Lamport Clocks: Verifying A Directory Cache-Coherence Protocol*

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

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

[Appears in SPAA ´98]

*. 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
  - Servers: computation, databases, files, mail & web
  - Future clients
  - Small & medium symmetric multiprocessors (SMPs)
  - Large cache-coherent non-uniform memory access (CC-NUMAs) machines (e.g., SGI Origin 2000)

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

Motivation & Problem, cont.

- Modern high-performance implementations:
  - Very aggressive: out-of-order processors, hierarchies of non-blocking caches, interleaved memories
  - Existence of required order is not evident

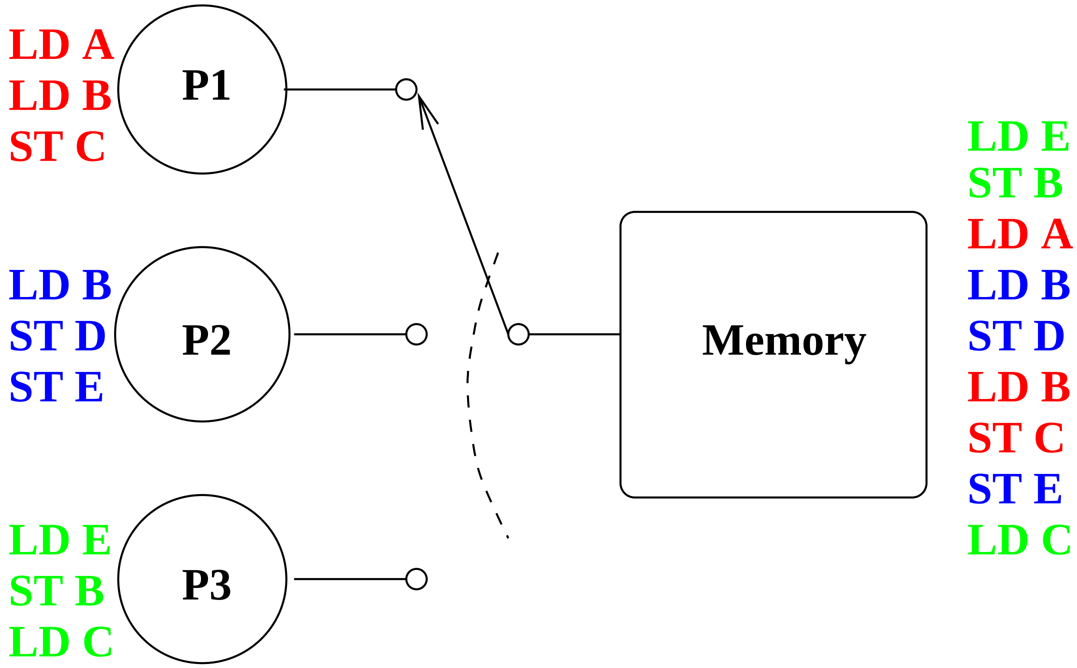
- *Correct = Implementing a memory consistency model*
  - Most memory consistency models define correctness through (off-line) existence of a total/partial order on memory references
  - e.g., Sequential Consistency (SC) requires a total order

Our Proposal: Lamport Clocks

- Memory consistency models require (off-line) existence of order
- Borrow from Lamport's logical clocks:
  - Assign timestamps to memory references
  - Timestamps create *on-line* ordering among references
  - Prove that requirements of memory model are satisfied in this constructed ordering
  - Timestamps are *not in hardware* - they are reasoning tool
- Since proof works for orderings created by any execution:
  - Every execution is correct => *Therefore, system is correct*
- Current proofs for SC
  - SPAA ´98: ftp://ftp.cs.wisc.edu/wwt/spaa98_lamport.ps
  - UW TR ´98: ftp://ftp.cs.wisc.edu/wwt/tr98_lamport.ps

Sequential Consistency (SC)

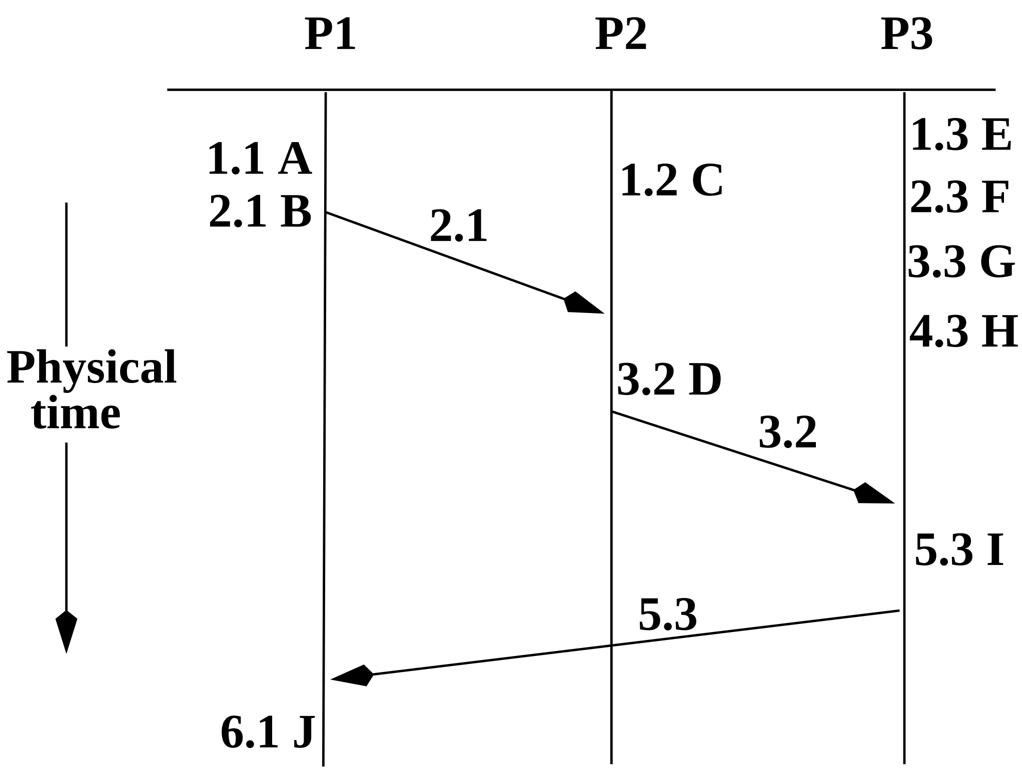
- For each execution, *there exists a total order:*
  - This order respects the *program order* of each processor
  - Loads return the *value of the last store* in that order



- (Like a multiprogrammed uniprocessor)

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

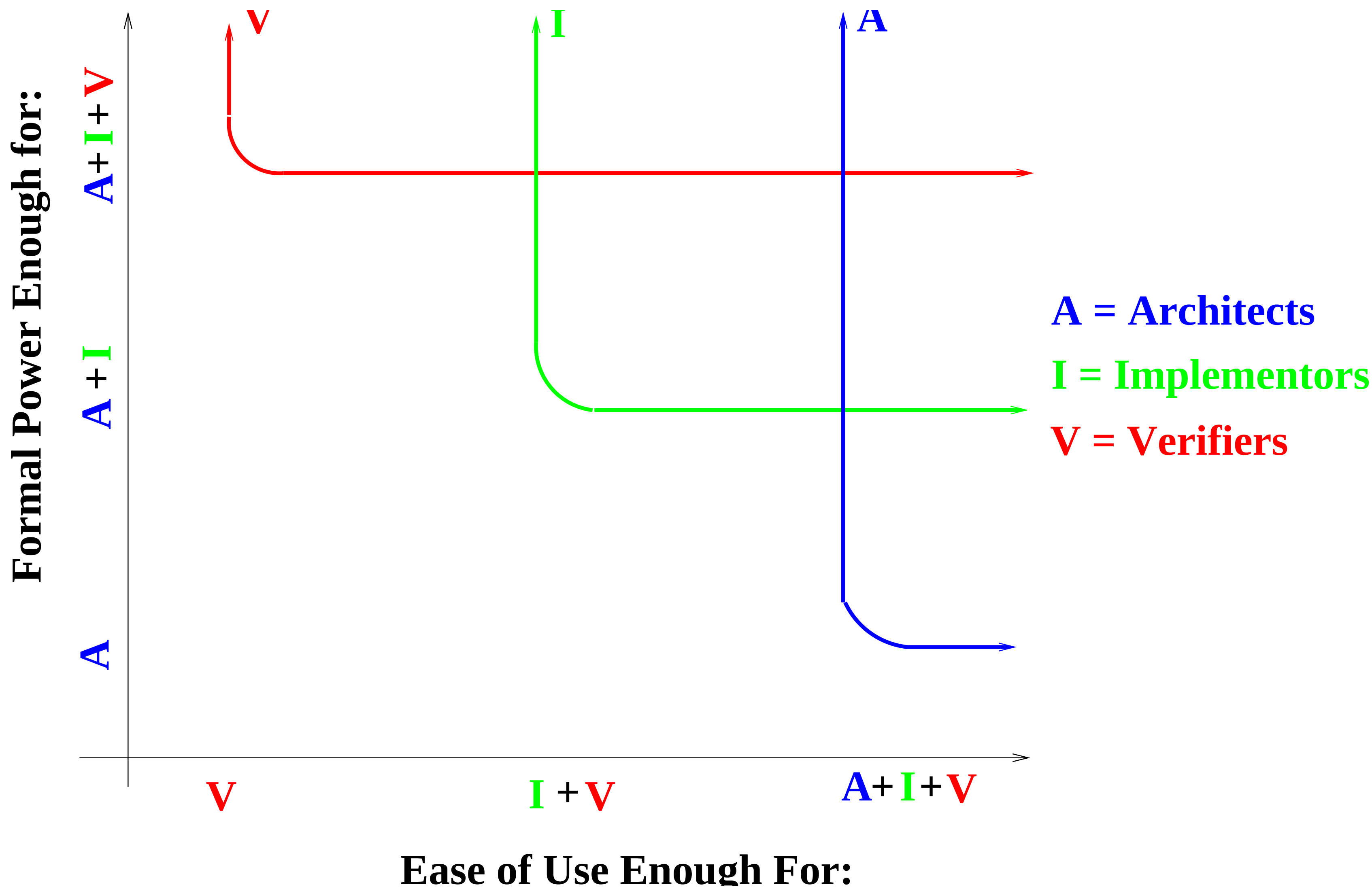
Comparing Our Solution to Lamport's Solution

	Lamport's Solution	**Our Solution**
Logical Clock	Per Node	Per Node, 2 components
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

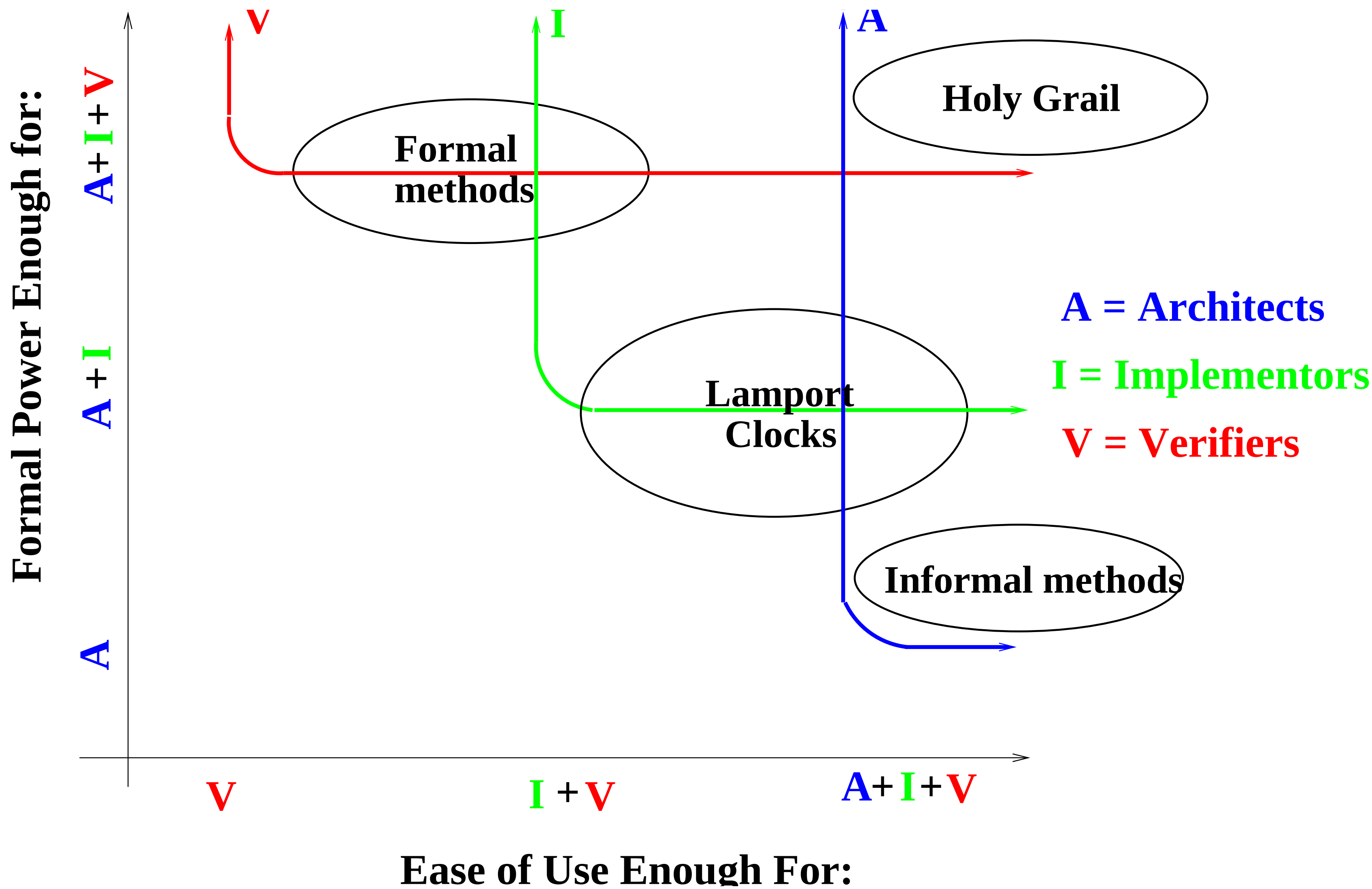
Lamport Clock Details

- Coherence messages delimit per-processor & per-block *coherence epochs*
- Processors *timestamp* loads/stores in program order with appropriate coherence permissions
- Load/store timestamps **Global . Local . Node-id** = *MAXIMUM*
  - **G.(L+1).P1** (cache hit where **G.L.P1** was P1's last load/store)
  - **H.1.P1** (cache miss/prefetch bound to **H**)

Where Our Scheme Fits In



Where Our Scheme Fits In (contd.)



Summary and Future Work

Summary

- Memory consistency model requires (off-line) existence of order
- Construct order on-line by assigning timestamps
- Prove requirements of model in constructed order

Future Work

- Other memory consistency models (PC, TSO, RC, RMO, Alpha)
- Other memory systems (clusters of SMPs)
- Handling deadlock, starvation, and live-lock
- Automating the whole process (thereby losing our jobs)