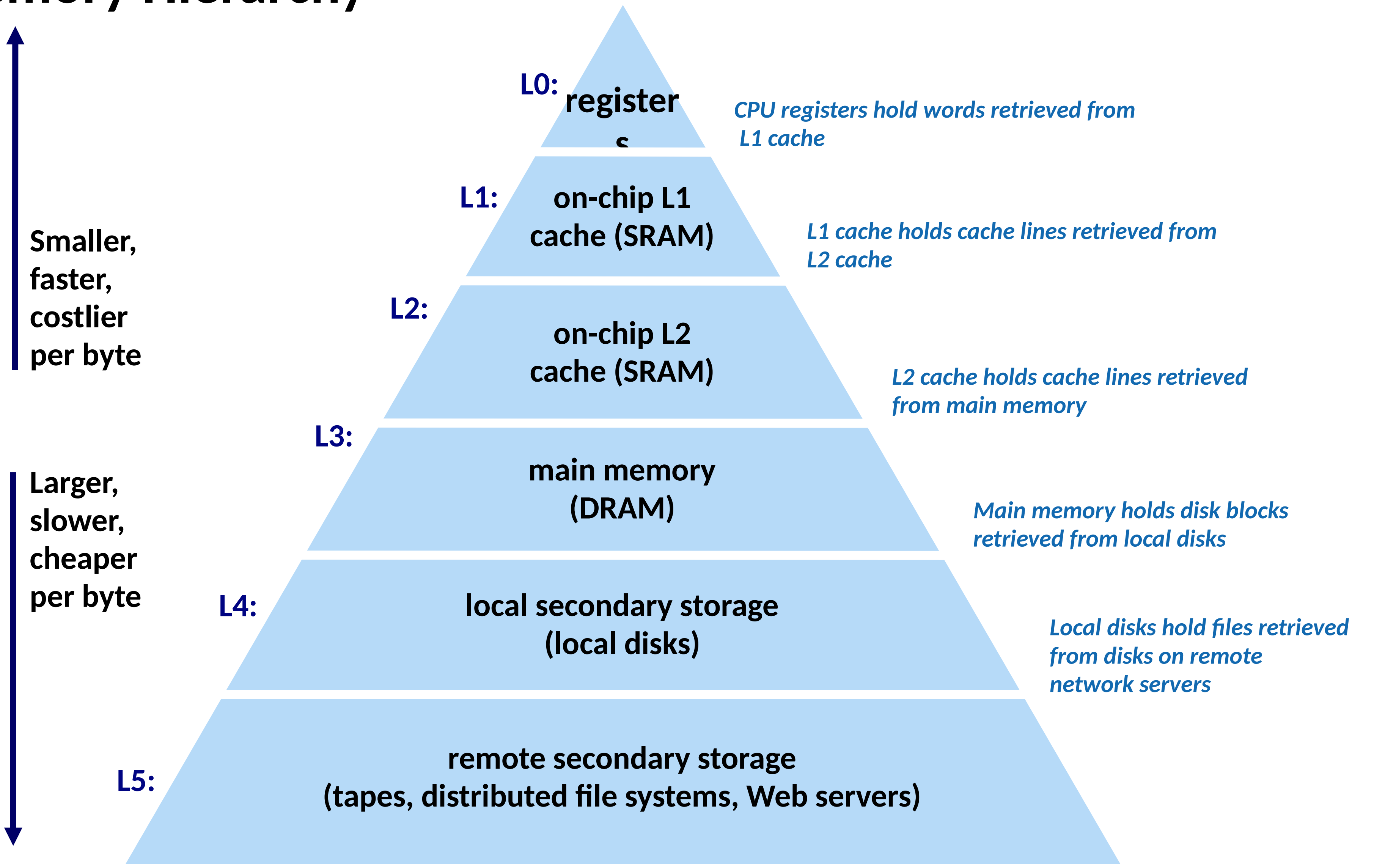
TIMO SCHNEIDER <TIMOS@INF.ETHZ.CH>

DPHPC: Caches

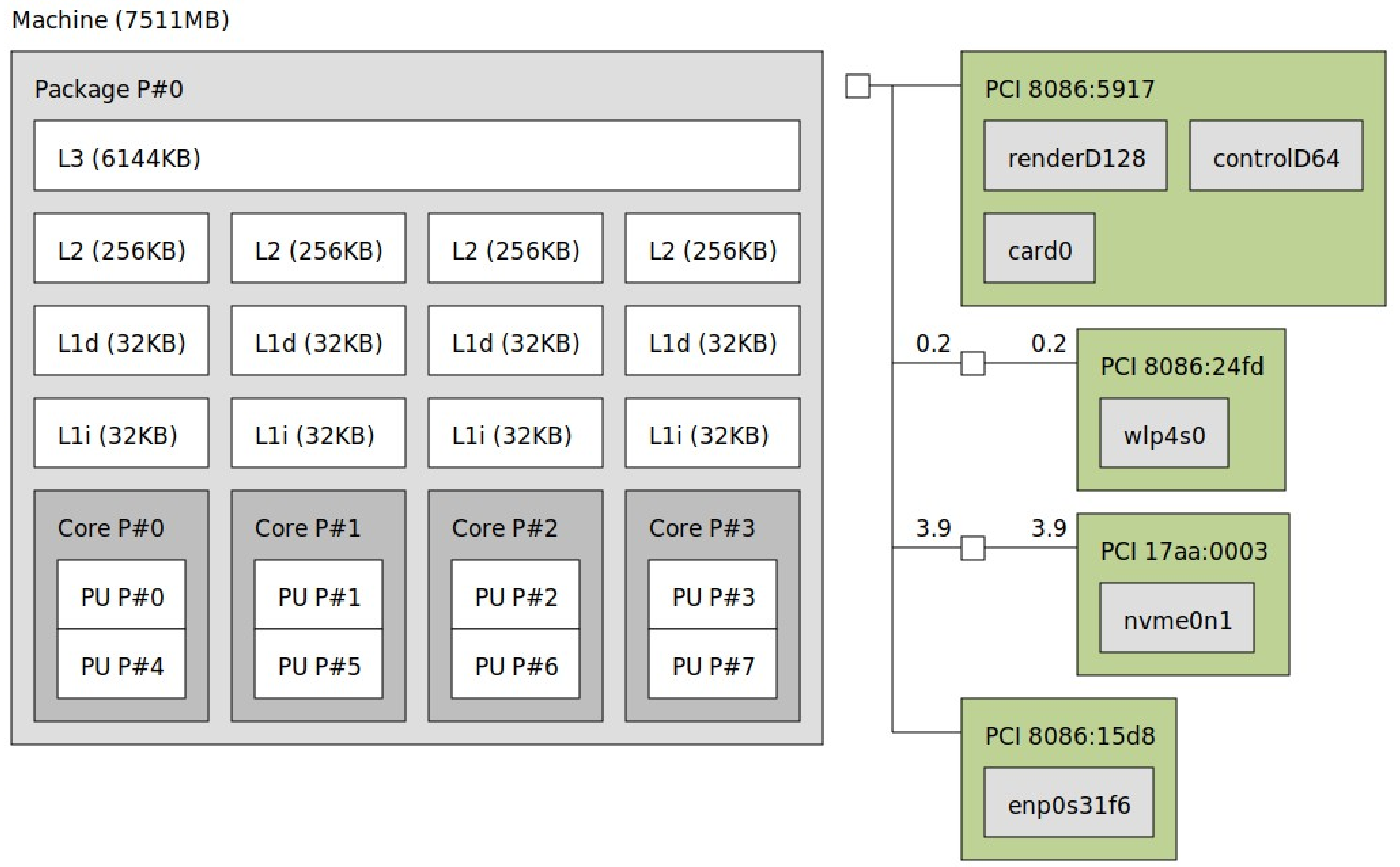
Recitation session

Systems@ETH Zürich

Typical Memory Hierarchy


Smaller,
faster,
costlier
per byte
Larger,
slower,
cheaper
per byte
L0:
registers
CPU registers hold words retrieved from L1 cache
L1:
on-chip L1 cache (SRAM)
L1 cache holds cache lines retrieved from L2 cache
L2:
on-chip L2 cache (SRAM)
L2 cache holds cache lines retrieved from main memory
L3:
main memory (DRAM)
Main memory holds disk blocks retrieved from local disks
L4:
local secondary storage (local disks)
Local disks hold files retrieved from disks on remote network servers
L5:
remote secondary storage (tapes, distributed file systems, Web servers)

In practice (hwloc-ls)



Host: dirac

Indexes: physical

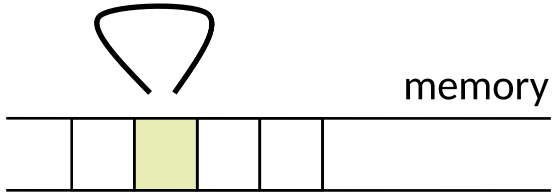
Date: Thu 26 Sep 2019 11:37:08 AM CEST

Why Caches Work: Locality

- ***Locality:*** **Programs tend to use data and instructions with addresses near or equal to those they have used recently, cf. "Denning: "The locality principle", CACM'05**

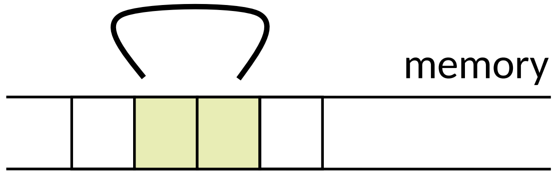
- ***Temporal locality:***

  Recently referenced items are likely
  to be referenced again in the near future



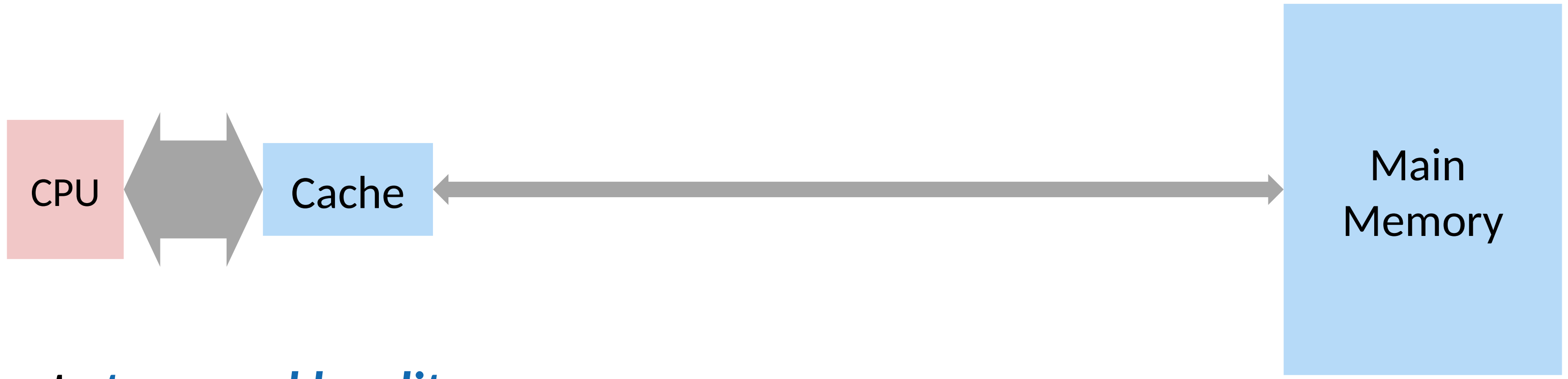
- ***Spatial locality:***

  Items with nearby addresses tend
  to be referenced close together in time



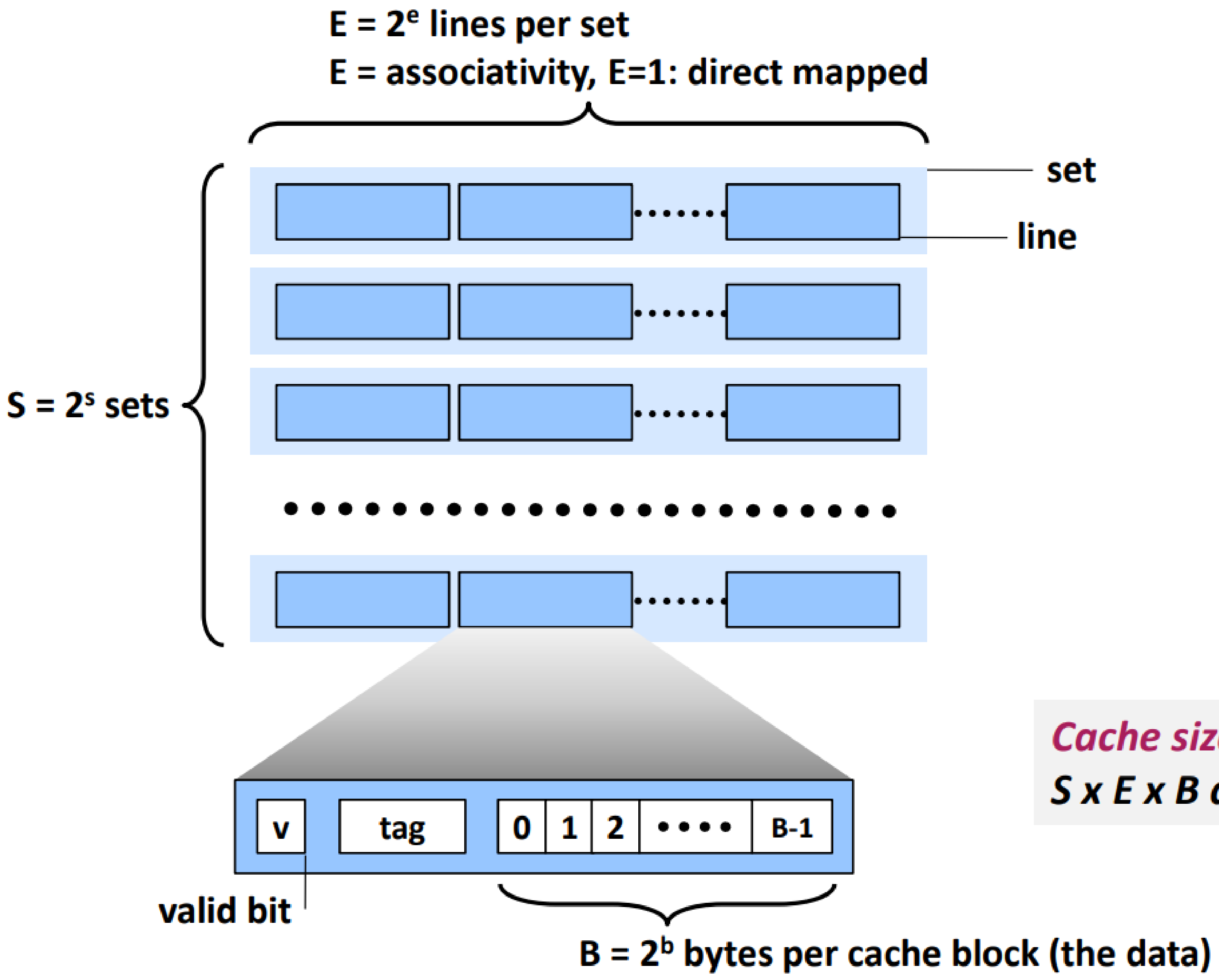
Cache

- *Definition:* **Computer memory with short access time used for the storage of frequently or recently used instructions or data**



- **Naturally supports *temporal locality***
- ***Spatial locality* is supported by transferring data in blocks**
  - E.g., Intel's Core family: one block = 64 B = 8 doubles

General Cache Organization (S, E, B)


$E = 2^e$ lines per set
E = associativity, E=1: direct mapped
set
line
$S = 2^s$ sets
v
tag
0
1
2
B-1
valid bit
$B = 2^b$ bytes per cache block (the data)

Cache size:
S x E x B data bytes

Terminology

- **Direct mapped cache:**
  - Cache with E = 1
  - Means every block from memory has a unique location in cache
- **Fully associative cache**
  - Cache with S = 1 (i.e., maximal E)
  - Means every block from memory can be mapped to any location in cache
  - In practice to expensive to build
  - One can view the register file as a fully associative cache
- **LRU (least recently used) replacement**
  - when selecting which block should be replaced (happens only for E > 1), the least recently used one is chosen

Types of Cache Misses (The 3 C's)

- ***Compulsory (cold)*** **miss**

  Occurs on first access to a block

- ***Capacity*** **miss**

  Occurs when working set is larger than the cache

- ***Conflict*** **miss**

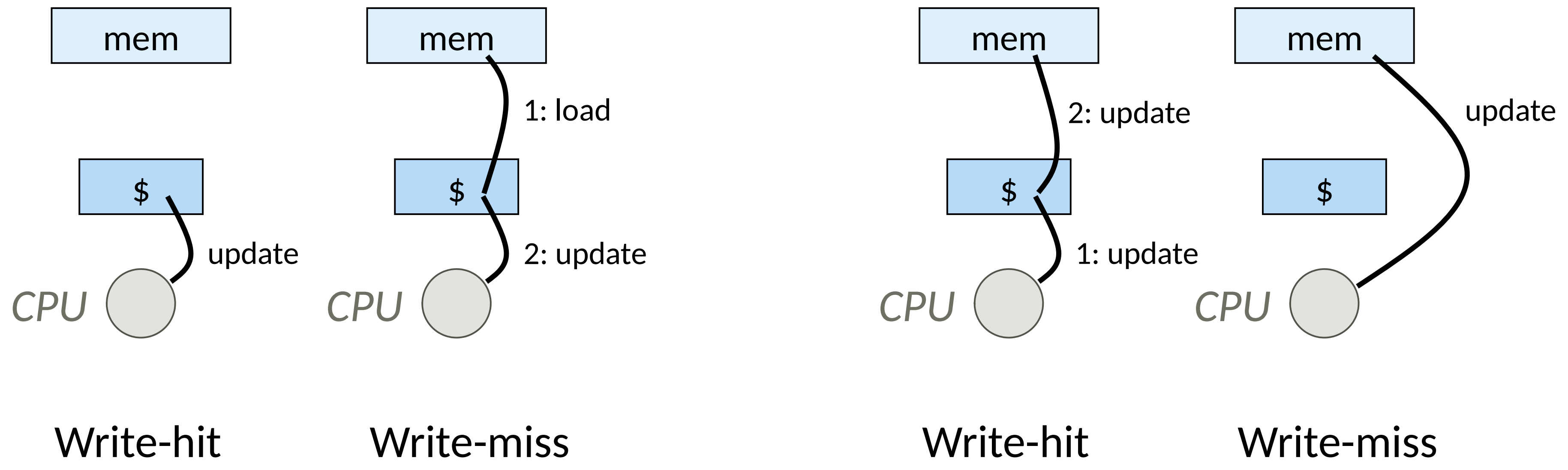
  Conflict misses occur when the cache is large enough, but multiple data objects all map to the same slot

- **Not a clean classification but still useful**

What about writes?

- **What to do on a write-hit?**
  - ***Write-through:*** write immediately to memory
  - ***Write-back:*** defer write to memory until replacement of line
- **What to do on a write-miss?**
  - ***Write-allocate:*** load into cache, update line in cache
  - ***No-write-allocate:*** writes immediately to memory

Write-back/write-allocate *Write-through/no-write-allocate*



The actual topic: Cache Coherence in Multiprocessors

- **Different caches may have a copy of the same memory location!**
- **Cache coherence (later / next lecture)**
  - Manages existence of multiple copies
- **Cache architectures**
  - Multi level caches
  - Shared vs. private (partitioned)
  - Inclusive vs. exclusive
  - Write back vs. write through
  - Victim cache to reduce conflict misses
  - …

Cache Coherence Protocol

- **Programmer can hardly deal with unpredictable behavior!**
- **Cache controller maintains data integrity**
  - All writes to different locations are visible

Fundamental Mechanisms

- **Snooping**
  - Shared bus or (broadcast) network
- **Directory-based**
  - Record information necessary to maintain coherence:
    E.g., owner and state of a line etc.

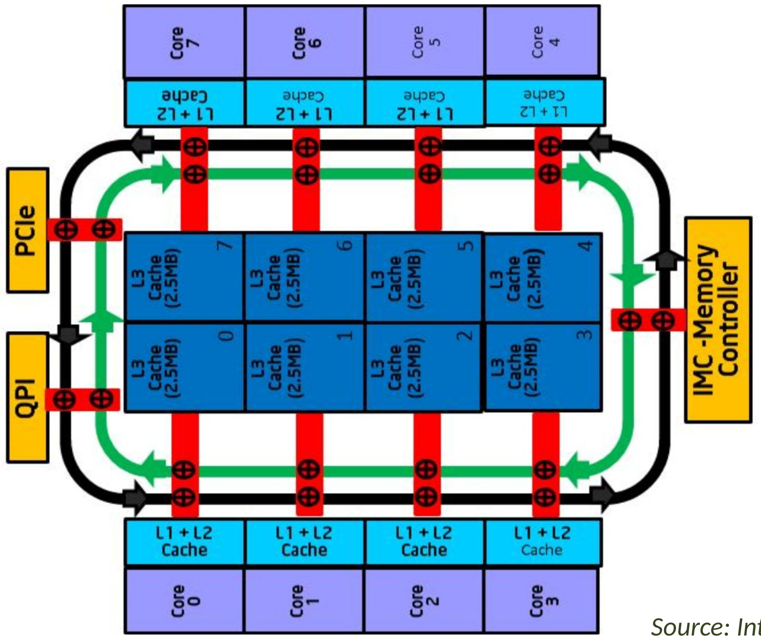
Fundamental CC mechanisms

- **Snooping**
  - Shared bus or (broadcast) network
  - Cache controller “snoops” all transactions
  - Monitors and changes the state of the cache’s data
  - Works at small scale, challenging at large-scale

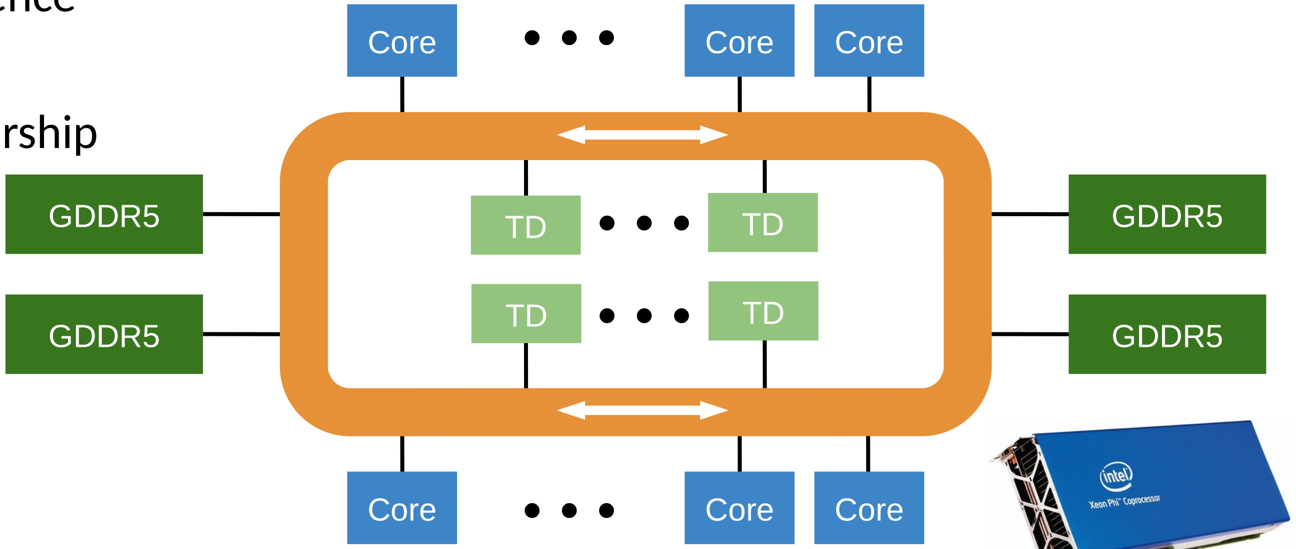
    E.g., Intel Broadwell
- **Directory-based**
  - Record information necessary to maintain coherence

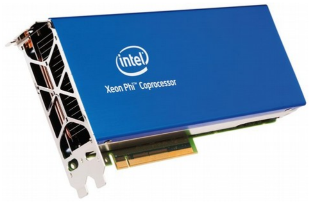
    E.g., owner and state of a line etc.
  - Central/Distributed directory for cache line ownership
  - Scalable but more complex/expensive

    E.g., Intel Xeon Phi KNC/KNL



Source: Intel





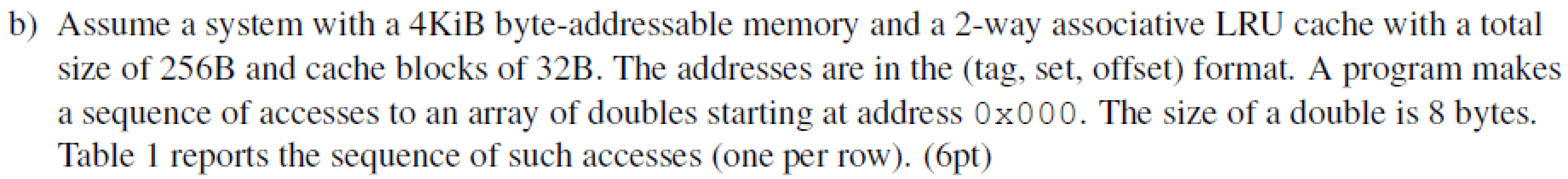
Exam question (6 min to solve in exam, 10 min now, in pairs)

b) Assume a system with a 4KiB byte-addressable memory and a 2-way associative LRU cache with a total size of 256B and cache blocks of 32B. The addresses are in the (tag, set, offset) format. A program makes a sequence of accesses to an array of doubles starting at address `0x000`. The size of a double is 8 bytes. Table 1 reports the sequence of such accesses (one per row). (6pt)

Address	Tag	Set	Offset	Miss?
0x050	0	2	16	Y
0x028				
0x158				
0x0E0				
0x040				
0x080				

	Block 0	Block 1
Set 0		
Set 1		
Set 2		
Set 3		

Solution


b) Assume a system with a 4KiB byte-addressable memory and a 2-way associative LRU cache with a total size of 256B and cache blocks of 32B. The addresses are in the (tag, set, offset) format. A program makes a sequence of accesses to an array of doubles starting at address `0x000`. The size of a double is 8 bytes. Table 1 reports the sequence of such accesses (one per row). (6pt)

How many bits wide are memory addresses?
4 KiB byte-addressable memory = 2^12 elements => 12 bit wide address.

How many bits for offset?
32B byte blocks of byte-addressabe memory= 2^5 => 5 offset bits

How many bits for set?
256B / 32B = 8 blocks, 8 / 2 = 4 sets (due to 2-way assoc.), 4=2^2 => 2 set bits.

How many bits for tag?
All remaining ones, 12 – (5+2) = 5 tag bits.

Now we decompose each address:
0x050 (hexadecimal) in binary = 0000 0101 0000 =>
rewrite as (tag(5b),set(2b),offset(5b)) 00000 10 10000,
convert to decimal => tag=0, set=2, offset=16

Solution

0x050 (hexadecimal) in binary = 0000 0101 0000 =>
rewrite as tag(5b) set(2b) offset(5b) 00000 10 10000,
convert to decimal => tag=0, set=2, offset=16

0x028 = 0000 0010 1000 => tag=0, set=1, offset=8
0x158 = 0001 0101 1000 => tag=2, set=2, offset=24
0x0E0 = 0000 1110 0000 => tag=1, set=3, offset=0
0x040 = 0000 0100 0000 => tag=0, set=2, offset=0
0x080 = 0000 1000 0000 => tag=1, set=0, offset=0

Now we can fill most of the first table.

Address	**Tag**	**Set**	**Offset**	**Miss?**
0x050	0	2	16	Y
0x028	0	1	8	
0x158	2	2	24	
0x0E0	1	3	0	
0x040	0	2	0	
0x080	1	0	0	

Solution

Address	Tag	Set	Offset	Miss?
0x050	0	2	16	Y
0x028	0	1	8	Y
0x158	2	2	24	Y
0x0E0	1	3	0	Y
0x040	0	2	0	N
0x080	1	0	0	Y

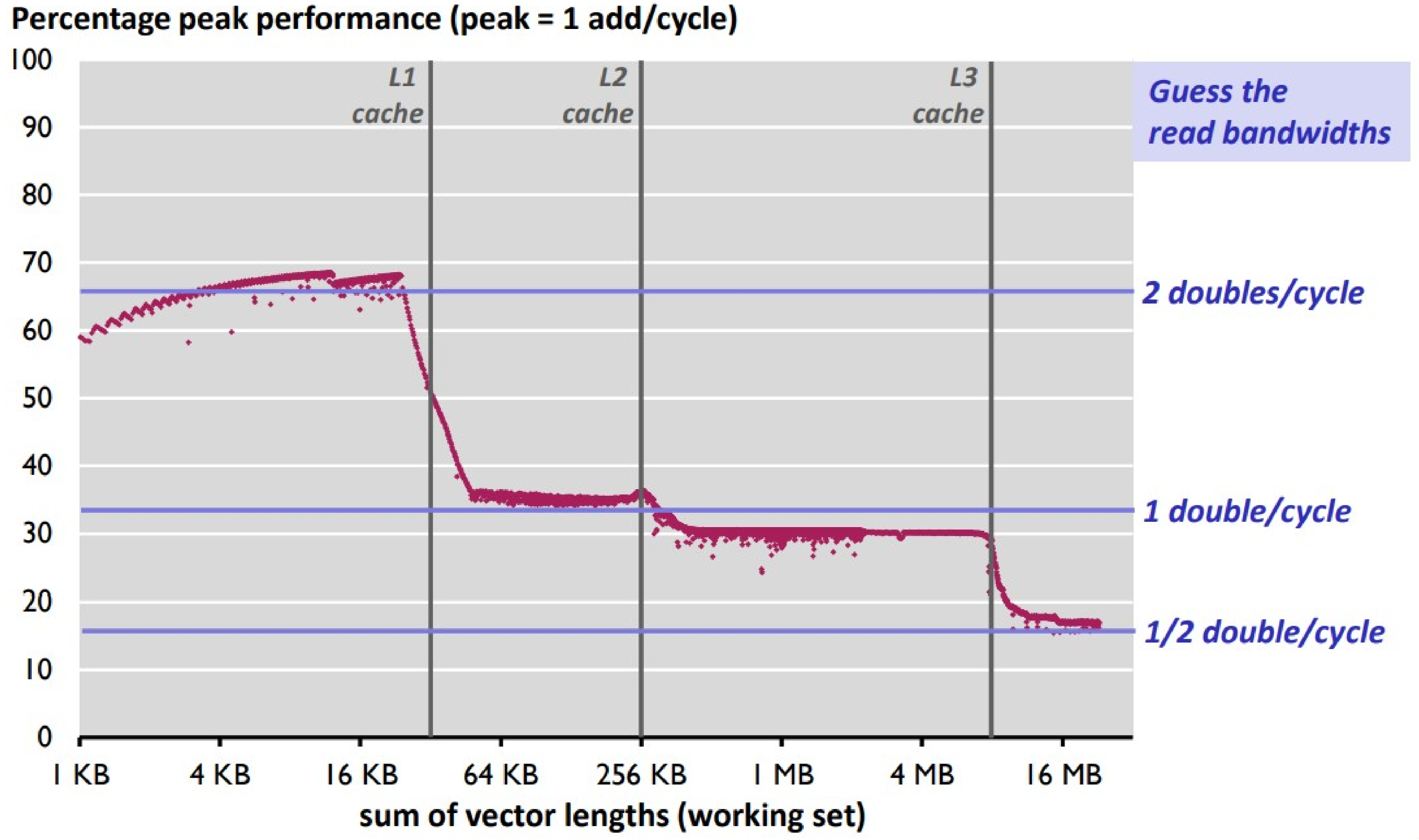
Now we go through the table and check for misses/hits and update the state of the cache.

The first number is the tag, the one in brackets the “timestep” / line in the left table.

	Block 0	**Block 1**
Set 0	1 (6)	
Set 1	0 (2)	
Set 2	0 (1 / hit in 5)	2 (3)
Set 3	1 (4)	

Example: Vector Add, Warm Data & Code

z = x + y on Core i7 (Nehalem, one core, no SSE), icc 12.0 /O2 /fp:fast /Qipo



Homework

Write a program which allows you to determine the sizes of the different caches in your laptop / computer.

Do not query them, measure the time it takes to perform some operation.