

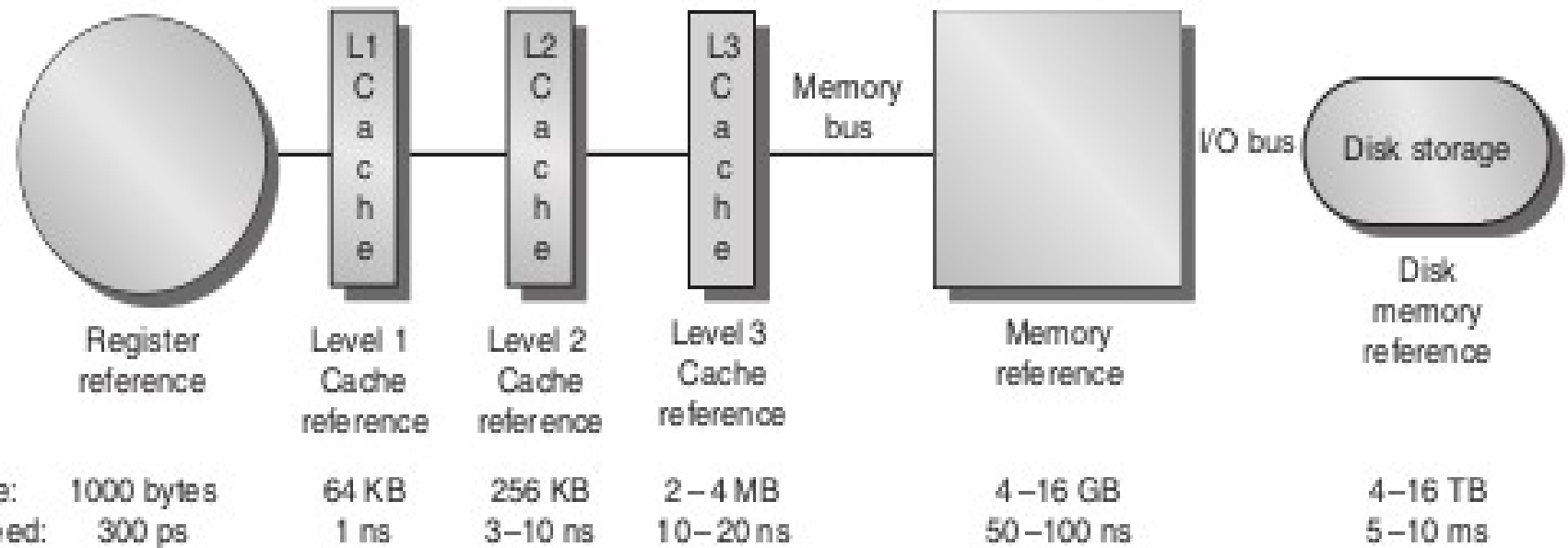


MEMORY HIERARCHY

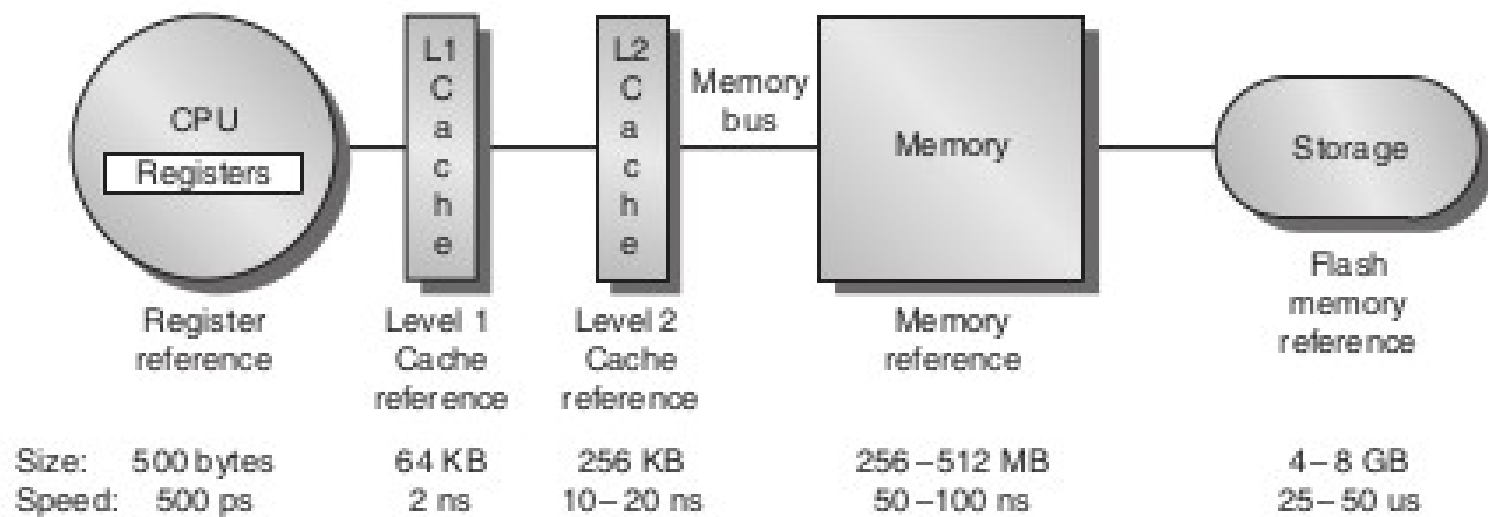


Contents

- Introduction to memory hierarchy
- Why memory hierarchy
- Caching



(a) Memory hierarchy for server



(b) Memory hierarchy for a personal mobile device


Motivation

- The design and implementation of a computer memory system impacts the system's performance as the design and implementation of the processor

-

Why Memory hierarchy



- No single memory device possesses all the ideal characteristics of a memory device
 - Therefore: We intelligently combine multiple types of memory devices in one system hoping to obtain the advantages of each while minimizing their disadvantages
- 

characteristics of an ideal memory device

- Low cost
 - Dollars per bit, byte, Gigabyte, Terabyte
-

characteristics of an ideal memory device

- High speed
 - Shortening the access time and cycle time
 - Access time: Time to read/write a single piece of information
 - cycle time: Time bwn repetitive reads or writes
 - Sometimes longer than access time due to overheads
 - The idea is to make: $\text{access time} \leq \text{cpu time}$

characteristics of an ideal memory device

- High Density
 - Huge amount of information in a small physical space
 - Infinite amount of information on a finite space or a finite amount of information in an infinitesimal space: Certainly impossible
 - The closer the better

characteristics of an ideal memory device

- Non-Volatile
 - Murphy's law on power outages: "If anything can go wrong, it will"
- Read/Write capable
 - At anytime
 - Not ROM
 - Not read mostly memory like flash memory
 - Writes are more costly than reads

characteristics of an ideal memory device

- Low power
 - Even non-volatile memories require power for information read or write
 - A considerable issue when heating is a problem or when a system must run off batteries


characteristics of an ideal memory device


- Durability
 - Lasts forever!
 - Manufacturers may provide an estimate of MTBF (Mean Time Before Failure) Or
 - Total number of reads and writes it should be able to perform before failing
 - Ability to survive various forms of abuse: Impact, temperature, Humidity
 - Memory technologies employing moving mechanical parts have shorter life spans

characteristics of an ideal memory device

- Removable
 - Transporting memory and contents from one system to another
 - Desirable for sharing and backup
 - Undesirable in certain cases - Physical security

Characteristics of real memory devices



- Semiconductor memories
 - Advantage of speed - occupies the main memory space
 - DRAM: Highest information density
 - Read/write
 - Low cost per gigabyte
- 

characteristics of an ideal memory device

- SRAM: Highest speed
 - Read/write
 - Information stored as states of latches or flip flops rather than capacitors
 - Less dense than DRAM
 - More expensive

characteristics of an ideal memory device

- ROMs - PROMs & EPROMs
- Cost and density comparable to SRAM
- Operate at DRAM speed or lower
- Not writable

characteristics of an ideal memory device

- EEPROMs + Flash memory : Read mostly
- Re-writable in-circuit
- $t(\text{writes}) > t(\text{reads})$
- Expensive than most other semiconductor memories

characteristics of an real memory device

- Limited number of writes
 - To cope with such limits, most flash products include a controller to spread the writes by remapping blocks that have been written many times to less trodden blocks.
 - This technique is called **wear leveling**.
 - With wear leveling, personal mobile devices are very unlikely to exceed the write limits in the flash.

characteristics of an ideal memory device

- Magnetic memories
 - Much older than semiconductor memories
 - 1950's mainframes
 - Non-volatile
 - Access time in micro seconds
 - Went out of favor after faster semiconductor memories became cost-competitive

characteristics of an ideal memory device

- MRAM - Magnetoresistive RAM
- Has the potential to eventually replace DRAM!
- Non-volatile: Bits stored as magnetic fields
- Will enable the development of instant-on computers
- Retain OS, apps and data in main memory even when the system is turned off
- Density, speed and cost still major constraints to its development

characteristics of an ideal memory device


- Optical memories
 - Low cost
 - Non-volatile
 - High density 50GB
 - In-affected by magnetic fields
 - Too much slow

Why Memory hierarchy

- Therefore the current solution is to have a hierarchical memory system

Why Memory hierarchy



- A modern high end processor such as the Intel Core i7 can generate two data memory references per core each clock cycle; with four cores and a 3.2 GHz clock rate,
 - The i7 can generate a peak of 25.6 billion 64-bit data memory references per second,
 - In addition to a peak instruction demand of about 12.8 billion 128-bit instruction references;
 - This is a total peak bandwidth of 409.6 GB/sec!
- 

Why Memory hierarchy


- In contrast, the peak bandwidth to DRAM main memory is only 6% of this (25 GB/sec).



CACHE

- Its ok to have a cache that is 0.1% or less of the size of the main memory
- As long as it contains the right 0.1% of the information that is likely to be used in the near future
- **Challenge:** Determining which 0.1% to keep in the cache at any given time

Why Memory hierarchy



- Locality of reference principle
 - Programs tend to access code and data that have recently been accessed or which are near code or data that have recently been accessed.
- 

- 
- Three types of localities
 - Temporal locality: Time oriented
 - Spatial locality: location/space oriented
 - Sequential locality: special case of spatial locality
- 

Four basic questions

- Q1: Where can a block be placed in a cache?
- If each block has only one place it can appear in the cache, the cache is said to be **direct mapped**.
 - The mapping is usually $(\text{Block address}) \bmod (\text{Number of blocks in cache})$
- If a block can be placed anywhere in the cache, the cache is said to be **fully associative**.
- If a block can be placed in a restricted set of places in the cache, the cache is **set associative**.

Why Memory hierarchy

- A set is a group of blocks in the cache.
- A block is first mapped onto a set, and then the block can be placed anywhere within that set.
- The set is usually chosen by bit selection; that is, $(\text{Block address}) \bmod (\text{Number of sets in cache})$

Why Memory hierarchy



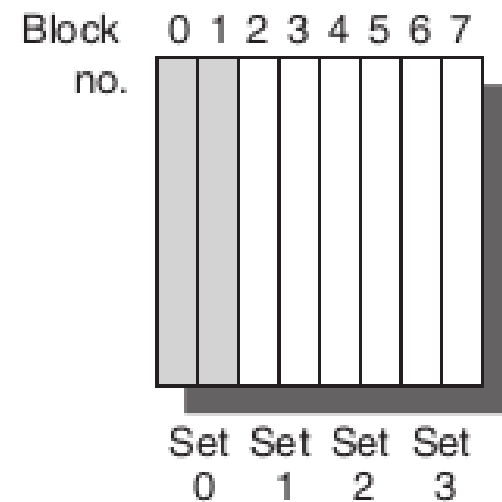
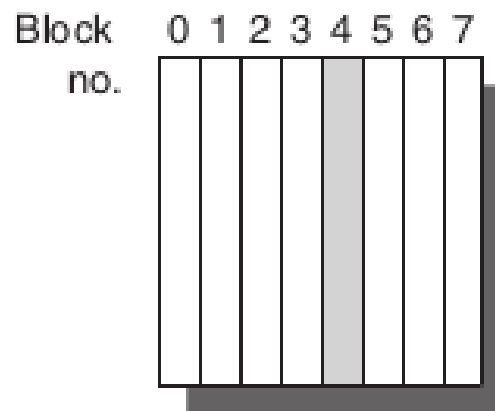
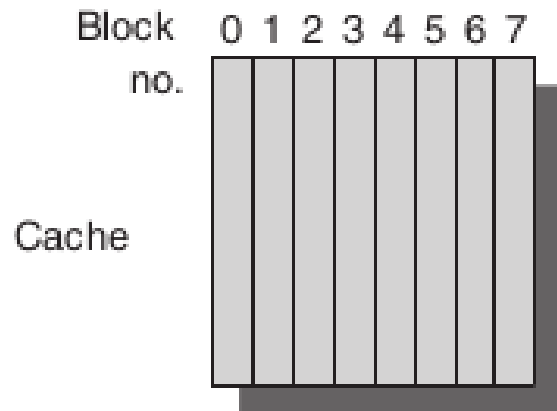
- If there are n blocks in a set, the cache placement is called n -way set associative.
- The vast majority of processor caches today are **direct mapped, two-way set associative, or four-way set associative**

Why Memory hierarchy

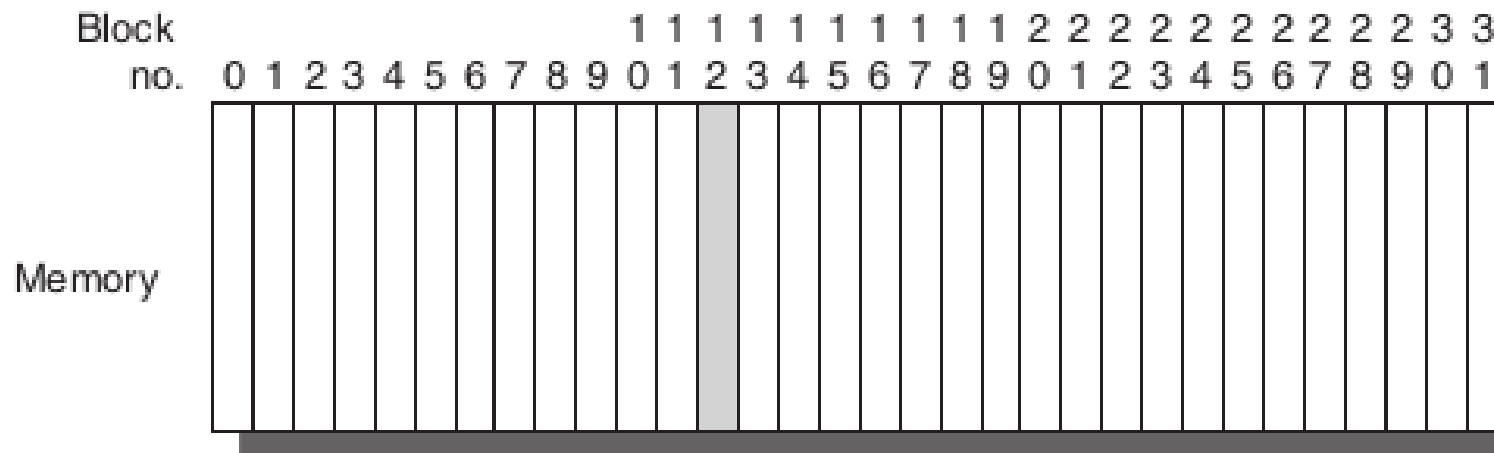
Fully associative:
block 12 can go
anywhere

Direct mapped:
block 12 can go
only into block 4
($12 \text{ MOD } 8$)

Set associative:
block 12 can go
anywhere in set 0
(12 MOD 4)




Block frame address




Why Memory hierarchy



- Q2: How Is a Block Found If It Is in the Cache?
 - Caches have an address tag on each block frame that gives the block address.
 - The tag of every cache block that might contain the desired information is checked to see if it matches the block address from the processor.
 - As a rule, all possible tags are searched in parallel because speed is critical.
- 

Why Memory hierarchy




- There must be a way to know that a cache block does not have valid information.
 - The most common procedure is to add a valid bit to the tag to say whether or not this entry contains a valid address.
 - If the bit is not set, there cannot be a match on this address.
- 



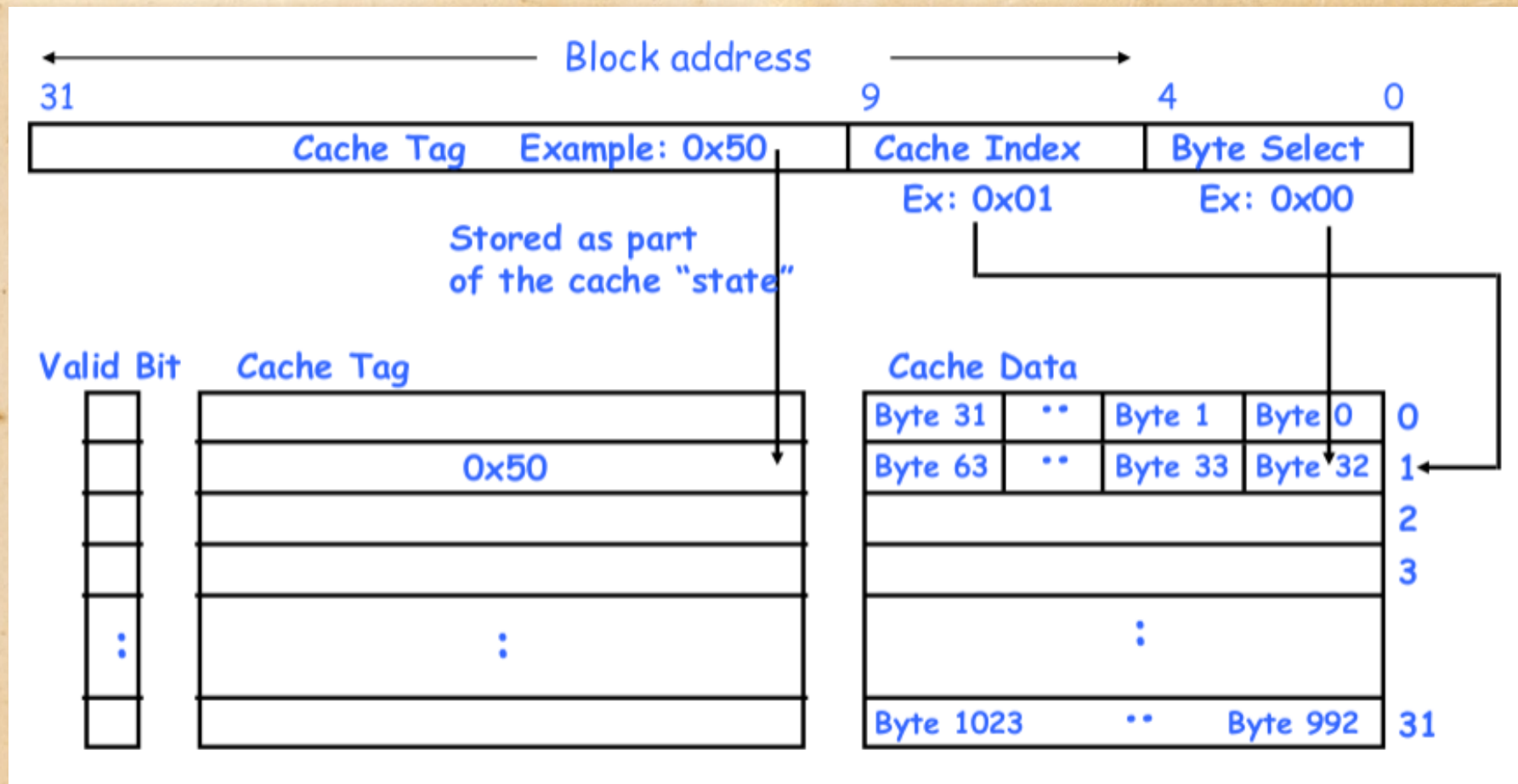
Block address		Block offset
Tag	Index	

Why Memory hierarchy



- The block offset field selects the desired data from the block, the index field selects the set, and the tag field is compared against it for a hit
- 

Why Memory hierarchy





Why Memory hierarchy



- Although the comparison could be made on more of the address than the tag, there is no need because of the following:
 - 1. The offset should not be used in the comparison - the entire block is either present or not
 - 2. Checking the index is redundant - it was used to select the set to be checked
- This optimization saves hardware and power by reducing the width of memory size for the cache tag.
- Increasing associativity reduces the size of the index tag until its no more - fully associative cache


Why Memory hierarchy






- Q3: Which Block Should Be Replaced on a Cache Miss?
 - When a miss occurs, the cache controller must select a block to be replaced with the desired data
 -
- 

- 
- Only one block frame is checked for a hit, and only that block can be replaced - DIRECT-MAPPED - No choice - so simple
 - For fully assoc and set assoc there are many blocks to choose from a miss

- 
- There are three primary strategies employed for selecting which block to replace
 - Random: To spread allocation uniformly candidate blocks are randomly selected
 - Least recently used (LRU) - relying on the past to predict the future - Accesses to blocks are recorded-relies on a corollary of locality: If recently used blocks are likely to be used again, then a good candidate for disposal is the **least recently used block**.
- 


- 
- First in, first out (FIFO) - LRU can be complicated to calculate, this approximates LRU by determining the oldest block rather than the LRU.



- 
- Q4: What Happens on a Write?
 - Reads dominate processor cache accesses.
 - All instruction accesses are reads, and most instructions don't write to memory.
 - hence: The common case: Fast means optimizing cache for reads
 - However high-performance designs cannot neglect the speed of writes.
- 


- 
- The block can be read from the cache at the same time that the tag is read and compared, so the block read begins as soon as the block address is available.
 - For writes, Modifying a block cannot begin until the tag is checked to see if the address is a hit.
 - Because tag checking cannot occur in parallel, writes normally take longer than reads



- Reads can access more bytes than necessary without fear; processor specifies the size of the write; usually 1 and 8 bytes



-


- 
- The write policies often distinguish cache designs.
 - There are two basic options when writing to the cache



- 
- Write-through—The information is written to both the block in the cache and to the block in the lower-level memory.
 - Write-back—The information is written only to the block in the cache.
 - The modified cache block is written to main memory only when it is replaced
- 



- 
- Dirty bit: its a status bit indicates whether the block is dirty ; modified while in the cache OR
 - clean: not modified - If it is clean, the block is not written back on a miss, since identical information to the cache is found in lower levels.



- 
- With write-back,
 - Writes occur at the speed of the cache memory, and multiple writes within a block require only one write to the lower-level memory.
 - Since some writes don't go to memory, write-back uses less memory bandwidth, making write-back attractive in multiprocessors.
- 

- 
- Since write-back uses the rest of the memory hierarchy and memory interconnect less than write-through, it also saves power, making it attractive for embedded applications.
- 

- 
- Write-through is easier to implement than write-back.
 - Write-through also has the advantage that the next lower level has the most current copy of the data, which simplifies data coherency.
 - Data coherency is important for multiprocessors and for I/O,

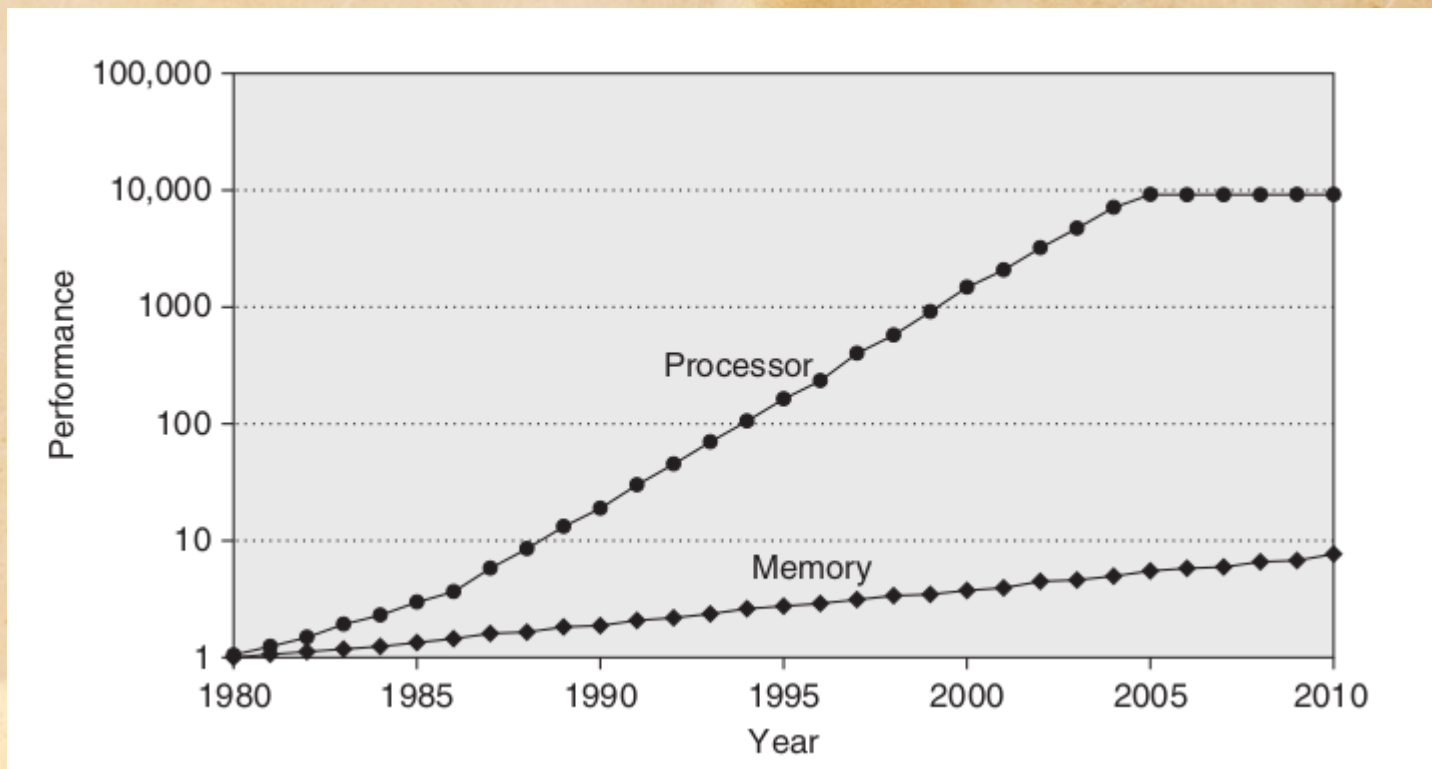
- 
- A common optimization to reduce **write stalls** is a write buffer, which allows the processor to continue as soon as the data are written to the buffer, thereby overlapping processor execution with memory updating.
- 

- 
- Since the data are not needed on a write, there are two options on a write miss:
 - Write allocate—The block is allocated on a write miss, followed by the write hit actions.
 - In this natural option, write misses act like read misses.
- 

- 
- No-write allocate—This apparently unusual alternative is write misses do not affect the cache. Instead, the block is modified only in the lower-level memory.
- 

Summary

- Why memory Hierarchy? No ideal memory device
- Why memory hierarchy? Processor - Memory performance gap



Summary

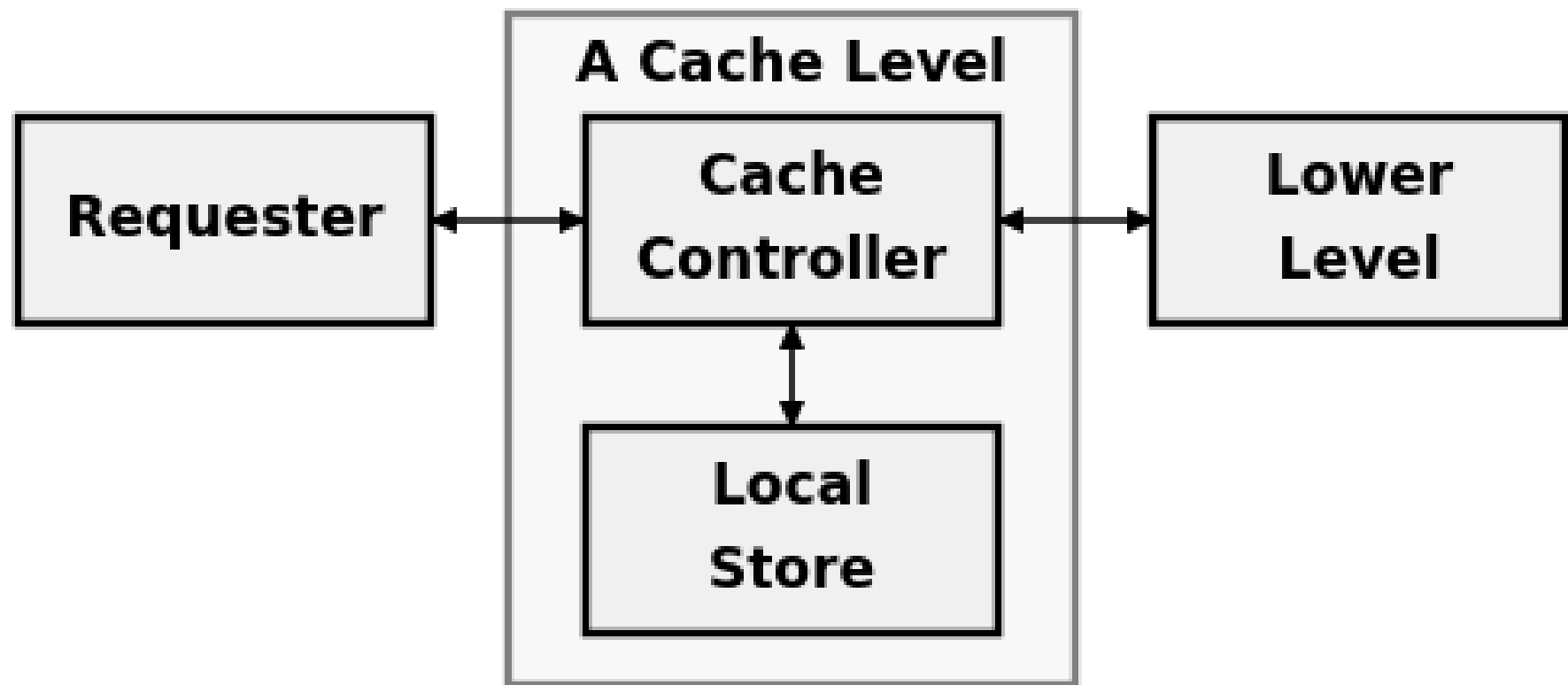
- Caching and Locality principle
- Cache mapping:
- How to place a block in cache
 - $(\text{Block address}) \bmod (\text{Number of blocks in cache})$
 - $(\text{Block address}) \bmod (\text{Number of sets in cache})$
 - $(\text{Block address}) \bmod (1)$
 - $(\text{Block address}) \bmod (\text{Number of sets in cache}=n)$: n way set assoc

Summary

- The first optimization: Comparison made on a tag
- The 3 replacement policies: Random,LRU,FIFO
- Writes are complicated than reads
- The two write policies: Write back,Write through.
- Write stall
- Write miss --> write allocate OR no write allocate



Lecture 3:


Cache performance and optimization, Virtual
memory & Memory technologies



- Cache controller's algorithm in order to take advantage of both temporal and spatial locality

```
if the requested data is in the local store // cache hit, cache access time  
    return the requested data  
else // cache miss, lower-level access time  
    get the requested and nearby data from the lower level  
    save it in the local store  
    return the requested data
```


- 
- Hit rate: The fraction or percentage of accesses that result in a hit
 - Miss rate: The fraction or percentage of accesses that result in a miss
 - Therefore: $\text{hit rate} + \text{miss rate} = 1.0$ (100%).
 - Miss penalty: The difference between lower level access time and cache access time
- 





effective-access-time = hit-rate * cache-access-time
+ miss-rate * lower-level-access-time



Miss penalty = lower level access time - cache access time



lower level access time = Miss penalty + cache access time


effective-access-time = hit-rate * cache-access-time + miss-rate * (Miss penalty + cache access time)
= cache-access-time + miss-rate * miss-penalty

- 
- Due to locality of reference, many requests are not passed on to the lower level store.
 - $\text{lower-level-traffic} = \text{miss-rate} * \text{requested-traffic}$


- 
- In order to make this more accurate, consideration must be given to data that is fetched from the lower level but not used by the requester
 - The requester might not need all of the data in a cache line
 - In any case, the lower level must have the throughput capacity to support the lower level traffic


- 
- What causes high miss rate? - For better cache designs
 - Just remember the 3 C's Model.
 - It sorts all the misses under the 3 C's
 - All misses fall into three simple categories
- 

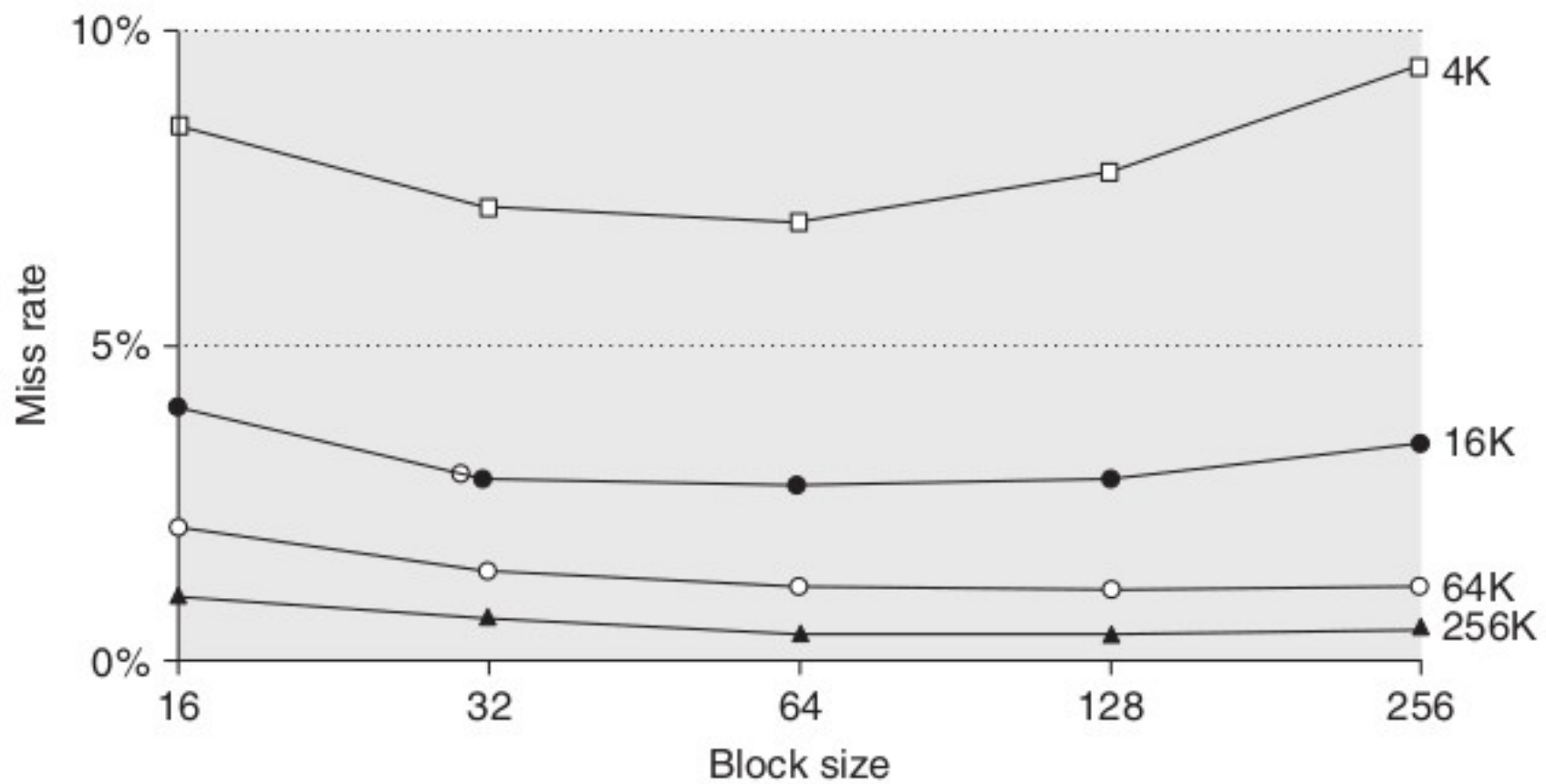
- 
- Compulsory—The very first access to a block cannot be in the cache, so the block must be brought into the cache - Doesn't matter how large your cache is
 - Capacity—If the cache cannot contain all the blocks needed during execution of a program, **capacity misses + compulsory misses** will occur because of blocks being discarded and later retrieved.
- 


- 
- Conflict—If the block placement strategy is not fully associative, **conflict misses + compulsory + capacity misses** will occur
 - Why? because a block can be discarded and later retrieved if too many blocks map to its set

Six basic cache optimizations- based on the 3 Cs




- 1) Larger block size to reduce miss rate
 - Reduce compulsory misses, but they also increase the miss penalty
 - lower the number of tags, thus can slightly reduce static power
 - Can also increase capacity or conflict misses, especially in smaller caches
 - Choosing the right block size is a complex trade-off that depends on the size of cache and the miss penalty.
- 





2) Bigger caches to reduce miss rate

- The obvious way to reduce capacity misses is to increase cache capacity
 - Drawbacks include: potentially **longer hit time** of the larger cache memory and **higher cost** and **power**.
 - Larger caches increase both static and dynamic power.
- 

3) Higher associativity to reduce miss rate


- Increasing associativity reduces conflict misses.
- Greater associativity can come at the cost of increased hit time.
- Associativity also increases power consumption.

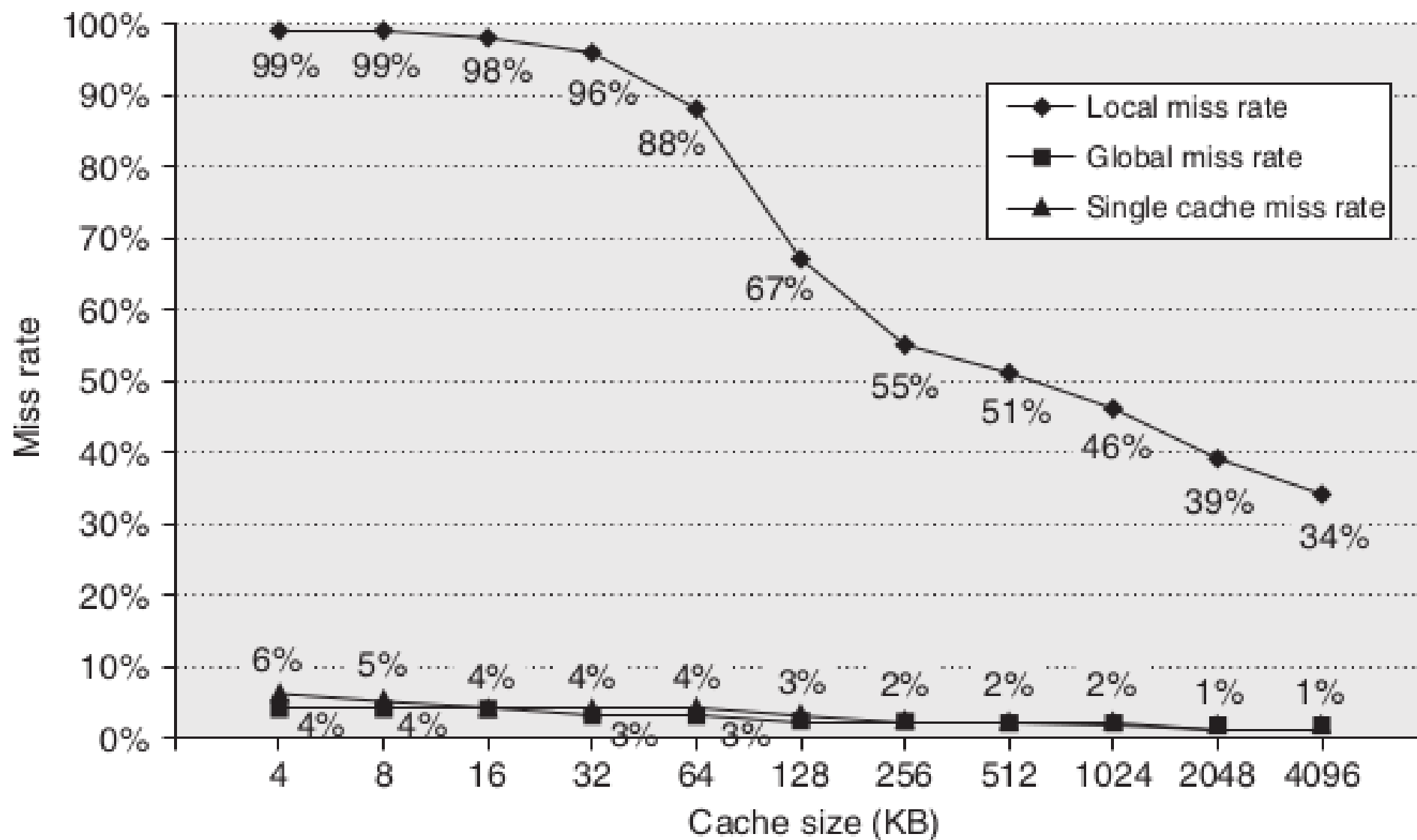
Miss rate components (relative percent)
(sum = 100% of total miss rate)


Cache size (KB)	Degree associative	Total miss rate						
			Compulsory		Capacity		Conflict	
4	1-way	0.098	0.0001	0.1%	0.070	72%	0.027	28%
4	2-way	0.076	0.0001	0.1%	0.070	93%	0.005	7%
4	4-way	0.071	0.0001	0.1%	0.070	99%	0.001	1%
4	8-way	0.071	0.0001	0.1%	0.070	100%	0.000	0%
8	1-way	0.068	0.0001	0.1%	0.044	65%	0.024	35%
8	2-way	0.049	0.0001	0.1%	0.044	90%	0.005	10%
8	4-way	0.044	0.0001	0.1%	0.044	99%	0.000	1%
8	8-way	0.044	0.0001	0.1%	0.044	100%	0.000	0%
16	1-way	0.049	0.0001	0.1%	0.040	82%	0.009	17%
16	2-way	0.041	0.0001	0.2%	0.040	98%	0.001	2%
16	4-way	0.041	0.0001	0.2%	0.040	99%	0.000	0%
16	8-way	0.041	0.0001	0.2%	0.040	100%	0.000	0%
32	1-way	0.042	0.0001	0.2%	0.037	89%	0.005	11%
32	2-way	0.038	0.0001	0.2%	0.037	99%	0.000	0%
32	4-way	0.037	0.0001	0.2%	0.037	100%	0.000	0%
32	8-way	0.037	0.0001	0.2%	0.037	100%	0.000	0%
64	1-way	0.037	0.0001	0.2%	0.028	77%	0.008	23%
64	2-way	0.031	0.0001	0.2%	0.028	91%	0.003	9%
64	4-way	0.030	0.0001	0.2%	0.028	95%	0.001	4%
64	8-way	0.029	0.0001	0.2%	0.028	97%	0.001	2%
128	1-way	0.021	0.0001	0.3%	0.019	91%	0.002	8%
128	2-way	0.019	0.0001	0.3%	0.019	100%	0.000	0%
128	4-way	0.019	0.0001	0.3%	0.019	100%	0.000	0%
128	8-way	0.019	0.0001	0.3%	0.019	100%	0.000	0%

4) Multilevel caches to reduce miss penalty


- A difficult decision is whether to make the cache hit time fast, to keep pace with the high clock rate of processors, or to make the cache large to reduce the gap between the processor accesses and main memory accesses

- 
- If L1 and L2 refer, respectively, to first- and second-level caches, we can redefine the average memory access time:
 - $\text{Hit time L1} + \text{Miss rate L1} \times (\text{Hit time L2} + \text{Miss rate L2} \times \text{Miss penalty L2})$






5) Giving priority to read misses over writes to reduce miss penalty

- A write buffer is a good place to implement this optimization.
 - Write buffers create hazards because they hold the updated value of a location needed on a read miss
 - Most processors give reads priority over writes.
 - This choice has little effect on power consumption.
- 




6) Avoiding address translation during indexing of the cache to reduce hit time


- Translation of a virtual address from the processor to a physical address to access memory.
- A common optimization is to use the page offset—*the part that is identical in both virtual and physical addresses*—to index the cache.


- 
- The advantages of removing the translation look aside buffer (TLB) access from the critical path outweigh the disadvantages.

AMAT

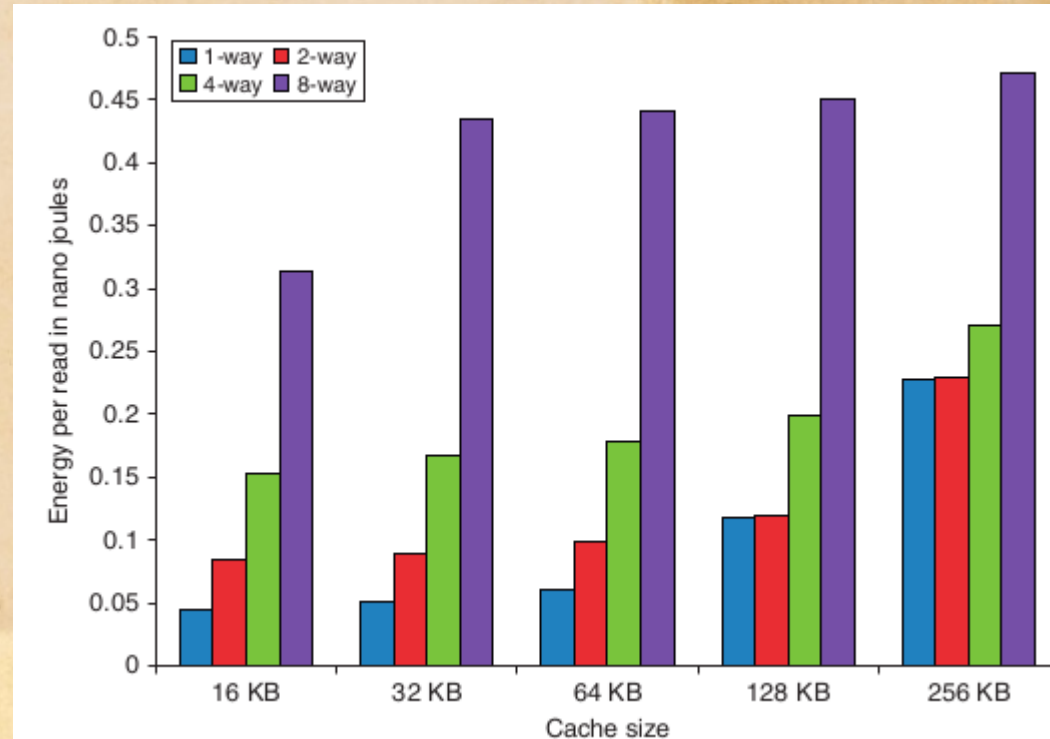
- $\text{effective-access-time} = \text{cache-access-time} + \text{miss-rate} * \text{miss-penalty}$
- We may also say that: $\text{effective-access-time} = \text{average memory access time (AMAT)}$
- when: $\text{cache-access-time} = \text{hit time}$
- AMAT: The average time to access memory considering both hits and misses and the frequency of different accesses


- 
- $AMAT = \text{hit time} + \text{Miss-rate} \times \text{Miss-penalty} \rightarrow \text{eqn 1}$
 - Designers sometime use AMAT as a way to examine alternative cache designs.

- 
- Also revisiting the CPU time equation:
 - CPU execution time = (CPU clock cycles + Memory stall cycles) \times Clock cycle time
 - Memory stall cycles = Number of misses \times Miss penalty
 - $\quad \quad \quad = IC \times (\text{Misses/Instruction}) \times \text{Miss penalty}$
 - $= IC \times (\text{Memory access/Instruction}) \times \text{Miss rate} \times \text{Miss penalty} \rightarrow \text{eqn 2}$

- 
- The good thing about *eqn 2* is that the components can be easily measured
 - The two equations give us three metrics for cache optimizations
 - **hit time, miss-rate, and miss-penalty.**
 - And two more: **Cache bandwidth and Power consumption.**

- You may come across different cache optimization techniques and most of them will be aiming at
- Reducing hit time – using smaller L1 caches – Lowers power consumption



- 
- Reducing the Miss-penalty – compiler-controlled prefetching – Merging write buffer etc
 - Reducing the miss rate- Compiler optimizations - any improvement at compile time improves power consumption.
 - Increasing cache bandwidth – Pipelined cache access – Multi-banked caches.

Summary



- Cache controller's algorithm
 - The 3Cs model
 - Six basic cache optimizations based on the 3Cs
 - AMAT
 - Cache optimization using the AMAT equation and how they affect power and bandwidth
 - The sweetness of compiler optimizations
- 