Computerarchitectuur

H&P Ch 2. Memory Hierarchy Design

Kristian Rietveld

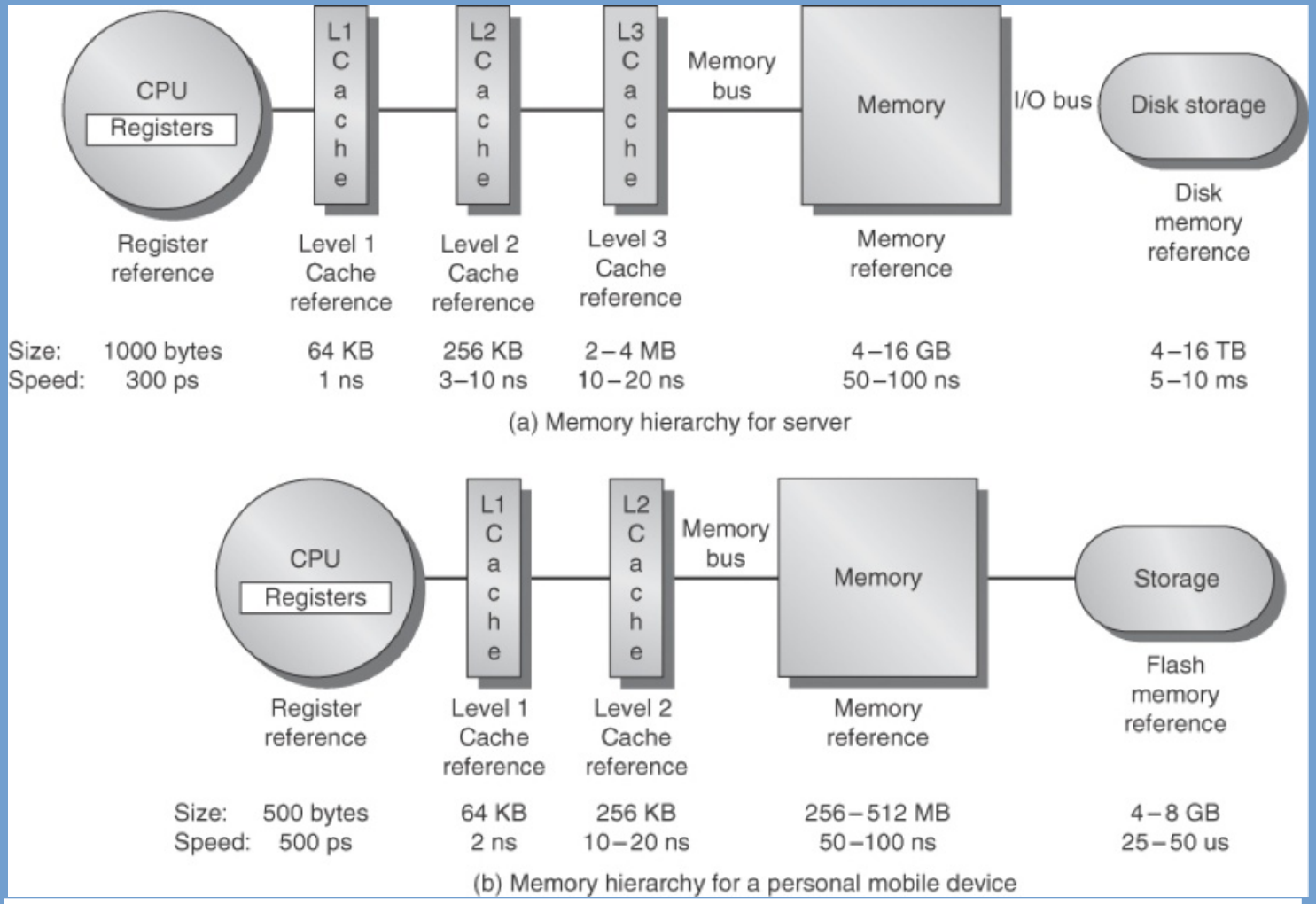
http://ca.liacs.nl/

Universiteit Leiden
The Netherlands

Motivatie

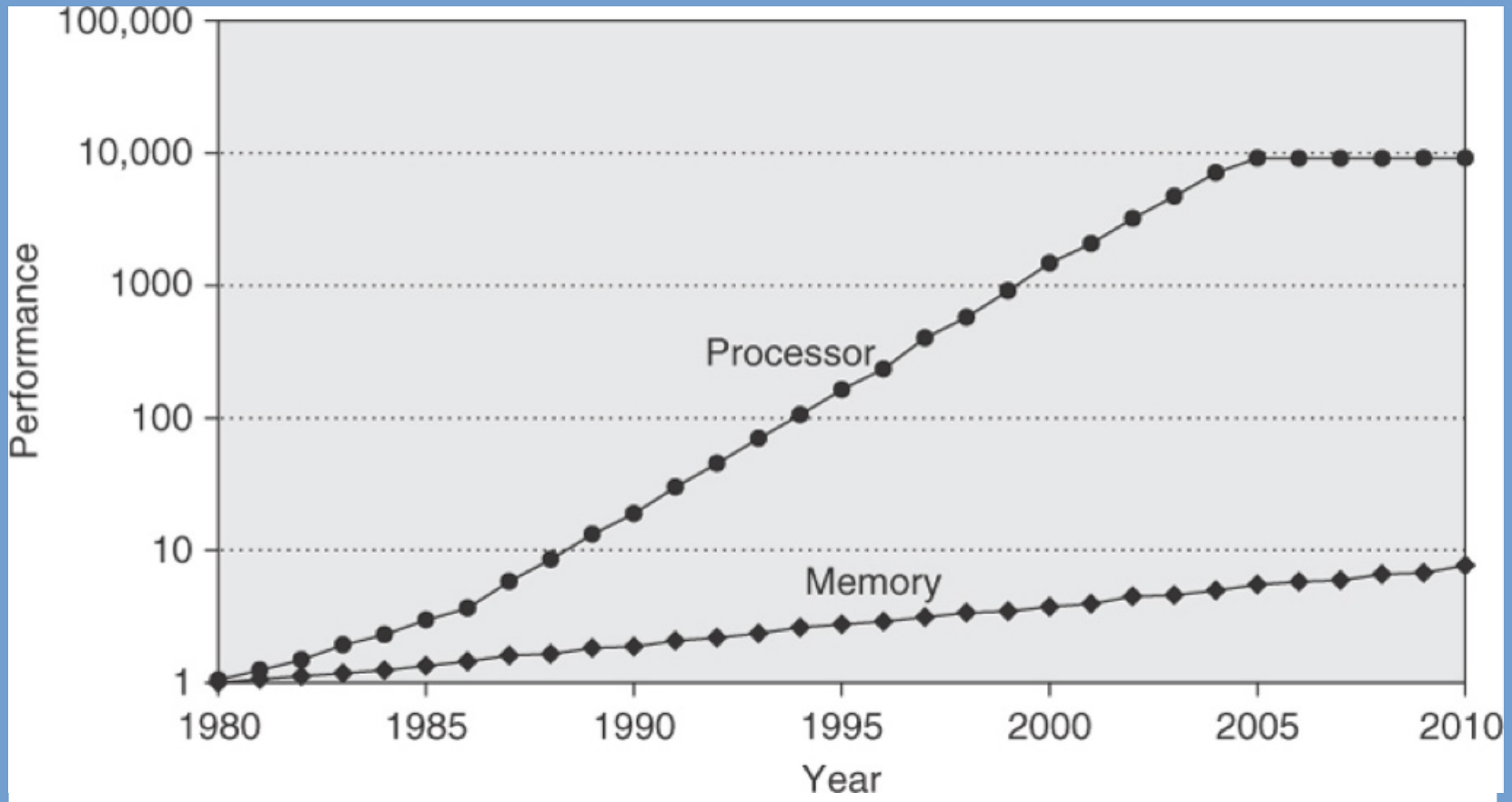
- Hoe dichter bij de CPU, hoe sneller het geheugen.
  - Maar ook: kleiner en duurder.
- Programmeurs willen een oneindige hoeveelheid snel geheugen.
- We maken gebruik van de eigenschap van locality en bouwen een geheugenhiërarchie.
  - Een aaneenschakeling van steeds goedkopere en grotere geheugens.
  - Doel: kosten per byte bijna zo laag als het laagste geheugen-niveau; snelheid bijna zo snel als het hoogste niveau.

Voorbeelden



Taken from Computer Architecture: A Quantitative Approach, fifth edition. Fig. 2.1.

Motivatie (2)



Taken from Computer Architecture: A Quantitative Approach, fifth edition. Fig. 2.2.

Multi-core impact

- Multi-cores hebben een nog hogere vereiste voor bandbreedte.
  - Samengenomen bandbreedte groeit met aantal cores.
- Voorbeeld:
  - Core i7 CPU: 2 data memory accesses per core per clock cycle.
  - 4 cores, kloksnelheid 3.2 GHz.
  - Piek: 25.6 miljard 64-bit data references per seconde. (2 * 4 * 3.2e9).
  - Ook nog: 12.8 miljard 128-bit instructies per seconde.
  - Totale piekbandbreedte: 409.6 GB/sec (!).
- Contrasteer met piekbandbreedte naar DRAM main memory: 6% hiervan (25 GB/s).
- Dit zijn maar 4 cores, moderne Intel server CPUs hebben 10 – 18 cores PER CHIP.

Impact of power

- Traditioneel werd de *average memory access time* geoptimaliseerd.
  - We zagen al dat deze is opgebouwd uit: cache hit time, miss rate, miss penalty.
- Energiegebruik wordt echter steeds belangrijker.
- Grotere on-chip caches (10 – 40MB niet ongebruikelijk) gebruiken veel static en dynamic power.
- In embedded systemen nog het grootste probleem: veel minder vermogen beschikbaar, caches soms 25% tot 50% van totale energieverbruik.

10 advanced optimizations of cache performance

- Gezien het belang van het optimaliseren van de geheugenhiërarchie, bekijken we nog 10 geavanceerde optimalisaties.
- We zagen al optimalisaties voor: hit time, miss rate en miss penalty.
- We voegen toe: *increasing cache bandwidth*, *reducing miss rate via parallelism*.

1. Small & simple L1 caches

- Kritische momenten in een cache hit:
  - Extraheer index uit adres om de cache te adresseren.
  - Vergelijk de gevonden tags met het te lezen adres.
  - Multiplexer instellen, in geval set-associative cache.
- We zitten gebonden aan de zeer kortere klokperioden.
  - Grootte caches eigenlijk niet meer toegenomen.
  - Soms wordt wel gekozen voor een hogere associativiteit.
- Andere overwegingen:
  - L1 cache access kost soms al minstens 2 cycles, dus hogere hit time door grotere associativiteit niet een groot probleem.
  - L1 cache virtually indexed, hierdoor is de grootte van de cache gelimiteerd door de page size (aantal sets) maal associativiteit.

2. Way prediction

- Idee: houd een aantal bits bij en gebruik deze om het blok (way) van de volgende cache access te voorspellen.
  - Doel: reduce hit time.
- Multiplexer wordt eerder ingesteld en maar 1 tag wordt vergeleken binnen een enkele klokperiode.
- In geval miss: in de volgende klokperiode alle andere blokken bekijken.
- Simulaties suggereren prediction accuracy boven 90% voor 2-way associative caches.
- Wordt gebruikt in ARM Cortex-A8.

3. Pipelined cache access

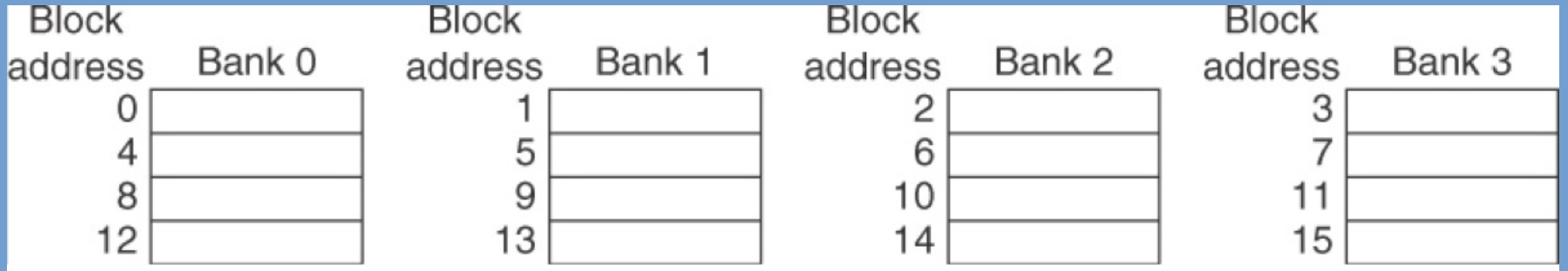
- Doel: verhogen cache bandwidth.
- Pipeline de cache, dus elke klokperiode aan een cache access beginnen.
  - Effectieve cache latency gaat wel omhoog.
  - Bandbreedte gaat ook omhoog, we kunnen meer cache reads per tijdseenheid afhandelen.
- Voorbeeld: pipeline voor L1 cache access in de i7 duurt 4 klokperioden.
  - Hierdoor kan de associativiteit omhoog.

4. Non-blocking caches

- Stel we hebben een cache miss. Wat gebeurt er dan?
  - Blocking cache: de cache 'blokkeert' totdat de miss is verholpen.
  - In feite staat alles dan stil ...
- We gaan nog out-of-order computers bekijken, die nog nuttig (ander) werk kunnen verrichten in afwachting van de data.
- Hier komt het idee voor de non-blocking cache vandaan: terwijl de cache bezig is met een miss, kan het nog steeds cache hits afhandelen (“hit under miss”).
- Effectieve miss penalty wordt verlaagd, cache bandbreedte gaat omhoog.

5. Multi-banked caches

- Verdeel de cache in onafhankelijke “banken” die tegelijkertijd kunnen worden aangesproken.
  - Idee komt voort uit optimalisaties voor RAM geheugen, hier wordt al geruime tijd banking toegepast.
- Aparte banken kunnen tegelijkertijd worden uitgelezen, dus potentie om cache bandbreedte te verhogen.
  - Bijv. i7 kan twee memory accesses per clock cycle, wanneer deze in verschillende banks staan.



Taken from Computer Architecture: A Quantitative Approach, fifth edition. Fig. 2.6.

6. Critical word first

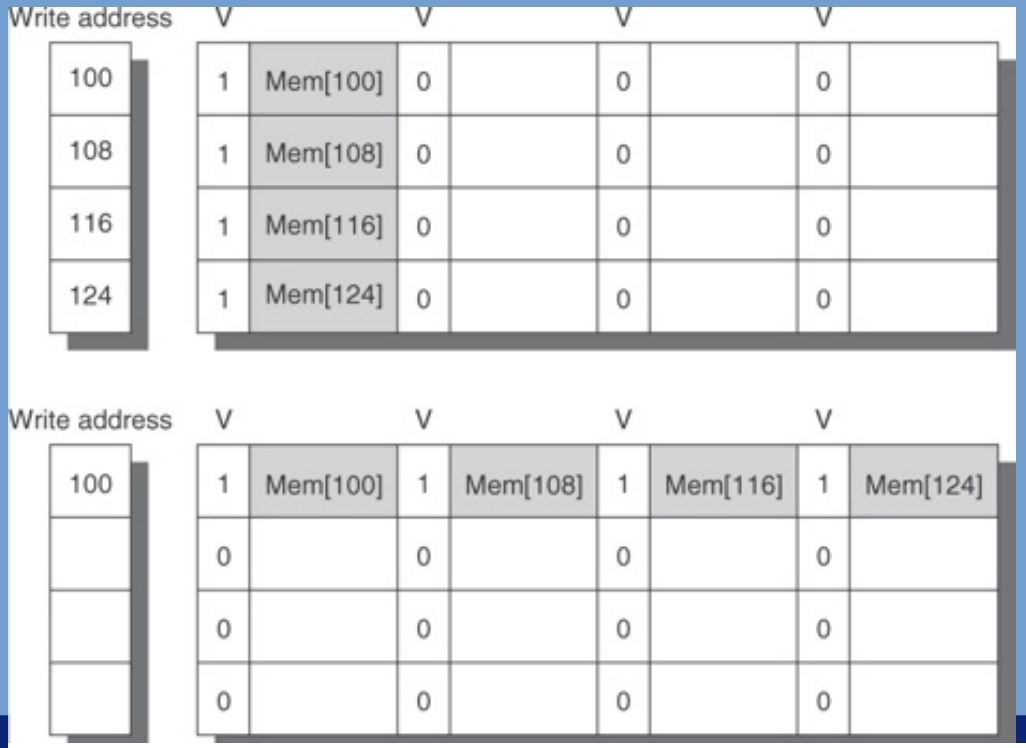
- Caches werken altijd per cache block, dus in geval miss wordt een heel block geladen.
  - Processor hebben vaak maar 1 word of 1 double word nodig.

- *Critical word first*: stuur het benodigde word naar de CPU zodra het beschikbaar is, terwijl de CPU verder gaat wordt de rest van het block ingeladen.
- *Early restart*: laad data in normale volgorde, laat CPU verder gaat zodra het benodigde word binnen is.

- Doel: verkleinen miss penalty.

7. Write buffer merging

- Zowel write-through als write-back caches maken gebruik van write buffers.
  - Write buffer schrijft data naar geheugen, terwijl CPU met andere dingen verder gaat.
- Bij het invoegen nieuw element in de write buffer, kunnen de huidige elementen worden bekeken of er sprake is van overlap.
  - Minder en grotere writes -> altijd beter.
  - Minder stalls door volle write buffer.
  - Verkleint (write) miss penalty.



Taken from Computer Architecture: A Quantitative Approach, fifth edition. Fig. 2.7.

8. Compiler optimizations

- Verlaag de miss rate door middel van software optimalisatie.
- Gebruik compilertransformaties om instructie en data miss rate te verlagen.
- We bespreken kort Loop Interchange en Loop Blocking, het doel is dit te laten terugkomen in het practicum.

Loop interchange

- Aannames, arrays passen niet in cache, opgeslagen in row-major order. Grootte [5000, 100].

```
/* Before */
for (j = 0; j < 100; j = j + 1)
  for (i = 0; i < 5000; i = i + 1)
     x[i][j] = 2 * x[i][j];
```

Loop interchange

- Aannames, arrays passen niet in cache, opgeslagen in row-major order. Grootte [5000, 100].

```
/* Before */
for (j = 0; j < 100; j = j + 1)
  for (i = 0; i < 5000; i = i + 1)
     x[i][j] = 2 * x[i][j];

/* After */
for (i = 0; i < 5000; i = i + 1)
  for (j = 0; j < 100; j = j + 1)
     x[i][j] = 2 * x[i][j];
```

- Dus: we maken gebruik van de eigenschap van spatial locality.

Loop Blocking

- Drie matrices, x, y en z; N bij N elementen.

```
/* Before */
for (i = 0; i < N; i = i + 1)
  for (j = 0; j < N; j = j + 1) {
     r = 0;
     for (k = 0; k < N; k = k + 1)
        r = r + y[i][k] * z[k][j];
     x[i][j] = r;
  }
```

Loop Blocking (2)

- Drie matrices, x, y en z; N bij N elementen.
- Blocking factor *B*.

```
/* After */
for (jj = 0; jj < N; jj = jj+B)
for (kk = 0; kk < N; kk = kk+B)
for (i = 0; i < N; i = i + 1)
  for (j = jj; j < min(jj+B,N); j = j + 1) {
     r = 0;
     for (k = kk; k < min(kk+B,N); k = k + 1)
        r = r + y[i][k] * z[k][j];
     x[i][j] += r;
  }
```

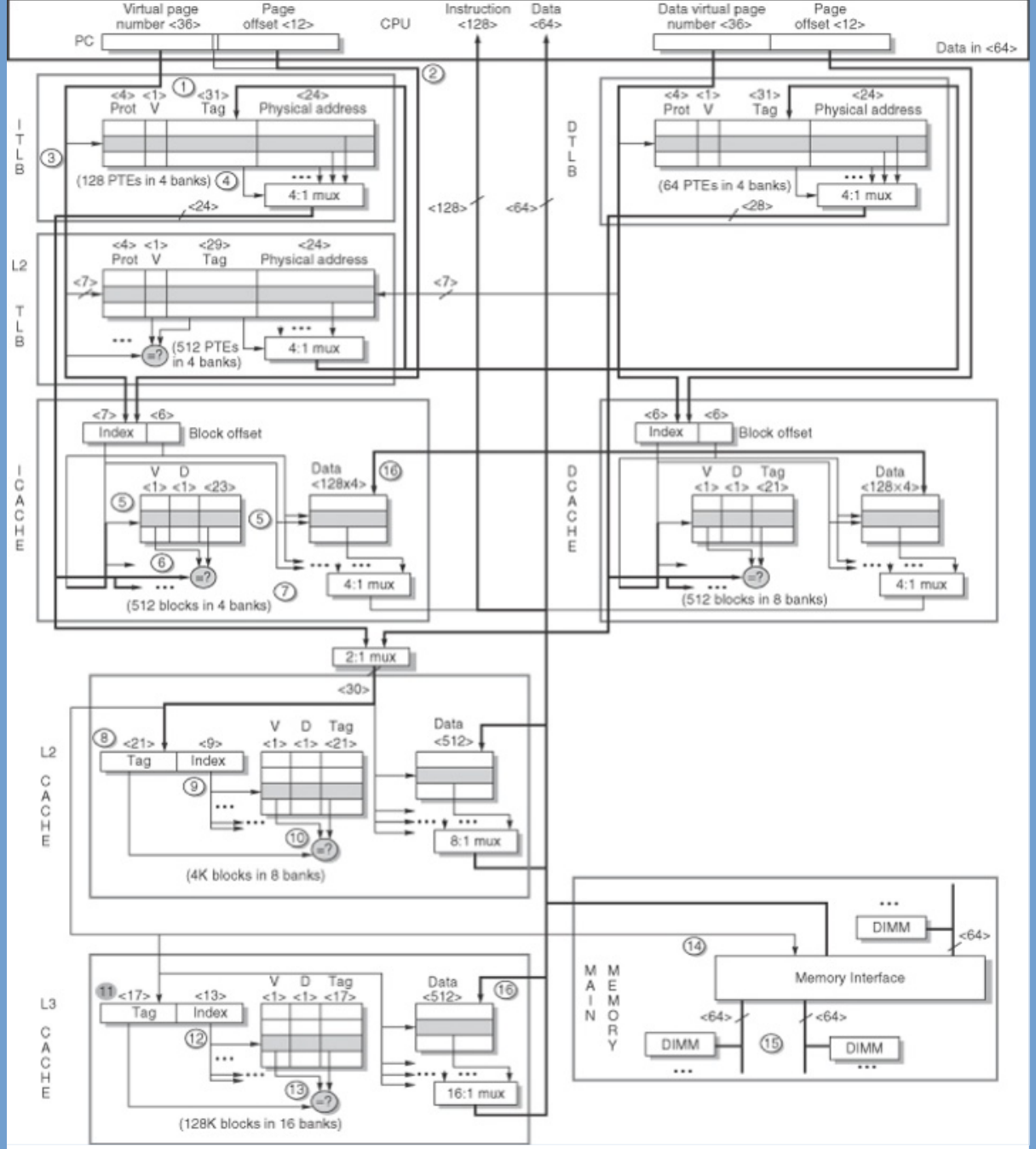
9. Hardware prefetching

- Idee: data (of instructies) alvast ophalen, voordat het programma er expliciet om vraagt.
- Vaak gedaan door hardware, los van de cache.
- Gebaseerd op “stride detection”, het door de data lopen met stapjes van 1, 4, 8, … bytes.

- In feite gebruiken we geheugenbandbreedte die anders niet zou worden gebruikt.

10. Compiler-directed prefetching

- Compilers hebben vaak meer overzicht over het gehele programma, sommige architecturen ondersteunen prefetch-instructies.
- In een bepaalde loop-iteratie kun je bijvoorbeeld alvast de data voor over *n* iteraties alvast laten “prefetchen”. Hiermee hoop je een data miss te omzeilen.
- Heeft natuurlijk alleen zin als je zeker weet dat een load instructie tot een data miss zal leiden, anders bedragen de kosten het uitvoeren van de extra instructies.



Taken from Computer Architecture: A Quantitative Approach, fifth edition. Fig. 2.21.

Memory Technology and Optimizations

Main memory

- Main memory bandbreedte neemt sneller toe dan de verbetering in main memory latency.
- Met alleen verbeteringen aan cache kan het CPU-geheugen gat niet worden gedicht, ook verbeteringen aan main memory nodig.
- Voorgaande trends: banked memory, verbreden geheugenchips, verbreden van de bus.

- *Access time:* tijd tussen aanvraag leesactie en het beschikbaar komen van de data.
- *Cycle time:* minimum tijd tussen twee “losse” geheugenacties.

SRAM

- SRAM: Static RAM
  - Geen refresh nodig, data blijft waar het is (i.t.t. DRAM).
- 6 transistoren per bit nodig. (cf. flip-flop).
- Voorheen werden aparte SRAM chips gebruikt voor caches, nu zit alles op de processor die.

- Grootste L3 caches on-chip: 12 tot 40 MB.
- Main memory echter: > 16 GB

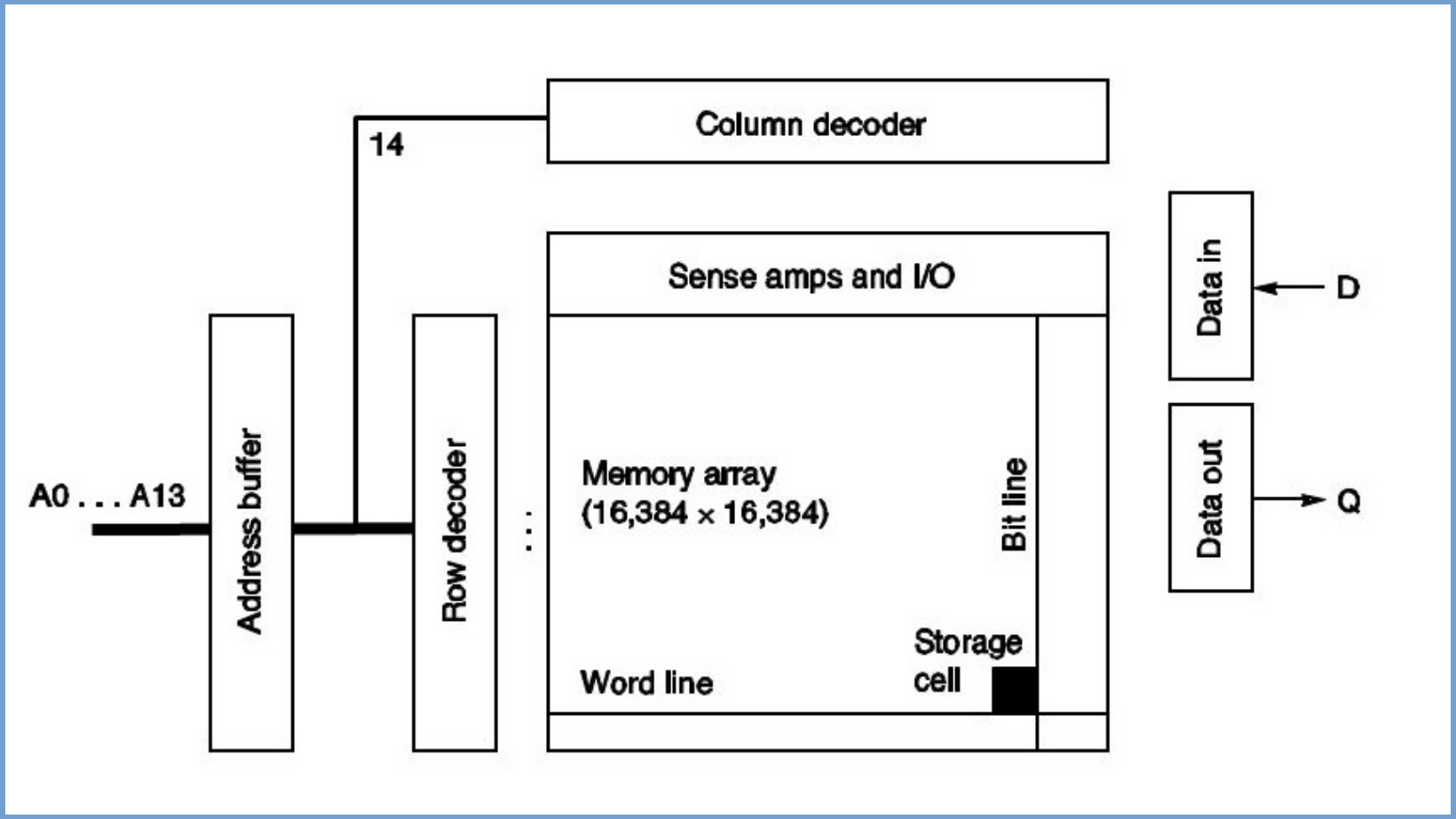
DRAM

- DRAM: Dynamic RAM.
- Per bit maar 1 transistor nodig. Maar het lezen van de bit vernietigt de informatie …
  - Maar wel een veel hogere informatiedichtheid.
- Tijdens de cycle moet de informatie na het lezen dus ook weer opnieuw worden weggeschreven.
- Daarnaast moet de gehele geheugenmodule periodiek worden ververst. Typisch elke 8 ms. Geheugencontrollers bevatten hardware om dit automatisch te doen.
  - Alle rijen moeten worden uitgelezen, zodat deze weer worden weggeschreven.

DRAM (2)

- DRAM wordt georganiseerd als rechthoekige matrix.
- Adres wordt in 2 delen naar de DRAM module gestuurd:
  - Eerst de rij: row access strobe (RAS). Hiermee wordt de hele rij in een buffer gezet.
  - Daarna de kolom: column access strobe (CAS). Het gewenste deel wordt uit de buffer gelezen.
- Tijdens een refresh is de module niet beschikbaar, memory access time kan dus variëren!

DRAM (2)


Column decoder
14
Sense amps and I/O
Data in
D
Data out
Q
A0 . . . A13
Address buffer
Row decoder
Memory array
(16,384 × 16,384)
Bit line
Storage
cell
Word line

DRAM (3)

- Verbeteringen aan DRAM door de jaren heen:
  - Row address hergebruiken als er meerdere kolommen uit dezelfde rij worden gelezen.
  - SDRAM: Synchronous DRAM. Voorheen was DRAM asynchroon, bij elke transfer was er synchronisatie met de controller nodig. Met een aparte DRAM klok is dit niet meer nodig.
  - SDRAM burst mode: 8 of meer 16-bit transfers zonder steeds een nieuw adres te sturen.
  - DDR: Double Data Rate: verstuur data zowel bij de “rising edge” en “falling edge” van de DRAM klok. Potentieel voor bandbreedte verdubbeling.

DRAM (4)

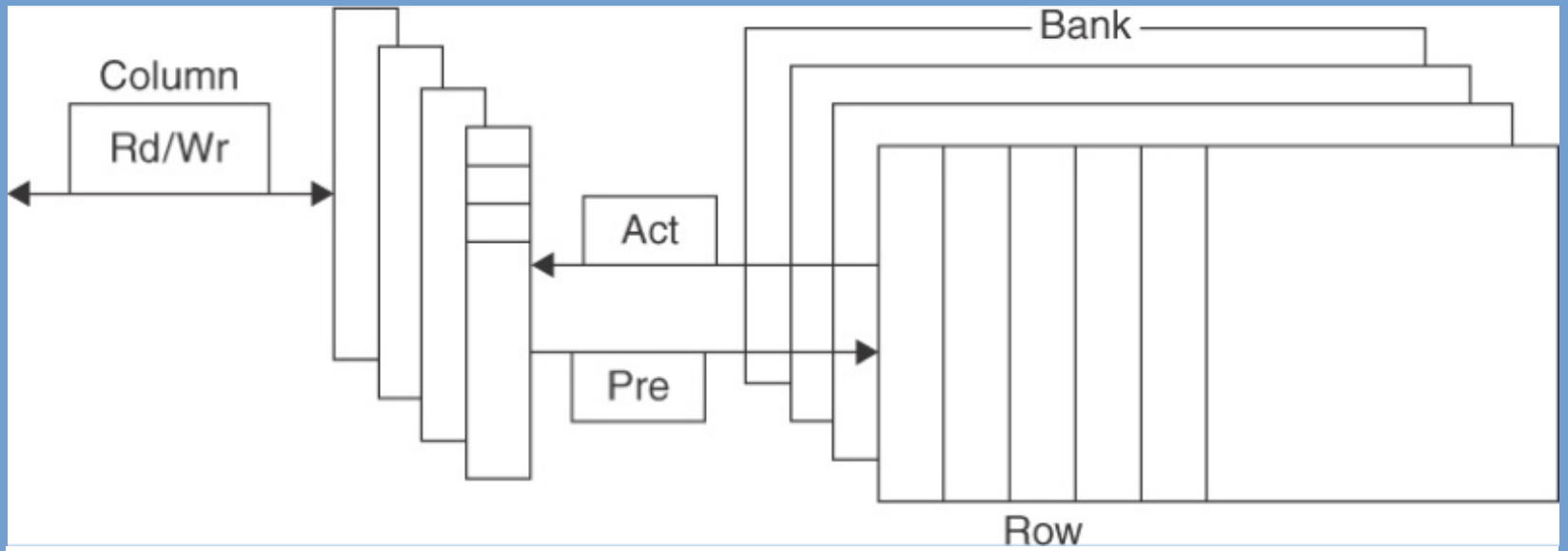


Figure 2.12 Internal organization of a DRAM. Modern DRAMs are organized in banks, typically four for DDR3. Each bank consists of a series of rows. Sending a PRE (precharge) command opens or closes a bank. A row address is sent with an Act (activate), which causes the row to transfer to a buffer. When the row is in the buffer, it can be transferred by successive column addresses at whatever the width of the DRAM is (typically 4, 8, or 16 bits in DDR3) or by specifying a block transfer and the starting address. Each command, as well as block transfers, are synchronized with a clock.

DRAM (5)

- PC25600 DIMM?
  - DDR4 chips
  - Kloksnelheid: 1066 – 1600 MHz.
  - Miljoenen transfers per seconde: 2133 – 3200 (x2)
  - Naam: DDR4-3200 (dus niet naar de kloksnelheid).
  - 1600 M x 2 (transfers) x 8 (bytes) = 25600 MB/sec

GDDR5

- GDDR5 is gebaseerd op DDR3.
  - G van Graphics.
- GPUs hebben hogere bandbreedte vereisten, code laat minder locality zien.
  - Bredere interface (bijv. 32-bit).
  - Hogere kloksnelheid. Om problemen te voorkomen met signaling, GDRAM vaak direct met GPU verbonden en geen sprake van een DIMM-socket systeem.

Flash geheugen

- Enorm in opkomst: smart phones, SSDs, ...
- Data wordt ook zonder stroom vastgehouden.
- Wel anders dan SDRAM:
  - Wissen (per blok), voordat het geheugen kan worden overschreven.
  - Veel lager stroomgebruik als er niet wordt gelezen/geschreven.
  - Elk blok kan niet veel vaker dan 100000 keer worden herschreven.
  - Flash is trager dan SDRAM, maar veel sneller dan (magnetische) disk.

ECC RAM

- ECC: Error Correcting Code
- Gebruik van parity bits om dynamische fouten te ontdekken en op te lossen.
  - Dynamische fouten komen voor door bijv. kosmische achtergrondstraling.
  - Hoe groter het geheugen (en kleiner het fabricageproces), hoe groter de kans dat dit gebeurt.
- ECC in principe standaard in serversystemen.

- (IBM analyse: bij gebruik ECC in 10000 CPU server, 4 GB RAM per processor: ~1 undetected, unrecoverable error per 7.5 uur).

Looking ahead

- Een aantal trends:
  - Verbeteringen in DRAM technologie remt af.
  - Flash wordt steeds meer gebruikt. SDRAM blijft nodig door bulk erase-rewrite cycle.
  - Magnetic RAM, phase change RAM.
    - Beide nonvolatile (data blijft behouden zonder stroom)
    - Grotere dichtheid als DRAM.
    - Alleen vervanging voor flash? Of misschien ook DRAM?

Looking ahead

- Tot nu toe hebben we een hoop kunnen doen om de memory latency te verbergen …
  - Multi-level cache, compiler techniques, out-of-order execution.

- Grotere kans dat meer parallellisme ook meer mogelijkheden geeft om latency te verbergen.
- Memory delays blijven tegengaan met instruction- en thread-level parallelism (hoofdstukken 3 en 5).