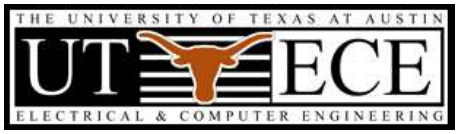
# Performance-Directed Secondary Cache Design

Moinuddin K. Qureshi

Department of Electrical and Computer Engineering,
The University of Texas at Austin.



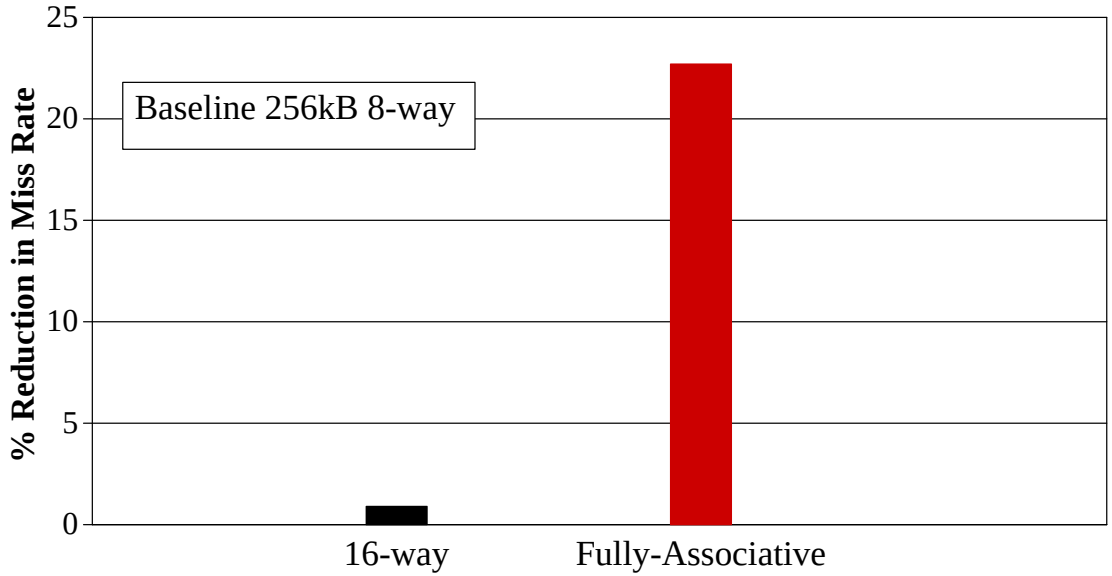
# Background

- Memory latency is well over 300 cycles [Sprangle+ ISCA'02] and it continues to increase
- Caches reduce the number of memory accesses
  - Cache hit provides fast access to recently accessed data
  - Cache miss can stall the processor for hundreds of cycles
- To sustain high performance, it is important to reduce the number of cache misses
- Cache misses can be reduced by intelligent management and design of cache resources

# The Problem of L2 Cache Misses

- In a two level cache hierarchy, L1 is constrained by access time and L2 is constrained by transistor budget. In this talk we will focus on only L2 caches
- Even with more than half of the on-chip transistors devoted to L2 cache, cache misses occur frequently and cause performance loss [Hankins+ MICRO'03]
- Existing techniques for managing L2 cache do not work well enough, which results in increased number of cache misses and reduced performance. Three key reasons of inefficiency:
  - Problem with cache organization
  - Problem with cache replacement
  - Problem with cache hierarchy

# A Case for Revisiting L2 Cache Organization




- Current L2 designs [Pentium, Power, Athlon, Itanium] use a set-associative organization
- L2 caches use sequential tag and data lookup to save power [Itanium, Alpha]
- Can we leverage the sequential lookup to get the benefit of a highly associative organization?

# A Case for Revisiting Cache Replacement

- Current processors try to service L2 miss in parallel
- Isolated misses are more costly on performance than parallel misses
- Current cache replacement algorithms assume uniform cost for all misses
- Performance can be improved by making the replacement engine aware of the cost differential, so that it can reduce the number of costly misses

# Outline

- Introduction
- The V-Way Cache
- MLP-Aware Cache Replacement
- Summary

# Fully Associative Caches: Cost vs. Benefit

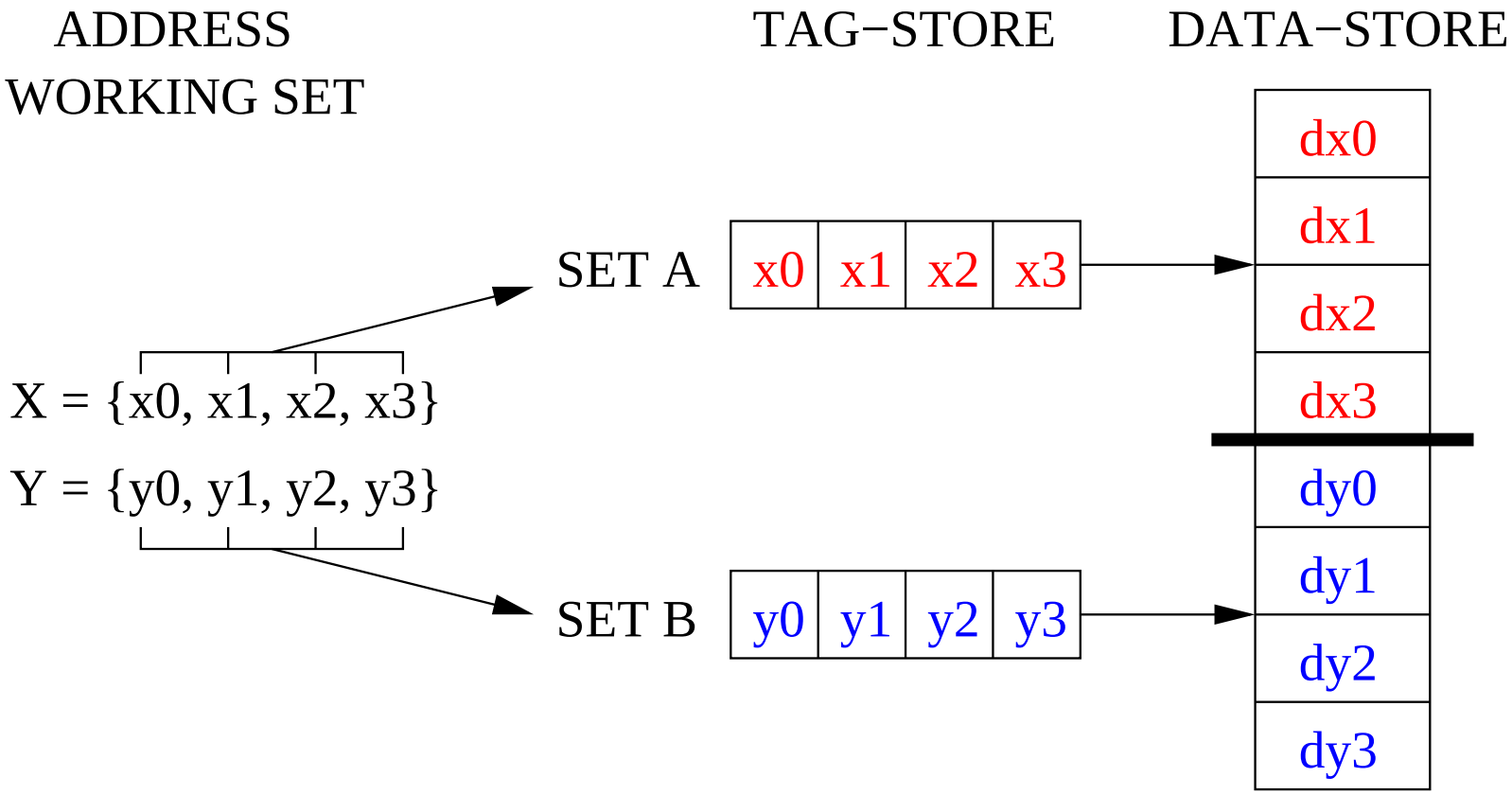
- Benefits
  - Conflict miss elimination
  - Global replacement (finds the best victim)
- Cost
  - Significant increase in the number of tag comparisons
  - Increased access latency
  - Increased power consumption

# Fully Associative Caches: Cost vs. Benefit

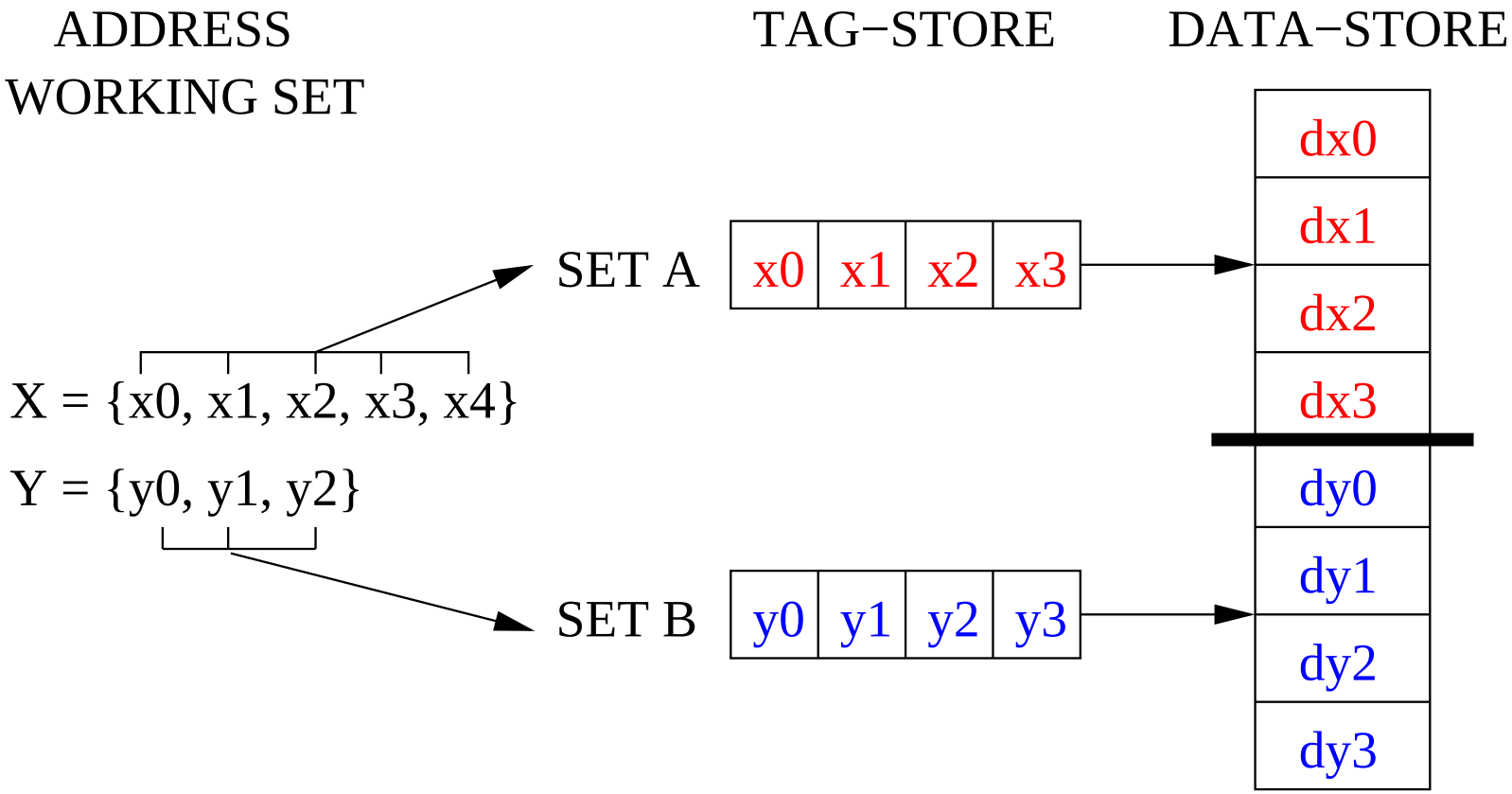
- Benefits
  - Conflict miss elimination
  - Global replacement (finds the best victim)
- Cost
  - Significant increase in the number of tag comparisons
  - Increased access latency
  - Increased power consumption

Can we get the benefits of a fully associative cache without paying the cost?

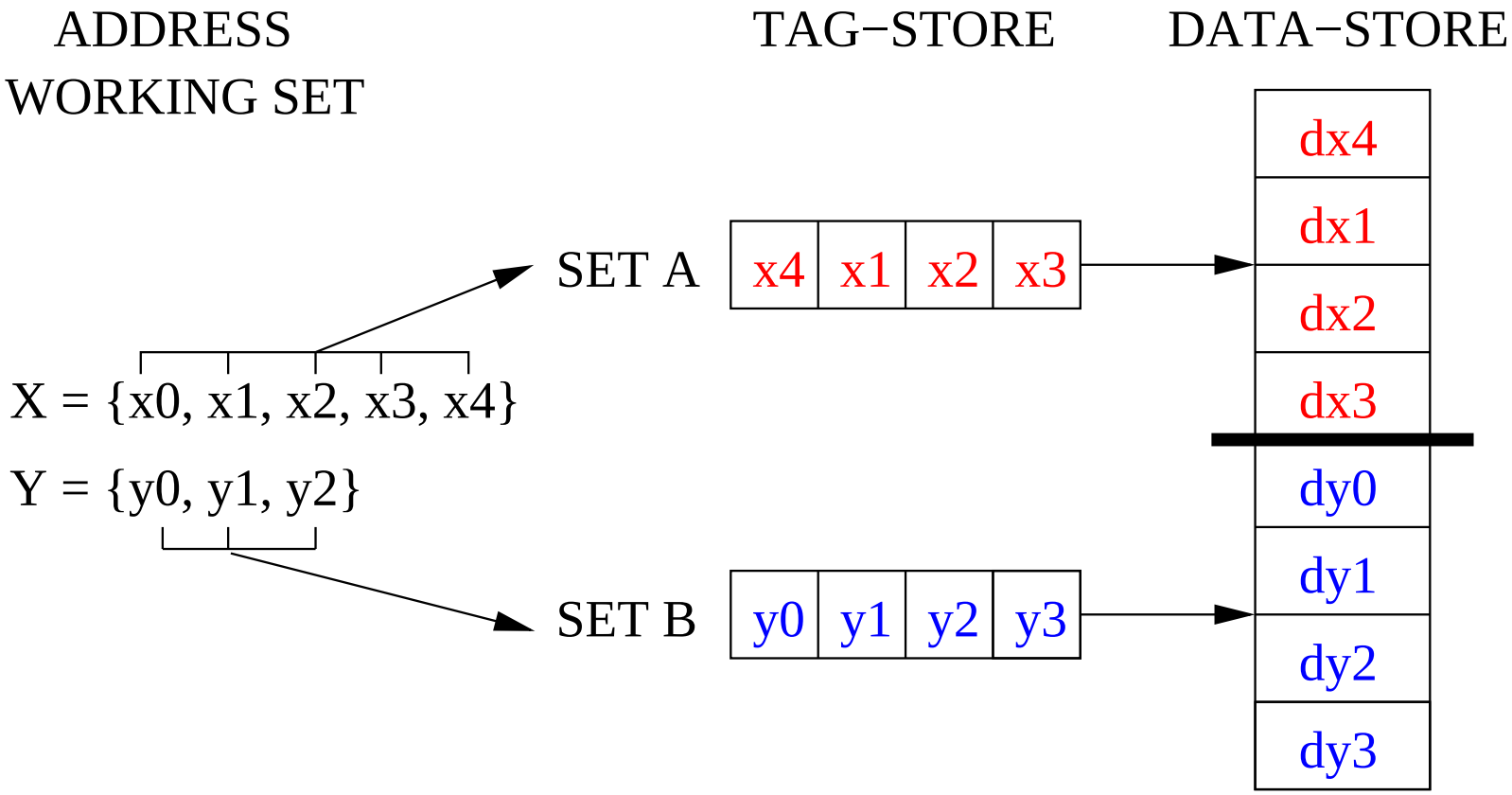
# Example of Local Replacement



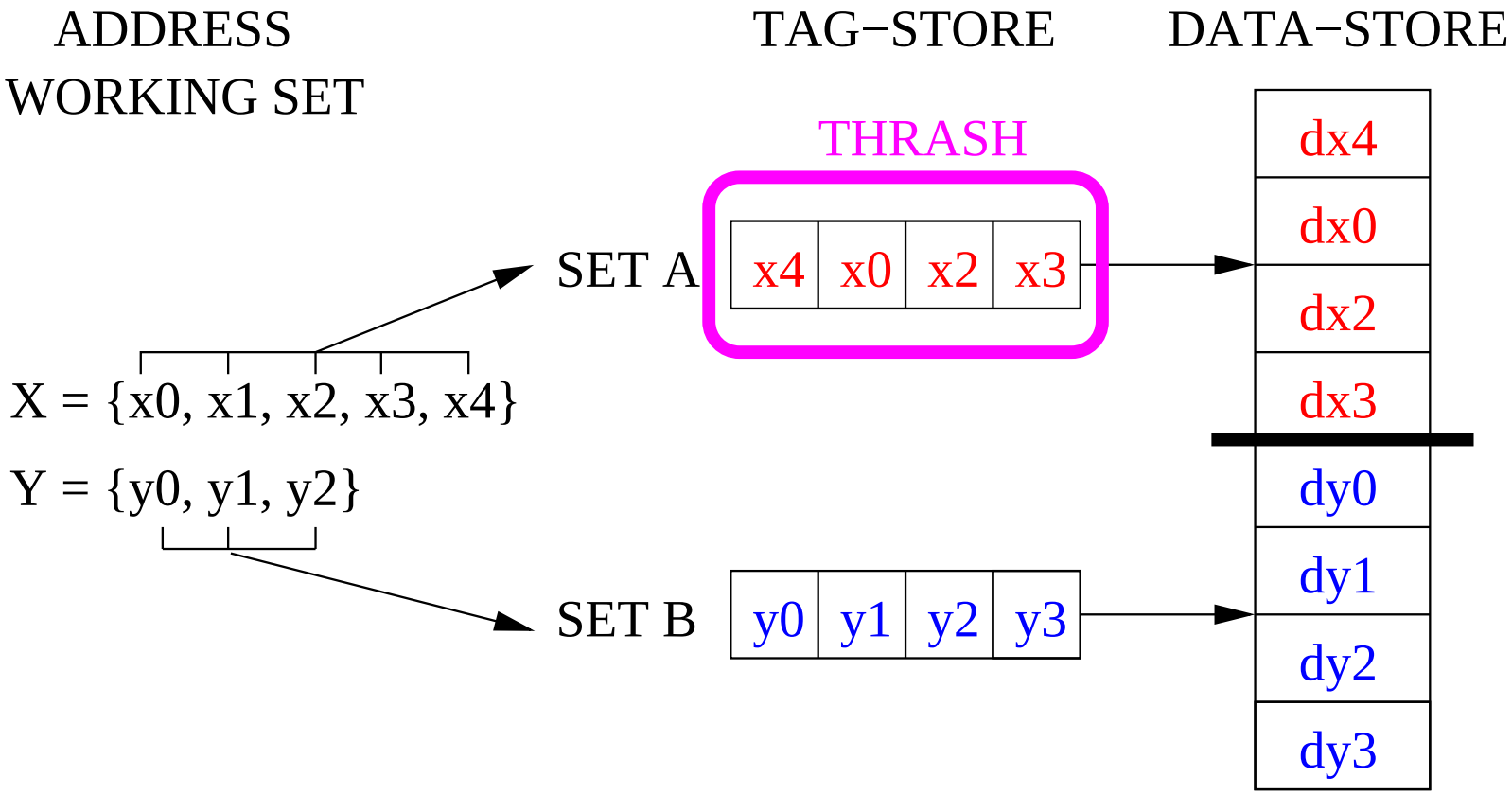
# Example of Local Replacement



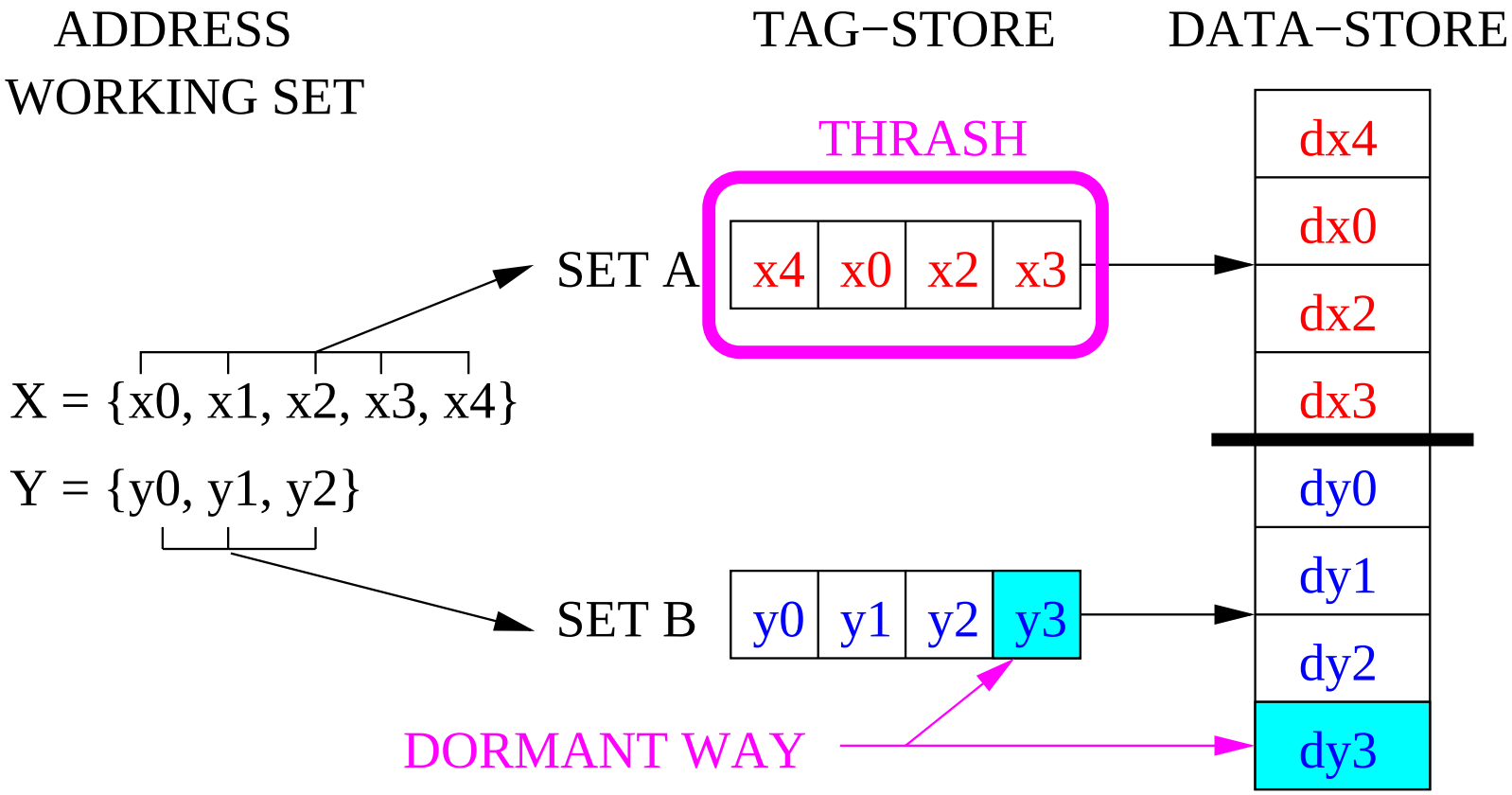
# Example of Local Replacement



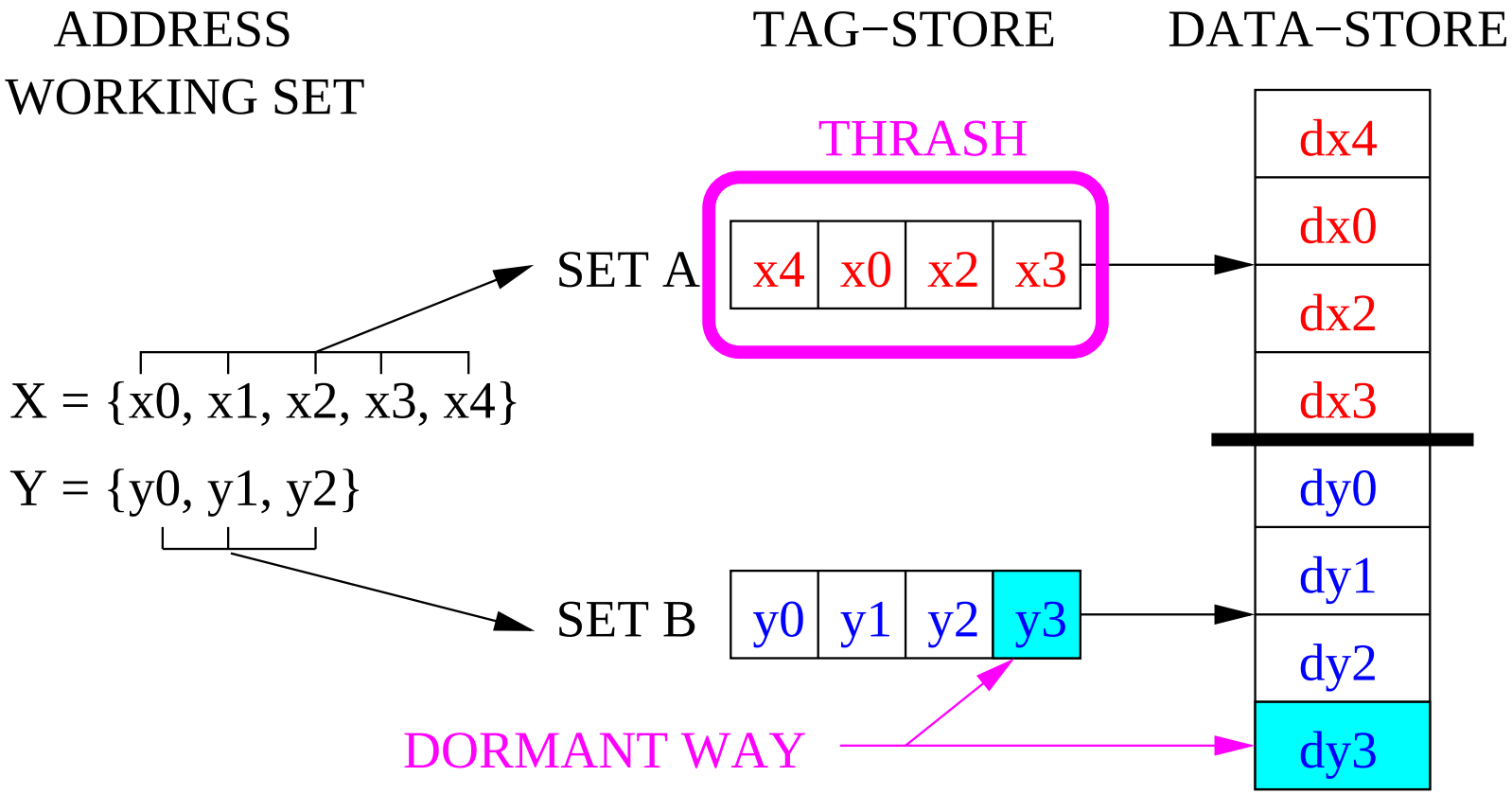
# Example of Local Replacement



# Example of Local Replacement

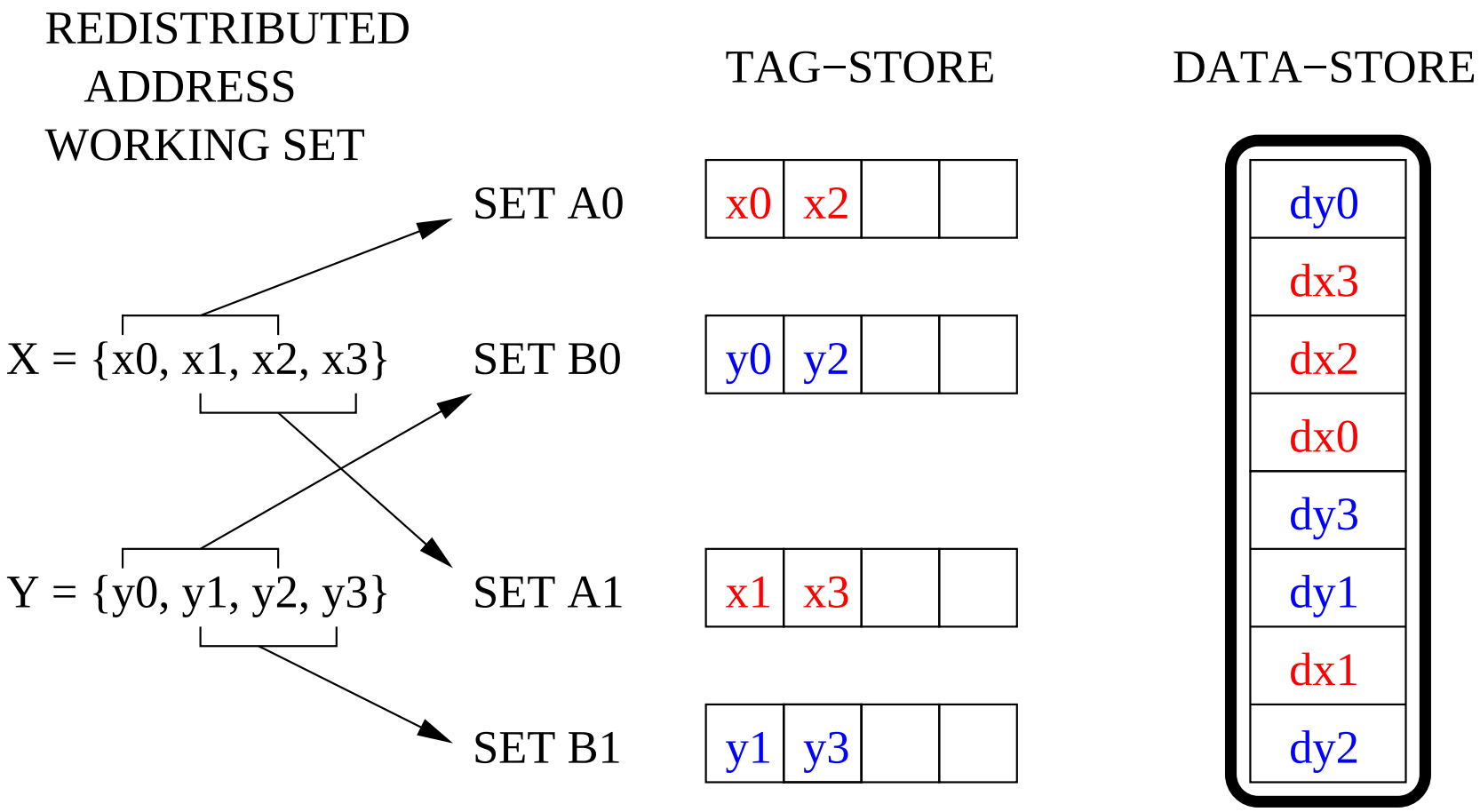


# Example of Local Replacement

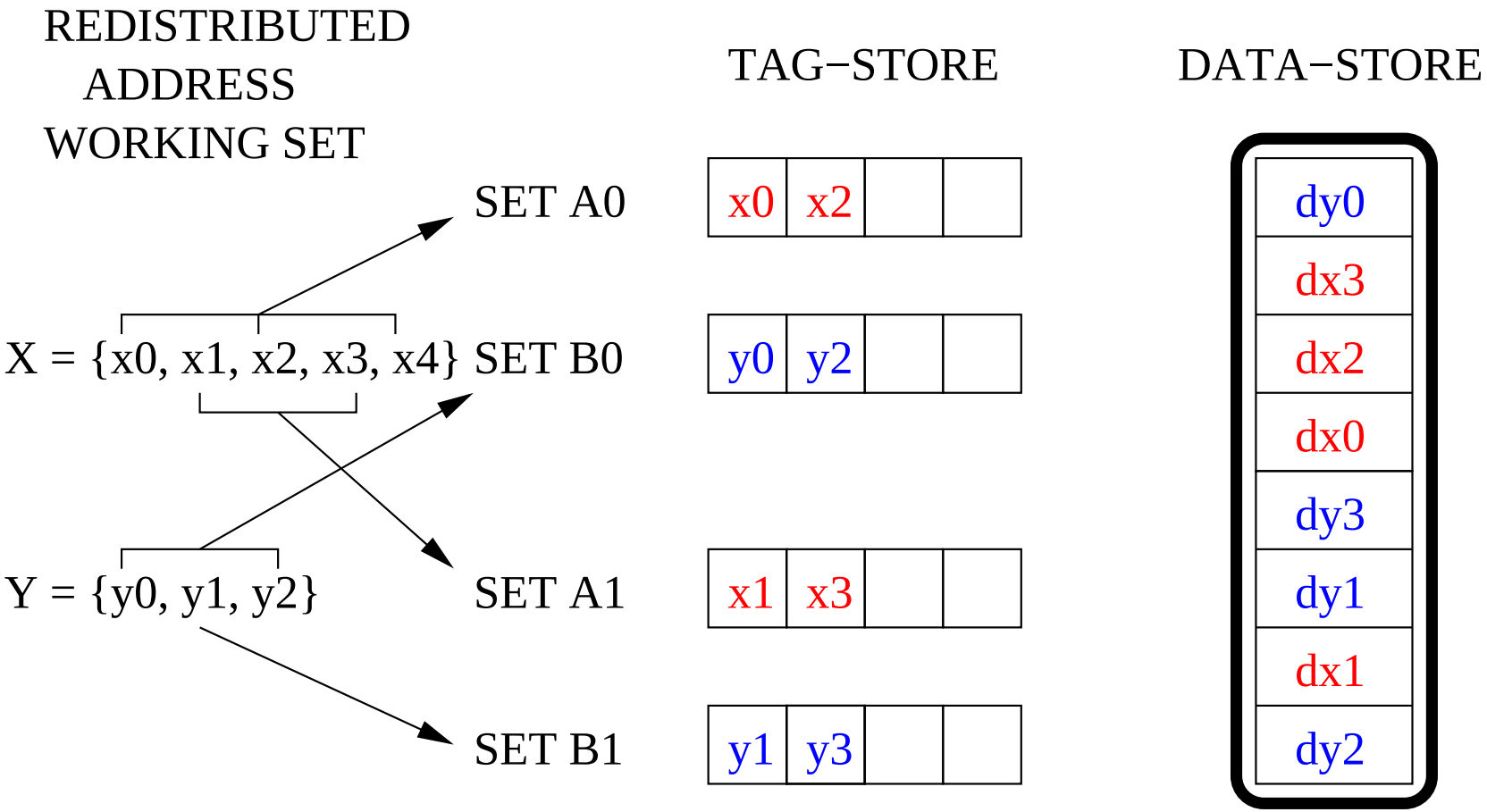



Static partitioning of resources.

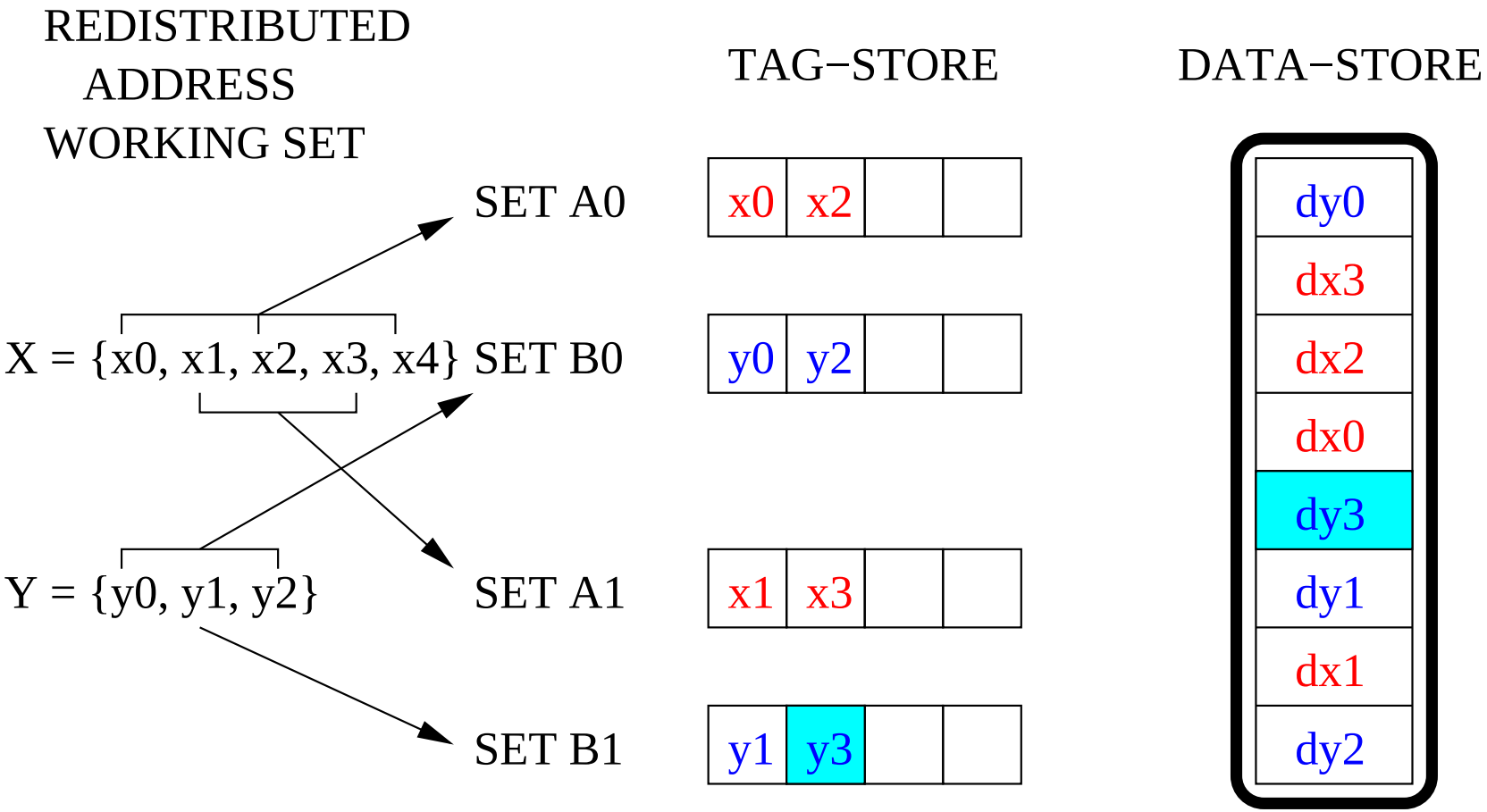
# Example of Global Replacement



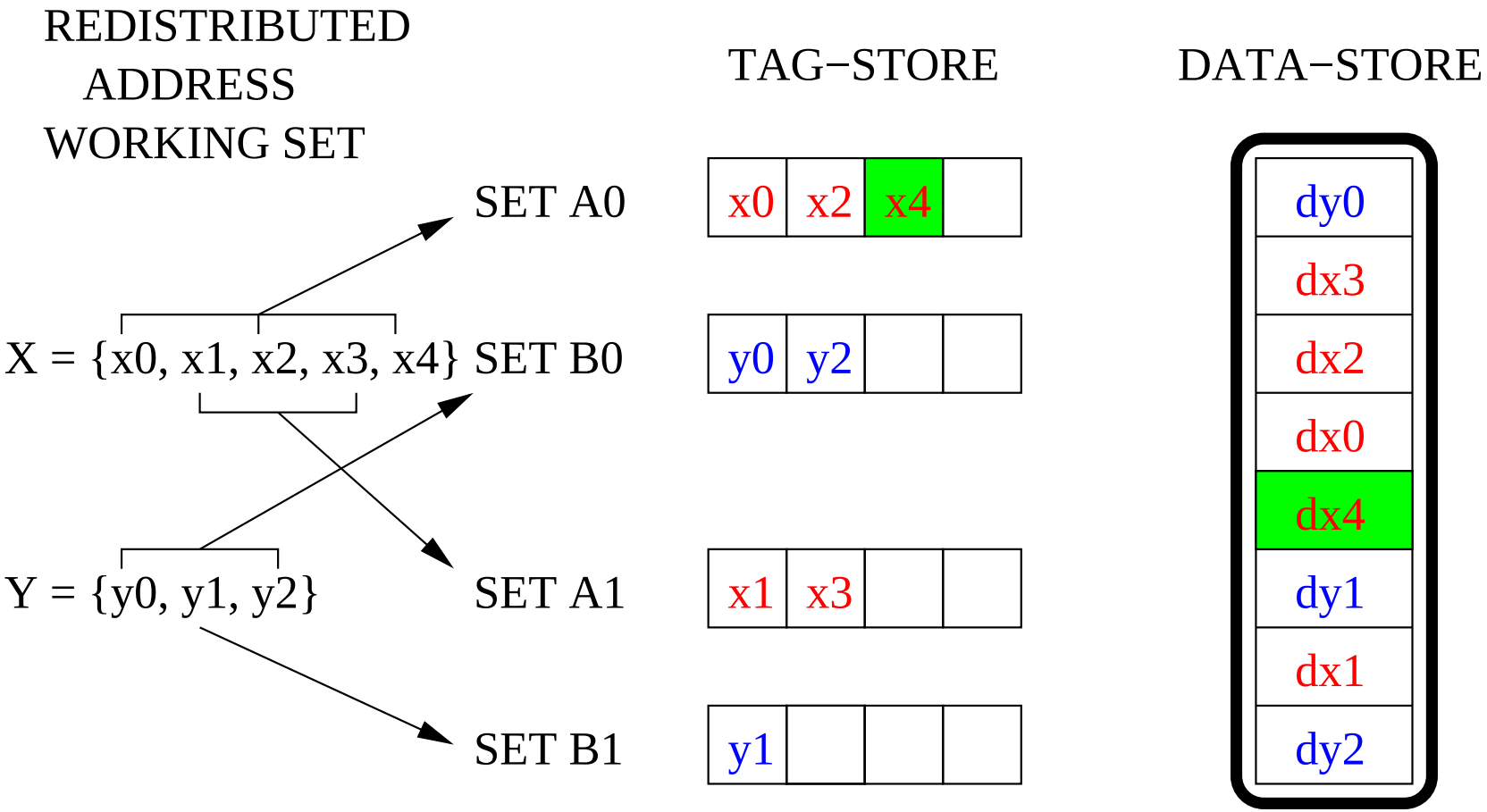
# Example of Global Replacement



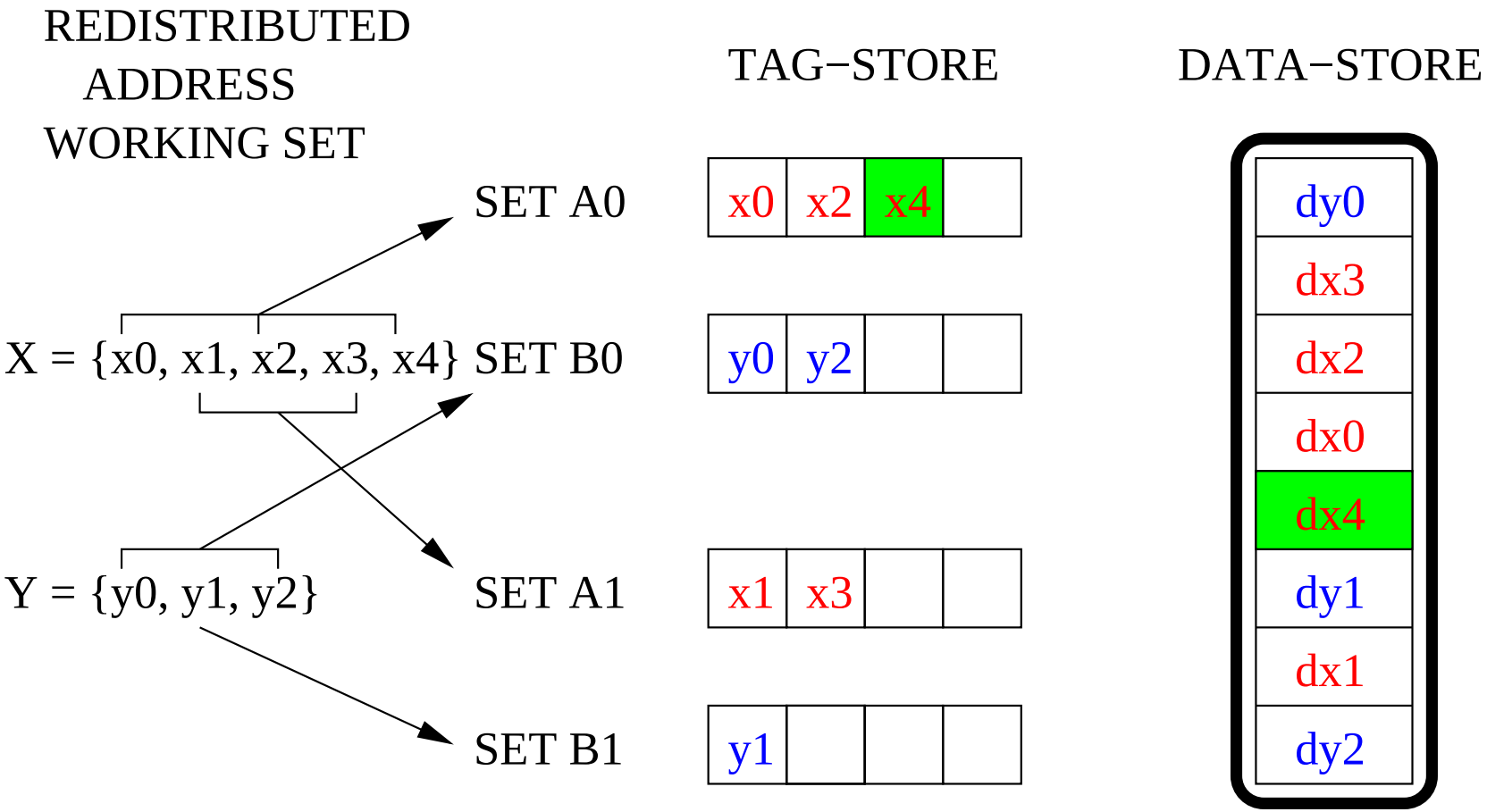
# Example of Global Replacement



# Example of Global Replacement

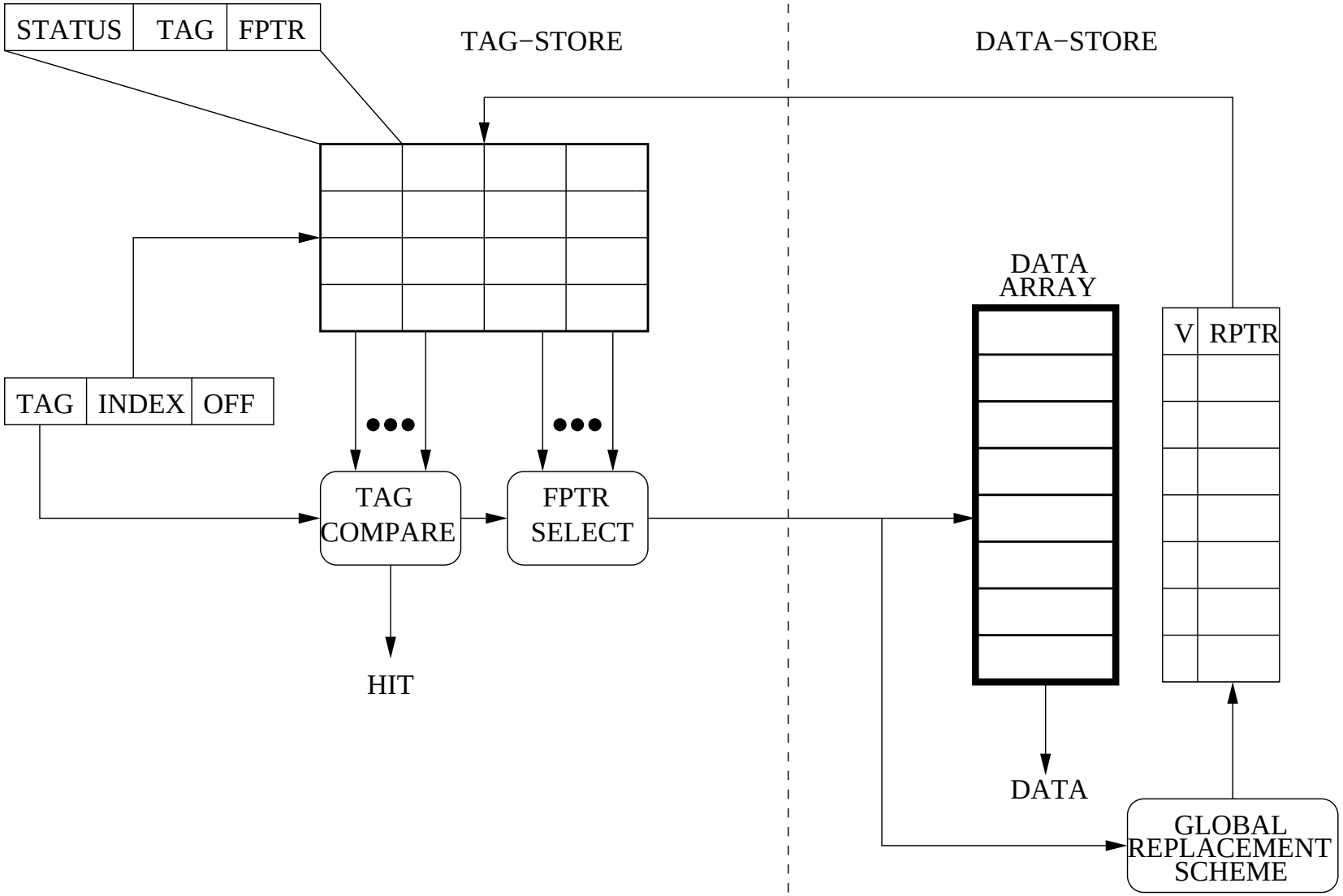


# Example of Global Replacement

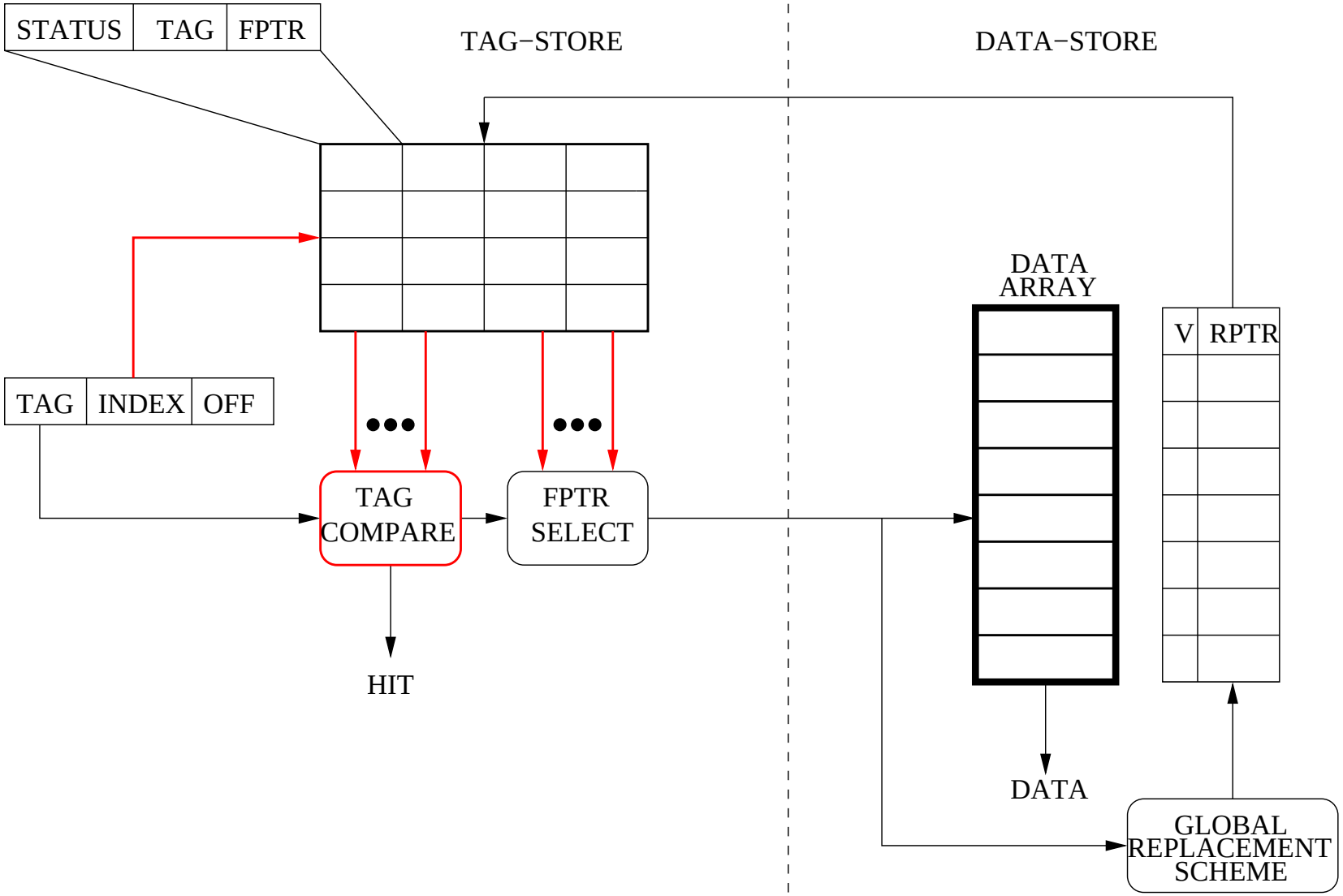



Dynamic sharing of resources!!

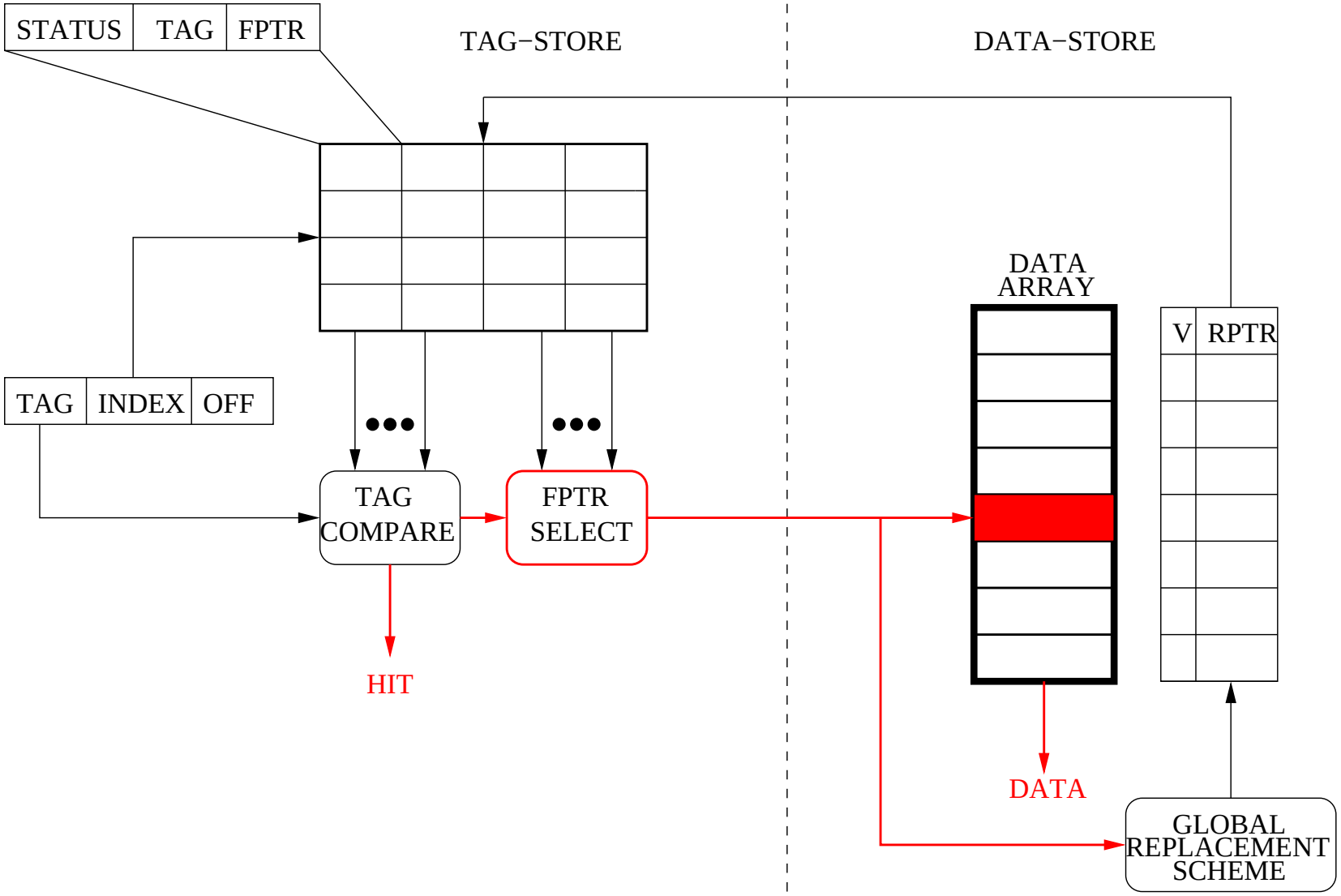
# The V-Way Cache



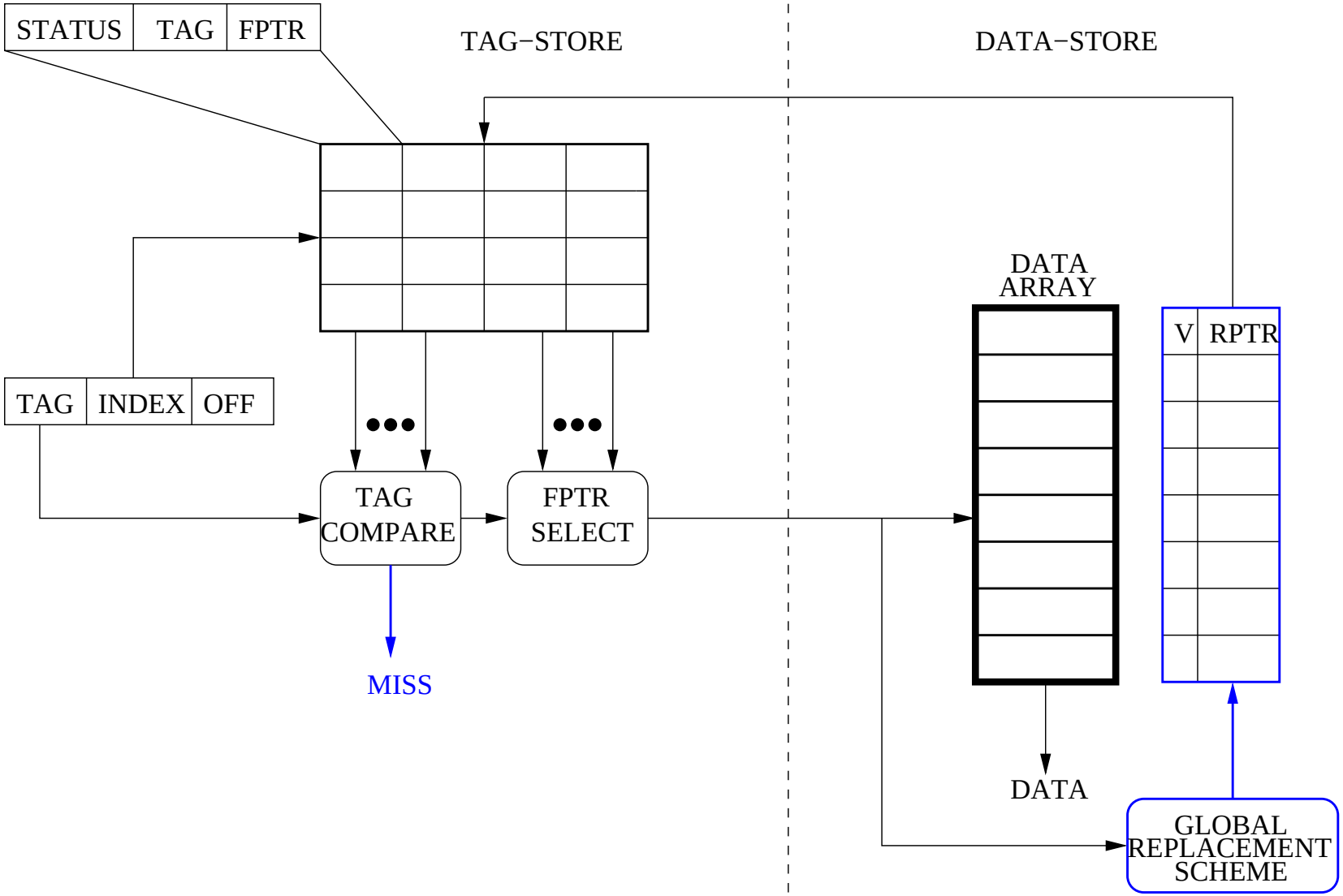
# The V-Way Cache



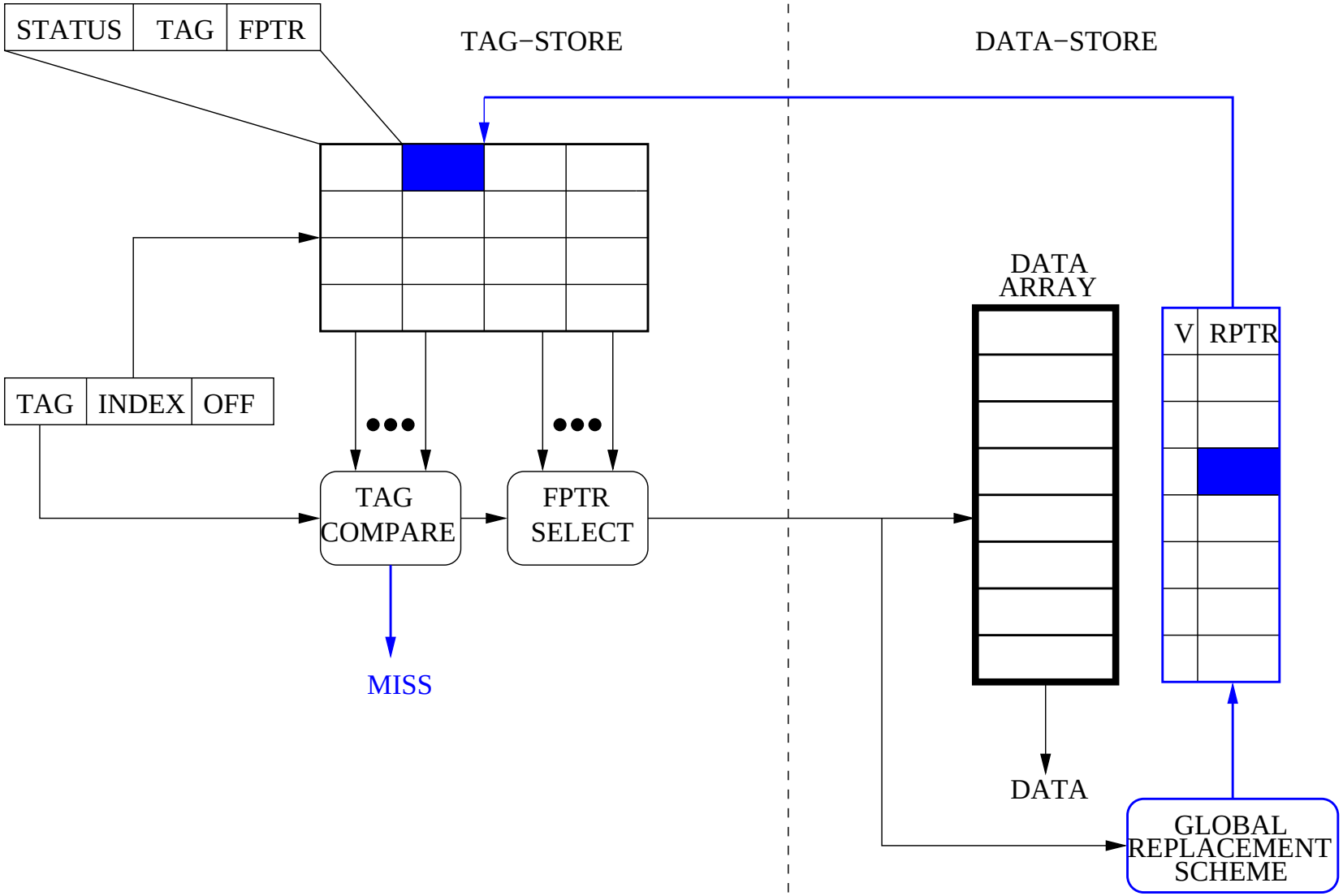
# The V-Way Cache



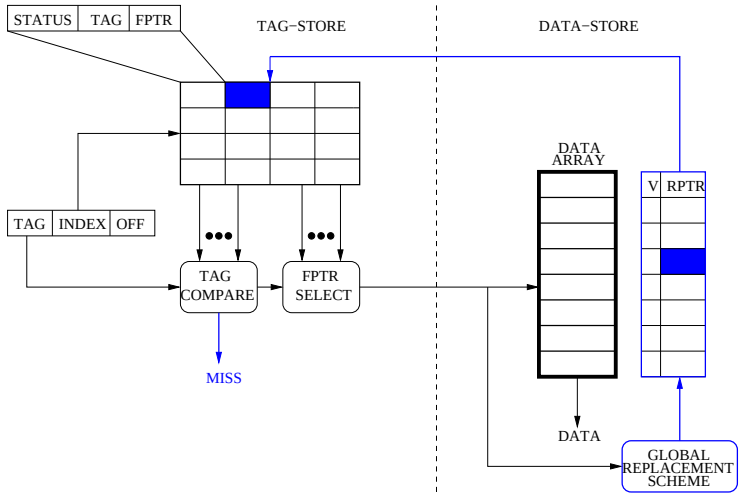
# The V-Way Cache



# The V-Way Cache

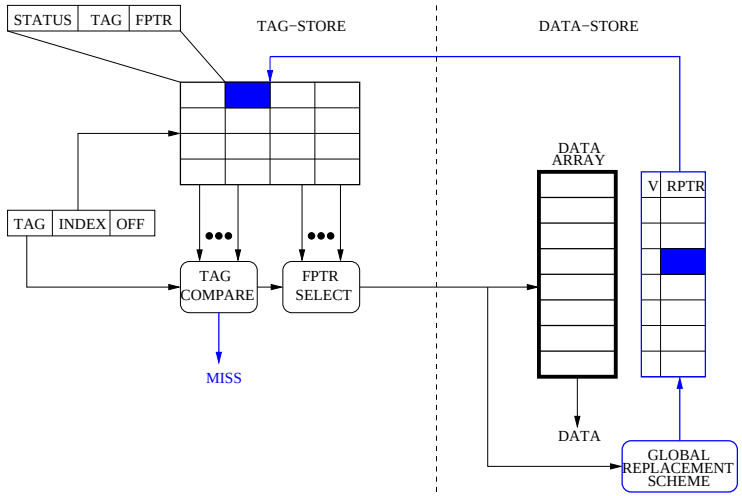


# The V-Way Cache



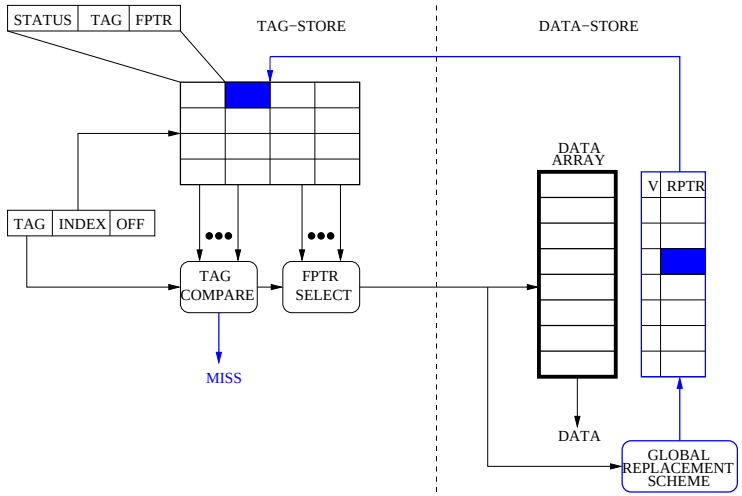

| Configuration | Tag Access | Data Replacement |
|---|---|---|
| Set-Associative | Fast | Local |
| Fully-Associative | | |
| V-Way | | |

# The V-Way Cache




| Configuration | Tag Access | Data Replacement |
|---|---|---|
| Set-Associative | Fast | Local |
| Fully-Associative | Slow | Global |
| V-Way | | |

# The V-Way Cache



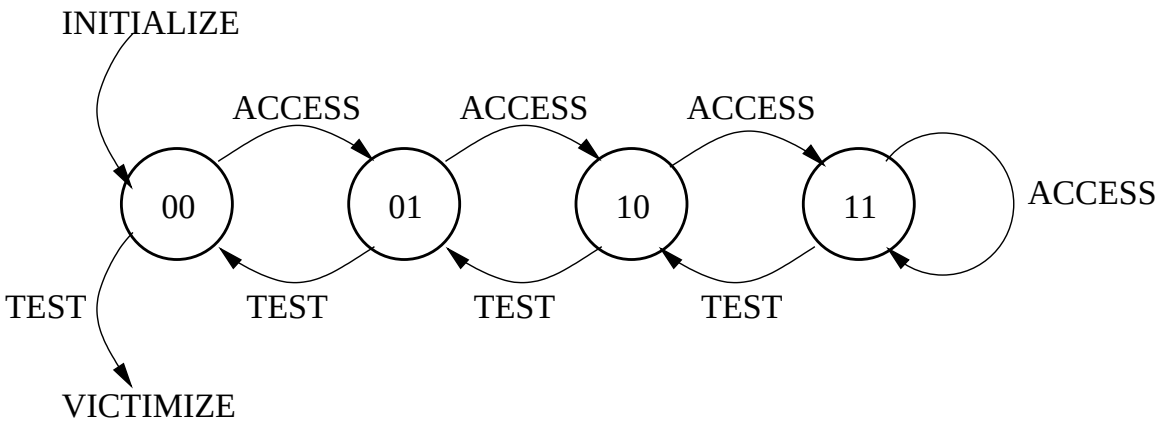

| Configuration | Tag Access | Data Replacement |
|---|---|---|
| Set-Associative | Fast | Local |
| Fully-Associative | Slow | Global |
| V-Way | Fast | Global |

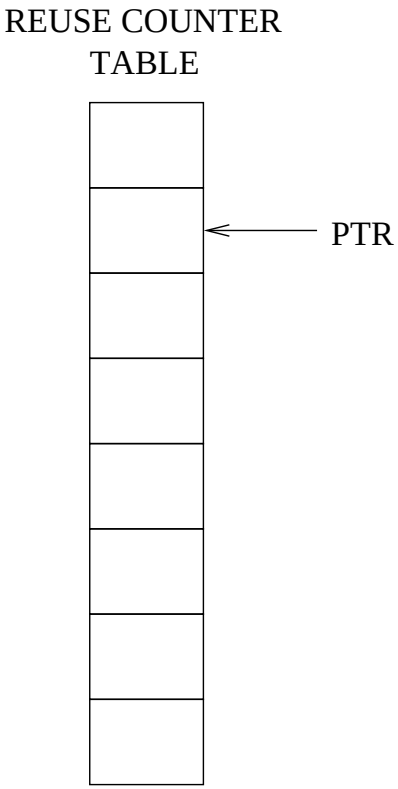
# A Practical Global Replacement Algorithm

- LRU is impractical because there are thousands of lines
- Second level cache access stream is a filtered version of the program access stream
- Reuse frequency of cache lines is very low. On average, more than 80% of the lines are reused less than four times
- We can track the reuse counts of a line with just two bits

We propose a frequency-based, clock-style, replacement scheme called *Reuse Replacement*

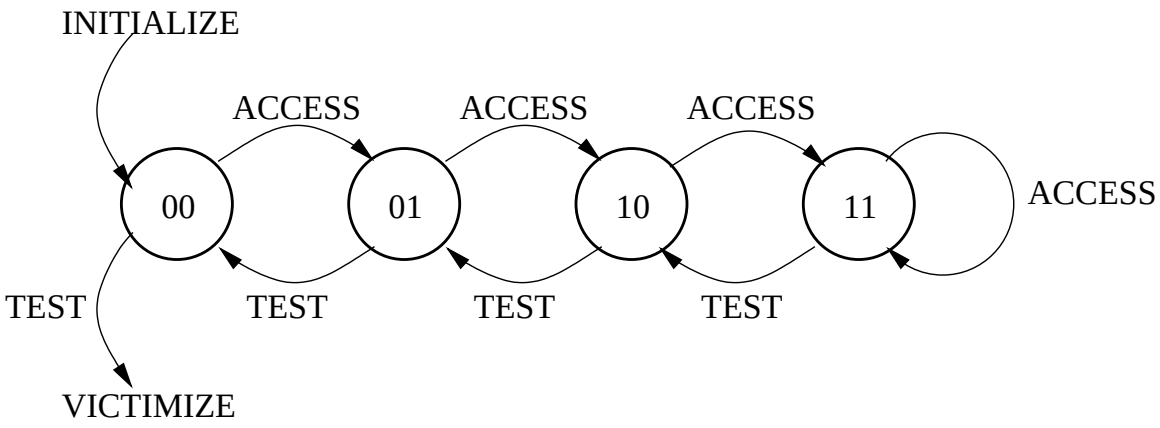
# Reuse Replacement

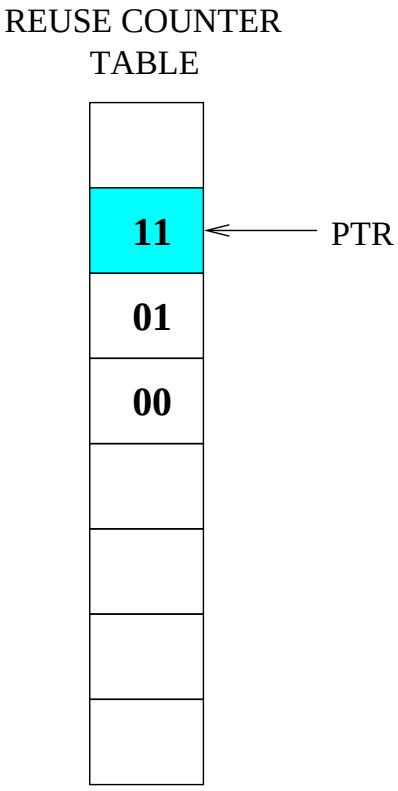



- Victim is the first “00” counter
- Decrement counter value if not victim
- Increment PTR after victim is found

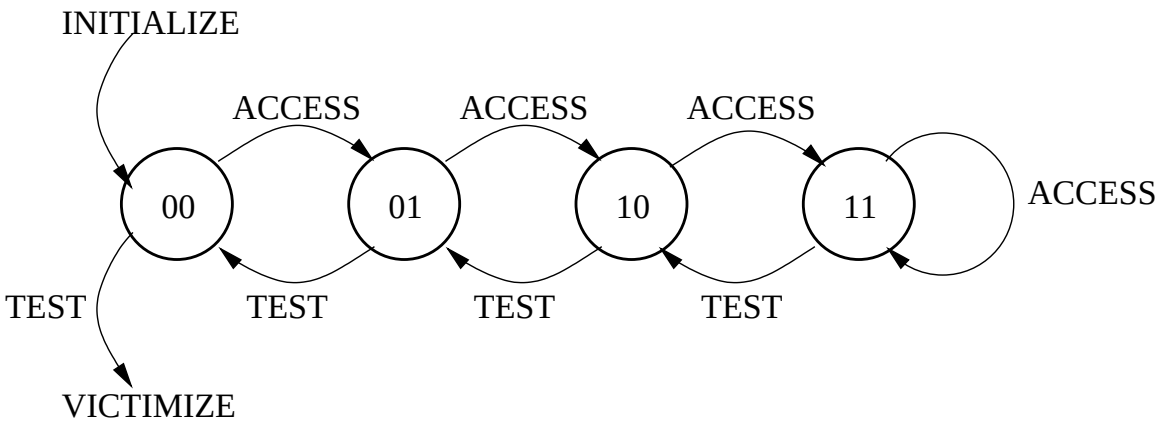
# Reuse Replacement

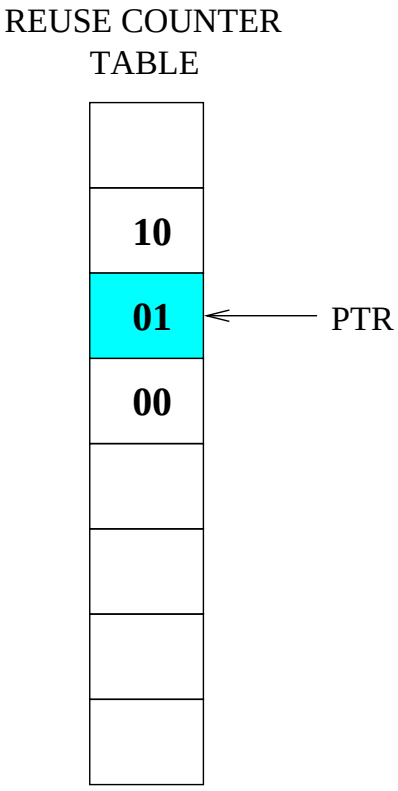



- Victim is the first “00” counter
- Decrement counter value if not victim
- Increment PTR after victim is found

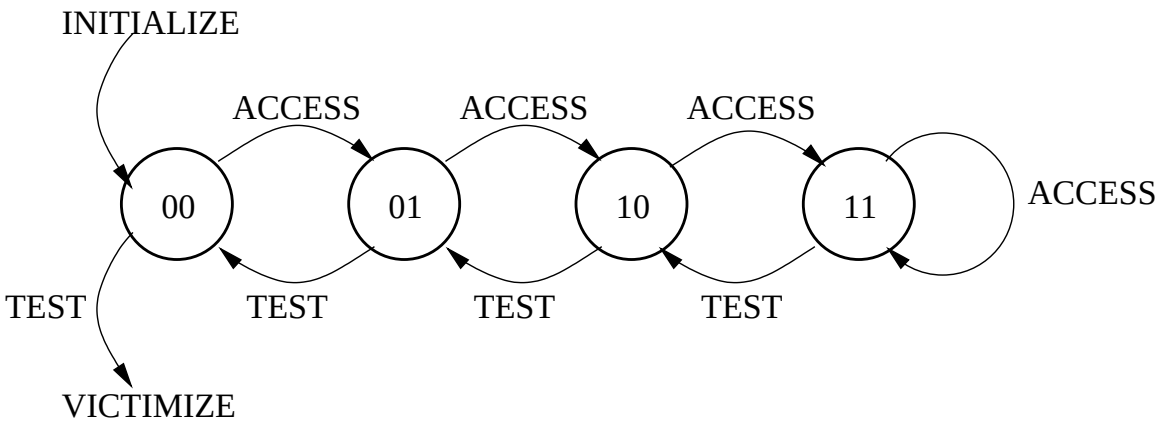
# Reuse Replacement

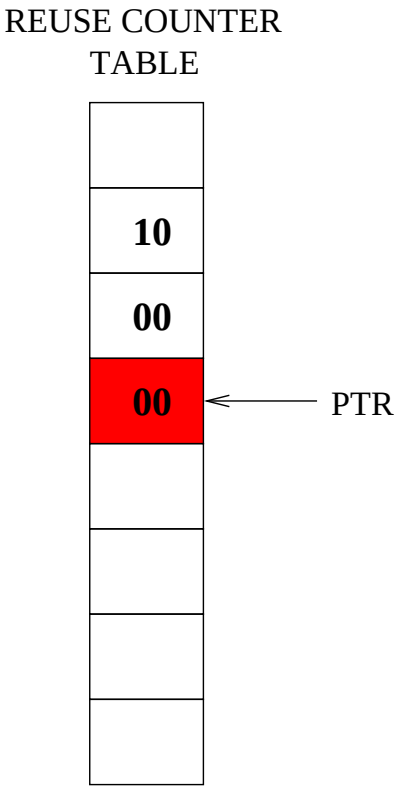



- Victim is the first “00” counter
- Decrement counter value if not victim
- Increment PTR after victim is found

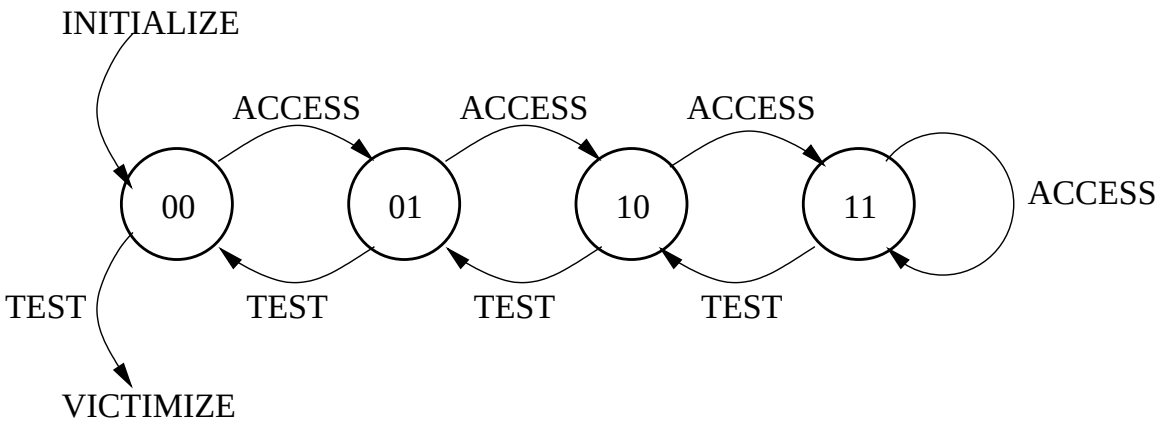
# Reuse Replacement

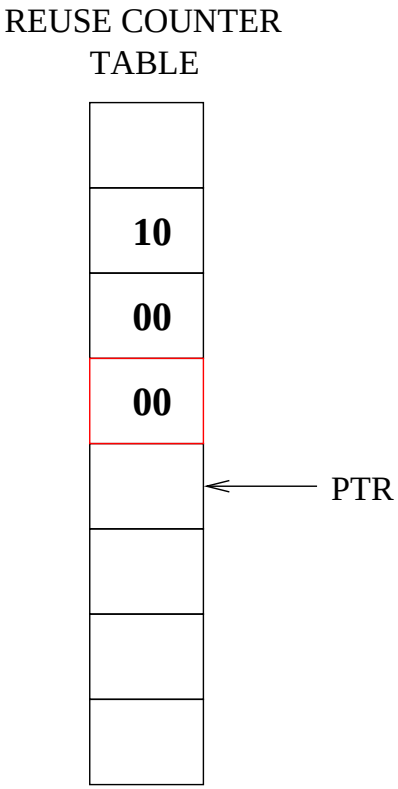



- Victim is the first “00” counter
- Decrement counter value if not victim
- Increment PTR after victim is found

# Reuse Replacement



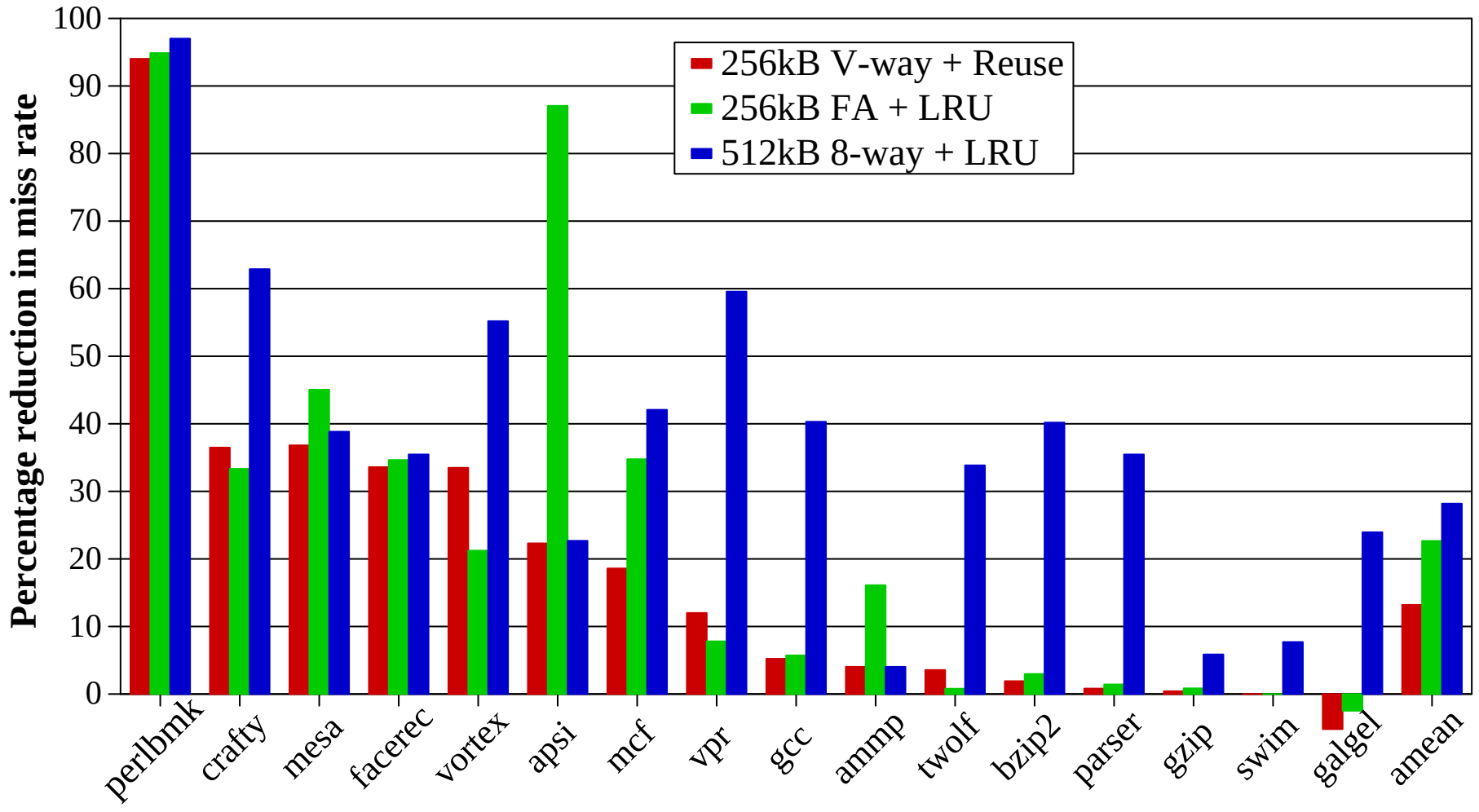

- Victim is the first “00” counter
- Decrement counter value if not victim
- Increment PTR after victim is found

# Victim Distance for Reuse Replacement

- Problem of variable replacement latency
  - Average victim distance: 3.9
  - Worst case victim distance: 1888
- Solution
  - Test eight counters each cycle
  - Limit search to five cycles

| Latency (in cycles) | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| Probability (victim) | 91.3% | 96.9% | 98.3% | 98.9% | 99.2% |

# Reduction in Misses with the V-Way Cache



# Storage, Latency, and Energy Cost

- Storage needed for extra tags, FPTR, RPTR, and Reuse bits

| Line-size | Miss-rate reduction | Increase in area |
|---|---|---|
| 128 B | 13.2% | 5.8% |

# Storage, Latency, and Energy Cost

- Storage needed for extra tags, FPTR, RPTR, and Reuse bits

| Line-size | Miss-rate reduction | Increase in area |
|---|---|---|
| 128 B | 13.2% | 5.8% |

- Delay due to more tags and FPTR selection: 130 ps

# Storage, Latency, and Energy Cost

- Storage needed for extra tags, FPTR, RPTR, and Reuse bits

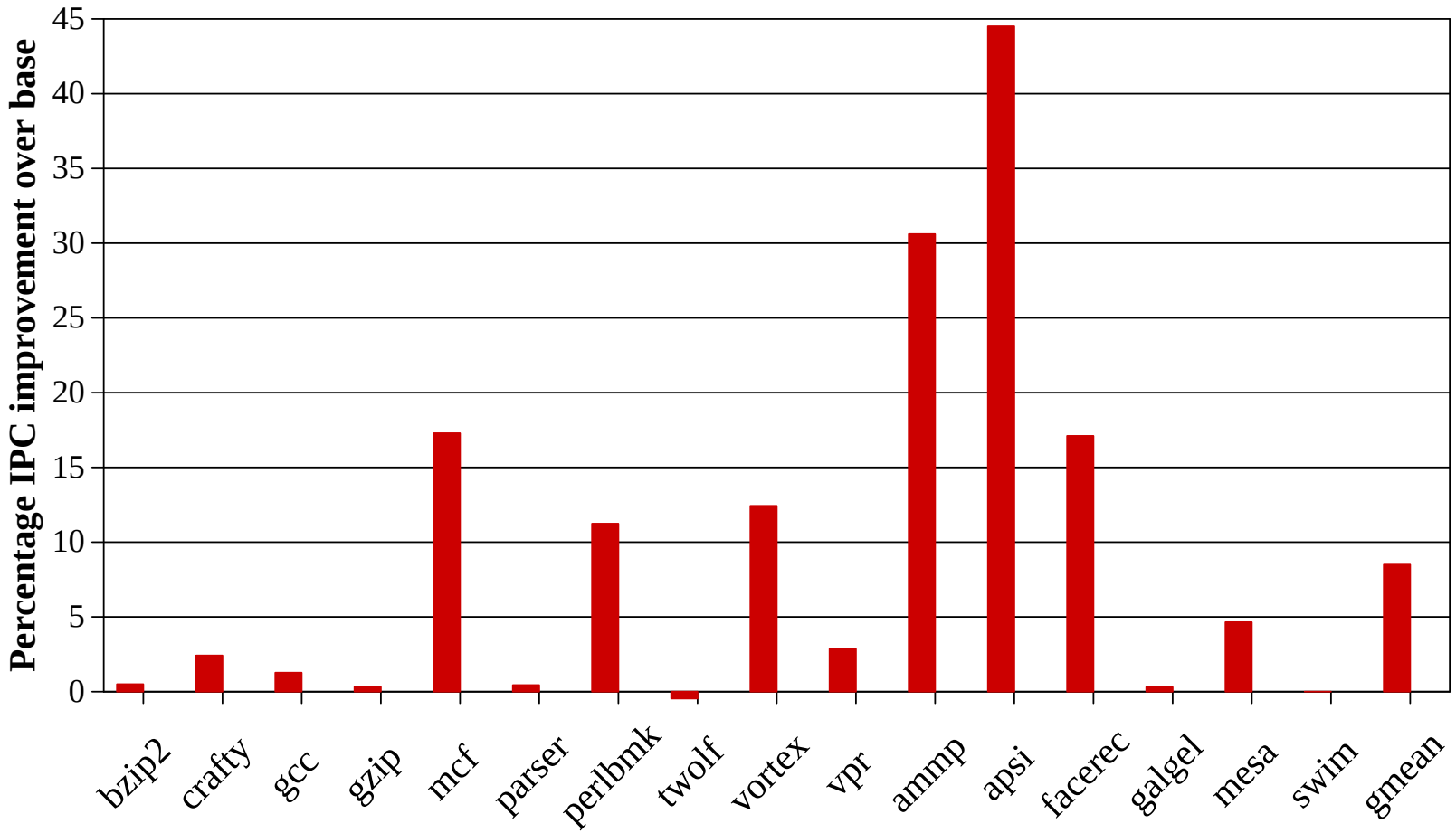
| Line-size | Miss-rate reduction | Increase in area |
|---|---|---|
| 128 B | 13.2% | 5.8% |

- Delay due to more tags and FPTR selection: 130 ps

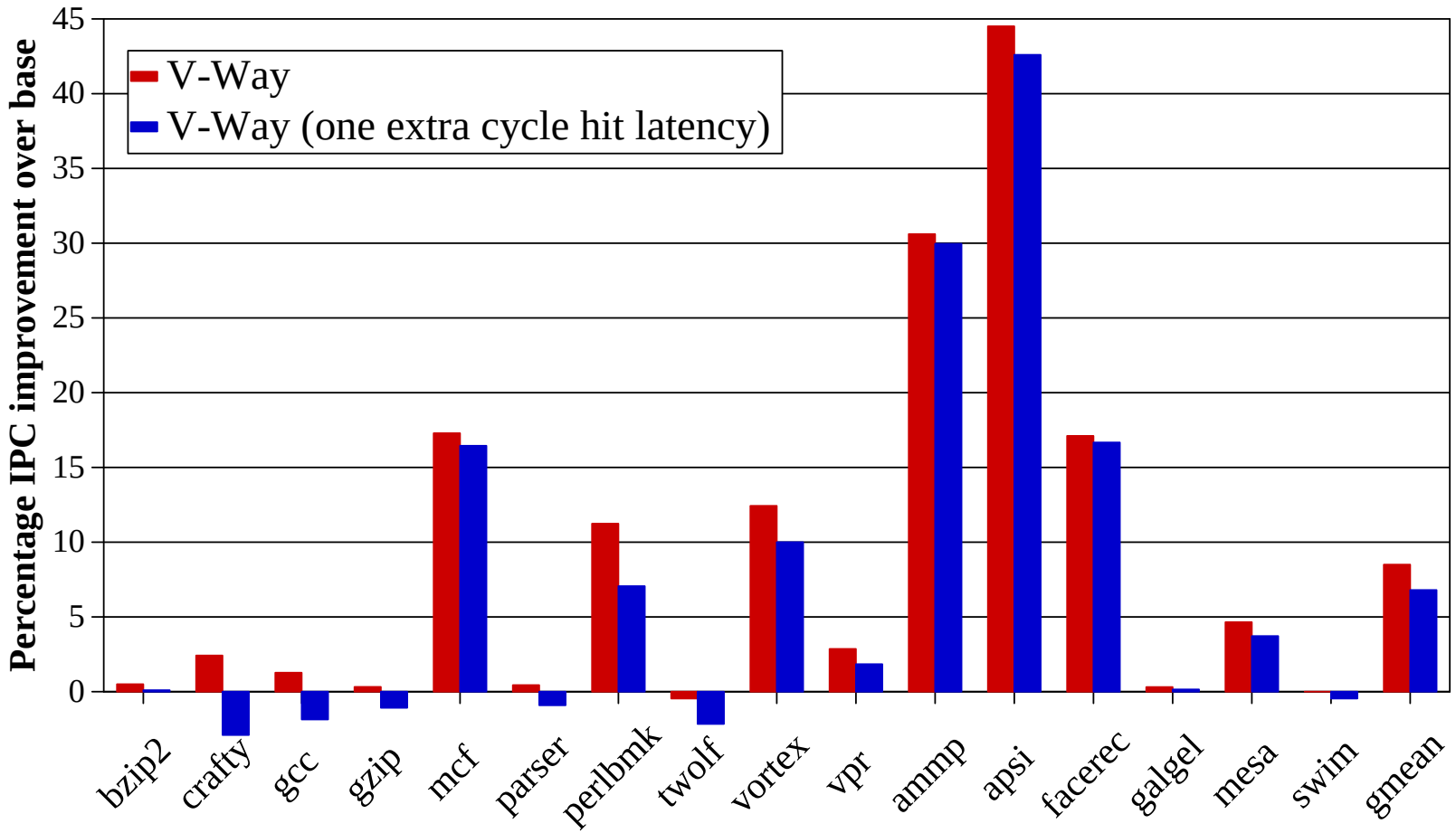
- Energy in accessing bigger tag-store

| Parallel lookup | Baseline | V-Way |
|---|---|---|
| 1.02nJ | 0.35nJ | 0.40nJ |

# Impact on IPC



# Impact on IPC



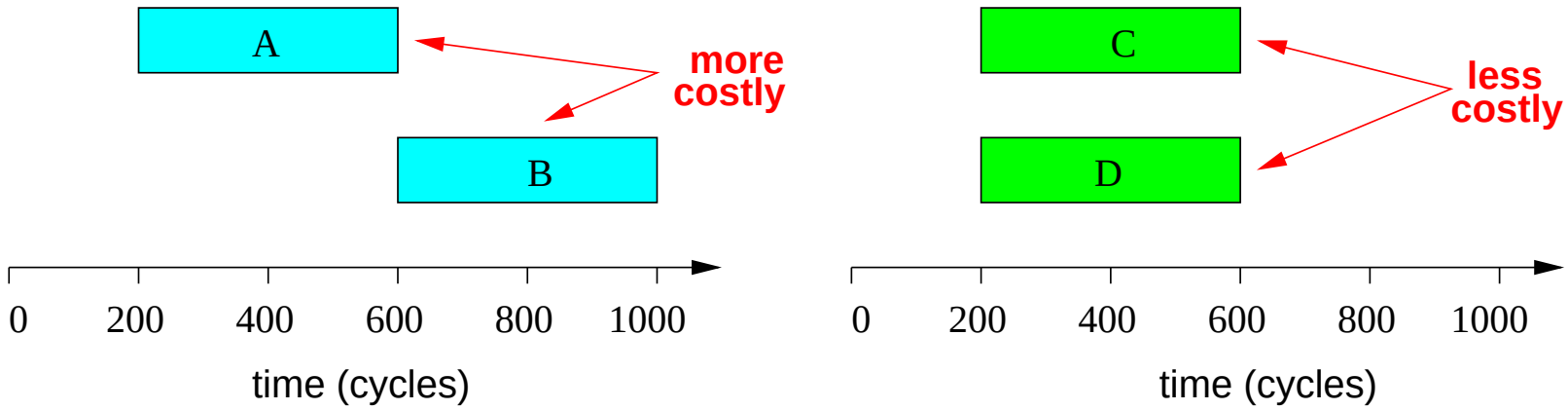
# Outline

- Introduction
- The V-Way Cache
- MLP-Aware Cache Replacement
- Summary

# Memory Level Parallelism (MLP)

- Performance loss due to L2 misses can be reduced by servicing the misses in parallel
- The notion of servicing multiple misses in parallel is termed Memory Level Parallelism (MLP) [Glew WACI'98]
- Several techniques to improve MLP
  - Made possible by non-blocking caches [Kroft ISCA'81]
  - Out-of-order processing [Choi+ ISCA'04]
  - Runahead execution [Mutlu+ HPCA'03]
  - Read miss clustering at compile time [Pai+ MICRO'99]

# MLP is Not Uniform for All Misses




- MLP is not uniform across all misses
- Isolated misses are more costly than parallel misses
- Current replacement schemes assume uniform cost for all lines
- MLP-aware replacement can improve performance. Need a model to quantify the MLP-based cost of each miss.

# Algorithm to Compute MLP-Based Cost

- MLP-based cost depends on:
  - The number of parallel misses (more misses $\Rightarrow$ less costly)
  - The amount of overlap (more overlap $\Rightarrow$ less costly)

# Algorithm to Compute MLP-Based Cost

- MLP-based cost depends on:
  - The number of parallel misses (more misses $\Rightarrow$ less costly)
  - The amount of overlap (more overlap $\Rightarrow$ less costly)

- MSHR keeps track of all in-flight misses

# Algorithm to Compute MLP-Based Cost

- MLP-based cost depends on:
  - The number of parallel misses (more misses $\Rightarrow$ less costly)
  - The amount of overlap (more overlap $\Rightarrow$ less costly)

- MSHR keeps track of all in-flight misses

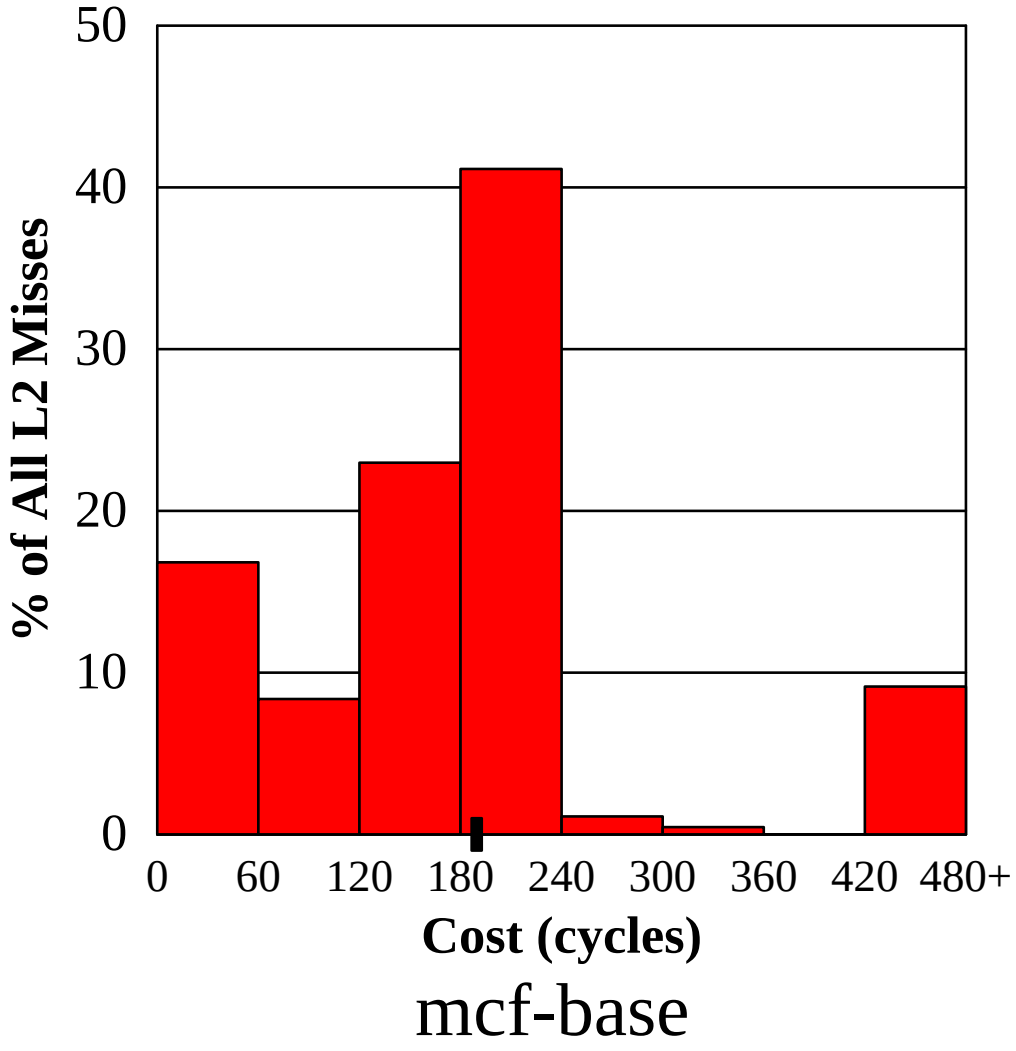
- Add a field *mlp_cost* to each MSHR entry

```
update_mlp_cost()    /* gets called every cycle */
begin:
   N ⇐ Number of outstanding demand misses in MSHR
   for each demand miss in the MSHR
     miss.mlp_cost+ = (1/N)
end
```

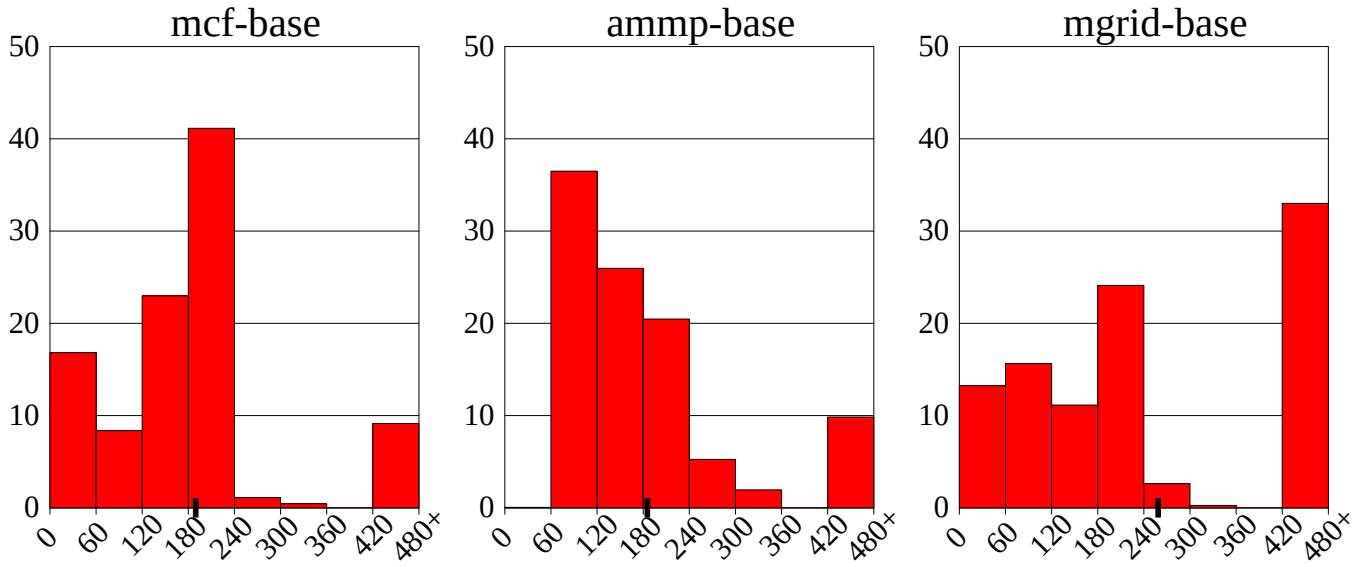
# Machine Configuration

- Pipeline: Out-of-order, 8-wide, 128-entry window
- First level caches: 16kB, 2-way, 64B linesize
- Second level cache: 1MB, 16-way, 64B linesize
- Memory latency: 400 cycles
- Bus latency: 40 cycles (without contention)

# Distribution of MLP-Based Cost



# Distribution of MLP-Based Cost

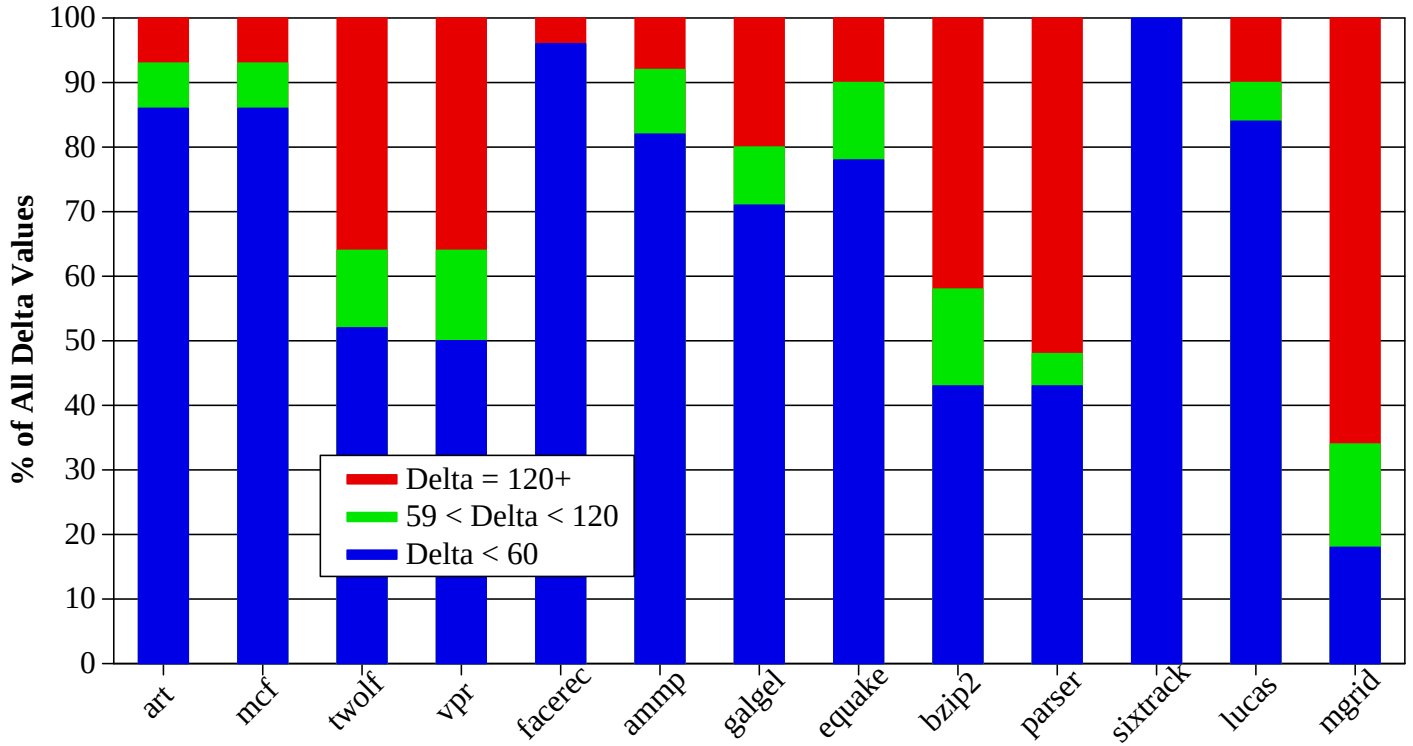


# Repeatability of MLP-Based Cost

- A miss that was serviced in isolation can be serviced in parallel the next time
- Given the current cost we need to estimate the future cost
- How stable is the cost for consecutive misses for a cache line?

- Let *delta* be the absolute difference between the cost for consecutive misses to a given line
- Lower values of delta denotes that the similar cost repeats
- Higher values of delta means cost varies significantly

# Repeatability of MLP-Based Cost

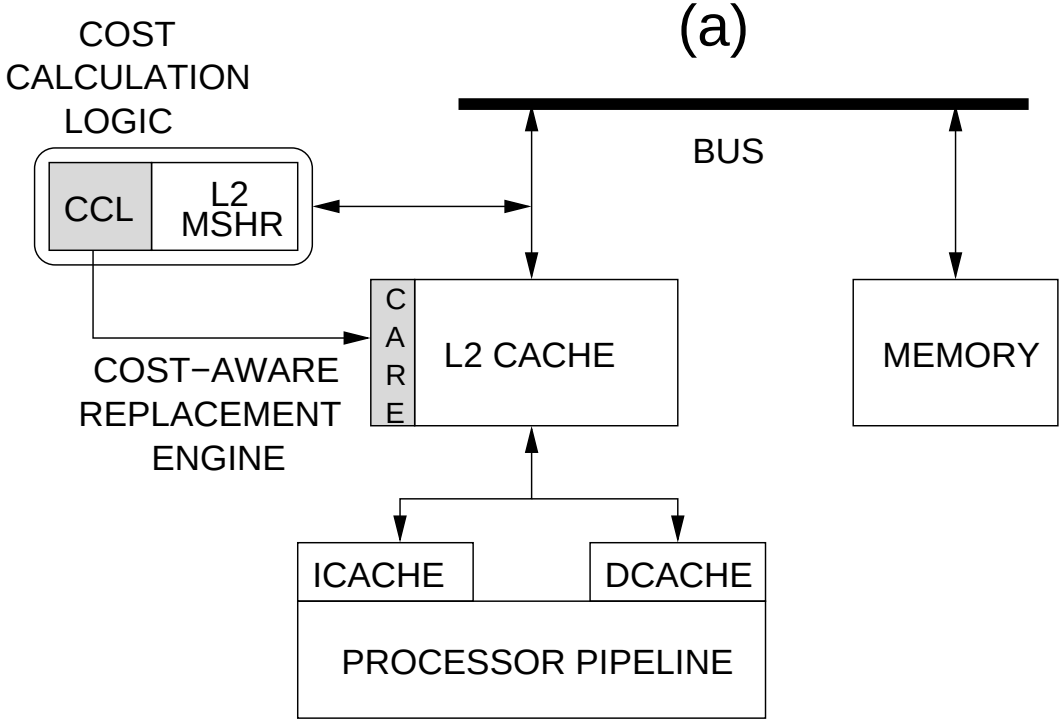



For most benchmarks, the delta value is small indicating that we can use our cost metric for cost-sensitive cache replacement

However, the high value of delta for parser and mgrid indicate that we will need an adaptive mechanism to revert back

# Design of MLP-Aware Replacement Scheme

(a)

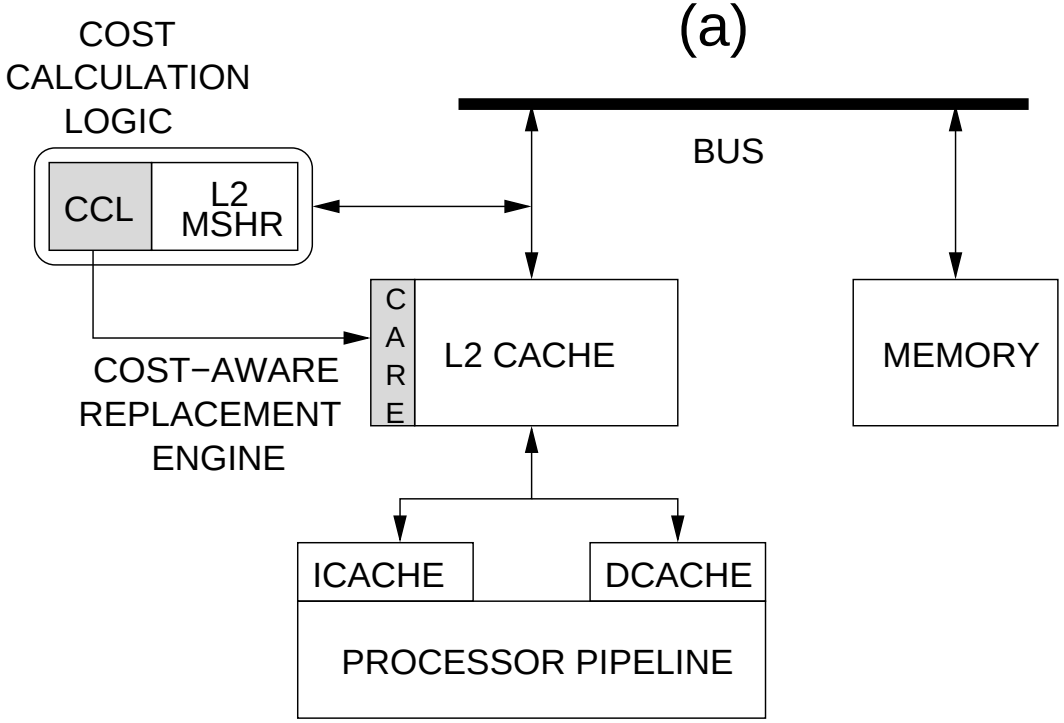



(b)

| Computed value of MLP−Based Cost | Quantized value |
|---|---|
| 0 to 59 cycles | 0 |
| 60 to 119 cycles | 1 |
| 120 to 179 cycles | 2 |
| 180 to 239 cycles | 3 |
| 240 to 299 cycles | 4 |
| 300 to 359 cycles | 5 |
| 360 to 419 cycles | 6 |
| 420+ cycles | 7 |

# Design of MLP-Aware Replacement Scheme

(a)



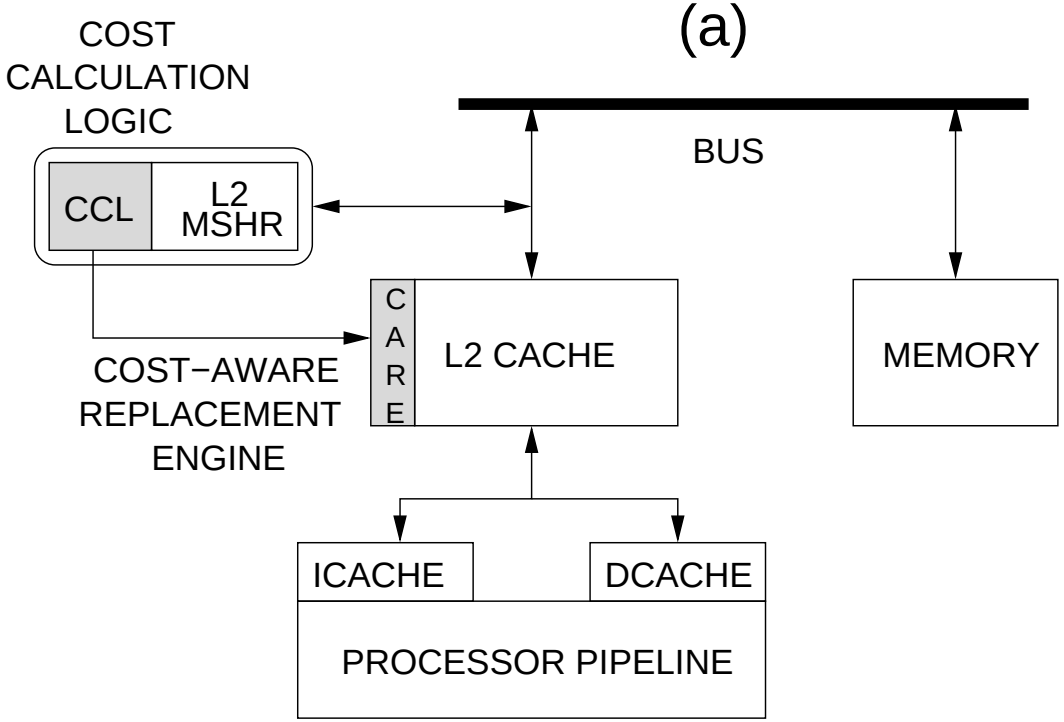

(b)

| Computed value of MLP−Based Cost | Quantized value |
|---|---|
| 0 to 59 cycles | 0 |
| 60 to 119 cycles | 1 |
| 120 to 179 cycles | 2 |
| 180 to 239 cycles | 3 |
| 240 to 299 cycles | 4 |
| 300 to 359 cycles | 5 |
| 360 to 419 cycles | 6 |
| 420+ cycles | 7 |

Need to include both recency as well as cost information in replacement decisions.

# Design of MLP-Aware Replacement Scheme

(a)



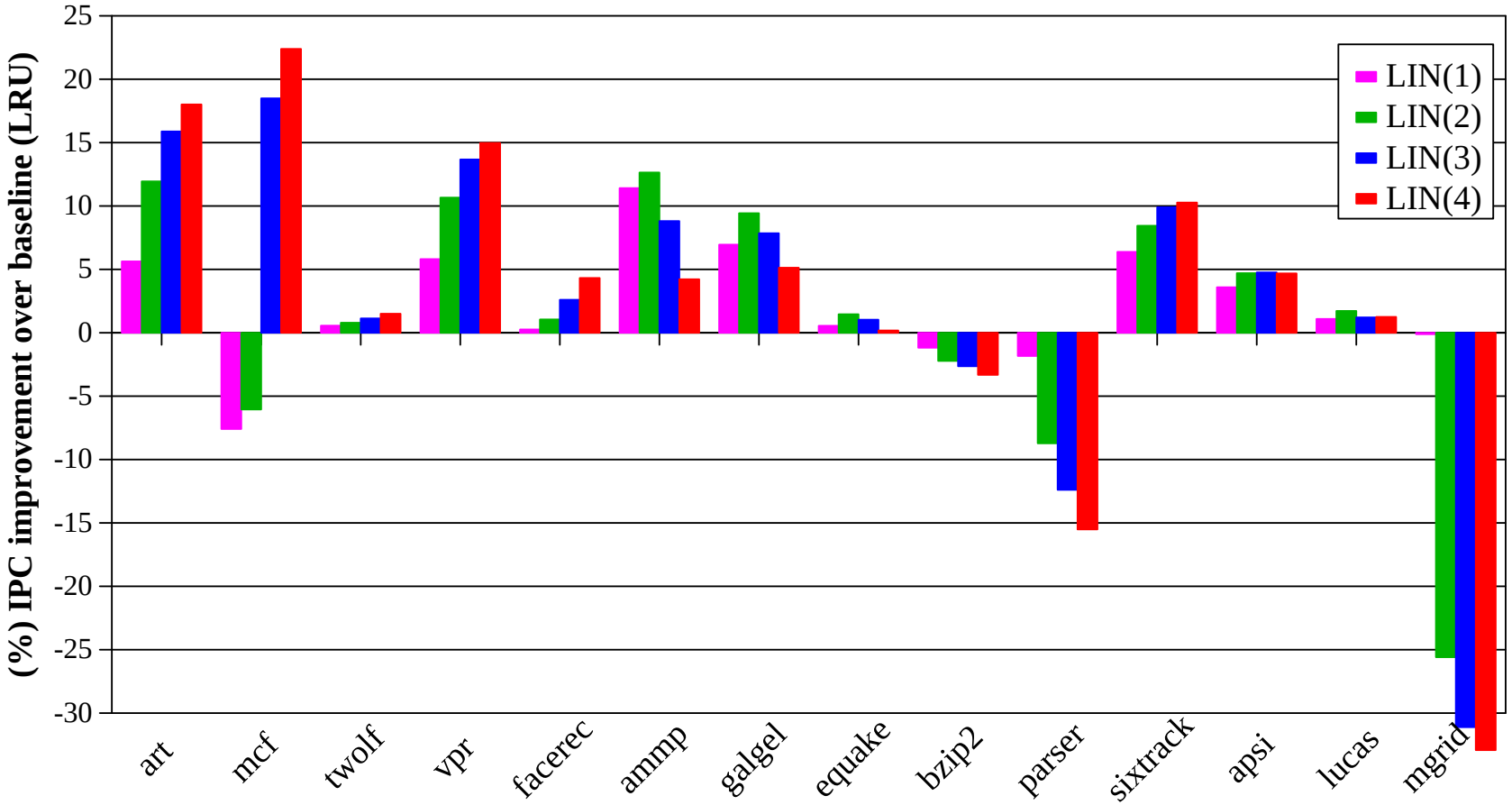

(b)

| Computed value of MLP−Based Cost | Quantized value |
|---|---|
| 0 to 59 cycles | 0 |
| 60 to 119 cycles | 1 |
| 120 to 179 cycles | 2 |
| 180 to 239 cycles | 3 |
| 240 to 299 cycles | 4 |
| 300 to 359 cycles | 5 |
| 360 to 419 cycles | 6 |
| 420+ cycles | 7 |

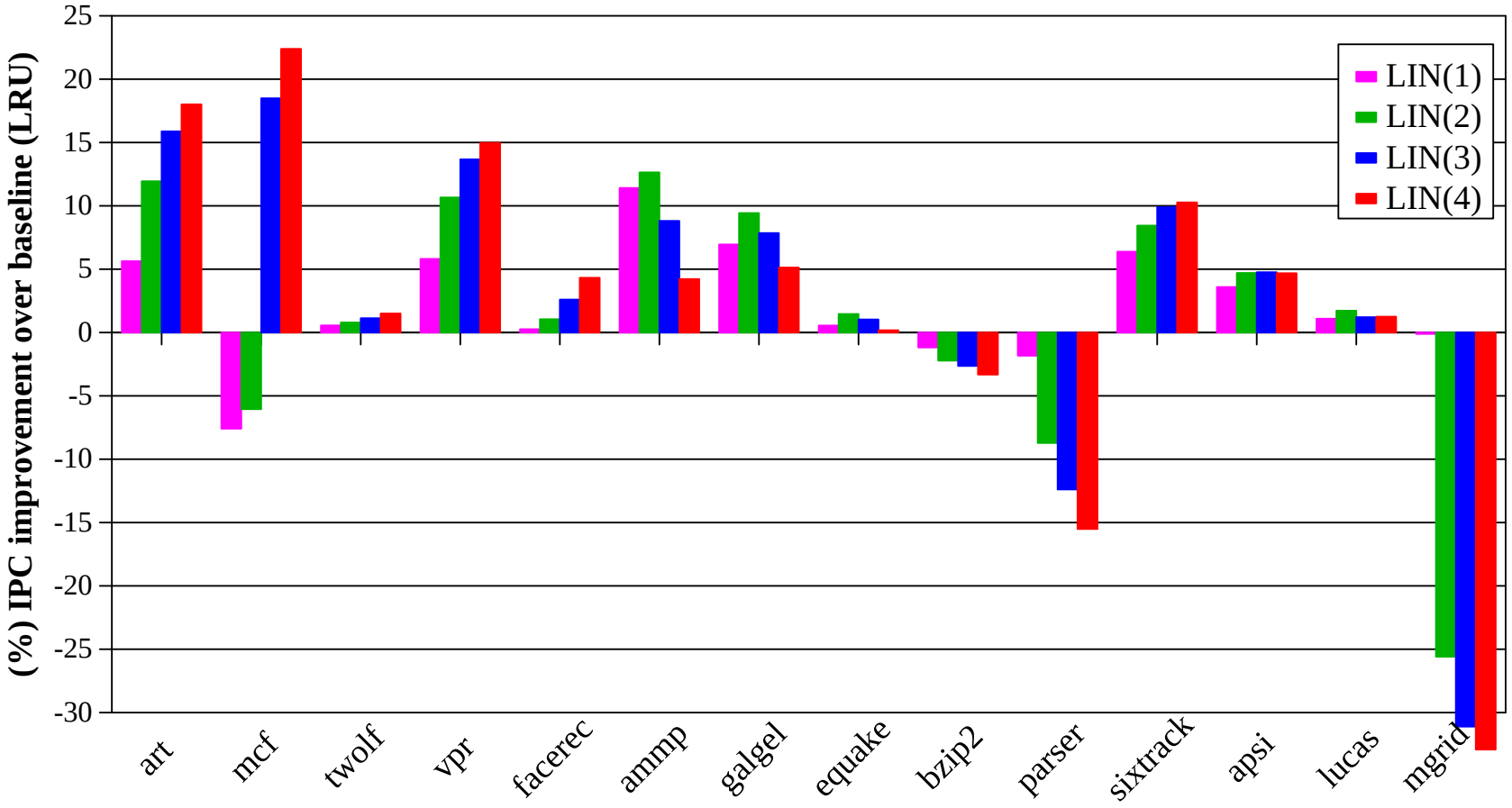
Need to include both recency as well as cost information in replacement decisions. Proposed a linear (LIN) function

$$Victim = min\{Recency(i) + \lambda * cost(i)\}$$

# Results for the LIN Policy

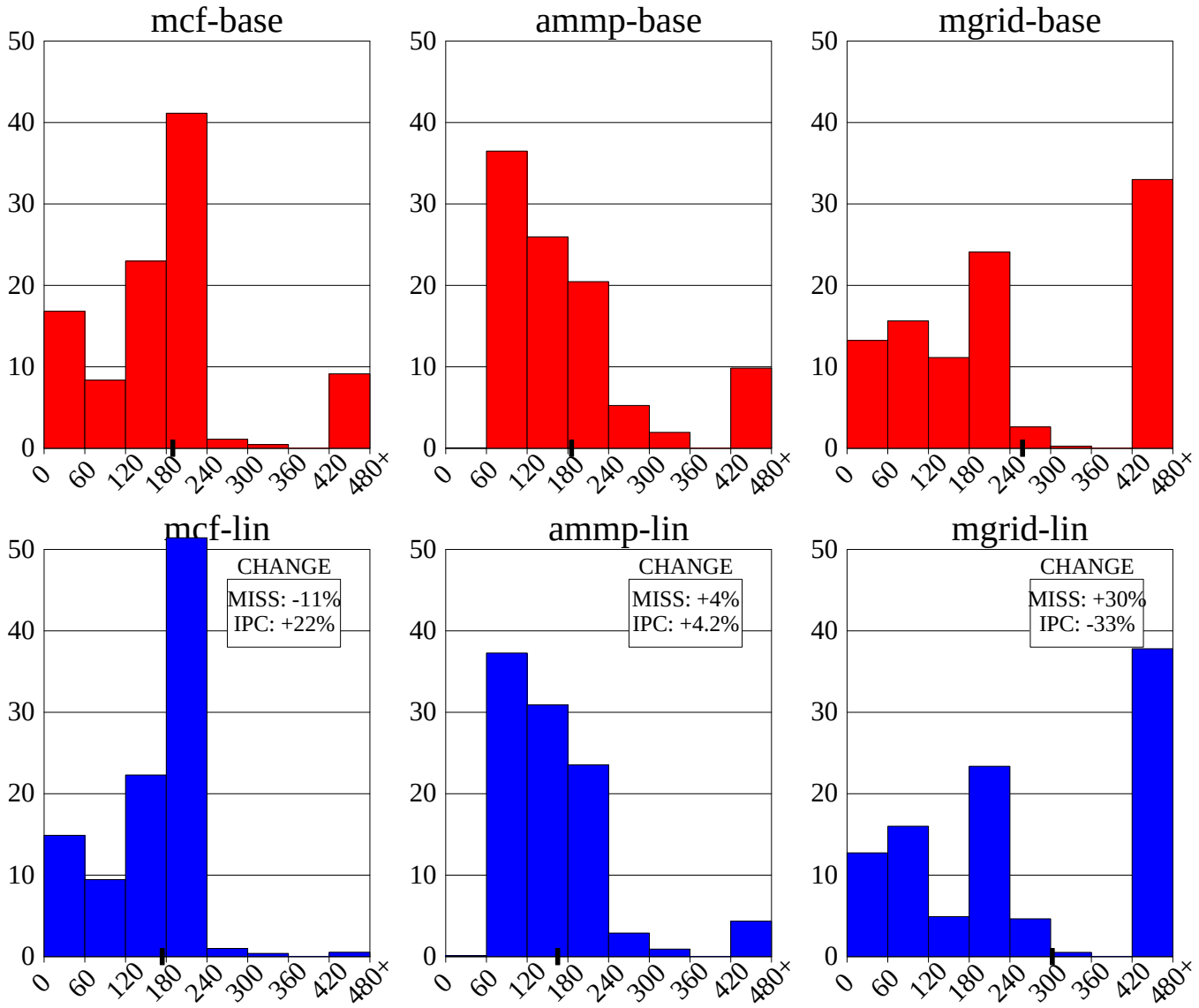


# Results for the LIN Policy

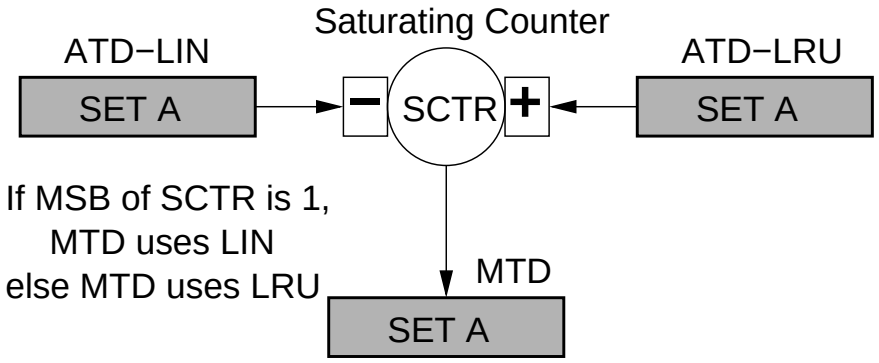



Decisions based on MLP cost hurts performance on parser and mgrid because of the high delta values. Need a mechanism to turn on LIN only if it is likely to benefit.

# Cost Distribution for the LIN policy

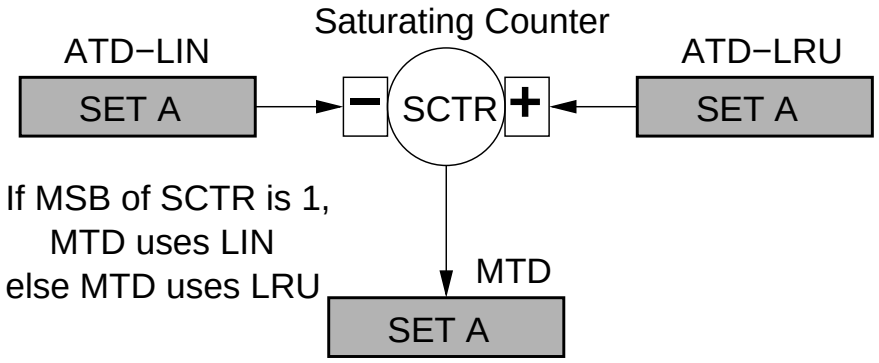


# Tournament Selection of Replacement Policy




| ATD−LIN | ATD−LRU | Action |
|---|---|---|
| HIT | HIT | SCTR unchanged |
| MISS | MISS | SCTR unchanged |
| MISS | HIT | Decrement SCTR by cost$_q$ of Miss in ATD−LIN |
| HIT | MISS | Increment SCTR by cost$_q$ of Miss in ATD−LRU |

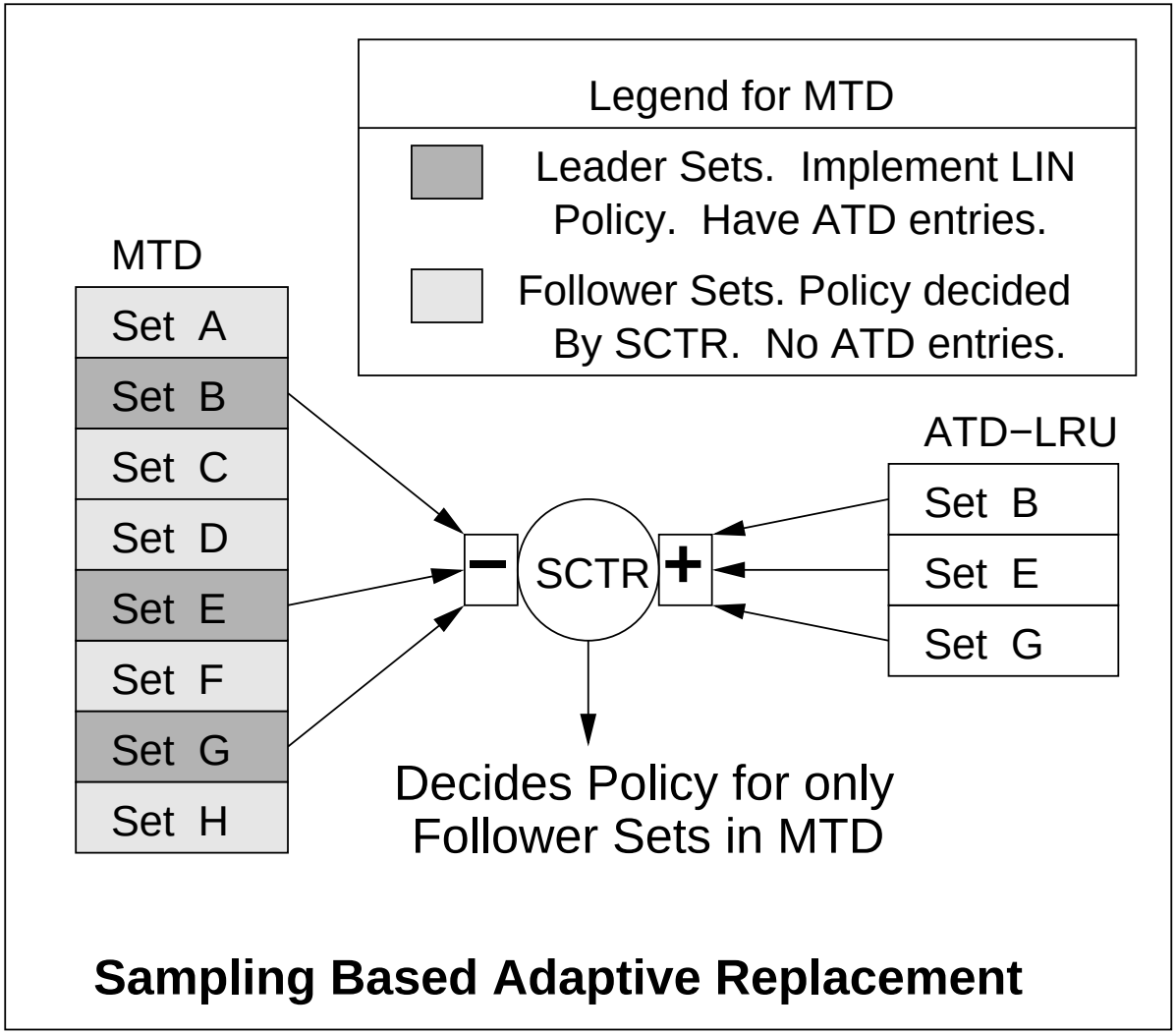
# Tournament Selection of Replacement Policy



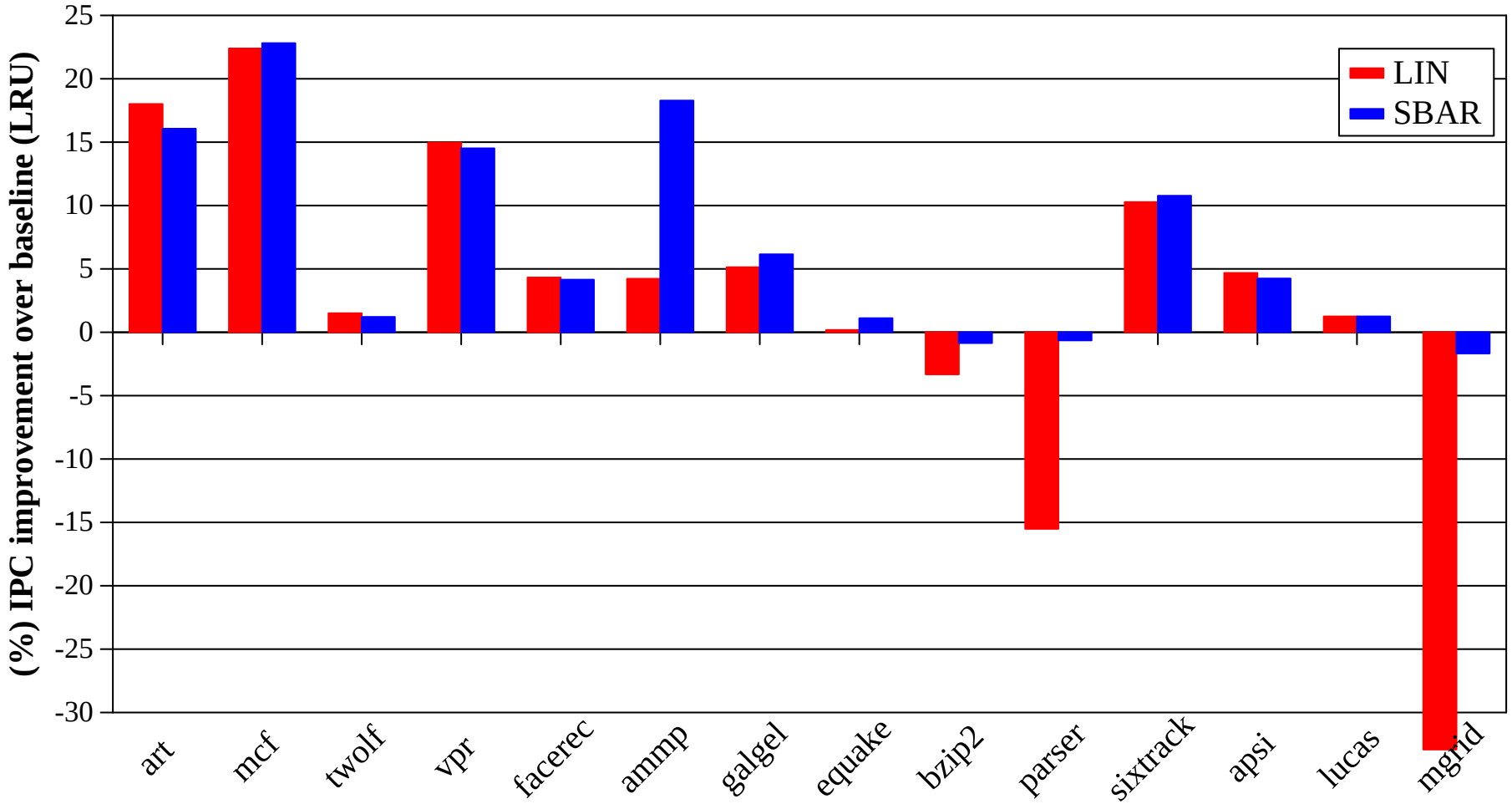

| ATD−LIN | ATD−LRU | Action |
|---|---|---|
| HIT | HIT | SCTR unchanged |
| MISS | MISS | SCTR unchanged |
| MISS | HIT | Decrement SCTR by cost$_q$ of Miss in ATD−LIN |
| HIT | MISS | Increment SCTR by cost$_q$ of Miss in ATD−LRU |

- Tournament selection is expensive in terms of hardware cost if implemented on a per-set basis.
- We can make the decision based on a few sampled-set and use this decision globally for the entire cache.

# Sampling Based Adaptive Replacement



# Results for SBAR



# Outline

- Problem Description
- The V-Way Cache
- MLP-Aware Cache Replacement
- Summary

# Summary

- What is the problem?
  - Traditional designs do not use the L2 cache efficiently
  - This leads to increased L2 misses and reduced performance

# Summary

- What is the problem?
  - Traditional designs do not use the L2 cache efficiently
  - This leads to increased L2 misses and reduced performance
- The V-Way cache can lower miss rate by allowing global replacement

# Summary

- What is the problem?
  - Traditional designs do not use the L2 cache efficiently
  - This leads to increased L2 misses and reduced performance
- The V-Way cache can lower miss rate by allowing global replacement
- MLP-Aware replacement can improve performance by reducing the number of costly misses.

# Summary

- What is the problem?
  - Traditional designs do not use the L2 cache efficiently
  - This leads to increased L2 misses and reduced performance
- The V-Way cache can lower miss rate by allowing global replacement
- MLP-Aware replacement can improve performance by reducing the number of costly misses.
- For more information:
  - *"The V-Way Cache: Demand-Based Associativity via Global Replacement"* by Moinuddin K. Qureshi, David Thompson, and Yale N. Patt, ISCA 2005. (http://www.ece.utexas.edu/-qk/papers/vway.pdf)
  - *"A Case for MLP-Aware Cache Replacement"* by Moinuddin K. Qureshi, Daniel N. Lynch, Onur Mutlu, and Yale N. Patt, ISCA 2006. (http://www.ece.utexas.edu/-qk/papers/mlp.pdf)