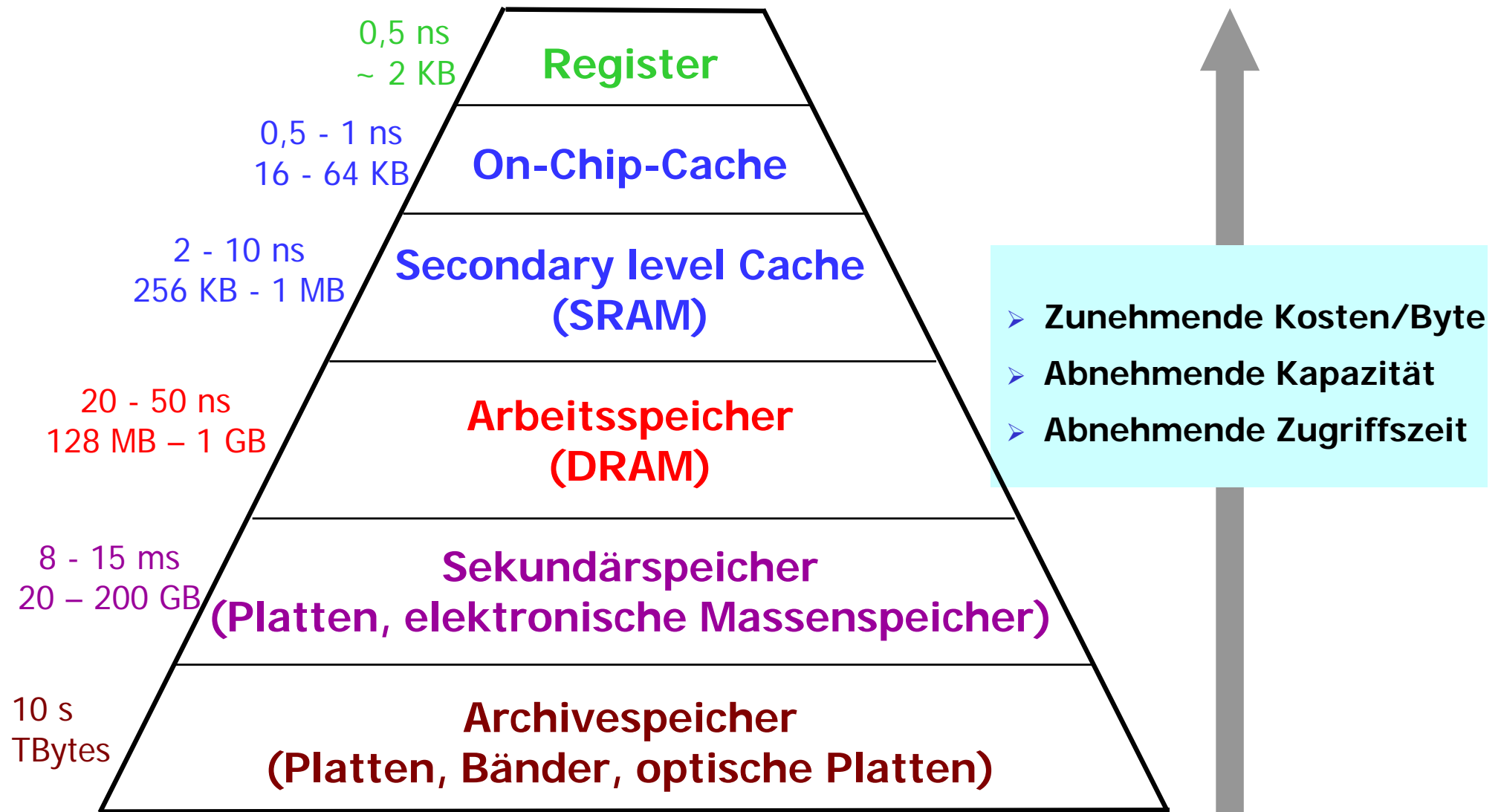
---

## Cache-Speicher

- ❑ Speicherhierarchie
- ❑ Funktionsweise
- ❑ Aufbau
- ❑ Organisationsformen



# Wdh: Speicherhierarchie



# Cache- Speicher

---

- Pufferspeicher mit schnellem Zugriff
- Geringe Kapazität im Vergleich zum Hauptspeicher
- Cache-Speicher stellen die während einer Programmausführung aktuellen Hauptspeichereinhalte für Prozessorzugriffe als **Kopien** möglichst schnell zur Verfügung.
- **Cache-Speicher-Verwaltung** sorgt dafür, dass der Cache-Speicher das Datum enthält, auf dem der Prozessor als nächstes zugreift.
- **Cache-Controller (Hardware)** kopiert automatisch die Daten in den Cache, auf die der Prozessor zugreift.



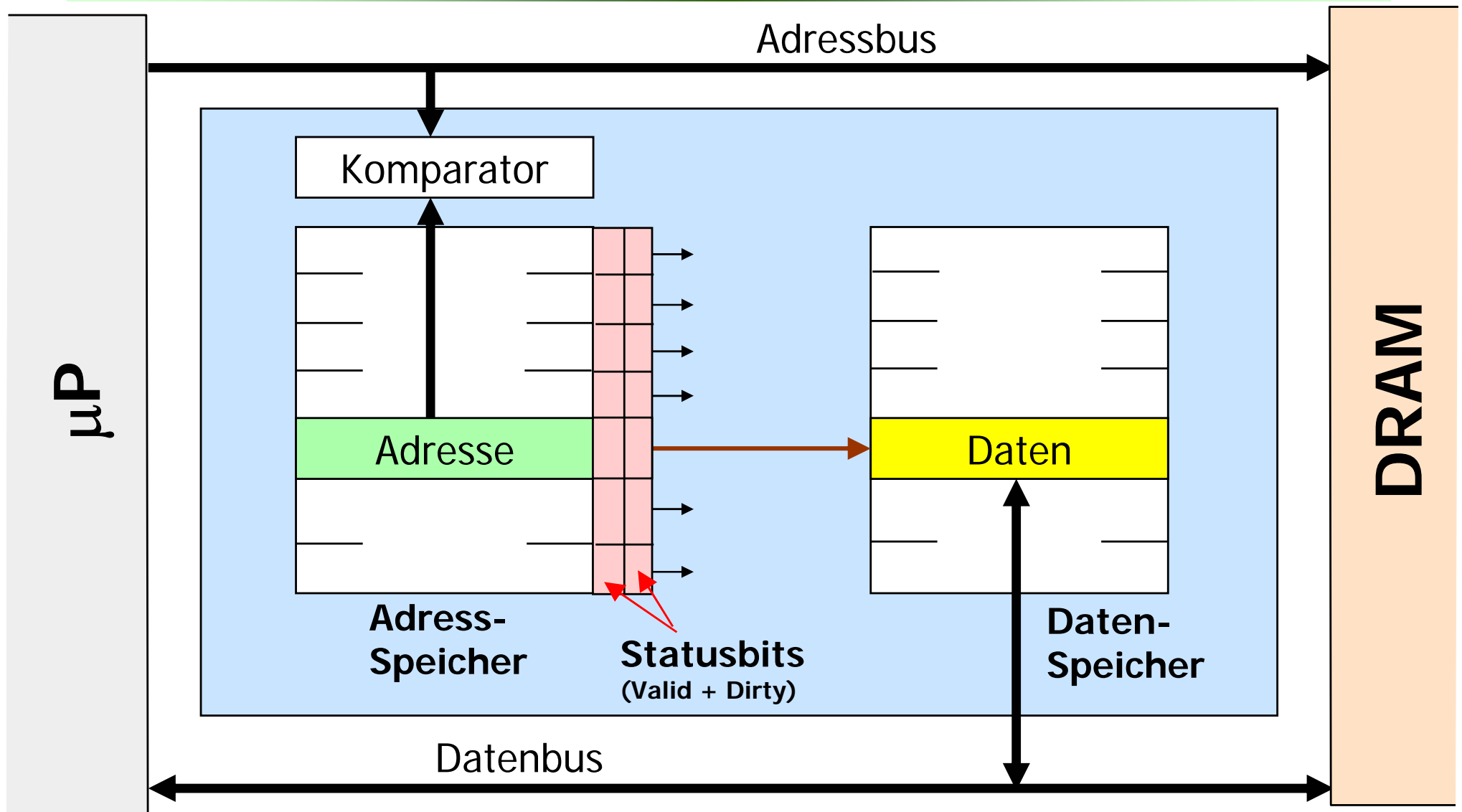
# Cache- Speicher

---

- Auf den Cachespeicher soll der Prozessor fast so schnell wie auf seine Register zugreifen können.  
Er ist deshalb bei Mikroprozessoren direkt auf dem Prozessorchip angelegt oder in der schnellen und teuren SRAM-Technologie realisiert.
- **Cache-Treffer (Hit):** das angeforderte Datum ist im Cachespeicher vorhanden.
- **Cache-Fehlzugriff (Miss):** das angeforderte Datum steht nur im Hauptspeicher.
- **Hit-Ratio:** Trefferquote



# Wdh. Aufbau eines Cache-Speichers



# Cache-Strukturen

---

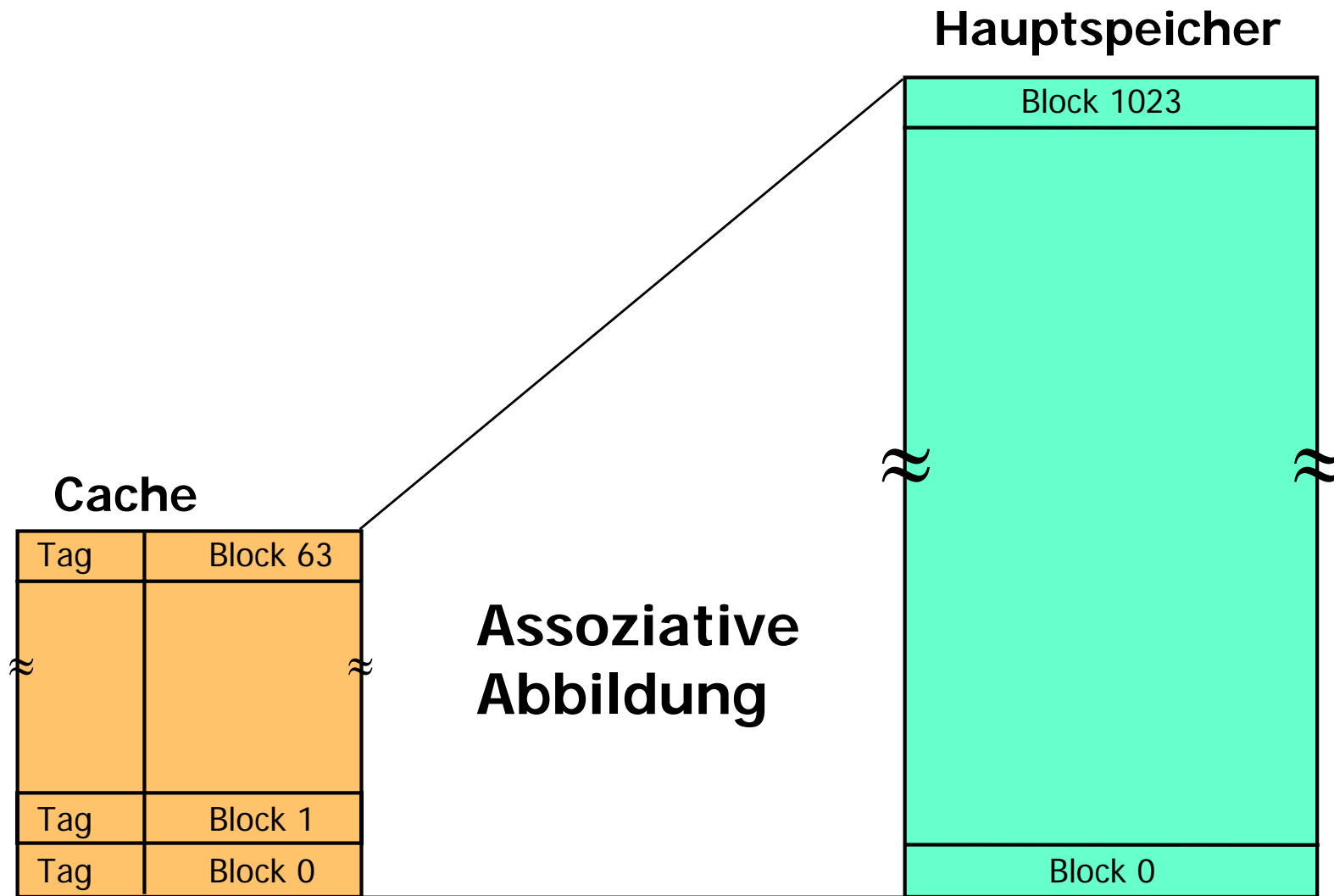
**Wie wird festgestellt, ob die benötigten Daten im Cache sind und falls ja wie können diese gefunden werden?**

**3 Techniken für den Adressvergleich → 3 Cache-Typen:**

- **Voll-Assoziativer Cache**
- **Direct Mapped Cache**
- **n-Way Set Associative Cache**



# Voll-assoziativer Cache



# 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:**

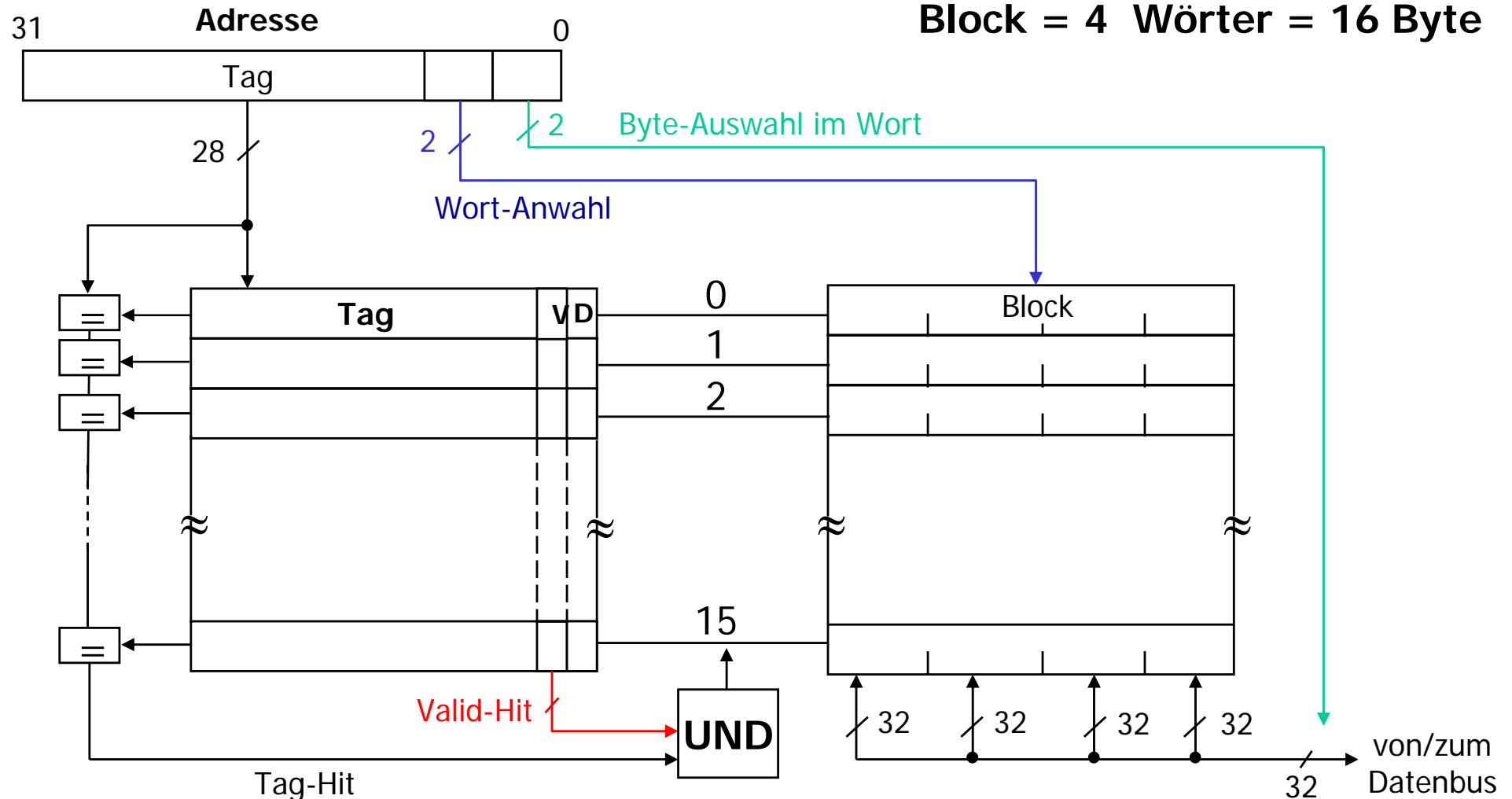
- Hoher Hardwareaufwand (für jede Cache-Zeile ein Vergleich)  
➔ nur für sehr kleine Cachespeicher realisierbar
- Die große Flexibilität der Abbildungsvorschrift erfordert eine weitere Hardware, welche die Ersetzungsstrategie (welcher Block soll überschrieben werden, wenn der Cache voll ist) realisiert.



# Beispiel: Vollassoziativer Cache

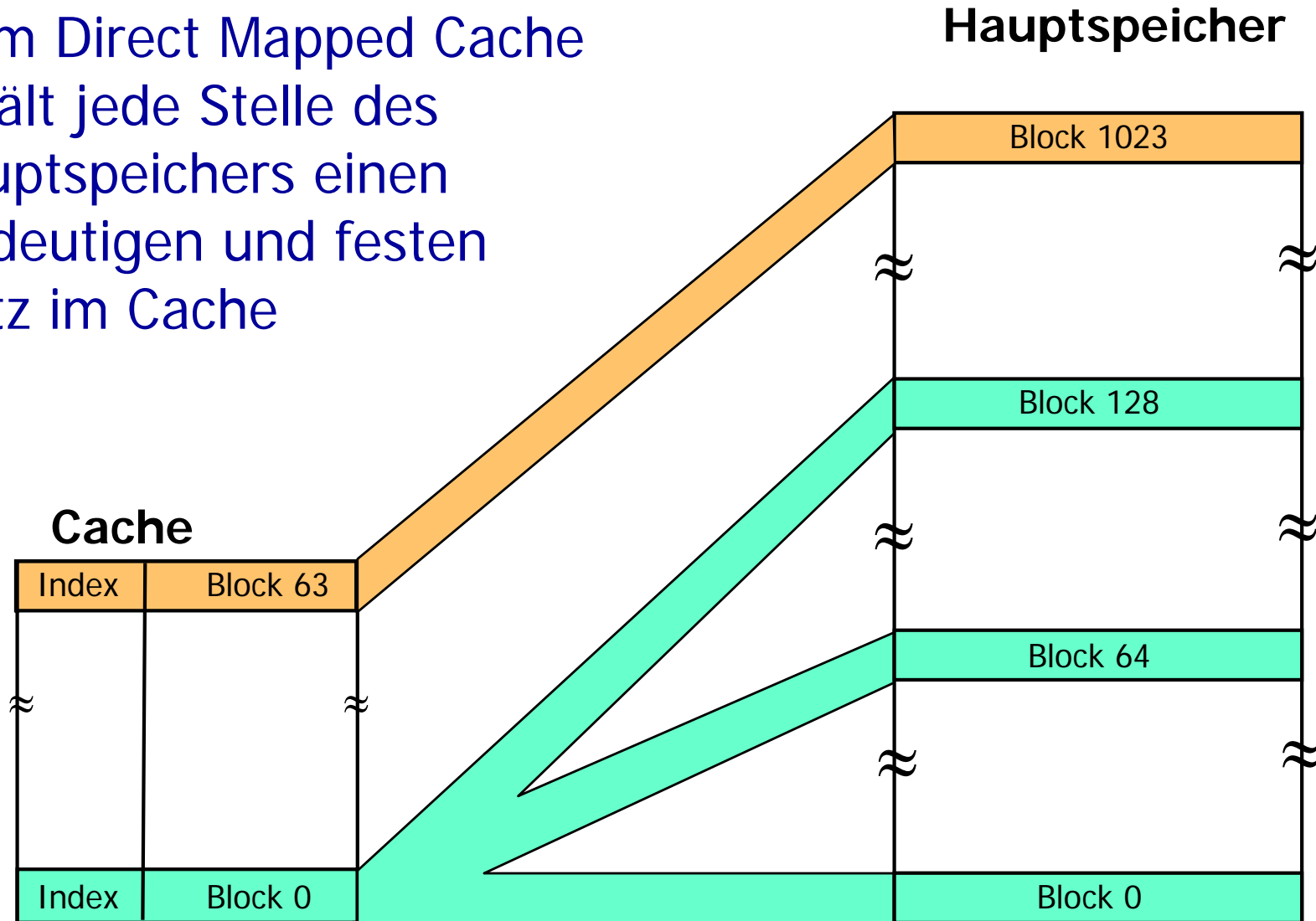
Kapazität: 256 Byte

Block = 4 Wörter = 16 Byte



# Direct-mapped-Cache

Beim Direct Mapped Cache erhält jede Stelle des Hauptspeichers einen eindeutigen und festen Platz im Cache



# Direct-mapped-Cache

---

## Nachteile:

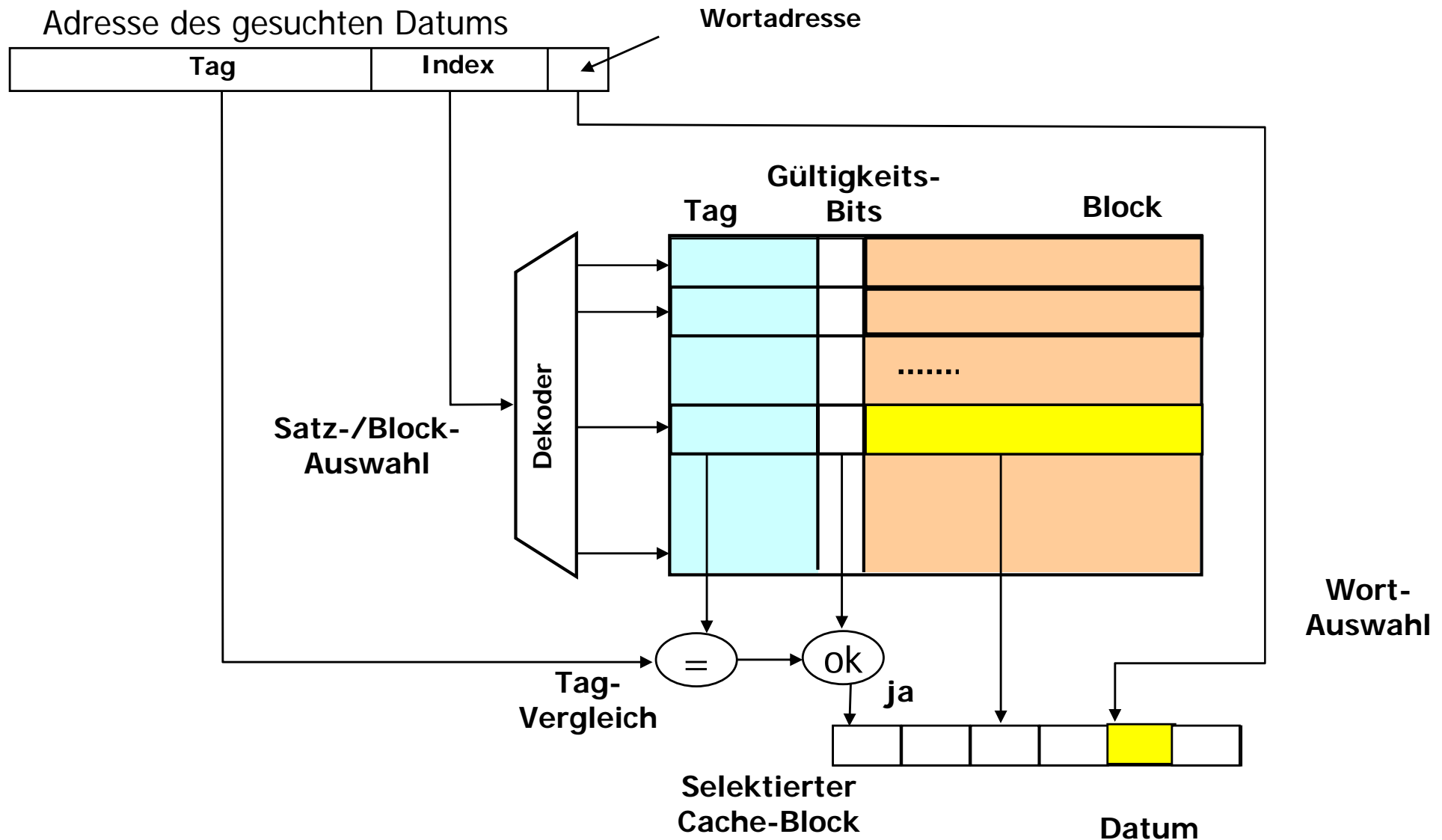
Ständige Konkurrenz der Blöcke (z. B. 0, 64, 128,...), obwohl andere Blöcke im Cache frei sein können.

## Vorteile:

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



# Adressierung im Direct-Mapped Cache



# Merkmale des Direct Mapped Cache

---

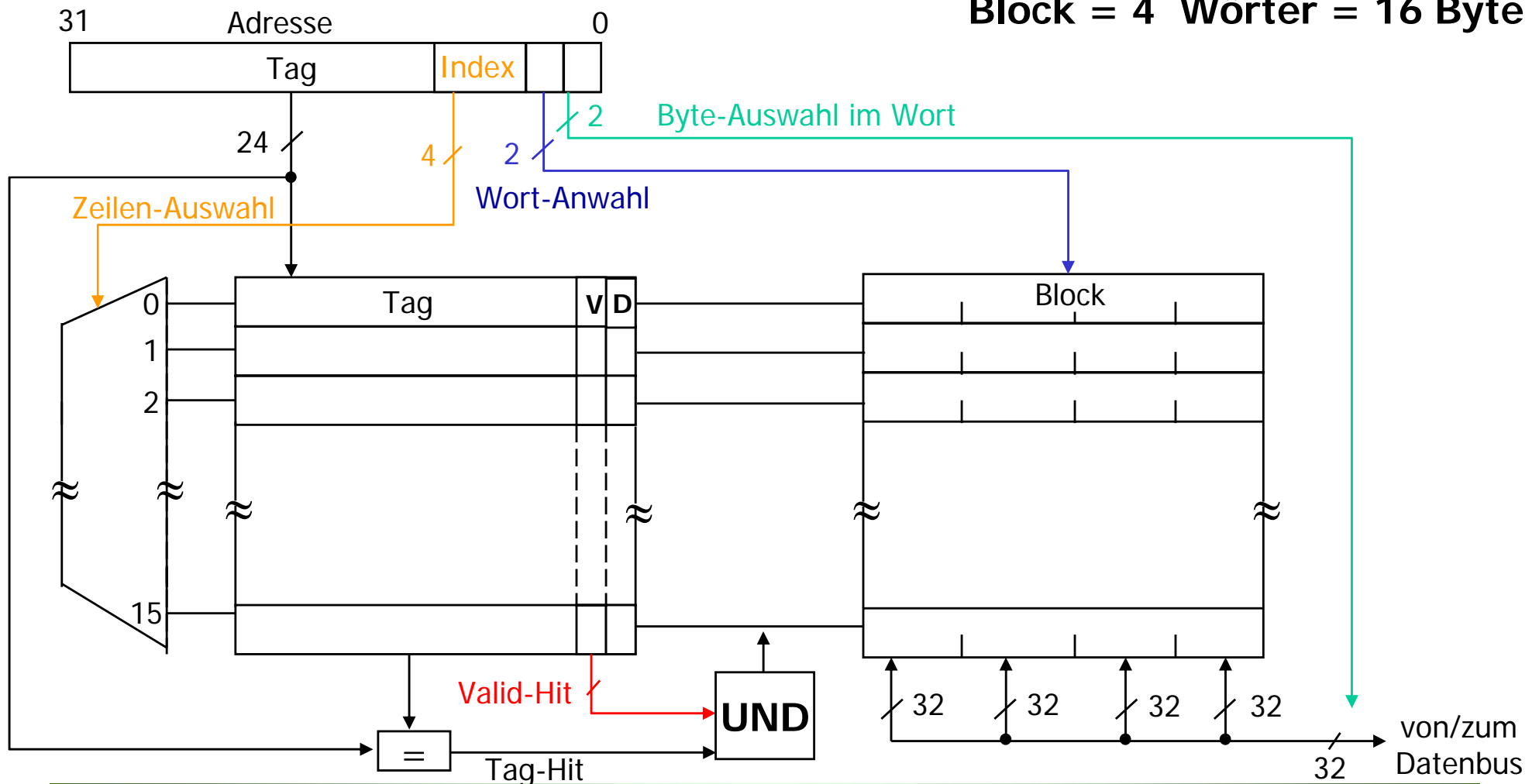
- ❑ Einfache Hardware-Realisierung (nur ein Vergleich und ein Tag-Speicher)
- ❑ Der Zugriff erfolgt schnell, weil das Tag-Feld parallel mit dem zugehörigen Block gelesen werden kann
- ❑ Es ist keine Ersetzungsstrategie erforderlich, weil die direkte Zuordnung keine Alternativen zulässt
- ❑ Auch wenn 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" (trashing)



# Beispiel: Direct-mapped-Cache

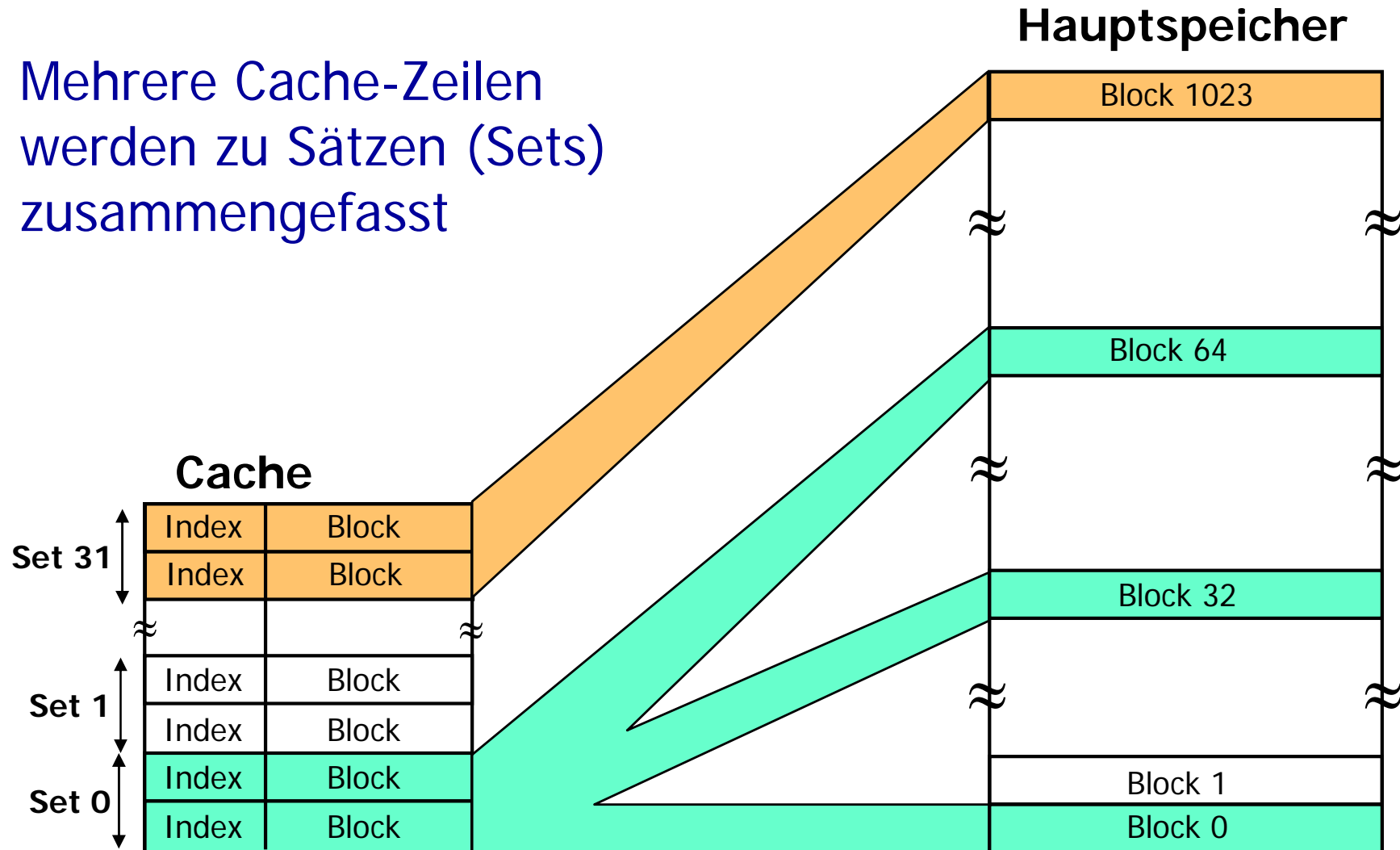
Kapazität: 256 Byte

Block = 4 Wörter = 16 Byte



# n-way-set-assoziativer Cache

Mehrere Cache-Zeilen  
werden zu Sätzen (Sets)  
zusammengefasst



# n-way-set-assoziativer Cache

---

- ❑ Kompromiss zwischen direct-mapped-Cache und vollassoziativen Cache.
- ❑ Verbesserte Trefferrate, da hier eine Auswahl möglich ist (der zu verdrängende Eintrag kann unter n ausgewählt werden).
- ❑ **Ersetzungsstrategie notwendig:**
  - Zyklisch (der zuerst eingelagerte Eintrag wird auch wieder verdrängt, FIFO Strategie)
  - LRU Strategie (*least recently used*) der am längsten nicht mehr benutzte Eintrag wird entfernt.
  - Zufällig (durch Zufallsgenerator)



# n-Way Set Associative Cache

---

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

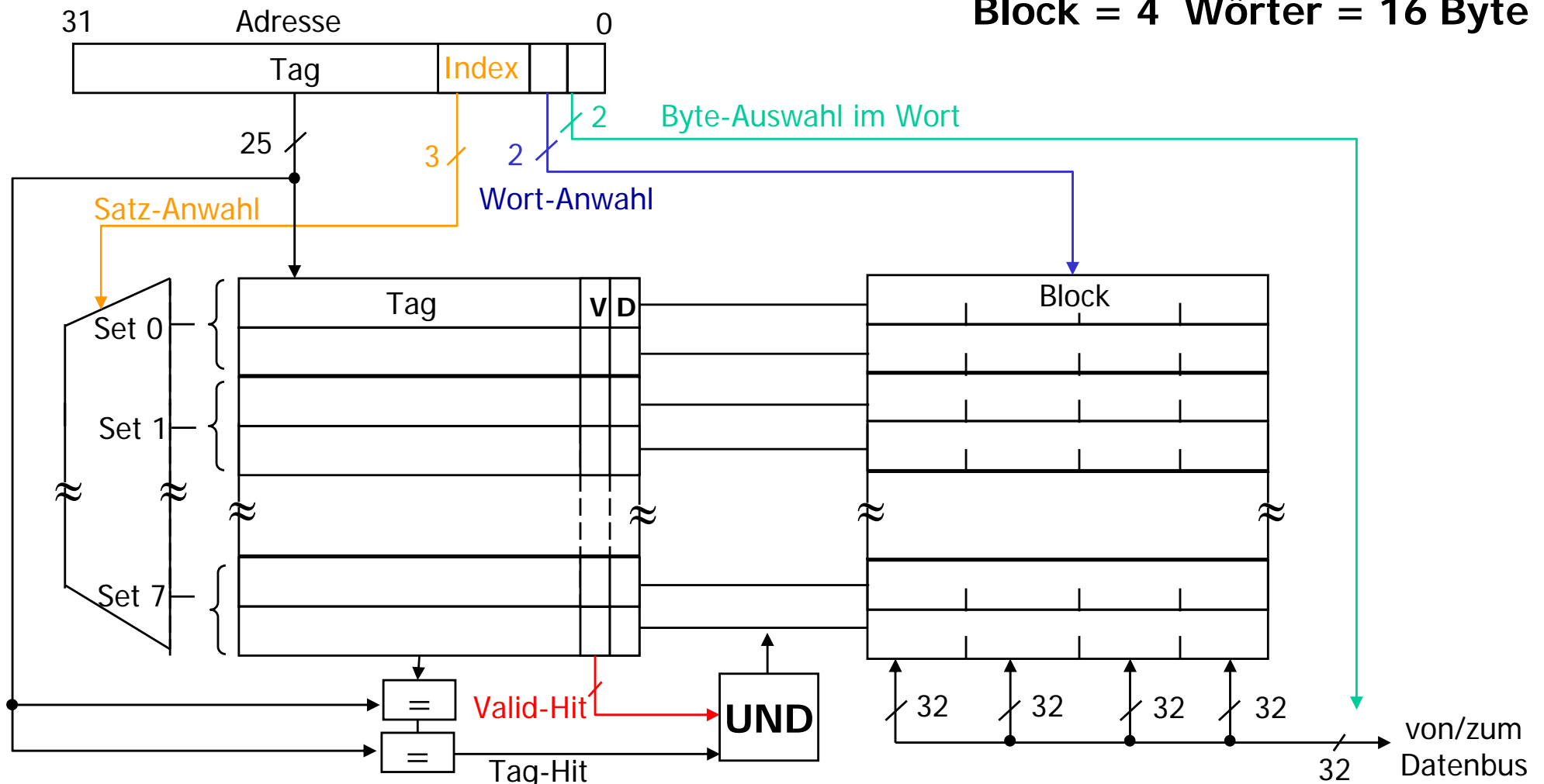
- der Aufwand steigt mit der Zahl  $n$ ,  
für große  $n$  nähert sich der Aufwand den  
voll-assoziativen Caches
- Kompromiß zwischen Direct Mapped Cache  
und voll-assoziativem Cache



# Beispiel: 2-way-set-assoziativer Cache

Kapazität: 256 Byte

Block = 4 Wörter = 16 Byte

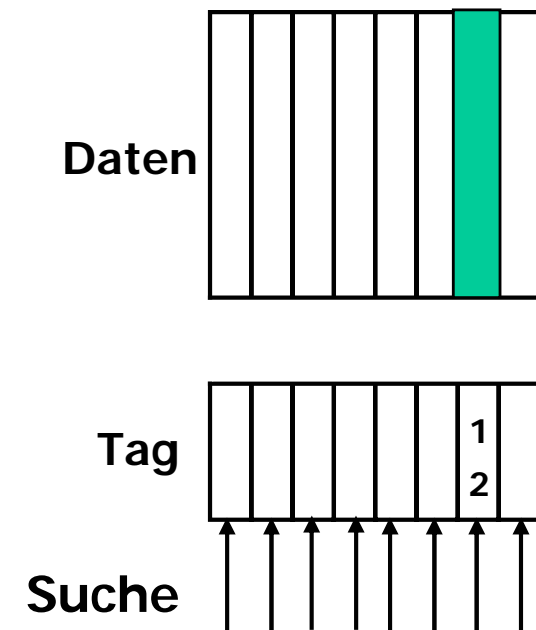
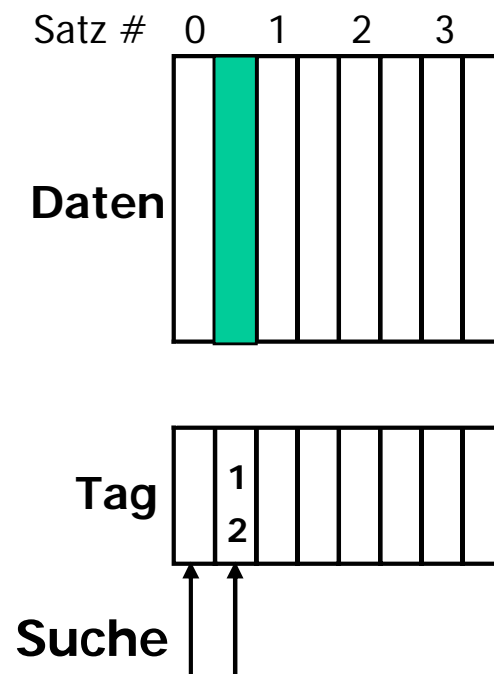
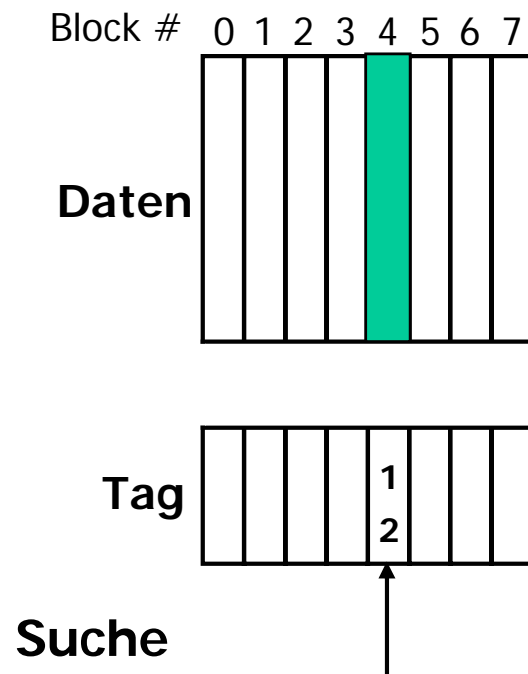


# Beispiel: Organisation eines Caches mit 8 Speicherplätzen

**Direct-mapped**

**Set-associative**

**fully associative**



# Beispiel: Organisation eines Caches mit 8 Speicherplätzen

---

## **Direct Mapped:**

Speicherblock 12 kann nur an einer Stelle stehen

## **2-Way Set Associative:**

Speicherblock 12 kann an zwei Stellen stehen

## **Voll-Assoziativ:**

Speicherblock 12 kann an allen Stellen stehen



# Ersetzungsstrategie

---

- ❑ **Ersetzungsstrategie** gibt an, welcher Teil des Cachespeichers nach einem Cache-Miss durch eine neu geladene Speicherportion überschrieben wird.
- ❑ **Ersetzungsstrategie** nur bei voll- oder n-fach satzassoziativer Cachespeicherorganisation angewandt
- ❑ meist die sehr einfache Strategie gewählt, die am längsten nicht benutzte Speicherportion zu ersetzen (**LRU-Strategie**, *Least Recently Used*).



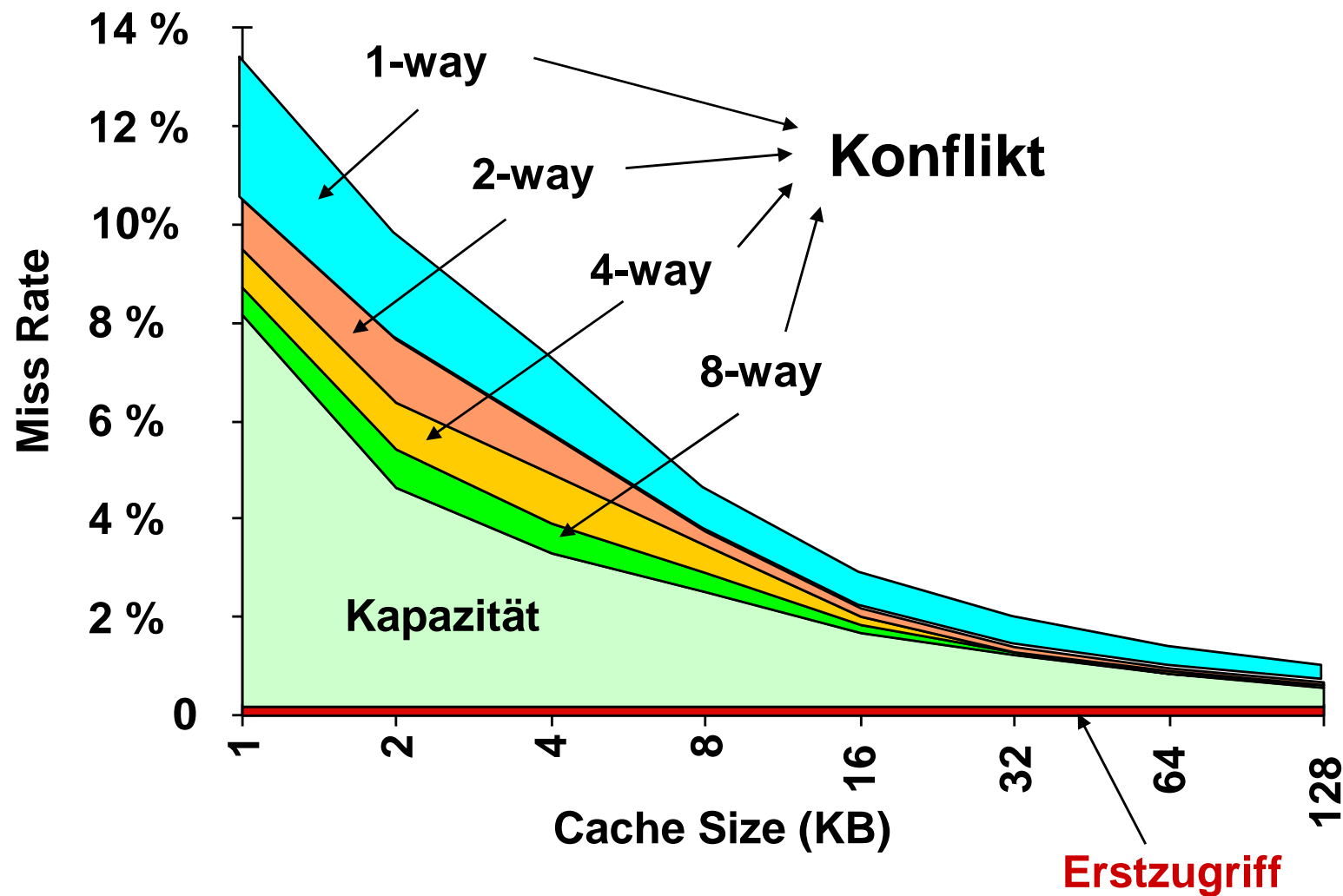
# Ursachen für die Fehlzugriffe

---

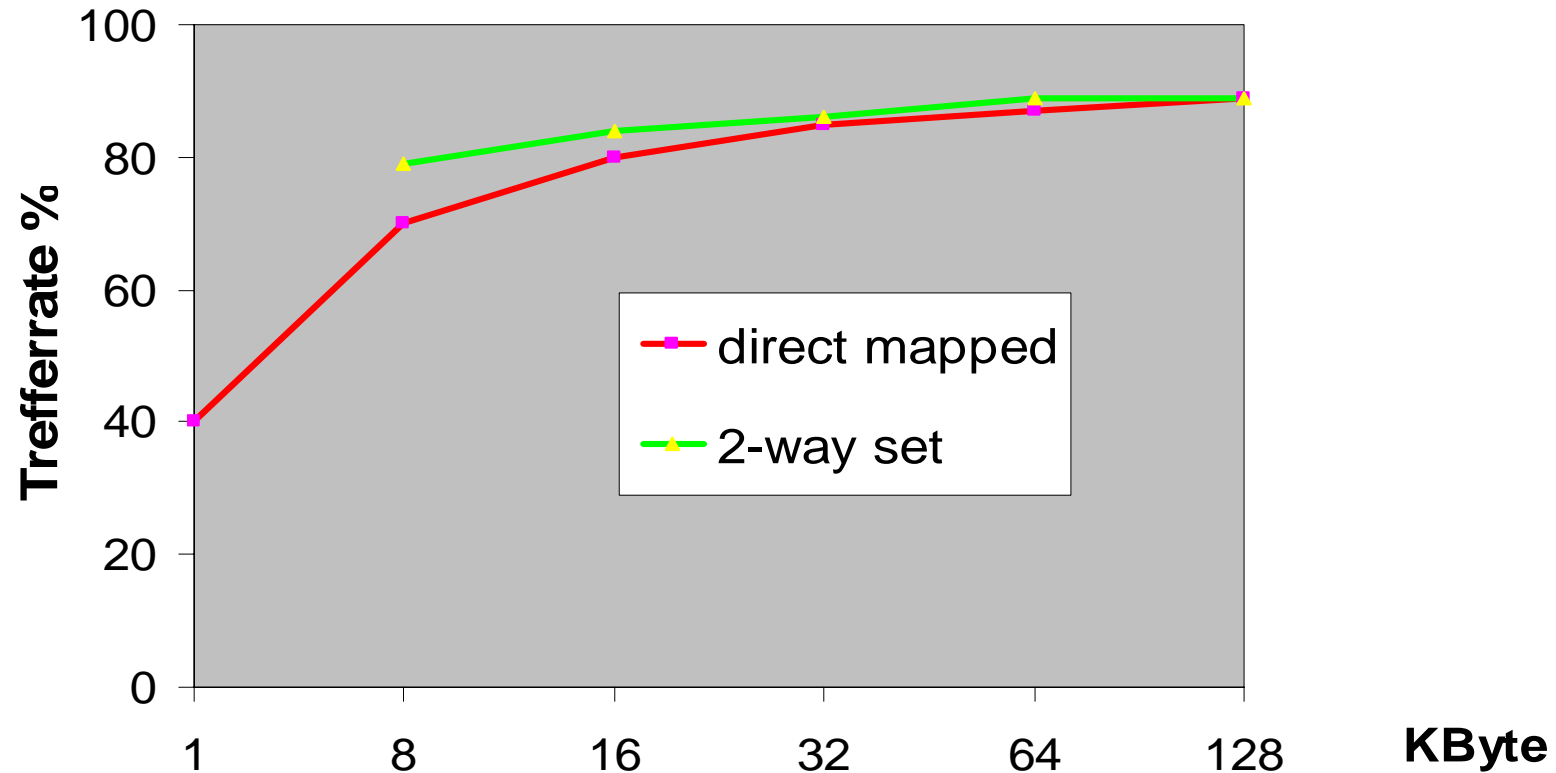
- ❑ **Erstzugriff** (*compulsory* - obligatorisch): Beim ersten Zugriff auf einen Cache-Block befindet sich dieser noch nicht im Cache-Speicher und muss erstmals geladen werden. Kaltstartfehlzugriffe (*cold start misses*) oder Erstbelegungsfehlzugriffe (*first reference misses*).
- ❑ **Kapazität** (*capacity*): Falls der Cache-Speicher nicht alle benötigten Cache-Blöcke aufnehmen kann, müssen Cache-Blöcke verdrängt und eventuell später wieder geladen werden.
- ❑ **Konflikt** (*conflict*): ein Cache-Block wird verdrängt und später wieder geladen, falls zu viele Cache-Blöcke auf denselben Satz abgebildet werden. Kollisionsfehlzugriffe (*collision misses*) oder Interferenzfehlzugriffe (*interference misses*). Kollisionsfehlzugriffe treten nur bei direkt abgebildeten oder satzassziativen Cache-Speichern beschränkter Größe auf.



# Ursachen für die Fehlzugriffe



# Erzielbare Cache-Trefferquoten



**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 - 8 fach assoziativer Organisation.



# Verwendung mehrerer Caches

---

Oft findet sich eine mehrstufige Cache-Organisation:

- ❑ Auf dem Prozessor-Chip befindet sich der sogenannte ***First-Level-Cache (On-Chip-Cache)***
  - ❑ Häufig getrennte On-Chip-Caches: **Befehlscache** für die Befehle und **Datencache** für die Daten.
- ➔ paralleler Zugriff auf Programm und Daten, wodurch die hohen Anforderungen bei heutigen Superskalar-Prozessoren an die Speicherbandbreiten erfüllt werden können



# Verwendung mehrerer Caches

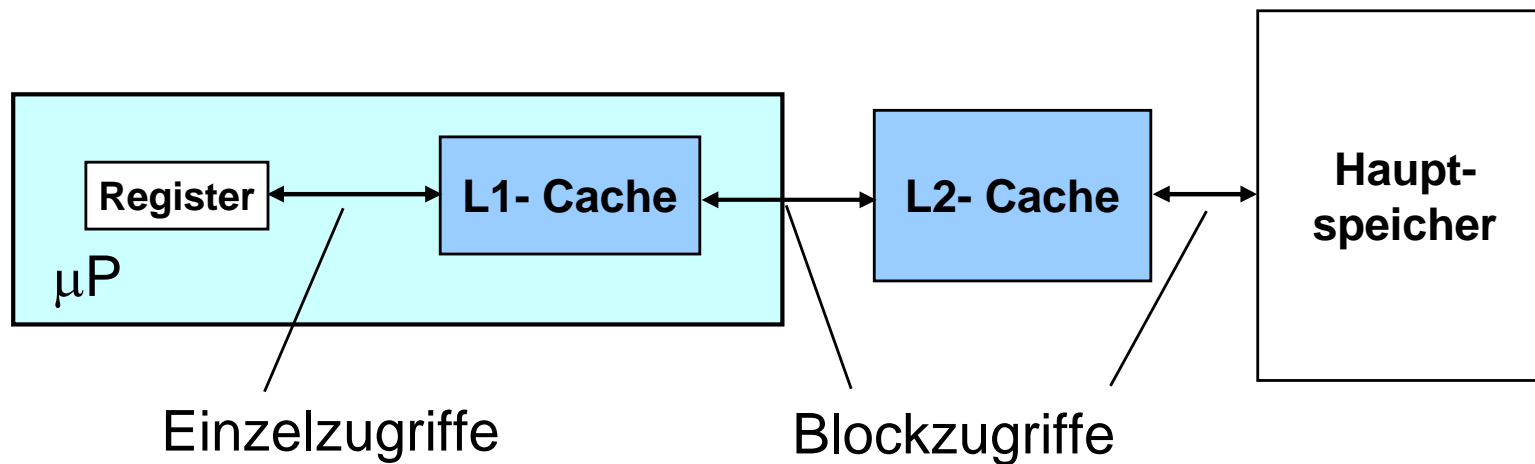
---

- ❑ Auf dem Chip wird eine **Harvard-Architektur** realisiert, bei der Programm und Daten in getrennten Speichern liegen
- ❑ Außerhalb des Prozessor-Chips befindet sich häufig ein weiterer, größerer Cache, der sogenannte **Secondary-Level-Cache** (**On-Board-Cache**, 64 - 1024 KByte groß)
- ❑ Der Secondary-Level-Cache kann parallel zum Hauptspeicher an den Bus angeschlossen werden (**Look-Aside-Cache**). Er sorgt dafür, dass bei einem First-Level-Cache-Miss die Daten schnell nachgeladen werden können

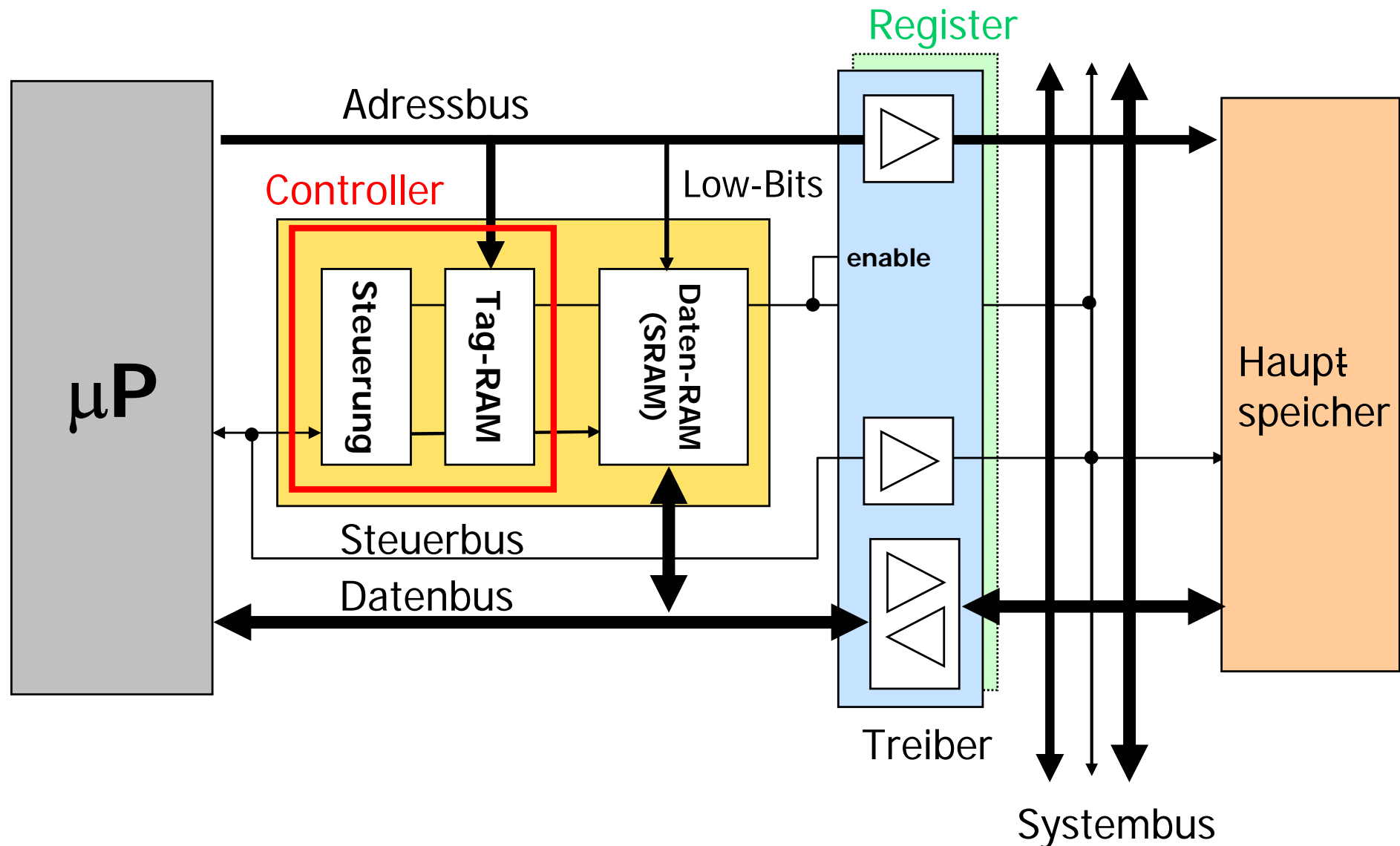


# On Chip und Off Chip Cache

- **On-Chip-Cache:** integriert auf dem Prozessorchip
  - Sehr kurze Zugriffszeiten (wie die der prozessorinternen Register)
  - Aus technologischen Gründen begrenzte Kapazität
- **Off-Chip-Cache:** prozessorextern (SRAM-Speicher)



# Anbindung des Caches an den Systembus



# Anbindung des Caches an den Systembus

---

- ❑ **Cache-Controller:**

Tag-RAM + Steuerung + Tag-Komparator

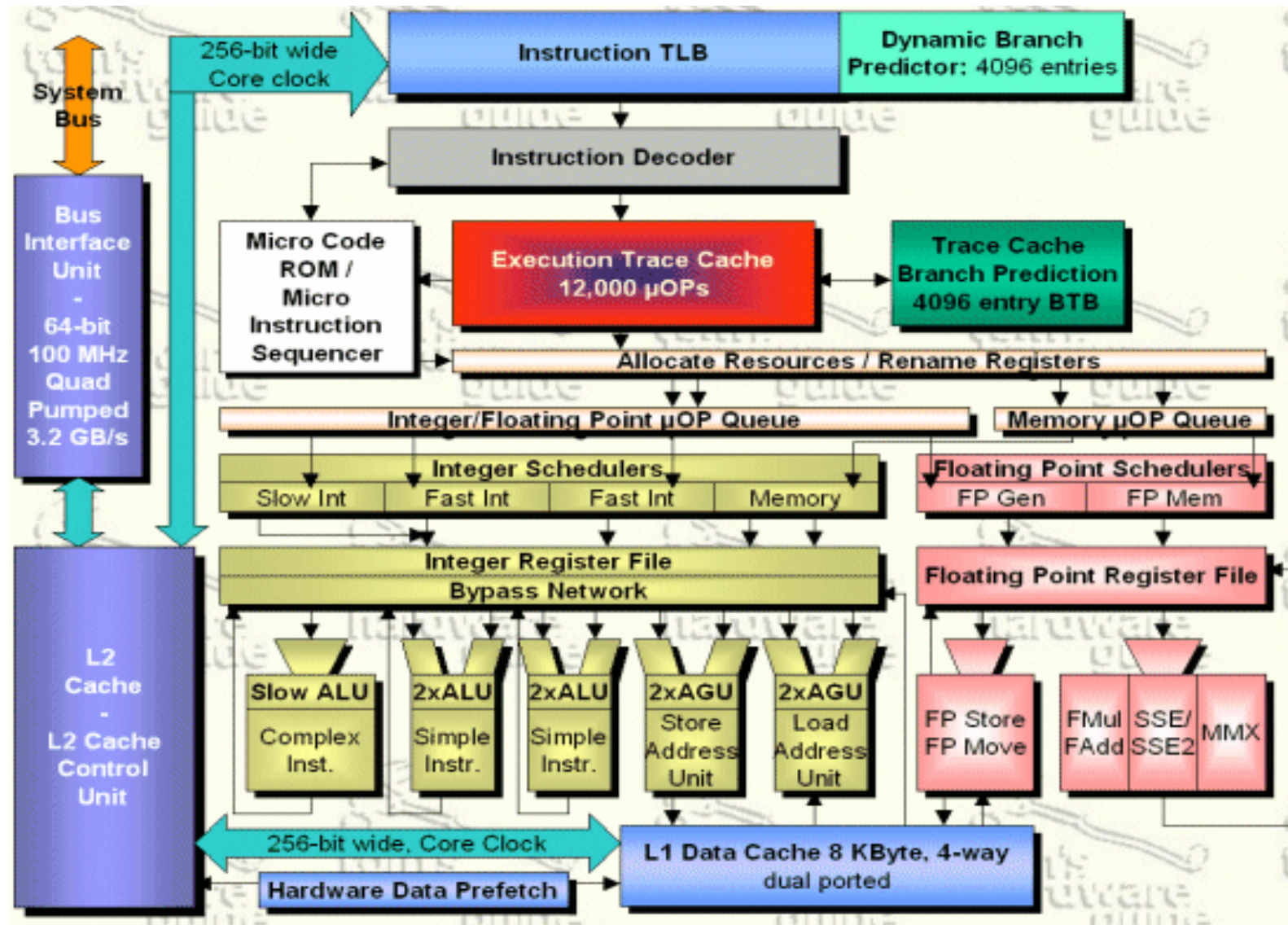
Da dieser sehr schnell sein muß → auf einem Chip integriert

- ❑ Cachespeicher selbst ist separat mit SRAM-Bausteinen aufgebaut

- ❑ Cache-Controller übernimmt die Steuerung der Treiber zum Systembus (Systembuszugriff nur bei Cache-Miss, sonst ist der Systembus für andere Komponenten frei), sowie der Systembussignale zur Einfügung von Wartezyklen bei Cache-Miss (READY, HOLD, HOLDA, ...)



# Cache-Speicher in Pentium 4



# Cache-Speicher in Pentium 4

---

## □ L1-Cache:

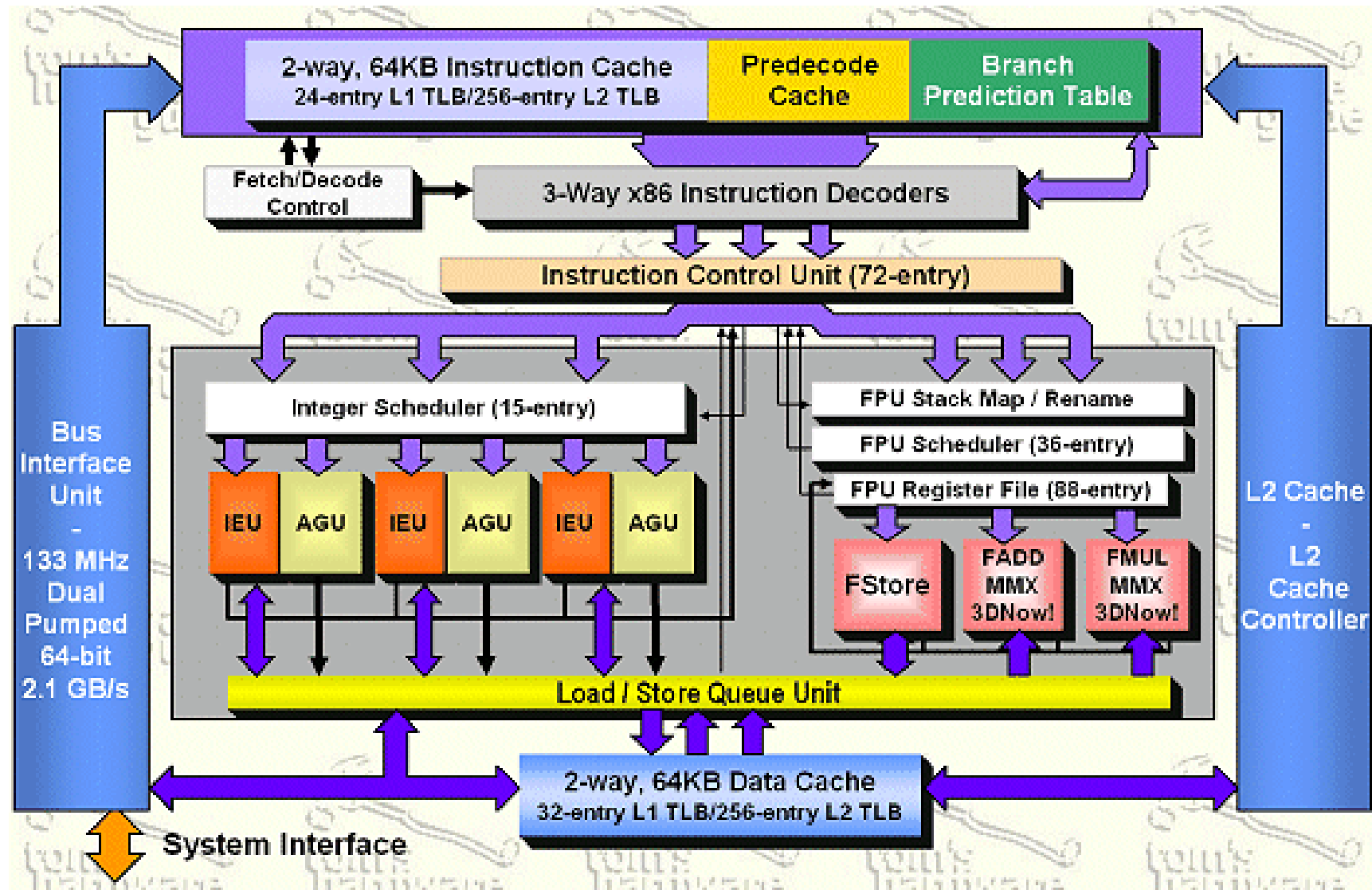
- 8 Kbyte Daten-Cache (Bei Pentium III: 16 Kbyte; bei Athlon: 64 Kbyte)
- 4-way set associative Daten-Cache
- Cache-lines mit 64 Byte (dual-pot-Architektur)
- Write-Through
- 12K  $\mu$ OPs „Trace“-Cache

## □ L2-Cache:

- 256 Kbyte
- 8-way set associative
- Cache-lines mit 128 Byte
- Write back
- Bandbreite 44,8 Gbyte/s (16 Gbyte/s bei Pentium III)



# Cache-Speicher bei Athlon



## Fragen, die sich ein Speicherhierarchie-Designer stellen muss:

---

- ❑ Wohin kann ein Block abgebildet werden?  
(Block-Abbildungsstrategie)
  - Vollasoziativ, Satz Assoziativ, Direct- Mapped
- ❑ Wie kann ein Block gefunden werden?  
(Block-Identifikation)
  - Tag/Block
- ❑ Welcher Block soll bei einem Miss ersetzt werden?  
(Block-Ersetzungsstrategie)
  - Random, FIFO, LRU
- ❑ Was passiert bei einem Schreibzugriff?  
(Schreibe-Strategie)
  - Write back oder Write Through (mit Write Buffer)

