Multi-agent systems

Andrej Lúčny

KAI FMFI UK

lucny@fmph.uniba.sk

http://www.agentspace.org/mas

MAS API

- What interface is provided to the application layer by a multi-agent system implementation?

There are two choices

- Prevailing direct communication (peer-to-peer) (e.g. JADE)
- Prevailing indirect communication (stigmergic communication) (e.g. Cougaar)

MAS
with prevailing indirect communication

Indirect communication services

- Space provides to agents services, by which they can manipulate data stored in the Space
- The services are

  READ

  WRITE

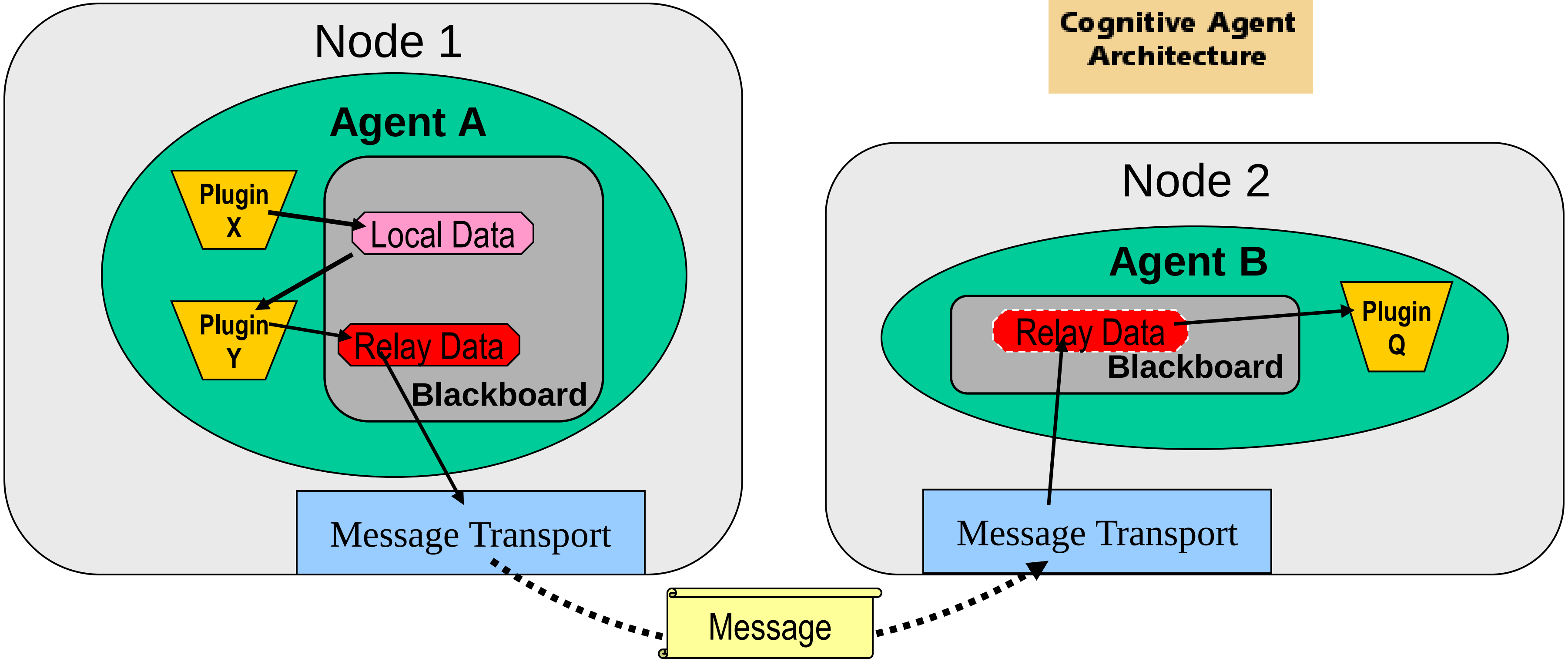
  DELETE
- Non-blocking and blocking (synchronized)

Indirect communication platform example

Cougaar

Cougaar

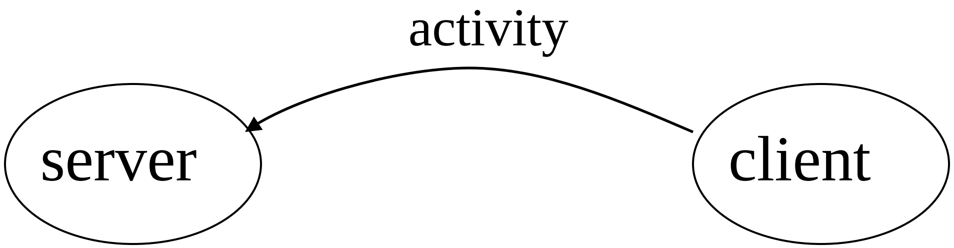
Cognitive Agent Architecture



- MAS with indirect communication is a distributed system of type:

Client-Server

- relation between two processes, one of them (server) provides a service from another one (client) on its request



Structural organization of server

Server can process request in various ways:

1. Each request separately
2. It can remember state of communication in data attached to request and response
3. It can remember state of communication at the server side

Structural organization of server

Server can process request in various ways:

easy

1. Each request separately
2. It can remember state of communication in data attached to request and response

   dangerous

3. It can remember state of communication at the server side

   universal

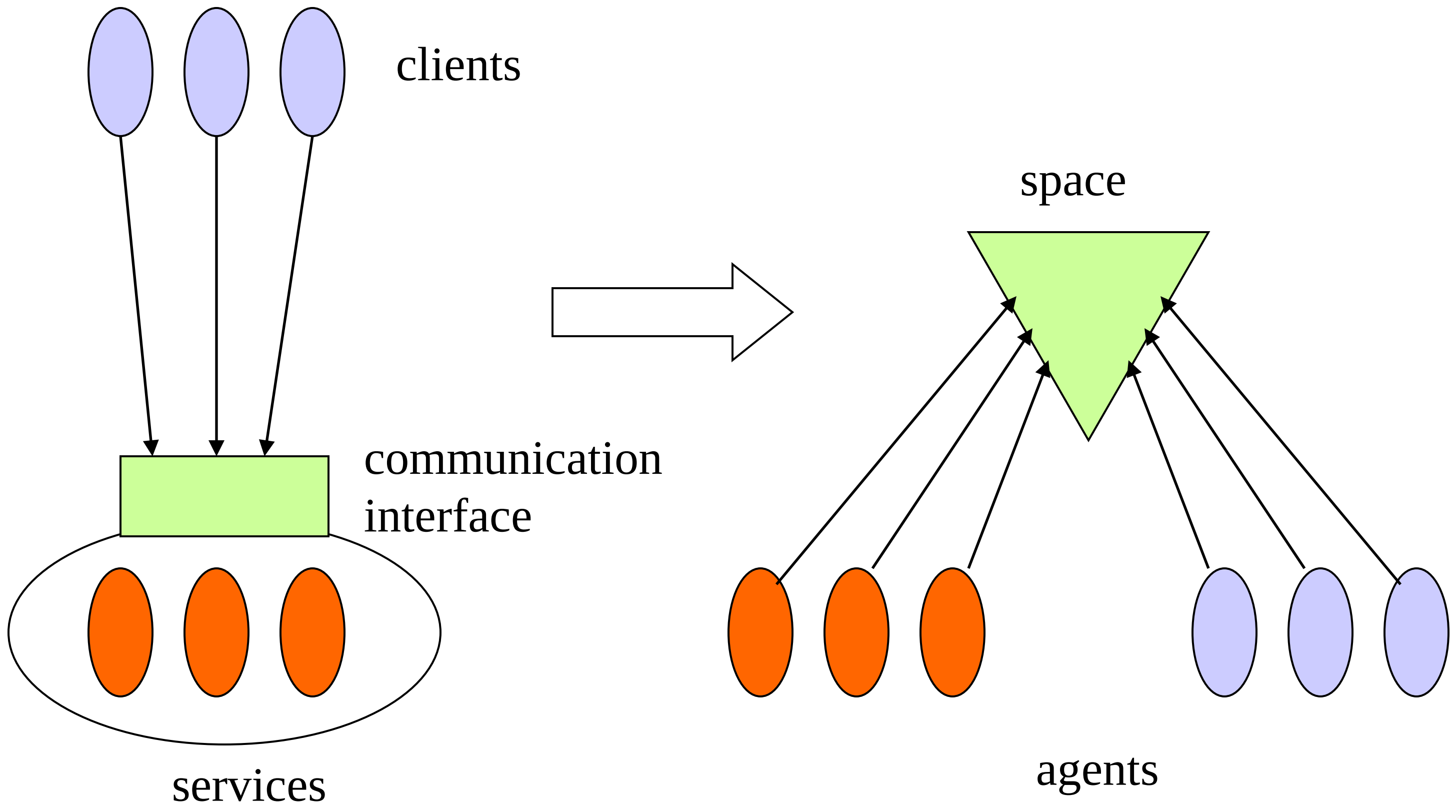
   - in structure called **port**

MAS is a special kind of DS

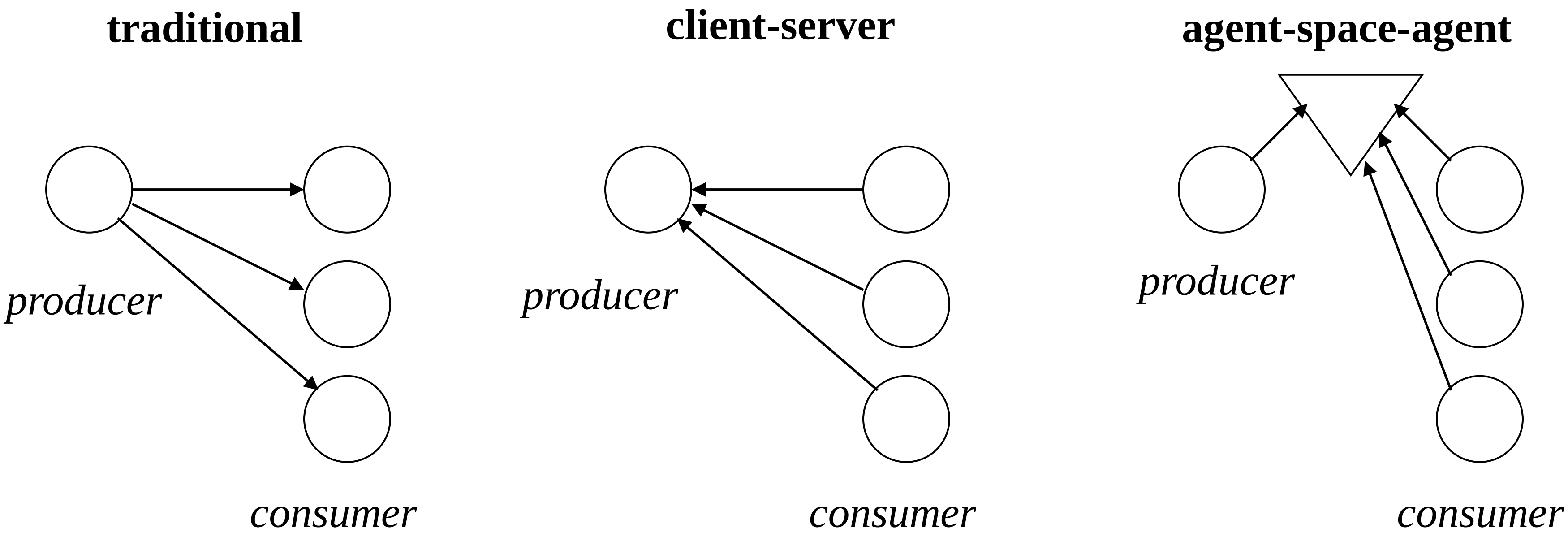
MAS can be treated as a special case of distributed system of the client-server type where:

- server does not contain any application code
- server provides just communication services
- we aim to re-use sever for another application (reusability)
- client is equipped by a library which provide comfort access to the server
- the server + the library = middleware

Transformation from Client-Server to Agent-Space-Agent



Data flows



traditional

producer

consumer

client-server

producer

consumer

agent-space-agent

producer

consumer

Features of Space

- It is server for agents which are its clients
- It is independent from application domain
- It must be reliable and fast (effective algorithms have to used) as it is a bottle-neck)
- It provides services which materialize communication among klients
- It works with a kind of marshalling (which is usually based on a representation language)

Representation and communication language

As a result, we can (for indirect communication) make definition of representation and communication more exact:

- Only agent knows representation language, it is not a part of middleware, it codes that part of data which space does not unpack and/or touch
- Both space and agent library knows communication language, it is a necessary part of middle ware

Further services of the indirect communication

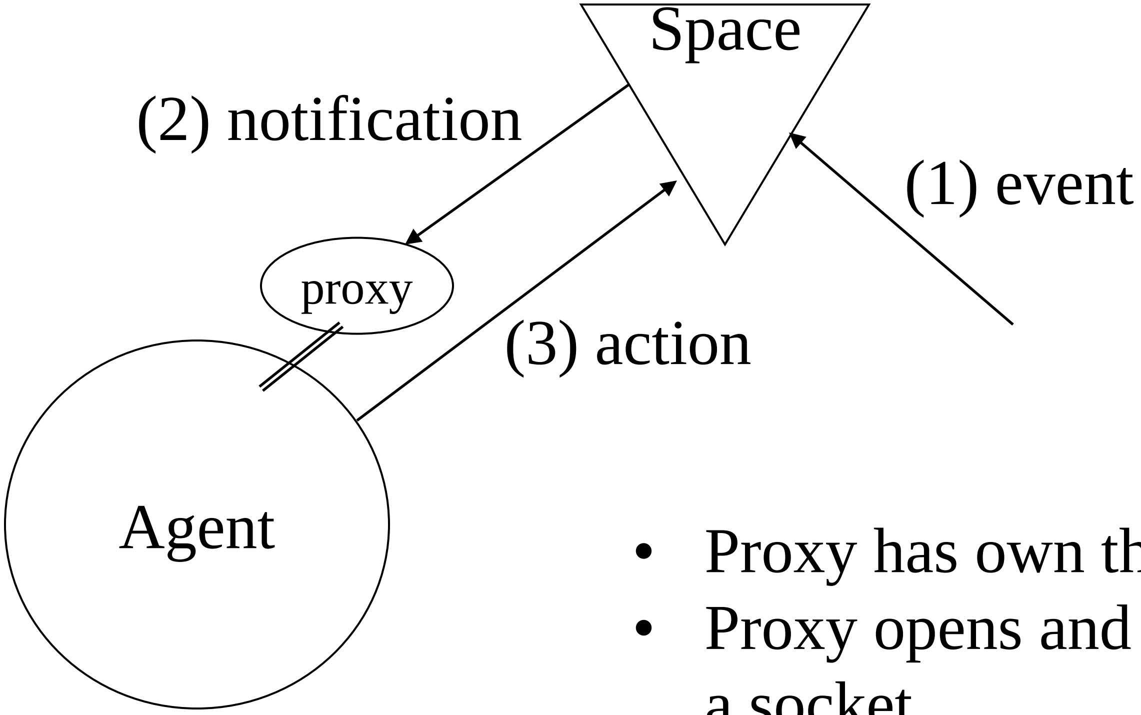
- Further services:

  - trigger registration (notification)

  - mass operations over the data in the Space, e.g. based on mask

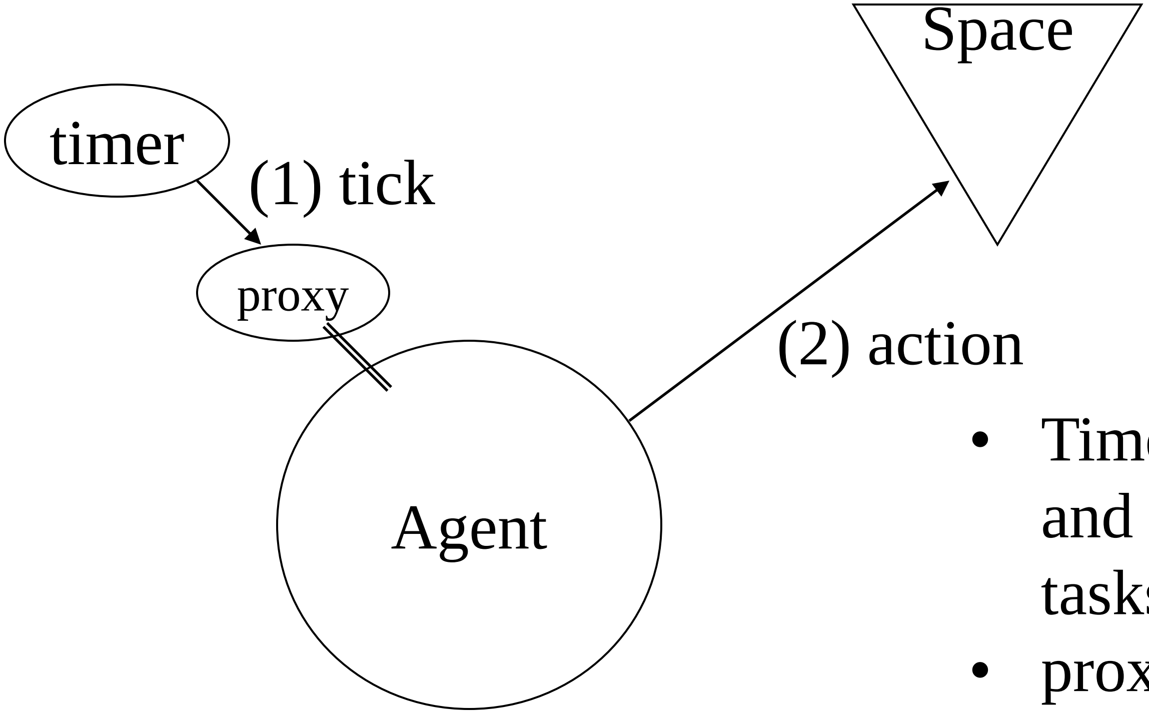
- Synchronization: each service is perfomed without interruption by another one, agent can perform a list of the services without interruption

Trigger



- Proxy has own thread
- Proxy opens and keeps alive a socket
- Notification is implemented as a delayed answer

Timer



- Timer has an own thread and a queue of timered tasks, sorted by time
- proxy is a timered task managed by the timer
- Proxy is unblocking the agent thread

Reference of the stored data

- UUID and so
- name
- name unification
- data unification (in representation language)

Implementation

- history: all implementation are comming from the LINDA programming language (1985, for parallel programming)
- no standards
- proprietary solutions

LINDA Tupple Space

Data structure containing tuples of terms in form of LISP lists equipped by

- **out(t)** writes a new tuple
- **in(t)** read and remove certain tuple; if such a tuple is not available, the reading process is blocked until it occurs
- **rd(t)** does the same as in(t), just it does not remove the tuple
- **inp(t)** return TRUE and remove certain tuple if it is available; it returns FALSE otherwise
- **rdp(t)** does the same as inp(t), just it does not remove the tuple

Specification of the read tuple is based on data unification

Java Space

- part of Java Jini package, which was develop to change networks of computers and services to network of services and things, its main part is Java Lookup Service
- It is a middleware built over RMI

Java Space

```
package net.jini.space;
import java.rmi.*
public interface JavaSpace {
  Lease write(Entry entry, Transaction txn, long lease);
  Entry read(Entry tmpl, Transaction txn, long timeout);
  Entry readIfExists(Entry tmpl, Transaction txn, long
            timeout);
  Entry take(Entry tmpl, Transaction txn, long timeout);
  Entry takeIfExists(Entry tmpl, Transaction txn, long
            timeout);
  EventRegistration notify(Entry tmpl, Transaction txn,
            RemoteEventListener ln, long lease,
            MarshalledObject handback);
  Entry snapshot(Entry e);
}
//throws clauses not shown
```

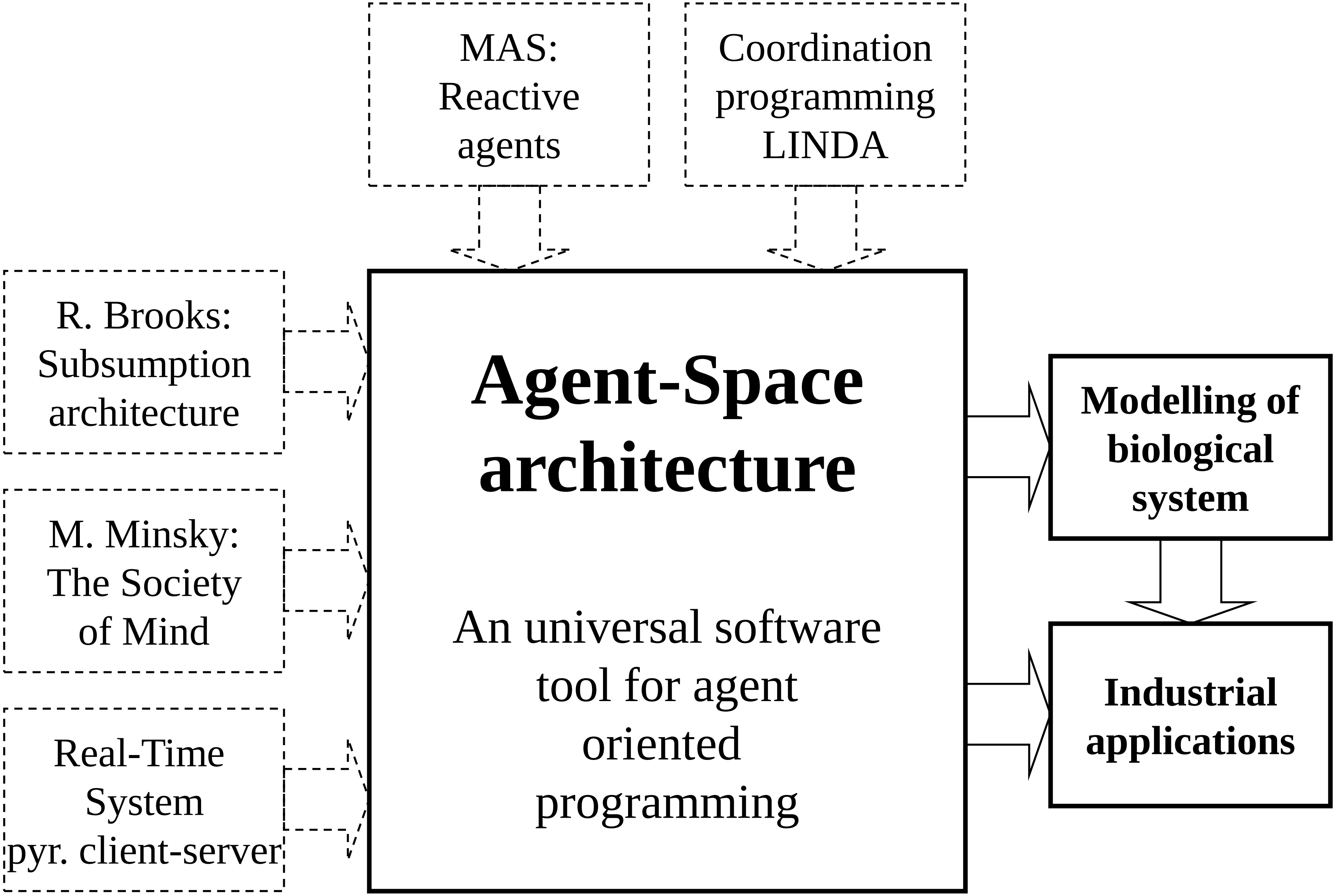
Data leasing
– time validity

- Java Space introduced limited time validity of the data stored in Space
- Space is taking care of the leased data removal when they expires after some period.
- The removal can be implemented passively, but it can be more accurate if it is treated by a timerred task or a dedicated thread.

Agent-Space

- Multi-agent architecture developed at FMFI UK Bratislava in 1997-2004
- It is an expression of traditional ideas of Brooks and Minsky by a new language (MAS with indirect communication)

Jozef Kelemen



MAS:
Reactive
agents

Coordination
programming
LINDA

R. Brooks:
Subsumption
architecture

M. Minsky:
The Society
of Mind

Real-Time
System
pyr. client-server

Agent-Space architecture

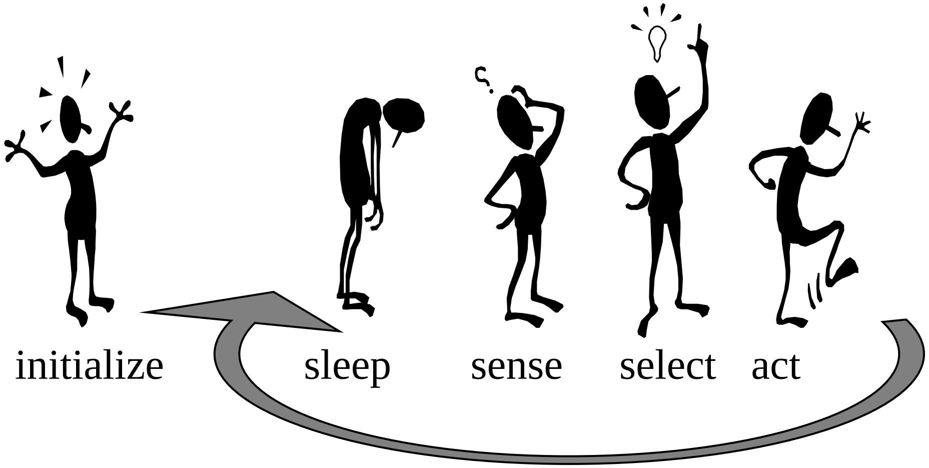
An universal software tool for agent oriented programming

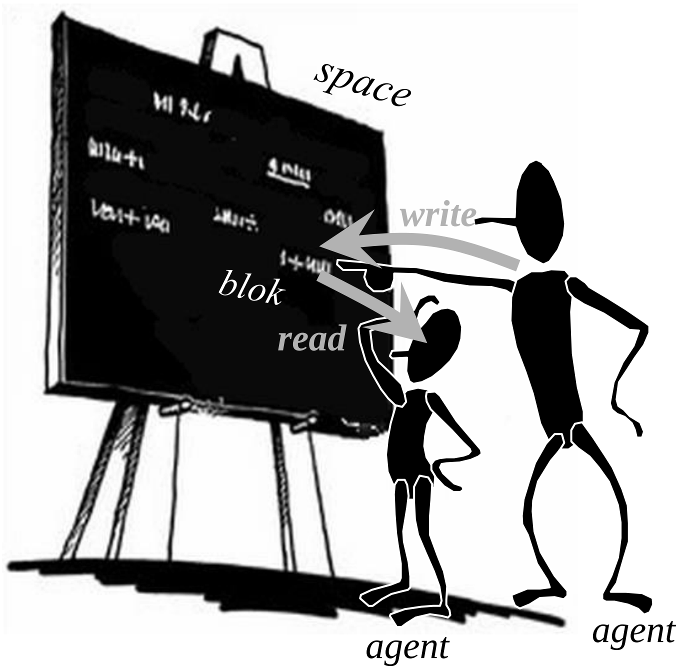
Modelling of biological system

Industrial applications

Architecture Agent-Space

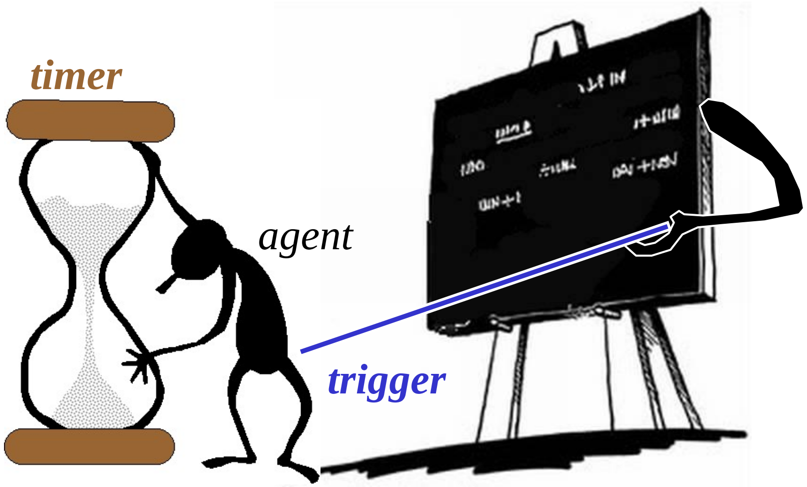
- System consists of agents
- Agents communicate through Space





Implementation in C++/Java

- Each agent is object with own thread
- It calls *read* and *write* methods of singleton object *Space*
- Agent is regularly waken up by timer or trigger (by the *write* operation performed by another agent)



Code example

```
#include <iostream>
#include <conio.h>
#include "agentspace.h"
using namespace std;

class MyAgent1 : public Agent {
  private:
    int i;
  protected:

    void init (string args) {
      i = 0;
      timer_attach(1000,1000);
    }

    void sense_select_act (int pid) {
      i++;
      cout << "a := " << i << endl;
      space_write("a",i,1500);
    }

  public:
    MyAgent1 (string args) :
      Agent(args) {};
};
```

```
class MyAgent2 : public Agent {
  protected:

    void init (string args) {
    trigger_attach("*",TRIGGER_MATCHING);
    }

    void sense_select_act (int pid) {
      int a = space_read("a",0);
      cout << "a = " << it->value << endl;
    }

  public:
    MyAgent2 (string args) :
      Agent(args) {};
};

int main () {
        MyAgent1 a1("");
        MyAgent2 a2("");
        getch();
}
```

Code example

```
package org.agentspace.demo;
import org.agentspace.*;

public class Agent1 extends Agent {

  int i = 0;

  public void init(String[] args) {
    attachTimer(1000);
  }

  public void senseSelectAct() {
    System.out.println("write:  "+i);
    write("a",i++);
  }

}
```

```
public class Agent2 extends Agent {

  int i;

  public void init(String args[]) {
    attachTrigger("a");
  }

  public void senseSelectAct() {
    i =  (Integer) read("a",-1);
    System.out.println("read "+i);
  }

}
```

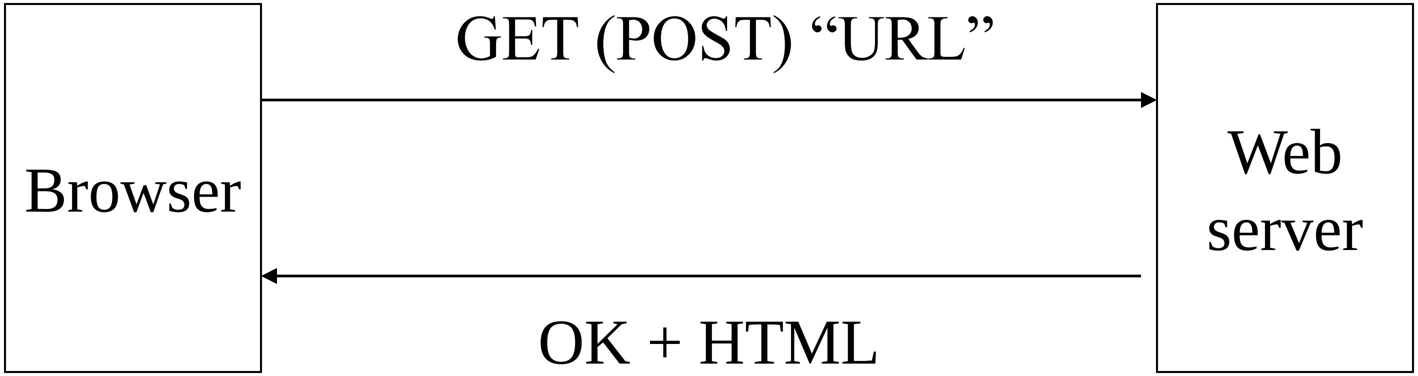
```
public class Starter {
  public static void main(String[] args) {
    new SchdProcess("space",“org.agentspace.SpaceFactory",new String[]{"DATA"});
    new SchdProcess("agent1",“org.agentspace.demo.Agent1",new String[]{});
    new SchdProcess("agent2",“org.agentspace.demo.Agent2", new String[]{});
  }
}
```

Implementation of
the Space services
in distributed environments

- via RMI
- via web services
- ...

Web services

- Extension of the HTTP protocol



GET (POST) “URL”

Browser

Web server

OK + HTML

```
GET /info/index.html HTTP/1.1
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/vnd.ms-excel,
application/vnd.ms-powerpoint, application/msword, application/x-shockwave-flash, */*
Referer: http://www.swim.sk
Accept-Language: sk,en-us;q=0.5
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0)
Host: www.swim.sk
Connection: Keep-Alive
Cache-Control: no-cache
```

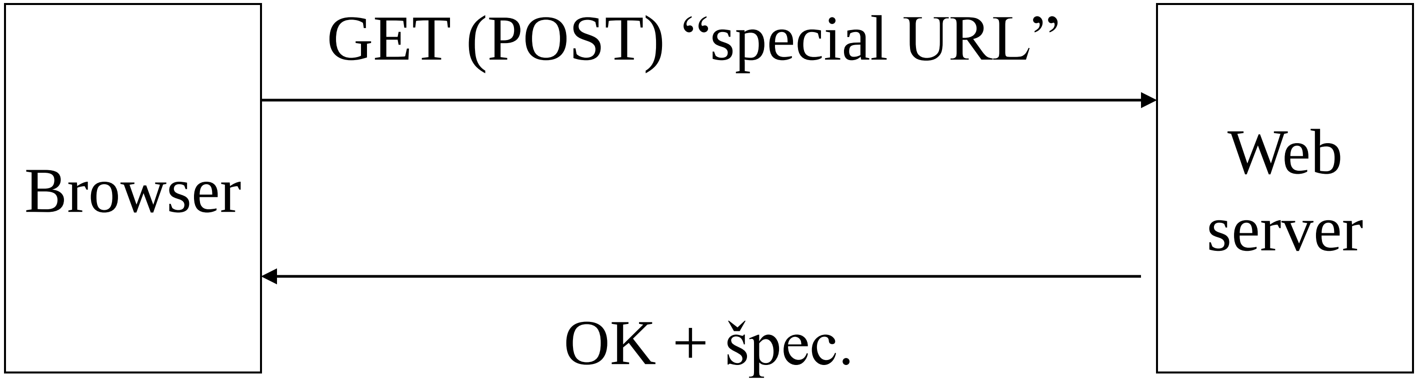
```
HTTP/1.1 200 OK
Date: Sun, 11 Sep 2005 11:09:03 GMT
Server: Apache/2.0.54 (Debian GNU/Linux) mod_python/3.1.3 Python/2.3.5 PHP/4.3.10-16
mod_ssl/2.0.54 OpenSSL/0.9.7e mod_perl/1.999.21 Perl/v5.8.4
X-Powered-By: PHP/4.3.10-16
Content-Length: 2178
Connection: close
Content-Type: text/html

<html>
<meta http-equiv='Content-Type' content='text/html; charset=windows-1250'>
<body> ahoj </body>
</html>
```

HTTP

Web services

?param1=value1&param2=value2 pre GET
alebo napr. XML pre POST



for instance:
XMLHttpRequest

for instance:
servlet