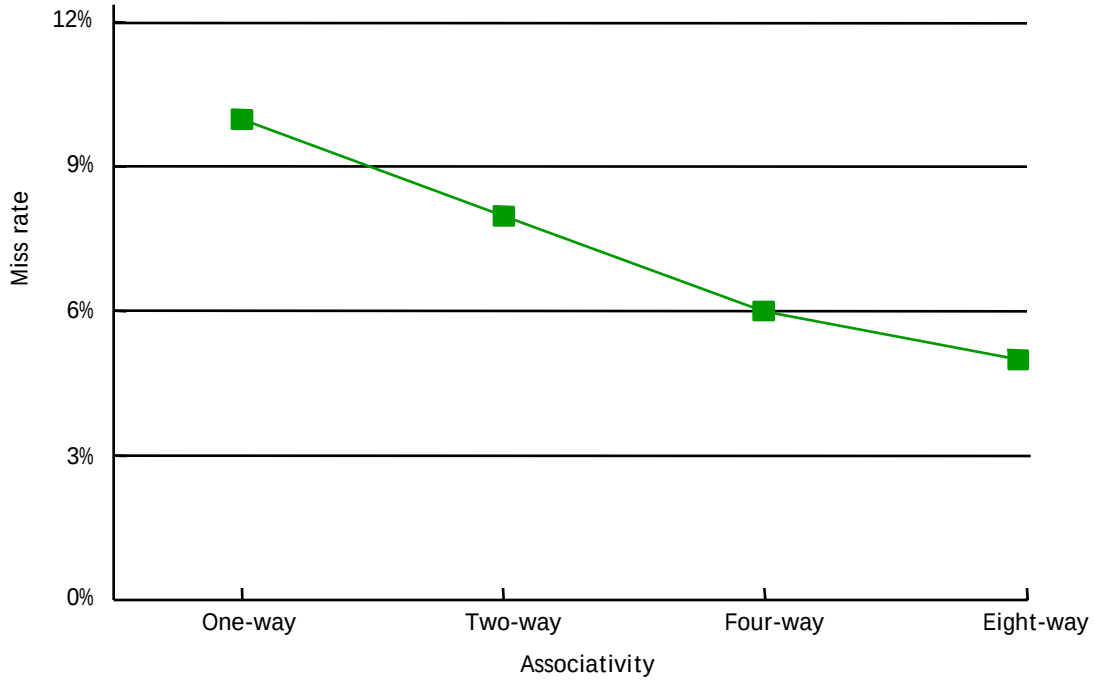
Associativity tradeoffs and miss rates

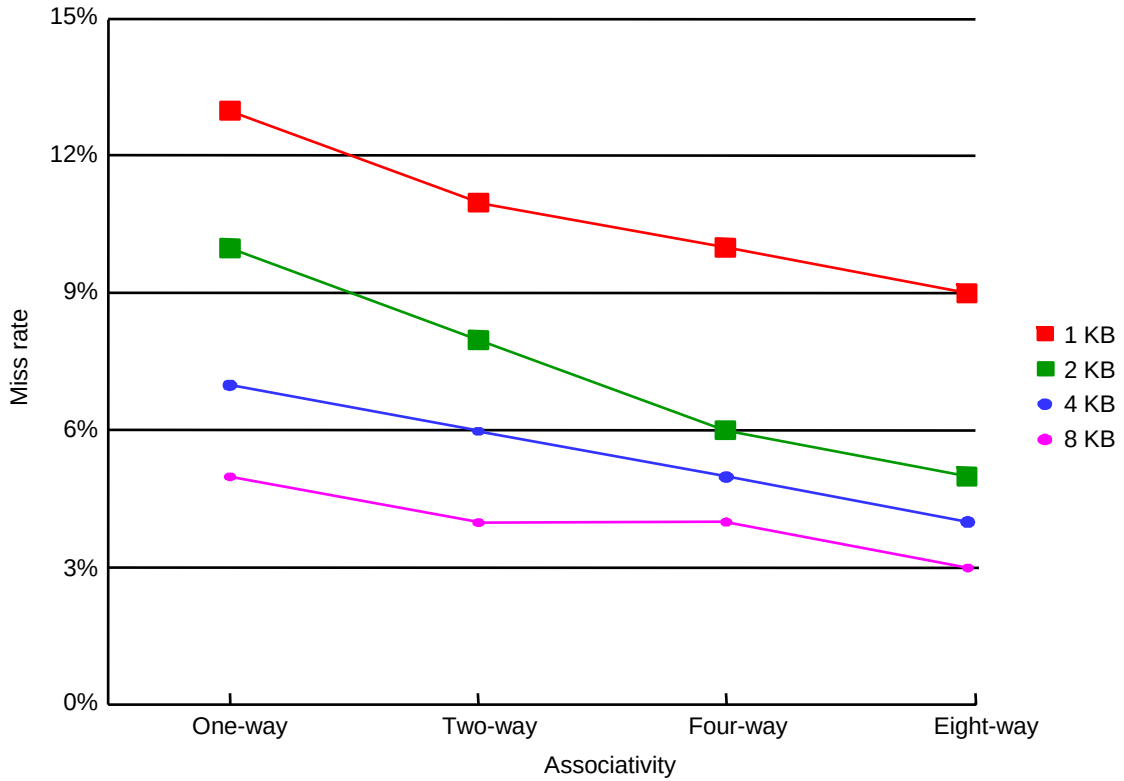
- Higher associativity means more complex hardware
- But a highly-associative cache will also exhibit a lower miss rate
  - Each set has more blocks, so there's less chance of a conflict between two addresses which both belong in the same set
- Figure from the textbook shows the miss rates decreasing as the associativity increases



Cache size and miss rates

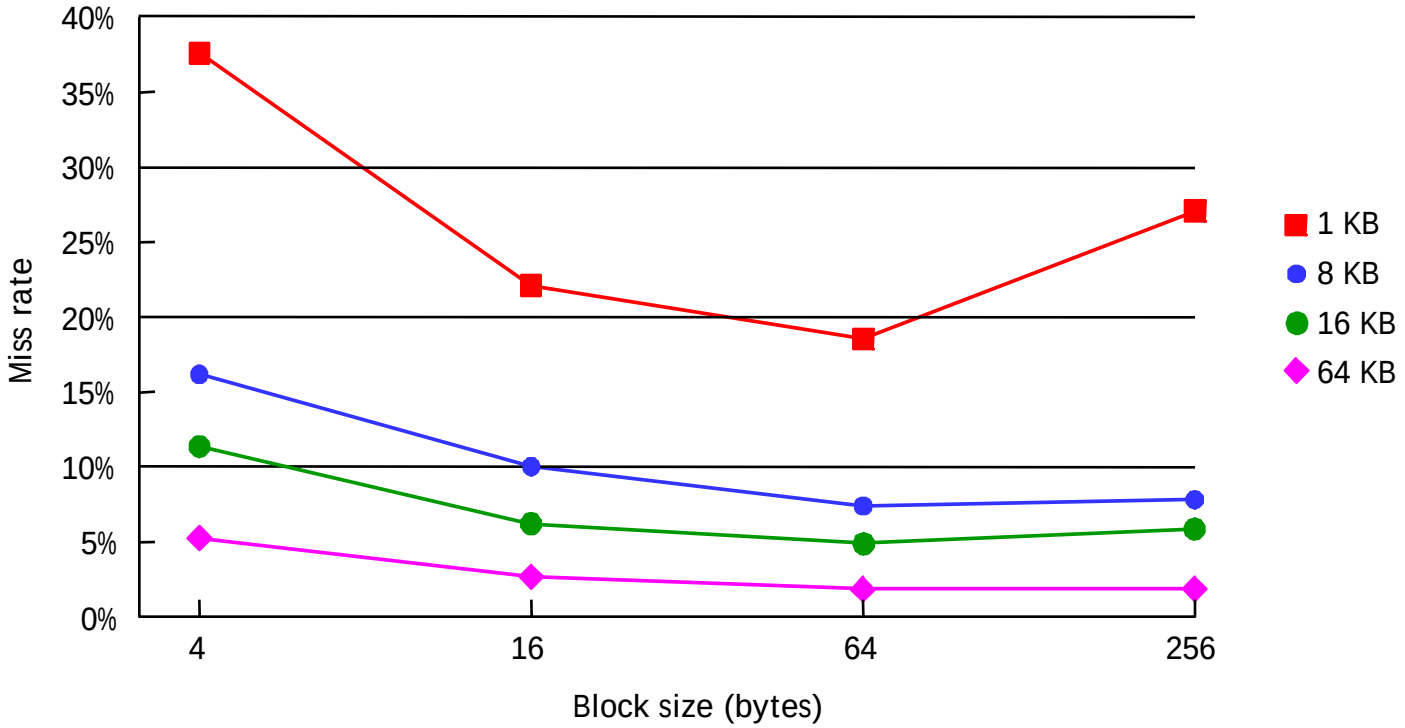
- The cache size also has a significant impact on performance
  - The larger a cache is, the less chance there will be of a conflict
  - Again this means the miss rate decreases, but the hit time increases

- Miss rate as a function of both the cache size and its associativity



Block size and miss rates

- Finally, miss rates relative to the block size and overall cache size
  - Smaller blocks do not take maximum advantage of spatial locality
  - But if blocks are *too* large, there will be fewer blocks available, and more potential misses due to conflicts



Performance example

- Assume that 33% of the instructions in a program are data accesses, the cache hit ratio is 97% and the hit time is one cycle, but the miss penalty is 20 cycles

$$\begin{aligned}\text{Memory stall cycles} &= \text{Memory accesses} \times \text{Miss rate} \times \text{Miss penalty} \\ &= 0.33\,\mathbf{N} \times 0.03 \times 20 \text{ cycles} \\ &\approx 0.2\,\mathbf{N} \text{ cycles}\end{aligned}$$

- If **N** instructions are executed, then the number of wasted cycles will be 0.2 x **N**

This code is 1.2 times slower than a program with a "perfect" CPI of 1!

Memory systems are a bottleneck

CPU time = (CPU execution cycles + Memory stall cycles) x Cycle time

- Processor performance traditionally outpaces memory performance, so the memory system is often the system bottleneck

- For example, with a base CPI of 1, the CPU time from the last page is:

CPU time = (**I** + 0.2 **I**) x Cycle time

- What if we could *double* the CPU performance so the CPI becomes 0.5, but memory performance remained the same?

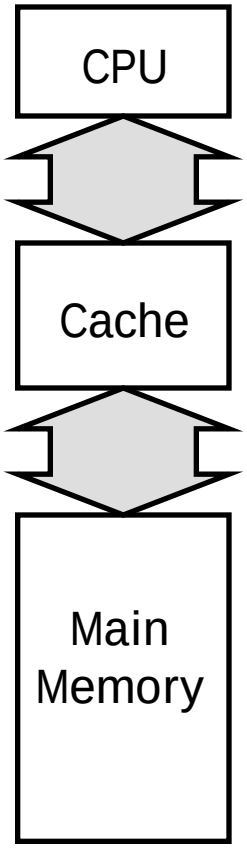
CPU time = (0.5 **I** + 0.2 **I**) x Cycle time

- The overall CPU time improves by just 1.2/0.7 = 1.7 times!

- Amdahl's Law again:
  - Speeding up only part of a system has diminishing returns

Basic main memory design

- Let's assume the following three steps are taken when a cache needs to load data from the main memory:
  1. It takes 1 cycle to send an address to the RAM
  2. There is a 15-cycle latency for each RAM access
  3. It takes 1 cycle to return data from the RAM



- In the setup shown here, the buses from the CPU to the cache and from the cache to RAM are all one word wide

- If the cache has one-word blocks, then filling a block from RAM (*i.e.*, the miss penalty) would take 17 cycles

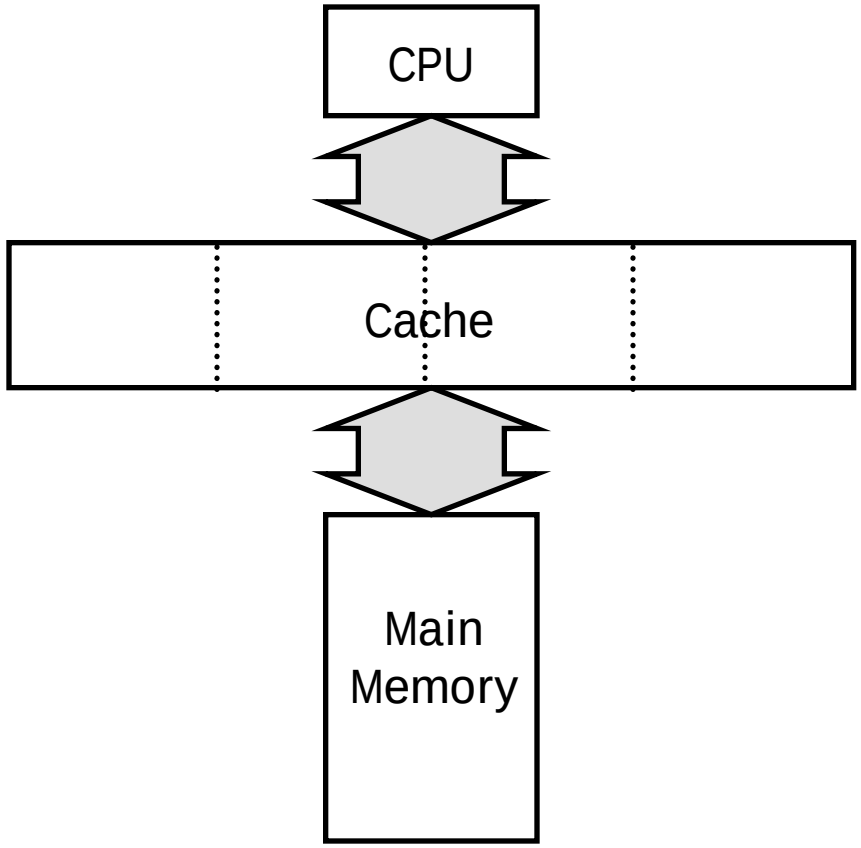
  1 + 15 + 1 = 17 clock cycles

- The cache controller has to send the desired address to the RAM, wait and receive the data

Miss penalties for larger cache blocks

- If the cache has four-word blocks, then loading a single block would need four individual main memory accesses, and a miss penalty of 68 cycles!

4 x (1 + 15 + 1) = 68 clock cycles



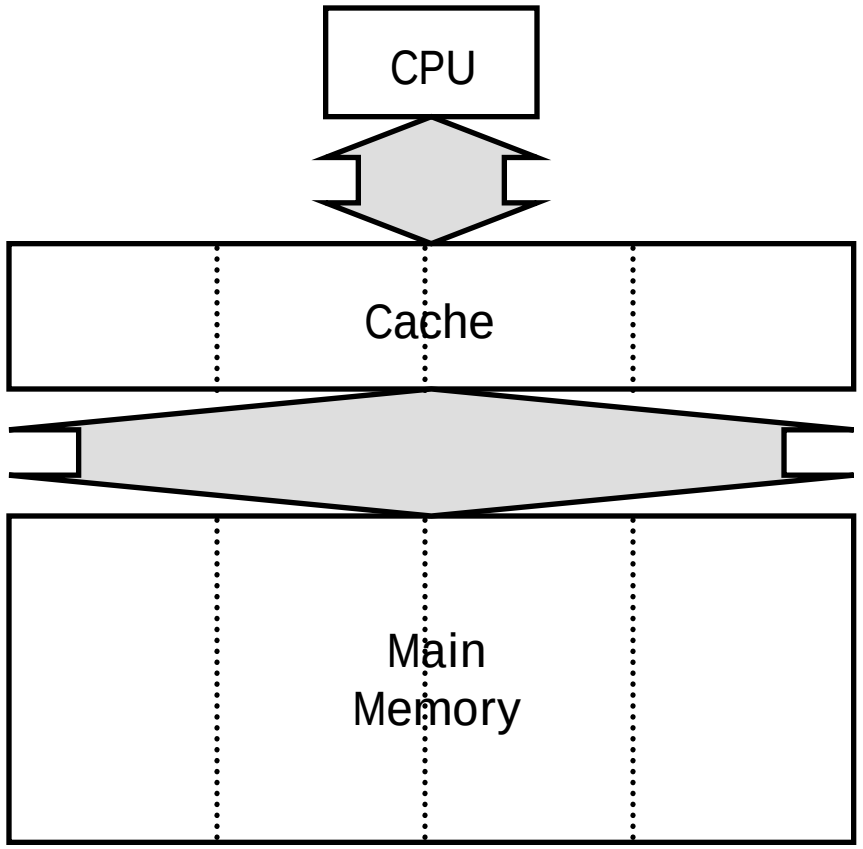
A wider memory

- A simple way to decrease the miss penalty is to widen the memory and its interface to the cache, so we can read multiple words from RAM in one shot

- If we could read four words from the memory at once, a four-word cache load would need just 17 cycles:

  1 + 15 + 1 = 17 cycles

- The disadvantage is the cost of the wider buses—each additional bit of memory width requires another connection to the cache

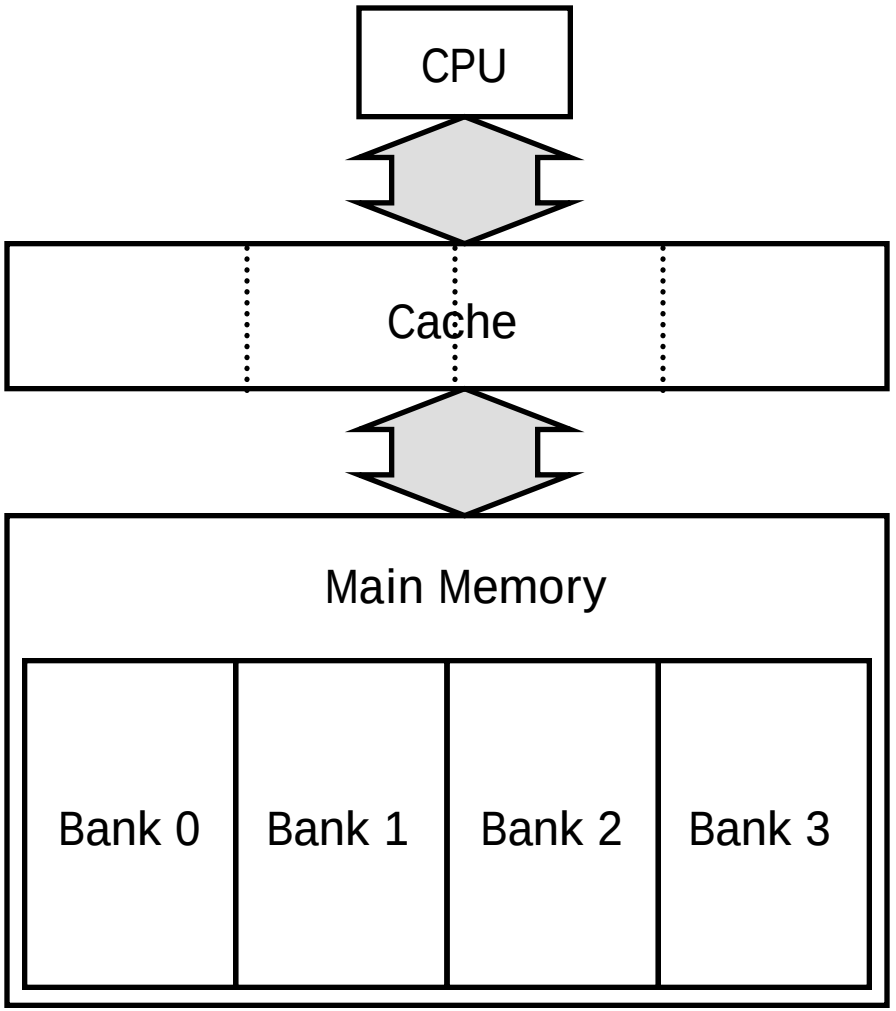


An interleaved memory

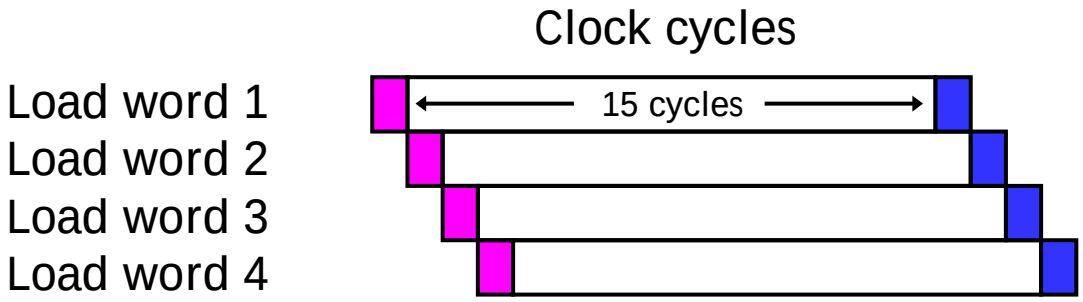
- Another approach is to interleave the memory, or split it into "banks" that can be accessed individually
- The main benefit is overlapping the latencies of accessing each word
- For example, if our main memory has four banks, each one byte wide, then we could load four bytes into a cache block in just 20 cycles

  1 + 15 + (4 x 1) = 20 cycles

- Our buses are still one byte wide here, so four cycles are needed to transfer data to the caches
- This is cheaper than implementing a four-byte bus, but not too much slower



Interleaved memory accesses



- Here is a diagram to show how the memory accesses can be interleaved
  - The magenta cycles represent sending an address to a memory bank
  - Each memory bank has a 15-cycle latency, and it takes another cycle (shown in blue) to return data from the memory

- This is the same basic idea as pipelining!
  - As soon as we request data from one memory bank, we can go ahead and request data from another bank as well
  - Each individual load takes 17 clock cycles, but four overlapped loads require just 20 cycles

Summary

- Writing to a cache poses a couple of interesting issues
  - Write-through and write-back policies keep the cache consistent with main memory in different ways for write hits
  - Write-around and allocate-on-write are two strategies to handle write misses, differing in whether updated data is loaded into the cache.

- Memory system performance depends upon the cache hit time, miss rate and miss penalty, as well as the actual program being executed
  - We can use these numbers to find the average memory access time
  - We can also revise our CPU time formula to include stall cycles

AMAT = Hit time + (Miss rate x Miss penalty)

Memory stall cycles = Memory accesses x miss rate x miss penalty

CPU time = (CPU execution cycles + Memory stall cycles) x Cycle time

- The organization of a memory system affects its performance
  - The cache size, block size, and associativity affect the miss rate
  - We can organize the main memory to help reduce miss penalties. For example, interleaved memory supports pipelined data accesses.

Inferring the cache structure

- Consider the following program

```
char a[LENGTH]; int sum = 0;
for(int i = 0; i < 10000; ++i)
  for(int j = 0; j < LENGTH; j += STEP)
    sum += a[j];
```

- Key idea: compute the *average* time it takes to execute **`sum += a[j]`**

	STEP							
LENGTH	1	2	4	8	16	32	64	128
8	7	7	7	7	6	7	7	7
16	8	7	7	6	6	7	7	7
32	7	8	7	7	7	7	6	7
64	16	21	45	45	7	8	8	9
128	14	25	47	45	45	7	7	7
256	17	25	43	45	45	46	8	7

average time (in ns)

- What is the cache size (data)? What is the block size? What is the associativity?