JADE: the new kernel and last developments

Giovanni Caire

JADE Board Technical Leader

giovanni.caire@tilab.com

Pisa 2004

Summary

- JADE
- The JADE Board
- The new Kernel
  - Ideas and motivations
  - Main elements
  - An example
  - Advanced
- The security add-on
- Roadmap

Confidential

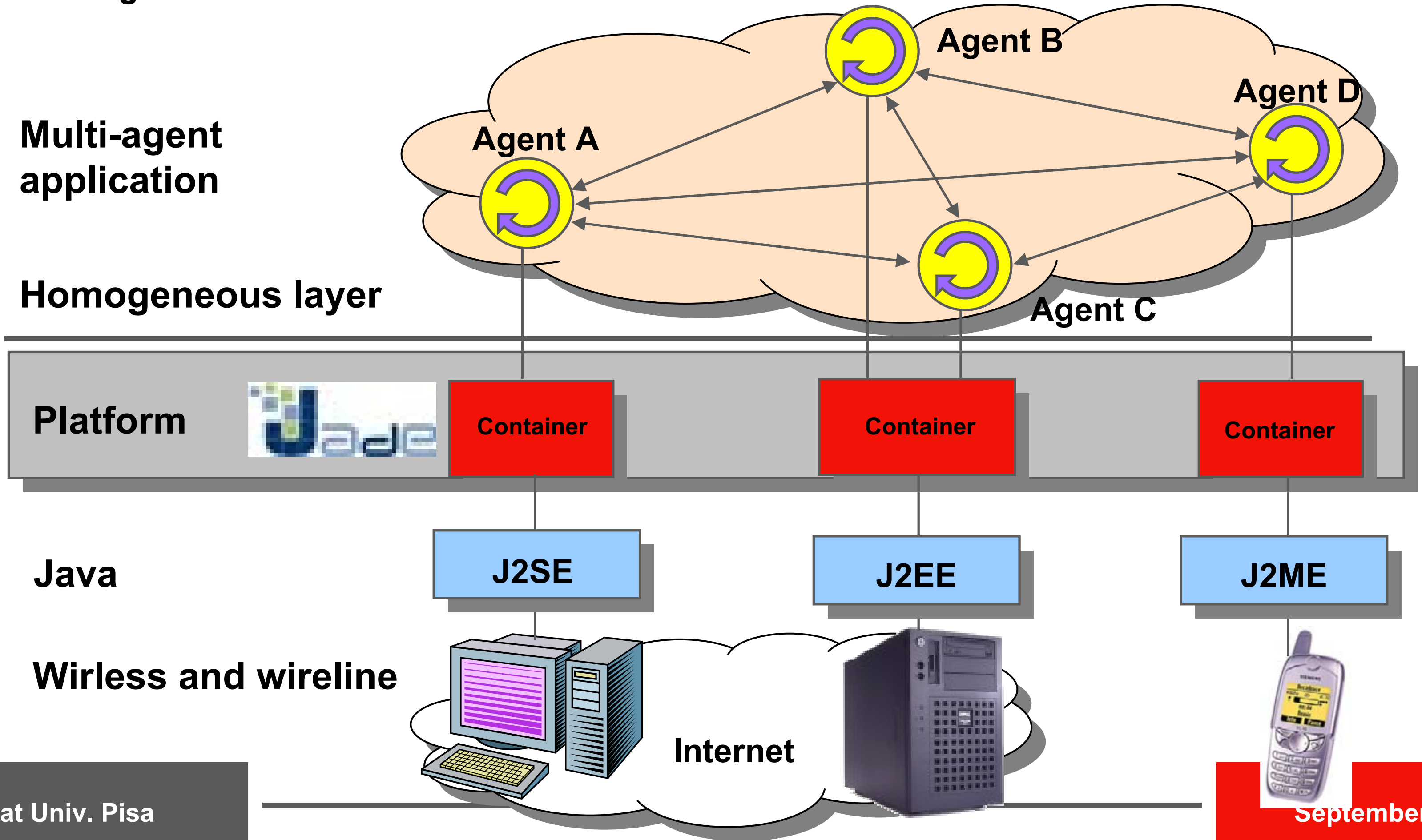
The JADE platform

- JADE is a middleware that facilitates the development of Multi Agent Peer-to-Peer applications.
- Full Java
- Runs on all JVM from J2EE to J2ME MIDP1.0
- Distributed in Open Source under the LGPL license
- Downloadable from http://jade.tilab.com
- The JADE Project was initiated by TILAB and is now governed by an international Board

Confidential

The architectural model

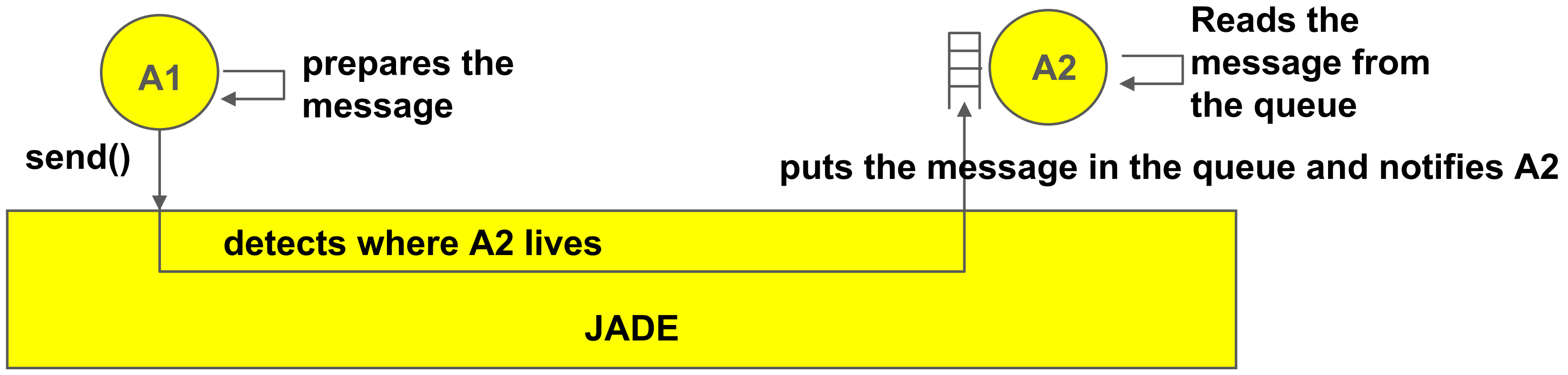
- ✓ A JADE-based application is composed of a collection of active components called Agents
- ✓ Each agent has a unique name
- ✓ Each agent is a peer since he can communicate in a bidirectional way with all other agents
- ✓ Each agent lives in a container



Confidential

The communication model

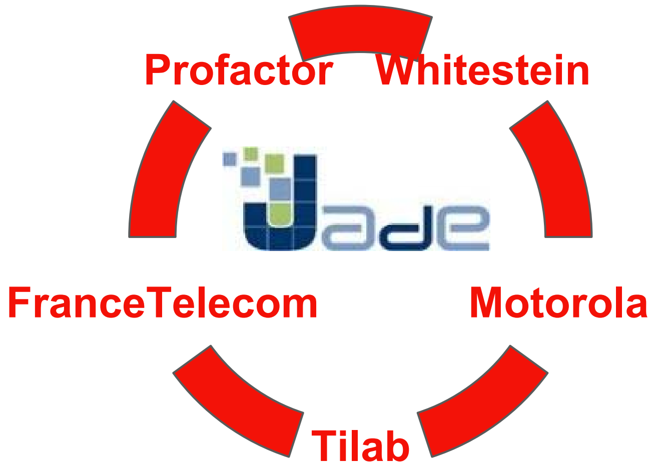
- Based on asynchronous message passing
  - Each agent has a sort of mailbox where messages for that agent are inserted. When a message is put in the mailbox the agent is notified. However it will be up to him to decide if and when to read the message and how to react to it.


A1
prepares the message
send()
A2
Reads the message from the queue
puts the message in the queue and notifies A2
detects where A2 lives
JADE

JADE Board

- Founded on March 2003 by TILAB and Motorola
  - as a follow-up of their collaboration in the LEAP project
  - as a not-for-profit contractual consortium among the Members
- Mission
  - Promote, govern, and implement the evolution of JADE
- Goal
  - JADE adoption by the mobile industry as a standard middleware for mobile Peer-To-Peer intelligent agent applications completely interoperating on different terminals and networks

Board Members (May 2004)



Confidential

Summary

- JADE
- The JADE Board
- The new Kernel
  - Ideas and motivations
  - Main elements
  - An example
  - Advanced
- The security add-on
- Roadmap

Confidential

Ideas and motivations

Requirements

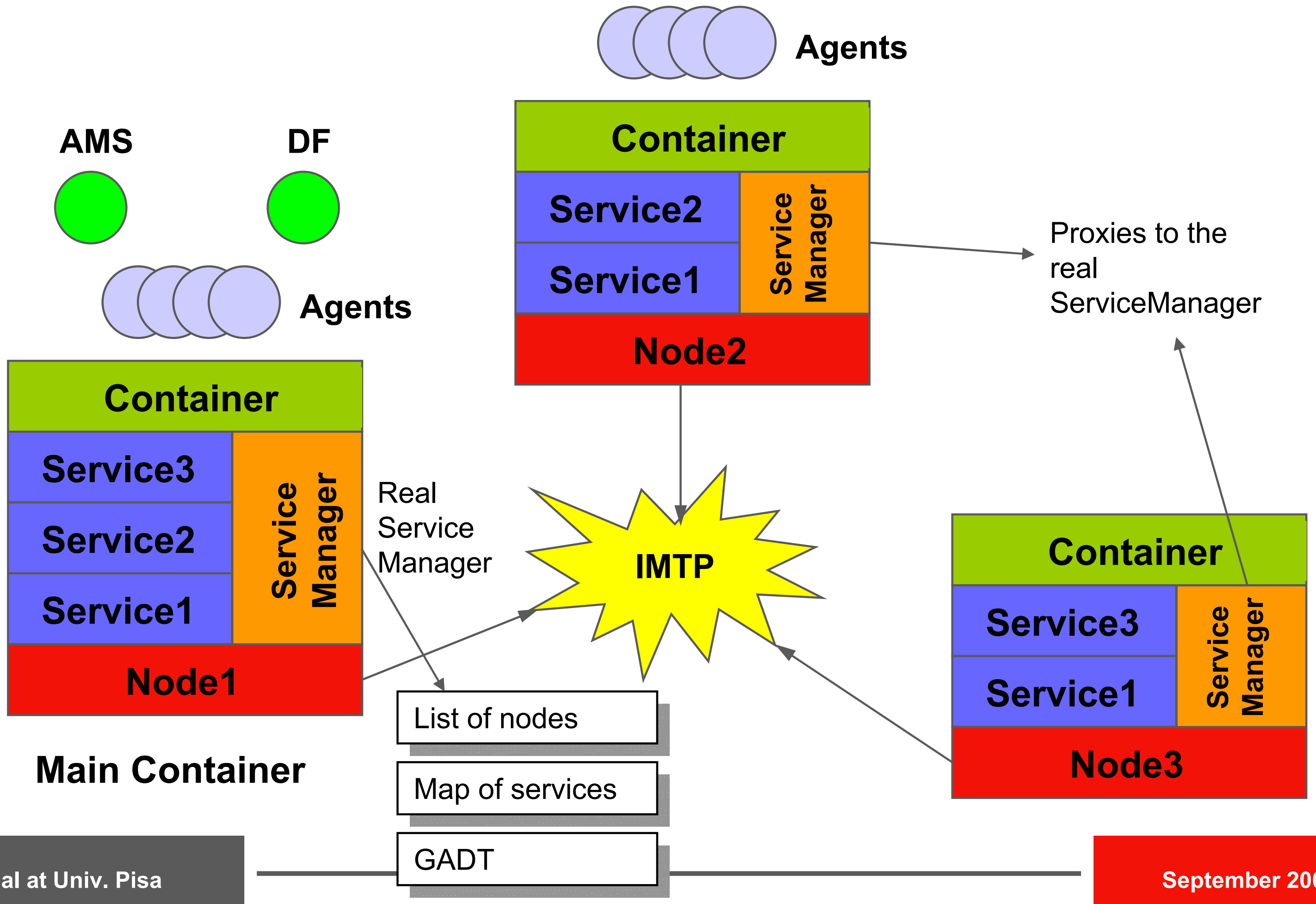
- 1) Fine grained granularity of platform features
- 2) Open-ended set of features
- 3) Distribution
- 4) Flexible deployment strategy to target the hybrid wireline/wireless environment

Main abstractions

- Service
- Node

Confidential

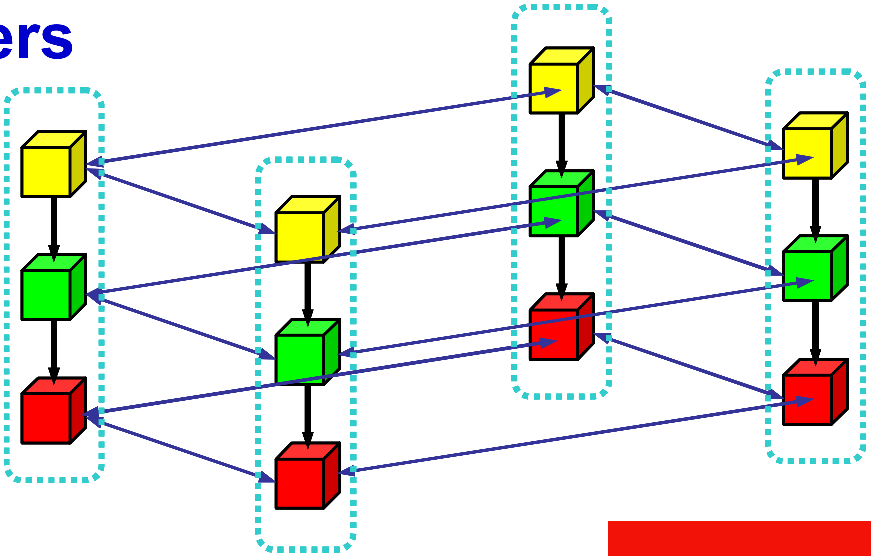
Architecture overview



Confidential

The Distributed coordinated filters

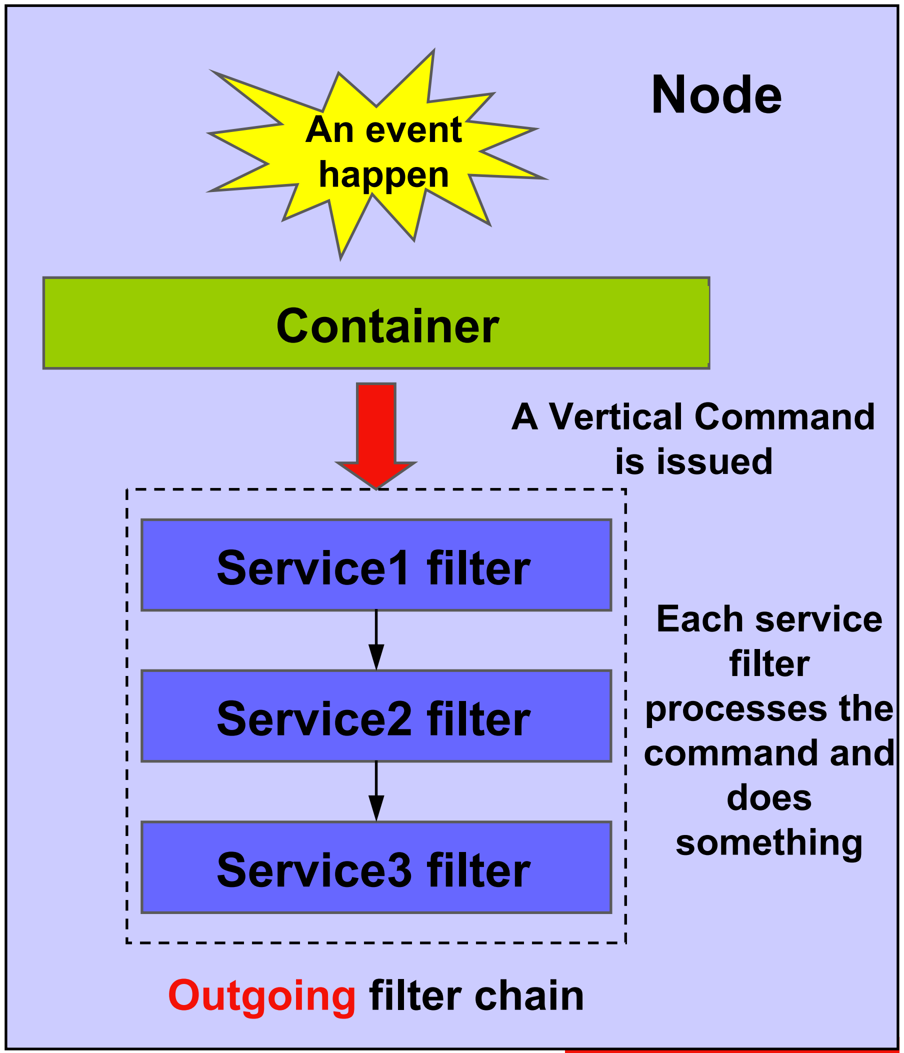
- Inspiration from Aspect Oriented Programming
  - Separation of concerns + Aspect Weaving
  - Composition Filter approach: Each object is provided with
    - An incoming filter chain whose filters are invoked whenever the object receives a method call
    - An outgoing filter chain whose filters are invoked when the object is about to call another object's method
- Distributed coordinated filters
  - Aspect => Service
  - Object => Node
  - Each Service is "sliced" over the nodes



Confidential

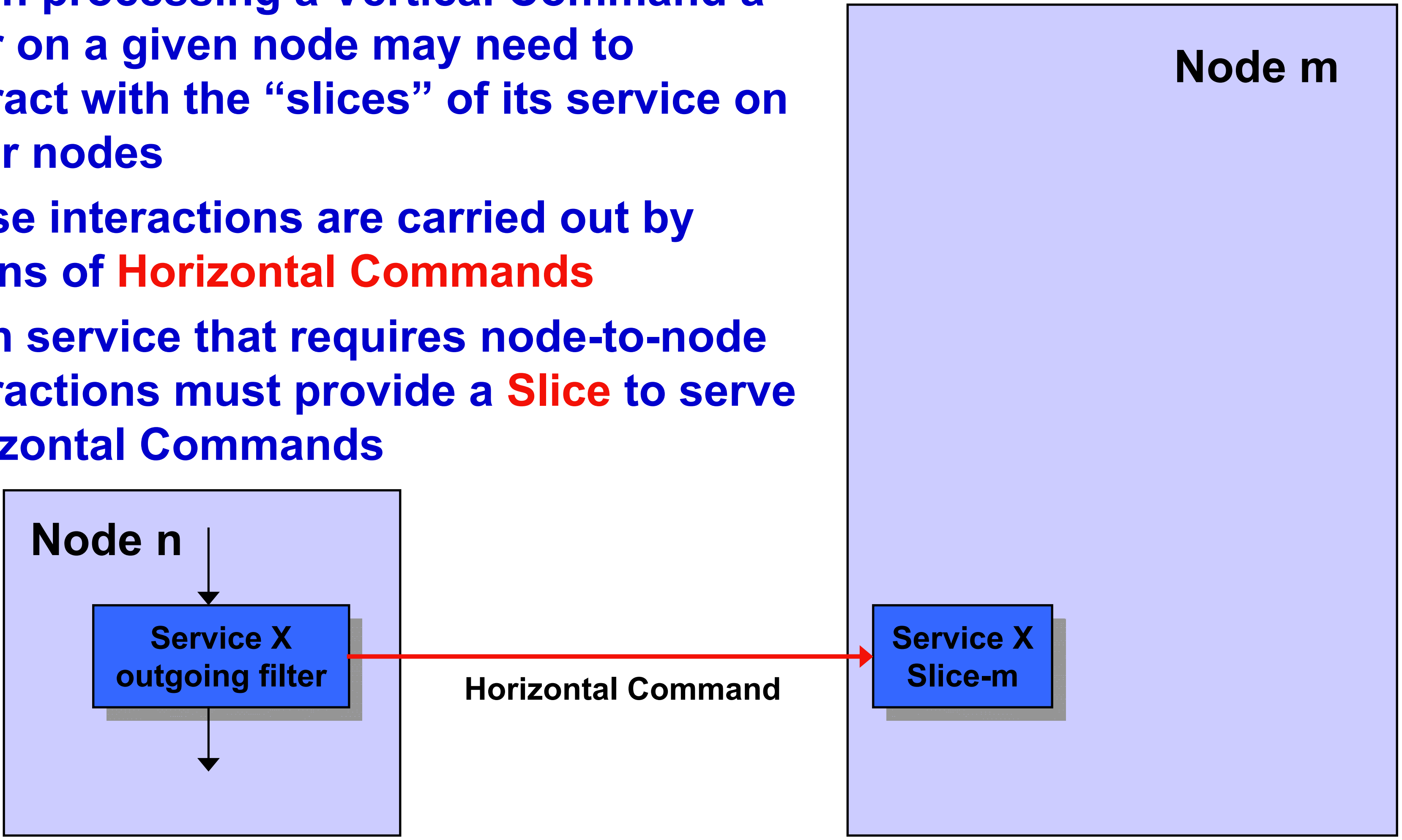
The Outgoing filter chain

- All events that happen at the agent level trigger a Vertical Commands
- Each Service may provide an Outgoing Filter and all Vertical Commands are processed sequentially by the filters of all services installed in the local node.
- Each filter can act on certain Vertical commands and ignore the others



Horizontal Commands and Slices

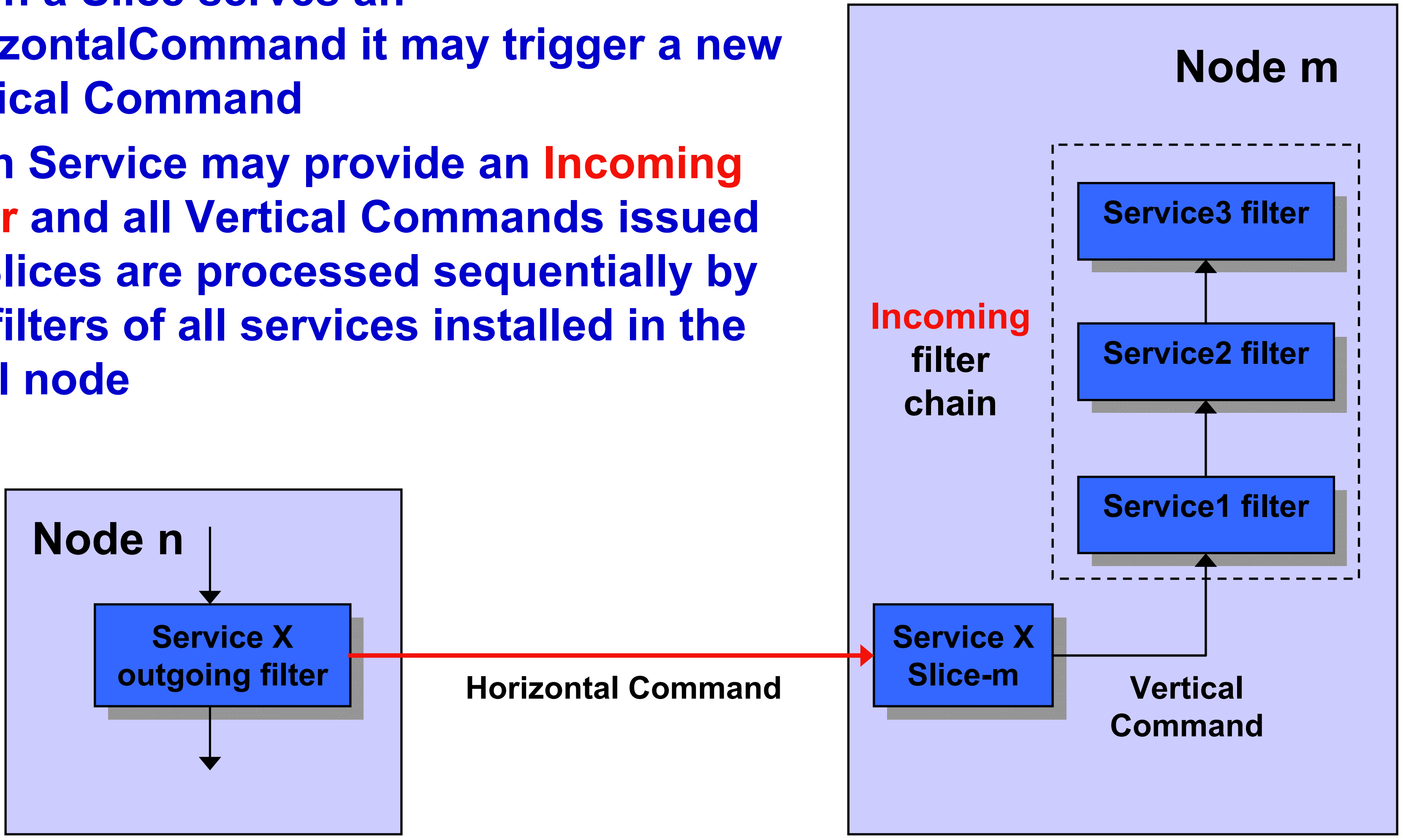
- When processing a Vertical Command a filter on a given node may need to interact with the “slices” of its service on other nodes
- These interactions are carried out by means of Horizontal Commands
- Each service that requires node-to-node interactions must provide a Slice to serve Horizontal Commands



Confidential

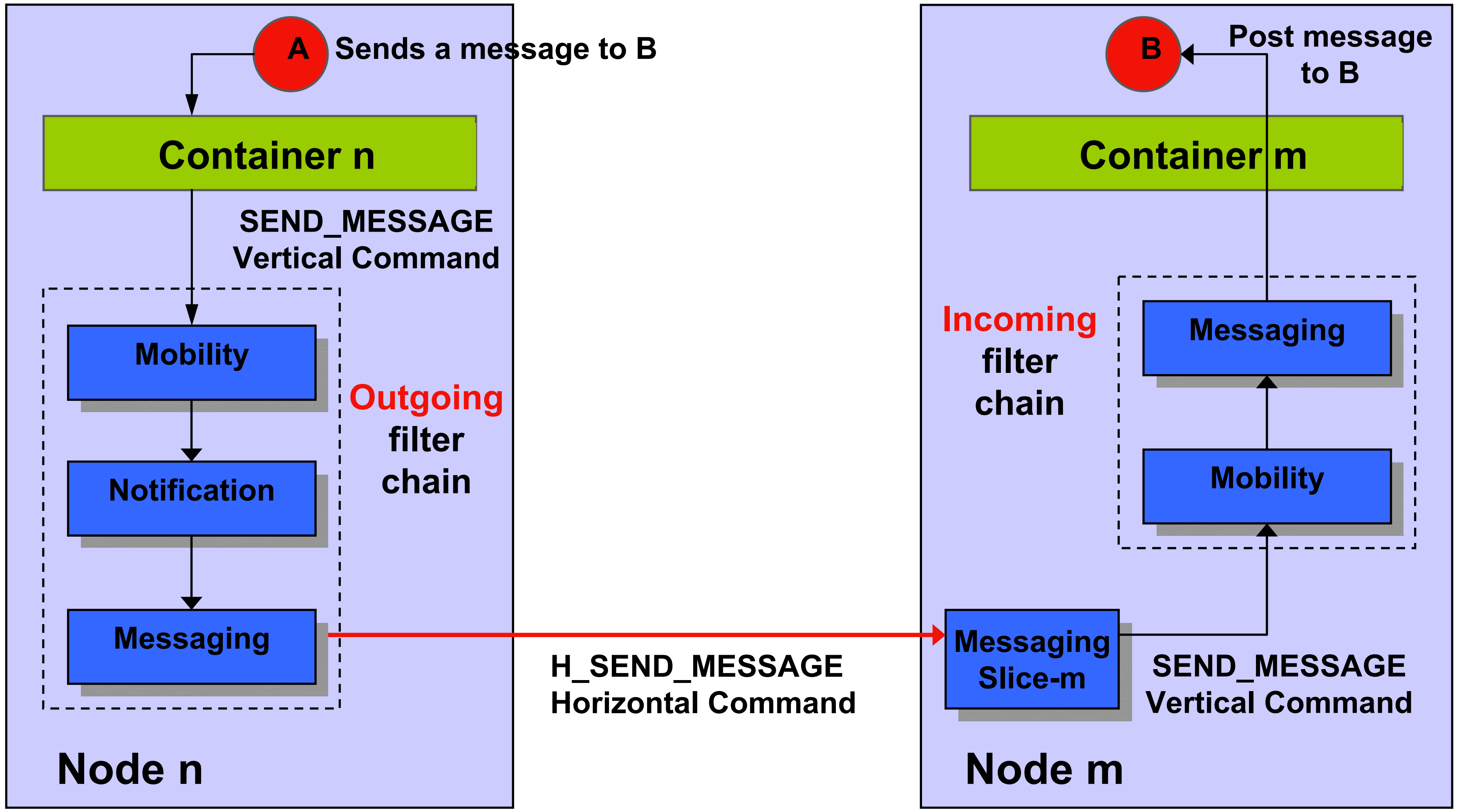
The Incoming filter chain

- When a Slice serves an HorizontalCommand it may trigger a new Vertical Command
- Each Service may provide an Incoming Filter and all Vertical Commands issued by Slices are processed sequentially by the filters of all services installed in the local node



Confidential

An example



Confidential

How an agent can interact with a service

- Some services may be directly accessed by agents through a ServiceHelper
- Service helpers can be retrieved by means of the `getHelper()` method of the `Agent` class
- E.g.
- `SecurityHelper sh = (SecurityHelper) getHelper(“security”);`
- For backward compatibility reasons some services are not accessed by means of a ServiceHelper, but by means of methods of the Agent class.
- E.g.
  - `Agent.send()` instead of `MessagingHelper.send()`
  - `Agent.doMove()` instead of `MobilityHelper.doMove()`

Confidential

Sample code: The SniffingService

```
public class SniffingService extends BaseService {
   private Filter myFilter = new SniffingFilter();
   private ServiceHelper myHelper = new SniffingHelperImpl();

   /**
      Retrieve the filters of this Service
    */
   public Filter getCommandFilter(boolean direction) {
               if (direction == Filter.OUTGOING) {
                              return myFilter;
               }
               else {
                              // We are only interested in sent messages → No incoming Filter
                              return null;
               }
   }

   /**
      Retrieve the helper of this Service
    */
   public ServiceHelper getHelper(Agent a) {
               return myHelper;
   }
   .....
```

Confidential

Sample code: The SniffingFilter

```
private MessageTemplate myTemplate;

/**
   Inner class SniffingFilter
   The filter that actually sniffs messages.
 */
private class SniffingFilter extends Filter {
		public boolean accept(VerticalCommand vc) {
				if (vc.getName().equals(MessagingSlice.SEND_MESSAGE)) {
						Object[] params = vc.getParams();
						AID sender = (AID) params[0];
						GenericMessage gMsg = (GenericMessage) params[1];
						ACLMessage msg = gMsg.getACLMessage();
						if (myTemplate != null && myTemplate.match(msg)) {
								System.out.println("Matching message”);
								System.out.println(msg);
						}
				}
				return true;
		}
} // END of inner class SniffingFilter
```

Confidential

Sample code: The SniffingHelper

```
  /**
     Inner class SniffingHelperImpl
     Allows agents to set templates for messages to be sniffed
   */
  private class SniffingHelperImpl implements SniffingHelper {
          public void init(Agent a) {
          }

          public void setTemplate(MessageTemplate tpl) {
                  myTemplate = tpl;
          }
  } // END of inner class SniffingHelperImpl

......

public interface SniffingHelper extends ServiceHelper {
  public void setTemplate(MessageTemplate tpl);
}
```

Confidential

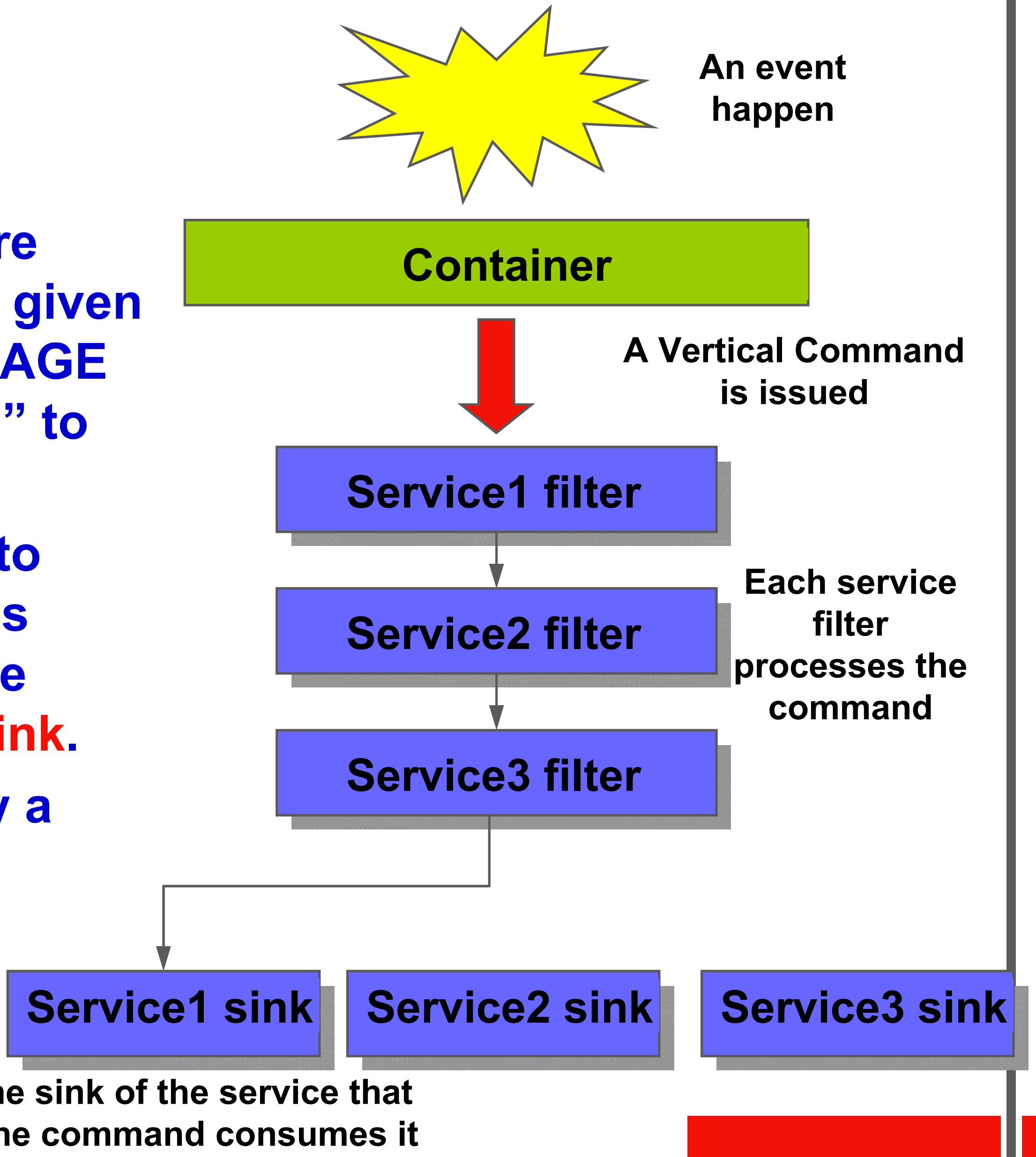
Sample code: Accessing the helper

```
.....
try {
   SniffingHelper myHelper = (SniffingHelper) getHelper(SniffingService.NAME);
   myHelper.setTemplate(MessageTemplate.MatchPerformative(ACLMessage.REQUEST
   ));
}
catch (ServiceException se) {
   se.printStackTrace();
}
.....
```

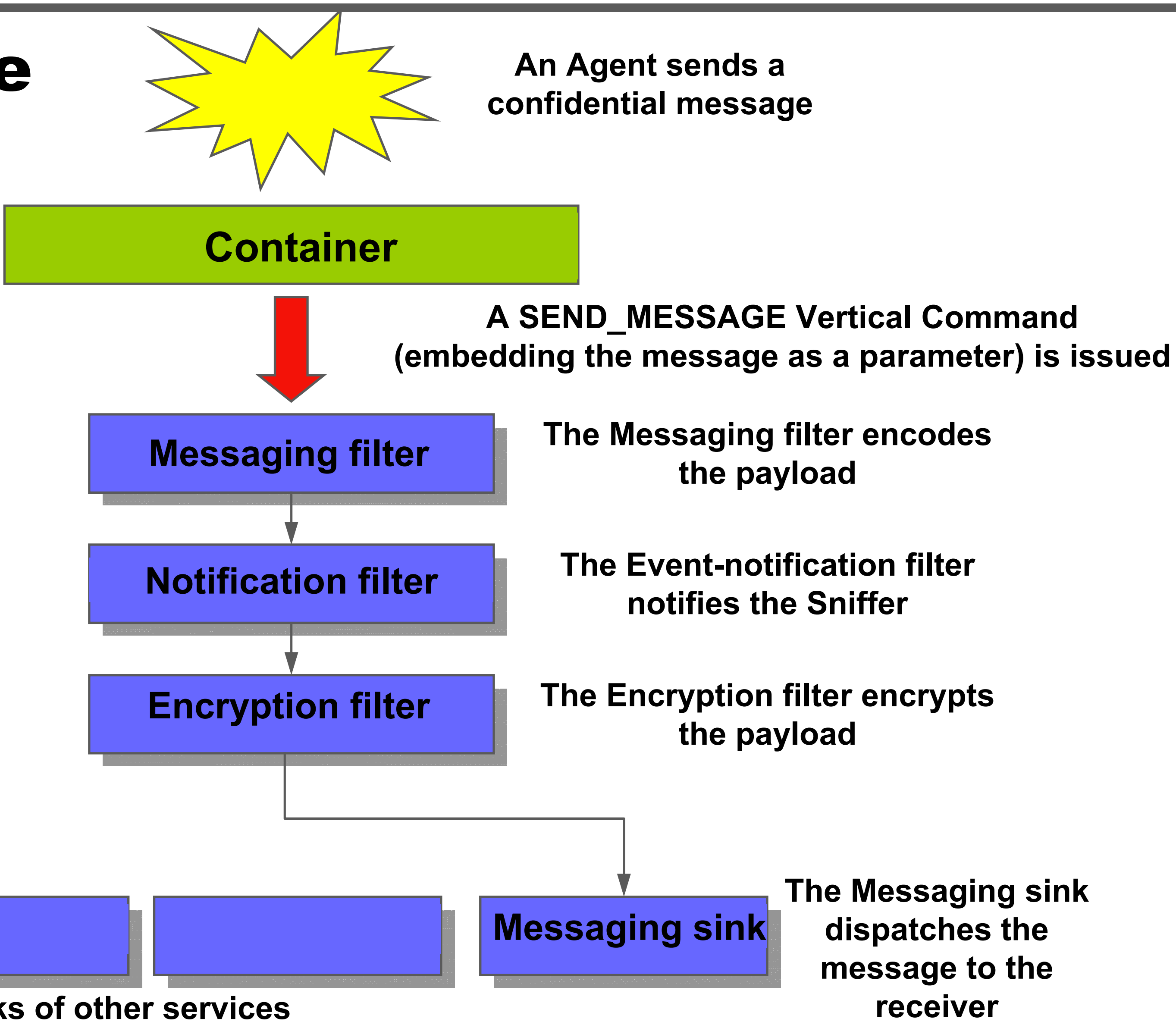
Confidential

Sinks

- Some vertical commands are intrinsically associated to a given service (e.g. a SEND_MESSAGE vertical command "belongs" to the Messaging Service)
- Each service has a chance to consume its own commands after they have traversed the filter chain by means of a Sink.
- Each service may have only a filter, only a sink or both

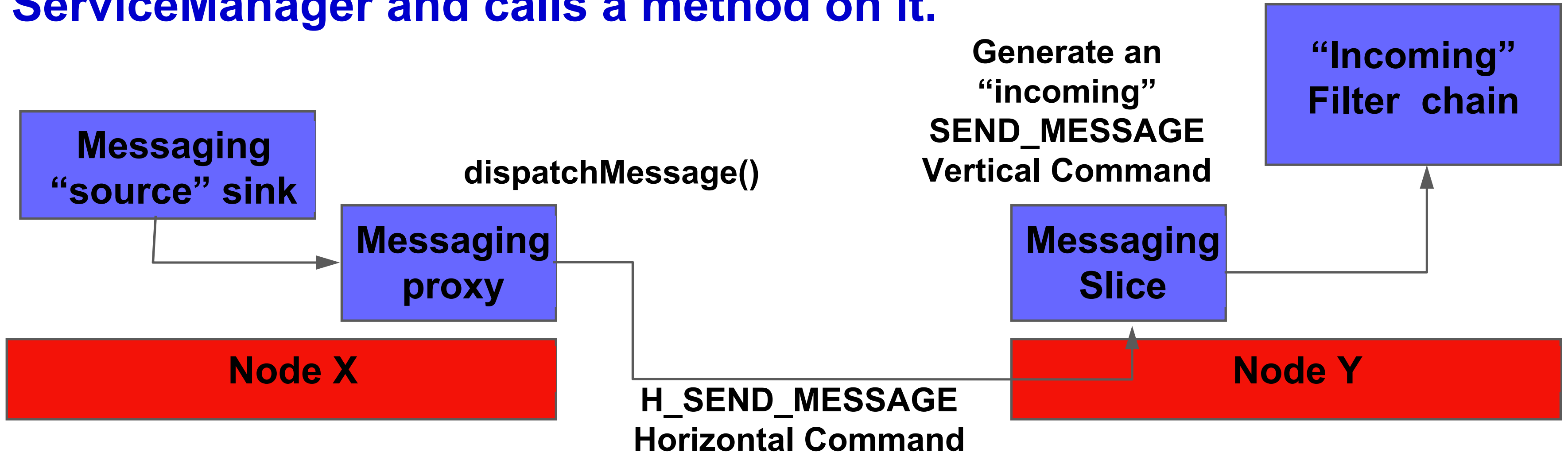

An event happen
Container
A Vertical Command is issued
Service1 filter
Service2 filter
Service3 filter
Each service filter processes the command
Service1 sink
Service2 sink
Service3 sink
Only the sink of the service that owns the command consumes it

An example


An Agent sends a confidential message
Container
A SEND_MESSAGE Vertical Command (embedding the message as a parameter) is issued
Messaging filter
The Messaging filter encodes the payload
Notification filter
The Event-notification filter notifies the Sniffer
Encryption filter
The Encryption filter encrypts the payload
Messaging sink
The Messaging sink dispatches the message to the receiver
Sinks of other services

SliceProxies

- Are in charge to transform method calls to remote slices into Horizontal Commands
- Implement the Horizontal Interface of a service (method `getHorizontalInterface()`)
- The name must be <Service-name>Proxy
- When a service filter or sink needs to interact with a slice on a remote node it retrieves a proxy to that slice through the ServiceManager and calls a method on it.



Confidential

Summary

- A Service is composed of
  - Outgoing & incoming filters (for processing Vertical Commands)
  - Source & target sinks (for consuming owned Vertical Commands)
  - Slice and SliceProxy (for node-to-node interactions)
  - Helper (for agent interactions)
- Mostly all JADE features are currently implemented as services
- People interested in modifying/extending JADE features should consider the development of new services as the first option

Confidential

Summary

- JADE
- The JADE Board
- The new Kernel
  - Ideas and motivations
  - Main elements
  - An example
  - Advanced
- The security add-on
- Roadmap

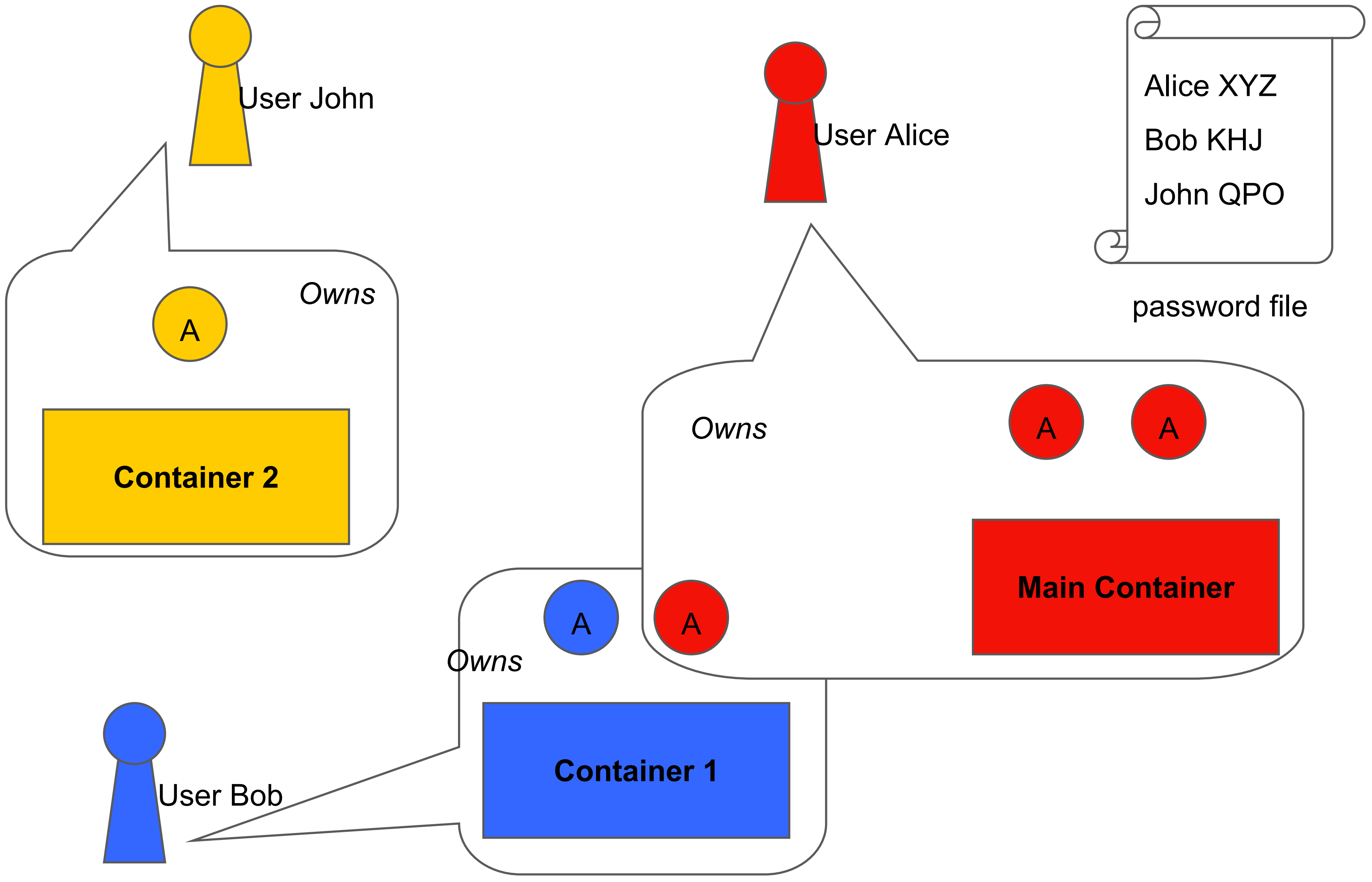
Confidential

The security add-on

- Completely replace JADE-S 1 that included some architectural limitations
- Provides support for:
  - Multi-user environment (ownership, permissions...)
  - End-to-end message integrity and confidentiality
- Fully integrated in the new services architecture
  - Security Service
  - Permission Service
  - Signature Service
  - Encryption Service
- A single Helper (SecurityHelper) provides access to all services

Confidential

JADE as a multi-user environment



Confidential

Policy file

- grant principal jade.security.Name "alice" {
  permission jade.security.PlatformPermission "", "create";
  permission jade.security.ContainerPermission "", "create";
  permission jade.security.AMSPermission "agent-class=*", "register, derister,modify";
  permission jade.security.AgentPermission "agent-class=*", "create, kill";
  permission jade.security.MessagePermission "agent-owner:alice", "send-to";

};

Confidential

Signing messages

```
// Create the message
ACLMessage msg = new ACLMessage(ACLMessage.INFORM);
......

// Retrieve the SecurityHelper
SecurityHelper myHelper = (SecurityHelper)
  getHelper("jade.core.security.Security");

// The message must be signed
mySecurityHelper.setUseSignature(msg);

// Send the message
send(msg)';
```

Confidential

Roadmap 2003-2004

JADE 3.0b1
- consolidation of previous features
- integrated JADE and LEAP
- split container mode
- update for compliance to new FIPA standards
- persistent DF
- XMLCodec
- misc add-on

19/3/03
3.0b1

JADE 3.1
- consolidation of previous features
- tutorial for beginners + more documentation
- new service-based kernel based on a Distributed Composition Filter pattern
- support for replication of the main container
- support for application specific message persistency through the PersistentDeliveryService
- support for FIPA Propose, 2phase-commit interaction protocol
- HTTP, JMS MTPs
- Bean-generator, and RDFCodec add-ons

18/12/03
3.1

JADE 3.2
- consolidation of previous features
- security add-on including support for authentication, authorization, and end-to-end message integrity and confidentiality (J2SE only)
- HTTP MTP
- Test Suite Framework
- Persistence Service
- threaded behaviours
- agent loading from jar file
- logging
- SerializableOntology that handles homogeneously ontological concepts and serializable objects
- Automatic DF clean-up when a registered agent dies

15/7/04
3.2

JADE 3.3
- consolidation of previous features
- security add-on (J2ME)
- Web Service Integration
- Agent Launcher (Univ. Aachen)
- Performance Scalability

15/12/04
3.3

Confidential

Thanks for your attention!

Questions?

Confidential

Currently available services

- AgentManagement
- Messaging
- Mobility
- EventNotification
- PersistentDelivery
- MainReplication

Included in the JADE standard distribution

- Persistence (under test)
- Security (Not yet completed)
  - Permission
  - Signature
  - Encryption

Distributed as separate add-ons

Confidential