Thread Clustering:

Sharing-Aware Thread Scheduling on SMP-CMP-SMT Multiprocessors

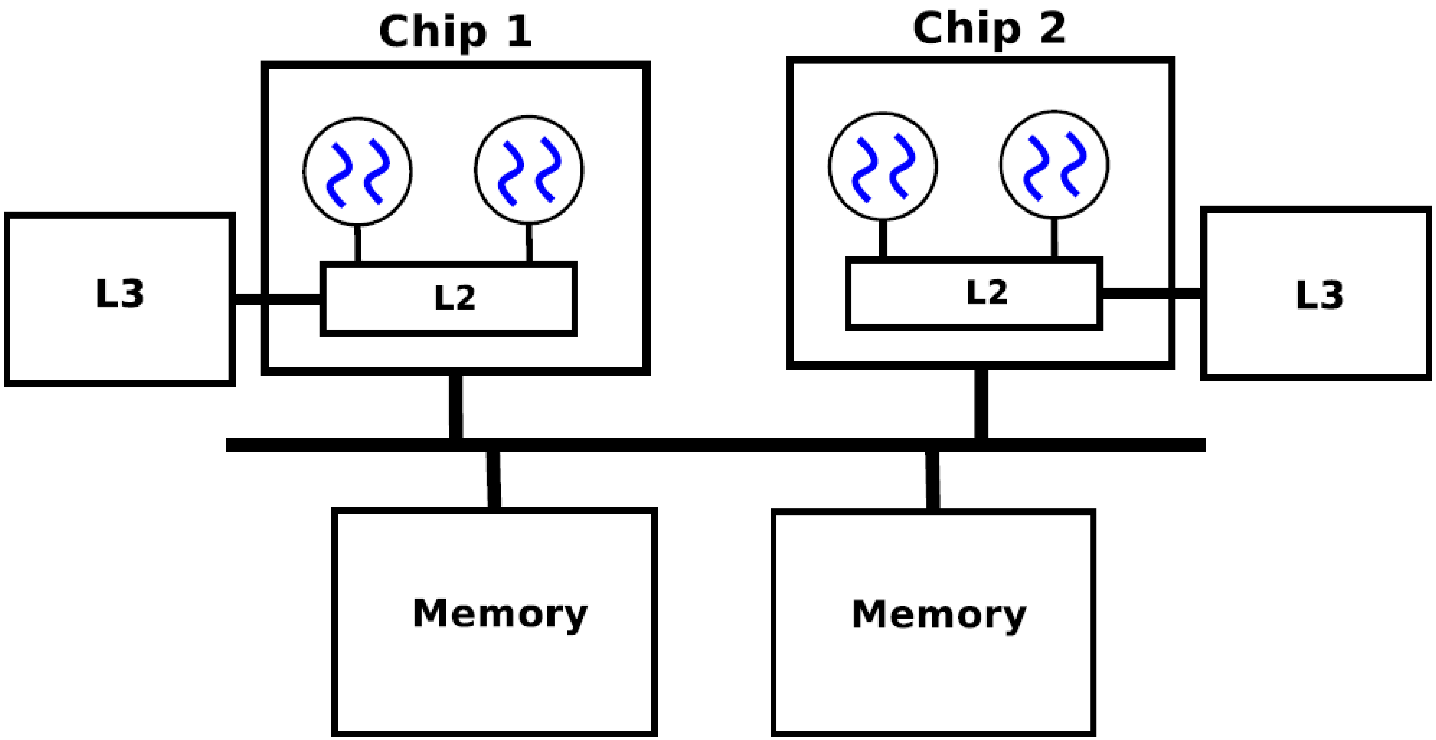
David Tam, Reza Azimi, Michael Stumm

University of Toronto

{tamda, azimi, stumm}@eecg.toronto.edu

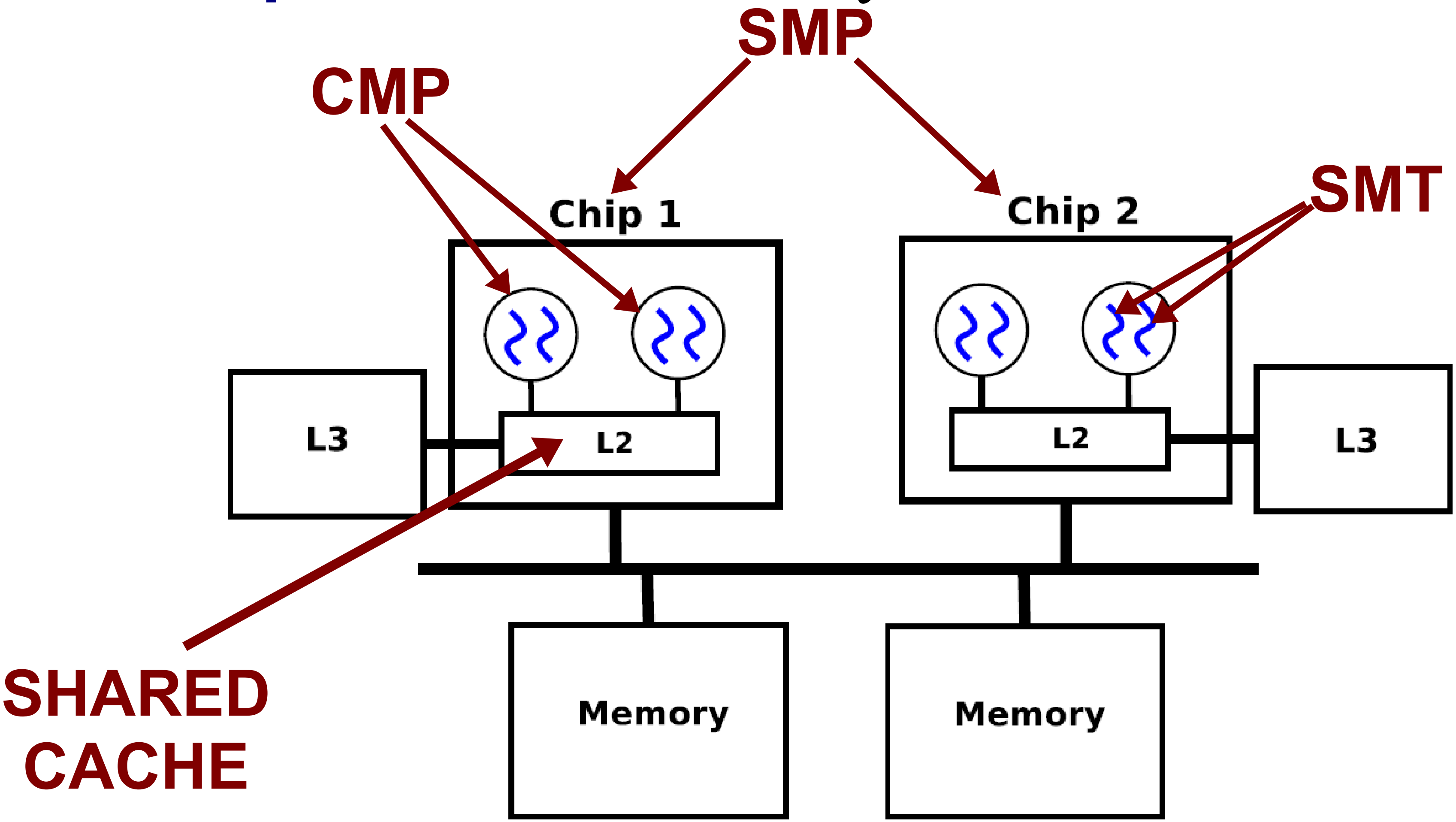
Multiprocessors Today

Example: IBM Power 5 system



Multiprocessors Today

Example: IBM Power 5 system


SMP

CMP

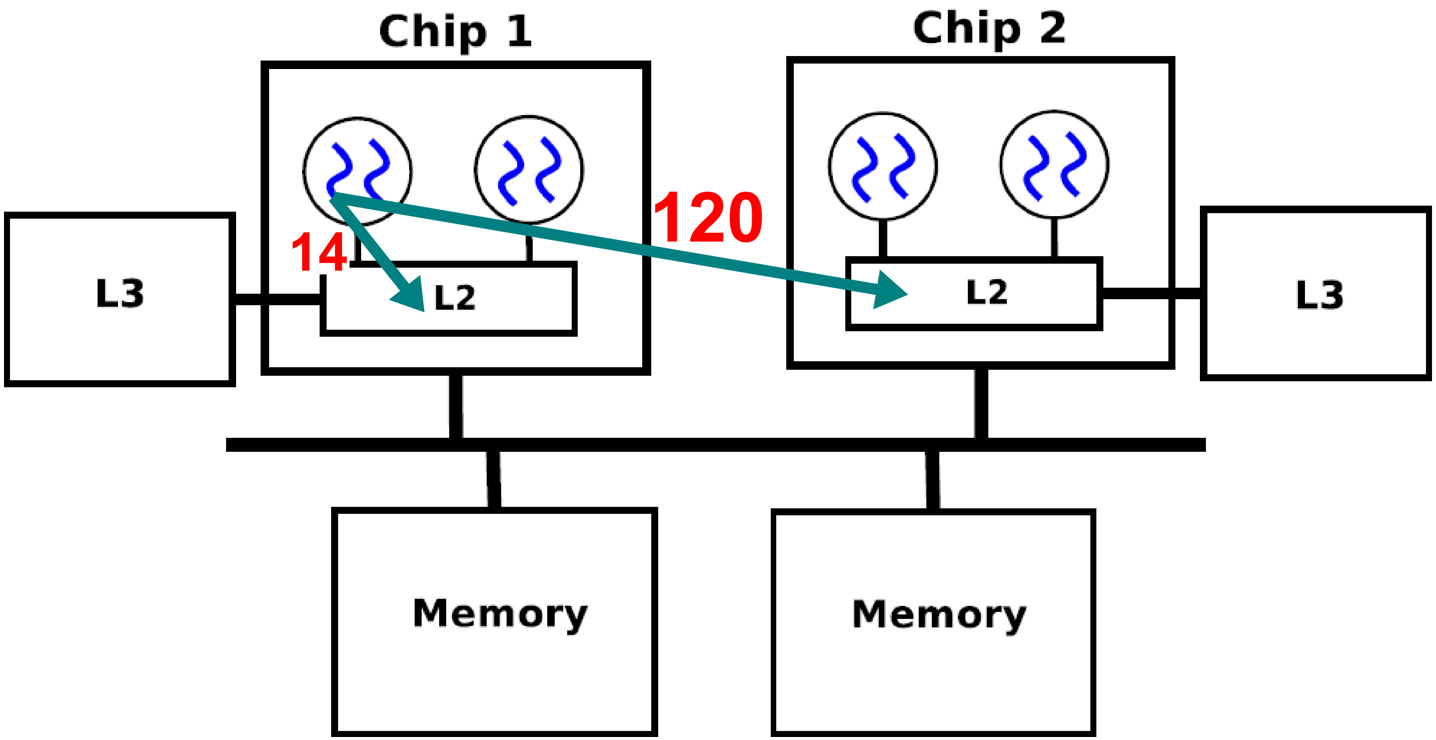
SMT

SHARED CACHE

Multiprocessors Today

Example: IBM Power 5 system

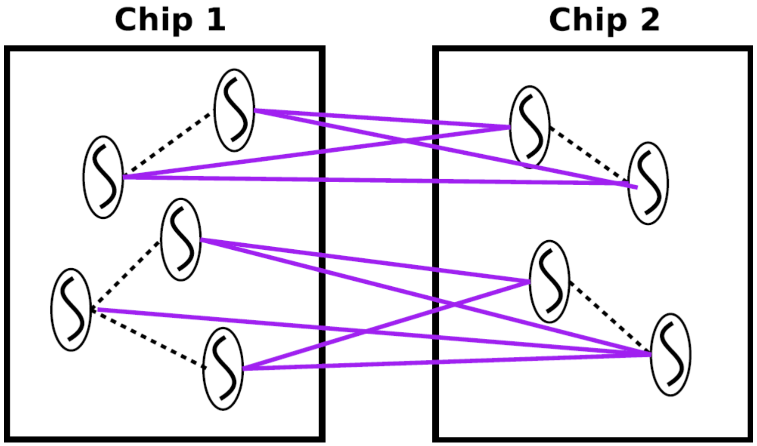
Disparity in L2 latencies



Operating Systems Today

CPU Schedulers:

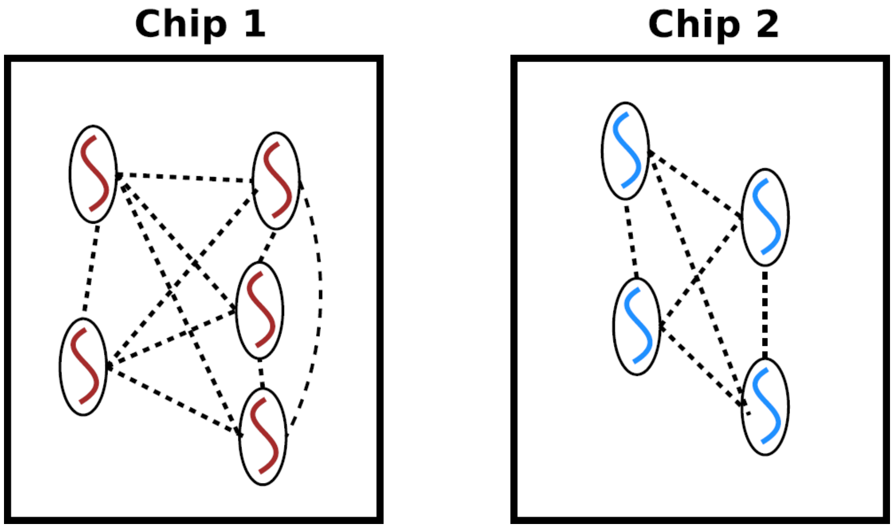
- Ignore disparity in L2 latencies
- Ignore data sharing among threads
  - Distribute threads poorly
- **Cross-chip traffic**
  - **Remote L2 cache accesses**



- **Causes performance problem**

Our Goal: Sharing-Aware Scheduling

- Detect sharing patterns
- Cluster threads



Benefits:

- Decrease cross-chip traffic
- Increase on-chip cache locality
- Exploit shared L2 caches

Our Online Technique

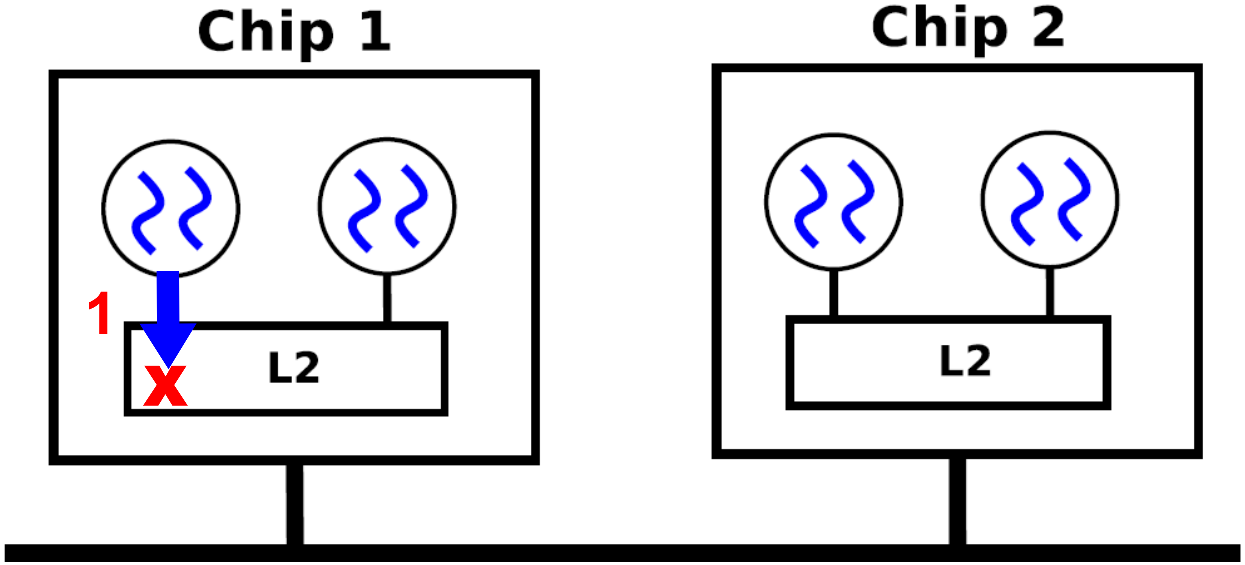
STEPS:

REPEAT

1) Monitor remote cache access rate
2) Detect thread sharing patterns
3) Determine thread clusters
4) Migrate thread clusters

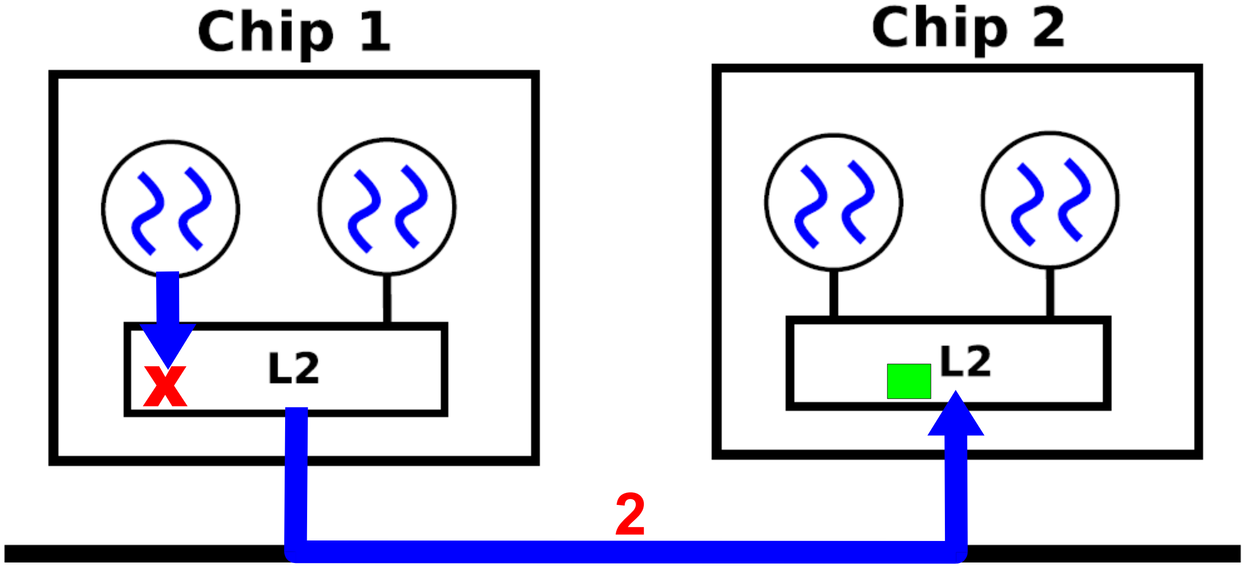
Sharing Detection

- To observe remote cache accesses:
  - Exploit **HPCs** (hardware performance counters)
  - Sample *remote cache miss* addresses
    - Local cache misses satisfied by remote cache
    - IBM Power 5 *continuous data sampling*



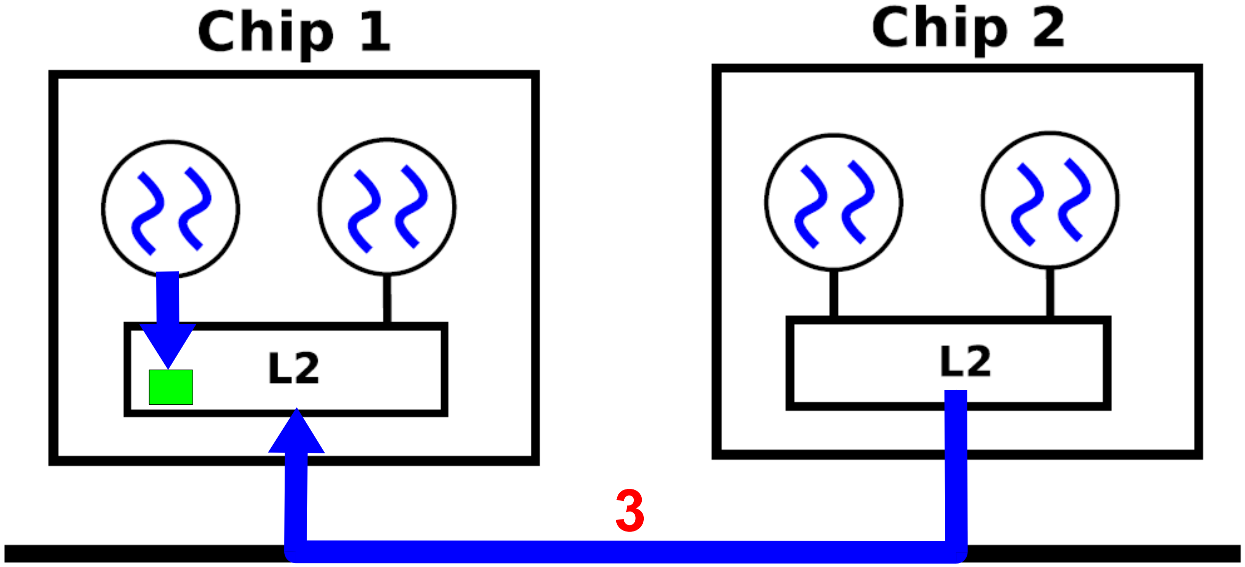
Sharing Detection

- To observe remote cache accesses:
  - Exploit **HPCs** (hardware performance counters)
  - Sample *remote cache miss* addresses
    - Local cache misses satisfied by remote cache
    - IBM Power 5 *continuous data sampling*



Sharing Detection

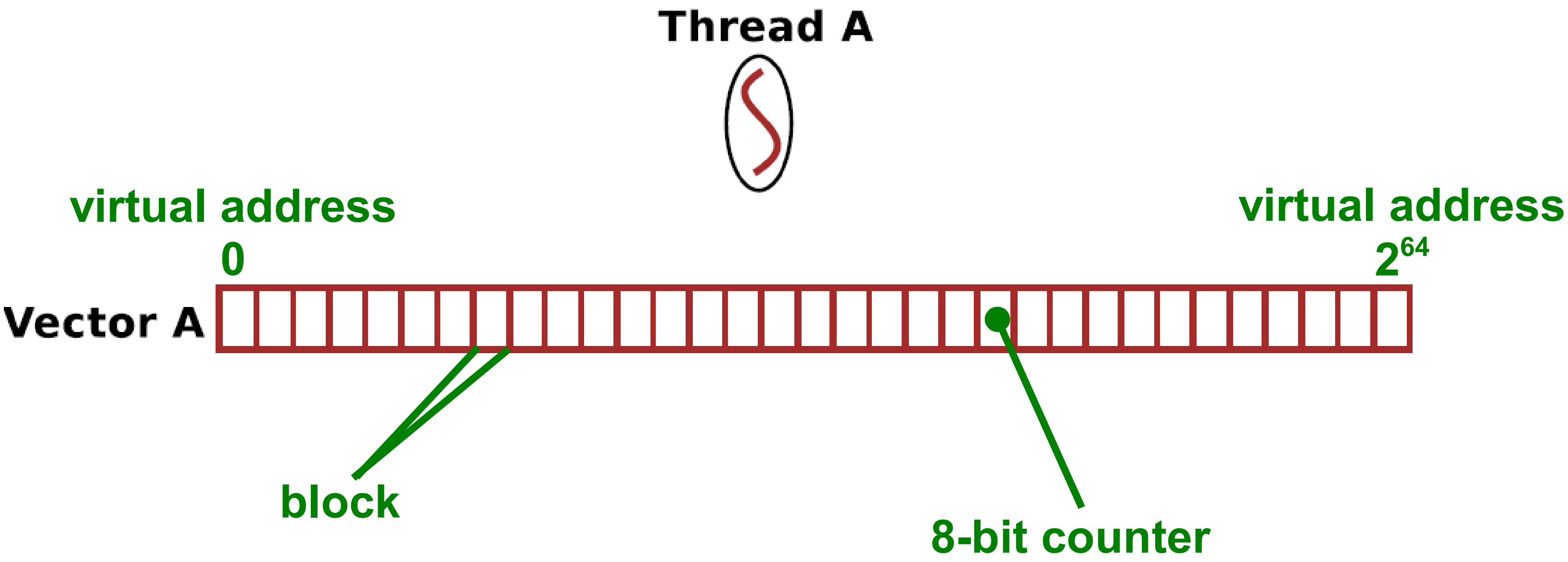
- To observe remote cache accesses:
  - Exploit **HPC**s (hardware performance counters)
  - Sample *remote cache miss* addresses
    - Local cache misses satisfied by remote cache
    - IBM Power 5 *continuous data sampling*



Sharing Signatures

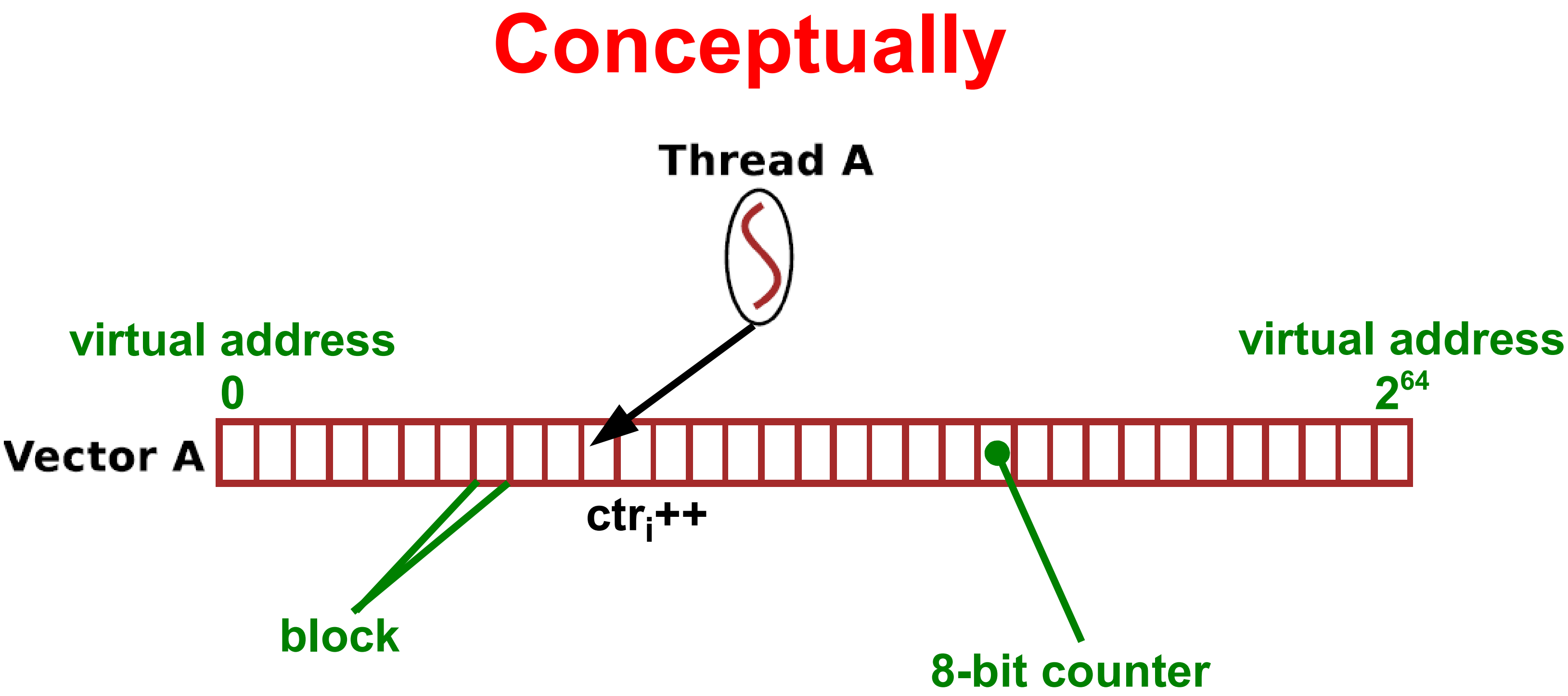
- Construct for each thread
  - Counts remote cache accesses

Conceptually



Sharing Signatures

- Construct for each thread
  - Counts remote cache accesses

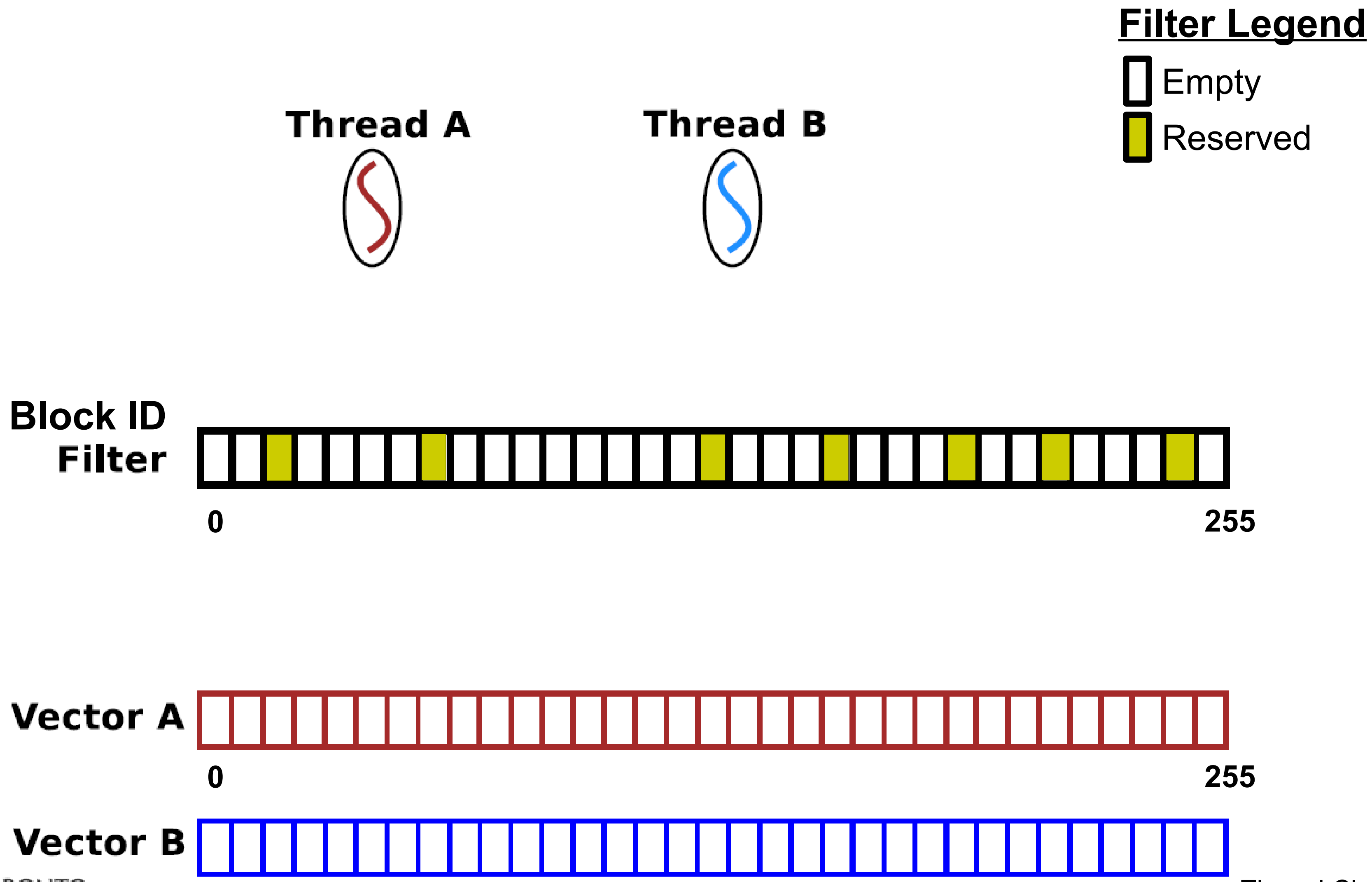


Optimizations

- **CPU:** Temporal Sampling
  - Sample every Nth remote cache access
- **Memory:** Spatial Sampling
  - 256-entry vector
  - Hash function
  - Block ID filter
- Vectors still effective at indicating sharing

Spatial Sampling

- Hash collision & alias removal



Filter Legend

□ Empty

🟨 Reserved

Thread A

Thread B

Block ID Filter

0

255

Vector A

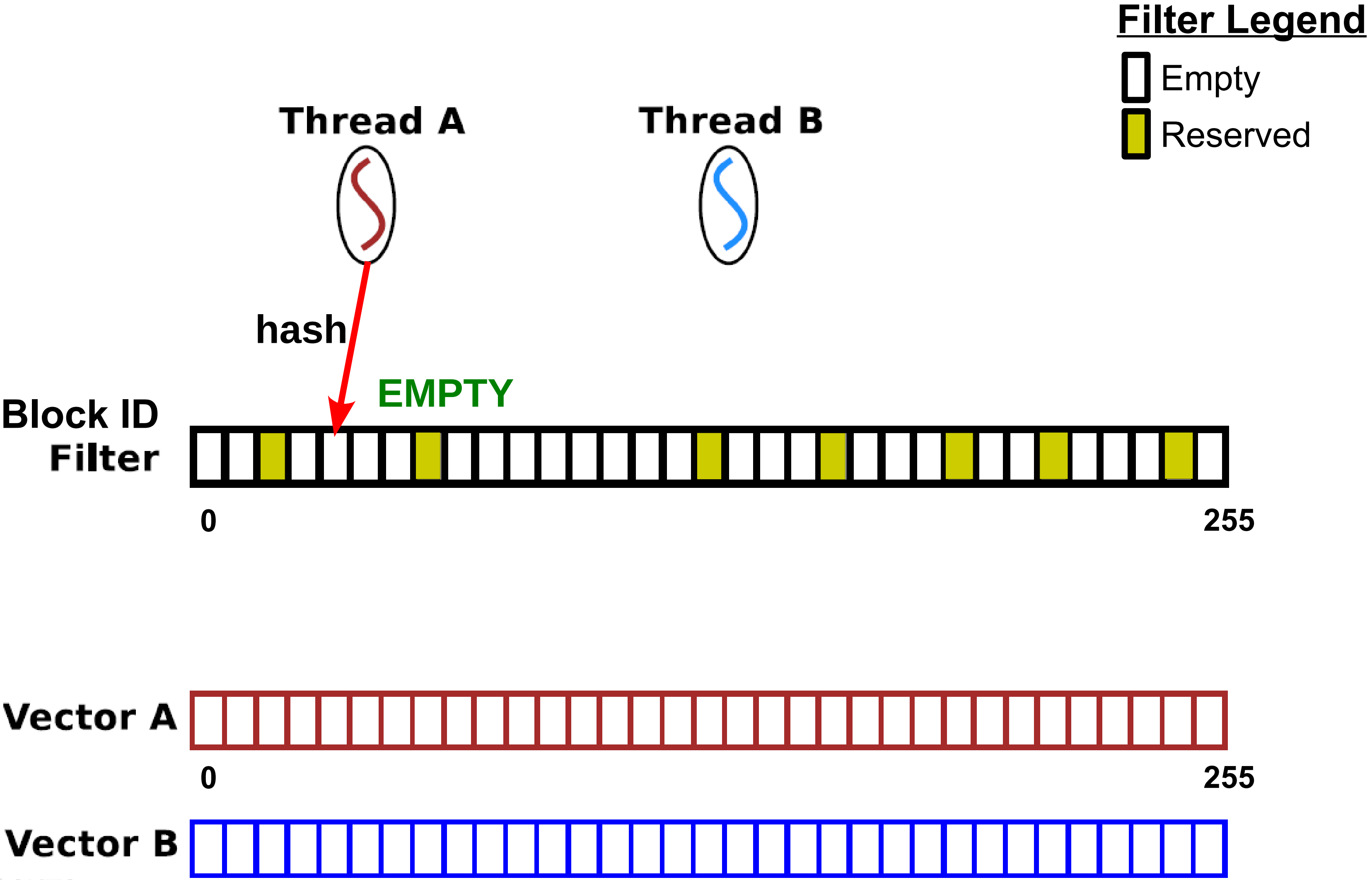
0

255

Vector B

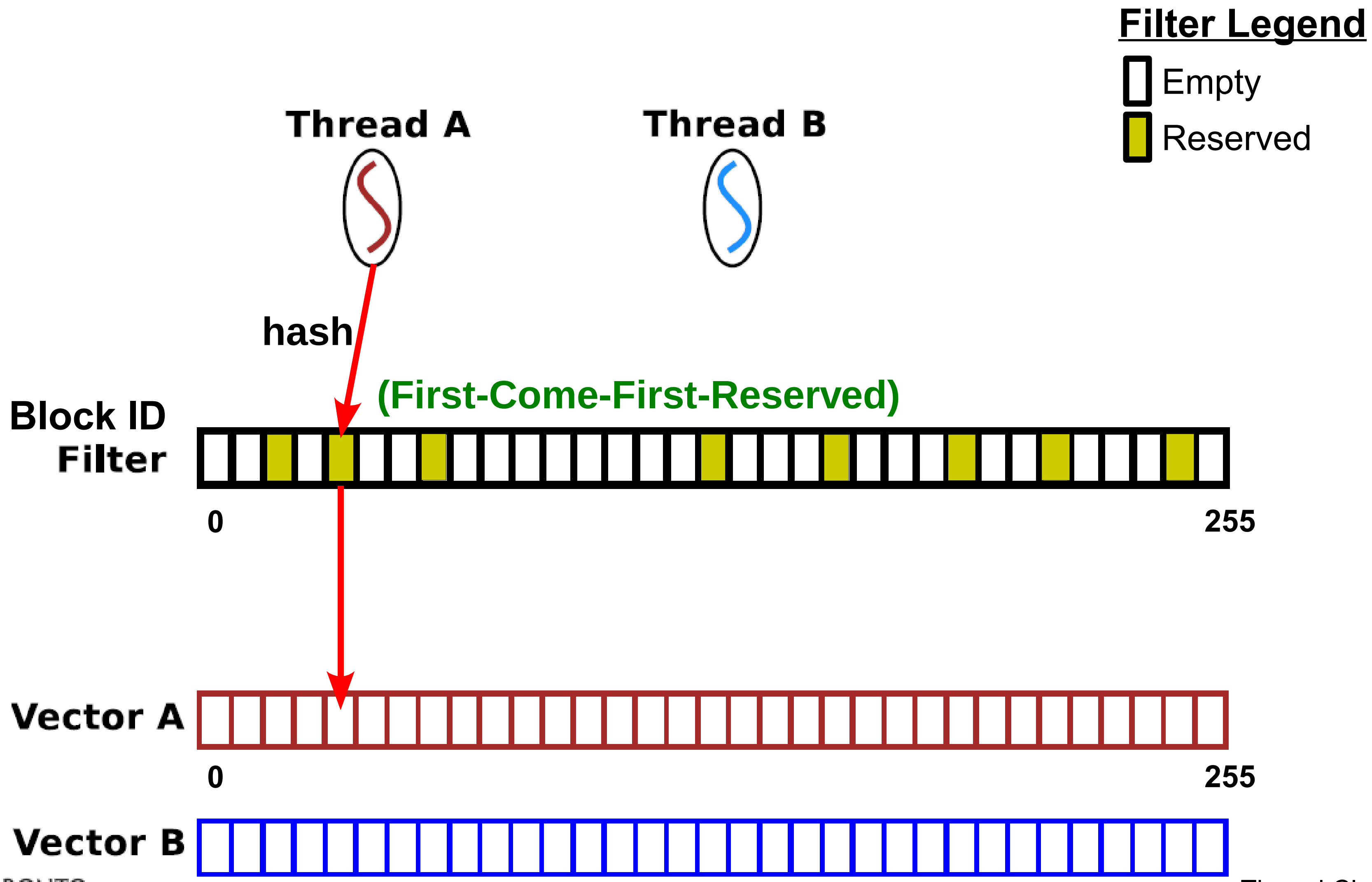
Spatial Sampling

- Hash collision & alias removal



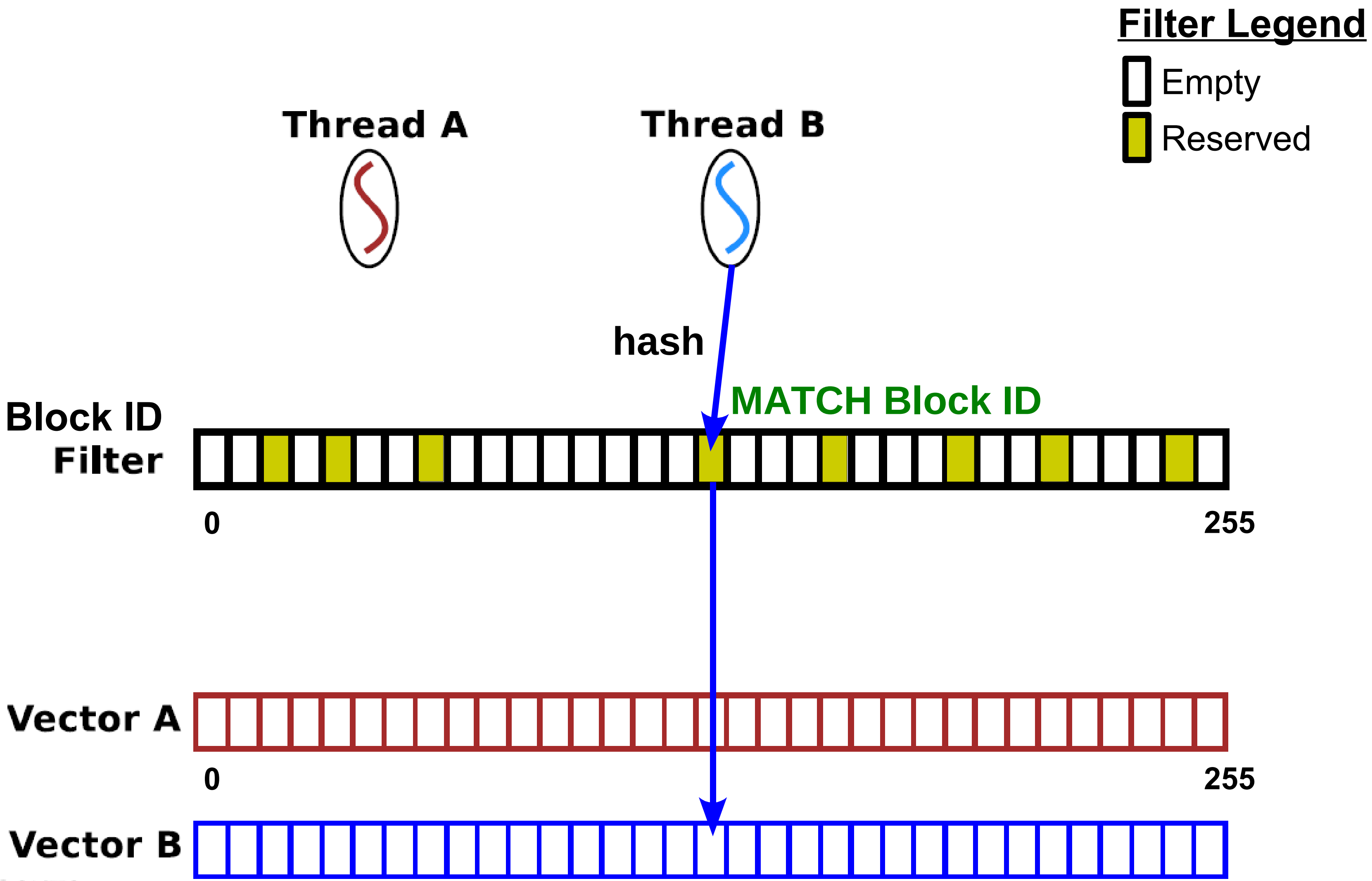
Spatial Sampling

- Hash collision & alias removal



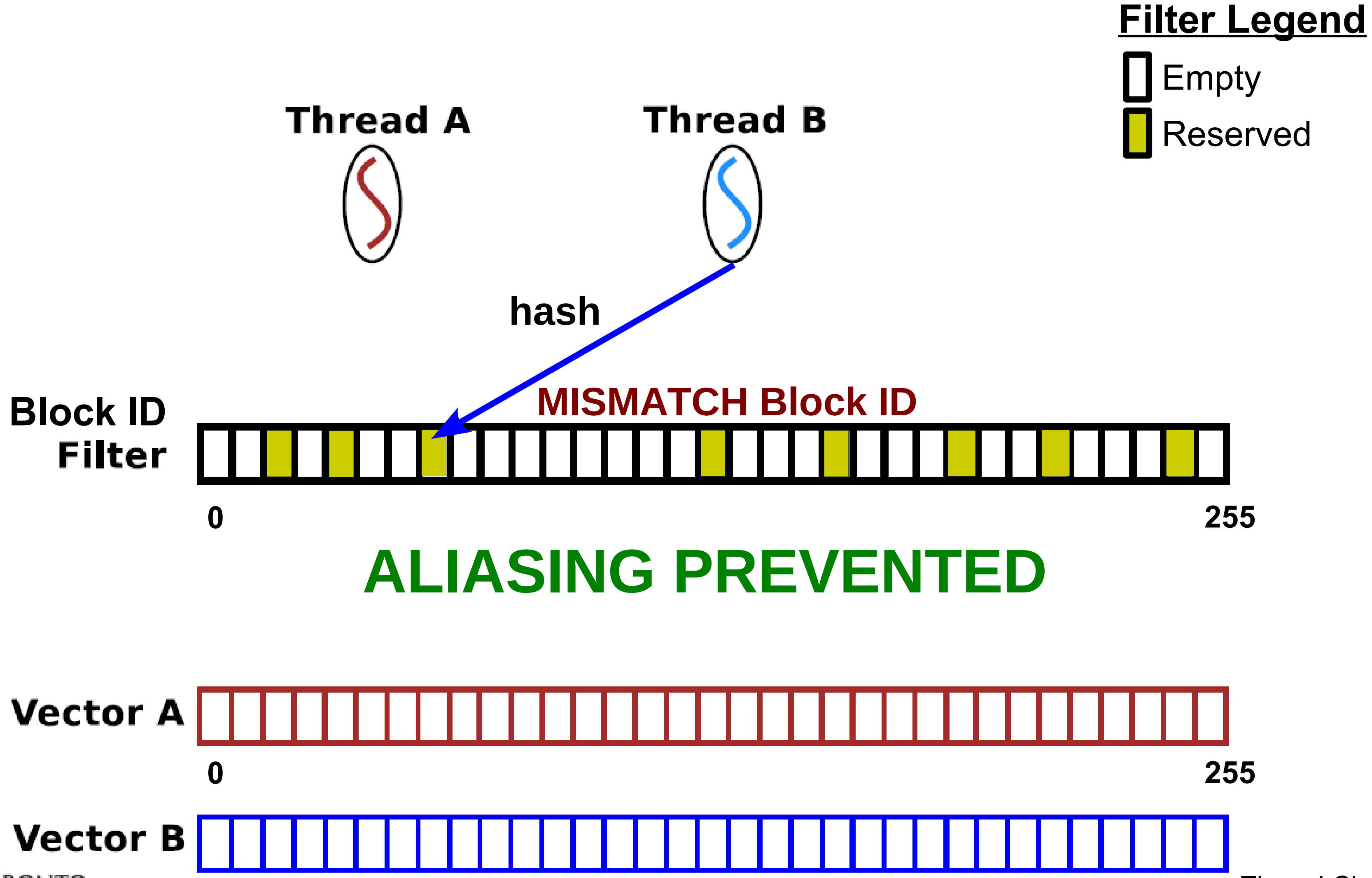
Spatial Sampling

- Hash collision & alias removal



Spatial Sampling

- Hash collision & alias removal



Automated Clustering

Clustering Heuristic:

- Simple, one-pass algorithm
- Compare vector against existing clusters
- If not similar, create a new cluster

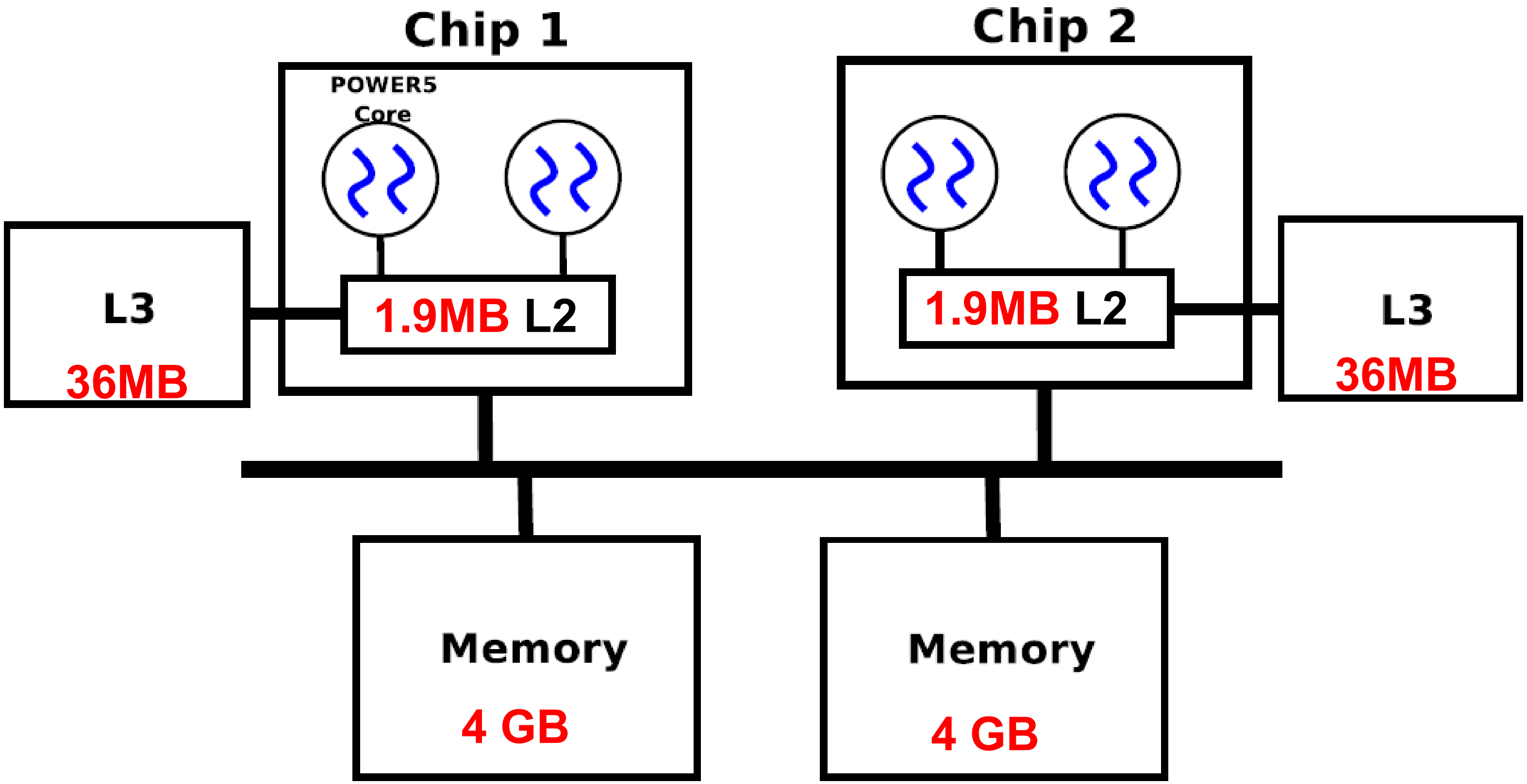
Similarity Metric:

$$\sum_{i=0}^{N} \mathrm{V}_1[i] * \mathrm{V}_2[i]$$

- Shared blocks amplified
- Non-shared blocks nullified

Experimental Platform

- 8-*way* Power 5, 1.5GHz
- Linux 2.6
- IBM J2SE 5.0 JVM



Workloads

Microbenchmark

- expect 4 clusters
  - 4 threads per cluster

SPECjbb2000 (modified)

- expect 2 clusters
  - 2 warehouses, 8 threads per warehouse

RUBiS + MySQL

- expect 2 clusters
  - 2 *databases*, 16 threads per *database*

VolanoMark chat server

- expect 2 clusters
  - 2 rooms, 8 threads per room

Visualizing Clusters

- An example

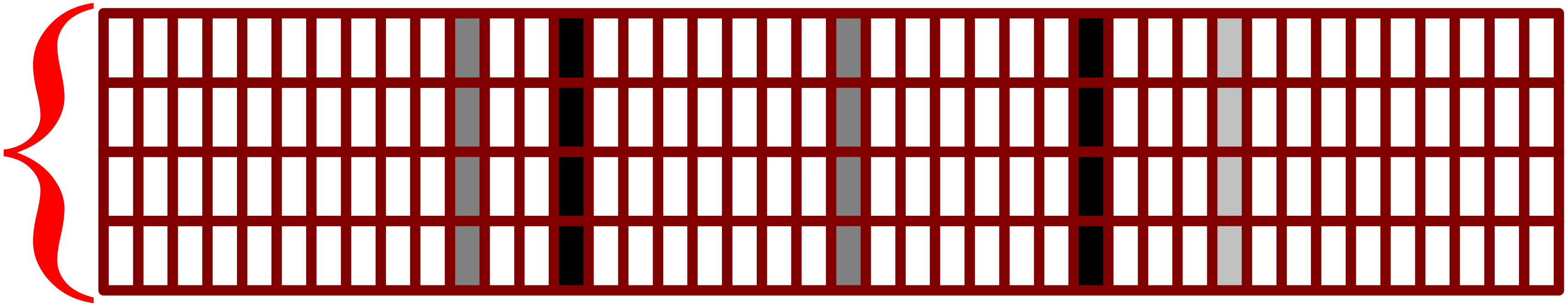
Counter Values

■ 255

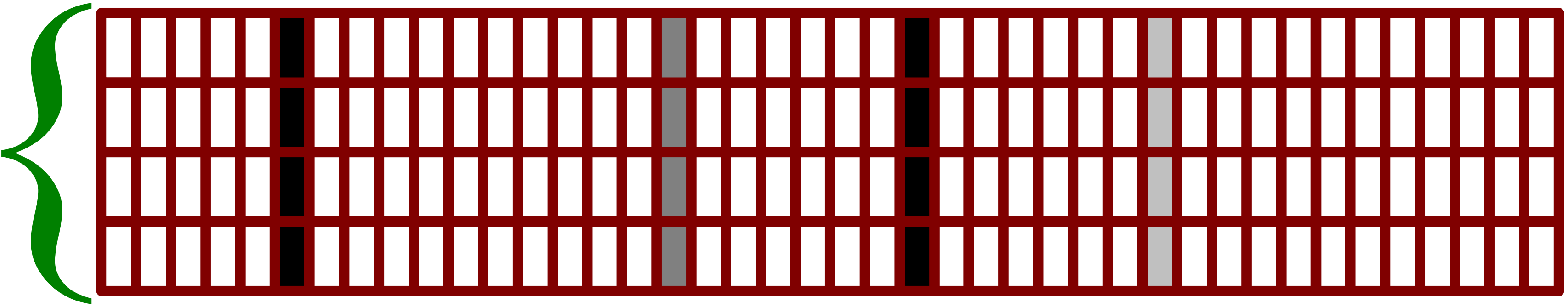
▩ 128

▨ 64

□ 0



Cluster A, 4 vectors



Cluster B, 4 vectors

Visualizing Clusters

- An example

Counter Values

■ 255

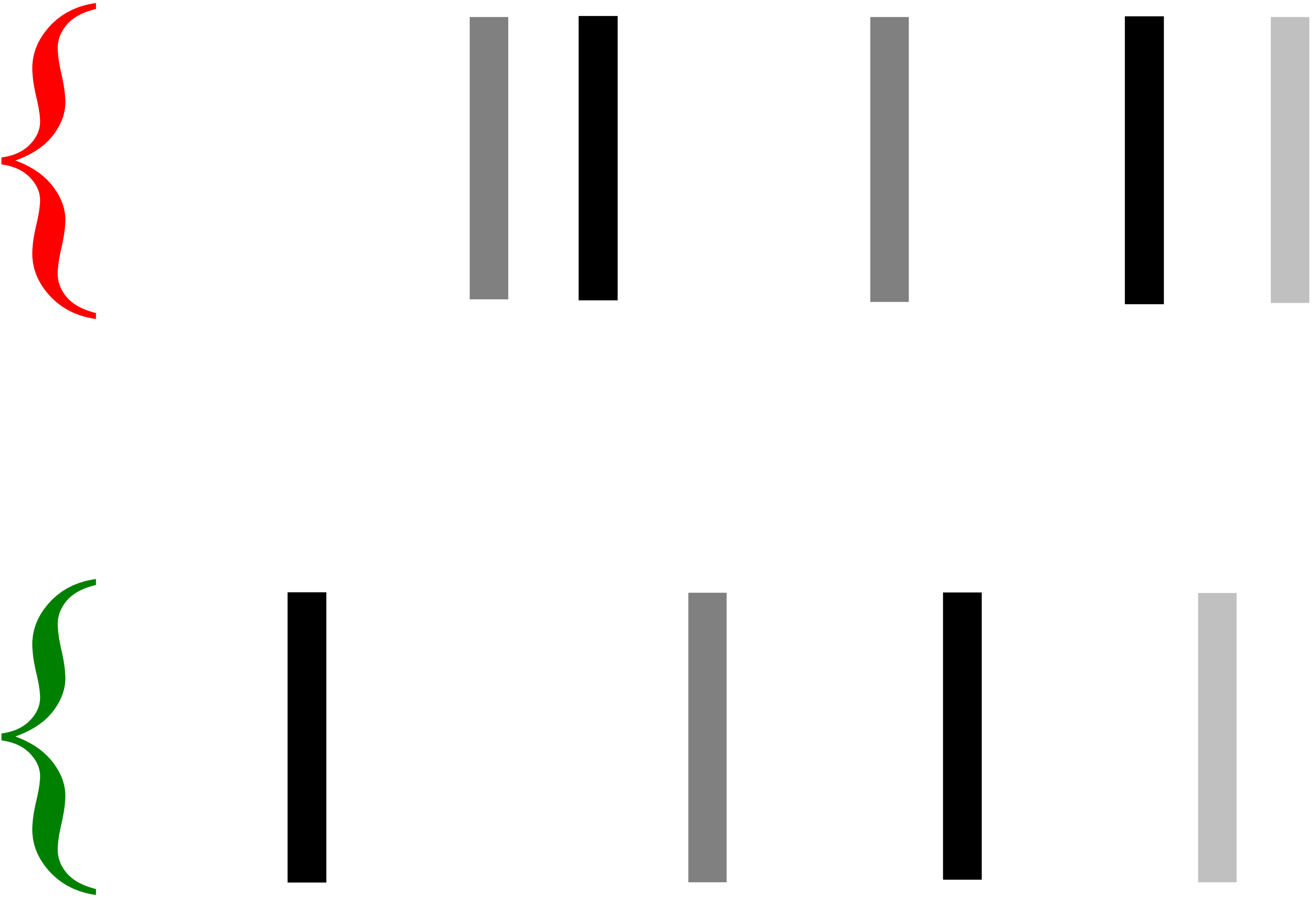
▩ 128

▨ 64

□ 0

Cluster A,
4 vectors

Cluster B,
4 vectors



Visualizing Clusters

- An example

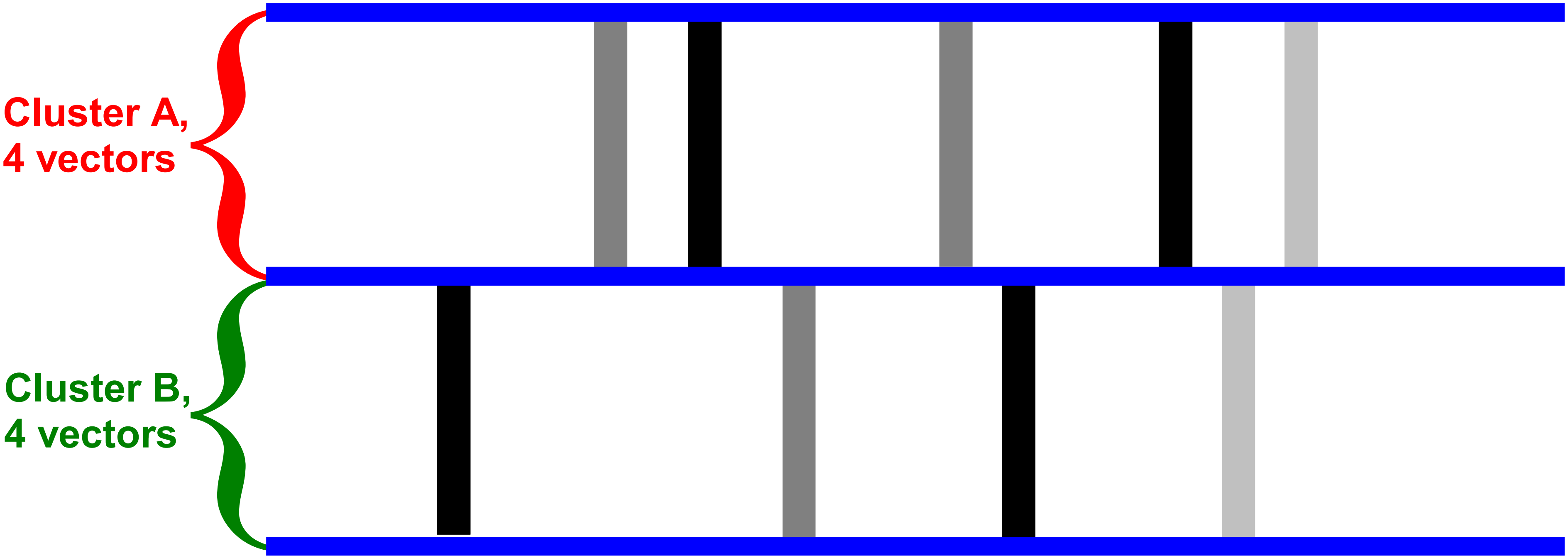
Counter Values

■ 255

▩ 128

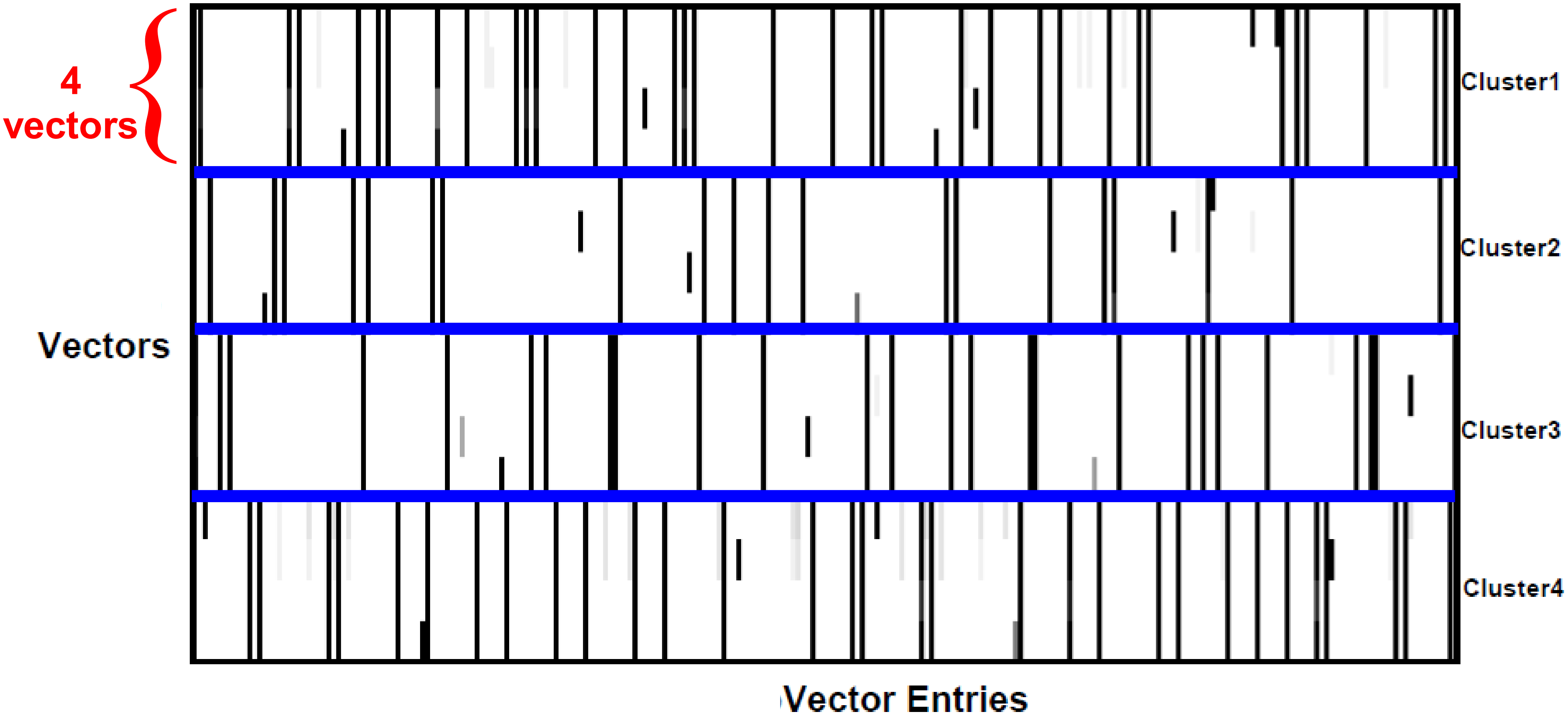
▨ 64

□ 0



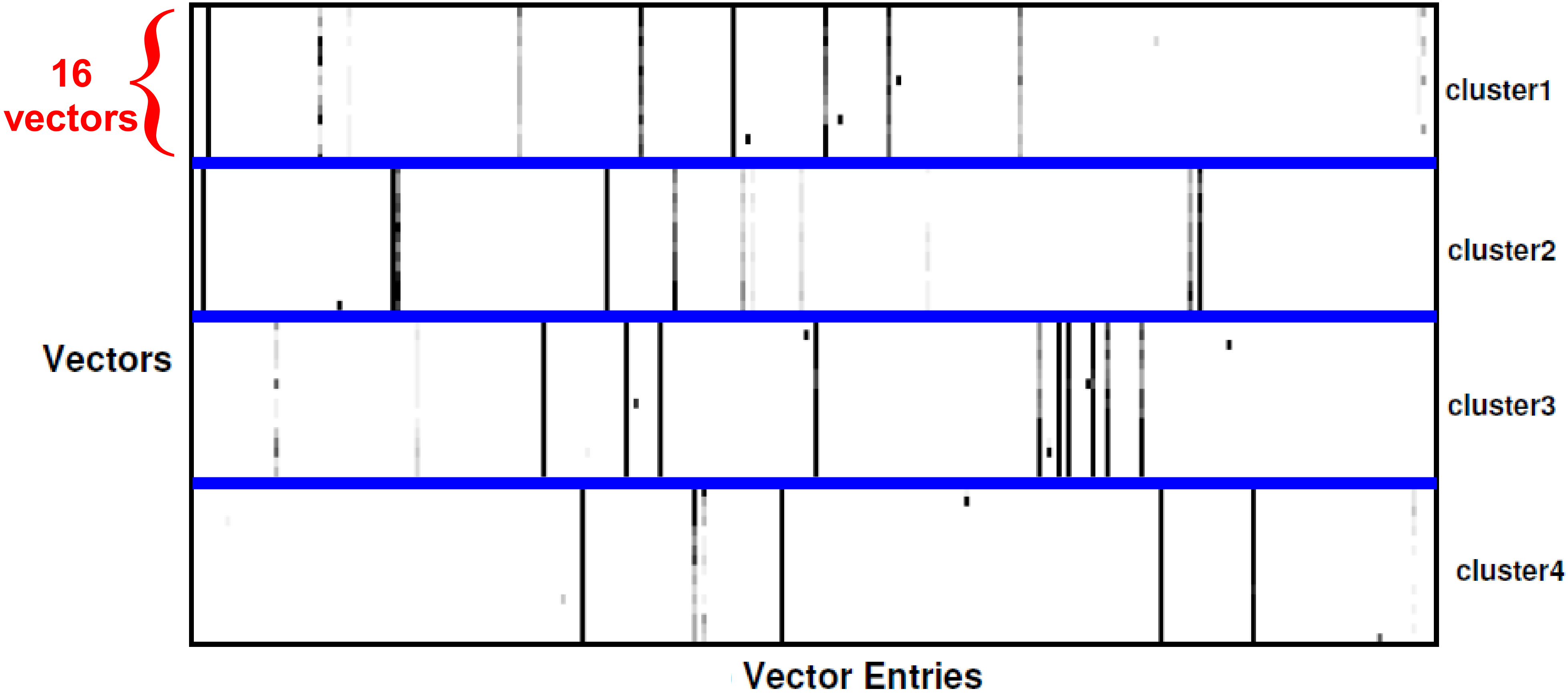
Visualizing Clusters

- Microbenchmark



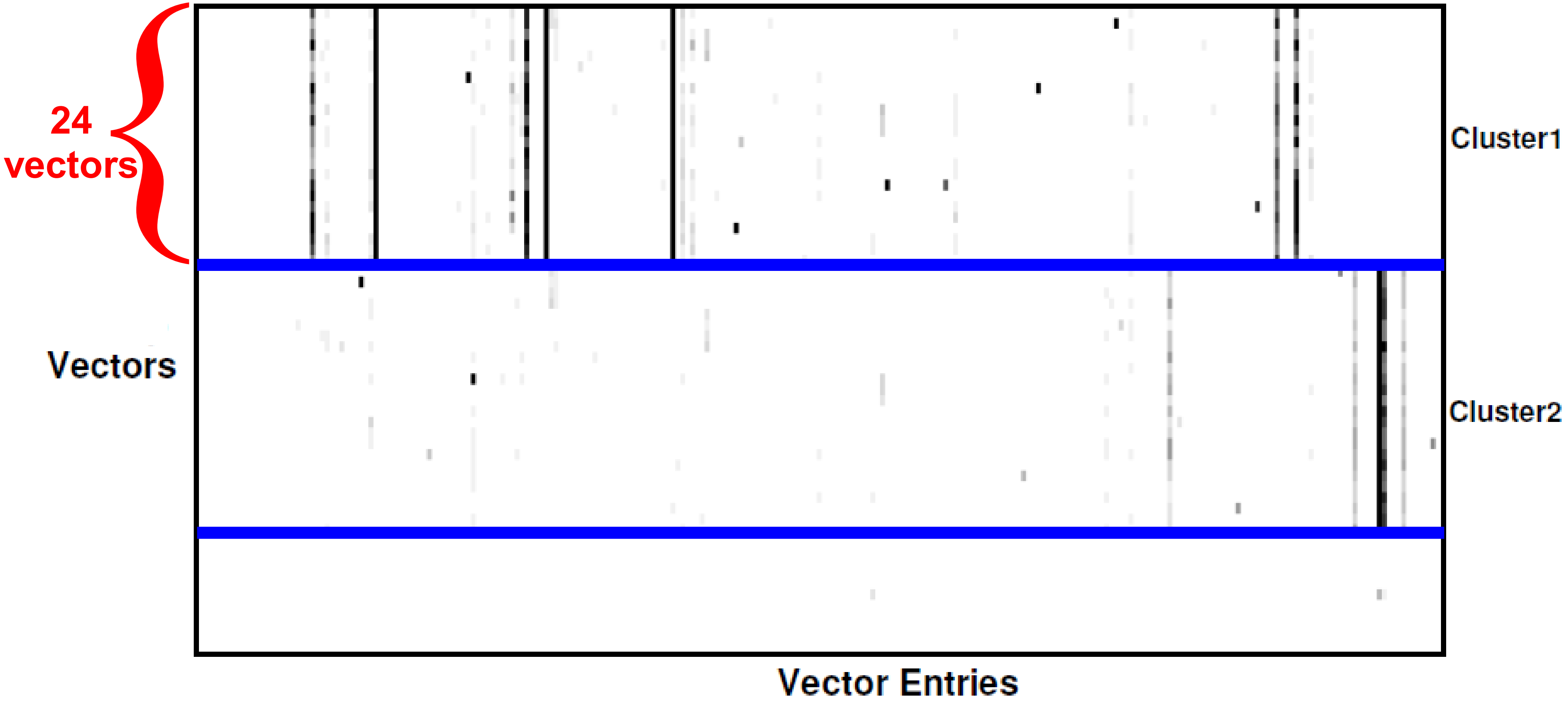
Visualizing Clusters

- Modified SPECjbb2000 (4 warehouses)



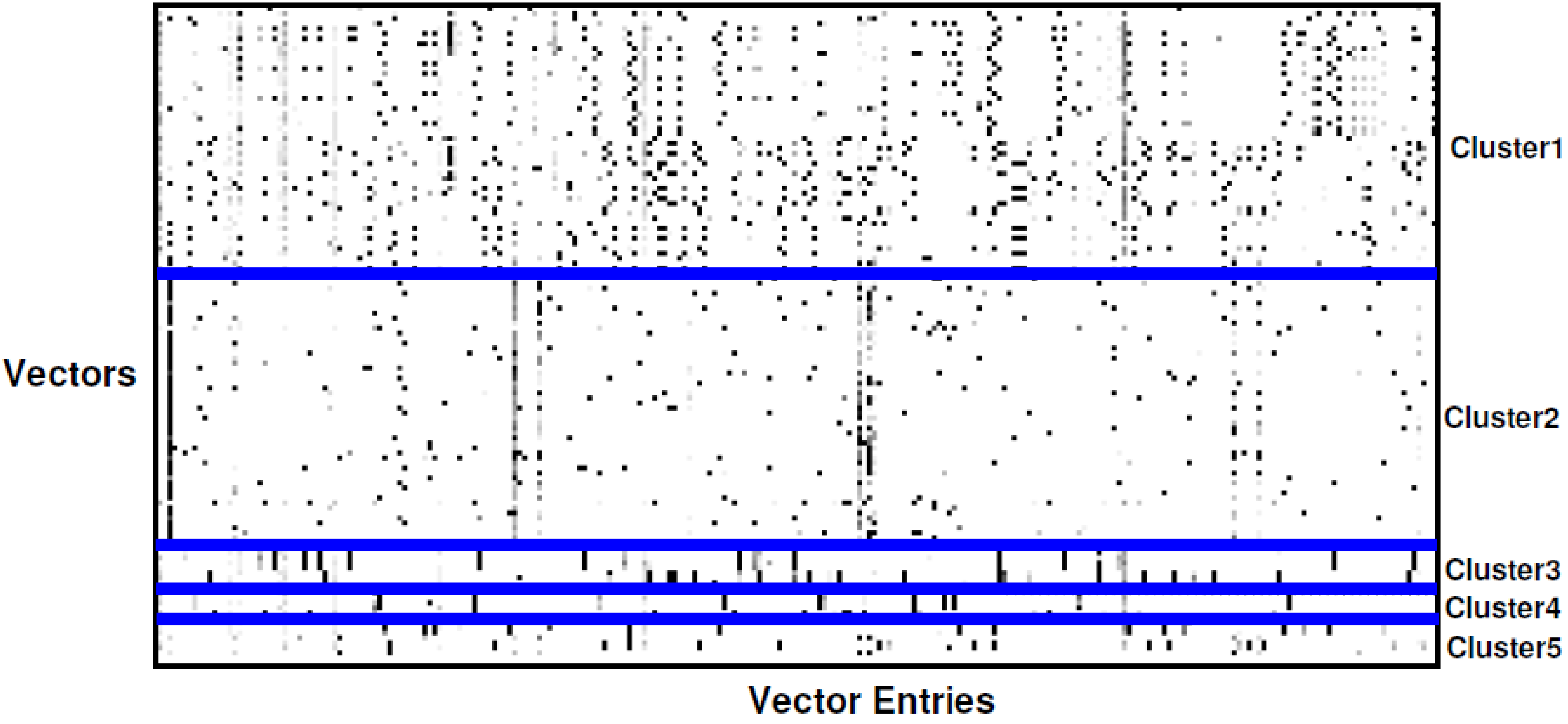
Visualizing Clusters

- RUBiS + MySQL (2 *databases*)



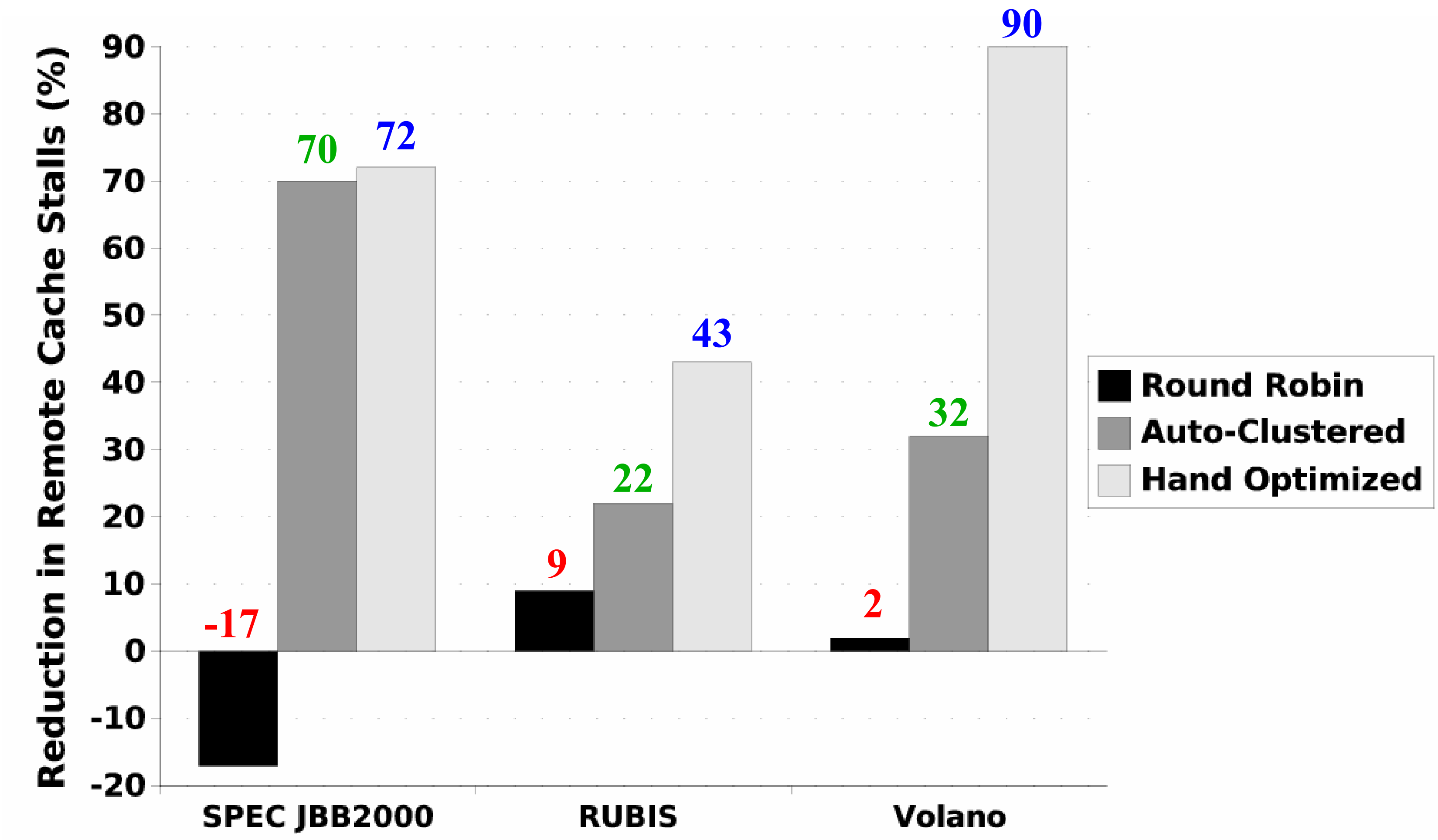
Visualizing Clusters

- VolanoMark (4 rooms)



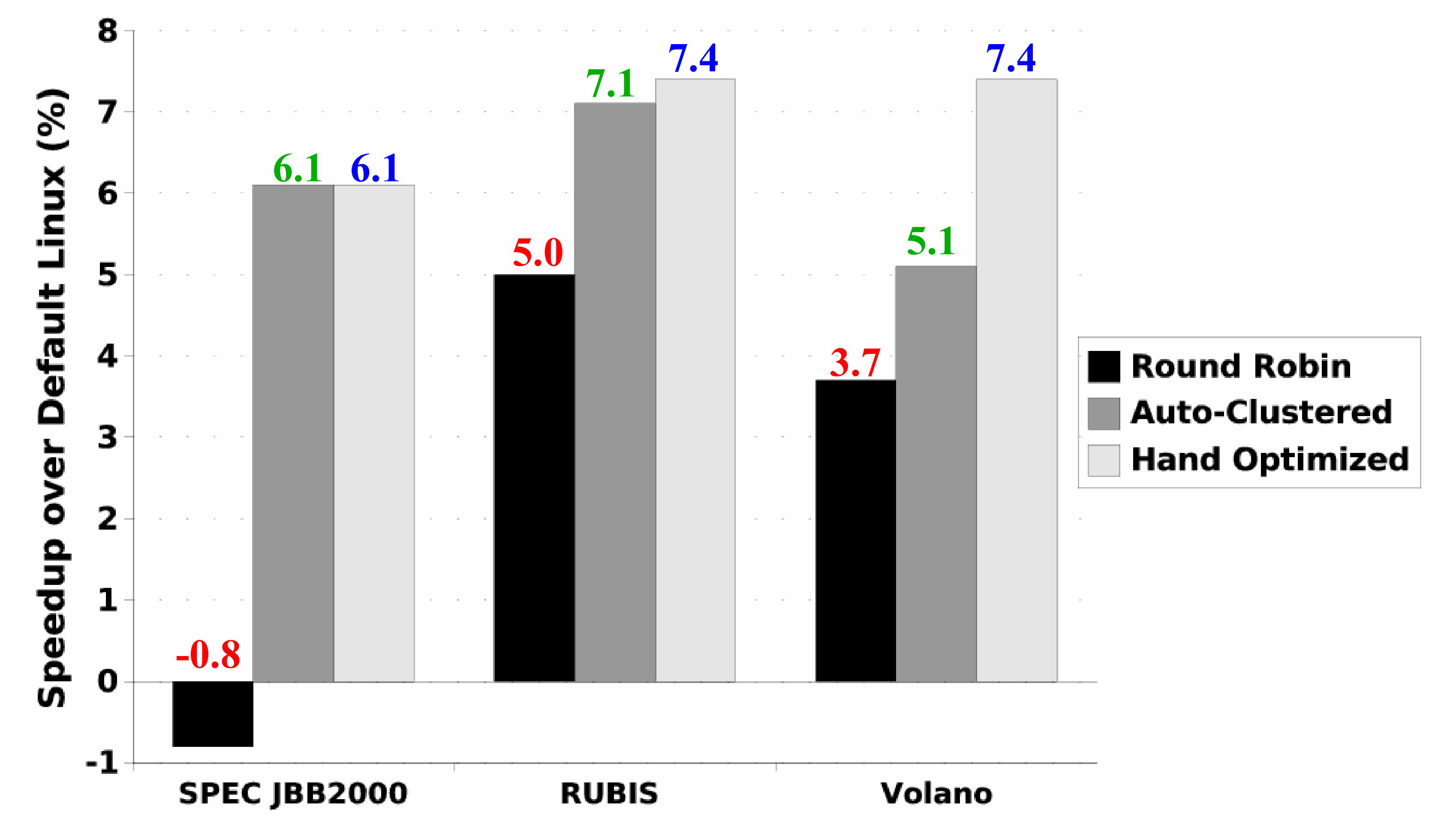
Remote Cache Impact

- Normalized to default Linux



Performance Impact

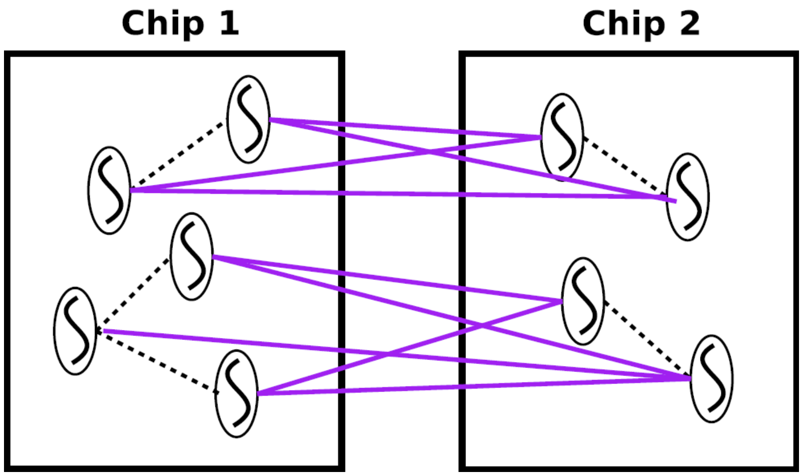
- IPC: instructions per cycle
- Normalized to default Linux



Summary

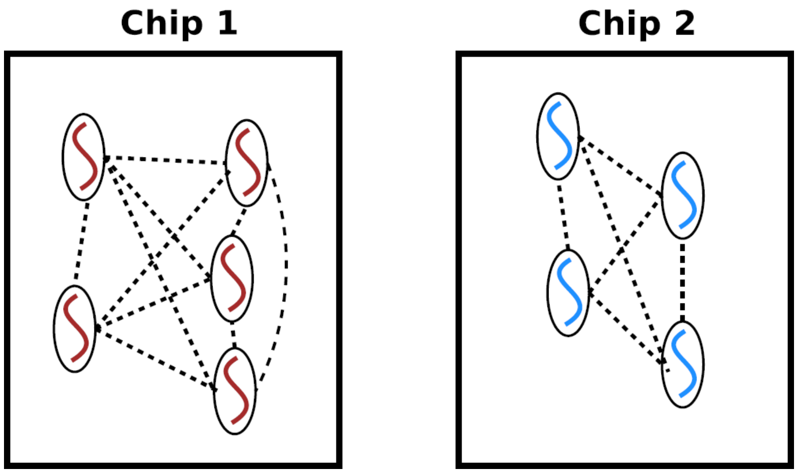
BEFORE:

Current Operating Systems



AFTER:

Operating System With Thread Clustering



Conclusions

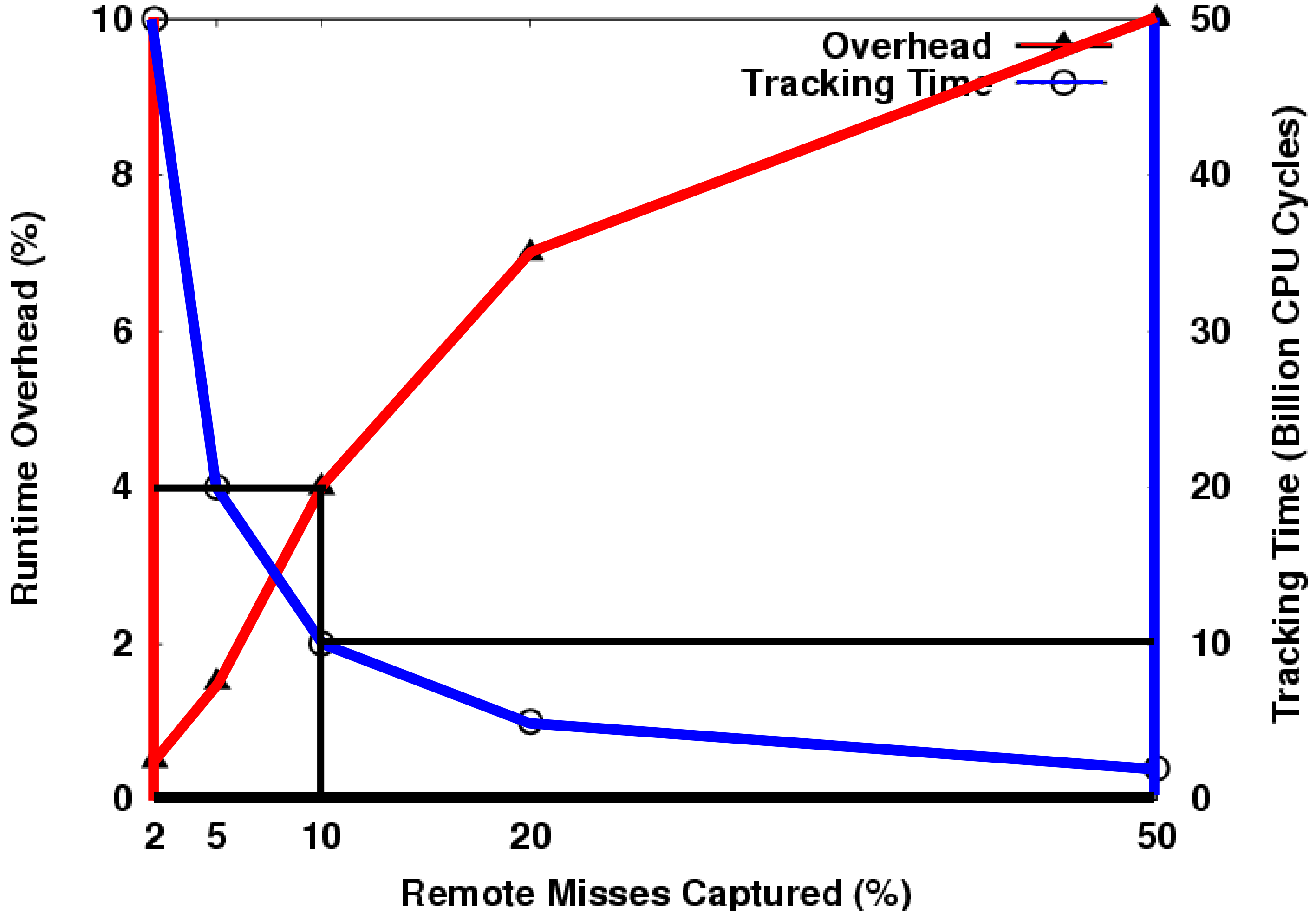
- HPCs can detect sharing
- Sharing signatures are effective
- Automated thread clustering:
  - Reduces remote cache access up to 70%
  - Improves performance up to 7%
- All with low overhead

Future Work:

- More workloads
- Improve clustering algorithm
- Integration with load-balancing aspects

Sampling Overhead

- Modified SPECjbb2000


Overhead
Tracking Time
Runtime Overhead (%)
Tracking Time (Billion CPU Cycles)
Remote Misses Captured (%)