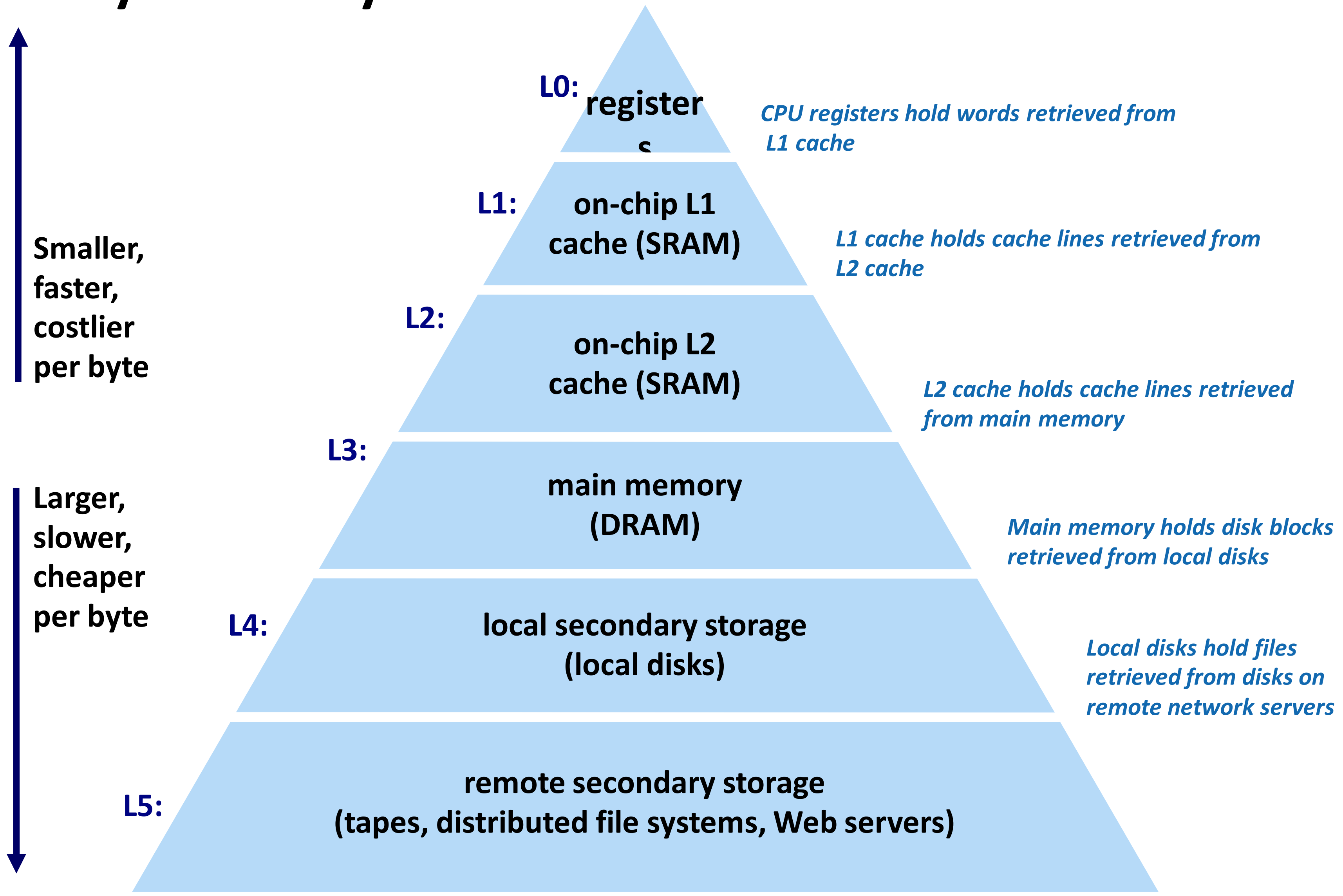
SALVATORE DI GIROLAMO <DIGIROLS@INF.ETHZ.CH>

DPHPC: Caches

Recitation session

Systems@ETH zürich

Typical Memory Hierarchy

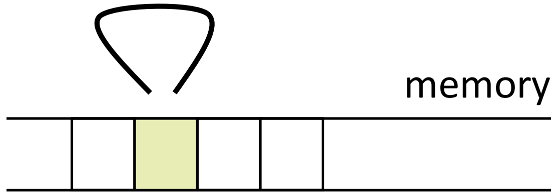

Smaller, faster, costlier per byte
Larger, slower, cheaper per byte
L0: registers
L1: on-chip L1 cache (SRAM)
L2: on-chip L2 cache (SRAM)
L3: main memory (DRAM)
L4: local secondary storage (local disks)
L5: remote secondary storage (tapes, distributed file systems, Web servers)
CPU registers hold words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

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**

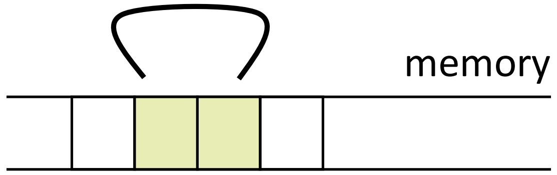
- ***Temporal locality:***

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



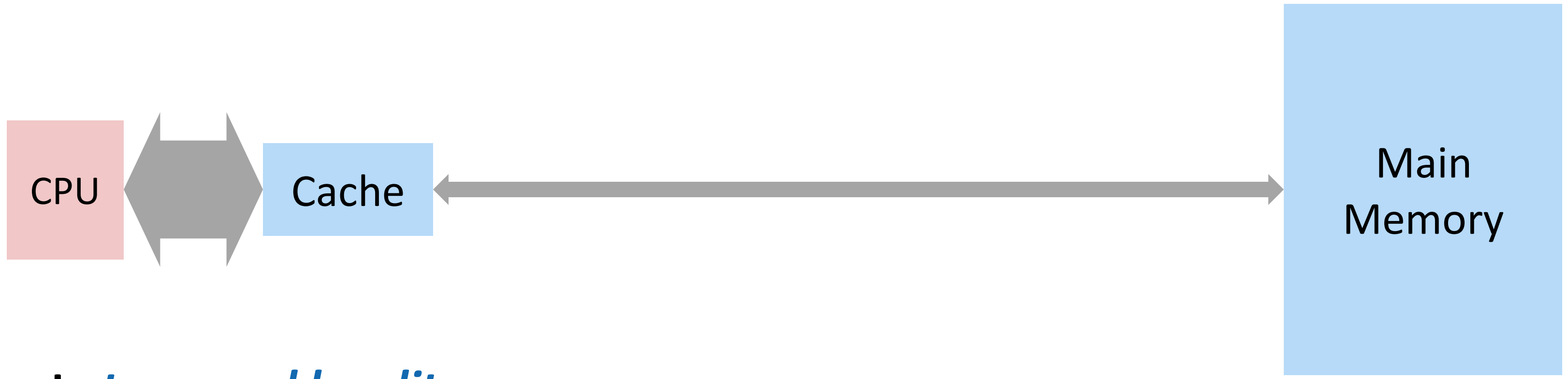
- ***Spatial locality:***

  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

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**

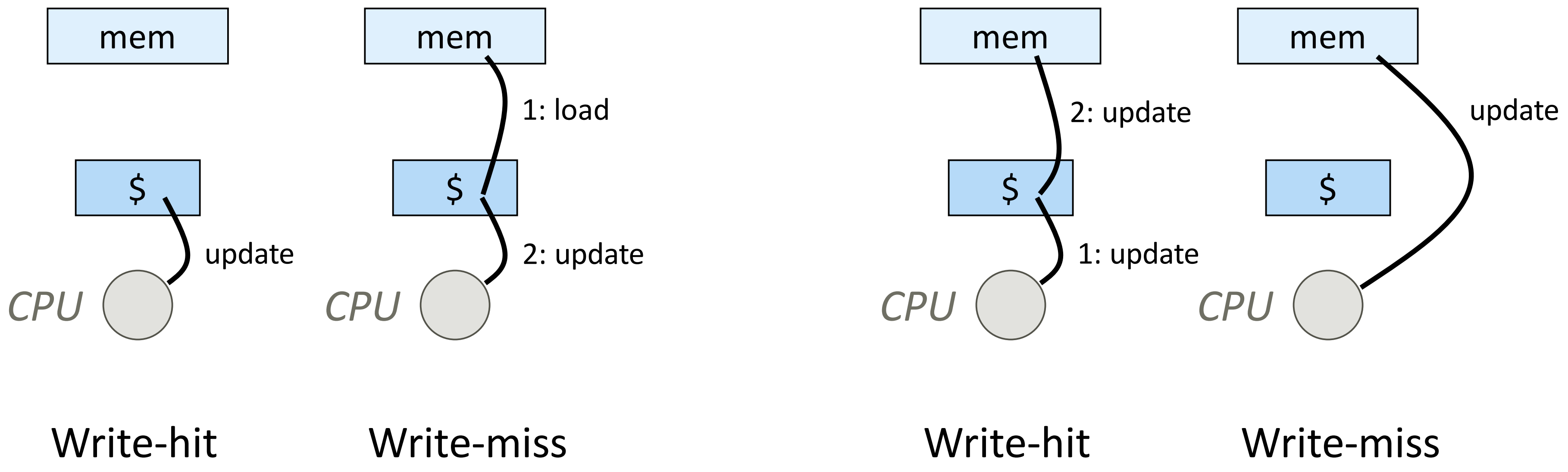
  Conflict misses occur when the cache is large enough, but multiple data objects all map to the same slot

- **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 (Core) *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**
  - 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.

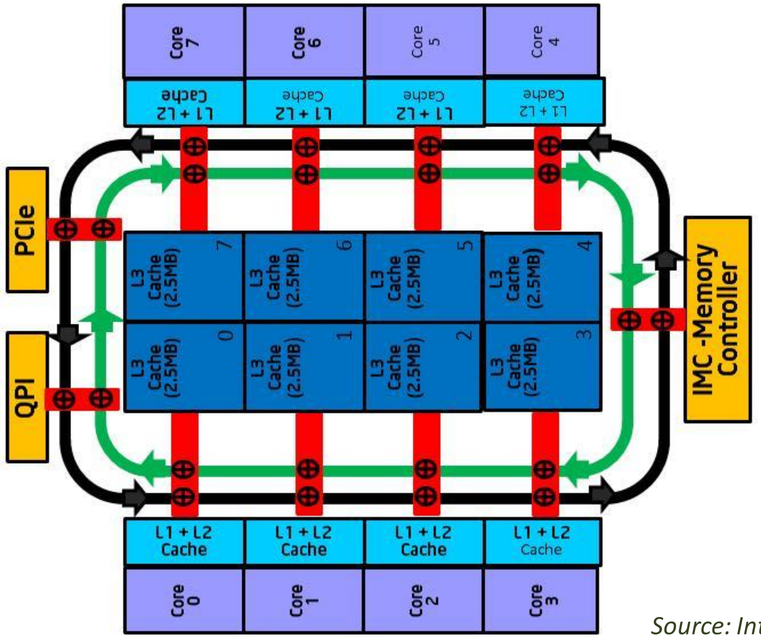
Fundamental CC mechanisms

- **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

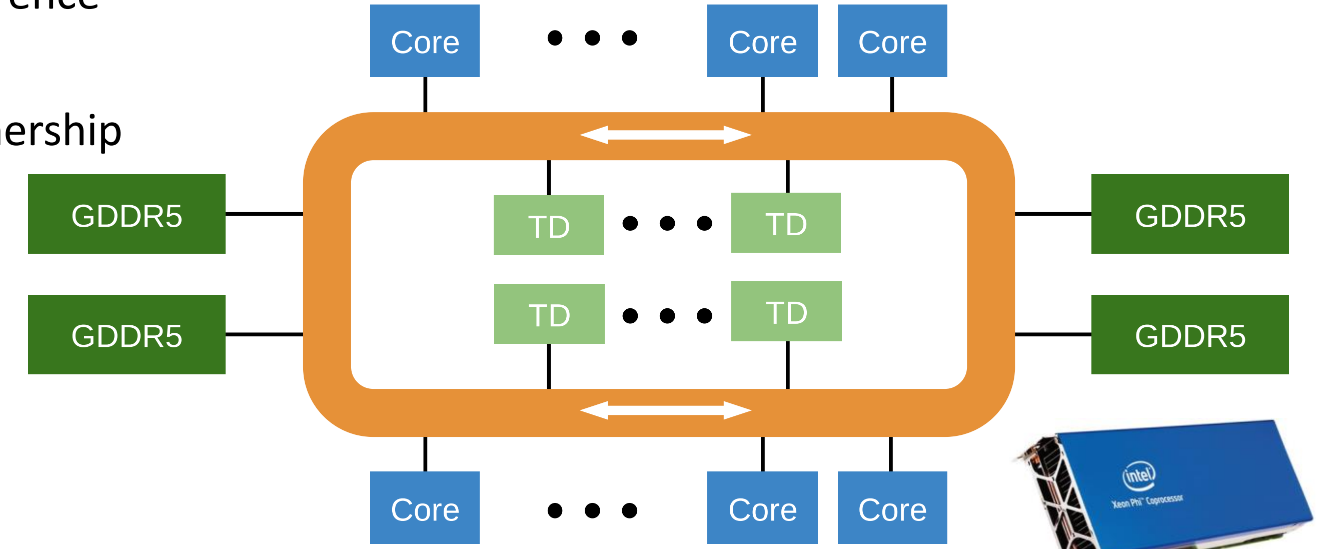
    E.g., Intel Broadwell
- **Directory-based**
  - Record information necessary to maintain coherence

    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



MESI Cache Coherence

- **Most common hardware implementation of discussed requirements**

  aka. “Illinois protocol”

Each line has one of the following states (in a cache):

- **Modified (M)**
  - Local copy has been modified, no copies in other caches
  - Memory is stale
- **Exclusive (E)**
  - No copies in other caches
  - Memory is up to date
- **Shared (S)**
  - Unmodified copies *may* exist in other caches
  - Memory is up to date
- **Invalid (I)**
  - Line is not in cache

Terminology

- **Clean line:**
  - Content of cache line and main memory is identical (also: memory is up to date)
  - Can be evicted without write-back
- **Dirty line:**
  - Content of cache line and main memory differ (also: memory is stale)
  - Needs to be written back eventually

    Time depends on protocol details
- **Bus transaction:**
  - A signal on the bus that can be observed by all caches
  - Usually blocking
- **Local read/write:**
  - A load/store operation originating at a core connected to the cache

Transitions in response to local reads

- **State is M**
  - No bus transaction
- **State is E**
  - No bus transaction
- **State is S**
  - No bus transaction
- **State is I**
  - Generate bus read request (BusRd)
    May force other cache operations (see later)
  - Other cache(s) signal “sharing” if they hold a copy
  - If shared was signaled, go to state S
  - Otherwise, go to state E
- **After update: return read value**

Transitions in response to local writes

- **State is M**
  - No bus transaction
- **State is E**
  - No bus transaction
  - Go to state M
- **State is S**
  - Line already local & clean
  - There may be other copies
  - Generate bus read request for upgrade to exclusive (BusRdX*)
  - Go to state M
- **State is I**
  - Generate bus read request for exclusive ownership (BusRdX)
  - Go to state M

Transitions in response to snooped BusRd

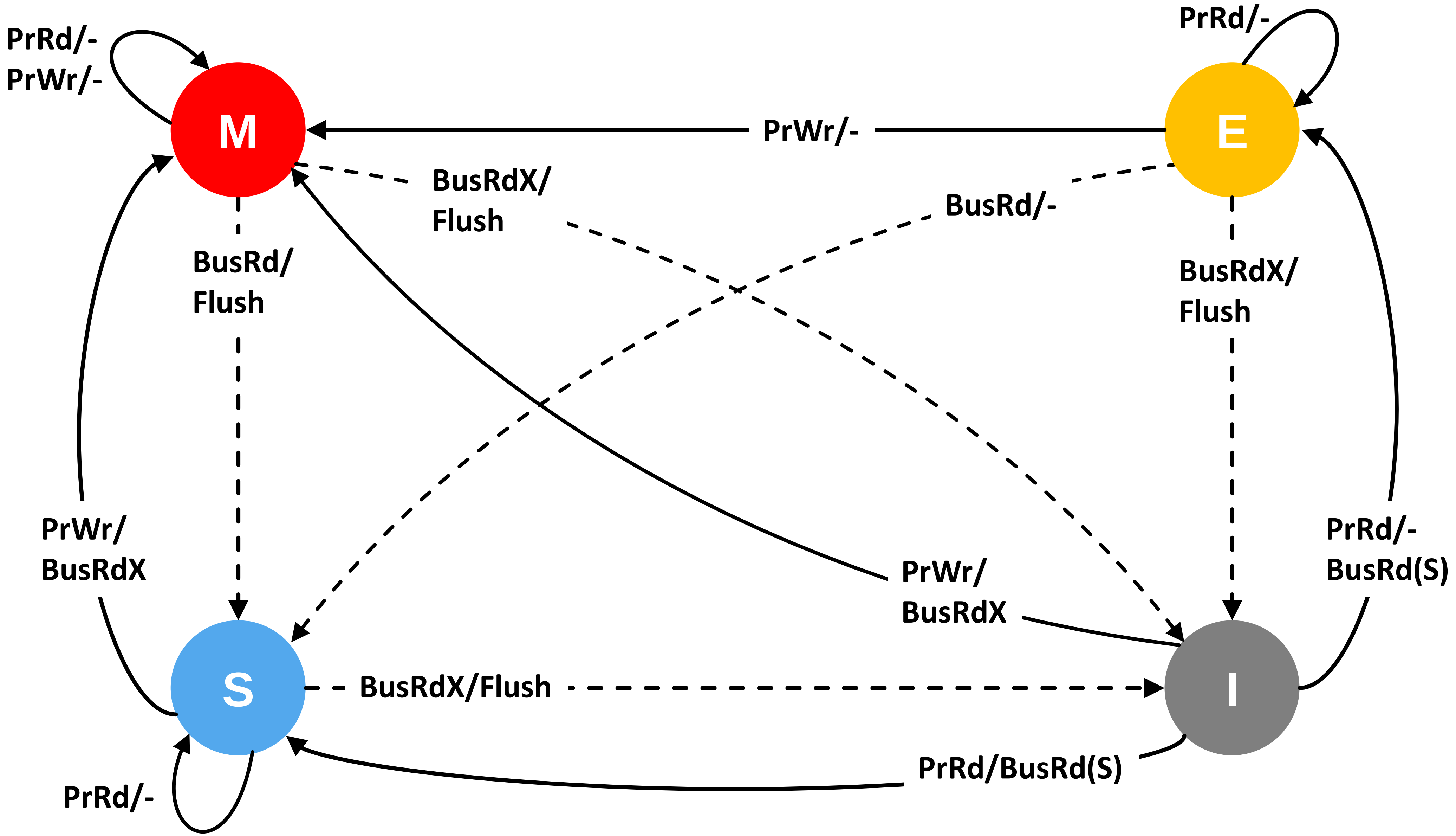
- **State is M**
  - Write cache line back to main memory
  - Signal “shared”
  - Go to state S (or E)
- **State is E**
  - Signal “shared”
  - Go to state S and signal “shared”
- **State is S**
  - Signal “shared”
- **State is I**
  - Ignore

Transitions in response to snooped BusRdX

- **State is M**
  - Write cache line back to memory
  - Discard line and go to I
- **State is E**
  - Discard line and go to I
- **State is S**
  - Discard line and go to I
- **State is I**
  - Ignore

- **BusRdX* is handled like BusRdX!**

MESI State Diagram (FSM)



How to monitor cache misses?

- **Look at performance counters!**
  - PAPI http://icl.utk.edu/papi/
  - perf stat

```
perf stat -B -e cache-references,cache-misses,cycles
```

Use `perf list` *to get the list of events you can ask for!*

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		

e) Assume a machine with 32bit addresses and 4 processors with directly-mapped L1 caches. Each cache is 4MiB in size, with 128B wide cache lines. MESI is used as cache-coherency mechanism. The memory is byte-addressable.

Tag/Set/Offset: How many bits of the address are used as tag-, set-, and offset-bits? Assume the offset bits are the least significant ones, while the tag bits are the most significant. (3pt)

Cache Coherence: For the following sequence of instructions, list the cacheline index (CL) that they target, the new state of the cacheline, and the bus signal. Use the provided table, filling only the fields that change their values after an instruction. Assume that initially all the cachelines are invalid (I). (8pt)

Inst.	Processor 1		Processor 2		Processor 3		Processor 4		Bus Signal
	CL	State	CL	State	CL	State	CL	State	
`P1: R(0xFF001001)`	32	E							BusRd
`P2: R(0xFF001034)`									
`P3: W(0xFF040023)`									
`P1: W(0xFF001023)`									
`P3: R(0xFF040078)`									
`P4: W(0xFF040000)`									