9.5 Timekeeping Techniques for Predicting and Optimizing Memory Behavior

Zhigang Hu[1], Stefanos Kaxiras[2], Margaret Martonosi[3]

[1]IBM T.J. Watson Research Center, Yorktown Heights, NY
[2]Agere Systems, Allentown, PA
[3]Princeton University, Princeton, NJ

Computer architects have long exploited application memory referencing characteristics to optimize memory performance. While most prior work has looked at only time-independent aspects of memory behavior, such as event ordering and interleaving, recent work introduced timekeeping metrics to improve memory performance and power dissipation [1]. The performance and power results for timekeeping structures are examined and implementation options are discussed.

Much like natural lifetimes in the real world, memory lifetimes show generational patterns [1, 4]. Each *cache line generation* (Fig. 9.5.1) begins with a cache miss that brings new data to this memory hierarchy level. Generations end when data leaves the cache because a miss to other data causes eviction. Cache line generations have two parts: *live time* starts from the beginning of a generation and ends with the last successful use before eviction. *Dead time* lies between last use and when the data is actually evicted. *Access interval* is the duration between successive accesses to the same cache line within a generation's live time. Tracking time intervals at run time is useful because they are strongly predictive of future program behavior. Figure 9.5.2 shows the distributions of access intervals and dead times for the SPEC2000 benchmarks. Access intervals are quite short, while both short and long dead times are common. Such statistical observations form the basis of timekeeping predictor mechanisms.

Example 1: Cache Decay These techniques use timekeeping metrics to create simple mechanisms for leakage energy control [2]. Observe that cache lines often spend thousands of cycles in their dead times, consuming leakage energy while not contributing to performance. If one predicts a cache line is currently in its dead time, the line can be "turned off" to save leakage energy. No extra misses occur, because the cache line is dead and the next access will be a miss anyway. Cache Decay's timekeeping-based dead-block predictor harnesses the fact that typical access intervals are much shorter than dead times. Prediction of a cache line as dead occurs when its idle time exceeds a pre-set threshold. From simulation results for an 8000-cycle threshold, cache decay can reduce cache leakage energy by 4X, with little impact on miss rate or performance. Figure 9.5.3 depicts a cache decay implementation. When a coarse-grained cache line timer receives a tick while in the uppermost "10" state, it switches off the gate transistor [5] so (tag and data) cells are disconnected from VSS and the leakage path to the ground is cut off. (Cache lines with dirty data are written back before deactivation.) Valid bits are always fully-powered to correctly track cache line validity.

Example 2: Timekeeping for Conflict Miss Identification Shown here is a use for timekeeping predictions of generations with short dead times. From simulation data, generations with short dead times are likely to occur due to pre-mature evictions from mapping conflicts. If short dead times are predictive of conflict misses, this prediction leads to a victim cache filter. The filter ensures that victim cache entries are mainly used to store cache lines likely to be reused soon. Results show the effectiveness of filtering victim cache entries to store only items with dead times <1024 cycles. This approach reduces victim cache traffic by 87% while also improving overall system performance. As shown in Fig. 9.5.4, counters gauge the time-since-last-reference of each victim; a comparator allows only victims with dead times smaller than a threshold to enter the victim cache.

Example 3: Timekeeping Prefetch This third timekeeping structure predicts both what and when to prefetch. Revisiting Fig. 9.5.1, a new cache line can be safely brought in when the current cache line is dead, at the end of its live time. The current live time cannot be exactly known beforehand, but can be predicted from the live time of the same line's previous generation. If, in a cache set in the past, line A was followed by B, then C, and the live time of B was lt(B), then the next time A is followed by B, line C is predicted as the next address and lt(B) as the live time of B. A prefetch of C is scheduled about lt(B) after B's appearance. Comparing a "timekeeping" prefetcher using an 8KB correlation table to an 8MB Dead-block Correlating Prefetcher (DBCP) [3], non-timekeeping prior work, the timekeeping prefetcher outperforms DBCP with an average of 11% performance improvement over baseline [1]. Figure 9.5.5 shows the timekeeping prefetcher implementation.

Implementation Options Tracking time intervals at run time is a common requirement for all timekeeping techniques. Only fairly coarse-grained time estimates are needed. One can get these via hierarchical counters (Fig. 9.5.6). For example, in cache decay, each cache line is augmented with a local counter that is 2-5 bits wide. All local counters are triggered by a global cycle counter, shared by the whole cache. The size of a local counter is small: ~1% of a cache line. Counter ticks are infrequent, so dynamic power dissipation is also quite small. To further reduce power dissipation, counter states are gray-coded. Local timekeeping counters gauge time intervals dynamically at the needed granularity. A cache line's idle time is obtained by resetting its local counter upon each access (hit or miss) and increasing the local counter with each global tick. To avoid switching too many counters simultaneously, the global counter tick is cascaded; each local counter receives the global counter tick from its previous neighbor and passes it on to the next neighbor. In Cache Decay, this staggered design also spreads out dirty-line writebacks, avoiding large writeback bursts on a global tick. Timekeeping techniques need not impact processor cycle time. At 2-5 bits of precision, the counters are narrow, so updates happen quickly. Updating local counters is not on a cache access critical path; it is done in parallel with tag comparison or data output. In extreme cases, timekeeping bookkeeping is pipelined across several processor cycles to avoid being on a critical path. Overall, how to improve memory performance by exploiting typical statistics of memory referencing events is presented. Since simple counters can track relevant time intervals, effective and power-efficient timekeeping-based hardware can improve processor power and performance.

References

[1] Z. Hu et al. Timekeeping in the Memory System: Predicting and Optimizing Memory Behavior. Proc. *29th Intl. Symp. on Computer Arch.*, 2002.

[2] S. Kaxiras et al. Cache Decay: Exploiting Generational Behavior to Reduce Cache Leakage Power. Proc. *28th Intl. Symp. on Computer Arch.,* 2001.

[3] A.-C. Lai et al. Dead-Block Prediction and Dead-Block Correlating Prefetchers. Proc. *28th Intl. Symp. on Computer Arch.*, 2001.

[4] D. A. Wood et al. A Model for Estimating Trace-Sample Miss Ratios. *1991 ACM Sigmetrics Conference on Measurement and Modeling of Computer Systems,* 1991.

[5] S.-H. Yang et al. An integrated circuit/architecture approach to reducing leakage in deep-submicron high-performance I-Caches. Proc. *7th Int'l Symp. on High-Perf. Computer Arch*, 2001.

 0-7803-7707-9/03/$17.00

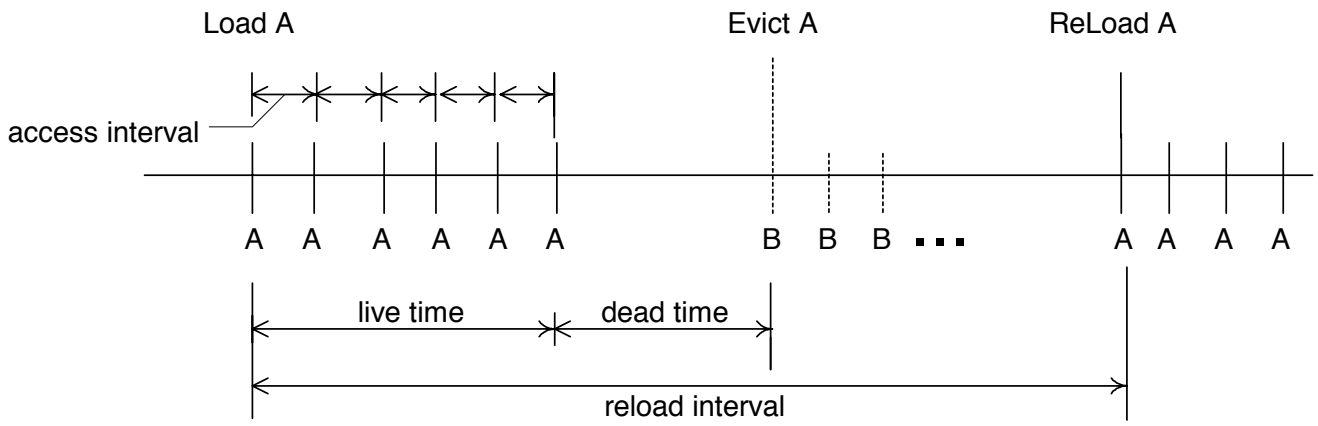

Figure 9.5.1: Timeline depicting a generation of the cache line with A resident, followed by A's eviction to begin a generation with B resident. Eventually, A is referenced to begin yet another generation.

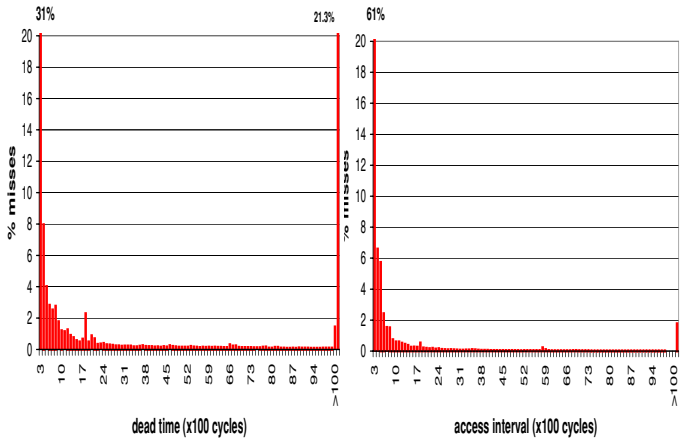

Figure 9.5.2: Distribution of access intervals (top) and dead times (bottom) for the SPEC2000 benchmark suite.

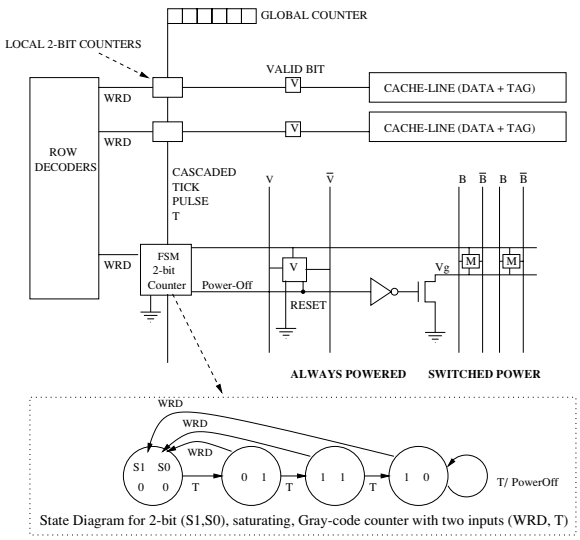

Figure 9.5.3: Cache decay implementation.

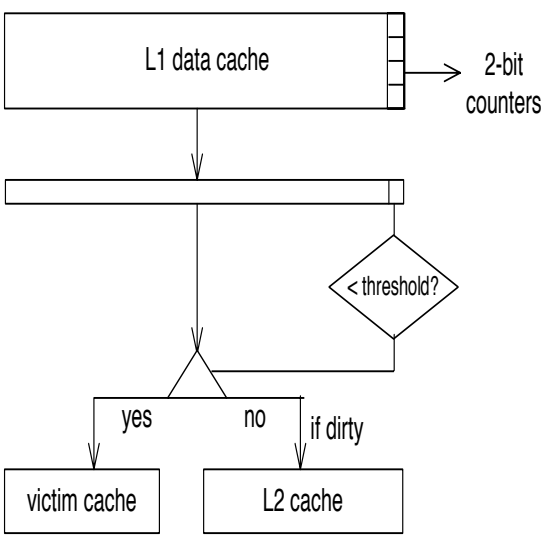

Figure 9.5.4: Timekeeping victim cache filter implementation.

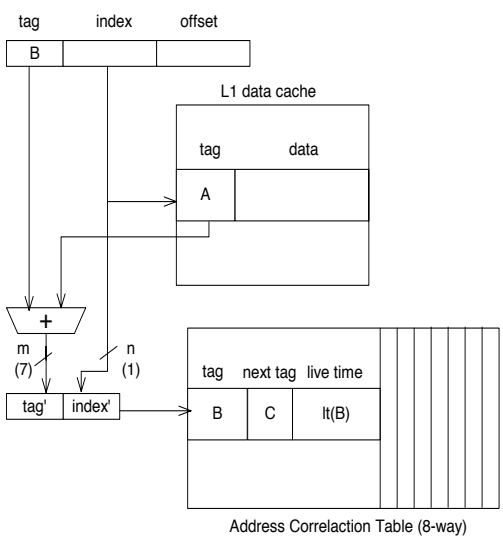

Figure 9.5.5: Timekeeping prefetcher implementation.

0-7803-7707-9/03/$17.00

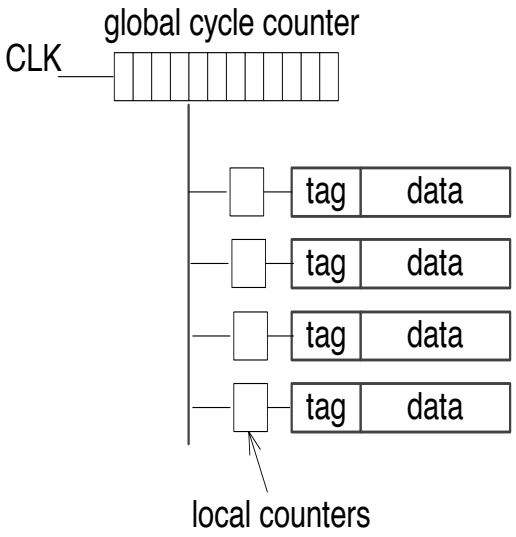

Figure 9.5.6: Hierarchical counter gauge time intervals per cache line.

 0-7803-7707-9/03/$17.00

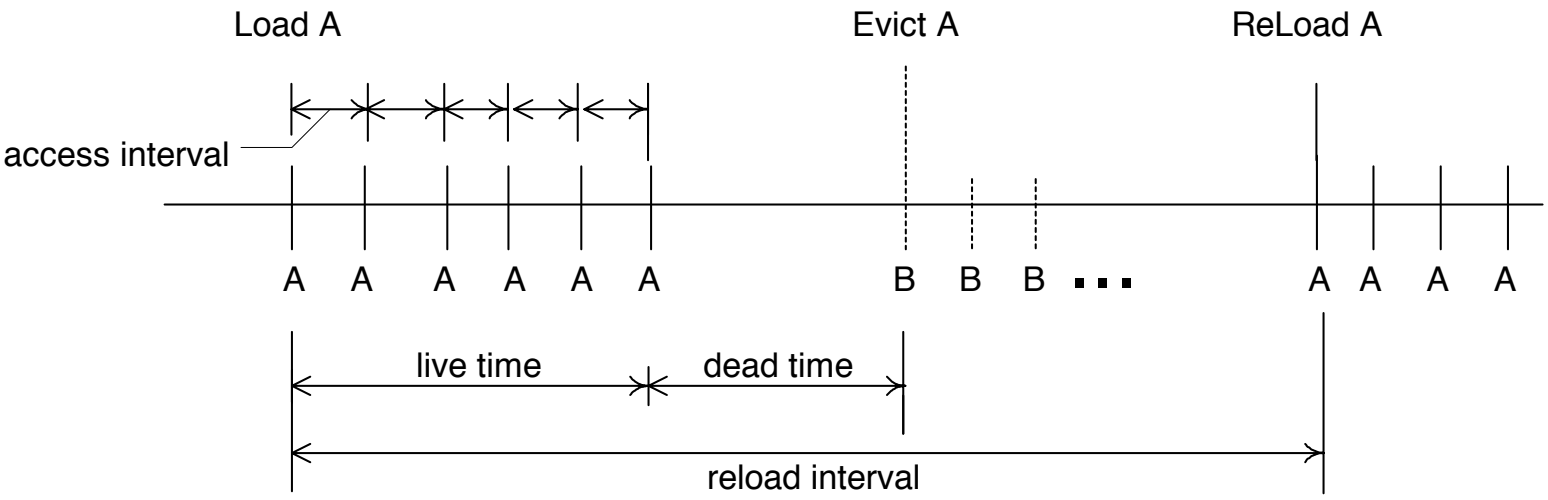

Figure 9.5.1: Timeline depicting a generation of the cache line with A resident, followed by A's eviction to begin a generation with B resident. Eventually, A is referenced to begin yet another generation.

0-7803-7707-9/03/$17.00

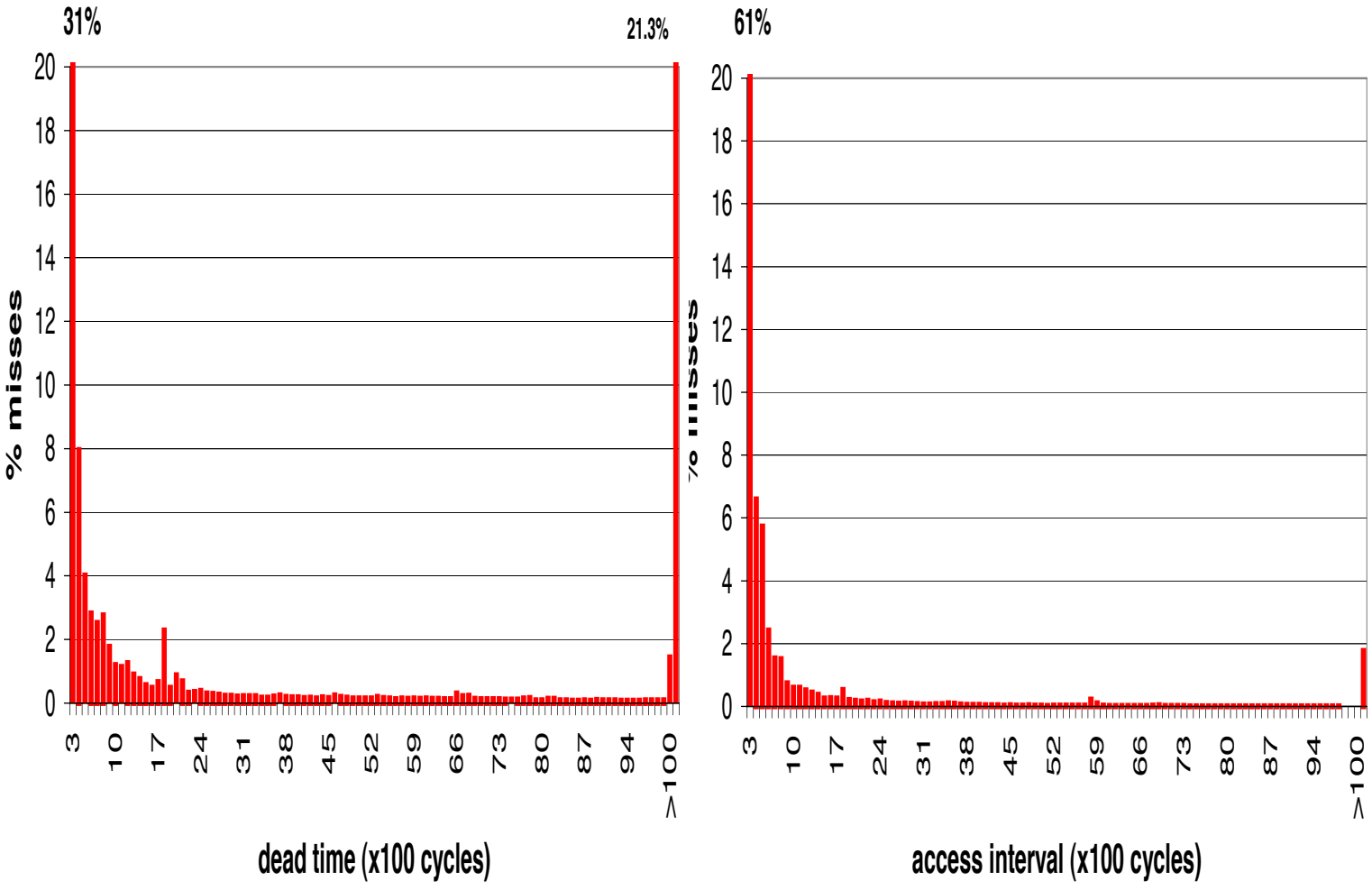

Figure 9.5.2: Distribution of access intervals (top) and dead times (bottom) for the SPEC2000 benchmark suite.

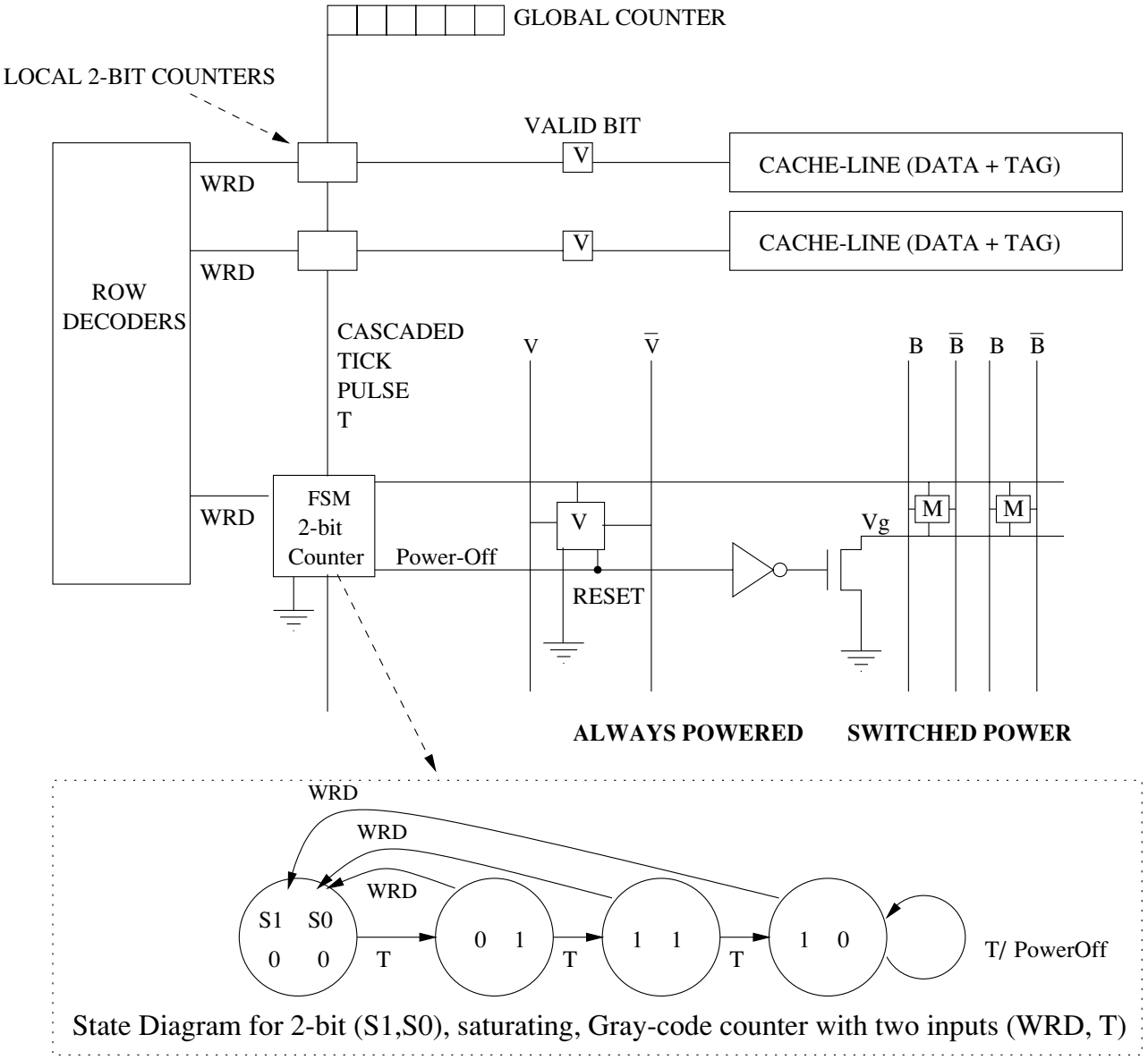

Figure 9.5.3: Cache decay implementation.

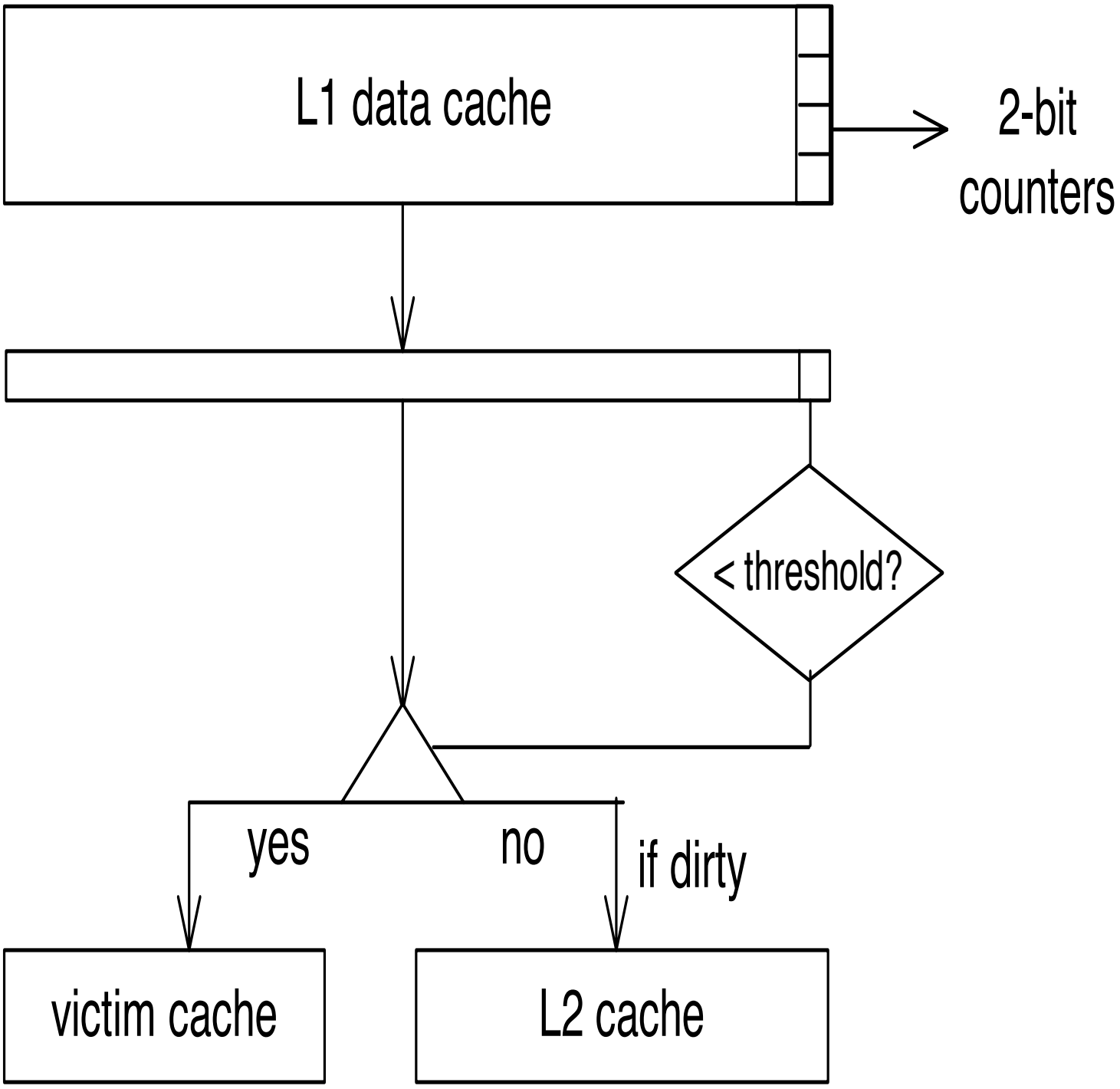

Figure 9.5.4: Timekeeping victim cache filter implementation.

0-7803-7707-9/03/$17.00

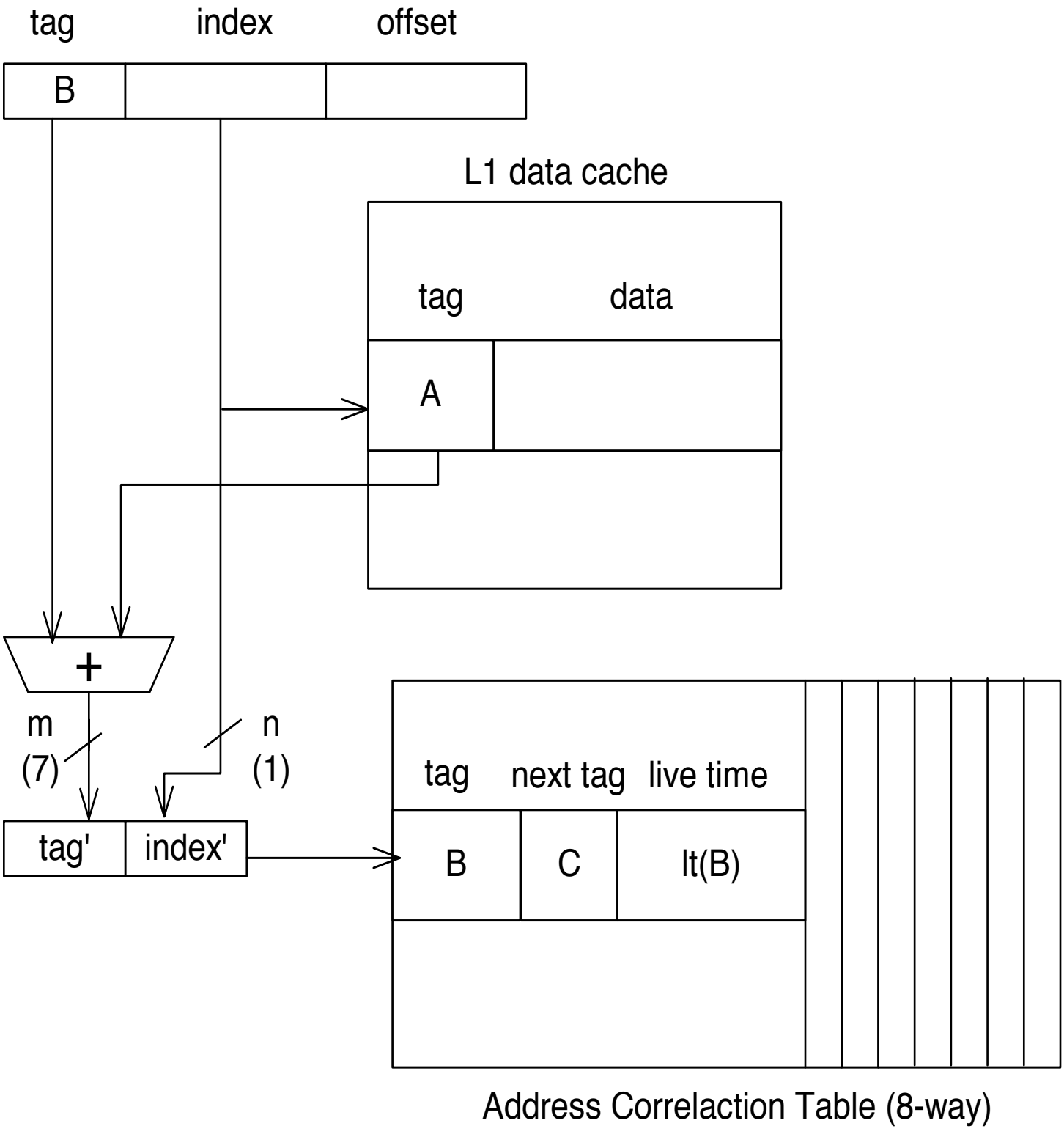

Figure 9.5.5: Timekeeping prefetcher implementation.

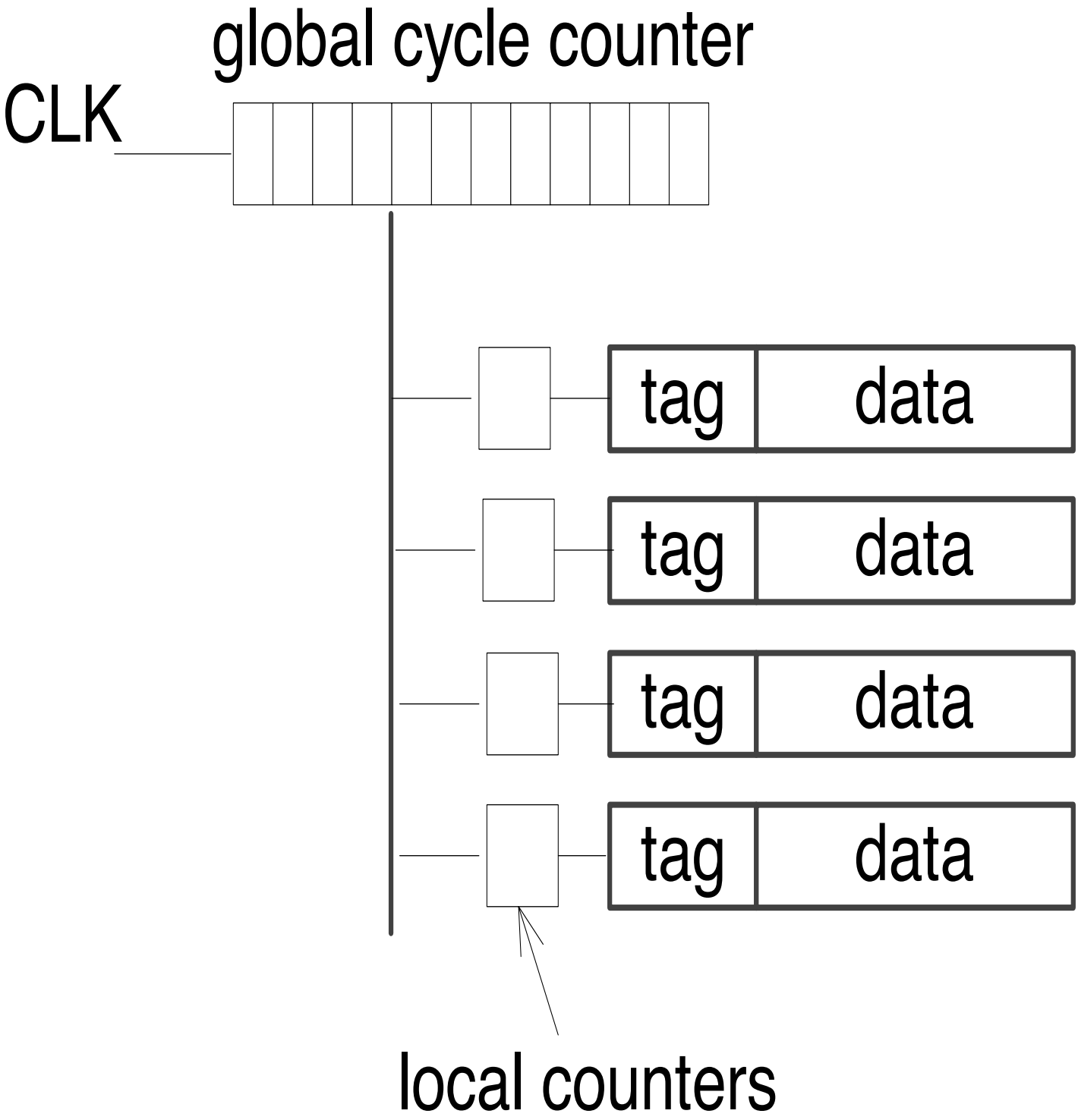

Figure 9.5.6: Hierarchical counter gauge time intervals per cache line.

0-7803-7707-9/03/$17.00