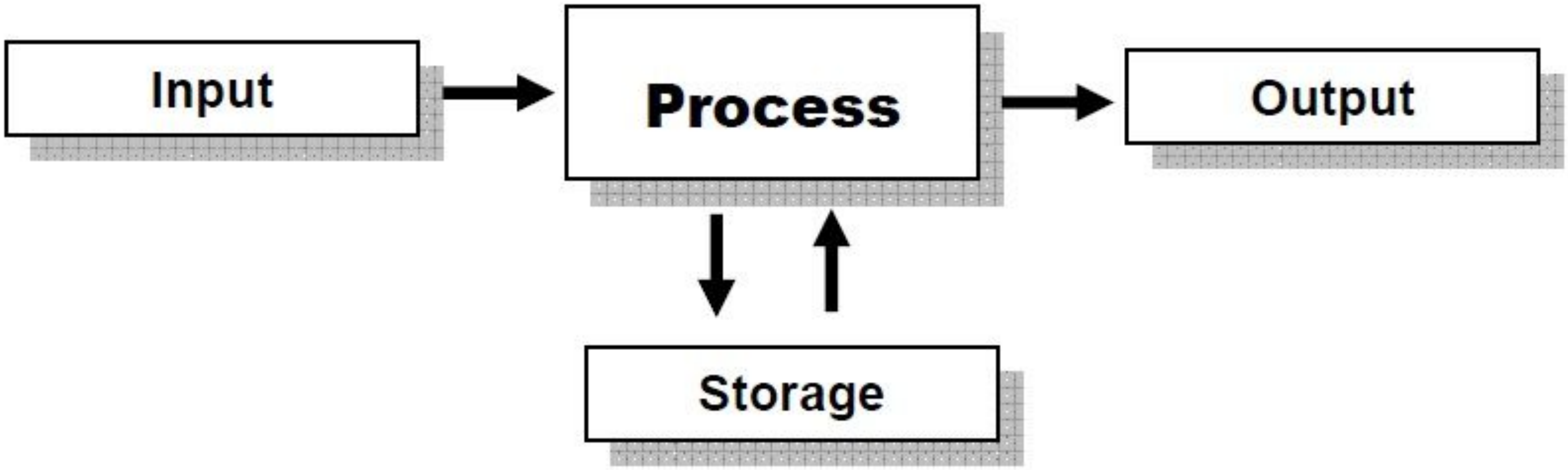
Intelligent Agents: A Gentle Introduction

Giovanni Ciatto @ ASAI-ER, 2023

Algorithms are not enough

What is *computer science* essentially about



What is *computer science* essentially about

Algorithm ≈ *finite* lists of **reproducible** steps to transform input in output

- Defining **algorithms** as the recipe for processing interesting **problems**
  - requires clear **representations** for input / output / storage **data**
- Studying algorithms' **time/memory requirements**, formally
  - as well as their **termination**
- Algorithms can be **combined** to solve **more complex** problems

Example: sorting algorithm (Bubble sort)

Algorithm 1: Bubble sort

```
Data: Input array A[]
Result: Sorted A[]
int i, j, k;
N = length(A);
for j = 1 to N do
    for i = 0 to N-1 do
        if A[i] > A[i+1] then
            temp = A[i];
            A[i] = A[i+1];
            A[i+1] = temp;
        end
    end
end
```

- Input: array of **comparable** items
  - several algorithms to compare items
  - depending on items type
- Output: **sorted** array
  - according to comparison strategy
- Many algorithms with different properties
  - e.g. bubble sort

From computer science to software engineering

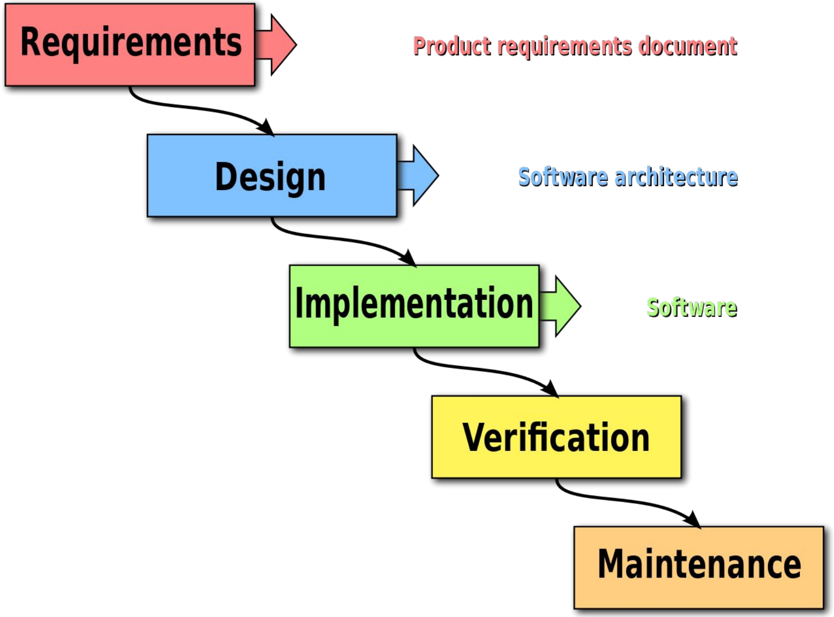
Computer Science

- algorithms **in theory**

Software Engineering

- complex systems made of algorithms,
    - in practice

What is *software engineering* actually about

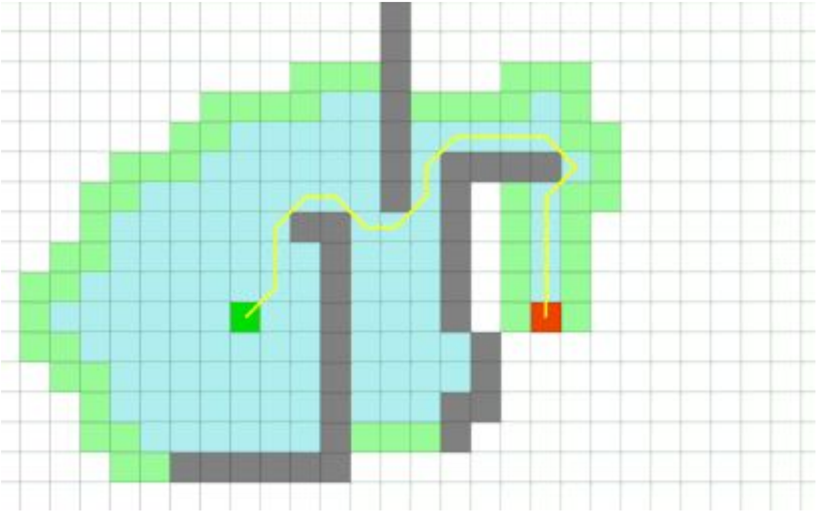


- How to **combine** algorithms into **systems**
  - To create **effective/efficient** software
  - Keeping them operating and **sustainable**
- Algorithms written as **programs**
  - in some **programming language**
- Combination:
  - one program's **output**...
  - ... becomes another program's **input**

What is *artificial intelligence* actually about

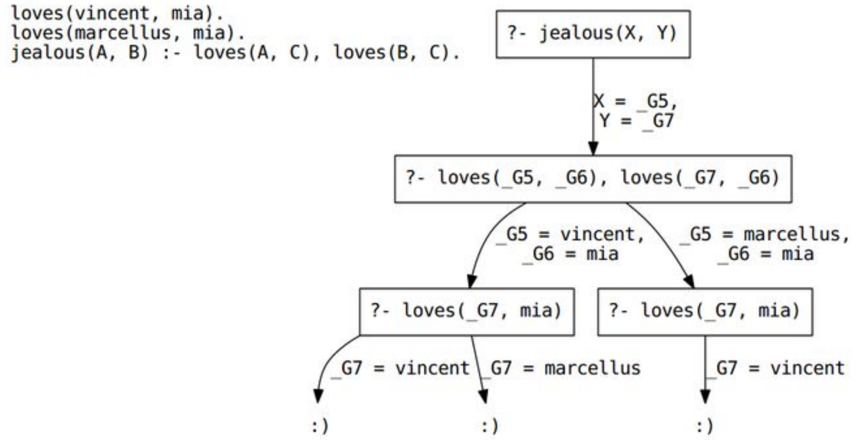
- Creating **algorithms** which **emulate** some typically-human capability
  - e.g. **path-finding** (find path between location A and B)
  - e.g. logical **reasoning** (infer consequences from premises)
  - e.g. **planning** (figure out which actions to do to reach a goal)
  - e.g. **learning** (learn new behaviours from examples)
- Creating **efficient software** implementation for such algorithms

Example: path-finding with A*



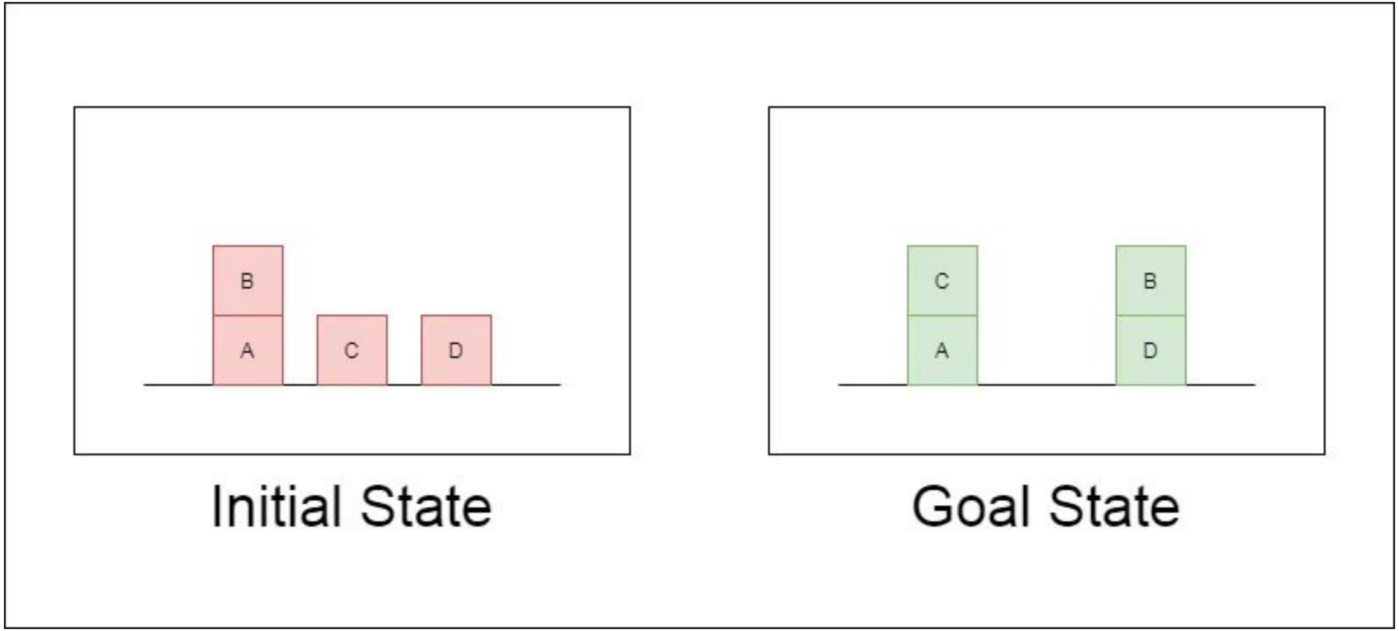
- Problem **modelled** as a **search-problem**
  - on a **graph**-like space
    - i.e. locations and reachability
  - input = source/destination as **nodes**
  - output = sequence of **connected** nodes
- **Combinatorial** problem
  - the bigger the graph...
  - ... the harder to find a solution
- Path-finding ≠ **navigation**
  - navigation is far harder

Example: logic reasoning with Prolog



- Problem **modelled** as a **search-problem**
  - on a **tree**-like space (the **proof tree**)
    - dependencies among **logic rules**
  - input = logic statements to be proved
  - output = yes/no + variables assignment
- **Combinatorial** problem
  - the more the rules...
  - ... the harder to find a solution

Example: planning with STRIPS (pt. 1)



Initial state

- on(b, a)
- ontable(a)
- ontable(c)
- ontable(d)
- clear(b)
- clear(c)
- clear(d)
- armempty

Goal:

- on(c, a)
- on(b, d)

Example: planning with STRIPS (pt. 2)

OPERATORS	PRECONDITION	DELETE	ADD
STACK(X,Y)	CLEAR(Y)∧ HOLDING(X)	CLEAR(Y) HOLDING(X)	ARMEMPTY ON(X,Y)
UNSTACK(X,Y)	ARMEMPTY∧ ON(X,Y)∧ CLEAR(X)	ARMEMPTY∧ ON(X,Y)	HOLDING(X) ∧CLEAR(Y)
PICKUP(X)	CLEAR(X)∧ ONTABLE(X)∧ ARMEMPTY	ONTABLE(X)∧ ARMEMPTY	HOLDING(X)
PUTDOWN(X)	HOLDING(X)	HOLDING(X)	ONTABLE(X)∧ ARMEMPTY

Example: planning with STRIPS (pt. 3)

Possible output plans

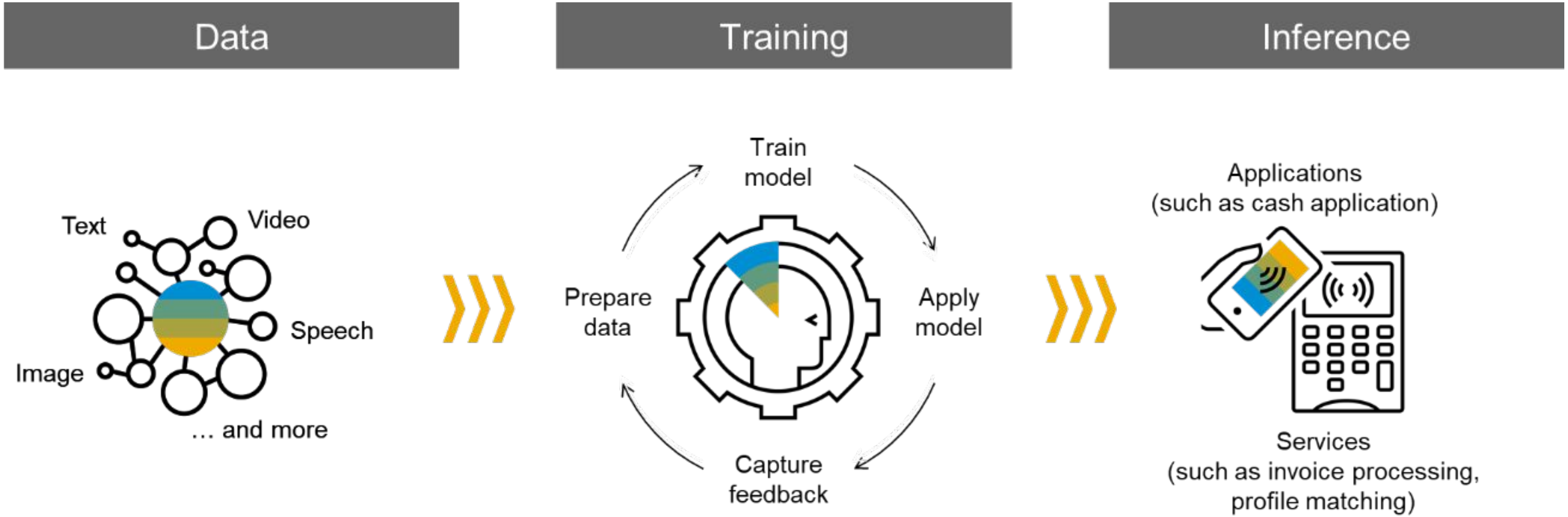
1. unstack(b, a) ; stack(b, d) ; pickup(c) ; stack(c, a)

2. unstack(b, a) ; stack(b, d) ; pickup(a) ; stack(a, b) ; pickup(c) ; stack(c, a)

3. ...

- Problem **modelled** as a **search-problem**
  - on a **graph**-like space
    - i.e. states as nodes, actions as arcs
  - input = initial state + goal
  - output = sequence of **actions**
- **Combinatorial** problem
  - the bigger the state...
  - the more the admissible actions...
  - ... the harder to find a solution
- Planning ≠ **plan execution**
  - e.g. action execution may fail...

Example: machine learning (pt. 1)



Example: machine learning (pt. 2)

- Problem **modelled** as an **optimization-problem**
  - on the **parameters** space of some **parametric algorithm**
    - e.g. linear model, neural network, etc.
  - input = training data + parametric algorithm
  - output = parameters assignment
- Many **statistical issues** to be taken into account
  - e.g. training set separation
  - e.g. over- or under-fitting
- The parametric algorithm is commonly tailored on **one particular task**
  - e.g. image classification, text recognition, user proliferation, etc.
- Better to use learning when the task is not easy to formalise/program manually

Are these algorithm *intelligent*?

- Put it simply:
  - **no,** not if considered **alone**
- BTW, what is **intelligence** after all?
  - philosophical question, many context-dependent answers
  - roughly speaking, just the sake of this talk:

Intelligence ≈ **algorithms** for smart capabilities
+ **criterion** about when/how to use/combine them

What is missing?

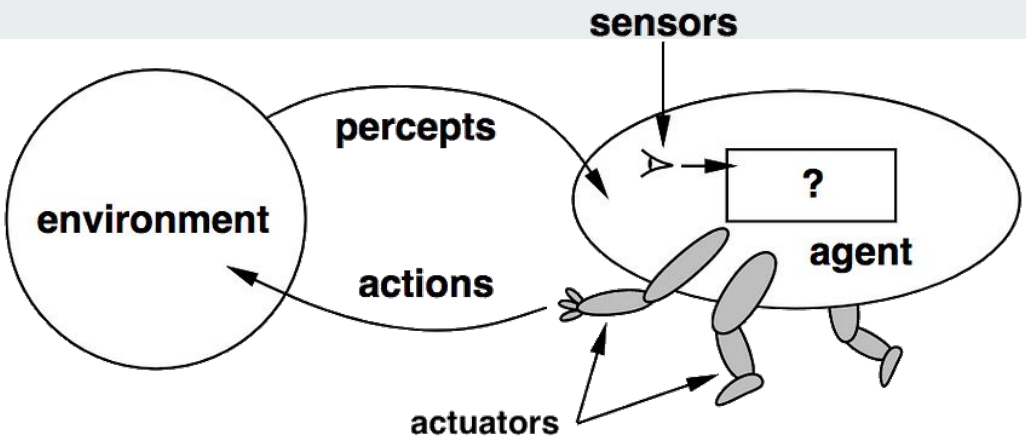
- Abstractions for
  - describing,
  - designing,
  - engineering,
  - implementing

intelligent software entities (spoiler alert: **agents**)

Agents and Multi-Agent Systems

Main Notions

What is an *agent*



- Any entity capable of **acting** to achieve some **goal**
  - while being **situated** into some **environment**
    - which can be both **perceived** and **affected**
  - possibly, along with other agents
    - with which **interaction** is possible
- Possible examples of agents:
  - human beings, OS processes, OS threads, logic solvers, robots, BDI agents

What is a *goal*

- A (possibly **partial**) **description** of the **state** of the world **to be reached**
  - either by a single agent (**individual** goal)
  - or by a multi-agent system (**collective** goal)
- “world” ≈ “the environment + other agents”
- Examples of goals:
  - vacuum robot → “the floor should be clean”
  - autonomous car → “reach destination X”
  - virtual personal assistant → “reminder of meeting, 15 minutes before its start”

Many kinds of goals

Weak goal

- goal “hard-coded” in the agent
- e.g. thermostat

vs

Strong goal

- explicit representation of the goal
- e.g. autonomous car reaching a destination

Achievement goal

- situation to reach (then the goal is achieved)
- e.g. reaching a destination

vs

Test goal

- information to acquire (may imply reasoning)
- e.g. finding missing information, asking to someone

vs

Maintenance goal

- situation to keep stable (may need continuous action)
- e.g. maintain temperature in a given range

Sub-goal

- a goal necessary to reach another goal
- e.g. reaching Tokyo, requires:
    a. drive to the local airport
    b. fly to Tokyo

What is the *environment*

- The space where agents **live** and **(inter)act**.
  - a.k.a. what is **external** w.r.t. agents
  - **enables** & **constraints** agents' **interaction**, **perception**, and **action**
- Examples of environments:
  - human beings → physical world / social media / . . .
  - Roomba → a house and its floor
  - chat bot → chat history
  - autonomous car → the road
  - OS process/thread → file system + network + environment variables + I/O

What is *perception*

- The operation by which agents **gather information** from the environment
  - agents may then represent, memorise, and process perceived information
  - notice that perception may be subject to **error**
- **Percept** = the raw information being gathered
  Sensor = the interface among the environment and the agent
- Examples of perception:
  - human beings → 5 sense + introspection + proprioception
  - robots → input sensors providing raw measurements (cameras, lisars, etc.)
  - chat bot → chat history
  - OS processes/threads → stdin + other input files, environment variables, system clock, network channels, serial ports, etc.

What is *action*

- The operation by which agents **affect** the environment
  - or at least attempt to so
  - notice that **actions may fail** in so many way
- **Actuator** (a.k.a. **effector**) = the interface among the agent and the environment
- Examples of actuation:
  - human beings → hands, feets, virtually any limb of our bodies, speech
  - robots → actuators (wheels, arms, leds, etc.)
  - chat bot → sending messages
  - OS processes/threads → stdout + other output files, environment variables, network channels, serial ports, etc.

Sensors and actuators are commonly coupled

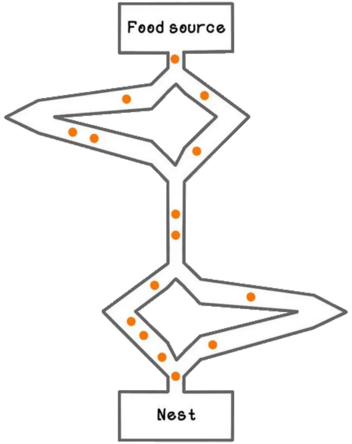
- e.g. wheel + odometry
- e.g. touch + haptic sensation
- e.g. limbs movement + proprioception

What is *interaction*

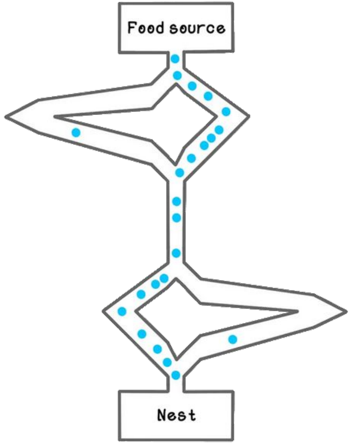
- Where agents **affect** (and are affected by) **each others**
  - may involve both perception and actuation
- Examples of interaction:
  - human beings → speech, mails, chats, non-verbal communication, etc.
  - robots → stigmergy, mutual perception, . . .
  - chat-bot → messages, buttons, etc
  - OS processes/threads → message passing, tuple spaces/centres, etc.

Interaction ⊃ Communication

- Communication = **direct** (commonly, *deliberate*) exchange of information
- Interaction may also occur **indirectly**
  - e.g. via **stigmergy**
    - e.g. ants and **pheromone**



After 4 minutes

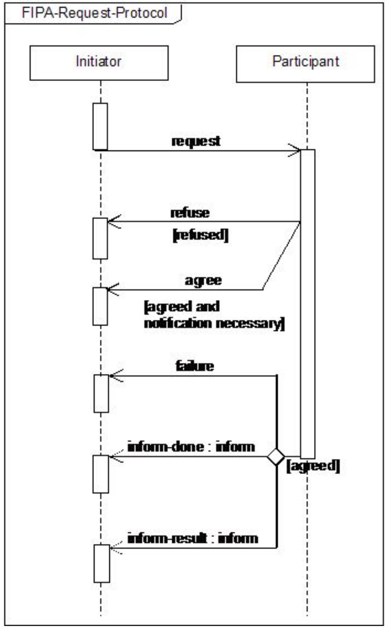


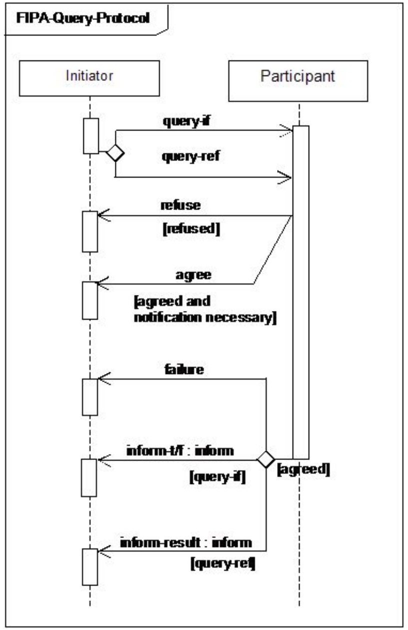
After 8 minutes

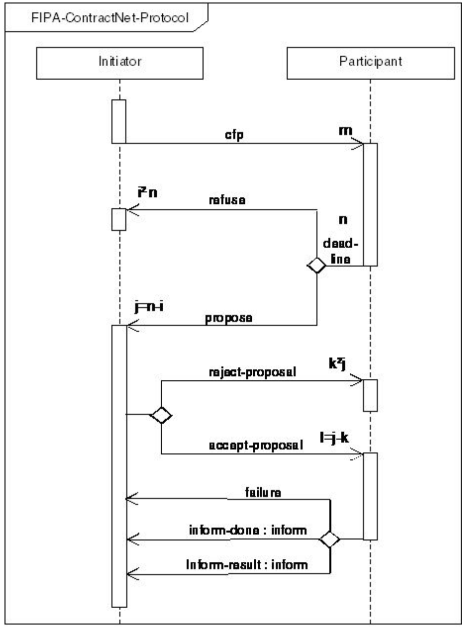
What is *communication*

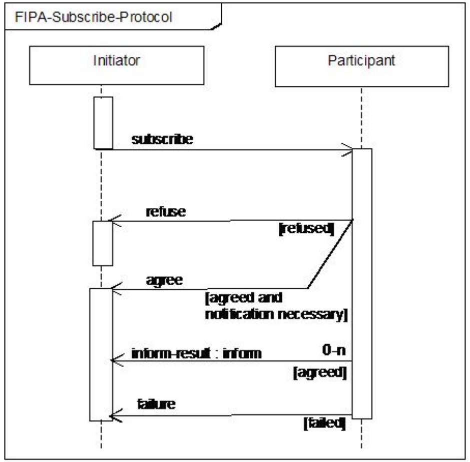
- Roughly speaking: the **exchange** of **messages** to pass information among multiple agents
  - basic mechanism: **message passing**
- Two roles: **sender** and **receiver** exchanging one message
- Message ≈ sender name + receiver name + **payload** (content) + metadata
- The mechanism may be repeated several to times to create **interaction protocols**

Examples of interaction protocols (cf. FIPA IPs)









The role of the receiver in communication

- **Receiver** agent may need to eventually **handle** the message (i.e. trigger some computation)

- How the message is handled depends on the nature of the agent
  - **reactive** agent: will start a computation as soon as the message is receive
  - **computationally-autonomous** agent: memorises the message and decides when/how to handle it

About *autonomy* (of agents)

- Agents are **autonomous** when they **encapsulate** (i.e. control) the **criterion**
  - by which they **select** which **goals** to pursue
    - (**motivational** autonomy)
  - or by which they **choose** which **action** to do to while pursuing a goal
    - (**executive** autonomy)
- Examples of autonomous agents:
  - **human** agents are **autonomous**
  - software agents may be more or less autonomous
    - depending on how they have been programmed

About *intelligence* (of Agents) pt. 1

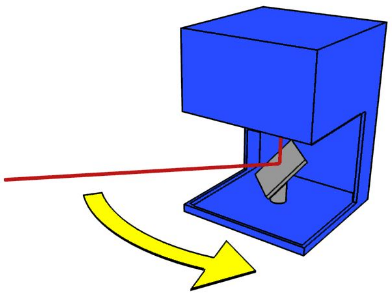
- Agents are **intelligent** when they have **cognitive capabilities**...
- ... and they **know when/how to use** them to pursue their goal(s)
    - **perceiving** stimuli and **recognise abstractions** on top of them
    - **representing** knowledge (e.g. perceptions, abstractions, goals, actions, etc)
    - and **memorising** it for later **re-use**
    - **learning** from the **experience** (i.e. **generalise** the gathered knowledge)
    - **planning** courses of action to pursue goals
    - **reasoning** about knowledge (to deduce implicit knowledge, to induce new knowledge, to abduce hypotheses)
    - **interact** with other agents to exchange information (goals, knowledge, plans)

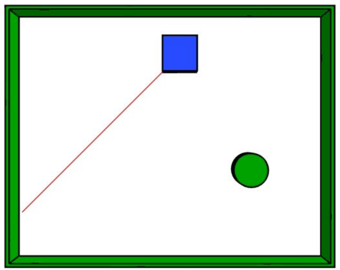
About *intelligence* (of Agents) pt. 2

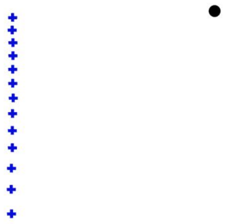
- **Cognitive capabilities ⇏ Intelligence**
  - Cognitive behaviours may or may not be considered as intelligent depending on the context they are applied into, and on the observer
- Example:
  - agent stepping through the window at ground floor
  - agent stepping through the window at Nth floor

Focus on cognitive capabilities

About *perception*

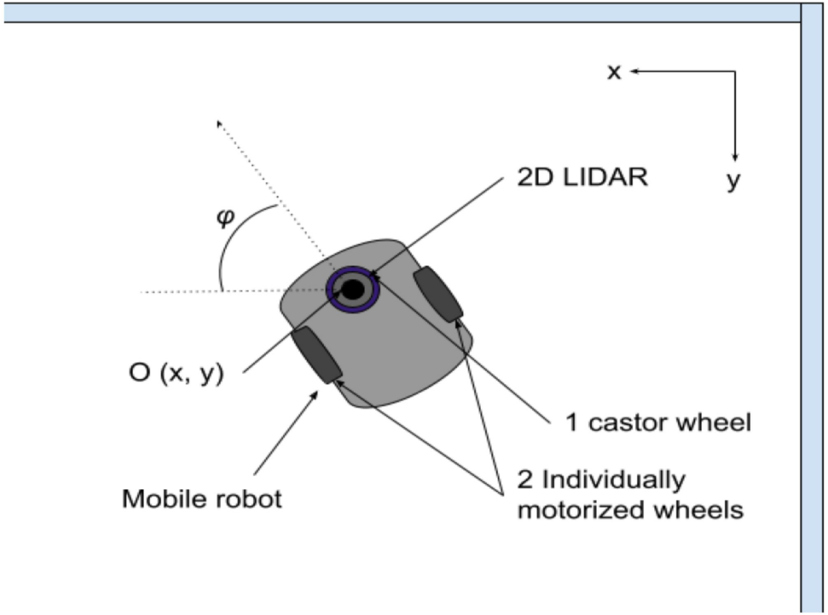






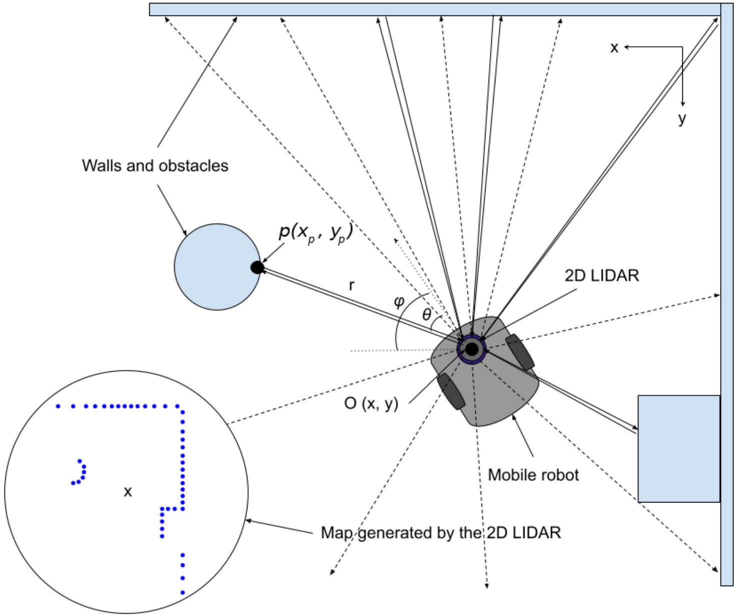
- Commonly consists of getting **raw numbers** from sensors
  - especially in physical environments
- Relevant examples:
  - e.g. LIDAR (Laser Imaging Detection and Ranging)
    - returns distance of obstacle (maybe + angle)
  - e.g. light sensor
    - the more intense the light, the higher the percept value
  - e.g. proximity sensor
    - the closest the obstacle, the higher percept value

About the *reference frame*



- Perception is rooted into a **reference frame**
- ... which rotates and moves with the agent

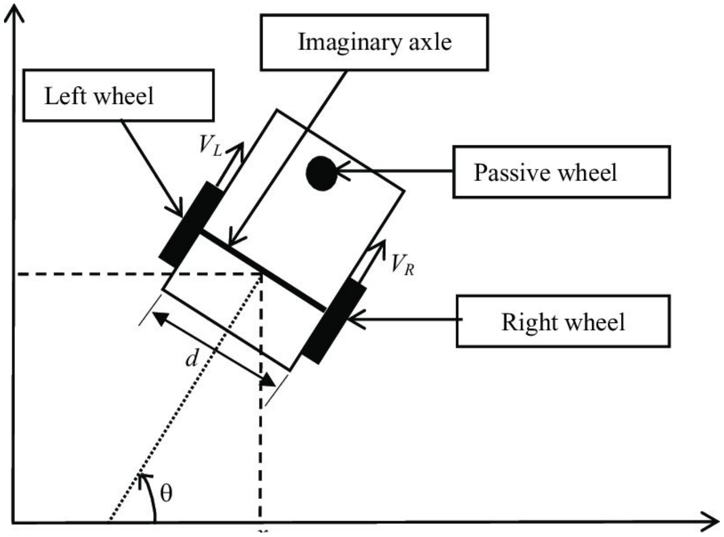
The *role* of *representation*



- **In-memory** representation of the **environment**
- Leveraging on some **model** (of the environment)
  - e.g. relative distances on the plane
- Reified into some **language** / data schema
  - e.g. matrices, first order logic
- Constructed on top of **perception**
  - updated when new percept are sensed
- Enabling (more) complex **deliberation**
  - e.g. compute path, self-localise, etc.
- ... and therefore smarter / articulated **actions**
  - e.g. navigate to destination

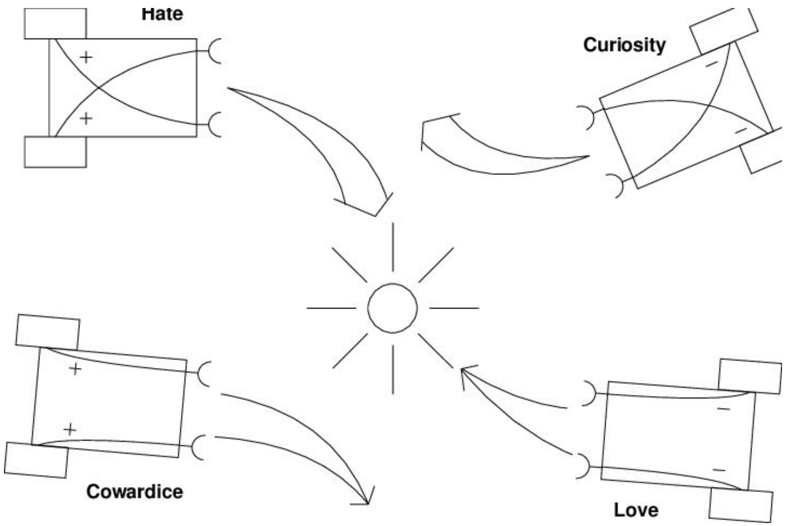
About *actuation*

- Commonly consists of sending **raw numbers** to actuators
- The choice of actuators is commonly **constrained / limited**
  - e.g. because of cost- or physics-related reasons
- **Complex** actions can be **engineered** on top of simple actuators
- Notable example: differential wheeled robot
  - 2 parallel, independent wheels
  - actuators can only regulate angular speeds
  - the robot must be able to
    - go straight (equal angular speed)
    - turn right (left angular speed > right angular speed)
    - turn left (left angular speed < right angular speed)
    - rotate (opposite angular speeds)

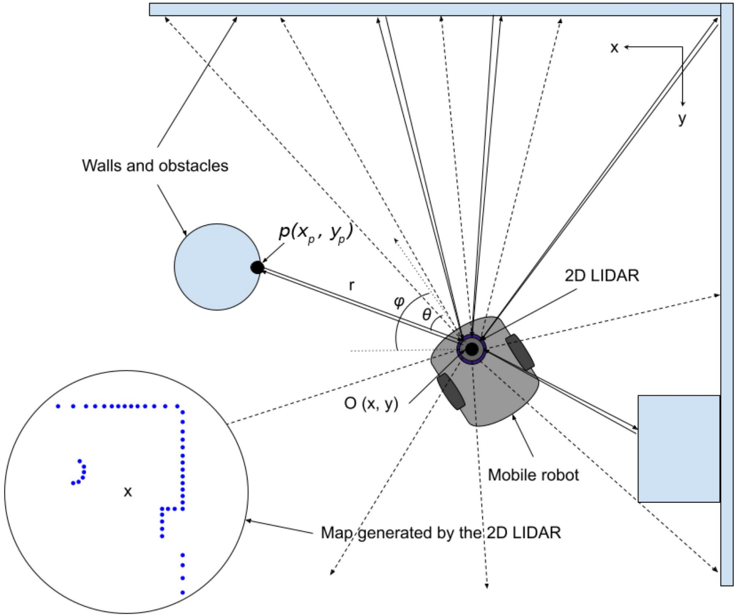


Is representation always needed?

- According to Brooks, not really: cf. [Intelligence without representation]
- e.g. Braitemberg vehicles
  - simple robots
  - stimulus-response hardwired
  - exhibit non-trivial behaviour
- e.g. thermostat
  - repeat forever:
    - if temperature high, cool down
    - if temperature low, warm up
- in general, no representation is ok:
  - for simple control systems
  - to serve some weak goals



How to represent the environment?



Reference frame:

- 2D plane centered on the robot
- obstacles and walls are non-walkable

Raw percepts:

- sample(θ_1, d_1)
- sample(θ_2, d_2)
- ....

Maybe, after data fusion:

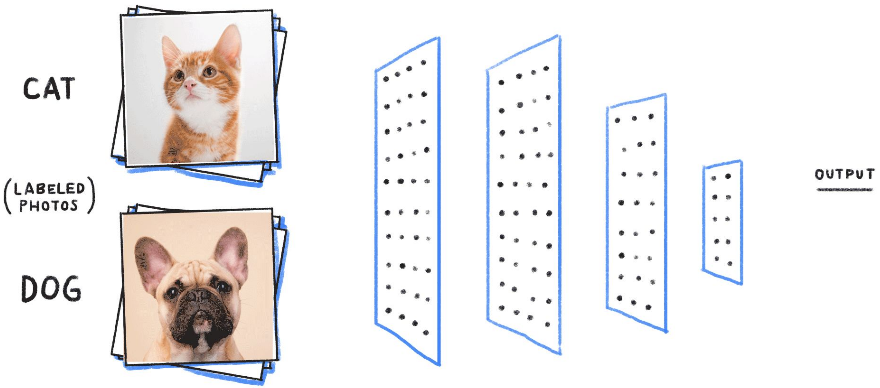
- obstacle(x, y)
- wall(dx_1, dy_1)
- wall(dx_2, dy_2)
- ....

Why to represent the environment?

- To make it possible to exploit **algorithms** to automate **decision-making**
  - hence performing adequate / smart / intelligent actions
  - recall that algorithms require clear definitions of input data
- Example: the vacuum robot
  - keep track of portions of floor not yet / already clean
  - compute path from one room to the other
  - localise self into the house
  - navigate to target room
  - go back to recharge station

Learning what?

Data-driven learning algorithms require fixing what to learn



Common choices made data scientists:

1. the target task
   a. e.g. classification, regression, clustering, etc.
2. the admissible input data
   a. e.g. tables, pictures, time series, audio, video, etc.
3. choose the admissible outcomes
   a. classes, amount of clusters, etc.

Not fully automatic workflow:

- training and inference are algorithms
- design choices are for humans

Can software agents learn from data?

Agents who actually learn should:

- have access to data
- be autonomous in design decisions
  - unless the learnable behaviour is known at design time
- have computational power

This is an open research problem!

Agents exploiting pre-trained models:

- no need to access data
- they can use the model for perception
- no need to take decisions or compute

This is state of the art

Can software agents...
... *plan* their course of action?
... *reason* to draw novel, original conclusions?

- Same argument as previous slide
  - either the modelling of planning / reasoning is modelled at design time...
  - ... or the agent should be autonomous in taking the decisions commonly taken by designers
    - and this is an open research issue

Many sorts of agents out there

Oversimplified, yet useful, map of what people mean by “agent”

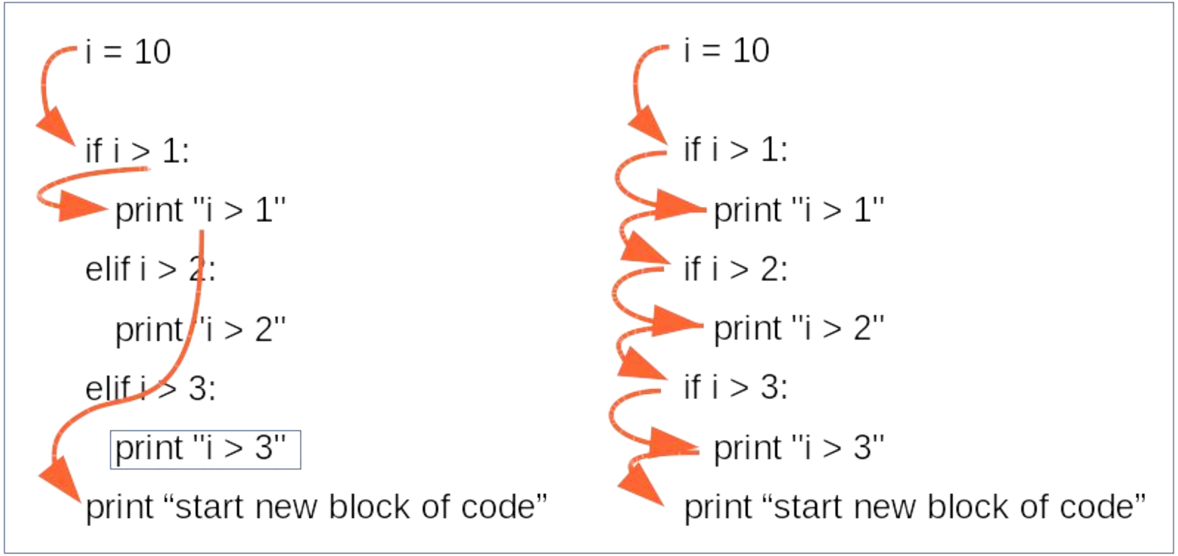
Classic AI agent

- Agents are the entities encapsulating **behaviour**
  - and, therefore, **intelligence**
- "Agent" is a useful **abstraction** to
  - comunicate AI
  - lue AI algorithms together
  - model AI-based systems
  - contextualise individual AI contributions
  - ...
- People in this context may refer to agents meaning "intelligent entities"
  - most commonly, but not necessarily, **software** entities

Concurrency: focus on control flow issues

- Agents are the entities encapsulating **control flow**
  - as a precondition for their **autonomy**
- “Agent” is an **active** software component doing some long-lasting task
  - e.g. threads, processes, daemons, etc.
- People in this context may refer to agents meaning “active software entities” i.e. “software entities having their own control flow”

Important notion: *control flow*



Roughly:

- the sequence of instructions executed a untime

Sequential program

- 1 control flow

Concurrent program

- multiple-control flows

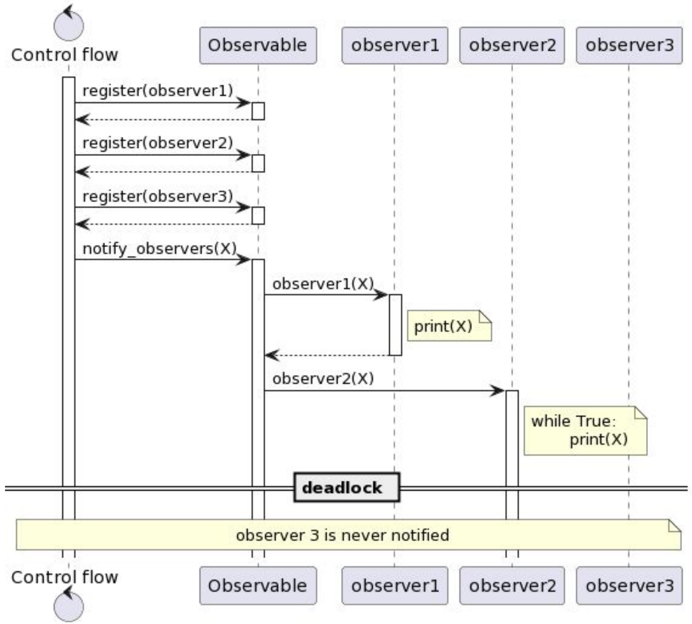
Control flow and objects (pt. 1)

```python
class Observable:
    def __init__(self):
        self.observers = []

    def register(self, observer):
        self.observers.append(observer)

    def notify_observers(self, arg):
        for observer in self.observers:
            observer(self, arg)
```

```python
def observer1(observable, arg):
    print(f'observer1 receives notification form {observable}:', arg)

def observer2(observable, arg):
    while True:
        pass
    print(f'observer2 receives notification form {observable}:', args)

def observer3(observable, arg):
    print(f'observer3 receives notification form {observable}:', arg)
```

Control flow and objects (pt. 2)

```
observable = Observable()

observable.register(observer1)
observable.register(observer2)
observable.register(observer3)

observable.notify_observers("X")
```



Control flow and objects (pt. 3)

- Control flow steps through objects
- Objects are passive entities
  - they only act when traversed by control flow
  - objects do not encapsulate control flow
- Agents do encapsulate control flow
  - agents can say no

Distributed systems: focus on interaction protocols

- Agents are the entities in charge of **communicating over the network**
  - commonly, by **message-passing**
  - possibly, enacting several **protocols** simultaneously
- "Agent" is a party in some **protocol**
  - e.g. the client, the server, the broker, the proxy, etc.
- People in this context may refer to agents meaning the "software parties involved in a procol" or "one node of the distributed system"

Robotics: focus on embodiment & physical world

- Agents are the **control software** of **robots**
  - robots have **bodies** which are immersed in the **physical world**
- “Agent” is a robot (there including **mind + body**)
  - e.g. the autonomous car, the vacuum robot, the robotic arm
- People in this context may refer to agents meaning the “robot” as an animated entity, or “the mind of the robot”

Simulation: focus on behaviour of simulated entities

- Agents are the **entities** in the **simulated** world
  - immersed in a simulated environment
  - subject to **simulated time**
- “Agent” is a **simulated entity**
  - e.g. a pedestrian in a city, a car in the traffic, a molecule in a solution, etc
- People in this context may refer to agents meaning any “active entity” involved in a simulation

Important notion: *simulation*

- (Computational) Simulation ≈ reproducing a system dynamics via software

- General **workflow** of in-silico experiments:
  a. system under study is **modelled** in a parametric way
  b. several simulations are run with different **parameters**
  c. **statistics** are computed, and patterns are identified
  d. **conclusions** are drawn from those statistics/patterns

- Important aspects of simulations
  a. **reproducibility**
     - determinism

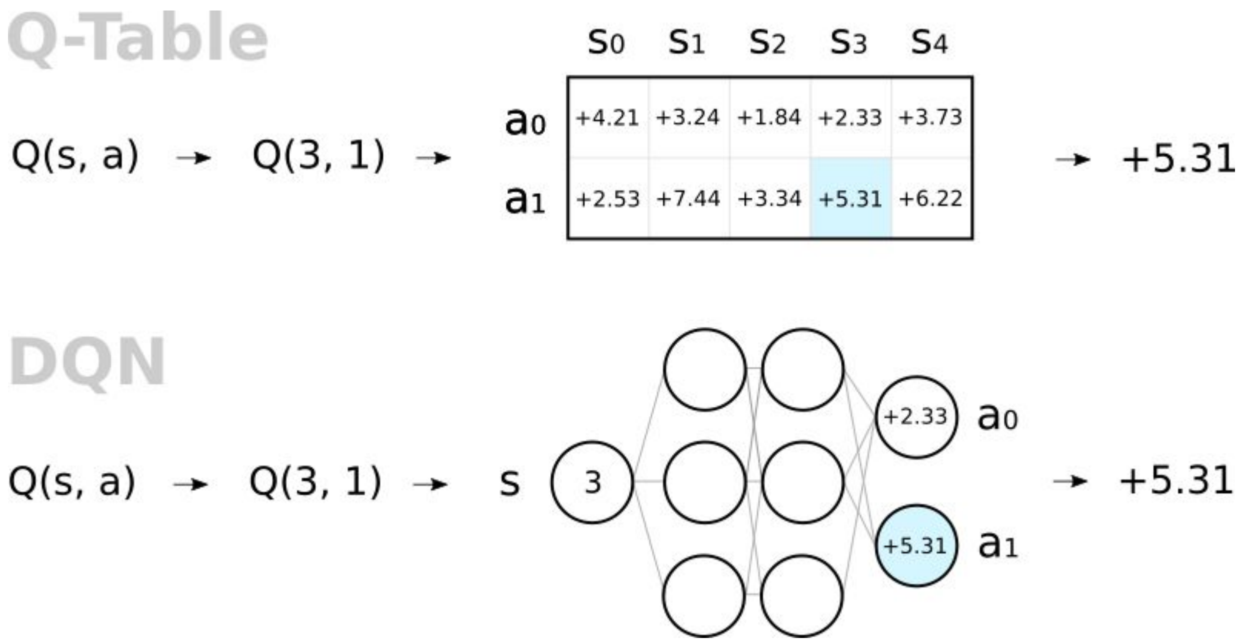
Important notion: *multi-agent based simulation*

- System under study is modelled as a MAS
    a. i.e. several interacting agents into a virtual environment
- Scientists are commonly interested in
    a. analysing the **evolution** of the system
    b. analysing the **state reached** by the system after a while
- Two sorts of simulators: **discrete time** vs. **discrete events**
- MABS often opposed to numerical resolution of **differential equations**

Reinforcement learning: focus on learning policies

- Agents are the entities subject to training (by reward)
- The goal is to learn a **policy**
  - policy = function returning the best action to do in each possible state
    - state = perceived configuration of the environment
  - optimal policy is the one which maximises expected reward on the long run
    - according to past reward
- People in this context may refer to agents meaning the entity whose perception, action, and reward should be modelled

Reinforcement learning: Q-table vs DQN



Agent-oriented programming: focus on languages

- Agents are yet another syntactic category of **programming languages**
  - such as functions, classes, etc.
- **AOP** is considered the **next leap** in programming languages, after OOP
- “Agent” is an **active object**, encapsulating **control flow** and communication
  - e.g. JADE agents
- People in this context may refer to agents in the same way OOP programmers refer to classes

AOP as the next leap in programming (after OOP)

	Monolithic Programming	Modular Programming	Object-Oriented Programming	Agent Programming
Unit Behavior	Nonmodular	Modular	Modular	Modular
Unit State	External	External	Internal	Internal
Unit Invocation	External	External (CALLed)	External (message)	Internal (rules, goals)

How to program software agents

In particular, cognitive agents (e.g. BDI)

Agents programming 101: the *control-loop*

```python
# some data structure here
memory = dict()

while True:
    percepts = sense()
    memory = update(memory, percepts)
    action = deliberate(memory)
    act(action)
```

```python
def sense():
    ... # return set of percepts

def update(old_memory, percepts):
    ... # return updated memory

def deliberate(memory):
    ... # return action representation

def act(action):
    ... # actually performs action
```

Example: the *thermostat* agent (pt. 1)

```
# temperature sensor modelled as Unix file (to be read)
temperature_sensor = open('/dev/temperature_sensor', 'rb')

# air pump actuator modelled as Unix file (to be written)
air_pump_actuator = open('/dev/air_pump', 'wb')

# memory modelled as key-value dictionary
memory = { 'hot_threshold': 30, 'cold_threshold': 20 } # °C
```

Example: the *thermostat* agent (pt. 2)

```
def sense():
    return temperature_sensor.read(1)[0]
```

- Read 1 byte from the temperature sensor
  - as simple as reading a file
- Assumption: the sensor outputs current temperature in **Celsius degrees**
  - encoded as **bytes**

Example: the *thermostat* agent (pt. 3)

```
def update(old_memory, percepts):
    new_memory = dict(**old_memory)
    new_memory['current_temperature'] = percepts
    return new_memory
```

- Copy-pastes the old memory into a new memory, updating current temperature
- Assumption: percepts actually consist of a single integer number
  - i.e. temperature value in Celsius degrees

Example: the *thermostat* agent (pt. 4)

```
def deliberate(memory):
    if memory['current_temperature'] >= memory['hot_threshold']:
        return 'cooldown'
    elif memory['current_temperature'] <= memory['cold_threshold']:
        return 'heatup'
    else:
        return None
```

- Assumption: actions encoded as **strings**
  - “**cooldown**” or “**headtup**”
  - **None** denotes the lack of action

Example: the *thermostat* agent (pt. 5)

```
def act(action):
    if action == 'cooldown':
        air_pump_actuator.write(b'\0x01')
    elif action == 'heatup':
        air_pump_actuator.write(b'\0x02')
```

- Assumption: actuator expects commands to be provided as bytes
    - **01** will pump **cold** hair
    - **02** will pump **hot** hair

Example: the *thermostat* agent (Q/A)

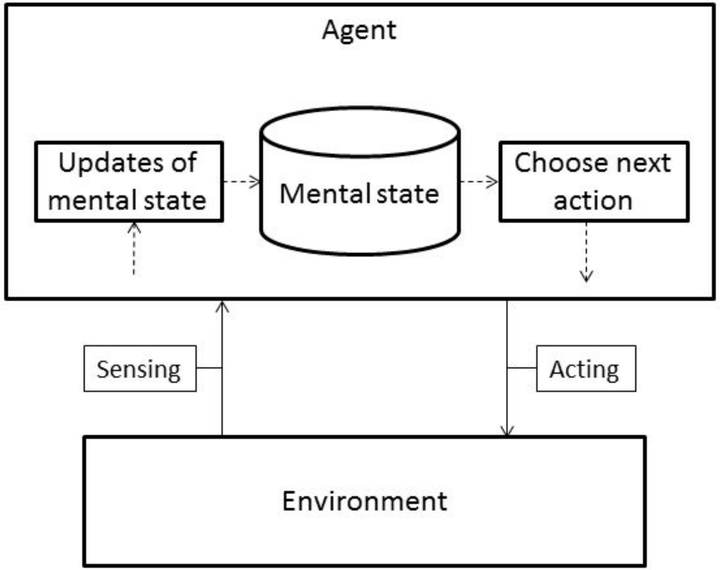
- Is the thermostat control-loop an **algorithm**?
- What is the **goal** of the thermostat agent?
- Is that a **strong** or **weak** goal?
- Is that a **maintenance-**, **achievement-**, or **test-**goal?
- Is the thermostat agent **reactive** or **proactive**?
- Is the thermostat agent **representing** the environment?
- Imagine **2 or more** thermostat agents in the same closed room:
    - are they **communicating**? are they **interacting**?

About *cognitive* agents

Cognitive ≈ focus on human-like capabilities/abstractions

Central aspect: **mental state**

- i.e., internal representation of:
  - **self**
  - environment
  - **other agents**



BDI agents: a particular case of cognitive agents

Cognitive aspects fitting the mental state of each agent:

- **Beliefs**: the (possibly imprecise) things the agent knows
- **Desires**: the goals the agent is willing to (eventually) pursue
- **Intentions**: the activities the agent is currently performing
    - possibly, following some **plan**
- **Plans**: the procedural knowledge about how to pursue goals

The role of *events* in BDI agents

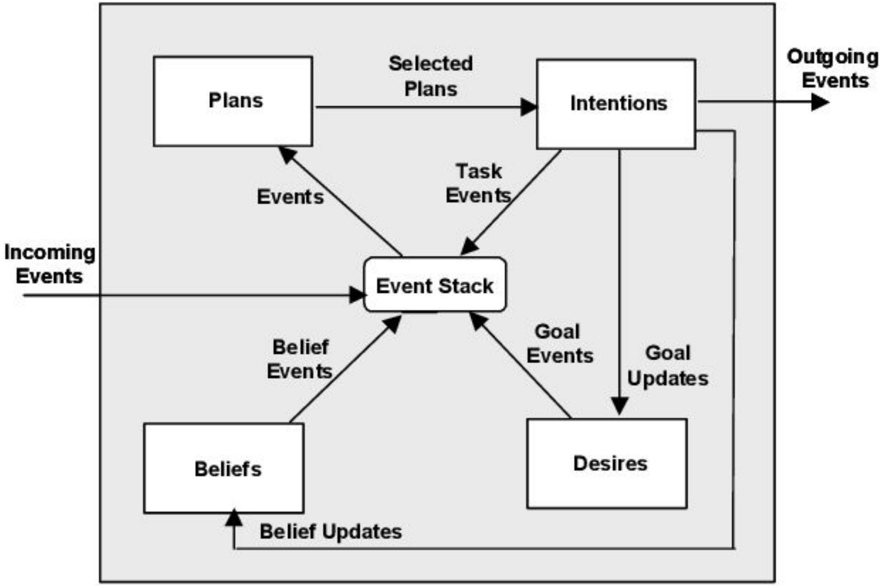


Figure 1: Architecture of the Basic BDI Agent

Event-driven architecture:

- everything happens in response to events

What is an **event**:

- data structure representing relevant **happening**
  - belief addition / update / removal
  - new (sub-)goal to achieve / test / maintain
  - failed (sub-)goal
  - etc.

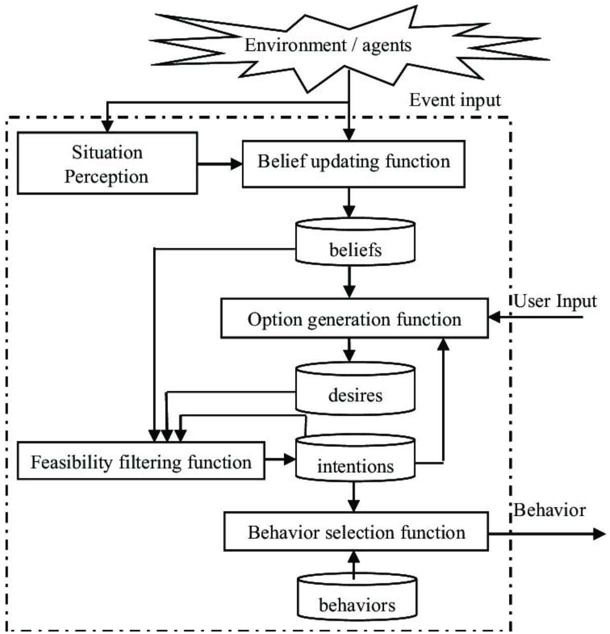
New events may spawn **intentions** to execute **plans**

- i.e. control flows controlling the agent' behaviour
  - with some predefined course of action

Control-loop of BDI agents

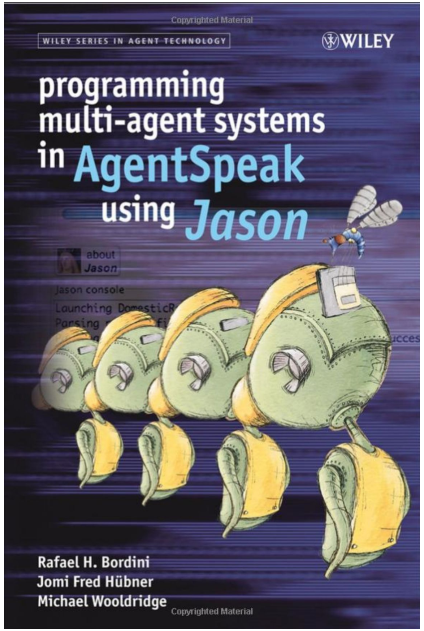
repeat forever

1. **revise** belief base with new messages and percepts
2. add relevant events to the **events stack**
3. **pick** next event from the event stack, if any
    a. if none, do nothing
4. if event is about some **prior intention**
    a. select that intention
5. otherwise, select a **plan** for the event
    a. create a **new** intention out of that plan, if any
    b. otherwise, add **failure-event** to events stack
6. **execute one step** of the intention
    a. in case of new/failed goals, update



AgentSpeak and *Jason*

- AgentSpeak: formal semantics for BDI agents
  - introduced by Anand S. Rao in [seminal paper]
- Jason: actual programming language for BDI agents
  - introduced by Bordini, Hubner, and Woolridge
  - inheriting & extending Prolog's syntax
  - technologically rooted on the JVM
- Both languages enable AOP with BDI abstractions



Jason: syntax overview (pt. 1)

Beliefs:

- `human(socrates).` — a **fact** which is known to be true
- `mortal(X) :- human(X).` — a **rule** for deducing what's true from axioms

Desires (goals):

- `!reach(Destination)` — an **achievement** goal (i.e. something to do)
  - commonly, via actions
- `?discover(Information)` — a **test** goal (i.e. some information to be acquired)
  - commonly, via reasoning, or actions

Jason: syntax overview (pt. 2)

Events:

- `+Belief` — some new **belief** / message / percept has been **added**
- `-Belief` — some **belief** / message / percept has been **removed**
- `+Goal` — some new **goal** has been **added**
- `-Goal` — some **goal** has **failed** (cannot be reached)
  - because of: lack of plans, or failure in action, etc.

Jason: syntax overview (pt. 3)

Actions:

- `move(Direction)` — **external action** (involving the environment)
- `.send(Message)` — **internal action** (involving the agent)
- `!Goal` — pursue **achievement** goal as sub-goal
- `?Goal` — pursue **test** goal as sub-goal
- `+Belief` — **add** belief
- `-Belief` — **remove** belief
- `-+Belief` — **update** belief

Jason: syntax overview (pt. 4)

Plans:

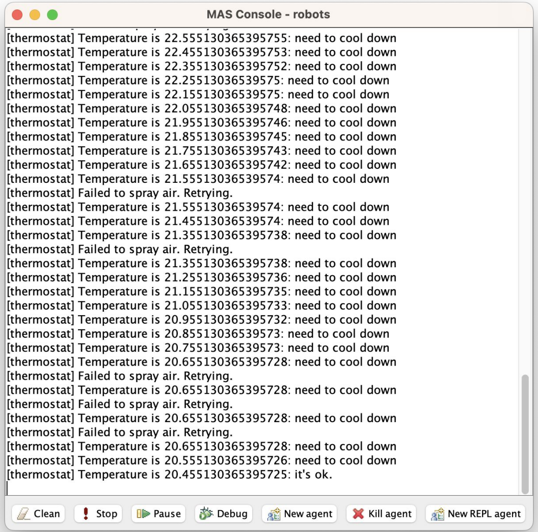
- `Event : Guard <- Action1; ...; ActionN.`
  - actions to be executed in reaction to `Event`, if `Guard` is true
- `Event <- Action1; ...; ActionN.`
  - lacking `Guard` = no restrictions

Example: thermostat agent, in Jason

```
target(20).
+temperature(X) <- !regulate_temperature(X).

+!regulate_temperature(X) : target(Y) & (X - Y > 0.5) <-
   .print("Temperature is ", X, ": need to cool down");
   spray_air(cold).

+!regulate_temperature(X) : target(Y) & (Y - X > 0.5) <-
   .print("Temperature is ", X, ": need to warm up");
   spray_air(hot).

+!regulate_temperature(X) : target(Y) & (Z = X - Y) & (Z >= -0.5) & (Z <= 0.5) <-
   .print("Temperature is ", X, ": it's ok.").

-!regulate_temperature(X) <- .print("Failed to spray air. Retrying."); !regulate_temperature(X).
```

Example: thermostat agent’s environment, in Java

```java
public class TemperatureEnvironment extends Environment {
    private static final Random RAND = new Random();
    public static final Literal hotAir = Literal.parseLiteral("spray_air(hot)");
    public static final Literal coldAir = Literal.parseLiteral("spray_air(cold)");
    private double temperature;
    private static final double FAILURE_PROBABILITY = 0.2;

    public boolean executeAction(final String ag, final Structure action) {
        boolean result = true;
        if (RAND.nextDouble() < FAILURE_PROBABILITY) {
            result = false;
        } else if (action.equals(hotAir)) {
            temperature += 0.1;
        } else if (action.equals(coldAir)) {
            temperature -= 0.1;
        }
        return result;
    }
}
```



Communication in Jason: `.send` internal action

`.send(`**`ReceiverID, ILF, Message`**` [, `**`Answer, Timeout`**` ])`

- **`ReceiverID`** is the name of receiver agent
- **`ILF`** is the illocutionary force of the message
- **`Message`** is the payload the message
- **`Answer`** the answer of an ask message, provided by the receiver
- **`Timeout`** is the timeout (in milliseconds) when waiting for an `ask` answer

Illocutionary what?!

Let S be the sender and R be the receiver of `.send(R, ILF, Message)`, then `ILF` =

- `tell` means S intends R to **believe** `Message` is true
- `untell` means S intends R to **believe** `Message` is **not** true
- `achieve` means S requests R to **achieve goal** `!Message`
- `unachieve` means S requests R to **drop** the **goal** `!Message`
- `askOne` means S requests R to **test** `?Message` **once**
- `askAll` means S requests R to **test all** answers for `?Message`
- `tellHow` means S transfers **plan** `Message` to R
- `untellHow` means S wants R to **forget** about the **plan** named `Message`
- `askHow` means S requests R to **provide all** its **plans** for goal `Message`

Complex example: domotic robot

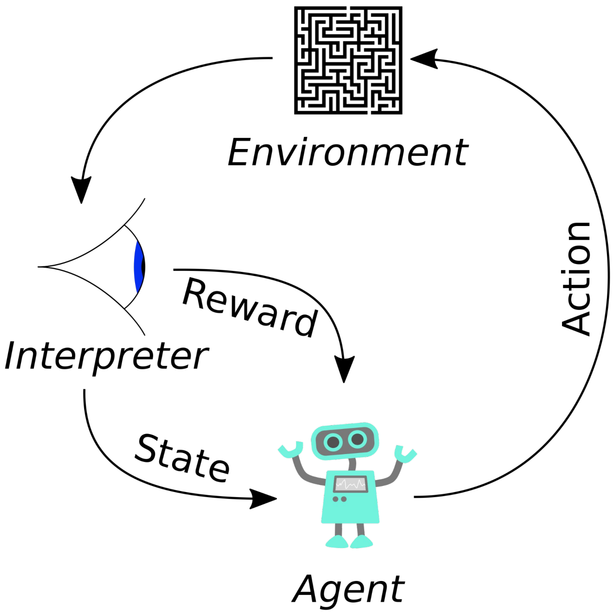
- Example code: here
- How to run:
  - `git clone https://github.com/pikalab-unibo/ise-lab-code-jason`
  - `cd ise-lab-code-jason`
  - `./gradlew runDomoticMas`

Cognitive agents, what is missing?

- In a nutshell: imagination & imitation
  - capability to invent goals
  - capability to generate beliefs
  - capability to figure out which actions are possible
  - capability to learn by observing others

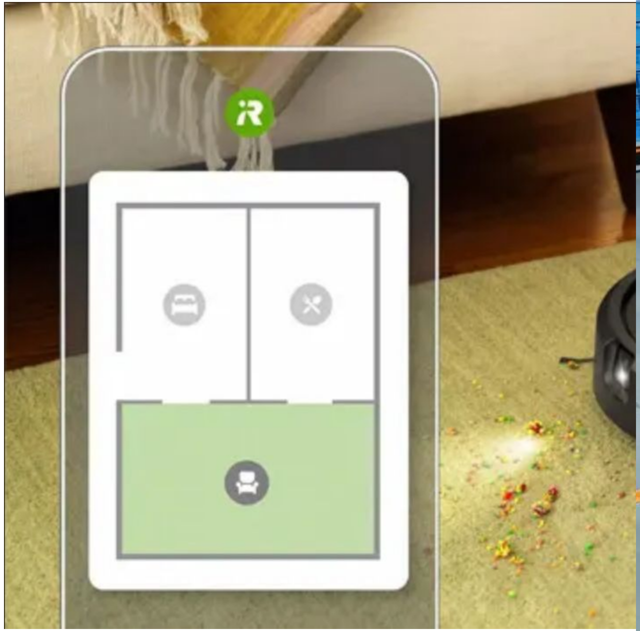
Where are all the agents?

Still a fundamental notion in *reinforcement learning*

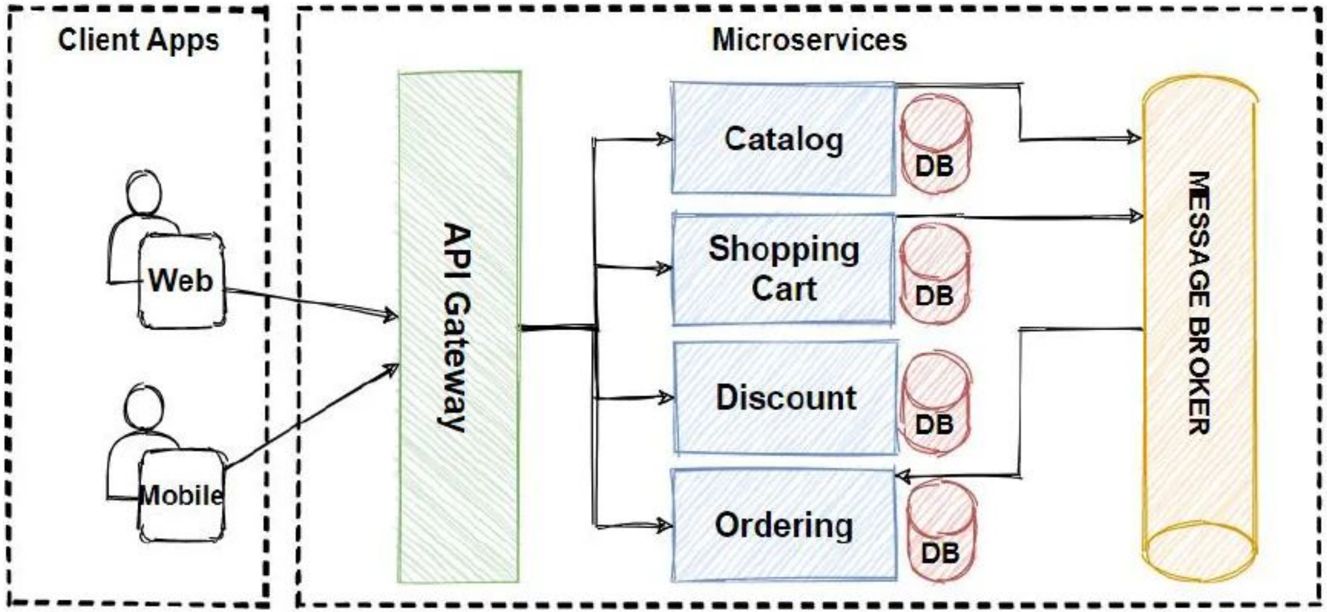


	actions			
states	a_0	a_1	a_2	...
s_0	$Q(s_0,a_0)$	$Q(s_0,a_1)$	$Q(s_0,a_2)$	...
s_1	$Q(s_1,a_0)$	$Q(s_1,a_1)$	$Q(s_1,a_2)$	...
s_2	$Q(s_2,a_0)$	$Q(s_2,a_1)$	$Q(s_2,a_2)$	...
⋮	⋮	⋮	⋮	⋮

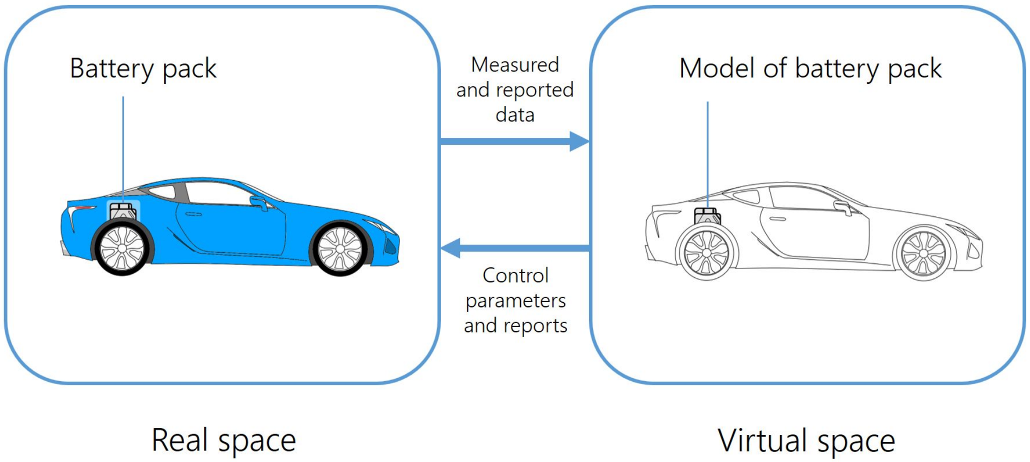
Underlying metaphor in many industrial applications



Microservices are essentially reactive agents



Digital twins? Agents mapping physical entities



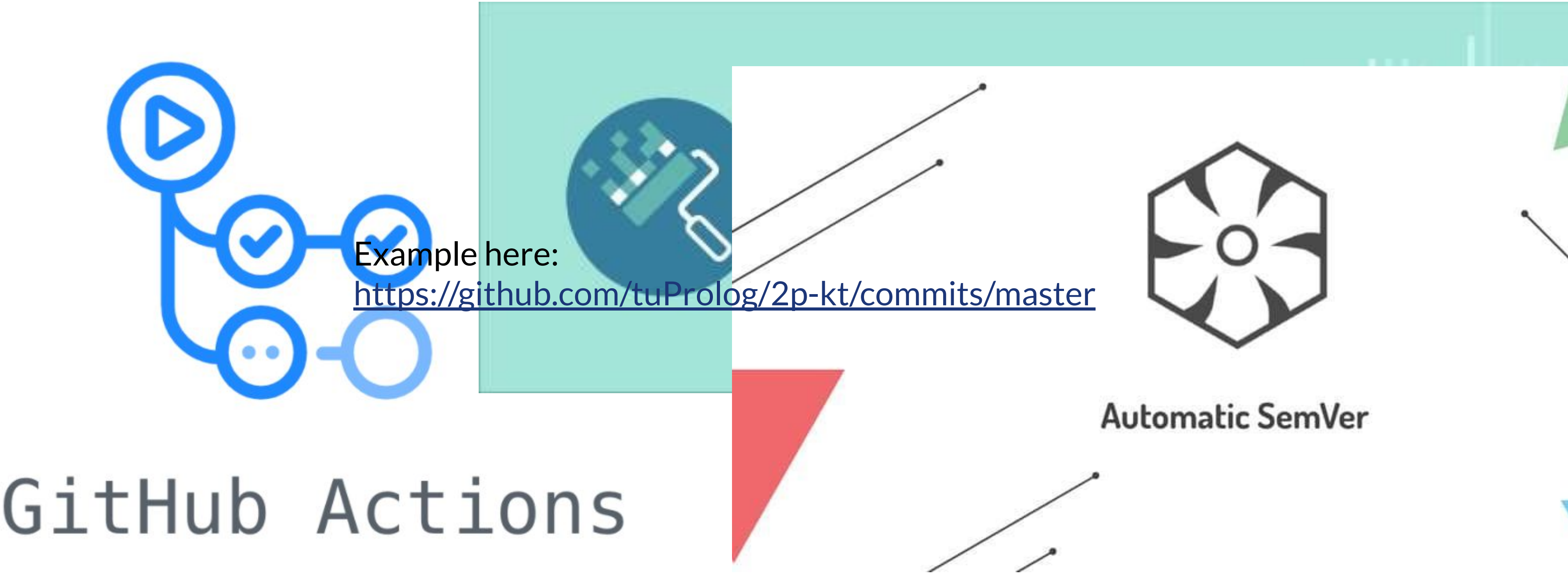
One can buy “AI bots” for stock or crypto trading

Strategy Trades (Holly Grail) - History: 26/08/2020 to 30/10/2020

Conservative Profit 16.99% Filter None Strategy All Str

L/S	Strategy	Symbol	Lst Price	Shares	Aggrssv Pft	Aggrssv Pft %	Moderat Pft	Moderat Pft %	Consrv Pft	Consrv Pft %	Pft Chg Lst 5	Consrv Exit Reason
L	Bullish Pullback	SPCE	23.06	100	$-60.04	-2.54%	$-60.04	-2.54%			$-1.00	Profit Save
L	Buyers Stepping In	CBLI	3.19	100	$-57.00	-15.16%	$-14.00	-3.72%	$94.50	25.13%	$4.00	Profit Save
L	Nice Chart	NLS	26.80	100	$97.50	3.78%	$97.50	3.78%			$-20.00	Profit Save
S	Bon Shorty	CBLI	3.19	100	$159.00	33.26%	$-22.00	-4.60%	$-22.00	-4.60%	$-4.00	Stop Hit
S	Knocking on Resistance	DKNG	44.80	100	$-36.10	-0.81%	$-36.10	-0.81%	$25.00	0.56%	$-21.00	Profit Save
S	The Anvil	MTCH	113.02	100	$63.00	0.55%	$-108.00	-0.95%	$-108.00	-0.95%	$-1.00	Stop Hit
L	Looking for Bounce	CEQP	14.31	100	$-16.00	-1.11%	$-20.15	-1.39%	$-17.00	-1.17%	$-6.00	Reduce Risk
L	Tailwind	CHWY	67.18	100	$-94.00	-1.38%	$-94.00	-1.38%	$-50.00	-0.73%	$-8.00	Reduce Risk
L	Tailwind	TRUP	92.49	100	$-97.00	-1.04%	$-129.00	-1.38%	$-101.00	-1.08%	$-7.00	Reduce Risk
L	Tailwind	CCK	83.29	100	$-30.00	-0.36%	$-30.00	-0.36%	$18.00	0.22%	$-3.00	Profit Save
L	Strong Stock Pulling Back	JKS	70.00	100	$93.50	1.35%	$93.50	1.35%			$-13.50	Profit Save
L	Pushing Through Resistance	SPCE	22.40	100	$-96.00	-4.11%	$-96.00	-4.11%	$-42.91	-1.84%	$2.00	Reduce Risk
L	Looking for Bounce	USFD	24.84	100	$-15.00	-0.60%	$-15.00	-0.60%	$15.25	0.61%	$-6.50	Profit Save
L	On Support	DHI	77.03	100	$-241.00	-3.03%	$-75.00	-0.94%	$-1.30	-0.02%	$-12.00	Profit Save
S	Knocking on Resistance	TXT	36.35	100	$-76.50	-2.15%	$-25.00	-0.70%	$-25.00	-0.70%	$7.00	Stop Hit
S	The Anvil	OMCL	81.64	100	$41.00	0.50%	$-31.00	-0.38%	$-31.00	-0.38%	$24.00	Stop Hit
S	The Anvil	HON	173.52	100	$73.00	0.42%	$73.00	0.42%	$75.00	0.43%	$49.00	Profit Save
S	Downward Dog	NETE	8.83	100	$-49.00	-5.88%	$-36.00	-4.32%	$20.76	2.49%	$-3.00	Profit Save
L	Got Dough Wants To Go	CYTK	16.51	100	$-100.69	-5.75%	$-74.69	-4.26%	$-74.69	-4.26%		Stop Hit
L	Power Hour Long	JKS	62.70	100	$-10.00	-0.16%	$-10.00	-0.16%	$-18.00	-0.29%	$18.00	Reduce Risk
L	Bullish Pullback	SMAR	55.46	100	$16.00	0.29%	$16.00	0.29%	$24.00	0.43%	$-3.00	Timed Exit

CI/CD bots for software development automation



Example here:
https://github.com/tuProlog/2p-kt/commits/master