



Grid4All

Self-Management

Grid4All Perspective



Per Brand, SICS, Stockholm, Sweden



European Commission



Information Society

Grid4All Goals

‘Democratization of the Grid’

A Grid4All VO

- Collaboration— individuals and organizations are members
- Members
 - Bring some resources to the VO (not necessarily available all or even most of the time)
- VO-wide infrastructure
 - Encompassing member resources and upon need additional market acquired resources
 - Applications/services may be launched/regulated/managed/stopped on this infrastructure

Grid4All

Grid4All Assumptions - 1

Three characteristics of Grid4All VOs

- **Massive churn**
Individuals entities come and go while totality remains roughly the same
- **Evolution**
The totality of involved entities changes in time
Non-planned
- **Non-IT professional**
Less capable/willing human management of the VO
Need low cost

Grid4All

Grid4All Assumptions - 2

High Dynamism

- **Resources**
 - Internal VO resources: member resources
 - External: acquired temporarily from market
- **Members**
- **Failures**
 - More unreliable machines and networks
- **Usage**
 - Crowds & fashion: e.g. flash crowds
- **Heterogeneity**
 - Machines, set of applications, e.g.
- **Software change**
 - Updates in a service/application
 - Adding/removing service/applications
- **Policy**
 - A democratic grid: Analogy - a change of government
 - Changes in member composition

Grid4All

Management

What needs to be managed ?

- The VO
Mostly addressed yesterday (Vlad)
- Applications/Services
All kinds.

Why self-management ?

- Humans too expensive/unwilling
- Humans too slow/inaccurate for high levels of dynamism
- The goal of autonomic computing - humans should only need to set high-level policies

Note: this is from the user/administrator point-of-view!!

Grid4All

Axes of Self-Management

Self-configuration

changing the configuration in response to changes in needs or the environment. Both components and resources can be added or removed to running systems and the system reconfigures itself appropriately (initial configuration as a special case)

Self-healing

prevent and recover from failures by discovering, diagnosing, circumventing, and recovering from hardware and software faults

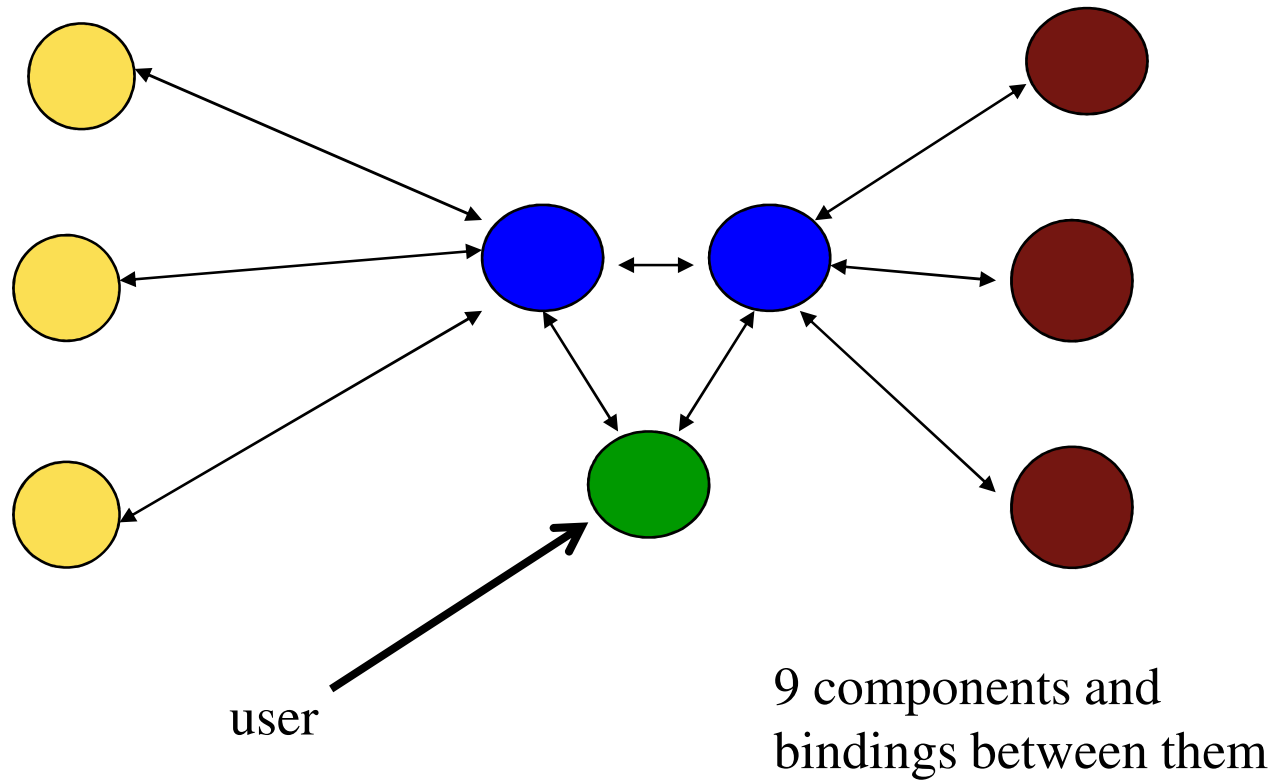
Self-tuning (or self-optimization)

system continuously tunes itself in response to changes in resource availability, the environment or usage

Self-protecting

the system can detect, identify, and defend against threats such as viruses, unauthorized access, and denial-of-service attacks

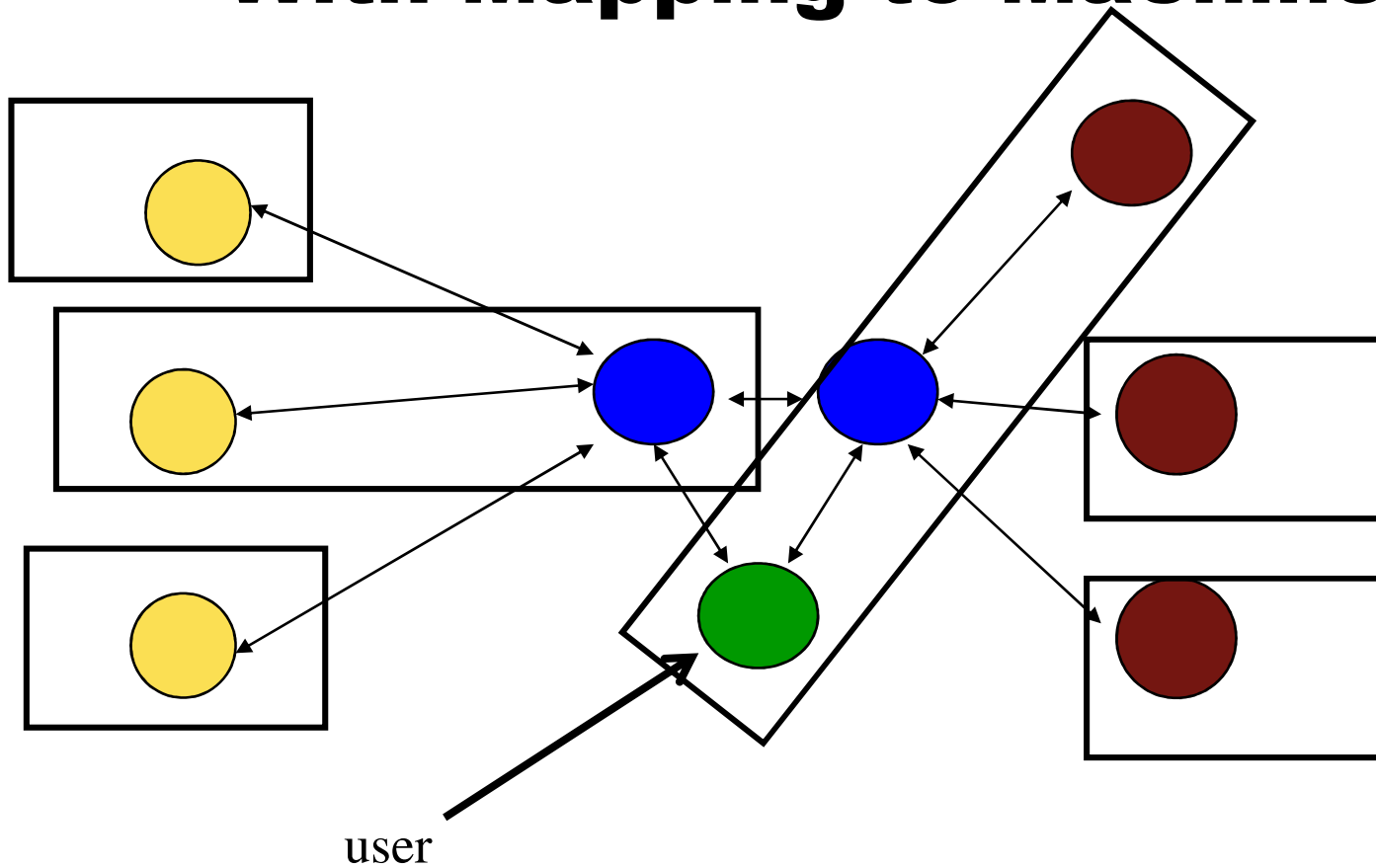
Pictorial View of an Application Architecture



At a given moment in time

Grid4All

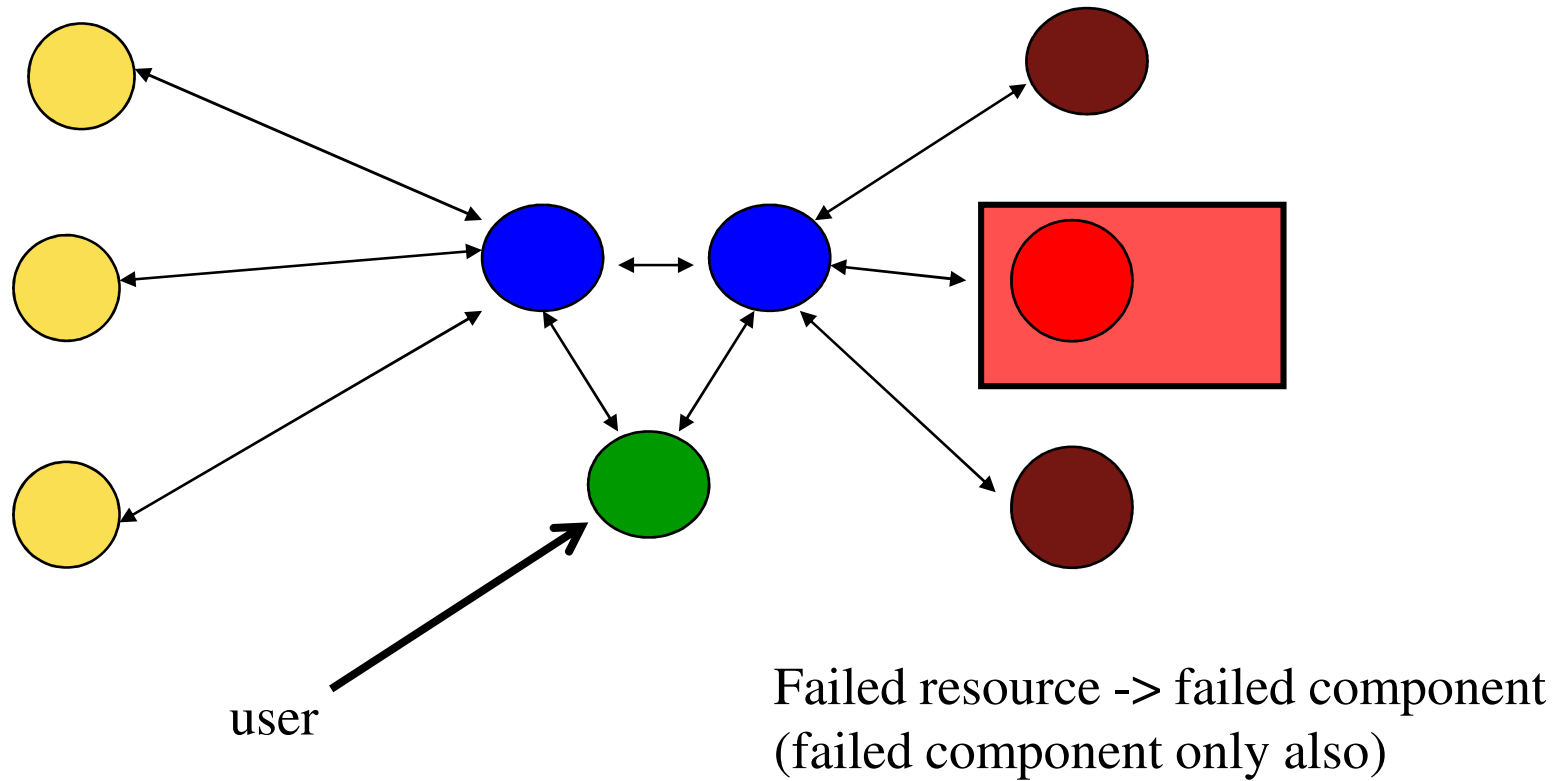
With Mapping to Machines



Initial configuration:
find suitable resources

Grid4All

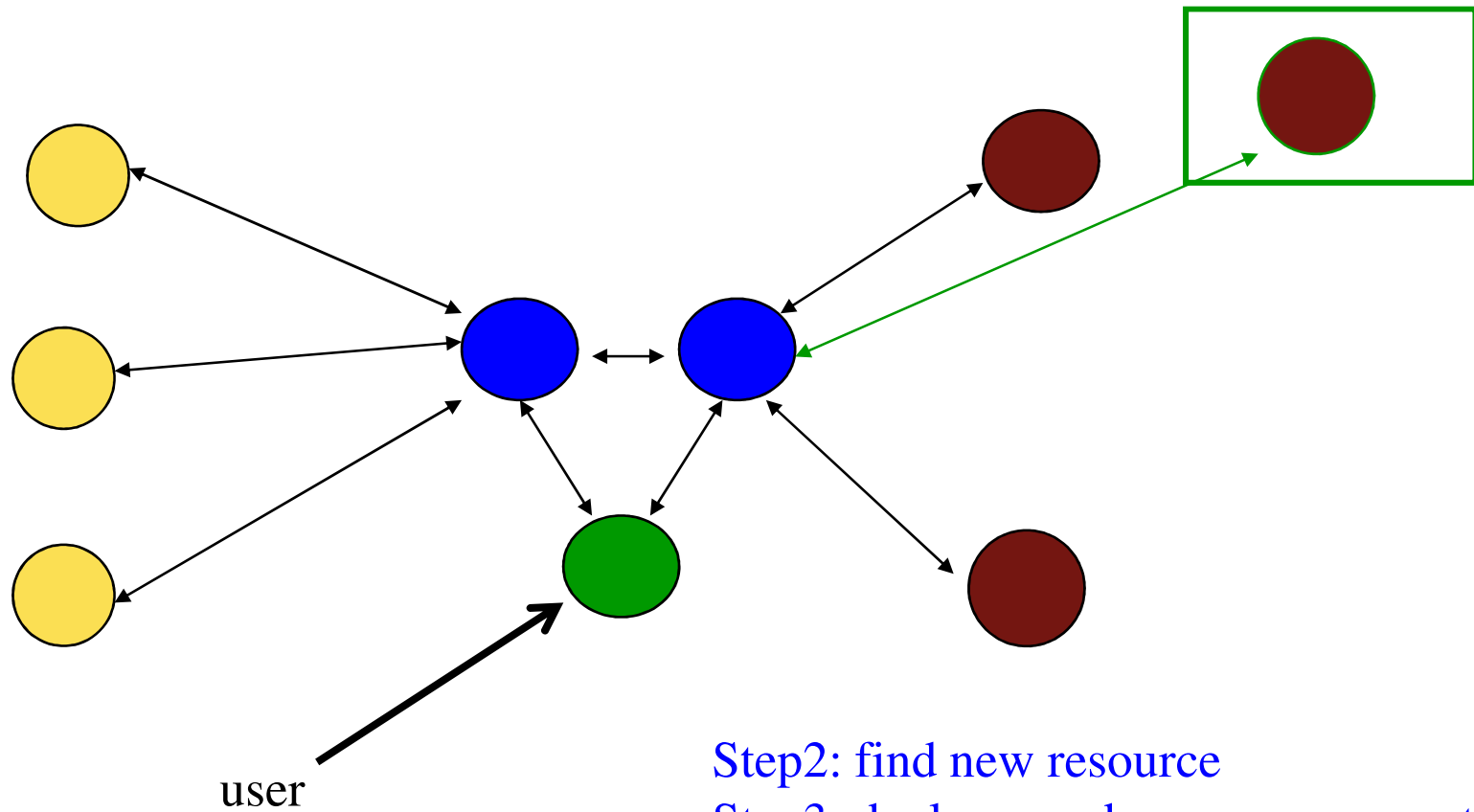
Self-healing



Step1: sense failure

Grid4All

Self-healing - 2



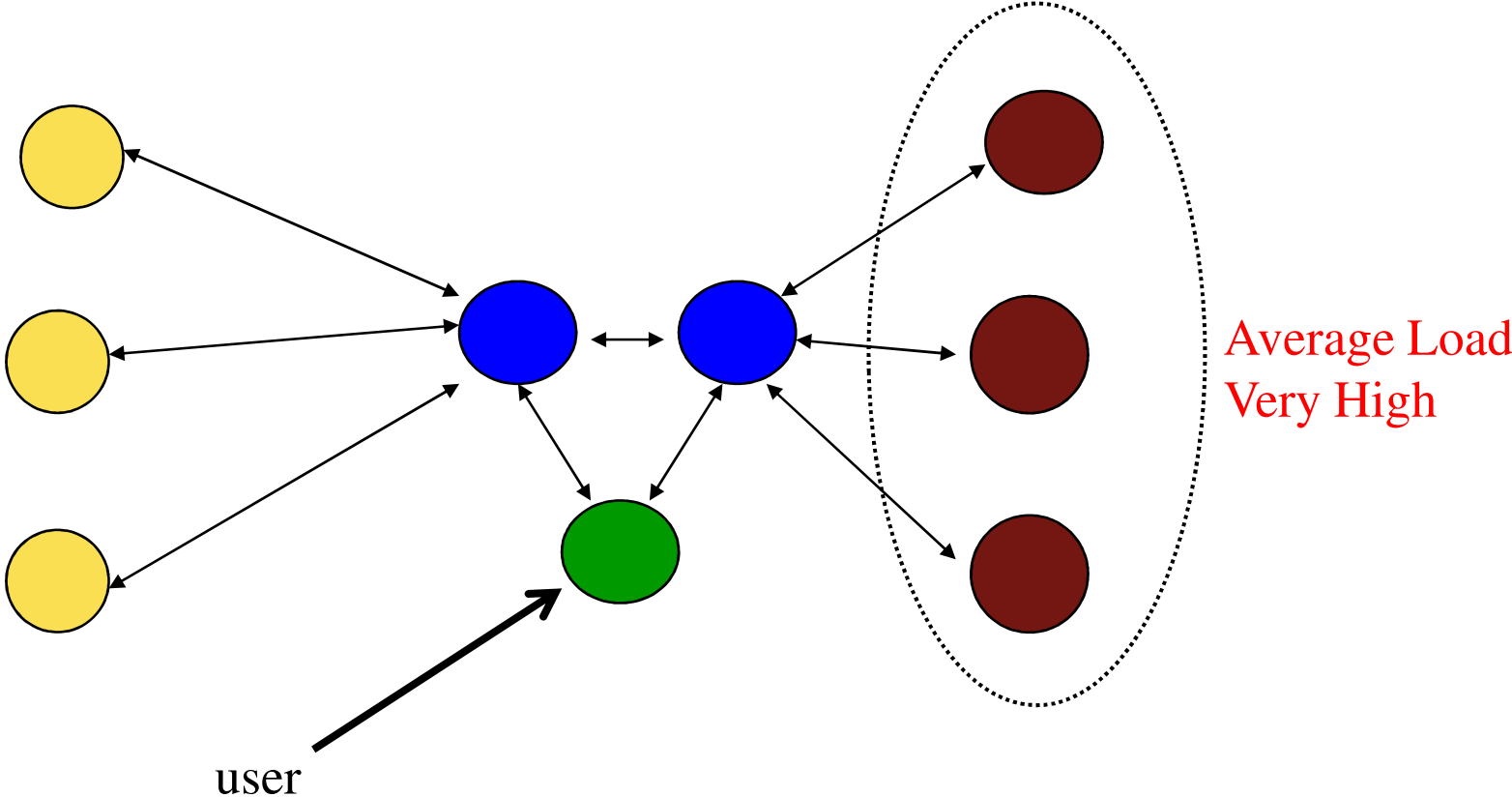
Step2: find new resource

Step3: deploy new brown component

Step4: bind (or something similar)

Grid4All

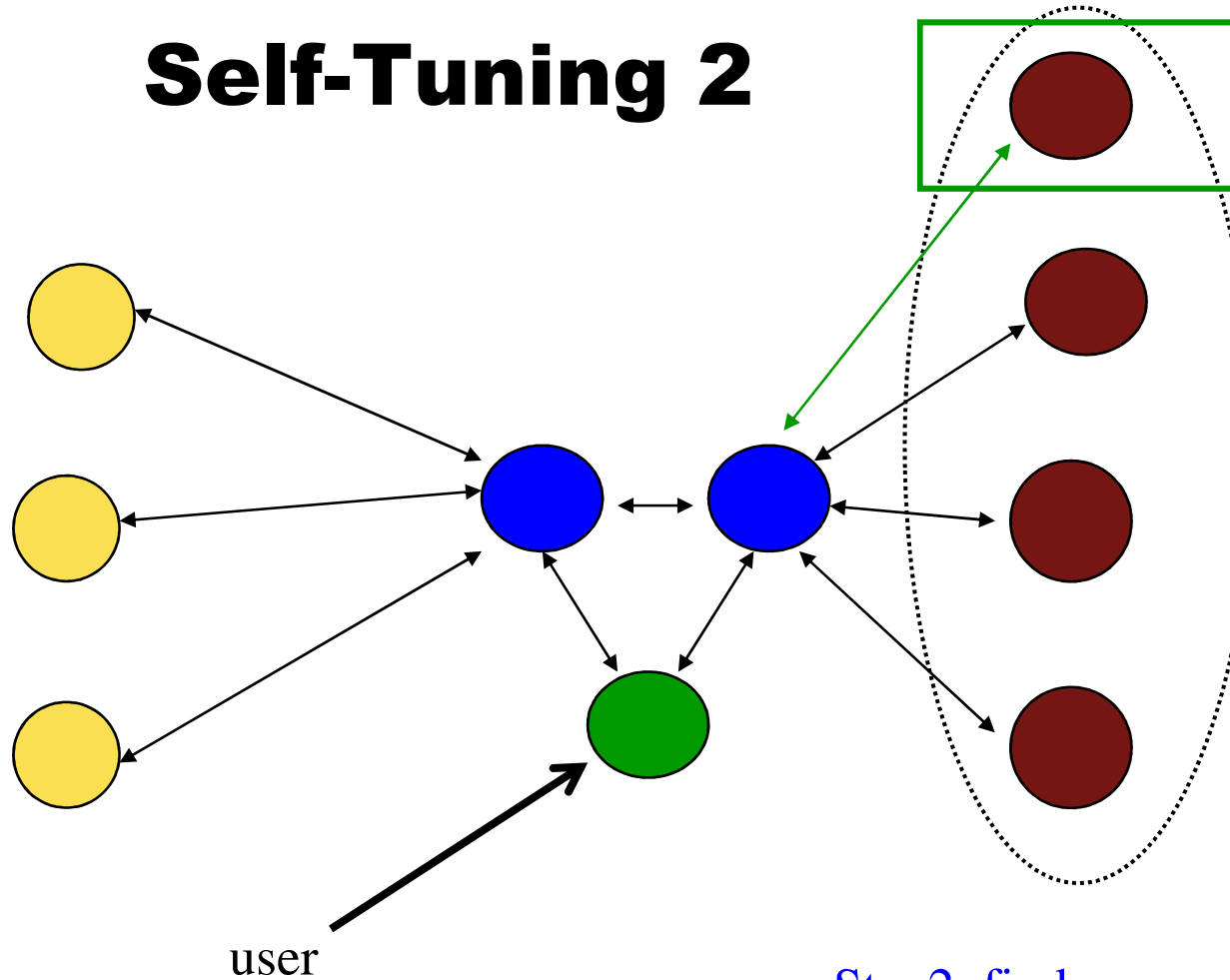
Self-Tuning



Step One: sense env. change

Grid4All

Self-Tuning 2



Step2: find new resource

Step3: deploy new brown component

Step4: bind (or something similar)

Grid4All

Self*

Require change

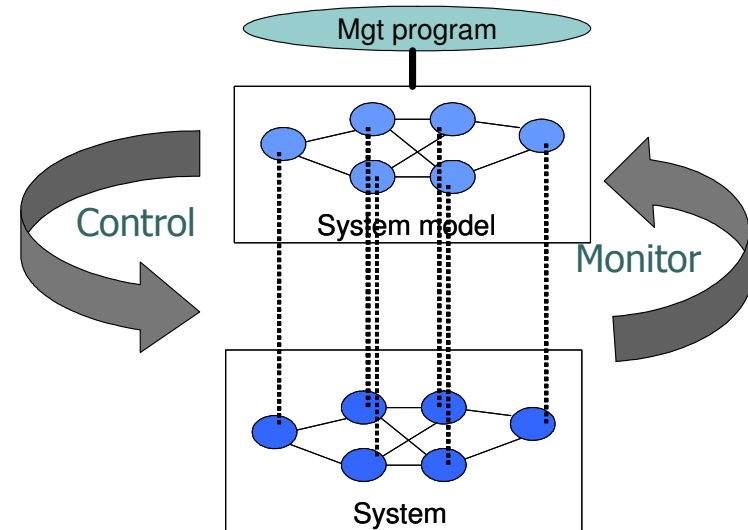
- Number/kind of components
- Connections between them
- Attributes/properties of components

General approach

- Feedback loops
Sense/monitor
Plan/decide/
Actuate/control

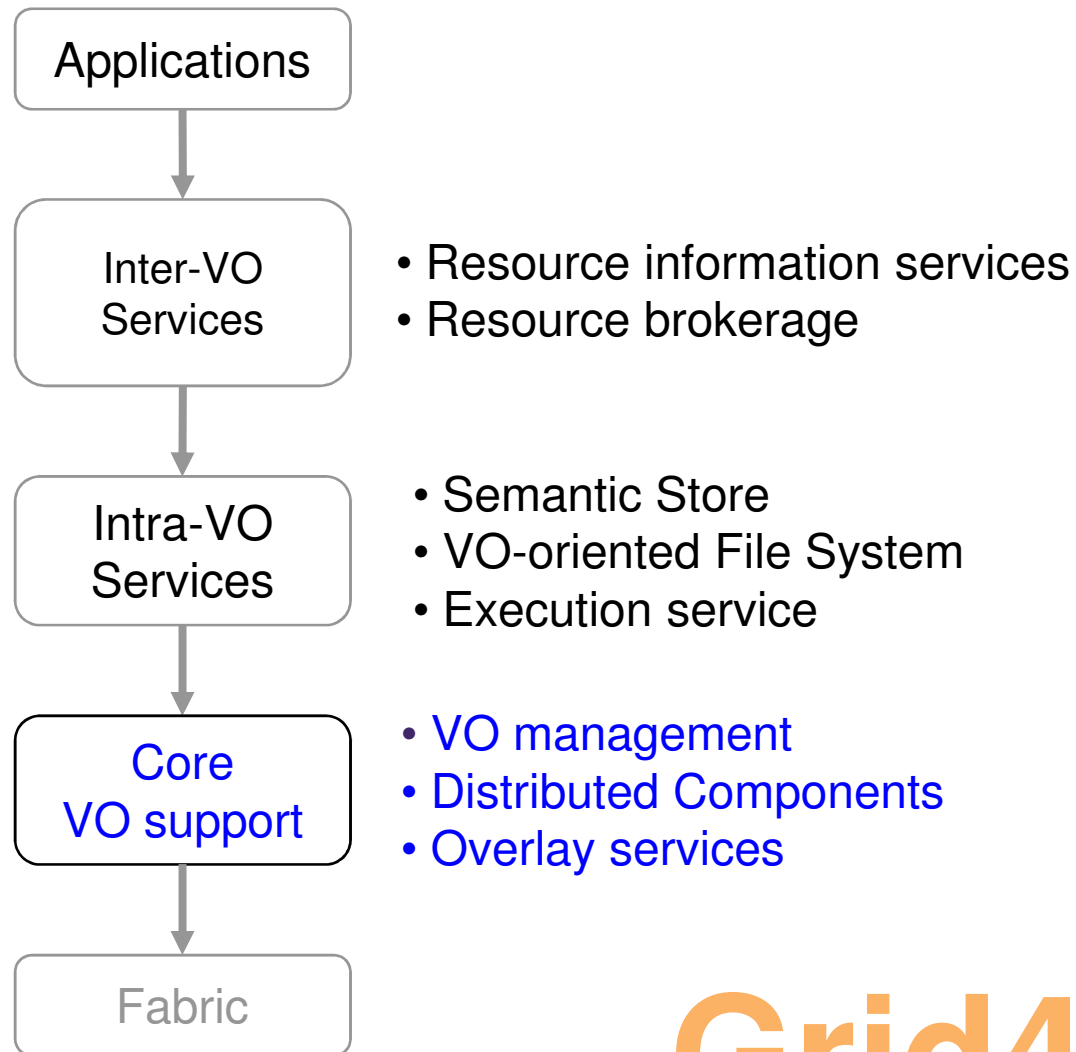
Divide and conquer

Hierarchical
System, subsystems,



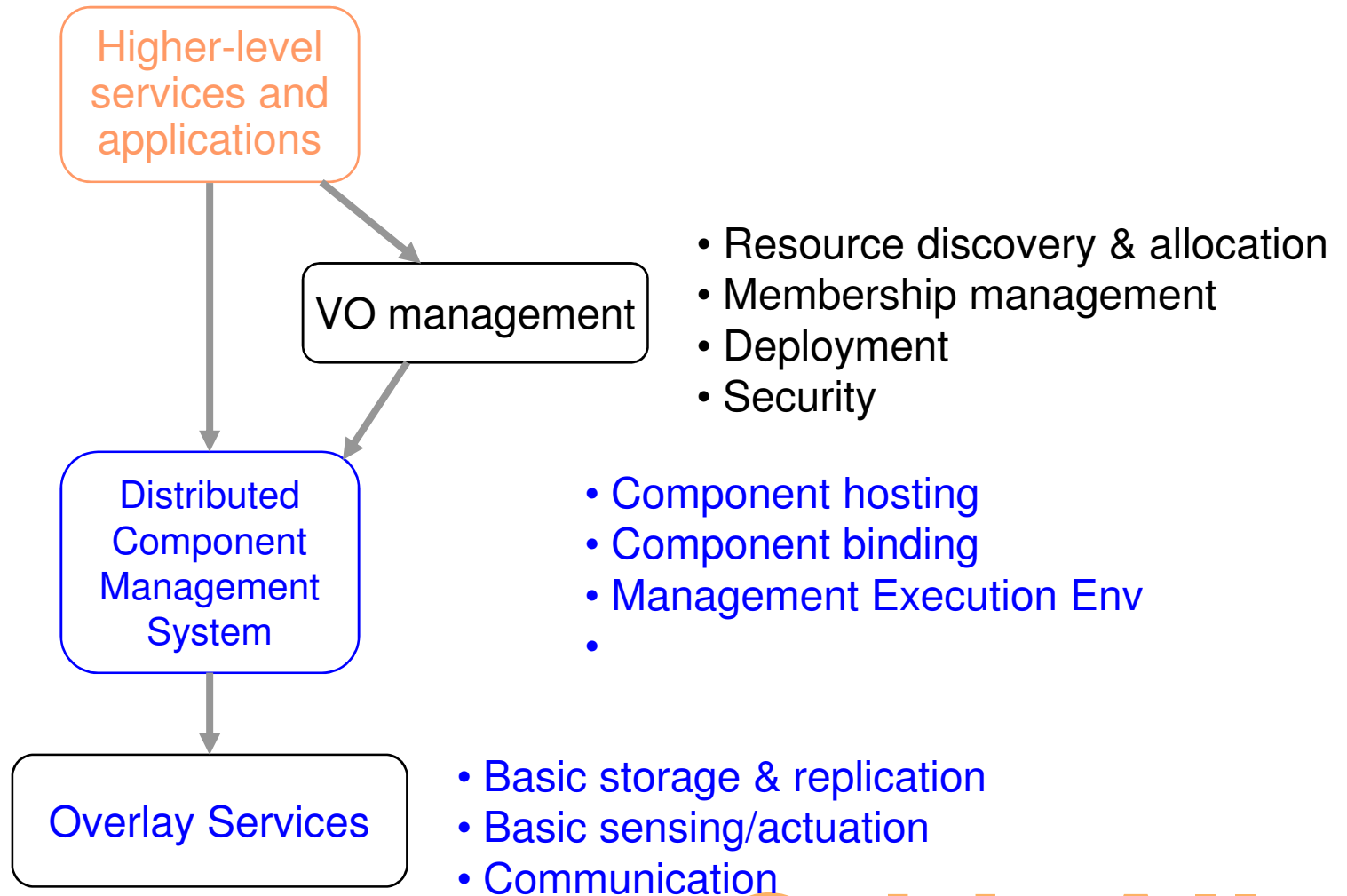
Grid4All

Architectural layering



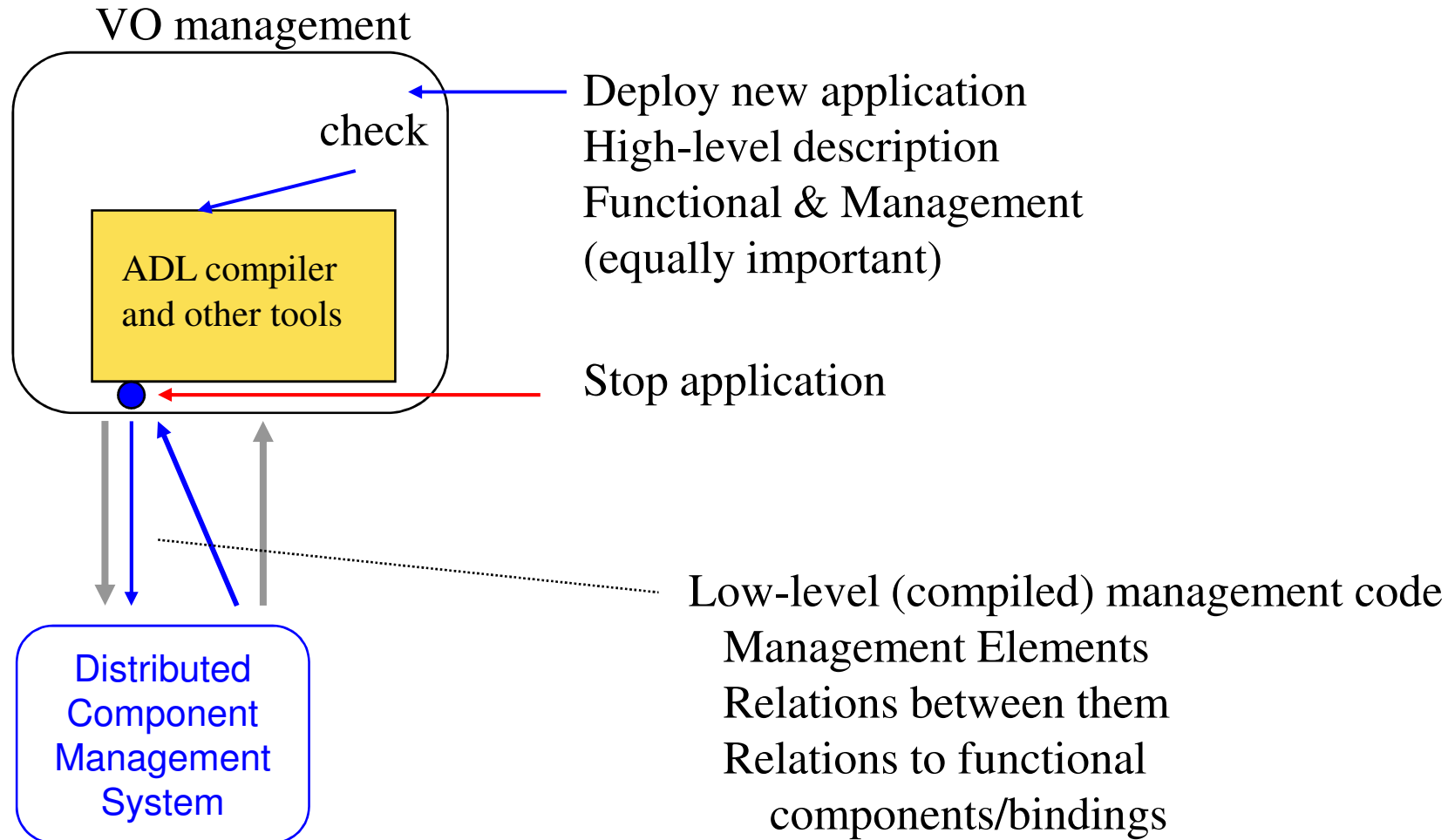
Grid4All

Core VO support



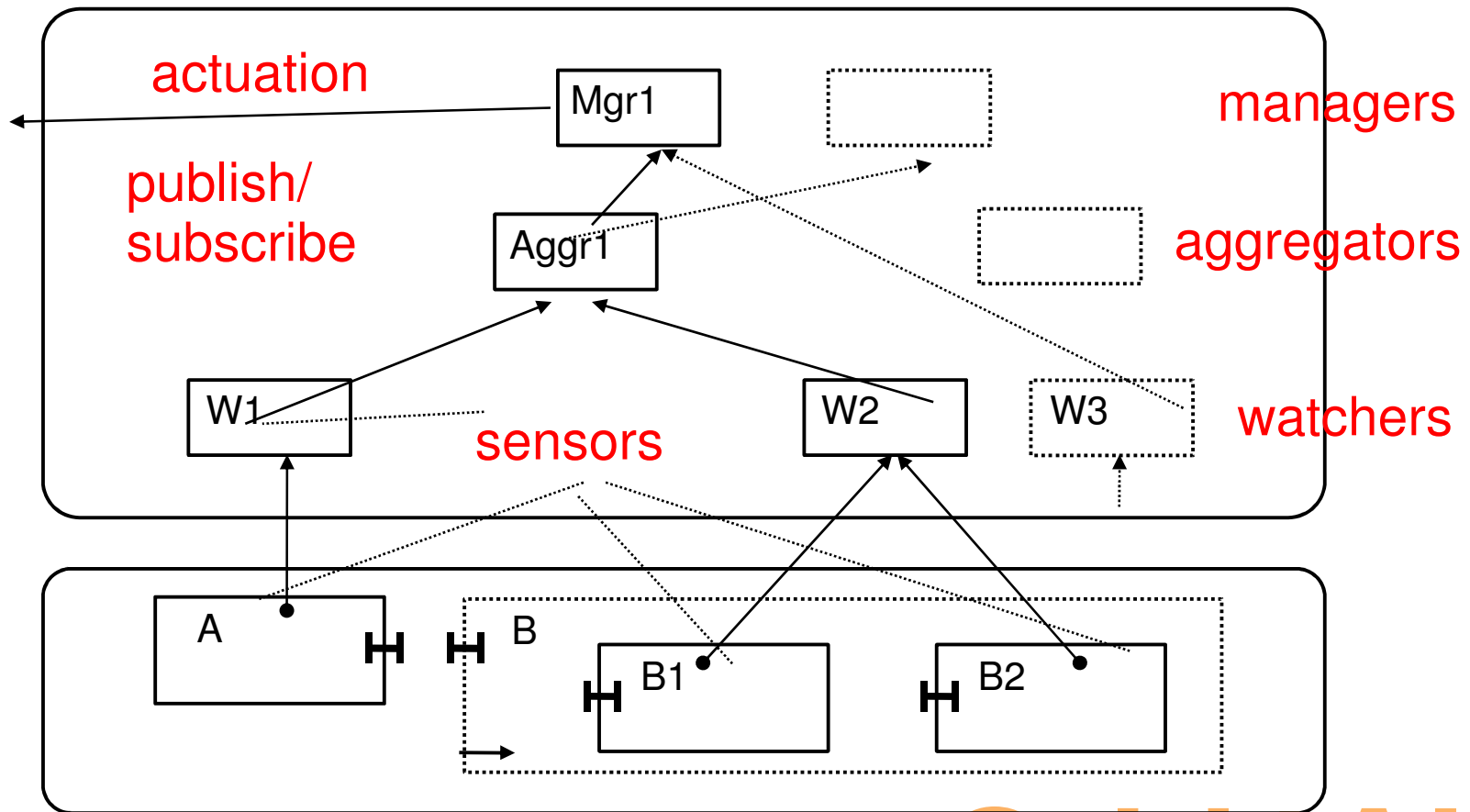
Grid4All

New Application/Service



Grid4All

Architecture of Self-Management of DCMS Applications



Grid4All

Elements of Self-Managing Applications with DCMS

Management Elements (MEs):

- *watchers* monitor status of elements of the architecture
- *aggregators* maintain some type(s) of information about application state, using a set of watchers
- *managers* collect information on all aspects of application state using watchers and aggregators, and change the architecture as needed

Sensors provide information about status of individual components

- application-specific or provided by DCMS

Communication between MEs

Communication between MEs

- Event-based, unidirectional
- MEs are *subscribed* to certain (specified) types of events produced by certain Mes
 - can be seen as a *restricted* form of publish/subscribe
 - .. but also as a *binding* with *unidirectional* communication
- sensors also produce events consumed by watchers

Architecture Representation in Self-Management Code

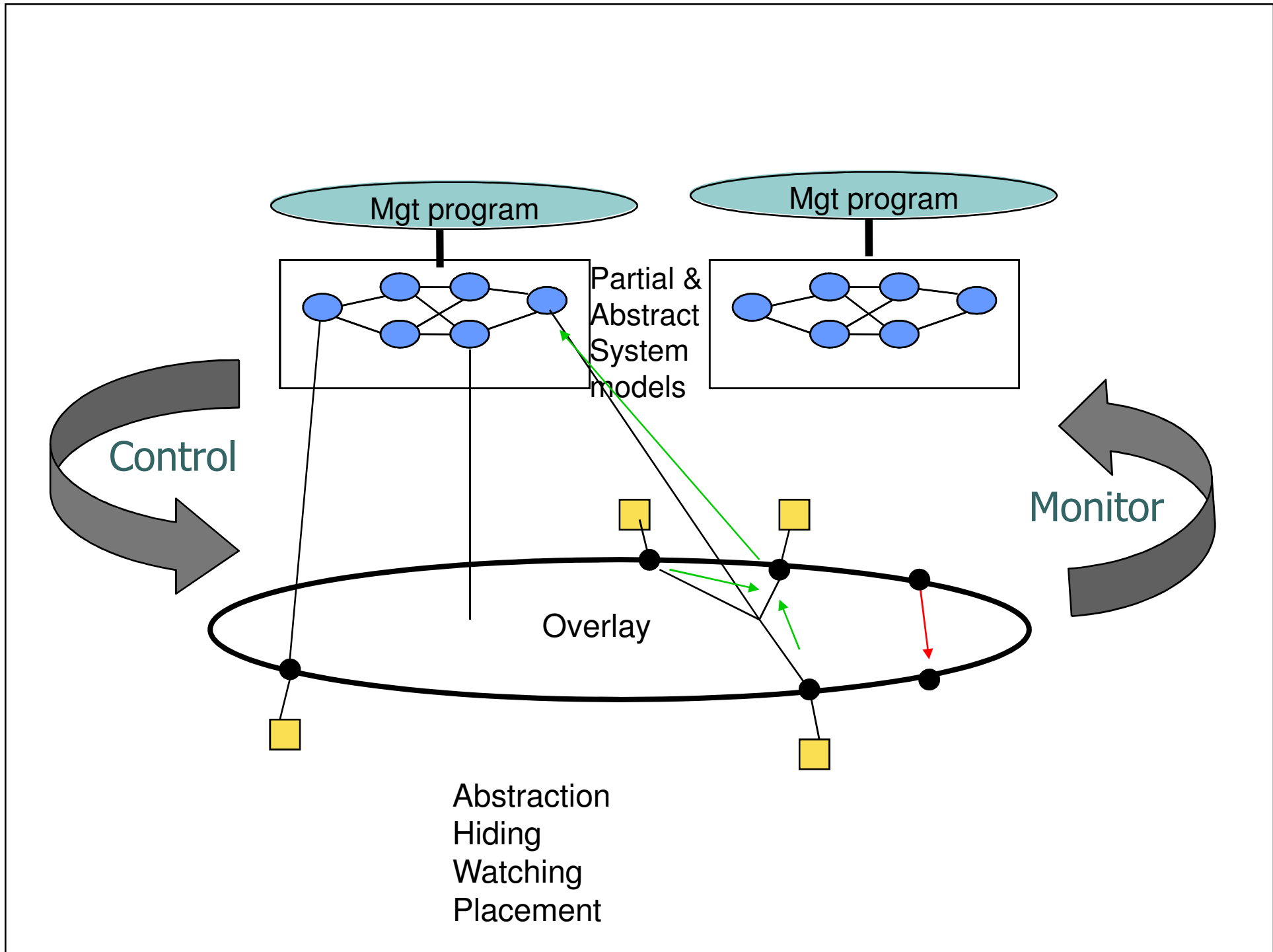
Elements of the architecture – components, bindings, groups – are identified by unique DCMS-wide *identifiers (ID"s)*

- DCMS API is defined in terms of those ID"s

May encapsulate the location of the entity

For some elements of the architecture their locations can be changed

- Using *references* that are reliably stored by DCMS
- Network transparent



Groups

Group abstractions

- Component Groups

- One to * (all, any) bindings

- Network transparent

- Important for efficiency/robustness (dist. algorithm sense)

Sensing

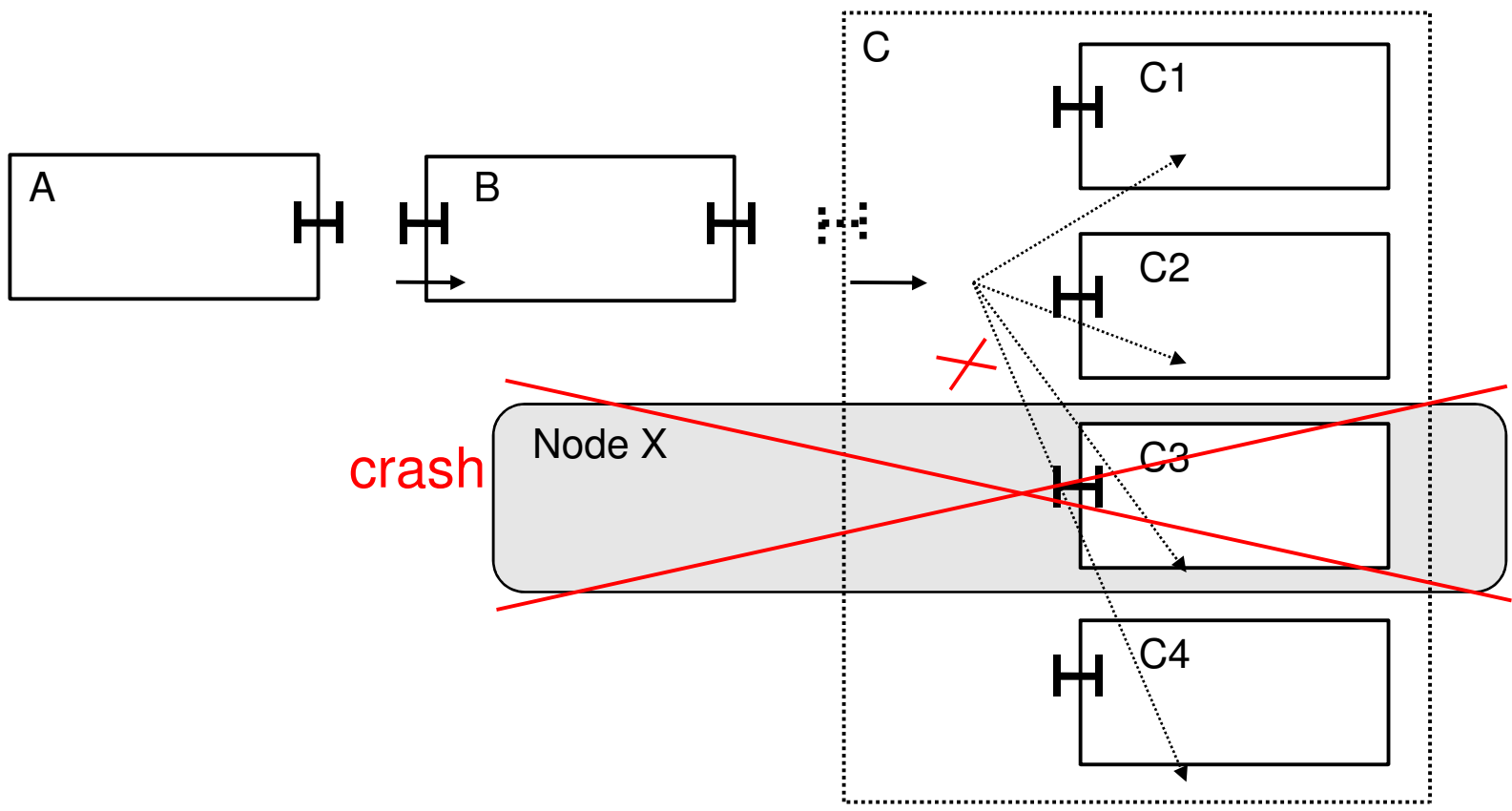
- Watchers sense groups as a whole

- Programmer specifies a sensor implementation for a given group

- Individual sensors are installed for group members as the latter come and go

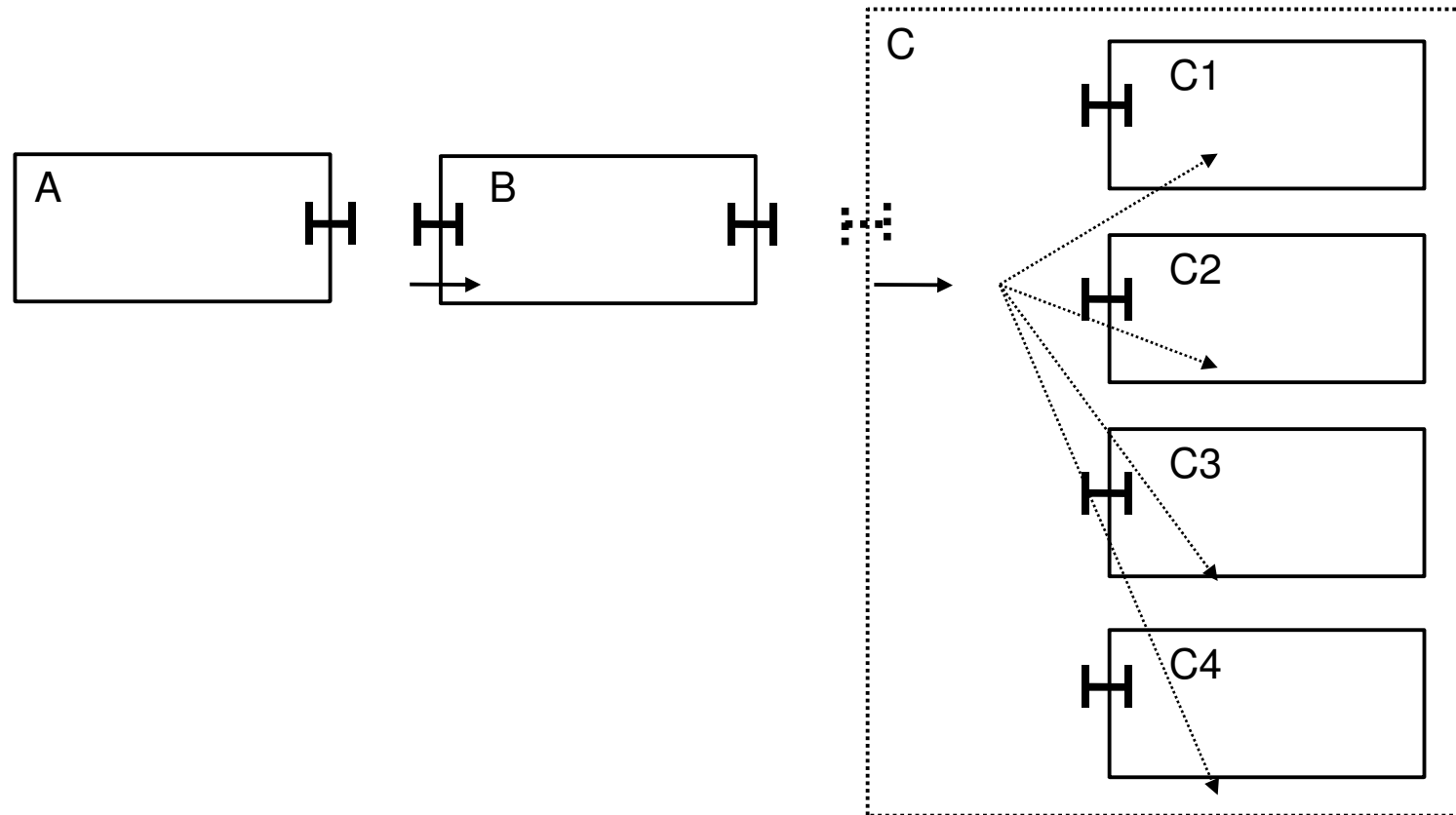
Grid4All

Groups and One-to-* Bindings (III)



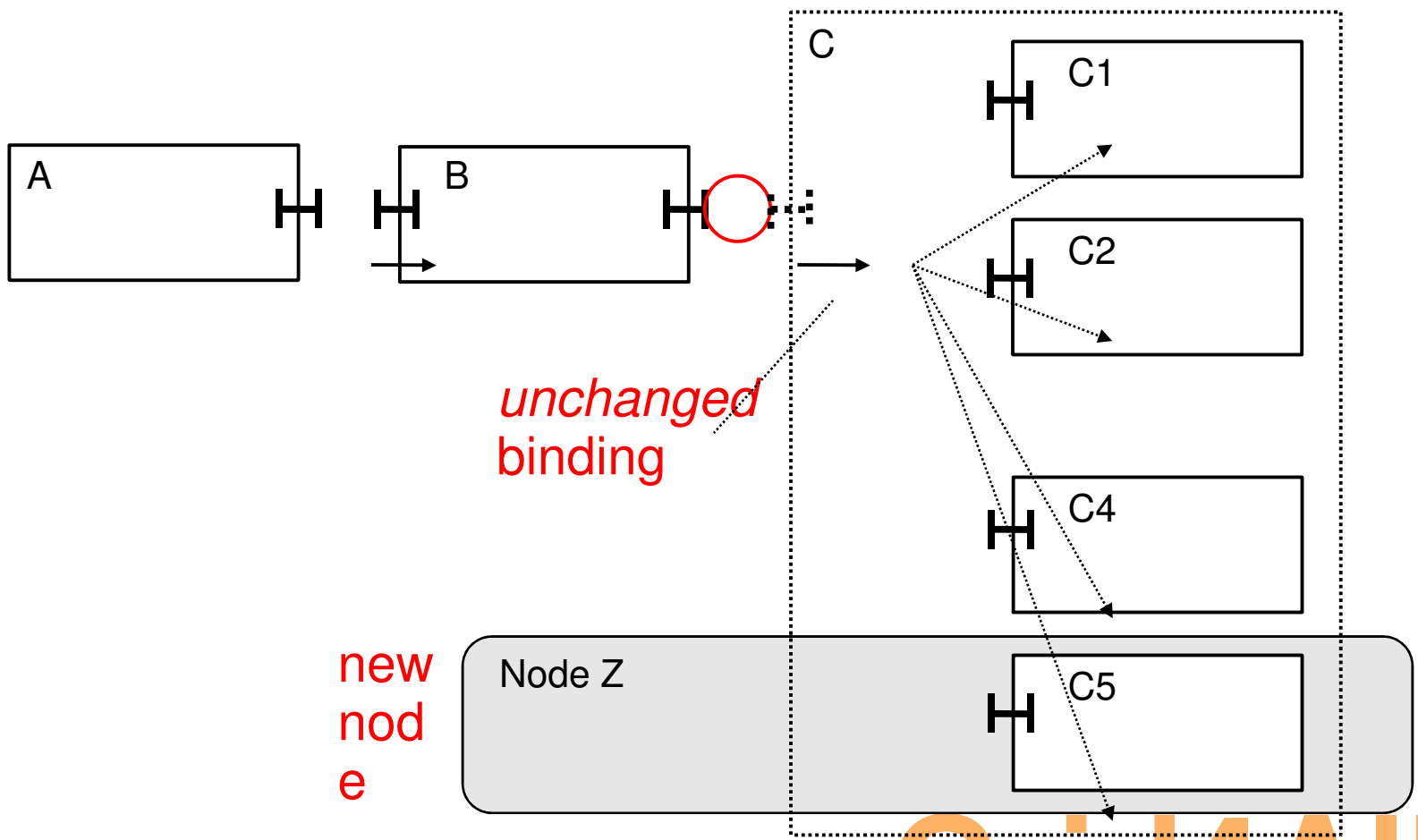
Grid4All

Groups and One-to-* Bindings (I)



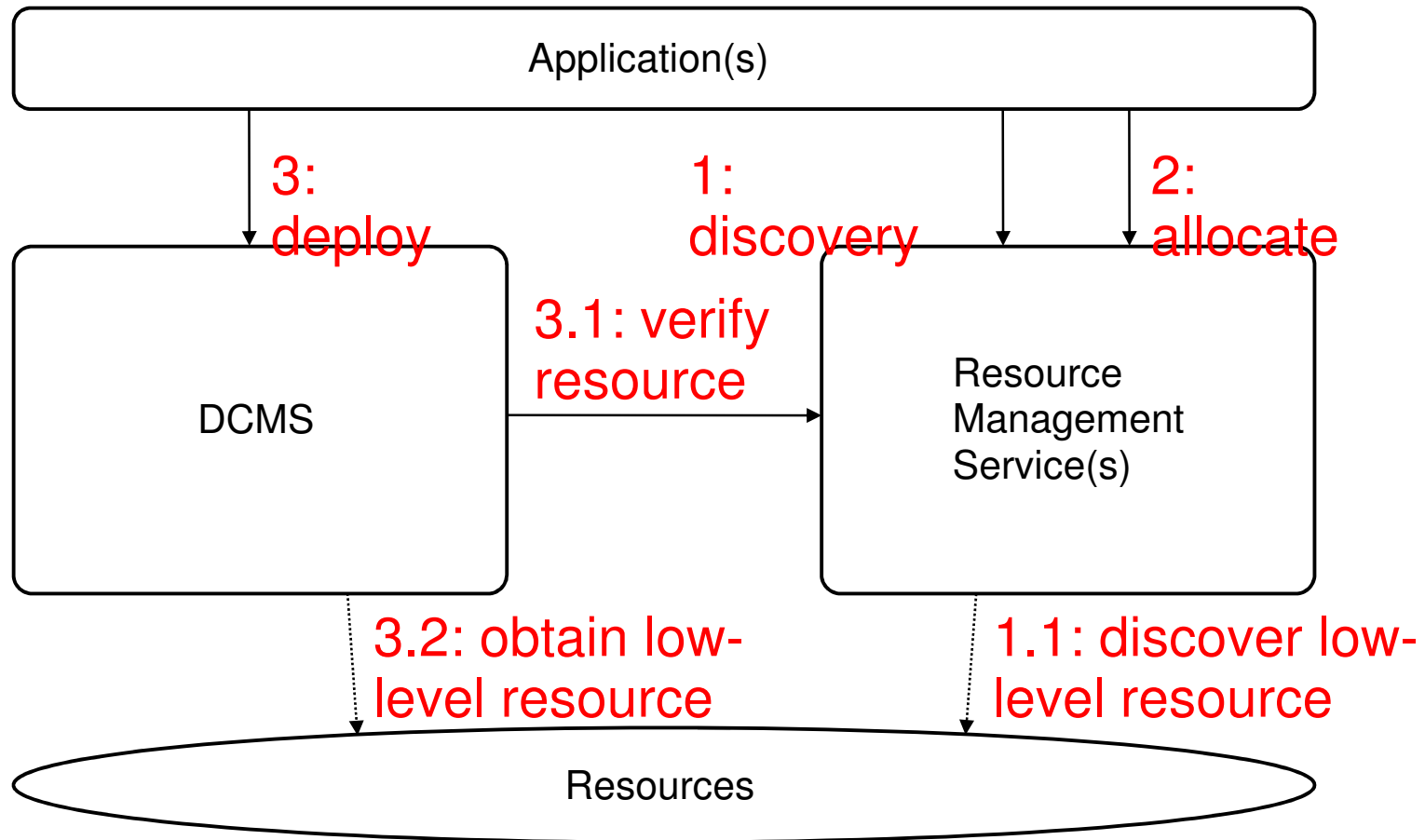
Grid4All

Groups and One-to-* Bindings (IV)

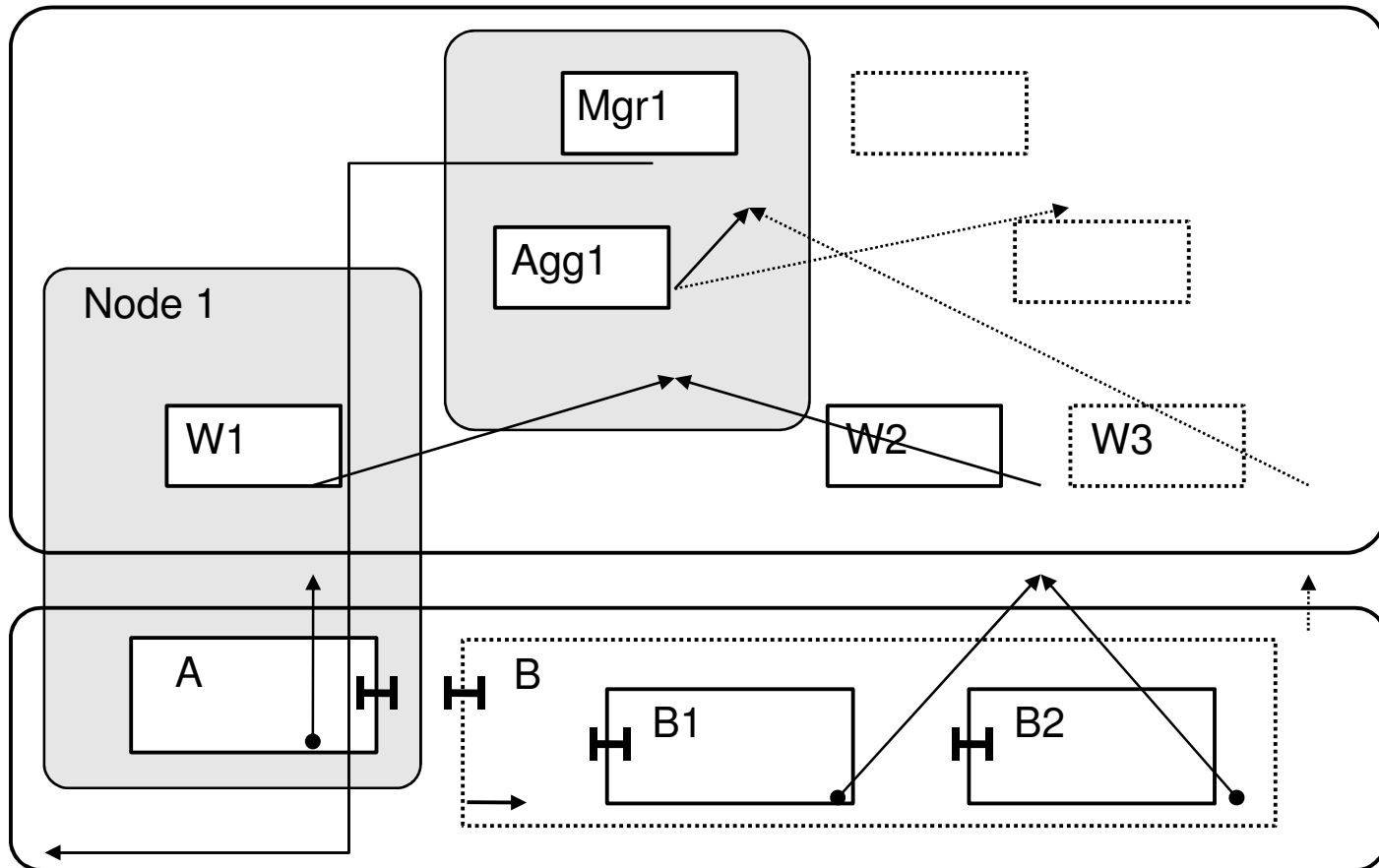


Grid4All

Resource Discovery and Allocation for Applications with DCMS



Co-location of MEs



Grid4All

Advantages of DCMS (I)

Relieves application programmer from low-level details of managing application architecture

- For both functional and non-functional parts
- Provides the `deploy/undeploy` abstractions
- Provides *component sensing abstractions*

Explicitly separates component management and self- code from resource management*

- *DCMS can be used in different environments*

Advantages of DCMS (II)

Is network-transparent yet network-aware

- Enables programming functional and non-functional parts independently of deployment scenarios
- Facilitates dealing with churn
 - DCMS will be able to move MEs when nodes with MEs are about to leave, and
 - move MEs for load balancing when new nodes join
- Distribution of elements of self-management facilitates application scalability
- Allows to control co-location of MEs

Advantages of DCMS (III)

Provides the network-transparent view of system architecture

- Architecture representation is available everywhere
- .. and will be made fault- and churn-tolerant

We believe the framework can be extended for flexible and application-transparent replication of management elements

- giving the programmer a conceptually simple platform for programming *robust* self-management

Grid4All

Grid4All

Deploying Application Self-Management: “Initial Script”

```
DCMService myDCMS;
```

```
ComponentId compA = myDCMS.deploy(ResourceA, ImplA, "A");
```

```
ComponentId compB = myDCMS.deploy(ResourceB, ImplB, "B");
```

```
ManagementElementId manager =
```

```
    myDCMS.deploy(ManagerComponent, {compA, compB});
```

```
ManagementElementId aggregator =
```

```
    myDCMS.deploy(AggregatorComponent, {compA, compB});
```

```
(void) myDCMS.subscribe(aggregator, manager, "status");
```

```
(void) myDCMS.sense(compA, aggregator, "componentFailure");
```

```
(void) myDCMS.sense(compB, aggregator, "componentFailure");
```

Deploying Self-Management after ADL-based Application Deployment

```
DCMService myDCMS;
```

```
// ComponentId compA = myDCMS.deploy(ResourceA, ImplA, "A");  
// ComponentId compB = myDCMS.deploy(ResourceB, ImplB, "B");  
ComponentId compA = myDCMS.lookup("A");  
ComponentId compB = myDCMS.lookup("B");
```

```
ManagementElementId manager =  
    myDCMS.deploy(ManagerComponent, {compA, compB});  
ManagementElementId aggregator =  
    myDCMS.deploy(AggregatorComponent, {compA, compB});
```

```
(void) myDCMS.subscribe(aggregator, manager, "status");  
(void) myDCMS.sense(compA, aggregator, "componentFailure");  
(void) myDCMS.sense(compB, aggregator, "componentFailure");
```

Grid4All