Lamport Clocks: Verifying A Directory Cache-Coherence Protocol*

Manoj Plakal, Daniel J. Sorin, Anne E. Condon, Mark D. Hill

Computer Sciences Department
University of Wisconsin-Madison
`{plakal,sorin,condon,markhill}@cs.wisc.edu`

*. This work is supported in part by Wright Laboratory Avionics Directorate, Air Force Material Command, USAF, under grant #F33615-94-1-1525 and ARPA order no. B550, National Science Foundation with grants MIP-9225097, MIPS-9625558, CCR 9257241, and CDA-9623632, a Wisconsin Romnes Fellowship, and donations from Sun Microsystems.

Motivation & Problem

- Shared-memory *multiprocessors* are important
  - Current servers: computation, databases, files, mail & web
  - Future clients

- Shared-memory multiprocessor implementations must:
  - *Be correct*
  - *Have high performance*
  - Doing both is hard!

Motivation & Problem (contd.)

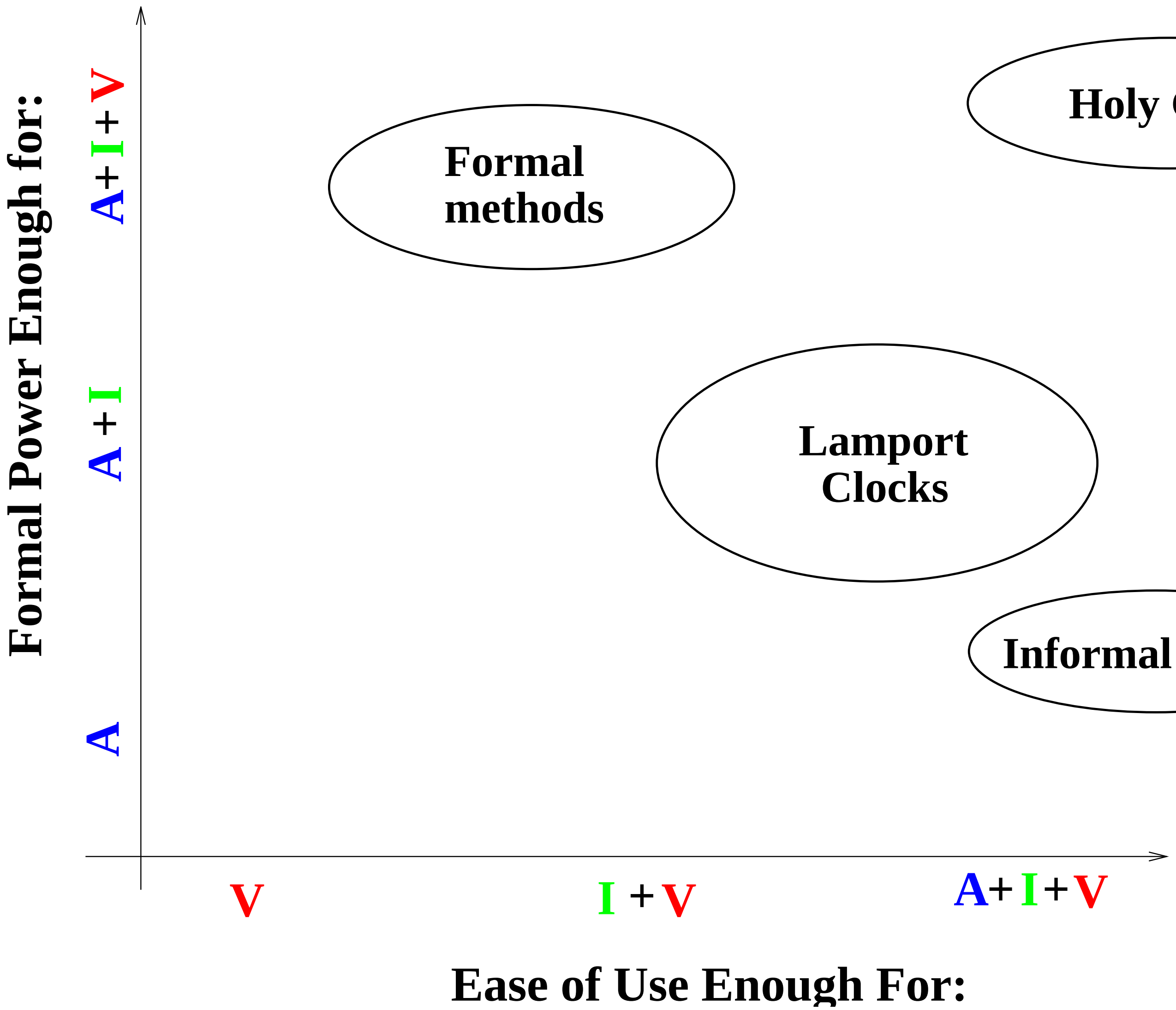
- *Correct == Implementing a memory consistency model*
  - Most memory consistency models define correctness through (off-line) existence of a total/partial order on memory references
  - E.g., Sequential Consistency (SC) requires a total order

- Modern high-performance implementations:
  - Cache coherence (e.g., snooping, directories)
  - Very aggressive (e.g., out-of-order processors, hierarchies of non-blocking caches, interleaved memories)
  - Optimizations $\Rightarrow$ memory operations re-ordered/non-atomic
  - Existence of required order is not evident

Our Proposal: Use Lamport Clocks

- Sequential Consistency requires (off-line) existence of total order
- Borrow from Lamport's logical clocks:
  - Assign timestamps to memory references in any execution
  - Prove that every load returns value written by previous store to same address in timestamp ordering
  - No hardware added!
- Since proof works for orderings created by any execution:
  - Every execution satisfies SC
  - *Therefore, system satisfies SC*

Where Our Scheme Fits In



A = Architects

I = Implementors

V = Verifiers

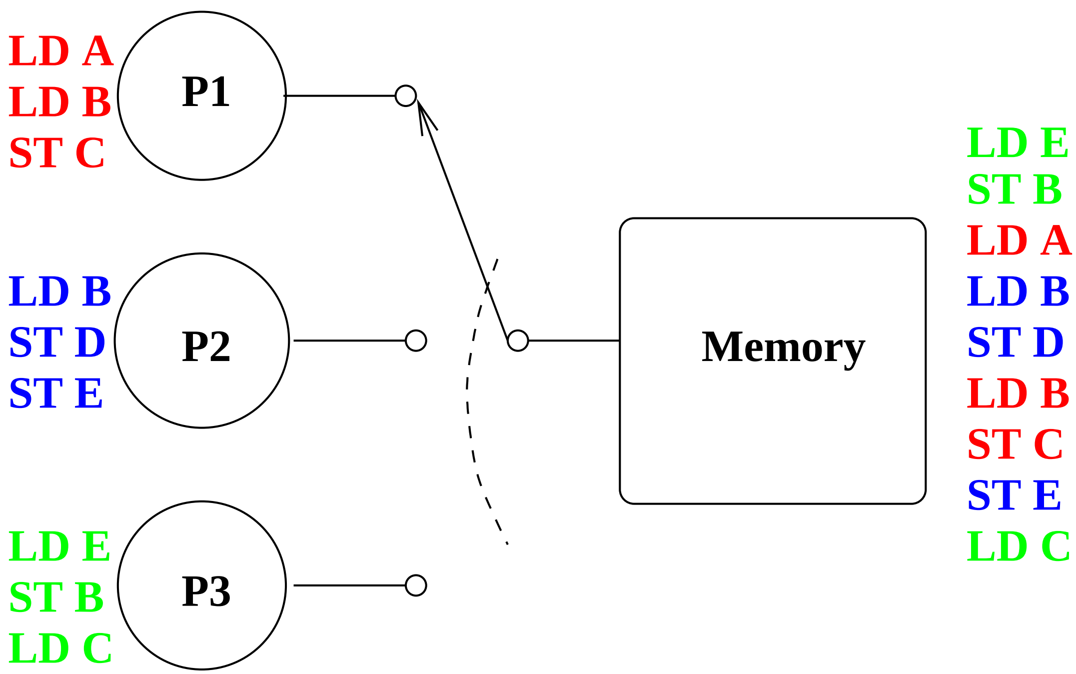
Outline

- Motivation, Problem & Summary
- *Background*
  - *Sequential Consistency (SC)*
  - *Implementing Cache-Coherent Multiprocessors (Directories)*
  - *Lamport Clocks and how we use them*
- Our Verification Technique
- Summary

Sequential Consistency (SC)

- SC must insure that [Lamport '79]:

    the result of any execution is the same as if the operations of all the processors were executed in some sequential order, and the operations of each individual processor appear in this sequence in the order specified by its program



- (Like a multiprogrammed uniprocessor)

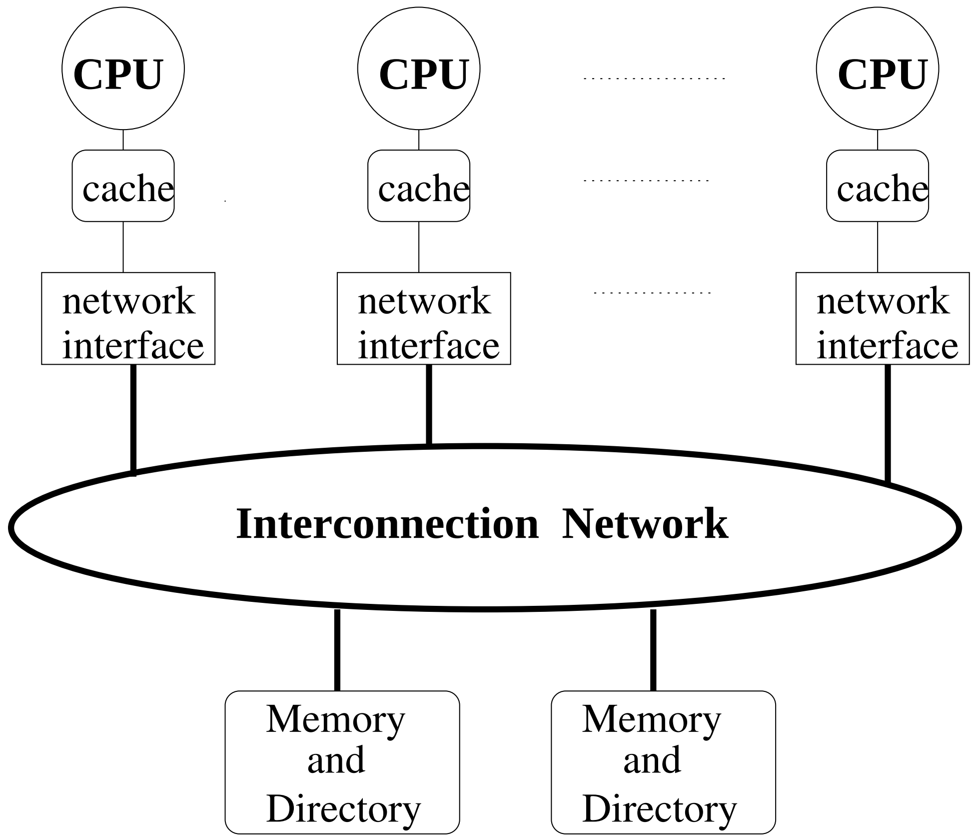
Sequential Consistency (contd.)

- For each execution
  - *There exists a total order*
  - That respects the *program order* of each processor
  - Loads return the *value of the last store* in that order (to the same address)

- Why not reason about correctness by:
  (1) Think of every dynamic load/store getting a timestamp
  (2) Have memory hand out the timestamps
  (3) Show every load returns an appropriate value

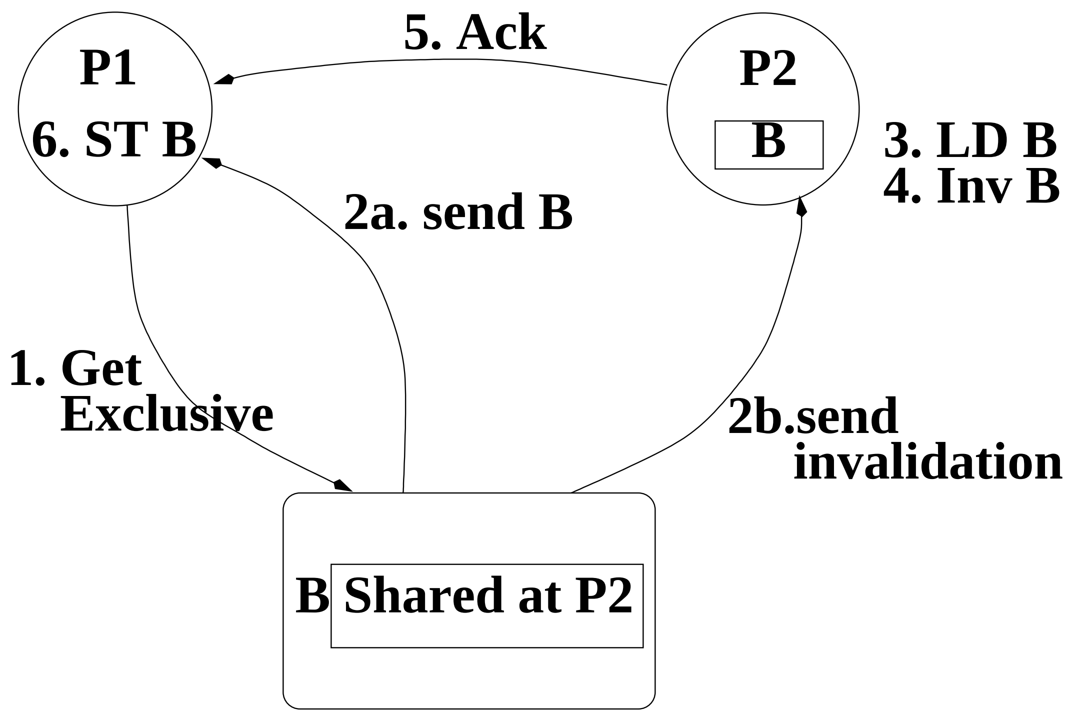
- Can do (1) and (3), *but real implementations can't do (2)*
  - Don't have a single physical memory
  - Often have distributed coherent caches

Implementing Cache-Coherent Multiprocessors



- Most implementations use:
  - Caches (**invalid**, read-only **shared**, or writable **exclusive**)
  - Cache coherence protocols (snooping or directories)

Directory Coherence Protocol Example

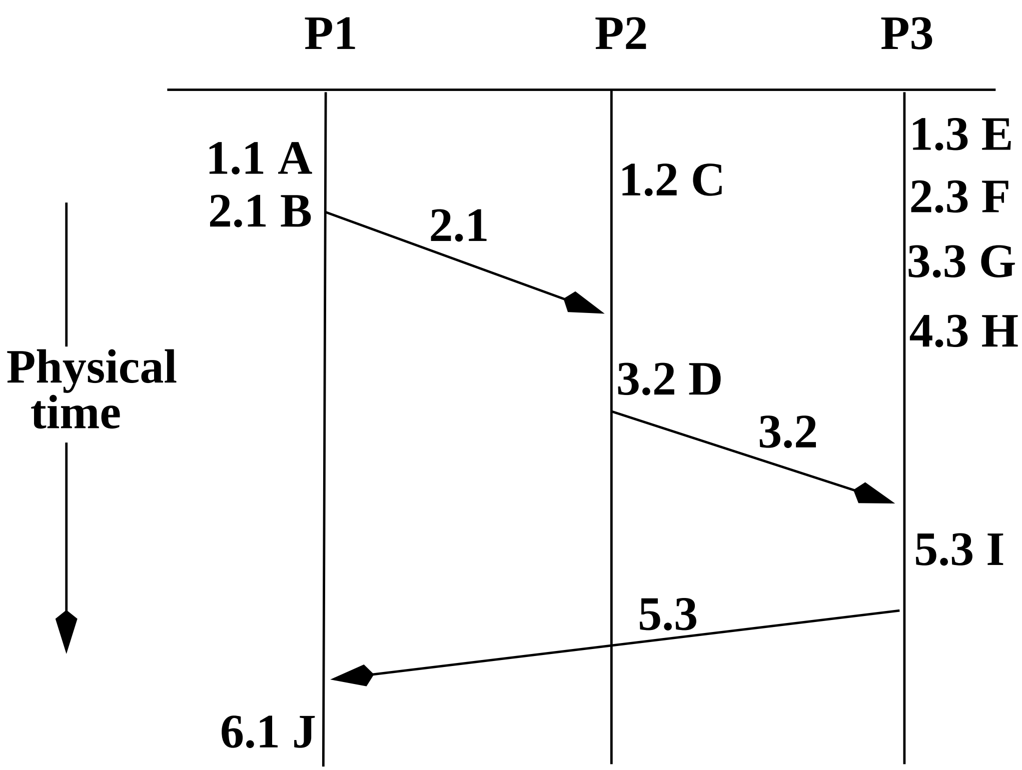


A Directory Protocol in action

- Events at a processor:
  - Memory operations (LD, ST)
  - Coherence transactions (Get-Exclusive, Invalidate)

Lamport Clocks (CACM, 1978)

- Problem: notion of “happens before” in a distributed system
- Solution: Use logical clocks (counters) at each node
- Update clocks respecting causality and real-time local order



Logical order

1.1 A
1.2 C
1.3 E
2.1 B
2.3 F
3.2 D
3.3 G
4.3 H
5.3 I
6.1 J

Outline

- Motivation, Problem & Summary
- Background
- *Our Verification Technique*
  - *Our extensions to Lamport's solution*
  - *Applying our technique*
  - *Case Study*
- Summary

Comparing Our Solution to Lamport's Solution

	Lamport's Solution	**Our Solution**
Logical Clock	2-tuple	3-tuple
Timestamp	Messages	Memory operations and Coherence transactions
Increment	Local Events (in real-time order)	Loads/Stores (in program order)
Update	Message Receives	Coherence Message Receives
Break Ties	Node ID	Node ID

Timestamps used by our scheme

- Timestamps:
  - Assigned to coherence transactions and loads/stores
  - Rules specific to implementation
  - Respect program order
  - Respect coherence protocol

- Timestamp format = **<Global.Local.Node-ID>**
  - Coherence transactions update global component
  - Loads/stores update local component
  - Global-Local distinction provides intuition to architects

Application: Levels of Formality

- *Apply in a paper-and-pencil way to examine ideas:*
  - Deferred invalidation (Scheurich’s optimization)
  - Out of order coherence transactions
  - Parallel SMP buses (Sun E10000)

- *Can do detailed proofs:*
  - SC on SGI Origin 2000-like directory protocol (SPAA’98)
  - SC on Sun E6000-like SMP protocol
    - [UW CS-TR-1367, 3/98] (submitted for publication)
    - URL: **ftp://ftp.cs.wisc.edu/wwt/tr98_lamport.ps**
    - Proof very similar to directory protocol

A (brief) Case Study: A Directory Protocol

- SC system using an SGI Origin 2000-like directory protocol:
  - Processors have one level of cache
  - Single-writer, multiple-reader, invalidation-based protocol
- Abstraction level of model:
  - Internals of processor/interconnect not modeled
  - Real-time ordering assumptions about processor behavior
- Specification of implementation:
  - Informal text description of protocol
  - Possible to convert to state-transition tables (for automation)

Case Study (contd)

- How we timestamp Loads/Stores:
  - Loads/Stores are *bound* to the coherence transactions that ensured the appropriate coherence state
  - Rules:

Global time = max(global time of transaction to which load/store is bound,
global time of previous load/store in program order)

Local time = 1 + local time of previous load/store in program order with same global time,
1, if this is the first.

Case Study (contd)

- Example of timestamping at a single node:

Physical time		Lamport time
GETX A	1.0.1	GETX A
LD A	1.1.1	LD A
GETS B	1.2.1	ST A
ST A	2.0.1	GETS B
LD B	2.1.1	LD B

Case Study (contd)

- Example of timestamping across nodes:

Physical time

N_1	N_2
send GETX A	store B
	bind load A
	recv INV A send ack
recv ACK	perform load A, invalidate A
store A	

Lamport time

	N_1	N_2
1.10.2		store B
1.11.2		load A
2.0.2		invalidate A, send ack
3.0.1	recv ACK	
3.1.1	store A	

Alternative Methods

- Informal Techniques:
  - Extensive simulation and stress testing
  - Thought experiments (a.k.a. hand-waving)
- Formal Techniques:
  - State-space search of finite-state coherence engines
  - Verification using theorem-provers
  - Precise, but not scalable to practical systems
- Current promising research in formal systems:
  - Symbolic states [Pong & Dubois, SPAA'93]
  - Aggregation of transactions [Park & Dill, SPAA'96]
  - Term rewriting [Shen & Arvind]

Summary

- Memory consistency model requires (off-line) existence of order
- Construct order on-line by assigning *timestamps*
- Prove requirements of model in constructed order
- Since proof works for any execution:
  - Every execution is correct
  - *Therefore, system is correct*
- In our view
  - Ease of use: Formal Verification < Lamport < Informal
  - Formal Power: Informal < Lamport < Formal Verification

Future Work

- Other memory consistency models (PC, TSO, RC, RMO, Alpha)
- Other memory systems (clusters of SMPs)
- Handling deadlock, starvation, & live-lock
- A good way of formally specifying protocols
- Automating the whole process (thereby losing our jobs)

Directory Protocol Proof Sketch, 1 of 2

- Use *Claims*, *Lemmas*, & *Main Theorem* for modular proof
- *Claims* about ordering induced by timestamps (in Lamport time)
  - *Messages are always sent to all nodes affected by a coherence epoch end/begin*
  - For block B, real-time order of coherence epochs at each node matches the order at block B's directory entry
  - Load/store gets timestamps before epoch "end" or after epoch "begin"
  - Load/store bound to most-recent epoch for block B at P1
- *Lemmas* about coherence epoch invariants (in Lamport time)
  - *Block B's epochs have appropriate non-overlap*
  - Loads/stores timestamped with appropriate epoch
  - Load value from coherence transaction or "recent" stores

Directory Protocol Proof Sketch, 2 of 2

- *Main Theorem*: *Every load returns appropriate value*
  - Initial value of memory
  - Value of last store for same address

- Main Proof
  - Consider any load
  - Value it returns should be from the “last” store
  - Last store is bound to same coherence transaction as load?
  - Show both “yes” and “no” work correctly

- Since proof works for any execution
  - Every execution satisfies SC
  - *System satisfies SC*