Class quiz: Why does the miss rate go up when we keep increasing the block size?
Impact of block size

- We cannot keep increasing the block size indefinitely:
  - The larger the block size, the less the number of entries in the cache, and the more the competition between program data for these entries!
  - The larger the block size, the more time it takes to fetch this block size from memory. Increases miss penalty, and consumes more memory bandwidth!
  - Always remember: Performance depends on the latency of hits and misses, not just the number of hits and misses. Even if we reduce misses, with a large miss penalty we may end up losing in performance!
Handling cache misses

• Processor needs to stall until data is fetched from memory.

• For an instruction cache, we need to fetch data pointed to by PC-4, wait until data is in from memory, load data in the cache, validate the data and resume execution.

• Similar handling for a data cache.

• Processor can not do anything while waiting for memory
  - The memory wall!
Handling writes

• If the processor writes in the cache, then in an instant, the cache and the main memory will have different versions of the same data!

• Need to resolve inconsistency:
  - Write-through: Propagate the update to the data down to main memory, so that cache and main memory always have consistent copies of the data.
  - Write-back: Do not propagate update down to main memory, unless the updated data gets replaced from the cache. Perform the update of main memory upon replacement.
Handling writes

• Problems:
  - A write-through scheme forces the processor to wait as long as the latency of main memory on every write!
  - With 10% stores and 100-cycle memory latency, CPI will jump from 1 to 11!

• Solutions:
  - Write buffer, mini-cache for holding updated data from stores, while waiting to be committed to main memory.
  - Simple, but depends on the rate at which the processor writes data to memory (needs capacity planning).
Handling writes

• Write-back:
  - Identify whether a cache block has been modified or not, since the time it was brought in the cache
  - Can be done with a single bit, called the dirty bit, which is set upon a store to the cache block
  - If dirty cache line is replaced, then data needs to be written back to memory
  - If clean cache line is replaced, then data need not be written back to memory
  - Reduces latency for stores and saves memory bandwidth
Implications of write policies

• A write-through cache guarantees that data is always up-to-date in memory.

• A store in a write-through cache can safely overwrite the old data, regardless of whether the store is a hit or a miss.

• A store in a write-back cache needs first to know whether it is a hit or a miss and then proceed with writing the data (harder to implement, perhaps also longer latency for handling the miss).
Measuring Cache Performance

\[ CPU \text{ time} = (CPU \text{ execution clock cycles} + Memory \text{ stall clock cycles}) \times Clock \text{ cycle time} \]

Stall cycles stem from cache misses.

\[ Memory \text{ stall clock cycles} = Read \text{ stall cycles} + Write \text{ stall cycles} \]

\[ Read \text{ stall cycles} = \frac{Reads}{Program} \times Read \text{ miss rate} \times Read \text{ miss penalty} \]
Measuring Cache Performance

\[
\text{Write stall cycles} = \frac{\text{W}}{\text{P}} \times \text{Write miss rate} \times \text{Read miss penalty} + \text{Write buffer stalls}
\]

Why? Assume a write-through policy. A write which does not find a matching tag in the cache, needs to fetch the word from memory. A write which does find a matching tag in the cache, yet still needs to propagate the update down to memory, may stall if the write buffer is full.

Unfortunately, write buffer stalls depend on the timing of the writes (i.e. the workload in the program). We'll ignore them for simplicity, assuming that we have a large-enough write buffer.
Measuring cache performance

\[ \text{Memory stall cycles} = \frac{\text{Memory accesses}}{\text{Program}} \times \text{Miss rate} \times \text{Miss penalty} \]

\[ \text{Memory stall cycles} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Misses}}{\text{Instruction}} \times \text{Miss penalty} \]

\[ \times \text{Read miss penalty} \]
Example

• Assume a workload with:
  - 36% of instructions being loads and stores
  - 2% of instructions missing in the instruction cache
  - 4% of memory instructions missing in the data cache
  - Miss takes 100 cycles
  - CPI = 2

\[
\text{Instruction miss cycles} = I \times 2\% \times 100 = 2.00 \times I
\]
\[
\text{Data miss cycles} = I \times 36\% \times 4\% \times 100 = 1.44 \times I
\]
\[
\text{Memory stall cycles} = 3.44 \times I
\]
\[
\text{CPI} = 5.44
\]
Example

- What happens if we make the processor faster?
  - CPI = 1
  - Memory wall: the impact of memory latency is multiplied as we make the processor faster!

\[
\begin{align*}
\text{Instruction miss cycles} &= I \times 2\% \times 100 = 2.00 \times I \\
\text{Data miss cycles} &= I \times 36\% \times 4\% \times 100 = 1.44 \times I \\
\text{Memory stall cycles} &= 3.44 \times I \\
\text{CPI} &= 4.44
\end{align*}
\]
Example

• What happens if we increase the clock rate?
  – Miss penalty = 200 cycles

\[
\begin{align*}
\text{Instruction miss cycles} &= I \times 2\% \times 200 = 4.00 \times I \\
\text{Data miss cycles} &= I \times 36\% \times 4\% \times 200 = 2.88 \times I \\
\text{Memory stall cycles} &= 6.88 \times I \\
\text{CPI} &= 8.88
\end{align*}
\]

Greater impact of memory latency on performance!
Example

\[
\frac{\text{Performance with fast clock}}{\text{Performance with slow clock}} = \frac{\text{Execution time with slow clock}}{\text{Execution time with fast clock}}
\]

\[
= \frac{\text{IC} \times \text{CPI}_{\text{slow clock}} \times \text{Clock cycle}}{\text{IC} \times \text{CPI}_{\text{fast clock}} \times \frac{\text{Clock cycle}}{2}}
\]

\[
= \frac{5.44}{8.88 \times 0.5} = 1.23
\]

Although you doubled the clock cycle, the processor is only about 1.2 times faster, because of cache misses.
Trends

• As we lower CPI, caches have a more pronounced impact on performance.

• Main memory does not improve as fast as clock rates, therefore the impact of memory latency becomes even more pronounced.

• Don't forget hit time: If we introduce a modification to the cache to improve miss rate and increase hit time, we may end up with no performance benefit!