Towards the Hierarchical Group consistency for DSM systems : an efficient way to share data objects

Laurent Lefèvre, Alice Bonhomme

INRIA RESO Team – LIP Laboratory – Lyon, France
laurent.lefevre@inria.fr

DSM

- Scalability for large scale systems (clusters with hundred of nodes)

Plan

- Consistencies in DSM : formal comparison and graphical visualization
- The Hierarchical Group Consistency proposal
- Deployment in DOSMOS system
- First experiments
- Conclusions

Consistencies

- To be efficient : DSM must manage different copies of shared data (objects or pages) to allow concurrent operations
- How to be sure of the value read in a data copy ?
- Since last decade : dozens of consistencies have been proposed for DSM
- Most of them : models with slight differences

Consistencies

- 5 main consistencies
  - Strong :
    - Atomic consistency : Perfect model, difficult to implement on multi processor architecture
    - Sequential consistency : from Lamport. All processes see same actions on shared memory. Execution result like in sequential order.
  - Weak :
    - Release consistency : based on Acquire / Release operations - 3 conditions must be respected :
      - Before any access operation all previous acquire must be processed
      - Before a Release, all pending access (writing or reading) must be processed for all processes
      - Synchronization operations must be sequentially consistent
      - Lazy release consistency

Consistencies

- Entry consistency : Each shared data is explicitly associated to a synchronization variable. Before an Acquire all pending accesses associated with this Acquire must be processed.
- Scope Consistency : based on Entry consistency. Add an implicit association between synchronization variables and shared data.
    - Cohrence domain : limited view of memory where we can perform acquire and release opeartions. All modifications only visible in a dmoain.
    - Conditions :
        - Before an Acquire in a domain, all pending operations must be performed
        - Before a shared access done by process P, all pending Acquire done by P must be performed

Problems

- There are many more, but equivalent for a programmer… and difficult to add them in a distributed application.
- How to clearly understand their difference and compare them ?
- We need 3 definitions : memory consistency, execution of program and synchronization order

Memory consistency

- A ***memory consistency model*** M is a two-tuple (C_M, SYN_M) where C_M is the set of possible memory accesses (read, write, synchronization) and SYN_M is an inter-processes synchronization mechanism to order the execution of operations from different processes.

- Execution order of synchronization accesses determines the order in which memory accesses are perceived by a process.
- For each application : several possible executions.

Execution of program

- An ***execution of the program*** PRG under consistency model M, denoted as $E_M(PRG)$, is defined as an ordering of synchronization operations of the program

- *With the ordering of synchronization operations, the execution of all related operations are also ordered. Thus, we define the synchronization order of an execution.*

Synchronization order

- The **synchronization order** of an execution $E_M(PRG)$ under consistency model M, denoted as $SO_M(E_M(PRG))$, is defined as the set of ordinary operation pairs ordered by the synchronization mechanism SYN_M
- Hence, for any consistency model M, we can define C_M and $SO_M(E_M(PRG))$. C_M deals with how the programmer has to program, and SO_M gives the rules used to generate the result.

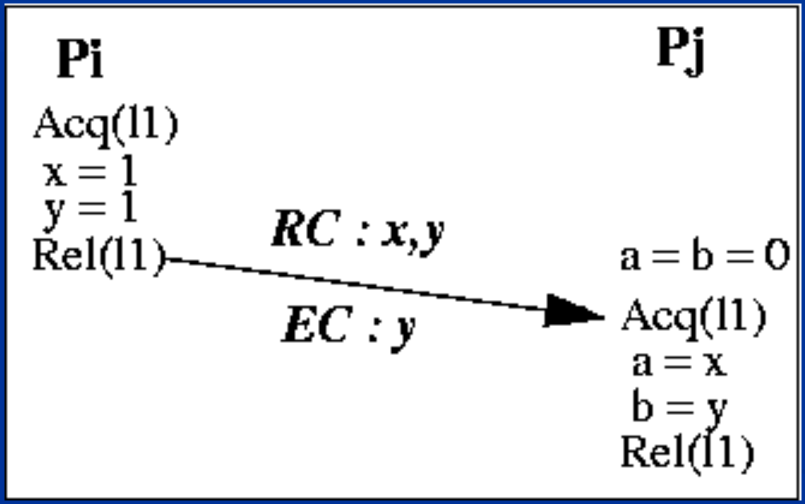
Formal comparison

- 2 models M1 and M2 are equivalent iff :
  - $C_{M1} = C_{M2}$
  - a correct program PRG for M1 is also correct for M2
  - if 2 compatible executions E_{M1}(PRG) and E_{M2}(PRG) give the same result.

- E_{M1}(PRG) and E_{M2}(PRG) are said compatible executions if
  - there does not exist (u,v), 2 synchronization operations such that $(u,v) \in E_{M1}(PRG)$ and $(v,u) \in E_{M2}(PRG)$

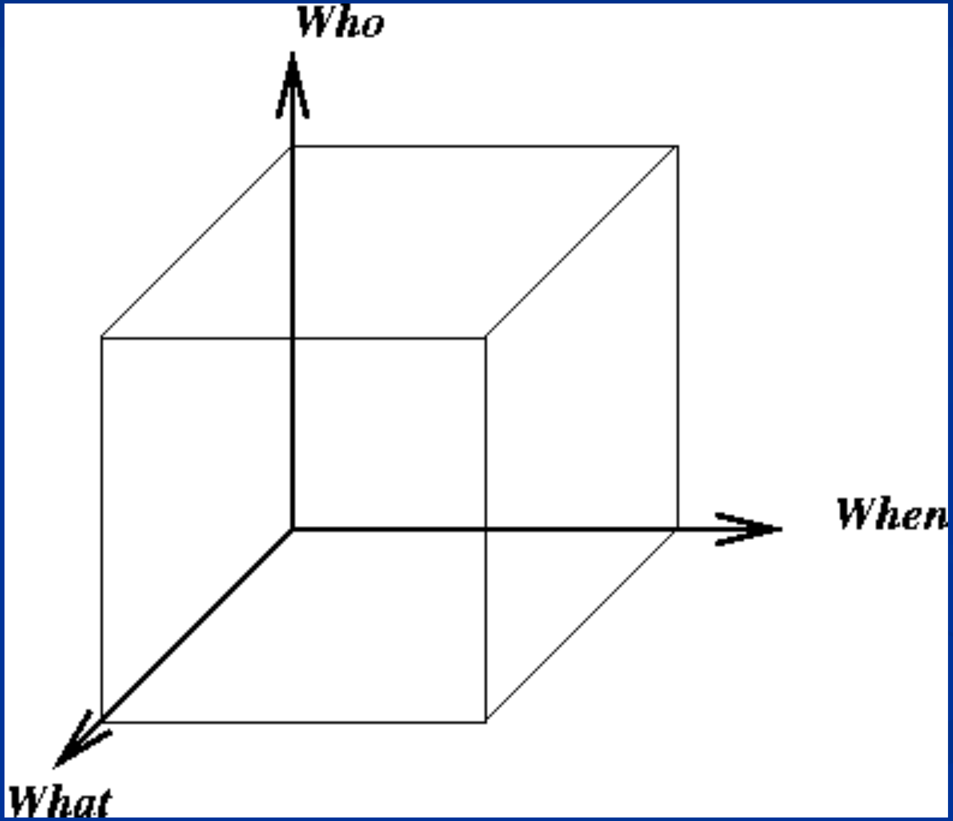
Example 1

- Release consistency RC different from Entry Consistency EC

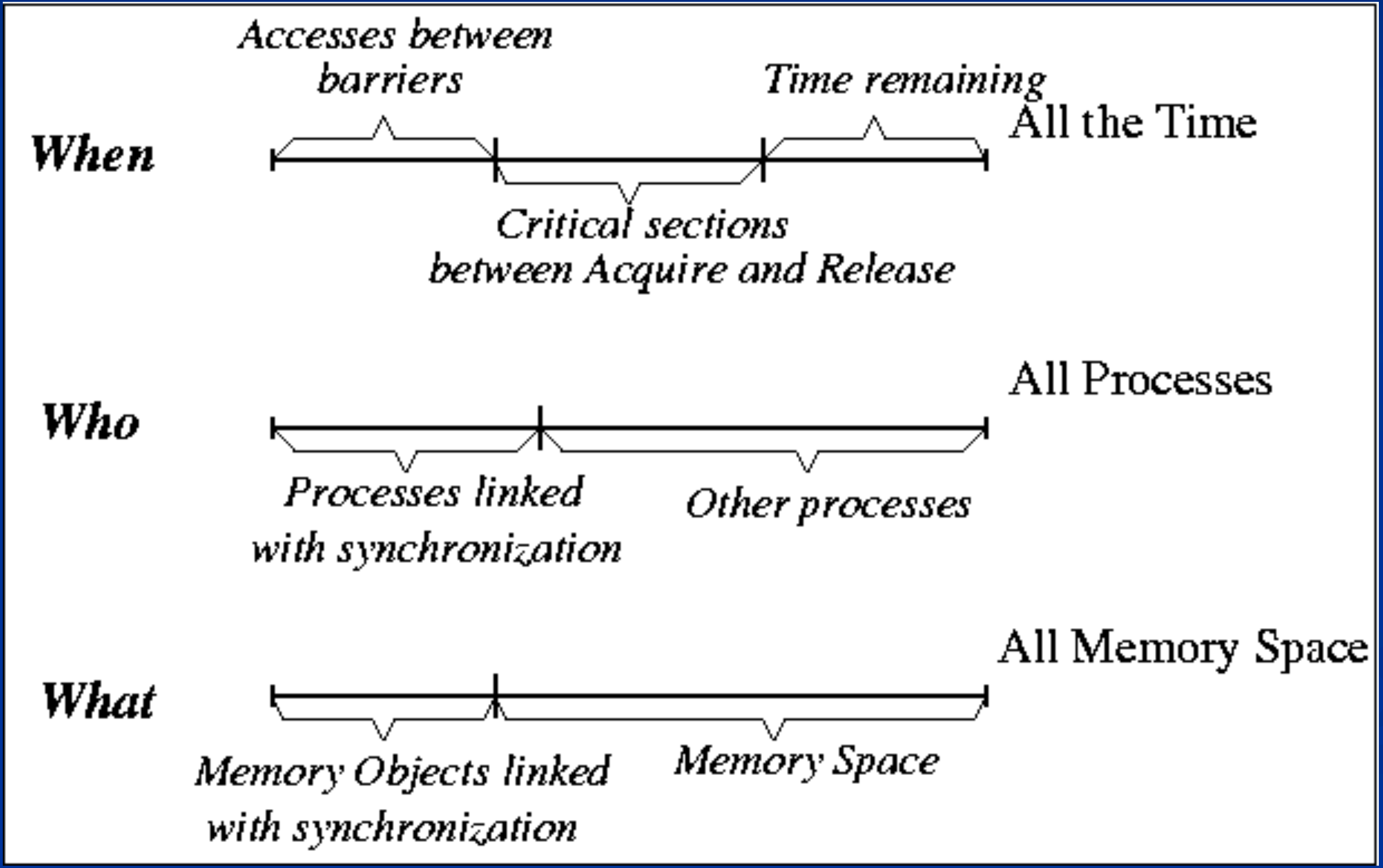


Example 2

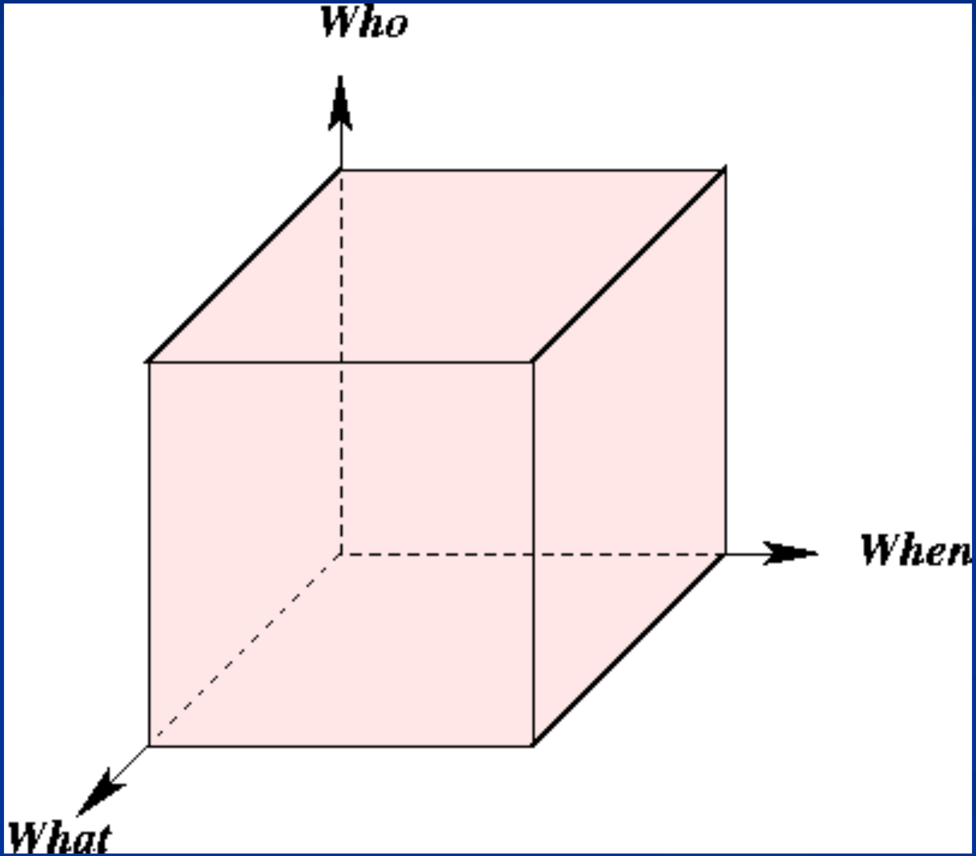
Graphical visualization



Graphical visualization

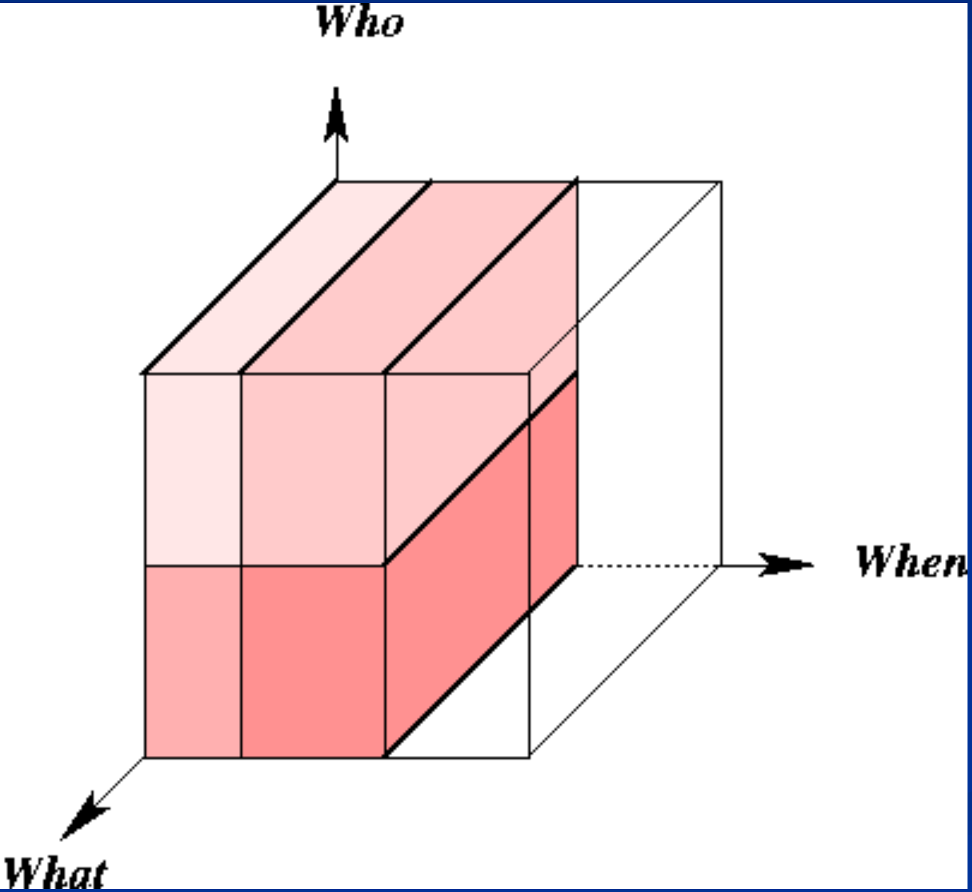


Strong consistencies



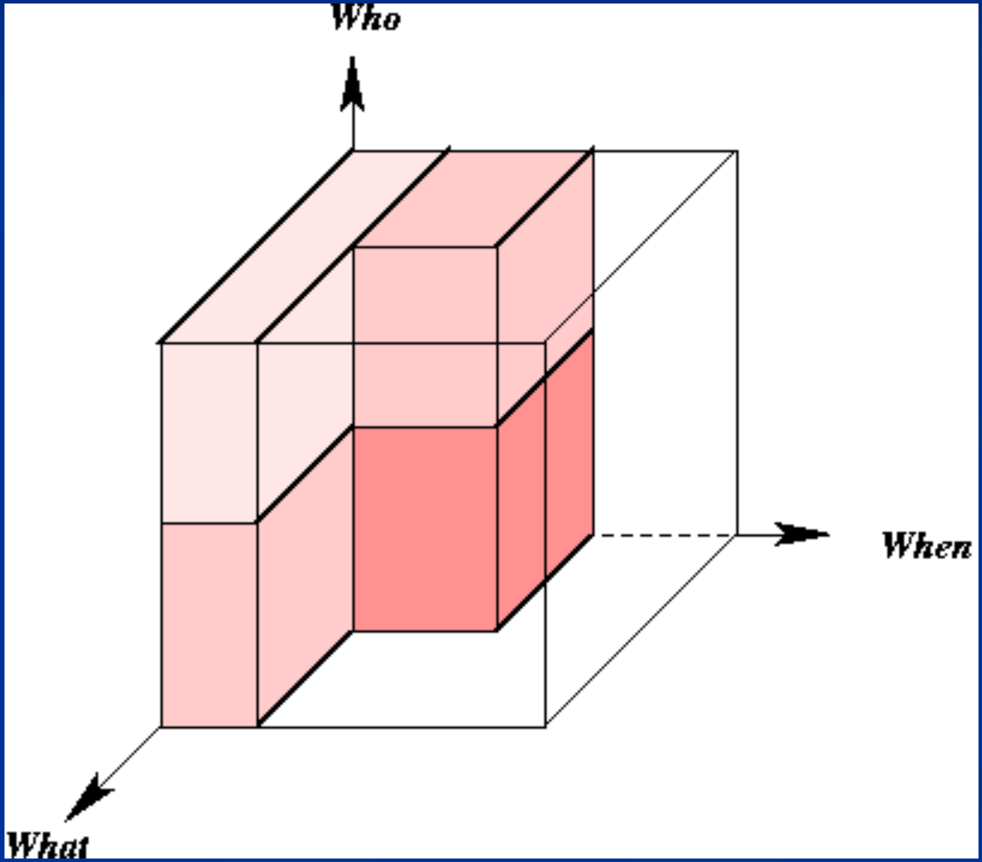
Release consistencies

- Relaxes When axis



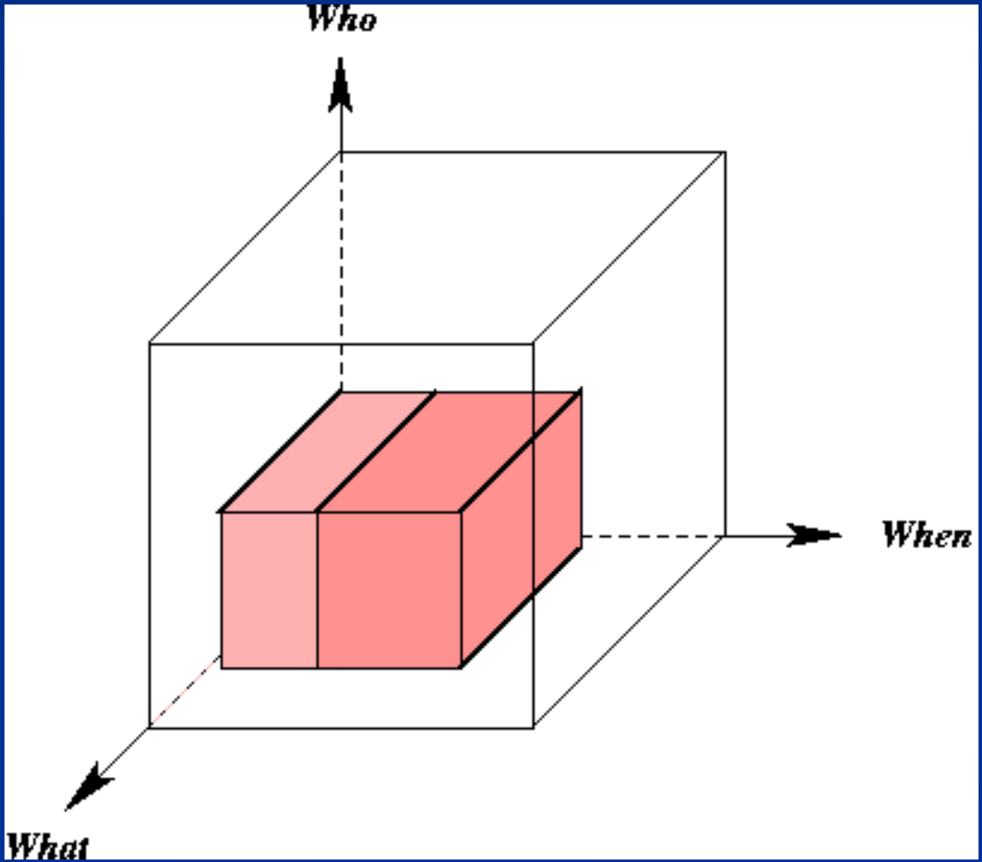
Entry consistency

- Relaxes What axis



Need a new model

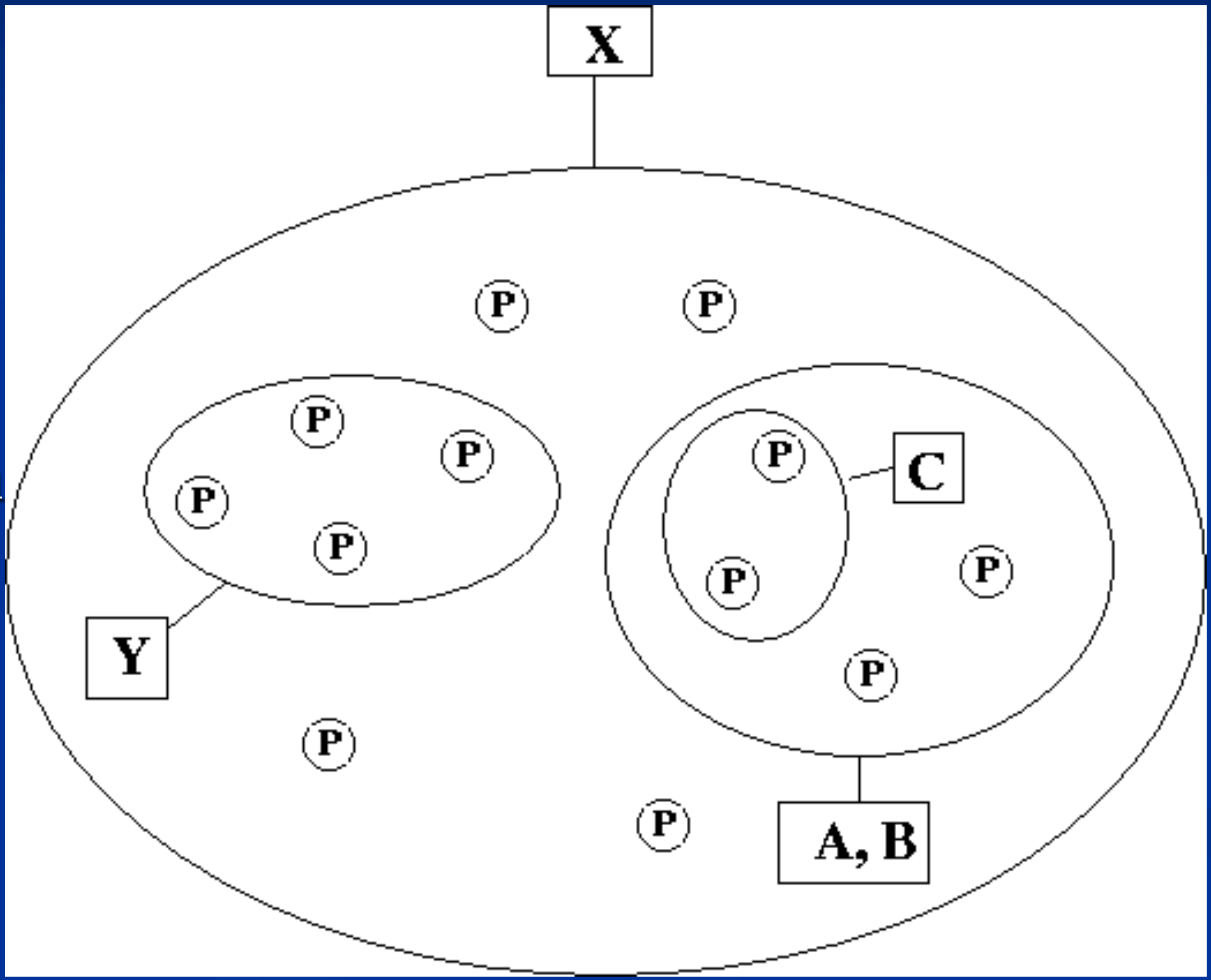
- Relaxes Who axis
- Not all processes share same data
- Do not apply consistency on all data



Hierarchical Group Consistency

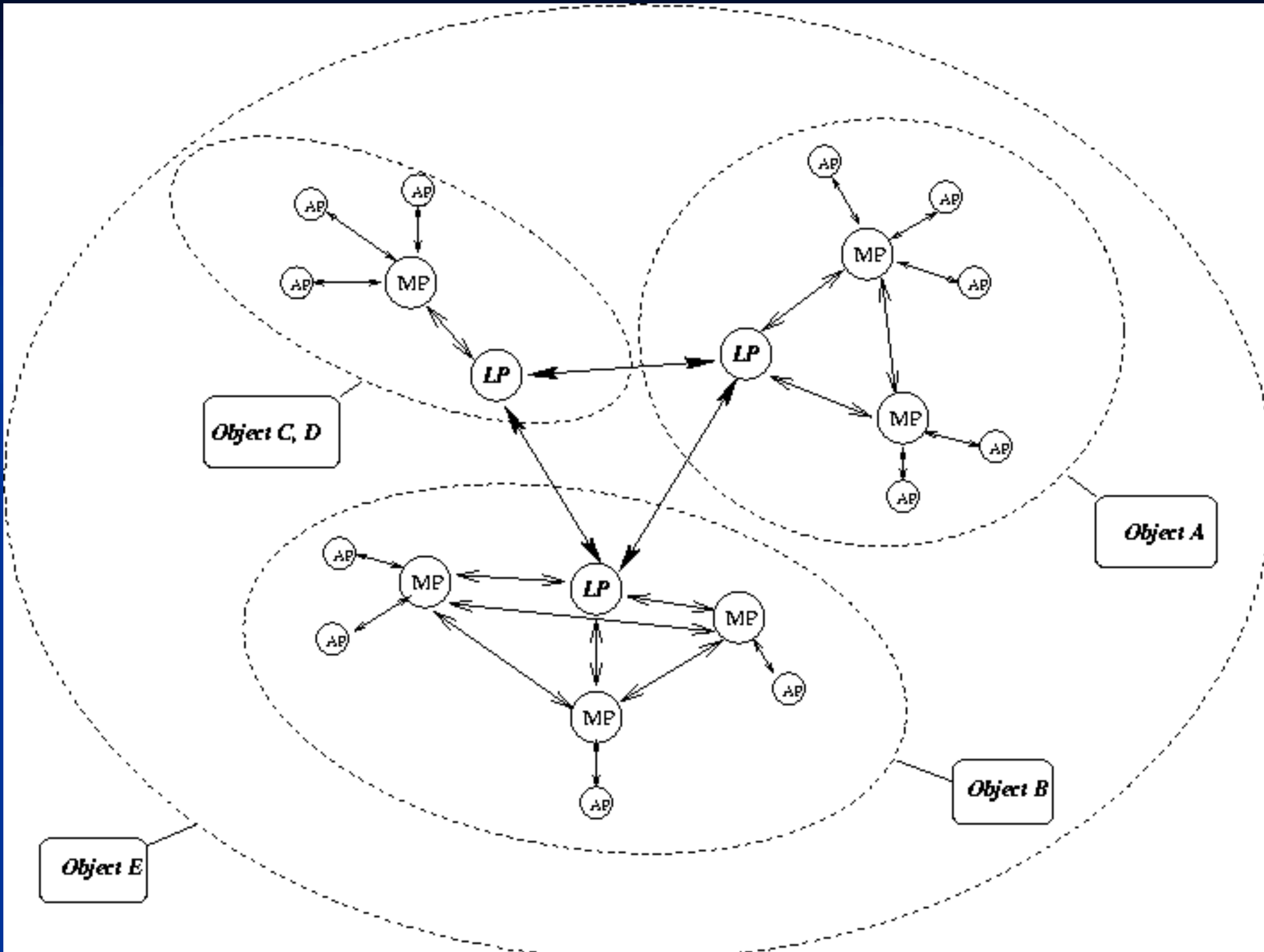
- HGC model is defined by :
  - C_{HGC} = read(x), write(x), Acq(l), Rel(l), Sync(l)
  - u,v $\in SO_{HGC}(E_{HGC}(PRG))$ iff $\exists$ a synchronization variable l to which u and v are associated such that: u is performed before Rel(l) and v is performed after Rel(l).
  - OR u is performed before Sync(l) and v is performed after Sync(l)
- HGC is different from EC and RC due to the add of new sync operation (barrier restricted to a synchronization variable).

- Groups : set of processes sharing same data
- Can be organized hierarchically
- Different consistencies can be deployed in different groups or on different data
- No consistency is maintained between groups



DOSMOS

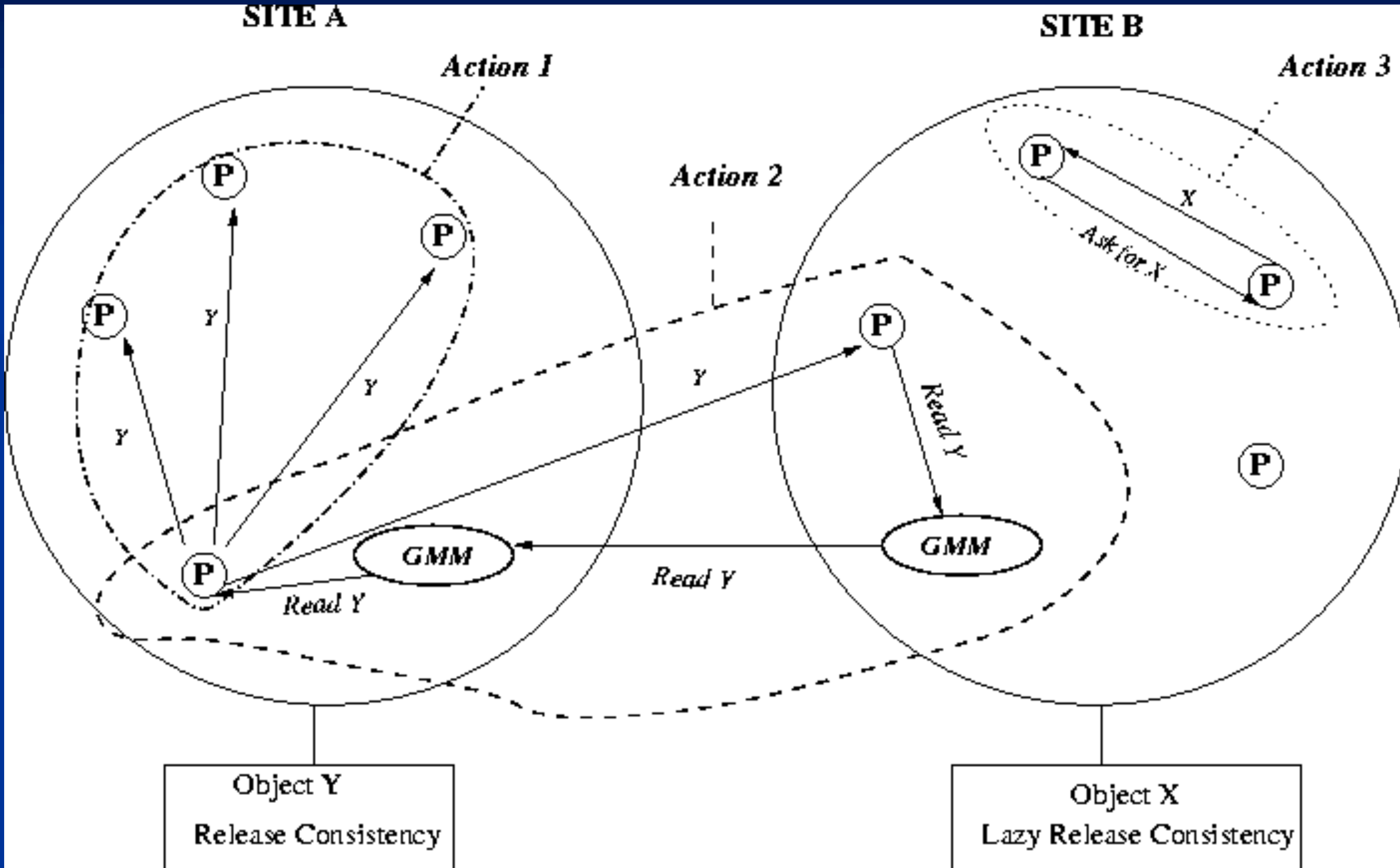
- Distributed Objects Shared MemOry System
- Provided on top of standard message passing libraries (PVM / MPI)
- Multi-threaded / multi-processes
- 3 classes of processes :
  - Application processes
  - Memory processes
  - Link processes
- Implements Release consistency and HGC model
- Invalidation / update protocols
- Dynamic / static owner



AP
AP
AP
MP
LP
AP
AP
MP
AP
LP
MP
AP
AP
Object C, D
Object A
AP
MP
LP
AP
MP
AP
MP
AP
Object B
Object E

2 kind of accesses

- Local operations inside a group with the same consistency
- Distant operations between groups through the Group Memory Manager (Link Process)


SITE A
SITE B
Action 1
Action 2
Action 3
P
Y
X
Ask for X
Read Y
GMM
Object Y
Release Consistency
Object X
Lazy Release Consistency

2 kind of accesses

	Intra group acess	Inter group access
Reading operation	max : 2 min : 0	max : 4 min : 2
Writing operation	max : 1 min : 0	max : 3 min : 3

- Easily allow a personalized consistency for each shared data
- Groups statically defined by user
  - May be difficult : assisted development tools to design applications

Experiments

# global synchro	1	2	4
Performance Gain	43.3 %	41.9 %	39 %

Experiments

	2 groups	4 groups
1 synchro	32.7 %	39.8 %
2 synchro	29.8 %	33.5 %
4 synchro	19.5 %	28.7 %

- First experiments on multi-cluster architecture show improvement of around 20 % for 2 groups

Conclusion and future works

- Presented of a new consistency model and implementation
- Focus more on programmer point of view than of consistency differences
- Providing dynamic adaptive groups
- Deploying HGC based systems on high performance dedicated Grid
- Using HGC in DSM based clustered high performance active nodes