



Architect of an Open World™

Making RMI clustering easy with CMI

Loris BOUZONNET (Loris.Bouzonnet@bull.net)

LIBERATE IT

OW2
Consortium

Leading Open Source Middleware

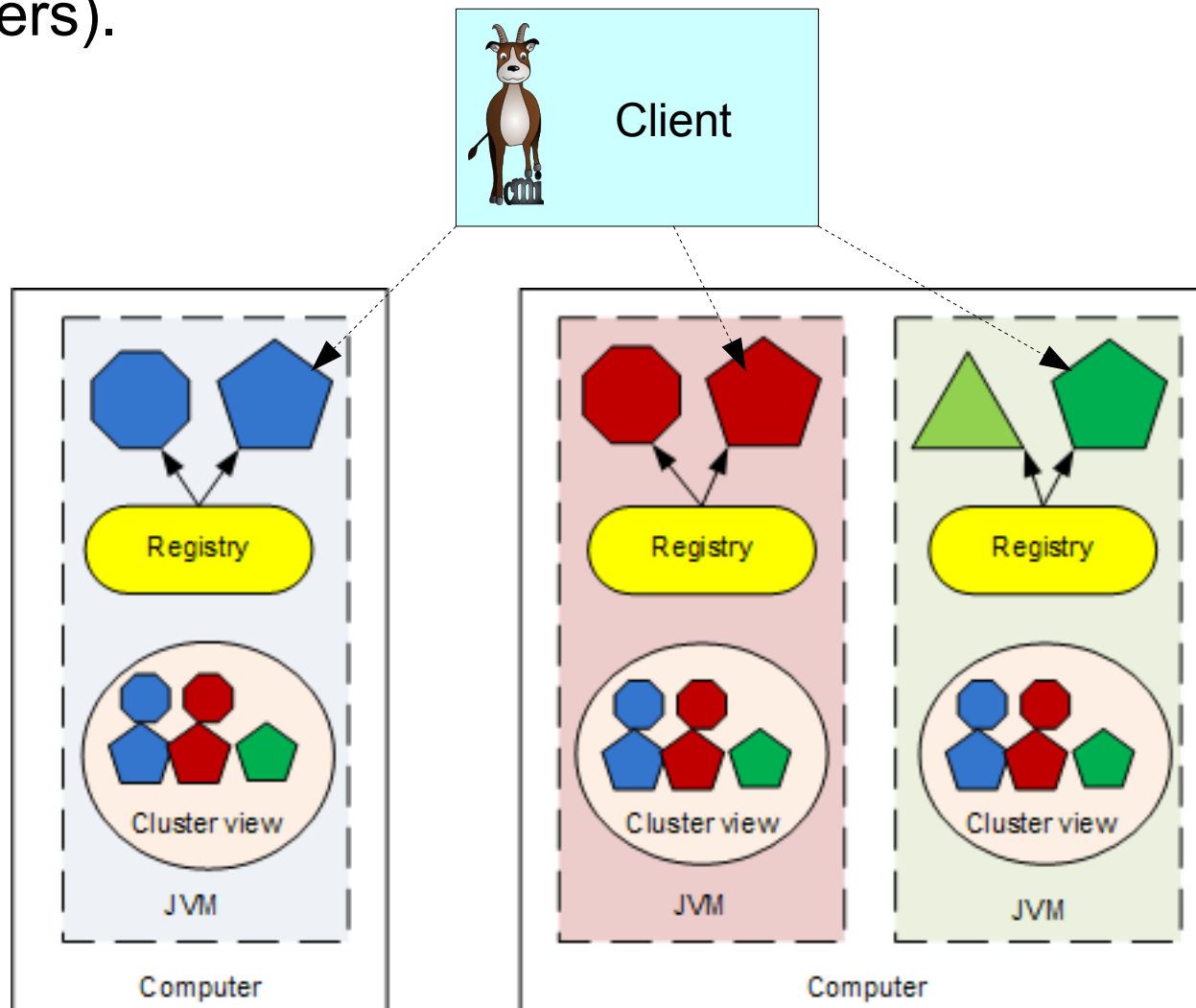


- A short history
- Toward the project CMI
- The project CMI
- Features
- Architecture
- Related work
- Incoming works
- Demo

Introduction



- CMI defines cluster of RMI objects (and not of servers).





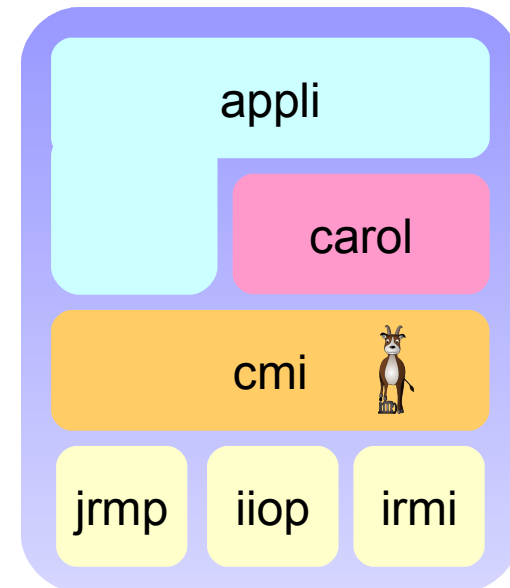
Overview of CMI v1

- Started in 2005 to provide clustering of EJB2.
 - Not yet a standalone project but rather an extension for JOnAS
- Drawbacks:
 - Too static
 - Bad Maintainability
- Need an independent mechanism to support EJB3.



Beyond the clustering

- CMI v2 started from scratch in 2007 to become a new OW2 project.
- Its domain application is no more limited to clustering
 - Distributed applications
 - SOA
- Requirements:
 - Support any RMI objects, whatever the protocol
 - Easy to use for developers and administrators
 - Support of dynamic changes of topology and weights
 - Easy to extend and follow standard evolutions





Project management

- Software project management
 - Maven
- Version control
 - OW2 SVN: `svn://svn.forge.objectweb.org/svnroot/cmi`
- Test
 - TestNG
- Continuous integration
 - OW2 Bamboo: <http://forge.ow2.org/bamboo/>
- Documentation
 - Docbook
- Packaging
 - OSGi or not
 - Modules: admin, jndi, rpc, client, server/JGroups, server/JMS

The project CMI



Community

Leader



HA algorithms



Selfware project



Group com layer



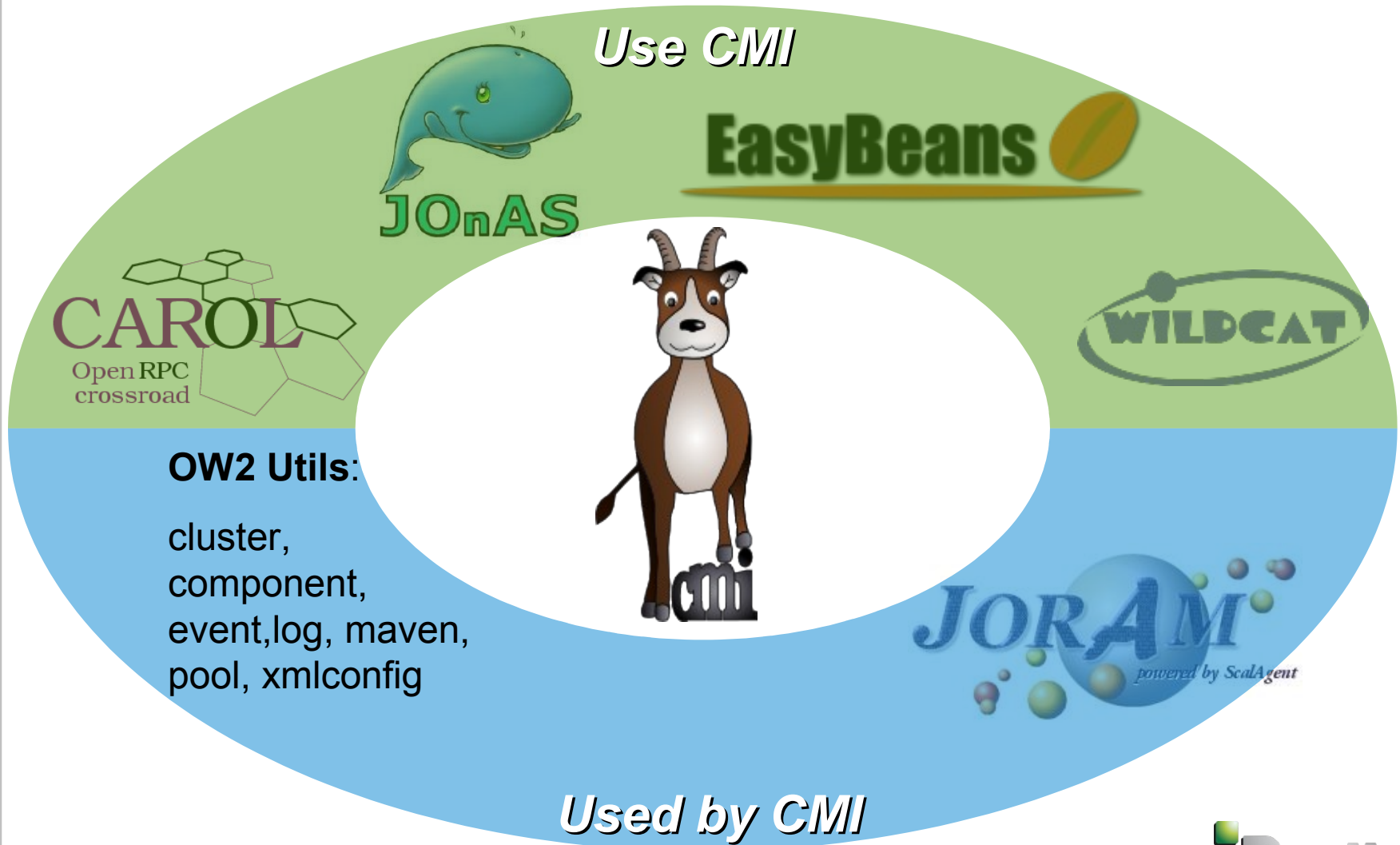
Dynamic replication



The CMI project



Relationship with some other OW2 projects





Defining a clustered object

`@Cluster(name="my_cluster")`



Tell that instances of Hello
will be clustered

`@Policy(RoundRobin.class)`



Tell how to access them

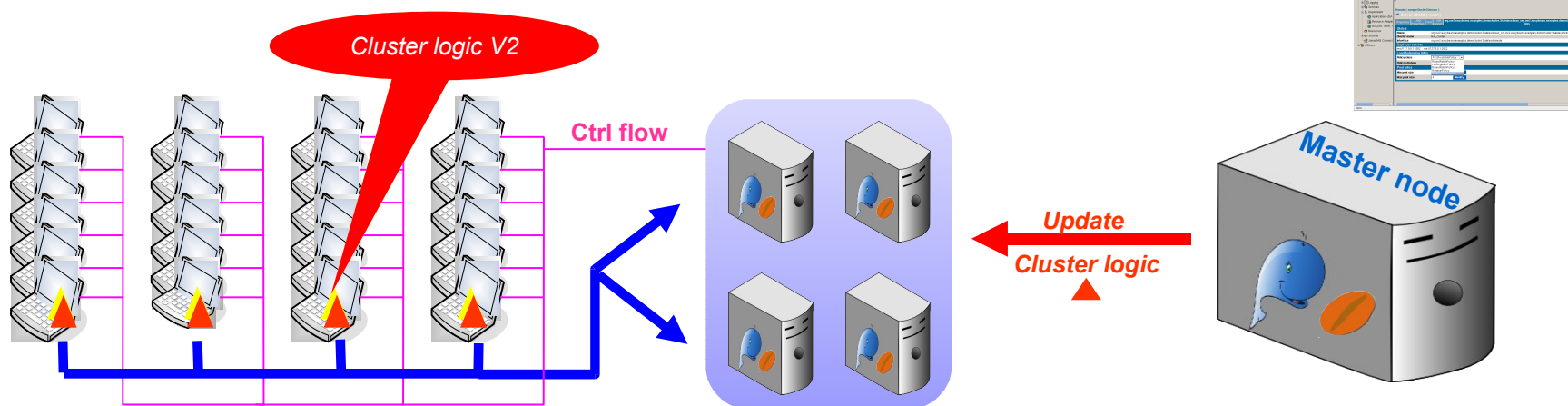
```
public class Hello implements HelloItf, Remote {  
    public void sayHello() {  
        System.out.println("Hello!");  
    }  
}
```

Features



CMI added value for every RMI objects

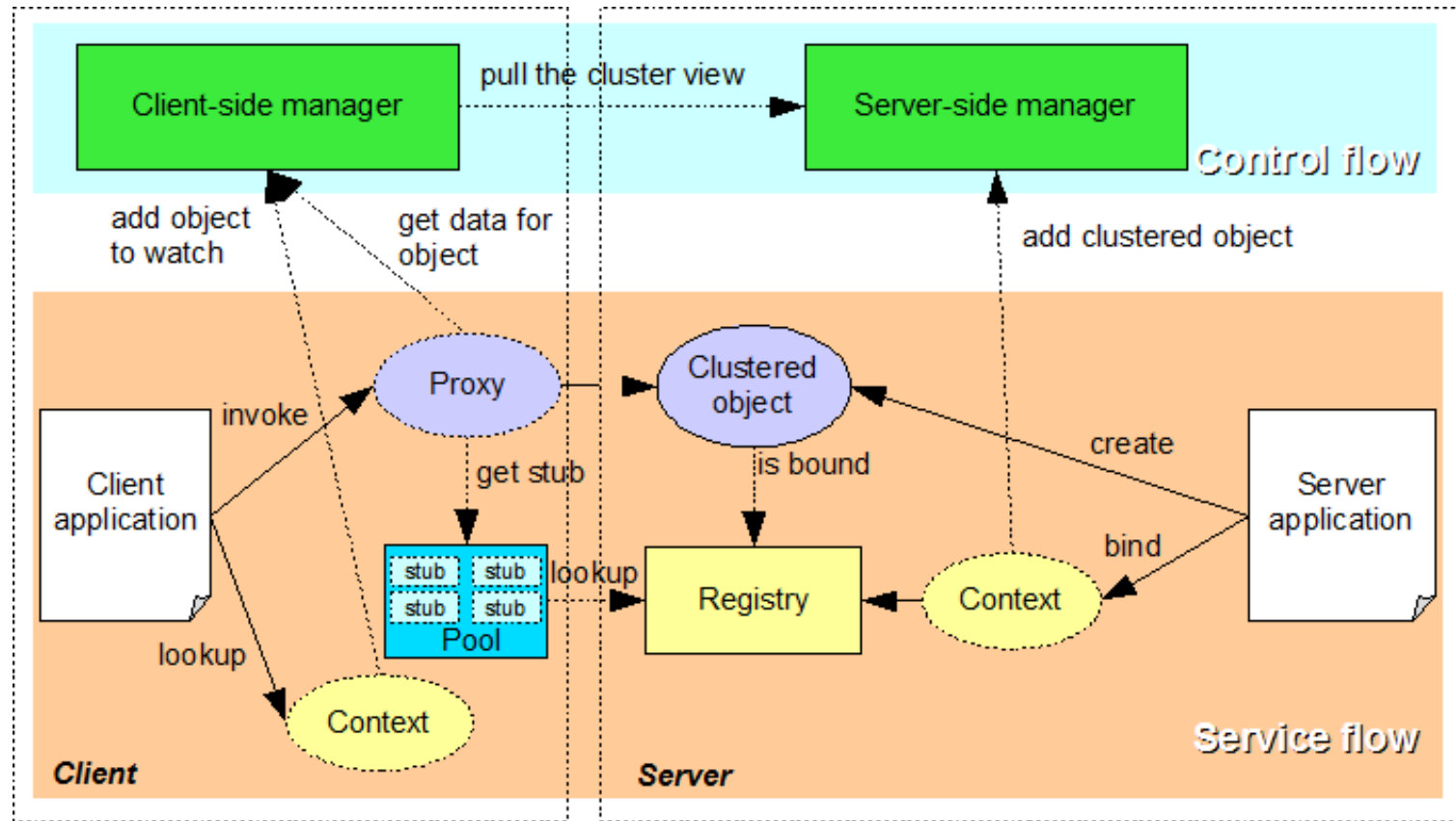
- Dynamic update of the cluster configuration
- Easy administration
 - Update of the algorithm from the JOnAS admin console, JMX or OSGi.





Performance oriented

Separation of control and server flows





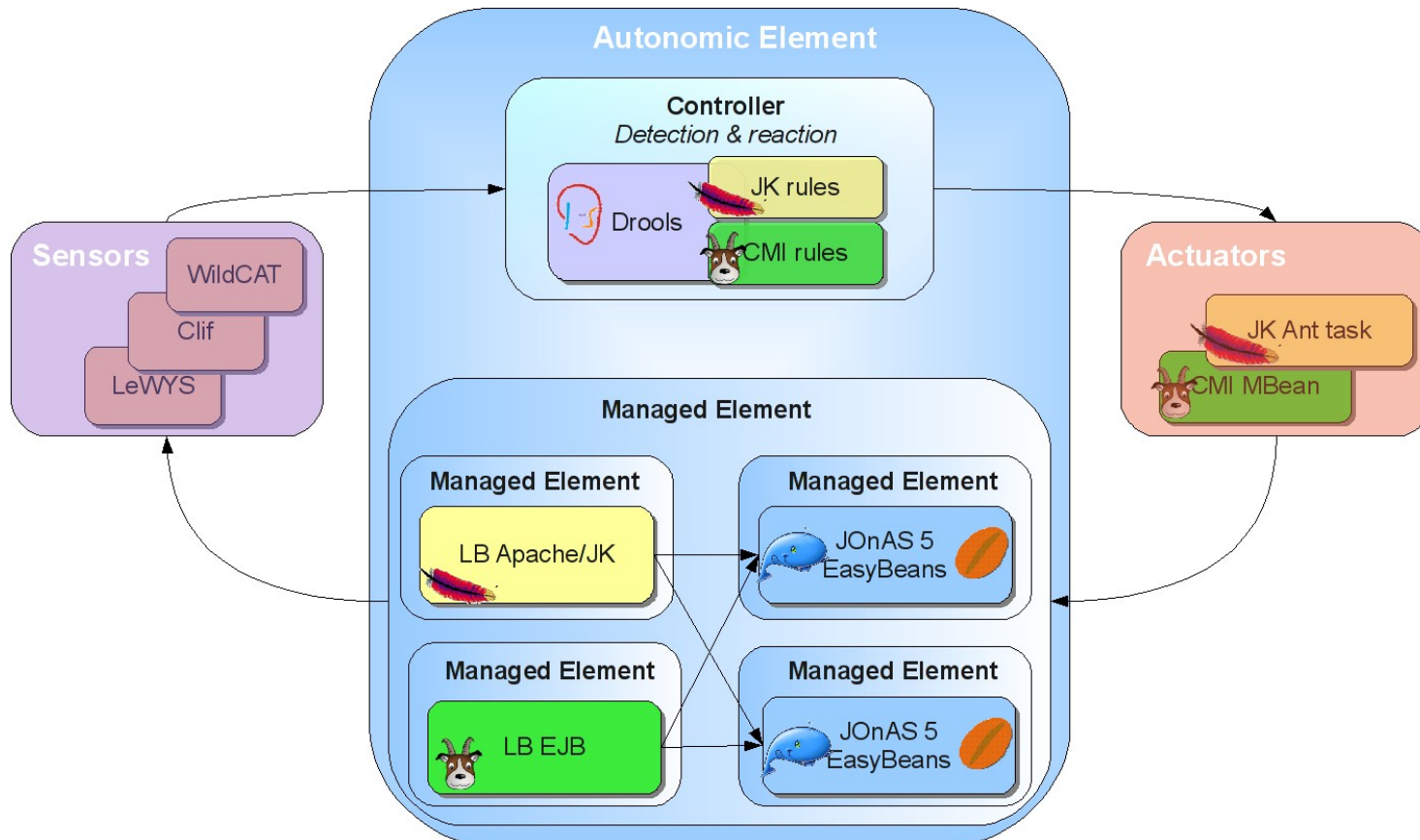
2 available implementations

- JGroups
 - Multicast/Unicast
 - The initial implementation: most mature
 - Flat cluster views
- JMS (e.g. Joram)
 - Unicast
 - More lightweight
 - Allow to define hierarchical cluster views.
 - Exploited by WildCAT



Self-optimization of CMI clusters

- Feature in the autonomic loop powered by JASMINe
 - Provides rules to configure CMI
 - Computes load factors from JASMINe Events





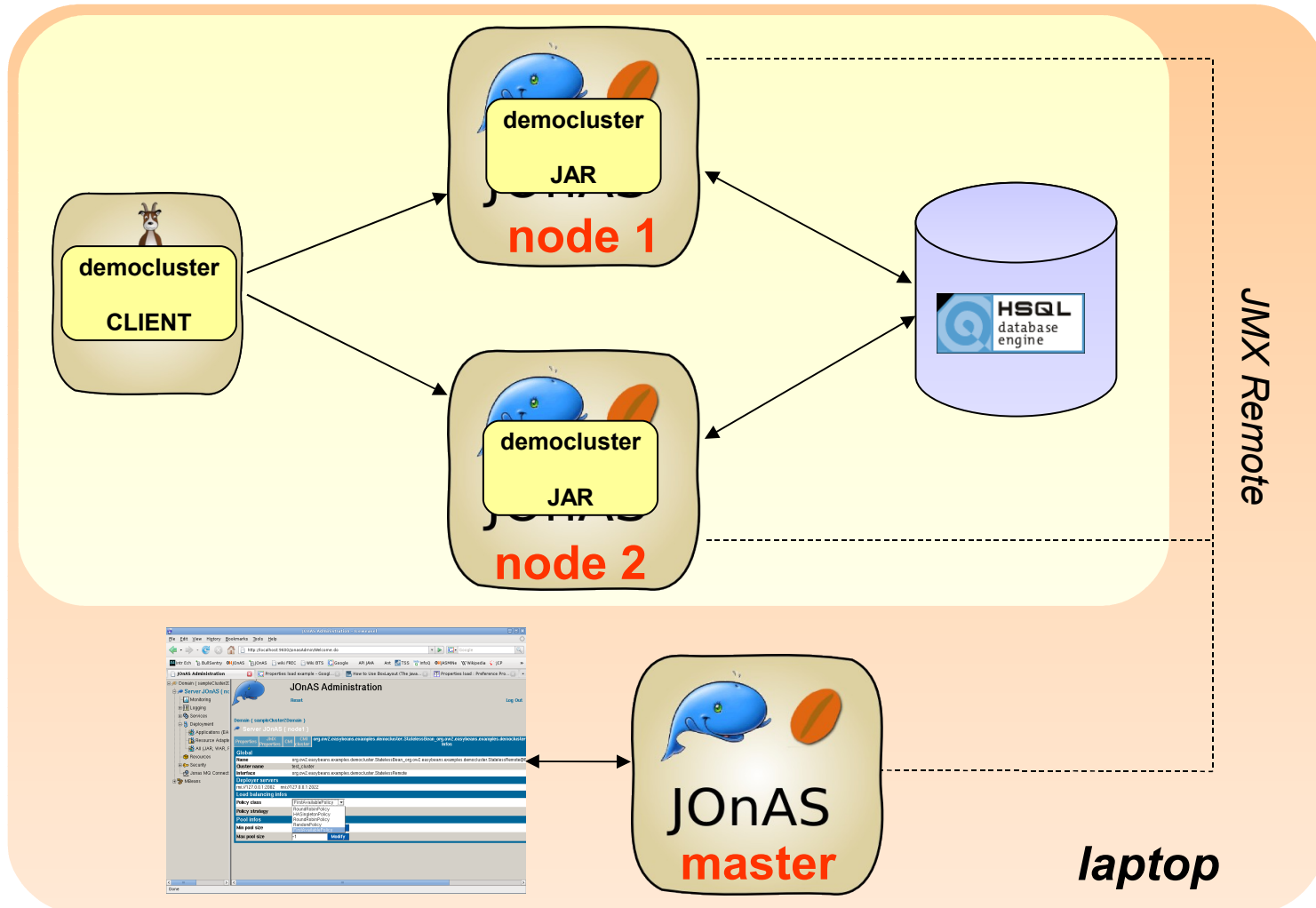
- Platform to launch tests of clusters in the continuous integration
- Terracotta support
- Computation of the cluster view according to an identity
- Dynamic deployment of client interceptors
- Dynamic replication



- CMI already provides personalities for EJB2 and EJB3
- Easiness of the administration thanks to the dynamism of CMI



Architecture





Remote interface

```
package org.ow2.easybeans.examples.democluster;

public interface StatelessRemote {
    public String getNodeName(String msg);
}
```



Bean

```
@Stateless
@Remote(StatelessRemote.class)
@Cluster(name="test_cluster")
@Policy(RoundRobin.class)

public class StatelessBean implements StatelessRemote {
    public String getNodeName(String msg) {
        String jonasInstanceName = "unknown";
        // get the node name
        try {
            JProp jp = JProp.getInstance();
            jonasInstanceName = jp.getValue("jonas.name");
        } catch (Exception e) {
            //
        }
        System.out.println(msg);
        return jonasInstanceName;
    }
}
```

Admin name



Client

```
public static void main(final String[] args) throws Exception {
    Context initialContext = getInitialContext();
    StatelessRemote statelessBean = (StatelessRemote) initialContext
        .lookup("org.ow2.easybeans.examples.democluster.StatelessBean"
            + "_" + StatelessRemote.class.getName() + "@Remote");
    while (true) {
        i++;
        msg="Request[T=" + pid + ", HD="+ dateFormat.format(new Date())
            + ", S=" + i + "];"
        // invoke the EJB
        String nodeName = statelessBean.getNodeName(msg);
        System.out.println(msg + " -> " + nodeName);
        Thread.sleep(200);
    }
}
```



Scenario

- 1) Load-balancing checking
 - Default policy = round-robin

- 2) Dynamic change of the load-balancing policy
 - To first available

- 3) Fail-over and server restart

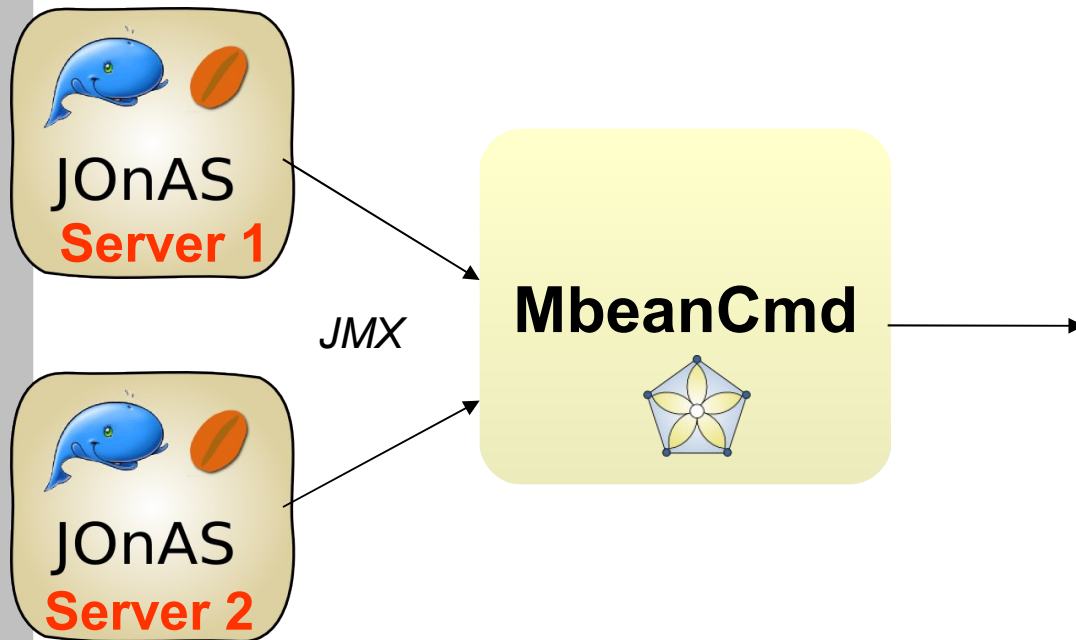
- 4) Go back to the RoundRobin policy

- 5) Smooth stopping of a server
 - Blacklist

Demo



Monitoring with MbeanCmd





Architect of an Open World™

LIBERATE IT