Computerarchitectuur

App. B. Review of Memory Hierarchy

Kristian Rietveld

http://ca.liacs.nl/

Universiteit Leiden
The Netherlands

Caching

- De “memory wall” is een zeer groot probleem dat vraagt om oplossingen.
- De gebruikelijke oplossing is het creëren van een hiërarchie van (verschillende) geheugens.
  - We gebruiken het principe van caching: het tijdelijk opslaan van data in een sneller geheugen.
  - We verbergen in feite de latency van lagere niveau's. Frans: *caché* (verbergen).
- Caching komt op veel plekken voor: buffer cache (OS), DNS cache, web browser cache, …

Memory hierarchies

Level	1	2	3	4
Name	Registers	Cache	Main memory	Disk storage
Typical size	<1 KB	32 KB–8 MB	<512 GB	>1 TB
Implementation technology	Custom memory with multiple ports, CMOS	On-chip CMOS SRAM	CMOS DRAM	Magnetic disk
Access time (ns)	0.15–0.30	0.5–15	30–200	5,000,000
Bandwidth (MB/sec)	100,000–1,000,000	10,000–40,000	5000–20,000	50–500
Managed by	Compiler	Hardware	Operating system	Operating system/ operator
Backed by	Cache	Main memory	Disk	Other disks and DVD

Taken from Computer Architecture: A Quantitative Approach, fifth edition. Fig.. B.1.

Table 2.2 Example Time Scale of System Latencies

Event	Latency	Scaled
1 CPU cycle	0.3 ns	1 s
Level 1 cache access	0.9 ns	3 s
Level 2 cache access	2.8 ns	9 s
Level 3 cache access	12.9 ns	43 s
Main memory access (DRAM, from CPU)	120 ns	6 min
Solid-state disk I/O (flash memory)	50–150 µs	2–6 days
Rotational disk I/O	1–10 ms	1–12 months
Internet: San Francisco to New York	40 ms	4 years
Internet: San Francisco to United Kingdom	81 ms	8 years
Internet: San Francisco to Australia	183 ms	19 years
TCP packet retransmit	1–3 s	105–317 years
OS virtualization system reboot	4 s	423 years
SCSI command time-out	30 s	3 millennia
Hardware (HW) virtualization system reboot	40 s	4 millennia
Physical system reboot	5 m	32 millennia

Source: *Systems Performance: Enterprise and the Cloud*, Brendan Gregg.

Cache terminologie

- ***CPU cache***: het eerste niveau van de memory hiërarchie dat wordt aangetroffen zodra een adres de processor verlaat.
- ***Cache hit***: de data voor dit adres wordt in de gevonden.
- ***Cache miss***: de data wordt niet gevonden.
- In geval van een cache miss wordt een blok, waarin zich de benodigde data bevindt, uit het geheugen geladen.

Cache terminologie (2)

- CPU caches zijn georganiseerd in blokken:
  - Blocks, line runs, cache lines.
  - Vaste grootte: vaak 64 bytes.
- Caches zijn ontworpen naar het *principe van locality*:
  - ***Temporal locality***: data blijft in de cache aanwezig, zodat een volgend gebruik van dezelfde data veel sneller kan worden afgehandeld.
  - ***Spatial locality*****:** omdat data uit het geheugen wordt opgehaald in de vorm van cache lines, wordt naastgelegen data alvast opgehaald. Eventueel gebruik van naastgelegen data kan dus ook veel sneller worden afgehandeld.

Cache performance

- Om te kunnen redeneren over cache performance, breiden we onze CPU performance equations uit met memory stall cycles.
  - Een memory stall cycle is een clock cycle waarin de processor aan het wachten is op geheugentoegang.

$$CPU\ execution\ time = (CPU\ clock\ cycles + Memory\ stall\ cycles) \times Clock\ cycle\ time$$

- Het aantal memory stall cycles hangt af van het aantal cache misses en de kosten van een miss (*miss penalty*).

$$Memory\ stall\ cycles = Number\ of\ misses \times Miss\ penalty = IC \times \frac{Misses}{Instruction} \times Miss\ penalty$$

$$= IC \times \frac{Memory\ accesses}{Instruction} \times Miss\ rate \times Miss\ penalty$$

- Miss penalty is hier een gemiddelde, houd in de gaten dat dit een vereenvoudiging is van de werkelijkheid.
  - Eigenlijk zou je miss rate en miss penalty apart moeten bepalen voor reads en writes.

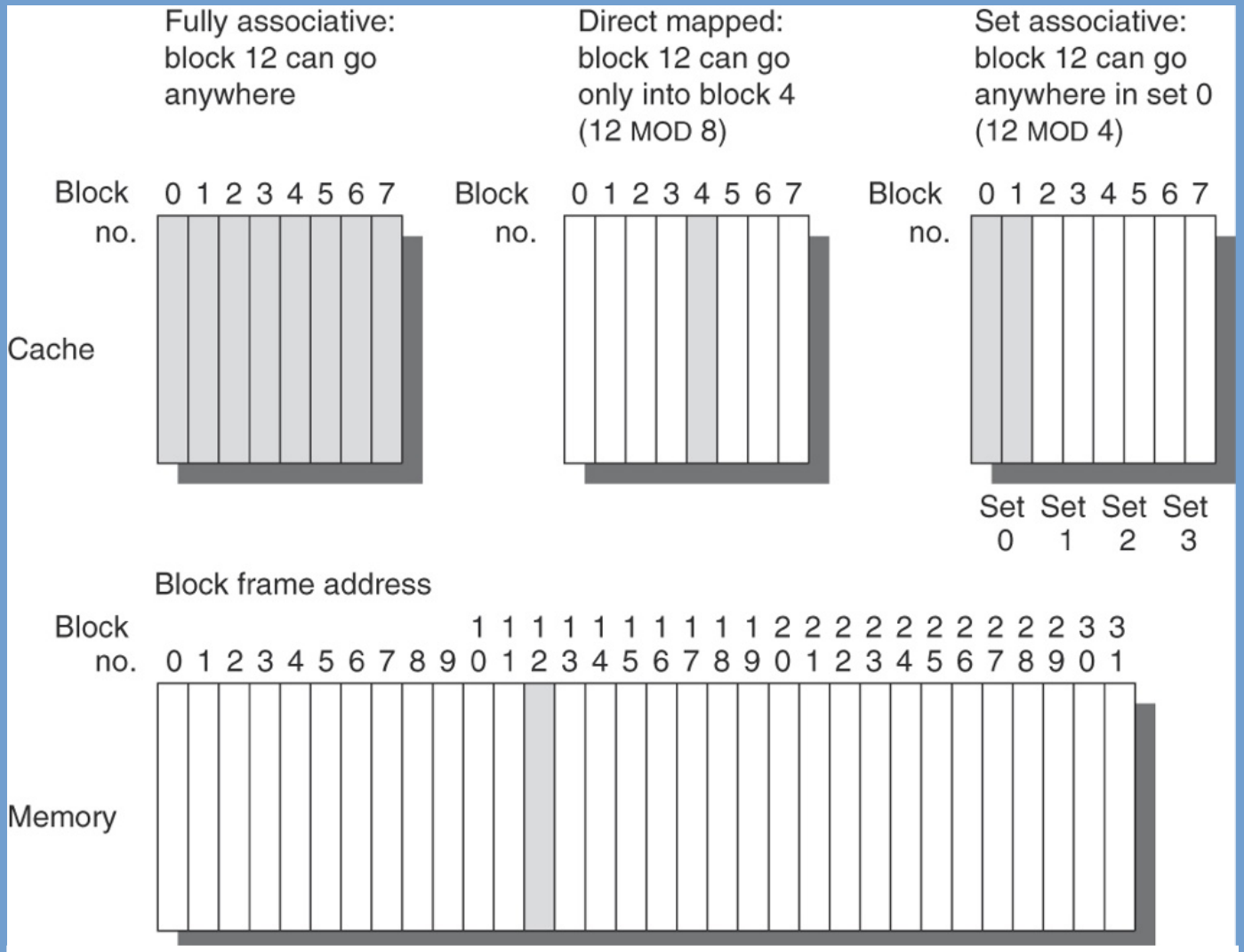
Cache performance (2)

- Miss rate is de fractie van het aantal cache reads die resulteren in een cache miss (#miss / #accesses).
- Hoe kunnen we de miss rate bepalen:
  - Gebruik een “cache simulator” om een address trace van een programma door te rekenen.
  - Gebruik de hardware performance counters die te vinden zijn op moderne chips.

Cache architecture

- Er bestaan verschillende cache organisaties, die verschillende beperkingen hebben waar cache blokken kunnen worden opgeslagen.
  - ***Direct mapped***: een blok kan maar op 1 bepaalde plaats in de cache worden opgeslagen. (Blok addr) MOD (#blokken in cache).
  - ***Fully associative***: een blok kan op alle plaatsen in de cache worden opgeslagen.
  - ***Set associative***: een blok kan op een vaste set van plaatsen worden opgeslagen.
    - Een "*set*" is een groep blokken in de cache.
    - Eerst wordt een set voor een blok gekozen, daarna kan een blok overal in die set worden opgeslagen.
    - (Blok addr) MOD (#sets in de cache).
    - '*n*' blokken in een set -> de cache is *n-way set associative*.

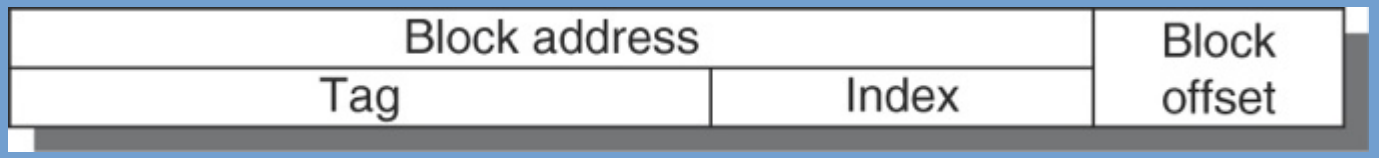
Cache architecture (2)



Taken from Computer Architecture: A Quantitative Approach, fifth edition.Fig. B.2.

Cache architecture (3)

- Als de CPU een geheugenoperatie doet op een gegeven adres, hoe gaan we na of dit wel of niet in de CPU cache te vinden is?
- Een geheugenadres valt als volgt uiteen:



- We moeten nagaan of “block address” in de cache staat.
  - Block offset is de byte-offset in het blok en doet er niet toe.
- Elk cache frame heeft een “tag” met het “block address” dat daar is opgeslagen.
- Index kiest de set. Voor alle frames binnen die set moet de “tag” worden vergeleken.
  - Deze vergelijking in parallel, moet zeer snel zijn!

Cache architecture (4)

- In geval cache miss wordt er data in de cache geladen. Welk bestaande blok binnen een set wordt vervangen met deze data?
- Verschillende strategieën:
  - ***Random***: random keuze.
  - ***LRU***: Least recently used. (Denk aan temporal locality).
  - ***FIFO***: First in, first out. Benadering van LRU door steeds het oudste blok te kiezen.
- Omdat LRU vaak lastig is te implementeren, wordt vaak een benadering gebruikt.

Cache architecture (5)

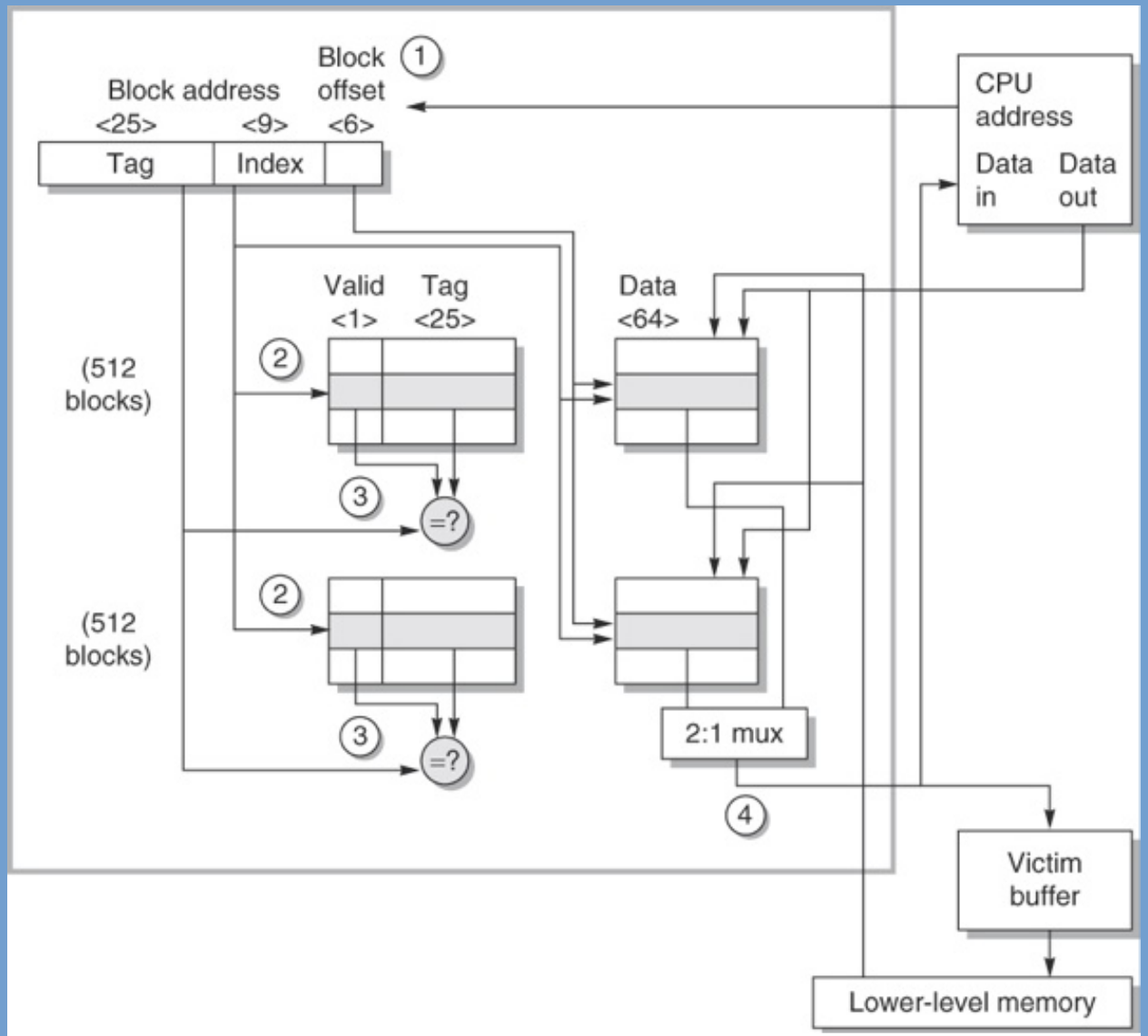
- Er kan op verschillende manieren met writes (stores) worden omgegaan:
  - ***Write-through***: schrijf naar blok in de cache **en** blok in lower-level geheugen.
    - Eenvoudiger om te implementeren; low-level geheugen bevat altijd de meest recente data.
  - ***Write-back***: schrijf alleen naar blok in de cache. Cache blok wordt pas naar geheugen geschreven in geval van replacement.
    - *Dirty bit* wordt gebruikt om bij te houden of een cache frame nog moet worden weggeschreven.
    - Bij meerdere writes naar dit blok, wordt er maar eenmalig naar lower-level geheugen geschreven. (Veel minder geheugenverkeer).

Cache architecture (6)

- ***Write miss***: blok waarin data wordt weggeschreven staat niet in de cache.
- Twee manieren om hiermee om te gaan:
  - ***Write allocate***: haal blok naar de cache (zelfde als read miss) en voer daarna de schrijfactie uit.
  - ***No-write allocate***: cache wordt niet aangepast, schrijf blok direct naar lower-level geheugen.

- Write-back caches vaak write-allocate (in de hoop dat volgende writes hiermee worden opgevangen).
- Write-through caches vaak no-write allocate (volgende writes moeten sowieso worden doorgezet naar geheugen).

Voorbeeld: AMD Opteron Data cache



Taken from Computer Architecture: A Quantitative Approach, fifth edition.Fig. B.5.

- 64 KB cache
- 64-byte cache block size
- 2-way set associative (2 blokken in een set)
- 512 sets

Klassieke architecturen

- Von Neumann architectuur
  - Stored-program computer.
  - Data en programma in hetzelfde geheugen.
  - Ophalen instructie en data kan niet tegelijkertijd (Von Neumann bottleneck).

- Harvard architectuur
  - Aparte geheugens en bussen voor data en instructies.

Klassieke architecturen (2)

- Maar gescheiden data en instruction cache?
- “Modified Harvard architecture”
  - Wanneer CPU met de caches werkt, heeft het in feite aparte geheugens voor data en instructies (Harvard).
  - Maar de gescheiden caches worden vanuit 1 geheugen gevoed (von Neumann).

Cache performance equation

- Je zou kunnen zeggen dat miss rate een goede maat is voor het vergelijken van memory hierarchy performance, aangezien dit hardware-onafhankelijk is.
- Echter vertelt dit niet het hele verhaal: een systeem met een hogere miss rate kan door bepaalde keuzes in de hardware-implementatie toch een betere average memory access time hebben.

$$Average\ memory\ access\ time = Hit\ time + Miss\ rate \times Miss\ penalty$$

- Bijv. als het systeem met de lagere miss rate een hogere hit time heeft.

Basic cache optimizations

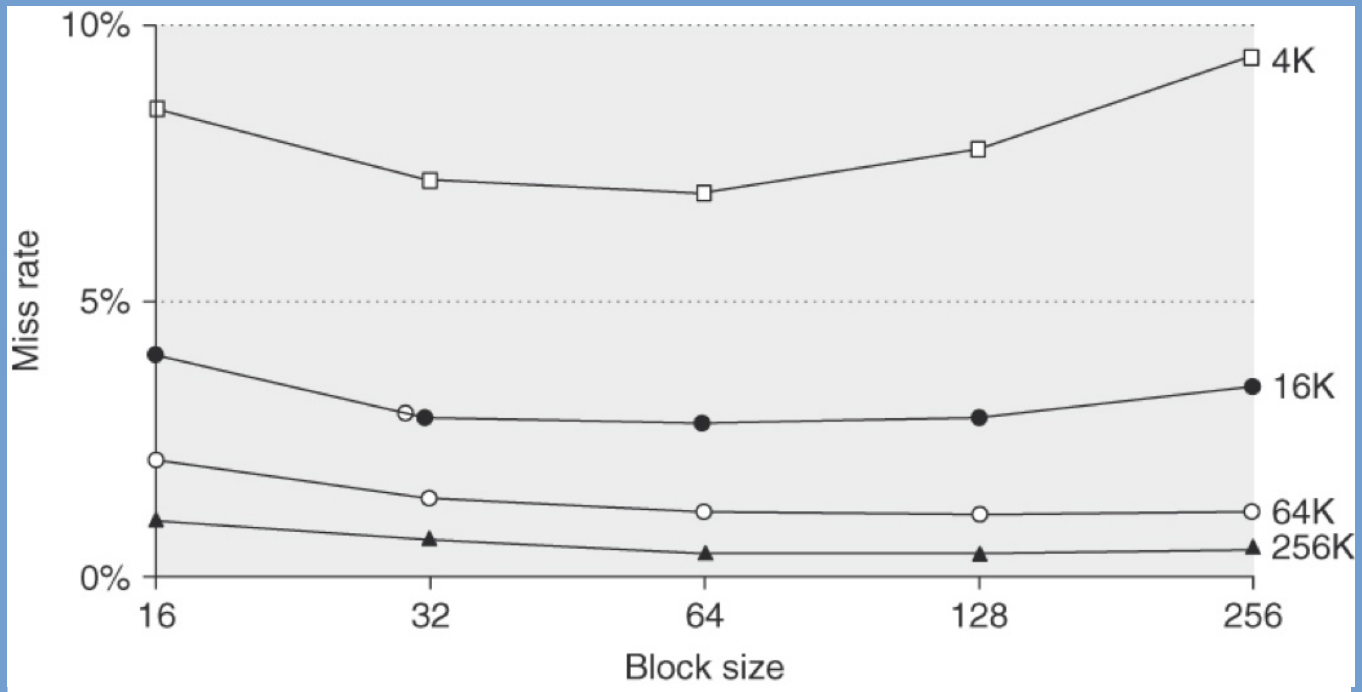
- We gebruiken nu de average memory access time om een aantal basale cache optimalisaties te bestuderen.
- Uit die formule volgen direct drie categorieën van optimalisaties:
  - Miss rate verkleinen (grotere blokken, grotere caches, hogere associativiteit).
  - Miss penalty verkleinen (multi-level cache, reads voorrang geven over writes).
  - Hit time verkleinen (address translation vermijden bij cache indexing).

Miss rate verkleinen

- Waardoor kan een miss optreden?
  - ***Compulsory:*** bij eerste toegang tot een blok staat deze nooit in de cache (cold cache).
  - ***Capacity***: de cache is te klein om alle benodigde blokken in de cache te houden.
  - ***Conflict:*** deze worden veroorzaakt door de cache configuratie. Voorbeeld: teveel blokken vallen binnen dezelfde set. "*Collision misses*".

- Aantal compulsory misses is over het algemeen klein.
- Het meeste valt te halen bij capacity en conflict misses.

Cache blokken vergroten



Taken from Computer Architecture: A Quantitative Approach, fifth edition. Fig. B.10.

- Grotere blokken: maak gebruik van spatial locality.
- Door het kleinere aantal blokken neemt het aantal conflict misses toe en dus de miss rate.
- De miss penalty neemt toe bij grotere blokken, omdat het langer duurt de blokken uit geheugen te halen.
- Blokgrootte hangt ook af van latency en bandwidth lower-level memory.

Grotere caches

- Verlaagt de miss rate door het aantal capacity misses terug te dringen.
- Grote problemen hiermee zijn de langere hit time en hogere kosten en energiegebruik.
- We zien dit voornamelijk terug bij off-chip caches.

Associativiteit verhogen

- Een hogere associativiteit verlaagt de miss rate.
- Vuistregels:
  - 1. 8-way set associative is voor praktische toepassingen even effectief als fully associative (voor cache grootte 4 – 512 KB).
  - 2. Direct-mapped cache met grootte N heeft ongeveer dezelfde miss rate als een 2-way set associative cache met grootte N/2 ("*2:1 cache rule of thumb*").
- Het verhogen van de associativiteit heeft vaak tot gevolg dat de hit time wordt verhoogd.

Multi-level caches

- Combinatie van caches zowel sneller als groter te maken:
  - Houd de first-level cache klein en simpel (kleine hit time).
    - Belangrijk om hoge kloksnelheden te blijven behalen.
  - Second-level cache:
    - Grotere hit-time, maar ook grotere capaciteit.
    - Bij veel hits voorkomen we geheugentoegang, waardoor de miss penalty afneemt.

- Performance analyse wordt echter een stuk gecompliceerder …

Multi-level caches (2)

- Wat voor keuzes maken we voor een second-level cache?
  - Grootte: in ieder geval groter dan first-level cache.
  - Blokgrootte: soms heeft een second-level cache grotere blokken.
  - Direct mapped of set associative?
    - Set associative heeft potentie om miss rate en miss penalty te verbeteren.
  - Multi-level inclusion vs. multi-level exclusion.
    - Inclusion vergemakkelijkt consistency/coherence.
    - Exclusion: gaat verspilling van ruimte in L2 tegen.

- Observatie: veel minder hits vergeleken met first-level cache, dus focus verschuift naar het terugdringen van misses.
  - Grotere caches, grotere blocks, grotere associativiteit.

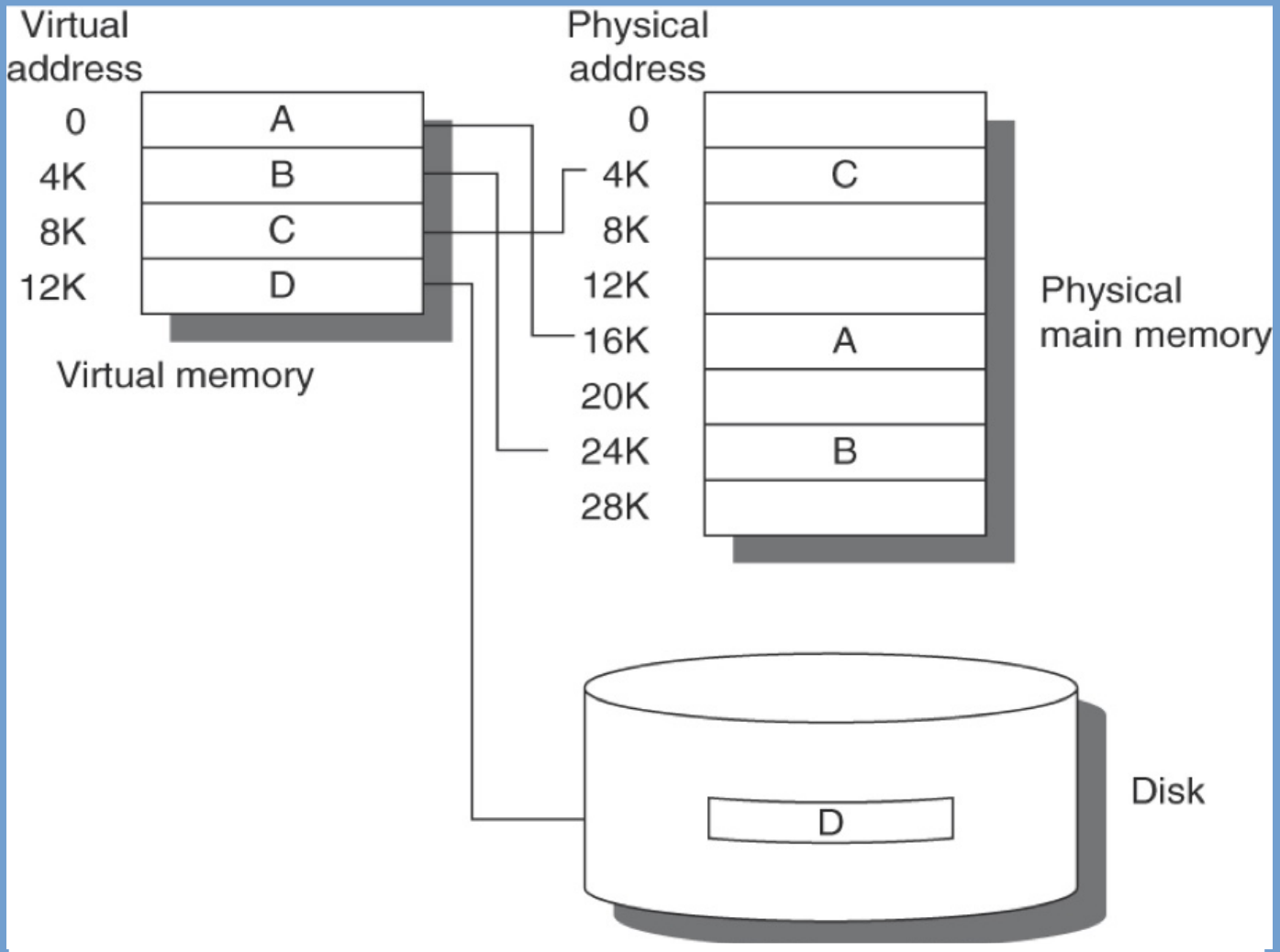
Geef read misses voorrang

- In geval write-through cache worden write buffers gebruikt om de performance te verbeteren.
- Wat nu wanneer de data nodig voor een read miss zich nog in de write buffer bevindt?
  - Wacht tot de write buffer leeg is.
  - Bekijk de inhoud van de write buffer, als er geen conflicten zijn handel dan eerst de read miss af.
- Dus de read miss krijgt voorrang boven het legen van de write buffer.

Intermezzo: virtual memory

- Moderne systemen implementeren “virtual memory”.
  - Elk 'proces' krijgt zijn eigen, virtuele adresruimte, om zelf in te richten.
  - Er kan meer virtueel geheugen worden gebruikt, dan er fysiek RAM geheugen aanwezig is.
  - De verschillende virtuele adresruimten worden van elkaar afgeschermd.

Virtual memory (2)

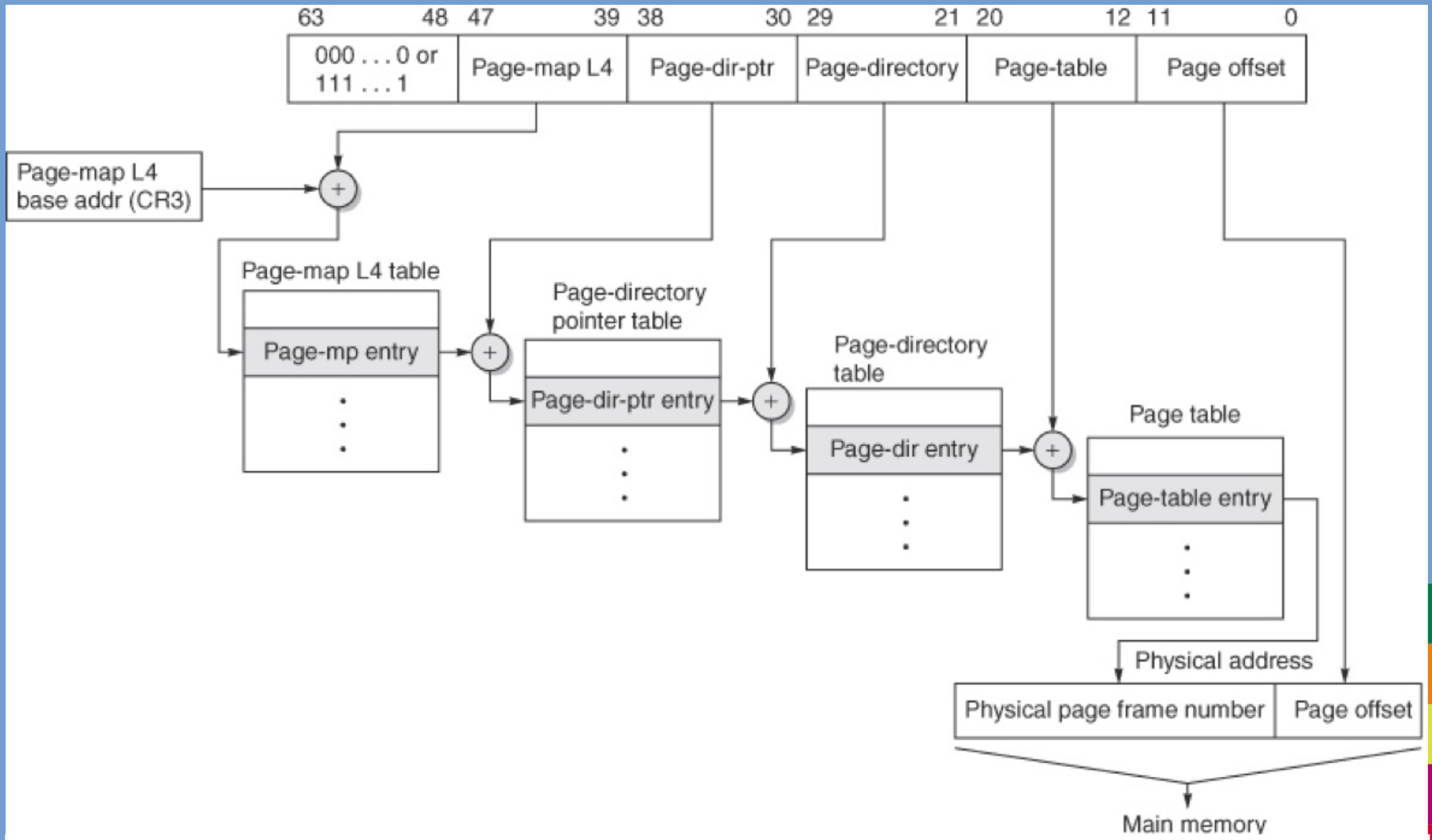


Taken from Computer Architecture: A Quantitative Approach, fifth edition. Fig. B.19.

Virtual memory (3)

- De afbeelding van virtuele op fysieke adressen moet ergens worden opgeslagen.
- Dit gebeurt in zogenaamde “page tables” en deze staan in het RAM geheugen.
  - De page tables moeten worden bijgehouden door het Operating System.
  - Het meest gebruikt zijn hiërarchische page tables.
- Deze tabellen kunnen behoorlijk groot worden, het zoeken in deze tabellen is kostbaar.

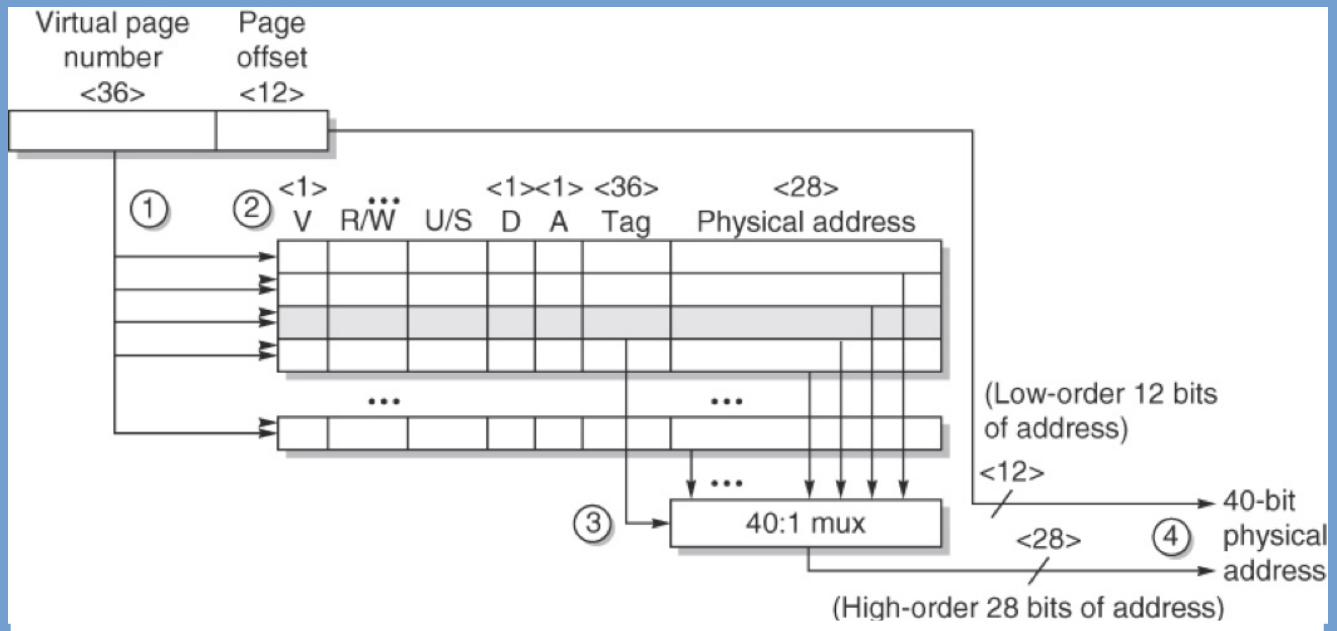
Virtual memory (4)



Taken from Computer Architecture: A Quantitative Approach, fifth edition. Fig. B.27.

Virtual memory (5)

- De penalty van een page table walk is erg groot.
- Oplossing: een cache voor adresvertalingen.
  - TLB: Translation Lookaside Buffer.
  - Fully associative, tags bevatten virtuele adressen. De data bevat het fysieke adres.



Taken from Computer Architecture: A Quantitative Approach, fifth edition. Fig. B.24.

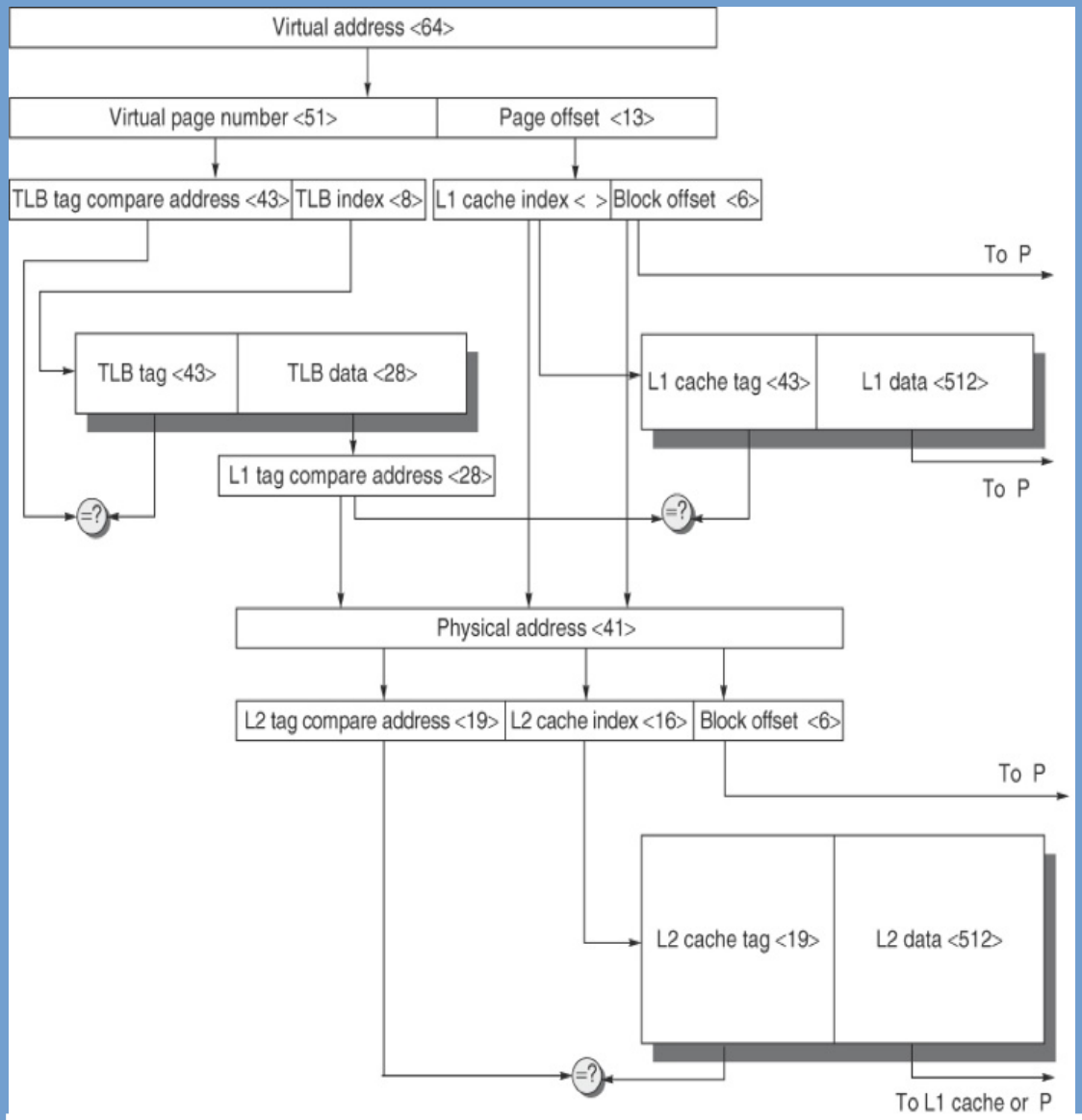
Vermijd address translation tijdens cache indexing

- “Make the common case fast”, common case zijn de hits.
  - Suggereert de caches virtueel te indexen (virtual caches).
  - Cache indexing kan plaatsvinden zonder address translation.
- Problemen met virtual caches:
  - Maar: virtuele adressen veranderen na context switch: cache flush vereist.
  - Verschillende virtuele adressen kunnen naar hetzelfde fysieke adres wijzen.

Vermijd address translation tijdens cache indexing (2)

- In caches vindt indexing en tagging plaats. Voor beide kan de keuze virtueel of fysiek worden gemaakt.
- Bijvoorbeeld: virtueel indexen, fysiek taggen.
  - Gebruik page offset uit het virtuele adres voor cache indexing.
    - (Dit is equivalent aan de page offset in het fysieke adres).
  - Terwijl de cache wordt gelezen vindt de address translation plaats.
  - In de laatste stap kan de tag match uiteindelijk op basis van fysieke adressen gebeuren.

Putting it all together



Taken from Computer Architecture: A Quantitative Approach, fifth edition. Fig. B.25.