

A Basic Cache

⇒ Memory block

P&H = Patterson & Hennessy; to indicate stuff from/based on material from 2-4th editions of the famous text, used by author in undergrad course on the subject taught 2000-2011.

Processors deal with **byte addresses** that programs generate, which may not reference real/physical memory locations at all (actually don't).

⇒ **Processor may use any memory address but is only physically connected to locations in cache**

Memory bits are moved in **generic blocks** to hide details of how processors and instrs access memory (and other ISA and implementation stuff).

⇒ **Architecture independent unit**

A running example

Processor word, typically 2's complement operand, in the original MIPS 32 bits stored in GPR that also hold addresses.

✎ 32 memory **words** (L2)

✎ Cache 8 words (L1)

Quiz

How many bytes of memory in example?

✎ Same as processor & instr words (32 bits)

A Basic Cache Direct Mapped

⇒ Cache block

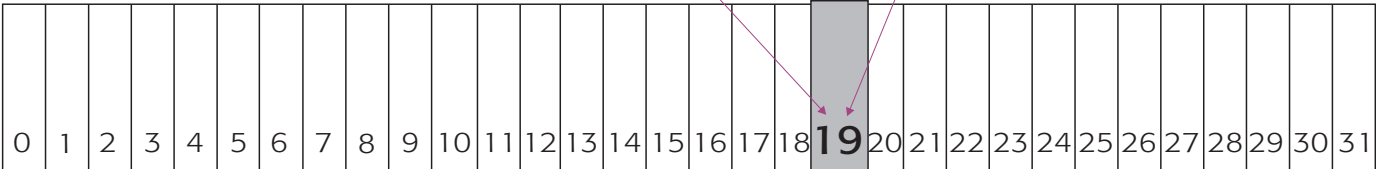
Simplest scenario for a fixed amount of memory (128 bytes) to be progressively reorganized as we go.

Modulo (C-like % operator) is remainder of integer division, e.g., $19 \bmod 8 = 3$.

Cache block = *in blocks*
Mem block **modulo** Cache size

Recall, **memory block** number originates in a reference (a byte address) generated by a running program.

block = one 32-bit word



Reproduction (example/figure concept P&H 1998-2012)

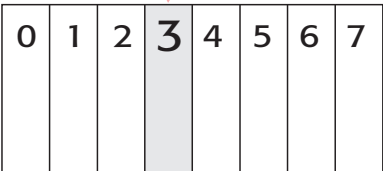


The lower 3 bits give the remainder of division since cache size is 2^k (power of 2), where $k=3$.

Quiz

Where would memory block 27 map in cache?
Hint: write the binary value.

10011
16 8 4 2 1

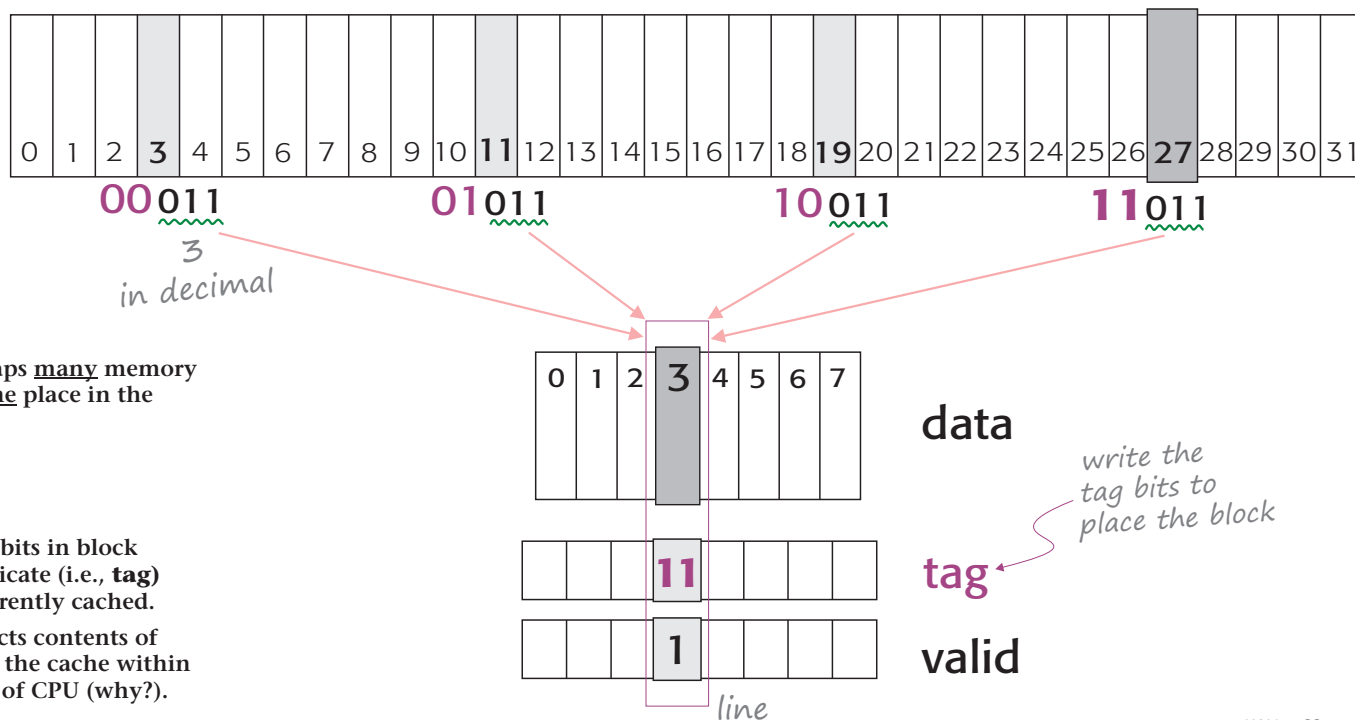


A **cache block** number, generated invisibly, is what the processor needs to access the memory item referenced by the program.

Direct Mapped Block Placement

⇒ [Address] Tag

⇒ Cache [block] line



Formula maps many memory blocks to one place in the cache.

Remaining bits in block address indicate (i.e., **tag**) the one currently cached.

Figure depicts contents of block 27 in the cache within close reach of CPU (why?).

Direct Mapped Block Location

⇒ **Cache index**

is it in the cache?

17

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
---	---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----



Cache indexed via directly accessible bit fields within a memory address.

Exercise

List the other memory block numbers that map to cache block 1. **Hint:** start with right-most index bits 001, work backwards.



The tag is the only portion of an address to save on silicon (somewhere in the hardware).

10 001
match addr tag determine index

data

tag

valid

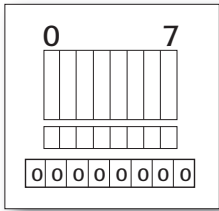
0	1	2	3	4	5	6	7
	10						
	1						

A requested block is in the cache if it is valid and its addr tag matches

check if info valid

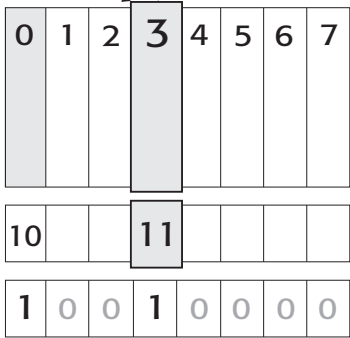
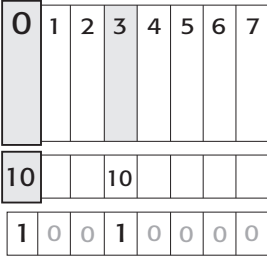
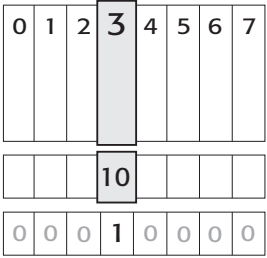
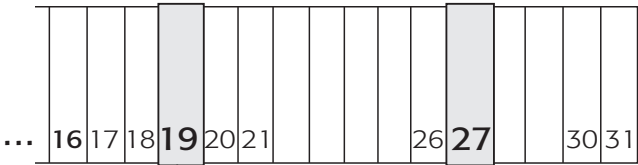
Direct Mapped Cache Operation

⇒ Compulsory miss



✓ Hit ✗ Miss

- ✗ 19 ⌚
- ✗ 16 ⌚
- ✗ 27 ⌚
- ✓ 27
- ✓ 16
- ▶ ✗ 19 ⌚



👁 The initial 3 misses must occur (**compulsory**), whereas the last one (marked) is only due to competition on the same cache block.

Exercise
Construct the block address by inspecting the cache contents in figures.

A Basic Cache Review: Main Concept

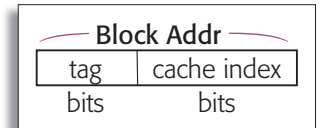
Why wrong question to ask if block is in cache or main memory? (Answer on this slide).

⇒ **Block mapping: a foundation, relates seats/spots in the cache**

Size of index bits depends on the number of cache lines (blocks), e.g., 6 bits index 64 lines.

 **Block location** (find in cache)

 **Block (re)placement**



Quiz

Associate cache ops with memory access instructions in MIPS. (**Hint:** i.e., loads and stores; tricky).

⇒ **Miss/hit ratio**

⇒ **Miss penalty (later)**

If in cache then must also be in memory, by definition of a hierarchy (if not in main memory it wouldn't be cached in the first place). The question is can it be referenced quickly? Clearly, lw: mem reads, and sw: mem writes. Address requests in prev slide could have come from any mix of lw/sw. However cache ops are more primitive. Both reads and writes attempt to locate blocks in cache, and on miss both (re)place blocks.

A Basic Cache Design Review

⇒ Spatial locality

Direct mapped, 1-word block

 Simple (block addr same as word)

A minimal block transfer time involving only one processor word (instruction or operand perhaps) that may be accessed repeatedly in a short time.

 Fast (find quickly + relatively min miss penalty)

 Temporal locality only (no nearby addrs)

Hit and miss similar, involving a direct write to cache with the same overhead.

 Simplified writes (create *versions*^{squint} of a block)

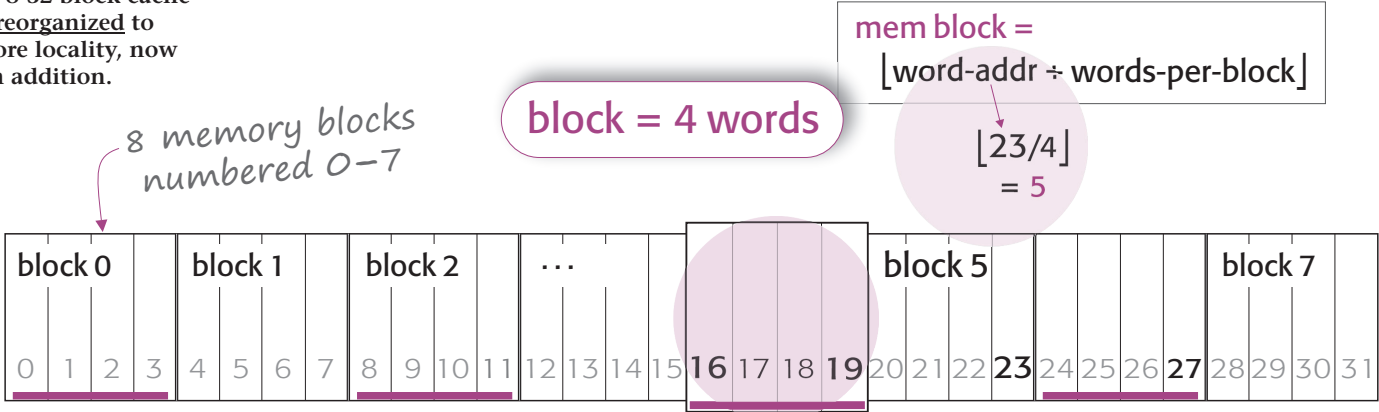


The 2nd miss (16) could perhaps have been avoided if some neighboring words to 19 were included in the block (i.e., some **spatial locality**).

 Miss ratio? Clearly a desirable performance metric

Improving Performance

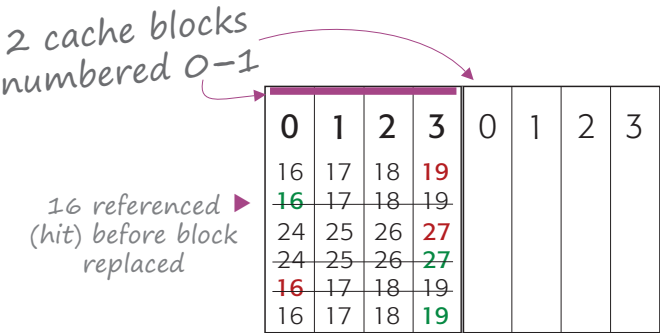
The same 8-32 block cache-memory reorganized to utilize more locality, now spatial, in addition.



Quiz
Use formula to find **memory blocks** where words 16, 19, 27 belong. Verify in figure.

✗	19
✓	16
✗	27
✓	27
✗	16
✓	19

Lower miss ratio in previous sequence of memory requests due to spatial locality (words 19 and 27 still in *competing* blocks).





Improving Performance Multi-word Mapping

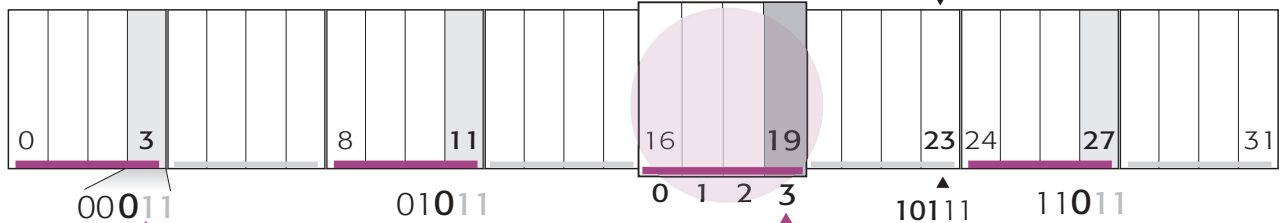
Quiz

Use formula to determine the **cache block** where words 0, 8, 16, 24 map. Verify answers in figure.

$$\text{Cache block} = \text{Mem block} \% 2$$

$$\text{mem block} = \lfloor \text{word-addr} \div \text{words-per-block} \rfloor$$

$(\lfloor 23/4 \rfloor = 5) \% 2$ *Still cache size (wider blocks)*

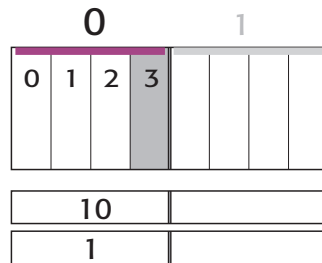


A **word offset** field (grayed) in word address now selects words inside a block (numbered internally 0-3). List the words numbered 3 in marked mem blocks.

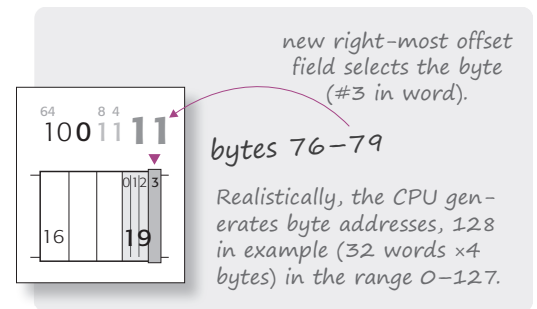
Dropping offset bits is the same as dividing to convert address units.

Example: byte-addr $95 \div 16$ (bytes/block) = block-addr 5, same as dropping 4 right-most bits since $16=2^4$.

last byte in word 23



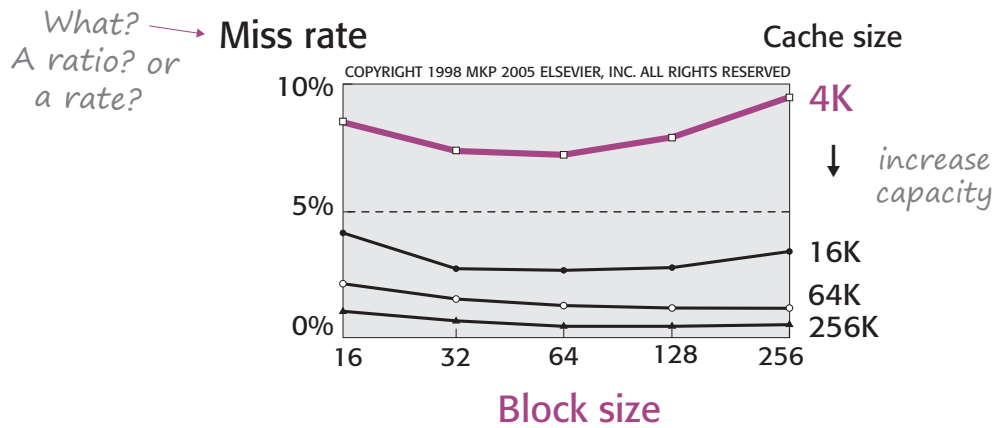
block 4 10011



Cache Performance 101

⇒ Miss rate

⇒ Capacity miss



A design parameter may affect more than one performance metric in different ways.

⇒ Block size

Quiz
Specify consequences of using chip space for one big instr + data cache?

⇒ Cache size: split or combine?

Cache Performance 101 Understanding

⇒ Conflict miss

Quiz

Identify compulsory misses for the request scenario in the multi-word cache. Compare to the 1-word block cache.

⇒ Lower miss rate 

Better use of spatial locality

initially less misses

Quiz

Which performance metric is affected in each case? State how.

⇒ Side-effects 

 More competition between blocks over limited cache spots

 More time to fetch a missed block from memory

 *tradeoffs are fundamental to cache design*



Most words unused before a block is replaced, underuse more likely as blocks become wider (ironically, less use of better spatial locality) + rise in less productive bus traffic.

A Basic Cache Design Review 2



Less tags to save vs 1-word blocks (why? *Ans. next slide*). What about total tag bits?

Write misses may incur additional overheads (later).

Misses can be reduced but those due to conflict may become more prominent.

Direct mapped, multi-word block

- ✎ Temporal and spatial locality
- ✎ Still fast finds (but careful with miss penalty)
- ✎ Better miss rate (with conflict miss issues)
- ✎ Size prevents some misses due to capacity

Pursuing lower miss ratios

increased
eye penalty

More conflicts lead to larger blocks replaced too soon at a higher cost before reasonably utilizing their locality, a by-product of (direct) mapping of blocks.

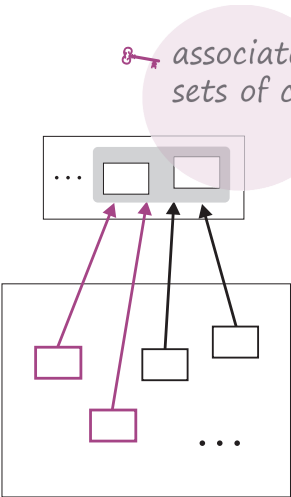
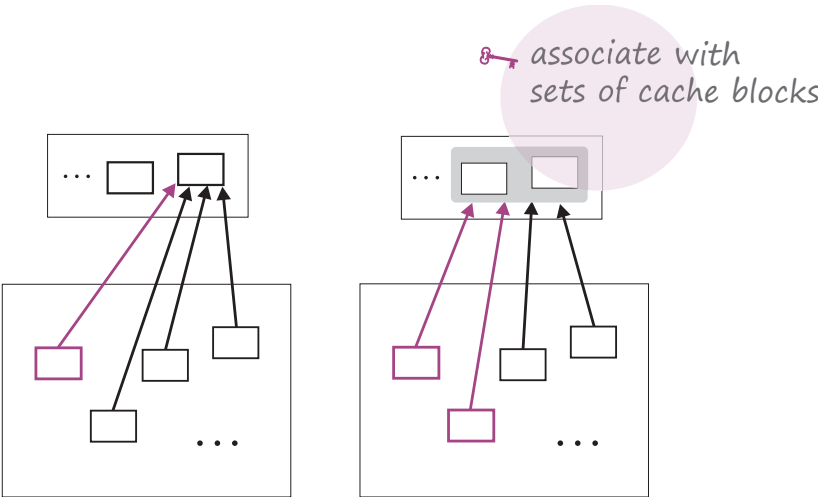
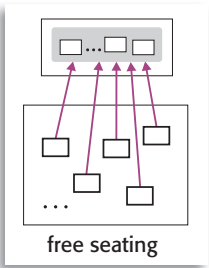
- ✎ Pressures to keep misses in check
- ✎ Rethink block mapping

Improving Performance Mapping Strategies

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

Tough luck if more than one of the passengers assigned an overbooked seat showed up (overbooking a 2-seat row could accommodate up to two at a time).

Overbooking all seats as a set (the trip) causes issues only if there is no more room.

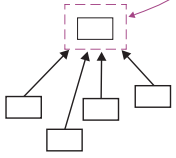


A wide block leads to fewer blocks, therefore, fewer tags since each one needs a tag. In the running example, 4-word blocks resulted in 8 blocks-tags rather than 32 for the 1-word cache. Bits in a tag, however, increase.

Associative Cache

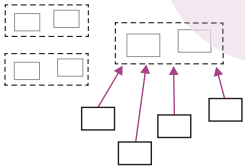
⇒ **Multi-block mapping**

set-of-one? Blocks individually indexed



⇒ **Direct-mapped**
Map to one block in cache

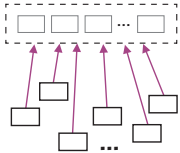
Map a mem block to set of cache blocks, not just one (fig: 2-way); blocks *fade*, mapping sees sets.



⇒ **n-way set associative**

✎ Divide cache into sets of $n > 1$ blocks

✎ Map to set of blocks in cache

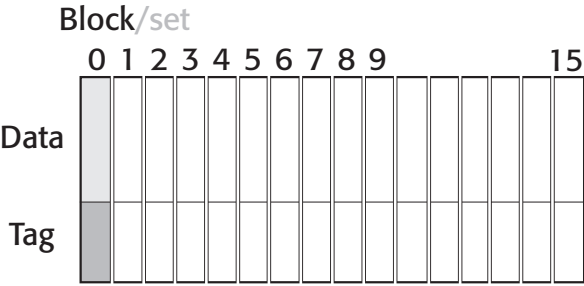


⇒ **Fully associative**
Map to all cache blocks (place anywhere)

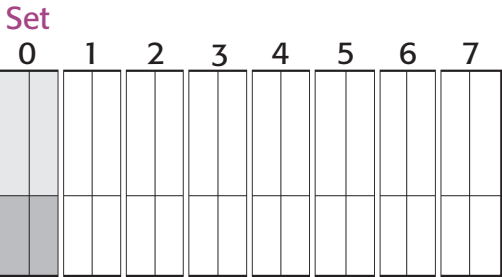
Associative Cache Block Organization

A 16-block Cache

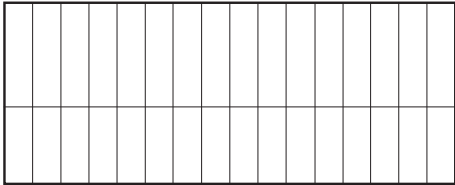
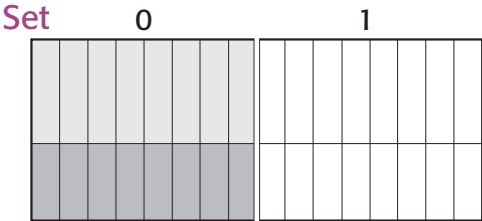
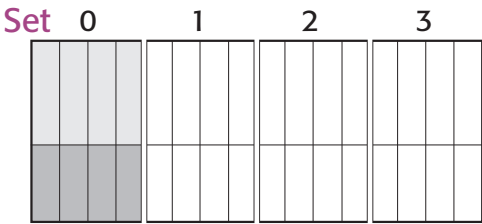
Same 16 blocks re-organized to different degrees of associativity.



Direct mapped (1-way)



2-way set associative



Fully associative (16-way)

Note the **mappable unit** (numbered) and resulting cache size from mapping formula viewpoint.

Verify the **index field** indicating set number (block# for 1-way) in the subfigure.

4
sets

Mem block: 12, 15, 8, ...

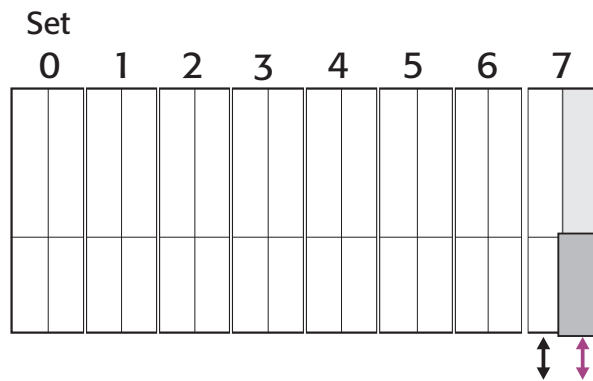
Byte-addr 102
(1 to 16-way)

	Set 0	Set 1	Set 2	Set 3
Data	12			
Tag				

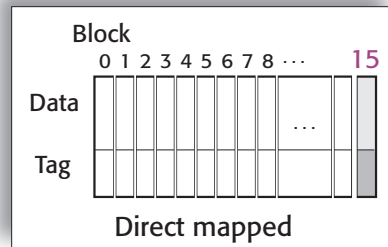
...00110
...00110
...00110
...00110
...00110

KAU • CS-704 16

Associative Cache Block Location



Mem block 15
(recall placement)

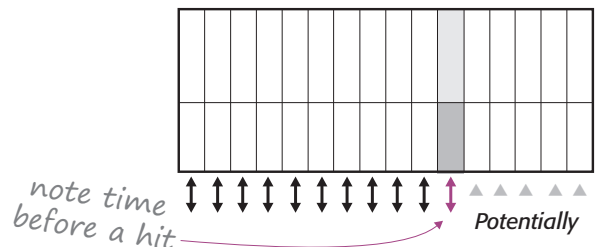
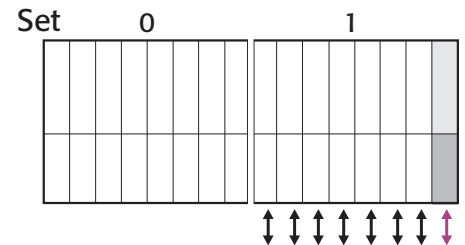
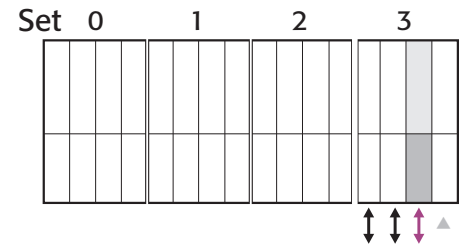


Exercise
Compare for each case: miss, replacement, and find for the scenario: fresh 31 followed by 15 (assuming 15 was already in the shaded block).



Fully associative only practical for small caches since all blocks may have to be checked.

...01111xxxx
Tag
part of addr



Associative Cache Operation

Exercise
Use formulas to verify: block numbers (formula Slide 9), and the cache index in each case.

Essentially, two fresh requests competing for the same mappable unit (block/set) must generate the same compulsory misses in all caches (marked ►).

Words from up to 2 competing memory blocks may be accommodated by both associative caches.

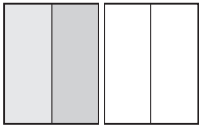
Another word from a 3rd competing block, after a compulsory miss on all (including replacement on 1/2-way), will hit on a subsequent reference only in the fully associative.

Exercise
Suggest a word from a 3rd competing block. Write the miss ratio for each case. *Hint: write words in mem blocks 0-9.*

$$\text{cache index} = \frac{\text{mem block}}{\% \text{ cache size (sets)}} \rightarrow 4\text{-word}$$

✗ 19
✓ 16
✗ 33
✓ 33
✓ 16
✓ 19

2-way set associative
2 sets (0-1)



fresh requests

Direct mapped

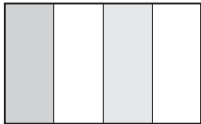
Word	Blk	Cache
✗ 19	4	0
✓ 16	4	0
✗ 33	8	0
✓ 33		
✗ 16		
✓ 19		

1-way (direct)
4 sets (0-3)

16	17	18	19
16	17	18	19
32	33	34	35
32	33	34	35
16	17	18	19
16	17	18	19

✗ 19
✓ 16
✗ 33
✓ 33
✓ 16
✓ 19

4-way (fully associative)
1 set



Associative Cache Performance



⇒ Miss rate improvement

P&H: FastMATH-like 64 KB (16-w block)

Benchmark: SPEC2000, data cache

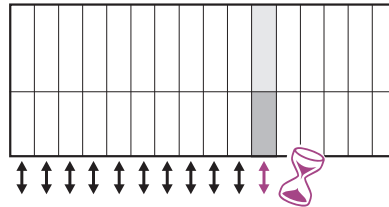
D-mapped: 10.3%

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



⇒ Hit time: the access latency

In contrast, direct mapped locates blocks via a formula hence will not involve a similar effect.



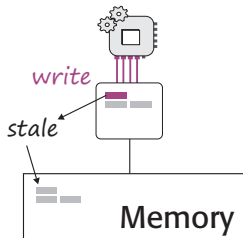
Cache Writes

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

⇒ **Write hit in general**
Block in cache, write word to cache

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

⇒ **Write miss (multi-word)**
Fetch correct block then write (later)



⇒ **Block versions: *stale* info**
A block in cache will have other copies elsewhere in a hierarchy

Cache Writes Block Consistency

⇒ **Write-through policy**

Simpler, may be a practical choice when chip resources are limited.

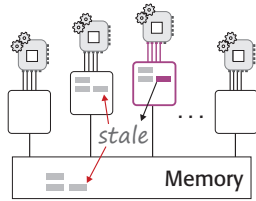
Write both cache and main memory (carbon-copy) on every write

⇒ **Write-back policy**

Write cache; update memory on block replacement

⇒ **Cache coherence**

Same blocks in processor-level (private) caches



0	1	2	3				
16			19				

10	
----	--

1	
---	--

0	1	2	3				
x	x	x	✓				
16		18					

⊕ 11	
------	--

1	
---	--

← Correct block (write cache)
→ Update memory (write-through)

Double penalty with write-through on write miss

Write Buffers

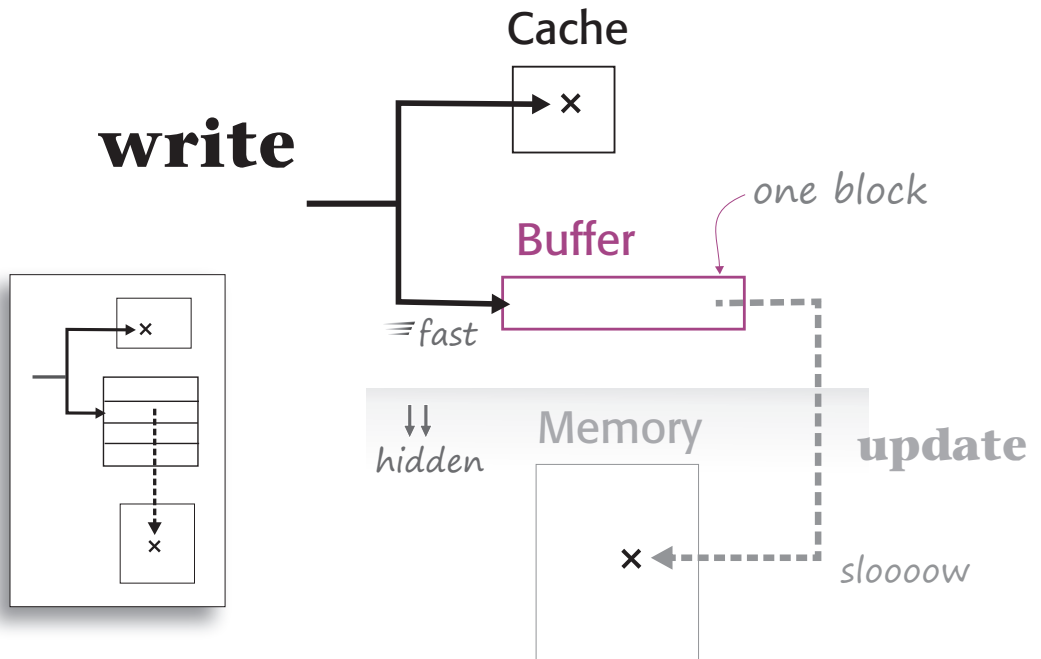
Write-through a fast buffer instead of directly to slow memory, i.e., complete a **consistent** write quickly + free processor early.

Deeper write buffers (4 in fig) allow the CPU to work with little or no stalls despite higher write misses (up to 4 updates in-progress may be accommodated).

Generally, a **buffer** (in the context) is any intermediate storage used to isolate access (e.g., fast from slow or read from write or user from system).

Quiz

What if writes occur in large bursts?



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**

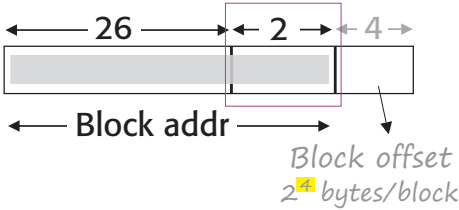
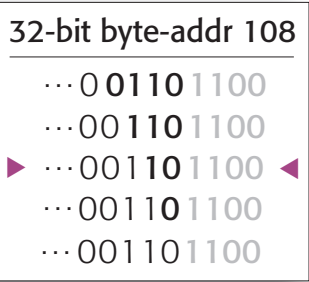
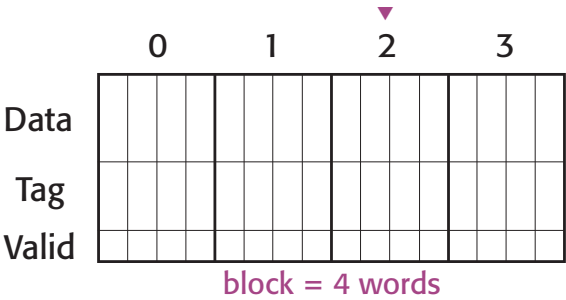
✎ Writes occur at less rate than mem update completion

✎ Writes don't occur in bursts (too many in short time)

Write bursts overwhelm buffers

Storage Cost

Quiz
Identify the organization and the amount of data bytes the cache can hold, assuming 32-bit address and data words. Where would byte-address 108 go? *Answers last slide.*



To calculate storage for 256 bytes of data (figure), determine #blocks (16, regardless of org), block size (128 bits), and tag size (26 bits, based on org), resulting in 2480 bits or 310 bytes.

Exercise
Compare storage cost for 1 to 16-way. Repeat for 16kB data.
Hint: block address length does not change (why?).

Exercise Write a formula for storage bits if #blocks is 2^n , #bytes/block is 2^m .




Observations

- Extra tag and valid bits
- Tag depends on block organization
- Tag size increases with associativity
- May need all bits in block addr to tag (when?)

Improving Performance Design Review 3

A larger or highly associative cache will eliminate more avoidable misses but may become less responsive than design requirements.



Associative (multi-block), multi-word

-  Better use of temporal & spatial locality
-  More storage & hit cost (careful with associativity)
-  Best miss rate (within capacity and responsiveness tolerances)

Exercise

Is it a good idea to restrict a cache to reads to keep design simple? Research credible data about writes to support or refute bypassing cache for memory writes.

Optimizing performance

-  Other cache design factors?
-  Component or system concerns?

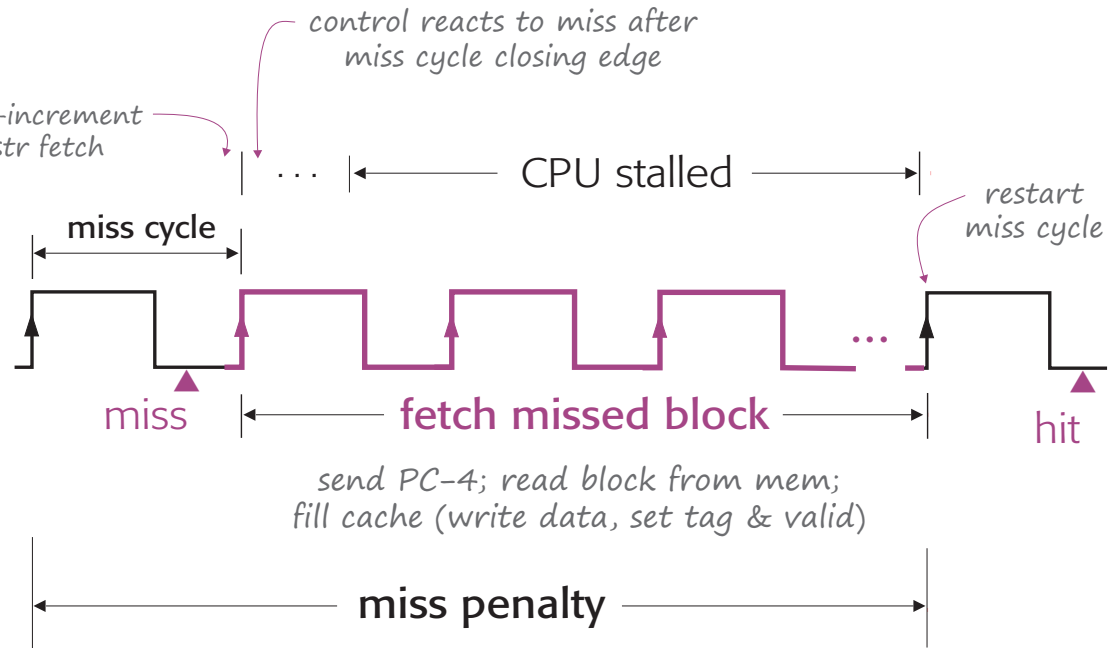
A Basic Cache Miss

⇒ Miss penalty

There is more to **miss penalty** than time to move blocks; the diagram depicts concepts in a simplified MIPS scenario (an instr miss).

note bad PC pre-increment after failed instr fetch

In addition to transfer time, block fetch may include architectural preparations (such as rolling back PC in MIPS), request set up, and wait times due to memory and interface characteristics (see *Reducing Miss Penalty* slides), during which CPU stalls.





Cache Performance 101

A Simplified Model

Focus on cycles added by memory (hits excluded since hidden in CPU cycles).

Assume mem stall cycles are due to cache miss only (isolate cache effects). **1**

A write miss is complex since writes must manage changes in copies of the same block.

Assume write-through policy with deep write buffers (make reads and writes suffer similar miss penalties, mostly block fetch time). **2**



2nd assumption makes the cost of write miss (in cycles) similar to read so that they may be counted together (combined) to simplify.

$$\text{exec time (s)} = \text{exec cycles} \times \text{cycle time}$$

$$\text{CPU cycles} + \text{memory stall cycles} \quad \text{1}$$

come from
reads and writes

$$= \text{\#misses} \times \text{miss penalty} \quad \text{2}$$

$$= \text{memory requests} \times \text{miss rate} \times \text{miss penalty}$$

Cache Performance 101 Improvement

Dividing the formula by an IC gives a memory-based CPI to add to a processor-focused CPI based on a perfect cache (that always hits) to obtain a slightly more realistic CPI/IPC.

$$\text{memory stall cycles} = \text{memory requests} \times \underbrace{\text{miss rate}}_{\text{change characteristics}} \times \underbrace{\text{miss penalty (cycles)}}_{\text{re-arrange subsystem}}$$

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

Suggest 2 strategies



Reduce miss rate (e.g., associativity)



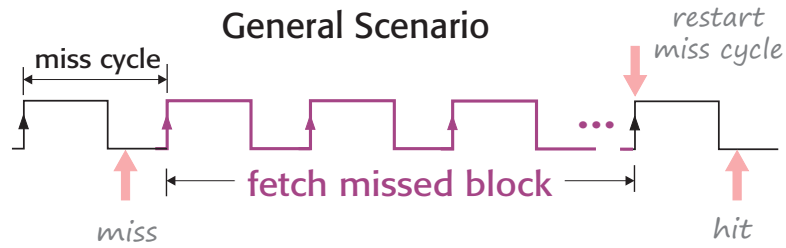
Reduce miss penalty

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

Improving Performance Reducing Miss Penalty



Check 2 places to reduce the miss penalty: a) where a miss is satisfied, and b) the interfaces used to satisfy misses.



A miss from main memory, typically DRAM, is relatively costly.

Miss addresses are typically passed to a dedicated cache controller that talks to a DRAM controller (some request setup time may also apply).

Two main interfaces are involved: a) the DRAM chips interface and b) the cache-to-memory subsystem.

Main parts of miss penalty



Request (send addr) time



DRAM first access delay (latency)



Block transfer time

Reducing Miss Penalty Multilevel Cache

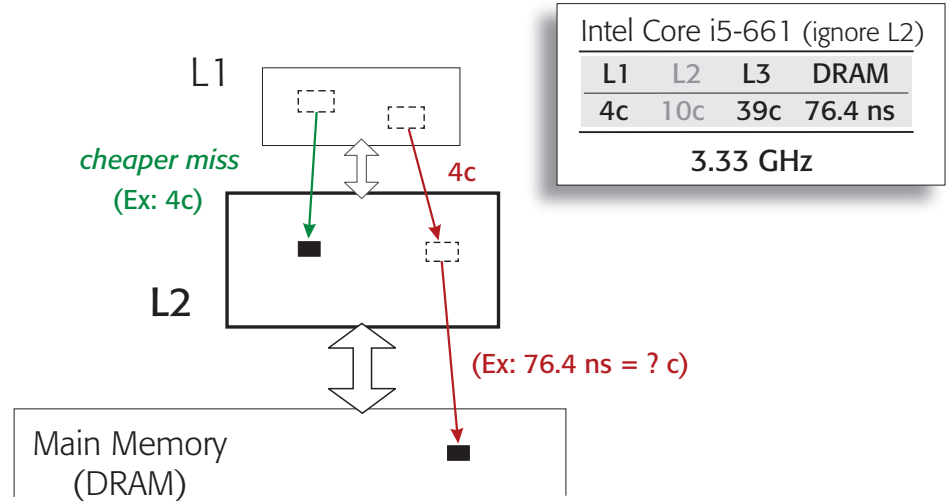
To reduce miss penalty
change: a) *where a miss is
satisfied*, ...

Design flexibility Optimized cache performance

An L2 cache reduces expensive misses from DRAM (splits original miss rate) + frees L1 to focus on keeping up with the processor (reducing access latency).

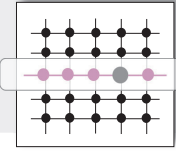
Exercise

Calculate the cost in cycles when a miss can not be satisfied from L2 (along red path).



Reducing Miss Penalty Cache-Memory Interfaces

⇒ Memory bank



Physical cell array, usually in a chip; cells accessed via a common row-column scheme.

To reduce miss penalty change: ... b) interfaces used to satisfy misses.

Cost mostly

Physical characteristics and organization of internal circuitry/devices determine the latency.

Bit width and clock speed contribute to **bandwidth**.



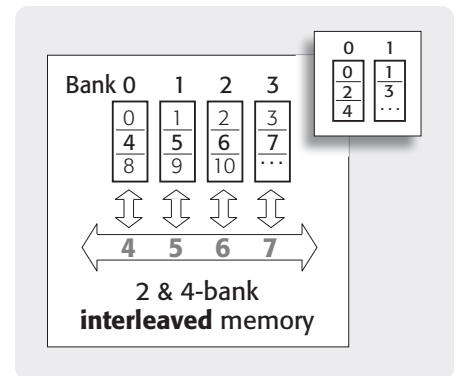
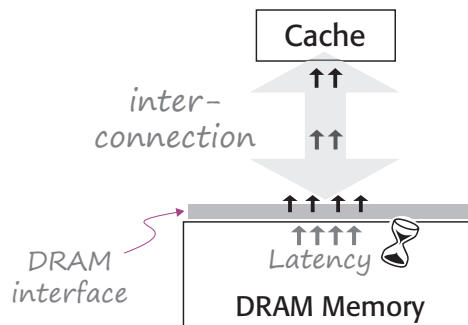
DRAM latency: cell access initiation



Block transfer: interface bandwidth

A fast subsystem can transfer a full block at once (perhaps at the same clock as the processor) or **interleave** addressing to overlap access penalty.

A fast **DRAM interface** may use more data channels or a higher clock to deliver bits off the chip package or increase data rate.



Reducing Miss Penalty Architectural Concerns

Main memory in CPU-like general-purpose processors and graphics memory in SIMD-like GPU.

DRAM: modern primary storage

⇒ Shifting complexity

Mem system vs DRAM chips package

Details of internal org and access increasingly transparent to system, DRAM interface growing in complexity.



DRAM: respond quickly, transfer in bulk



Memory system: fast transport lanes

Exercise

Outline the conflicting demands that low response time and high bandwidth place on communication channels. Give examples for solutions and tradeoffs.

⇒ A balancing act

Response time vs throughput



Reducing Miss Penalty Speeding Memory/DRAM

Exercise

Lookup terms (note architectural/system implications of move to **DDR5 SDRAM**).

Interleaving may be implemented directly by a memory system or transparently by the DRAM package via internal separately accessible memory banks (see SDRAM).

Exercise

Identify earliest Intel or AMD mainstream μ -architectures to get a DRAM controller on die.

Speed contiguous address blocks (DDR transfer data on rising and falling edges of a clock; multiple channels interleave access to data in or across modules).

Burst data size/channel doubled in DDR5 SDRAM from 8 to 16 bytes (16×2 channels/DIMM $\times 2$ transfers/clock = 64 bytes, typical cache block size, from 1 DIMM in 1 op).

⇒ **Interleaved memory** (beat latency)

⇒ **Multiple (cell-array) banks**

⇒ **Tightly-integrated controller**

⇒ **Multi-channel memory** (boost bandwidth)

⇒ **Burst mode**

⇒ **Double data rate**

⇒ **HBM: die stacking**

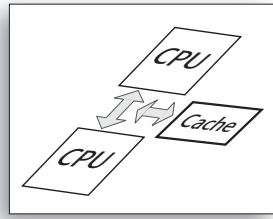
<https://en.wikipedia.org/wiki/DRAM>

DRAM chip packages historically offered a variety of ways to interface the memory cells, now mainly **SDRAM** (synchronous, i.e., transfer over a common clock) in DIMM (dual in-line memory module) chip package.



Reducing Miss Penalty High Bandwidth Cache

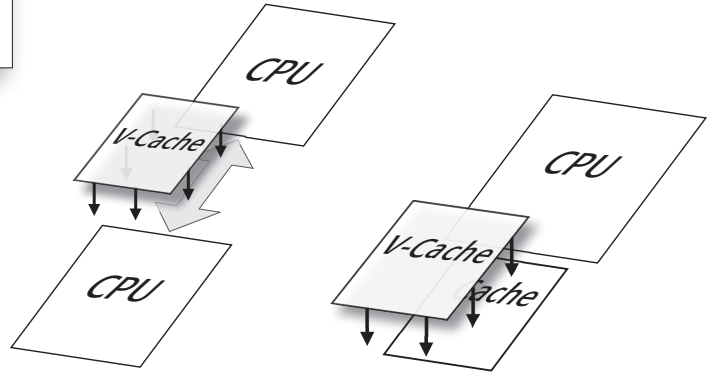
⇒ V-Cache



Planar designs, which lay caches over the same 2D plane as processor(s), incur increasing cache latency as they scale up in size and complexity due to interconnection distances.

Cache embedded in vertically stacked dies allow closer hence faster connections.

Figures depict single and multiprocessor (with shared cache) scenarios.



Quiz

Which aspect of misses (rate or penalty) should benefit?

- ⇒ **Higher density** more cache.. less chip space
- ⇒ **Less latency** more (instr/data) processing bandwidth
- ⇒ **Better thermals** more power efficiency

Exercise
Review textbook carefully to fill this table (or as you go through reading material).

Notes

.....

.....

.....

.....

.....

- Storage Cost Answers**
- 16-block (tags mark blocks)
 - 4-way set associative (groups of 4 blocks assigned a common index number)
 - Data size: 256 bytes = 16 blocks × 4 words/block × 4 bytes/word
 - 108 may be placed anywhere in set 2 (indicated by 2-bit cache index field in address)
 - Block addr length depends only on #bytes in block (16), hence always 28 bits (max tag size) for 32-bit address
 - Remember to include the valid bit in storage calculations

<div>Perf Metric</div> <div>Design Factor</div>	Miss/hit rate	Hit time	Miss penalty	Cache bandwidth
Associativity	Reduce block competition Flexible block placement👍	How?		
Block size				
Mem bandwidth				
Cache size				
Multilevel				
Split/combine				

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