A Hardware Evaluation of Cache Partitioning to Improve Utilization and Energy-Efficiency while Preserving Responsiveness

Henry Cook, Miquel Moreto, Sarah Bird,
Kanh Dao, David Patterson, Krste Asanovic

University of California, Berkeley
Universitat Politecnica de Catalunya

Responsiveness (In The Cloud)

- Introducing server-side search result delays of **< 0.5** seconds impacts critical business metrics
  - Time to click, satisfaction, daily searches per user
- The cost of added delay increases over time and **persists** afterwards
- Results were so negative that some A/B experiments were halted **ahead of schedule**

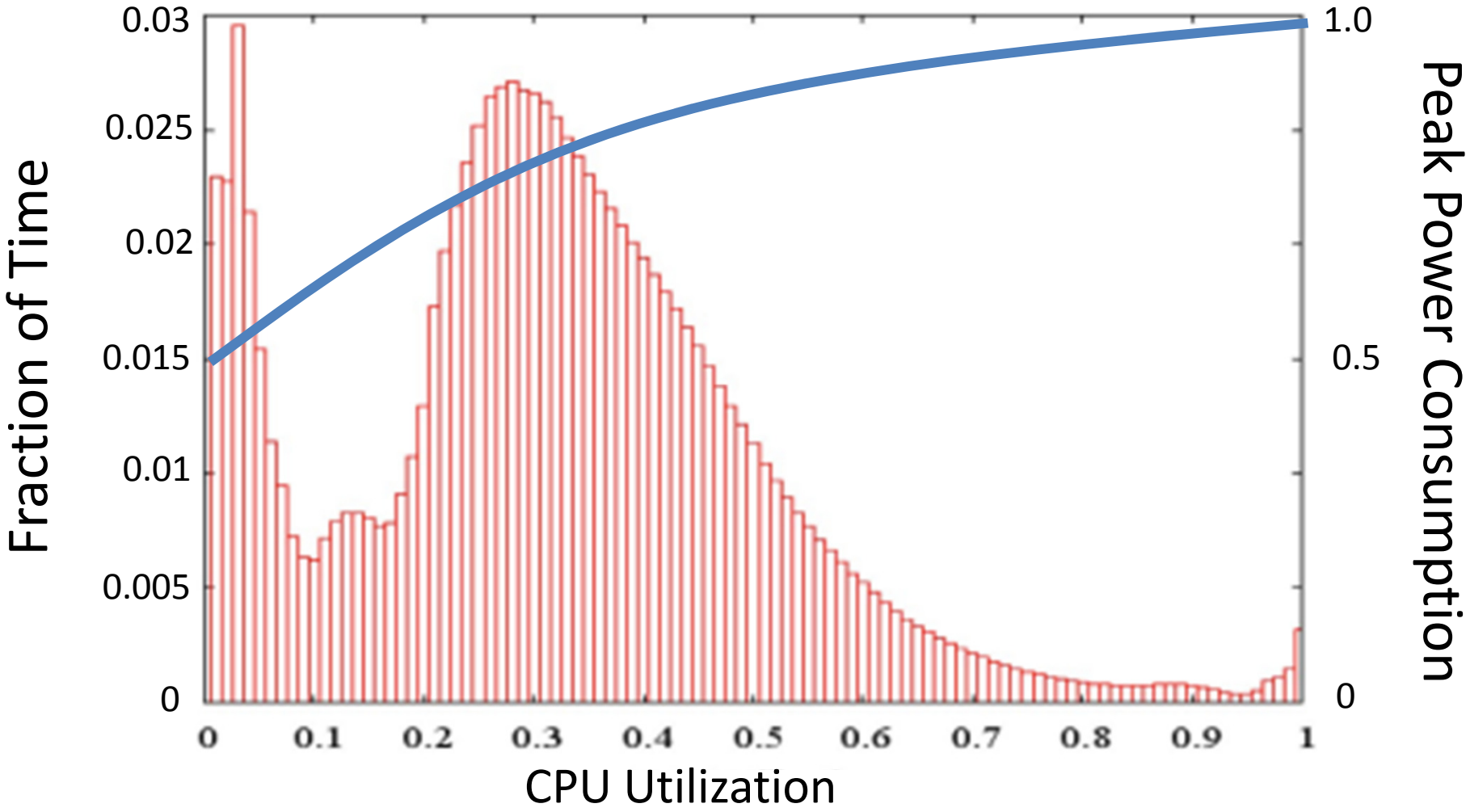
E. Schurman and J. Brutlag. The user and business impact of server delays, additional bytes, and http chunking in web search. In Velocity, 2009.

Responsiveness (On The Client)

- "Save user data and app state information. …This step is necessary because your app might be quietly killed while in the background **for any number of reasons**."
- "Using these calls causes your app to be killed immediately."
- "When your app is suspended, if it is found to be using a **shared resource**, the app is killed."

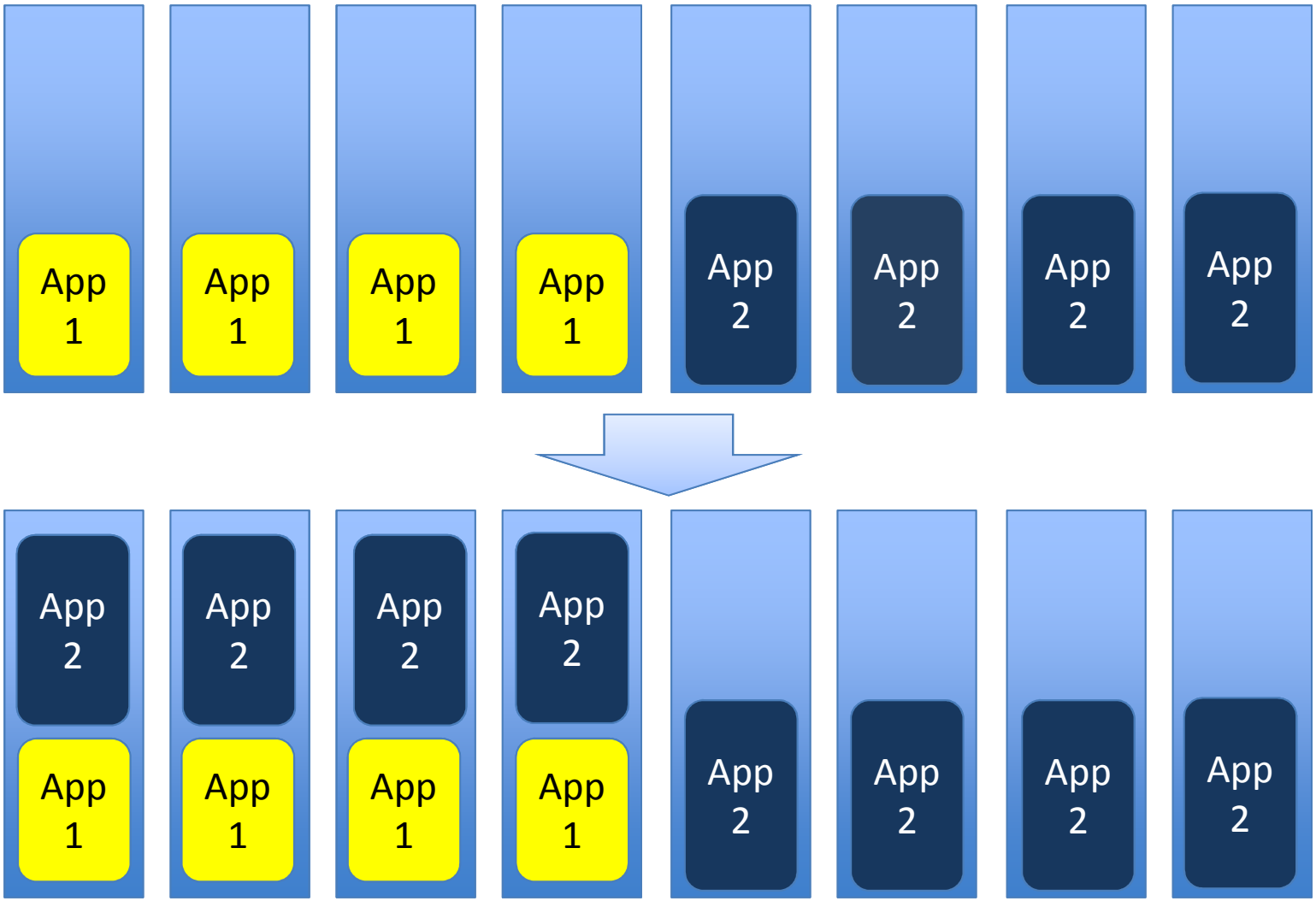
Apple iOS Developer Library, Chapter 3: App States and Multitasking, 2013

Underutilization



"The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines"
Luiz André Barroso and Urs Hölzle, 2009.

Consolidation Challenge



A Well-Studied Problem

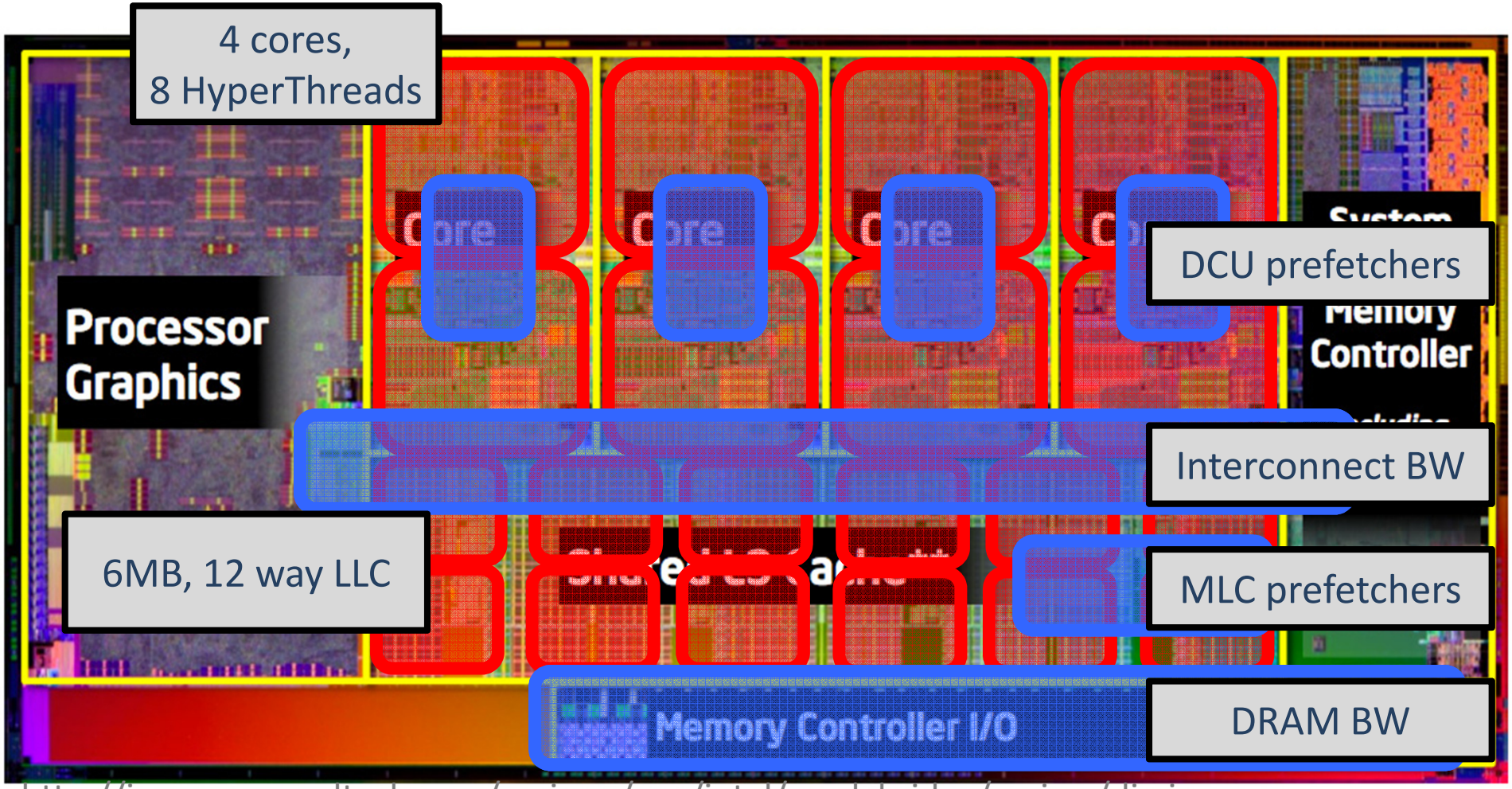
1. Ravi Iyer. 2004. CQoS: a framework for enabling QoS in shared caches of CMP platforms. In*Proceedings of the 18th annual international conference on Supercomputing* (ICS '04). ACM, New York, NY, USA, 257-266.
2. F. J. Cazorla, P. M. W. Knijnenburg, R. Sakellariou,E. Fernandez, A. Ramirez, and M. Valero. Predictable Performance in SMT Processors: Synergy between the OS and SMTs. IEEE Trans. Computers, 55(7):785–799, 2006.
3. D. Chandra, F. Guo, S. Kim, and Y. Solihin. Predicting inter-thread cache contention on a chip multi-processor architecture. In HPCA, pages 340, 2005.
4. S. Cho and L. Jin. Managing distributed, shared l2 caches through os-level page allocation. In MICRO, pages 455–468, 2006.
5. A. Fedorova, S. Blagodurov, and S. Zhuravlev. Managing contention for shared resources on multicore processors. Commun. ACM, 53(2):49–57, 2010.
6. F. Guo, Y. Solihin, L. Zhao, and R. Iyer. A framework for providing quality of service in chip multi-processors. In MICRO, 2007.
7. R. R. Iyer, L. Zhao, F. Guo, R. Illikkal, S. Makineni, D. Newell, Y. Solihin, L. R. Hsu, and S. K. Reinhardt. QoS policies and architecture for cache/memory in CMP platforms. In SIGMETRICS, pages 25–36, 2007.
8. J. W. Lee, M. C. Ng, and K. Asanovic. Globally-synchronized frames for guaranteed quality-of-service in on-chip networks. In ISCA, pages 89, 2008.
9. J. Lin, Q. Lu, X. Ding, Z. Zhang, X. Zhang, and P. Sadayappan. Gaining insights into multicore cache partitioning: Bridging the gap between simulation and real systems. In HPCA, Feb. 2008.
10. M. Moreto, F. J. Cazorla, A. Ramirez, R. Sakellariou, and M. Valero. FlexDCP: a QoS framework for CMP architectures. SIGOPS Oper. Syst. Rev., 2009.
11. M. K. Qureshi and Y. N. Patt. Utility-Based Cache Partitioning: A Low-Overhead, High-Performance, Runtime Mechanism to Partition Shared Caches. In MICRO, pages 423–432, 2006.
12. D. Sanchez and C. Kozyrakis. Vantage: Scalable and Efficient Fine-Grain Cache Partitioning. In ISCA), June 2011.
13. G. E. Suh, S. Devadas, and L. Rudolph. A New Memory Monitoring Scheme for Memory-Aware Scheduling and Partitioning. In HPCA, 2002.
14. D. Tam, R. Azimi, L. Soares, and M. Stumm. Managing shared l2 caches on multicore systems in software. In WIOSCA, 2007.
15. L. Tang, J. Mars, N. Vachharajani, R. Hundt, and M. L. Soffa. The impact of memory subsystem resource sharing on datacenter applications. In ISCA, pages 283–294, 2011.
16. Y. Xie and G. H. Loh. Scalable shared-cache management by containing thrashing workloads. In HiPEAC, pages 262–276, 2010.
17. C.-J. Wu and M. Martonosi. Characterization and dynamic mitigation of intra-application cache interference. In ISPASS, pages 2–11, 2011.
18. E. Z. Zhang, Y. Jiang, and X. Shen. Does cache sharing on modern CMP matter to the performance of contemporary multithreaded programs? In PPoPP, pages 203–212, 2010.
19. Fei Guo, Hari Kannan, Li Zhao, Ramesh Illikkal, Ravi Iyer, Don Newell, Yan Solihin, and Christos Kozyrakis. 2007. From chaos to QoS: case studies in CMP resource management. *SIGARCH Comput. Archit. News* 35, 1 (March 2007), 21-30.
20. Moinuddin K. Qureshi, Daniel N. Lynch, Onur Mutlu, and Yale N. Patt. 2006. A Case for MLP-Aware Cache Replacement. *SIGARCH Comput. Archit. News* 34, 2 (May 2006), 167-178.
21. Andrew Herdrich, Ramesh Illikkal, Ravi Iyer, Don Newell, Vineet Chadha, and Jaideep Moses. 2009. Rate-based QoS techniques for cache/memory in CMP platforms. In *Proceedings of the 23rd international conference on Supercomputing* (ICS '09). ACM, New York, NY, USA, 479-488.
22. Jichuan Chang and Gurindar S. Sohi. 2007. Cooperative cache partitioning for chip multiprocessors. In *Proceedings of the 21st annual international conference on Supercomputing*(ICS '07). ACM, New York, NY, USA, 242-252.
23. Jaehyuk Huh, Changkyu Kim, Hazim Shafi, Lixin Zhang, Doug Burger, and Stephen W. Keckler. 2005. A NUCA substrate for flexible CMP cache sharing. In *Proceedings of the 19th annual international conference on Supercomputing* (ICS '05). ACM, New York, NY, USA, 31-40.
24. Michael R. Marty and Mark D. Hill. 2007. Virtual hierarchies to support server consolidation. *SIGARCH Comput. Archit. News* 35, 2 (June 2007), 46-56.
25. Yi Guo, Jisheng Zhao, Vincent Cave, and Vivek Sarkar. 2010. SLAW: a scalable locality-aware adaptive work-stealing scheduler for multi-core systems. In *Proceedings of the 15th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming* (PPoPP '10)

A New Opportunity

- Sandy Bridge client device prototype HW
  - Way-based LLC partitioning
  - Energy counters
- Full size parallel benchmarks, full system stack

- Goal: Evaluate the energy-saving potential of consolidation with HW for cache partitioning

Machine Resources

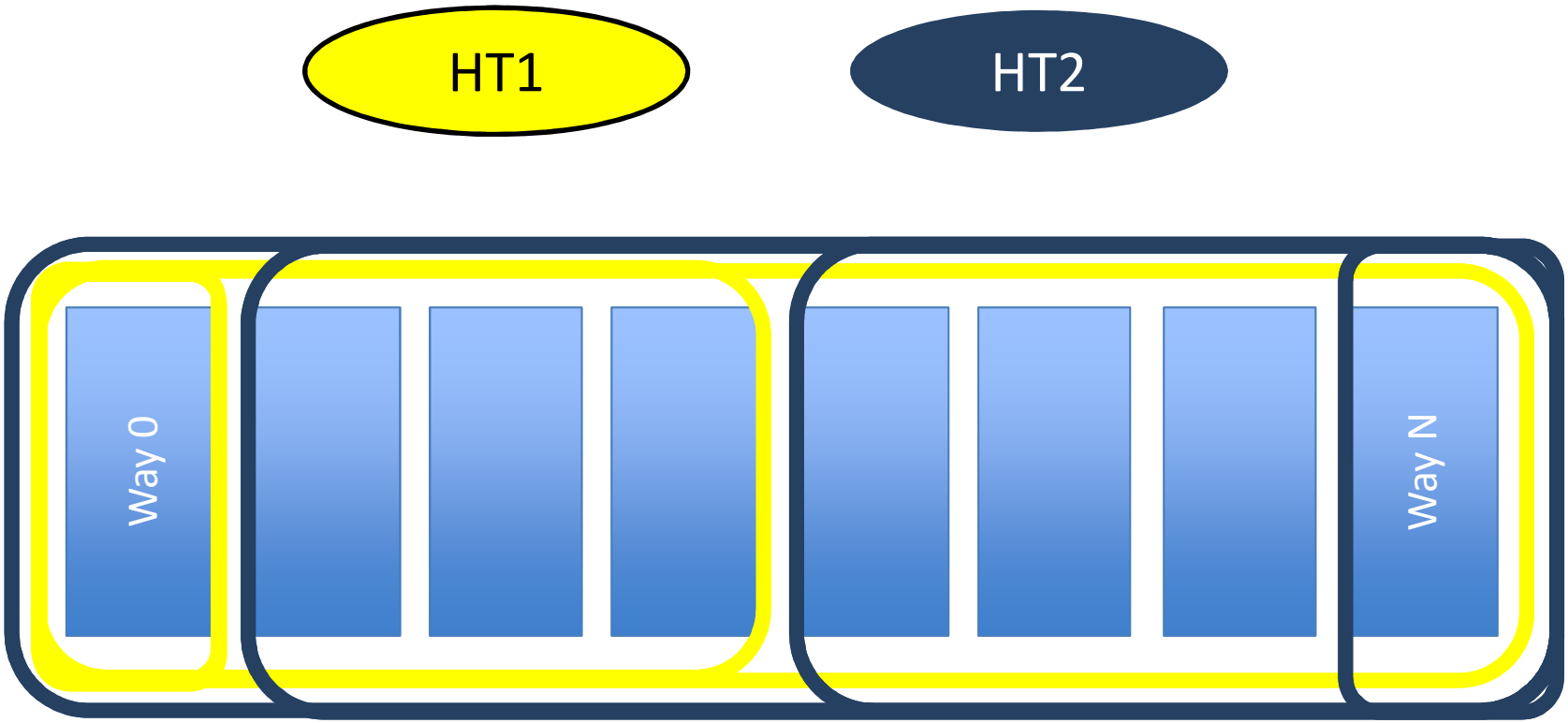


http://images.anandtech.com/reviews/cpu/intel/sandybridge/review/die.jpg

Partitionable

Unpartitionable

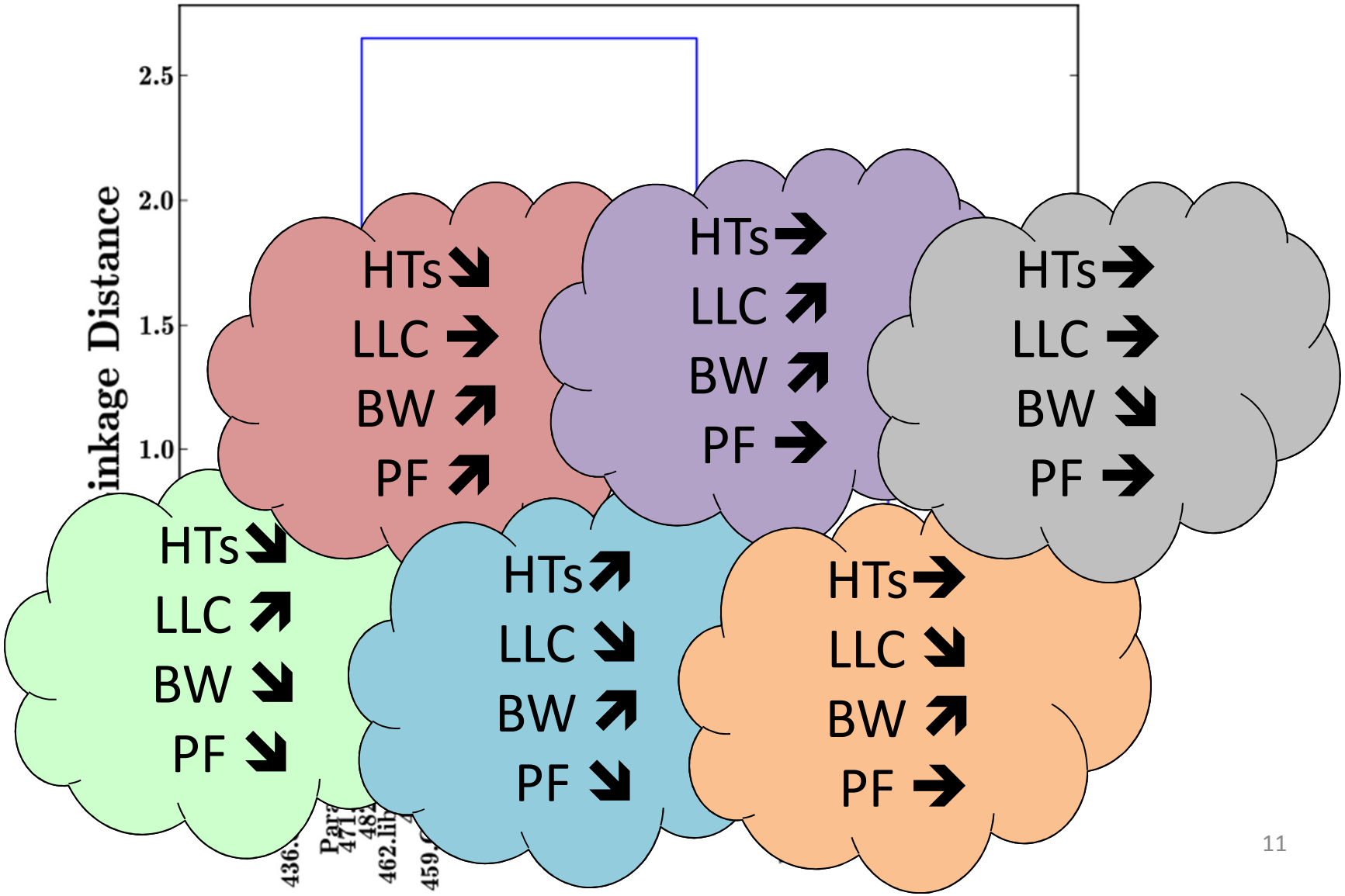
Way-Based Partitioning



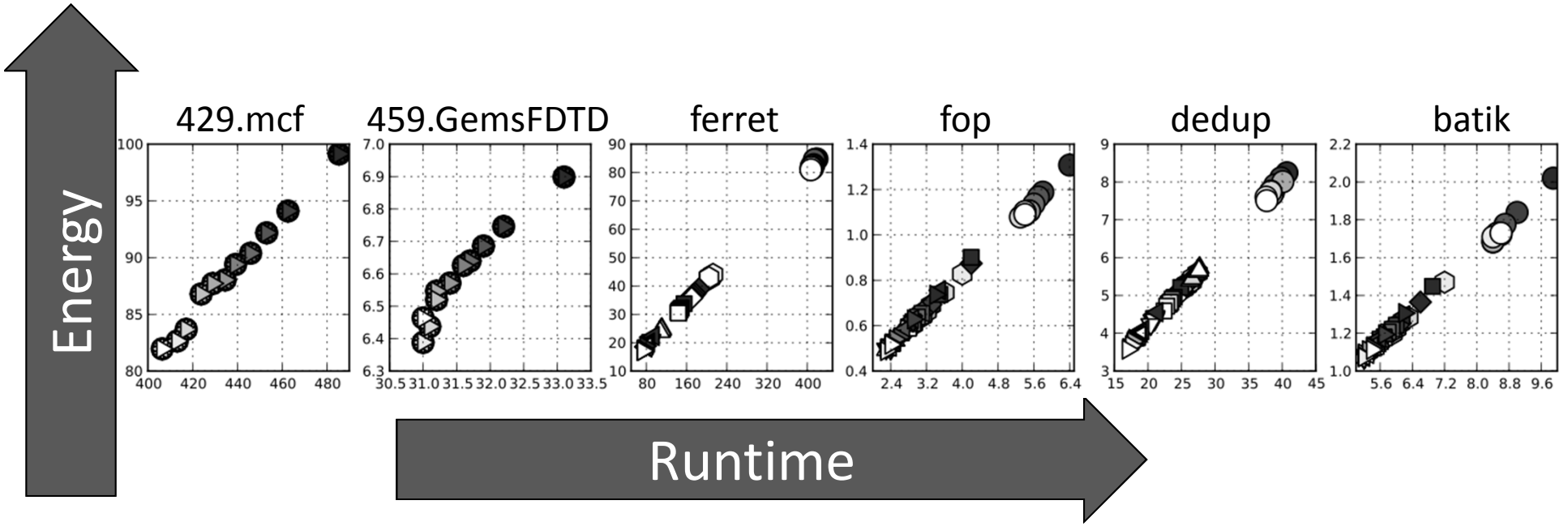
Methodology

- Multiple benchmark suites
  - Spec2006, PARSEC, DaCapo, other parallel kernels
  - **Full/large/native** input sets
- Unmodified Linux 2.6.36
- `Libpfm` library built on `perf_events`
- Running Average Power Limit (**RAPL**) interfaces
  - 16us granularity
- ACme external power meter
  - 1 sec granularity
  - http://acme.cs.berkeley.edu

Hierarchical K-means Clustering

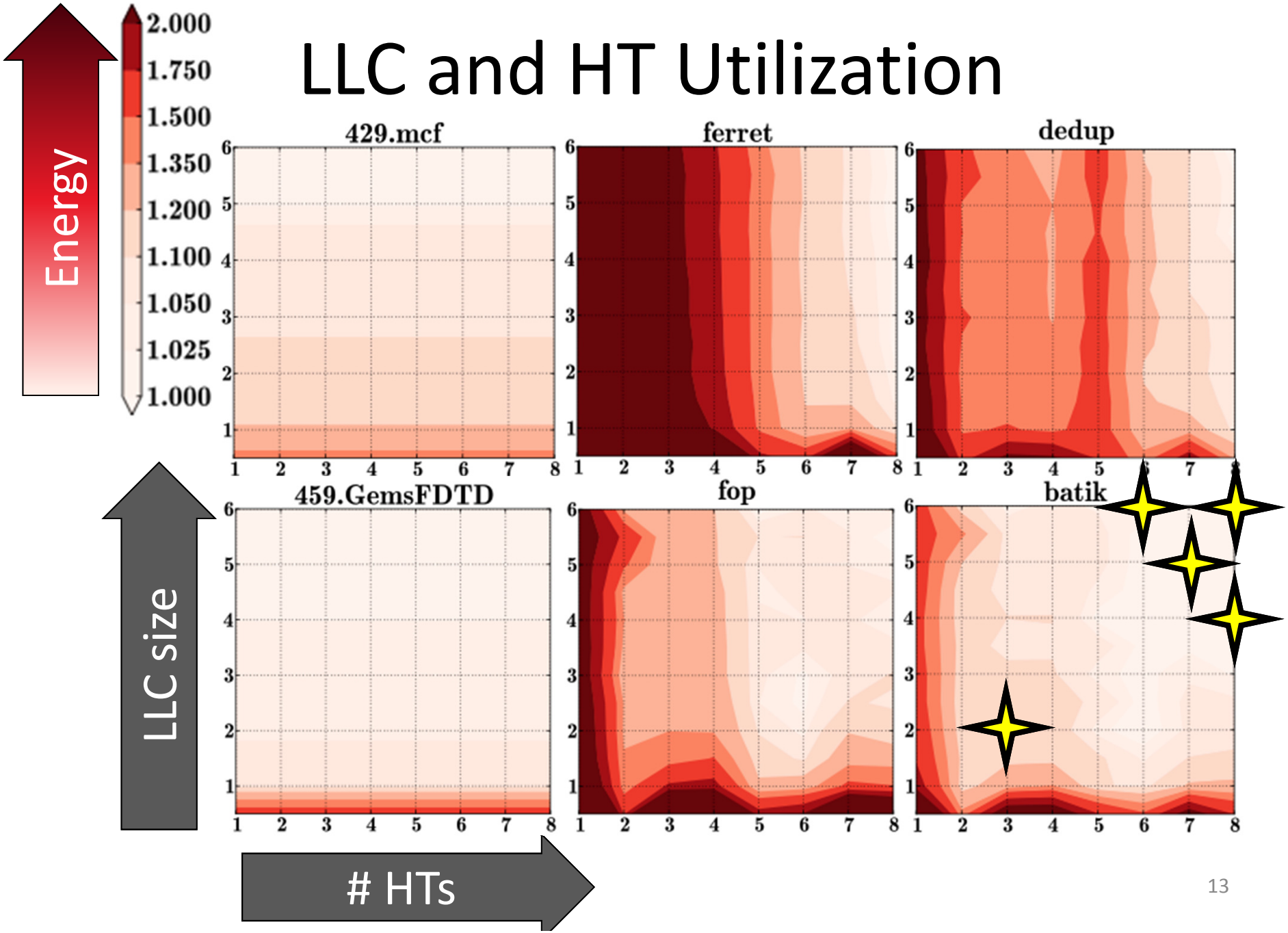


Race to Halt

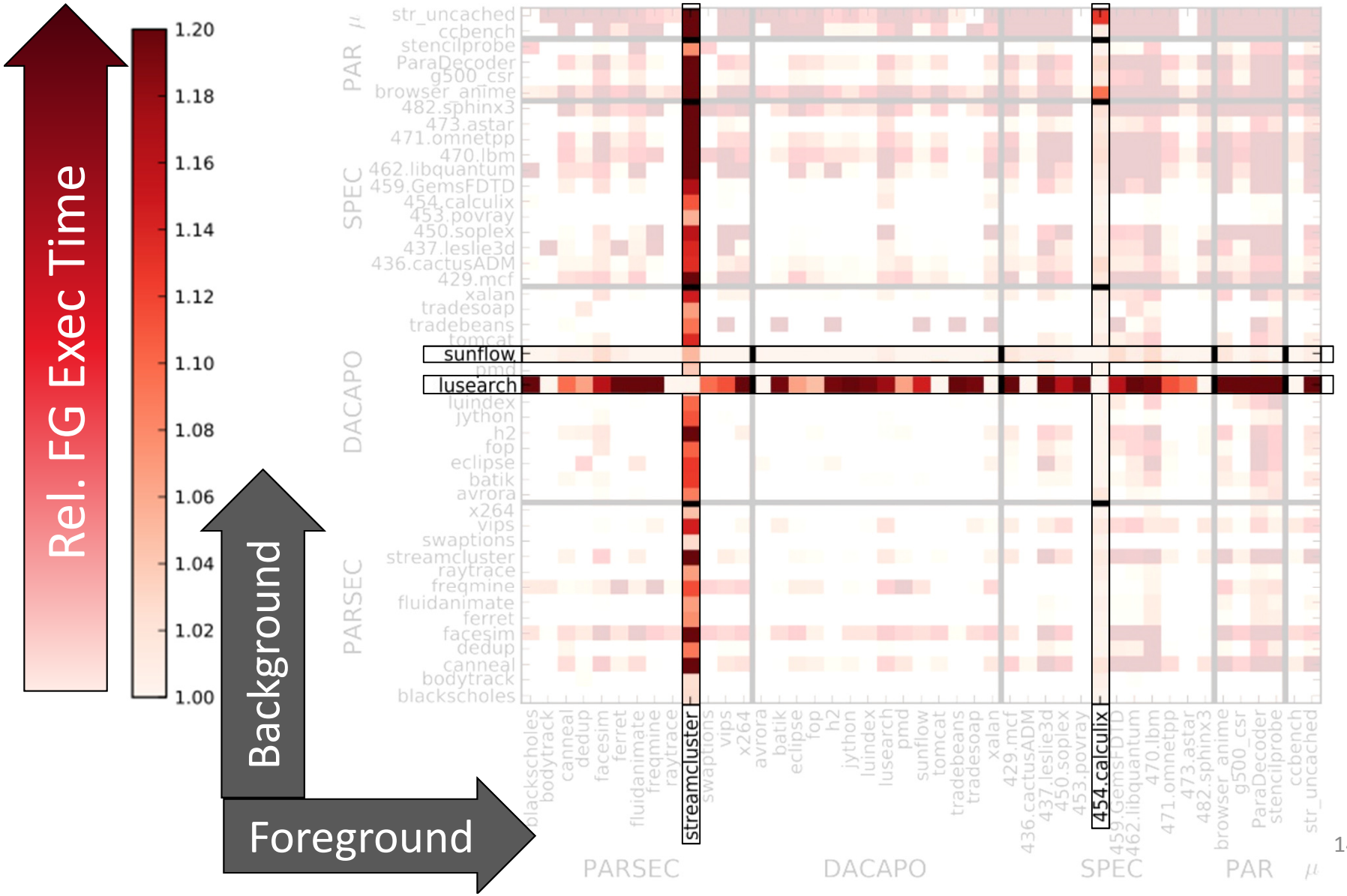


- Scattered points are the 8x12 possible allocations
- Energy α performance
- Applies across all benchmarks and allocations

LLC and HT Utilization



Multiprogram Contention



Static Partitioning: Unpartitioned

- Baseline for measuring foreground app degradation is to just let apps share each way of the LLC
- Replacement policy evicts based on usage patterns

Static partitioning	Average slowdown	Worst-case slowdown
Unpartitioned	5.9%	34.0%

Static Partitioning: Fair Partitioning

- Fair partitioning gives each app half of the cache, regardless of need
- Most naïve use of partitioning

Static partitioning	Average slowdown	Worst-case slowdown
Unpartitioned	5.9%	34.0%
Fair	6.0%	16.3%

Static Partitioning: Ideal Partitioning

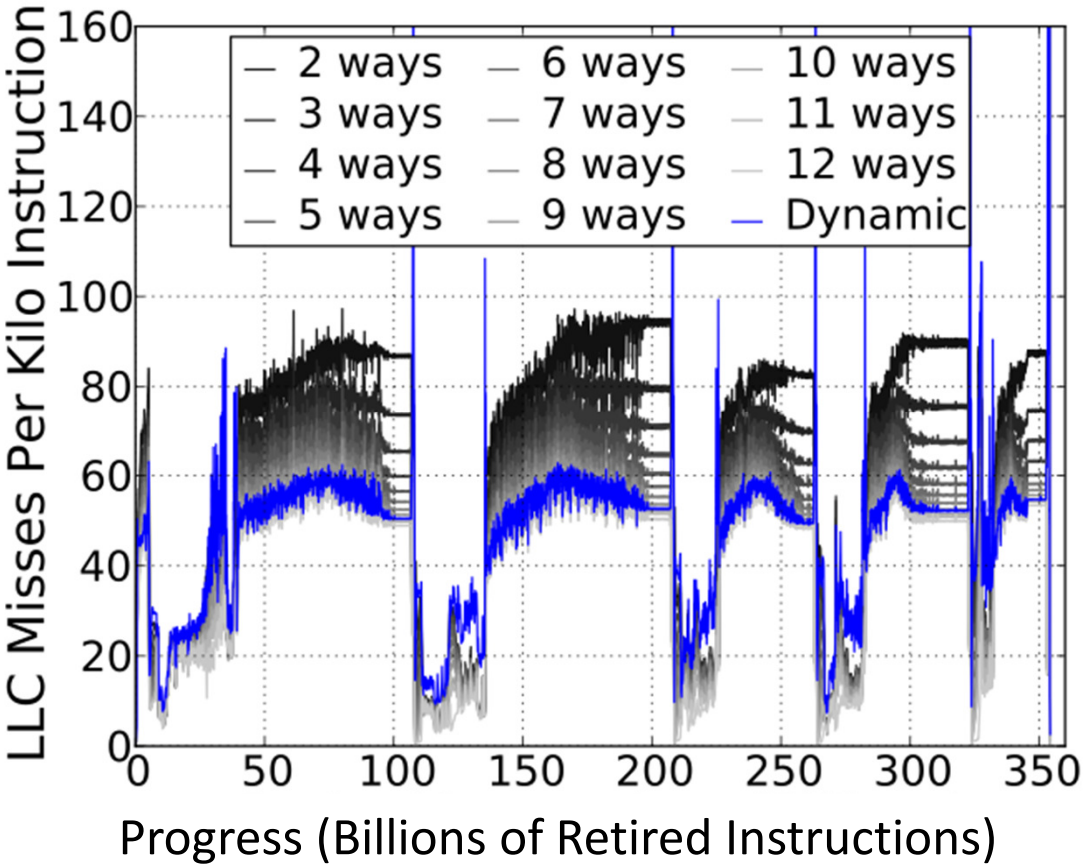
- Ideal partitioning uses the "best" allocation
  - Heuristic is: smallest FG alloc whose perf was within 1% of giving FG the whole machine, yet allows BG to run in remainder
- Oracular static partitioning

Static partitioning	Average slowdown	Worst-case slowdown
Unpartitioned	5.9%	34.0%
Fair	6.0%	16.3%
Ideal	2.3%	7.4%

Static Partitioning: Takeaways

- Partitioning mitigates worst-case degradation
- For metrics like energy or weighted speedup, consolidation is effective but differences between sharing strategies are small on average
- High variance across application pairs
- Pairing strategy >> sharing strategy

Applications Have Phases

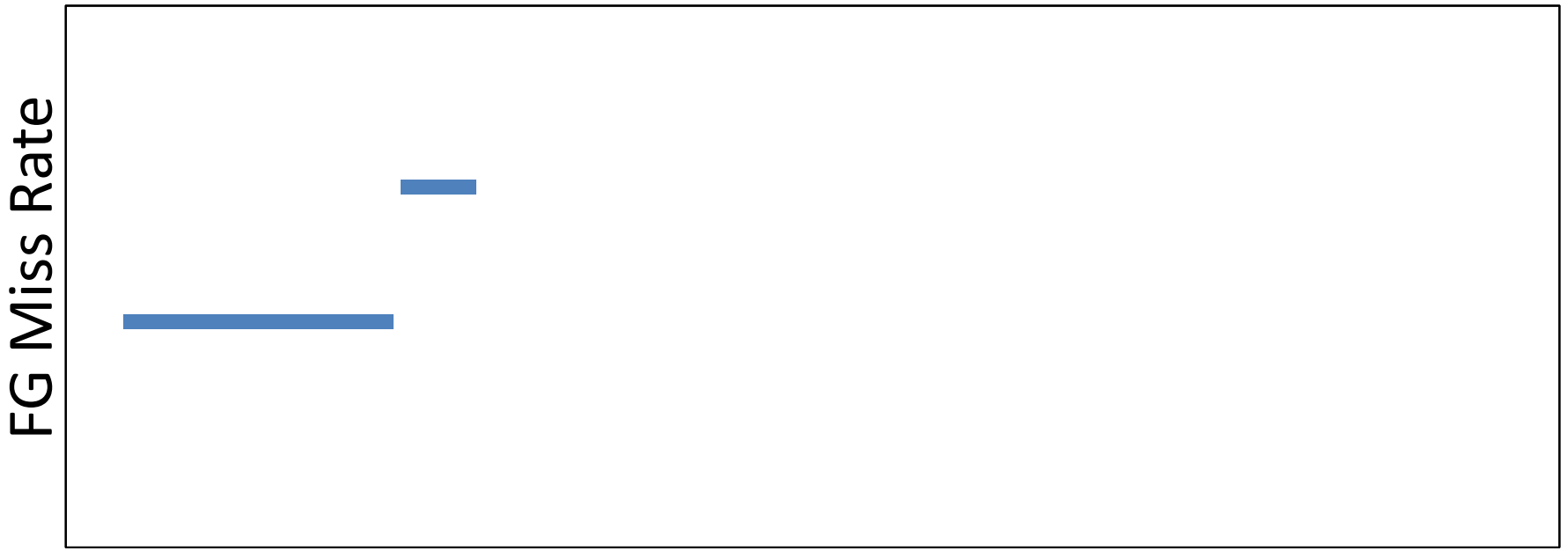


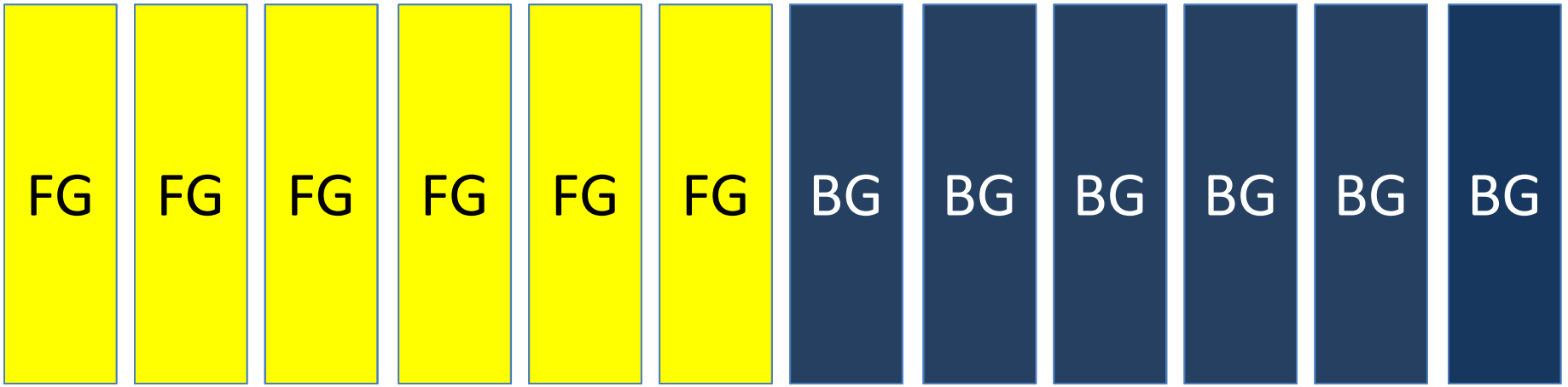
- Can we dynamically determine the LLC requirements and further consolidate?

Dynamic Algorithm

- Use performance counters to detect changes in required LLC alloc, via miss rate
- When a phase change is detected, explore allocations to determine new required size
- Give FG maximum alloc, then shrink alloc until miss rate is negatively impacted
- Hold allocation fixed until another change in miss rate is detected

Dynamic Algorithm

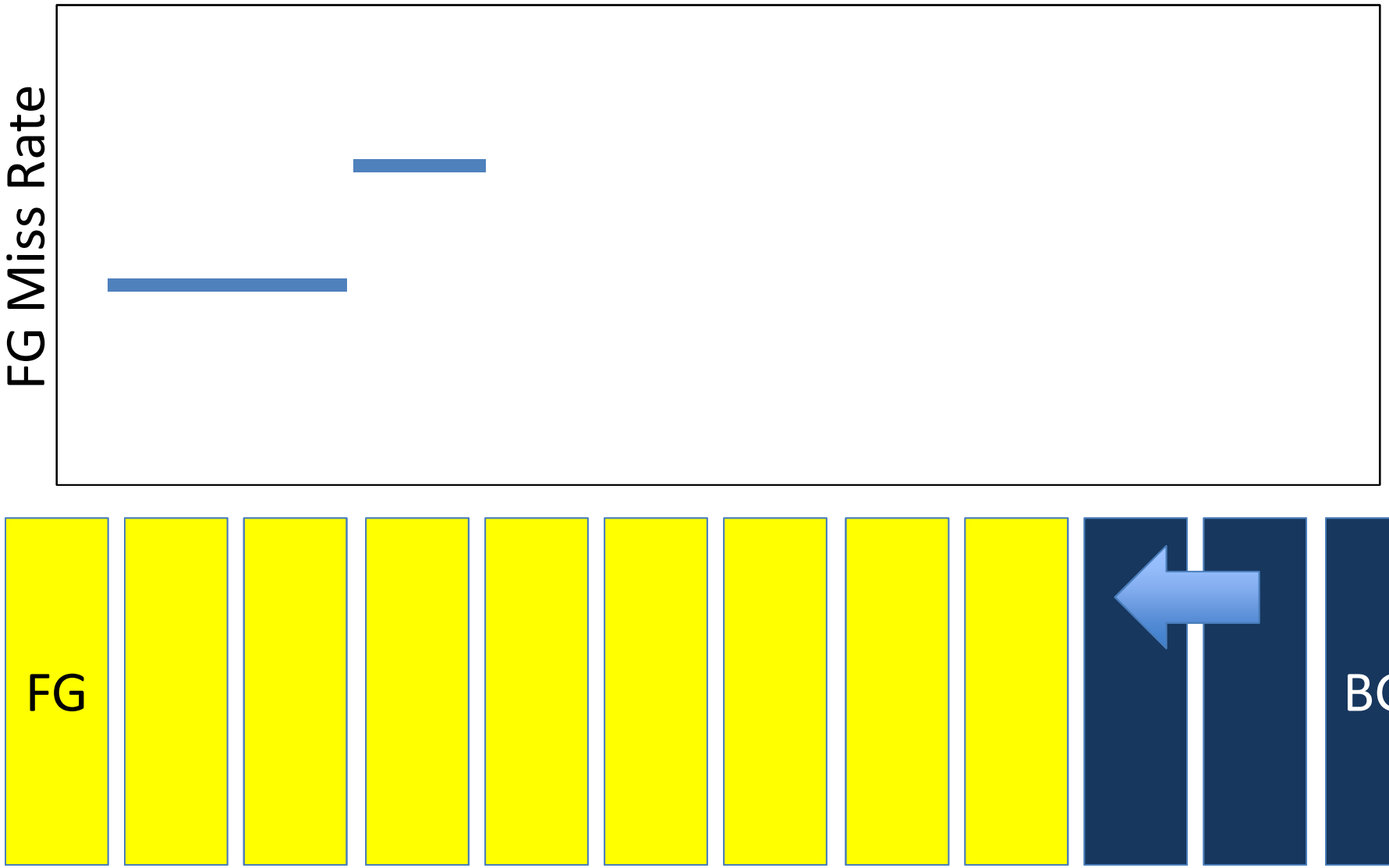




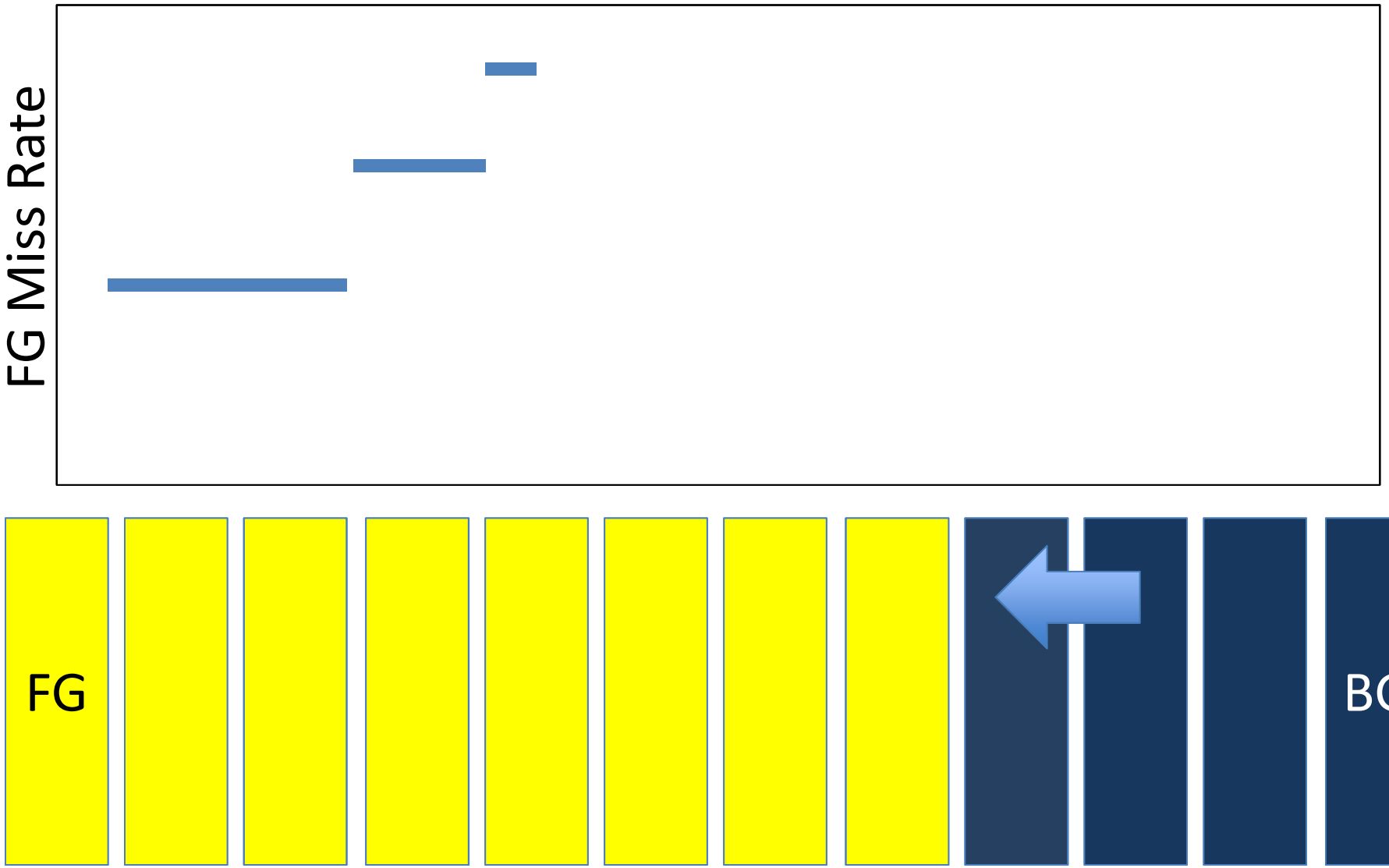
Dynamic Algorithm

Dynamic Algorithm

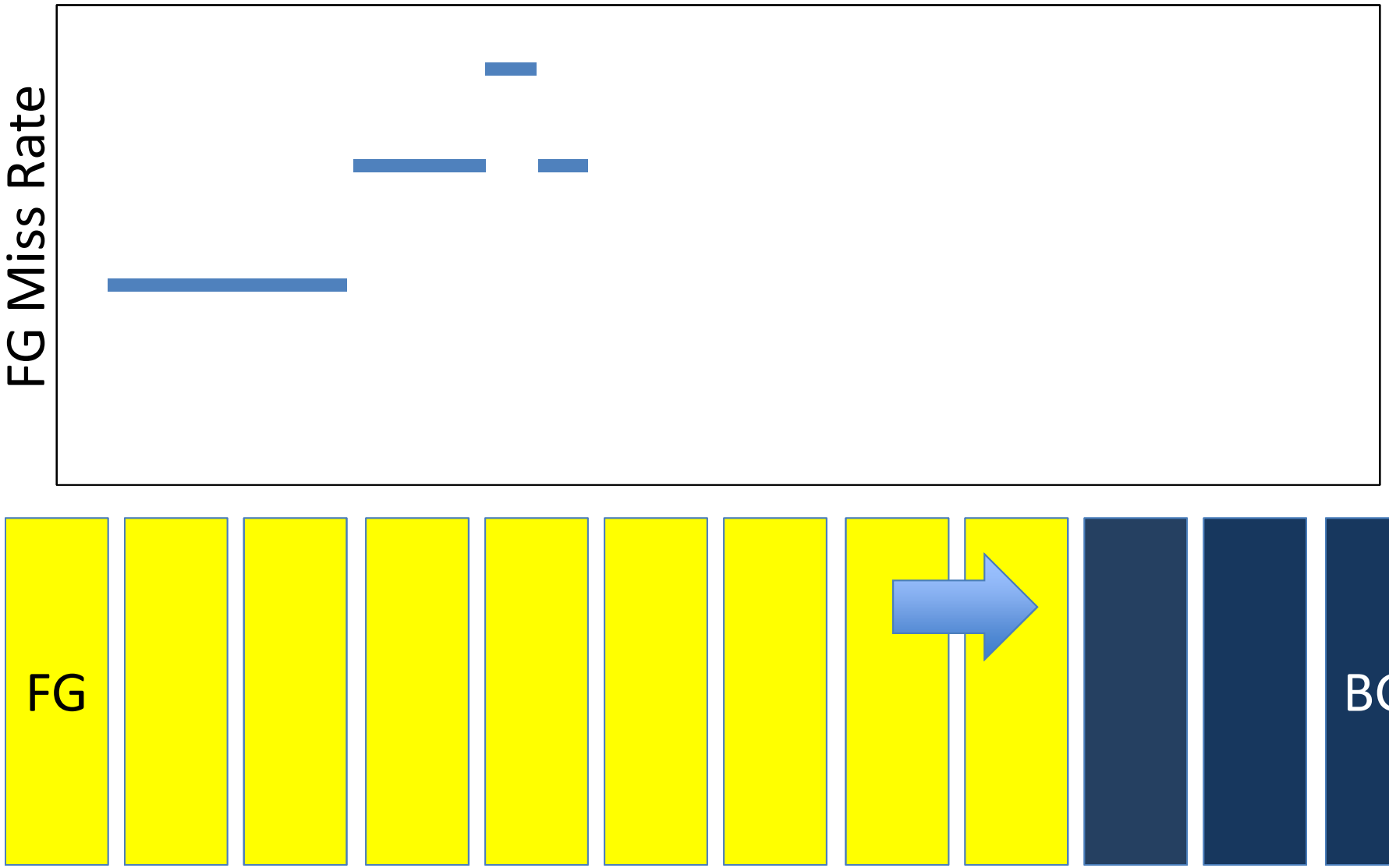
Dynamic Algorithm



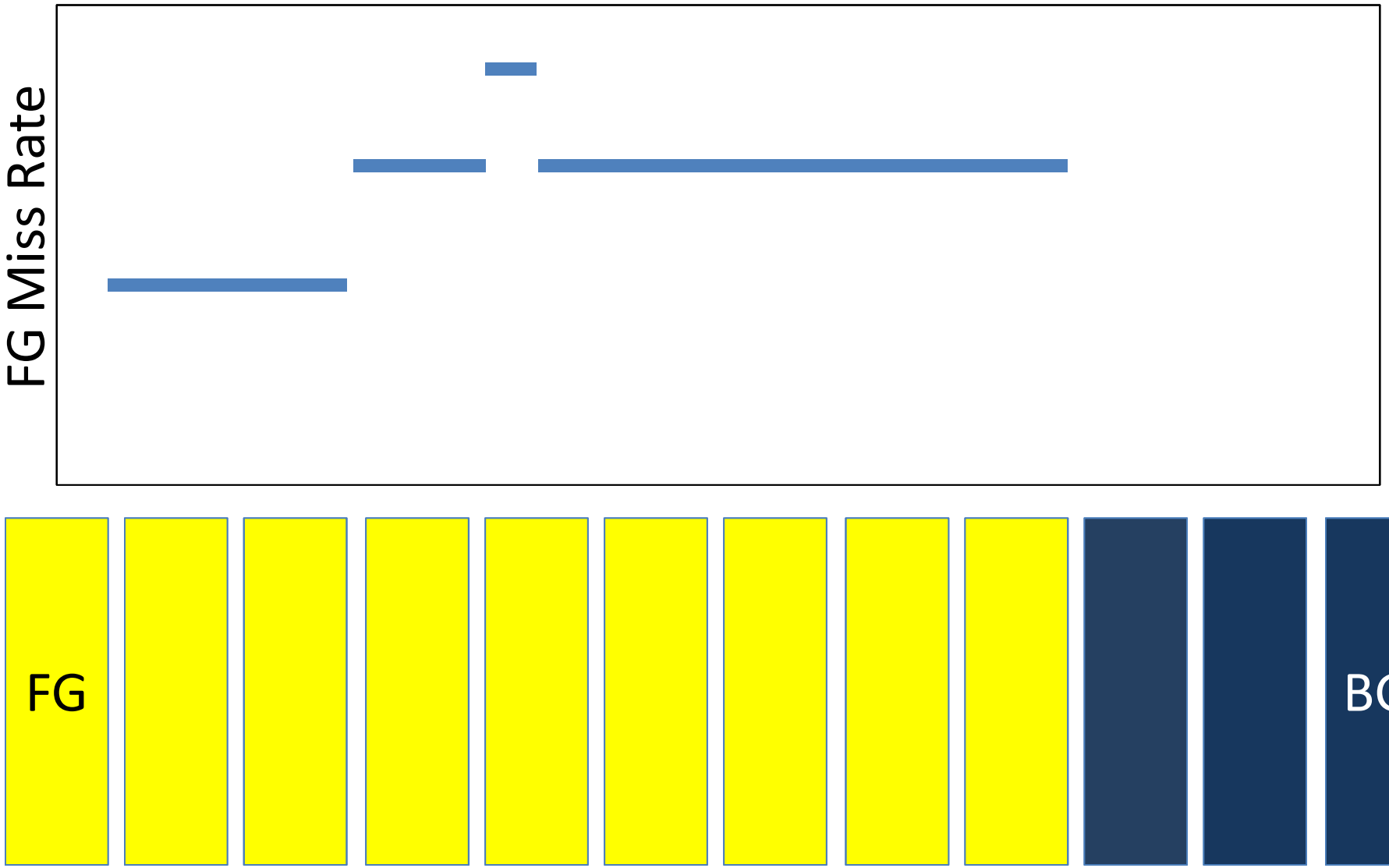
Dynamic Algorithm



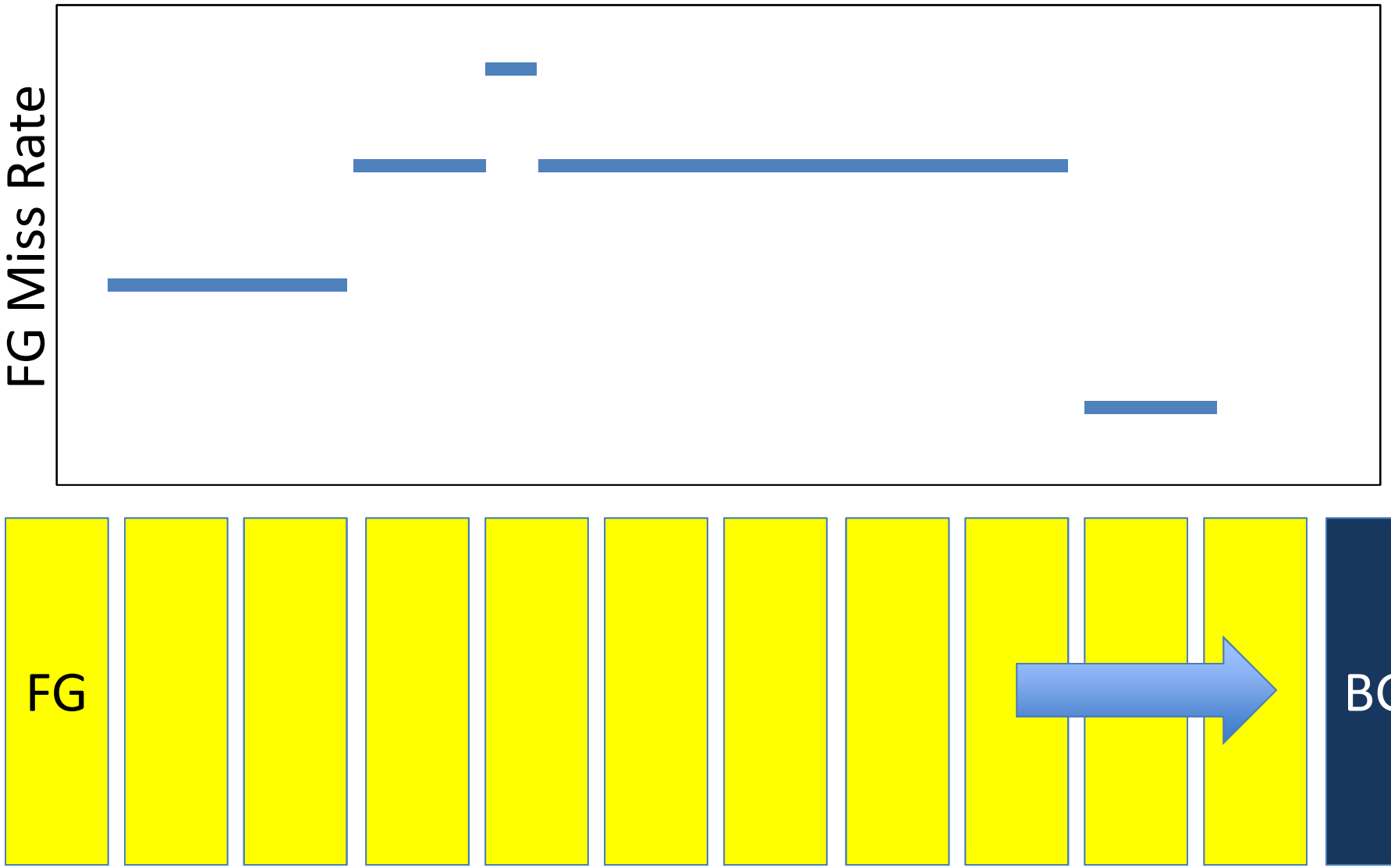
Dynamic Algorithm



Dynamic Algorithm



Dynamic algorithm



Dynamic Algorithm Results

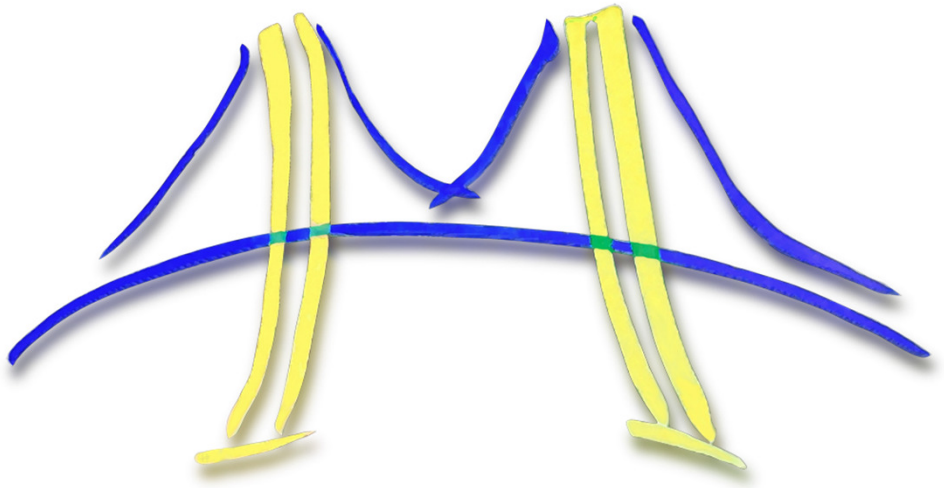
- In some cases we see significant throughput increases (up to 2.5x), resulting in a 19% throughput improvement on average
  - FG performance never worsens more than 2%
- Using a shared LLC results in a 53% throughput improvement on average
  - However, this scenario can often result in significant perf loss (up to 35%) for FG app
- Throughput correlated with energy/task

Future Work

- Explain discrepancies between real machine utilities and others' simulated results
- More big data workloads
- App-pair-specific dynamic mechanism tuning
- Mechanisms for BW partitioning
- Mechanisms to preserve prefetcher efficacy

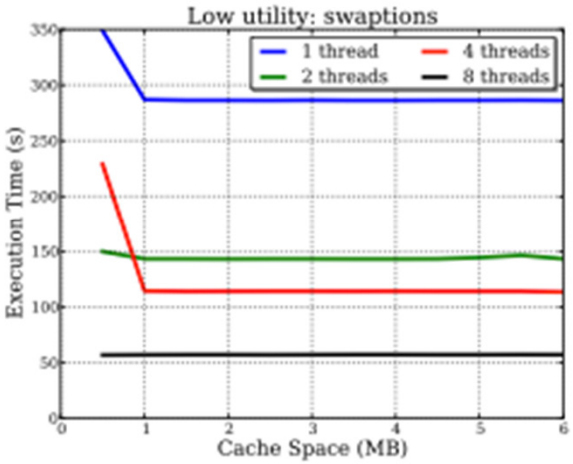
Conclusions

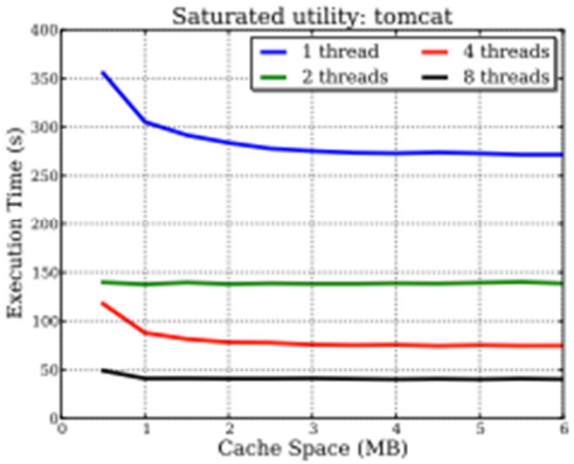
- The race-to-halt paradigm still allows for consolidation opportunities
- LLC partitioning alone is not enough to prevent degradation, but mitigates worst case
- Consolidation is very effective for saving energy, but pairing strategy >> static sharing strategy
- Dynamic LLC partitioning can be effective at reducing energy per background task while preserving FG performance

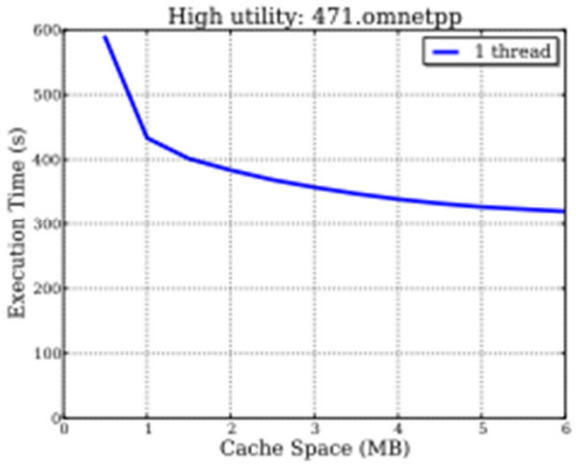


www.eecs.berkeley.edu/~hcook

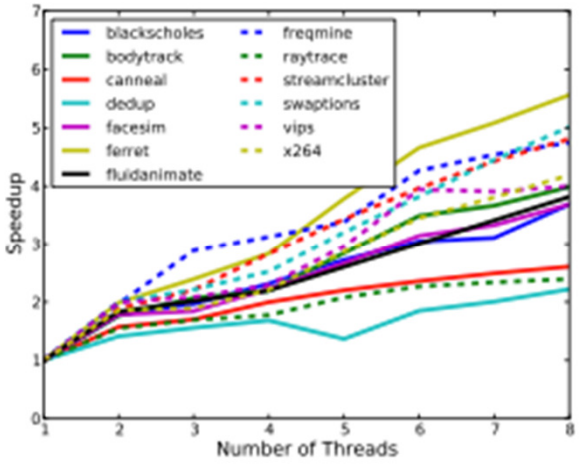
LLC sensitivity



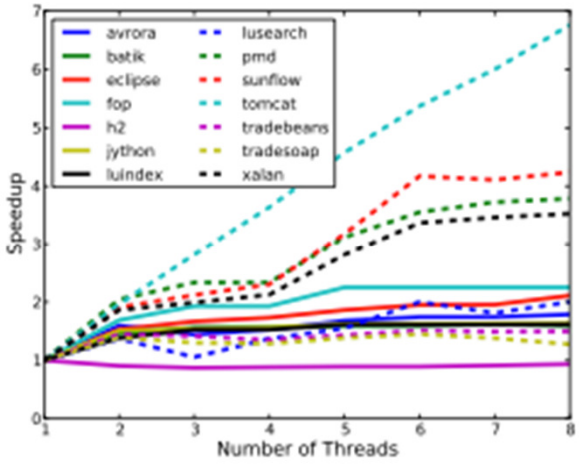




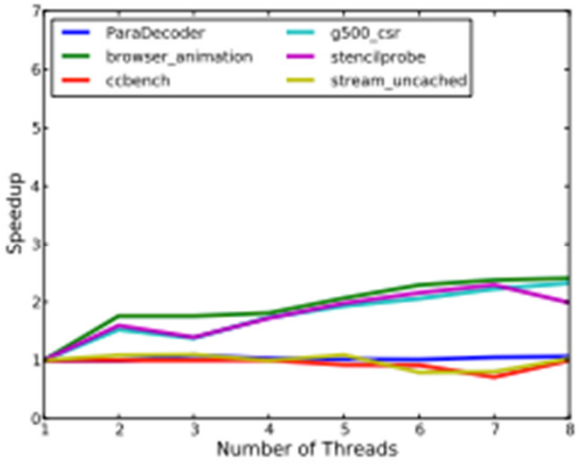
Thread scalability



(a) PARSEC applications

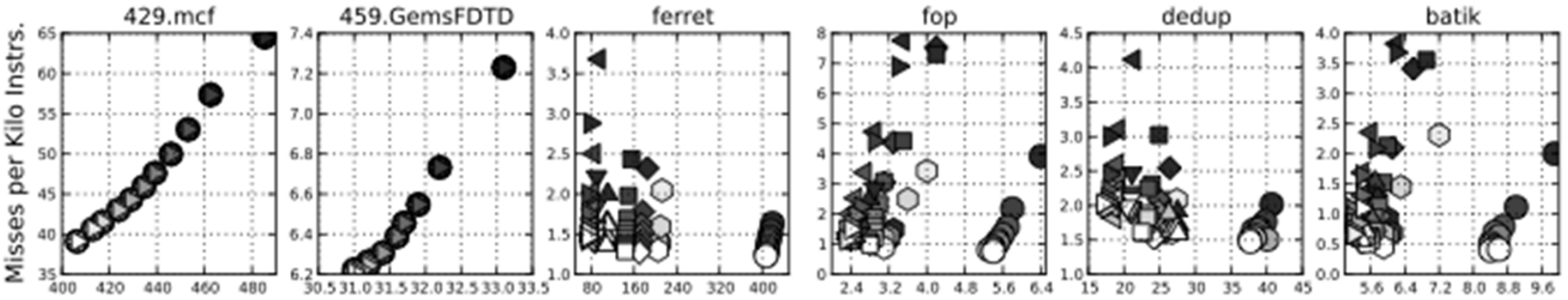


(b) DaCapo applications

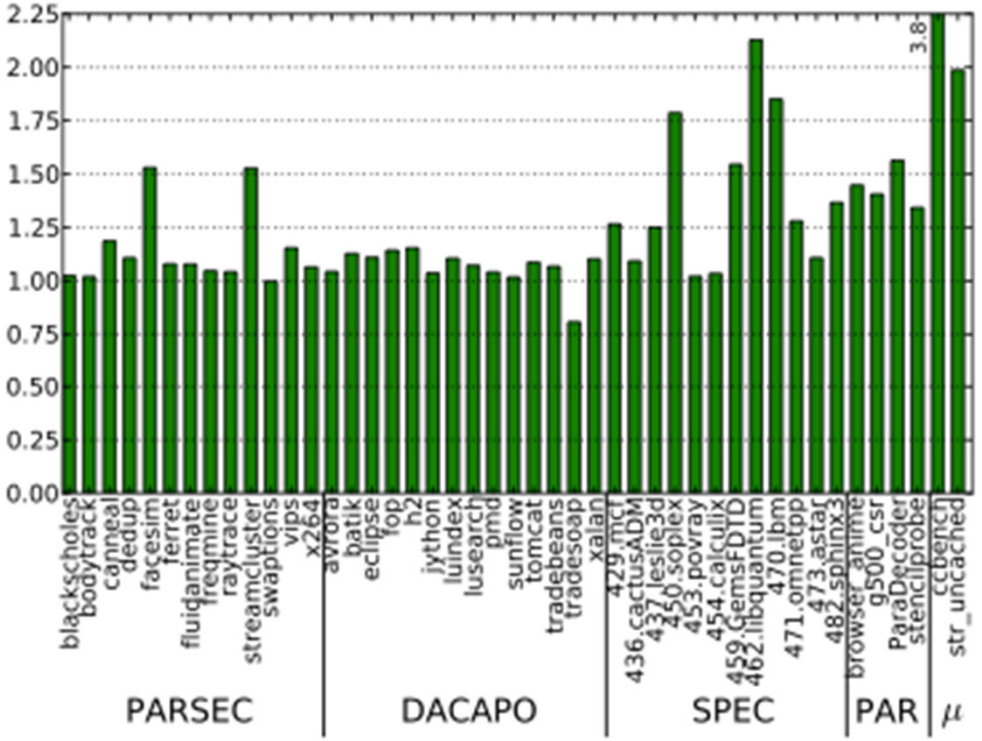


(c) Parallel applications and microbenchmarks

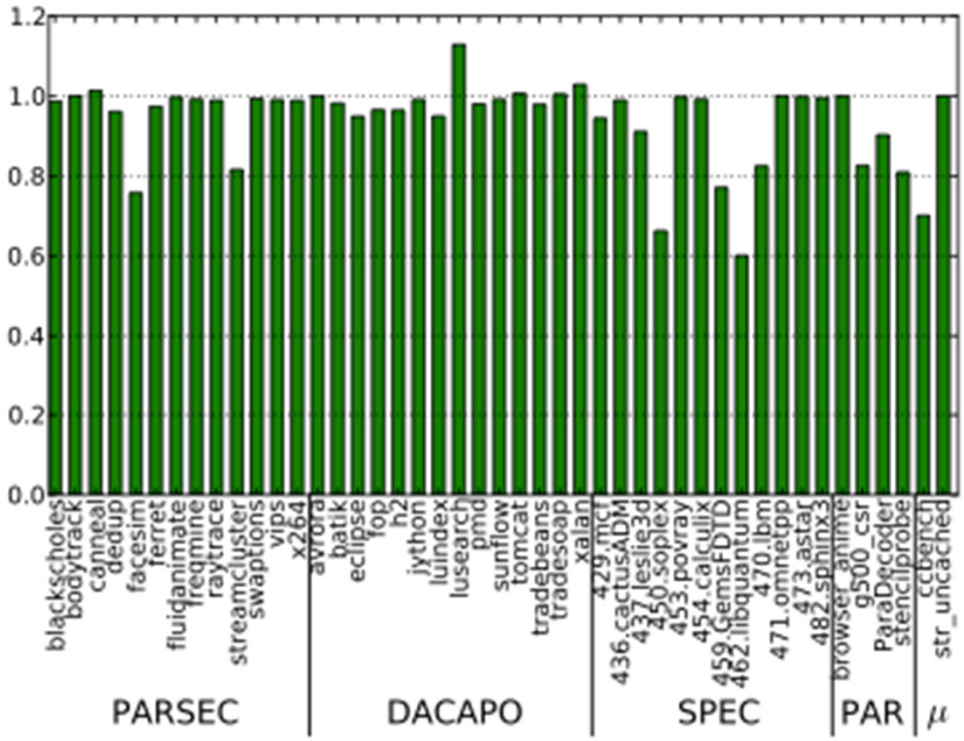
Utilization Diversity



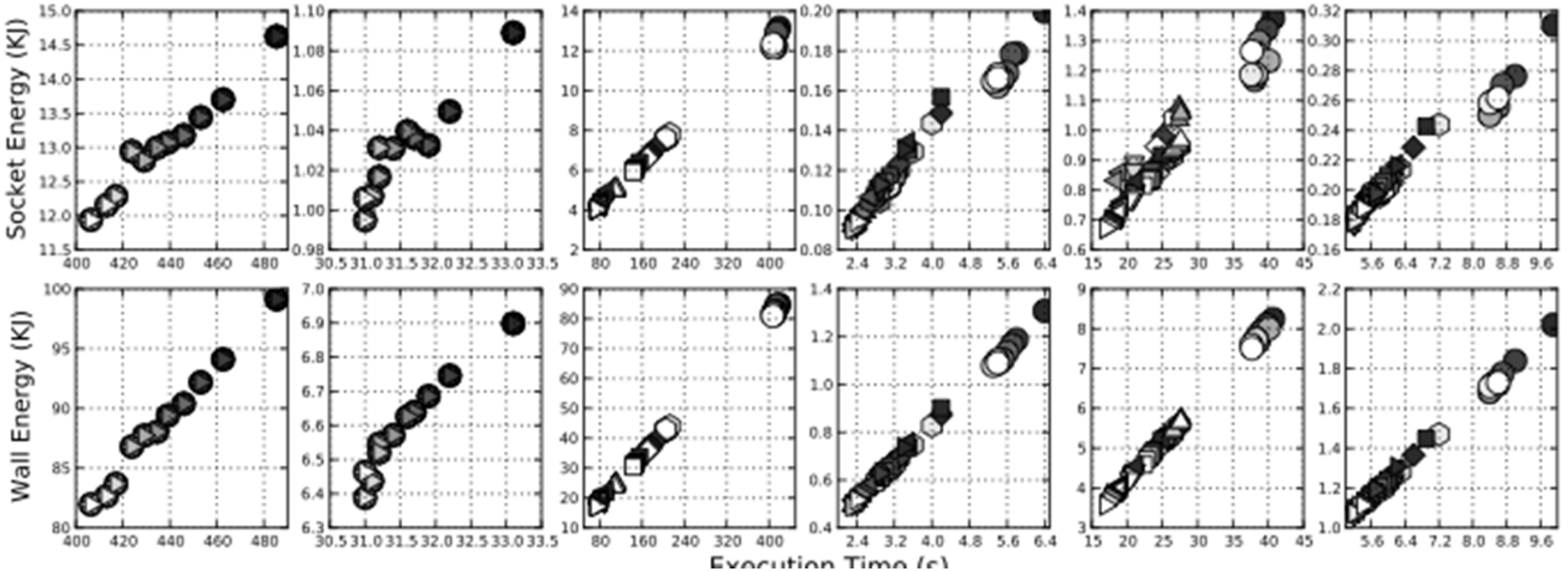
BW Hog

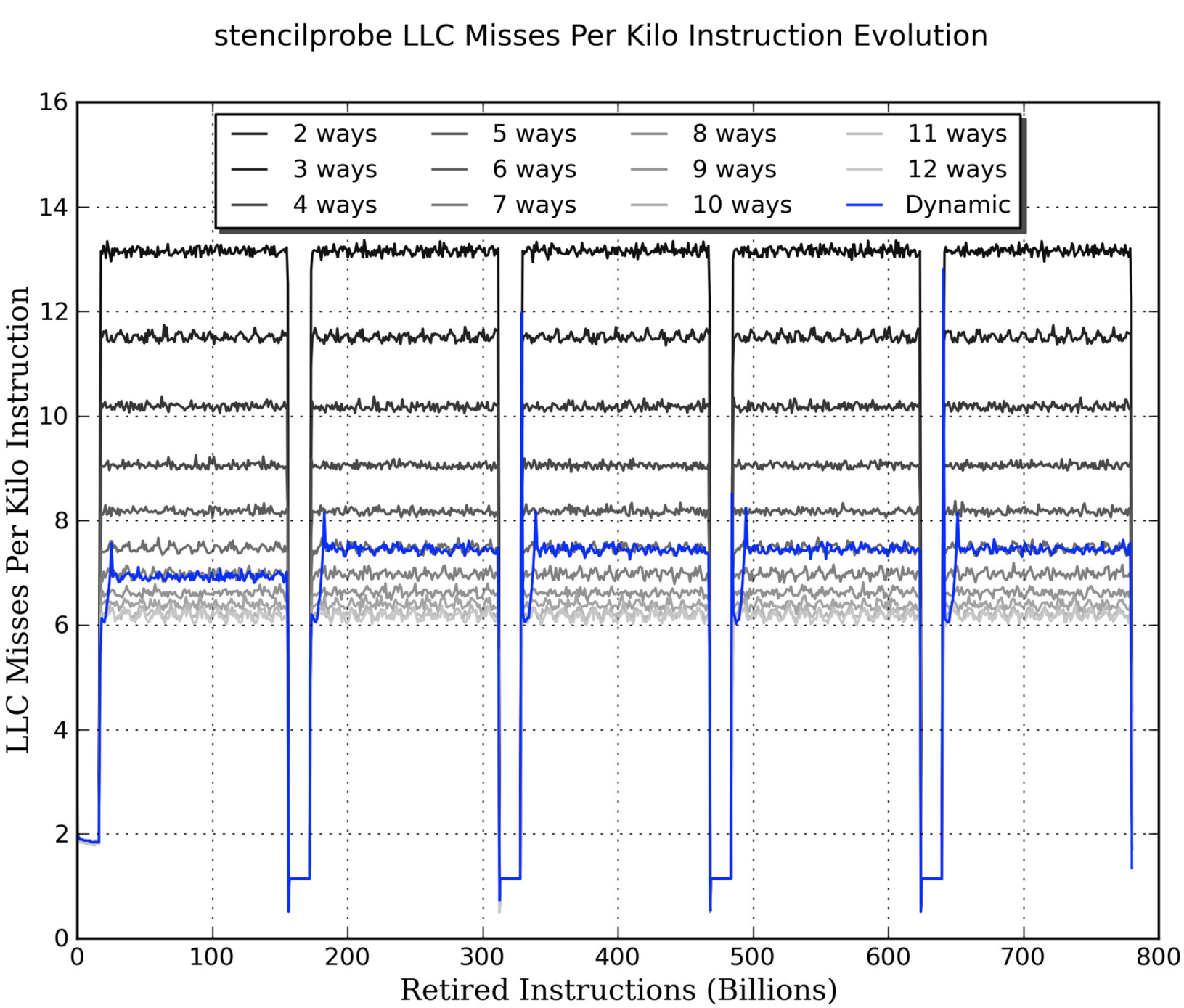

2.25
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
3.8
blackscholes
bodytrack
canneal
dedup
facesim
ferret
fluidanimate
freqmine
raytrace
streamcluster
swaptions
vips
x264
avrora
batik
eclipse
fop
h2
jython
luindex
lusearch
pmd
sunflow
tomcat
tradebeans
tradesoap
xalan
429.mcf
436.cactusADM
437.leslie3d
450.soplex
453.povray
454.calculix
459.GemsFDTD
462.libquantum
470.lbm
471.omnetpp
473.astar
482.sphinx3
browser_animex
g500_csr
ParaDecoder
stencilprobe
ccbench
str_uncached
PARSEC
DACAPO
SPEC
PAR
μ

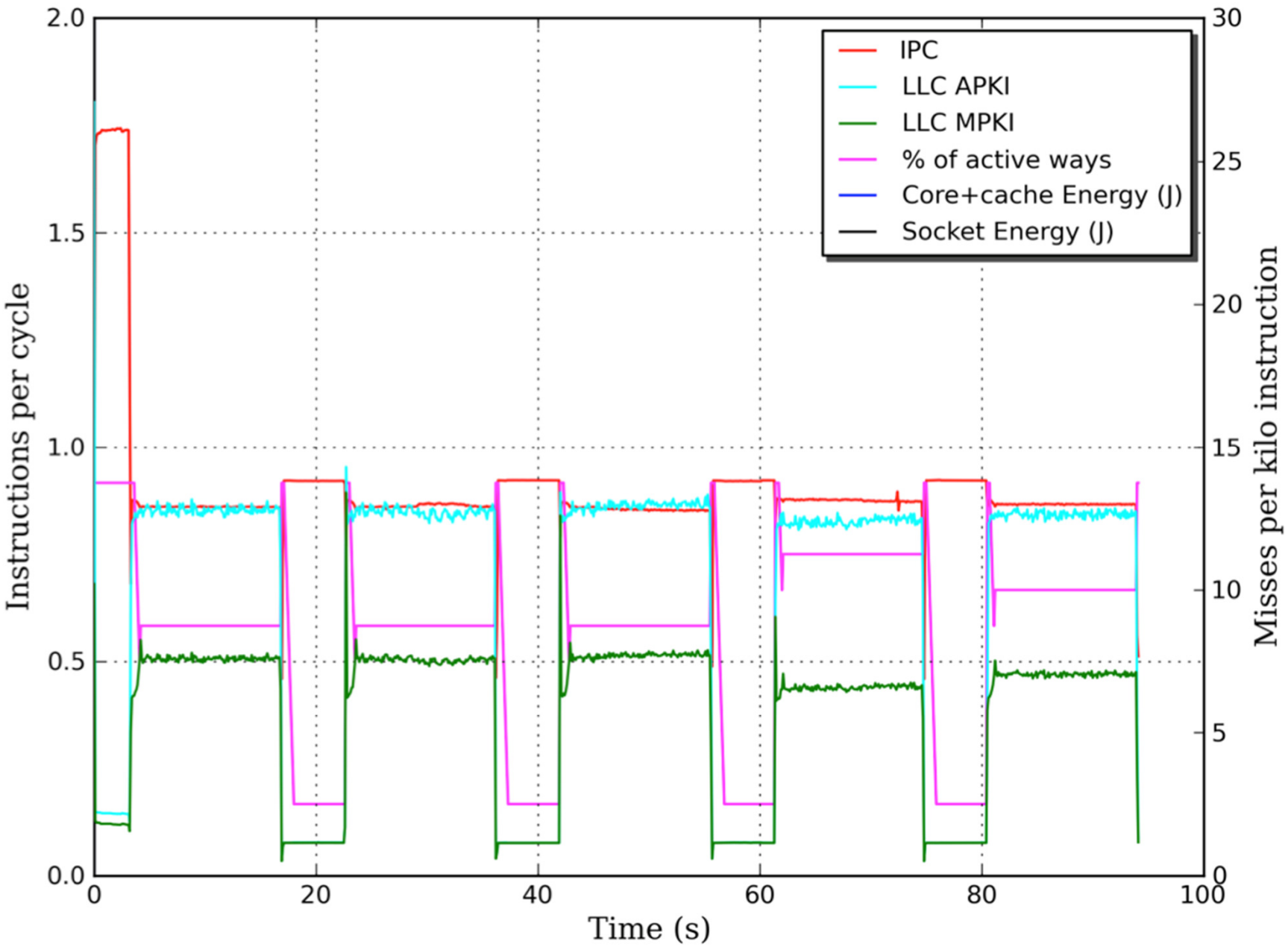
Prefetcher Sensitivity



Wall vs socket




stencilprobe LLC Misses Per Kilo Instruction Evolution
2 ways
3 ways
4 ways
5 ways
6 ways
7 ways
8 ways
9 ways
10 ways
11 ways
12 ways
Dynamic
LLC Misses Per Kilo Instruction
Retired Instructions (Billions)
0
2
4
6
8
10
12
14
16
0
100
200
300
400
500
600
700
800


stencilprobe-ccbench-native-ht4-g1-dynamic.out IPC
IPC
LLC APKI
LLC MPKI
% of active ways
Core+cache Energy (J)
Socket Energy (J)
Instructions per cycle
Misses per kilo instruction
Time (s)
0.0
0.5
1.0
1.5
2.0
0
5
10
15
20
25
30
0
20
40
60
80
100