CG2028 Lecture 5 : Cache Memory Principles

Ankit Srivastava, NUS

CG2028

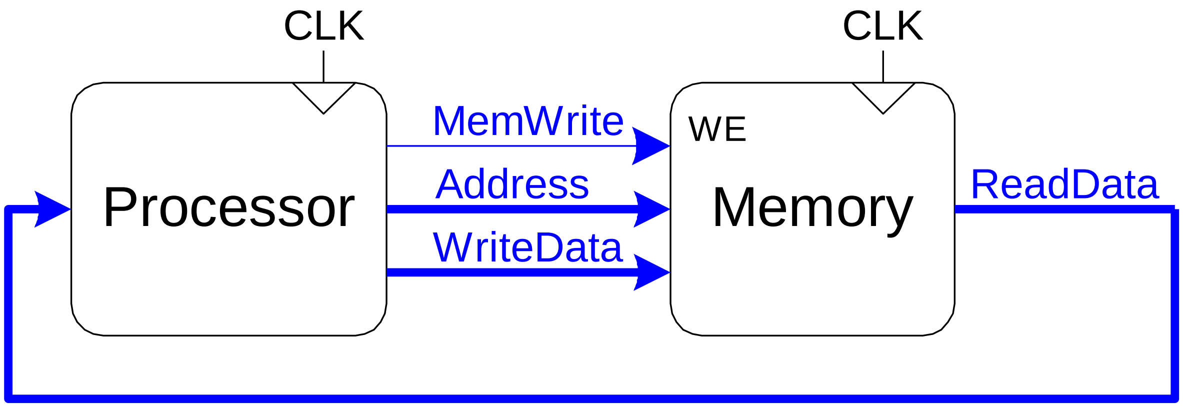
Acknowledgement :

- Slides from Dr Rajesh Panicker
- Some slides from Prof. Bharadwaj Veeravalli
- Text by Patterson and Hennessey and companion slides
- Text and companion slides by Harris and Harris

Note:

- Not all slides will be covered in the lecture. The rest are left as a self-learning exercise.

Data Memory Interface



- WriteData and ReadData often combined into a single bidirectional bus interface
- Data Memory is typically composed of non-volatile memories (ROM/Flash) and volatile (RAM) memories. RAM has no contents at power on – a location should be written before it is read (i.e., there should be an STR to a location before LDR)
- Instruction memory is also external to the processor (not shown here)

Storage Media – Access Times and Costs

- Static RAM (SRAM)
  - Data stored in flip-flops
  - Usually used for cache
  - 0.5ns – 2.5ns, $2000 – $5000 per GB
- Dynamic RAM (DRAM)
  - Data stored in capacitors – refresh circuitry required
  - Usually used for Main Memory (MM)
  - 50ns – 70ns, $20 – $75 per GB
- Magnetic disk (hard disk)
  - 5ms – 20ms, $0.20 – $2 per GB
  - Getting replaced by faster and more reliable (but costlier) flash-based solid state drives (SSDs)
- Ideal memory – best of both worlds
  - Access time of SRAM; capacity and cost/GB of disk

Memory Hierarchy

registers in CPU M_0

| bytes/words

cache memories (SRAM) M_1

| blocks

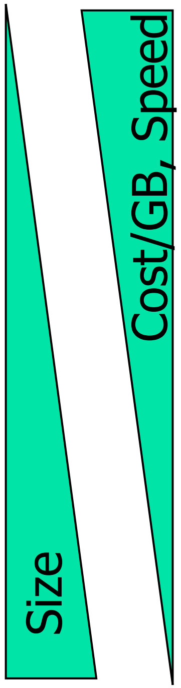
Main memory (DRAM) M_2

| pages

disk storage M_3

| file segments

tape units/optical disks, etc. M_4



- Basic idea
  - Each level holds the most frequently accessed data from the immediate higher level
  - Reduces the effect of lower speed of the higher level without increasing the overall cost significantly

Memory Hierarchy Properties

- Coherence (consistency) Property
  - Emphasizes the need for the copies of same data to have same information at all the levels where the data is currently residing
  - If a word is modified in the cache, it must be updated at all levels
- Locality of references
  - The memory access pattern tends to be clustered in certain regions in time, space, and ordering
  - 90-10 rule by Hennessy and Patterson (1990) - a typical program may spend 90% of its execution time on only 10% of the code such as the innermost loop of a nested loop
  - Temporal: Recently referenced items are likely to be referenced in the near future - keep recently accessed data at a faster level
  - Spatial: Refers to the tendency of a process to access the items whose addresses are near to one another - when accessing data, bring nearby data also into a faster level

Memory Capacity Planning

- Hit ratios
  - When a memory M_i is accessed and if the desired word is found, it is referred to as a *hit*, otherwise *miss*
  - The hit ratio (h_i) is the probability that a word/information will be found when accessed in M_i. Miss ratio is $1-h_i$
  - The hit ratios at successive levels are a function of memory capacities, management policies, and program behaviour
  - $h_0=0$ and $h_n=1$. This means that the CPU always access M_1 first and access to the outermost level is always a hit
- Access frequency at a level i is defined as
  - $f_i = (1-h_1)(1-h_2)\ldots(1-h_{i-1})\, h_i$
  - Note that $f_1 + f_2 + \ldots + f_n = 1$ and $f_1 = h_1$
  - Due to the locality property, the access frequencies decrease rapidly from the lower levels, i.e., access freq at level i is greater than i+1
  - This means that the inner levels are accessed more often than the outer levels

Memory Capacity Planning …

- Effective Access Time is defined as

$$T_{eff} = f_1t_1 + f_2t_{2+\ldots}\ f_nt_n$$

where t_i is the access time at level i

- The total cost of a memory hierarchy is estimated as

$$C_{total} = c_1s_1+c_2s_2+\ldots+c_ns_n$$

where c_i is the cost/MB and s_i is the size (in MB) at level i

- Hierarchy optimization involves minimizing

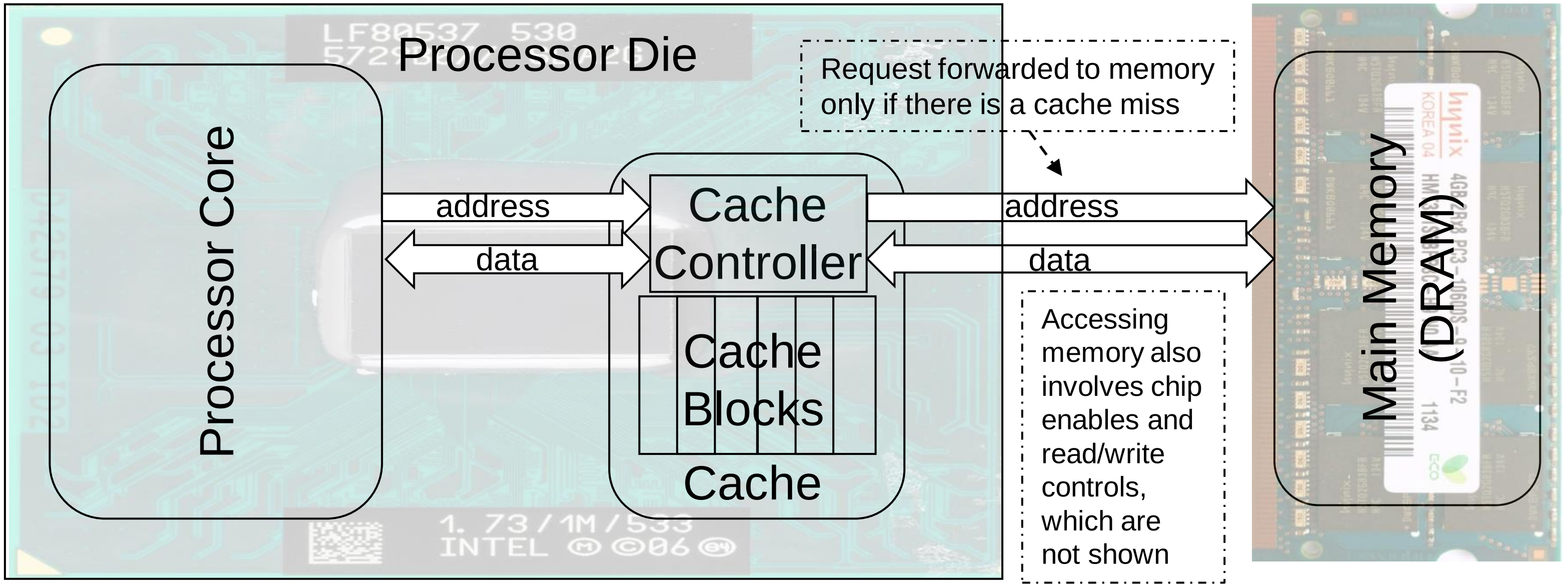
T_{eff} given $C_{total} < C_{max}$ or C_{total} given $T_{eff} < T_{max}$

- The optimal design should result in a T_{eff} close to t_1 and a total cost close to c_n

Cache Working Principle

- When a read request is received from CPU, the contents of a block of memory words containing the location specified are transferred to the cache
  - Block is also called *cache line*, typically ~64 bytes
  - *Where* to place the incoming block in the cache is decided by the *mapping function*
- Subsequently, when the program asks for any of the locations from this block, the desired contents are read directly from cache
  - CPU need not even be aware of the presence of the cache and issues addresses meant for main memory (MM, usually DRAM)
  - Checking if the required data/block is present in the cache is performed by the cache controller
  - If yes, a *cache hit* is said to occur

Cache Working Principle …



Request forwarded to memory only if there is a cache miss

Accessing memory also involves chip enables and read/write controls, which are not shown

- When a block occupying cache is not referenced for a long time, it is pushed back to the MM to make space for another block
  - *Which* block to replace is decided by *replacement algorithms*
- Miss penalty: time taken to retrieve a block from slower level in the hierarchy

Read Misses

- Read miss
  - When a read miss happens, the block containing the word is loaded into the cache and then the desired word is sent to the CPU
- Load-through (*early restart*)
  - Alternatively, this word may be sent to the CPU as soon as it is read from the MM
  - Reduces CPU's waiting time, but additional circuitry needed
- Valid bit
  - If a location which is currently cached is modified in the main memory by an action which bypasses the CPU (eg : DMA), a *valid* bit for the corresponding cache block is cleared
  - The cache controller treats access to this location as a cache miss
  - Valid bits are set to 0 on power on!

DMA : A technique for moving data between memory and secondary storage / IO devices where the data transfer is managed by a separate hardware called DMA controller rather than through repeated LDR-STR by the processor

Handling Writes

- Write-through
  - In this case, the cache and MM locations are simultaneously updated
  - Simple, but results in unnecessary write operations in MM when cache is updated several times
- Write-back
  - Update only the cache location and mark it as updated with an associated flag bit, often called as *dirty* or modified bit
  - The MM word is updated later, when the block containing the word is removed from the cache by a replacement algorithm
  - May also lead to unnecessary write operations – when a cache block is written back to the memory, all the words of the block are written back, even if only a single word in that block was modified when it was in the cache

Mapping Techniques

- There are three different mapping techniques that are followed in practice
  - Direct mapping
  - Associative mapping
  - Set-Associative mapping
- The following example is used to illustrate the mapping algorithms
  - The cache consists of 128 blocks of 16 words each; a total of 2048 (2K) words
  - Assume that the MM is addressable by a 16-bit **word address** (not byte address, for simplicity)
  - MM has 64K words, which we will view as 4K blocks of 16 words each

Note: Real-word memories are usually byte-addressable. If a word = 32 bits, the word is spread over 4 locations (bytes). When accessing words, the last 2 bits of addresses are always 0s ($4=2^2$). The exact arrangement of bytes within a word is called *endianness*. A **L**ittle-endian computer stores the **L**east significant byte at the **L**owest address.

Direct Mapping

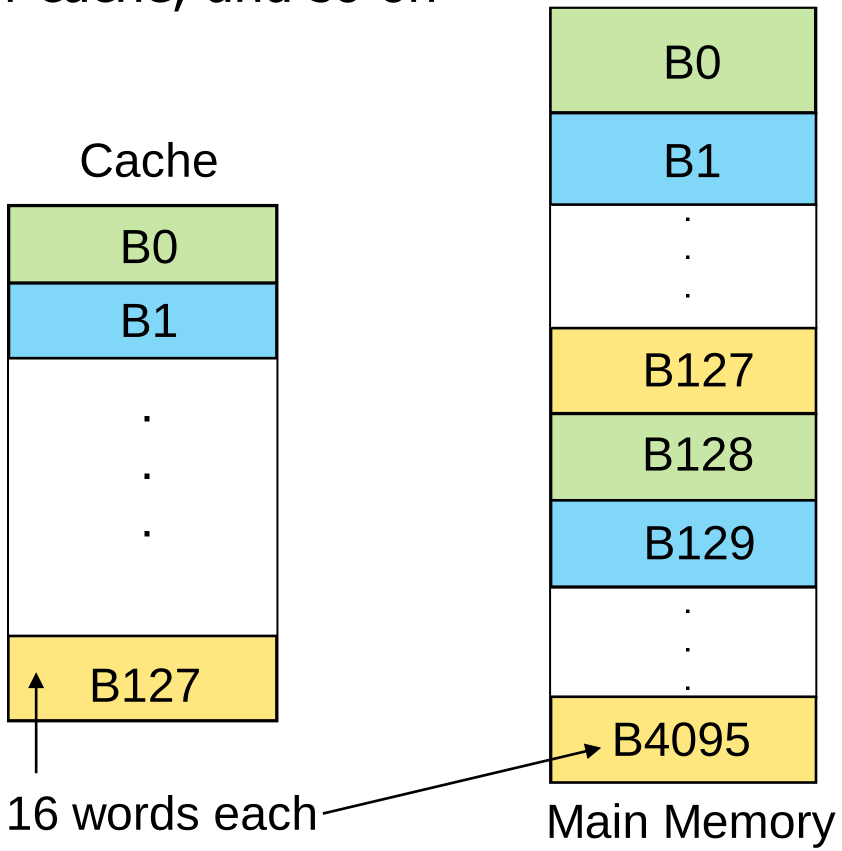
- Direct mapping

  block j of MM -> block j modulo 128 of Cache
- MM blocks 0,128,256,… -> block 0 of cache

  MM blocks 1,129,257,… -> block 1 of cache, and so on

5	7	4
Tag	Block	Word

- Total of 16 bits
- lower order -> select a word within the block
- middle order -> block number in the cache
- high order -> which of the 32 blocks (4K/128 = 32 = 2^5) from MM is residing currently in the cache block

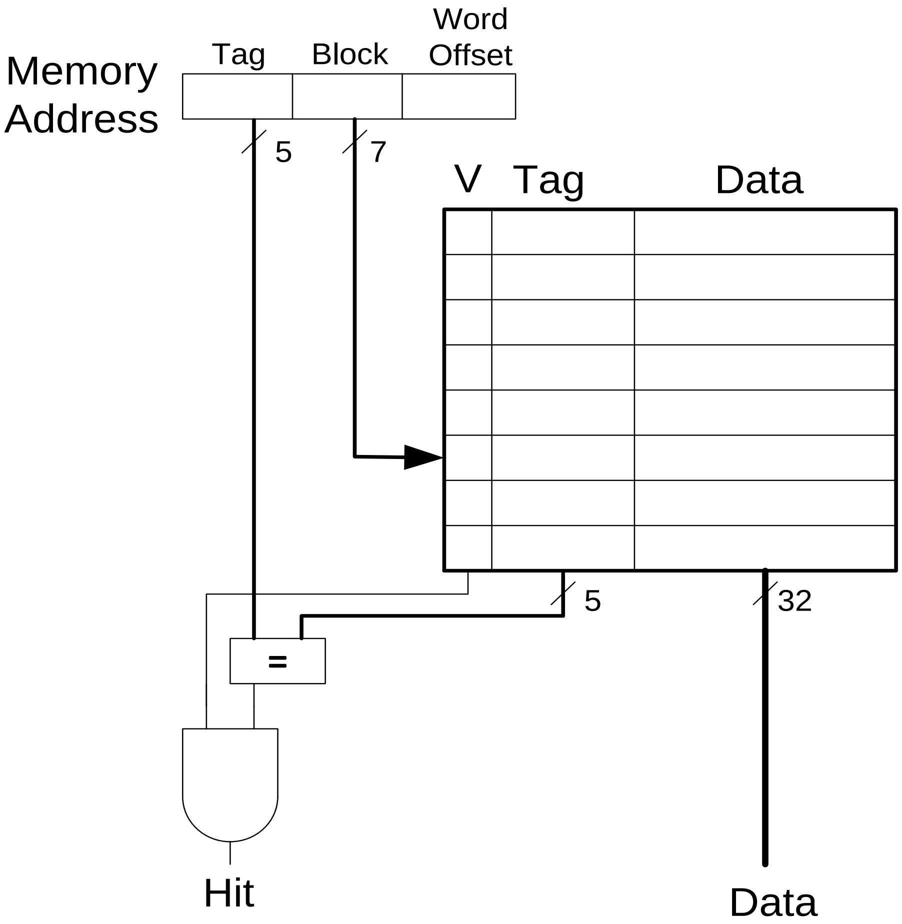


Direct Mapping …

- Note that the tag field in the above example is nothing but the higher order 5 bits of the word address
- These 5 bits are stored along with that block in the cache
- The tag field can be used to determine whether the block at this location is the required block – the tag field is unique for each block from MM which can be mapped to the same block in the cache
- Note that even when the cache is not full, contention may arise for a location
- In this case, the replacement algorithm is trivial (a main memory block is mapped to a unique cache block)

Direct Mapping …

Note: Word access within the block is not shown



Associative Mapping

- (Fully) Associative mapping
- In this technique, a block of MM can be placed anywhere in the cache

12	4
Tag	Word

- From the CPU generated address, the higher order 12 bits are stored along with the block in the cache (which makes sense as each cache block can be from any of the 4096 = 2^{12} MM blocks)
- When the request arrives, the tag field is compared for all the blocks in the cache to see if there is a match

Associative Mapping

- This technique gives a complete freedom in choosing where in the cache a particular MM block is placed
  - Cache space is utilized more efficiently
- Disadvantage: Search 128 blocks to match for a single tag
  - This comparison has to be done for every memory access!
  - Parallel search schemes can be used
  - Still, costly and difficult to achieve high speeds
- The replacement follows one of the standard techniques such as LRU, FIFO, etc.

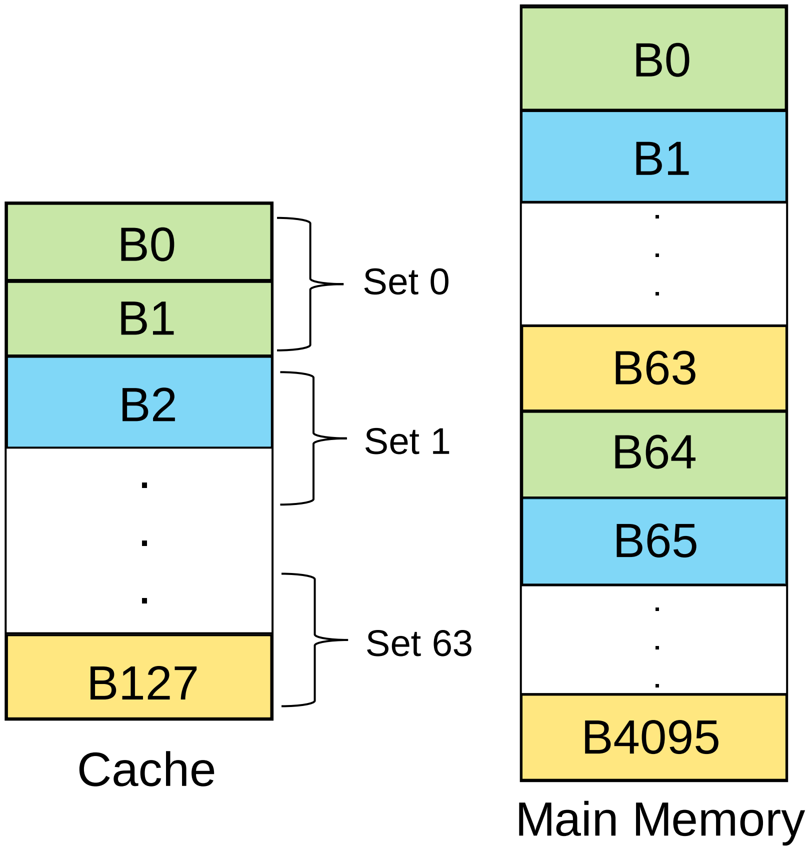
Set-Associative Mapping

- Set-Associative mapping
- This is a combination of / compromise between the previous techniques
- Here, blocks of cache are grouped into sets, and the mapping allows a block of the MM to reside in any block within a specific set (there is associativity within a set)
  - The contention problem of the direct method is eased by having a few choices for block placement
  - The hardware cost is reduced and speed is increased by decreasing the size of the associative search procedure
  - If there are N blocks per set, the memory is called N-way set associative

Set-Associative Mapping …

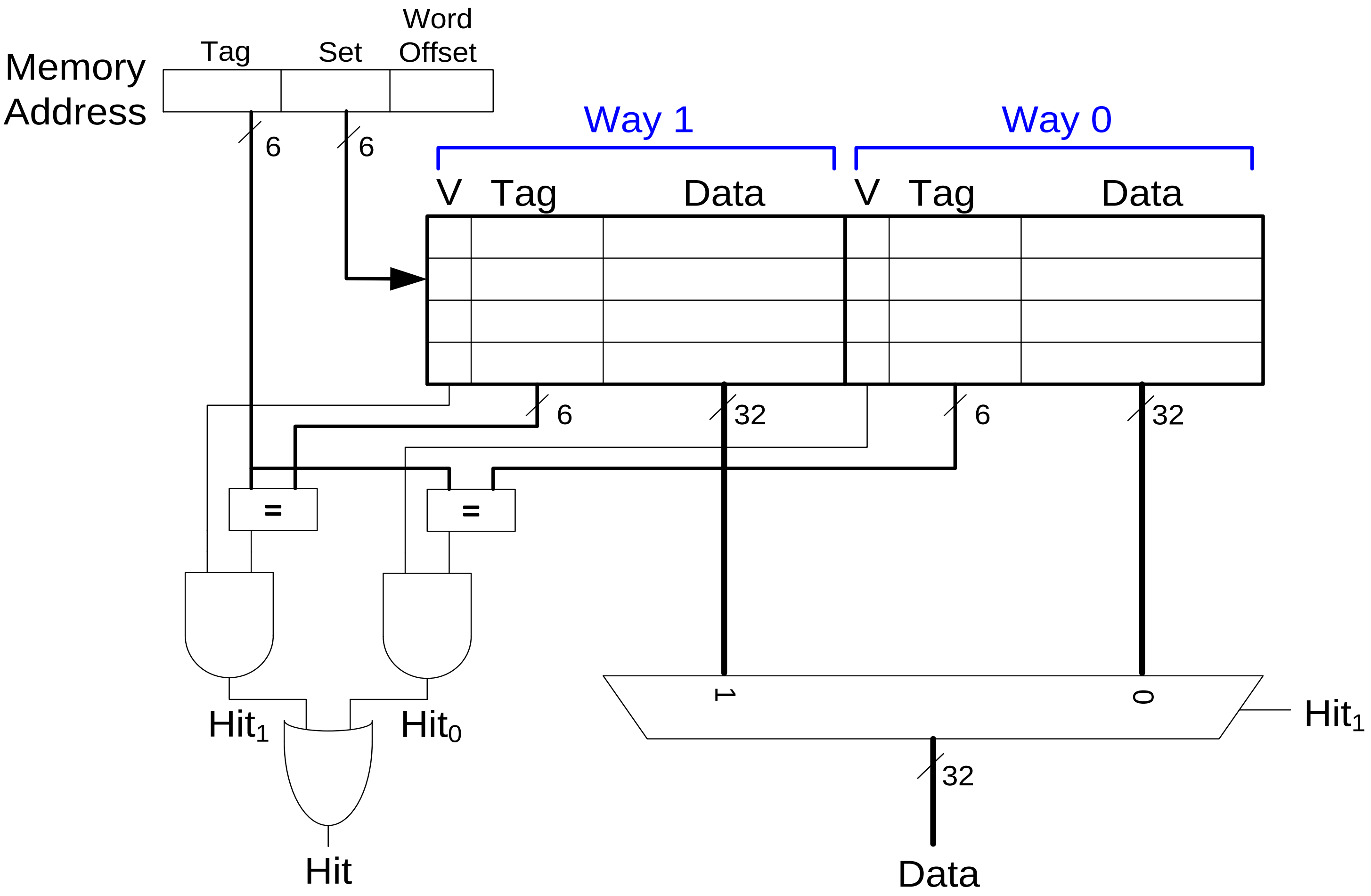
- Suppose if we allow two blocks per set in the cache. The memory blocks 0,64,128,…,4032 map into cache set 0, and they can occupy either of the two block positions within the set
- With 128 cache blocks and 2 blocks per set, we have 64 sets -> we need 6 bits to identify the right set and 4 bits for a word, leaving 6 bits for the Tag field (which makes sense as each cache block can be from any of the 4096/64 = 64 = 2^6 MM blocks)

6	6	4
Tag	Set	Word



2-Way Set-Associative Mapping

Note: Word access within the block is not shown



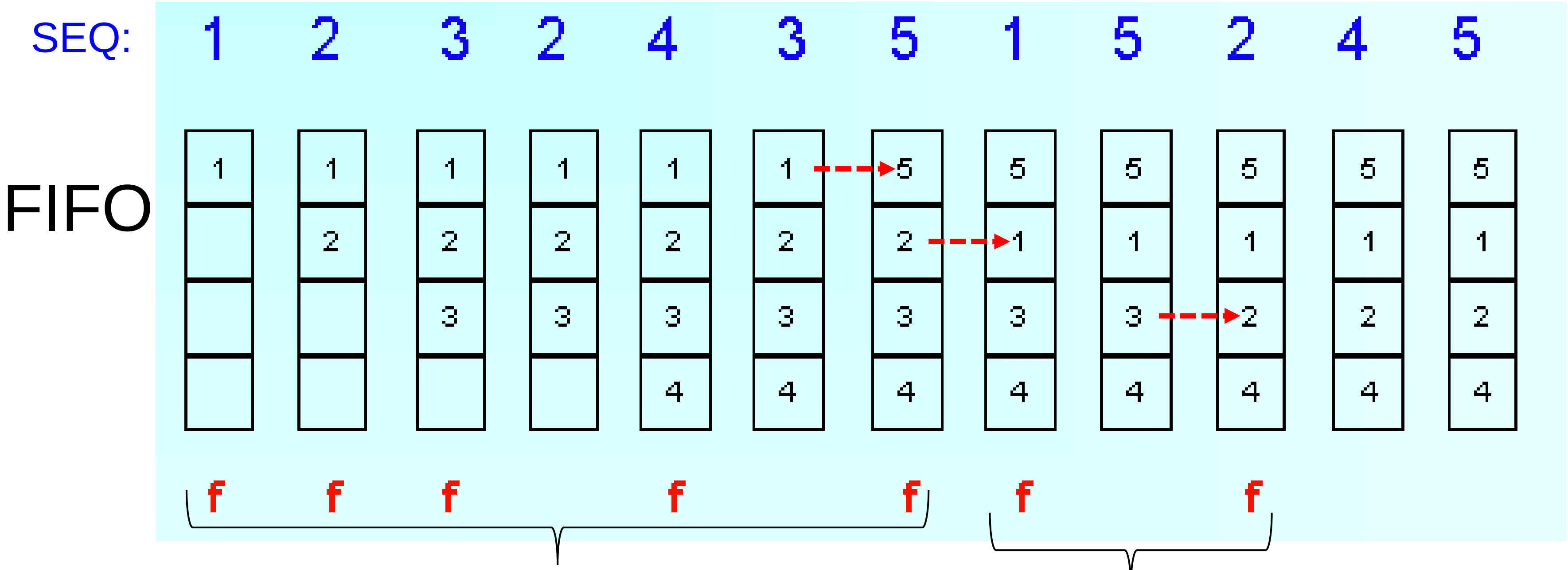
Replacement Algorithms

- In the case of associative and set-associative mapping, there is some flexibility in deciding which block should be thrown out if a new block is brought into the cache
  - Retain blocks that are likely to be referenced in the "near future"
  - Can't predict future - not easy to decide on "how long to hold a block"
- First-In-First-Out
  - Replace the oldest block in the memory
- Least Recently Used (LRU)
  - Replace the block that has not been referenced for a long time
- Optimal Algorithm: (Ideal - assumes knowing the future)
  - Replace the block that will not be used for a longest period of time
  - Cannot be implemented in practice, used only for analysis purpose

Optional to do: Read up about other misses such as *conflict* miss and *coherence* miss

Replacement Algorithm : FIFO

- In the example, assume that the cache is fully associative and has 4 blocks
- FIFO works well if the access follows a sequential pattern (arrays etc.)



SEQ:	1	2	3	2	4	3	5	1	5	2	4	5
FIFO	1	1	1	1	1	1	5	5	5	5	5	5
		2	2	2	2	2	2	1	1	1	1	1
			3	3	3	3	3	3	3	2	2	2
					4	4	4	4	4	4	4	4
	f	f	f		f		f	f		f		

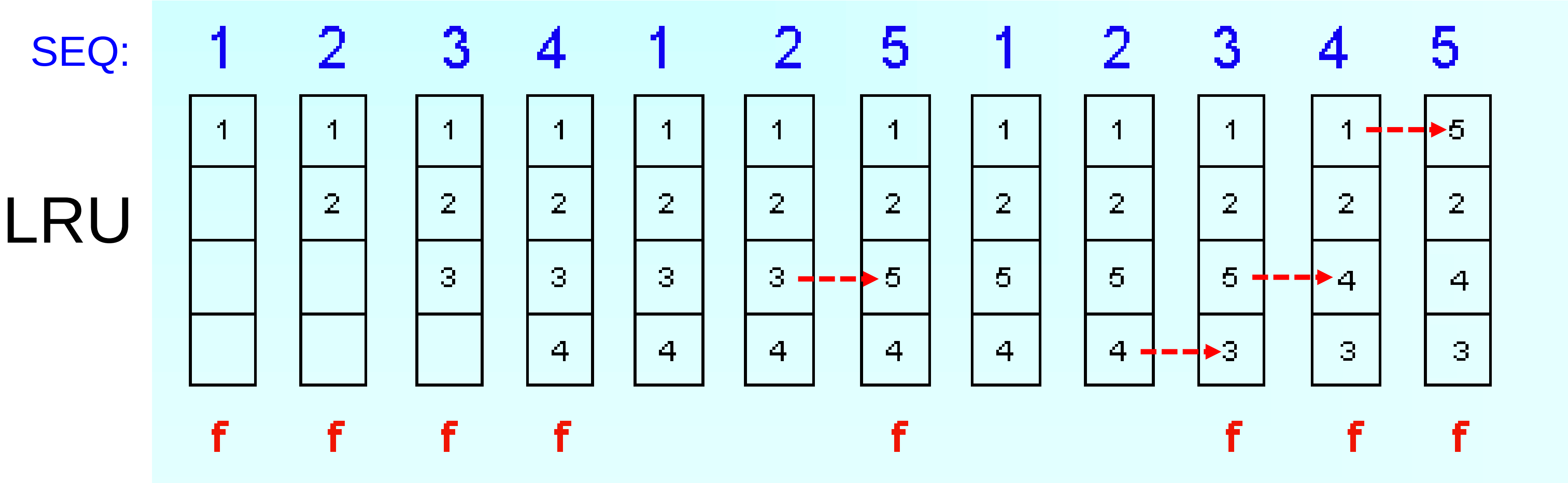
compulsory miss: data was never in the cache

capacity miss: data was there at some point, but evicted due to capacity limitations

f: miss

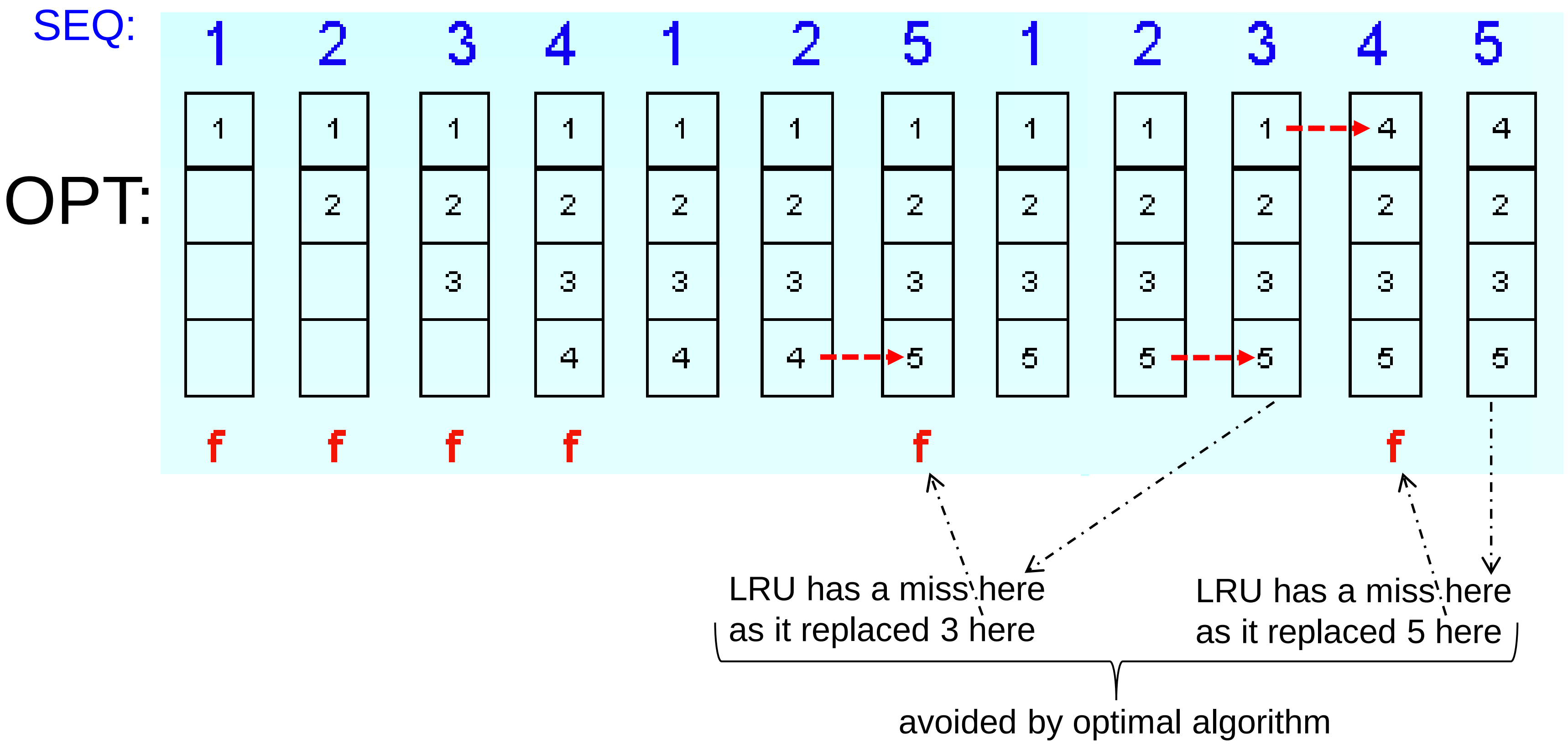
Replacement Algorithm : LRU

- It makes sense to overwrite a block that resided in the cache for a long time (LRU block) without being referenced
  - Temporal locality of reference



SEQ:	1	2	3	4	1	2	5	1	2	3	4	5
	1	1	1	1	1	1	1	1	1	1	1	5
LRU		2	2	2	2	2	2	2	2	2	2	2
			3	3	3	3	5	5	5	5	4	4
				4	4	4	4	4	4	3	3	3
	f	f	f	f			f			f	f	f

Replacement Algorithm : Optimal



SEQ:	1	2	3	4	1	2	5	1	2	3	4	5
OPT:	1	1	1	1	1	1	1	1	1	1	4	4
		2	2	2	2	2	2	2	2	2	2	2
			3	3	3	3	3	3	3	3	3	3
				4	4	4	5	5	5	5	5	5
	f	f	f	f			f				f	

LRU has a miss here as it replaced 3 here

LRU has a miss here as it replaced 5 here

avoided by optimal algorithm

Multilevel Caches

- Larger caches have lower miss rates, but longer access times
- Expand memory hierarchy to multiple levels of caches
- Most modern PCs have L1, L2, and L3 cache
  - Intel Coffee Lake has
    - Level 1: small and fast, 32 KB, 5-6 cycles
    - Level 2: larger and slower, 256 KB, 12 cycles
    - Level 3: 2MB x number of cores, shared among cores, 42 cycles
    - Level 4: 128MB (only in some models) per package
    - In contrast, main memory has a 42 cycles + 51 ns latency (about 200 cycles in total)

Summary

- Cache – basic principles
- Hit rate and effective access times
- Handling read and write misses
- Mapping techniques
- Replacement algorithms
- More of memory stuff, such as memory management algorithms, virtual memory etc. in CG2271
- Caches need to be coherent, which is tricky in multiprocessor systems. More on this, DMA etc. in CG3207
- Creating systems with specialized accelerators (co-processors) to accelerate certain functions (e.g., Machine learning) in EE4218
- EE4415, CS4223, CS5222, CEG5201, CEG5203,..