CONTENTION AWARE EXECUTION

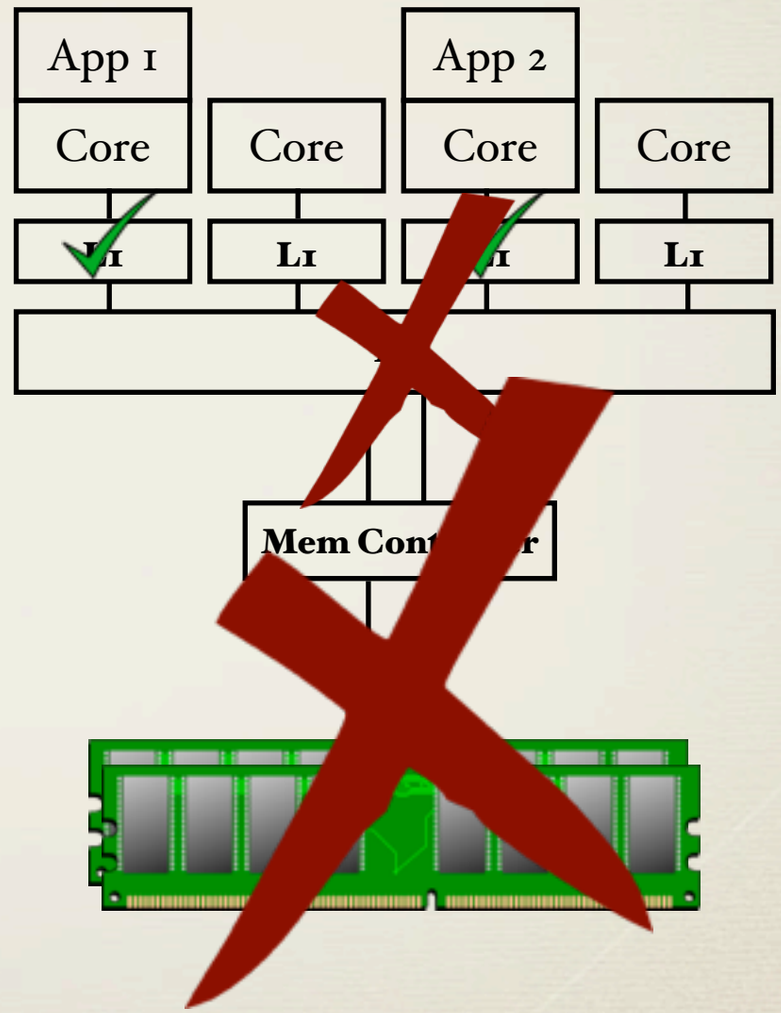
Online contention detection and response

Jason Mars[1], Neil Vacharajani[2], Robert Hundt[2], Mary Lou Soffa[1]

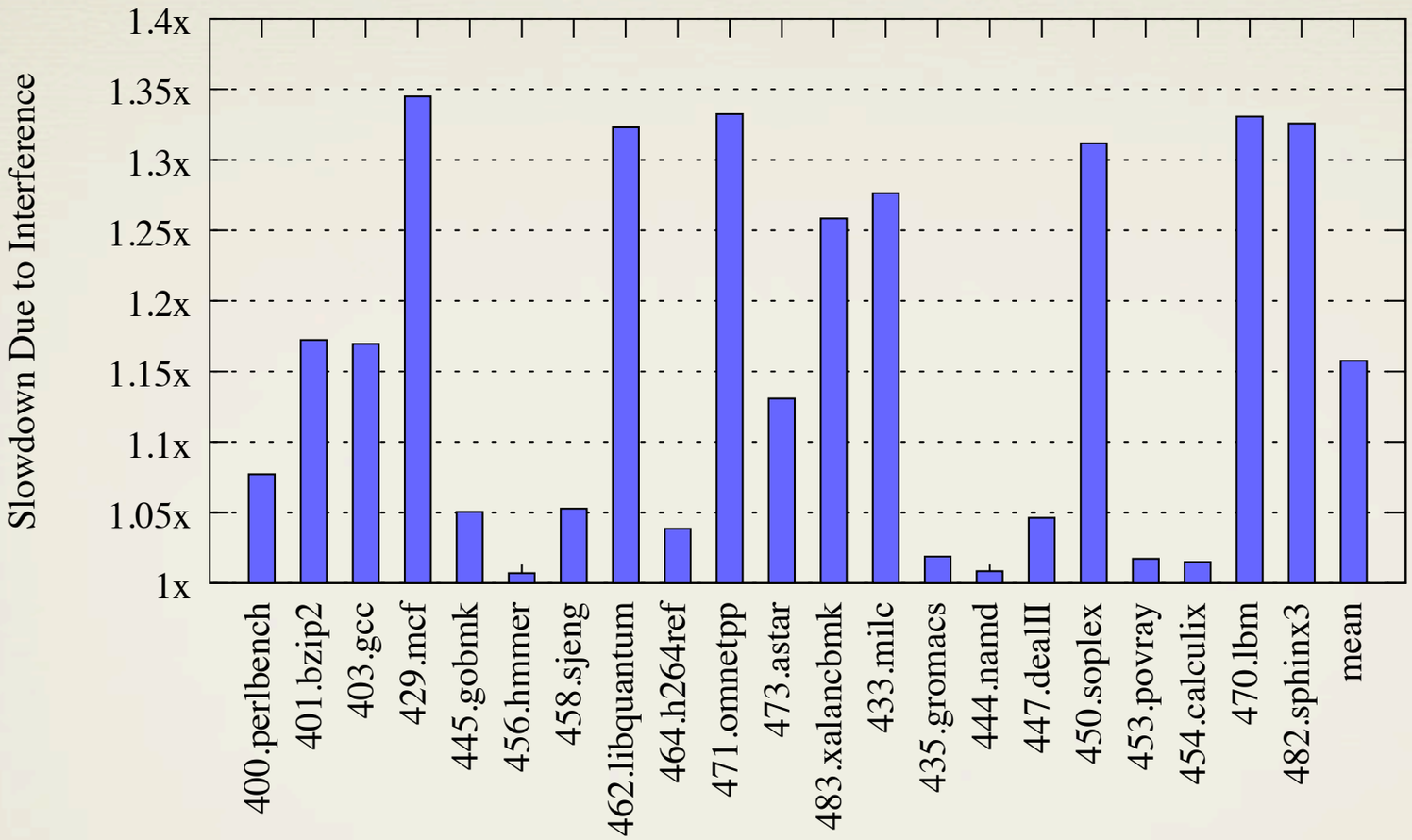
[1]University of Virginia
[2]Google

Problem

- Multicore is ubiquitous
  - Commodity chips
    - On desktop
    - In the datacenter
  - Promises parallel processing
  - Does not always deliver
- Memory subsystem is shared
  - On chip last level cache
  - Bus, memory, disk
  - Contention occurs



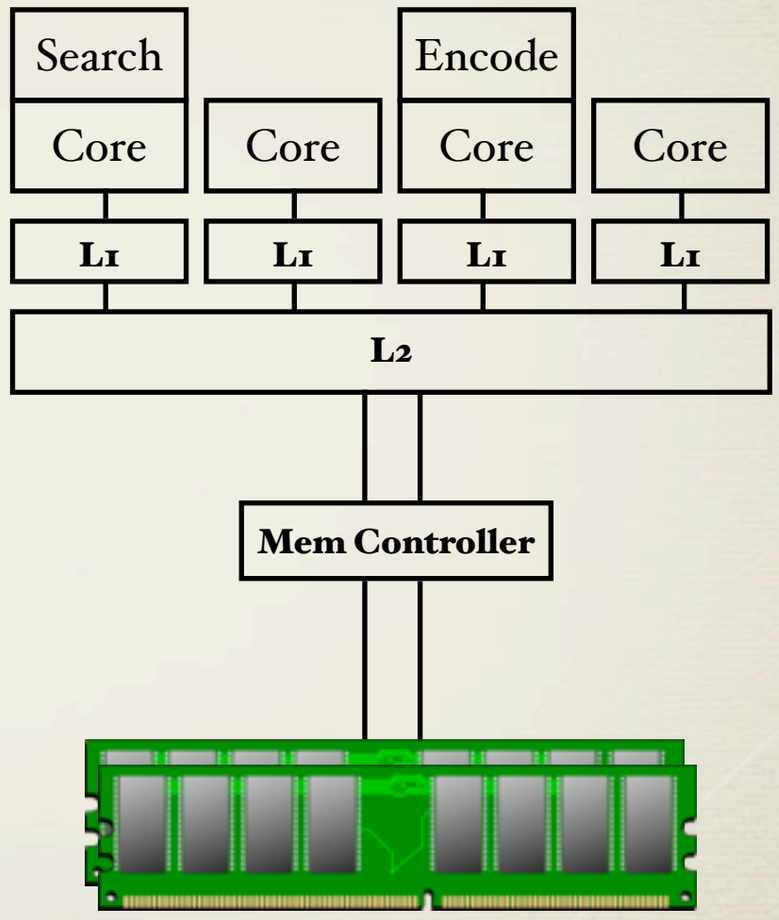
Contention



- Contention causes cross-core interference
  - Just 2 co-running SPEC2006 applications
  - 35% Slowdown
  - State of the art Quad-core (Core i7)
  - Simply not tolerable in many application domain (QoS) e.g. datacenter

Contention in the datacenter

- Cannot withstand performance interference
  - Latency sensitive applications
    - Search, Email
    - User facing services
  - Batch/Throughput applications
    - Compression, Video Encode
    - Behind-the-scenes work
- Current proposed solution
  - Disallow co-location
    - Wasteful, sacrifices utilization



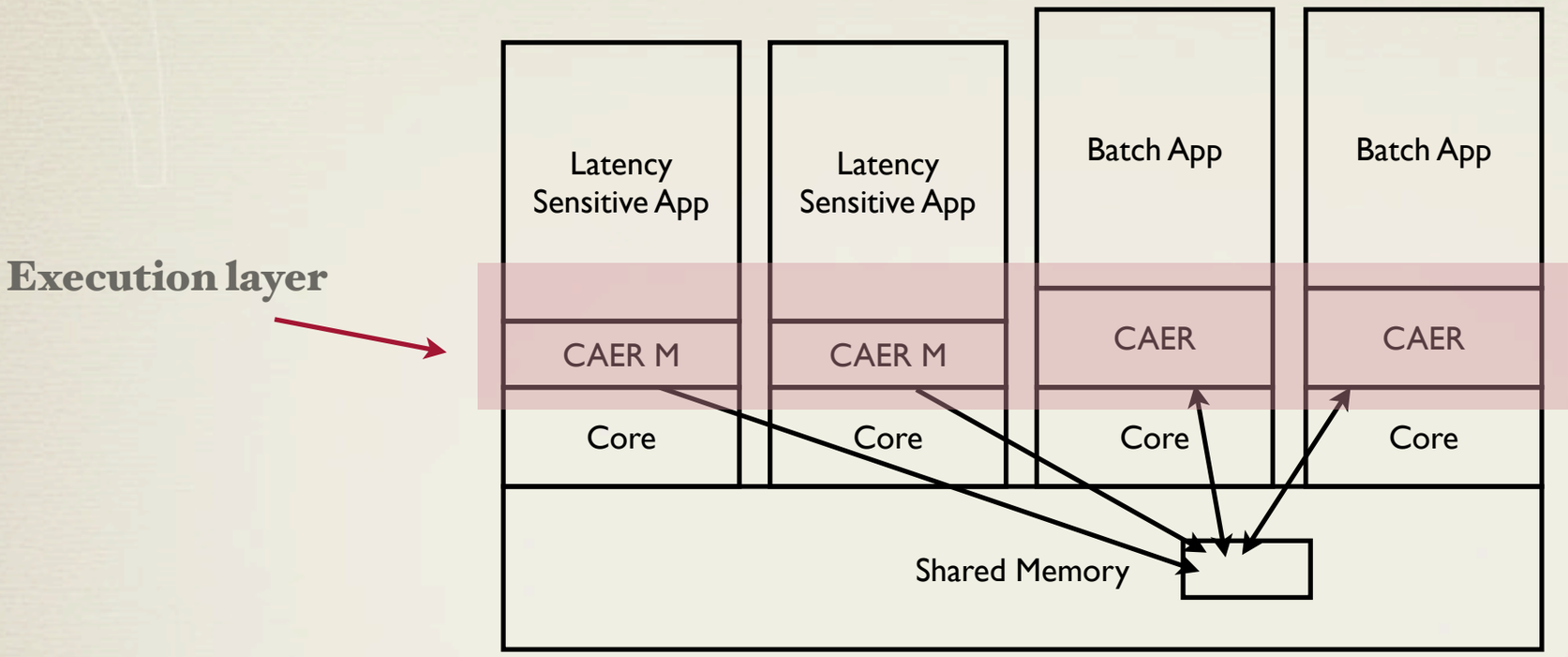
Goal

- Provide a better solution for performance isolation for the QoS of applications
  - On commodity multicore processors
  - Improves multicore processor utilization

Approach: Contention Aware Execution

- Address contention using a runtime layer
- A contention aware execution environment
    1. Detect contention
    2. Respond to contention
- CAER: Contention Aware Execution Runtime
- Key Outcome: It *Can* be done on current hardware
    - Detect and respond without hardware support

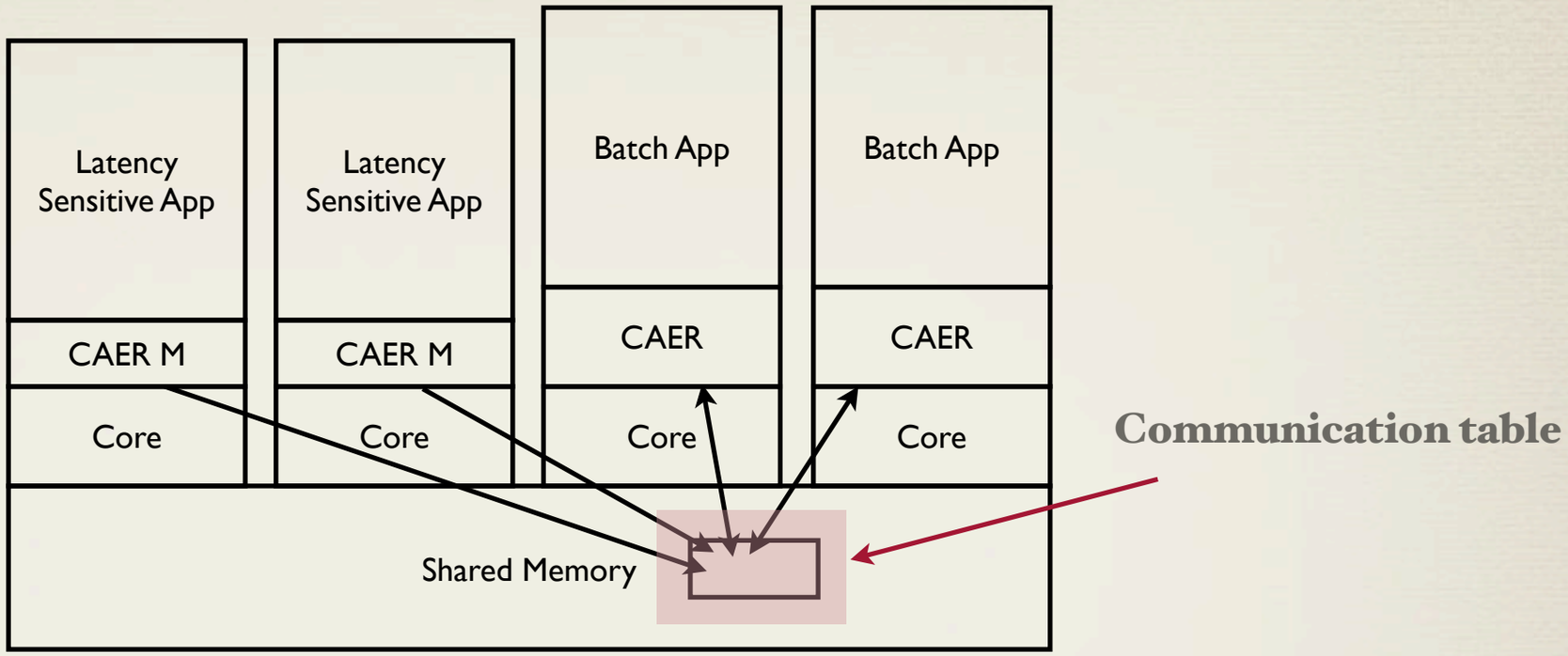
CAER: Execution Layer



CAER Architecture

- Cross-core application cooperation
- A runtime under each application of interest

CAER: Execution Layer

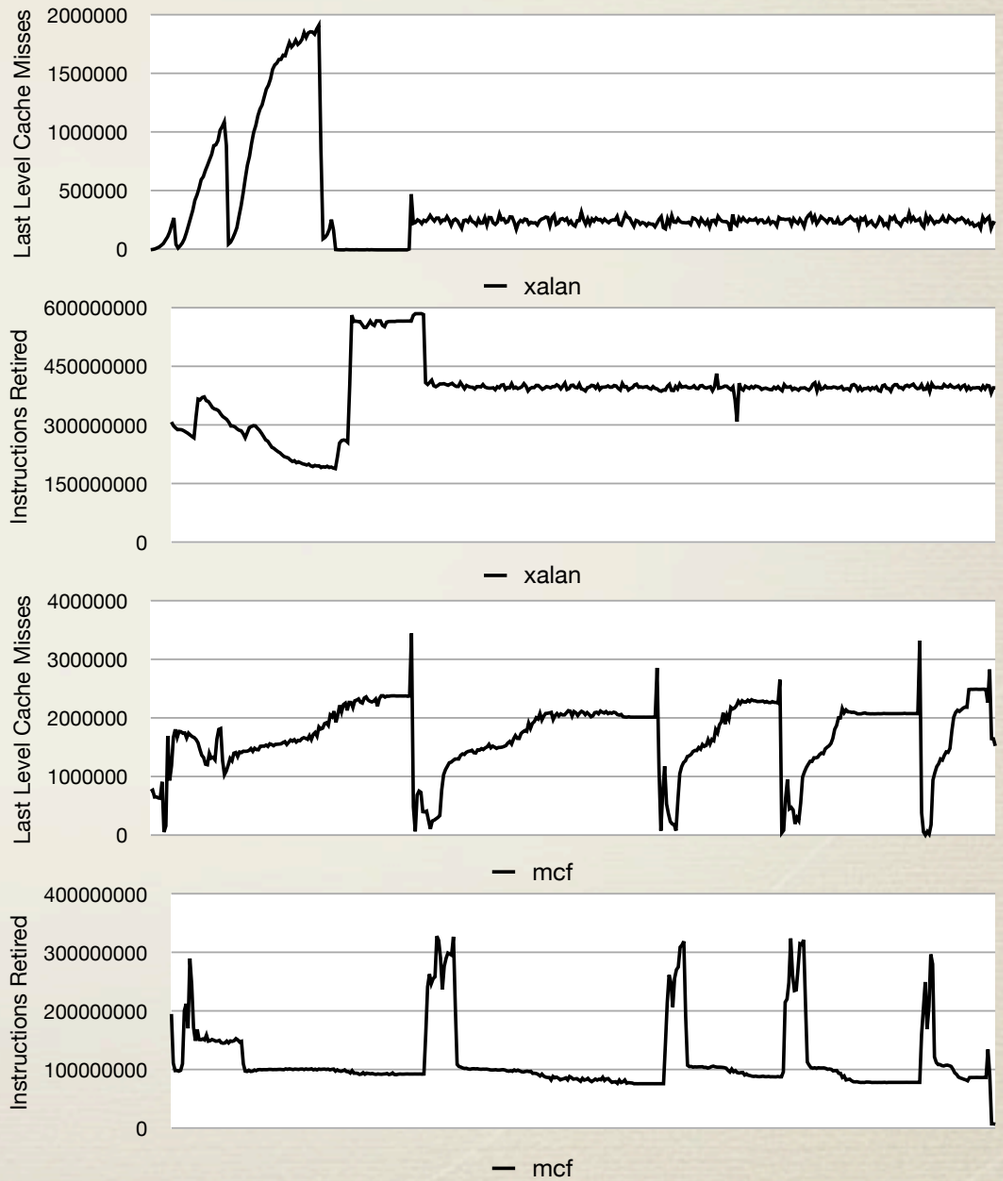


CAER Architecture

- Communication Table
  - Record monitoring information (via periodic probing)
  - Infer contention using performance information - react accordingly

CAER: Execution Layer

- Uses hardware performance monitors (HPM)
- CAER monitors and exploits cache behavior
- Last level cache misses indicates memory subsystem usage and performance

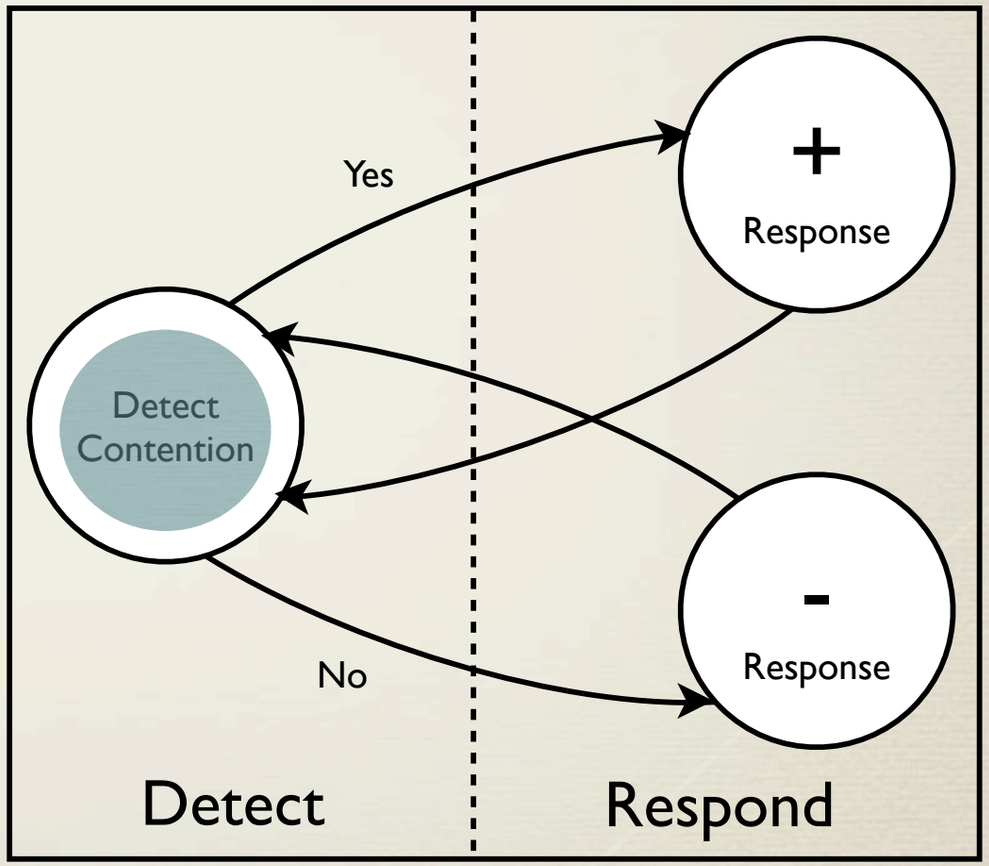


Outline

- ✓ Problem / Motivation
- ✓ CAER overview
- Achieving detection and response
  - Burst shutter approach
  - Rule based approach
- Evaluation
- Conclusion

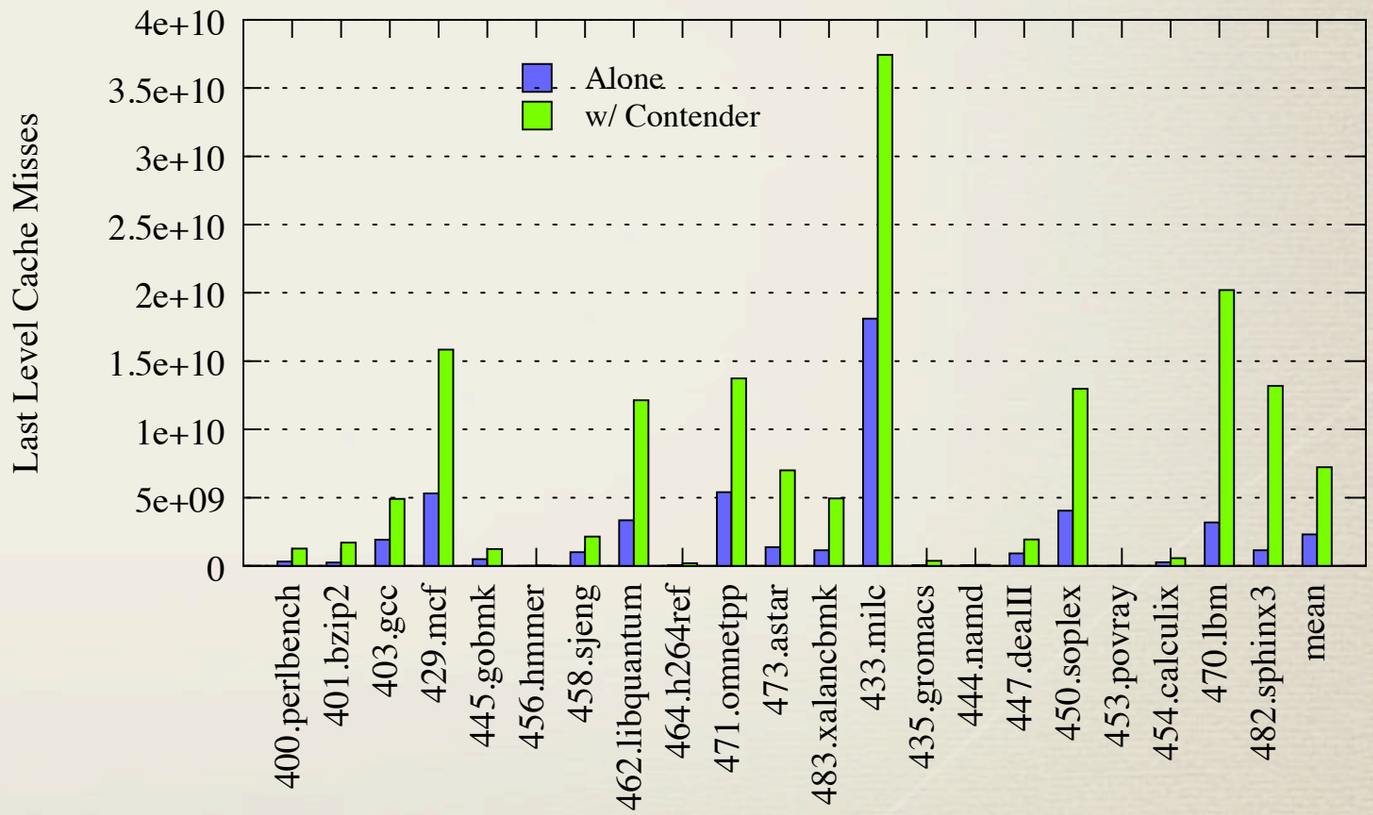
Achieving Contention Aware Execution

- Execution layer duties
  - Detect Contention
  - Enact Contention Response



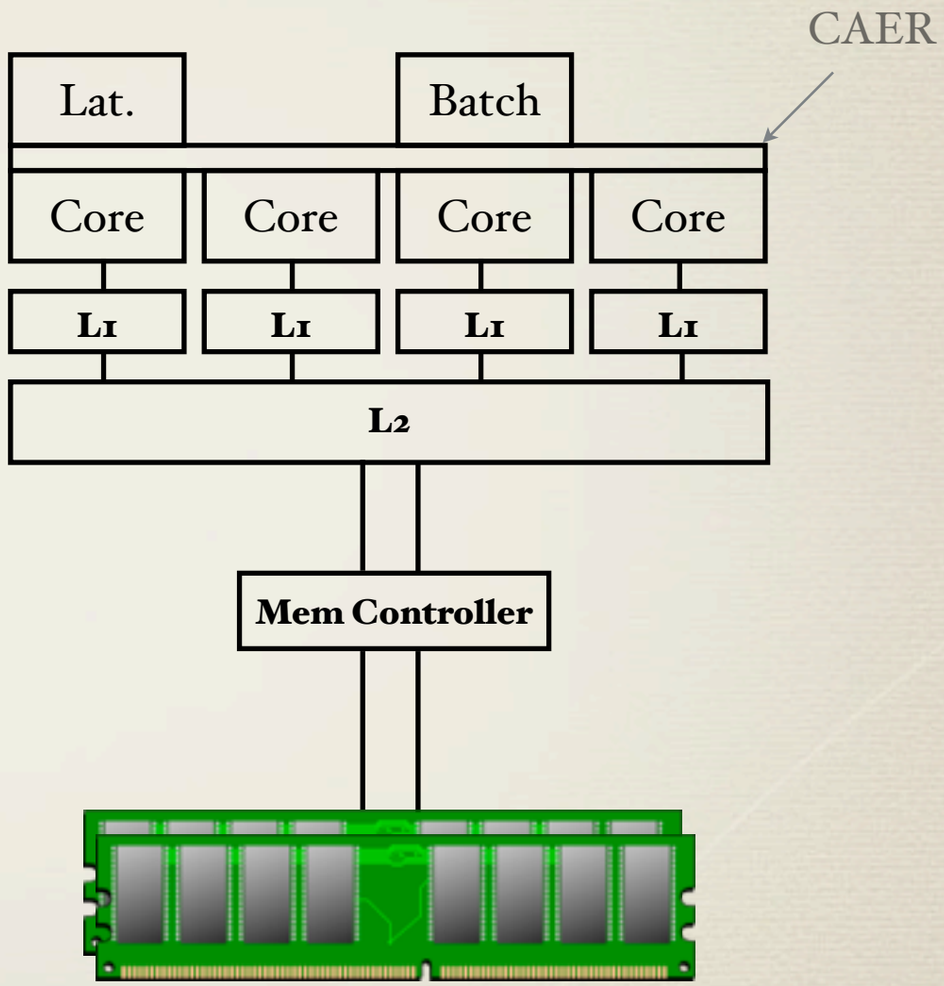
Detecting Contention with Burst Shutter

- Observation and insight
  - Contending case vs not contending
  - Observable with HPM (LLC misses)
  - Leads to designing burst shutter



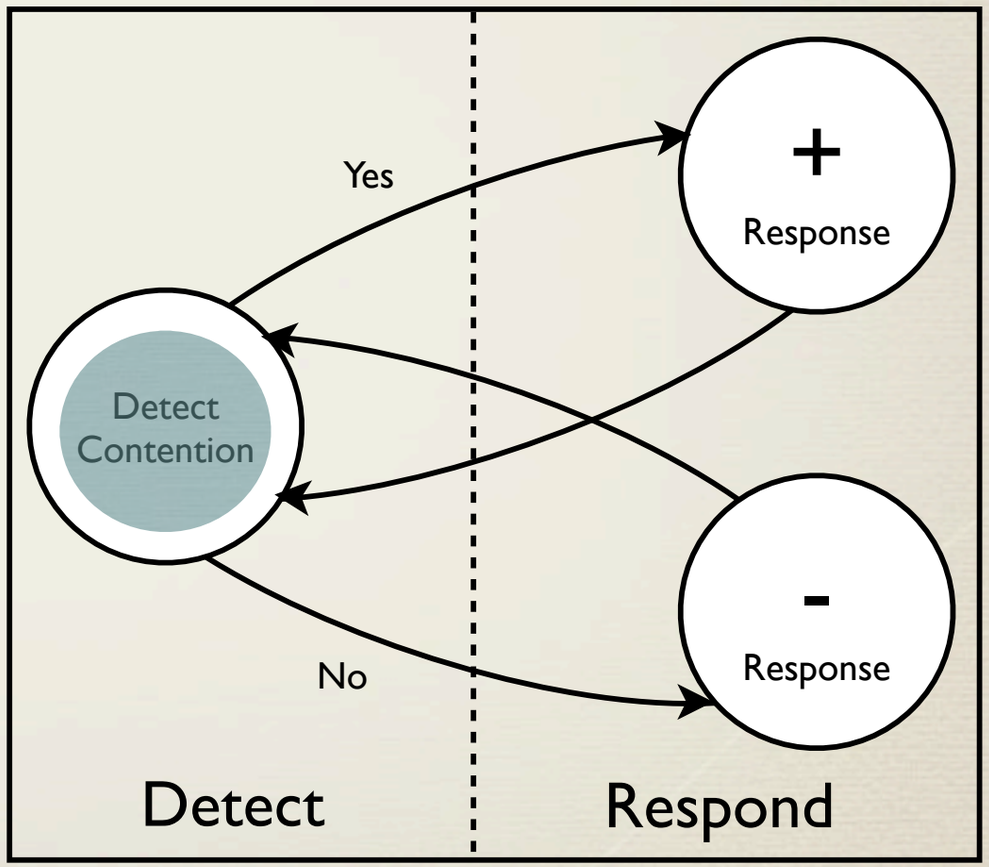
Detecting Contention with Burst Shutter

- Burst shutter heuristic
  - 2 applications are co-located
  - Stagger execution of 'batch' job
  - Observe impact on 'latency sensitive' job
  - Threshold based heuristic
- Complete algorithm and discussion in paper



Contention Response with Burst Shutter

- Reduce pressure on memory subsystem
  - Pausing ‘batch’ application for fixed period
  - Retest for contention

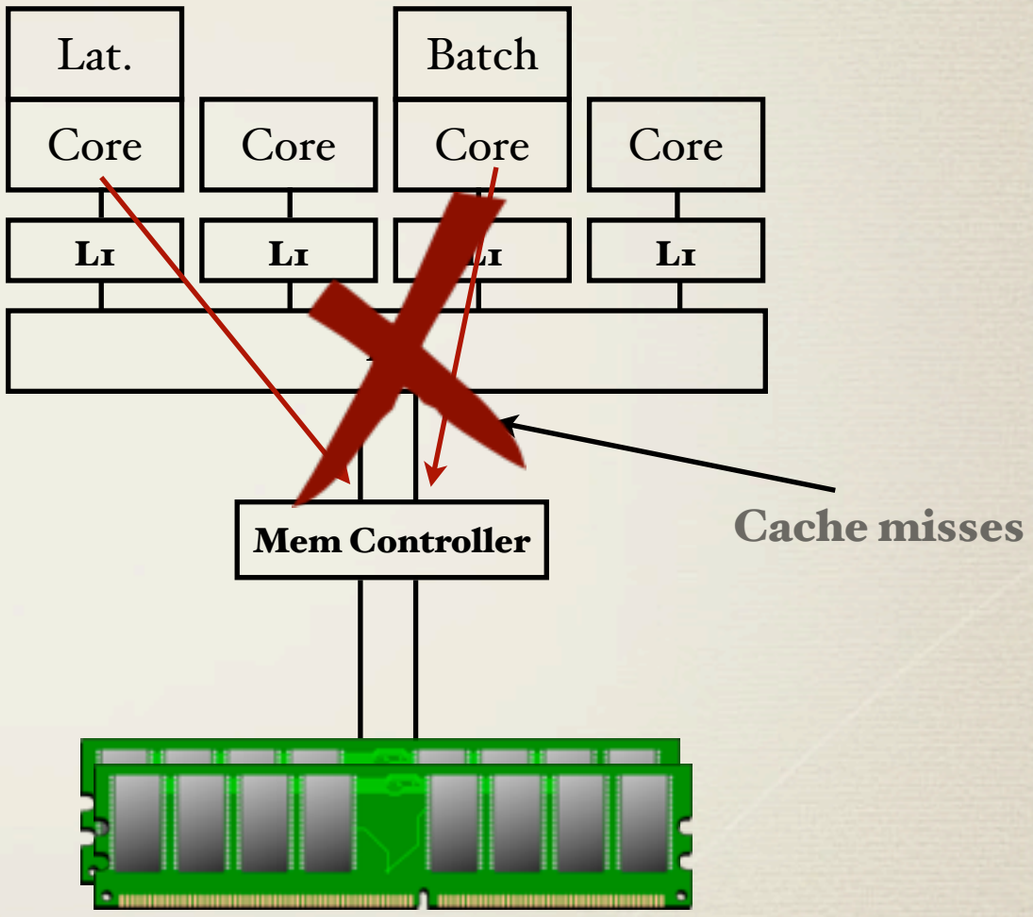


Detecting Contention with Rule Based Approach

- Insight and observation
  - Use indication working set size
  - LLC Cache misses as an indicator
  - Implies larger working set

Detecting Contention with Rule Based Approach

- Rule based heuristic
  - Monitor LLC miss rate of applications
  - If both batch are missing, assume contention in LLC
  - Also threshold based
- Full Algorithm and description in paper



Contention Response with Rule Based Approach

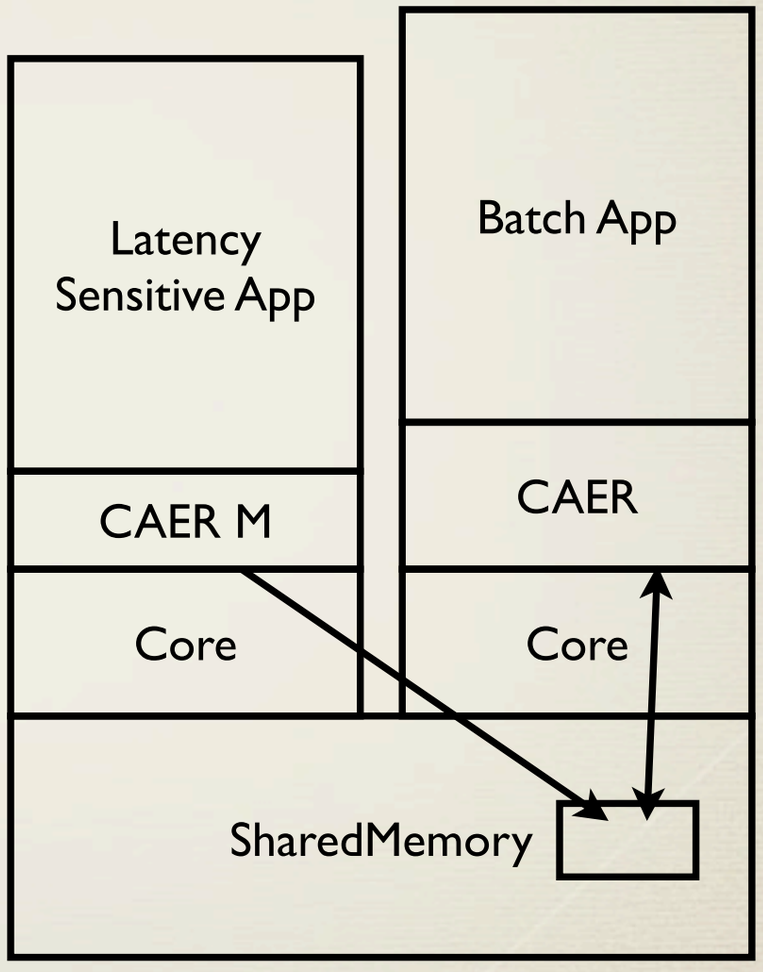
- Soft locking
  - When contention detected, lock cache
  - Continue monitoring
  - Release lock, when LLC demand subsides

Outline

- ✓ Problem / Motivation
- ✓ CAER overview
- ✓ Achieving detection and response
  - ✓ Burst shutter approach
  - ✓ Rule based approach
- Evaluation
- Conclusion

Evaluation

- Implemented CAER prototype for current Multicore
- Supports a batch and latency sensitive applicaton
- All experiments run on Core i7 (Nehalem)
- Spec 2006 suite (C/C++)



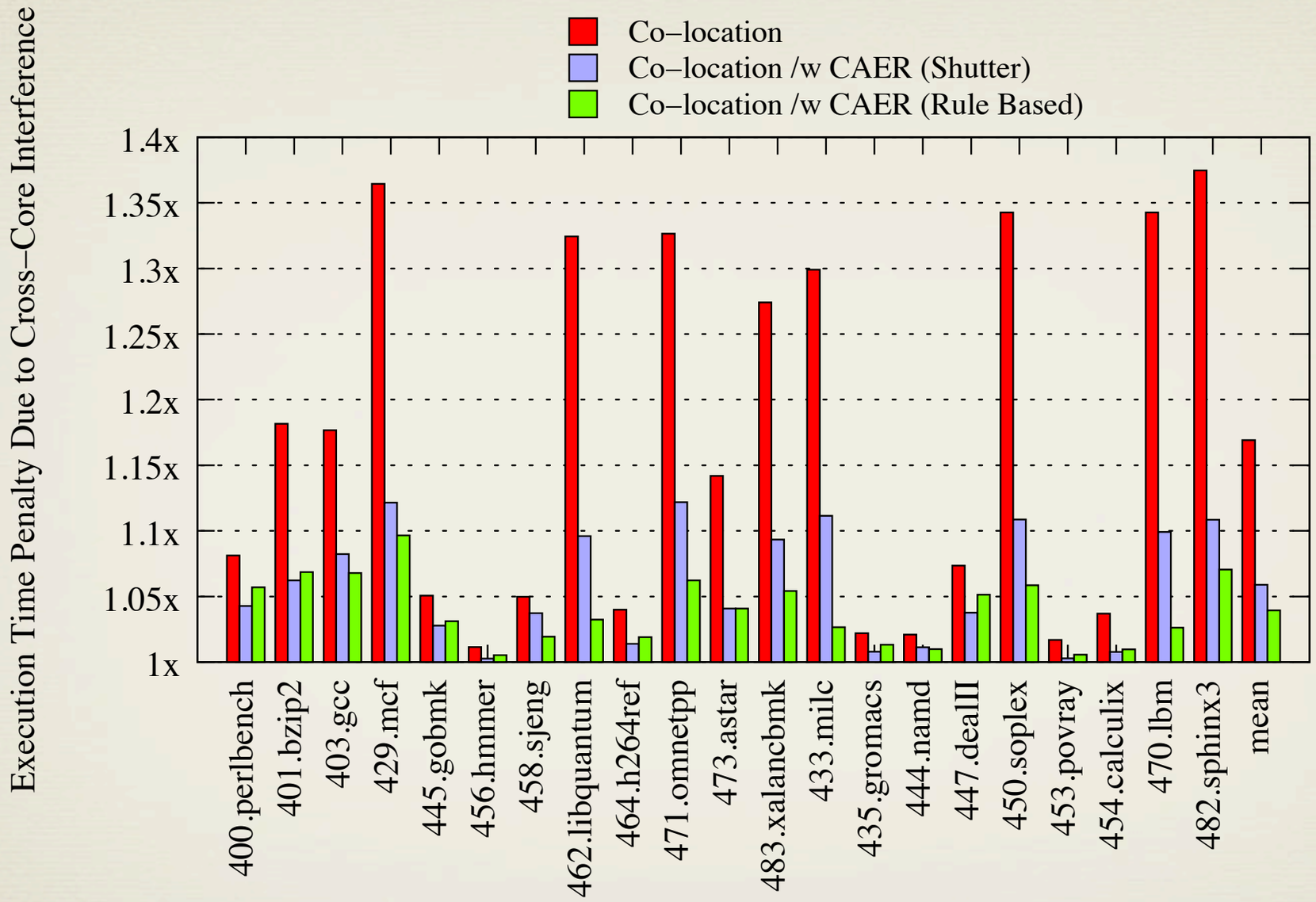
CAER Prototype

Evaluation

- **Goals:** Provide contention detection and response approach to reduce cross-core interference and improve utilization

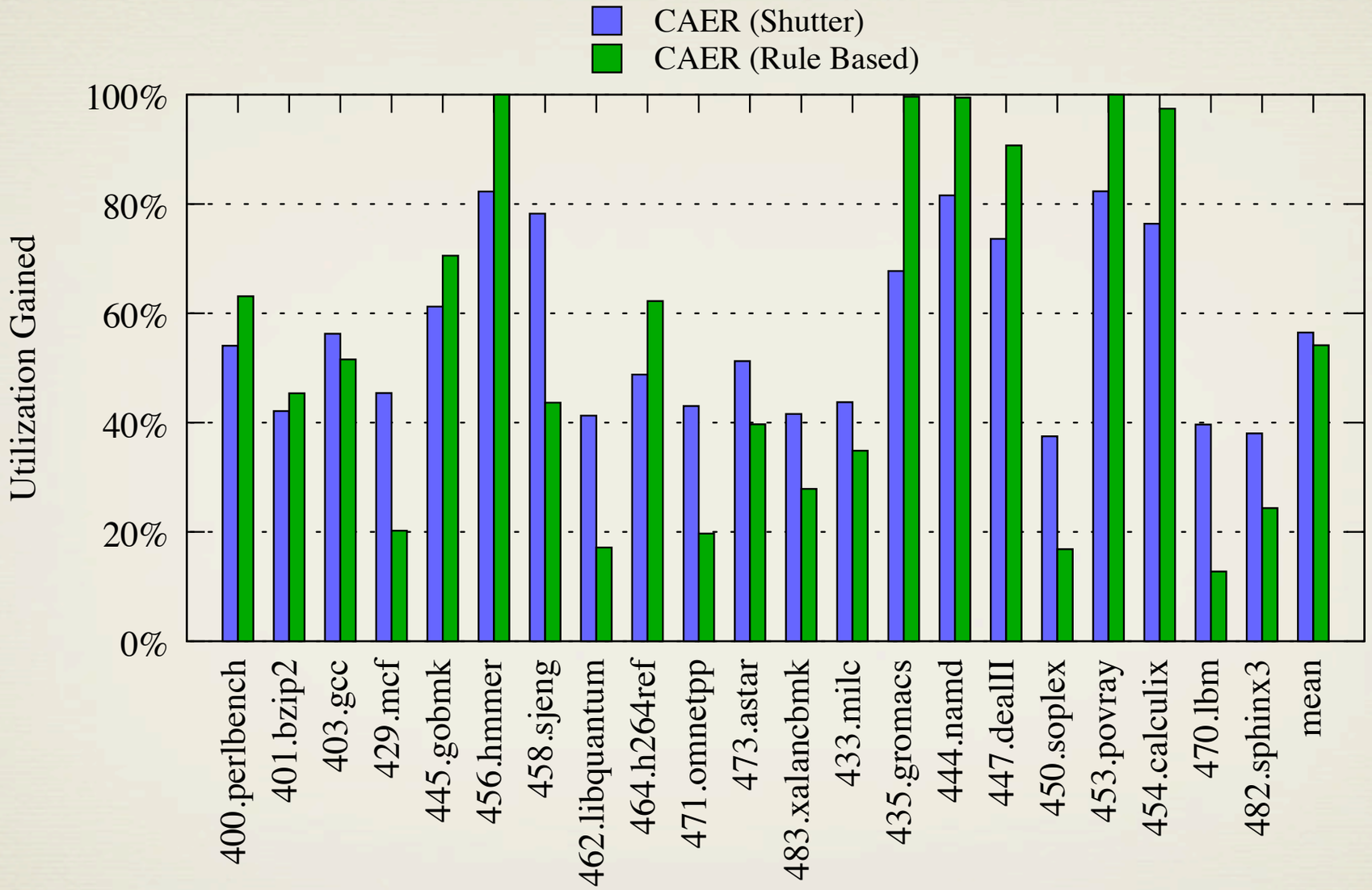
1. Impact of Co-location with CAER
2. Utilization gained when using CAER

Evaluation



- Impact of Allowing Co-location with CAER
- Lower is Better

Evaluation



- Utilization gained from CAER Co-location
- Higher is better

Take-away Insights

- HPM allow very lightweight online monitoring
  - No need for instrumentation
  - No need to ‘control’ execution
- Online adaptation sometime necessitates an empirical approach
  - Try something, measure the effects, react accordingly

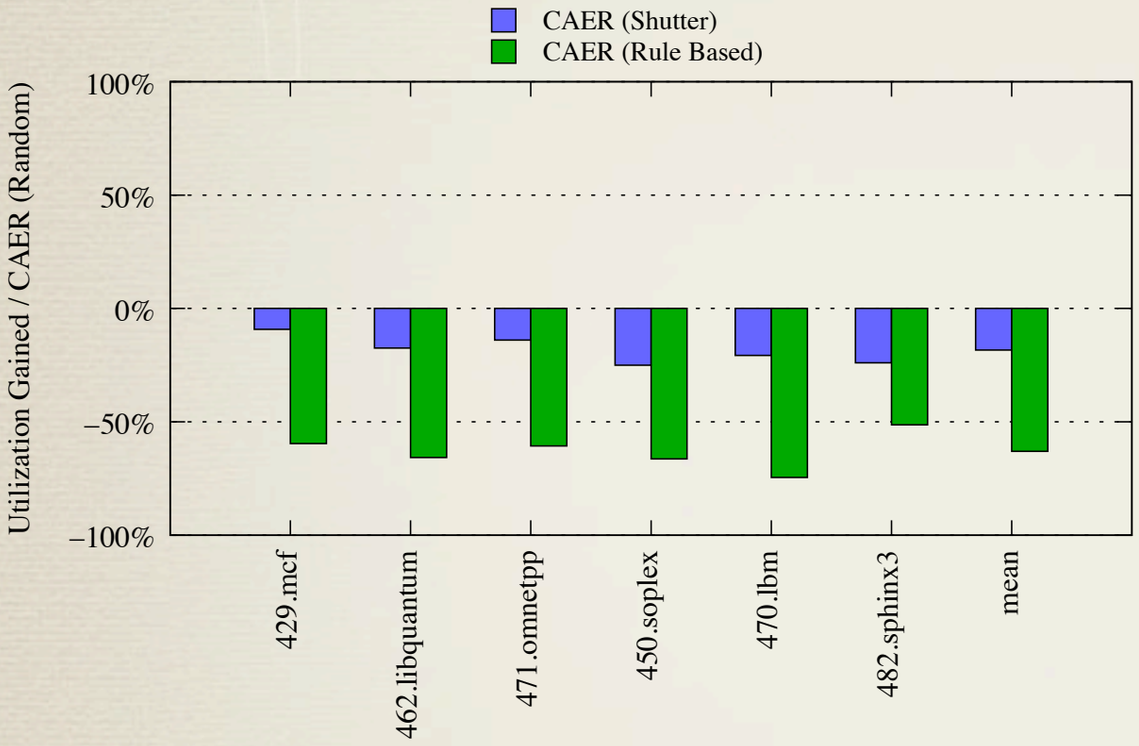
Summary and Conclusion

- Cross core interference poses significant challenge
- QoS, latency sensitive applications in data-center
- We *Can* detect and respond current commodity CMPs
- We show how, using CAER
- Much more details in the paper
- Brings performance interference degradation from 17% to just 4% while gain ~58% utilization of neighboring core

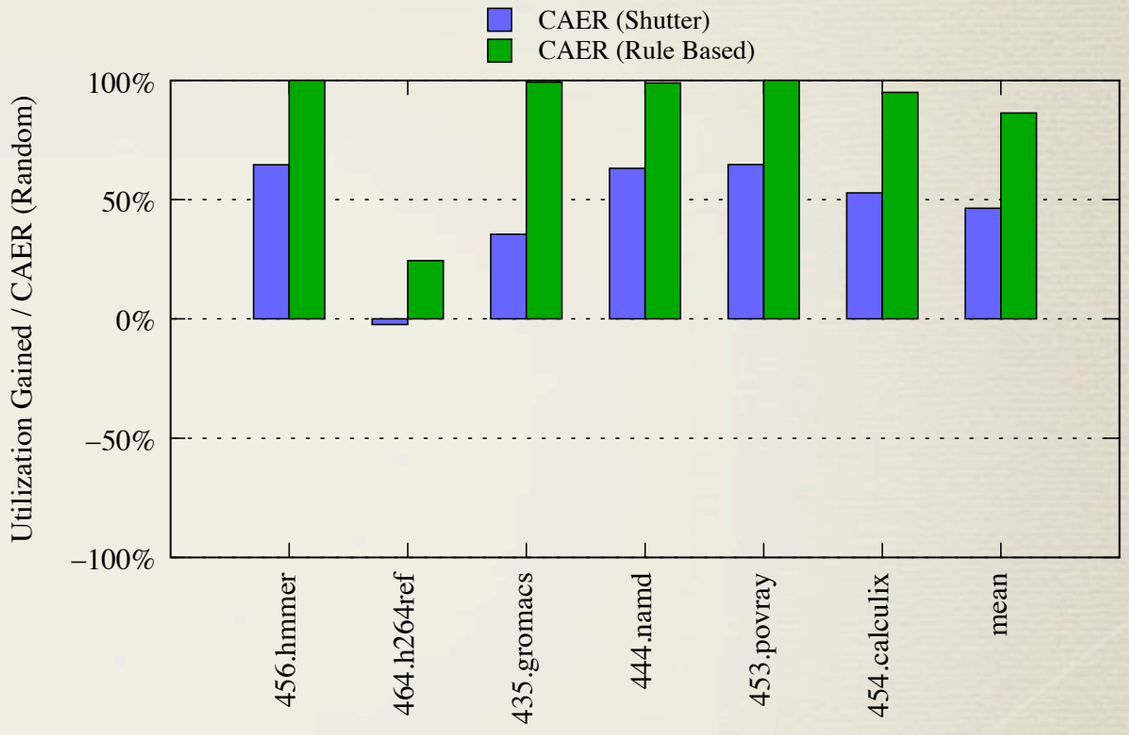
Questions?

Accuracy

Sensitive



Insensitive



- Baseline: (used to illustrate contention detection accuracy)
  - Random heuristic (Correctly identifies contention 50% of the time)
  - Sensitive applications vs insensitive applications