

Technical University of Denmark



Software Managed Cache for Parallel Systems

Schleuniger, Pascal; Karlsson, Sven

Publication date:
2012

Document Version
Publisher's PDF, also known as Version of record

[Link back to DTU Orbit](#)

Citation (APA):
Schleuniger, P., & Karlsson, S. (2012). Software Managed Cache for Parallel Systems. Poster session presented at 7th International Conference on High-Performance and Embedded Architectures and Compilers , Paris, France.

DTU Library

Technical Information Center of Denmark

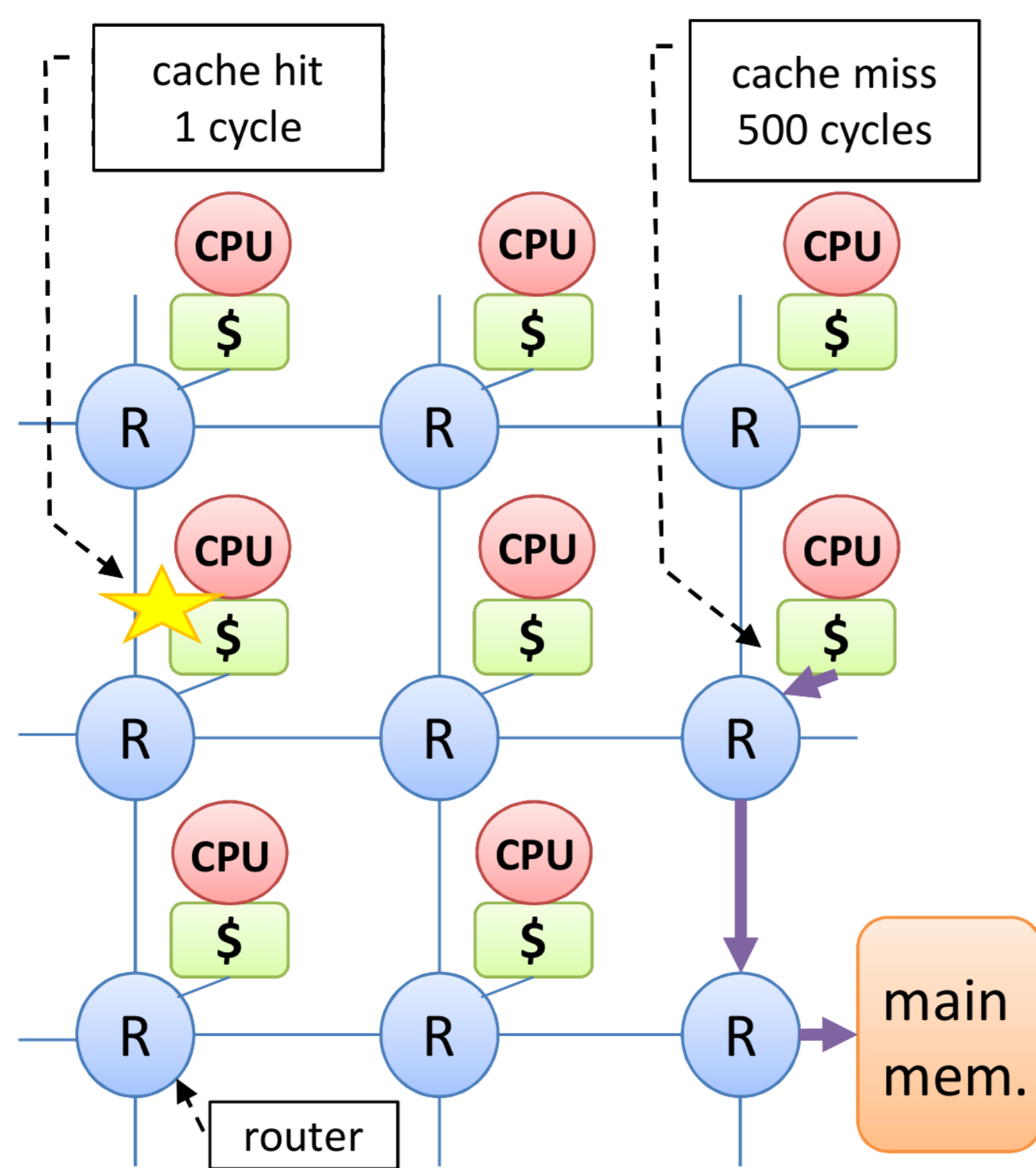
General rights

Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal

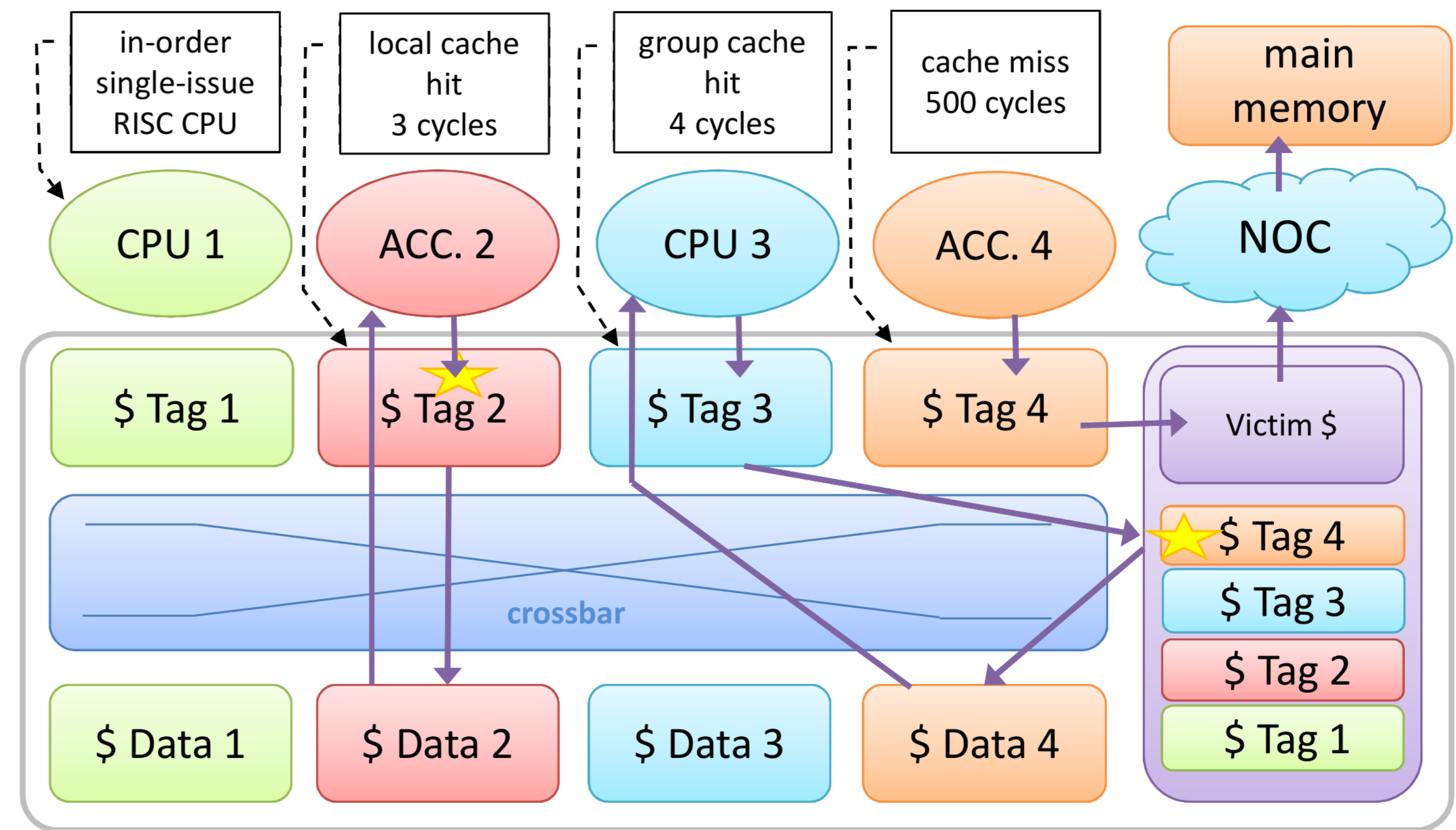
If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

Current Systems



Homogeneous Multicore

Envisioned Future Systems



Heterogeneous Multicore

Motivation

- ▶ Homogeneous multicores tend to have small private caches
- ▶ We envision heterogeneous multicores with a shared cache
 - ▶ can benefit from data stored in neighboring cores
 - ▶ cheap memory consistency within a group of processing elements
 - ▶ data traffic on the global interconnect is reduced
- ▶ We argue for a software managed cache!

Contributions

- ▶ Software managed multi-banked first level data cache
- ▶ Application aware software controlled replacement strategies

Implementation

- ▶ Use a hardware efficient and energy efficient 4-way set associative cache
- ▶ Cache hit check is done in the following way
 - ▶ tags are checked in sequence, one tag per cycle
 - ▶ a hashing function is used to predict what tag to check first
 - ▶ we end when we find a hit or there are no more tags
 - ▶ this leads to increased associativity at a low power consumption
 - ▶ however, unless the first tag checked is a hit, the cache hit time is increased
- ▶ Use hardware and software to implement replacement policy
 - ▶ on cache misses, CPUs can execute a cache replacement algorithm
 - ▶ balancing cache by relocating cache lines
 - ▶ replacement policy may change dynamically
 - ▶ try to avoid the software replacement algorithm when the expected memory latency is low
 - ▶ use a simple algorithm implemented in hardware in cases where the software replacement algorithm is too slow
- ▶ Specific memory regions can be labeled
 - ▶ often used variables may have a higher priority
 - ▶ specific memory regions may have preferred locations in the cache
 - ▶ specific memory regions may be locked in cache
- ▶ Additional costs: duplication of cache tags

Replacement Policy

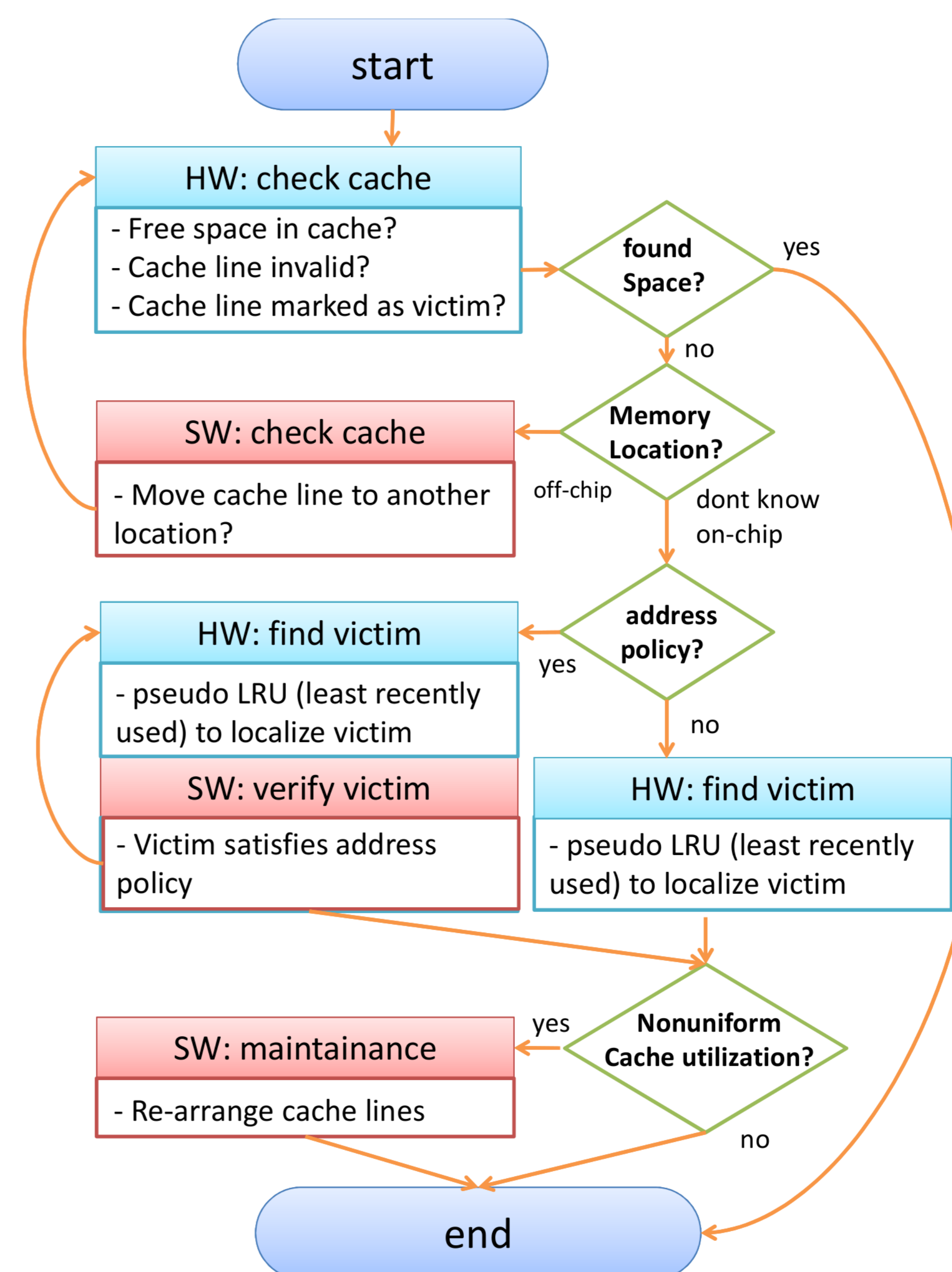


Figure: replacement policy flow graph

Depending on where memory is fetched up to hundreds of clock cycles are used. Embedded processors typically stall on a cache miss. Instead of waiting for the cache miss to be resolved processors can execute an advanced cache replacement algorithm.

Conclusions

- ▶ We propose a software managed multi-banked first level data cache for parallel systems
 - ▶ highly configurable
 - ▶ more area and power efficient than a pure hardware implementation of highly associative cache
- ▶ We propose an application aware software controlled replacement strategy
 - ▶ use both hardware and software to implement replacement policy