CLRF - Common LISP Rescue Framework

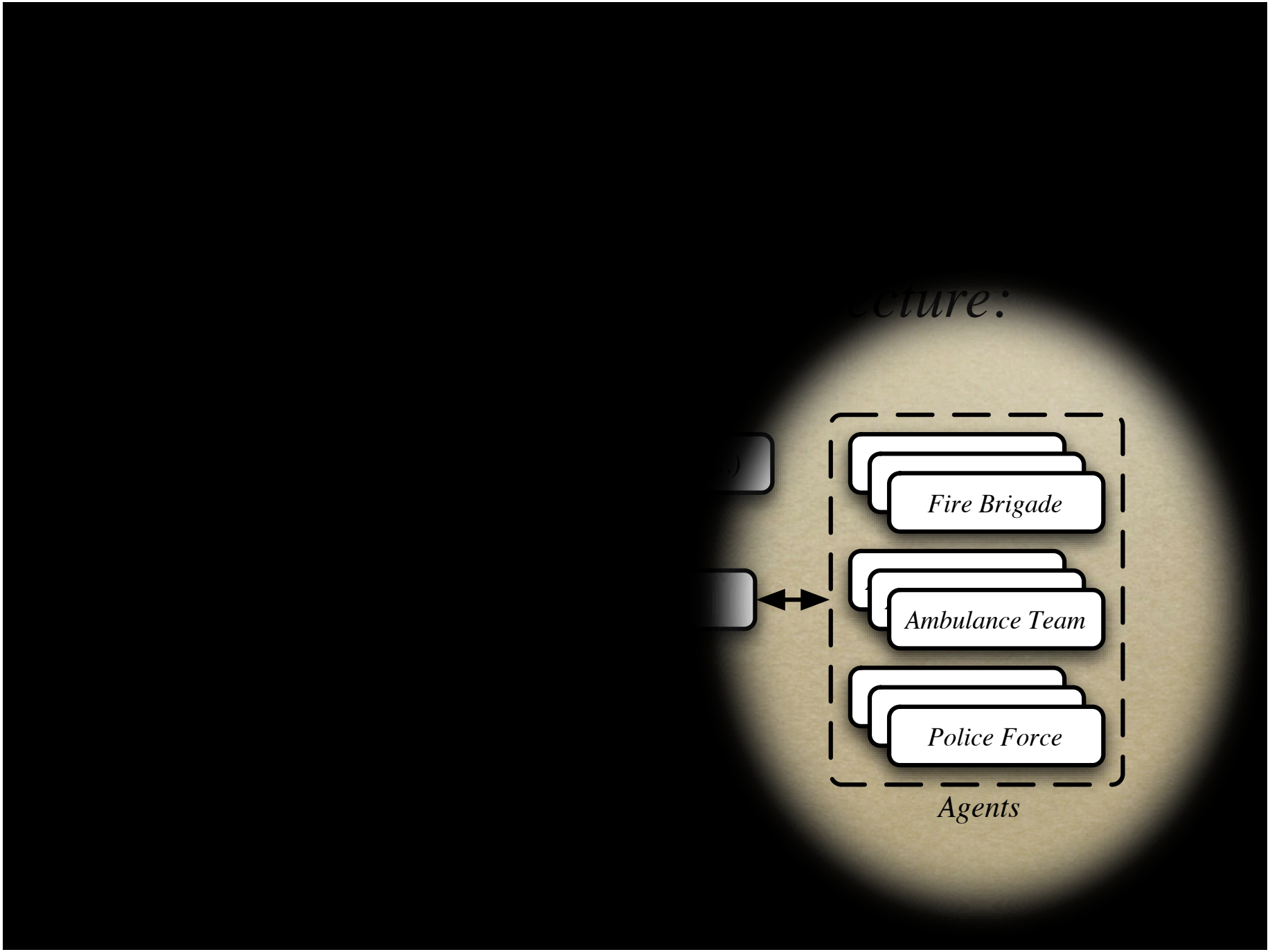
A software framework for building RoboCup Rescue agents in LISP

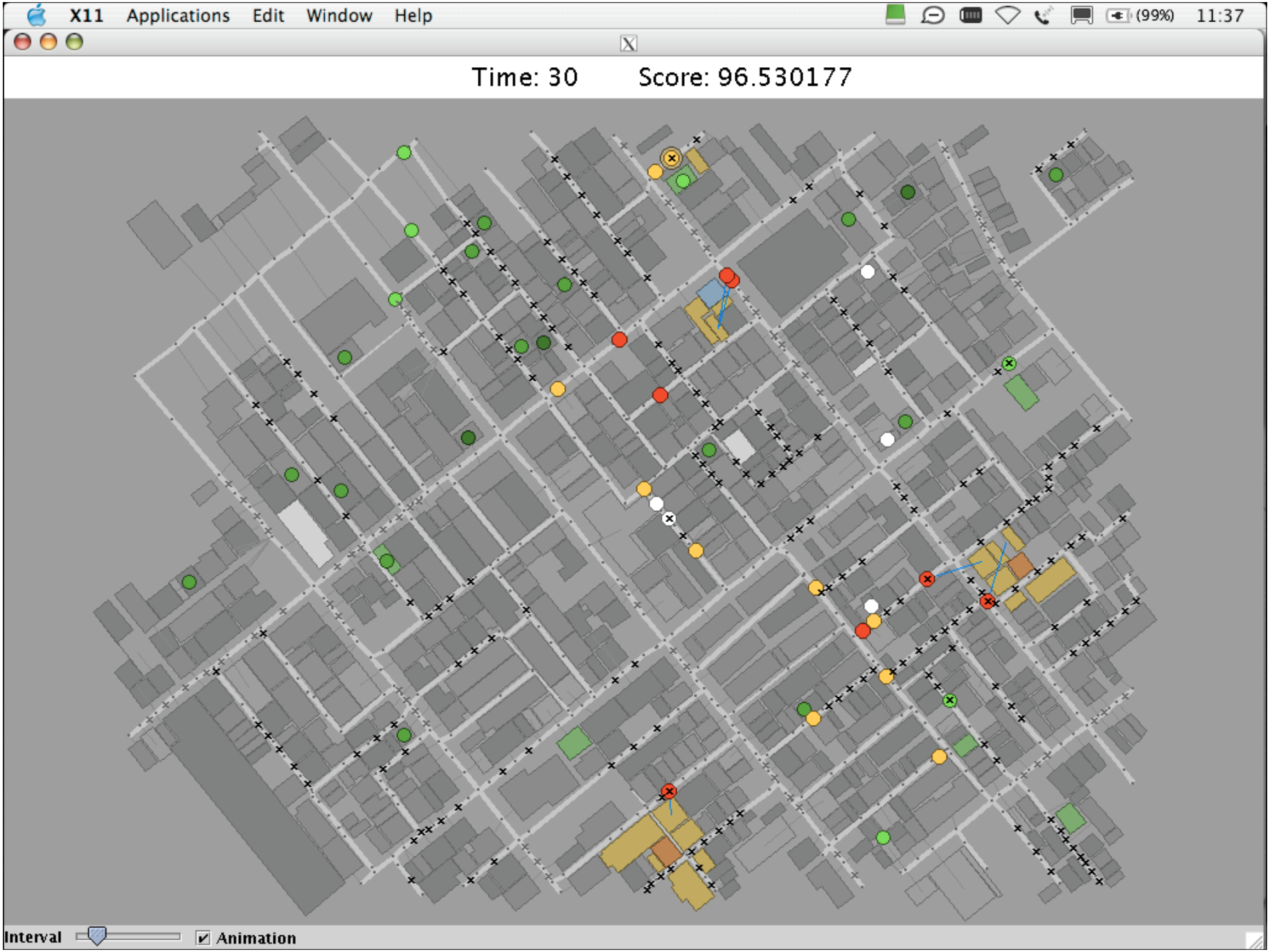
Introduction

- *RoboCup is a worldwide organization that pursues new technology development*
- *Initial goal: building a team of robots that may win the best human football team in 2050.*
- *Influenced by the Kobe earthquake in 1995, some RoboCup related people created the RoboCup Rescue competition*

Introduction

- *RoboCup Rescue simulates an earthquake in a city.*
- *Detailed city map, based on GIS.*
- *Six types of "controllable" agents:*
  - *Fire brigades (and fire stations)*
  - *Police forces (and police stations)*
  - *Ambulance Teams (and ambulance centers)*


Fire Brigade
Ambulance Team
Police Force
Agents


X11 Applications Edit Window Help
(99%) 11:37
Time: 30 Score: 96.530177
Interval
Animation

Overview

- *CLRF is a framework, made in LISP, that allows a rapid development of RR agents*
- *CLRF comes with all the communication layer implemented, a flexible world representation model, and comprehensive documentation*

Overview

- *CLRF advantages over existing Java framework:*
  - *It's not in Java!*
  - *Thread-based structure*
  - *Easy graphical world representation*
  - *Very flexible world representation support*

Overview

- *CLRF advantages (cont.):*
  - *Automatically updated internal world representation*
  - *Command prompt support (allows for the introduction of LISP commands during agent's execution, or even code replacement)*
  - *Easy information logging*

Why LISP?

- *LISP Macros are a very powerful feature that allows the language to extend itself, using itself to do it!*
  - *LISP Macros are not just find/replace operations, but LISP code that is actually executed*
  - *CLOS is a spectacular example of the power of LISP Macros*

Why LISP?

- *LISP programs may be compiled, interpreted, or both. You don't even need to know if your code is compiled or not*
- *LISP allows the programmer to replace code in the runtime, reducing tedious fix/compile/run cycles*

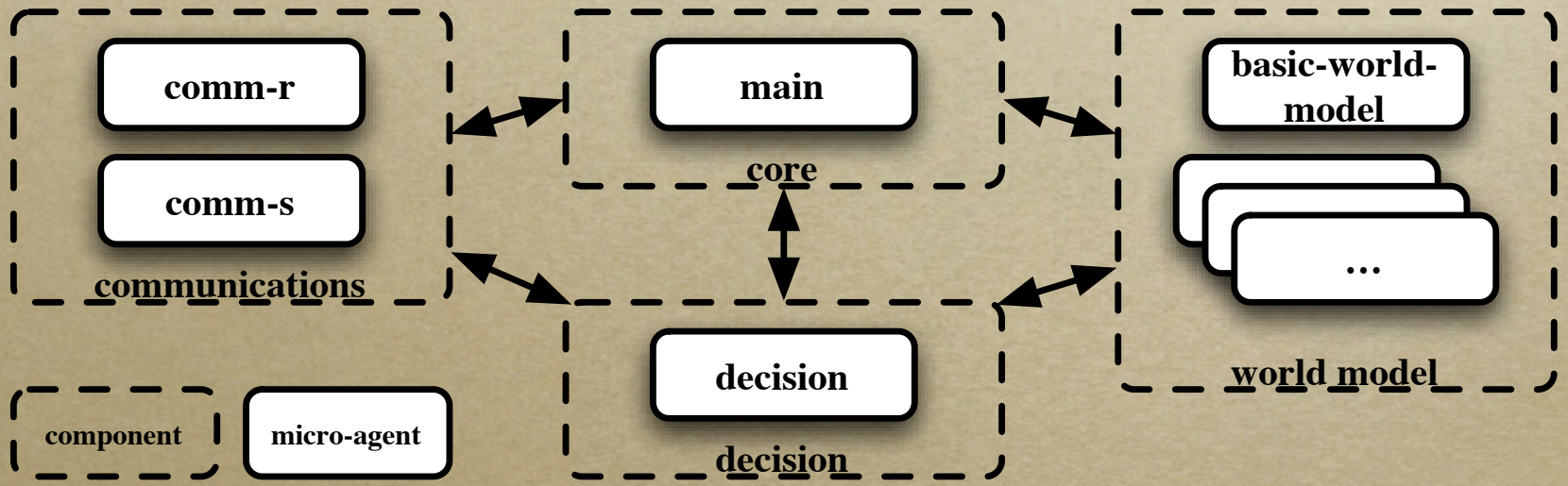
Why LISP?

- *Automatic memory management*
- *Advanced collection types built in the language itself (not external libraries)*
- *Weak typification – only care about the variable types when needed (performance)*

Why LISP?

- *Symbol management (no need to "re-invent" the symbol type, it's already there)*
- *Advanced control structures*

CLRF Architecture

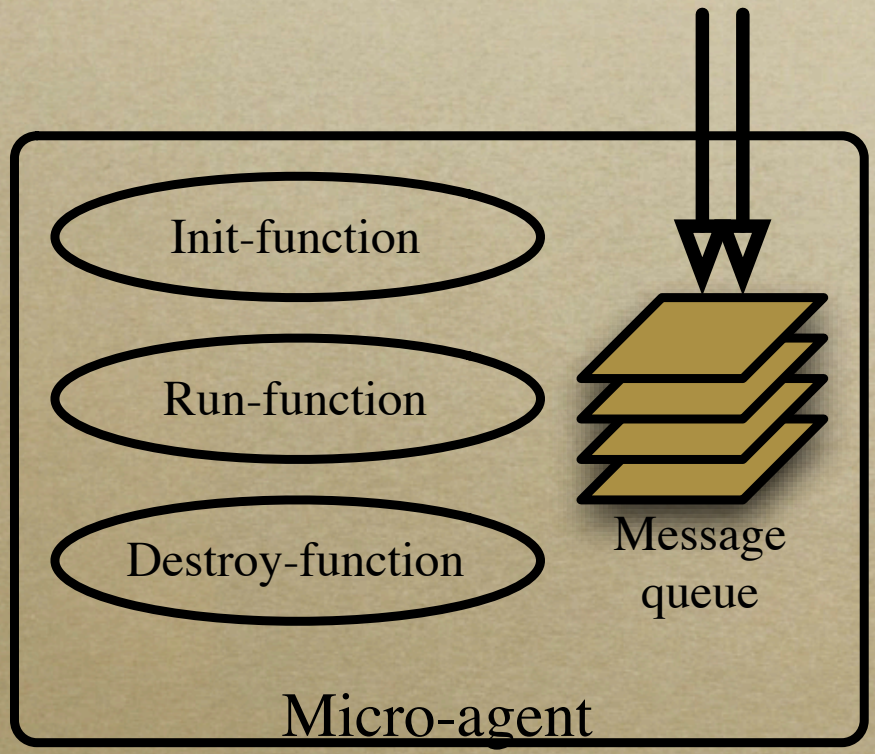


CLRF Architecture

- *A microagent is an "intelligent" thread*
- *A microagent knows how to initialize, run and destroy itself*
- *Each microagent has a message queue for receiving messages (no blackboard)*

CLRF Architecture

- *Microagent representation:*



CLRF Architecture

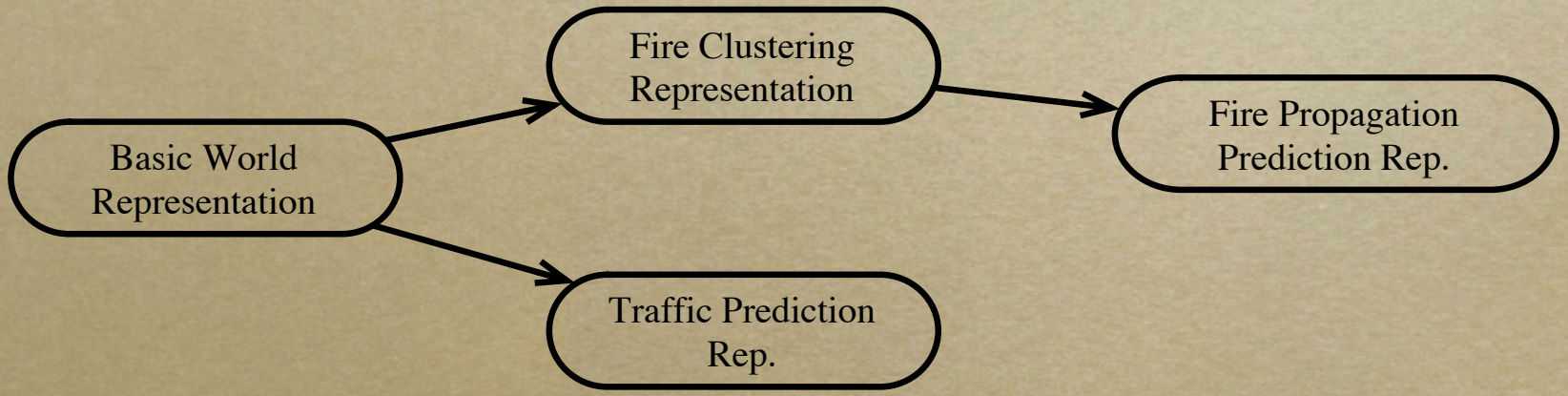
- *Improvements over SocRob microagents:*
  - *Each microagent is responsible for it's own data (approaches OOP paradigm)*
  - *Data is represented in a organized way, and not just a bunch of variables*
  - *Each microagent handles data access synchronization and timing (ex: wait for next cycle)*

World Representation

- *An agent may need different kinds of world models, each one with a different level of granularity*
- *Problems:*
  - *How to integrate all the world models into the agent*
  - *How to keep all the models updated in a coherent way*
  - *How to resolve dependencies between models?*

World Representation

- *Solution: graph of microagents that controls information flow*



World Representation

- *World representation micro-agents register themselves, informing the framework of their dependencies*
- *When a "sense" message arrives, it's "injected" in the graph, on the nodes that don't depend on others*

World Representation

- *After processing a message, the micro-agent asks the framework to forward the message to the next micro-agent(s) in the graph*
- *A microagent may safely request informations from a previous microagent (it's updated for sure)*

Framework Expansion

- *Easy to add new sent or received messages:*

```
(defclass ak-clear-message (message)
  ((id :accessor message-id :initarg :id)
   (target :accessor message-target :initarg :target)))


(defmethod build-message-data ((msg ak-clear-message))
  (write-value
   (write-value
    (write-value
     (write-header *AK-CLEAR-HEADER*)
     (message-id msg))
    (message-target msg))
   *HEADER-NULL*))
```

Build an agent in 3 steps

- *1. Write the "decision" microagent*
- *2. Adjust the message dispatch table*
- *3. Test it!*

What CLRF Offers

- *All the communication code implemented*
- *A strong support for multi-thread execution, including data representation and transfer*

What CLRF Offers

- *Flexible world models*
- *Precise documentation*
- *Opportunity to use the best language for this job – LISP – with minimal effort*

Q&A

Miguel Arroz
arroz@guiamac.com