18742 Parallel Computer Architecture Caching in Multi-core Systems

Vivek Seshadri
Carnegie Mellon University

Fall 2012 – 10/03

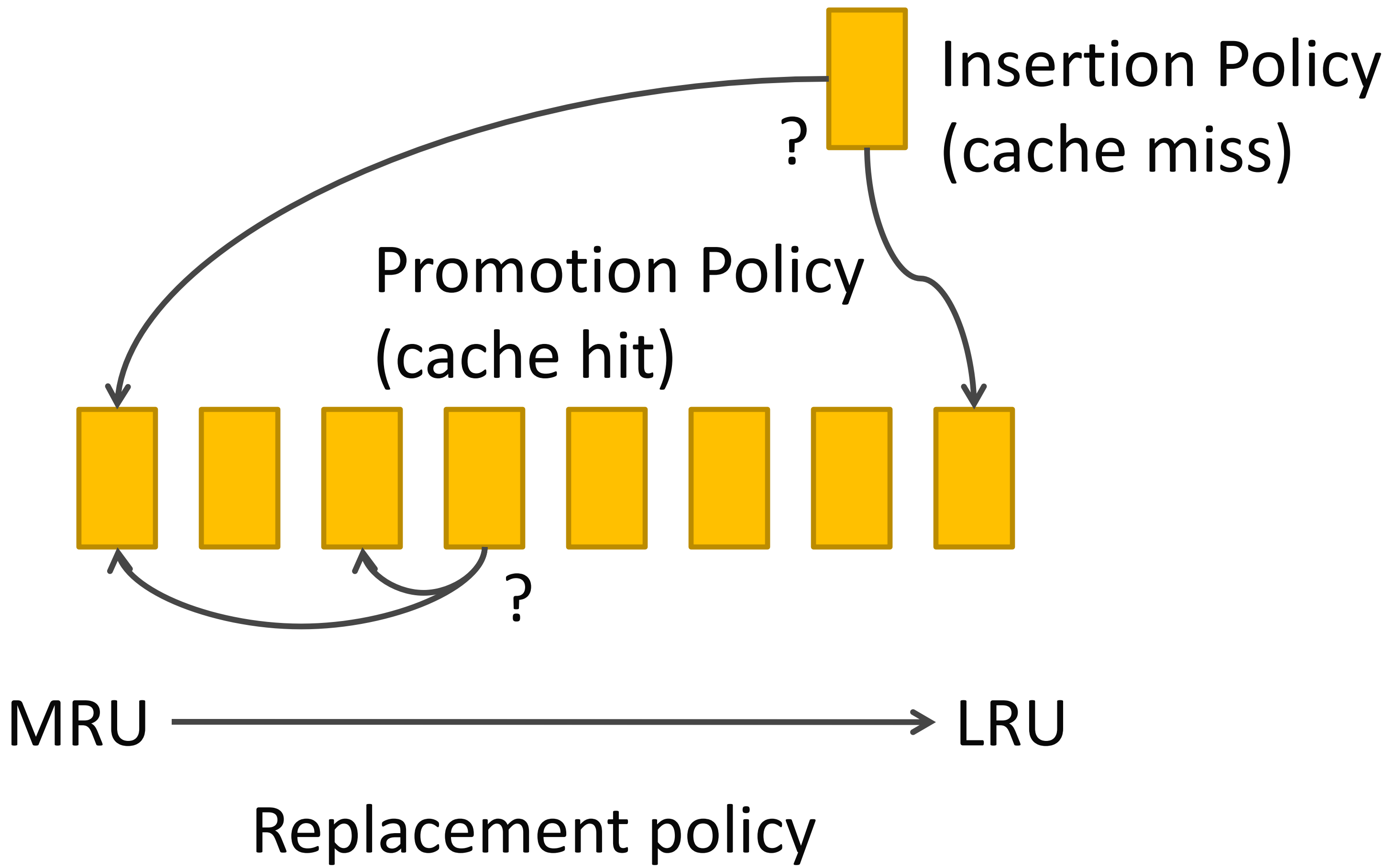
Problems in Multi-core Caching

- Managing individual blocks
  - Demand-fetched blocks
  - Prefetched blocks
  - Dirty blocks
- Application awareness
  - High system performance
  - High fairness

Part 1

Managing Demand-fetched Blocks

Cache Management Policy



Traditional LRU Policy

- Insertion Policy
  - Insert at MRU
  - Rationale: Access => More access

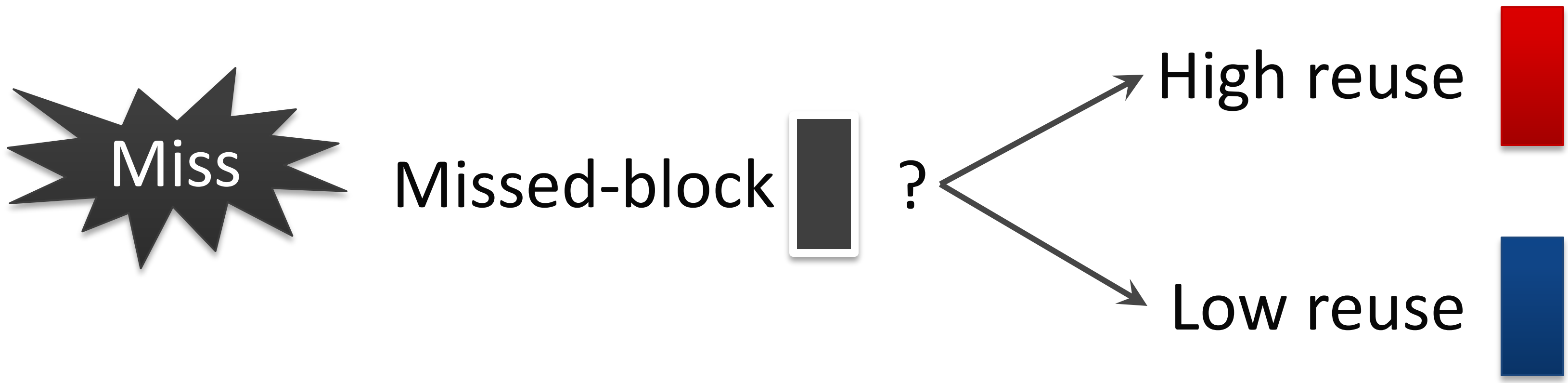
- Promotion Policy
  - Promote to MRU
  - Rationale: Reuse => More reuse

Problem with LRU's Insertion Policy

- Cache pollution
  - Blocks may be accessed only once
  - Example: Scans

- Cache thrashing
  - Lot of blocks may be reused
  - Example: Large working sets

Addressing Cache Pollution



High Reuse: Insert at MRU
Low Reuse: Insert at LRU

Keep track of the reuse behavior of every cache block in the system. **Impractical.**

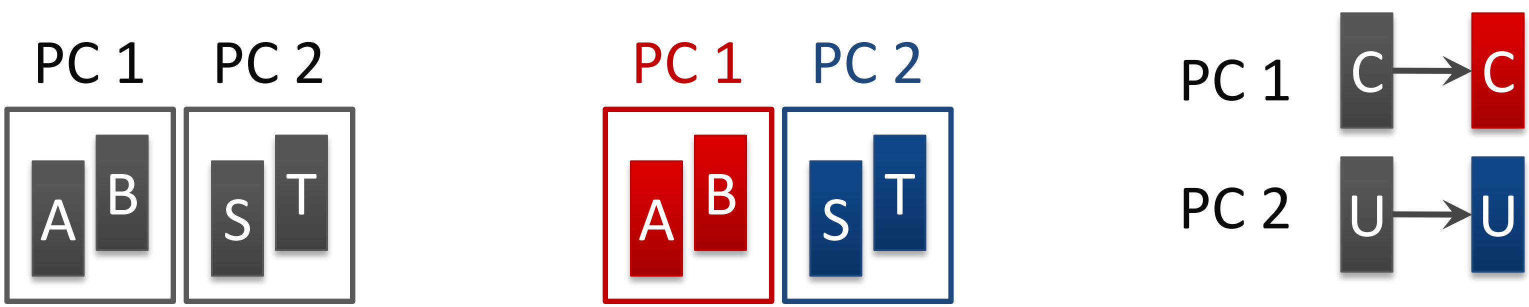
Work on Reuse Prediction

Use program counter or memory region information.

1. Group Blocks

2. Learn group behavior

3. Predict reuse



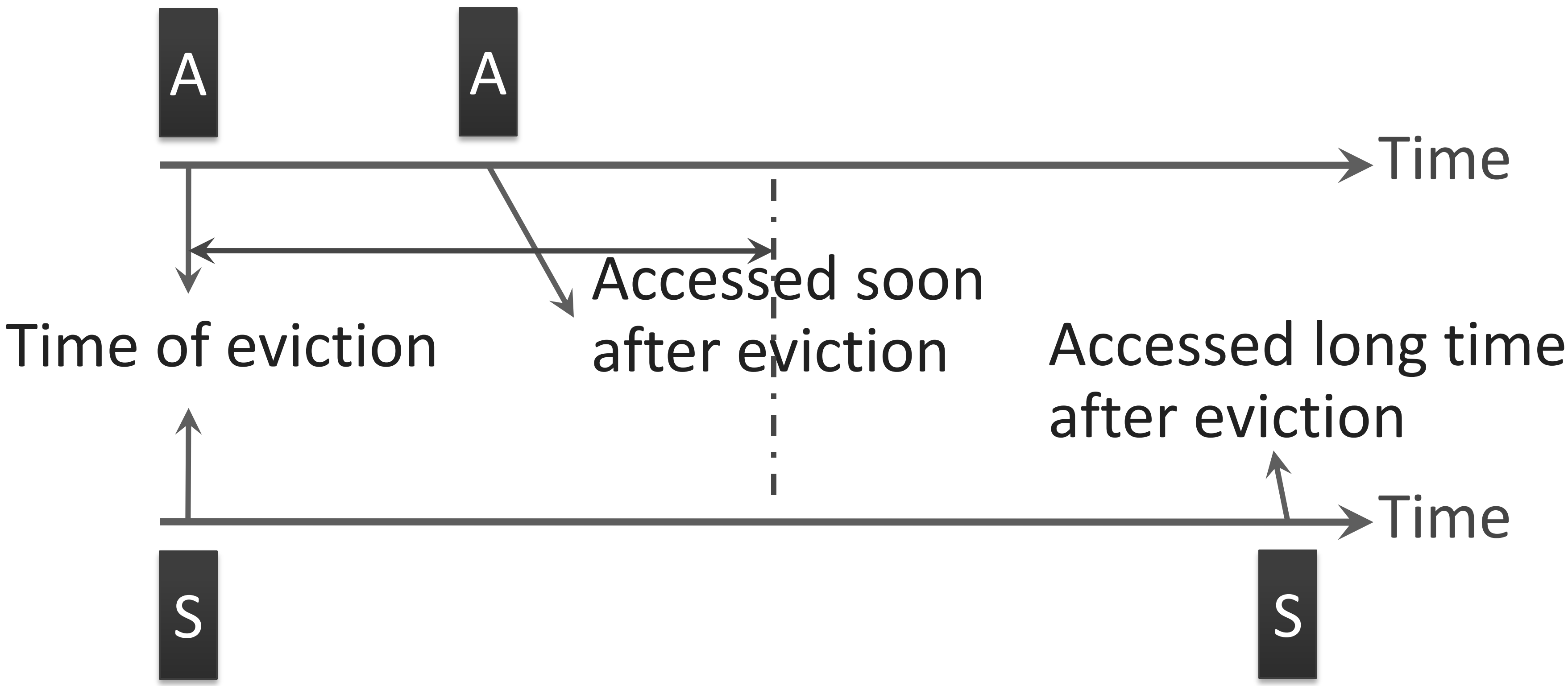
Run-time Bypassing (RTB) – Johnson+ ISCA'97

Single-usage Block Prediction (SU) – Piquet+ ACSAC'07

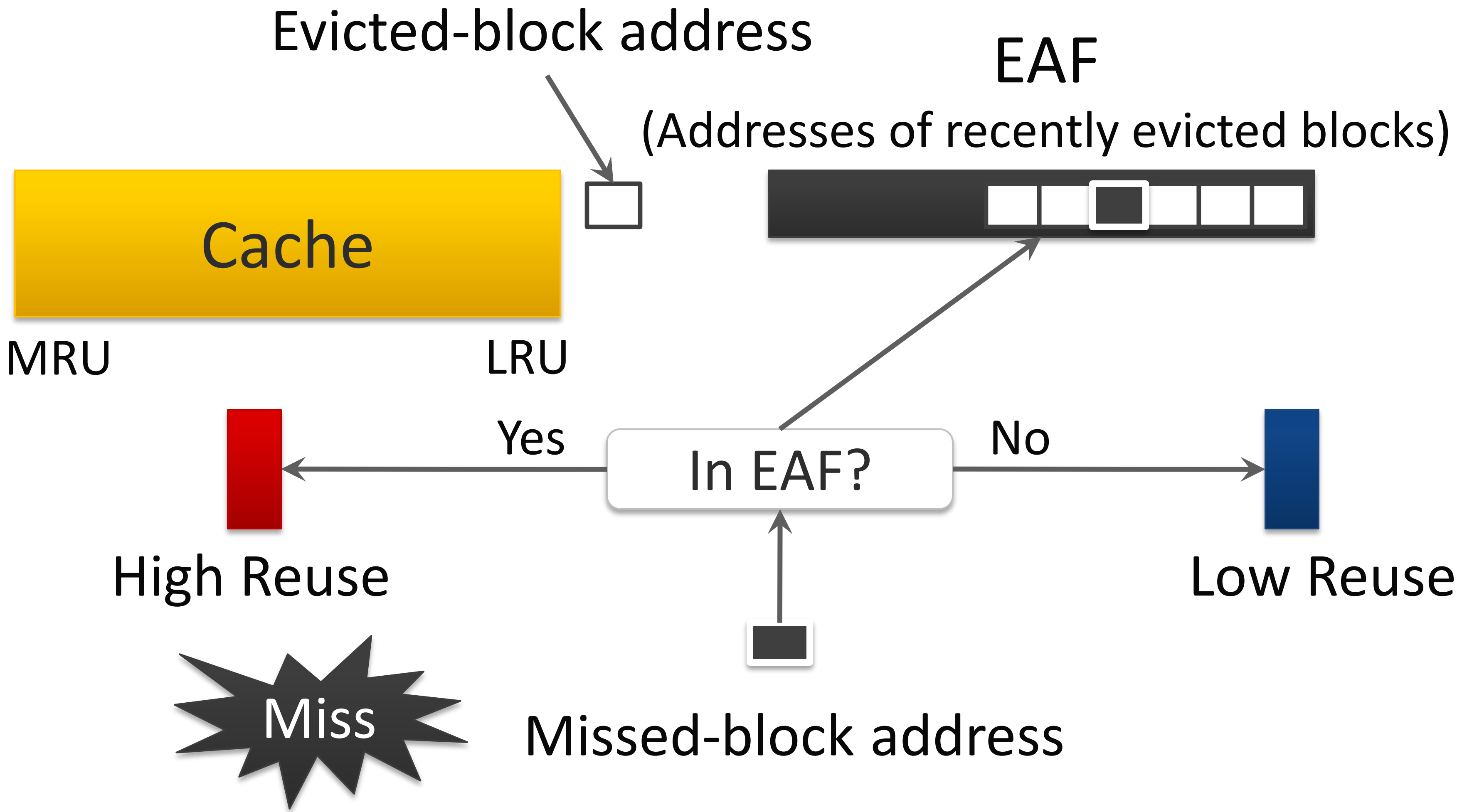
Signature-based Hit Prediction (SHIP) – Wu+ MICRO'11

Evicted-Address Filters: Idea

Use recency of eviction to predict reuse



Evicted-Address Filter (EAF)



Addressing Cache Thrashing

Bimodal Insertion Policy

Insert at MRU with low probability

Insert at LRU with high probability

A fraction of the working set retained in the cache

TA-DIP – Qureshi+ ISCA'07, Jaleel+ PACT'08

TA-DRRIP – Jaleel+ ISCA'10

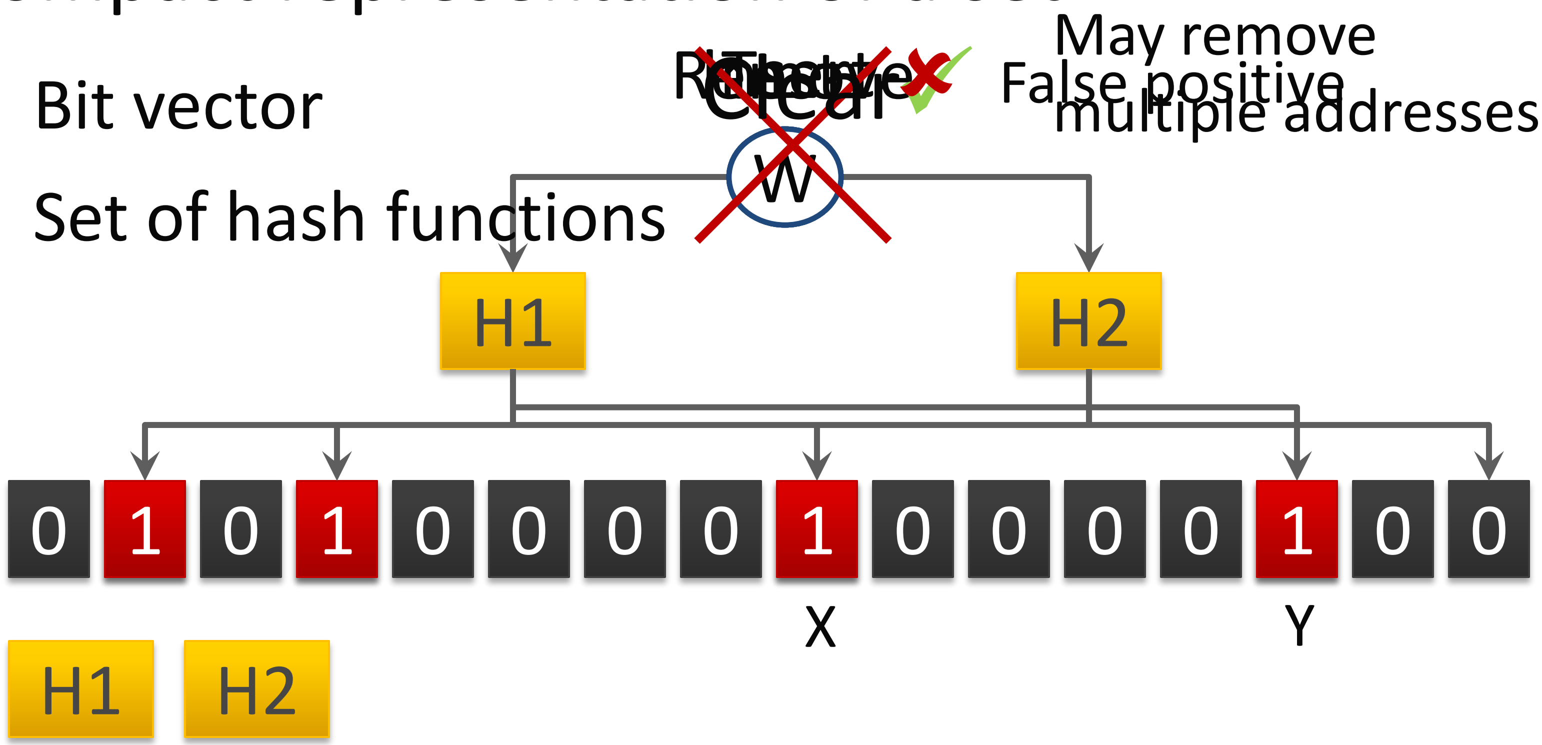
Addressing Pollution **and** Thrashing

- Combine the two approaches?
- Problems?
- Ideas?
- EAF using a Bloom filter

Bloom Filter

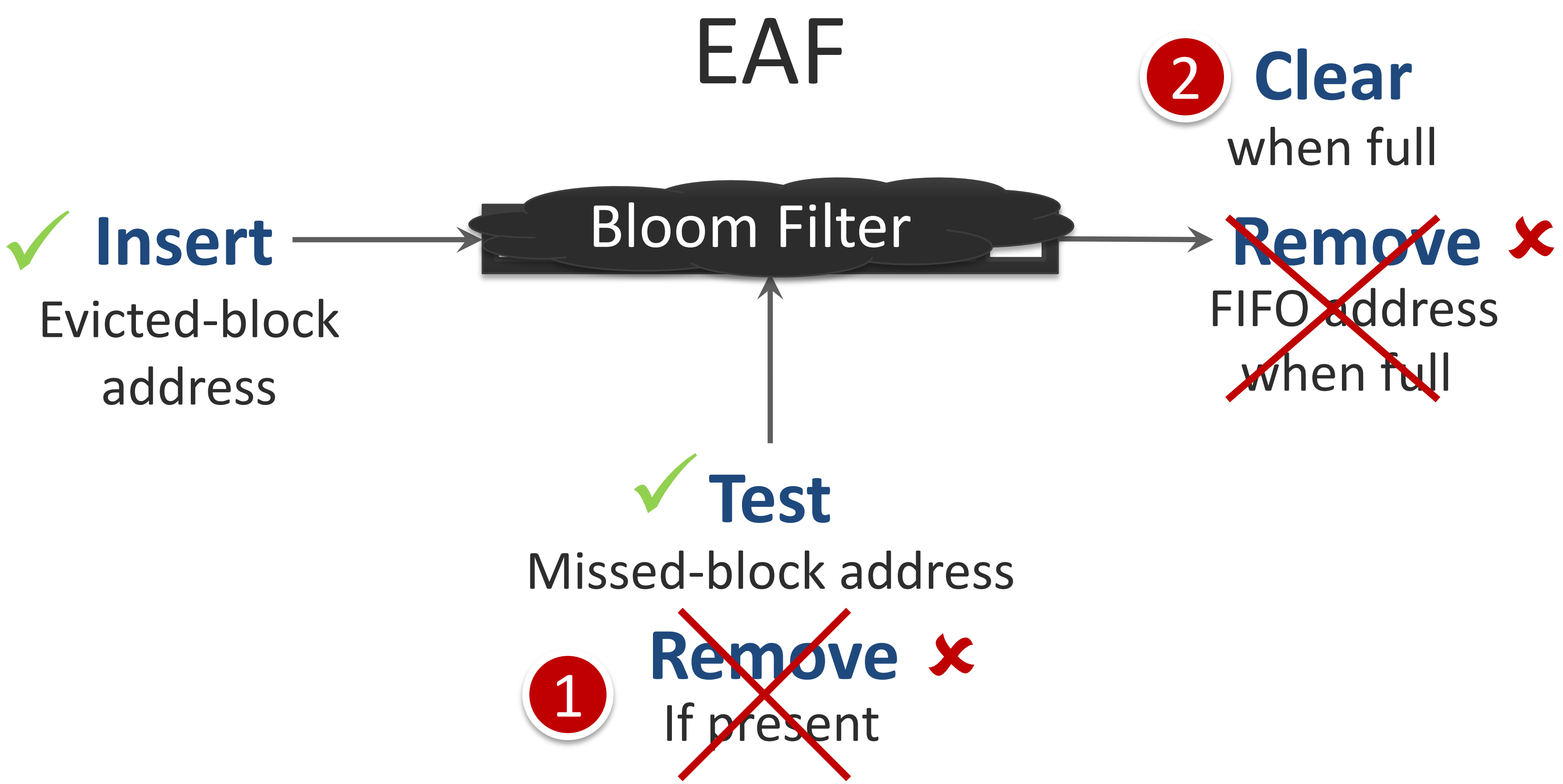
Compact representation of a set

1. Bit vector
2. Set of hash functions



Inserted Elements: X Y

EAF using a Bloom Filter



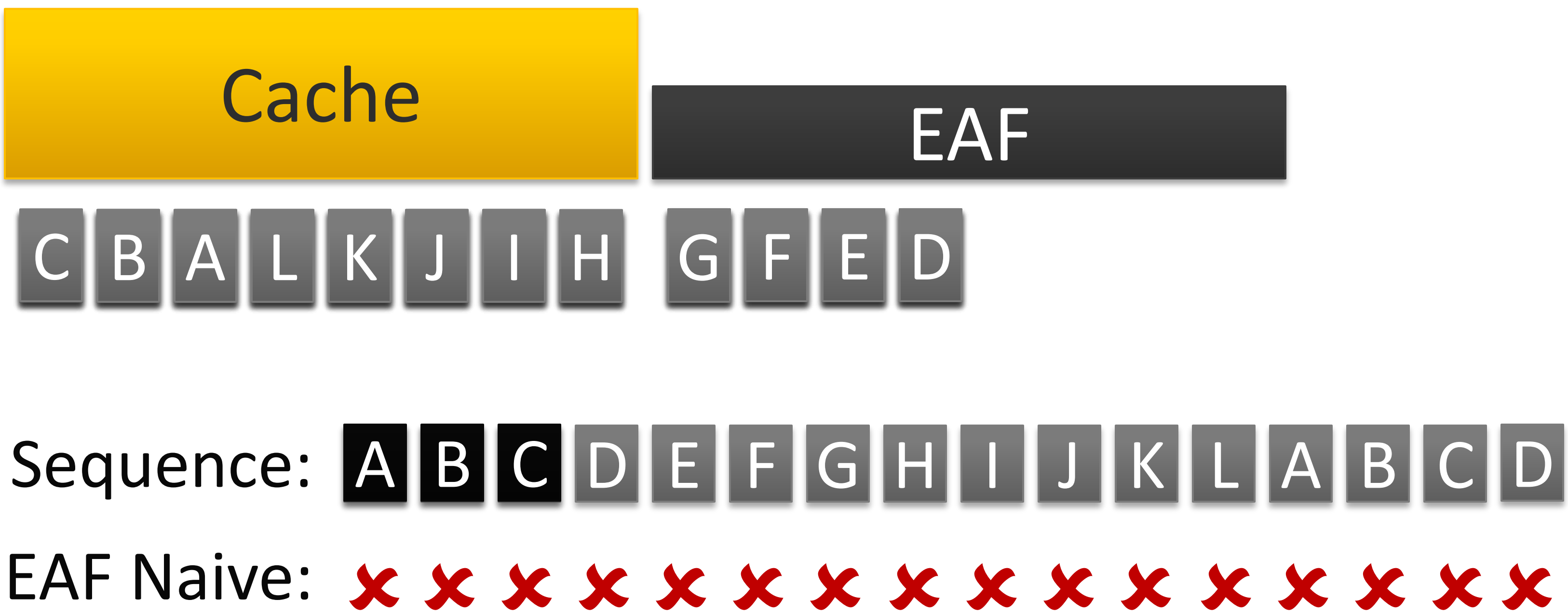
Large Working Set: 2 Cases

1 Cache < Working set < Cache + EAF

2 Cache + EAF < Working Set

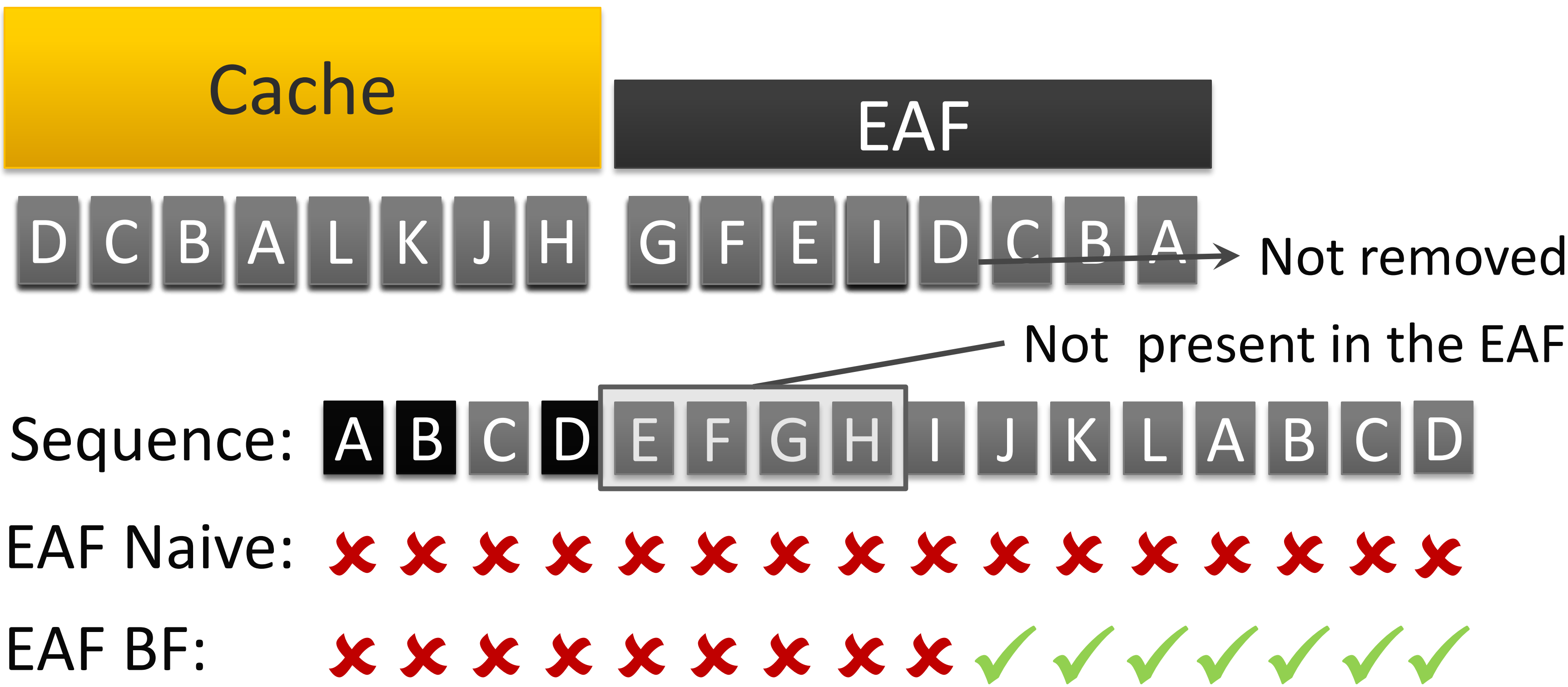
Large Working Set: Case 1

Cache < Working set < Cache + EAF



Large Working Set: Case 1

Cache < Working set < Cache + EAF



Bloom-filter based EAF mitigates thrashing

Large Working Set: Case 2

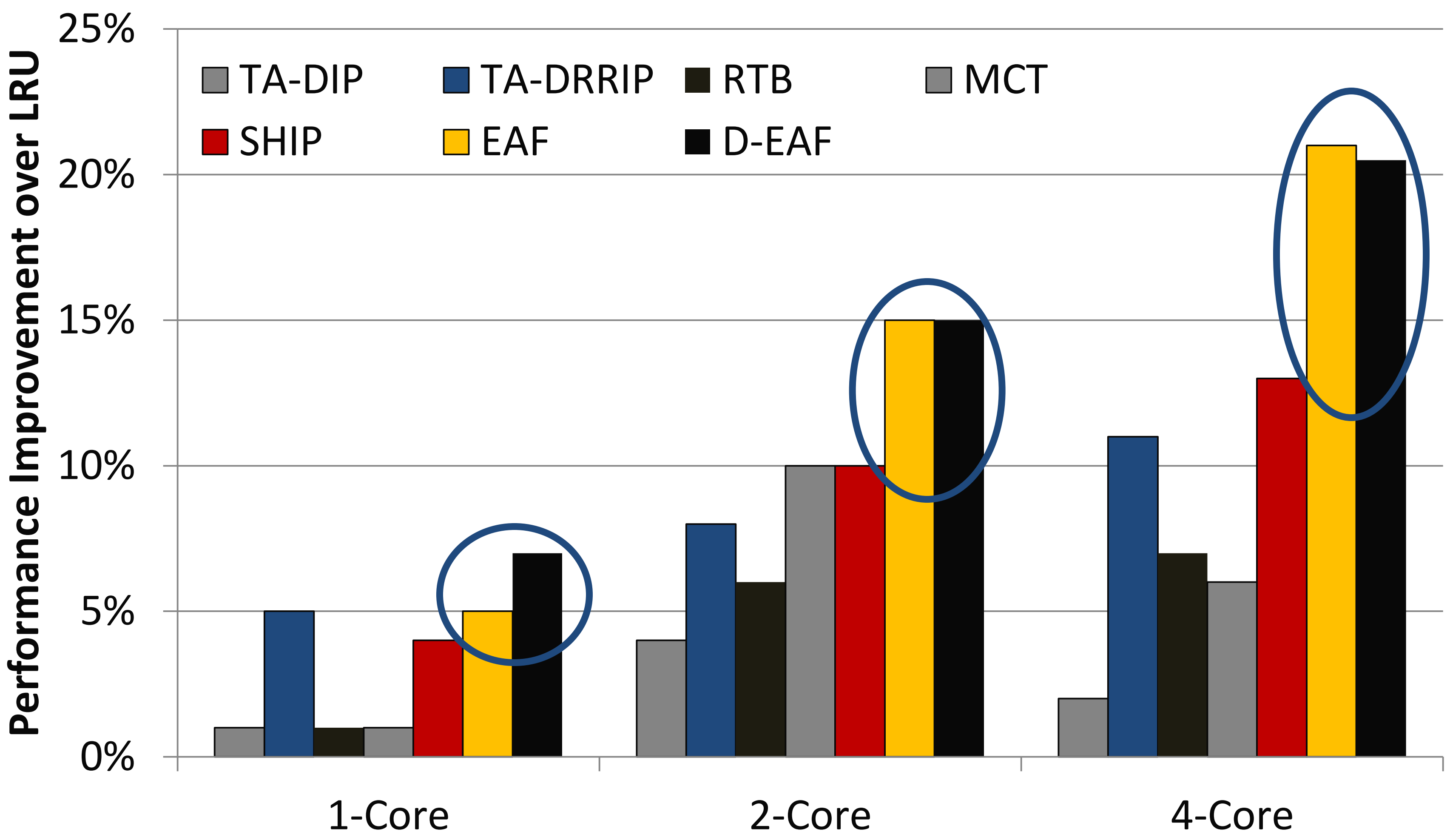
Cache + EAF < Working Set

Problem: All blocks are predicted to have low reuse

Allow a fraction of the working set to stay in the cache

Use **Bimodal Insertion Policy** for low reuse blocks. Insert few of them at the MRU position

Results – Summary



Part 2
Managing Prefetched Blocks

Hopefully in a future course!

Part 2
Managing Dirty Blocks

Hopefully in a future course!

Part 2
Application Awareness

Cache Partitioning

- Goals
  - High performance
  - High fairness
  - Both?
- Partitioning Algorithm/Policy
  - Determine how to partition the cache
- Partitioning Enforcement
  - Enforce the partitioning policy

Utility-based Cache Partitioning

- Way-based partitioning
- More benefit/utility => More cache space
- Problems
  - # Cores > # ways
  - Need core ID with each tag

Promotion-Insertion Pseudo Partitioning

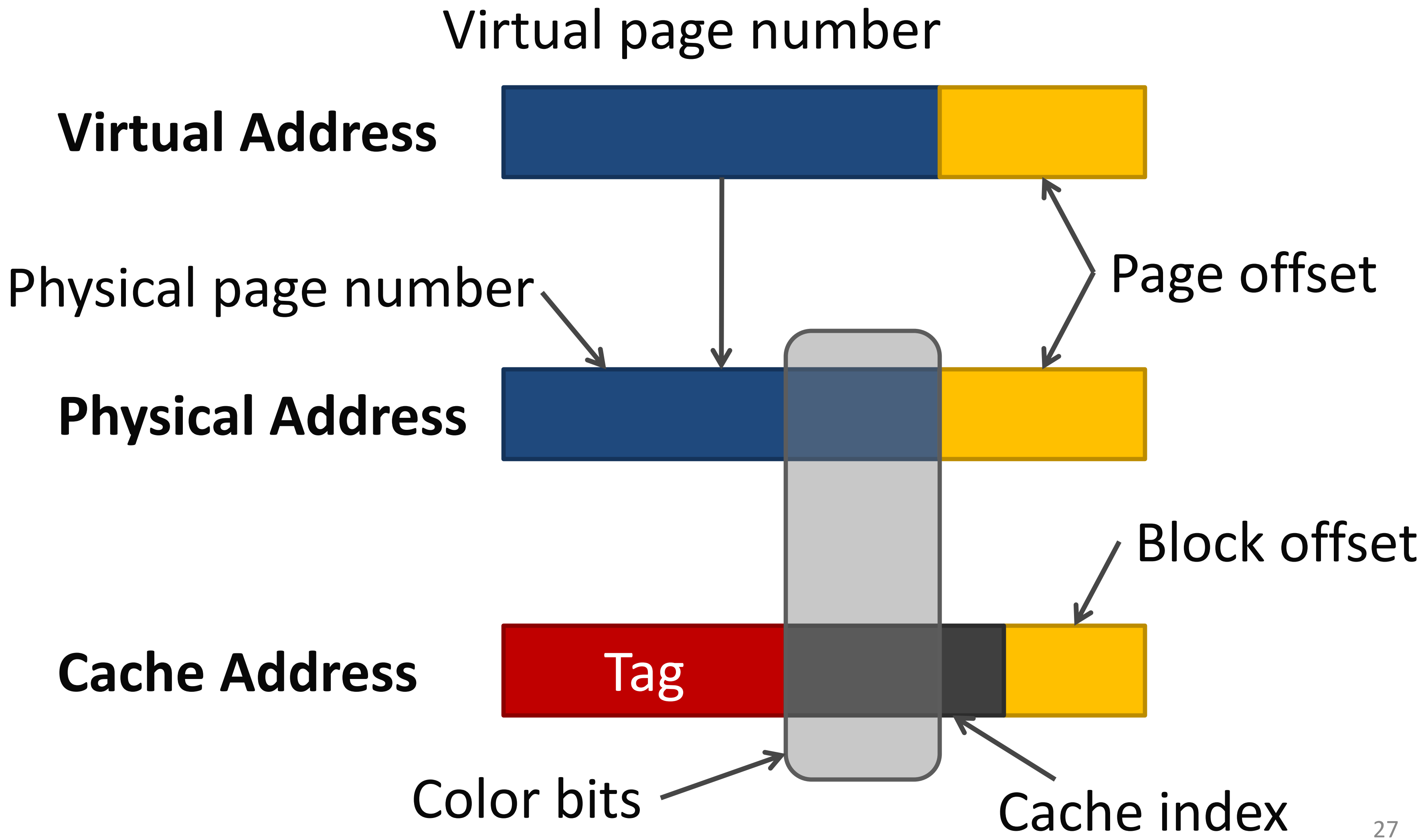
- Partitioning Algorithm
  - Same as UCP
- Partitioning Enforcement
  - Modify cache insertion policy
  - Probabilistic promotion

Promotion Insertion Pseudo Partitioning – Xie+ ISCA'09

Software-based Partitioning

- Cho and Jin, "Managing Distributed, Shared L2 Caches through OS-Level Page Allocation," MICRO 2006.
- Lin et al., "Gaining Insights into Multi-Core Cache Partitioning: Bridging the Gap between Simulation and Real Systems," HPCA 2008

Page Coloring



OS-based Partitioning

- Enforcing Partition
  - Colors partition the cache
  - Assign colors to each application
  - Application’s pages are allocated in the assigned colors
  - Number of colors => amount of cache space
- Partitioning algorithm
  - Use hardware counters
  - # Cache misses

Set Imbalance

- Problem
  - Some sets may have lot of conflict misses
  - Others may be under-utilized

- Solution approaches
  - Randomize index
    - Not good for cache coherence. Why?
  - Set balancing cache
    - Pair an under-utilized set with one that has frequent conflict misses

That's it!