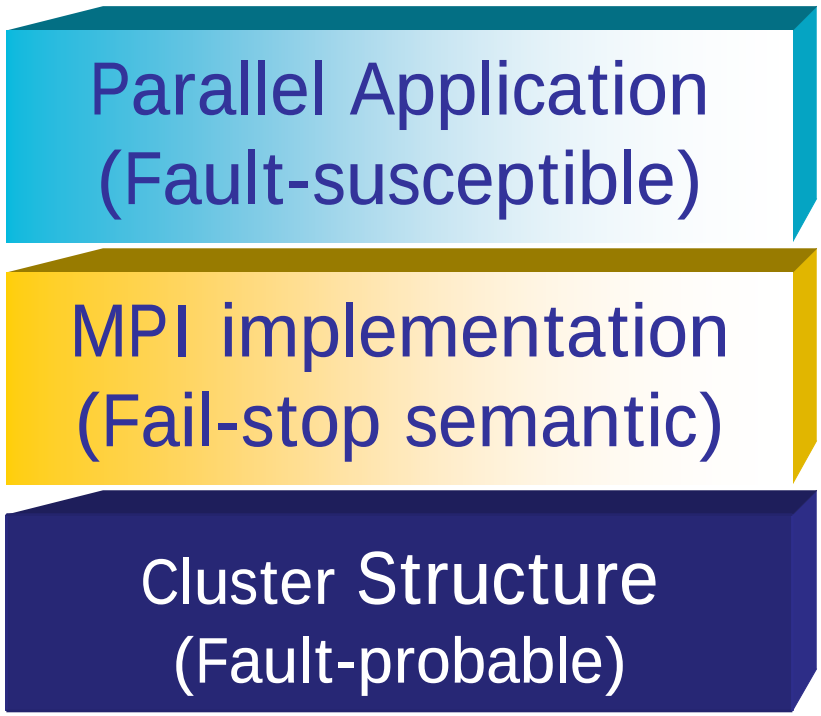
An Intelligent Management of Fault Tolerance in Clusters using RADICMPI

Angelo Duarte, Dolores Rexachs, Emilio Luque
Computer Arquit. and Oper. System Departament - CAOS
University Autonoma of Barcelona – UAB

Euro PVM/MPI – 09/20/2006

Legacy MPI implementations and Faults



Error / Stall

↑

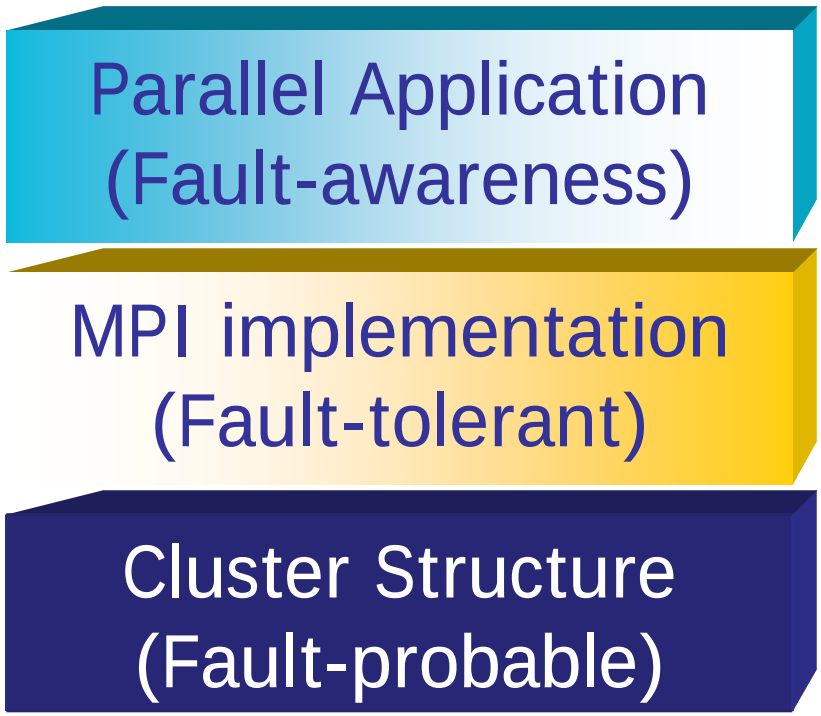
Collapse

↑

Fault

UAB caos Euro PVMPI '06

Non-transparent fault tolerant MPI implementations and faults



Adaptation

↑

Fault handling

↑

Fault

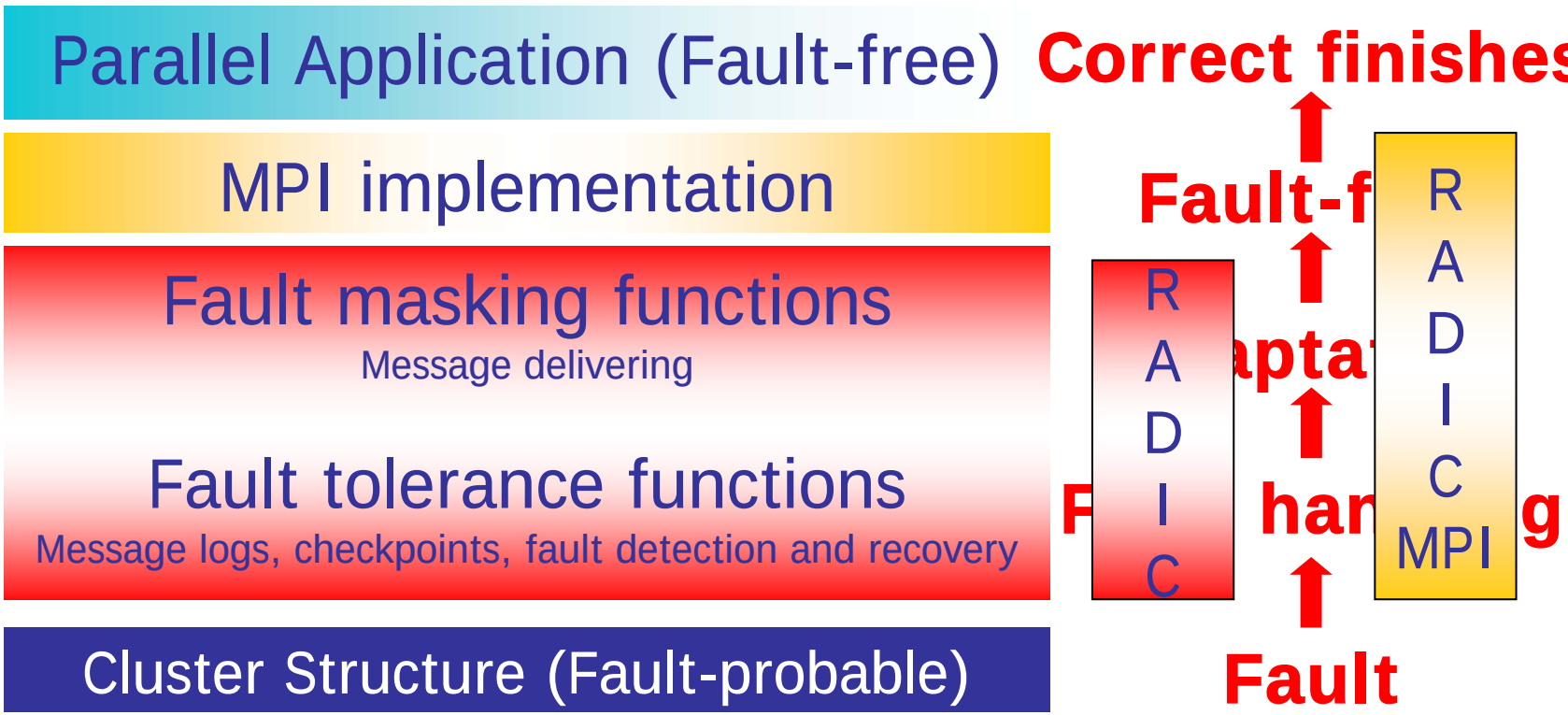
UAB caos EuroPVMPI'06

Problem Statement

To design a fault tolerant architecture that requires neither central elements nor stable elements, and that transparently handles faults in clusters to assure the applications based on message passing correctly finish

- Solution requirements
  - Transparent to Programmers and System Administrators
  - Decentralized control in order to not constratint the scalability
  - Independence of dedicated resources or full-time stable elements

Our approach (Transparent)



Summary

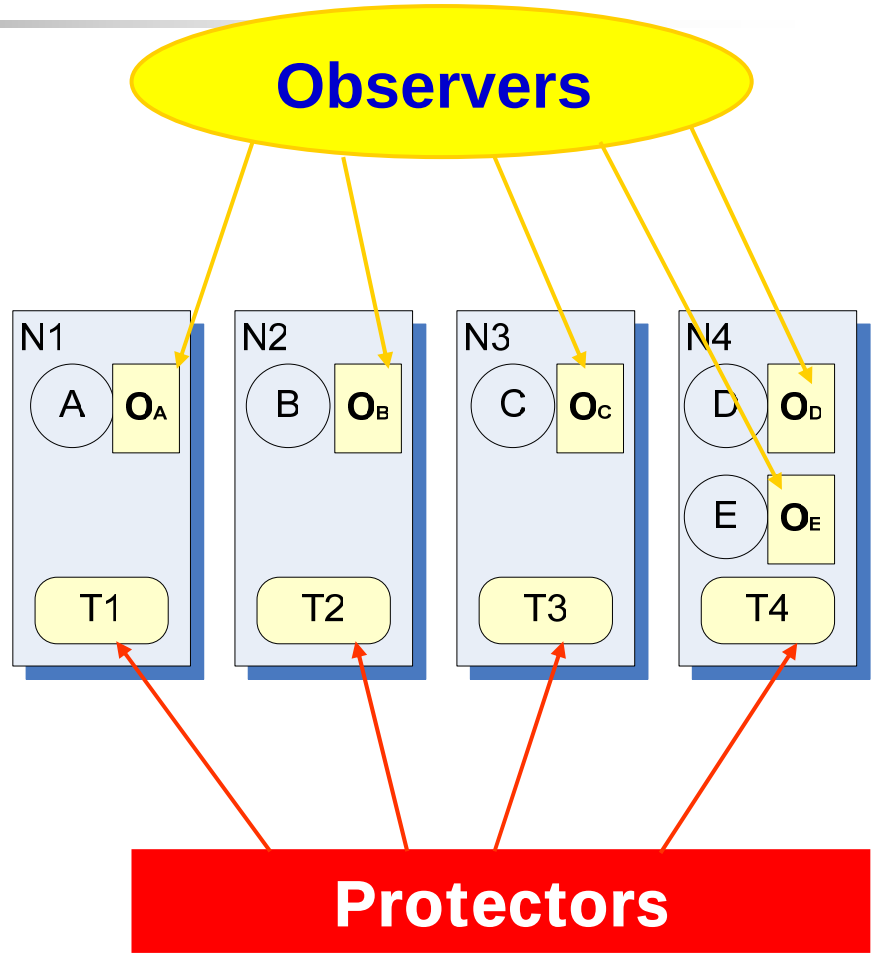
- RADIC
- RADICMPI
- Case study
- Conclusions
- Present and future work

What is RADIC?

- Redundant Array of Distributed Independent Checkpoint
- **RADIC** is an architecture based on a set of dedicated distributed processes (**protectors** and **observers**)
  - **Protectors** and **observers** compose a fully distributed fault tolerance controller

RADIC processes

1. Every application process has an ***observer*** process (O)
2. Every node has a ***protector*** process (T)

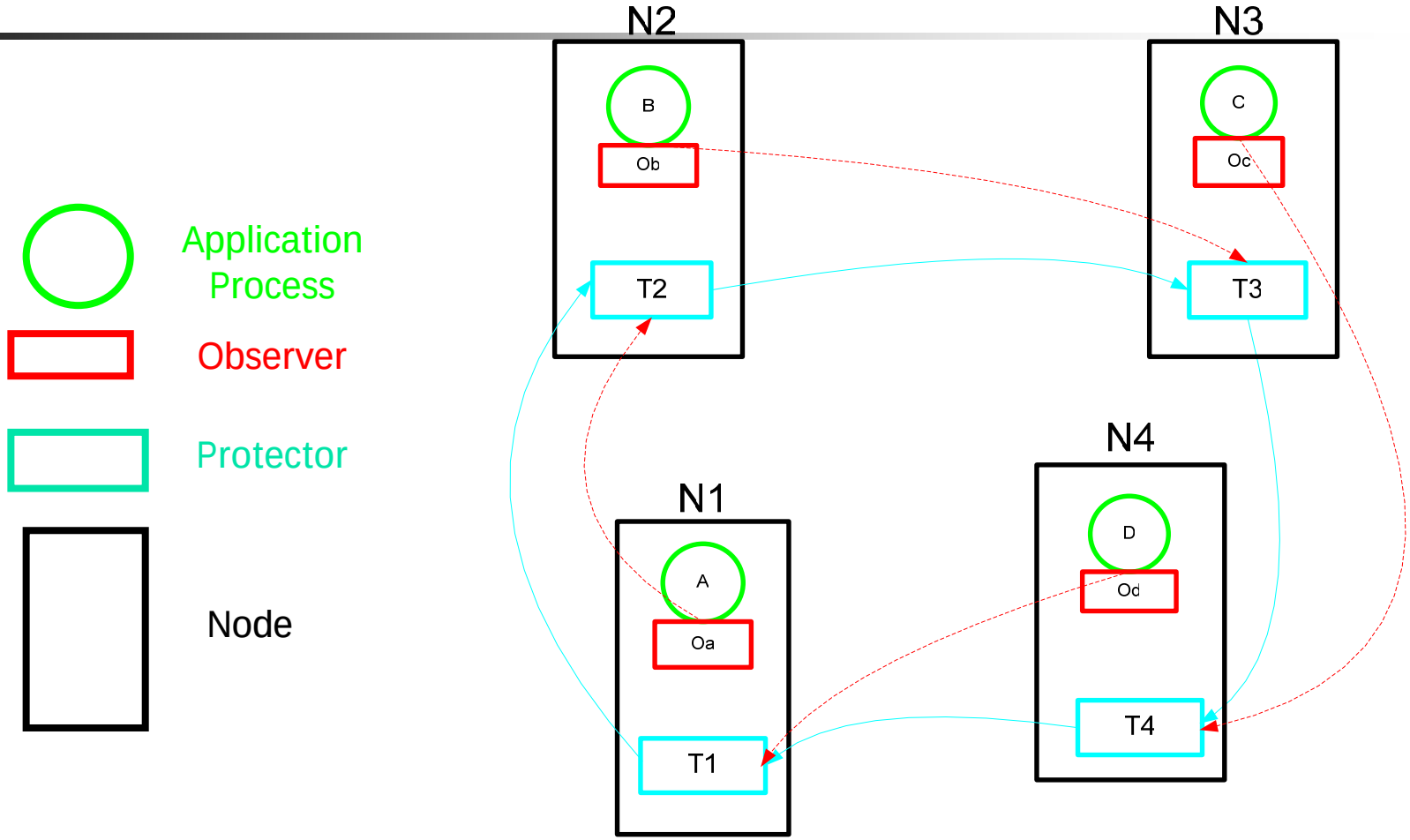


RADIC – Protection

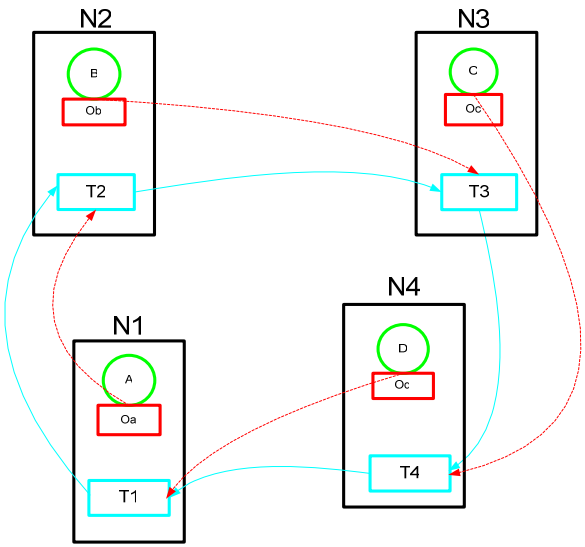
- ***Protectors*** *detect faults, receive and* store checkpoints and message logs and also implement the rollback-recovery protocol
  - In RADICMPI protectors are created before application processes
- ***Observers*** handle message delivering and take checkpoints and message logs of its application process in order to send them to a protector in a different node
  - In RADICMPI observers are created together with application processes by MPI_Init()
  - An observer establishes a neighbor protector whenever:
    - The observer starts or is recovered
    - The neighbor protector fails

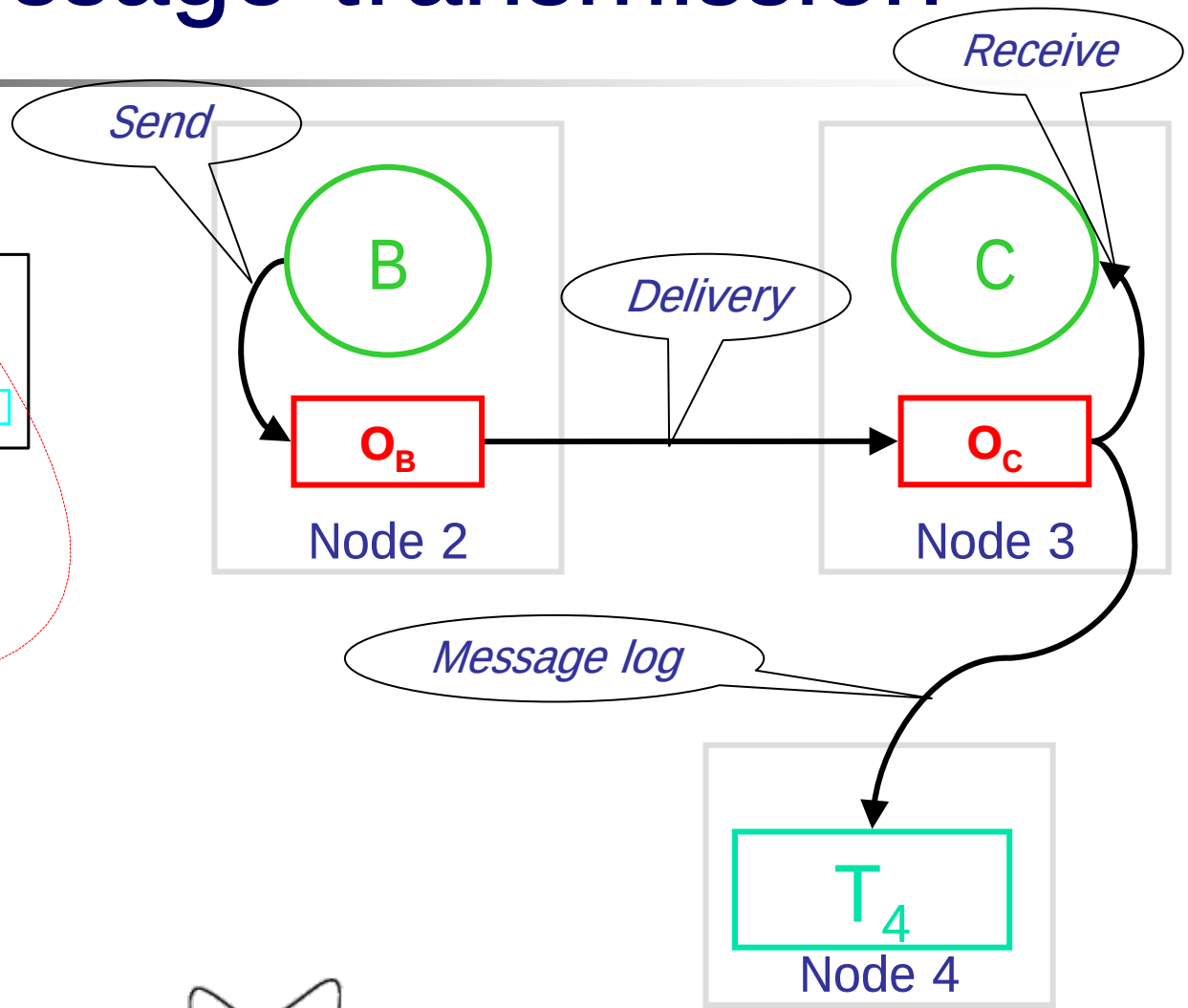
UAB caos EuroPVMPI'06

Cluster with RADIC



Message transmission





RADIC – Fault detection

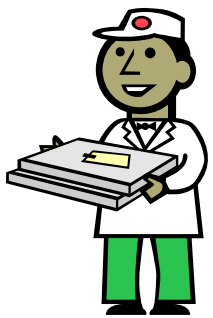
- Each ***protector*** monitors a neighbor and is monitored by a different neighbor protector
- Each ***observer*** detect faults when it communicates to :
  - its protector
  - another observer

RADIC – Recovering

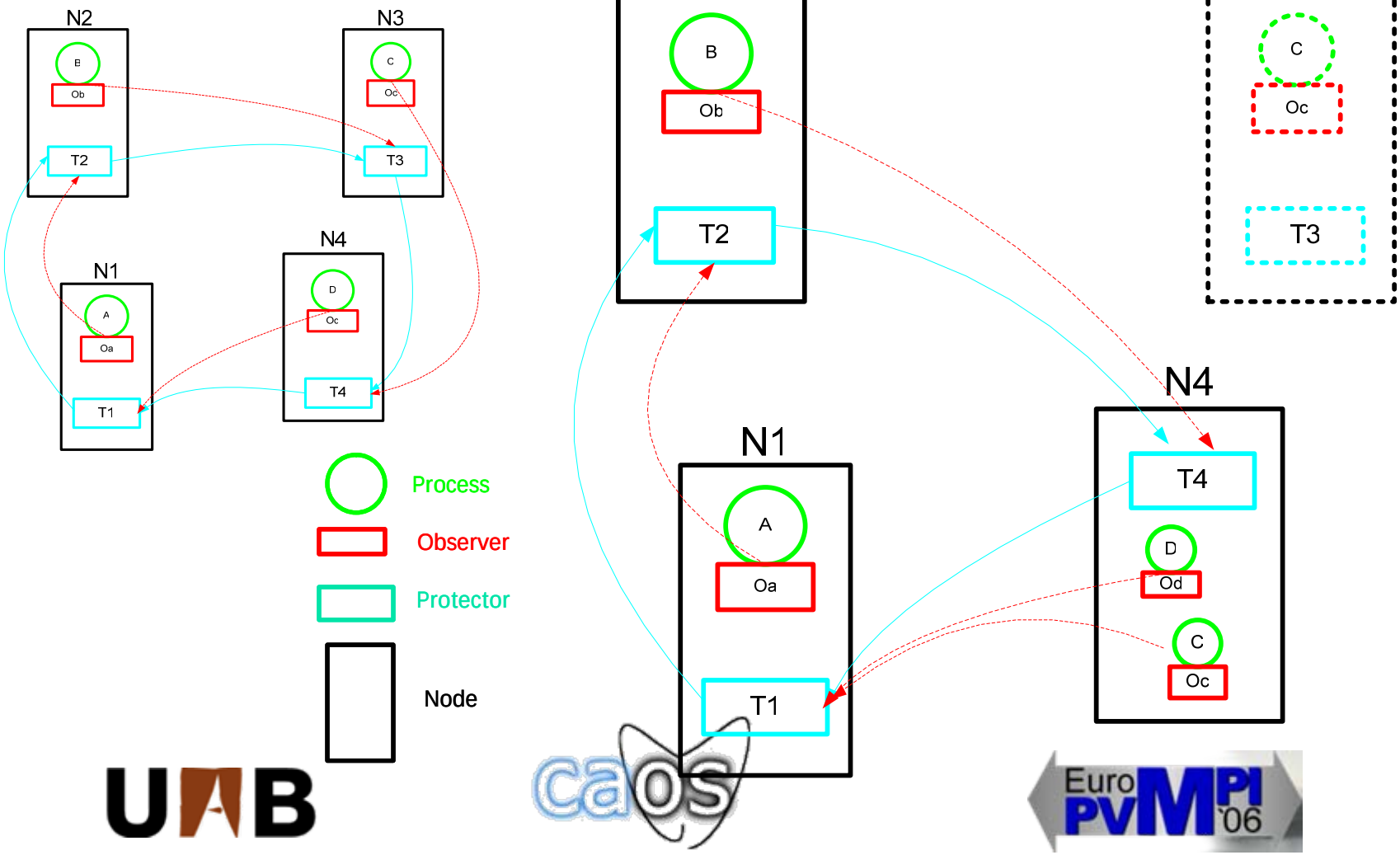
- A ***protector*** recovers faulty (application process + its observer)
  - State consistence depends on rollback-recovery protocol
- A recovered ***observer***
  - Establishes its new protector
  - Handles the message log for its application process
- A recovered ***application process*** maintains the same rank in MPI_Comm_world
  - But its node address do change

RADIC – Fault masking

- Each observer maintains a local routing table with the address of all other observers
  - The routing table has all processes node address and protector address in the cluster
  - When a communication with a destination process fails, the observer "calculates" in which node the faulty destination will be recovered
  - Processes rank remains the same



RADIC - Fault handling

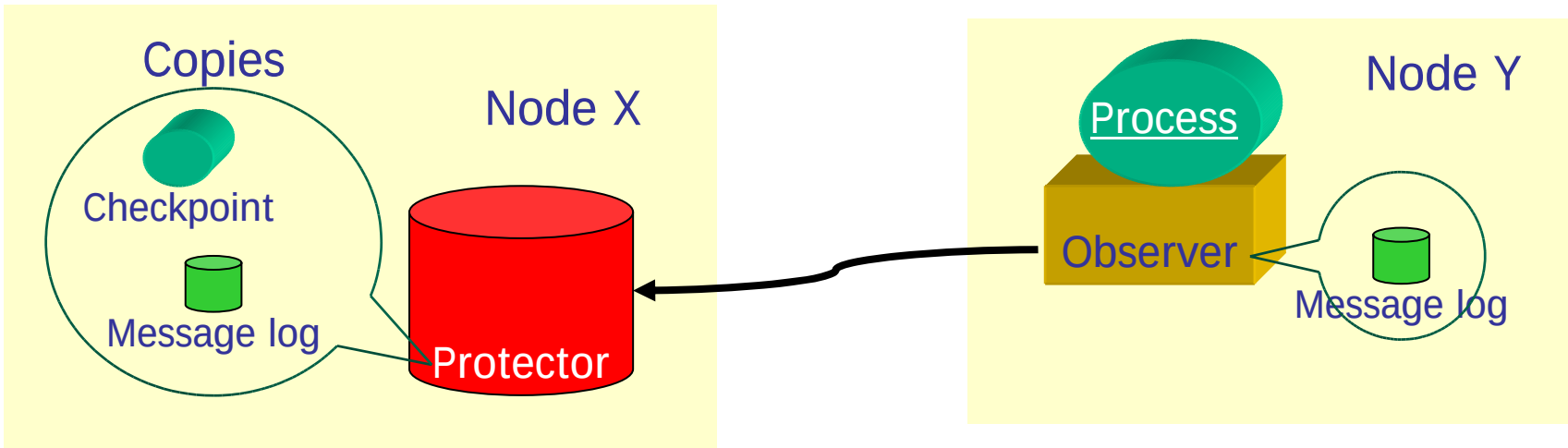


RADICMPI

- RADICMPI is a MPI prototype implementation created to test the RADIC concepts
  - Implementation of *protectors* and *observers* processes
  - Fully transparent to programmers
    - System administrators just need to define protectors/observers relationship using the machine file
  - Receiver-based pessimistic log protocol
    - Assure independence between processes for checkpointing and recovering

RADICMPI Distributed Storage

- No dedicated stable storage required
  - Reliability is achieved by duplicating logs and checkpoints like in RAID-1 scheme



RADICMPI - Implementation

- IBM-PC architecture (32-bit)
- Communication based on TCP sockets over Ethernet
- Heartbeat/watchdog scheme to fault detection
- Checkpoint using BLCR (Berkeley Labs Checkpoint/Restart)

RADICMPI - Middleware

- RADICMPI functions
  - `MPI_Init`
  - `MPI_Send`
  - `MPI_Recv`
  - `MPI_Sendrecv`
  - `MPI_Finalize`
  - `MPI_Wtime`
  - `MPI_Comm_rank`
  - `MPI_Comm_size`
  - `MPI_Get_processor_name`
  - `MPI_Type_size`
- RADICMPI environment
  - `radiccc, radicrun`
  - `customizable machinefile`

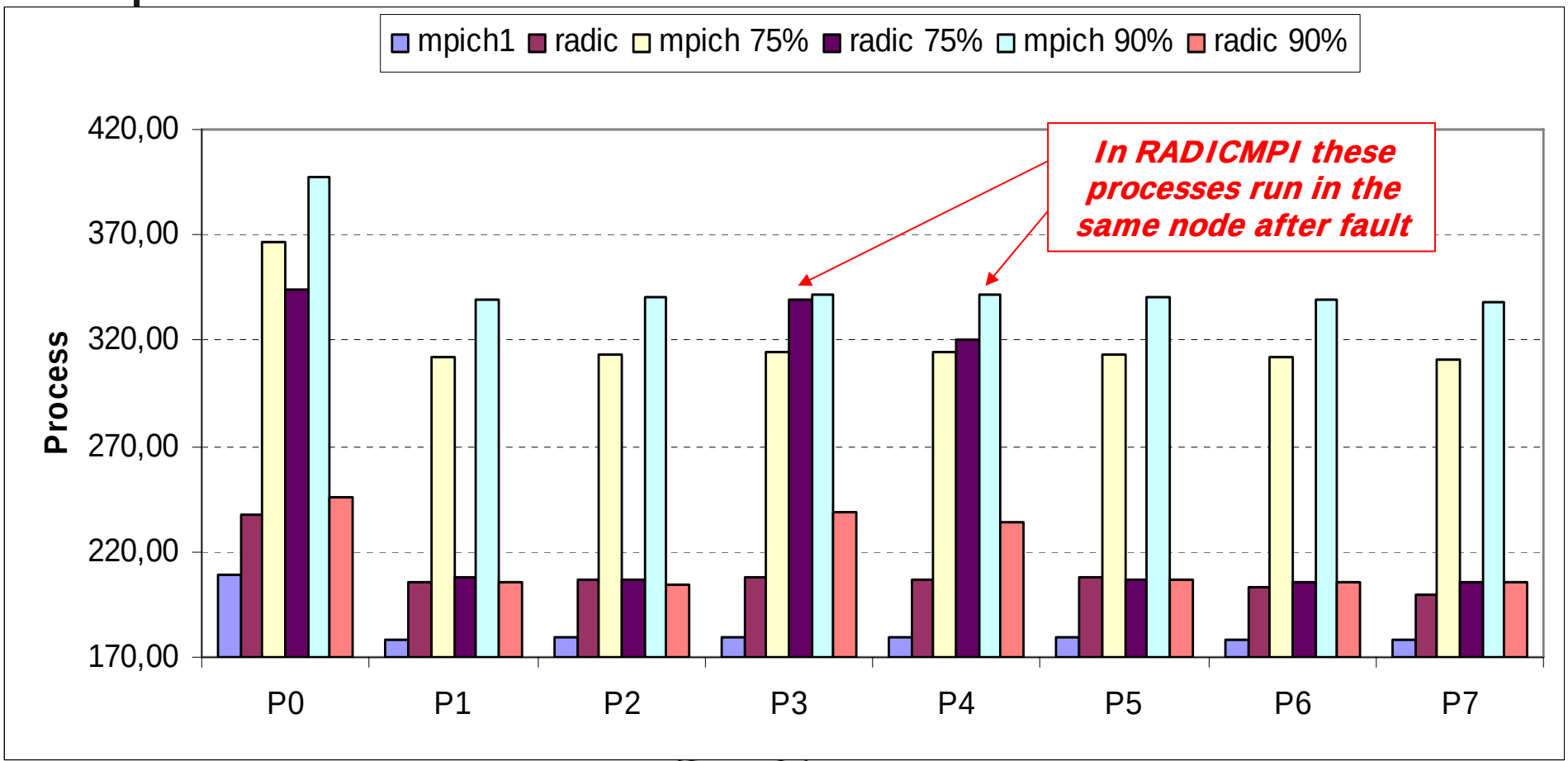
Experiments

- Homogeneous Cluster with 12 nodes
  - AMD Athlon(TM) XP 2600+, 256MB RAM, 40GB, Linux FC2 y FC5, 100 BaseT Ethernet Switch
- Algorithms
  - Ping-pong
  - Matrix multiplication (MW and SPMD)
  - Minimum string search

Case study

- Multiplication of 2 Hilbert matrix (3000x3000 double float)
- 1 master (P0), 7 workers (P1-P7).
  - Fault injected in process P3 at 75% and 90% of total execution time without faults
- Focus on time saved using RADIC compared against restarting the application from the beginning
  - Restart time was considered null (very optimistic)
  - Compared with MPICH1

Processes execution times



Time saving

- Time saved using RADIC compared against restart the application from the beginning after a node failure in process P3
  - Restart time was considered null

Fault at	MPICH1	RADIC	Time saving	
75%	366,38	343,89	22,49	6,1%
90%	397,78	245,97	151,81	38,2%

Conclusions

- Transparent
  - No concerns for programmers
  - Few concerns for system administrators
- Fully decentralized
  - No constraint to scalability
- Does not need stable elements to operate
- All nodes are used simultaneously to computation and protection
- Customizable
  - 1 protector x N observers, Dedicated spare nodes for protection, Protection groups inside the cluster, Rollback-recovery protocol parameters, ...

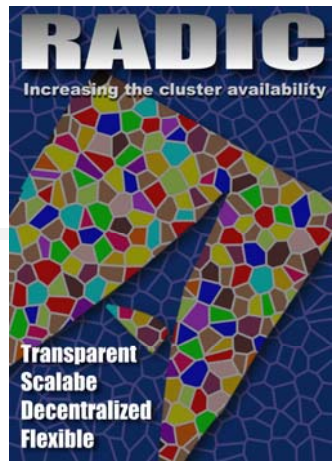
In the present we are...

- Improving checkpoint strategy to reduce resource consumption
- Implementing automatic cluster reconfiguration
  - Faulty nodes are replaced “on the fly”
  - Spare nodes assume faulty nodes
- Implementing non-blocking communication functions
- Testing functionality in several scenarios

In the future we shall...

- Have a complete set of MPI functions
  - Continue RADICMPI or adapt an existing MPI-1/2 implementation ?
  - RADIC-OpenMPI ?
- Define a suite of application benchmarks in order to better control program state size, message patterns and granularity
- Develop a simulator to study the influence of:
  - Checkpoint interval, Number of nodes, Application granularity, Fault distribution, Protectors structure, ...

Thank you very much for your attention



Angelo.Duarte@aomail.uab.es
http://www.caos.uab.es/

UAB caos EuroPVMPI'06