Dynamic Verification of Cache Coherence Protocols

Jason F. Cantin
Mikko H. Lipasti
James E. Smith

Introduction

- Multiprocessors are used for a variety commercial and mission-critical tasks
- Reliability is a growing concern
- Coherence is a fundamental feature of shared-memory MPs
  - High design complexity
  - Relatively low interconnect reliability

Introduction: Cache Coherence Protocols

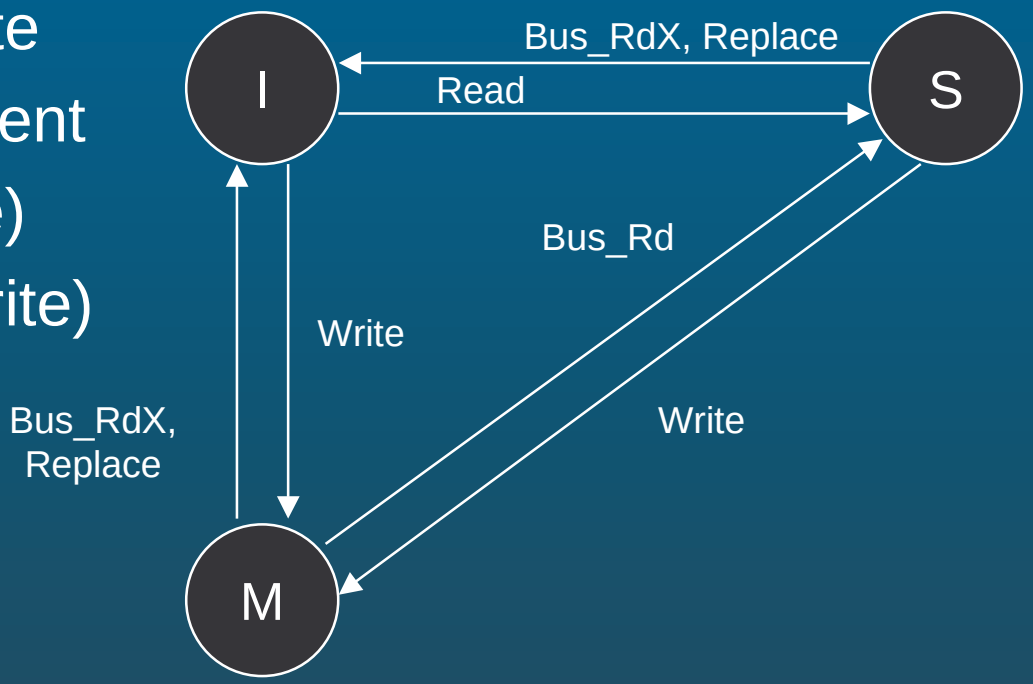
- Notoriously difficult to design and verify
- Often conceptually simple, but with complex implementations for efficiency and handling special cases
- Multiple finite state machines operating concurrently

Introduction: A Simple Example

MSI Protocol

"Architected" State

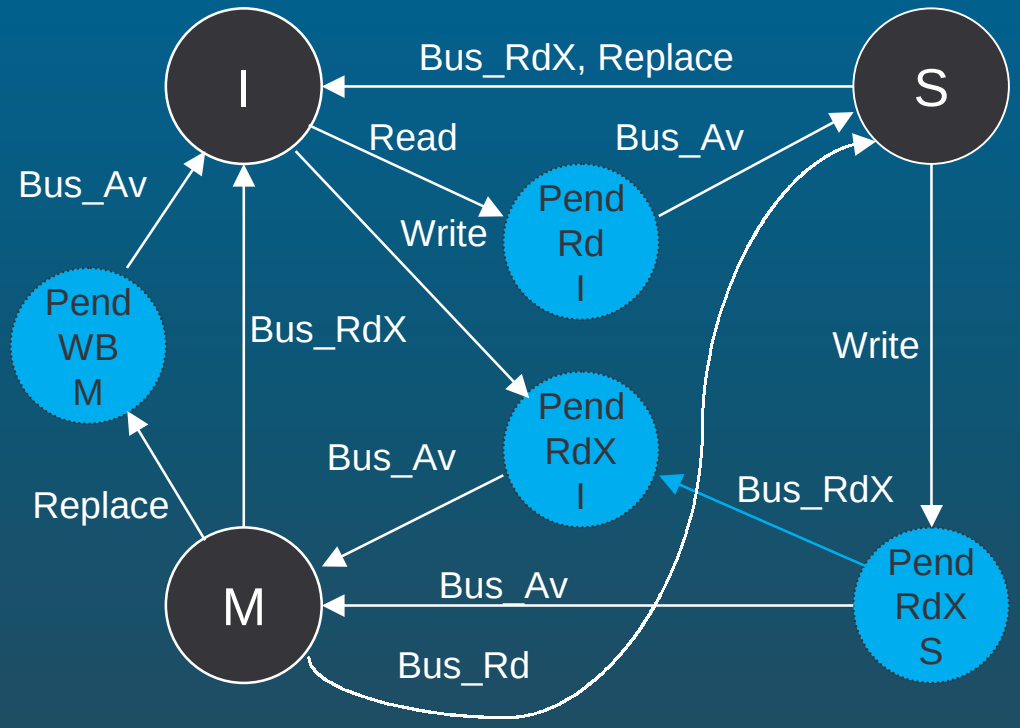
- Invalid / Not Present
- Shared (readable)
- Modified (read/write)



Introduction: Simple Example with a Bus

MSI Protocol "Implementation" State

- Transient states for pending operations
- Arcs to satisfy requests while operations pending



Problem

- In practice, implementations can have dozens of states
  - Atomic memory operations
  - Split transaction buses
  - Protocol optimizations
- Complexity grows exponentially with added states
  - Random testing: Low Coverage
  - Exhaustive testing: Too time consuming

Dynamic Verification

Check the implementation at runtime

- *It is easier to check a computation than to do the actual computation, provided there is a delay between the computation and the check* (Rotenberg, AR-SMT)
- Simplified version of a processor implementation can be used for online verification (Austin, DIVA)

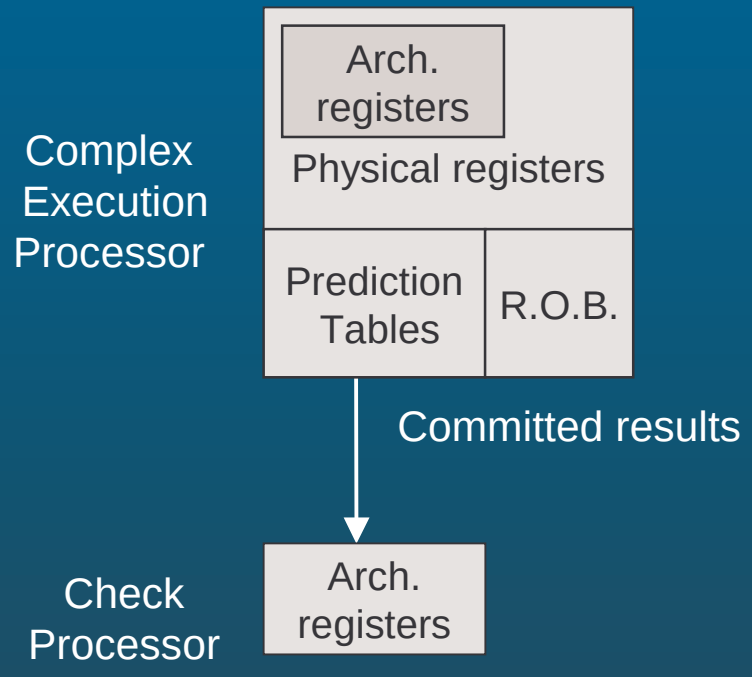
Dynamic Verification of Cache Coherence

A distributed form of dynamic verification for multiprocessor memory systems

- Simplified version of protocol added to each node
  - Maintains *architected* state
- Check completed transitions and actions against simple protocol
- Additional messages (assertions) sent between nodes to ensure coherence

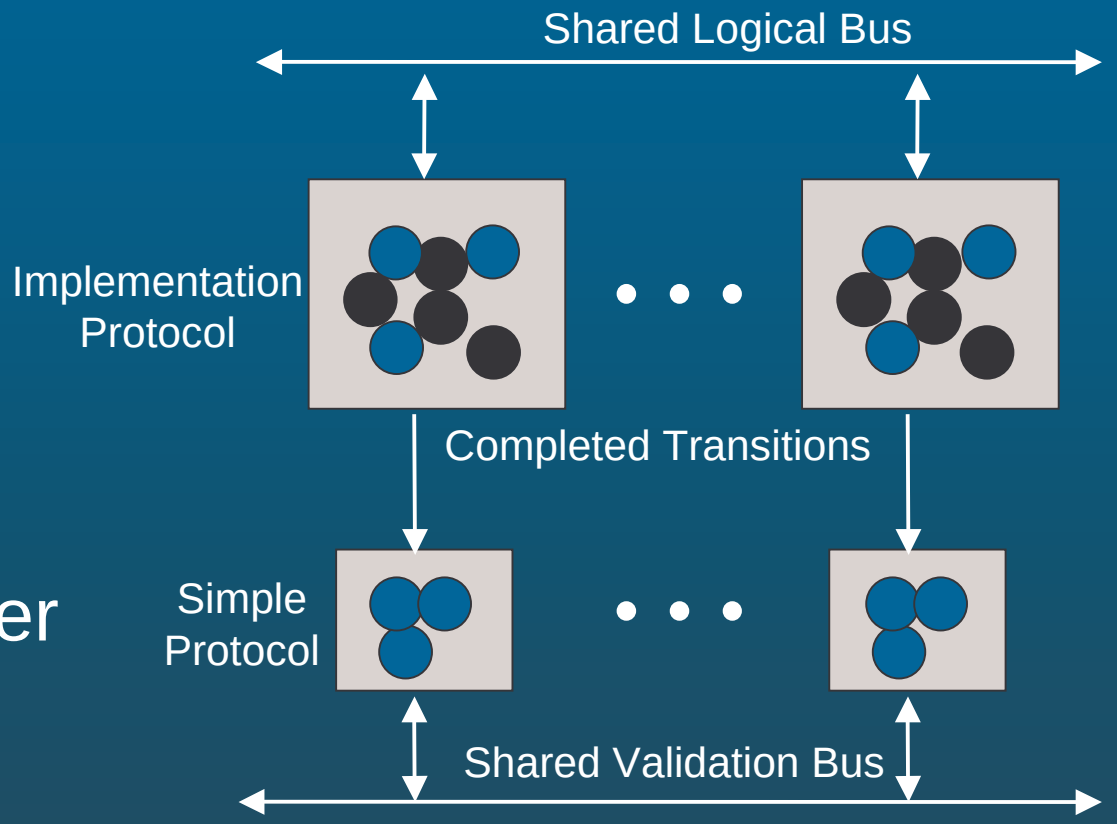
Conceptual View for Superscalar Processors (DIVA)

- Single, centralized check processor
- Receives instructions serially in program order from implementation

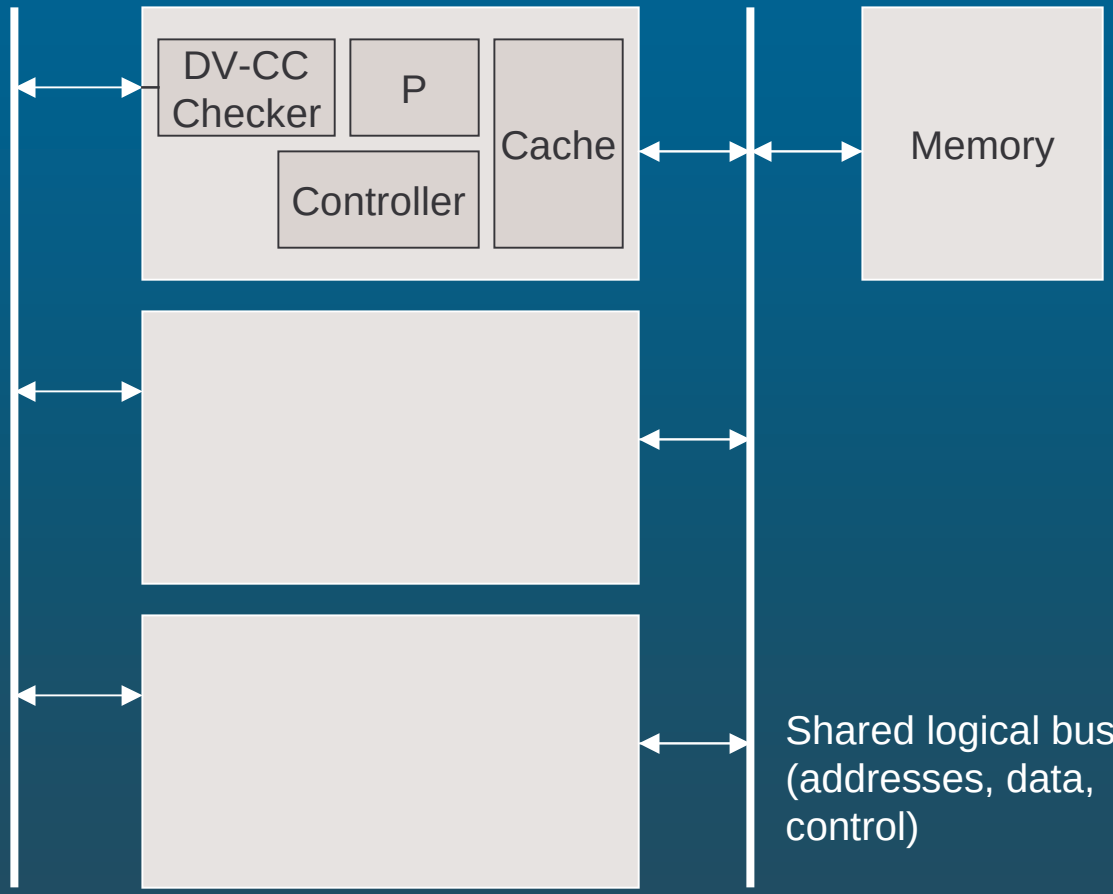


Conceptual View for Coherence

- Distributed checking hardware
- Transitions received in parallel, in completion order



High Level Organization



Benefits

- Detects hardware faults
  - Redundant computation
  - Including intermittent network failures
- Detects design mistakes
  - Checker is simple and easy to verify

Drawbacks

- Time is required for checking, but…
  - May be overlapped with other activities
  - Simple protocol requires fewer transitions
- Assertions consume bandwidth
  - May need second bus / network
- Additional hardware
  - But not much

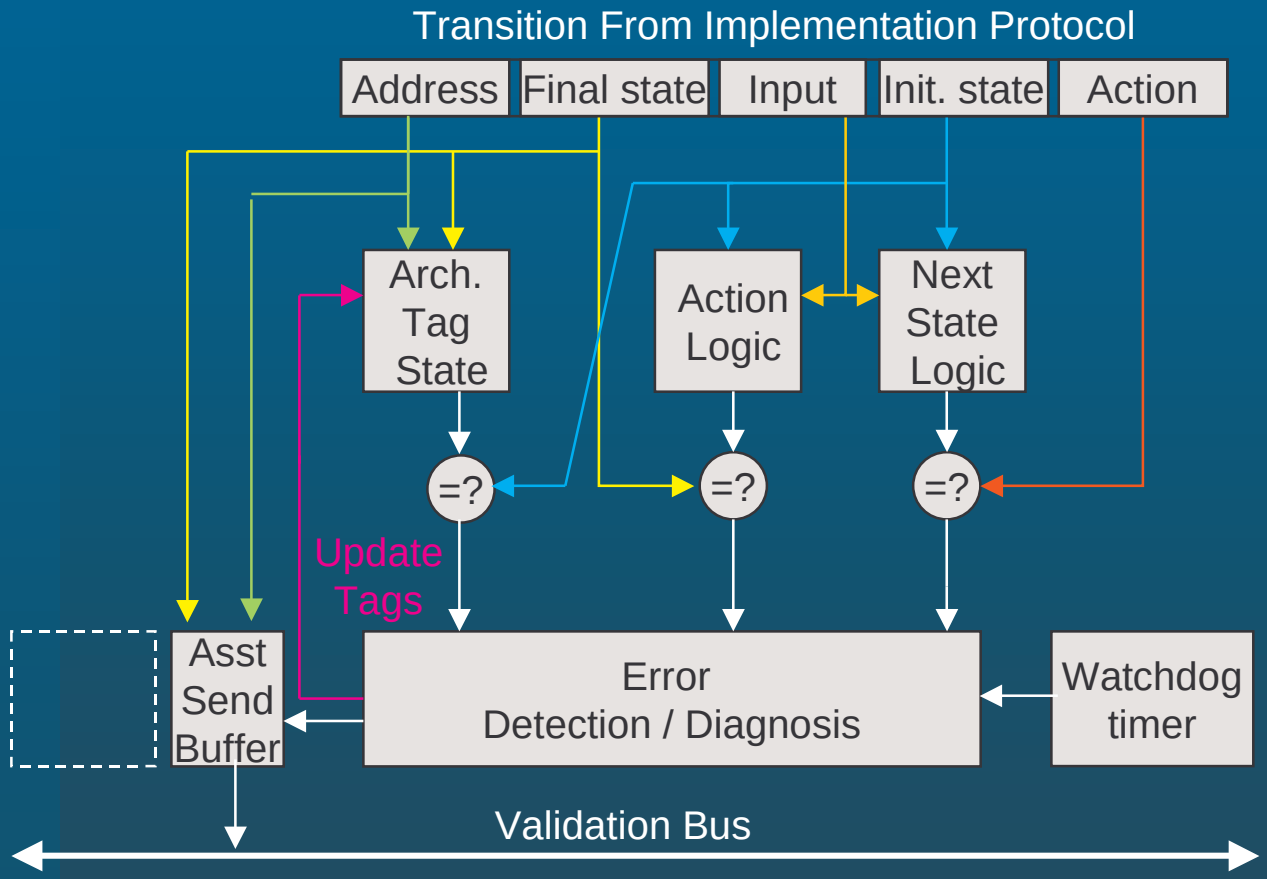
DV for coherence in an SMP

- Architected state stored in a second tag array
- Transactions sent to the checker when architected state changes
  - Address
  - Initial State and Final States
  - Input (Request, Snoop Responses, etc)
  - Action (Send Data, Respond Shared, etc)

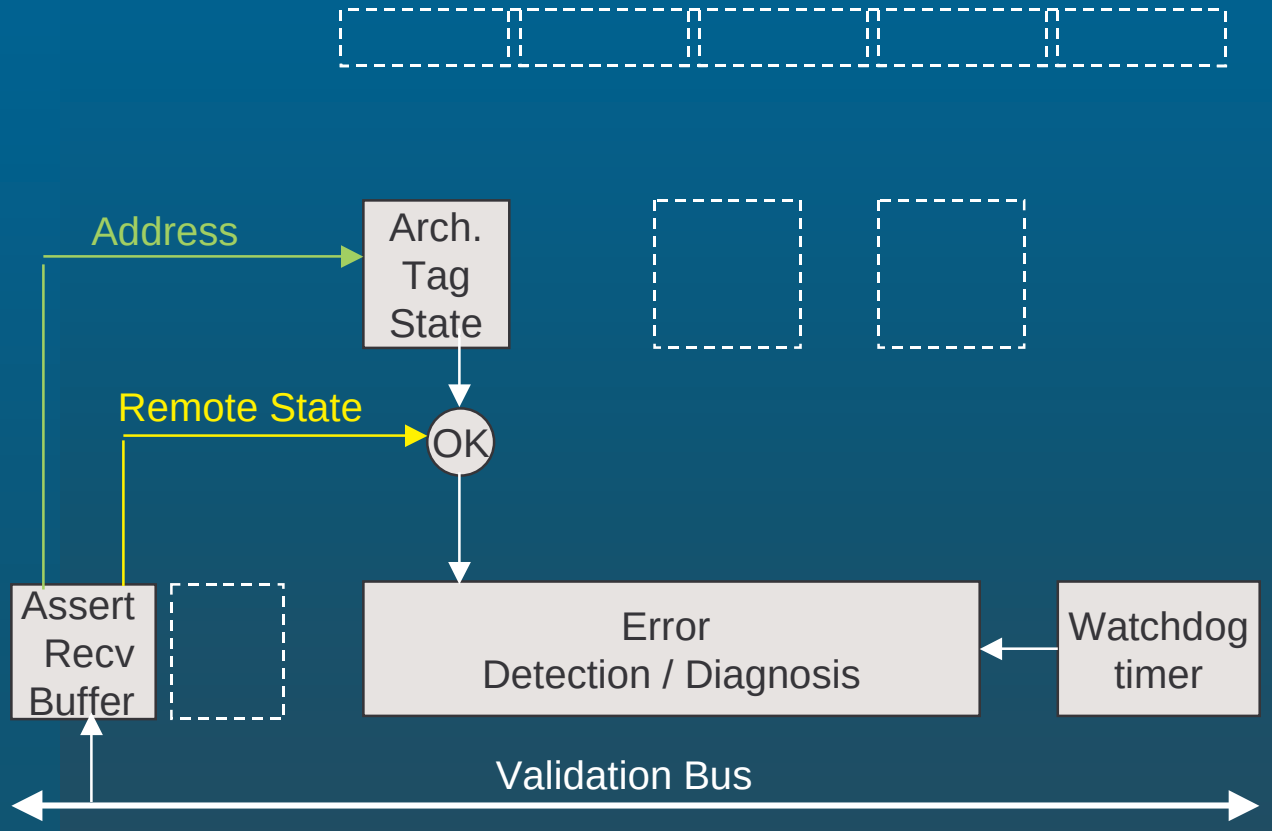
DV for coherence in an SMP (2)

- Checker compares the initial state of a transition against the architected state
- Final state and action recomputed and compared to implementation's result
- Assertions broadcast to other nodes to check coherence and confirm completion of transactions
- Watchdog timer detects deadlock, livelock, and other omission failures

Checking a State Transition



Checking an Assertion



When to Broadcast Assertions

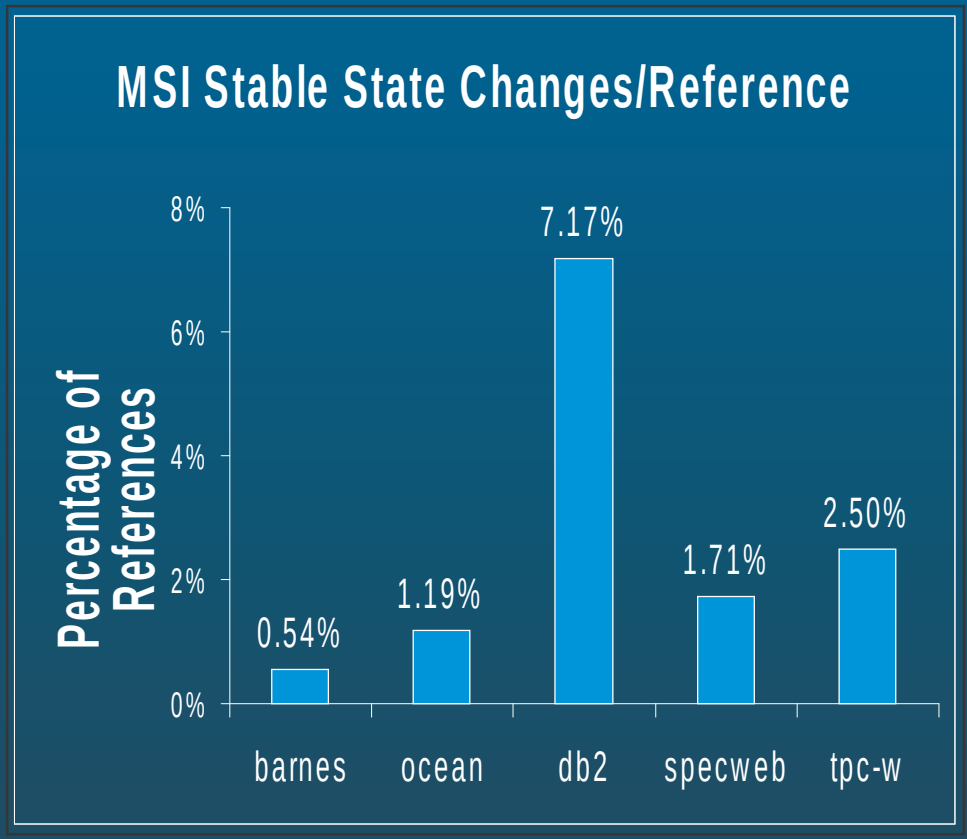
For MSI:

1. I → S (readable copy loaded)
2. I → M (writeable copy loaded)
3. S → M (upgrade)
4. M → I (writeback)

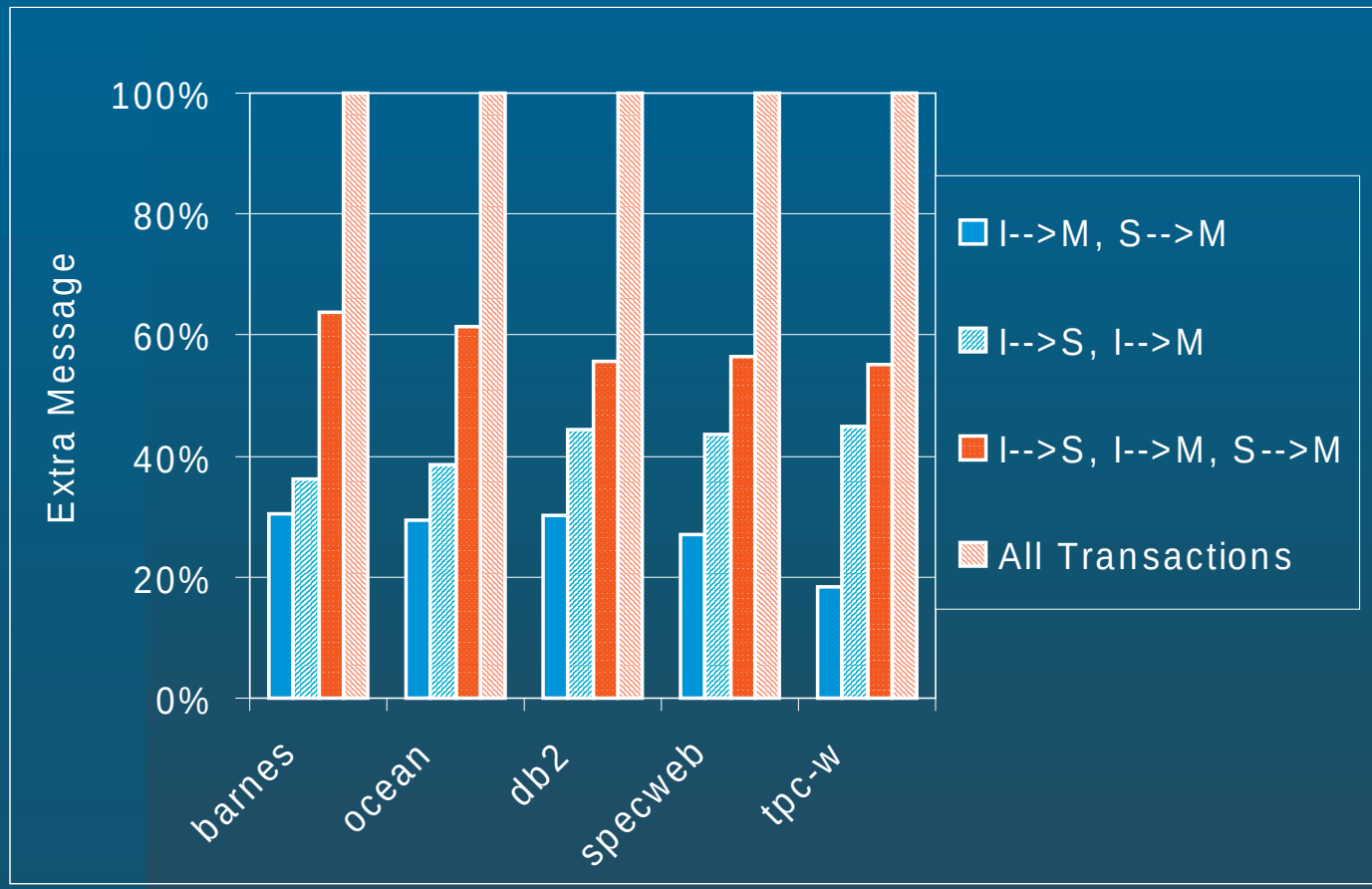
Note: The M→S transition results from remote reads, and doesn't require an extra assertion. Replacements (S→I) are not considered here.

Preliminary Data (4-way SMP)

Most memory references do not change cache state (checker need not have high bandwidth)



Preliminary Data (4-way SMP)



Future Work

- Performance impact for a real SMP protocol implementation
  - In progress
- Directory-based protocols
- Dynamically verifying memory models
- Recovery
  - Can stall to avoid error propagation
  - Can write checkpoints periodically

In Summary

- Dynamic verification can be applied to multiprocessor systems (in a distributed manner)
- Improves fault-tolerance, and design verification may be relaxed
- More to come