EEC 485 High Performance Arch. (Fall 2006)

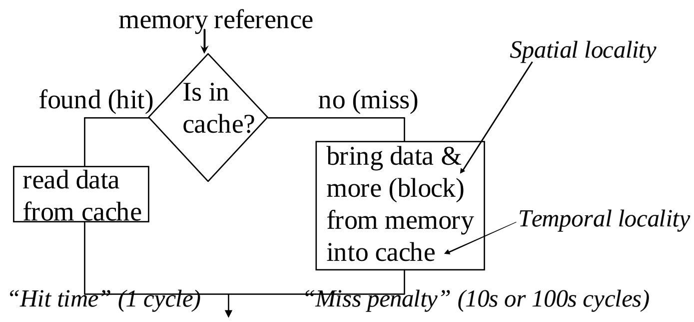
7.2 The Basics of Cache

Chansu Yu

Cleveland State University

Caches: The Basic Idea

- A smaller set of storage locations storing a subset of information from a larger set (memory)
  - Unlike registers or memory, invisible to programmer/ISA



Cleveland State University

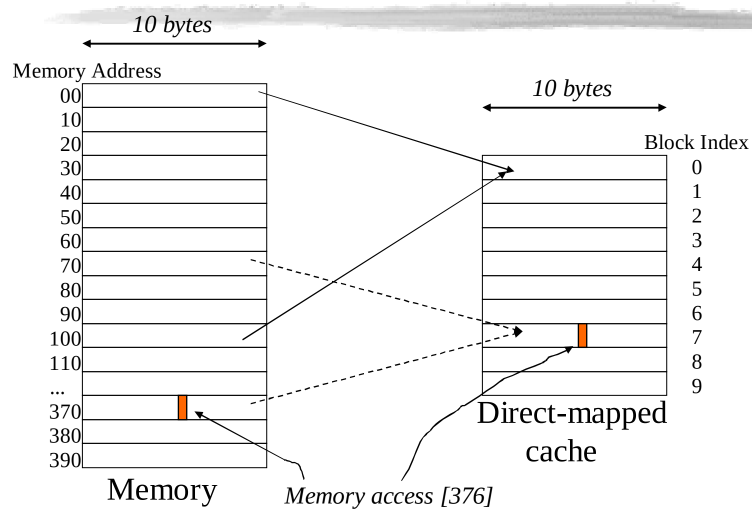
c.yu91@csuohio.edu

Cache Block Placement

- Two issues:
  - How do we know if a data item is in the cache?
  - If it is, how do we find it?
- Our first example: (with decimal numbers, which is not the actual case)
  - Block size is 10 bytes of data
  - Memory has 400 bytes (40 blocks)
  - Cache has 100 bytes (10 blocks)
  - “Direct mapped"

Cleveland State University 3 c.yu91@csuohio.edu

Direct-mapped Block Placement

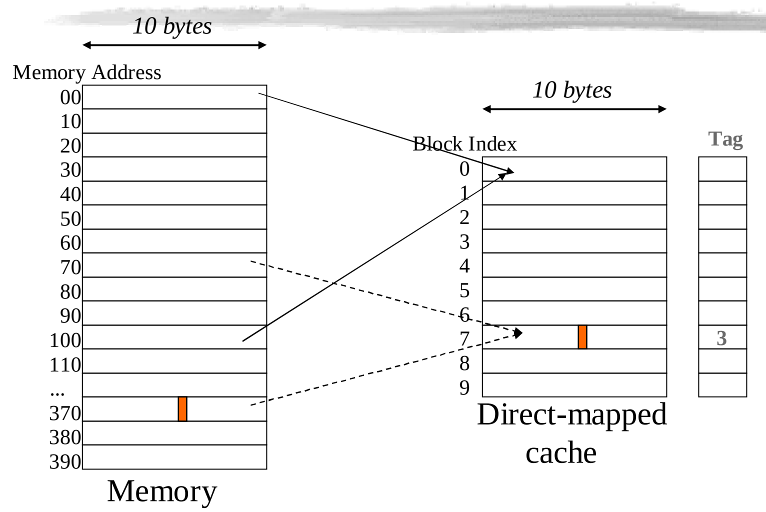


Cleveland State University 4 c.yu91@csuohio.edu

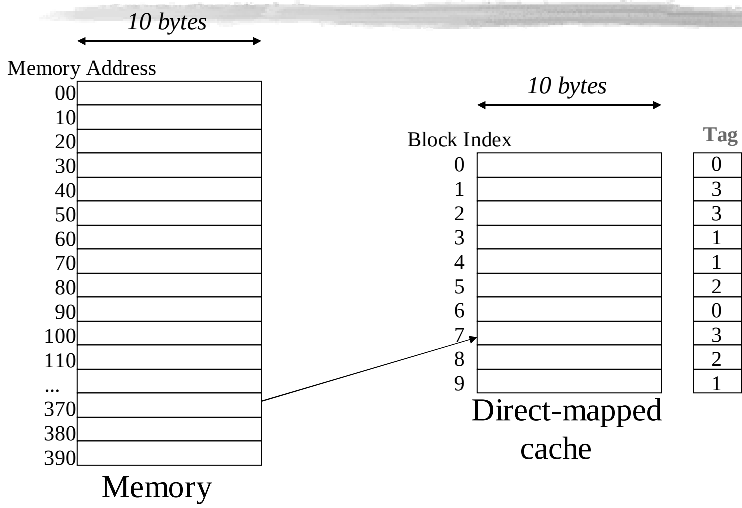
Decoding Memory Address

- Memory address [376]
  - offset within the block
  - cache block index
  - for identifying the original memory block
- Given the memory address (e.g. 376)
  - Extract the cache block index (7)
  - Check if the cache block #7 corresponds to memory 370~379
    - For this, each cache entry remembers the “tag” data (e.g. “3”)
    - If “tag” matches with the first digit in memory address, HIT
  - Extract the byte within the block with offset address (e.g. 6)

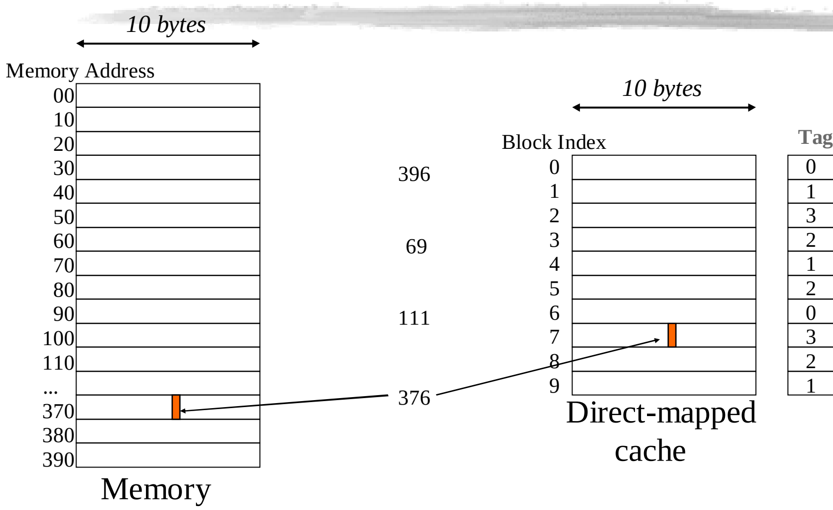
Direct-mapped Block Placement



Exercise1: Make connections!



Exercise2: Find them!



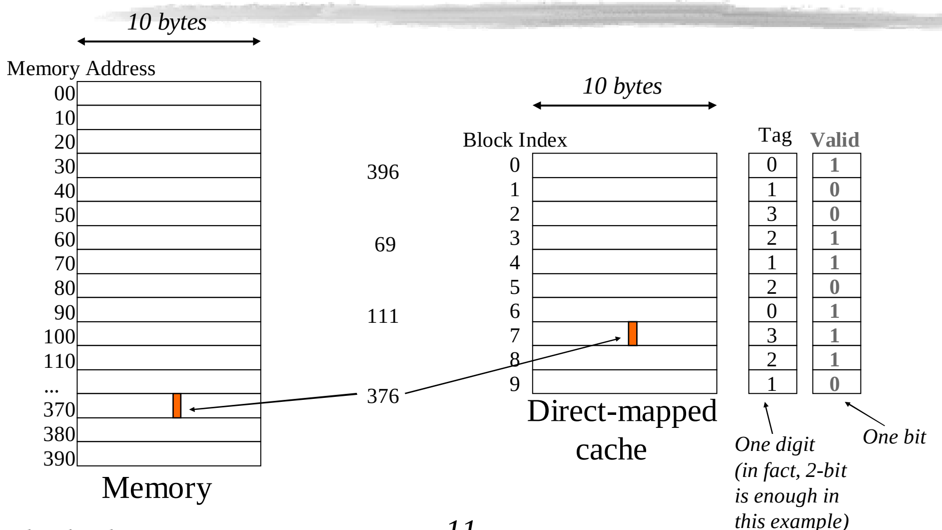
Decoding Memory Address

- Given the memory address (e.g. 396)
  - Extract the cache block index (9)
  - Check if the cache block #9 corresponds to memory 390~399
    - Since tag is 1 and the first digit in memory address is 1, “MISS”
- So, what happens when a miss occurs? (see pages 482-483)
  - The current cache block #9 contains 190~199
  - Read memory 390~399 and replace the cache block #9
  - Change the Tag to “3”

Decoding Memory Address

- Given the memory address (e.g. 069)
  - Extract the cache block index (6)
  - Check if the cache block #6 corresponds to memory 060~069
    - Tag is 0 and therefore, “HIT”
  - However, it is possible that Tag=0 because the cache is initially empty
    - This is not a HIT ??? => We need a “valid bit” !

Exercise3: What is invalid?



Decoding Memory Address

- Given the memory address (e.g. 069)
  - Extract the cache block index (6)
  - Check if the cache block #6 corresponds to memory 060~069
    - Since “Tag”=0 AND “valid bit”=1, “HIT”
- Given the memory address (e.g. 111)
  - Extract the cache block index (1)
  - Check if the cache block #1 corresponds to memory 110~119
    - Since “Tag”=1 but “valid bit”=0, “MISS”
- So, what happens when a miss occurs?
  - The current cache block #1 contains an invalid data
  - Read memory 110~119 and replace the cache block #1
  - Change the Tag to “1” and set the valid bit to “1”

Decoding Memory Address

This example:
Block size is 10 bytes of data
Memory has 400 bytes (40 blo
Cache has 100 bytes (10 block
"Direct mapped"

- Memory address [376]

offset within the block
cache block index
for identifying the original memory block

Another example:
Block size is 10 bytes of data
Memory has <u>4000</u> bytes (<u>400</u> blo
Cache has <u>1000</u> bytes (<u>100</u> block
"Direct mapped"

- Memory address [3276]

offset within the block
cache block index
for identifying the original memory block

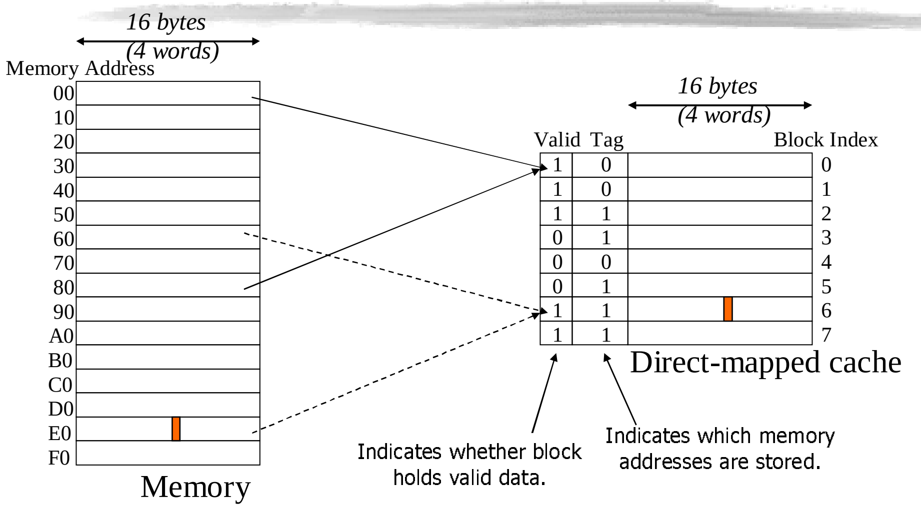
Decoding Memory Address

- In reality, the principle is the same but it looks not simple
  - It is not a decimal world
  - 1-digit for tag, 1-digit for block index, and 1-digit for offset: Not the case!
- Memory address
  - 256 bytes memory (address 0-255 or FF) : 8-bit address
  - 16 blocks in memory
  - 8 blocks in cache (block index 0-7 or 000 - 111)
- Given 8-bit memory address [01101011]
  - What is the offset within the block, cache block index, and for identifying the original memory block?

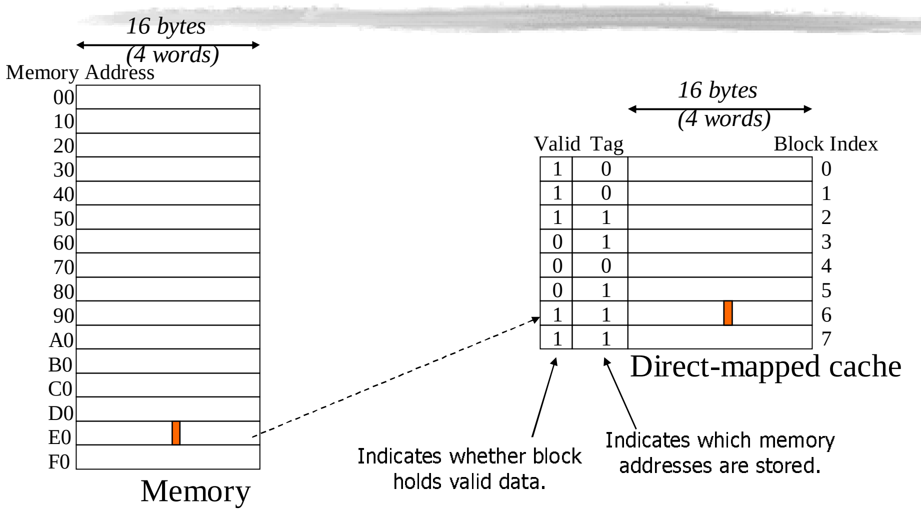
tag	block #	offset in the block
	3	4

8-bit memory address

Direct-mapped Block Placement



Exercise4: Make connections!



Exercise5: Cache behavior

16 bytes (4 words)

Memory Address

Memory Address	
00	
10	
20	
30	
40	
50	
60	
70	
80	
90	
A0	
B0	
C0	
D0	
E0	
F0	

Memory

Example memory access sequence:

[24]

[28]

[40]

[AC]

16 bytes (4 words)

Valid	Tag		Block Index
1	0		0
1	0		1
1	1		2
0	1		3
0	0		4
0	1		5
1	1		6
1	1		7

Direct-mapped cache

What if [28] is write?

16 bytes (4 words)

Memory Address

Memory Address	
00	
10	
20	
30	
40	
50	
60	
70	
80	
90	
A0	
B0	
C0	
D0	
E0	
F0	

Memory

Example memory access sequence:

[24]

[28]

[40]

[AC]

16 bytes (4 words)

Valid	Tag		Block Index
1	0		0
1	0		1
1	1		2
0	1		3
0	0		4
0	1		5
1	1		6
1	1		7

Direct-mapped cache

Handling Writes

Write strategy (see pages 483-485)

- When write hit
  - Write through: data is written to both the cache and the memory
  - Write back : data is written only to the cache
    - the modified (dirty) cache block is written to memory when replaced
    - requires dirty bit for each cache block (more complexity)
    - Better performance but semantic problem
- When write miss
  - Write allocate : fetch-on-write - with Write-back ???
  - No write allocate : write-around - with Write-through ???

Cache Block Placement

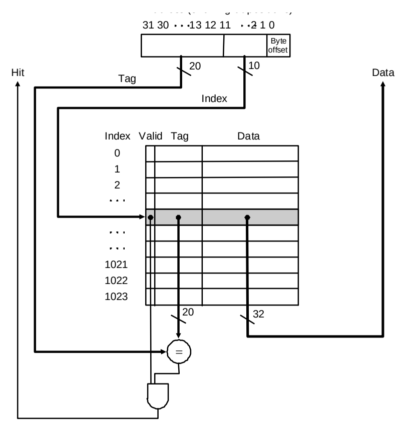
- Two issues:
  - How do we know if a data item is in the cache?
  - If it is, how do we find it?
- Our first example:
  - block size is 4 words of data
  - "direct mapped"

For each item of data in memory,
there is exactly one location in the cache where it might be.

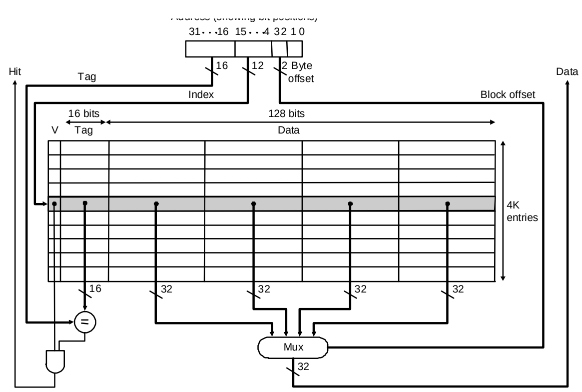
e.g., lots of items in memory share locations in the cache

Single Word Cache Block : Exercise6

- Memory address
  - Memory size ?
    (# bits for memory address ?)
  - Block size ?
    (# bits for offset ?)
  - # Memory blocks ?
    (# bits for block index ?)
- Memory address to Cache address
  - # Cache blocks in direct-mapped cache ?
    (# bits for block index ?)
  - Which (how many) memory blocks are candidates for cache block #5 ?
  - Tag size ?
  - How memory address is decomposed ?

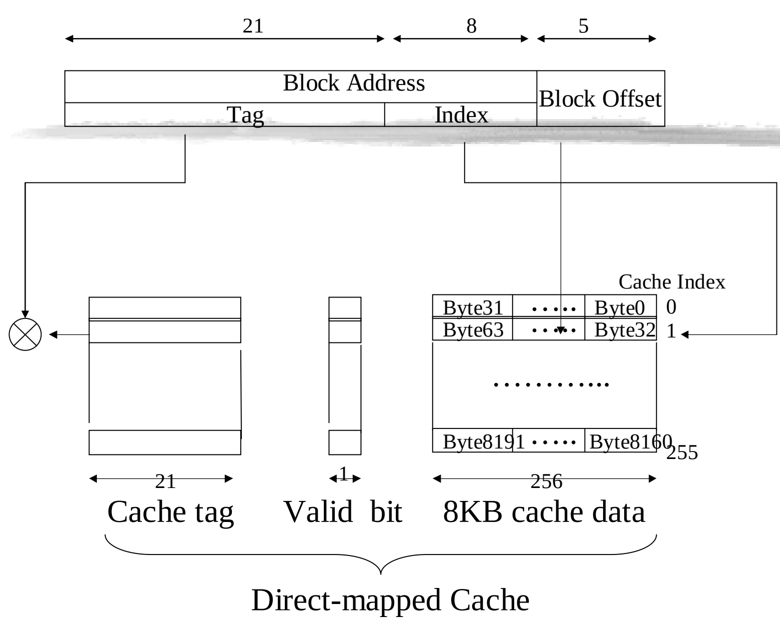


Exercise7 – Multiword Cache Block



Block Identification

- Ex) Alpha 21064 (Direct-mapped cache)
  - 32-byte Block size : Requires 5-bit for block offset
  - 8KB cache size : 256 (=8KB/32B) cache blocks
    Requires 8-bit for block index
  - 34-bit memory address supports up to 16GB :
    512M (=16GB/32B) memory blocks
  - Mapping : 2M (=512M/256) candidates for a cache block (compete with each other) => Requires 21-bit to identify



Cache Performance: Tradeoffs

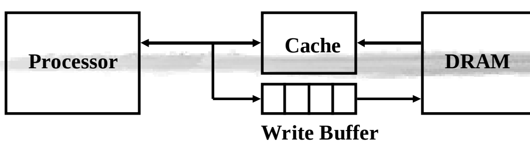
(1) Increasing block size
- + decreases miss rate, until block gets large (spatial locality)
- - increases miss penalty

(2) Increasing cache size
- + decreases miss rate
- - increases hit time
- - increases hardware cost

(3) Increasing associativity (Section 7.3)
- + increases hit rate
- - increases hit time
- - increases hardware cost

To make exact tradeoffs, need to know specific numbers.
- Calculation & measurement
- See book for formulae.

Cache Performance: Tradeoffs

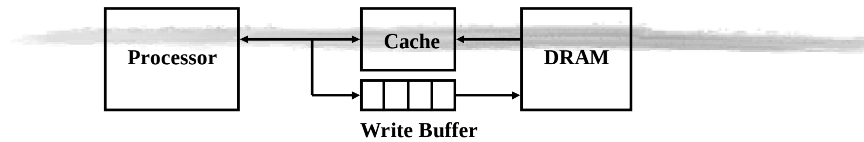


(4) A Write Buffer is needed between the Cache and Memory
- ➢Processor: writes data into the cache and the write buffer
- ➢Memory controller: write contents of the buffer to memory

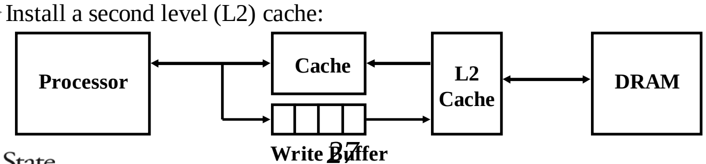
❑Write buffer is just a FIFO:
- ➢Typical number of entries: 4
- ➢Works fine if: Store frequency (w.r.t. time) << 1 / DRAM write cycle

❑Memory system designer's nightmare:
- ➢Store frequency (w.r.t. time) >> 1 / DRAM write cycle
- ➢Write buffer saturation

Write Buffer Saturation

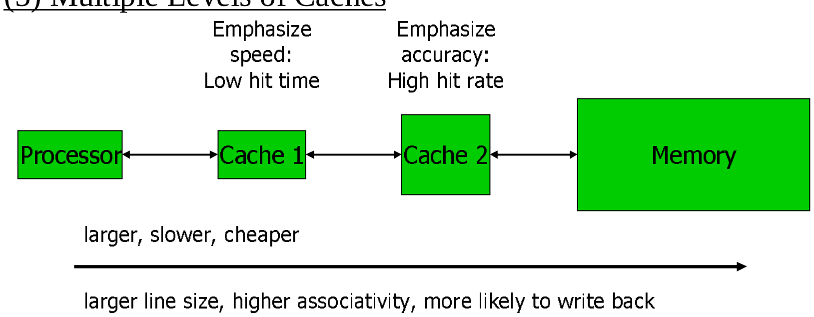


- ❑Store frequency (w.r.t. time) >> 1 / DRAM write cycle
  - ➢If this condition exist for a long period of time (CPU cycle time too quick and/or too many store instructions in a row):
    - Write buffer will overflow no matter how big you make it
    - The CPU Cycle Time <= DRAM Write Cycle Time
- ❑Solution for write buffer saturation:
  - ➢Use a write back cache
  - ➢Install a second level (L2) cache:



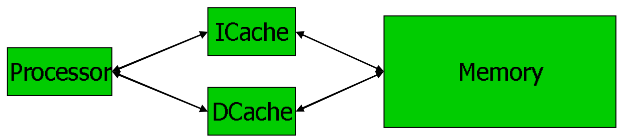
Cache Performance: Tradeoffs

(5) Multiple Levels of Caches



Understanding performance is tricky. Use lots of simulation & testing.

Cache Performance: Tradeoffs



(6) Independent Instruction & Data Caches

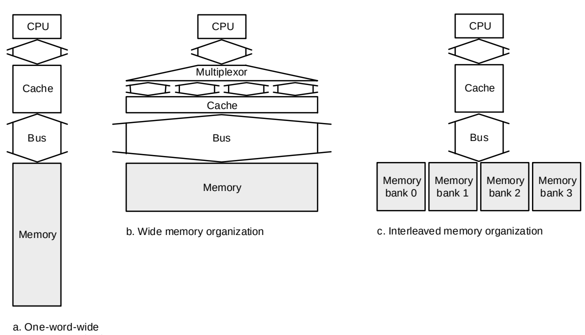
- Possible if locations accessed are independent (different segments)
  - Avoids conflict misses between instructions & data
  - But increases conflict misses among instructions & among data

- Can be of different sizes & strategies
  - Instructions typically have greater locality
  - ICaches often smaller than DCaches
  - ICaches often don't support writes
  - DCaches often support multiple simultaneous accesses (multi-porting)

Cache Performance: Tradeoffs

(7) Memory organization

- Make reading multiple words easier by using banks of memory: It can get a lot more complicated...



Cache Performance: Tradeoffs

<u>(8) Performance equation</u>

- Simplified model:

  execution time = (execution cycles + stall cycles) x cycle time

  stall cycles = # of instructions x miss ratio x miss penalty

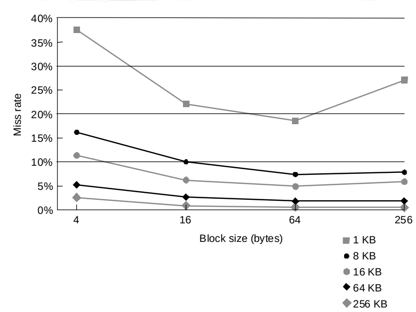
- Two ways of improving performance:
  - decreasing the miss ratio
  - decreasing the miss penalty

What happens if we increase block size?

Performance

- Increasing the block size tends to decrease miss rate

- Use split caches because there is more spatial locality in code



Program	Block size in words	Instruction miss rate	Data miss rate	Effective combined miss rate
gcc	1	6.1%	2.1%	5.4%
	4	2.0%	1.7%	1.9%
spice	1	1.2%	1.3%	1.2%
	4	0.3%	0.6%	0.4%