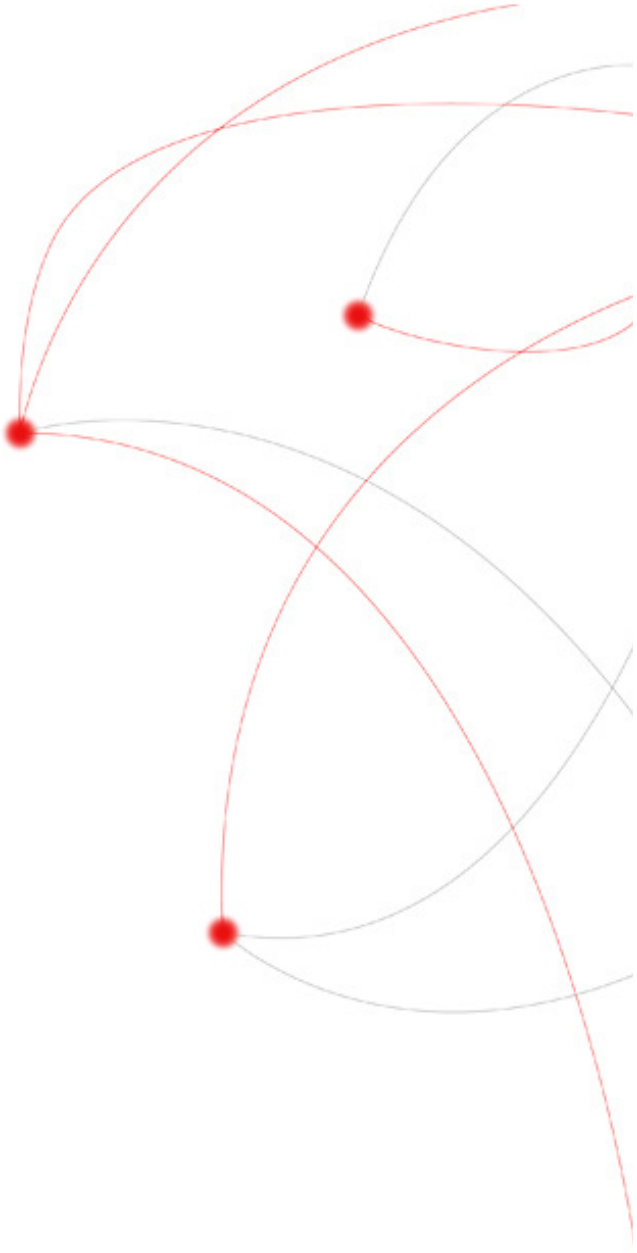
TELECOM ITALIA GROUP

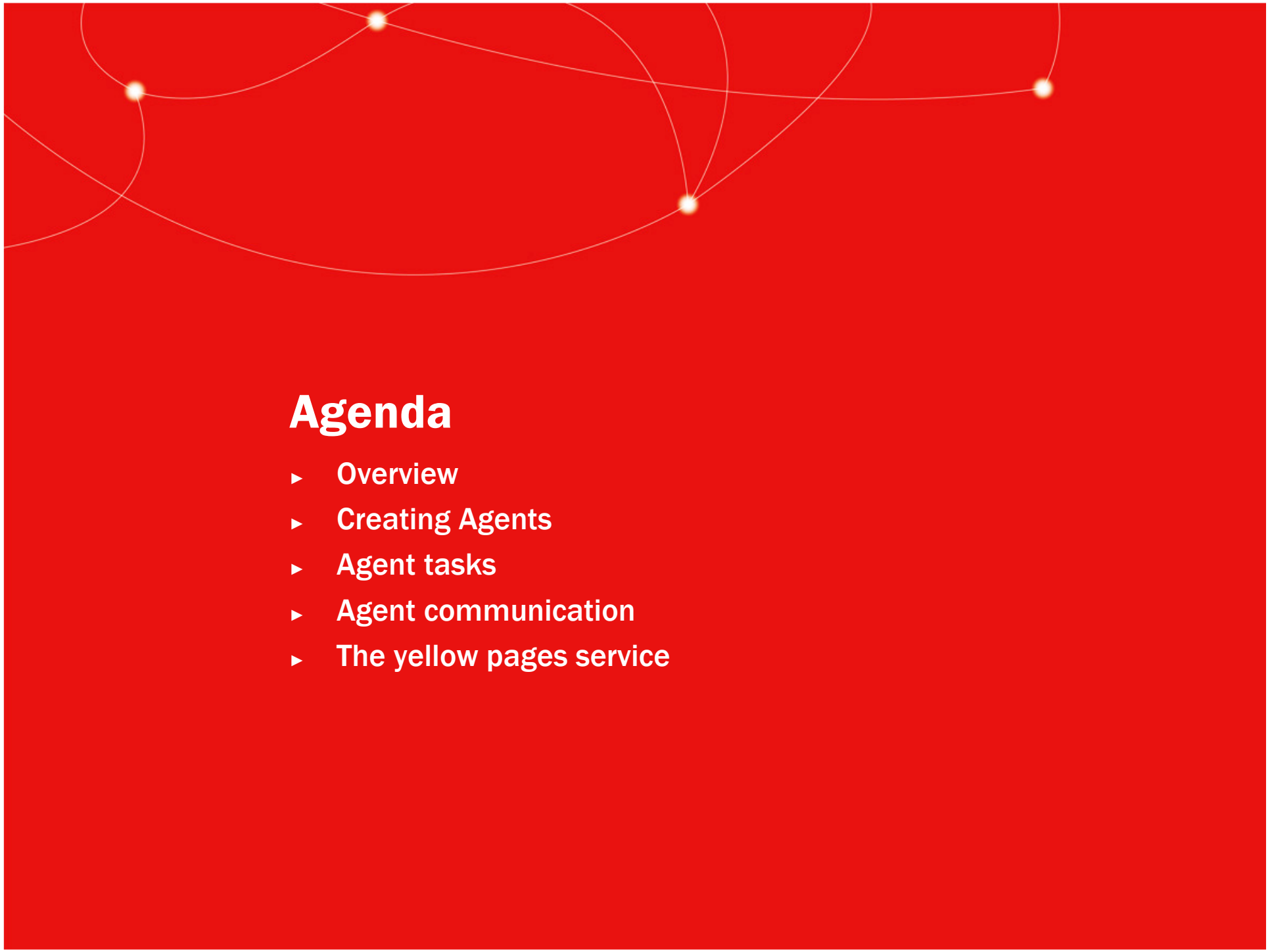


JADE

14 September 2010

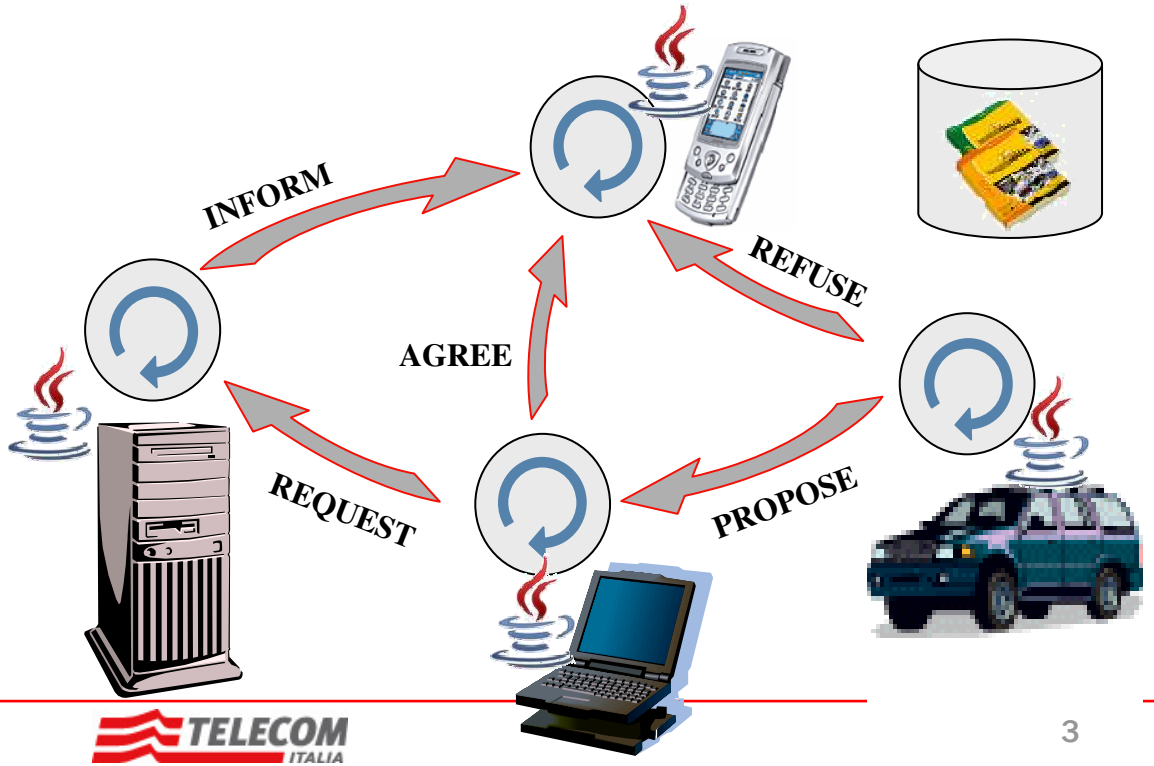
Giovanni Caire
Giovanni.caire@telecomitalia.it

TELECOM ITALIA

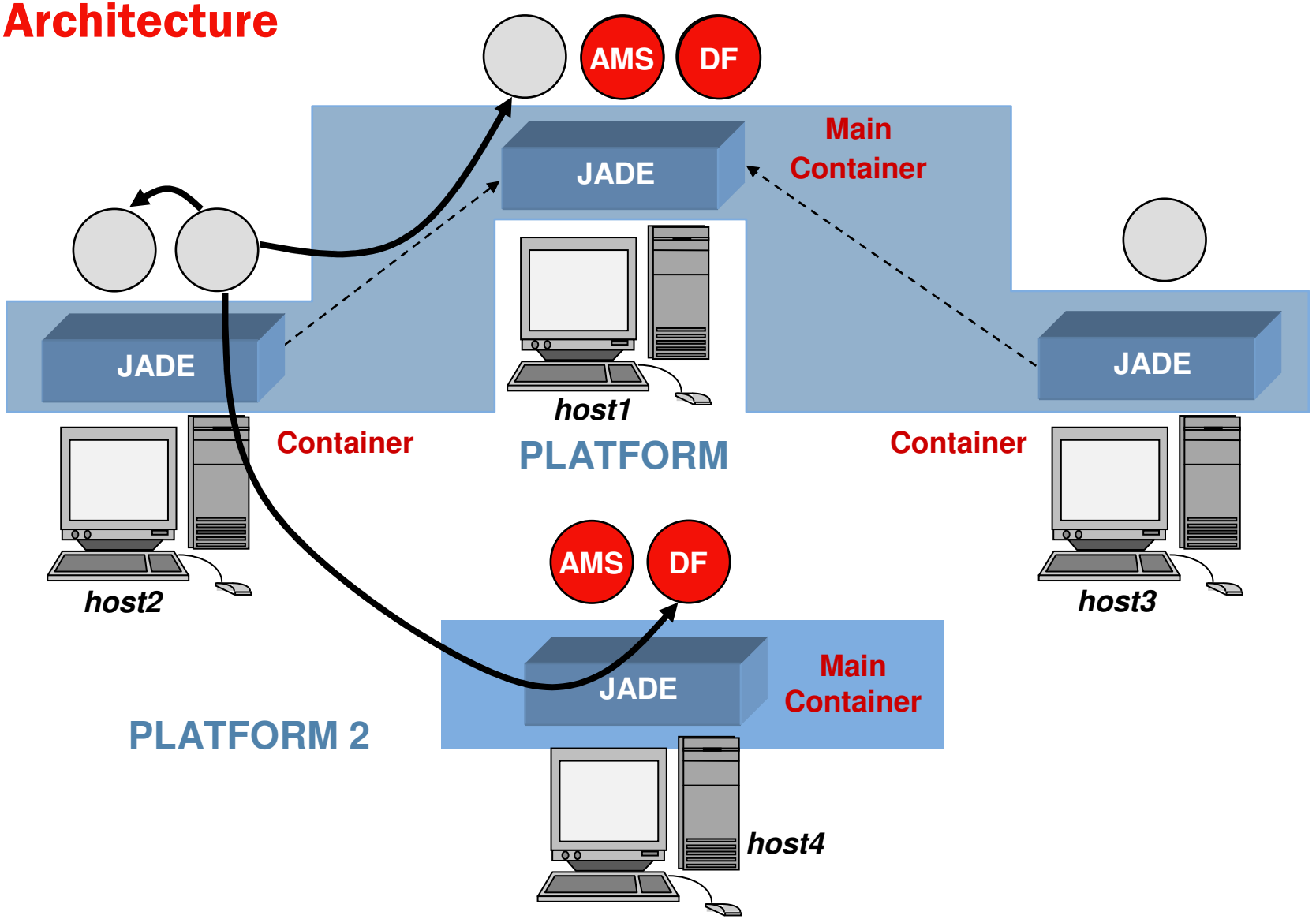

Agenda
Overview
Creating Agents
Agent tasks
Agent communication
The yellow pages service

JADE

- A middleware for applications based on the agent paradigm
- Provides
  - The Agent and Behaviour (a task that an agent can execute) abstractions
  - Transparent distribution of components (agents)
  - Peer-to-peer communication based on asynchronous message passing
  - Publish-subscribe discovery mechanisms
- Fully written in Java
- Open Source
  - http://jade.tilab.com
  - ~230.000 downloads
  - Current version: 4.01

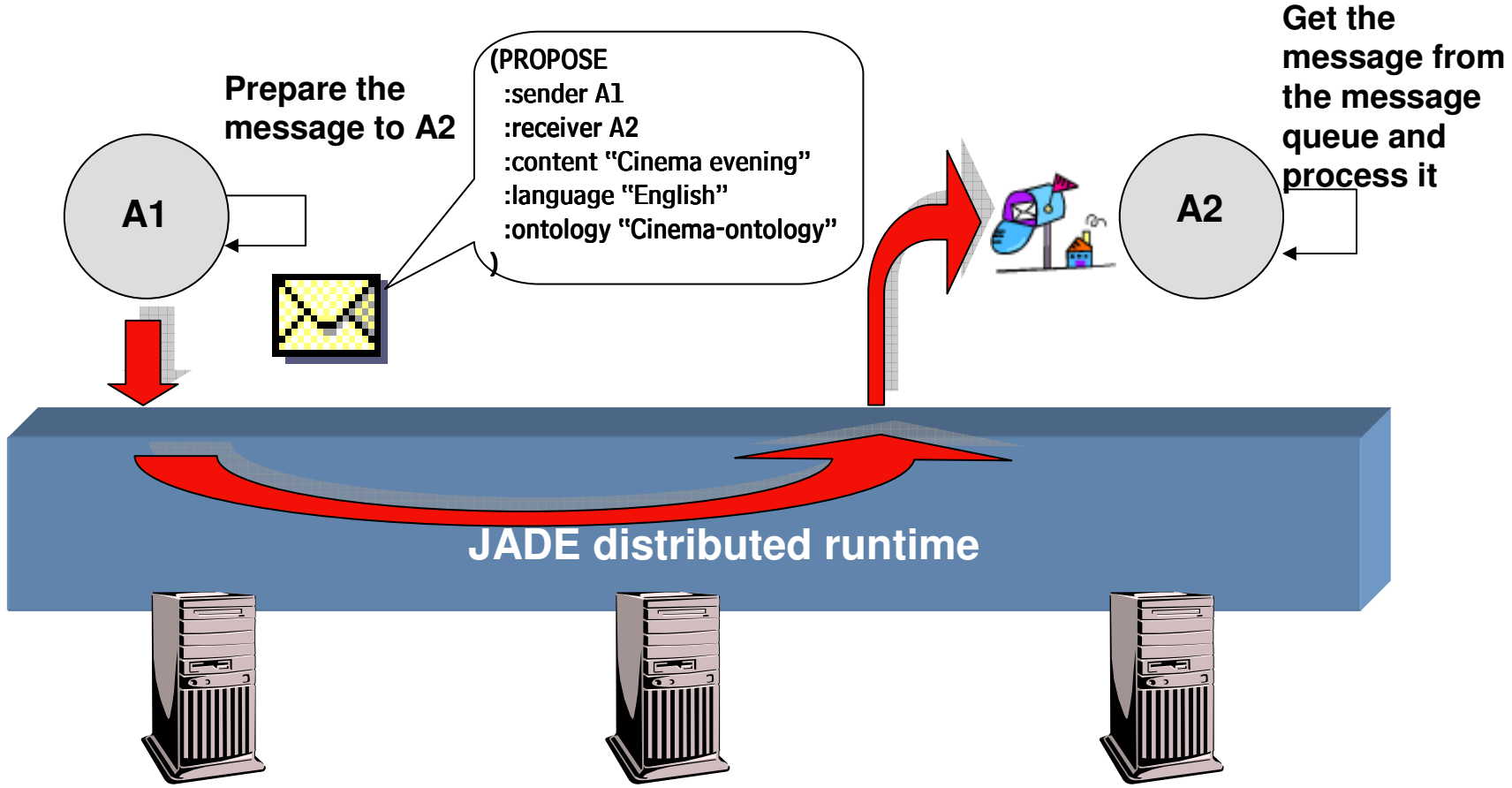


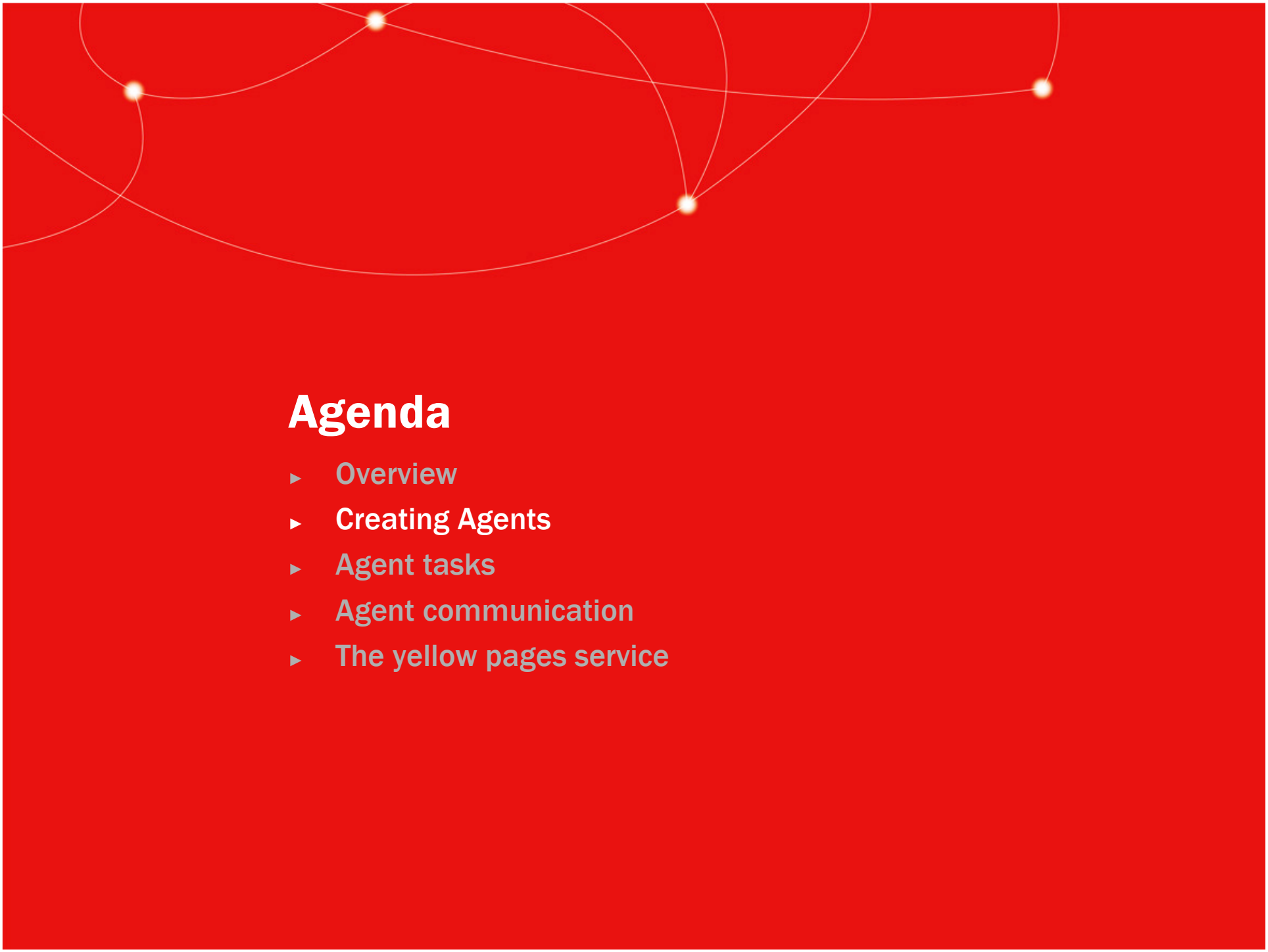
Architecture



The communication model

- Based on asynchronous message passing
- Message format defined by the ACL language (FIPA)





Agenda

- Overview
- Creating Agents
- Agent tasks
- Agent communication
- The yellow pages service

The HelloWorld agent

- A type of agent is created by extending the `jade.core.Agent` class and redefining the `setup()` method.
- Each Agent instance is identified by an AID (`jade.core.AID`).
  - An AID is composed of a unique name plus some addresses
  - An agent can retrieve its AID through the `getAID()` method of the `Agent` class

```
import jade.core.Agent;

public class HelloWorldAgent extends Agent {

  protected void setup() {
    System.out.println("Hello World! my name is "+getAID().getName());
  }
}
```

Local names, GUID and addresses

- Agent names have the form <local-name>@<platform-name>
- The complete name of an agent must be globally unique.
- The default platform name is <main-host>:<main-port>/JADE
- The platform name can be set using the `-name` option
- Within a single JADE platform agents are referred through their names only.
- Given the name of an agent its AID can be created as
  - `AID id = new AID(localname, AID.ISLOCALNAME);`
  - `AID id = new AID(name, AID.ISGUID);`
- The addresses included in an AID are those of the platform (MTPs) and are ONLY used in communication between agents living on different FIPA platforms

Main startup options summary

- Switch options
  - -gui (activate the management GUI)
  - -container (launch a peripheral container instead of a main container)
- Key-value pair options
  - -host <host> (the host where the Main Container is running)
  - -port <port> (the port where the Main Container is running)
  - -detect-main <true|false> ((find the Main Container automatically)
  - -local-port <port> (the port to be used by the starting container)
  - -agents <agent specifier list> (the agents to be started at bootstrap)
  - -conf <properties file> (get startup options from a property file)
- Command line examples
  - java –cp … jade.Boot –gui –agents john:hello.HelloWorldAgent
  - java –cp … jade.Boot –container –host avalon.telecomitalia.it

Passing arguments to an agent

- It is possible to pass arguments to an agent
  - `java jade.Boot .... a:myPackage.MyAgent(arg1,arg2)`
  - The agent can retrieve its arguments through the `getArguments()` method of the `Agent` class

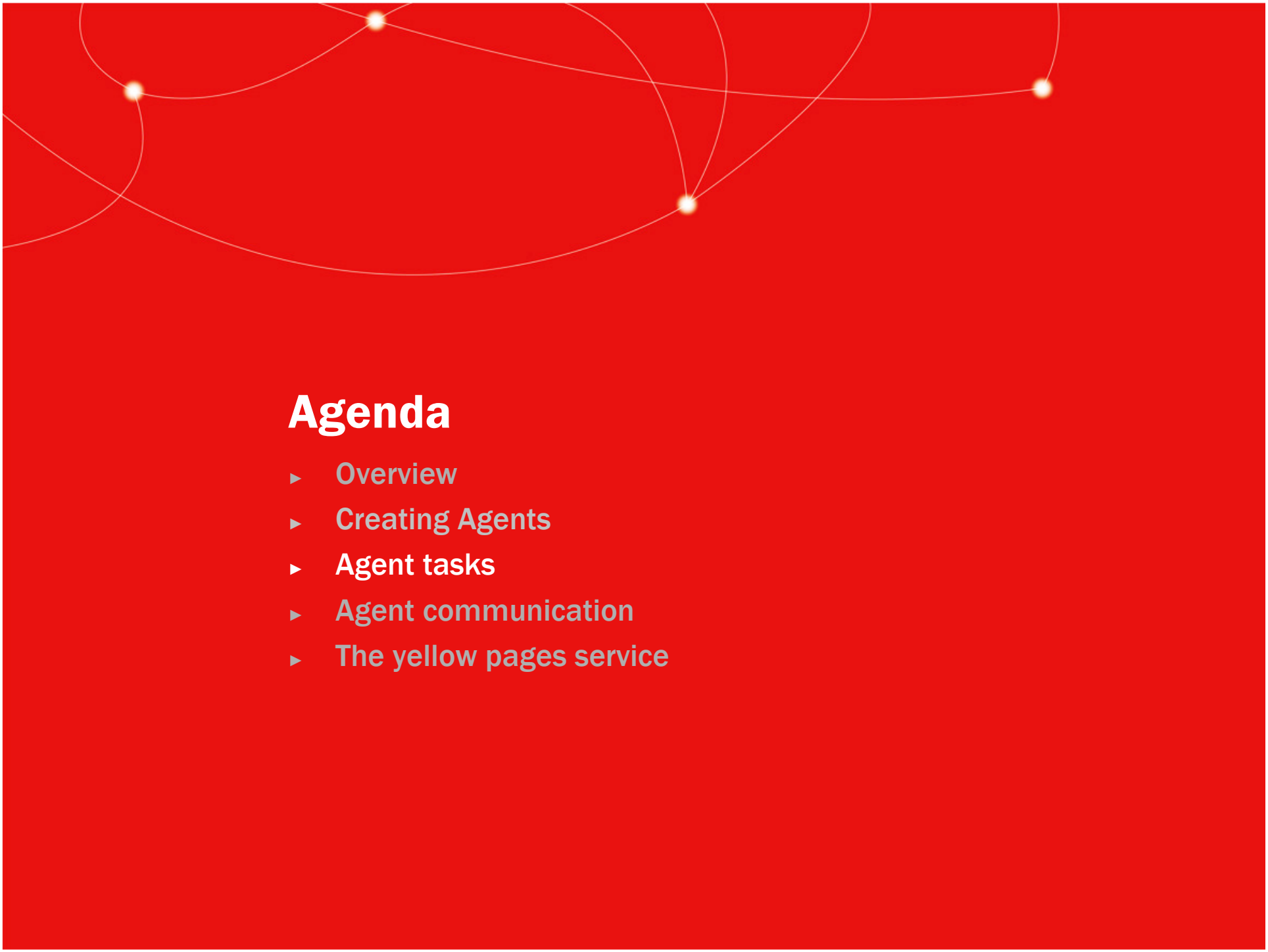
```
protected void setup() {
  System.out.println("Hallo World! my name is "+getAID().getName());
  Object[] args = getArguments();
  if (args != null) {
    System.out.println("My arguments are:");
    for (int i = 0; i < args.length; ++i) {
      System.out.println("- "+args[i]);
    }
  }
}
```

Agent termination

- An agent terminates when its `doDelete()` method is called.
- On termination the agent's `takeDown()` method is invoked (intended to include clean-up operations).

```
protected void setup() {
  System.out.println("Hallo World! my name is "+getAID().getName());
  Object[] args = getArguments();
  if (args != null) {
    System.out.println("My arguments are:");
    for (int i = 0; i < args.length; ++i) {
      System.out.println("- "+args[i]);
    }
  }
  doDelete();
}

protected void takeDown() {
  System.out.println("Bye...");
}
```



Agenda

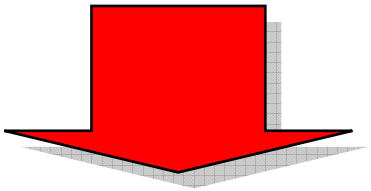
- Overview
- Creating Agents
- Agent tasks
- Agent communication
- The yellow pages service

The Behaviour class

- The actual job that an agent does is typically carried out within "behaviours"
- Behaviours are created by extending the `jade.core.behaviours.Behaviour` class
- To make an agent execute a task it is sufficient to create an instance of the corresponding `Behaviour` subclass and call the `addBehaviour()` method of the `Agent` class.
- Each Behaviour subclass must implement
  - `public void action()`: what the behaviour actually does
  - `public boolean done()`: whether the behaviour is finished

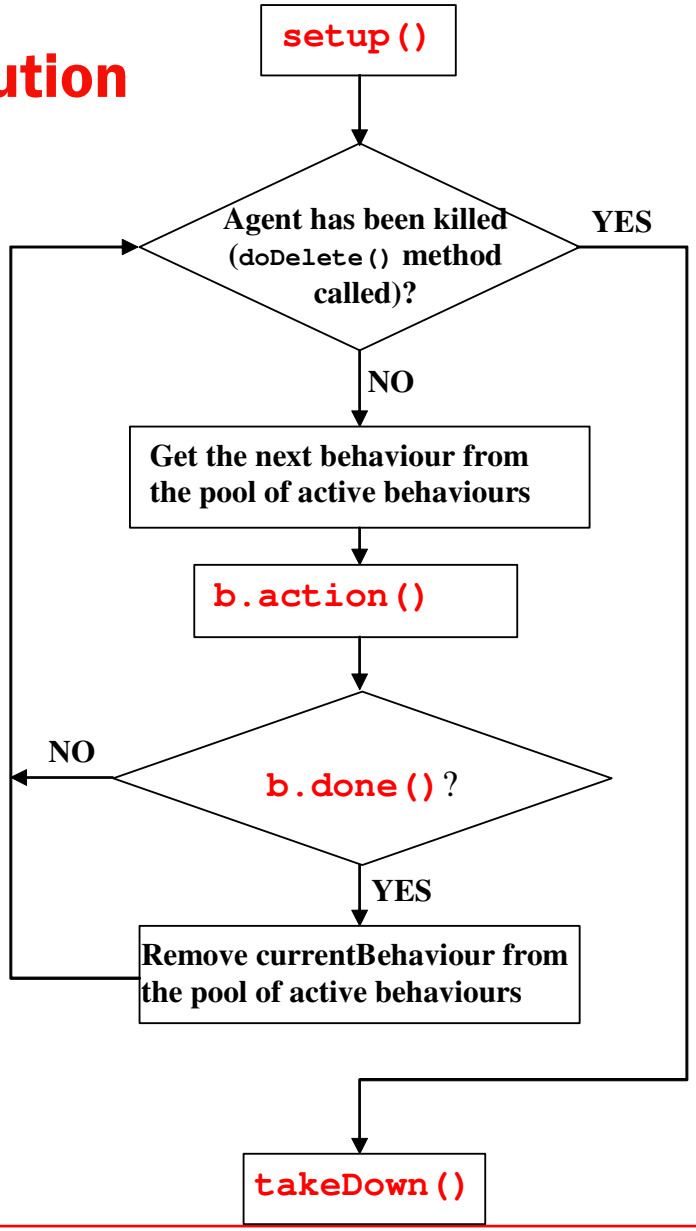
Behaviour scheduling and execution

- An agent can execute several behaviours in parallel, however, behaviour scheduling is not preemptive, but cooperative and everything occurs within a single Java Thread



Behaviour switch occurs only when the `action()` method of the currently scheduled behaviour returns.

The agent execution model



- Initializations
- Addition of initial behaviours

- Agent "life" (execution of behaviours)

- Clean-up operations

Highlighted in red the methods that programmers have to/can implement

Behaviour “types”

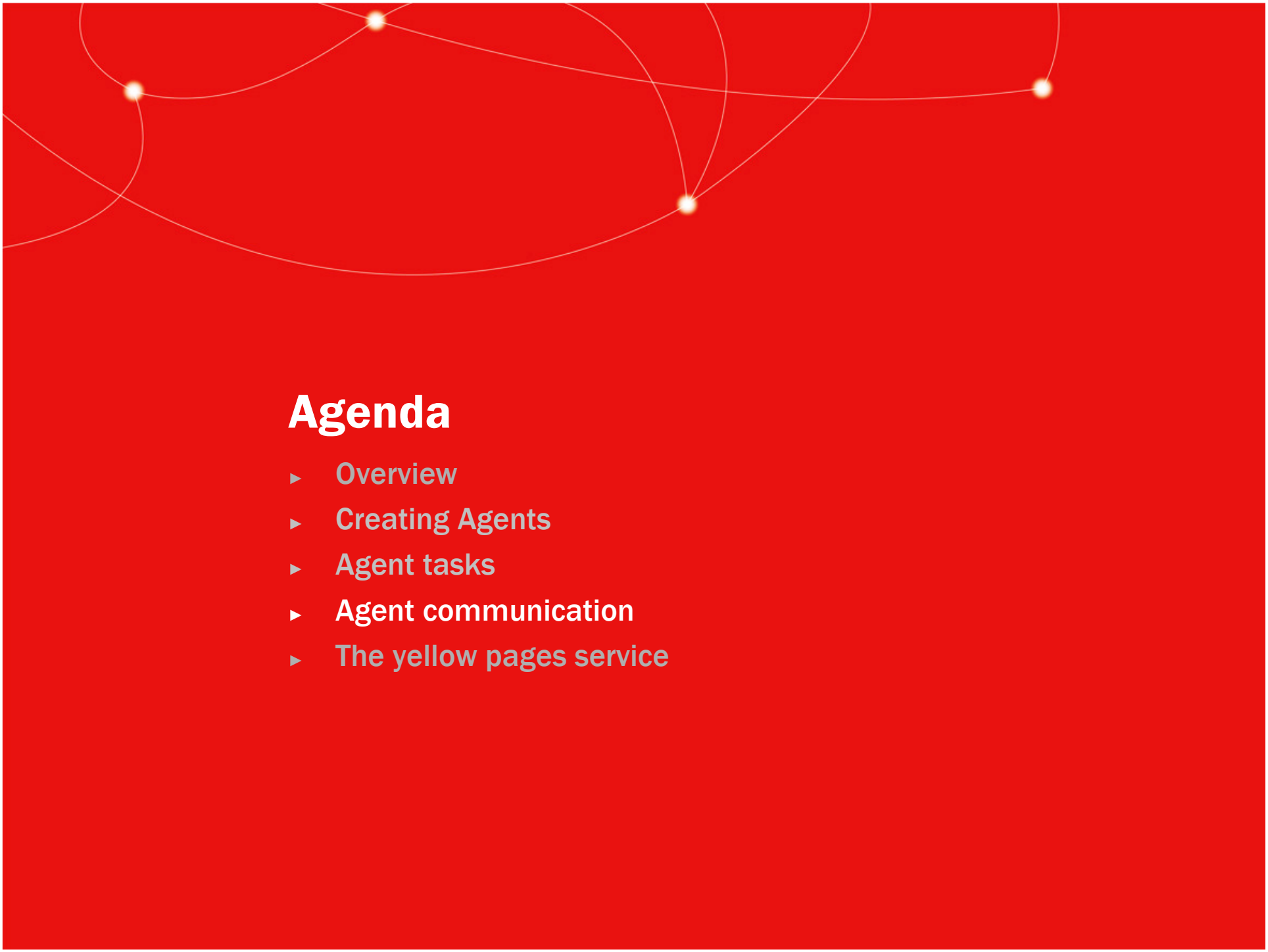
- “One shot” behaviours.
  - Complete immediately and their `action()` method is executed only once.
  - Their `done()` method simply returns `true`.
  - `jade.core.behaviours.OneShotBehaviour` class
- “Cyclic” behaviours.
  - Never complete and their `action()` method executes the same operation each time it is invoked
  - Their `done()` method simply returns `false`.
  - `jade.core.behaviours.CyclicBehaviour` class
- “Complex” behaviours.
  - Embed a state and execute in their `action()` method different operations depending on their state.
  - Complete when a given condition is met.

Scheduling operations at given points in time

- JADE provides two ready-made classes by means of which it is possible to easily implement behaviours that execute operations at given points in time
- **`WakerBehaviour`**
  - The `action()` and `done()` method are already implemented so that the `onWake()` method (to be implemented by subclasses) is executed after a given timeout
  - After that execution the behaviour completes.
- **`TickerBehaviour`**
  - The `action()` and `done()` method are already implemented so that the `onTick()` (to be implemented by subclasses) method is executed periodically with a given period
  - The behaviour runs forever unless its `stop()` method is called.

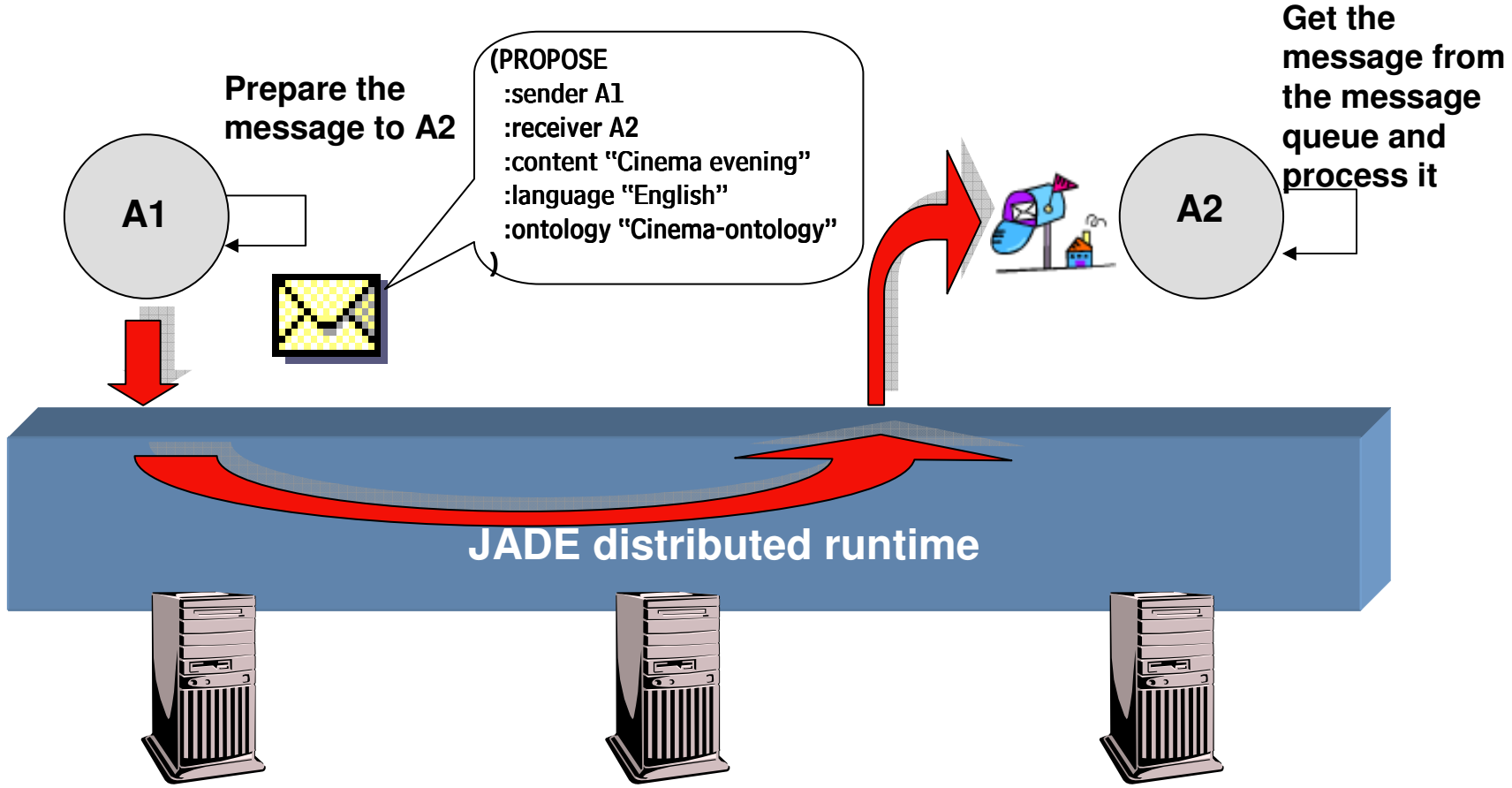
More about behaviours

- The `onStart()` method of the `Behaviour` class is invoked only once before the first execution of the `action()` method. Suited for operations that must occur at the beginning of the behaviour
- The `onEnd()` method of the `Behaviour` class is invoked only once after the `done()` method returns `true`. Suited for operations that must occur at the end of the behaviour
- Each behaviour has a pointer to the agent executing it: the protected member variable `myAgent`
- The `removeBehaviour()` method of the `Agent` class can be used to remove a behaviour from the agent pool of behaviours. The `onEnd()` method is not called.
- When the pool of active behaviours of an agent is empty the agent enters the IDLE state and its thread goes to sleep


Agenda
Overview
Creating Agents
Agent tasks
Agent communication
The yellow pages service

The communication model

- Based on asynchronous message passing
- Message format defined by the ACL language (FIPA)



The ACLMessage class

- Messages exchanged by agents are instances of the `jade.lang.acl.ACLMessage` class.
- Provide accessor methods to get and set all the fields defined by the ACL language
  - get/setPerformative();
  - get/setSender();
  - add/getAllReceiver();
  - get/setLanguage();
  - get/setOntology();
  - get/setContent();
  - ....

Sending and receiving messages

- Sending a message is as simple as creating an `ACLMessage` object and calling the `send()` method of the `Agent` class

```
ACLMessage msg = new ACLMessage(ACLMessage.INFORM);
msg.addReceiver(new AID("Peter", AID.ISLOCALNAME));
msg.setLanguage("English");
msg.setOntology("Weather-Forecast-Ontology");
msg.setContent("Today it's raining");
send(msg);
```

- Reading messages from the private message queue is accomplished through the `receive()` method of the `Agent` class.

```
ACLMessage msg = receive();
if (msg != null) {
  // Process the message
}
```

Blocking a behaviour waiting for a message

- A behaviour that processes incoming messages does not know exactly when a message will arrive
- The `block()` method of the `Behaviour` class removes a behaviour from the agent pool and puts it in a blocked state (not a blocking call!!).
- Each time a message is received all blocked behaviours are inserted back in the pool and have a chance to read and process the message.

```
public void action() {
  ACLMessage msg = myAgent.receive();
  if (msg != null) {
    // Process the message
  }
  else {
    block();
  }
}
```

This is the strongly recommended pattern to receive messages within a behaviour

Selective reading from the message queue

- The `receive()` method returns the first message in the message queue and removes it.
- If there are two (or more) behaviours receiving messages, one may "steal" a message that the other one was interested in.
- To avoid this it is possible to read only messages with certain characteristics (e.g. whose sender is agent "Peter") specifying a `jade.lang.acl.MessageTemplate` parameter in the `receive()` method.

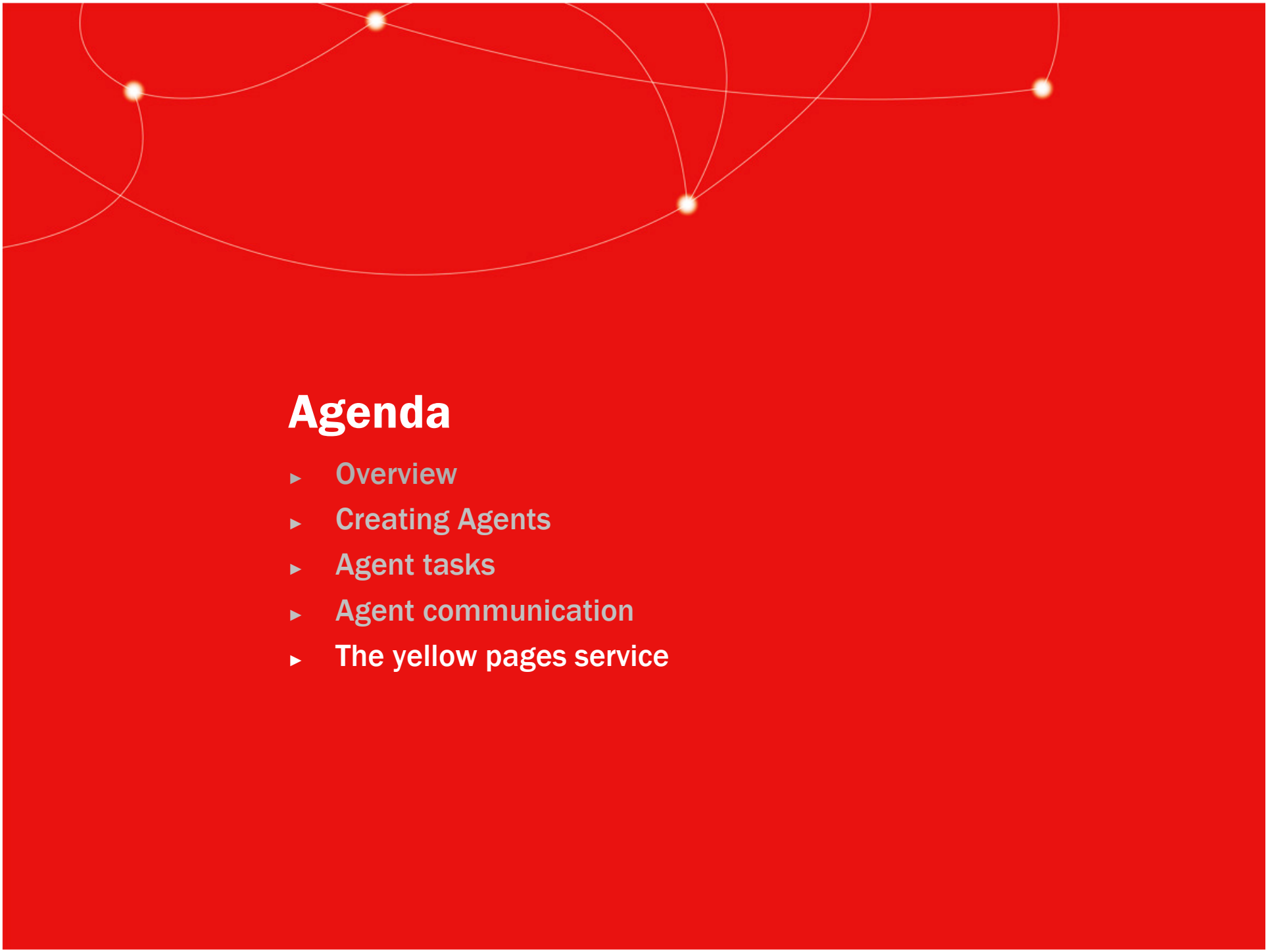
```
MessageTemplate tpl = MessageTemplate.MatchOntology("Test-Ontology");

public void action() {
  ACLMessage msg = myAgent.receive(tpl);
  if (msg != null) {
    // Process the message
  }
  else {
    block();
  }
}
```

Receiving messages in blocking mode

- The `Agent` class also provides the `blockingReceive()` method that returns only when there is a message in the message queue.
- There are overloaded versions that accept a `MessageTemplate` (the method returns only when there is a message matching the template) and or a timeout (if it expires the method returns null).
- Since it is a blocking call it is "dangerous" to use `blockingReceive()` within a behaviour. In fact no other behaviour can run until `blockingReceive()` returns.

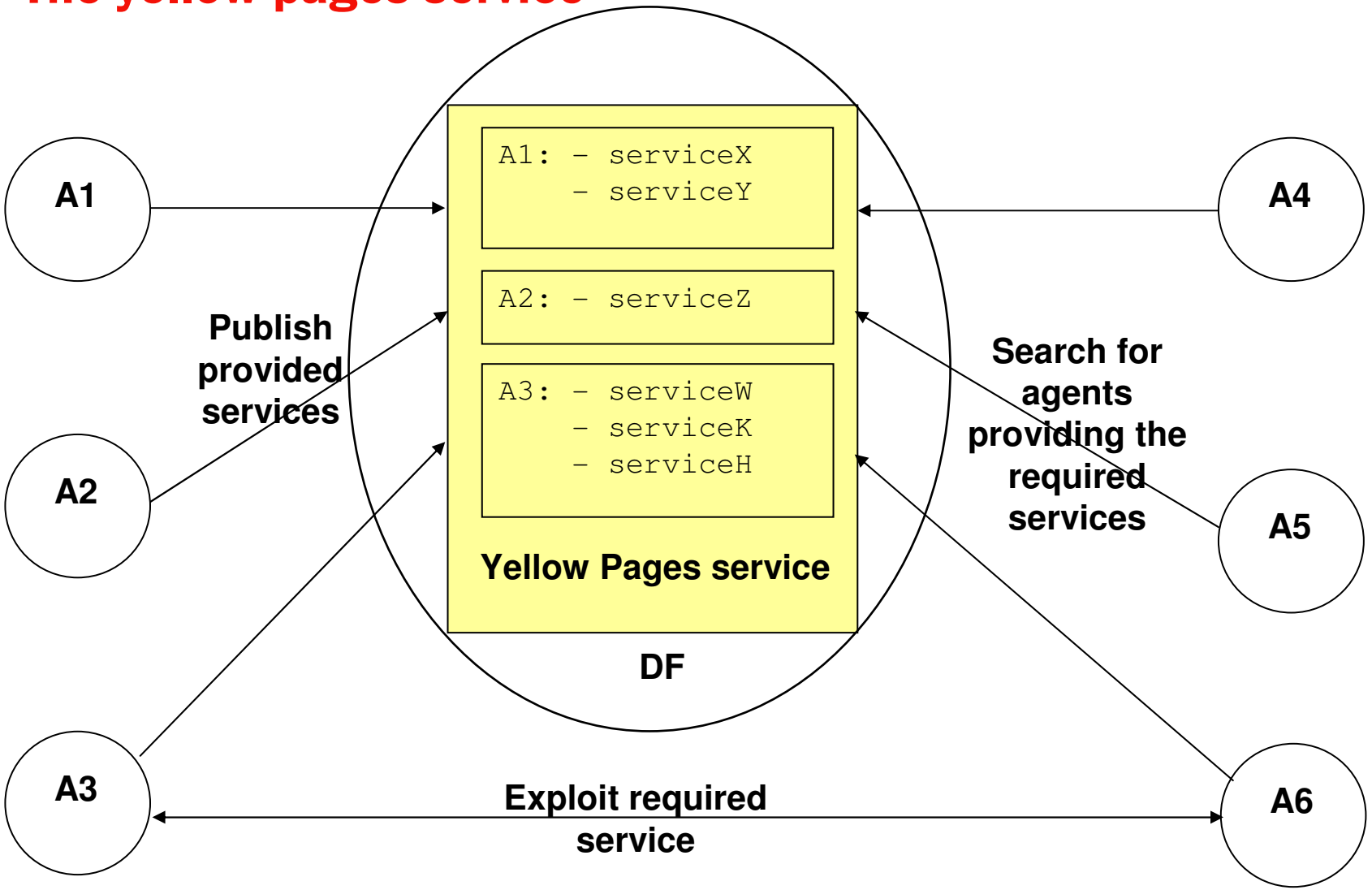
- Use `receive()` + `Behaviour.block()` to receive messages within behaviours.
- Use `blockingReceive()` to receive messages within the agent `setup()` and `takeDown()` methods.



Agenda

- Overview
- Creating Agents
- Agent tasks
- Agent communication
- The yellow pages service

The yellow pages service



Interacting with the DF Agent

- The DF is an agent and as such it communicates using ACL
- The ontology and language that the DF "understands" are specified by FIPA → It is possible to search/register to a DF agent of a remote platform.
- The `jade.domain.DFService` class provides static utility methods that facilitate the interactions with the DF
  - register();
  - modify();
  - deregister();
  - search();
- The JADE DF also supports a subscription mechanism

DFDescription format

- When an agent registers with the DF it must provide a description (implemented by the `jade.domain.FIPAAgentManagement.DFAgentDescription` class) basically composed of
  - The agent `AID`
  - A collection of service descriptions (implemented by the class `ServiceDescription`). This, on its turn, includes:
    - The service type (e.g. “Weather forecast”);
    - The service name (e.g. “Meteo-1”);
    - The languages, ontologies and interaction protocols that must be known to exploit the service
    - A collection of service-specific properties in the form key-value pair
- When an agent searches/subscribes to the DF it must specify another `DFAgentDescription` that is used as a template