MASI CACHE COHERENCE PROTOCOL FOR MULTICORE ARCHITECTURE

Shivam Pant, Tannishtha Som, Shivani Bansal

April 29, 2014

Abstract

This work proposes a novel 4-state cache coherence protocol that ensures better efficiency in Chip Multiprocessors (CMPs) compared to that of conventional 4 and 5-state protocols, e.g. MESI/MOESI. The proposed MASI coherence protocol realizes judicious cache line state transition that has considerable impact on the reduction in number of data block forwarding. Its feature is to keep trace of the eviction of data block from the upper level cache. It further brings down the overhead of writeback. The simulation results establish the effectiveness of the MASI protocol for a directory based system that is conventionally employed for large scale CMPs.

Acknowledgements

This dissertation is dedicated to our friendship because it would not have happened without each others' support, encouragement, patience, and love. We would like to thank our parents who always supported our endeavors, never doubted our abilities, and deserve much thanks. National Institute of Technology, Durgapur provided us with the opportunities that opened the doors in our lives, both personally and professionally. We thank our advisor Mrs. Mamata Dalui for supporting our pursuit of this project. Mrs. Mamata is truly one of the best in the field and we had no idea how good she really is until the latter stages of our undergraduate student career. We thank the rest of our committee for their helpful feedback: Suraj, Shalini, Anirban, Gaurav and Siddhant. Undergraduate school was a rewarding experience because of all the great students we've met and friends we've made. We can't possibly list the names of all the people that have made a positive impact on our experience over the last 4 years. We thank Ashok Pradhan for becoming an excellent colleague and providing invaluable advices from time to time; Manisha De for always offering sound technical opinions and witty entertainment; Himanshu for coffee break conversations and encouragement. Along with Aman, we studied for the examinations with Subhra, Anubhav, Harshit and Sindhu. We thank them for their support and for helping us pass. Famee for the support, encouragement, and for listening to our various rants and raves over the years. Sandy for providing wisdom that only an experienced engineer can give. Ipsita for offering key advice during the critical moments of the project. Nidhi and Sukriti for making group meetings more fun. Sumana for being an easy-going labmate. Finally we thank the Computer Systems Laboratory for supporting our research.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In present day scenario, Computers are an integral part of everyones' day-to-day life. It is all possible because of decreasing cost of microprocessors. Improvements in microprocessors largely depends on Moore's Law, which predicts that the number of transistors per silicon area doubles every eighteen months [21]. Initially performance of microprocessors were improved by increasing the processor speed (its frequency), and by increasing the amount of parallelism. Parallelism was improved by increasing the basic word length of machines from 4-bits to currently 64-bits. Architects then sought to increase parallelism by executing multiple instructions simultaneously (instruction-level parallelism or ILP) through pipelining techniques and superscalar architectures and to reduce the latency of accessing memory with ever larger on-chip caches. It now appears that existing techniques for increasing ILP can no longer deliver performance improvements that track Moore's Law due to energy, heat, and wire delay issues [3]. Therefore, now the architects are moving towards thread-level parallelism (TLP) by designing chips with multiple processors, otherwise known as Multicore or Chip Multiprocessors (CMPs). By extracting higher-level TLP on multicores, performance can continue to improve while managing the technology issues faced by increasing the performance of conventional single-core designs (uniprocessors). Industry is embracing multicore by rapidly increasing the number of processing cores per chip. In 2005, AMD and Intel both offered dual-core x86 products [1]. Meanwhile Sun shipped an 8-core, 32-threaded CMP in 2005[17] and plans a 16-core version in 2008.

1.1 Cache Coherence

With the rise of multicore architecture, most of the softwares are now becoming parallel in nature. Parallel softwares rely on the shared-memory model in which all processors access the same physical address space. Although processors logically access the same memory, on-chip cache levels are crucial to achieving fast performance for the majority of memory accesses made by processors. Thus the major issue with shared-memory multiprocessors is to maintain the consistency of memory with various cache levels. This cache coherence problem is a critical correctness and performance-sensitive design point for supporting the shared-memory model.The cache coherence mechanisms not only govern communication in a shared-memory multiprocessor, but also typically determine how the memory system transfers data between processors, caches, and memory. Assuming the shared- memory programming model remains prominent, future workloads will depend upon the performance of the cache coherent memory system and continuing innovation in this field is necessary for progress in computer design.

1.2 Thesis Contribution

This section describes the research contributions of the dissertation.

1.2.1 MASI Cache Coherence Protocol

In Chapter 4 we have proposed a protocol called MASI Cache Coherence Protocol. This 4 state protocol mimics the features of 5 state MOESI protocol and performs better than classical 4 state MOSI protocol.

1.3 Dissertation Structure

Chapter 2 presents a background on the cache coherence problem and an overview of classical Cache Coherence Protocols. Chapter 3 discusses the tool, performance metrics, and workloads used for evaluation. Chapter 4 presents our work on MASI Cache Coherence Protocol. Finally, Chapter 5 provides conclusion of our research work.

Chapter 2

Background of Cache Coherence

This chapter presents an overview of the cache coherence problem and classical Cache Coherence Protocols. We focus on the aspects most fundamental and related to the research in this dissertation. Section 2.1 develops the cache coherence problem in terms of multiprocessor memory consistency. Section 2.2 presents back- ground on classical coherence techniques.

2.1 Memory consistency in multiprocessor

2.1.1 Overview

In serial programming running on von Neumann machines, instructions appear to execute in a sequential order.Importantly, a programs load returns the last value written to the memory location. Likewise a store to a memory location determines the value of the next load. This definition leads to straightforward implementations and semantics for programs running on a single uniprocessor.

Whereas, in multithreaded programs running on multiprocessor architecture the situation becomes quite complicated. In particular, the value returned by a given load is not clear because the most recent store may have occurred on a different processor core. Thus architects define *memory consistency models*[2] to specify how a processor core can observe memory accesses from other processor cores in the system. The way in which consistency is maintained is influenced by the presence of cache memories.

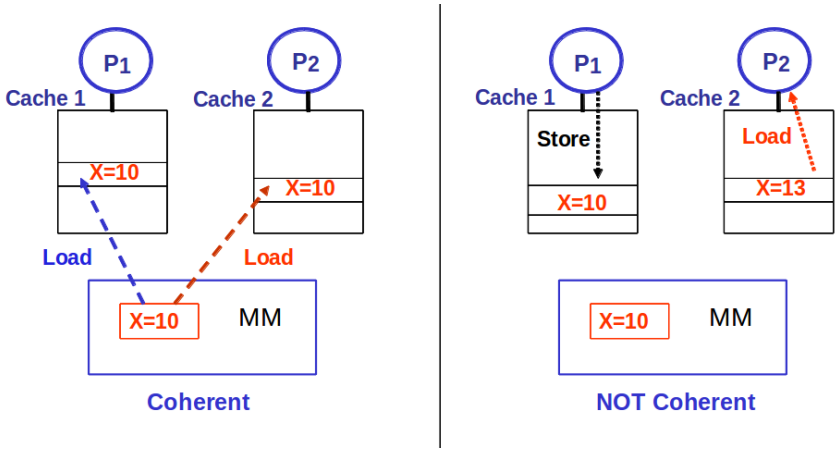

Figure 2.1: Cache Coherence Problem

2.1.2 Impact of cache on memory consistency

Cache memories allow processor speeds to increase at a greater rate than DRAM speeds by exploiting locality in memory accesses. Their operation is all hardware-based and automatic from a programmer's point-of-view. While implementing a cache hierarchy had little ramification on a uniprocessor's memory consistency, caches complicate multiprocessor memory consistency. The root of the problem lies in store propagation.As we can see from the Figure 2.1, while two processors in a system, P_1 and P_2, may both load the same memory block into their respective private caches, a subsequent store by either of the processors would cause the values in the caches to differ. Thus if P_1 stores to a memory block present in both the caches of P_1 and P_2, P_2's cache holds a potentially stale value because of P_1's default operation of storing to its own cache. This cache incoherence would not be problematic if P_2 never again loads to the block while still cached. But since the point of multiprocessor memory models is to support shared-memory programming, at some point future loads of the block by P_2 must receive the new value stored by P_1, as defined by the model. That is, P_1's store must potentially affect the value in the block of P_2's cache to maintain consistency, and the mechanisms for doing so are defined as cache coherence.

2.2 Classical Cache Coherence Protocols

This section explains some of the fundamental Cache Coherence Protocols.

State	Processor Action			Incoming	
	Load	**Store**	**Eviction**	**Read Req.**	**Write Req.**
M	Hit	Hit	Writeback →**I**	send data and write-back →**S**	send data →**I**
S	Hit	write request →**M**	Silent Evict →**I**	(none)	(none) →**I**
I	Read request →**S**	write request →**M**	(none)	(none)	(none)

Table 2.1: State Transitions for MSI cache coherence protocol

2.2.1 MSI Cache Coherence Protocol

The first protocol proposed was the classic MSI protocol which has three states: Modified(M), Shared(S) and Invalid(I). If a cache has block in Modified state then it means that the block is present in only the current processor and the caches of the other processors don't have that block. The block is inconsistent with the lower level memory and it is the duty of the cache in 'M' state to perform the writeback to lower memory level whenever the block is evicted.The Shared state means that the block is not modified and is present in at least one cache. Whenever a block in 'S' state is evicted it does not have to perform a writeback to lower memory level. The Invalid state implies that the block is not present in the cache [14]. Table 2.1 shows the state transition for MSI cache coherence protocol[18].

State	Processor Action			Incoming	
	Load	**Store**	**Eviction**	**Read Req.**	**Write Req.**
M	Hit	Hit	Writeback →**I**	send data and write-back →**S**	send data →**I**
E	Hit	hit →**M**	Silent Evict →**I**	send data →**S**	send data →**I**
S	Hit	write request →**M**	Silent Evict →**I**	send data	(none) →**I**
I	Read request (response: shared) →**S** or Read request (response: clean) →**E**	write request →**M**	(none)	(none)	(none)

Table 2.2: State Transitions for MESI cache coherence protocol

2.2.2 MESI Cache Coherence Protocol

MESI protocol is an improvement to the MSI protocol, which adds a fourth state called the Exclusive (E) state. A cache having a block in the Exclusive state means that it is the only cache which has the data and it is clean, i.e, it matches with the copy in the main memory. It may be changed to the Shared state at any time, in response to a read request. Alternatively, it may be changed to the Modified state when writing to it. The advantage of having the Exclusive state is that this block can be evicted without updating the block at the home memory. Also unnecessary invalidations will not be send for transition of Exclusive state to Modified state to the other processors [23] [27]. Table 2.2 shows the state transition for MESI cache coherence protocol.

State	**Processor Action**			**Incoming**	
	Load	**Store**	**Eviction**	**Read Req.**	**Write Req.**
M	Hit	Hit	Writeback →**I**	send data →**O**	send data →**I**
O	Hit	write request →**M**	Writeback →**I**	send data	send data →**I**
S	Hit	write request →**M**	Silent Evict →**I**	(none)	(none) →**I**
I	Read request →**S**	write request →**M**	(none)	(none)	(none)

Table 2.3: State Transitions for MOSI cache coherence protocol

2.2.3 MOSI Cache Coherence Protocol

MOSI protocol adds a fourth state: Owned(O). The Owned state in a processor's cache allows read-only access to the block (much like 'S' state), but also signifies that the value in main memory is incoherent or stale. Thus the processor in Owned state must update the memory before evicting a block. As with the Modified state, only a single processor is allowed to be in the Owned state at one time. Unlike Modified, however, other processors are allowed to be in the Shared state when one processor is in the Owned state. The advantage of having a cache block in Owned state is that it is often given the additional responsibility of responding to requests for data, providing a convenient mechanism for selecting a single responder. Thus there is less traffic[5] [27]. Table 2.3 shows the state transition for MOSI cache coherence protocol[18].

State	Processor Action			Incoming	
	Load	**Store**	**Eviction**	**Read Req.**	**Write Req.**
M	Hit	Hit	Writeback →**I**	send data →**O**	send data →**I**
O	Hit	write request →**M**	Writeback →**I**	send data	send data →**I**
E	Hit	hit →**M**	Silent Evict →**I**	send data →**S**	send data →**I**
S	Hit	write request →**M**	Silent Evict →**I**	(none)	(none) →**I**
I	Read request (response: shared) →**S** or Read request (response: clean) →**E**	write request →**M**	(none)	(none)	(none)

Table 2.4: State Transitions for MOESI cache coherence protocol

2.2.4 MOESI Cache Coherence Protocol

MOESI protocol is an extension of MESI or MOSI protocol. It is a five state protocol and includes both the Exclusive state and the Owned state. Thus, it includes the advantages of both MESI and MOSI [27] [9] [24]. Table 2.4 shows the state transition for MOESI cache coherence protocol[18].

Chapter 3

Evaluation Methodology

This chapter presents the common evaluation methodology used for the dissertation.

3.1 Multi2Sim

We are using a open-source simulator called Multi2sim to study the Cache Coherence Protocols.It is a simulation framework for CPU-GPU heterogeneous computing with its code written in C.It includes models for superscalar, multithreaded, and multicore CPUs, as well as GPU architectures.It can perform detailed simulations, take the number of cores on which we want to run the program explicitly, generate memory report and the memory trace.It has a visual tool with which we can visualize the state of the CPU and GPU pipeline and the state of the memory, providing features such as simulation pausing, stepping through cycles, and viewing properties of in-flight instructions and memory accesses.Regarding memory hierarchy-Multi2Sim provides a very flexible configuration of the memory hierarchy. Any number of cache levels can be used, with any number of caches in each level [30].

3.2 Method

Evaluating the performance of a proposed protocol requires meaningful metrics.

We use *forwardings per 100,000 immediate lower level cache* accesses as a metric to judge performance improvements of the Cache Coherence Protocols, where forwarding is the movement of data block from immediate lower level to current level cache.

In addition to this we add *writebacks per 100,000 immediate lower level cache* accesses as another metric for evaluating the performance of Cache Coherence Protocols, where writeback

is the movement of data block from current level to immediate lower level cache to maintain data consistency.

The performance of a protocol depends upon the number of lower level references. Due to higher latency to access the lower level memory, we try to reduce these memory references. Hence we have used normalized forwardings and normalized writebacks as the performance metrics.

3.3 Workload Descriptions

Here we describe the SPLASH-2 Application Suite used for evaluation of Cache Coherence Protocols. It includes the following standard program [31]

- **Barnes**: The Barnes application simulates the interaction of a system of bodies (galaxies or particles, for example) in three dimensions over a number of time-steps, using the Barnes-Hut hierarchical N-body method. It represents the computational domain as an octree with leaves containing information on each body, and internal nodes representing space cells. Most of the time is spent in partial traversals of the octree (one traversal per body) to compute the forces on individual bodies. The communication patterns are dependent on the particle distribution and are quite unstructured. No attempt is made at intelligent distribution of body data in main memory, since this is difficult at page granularity and not very important to performance.

- **Cholesky**: The blocked sparse Cholesky factorization kernel factors a sparse matrix into the product of a lower triangular matrix and its transpose. It is similar in structure and partitioning to the LU factorization kernel (see LU), but has two major differences: (i) it operates on sparse matrices, which have a larger communication to computation ratio for comparable problem sizes, and (ii) it is not globally synchronized between steps.

- **FFT**: The FFT kernel is a complex 1-D version of the radix- $\sqrt{n}$ six-step FFT algorithm described in [6], which is optimized to minimize interprocessor communication. The data set consists of the n complex data points to be transformed, and another n complex data points referred to as the *roots of unity*. Both sets of data are organized as $\sqrt{n} * \sqrt{n}$ matrices partitioned so that every processor is assigned a contiguous set of rows which are allo-

cated in its local memory. Communication occurs in three matrix transpose steps, which require all-to-all interprocessor communication. Every processor transposes a contiguous submatrix of $\frac{\sqrt{n}}{p} * \frac{\sqrt{n}}{p}$ from every other processor, and transposes one submatrix locally. The transposes are blocked to exploit cache line reuse. To avoid memory hotspotting, submatrices are communicated in a staggered fashion, with processor i first transposing a submatrix from processor $i + 1$, then one from processor $i + 2$, etc. See [33] for more details.

- **FMM**: Like Barnes, the FMM application also simulates a system of bodies over a number of timesteps. However, it simulates interactions in two dimensions using a different hierarchical N-body method called the adaptive Fast Multipole Method [11]. As in Barnes, the major data structures are body and tree cells, with multiple particles per leaf cell. FMM differs from Barnes in two respects: (i) the tree is not traversed once per body, but only in a single upward and downward pass (per timestep) that computes interactions among cells and propagates their effects down to the bodies, and (ii) the accuracy is not controlled by how many cells a body or cell interacts with, but by how accurately each interaction is modeled. The communication patterns are quite unstructured, and no attempt is made at intelligent distribution of particle data in main memory.

- **LU**: The LU kernel factors a dense matrix into the product of a lower triangular and an upper triangular matrix. The dense $n * n$ matrix A is divided into an $N * N$ array of $B * B$ blocks ($n = NB$) to exploit temporal locality on submatrix elements. To reduce communication, block ownership is assigned using a 2-D scatter decomposition, with blocks being updated by the processors that own them. The block size B should be large enough to keep the cache miss rate low, and small enough to maintain good load balance. Fairly small block sizes ($B = 8$ or $B = 16$) strike a good balance in practice. Elements within a block are allocated contiguously to improve spatial locality benefits, and blocks are allocated locally to processors that own them. See [33] for more details.

- **Ocean**: The Ocean application studies large-scale ocean movements based on eddy and boundary currents, and is an improved version of the Ocean program in SPLASH. The major differences are: (i) it partitions the grids into square-like subgrids rather than groups of columns to improve the communication to computation ratio, (ii) grids are conceptually

represented as 4-D arrays, with all subgrids allocated contiguously and locally in the nodes that own them, and (iii) it uses a red-black Gauss-Seidel multigrid equation solver [8], rather than an SOR solver. See [32] for more details.

- **Radiosity**: This application computes the equilibrium distribution of light in a scene using the iterative hierarchical diffuse radiosity method [13]. A scene is initially modeled as a number of large input polygons. Light transport interactions are computed among these polygons, and polygons are hierarchically subdivided into patches as necessary to improve accuracy. In each step, the algorithm iterates over the current interaction lists of patches, subdivides patches recursively, and modifies interaction lists as necessary. At the end of each step, the patch radiosities are combined via an upward pass through the quadtrees of patches to determine if the overall radiosity has converged. The main data structures represent patches, interactions, interaction lists, the quadtree structures, and a BSP tree which facilitates efficient visibility computation between pairs of polygons. The structure of the computation and the access patterns to data structures are highly irregular. Parallelism is managed by distributed task queues, one per processor, with task stealing for load balancing. No attempt is made at intelligent data distribution. See [25] for more details.

- **Radix**: The integer radix sort kernel is based on the method described in [7]. The algorithm is iterative, performing one iteration for each radix r digit of the keys. In each iteration, a processor passes over its assigned keys and generates a local histogram.The local histograms are then accumulated into a global histogram.Finally, each processor uses the global histogram to permute its keys into a new array for the next iteration. This permutation step requires all-to-all communication. The permutation is inherently a sender-determined one, so keys are communicated through writes rather than reads. See [32] [15] for details.

- **Raytrace**: This application renders a three-dimensional scene using ray tracing. A hierarchical uniform grid (similar to an octree) is used to represent the scene, and early ray termination and antialiasing are implemented, although antialiasing is not used in this study. A ray is traced through each pixel in the image plane, and reflects in unpredictable ways off the objects it strikes. Each contact generates multiple rays, and the recursion

results in a ray tree per pixel. The image plane is partitioned among processors in contiguous blocks of pixel groups, and distributed task queues are used with task stealing. The major data structures represent rays, ray trees, the hierarchical uniform grid, task queues, and the primitives that describe the scene.The data access patterns are highly unpredictable in this application.See [25] for more information.

- **Water-Nsquared**: This application is an improved version of the Water program in SPLASH [26]. This application evaluates forces and potentials that occur over time in a system of water molecules. The forces and potentials are computed using an O(n^2) algorithm (hence the name), and a predictor-corrector method is used to integrate the motion of the water molecules over time. The main difference from the SPLASH program is that the locking strategy in the updates to the accelerations is improved. A process updates a local copy of the particle accelerations as it computes them, and accumulates into the shared copy once at the end.

- **Water-Spatial**: This application solves the same problem as Water- Nsquared, but uses a more efficient algorithm. It imposes a uniform 3-D grid of cells on the problem domain, and uses an O(n) algorithm which is more efficient than Water-Nsquared for large numbers of molecules. The advantage of the grid of cells is that processors which own a cell need only look at neighboring cells to find molecules that might be within the cutoff radius of molecules in the box it owns. The movement of molecules into and out of cells causes cell lists to be updated, resulting in communication.

3.4 Modeling CMP with Multi2Sim

This section describes the common test environment used in this dissertation using Multi2Sim simulator. Details of how the major components of the system are modeled are as follows:

- **Cores**: Evaluations in this thesis use $16, 32, 64$ cores.

- **Memory Hierarchy**[30]: As we can see from Figure 3.1 all evaluations in this thesis use 3 level memory hierarchy- L_1 cache followed by L_2 cache and then *Main Memory* module. Each CPU core has individual L_1 cache, unified for instructions and data, and shared for

every hardware thread if the cores are multithreaded. All L_1 caches share a common L_2 cache. The page replacement policy used is LRU.

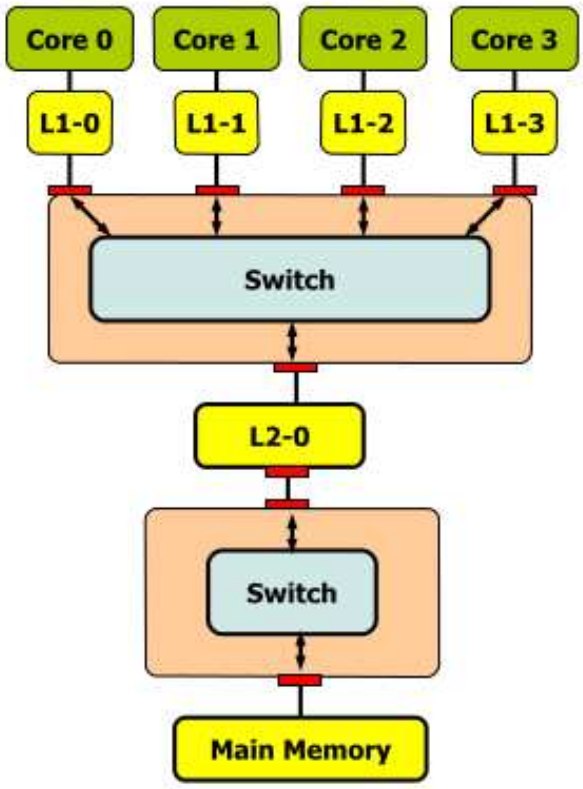

Figure 3.1: Memory Hierarchy Configuration

- **Interconnects**: A default interconnect based on a single switch connects all L_1 caches with a common L_2 cache, and another default network connects the L_2 cache with a single *main memory* module.

Where appropriate, each of the evaluations in the subsequent chapter will elaborate on more specified details of the above components.

Chapter 4

MASI Cache Coherence Protocol

This chapter explores cache coherence using MASI protocol in multiprocessor architecture.

4.1 Motivation

The classical MOESI Cache Coherence Protocol includes the advantages of both 'E' state and 'O' state. So we came up with an idea of combining the two states together to make a 4 state protocol (MASI) which performs better than the classical 4 state protocols. Thus, we can now save 1 bit per block representing its state as opposed to MOESI protocol.

4.2 MASI

In this proposed cache coherence protocol the 'M', 'S', 'I' states have the same definition as those in MSI cache coherence protocol. The newly introduced Advanced(A) state is similar to Owned state with slight features of Exclusive state as well thereby merging the two states to form Advanced state. 'A' state is given the additional responsibility of responding to requests for data, providing a convenient mechanism for selecting a single responder. State of a block in cache becomes 'A' under any of the following situations:

1. Initially, when a block is referenced by a single core only, then the state of the block in cache becomes 'A'(*similar to the 'E' state of MOESI*).

2. Whenever a cache requests for a block which is already present in any cache at the same level of memory in Modified state or in Advanced state, then the cache having the block in 'M' or 'A' state will directly send the copy of the block to the requesting cache and

the state of the block in the requesting cache will now become Advanced and in the cache sending the block it will be changed to Shared state(*almost similar to the 'O' state of MOESI*).

3. Whenever a block in Advanced state, which is dirty with respect to the lower memory level, gets evicted, then the state of one of the remaining sharers of that block (if any) changes to Advanced state.

If a block in 'A' state is a candidate for eviction then:

1. If the block is clean with respect to lower memory level, then there is no need to perform writeback during eviction.

2. If the block is dirty with respect to lower memory level, then:

   (a) If that block has any other sharer, then no writeback occurs during eviction and the state of one of the sharers changes to Advanced state.

   (b) If that block has no other sharer, then it has to perform writeback during eviction.

Table 4.1 shows the state transition for MASI cache coherence protocol.

State	**Processor Action**			**Incoming**	
	Load	**Store**	**Eviction**	**Read Req.**	**Write Req.**
M	Hit	Hit	Writeback →**I**	send data →**S**	send data →**I**
A	Hit	write request →**M**	writeback (response: dirty and no sharers) →**I** or Silent Evict (response: clean or sharers present) →**I**	send data →**S**	send data →**I**
S	Hit	write request →**M**	Silent Evict →**I**	(none)	(none) →**I**
I	Read request →A	write request →**M**	(none)	(none)	(none)

Table 4.1: State Transitions for MASI cache coherence protocol

4.3 Evaluation

We evaluate the performance of our MASI cache coherence protocol(Section 4.2) and compare its performance with classical MOSI cache coherence protocol(Section 2.2.3) and MOESI cache coherence protocol(Section 2.2.4). To improve the performance of MASI protocol, we experimented with different definitions of 'A' state which are discussed in the following subsections.

4.3.1 MOSI vs MASI : With forwardings and writebacks only

Since our performance metrics are forwardings and writebacks, in this subsection, we have calculated L_2 to L_1 forwardings and L_1 to L_2 writebacks and then calculated the net performance gain of MASI over MOSI. In this section we have not taken the complete definition of Advanced (A) state. There are two changes in the definition of 'A' state:

1. 'A' state can occur only in the first 2 situations as described in Section 4.2 .
2. Whenever a block in 'A' state is a candidate for eviction, it performs writeback and gets evicted.

Appendix E shows the control flow of MOSI and MASI cache coherence protocols as imple-mented in Multi2Sim for this subsection.

4.3.1.1 Test Environment and Parameters

- **Operating System**: Ubuntu 12.04LTS (64 bit)
- **Number of Cores used**: 16, 32
- **Cache Structure**:
  - L_1 cache: Sets = 64, Associativity = 8, Policy = LRU, Block Size = 256, Latency = 2 cycles, Ports = 2
  - L_2 cache: Sets = 128, Associativity = 8, Policy = LRU, Block Size = 256, Latency = 20 cycles, Ports = 2
  - ratio of read to write latency = 0.1

- Let,

  x = Average no. of forwardings per $100,000$ access in MOSI

  y = Average no. of writebacks per $100,000$ access in MOSI

  u = Average no. of forwardings per $100,000$ access in MASI

  v = Average no. of writebacks per $100,000$ access in MASI

$$NetImprovement = \frac{(0.1x + y) - (0.1u + v)}{0.1x + y}$$

4.3.1.2 Performance

In this section, we present the results obtained from the simulations of the programs of Splash2 Benchmark. Figures 4.1, 4.2 represents the results and Table 4.2 shows the net improvement for 16 processors. Figures 4.3, 4.4 represents the results and Table 4.3 shows the net improvement for 32 processors. For more details, refer to the Appendix A.

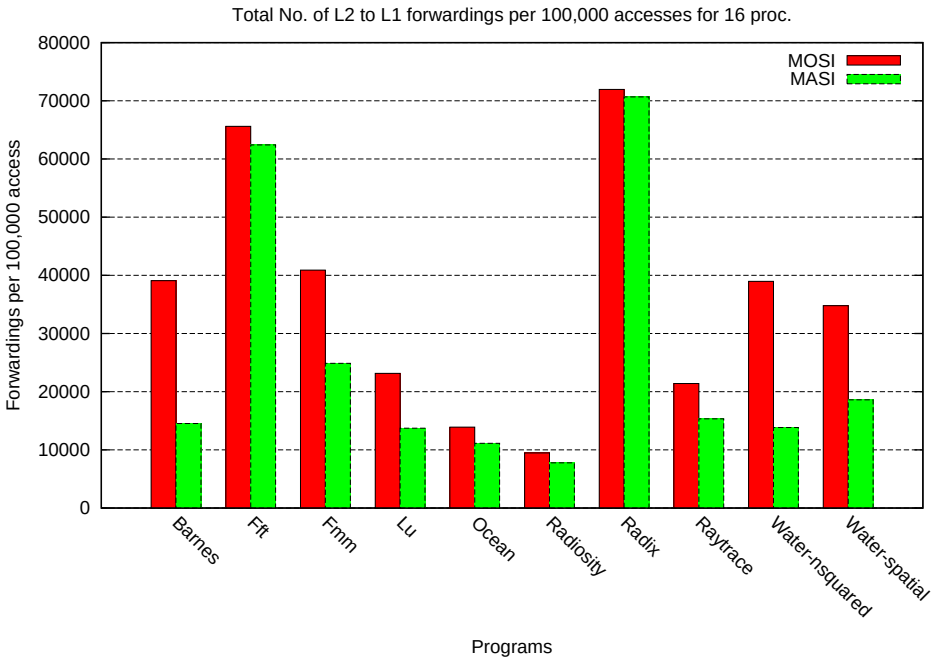

Figure 4.1: Normalized forwardings for 16 processors

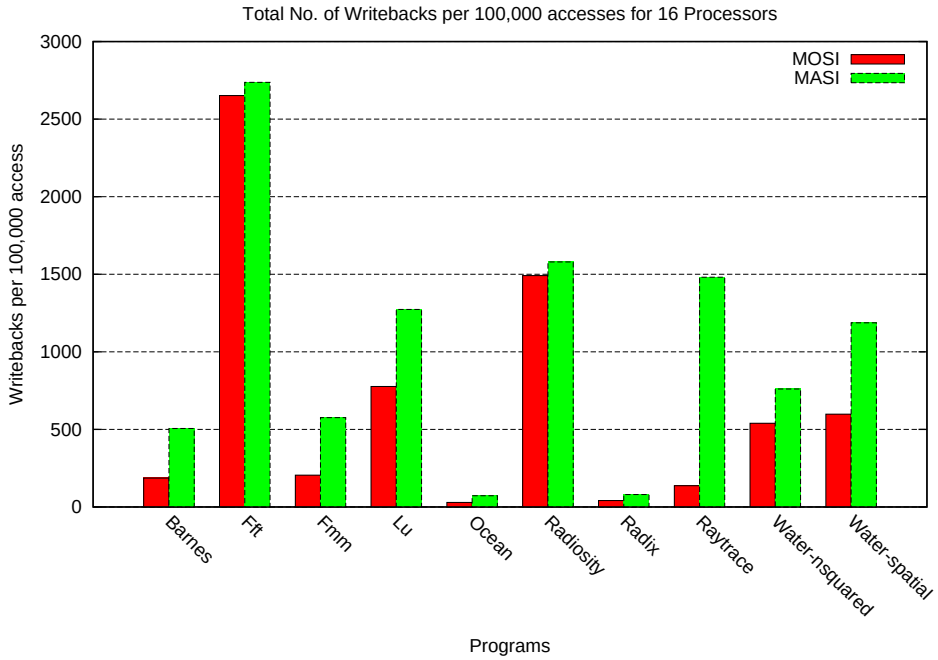

Figure 4.2: Normalized writebacks for 16 processors

Program	$0.1*$**Average no. of forwardings per** $100,000$ **accesses in MOSI** + **Average no. of writebacks per** $100,000$ **accesses in MOSI**	$0.1*$**Average no. of forwardings per** $100,000$ **accesses in MASI** + **Average no. of writebacks per** $100,000$ **accesses in MASI**	**Percentage improvement**
Barnes	4092.866807	1958.905507	**52.138547**
Fft	9213.528875	8979.278863	**2.542457**
Fmm	4293.806785	3062.738412	**28.670791**
Lu	3090.744973	2645.186901	**14.41588**
Ocean	1419.396134	1185.92334	**16.448741**
Radiosity	2437.907247	2357.281177	**3.307184**
Radix	7237.458066	7145.592155	**1.269312**
Raytrace	2277.155567	3014.135199	**-32.364044**
Water-nsquared	4435.069353	2143.91996	**51.659832**
Water-spatial	4075.65382	3047.216289	**25.233682**

Table 4.2: Net Improvement for 16 processors

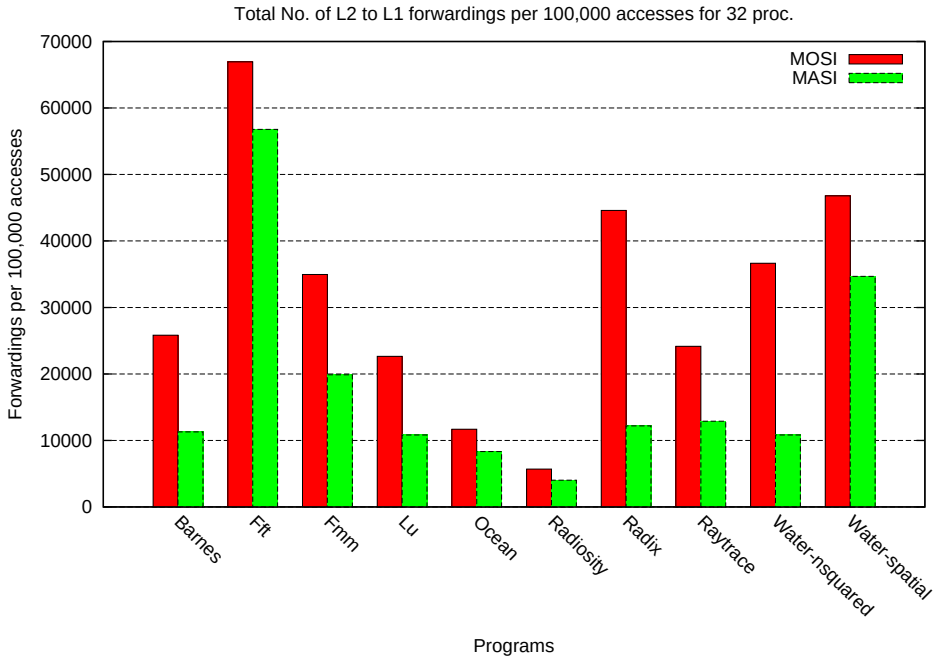

Figure 4.3: Normalized forwardings for 32 processors

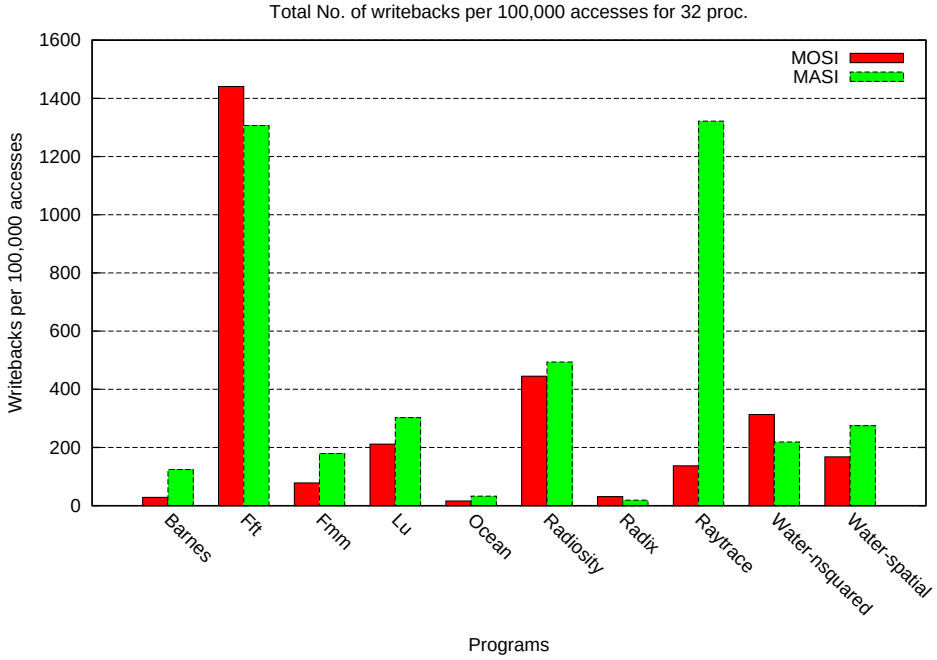

Figure 4.4: Normalized writebacks for 32 processors

Program	0.1∗**Average no. of forwardings per** 100,000 **accesses in MOSI + Average no. of writebacks per** 100,000 **accesses in MOSI**	0.1∗**Average no. of forwardings per** 100,000 **accesses in MASI+ Average no. of writebacks per** 100,000 **accesses in MASI**	**Percentage improvement**
Barnes	2611.56541	1252.63127	**52.035233**
Fft	8135.230513	6985.450213	**14.133346**
Fmm	3574.809131	2166.236208	**39.402745**
Lu	2477.165055	1384.982199	**44.090032**
Ocean	1184.687134	866.532821	**26.855556**
Radiosity	1014.79369	896.71476	**11.635757**
Radix	4490.798193	1238.381314	**72.424027**
Raytrace	2553.774042	2612.167954	**-2.286573**
Water-nsquared	3977.363424	1301.55014	**67.276057**
Water-spatial	4845.612566	3741.706421	**22.78156**

Table 4.3: Net Improvement for 32 processors

4.3.2 MOSI vs MASI: With forwardings and writebacks (dirty bit included)

Since our performance metrics are forwardings and writebacks, in this subsection, we have calculated L_2 to L_1 forwardings and L_1 to L_2 writebacks by implementing dirty bit in the directory structure of MOSI and MASI cache coherence protocols and then calculated the net performance improvement of MASI over MOSI. In this section also, we have not taken the complete definition of Advanced (A) state. There are two changes in the definition of 'A' state:

1. 'A' state can occur only in the first 2 situations as described in Section 4.2 .
2. Whenever a block in 'A' state is a candidate for eviction, it performs writeback only if it is dirty with respect to the lower memory level.

Appendix F shows the control flow of MOSI and MASI cache coherence protocols as imple-mented in Multi2Sim for this subsection.

4.3.2.1 Test Environment and Parameters

- **Operating System**: Ubuntu 12.04LTS (64 bit)
- **Number of Cores used**: 16, 32
- **Cache Structure**:

- L_1 cache: Sets = 64, Associativity = 8, Policy = LRU, Block Size = 256 , Latency = 2 cycles, Ports = 2
- L_2 cache: Sets = 128, Associativity = 8,Policy = LRU, Block Size = 256, Latency = 20 cycles, Ports = 2
- ratio of read to write latency = 0.1

- Let,

  x = Average no. of forwardings per $100,000$ access in MOSI

  y = Average no. of writebacks per $100,000$ access in MOSI

  u = Average no. of forwardings per $100,000$ access in MASI

  v = Average no. of writebacks per $100,000$ access in MASI

$$NetImprovement = \frac{(0.1x + y) - (0.1u + v)}{0.1x + y}$$

4.3.2.2 Performance

In this section, we present the results obtained from the simulations of the programs of Splash2 Benchmark. Figures 4.5, 4.6 represents the results and Table 4.4 shows the net improvement for 16 processors. Figures 4.7, 4.8 represents the results and Table 4.5 shows the net improvement for 32 processors. For more details, refer to the Appendix B.

4.3.3 MOSI vs MASI: With forwardings, writebacks (dirty bit included) and new eviction policy

In this third experiment, we have used the final definition of Advanced (A) state as defined in Section 4.2 and compared the performance with MOSI protocol which has dirty bit included in its directory. Appendix G shows the control flow of MASI cache coherence protocol as implemented in Multi2Sim for this subsection. For MOSI cache coherence protocol, refer to Appendix F.

4.3.3.1 Test Environment and Parameters

- **Operating System**: Ubuntu 12.04LTS (64 bit)

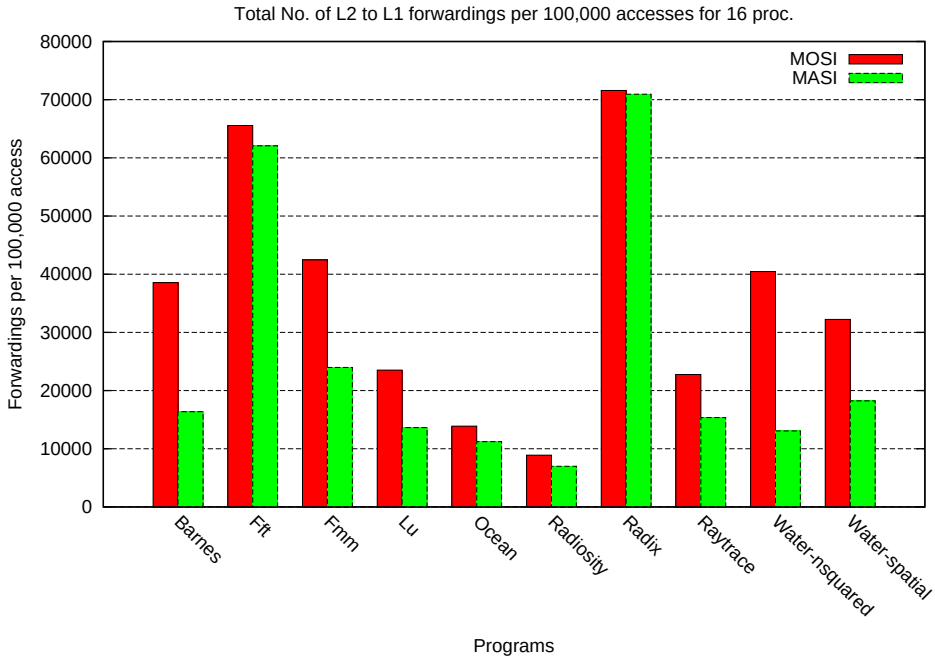

Figure 4.5: Normalized forwardings for 16 processors with dirty bit only

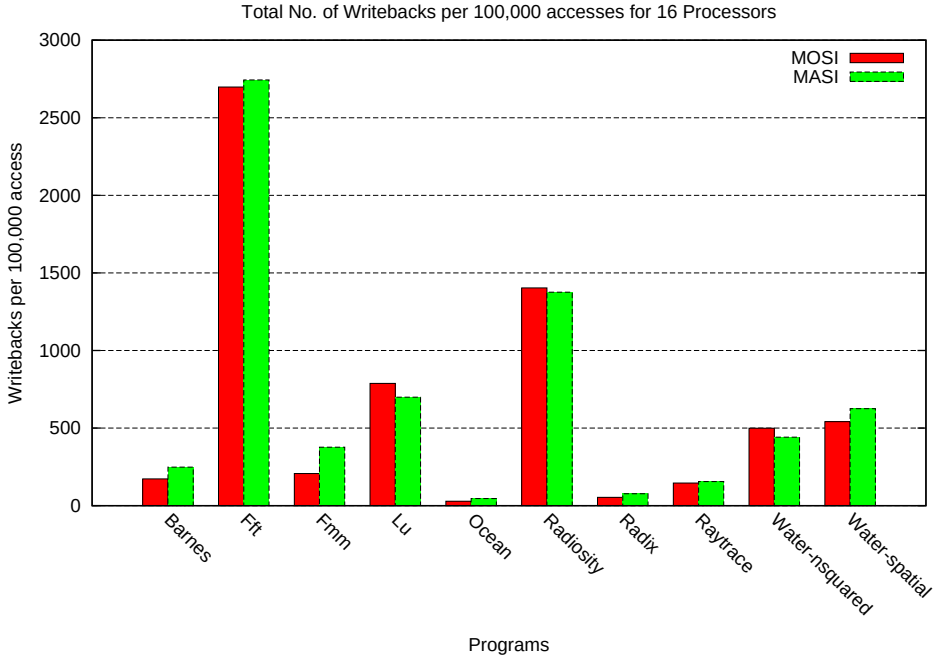

Figure 4.6: Normalized writebacks for 16 processors with dirty bit only

Program	$0.1*$**Average no. of forwardings per** $100,000$ **accesses in MOSI** + **Average no. of writebacks per** $100,000$ **accesses in MOSI**	$0.1*$**Average no. of forwardings per** $100,000$ **accesses in MASI** + **Average no. of writebacks per** $100,000$ **accesses in MASI**	**Percentage improvement**
Barnes	4026.830721	1884.139528	**53.210362**
Fft	9254.015421	8949.555248	**3.290033**
Fmm	4453.214054	2773.750908	**37.713506**
Lu	3139.337125	2062.166165	**34.312051**
Ocean	1416.261509	1167.355901	**17.574834**
Radiosity	2293.558991	2073.931468	**9.575839**
Radix	7210.882372	7172.128277	**0.537439**
Raytrace	2422.344116	1689.523662	**30.252533**
Water-nsquared	4545.081212	1746.544325	**61.572869**
Water-spatial	3764.185585	2447.700967	**34.973956**

Table 4.4: Net Improvement for 16 processors with dirty bit only

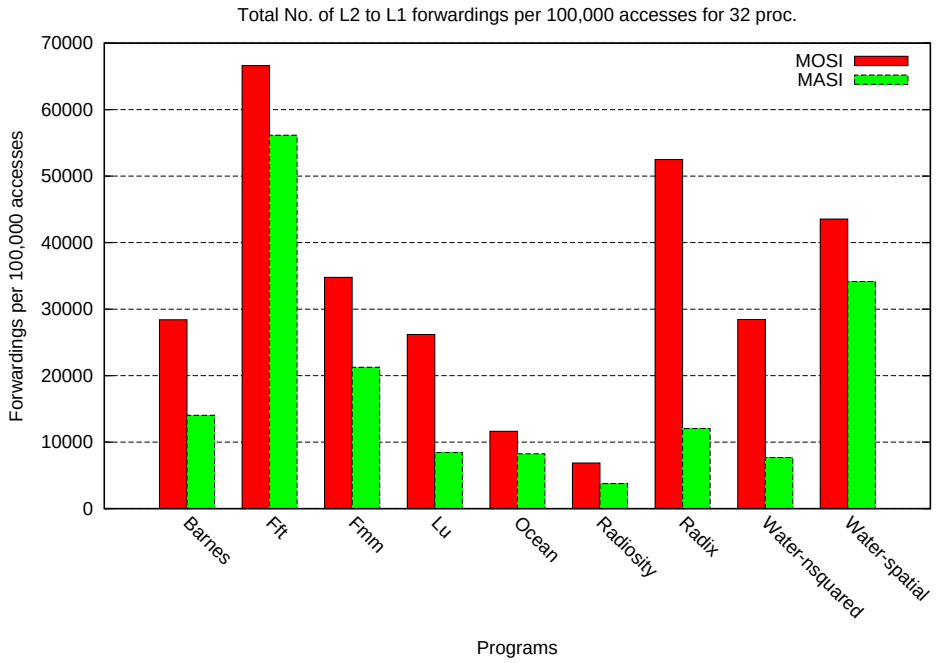

Figure 4.7: Normalized forwardings for 32 processors with dirty bit only

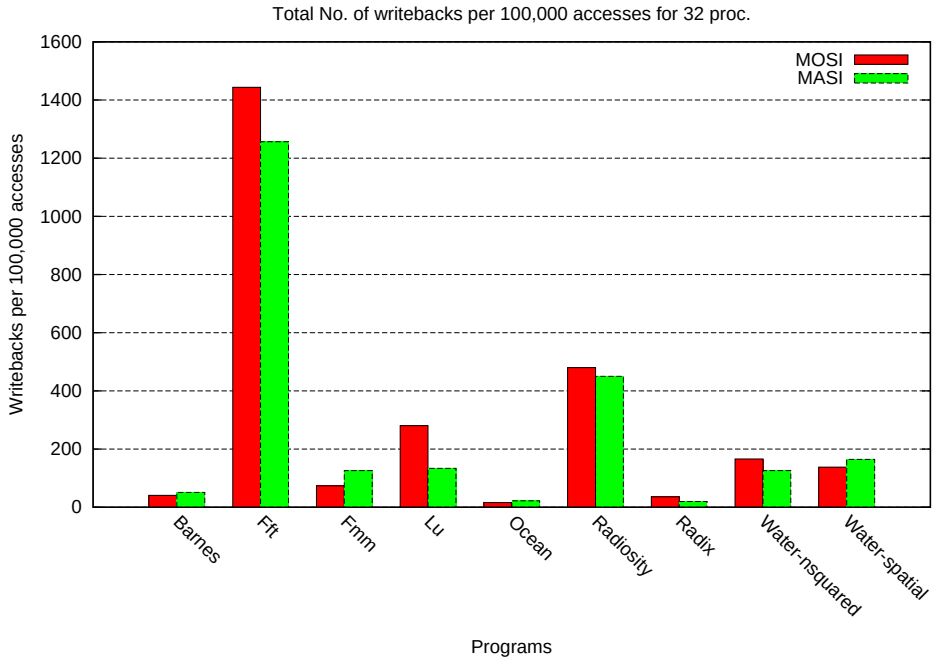

Figure 4.8: Normalized writebacks for 32 processors with dirty bit only

Program	$0.1*$**Average no. of forwardings per** $100,000$ **accesses in MOSI** + **Average no. of writebacks per** $100,000$ **accesses in MOSI**	$0.1*$**Average no. of forwardings per** $100,000$ **accesses in MASI** + **Average no. of writebacks per** $100,000$ **accesses in MASI**	**Percentage improvement**
Barnes	2880.295095	1454.189048	**49.512498**
Fft	8106.817144	6873.111344	**15.218128**
Fmm	3550.570023	2251.934704	**36.575404**
Lu	2898.581659	977.89851	**66.262861**
Ocean	1179.177959	847.077299	**28.163744**
Radiosity	1165.123834	825.877908	**29.116727**
Radix	5285.905309	1224.084282	**76.842486**
Water-nsquared	3010.262123	893.261645	**70.326118**
Water-spatial	4492.844979	3579.962954	**20.318574**

Table 4.5: Net Improvement for 32 processors with dirty bit only

- **Number of Cores used**: $16, 32, 64$ for 3 experiments
- **Cache Structure**:
  - L_1 cache: Sets = 64, Associativity = 8, Policy = LRU, Block Size = 256 , Latency = 2 cycles, Ports = 2
  - L_2 cache: Sets = 128, Associativity = 8,Policy = LRU, Block Size = 256, Latency = 20 cycles, Ports = 2
  - ratio of read to write latency = 0.1
- Let,

  x = Average no. of forwardings per $100,000$ access in MOSI

  y = Average no. of writebacks per $100,000$ access in MOSI

  u = Average no. of forwardings per $100,000$ access in MASI

  v = Average no. of writebacks per $100,000$ access in MASI

$$NetImprovement = \frac{(0.1x + y) - (0.1u + v)}{0.1x + y}$$

4.3.3.2 Performance

In this section, we present the results obtained from the simulations of the programs of Splash2 Benchmark. Figures 4.9, 4.10 represents the results and Table 4.6 shows the net improvement for 16 processors. Figures 4.11, 4.12 represents the results and Table 4.7 shows the net improvement for 32 processors. Figures 4.13, 4.14 represents the results and Table 4.8 shows the net improvement for 64 processors. For more details, refer to the Appendix C.

4.3.4 MOESI vs. MASI - With forwardings, writebacks (dirty bit included) and new eviction policy

In this fourth experiment, we have used the final definition of Advanced (A) state as defined in Section 4.2 and compared the performance with MOESI protocol which has dirty bit included in its directory. Appendix H shows the control flow of MOESI cache coherence protocol as

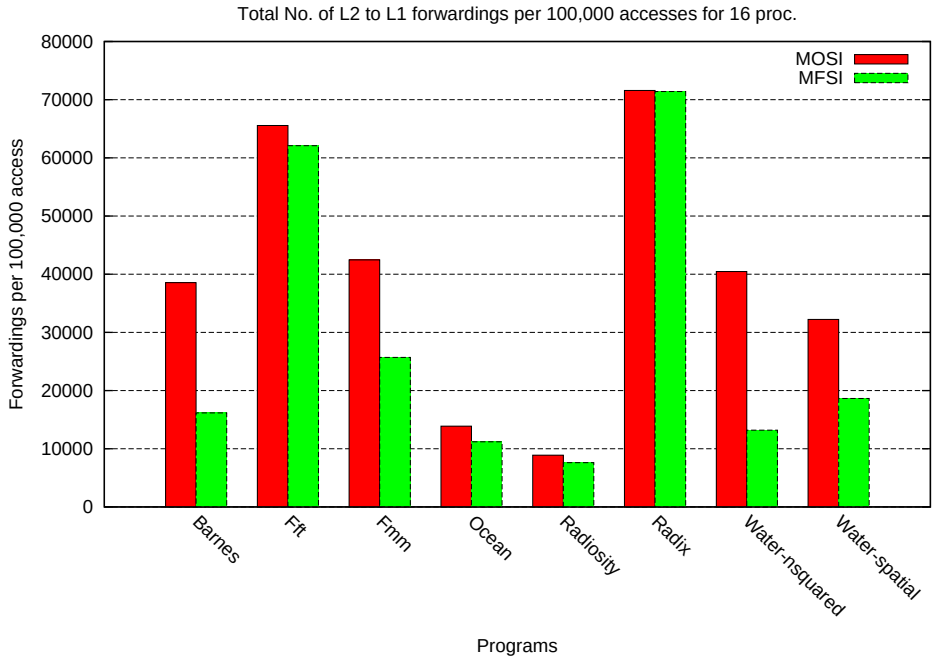

Figure 4.9: Normalized forwardings for 16 processors with new eviction policy

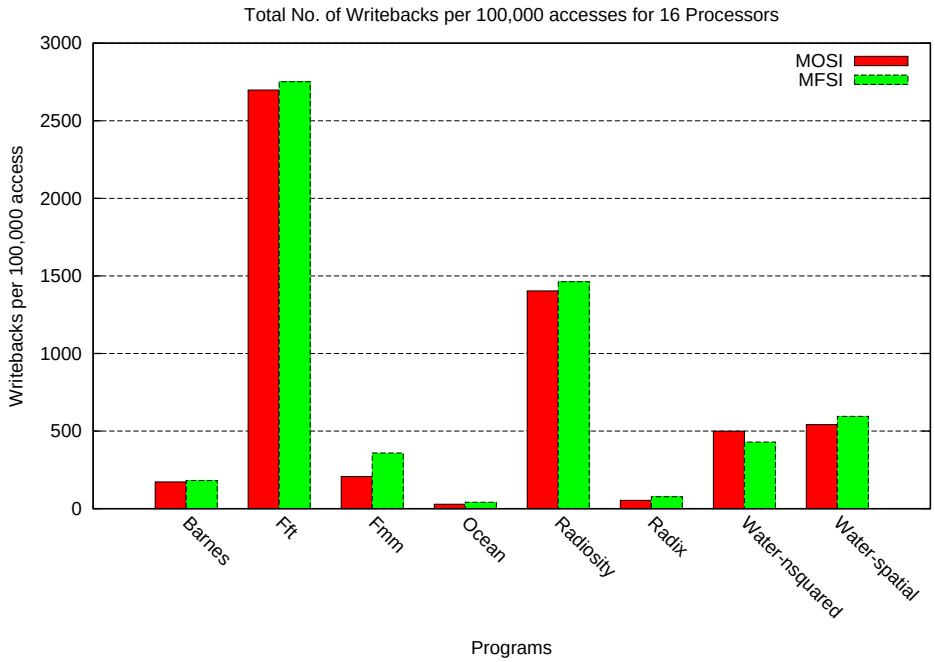

Figure 4.10: Normalized writebacks for 16 processors with new eviction policy

Program	0.1∗**Average no. of forwardings per** 100,000 **accesses in MOSI + Average no. of writebacks per** 100,000 **accesses in MOSI**	0.1∗**Average no. of forwardings per** 100,000 **accesses in MASI + Average no. of writebacks per** 100,000 **accesses in MASI**	**Percentage improvement**
Barnes	4026.830721	1797.049173	**55.373114**
Fft	9254.015421	8960.244506	**3.174524**
Fmm	4453.214054	2928.996387	**34.227361**
Ocean	1416.26151	1160.028345	**18.092221**
Radiosity	2293.559723	2223.782533	**3.04231**
Radix	7210.882372	7216.884422	**-0.083236**
Water-nsquared	4545.081213	1747.82031	**61.544795**
Water-spatial	3764.185585	2460.705106	**34.628486**

Table 4.6: Net Improvement for 16 processors with new eviction policy

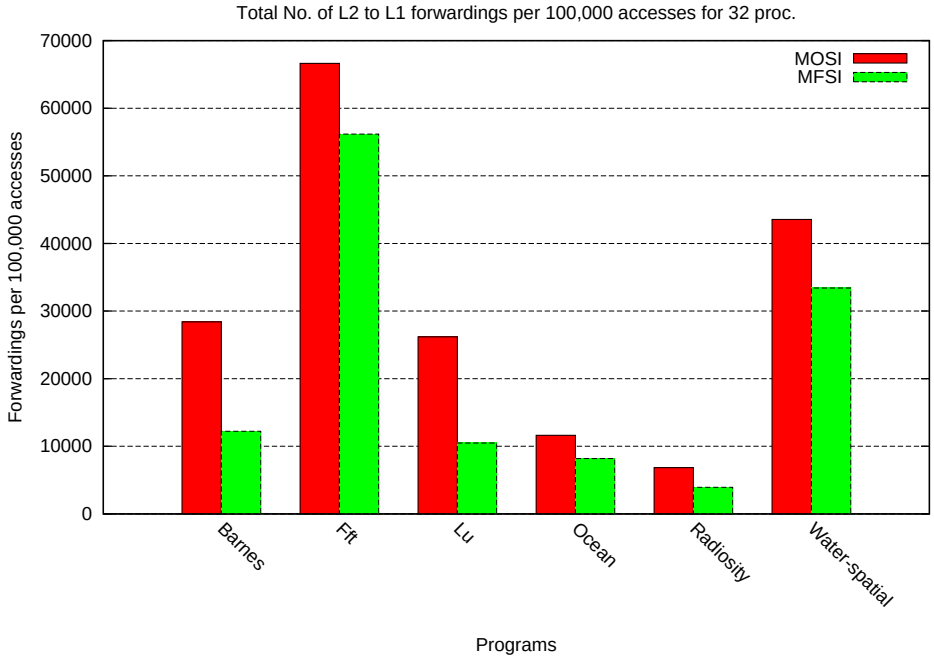

Figure 4.11: Normalized forwardings for 32 processors with new eviction policy

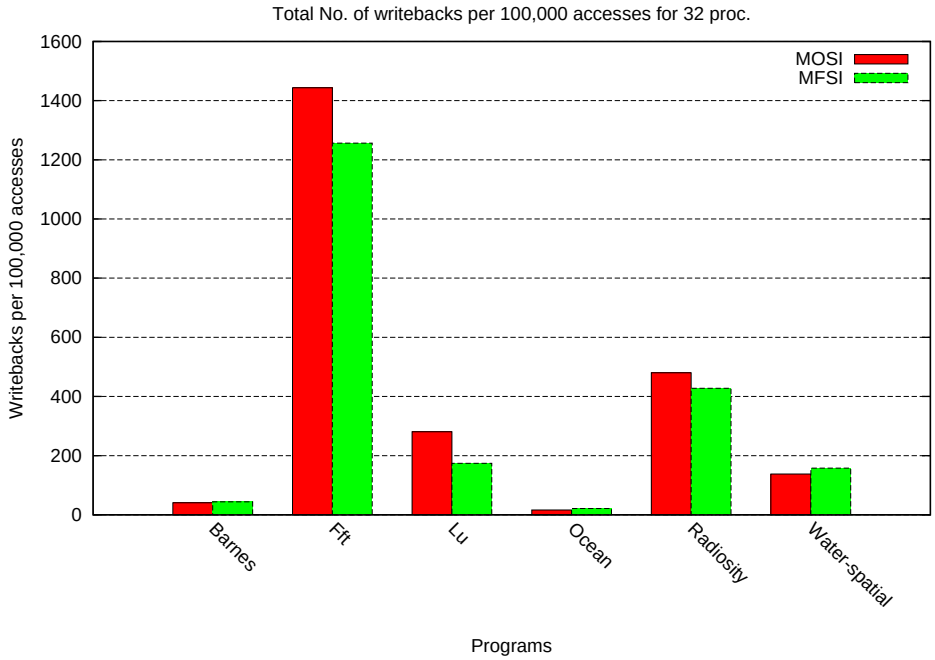

Figure 4.12: Normalized writebacks for 32 processors with new eviction policy

Program	0.1∗**Average no. of forwardings per** 100,000 **accesses in MOSI + Average no. of writebacks per** 100,000 **accesses in MOSI**	0.1∗**Average no. of forwardings per** 100,000 **accesses in MASI + Average no. of writebacks per** 100,000 **accesses in MASI**	**Percentage improvement**
Barnes	2880.295095	1266.874827	**56.015798**
Fft	8106.817144	6872.919283	**15.220497**
Lu	2898.581659	1222.799655	**57.813862**
Ocean	1179.177959	840.691221	**28.705314**
Radiosity	1165.123834	820.5902	**29.570559**
Water-spatial	4492.84498	3498.592913	**22.129677**

Table 4.7: Net Improvement for 32 processors with new eviction policy

Program	0.1∗**Average no. of forwardings per** 100,000 **accesses in MOSI + Average no. of writebacks per** 100,000 **accesses in MOSI**	0.1∗**Average no. of forwardings per** 100,000 **accesses in MASI + Average no. of writebacks per** 100,000 **accesses in MASI**	**Percentage improvement**
Fft	4707.206793	4004.033899	**14.93822**
Lu	2061.066877	613.81304	**70.218674**
Ocean	1622.590742	1086.066058	**33.065928**
Water-spatial	4589.408748	3442.735567	**24.985205**

Table 4.8: Net Improvement for 64 processors with new eviction policy

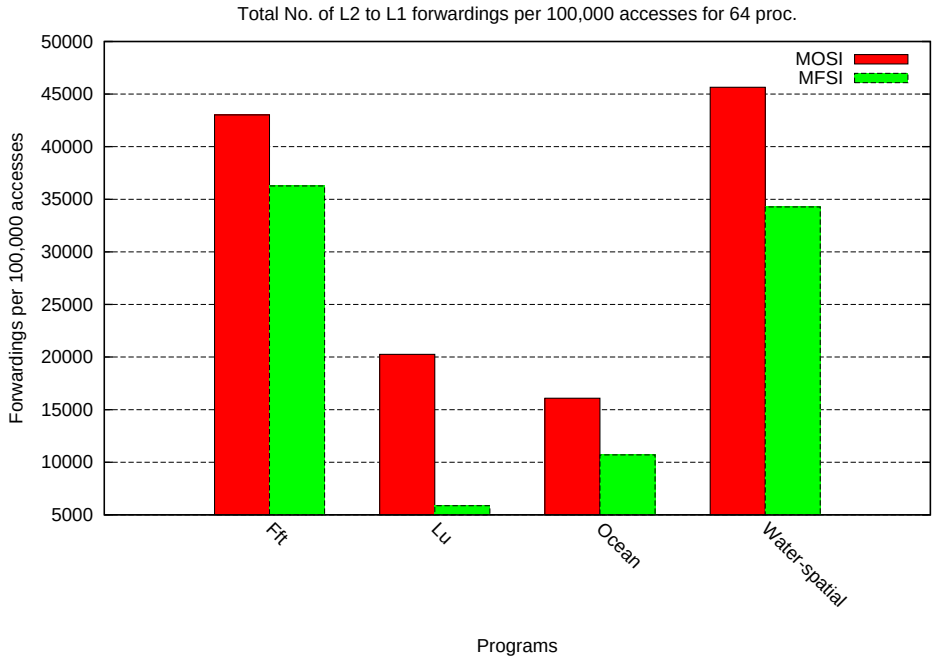

Figure 4.13: Normalized forwardings for 64 processors with new eviction policy

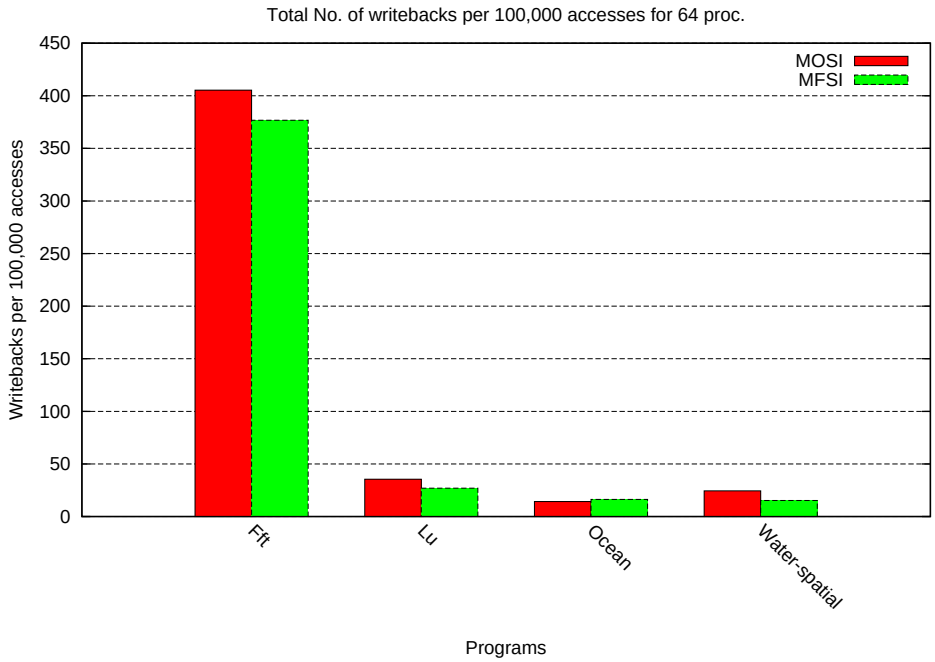

Figure 4.14: Normalized writebacks for 64 processors with new eviction policy

implemented in Multi2Sim for this subsection. For MASI cache coherence protocol, refer to Appendix G.

4.3.4.1 Test Environment and Parameters

- **Operating System**: Ubuntu 12.04LTS (64 bit)
- **Number of Cores used**: $16, 32$ for 2 experiments
- **Cache Structure**:
  - L_1 cache: Sets = 64, Associativity = 8, Policy = LRU, Block Size = 256 , Latency = 2 cycles, Ports = 2
  - L_2 cache: Sets = 128, Associativity = 8,Policy = LRU, Block Size = 256, Latency = 20 cycles, Ports = 2
  - ratio of read to write latency = 0.1
- Let,

  x = Average no. of forwardings per $100,000$ access in MOSI

  y = Average no. of writebacks per $100,000$ access in MOSI

  u = Average no. of forwardings per $100,000$ access in MASI

  v = Average no. of writebacks per $100,000$ access in MASI

$$NetImprovement = \frac{(0.1x + y) - (0.1u + v)}{0.1x + y}$$

4.3.4.2 Performance

In this section, we present the results obtained from the simulations of the programs of Splash2 Benchmark. Figures 4.15, 4.16 represents the results and Table 4.9 shows the net improvement for 16 processors. Figures 4.17, 4.18 represents the results and Table 4.10 shows the net improvement for 32 processors. For more details, refer to the Appendix D.

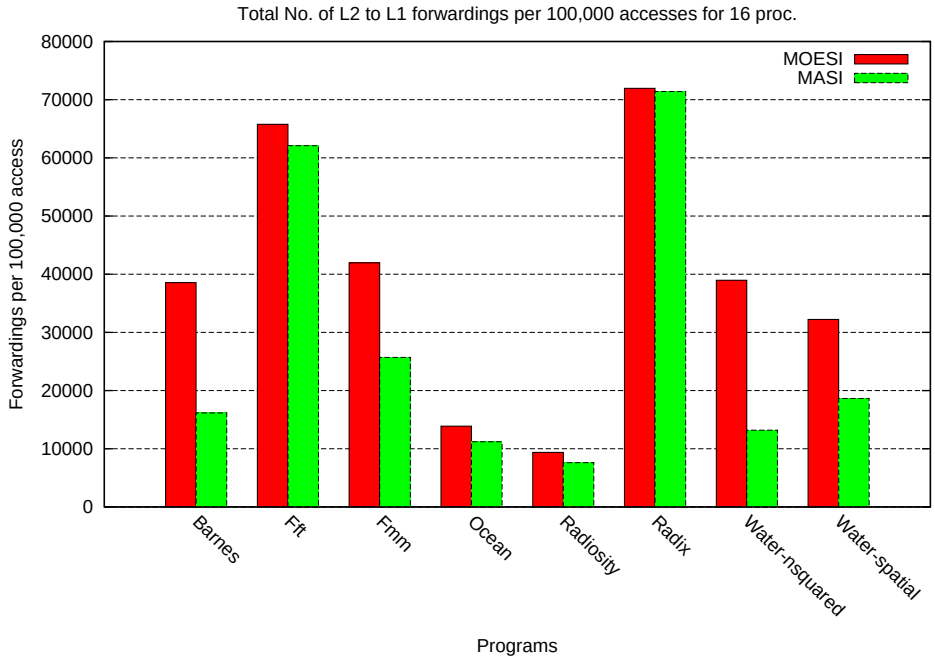

Figure 4.15: Normalized forwardings for 16 processors new eviction policy and dirty MOESI

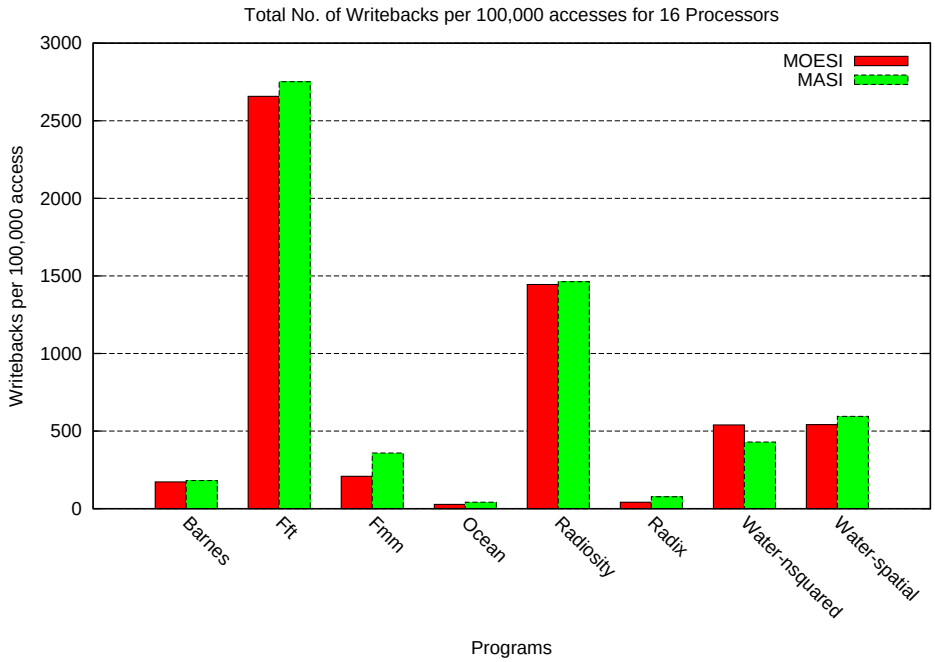

Figure 4.16: Normalized writebacks for 16 processors new eviction policy and dirty MOESI

Program	$0.1*$**Average no. of forwardings per** $100,000$ **accesses in MOESI + Average no. of writebacks per** $100,000$ **accesses in MOESI**	$0.1*$**Average no. of forwardings per** $100,000$ **accesses in MASI + Average no. of writebacks per** $100,000$ **accesses in MASI**	**Percentage improvement**
Barnes	4026.918793	1797.049173	**55.37409**
Fft	9234.79717	8960.244506	**2.973023**
Fmm	4403.164449	2928.996387	**33.479741**
Ocean	1416.836465	1160.028345	**18.12546**
Radiosity	2382.584281	2223.782533	**6.665105**
Radix	7237.458066	7216.884422	**0.284266**
Water-nsquared	4435.069352	1747.82031	**60.590914**
Water-spatial	3763.915505	2460.705106	**34.623795**

Table 4.9: Net Improvement for 16 processors new eviction policy and dirty MOESI

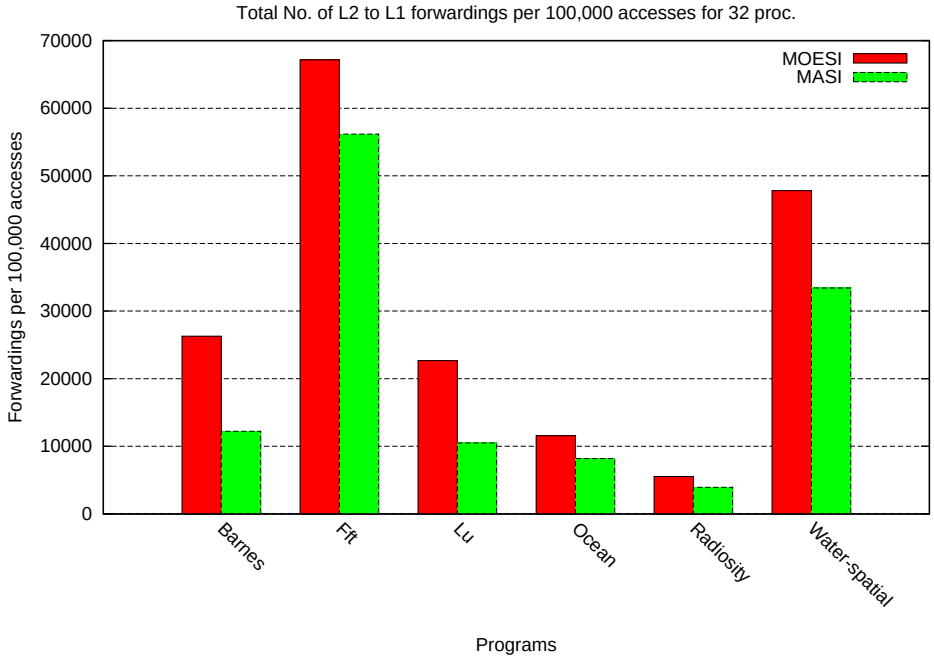

Figure 4.17: Normalized forwardings for 32 processors new eviction policy and dirty MOESI

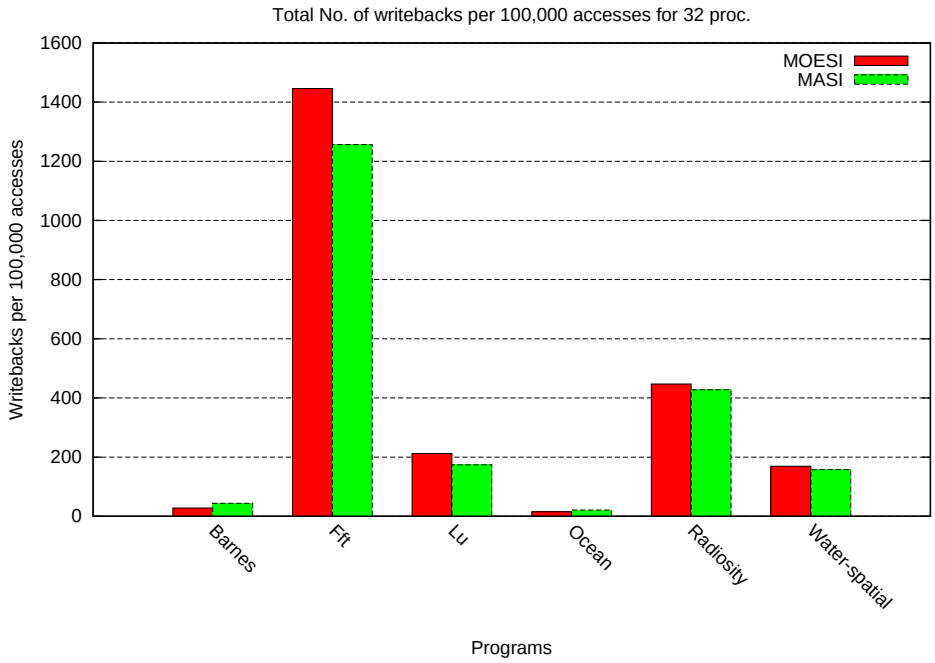

Figure 4.18: Normalized writebacks for 32 processors new eviction policy and dirty MOESI

Program	$0.1*$**Average no. of forwardings per** $100,000$ **accesses in MOESI** + **Average no. of writebacks per** $100,000$ **accesses in MOESI**	$0.1*$**Average no. of forwardings per** $100,000$ **accesses in MASI** + **Average no. of writebacks per** $100,000$ **accesses in MASI**	**Percentage improvement**
Barnes	2655.829108	1266.874827	**52.29833**
Fft	8161.026693	6872.919283	**15.783644**
Lu	2479.456586	1222.799655	**50.682756**
Ocean	1172.990117	840.691221	**28.329215**
Radiosity	1000.793953	820.5902	**18.006079**
Water-spatial	4949.064265	3498.592913	**29.307992**

Table 4.10: Net Improvement for 32 processors new eviction policy and dirty MOESI

4.3.5 Summary of Evaluation

The following list summarizes some of the key findings from our evaluation of MASI cache coherence protocol:

- Number of *Normalized forwardings* in our proposed MASI cache coherence protocol is comparatively *lesser* than that in classical MOSI and MOESI.
- Number of *Normalized writebacks* in our proposed MASI cache coherence protocol have decreased as we have improved the definition of 'A' state in successive experiments.
- The proposed MASI cache coherence protocol performs better than the classical MOSI and MOESI protocols.

4.4 Related Work

- **Firefly protocol**: The Firefly cache coherence protocol is the schema used in the DEC Firefly multiprocessor workstation, developed by DEC Systems Research Center. This protocol uses a write-through policy.In this protocol, the following states can be assigned to each block:
  - Valid-Exclusive: This block has a coherent copy of the memory. There is only one copy of the data in caches.
  - Shared: This block has a coherent copy of the memory. The data may be possibly shared, but its content is not modified.
  - Dirty: The block is the only copy of the memory and it is incoherent. This is the only state that generates a write-back when the block is replaced in the cache.

  These states correspond to the Exclusive, Shared, and Modified states of the MESI protocol. This protocol never causes invalidation, so the Invalid state is not listed here.[5]
- **Dragon protocol**: The Dragon cache coherence protocol is the schema used in the Xerox Dragon multiprocessor workstation, developed by Xerox PARC. This protocol uses a write-back policy. In this protocol, the following states can be assigned to each block:

- Invalid: No data in cache block.
- Clean: Clean, only copy.
- Shared-Clean: Clean, might be shared.
- Dirty: Modified, only copy.
- Shared-Dirty: Modified, might be shared. This implies that there might be up-to-date other copies of the data (in Shared-Clean state) but that the memory copy is not up-to-date.

These five states correspond to the five states of the MOESI protocol, although they are listed above in IESMO order.[19]

- **MERSI protocol**: The protocol consists of five states, Modified (M), Exclusive (E), Read Only or Recent (R), Shared (S) and Invalid (I). The M, E, S and I states are the same as in the MESI protocol. The R state is similar to the E state in that it is constrained to be the only clean, valid, copy of that data in the computer system. Unlike the E state, the processor is required to initially request ownership of the cache line in the R state before the processor may modify the cache line and transition to the M state [12] [22].

- **Modified- KEIO protocol**: It provides the following six states: Invalid(I), Clean-Exclusive-Owner(CEO), Clean-Shared-Owner(CSO), Dirty-Exclusive-Owner(DEO), Dirty-Shared-Owner(DSO), Shared-non-Owner(S).[28]

  Principle:

  - If no cache contains the line, memory owns it.
  - Only the owner can supply the cacheline to other caches.
  - Reading a line from another cache does not change ownership.
  - The cache reading a line from memory becomes the owner.
  - Writing a line owned by another cache, changes ownership.
  - A dirty line is written back to memory by the owner when the cacheline is flushed. Memory is then owner.

- **MESIF protocol**: It is an extension of MESI protocol. This protocol adds a different fifth state known as Forward. The Forward state is a specialized form of the shared

state,This state is available in only one of the caches that have shared data. It indicates that this cache should act as the designated responder for any requests for the memory line.This protocol ensures that, if any cache holds the line in Shared state, at most one cache holds it in Forward state. Thus, it helps save bus operations in a way that only one processor has to respond to the requests for memory line.It may also save memory accesses [16] [29] [10] [20].

- **Improved-MOESI**: The states in this protocol have the same definition like the states in classic MOESI. But Improved-MOESI differs from classic MOESI in its operation. The Owned state in Improved-MOESI differs from classic MOESI in the sense that whenever a cache requests for a block which is already present in cache of other processor in Owned state, then the cache having the block in Owned state will directly send the copy of the block to requesting cache and the state of the block in the requesting cache will become Owned and in the cache sending the block it will be changed to Shared state [4].

Chapter 5

Conclusion

The transition to multicore processors places great focus and importance on memory system design. The cache coherence mechanisms are a key component toward achieving the goal of continuing exponential performance growth through widespread thread-level parallelism. This dissertation makes several contributions in the space of cache coherence for multicore chips. In this chapter, we summarize the contributions. Related Works are already discussed in the Section 4.4.

Our 4 state cache coherence protocol (MASI) performs better than the classical 4 state protocols and has comparable performance with 5 state MOESI protocol. We can now save 1 bit per block representing its state as opposed to MOESI protocol. Since number of on-chip processors are increasing rapidly, our 4 state MASI protocol will be much more cost e ective (*in terms of saving bits to represent the state of block*).

References

[1] *Dual Core Era Begins, PC Makers Start Selling Intel-Based PCs.* http://www.intel.com/pressroom/archive/releases/20050418comp.htm, Apr. 2005.

[2] Sarita V Adve and Kourosh Gharachorloo. Shared memory consistency models: A tutorial. *computer*, 29(12):66–76, 1996.

[3] Vikas Agarwal, MS Hrishikesh, Stephen W Keckler, and Doug Burger. *Clock rate versus IPC: The end of the road for conventional microarchitectures*, volume 28. ACM, 2000.

[4] Hesham Altwaijry and Diyab S Alzahrani. Improved-moesi cache coherence protocol. *Arabian Journal for Science and Engineering*, pages 1–10, 2013.

[5] James Archibald and Jean-Loup Baer. Cache coherence protocols: Evaluation using a multiprocessor simulation model. *ACM Transactions on Computer Systems (TOCS)*, 4(4):273–298, 1986.

[6] David H Bailey. Ffts in external of hierarchical memory. In *Proceedings of the 1989 ACM/IEEE conference on Supercomputing*, pages 234–242. ACM, 1989.

[7] Guy E Blelloch, Charles E Leiserson, Bruce M Maggs, C Greg Plaxton, Stephen J Smith, and Marco Zagha. A comparison of sorting algorithms for the connection machine cm-2. In *Proceedings of the third annual ACM symposium on Parallel algorithms and architectures*, pages 3–16. ACM, 1991.

[8] Achi Brandt. Multi-level adaptive solutions to boundary-value problems. *Mathematics of computation*, 31(138):333–390, 1977.

[9] A Micro Devices. Amd64 architecture programmers manual volume 2: System programming, 2006.

[10] James R Goodman and Herbert HJ Hum. Forward state for use in cache coherency in a multiprocessor system, July 26 2005. US Patent 6,922,756.

[11] Leslie Greengard. *The rapid evaluation of potential fields in particle systems.* MIT press, 1988.

[12] Edward T Grochowski and Vinod Sharma. Method and apparatus for maintaining cache coherence in a computer system, February 15 2005. US Patent 6,857,051.

[13] Pat Hanrahan, David Salzman, and Larry Aupperle. A rapid hierarchical radiosity algorithm. In *ACM SIGGRAPH Computer Graphics*, volume 25, pages 197–206. ACM, 1991.

[14] John L Hennessy and David A Patterson. *Computer architecture: a quantitative approach.* Elsevier, 2012.

[15] Chris Holt, Mark Heinrich, Jaswinder Pal Singh, Edward Rothberg, and John Hennessy. *The effects of latency, occupancy, and bandwidth in distributed shared memory multiprocessors*. Computer Systems Laboratory, Stanford University, 1995.

[16] David Kanter. The common system interface: Intels future interconnect. *Real World Technologies*, 2007.

[17] Poonacha Kongetira, Kathirgamar Aingaran, and Kunle Olukotun. Niagara: A 32-way multithreaded sparc processor. *Micro, IEEE*, 25(2):21–29, 2005.

[18] Milo MK Martin. *Token coherence*. PhD thesis, UNIVERSITY OF WISCONSIN, 2003.

[19] Edward M McCreight. The dragon computer system. In *Microarchitecture of VLSI Computers*, pages 83–101. Springer, 1985.

[20] Daniel Molka, Daniel Hackenberg, Robert Schone, and Matthias S Muller. Memory performance and cache coherency effects on an intel nehalem multiprocessor system. In *Parallel Architectures and Compilation Techniques, 2009. PACT'09. 18th International Conference on*, pages 261–270. IEEE, 2009.

[21] Gordon E Moore et al. Cramming more components onto integrated circuits, 1965.

[22] G Nicoletta, Jose Alvarez, Eric Barkin, Chai-Chin Chao, Brad R Johnson, Franklin M Lassandro, Paresh Patel, Douglas Reid, Hector Sanchez, J Seigel, et al. A 450-mhz risc microprocessor with enhanced instruction set and copper interconnect. *Solid-State Circuits, IEEE Journal of*, 34(11):1478–1491, 1999.

[23] Mark S Papamarcos and Janak H Patel. A low-overhead coherence solution for multiprocessors with private cache memories. In *ACM SIGARCH Computer Architecture News*, volume 12, pages 348–354. ACM, 1984.

[24] Sivakumar Radhakrishnan, Sundaram Chinthamani, and Kai Cheng. The blackford northbridge chipset for the intel 5000. *Micro, IEEE*, 27(2):22–33, 2007.

[25] Jaswinder Pal Singh, Anoop Gupta, and Marc Levoy. Parallel visualization algorithms: Performance and architectural implications. *Computer*, 27(7):45–55, 1994.

[26] Jaswinder Pal Singh, Wolf-Dietrich Weber, and Anoop Gupta. Splash: Stanford parallel applications for shared-memory. *ACM SIGARCH Computer Architecture News*, 20(1):5–44, 1992.

[27] Paul Sweazey and Alan Jay Smith. A class of compatible cache consistency protocols and their support by the ieee futurebus. In *ACM SIGARCH Computer Architecture News*, volume 14, pages 414–423. IEEE Computer Society Press, 1986.

[28] Takuya Terasawa, Satoshi Ogura, Keisuke Inoue, and Hideharu Amano. A cache coherency protocol for multiprocessor chip. In *Wafer Scale Integration, 1995. Proceedings., Seventh Annual IEEE International Conference on*, pages 238–247. IEEE, 1995.

[29] Michael E Thomadakis. The architecture of the nehalem processor and nehalem-ep smp platforms. *Resource*, 2011.

[30] Rafael Ubal, Julio Sahuquillo, Salvador Petit, Pedro Lopez, Zhongliang Chen, and David R Kaeli. The multi2sim simulation framework: A cpu-gpu model for heterogeneous computing. 2011.

[31] Steven Cameron Woo, Moriyoshi Ohara, Evan Torrie, Jaswinder Pal Singh, and Anoop Gupta. The splash-2 programs: Characterization and methodological considerations. In *ACM SIGARCH Computer Architecture News*, volume 23, pages 24–36. ACM, 1995.

[32] Steven Cameron Woo, Jaswinder Pal Singh, and John L Hennessy. *The Performance Advantages of Integrating Message Passing in Cache-Coherent Multiprocessors*. 1993.

[33] Steven Cameron Woo, Jaswinder Pal Singh, and John L Hennessy. *The performance advantages of integrating block data transfer in cache-coherent multiprocessors*, volume 29. ACM, 1994.

Appendix A

For Section 4.3.1

Program	L_2 **to** L_1 **forwarding in MOSI**	**No. of** $L2$ **accesses in MOSI**	$L2$ **to** $L1$ **forwarding in MASI**	**No. of** $L2$ **accesses in MASI**	**Average no. of forwarding per** $100,000$ **access in MOSI**	**Average no. of forwarding per** $100,000$ **access in MASI**	**Percentage improvement**
Barnes	957554	2450908	405920	2793805	39069.357152	14529.288909	**62.811548**
Fft	698143	1063917	678503	1087006	65620.062467	62419.434667	**4.877514**
Fmm	1006952	2462598	584660	2350968	40889.824486	24868.905064	**39.180700**
Lu	443830	1917693	306209	2231899	23143.954741	13719.662046	**40.720321**
Ocean	1882093	13540709	1545358	13882668	13899.515897	11131.563471	**19.914020**
Radiosity	650885	6876205	567120	7290051	9465.759092	7779.369445	**17.815683**
Radix	1014449	1409651	994635	1407672	71964.550091	70658.150478	**1.815338**
Raytrace	3111902	14542230	2310248	15070817	21399.070156	15329.281750	**28.364730**
Water-nsquared	540020	1386179	239952	1733936	38957.450661	13838.573050	**64.477724**
Water-spatial	115937	333387	65333	351150	34775.501144	18605.439271	**46.498429**

Table A.1: Forwarding for 16 processors

Program	**L_1 to L_2 write-backs in MOSI**	**No. of L_2 accesses in MOSI**	**L_1 to L_2 write-backs in MASI**	**No. of L_2 accesses in MASI**	**Average no. of writebacks per $100,000$ access in MOSI**	**Average no. of writebacks per $100,000$ access in MASI**	**Percentage improvement**
Barnes	4557	2450908	14136	2793805	185.931092	505.976616	**-172.131256**
Fft	28210	1063917	29755	1087006	2651.522628	2737.335396	**-3.236358**
Fmm	5044	2462598	13538	2350968	204.824336	575.847906	**-181.142328**
Lu	14888	1917693	28417	2231899	776.349499	1273.220697	**-64.000968**
Ocean	3987	13540709	10102	13882668	29.444544	72.766993	**-147.132348**
Radiosity	102547	6876205	115135	7290051	1491.331338	1579.344232	**-5.901632**
Radix	578	1409651	1123	1407672	41.003057	79.777107	**-94.563804**
Raytrace	19959	14542230	223230	15070817	137.248551	1481.207024	**-979.215052**
Water-nsquared	7476	1386179	13179	1733936	539.324286	760.062655	**-40.928691**
Water-spatial	1994	333387	4167	351150	598.103705	1186.672362	**-98.405787**

Table A.2: Writebacks for 16 processors

Program	**L_2 to L_1 forwarding in MOSI**	**No. of L_2 accesses in MOSI**	**L_2 to L_1 forwarding in MASI**	**No. of L_2 accesses in MASI**	**Average no. of forwarding per $100,000$ access in MOSI**	**Average no. of forwarding per $100,000$ access in MASI**	**Percentage improvement**
Barnes	3350723	12972614	1143574	10132463	25829.204507	11286.239091	**56.304348**
Fft	1262197	1885536	1196317	2106560	66941.018363	56790.074814	**15.164011**
Fmm	2086868	5968369	1277124	6428034	34965.465440	19868.034301	**43.178121**
Lu	1965552	8676176	1161151	10724766	22654.588842	10826.818972	**52.209157**
Ocean	3102088	26546148	2293074	27505294	11685.642678	8336.845990	**28.657360**
Radiosity	1308830	22965062	920206	22825943	5699.222584	4031.404091	**29.263965**
Radix	1946140	4363545	2415508	19806242	44599.975479	12195.690631	**72.655387**
Raytrace	3531539	14613231	2327887	18041784	24166.722609	12902.753963	**46.609418**
Water-nsquared	1090390	2975715	548726	5067542	36642.958079	10828.247699	**70.449308**
Water-spatial	563204	1203984	515463	1486843	46778.362503	34668.287102	**25.888199**

Table A.3: Forwarding for 32 processors

Program	L_1 **to** L_2 **write-backs in MOSI**	**No. of** L_2 **accesses in MOSI**	L_1 **to** L_2 **write-backs in MASI**	**No. of** L_2 **accesses in MASI**	**Average no. of writebacks per** $100,000$ **access in MOSI**	**Average no. of writebacks per** $100,000$ **access in MASI**	**Percentage improvement**
Barnes	3716	12972614	12565	10132463	28.644959	124.007361	**-332.911634**
Fft	27173	1885536	27521	2106560	1441.128676	1306.442731	**9.345865**
Fmm	4671	5968369	11534	6428034	78.262587	179.432778	**-129.270184**
Lu	18368	8676176	32421	10724766	211.706171	302.300302	**-42.792390**
Ocean	4280	26546148	9035	27505294	16.122866	32.848222	**-103.736860**
Radiosity	102165	22965062	112663	22825943	444.871431	493.574351	**-10.947639**
Radix	1344	4363545	3726	19806242	30.800645	18.812251	**38.922541**
Raytrace	20035	14613231	238493	18041784	137.101781	1321.892558	**-864.168770**
Water-nsquared	9316	2975715	11084	5067542	313.067616	218.725370	**30.134783**
Water-spatial	2020	1203984	4087	1486843	167.776316	274.877710	**-63.835824**

Table A.4: Writebacks for 32 processors

Appendix B

For Section 4.3.2

Program	L_2 **to** L_1 **forwarding in MOSI**	**No. of** $L2$ **accesses in MOSI**	$L2$ **to** $L1$ **forwarding in MASI**	**No. of** $L2$ **accesses in MASI**	**Average no. of forwarding per** $100,000$ **access in MOSI**	**Average no. of forwarding per** $100,000$ **access in MASI**	**Percentage improvement**
Barnes	943374	2447642	424987	2597987	38542.156083	16358.318960	**57.557333**
Fft	683687	1042917	658687	1061379	65555.264705	62059.547061	**5.332474**
Fmm	960633	2262485	600643	2506220	42459.198624	23966.092362	**43.555005**
Lu	444292	1889195	306099	2245498	23517.529953	13631.675468	**42.036109**
Ocean	1878536	13535890	1545860	13777118	13878.186067	11220.488930	**19.150176**
Radiosity	649317	7295851	527737	7556214	8899.804368	6984.145764	**21.524727**
Radix	1007121	1407083	992668	1399038	71575.095428	70953.612411	**0.868295**
Raytrace	3126312	13729932	2341160	15256963	22770.047222	15344.862539	**32.609439**
Water-nsquared	589212	1456172	242309	1857090	40463.077164	13047.779052	**67.753864**
Water-spatial	116367	361106	65220	357846	32225.163802	18225.717208	**43.442592**

Table B.1: Forwarding for 16 processors with dirty bit only

Program	L_1 **to** L_2 **write-backs in MOSI**	**No. of** L_2 **accesses in MOSI**	L_1 **to** L_2 **write-backs in MASI**	**No. of** L_2 **accesses in MASI**	**Average no. of writebacks per** $100,000$ **access in MOSI**	**Average no. of writebacks per** $100,000$ **access in MASI**	**Percentage improvement**
Barnes	4225	2447642	6451	2597987	172.615113	248.307632	**-43.850459**
Fft	28143	1042917	29120	1061379	2698.488950	2743.600542	**-1.672975**
Fmm	4690	2262485	9452	2506220	207.294192	377.141672	**-81.935474**
Lu	14879	1889195	15696	2245498	787.58413	698.998619	**11.247752**
Ocean	3850	13535890	6242	13777118	28.442902	45.307008	**-59.291088**
Radiosity	102403	7295851	103937	7556214	1403.578554	1375.516892	**1.999294**
Radix	751	1407083	1074	1399038	53.372829	76.767036	**-43.831679**
Raytrace	19955	13729932	23654	15256963	145.339394	155.037408	**-6.672667**
Water-nsquared	7263	1456172	8204	1857090	498.773496	441.766420	**11.429452**
Water-spatial	1956	361106	2237	357846	541.669205	625.129246	**-15.407935**

Table B.2: Writebacks for 16 processors with dirty bit only

Program	L_2 **to** L_1 **forwarding in MOSI**	**No. of** L_2 **accesses in MOSI**	L_2 **to** L_1 **forwarding in MASI**	**No. of** L_2 **accesses in MASI**	**Average no. of forwarding per** $100,000$ **access in MOSI**	**Average no. of forwarding per** $100,000$ **access in MASI**	**Percentage improvement**
Barnes	2308763	8131087	1382328	9851869	28394.272500	14031.124450	**50.584666**
Fft	1253639	1881477	1219086	2170525	66630.577998	56165.489916	**15.706134**
Fmm	2126208	6114759	1321492	6215420	34771.738346	21261.507670	**38.854056**
Lu	1696307	6479331	1189629	14095113	26180.280032	8440.010378	**67.761955**
Ocean	3027609	26033721	2310725	28015448	11629.566899	8248.038725	**29.076991**
Radiosity	1441830	21047119	861785	22939892	6850.486283	3756.709055	**45.161425**
Radix	1921445	3659988	2437482	20236368	52498.669394	12045.056702	**77.056453**
Water-nsquared	1712021	6019612	667389	8696321	28440.720100	7674.383225	**73.016213**
Water-spatial	610734	1402372	514076	1505172	43550.070880	34153.970443	**21.575396**

Table B.3: Forwarding for 32 processors with dirty bit only

Program	L_1 **to** L_2 **write-backs in MOSI**	**No. of** L_2 **accesses in MOSI**	L_1 **to** L_2 **write-backs in MASI**	**No. of** L_2 **accesses in MASI**	**Average no. of writebacks per** $100,000$ **access in MOSI**	**Average no. of writebacks per** $100,000$ **access in MASI**	**Percentage improvement**
Barnes	3323	8131087	5032	9851869	40.867845	51.076603	**-24.979927**
Fft	27164	1881477	27274	2170525	1443.759344	1256.562352	**12.965942**
Fmm	4488	6114759	7818	6215420	73.396188	125.783937	**-71.376662**
Lu	18178	6479331	18873	14095113	280.553656	133.897472	**52.273845**
Ocean	4223	26033721	6240	28015448	16.221269	22.273426	**-37.310009**
Radiosity	101042	21047119	103277	22939892	480.075206	450.207002	**6.221568**
Radix	1319	3659988	3962	20236368	36.038370	19.578612	**45.672870**
Water-nsquared	10004	6019612	10942	8696321	166.190113	125.823322	**24.289526**
Water-spatial	1933	1402372	2477	1505172	137.837891	164.565910	**-19.390908**

Table B.4: Writebacks for 32 processors with dirty bit only

Appendix C

For Section 4.3.3

Program	L_2 **to** L_1 **forwarding in MOSI**	**No. of** L_2 **accesses in MOSI**	L_2 **to** L_1 **forwarding in MASI**	**No. of** L_2 **accesses in MASI**	**Average no. of forwarding per** $100,000$ **access in MOSI**	**Average no. of forwarding per** $100,000$ **access in MASI**	**Percentage improvement**
Barnes	943374	2447642	449238	2780102	38542.156083	16159.047402	58.074355
Fft	683687	1042917	666706	1073839	65555.264705	62086.215904	5.291793
Fmm	960633	2262485	617175	2400737	42459.198624	25707.730584	39.453095
Ocean	1878536	13535890	1538208	13746371	13878.186067	11189.920598	19.370438
Radiosity	649317	7295851	540134	7095451	8899.811687	7612.398423	14.465624
Radix	1007121	1407083	994682	1393000	71575.095428	71405.743001	0.236608
Water-nsquared	589212	1456172	241746	1833518	40463.077164	13184.817384	67.415188
Water-spatial	116367	361106	64814	347437	32225.163802	18654.892829	42.110791

Table C.1: Forwardings for 16 processors with new eviction policy

Program	L_1 **to** L_2 **write-backs in MOSI**	**No. of** L_2 **accesses in MOSI**	L_1 **to** L_2 **write-backs in MASI**	**No. of** L_2 **accesses in MASI**	**Average no. of writebacks per** $100,000$ **access in MOSI**	**Average no. of writebacks per** $100,000$ **access in MASI**	**Percentage improvement**
Barnes	4225	2447642	5036	2780102	172.615113	181.144433	-4.941236
Fft	28143	1042917	29548	1073839	2698.488950	2751.622916	-1.969027
Fmm	4690	2262485	8600	2400737	207.294192	358.223329	-72.809149
Ocean	3850	13535890	5641	13746371	28.442903	41.036285	-44.276008
Radiosity	102403	7295851	103774	7095451	1403.578554	1462.542691	-4.200986
Radix	751	1407083	1063	1393000	53.372829	76.310122	-42.975599
Water-nsquared	7263	1456172	7872	1833518	498.773497	429.338572	13.921134
Water-spatial	1956	361106	2068	347437	541.669205	595.215823	-9.885483

Table C.2: Writebacks for 16 processors with new eviction policy

Program	L_2 **to** L_1 **forwarding in MOSI**	**No. of** L_2 **accesses in MOSI**	L_2 **to** L_1 **forwarding in MASI**	**No. of** L_2 **accesses in MASI**	**Average no. of forwarding per** $100,000$ **access in MOSI**	**Average no. of forwarding per** $100,000$ **access in MASI**	**Percentage improvement**
Barnes	2308763	8131087	1099434	8989870	28394.272500	12229.698538	56.928995
Fft	1253639	1881477	1219085	2170526	66630.577998	56165.417968	15.706242
Lu	1696307	6479331	1135050	10825322	26180.280032	10485.138456	59.950243
Ocean	3027609	26033721	2291323	27967379	11629.566899	8192.841381	29.551620
Radiosity	1441830	21047119	944260	24037455	6850.486283	3928.286085	42.656829
Water-spatial	610734	1402372	492764	1475033	43550.070880	33406.981403	23.290638

Table C.3: Forwardings for 32 processors with new eviction policy

Program	L_1 **to** L_2 **write-backs in MOSI**	**No. of** L_2 **accesses in MOSI**	L_1 **to** L_2 **write-backs in MASI**	**No. of** L_2 **accesses in MASI**	**Average no. of writebacks per** $100,000$ **access in MOSI**	**Average no. of writebacks per** $100,000$ **access in MASI**	**Percentage improvement**
Barnes	3323	8131087	3947	8989870	40.867845	43.904973	-7.385240
Fft	27164	1881477	27270	2170526	1443.759344	1256.377486	12.978746
Lu	18178	6479331	18867	10825322	280.553656	174.285809	37.877906
Ocean	4223	26033721	5987	27967379	16.221269	21.407083	-31.969225
Radiosity	101042	21047119	102823	24037455	480.075206	427.761591	10.896962
Water-spatial	1933	1402372	2329	1475033	137.837892	157.894773	-14.551065

Table C.4: Writebacks for 32 processors with new eviction policy

Program	L_2 **to** L_1 **forwarding in MOSI**	**No. of** L_2 **accesses in MOSI**	L_2 **to** L_1 **forwarding in MASI**	**No. of** L_2 **accesses in MASI**	**Average no. of forwarding per** $100,000$ **access in MOSI**	**Average no. of forwarding per** $100,000$ **access in MASI**	**Percentage improvement**
Fft	2917858	6782893	2635223	7264931	43017.898115	36273.200668	15.678817
Lu	8769556	43293093	3100687	52838190	20256.247342	5868.268765	71.029832
Ocean	7768734	48302901	5584798	52203344	16083.369402	10698.161405	33.483083
Water-spatial	4932842	10805710	5421539	15817593	45650.327466	34275.372998	24.917575

Table C.5: Forwardings for 64 processors with new eviction policy

Program	L_1 **to** L_2 **writebacks in MOSI**	**No. of** L_2 **accesses in MOSI**	L_1 **to** L_2 **writebacks in MASI**	**No. of** L_2 **accesses in MASI**	**Average no. of writebacks per** $100,000$ **access in MOSI**	**Average no. of writebacks per** $100,000$ **access in MASI**	**Percentage improvement**
Fft	27499	6782893	27368	7264931	405.416981	376.713832	7.079908
Lu	15344	43293093	14259	52838190	35.442143	26.986163	23.858546
Ocean	6885	48302901	8483	52203344	14.253802	16.249917	-14.004088
Water-spatial	2634	10805710	2404	15817593	24.376001	15.198267	37.650696

Table C.6: Writebacks for 64 processors with new eviction policy

Appendix D

For Section 4.3.4

Program	L_2 **to** L_1 **forwarding in MOESI**	**No. of** L_2 **accesses in MOESI**	L_2 **to** L_1 **forwarding in MASI**	**No. of** L_2 **accesses in MASI**	**Average no. of forwarding per** $100,000$ **access in MOESI**	**Average no. of forwarding per** $100,000$ **access in MASI**	**Percentage improvement**
Barnes	943382	2447658	449238	2780102	38542.230982	16159.047402	**58.074437**
Fft	698150	1061431	666706	1073839	65774.412091	62086.215904	**5.607342**
Fmm	978770	2333549	617175	2400737	41943.408944	25707.730584	**38.708533**
Ocean	1877688	13516507	1538208	13746371	13891.813913	11189.920598	**19.449536**
Radiosity	662219	7059851	540134	7095451	9380.0704859	7612.398423	**18.844976**
Radix	1014449	1409651	994682	1393000	71964.550091	71405.743001	**0.776503**
Water-nsquared	540020	1386179	241746	1833518	38957.450661	13184.817384	**66.155852**
Water-spatial	116368	361108	64814	347437	32225.262248	18654.892829	**42.110966**

Table D.1: Forwarding for 16 processors with new eviction policy and dirty MOESI

Program	L_1 **to** L_2 **write-backs in MOESI**	**No. of** L_2 **accesses in MOESI**	L_1 **to** L_2 **write-backs in MASI**	**No. of** L_2 **accesses in MASI**	**Average no. of writebacks per** $100,000$ **access in MOESI**	**Average no. of writebacks per** $100,000$ **access in MASI**	**Percentage improvement**
Barnes	4227	2447658	5036	2780102	172.695695	181.144433	**-4.892269**
Fft	28206	1061431	29548	1073839	2657.355961	2751.622916	**-3.547396**
Fmm	4873	2333549	8600	2400737	208.823555	358.223329	**-71.543544**
Ocean	3738	13516507	5641	13746371	27.6550739	41.036285	**-48.386097**
Radiosity	101985	7059851	103774	7095451	1444.577232	1462.542691	**-1.243648**
Radix	578	1409651	1063	1393000	41.0030568	76.310122	**-86.108373**
Water-nsquared	7476	1386179	7872	1833518	539.324286	429.338572	**20.393243**
Water-spatial	1955	361108	2068	347437	541.389280	595.215823	**-9.942299**

Table D.2: Writebacks for 16 processors with new eviction policy and dirty MOESI

Program	L_2 **to** L_1 **forwarding in MOESI**	**No. of** L_2 **accesses in MOESI**	L_2 **to** L_1 **forwarding in MASI**	**No. of** L_2 **accesses in MASI**	**Average no. of forwarding per** $100,000$ **access in MOESI**	**Average no. of forwarding per** $100,000$ **access in MASI**	**Percentage improvement**
Barnes	3248927	12362870	1099434	8989870	26279.714985	12229.698538	**53.463351**
Fft	1262035	1879353	1219085	2170526	67152.631783	56165.417968	**16.361553**
Lu	1967976	8679789	1135050	10825322	22673.085717	10485.138456	**53.755132**
Ocean	3088768	26683669	2291323	27967379	11575.499606	8192.841381	**29.222567**
Radiosity	1248566	22556751	944260	24037455	5535.2209190	3928.286085	**29.031087**
Water-spatial	564222	1180409	492764	1475033	47798.856159	33406.981403	**30.109245**

Table D.3: Forwarding for 32 processors with new eviction policy and dirty MOESI

Program	L_1 **to** L_2 **write-backs in MOESI**	**No. of** L_2 **accesses in MOESI**	L_1 **to** L_2 **write-backs in MASI**	**No. of** L_2 **accesses in MASI**	**Average no. of writebacks per** $100,000$ **access in MOESI**	**Average no. of writebacks per** $100,000$ **access in MASI**	**Percentage improvement**
Barnes	3444	12362870	3947	8989870	27.857609	43.904973	**-57.604958**
Fft	27171	1879353	27270	2170526	1445.763515	1256.377486	**13.099378**
Lu	18414	8679789	18867	10825322	212.148014	174.285809	**17.847070**
Ocean	4120	26683669	5987	27967379	15.440156	21.407083	**-38.645510**
Radiosity	100890	22556751	102823	24037455	447.271861	427.761591	**4.362061**
Water-spatial	1997	1180409	2329	1475033	169.178649	157.894773	**6.669799**

Table D.4: Writebacks for 32 processors with new eviction policy and dirty MOESI

Appendix E

Control Flow for Section 4.3.1

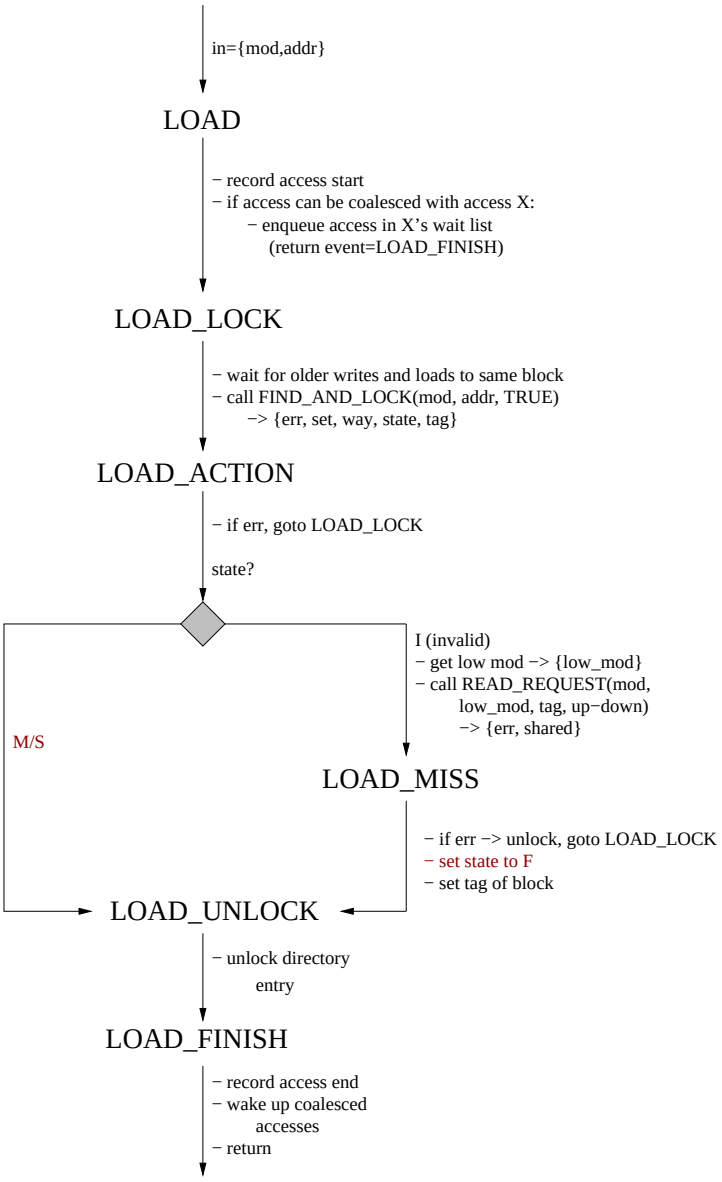

Figure E.1: LOAD Function for **MASI**

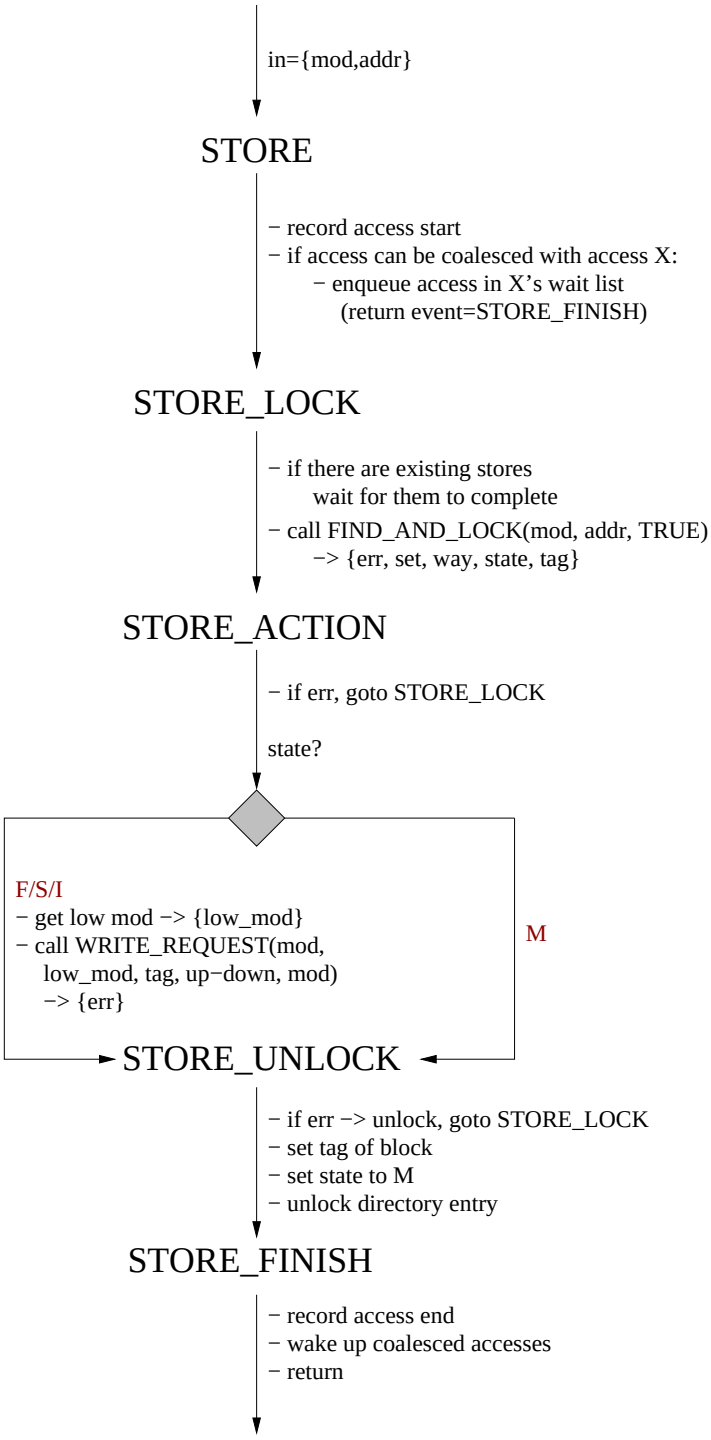

Figure E.2: STORE Function for **MASI**

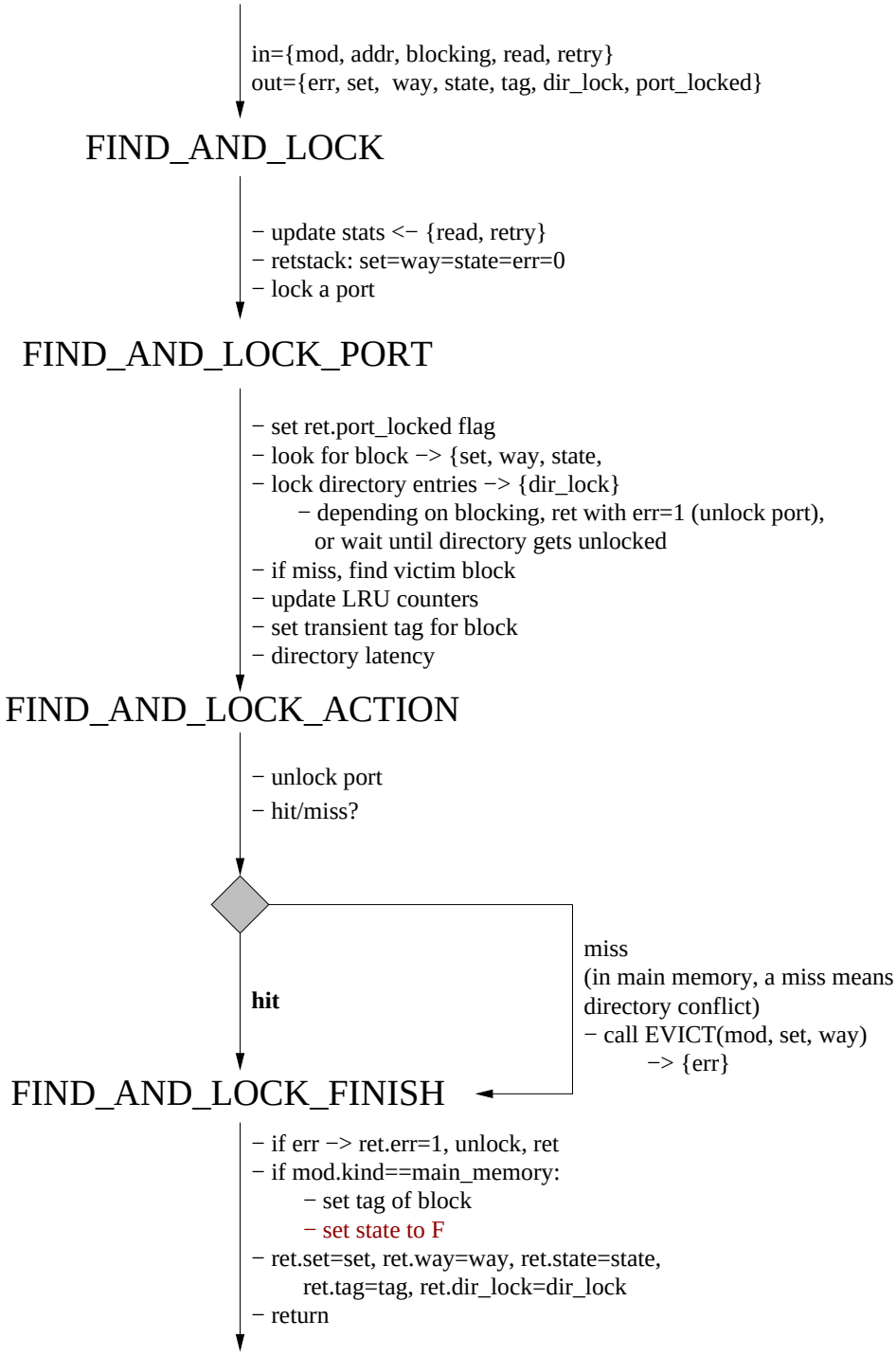

Figure E.3: FIND_ AND_ LOCK Function for **MASI**

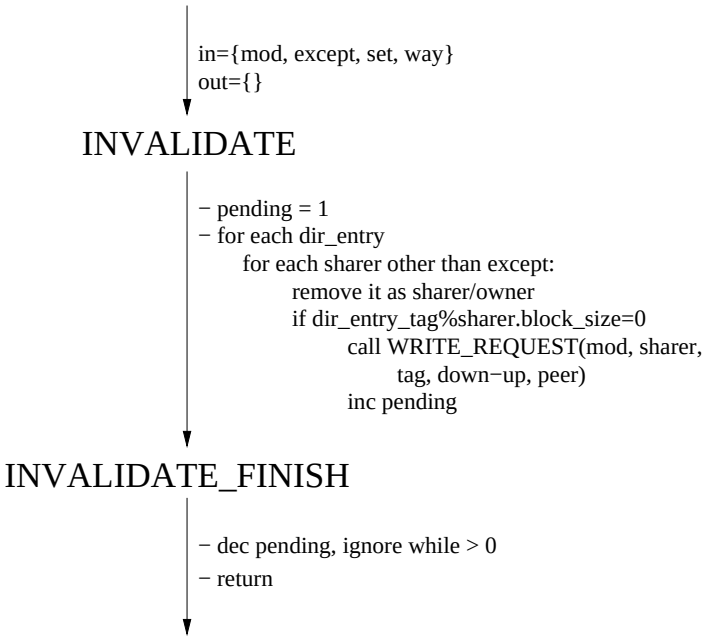

Figure E.4: INVALIDATE Function for **MASI**

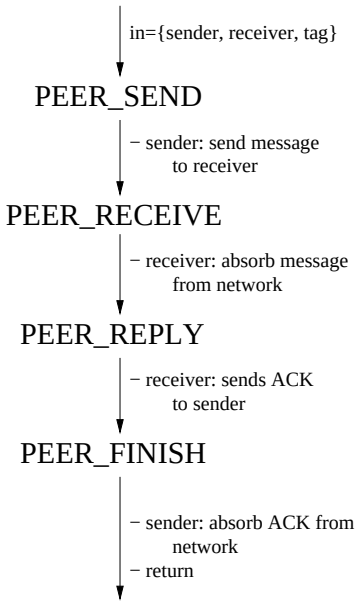

Figure E.5: PEER Function for **MASI**

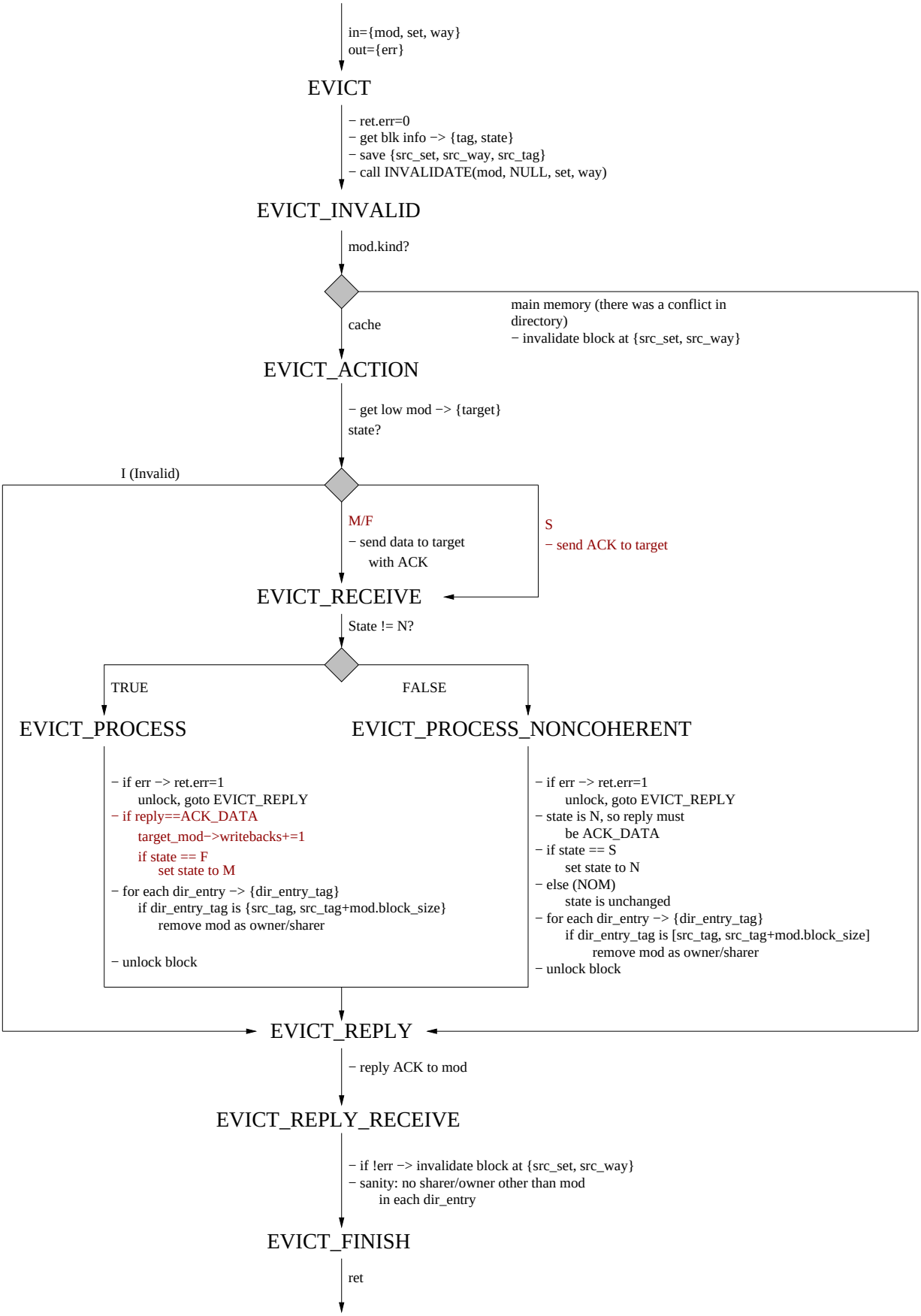

Figure E.6: EVICT Function for **MASI**

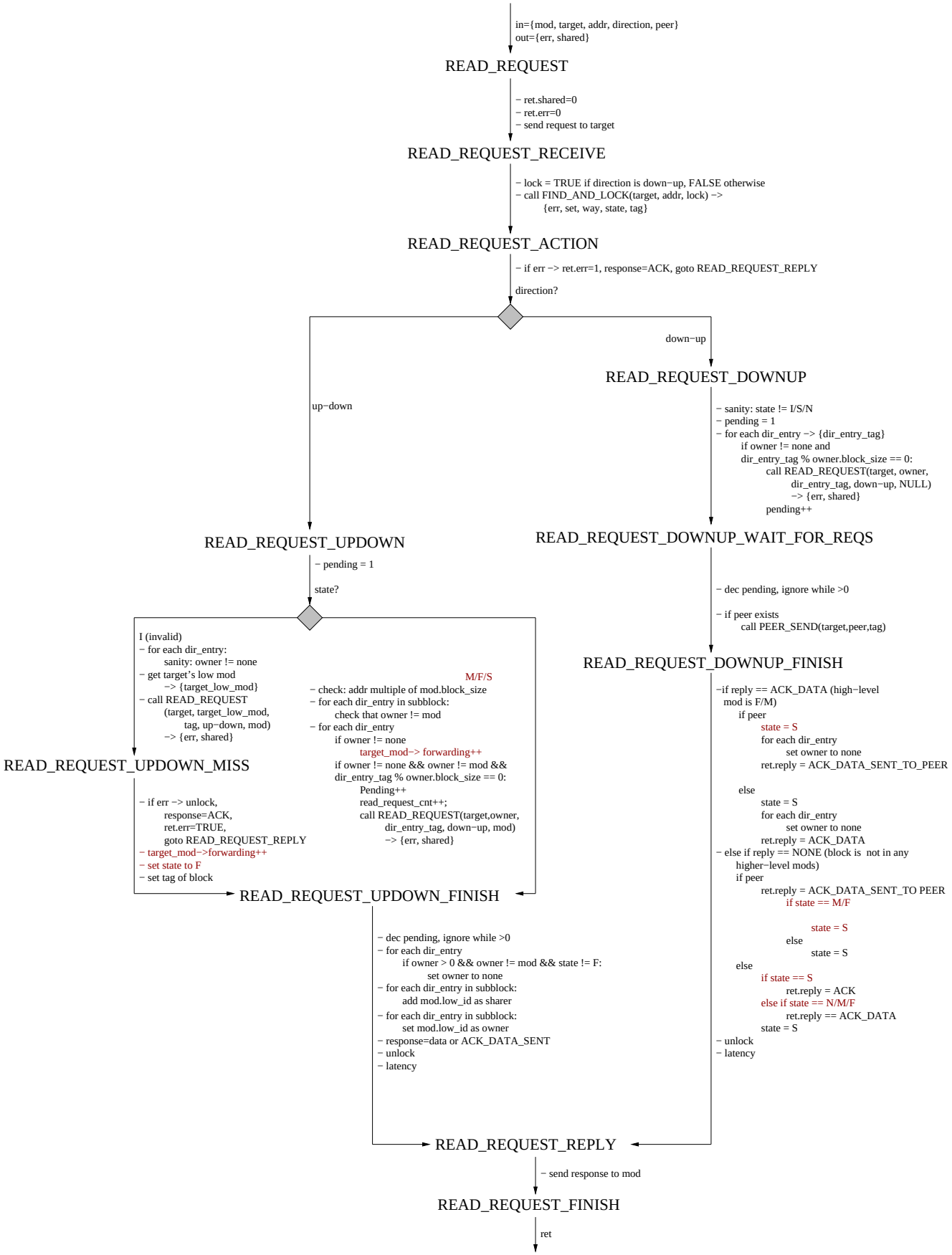

Figure E.7: READ_ REQUEST Function for **MASI**

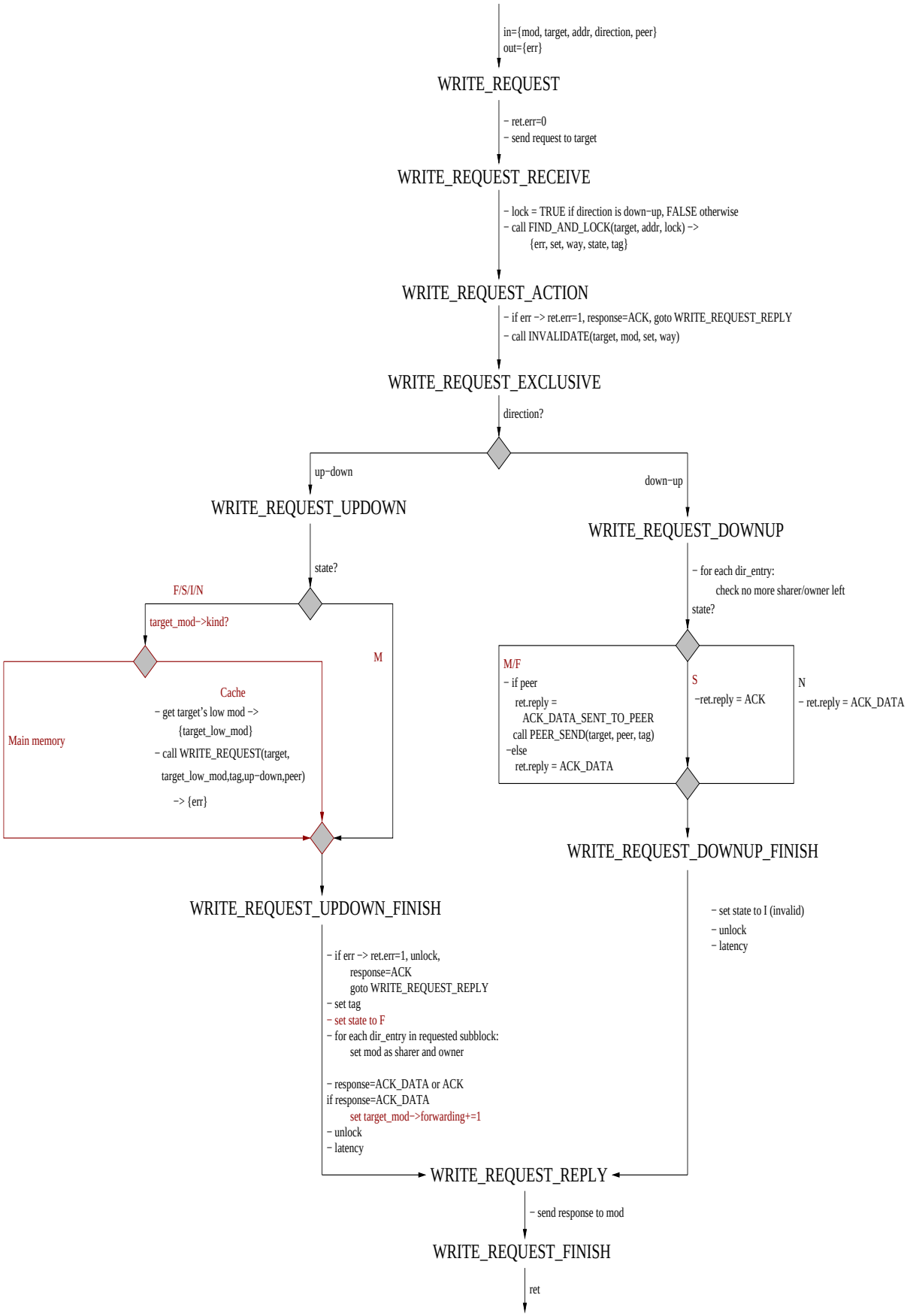

Figure E.8: WRITE_ REQUEST Function for **MASI**

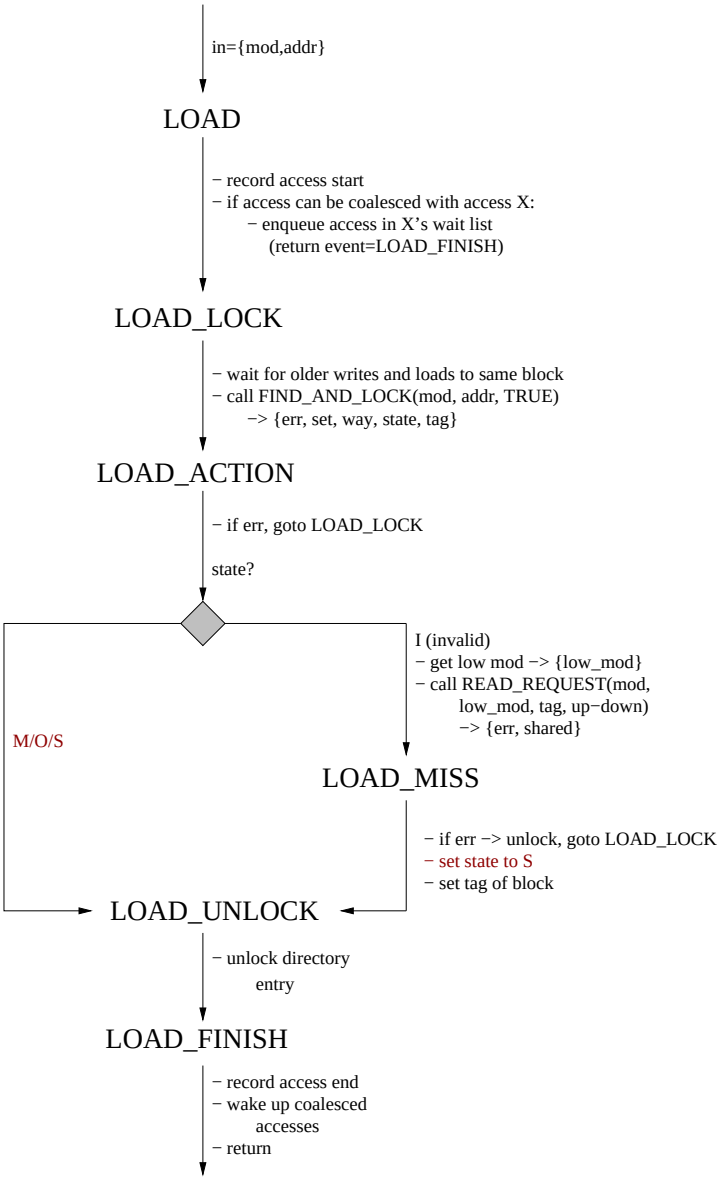

Figure E.9: LOAD Function for **MOSI**

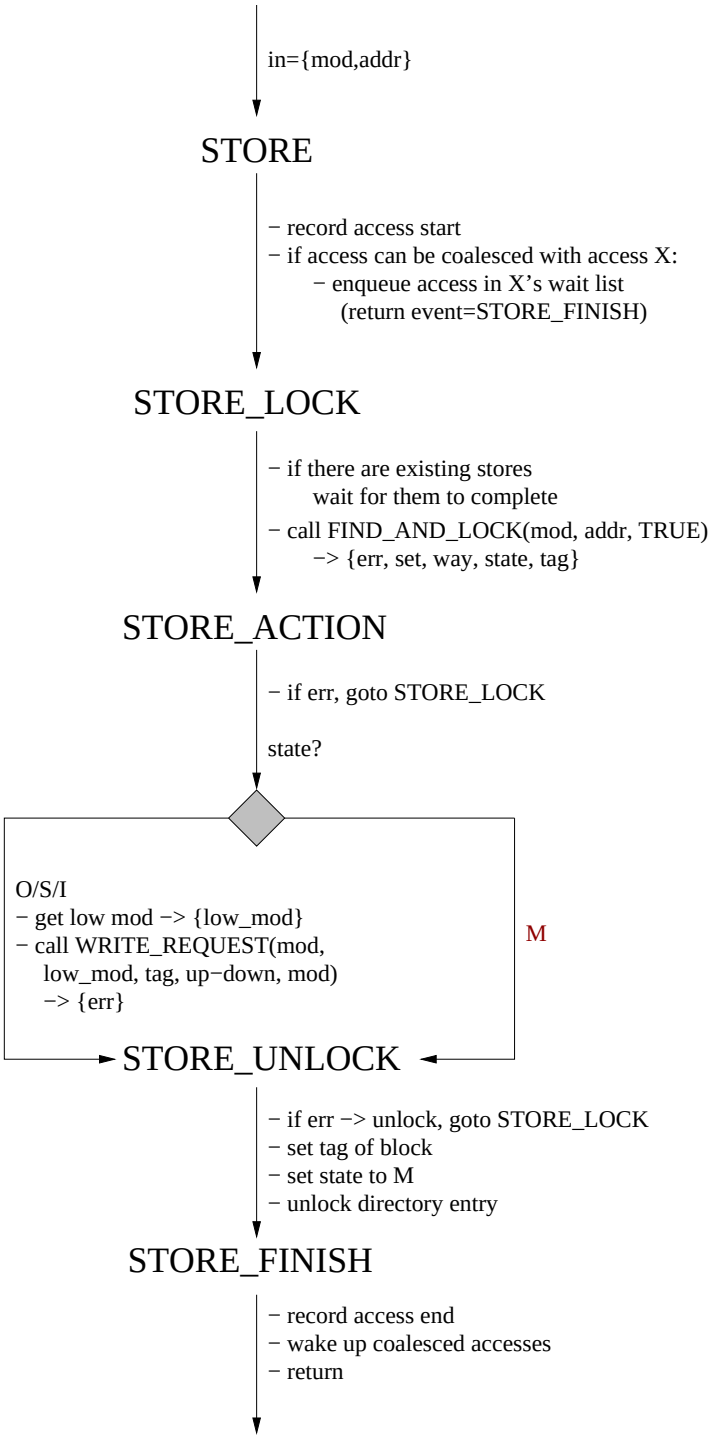

Figure E.10: STORE Function for **MOSI**

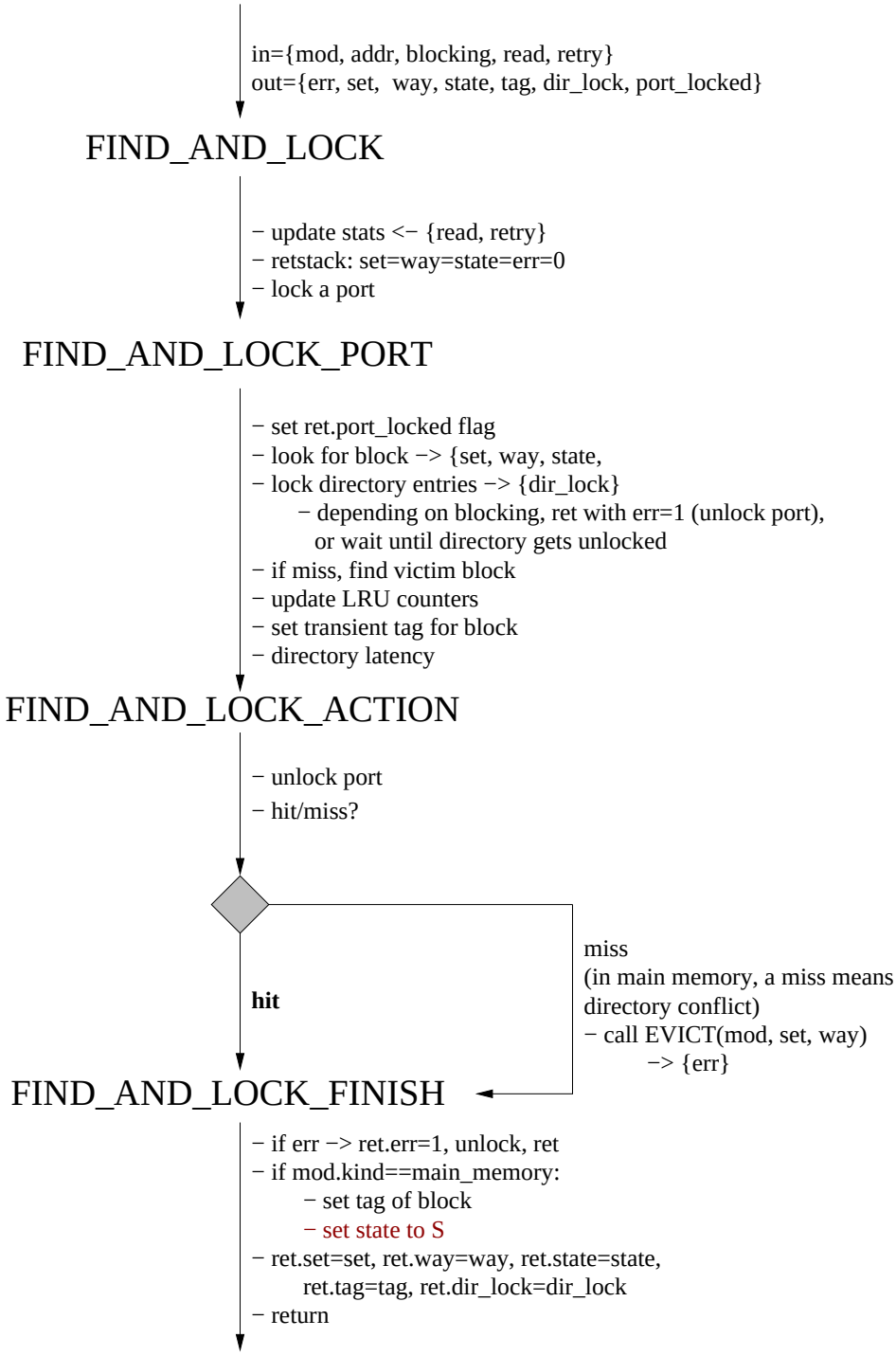

Figure E.11: FIND_ AND_ LOCK Function for **MOSI**

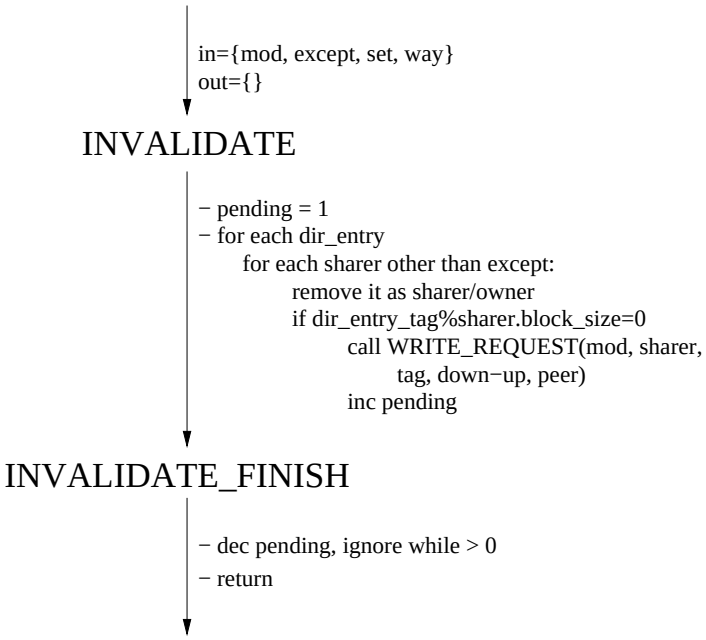

Figure E.12: INVALIDATE Function for **MOSI**

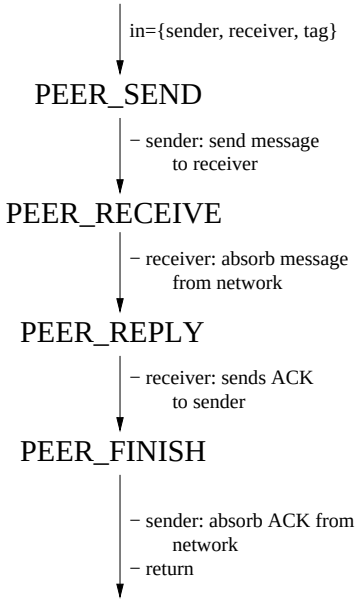

Figure E.13: PEER Function for **MOSI**

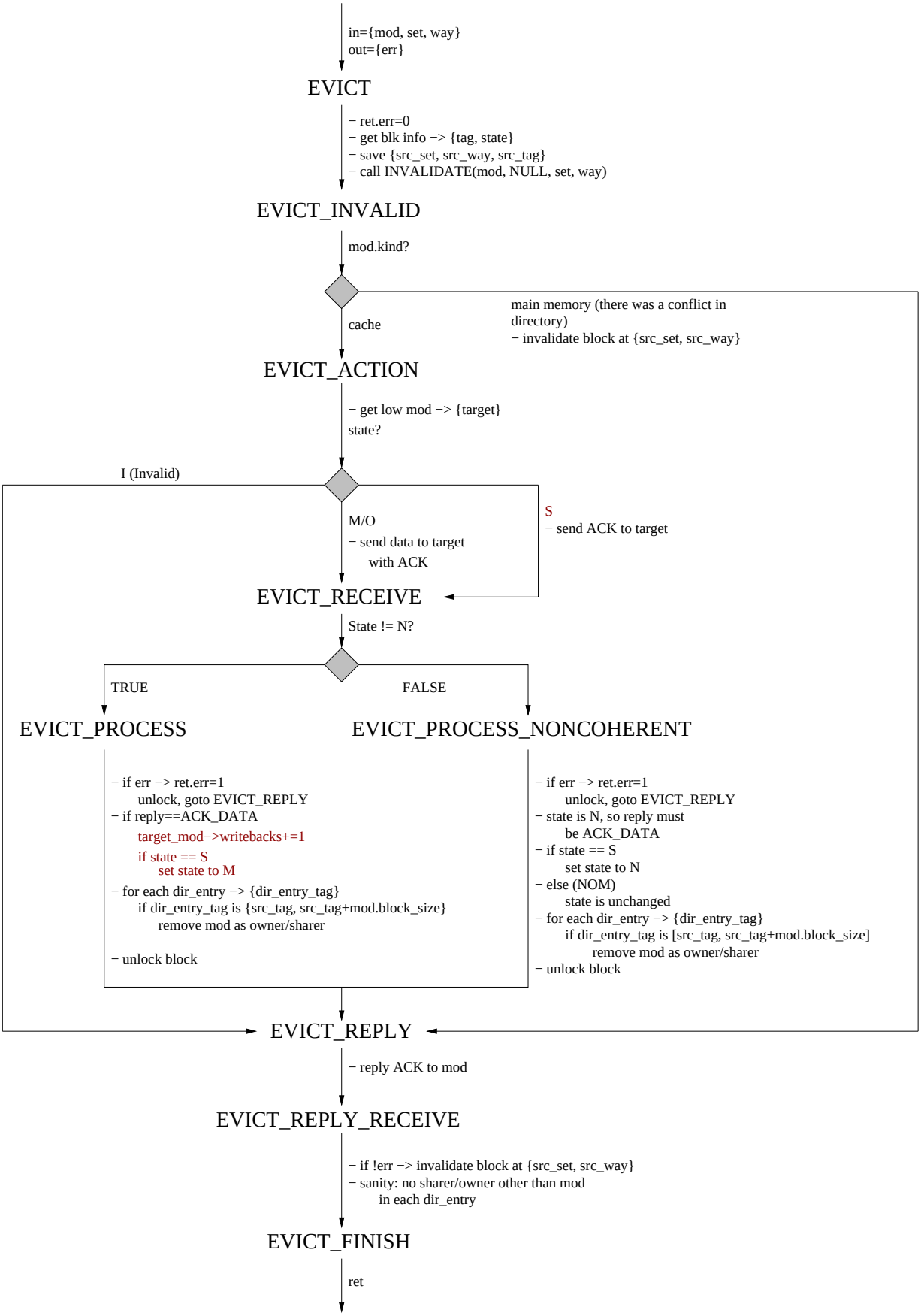

Figure E.14: EVICT Function for **MOSI**

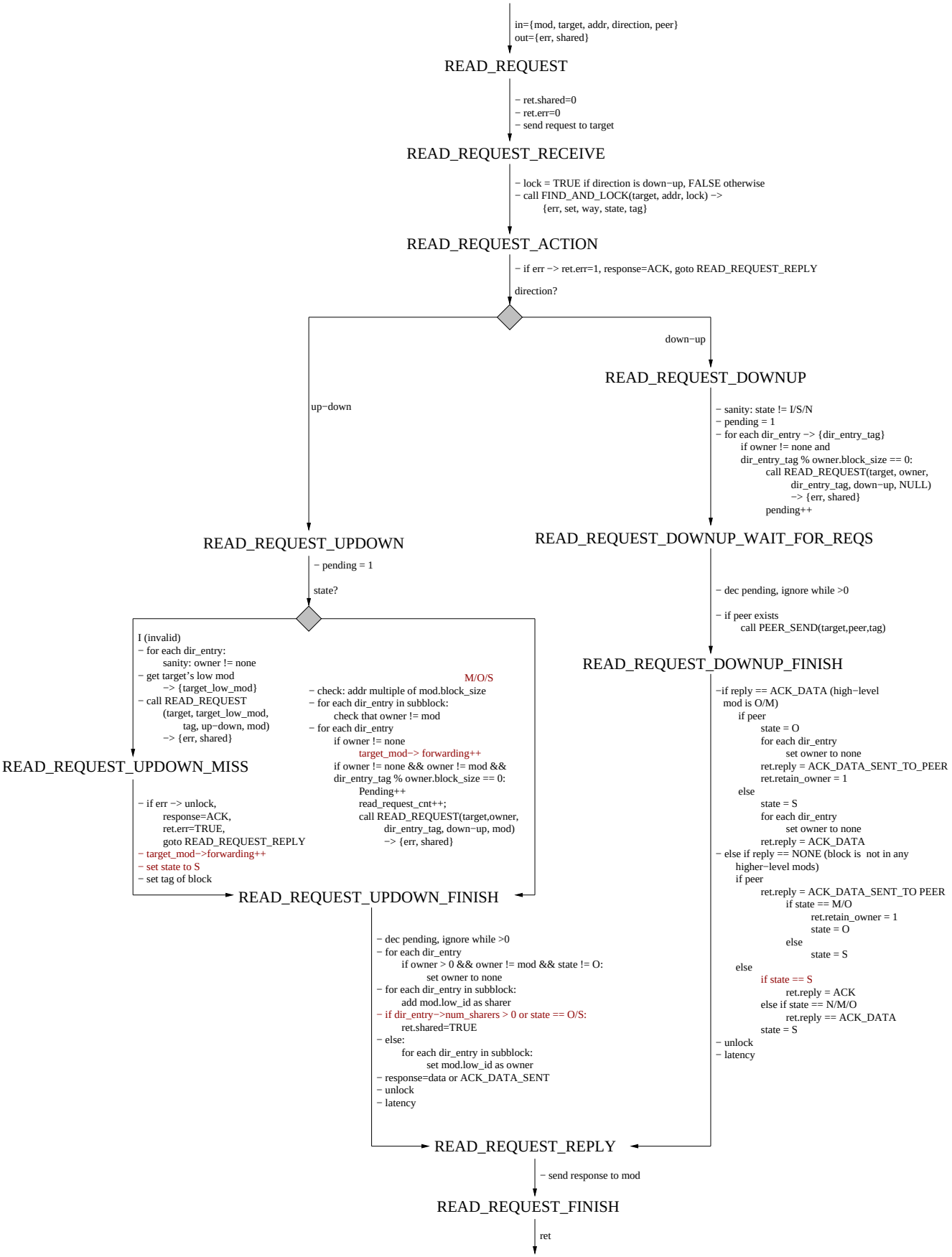

Figure E.15: READ_ REQUEST Function for **MOSI**

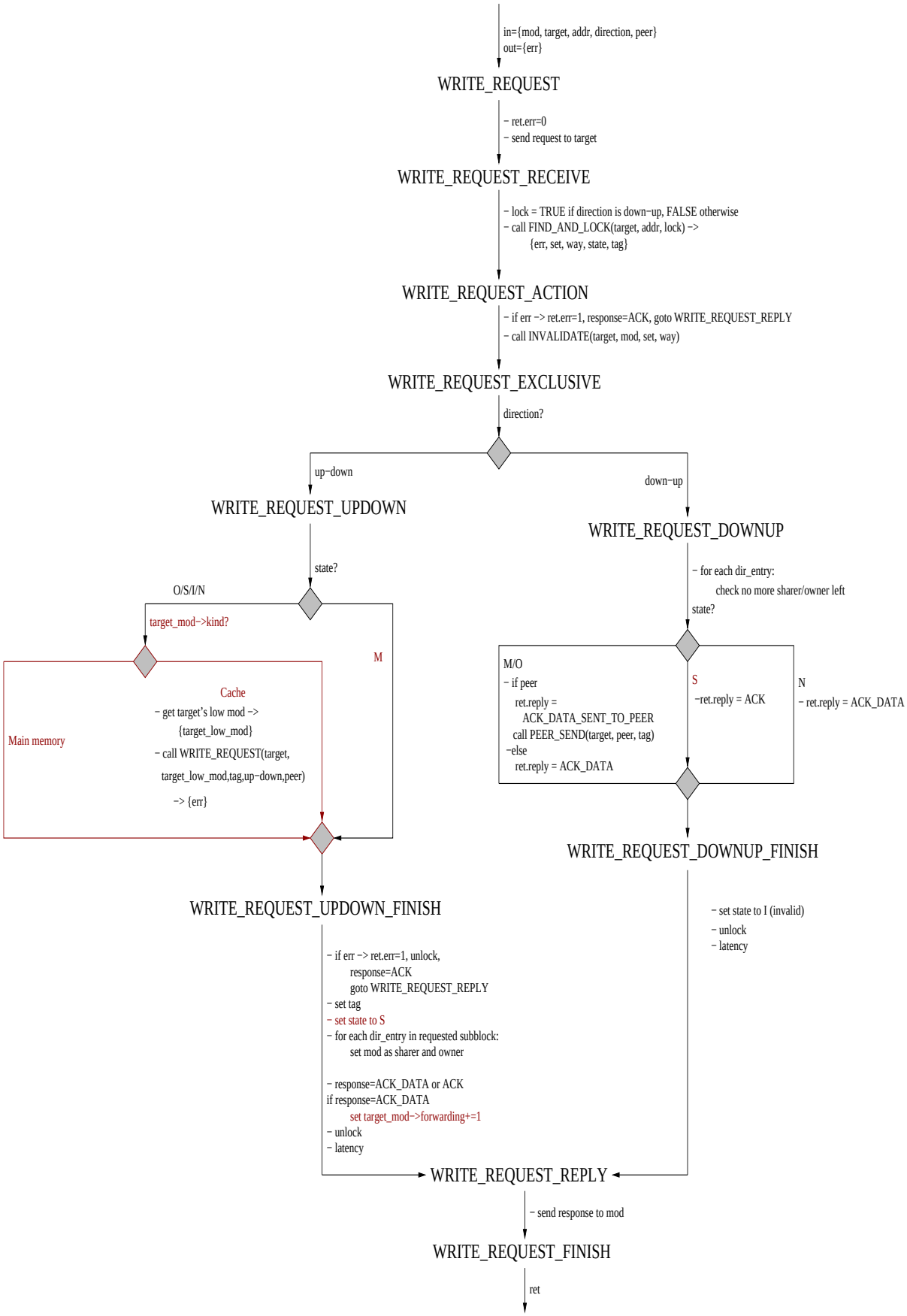

Figure E.16: WRITE_ REQUEST Function for **MOSI**

Appendix F

Control Flow for Section 4.3.2

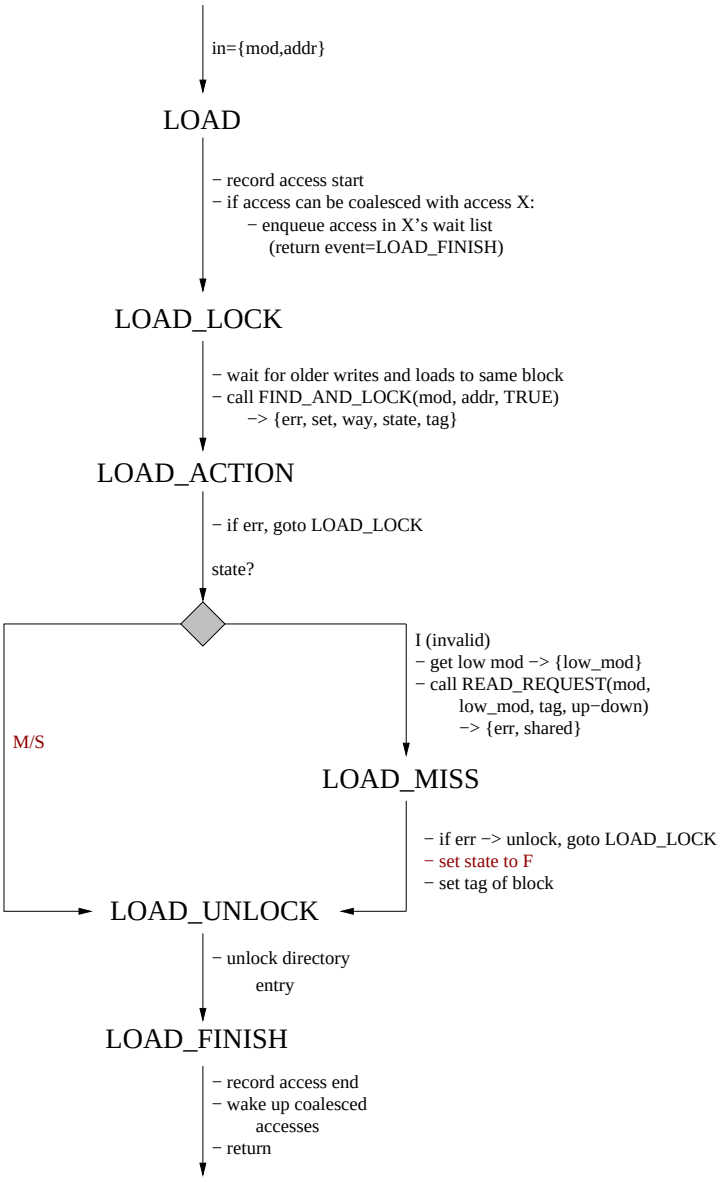

Figure F.1: LOAD Function for **MASI**

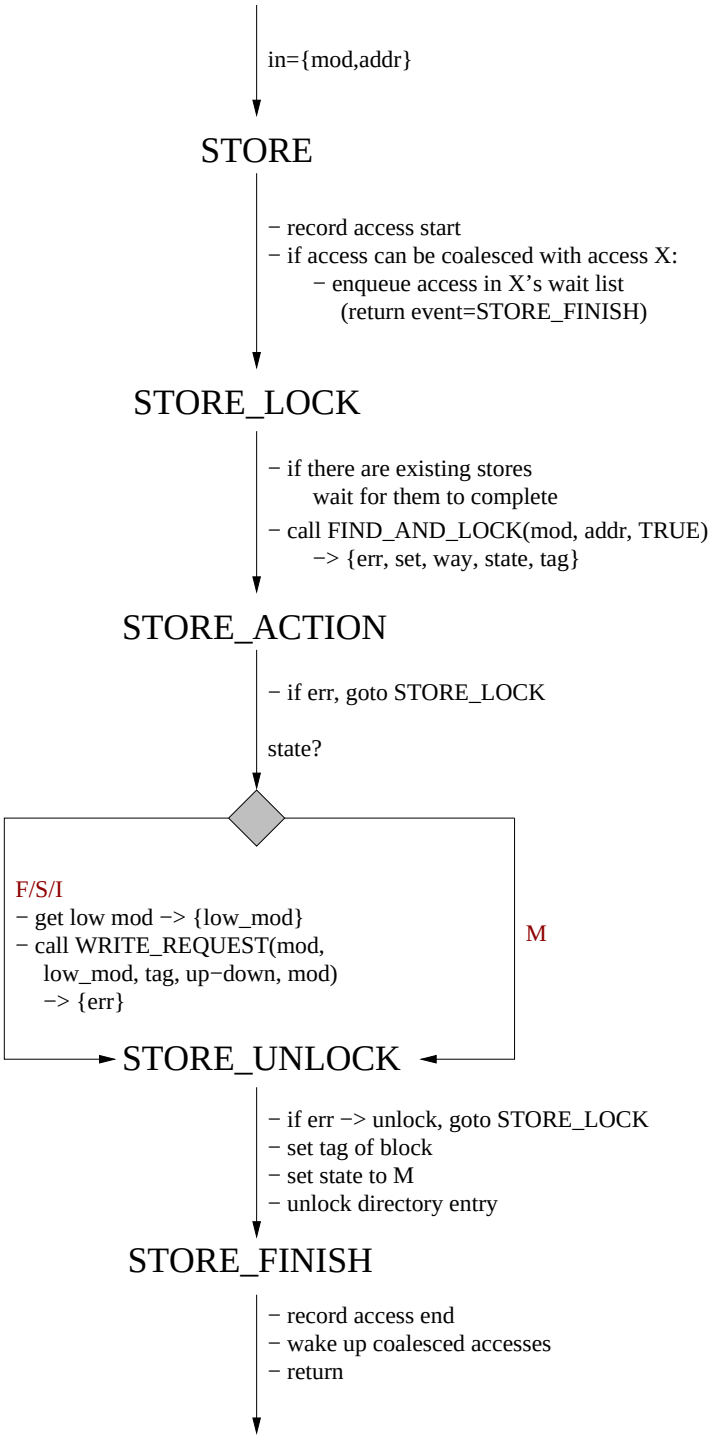

Figure F.2: STORE Function for **MASI**

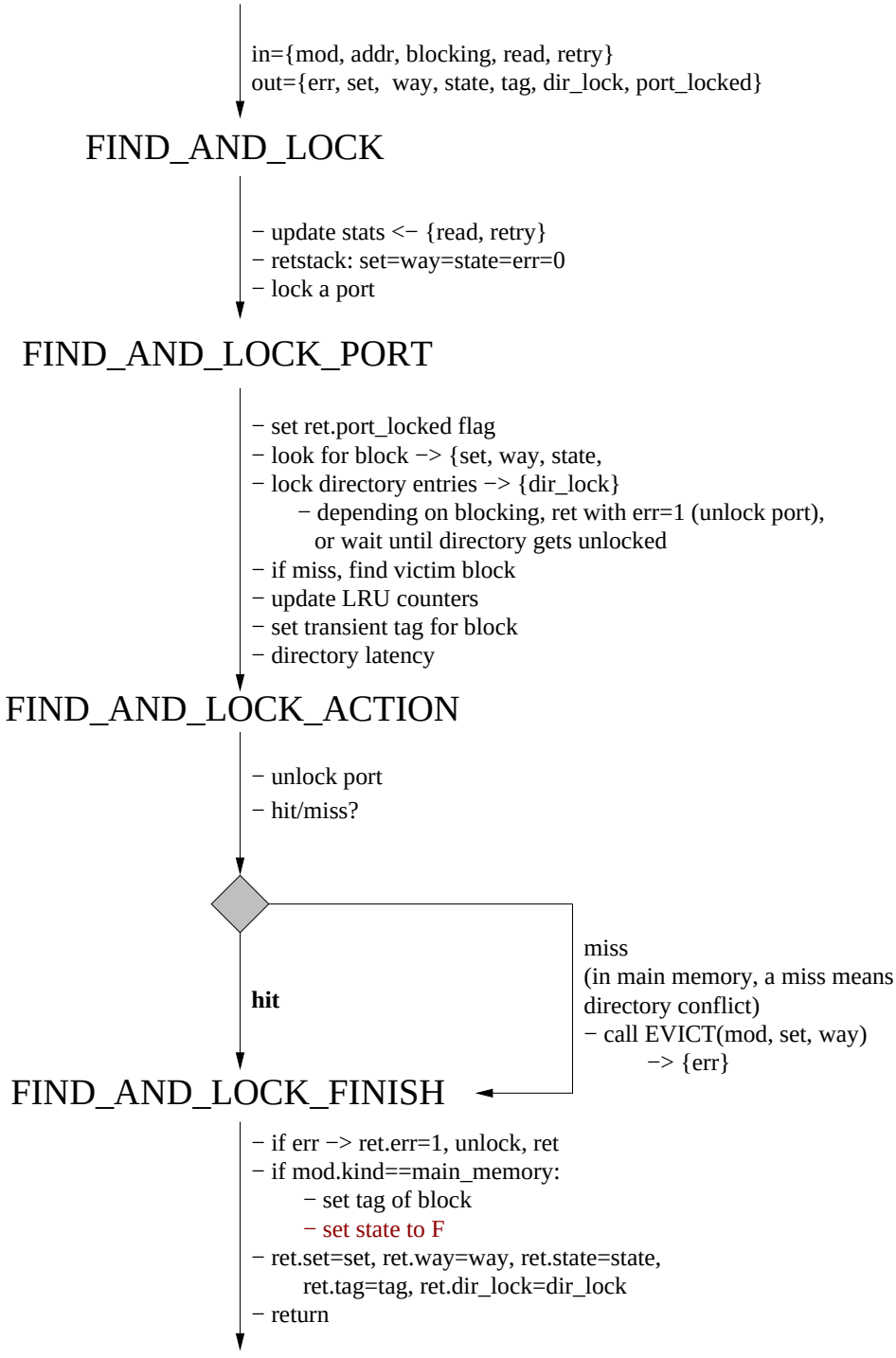

Figure F.3: FIND_ AND_ LOCK Function for **MASI**

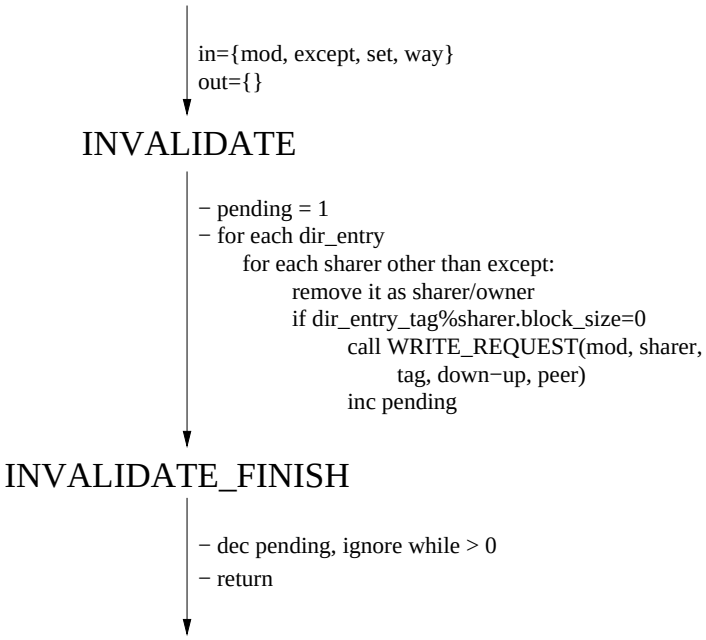

Figure F.4: INVALIDATE Function for **MASI**

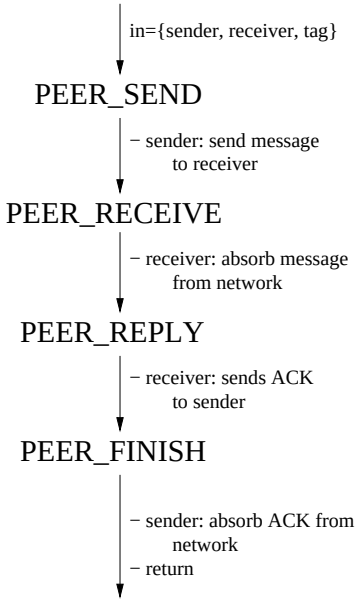

Figure F.5: PEER Function for **MASI**

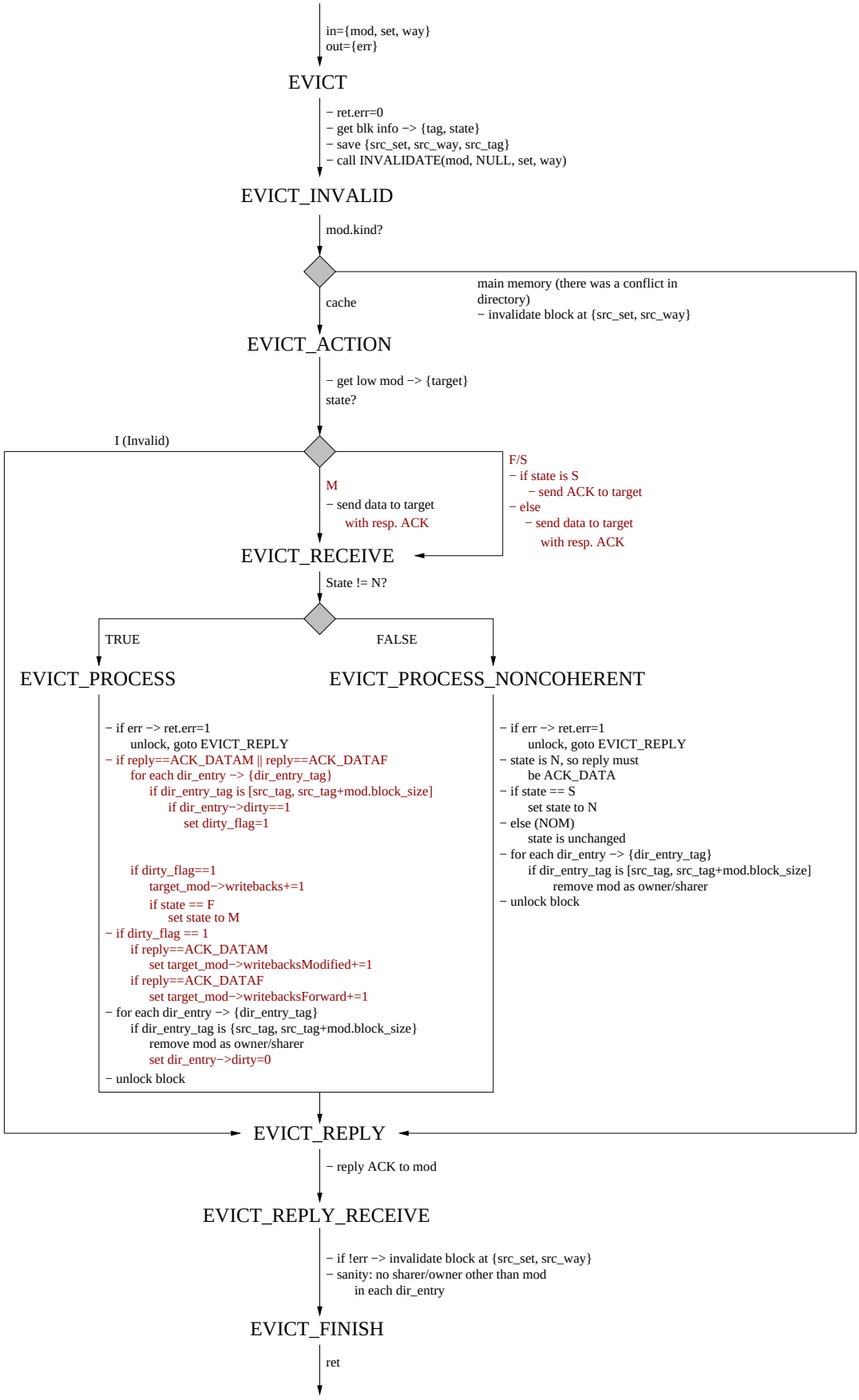

Figure F.6: EVICT Function for **MASI**

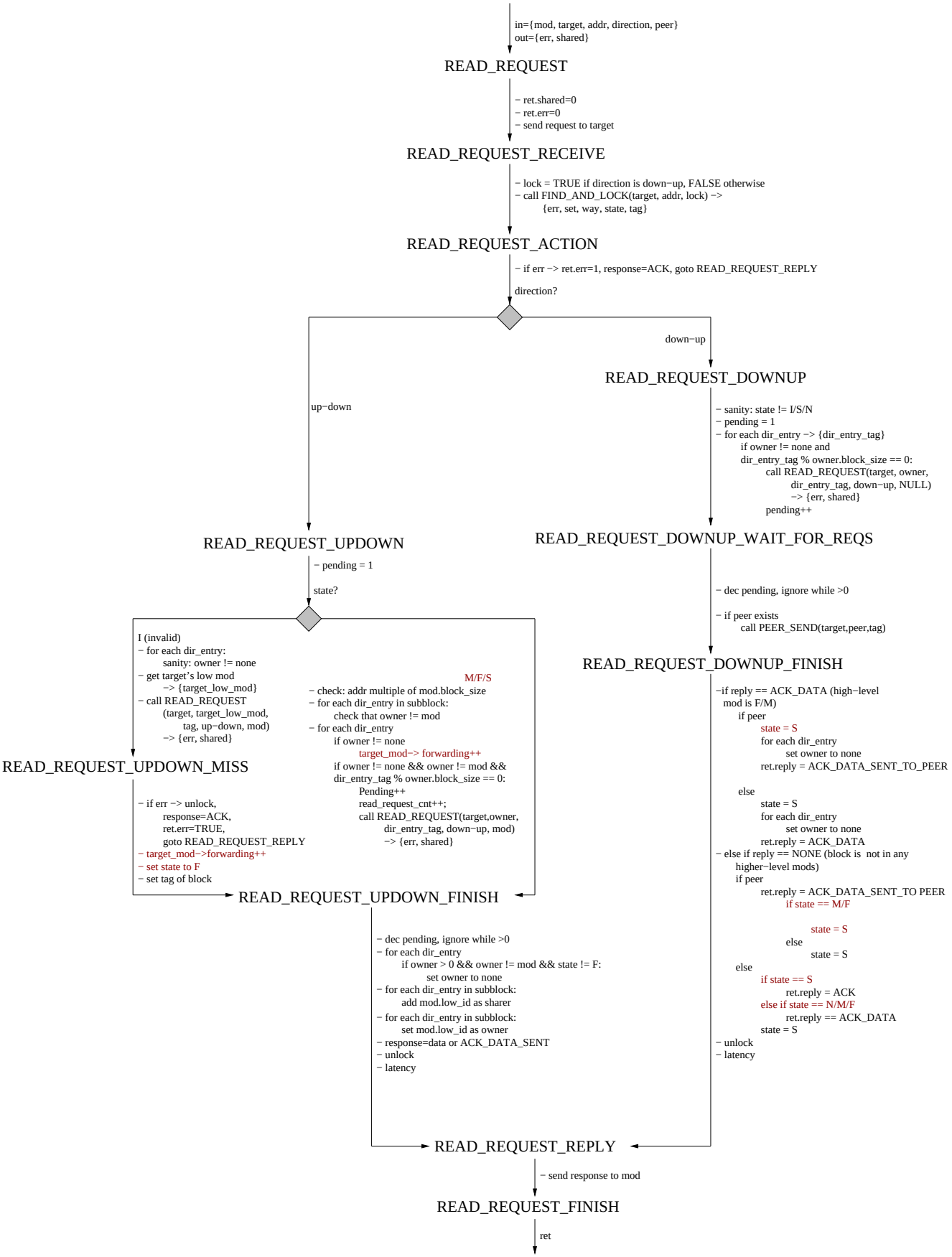

Figure F.7: READ_ REQUEST Function for **MASI**

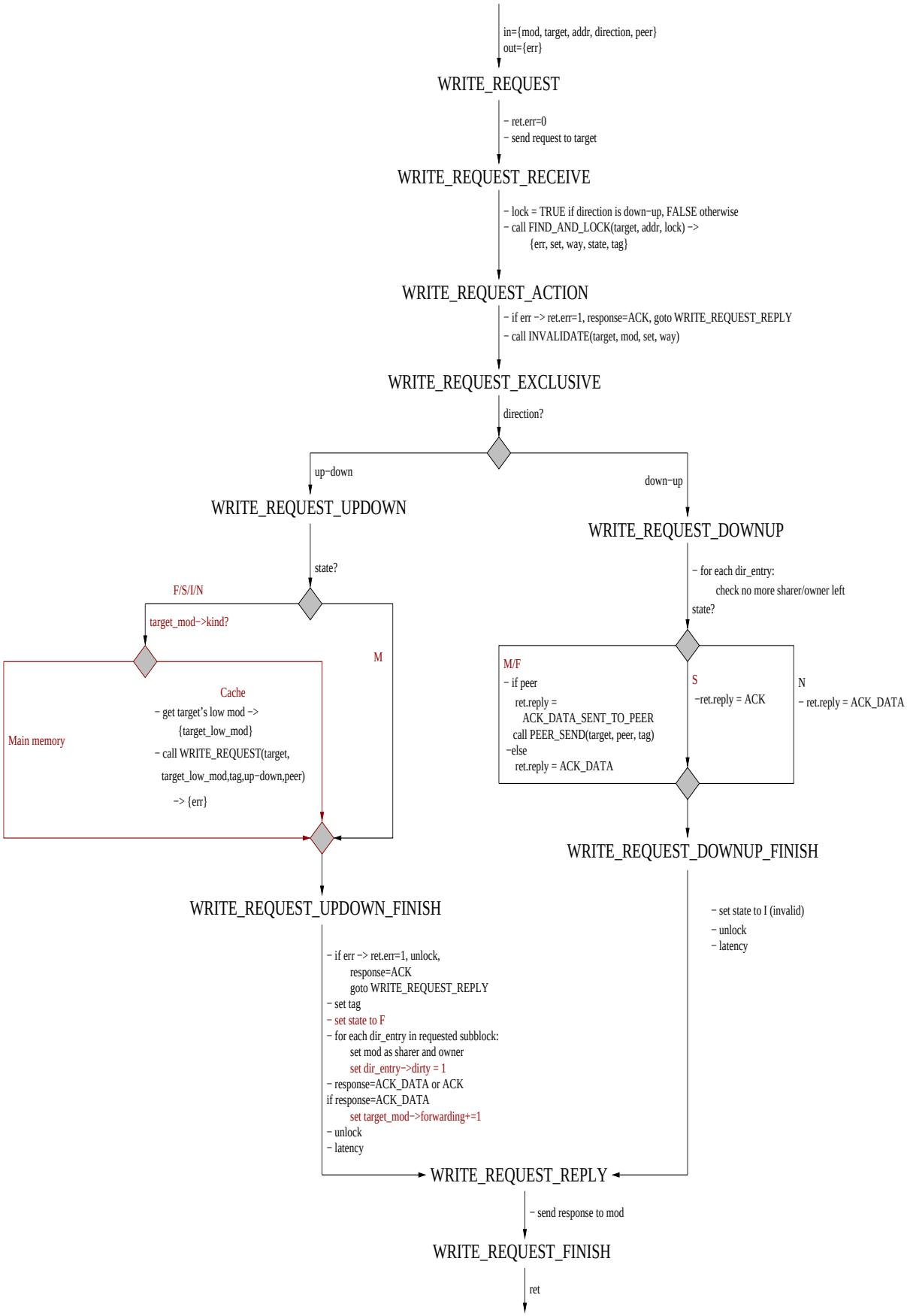

Figure F.8: WRITE_REQUEST Function for **MASI**

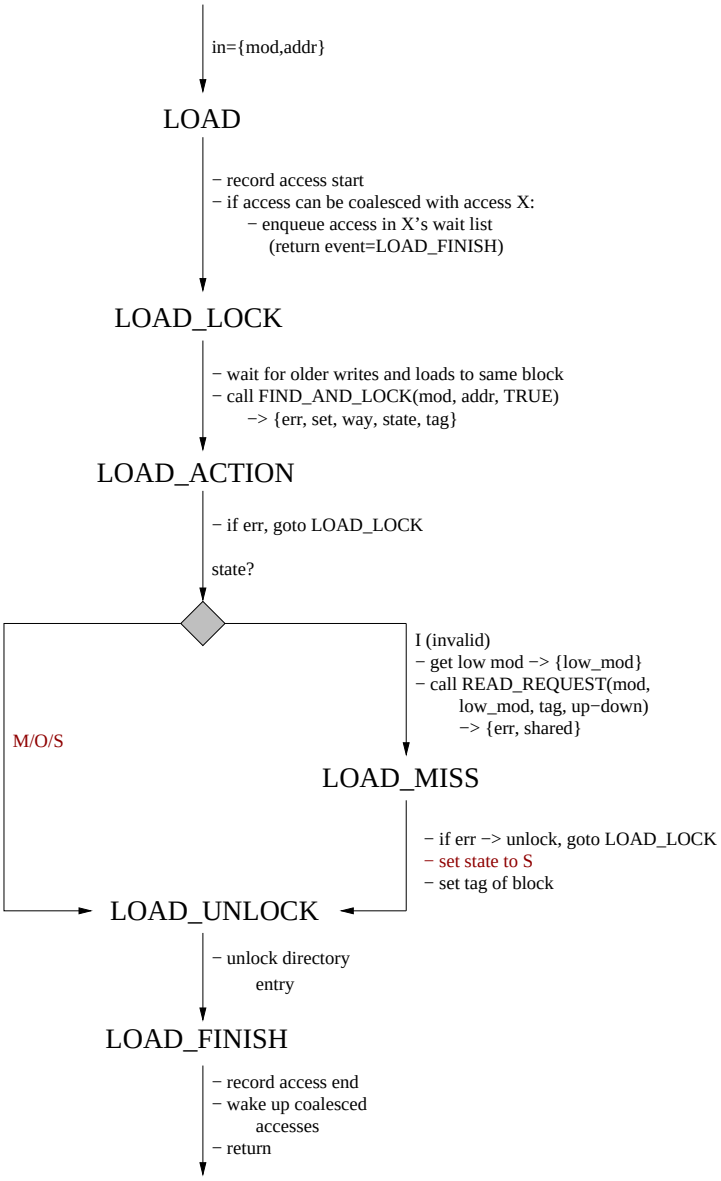

Figure F.9: LOAD Function for **MOSI**

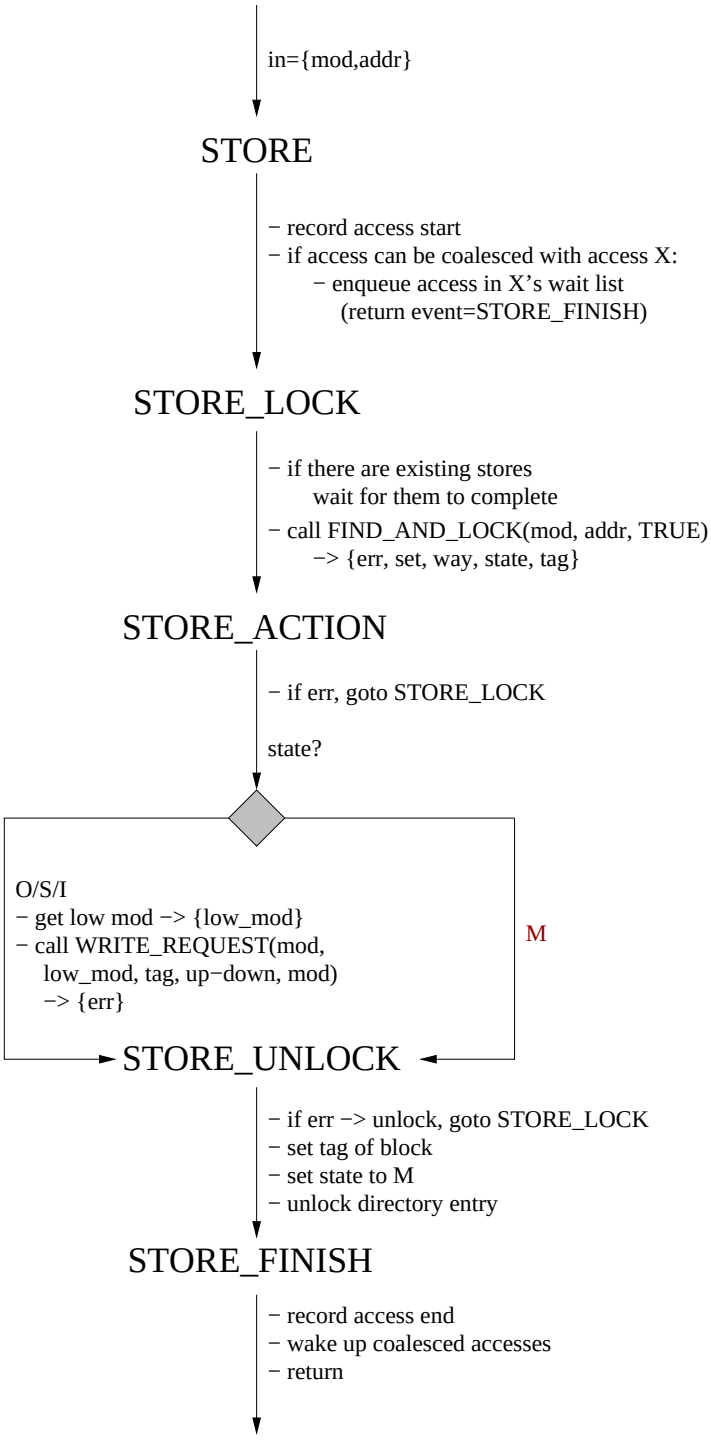

Figure F.10: STORE Function for **MOSI**

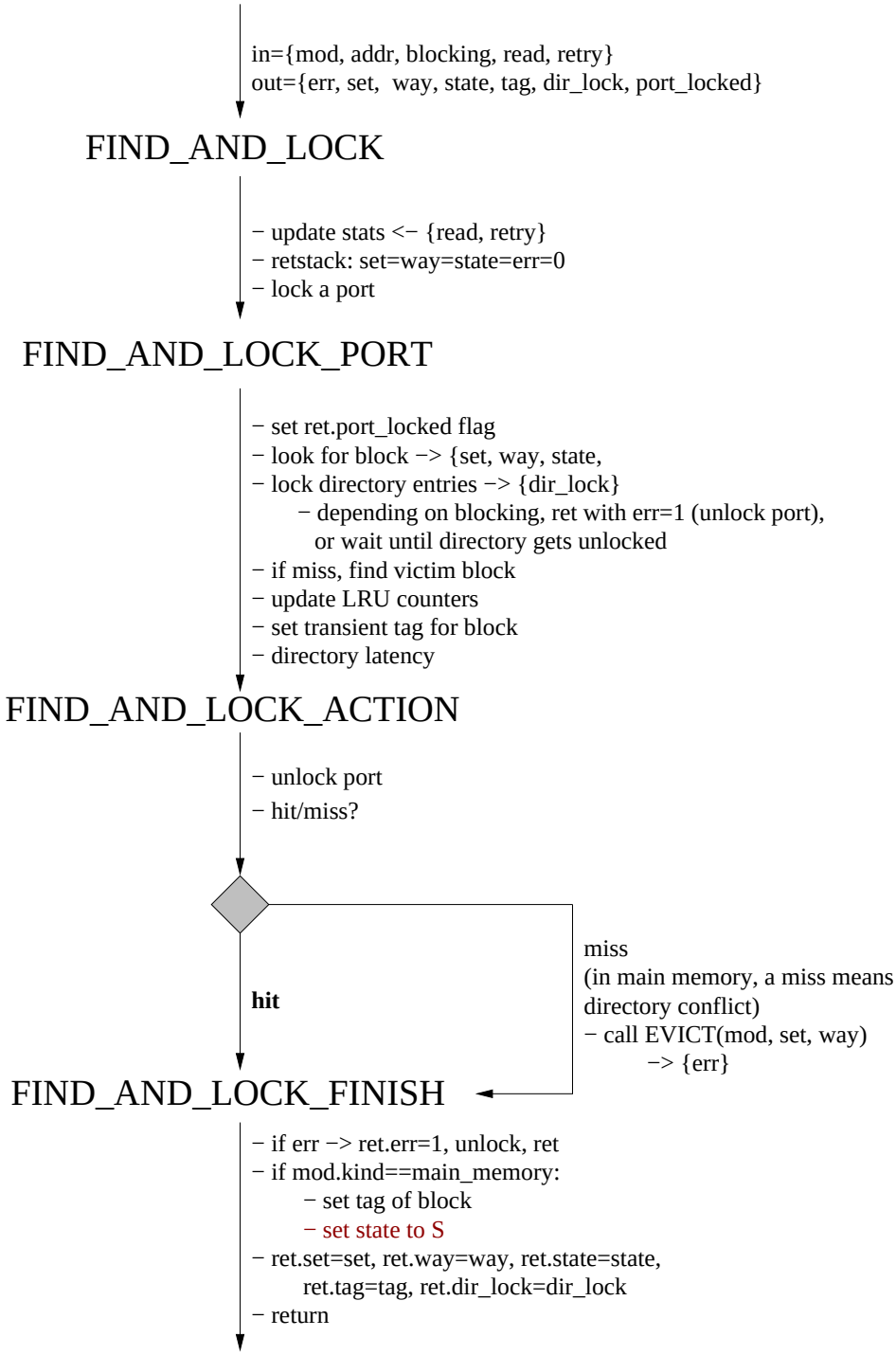

Figure F.11: FIND_ AND_ LOCK Function for **MOSI**

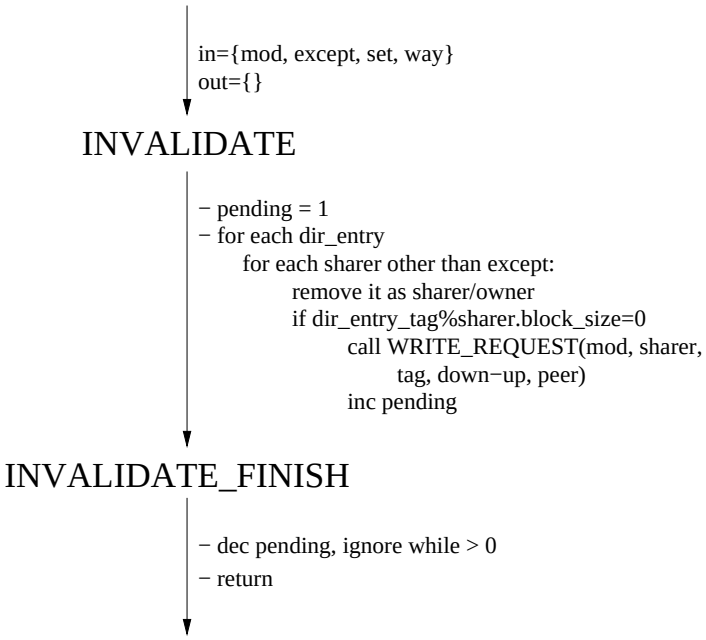

Figure F.12: INVALIDATE Function for **MOSI**

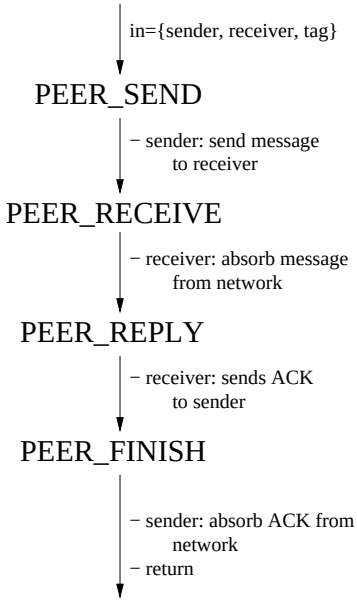

Figure F.13: PEER Function for **MOSI**

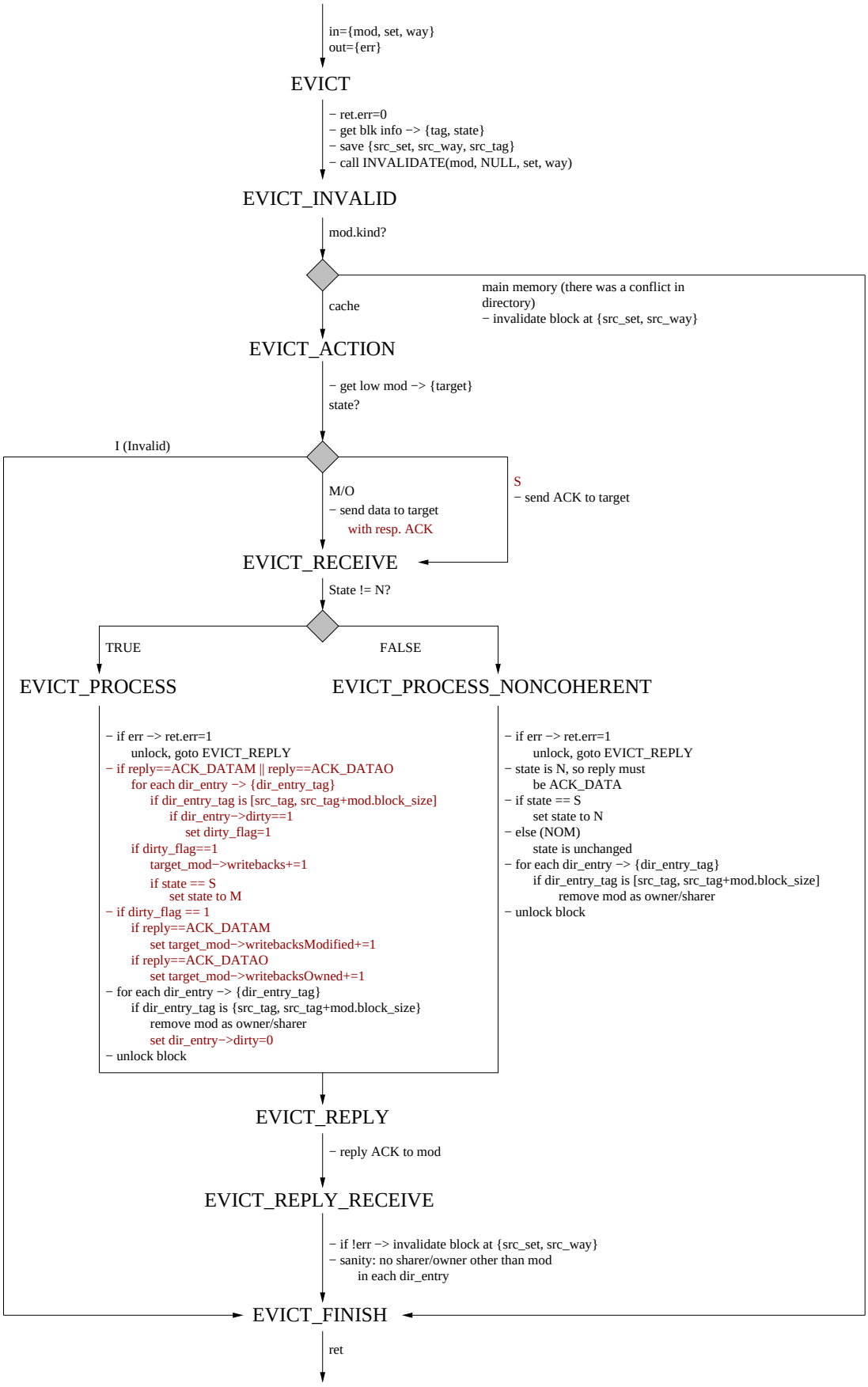

Figure F.14: EVICT Function for **MOSI**

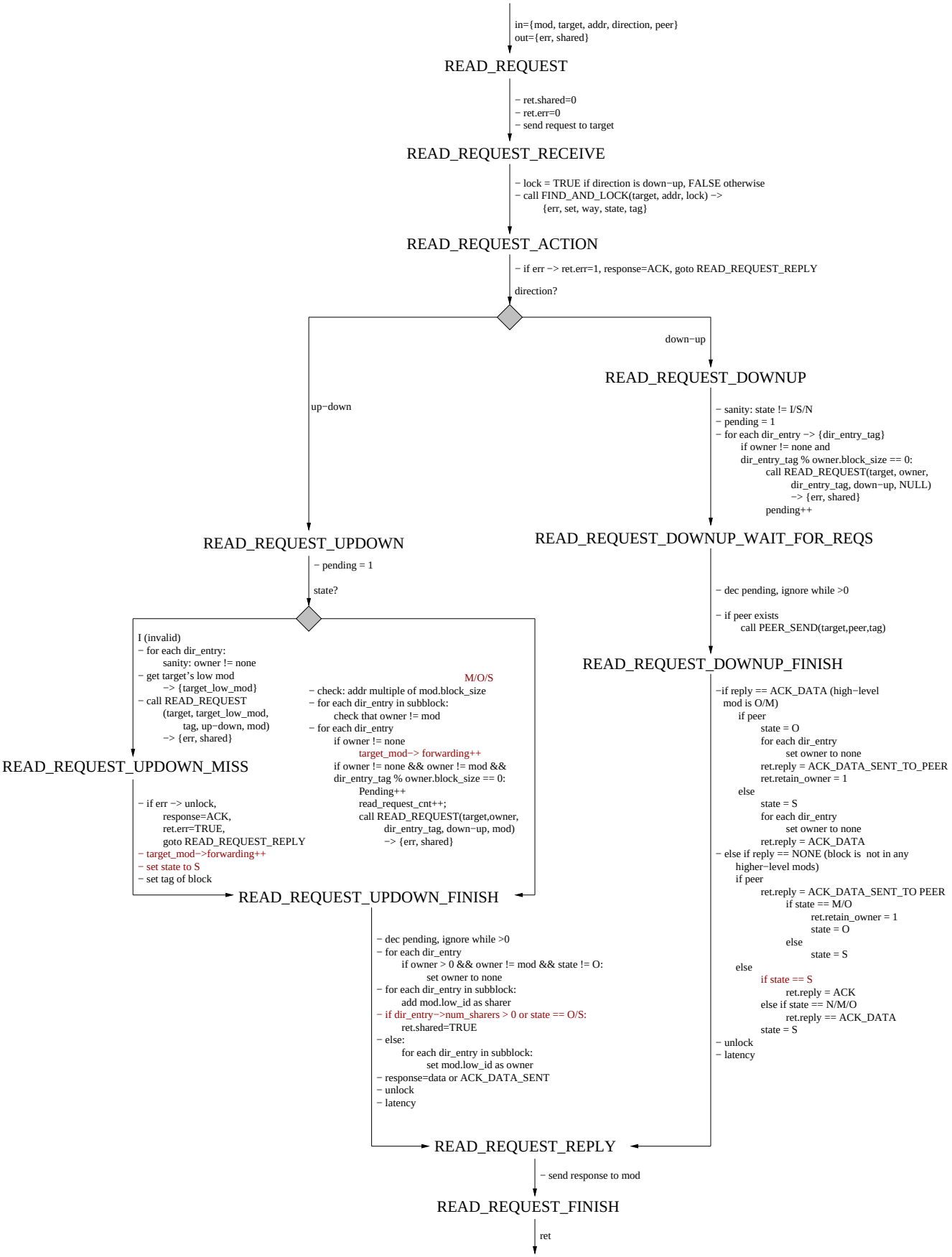

Figure F.15: READ_ REQUEST Function for **MOSI**

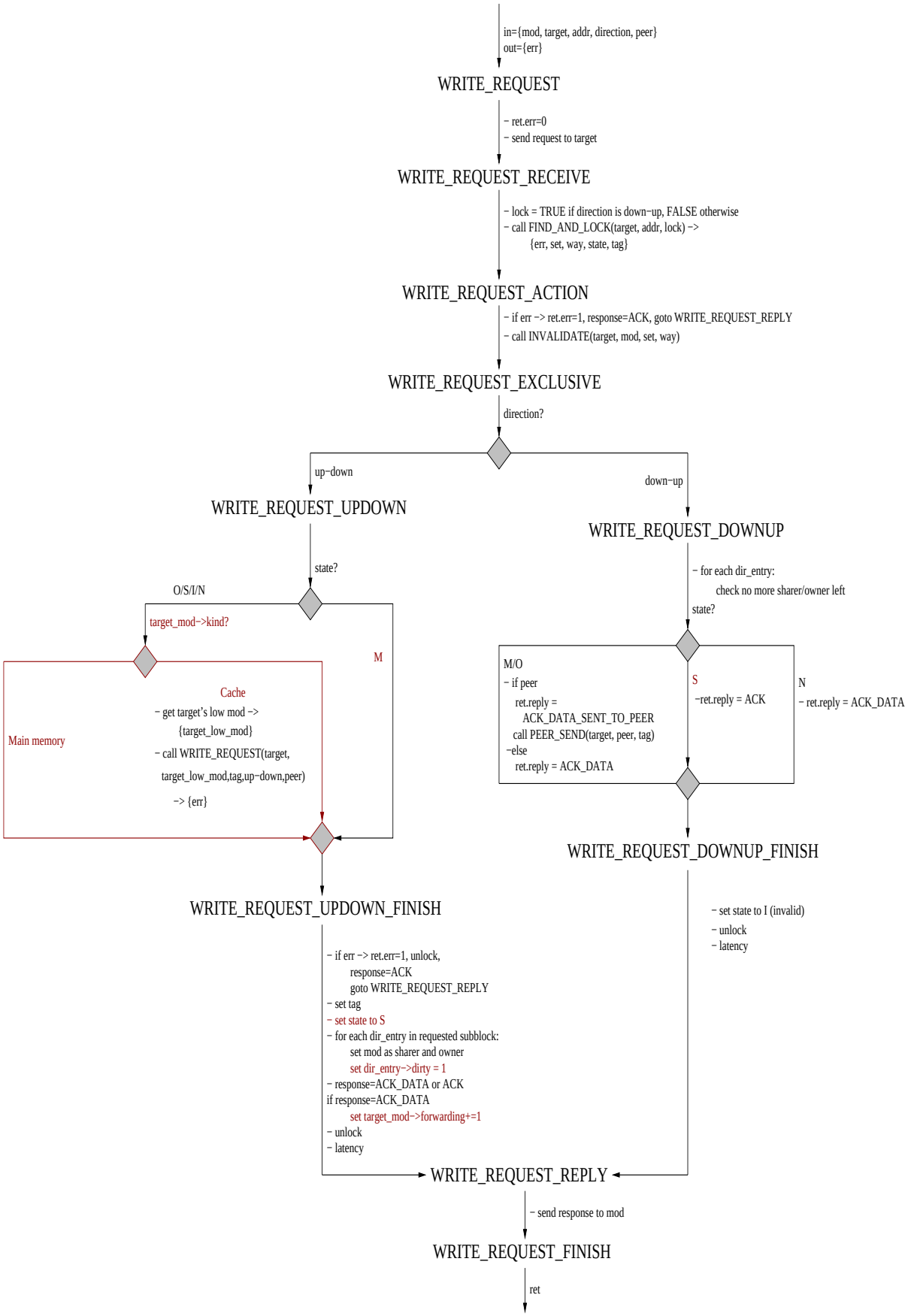

Figure F.16: WRITE_ REQUEST Function for **MOSI**

Appendix G

Control Flow for Section 4.3.3

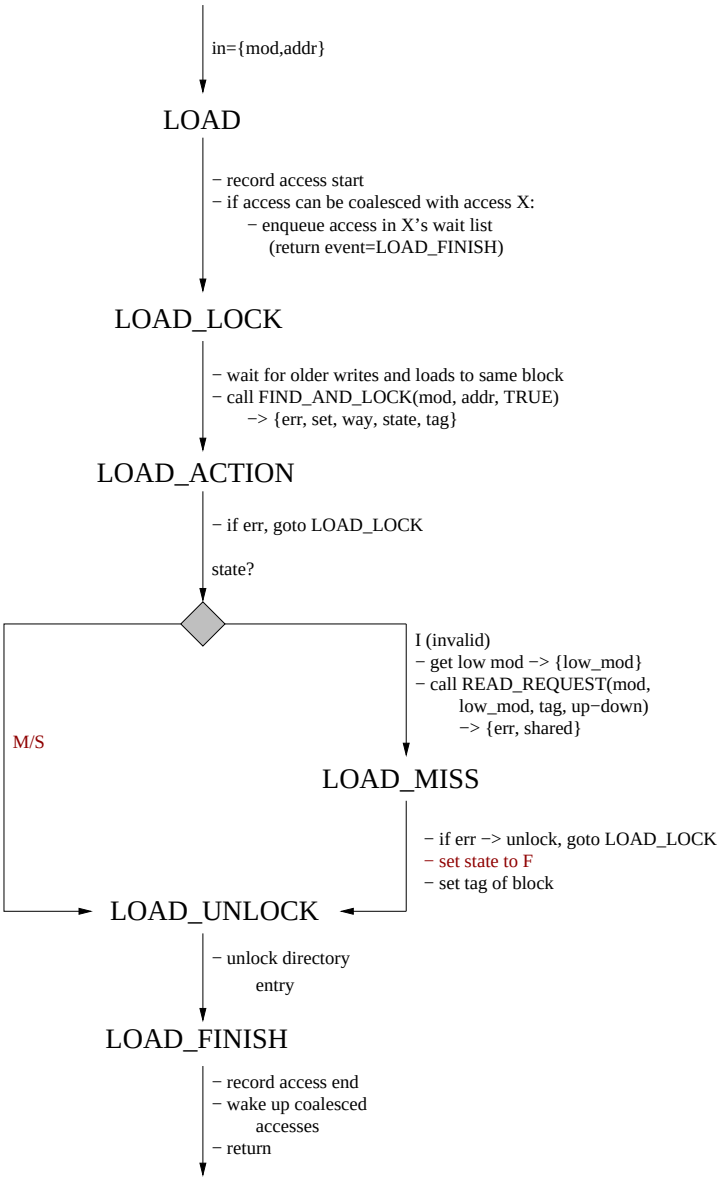

Figure G.1: LOAD Function for **MASI**

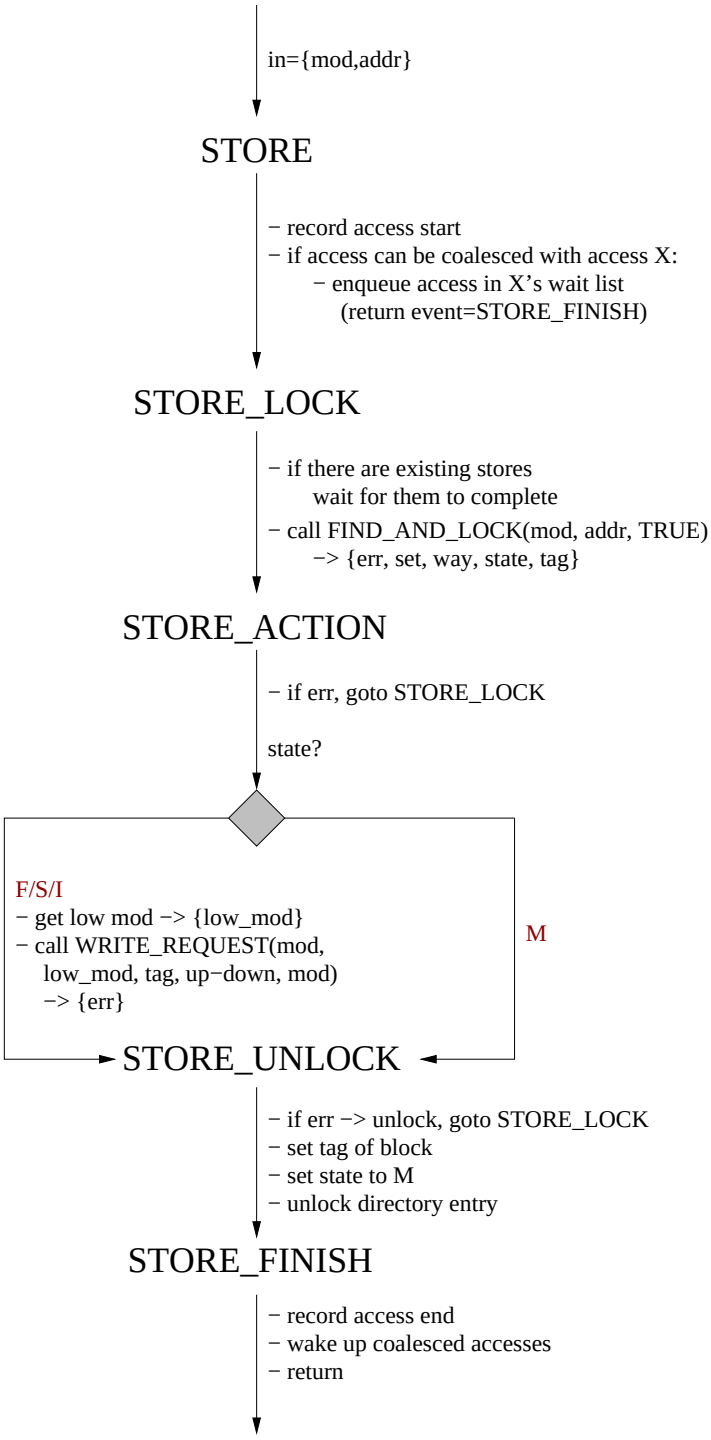

Figure G.2: STORE Function for **MASI**

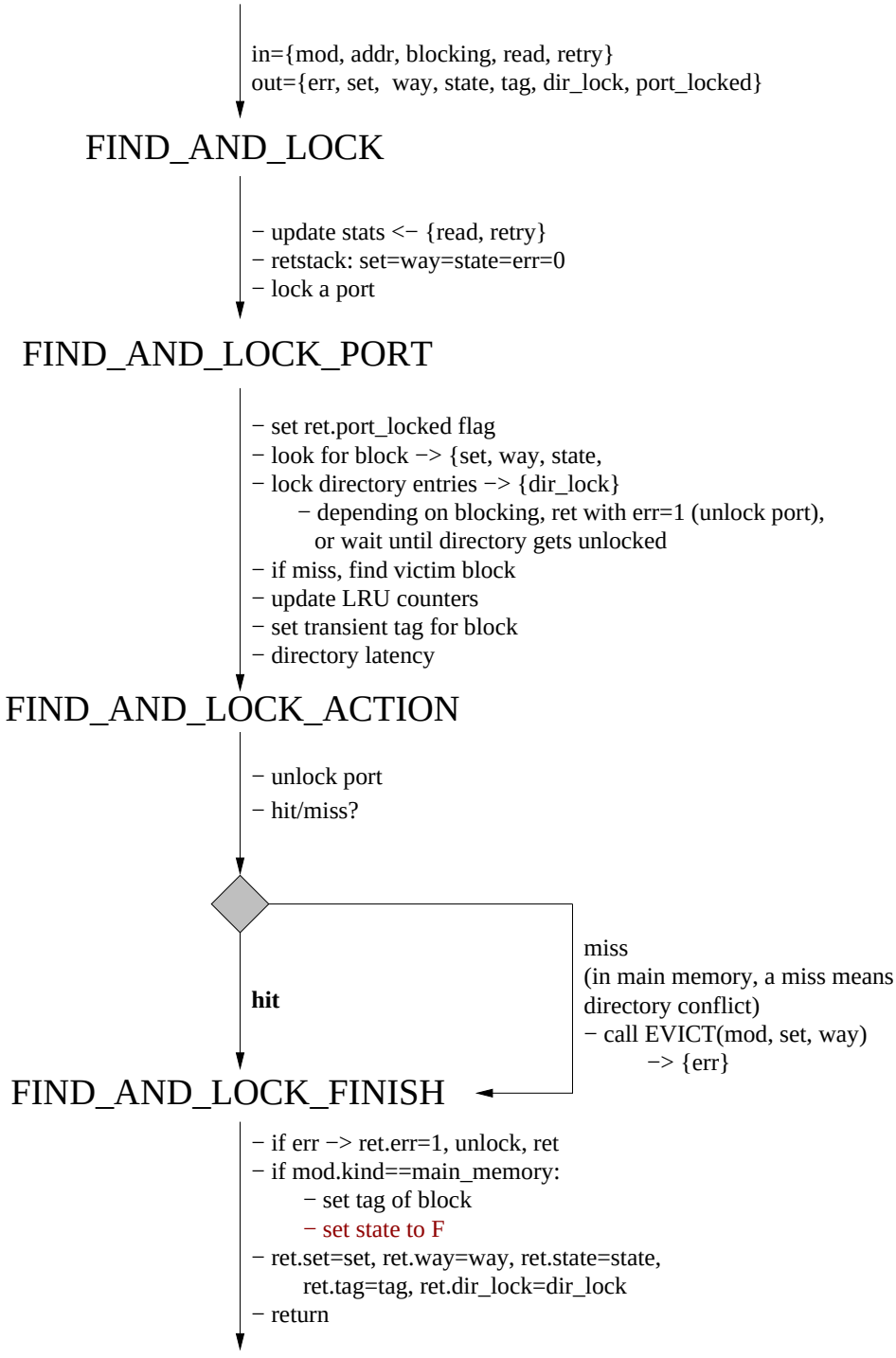

Figure G.3: FIND_ AND_ LOCK Function for **MASI**

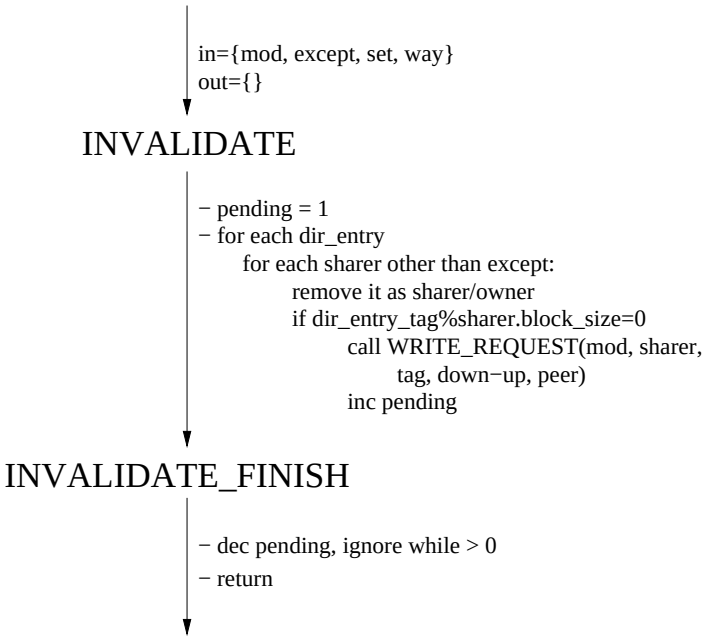

Figure G.4: INVALIDATE Function for **MASI**

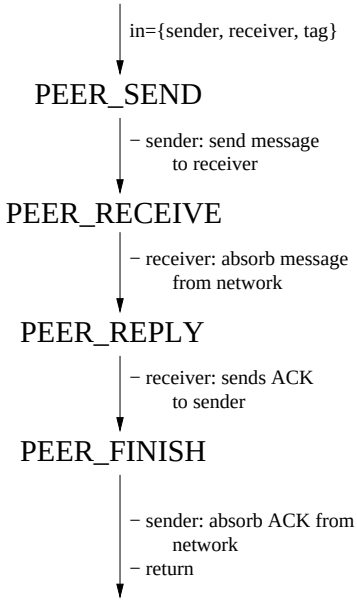

Figure G.5: PEER Function for **MASI**

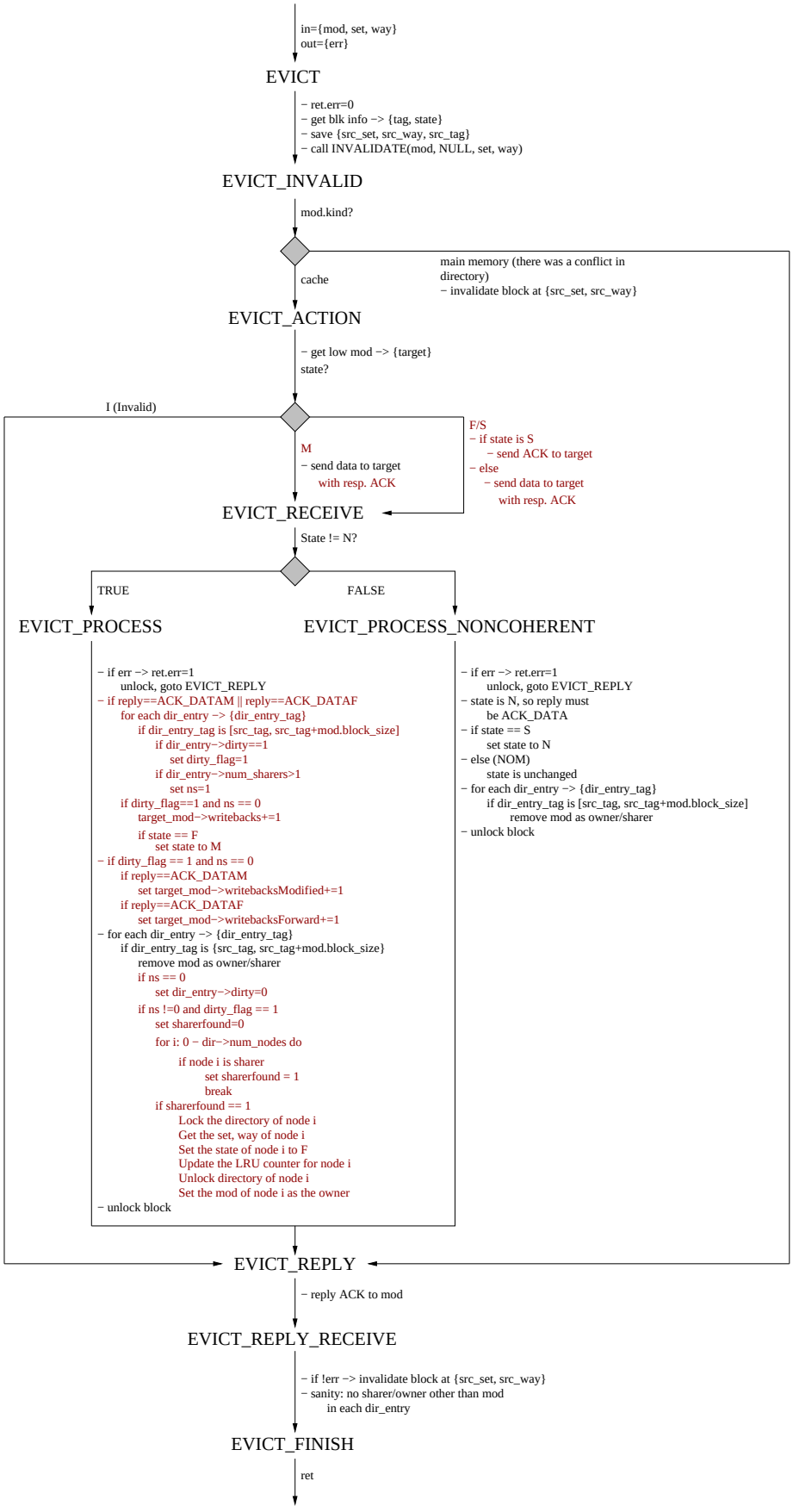

Figure G.6: EVICT Function for **MASI**

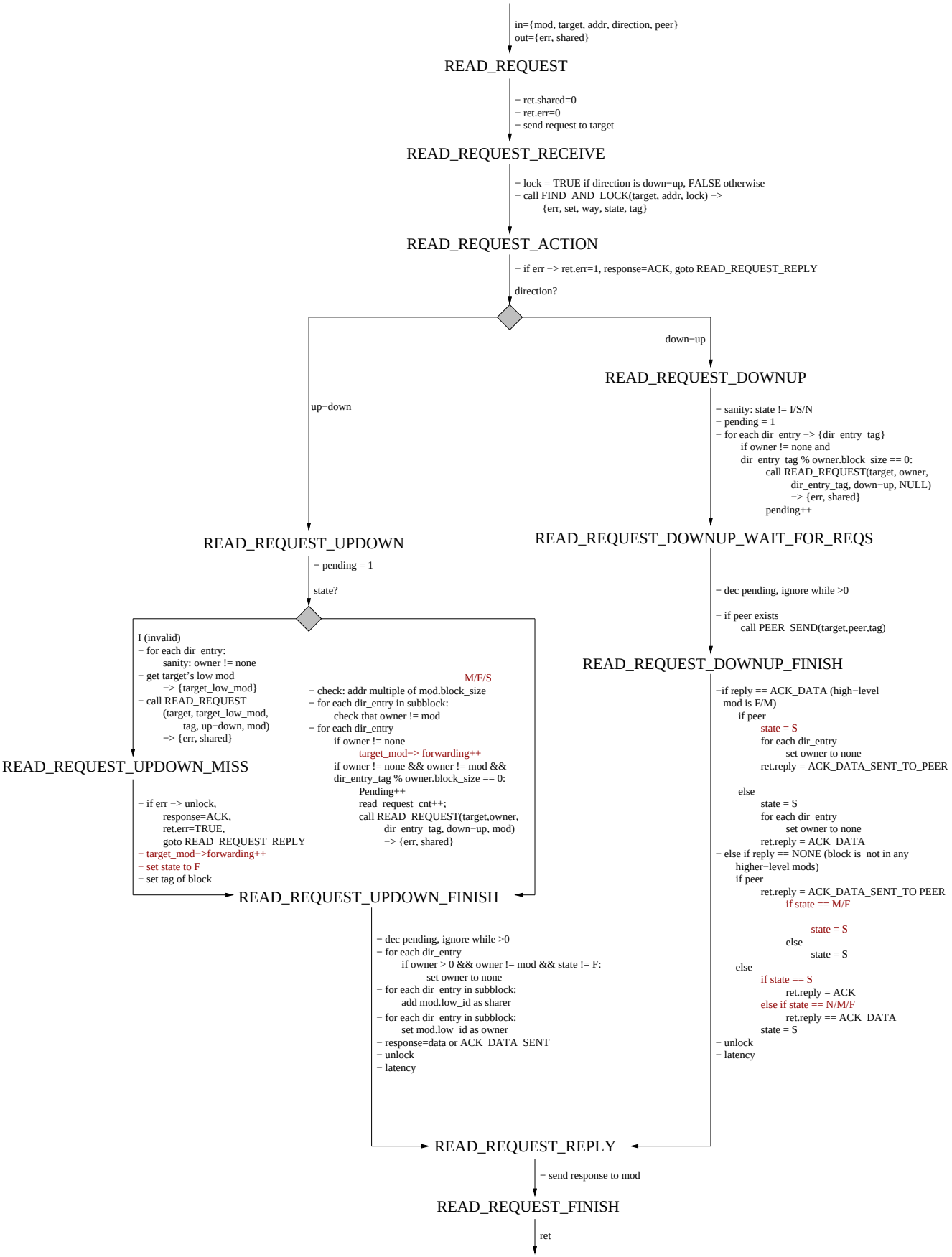

Figure G.7: READ_ REQUEST Function for **MASI**

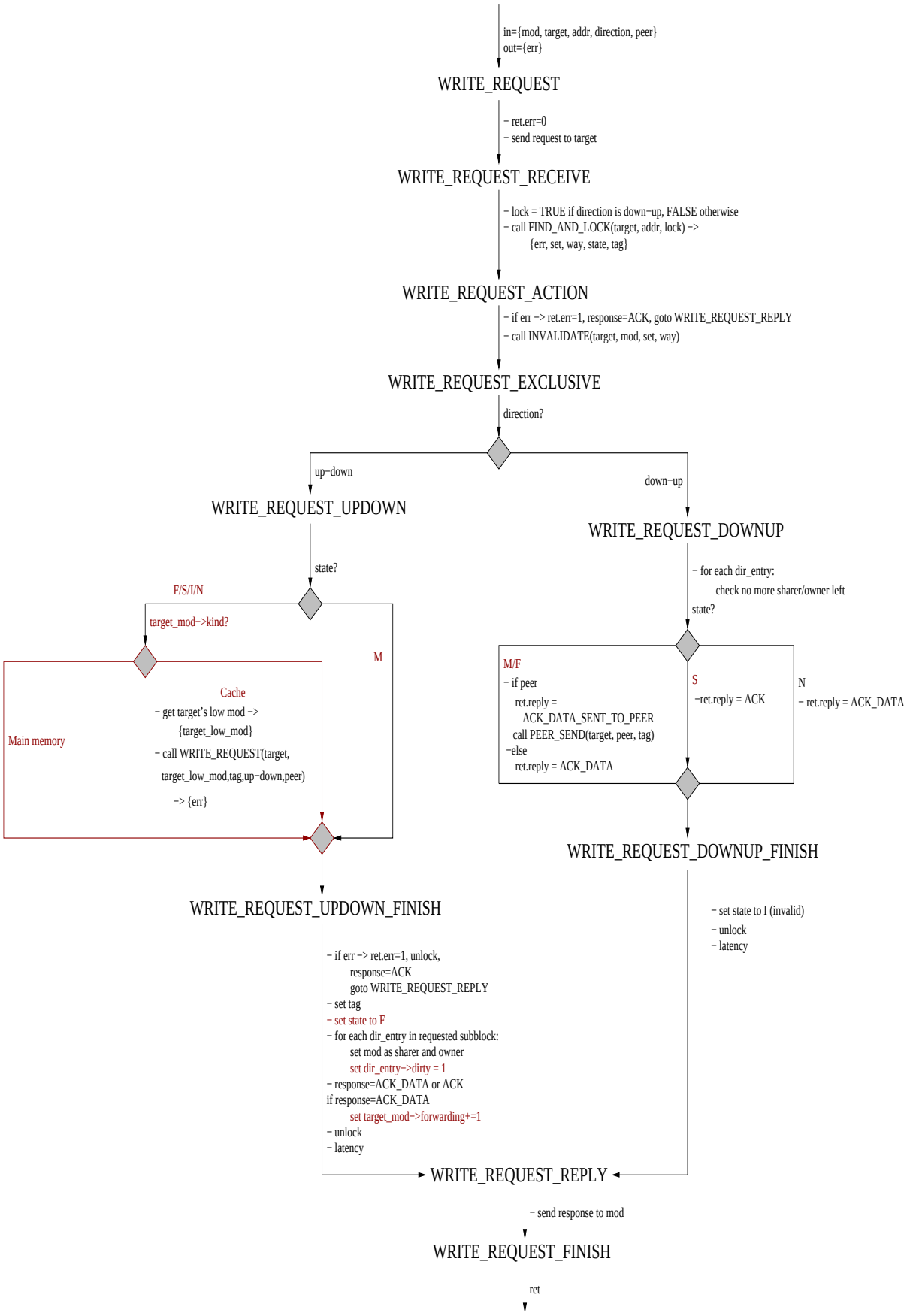

Figure G.8: WRITE_REQUEST Function for **MASI**

Appendix H

Control Flow for Section 4.3.4

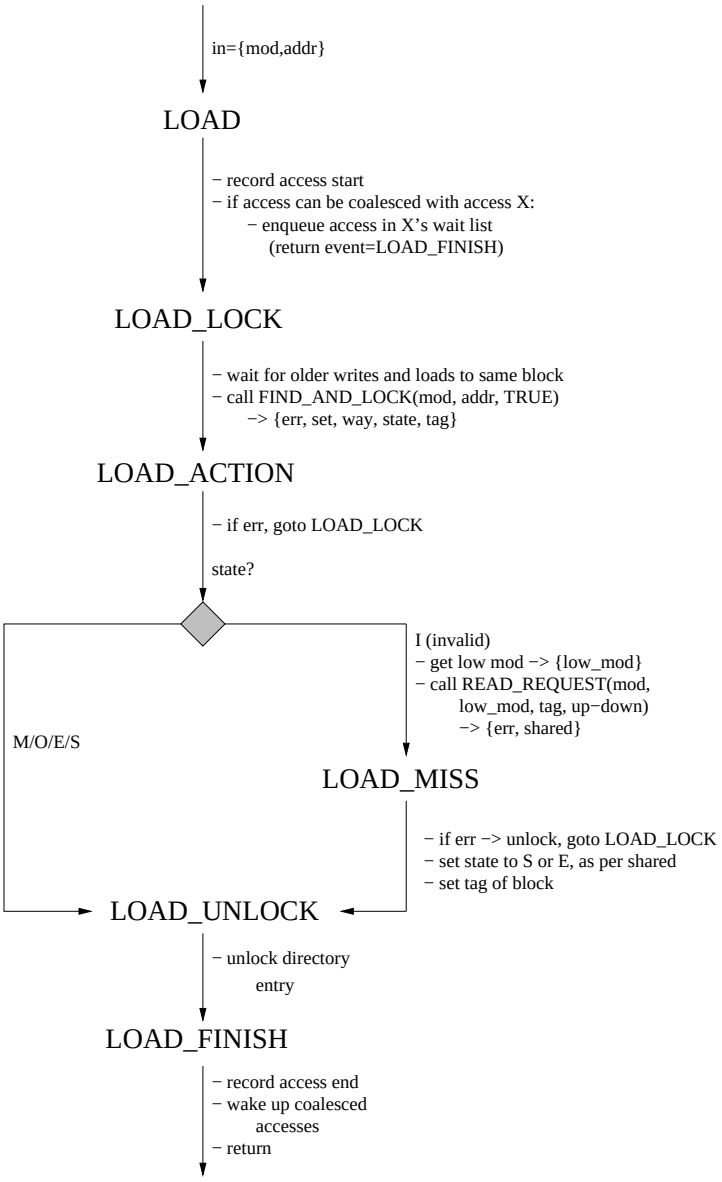

Figure H.1: LOAD Function for **MOESI**

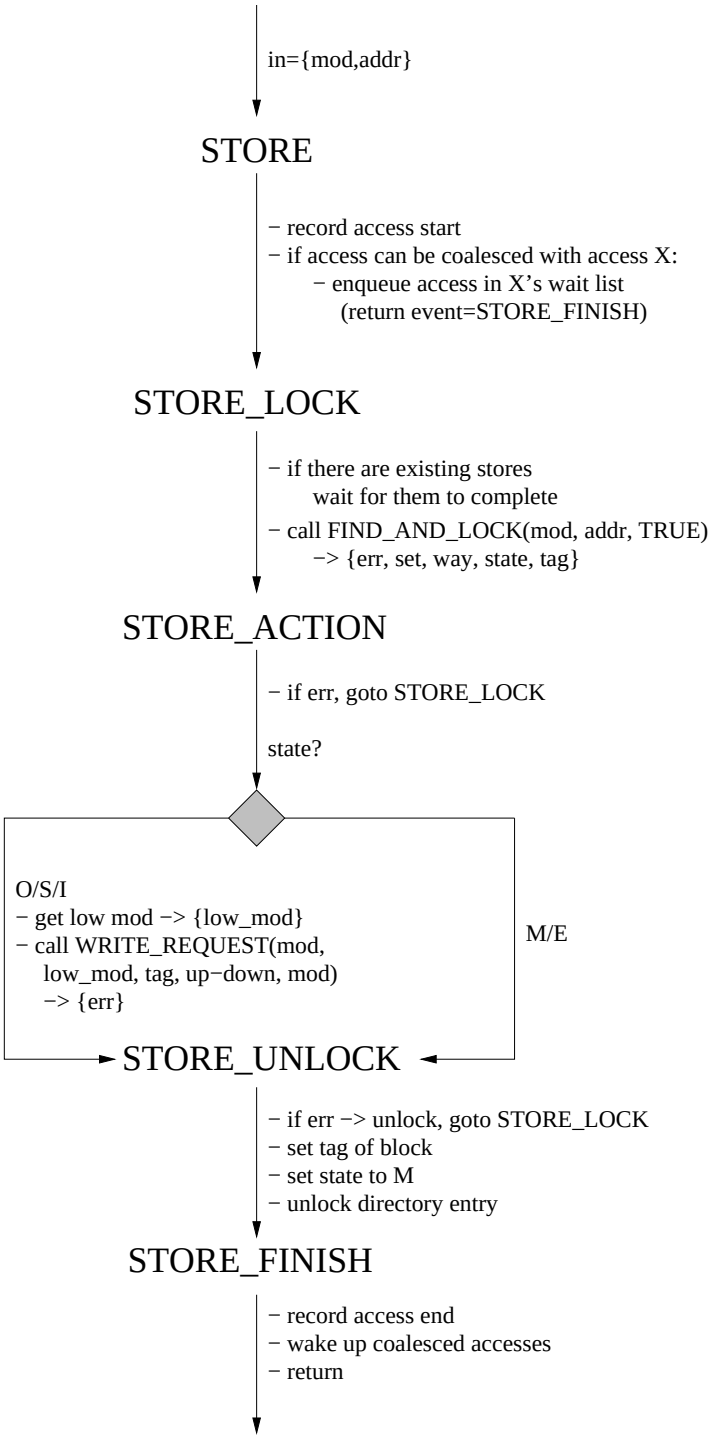

Figure H.2: STORE Function for **MOESI**

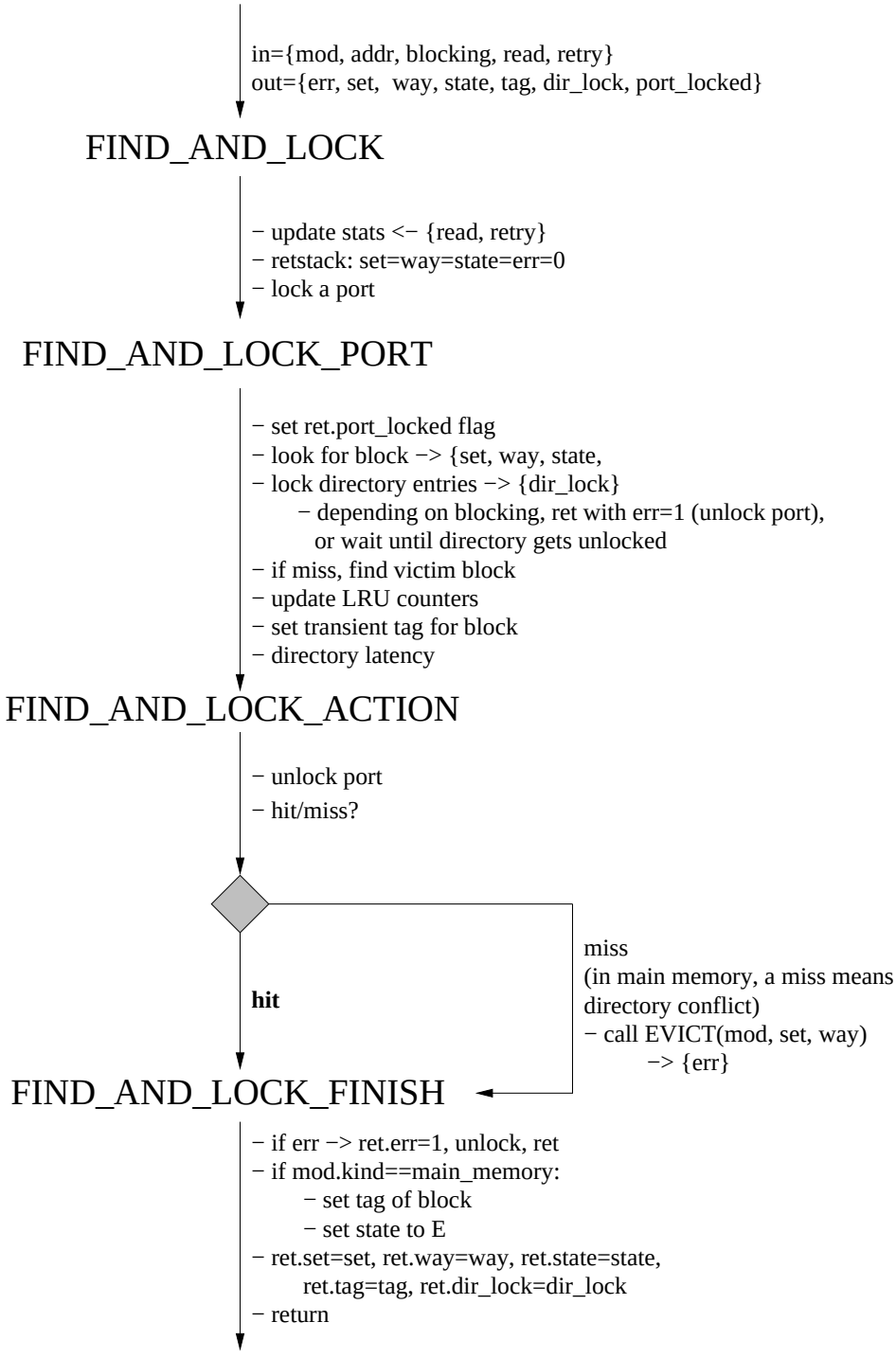

Figure H.3: FIND_ AND_ LOCK Function for **MOESI**

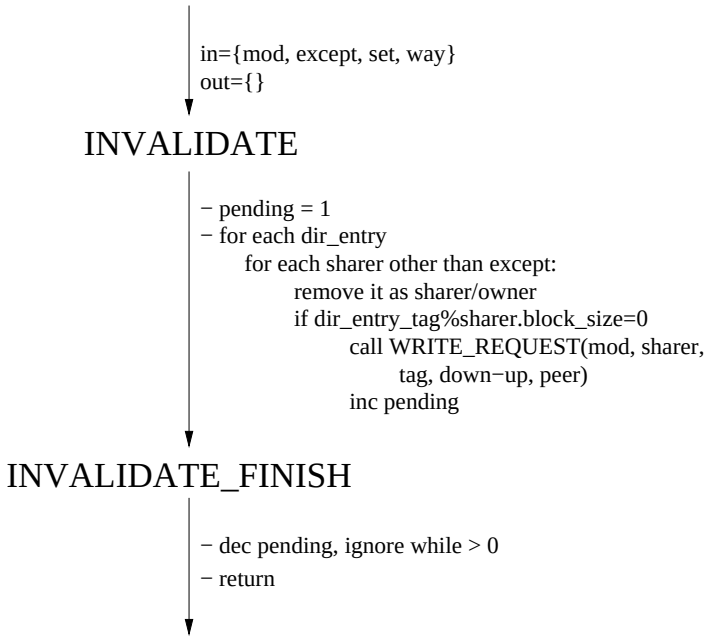

Figure H.4: INVALIDATE Function for **MOESI**

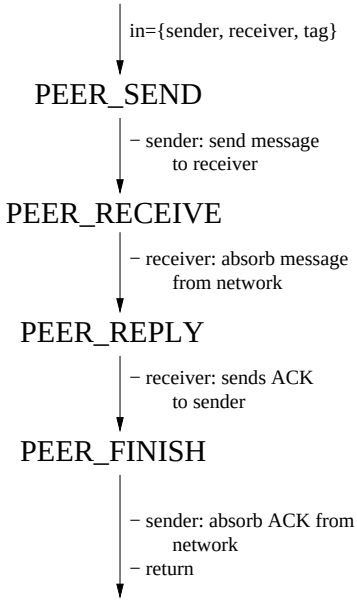

Figure H.5: PEER Function for **MOESI**

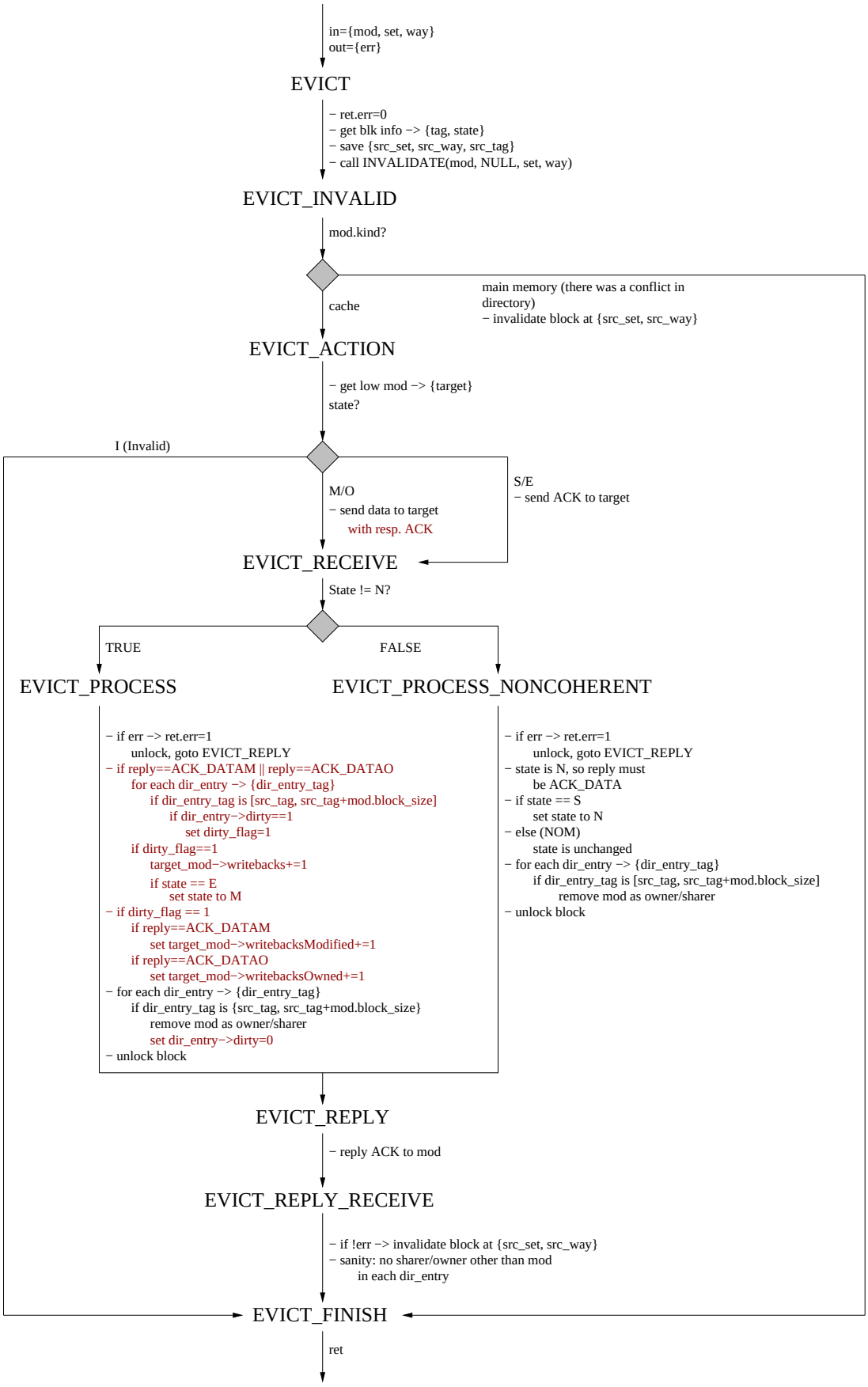

Figure H.6: EVICT Function for **MOESI**

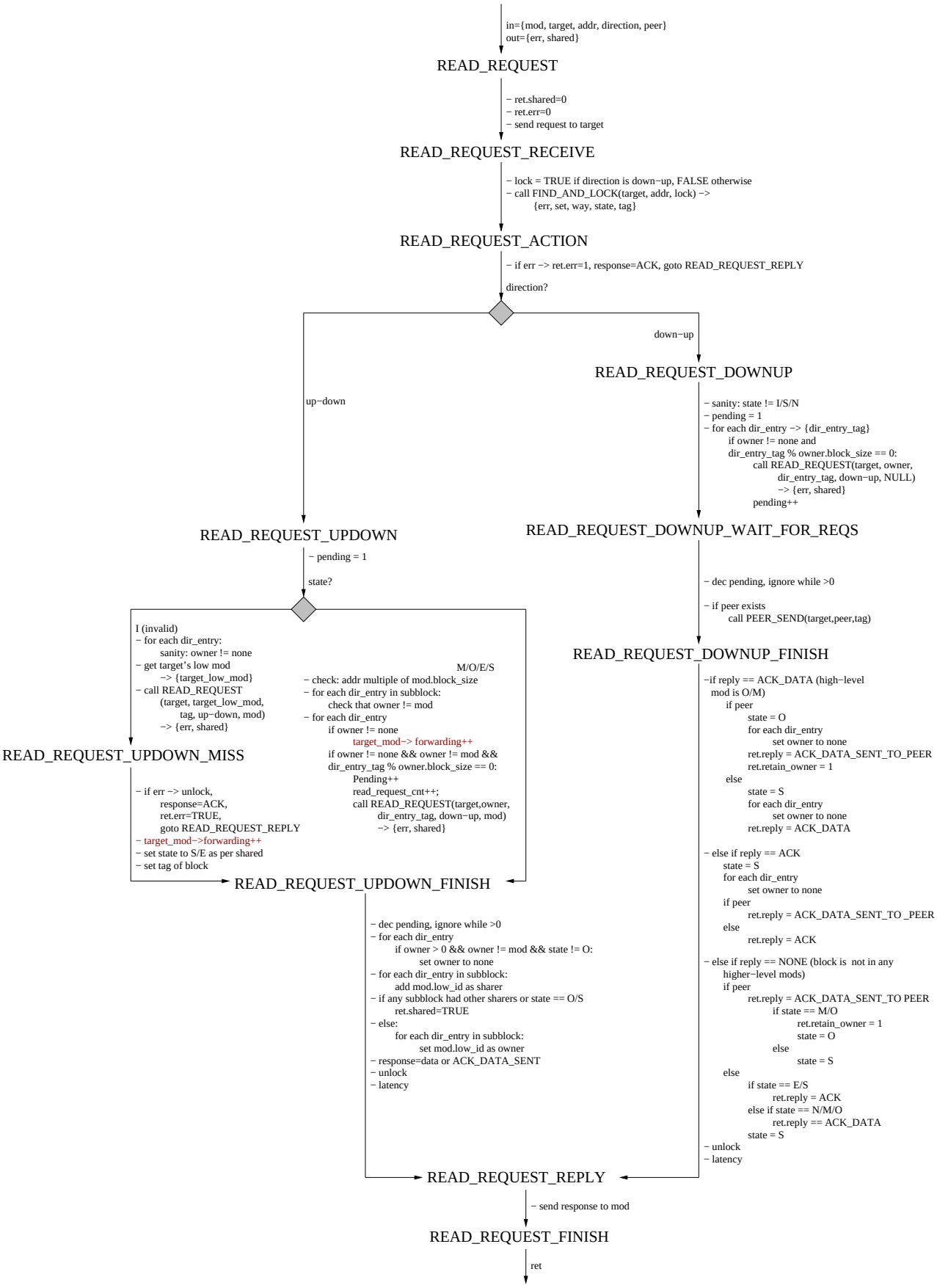

Figure H.7: READ_ REQUEST Function for **MOESI**

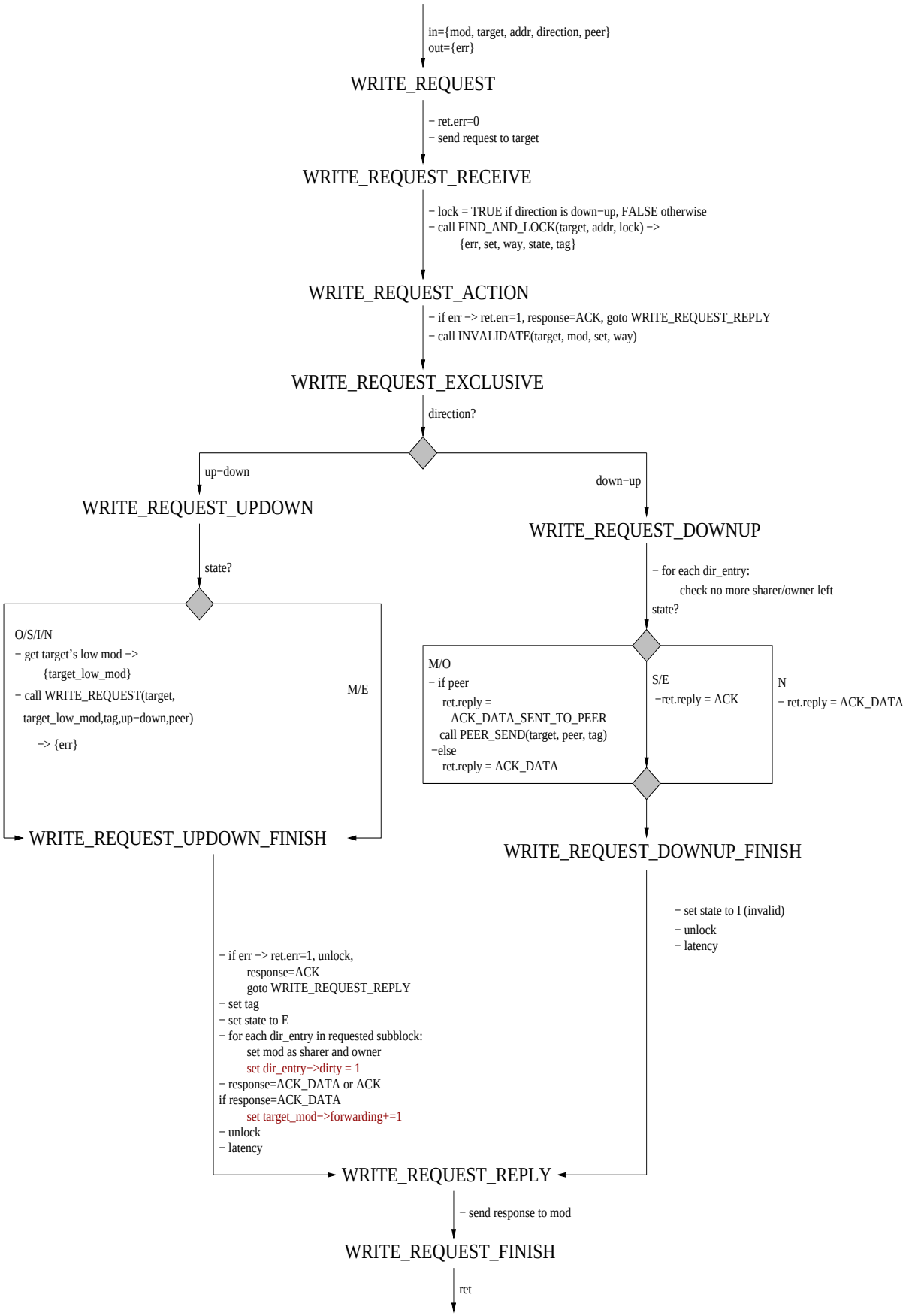

Figure H.8: WRITE_ REQUEST Function for **MOESI**