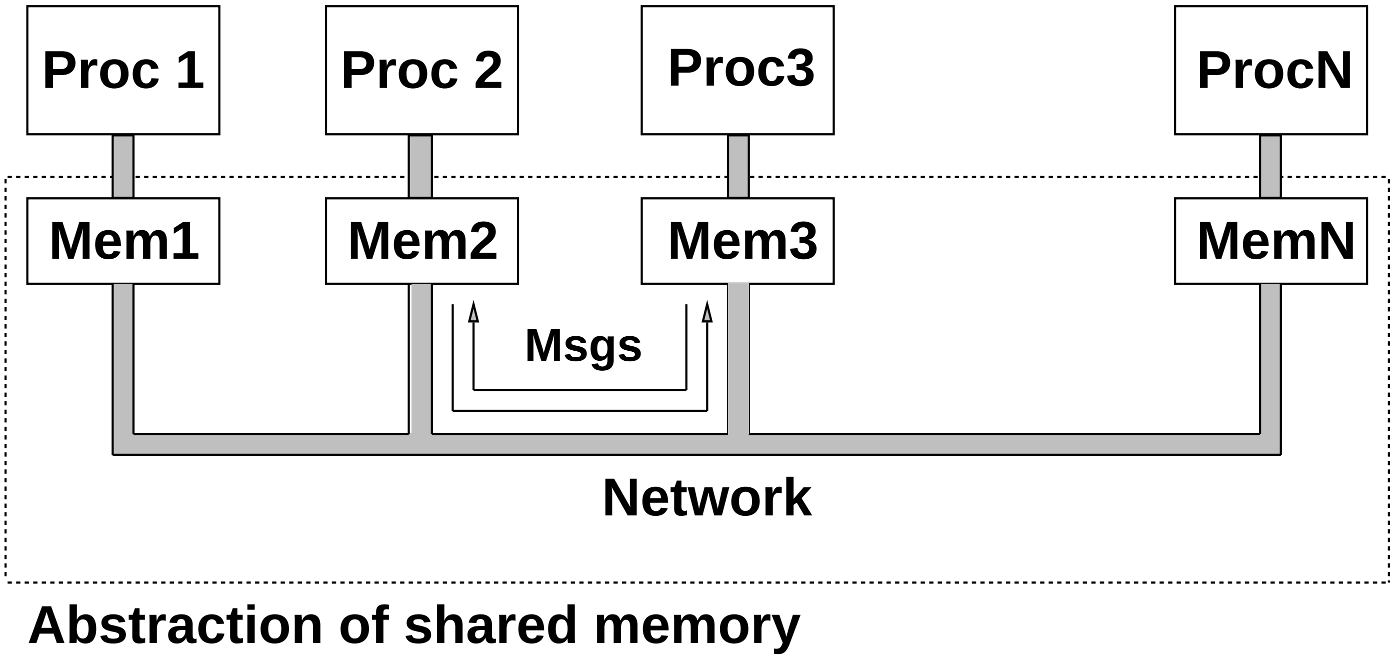
Distributed Shared Memory (DSM)

Definition: A software abstraction of shared memory for distributed memory multiprocessors [Li 86]



Motivation:

Easier to program than message passing

Page-Based DSM (Ivy)

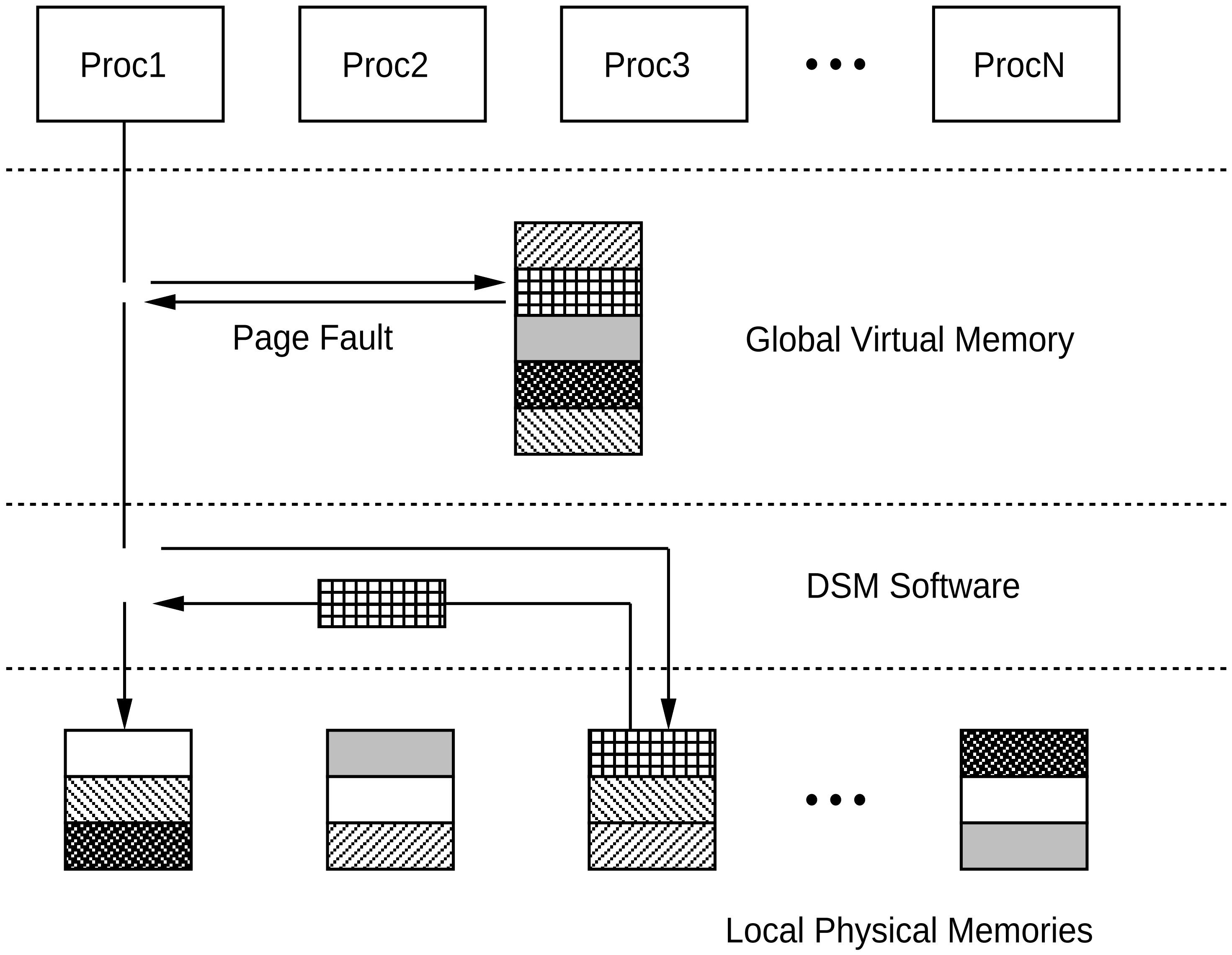
Mechanism

- remote-fork same program on each node
- data resides in common virtual address space
- use virtual memory trap handler to detect read/write to page

Issues

- how to keep illusion of shared memory?
- reduce interprocessor communication
- can processors keep local (cache) copy of page?

Conventional DSM Implementation [Li 86]



Definitions

Coherence

- ensure modifications propagate to all (cached) copies of data
- preserve program order
- serialize writes
- defines behavior of reads/writes to a memory location

Consistency

- defines when and in what order modifications are propagated
- defines behavior of reads and writes with respect to accesses to other memory locations

Coherence Protocol (Ivy)

Ownership

- static vs dynamic - can ownership change?
- centralized vs distributed - which node maintain ownership info?
- copyset - list of all nodes with copies

Mechanism

- invalidate - send message to discard local copy
- update - send new data for local copy

Page modes

- no access - cannot read, cannot write
- exclusive access - can read, can write
- shared access - can read, cannot write

Coherence Protocol (Ivy)

No access

- on transition from no-access, fetch copy of page
- any node with access has latest copy of page

Exclusive access

- node has only copy, at most one node
- on transition to exclusive, invalidate all remote copies and set their mode to no-access

Shared access

- multiple nodes may hold shared page
- on transition to shared mode, invalidate exclusive remote copy (if any) and set it to shared mode

Consistency Protocol (Ivy)

Sequential consistency (SC)

“A system is sequentially consistent if 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.” [Lamport79]

In practice, every write must be seen on all processors before any succeeding read or write can be issued.

Example:

```
    A = 0;              B = 0;
    A = 1;              B = 1;
    if (B == 0)          if (A == 0)
      ...                 ...
```

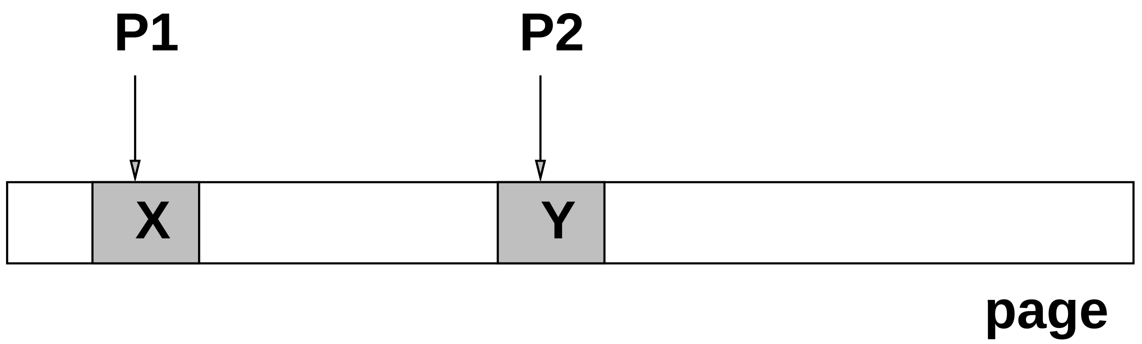
Problem: High Communication Cost

Communication is expensive.

- High latency.
- High processor cost.

Communication can be frequent.

Example: false sharing.

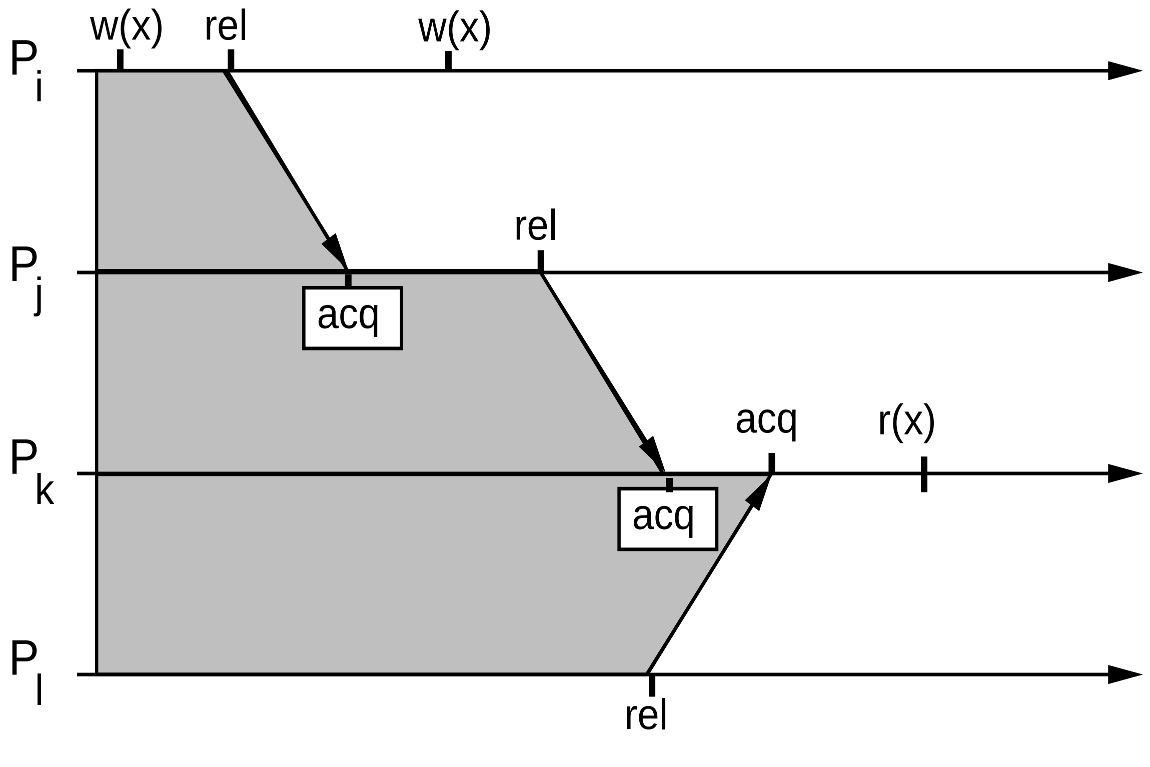


Leads to "ping-pong" effect

Release Consistency (RC) [Ghara. et al. 90]

Distinguish ordinary accesses and synchronization (acquire and release)

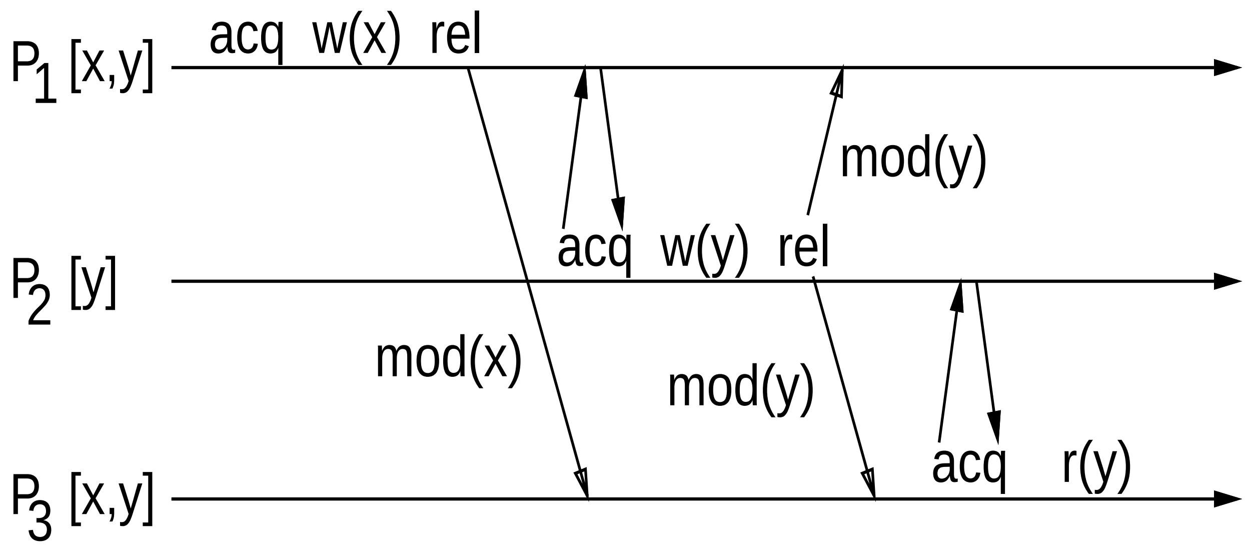
Paraphrased: "read the last value written by a processor that you synchronized with"



For properly-synchronized programs RC behaves as conventional SC memory

Eager RC

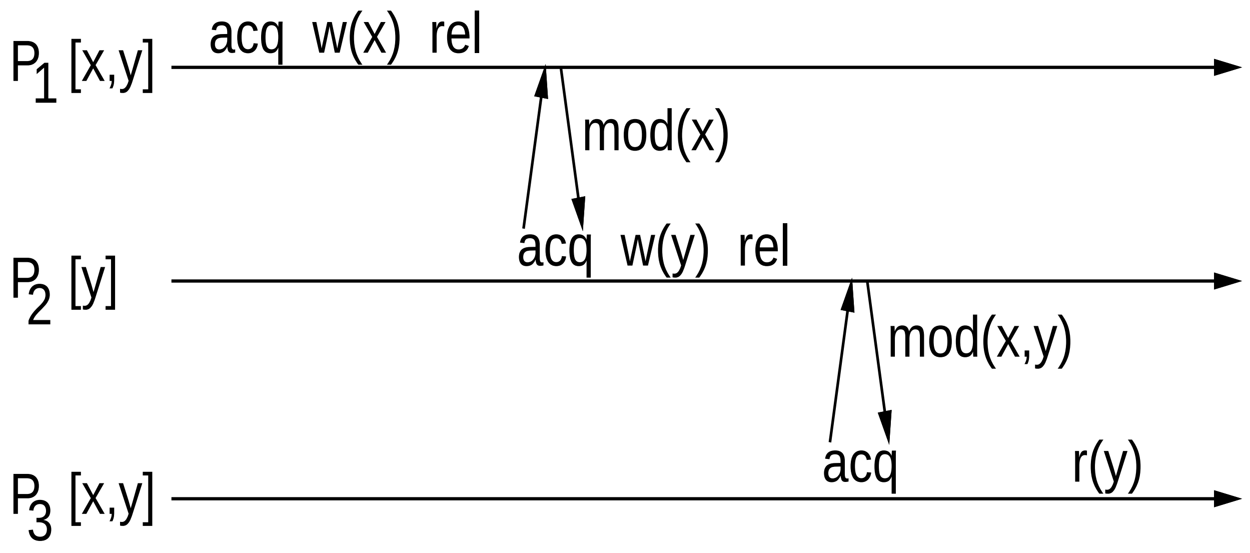
Push the modifications at a release.



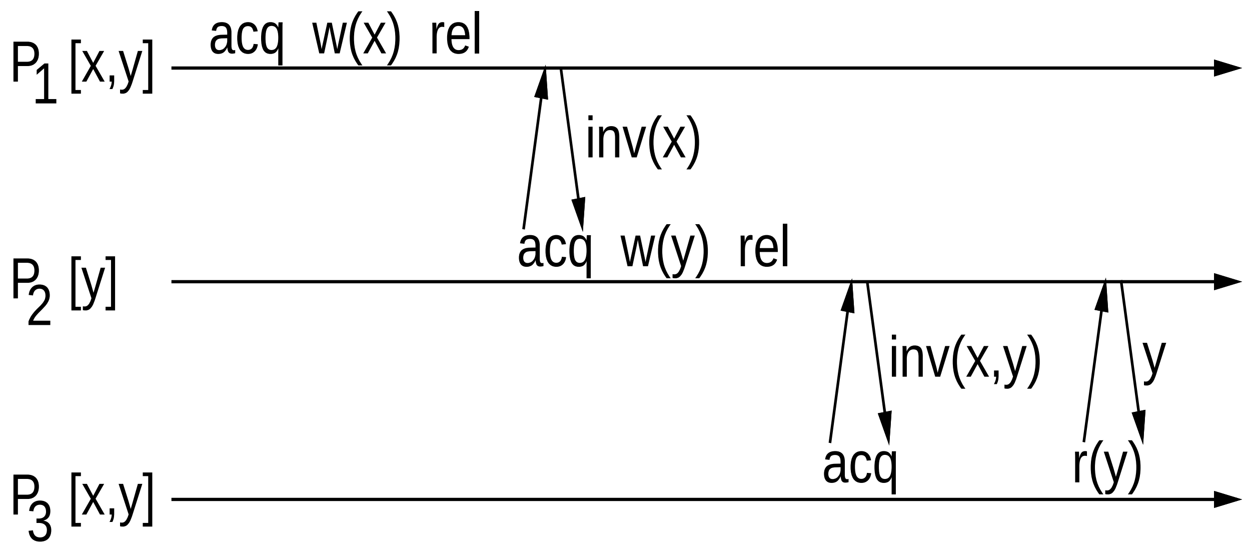
Lazy RC

Rather than push modifications at release,

pull them at acquire.



Lazy Invalidate Protocol

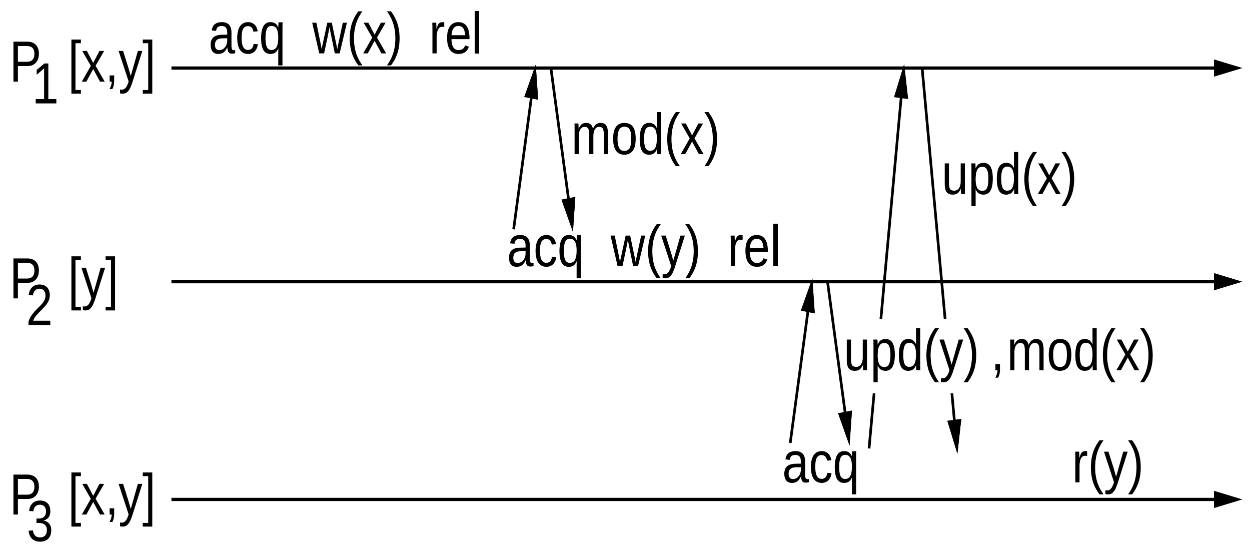


Data: low

Messages: high (access misses)

Lock latency: low

Lazy Update Protocol



Data: high

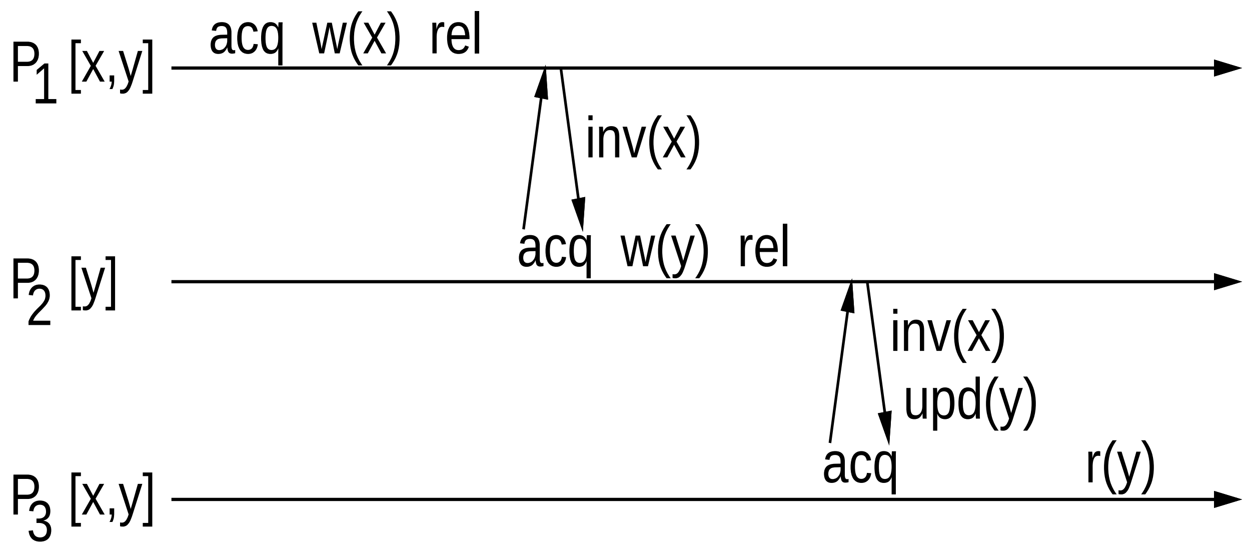
Messages: low

Lock latency: high

Lazy Hybrid Protocol

Send updates for shared values in cache.

Send invalidations for others.



Data: low

Messages: low

Lock latency: low

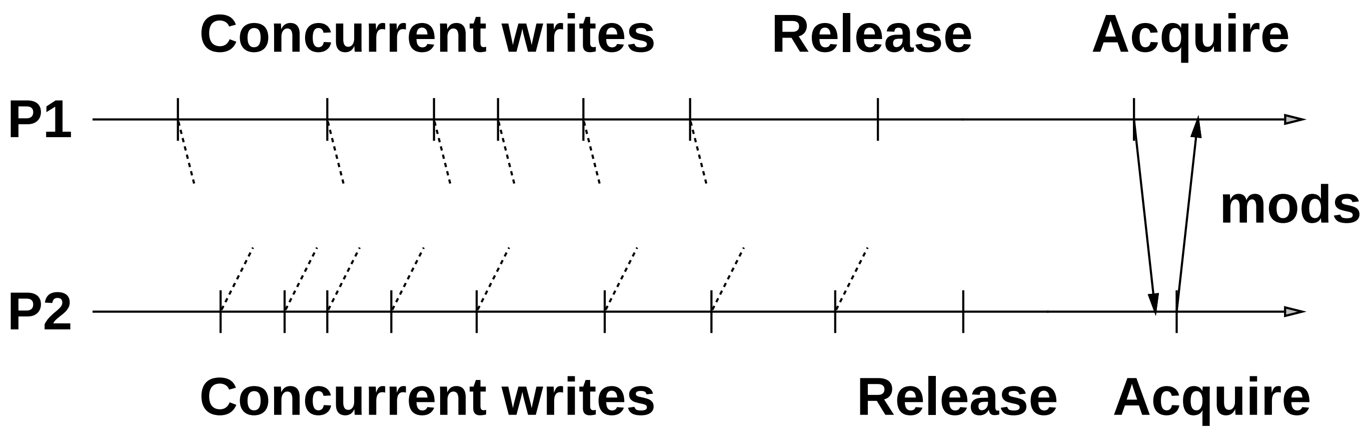
Multiple Writer Protocol

Reduces false sharing overhead

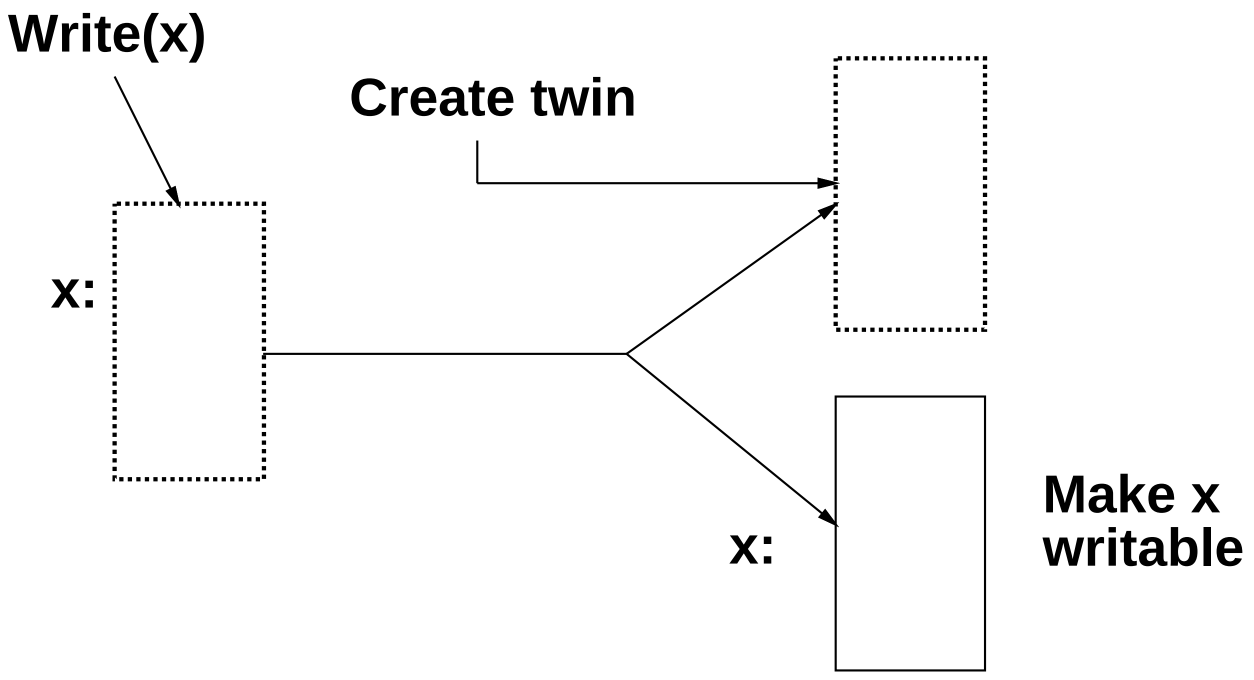
Buffer writes until release

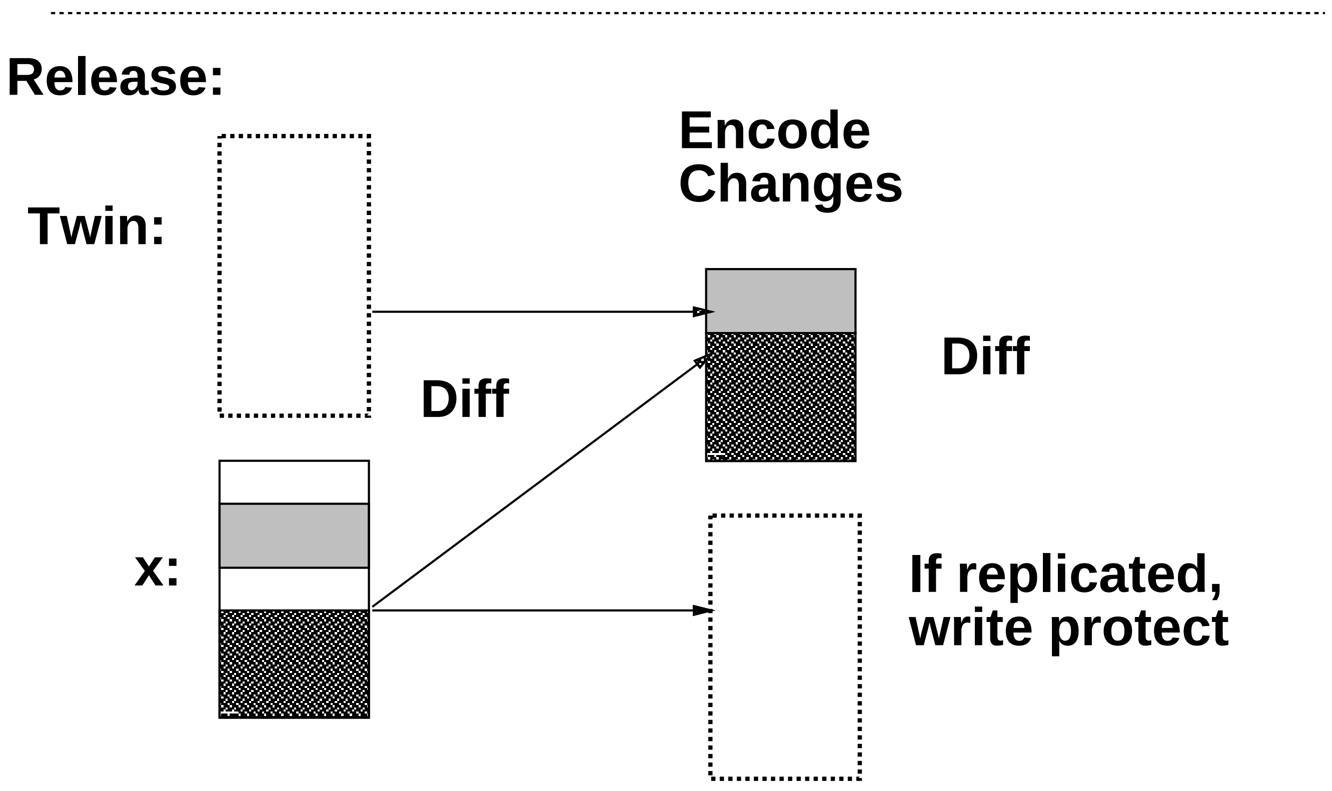
Create diffs

Acquire lock → pull in modifications



Diff Creation





TreadMarks Implementation

Hardware:

8 40Mhz DECStation-5000/240

100 Mbps Fore ATM LAN

Ultrix v4.3

Software:

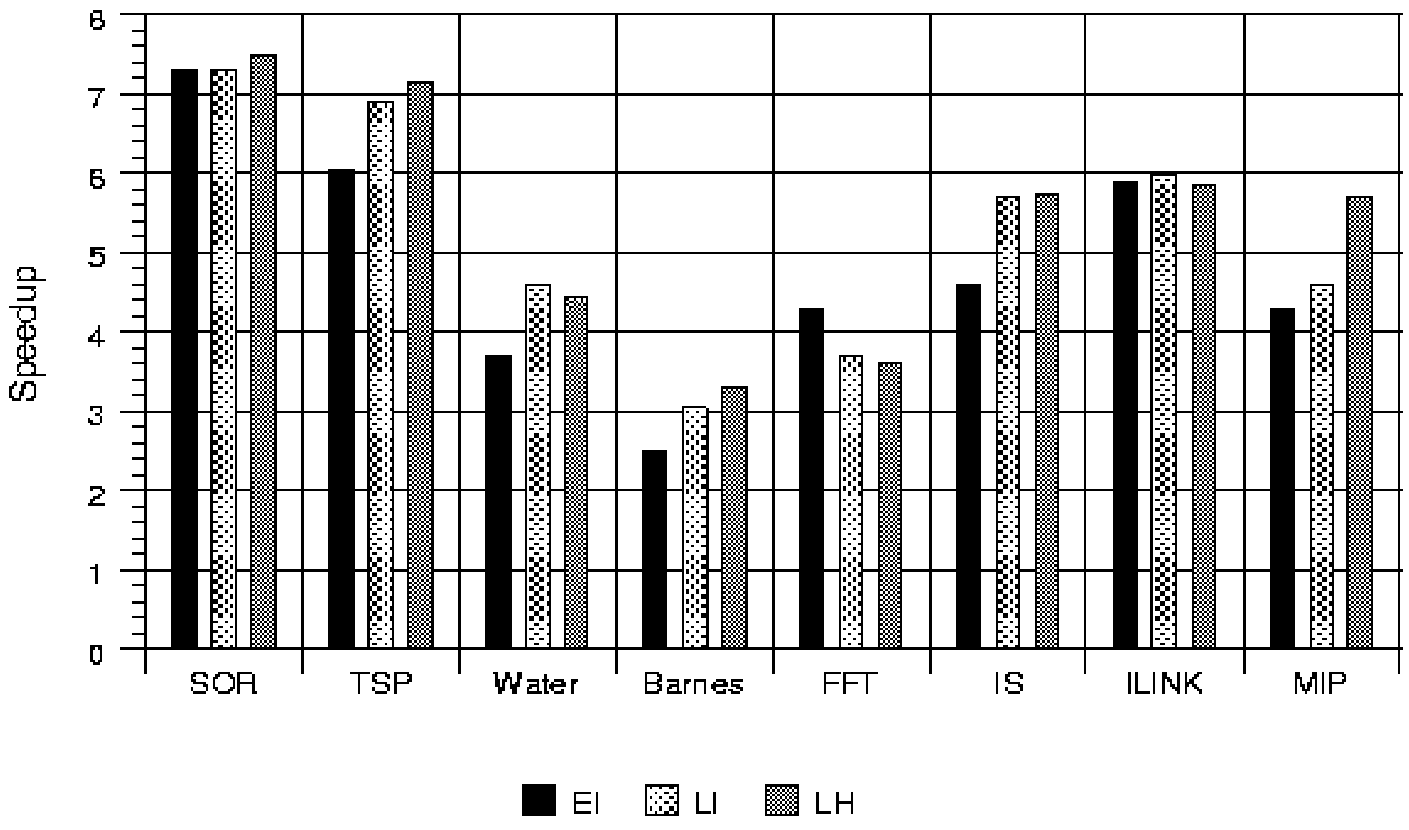
User-level library

Communication: AAL4 and UDP

SIGIO handler for incoming messages

mprotect and SIGSEGV handler

TreadMarks Speedups



Versions

- eager invalidate (EI)
- lazy invalidate (LI)
- lazy hybrid (LH)

Conclusions

Real applications can be run on DSMs.

LRC reduces communication...

...but communication primitives still dominate overhead.

Usefulness depends on application.