# Distributed Shared Memory

**Presented by**
Humayun Arafat

# Outline

*Background*

Shared Memory, Distributed memory systems

Distributed shared memory

Design

Implementation

TreadMarks

Comparison TreadMarks with Princeton's home based protocol

Colclusion

# SM vs DM

- Shared Memory
  - Global physical memory equally accessible to all processors
  - Programming ease and portability
  - Increased contention and longer latencies limit scalability
- Distributed memory
  - Multiple independent processing nodes connected by a general interconnection network
  - Scalable, but requires message passing
  - Programmer manages data distribution and communication

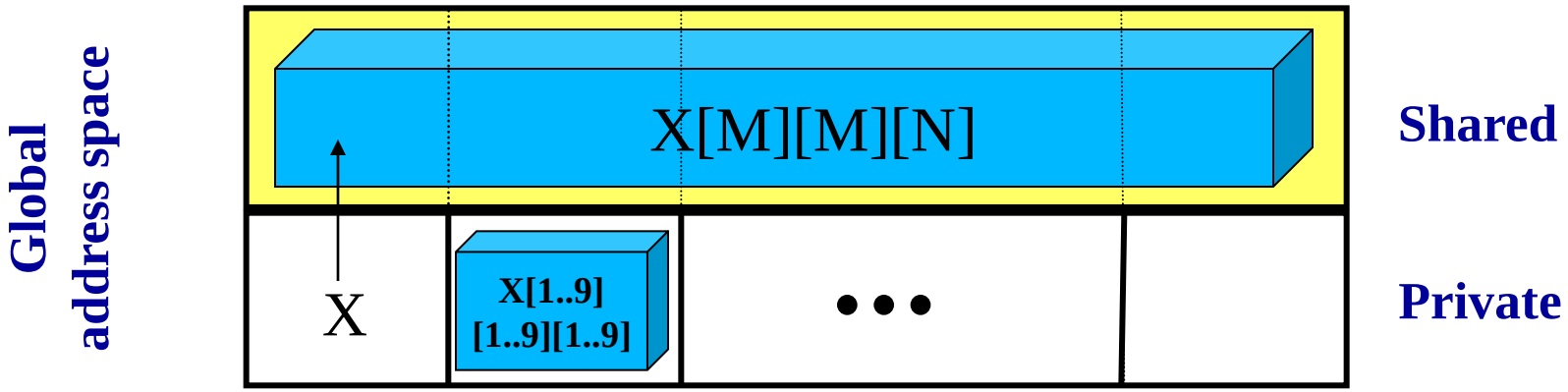
# Distributed shared memory

All systems providing a shared-memory abstraction on distributed memory system belongs to the DSM category

- DSM system hides remote communication mechanism from programmer

- Relatively easy modification and efficient execution of existing shared memory system application

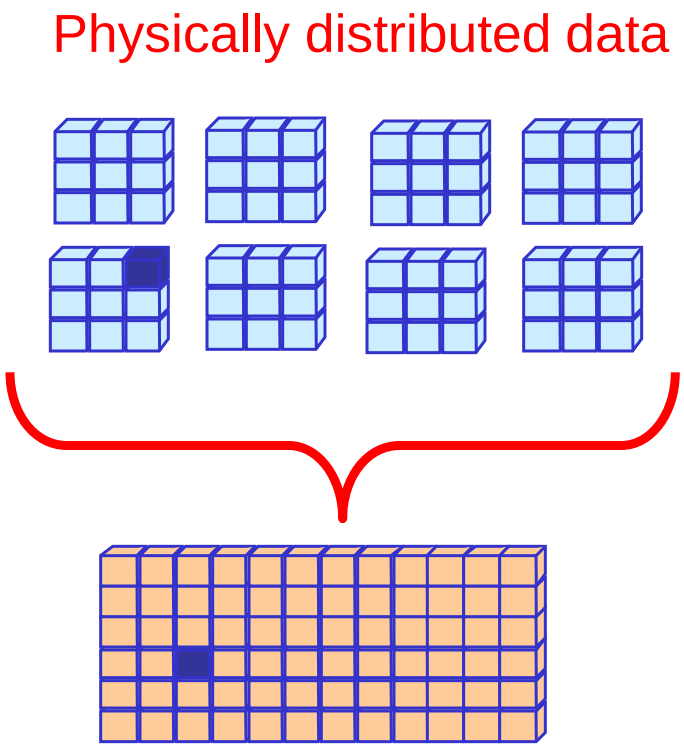
- Scalability and cost are similar to the underlying distributed system




- Aggregate distributed memories into global address space
  - Similar to the shared memory paradigm
  - Global address space is logically partitioned
  - Local vs. remote accessible memory
  - Data access via *get(..)* and *put(..)* operations
  - Programmer control over data distribution and locality

# Global Array

The Global Arrays (GA) Toolkit is an API for providing a portable "shared-memory" programming interface for "distributed-memory" computers.




single, shared data structure/ global indexing

e.g., access A(4,3) rather than buf(7) on task 2

Source: GA tutorial

# Outline

*Background*

Shared Memory, Distributed memory systems

Distributed Shared Memory

Design

Implementation

TreadMarks

Comparison TreadMarks with home based protocol

Colclusion

# Key Issues in designing DSM

Three key issues when accessing data in the DSM address space

DSM algorithm:

How the access of data actually happens

Implementation:

Implementation level of DSM mechanism

Consistency:

Legal ordering of memory references issued by a processor, as observed by other processors

# DSM algorithms

*Single reader/single writer algorithms*

- Prohibits replication, central server algorithm
- One unique server handles all requests from other nodes to shared data
- Only one copy of data item can exist at one time
- Improvement- distribution of responsibilities for parts of shared address space and static distribution of data
- Performance is very low
- Does not use the parallel potential of multiple read or write

# DSM algorithms

*Multiple reader/single writer algorithms*

- Reduce cost of read operations because read is the most used pattern in parallel applications
- Only one host can update a copy
- One write will invalidate other replicated copies which increases the cost of write operation

# DSM algorithms

*Multiple reader/Multiple writer algorithms*

- Allows replication of data blocks with both read and write
- Cache coherency is difficult to maintain. Updates must be distributed to all other copies on remote sites
- Write update protocol
- High coherence traffic

# Implementation of DSM

Implementation Level

One of the most important decisions of implementing DSM

Programming , performance and cost depend on the level

- Hardware
  - Automatic replication of shared data in local memory and cahe
  - Fine grain sharing minimize effects of false sharing
  - Extension of cache coherence scheme of shared memory
  - Hardware DSM is often used in high-end system where performance is more important than cost
- Software
  - Larger grain sizes are typical because of virtual memory
  - Applications with high locality benefit from this
  - Very flexible
  - Performance not comparable with hardware DSM

# Implementation of DSM

- Hybrid
  - Software features are already available in hardware DSM
  - Many software solutions require hardware support
  - Neither software or hardware has all the advantages
  - Use hybrid solutions to balance the cost complexity trade offs

# Memory consistency model

## Consistency

- Sequential consistency
- Processor consistency
- Weak consistency
- Release consistency
- Lazy release consistency
- Entry consistency

# Memory consistency model

## Sequential Consistency

- Result of any execution is the same as if the read and write occurred in the same order by individual processors
- DSM system serialize all requests in a central server node

## Release Consistency

- Divides synchronization accesses to acquire and release
- Read and write can happen after all previous acquires on the same processor. Release, after all previous read, write execute
- acquire and release synchronization accesses must hold processor consistency

# TreadMarks

- Shared memory as a linear array of bytes via a relaxed memory model called release consistency
- Uses virtual memory hardware to detect accesses
- Multiple writer protocol to alleviate problems caused by mismatches between page size and application granularity
- Portable, run at user level on Unix machine without kernel modifications
- Synchronizations – locks, barriers

# Anatomy of a TreadMarks Program:

## Starting remote processes

```
Tmk_startup(argc, argv);
```

## Allocating and sharing memory

```
shared = (struct shared*) Tmk_Malloc(sizeof(shared));
Tmk_distribute(&shared, sizeof(shared));
```

## Barriers

```
Tmk_barrier(0);
```

## Acquire/Release

```
Tmk_lock_acquire(0);
shared->sum += mySum;
Tmk_lock_release(0);
```

# Implementation

```
/* Initialize TreadMarks and start the remote processes. */
void    Tmk_startup(int argc, char **argv)
/* Terminate the calling process. Other processes are unaffected. */
void    Tmk_exit(int status)
/* Block the calling process until every other process arrives at the barrier. *
void    Tmk_barrier(unsigned id)
/* Block the calling process until it acquires the specified lock. */
void    Tmk_lock_acquire(unsigned id)
/* Release the specified lock. */
void    Tmk_lock_release(unsigned id)
/* Allocate the specified number of bytes of shared memory. */
char   *Tmk_malloc(unsigned size)
/* Free shared memory allocated by Tmk_malloc. */
void    Tmk_free(char *ptr)
```

# Sample TreadMarks program

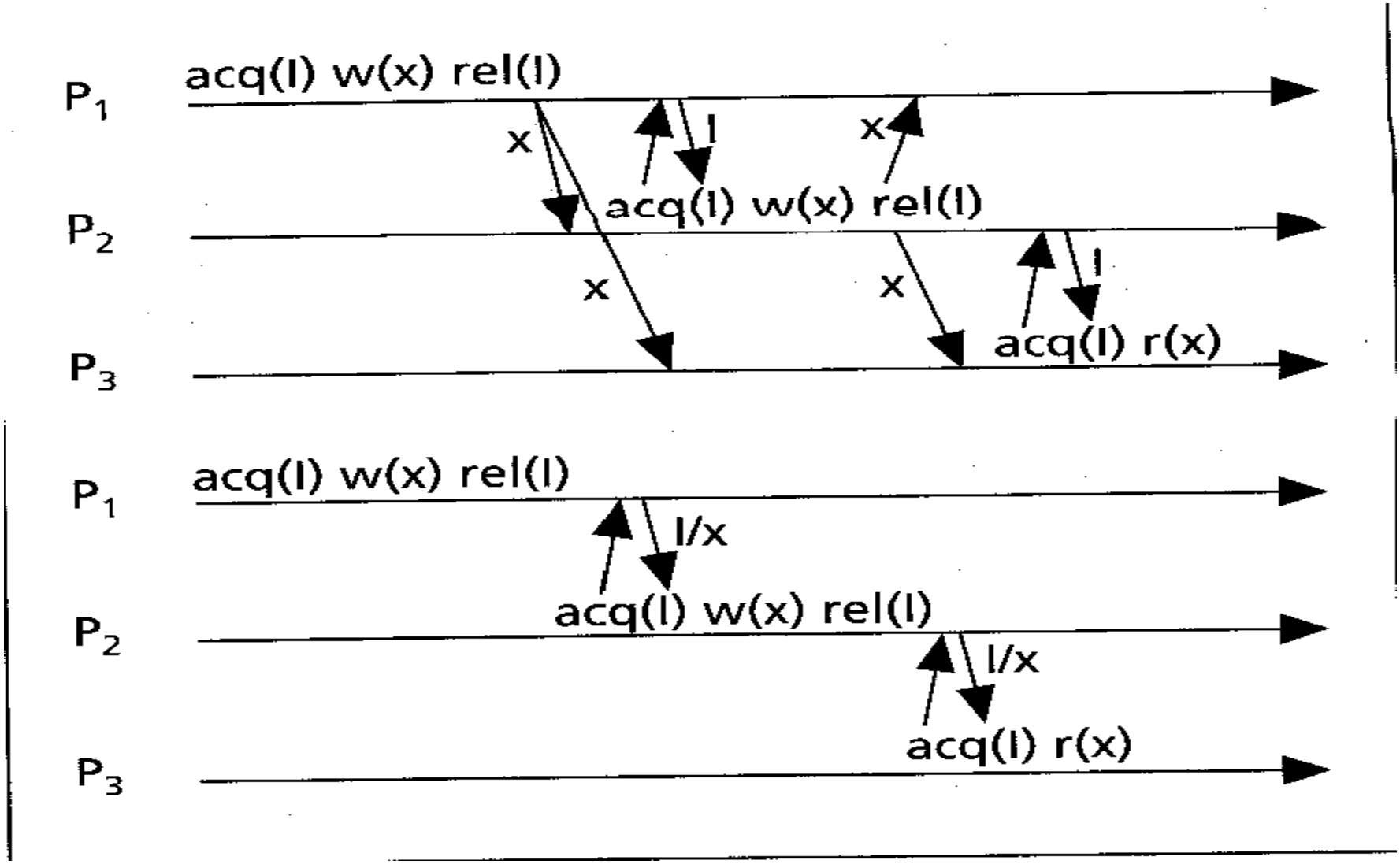
```
main()

{
    Tmk_startup();

    if( Tmk_proc_id == 0) {
        grid = Tmk_malloc( M*N*sizeof(float) );
        initialize grid;
    }

    Tmk_barrier(0);

    length = M / Tmk_nprocs;
    begin = length * Tmk_proc_id;
    end = length * (Tmk_proc_id+1);

    for( number of iterations ) {

        for( i=begin; i<end; i++ )
            for( j=0; j<N; j++ )
                scratch[i][j] = (grid[i-1][j]+grid[i+1][j]+
                                 grid[i][j-1]grid[i][j+1])/4;

        Tmk_barrier(1);

        for( i=begin; i<end; i++ )
            for( j=0; j<N; j++ )
                grid[i][j] = scratch[i][j];

        Tmk_barrier(2);


    }
}
```

# Lazy release consistency

## Release consistency model

- Synchronization must be used to prevent data races
- Multiple writer
- Twin
- Reduce false sharing
- Modified pages invalidated at acquire
- Page updated at access time
- Updates are transferred as diffs
- Lazy diffs- make diffs only when they are requested

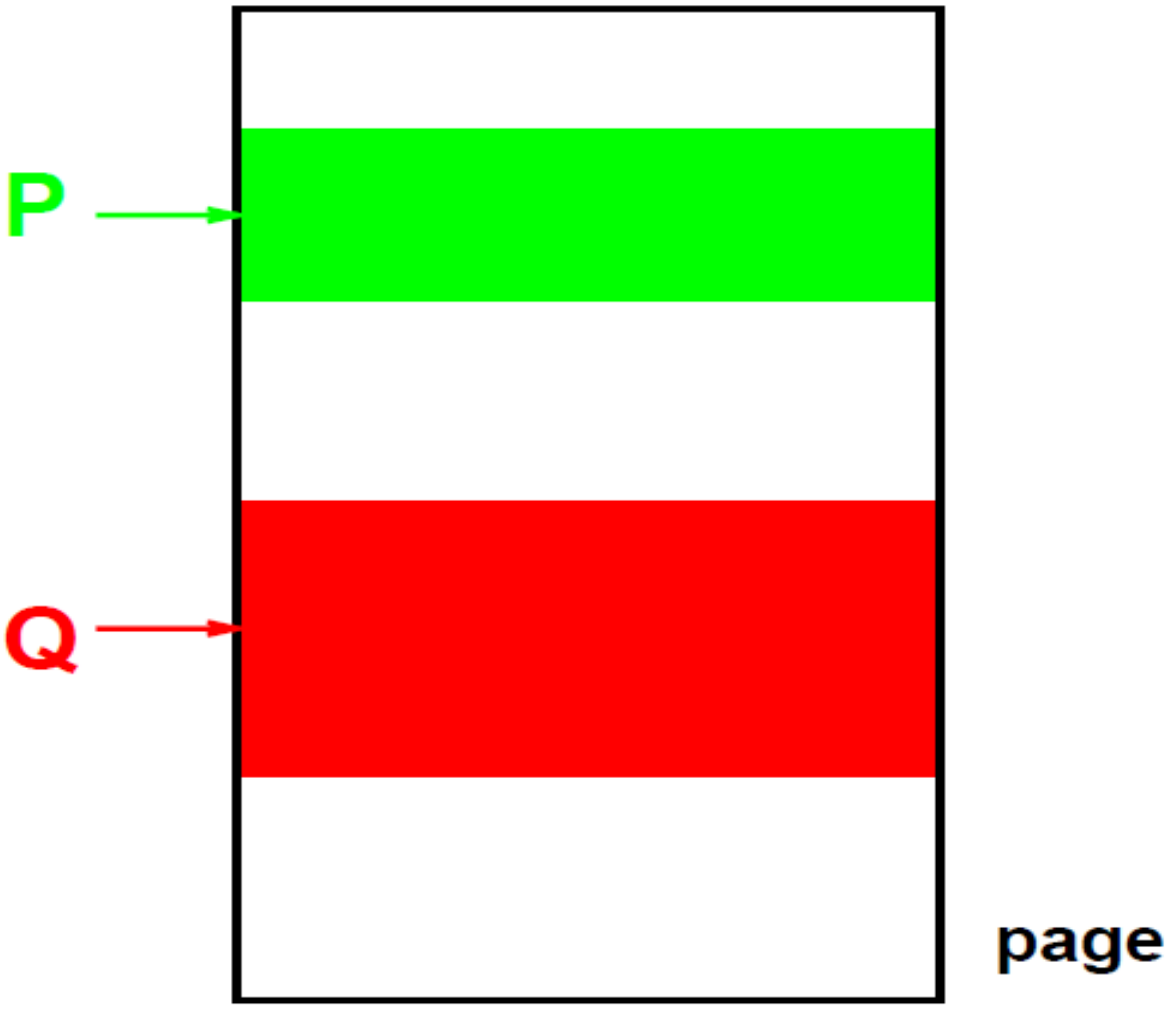
# Eager release versus Lazy release

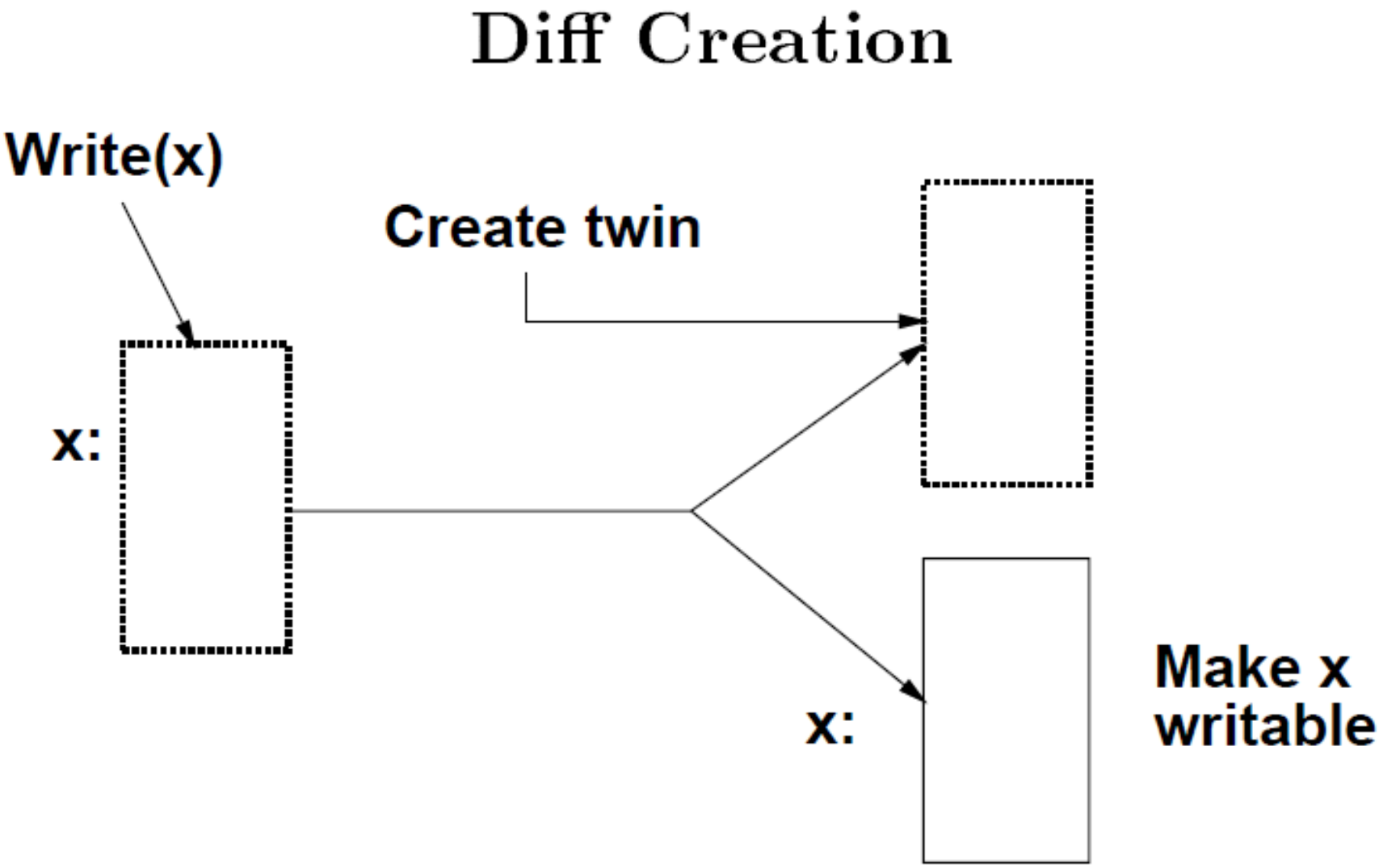


# Multiple writer protocol

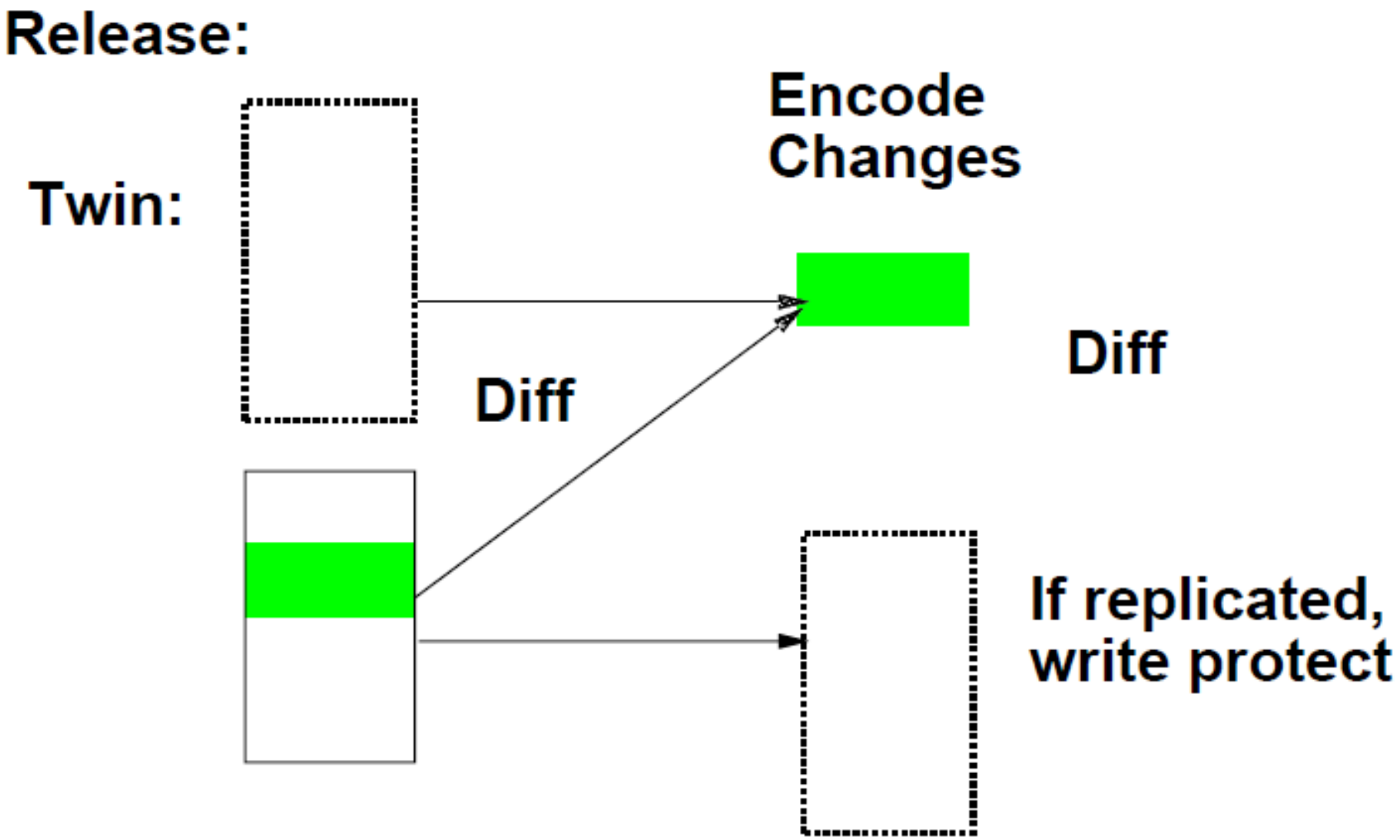
- False sharing handle
- Buffer written until synchronization
- Create diffs, run length encoding page modifications
- Diffs reduce bandwidth requirements

# False sharing




Diff Creation
Write(x)
Create twin
x:
x:
Make x writable

# Diff


Release:
Encode
Changes
Twin:
Diff
Diff
If replicated,
write protect

# TreadMarks system

- Implemented as a user-level library on top of Unix
- Inter-machine communication using UDP/IP through the Berkeley socket interface
- Messages are sent as a result of an call to library routine or page fault
- It uses SIGIO signal handler for receive request messages
- For consistency protocol, TreadMarks uses the mprotect system call to control access to shared pages. Shared page access generates a SIGSEGV signal
-

# Homeless and home-based Lazy release

- Two most popular multiple writer protocols that are compatible with LRC
  - TreadMarks protocol(Tmk)
  - Princeton's homebased protocol(HLRC)
- Similarity

  In both protocols, modifications to shared pages are detected by virtual memory faults(twinning) and captured by comparing the page to its own twin

- Differences

  Location where the modifications are kept

  Method by which they get propagated

# HLRC

- Shared page is statically assigned a home processor by the program
- At a release, a processor immediately generates the diffs for the pages that it has modified since its last release
- Then send the diffs to their home processor. Immediately update the home's copy of the message
- Processor access an invalid page, it sends a request to the home processor. Home processor always responds with a complete copy of the message

# Tmk vs HLRC

- For migratory data, Tmk uses half as many messages, because transfer the diff from last writer to the next writer
- For producer/consumer data, the two protocols use the same number of messages
- HLRC uses significantly fewer messages during false sharing
- Assignment of pages to homes is important for good performance
- Tmk creates fewer diffs because their creation is delayed

# Conclusion

- DSM viable solution for large scale because of the combined advantages of shared memory and distributed memory

- Very active research area

- With suitable implementation technique distributed shared memory can provide efficient platform for parallel computing on networked workstations

Questions?

**THANK YOU**