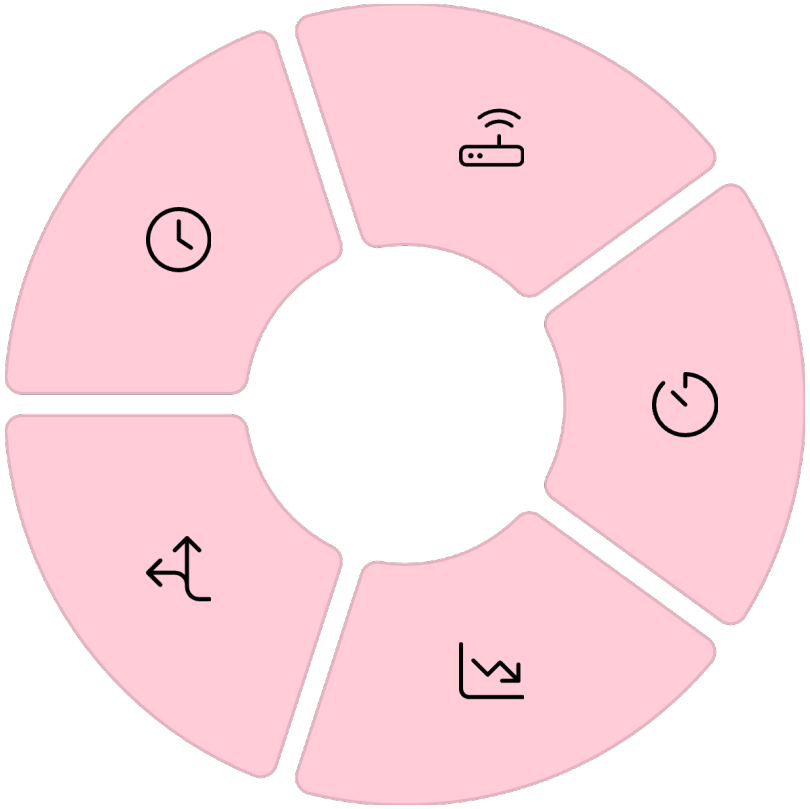
Cache Optimization Categories

Reducing Hit Time

Small and simple first-level caches and way prediction

Using Parallelism

Hardware and compiler prefetching



Increasing Cache Bandwidth

Pipelined caches, multibanked caches, and nonblocking caches

Reducing Miss Penalty

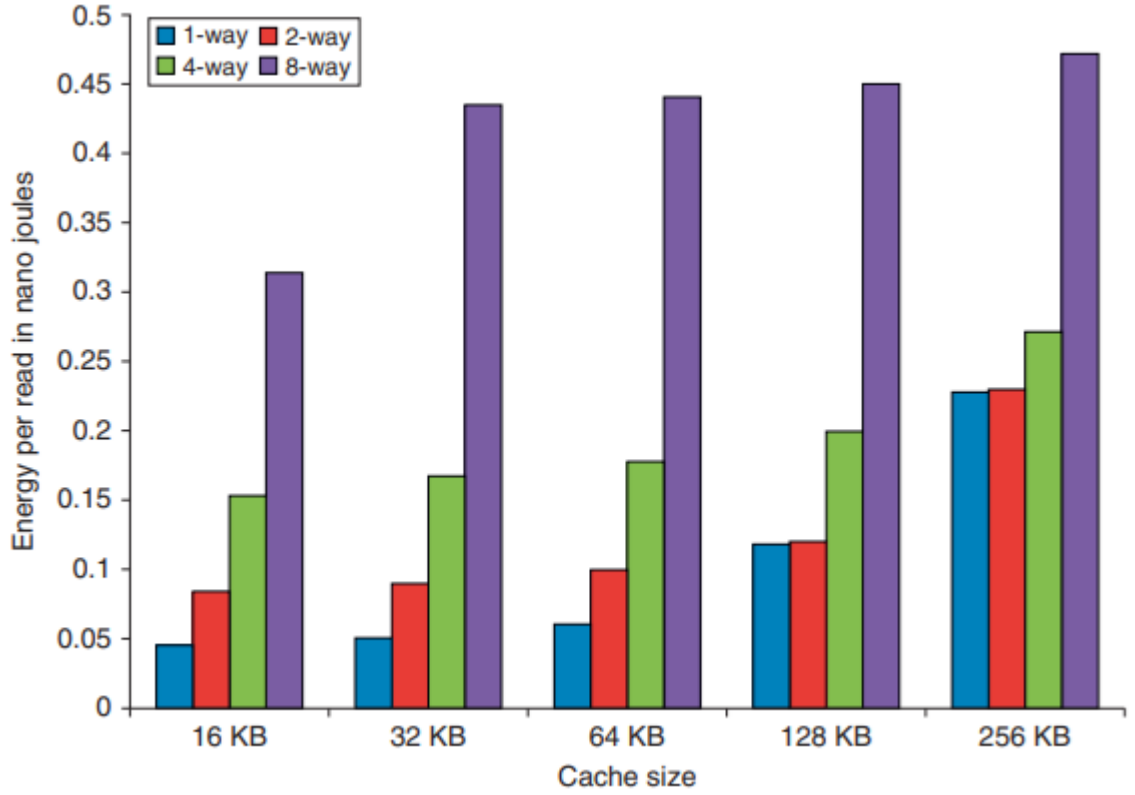
Critical word first and merging write buffers

Reducing Miss Rate

Compiler optimizations

#1: Small and Simple First-Level Caches

- Key Benefits
  - Faster hit time due to smaller size
  - Lower power consumption
  - Direct-mapped caches can overlap tag check with data transmission
- The pressure of both a fast clock cycle and power limitations encourages limited size for first-level caches.



Example

- Using the data on the right, determine whether 2-way L1 cache is faster or 4-way one.
- $Average\ memory\ access\ time = Hit\ time + Miss\ rate \times Miss\ penalty$
  - $AMAT_{2-way} = 1 + 0.038 \times 15 = 1.57$
  - $AMAT_{4-way} = 1.4 + 0.033 \times 15 = 1.895$

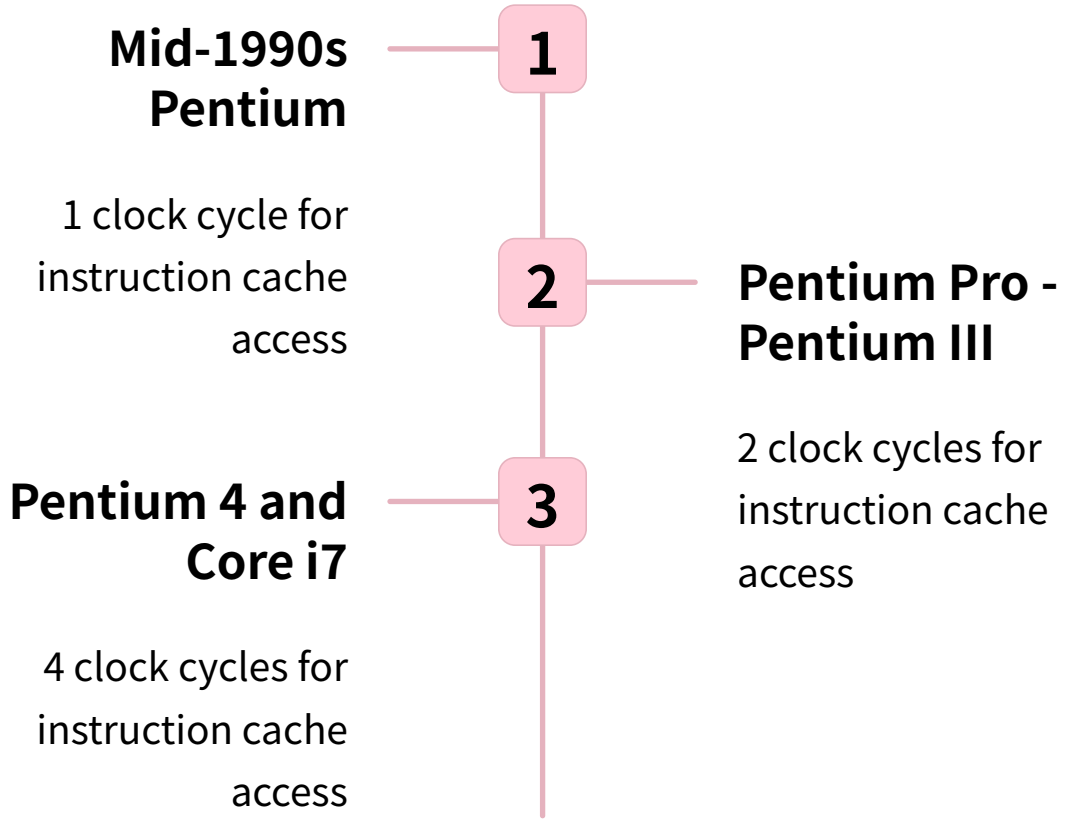
Type	Item	Value
2-way	Hit time	1
	Miss rate	0.038
4-way	Hit time	1.4
	Miss rate	0.033
Both	Miss penalty	15

#2: Way Prediction

- Way prediction keeps extra bits in the cache to predict which way, or block within the set, will be accessed next.
  - Prediction bits select which block to try on the next cache access
  - If prediction is correct: fast hit time
  - If prediction is wrong: try other block, change predictor, add one cycle latency
- Set prediction accuracy is typically >90% for two-way set associative caches and >80% for four-way set associative caches.
- First used in MIPS R10000 in mid-1990s; now used in ARM Cortex-A8.

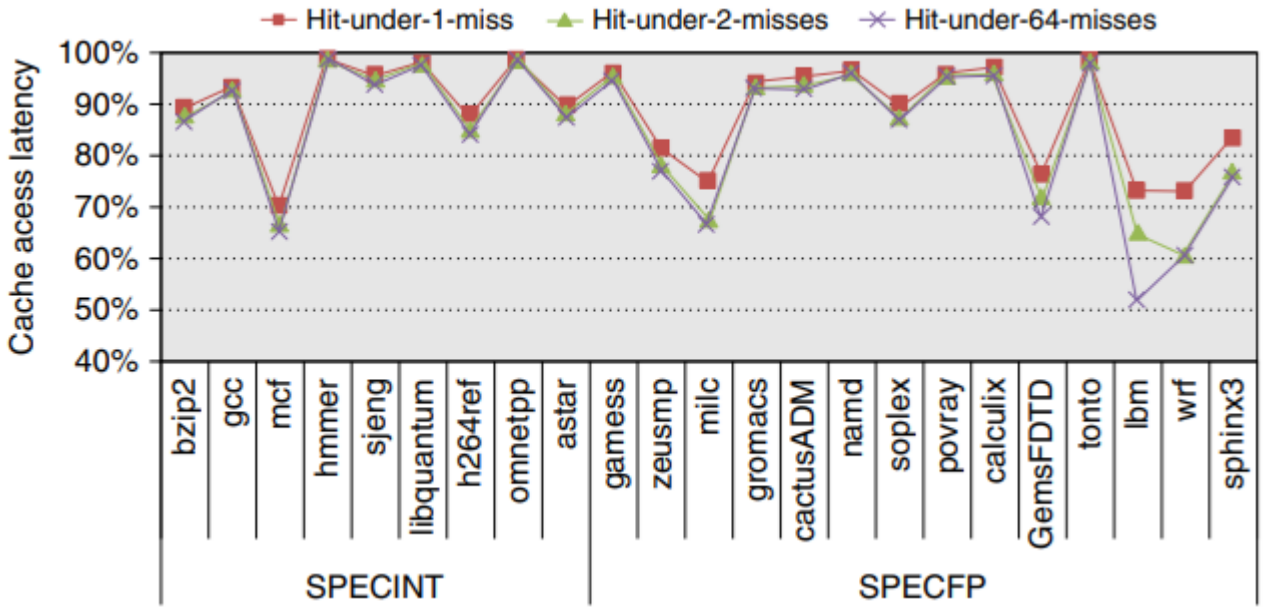
#3: Pipelined Cache Access

- Pipeline cache access so that the effective latency of a first-level cache hit can be multiple clock cycles, allowing:
  - Faster clock cycle time
  - Higher bandwidth
  - Support for higher associativity



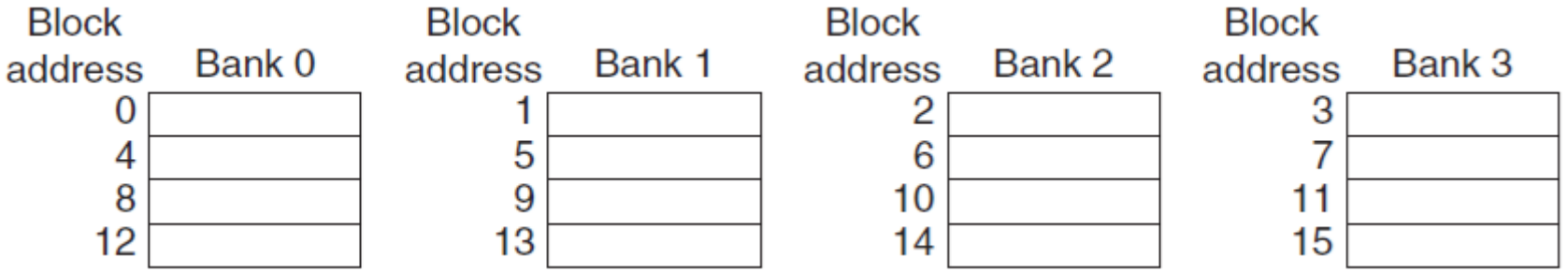
#4: Nonblocking Caches

- Key Capabilities
  - Hit under miss: Continue to supply cache hits during a miss
  - Hit under multiple miss: Handle multiple outstanding misses
  - Miss under miss: Process multiple misses in parallel
- Effectiveness depends on memory system's ability to handle multiple misses and the program's memory access patterns.



#5: Multibanked Caches

- Divide the cache into independent banks that can support simultaneous accesses, increasing bandwidth.
  - Originally used for main memory, now used in DRAM chips and caches
  - Addresses are typically spread sequentially across banks (sequential interleaving)
  - Also helps reduce power consumption
- The Arm Cortex-A8 supports one to four banks in its L2 cache; the Intel Core i7 has four banks in L1 (to support up to 2 memory accesses per clock), and the L2 has eight banks.



#6: Critical Word First and Early Restart

1

Critical Word First

Request the missed word first from memory and send it to the processor as soon as it arrives; continue filling the rest of the block while the processor resumes execution.

2

Early Restart

Fetch words in normal order, but as soon as the requested word arrives, send it to the processor and allow execution to continue while filling the rest of the block.

These techniques are most beneficial with large cache blocks, as they reduce the effective miss penalty by not waiting for the entire block to be loaded before resuming execution.

The effectiveness depends on the likelihood of needing other parts of the block soon after the critical word, due to spatial locality.

#7: Merging Write Buffer

The Problem

Write-through caches and write-back caches during block replacement need buffers to hold writes to memory.

If the write buffer is full and there's no address match, the cache (and processor) must wait for an empty entry.

The Solution: Write Merging

When new data is written to the buffer, check if the address matches an existing entry. If so, combine the new data with that entry.

- Uses memory more efficiently with multiword writes
- Reduces stalls from a full write buffer
- Used in Intel Core i7 and many other processors