Memory Systems

In the von Neumann model, the same mem holds both instr and data.

Part of computation is kept also in the processor, portions must move back and forth as needed.

⇨ Essential role: hold computation

- Close interaction with processor
- Limit computation size
- Limit performance

⇨ Design goal (ideally)

Maximum amount of fastest memory

Memory Solutions Reality

⇨ CPU-memory technology gap

Not all storage systems are usable as memory (i.e., to hold jobs marked or selected for execution)

⇨ Revised Design Goal

Maximum amount of fastest memory economically

Memory posed major challenges to builders due to a growing performance gap between two parts essential to performing computations which must interact closely.

Memory: Looking Deeper

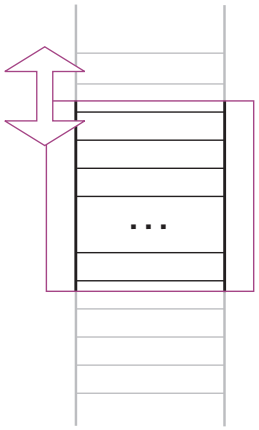
Principle of Locality

➪ [Memory] address

Mem generally stores program instr and data, but how exactly are they actually used?

➪ **Memory access patterns**

- ✎ Most recently used
- ✎ Close locations



In other words, how memory addresses are generated?

➪ **Mem *reference* locality**

- ✎ Temporal ~ access times
- ✎ Spatial ~ access locations

Memory Reference Locality Examples

Exercise
Give 3 examples of temporal and 3 examples of spatial locality for data. Repeat for instructions.

Think about your programs

- ✎ Data locality examples
- ✎ Instruction locality examples

Quiz
Which type of locality is exploited?

Note MIPS PC-relative addr

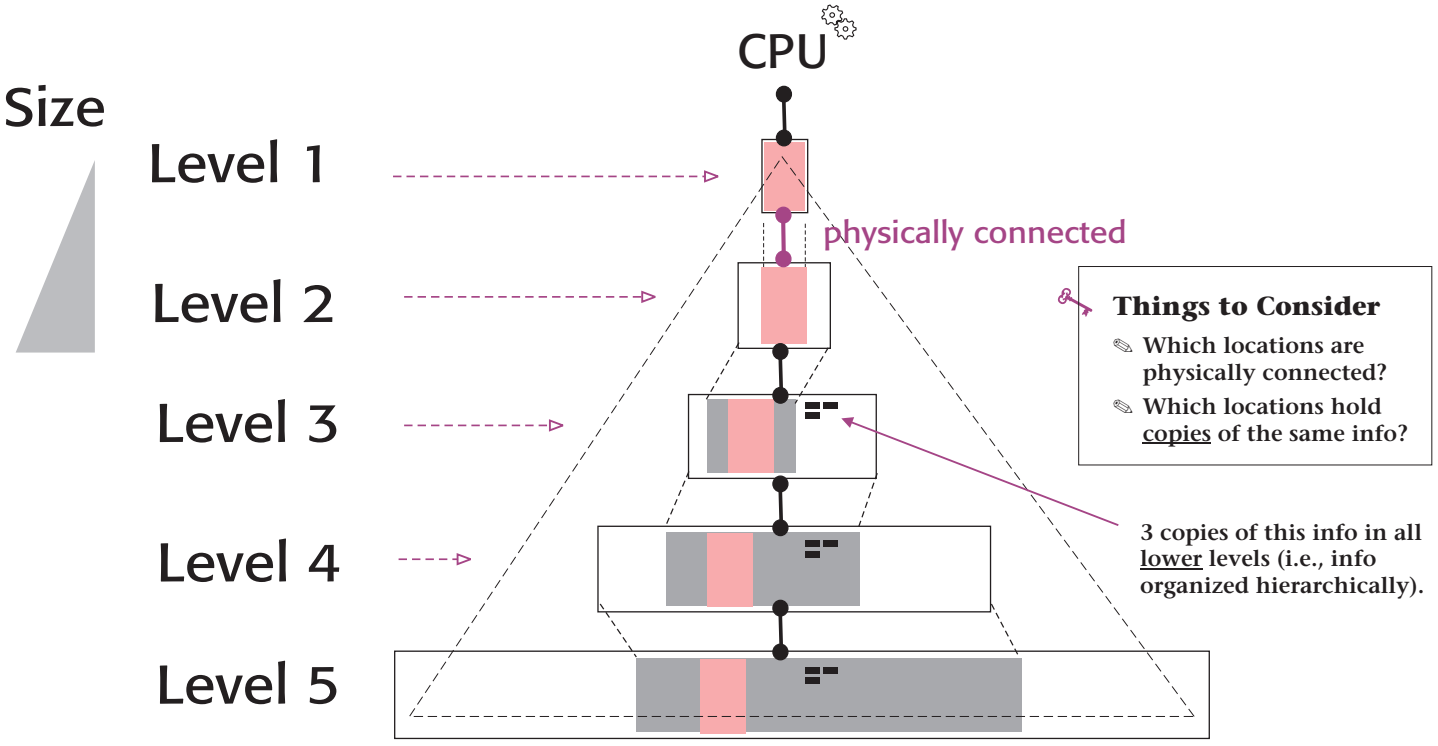
Store branch target offset

Keywords
Amount (**capacity**), economical (**cost**), and access speed (**performance**).

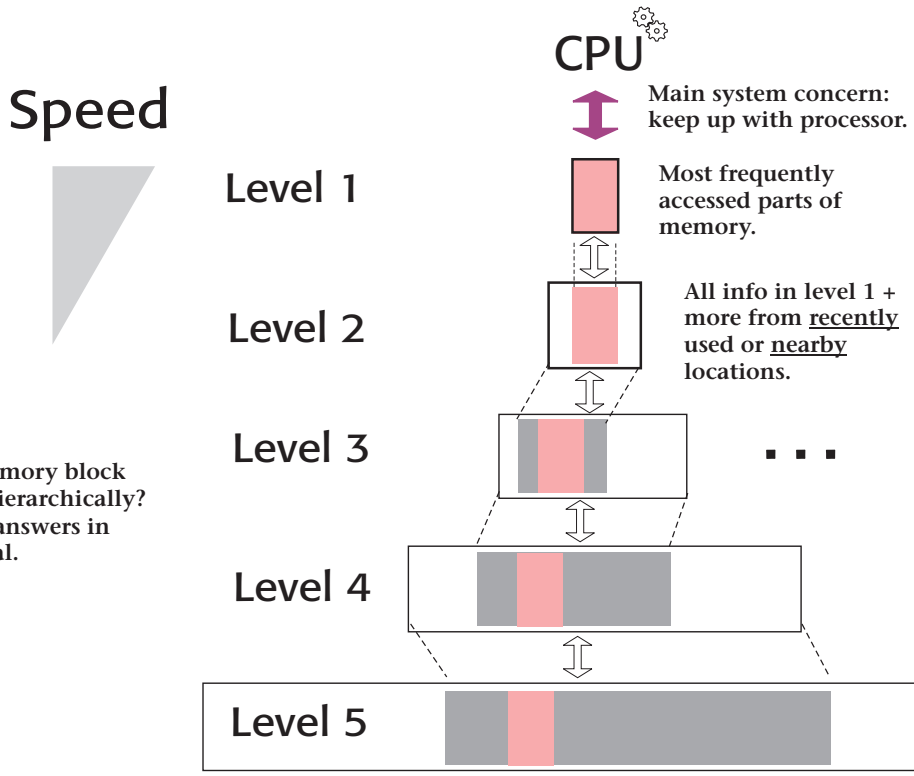
Storage hierarchy: a solution

Largest amount of economical storage accessed, most of the time, at the speed of the fastest storage

Memory Systems Design
A Memory Hierarchy



Memory Systems Design
Hierarchy Operation



Things to Consider

- Locations exchange info if physically connected
- Faster locations hold fewer but faster copies of the same info in lower levels

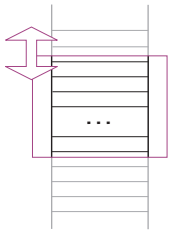
Quiz
Why would memory block sets be stored hierarchically?
Hint: look for answers in reading material.

Memory Systems Design
Summary

⇨ What is a memory hierarchy?

- ✎ A speed-size tradeoff
- ✎ Cost-driven
- ✎ Access locality-enabled

⇨ Why it works?

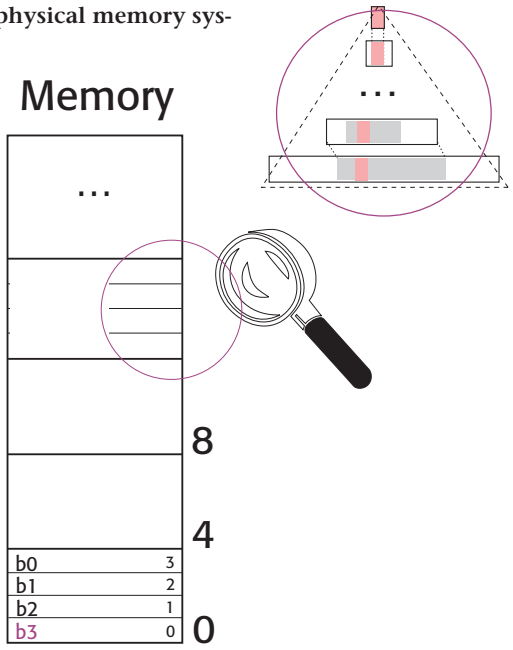


As much of the most actively used memory is kept in the fastest locations (upper part of the hierarchy) close to the CPU as long as possible.

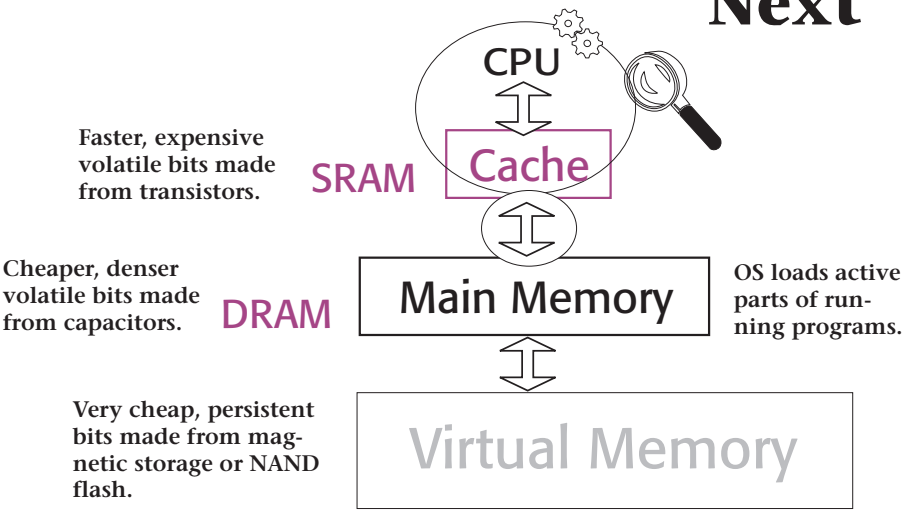
Memory Systems

Classic Hierarchy

Some logical memory addresses will have <u>many</u> copies of their contents in the physical memory system.



Next



A Basic Cache

➪ **Memory block**

Processors deal with byte addresses generated by programs which may not (actually don't) reference real memory locations at all (?).

Processor can use any memory address but is only physically connected to locations in cache

Memory bits are moved in **blocks** between cache and processor (architecture here refers to **processor word**, also registers where default/core 2s complement operands are stored).

- ✎ Architecture independent unit
- ✎ Example: 32 mem blocks (L2), 8 cache (L1) blocks

Typical RISC assumed, where addresses and operands are held in the same GPR.

- ✎ 1 block = 32-bit (processor) word

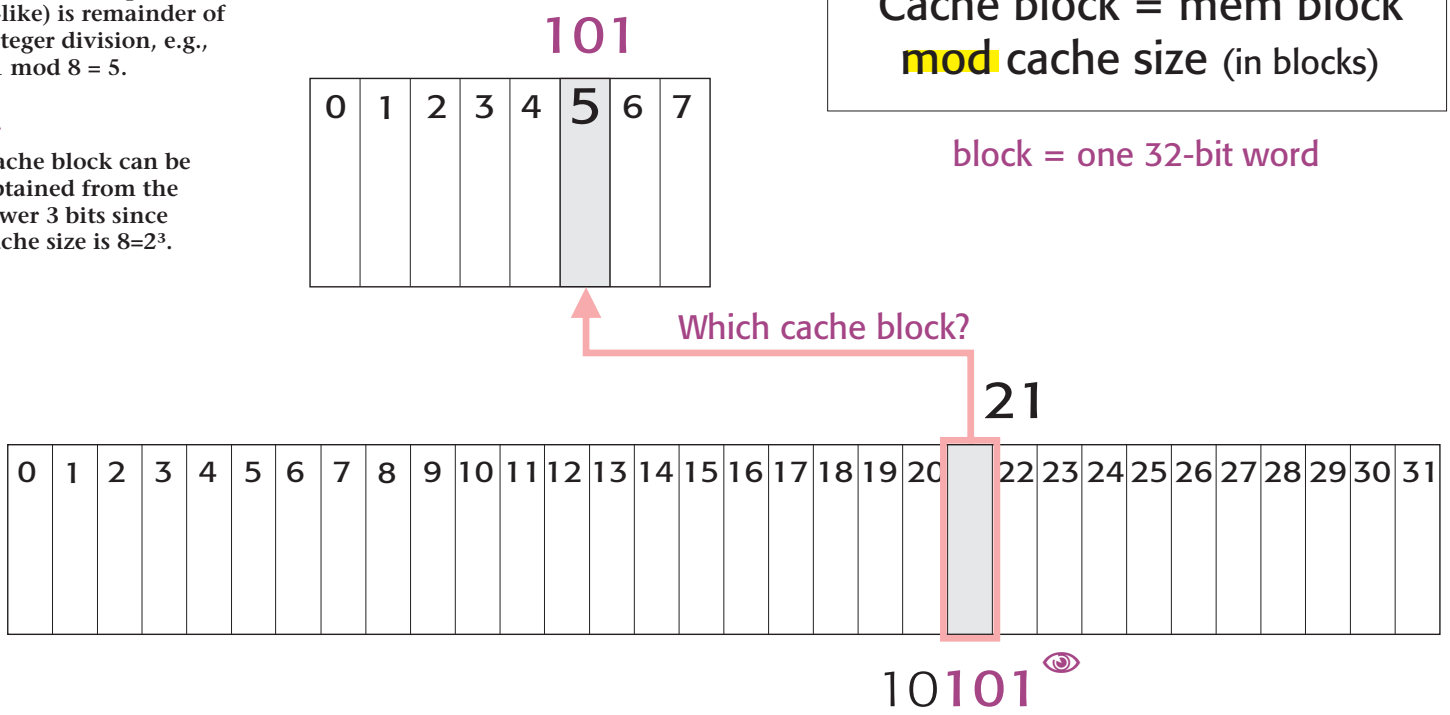
A Basic Cache
Direct Mapped

Modulo (% operator in C-like) is remainder of integer division, e.g., 21 mod 8 = 5.

Cache block can be obtained from the lower 3 bits since cache size is 8=2^3.

Cache block = mem block **mod** cache size (in blocks)

block = one 32-bit word



Direct Mapped Cache
Block Placement

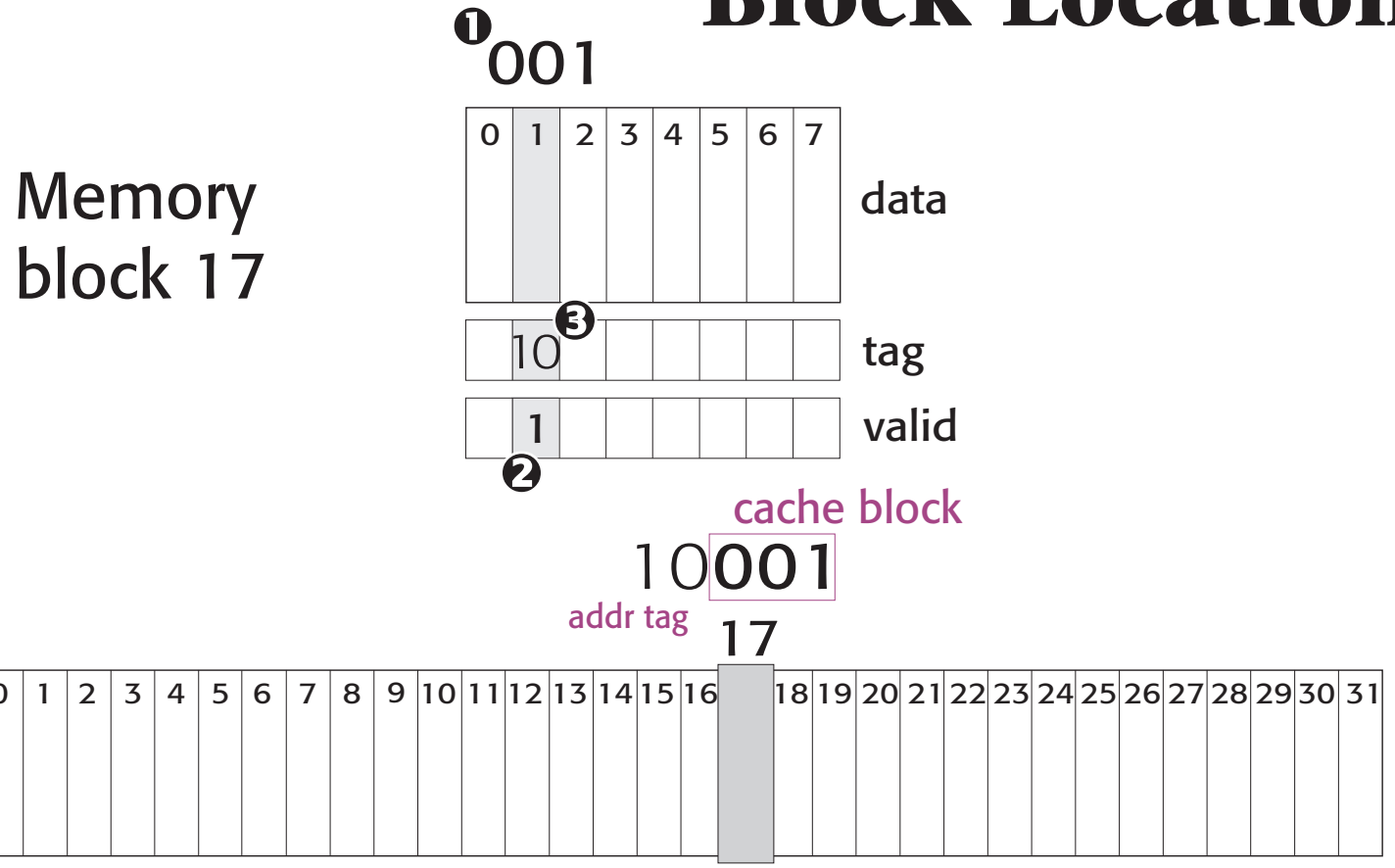
Cache block = mem block % cache size (blocks)

Formula maps many memory blocks to one location in cache .

Contents of memory block 21 are currently available within close reach in the cache

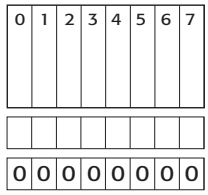
Tag bits indicate which one of a set of memory blocks sharing the same cache location is currently in memory.

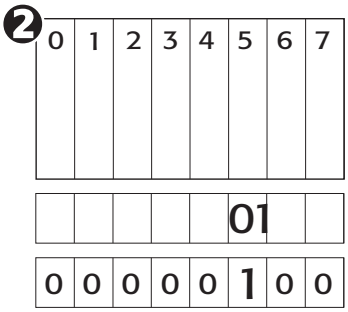
Direct Mapped Cache
Block Location

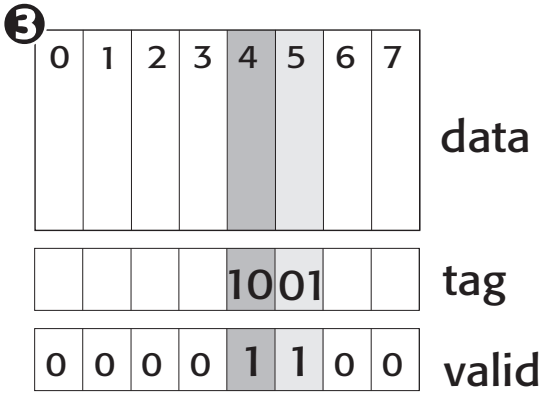


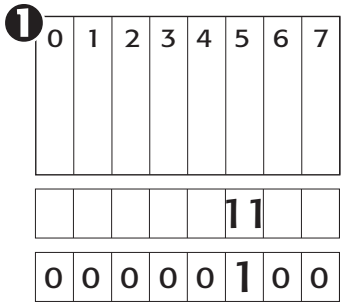
Direct Mapped Cache
Cache Operation

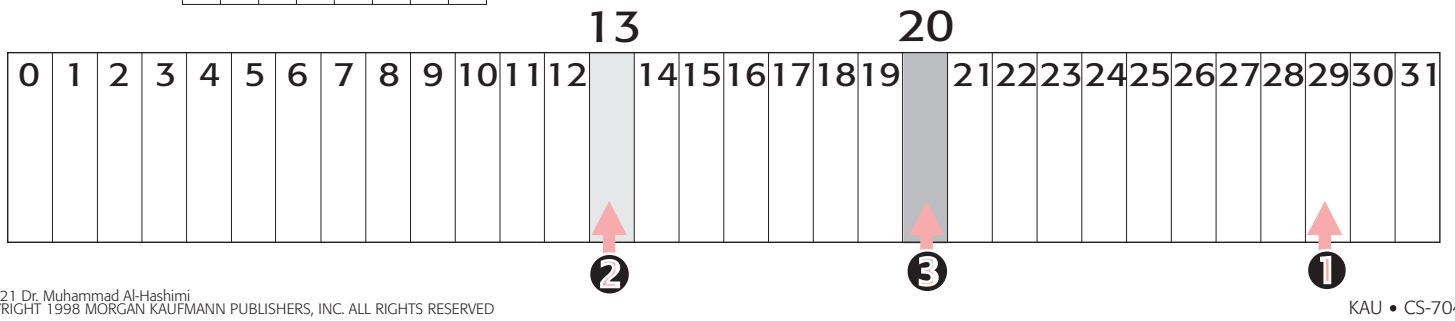
× 29
× 13
× 20
✓ 13
× 29
✓ 20











A Basic Cache
Main Concepts Review

Why wrong question: is it in cache *or* main memory? (Answer on this slide).

- ⇨ **Block mapping: where to look in cache?**
  - ✎ Block location: find in cache
  - ✎ Block (re)placement
- ⇨ **Miss/hit ratio**
- ⇨ **Miss penalty**

Quiz
Which cache op is associated with which memory access instructions in MIPS? (Hint: tricky).

If in cache then must also be in memory, by definition of a hierarchy! Right question: can it be referenced quickly.

Clearly, lw: mem reads, and sw: mem writes, however cache ops are more elementary. Both reads and writes attempt to locate blocks in cache, and on miss both (re)place blocks. Addresses requests in prev slide could have come from any mix of lw/sw.

Direct Mapped Cache
Multi-word Blocks

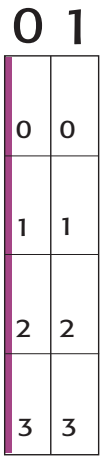
1-word blocks use temporal locality only!

Block = 4 words

Same 8-32 block cache-memory is re-organized to better utilize more locality (now spatial in addition).

2 cache blocks numbered 0-1

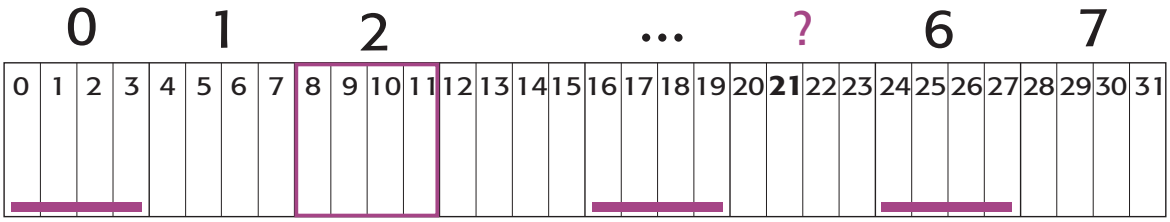
Memory block 1 is shown to be currently loaded in cache block 1.



mem block = ⌊ word-addr ÷ words-per-block ⌋

⌊21/4⌋ = 5

Quiz
Use formula to find **memory blocks** where words 5, 21, 24 belong? Verfy in figure.



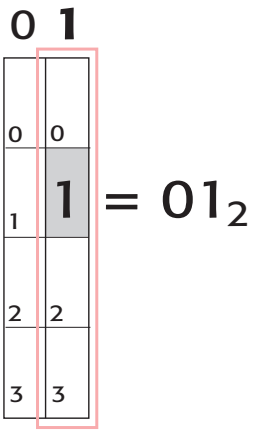
8 memory blocks numbered 0-7

Multi-word Direct Mapped Block Mapping

block = 4 words

cache block = mem block % 2

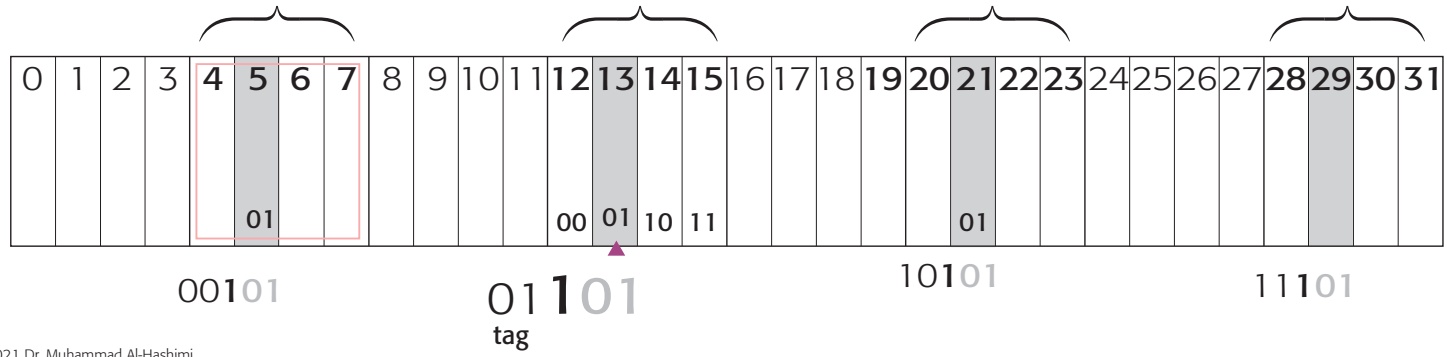
mem block = ⌊ word-addr ÷ words-per-block ⌋



Quiz
Use formula to determine the **cache block** for words 5, 21, 24. Verify answers in figure.

(⌊21/4⌋ = 5) % 2

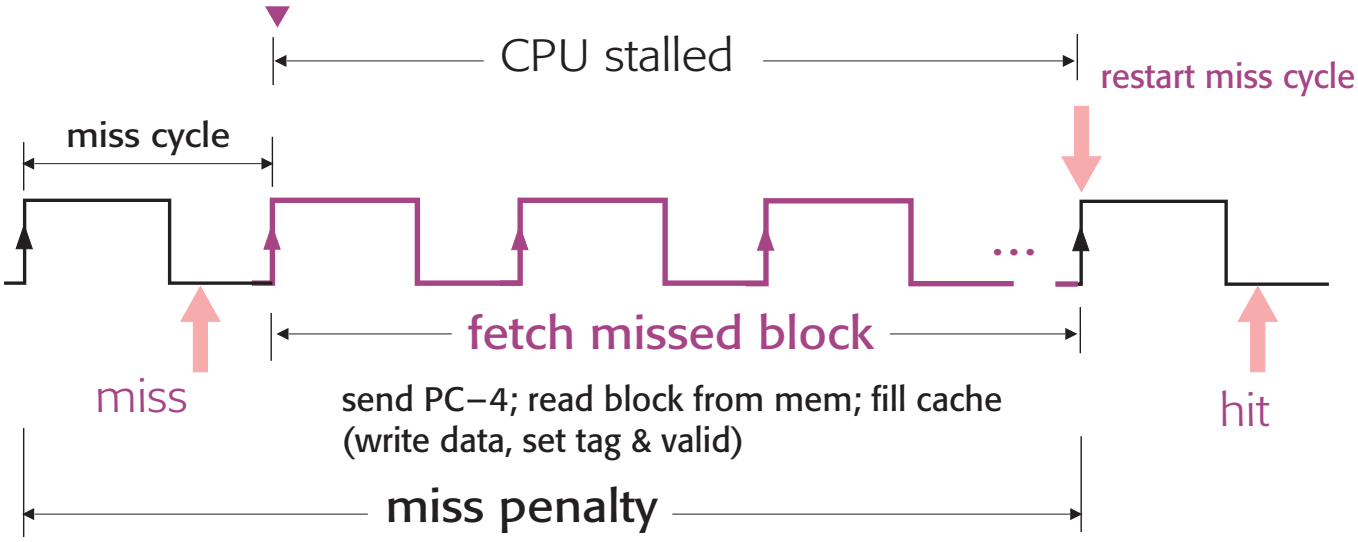
Any one of the 4 shown memory blocks could be placed in cache block 1 depending on the stored tag.



A Cache Miss

Diagram shows main concepts in a basic scenario (instr miss assumed).

Control reacts to miss after cycle closing edge (bad PC increment).



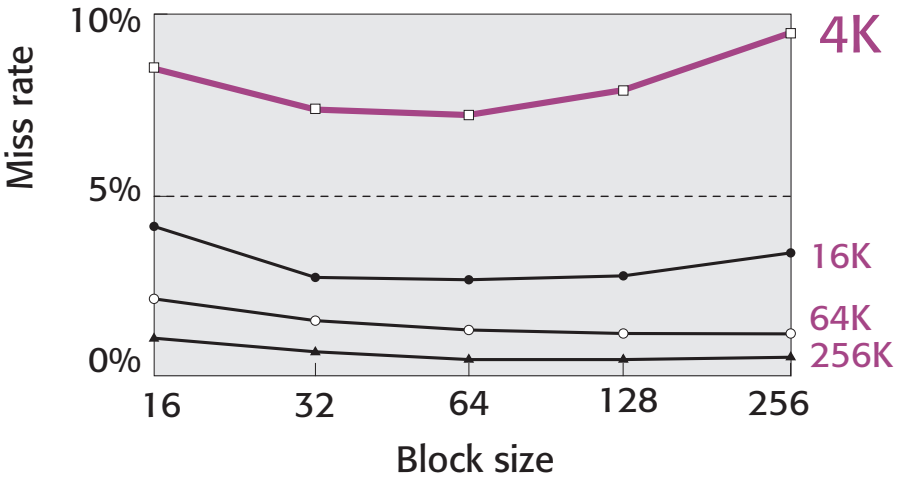
Block fetch time, in addition to block transfer time, may include setup, preparation and wait times depending on memory latency and details of interfaces to cache.

Instr and data misses are essentially similar but do implement the CPU restart slightly differently (check your textbook).

Cache Performance 101

Cache design parameter vs. performance metric

Each design parameter affects one or more performance metric.



Quiz
Which aspect of cache performance is affected by each design factors?

- ⇨ **Block size**
- ⇨ **Cache size**
- ⇨ **Split or combined**

Cache Performance 101
Understanding

➪ Block fetch time

🗝 Tradeoffs are fundamental to cache design

3 side-effects to block size increase

Quiz
Which cache performance metric is affected by each side-effect? In each case how is cache performance affected?

- ✎ Better use of spatial locality
- ✎ More competition between blocks over limited cache spots
- ✎ More time to fetch missed blocks from memory

Cache Writes

Writes cause copies in cache and memory of the same blocks to differ (become *inconsistent*).

⇨ Write hit

Block in cache, write word to cache

Quiz
Why is there no difference between write hits and misses in 1-word block cache?

⇨ Write miss (multi-word)

Fetch block then write (later)

Cache Writes
Block Consistency

There was a time when limited chip resources made the simpler write-through a practical choice.

⇨ Write-through policy

Write both cache and main mem (carbon-copy), on every write

⇨ Write-back policy

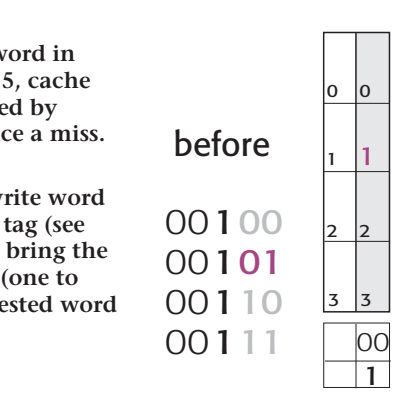
Write cache; update mem on block replacement

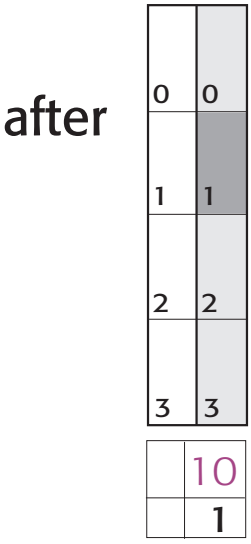
Cache Writes
Multi-word Miss

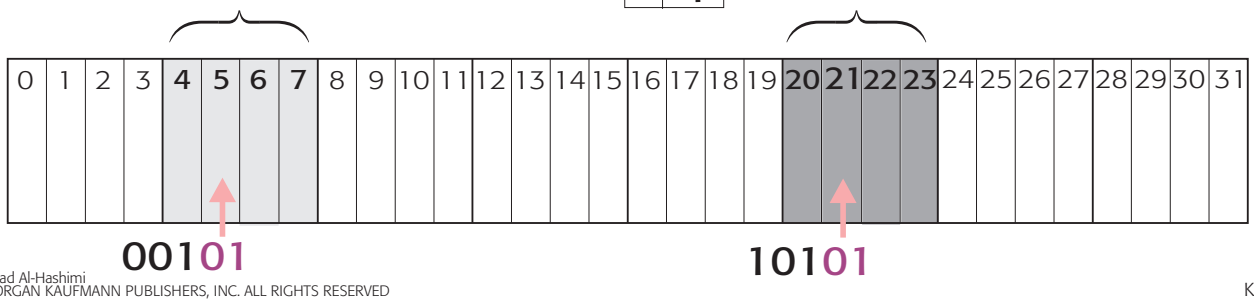
Write word 21 (10101)

Requested word in mem block 5, cache spot occupied by block 1 hence a miss.

Can't just write word and update tag (see after), must bring the right block (one to which requested word belongs).



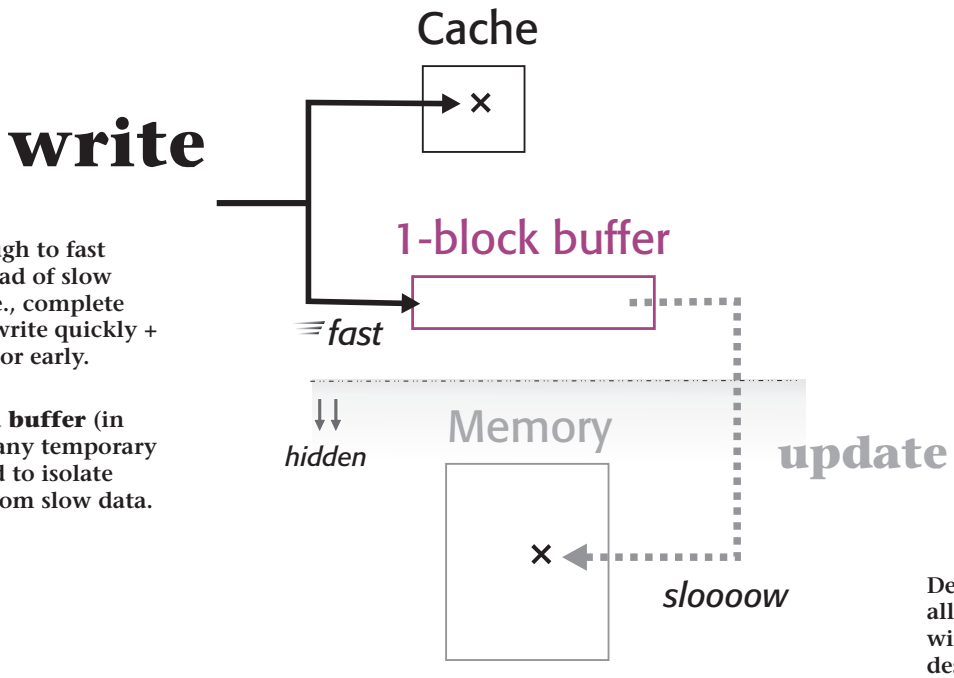




Multi-word blocks incur double penalty with write-through on write miss

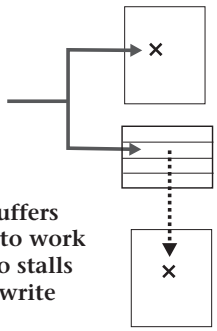
1) fetch new block, 2) update main mem after new word is written correctly if write-through.

Write Buffers



Write-through to fast buffer instead of slow memory, i.e., complete consistent write quickly + free processor early.

Generally, a **buffer** (in context) is any temporary storage used to isolate processor from slow data.



Deeper write buffers allow the CPU to work with little or no stalls despite higher write misses.

Cache Writes
Performance

A write buffer reduces CPU stalls due to memory update (fixes write-through somewhat).

➪ **Write-through via write buffer**

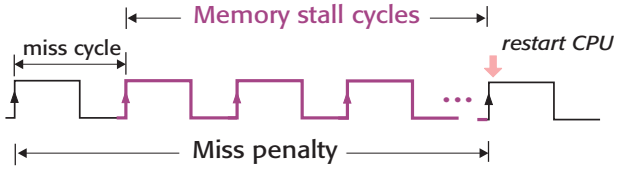
Essentially hides mem updates from CPU

➪ **Write buffer works if**

- ✎ Writes occur at less rate than mem update completion
- ✎ Writes don't occur in bursts (too many in short time)

Write bursts overwhelm buffers.

Cache Performance 101
Measurement



exec time (s) = exec cycles × cycle time

cpu cycles + memory stall cycles

❶

❷ #misses × miss penalty

= memory requests × miss rate
× miss penalty

Assumptions simplify model

1) memory stall cycles are due to cache miss, 2) write-through policy with deep write buffers (that is, reads and writes suffer similar miss penalties, mostly block fetch time).

Cache Performance 101
Improvement

memory stall cycles =

memory requests × miss rate × miss penalty (cc)

change characteristics

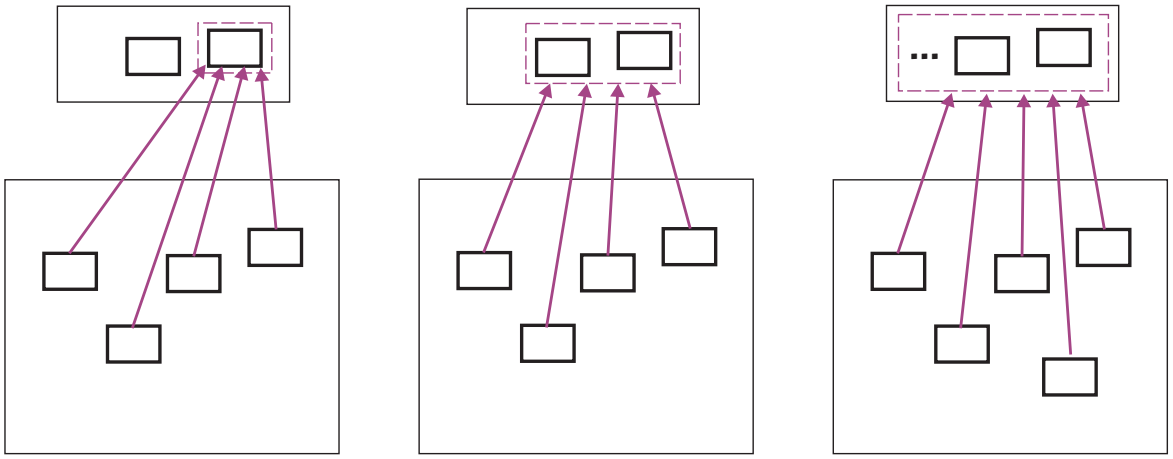
re-arrange subsystem

The number of memory requests generated by the program is not controlled by the memory system.

2 strategies

- Reduce miss rate
- Reduce miss penalty

Improving Cache Performance
Block Mapping Strategies



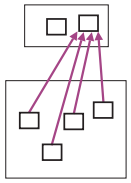
assigned seat

assigned row of seats

free seating

Seat assignment in a bus or a plane is a good example of mapping strategies.

Associative Cache



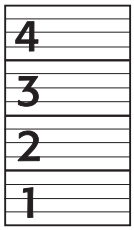
Direct-mapped

One block in cache for a memory block

n-way set associative

Map mem block to (or *associate* with) a set of blocks in cache instead of just one.

- Divide cache into sets of n>1 blocks
- Set of blocks for a memory block



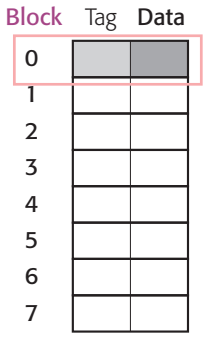
16 blocks divided into 4-block sets result in 4 sets.

sets = cache size (blocks) ÷ set size (blocks)

Fully associative: any block

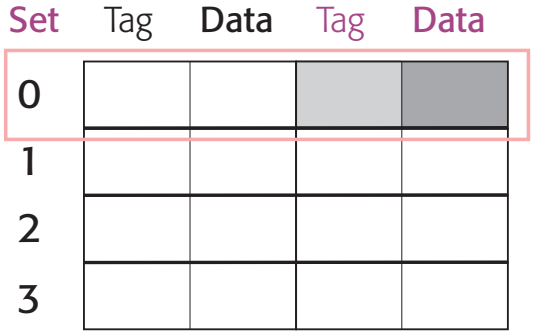
Associative Cache Block Organization

1-way set associative (direct mapped)

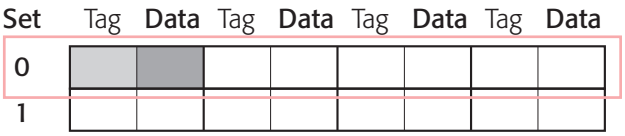


8-block Cache

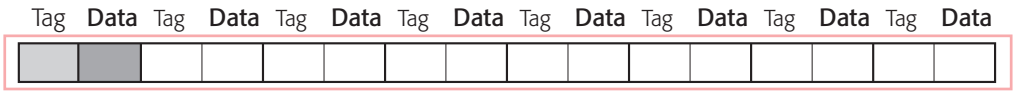
2-way set associative



4-way set associative



8-way set associative (fully associative)



Associative Cache
Block Placement

Mapping

cache set =
mem block % cache size (sets)
4

Mem block: 12, 13

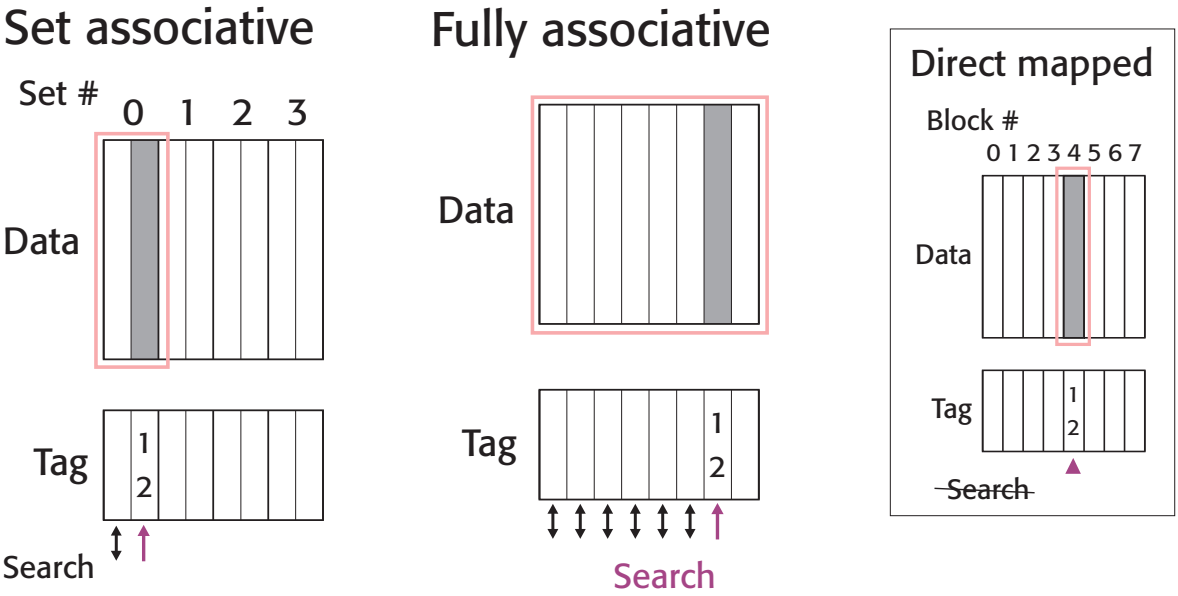
Any spot in the set is usable (block replacement is more involved in associative cache than in direct-mapped cache).

2-way set associative

Set	Tag	Data	Tag	Data
0				
1				
2				
3				

Associative Cache
Block Location

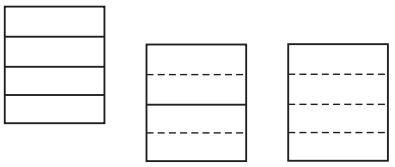
Mem block 12



Fully associative only practical for small caches since all blocks belong to one set which must be searched.

Associative Cache
Cache Operation

Block Request	Cache Block	Cache Set
0	0	0
6	2	0
8	0	0



Hit	✓
Miss	✘

Block Request	Direct Mapped	Set Associative	Fully Associative
0			
8			
0			
6			
8			

Associative Cache Performance

⇨ Miss rate improvement

FastMATH-like 64 KB (16-w block)

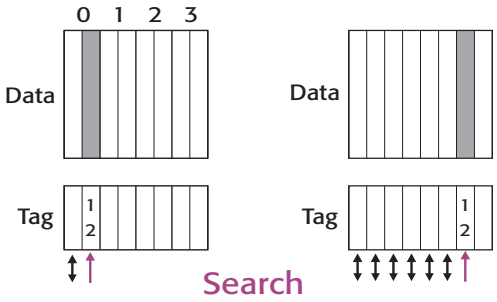
Benchmark: SPEC2000, data cache

D-mapped: 10.3%

2-way set-assoc: 8.6% (16.5% better)

⇨ Hit time

Direct mapped locates blocks via a formula hence will not involve a similar hit time.

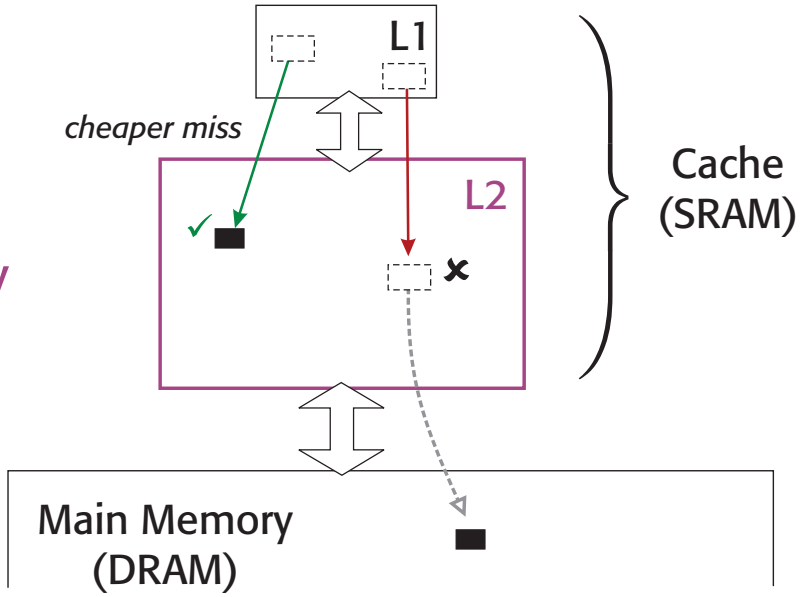


Improving Cache Performance
Multilevel Cache

2 strategies

- ✎ Reduce miss rate
- ✎ Reduce miss penalty

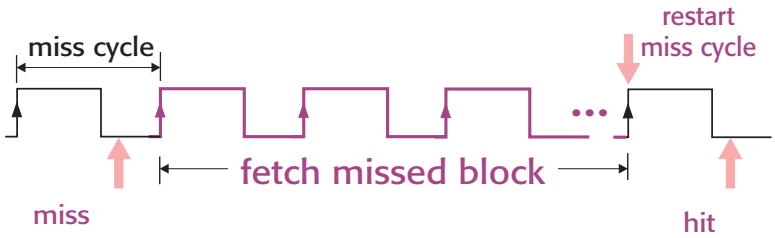
There are 2 places to look at when attempting to reduce the miss penalty: where a miss is satisfied, and the interfaces used to satisfy misses.



Multi-word Block Performance
Reducing Miss Penalty

Recall?

Writes increase the need to access memory while a larger block increases the cost of each access.



Components of miss penalty

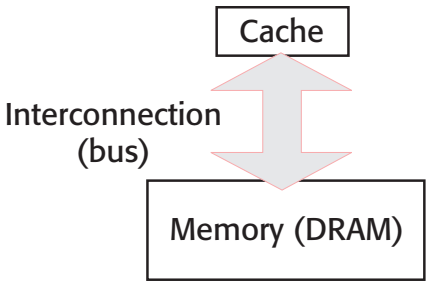
- Request (send addr) time
- DRAM first access delay (latency)
- Block transfer time

Reducing Miss Penalty
Cache-Memory Interface

Miss penalty mostly

- ✎ DRAM latency: access initiation penalty
- ✓ ✎ Block transfer: cache-mem bandwidth

Both width and clock speed affect subsystem bandwidth.



Http://en.wikipedia.org/wiki/DRAM

DRAM is available in a variety of interfaces, most recently: SDRAM (synchronous DRAM, sync with CPU over a common clock), DDR DRAM (double data rate, transfer twice per cycle).

Cache Design

Exercise
Review your textbook carefully to fill this table.

Design Factor \ Perf Metric	Miss/hit rate	Hit time	Miss penalty	Cache bandwidth
Cache associativity	Reduce block competition Flexible block placement	How?		
Block size				
Memory bandwidth				
Cache size				
Multilevel				
Split/combined				

Performance: Affect positively (increase) Affect negatively (decrease)