RADIC II

A Fault Tolerant Architecture with Flexible Dynamic Redundancy

By Guna Santos

Key Points

- The increasing number of applications demanding high performance and availability (*performability)*
- Fault tolerance plays a major role in order to keep high availability
- A fault tolerant solution must generates minimal overhead in its activities, in order to not affect the application performance
- After a fault ocurrence, some fault tolerant system generate system degradation, affecting the performance.

Challenges

- To provide a fault tolerant solution able to mitigate or to avoid the system degradation
  - Restoring the system configuration
  - Avoiding changes in number of active nodes
  - Allowing perform maintenance tasks, preventing faults

Contents

- Goals
- Fault Tolerance and the RADIC Architecture
- Recovery Side-Effect
- Protecting the System
- Implementation and Experimental Results
- Conclusions
- Future Work and Open Lines

caos

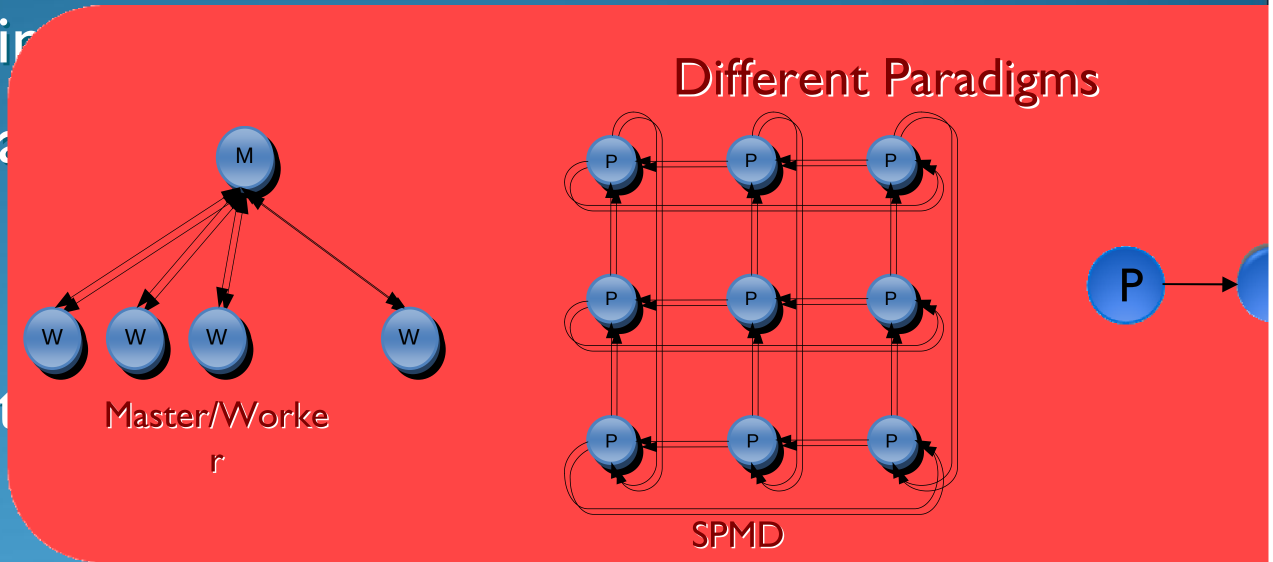
Goals

- To incorporate the ability to control the system degradation to a fault tolerant architecture with following characteristics:
  - Flexible, Decentralized, Transparent, Scalable
- To extend the functionality of this architecture implementation

caos

A Fault Tolerance Approach

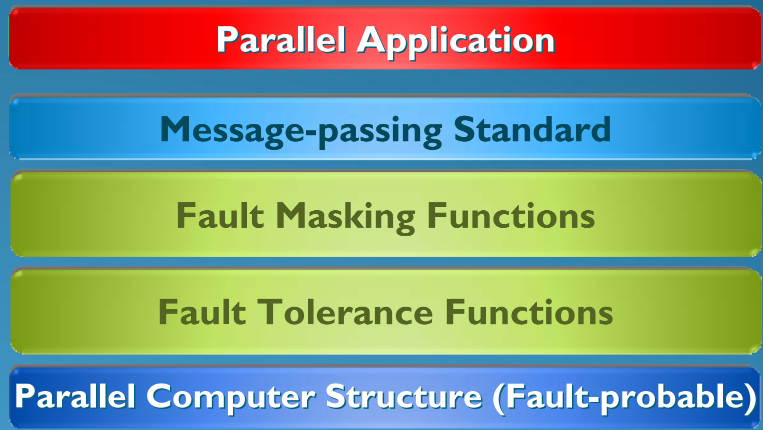
- Fault tolerant architecture for message-passin
- It based on Rollba protocol
- It creates a layer application from t failures



caos

A Fault Tolerance Approach

- Fault tolerant architecture for message-passing systems
- It based on Rollback-recovery protocol
- It creates a layer isolating the application from the cluster failures

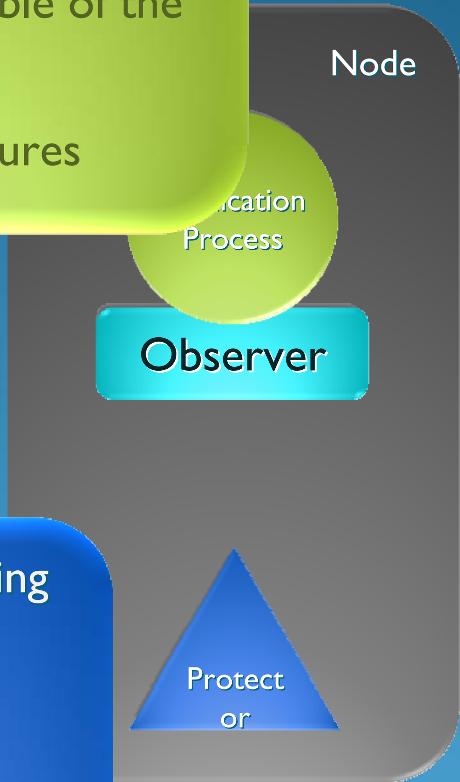


caos

The RA
Architec

- RADIC is a architecture passing systems
  - Two kind of process perform the fault toler

- It manages the communications between processes
- It takes checkpoints and event logs and sends to storage
- It masks the faults using a table of the processes.
- It detect communication failures

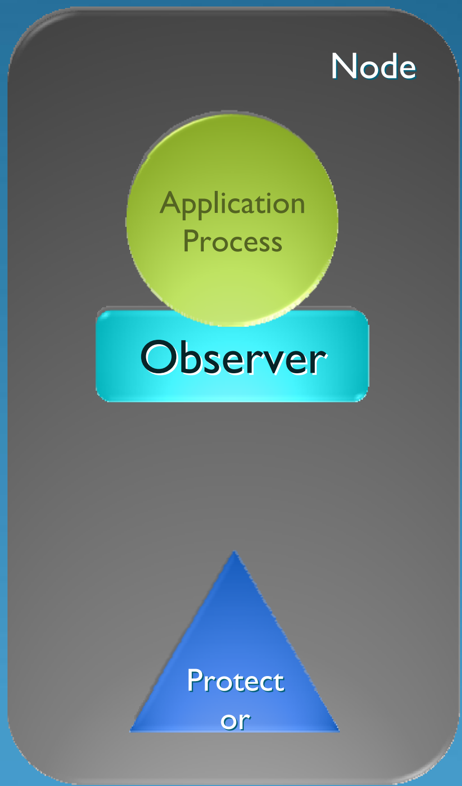


- It performs fault detection tasks using a heartbeat/watchdog scheme
- It stores the redundancy data (checkpoints and logs)
- It starts the recovery process

caos

The RADIC Architecture

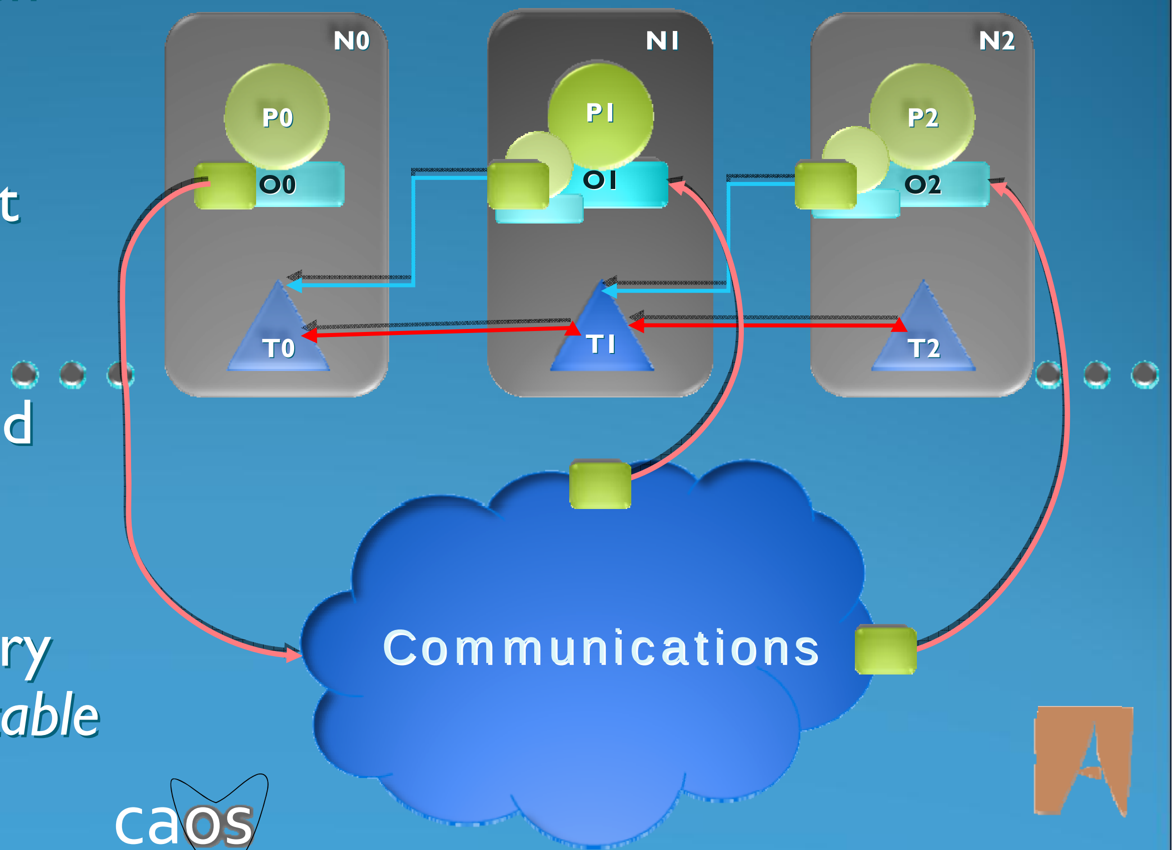
- RADIC in Practice



The RADIC Architecture

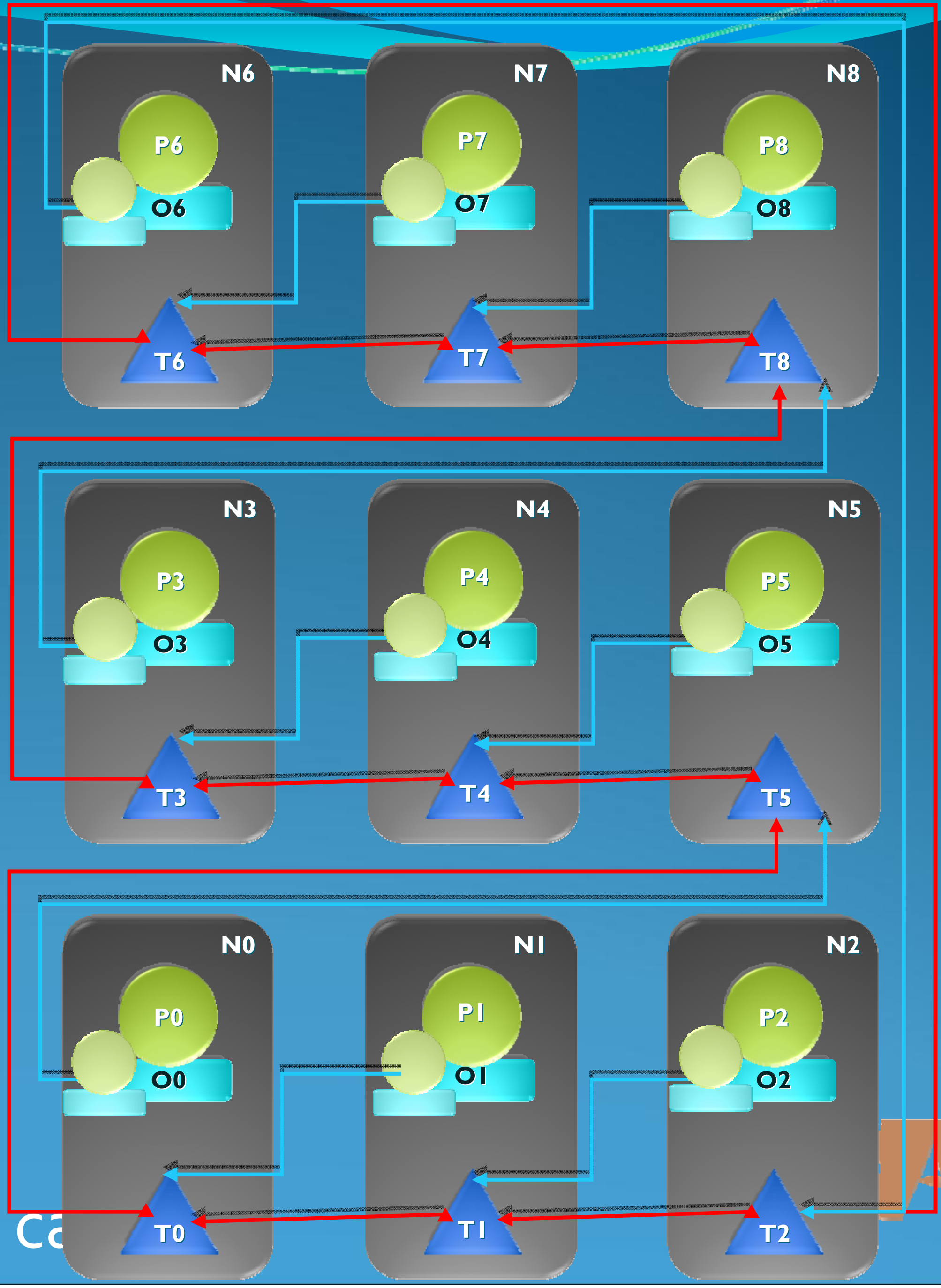
- RADIC functioning
  - Protectors establish the heartbeat/watchdog
  - Observers connect with Protectors
    - It sends checkpoints and event logs
    - It manages the message delivery using the *radictable*

RADICTABLE		
PID	Address	Protector
0	Node 0	Node 8
1	Node 1	Node 0
...	...	...



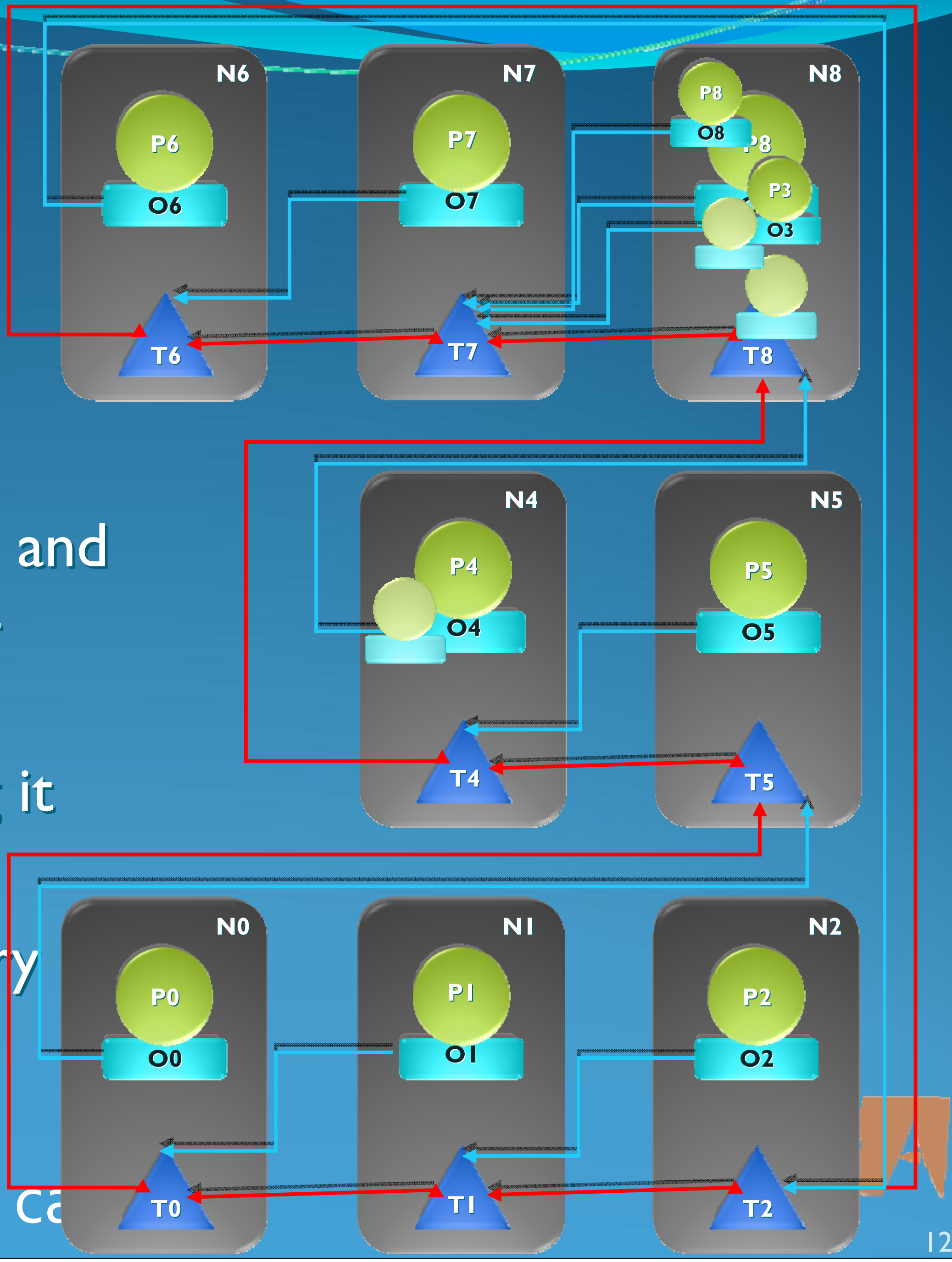
The RADIC Architecture

- The recovery process
  - N3 fails



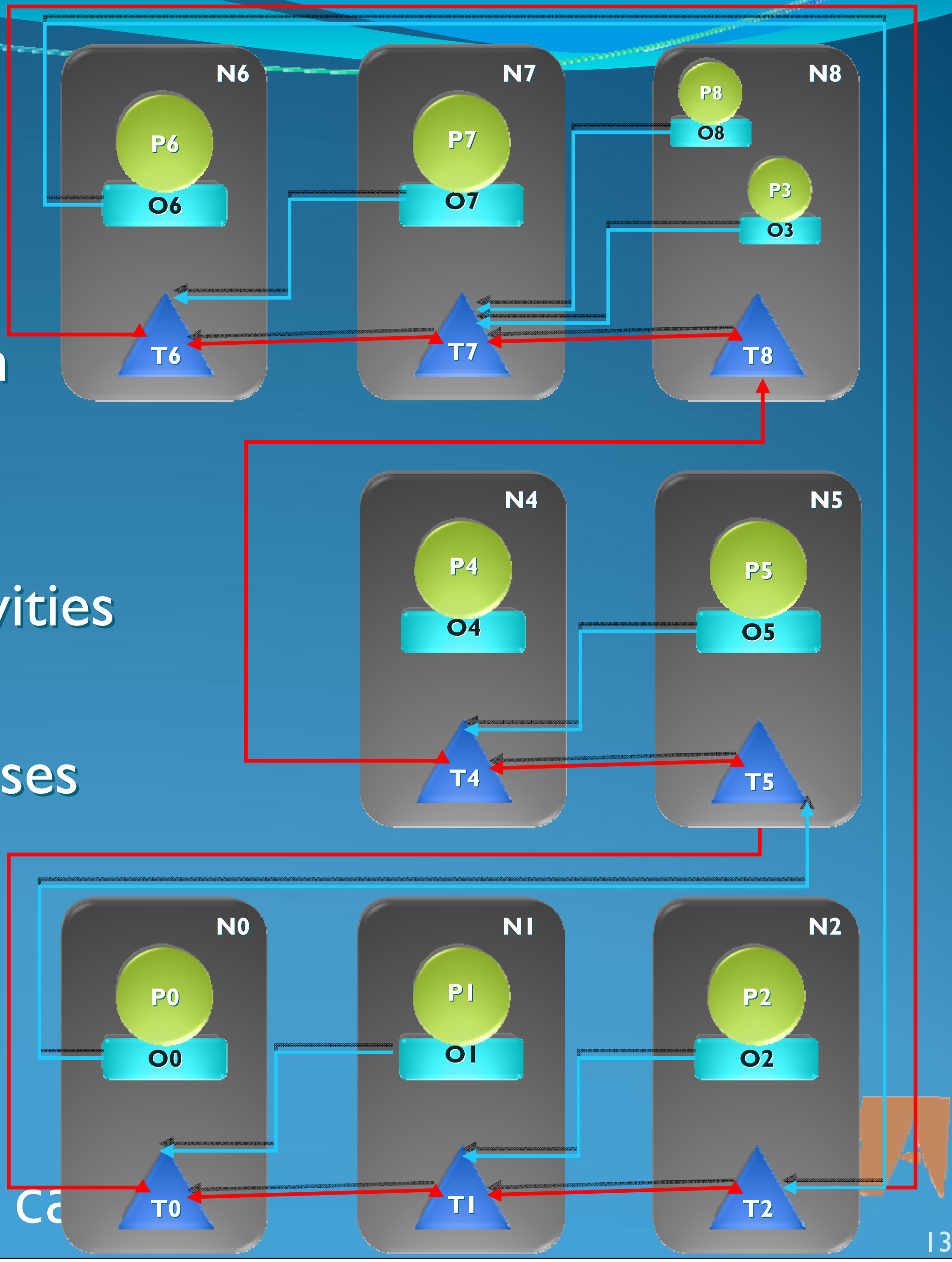
The RADIC Architecture

- The recovery process
  - N3 fails
  - T8 detects the faults and waits the T4 and O4 connection
  - T8 launches P3 using it checkpoint
  - O3 start the recovery process of P3



Recovery Side-Effect

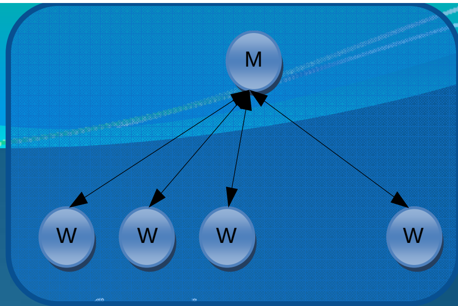
- The sytem configuration was changed
  - Processes P3 and P8 slowdown their activities
  - Protector T7 now protects two processes
  - The memory usage in N8 rises



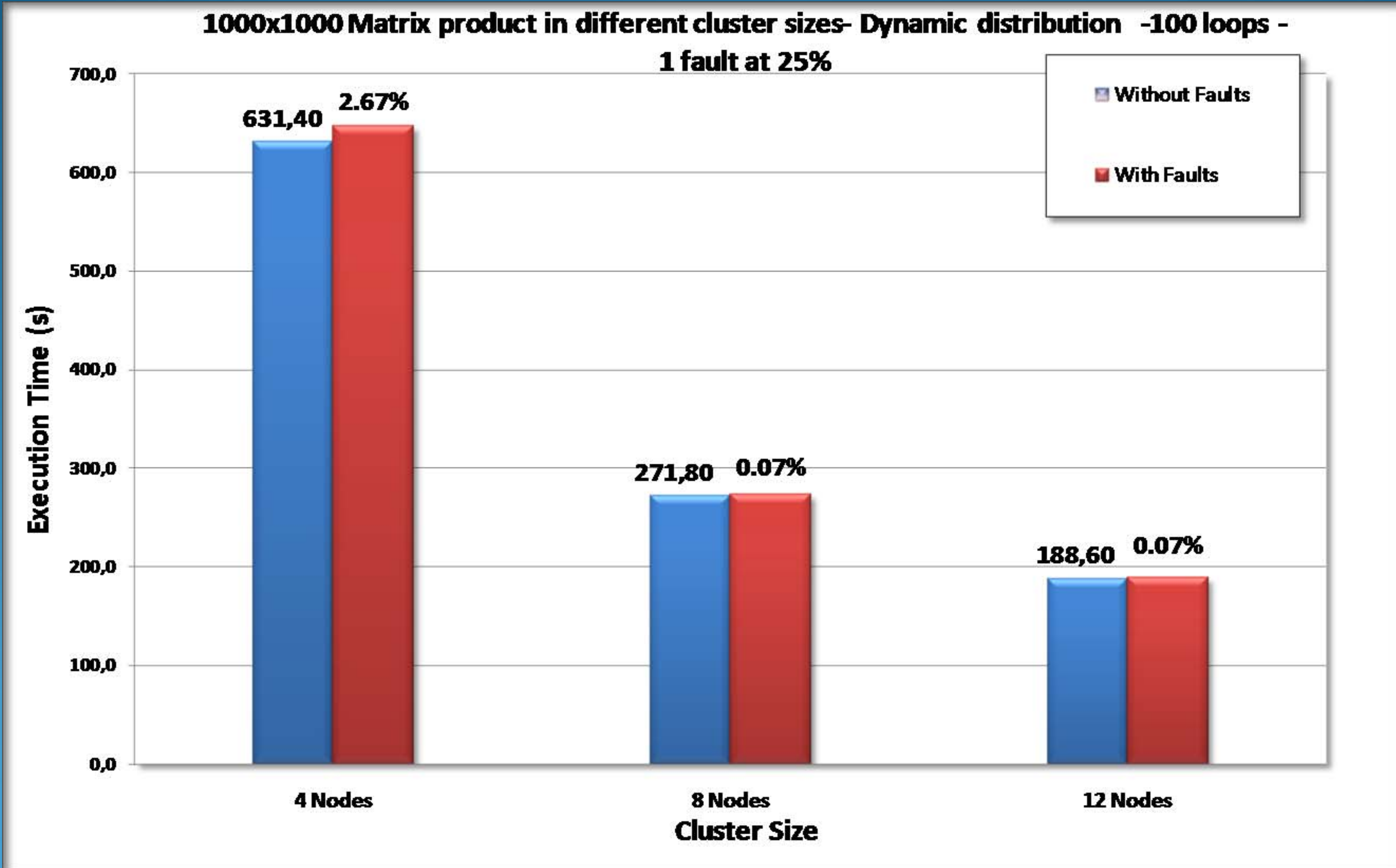
Recovery Side-Effect

- In order to quantify these side-effects facing different scenarios, we performed some experiments:
  - We applied three variations of a matrix product program:
    - Dynamic M/W – Loosely coupled
    - SPMD – High coupling in all processes
    - Static M/W – High dependency

Recovery Side-Effect

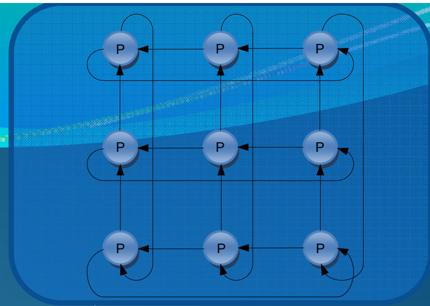


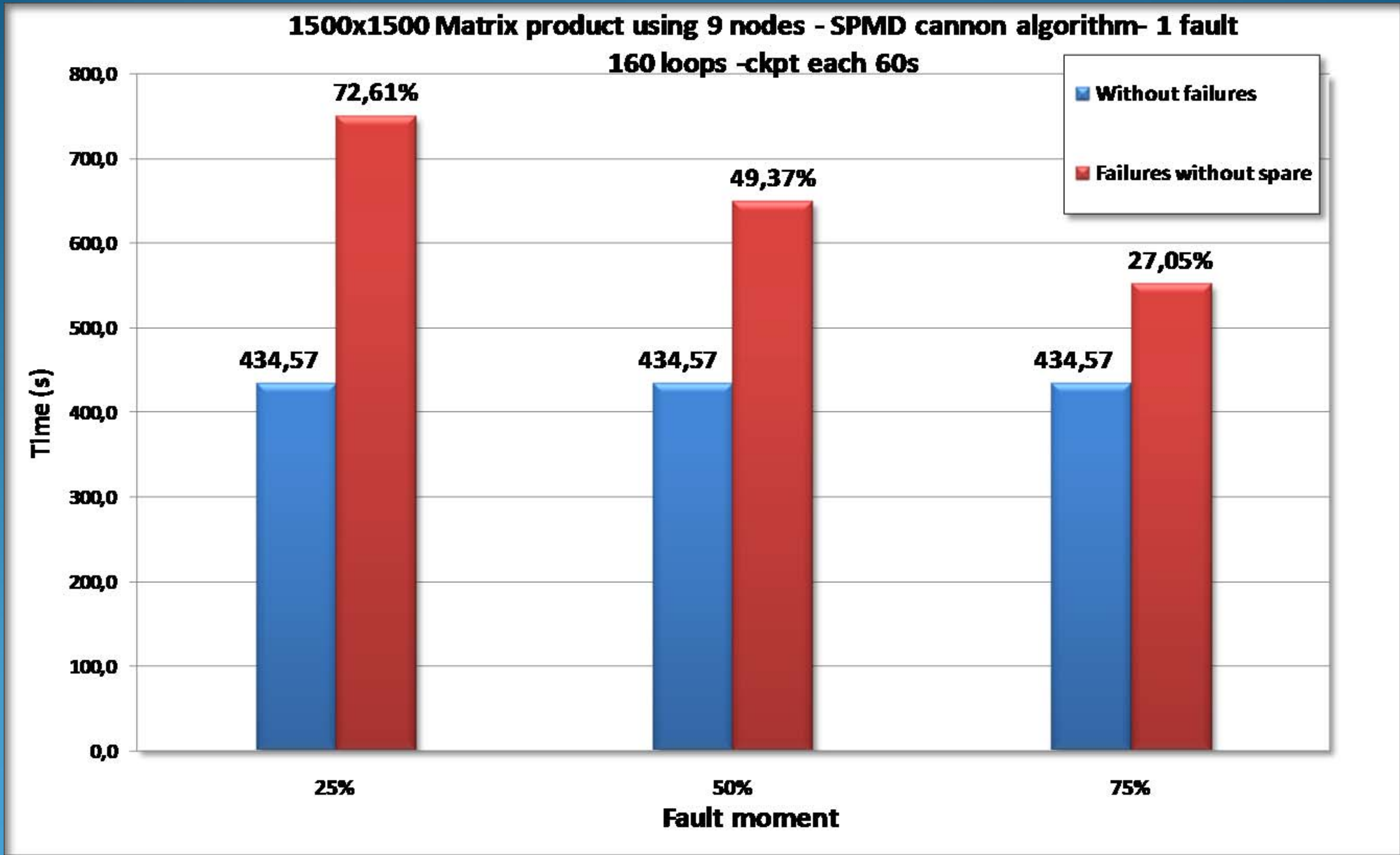
Dynamic M/W – different number of nodes



Recovery Side-Effect

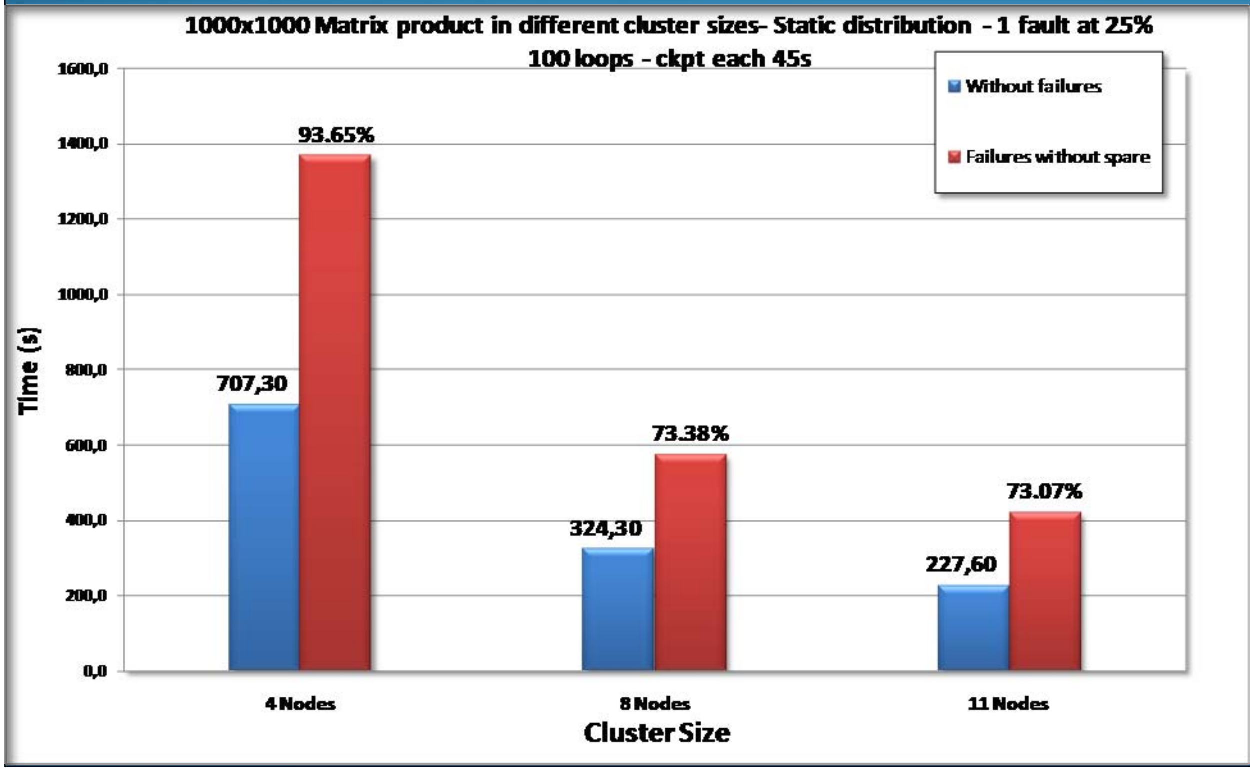
SPMD – one fault in different moments





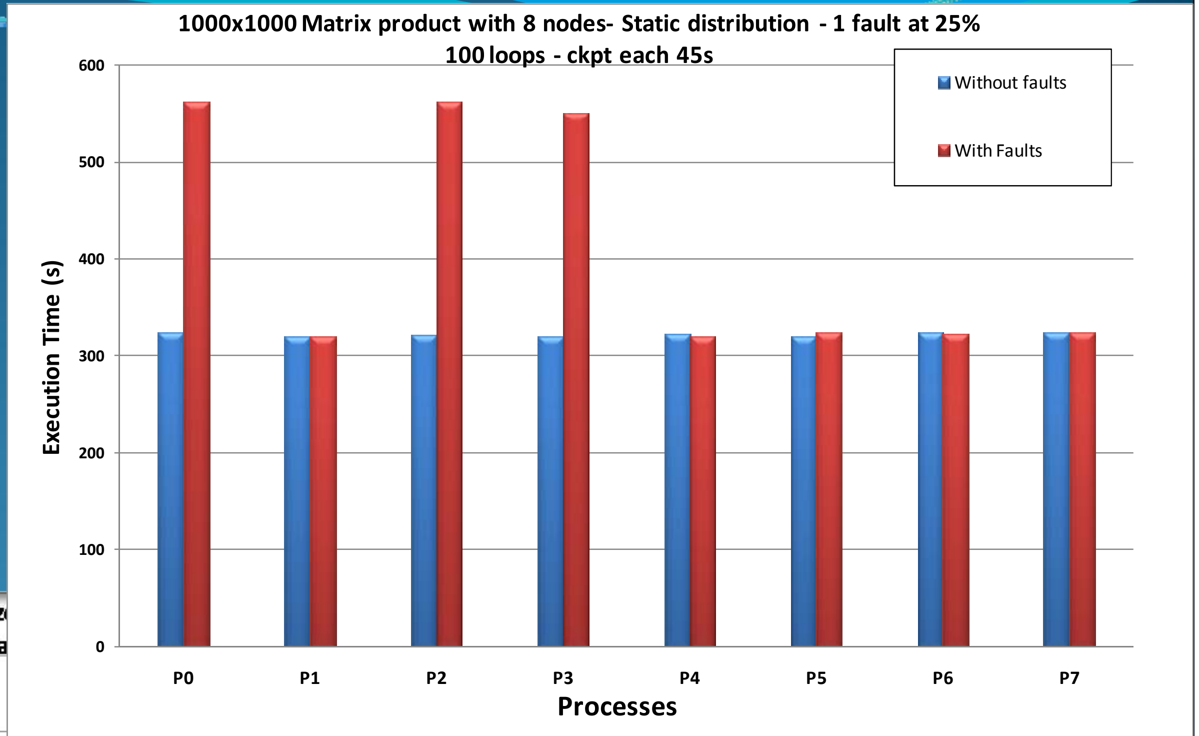
Recovery Side-Effect

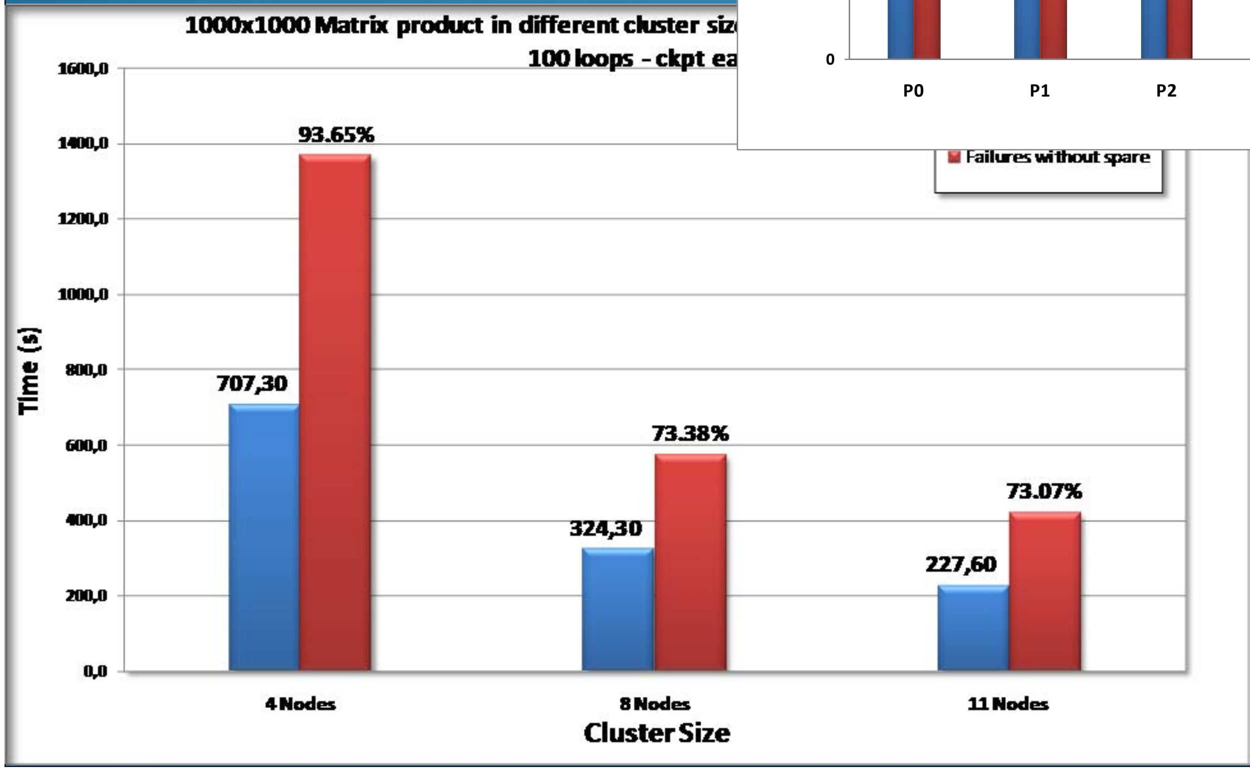
Static M/W



Recovery Side-Effect

Static M/W





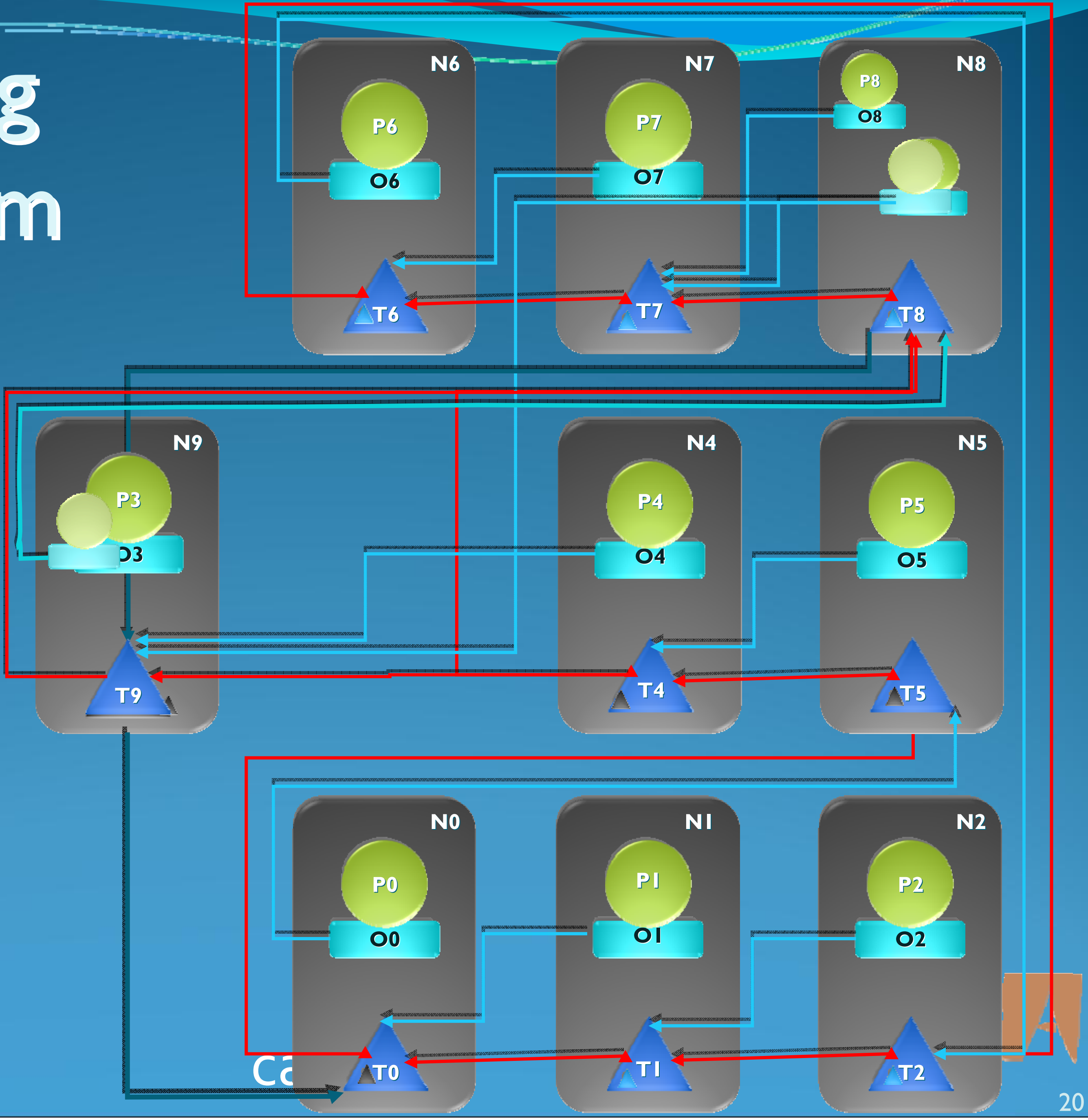
Protecting the System

- We incorporate a new protection level in RADIC using a dynamic redundancy functionality, allowing to mitigate or to avoid the recovery side-effects
  - Restoring the system configuration
  - Avoiding system configuration changes
  - Allowing to prevent faults
    - It incorporates a transparent management of **spare nodes**
    - It does not mantain any central information about the spare nodes
    - The spare nodes use does not affect the scalability

caos

Protecting the System

- Restoring the configuration

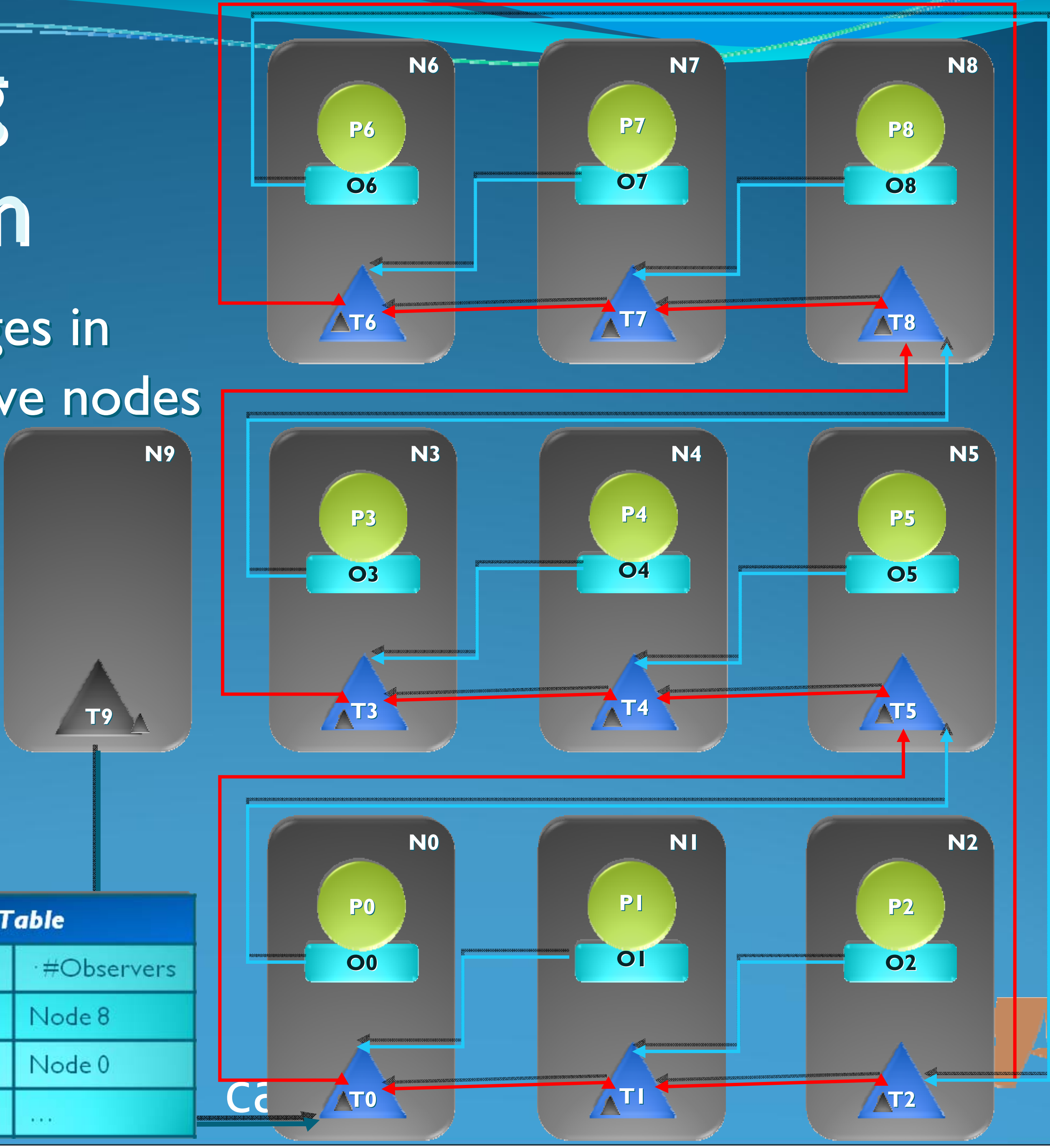


- Insert a replacement node running a protector in spare mode
- The Protector announces itself by message forwarding
- T8 requests the new node because it is overloaded
- The new node incorporates the protecting scheme
- T8 tells to P3 to take its checkpoint in N9
- T8 commands T9 to recover P3
- The old P3 suicides
- From now, the message forwarding informs the new spare state

Protecting the System

- Avoiding changes in number of active nodes

- Insert a new node running a Protector in spare mode (waiting a request)
- The spare Protector announces itself by message forwarding
- Each Protector adds the new spare in its *spare table* and forward it
- This procedure … when a protect… receives a repe… information



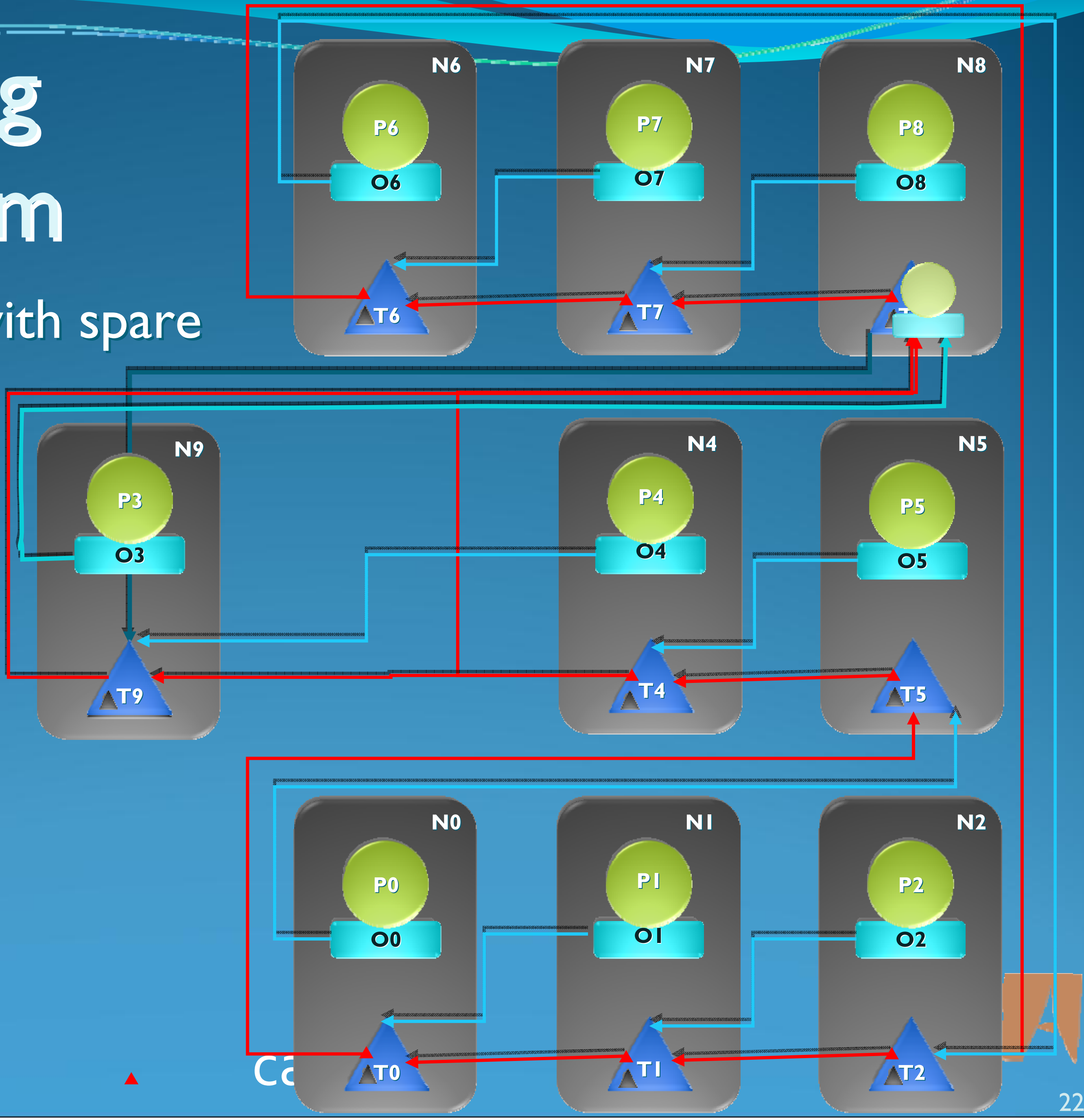
Spare Table

SID	Address	#Observers
0	Node 0	Node 8
1	Node 1	Node 0
…	…	…

Protecting the System

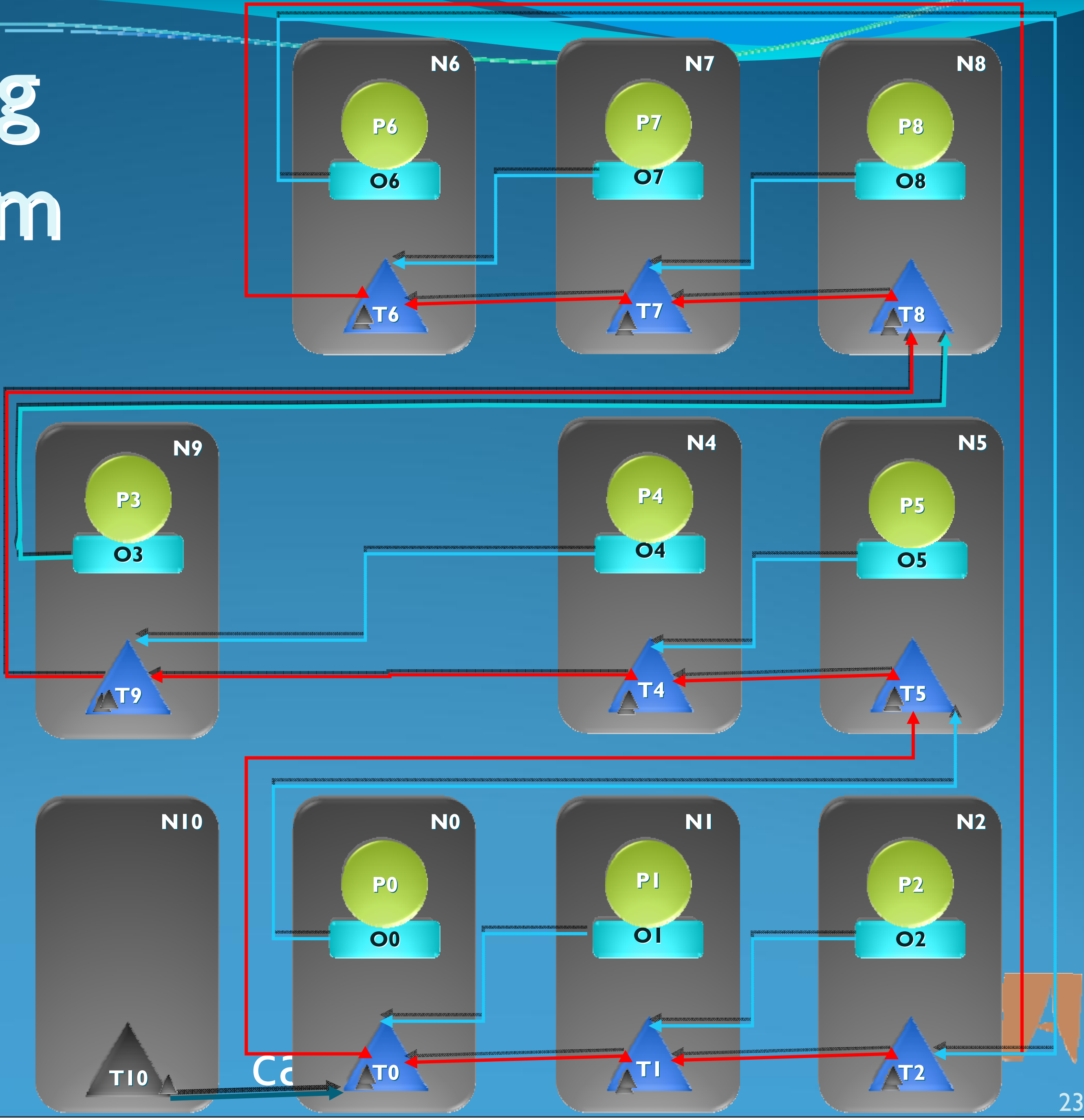
- Recovering with spare node

- T8 detects a fault and connects to spare, activating it
- It queries about its state (if still is a spare)
- It commands the spare to join to the protecting scheme
- O4 also connects the spare
- T8 sends the check-point and log to spare
- T8 commands to recover P3



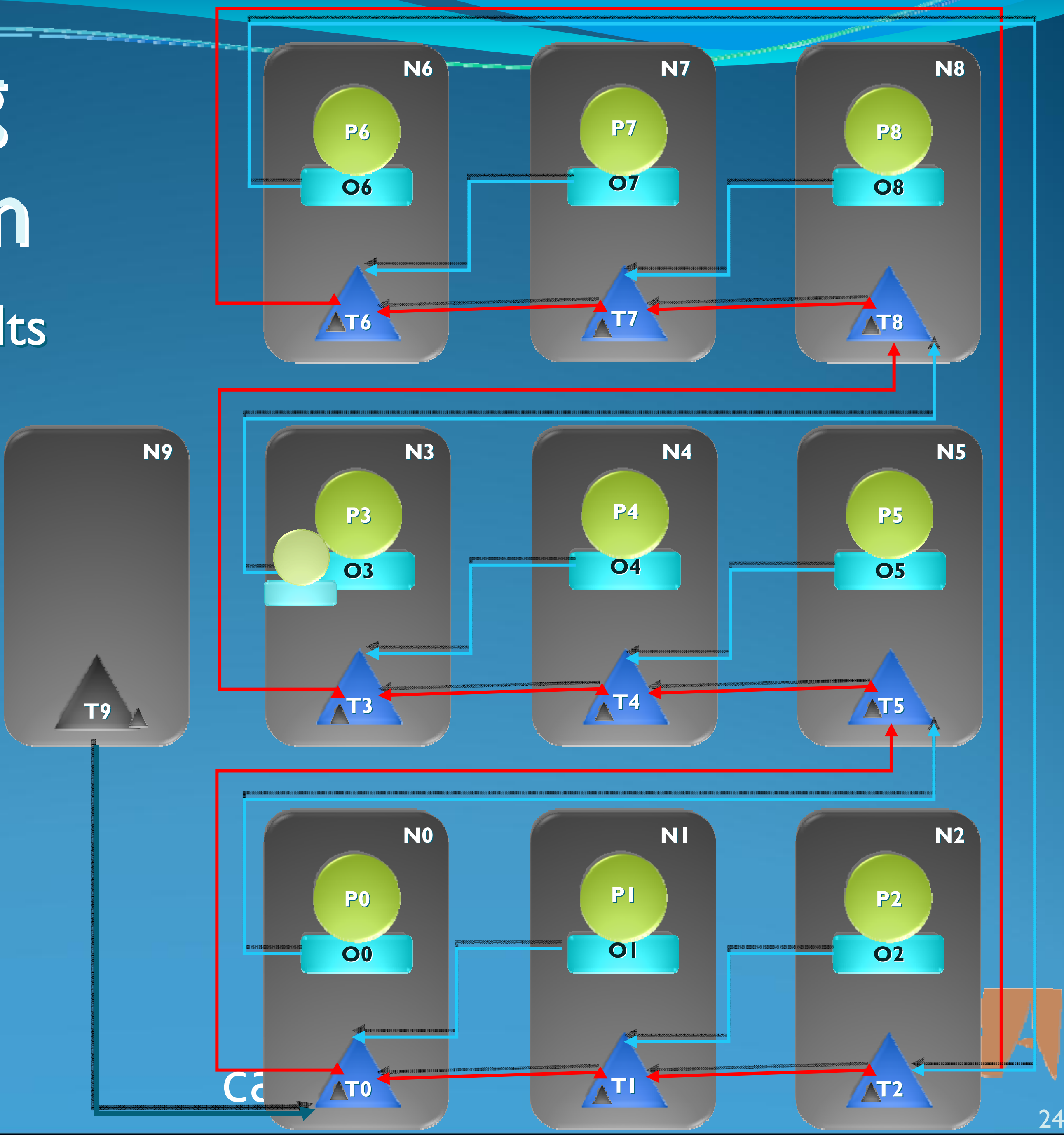
Protecting the System

- Restoring the configuration



Protecting the System

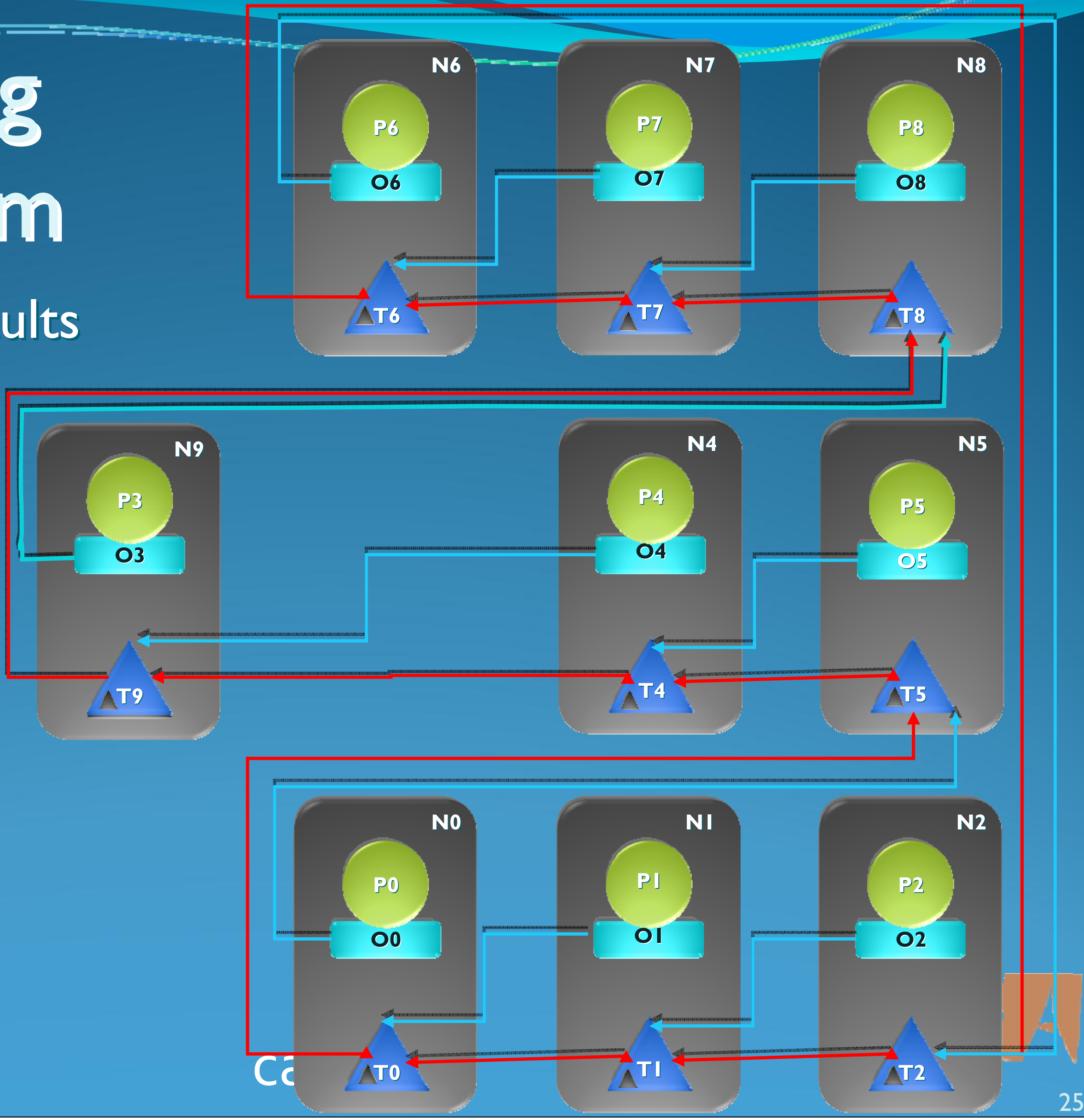
- Preventing Faults



- N3 is a fault probable node
- Insert a spare node
- Oportuniscally inject a fault in N3

Protecting the System

- Preventing Faults



- N3 is a fault probable node
- Insert a spare node
- Oportuniscally inject a fault in N3
- P3 will recover in N9

Implementation

- We adapted the RADIC prototype (RADICMPI) in order to incorporate the Flexible Dynamic Redundancy
  - Creation of management functions
  - Definition of a new protocol to communicate with spares
  - Increment of communication between observers and local protector
  - Changes in the RADIC fault masking procedure including search in the spare nodes

caos

Implementation

- We also implemented a set of MPI non-blocking functions, allowing to run more kind of application
  - We take care about the fault tolerance issues
  - We had to change the message management kernel of RADICMPI to deal with asynchronous communications
  - Other issues are solved too:
  - We changed the original message log approach of RADICMPI to an event log approach in order to assure the determinism of the recovery process.

caos

Experiments Methodology

- We conduced two kind of experiments:
  - Validation
    - Spare adding task
    - Recovery task using spare

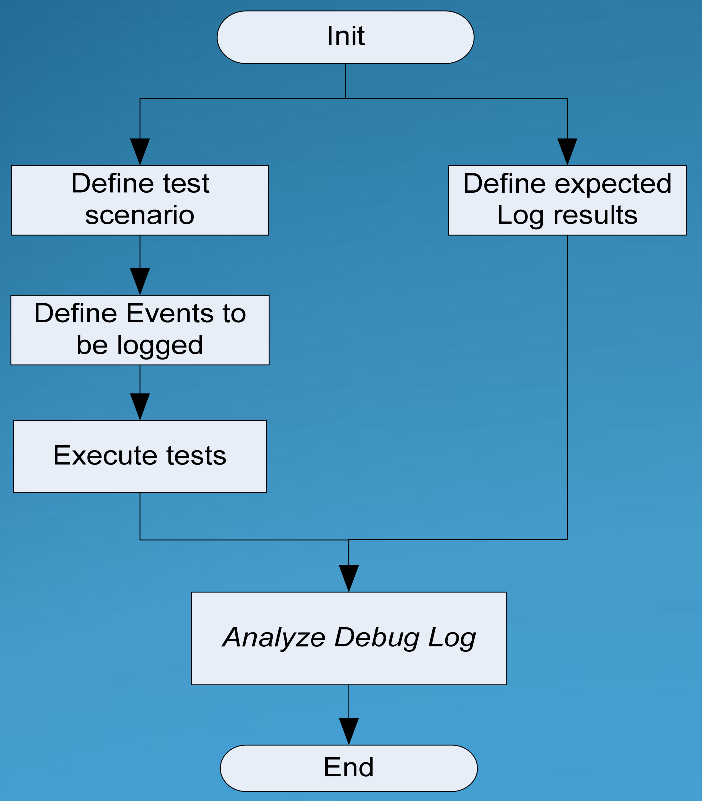
    { Using the RADICMPI Debug Log / Running a Ping-Pong program }
  - Evaluation
    - According with the fault moment
    - According with the number of nodes
    - Throughput behavior in continuous running applications

    {
    - Matrix Product – Static
    - Matrix Product – Dynamic
    - Matrix Product – SPMD
    - N-Body simulation
    }

caos

Experiments Methodology

- Validation metodology



Col	Field Name	Description
1	Event Id	Identifies the Event Type
2	Event time	Elapsed time since startup
3	Process ID	Rank of Observer/Protector
4	Function name	Internal Function triggering this event
5	Event	Description of the event

caos

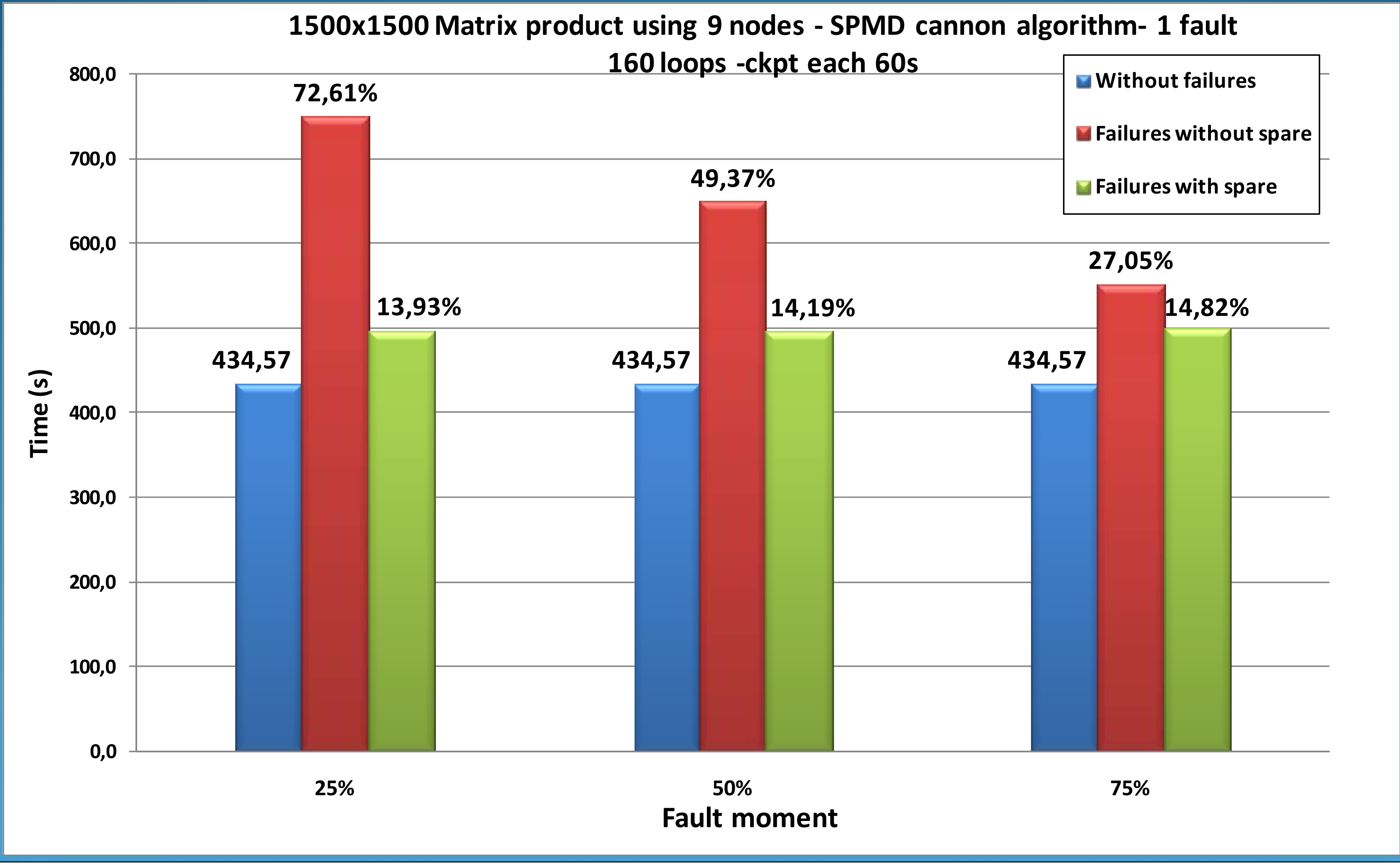
Experiment Design

- **Evaluating according the fault moment**
  - The experiments were conduced in a twelve node cluster
  - We executed a product of two 1500x1500 matrix using the SPMD paradigm over 9 nodes and a product of two 1000x1000 matrixes using a static distribution over 11 nodes
  - We injected one fault at 25%, 50% and 75%
  - We measured the execution times in three situations
    - Fault-free
    - Without spare
    - With spare

caos

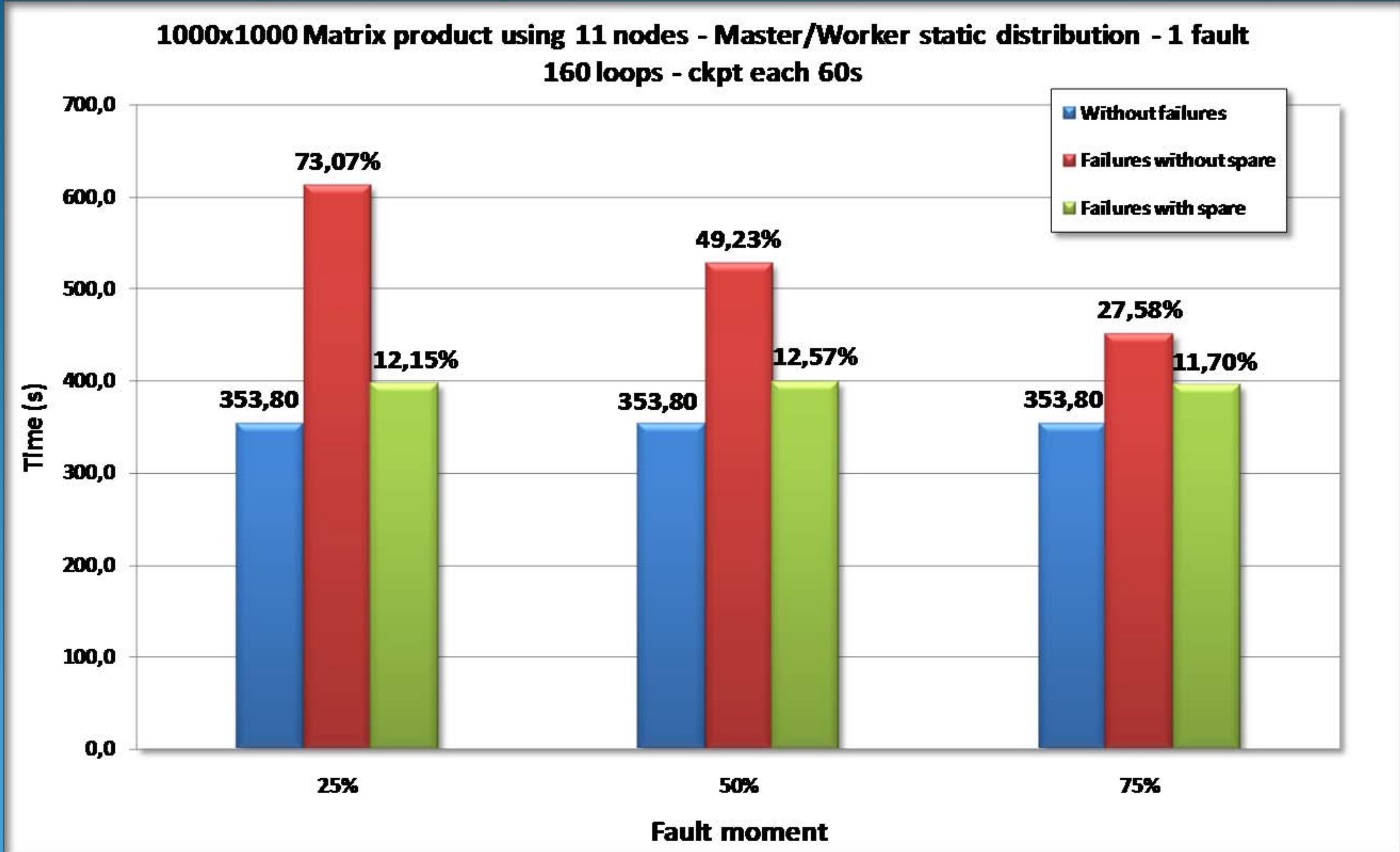
Experimental Results

- Evaluating according with the fault moment



Experimental Results

- Evaluating according with the fault moment



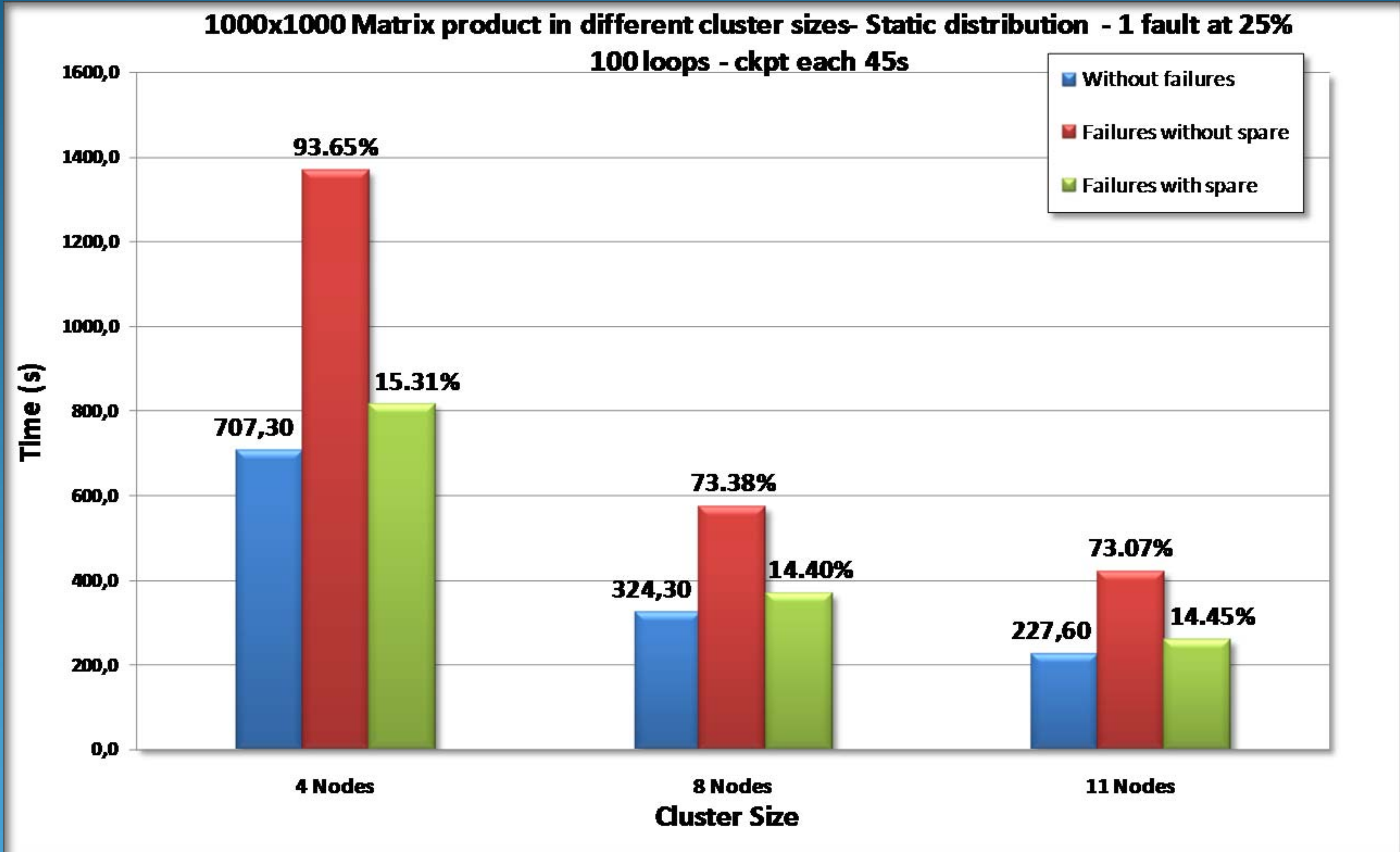
Experiment Design

- **Evaluating according the number of nodes**
  - The experiments were conduced in a twelve node cluster
  - We executed a product of two 1000x1000 matrix using a dynamic distribution and a product of two 1000x1000 matrixes using a static distribution
  - We injected one fault at 25%
  - We ran the programs over three cluster sizes
    - 4 nodes + spare
    - 8 nodes + spare
    - 11 nodes + spare
  - We measured the execution times in three situations
    - Fault-free
    - Without spare
    - With spare

caos

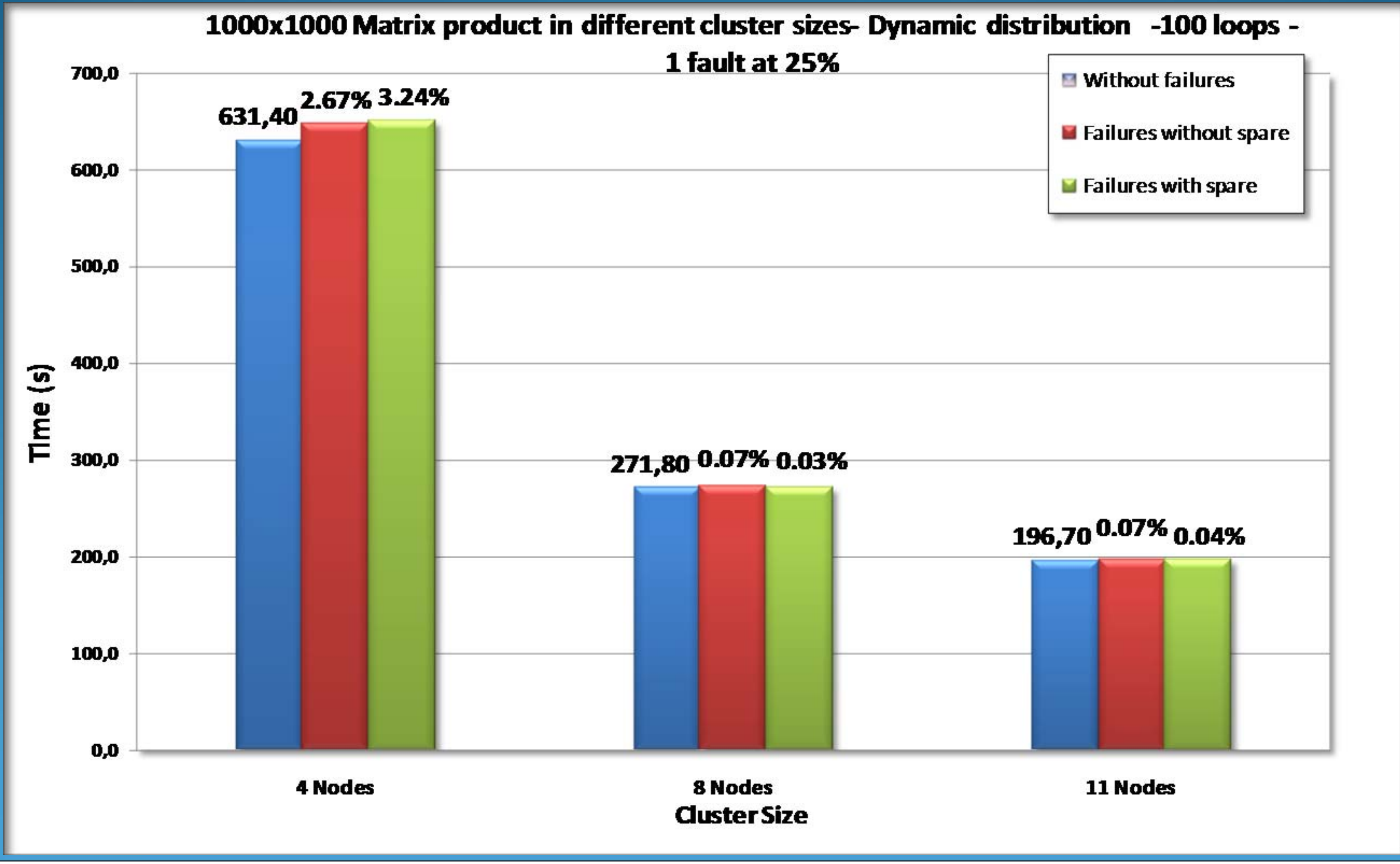
Experimental Results

- Evaluating according with the number of nodes



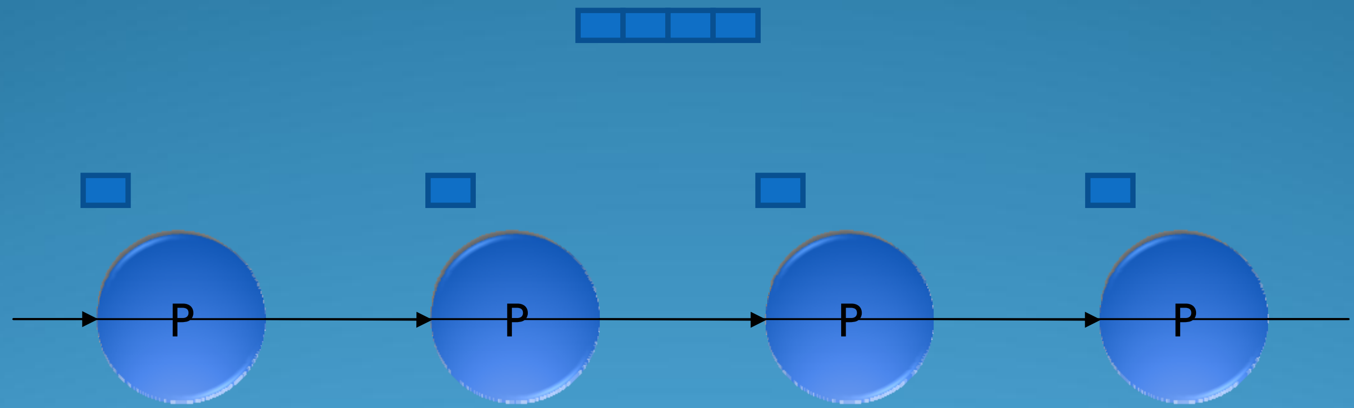
Experimental Results

- Evaluating according with the number of nodes



Experimental Results

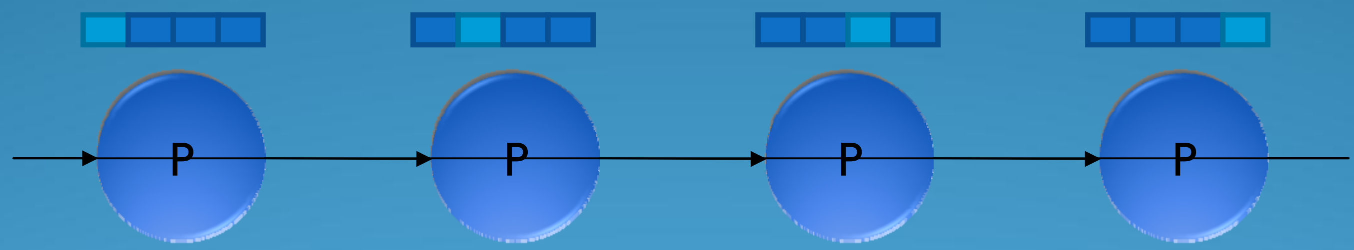
- We implemented a N-Body simulation in order to study the behavior of our solution over continuous running applications



caos

Experimental Results

- We implemented a N-Body simulation in order to study the behavior of our solution over continuous running applications



caos

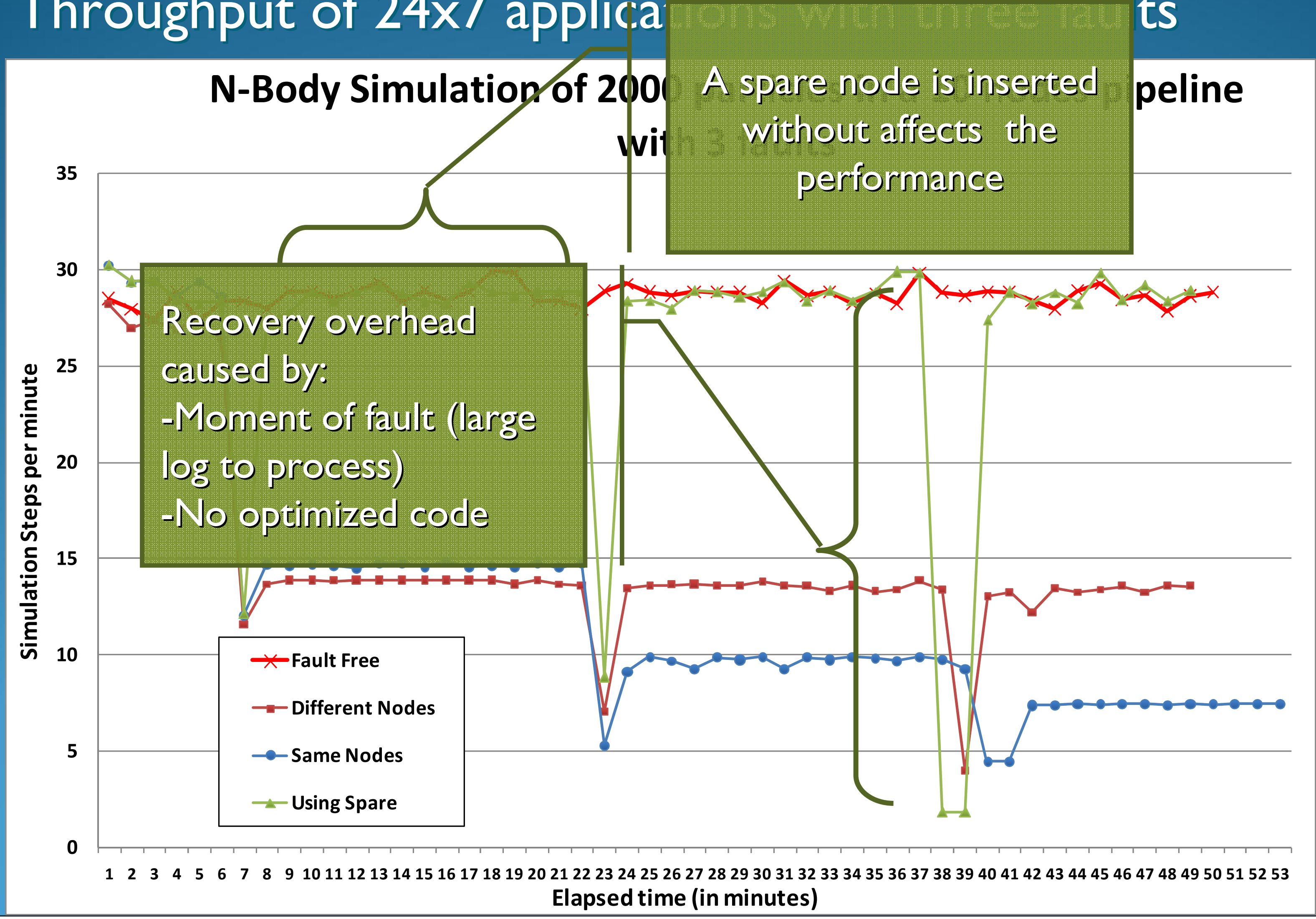
Experiment Design

- **Throughput of 24x7 applications**
  - We executed this application simulating 2000 particles in a ten node pipeline
  - We measured the throughput (in simulation steps per minute) in four scenarios
    - Fault-free
    - Injecting three faults without spare:
      - Processes recovering in different nodes
      - Process recovering in same node without spare
    - Injecting three faults using two initial spares and re-insert a "repaired" one after the first fault.

caos

Experimental Results

- Throughput of 24x7 applications with three faults



Conclusions

- We implemented a dynamic redundancy functionality that avoids or mitigates the recovery side-effects
- This functionality is flexible because
  - It allows system configuration restablishment by dynamic insertion of spare nodes - **RESTORING**
  - It allows a transparent management of the spare nodes use - **AVOIDING**
  - It incorporates a **maintenance** feature that also allows to prevent failures by replacing fault-probable nodes – **PREVENTING**
  - Configurable, allowing from 0 to N number of spares
- We extend the RADICMPI functionality, implementing a set of MPI non-blocking functions

caos

Conclusions

- Our results show the benefits of the dynamic redundancy solution in different scenarios.
- The results also show a strong dependency between the recovery side-effects and the application characteristics and how we can adapt to each one.

caos

Future Work

- To study the spare nodes allocating facing factors like degradation level acceptable or memory limits of a node
- To integrate a fault prediction mechanism in the maintenance feature
- To continue expanding the RADICMPI functionality

caos

Open Lines

- To investigate how adapt and use RADIC II with new HPC trends like the clusters of multicore computers.
- To achieve a RADIC II analytical model, allowing to determine better parameter values.
- To develop a RADIC II simulator allowing to assess its behavior in large clusters
- To incorporate other features towards an autonomic fault tolerant system.

Thank you