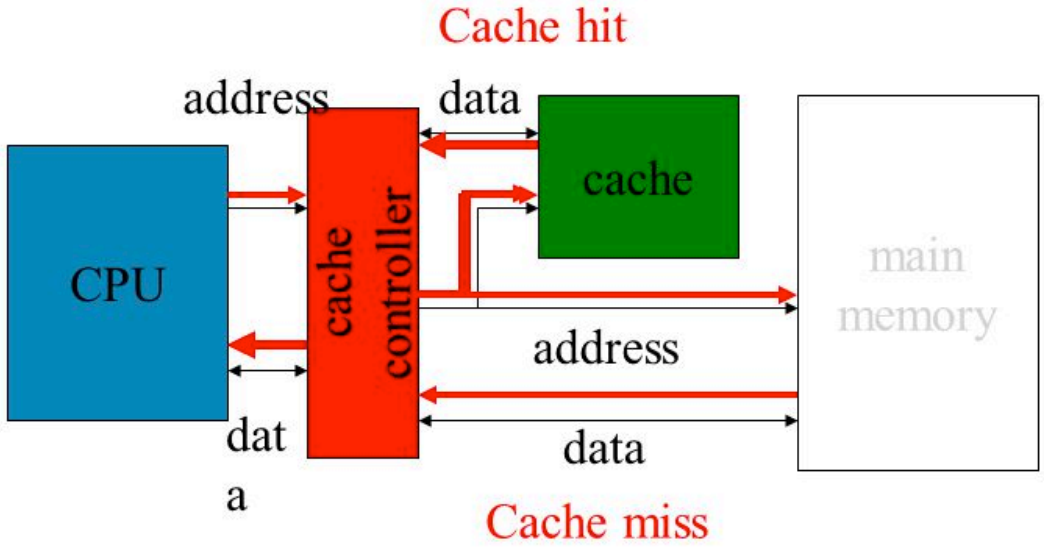
Improving Cache Performance

Average Memory Access Time = Hit Time + Miss Rate * Miss Penalty



Goals	Basic Approaches
Reducing Miss Rate	Larger block size, larger cache size and higher associativity
Reducing Miss Penalty	Multilevel caches, and higher read priority over writes
Reducing Hit Time	Avoid address translation when indexing the cache

1. Reduce Miss Rate via Larger Block Size

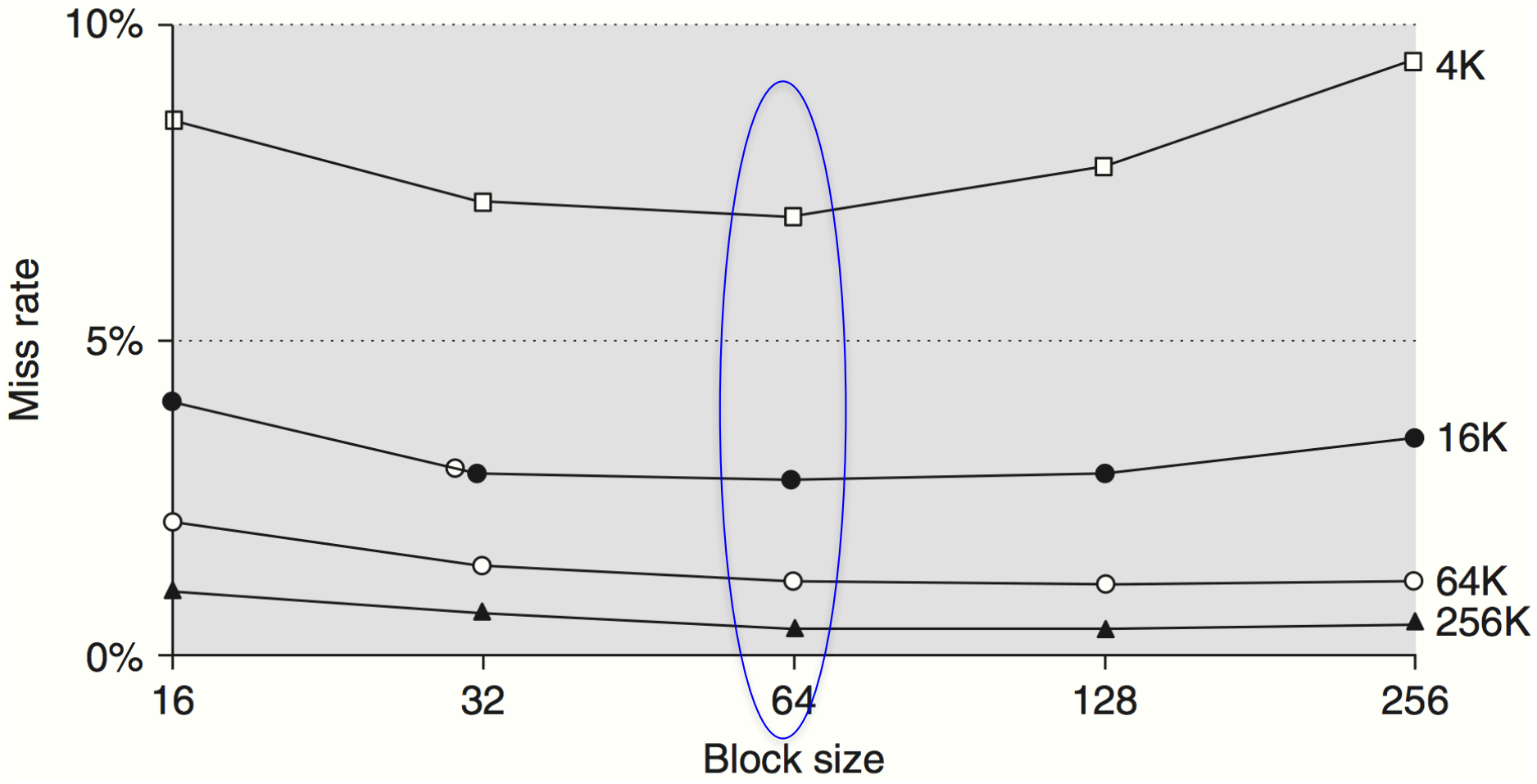


Figure B.10 Miss rate versus block size for five different-sized caches. Note that miss rate actually goes up if the block size is too large relative to the cache size. Each line represents a cache of different size. Figure B.11 shows the data used to plot these lines. Unfortunately, SPEC2000 traces would take too long if block size were included, so

Much Larger Block Size: → Increase Miss Rate

	Cache size			
Block size	4K	16K	64K	256K
16	8.57%	3.94%	2.04%	1.09%
32	7.24%	2.87%	1.35%	0.70%
64	7.00%	2.64%	1.06%	0.51%
128	7.78%	2.77%	1.02%	0.49%
256	9.51%	3.29%	1.15%	0.49%

Figure B.11 Actual miss rate versus block size for the five different-sized caches in Figure B.10. Note that for a 4 KB cache, 256-byte blocks have a higher miss rate than 32-byte blocks. In this example, the cache would have to be 256 KB in order for a 256-byte block to decrease misses.

Example: Miss Rate vs Reduce AMAT

Example Figure B.11 shows the actual miss rates plotted in Figure B.10. Assume the memory system takes 80 clock cycles of overhead and then delivers 16 bytes every 2 clock cycles. Thus, it can supply 16 bytes in 82 clock cycles, 32 bytes in 84 clock cycles, and so on. Which block size has the smallest average memory access time for each cache size in Figure B.11?

Answer Average memory access time is

$$\text{Average memory access time} = \text{Hit time} + \text{Miss rate} \times \text{Miss penalty}$$

If we assume the hit time is 1 clock cycle independent of block size, then the access time for a 16-byte block in a 4 KB cache is

$$\text{Average memory access time} = 1 + (8.57\% \times 82) = 8.027 \text{ clock cycles}$$

and for a 256-byte block in a 256 KB cache the average memory access time is

$$\text{Average memory access time} = 1 + (0.49\% \times 112) = 1.549 \text{ clock cycles}$$

Larger Block Size: → Increase Miss Penalty
Choose a Block Size Based on AMAT

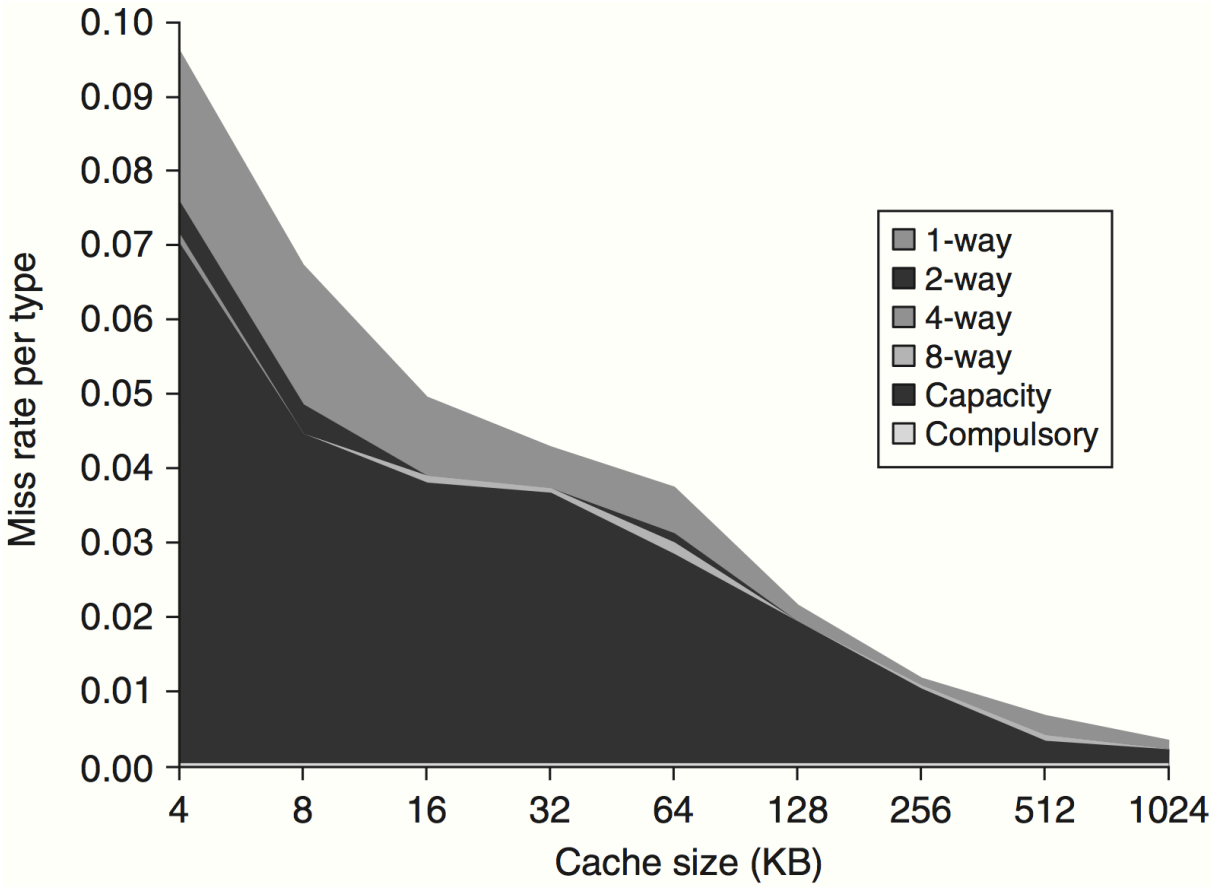
Block size	Miss penalty	Cache size 4K	16K	64K	256K
16	82	8.027	4.231	2.673	1.894
32	84	**7.082**	3.411	2.134	1.588
64	88	7.160	**3.323**	**1.933**	**1.449**
128	96	8.469	3.659	1.979	1.470
256	112	11.651	4.685	2.288	1.549

Figure B.12 Average memory access time versus block size for five different-sized caches in Figure B.10. Block sizes of 32 and 64 bytes dominate. The smallest average time per cache size is boldfaced.

$$\text{Average Memory Access Time} = \text{Hit Time} + \text{Miss Rate} * \text{Miss Penalty}$$

2. Reduce Miss Rate via Larger Cache

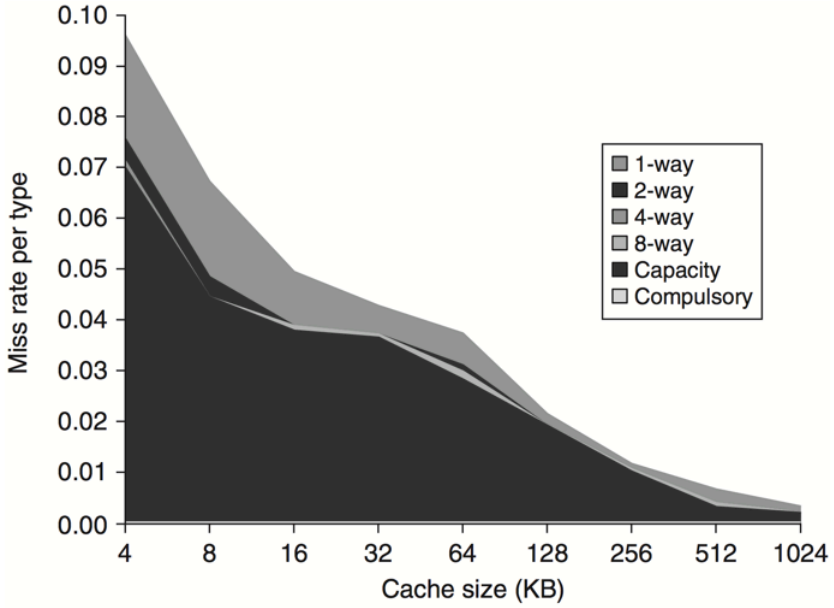
- Increasing capacity of cache reduces capacity misses
- May be longer hit time and higher cost
- Trends: Larger L2 or L3 off-chip caches



3. Reduce Miss Rate via Higher Associativity

- 2:1 Cache Rule:
  - Miss Rate DM cache size N = Miss Rate 2-way cache size N/2
- 8-way set associative is as effective as fully associative for practical purposes
- Tradeoff: higher associative cache complicates the circuit
  - May have longer clock cycle

- Beware: Execution time is the only final measure!
  - Will Clock Cycle time increase?
  - Hill [1988] suggested hit time for 2-way vs. 1-way external cache +10%, internal + 2%



Associativity ←→ AMAT

Example Assume that higher associativity would increase the clock cycle time as listed below:

$$\text{Clock cycle time}_{2\text{-way}} = 1.36 \times \text{Clock cycle time}_{1\text{-way}}$$
$$\text{Clock cycle time}_{4\text{-way}} = 1.44 \times \text{Clock cycle time}_{1\text{-way}}$$
$$\text{Clock cycle time}_{8\text{-way}} = 1.52 \times \text{Clock cycle time}_{1\text{-way}}$$

Assume that the hit time is 1 clock cycle, that the miss penalty for the direct-mapped case is 25 clock cycles to a level 2 cache (see next subsection) that never misses, and that the miss penalty need not be rounded to an integral number of clock cycles. Using Figure B.8 for miss rates, for which cache sizes are each of these three statements true?

$$\text{Average memory access time}_{8\text{-way}} < \text{Average memory access time}_{4\text{-way}}$$
$$\text{Average memory access time}_{4\text{-way}} < \text{Average memory access time}_{2\text{-way}}$$
$$\text{Average memory access time}_{2\text{-way}} < \text{Average memory access time}_{1\text{-way}}$$

	Associativity			
Cache size (KB)	**1-way**	**2-way**	**4-way**	**8-way**
4	3.44	3.25	3.22	**3.28**
8	2.69	2.58	2.55	**2.62**

Associativity ←→ AMAT: High Associativity Leads to Higher Access Time

Answer Average memory access time for each associativity is

$$\text{Average memory access time}_{8\text{-way}} = \text{Hit time}_{8\text{-way}} + \text{Miss rate}_{8\text{-way}} \times \text{Miss penalty}_{8\text{-way}}$$
$$= 1.52 + \text{Miss rate}_{8\text{-way}} \times 25$$
$$\text{Average memory access time}_{4\text{-way}} = 1.44 + \text{Miss rate}_{4\text{-way}} \times 25$$

	Associativity			
Cache size (KB)	**1-way**	**2-way**	**4-way**	**8-way**
4	3.44	3.25	3.22	**3.28**
8	2.69	2.58	2.55	**2.62**
16	2.23	**2.40**	**2.46**	**2.53**
32	2.06	**2.30**	**2.37**	**2.45**
64	1.92	**2.14**	**2.18**	**2.25**
128	1.52	**1.84**	**1.92**	**2.00**
256	1.32	**1.66**	**1.74**	**1.82**
512	1.20	**1.55**	**1.59**	**1.66**

Figure B.13 Average memory access time using miss rates in Figure B.8 for parameters in the example. Boldface type means that this time is higher than the number to the left, that is, higher associativity *increases* average memory access time.

4. Reduce Miss Penalty via Multilevel Caches

- Approaches
  - Make the cache faster to keep pace with the speed of CPUs
  - Make the cache larger to overcome the widening gap
- **L1: fast hits, L2: fewer misses**
- L2 Equations

$$\text{Average memory access time} = \text{Hit time}_{L1} + \text{Miss rate}_{L1} \times \text{Miss penalty}_{L1}$$

and

$$\text{Miss penalty}_{L1} = \text{Hit time}_{L2} + \text{Miss rate}_{L2} \times \text{Miss penalty}_{L2}$$

so

$$\text{Average memory access time} = \text{Hit time}_{L1} + \text{Miss rate}_{L1} \times (\text{Hit time}_{L2} + \text{Miss rate}_{L2} \times \text{Miss penalty}_{L2})$$

- $\text{Hit Time}_{L1} << \text{Hit Time}_{L2} << \ldots << \text{Hit Time}_{Mem}$
- $\text{Miss Rate}_{L1} < \text{Miss Rate}_{L2} < \ldots$

Miss Rate in Multilevel Caches

- **Local miss rate**— misses in this cache divided by the total number of memory accesses to this cache (Miss rateL1 , Miss rateL2)
  - L1 cache skims the cream of the memory accesses
- **Global miss rate**—misses in this cache divided by the total number of memory accesses generated by the CPU (Miss rateL1, Miss RateL1 x Miss RateL2)
  - Indicate what fraction of the memory accesses that leave the CPU go all the way to memory

Miss Rates in Multilevel Caches

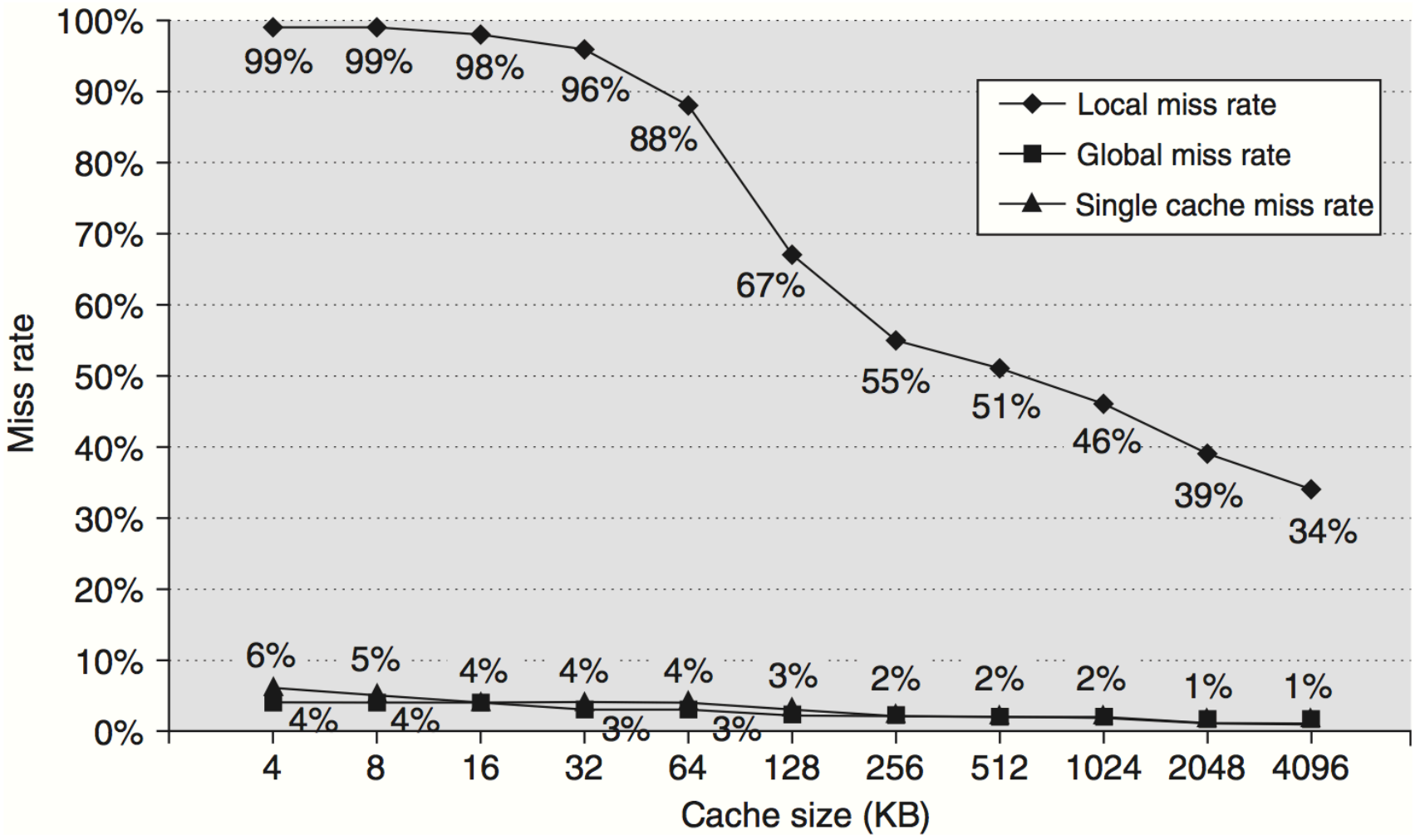


Figure B.14 Miss rates versus cache size for multilevel caches. Second-level caches *smaller* than the sum of the two 64 KB first-level caches make little sense, as reflected in the high miss rates. After 256 KB the single cache is within 10% of the global miss rates. The miss rate of a single-level cache versus size is plotted against the local miss rate and

Multilevel Caches: Design of L2

- Size
  - Since everything in L1 cache is likely to be in L2 cache, L2 cache should be much bigger than L1
- Whether data in L1 is in L2
  - novice approach: design L1 and L2 independently
  - multilevel inclusion: L1 data are always present in L2
    - Advantage: easy for consistency between I/O and cache (checking L2 only)
    - Drawback: L2 must invalidate all L1 blocks that map onto the 2nd-level block to be replaced => slightly higher 1st-level miss rate
      - i.e. Intel Pentium 4: 64-byte block in L1 and 128-byte in L2
  - multilevel exclusion: L1 data is never found in L2
    - A cache miss in L1 results in a swap of blocks between L1 and L2
    - Advantage: prevent wasting space in L2
      - i.e. AMD Athlon: 64 KB L1 and 256 KB L2

Multilevel Caches: Example

Example Suppose that in 1000 memory references there are 40 misses in the first-level cache and 20 misses in the second-level cache. What are the various miss rates? Assume the miss penalty from the L2 cache to memory is 200 clock cycles, the hit time of the L2 cache is 10 clock cycles, the hit time of L1 is 1 clock cycle, and there are 1.5 memory references per instruction. What is the average memory access time and average stall cycles per instruction? Ignore the impact of writes.

Answer The miss rate (either local or global) for the first-level cache is 40/1000 or 4%. The local miss rate for the second-level cache is 20/40 or 50%. The global miss rate of the second-level cache is 20/1000 or 2%. Then

$$\begin{aligned}\text{Average memory access time} &= \text{Hit time}_{L1} + \text{Miss rate}_{L1} \times (\text{Hit time}_{L2} + \text{Miss rate}_{L2} \times \text{Miss penalty}_{L2}) \\ &= 1 + 4\% \times (10 + 50\% \times 200) = 1 + 4\% \times 110 = 5.4 \text{ clock cycles}\end{aligned}$$

Multilevel Caches: Example

To see how many misses we get per instruction, we divide 1000 memory references by 1.5 memory references per instruction, which yields 667 instructions. Thus, we need to multiply the misses by 1.5 to get the number of misses per 1000 instructions. We have 40 × 1.5 or 60 L1 misses, and 20 × 1.5 or 30 L2 misses, per 1000 instructions. For average memory stalls per instruction, assuming the misses are distributed uniformly between instructions and data:

$$\begin{aligned}\text{Average memory stalls per instruction} &= \text{Misses per instruction}_{L1} \times \text{Hit time}_{L2} + \text{Misses per instruction}_{L2} \\ &\quad \times \text{Miss penalty}_{L2} \\ &= (60/1000) \times 10 + (30/1000) \times 200 \\ &= 0.060 \times 10 + 0.030 \times 200 = 6.6 \text{ clock cycles}\end{aligned}$$

If we subtract the L1 hit time from the average memory access time (AMAT) and then multiply by the average number of memory references per instruction, we get the same average memory stalls per instruction:

$$(5.4 - 1.0) \times 1.5 = 4.4 \times 1.5 = 6.6 \text{ clock cycles}$$

As this example shows, there may be less confusion with multilevel caches when calculating using misses per instruction versus miss rates.

5. Reduce Miss Penalty by Giving Priority to Read Misses over Writes

- Serve reads before writes have been completed
- Write through with write buffers

```
SW    R3, 512(R0)     ; M[512] <- R3      (cache index 0)
LW    R1, 1024(R0)    ; R1 <- M[1024]     (cache index 0)
LW    R2, 512(R0)     ; R2 <- M[512]      (cache index 0)
```

  Problem: write through with write buffers offer RAW conflicts with main memory reads on cache misses
  - If simply wait for write buffer to empty, might increase read miss penalty (old MIPS 1000 by 50%)
  - Check write buffer contents before read; if no conflicts, let the memory access continue
- Write Back

  Suppose a read miss will replace a dirty block
  - Normal: Write dirty block to memory, and then do the read
  - Instead: Copy the dirty block to a write buffer, do the read, and then do the write
  - CPU stall less since restarts as soon as do read

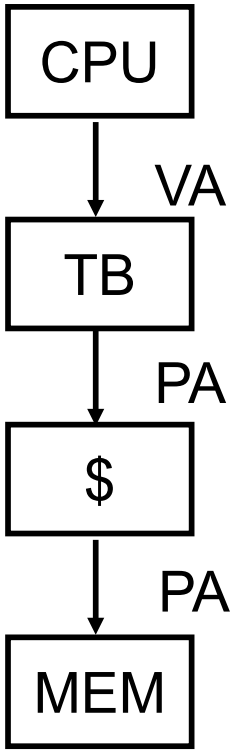
6. Reduce Hit Time by Avoiding Address Translation during Indexing of the Cache

- Importance of cache hit time
  - Average Memory Access Time = **Hit Time** + Miss Rate * Miss Penalty
  - More importantly, cache access time **limits the clock cycle rate** in many processors today!
- Fast hit time:
  - Quickly and efficiently find out if data is in the cache, and
  - if it is, get that data out of the cache
- Four techniques:
  1. Small and simple caches
  2. Avoiding address translation during indexing of the cache
  3. Pipelined cache acces
  4. Trace caches

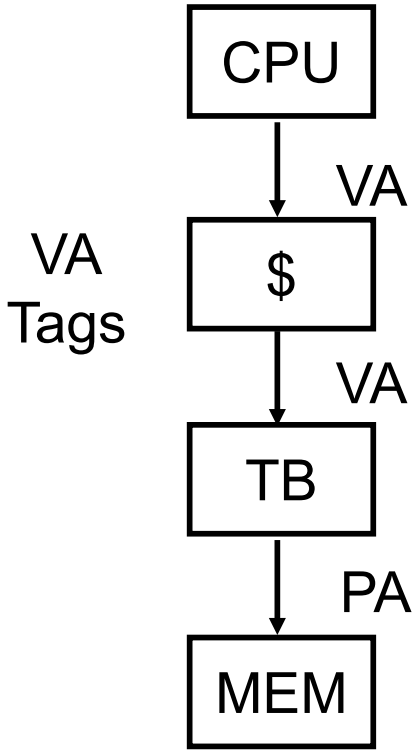
Avoiding address translation during cache indexing

- Two tasks: indexing the cache and comparing addresses
- virtually vs. physically addressed cache
  - –virtual cache: use virtual address (VA) for the cache
  - –physical cache: use physical address (PA) after translating virtual address
- Challenges to virtual cache
  1. **Protection**: page-level protection (RW/RO/Invalid) must be checked
     - –It's checked as part of the virtual to physical address translation
     - –solution: an addition field to copy the protection information from TLB and check it on every access to the cache
  2. **context switching**: same VA of different processes refer to different PA, requiring the cache to be flushed
     - –solution: increase width of cache address tag with process-identifier tag (PID)
  3. **Synonyms or aliases**: two different VA for the same PA
     - –inconsistency problem: two copies of the same data in a virtual cache
     - –hardware ***antialiasing*** solution: guarantee every cache block a unique PA
       - –Alpha 21264: check all possible locations. If one is found, it is invalidated
     - –software ***page-coloring*** solution: forcing aliases to share some address bits
       - –Sun's Solaris: all aliases must be identical in last 18 bits => no duplicate PA
  4. **I/O**: typically use PA, so need to interact with cache (see Section 5.12)

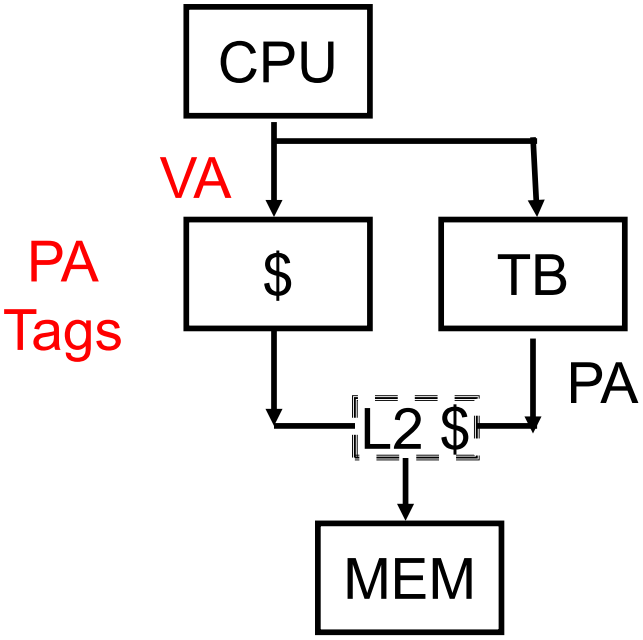
Virtually indexed, physically tagged cache



Conventional Organization



Virtually Addressed Cache
Translate only on miss
Synonym Problem



Overlap cache access with VA translation: requires $ index to remain invariant across translation

Summary of the 6 Basic Cache Optimization Techniques (Textbook B.3)

Technique	Hit time	Miss penalty	Miss rate	Hardware complexity	Comment
Larger block size		–	+	0	Trivial; Pentium 4 L2 uses 128 bytes
Larger cache size	–		+	1	Widely used, especially for L2 caches
Higher associativity	–		+	1	Widely used
Multilevel caches		+		2	Costly hardware; harder if L1 block size ≠ L2 block size; widely used
Read priority over writes		+		1	Widely used
Avoiding address translation during cache indexing	+			1	Widely used

Figure B.18 Summary of basic cache optimizations showing impact on cache performance and complexity for the techniques in this appendix. Generally a technique helps only one factor. + means that the technique improves the factor, – means it hurts that factor, and blank means it has no impact. The complexity measure is subjective, with 0 being the easiest and 3 being a challenge.