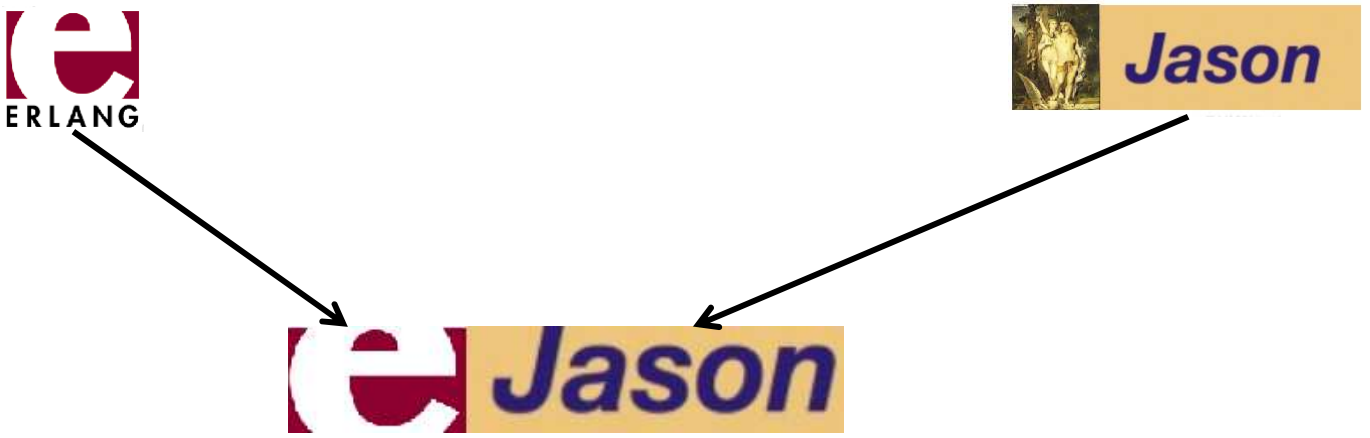
Adding Distribution and Fault Tolerance to Jason

Álvaro Fernández Díaz
Clara Benac Earle
Lars-Åke Fredlund
Universidad Politécnica de Madrid

eJason

- Concurrency
- Distribution
- Fault tolerance

- Multi-Agent Systems programming
- BDI Architecture



Jason

- Based on the Belief-Desire-Intention (BDI) Architecture
  - It is a higher-level abstraction for multiagent systems

Jason

- Based on the Belief-Desire-Intention (BDI) Architecture
  - It is a higher-level abstraction for multiagent systems

- **Actors:** (intelligent) agents

Jason

- Based on the Belief-Desire-Intention (BDI) Architecture
  - It is a higher-level abstraction for multiagent systems

- **Actors:** (intelligent) agents
  - Autonomous

Jason

- Based on the Belief-Desire-Intention (BDI) Architecture
  - It is a higher-level abstraction for multiagent systems

- **Actors:** (intelligent) agents
  - Autonomous
  - Interact via message-passing

Jason

- Based on the Belief-Desire-Intention (BDI) Architecture
  - It is a higher-level abstraction for multiagent systems

- **Actors:** (intelligent) agents
  - Autonomous
  - Interact via message-passing
  - Rational

BDI Software Architecture

- Identifies different program concerns (mental state):

BDI Software Architecture

- Identifies different program concerns (mental state):
  - Beliefs: the facts that the agents consider to be true about the world

BDI Software Architecture

- Identifies different program concerns (mental state):
  - Beliefs: the facts that the agents consider to be true about the world
  - Desires: objectives or situations that the agents would like to realise

BDI Software Architecture

- Identifies different program concerns (mental state):
  - Beliefs: the facts that the agents consider to be true about the world
  - Desires: objectives or situations that the agents would like to realise
  - Intentions: what the agents have decided to do

BDI Software Architecture

- Identifies different program concerns (mental state):
  - Beliefs: the facts that the agents consider to be true about the world
  - Desires: objectives or situations that the agents would like to realise
  - Intentions: what the agents have decided to do

- The agents also react to events.

Example: Bit Transfer Protocol

"A sender should transmit a string of bits to a receiver. It can only send one bit at a time."

Example: Bit Transfer Protocol

"A sender should transmit a string of bits to a receiver. It can only send one bit at a time."

Sender

Receiver

DESIRES

Sender

Receiver

DESIRES

The string of bits is correctly transmitted from the sender to the receiver

Sender

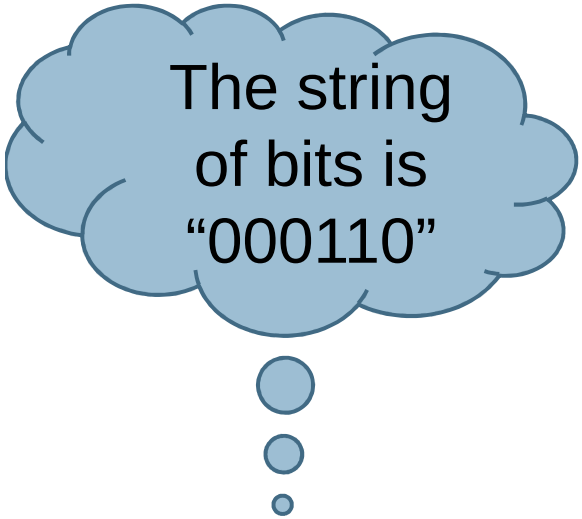
Receiver

BELIEFS

Sender

Receiver

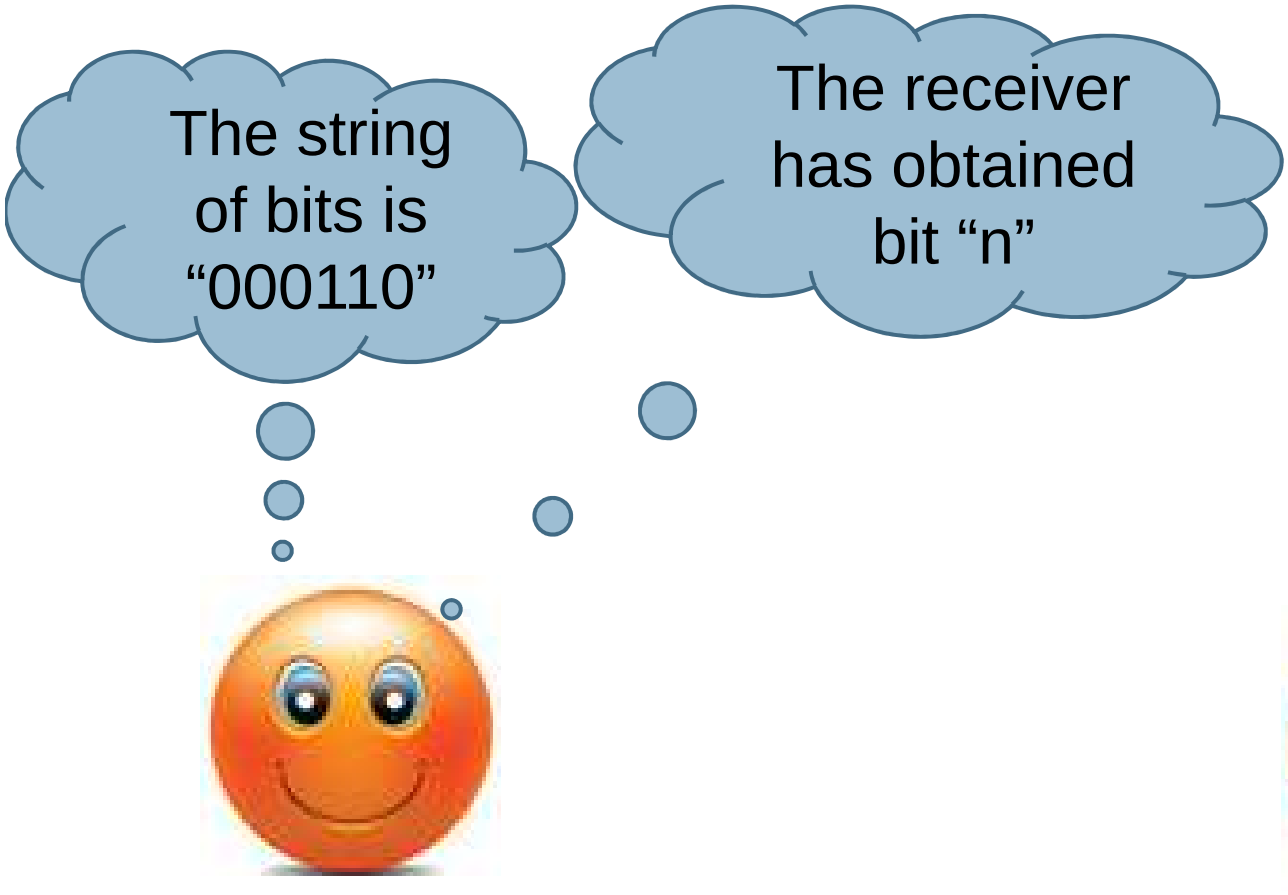
BELIEFS



Sender

Receiver

BELIEFS


The string of bits is "000110"
The receiver has obtained bit "n"
Sender
Receiver

BELIEFS

The string of bits is “000110”

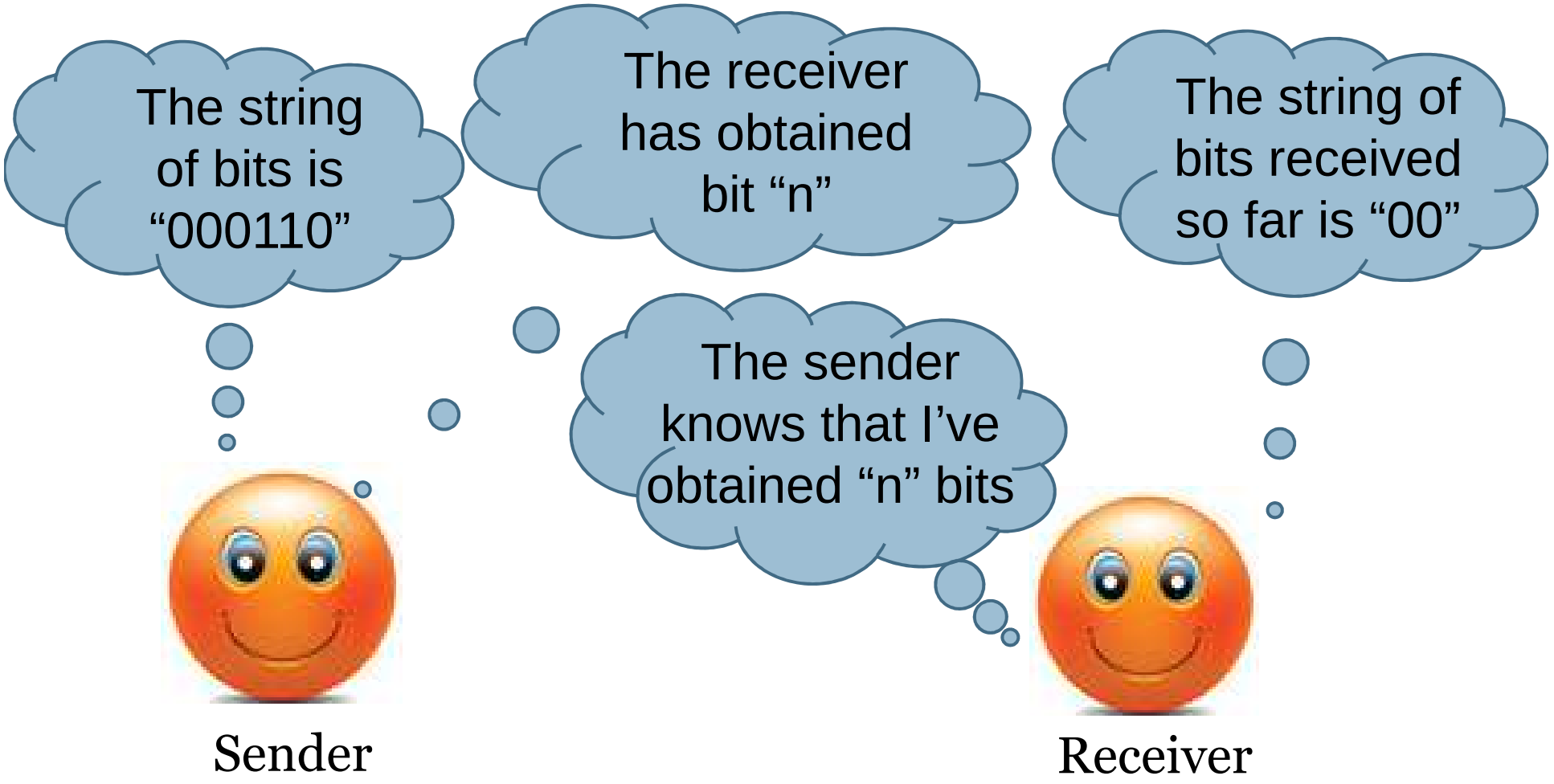
The receiver has obtained bit “n”

The string of bits received so far is “00”

Sender

Receiver

BELIEFS


The string of bits is “000110”
The receiver has obtained bit “n”
The string of bits received so far is “00”
The sender knows that I’ve obtained “n” bits
Sender
Receiver

INTENTIONS

Sender

Receiver

INTENTIONS

If I believe that the receiver has not obtained any bit, I send the first bit to it.

If I believe that the receiver has obtained the bit at position “n”, I send the bit at “n+1”.

Sender

Receiver

INTENTIONS

If I believe that the receiver has not obtained any bit, I send the first bit to it.

If I believe that the receiver has obtained the bit at position “n”, I send the bit at “n+1”.

Sender

If I know the bit at position “n”, I send an acknowledgement message “n” to the receiver.

Receiver

INTENTIONS

If I believe that the receiver has not obtained any bit, I send the first bit to it.

If I believe that the receiver has obtained the bit at position “n”, I send the bit at “n+1”.

Sender

If I know the bit at position “n”, I send an acknowledgement message “n” to the receiver.

Event:
Bit message

Receiver

INTENTIONS

If I believe that the receiver has not obtained any bit, I send the first bit to it.

If I believe that the receiver has obtained the bit at position “n”, I send the bit at “n+1”.

If I know the bit at position “n”, I send an acknowledgement message “n” to the receiver.

Sender

Event: Ack message

Event: Bit message

Receiver

Jason: JAVA Implementation

Jason: JAVA Implementation

- Distribution by interfacing to JADE
  - Own message-passing implementation

Jason: JAVA Implementation

- Distribution by interfacing to JADE
  - Own message-passing implementation

- No built-in fault tolerance mechanisms

Jason: JAVA Implementation

- Distribution by interfacing to JADE
  - Own message-passing implementation
- No built-in fault tolerance mechanisms
  - Can be implemented using e.g. timers
    - Error prone

Jason: JAVA Implementation

- Distribution by interfacing to JADE
  - Own message-passing implementation
- No built-in fault tolerance mechanisms
  - Can be implemented using e.g. timers
    - Error prone
- Efficiency problems
  - Up to few thousands of agents
  - Slow message-passing

Erlang

- **Actors:** (light-weight) processes
  - Efficient creation
  - Asynchronous communication

Erlang

- **Actors:** (light-weight) processes
  - Efficient creation
  - Asynchronous communication

- Provides easy-to-use mechanisms supporting:
  - Process distribution
  - Fault tolerance

Erlang

- Process distribution:
  - The Erlang processes run in Erlang nodes
  - A host may contain several running Erlang nodes

Erlang

- Process distribution:
  - The Erlang processes run in Erlang nodes
  - A host may contain several running Erlang nodes

- Fault tolerance:

Erlang

- Process distribution:
  - The Erlang processes run in Erlang nodes
  - A host may contain several running Erlang nodes

- Fault tolerance:
  - Erlang processes can observe process and node failures.

eJason

Efficiency &
Ease of use

- Concurrency
- Distribution
- Fault tolerance

ERLANG

Abstraction

- Multi-Agent Systems programming
- BDI Architecture

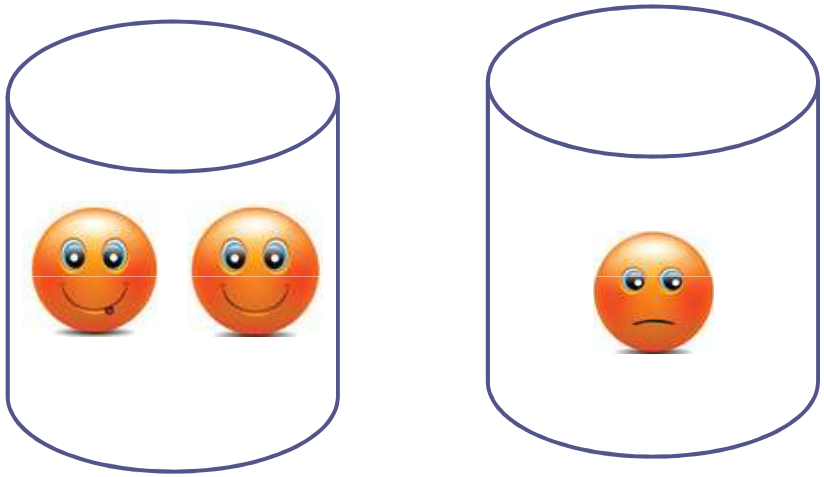
Jason

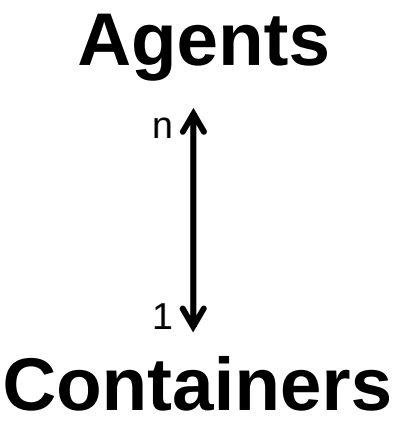
Distribution Schema

Distribution Schema

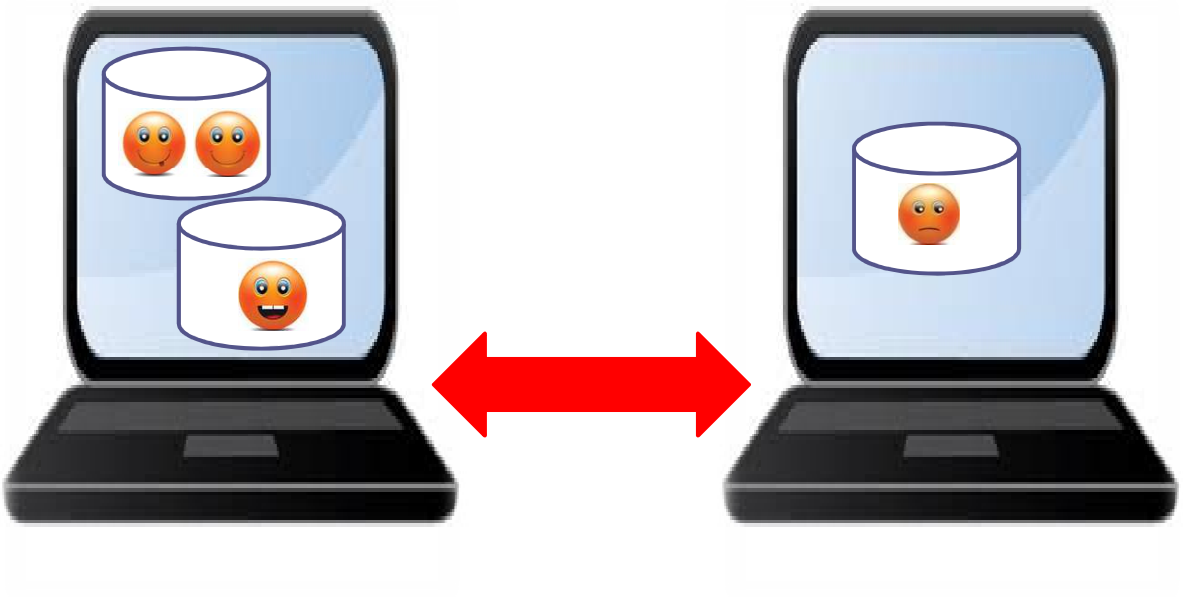
Agents

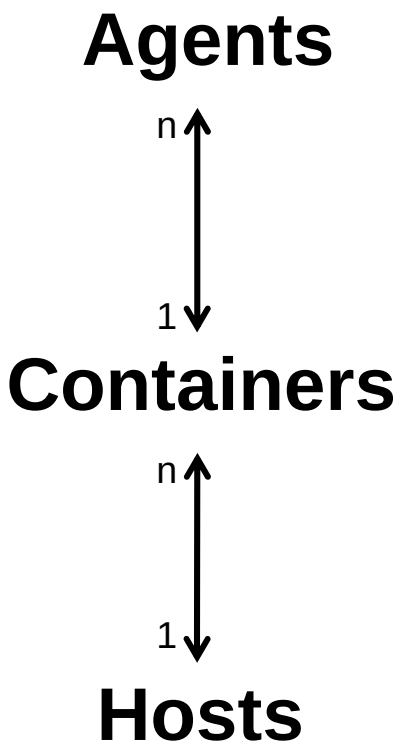
Distribution Schema





Distribution Schema





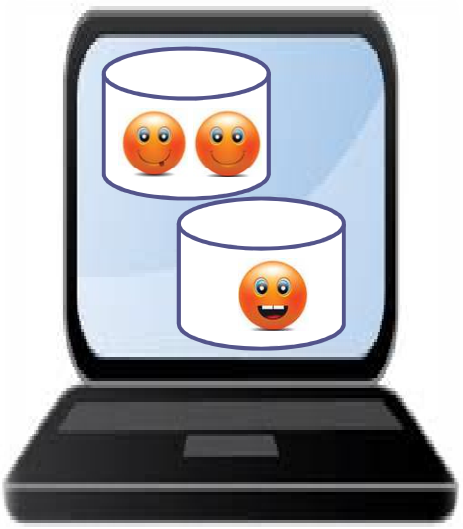
Distribution mapping

Jason	Erlang Counterpart
Agent	Erlang process
Container	Erlang node (Erlang runtime system)

Distribution mapping

Jason	Erlang Counterpart
Agent	Erlang process
Container	Erlang node (Erlang runtime system)
Agent Naming	Erlang process registry

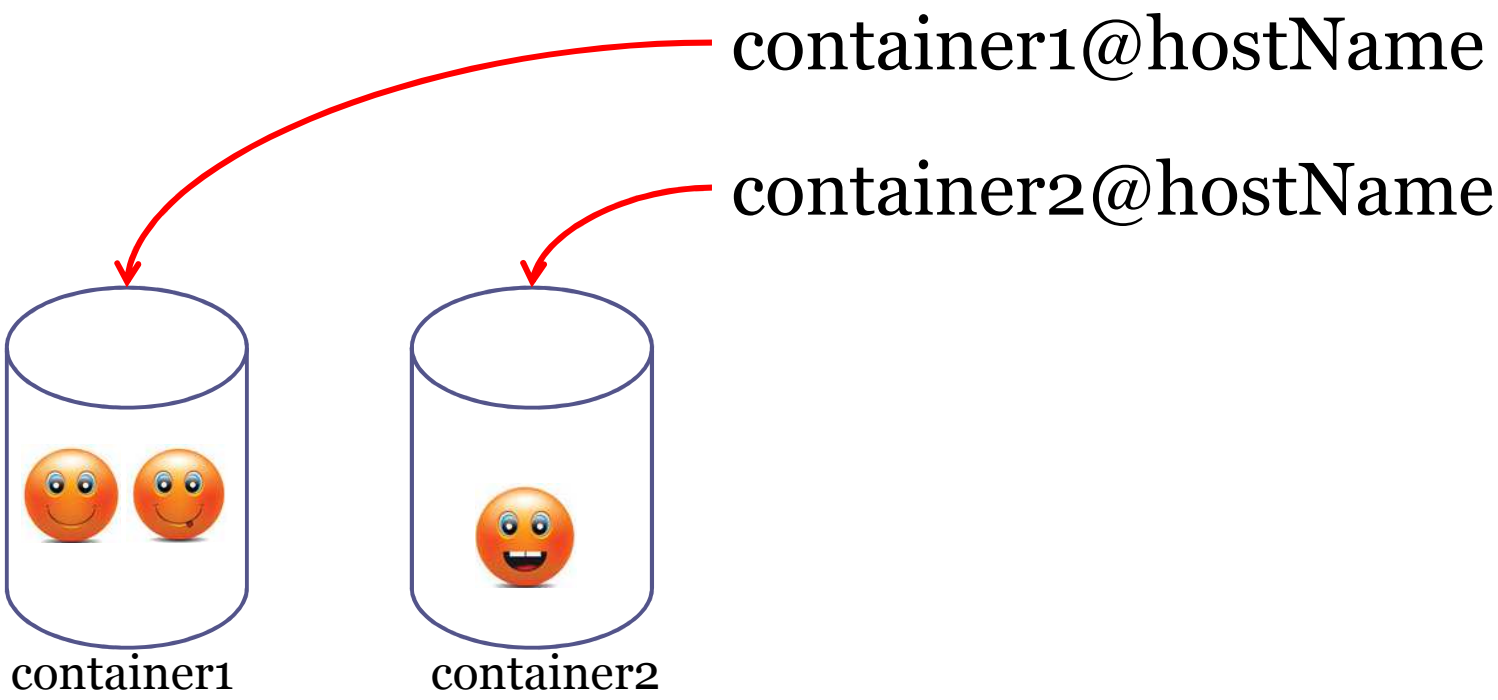
Distribution Schema:naming



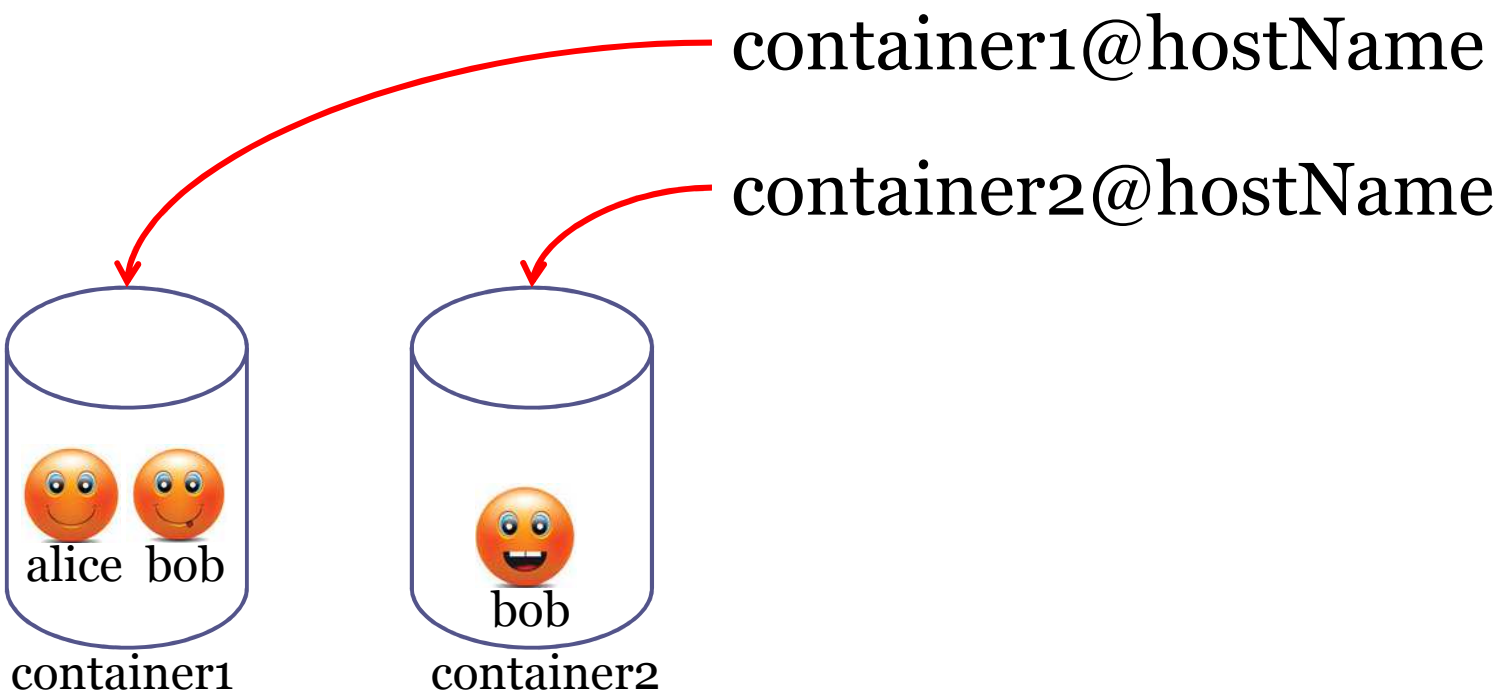
hostName

hostName

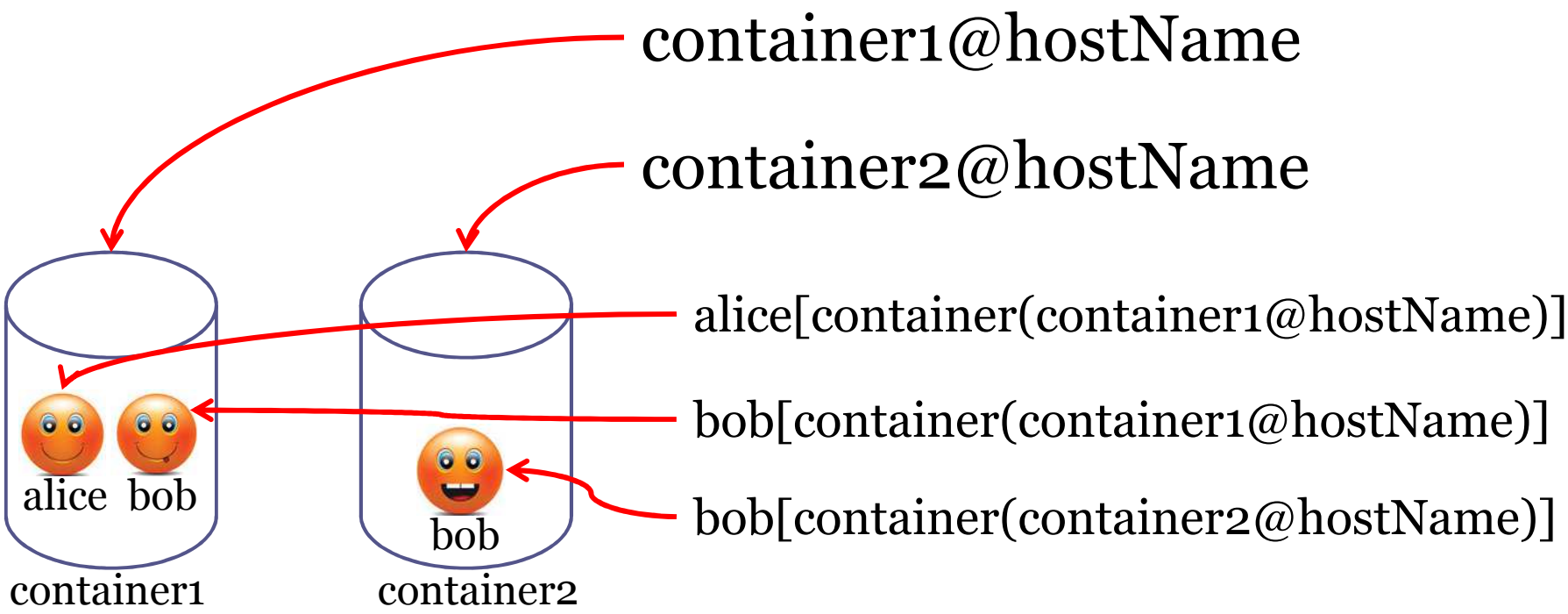
Distribution Schema:naming



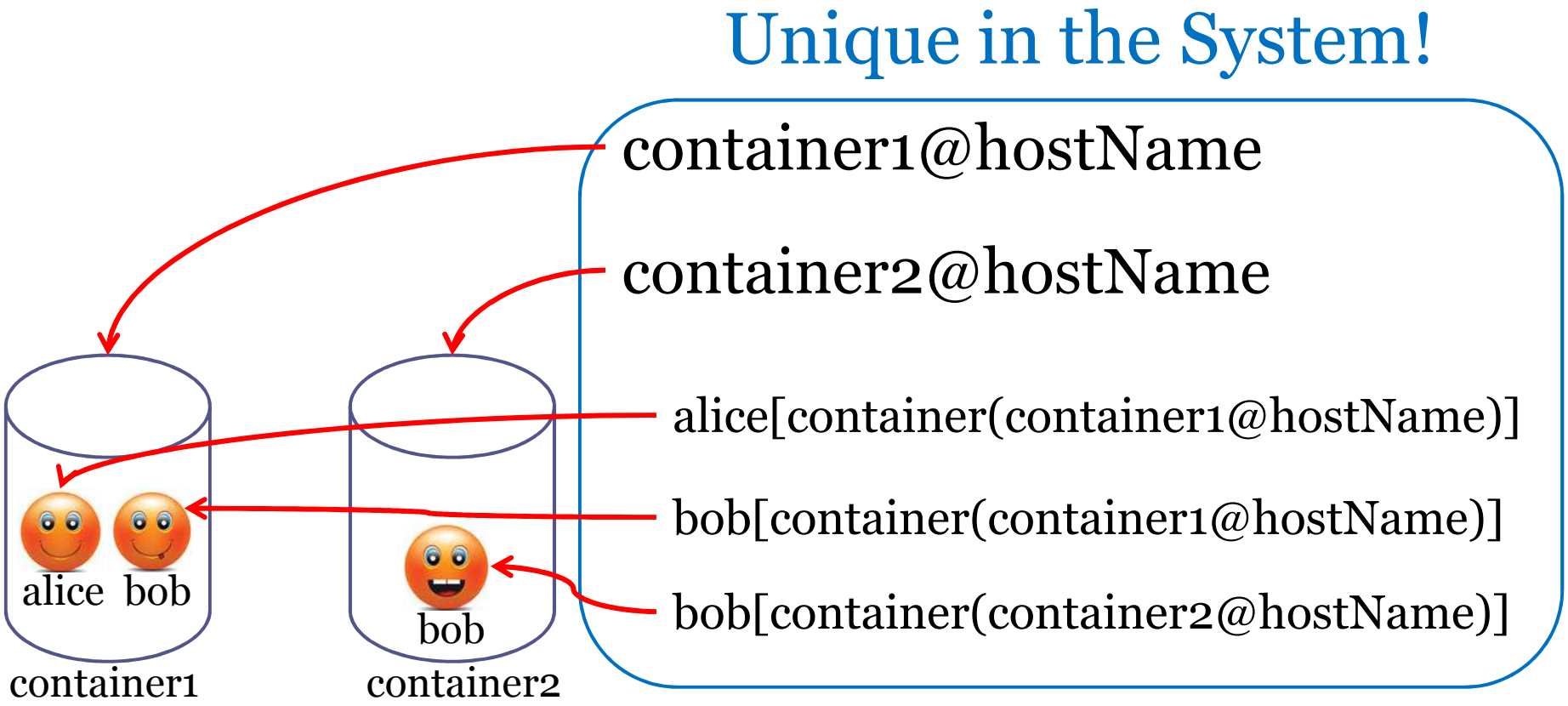
Distribution Schema:naming



Distribution Schema:naming



Distribution Schema:naming

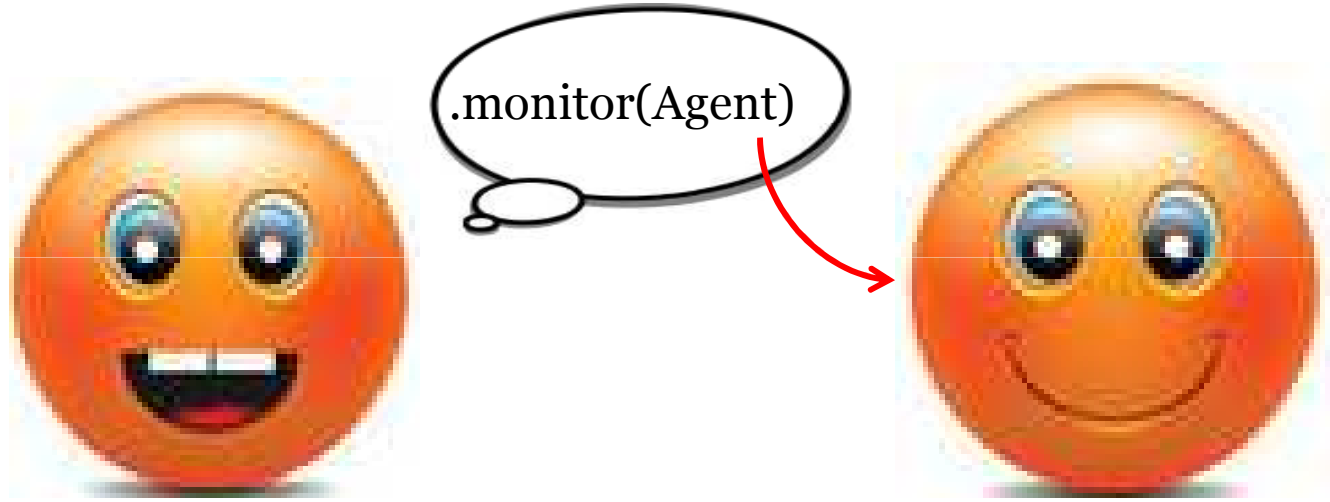


Fault Tolerance: agent monitoring

Monitoring Agent

Monitored Agent

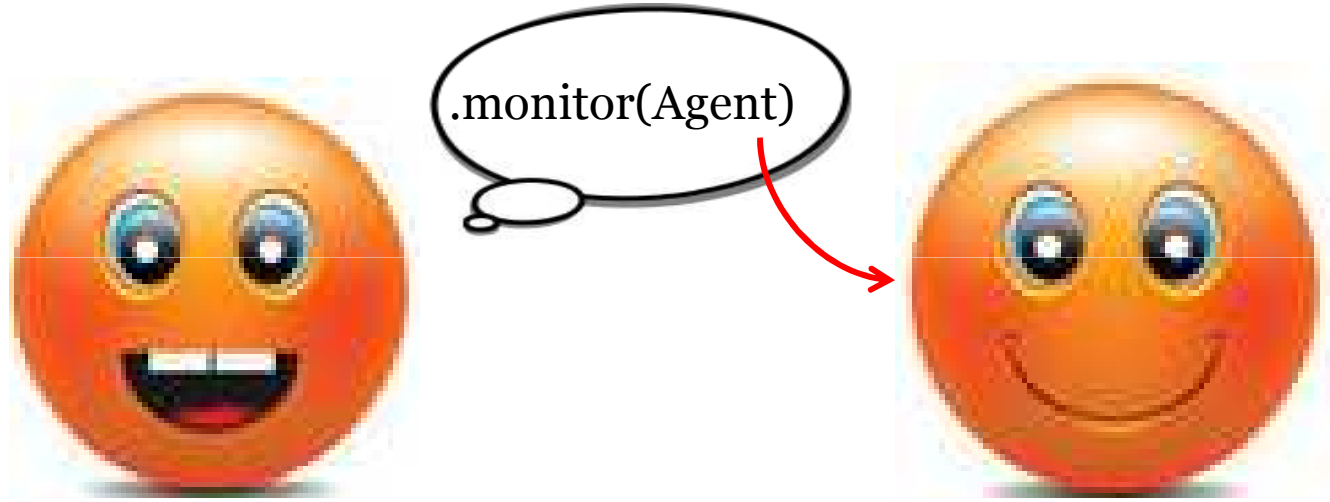
Fault Tolerance: agent monitoring



Monitoring Agent **Monitored Agent**

- Explicit monitoring request

Fault Tolerance: agent monitoring

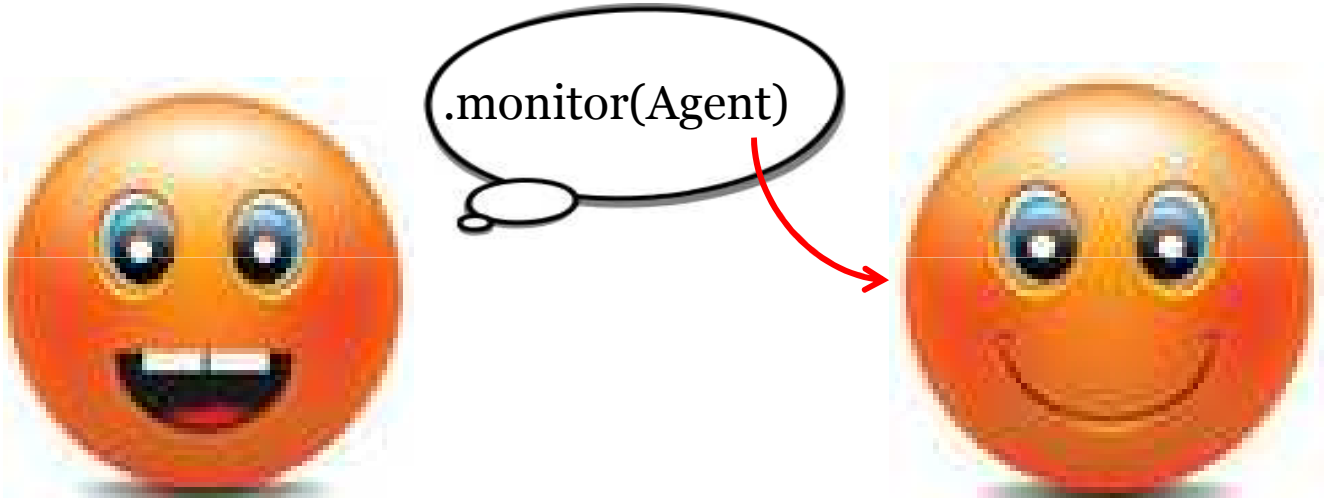


Monitoring Agent

Monitored Agent

- Explicit monitoring request
- Monitors the state of the monitored agent.

Fault Tolerance: agent monitoring



Monitoring Agent

- Explicit monitoring request
- Monitors the state of the monitored agent.

Monitored Agent

- Passive member

Fault Tolerance: failures

Monitoring Agent

Monitored Agent

Fault Tolerance: failures

runtime exception

Monitoring Agent

Monitored Agent

Fault Tolerance: failures

Monitoring Agent

Monitored Agent

runtime exception

kill_agent

Fault Tolerance: failures

Monitoring Agent

Monitored Agent

runtime exception

kill_agent

Container destroyed

Fault Tolerance: failures

Monitoring Agent

Monitored Agent

runtime exception

kill_agent

Container destroyed

Fault Tolerance: failures

Monitoring Agent

dead_agent

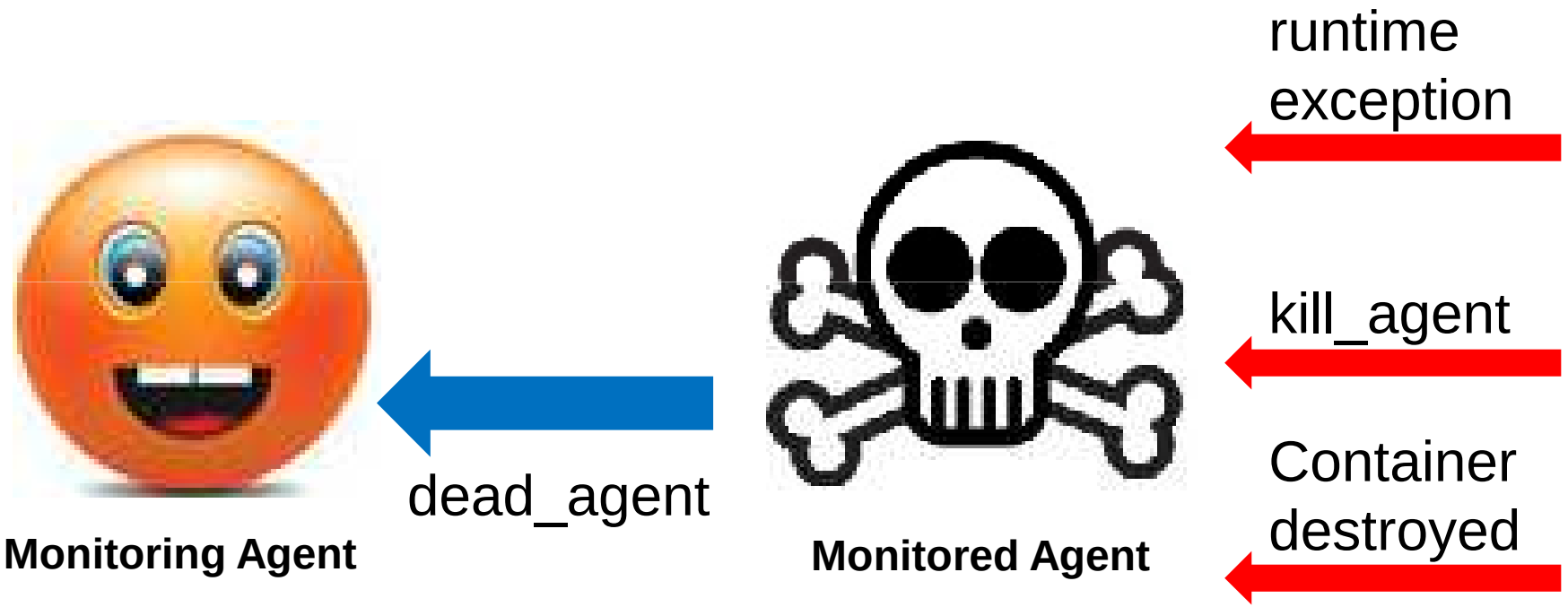
Monitored Agent

runtime exception

kill_agent

Container destroyed

Fault Tolerance: failures



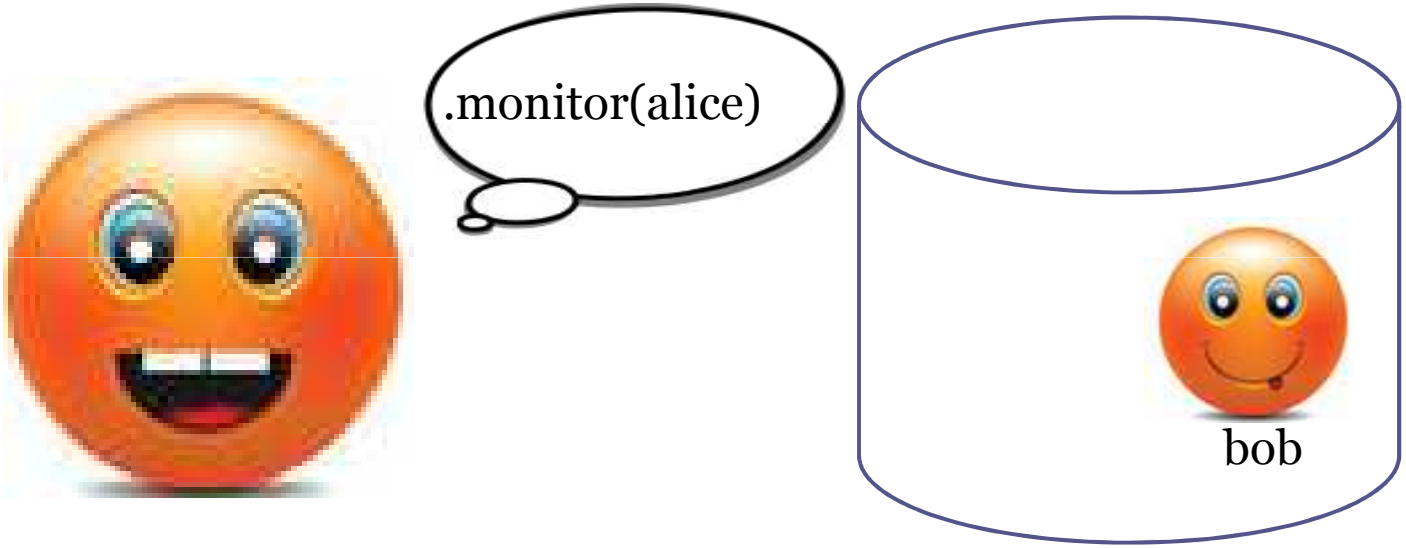
New Jason belief:

```
+agent_down(Agent)[reason(dead_agent)].
```

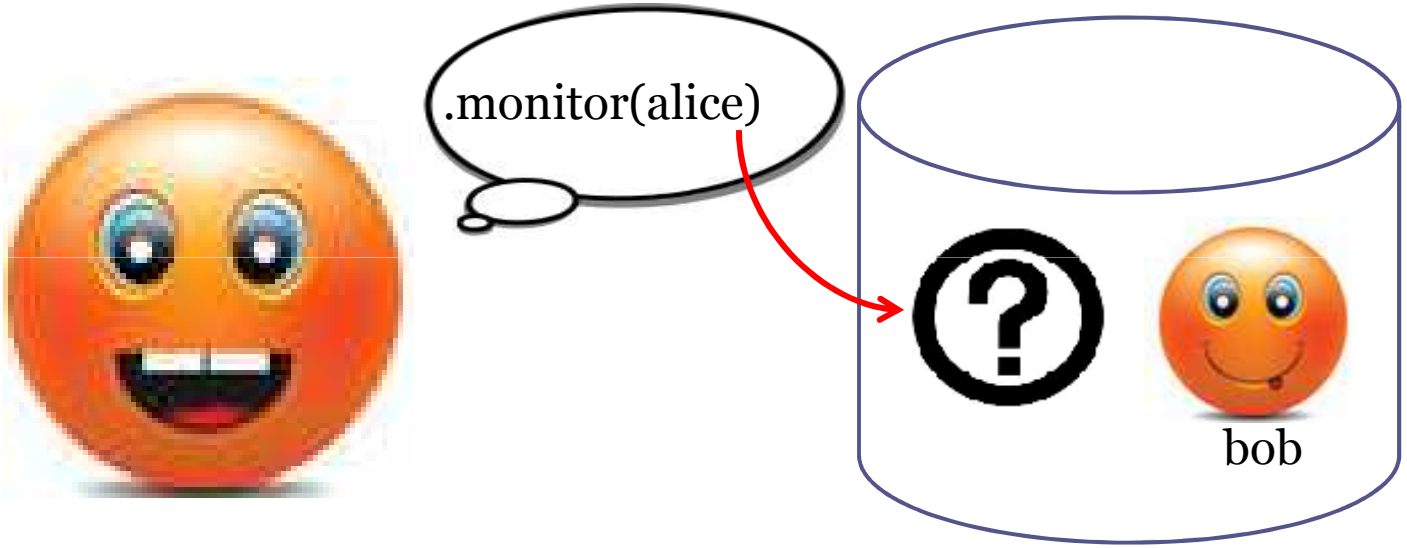
Fault Tolerance: failures

Fault Tolerance: failures

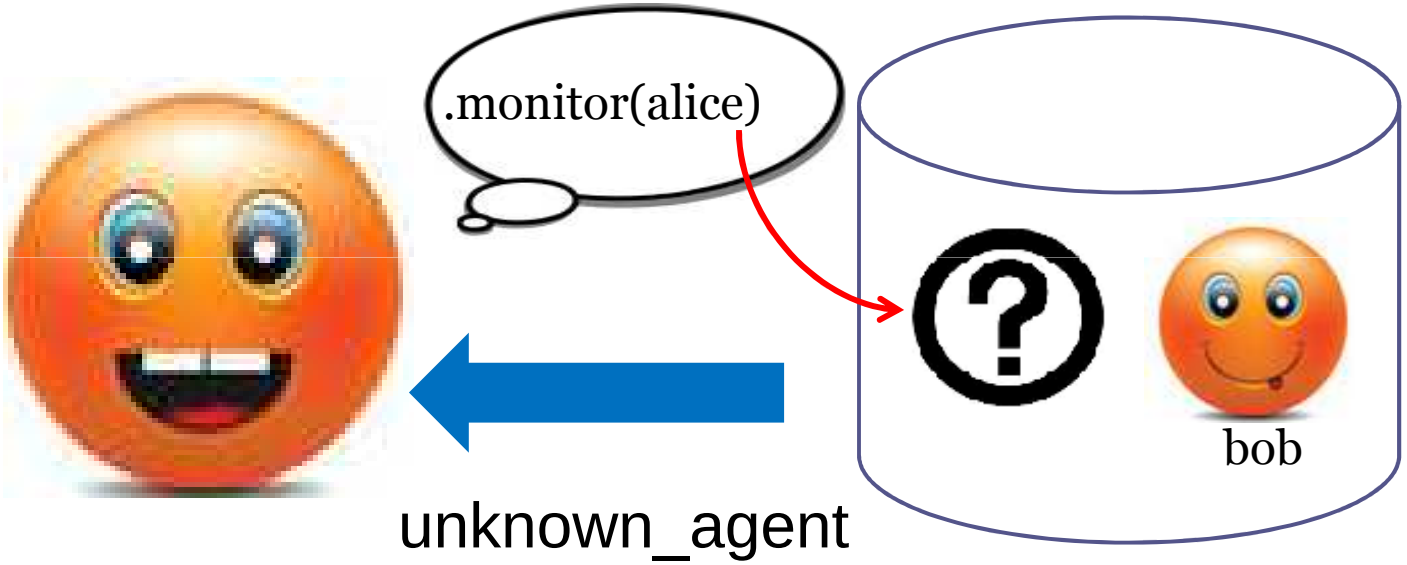
Fault Tolerance: failures



Fault Tolerance: failures



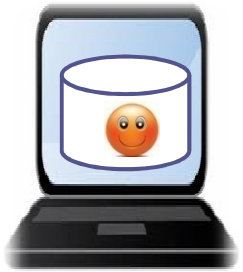
Fault Tolerance: failures



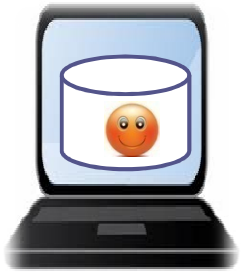
New Jason belief:

```
+agent_down(Agent)[reason(unknown_agent)].
```

Fault Tolerance: failures

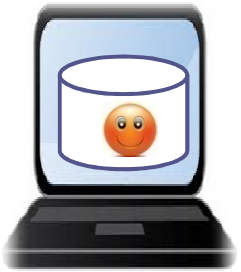


Fault Tolerance: failures



Broken link

Fault Tolerance: failures



Broken link

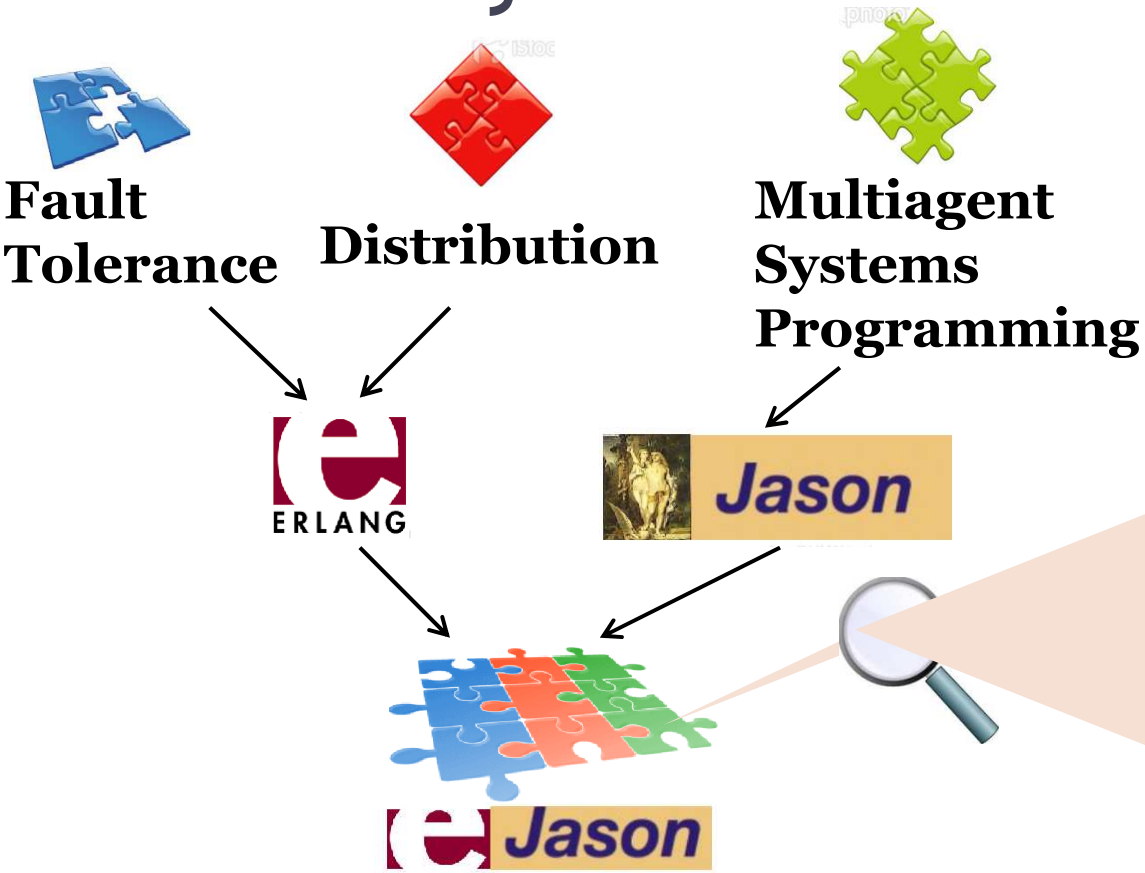
Unknown host

Fault Tolerance: failures

New Jason belief:

```
+agent_down(Agent)[reason(unreachable_agent)].
```

Summary


Fault Tolerance
Distribution
Multiagent Systems Programming
ERLANG
Jason
Jason
Distributed and Fault Tolerant Multiagent Systems
Agents aware of:
•Existence of other agents
•Deaths of other agents
•Isolation from other agents

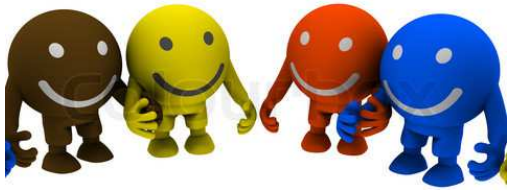
Future Lines of Work

- Continue integrating in eJason all Jason features and capabilities.
  - Interaction with the environment

Future Lines of Work

- Continue integrating in eJason all Jason features and capabilities.
  - Interaction with the environment

- Interoperability with agents running in other platforms (JADE, Java-Jason)
  - FIPA-compliance



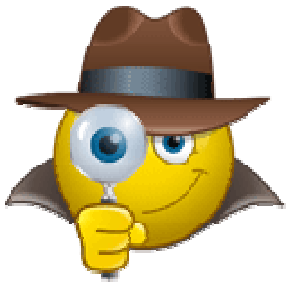
Future Lines of Work

- Improve agent monitoring
  - Define and implement semantic termination
  - Notification upon agent creation

Future Lines of Work

- Improve agent monitoring
  - Define and implement semantic termination
  - Notification upon agent creation

- Higher-level agents
  - Predefined behavior (i.e. built-in mental state)



Future Lines of Work

- Agent (inter-container) mobility

Future Lines of Work

- Agent (inter-container) mobility

- Study the applicability of eJason
  - Can we turn Jason into a good distributed systems programming language?

Download eJason

- https://github.com/avalor/eJason

Download eJason

- https://github.com/avalor/eJason

Questions & Answers