Speichermanagement 1: *Geschwindigkeit*

Caches und Hierarchien

Technische Grundlagen der Informatik

Automation Systems Group E183-1

Institute of Computer Aided Automation
Vienna University of Technology

email: tgi@auto.tuwien.ac.at

Recap

Micro16 Upgrades bisher

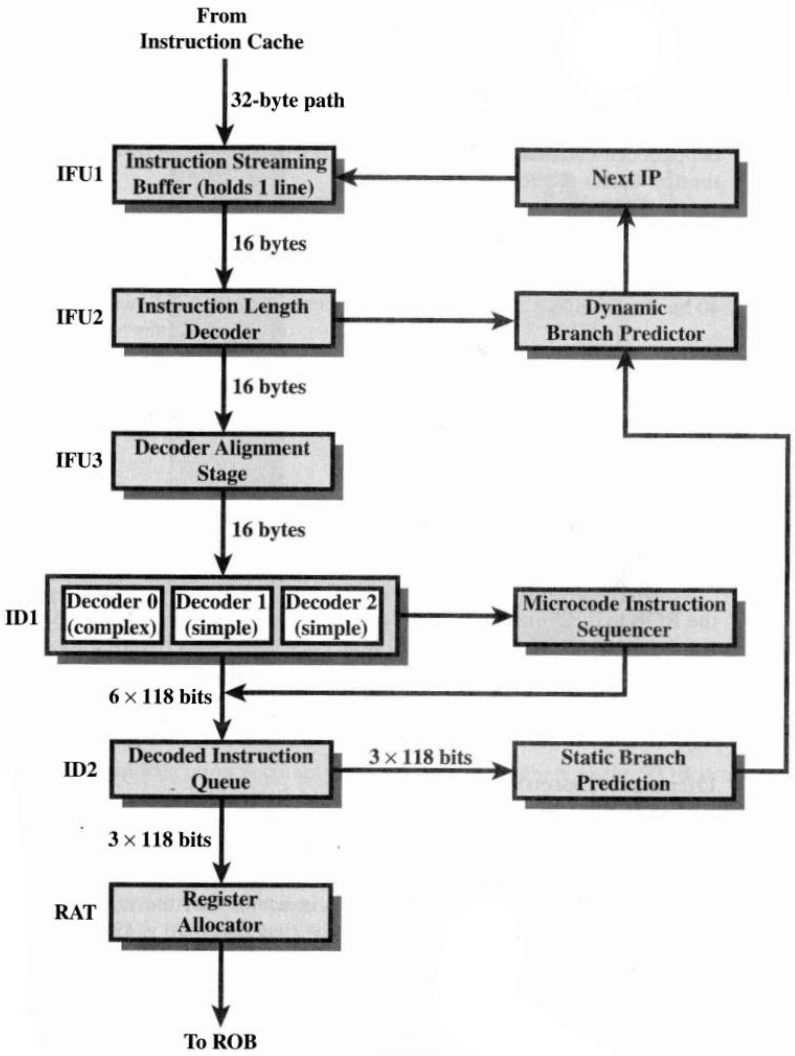
- Befehlssatz zu eingeschränkt / im ROM
  - Mehr Befehle in Hardware verbauen (CISC)
  - Interpreter in Mikrocode – Befehle im RAM (RISC)
- Hardware nicht optimal genutzt
  - Pipelining
- Heute: Speicher

- David A. Patterson, John L. Hennessy, Computer Organization and Design: The Hardware/Software Interface, 2nd ed., Morgan Kaufmann, 2001

Pipeline – Pentium II (1997)

IFU1	IFU2	IFU3	ID1	ID2	RAT	ROB	DIS	EX	RET1	RET2

IFU = Instruction Fetch Unit
ID = Instruction Decode
RAT = Register Allocator
ROB = Reorder Buffer
DIS = Dispatcher
EX = Execute Stage
RET = Retire Unit

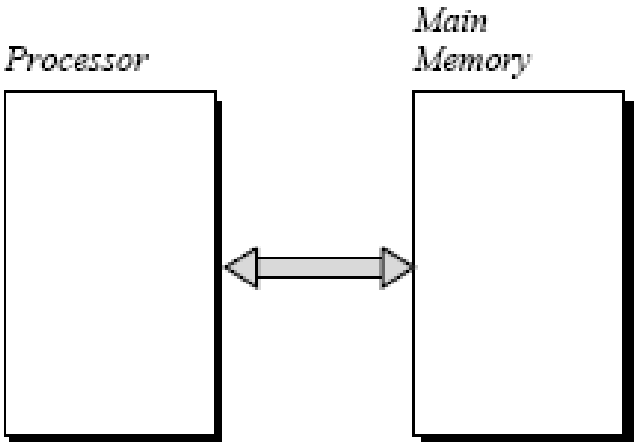


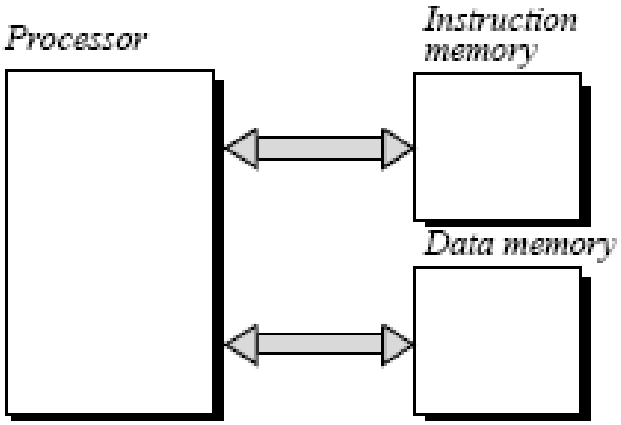
Rückblick - Speicherbausteine

- Statisches RAM
  - Information wird in Latches gespeichert
  -  Sehr kurze Zugriffszeiten
  -  Hoher Preis, kaum hoch integrierbar

- Dynamisches RAM
  - Information wird in Kondensatoren gespeichert
  - Refresh Cycle
    - Burst Refresh
    - Cycle Stealing
    - Transparent Refresh

Von Neumann vs Harvard Architektur

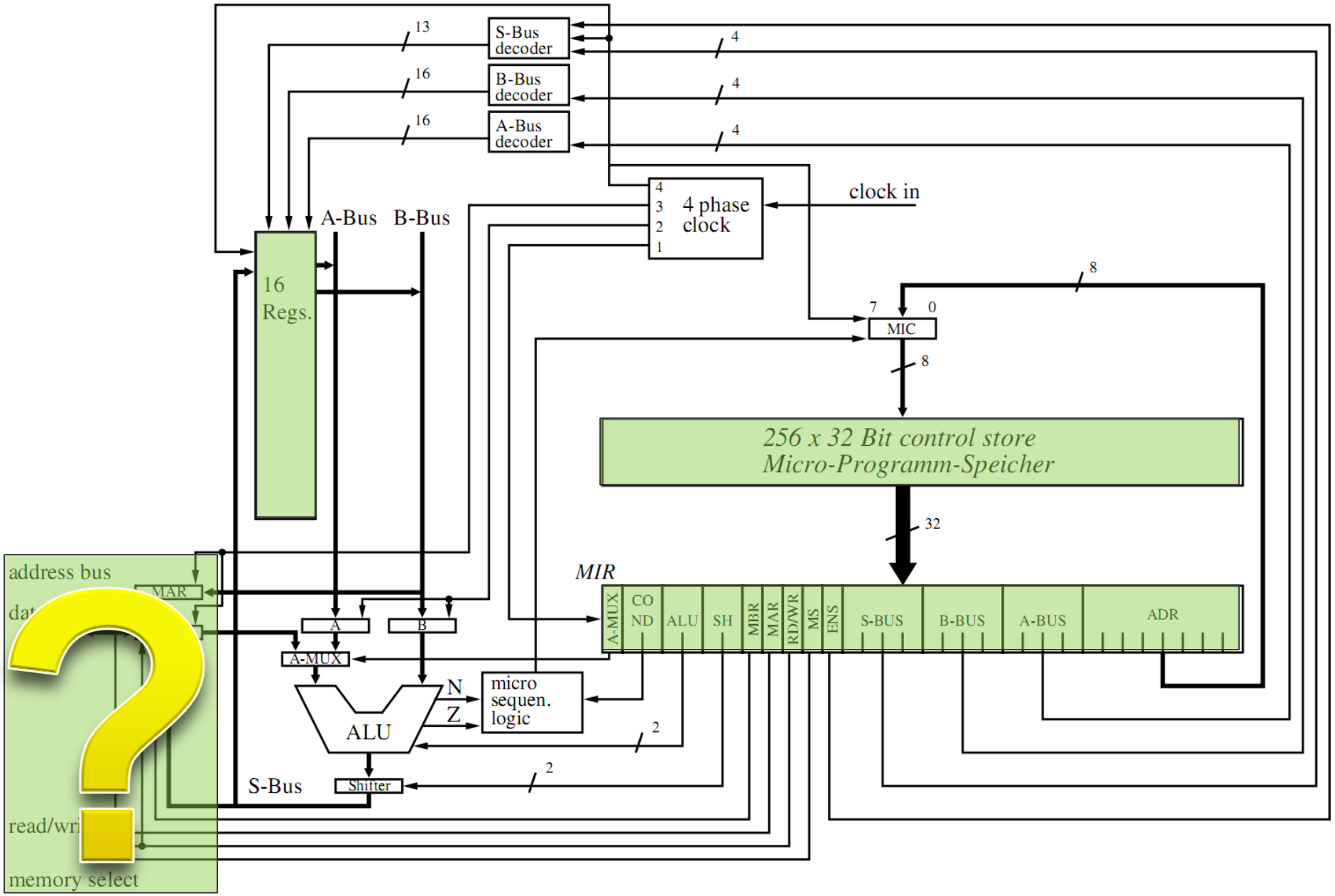




- Von-Neumann Architektur
- Speicher enthält sowohl Programme als auch Daten

- Harvard Architektur
- Getrennter Programm- und Datenspeicher
- Befehle und Daten können gleichzeitig geladen bzw. geschrieben werden

Preisfrage:
Welche Architektur hat die Micro16 ?


S-Bus decoder
B-Bus decoder
A-Bus decoder
13
16
16
4
4
4
4 phase clock
clock in
A-Bus
B-Bus
16 Regs.
MIC
8
8
7
0
256 x 32 Bit control store
Micro-Programm-Speicher
32
MIR
A-MUX
COND
ALU
SH
MBR
MAR
RD/WR
MS
ENS
S-BUS
B-BUS
A-BUS
ADR
address bus
MAR
A
B
A-MUX
ALU
N
Z
micro sequen. logic
2
2
S-Bus
Shifter
read/wri
memory select

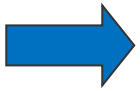
Das Problem – €

große & schnelle Speicher sind unbezahlbar

Speichertyp	Kapazität	Zugriffszeit	Kosten / Einheit
Synchronous SRAM chip	18 Mbit	3.4ns	11 €/MB
DDR3-1600 Modul	8GiB	10-40ns / ~200ns mit Speichercontroller	7 €/GiB
Flash SSD	1TB	100 ns	0,33 €/GB
Festplatte	4TB	10ms	30 €/TB
Blue Ray 4x	25GB	180ms	19,5 €/TB

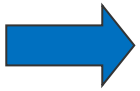
Lichtblick: “Locality”

- Wurde auf einen Speicherinhalt erst kürzlich zugegriffen, so ist die Wahrscheinlichkeit eines baldigen neuerlichen Zugriffs relativ hoch.



Temporal Locality

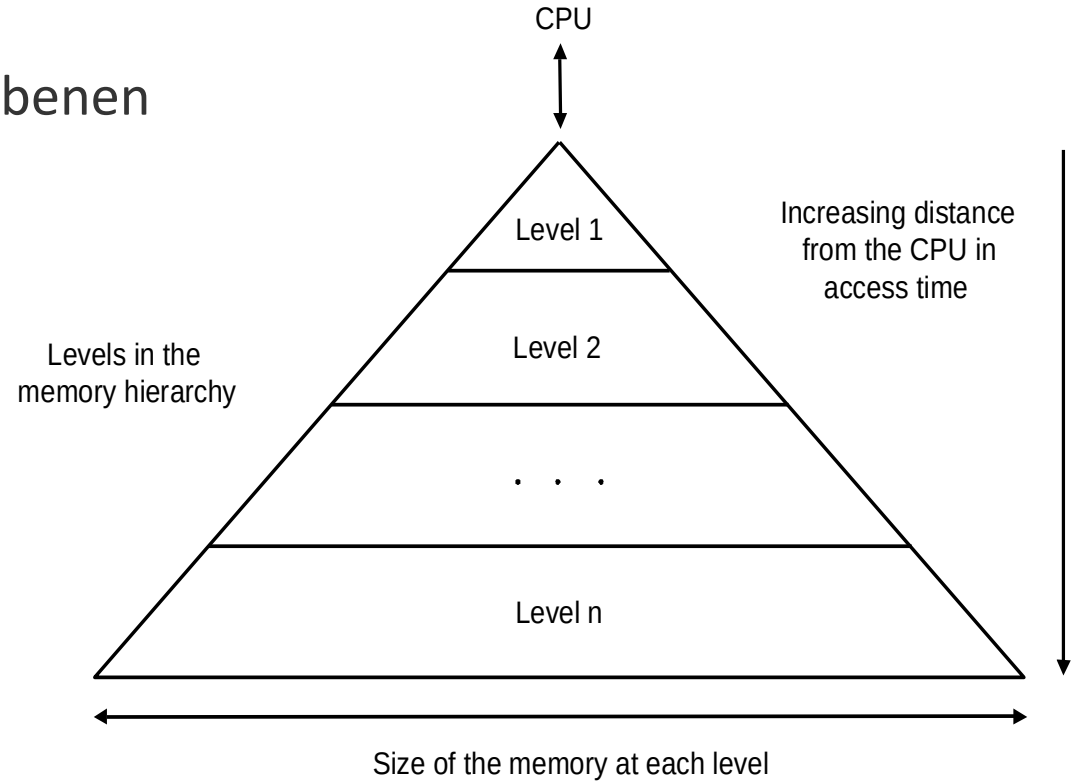
- Wird auf einen Speicherinhalt gerade zugegriffen, so ist es relativ wahrscheinlich, dass der nächste Zugriff in dessen Nachbarschaft erfolgen wird.



Spatial Locality

Die Lösung: Speicherhierarchie

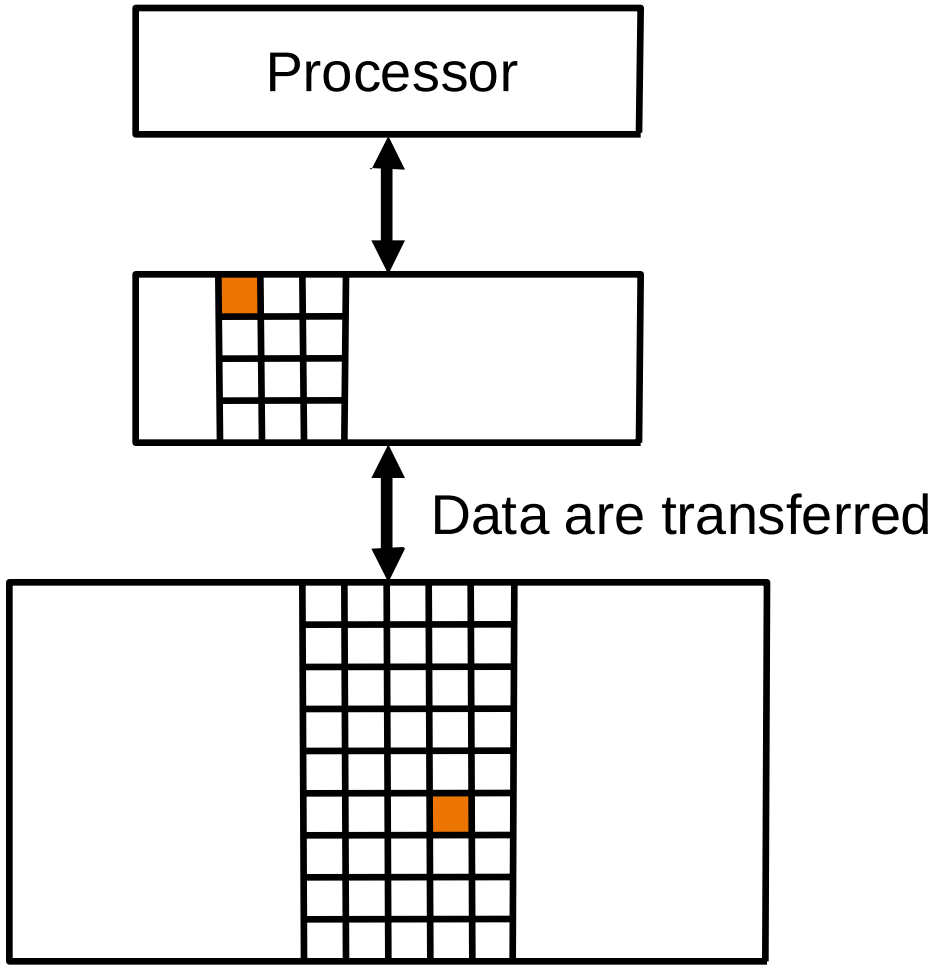
- Speicherinhalte werden “gepuffert”
- Staffelung in verschiedene Ebenen
- Je näher an der CPU umso
  - kleiner und
  - schneller der Speicher



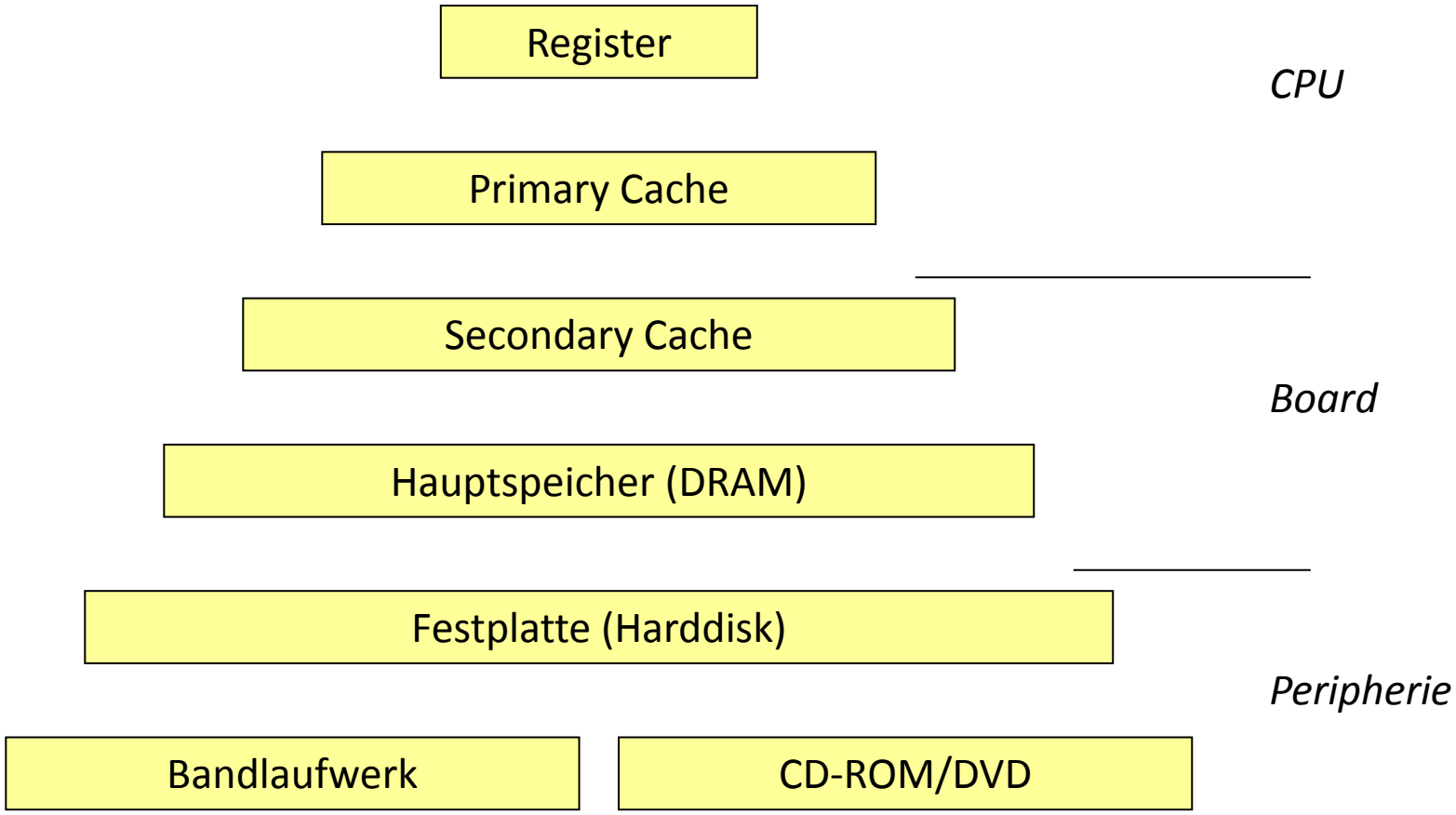
Ziel: Zugriffszeit von Ebene 1 für alle Daten von Ebene n

Terminologie

- Block
- Hit
- Hit-rate (Hit-ratio)
- Hit-time
- Miss
- Miss-rate
- Miss-penalty



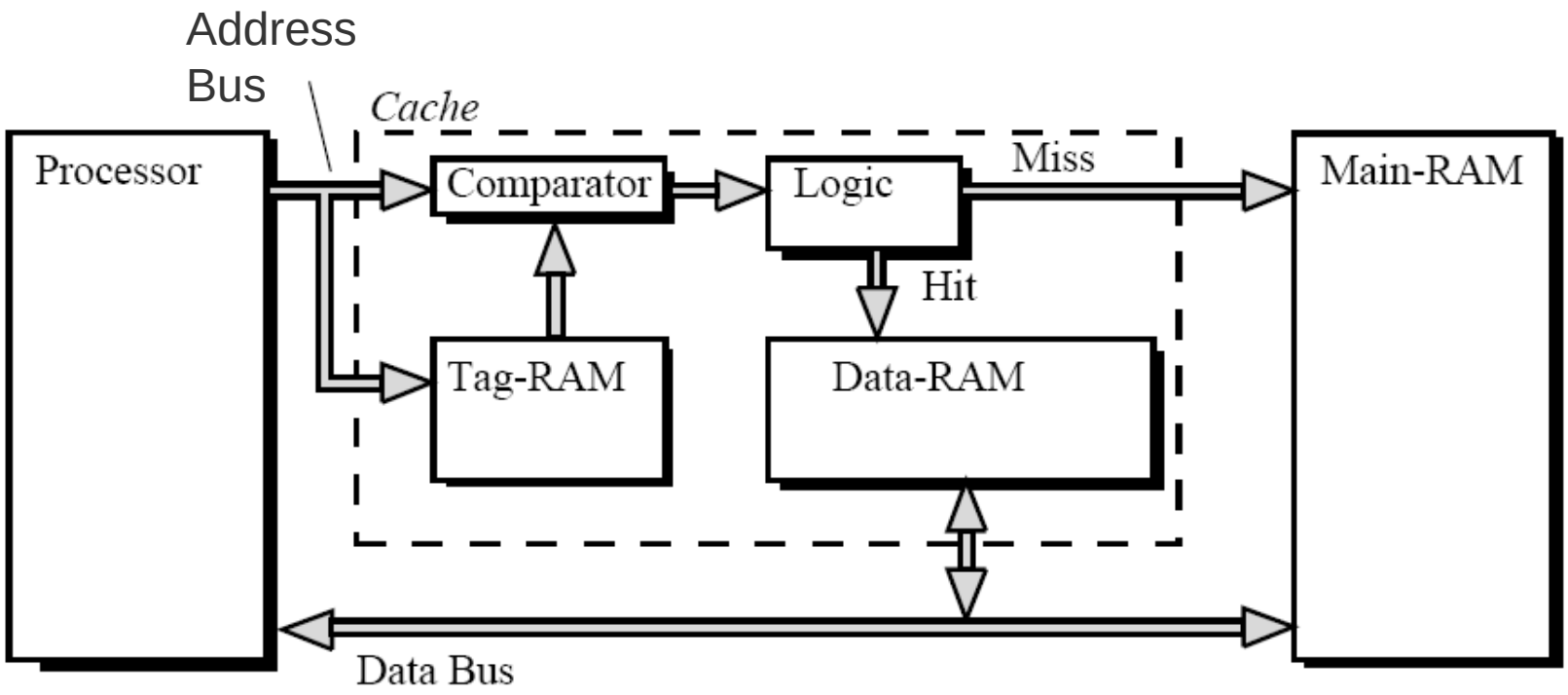
Typische Speicherstruktur



Cache: Prinzip

- SRAM ermöglicht extrem schnellen Zugriff der CPU auf Daten/Befehle
- Bei “miss” werden Daten aus dem Hauptspeicher (DRAM) nachgeladen
- Probleme:
  - Wie weiß ich, ob die benötigten Daten im Cache sind ?
  - Falls ja, wie finde ich sie ?

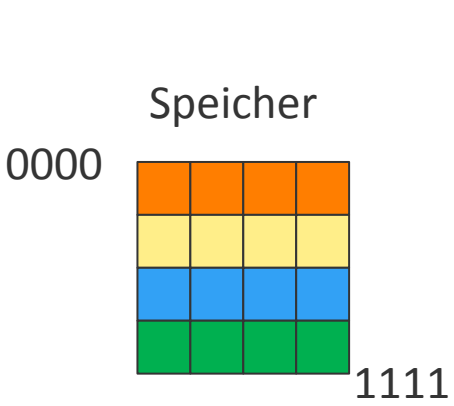
Cache Memory

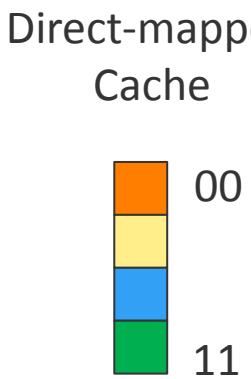


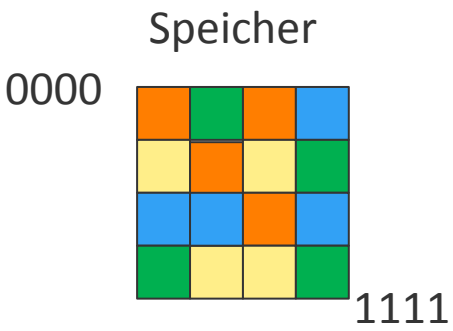
Cache-Verwaltungsstrategien

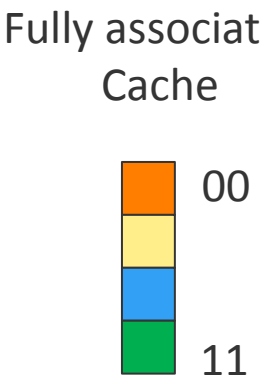
- Direct-mapped Cache
  - Zu jeder Speicheradresse gibt es eine genau festgelegte Position im Cache

- Fully associative Cache
  - Jede beliebige Position im Cache darf verwendet werden









Direct-mapped Cache

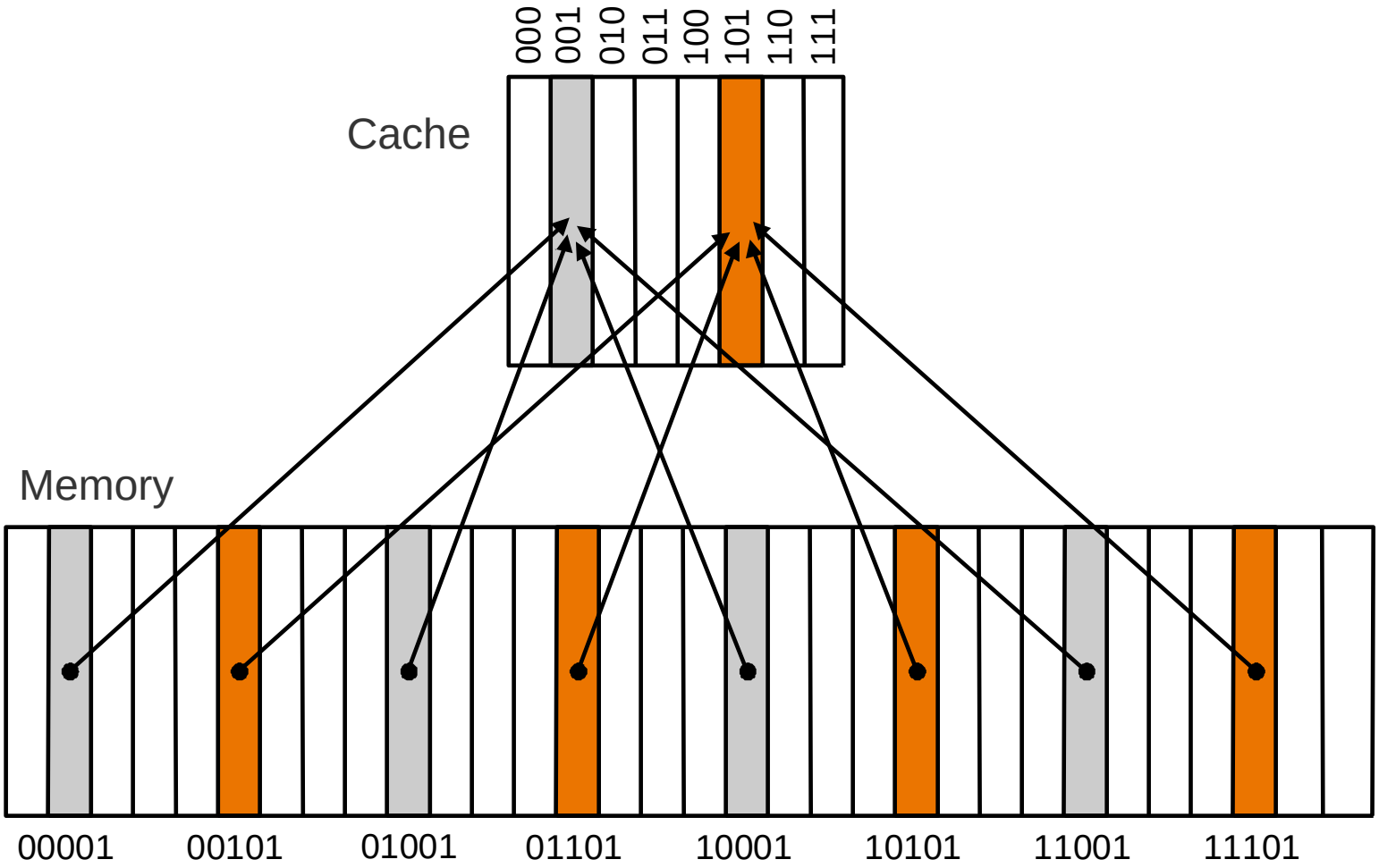
- Ein Teil der Adresse bestimmt die Position im Cache (z.B. niederwertige Bits): “Cache-Index”

  Für jede Speicherzelle im DRAM ist daher die Position im Cache fix vorgegeben => “direct mapped”

- Restliche Adressbits werden zusätzlich zu den Daten abgespeichert: “Cache-Tag”

- “Valid-Bit” zur Erkennung bisher unbenutzter Positionen

Direct-mapped Cache: Prinzip



Direct-mapped Cache: Zugriff

Index	Valid	Tag	Data
$(000)_2$	N		
$(001)_2$	N		
$(010)_2$	N		
$(011)_2$	N		
$(100)_2$	N		
$(101)_2$	N		
$(110)_2$	N		
$(111)_2$	N		

Direct-mapped Cache: Zugriff

Address	Address	Data	Hit/Miss
$(22)_{10}$	$(10110)_2$	12	

Index	Valid	Tag	Data
$(000)_2$	N		
$(001)_2$	N		
$(010)_2$	N		
$(011)_2$	N		
$(100)_2$	N		
$(101)_2$	N		
$(110)_2$	N		
$(111)_2$	N		

Direct-mapped Cache: Zugriff

Address	Address	Data	Hit/Miss
$(22)_{10}$	$(10110)_2$	12	M

Index	Valid	Tag	Data
$(000)_2$	N		
$(001)_2$	N		
$(010)_2$	N		
$(011)_2$	N		
$(100)_2$	N		
$(101)_2$	N		
$(110)_2$	N		
$(111)_2$	N		

Direct-mapped Cache: Zugriff

Address	Address	Data	Hit/Miss
$(22)_{10}$	$(10110)_2$	12	M

Index	Valid	Tag	Data
$(000)_2$	N		
$(001)_2$	N		
$(010)_2$	N		
$(011)_2$	N		
$(100)_2$	N		
$(101)_2$	N		
$(110)_2$	Y	$(10)_2$	12
$(111)_2$	N		

Direct-mapped Cache: Zugriff

Address	Address	Data	Hit/Miss
$(22)_{10}$	$(10110)_2$	12	M
$(26)_{10}$	$(11010)_2$	AB	

Index	Valid	Tag	Data
$(000)_2$	N		
$(001)_2$	N		
$(010)_2$	N		
$(011)_2$	N		
$(100)_2$	N		
$(101)_2$	N		
$(110)_2$	Y	$(10)_2$	12
$(111)_2$	N		

Direct-mapped Cache: Zugriff

Address	Address	Data	Hit/Miss
$(22)_{10}$	$(10110)_2$	12	M
$(26)_{10}$	$(11010)_2$	AB	M

Index	Valid	Tag	Data
$(000)_2$	N		
$(001)_2$	N		
$(010)_2$	N		
$(011)_2$	N		
$(100)_2$	N		
$(101)_2$	N		
$(110)_2$	Y	$(10)_2$	12
$(111)_2$	N		

Direct-mapped Cache: Zugriff

Address	Address	Data	Hit/Miss
$(22)_{10}$	$(10110)_2$	12	M
$(26)_{10}$	$(11010)_2$	AB	M

Index	Valid	Tag	Data
$(000)_2$	N		
$(001)_2$	N		
$(010)_2$	Y	$(11)_2$	AB
$(011)_2$	N		
$(100)_2$	N		
$(101)_2$	N		
$(110)_2$	Y	$(10)_2$	12
$(111)_2$	N		

Direct-mapped Cache: Zugriff

Address	Address	Data	Hit/Miss
$(22)_{10}$	$(10110)_2$	12	M
$(26)_{10}$	$(11010)_2$	AB	M
$(22)_{10}$	$(10110)_2$	12	

Index	Valid	Tag	Data
$(000)_2$	N		
$(001)_2$	N		
$(010)_2$	Y	$(11)_2$	AB
$(011)_2$	N		
$(100)_2$	N		
$(101)_2$	N		
$(110)_2$	Y	$(10)_2$	12
$(111)_2$	N		

Direct-mapped Cache: Zugriff

Address	Address	Data	Hit/Miss
$(22)_{10}$	$(10110)_2$	12	M
$(26)_{10}$	$(11010)_2$	AB	M
$(22)_{10}$	$(10110)_2$	12	H

Index	Valid	Tag	Data
$(000)_2$	N		
$(001)_2$	N		
$(010)_2$	Y	$(11)_2$	AB
$(011)_2$	N		
$(100)_2$	N		
$(101)_2$	N		
$(110)_2$	Y	$(10)_2$	12
$(111)_2$	N		

Direct-mapped Cache: Zugriff

Address	Address	Data	Hit/Miss
$(22)_{10}$	$(10110)_2$	12	M
$(26)_{10}$	$(11010)_2$	AB	M
$(22)_{10}$	$(10110)_2$	12	H
$(16)_{10}$	$(10000)_2$	FE	

Index	Valid	Tag	Data
$(000)_2$	N		
$(001)_2$	N		
$(010)_2$	Y	$(11)_2$	AB
$(011)_2$	N		
$(100)_2$	N		
$(101)_2$	N		
$(110)_2$	Y	$(10)_2$	12
$(111)_2$	N		

Direct-mapped Cache: Zugriff

Address	Address	Data	Hit/Miss
$(22)_{10}$	$(10110)_2$	12	M
$(26)_{10}$	$(11010)_2$	AB	M
$(22)_{10}$	$(10110)_2$	12	H
$(16)_{10}$	$(10000)_2$	FE	M

Index	Valid	Tag	Data
$(000)_2$	N		
$(001)_2$	N		
$(010)_2$	Y	$(11)_2$	AB
$(011)_2$	N		
$(100)_2$	N		
$(101)_2$	N		
$(110)_2$	Y	$(10)_2$	12
$(111)_2$	N		

Direct-mapped Cache: Zugriff

Address	Address	Data	Hit/Miss
$(22)_{10}$	$(10110)_2$	12	M
$(26)_{10}$	$(11010)_2$	AB	M
$(22)_{10}$	$(10110)_2$	12	H
$(16)_{10}$	$(10000)_2$	FE	M

Index	Valid	Tag	Data
$(000)_2$	Y	$(10)_2$	FE
$(001)_2$	N		
$(010)_2$	Y	$(11)_2$	AB
$(011)_2$	N		
$(100)_2$	N		
$(101)_2$	N		
$(110)_2$	Y	$(10)_2$	12
$(111)_2$	N		

Direct-mapped Cache: Zugriff

Address	Address	Data	Hit/Miss
$(22)_{10}$	$(10110)_2$	12	M
$(26)_{10}$	$(11010)_2$	AB	M
$(22)_{10}$	$(10110)_2$	12	H
$(16)_{10}$	$(10000)_2$	FE	M
$(3)_{10}$	$(00011)_2$	AB	

Index	Valid	Tag	Data
$(000)_2$	Y	$(10)_2$	FE
$(001)_2$	N		
$(010)_2$	Y	$(11)_2$	AB
$(011)_2$	N		
$(100)_2$	N		
$(101)_2$	N		
$(110)_2$	Y	$(10)_2$	12
$(111)_2$	N		

Direct-mapped Cache: Zugriff

Address	Address	Data	Hit/Miss
$(22)_{10}$	$(10110)_2$	12	M
$(26)_{10}$	$(11010)_2$	AB	M
$(22)_{10}$	$(10110)_2$	12	H
$(16)_{10}$	$(10000)_2$	FE	M
$(3)_{10}$	$(00011)_2$	AB	M

Index	Valid	Tag	Data
$(000)_2$	Y	$(10)_2$	FE
$(001)_2$	N		
$(010)_2$	Y	$(11)_2$	AB
$(011)_2$	N		
$(100)_2$	N		
$(101)_2$	N		
$(110)_2$	Y	$(10)_2$	12
$(111)_2$	N		

Direct-mapped Cache: Zugriff

Address	Address	Data	Hit/Miss
$(22)_{10}$	$(10110)_2$	12	M
$(26)_{10}$	$(11010)_2$	AB	M
$(22)_{10}$	$(10110)_2$	12	H
$(16)_{10}$	$(10000)_2$	FE	M
$(3)_{10}$	$(00011)_2$	AB	M

Index	Valid	Tag	Data
$(000)_2$	Y	$(10)_2$	FE
$(001)_2$	N		
$(010)_2$	Y	$(11)_2$	AB
$(011)_2$	Y	$(00)_2$	AB
$(100)_2$	N		
$(101)_2$	N		
$(110)_2$	Y	$(10)_2$	12
$(111)_2$	N		

Direct-mapped Cache: Zugriff

Address	Address	Data	Hit/Miss
$(22)_{10}$	$(10110)_2$	12	M
$(26)_{10}$	$(11010)_2$	AB	M
$(22)_{10}$	$(10110)_2$	12	H
$(16)_{10}$	$(10000)_2$	FE	M
$(3)_{10}$	$(00011)_2$	AB	M
$(16)_{10}$	$(10000)_2$	FE	

Index	Valid	Tag	Data
$(000)_2$	Y	$(10)_2$	FE
$(001)_2$	N		
$(010)_2$	Y	$(11)_2$	AB
$(011)_2$	Y	$(00)_2$	AB
$(100)_2$	N		
$(101)_2$	N		
$(110)_2$	Y	$(10)_2$	12
$(111)_2$	N		

Direct-mapped Cache: Zugriff

Address	Address	Data	Hit/Miss
$(22)_{10}$	$(10110)_2$	12	M
$(26)_{10}$	$(11010)_2$	AB	M
$(22)_{10}$	$(10110)_2$	12	H
$(16)_{10}$	$(10000)_2$	FE	M
$(3)_{10}$	$(00011)_2$	AB	M
$(16)_{10}$	$(10000)_2$	FE	H

Index	Valid	Tag	Data
$(000)_2$	Y	$(10)_2$	FE
$(001)_2$	N		
$(010)_2$	Y	$(11)_2$	AB
$(011)_2$	Y	$(00)_2$	AB
$(100)_2$	N		
$(101)_2$	N		
$(110)_2$	Y	$(10)_2$	12
$(111)_2$	N		

Direct-mapped Cache: Zugriff

Address	Address	Data	Hit/Miss
$(22)_{10}$	$(10110)_2$	12	M
$(26)_{10}$	$(11010)_2$	AB	M
$(22)_{10}$	$(10110)_2$	12	H
$(16)_{10}$	$(10000)_2$	FE	M
$(3)_{10}$	$(00011)_2$	AB	M
$(16)_{10}$	$(10000)_2$	FE	H
$(6)_{10}$	$(00110)_2$	00	

Index	Valid	Tag	Data
$(000)_2$	Y	$(10)_2$	FE
$(001)_2$	N		
$(010)_2$	Y	$(11)_2$	AB
$(011)_2$	Y	$(00)_2$	AB
$(100)_2$	N		
$(101)_2$	N		
$(110)_2$	Y	$(10)_2$	12
$(111)_2$	N		

Direct-mapped Cache: Zugriff

Address	Address	Data	Hit/Miss
$(22)_{10}$	$(10110)_2$	12	M
$(26)_{10}$	$(11010)_2$	AB	M
$(22)_{10}$	$(10110)_2$	12	H
$(16)_{10}$	$(10000)_2$	FE	M
$(3)_{10}$	$(00011)_2$	AB	M
$(16)_{10}$	$(10000)_2$	FE	H
$(6)_{10}$	$(00110)_2$	00	M

Index	Valid	Tag	Data
$(000)_2$	Y	$(10)_2$	FE
$(001)_2$	N		
$(010)_2$	Y	$(11)_2$	AB
$(011)_2$	Y	$(00)_2$	AB
$(100)_2$	N		
$(101)_2$	N		
$(110)_2$	Y	$(10)_2$	12
$(111)_2$	N		

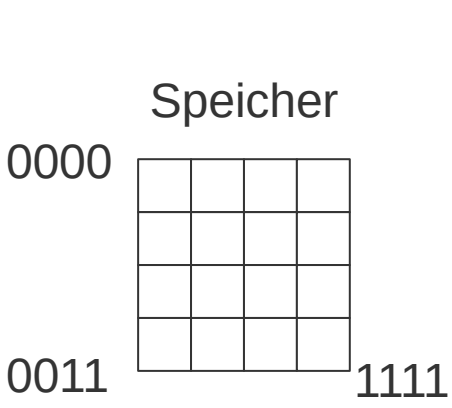
Direct-mapped Cache: Zugriff

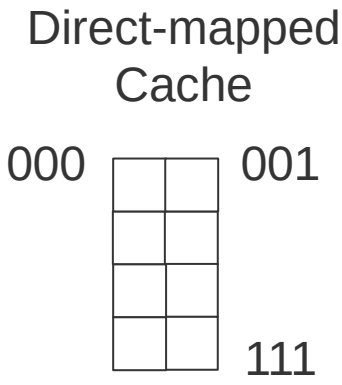
Address	Address	Data	Hit/Miss
$(22)_{10}$	$(10110)_2$	12	M
$(26)_{10}$	$(11010)_2$	AB	M
$(22)_{10}$	$(10110)_2$	12	H
$(16)_{10}$	$(10000)_2$	FE	M
$(3)_{10}$	$(00011)_2$	AB	M
$(16)_{10}$	$(10000)_2$	FE	H
$(6)_{10}$	$(00110)_2$	00	M

Index	Valid	Tag	Data
$(000)_2$	Y	$(10)_2$	FE
$(001)_2$	N		
$(010)_2$	Y	$(11)_2$	AB
$(011)_2$	Y	$(00)_2$	AB
$(100)_2$	N		
$(101)_2$	N		
$(110)_2$	Y	$(00)_2$	00
$(111)_2$	N		

Direct-mapped Cache

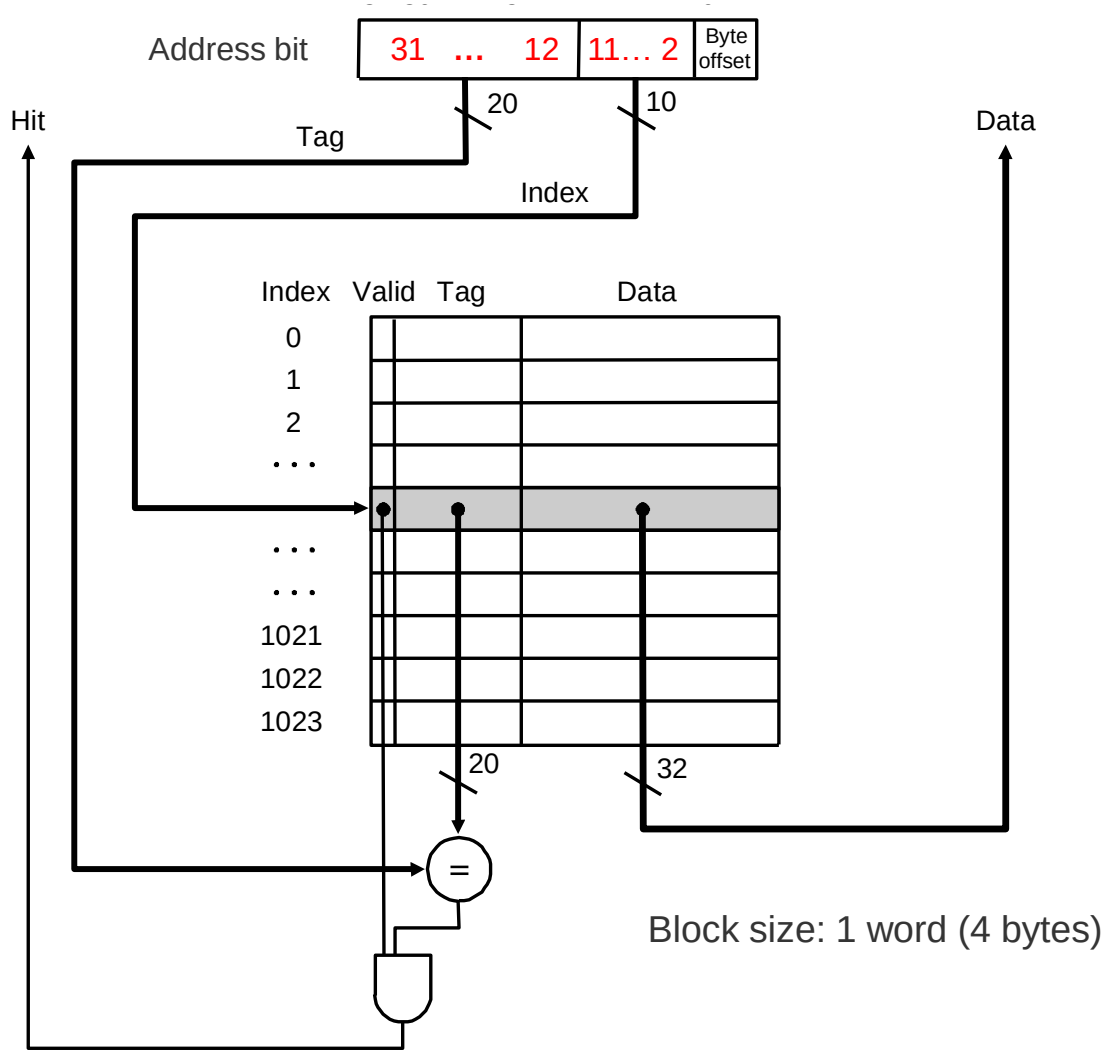
- Ein einzelnes Wort (Byte) zu cachen ist viel Aufwand
  - Besser gleich einen ganzen Block (z.b. 2, 4 oder 8 Worte) auf einmal cachen





- Komponenten einer Speicheradresse
  - Index: Gibt an, an welcher Position der Cache beschrieben wird.
  - Tag: Gibt an, welcher Block des Hauptspeichers im Cache steht.
  - Blockgröße: Gibt an, wie viel Worte auf einmal abgespeichert werden.

Direct-mapped Cache: Zugriff



Block size: 1 word (4 bytes)

Speicherzugriffszeit / Cache-Performance

*t_eff= h * t_cache + (1-h) * t_main*

t_cache … Zugriffszeit auf Cache

t_main … Zugriffszeit auf Hauptspeicher

$$T_{ex} = (CY_{CPU} + CY_{mem}) \cdot T_{cy}$$

T_{ex} … Ausführungszeit eines Programmes
CY_{CPU} … Verarbeitungszeit in der CPU
CY_{mem} … Wartezeit (stall) auf Speicher

$$CY_{mem} = Acc_{mem} \cdot R_{miss} \cdot P_{miss}$$

Acc_{mem} … Speicherzugriffe im Programm
R_{miss} … Miss-Rate
P_{miss} … Miss-Penalty

Gilt für Systeme mit Hit Penalty = 0

Speicher und Performance

- Die Cache-Performance ist ein wesentlicher Faktor für die Gesamt-Performance

- Entscheidend für die Cache-Performance sind:
  - Hit-Time
  - Hit-Rate und
  - Miss-Penalty

- Je höher die Performance der CPU desto größer der Einfluss der Cache-Performance

Hit & Miss beim Lesezugriff

- Hit: unser Plan ist aufgegangen ...

- Miss: Daten vom DRAM in den Cache laden
  - Stall der Pipeline
  - Adresse an das Speichermanagement
  - Daten vom DRAM in den Cache
  - Verarbeitung fortsetzen

Hit & Miss beim Schreibzugriff

- Hit → Write-Through
- Hit → Copy-Back
- Miss → Write-Around
- Miss → Fetch-on-Write

WR-Hit: “Write-Through (WT)”

- aktualisiere den Cache UND
- aktualisiere sofort auch den Hauptspeicher

  - Datenkonsistenz mit Hauptspeicher garantiert (I/O, Multiprozessor)
  - einfach, häufigster Fall (RD) optimiert
  - häufige Zugriffe auf den Hauptspeicher
  - Performance-Verlust

WR-Hit: “Copy-Back (CB)”

- aktualisiere den Cache UND markiere den Block “dirty”
- aktualisiere Hauptspeicher erst später, wenn der Block aus dem Cache entfernt wird

  - keine Datenkonsistenz mit dem Hauptspeicher
  - Write-Hit erfolgt wesentlich schneller
  - Read-Miss wird langsamer (wegen copy-back)
  - seltener Zugriffe auf den Hauptspeicher

  - oft auch als “Write-Back” bezeichnet

WR-Hit: “Write-Buffer”

- für Datenkonsistenz und schnelle Schreiboperation (Vorteile von WT+CB)

- Buffered Write-Through
  - neuer Wert wird in den Cache und zweiten schnellen Zwischenspeicher eingetragen
  - Prozessor kann mit weiterer Abarbeitung fortfahren
  - falls Puffer voll, muss Prozessor warten

WR-Miss: “Write-Around”

- Ignoriere den Cache UND schreibe direkt in den Speicher

- meist in Kombination mit WT

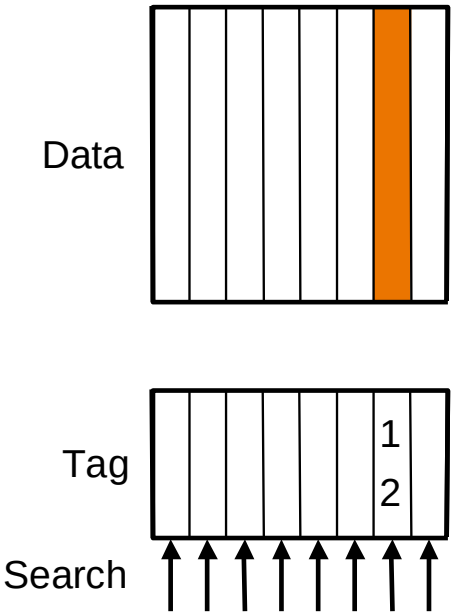
WR-Miss: “Fetch-on-Write”

- Ersetze den aktuellen Inhalt des Caches und aktualisiere Tag
- Falls Blockgröße > 1 Wort, lade die restlichen zum Block gehörigen Daten aus dem Hauptspeicher nach
  - Lesezugriff auf den Speicher und anschließend Write-Hit
  - Write-Hit je nach WR-Hit Strategie
- Am häufigsten verwendete Methode

Assoziativität in Caches

Direct mapped

Fully associative



Direct-mapped Cache

- Für jeden Block gibt es eindeutig nur eine mögliche Position im Cache

- einfache Cache-Verwaltung
- kein Multiplexer im Datenpfad
- mäßige Hit-Rate

Fully Associative Cache

- Jeder Block darf auf jede beliebige Position im Cache gelegt werden
- optimale Hit-Rate
- komplizierte Verwaltung des Caches:
  - Welcher Eintrag wird ersetzt? (globales Optimum)
  - Wo ist ein Block zu finden? (Suche im gesamten Cache)

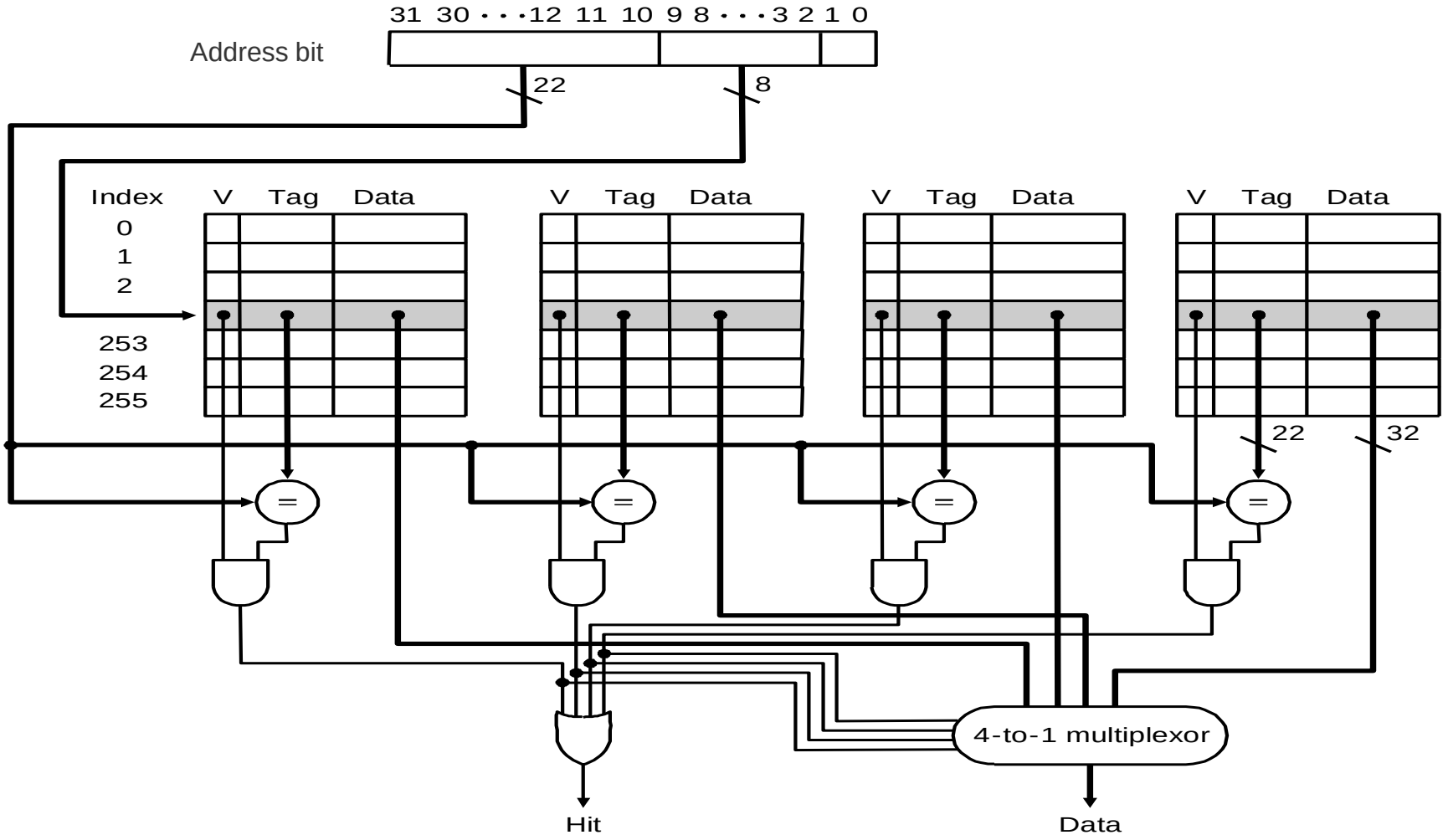
N-way Set-associative Cache

- Für jeden Block gibt es N verschiedene Möglichkeiten der Platzierung im Cache

- Cache-Verwaltung bleibt handhabbar:
  - Suche in beschränktem Bereich (set)
  - Einfachere Ersetzungsregeln (LRU)

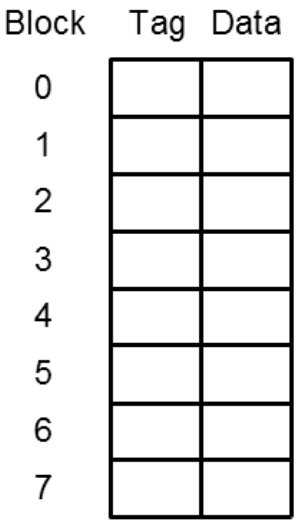
- Vernünftige Hit-Rate

Set-associative Cache: Aufbau



Assoziativität bei 8 Einträgen

One-way set associative
(direct mapped)



Assoziativität bei 8 Einträgen

One-way set associative
(direct mapped)

Block	Tag	Data
0		
1		
2		
3		
4		
5		
6		
7		

Two-way set associative

Set	Tag	Data	Tag	Data
0				
1				
2				
3				

Assoziativität bei 8 Einträgen

One-way set associative
(direct mapped)

Block	Tag	Data
0		
1		
2		
3		
4		
5		
6		
7		

Two-way set associative

Set	Tag	Data	Tag	Data
0				
1				
2				
3				

Four-way set associative

Set	Tag	Data	Tag	Data	Tag	Data	Tag	Data
0								
1								

Assoziativität bei 8 Einträgen

One-way set associative
(direct mapped)

Block	Tag	Data
0		
1		
2		
3		
4		
5		
6		
7		

Two-way set associative

Set	Tag	Data	Tag	Data
0				
1				
2				
3				

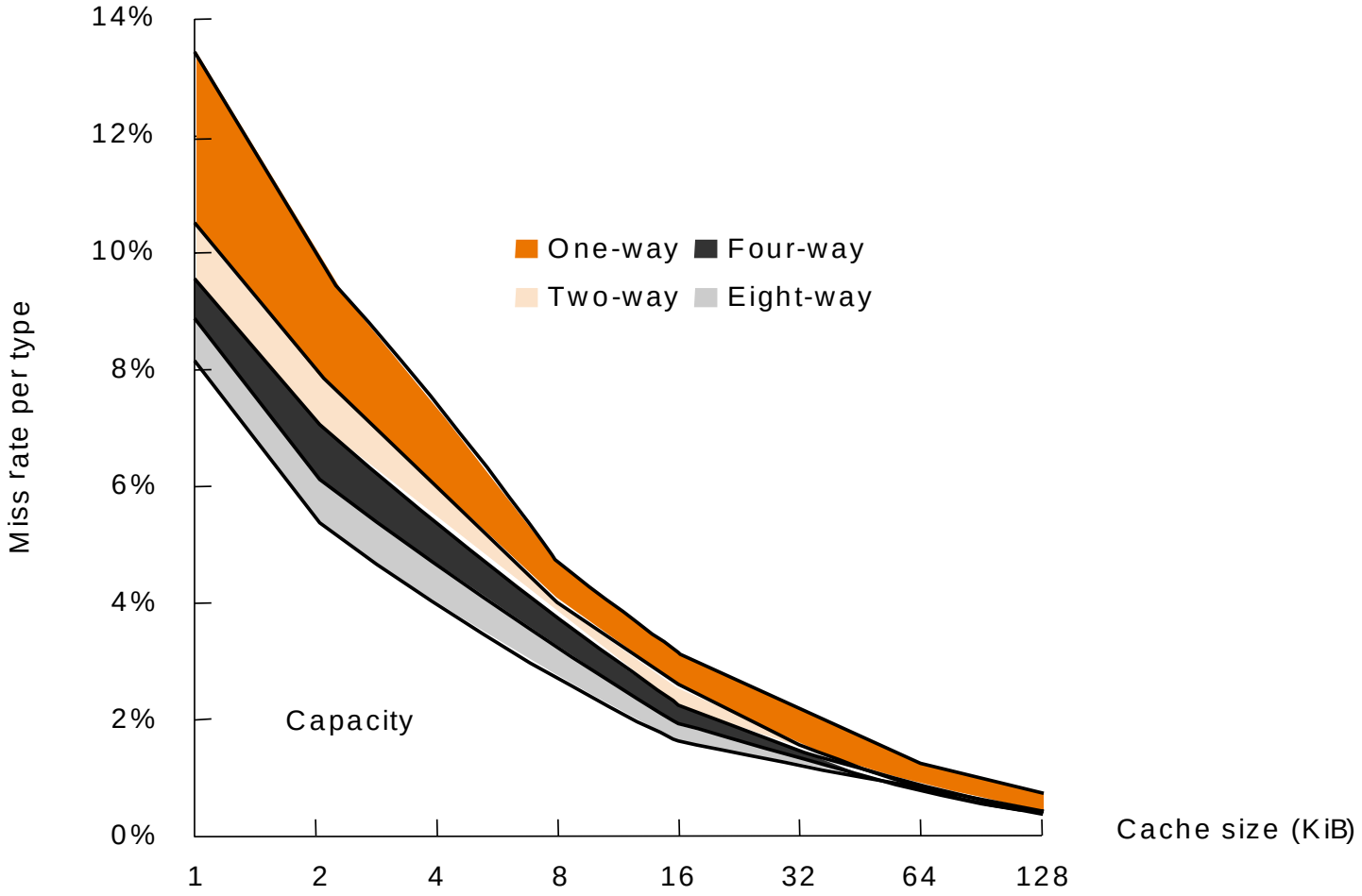
Four-way set associative

Set	Tag	Data	Tag	Data	Tag	Data	Tag	Data
0								
1								

Eight-way set associative (fully associative)

Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data

Assoziativität und Conflict Misses

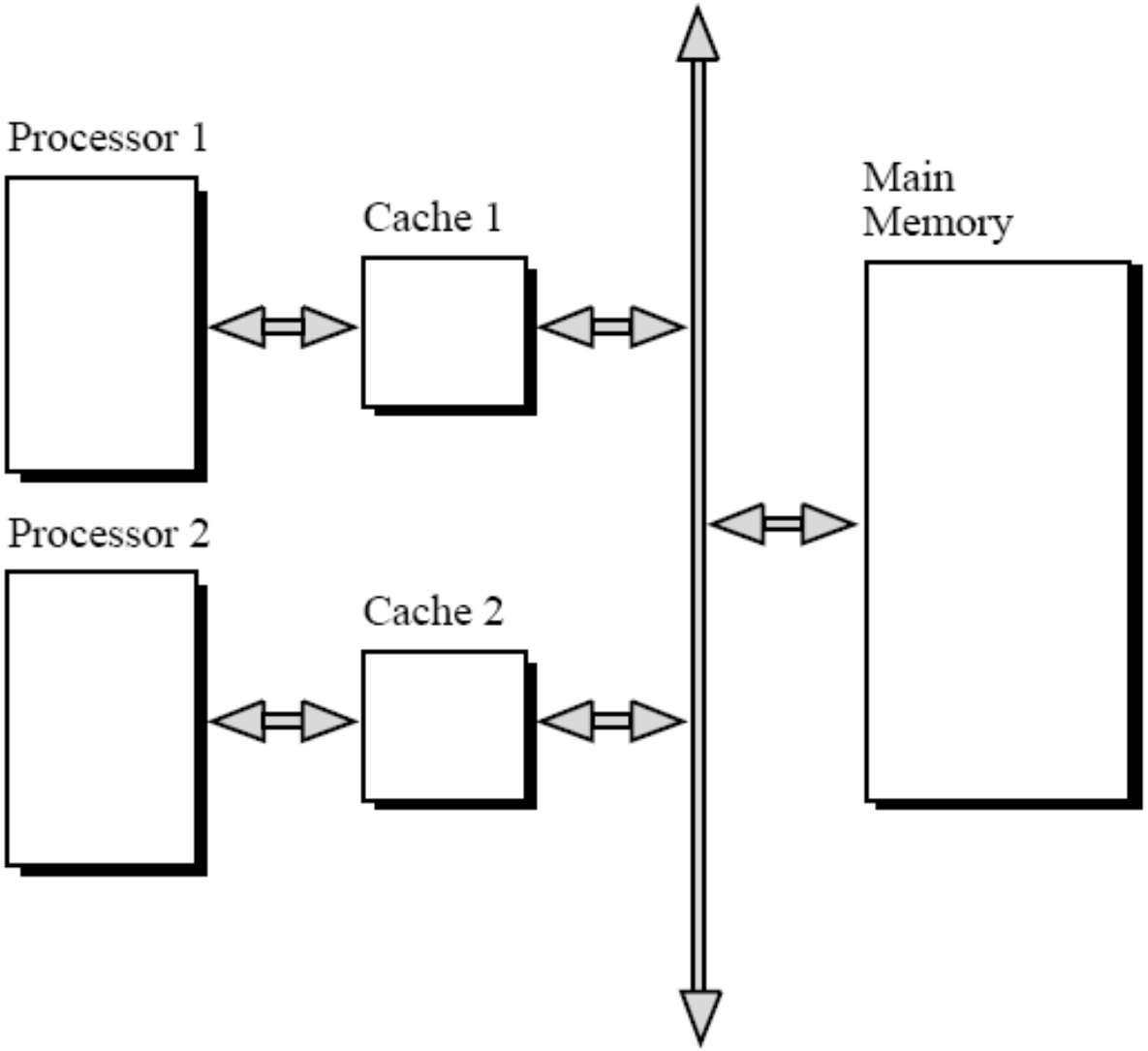


Cache Replacement Strategy

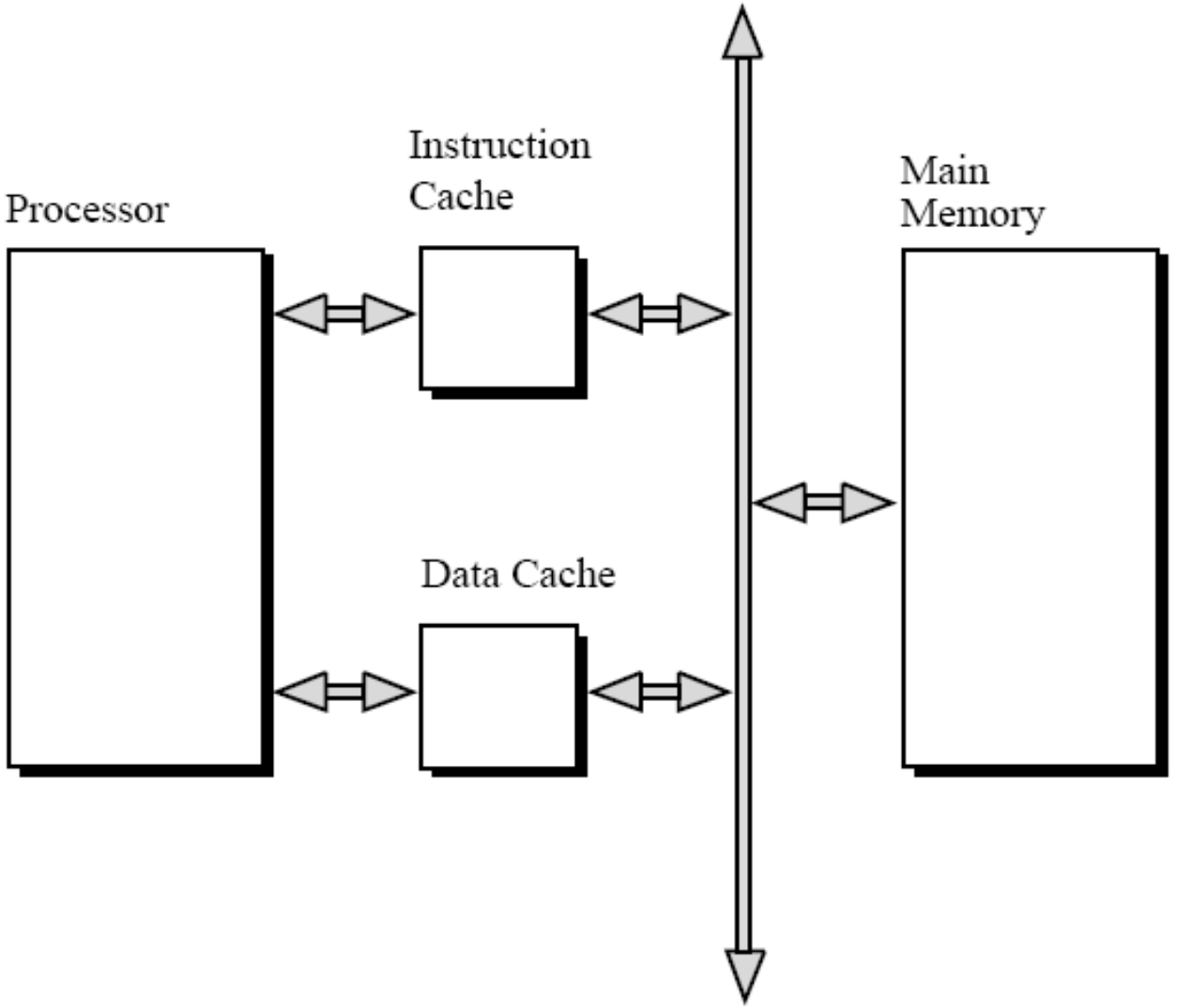
- Direct Mapped Cache: eindeutig über Index
- Set-Associative Cache: Auswahl innerhalb des Sets
- Fully Associative Cache: beliebige Auswahl

- Replacement-Strategien:
  - Least Recently Used (LRU)
  - Least Frequently Used (LFU)
  - “reference bit”
  - Random
  - FIFO (first-in first-out)

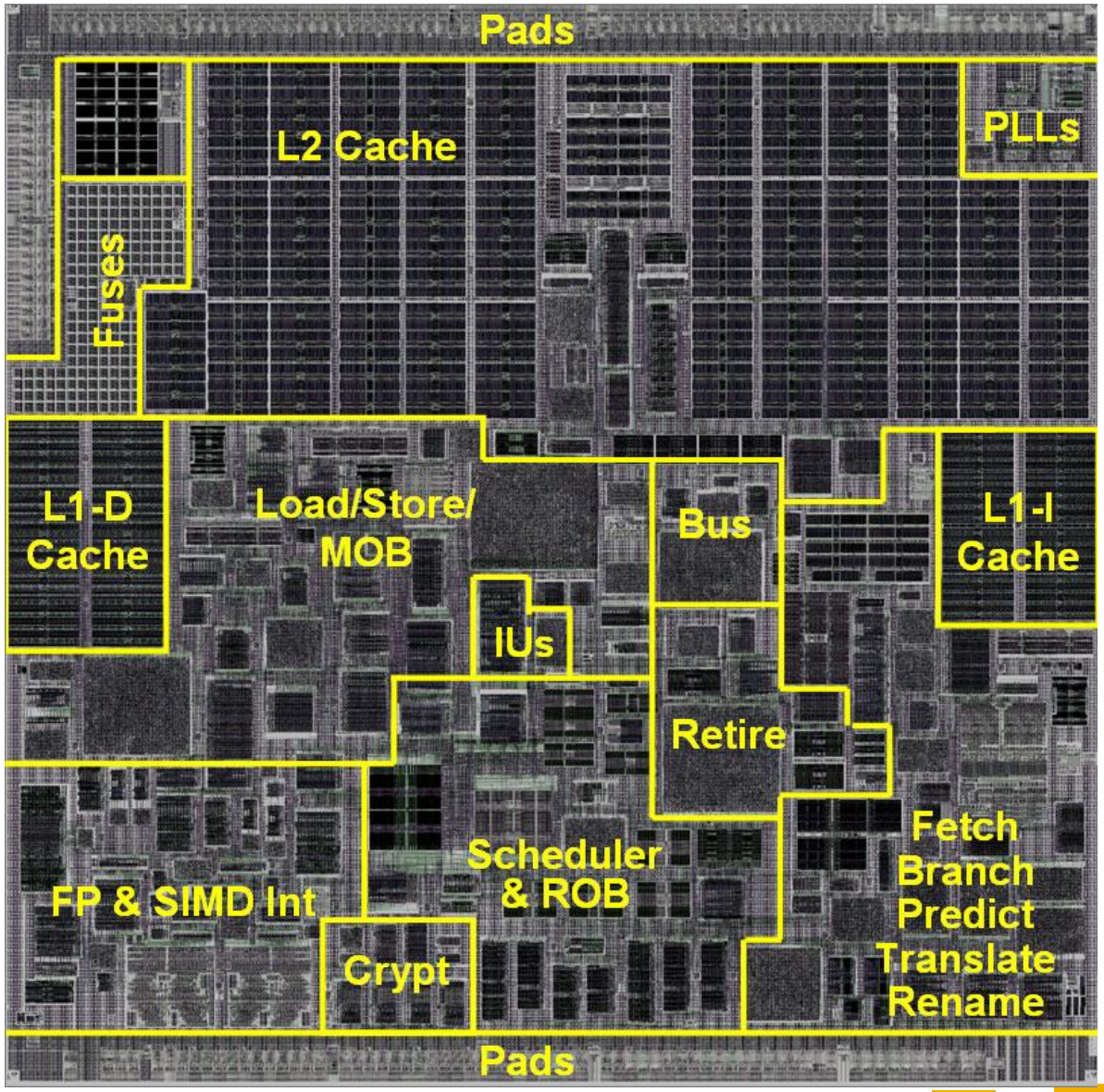
Mehrprozessorsystem mit Caches



„Harvard“ Architektur



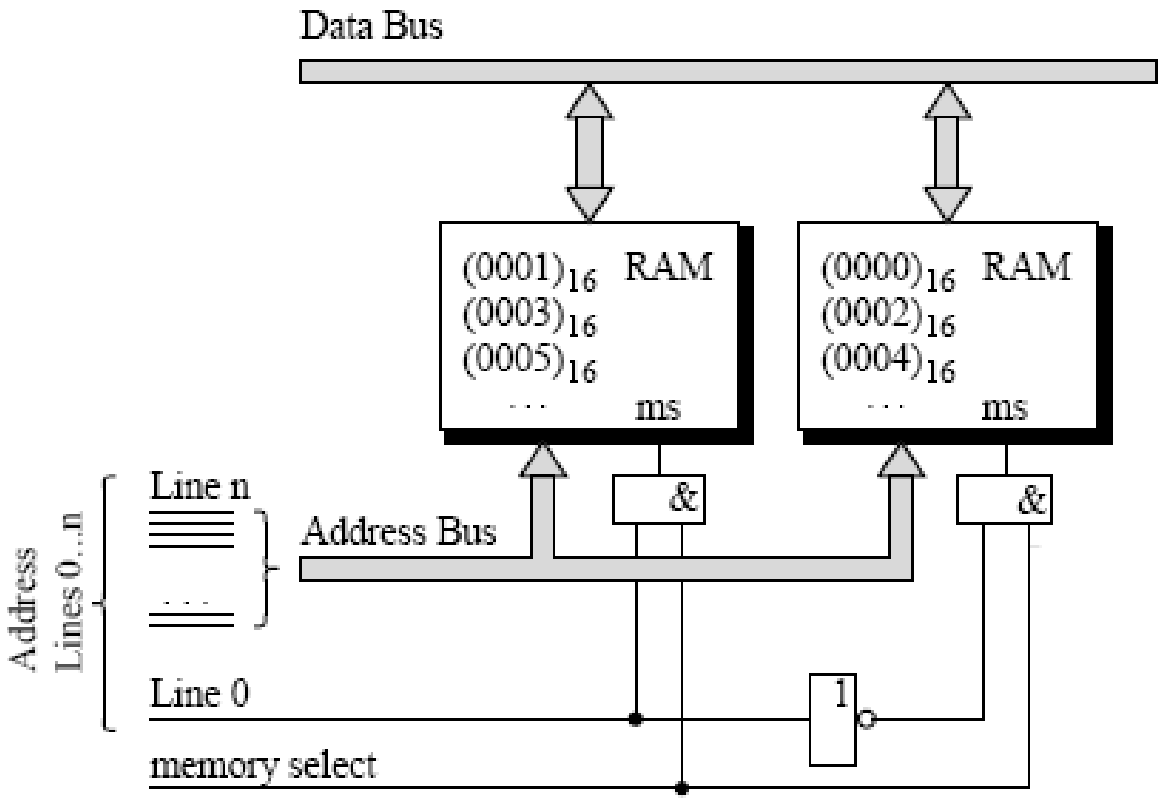
Via Nano 1,8 GHz (z.B. in eeePCs)



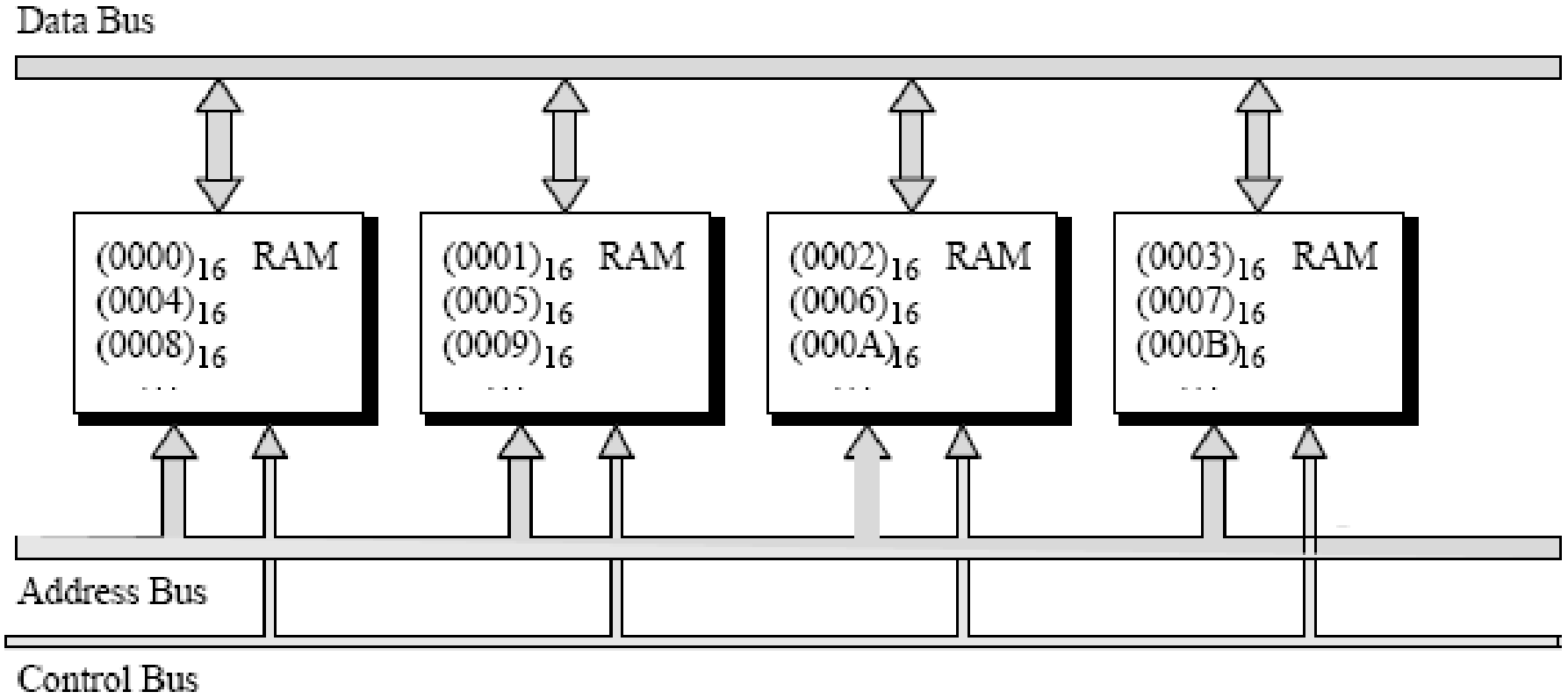
Speicherverwaltung

Interleaved Memory

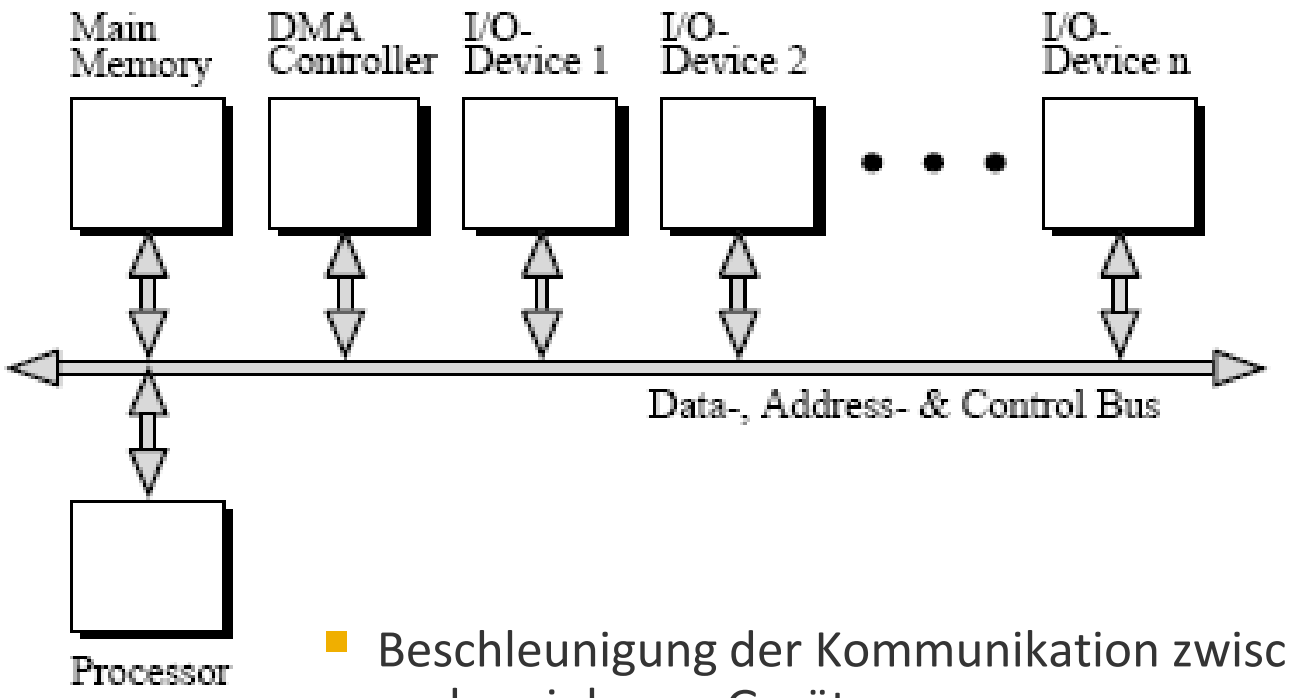
- Meist wird sequenziell auf Speicher zugegriffen
- Aufteilung des Speichers in gleich große Bereiche (Bänke)
- Aufeinanderfolgende Adressen liegen in anderer Bank



4-fach Interleaved Memory



Direct Memory Access



- Beschleunigung der Kommunikation zwischen Prozessor und peripheren Geräten

- Direkter Datenaustausch zwischen peripheren Geräten und Speicher
- Direct Memory Access Controller
- Burst mode vs Cycle stealing