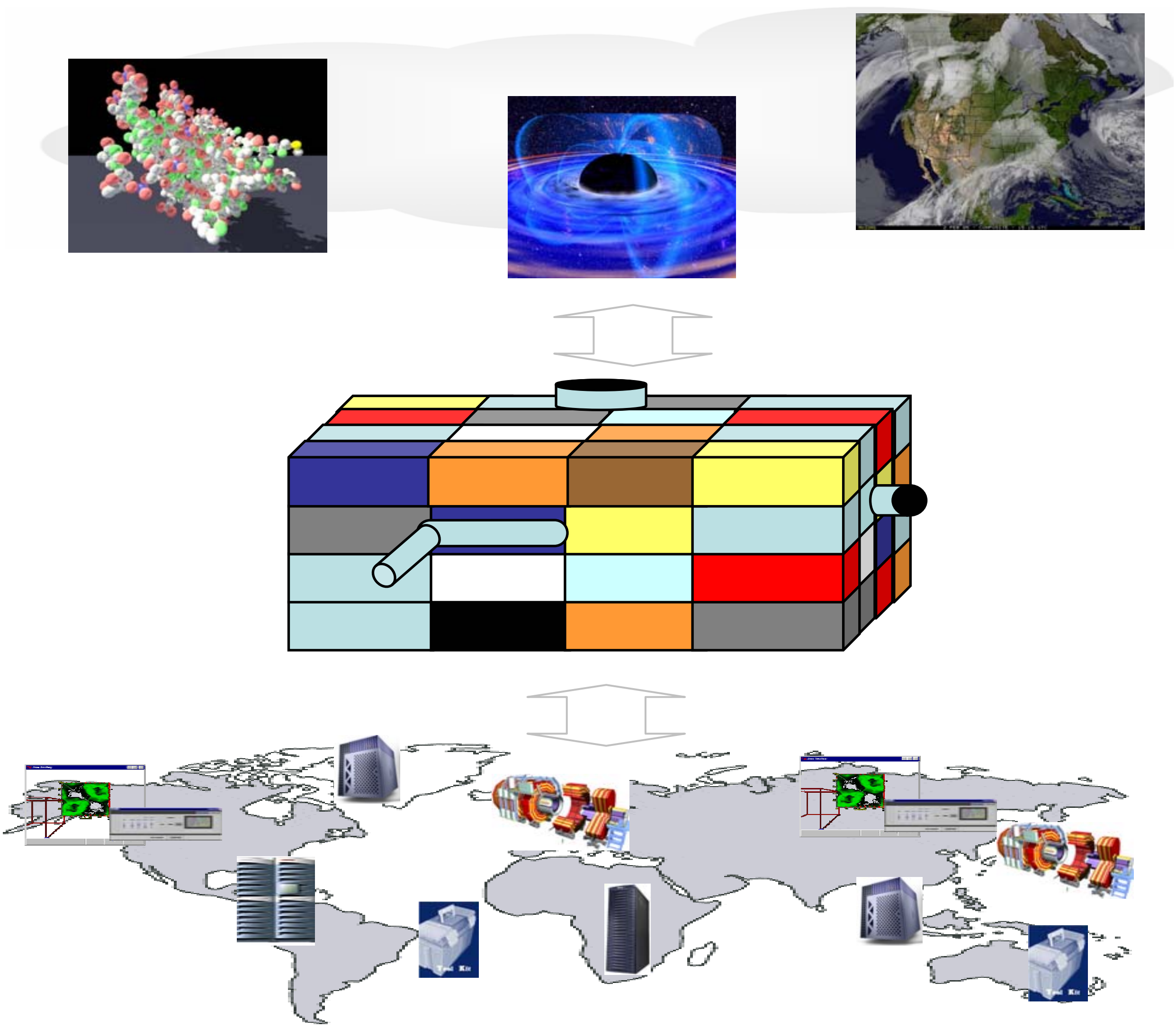
Architecture-Aware Autonomic Adaptations within the Common Component Architecture

J. Ray, M. Parashar, H. Liu, B. Allen, S. Lefantzi

Emerging Next-Generation of Scientific Applications



Physical phenomena

- Large
- Multi-phased / multi-scale
- Dynamic
- Heterogeneous

Scientific applications

- Long running
- Assembled from independently developed computational units
- Geographically distributed

Environments

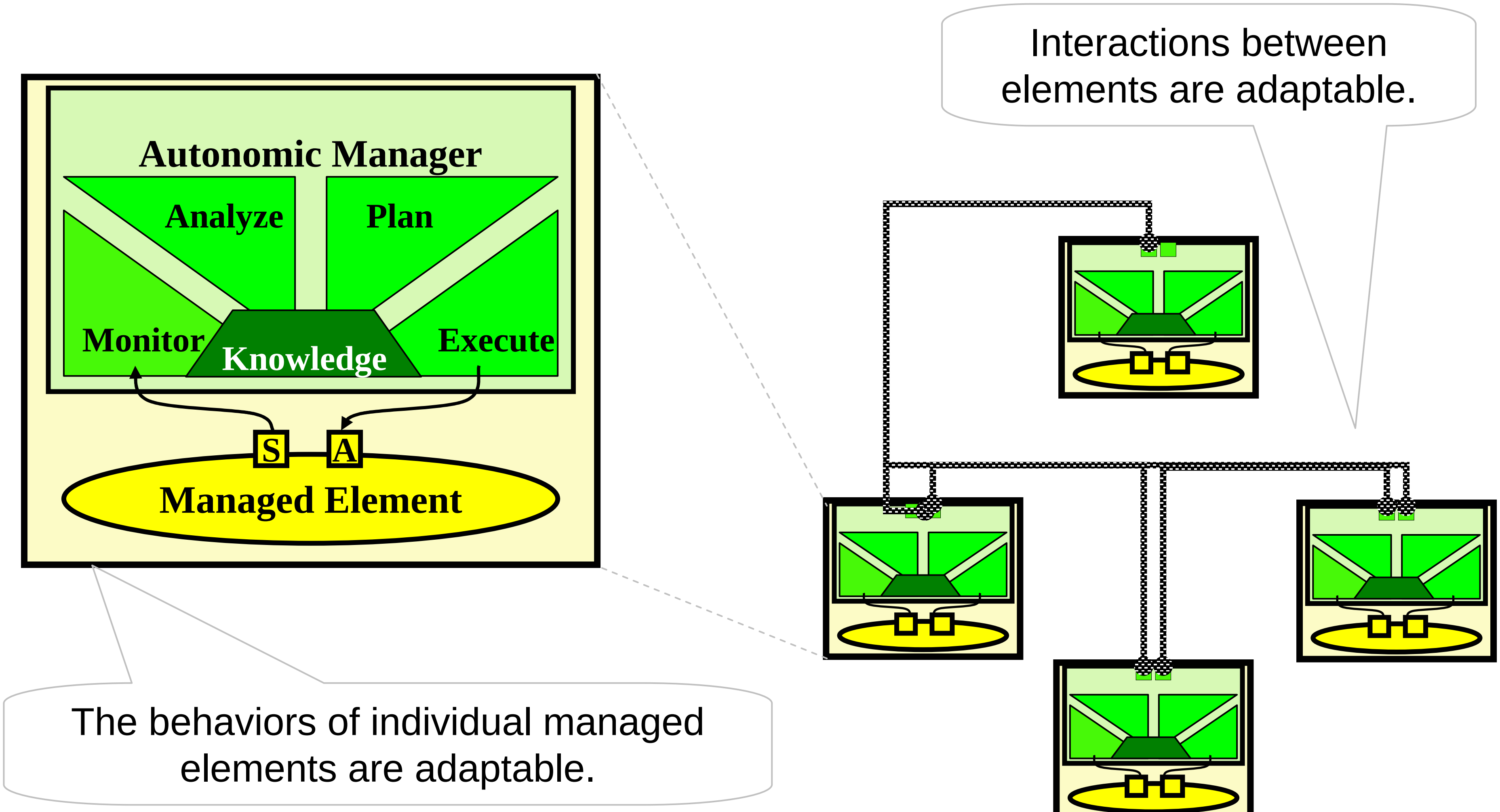
- Large
- Heterogeneous
- Dynamical

Challenges

- The emerging computing systems introduce a new set of challenges due to their scale and complexity
- The emerging simulations and the phenomena they model are similarly large, complex, multi-phased/multi-scale, dynamic, and heterogeneous (in time, space, and state).

Solution: Autonomic Computing

- Computing systems that can manage themselves given high-level objectives from administrators.
  - Self-configuration
    - Autonomic systems will configure themselves automatically in accordance with high-level policies.
  - Self-optimization
    - Autonomic systems will continually seek ways to improve their operation, identifying and seizing opportunities to make themselves more efficient in performance or cost.
  - Self-healing
    - Autonomic computing systems will detect, diagnose, and repair localized problems resulting from bugs or failures in software.
  - Self-protecting
    - Autonomic computing systems will automatically defend against malicious attacks or cascading failures.


Interactions between elements are adaptable.
Autonomic Manager
Analyze
Plan
Monitor
Knowledge
Execute
S
A
Managed Element
The behaviors of individual managed elements are adaptable.

- Source: IBM

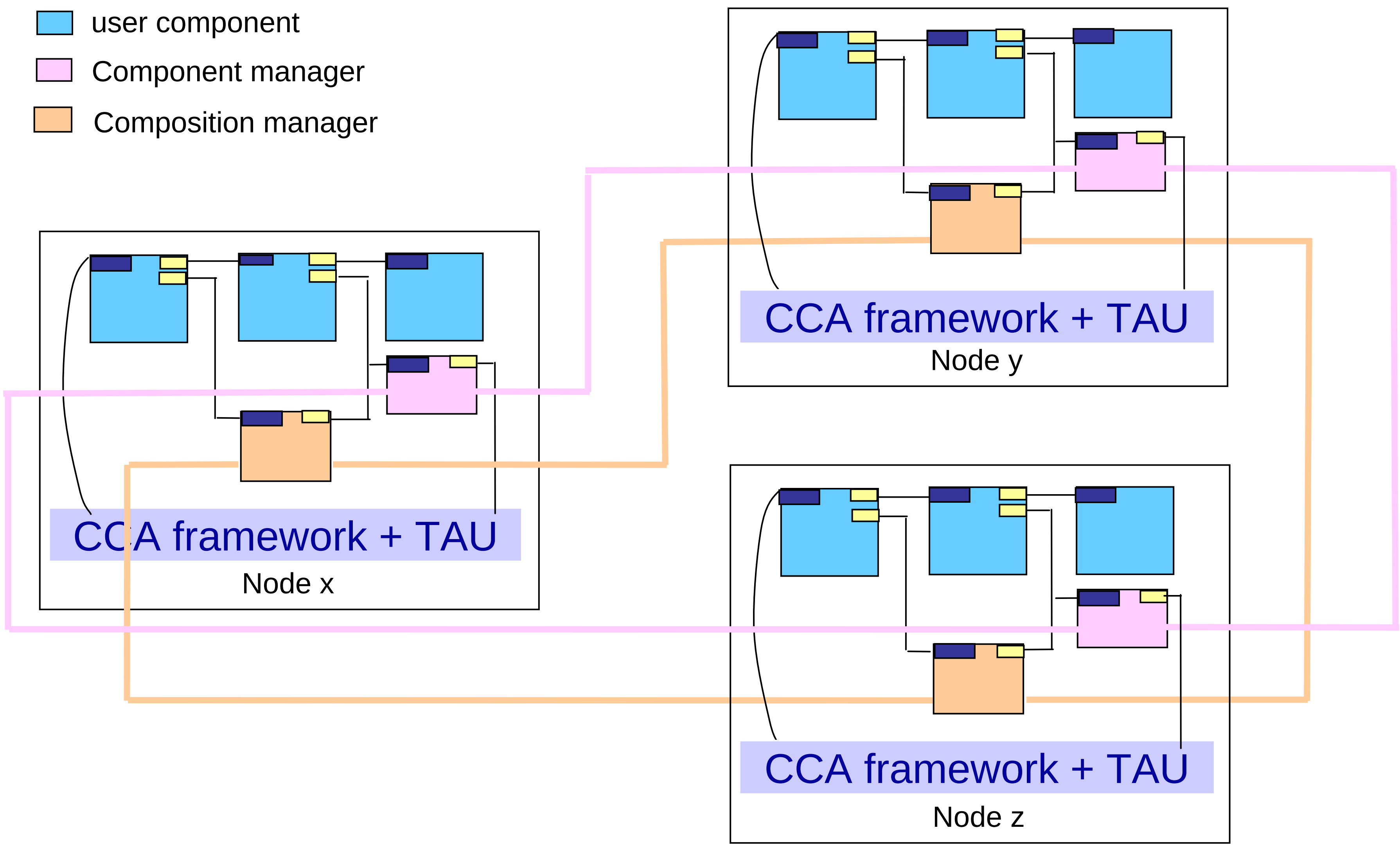
Overview of CCA

- Components
  - They are peers
  - They interact through provide/use ports
- A CCA framework (e.g., Ccaffeine)
  - It holds components and composes them into applications via connecting their ports
  - It supports SCMD and MCMD models

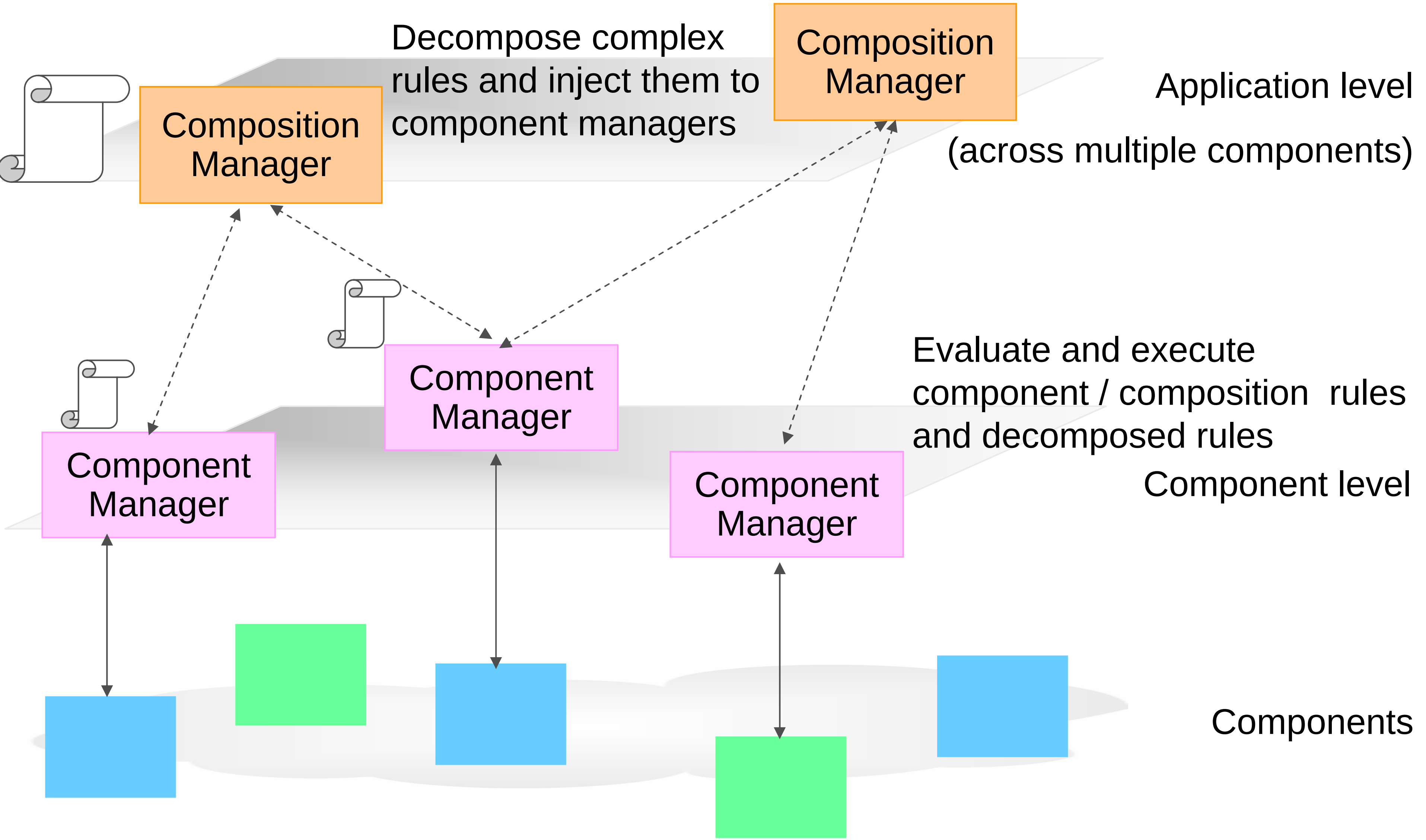
Extending Ccaffeine to Enable Autonomic Adaptations

- Component Manager
  - monitor and control the computational behaviors of individual components at runtime (e.g., dynamically selecting optimal algorithms, modifying internal states)
  - dynamically replace components
  - Dynamically coordinate with other component managers or composition managers to add / delete components
  - perform tasks assigned by composition managers
- Composition Manager
  - manage, adapt and optimize the overall execution of an application at runtime.
- Performance Toolkit (e.g., TAU)
  - Monitor resource utilization

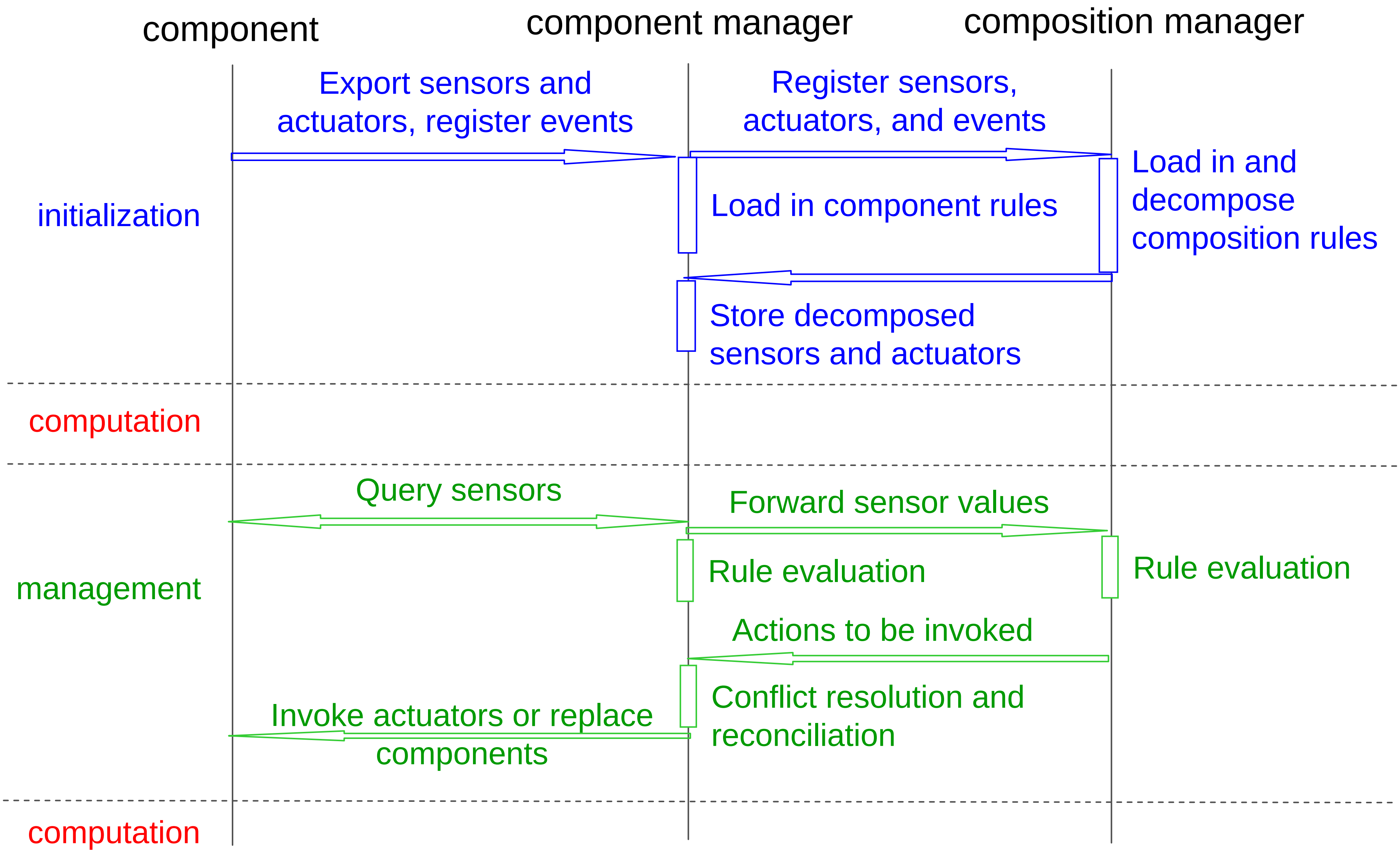
Extending Ccaffeine to Enable Autonomic Adaptations



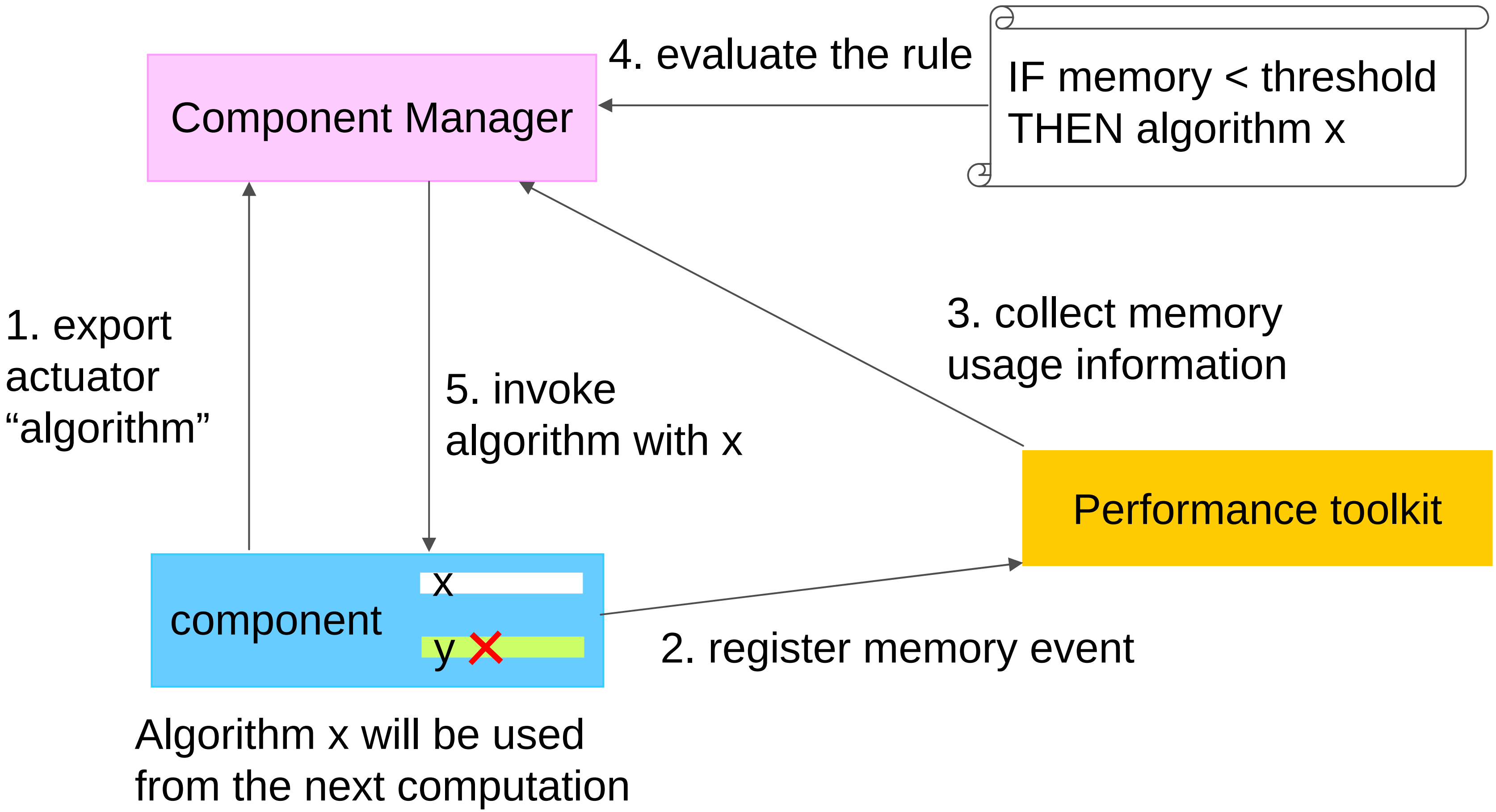
Management Hierarchy



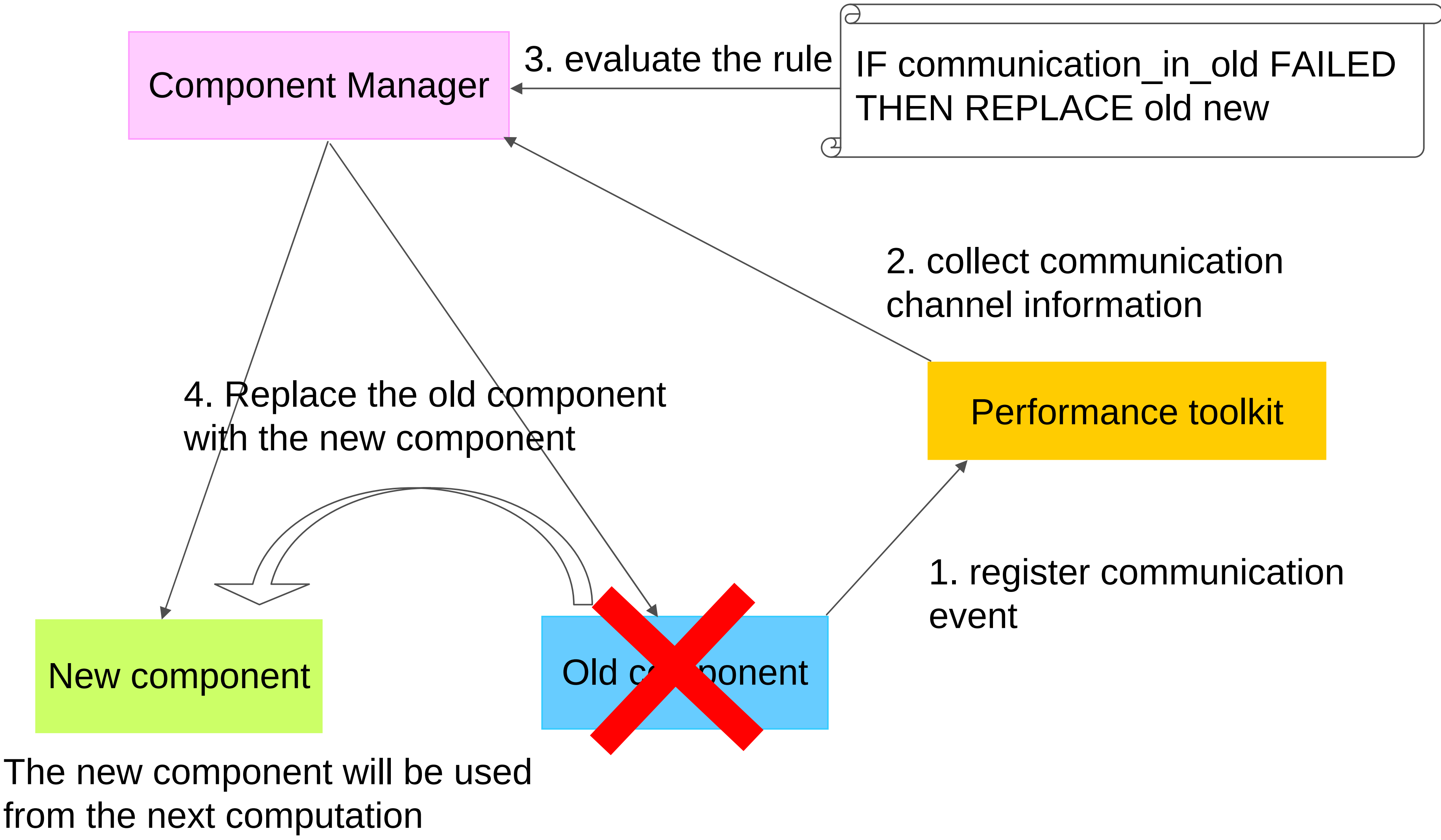
Workflow


component
component manager
composition manager
Export sensors and actuators, register events
Register sensors, actuators, and events
Load in and decompose composition rules
initialization
Load in component rules
Store decomposed sensors and actuators
computation
Query sensors
Forward sensor values
management
Rule evaluation
Rule evaluation
Actions to be invoked
Conflict resolution and reconciliation
Invoke actuators or replace components
computation

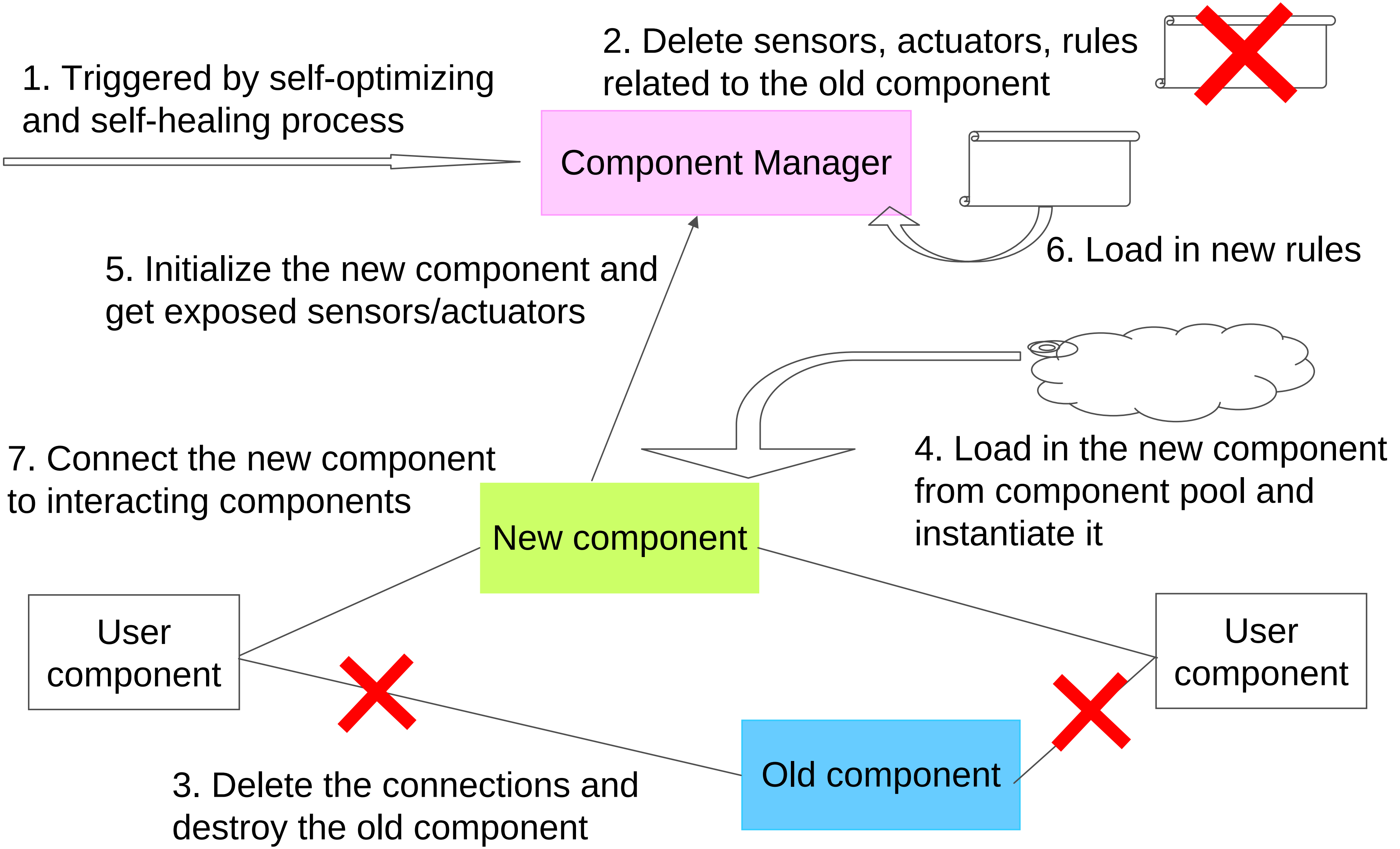
Self-optimizing



Self-healing



Self-configuring



Experimental Evaluations

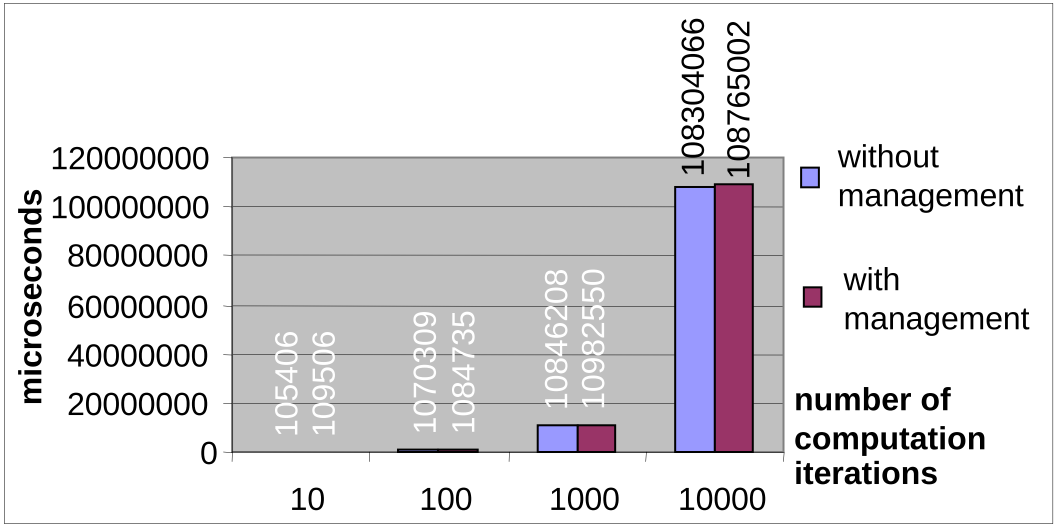


Fig 1. The runtime overhead introduced in the minimal model

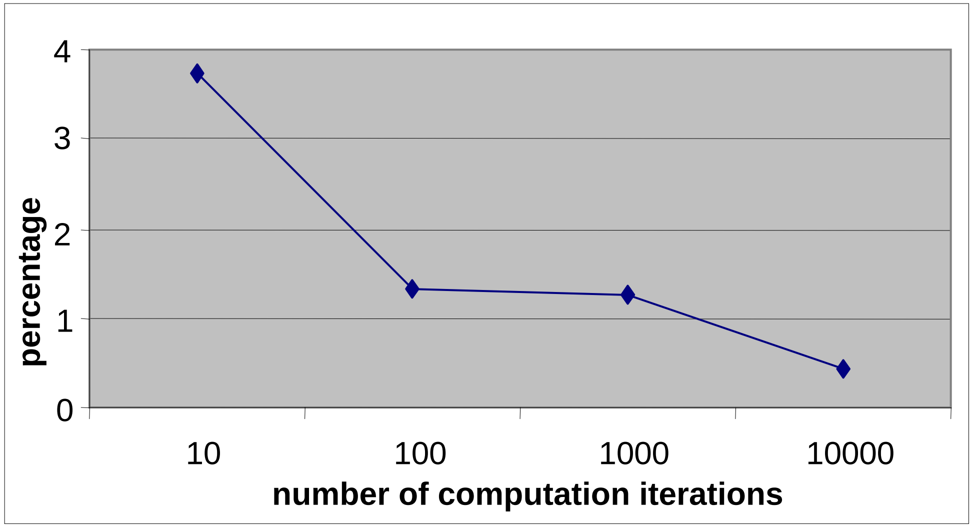


Fig 2. The percentage of overhead in the overall execution time

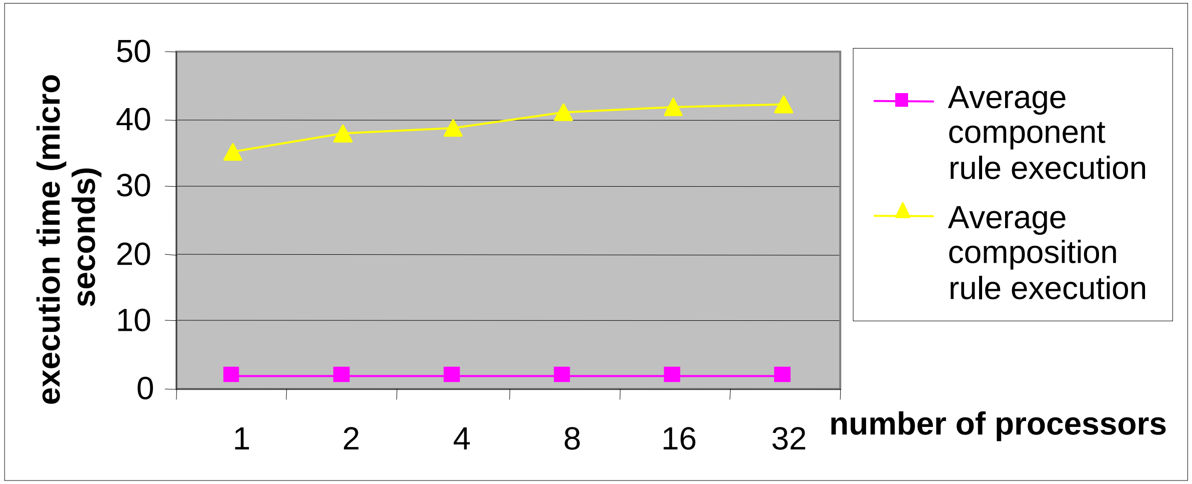


Fig 3. The average execution time of component rules and composition rules on parallel processors

Conclusion

- Challenges of the next generation of scientific applications
- Solution
- Extending Ccaffeine to Enable Autonomic Adaptations
- Self-management scenario
- Experimental evaluations