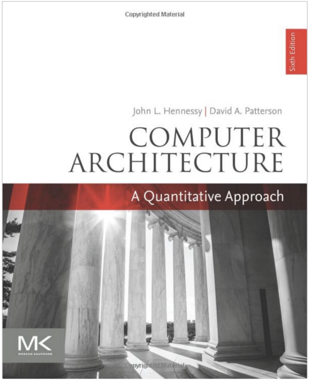
Master Informatics Eng.

2022/23

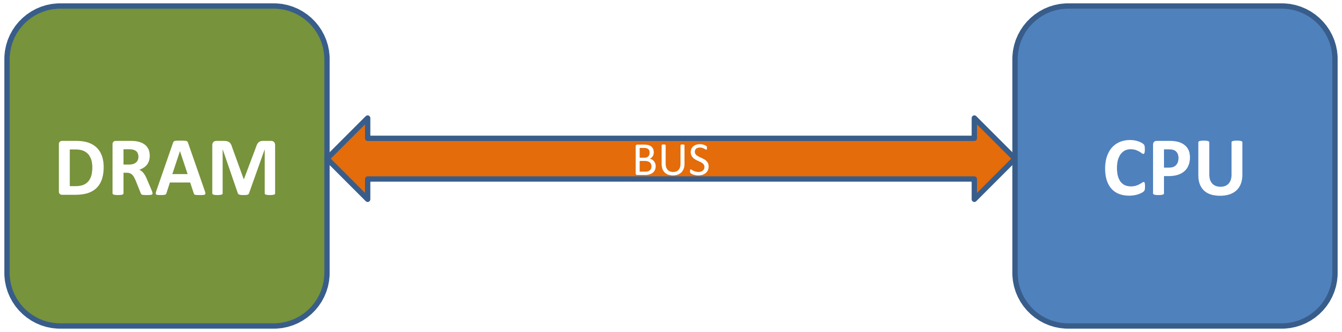
A.J.Proença

Memory Hierarchy

(most slides are from JLS & borrowed from this book:  *)*

Refresh

Hiato Processador-Memória

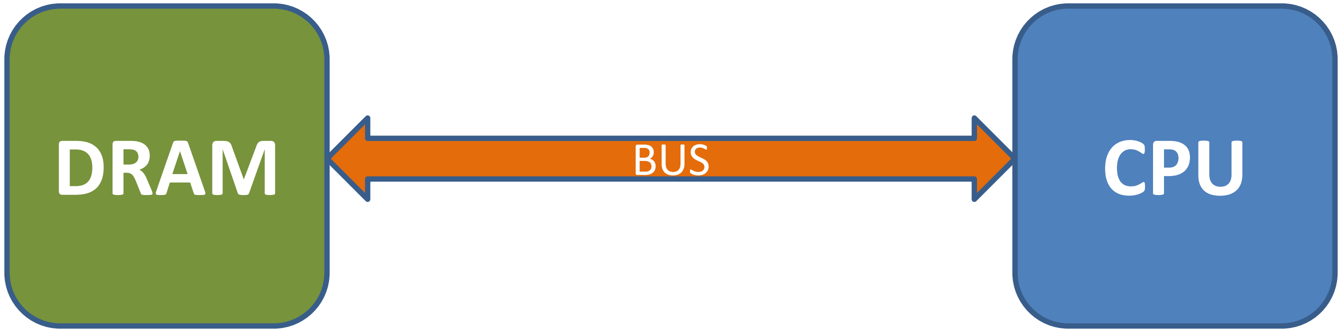


Para cada instrução:

1. Ler instrução
2. Ler operando
3. Escrever Resultado

Refresh

Hiato Processador-Memória



Suponhamos um processador a executar um programa que consiste numa longa sequência de instruções inteiras:

```
        addl  reg,  [Mem]
```

Se a **instrução** tiver **6 bytes** de tamanho e cada **inteiro 4 bytes** a execução de cada instrução implica um movimento de **6**+**2*4** = **16 bytes**.

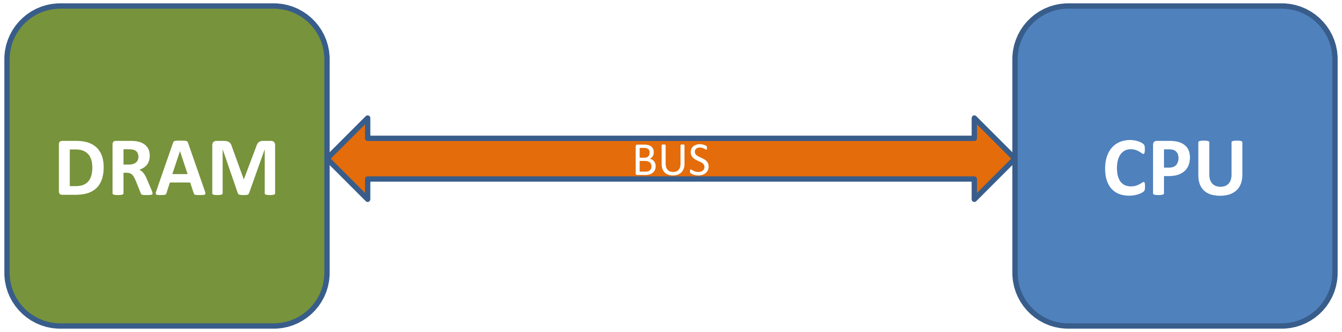
Se frequência = **2.5 GHz** e o CPI=1 então são executadas $2.5*10^9$ inst/seg

A largura de banda necessária para manter o processador alimentado é de:

$$\mathbf{2.5*10^9 * 16 = 40\ GB/s}$$

Refresh

Hiato Processador-Memória



Standard name **(single channel)**	**Peak transfer rate**
DDR2-800	6 400 MB/s
DDR3-1066	8 533 MB/s
DDR3-1600	12 800 MB/s
DDR4-2666	21 800 MB/s

Largura de banda exigida neste exemplo: $2.5*10^9 * 16 =$ **40 GB/s**

Hiato processador-memória:

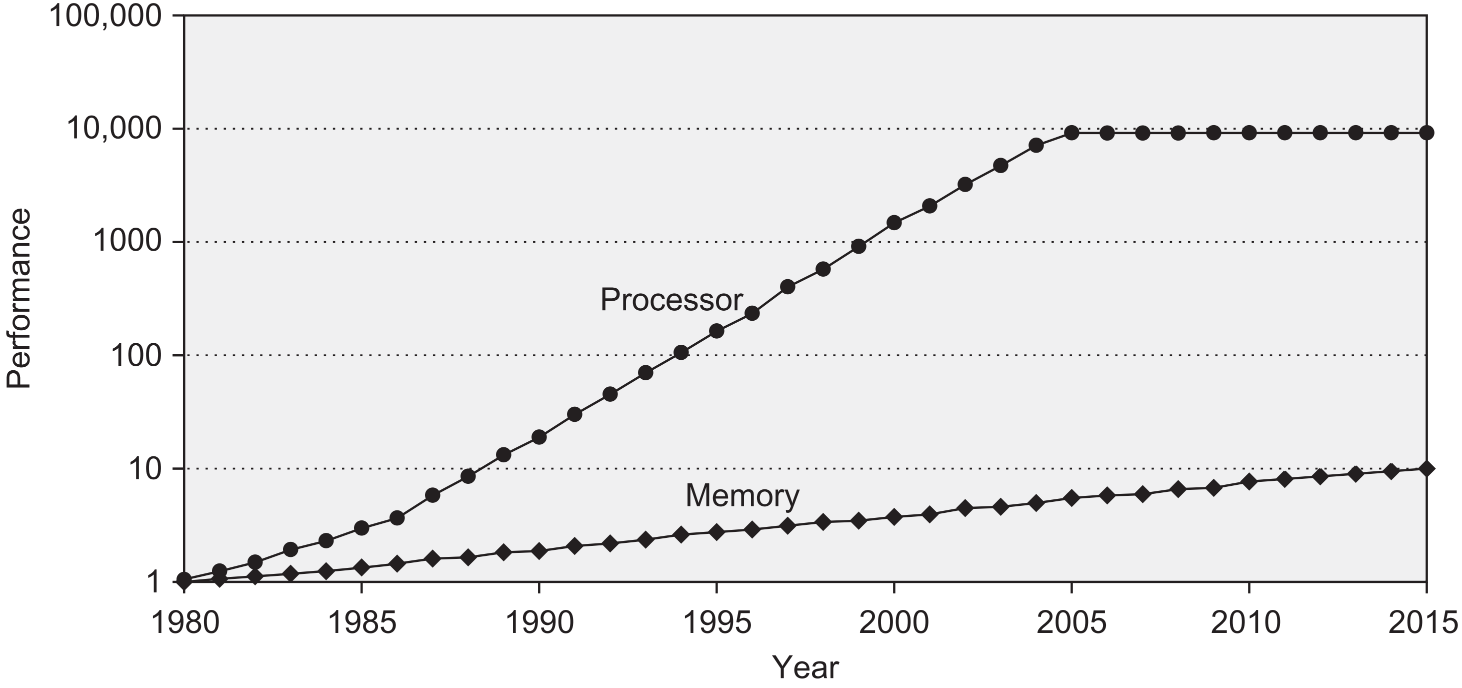
"A memória é incapaz de alimentar o processador com instruções e dados a uma taxa suficiente para o manter constantemente ocupado"

Refresh

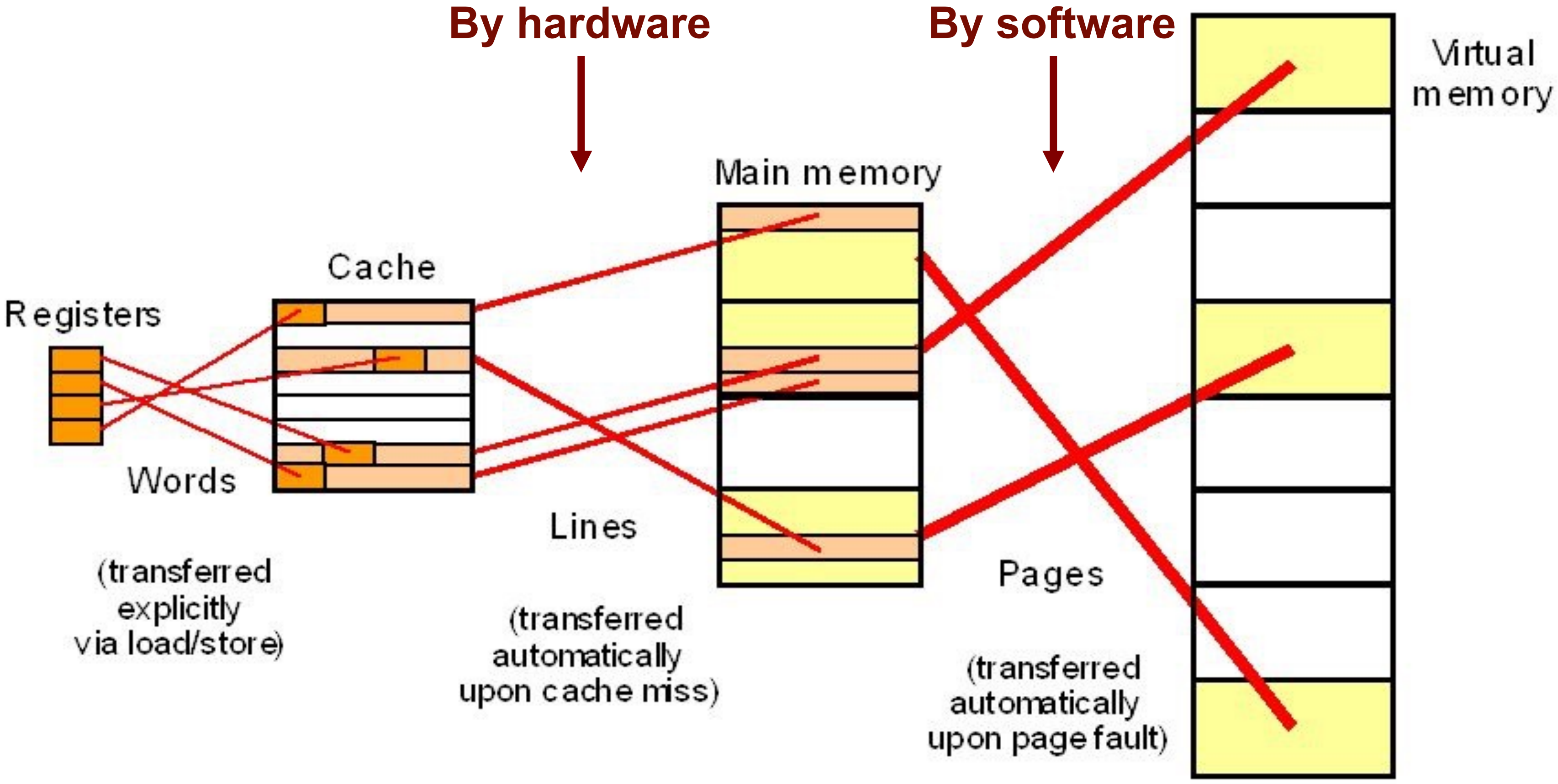
Hiato Processador-Memória

- As diferentes taxas de aumento do desempenho destes dois componentes levam a um aumento do hiato Processador-Memória (*“the memory gap”*) com o tempo

- O hiato processador-memória é um dos principais obstáculos à melhoria do desempenho dos sistemas de computação



Memory hierarchy: the big picture


By hardware
By software
Registers
Words
(transferred explicitly via load/store)
Cache
Lines
(transferred automatically upon cache miss)
Main memory
Pages
(transferred automatically upon page fault)
Virtual memory

Data movement in a memory hierarchy.

Refresh

Hierarquia da Memória: Localidade

O **princípio da localidade**, exibido pela maior parte dos programas no acesso à memória, permite acelerar os acessos à mesma através de uma hierarquia

> *“Os programas tendem a aceder a uma porção limitada de memória num dado período de tempo”*

O **princípio da localidade** permite utilizar memória mais rápida para armazenar a informação usada mais frequentemente/recentemente

Também permite tirar partido da largura de banda, uma vez que a informação transferida entre diferentes níveis da hierarquia é efectuada por blocos

Refresh

Hierarquia da Memória: Localidade Temporal

Localidade Temporal – um elemento de memória acedido pelo processador será, com grande probabilidade, acedido de novo num futuro próximo.

Exemplos: tanto as instruções dentro dos ciclos, como as variáveis usadas como contadores de ciclos, são acedidas repetidamente em curtos intervalos de tempo.

Consequência – a 1ª vez que um elemento de memória é acedido deve ser lido do nível mais baixo (por exemplo, da memória central).

Mas da 2ª vez que é acedido existem grandes hipóteses que se encontre na *cache*, evitando-se o tempo de leitura da memória central.

Refresh

Hierarquia da Memória: Localidade Espacial

Localidade Espacial – se um elemento de memória é acedido pelo CPU, então elementos com endereços na proximidade serão, com grande probabilidade, acedidos num futuro próximo.

Exemplos: as instruções do programa são normalmente acedidas em sequência, assim como, na maior parte dos programas, os elementos de vectores/matrizes.

Consequência – a 1ª vez que um elemento de memória é acedido, deve ser lido do nível mais baixo (por exemplo, memória central) não apenas esse elemento, mas sim um **bloco** de elementos com endereços na sua vizinhança.

Se o processador, nos próximos ciclos, acede a um endereço vizinho do anterior (ex.: próxima instrução ou próximo elemento de um vector) aumenta a probabilidade de esta estar na *cache*.

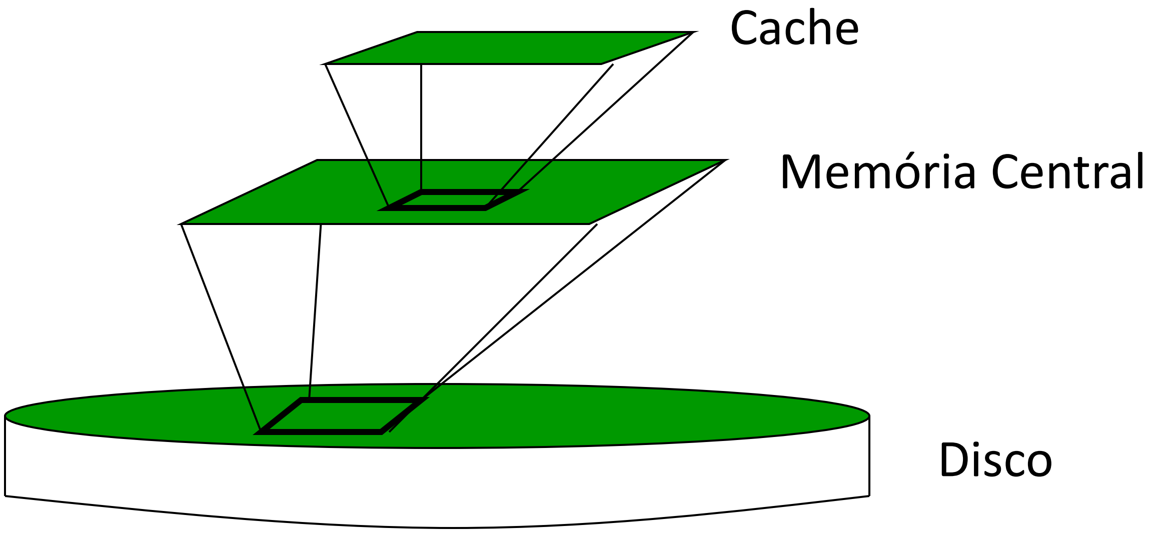
Refresh

Hierarquia de Memória: Inclusão

Os dados contidos num nível mais próximo do processador são um sub-conjunto dos dados contidos no nível anterior.

O nível mais baixo contém a totalidade dos dados.

Os dados são copiados entre níveis em *blocos*



Refresh

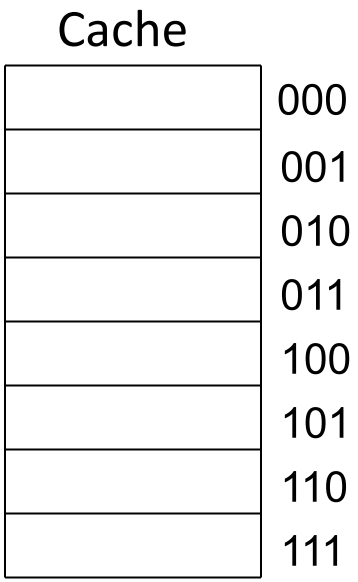
Hierarquia de Memória: Terminologia

Linha – a cache está dividida em linhas. Cada linha tem o seu endereço (índice) e tem a capacidade de um bloco

Bloco – Quantidade de informação que é transferida de cada vez da memória central para a cache (ou entre níveis de cache). É igual à capacidade da linha.

***Hit* –** Diz-se que ocorreu um *hit* quando o elemento de memória acedido pelo CPU se encontra na cache.

***Miss* –** Diz-se que ocorreu um *miss* quando o elemento de memória acedido pelo CPU não se encontra na cache, sendo necessário lê-lo do nível inferior da hierarquia.



Refresh

Hierarquia de Memória: Terminologia

***Hit rate* –** Percentagem de *hits* ocorridos relativamente ao total de acessos à memória.

***Miss rate* –** Percentagem de *misses* ocorridos relativamente ao total de acessos à memória. *Miss rate = (1 – hit rate)*

***Hit time* –** Tempo necessário para aceder à *cache*, incluindo o tempo necessário para determinar se o elemento a que o CPU está a aceder se encontra ou não na cache.

***Miss penalty* –** Tempo necessário para carregar um bloco da memória central (ou de um nível inferior) para a cache quando ocorre um *miss.*

Reading suggestions (from CAQA 6th Ed)

Review of memory hierarchy:	App. B
Basics: a Short Review	2.1
Memory Technology and Optimizations	2.2
Ten Advanced Optimizations of Cache Performance	2.3
Cross-Cutting Issues: The Design of Memory Hierarchies	2.5

Refresh

Appendix B
Review of Memory Hierarchy

Keywords

cache

virtual memory

memory stall cycles

direct mapped

valid bit

block address

write through

instruction cache

average memory access time

cache hit

page

miss penalty

fully associative

dirty bit

block offset

write back

data cache

hit time

cache miss

page fault

miss rate

n-way set associative

least recently used

tag field

write allocate

unified cache

misses per instruction

block

locality

address trace

set

random replacement

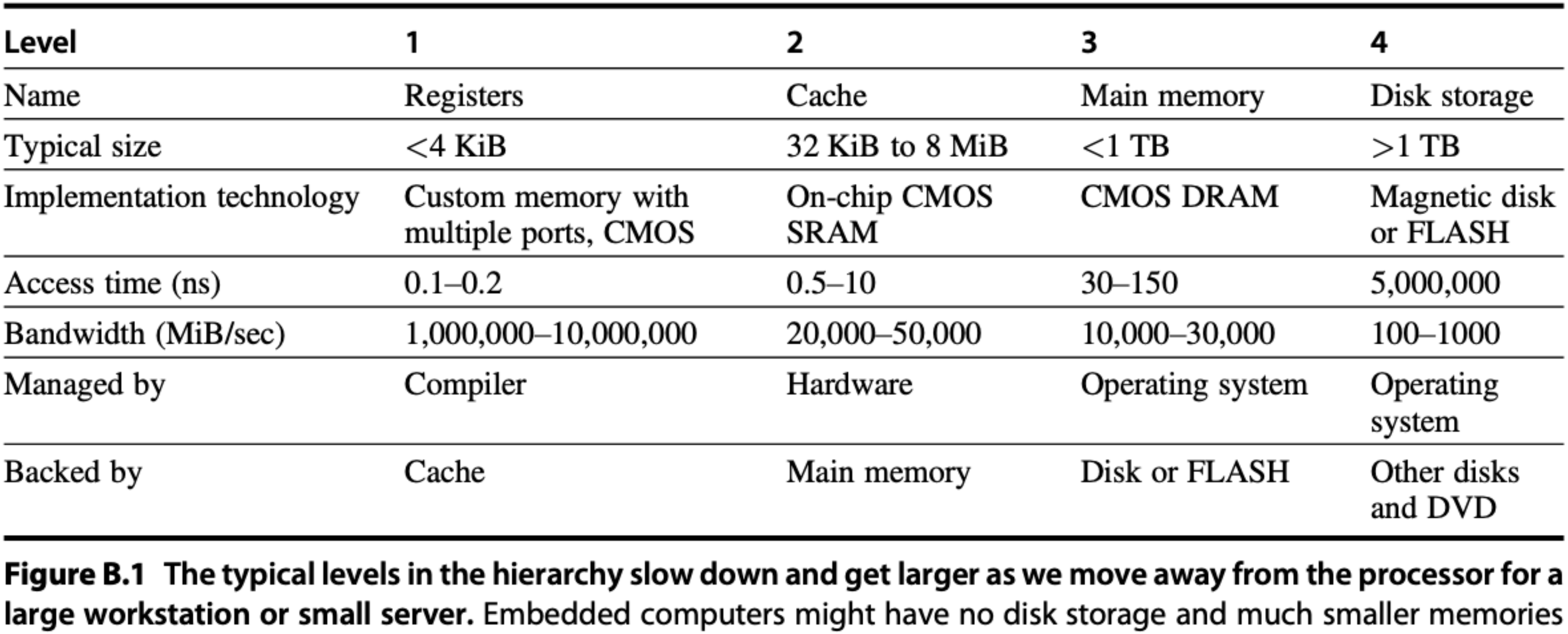
index field

no-write allocate

write buffer

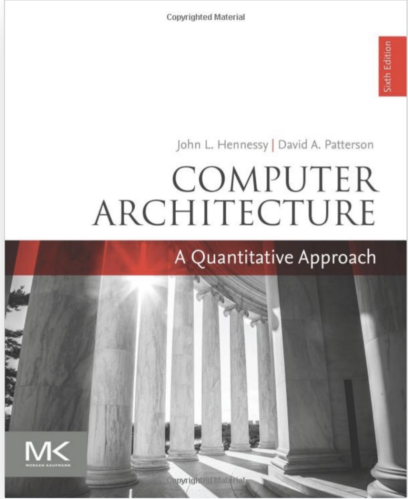
write stall

Typical levels in hierarchy *(2017)*



Level	1	2	3	4
Name	Registers	Cache	Main memory	Disk storage
Typical size	<4 KiB	32 KiB to 8 MiB	<1 TB	>1 TB
Implementation technology	Custom memory with multiple ports, CMOS	On-chip CMOS SRAM	CMOS DRAM	Magnetic disk or FLASH
Access time (ns)	0.1–0.2	0.5–10	30–150	5,000,000
Bandwidth (MiB/sec)	1,000,000–10,000,000	20,000–50,000	10,000–30,000	100–1000
Managed by	Compiler	Hardware	Operating system	Operating system
Backed by	Cache	Main memory	Disk or FLASH	Other disks and DVD

Figure B.1 The typical levels in the hierarchy slow down and get larger as we move away from the processor for a large workstation or small server. Embedded computers might have no disk storage and much smaller memories



Chapter 2

Memory Hierarchy Design

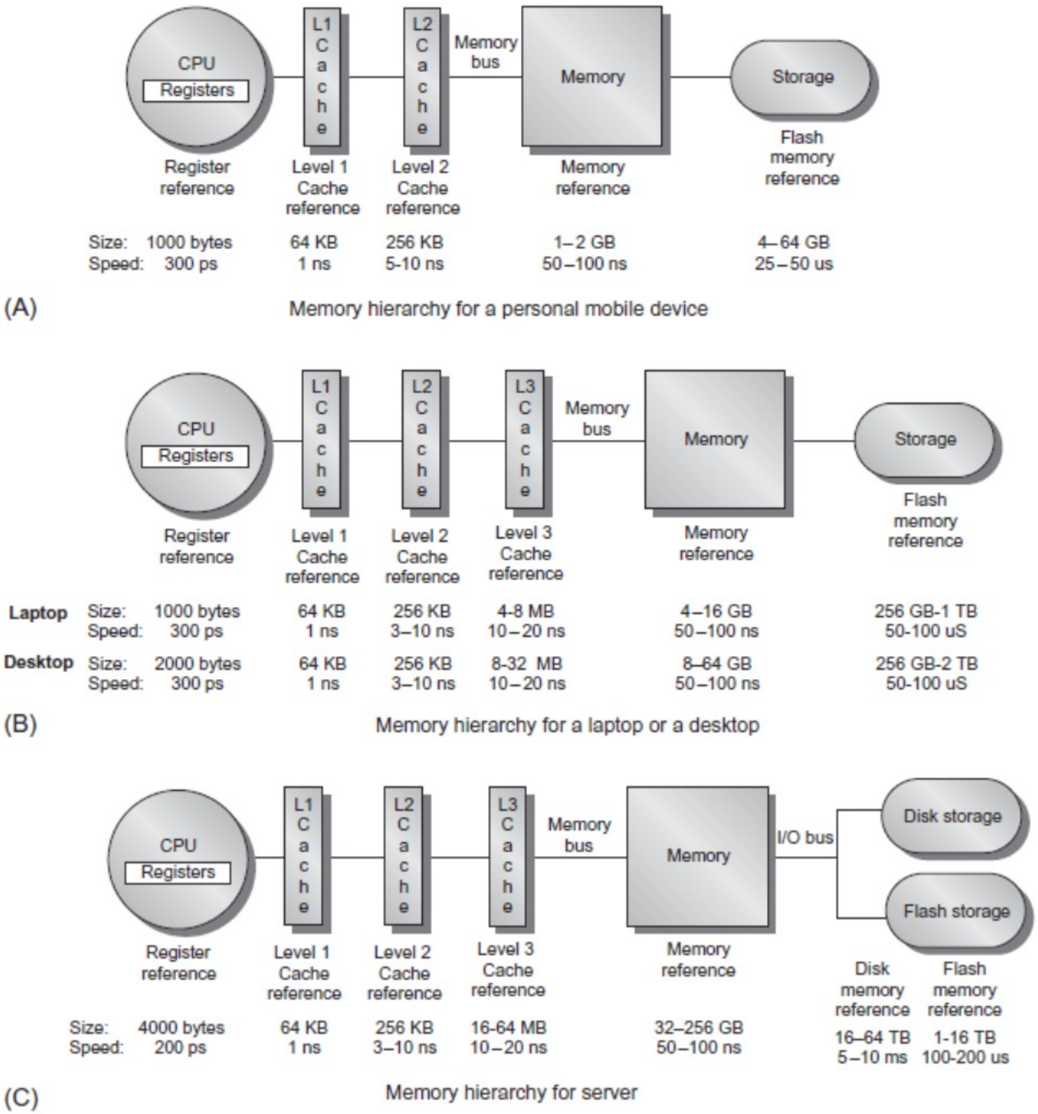
Introduction

Refresh

- Programmers want unlimited amounts of memory with low latency
- Fast memory technology is more expensive per bit than slower memory
- Solution: organize memory system into a hierarchy
  - Entire addressable memory space available in largest, slowest memory
  - Incrementally smaller and faster memories, each containing a subset of the memory below it, proceed in steps up toward the processor
- Temporal and spatial locality insures that nearly all references can be found in smaller memories
  - Gives the allusion of a large, fast memory being presented to the processor

Memory Hierarchy

Refresh



CPU

Registers

L1 Cache

L2 Cache

Memory bus

Memory

Storage

	Register reference	Level 1 Cache reference	Level 2 Cache reference	Memory reference	Flash memory reference
Size:	1000 bytes	64 KB	256 KB	1–2 GB	4–64 GB
Speed:	300 ps	1 ns	5-10 ns	50–100 ns	25–50 us

(A) Memory hierarchy for a personal mobile device

CPU

Registers

L1 Cache

L2 Cache

L3 Cache

Memory bus

Memory

Storage

		Register reference	Level 1 Cache reference	Level 2 Cache reference	Level 3 Cache reference	Memory reference	Flash memory reference
Laptop	Size:	1000 bytes	64 KB	256 KB	4-8 MB	4–16 GB	256 GB-1 TB
	Speed:	300 ps	1 ns	3–10 ns	10–20 ns	50–100 ns	50-100 uS
Desktop	Size:	2000 bytes	64 KB	256 KB	8-32 MB	8–64 GB	256 GB-2 TB
	Speed:	300 ps	1 ns	3–10 ns	10–20 ns	50–100 ns	50-100 uS

(B) Memory hierarchy for a laptop or a desktop

CPU

Registers

L1 Cache

L2 Cache

L3 Cache

Memory bus

Memory

I/O bus

Disk storage

Flash storage

	Register reference	Level 1 Cache reference	Level 2 Cache reference	Level 3 Cache reference	Memory reference	Disk memory reference	Flash memory reference
Size:	4000 bytes	64 KB	256 KB	16-64 MB	32–256 GB	16–64 TB	1-16 TB
Speed:	200 ps	1 ns	3–10 ns	10–20 ns	50–100 ns	5–10 ms	100-200 us

(C) Memory hierarchy for server

Key Memory Hierarchy Questions

Refresh

1. Block placement
   - *Q1: Where Can a Block be Placed in a Cache?*
2. Block identification
   - *Q2: How Is a Block Found If It Is in the Cache?*
3. Block replacement
   - *Q3: Which Block Should be Replaced on a Cache Miss?*
4. Write strategy
   - *Q4: What Happens on a Write?*

Block Placement

Refresh

- Determined by associativity
  - Direct mapped (1-way associative)
    - One choice for placement
  - n-way set associative
    - n choices within a set
  - Fully associative
    - Any location
- Higher associativity reduces miss rate
  - Increases complexity, cost, and access time

Block Placement

Refresh

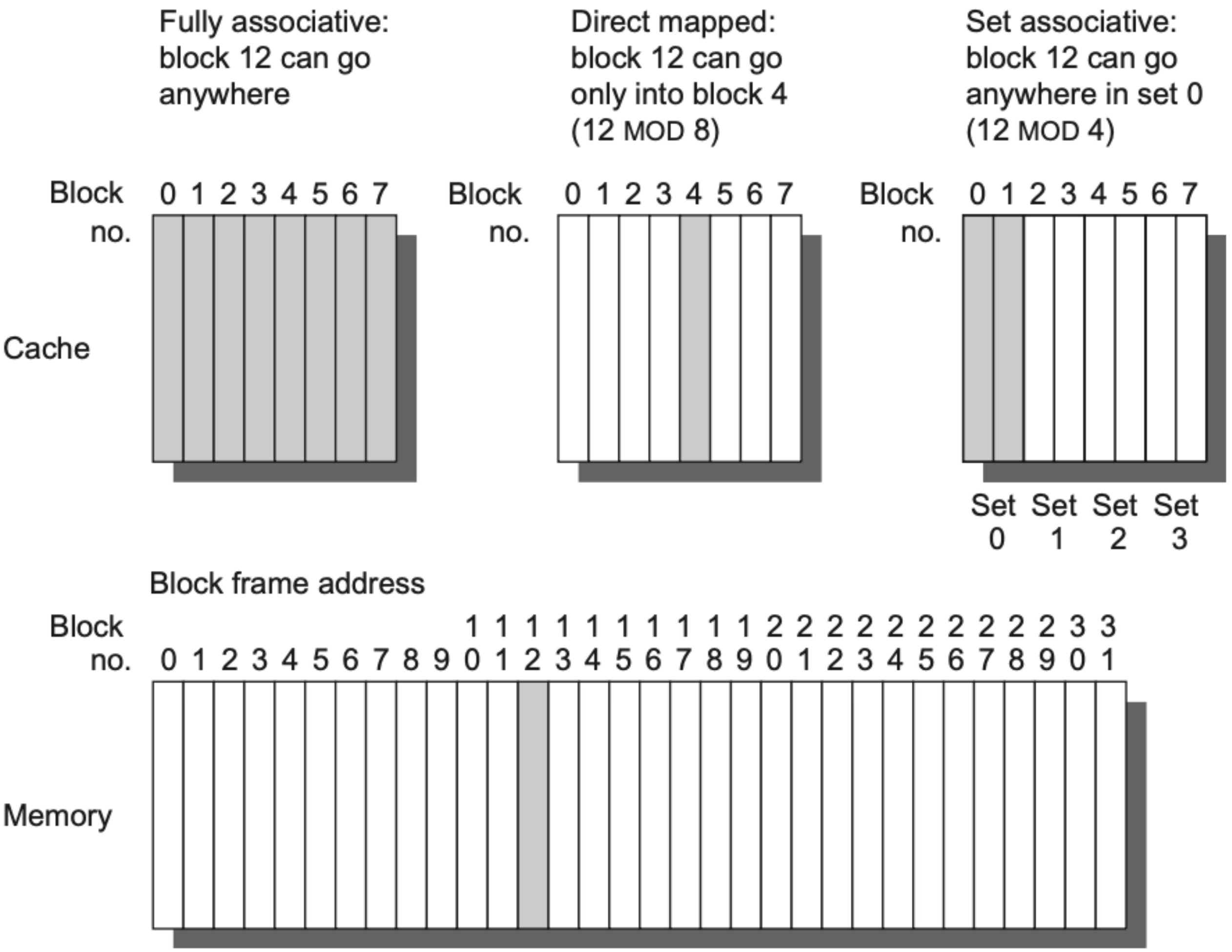
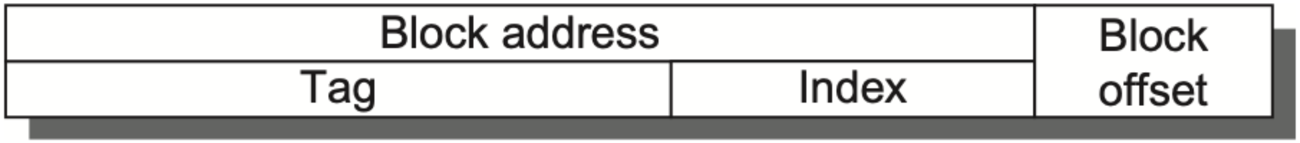


Figure B.2 This example cache has eight block frames and memory has 32 blocks.

Block Identification

Refresh



Block address

Tag

Index

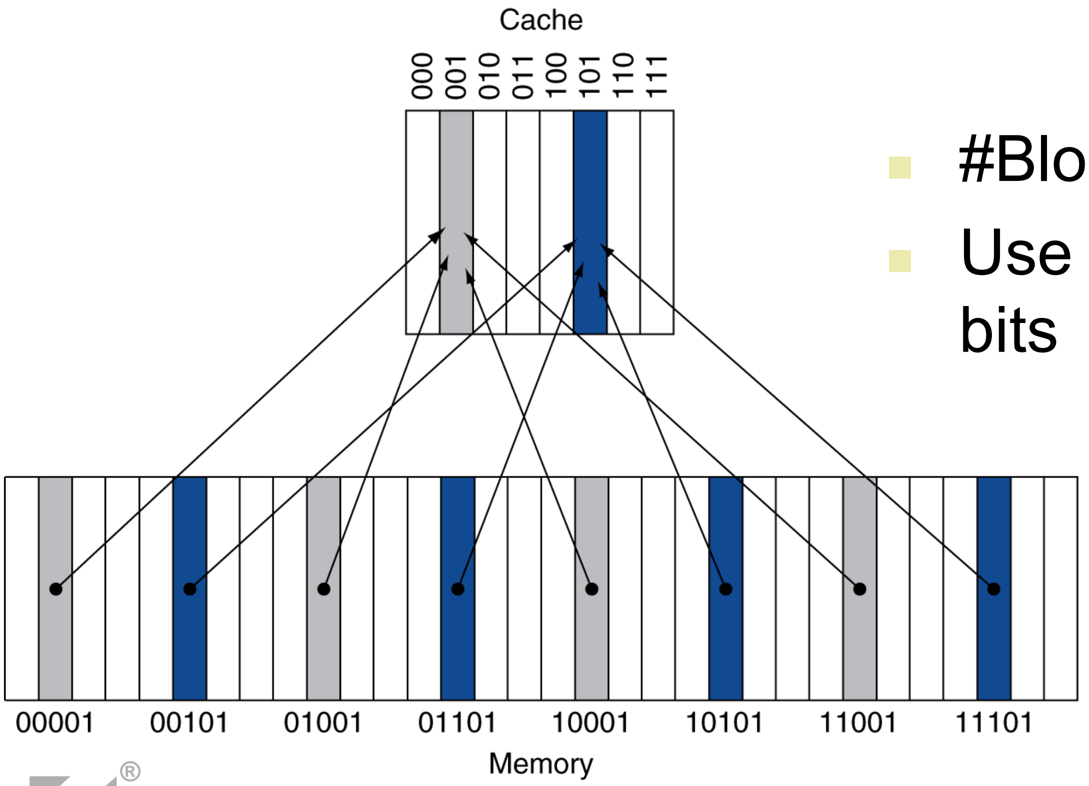
Block offset

Figure B.3 The three portions of an address in a set associative or direct-mapped cache. The tag is used to check all the blocks in the set, and the index is used to select the set. The block offset is the address of the desired data within the block. Fully associative caches have no index field.

Direct Mapped Cache

Refresh

- Location determined by address
- Direct mapped: only one choice
  - (Block address) modulo (#Blocks in cache)



- #Blocks is a power of 2
- Use low-order address bits

Associative Caches

Refresh

- Fully associative
  - Allow a given block to go in any cache entry
  - Requires all entries to be searched at once
  - Comparator per entry (expensive)
- *n*-way set associative
  - Each set contains *n* entries
  - Block number determines which set
    - (Block number) modulo (#Sets in cache)
  - Search all entries in a given set at once
  - *n* comparators (less expensive)

Block Replacement

Refresh

- Direct mapped: no choice
- Fully or set-associative
  - Prefer non-valid entry, if there is one
  - Otherwise, choose among entries in the set
- Least-recently used (LRU)
  - Choose the one unused for the longest time
    - Simple for 2-way, manageable for 4-way, too hard beyond that
- Random
  - Gives approximately the same performance as LRU for high associativity

Write strategy

Refresh

- Write-through
  - Update both upper and lower levels
  - Simplifies replacement, but may require write buffer
- Write-back
  - Update upper level only
  - Update lower level when block is replaced
  - Need to keep more state
- Virtual memory
  - Only write-back is feasible, given disk write latency

Memory Hierarchy Design

- Memory hierarchy design becomes more crucial with recent multi-core processors:
  - Aggregate peak bandwidth grows with # cores:
    - Intel Core i7 can generate two references per core per clock
    - Four cores and 3.2 GHz clock
      - 25.6 billion 64-bit data references/second +
      - 12.8 billion 128-bit instruction references/sec
      - = 409.6 GB/s!

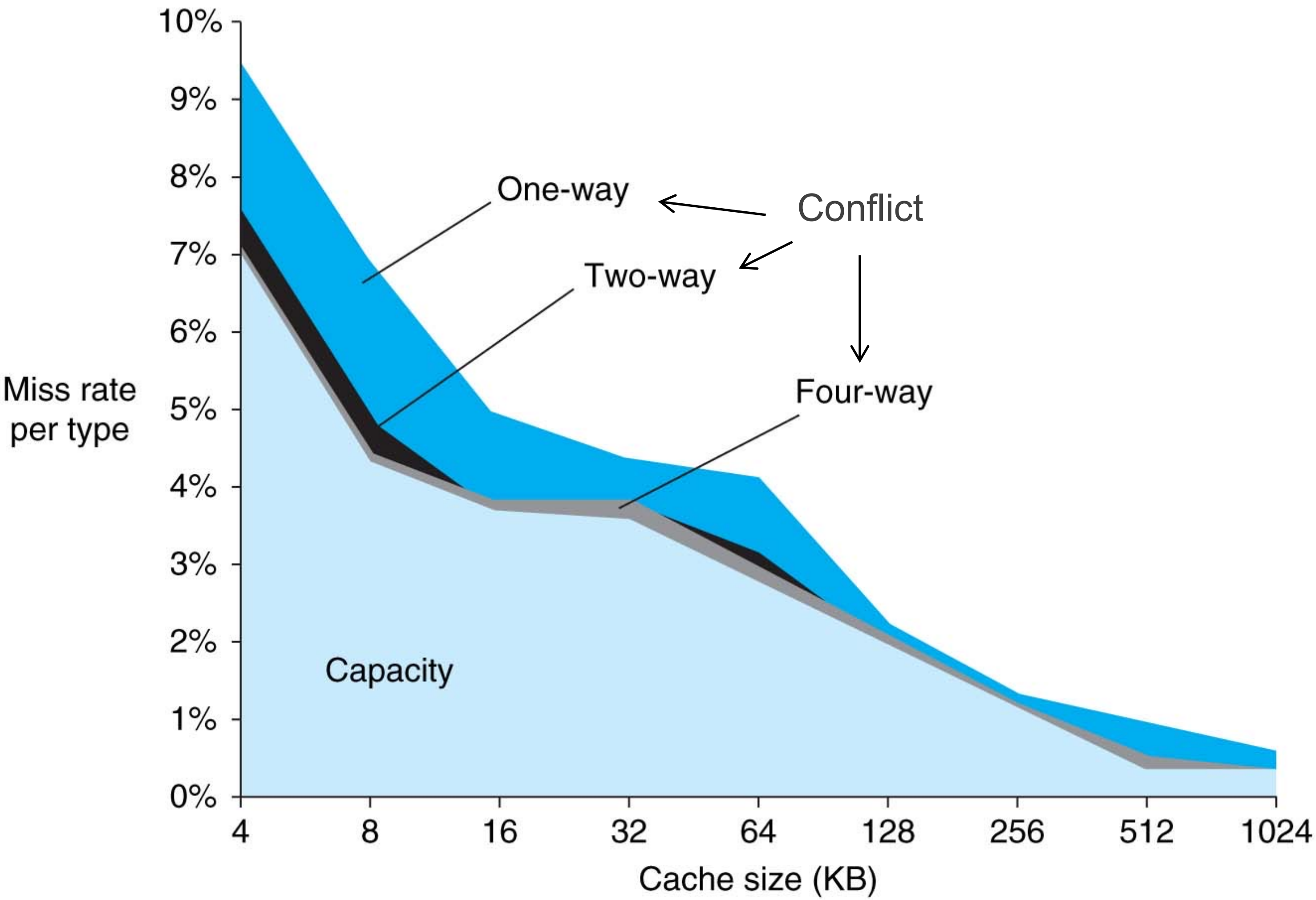
      See exercise slides ahead
    - DRAM bandwidth is only 8% of this (34.1 GB/s)
    - Requires:
      - Multi-port, pipelined caches
      - Two levels of cache per core
      - Shared third-level cache on chip

      See exercise slides ahead

Memory Hierarchy Basics

- Miss rate
  - Fraction of cache access that result in a miss
- Causes of misses
  - Compulsory
    - First reference to a block
  - Capacity
    - Blocks discarded and later retrieved
  - Conflict
    - Program makes repeated references to multiple addresses from different blocks that map to the same location in the cache

The 3C's in different cache sizes

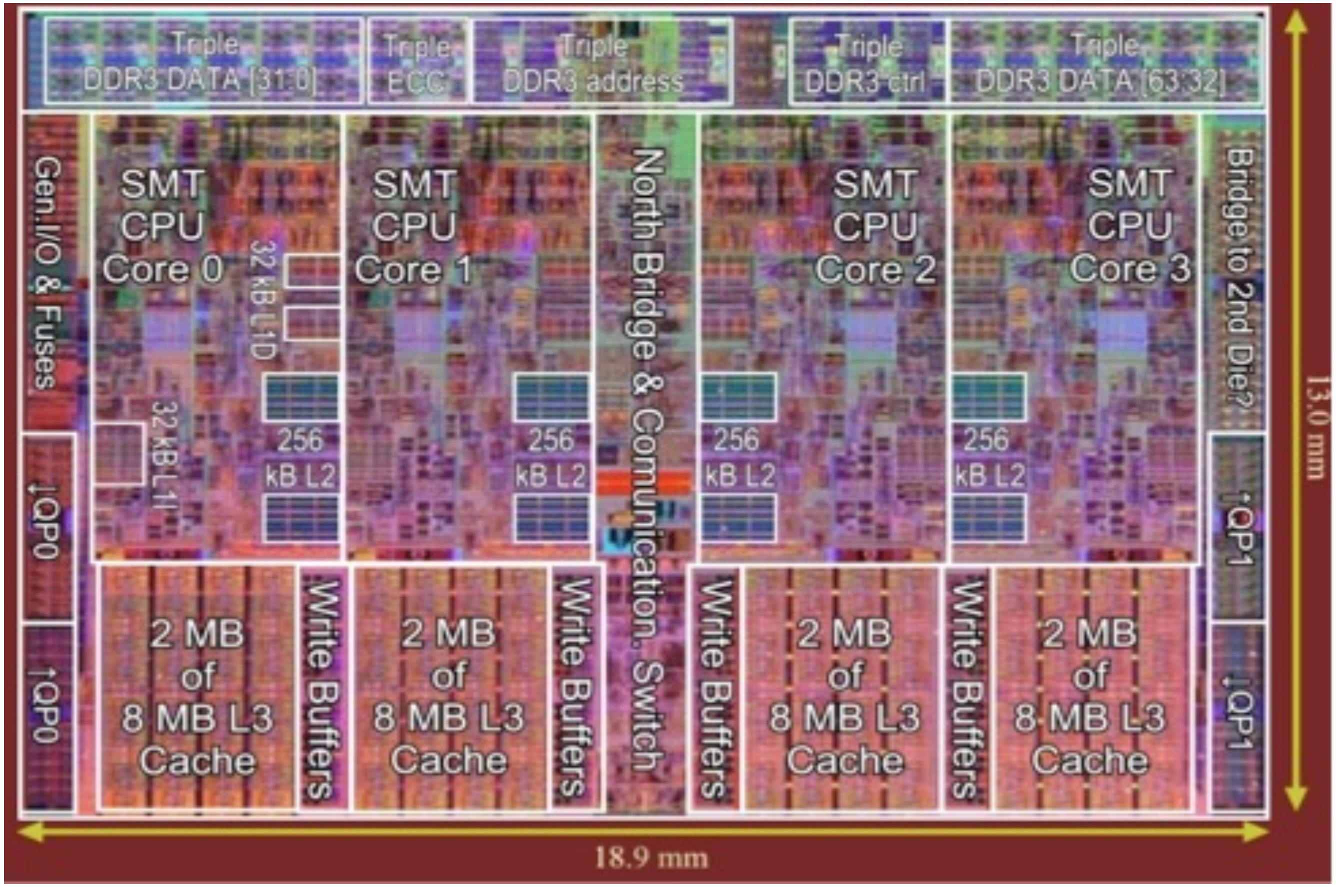


Multilevel Caches

- Primary cache attached to CPU
  - Small, but fast
- Level-2 cache services misses from L-1 cache
  - Larger, slower, but still faster than L3 or main memory
- L-3 cache or main memory services L-2 cache misses
  - Larger, slower, but still faster than main memory
- Main memory services L-3 cache misses
  (and eventually L-2 cache misses, if L-3 is non-inclusive)

Multilevel On-Chip Caches

Intel Nehalem 4-core processor



Per core: 32KB L1 I-cache, 32KB L1 D-cache, 512KB L2 cache

3-Level Cache Organization

	Intel Nehalem	AMD Opteron X4
L1 caches (per core)	L1 I-cache: 32KB, 64-byte blocks, 4-way, approx LRU replacement, hit time n/a L1 D-cache: 32KB, 64-byte blocks, 8-way, approx LRU replacement, write-back/allocate, hit time n/a	L1 I-cache: 32KB, 64-byte blocks, 2-way, approx LRU replacement, hit time 3 cycles L1 D-cache: 32KB, 64-byte blocks, 2-way, approx LRU replacement, write-back/allocate, hit time 9 cycles
L2 unified cache (per core)	256KB, 64-byte blocks, 8-way, approx LRU replacement, write-back/allocate, hit time n/a	512KB, 64-byte blocks, 16-way, approx LRU replacement, write-back/allocate, hit time n/a
L3 unified cache (shared)	8MB, 64-byte blocks, 16-way, replacement n/a, write-back/allocate, hit time n/a	2MB, 64-byte blocks, 32-way, replace block shared by fewest cores, write-back/allocate, hit time 32 cycles

n/a: data not available

Performance in multilevel caches

$$\text{CPU}_{\text{exec-time}} = (\text{CPU}_{\text{clock-cycles}} + \text{Mem}_{\text{stall-cycles}}) \times \text{Clock cycle time}$$

$$\text{CPU}_{\text{exec-time}} = (\text{IC} \times \text{CPI}_{\text{CPU}} + \text{Mem}_{\text{stall-cycles}}/\text{Instr}) \times \text{Clock cycle time}$$

With introduction of a single-level cache:

$$\text{Mem}_{\text{stall-cycles}} = \text{IC} \times \ldots \text{Miss rate} \ldots \text{Mem accesses} \ldots \text{Miss penalty}\ldots$$

$$\text{Mem}_{\text{stall-cycles}} = \text{IC} \times \text{Misses}/\text{Instruction} \times \text{Miss Penalty}$$

$$\frac{\text{Misses}}{\text{Instruction}} = \frac{\text{Miss rate} \times \text{Memory accesses}}{\text{Instruction count}} = \text{Miss rate} \times \frac{\text{Memory accesses}}{\text{Instruction}}$$

For each additional cache-level `i` (including LLC to memory):

$$\text{Mem\_accesses}_{\text{level\_i}} = \text{Misses/Instruction}_{\text{level\_i-1}}$$

$$\text{Miss\_penalty}_{\text{level\_i}} = (\text{Hit\_rate} \times \text{Hit\_time} + \text{Miss\_rate} \times \text{Miss\_penalty})_{\text{level\_i+1}}$$

Miss rates in multilevel caches

- **Local miss rate**
  - the number of misses in a cache divided by the total number of memory accesses to this cache
  - for the first-level cache it is equal to **Miss rate$_{L1}$**, and for the second-level cache it is **Miss rate$_{L2}$**
- **Global miss rate**
  - the number of misses in the cache divided by the total number of memory accesses generated by the PU
  - for the first-level cache is still just **Miss rate$_{L1}$**, but for the second-level cache it is **Miss rate$_{L1}$ × Miss rate$_{L2}$**

Exercise 1 on memory hierarchy

Consider the following study case:

- execution of a piece of code in the SeARCH node with the Xeon Skylake (Gold 6130); detailed info on this Intel microarchitetcure in https://en.wikichip.org/wiki/intel/microarchitectures/skylake_(server) ;
- execution of the same 2 instructions (that are already in the instruction cache) in all cores of a single chip: load a double in register, followed by a multiplication by another double in a memory distant location;
- the Skylake cores are 6-way superscalar and each core has 2 load units;
- these instructions are executed with a cold data cache.

Compute:

a) **The** max required bandwidth to access the external RAM when executing these 2 instructions (to simplify, consider clock rate = 2 GHz).

b) **The** aggregate peak bandwidth available in this Xeon device to access the installed DRAM-4 (using all memory channels).

Exercise 2 on memory hierarchy

Similar to problem 1 (same node/chip in the cluster), but consider now:

- execution of code taking advantage of the AVX-512 facilities;
- execution of the same 2 vector instructions (that are already in the instruction cache) in all cores: load in register a vector of doubles followed by a multiplication by another vector of doubles in memory;
- the Skylake cores are 6-way superscalar and 2-way MT, and each core supports 2 simultaneous vector loads;
- the Skylake 6130 base clock rate with **AVX-512** code is **1.3 GHz**;
- these instructions are executed with a cold data cache.

Compute/estimate:

The max required bandwidth to access the external RAM when executing these 2 vector instructions.
Compare with the peak bandwidth computed before.

* https://en.wikichip.org/wiki/intel/xeon_gold/6130

Exercise 3 on Cache Performance

- Given
  - I-cache miss rate = 2%
  - D-cache miss rate = 4%
  - Miss penalty = 100 cycles
  - Base CPI (ideal cache) = 2
  - Load & stores are 36% of instructions
- Miss cycles per instruction
  - I-cache: ?? x ?? = ??
  - D-cache: ?? x ?? x ?? = ??
- Actual CPI = 2 + ?? + ?? = **??**

Exercise 4 on Multilevel Cache

- Given
  - CPU base **CPI = 1**, clock rate = 4GHz
  - Miss rate/instruction = 2%
  - Main memory access time = 100ns
- With just primary cache
  - Miss penalty = 100ns/0.25ns = 400 cycles
  - Effective **CPI = 9** (= 1 + 0.02 × 400)
- Now add L-2 cache …
  - Access time = 5ns
  - Global miss rate to main memory = 0.5%
- CPI = 1 + ?? × ?? + ?? × ?? = **??**
- Performance ratio = 9 / ?? = **??**

Exercise 5 on multilevel performance

Characterize the memory system of Xeon Skylake Gold 6130:

1. **L1 I-cache**
   - size ? KiB/core, ?-way set associative, ? sets, line size ? B, hit time ? cycles, ? B/cycle on transfer bandwidth L1 to the instruction fetch unit

   L1 D-cache
   - size ? KiB/core, ?-way set associative, ? sets, line size ? B, hit time ? cycles, ? B/cycle on load bandwidth L1 to load buffer unit
2. **L2 cache**
   - size ? KiB/core, ?-way set associative, ? sets, line size ? B, hit time ? cycles, ? B/cycle on load bandwidth L2 to L1
3. **L3 cache**
   - size ? KiB/core, ?-way set associative, ? sets, line size ? B, hit time ? cycles, ? B/cycle on load bandwidth L3 to L2
4. **DRAM**, DDR4-2666
   - up to ? GT/s, bandwidth ? GB/s per channel, ? mem channels, aggregate bandwidth ? GB/s , ? B/cycle on peak load bandwidth DRAM to L3, NUMA-local latency ? ns, NUMA-remote latency ? ns

Exercise 6 on multilevel performance

Similar to problem 1 (same node/chip in the cluster, code), but consider now:

- execution of scalar code in a **2 GHz** single-core (already in L1 I-cache);
- code already takes advantage of all data cache levels (**L1, L2 & L3**), where 50% of data is placed on the RAM modules in the memory channels of the other PU chip (**NUMA architecture**);
- remember: the Skylake cores are **6-way superscalar** and **2-way MT**, and each core supports **2 simultaneous loads**;
- **cache latency time on hit**: take the average of the specified values;
- **memory latency**: 80 nsec (**NUMA local**), 120 nsec (**NUMA remote**);
- **miss rate per instruction** :
  - at **L1: 2%;** at **L2: 50%;** at **L3: 80%** (these are not global values!).

Compute/estimate:

1. **The miss penalty** per instruction at each cache level.
2. **The average memory stall cycles** per instruction that degrades CPI.

Refresh

Hierarquia da memória - Desempenho

$$T_{exec} = \# I * CPI * T_{cc} \quad ou \quad T_{exec} = \# CC * T_{cc}$$

Como é que a hierarquia de memória influencia Texec?

Cada acesso à memória vai originar ciclos adicionais na execução do programa (#CC$_{MEM}$) devido aos *misses*:

$$T_{exec} = (\# CC_{CPU} + \# CC_{MEM}) * T_{cc}$$

Cada *miss* implica um aumento do #CC em *miss penalty* ciclos, logo:

$$\# CC_{MEM} = n^{o}\, miss * miss\ penalty$$

miss rate * nº acessos à memoria

Refresh

Hierarquia da memória - Desempenho

$$T_{exec} = (\#CC_{CPU} + \#CC_{MEM}) * T_{cc}$$

$$\#CC = \#I * CPI$$

$$T_{exec} = \#I * (CPI_{CPU} + CPI_{MEM}) * T_{cc}$$

CPI$_{\text{CPU}}$ – nº de ciclos que o processador necessita, em média, para executar cada instrução;

O *hit time* considera-se incluído no CPI$_{\text{CPU}}$

CPI$_{\text{MEM}}$ – nº de ciclos que o processador pára, em média, à espera de dados da memória, porque não encontrou estes dados na cache.

Estes são vulgarmente designados por ***memory stall cycles*** ou ***wait states***.

Refresh

Hierarquia da memória - Desempenho

$$CPI_{MEM} = \%acessosMem * missrate * misspenalty$$

Os acessos à memória devem-se a:

- acesso a **dados** (instruções de *Load* e *Store* do programa)
- busca de **instruções**

Como estes têm comportamentos diferentes usam-se diferentes percentagens:

- **Dados –** Apenas uma determinada percentagem de instruções acede à memória (%Mem). ***missrate$_D$*** refere-se ao acesso a dados.
- **Instruções** – Todas as instruções são lidas da memória, logo a % de acesso à memória é de 100%. ***missrate$_I$*** refere-se ao acesso às instruções.

missrate$_I$ é geralmente menor que ***missrate$_D$*** devido à localidade espacial.

$$CPI_{MEM} = (missrate_I + \%Mem * missrate_D) * misspenalty$$

Refresh

Hierarquia da memória - Desempenho

Abreviando *missrate* por *mr* e *misspenalty* por *mp* temos

$$T_{exec} = \# I * (CPI_{CPU} + CPI_{MEM}) * T_{cc}$$

$$CPI_{MEM} = (mr_I + \% Mem * mr_D) * mp$$

substituindo

$$T_{exec} = \# I * [CPI_{CPU} + (mr_I + \% Mem * mr_D) * mp] * T_{cc}$$

NOTA: A *miss penalty (mp)* tem que ser expressa em ciclos do *relógio*.

Refresh

Hierarquia da memória - Desempenho

Considere uma máquina com uma *miss rate* de 4% para instruções, 5% para dados e uma *miss penalty* de 50 ciclos. Assuma ainda que 40% das instruções são *loads* ou *stores*, e que o $\mathrm{CPI}_{\mathrm{CPU}}$ é 1. Qual o CPI total?

$$CPI = CPI_{CPU} + CPI_{MEM} = CPI_{CPU} + (mr_I + \%Mem * mr_D) * mp$$

$$CPI = 1 + (0.04 + 0.4 * 0.05) * 50 = 1 + 3 = 4$$

Se a frequência do relógio for de 1 GHz e o programa executar 10^9 instruções qual o tempo de execução?

$$T_{exec} = \# I * CPI * T_{cc} = 10^9 * 4 * \frac{1}{10^9} = 4s$$

Refresh

Hierarquia da memória - Desempenho

Considere um programa com as características apresentadas na tabela, a executar numa máquina com memória de tempo de acesso 0. Se a frequência do processador for 1 GHz, qual o CPI médio e o tempo de execução?

Instrução	Nº Instruções	CPI
Cálculo	$3*10^8$	1,1
Acesso à Mem.	$6*10^8$	2,5
Salto	$1*10^8$	1,7
TOTAL:	10^9	

$$CPI = CPI_{CPU} + CPI_{MEM} = (3*1.1 + 6*2.5 + 1*1.7)/10 + 0 = 2$$

$$T_{exec} = \# I * CPI * T_{cc} = 10^9 * 2 * \frac{1}{10^9} = 2s$$

Refresh

Hierarquia da memória - Desempenho

Considere o mesmo programa e máquina do acetato anterior, mas agora com um tempo de acesso à memória de 10 ns (por palavra ou instrução). Suponha ainda que esta máquina não tem cache. Qual o CPI efectivo e T_{exec}?

$$CPI = CPI_{CPU} + CPI_{MEM} = CPI_{CPU} + (mr_I + \%Mem * mr_D) * mp$$

Se a máquina não tem cache, então $mr_I = mr_D = 100\%$.

Da tabela verificamos que *%Mem* = 60% ($6x10^8/10x10^8$).

mp expresso em ciclos do relógio é 10/1 = 10 ciclos (f=1 GHz)

$$CPI = CPI_{CPU} + CPI_{MEM} = 2 + (1 + 0.6 * 1) * 10 = 2 + 16 = 18$$

$$T_{exec} = \# I * CPI * T_{cc} = 10^9 * 18 * \frac{1}{10^9} = 18s$$

Refresh

Hierarquia da memória - Desempenho

Considere agora que existe uma *cache* com linhas de 4 palavras; a *miss rate* de acesso às instruções é de 6% e de acesso aos dados é de 10%; o tempo de acesso à memória central é constituído por uma latência de 40 ns mais 10 ns por palavra. Qual o CPI médio e o tempo de execução?

mp = 40 + 10*4 = 80 ns ; em ciclos *mp* = 80/1 = 80 ciclos

$$CPI = CPI_{CPU} + CPI_{MEM} = 2 + (0.06 + 0.6 * 0.1) * 80 = 2 + 9.6 = 11.6$$

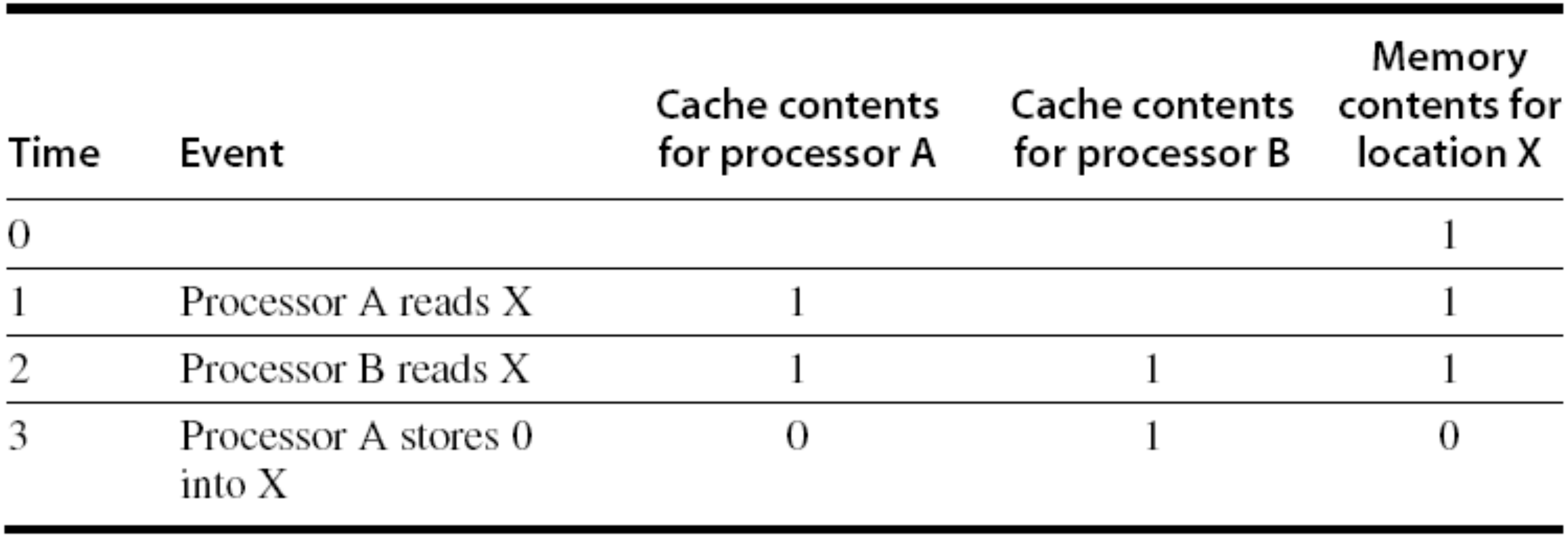
$$T_{exec} = \# I * CPI * T_{cc} = 10^9 * 11.6 * \frac{1}{10^9} = 11.6s$$

Memory Hierarchy Basics

- Miss rate
  - Fraction of cache access that result in a miss
- Causes of misses (3C's +1)
  - Compulsory
    - First reference to a block
  - Capacity
    - Blocks discarded and later retrieved
  - Conflict
    - Program makes repeated references to multiple addresses from different blocks that map to the same location in the cache
  - **Coherency**
    - Different processors should see same value in same location
    - Two sources: true-sharing and false-sharing *(section 5.3)*

The cache coherence pb

- Processors may see different values through their caches:



Time	Event	Cache contents for processor A	Cache contents for processor B	Memory contents for location X
0				1
1	Processor A reads X	1		1
2	Processor B reads X	1	1	1
3	Processor A stores 0 into X	0	1	0

Cache Coherence

- Coherence
  - All reads by any processor must return the most recently written value
  - Writes to the same location by any two processors are seen in the same order by all processors

    (Coherence defines the behaviour of reads & writes to the same memory location)

- Consistency
  - When a written value will be returned by a read
  - If a processor writes location A followed by location B, any processor that sees the new value of B must also see the new value of A

    (Consistency defines the behaviour of reads & writes with respect to accesses to other memory locations)

Enforcing Coherence

- Coherent caches provide:
  - *Migration*: movement of data
  - *Replication*: multiple copies of data
- Cache coherence protocols
  - Directory based
    - Sharing status of each block kept in one location
  - Snooping
    - Each core tracks sharing status of each block

Memory Hierarchy Basics

- Six basic cache optimizations:
  - Larger block size
    - Reduces compulsory misses
    - Increases capacity and conflict misses, increases miss penalty
  - Larger total cache capacity to reduce miss rate
    - Increases hit time, increases power consumption
  - Higher associativity
    - Reduces conflict misses
    - Increases hit time, increases power consumption
  - Higher number of cache levels
    - Reduces overall memory access time
  - Giving priority to read misses over writes
    - Reduces miss penalty
  - Avoiding address translation in cache indexing
    - Reduces hit time

Memory Technology and Optimizations

- Performance metrics
  - Latency is concern of cache
  - Bandwidth is concern of multiprocessors and I/O
  - Access time
    - Time between read request and when desired word arrives
  - Cycle time
    - Minimum time between unrelated requests to memory

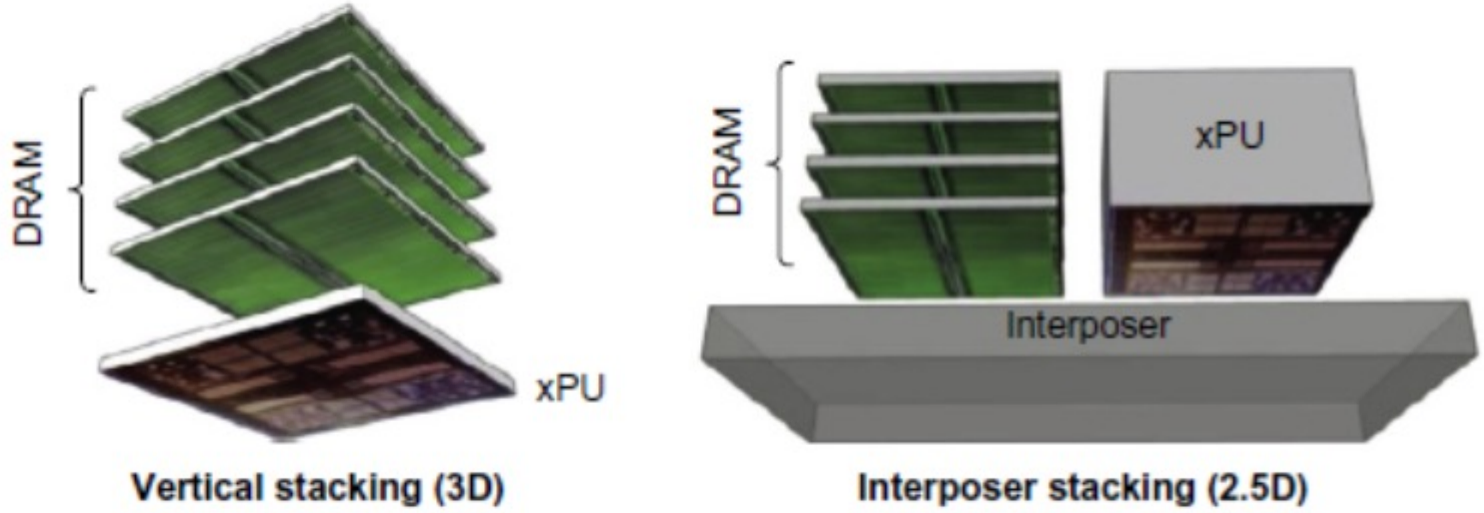
- SRAM memory has low latency, use for cache
- Organize DRAM chips into many banks for high bandwidth, use for main memory

Memory Technology

- SRAM
  - Requires low power to retain bit
  - Requires 6 transistors/bit
- DRAM
  - Must be re-written after being read
  - Must also be periodically refreshed
    - Every ~ 8 ms (roughly 5% of time)
    - Each row can be refreshed simultaneously
  - One transistor/bit
  - Address lines are multiplexed:
    - Upper half of address: row access strobe (RAS)
    - Lower half of address: column access strobe (CAS)

Stacked/Embedded DRAMs

- Stacked DRAMs in same package as processor
  - High Bandwidth Memory (HBM)



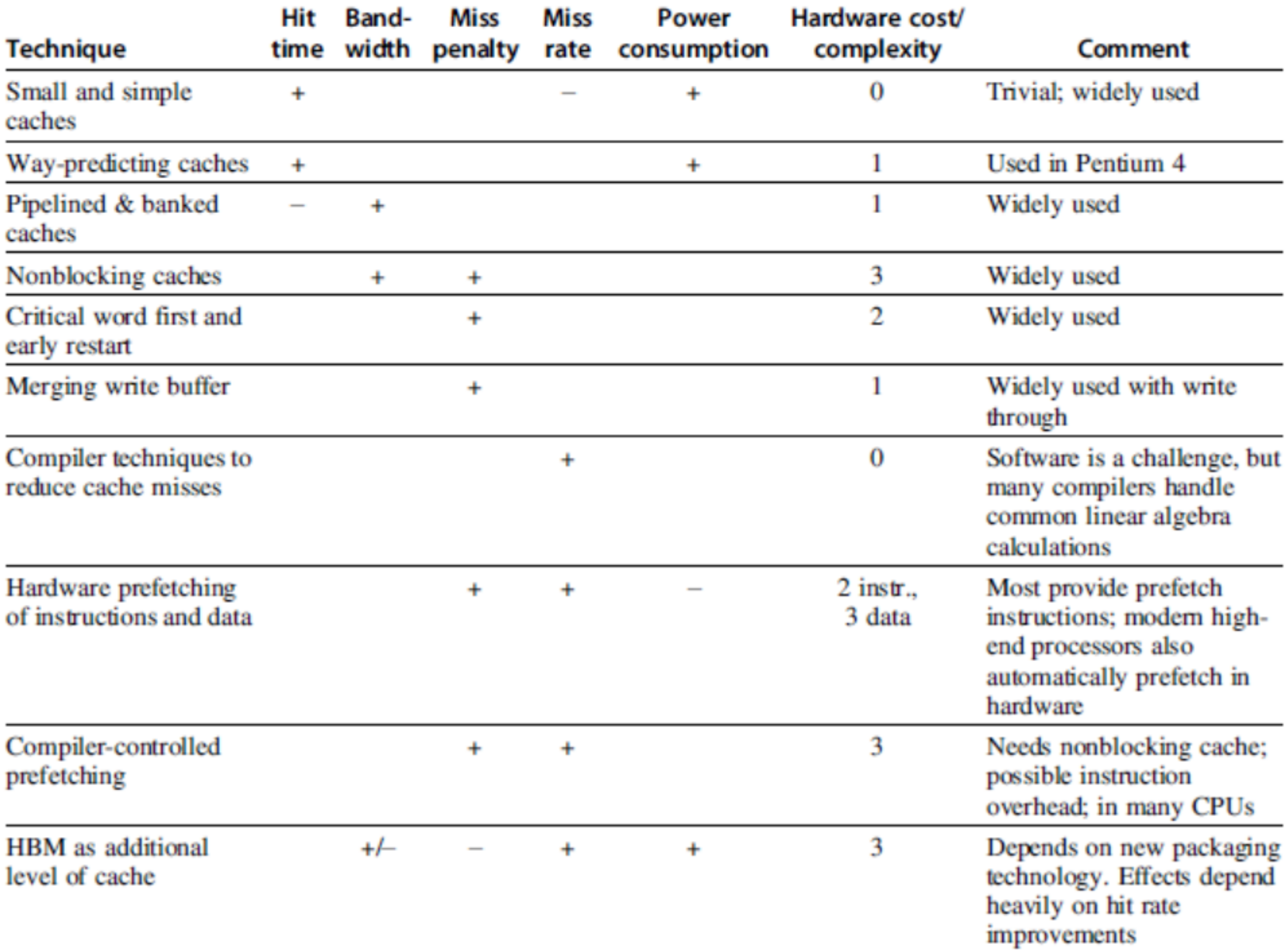
Flash Memory

- Type of EEPROM
- Types: NAND (denser) and NOR (faster)
- NAND Flash:
  - Reads are sequential, reads entire page (.5 to 4 KiB)
  - 25 us for first byte, 40 MiB/s for subsequent bytes
  - SDRAM: 40 ns for first byte, 4.8 GB/s for subsequent bytes
  - 2 KiB transfer: 75 uS vs 500 ns for SDRAM, 150X slower
  - 300 to 500X faster than magnetic disk

Advanced Optimizations

- Reduce hit time
  - Small and simple first-level caches
  - Way prediction
- Increase bandwidth
  - Pipelined caches, multibanked caches, non-blocking caches
- Reduce miss penalty
  - Critical word first, merging write buffers
- Reduce miss rate
  - Compiler optimizations
- Reduce miss penalty or miss rate via parallelization
  - Hardware or compiler prefetching

Summary



Technique	Hit time	Band-width	Miss penalty	Miss rate	Power consumption	Hardware cost/ complexity	Comment
Small and simple caches	+			–	+	0	Trivial; widely used
Way-predicting caches	+				+	1	Used in Pentium 4
Pipelined & banked caches	–	+				1	Widely used
Nonblocking caches		+	+			3	Widely used
Critical word first and early restart			+			2	Widely used
Merging write buffer			+			1	Widely used with write through
Compiler techniques to reduce cache misses				+		0	Software is a challenge, but many compilers handle common linear algebra calculations
Hardware prefetching of instructions and data			+	+	–	2 instr., 3 data	Most provide prefetch instructions; modern high-end processors also automatically prefetch in hardware
Compiler-controlled prefetching			+	+		3	Needs nonblocking cache; possible instruction overhead; in many CPUs
HBM as additional level of cache		+/–	–	+	+	3	Depends on new packaging technology. Effects depend heavily on hit rate improvements