Introduction to Robotics for cognitive science

Dr. Andrej Lúčny

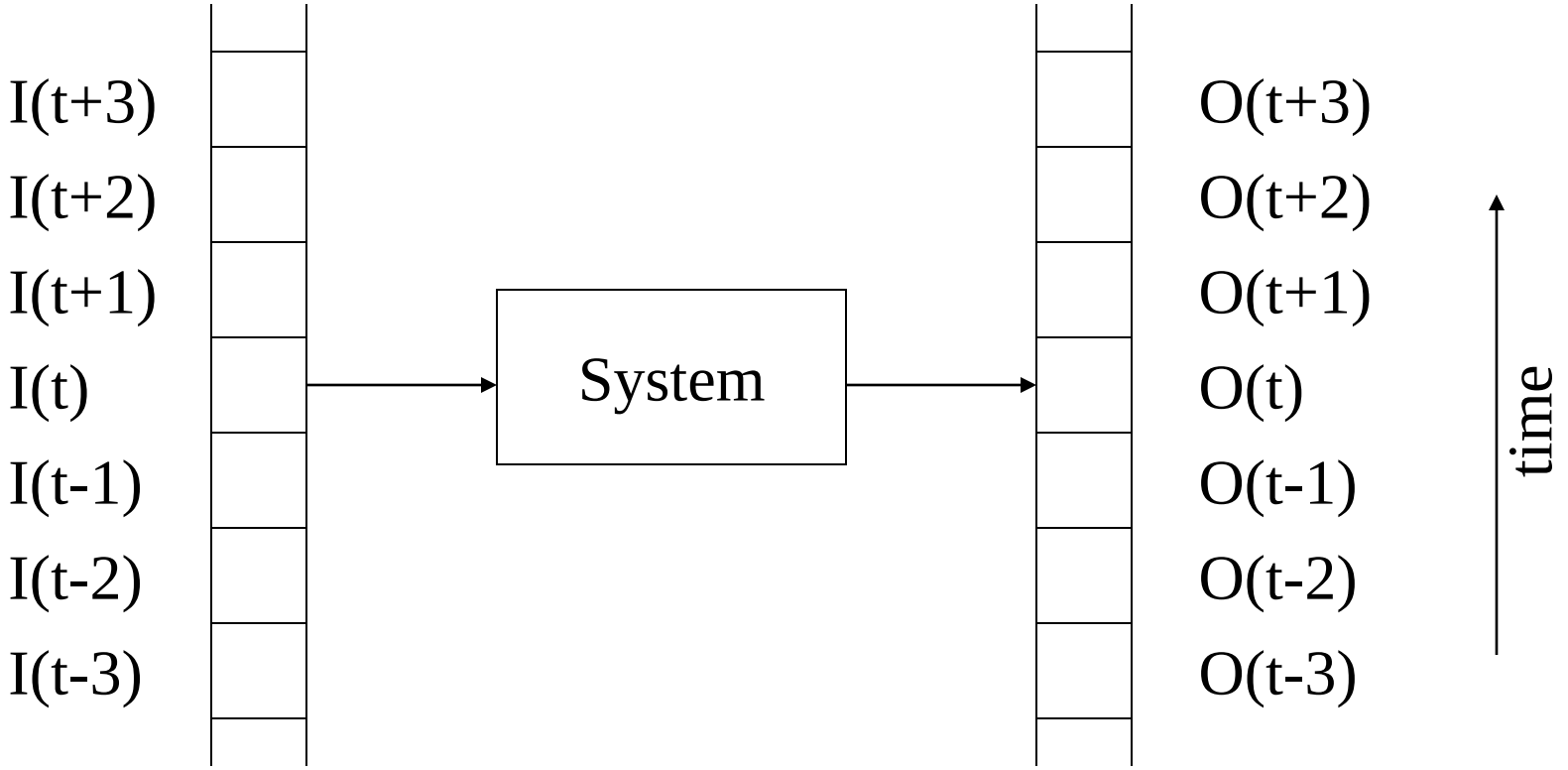
KAI FMFI UK

lucny@fmph.uniba.sk

Web page of the subject

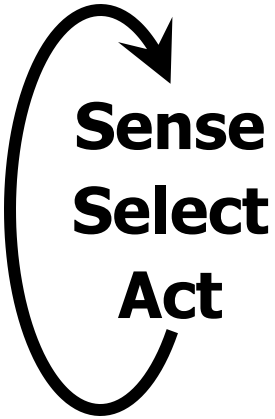
www.agentspace.org/kv

Control


I(t+3)
I(t+2)
I(t+1)
I(t)
I(t-1)
I(t-2)
I(t-3)
System
O(t+3)
O(t+2)
O(t+1)
O(t)
O(t-1)
O(t-2)
O(t-3)
time

Monolytic solution

- Single loop: read data from sensors, calculate actions, perform actions



Problems

- Code becomes complicated where more and more features are implemented
- Slow processes slow down fast processes, as a result the system is not working in real-time

Real-time

- Operation in real-time means that if an event should be performed in time t, it is sure it will be performed in time $t+L$, not later

- L is latency and it is expressed in ms

Real time

```
import time

t0 = time.time()

time.sleep(3)

t1 = time.time()

print(t1-t0)
```

Modular architectures

We need to decompose the control somehow

Technical means: e.g., threads

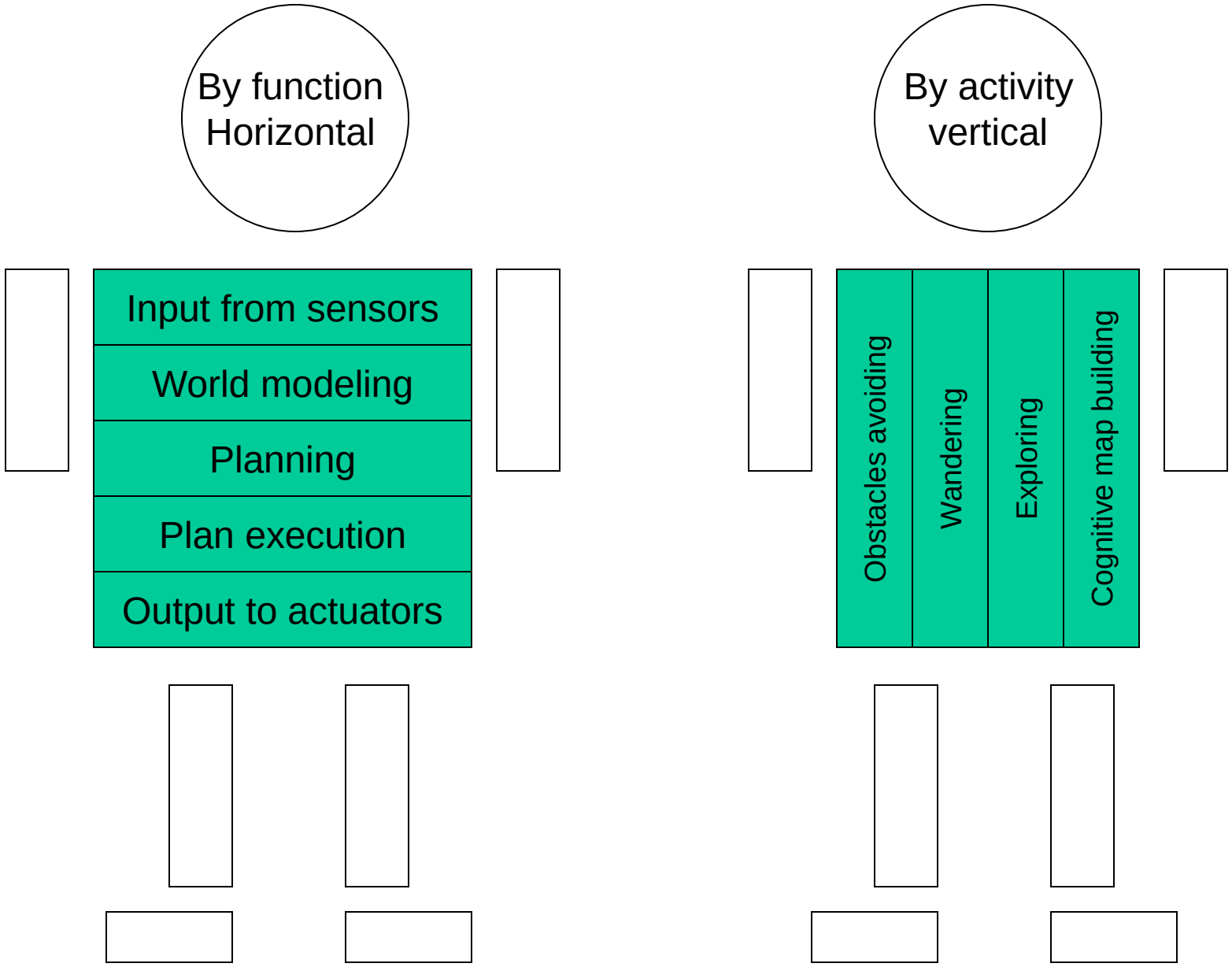
Threads

- We can start more program counters in parallel:

```
import threading

def process1():
    time.sleep(2.0)

def process2():
    time.sleep(3.0)

threading.Thread(target=process1).start()
threading.Thread(target=process2).start()
```

Decomposition

- by function = layers contain codes providing similar function (e.g. vision)
- by activity = layers contain codes providing similar activities (e.g. obstacles avoiding)


By function
Horizontal
Input from sensors
World modeling
Planning
Plan execution
Output to actuators
By activity
vertical
Obstacles avoiding
Wandering
Exploring
Cognitive map building

Modular architectures

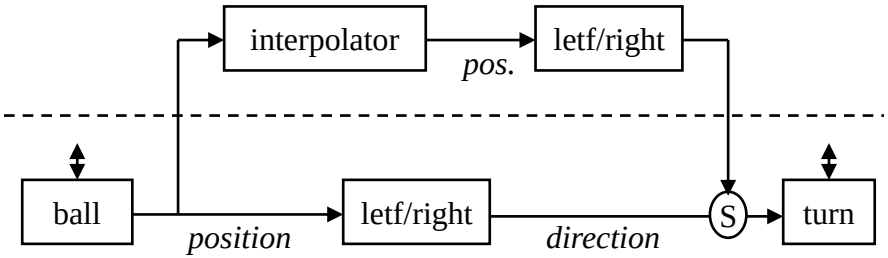
Historical architectures:

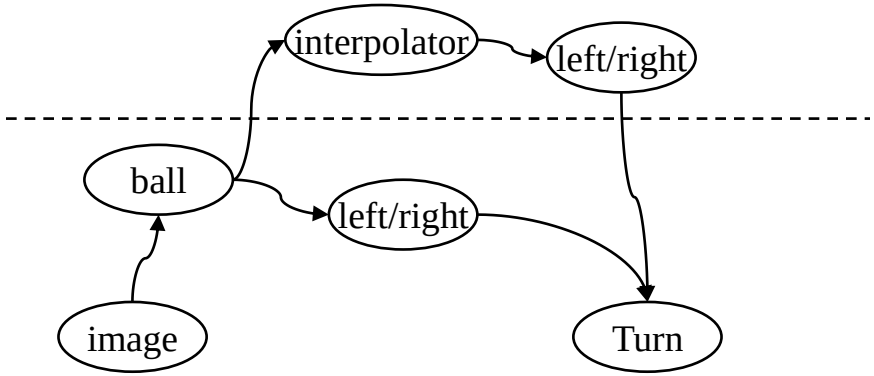
- Brooks' subsumption architecture
- Minsky's "society of mind"
- Payton & Rosenblat: Fine grained architecture
- RTOS architectures (pyramidal client-server)
- Arkin's Behavioral architecture
- …

Modular architectures

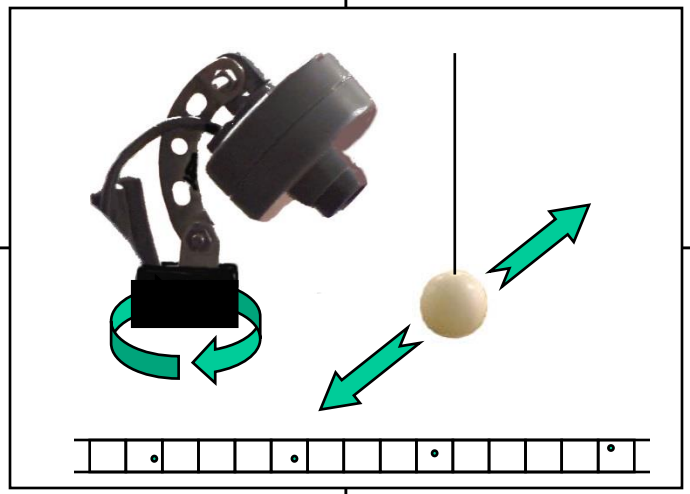
Implementations:

- Robot Operating System (ROS)
- Agent-Space architecture



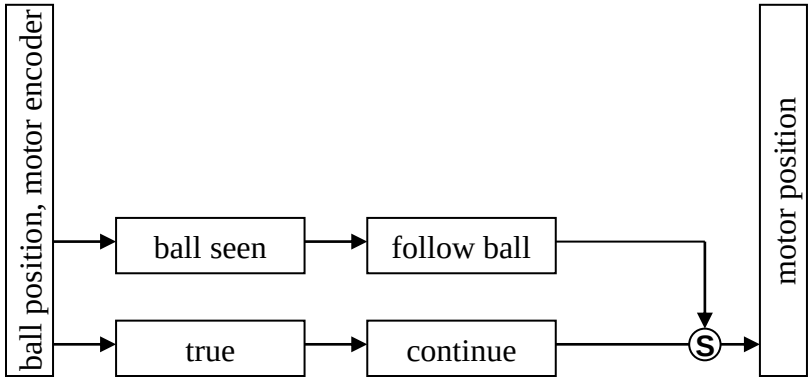


subsumption architecture [Brooks]

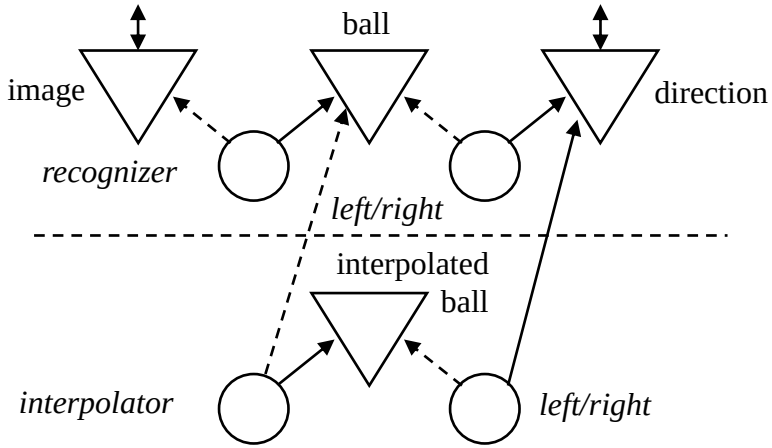


Society of mind [Minsky]

behavioral architecture [Arkin]

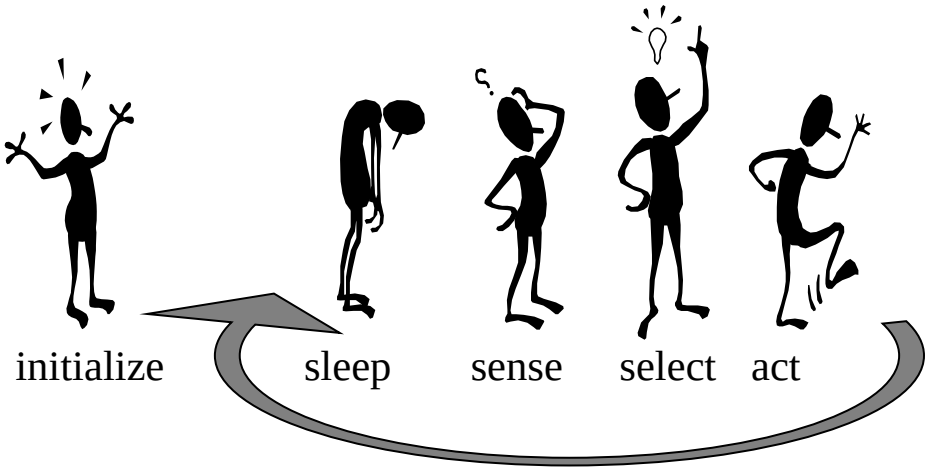


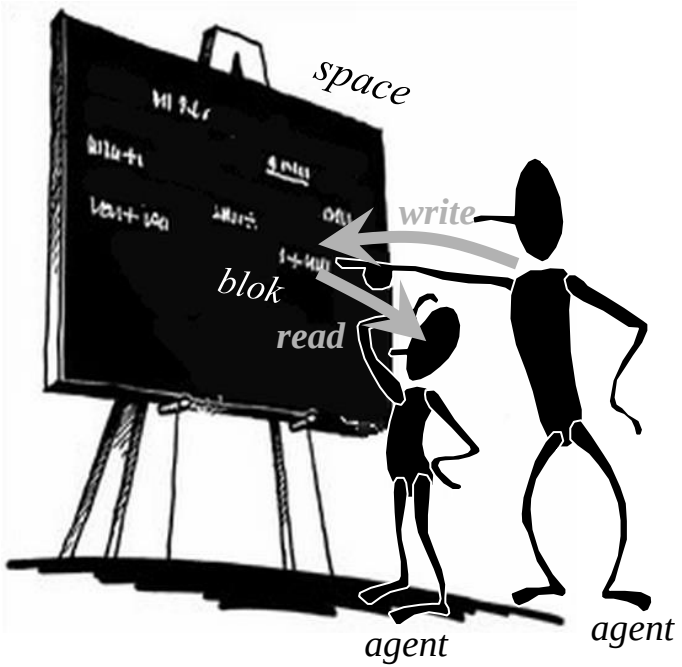
agent-space architecture [Lucny - Kelemen]



Architecture Agent-Space

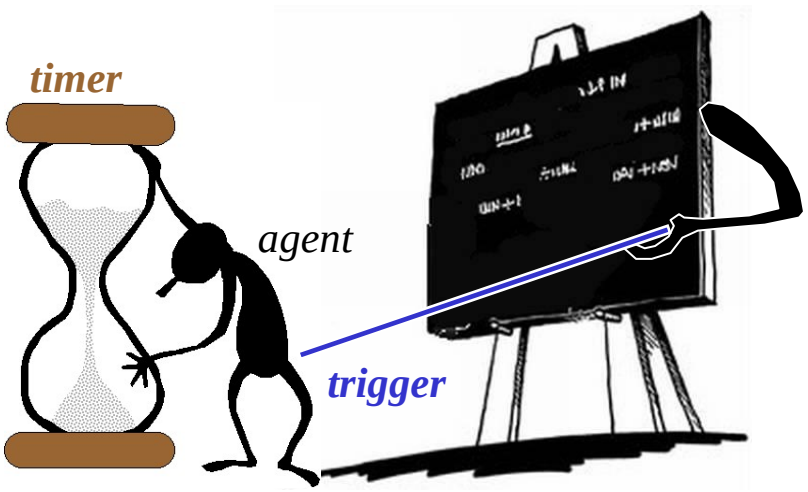
- System consists of agents
- Agents communicate through Space





Implementation in Python

- 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)



Implementation in Python

Blackboard (extended dictionary)

```
from agentspace import space

space['a'] = 2
print(space['a']) # 2

print(space['b']) # None
print(space(default=-1)['b']) # -1
```

Implementation in Python

validity and priority

```
space(validity=0.5)['b'] = 1
print(space['b']) # 1
time.sleep(1)
print(space(default=-1)['b']) # -1

space['c'] = 1 # default priority is 1.0
space(validity=0.5,priority=2.0)['c'] = 2
space(validity=2.0,priority=1.5)['c'] = 3
print(space(default=0)['c']) # 2
time.sleep(1)
print(space(default=0)['c']) # 0
```

Implementation in Python

```
from agentspace import Agent

class MyAgent(Agent):

    def __init__(self, args):
        process(args)
        super().__init__()

    def init(self):
        self.attach_timer(1.0)

    def senseSelectAct(self):
        print('hallo')

MyAgent()
```

agents (objects that have own thread)

invoked by timer

Implementation in Python

```
from agentspace import Agent

class MyAgent(Agent):

    def __init__(self, args):
        process(args)
        super().__init__()

    def init(self):
        space.attach_trigger('a',self)

    def senseSelectAct(self):
        print('hallo')

MyAgent()
```

agents (objects that have own thread)

invoked by trigger