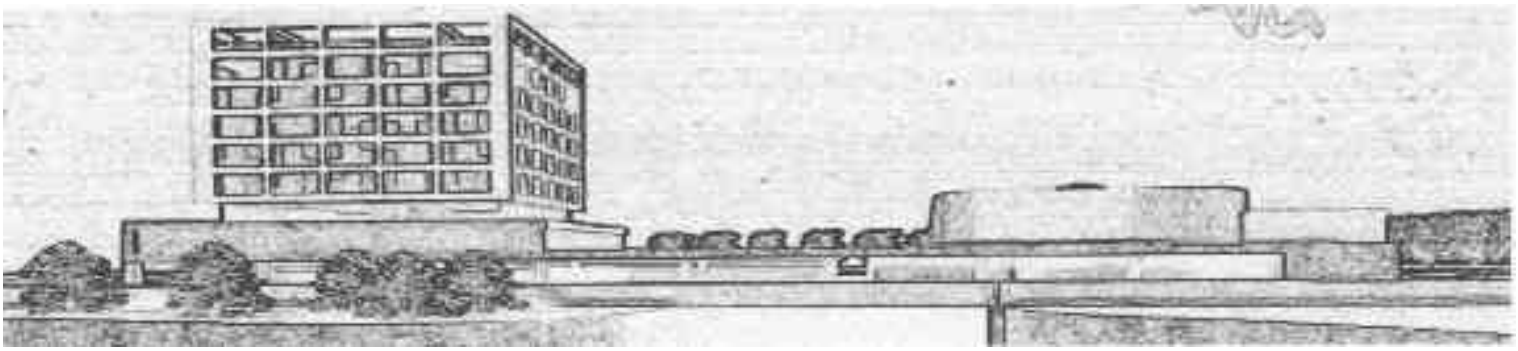
VOLUME III, NUMBER 2



REPORTS ON SYSTEMS AND COMMUNICATIONS

Móstoles (Madrid), October 2003
Depósito Legal: M-50653-2004
ISSN: 1698-7489

Table of Contents

Decoupled Interconnection of Distributed Memory Models*

Ernesto Jiménez, Antonio Fernández, and Vicente Cholvi

[1] U. Politécnica de Madrid, 28031 Madrid, Spain
ernes@eui.upm.es
[2] U. Rey Juan Carlos, 28933 Móstoles, Spain
afernandez@acm.org
[3] U. Jaume I, 12071 Castellón, Spain
vcholvi@inf.uji.es

Abstract. In this paper we present a framework to formally describe and study the interconnection of distributed shared memory systems. In our models we minimize the dependencies between the original systems and the interconnection system (that is, they are decoupled) and consider systems implemented with invalidation and propagation.
We first show that only fast (i.e. wait-free) memory models can be interconnected. We then show that causal and pRAM systems can be interconnected if they fulfill some restrictions, and for these cases, we present protocols to interconnect them. Finally, we present a protocol to interconnect cache systems.

Keywords: Distributed shared memory, memory models, interconnection systems, distributed algorithms, impossibility result, correctness proofs.

1 Introduction

Distributed shared memory is an abstraction used for process communication. In this abstraction, processes read and write variables of a shared memory, which is usually implemented with distributed memory and message passing. Depending on the semantics of the shared memory a number of consistency models have been proposed in the literature [2, 13]. Some of the most popular models are the sequential [20], causal [5], pRAM [23], and cache [16]. Several protocols have been proposed to implement distributed shared memory systems that implement these consistency models.

In this work we study the interconnection of distributed shared memory systems. By this we mean the adding of an *interconnection system* to several

* This work has been partially supported by the Spanish MCyT under grant TIC2001-1586-C03-01, the Comunidad de Madrid under grant 07T/0022/2003, and the Universidad Rey Juan Carlos under grant PPR-2003-37.

existing distributed shared memory systems that implement a given consistency model to obtain a single distributed shared memory system that implements the same consistency model. This line of work was started in [14], where the interconnection of causal propagation-based systems was studied. In this work we use much weaker assumptions on the systems to be interconnected, consider also invalidation-based systems, and explore other models as well.

Our Contributions In this work we first define a framework for the interconnection of systems. We formalize several classes of interactions, both for propagation and invalidation-based protocols, between the existing systems and the interconnection system. All these classes decouple the systems from the interconnection system, unlike the previous model [14].

Then, we show that systems that implement *non-fast* consistency models cannot be interconnected in these classes. A fast consistency model is one that allows implementations of read and write operations that return control after only local computations. After that, we study the interconnection of pRAM and causal systems. We show that they can not be interconnected in general, but can under certain restrictions. We give sufficient conditions and the corresponding protocols to do so. Finally, we also show that systems that implement cache consistency can always be interconnected.

Note that this is the first work that studies the interconnection of pRAM and cache systems, that considers both propagation and invalidation, and that shows that certain interconnections are in fact impossible. Our protocols need not be very useful nor efficient in practice, since all we try to do with this work is to define the bounds of the possibilities of interconnection.

Related Work As far as we know, the interconnection of distributed shared systems has only been studied in [14]. For message passing, Rodrigues and Verissimo [27], Adly and Nagi [1], and Baldoni et al. [10] have studied the interconnection of small causal message passing systems into larger systems. This could be an alternative way to obtain a large distributed shared memory system from smaller systems, since the causal message passing system can be directly used to obtain causal memory. However, the purpose of this work is to respect the original distributed shared memory systems to be interconnected, and hence this technique is not valid for our study.

The rest of the paper is organized as follows. In Section 2 we introduce our framework for the interconnection of systems. In Section 3 we show the impossibility result for non-fast consistency models. In Section 4 we study the interconnection of pRAM systems, in Section 5 the interconnection of causal systems, and in Section 5 we show how to interconnect cache systems. Finally, in Section 7 we present concluding remarks.

2 Definitions and Notation

We consider *distributed shared memory systems* (or *systems* for short) formed by a collection of *application processes* that interact via a *shared memory* formed

by a set of *variables*. All the interactions between the application processes and the memory are done through read and write operations (*memory operations*) on variables of the memory.

Each memory operation is applied on a named variable and has an associated value. A write operation of the value v in the variable x, denoted $w(x)v$, stores v in the variable x. A read operation of the value v from the variable x, denoted $r(x)v$, reports to the issuing application process that the variable x holds the value v. To simplify the analysis, we assume that a given value is written at most once in any given variable and that the initial values of the variables are set by using fictitious write operations.

2.1 Consistency Model of a System An *execution* α of a system S is the set of read and write operations observed in some run R of system S.

Definition 1 (Process Order). *Let p be a process of S and $op, op' \in \alpha$. Then op precedes op' in p's* process order, *denoted $op \prec_p op'$, if op and op' are operations issued by p, and op is issued before op'.*

Definition 2 (Execution Order). *Let $op, op' \in \alpha$. Then op precedes op' in the* execution order, *denoted $op \prec op'$, if:*
1. op and op' are operations from the same process p and $op \prec_p op'$, or
2. $op = w(x)v$ and $op' = r(x)v$, or
3. $\exists op'' \in \alpha : op \prec op'' \prec op'$

We denote by α_p the subset of operations obtained by removing from α all read operations issued by processes other than p. We also denote by $\alpha(x)$ the subset of operations obtained by removing from α all the operations on variables other than x.

Definition 3 (View). *Let $\prec^o$ be an order on execution α, and let $\alpha' \subseteq \alpha$. A* view *β of α' preserving $\prec^o$ is a sequence formed by all operations of α' such that this sequence preserves the order $\prec^o$.*

Note that if $\prec^o$ applied on α' is not a total order, there can be several views of α'. We use $op \xrightarrow{\beta} op'$ to denote that op precedes op' in a sequence of operations β. We will omit the name of the sequence when it is clear from the context. We will also use $\beta_1 \rightarrow \beta_2$, where β_1 and β_2 are sequences of operations, to denote that all the operations in β_1 precede all the operations in β_2.

Definition 4 (Legal View). *Let $\prec^o$ be an order on execution α, and let $\alpha' \subseteq \alpha$. A view β of α' preserving $\prec^o$ is* legal *if $\forall r(x)v \in \alpha'$:*
a) $\exists\ w(x)v \in \alpha' : w(x)v \xrightarrow{\beta} r(x)v$, and
b) $\nexists\ w(x)u \in \alpha' : w(x)v \xrightarrow{\beta} w(x)u \xrightarrow{\beta} r(x)v$.

By using this definition of legal view, we can define systems satisfying the causal, pRAM, and cache consistency models.

Definition 5 (Causal, pRAM, or Cache System). *A system S is causal if for every execution α and every process p there is a legal view β_p of α_p preserving $\prec$ on α. A system S is pRAM if for every execution α and every process p there is a legal view β_p of α_p, preserving $\prec_q$ on α_p, for all q. A system S is cache if for every execution α and every variable x there is a legal view β_x of $\alpha(x)$ preserving $\prec$ on $\alpha(x)$.*

2.2 System Architecture From a physical point of view, we consider distributed systems formed by a set of *nodes* and *a network* that provides communication among them. The essence of this model has been taken from [9]. The application processes of the system are actually executed in the nodes of the distributed system. We assume that the shared memory abstraction is implemented by a *memory consistency system* (*MCS*). The MCS is composed of *MCS-processes* that use the local memory at the various nodes and cooperate following a distributed algorithm, or *MCS-protocol*, to provide the application processes with the impression of having a shared memory. The MCS-processes are executed at the nodes of the distributed system and exchange information as specified by the MCS-protocol. They use the communication network to interact if they are in different nodes. Each MCS-process can serve several application processes, but an application process is assigned to only one local MCS-process. For each application process p we use $mcs(p)$ to denote its MCS-process.

An application process sequentially issues read or write operations on the shared variables by sending (read or write) *calls* to its MCS-process. After sending a call, the application process blocks until it receives the corresponding *response* from its MCS-process, which ends the operation. A read call contains the variable to be read, while the response to the read call contains the value of the variable as seen by the MCS-process. A write call contains the variable to be written and the value to be written in it. The response to a write call is the explicit acknowledgment of the call by the MCS-process.

We exclude from our study systems whose MCS does not satisfy the following property.

Property 1. Consider any execution α of the system S. If $w(x)u$ is a write operation of α, and $\nexists w(x)v$ such that $w(x)u \prec w(x)v$ on α, then eventually the response to any read call on x issued by any application process will contain the value u.

With this property we assure that at least the "last" write operation on every variable must be visible in every process of the system. Note that this property is preserved by every system that we have found in the literature.

We consider MCSs implemented with *propagation* and *invalidation.* For simplicity, in both cases we consider, as in [3, 5, 8, 18, 24, 26, 25], that each MCS-process has a copy (replica) of the whole shared memory. In an MCS with invalidation, some of the copies of a variable x can be "invalid" or outdated. If an MCS-process' copy of a variable x is invalid and one of its application processes tries to read x, the MCS-process has to obtain the current value of x from some

other MCS-process (following the MCS-protocol). When an application process issues a write operation $w(x)v$, the local copy of x in its MCS-process is updated with the current value v, and the valid copies of x in the rest of MCS-processes are marked as invalid [22, 24, 26, 25]. In an MCS with propagation no copy is ever invalid and holds the current value (as seen by the MCS-process). This value is returned to an application process that issues a read operation. New written values are propagated among MCS-processes to maintain the copies up to date [3, 5, 8, 18].

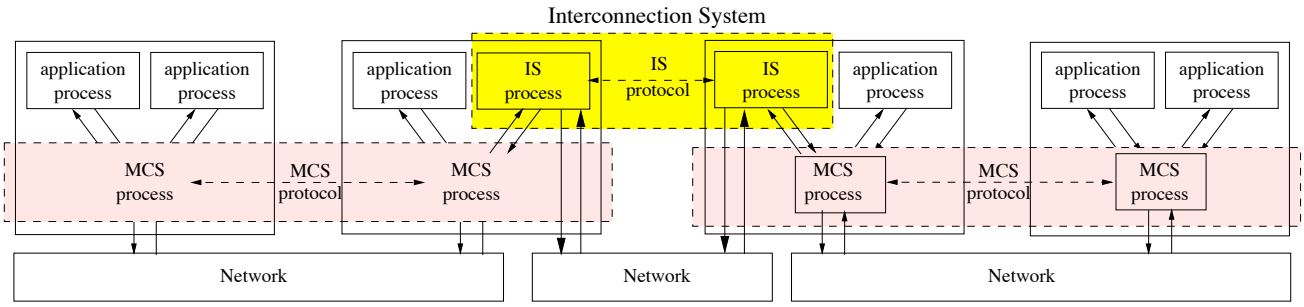


Fig. 1. Interconnection System.

2.3 The Interconnection System This paper deals with the interconnection of systems. This means that, after the interconnection, the set of original systems will behave as one single system. Using the terminology defined above, interconnecting systems is, in fact, interconnecting MCSs. In our model, the load of such an interconnection will fall on an *interconnection system* (IS). An IS is a set of processes (*IS-processes*) that execute some distributed algorithm or protocol (*IS-protocol*). For simplicity in the IS design, we consider one IS-process for each MCS to be interconnected. The IS-process of each system is at the same level as an application process and has its own MCS-process. The IS-process uses the MCS-process to read and write on the shared memory of the local system. In particular, the only way a value written by an application process in some system can be read by an application process in another system is if the IS-process of the latter system writes it. IS-processes exchange information among them (as specified by the IS-protocol) by using a communication network. Note that, after the interconnection, the overall system has a *global MCS* formed by the MCSs of the original systems plus the IS that interconnects them. Figure 1 presents an example of an IS interconnecting two systems.

For system interconnections we extend the interface between the MCS and the IS beyond read and write operations issued by IS-processes. We assume that MCS-processes are connected with its corresponding IS-process through a reliable FIFO channel and send messages to it with the changes on the local

memory replicas by using these FIFO channels. We consider the following classes of interfaces between the MCS and the IS.
(a) **Weak decoupled class with propagation (WDP).** The MCS-process of the IS-process sends a message to the IS-process every time a variable copy is updated. Each of these messages, denoted by $msg(p, x, u)$, carries the process p that issued the corresponding write operation, the variable x, and the new value u.
(b) **Strong decoupled class with propagation (SDP).** Every MCS-process in the system sends a message to its corresponding IS-process every time a variable copy is updated. Each of these messages, denoted by $msg(p, m, x, u)$, carries the application process p that issued the corresponding write operation, the MCS-process m that updated, the variable x, and the new value u. Trivially, in the SDP class, the IS-process receives at least as much information as in the WDP class. Thus, in this sense, SDP is stronger than WDP.
(c) **Strong decoupled class with invalidation (SDI).** Every MCS-process in the system sends a message to its corresponding IS-process every time a variable copy is invalidated or updated (by a write operation issued by one of its application processes). Each of these messages, denoted by $msg(p, q, x, u)$, carries the process p that issued the corresponding write operation, the MCS-process q that updated or invalidated this replica, the variable x, and the new value u (if it is an update). For each write operation $w(x)u$ issued by some process p, the IS-process will receive an update message $msg(p, mcs(p), x, u)$ from p's MCS-process, and an invalidation message $msg(p, m, x)$ from each MCS-process m that had a valid copy of x, and has invalidated it.

2.4 Model and Notation In this paper we assume an asynchronous model. This means that there is no bound on the time instructions and message transmissions take. We do not assume synchronized clocks among processes. We also assume that no system component (processes, nodes, and networks) fails.

In the rest of the paper we will use N to denote the number of systems to be interconnected. The systems to be inteconnected will be denoted by $S^0, \cdots, S^{N-1}$, and the resulting interconnected system by S^T. The IS-process for each system S^k (where $k \in \{0, \cdots, N-1\}$) is denoted by isp^k. It is worth to remark that isp^k is part of the system S^k. For that reason, the MCS-process $mcs(isp^k)$ has a local replica of each variable of the shared memory, and those replicas are updated or invalidated (depending on the method used to maintain the coherence of these replicas) following the MCS-protocol implemented in the MCS of S^k. We also assume that the IS-processes are interconnected among them through reliable FIFO communication channels which will be used to propagate write operations from one system to the other. We consider that the set of processes of S^T includes all the processes in $S^0, \cdots, S^{N-1}$ except for $isp^0, \cdots, isp^{N-1}$ (they are only used to interconnect the systems $S^0, \cdots, S^{N-1}$).

Regarding executions, we will use the next notation. α^T will denote an execution of S^T. Similarly, α^k (where $k \in \{0, \cdots, N-1\}$) will denote the execution of S^k obtained in the same run. Note that α^k and α^T have in common all the

operations issued by processes in S^k. We also extend the notation used with read and write operations. We denote by $w_p^k(x)v$ the write operation $w(x)v$ issued by process p of system S^k. Similarly, we denote by $r_p^k(x)v$ the read operation $r(x)v$ issued by process p of system S^k. A write operation $w_q^l(x)v$ in α^T issued by some processes q in S^l appears in α^k, $k \neq l$, as the write operation $w_{isp^k}^k(x)v$ issued by the process isp^k in S^k. This is so because *every* write operation issued by isp^k in α^k is, from the IS-protocol, just the propagation of a write operation issued by a process of another system S^m, $m \neq k$. We denote by $orig(op)$ the original write operation propagated as operation op in α_p^k by process isp^k. Similarly, given a write operation op issued in S^l, $l \neq k$, we denote by $prop(op)$ the write operation issued by isp^k as a result of propagating op to S^k as defined by the IS-protocol.

We will say that a consistency model can be interconnected if there is an IS-protocol that interconnects systems implementing this consistency model. The IS-protocol can specify the number of systems it interconnects. (However, it cannot restrict how applications processes are mapped to nodes.) The following observation will be useful for impossibility results.

Observation 1 *Every IS-protocol that interconnects $N > 2$ systems can be used to interconnect 2 systems.*

Proof. Let us consider that there is an IS-protocol that interconnects $N > 2$ systems through a set of N IS-processes. If we only have two systems, one of the two IS-processes can simulate $N - 2$ empty systems and their IS-processes. Then, we have an interconnected system of two systems.

3 Non-Fast Consistency Models

In this section we show that systems implementing "non-fast" consistency models can not be interconnected in any of the classes defined in the previous section. We say that a consistency model is *fast* if there is an MCS–protocol that implements it, such that memory operations only require local computations before returning control, even in systems with several nodes. There is a number of consistency models (e.g., causal or pipelined RAM) that are fast, while there are stronger memory models (e.g., the sequential or atomic) that are not. This implies that the property of being fast classifies the set of memory models in a non trivial way.

Theorem 1. *There is no IS that guarantees the interconection of systems implementing some non-fast memory model.*

Proof. We show the result by contradiction. Assume that there is a non–fast memory model M that can be interconnected. From Observation 1, we can consider the interconnnection of two systems. Therefore, let us assume there is an IS I that interconnects two systems implementing M. Let us first take a distributed system with two nodes. In each node we implement a system with one

MCS-process and at least one application process. Clearly, in these two single-node systems every memory operation only requires local computations. Now we use I to interconnect these two systems into a unique system implementing M. Then, every memory operation in the resulting system still requires only local computations, which contradicts the fact that M is not fast. This ends the proof.

As a consequence of this theorem, we can derive that a number of popular memory models can not be interconnected. In [9] it is shown that the sequential consistency model is not fast. Hence it cannot be interconnected and neither can the atomic consistency model, nor its derivations, safe and regular [21]. Similarly, Attiya and Friedman [6] have shown that the processor consistency models PCG and PCD [16, 4] are not fast and hence cannot be interconnected. Finally, in [6] Attiya and Friedman also proved that any algorithm for the mutual exclusion problem using fast operations must be cooperative. This implies that any synchronization operation that guarantees mutual exclusion must be non–fast. Therefore, any synchronized memory model that provides exclusive access cannot be interconnected. As a result, we have that memory models such as the *eager release* [15], the *lazy release* [19], the *entry* [11] or the *scope* [17] can not be interconnected.

4 pRAM Consistency Model

In this section we study the interconnection of pRAM systems. We first show that in general the interconnection is not possible. Then present IS-protocols for the different classes defined under different restrictions.

4.1 Impossibility for Interconnecting pRAM Systems In this subsection we show that we can not guarantee the interconection, through some IS in any class, of every pair of pRAM systems. The proof of the following theorem is based on the fact that, when some process p in S^k, $k \in \{0, 1\}$, issues several write operations, it may update the corresponding variables in its local memory in a different order from $p's$ process order.

Theorem 2. *There is no IS in SDP that guarantees pRAM interconection for every pair of pRAM systems.*

Proof. Let us assume, by way of contradiction, that there is a system S^T which is the result of interconnecting two pRAM systems S^0 and S^1 through some interconnection system I in the SDP class. From Definition 5, we know that for every execution α^T there is a legal view β_p^T of α_p^T, for all p, preserving $\prec_q$, for all q.

Assume that we have an execution α^0 with the following sequence of write operations issued by process p of S^0: $w_p^0(x)s \prec_p w_p^0(y)l$. We know, from Property 1, that there is a time t after which any read operation on x and y issued by any process in S^1 returns s and l, respectively. We now assume that after this time t the process p issues the write operations $w_p^0(x)u$ and $w_p^0(y)v$. We consider

that isp^0 has received the messages $msg(p, m, x, u)$ and $msg(p, m, y, v)$ in this order from each MCS-process m. Then, I can take one of the following actions:
Case 1: isp^1 issues $w^1_{isp^1}(x)u$ and $w^1_{isp^1}(y)v$, in this order, in S^1. Now, some process q of S^1 issues the read operations $r^1_q(x)u \prec_q r^1_q(y)l$. In this case, if $w^0_p(x)u$ and $w^0_p(y)v$ were issued by process p in the order: $w^0_p(y)v \prec_p w^0_p(x)u$, then it is impossible to form a legal view β^T_q preserving $\prec_p$. Hence, we reach a contradiction.
Case 2: isp^1 issues $w^1_{isp^1}(y)v$ and $w^1_{isp^1}(x)u$, in this order, in S^1. Now, some process q of S^1 issues the read operations $r^1_q(y)v \prec_q r^1_q(x)s$. In this case, if $w^0_p(x)u$ and $w^0_p(y)v$ were issued by process p in the order: $w^0_p(x)u \prec_p w^0_p(y)v$, then it is impossible to form a legal view β^T_q preserving $\prec_p$. Hence, we reach a contradiction.
Case 3: isp^1 does not issue $w^1_{isp^1}(y)v$ or $w^1_{isp^1}(x)u$ in S^1. From Property 1 this case is not possible.

We know, by definition, that SDP is stronger than WDP. Hence, we can also apply this impposibility result to ISs interconnecting pairs of pRAM systems in WDP. Note also that the above proof can be easily adapted to the SDI class. Hence, we have that there is no IS in SDI that guarantees pRAM interconection for every pair of pRAM systems.

4.2 IS-protocols for pRAM Systems In this section we show how to interconnect systems implementing the pRAM [23] consistency model in SDP, WDP, and SDI, as long as these systems satisfy certain restrictions. First, we present an IS-protocol in SDP, for MCSs that satisfy the following property, which is fulfilled by all pRAM MCSs we have found.

Property 2. In any computation α^k of system S^k ($k \in \{0, \cdots, N-1\}$), for each process p in S^k, there is an MCS-process $s(p)$, known by isp^k, such that if p issues two write operations $w^k_p(x)v \prec_p w^k_p(y)u$, then $s(p)$ updates its local replica of x with the value v before updating the variable y with the value u.

Each IS-process isp^k, $k \in \{0, \cdots, N-1\}$, contains two concurrent tasks, $Propagate^k_{out}$ and $Propagate^k_{in}$. $Propagate^k_{out}$ deals with transferring write operations issued in S^k to every S^l, $l \neq k$, while $Propagate^k_{in}$ deals with applying within S^k the write operations transfered from the systems S^l, $l \neq k$. The two tasks that form the pRAM IS-protocol in SDP operate as follows (see Fig. 2):

- Task $Propagate^k_{out}$ is activated after a message $msg(p, s(p), x, v)$, for some process p, is received by isp^k. Then, $Propagate^k_{out}$ sends the pair $\langle x, v\rangle$ to every isp^l, $l \neq k$. From the above Property 2, the sending of the pairs generated from the write operations issued by p follows p's process order. We avoid to re-propagate write operations received from other systems, by checking that the write operation was not issued in S^k by isp^k.
- Task $Propagate^k_{in}$ is activated whenever a pair $\langle x, v\rangle$ is received from some process isp^l, $l \neq k$. As a result, it performs a write operation $w^k_{isp^k}(x)v$, thus propagating the value v to all the replicas of variable x within S^k.

```
1  Task Propagate^k_out :: upon reception
     of msg(p, s(p), x, v)
2  begin
3     if p ≠ isp^k then
4        send ⟨x, v⟩ to every isp^l, l ≠ k
5  end
```

```
1  Task Propagate^k_in :: upon reception
     of ⟨x, v⟩ from isp^l, l ≠ k
2  begin
3     w^k_{isp^k}(x)v
4  end
```

Fig. 2. The pRAM IS-protocol in isp^k in SDP.

The correctness of the IS-protocol of Fig. 2 is placed in the Appendix due to space limitation. There, we show that the system S^T, obtained by connecting N systems $S^0, \cdots, S^{N-1}$ using this pRAM IS-protocol in SDP, is pRAM.

We now consider an IS-protocol in WDP such that this IS only interconnects MCSs that fulfill the following Property 3.

Property 3. In any computation α^k of system S^k ($k \in \{0, \cdots, N-1\}$), for each process p in S^k, if p issues two write operations $w_p^k(x)v \prec_p w_p^k(y)u$, then $mcs(isp^k)$ updates its local replica of x with the value v before updating its local replica of y with the value u.

We can observe that this Property 3 is a particular case of Property 2 where process $s(p)$ is now $mcs(isp^k)$. Hence, we can use the same IS-protocol of Figure 2.

Finally, to end this section, we consider an IS-protocol in SDI such that this IS only interconnects MCSs that fulfill the following Property 4.

Property 4. In any computation α^k of system S^k ($k \in \{0, \cdots, N-1\}$), for each process p in S^k, if p issues two write operations $w_p^k(x)v \prec_p w_p^k(y)u$, then $mcs(p)$ updates its local replica of x with the value v before updating its local replica of y with the value u.

Again, this property is a particular case of Property 2 where process $s(p)$ is now $mcs(p)$. Hence, we can use the same IS-protocol of Figure 2 to interconnect pRAM systems in SDI.

5 Causal Consistency Model

In this section we study the interconnection of causal systems. Note that the pRAM model is strictly weaker than causal model [5, 12]. Therefore, the results of impossibility of Section 4 are also applicable to causal systems.

In Section 4 we present an IS–protocol in SDP for interconnecting pRAM systems satisfying Property 2. We can show that there is no IS in SDP that interconnects every pair of causal systems satisfying Property 2. (The proof is placed in the appendix due to space limitations.) This result can be easily extended to WDP with Property 3 and SDI with Property 4.

We now propose an IS–protocol in SDP for causal systems. We also show in this subsection that the resulting system of this interconnection is causal. For our IS, we will only consider MCSs that fulfill the following Property 5 (which is in fact satisfied by all the causal protocols we have found).

Property 5. Consider any execution α^k of the causal system S^k (where $k \in \{0, \cdots, N-1\}$). For each two write operations $w_p^k(x)v \prec w_q^k(y)u$ on α^k, each MCS–process of system S^k updates its local replica of x with the value v before updating its local replica of y with the value u.

In the Figure 3 we present the causal IS-protocol we propose. This protocol requires that the communication among IS-processes is totally ordered. There are well-known message-passing protocols (e.g., [7, pp. 177-179]) to provide total ordering of messages.

```
1  Task Propagate^k_out :: upon reception
     of msg(p, q, x, v), from every MCS-process q
2  begin
3     if p ≠ isp^k then
4        send ⟨x, v⟩ to every isp^l, l ≠ k
5  end
```

```
1  Task Propagate^k_in :: upon reception
     of ⟨x, v⟩ from isp^l, l ≠ k
2  begin
3     w^k_{isp^k}(x)v
4  end
```

Fig. 3. The causal IS-protocol in isp^k in SDP.

It can be observed that the IS-protocol is composed by two task, like the IS-protocol of Fig. 2. In fact, the $Propagate_{in}^k$ task is the same. However, the key difference is found in task $Propagate_{out}$. In this task a pair $\langle x, v\rangle$ is not sent to the other systems until all the MCS replicas of x have been updated. Note that, from Property 5, write operations are propagated to the rest of systems following the causal order in system S^k. We need the communication among IS-processes to be totally ordered to enforce that two write operations from different systems and casually ordered, are applied in the rest of systems in the causal order.

Let us now show that the system S^T, obtained by connecting N systems $S^0, \cdots, S^{N-1}$ using the causal IS–protocol of Fig. 3 in SDP, is causal.

Let p be some process in system S^k, $k \in \{0, \cdots, N-1\}$, and let $mcs(p)$ be its MCS-process. Recall that α_p^k (resp. α_p^T) is the set obtained by removing from α^k (resp. α^T) all read operations except those from process p. We define β_p^k as a sequence with the same operations as α_p^k that preserves the order in which all operations of α_p^k are issued by process p, and the order in which every write operation is applied in $mcs(p)$. Formally,

Definition 6. *Let β_p^k a sequence of the operations in α_p^k. Let op and op' in α_p^k. Then $op \rightarrow op'$ in β_p^k, if any of the following happens:*
1. op and op' are operations from the same process p of S^k and $op \prec_p op'$.

2. $op = w_q^k(x)u$, $op' = w_s^k(y)v$, *and in* $mcs(p)$ *the local copy of* x *is updated with* u *before updating* y *with* v.
3. $op = w_q^k(x)u$, $op' = r_p^k(y)v$, *and in* $mcs(p)$ *the local copy of* x *is updated with* u *before* p *issues* op'.

Note that, like in α_p^k, every write operation of process isp^k in β_p^k is the propagation of a write operation issued by a process of S^l, $l \neq k$. We define β_p^T as the sequence obtained by replacing in β_p^k every write operation op from isp^k by the write operation $orig(op)$.

The proofs of the following two lemmas are omitted due to space limitations.

Lemma 1. β_p^T *is formed by all operations of* α_p^T.

Lemma 2. β_p^T *preserves the execution order* $\prec$ *on* α^T.

Lemma 3. β_p^T *is legal.*

Proof. If process p issues some read operation $op = r_p^k(x)u$ is because it has, when this operation is invoked, in its local copy of x the value u. Then, the latest write operation applied on x in p is $op' = w^k(x)u$. Hence, from the third condition of Definition 6, op' must be the previous nearest write operation on x in β_p^k. Therefore, from Definition 4, β_p^k is legal. Note that, by definition of β_p^T, if we replace in β_p^k every write operation op from isp^k by the write operation $orig(op)$, we obtain β_p^T. Then, β_p^T is legal.

Theorem 3. *The system* S^T *is causal.*

Proof. From Definition 5, S^T is causal if in each execution α^T there is a legal view of α_p^T, for all p, that preserves the execution order $\prec$ on α^T. From Lemma 1, β_p^T is formed by all operations of α_p^T. Also, from Lemma 2, β_p^T preserves the execution order $\prec$ on α^T. Finally, from Lemma 3, β_p^T is legal. Then, β_p^T is a legal view of α_p^T that preserves the execution order $\prec$ on α^T. Hence, S^T is a causal system.

6 Cache Consistency Model

In this section we study the interconnection of cache systems. We show that, unlike the previous models, the interconnection of cache systems is always possible, independently of how they are implemented. The interconnection only uses read and write operations, without any other consideration about the interface between the MCS and the IS. Hence, we can use the same IS-protocol for SDP, SDI and WDP classes.

The IS-protocol we propose only works for the interconnection of two systems. However, it can be repeatedly used to interconnect as many systems as desired. Each isp^k (in this case $k \in \{0, 1\}$) has one task for each variable of the shared memory, presented in Figure 4.

```
1   Task Propagate^k(x) :: upon reception of ⟨x, v⟩ from isp^{1−k}
2   begin
3         if v ≠ "NoData" then
4               w^k_{isp^k}(x)v
5               last(x) = v
6         r^k_{isp^k}(x)u
7         if u = last(x) then
8               u = "NoData"
9         send ⟨x, u⟩ to isp^{1−k}
10  end
```

Fig. 4. The cache IS-protocol in isp^k for variable x.

Note that each IS-process maintains a copy of the latest value propagated from the other system in $last(x)$ for each variable x. That copy must be initialized with a special value (e.g., "$NoData$"). Note also that initially one of the IS-processes (for instance isp^0) must send to the other a message with $\langle x, NoData \rangle$ for each variable x to start the interconnection.

The proof that this cache IS-protocol interconnects two cache systems is placed in the Appendix due to space limitations.

7 Conclusions

In this paper we have formalized and studied the interconnection of distributed shared memory systems. We have shown that non-fast, pRAM, and causal systems cannot be interconected in general, while cache systems can. Then, we have given sufficient conditions to interconnect pRAM and causal systems. Figure 5 sumarizes these results.

Memory model	SDP	SDI	WDP
Non-fast	No	No	No
Causal	Yes (Property 5)	No	No
pRAM	Yes (Property 2)	Yes (Property 4)	Yes (Property 3)
Cache	Yes	Yes	Yes

Fig. 5. Possibilities of interconnection under the different classes considered in this work.

References

1. N. Adly and M. Nagi. Maintaining causal order in large scale distributed systems using a logical hierarchy. In *Proceedings of the 12th IASTED International Conference on Applied Informatics*, 1995.

2. S.V. Adve. *Designing Memory Consistency Models for Shared-Memory Multiprocessors*. PhD thesis, University of Wisconsin-Madison, 1993.
3. Yehuda Afek, Geoffrey Brown, and Michael Merritt. Lazy caching. *ACM Transactions on Programming Languages and Systems*, 15(1):182–205, January 1993.
4. M. Ahamad, R. Bazzi, R. John, P. Kohli, and G. Neiger. The power of processor consistency. In *Proceedings of the 5th ACM Symposium on Parallel Algorithms and Architectures*, pages 251–260, 1993.
5. M. Ahamad, G. Neiger, J.E. Burns, P. Kohli, and P.W. Hutto. Causal memory: Definitions, implementation and programming. *Distributed Computing*, 9(1):37–49, August 1995.
6. H. Attiya and R. Friedman. Limitations of fast consistency conditions for distributed shared memories. *Information Processing Letters*, 57:243–248, 1996.
7. H. Attiya and J. Welch. *Distributed Computing: Fundamentals, Simulations, and Advanced Topics*. McGraw-Hill, 1998.
8. H. Attiya and J.L. Welch. Sequential consistency versus linearizability. Technical Report 674, Department of Computer Science, The Technion, October 1991.
9. H. Attiya and J.L. Welch. Sequential consistency versus linearizability. *ACM Transactions on Computer Systems*, 12(2):91–122, 1994.
10. R. Baldoni, R. Beraldi, R. Friedman, and R. van Renesse. The hierarchical daisy architecture for causal delivery. *Distributed Systems Engineering Journal*, 6, 1999.
11. B.N. Bershad, M.J. Zekauskas, and W.A. Sawdon. The Midway distributed shared memory system. In *COMPCON*, 1993.
12. V. Cholvi. *Formalizing Memory Models*. PhD thesis, Department of Computer Science, Polytechnic University of Valencia, December 1994.
13. V. Cholvi. Specification of the behavior of memory operations in distributed systems. *Parallel Processing Letters*, 8(4):589–598, December 1998.
14. A. Fernández, E. Jiménez, and V. Cholvi. On the interconnection of causal memory systems. In *Proceedings of the 19th Annual ACM Symposium on Principles of Distributed Computing*. ACM, July 2000.
15. K. Gharachorloo, D. Lenoski, J. Laudon, P. Gibbons, A. Gupta, and J. Hennessy. Memory consistency and event ordering in scalable shared-memory multiprocessors. In *Proceedings of the 17th Annual International Symposium on Computer Architecture*, pages 15–26. ACM, May 1990.
16. J.R. Goodman. Cache consistency and sequential consistency. Technical Report 61, IEEE Scalable Coherence Interface Working Group, March 1989.
17. L. Iftode, J. Singh, and K. Li. Scope consistency: A bridge between release consistency and entry consistency. In *Proc. of the 8th Annual ACM Symposium on Parallel Algorithms and Architectures*, 1996.
18. Ernesto Jiménez, Antonio Fernández, and Vicente Cholvi. A parametrized algorithm that implements sequential, causal, and cache memory consistency. In *Proceedings of the 10th Euromicro Workshop on Parallel, Distributed and Network-Based Processing (Euro PDP 2002)*, Canary Islands, Spain, January 2002. IEEE Computer Society Press.
19. P. Keleher. *Distributed Shared Memory Using Lazy Consistency*. PhD thesis, Rice University, 1994.
20. L. Lamport. How to make a multiprocessor computer that correctly executes multiprocess programs. *IEEE Transactions on Computers*, 28(9):690–691, September 1979.
21. L. Lamport. On interprocess communication: Parts I and II. *Distributed Computing*, 1(2):77–101, 1986.

22. K. Li and P. Hudak. Memory coherence in shared memory systems. *ACM Transactions on Computer Systems*, 7(4):321–359, November 1989.
23. R.J. Lipton and J.S. Sandberg. PRAM: A scalable shared memory. Technical Report CS-TR-180-88, Princeton University, Department of Computer Science, September 1988.
24. M. Mizuno, M. Raynal, and J.Z. Zhou. Sequential consistency in distributed systems. In *Proceedings of the International Workshop on Theory and Practice of Distributed Computing*, pages 224–241. Castle(Germany), Springer Verlag LNCS 938, 1994.
25. M. Raynal. Sequential consistency as lazy linearizability. In *Proceedings of the 14th Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 151–152. ACM, August 2002.
26. M. Raynal. Token-based sequential consistency. In *Proceedings of the IEEE International Conference on Advanced Information Networks and Applications (AINA'03)*. , March 2003.
27. L. Rodrigues and P. Verissimo. Causal separators and topological timestamping: an approach to support causal multicast in large-scale systems. In *Proceedings of the 15th International Conference on Distributed Systems*, May 1995.

8 Appendix

8.1 Correctness Proof of the pRAM IS-protocol of Figure 2

Let p be some process in S^k, $k \in \{0, \cdots, N-1\}$, and β_p^k be a legal view of α_p^k preserving $\prec_q$ on α_p^k, for all q in S^k, as described in Definition 4. From Definition 5, such a legal view must exist by the fact that S^k is a pRAM system. We define β_p^T as the sequence obtained by replacing in β_p^k every write operation op from isp^k by the write operation $orig(op)$.

Lemma 4. *β_p^T is formed by all operations of α_p^T.*

Proof. First of all, note that the difference between α_p^k and α_p^T is that, for each operation op issued by isp^k in α_p^k, α_p^T contains the original operation $orig(op)$. Since β_p^k is a sequence formed by all operations of α_p^k, and β_p^T is obtained by replacing in β_p^k every write operation op from isp^k by the write operation $orig(op)$, then the set of operations in β_p^T is the same as that of α_p^T.

The following lemmas shows that of β_p^T preserves the order in which the operations are issued in any process of S^T.

Lemma 5. *Let $op = w_q^k(x)v$ and $op' = w_q^k(y)u$ be two operations of α^T issued by the same process p of S^k. If $op \prec_q op'$ on α^k, then $Propagate_{out}^k$ will send to every S^l, $l \neq k$, $\langle x, v\rangle$ before $\langle y, u\rangle$.*

Proof. Directly since, by Property 2, isp^k receives in S^k the message $msg(q, s(q), x, v)$ before $msg(q, s(q), y, u)$, and then $Propagate_{out}^k$ sends to isp^{1-k} the pair $\langle x, v\rangle$ before $\langle y, u\rangle$.

Lemma 6. *Let op and op' be two write operations of α^T issued by the same process q of S^l, where $l \neq k$. If $op \prec_q op'$ on α^l, then $prop(op) \rightarrow prop(op')$ in β_p^k, for all p.*

Proof. We know that β_p^k is a legal view that preserves the q's process order $\prec_q$ on α^k, for all q. Then, the result follows from Lemma 5, the fact that the channel connecting isp^{1-k} to isp^k is reliable and FIFO, and the implementation of task $Propagate_{in}^k$ (see Fig. 2).

Lemma 7. *β_p^T preserves q's process order $\prec_q$ on α_p^T, for all q.*

Proof. By way of contradiction, let us assume that β_p^T does not preserve the order among operations issued by a process q of S^T. Hence, there must be at least two operations op and op' of α_p^T issued by q such that $op \prec_q op'$ but op' precedes op in β_p^T. Let us consider two possible cases.
Case 1: q is in S^k. Since op' precedes op in β_p^T, op' also precedes op in β_p^k, by definition of β_p^T. Then, β_p^k does not preserve q's process order $\prec_q$. However, this

is not possible since, by definition, β_p^k is a legal view preserving $\prec_q$, for all q. Hence, we reach a contradiction.
Case 2: q is in S^l, $l \neq k$. Since both operations are in β_p^T, which only contains read operations from process p of system S^k, both must be write operations. Let op and op' be propagated as operations $prop(op)$ and $prop(op')$, respectively, issued by process isp^k. From Lemma 6, we have that $prop(op) \rightarrow prop(op')$ in β_p^k. Observe now that, by definition, operation $prop(op)$ in β_p^k is replaced by op and operation $prop(op')$ is replaced by op' to obtain β_p^T. Then op precedes op' in β_p^T, and we reach a contradiction.

Lemma 8. *β_p^T is legal.*

Proof. By definition, β_p^k is legal. Also by definition, β_p^T is obtained by replacing in β_p^k every write operation op from isp^k by the write operation $orig(op)$. Therefore, β_p^T is legal.

Theorem 4. *The system S^T is pRAM.*

Proof. From Definition 5, S^T is pRAM if in each execution α^T there is a legal view of α_p^T, for all p, that preserves q's process order $\prec_q$, for all q. From Lemma 4, β_p^T is formed by all operations of α_p^T. Also, from Lemma 7, β_p^T preserves the process order $\prec_q$, for all q. Finally, from Lemma 8, β_p^T is legal. Then, β_p^T is a legal view of α_p^T that preserves q's process order $\prec_q$, for all q. Hence, S^T is a pRAM system.

8.2 Impossibility of causal interconnection in SDP with Property 2

Theorem 5. *There is no IS-protocol in SDP that guarantees causal interconnection for every pair of causal systems even if Property 2 holds.*

Proof. Let us assume, by way of contradiction, that there is a system S^T which is the result of interconnecting two causal systems S^0 and S^1 that satisfy Property 2 in SDP with the IS I. From Definition 5, we know that for every execution α^T there is a legal view β_p^T of α_p^T, for all p, preserving $\prec$.

Assume that we have an execution α^0 with the following write operations issued by process r of S^0: $w_r^0(x)s \prec w_r^0(y)l$. We know, from Property 1, that there is a time such that any read operation on x and y issued by any process in S^1 returns s and l, respectively. We now assume that after this time the processes p and t issue the write operations $w_p^0(x)u$ and $w_t^0(y)v$, causally related between them through read operations (detailed below in each case). We consider that isp^0 has received the messages $msg(p, m, x, u)$ and $msg(t, m, y, v)$ in this order from each MCS-process m. Then, I can take one of the following actions:
Case 1: isp^1 issues $w_{isp^1}^1(x)u$ and $w_{isp^1}^1(y)v$, in this order, in S^1. Now, some process q of S^1 issues the following read operations $r_q^1(x)u \prec_q r_q^1(y)l$. In this case, if $r_t^0(y)l \prec w_t^0(y)v \prec r_p^0(y)v \prec w_p^0(x)u$, then it is impossible to form a legal view β_q^T preserving $\prec$. Hence, we reach a contradiction.

Case 2: isp^1 issues $w^1_{isp^1}(y)v$ and $w^1_{isp^1}(x)u$, in this order, in S^1. Now, some process q of S^1 issues the following read operations $r^1_q(y)v \prec_q r^1_q(x)s$. In this case, if $r^0_p(x)s \prec w^0_p(x)u \prec r^0_t(x)u \prec w^0_t(y)v$, then it is impossible to form a legal view β^T_q preserving $\prec$. Hence, we reach a contradiction.
Case 3: isp^1 does not issue $w^1_{isp^1}(y)v$ or $w^1_{isp^1}(x)u$ in S^1. From Property 1 this case is not possible.

8.3 Auxiliary Lemmas for the Correctness Proof of the Causal IS-protocol of Figure 3

Definition 7 (Non-Transitive Execution Order). *Let op and op' be two operations in a execution α. Then op precedes op' in the non-transitive execution order ($op \prec_{nt} op'$) on α if some of the following holds:*
1. op and op' are operations from the same process p and $op \prec_p op'$ on α.
2. $op = w(x)v$ and $op' = r(x)v$.

Definition 8 ($\prec$–Related Sequence). *Let op and op' be two operations in a execution α such that $op \prec op'$ on α. A* $\prec$–related sequence *between op and op' is a sequence of operations $op^1, op^2, \ldots, op^m$ belonging to α such that $op^1 = op$, $op^m = op'$, and $op^i \prec_{nt} op^{i+1}$ on α, for $1 \leq i < m$.*

Note that at least one $\prec$–related sequence always exists between op and op' if $op \prec op'$ on α.

When considering the composed system S^T, a $\prec$–related sequence Seq between operations op and op' of execution α^T can be divided in n subsequences $subSeq_1$, $subSeq_2$, $\ldots, subSeq_n$, such that all the operations in subsequence $subSeq_i$ belong to the same system S^k and the operations in consecutive subsequences belong to different systems. We use $subSeq^k_i$ to express that all the operations of the ith subsequence belong to system S^k.

We use $first(subSeq^k_i)$ and $last(subSeq^k_i)$ to denote the first and last operation of the subsequence $subSeq^k_i$, respectively. Note that, in two consecutive subsequences $subSeq^k_i$ and $subSeq^{1-k}_{i+1}$ of a given sequence, $last(subSeq^k_i) = w^k_j(x)v$ and $first(subSeq^{1-k}_{i+1}) = r^{1-k}_l(x)v$, i.e. the first operation of the later subsequence reads the value written by the last operation of the former subsequence.

Lemma 9. *Let op and op' be two operations in α^T_p issued in system S^k such that $op \prec op'$ in α^T. If there is a $\prec$–related sequence between op and op' with one single subsequence $subSeq^k_1$, then $op \rightarrow op'$ in β^k_p.*

Proof. Let us assume, by way of contradiction, that the claim does not hold. Then, $op \prec op'$ on α^k, and $op' \rightarrow op$ in β^k_p. This is only possible if there are at least two "consecutive" operations op^i and op^{i+n} in $subSeq^k_1$ and belonging to α^k_t such that $op^{i+n} \rightarrow op^i$ in β^k_p. We say op^{i+n} and op^i are two consecutive operations in $subSeq^k_1$ if they are in α^k_t, $t \neq p$, and between them there is not any other operation belonging to α^k_p (i.e., every operation op^{i+l}, $1 \leq l < n$, is a read operation issued by a process other than p). Note that if $n > 1$ then these

two consecutive operations op^i and op^{i+n} only can be write operations. We have three cases:

Case 1: $op^i = w^k(x)v$ and $op^{i+n} = w^k(y)u$. From definition of $\prec$–related sequence, $op^i \prec op^{i+n}$ on α^k. From Property 5, if $op^i \prec op^{i+n}$ on α^k, then op^i must be applied in all processes of S^k (and, of course, in p) before op^{i+n}. Therefore, from the second condition of Definition 6, $op^i \rightarrow op^{i+n}$ in β_p^k, and we reach a contradiction.

Case 2: $op^i = w^k(x)v$ and $op^{i+1} = r_p^k(x)v$. From definition of $\prec$–related sequence, $op^i \prec_{nt} op^{i+1}$ on α^k. Obviously, the write operation $w^k(x)v$ must be applied before issuing $r_p^k(x)v$, since, otherwise, op^{i+1} could not obtain the value v in x. Therefore, from the third condition of Definition 6, $op^i \rightarrow op^{i+1}$ in β_p^k, and we reach a contradiction.

Case 3: op^i and op^{i+1} are issued by the same process p. From definition of $\prec$–related sequence, $op^i \prec_{nt} op^{i+1}$ on α^k, and, from case 1 of $\prec_{nt}$, $op^i \prec_p op^{i+1}$. Then, from the first condition of Definition 6, $op^i \rightarrow op^{i+1}$ in β_p^k, and we reach a contradiction.

Lemma 10. *Let op and op' be two operations in α_p^T.*

1. *If they are issued by system S^k and $op \prec op'$ on α^T, then $op \rightarrow op'$ in β_p^k.*
2. *If they are issued by system S^k and $op = w^k(x)v$ and $op' = w^k(y)u$ and $op \prec op'$ on α^T, then*
   $Propagate_{out}^k$ will send the pairs $\langle x, v\rangle$ and $\langle y, u\rangle$ to S^{1-k} in this order.
3. *If they are issued by system S^l, $l \neq k$, and are write operations and $op \prec op'$ on α^T, then*
   $prop(op) \rightarrow prop(op')$ in β_p^k.
4. *If they are issued respectively by systems S^{1-k} and S^k and that $op = w^{1-k}(x)v \prec op'$ on α^T, then*
   $prop(op) \rightarrow op'$ in β_p^k.
5. *If they are issued respectively by systems S^k and S^{1-k}, and $op \prec op' = w^{1-k}(x)v$ on α^T, then*
   $op \rightarrow prop(op')$ in β_p^k.
6. *If they are issued respectively by systems S^l and S^m (where $l \neq m$, $l \neq k$, and $m \neq k$) and are write operations and $op \prec op'$ on α^T, then $prop(op) \rightarrow prop(op')$ in β_p^k.*

Proof. Proof of Part 1: Let Seq be a $\prec$–related sequence between op and op'. We use induction on the number of subsequences of Seq to show the result. Note that this number has to be odd. In the base case, the sequence Seq has only one subsequence $subSeq_1^k$. Hence, from Lemma 9, $op = first(subSeq_1^k) \rightarrow op' = last(subSeq_1^k)$ in β_p^k. Assume the claim is true for sequences with i subsequences. We show it also holds if Seq has $i+2$ subsequences. By induction hypothesis, we have that $op = first(subSeq_1^k) \rightarrow last(subSeq_i^k)$ in β_p^k. Note

that $last(subSeq_i^k) = w_t^k(x)v$ is propagated to every system S^l, $l \neq k$, by process isp^k after, in all processes of S^k, the local copy of x is updated with the value v. Later on, isp^l propagates the pair (y, u) from $last(subSeq_{i+1}^l) = w_q^l(y)u$ as $w_{isp^k}^k(y)u$ (see Fig. 3). Then, $w_t^k(x)v$ is applied by all processes in S^k (and, of course, by p) before $w_{isp^k}^k(y)u$, and therefore, from the second condition of β_p^k in Definition 6, $w_t^k(x)v \rightarrow w_{isp^k}^k(y)u$ in β_p^k. From the second condition of $\prec$–related order, $w_{isp}^k(y)u \prec_{nt} first(subSeq_{i+2}^k) = r_s^k(y)u$ on α^k, and then, $w_{isp}^k(y)u \prec op' = last(subSeq_{i+2}^k)$ on α^k. Then, from Lemma 9, $w_{isp^k}^k(y)u \rightarrow op' = last(subSeq_{i+2}^k)$ in β_p^k. Hence, by transitivity, $op = first(subSeq_1^k) \rightarrow op' = last(subSeq_{i+2}^k)$ in β_p^k.

Proof of Part 2: If there is a $\prec$–related sequence between op and op' with a single subsequence, then $op \prec op'$ on α^k, and it follows from Property 5 that op is applied in all processes of S^k before op'. Otherwise, the proof of Part 1 shows the same fact when the $\prec$–related sequences between op and op' have more than one subsequence. Then, since the task $Propagate_{out}^k$ of our IS–protocol (see Fig.3) propagates operations in the order they are locally applied, it will send to S^{1-k} the pair $\langle x, v\rangle$ of op before the pair $\langle y, u\rangle$ of op'.

Proof of Part 3: From Part 1, $op \rightarrow op'$ in β_q^l, $l \neq k$. Then, the result follows from Part 2, the fact that the channel connecting isp^l to isp^k is reliable and FIFO, and the implementation of task $Propagate_{in}^k$ (see Fig. 3). The process isp^k issues $prop(op)$ and $prop(op')$ in S^k and then, from the first condition of execution order, $prop(op) \prec prop(op')$ on α^k. Hence, from Lemma 9, $prop(op) \rightarrow prop(op')$ in β_p^k.

Proof of Part 4: Let Seq be a $\prec$–related sequence with n subsequences between op and op'. Let us assume $last(subSeq_{n-1}^l) = w_q^l(y)u$ and $first(subSeq_n^k) = r_s^k(y)u$. From Part 3, $prop(op) \rightarrow prop(last(subSeq_{n-1}^l)) = prop(w_q^l(y)u) = w_{isp^k}^k(y)u$ in β_p^k. From second condition of $\prec$–related order, $w_{isp}^k(y)u \prec_{nt} first(subSeq_n^k) = r_s^k(y)u$ on α^k, and then, $w_{isp}^k(y)u \prec op' = last(subSeq_n^k)$ on α^k. We know, from Property 5 and Definition 6 of β_p^k, that $w_{isp^k}^k(y)u \rightarrow op' = last(subSeq_n^k)$ in β_p^k. Hence, by transitivity, $op = prop(op) \rightarrow op' = last(subSeq_n^k)$ in β_p^k.

Proof of Part 5: Similar to the proof of Part 4.

Proof of Part 6: Let Seq be a $\prec$–related sequence with n subsequences between op and op'. Let us assume $last(subSeq_{n-1}^l) = w_q^l(y)u$, $first(subSeq_n^m) = r_s^m(y)u$ and $op' = last(subSeq_n^m) = w_t^m(x)v$. From Part 3, $prop(op) \rightarrow prop(last(subSeq_{n-1}^l)) = prop(w_q^l(y)u) = w_{isp^k}^k(y)u$ in β_p^k. Note that $last(subSeq_{n-1}^l) = w_q^l(y)u$ is propagated, through a reliable totally ordered FIFO network, to every system different from S^l (hence, included S^m and S^k) by process isp^l after, in all processes of S^l, the local copy of x is updated with the value v. Later on, isp^m issues $w_{isp^m}^m(y)u$. By definition of $\prec$, and from definition of $\prec$–related sequence, we know that $w_{isp^m}^m(y)u \prec first(subSeq_n^m) = r_s^m(y)u \prec op'$. Then, isp^m issues $prop(last(subSeq_{n-1}^l)) = w_{isp^m}^m(y)u$ before $op' = w_t^m(x)v$ in S^m. Hence, the process isp^k (as the network connecting isp^l,

isp^m, and isp^k is reliable, totally ordered, FIFO, and seeing the implementation of task $Propagate^k_{in}$ of Fig. 3) issues $prop(last(subSeq^l_{n-1})) = w^k_{isp^k}(y)u$ before $prop(op') = prop(last(subSeq^m_n)) = w^k_{isp^k}(x)v$ in S^k. Therefore, from Property 5 and Definition 6 of β^k_p, $prop(last(subSeq^l_{n-1})) = w^k_{isp^k}(y)u \rightarrow prop(op') = prop(last(subSeq^m_n)) = w^k_{isp^k}(x)v$ in β^k_p. Thus, by transitivity, $prop(op) \rightarrow prop(op')$ in β^k_p.

Lemma 2 β^T_p preserves the execution order $\prec$ on α^T.

Proof. We show in this proof that if there are two operations op and op' in α^T_p such that $op \prec op'$ on α^T, then $op \rightarrow op'$ in β^T_p.

Let us make a case analysis:

1. Case op and op' are issued by processes in S^k: From Part 1 of Lemma 10 (see the Appendix), if $op \prec op'$ on α^T, then $op \rightarrow op'$ in β^k_p. Then, by definition of β^T_p, we have that $op \rightarrow op'$ in β^T_p.
2. Case op and op' are issued by processes in S^l, where $l \neq k$: Since both operations are in α^T_p, which only contains read operations from process p of system S^k, both operations must be write operations. Then, let op and op' be propagated as operations $prop(op)$ and $prop(op')$.
   From Part 3 of Lemma 10, we have that if $op \prec op'$ on α^T, then $prop(op) \rightarrow prop(op')$ in β^k_p. Hence, replacing $prop(op)$ and $prop(op')$ by op and op', respectively, we have that, by definition of β^T_p, $op \rightarrow op'$ in β^T_p.
3. Case op is issued by some process in S^l and op' is issued by some process in S^k, where $l \neq k$: op must be a write operation, since α^T_p only contains read operations from process p of system S^k. Such operation will be propagated from S^l to S^k as described by the IS–protocol and it will appear in S^k as a (write) operation $prop(op)$ issued by process isp^k.
   From Part 4 of Lemma 10, if $op \prec op'$ on α^T, then $prop(op) \rightarrow op'$ in β^k_p. Hence, replacing $prop(op)$ by op, we have that, by definition of β^T_p, $op \rightarrow op'$ in β^T_p.
4. Case op is issued by some process in S^k and op' is issued by some process in S^l, where $l \neq k$: op' must be a write operation, since α^T_p only contains read operations from process p of system S^k. Such operation will be propagated from S^l to S^k as described by the IS–protocol and it will appear in S^k as a (write) operation $prop(op')$ issued by process isp^k.
   From Part 5 of Lemma 10, if $op \prec op'$ on α^T, then $op \rightarrow prop(op')$ in β^k_p. Hence, replacing $prop(op')$ by op', we have that, by definition of β^T_p, $op \rightarrow op'$ in β^T_p.
5. Case op is issued by some process in S^l, where $l \neq k$, and op' is issued by some process in S^m, where $m \neq k$ and $m \neq l$: Since both operations are in α^T_p, which only contains read operations from process p of system S^k, both operations must be write operations. Then, let op and op' be propagated as operations $prop(op)$ and $prop(op')$.

From Part 6 of Lemma 10, we have that if $op \prec op'$ on α^T, then $prop(op) \rightarrow prop(op')$ in β_p^k. Hence, replacing $prop(op)$ and $prop(op')$ by op and op', respectively, we have that, by definition of β_p^T, $op \rightarrow op'$ in β_p^T.

8.4 Correctness Proof of the Cache IS-protocol of Figure 4

We now show that the system S^T, obtained by connecting two systems S^0 and S^1 using the cache IS–protocol in any class, is a cache system. This proof does not depend on whether the MCSs use propagation or invalidation to preserve the cache consistency of the copies of each object of the shared memory. This is so because the work of the IS–protocol of Fig. 4 is not based in the message reception.

Let $\beta(x)^k$ be a legal view of $\alpha(x)^k$ preserving $\prec$ on $\alpha(x)^k$, as described in Definition 4. Such a legal view must exist by the fact that S^k is a cache system. We define op_i as the i^{th} write operation propagated by process isp from one system to the other (independently of in which system it is issued). We use $op(x)_i^k$ to indicate that op_i is issued by some process in S^k on variable x. We use $prop^{1-k}(op(x)_i^k)$ to denote the write operation issued by the task $Propagate_i^{1-k}$ as a result of the propagation of $op(x)_i^k$.

We define $\beta(x)_i^k$ as the subsequence of operations of $\beta(x)^k$ issued by processes of S^k from $op(x)_i^k$ (or $prop^k(op(x)_i^{1-k})$) until $op(x)_{i+1}^k$ (or $prop^k(op(x)_{i+1}^{1-k})$) without including them.

We define $\beta(x)_i^T$ as the sequence formed by all operations issued by processes of S^T between the ith and $ith+1$ propagation of write operations on variable x so that operations belonging to $\alpha(x)^k$ follow the order they have in $\beta(x)^k$, and operations belonging to $\alpha(x)^{1-k}$ follow the order they have in $\beta(x)^{1-k}$. Formally, $\beta(x)_i^T$ can be obtained as follows: $op(x)_i \cdot head(x)_i^k \cdot head(x)_i^{1-k} \cdot tail(x)_i^k \cdot tail(x)_i^{1-k}$, where $head(x)_i^k$ denotes the subsequence of $\beta(x)_i^k$ that includes all read operations from the beginning of $\beta(x)_i^k$ until the first write operation in $\beta(x)_i^k$ (not included), and $tail(x)_i^k$ is the subsequence of $\beta(x)_i^k$ that includes all the operations in $\beta(x)_i^k$ that are not in $head(x)_i^k$.

Finally, we define $\beta(x)^T$ as the sequence obtained by concatenating the sequences of $\beta(x)_i^T$ such that $\beta(x)_i^T$ goes before $\beta(x)_{i+1}^T, \forall i$. In what follows, we will prove that it is a legal view of $\alpha(x)^T$.

Lemma 11. *$\beta(x)^T$ is a sequence formed by all operations of $\alpha(x)^T$.*

Proof. $\alpha(x)^T$ is, by definition, the set of all operations in $\alpha(x)^k$ and $\alpha(x)^{1-k}$ issued by all processes of S^k and S^{1-k} other than isp^k and isp^{1-k} (i.e., by all processes of S^T).

We know that $\beta(x)^k$ and $\beta(x)^{1-k}$ are sequences of all operations of $\alpha(x)^k$ and $\alpha(x)^{1-k}$, respectively, because they are legal views. Then, since $\beta(x)^T$ is formed as the sequence of operations of S^T obtained by concatenating the sequences of legal views $\beta(x)^k$ and $\beta(x)^{1-k}$, it is a sequence of all operations of $\alpha(x)^T$.

Lemma 12. *$\beta(x)^T$ preserves the execution order $\prec$ on $\alpha(x)^T$.*

Proof. We show that if there are two operations op and op' in $\alpha(x)^T$ such that $op \prec op'$ on $\alpha(x)^T$, then $op \rightarrow op'$ in $\beta(x)^T$. We have two possible cases.

Case 1. op and op' have been issued by processes of a same system. Let us suppose that op and op' are issued by processes of S^k. Note that, by definition, S^k is a cache system. Then, from Definition 5, there must be a legal view $\beta(x)^k$ preserving the execution order $\prec$ on $\alpha(x)^k$, and hence, from Definition 3, if $op \prec op'$ on $\alpha(x)^k$ then $op \rightarrow op'$ in $\beta(x)^k$. It is easy to check from the definition of $\beta(x)^T$ that operations of α^T and issued by processes of S^k appear in $\beta(x)^T$ and in $\beta(x)^k$ in the same order. Hence, $op \rightarrow op'$ in $\beta(x)^T$.

Case 2. op and op' have been issued by processes of different systems. Let us suppose that op is issued by some process of S^k, and op' is issued by some process of S^{1-k}. We know, from Case 1, that $\beta(x)^T$ preserves $\prec$ on $\alpha(x)^k$, and also preserves $\prec$ on $\alpha(x)^{1-k}$. Then, $\beta(x)^T$ will preserve $\prec$ on $\alpha(x)^T$ if it also preserves $\prec$ between any two operations from different systems. Then, by definition of $\beta(x)^T$, it is enough to show that the second condition of Definition 2 is preserved between two operations op and op' from different systems such that $op = w^k(x)u$ and $op = r^{1-k}(x)u$ in $\beta(x)_i^T$. We can see, by definition, that op must be the i^{th} write operation propagated from S^k to S^{1-k} (that is, $op(x)_i^k$), and op' is a read operation in $head(x)_i^{1-k}$. Then, by definition, $op \rightarrow op'$ in $\beta(x)_i^T$, and, hence, $op \rightarrow op'$ in $\beta(x)^T$.

Lemma 13. *$\beta(x)^T$ is legal.*

Proof. Let $op = r(x)u$ be a read operation of $\beta(x)^T$. From Definition 4, $\beta(x)^T$ is legal if $op' = w(x)u$ is the nearest previous write operation to op in $\beta(x)^T$. We know, by definition, that $\beta(x)^k$ is the same sequence than $\beta(x)^T$ but replacing each write operation op from isp^k by $prop(op)$. We have two possible cases.

Case 1. $op = r^k(x)u$ and $op' = w^k(x)u$ are operations in $\alpha(x)^T$ issued by processes of S^k. By definition, as $\beta(x)^k$ is a legal view of execution $\alpha(x)^k$ preserving $\prec$ on $\alpha(x)^k$, $op' = w^k(x)u$ is the nearest previous write operation to $op = r^k(x)u$ in $\beta(x)^k$. Then, by definition of $\beta(x)^T$, op' also is the nearest previous write operation to op in $\beta(x)^T$. Therefore, $\beta(x)^T$ is legal.

Case 2. $op = r^k(x)u$ and $op' = w^{1-k}(x)u$ are operations in $\alpha(x)^T$ issued respectively by systems S^k and S^{1-k}. Let $op' = w^{1-k}(x)u$ be the write operation $op(x)_i^{1-k}$. Then, its corresponding write operation in S^k is $prop^k(op(x)_i^{1-k}) = w^k_{isp^k}(x)u$. By definition, as $\beta(x)^k$ is a legal view of $\alpha(x)^k$ preserving $\prec$ on $\alpha(x)^k$, $prop^k(op(x)_i^{1-k})$ is the nearest previous write operation to op in $\beta(x)^k$. Then, by definition of $\beta(x)^T$, $prop^k(op(x)_i^{1-k})$ is replaced by $op' = op(x)_i^{1-k}$ to obtain $\beta(x)^T$, and $op' = op(x)_i^{1-k}$ also is the nearest previous write operation to op in $\beta(x)^T$. Therefore, $\beta(x)^T$ is legal.

Theorem 6. *The system S^T is cache.*

Proof. From Definition 5, S^T is cache if in each execution α^T there is, for all variable x, a legal view of $\alpha(x)^T$ that preserves the execution order $\prec$ on $\alpha(x)^T$. From Lemma 11, $\beta(x)^T$ is formed by all operations of $\alpha(x)^T$. From Lemma 12, $\beta(x)^T$ preserves the execution order $\prec$ on $\alpha(x)^T$. Finally, from Lemma 13, $\beta(x)^T$ is legal. Then, $\beta(x)^T$ is a legal view of $\alpha(x)^T$ that preserves the execution order $\prec$ on $\alpha(x)^T$. Hence, S^T is a cache system.