

# Autonomic Features in GCM

**M. Aldinucci, S. Campa, M. Danelutto**  
Dept. of Computer Science, University of Pisa

**P. Dazzi, D. Laforenza, N. Tonellotto**  
Information Science and Technologies Institute, ISTI-CNR

Programming model Institute Plenary Meeting  
Paris, France, Jan 17, 2008



# Outline

## Motivation

- GCM key points
- Autonomic Computing Paradigm

## Implementation

- Autonomic Computing in GCM
- Implementing Autonomic Features
- Managing Autonomic Features

## Experiments

- GridCOMP use case: Wing Design (from Grid Systems)
- Performance evaluation

## Conclusions

- Something to discuss





# GCM key points

## Hierarchical Model

- expressiveness
- structured composition

## Communication Structures

- interaction patterns
- collective communications

## Communication Semantics

- stream, RPC, file transfer, events

## Functional/Non-Functional Adaptivity

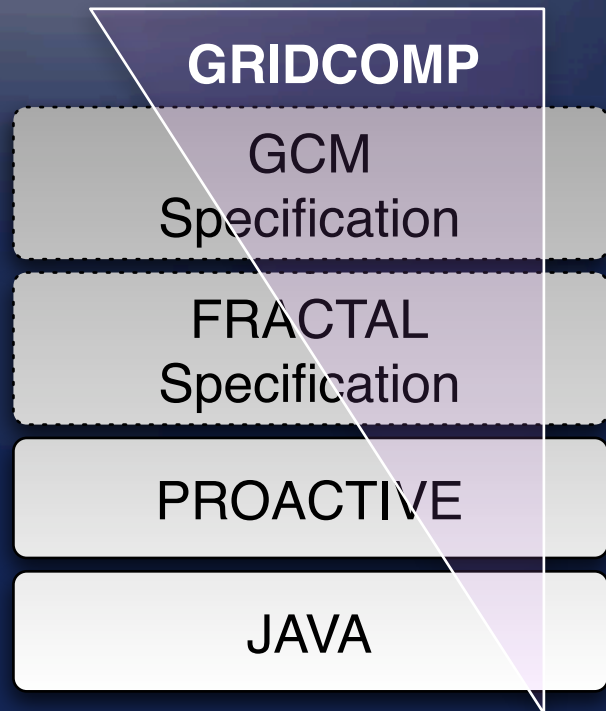
- QoS specification/enforcement

## Autonomic Management

- Self-\* Properties

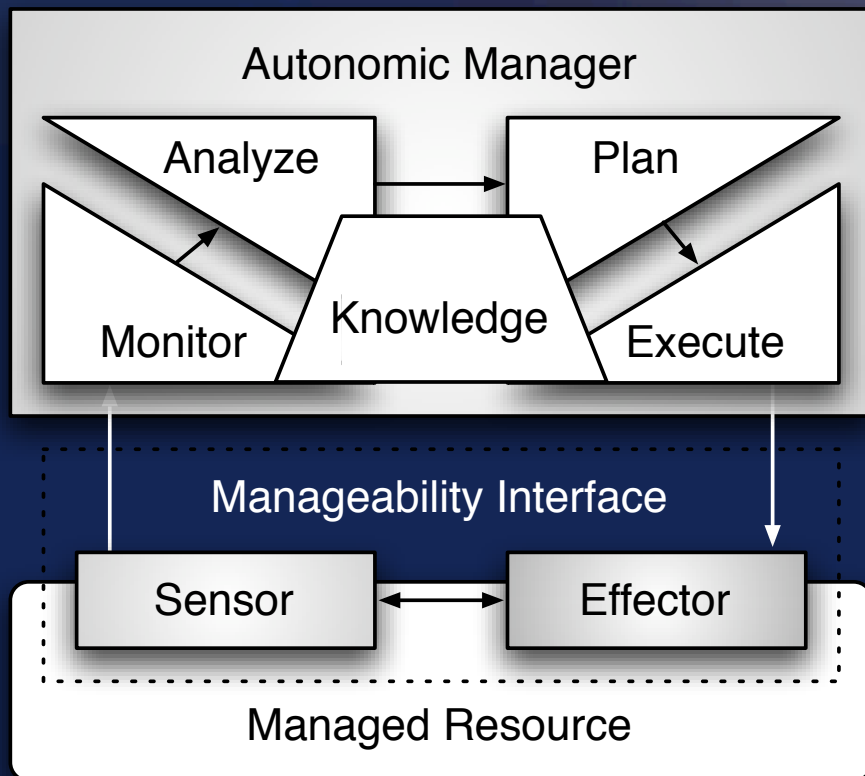
## Distributed Deployment

- Heterogeneous Systems





# Autonomic Computing Paradigm (I)



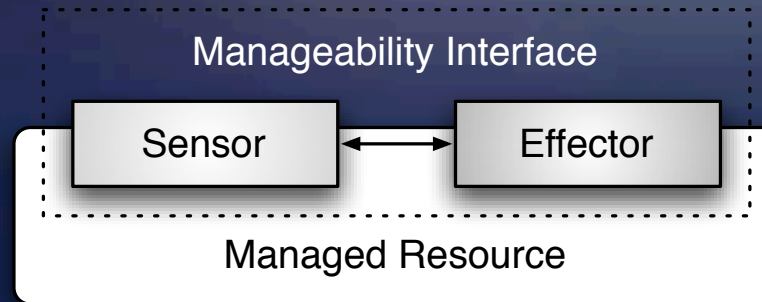
Autonomic systems will maintain and adjust their operation in the face of changing components, workloads, demands, and external conditions and in the face of hardware or software failures, both innocent and malicious.

Four fundamental aspects:

- \* self-configuring
- \* self-healing
- \* self-optimizing
- \* self-protecting



## Autonomic Computing Paradigm (II)

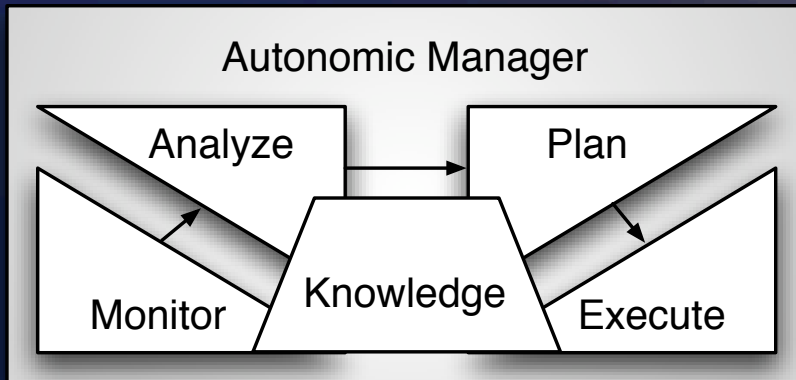


The sensors provide mechanisms to collect information about the state transitions of the managed resource.

The effectors are mechanisms to change the state (configuration) of the managed resource.



# Autonomic Computing Paradigm (III)



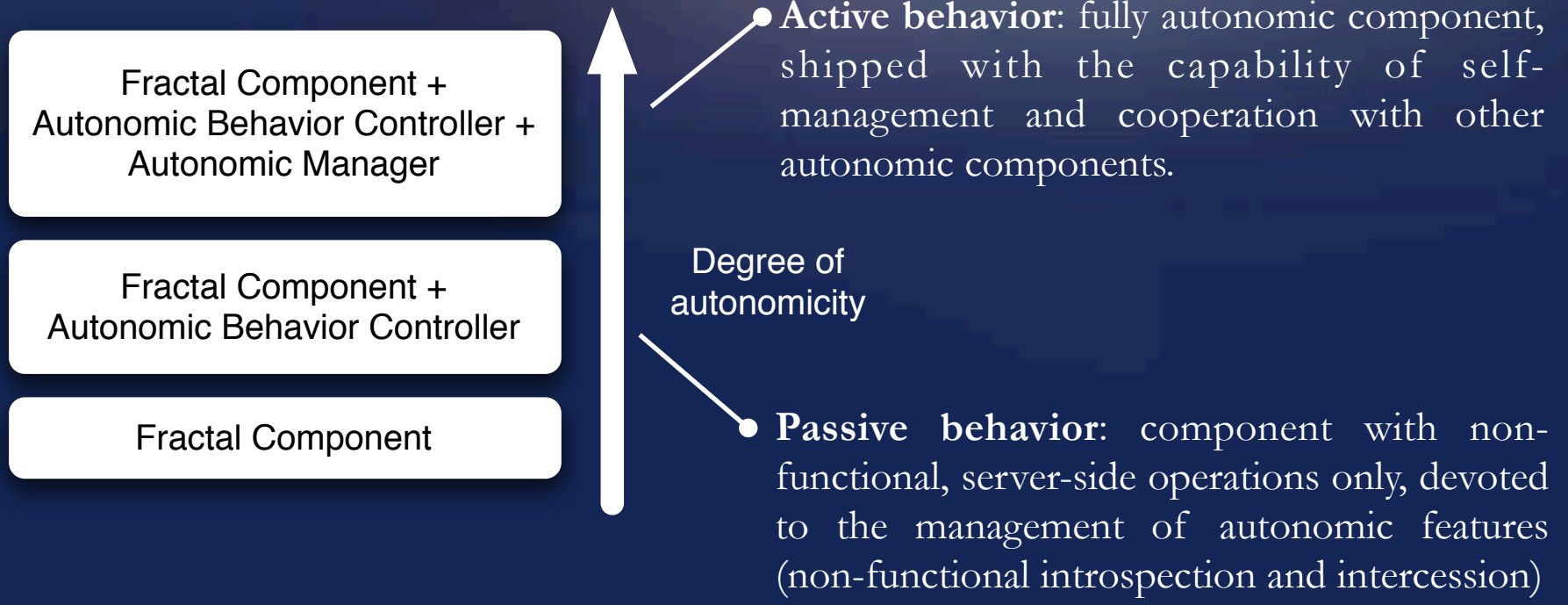
The autonomic manager is an entity that manages other software or hardware entities using a control loop. The control loop of the autonomic manager includes monitor, analyze, plan and execute functions.

- **monitor:** collect execution stats: machine load, service time, input/output queues lengths, ...
- **analyze:** instantiate performance models with monitored data, detect broken contract, in and in the case try to detect the cause of the problem
- **plan:** select a (predefined or user defined) strategy to re-convey the contract to validity.
- **execute:** leverage on mechanism to apply the plan



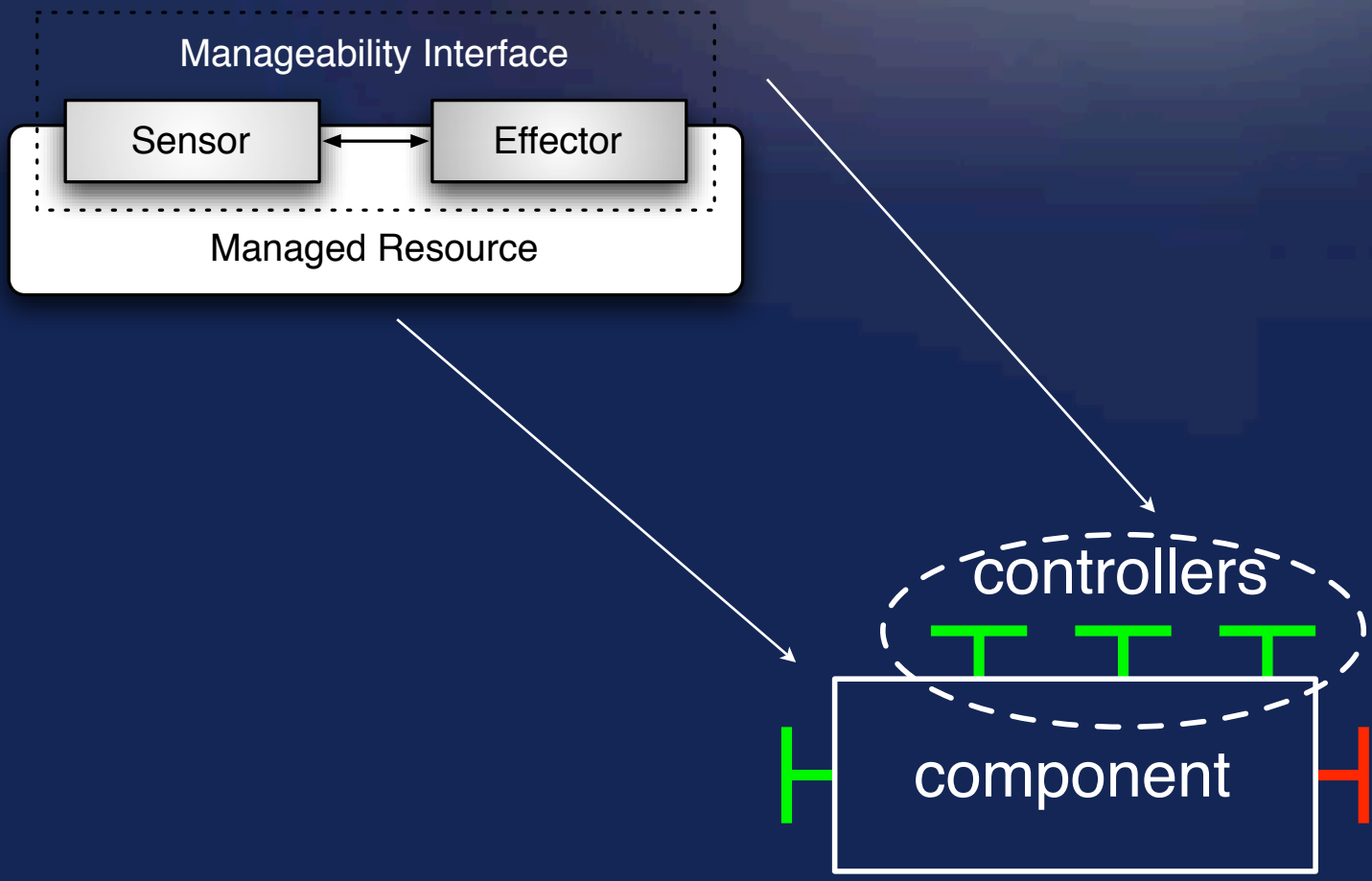


# Autonomic Computing in GCM (I)



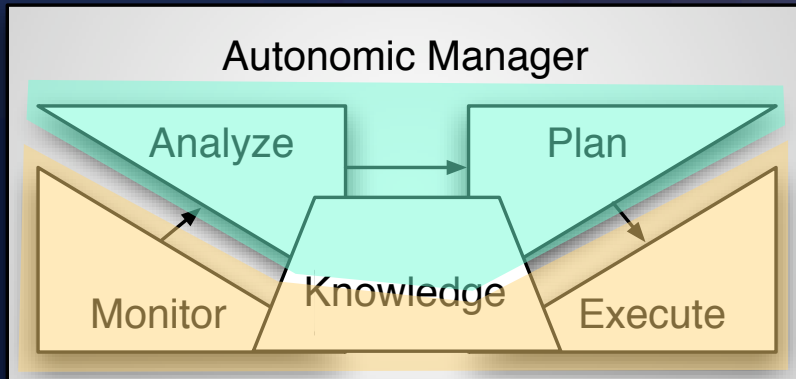


# Autonomic Computing in GCM (II)





# Autonomic Computing in GCM (III)



```
interface AutonomicServerManager
{
    any commitContract(String qosContract);
}

interface AutonomicClientManager
{
    any raiseViolation(any violationId);
}
```

```
interface AutonomicBehaviorController
{
    String[] listAutonomicOperations();
    any execOperation(String op, any ...);
}
```





# Implementing Autonomic Features (I)

## Autonomic Behavior Controller (ABC)

- is a controller (server interface)
- exposes a set of autonomic operations to dynamically change the component functional and non-functional aspects
- does not implement any optimization strategy
- imposes non-functional constraints on the underlying component

## Autonomic Operation

- in charge of the monitoring and executing phases of the autonomic computing loop.
- exploits basic Fractal controllers and implementation-dependent controllers
- exhibits a parametric but deterministic behavior
- a reduced set of operation can be exploited to implement different behaviors

## Autonomic Manager (AM)

- responsible for a strategy to enforce at runtime a particular QoS contract
- logically part of the membrane, but with its own lifecycle





## Implementing Autonomic Features (II)

### Management is difficult...

- Application structure and performance change along time
- How to “describe” functional, non-functional features and their inter-relations?
- Component reuse is already a problem...
- Component reuse + runtime management = nightmare!

### ...but...

- Several applications share the same interaction structure...
- ... the same non-functional semantics (but with different functional semantics)
- ... the same performance objectives/strategy

### ...then...

- We can abstract parametric paradigms of component assemblies specialized to solve one or more management objectives



**Behavioral Skeleton**





# Managing Autonomic Features (I)

## Behavioral Skeleton Properties

- Expose a description of the component functional behavior
- Establish a parametric orchestration schema of inner components
- May carry constraints that inner components are required to comply with
- May carry a number of pre-defined strategies/plans to cope with a pre-defined management goal
- Carry an implementation of AM and ABC (with its own operations)

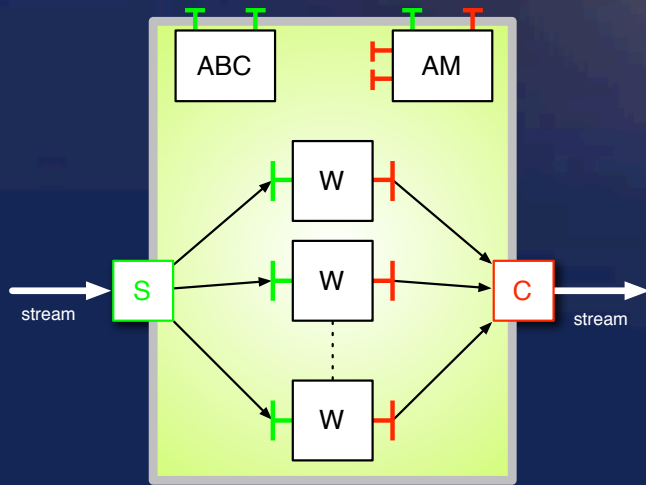
## Behavioral Skeleton Families

- Farm/Parameter Sweep (self-optimizing)
- Data-Parallel (self-configuring)
- Active/Passive Replication (self-healing)
- Facade (self-protecting)





# Managing Autonomic Features (II)



## Farm

S = unicast, C = from any,  
 W = stateless inner component

## Data-parallel

S = scatter, C = gather,  
 W = stateless inner component

## Active Replication

S = broadcast, C = get one,  
 W = stateless inner component

## GCM implementation how-to:

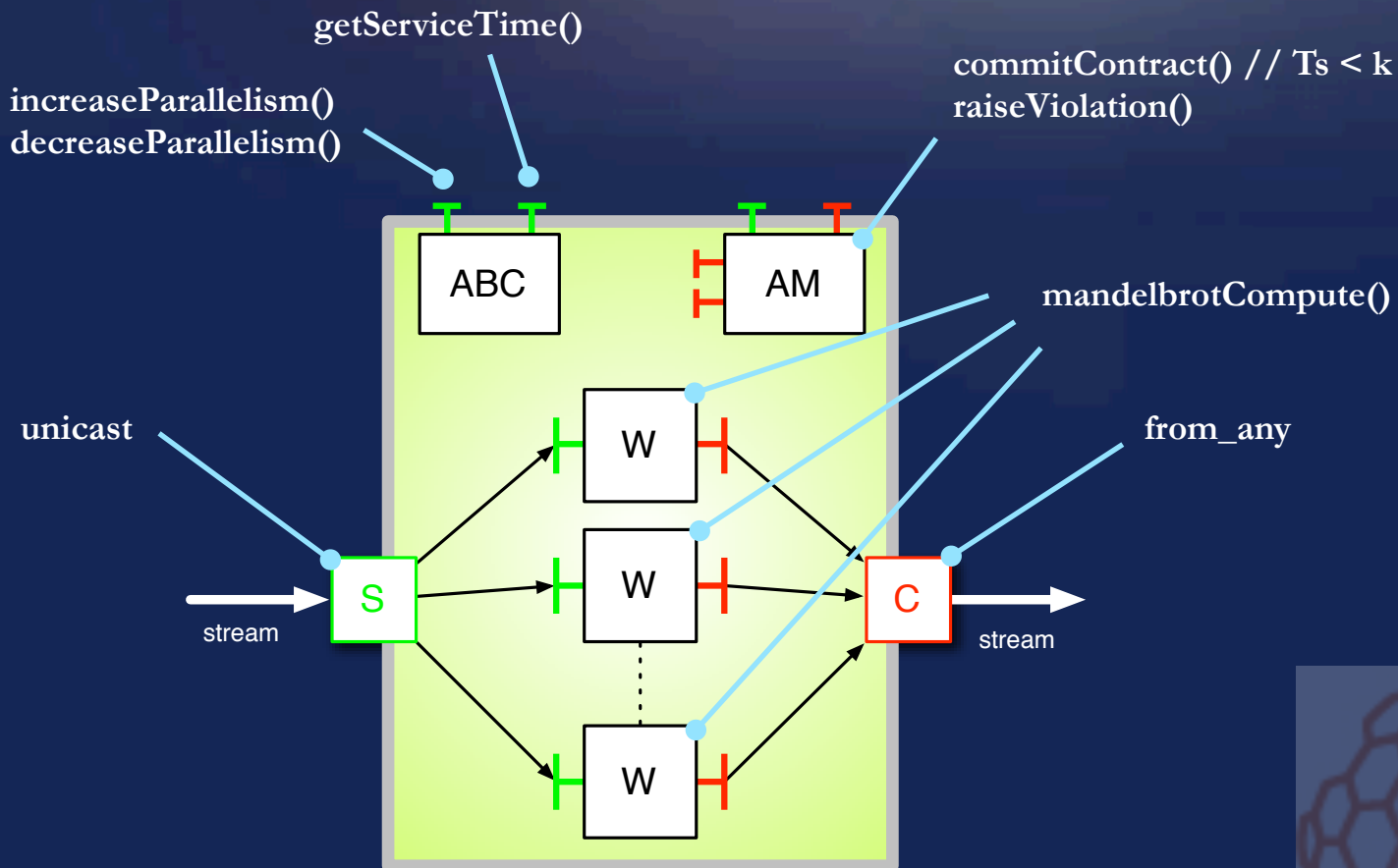
1. Choose a behavioral skeleton (ABC operations are chosen accordingly)
2. Choose an inner component (compliant to be-ske constraints)
3. Choose the behavior of ports (compliant to be-ske specifications/constraints)
4. Program the AM according to ABC operations and user goals
5. Wire the components, run the application, trigger adaptation





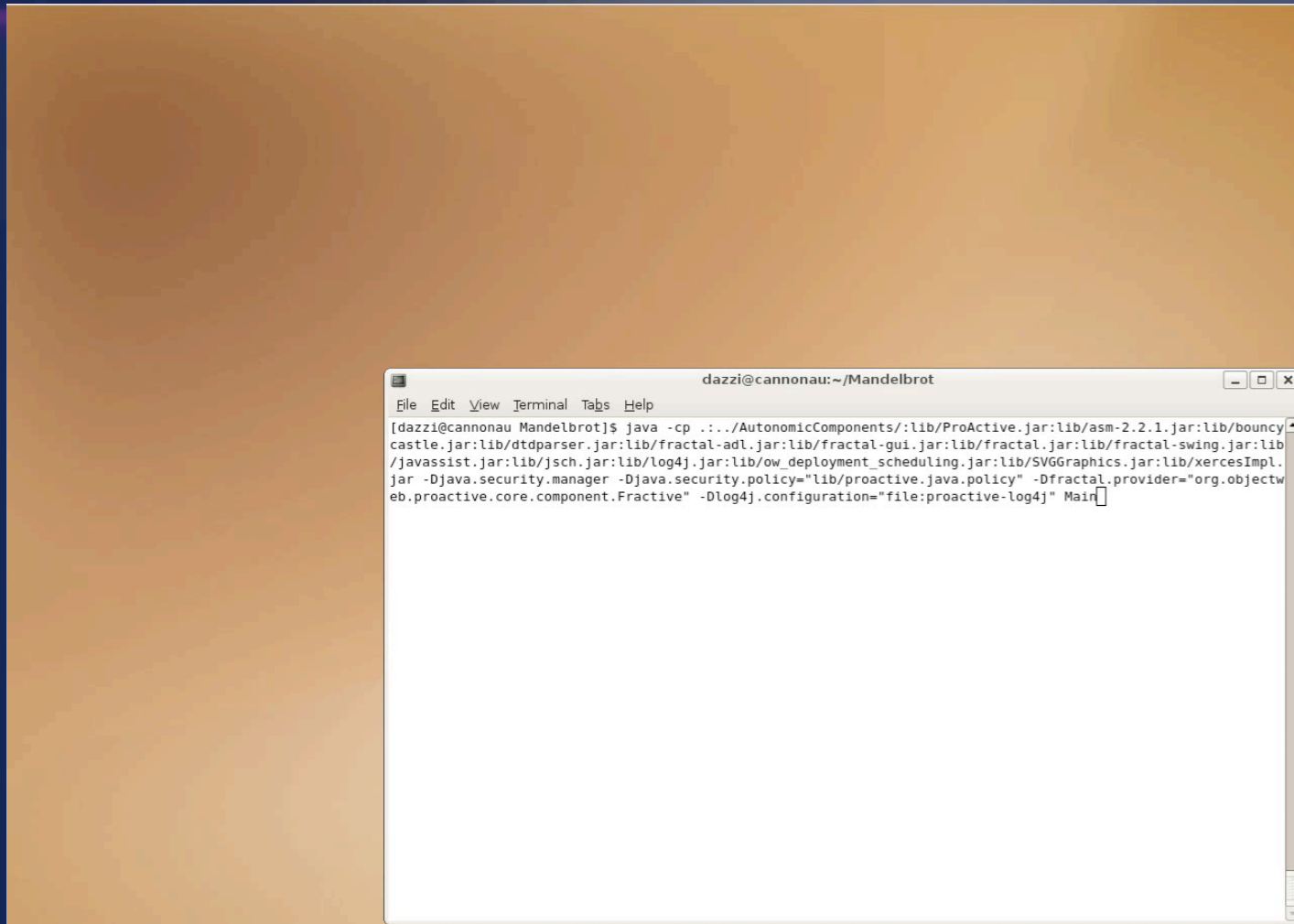
# Managing Autonomic Features (III)

## Farm Example





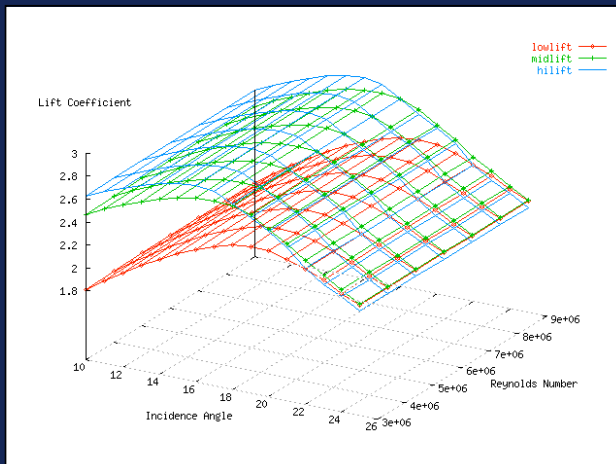
# Managing Autonomic Features (IV)





# Wing Design use case (I)

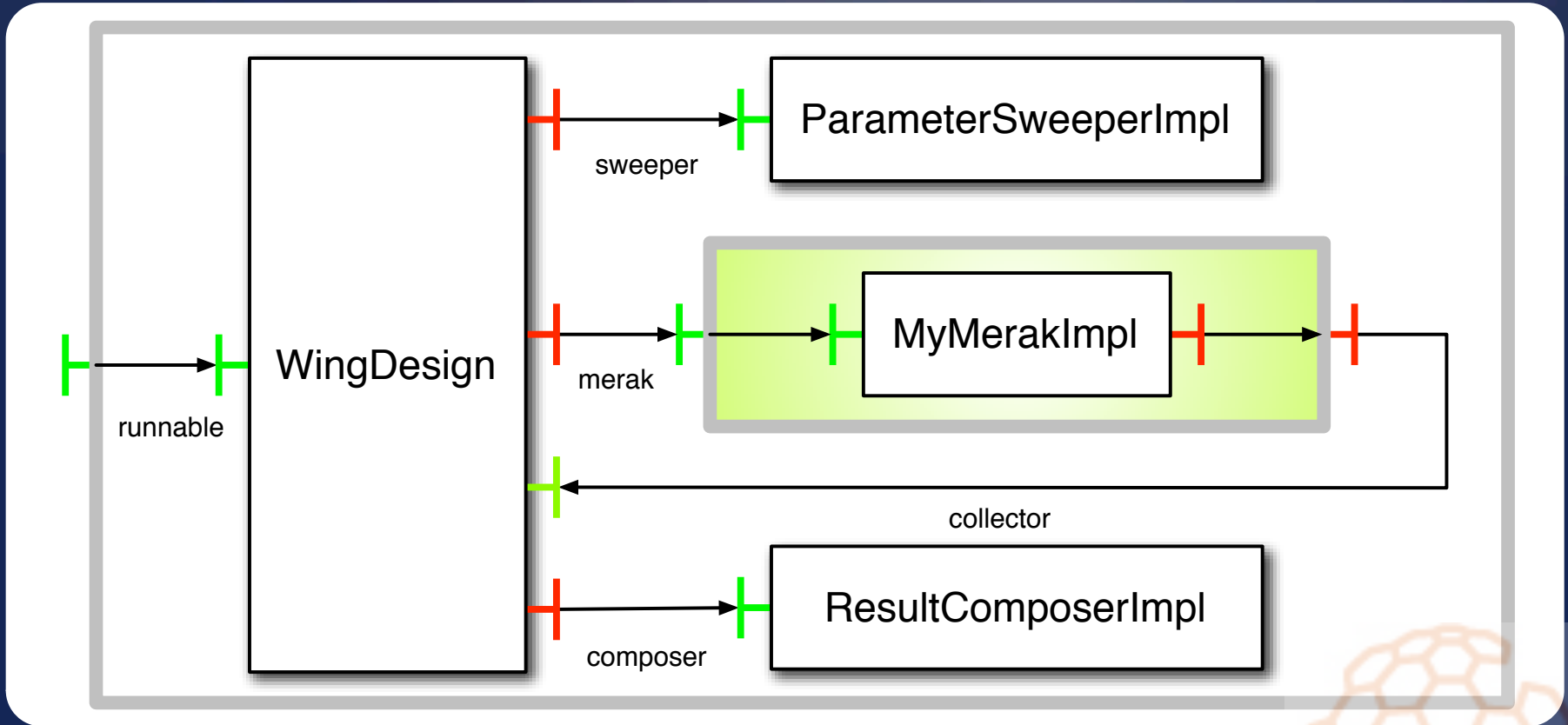
- Practical example from the aerospace sector
  - computes the aerodynamic wing performance for a given configuration.
- Merak processing
  - Merak is a fortran-77 program (binaries for Linux, Windows and Solaris)
  - Aerodynamic wing analysis
  - Massive sweeps by analyzing the 3-dimensional parameter space  
Wing geometries, Reynolds number, Incidence angle
- *Gnuplot* is used to generate *png* files from the result files



Courtesy of Grid Systems

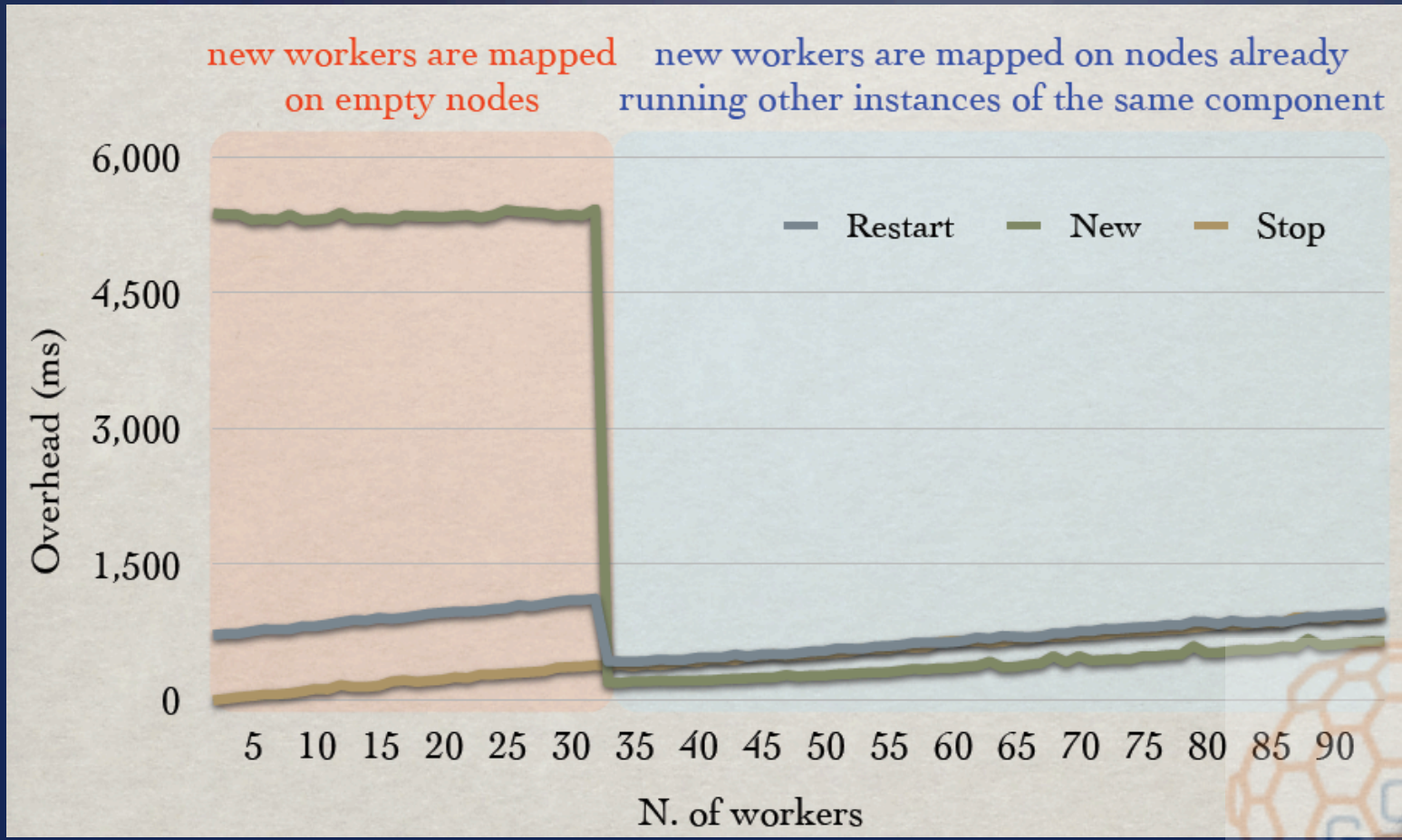


# Wing Design use case (II)





# Performances





# Conclusions

## GCM Autonomic Features

- Passive Behavior implemented in GridCOMP
- Active Behavior being implemented in GridCOMP

## Behavioral Skeletons

- Methodology to implement autonomic management
- Can be freely used with GCM components

## Everything is being tested on test applications from industry

- Atos, Grid Systems, IBM

## Some discussion points to improve GCM in CoreGRID

- Do we need a clear definition of **state** for a GCM component?
- Do we need a well-defined **semantics** for a GCM component **lifecycle**?
- Do we need the concept of **runtime deployment plan** for distributed autonomic components?





# Thank you!

# Questions?





# Thank you!

## Some discussion points to improve GCM in CoreGRID

- Do we need a clear definition of **state** for a GCM component?
- Do we need a well-defined **semantics** for a GCM component **lifecycle**?
- Do we need the concept of **runtime deployment plan** for distributed autonomic components?