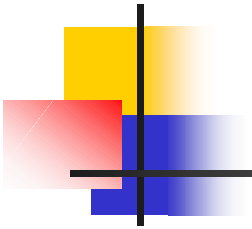


Computer Architecture
Lecture 12

Memory Hierarchy

Reference material:

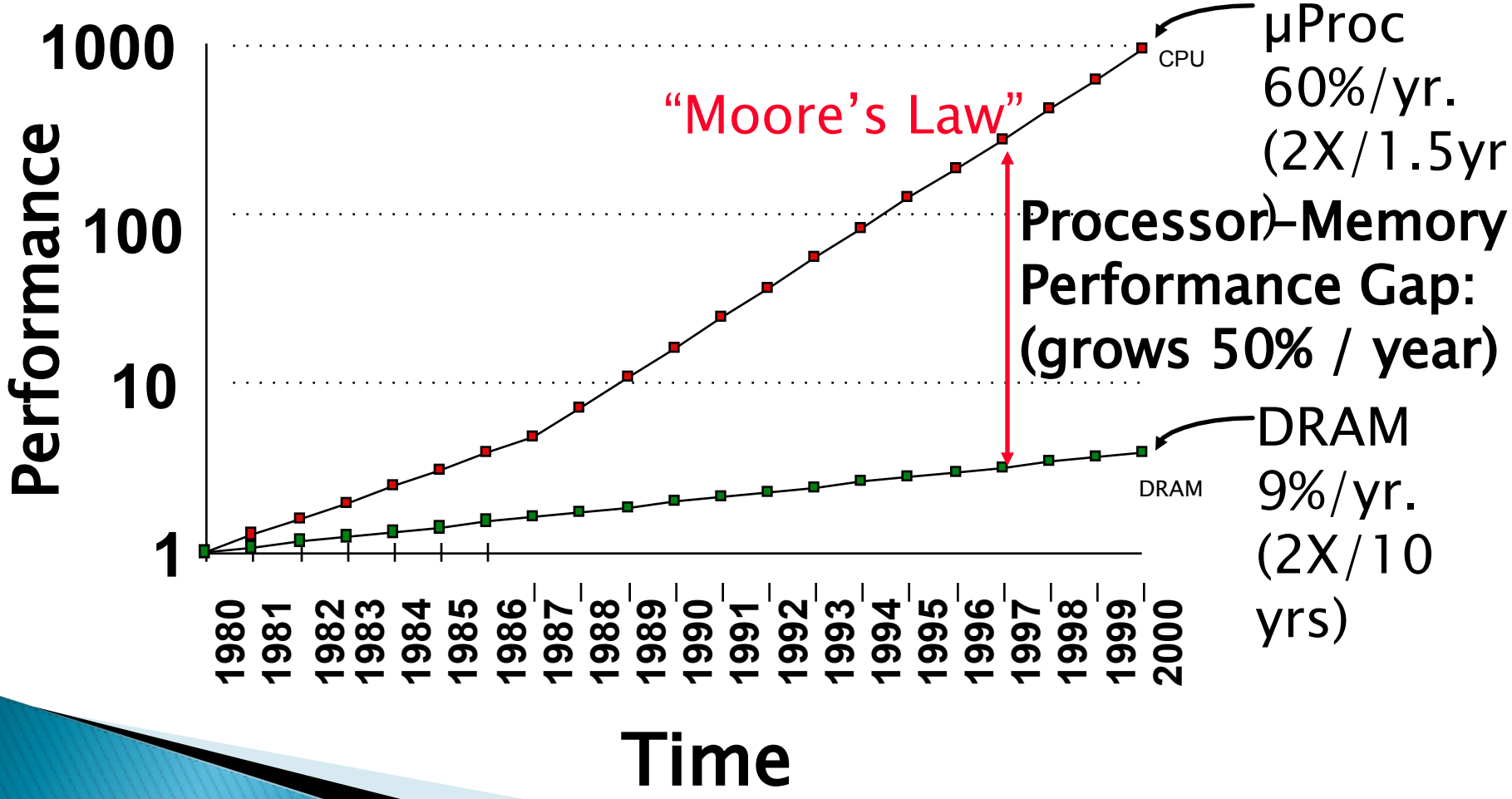
Computer Organization and Design: The Hardware and Software Interface by Patterson and Hennessy, 4th Edition, Morgan and Kauffman Publications, 2008 (chap-07)

Computer System Architecture by Morris Mano, 3rd ed. (chap-12)

Recommended YouTube links

- https://www.youtube.com/watch?v=LzsiFYVMqi8
- https://www.youtube.com/watch?v=Ref231Eg9QE
- https://www.youtube.com/watch?v=7IVE6vXqEoc

Processor–DRAM Gap

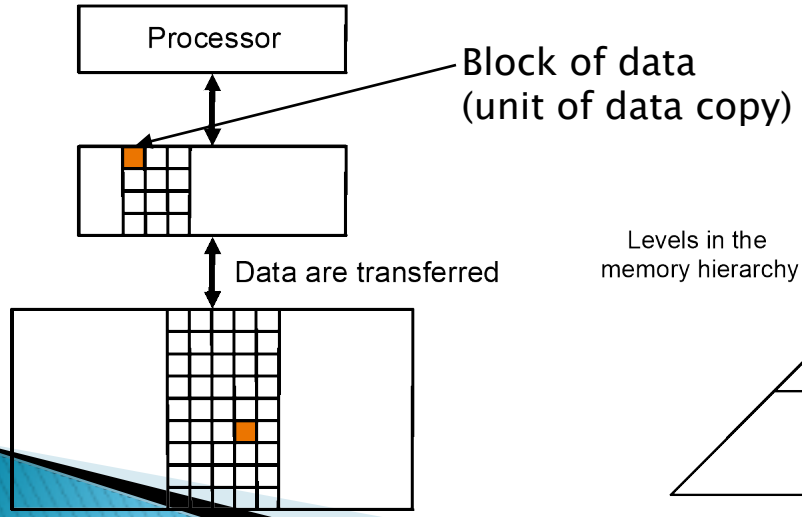


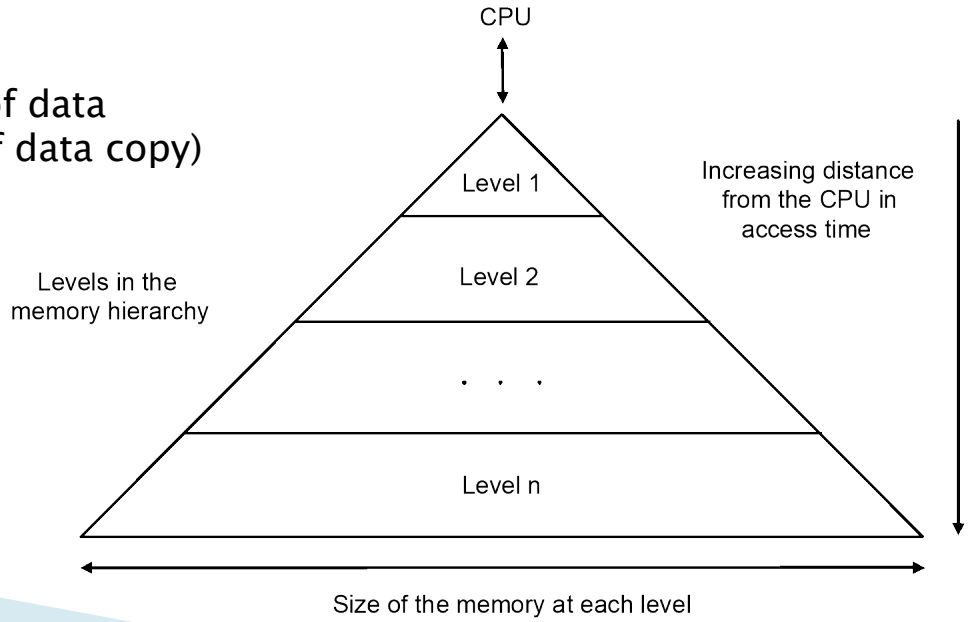
SRAM vs DRAM

- SRAM:
  - Value is stored on a pair of inverting gates
  - Very fast but takes up more space than DRAM (4 to 6 transistors)
  - Cost per bit is higher
  - Cache
- DRAM:
  - Value is stored as a charge on capacitor (must be refreshed)
  - Takes less space but slower than SRAM (factor of 5 to 10)
  - Cost per bit is less
  - Main Memory

Memory Hierarchy

- Users want large and fast memories…
  - expensive and they don't like to pay…
- Make it seem like they have what they want…
  - *memory hierarchy*
  - hierarchy is *inclusive*, every level is *subset* of lower level
  - performance depends on *hit rates*

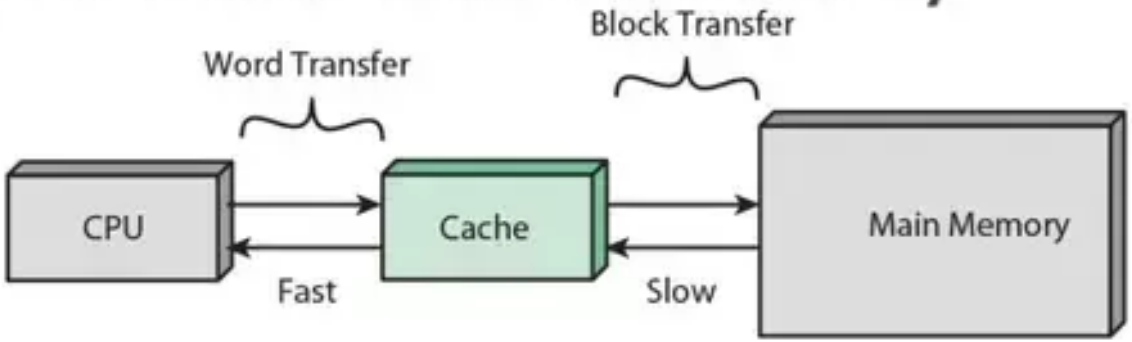




Memory requirement?

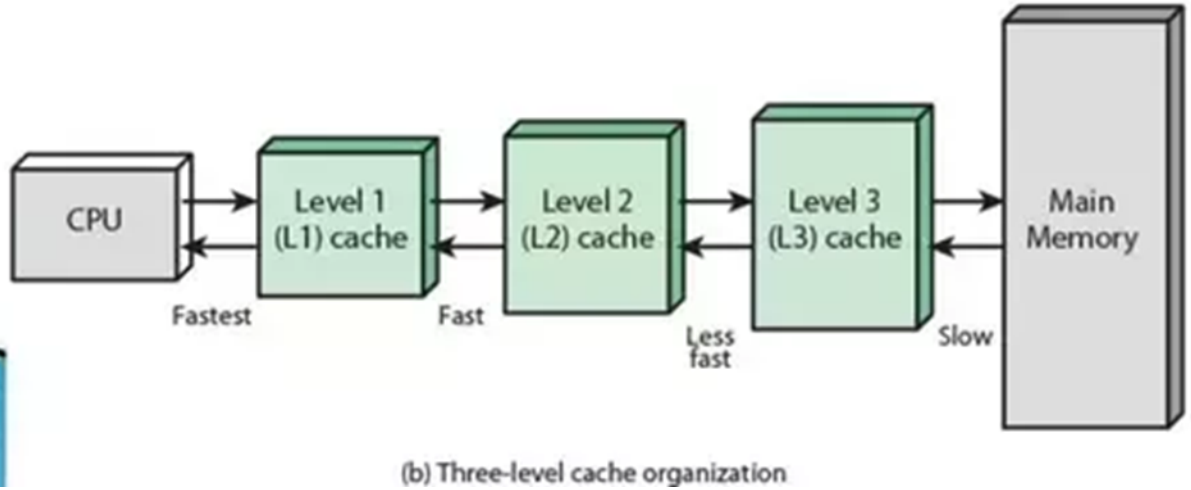
- What do we need?
  - A memory to store very large programs and to work at a speed comparable to that of the CPU.
- The reality is:
  - The larger a memory, the slower it will be;
  - The faster a memory, the greater the cost per bit.
- A solution:
  - To build a composite memory system which combines a small and fast memory with a large and slow memory, and behaves, most of the time, like a large and fast memory.
  - This two-level principle can be extended to a hierarchy of many levels.

Cache and Main Memory



(a) Single cache

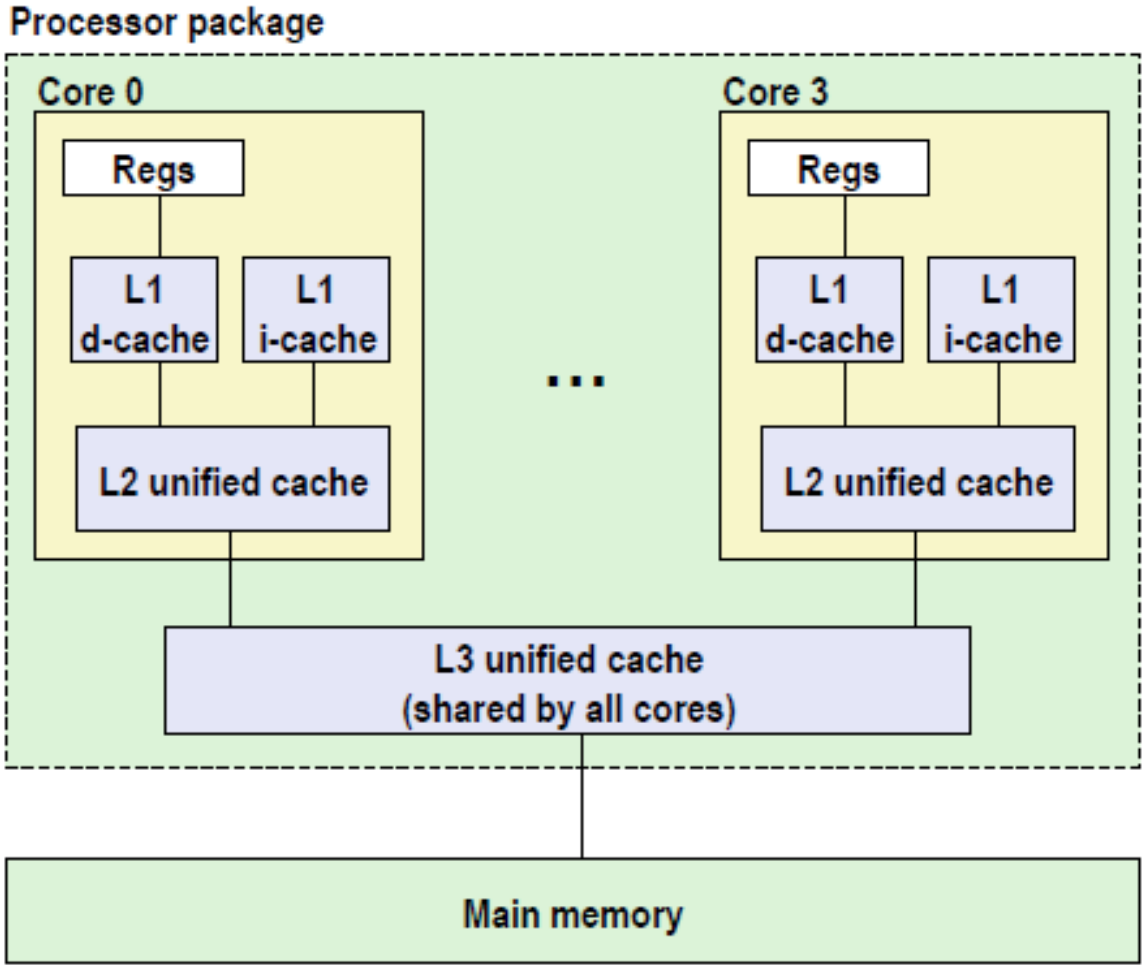
Multiple Cache Levels



(b) Three-level cache organization

- To build a composite memory system which combines a small and fast memory with a large and slow memory, and behaves, most of the time, like a large and fast memory.

Intel Core i7 Cache Hierarchy



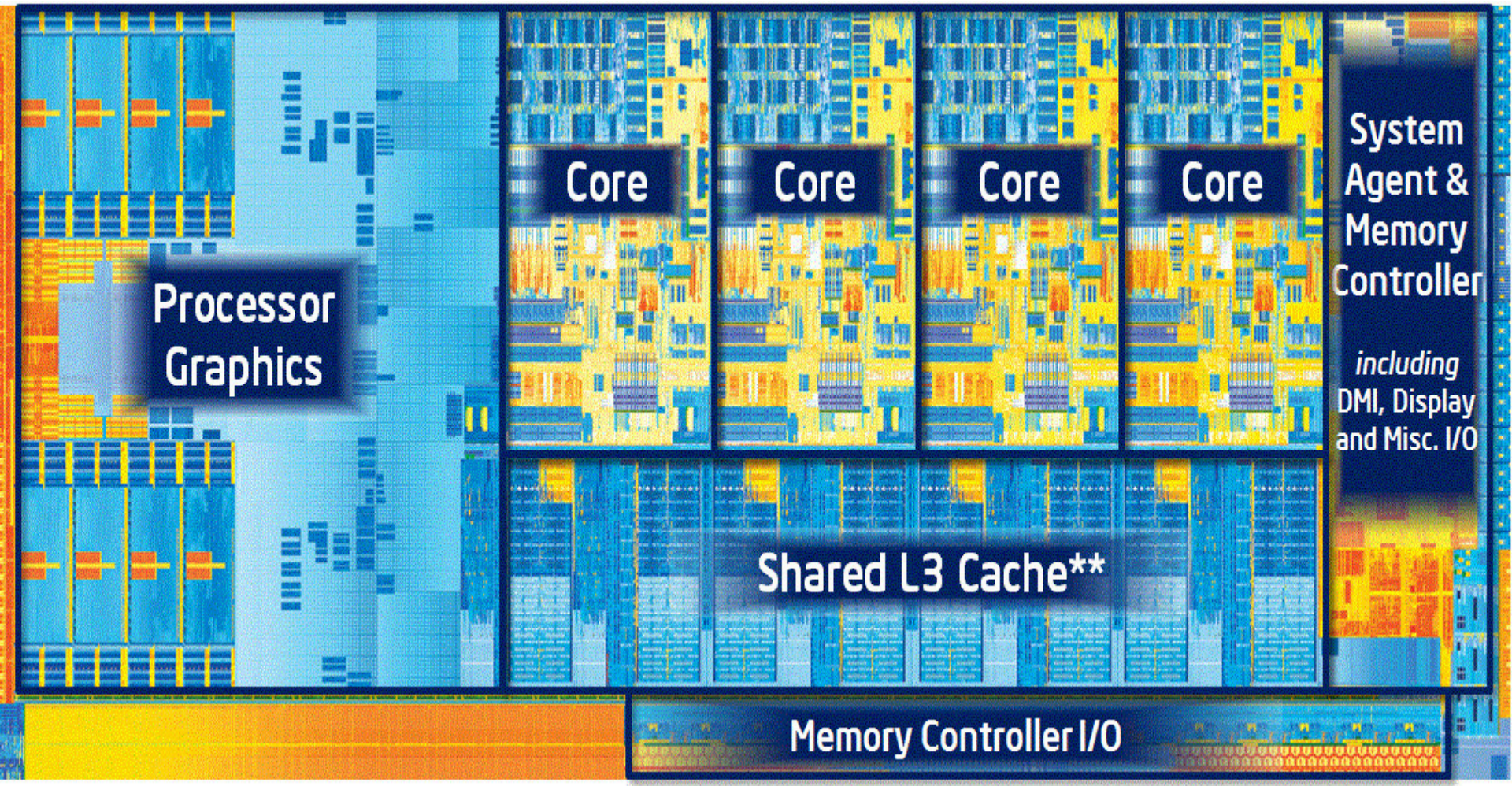
L1 i-cache and d-cache:
32 KB, 8-way,
Access: 4 cycles

L2 unified cache:
256 KB, 8-way,
Access: 11 cycles

L3 unified cache:
8 MB, 16-way,
Access: 30-40 cycles

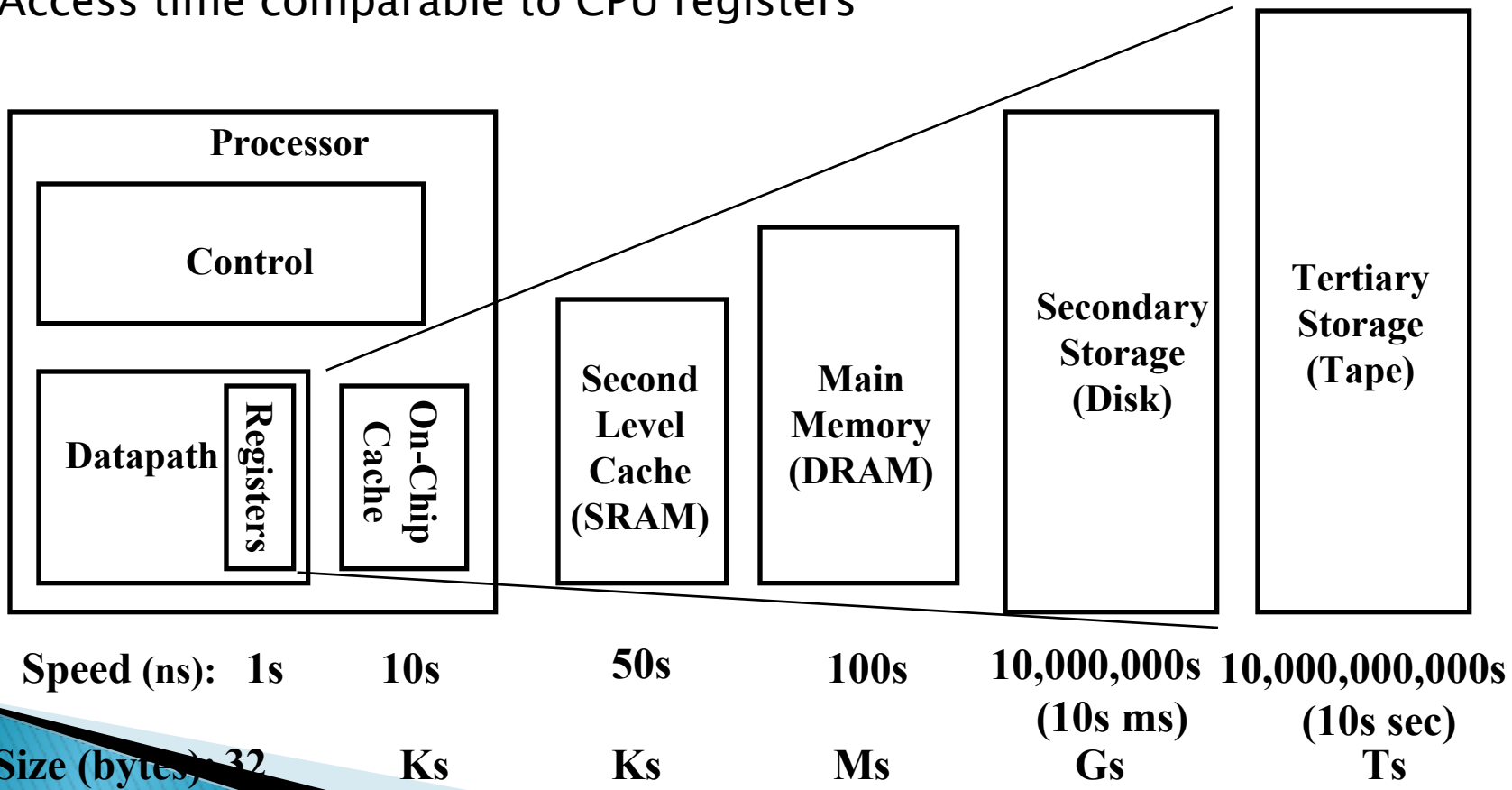
Block size: 64 bytes for all caches.

http://sabercomlogica.com/en/ebook/a-case-study-intel-nehalem-i7-coherence-in-cache/

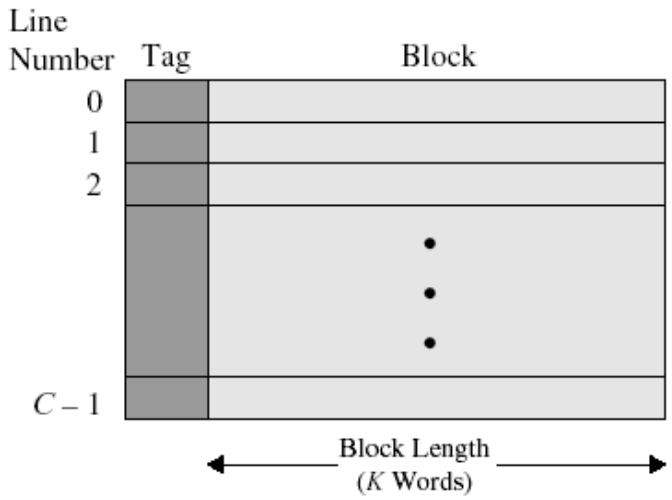

Processor Graphics
Core
Core
Core
Core
System Agent & Memory Controller
including DMI, Display and Misc. I/O
Shared L3 Cache**
Memory Controller I/O

Memory Hierarchy

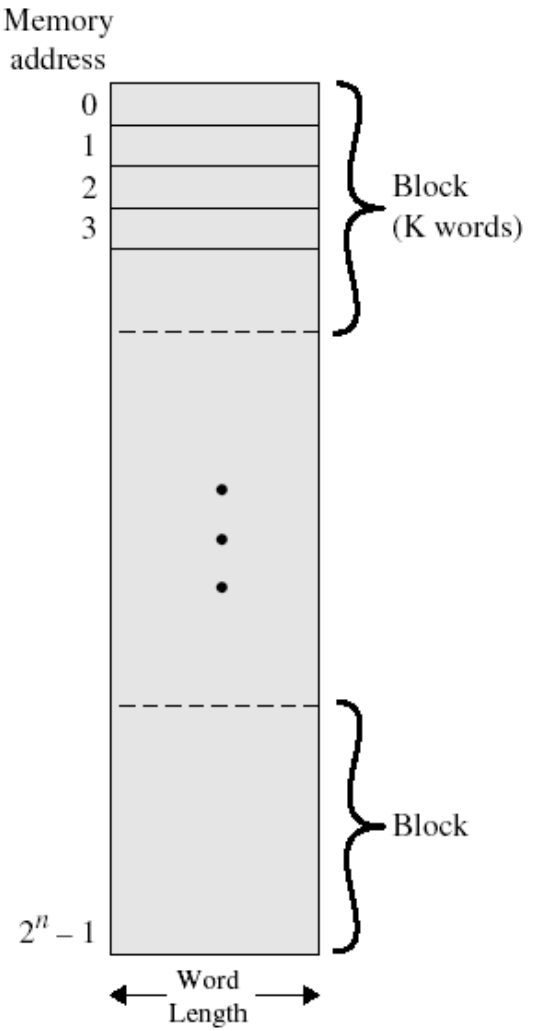
- A cache is a very fast memory which is put between the main memory and the CPU, and used to hold segments of program and data of the main memory.
- May be located on CPU chip or module
- Access time comparable to CPU registers



Cache/Memory Structure

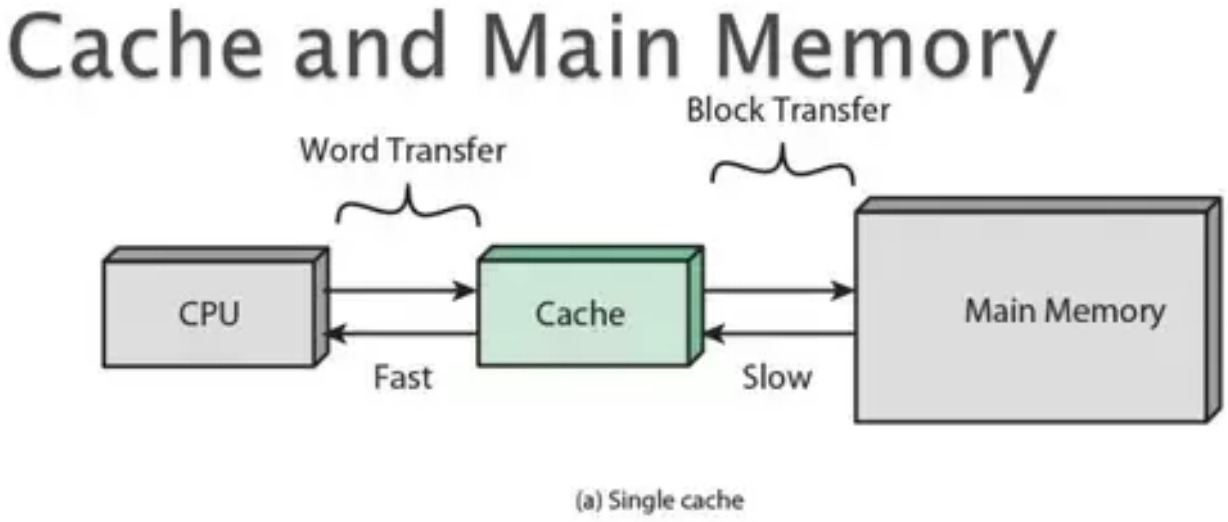


(a) Cache



(b) Main memory

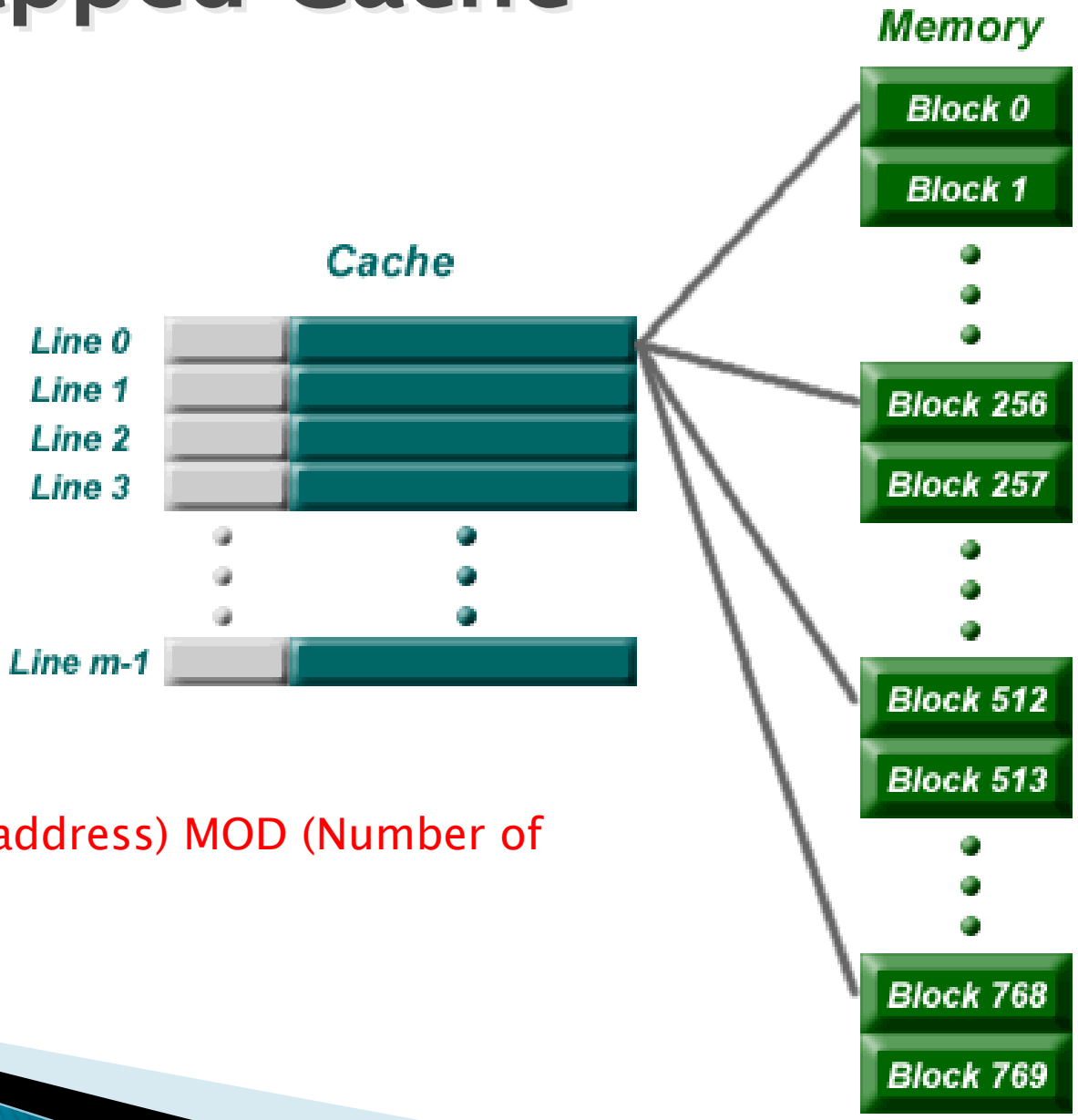
How Cache improves performance? When Instructions & data is still being fetched from Main Memory (if not found in Cache)?



Memory mapping- Where a block can be placed?

- **Direct Mapped:** Each block has only one place that it can appear in the cache.
- Location in Cache=
- (Main memory Block address) MOD (Number of blocks in cache)
- **Fully Associative**: Each block can be placed anywhere in the cache.
- **Set Associative**: Each block can be placed in a restricted set of places in the cache.
  - If there are n blocks in a set, the cache placement is called n-way set associative

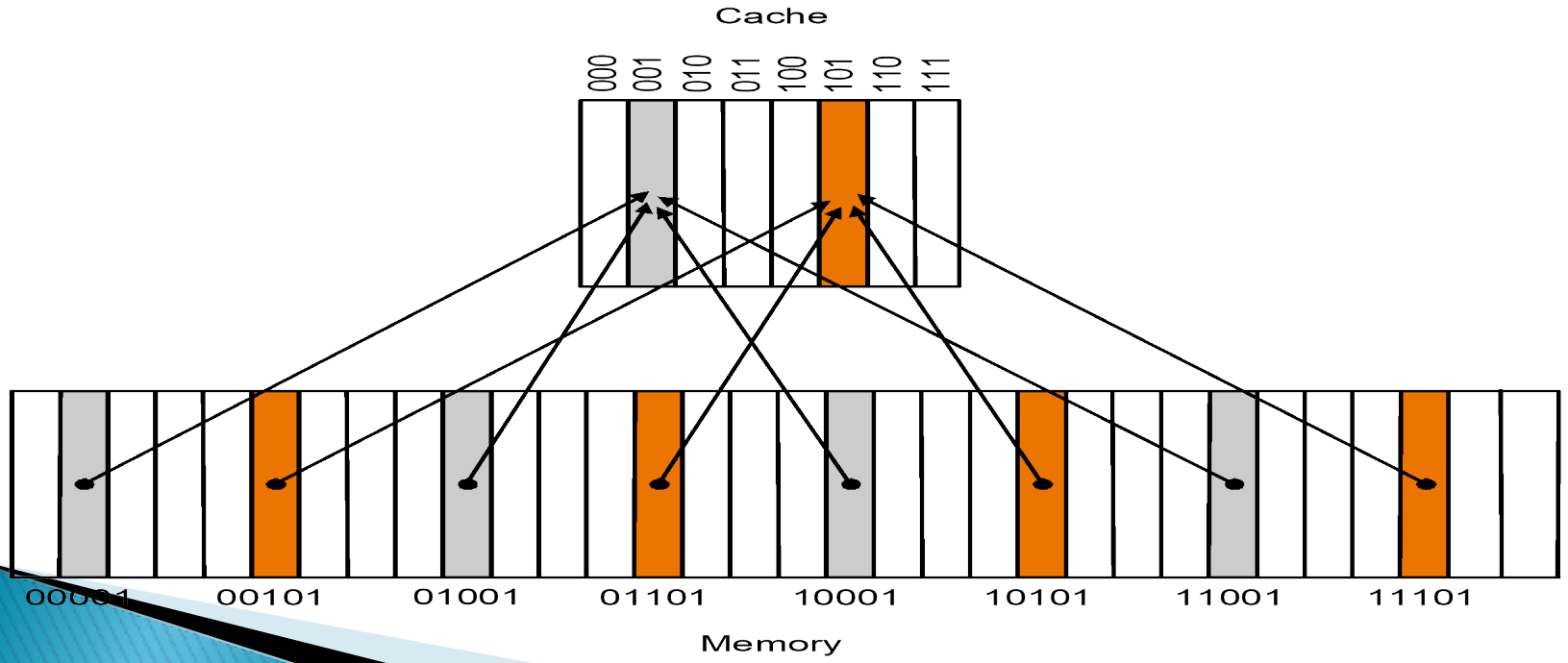
Direct Mapped Cache



Location in Cache=
(Main memory Block address) MOD (Number of blocks in cache)

Direct Mapping

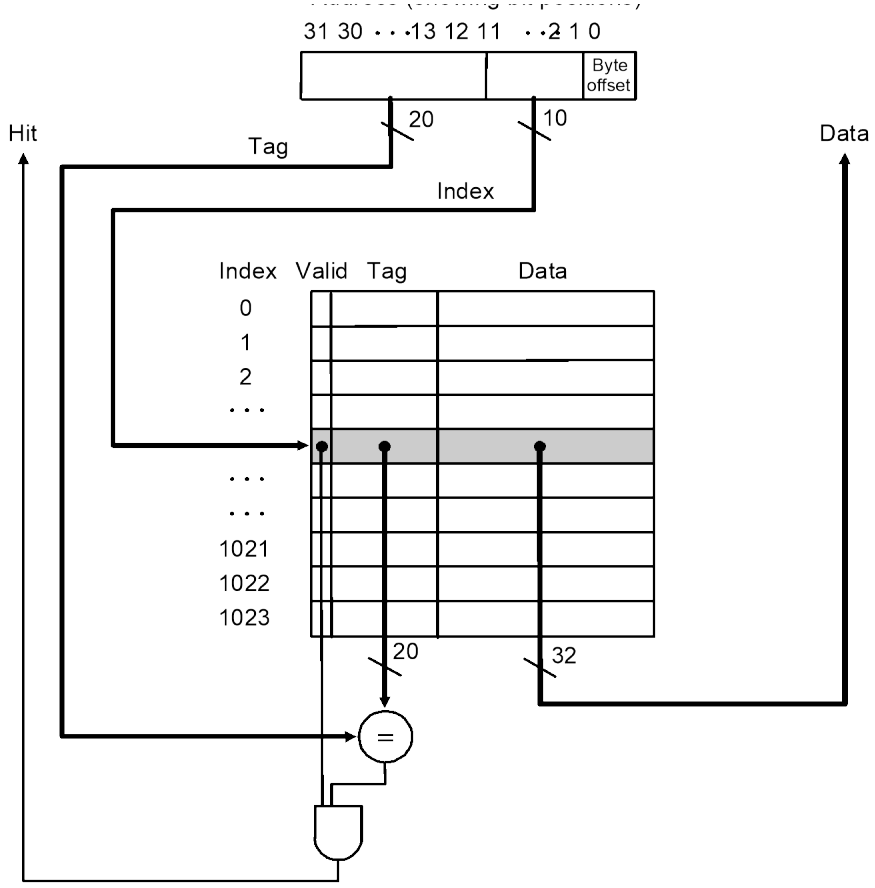
- Assuming 8 bytes Cache and 32 Bytes Main Memory.
- Mapping: memory mapped to one location in cache:
  (Block address) mod (Number of blocks in cache)
- Multiple blocks from main memory tend to occupy same memory location in cache. Only one block can be present at a time. Older blocks get replaced.



Direct Mapping

Locating data:

- Consider 1024 words cache having one word 4 bytes long.
- CPU generates 32 bits address. (MIPS-32)
- 10 bits required for Index, 2 bits for Byte offset (4- bytes in word) and 20 bits are left for Tag.
- We need to address 1024 (2^{10}) words
- We can have any of 2^{20} words at one cache location
- Valid bit (dirty bit) indicates whether an entry contains a valid address or not.
- Tag bits is usually indicated by
  - *address size – (log_2(memory size) + 2)*
  - E.g. 32 – (10 + 2) = 20

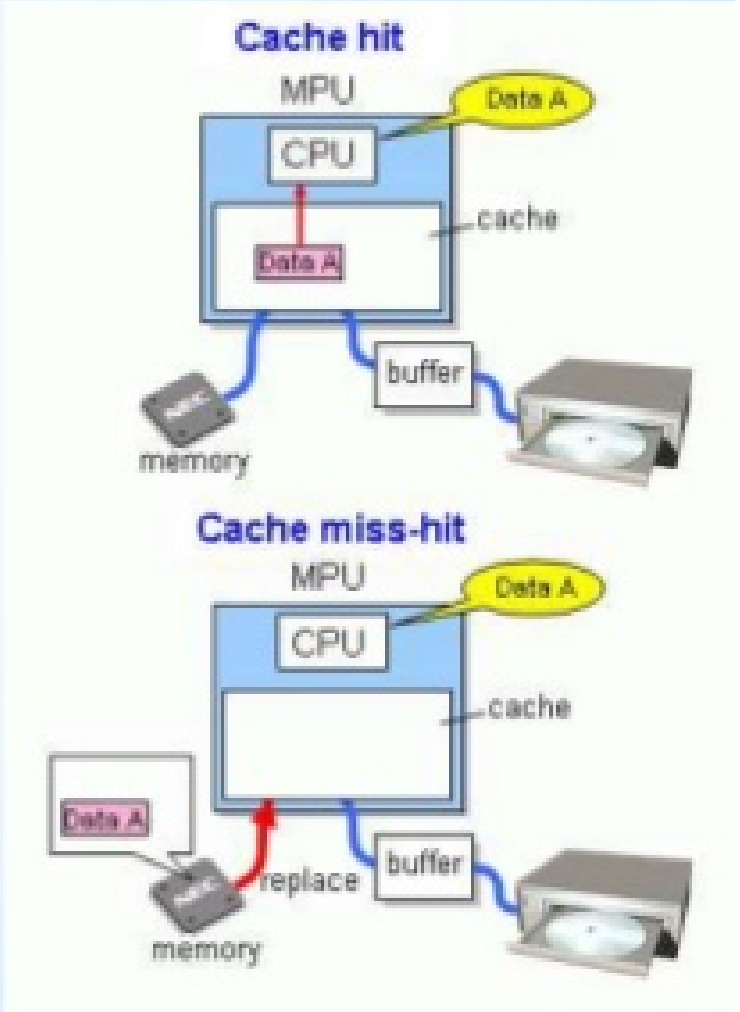


Pros n Cons

- A block from Main memory can only reside in one specific memory location in Cache.
- Multiple blocks having same index value can't be placed in Cache.
- Degrades performance drastically, if two blocks having same index but different Tag are to be accessed alternatively.
- The solution is to use Set-Associative mapping.
  - Multiple blocks having same index with different Tags can be placed in Cache.

Cache Hit/Miss

- **Cache Hit**
  - Data found in cache. Results in data transfer at maximum speed
- **Cache Miss**
  - Data not found in cache. Processor loads data from Memory and copies into cache. This results in extra delay, called miss penalty



Hit and Miss

- Focus on *any two adjacent* levels – called, *upper* (closer to CPU) and *lower* (farther from CPU) – in the memory hierarchy, because each block copy is always between two adjacent levels
- Terminology:
  - *block*: minimum unit of data to move between levels
  - *hit*: data requested is in upper level
  - *miss*: data requested is not in upper level
  - *hit rate*: fraction of memory accesses that are hits (i.e., found at upper level)
  - *miss rate*: fraction of memory accesses that are not hits
    - miss rate = 1 – hit rate
  - *hit time*: time to determine if the access is indeed a hit + time to access and deliver the data from the upper level to the CPU
  - *miss penalty*: time to determine if the access is a miss + time to replace block at upper level with corresponding block at lower level + time to deliver the block to the CPU

Locality

Principle of Locality:

- Programs tend to reuse data and instructions near those they have used recently, or that were recently referenced themselves.
- **Temporal locality:** Recently referenced items are likely to be referenced in the near future.
- **Spatial locality:** Items with nearby addresses tend to be referenced close together in time.

Locality Example:

```
sum = 0;
for (i = 0; i < n; i++)
    sum += a[i];
return sum;
```

- **Data**
  - Reference array elements in succession (stride-1 reference pattern): **Spatial locality**
  - Reference `sum` each iteration: **Temporal locality**
- **Instructions**
  - Reference instructions in sequence: **Spatial locality**
  - Cycle through loop repeatedly: **Temporal locality**

Locality of Reference

- **Spatial Locality**
  - Given an access to a particular location in memory, there is a high probability that other accesses will be made to either that or neighboring locations with the lifetime of the program.
- **Temporal Locality**
  - This is complementary to spatial locality. Given a sequence of reference to *n* locations, there is a high probability that references following this sequence will be made into the sequence. Elements of the sequence will again be referenced during the lifetime of the program.