JADE

« An adaptable infrastructure to build autonomic system »

Sara Bouchenak, Noel De Palma, Daniel Hagimont, Sacha Krakowiak
Jean-Bernard Stefani
SARDES Project, INRIA RHONE ALPES, GRENOBLE, FRANCE

Outline

- Motivations
- Design principle
- Use case
  - Jade for clustered J2EE application
  - Deployment and Repair management
- Performance
- Conclusion

Motivation

- Distributed Software
  - Complex, heterogenous and legacy
  - Management is required but nightmarish
- Example of management function
  - Software installation
  - Software configuration
  - Performance Tuning
  - Fault Tolerance
  - Securité

problems statement

- Management software
  - It can be a complex distributed application
- Management
  - Complex task
  - Achieved by human
- Consequence
  - Error (mainly configuration)
  - Low reactivity
  - Consume a lot of resources
    - human resources
    - hardware resources (overbooking)

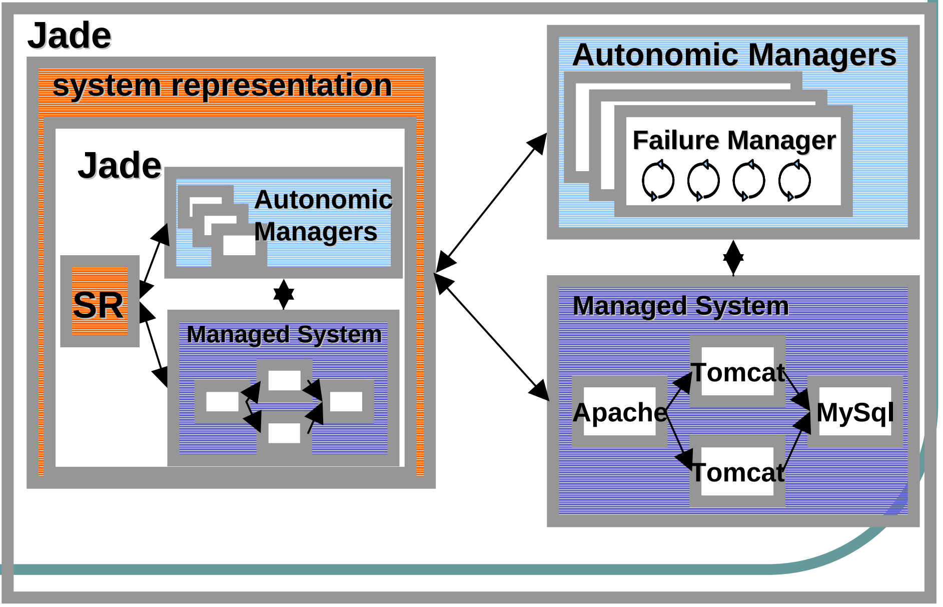
Approach : autonomic management software

- Management software
  - Installation, deploy, configure …
    - The managed system (Legacy)
    - The management system itself
- Autonomic behavior
  - monitor, decide, reconfigure
    - The Legacy application
    - The management system itself
- Benefit
  - Less error
  - More reactivity
  - Resources friendly
- We need the same abstraction
  - The managed system
  - The management system

Jade : design principle

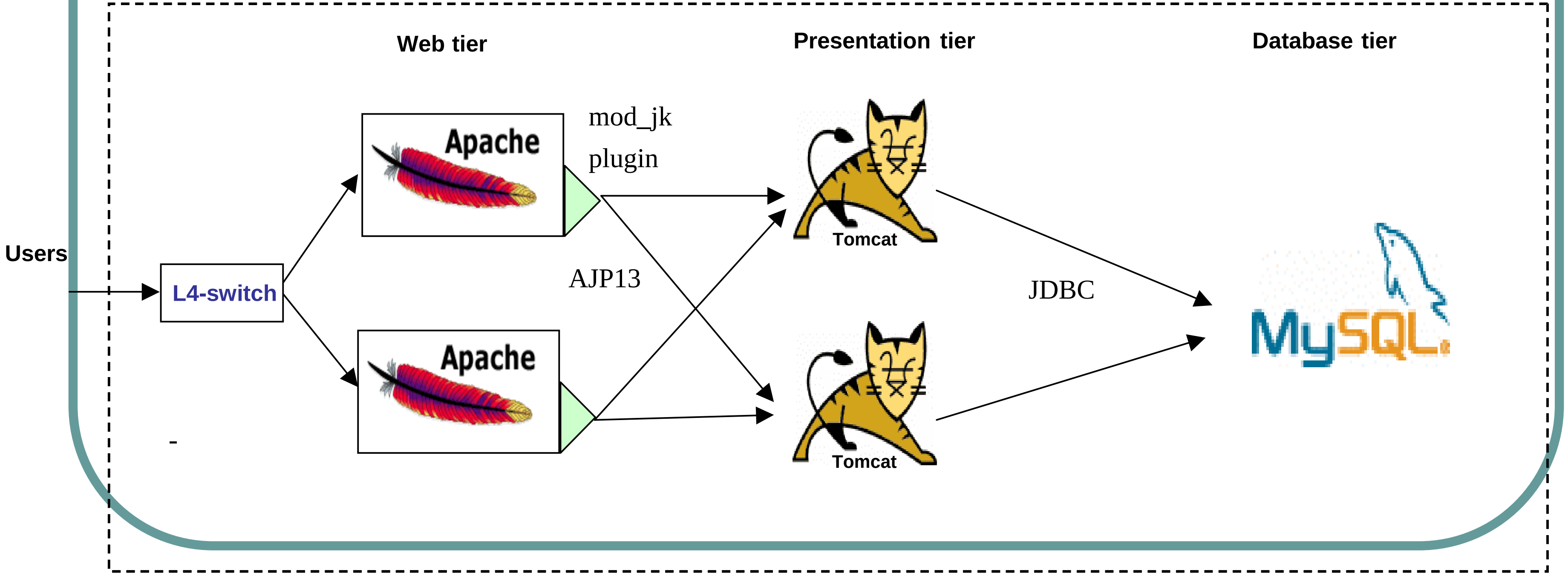
- Component model (Fractal - Julia - Think)
  - To model and wrap legacy managed resource
    - **Component abstraction over legacy software**
  - To build the management software
    - Jade is adaptable and jade is self managing
    - **Autonomic manager works on component abstraction**

- Management software
  - Bootstrap (self-deployable)
  - *Autonomic Manager*
  - *Managed Resource*
  - *Explicit Control Loop*
  - System representation



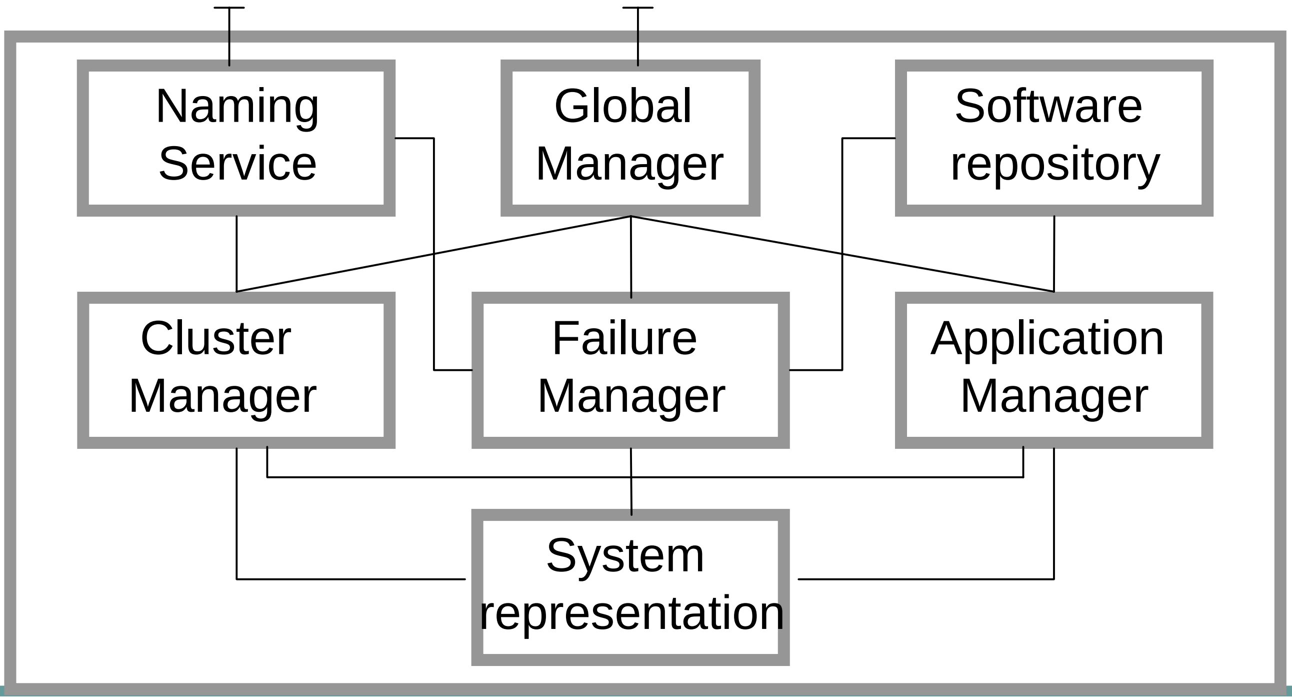
Viagra is Fault-tolerant

- A Jade personality to manage clustered J2EE application
- Repair management
- The application
  - RUBiS (servlet version based on e-bay model)



Jade in this context

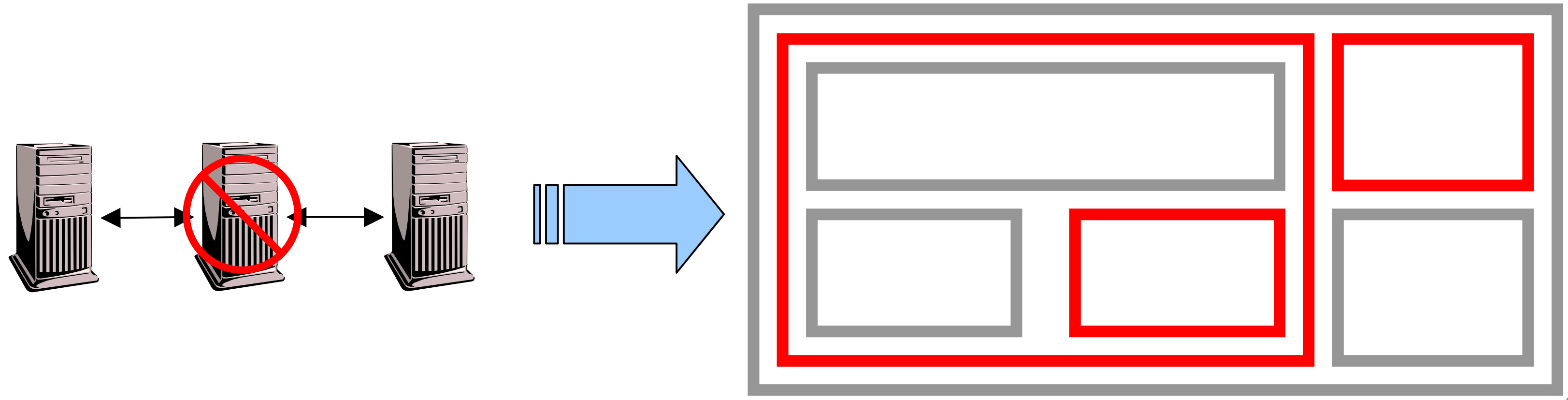
- Component abstraction is used to model/to control
  - Virtual cluster
  - Middleware
  - Application



Focus on the failure manager

Repair management

- **Component based** system's architecture (Fractal)
  - Repair of a components structure (legacy application or jade itself)
- **Fail stop** failure of node
- **Configurable** and **generic** repair policy
  - Updating the failed managed system conform to the configuration in place prior to the occurrence of failure

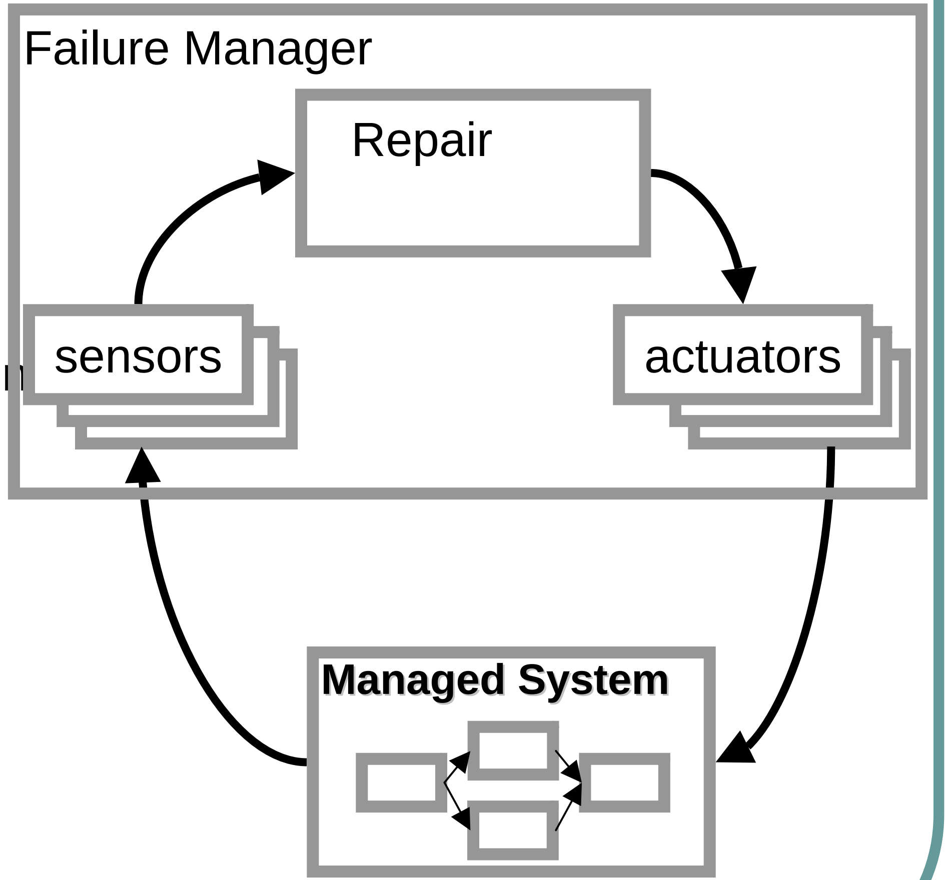


Failure Manager

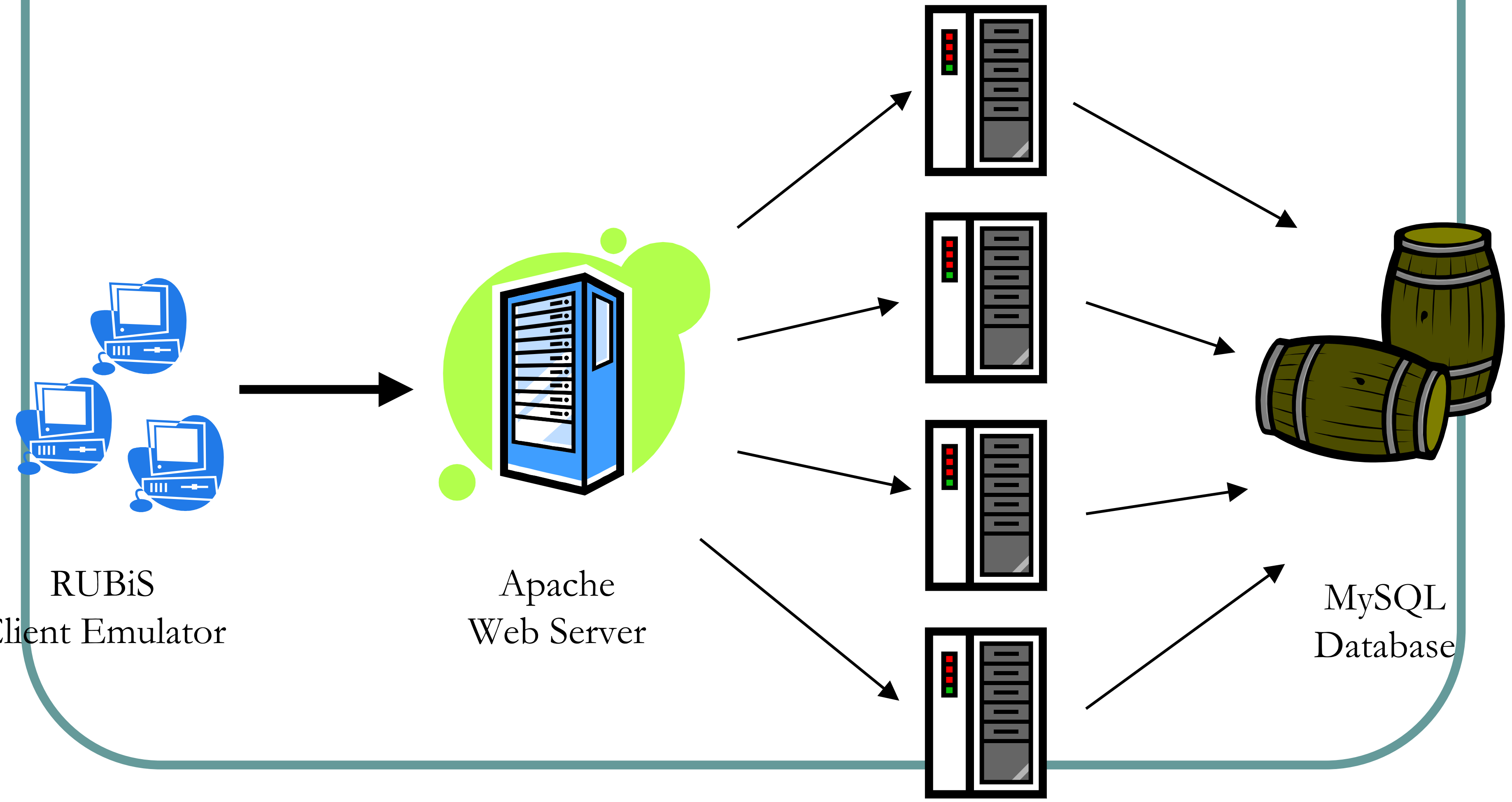
- **Explicit control loop**
  - Sensor
  - Actuator
  - Repair policy

- Manifests the relationship between managed system and repair management functions

- **Repair policy** described through an **ADL**
  - Set of actuators
  - Actuators scheduling
- **Self configurable**



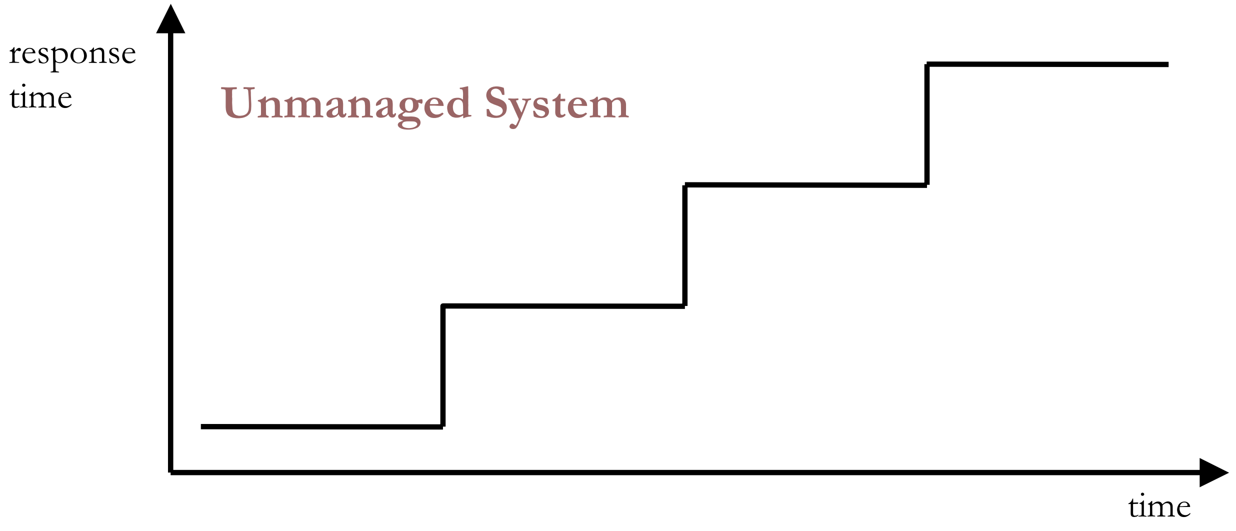
Evaluation Environment: Deployment Architecture

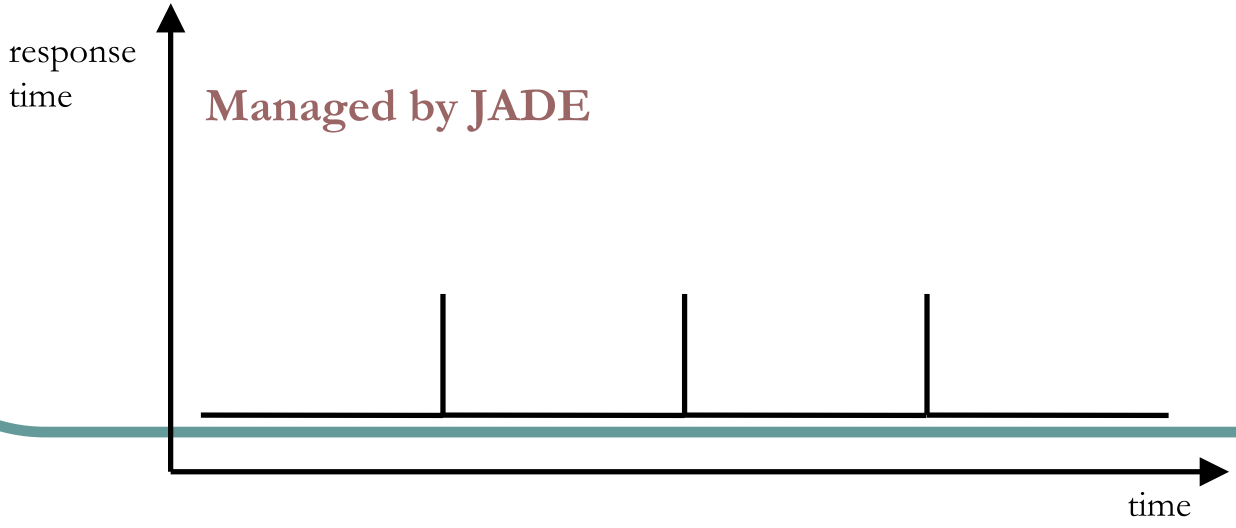


Evaluation Setup

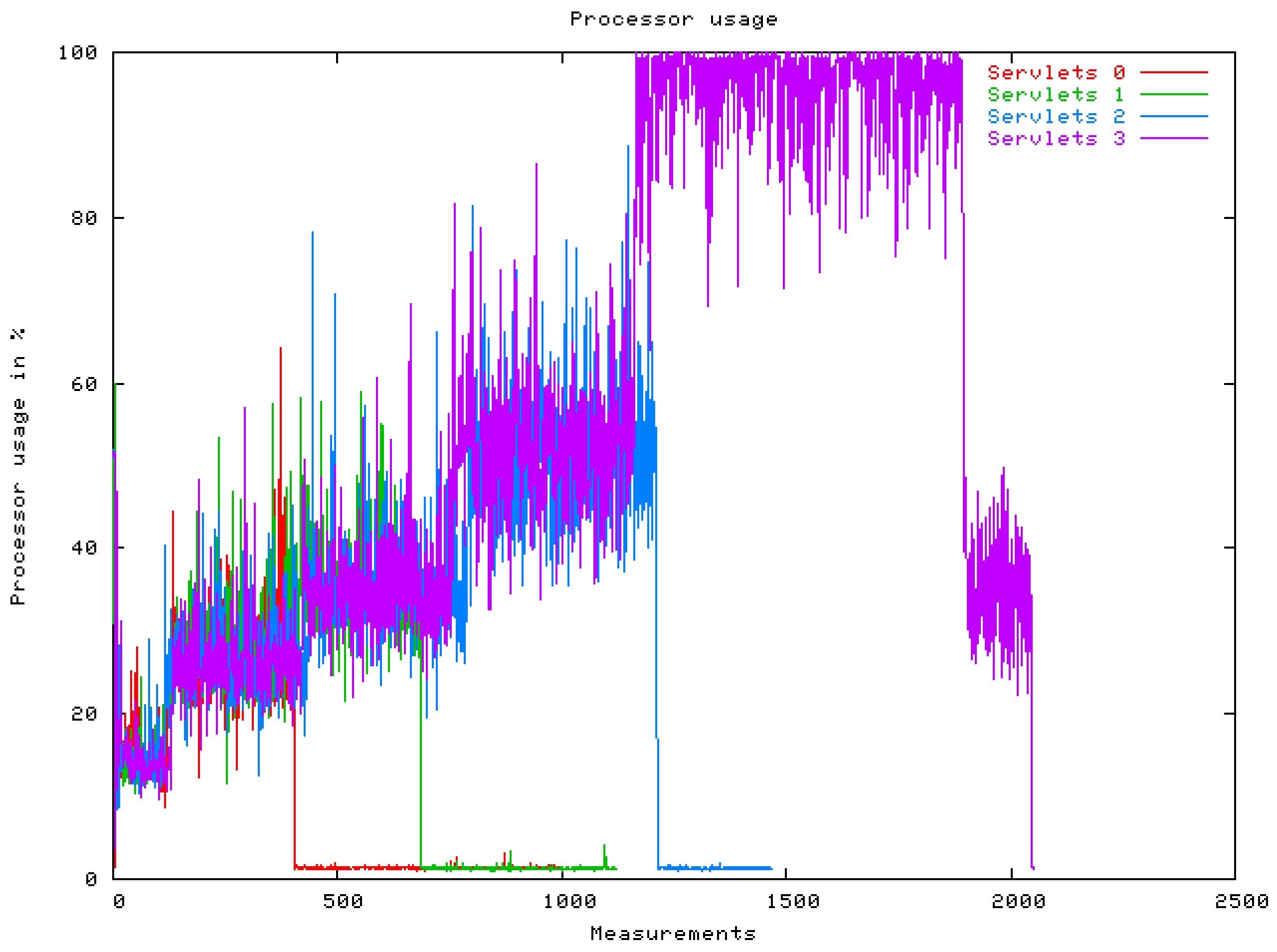
- RUBiS transition table → requests
- 600 users (TPCW think time)
- Test time
  - 500s ramp-up time
  - 4000s test time
  - 500s down-ramp time
- 3 consecutive Tomcat crashes (distance 1000s)

Expected / Ideal Graphs

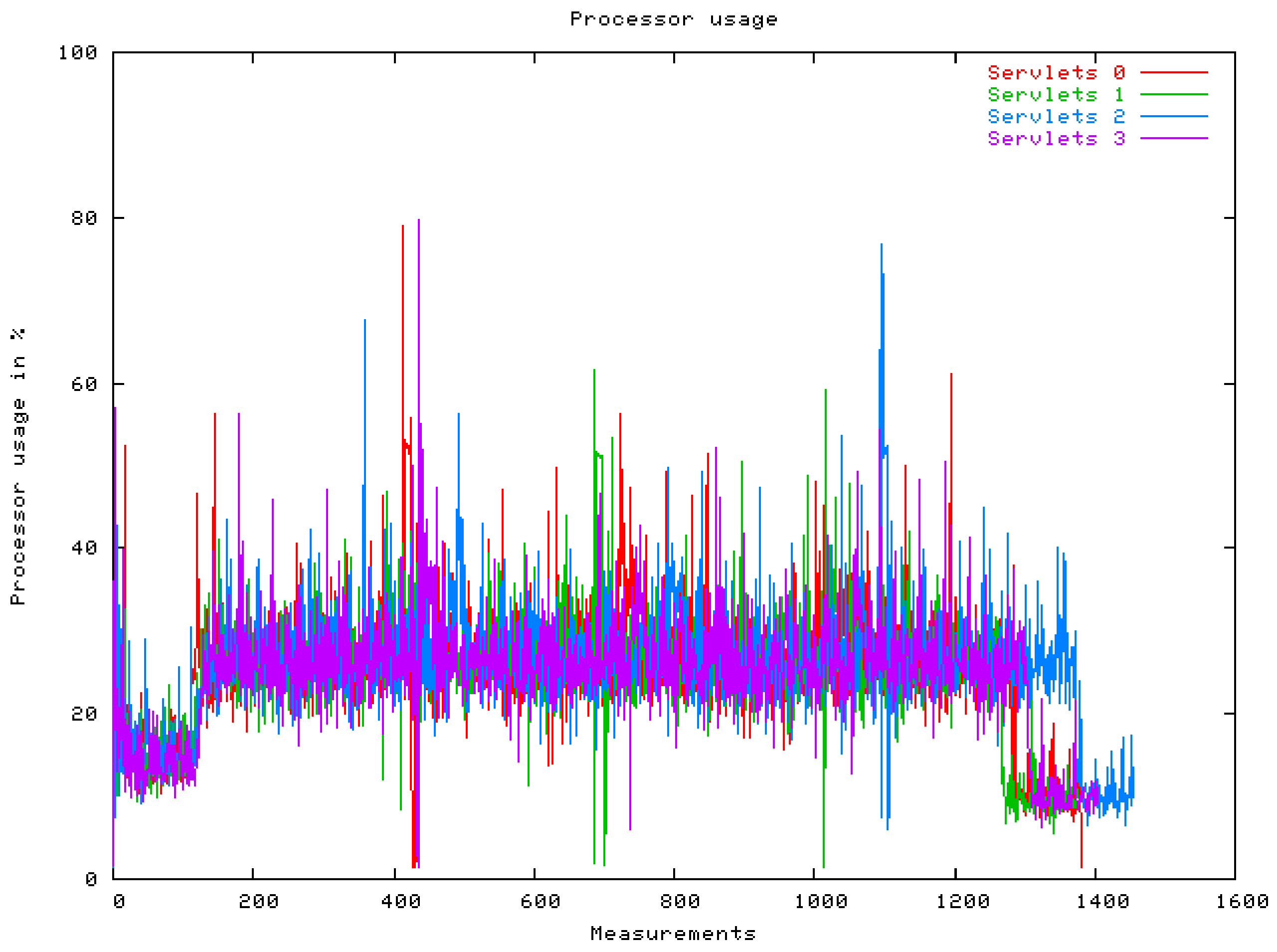




CPU Usage – no JADE



CPU Usage - JADE



conclusion and future work

- Management achieved by human
  - Error (mainly configuration)
  - Low reactivity
  - Consume a lot of resources
- Our approach : Autonomic Management
- Use Case and first evaluation
  - J2EE
  - Failure manager
- Under development
  - Autonomous Self sizing for J2EE applications
- Next
  - Autonomic management of message base application
- **Fractal, Julia, Think are in LGPL**
- **JADE is soon in LGLP**