A Scalable Approach to Thread-Level Speculation

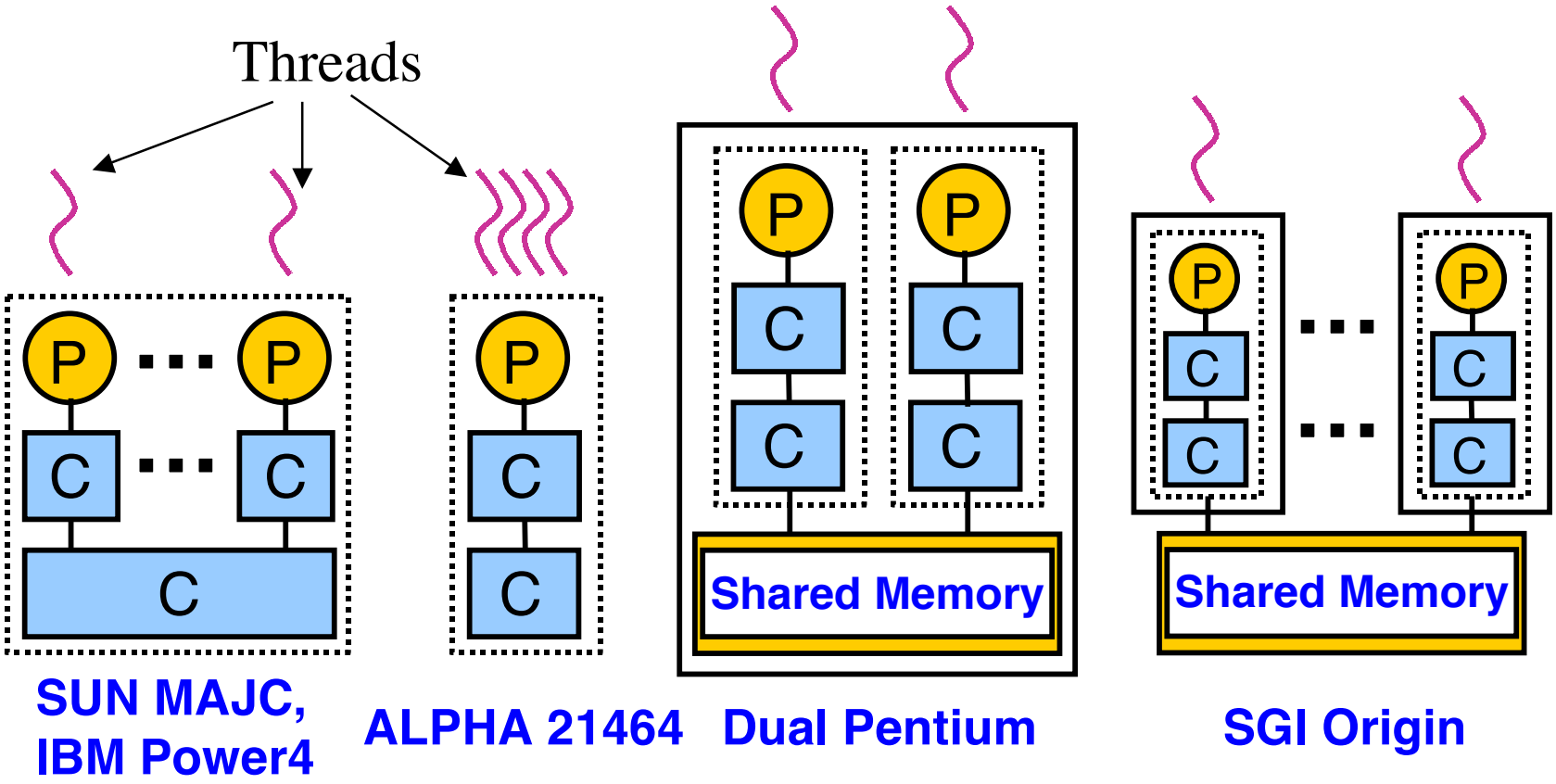
J. Gregory Steffan, Christopher B. Colohan,

Antonia Zhai, and Todd C. Mowry

Computer Science Department

Carnegie Mellon University

Multithreaded Machines Are Everywhere



☞ **How can we use them? Parallelism!**

Automatic Parallelization

Proving independence of threads is hard:

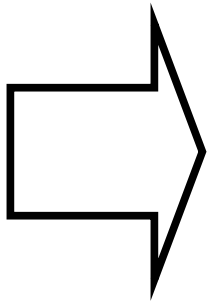
- complex control flow
- complex data structures
- pointers, pointers, pointers
- run-time inputs

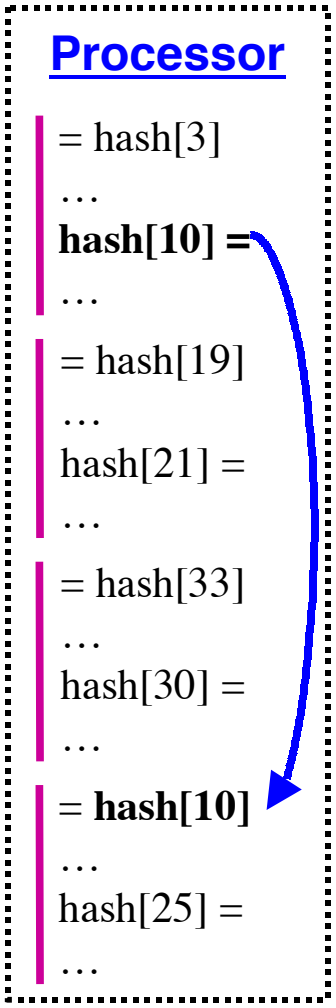
How can we make the compiler's job feasible?

☞ **Thread-Level Speculation (TLS)**

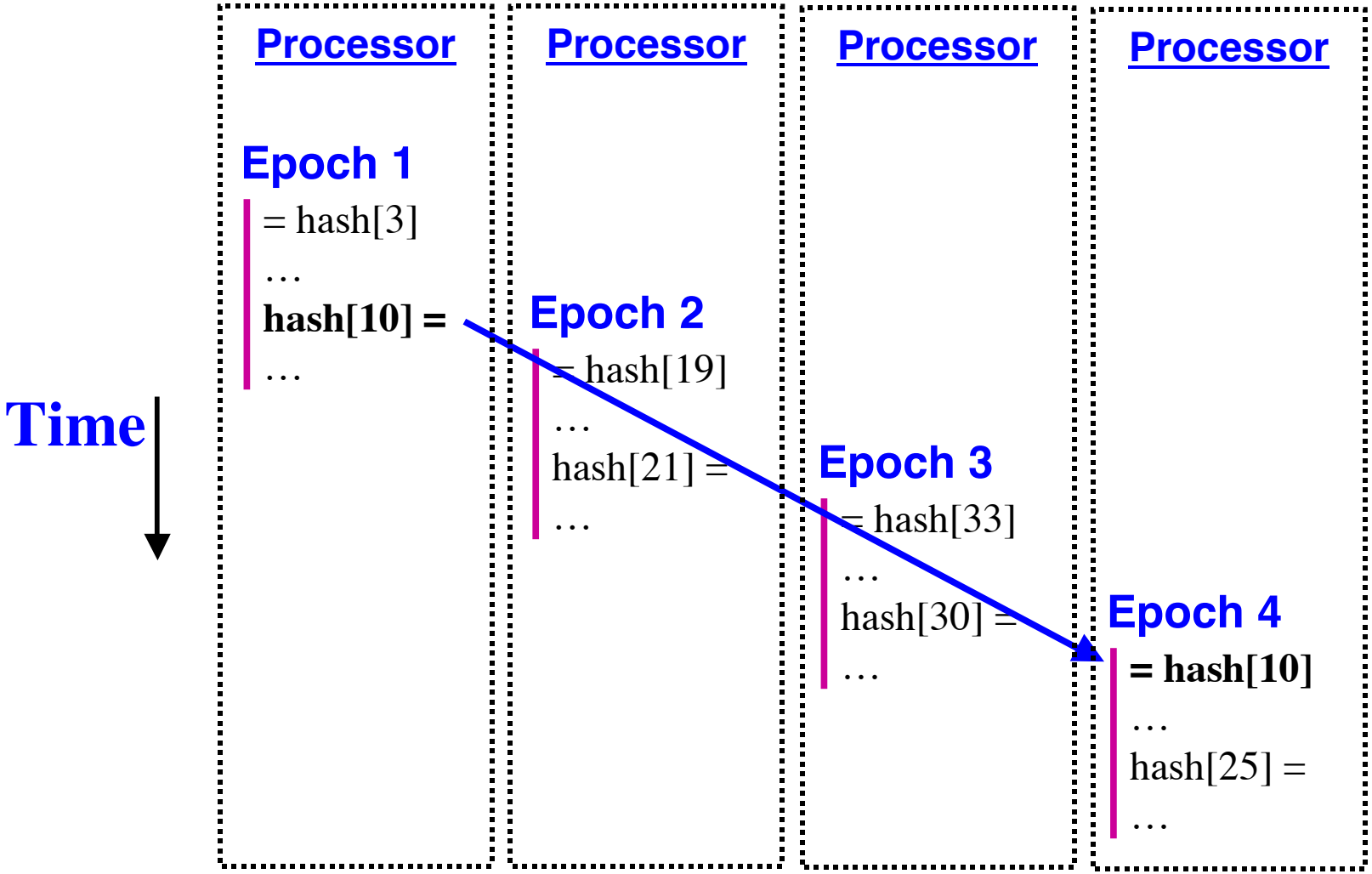
Example

```
while (...){
    x = hash[index1];
    …
    hash[index2] = y;
    ...
}
```

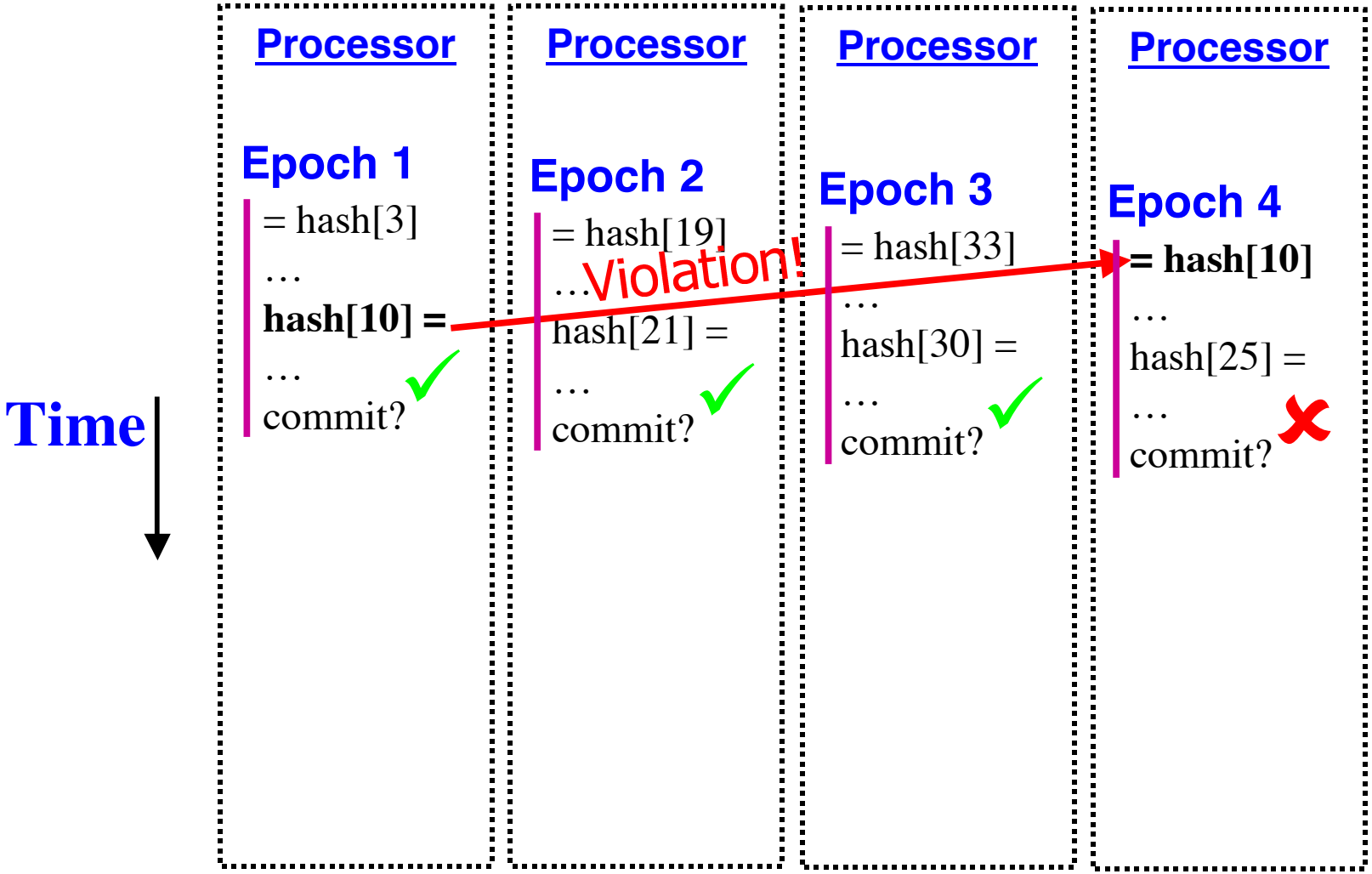




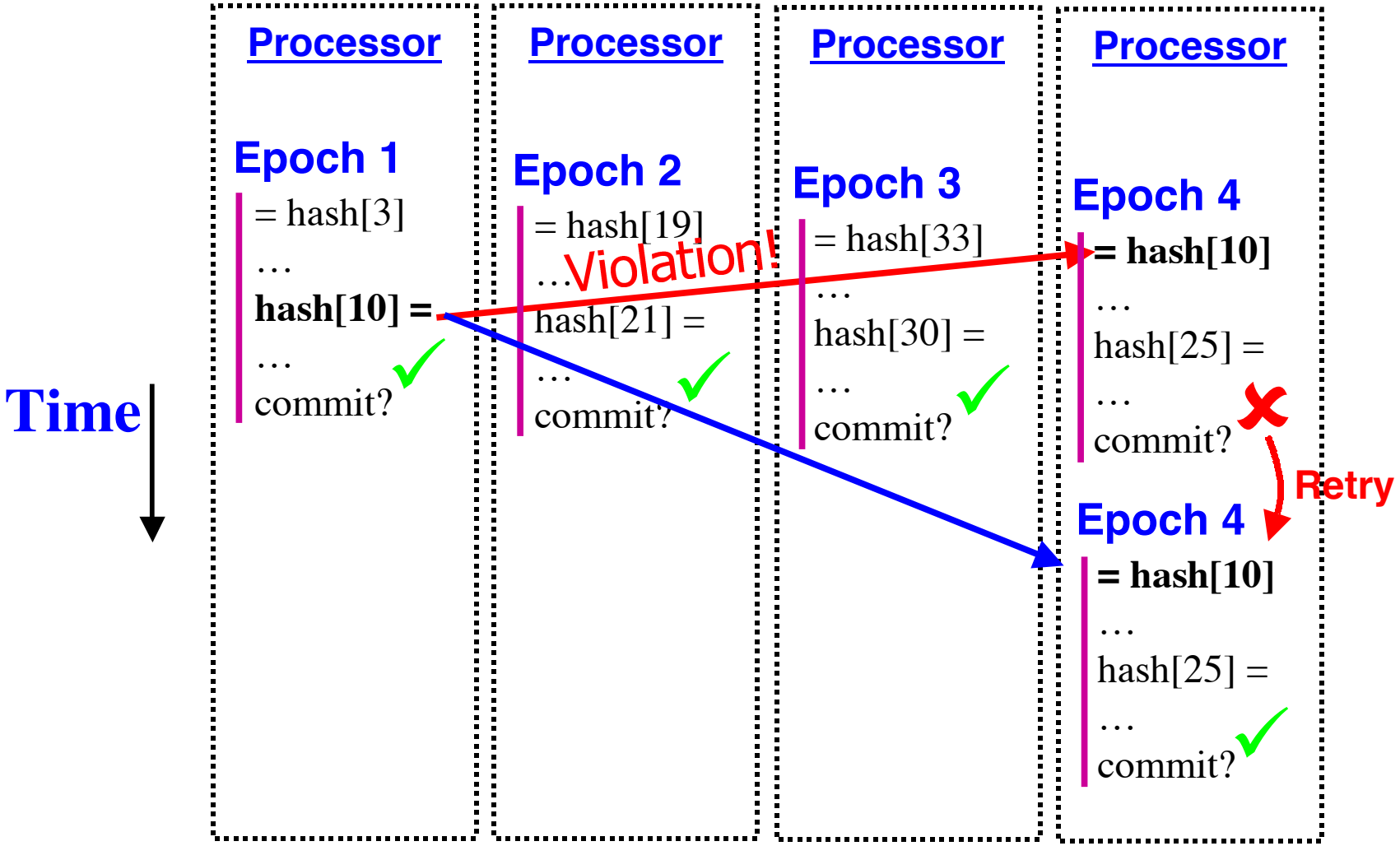
Example of Thread-Level Speculation



Example of Thread-Level Speculation



Example of Thread-Level Speculation



Goals of Our Approach

1) Handle arbitrary memory accesses
   - i.e. not just array references
2) Preserve performance of non-speculative workloads
   - keep hardware support minimal and simple
3) Apply to any scale of multithreaded architecture
   - CMPs, SMT processors, more traditional MPs

☞ **effective, simple, and scalable TLS**

Overview of Our Approach

System requirements:

1) Detect data dependence violations
   - extend invalidation-based cache coherence

2) Buffer speculative modifications
   - use the caches as speculative buffers

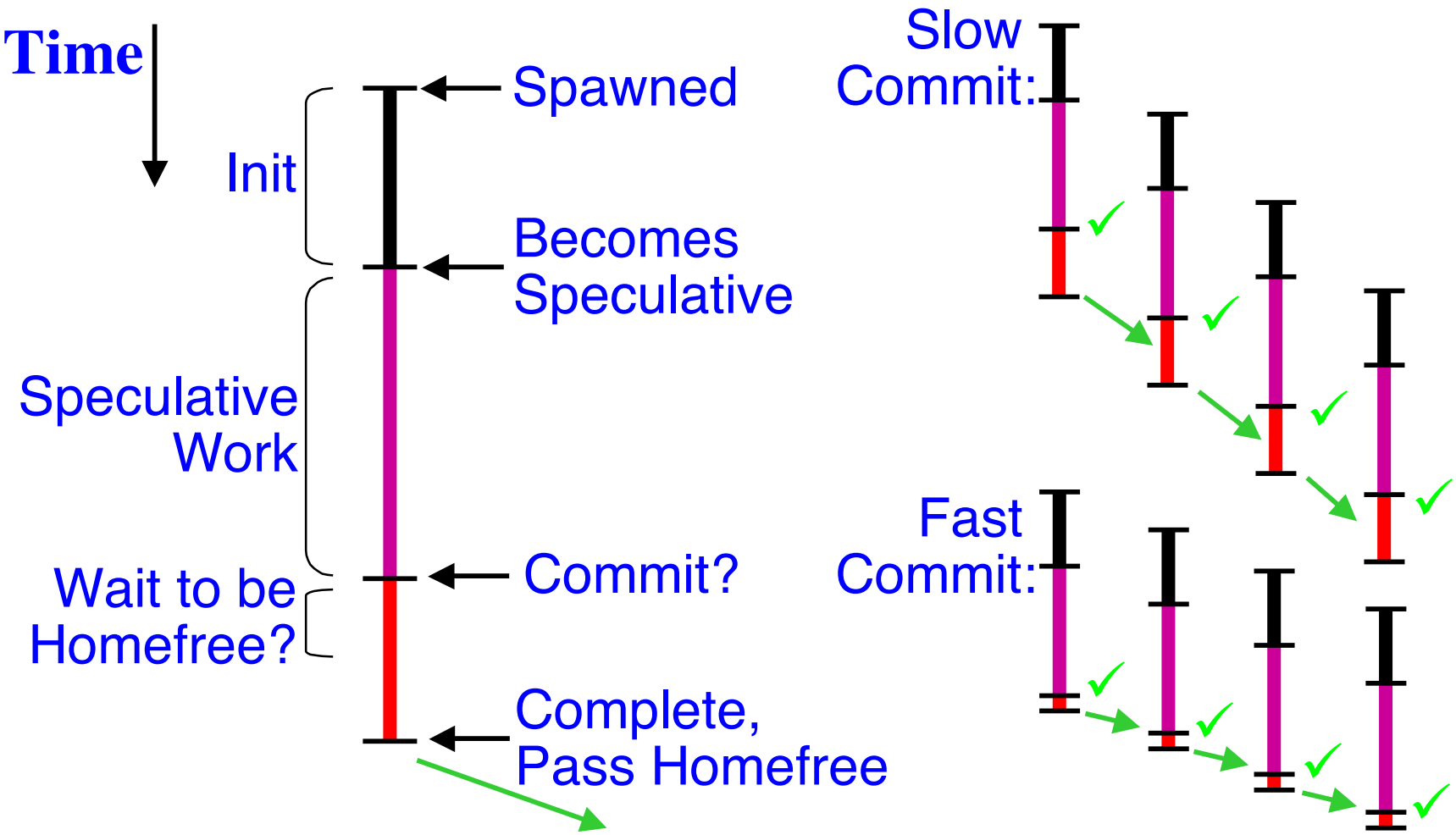
☞ **coherence already works at a variety of scales**

☞ **hence our scheme is also scalable**

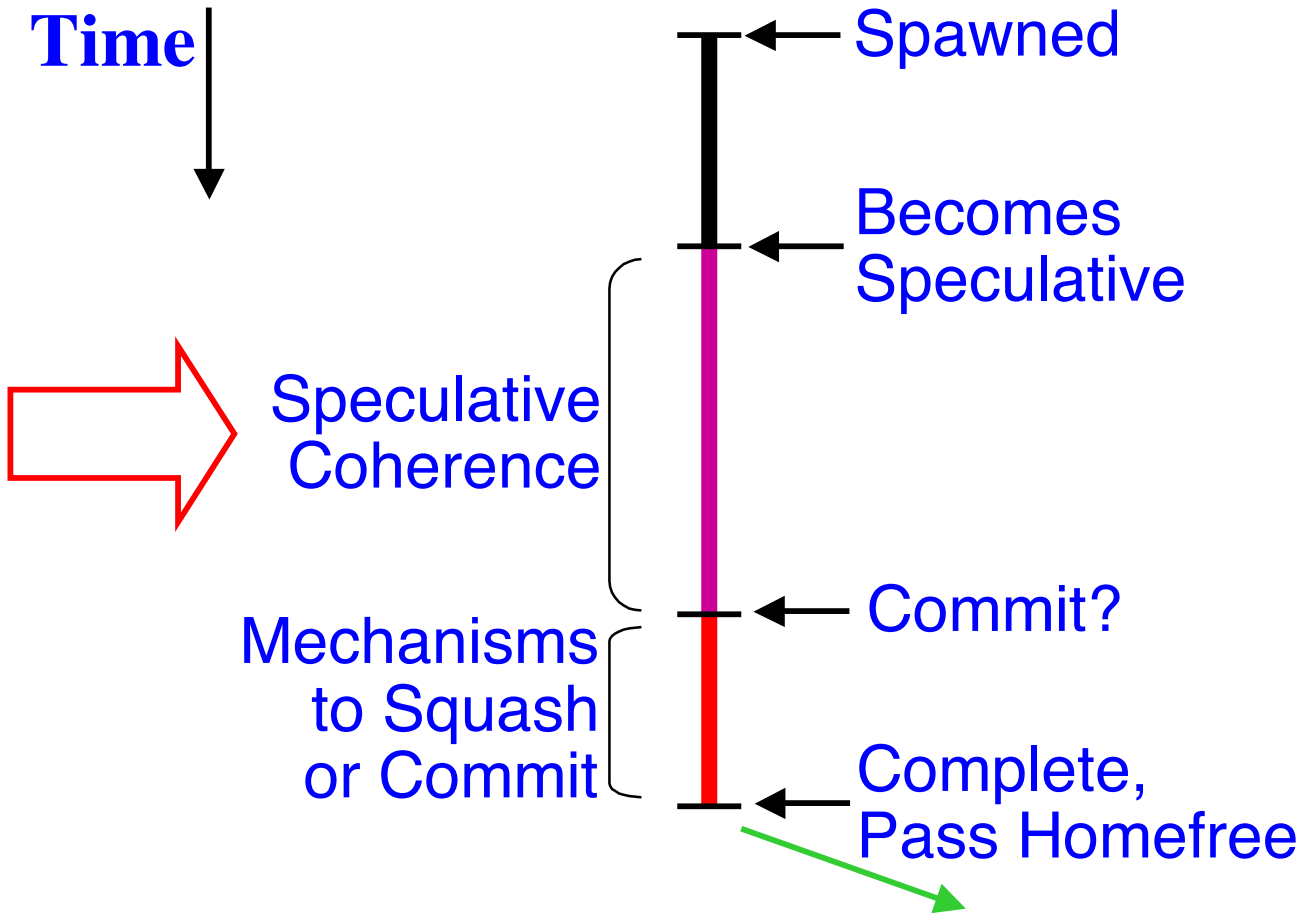
Outline

- ☞ Details of our Approach
  - life cycle of an epoch
  - speculative coherence
  - what happens at commit time
  - forwarding data between epochs
- Performance
- Conclusions

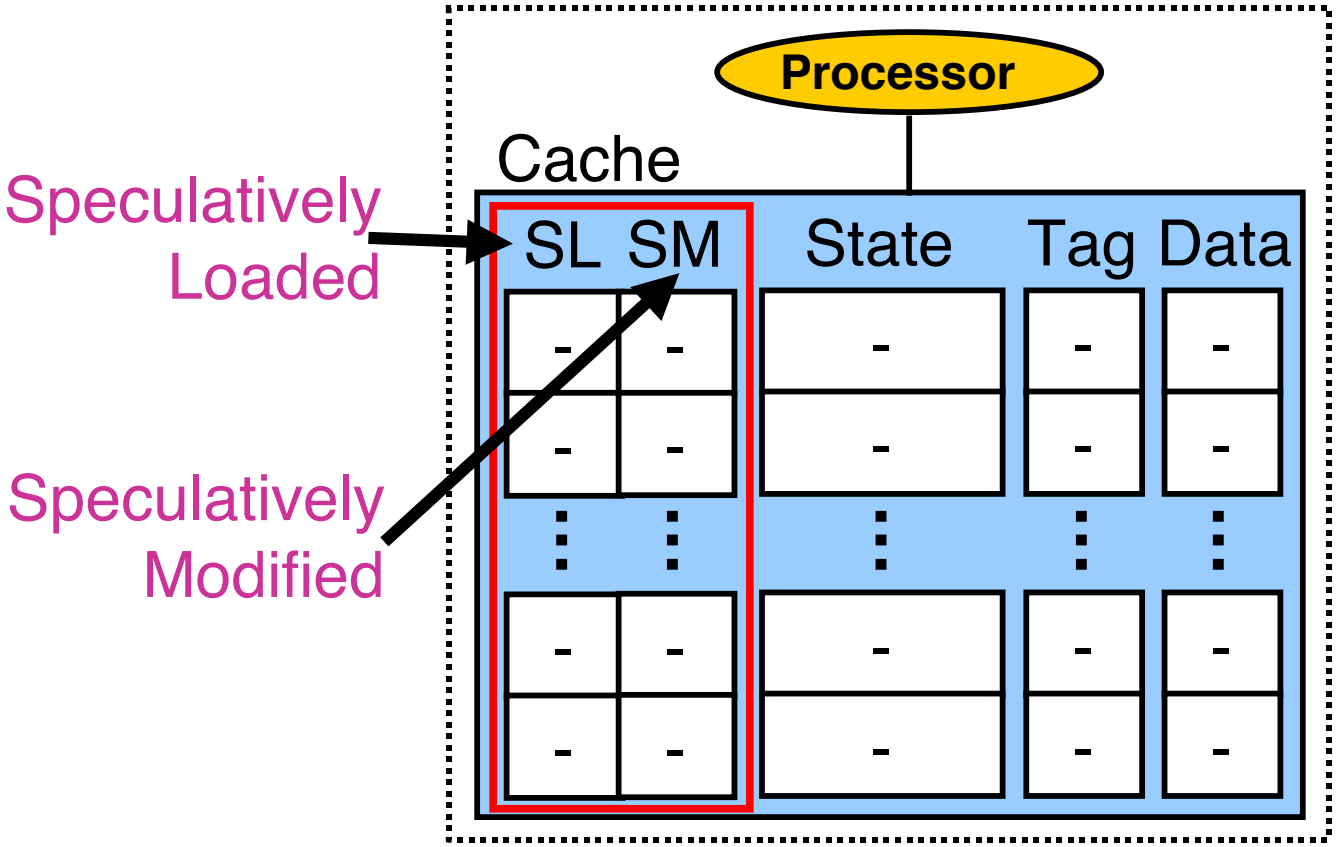
Life Cycle of an Epoch



Life Cycle of an Epoch

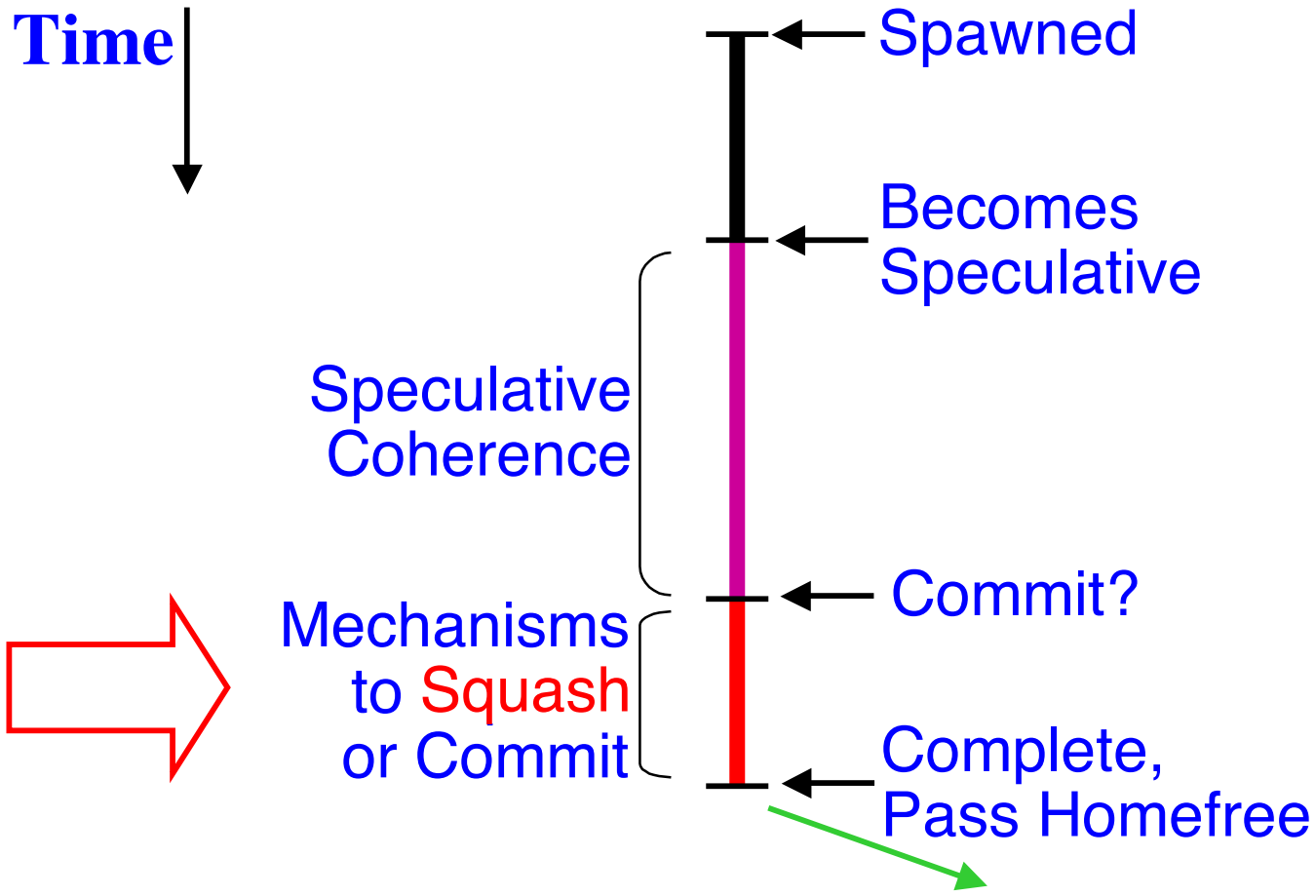


Implementation of Speculative State

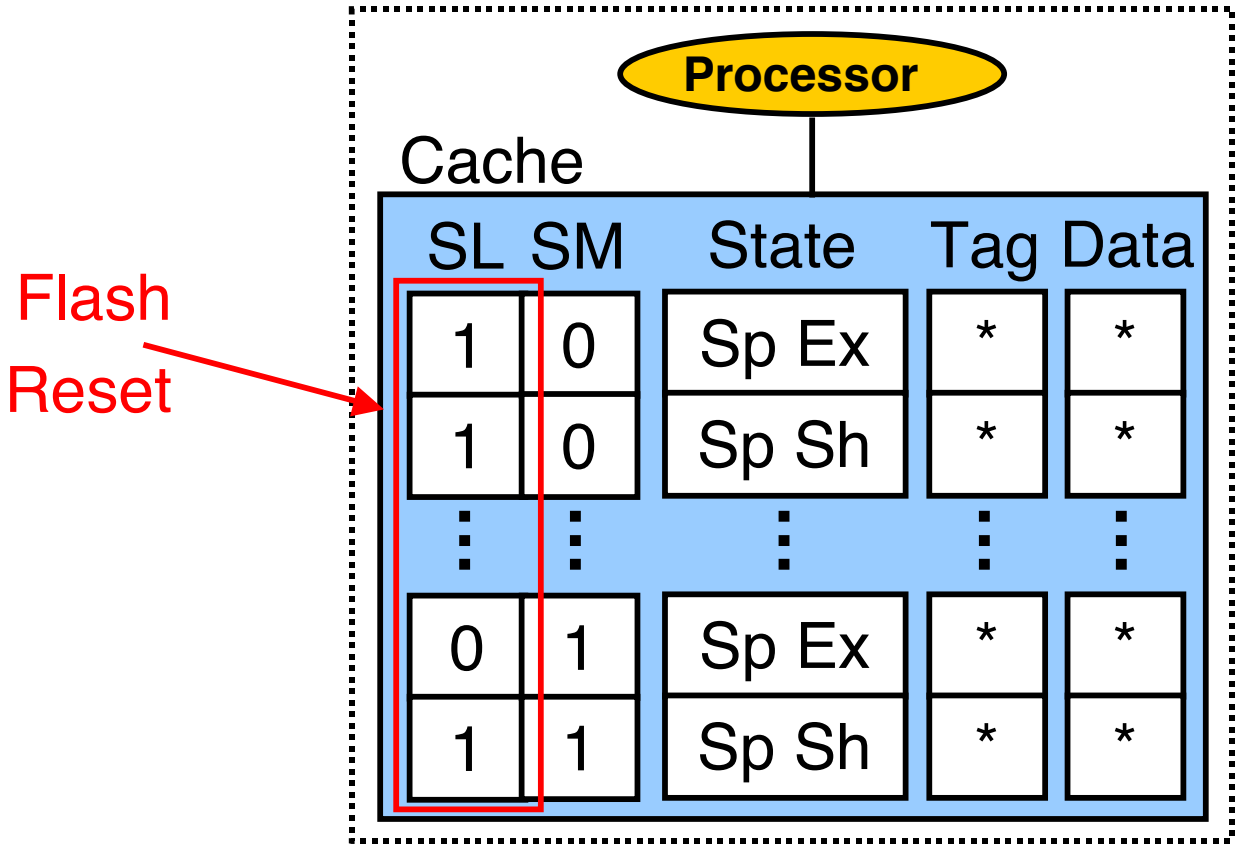


☞ **modest amount of extra space**

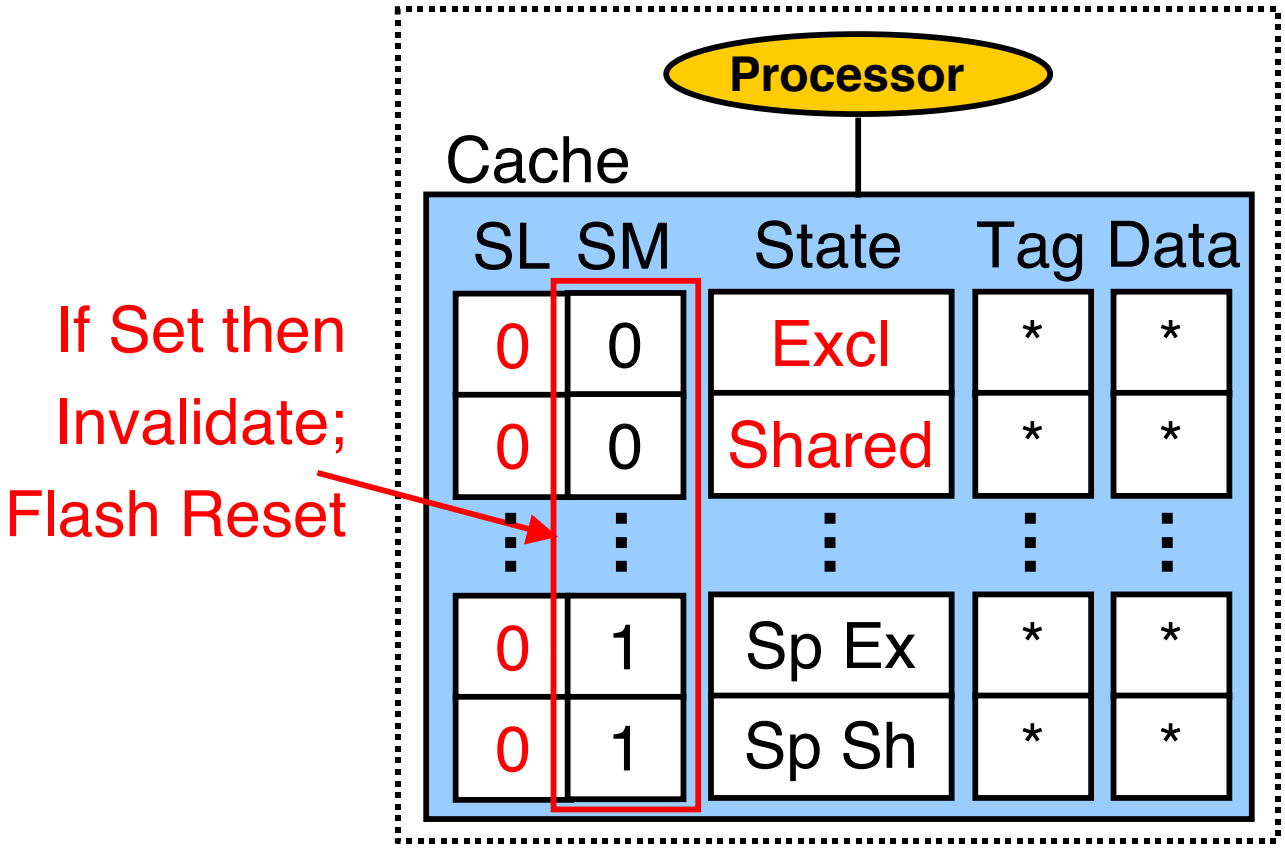
Life Cycle of an Epoch



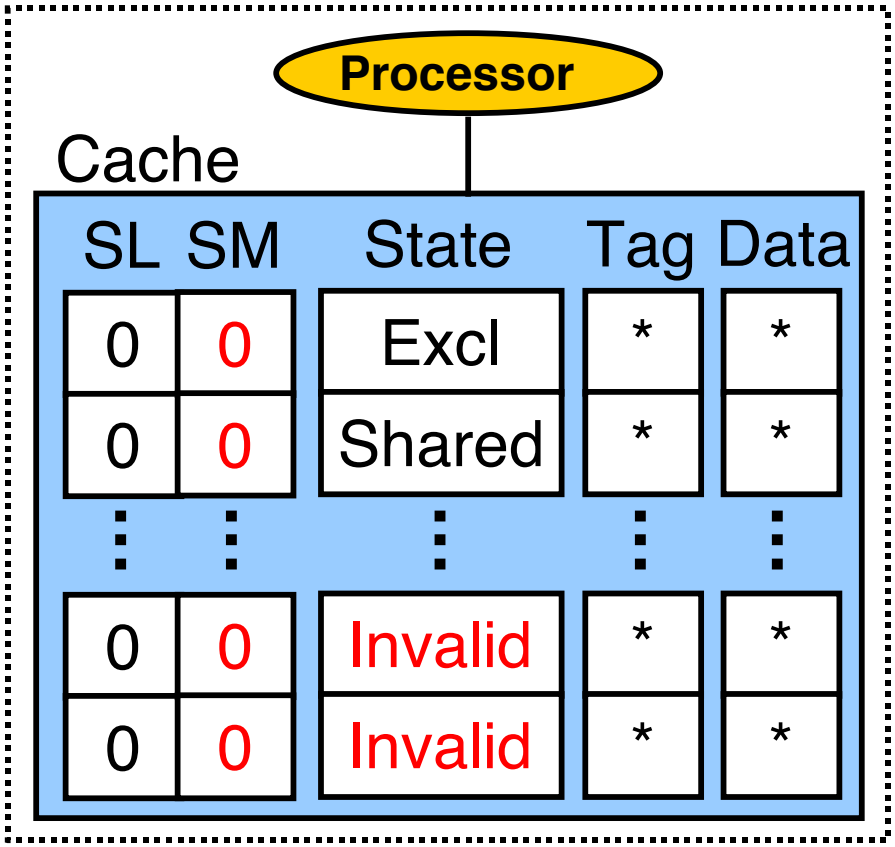
When Speculation Fails



When Speculation Fails

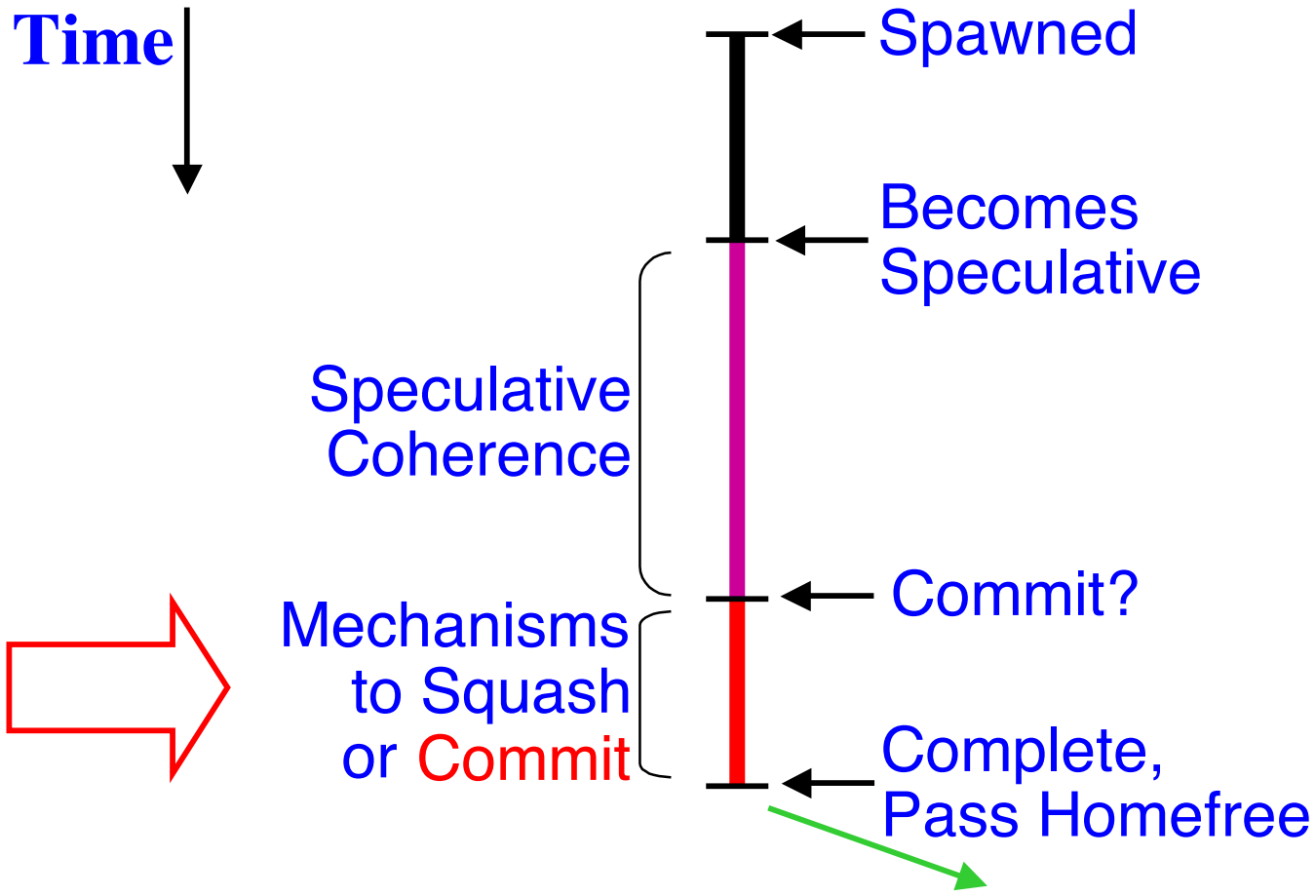


When Speculation Fails

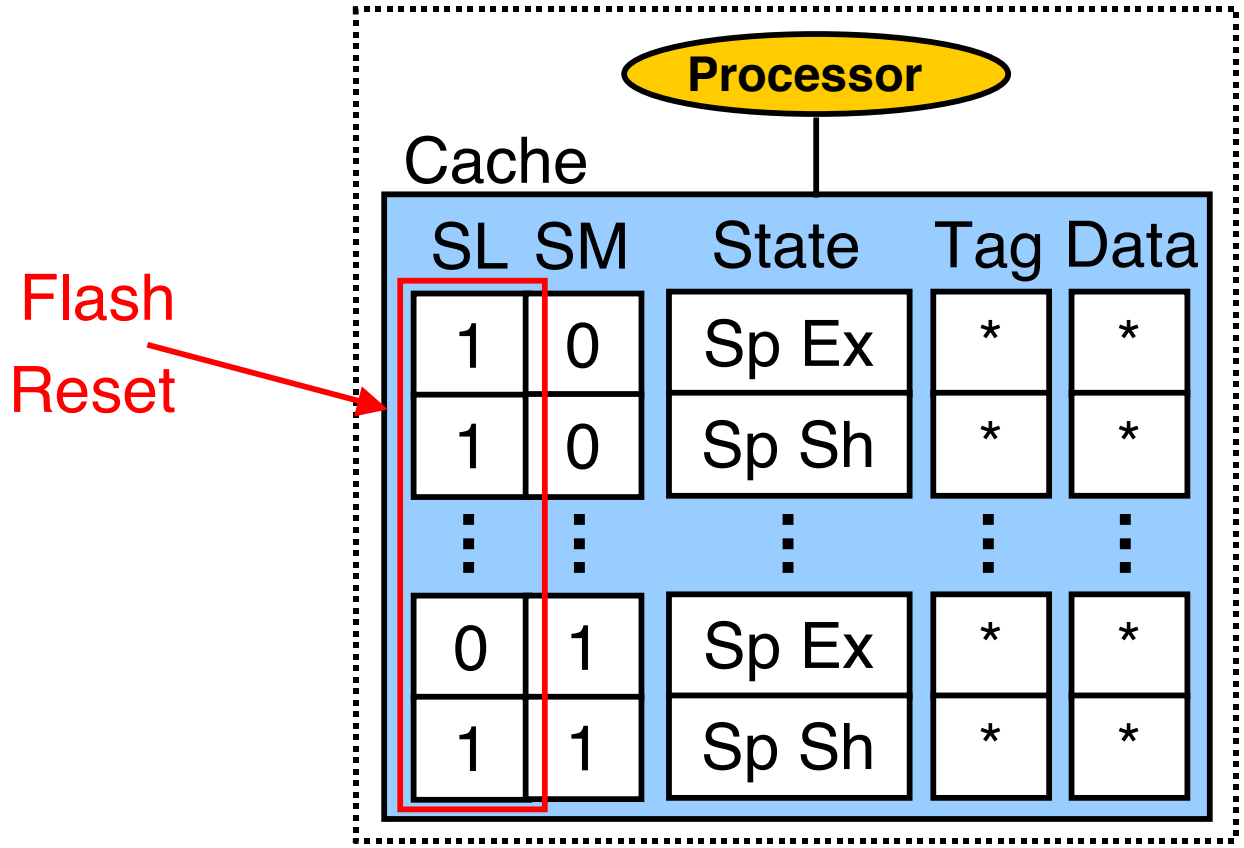


☞ **quick bit operation**

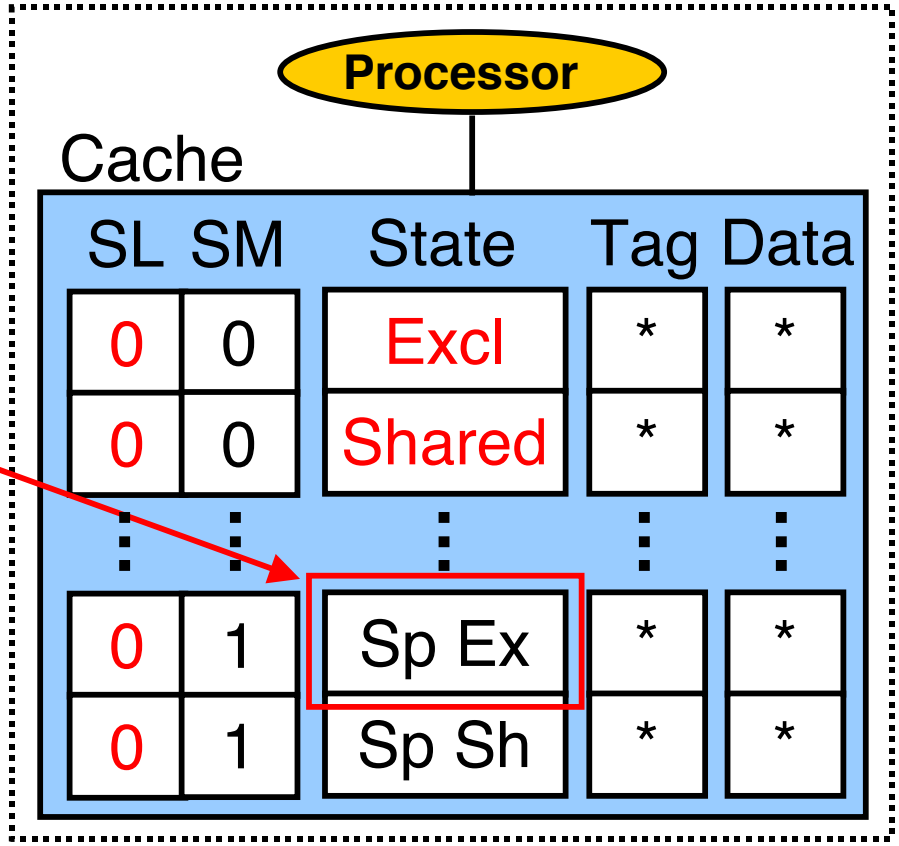
Life Cycle of an Epoch



When Speculation Succeeds

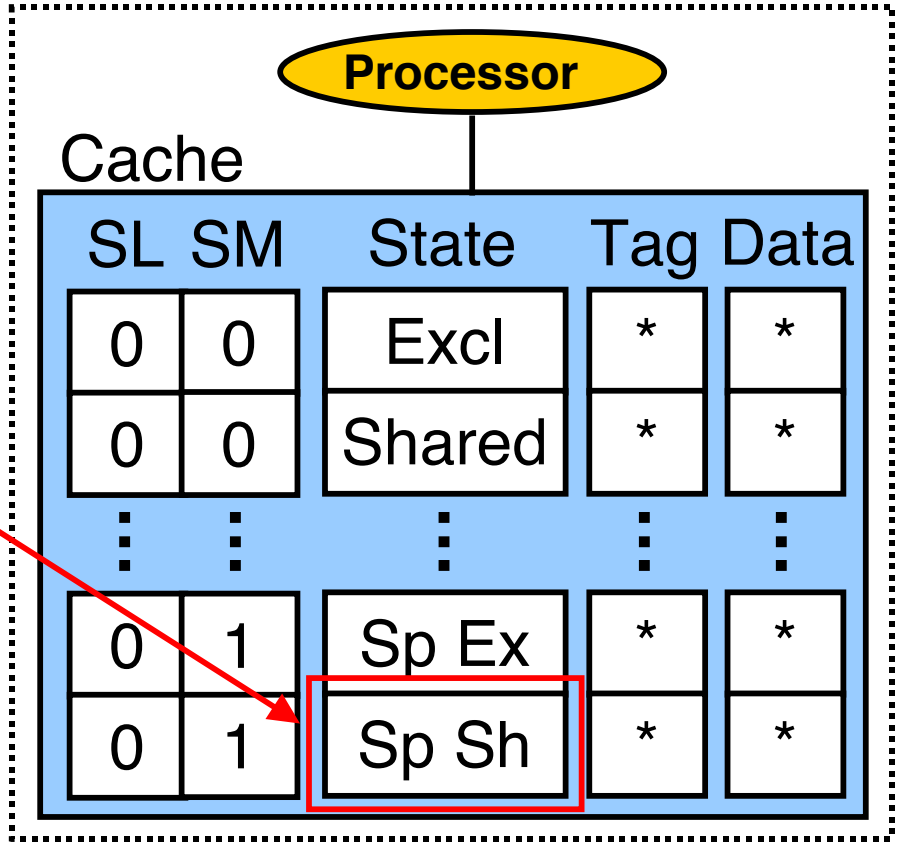


When Speculation Succeeds



SM & Exclusive:
Become Dirty

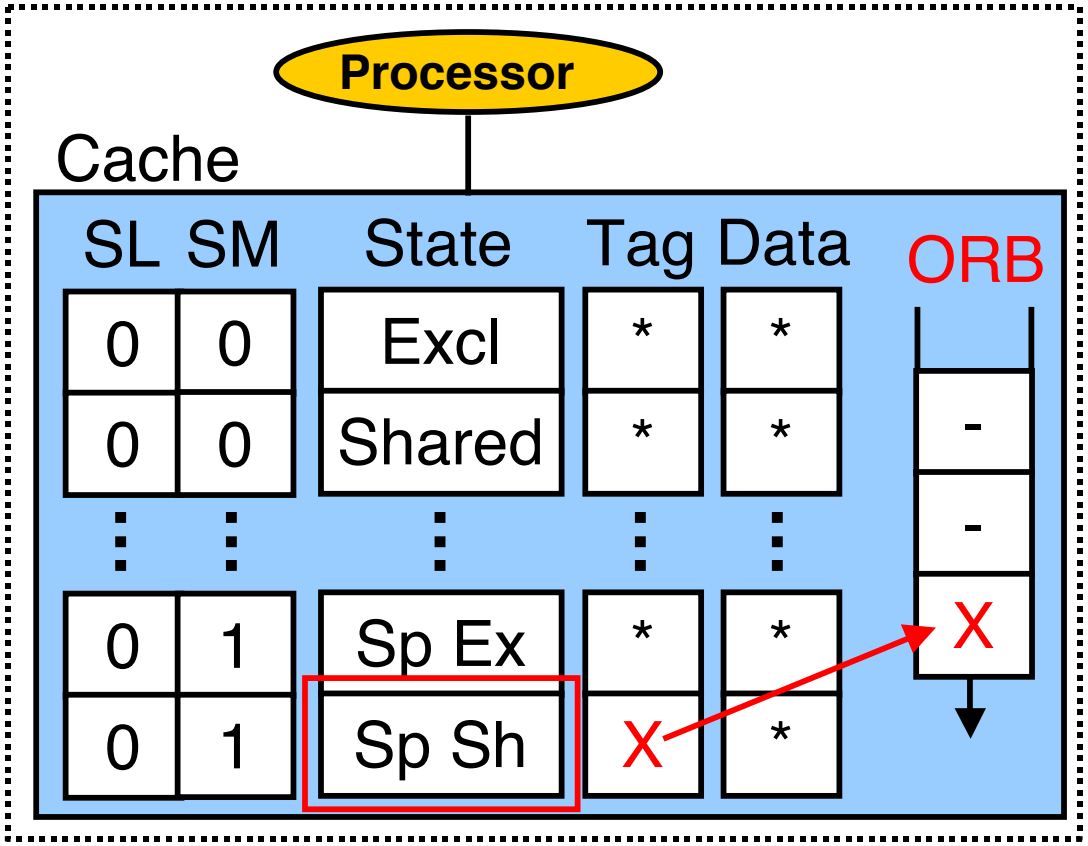
When Speculation Succeeds



SM & Shared: Need Exclusive Access

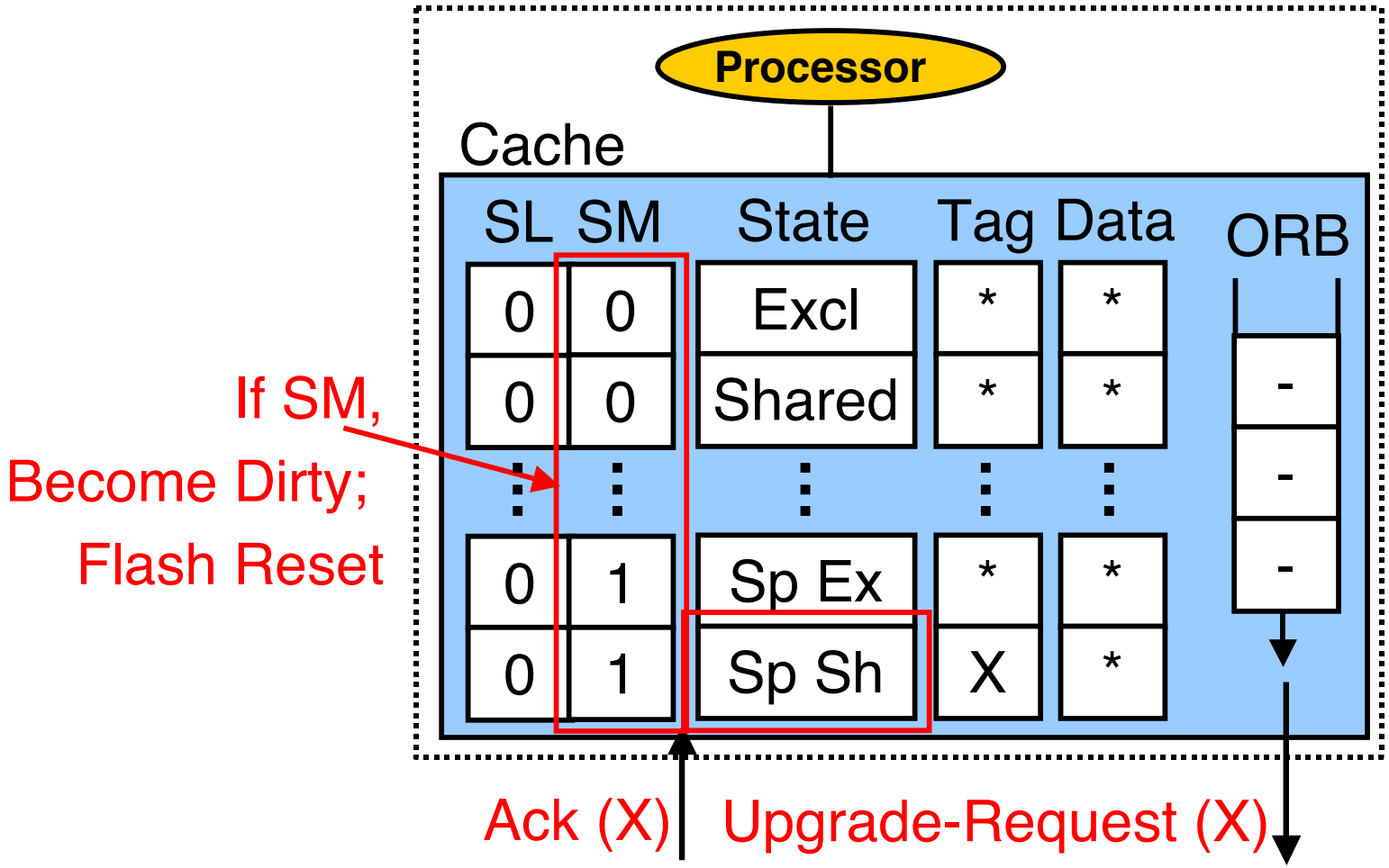
☞ **want to avoid searching entire cache**

When Speculation Succeeds

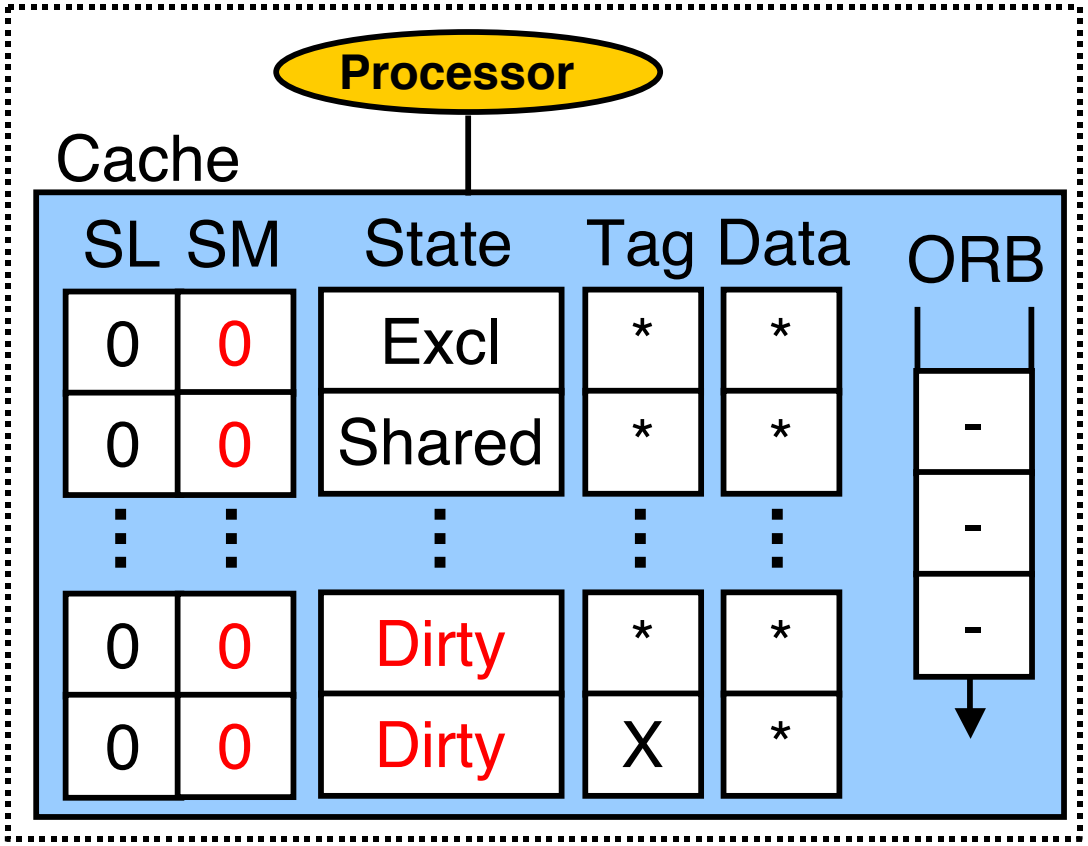


☞ ***ownership required buffer* (ORB)**

When Speculation Succeeds

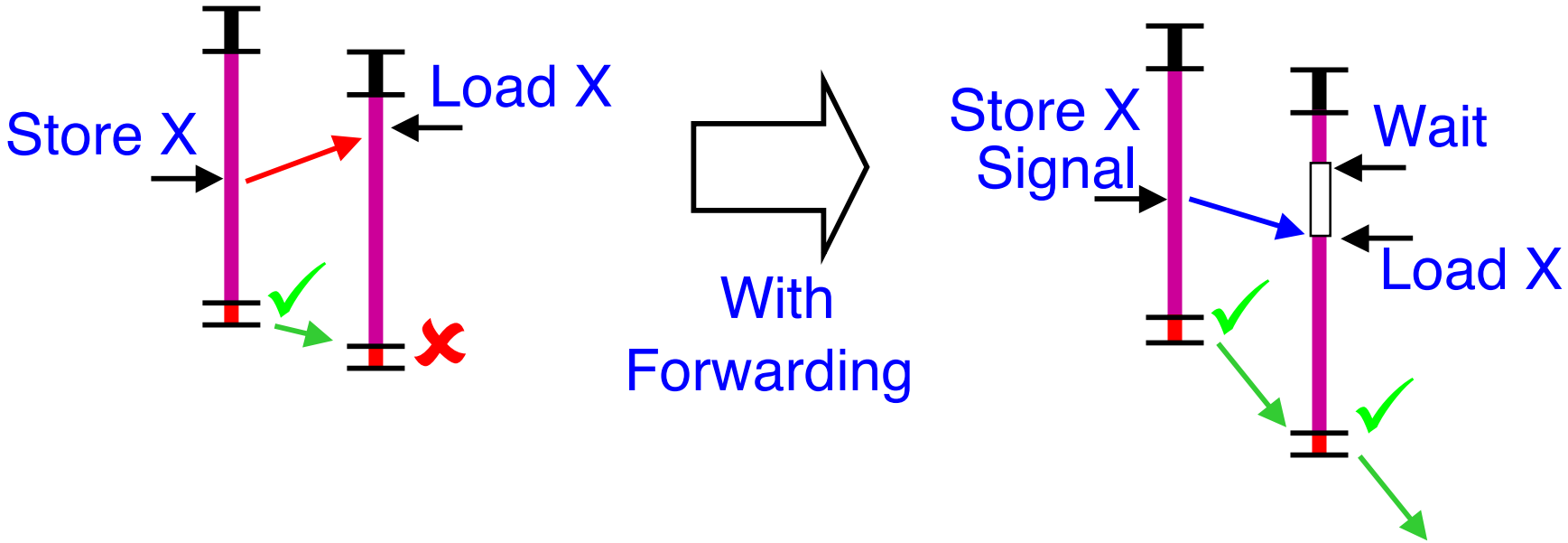


When Speculation Succeeds



☞ **flush the ORB, then quick bit operations**

Forwarding Data Between Epochs



- predictable dependences cause frequent violations
- compiler inserts wait-signal synchronization

☞ **synchronize to avoid violations**

Outline

- Details of our Approach

☞ Performance

  - simulation infrastructure
  - single-chip multiprocessor performance
  - scaling beyond chip boundaries

- Conclusions

Simulation Infrastructure

Compiler system and tools based on SUIF

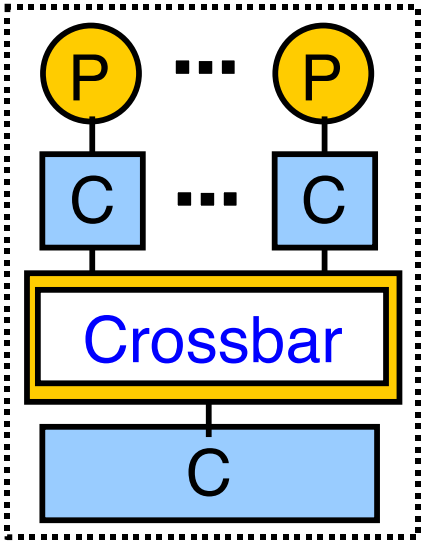
- help analyze dependences, insert synchronization
- produce **MIPS** binaries containing TLS primitives

Benchmarks (all run to completion)

- buk, compress95, ijpeg, equake

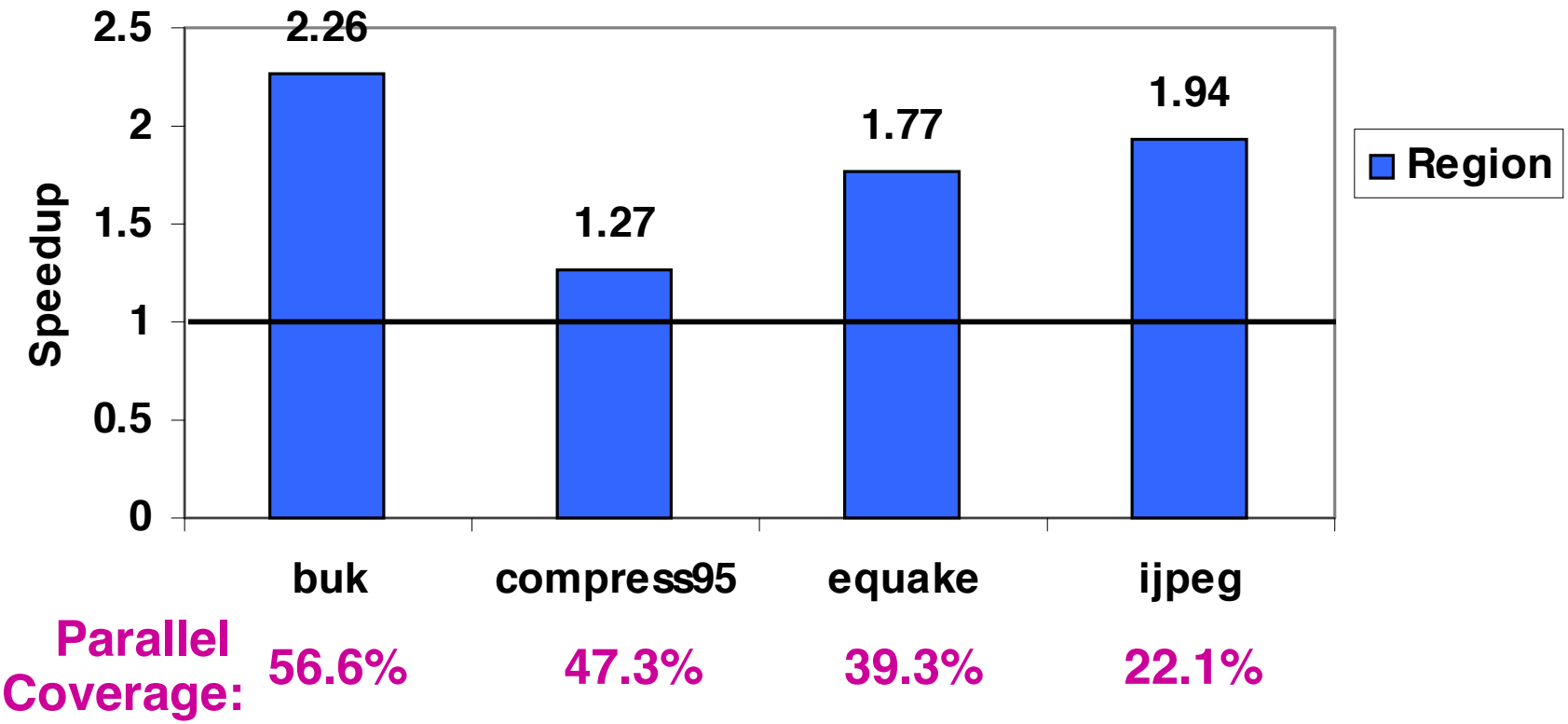
Simulator

- superscalar, similar to **MIPS R10K**
- models all bandwidth and contention

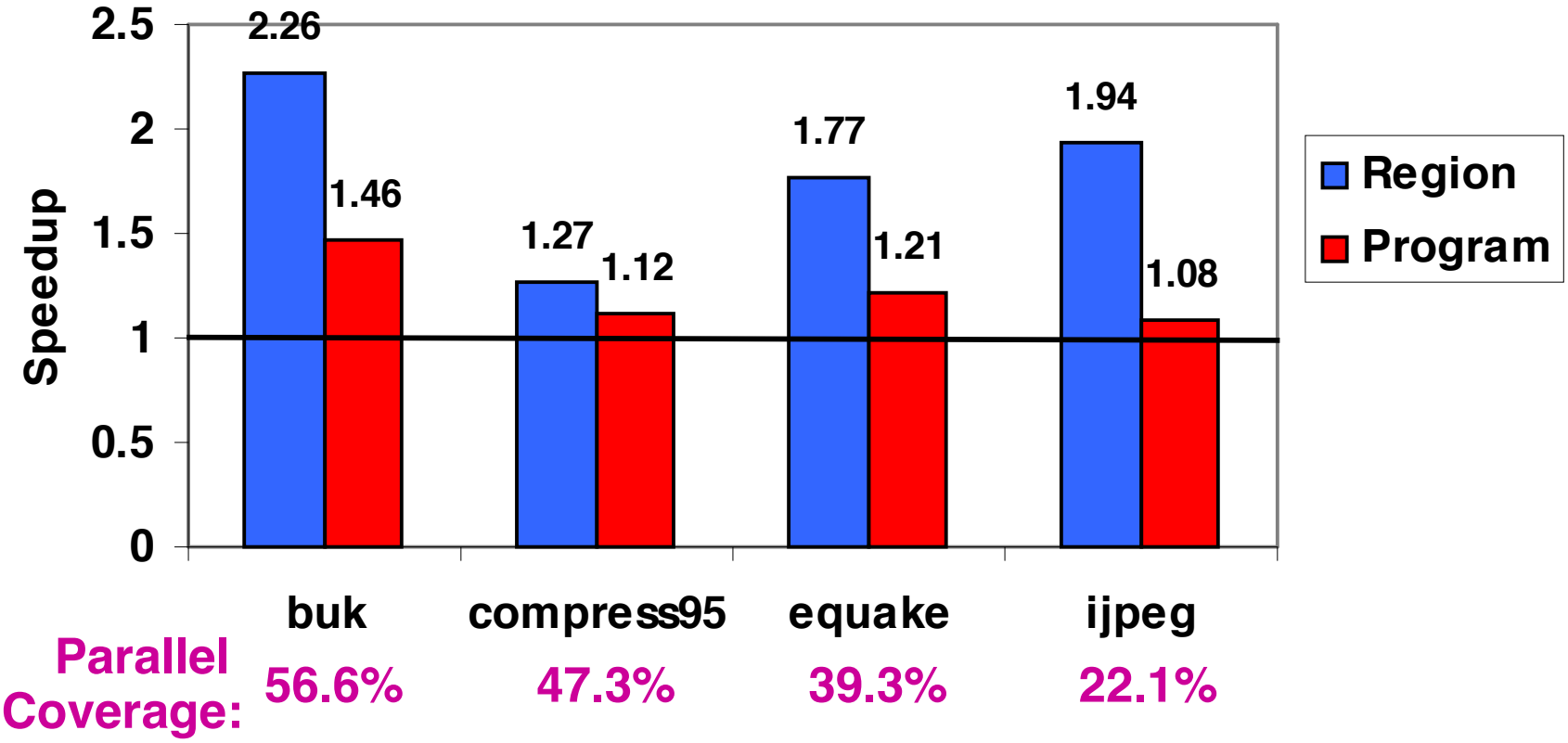


☞ **detailed simulation!**

Performance on a 4-Processor CMP



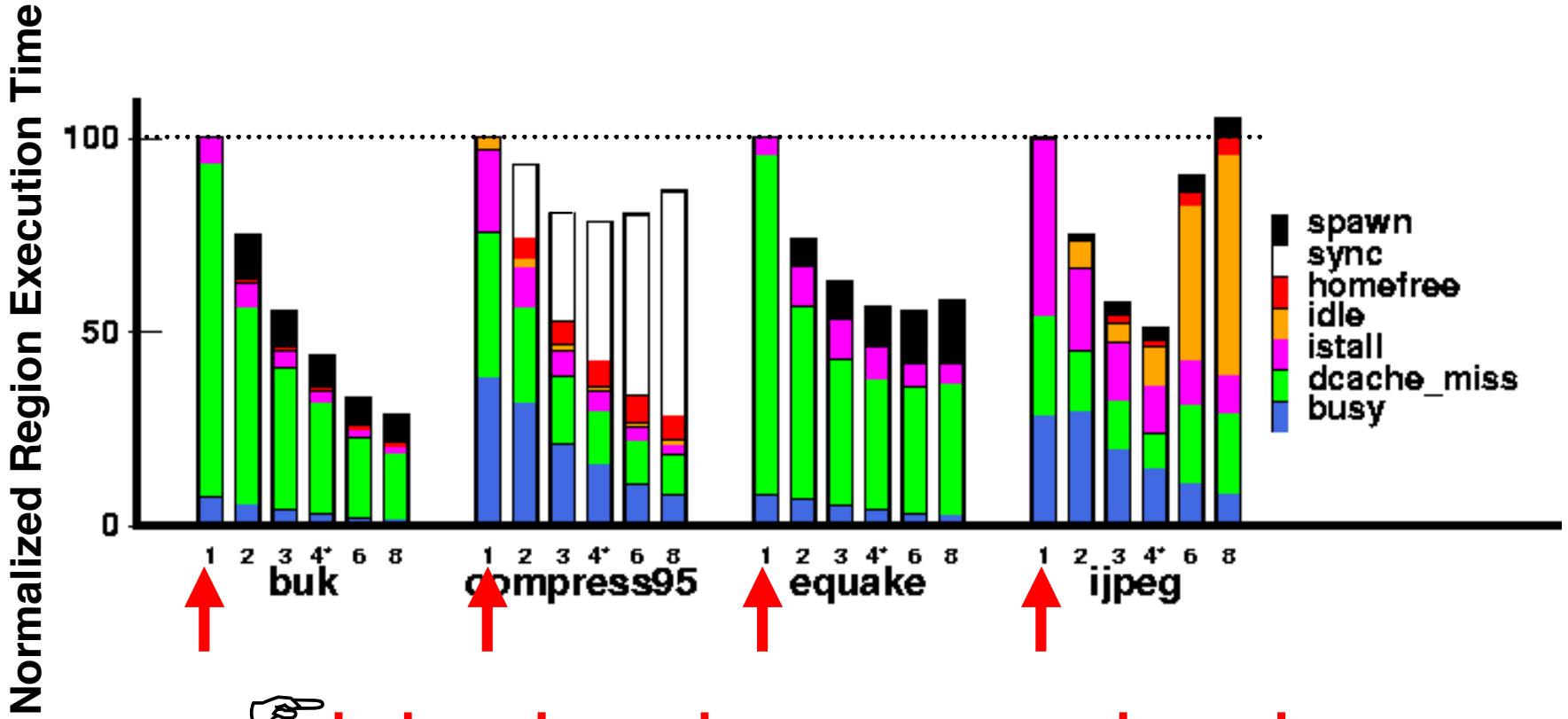
Performance on a 4-Processor CMP



	buk	compress95	equake	ijpeg
Parallel Coverage:	56.6%	47.3%	39.3%	22.1%

☞ **program speedups are limited by coverage**

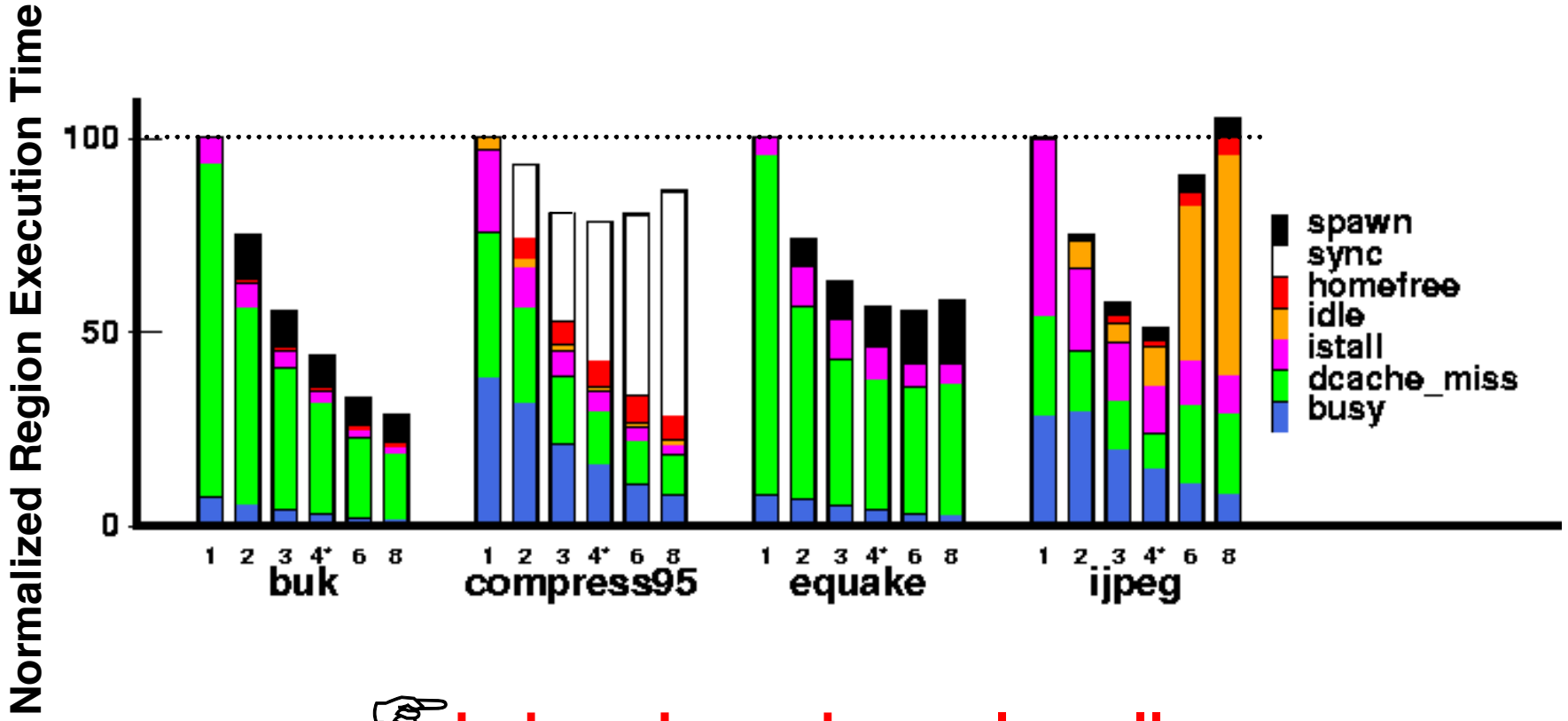
Varying the Number of Processors



☞ buk and equake are memory-bound

☞ compress95 and ijpeg are computation-intensive

Varying the Number of Processors



☞ buk and equake scale well

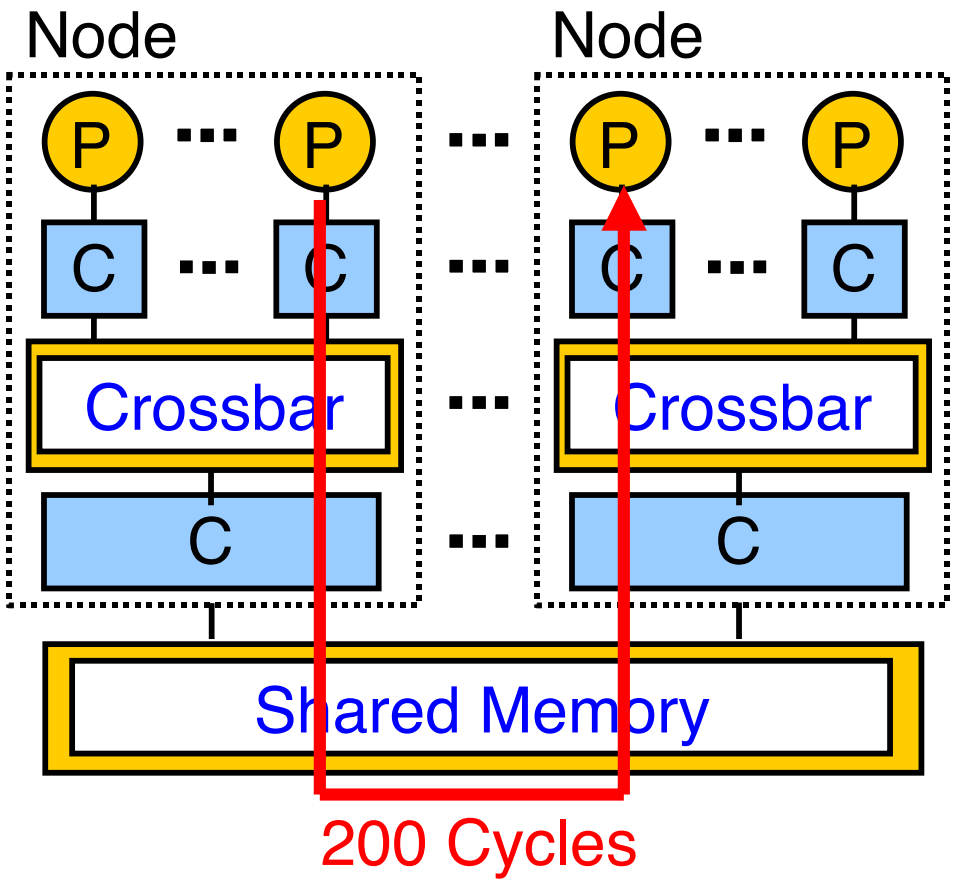
☞ passing the homefree token is not a bottleneck

Performance of the ORB (on a 4-CMP)

Application	Average Flush Latency (cycles)	ORB Size (entries)	
		Average	Maximum
buk	13.95	2.38	9
compress95	0.04	0.01	8
equake	0.13	0.04	12
ijpeg	1.06	0.17	5

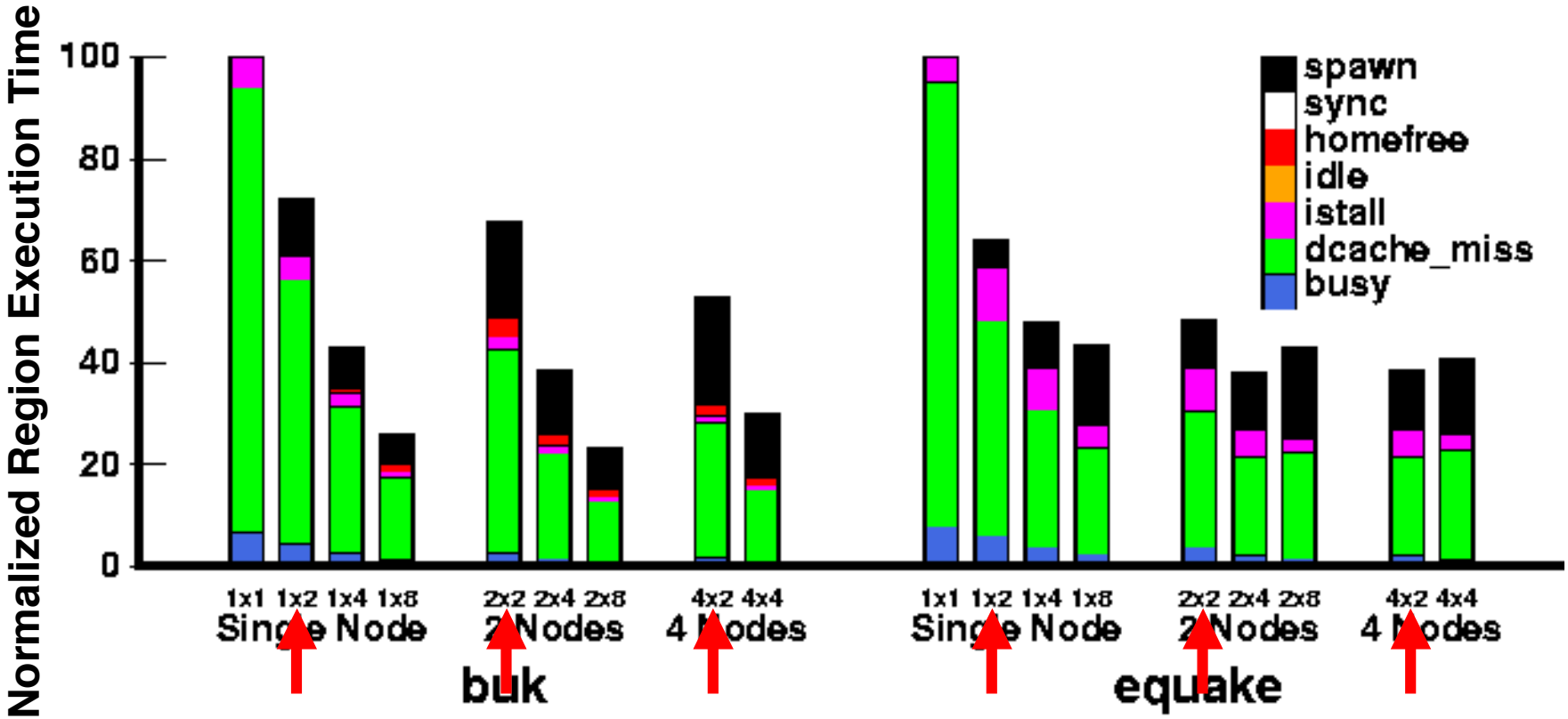
☞ **a small ORB is sufficient**

Scaling Beyond Chip Boundaries



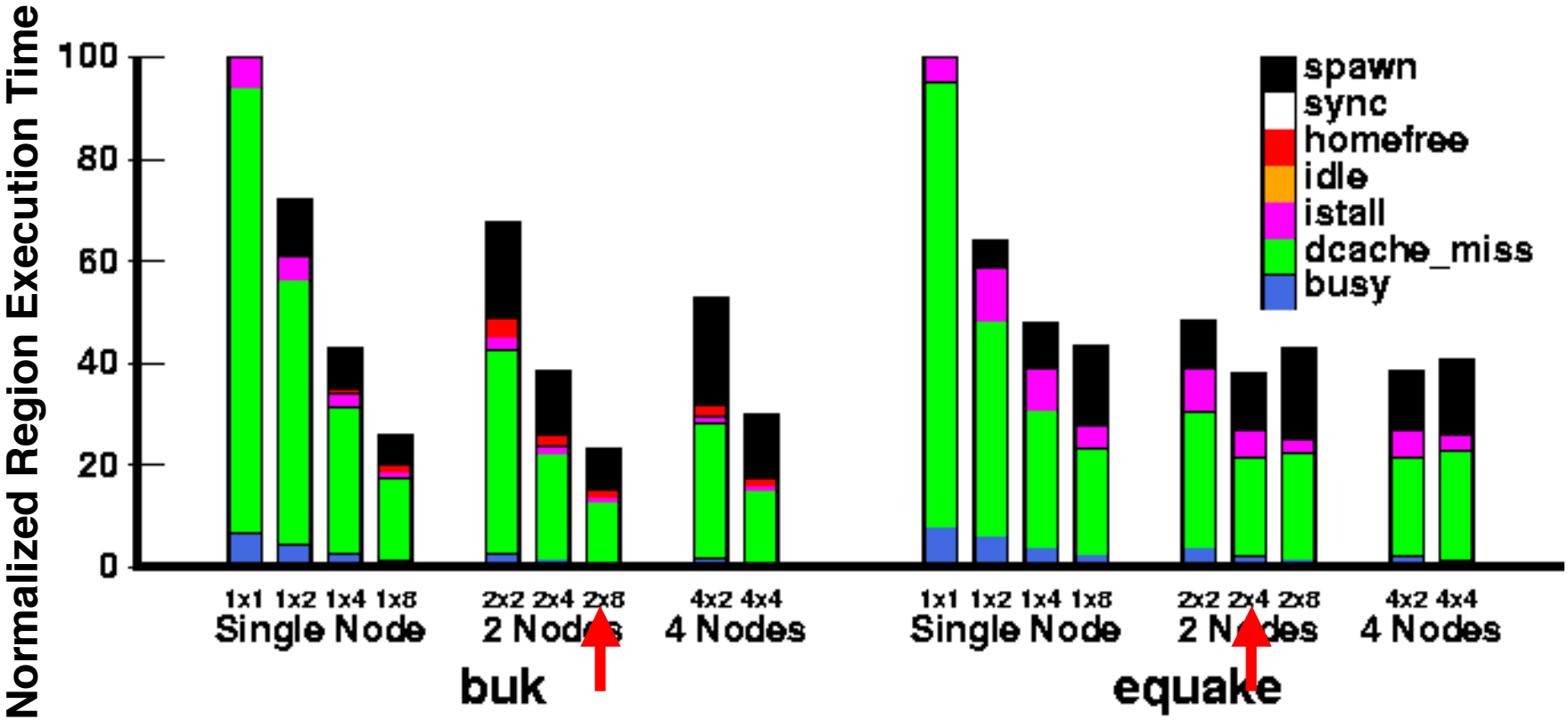
☞ **simulate architectures with 1, 2 and 4 nodes**

Scaling Beyond Chip Boundaries



☞ **multi-chip systems benefit from TLS**

Scaling Beyond Chip Boundaries



☞ **our scheme scales well**

Conclusions

The overheads of our scheme are low:

- mechanisms to squash or commit are not a bottleneck
- per-word speculative state is not always necessary

It offers compelling performance improvements:

- program speedups from 8% to 46% on a 4-processor CMP
- program speedups up to 75% on multi-chip architectures

It is scalable:

- coherence provides elegant data dependence tracking

☞ **seamless TLS on a wide range of architectures**