T-SPaCS – A Two-Level Single-Pass Cache Simulation Methodology

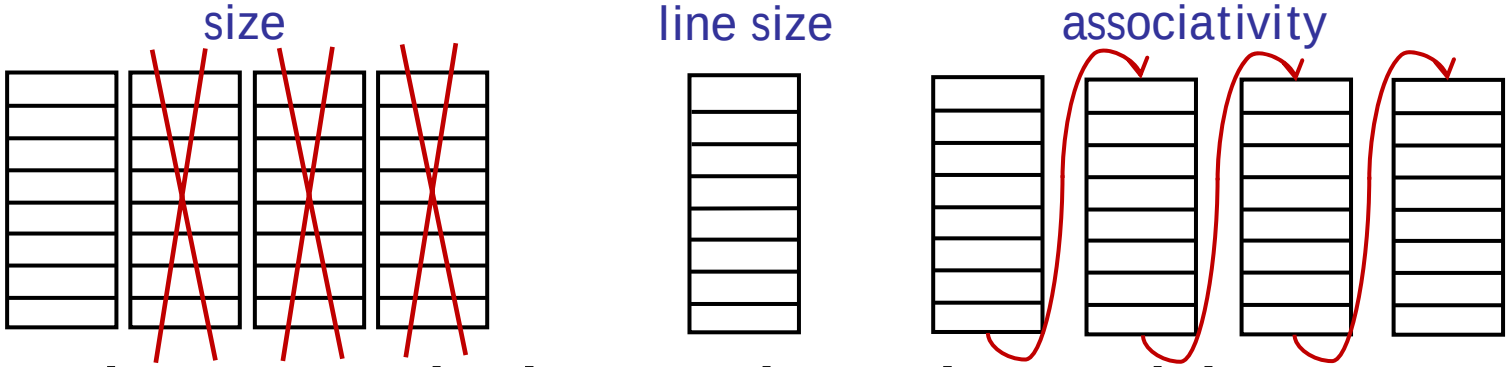
Wei Zang and Ann Gordon-Ross[+]

University of Florida
Department of Electrical and Computer Engineering

[+] Also Affiliated with NSF Center for High-Performance Reconfigurable Computing

Introduction

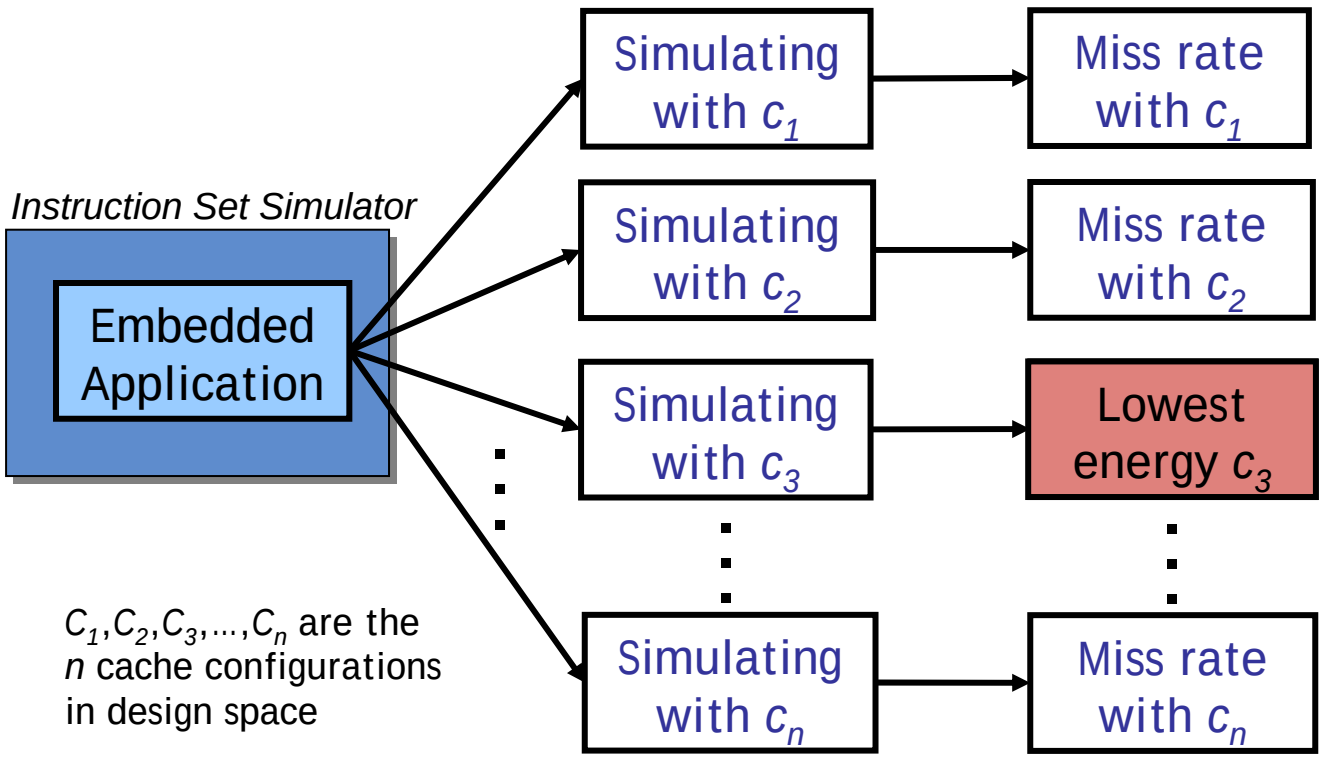
- Power hungry caches are a good candidate for optimizations
- Different applications have vastly different cache requirements
  - Configure cache parameters: size, line size, associativity



  - Cache parameters that do not match an application's behavior can waste over 60% of energy (Gordon-Ross 05)
- ***Cache tuning***
  - Determine appropriate cache parameters (***cache configuration***) to meet optimization goals (e.g., lowest energy)
  - Difficult to determine the ***best cache configuration*** given very large design spaces for highly configurable caches

Simulation-Based Cache Tuning

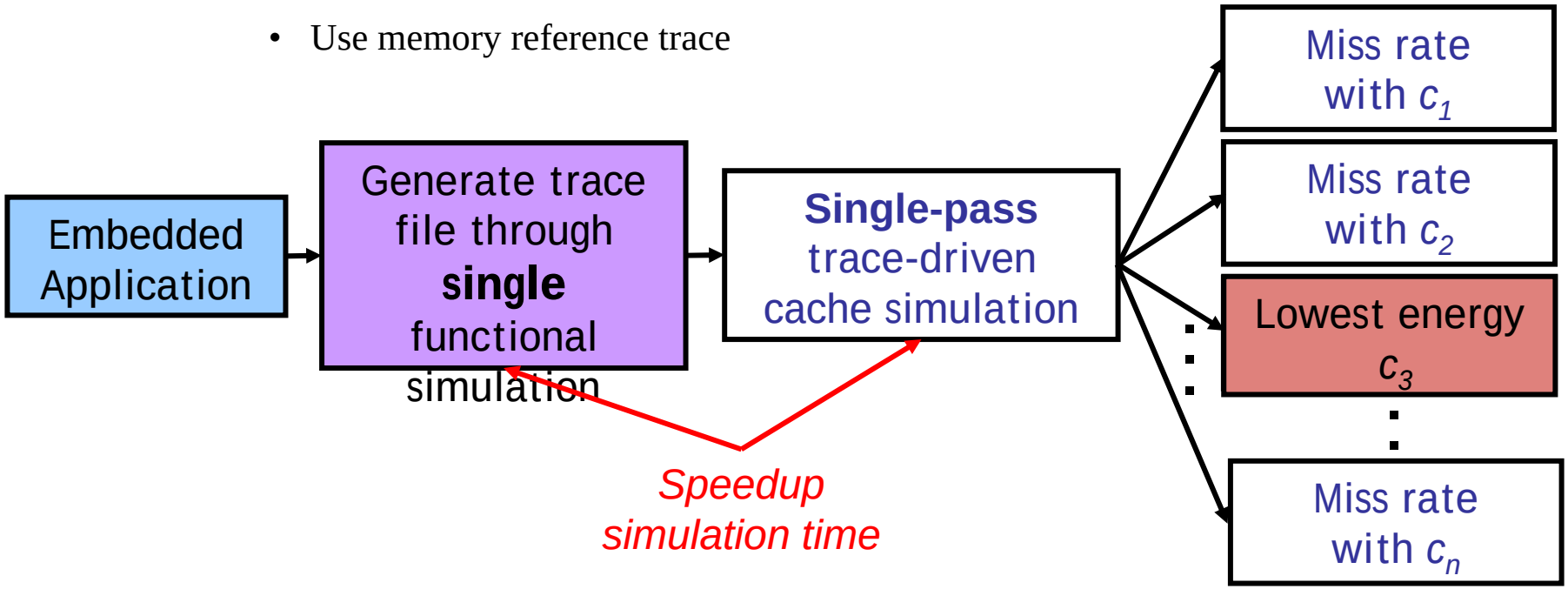
- Cache tuning at design time via simulation
  - Performed by the designer
  - Typically iterative simulation using exhaustive or heuristic methods



…very time consuming (setup and simulation time)…

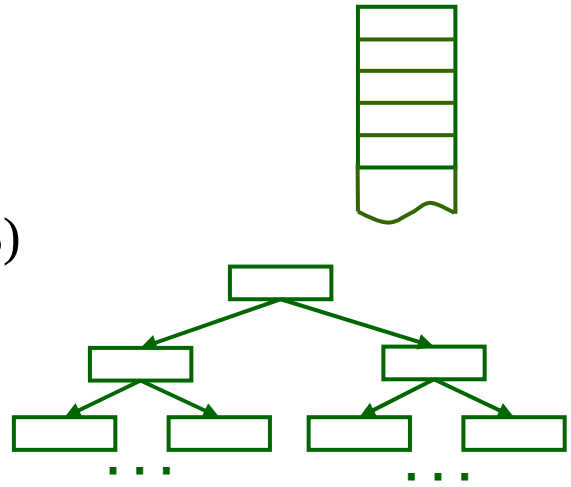
Single-Pass Cache Tuning

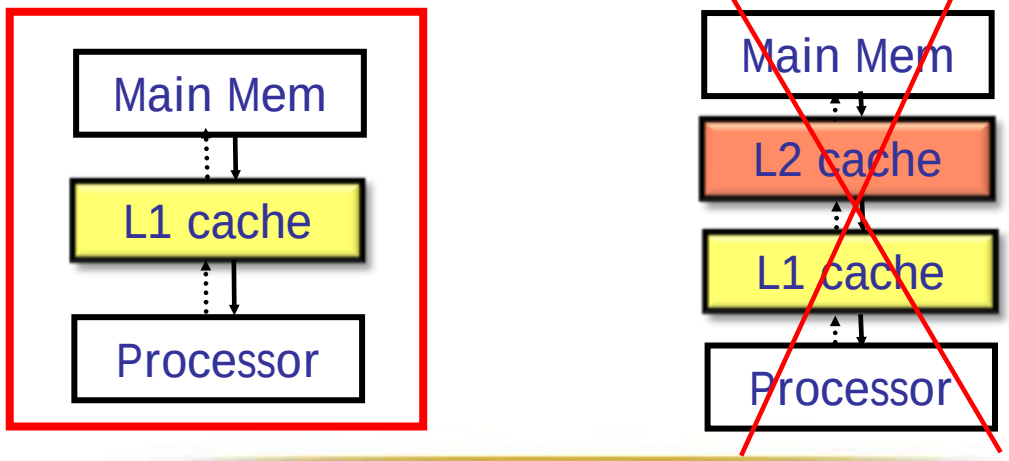
- Simultaneously evaluate multiple cache configurations during one execution
  - Trace-driven cache simulation
    - Use memory reference trace



Previous Work in Single-Pass Simulation

- Stack-based algorithm
  - Stack data structure stores access trace
  - State-of-the-art: 14X speedup over iterative (Viana 08)
- Tree data structure-based algorithm
  - Decreased simulation time
  - Complex data structures, more storage requirements
- Limitation





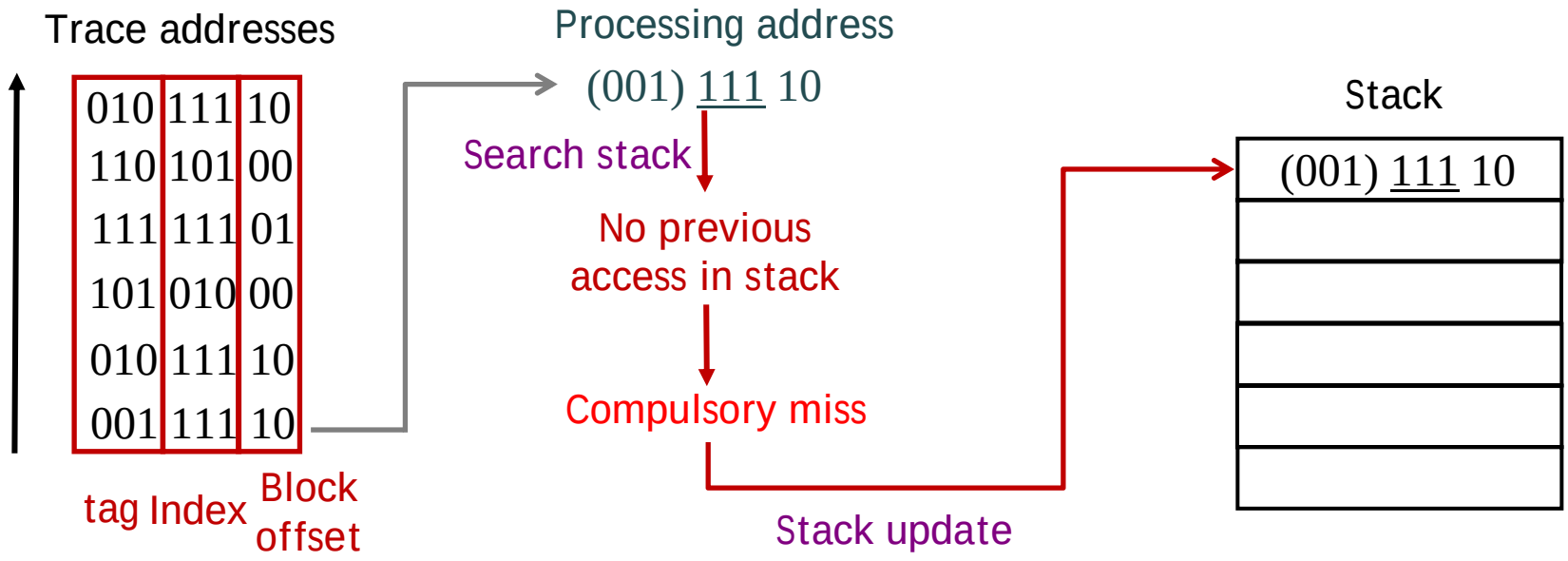
Contributions

- Two-level Single-Pass trace-driven Cache Simulation methodology – T-SPaCS
- Use a stack-based algorithm to simulate both the level one and level two caches simultaneously
- Accurately determine the optimal energy cache configuration with low storage and simulation time complexity

Single-level Cache Simulation

- Stack-based single-pass trace-driven cache simulation for single-level cache

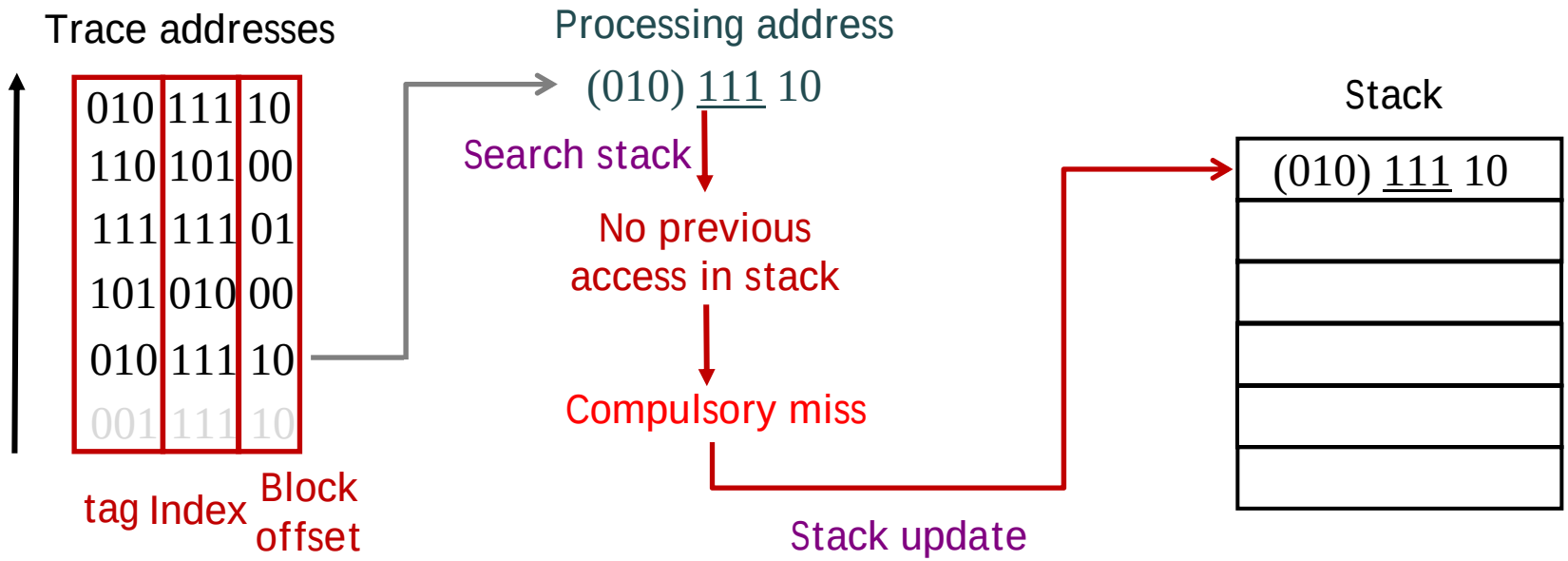
One cache configuration in design space:
block size = 4 (2^2), number of cache sets = 8 (2^3)



Single-level Cache Simulation

- Stack-based single-pass trace-driven cache simulation for single-level cache

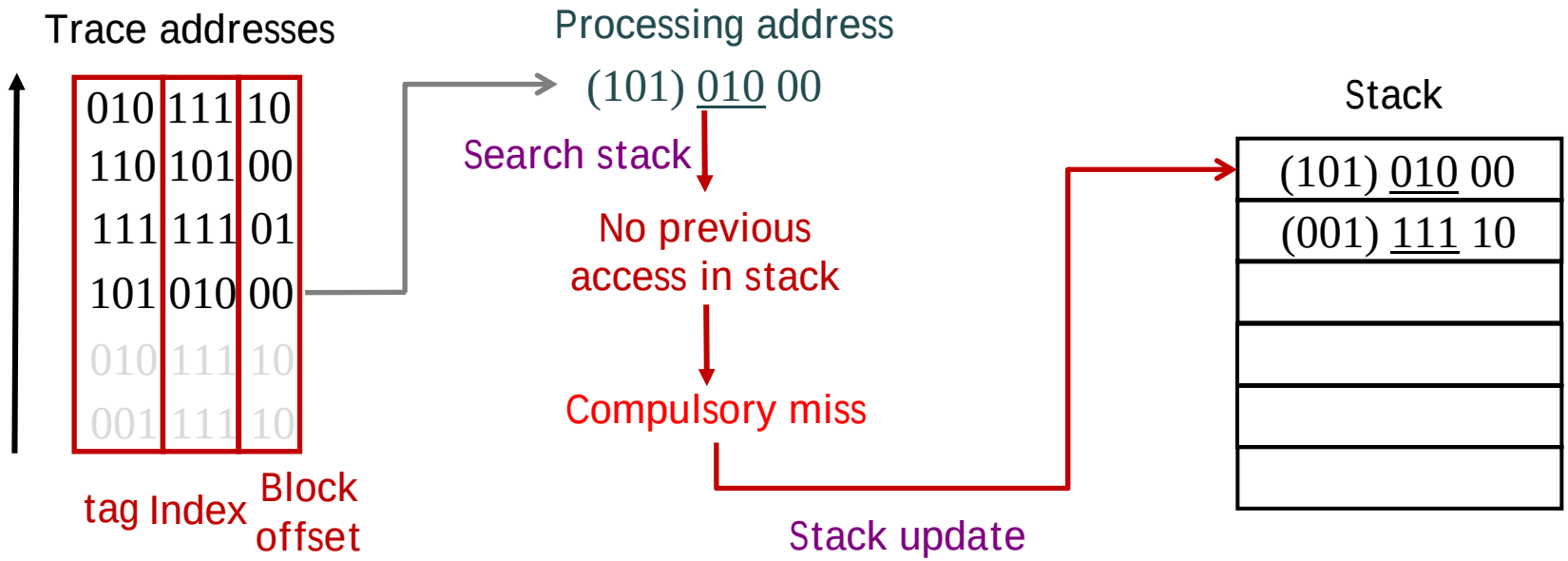
One cache configuration in design space:
block size = 4 (2^2), number of cache sets = 8 (2^3)



Single-level Cache Simulation

- Stack-based single-pass trace-driven cache simulation for single-level cache

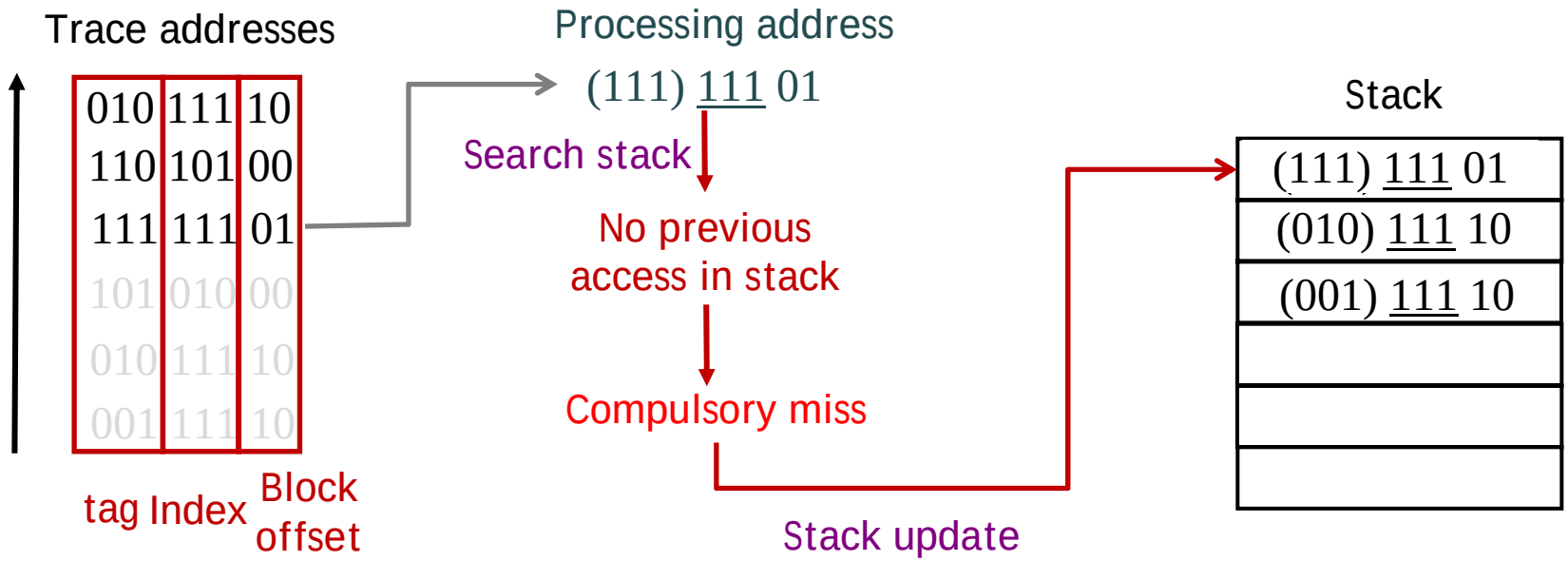
One cache configuration in design space:
block size = 4 (2^2), number of cache sets = 8 (2^3)



Single-level Cache Simulation

- Stack-based single-pass trace-driven cache simulation for single-level cache

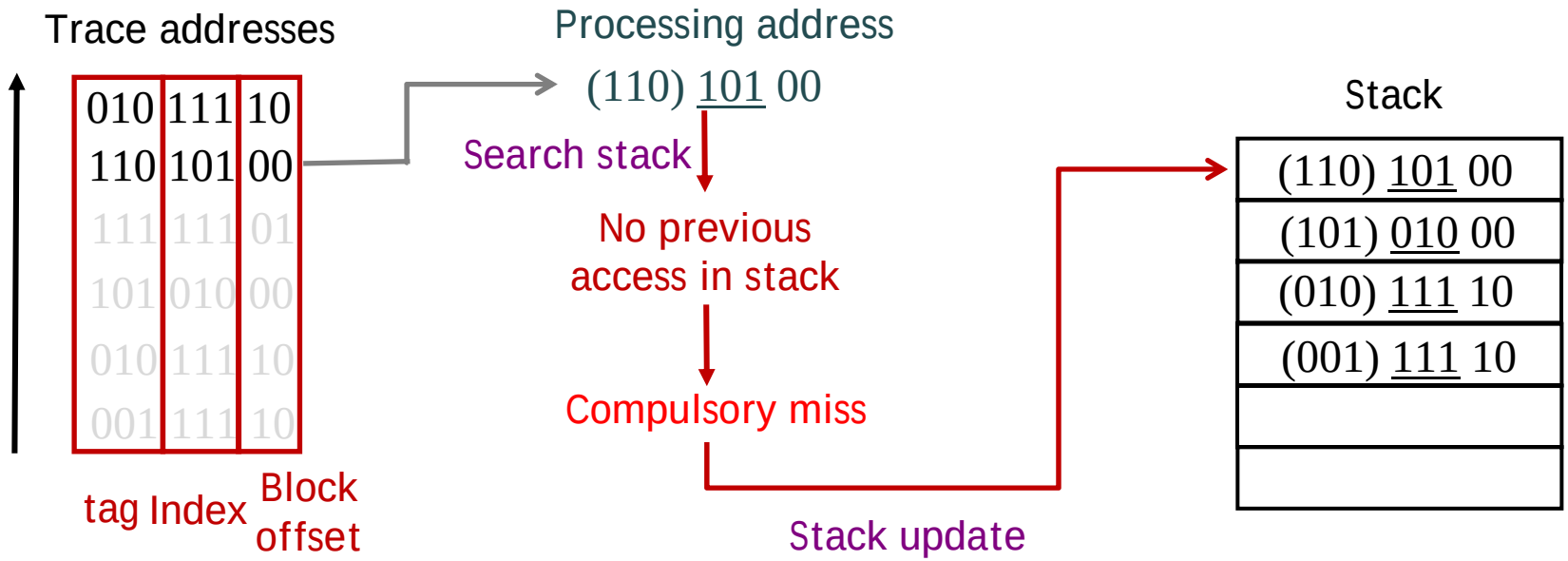
One cache configuration in design space:
block size = 4 (2^2), number of cache sets = 8 (2^3)



Single-level Cache Simulation

- Stack-based single-pass trace-driven cache simulation for single-level cache

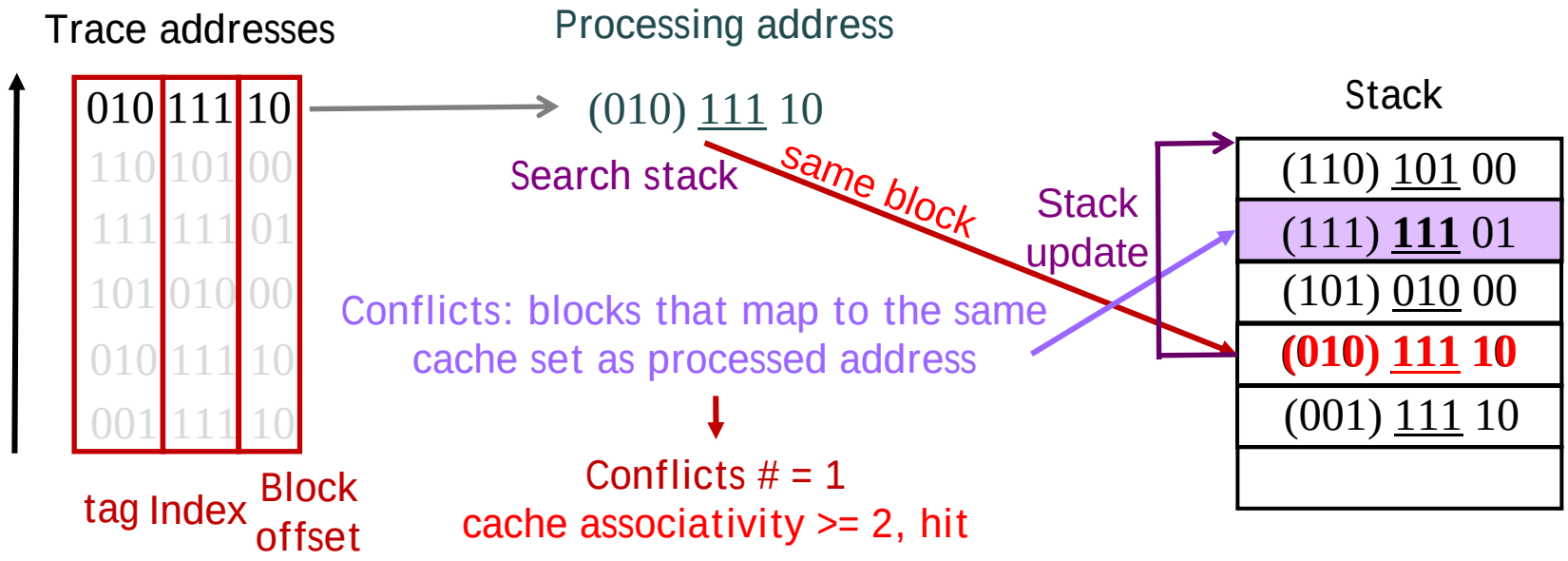
One cache configuration in design space:
block size = 4 (2^2), number of cache sets = 8 (2^3)



Single-level Cache Simulation

- Stack-based single-pass trace-driven cache simulation for single-level cache

One cache configuration in design space:
block size = 4 (2^2), number of cache sets = 8 (2^3)



Two-Level Cache Simulation

- Stack-based single-level cache simulation maintains one stack to record L1 access trace
- Naïve adaption of stack-based single-level cache simulation to two-level caches requires multiple stacks
  - Assumes inclusive cache hierarchy
  - L1 access trace: one stack based on memory reference trace
  - L2 access trace: depends on L1 miss
  - Requires *n* stacks for *n* L1 configurations
  - Disadvantage: large storage space and lengthy simulation time
- To reduce storage space and simulation time

Exclusive cache hierarchy!

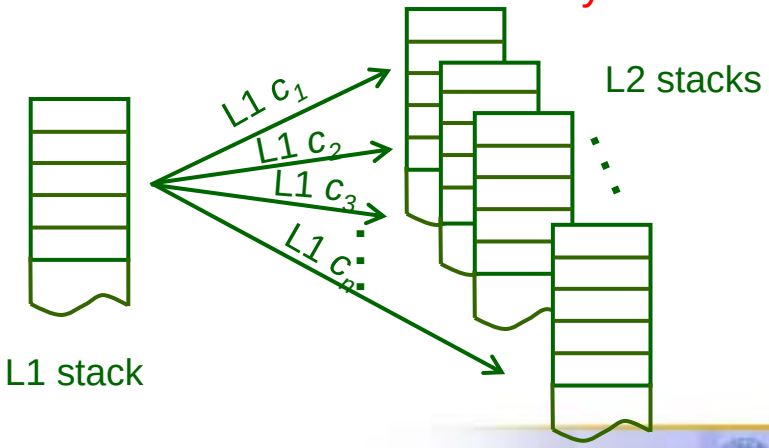
Inclusive vs. Exclusive Hierarchy

Inclusive Operation (L1/L2 LRU)

Trace	L1 (2-way)	L2 (2-way)	Hit/miss
A	A	A	L1/L2 miss
B	B A	B A	L1/L2 miss
A	A B	B A	L1 hit
C	C A	C B	L1/L2 miss
B	B C	B C	L2 hit

Seperate L1 and L2

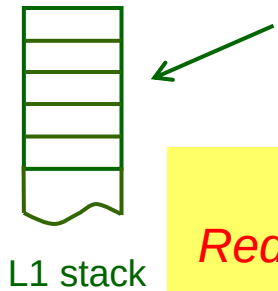
L2 access is decided by L1



Exclusive Operation (L1 LRU, L2 FIFO-like)

Trace	L1 (2-way)	L2 (2-way)	Hit/miss
A	A		L1/L2 miss
B	B A		L1/L2 miss
A	A B		L1 hit
C	C A	B	L1/L2 miss
B	B C	A	L2 hit

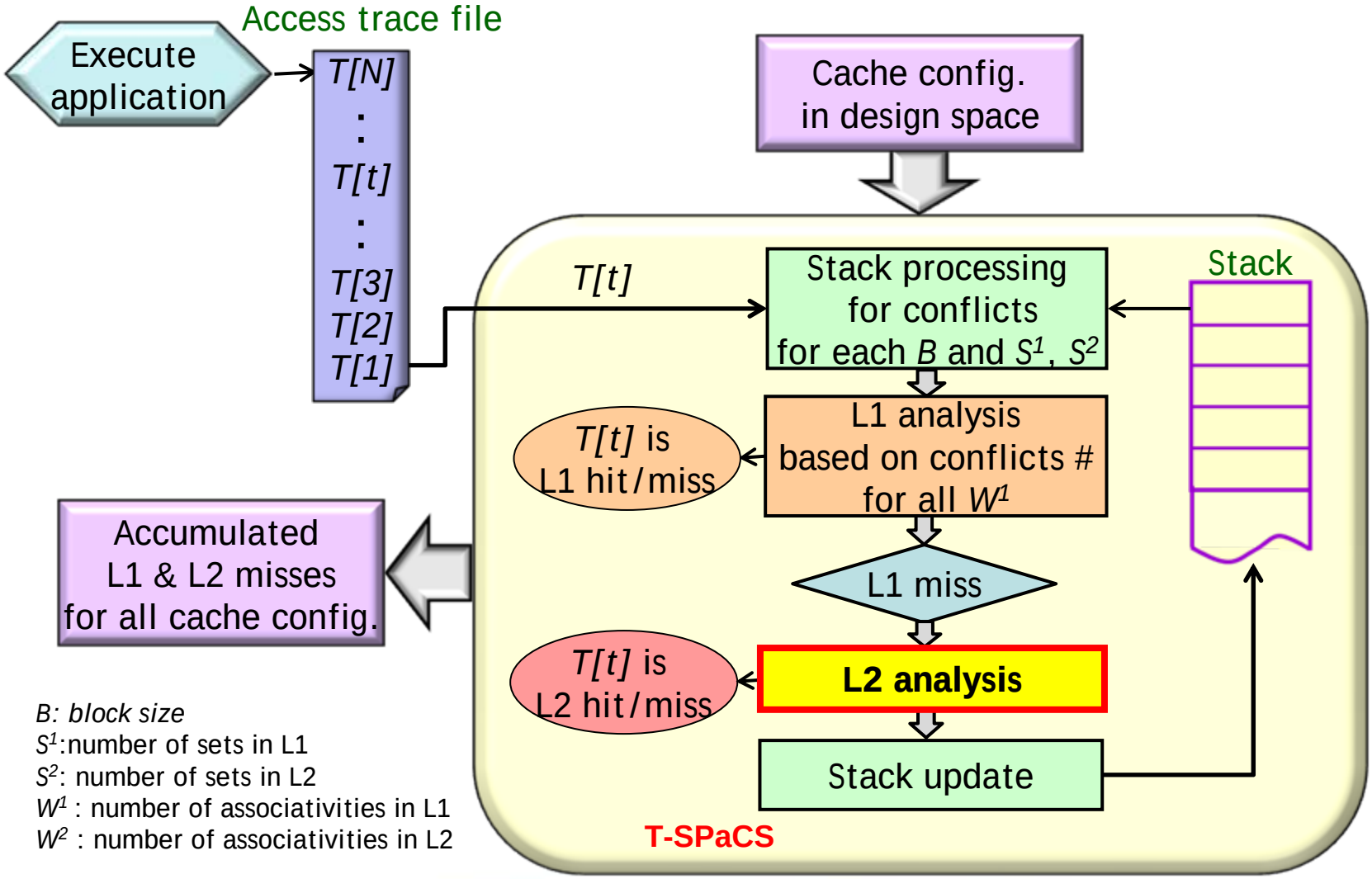
Combined cache



Simulate L1 & combined cache and derive L2 cache

One Stack!
Reduced storage space and simulation time

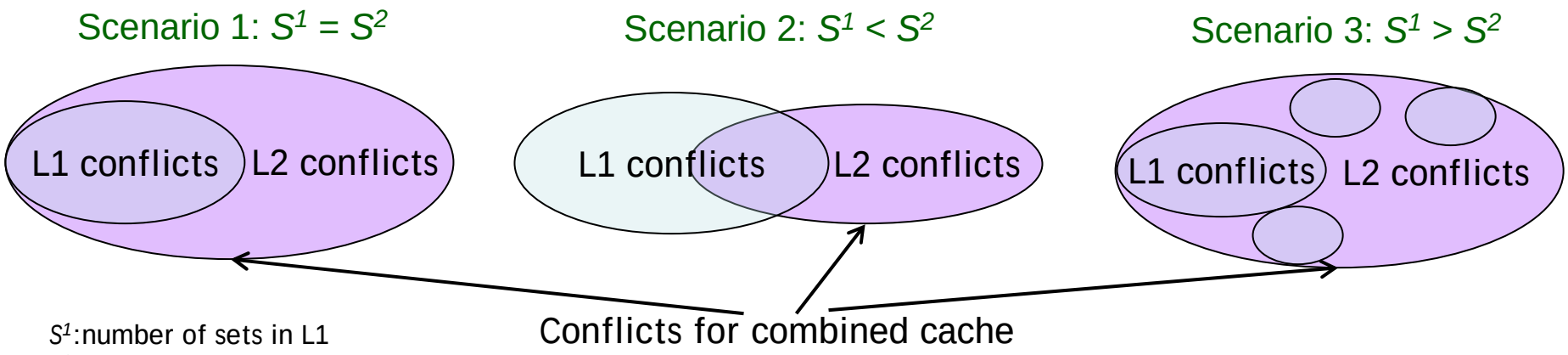
T-SPaCS Overview



B: block size
S^1:number of sets in L1
S^2: number of sets in L2
W^1 : number of associativities in L1
W^2 : number of associativities in L2

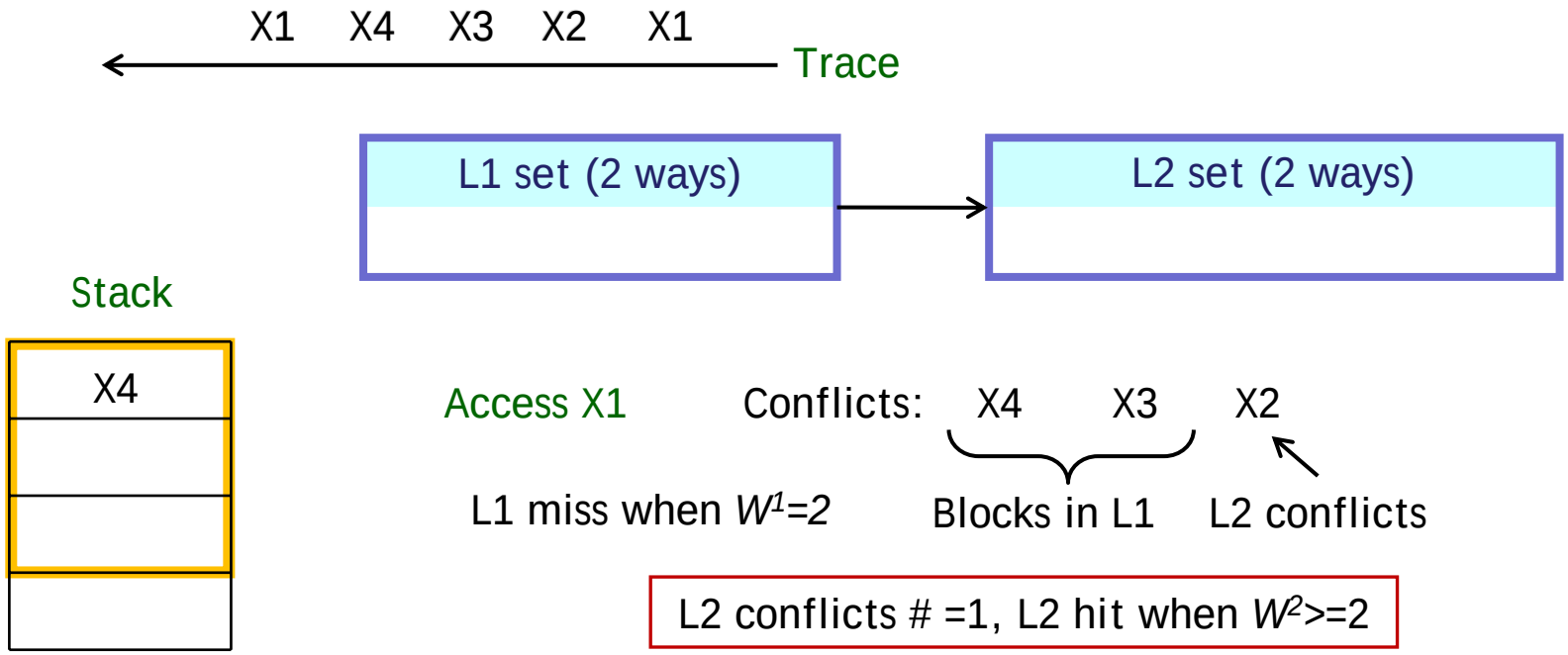
L2 Analysis

- Stack processing for combined cache
  - Conflict evaluation (same as single-level cache)
- ***Compare-exclude*** operation to derive L2 conflicts
  - Conflicts for combined cache still contain some conflicts stored in L1
  - Isolate the exclusive L2 conflicts
  - Based on three different inclusion relationships; consider as three scenarios



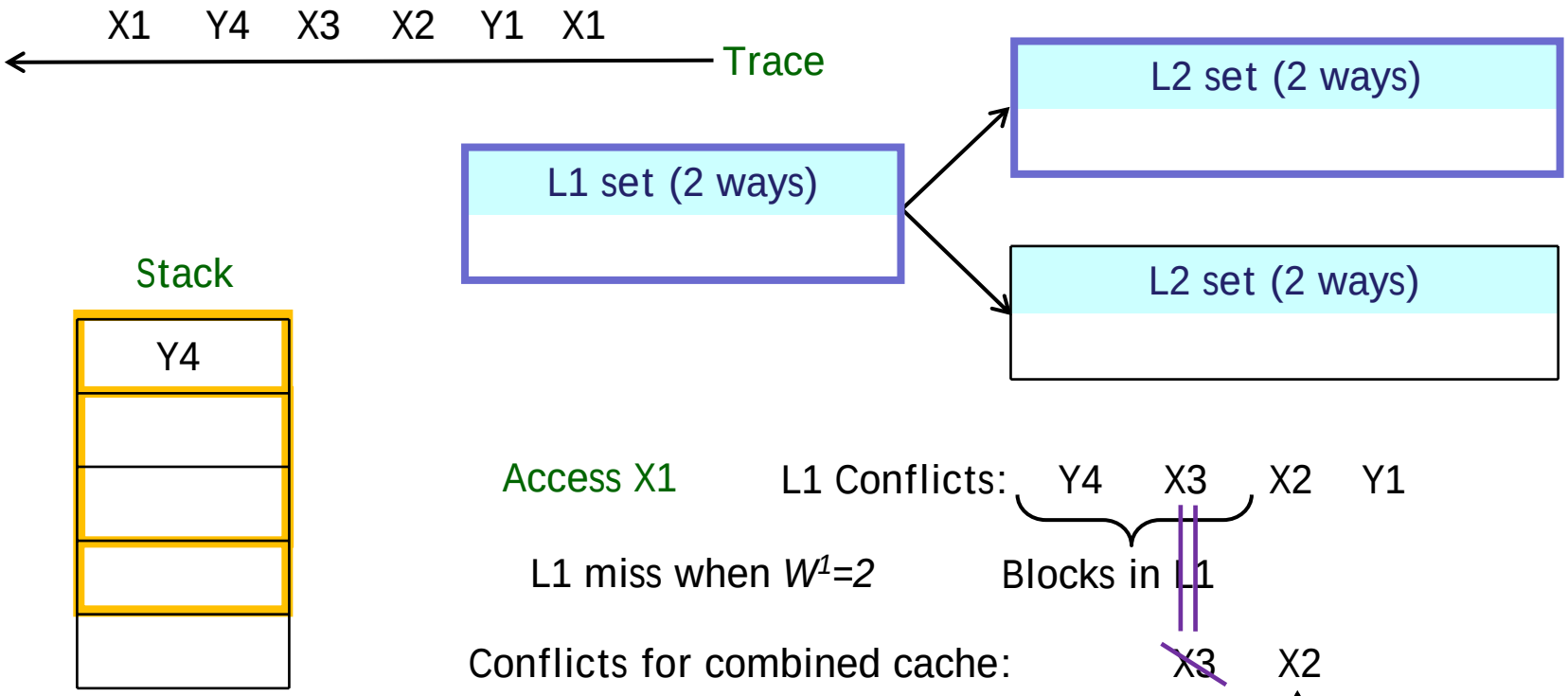
S^1:number of sets in L1
S^2: number of sets in L2

Scenario 1: $S^1 = S^2$



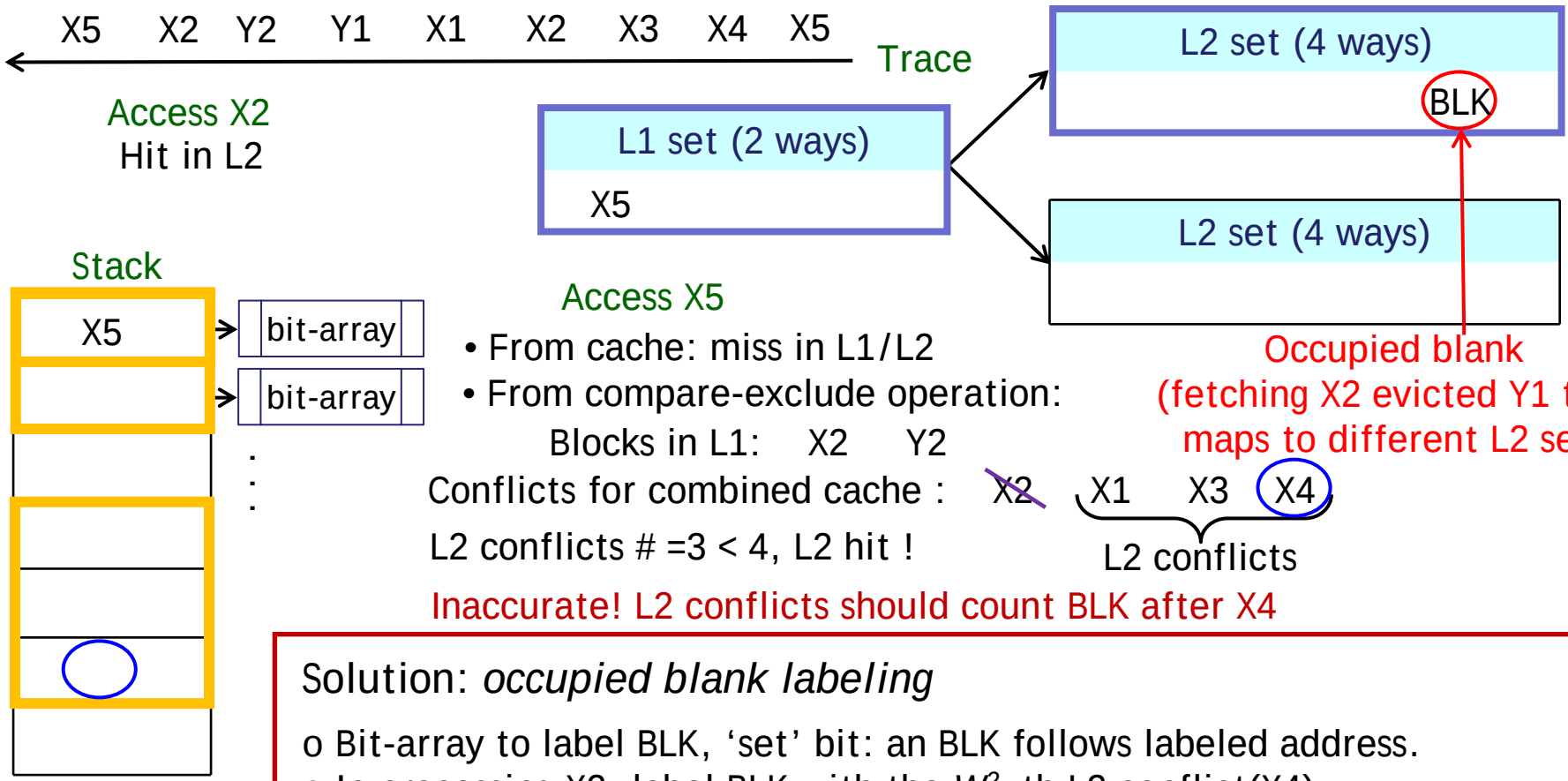
S^1:number of sets in L1
S^2: number of sets in L2

Scenario 2: $S^1 < S^2$



S^1:number of sets in L1
S^2: number of sets in L2

Special Case in Scenario 2

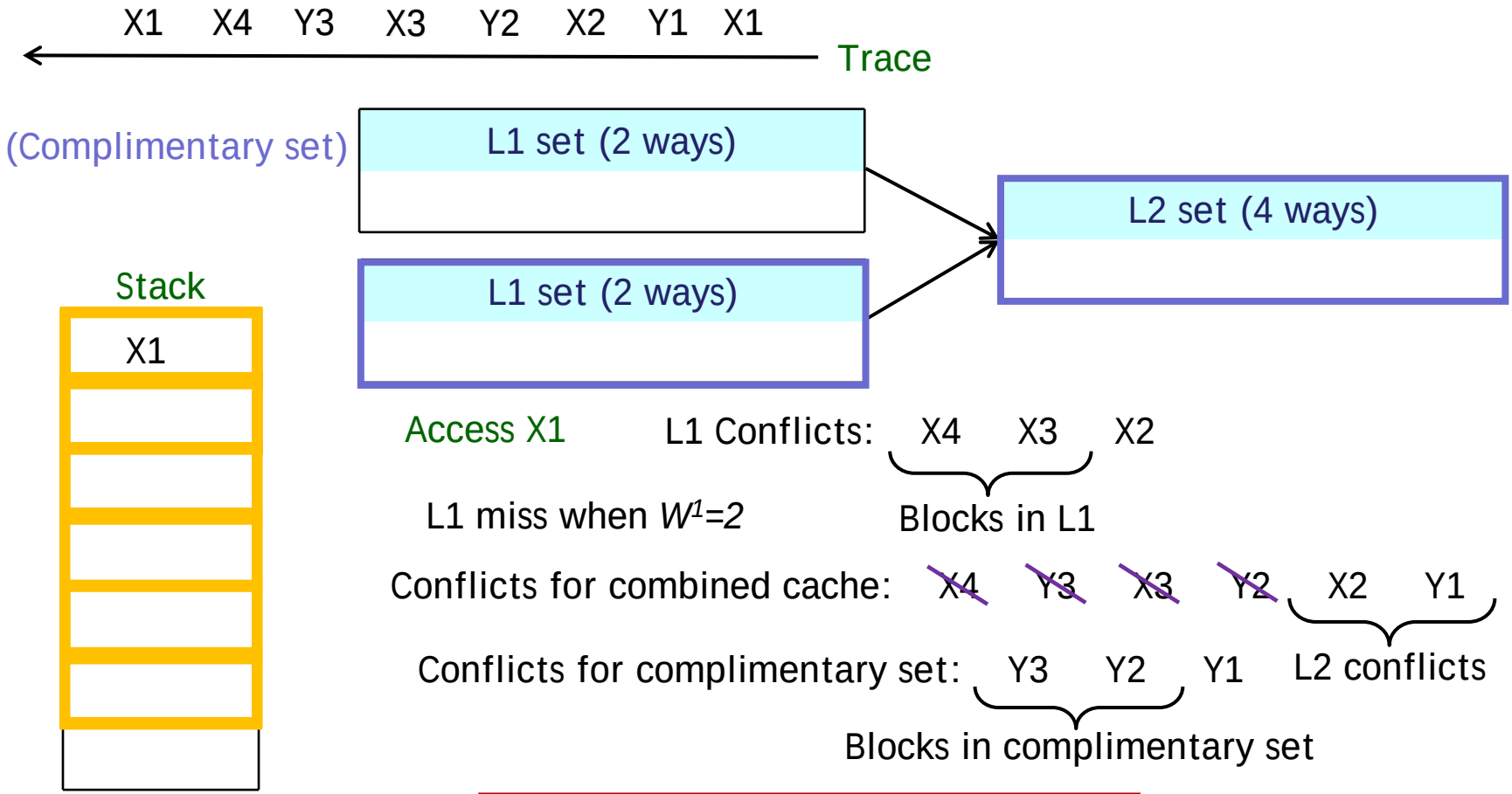


Solution: *occupied blank labeling*

- Bit-array to label BLK, 'set' bit: an BLK follows labeled address.
- In processing X2, label BLK with the W^2-th L2 conflict(X4).
- In processing X5, detected BLK in the bit-array of X4. (i.e., X4 is the last block in L2). X5 is L2 miss.

S^1:number of sets in L1
S^2: number of sets in L2

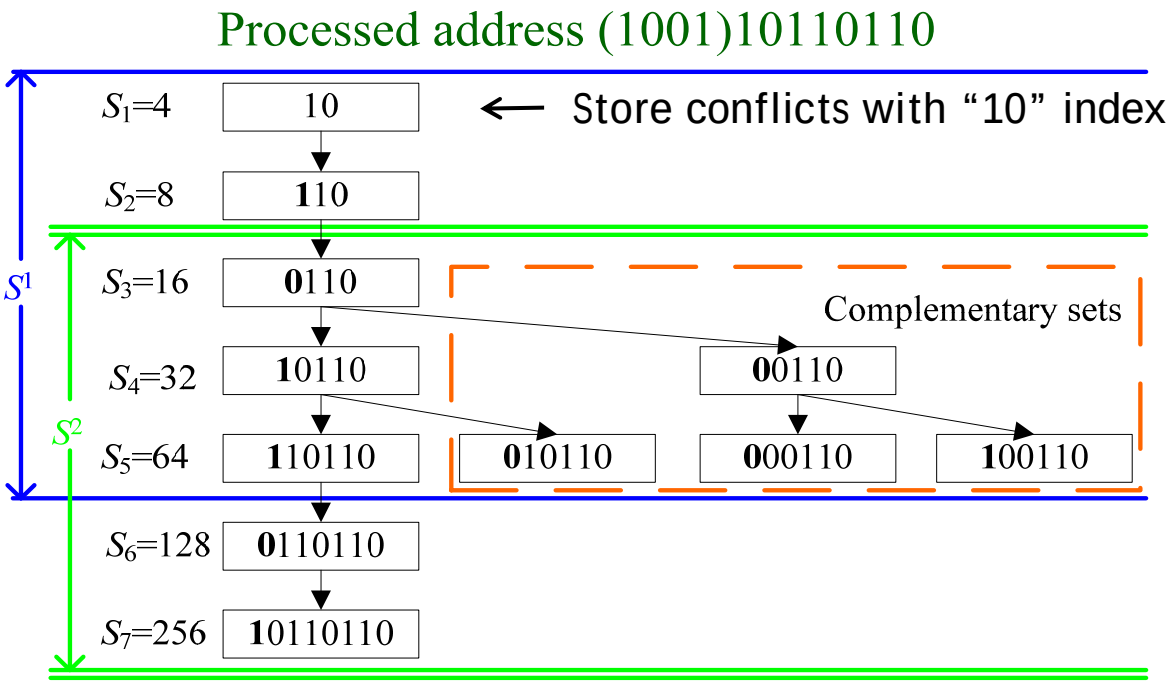
Scenario 3: $S^1 > S^2$

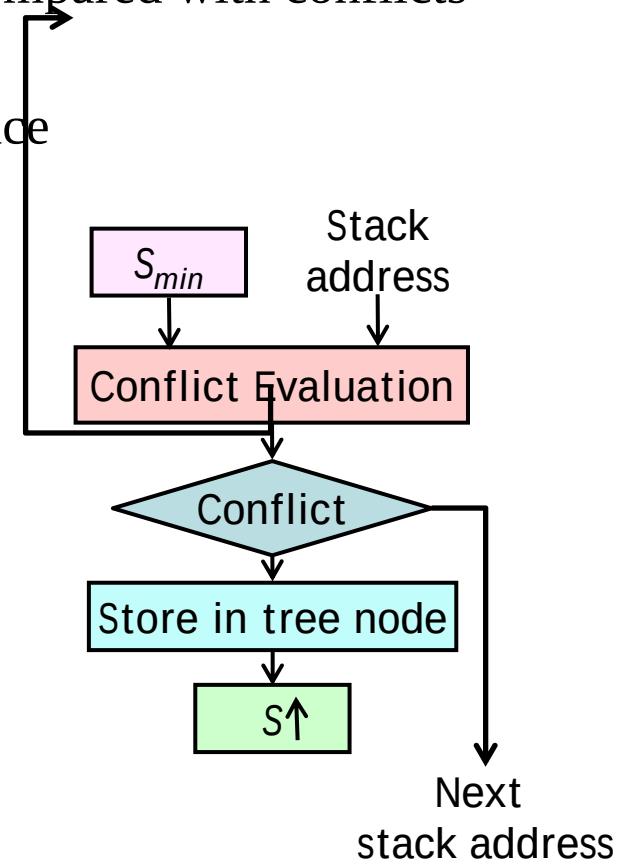


S^1:number of sets in L1
S^2: number of sets in L2

Accelerate Stack Processing

- Stack processing: ***very time consuming!***
- Conflicts for one L1 configuration repeatedly compared with conflicts for all L2 configurations
- Save conflicts in a tree structure for later reference





S^1:number of sets in L1
S^2: number of sets in L2

Experiment Setup

- Design space
  - L1: cache size (2k➔8k bytes); block size (16B➔64B); associativity (direct-mapped➔4-way)
  - L2: cache size (16k➔64k bytes); block size (16B➔64B); associativity (direct-mapped➔4-way)
  - 243 configurations
    - Exclusive cache requires L1 and L2 to have the same block size
- 24 benchmarks from EEMBC, Powerstone, and MediaBench
- Modify ‘sim-fast’ to generate access traces
- Modify ‘sim-cache’ to simulate exclusive hierarchy cache to produce the *exact* miss rates for comparison
- Build energy model to determine optimal cache configuration with minimum energy consumption (Gordon-Ross 09)

Results – Miss Rate Accuracy

- L1 miss rate
  - 100% accurate for all benchmarks
- L2 miss rate
  - Accurate for 240 configurations (99% of the design space)
  - Across all benchmarks

Max. average miss rate err.	Max. standard deviation	Max. absolute miss rate err.
1.16%	0.64%	1.55%

- Inaccuracy comes from Scenario 3: $S^1 > S^2$
  - Reason
    - Multiple L1 sets evict blocks in the same L2 set
    - Eviction order is not consistent to access order
  - Introduced error is small
- Tuning accuracy: accurately determined energy optimal cache!

S^1:number of sets in L1
S^2: number of sets in L2

Simplified-T-SPaCS

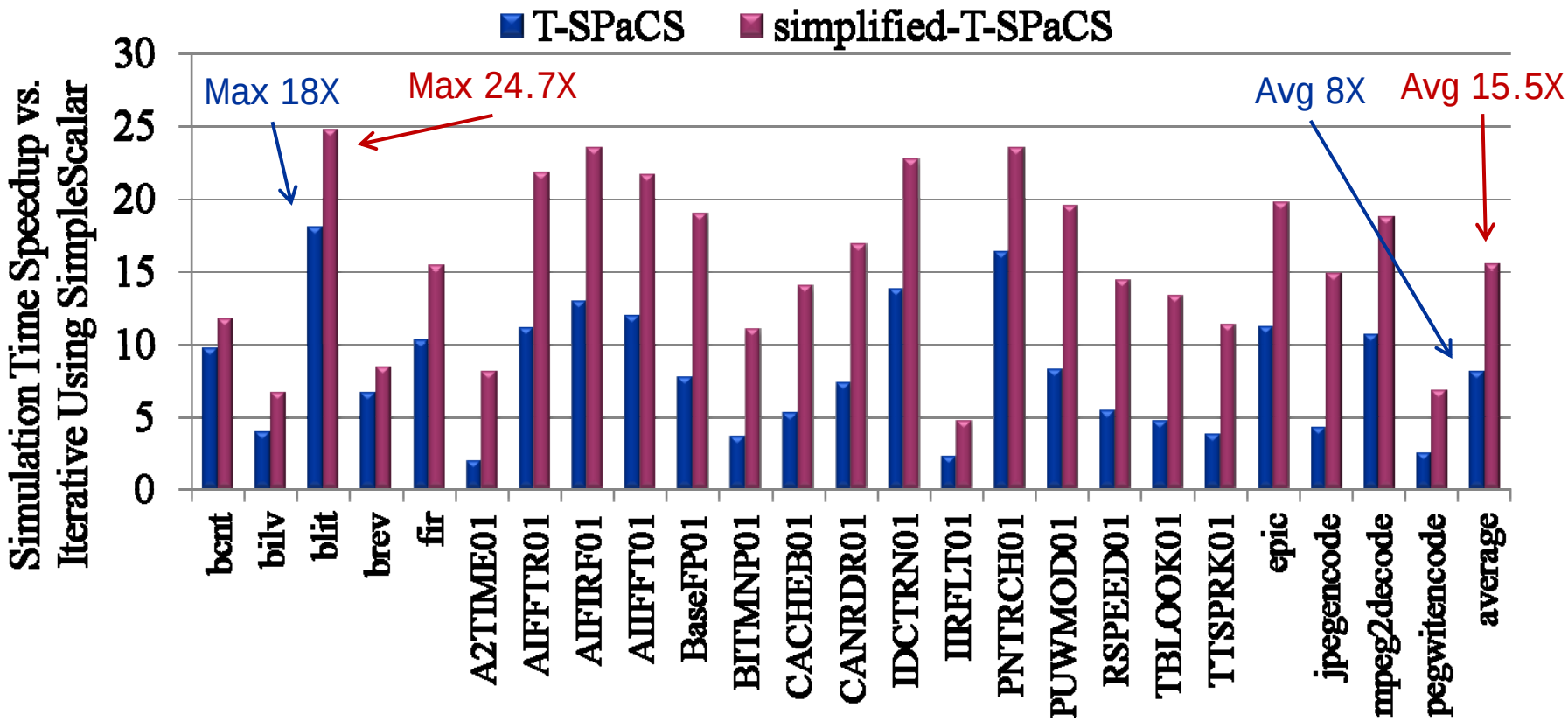
- Omit occupied blank labeling to reduce complexity and simulation time
- Tradeoff – additional miss rate error
  - L2 miss rate errors for additional 228 configurations where $S^1 < S^2$ (95% of the design space)
  - Across all benchmarks

Max. average miss rate err.	Max. standard deviation	Max. absolute miss rate err.
0.71%	0.90%	3.35%

- Tuning accuracy: accurately determined energy optimal cache!

S^1:number of sets in L1
S^2: number of sets in L2

Simulation Time Efficiency



Conclusions

- T-SPaCS simulates instruction cache with exclusive hierarchy in a single-pass
- T-SPaCS reduces the storage and time complexity
  - T-SPaCS is 8X faster than iterative simulation on average
  - Simplified-T-SPaCS increases average simulation speedup to 15X at the expense of inaccurate miss rates for 95% of the design space
  - Both T-SPaCS and simplified-T-SPaCS can determine accurate optimal energy configurations
- Our ongoing work extends T-SPaCS to simulate data and unified cache, and implement in hardware for dynamic cache tuning