POLITECNICO DI MILANO

High Performance Processors and Systems

Review of caches

Donatella Sciuto: sciuto@elet.polimi.it

HPPS

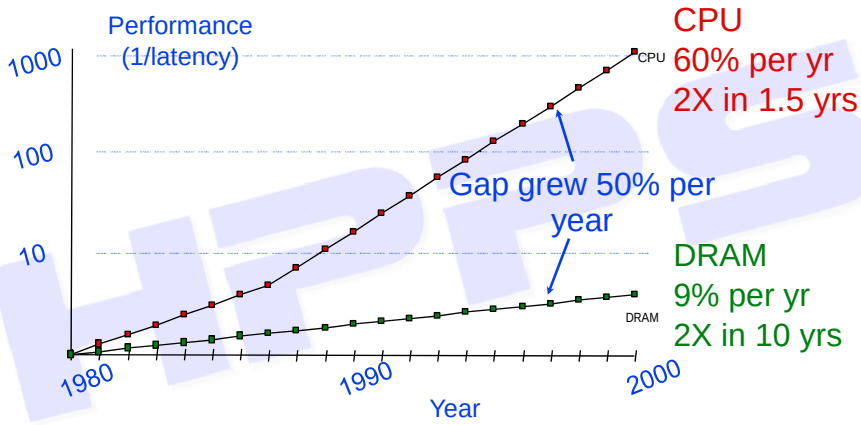
Outline

- Introduction to memory hierarchy
- Introduction to caches
- 4 Questions on caches
- Performance evaluation

HPPS

Since 1980, CPU has outpaced DRAM …

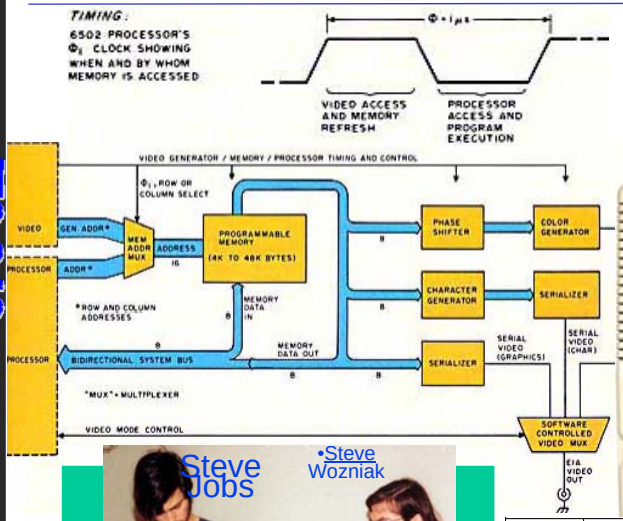
Four-issue 2GHz superscalar accessing 100ns DRAM could execute 800 instructions during time for one memory access!



How do architects address this gap?

Put small, fast "cache" memories between CPU and DRAM.
Create a "memory hierarchy"

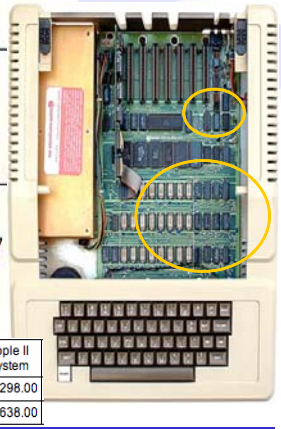
1977: DRAM faster than microprocessors



Apple][(1977)

CPU: 1000 ns

DRAM: 400 ns



RAM Complement	Apple II System
4K	$ 1,298.00
48K	2,638.00

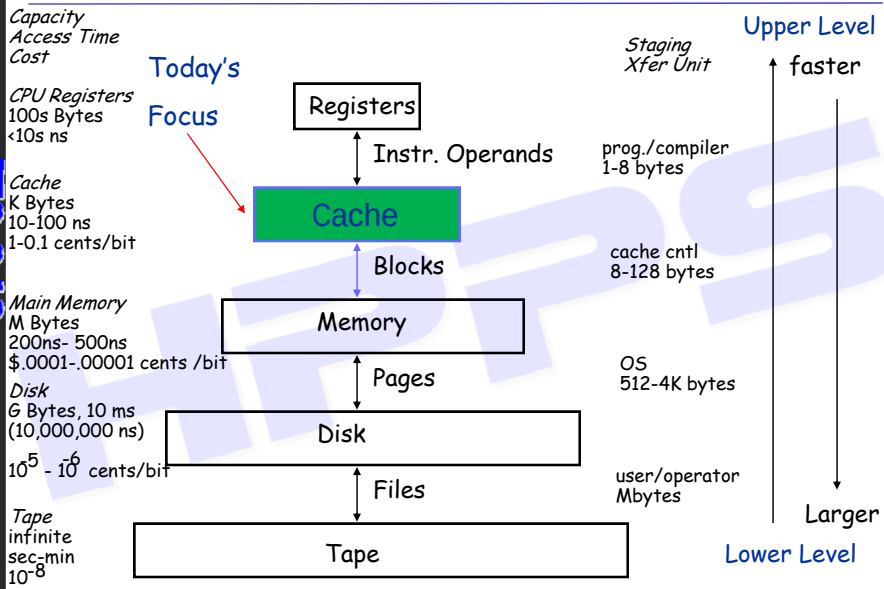
Processor – Memory gap

- From 1980 to 1986:
  - DRAM latency decreases 9% per year
  - CPU performance increases 1.35x per year
- After 1986:
  - Performance increase for CPUs to 1.55 x per year
  - DRAM performance constant

Addressing the processor-memory performance gap

- Goal
  - Illusion of large, fast, cheap memory.
  - Let programs address a memory space that scales to the disk size, at a speed that is usually as fast as register access
- Solution
  - Memory hierarchy with different technologies, costs and sizes and different access mechanisms
  - Put smaller, faster “cache” memories between CPU and DRAM. Create a “memory hierarchy”.

Levels of the Memory Hierarchy



The principle of locality

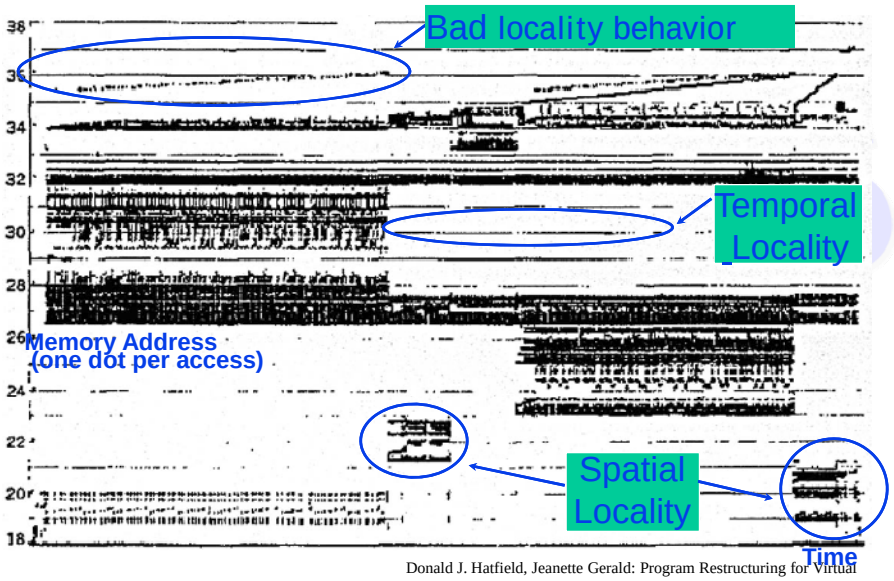
- The Principle of Locality:
  - Program access a relatively small portion of the address space at any instant of time.

Two predictable properties of memory references:

- Temporal Locality: If a location is referenced, it is likely to be referenced again in the near future (e.g., loops, reuse).
- Spatial Locality: If a location is referenced it is likely that locations near it will be referenced in the near future (e.g., straightline code, array access).

- Last 15 years, HW relied on locality for speed

Programs with locality cache well …



Donald J. Hatfield, Jeanette Gerald: Program Restructuring for Virtual Memory. IBM Systems Journal 10(3): 168-192 (1971)

9

Memory Hierarchy: Apple iMac G5

Managed by compiler

Managed by hardware

Managed by OS, hardware, application

07	Reg	L1 Inst	L1 Data	L2	DRAM	Disk
Size	1K	64K	32K	512K	256M	80G
Latency Cycles, Time	1, 0.6 ns	3, 1.9 ns	3, 1.9 ns	11, 6.9 ns	88, 55 ns	10^7, 12 ms

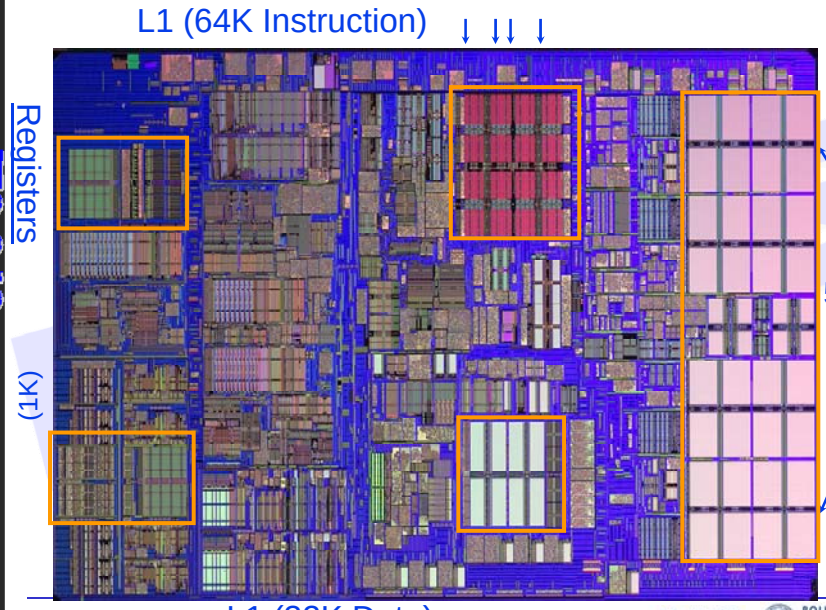
Mac G5 1.6 GHz

Goal: Illusion of large, fast, cheap memory

Let programs address a memory space that scales to the disk size, at a speed that is usually as fast as register access

10

iMac's PowerPC 970: All caches on-chip



11

Caches

- Caches exploit both types of predictability:
  - Exploit temporal locality by remembering the contents of recently accessed locations.
  - Exploit spatial locality by fetching blocks of data around recently accessed locations.

12

Cache algorithm: read

Look at Processor Address, search cache tags to find match. Then either

HIT - Found in Cache

Return copy of data from cache

MISS - Not in cache

Read block of data from Main Memory

Wait …

Return data to processor and update cache

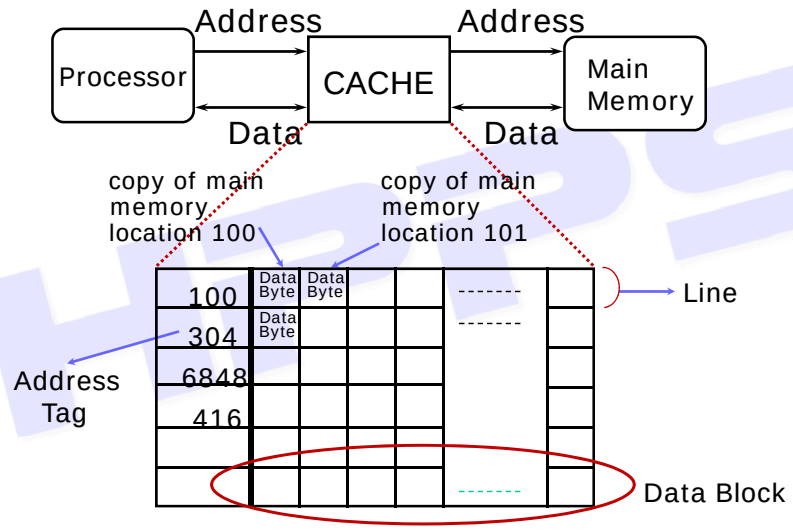
Hit Rate = fraction of accesses found in cache

Miss Rate = 1 - Hit rate

Hit Time = RAM access time + time to determine HIT/MISS

Miss Time = time to replace block in cache + time to deliver block to processor

Inside a cache

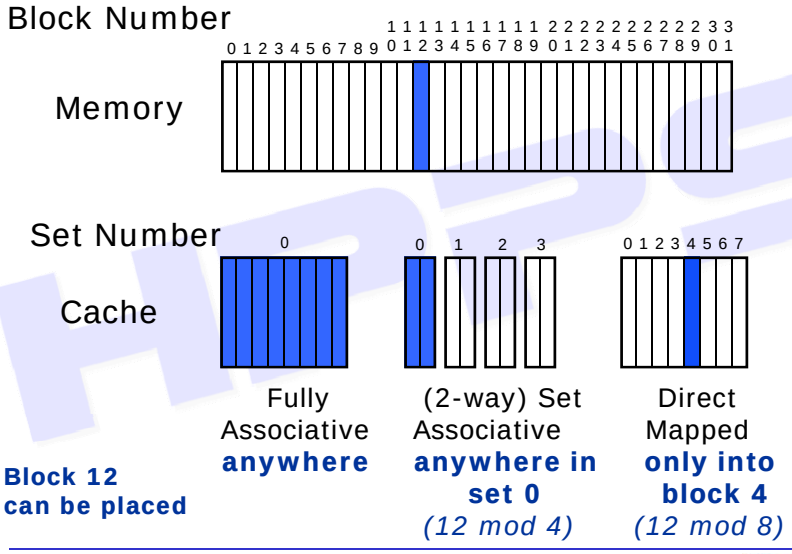


4 Questions on memory hierarchy

- Q1: Where can a block be placed in the cache?
  (Block placement)
- Q2: How is a block found if it is in the cache?
  (Block identification)
- Q3: Which block should be replaced on a miss?
  (Block replacement)
- Q4: What happens on a write?
  (Write strategy)

15

Q1: Where can a block be placed?



Block 12 can be placed

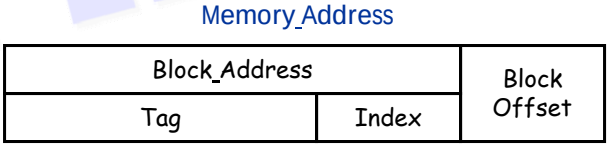
16

Sources of cache misses

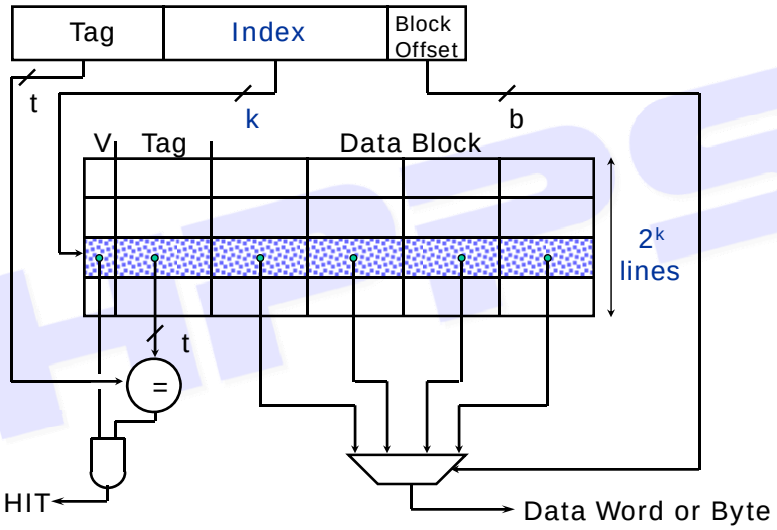
- Compulsory (cold start or process migration, first reference): first access to a block
  - "Cold" fact of life: not a whole lot you can do about it
  - Note: If you are going to run "billions" of instruction, Compulsory Misses are insignificant
- Capacity:
  - Cache cannot contain all blocks access by the program
  - Solution: increase cache size
- Conflict (collision):
  - Multiple memory locations mapped to the same cache location
  - Solution 1: increase cache size
  - Solution 2: increase associativity
- Coherence (Invalidation): other process (e.g., I/O) updates memory

Q2: How is a block found?

- Index selects which set to look in
- Tag used to identify actual copy
  - If no candidates match, then declare cache miss
- Block is minimum quantum of caching
  - Data select field used to select data within block
  - Many caching applications don't have data select field
- Tag on each block
  - No need to check index or block offset
- Increasing associativity shrinks index, expands tag. Fully Associative caches have no index field.

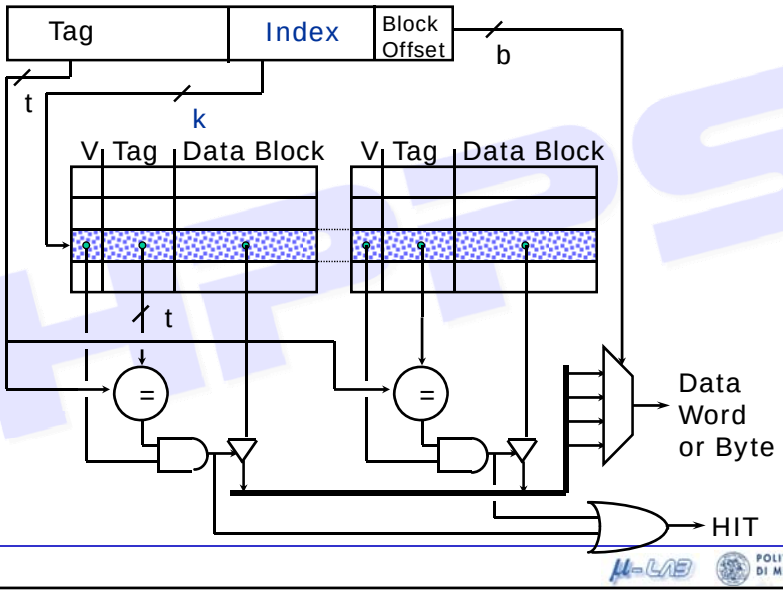


Direct-Mapped Cache



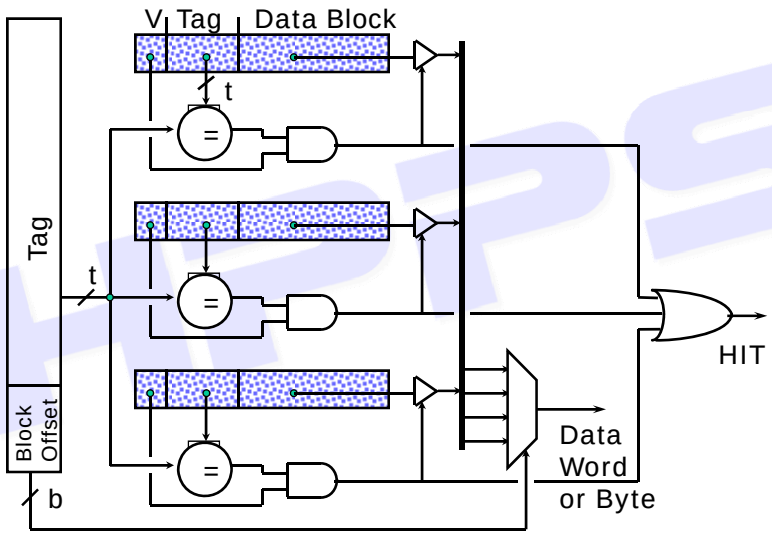
19

2-Way Set-Associative Cache



20

Fully Associative Cache



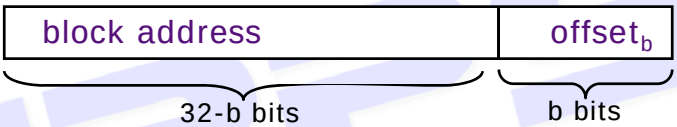
21

Block Size and Spatial Locality

Block is unit of transfer between the cache and memory

4 word block, b=2

Split CPU address



32-b bits

b bits

2^b = block size *a.k.a* line size (in bytes)

Larger block size has distinct hardware advantages

- less tag overhead
- exploit fast burst transfers from DRAM
- exploit fast burst transfers over wide busses

What are the disadvantages of increasing block size?

Fewer blocks => more conflicts. Can waste bandwidth.

22

Q3: Which block should be replaced on a miss?

- Easy for Direct Mapped
- Set Associative or Fully Associative:
  - Random
  - Least Recently Used (LRU)
    - LRU cache state must be updated on every access
    - true implementation only feasible for small sets (2-way)
    - pseudo-LRU binary tree often used for 4-8 way
  - First In, First Out (FIFO) a.k.a. Round-Robin
    - used in highly associative caches
- Replacement policy has a second order effect since replacement only happens on misses

How well random choice works

Assoc:	2-way		4-way		8-way	
Size	LRU	Ran	LRU	Ran	LRU	Ran
16K	5.2%	5.7%	4.7%	5.3%	4.4%	5.0%
64K	1.9%	2.0%	1.5%	1.7%	1.4%	1.5%
256K	1.15%	1.17%	1.13%	1.13%	1.12%	1.12%

Q4: What happens on a write?

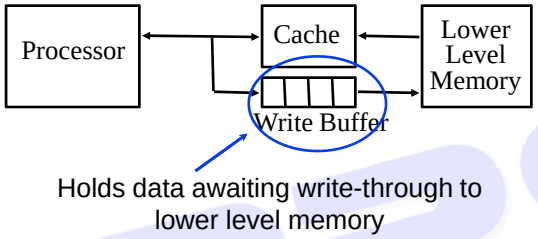
- Cache hit:
  - *write through:* write both cache & memory
    - generally higher traffic but simplifies cache coherence
  - *write back:* write cache only (memory is written only when the entry is evicted)
    - a dirty bit per block can further reduce the traffic
- Cache miss:
  - *no write allocate:* only write to main memory
  - *write allocate (aka fetch on write):* fetch into cache
- Common combinations:
  - write through and no write allocate
  - write back with write allocate

Q4: What happens on a write?

	Write-Through	Write-Back
Policy	Data written to cache block also written to lower-level memory	Write data only to the cache Update lower level when a block falls out of the cache
Debug	Easy	Hard
Do read misses produce writes?	No	Yes
Do repeated writes make it to lower level?	Yes	No

Additional option -- let writes to an un-cached address allocate a new cache line ("write-allocate").

Write Buffers for Write-Through Caches



Q. Why a write buffer ?

A. So CPU doesn't stall

Q. Why a buffer, why not just one register ?

A. Bursts of writes are common.

Q. Are Read After Write (RAW) hazards an issue for write buffer?

A. Yes! Drain buffer before next read, or check write buffers for match on reads

Remember

- The cache is faster than memories at lower levels of the hierarchy ⇨ hit time much less than the time requested to access memories at lower levels (main factor of miss penalty).
- Miss penalty derves mainly from technology
- High Hit rate ⇨ average access time near to hit time

How do we compute the **AVERAGE ACCESS TIME**?

How memories affect system performance

- **Memory stall cycles**: number of cycles in which the CPU is not working (*stalled*) waiting for a memory access;
- *Simplified assumptions*:
  - The cycle time includes the time necessary to manage a cache hit;
  - during a cache miss the CPU is stalled
- *CPU_execution_time = (CPU_clock_cycles + Memory_stall_cycles) * clock cycle time*
- *Memory_stall_cycles = Number_of_misses*miss_penalty*

29

A more detailed analysis

Memory_stall_cycles =
IC(Misses/Instruction)*Miss_penalty =*
*IC*Reads_per_instruction*Read_miss_rate*Read_miss_penalty*
+
*IC*Writes_per_instruction*Write_miss_rate*Write_miss_penalty*

We can simplify by averaging reads and writes
Memory_stall_cycles = IC(memory_accesses/instruction)* miss_rate*miss_penalty*

A different figure of merit
Misses/instruction = miss_rate (memory_accesses/instruction)*

Independent of the hardware implementation, dependent on the architecture (related to the average number of memory accesses per instructions)

30

Average access time

- $T_A = hit\_rate*hit\_time+miss\_rate*miss\_time$
  $= hit\_rate*hit\_time + miss\_rate*(hit\_time + miss\_penalty)=$
  $hit\_time*(hit\_rate+miss\_rate) + miss\_rate*miss\_penalty$
  $= hit\_time + miss\_rate*miss\_penalty$.

- Architectural choices may reduce the miss rate:
  ⇨ T_A nearer to *hit_time*.

5 Basic Cache Optimizations

- Reducing Miss Rate

1. Larger Block size (compulsory misses)
2. Larger Cache size (capacity misses)
3. Higher Associativity (conflict misses)

- Reducing Miss Penalty

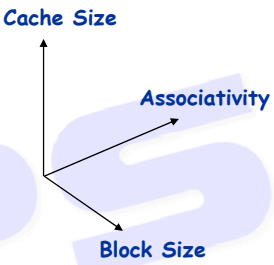
4. Multilevel Caches

- Reducing hit time

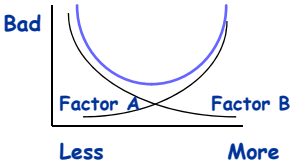
5. Giving Reads Priority over Writes
   - E.g., Read complete before earlier writes in write buffer

The cache design space

- Several interacting dimensions
  - cache size
  - block size
  - associativity
  - replacement policy
  - write-through vs write-back



- The optimal choice is a compromise
  - depends on access characteristics
    - workload
    - use (I-cache, D-cache, TLB)
  - depends on technology / cost
- Simplicity often wins



Questions