Advantages of Multi-agent Approach to Building of Monitoring Systems

Andrej Lúčny

DAI FMFI, Comenius University, Bratislava

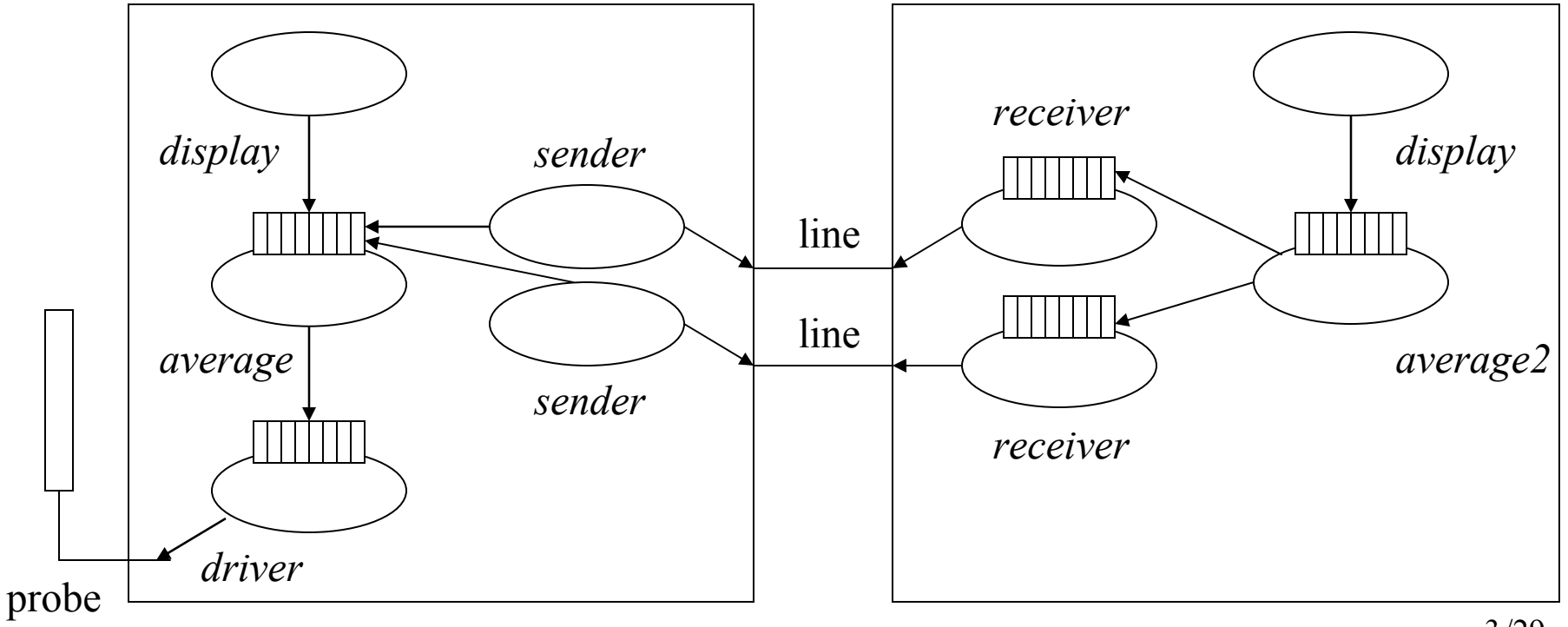
andy@microstep-mis.com

http://www.microstep-mis.com/~andy

- Monitoring system example
- Traditional approach
- Multi-agent approach
- Agent-Space architecture
- Features of indirect communication
- Soft crash landing
- Modification by subsumption

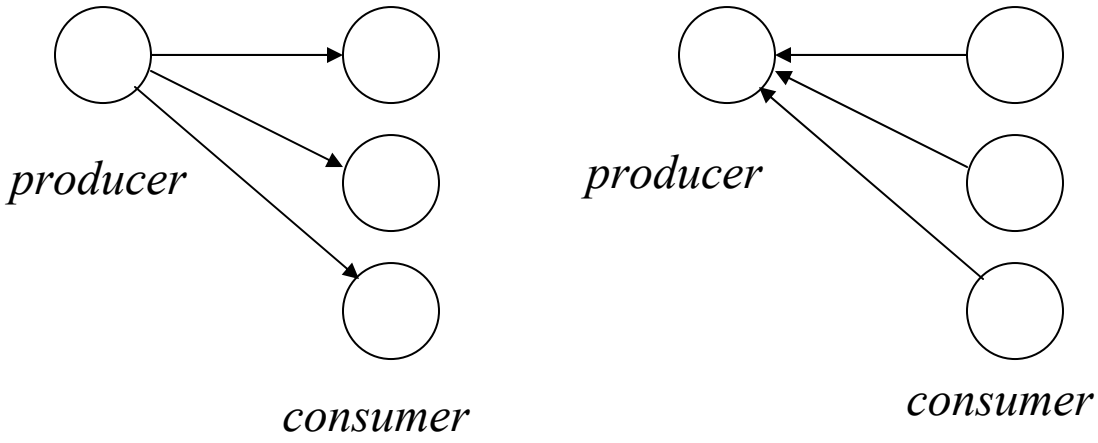
Monitoring system example

- interactive operation
- real-time operation



Traditional approach

- A producer calls several consumers and sends them a value
- Several consumers call a producer and ask it to provide the value as a response



Alternative solution

- Neither producer calls consumer, nor consumer calls producer

  But:

- Producer produces value, consumer consumes value

  Because:

- Just the value is important for both, their mutual relationship can be eliminated

Multi-agent approach

- The best way how to implement this idea is application of multi-agent modularity
- Under the modularity, producers and consumers turn to agents and values to (indirect) communication among them
- We will organize internal structure of monitoring system in similar way as typical for distributed and decentralized systems

Multi-agent system example

- Typical example: robot-soccer

Multi-agent system

- It would be too difficult to write a program which controls all the players.
- It is much easier to code programs for individual players and let the team control to emerge from their interaction
- Such interacting programs are called agents

Decentralization

Such solution is decentralized:

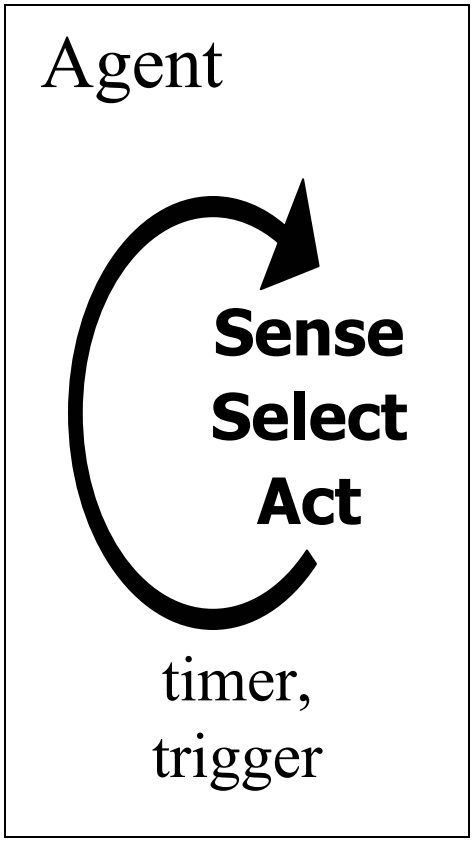
- for example, if we remove midfielder from the team, striker does not stop (endlessly waiting for a pass from the removed midfielder), just probably scores a goal less frequently. It is because striker do not need a midfielder, in fact it needs the ball
- Even if the striker never got a pass, he is still moving and ready to shoot ball whenever it is available to him.

Nature of agents

- agents can be implemented as objects equipped with an own thread of control and a mechanism of a mutual data exchange including sensation and action of the system environment

Nature of agents

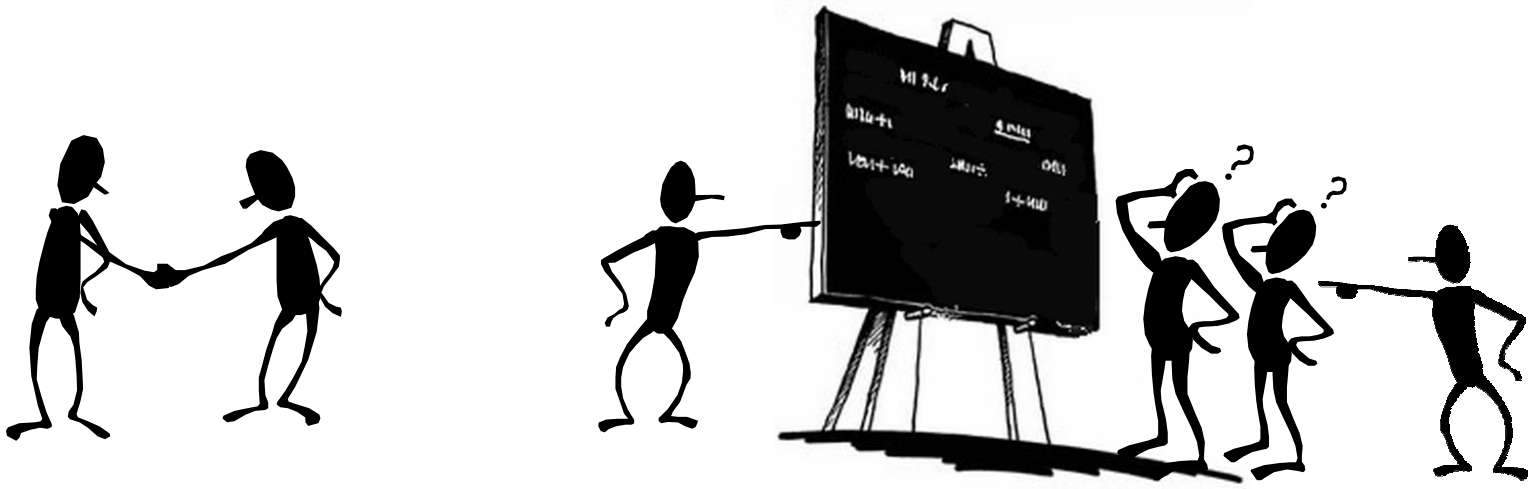
- each agent is endlessly running a sense-select-act cycle. Any course through this cycle calculates some actions upon information sensed from environment or provided by other agents



Communication among agents

The communication mechanism can be based on

- direct message passing
- **indirect communication through a more or less sophisticated blackboard (called also space)**



Back to architecture of monitoring system

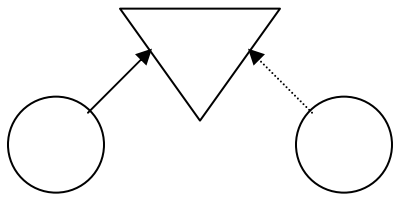
- Can we organize internal modules of one system in similar way as cooperating players in the team ?
- Can we use multi-agent modularity for building of monitoring system ?

Yes, we can

Agent – Space architecture

We transform:

- producers and consumers to agents
- calls among them to indirect communication via blocks in space (on blackboard)



producer *consumer*

Indirect communication

Agents:

- can read, write or delete particular blocks in space
- know nothing about other agents, just know names and structure of the blocks they manipulates with
- perform their code on timer and/or trigger (a change of selected blocks in space)

Indirect communication

Details of read, write and delete operations are:

- no method for block creation
- reading of non-existing blocks is handled by returning a default value specified by reader
- value stored in block can have a limited time validity specified by writer; after its expiration the block becomes automatically empty
- value stored in block can have a priority specified by writer; such value overwritten only by value with same or higher priority
- space has no knowledge about value meaning; the reader is responsible for correct interpretation

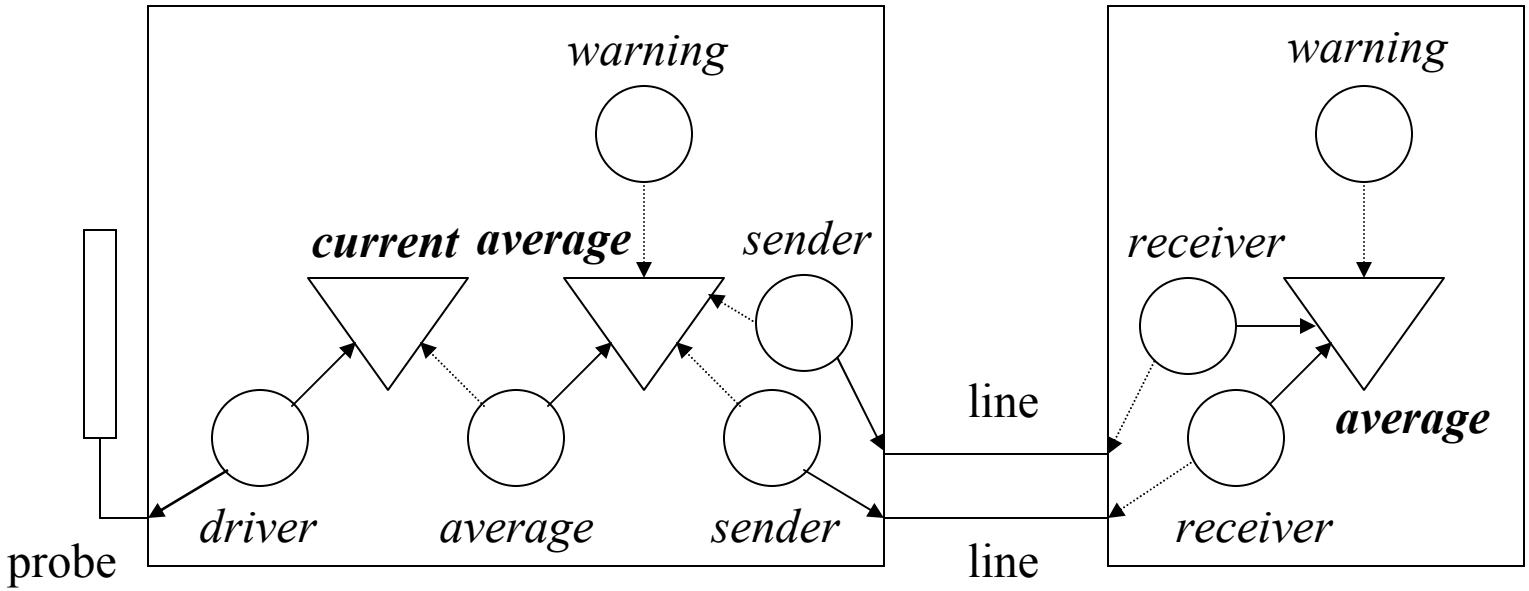
Code example

```
package com.microstepmis.agentspace.demo;
import com.microstepmis.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);
  }

}
```

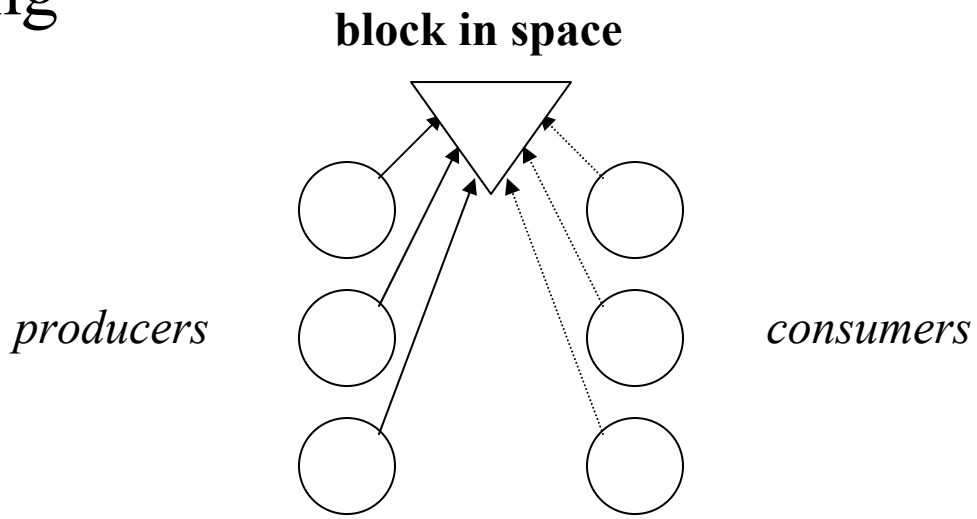
```
public class Starter {
  public static void main(String[] args) {
    new SchdProcess("space","com.microstepmis.agentspace.SpaceFactory",new String[]{"DATA"});
    new SchdProcess("agent1","com.microstepmis.agentspace.demo.Agent1",new String[]{});
    new SchdProcess("agent2","com.microstepmis.agentspace.demo.Agent2",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

Monitoring system example



Data flow many:many

- each block can be written by many producers and read by many consumers
- consumers do not know how much producers generates the value or from whom the read value is coming

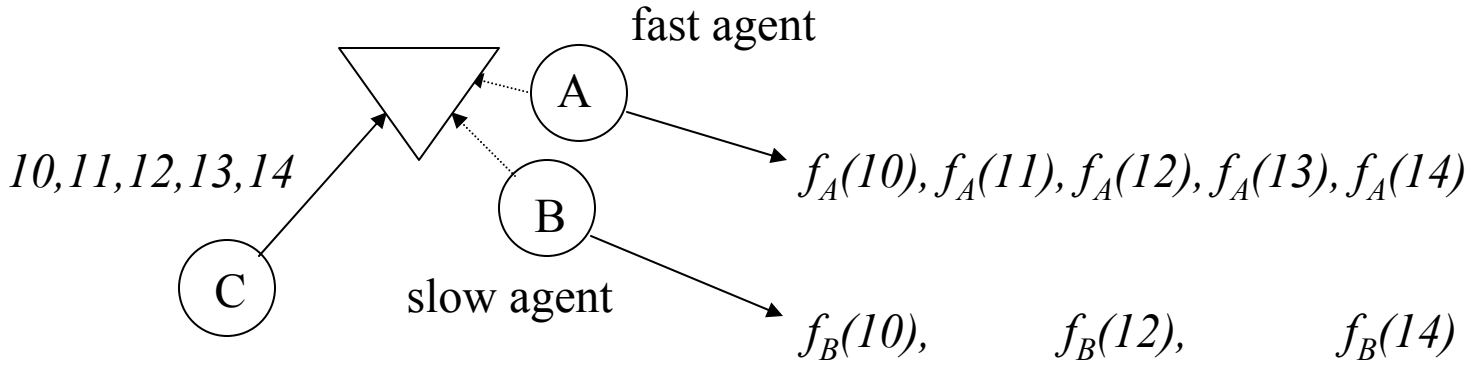


Time validity

- When we have more producers, neither of them can write “bad values”
- Rather such producer does not write a value at all
- But then it can happen that an old value persists in space and it is taken by consumers as valid
- Ideal solution is to define time validity for any written value. After its expiration, the value disappears from space (without agent intervention)
- Thus it can happen that block is empty, so consumer have to handle this state. Ideal solution is to use a default value specified when consumer calls read operation

Implicit sampling

- Since write operation overwrites data stored in a block regardless their consumers have undertaken them or not, any data flow is inherently (potentially) sampled.



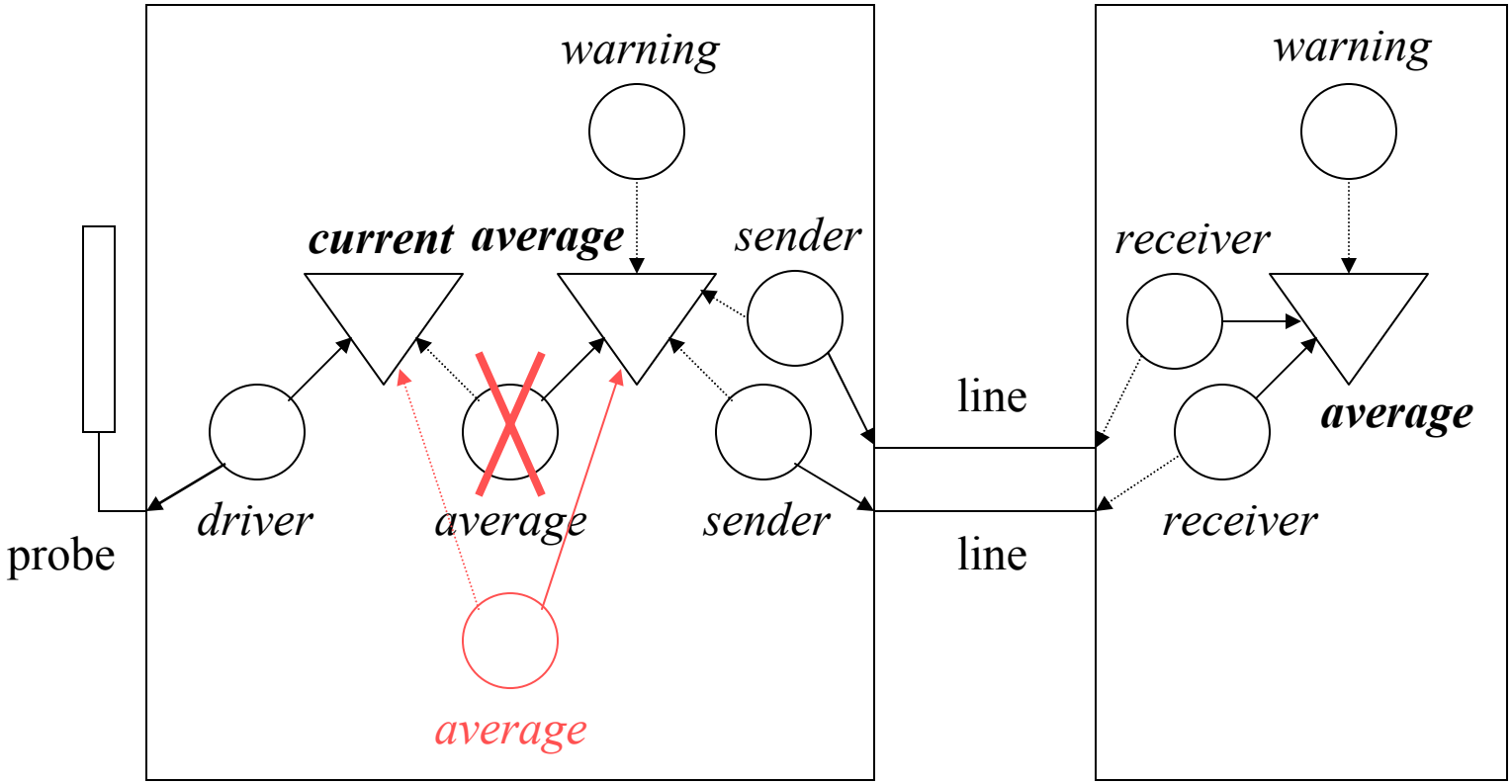
Advantages of Agent-Space architecture

- Reliability, configurability and soft crash landing
- Ability to be modified (incremental development)

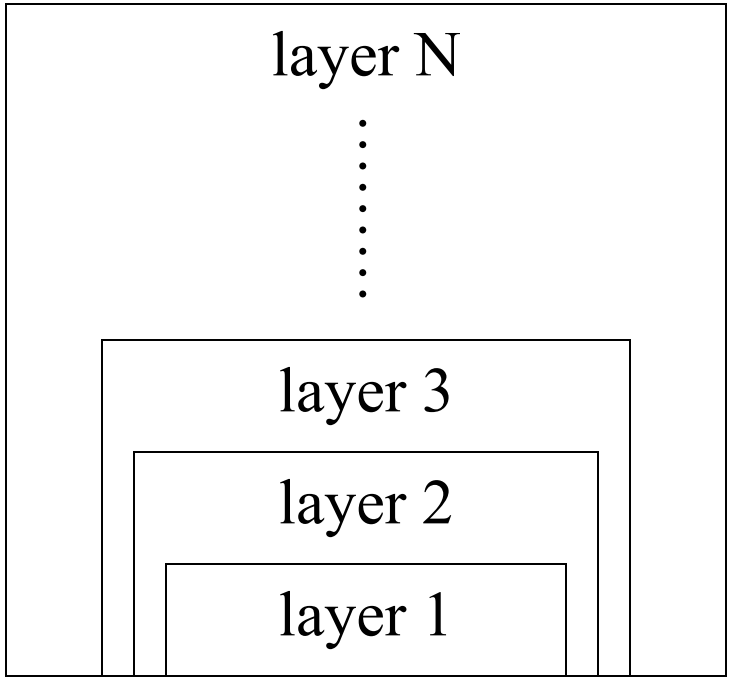
Soft crash landing

- each agent can be restarted without impact on system operation, mainly if they have no inner state (rather they can have analogical information in space)
- thus we can easily to add subsystem which starts crashed agents again and thus provide recovery from errors
- (each application specific code is concentrated in agents, space is independent from application domain)

Soft crash landing



Modification by subsumption



- a design principle of control which mimics simplified biological evolution
- any complex control has an origin in a simpler ancestor
- descendant mechanism subsumes the mechanism of its ancestor
- higher levels rather inhibit and regulate than active the lower levels

Subsumption

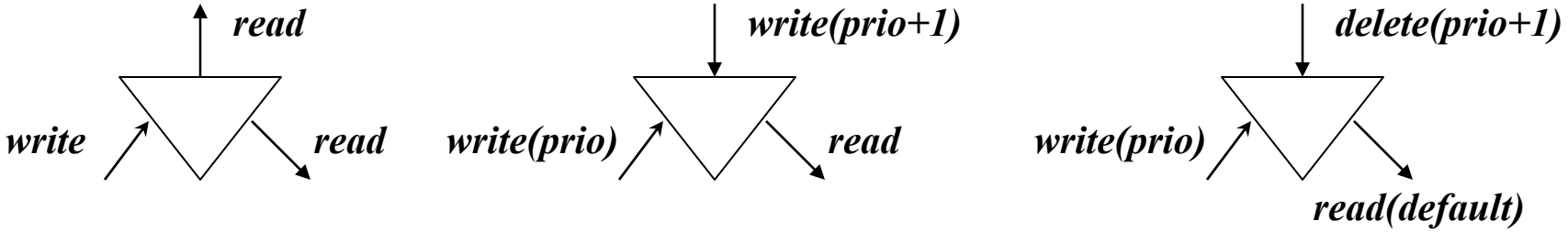
Question: How could the newer levels influence the older ones? The older levels have been designed for particular use and have no interfaces for future development!

Answer: they have to have modular structure which enables it !

Solution: concept of indirect communication is suitable to provide that

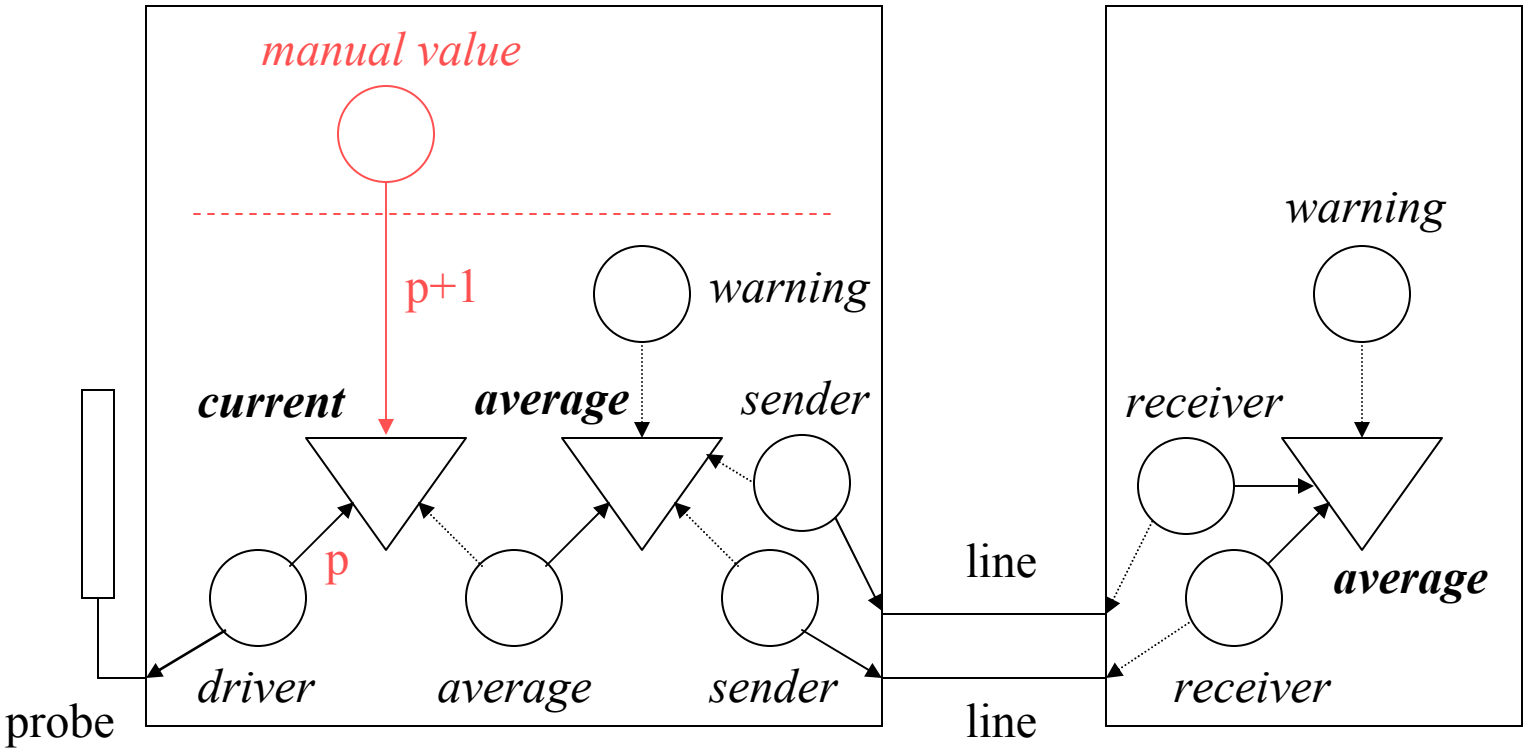
Priorities

- However, blocks need to be associated also with priorities



monitoring suppression inhibition

A simple example of subsumption



Thank you for attention!

Advantages of Multi-agent Approach to Building of Monitoring Systems

Andrej Lúčny
DAI FMFI UK, Comenius University, Bratislava
andy@microstep-mis.com
http://www.microstep-mis.com/~andy