Requirements

Basic idea of concurrent systems

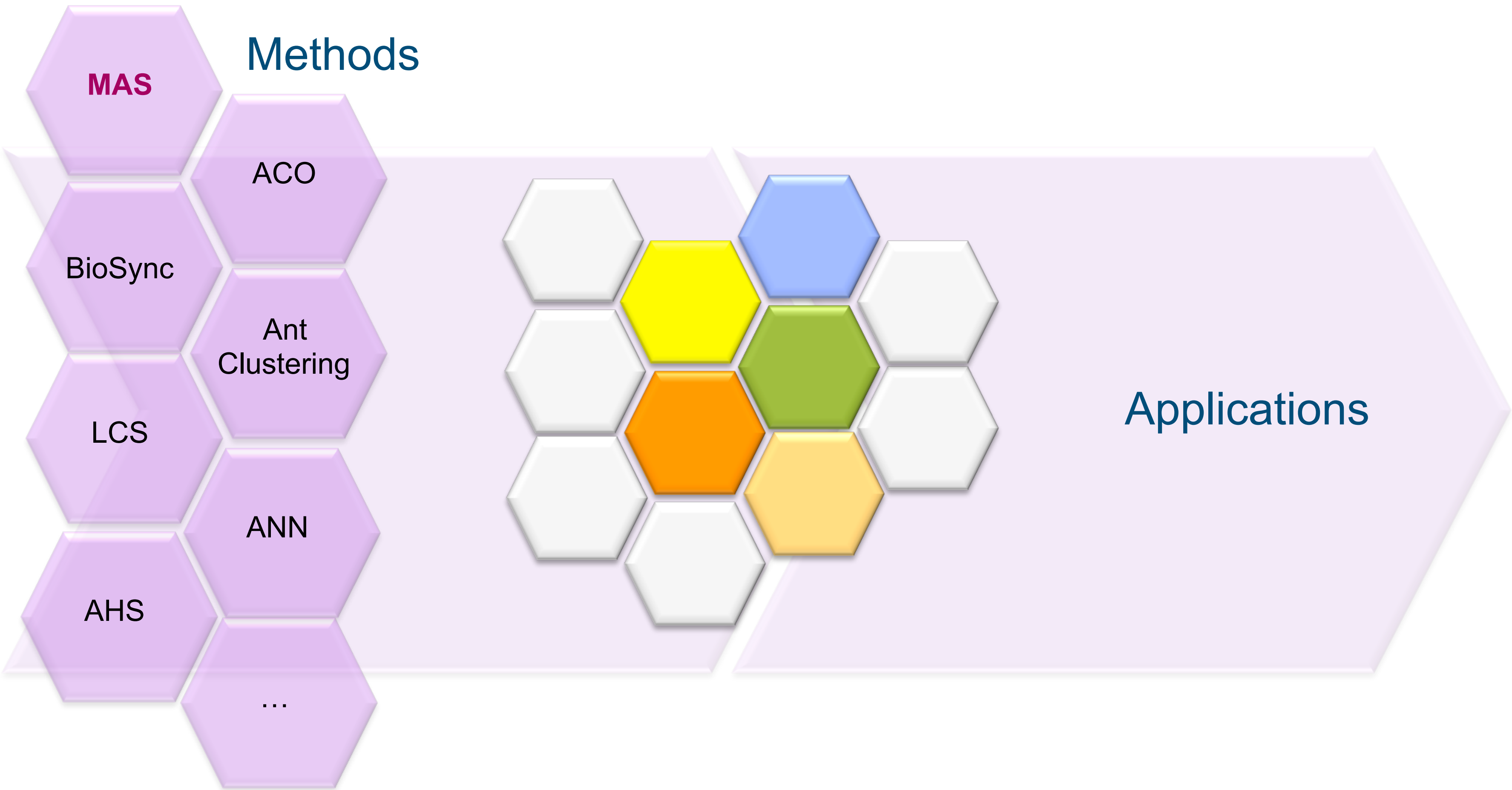
Goals

Understand how to model a concurrent system of multiple independent entities.

Content

- ❑ Examples of agents
- ❑ System environment
- ❑ Agent types
- ❑ Multi-Agent Systems
- ❑ Agreement in MAS
- ❑ Simulation of MAS

ISE SRA

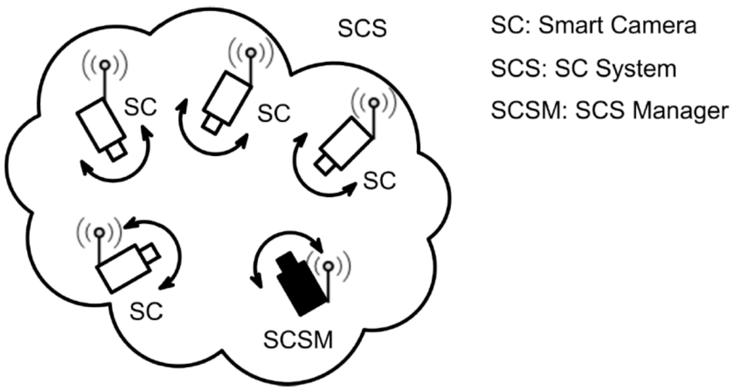

MAS
Methods
ACO
BioSync
Ant Clustering
LCS
ANN
AHS
…
Applications

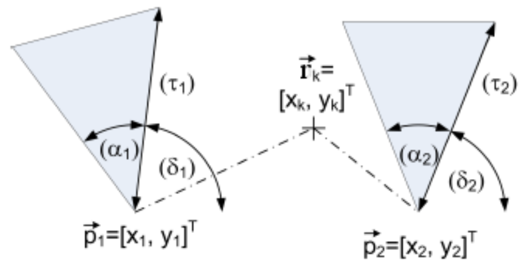
- Organic Computing Systems typically …
  - consist of large numbers of independent entities.
  - follow a system-wide goal (not necessarily known to individual entity).
- Each entity …
  - is context-aware: It can “sense” properties of their (physical or virtual) environment.
  - is able to make changes to its environment.
  - follows an individual goal and makes local decisions.
- Multiple entities …
  - co-exist in the same environment.
  - can act independently and concurrently.
  - sometimes interact with each other.
  - can be cooperative or competing.

- Examples of technical systems
  - Groups of mobile robots
  - Traffic control systems
  - Networked camera systems
  - Desktop Grid Computing
- Questions that arise
  - How can we model such systems?
  - What are the important properties?
  - What are generic problems that often arise? For example coordination and agreement.
- Modeling by agents or embodied agents

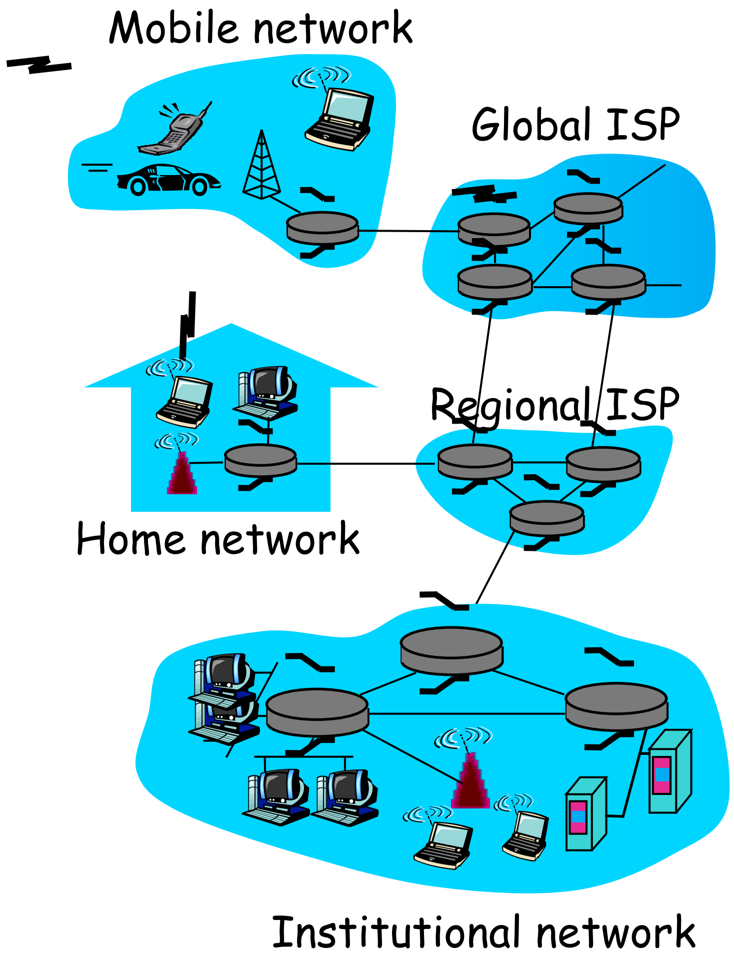
"Entities" = agents

- System consists of multiple networked cameras on a given area.
- Each camera can be modeled as an autonomous agent.
  - Has a partial view of the area (e.g. pattern recognition, object detection).
  - Autonomous actions: changes its field of view
  - Communication with other agents possible
- System goal:
  - Cooperatively observe given area
  - Detection of important events, e.g spatial partitioning, object tracking
  - Communication with users: alarm management





- System consists of interconnected routers.
- Each router can be modeled as an autonomous agent.
  - (In most cases) partial knowledge of the network topology
  - Autonomous action: routing decision
- System goal
  - Transport packets from source to destination
  - Transportation cost may influence forwarding decision → Routers may also compete with each other.



- The term "intelligent agent" refers to a single computer system that …
  - is situated in some environment,
  - is aware of its environment (can measure: sensory input),
  - can autonomously affect the environment (control: action output),
  - (is able to learn and self-optimize)
- Example 1 (cameras):
  - Environment: physical area
  - Awareness: Camera knows its position/heading; can evaluate images
  - Action output: change field of view
- Example 2 (routers):
  - Environment: network topology; data packets
  - Awareness: packet rate, size of queues, latencies
  - Action output: routing decision

Properties of an agent (1)

- What makes an agent intelligent? (Wooldrige and Jennings, 1995)
- Reactivity: An agent reacts to changes in its environment in a timely fashion (according to its design objectives)
  - In example 1: Follow moving object; compensate failure of other cameras
  - In example 2: Forward incoming packet; change local routing strategy when a link fails
- Pro-activeness: Agents exhibit goal-directed behavior to achieve their goals.
  - Not always easy to achieve: In some systems the precondition and/or the goal may change during execution (e.g. due to concurrency or uncertainty).
  - In example 1: “search” for objects
  - In example 2: probe link latencies to chose best route to destination
- Problem in general: Achieve proper balance between reactivity (fast) and pro-activeness (intelligent).

ISE SRA

- "Social ability": Agents can interact with each other.
  - Typically no single agent can achieve the system goal.
  - Agents often need to cooperate or negotiate.
  - Agents may be selfish: They try to maximize their own benefit.
  - In example 1:
    - Perspective of multiple cameras may be necessary.
    - Cameras may avoid cost (energy) of changing their field of view.
  - In example 2:
    - Routers need to cooperatively forward packets.
    - Routers may avoid forwarding cost.

- To define the system precisely, a (little) more formal wording is helpful ☺

ISE SRA

- An Agent architecture consists of defining
  - The environment
  - A perceptual model for the agent
  - The set of actions an agent can take
- Agents can then be divided into several classes
  - Purely reactive (stateless)
  - Stateful
    - follows a plan
    - Model-based reactive agent
    - Goal and utility oriented
  - Learning agents: in order to adapt and improve

- Classification of environments:
  - Accessible versus inaccessible: Whether or not the agent has accurate and up-to-date information about the complete environment (or something in-between)
  - Deterministic versus non-deterministic: Whether or not an agent's action has a predictable effect
  - Static versus dynamic: In a static environment only the agent can modify the environment (multiple agents → dynamic environment).
  - Discrete versus continuous: Discrete environments have a finite (albeit possibly large) number of percepts.
- In most technical systems: Environment can be regarded as discrete and is defined as
  - $E = \{ e_0, e_1, e_2 \ldots \}$ with e_i .. discrete sequential states
  - For example: all cameras' and tracked objects' positions

- The agent's (finite) action set is defined as
  - $Ac = \{ a_0, a_1, \dots \}$ with a_i the available actions (behavioral repertoire)
  - For example: a_1: forward packet to output o_2
- Interaction between agent and environment can be modeled as a run r:
  - $r : e_0 \rightarrow a_0 \rightarrow e_1 \rightarrow a_1 \rightarrow e_2 \rightarrow a_2 \rightarrow \dots \rightarrow a_{u-1} \rightarrow e_u$
  - Where e_0 is the initial state of the environment
  - The set R contains all possible finite runs.
  - For example: sequence of forwarding decisions
- R^{Ac} is the subset of R ending with an action.
- R^E is the subset of R ending with an environment state.

- In non-deterministic environments:
  - Same sequence of actions on e_0 may lead to different runs.
- State transformer function
  - maps a run R^{Ac} to a set of possible environment states.
  - $T(r): R^{Ac} \rightarrow 2^E$ *(power set of E)*

$2^{\{a,b\}} = \{\emptyset, \{a\}, \{b\}, \{a,b\}\}$

- Based on the previous definition, the abstract behavior of an agent can be modeled as a function Ag mapping environment states to actions Ac.
- Purely reactive agent (stateless):
  - Ag: E → Ac
  - Chosen action only depends on current state of the environment.
  - Example: thermostat
- Stateful agent:
  - Ag: R^E → Ac
  - Chosen action Ac depends on history of the system.
  - Example: Navigation system

R^E: Run ending with an environment state

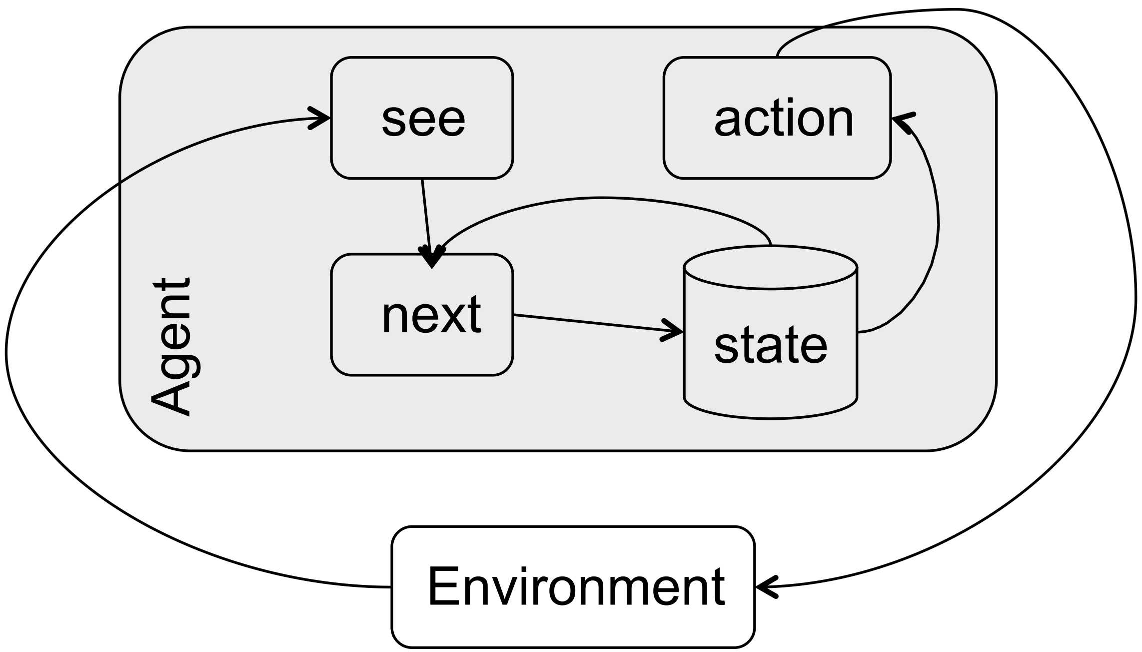
- In many systems agents cannot access the complete state of the environment.
- The limited perception can be modeled as:
  - see: E → *Per*
  - where *Per* is the agent's view of the environment.
- Now the action chosen by an agent is based on a (sequence of) perceptions:
  - action: *Per* → Ac (for a purely reactive agent)
  - action: *Per** → Ac (for a stateful agent)

- So far, the state of an agent is composed of the complete sequence of perceptions and actions $\rightarrow$ may be inconvenient
- Alternative: Explicit internal agent state
  - Goal: smaller number of possible states
  - State transition is derived from current state and perception:
    next: I x *Per* $\rightarrow$ I (where I is the set of possible states)
  - Example: Current field of view of a camera and objects currently in view instead of the sequence of all actions and perceptions
- Then choosing an action is a function that maps an internal state to one of the actions:

  action: I $\rightarrow$ Ac

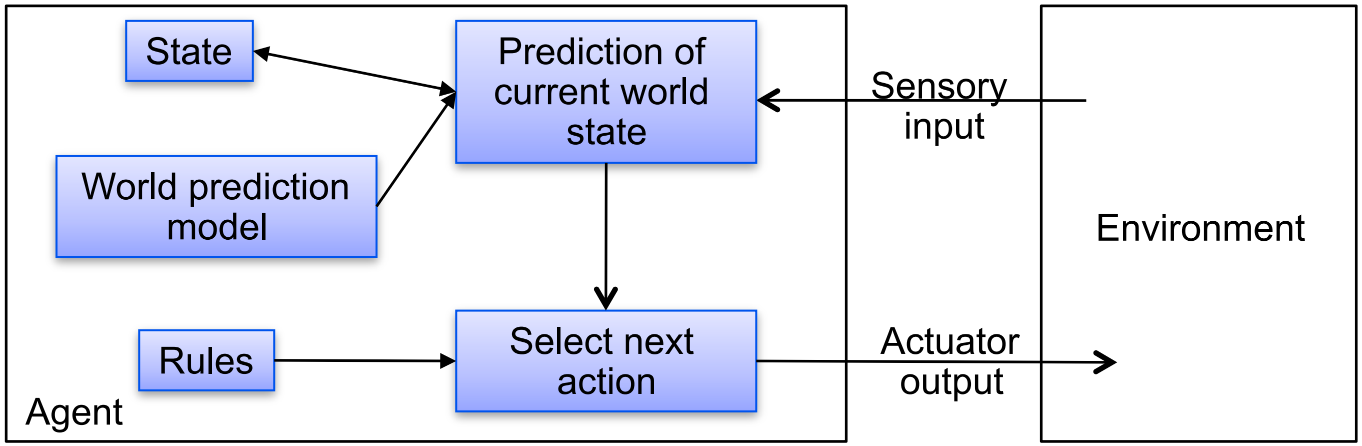
ISE SRA

1. Agent starts in some initial internal state $i \leftarrow i_0$.
2. Agent observes its environment state e, and generates a percept $see\ (e)$.
3. Internal state of the agent is then updated via $next$ function, becoming $i = next\ (i,\ see\ (e))$.
4. The action selected by the agent is $action\ (next\ (i,\ see\ (e)))$.
5. Goto 2



- Particular type of stateful agent
- Internal environment model
  - Model in addition to current sensory input of a purely reactive agent
  - Model reflects partial state of the “real” environment.
  - Sensed information may be incomplete, inconsistent, or outdated.
- Outdated information may be compensated
  - by letting the agent know “how the world works”.
  - How? A model to predict changes is needed.
  - Example:
    - A car started to overtake me: sensory input a while ago
    - Currently I cannot see it: currently out of sensor range
    - Assumption: the car is still there and will pass soon.

Model-based reactive agent (2)

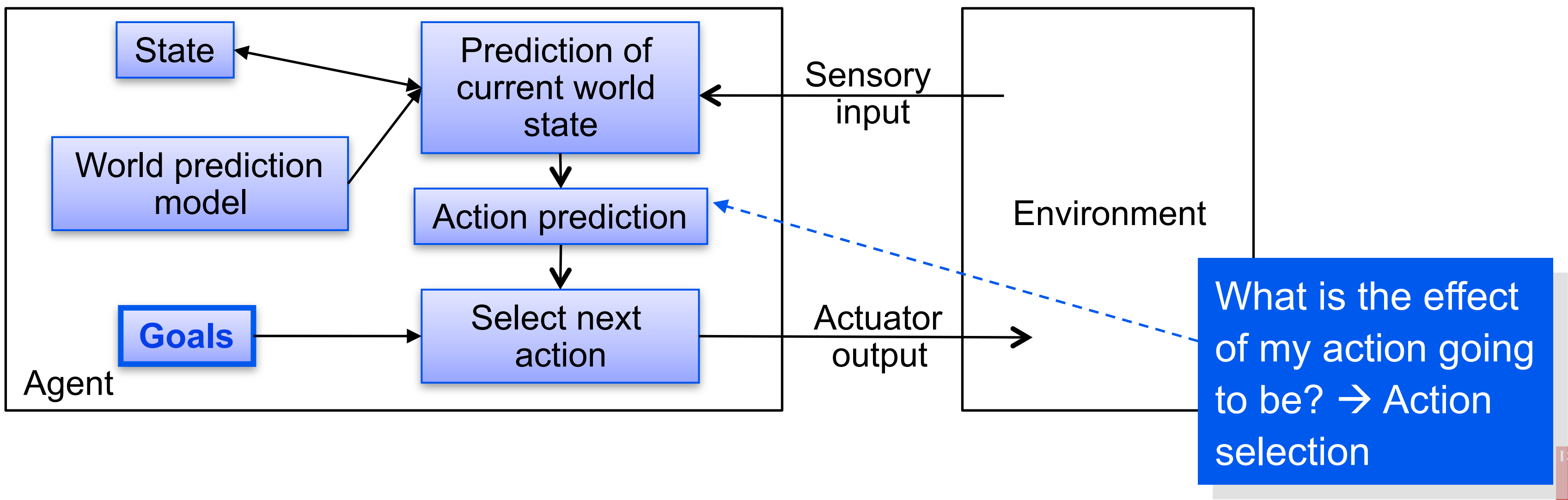



```
Function Reactive-Agent-with-state( percept ) returns action
    state: current known state of the world
    rules: set of condition/action rules
    action: previous action
    state ← update-state( state, action, percept )
    selected rule ← rule-match( state, rules )
    action ← rule-action( selected rule )
    Return action
```

Rules are fixed here! May be altered by learning algorithms.

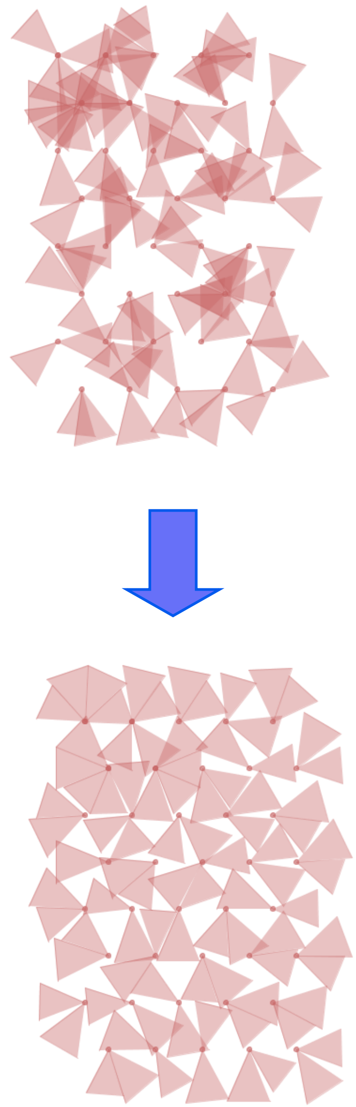
Goal- and utility-oriented agent

- Additionally to the reactive behavior:
  - Give the agent a description of what to achieve.
  - Define a metric for improvement caused by actions → utility
- Example: Additionally to a car driving around (without causing an accident), tell it where to go and what to optimize for (e.g. distance)!
- Which of the possible next actions brings the agent closest to its goal?



Example: Spatial partitioning in camera systems

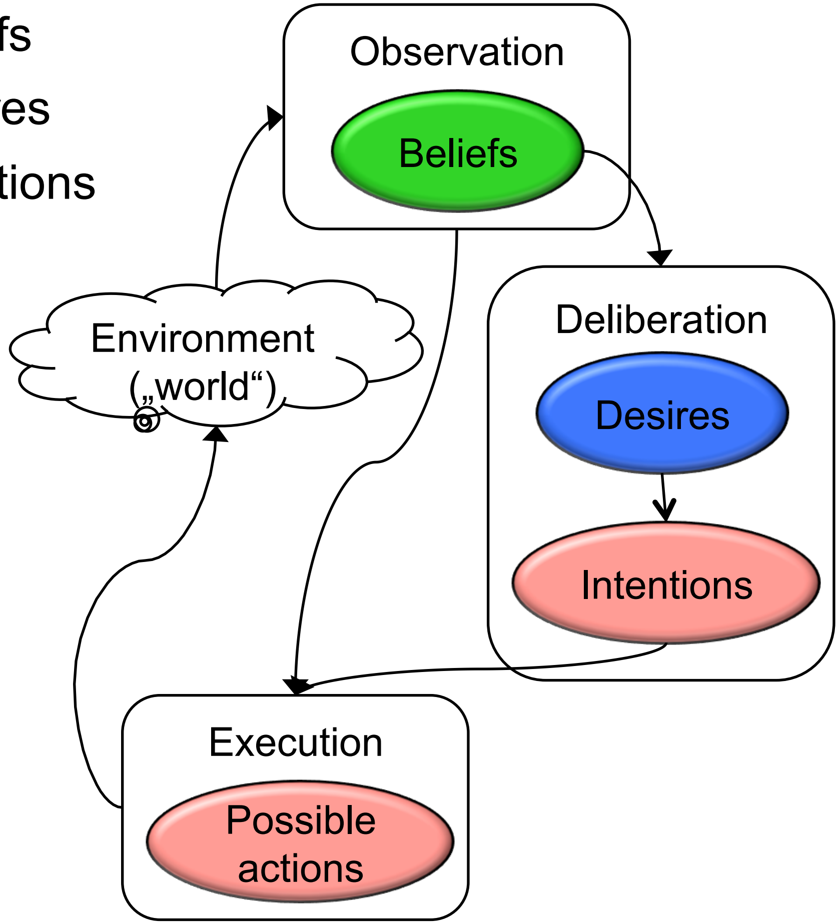
- Goal: Adjust camera's (agent's) field of view in order do minimize overlap with neighbors.
- Model of an agent:
  - Environment consists of the settings of all cameras (position, field of view).
  - Only changes of neighbors can be perceived.
  - Other agents' changes are perceived as environmental changes (no explicit interaction among agents).
  - Internal states I = { waiting, timeout, neighborChanged }
- Idea:
  - If change of neighbor is perceived → consider changing own settings
  - If nothing happened for a while → re-announce own settings

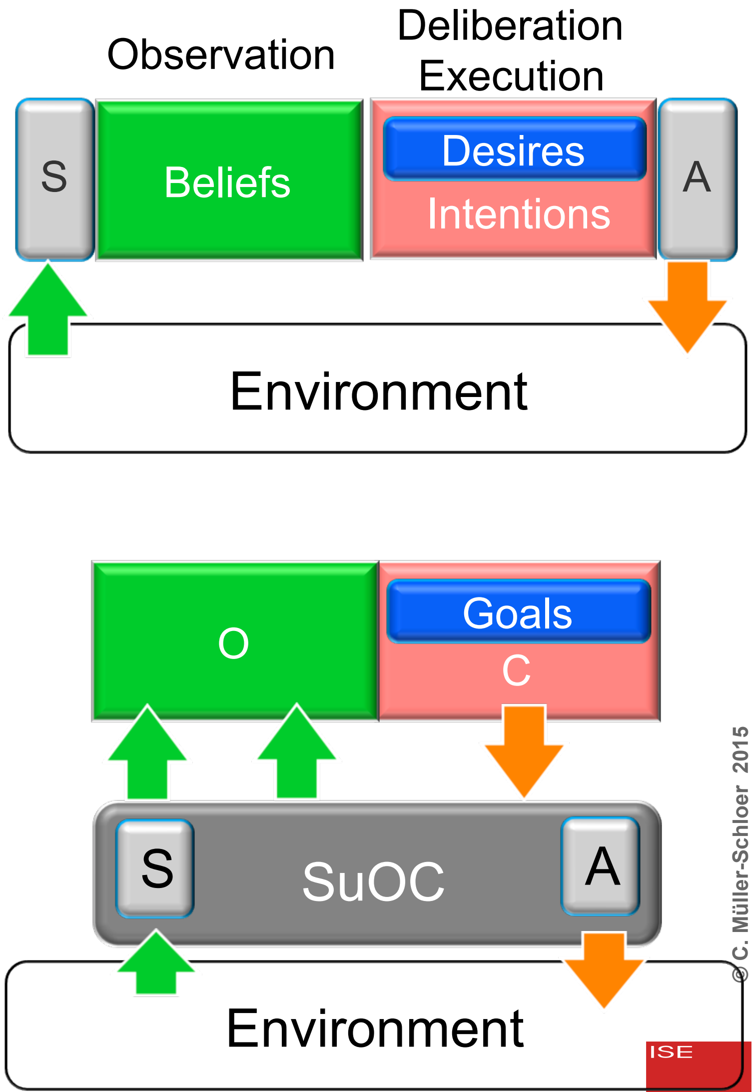


- The BDI software model implements the principal aspects of Michael Bratman's *) theory of human practical reasoning (also referred to as Belief-Desire-Intention, or BDI).
- Psychological model to explain future-directed intention.
- Used in AI and MAS
- Three main components:
  - Beliefs (about the world)
  - Desires (to change the world, = goals)
  - Intentions (how to do it)

*) **Michael E Bratman** (born 25 July 1945) is Durfee Professor in the School of Humanities & Sciences and Professor of Philosophy at Stanford University. His interests include philosophy of action and moral philosophy. His work in those areas led him to the Belief-Desire-Intention model that is used in many areas, including artificial intelligence, today.

- Beliefs
- Desires
- Intentions





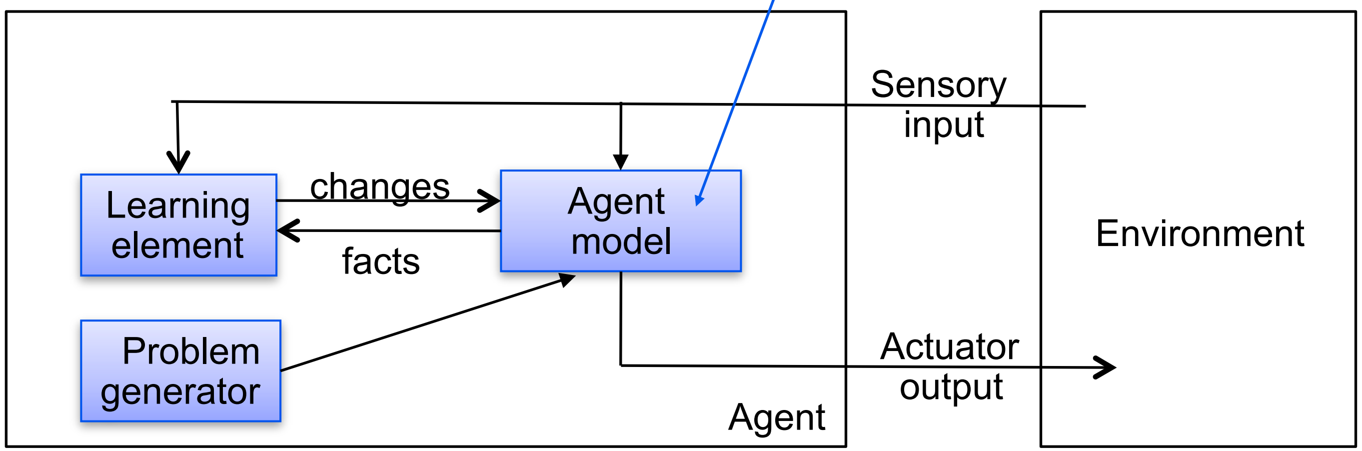
Problem: Practical usage in agents requires reasoning (AI), slow.

- Agents may improve their capabilities over time.
  - Adapt to changing environment.
  - Improve quality of performed tasks.
  - Find new solutions.
- Many machine learning algorithms available, e.g.
  - Artificial Neural Networks (ANN)
  - Learning Classifier Systems (LCS)
  - Evolutionary Algorithms (EA)
- Choice of algorithm class depends on particular problem.
- We will see some examples later in the lecture.

- Additional learning element …
  - can modify the agent in operation.
  - gets feedback from the agent's behavior.
- Problem generator encourages new behavior.
- Online changes may be dangerous and/or time consuming.
- Online versus offline-learning

Problem generator can be the world itself!

Any of the previous agent models

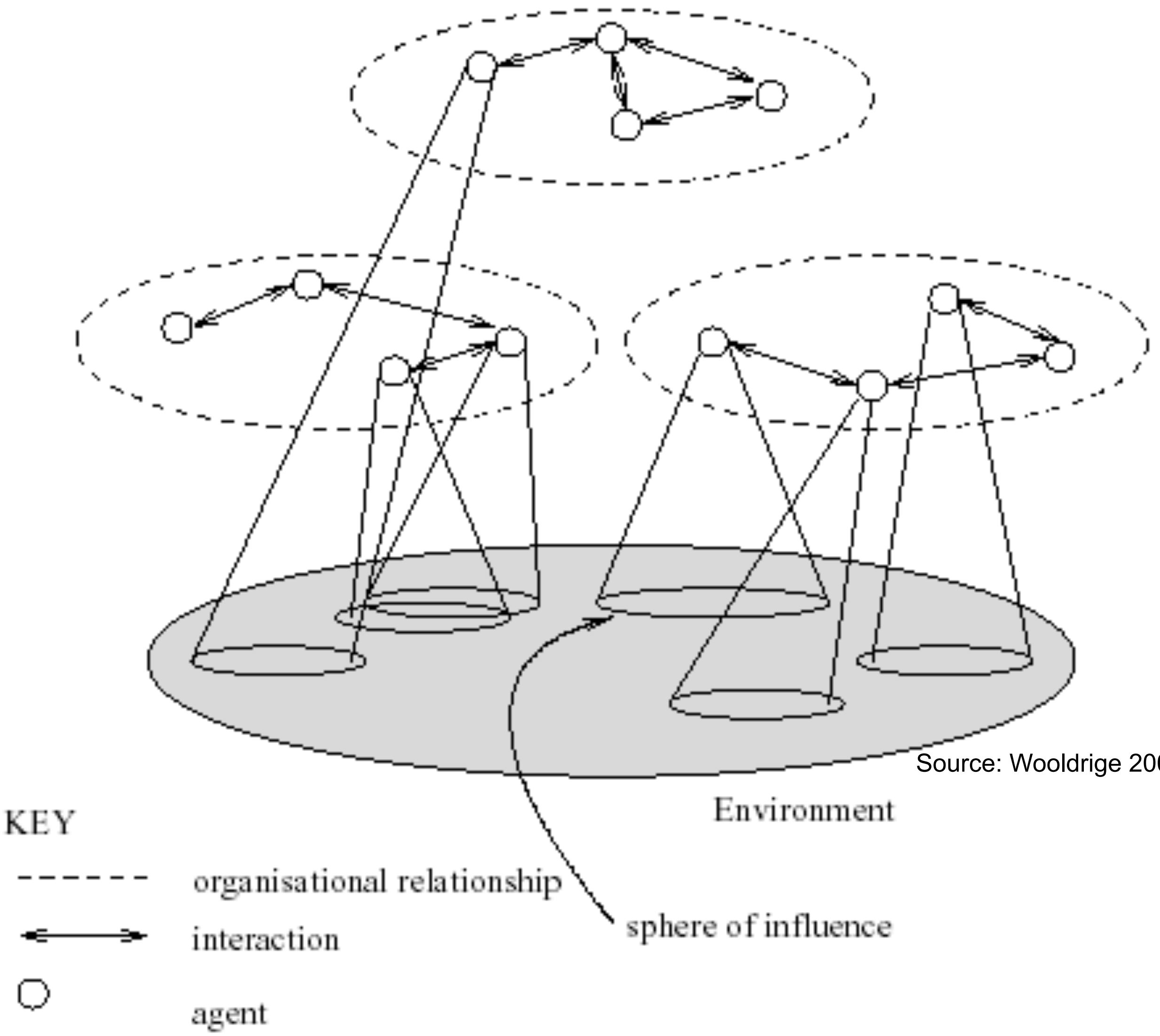


- So far we have considered individual agents. But:
  - Multiple agents share an environment.
  - Agents perceive other agents' changes.
  - So far no explicit coordination and collaboration among agents

→ What if we have multiple agents working together in the same environment?

- Goal of Multi-Agent Systems (MAS)
  - Make explicit use of collaboration mechanisms among agents!
  - Own action may be chosen after negotiating with other agents!
  - Example: Which camera should track a particular object?

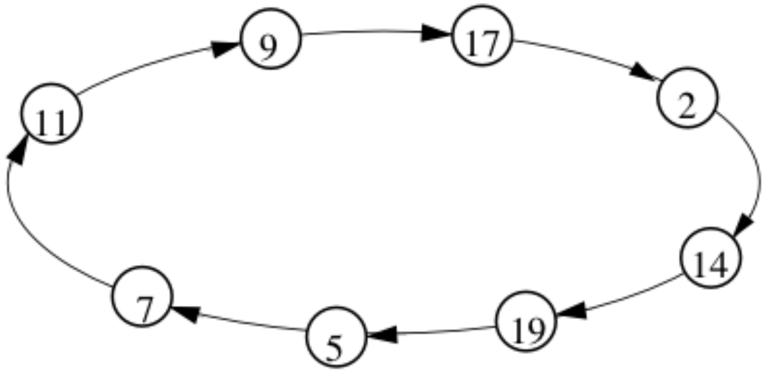
Multi-Agent System



Source: Wooldrige 2002

- In many technical systems all agents “belong to” a single owner.
  - These agents can typically be modeled as cooperative agents.
  - Example: Communication network of a single operator
- Whenever agents “serve” different owners …
  - agents should be modeled as competing or selfish agents.
  - Selfish (groups of) agents aim to maximize their utility.
    - Example: In a multi-operator network, each operator may try to reduce the cost of forwarding packets for others.
- Agreement problems become more difficult to solve, if agents are selfish, e.g.:
  - All the goodies: Resource allocation, election, voting, …
  - Auctions
  - …

- In case of cooperative agents: Agreement can be achieved by many algorithms for different purposes. (→ "classical" distributed systems)
- Example 1: Majority-consensus voting algorithms
  - Goal is to get a majority of agents to agree on doing something.
- Example 2: Leader Election
  - Goal: All agents must agree electing one single agent among them (e.g. to do a particular task).
  - Works because none of the agents "lies" to avoid getting the job.



Assumptions:
- unidirectional ring
- all processes have unique id
- highest id shall be the leader.

Unless someone lies!

An idea for leader election on a ring topology:

- ❑ Each process wakes up at some time, at the latest when it receives a message.
- ❑ Start a complete round on the ring: message contains the highest id found so far.
- ❑ When message returns, it contains the highest id on the ring.
- ❑ Highest id becomes the leader, and each agents knows it.

Auctions among selfish agents (1)

- Consider an item to be sold in an auction.
- Goal: Sell item to that agent who really wants it most!
- Each bidder has a private valuation of the item.
- If everyone was honest, we could use the previous algorithm
  - Collect (honest) bids → highest bid wins
- Assumption: Agents are rational (as opposed to many humans!)
  - They do not bid more than their valuation.
  - May bid less than their valuation in order to pay less.
- How is it achieved that bids are truthful in relation to valuation?

One solution: Vickrey auction

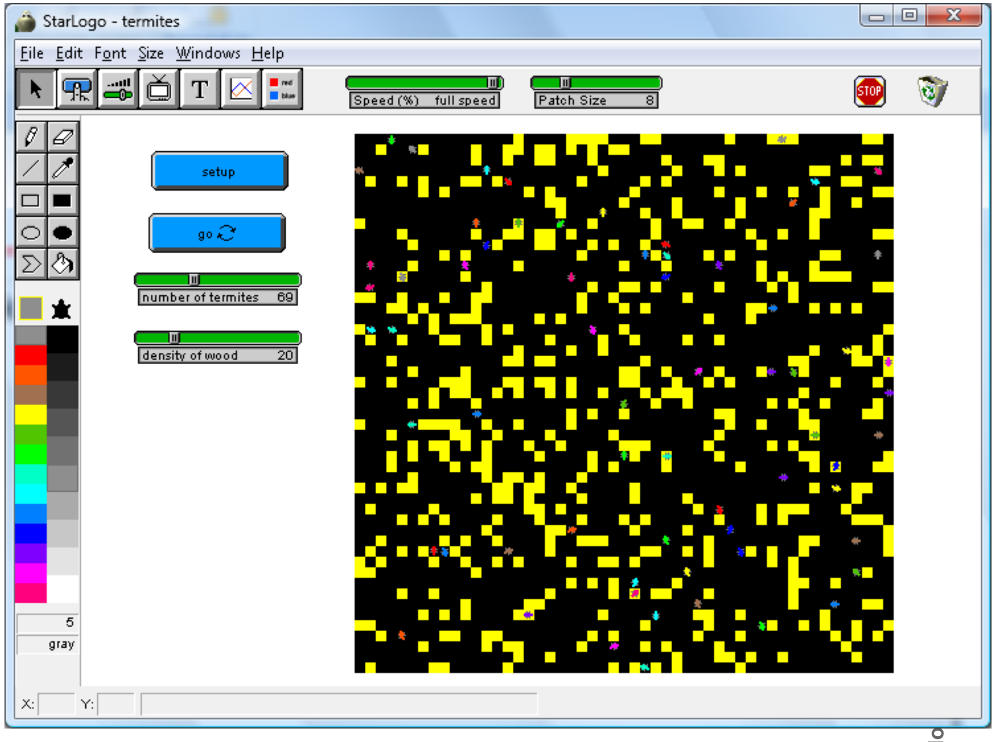
- Each agent places a sealed bid.
- When all bids have been placed:
  - The agent who gave the highest bid wins.
  - But only pays the price of the second highest bid.
- Encourages agents to use real valuations in their bids.
  - Bidding less than valuation → agent may loose auction to someone who wanted to pay a price closer to the real value.
  - Bidding more than valuation → risk to pay over budget
  - Intuitively: Agent does not risk paying “way more” than other bids.
- There are disadvantages: Think about cooperation between bidders!

Auction theory is part of Game theory.

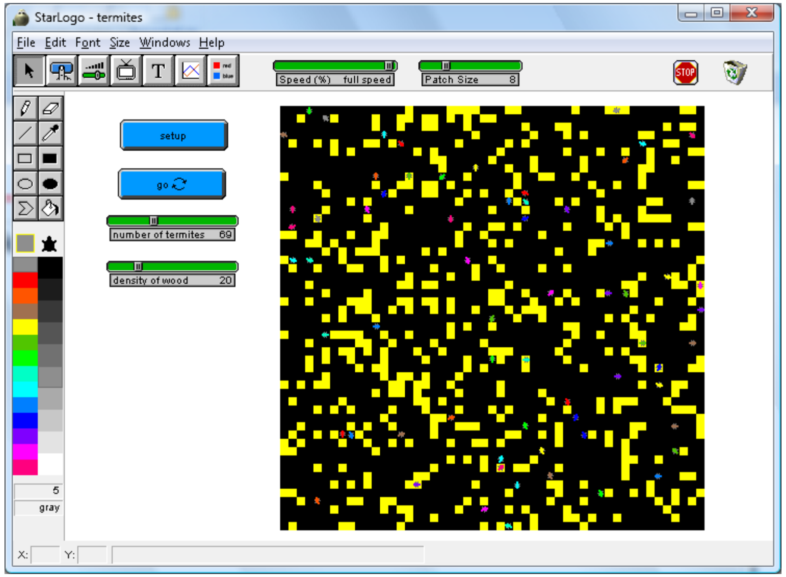
ISE SRA

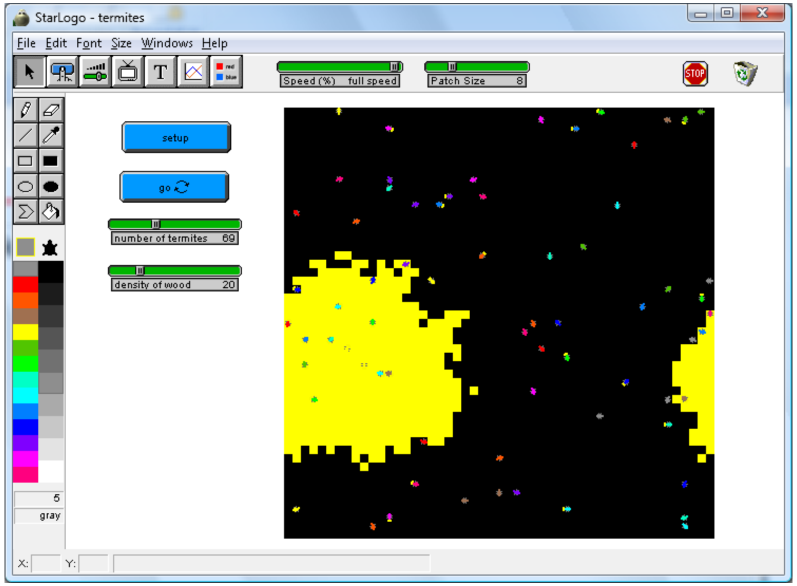
- Mathematical analysis of large Multi-Agent Systems is – at least – very difficult in general.
- Simulation tools are often used for testing and evaluating systems.
- Many tools are available, e.g.
  - Starlogo (http://education.mit.edu/starlogo/)
  - Repast (http://repast.sourceforge.net/)
  - MASON (http://cs.gmu.edu/~eclab/projects/mason/)
  - NetLogo (http://ccl.northwestern.edu/netlogo/)
  - …
- These tools offer/use a programming language for modeling environment and agents.

- ❑ 2D environment contains wood chips (yellow patches).
- ❑ Agents model termites. Termites can …
  - run around searching for wood chips.
  - pick up wood chips.
  - drop wood chips on a pile.
- ❑ Eventually there will be one single pile of wood chips.
- ❑ Example of a simple stateful agent.
- ❑ Included in the Starlogo bundle



Starlogo example: Termites (2)





- ❑ Random initialisation
- ❑ Almost one pile

- ❑ Implementation of the agent‘s behaviour
- ❑ Description of each agent’s high-level behaviour:

```
to go
  search-for-chip   ; find a wood chip and pick it up
  find-new-pile     ; find another wood chip
  find-empty-spot   ; find a place to put down wood chip
end
```

- ❑ Agent walks around and searches for a chip:

```
to search-for-chip
  if pc = yellow                    ; if find a wood chip...
    [stamp black                    ; remove wood chip from patch
     setshape termite-wood-shape    ; orange while carrying chip
     jump 20
     stop]                          ; exit procedure
  wiggle
  search-for-chip
end
```

- Now that the agent carries a chip, search for other chips (pile)

```
to find-new-pile
  if pc = yellow [stop]        ; if find a wood chip, stop
  wiggle
  find-new-pile
end
```

- Go one step in random direction (hopefully away from the pile):

```
to find-empty-spot
  if pc = black                ; if find a patch without a wood chip
   [stamp yellow               ; put down wood chip in patch
    setshape termite-shape     ; set own color back to red
    get-away
    stop]
  seth random 360
  fd 1
  find-empty-spot
end
```

- Once we found a pile and dropped the chip: go and look for new work:

```
to get-away                    ;leave the pile where you put your chip
  seth random 360
  jump 20
  if pc = black [stop]
  get-away
end
```

- Make one step and randomly change heading:

```
to wiggle
  fd 1
  rt random 50
  lt random 50
end
```

Summary

- Examples of agents
- System environment
- Agent types
- Multi-Agent Systems
- Agreement in MAS
- Simulation of MAS

❑ Textbooks

- Michael Wooldridge: An Introduction to MultiAgent Systems, Wiley, 2002
- Stuart Russel, Peter Norvig: Artificial Intelligence – A modern Approach / Künstliche Intelligenz – Ein moderner Ansatz, 2. Edition, Prentice Hall, 2004 (available in german and english)