THeME: A System for Testing by Hardware Monitoring Events

Kristen R. Walcott-Justice
kwalcott@uccs.edu
University of Colorado - Colorado Springs

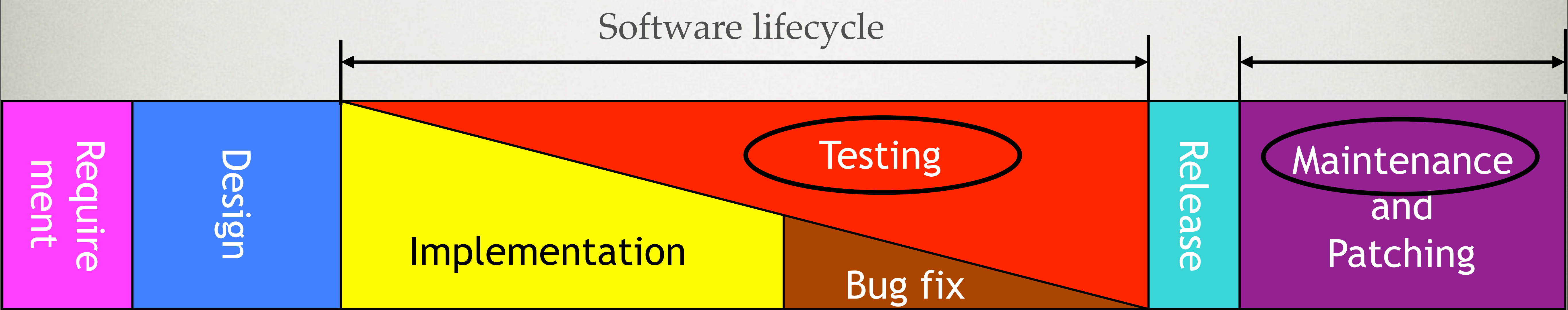
Jason Mars
jom5x@cs.virginia.edu
University of California - San Diego

Mary Lou Soffa
soffa@cs.virginia.edu
University of Virginia

ISSTA 2012, July 17, Minneapolis, MN

Computer Science at the University of Virginia

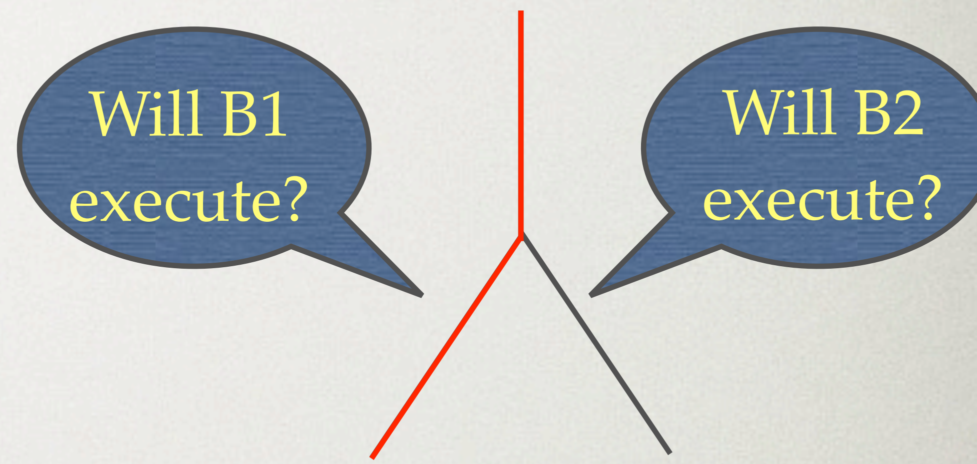
Developing Reliable Software



- Measuring test quality:
  - Recompilation
  - High run time overheads
  - Large code growth

Expense of Traditional Test Coverage Analysis

- Instrumentation
  - Probe
  - Payload
- Branch analysis overheads:
  - Time: 10% - 30%
  - Code growth: 60% - 90%



Branch	Executed?
B1	√
B2	

Efficient Program Monitoring

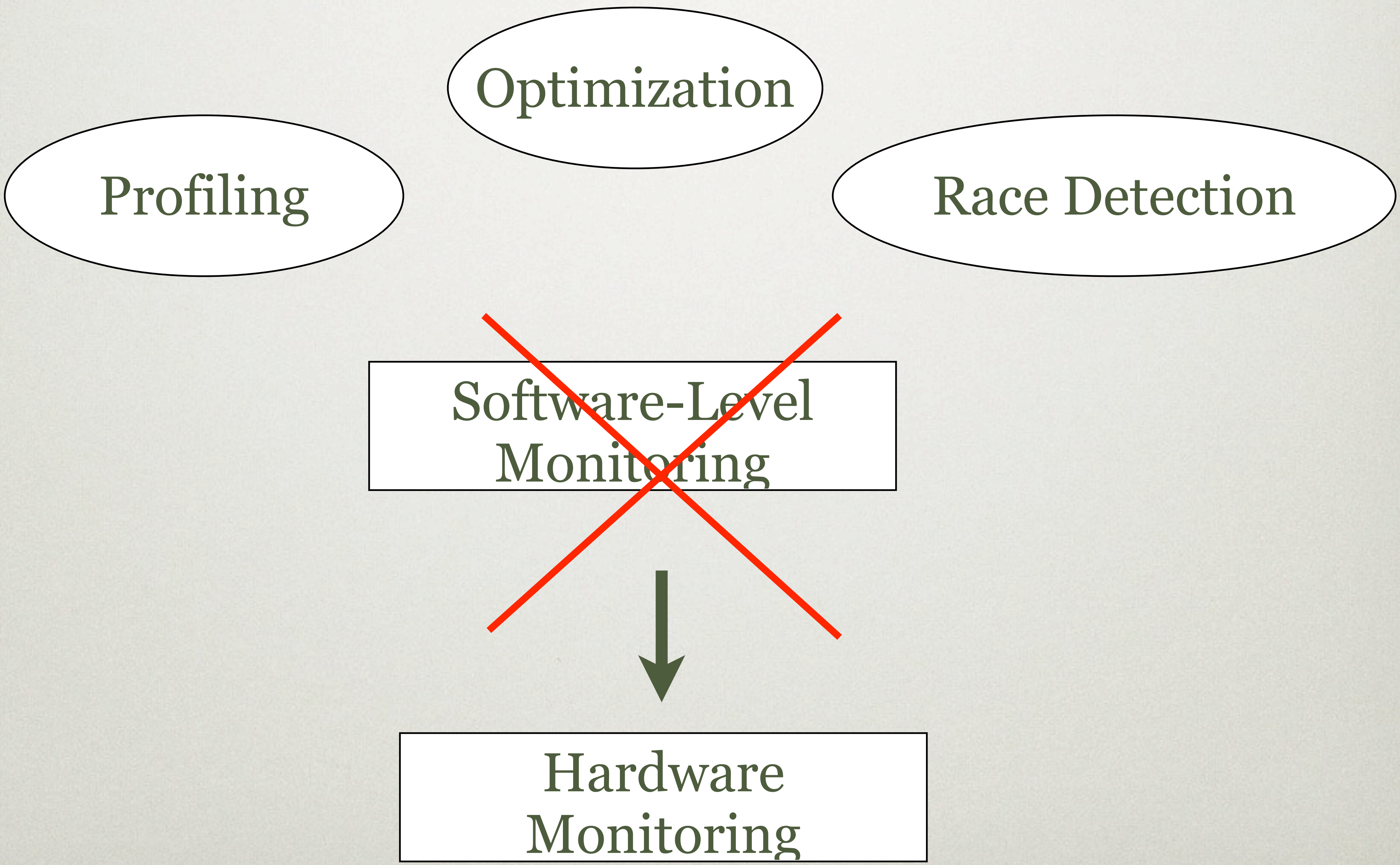
Optimization

Profiling

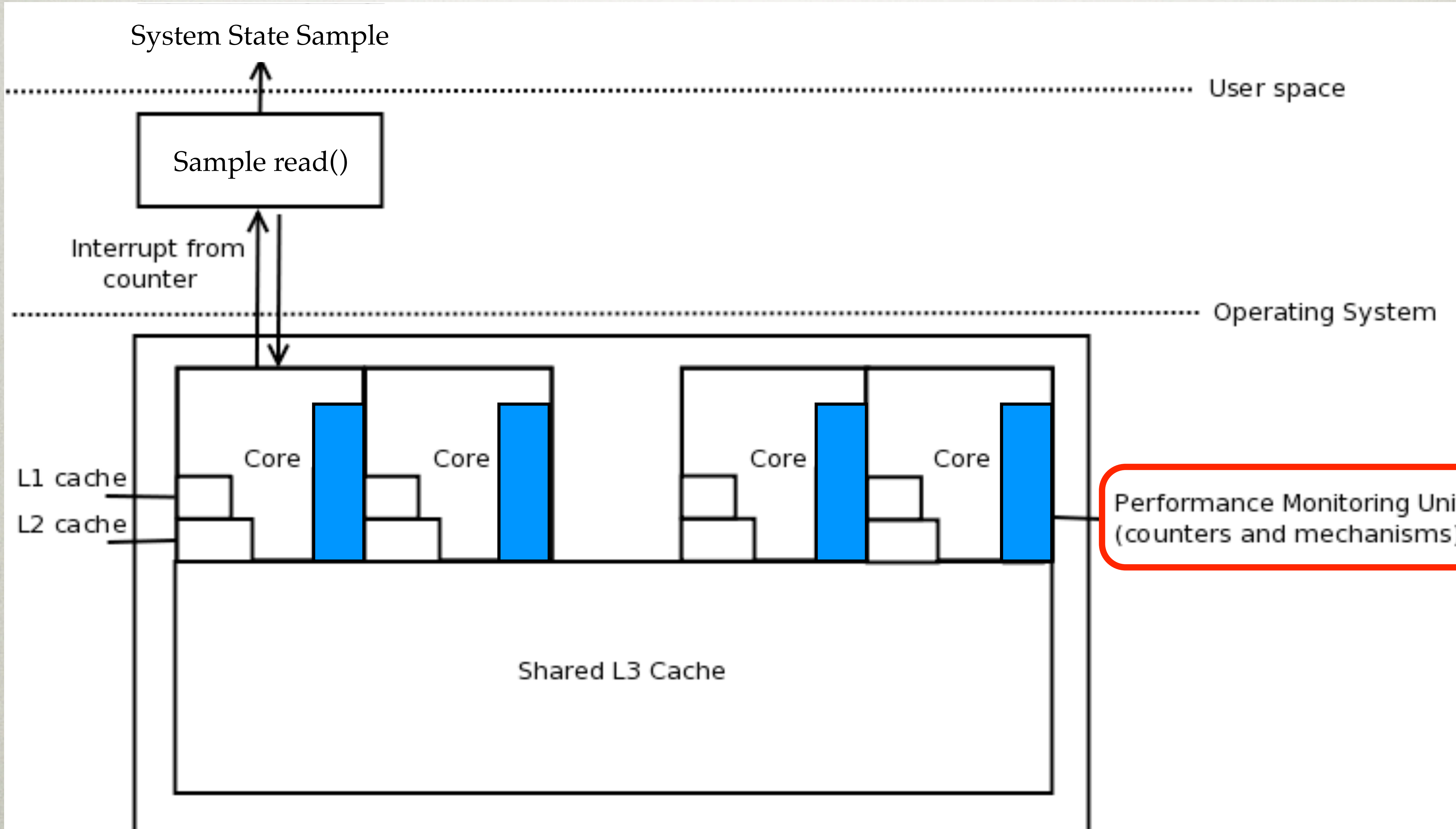
Race Detection

Software-Level Monitoring

Efficient Program Monitoring



What is a Hardware Mechanism?



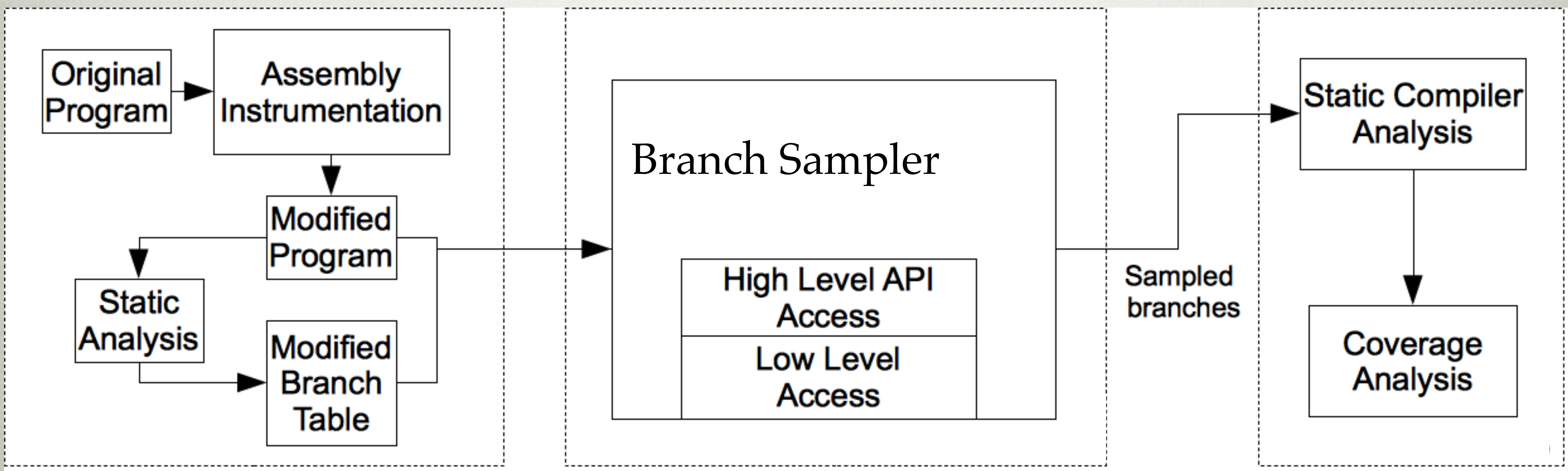
Using Hardware Mechanisms

- Developed for operating system performance analysis
- Widely available on nearly all processors
- Low overhead
  - Short setup time (318μs)
  - Quick read time (3.5μs)
- Use of samples
  - Estimate profiles
  - Reveal program execution behavior
- Removes need for instrumentation

Hardware Mechanisms in Testing: Goals and Challenges

- Structural testing requires more exact data
  - Can we capture ALL events with which we are concerned?
  - Can we capture ONLY the events with which we are concerned?
- Tradeoff:
  - Amount of information collected
  - Overhead of sampling

THeME: Testing by Hardware Monitoring Events

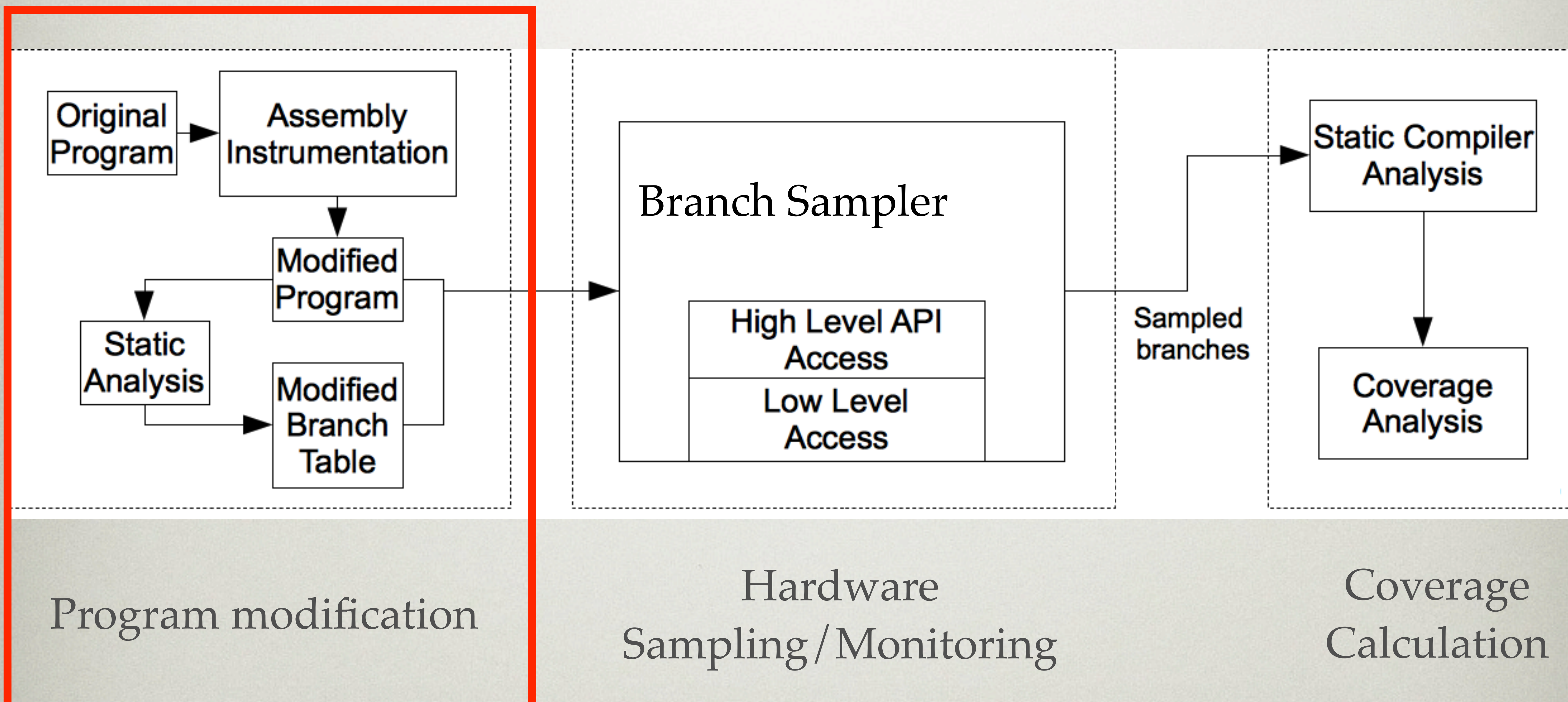


Program modification

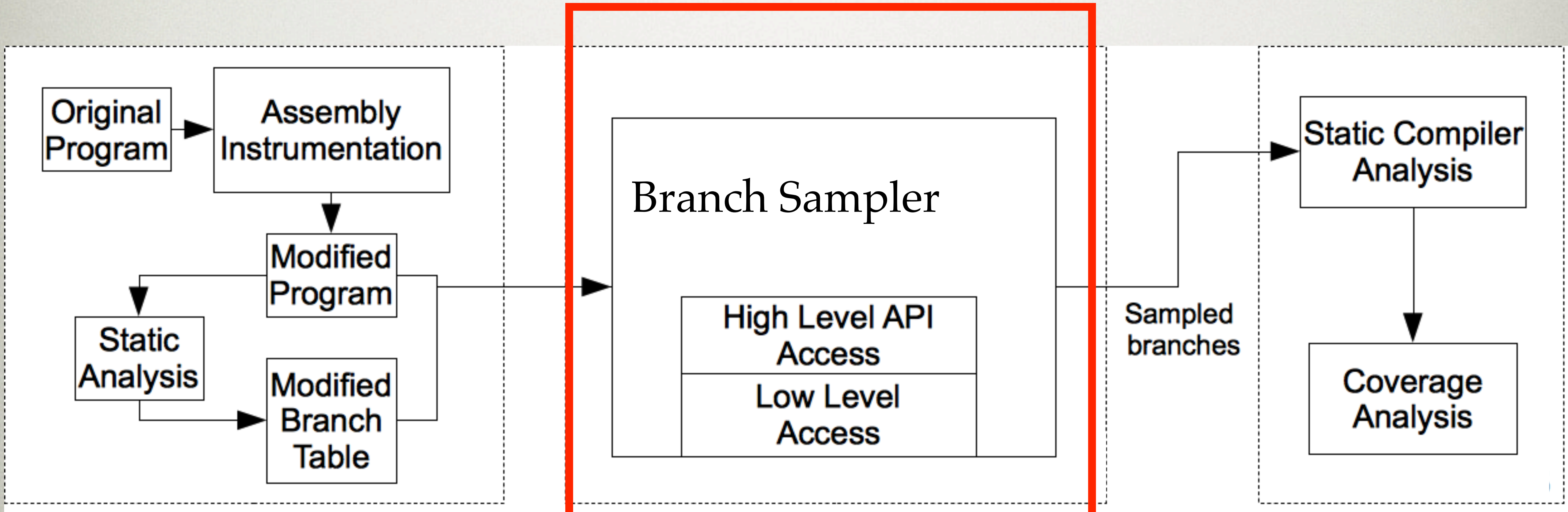
Hardware Sampling/Monitoring

Coverage Calculation

THeME: Testing by Hardware Monitoring Events



THeME: Testing by Hardware Monitoring Events

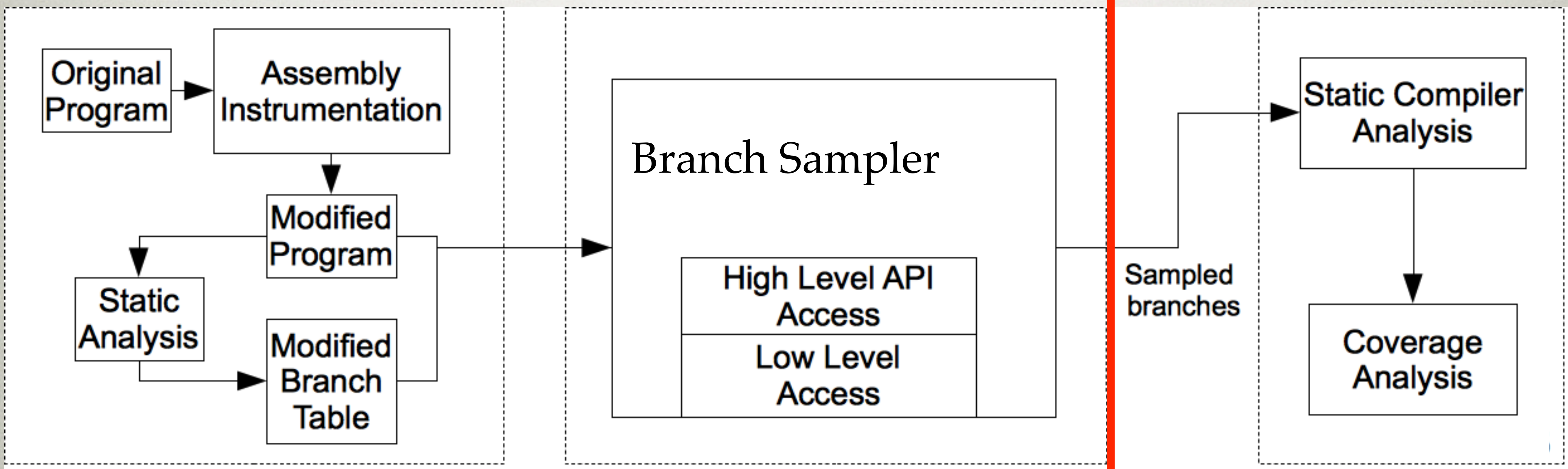


Program modification

Hardware Sampling/Monitoring

Coverage Calculation

THeME: Testing by Hardware Monitoring Events

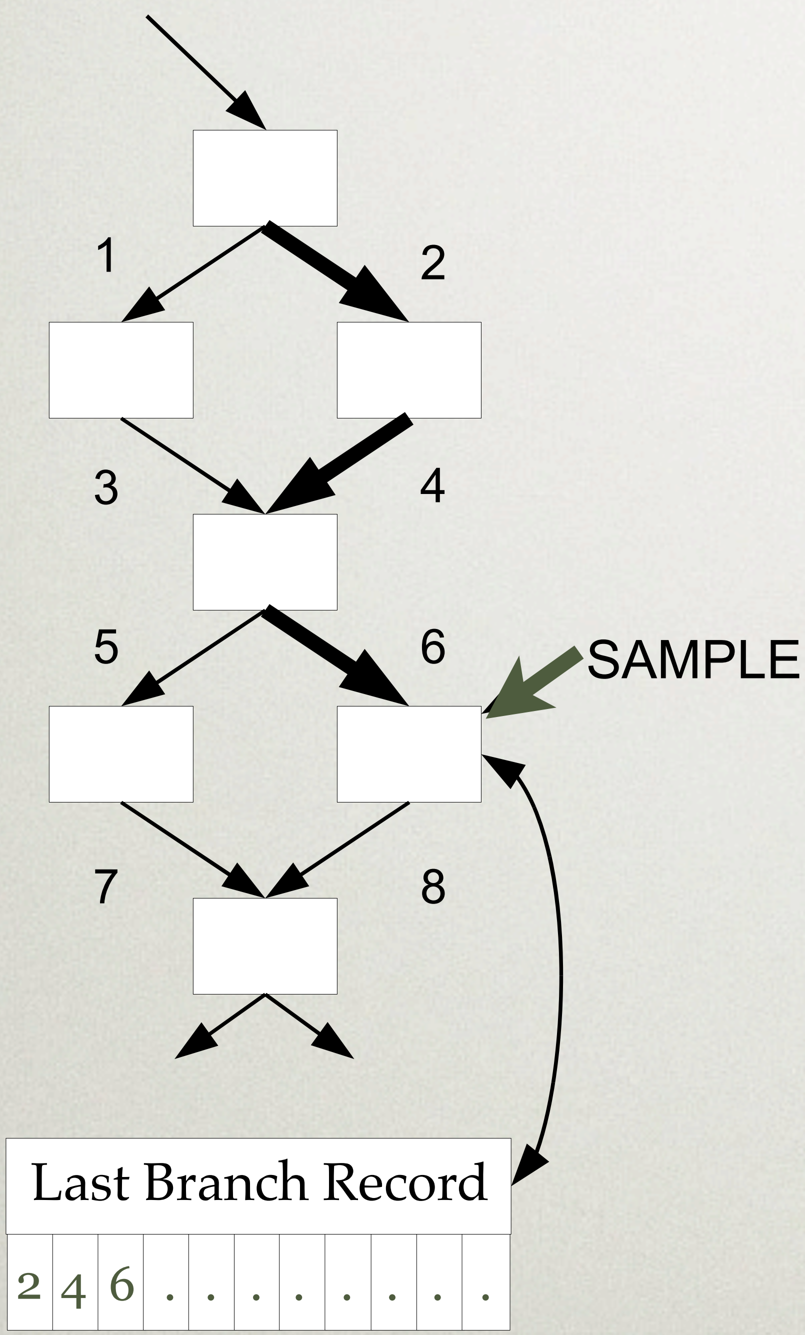


Program modification

Hardware Sampling/Monitoring

Coverage Calculation

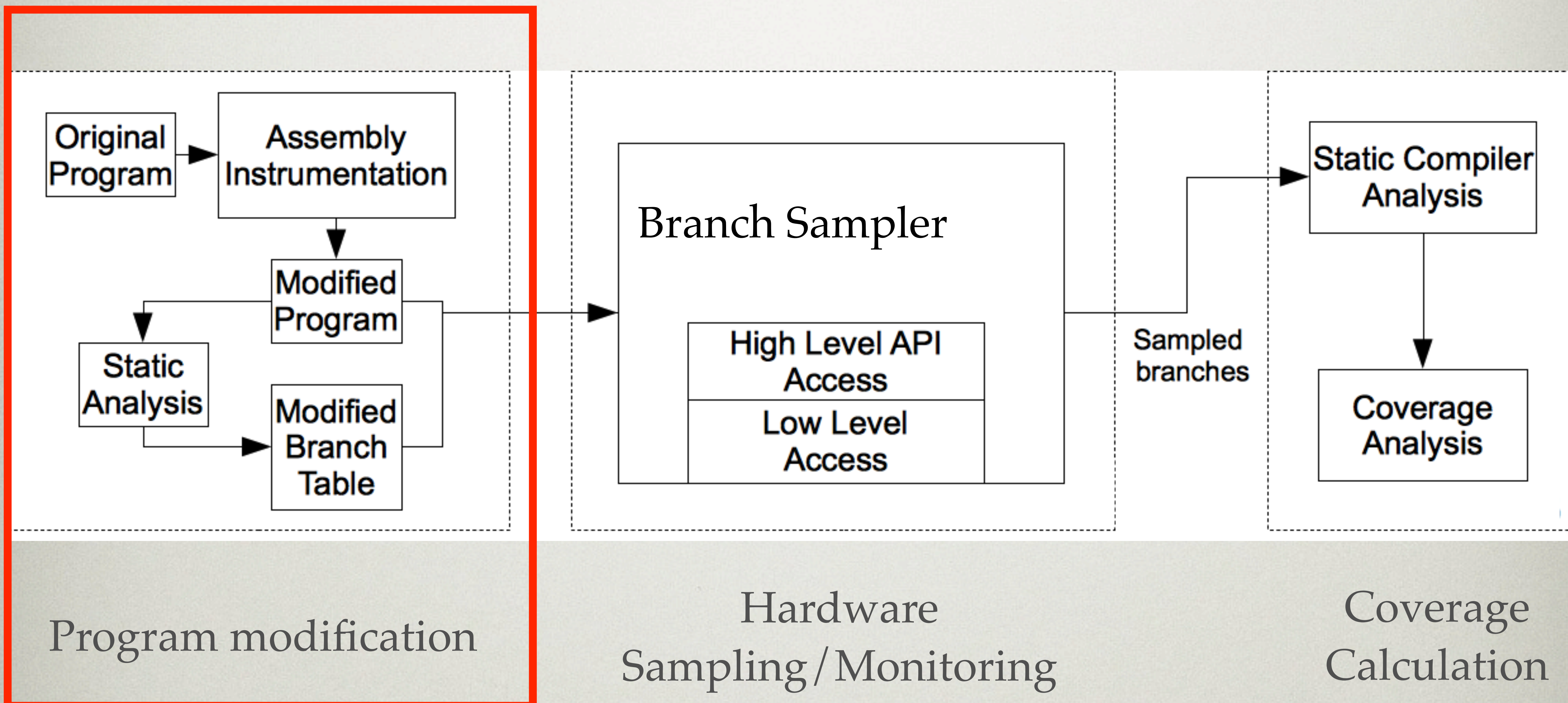
Branch Vector Recording: Last Branch Record (LBR)



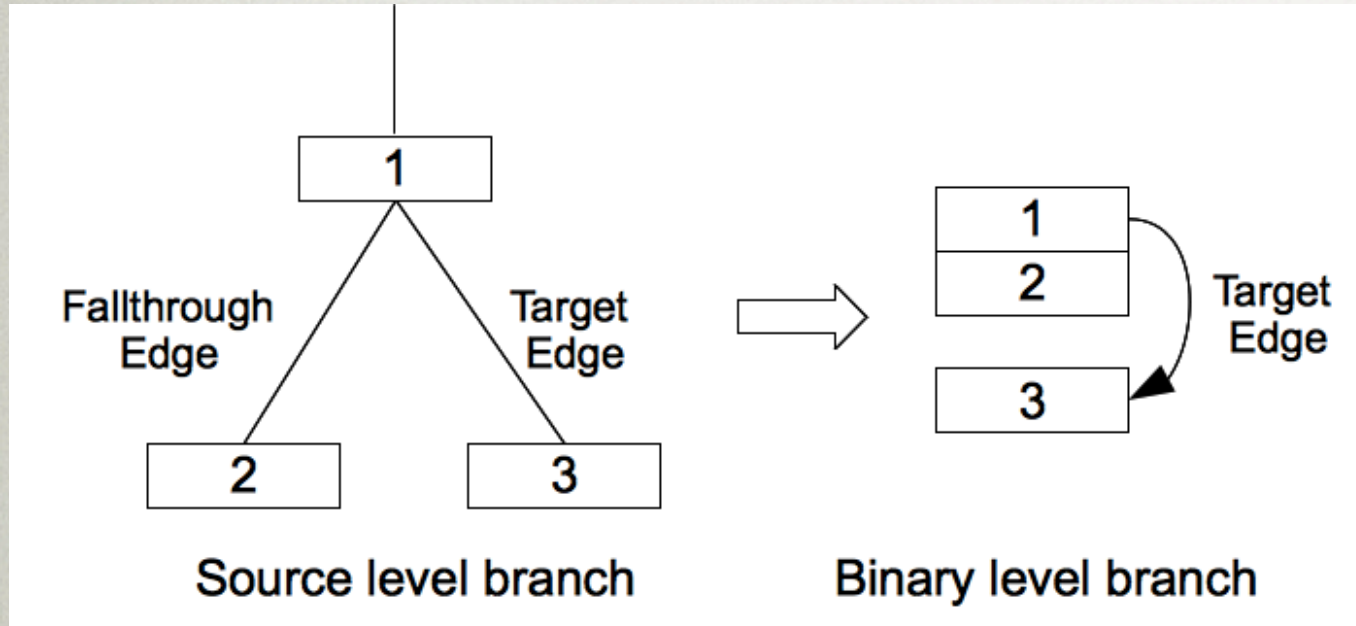
Branch Vector (≤16 branches)

- Mechanism for partial branch profiling
  - Intended for OS performance and debugging
- Tracks set of executed branches
  - Branch source
  - Branch destination
- Sample == Set of branches "Branch Vector"

THeME: Testing by Hardware Monitoring Events



Enabling Fall-through Visibility

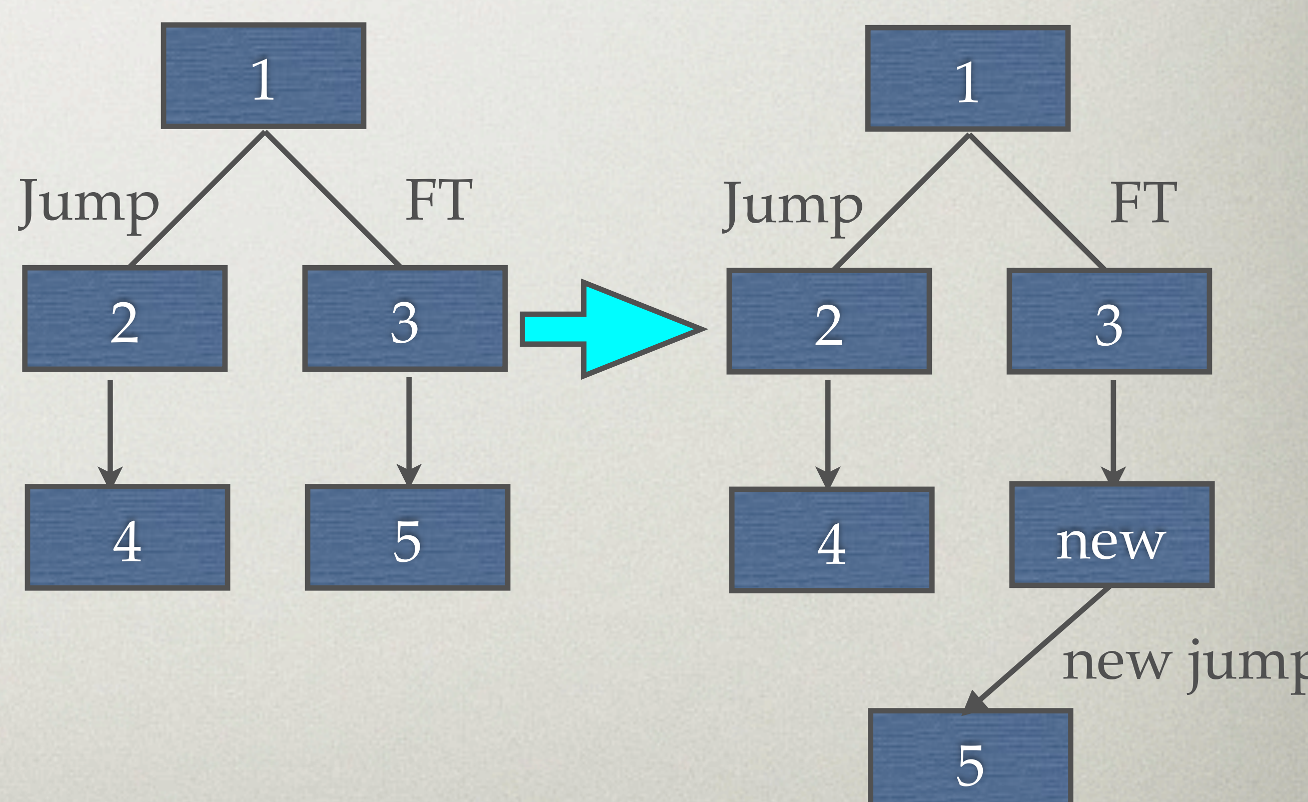


Challenge:

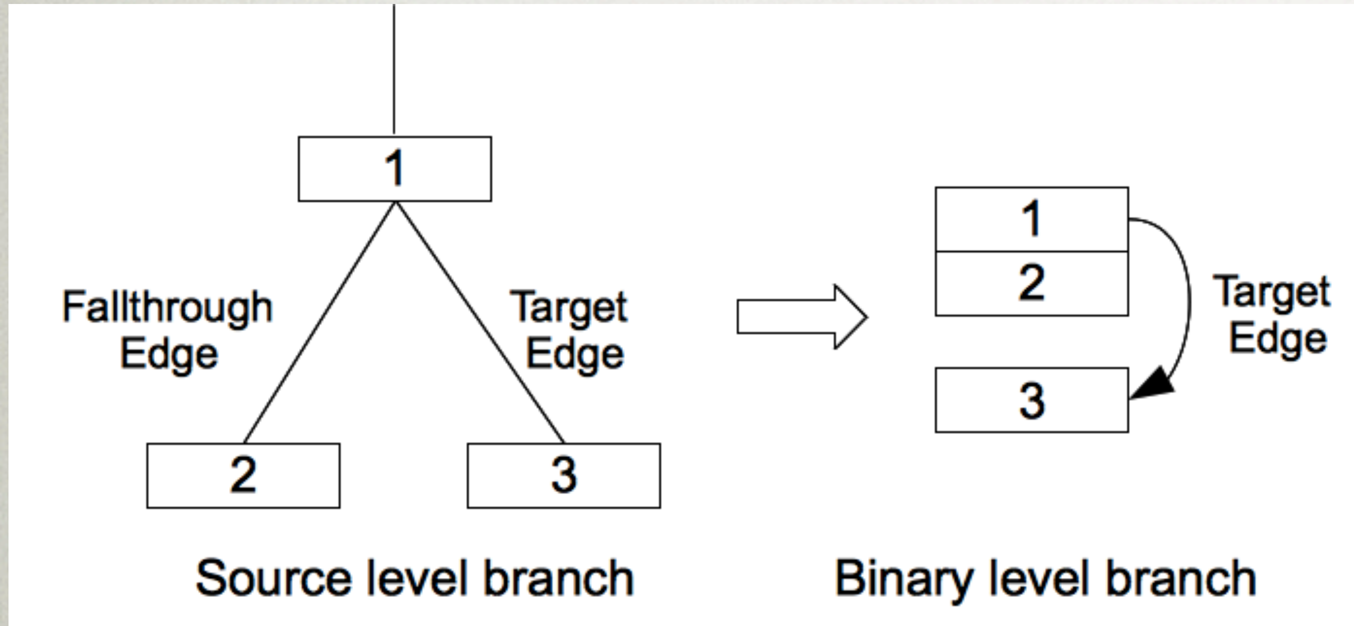
Hardware branch-based monitors can only see 1 of 2 branch edges

- Methods
  - Supplement with more samples
  - Use static analysis to infer branches
  - Minor program modification
- Our Solution:

  Insert innocuous unconditional branches



Enabling Fall-through Visibility

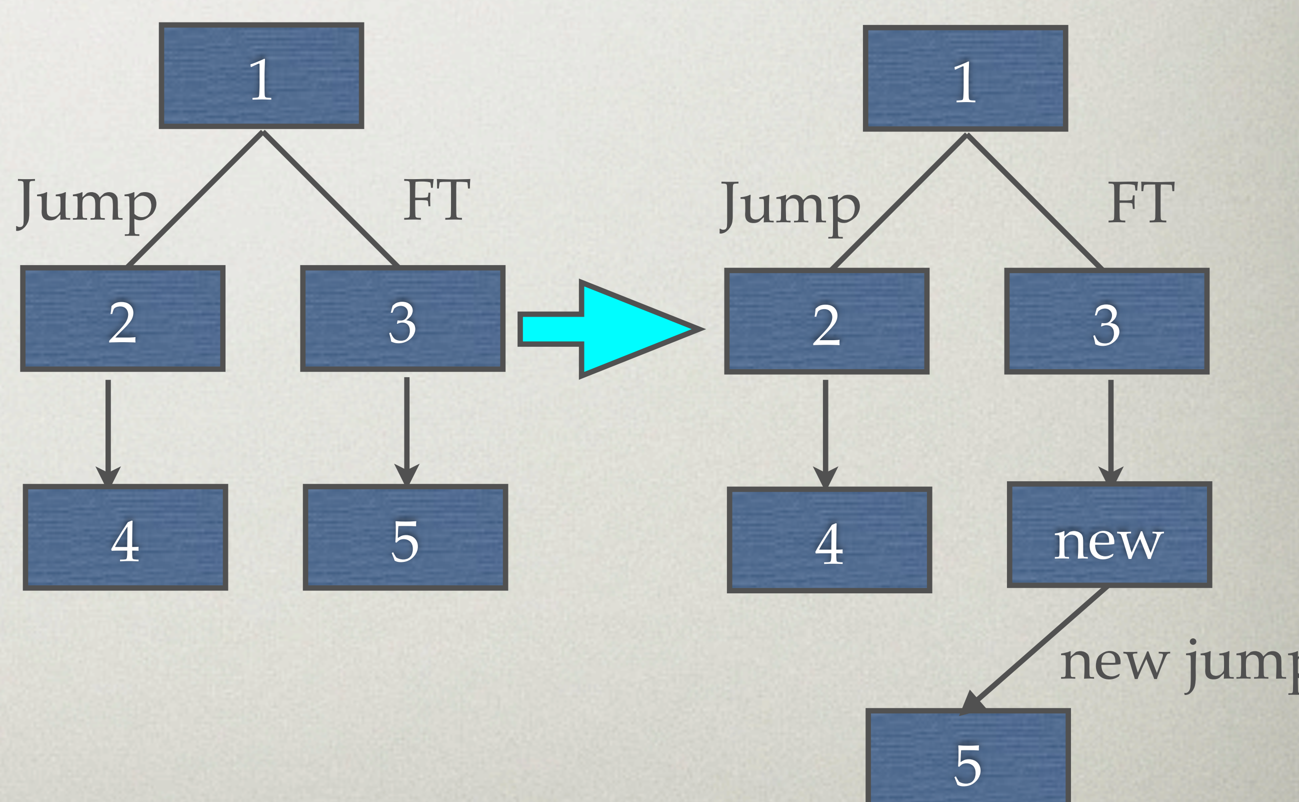


Challenge:

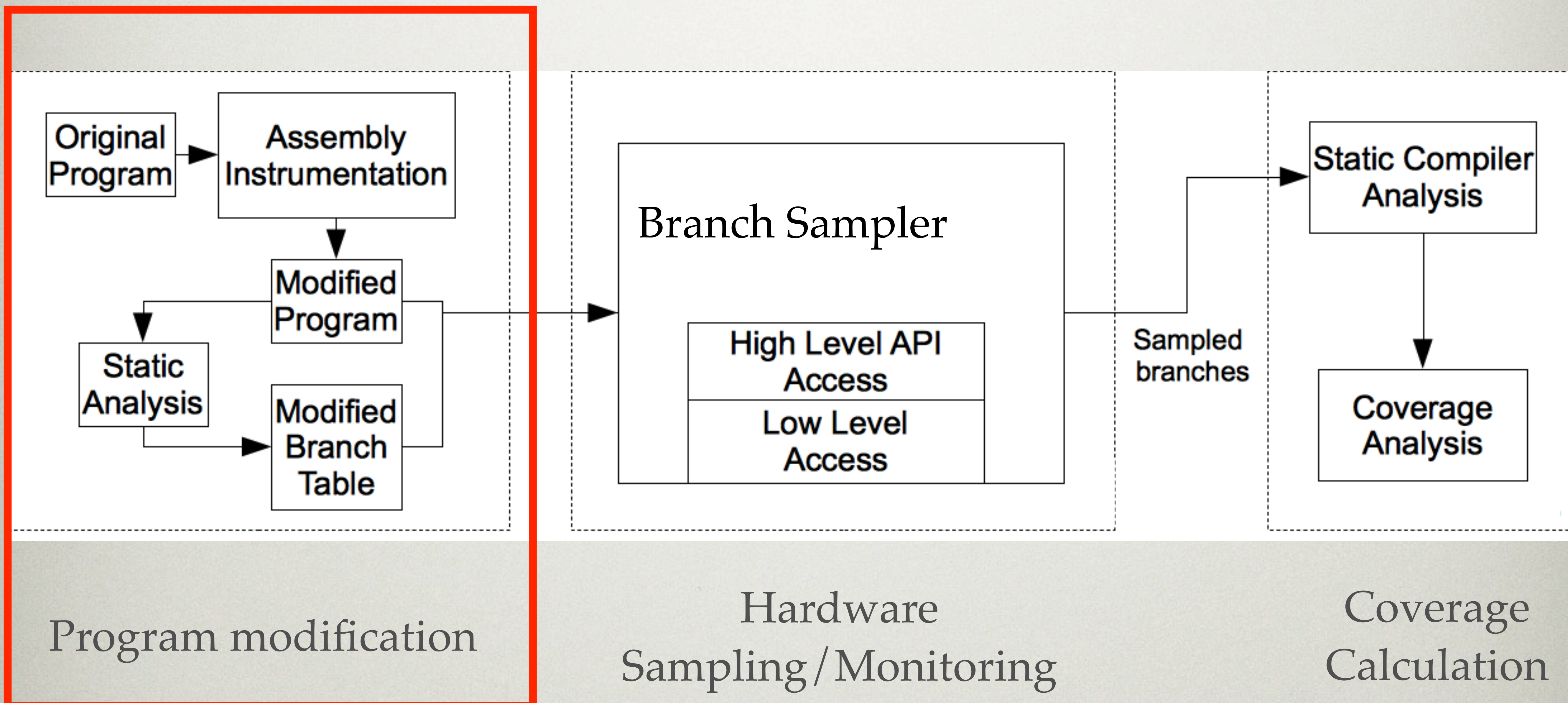
Hardware branch-based monitors can only see 1 of 2 branch edges

- Methods
  - Supplement with more samples
  - Use static analysis to infer branches
  - Minor program modification
- Our Solution:

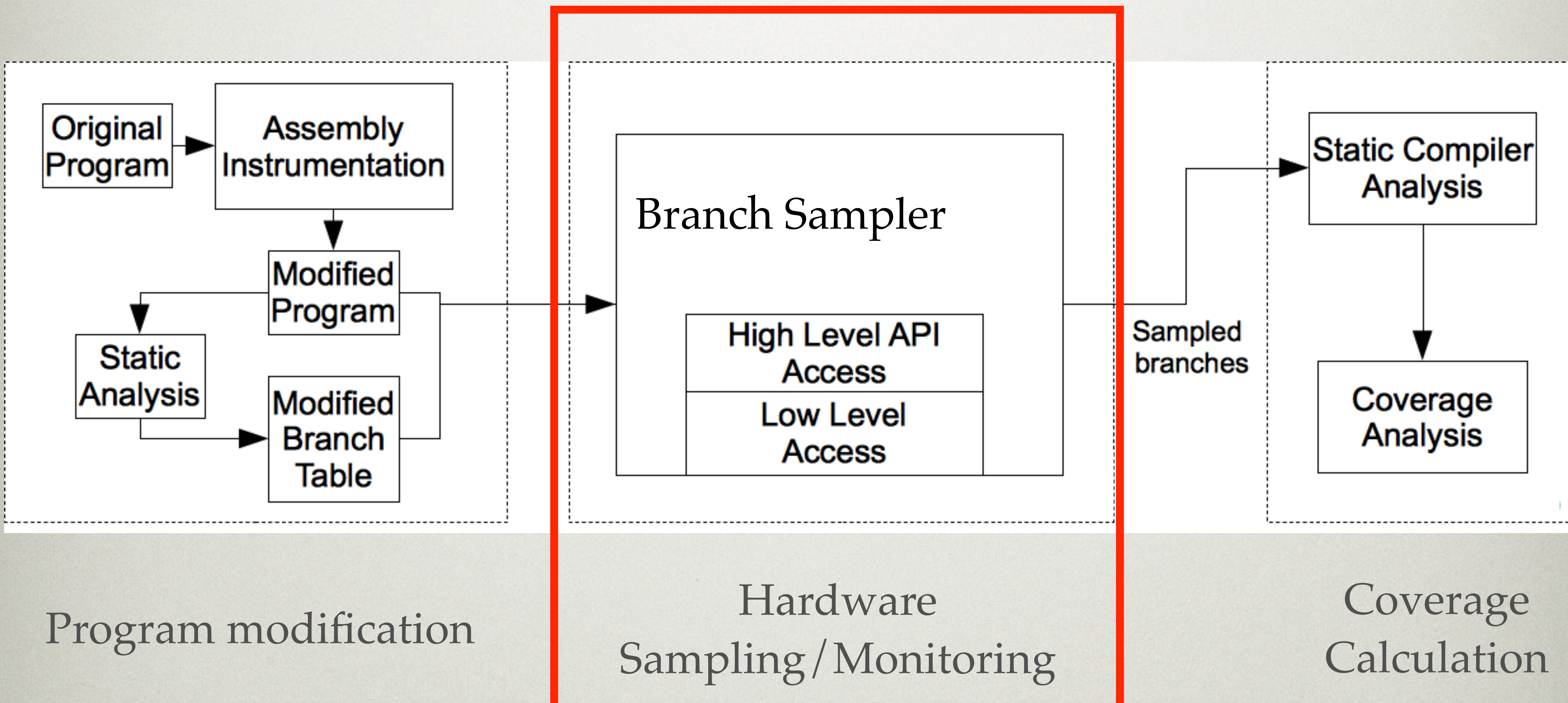
  Insert innocuous unconditional branches



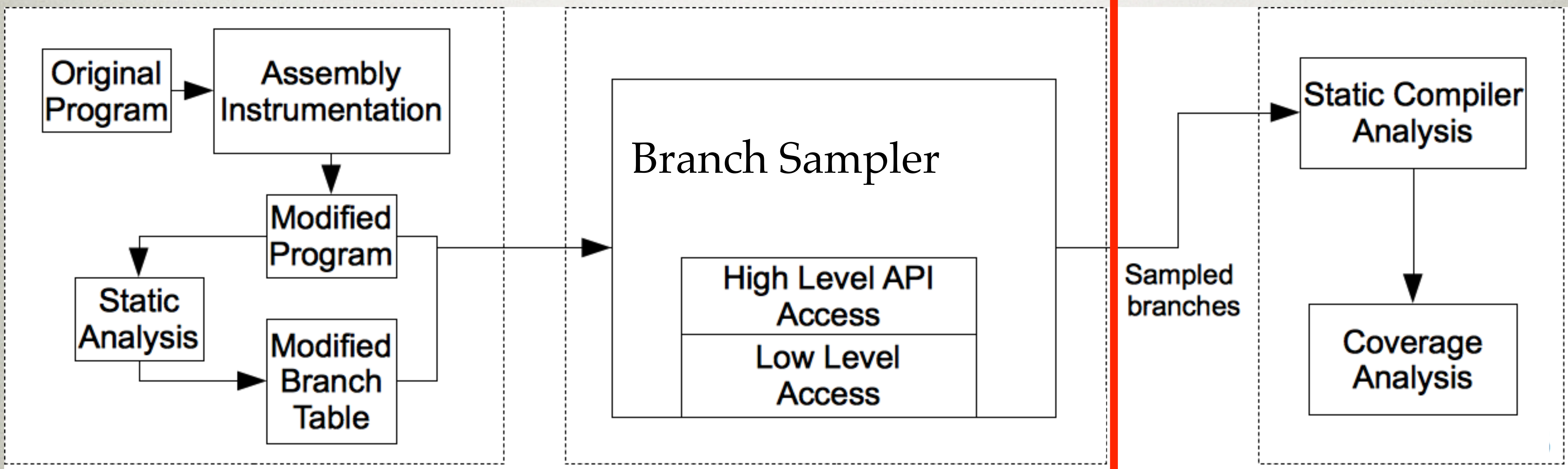
THeME: Testing by Hardware Monitoring Events



THeME: Testing by Hardware Monitoring Events



THeME: Testing by Hardware Monitoring Events



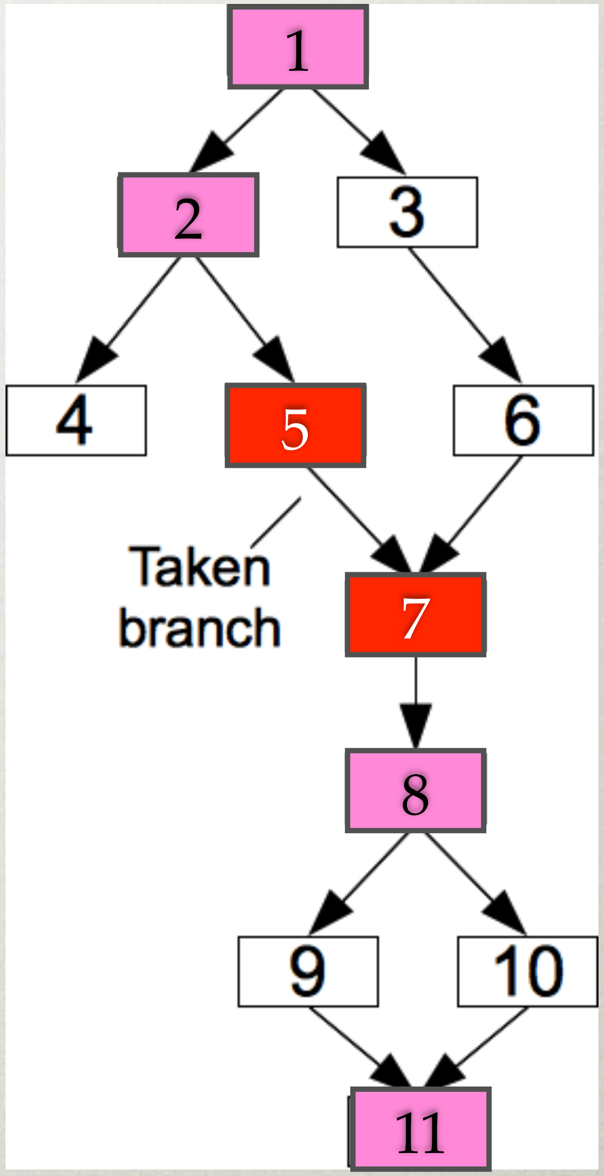
Program modification

Hardware Sampling/Monitoring

Coverage Calculation

Improving Branch Coverage

- Sampling → Some missed data
- Goal: Improve coverage using static analysis
- Dominator analysis
  - Associate seen branches with control flow graph
  - Branch *b* executed → branch *c* also executed



Experiment and System Design

- Intel Core i7 860 quad-core processor
  - LBR size of 16 branches
- Linux 2.6.34
- Hardware access tools: libpfm4 (user-level), perf (kernel-level)

- SPEC2006 C Benchmarks
- Metrics:
  - Efficiency- time
  - Code growth size
  - Effectiveness- branch coverage
    - Instrumented vs Hardware Monitoring

Results: Enabling Fall-Through Visibility

- Impact:
  - Increases time overhead
  - Increases code growth
- How compared to instrumentation?

Time overhead

Benchmark	Branch Cov.	Time (s)	Mod. Time (s)	Instr. Time (s)
bzip2	64.20%	1499	1514	1599
h264ref	35.72%	1753	1786	1890
libquantum	39.07%	1056	1178	1236
mcf	74.01%	529	539	575
sjeng	48.87%	1028	1162	1312

Avg: 5% increase

Avg: 14% increase

Results: Enabling Fall-Through Visibility

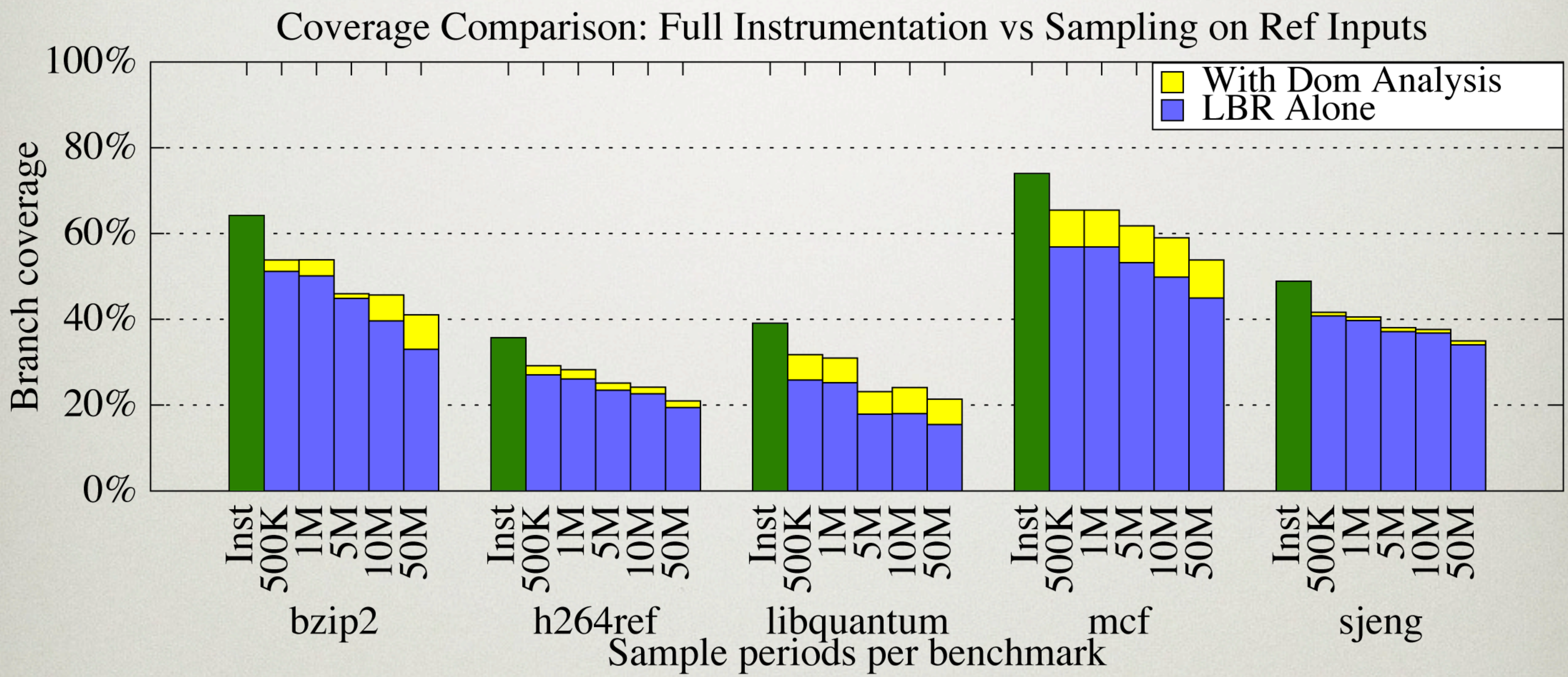
- Impact:
  - Increases time overhead
  - Increases code growth
- How compared to instrumentation?

Code Growth

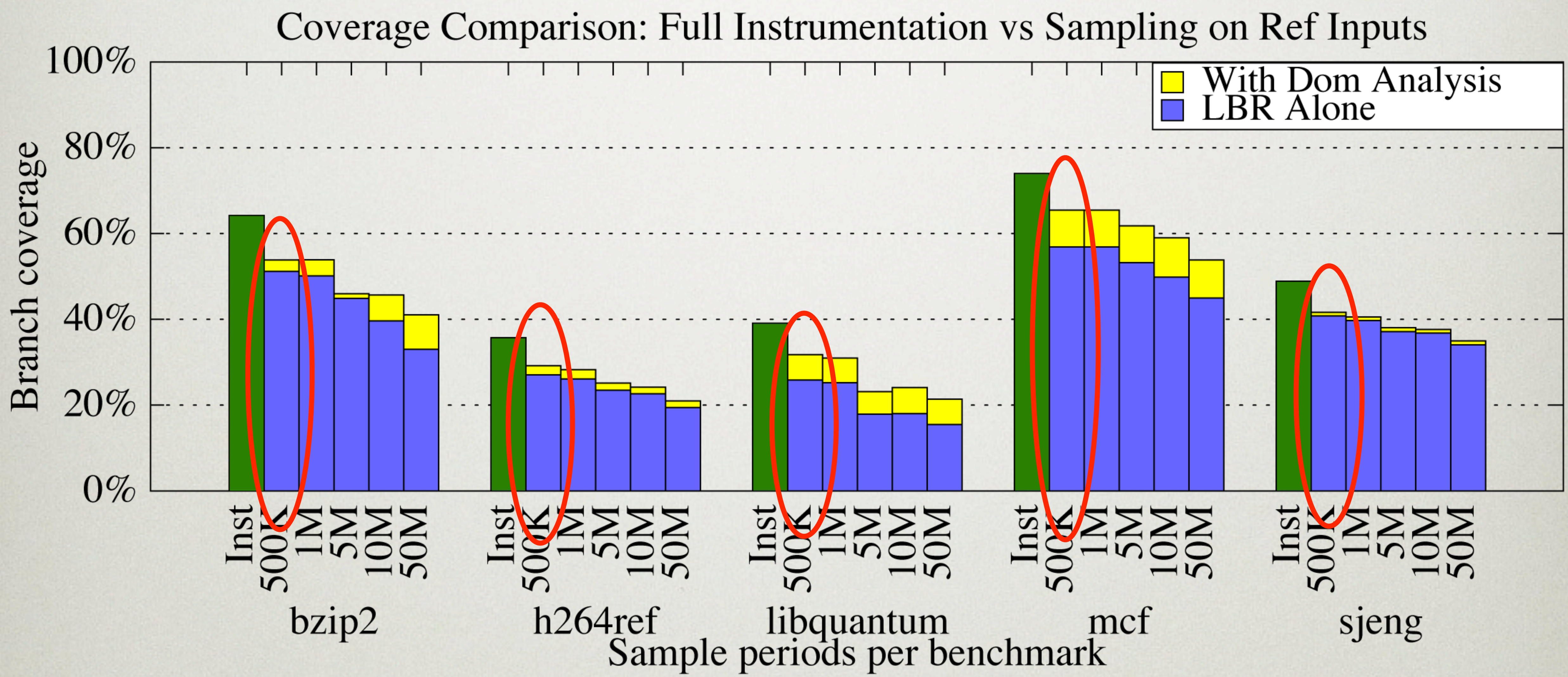
Benchmark	Native Size (kB)	Mod. % Increase	Instr. % Increase
bzip2	260 kB	1.52	32.65
h264ref	2892 kB	0.69	18.39
libquantum	208 kB	0	20.00
mcf	128 kB	0	17.95
sjeng	592 kB	0.67	30.05

Avg: 0.5% Avg: 24%

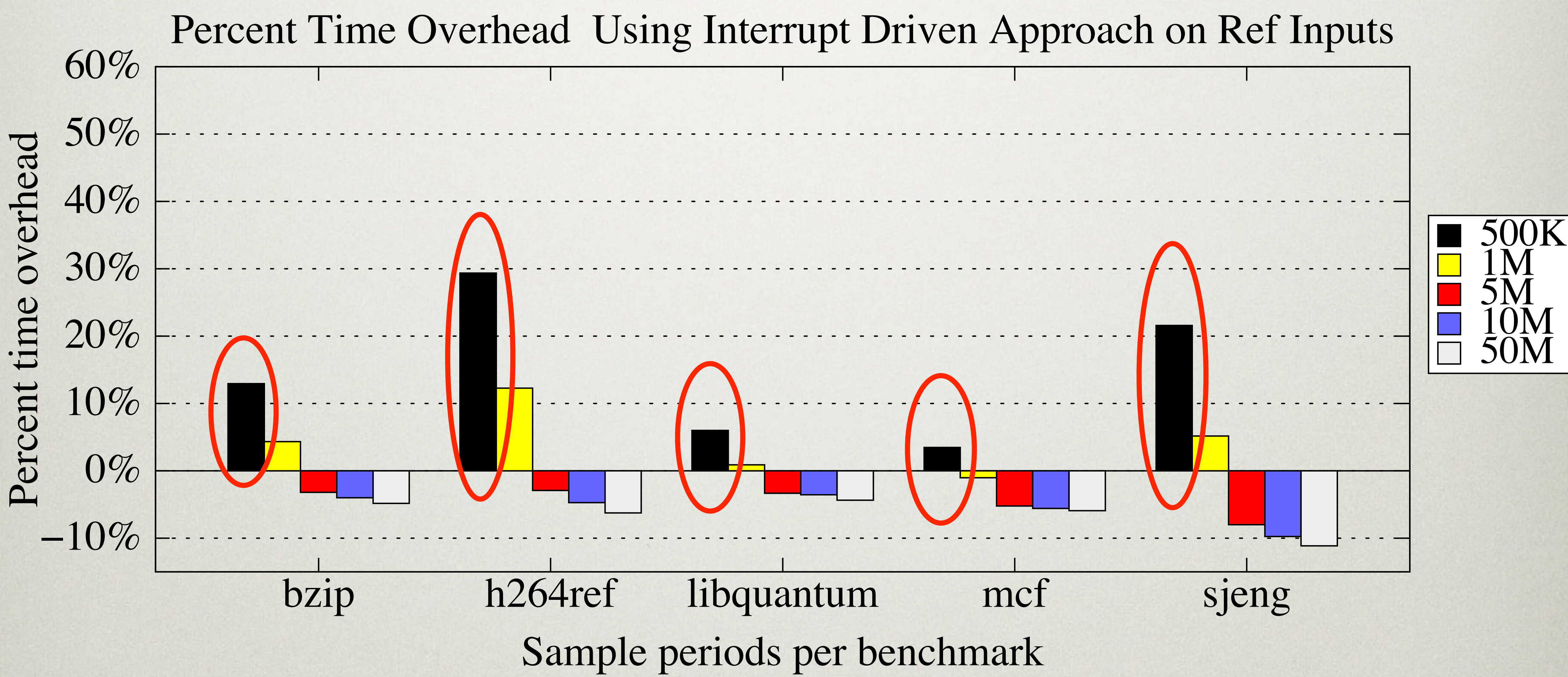
Results: Testing on a Single Core - Effectiveness



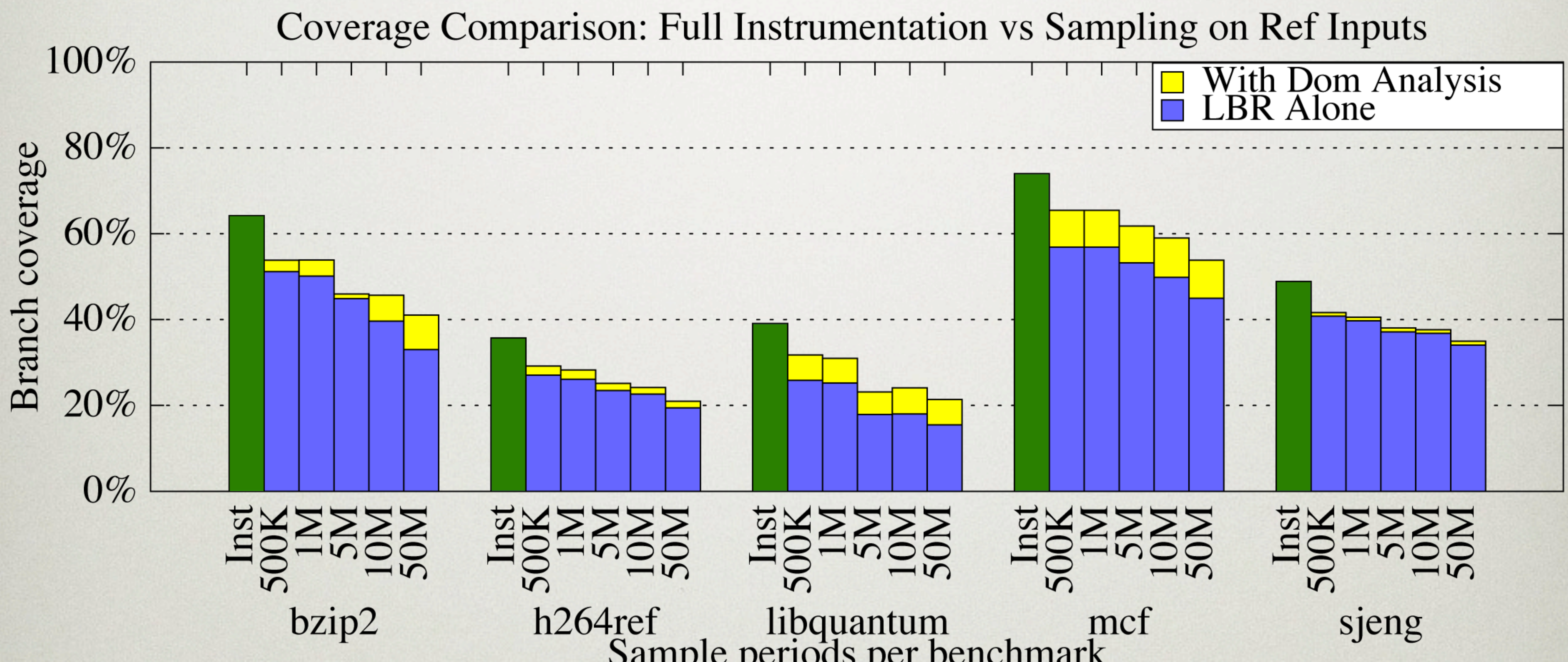
Results: Testing on a Single Core - Effectiveness



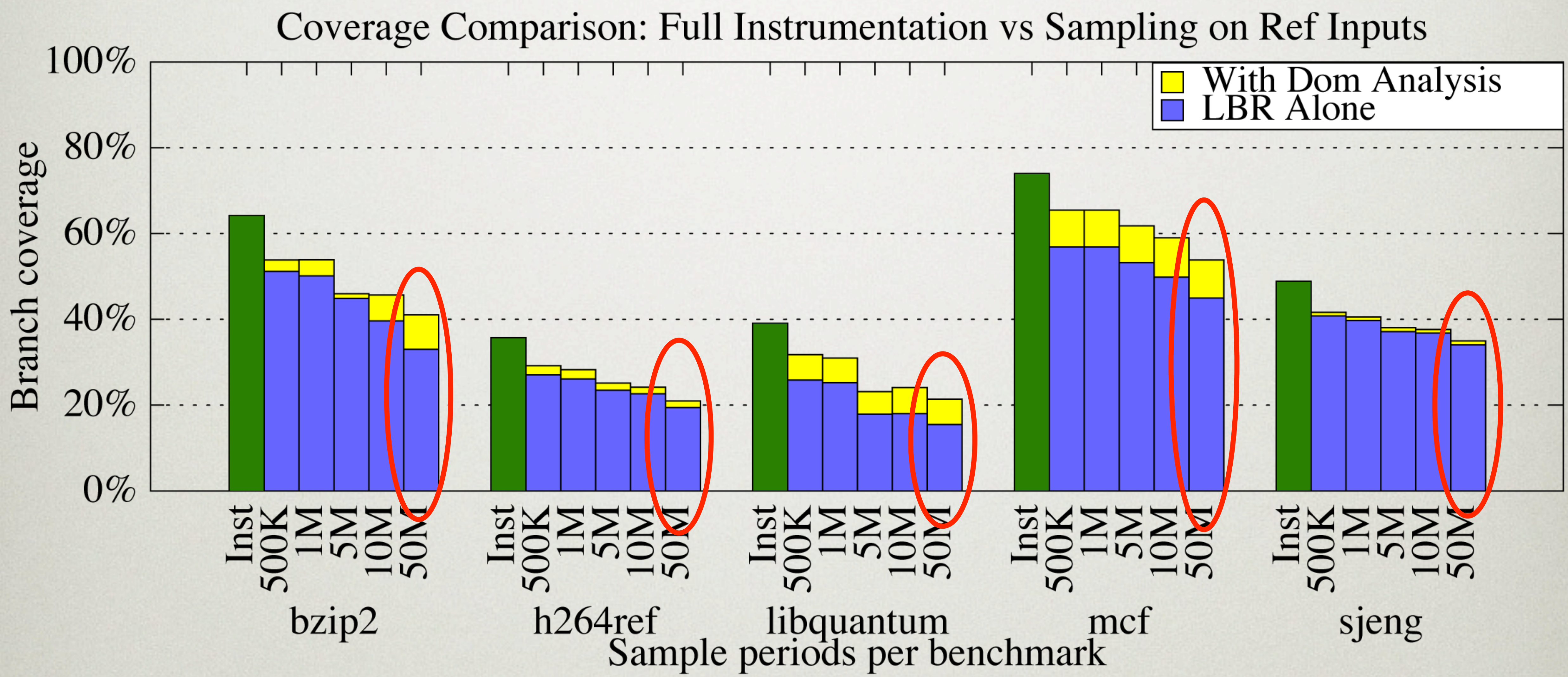
Results: Testing on a Single Core - Efficiency



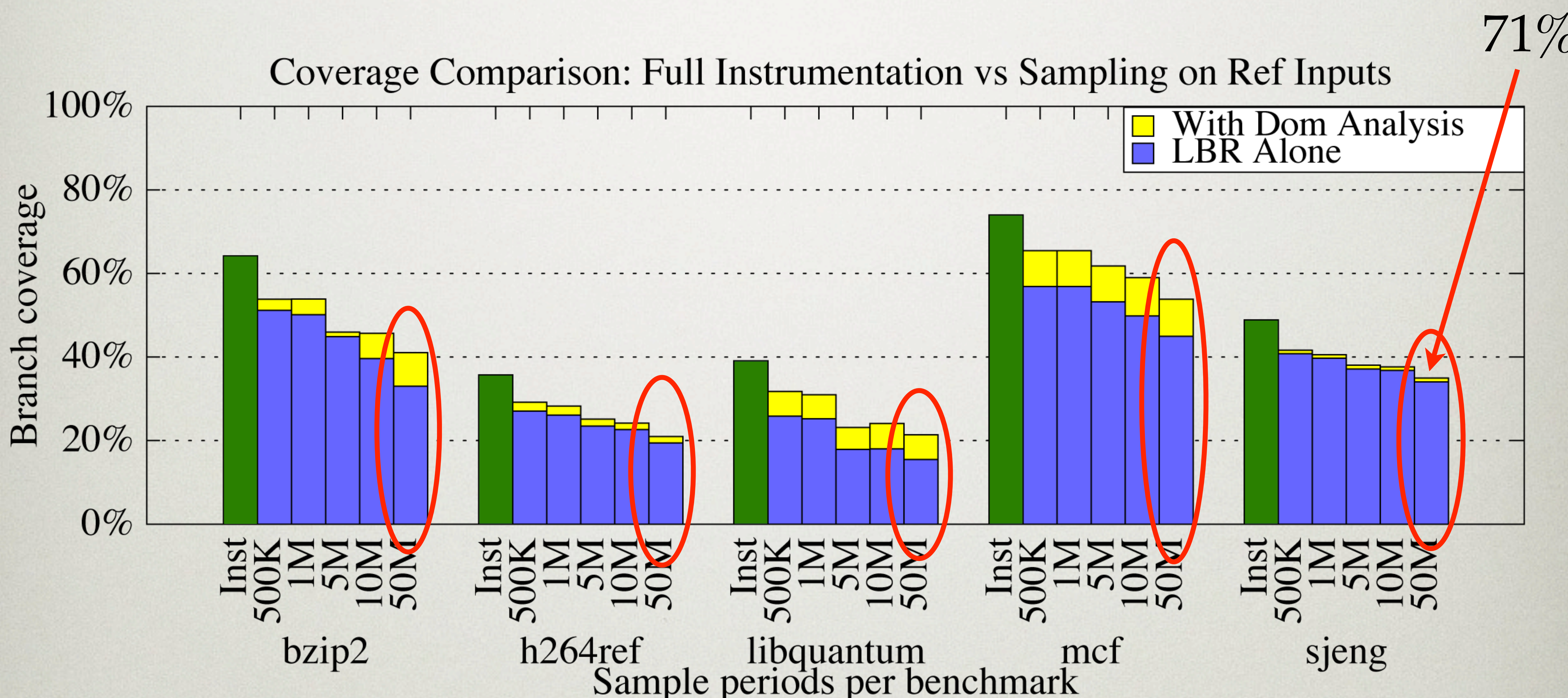
Results: Better Coverage at High Sample Rates



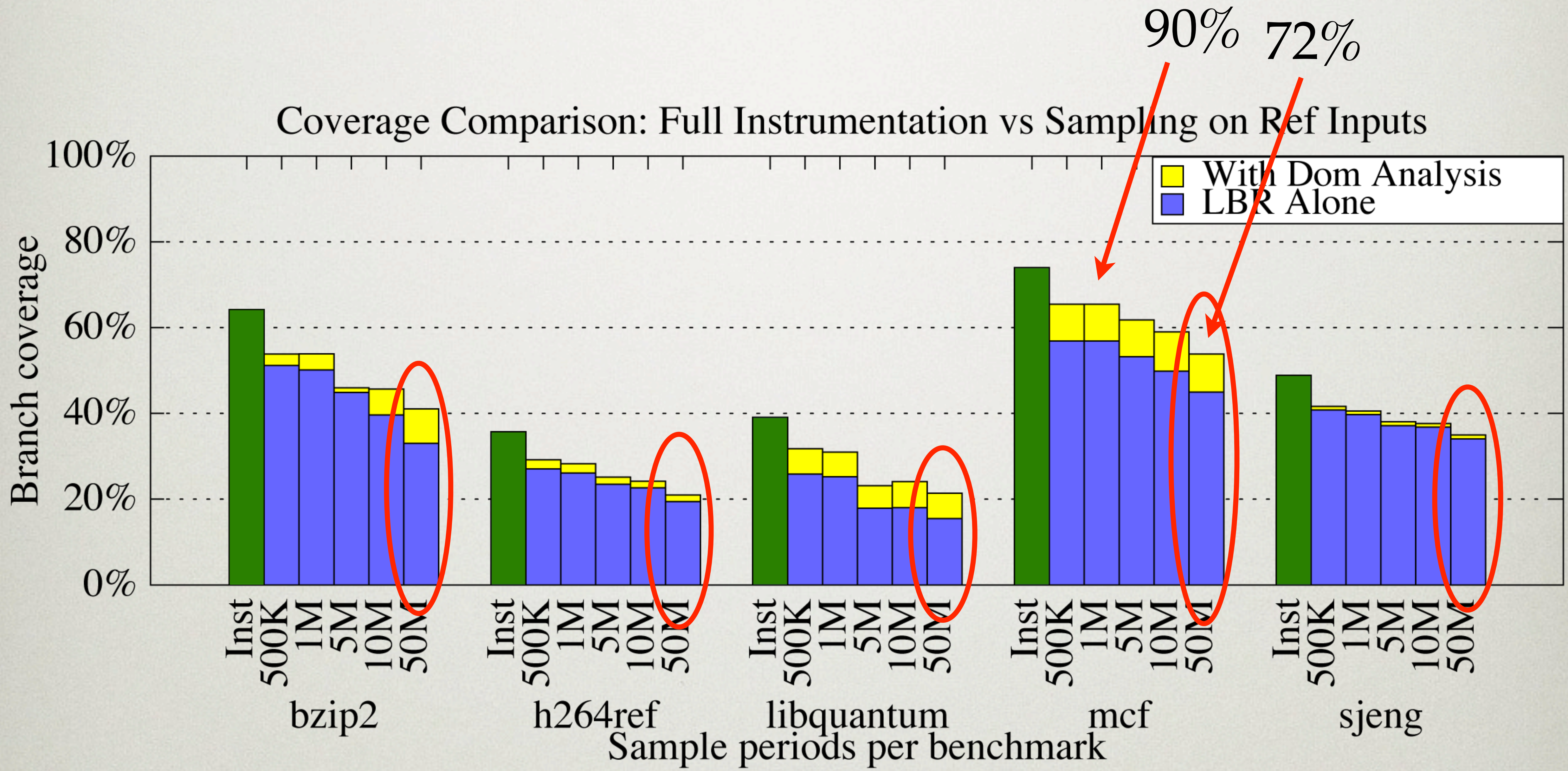
Results: Better Coverage at High Sample Rates



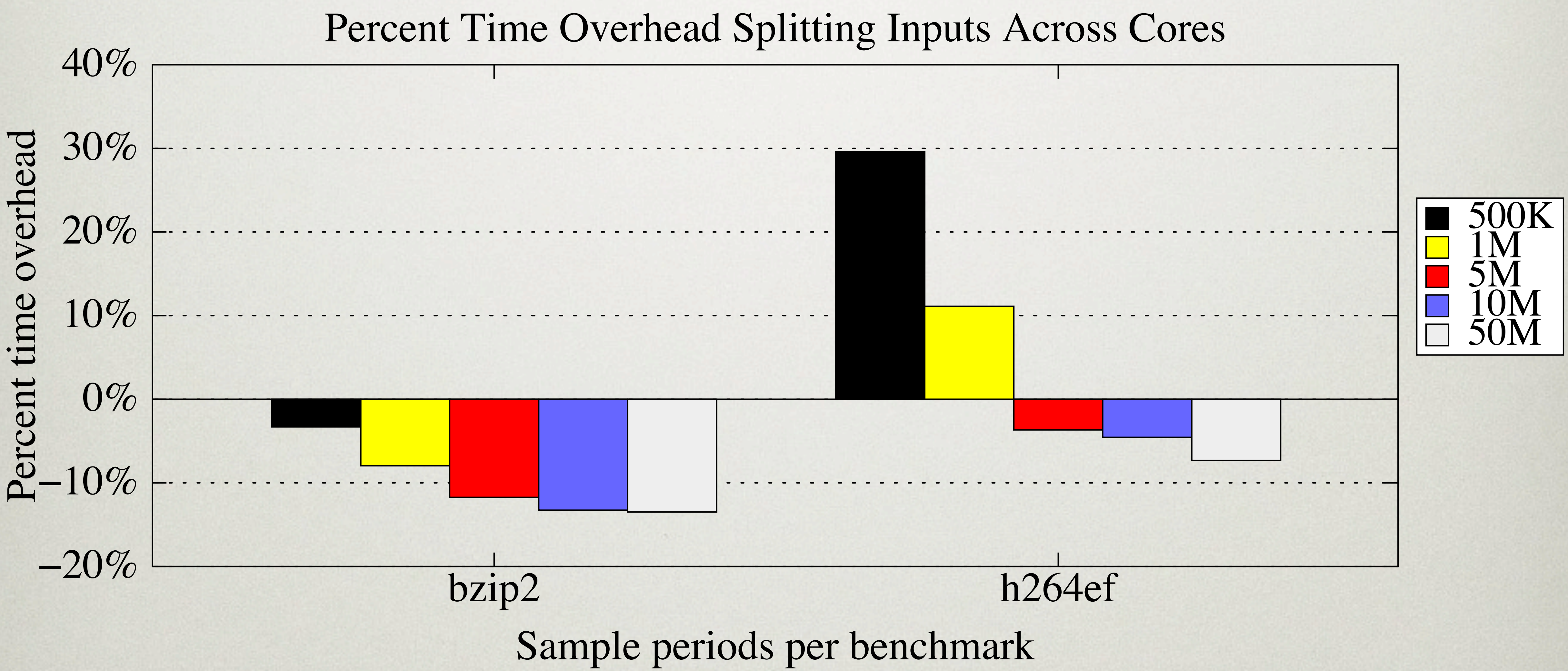
Results: Better Coverage at High Sample Rates



Results: Better Coverage at High Sample Rates



Results: Testing on a Multiple Cores - Efficiency



Hardware Monitoring Benefits

- Low overhead, effective branch testing technique
  - Up to 90% of branch coverage
  - 2% time improvement
  - 0.5% code growth (compared to 60% to 90%)
- Test coverage approximation
  - Testing on resource constrained devices
  - “Imprecise” tasks (e.g. regression test prioritization)
  - Partial program monitoring
- Significant benefits
  - Enable testing on resource constrained devices
  - Generates full picture of program execution

Conclusions and Future Work

- Extensible, portable system for single or multiple cores
- Up to 11.13% improvement in time overhead
- Up to 90% of the coverage reported by instrumentation
  - Reduced time overhead (~2%)
- Negligible code growth
- Future work:
  - Combine hardware monitoring with limited instrumentation
  - Implement on resource constrained device
  - Extend system to other coverage metrics

Thank You!

Website:

http://www.cs.virginia.edu/walcott

Questions?