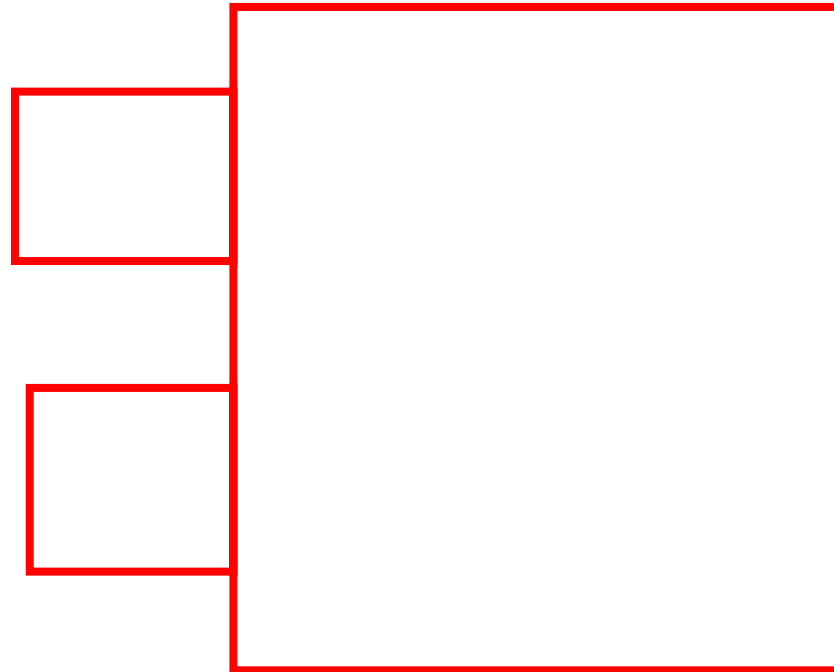


Real Designs



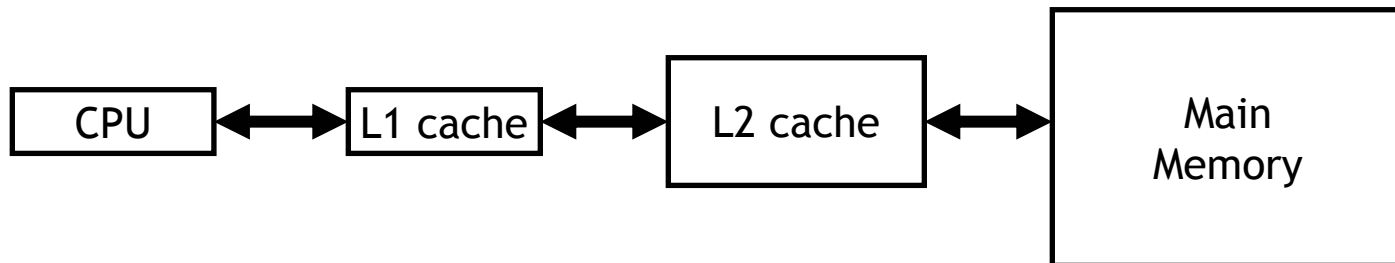
First Observations

□ Split Instruction/Data caches:

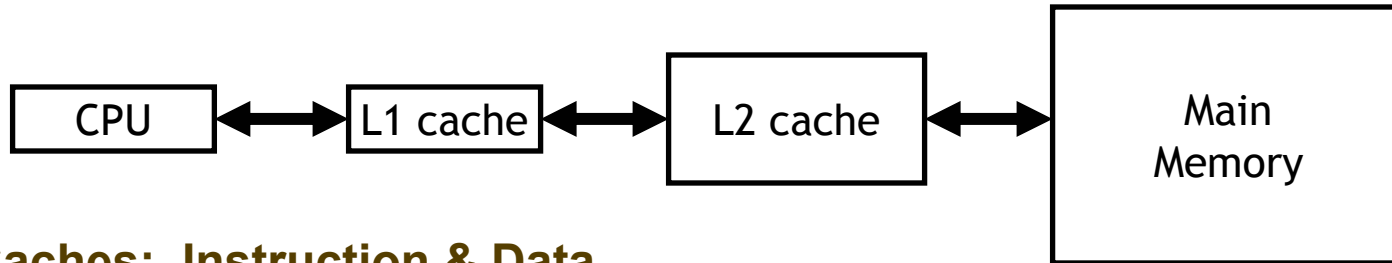
- **Pro: No structural hazard between IF & MEM stages**
 - A single-ported unified cache stalls fetch during load or store
- **Con: Static partitioning of cache between instructions & data**
 - Bad if working sets unequal: *e.g.*, `code/DATA` or `CODE/data`

□ Cache Hierarchies:

- **Trade-off between access time & hit rate**
 - L1 cache can focus on fast access time (okay hit rate)
 - L2 cache can focus on good hit rate (okay access time)
- **Such hierarchical design is another “big idea”**
- **We saw this in section.**



Vital Statistics



□ L1 Caches: Instruction & Data

- 64 kB
- 64 byte blocks
- 2-way set associative
- 2 cycle access time

□ L2 Cache:

- 1 MB
- 64 byte blocks
- 4-way set associative
- 16 cycle access time (total, not just miss penalty)

□ Memory

- 200+ cycle access time

Comparing cache organizations

- ❑ **Like many architectural features, caches are evaluated experimentally.**
 - As always, performance depends on the **actual instruction mix**, since different programs will have different memory access patterns.
 - **Simulating** or **executing** real applications is the most accurate way to measure performance characteristics.

- ❑ **The graphs on the next few slides illustrate the simulated miss rates for several different cache designs.**
 - Again lower miss rates are generally better, but remember that the miss rate is just one component of average memory access time and execution time.
 - You'll probably do some cache simulations in CS161L.

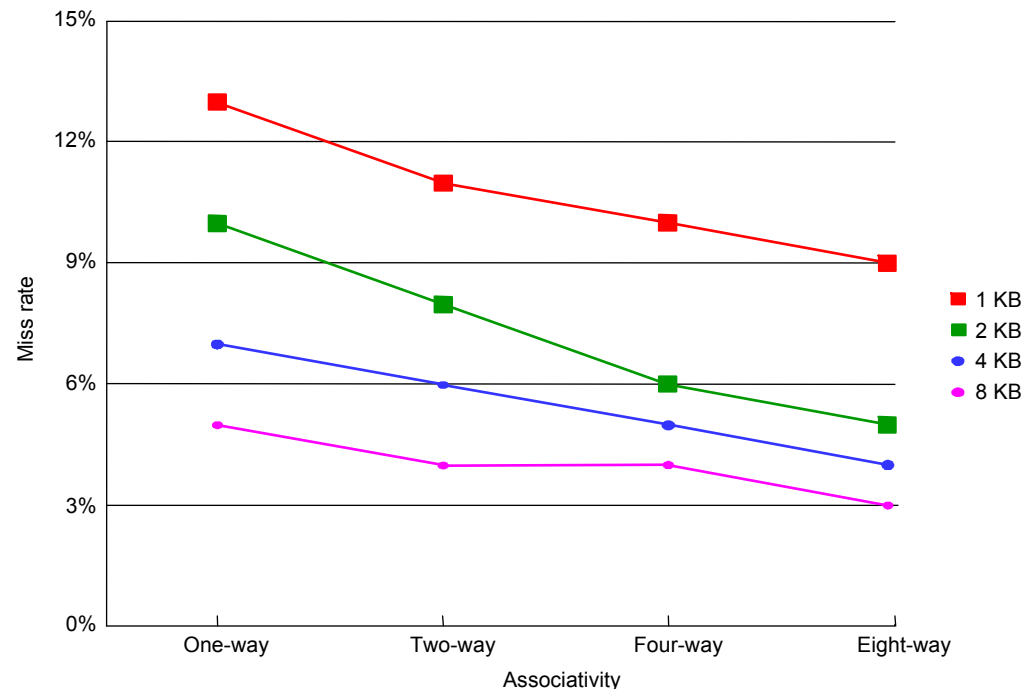
Associativity tradeoffs and miss rates

- ❑ As we saw last time, 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.
 - Overall, this will reduce AMAT and memory stall cycles.



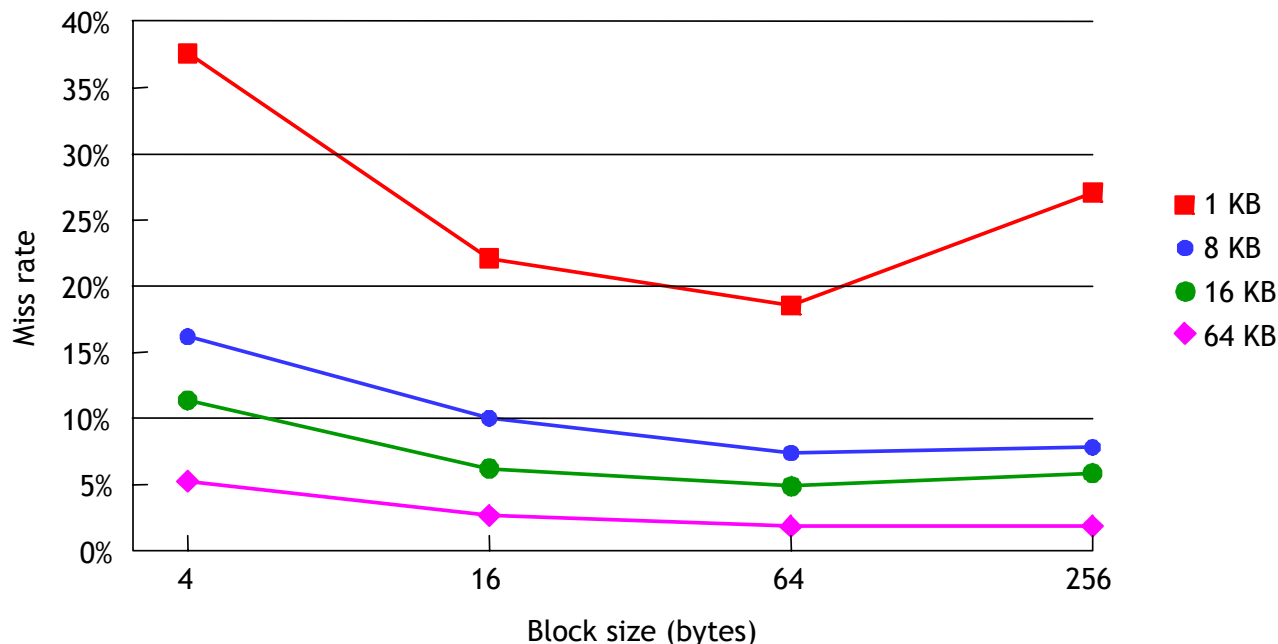
Cache size and miss rates

- 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, so the AMAT and number of memory stall cycles also decrease.



Block size and miss rates

- Finally, the 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.



Memory and overall performance

- ❑ How do cache hits and misses affect overall system performance?
 - Assuming a hit time of one CPU clock cycle, program execution will continue normally on a cache hit.
 - For cache misses, we'll assume the CPU must stall to wait for a load from main memory.
- ❑ The total number of stall cycles depends on the number of cache misses *and* the miss penalty.

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

- ❑ To include stalls due to cache misses in CPU performance equations, we have to add them to the “base” number of execution cycles.

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

Performance example

- Assume that 33% of the instructions in a program are data accesses. The I-cache hit ratio is 100%, D-cache hit ratio is 97% and the hit time is one cycle, but the miss penalty is 20 cycles.

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

- If I instructions are executed, then the number of wasted cycles will be $0.2 \times I$.

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:

$$\text{CPU time} = (1 + 0.2 \cdot 1) \times \text{Cycle time}$$

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

$$\text{CPU time} = (0.5 \cdot 1 + 0.2 \cdot 1) \times \text{Cycle time}$$

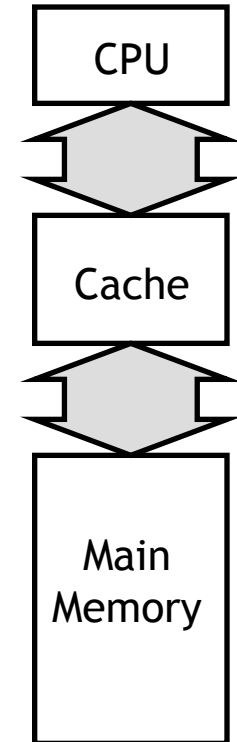
- ❑ The overall CPU time improves by just $1.2/0.7 = 1.7$ times!
- ❑ Refer back to Amdahl's Law.
 - Speeding up only part of a system has diminishing returns.

Basic main memory design

- ❑ There are some ways the main memory can be organized to reduce miss penalties and help with caching.
- ❑ For some concrete examples, 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.

$$1 + 15 + 1 = 17 \text{ clock cycles}$$

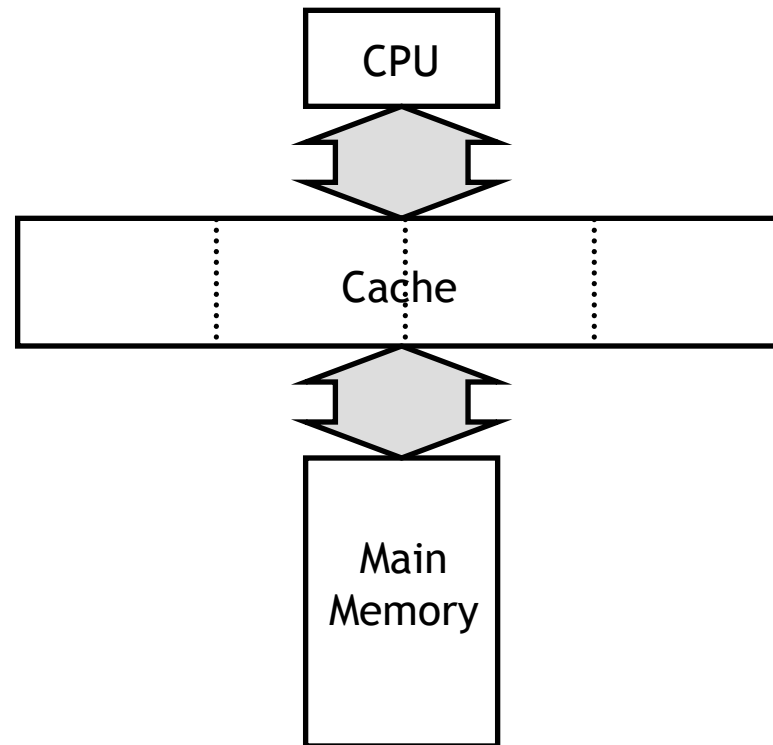
- ❑ The cache controller has to send the desired address to the RAM, wait and receive the data.



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!

$$4 \times (1 + 15 + 1) = 68 \text{ clock cycles}$$

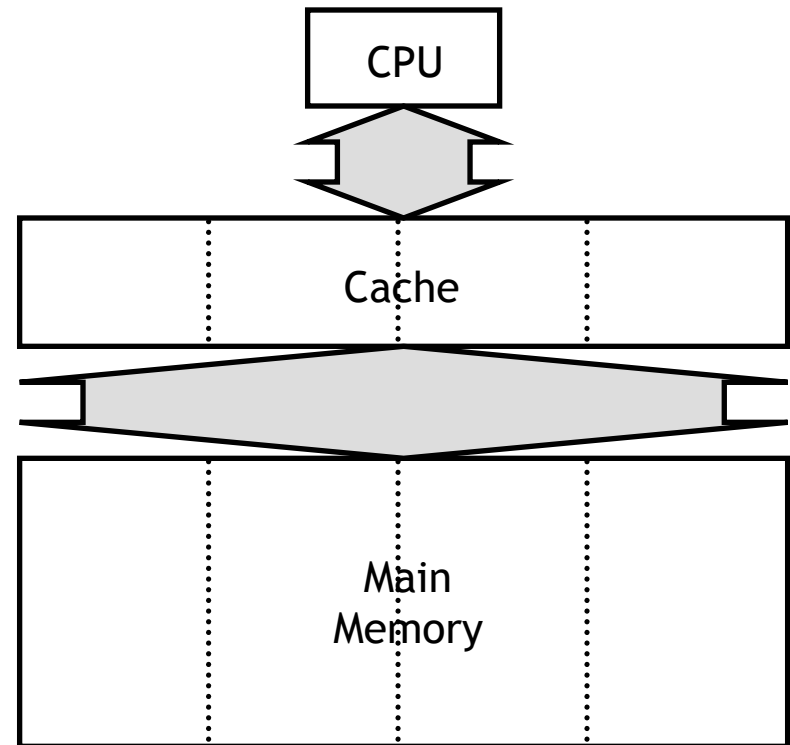


A wider memory

- ❑ 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.

$$1 + 15 + 1 = 17 \text{ cycles}$$

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

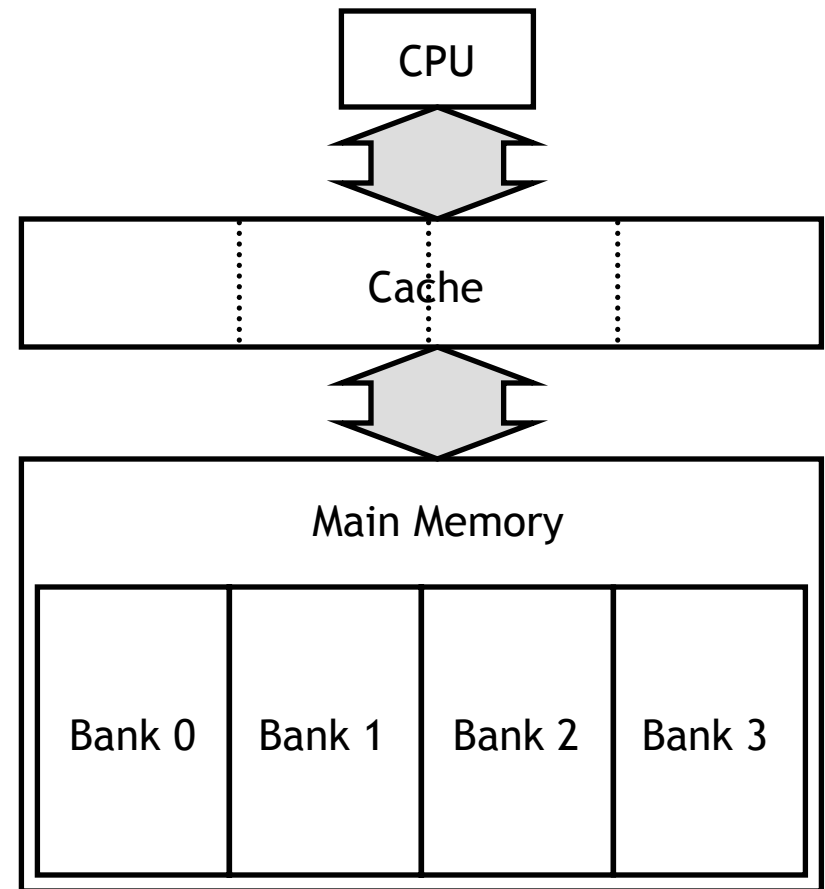


An interleaved memory

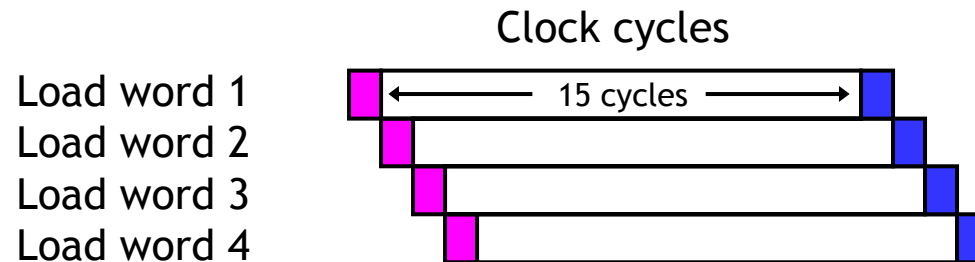
- ❑ Another approach is to **interleave** the memory, or split it into “banks” that can be accessed individually.
- ❑ The main benefit is overlapping the latencies of accessing each word.
- ❑ For example, if our main memory has four banks, each one byte wide, then we could load four bytes into a cache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

$$1 + 15 + (4 \times 1) = 20 \text{ cycles}$$

- ❑ 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.

Which is better?

- Increasing block size can improve hit rate (due to spatial locality), but transfer time increases. Which cache configuration would be better?

	Cache #1	Cache #2
Block size	32-bytes	64-bytes
Miss rate	5%	4%

- Assume both caches have single cycle hit times. Memory accesses take 15 cycles, and the memory bus is 8-bytes wide:

- i.e., an 16-byte memory access takes 18 cycles:

1 (send address) + 15 (memory access) + 2 (two 8-byte transfers)

Cache #1:

Miss Penalty = 1 + 15 + 32B/8B = 20 cycles

AMAT = 1 + (.05 * 20) = 2

Cache #2:

Miss Penalty = 1 + 15 + 64B/8B = 24 cycles

AMAT = 1 + (.04 * 24) = ~1.96

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

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**.

$$\text{AMAT} = \text{Hit time} + (\text{Miss rate} \times \text{Miss penalty})$$

$$\text{Memory stall cycles} = \text{Memory accesses} \times \text{miss rate} \times \text{miss penalty}$$

$$\text{CPU time} = (\text{CPU execution cycles} + \text{Memory stall cycles}) \times \text{Cycle time}$$

- ❑ **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.