# Dynamic Performance Tuning for Speculative Threads

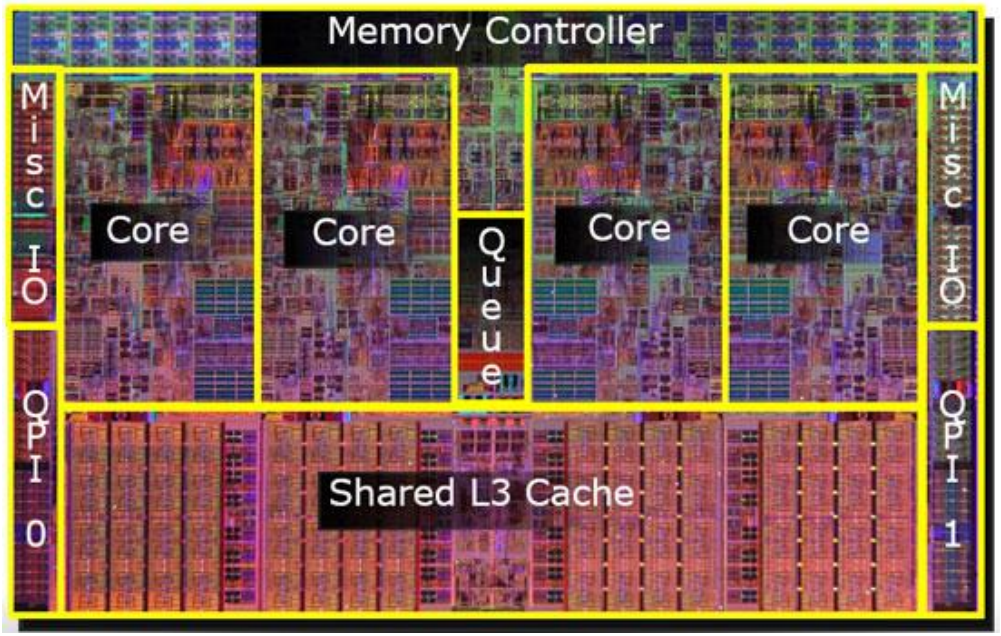
**Yangchun Luo**, Venkatesan Packirisamy,
Nikhil Mungre†, Ankit Tarkas†,
Wei-Chung Hsu, and Antonia Zhai

Dept. of Computer Science and Engineering
†Dept. of Electronic Computer Engineering
University of Minnesota – Twin Cities

UNIVERSITY OF MINNESOTA

# Motivation

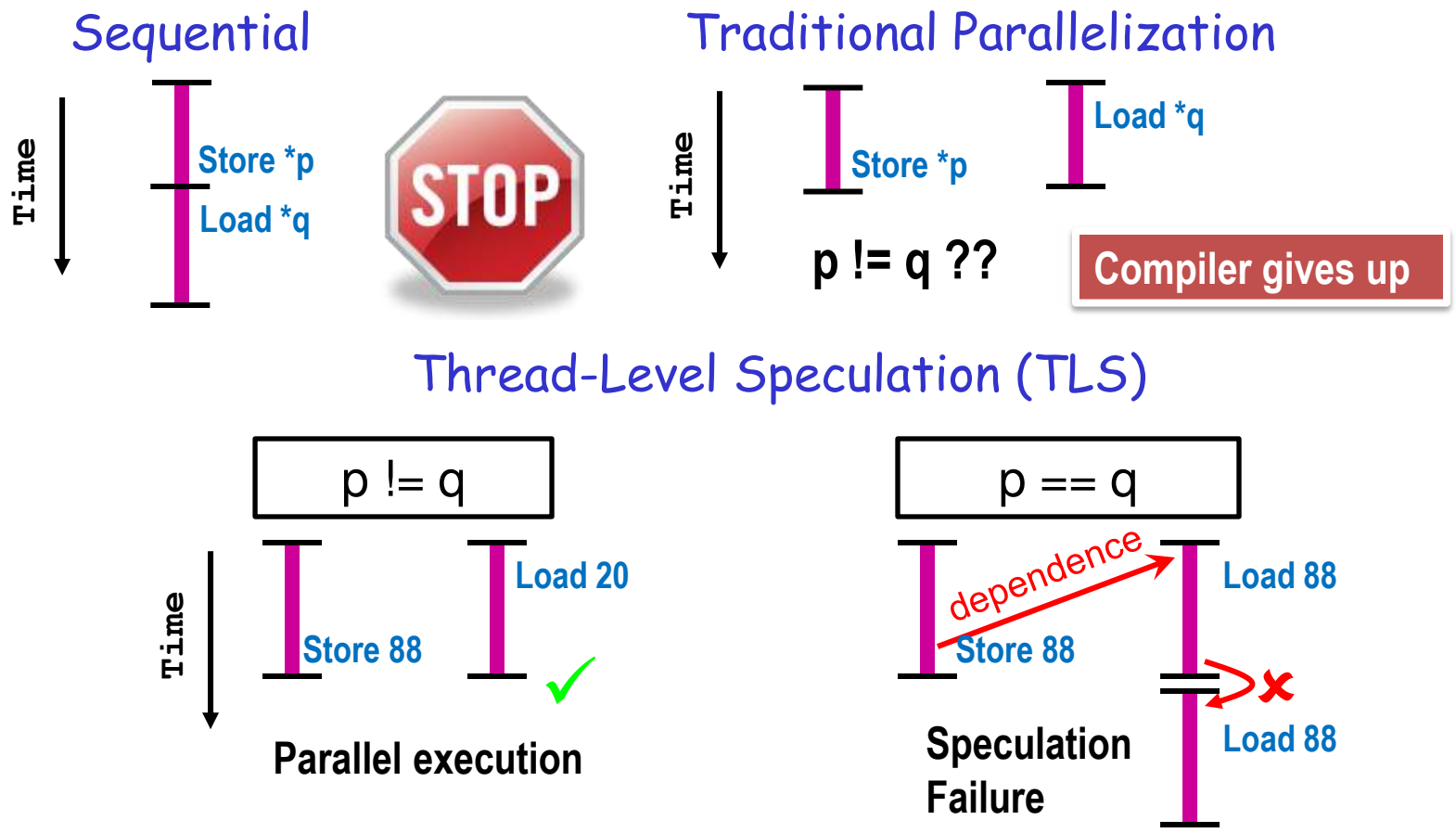
Multicore processors brought forth large potentials of computing power




Intel Core i7 die photo

**Exploiting these potentials demands thread-level parallelism**

# Exploiting Thread-Level Parallelism



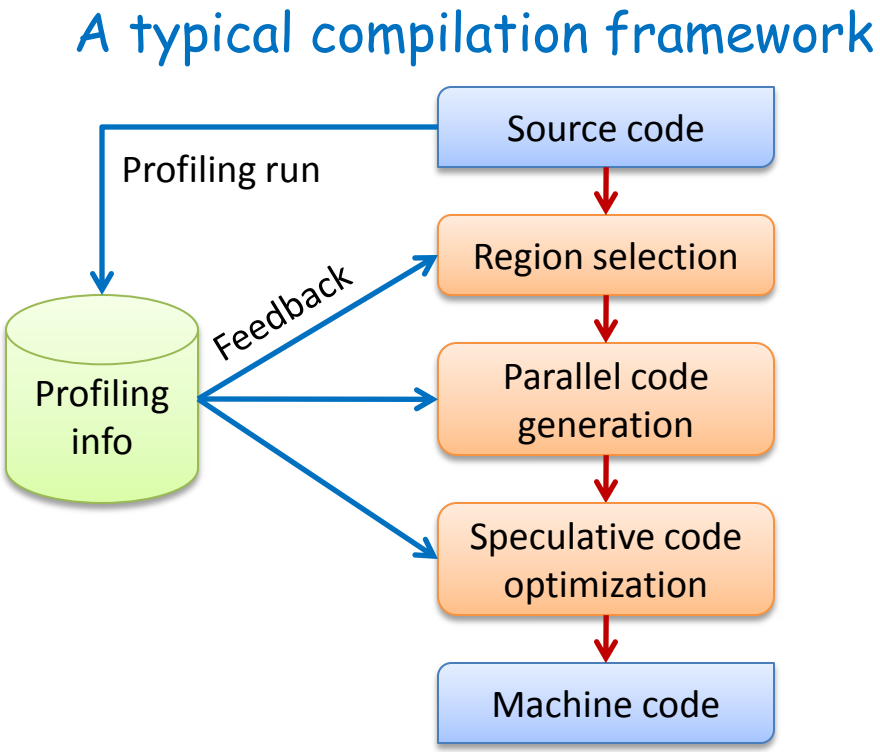

Potentially more parallelism with speculation

**But Unpredictable**

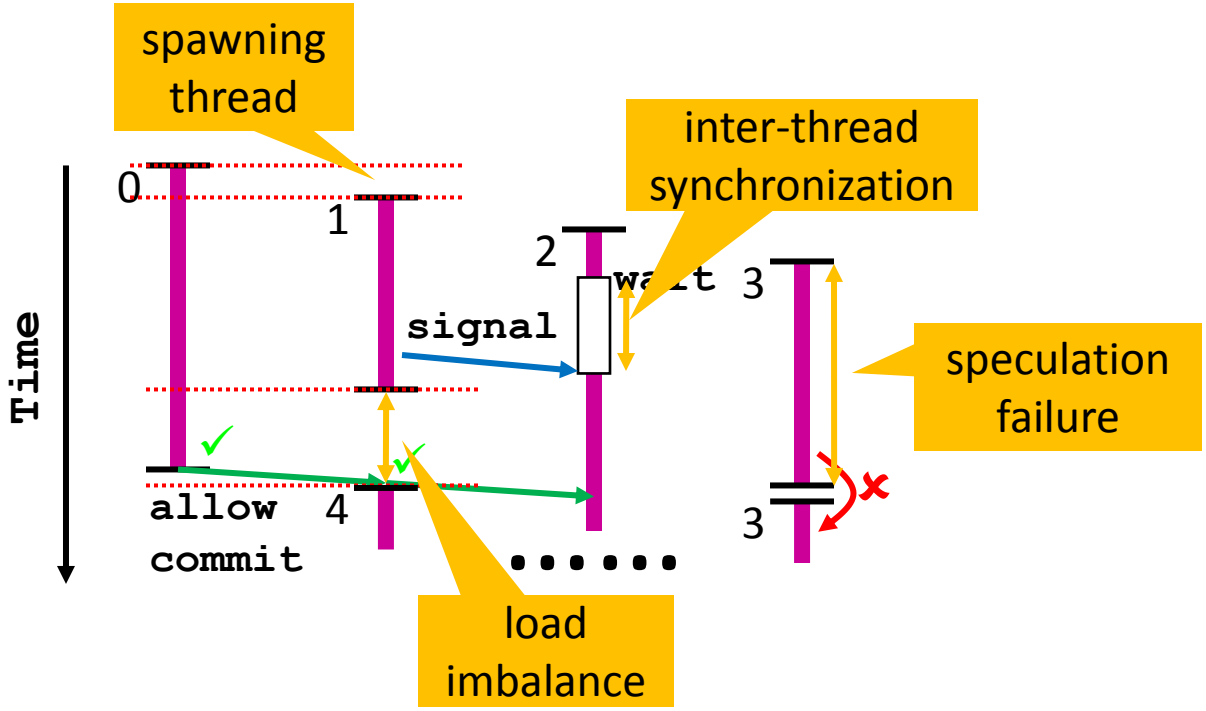
# Addressing Unpredictability

State-of-the-art approach:

- ❑ Profiling-directed optimization
- ❑ Compiler analysis

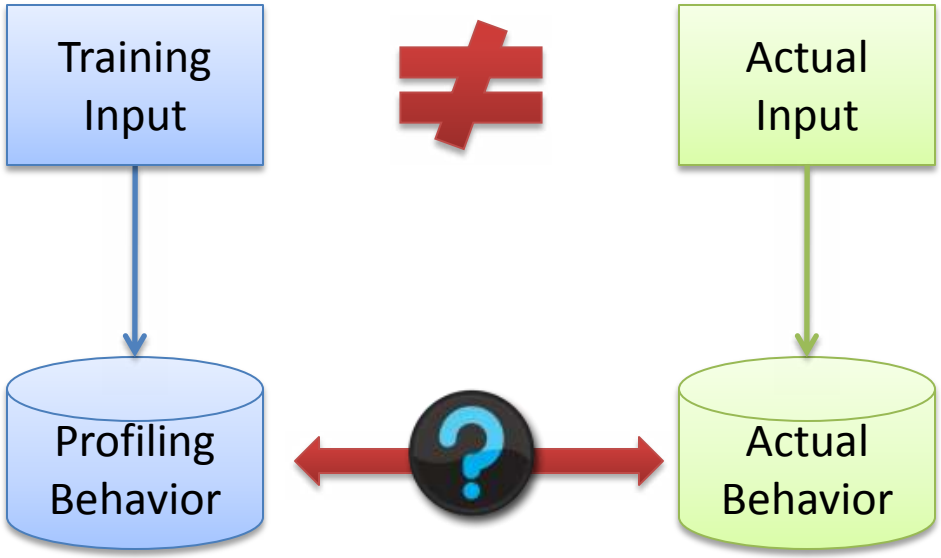


# Problem #1



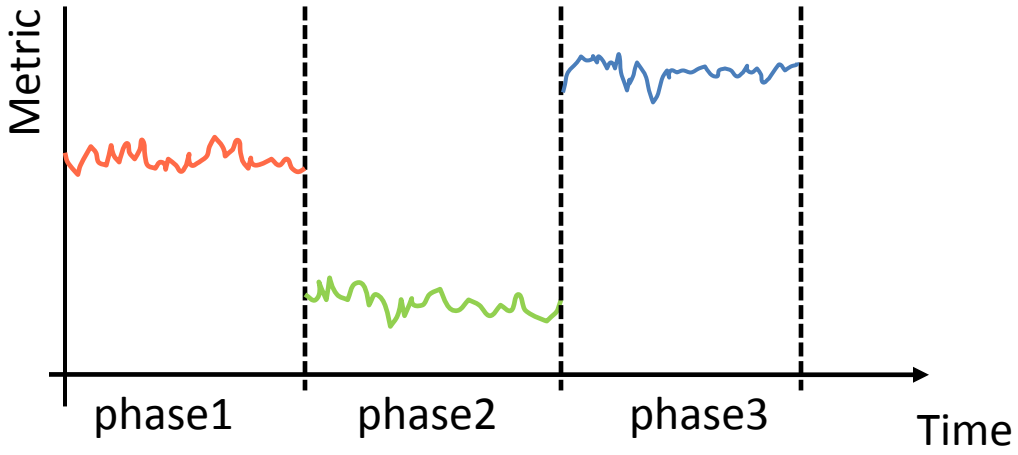

Hard to model TLS overhead statically, even with profiling

# Problem #2




Profile-based decision may not be appropriate for actual input

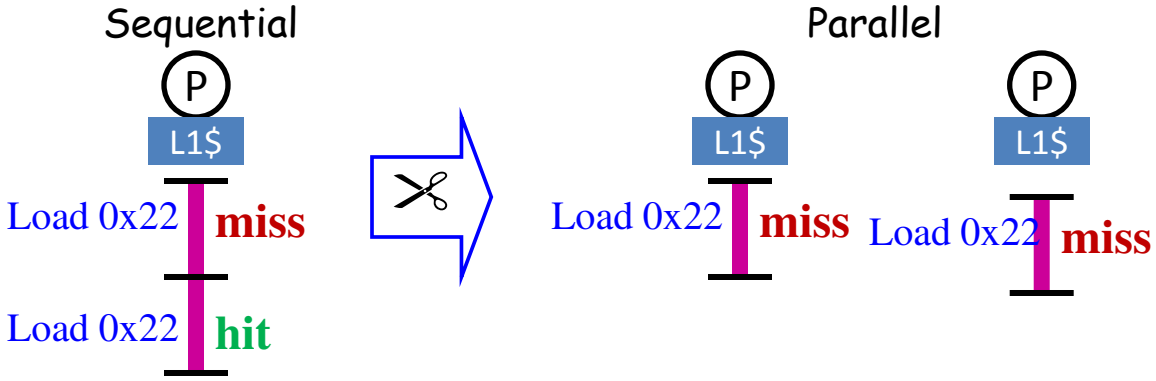
# Problem #3



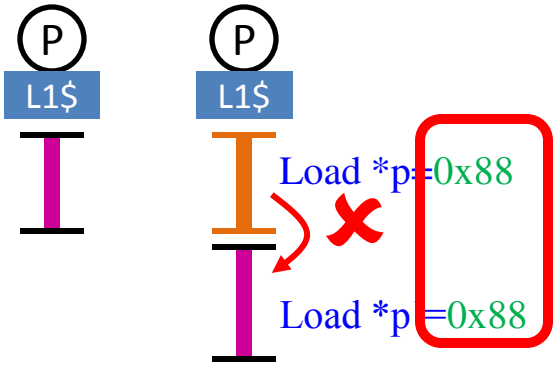

Behavior of the same code changes over time

# Problem #4

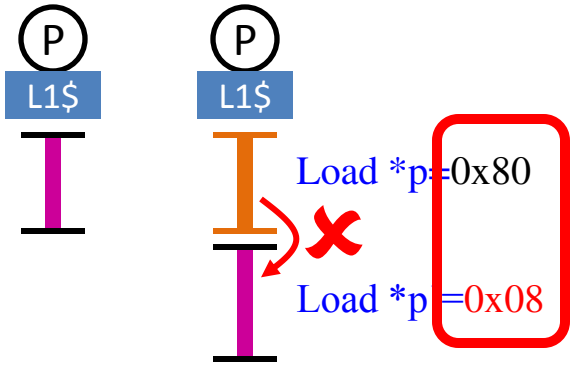
## Extra misses




## Prefetching



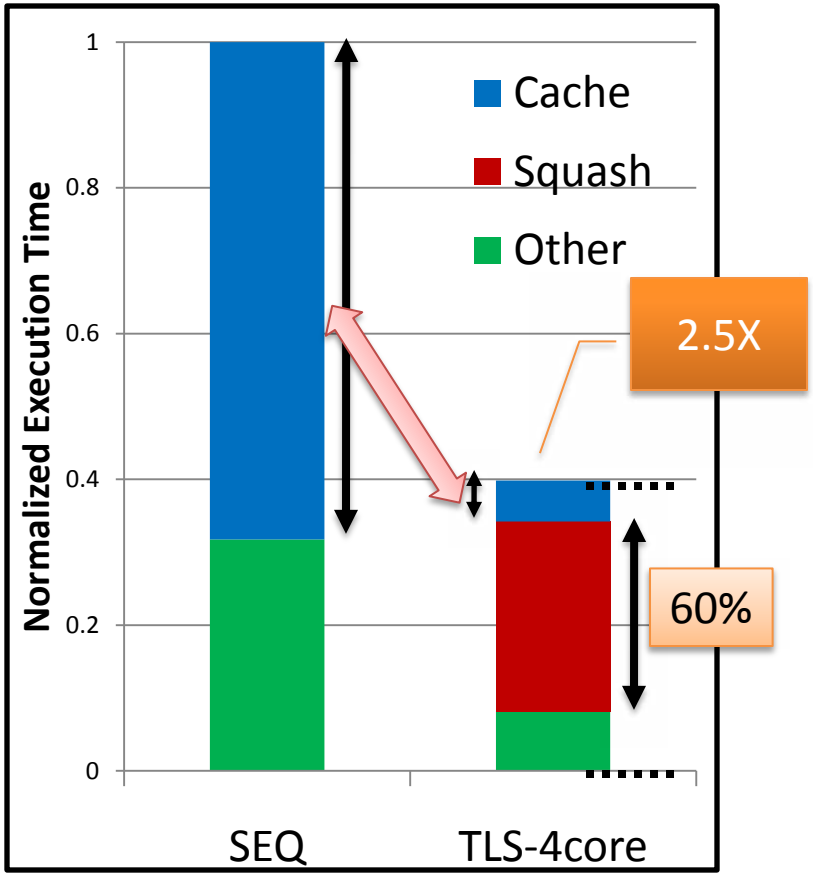

## Pollution

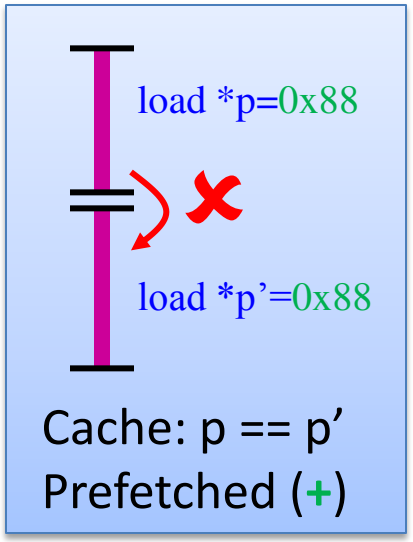



How important are cache impacts?

# Impact on Cache Performance

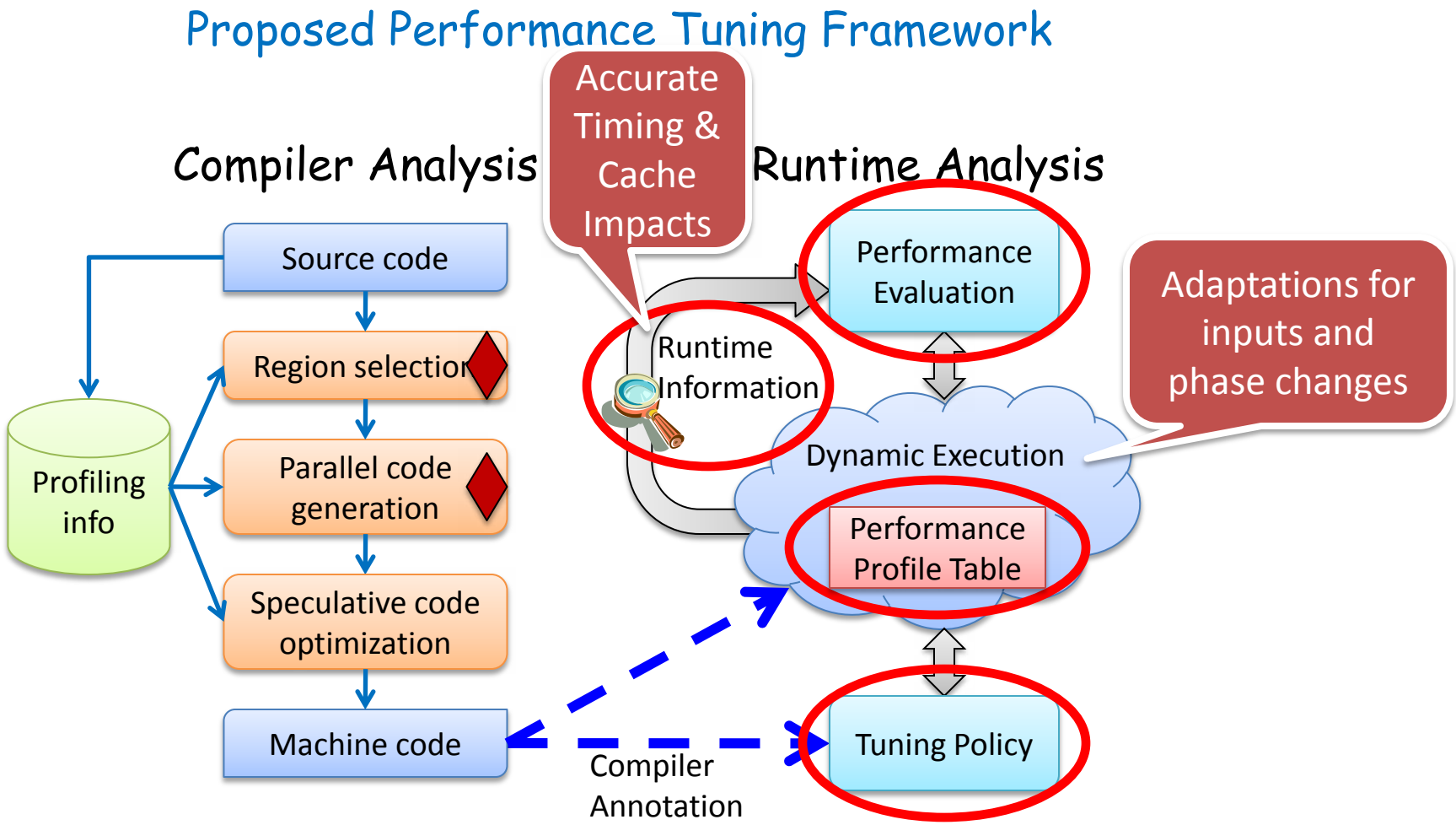
A major loop from benchmark art (scanner.c:594)



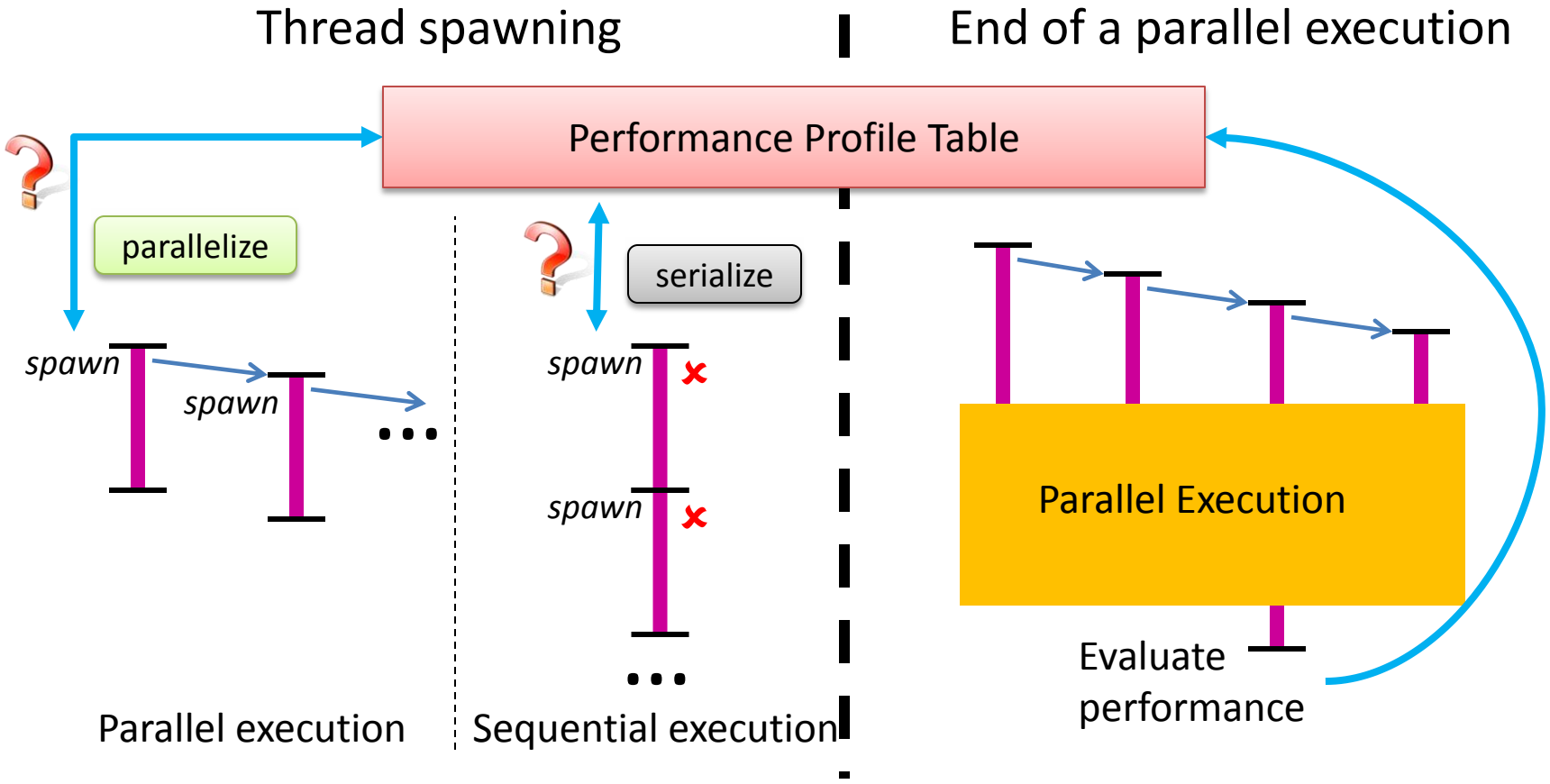




Cache impact is difficult to model statically

# Our proposal

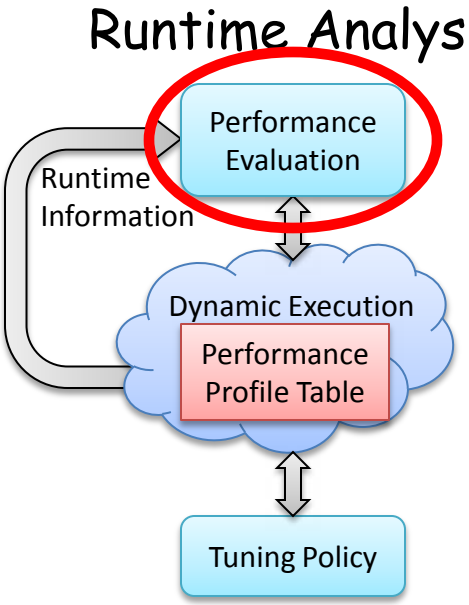


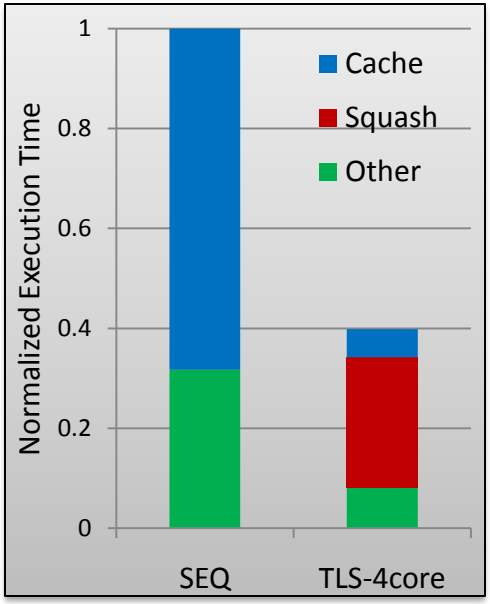
# Parallelization Decisions at Runtime




New execution model facilitates dynamic tuning

# Evaluating Performance Impact



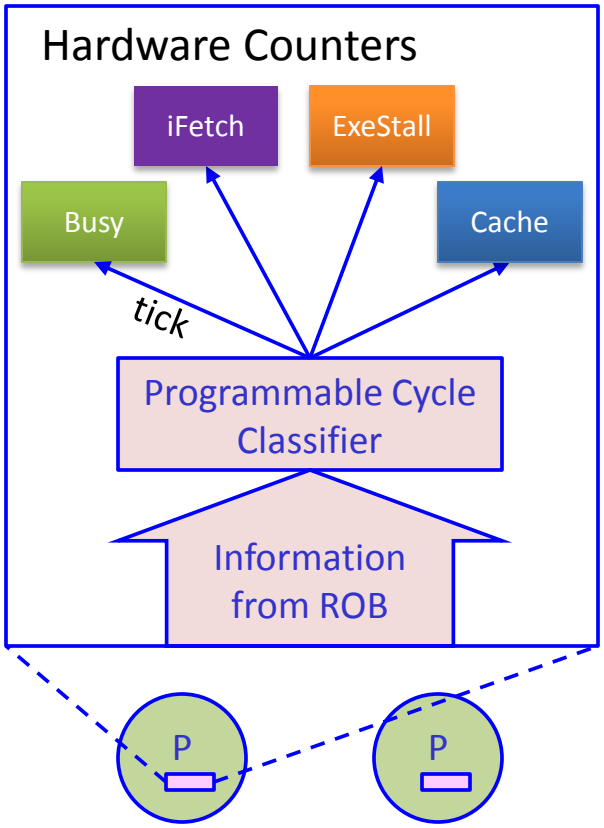

Simple metric:

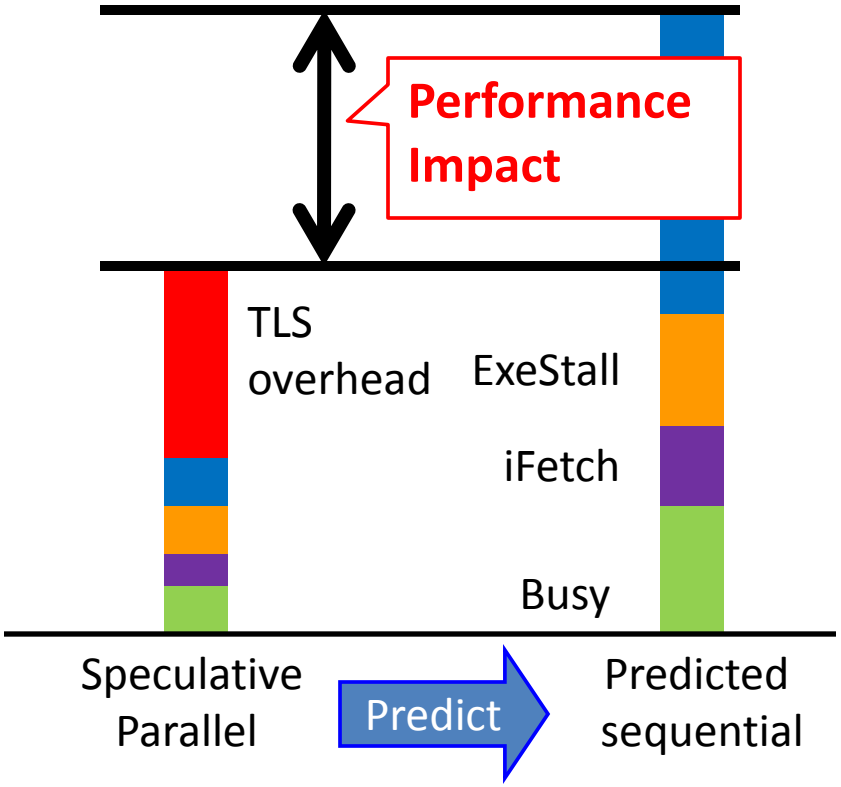
cost of speculative failure

Comprehensive evaluation:

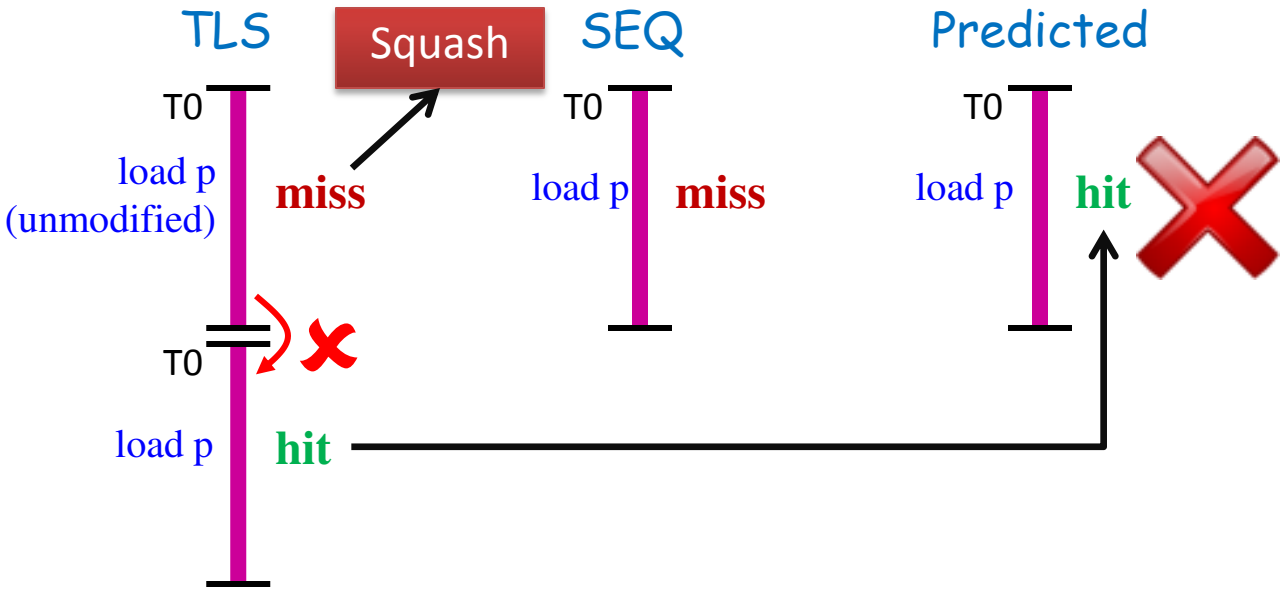
sequential performance prediction

# Sequential Performance Prediction



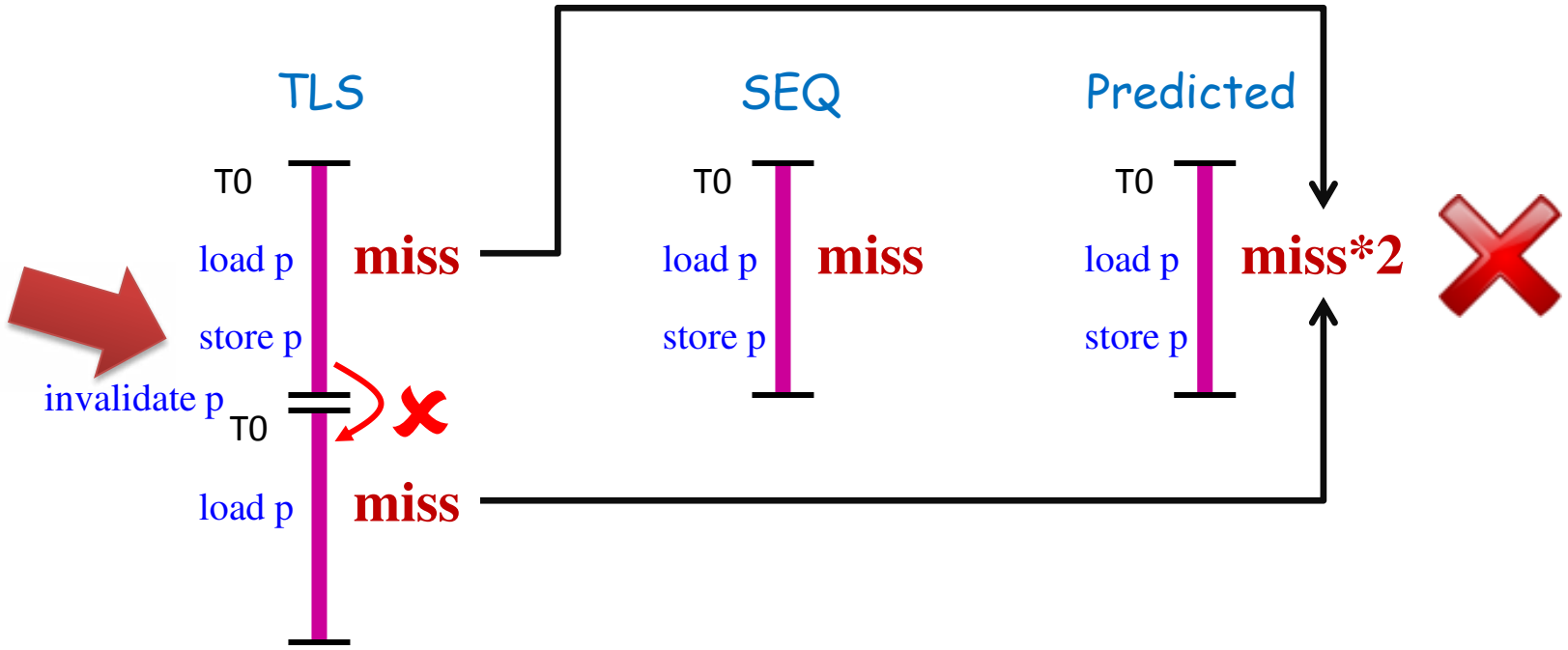



# Cache Behavior: Scenario #1




**Count** the miss in squashed execution

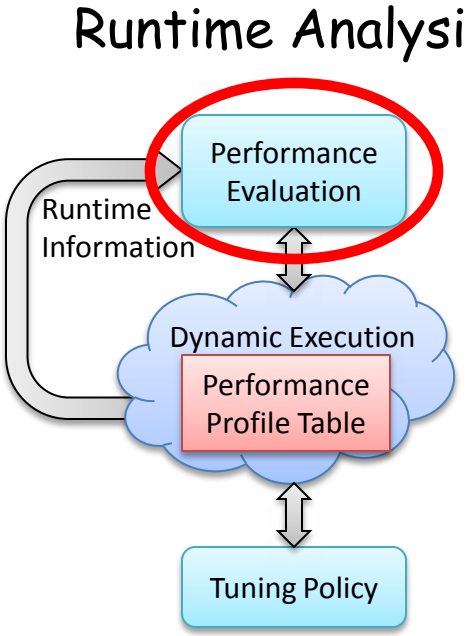
# Cache Behavior: Scenario #2




**Not count** a miss if

- tag matched
- invalid status

# Tune the performance




Exhaustive search

Prioritized search

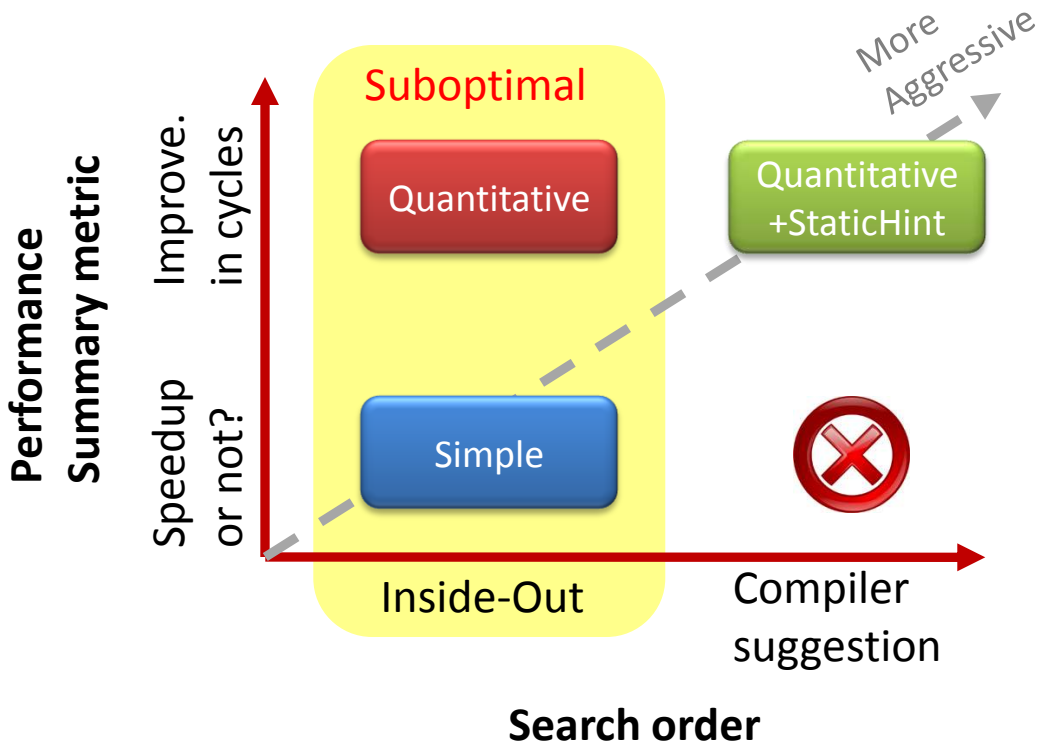
# How to prioritize the search

Search order:

- ❑ Inside-Out
- ❑ Compiler suggestion
- ❑ Outside-In

Performance summary metric:

- ❑ Speedup or not?
- ❑ Improvement in cycles



# Evaluation Infrastructure

## Benchmarks

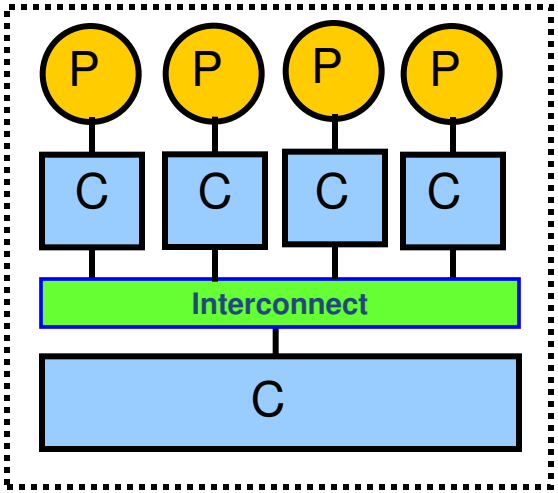
- SPEC2000 written in C, -O3 optimization

## Underlying architecture

- 4-core, chip-multiprocessor (CMP)
- speculation supported by coherence

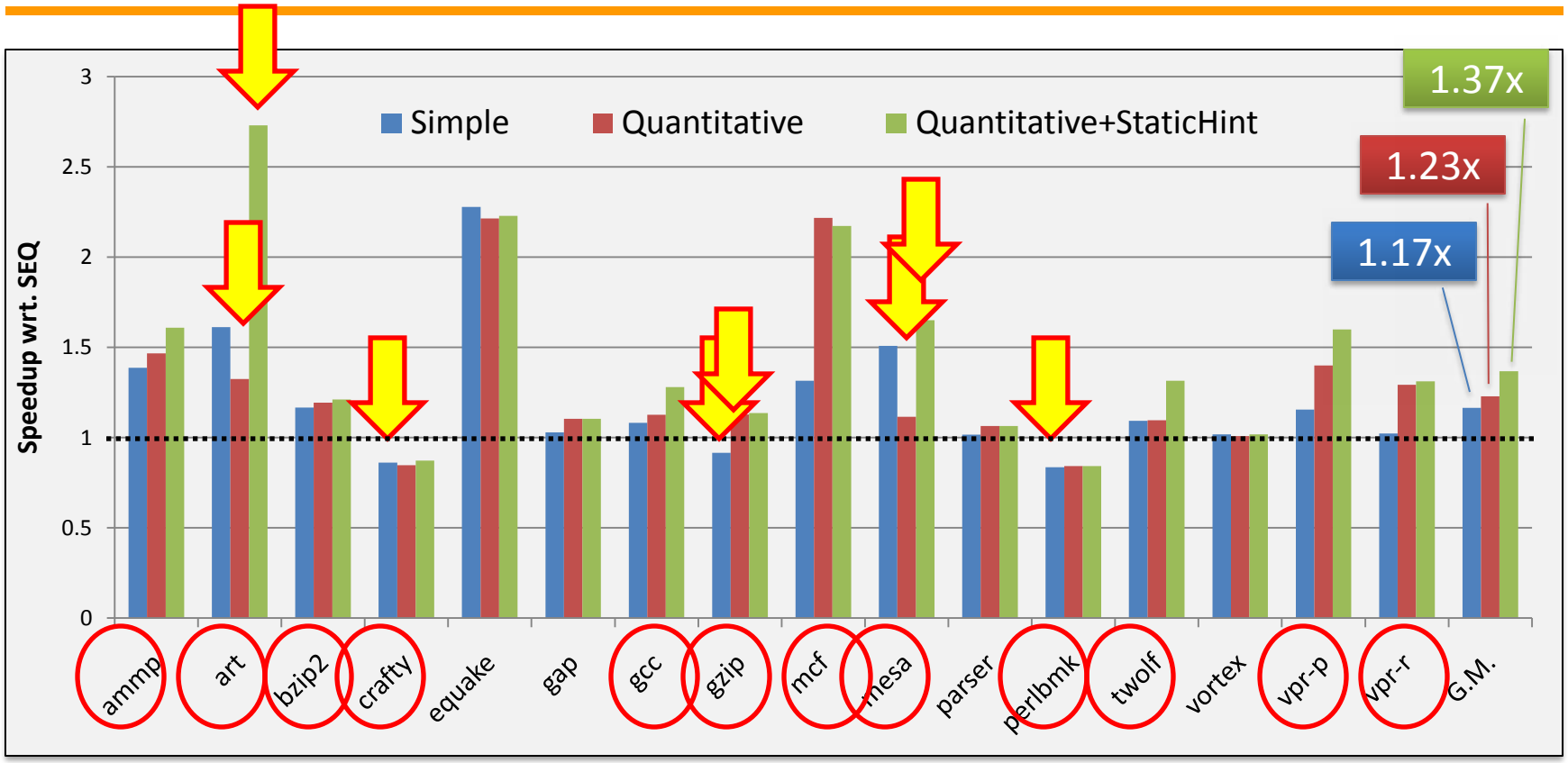
## Simulator

- Superscalar with detailed memory model
- simulates communication latency
- models bandwidth and contention



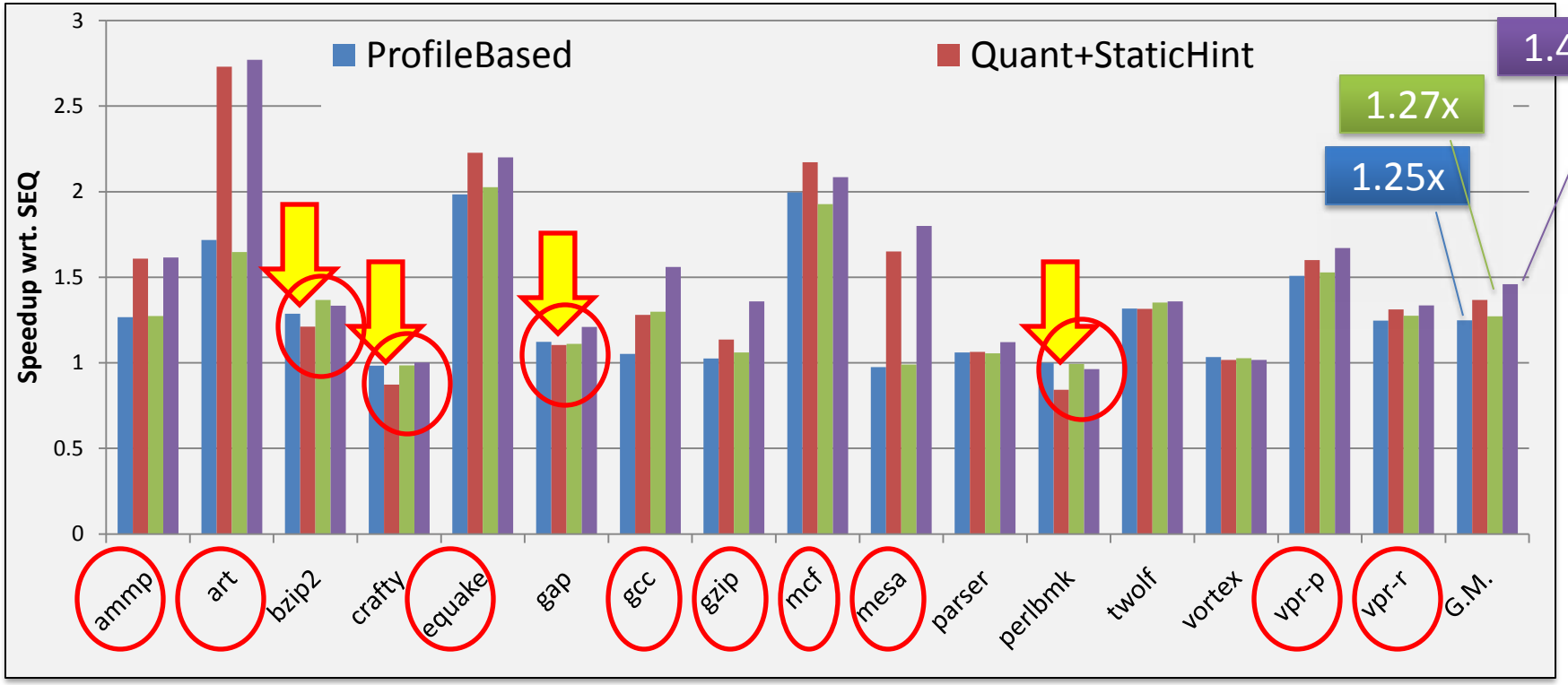

➢ **Detailed, cycle-accurate simulation**

# Comparing Tuning Policies




Parallel Code Overhead

# Profile-Based vs. Dynamic




Dynamic outperformed static by ≈10% (1.37x/1.25x)

Potentially go up to ≈15% (1.46x/1.27x)

# Related works: Region Selection

| Statically | Dynamically |
| --- | --- |
| **Compiler Heuristics** [Vijaykumar *Micro*'98] | **Runtime Loop Detection** [Tubella *HPCA*'98] [Marcuello *ICS*'98] |
| **Balanced Min-Cut** [Johnson *PLDI*'04] | |
| **Simple Profiling Pass** [Liu *PPoPP*'06] (POSH) | **Saving Power from Useless Re-spawning** [Renau *ICS*'05] |
| **Extensive Profiling** [Wang *LCPC*'05] | |
| **Profile-Based Search** [Johnson *PPoPP*'07] | |

Dynamic Performance Tuning
[Luo *ISCA*'09] (this work)

# Conclusions

Deciding how to extract speculative threads at runtime

- ❑**Quantitatively** summarize performance profile
- ❑**Compiler hints** avoid suboptimal decisions

Compiler and hardware work together.

- ❑ **37%** speedup over sequential execution
- ❑ ≈**10%** speedup over profile-based decisions

**Configurable HPMs enable efficient dynamic optimizations**

Dynamic performance tuning can improve TLS efficiency