Speicherarchitektur (16)

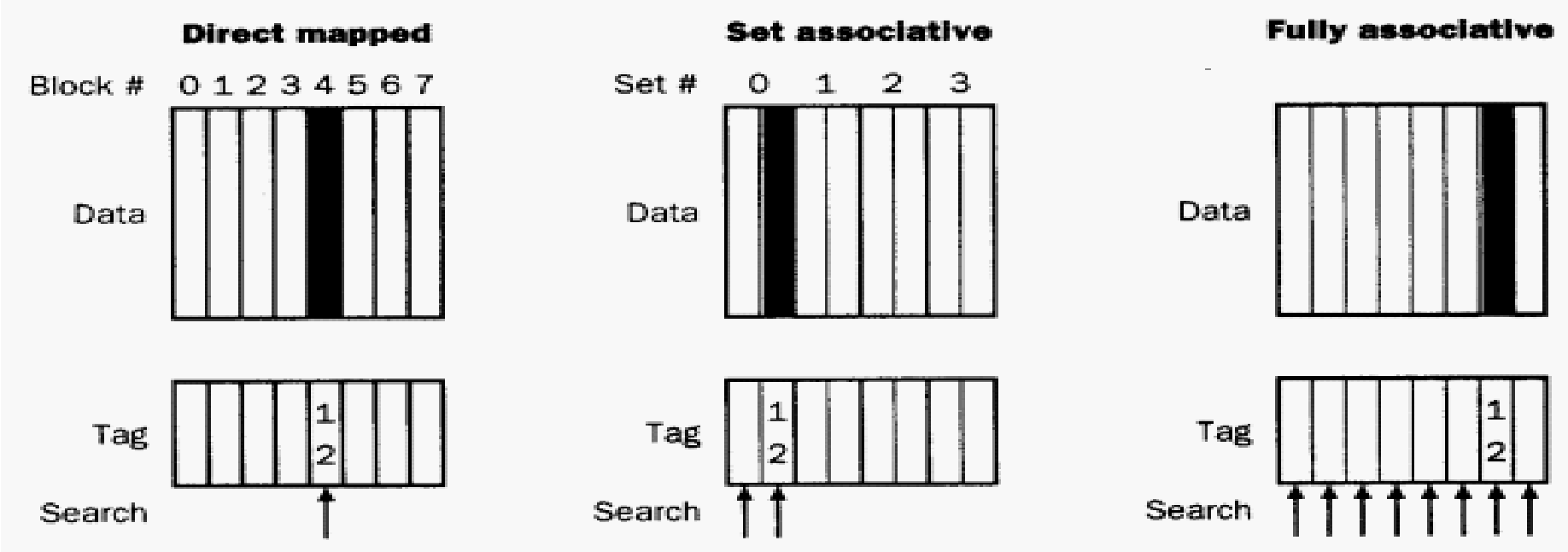
Zuweisungsstrategien für Cacheblocks:

- direct-mapped

- voll-assoziativ

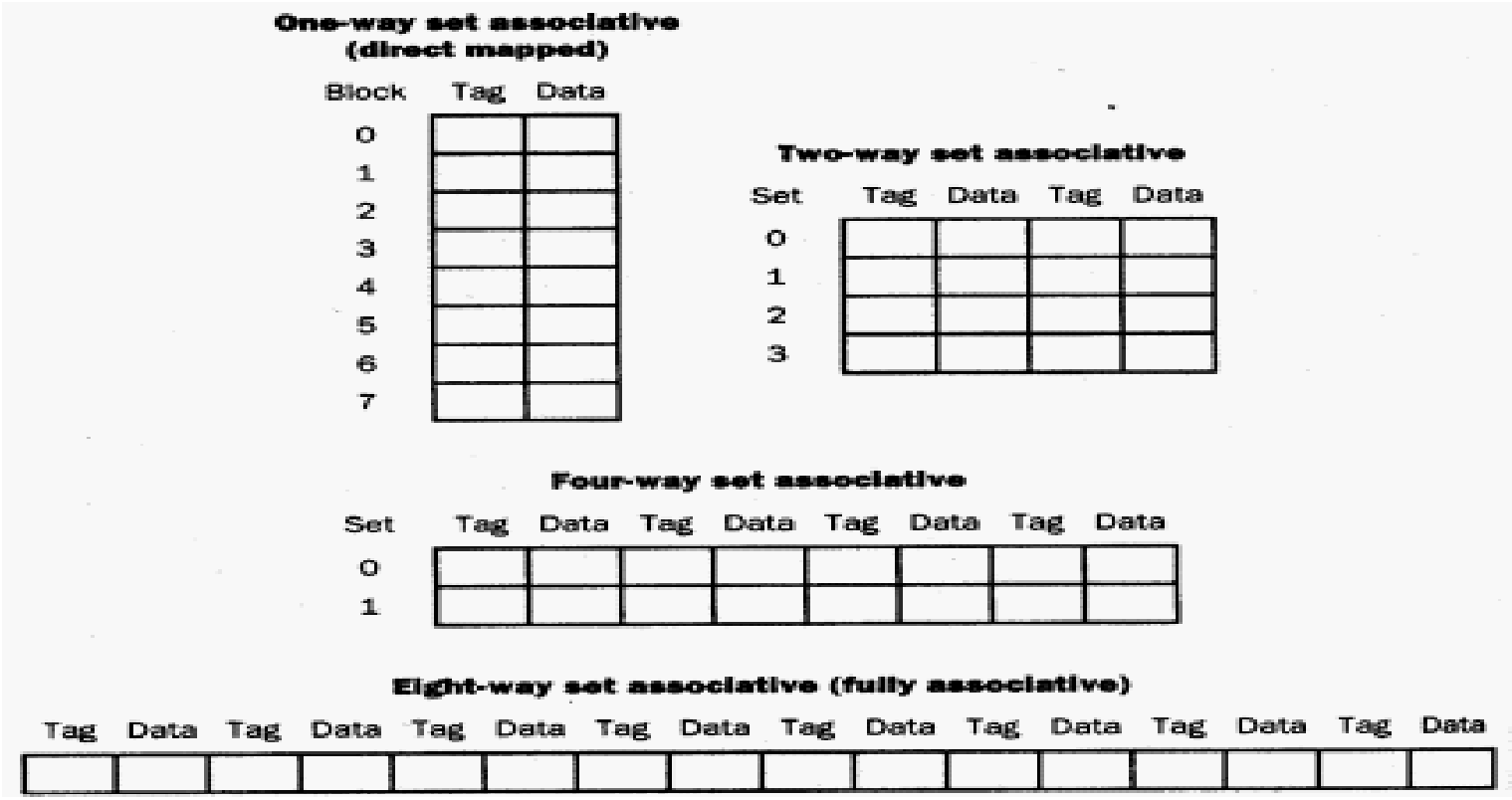
- mengen-assoziativ

Beispiel:



Speicherarchitektur (16)

Konfigurationsmöglichkeiten für einen 8-Block-cache:



Zusammenhang Cacheassoziativität <--> Anzahl der Cache misses

Beispiel:

3 caches mit je 4 Blöcken (direct-mapped, voll- bzw. 2-way set-assoziativ)
Folge von gewünschten Blockadressen: 0,8,0,6,8

Anzahl der cache misses?

Speicherarchitektur (18)

direct-mapped:

Block address	Cache block
0	(0 modulo 4) = 0
6	(6 modulo 4) = 2
8	(8 modulo 4) = 0

Address of memory block accessed	Hit or miss	Contents of cache blocks after reference			
		0	1	2	3
0	miss	Memory[0]			
8	miss	Memory[8]			
0	miss	Memory[0]			
6	miss	Memory[0]		Memory[6]	
8	miss	Memory[8]		Memory[6]	

set-assoziativ:

Block address	Cache set
0	(0 modulo 2) = 0
6	(6 modulo 2) = 0
8	(8 modulo 2) = 0

Address of memory block accessed	Hit or miss	Contents of cache blocks after reference			
		Set 0	Set 0	Set 1	Set 1
0	miss	Memory[0]			
8	miss	Memory[0]	Memory[8]		
0	hit	Memory[0]	Memory[8]		
6	miss	Memory[0]	Memory[6]		
8	miss	Memory[8]	Memory[6]		

Speicherarchitektur (19)

voll-assoziativ:

Address of memory block accessed	Hit or miss	Contents of cache blocks after reference			
		Block 0	Block 1	Block 2	Block 3
0	miss	Memory[0]			
8	miss	Memory[0]	Memory[8]		
0	hit	Memory[0]	Memory[8]		
6	miss	Memory[0]	Memory[8]	Memory[6]	
8	hit	Memory[0]	Memory[8]	Memory[6]	

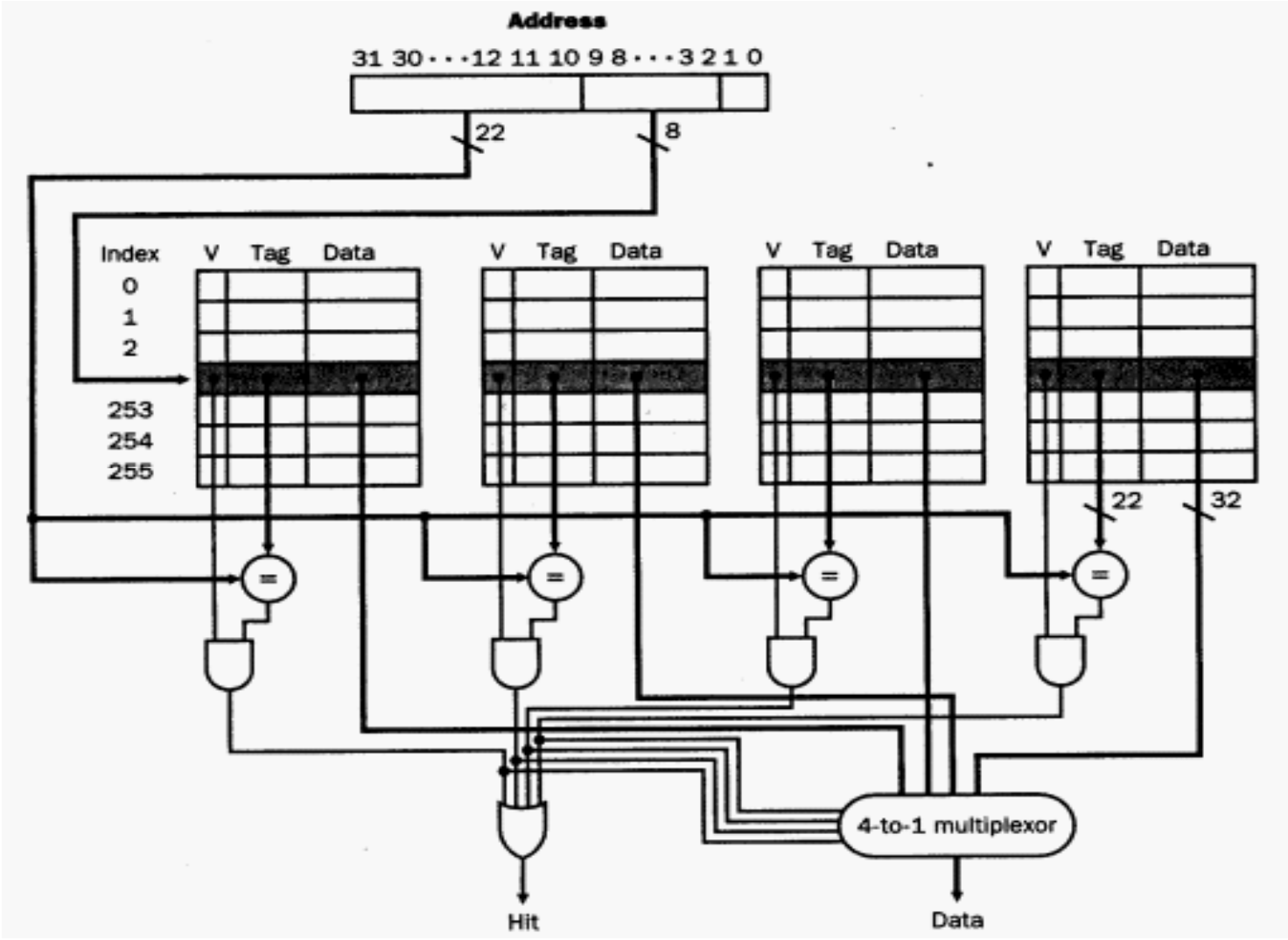
miss rate - Entwicklung am Beispiel von gcc und spice:

Program	Associativity	Instruction miss rate	Data miss rate	Effective combined miss rate
gcc	1	2.0%	1.7%	1.9%
gcc	2	1.6%	1.4%	1.5%
gcc	4	1.6%	1.4%	1.5%
spice	1	0.3%	0.6%	0.4%
spice	2	0.3%	0.6%	0.4%
spice	4	0.3%	0.6%	0.4%

Struktur einer Blockadresse:

Speicherarchitektur (19a)

Implementierung eines 4 Wege set-assoziativen Caches:



Speicherarchitektur (20a)

Berechnung der Cache - Performanz

Programmausführungszeit = (# CPU Ausführungszyklen + # CPU Aussetzerzyklen) x Zykluszeit

(# CPU Aussetzerzyklen = # Aussetzer wegen Lesefehler + # Aussetzer wegen Schreibfehler)

Der Einfachheit halber nehmen wir an, dass die Kosten gemessen in # Zyklen jeweils gleich groß sind.

---> # CPU Aussetzerzyklen = Misses/Programm x Miss - Strafe (= # CPU Aussetzerzyklen pro miss)

---> (absolute) Cache-Performanz steht in umgekehrten Verhältnis zur # CPU Aussetzerzyklen

Schlussfolgerung:

CPU - Leistung erhöhen:

- CPI verringern, d.h. insgesamt # CPU Ausführungszyklen verringern
- Taktrate erhöhen, d.h. Zykluszeit verringern (z.B. durch pipelining)

---> **relative Cache - Performanz verschlechtert sich**

- relative Anteil der # CPU Aussetzerzyklen an der Programmausführungszeit erhöht sich

----> **absolute Cache - Performanz verbessern durch**

- # Misses verringern (durch verbesserten Cacheentwurf wie z.B. Mengenassoziativität, Blockgröße, Verbesserung von Ersetzungsstrategien)
- Miss - Strafe verringern z.B. durch Einführung von *Multilevel (Mehrstufen -) Caches*,Minimierung von Blocktransferzeiten

Speicherarchitektur (20b)

Zusammenfassung:

• *Hauptziel des Cacheentwurfs ist*
  – Cacheperformanz optimieren im wesentlichen durch
    – Ausnutzung von Assoziativität zur Verringerung der miss-Raten
    – Einsatz von Multilevel-Caches zur Verringerung der miss-Zeiten

• *Programmausführungszeit* (bei fixer Taktrate) = *Prozessorausführungszyklen* + *Prozessoraussetzzyklen*
---> Speicherentwurf hat signifikanten Effekt auf Programmlaufzeit
---> je schneller Prozessor, je größer der Effekt von Speicheraussetzzyklen
---> Effekt ist letztendlich abhängig von miss-Rate **und** miss-Zeit

• *Assoziativität erlaubt flexiblere Plazierung von Blöcken im Cache*
aber: je flexibler, desto größer der Aufwand (Block-Suche und Auswahl)

• *Multilevel-Cache statt großer Einzel-Cache zur Verringerung der miss-Zeit*
---> Siliziumbeschränkungen und Erzielung prozessornaher Taktraten begrenzen Größe des Primärcache.
---> Entwurfsparameter sind unterschiedlich je nach Cacheebene

Speicherarchitektur (21)

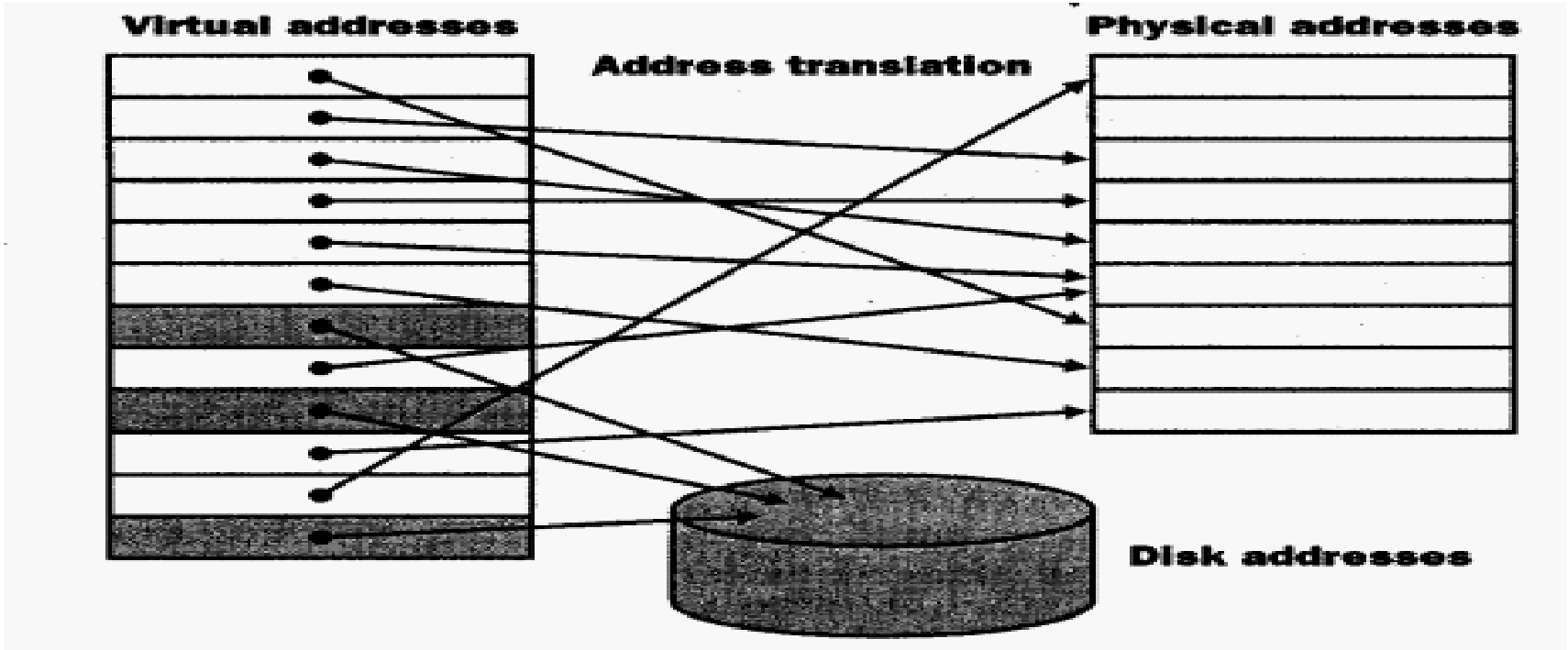
V i r t u e l l e r S p e i c h e r

Motivation für virtuellen Speicher:

• erlaubt es dem Programmierer, die Tatsache des nur begrenzt vorhandenen Hauptspeichers außer Acht zu lassen

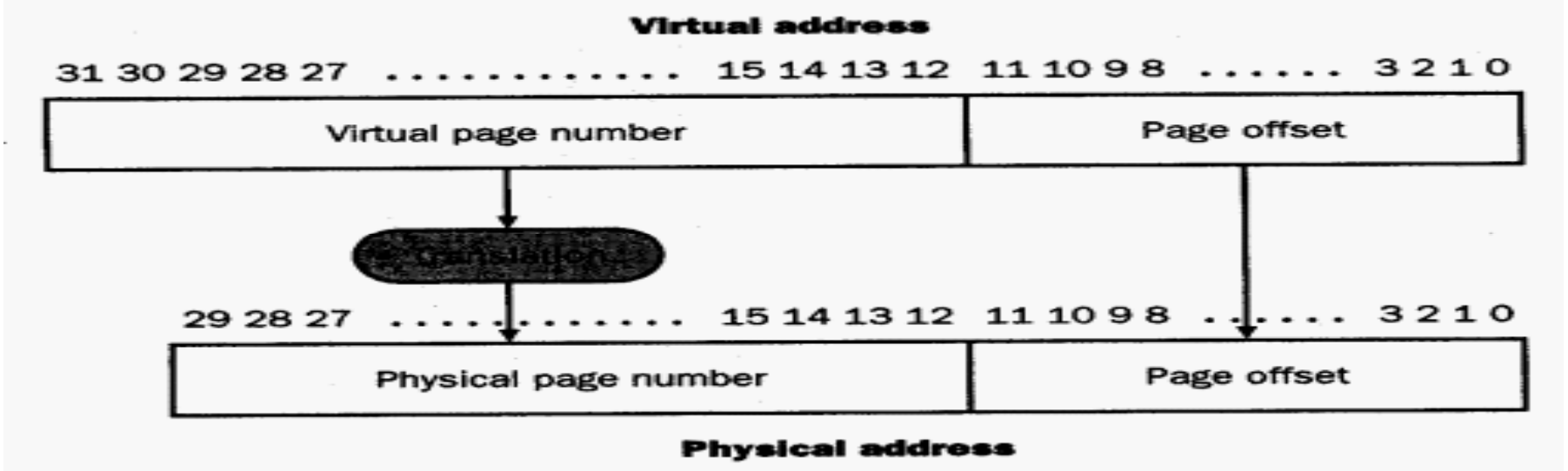
• erlaubt effizientes und sicheres Sharing von Speicher zwischen mehreren Programmen

Adressübersetzung:



Speicherarchitektur (22)

Struktur der Adressen:



Eckpunkte für den Entwurf von virtuellen Speichersystemen:

• Seiten müssen groß genug sein, um die hohe Zugriffszeit zu amortisieren durch Ausnutzung von Lokalität

• Reduzierung der Seitenfehlerrate ist von höchster Priorität

• ausgeklügelte Seitenersetzungsstrategien

• Es werden grundsätzlich write-back - Verfahren eingesetzt