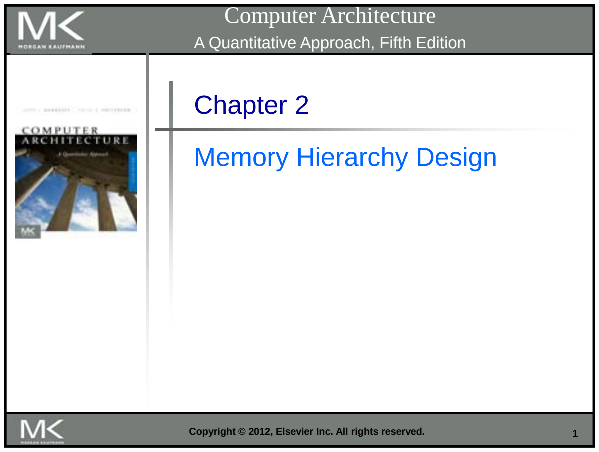
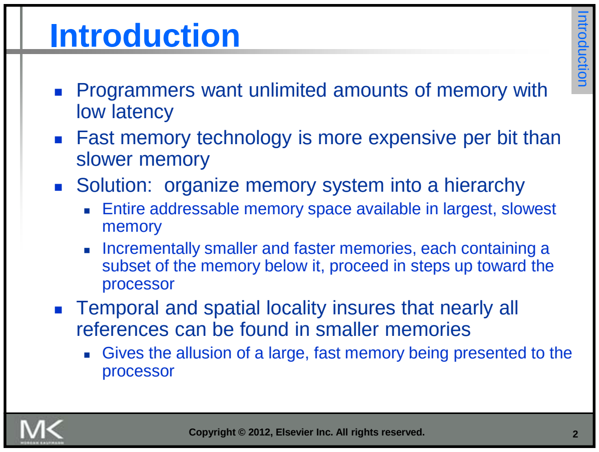


Computer Architecture

A Quantitative Approach, Fifth Edition

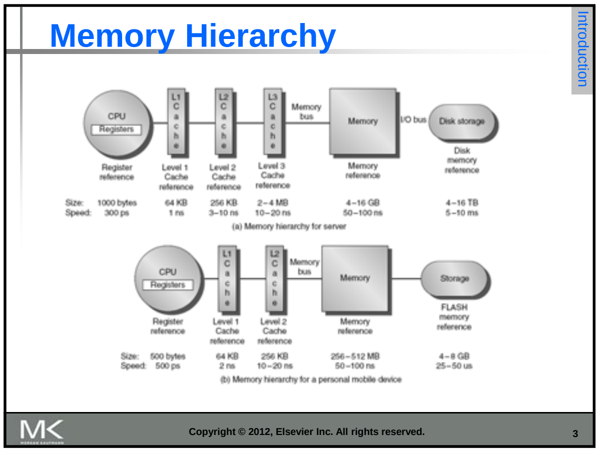
Chapter 2

Memory Hierarchy Design



Introduction

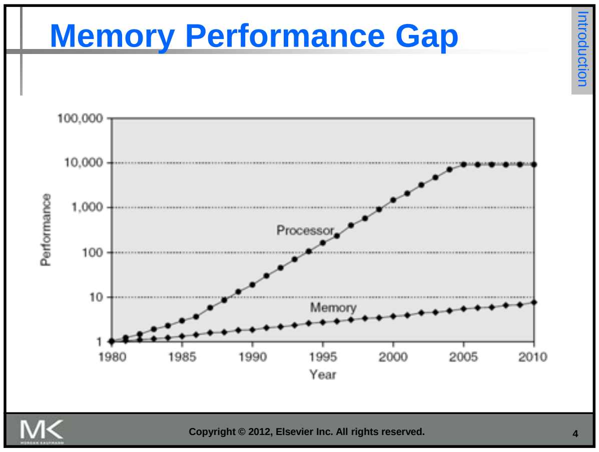
- Programmers want unlimited amounts of memory with low latency
- Fast memory technology is more expensive per bit than slower memory
- Solution: organize memory system into a hierarchy
  - Entire addressable memory space available in largest, slowest memory
  - Incrementally smaller and faster memories, 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



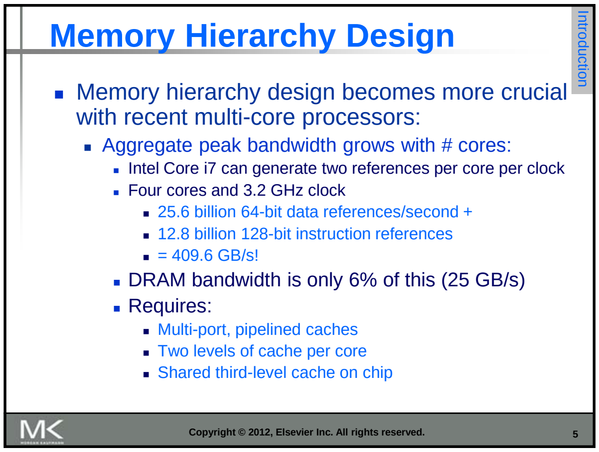
Memory Hierarchy

(a) Memory hierarchy for server

(b) Memory hierarchy for a personal mobile device

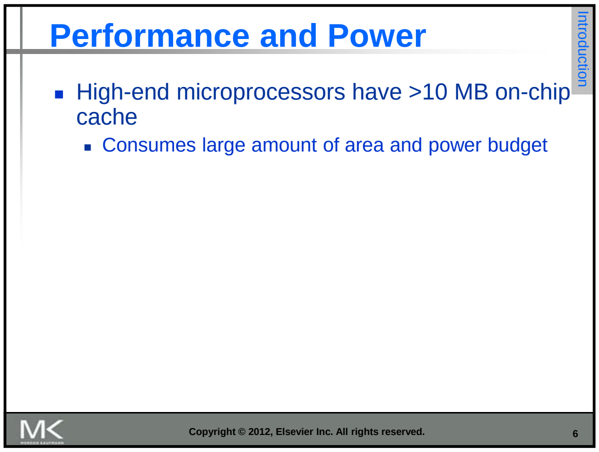


Memory Performance Gap



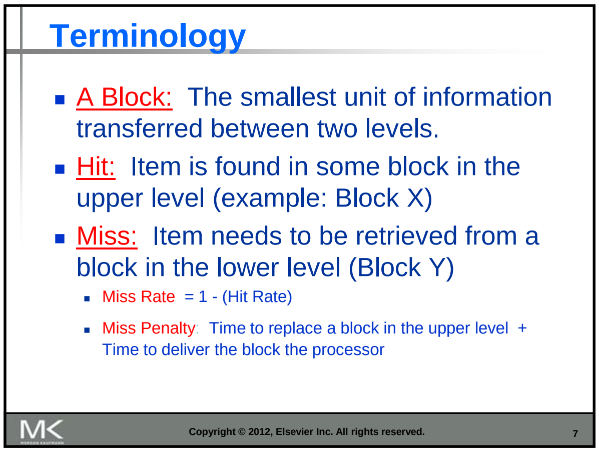
Memory Hierarchy Design

- 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
      - = 409.6 GB/s!
    - DRAM bandwidth is only 6% of this (25 GB/s)
    - Requires:
      - Multi-port, pipelined caches
      - Two levels of cache per core
      - Shared third-level cache on chip



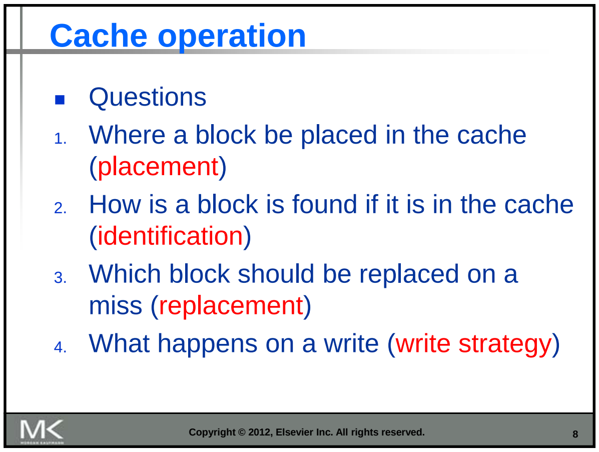
Performance and Power

- High-end microprocessors have >10 MB on-chip cache
  - Consumes large amount of area and power budget



Terminology

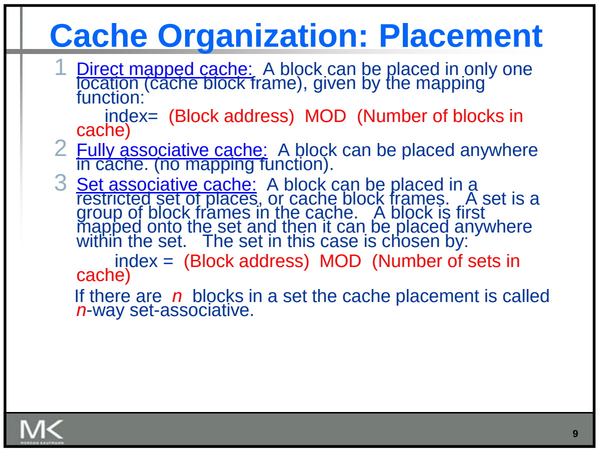
- A Block: The smallest unit of information transferred between two levels.
- Hit: Item is found in some block in the upper level (example: Block X)
- Miss: Item needs to be retrieved from a block in the lower level (Block Y)
  - Miss Rate = 1 - (Hit Rate)
  - Miss Penalty: Time to replace a block in the upper level + Time to deliver the block the processor



Cache operation

- Questions

1. Where a block be placed in the cache (placement)
2. How is a block is found if it is in the cache (identification)
3. Which block should be replaced on a miss (replacement)
4. What happens on a write (write strategy)



Cache Organization: Placement

1 Direct mapped cache: A block can be placed in only one location (cache block frame), given by the mapping function:

index= (Block address) MOD (Number of blocks in cache)

2 Fully associative cache: A block can be placed anywhere in cache. (no mapping function).

3 Set associative cache: A block can be placed in a restricted set of places, or cache block frames. A set is a group of block frames in the cache. A block is first mapped onto the set and then it can be placed anywhere within the set. The set in this case is chosen by:

index = (Block address) MOD (Number of sets in cache)

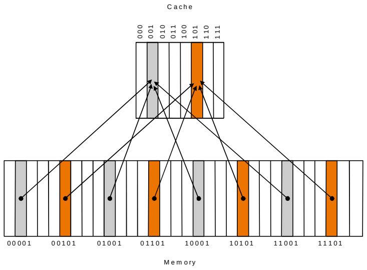
If there are *n* blocks in a set the cache placement is called *n*-way set-associative.

Cache Miss

- **Compulsory**: The very first access to a block is always a miss– Occurs even if you have an infinite cache
- **Capacity**: The cache is not big enough to hold all the blocks required for the execution of the program– A bigger cache helps
- **Conflict**: If not a fully associative, a block may be discarded and brought back again.

Cache Organization: Placement

- Direct mapped Cache



Placement: DM

1K = 1024 Blocks
Each block = one word

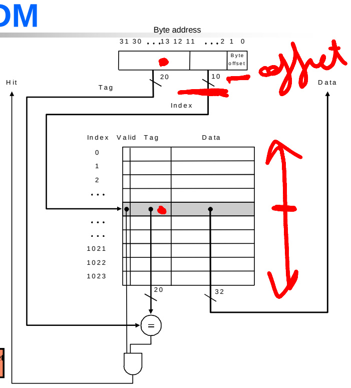
Can cache up to
2^{32} bytes = 4 GB
of memory

Mapping function:

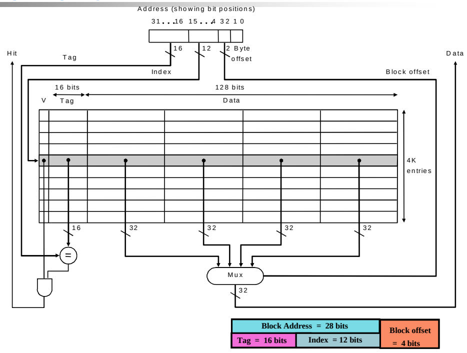
Cache Block frame number = (Block address) MOD (1024)

i.e. index field or
10 low bit of block address

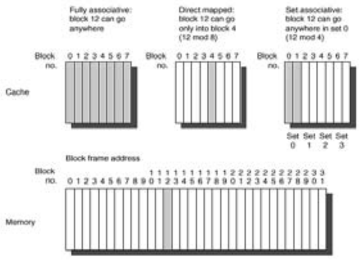
Block Address = 30 bits		**Block offset = 2 bits**
Tag = 20 bits	**Index = 10 bits**	



Placement DM

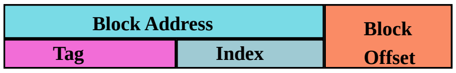


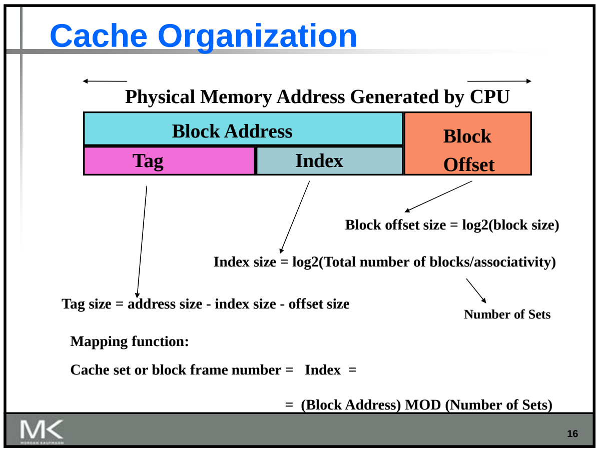
Cache Organization



Cache Organization

- Each block frame in cache has an address tag.
- The tags of every cache block that might contain the required data are checked in parallel.
- A valid bit is added to the tag to indicate whether this entry contains a valid address.
- The address from the CPU to cache is divided into:
  - A block address, further divided into:
    - An index field to choose a block set in cache.
    - (no index field when fully associative).
    - A tag field to search and match addresses in the selected set.
  - A block offset to select the data from the block.





Cache Organization

Physical Memory Address Generated by CPU

Block Address		Block Offset
Tag	Index	

Block offset size = log2(block size)

Index size = log2(Total number of blocks/associativity)

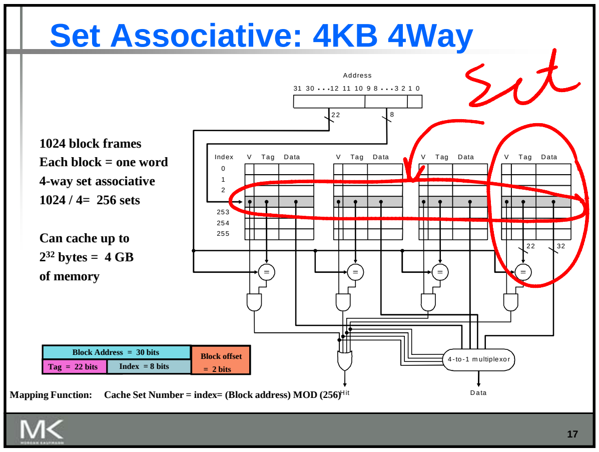
Number of Sets

Tag size = address size - index size - offset size

Mapping function:

Cache set or block frame number = Index =

= (Block Address) MOD (Number of Sets)



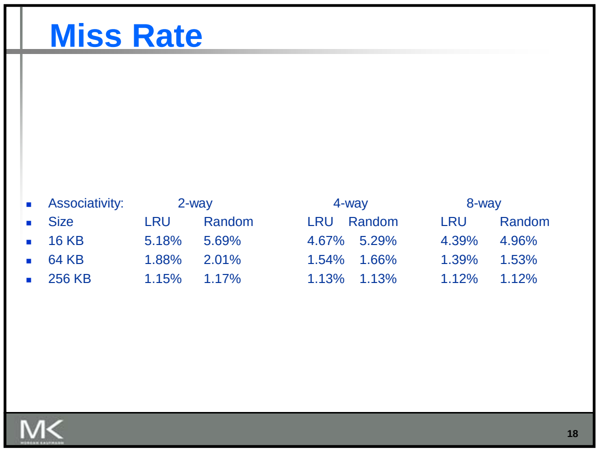
Set Associative: 4KB 4Way

1024 block frames
Each block = one word
4-way set associative
1024 / 4= 256 sets

Can cache up to
2^{32} bytes = 4 GB
of memory

Block Address = 30 bits		Block offset = 2 bits
Tag = 22 bits	Index = 8 bits	

Mapping Function: Cache Set Number = index= (Block address) MOD (256)



Miss Rate

Associativity:	2-way		4-way		8-way	
Size	LRU	Random	LRU	Random	LRU	Random
16 KB	5.18%	5.69%	4.67%	5.29%	4.39%	4.96%
64 KB	1.88%	2.01%	1.54%	1.66%	1.39%	1.53%
256 KB	1.15%	1.17%	1.13%	1.13%	1.12%	1.12%