Uppsala Programming for
Multicore Architectures
Research Center

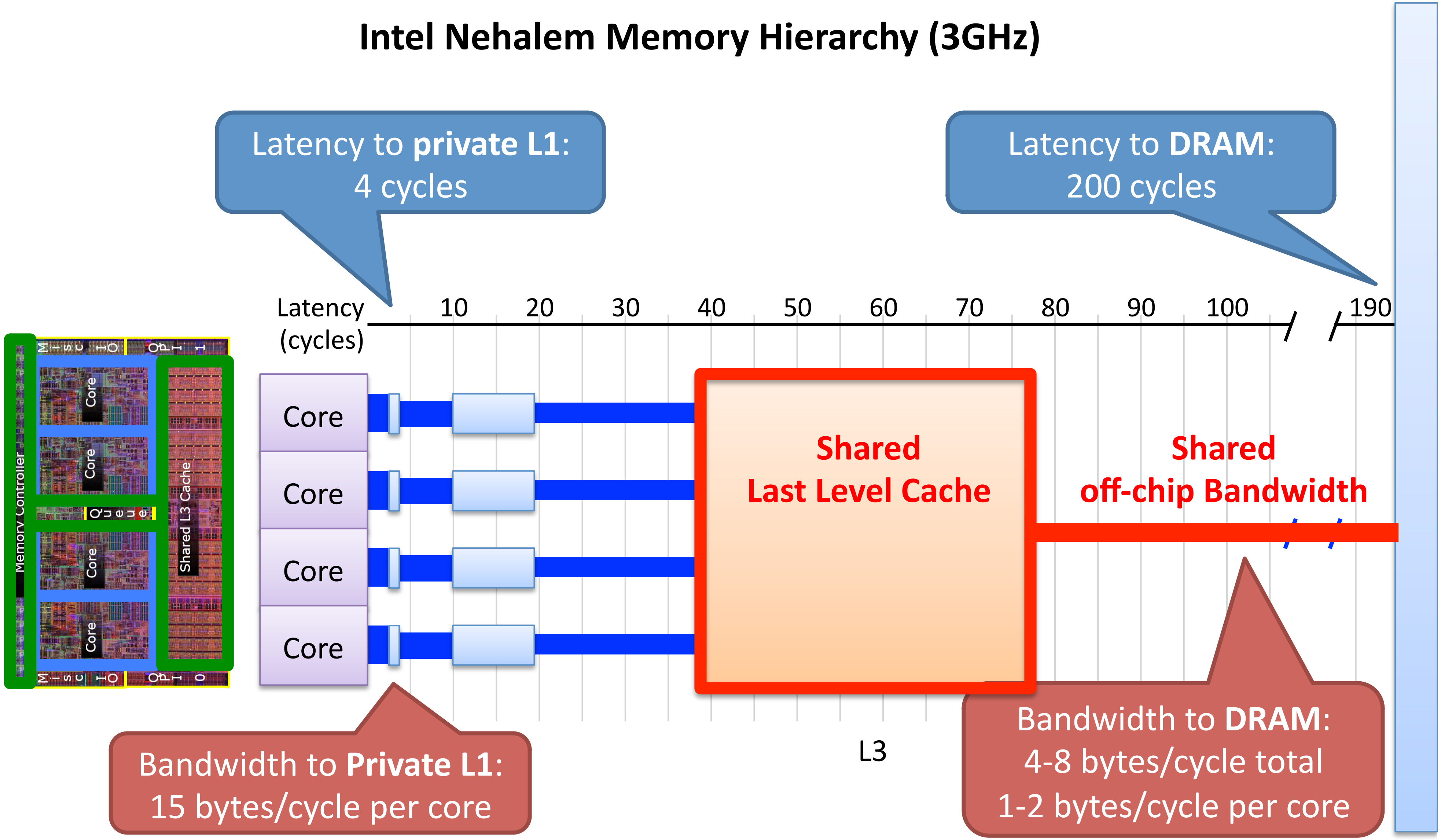
Understanding Application Sensitivities: The Key to Shared Resource Modeling

David Black-Schaffer

Assistant Professor, Department of Information Technology
Uppsala University

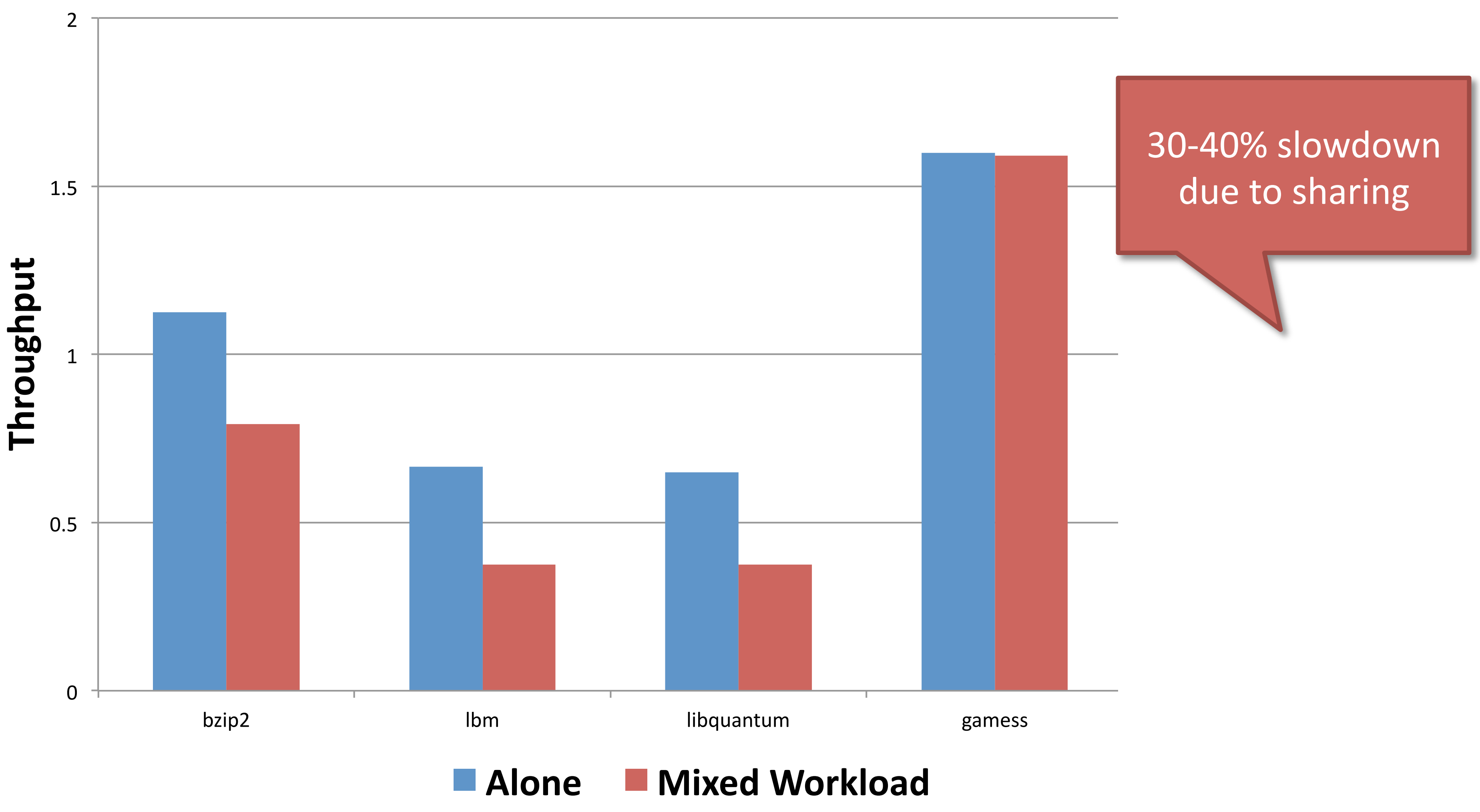
Special thanks to the **Uppsala Architecture Research Team**:
David Eklöv, Prof. Erik Hagersten, Nikos Nikoleris, Andreas Sandberg, Andreas Sembrant

Multicore Memory Systems



D. Molka, et. al., *Memory Performance and Cache Coherency Effects on an Intel Nehalem Multiprocessor System*, PACT 2009.

Impact of Resource Sharing

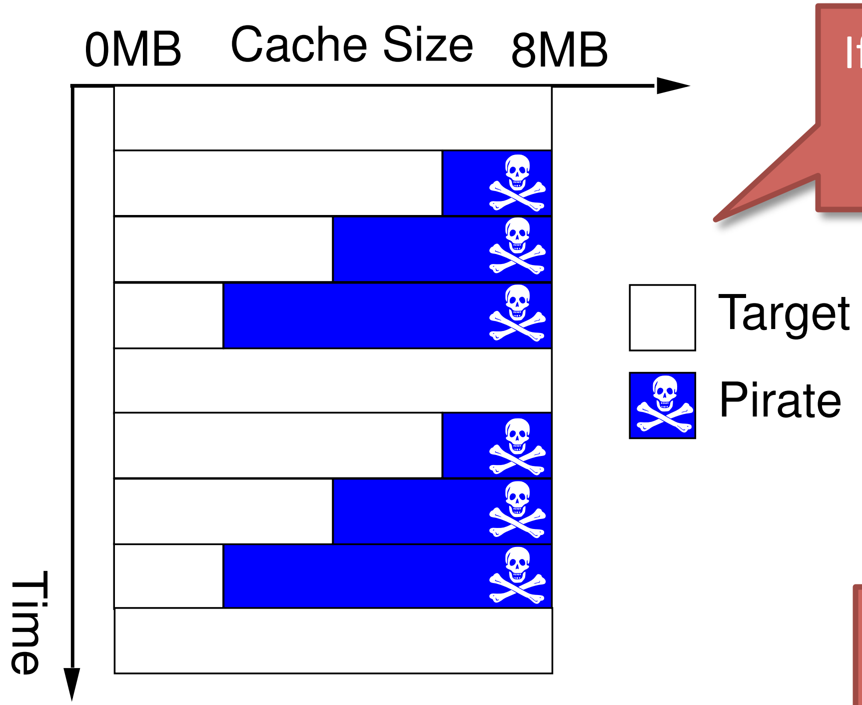


Measuring Shared Resource Sensitivity

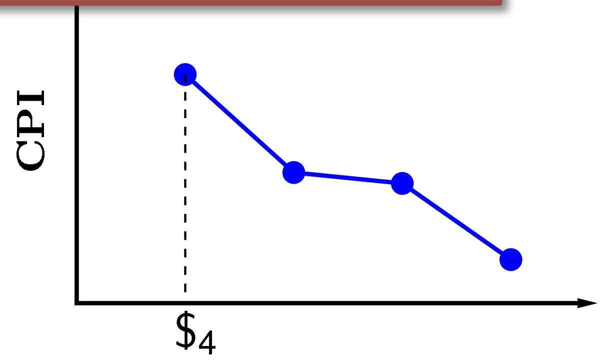
1. **Cache Pirate**
   – Measuring sensitivity to shared cache allocation
   – General technique for measuring sensitivity in real HW/SW
2. **Bandwidth Bandit**
   – Measuring sensitivity to shared bandwidth allocation
3. **Modeling Cache Usage**
   – Predicting shared cache allocation and performance impact
   – Use Cache Pirate data to include HW/SW complexities

1. Cache Pirating (David Eklöv)

- Measure cache sensitivity by stealing cache
  - **Steal cache** with a "Pirate" application
  - **Measure performance** of the Target
  - **Monitor the Pirate** to verify cache stolen

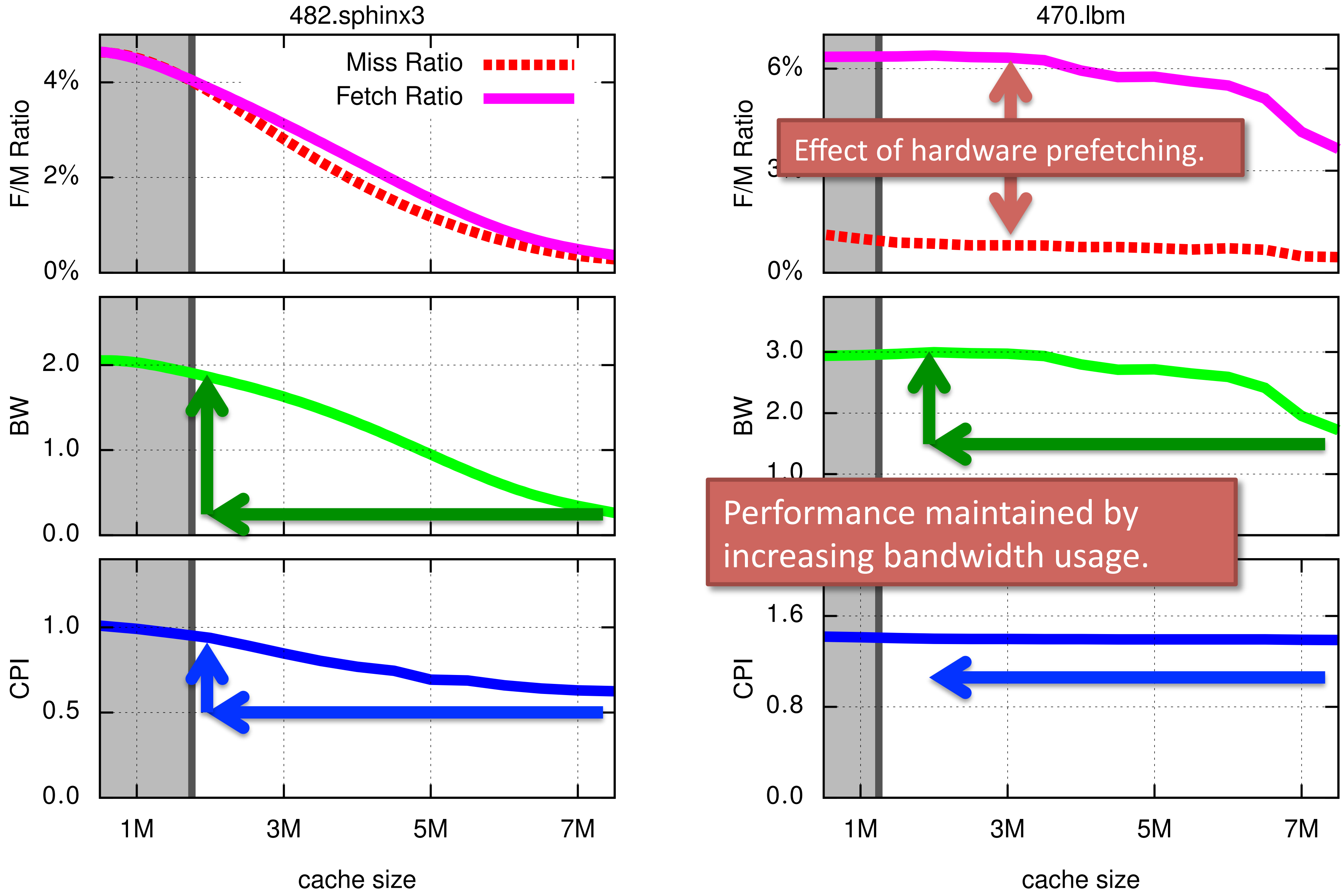


If the **Pirate misses** in the cache then we aren't stealing what we want.

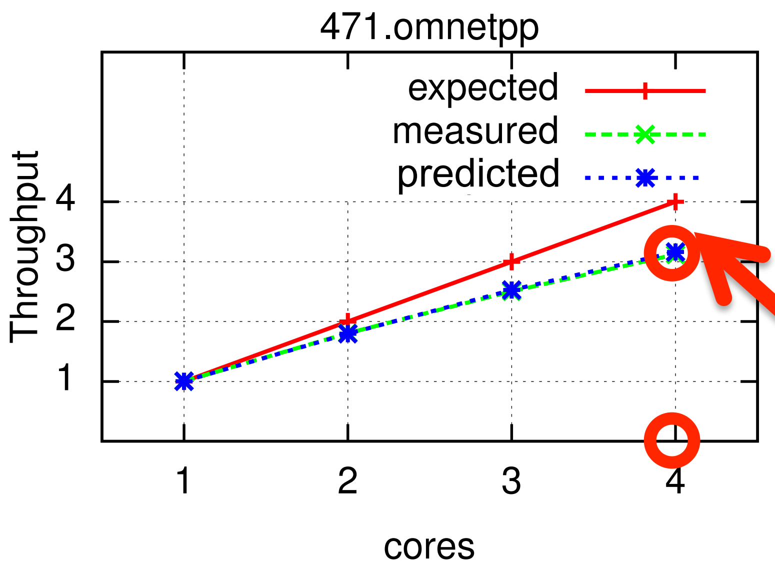


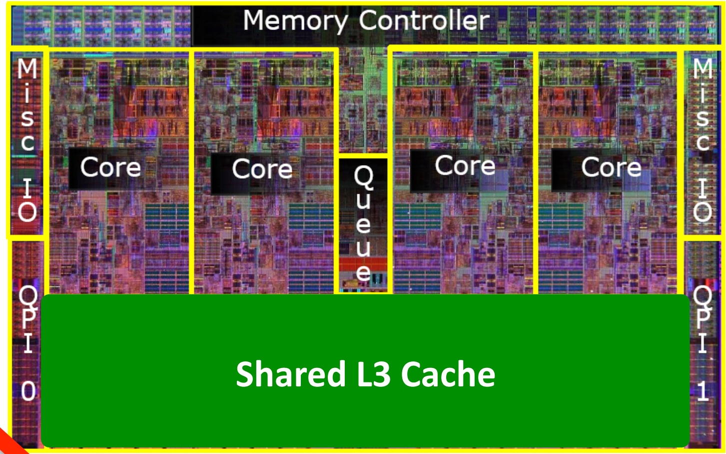
5% Overhead
Accurate: Includes all HW/SW effects

Application Cache Sensitivities



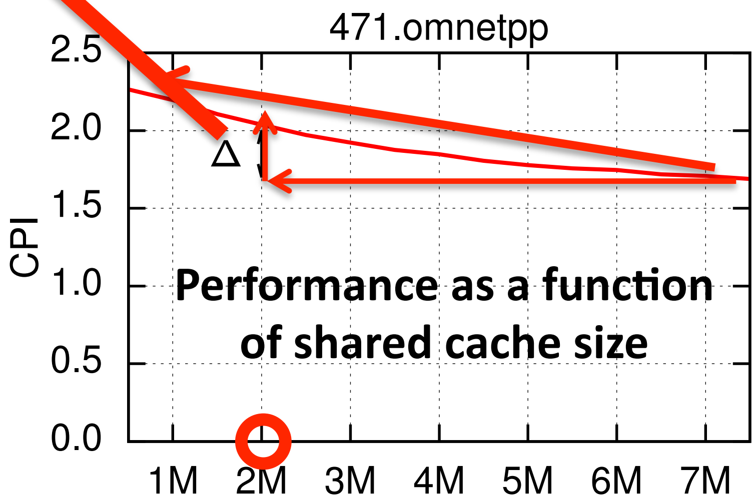
Predicting Multicore Scaling (Cache)





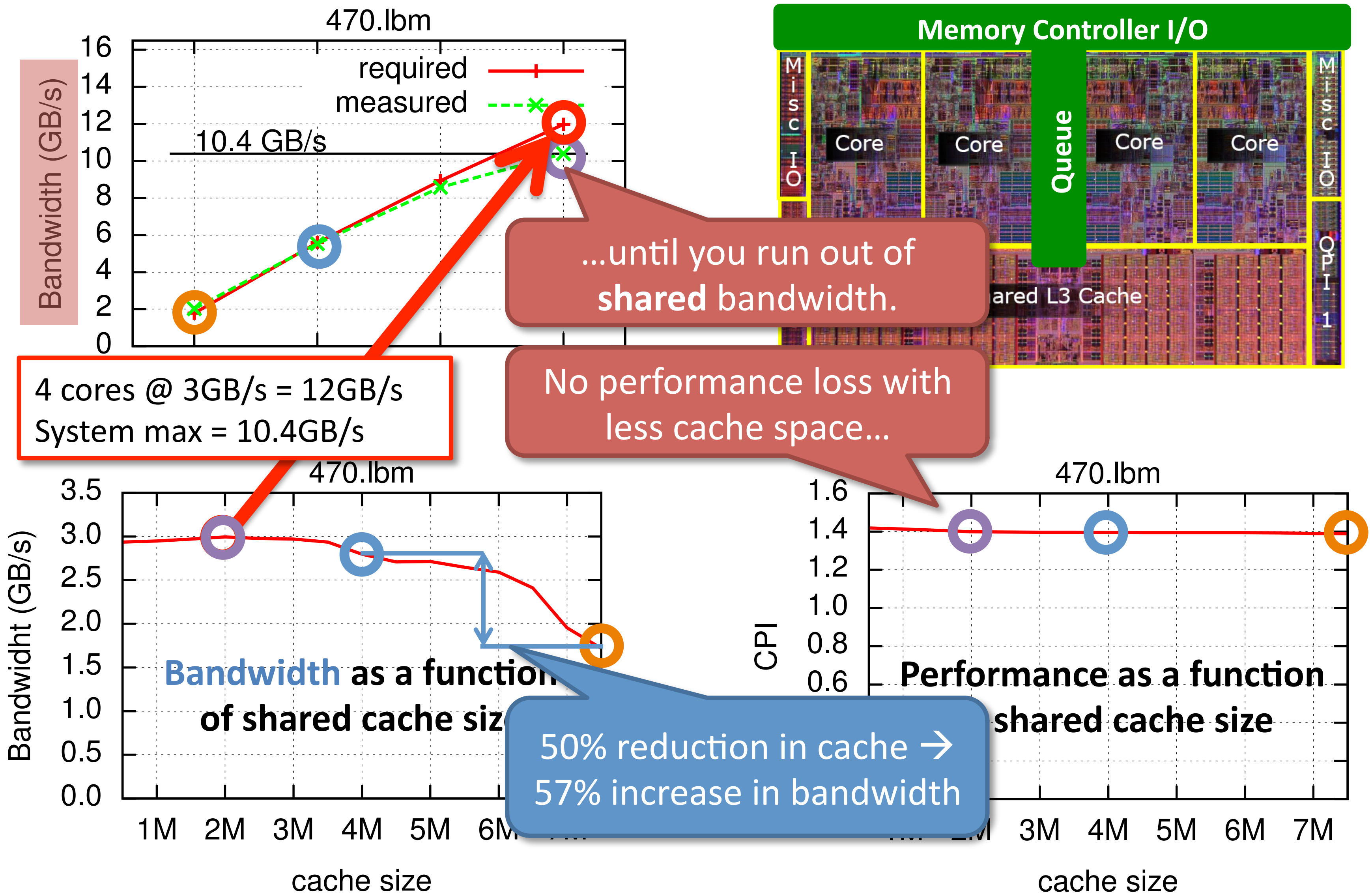
Experiment

- Run 1-4 **independent** instances of the same program on a 4-core Nehalem
- **Performance affected by shared cache**
  - ¼ of the shared cache → 20% slower



Profile performance as a function of shared cache
→ Predict multicore scalability

Bandwidth Limits

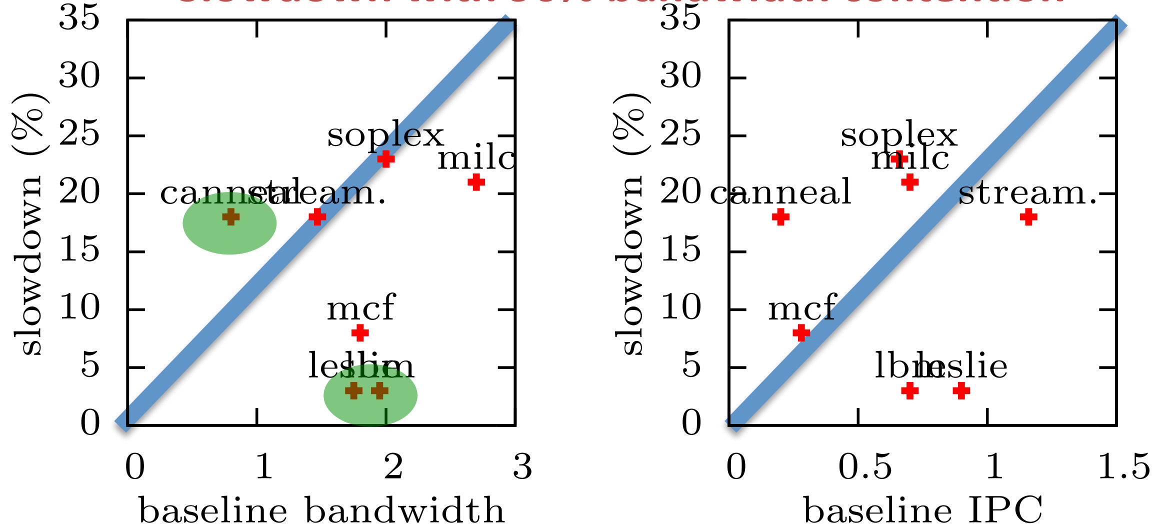


2. Bandwidth Bandit (David Eklöv)

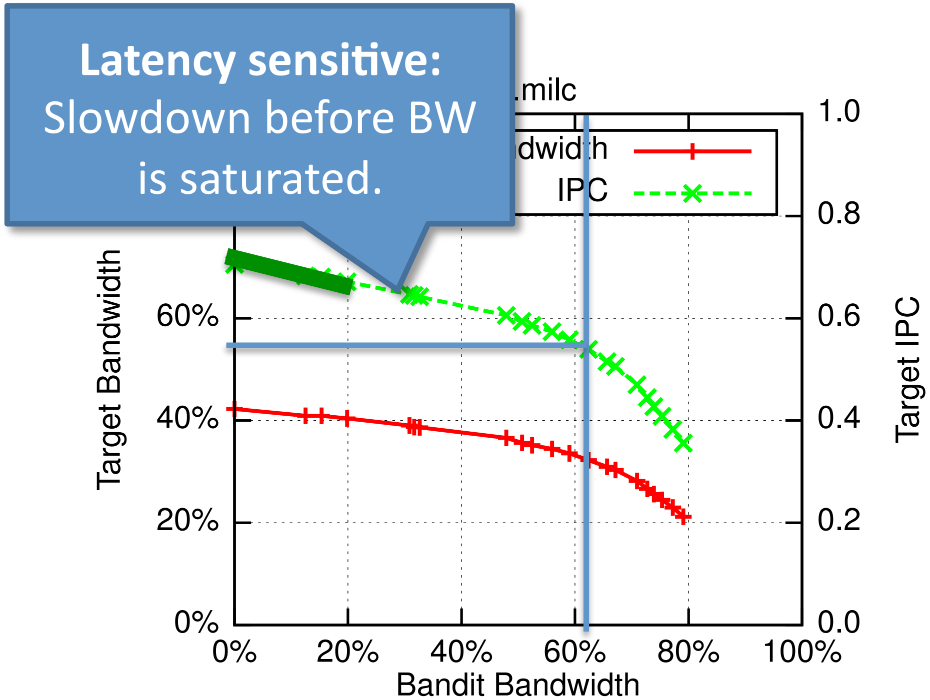
- Measure bandwidth sensitivity by stealing bandwidth
- More complex than Cache Pirating
  - Memory controller
    (access patterns, row buffers, re-ordering, page allocation)
  - Latency and throughput sensitivities

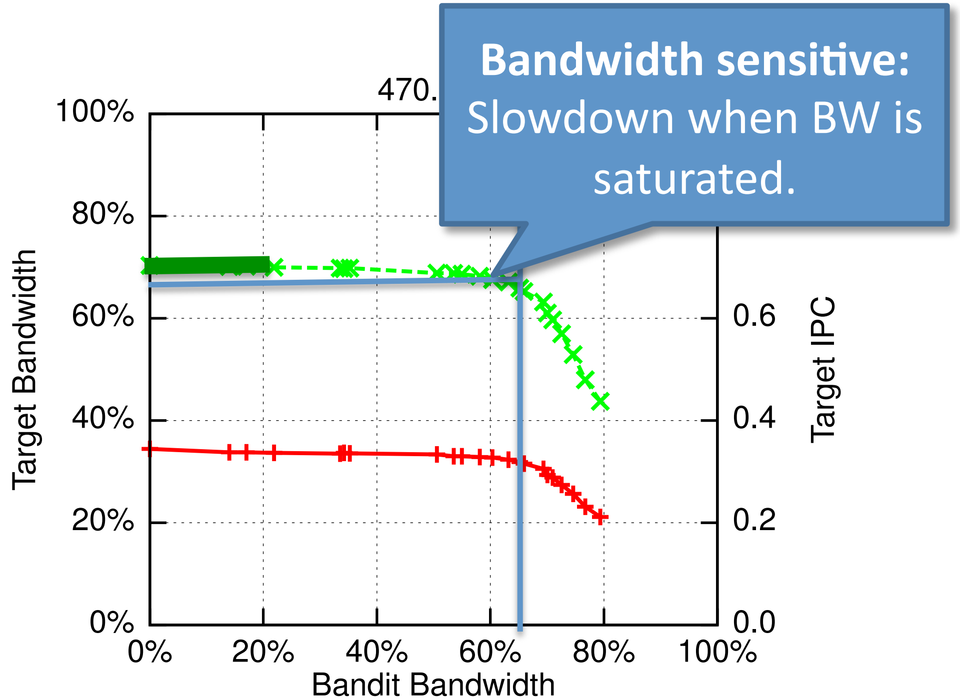
Insight: No correlation between bandwidth usage and sensitivity to bandwidth contention!

Slowdown with 90% bandwidth contention



Application Bandwidth Sensitivities



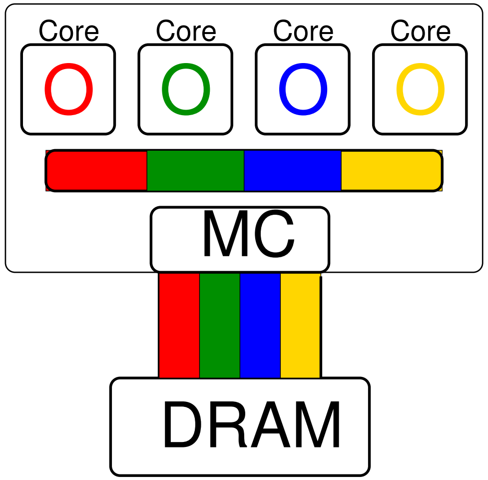


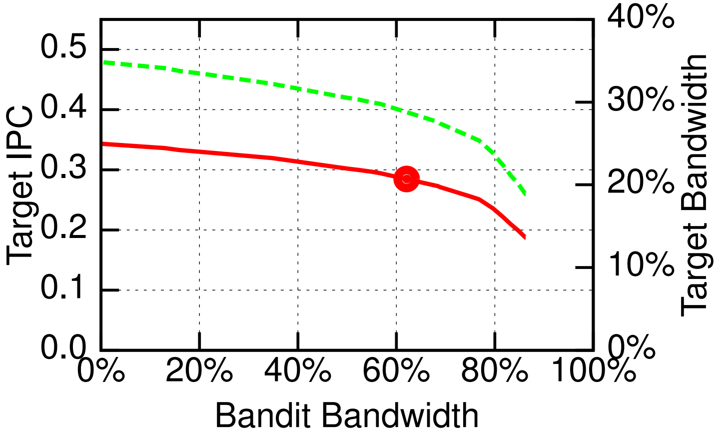
- Significant variation in application sensitivities
- Leads to different impact of resource sharing

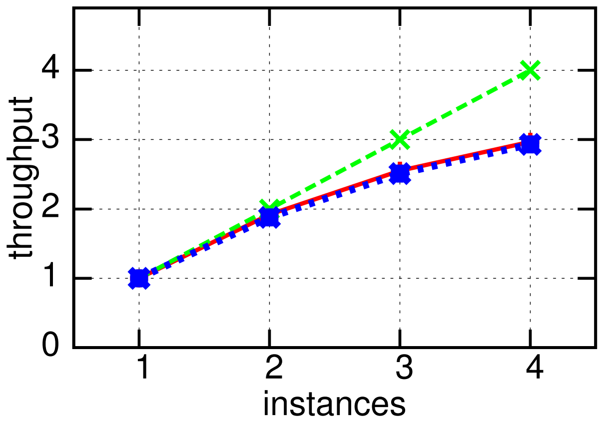
Predicting Multicore Scaling (BW)

- Run 4 instances of the same application

  (Page coloring to guarantee no cache interference. Each gets exactly ¼ of the cache; BW effects only)





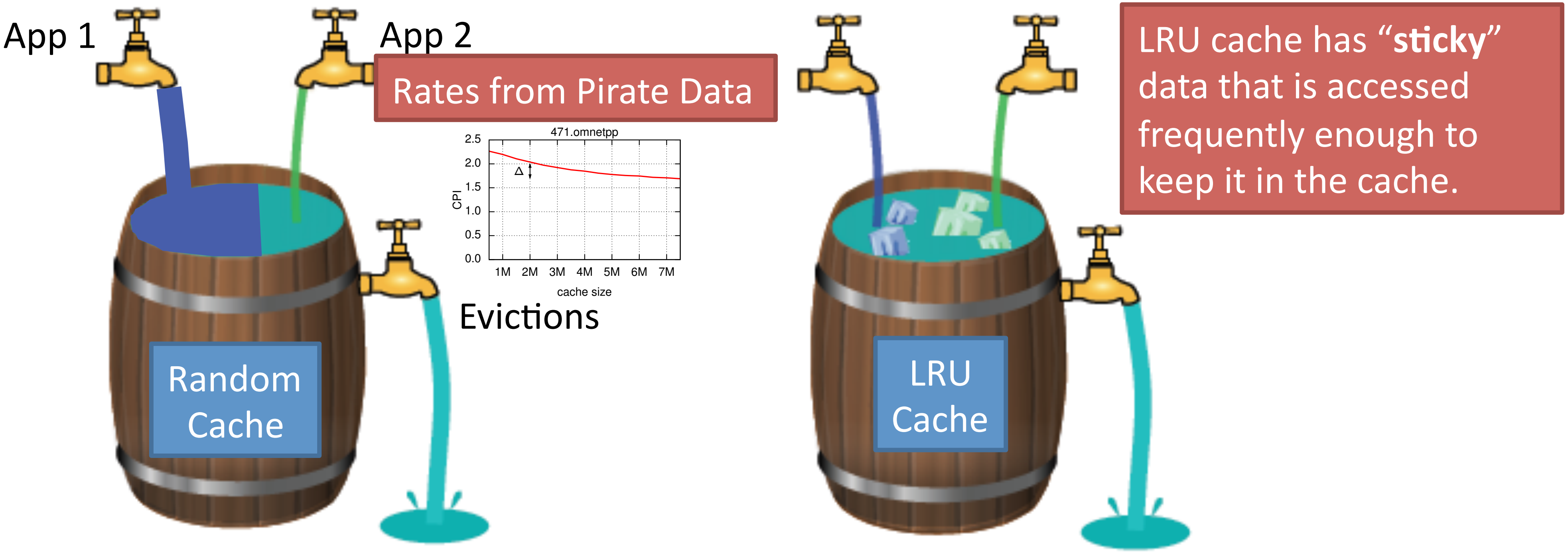


Profile performance as a function of shared bandwidth
→ Predict multicore scalability

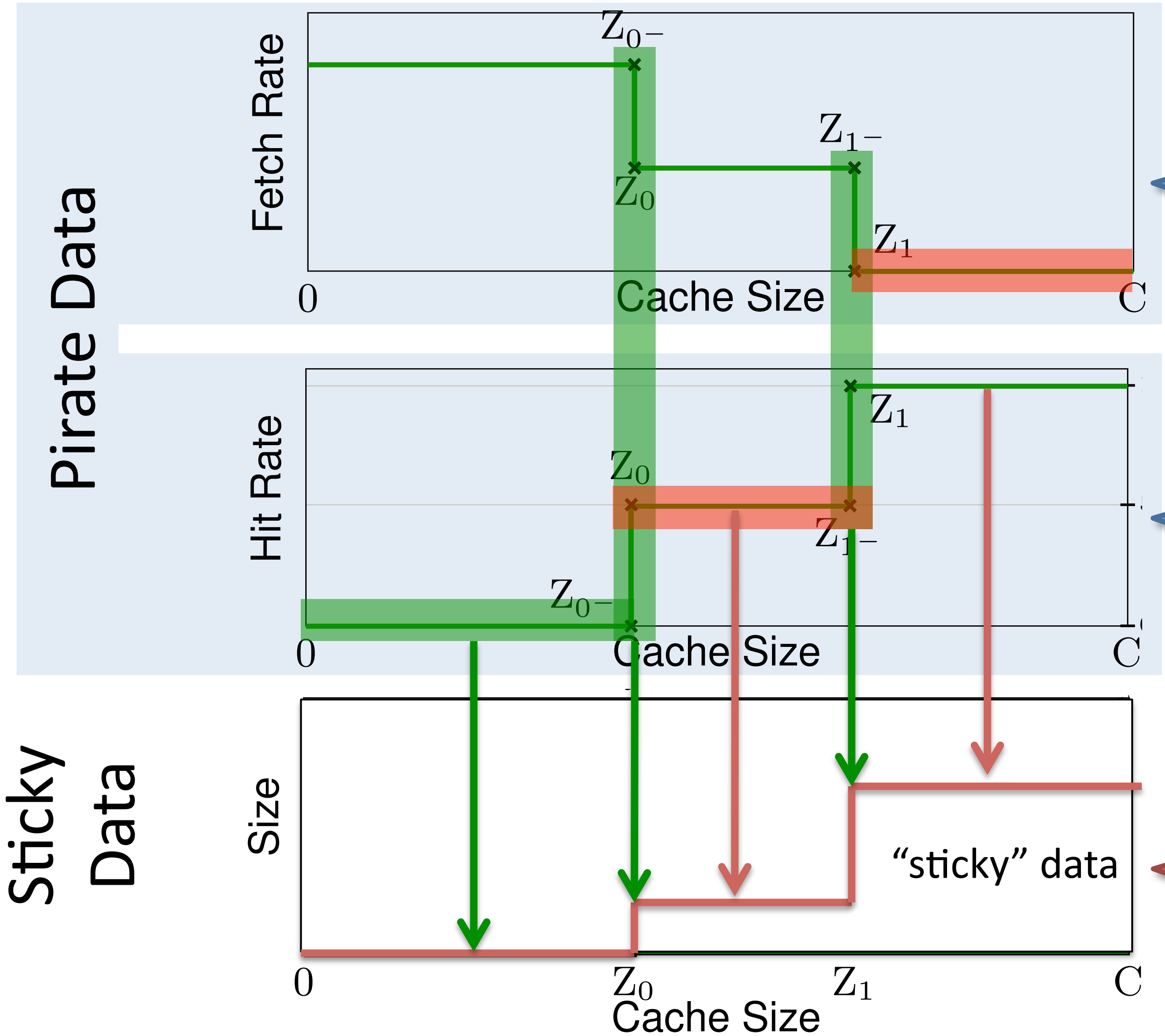
USING SENSITIVITY MEASUREMENTS TO MODEL SHARING

Modeling Cache Usage (Andreas Sandberg)

- Use Pirate data to model caches
- Caches have inflows and outflows
  - At steady state these are equal
  - Relative flow rates are proportional to the content of the cache



Pirate Data → Cache Contents



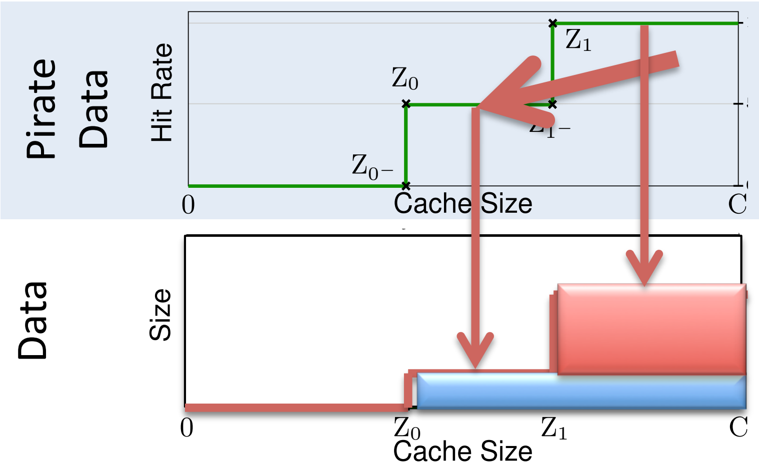
Gives us the **data set sizes**

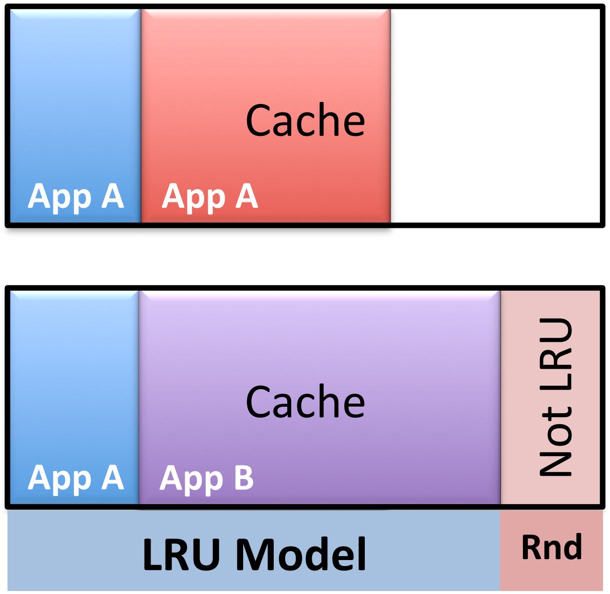
Gives us the **intensity of reuse** in each data set

Have sticky data **sizes** and **reuse intensities**

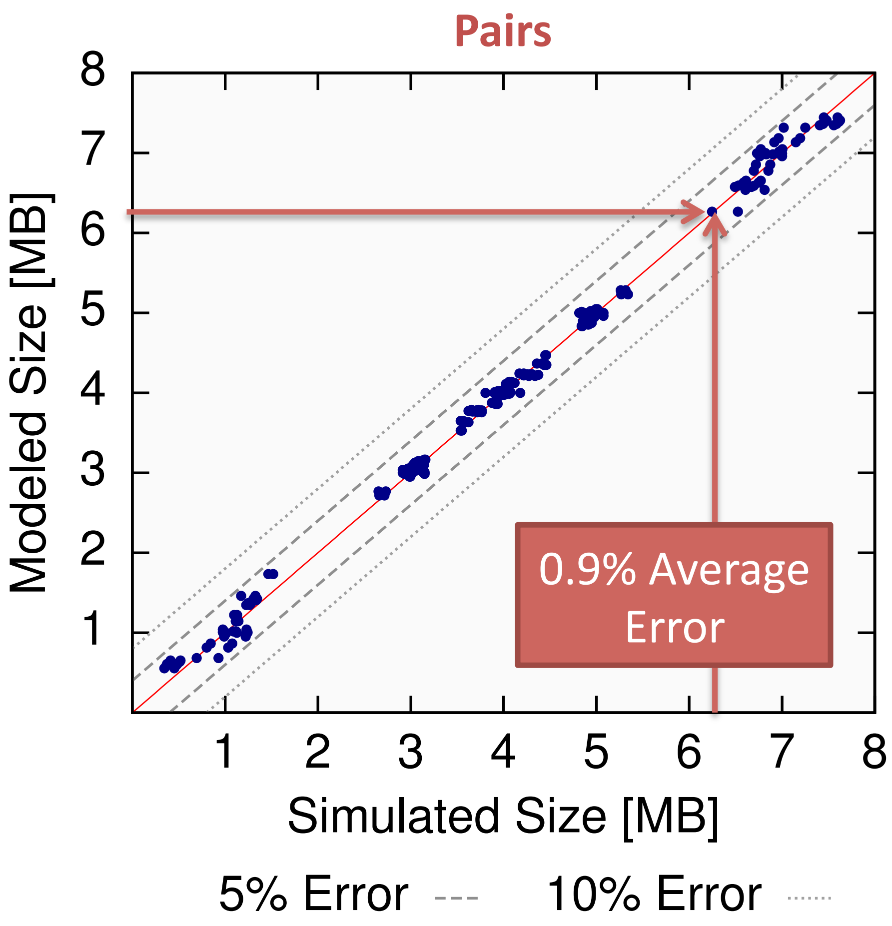
Fighting for Space in the Cache

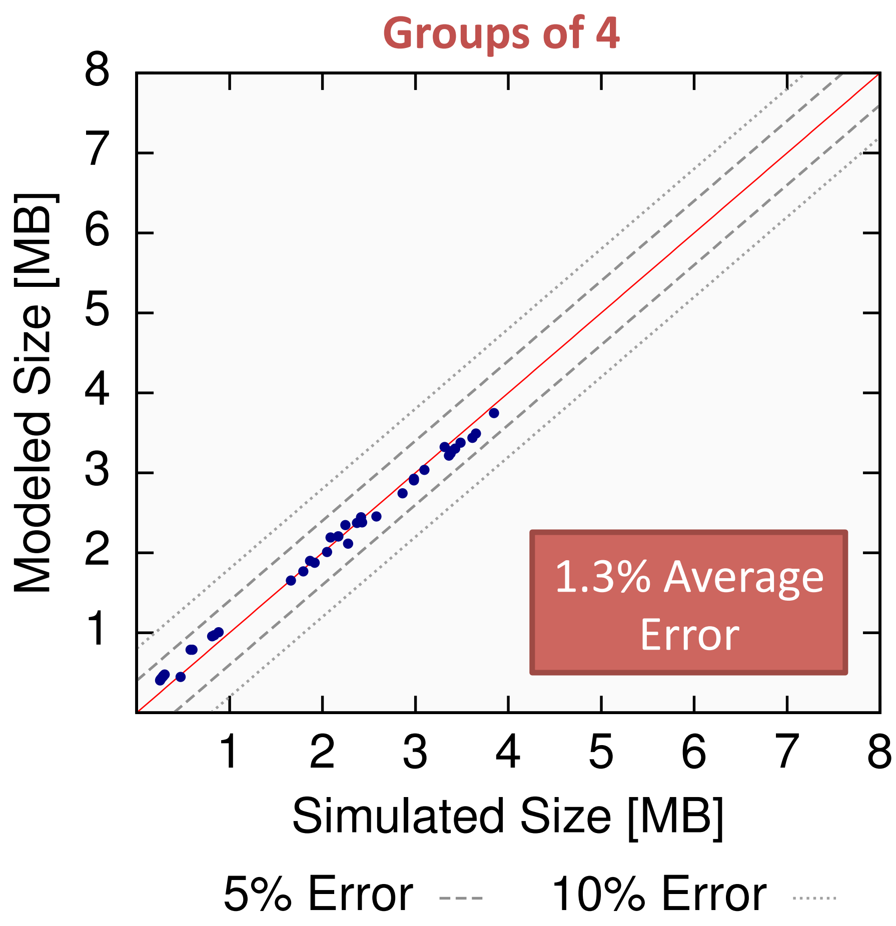
- We know the **sizes** and **access intensities** of each application's data sets
- Applications **fight for cache space** based on intensity of access (LRU)
- The data with the greatest access **intensities** stays in the cache
- If the data set won't fit, then it's reuse effectively goes away (data is evicted before it is used again)





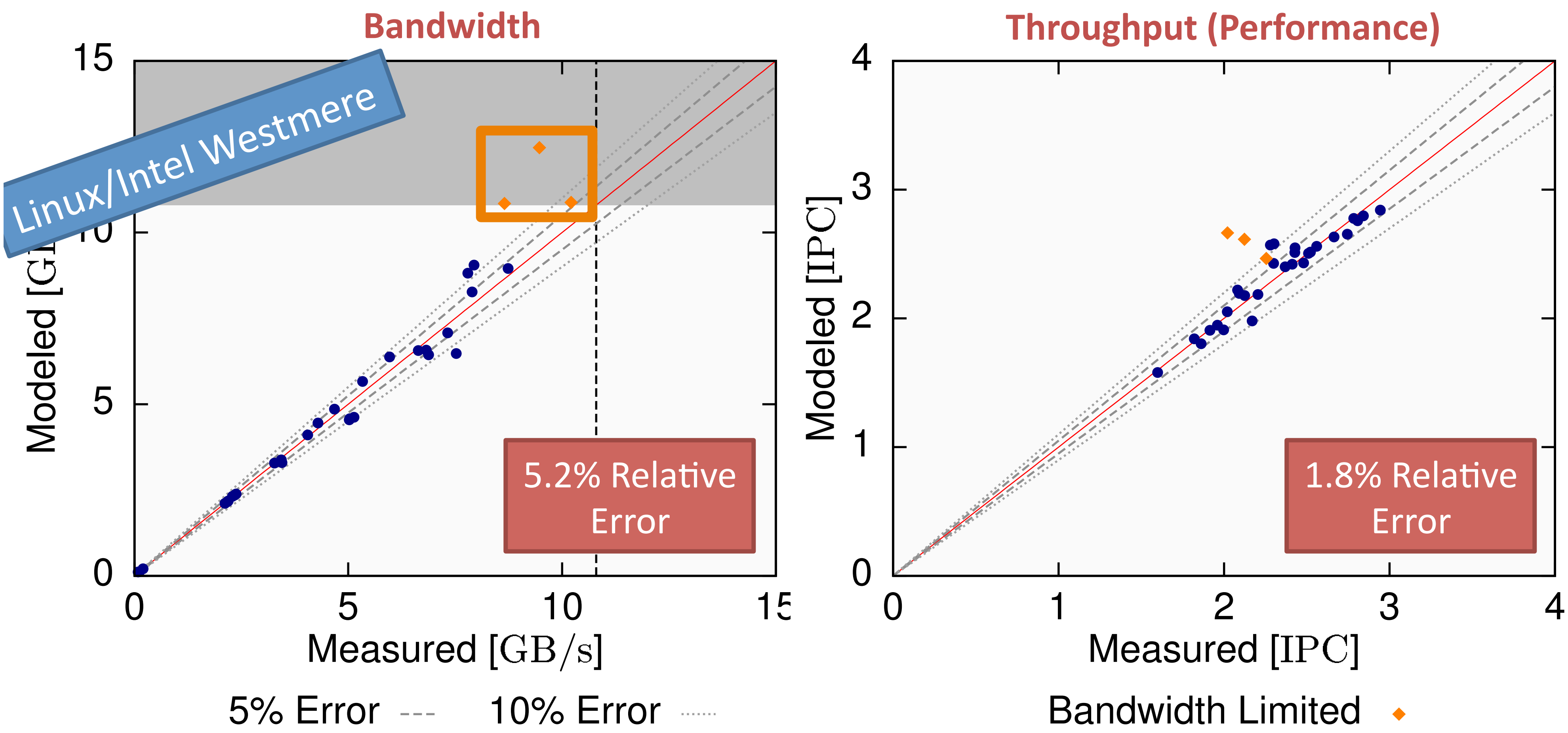
Predicting Shared Cache Usage





Simulator LRU

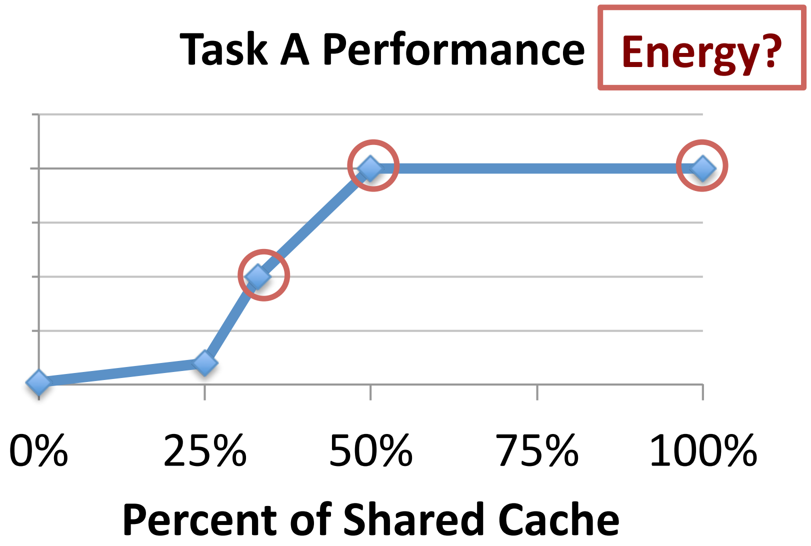
Predicting Throughput & Bandwidth

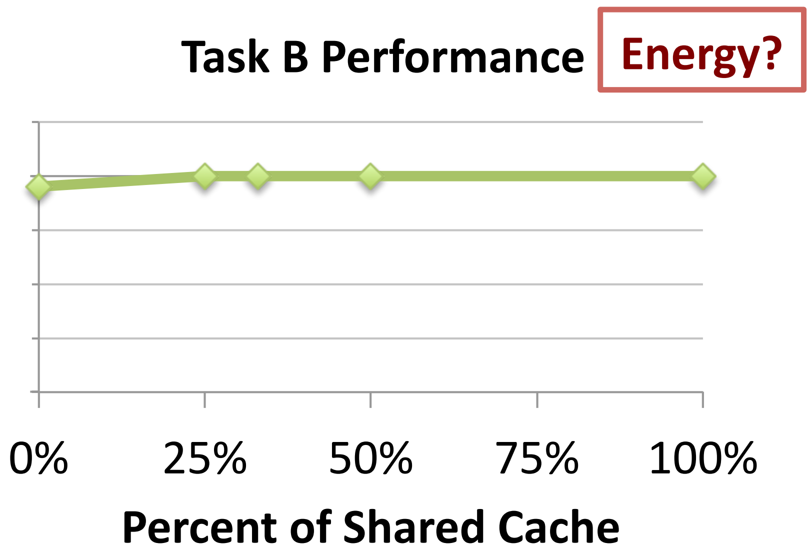


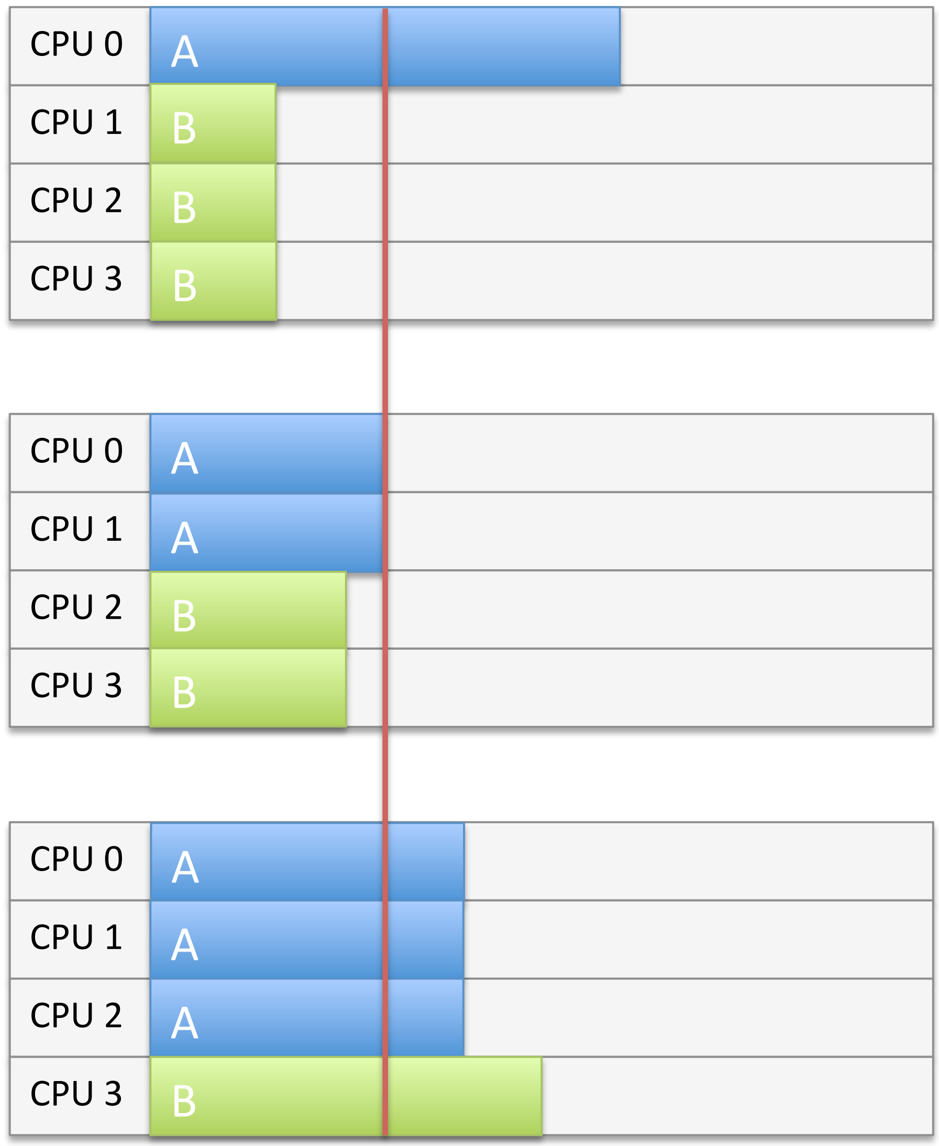
Individual profiles of performance as a function of shared cache
→ Predict workload scalability

APPLICATIONS TO TASK-BASED RUNTIMES

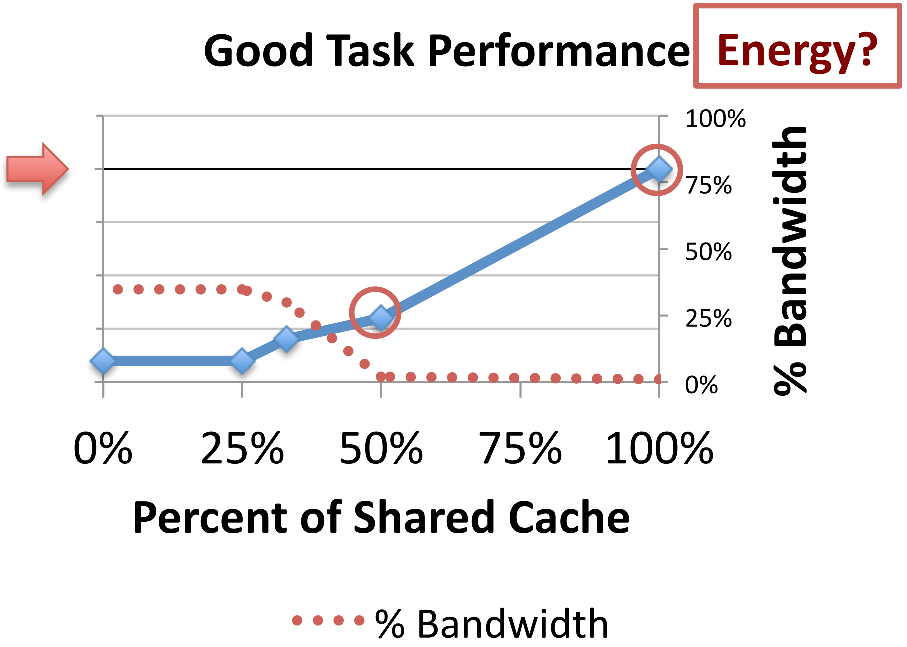
Understanding: Waste Cores

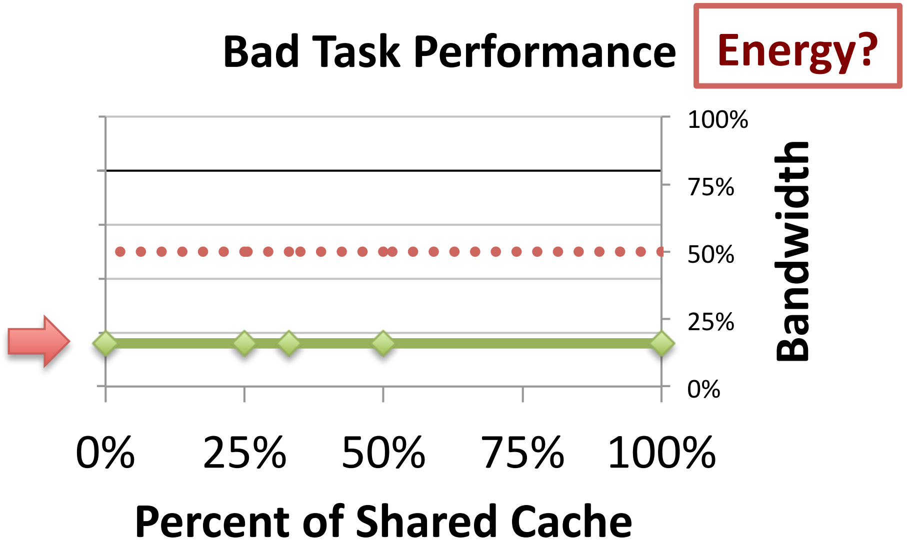


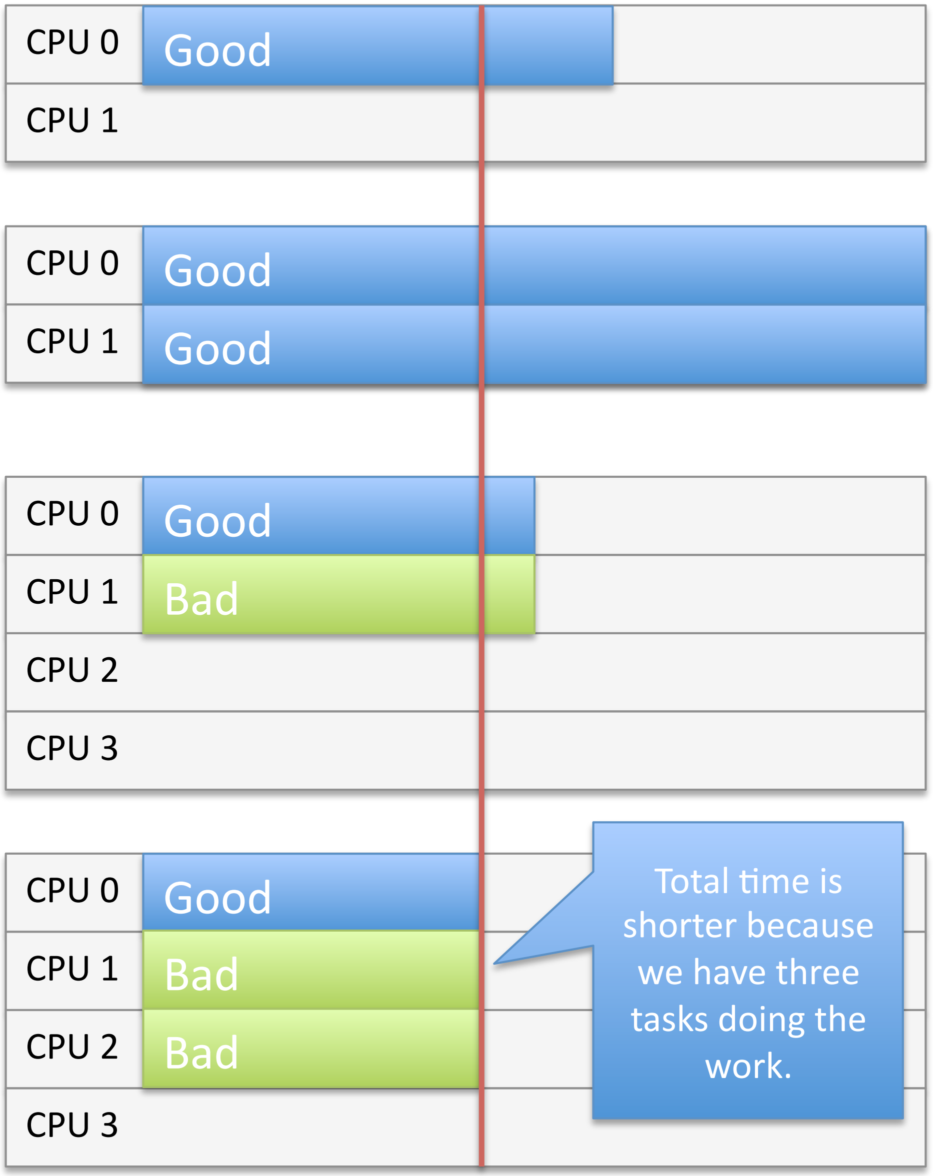




Adapting: Run Bad Code







Putting it Together

- **Profile tasks**
  - Need to run individually
  (Probably can't use other cores at the same time)
  - Can cache results for future runs
- **Predict performance**
  - Decide which tasks to run together
- **Adapt tasks**
  - Compiler/runtime interaction

- **Problems:**
  - Tasks sized for private caches → no shared resource use
  - Homogeneous tasks → little opportunity for scheduling
  - Combinatorial explosion → too many scheduling choices

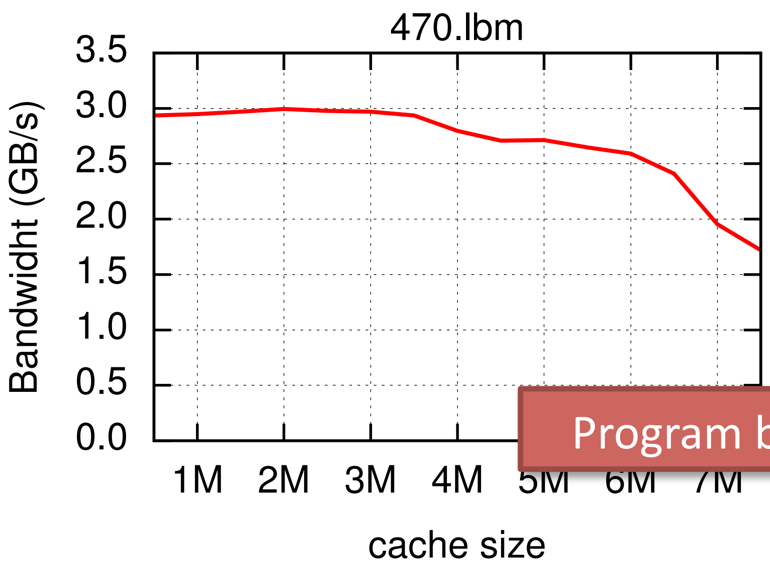
Acknowledgements

- **PhD Students:**
  - David Eklöv
    - StatStack, StatCC, Cache Pirate
  - David Eklöv and Nikos Nikoleris
    - Bandwidth Bandit
  - Andreas Sandberg
    - Cache Pollution, Cache Sharing Models
  - Andreas Sembrant
    - Phase Detection, Phase Memory Modeling
- **Colleagues:**
  - Erik Hagersten
  - Stefanos Kaxiras

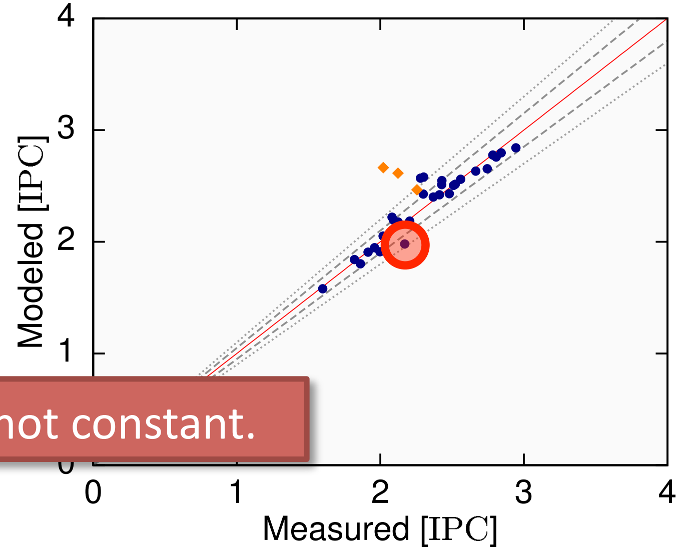
QUESTIONS?

PHASES

Bandwidth as a function of cache

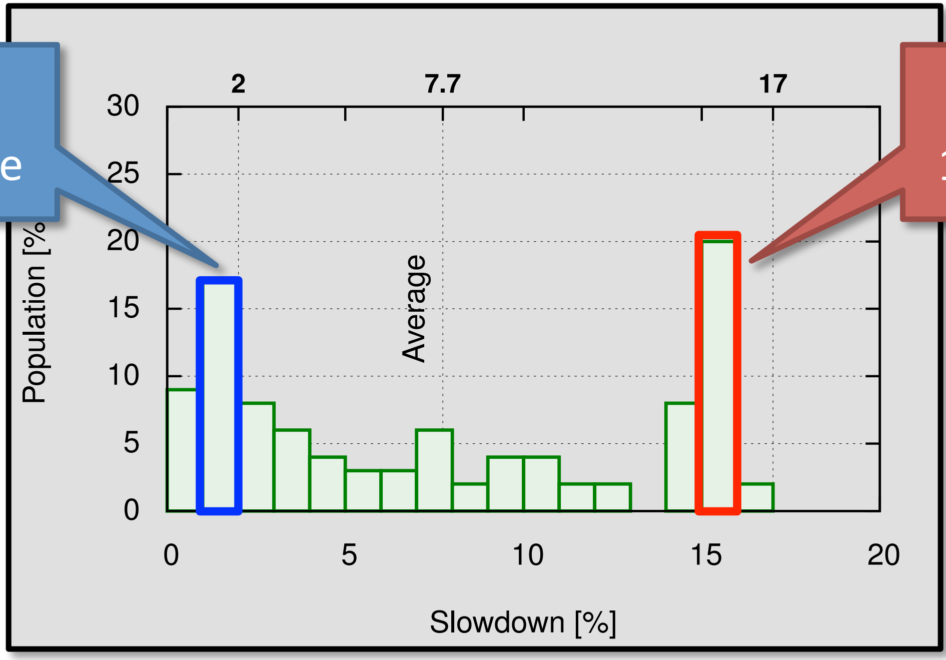


Throughput (Performance)



Program behavior is not constant.

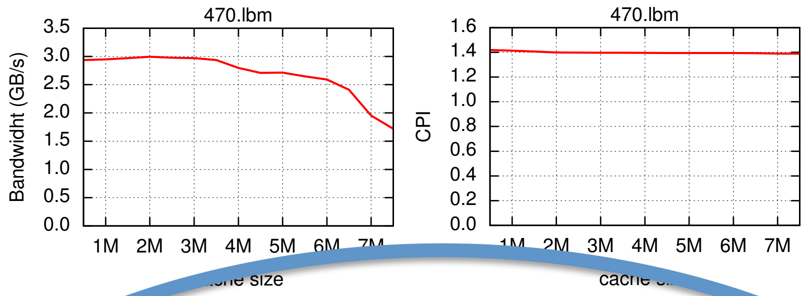
2% slowdown
17% of the time



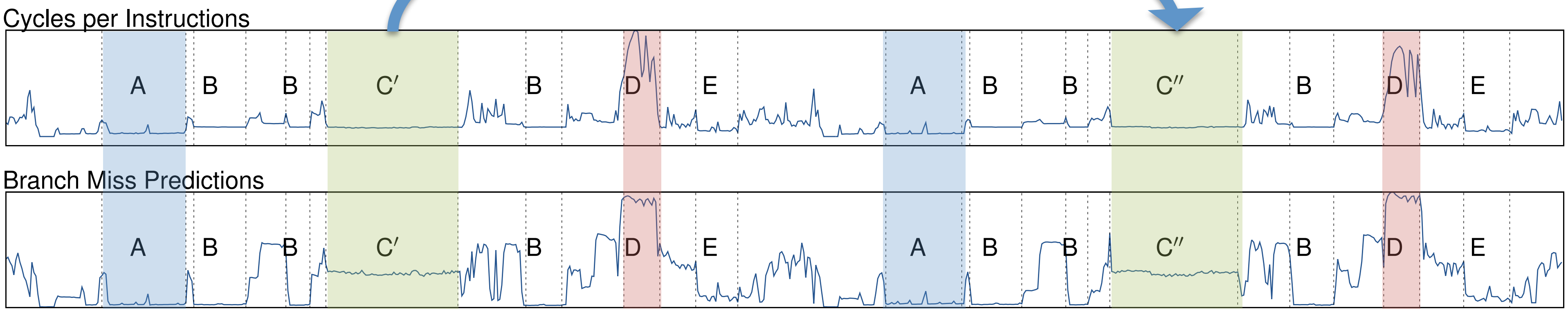
16% slowdown
15% of the time

Application Phases (Andreas Sembrant)

- Applications have time-varying behavior
- Need phase information for accurate insight
- With phase information we can do *smarter profiling*

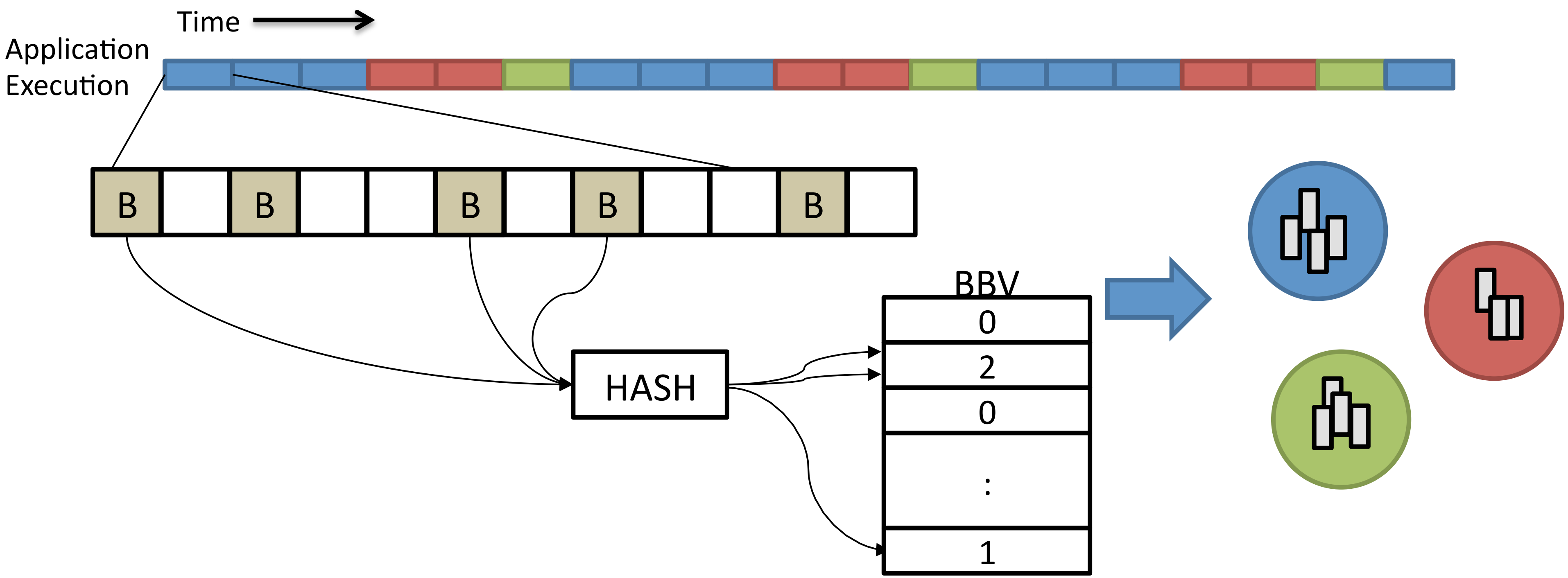


Average **Bandwidth** Average **CPI**



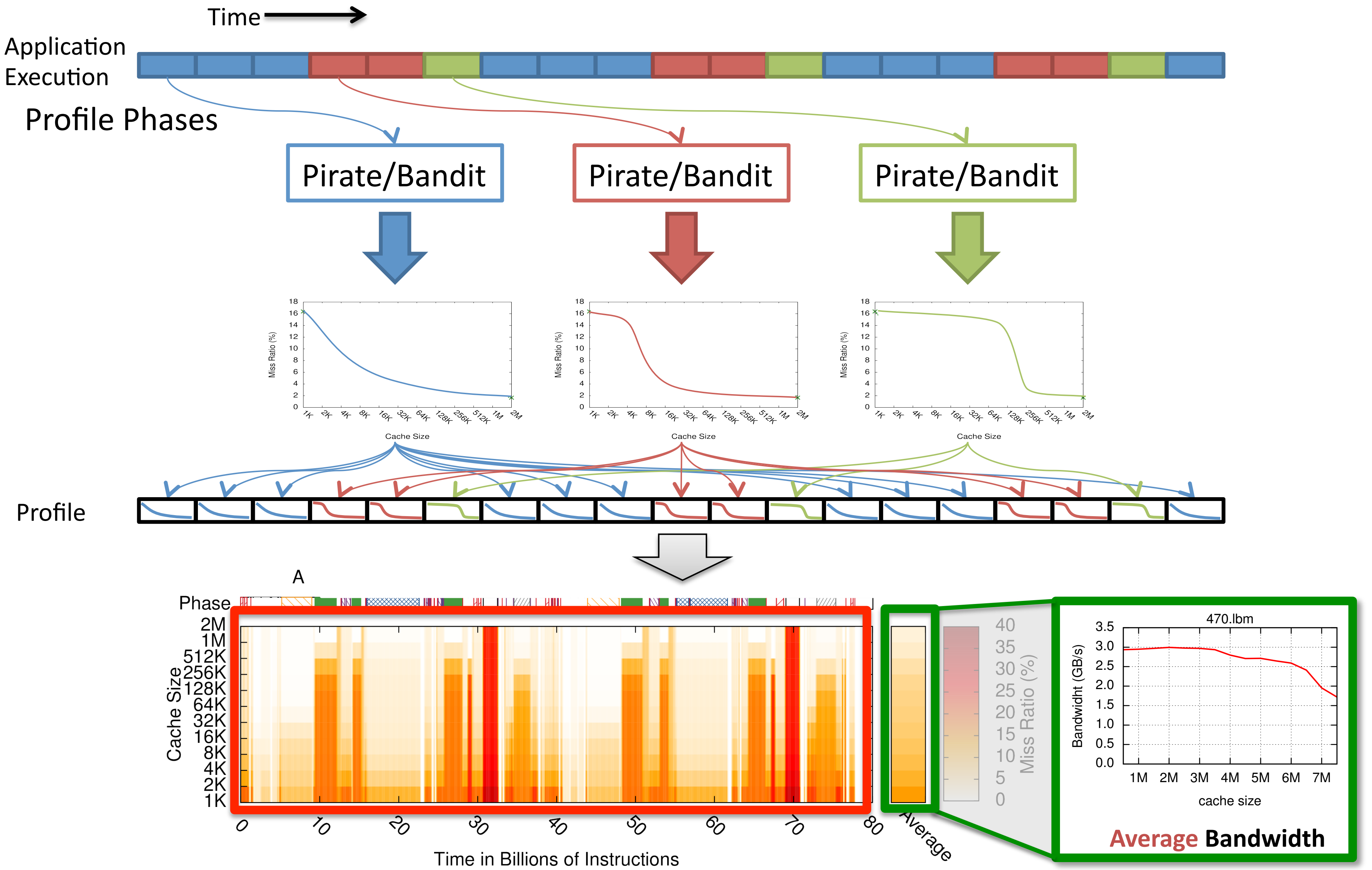
Phase Detection: ScarPhase

ScarPhase: Sample-based Classification and Analysis for Runtime Phases

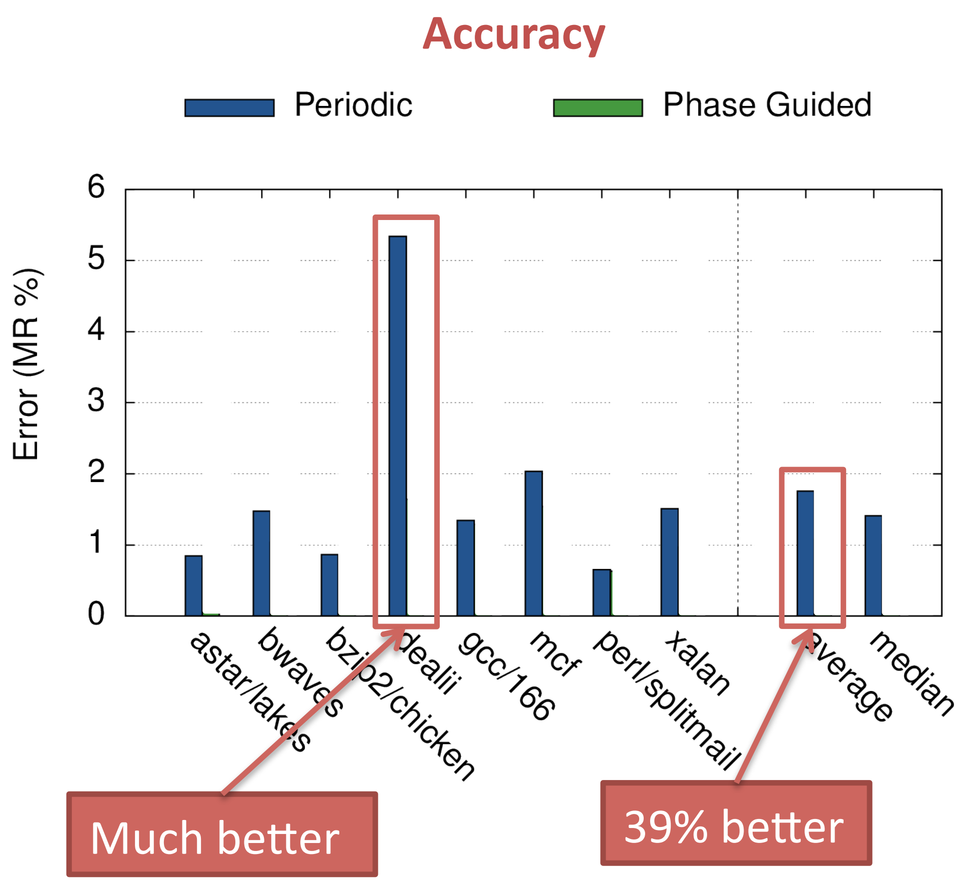


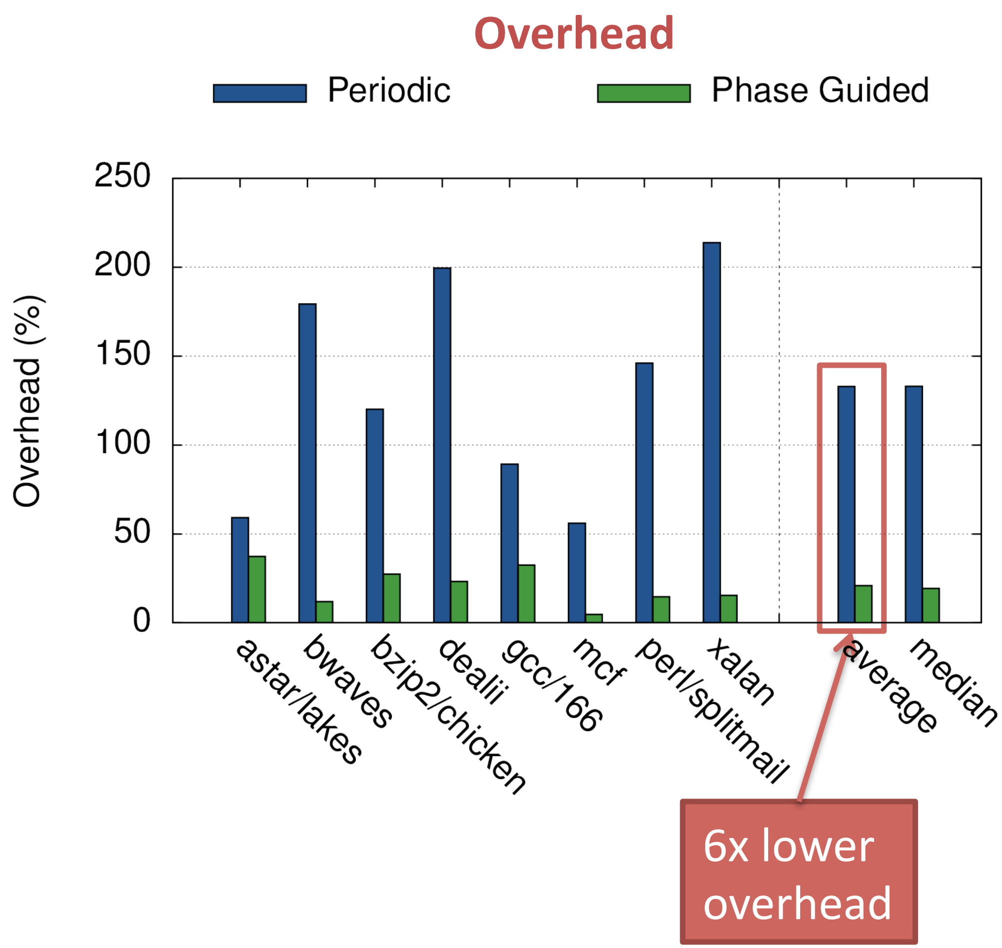
- Online (while the program is profiled)
- **2% overhead** via hardware performance counters (Intel PEBS)

Efficient Data Collection with Phases



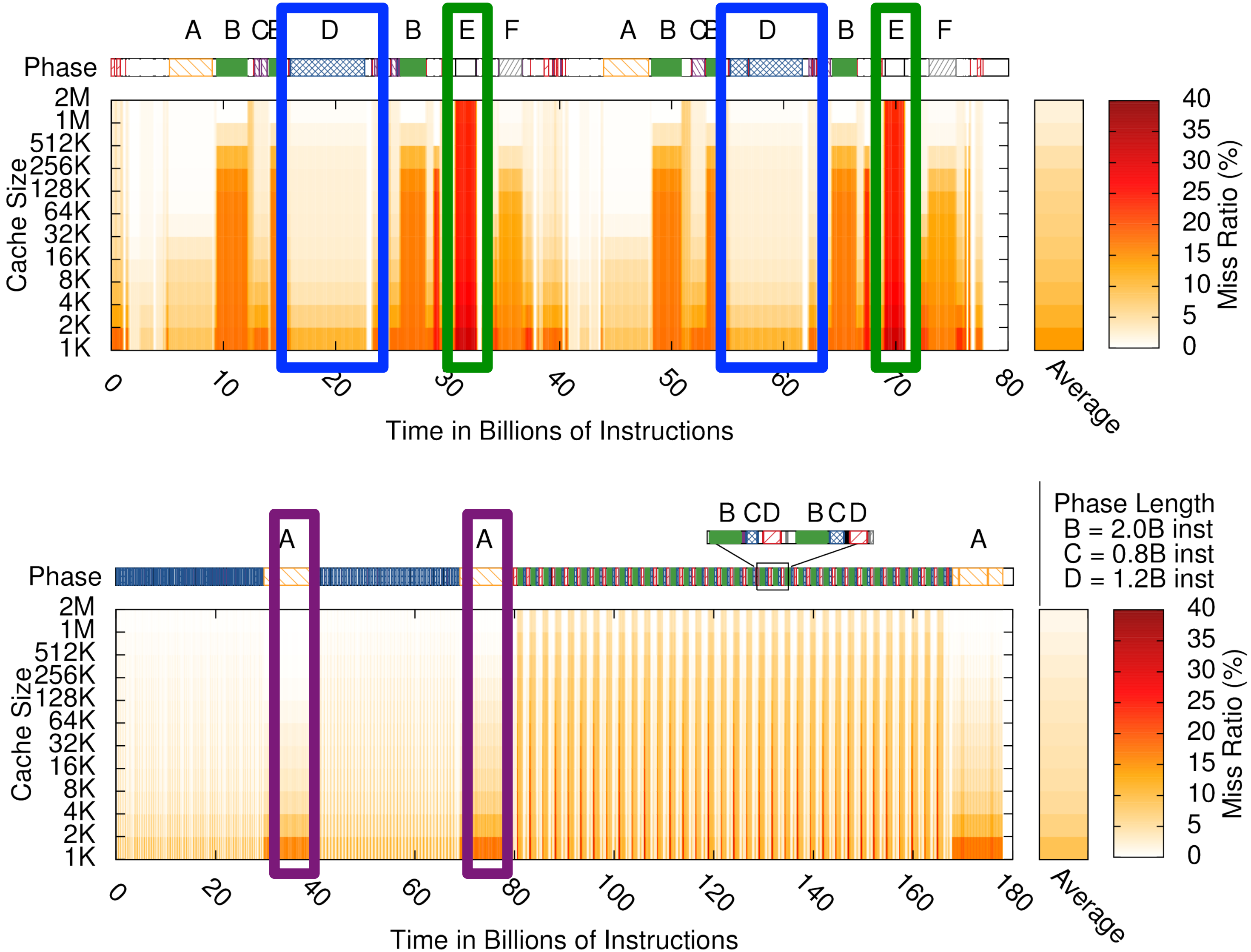
Better *and* Faster with Phases





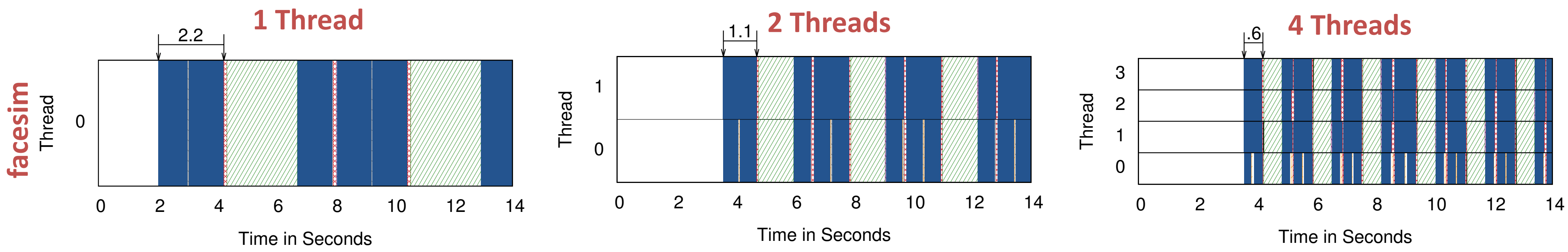
- Faster and more accurate
- Easier to use: adapts to complexity of application

Phases Key for Insight

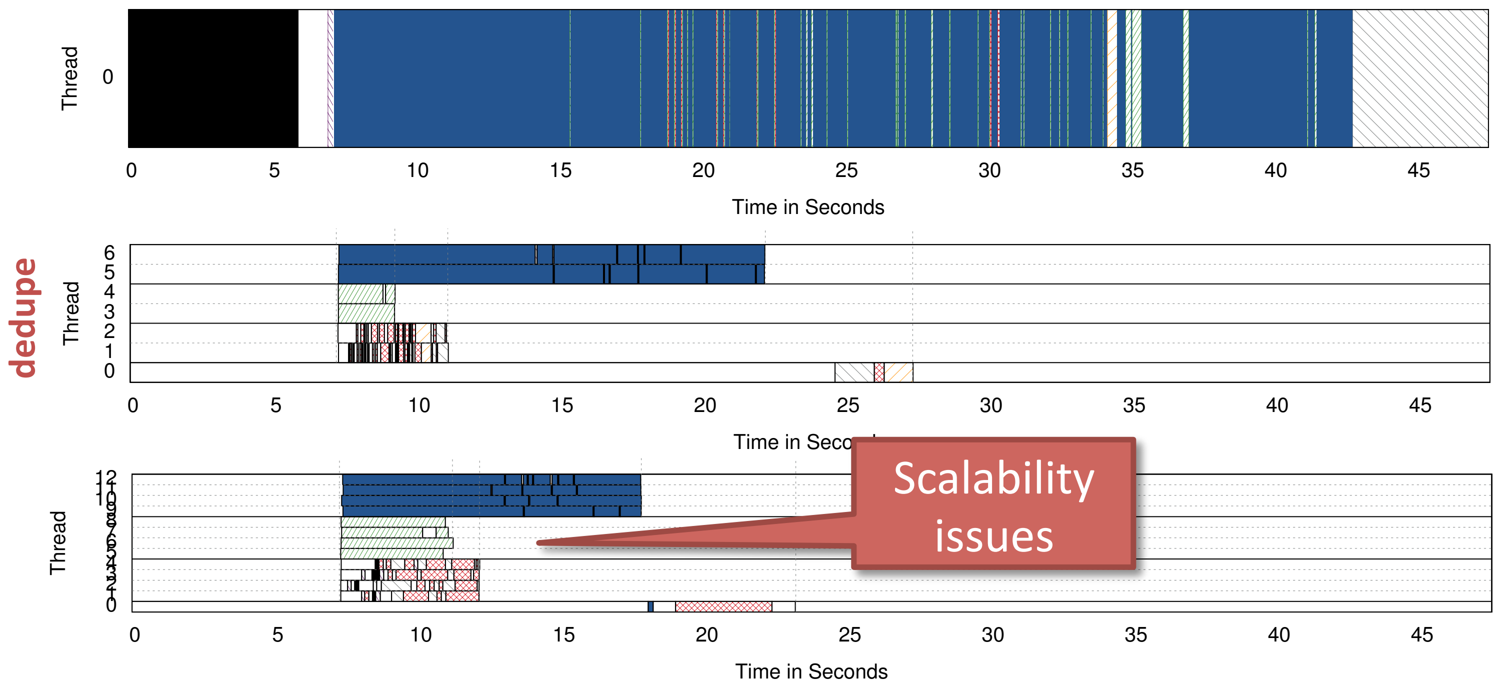


StatStack (cache modeling) with phases 20% overhead

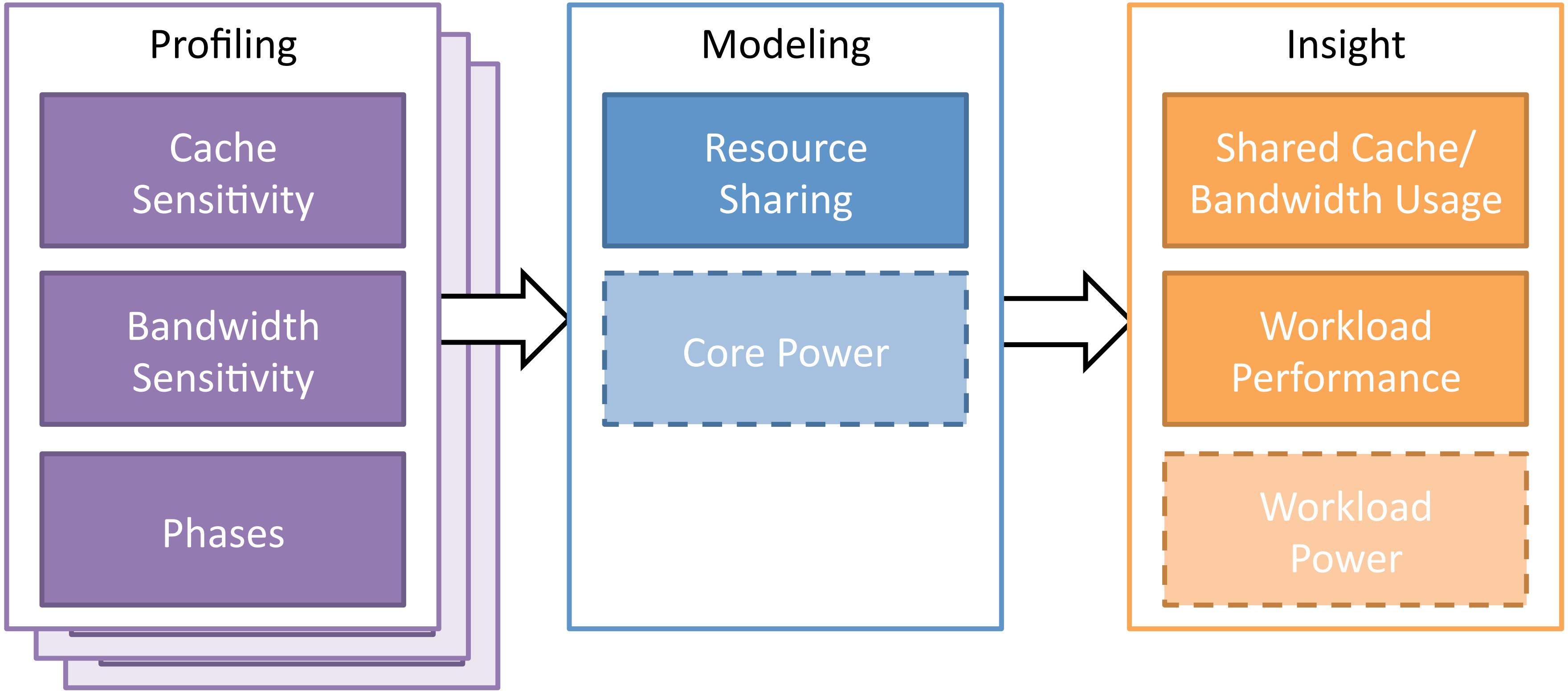
Phases in Parallel Applications



More threads → shorter phases → harder to optimize DVFS, cache size, etc.



The Big Picture



- **<u>Individual</u> task profiles enable:**
  - **Performance prediction** for co-execution (development)
  - **Efficient scheduling** for resource contention (runtime)

Future Work

- Phases + Pirate + Bandit
  - Lower-overhead profiling & more detailed information
- Sharing Model + Bandit
  - Remove the "unlimited" bandwidth assumption
- Sharing Model + Phases
  - Understand execution alignment (variability)
- Understanding the cause of slowdowns
  - Cache? Bandwidth? Synchronization?
- Plus power…

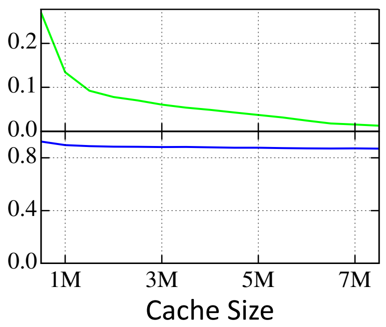
SENSITIVITY IN MORE DETAIL

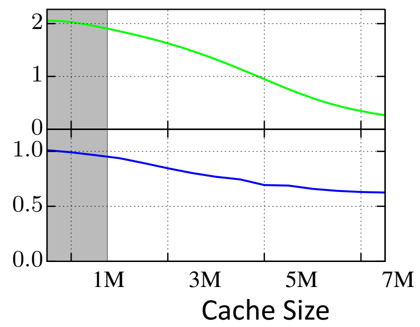
More Detail: Cache Sharing

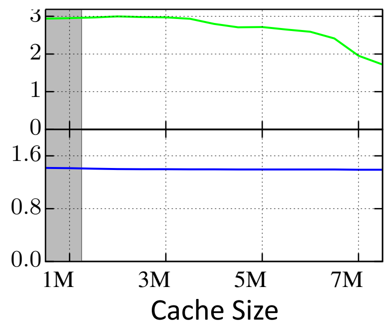
- Three SPEC benchmark applications
- Different behaviors due to different properties
- **Each will respond differently to cache sharing**

Bandwidth (GB/s)

Performance (lower better)







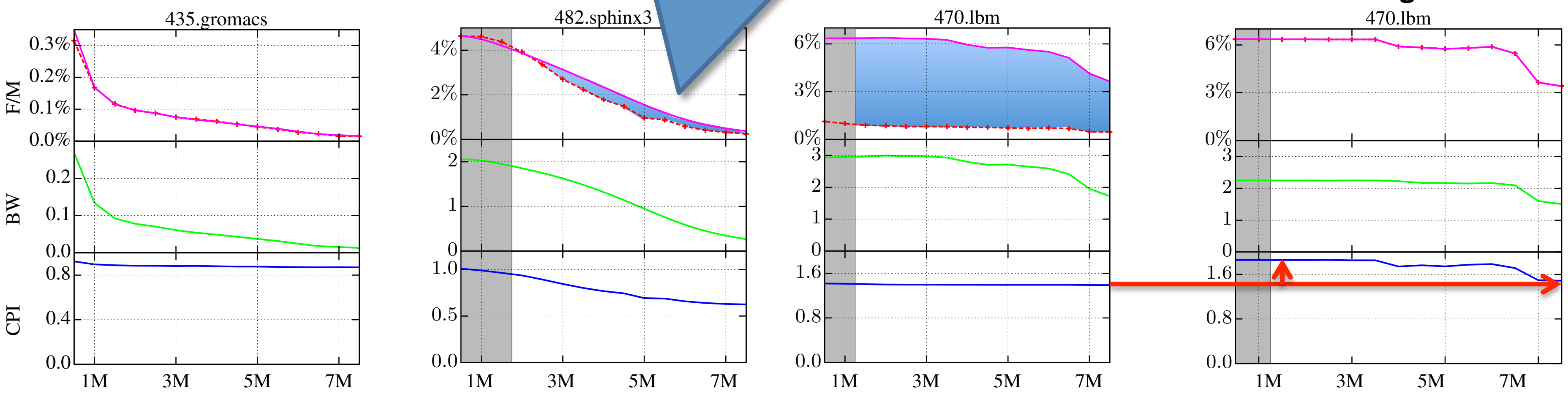
Perf:	**0% slower**	**50% slower**	**0% slower**
BW:	**50x bandwidth increase**	**10x bandwidth increase**	**2x bandwidth increase**
Size:	**Full** working set fits in the cache	**Most** of the working set fits in the cache	**Little** of the working set fits in the cache
Sensitivity:	**Insensitive** to latency	**Sensitivity** to latency	**Insensitive** to latency

More Detail: The Impact of Prefetching

- Different degrees of prefetching depending on application access pattern and hardware
- Prefetching reduces application sensitivity to latency

Difference between fetches and misses is prefetch rate.

Prefetching Disabled

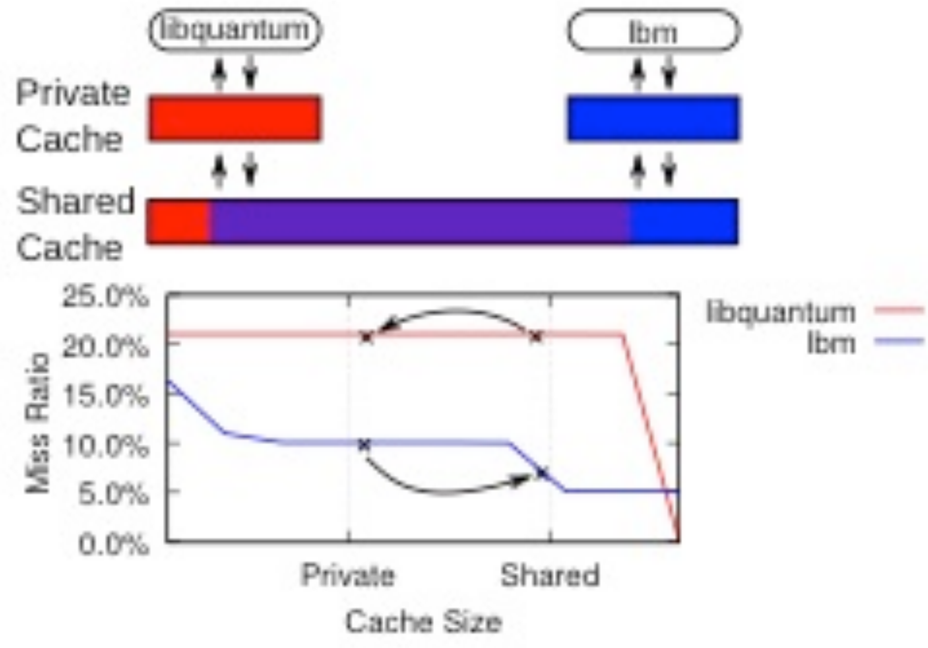


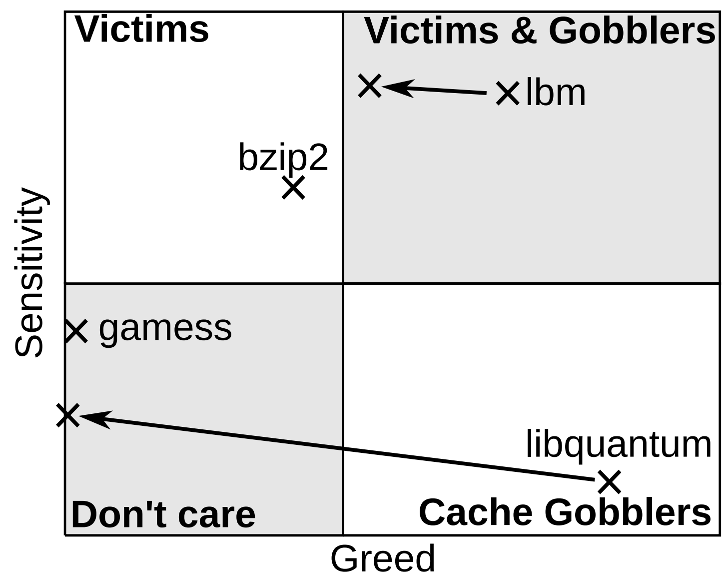
- **No Prefetching**
- **Minimal Prefetching**
- **Significant** prefetching, and it benefits from it

Miss Ratio Fetch Ratio CPI Bandwidth

More Detail: Cache Pollution

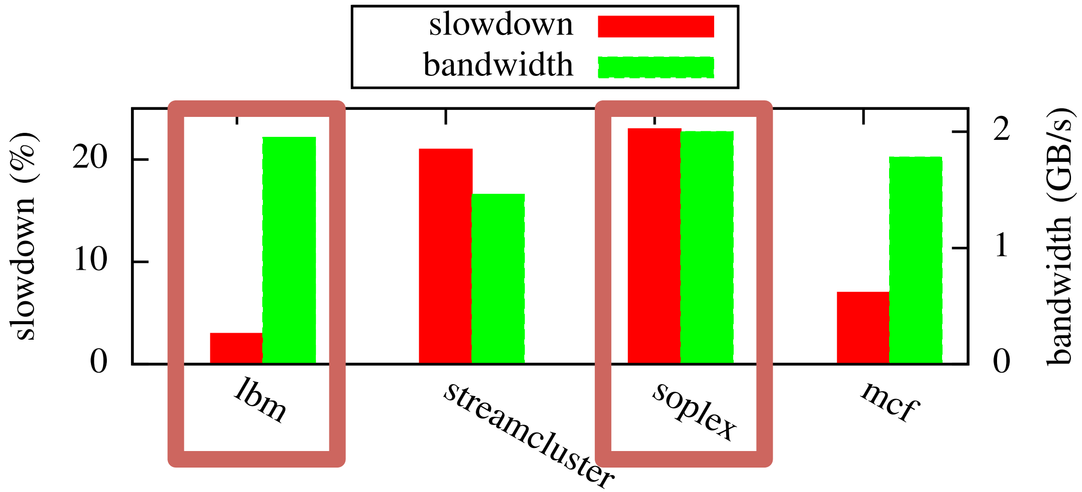
- We can measure how **greedy** an application is and how **sensitive** it is
- By changing the code to use non-caching instructions **we can make an application less greedy** without hurting performance





More Detail: Bandwidth Sharing

- Sensitivity is a function of the application
  - **Latency sensitivity** (memory level parallelism)
  - **Bandwidth requirement** (data rate)
- And the hardware
  - Ability to handle out-of-order requests (queue sizes)
  - Access pattern costs (streaming vs. random in DRAM banks)
- **BW consumption is not a good indicator of BW sensitivity**



Slowdown at 90% of saturation bandwidth