

# Introduction to the Course

Autonomous Software Agents

A.A. 2024-2025

**Prof. Paolo Giorgini**

**Dr. Marco Robol**



UNIVERSITY OF TRENTO - Italy

Department of Information  
and Communication Technology

# Outline

- Autonomous Software
  - Agents as new paradigm for software systems development
  - Multi-agent systems
- Objectives of the course
- Structure of the course
- Exam / Project

# Billions of AI agents ?



# Autonomous Software Agent

- A software that can **operate autonomously** (without any human or other software control):
  - Wait a minute ! ... any software is like that, right?  
Yes, knowing in advance that `cond_1` and `battery_NotOK` are the only variables that can change the execution  
But, what about `obstacle_in_front`, should we change the software?  
Also, when `cond_1`, is it always true that `move_right` and `pick_up` are the best actions to do? Are there no alternatives?

```
while(run){  
    if(cond_1) {  
        move_right()  
        pick_up()  
    } else {  
        move_left()  
        mov_on()  
    }  
    if(battery_NotOK) {  
        run = False  
    }  
}
```

# Amazon warehouse (Video)

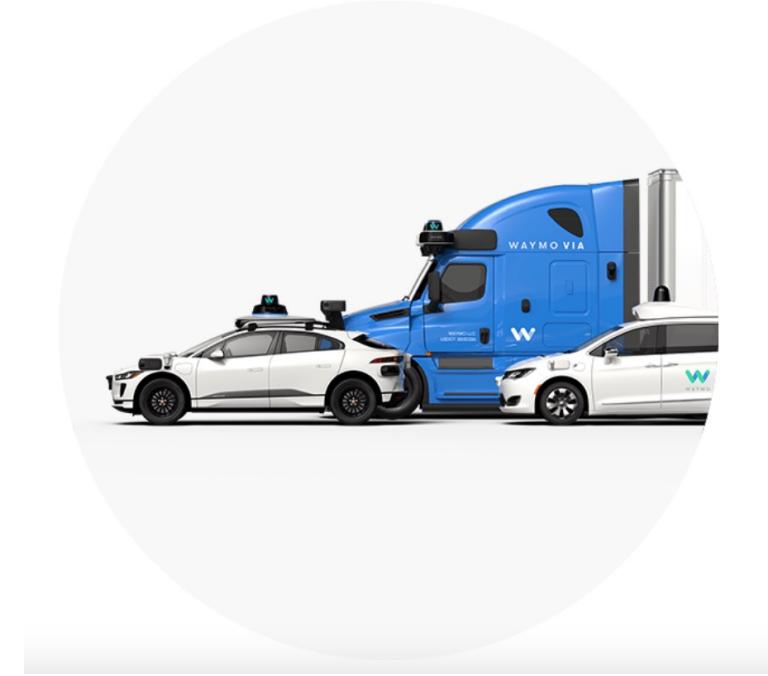
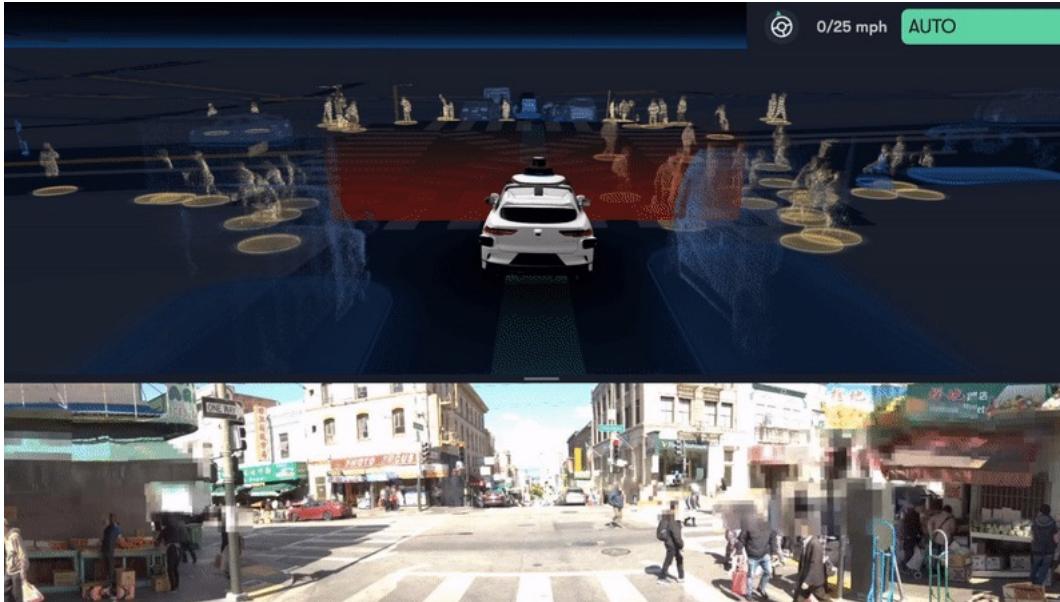


# Intelligence and delegation

- The complexity of tasks that we are capable of **automating** and **delegating** to computers has grown steadily
- Computers are doing more for us – without our intervention
- We are **giving control** to computers, even in safety critical tasks



# Google's Driverless Car – Waymo Driver



*How safe driver is your average robot? Safer than your average American, at least by one measure.*

- Since 2009, billions of miles driven (millions in public roads) over +13 US states.

# The 6 Levels of Vehicle Autonomy

- Level 0 (No Driving Automation)
  - The automated system issues warnings and may momentarily intervene but has no sustained vehicle control
- Level 1 (Driver Assistance)
  - The driver and the automated system share control of the vehicle
- Level 2 (Partial Driving Automation)
  - The automated system takes full control of the vehicle: accelerating, braking, and steering.
- **Level 3 (Conditional Driving Automation)**
  - The driver can safely turn their attention away from the driving tasks
- Level 4 (High Driving Automation)
  - As level 3, but no driver attention is ever required for safety
- Level 5 (Full Driving Automation)
  - No human intervention is required at all



January 23, 2025 Release

Research

Safety

ChatGPT

Sora

API Platform

For Business

Stories

Company

News



# Computer-Using Agent

Powering Operator with Computer-Using Agent, a universal interface for AI to interact with the digital world.

Go to Operator ↗

Operated for 17 seconds ^

Initializing

Navigating

Ask ChatGPT



# Computer-Using Agents

- An AI model designed to perform computer tasks using screenshots, cursor, and keyboard
- Combines GPT-4o's vision with advanced reasoning via reinforcement learning
  - Automates tasks like form-filling, travel booking, and content creation
  - Interacts with web browsers similar to human users
  - Pauses for user input on complex tasks (e.g., CAPTCHAs, passwords)
- Collaborative approach ensuring users can guide and intervene
  - Proactive refusals for high-risk actions
  - Confirmation prompts before critical tasks
  - Active monitoring systems to detect threats

# Where does it bring us?



- Delegation and Intelligence imply the need to build computer systems that can act effectively on our behalf
- This implies:
  - The ability of computer systems to act **independently**
  - The ability of computer systems to act in a way that **represents our best interests** while **interacting** with other humans or systems



# Programming progression...



Programming has progressed through:

- machine code;
- assembly language;
- machine-independent programming languages;
- sub-routines;
- procedures & functions;
- abstract data types;
- objects;
- services;

to **agents** and AI-based systems

# Human Orientation

- Movement away from machine-oriented views of programming toward concepts and metaphors that more closely reflect the way we ourselves understand the world
- Programmers (and users!) relate to the machine differently
- Programmers conceptualize and implement software in terms of **higher-level** – more human-oriented – **abstractions**
  - **Abstractions** to cope with complexity



# Agent, a first definition

An agent is a computer system that is capable of **independent** actions on behalf of its user or owner (figuring out what needs to be done to satisfy design objectives, rather than constantly being told)  
[Wooldridge&Jennigs]



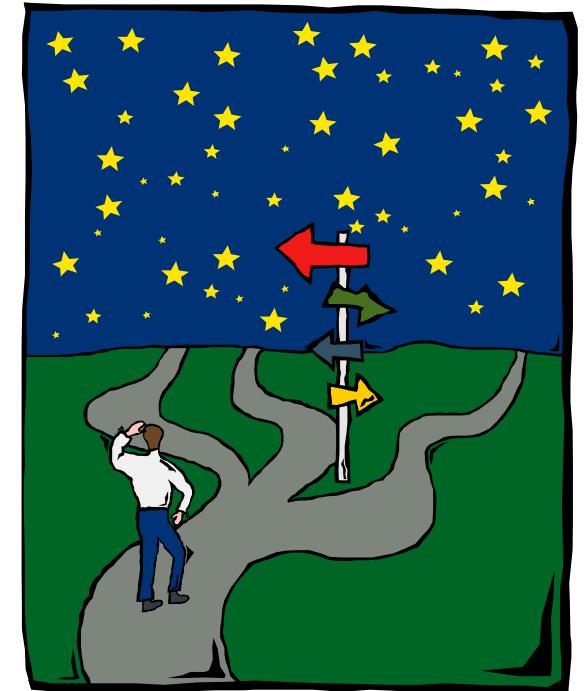
# Interconnection

- Computer systems today no longer stand alone, but are networked into **large distributed systems**
  - The internet is an obvious example, but networking is spreading its ever-growing tentacles...
- Since distributed and concurrent systems have become the norm, some researchers are putting forward theoretical models that portray computing as primarily a **process of interaction**



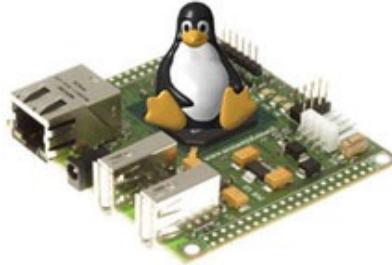
# Where we are ...

- Computing is now about **interaction** and **coordination** between **distinct piece of software**
- More and more **independent software** able to coordinate with others (including humans)
  - The final objective is obtained as an emergent behaviour of **multi-software behaviours**
  - **Open environment** (new piece of software can be added or removed)
  - **Multi-objective problem**



# Ubiquity

- The continual reduction in cost of computing capability has made it possible to introduce **processing power** into places and devices that would have once been uneconomic
- As processing capability spreads, sophistication (and intelligence of a sort) becomes **ubiquitous**



# Internet of Things

- The **Internet of Things** refers to uniquely identifiable objects (things) and their virtual representations in an Internet-like structure
  - If all objects of daily life were equipped with radio tags, they could be identified and inventoried by computers
- Ambient Intelligence (AI) and Autonomous Control (AC) are not part of the original concept of the Internet of Things
  - How to use the Internet of Things for AI and AC?



# Services, Cloud and Edge computing

- Services are made available to be used and combined together
  - Data storage, software as service, platform as service, infrastructure as service
  - Interaction between different systems is the key issue
  - Quality and security control problems
  - Services availability
  - Run-time composition
  - Flexibility and scalability issues
- Edge computing brings computation and data storage closer to the sources of data

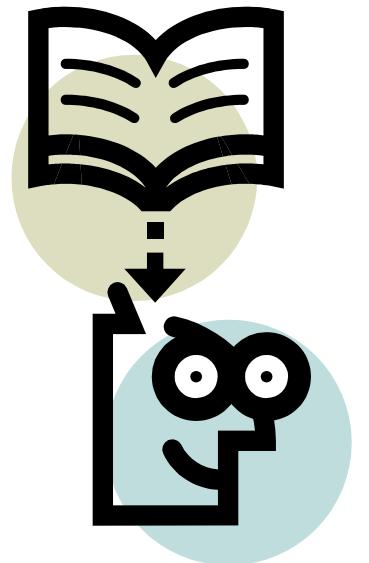


# Interconnection: smart city



# Interconnection and distribution

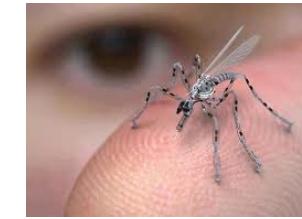
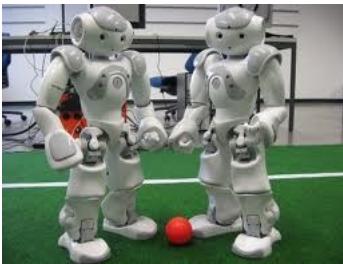
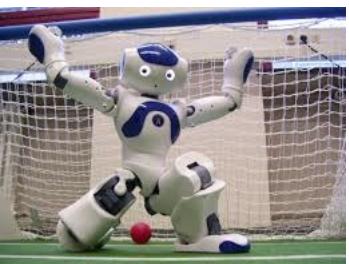
- Interconnection and distribution have become core motifs in Computer Science
- But Interconnection and distribution, coupled with the need for systems to represent our best interests, implies systems that can **cooperate** and **reach agreements** (or even **compete**) with other systems that have different interests (much as we do with other people)



# Examples



Robocup



Swarm intelligence



Distributed Sensing

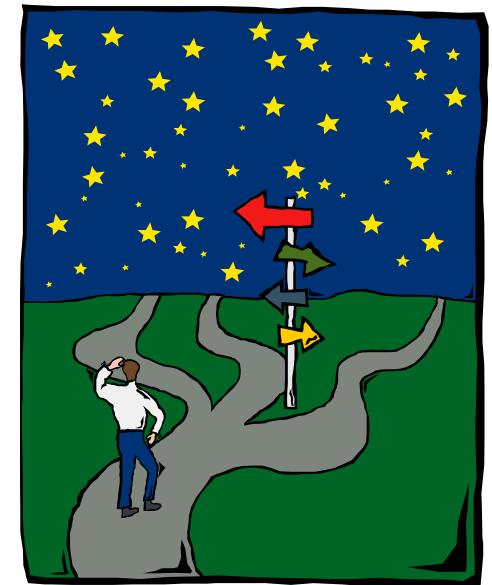


# Computer Science expands...

All of these trends have led to the emergence of a new field in Computer Science:

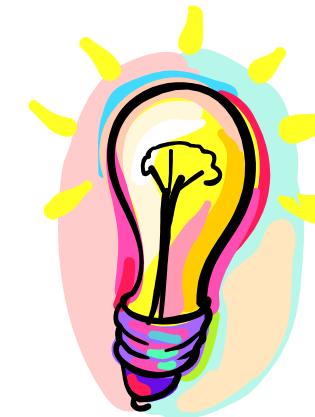
Agent Systems provide us with the means to design and implement interactive computing, whether between machines, or people, or both

[Michael Luck]



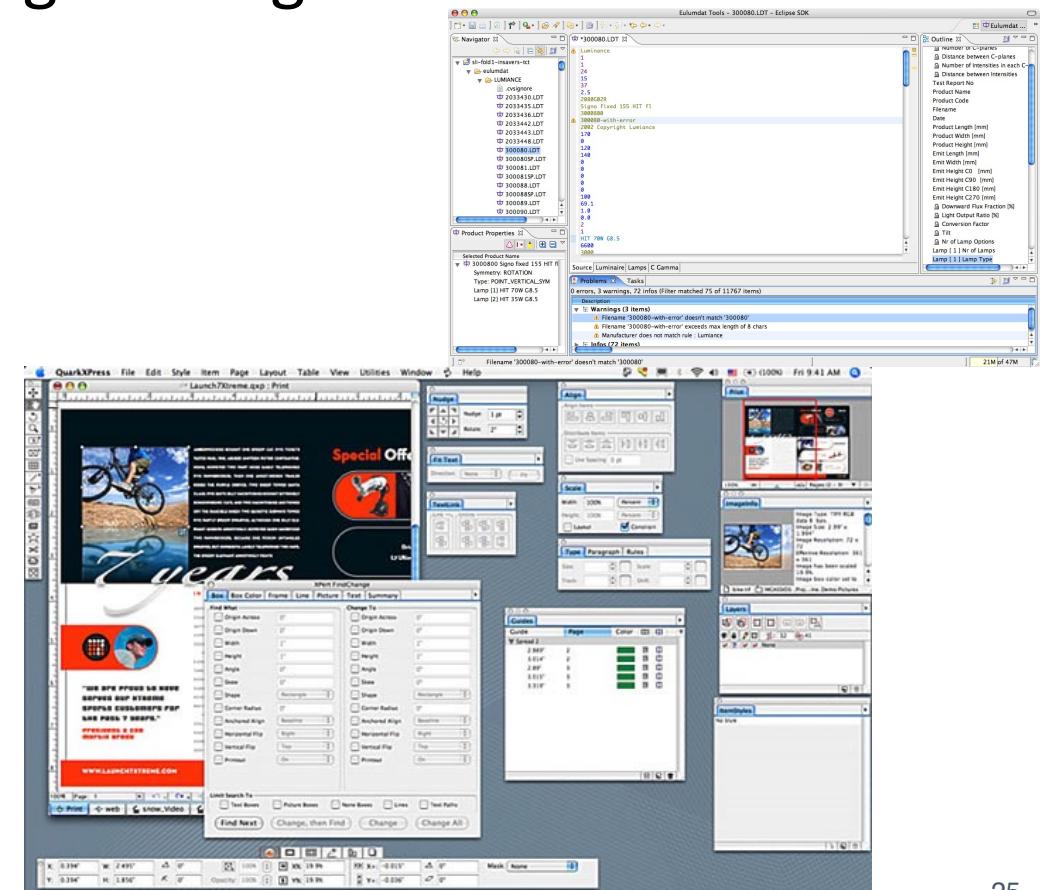
# Multi-Agent System, a definition

- A multi-agent system is one that consists of a number of agents, which **interact** with one-another
- In the most general case, agents will be **acting on behalf of users** with different **goals** and motivations
- To successfully interact, they will require the ability to **cooperate**, **coordinate**, and **negotiate** with each other, much as people do



# How can we develop such complex software?

- A good (**Agent-Oriented**) Software Engineering Methodology
    - Modeling languages
    - Analysis techniques
    - Design Techniques
    - Supporting Tools



# Objective of the Course

- The objective of the course is to examine and explore the credentials of **agent-based approaches** as a **software engineering paradigm**, and to gain an insight into what agent-oriented software engineering will look like
  - Agents as building blocks of a software system
  - Agent as basic concept to develop a software system
  - Knowledge level Software Engineering
    - Agent concept and its mentalist notions (e.g., goal, plan, and belief) are used along all the software engineering process
    - Agent as main concept to
      - model and analyse and organizational setting
      - design a system
      - implement a system ...

# Structure of the course

- Very practical course
  - You will learn how to develop both a single and a team of autonomous software systems
- Theory
  - We will explore concepts of autonomous software agents and multi-agent systems, software architectures, interaction and communication techniques, planning techniques
- Laboratory
  - Tutorial + project development

# Tentative Theory content



## Introduction

- Introduction to the agent paradigm and multi-agent systems
- Applications and problems

## Interaction and cooperation

- Types of interaction
- Forms of Cooperation
- Methods of cooperation
- Organization and cooperation

## Communication

- Aspects of communication
- Speech acts
- Conversation
- KQML/ACL

## Collaboration and distributed tasks

- Models of tasks allocation
- Centralized allocation
- Decentralized allocation
- Emergent allocation

## The BDI Architecture

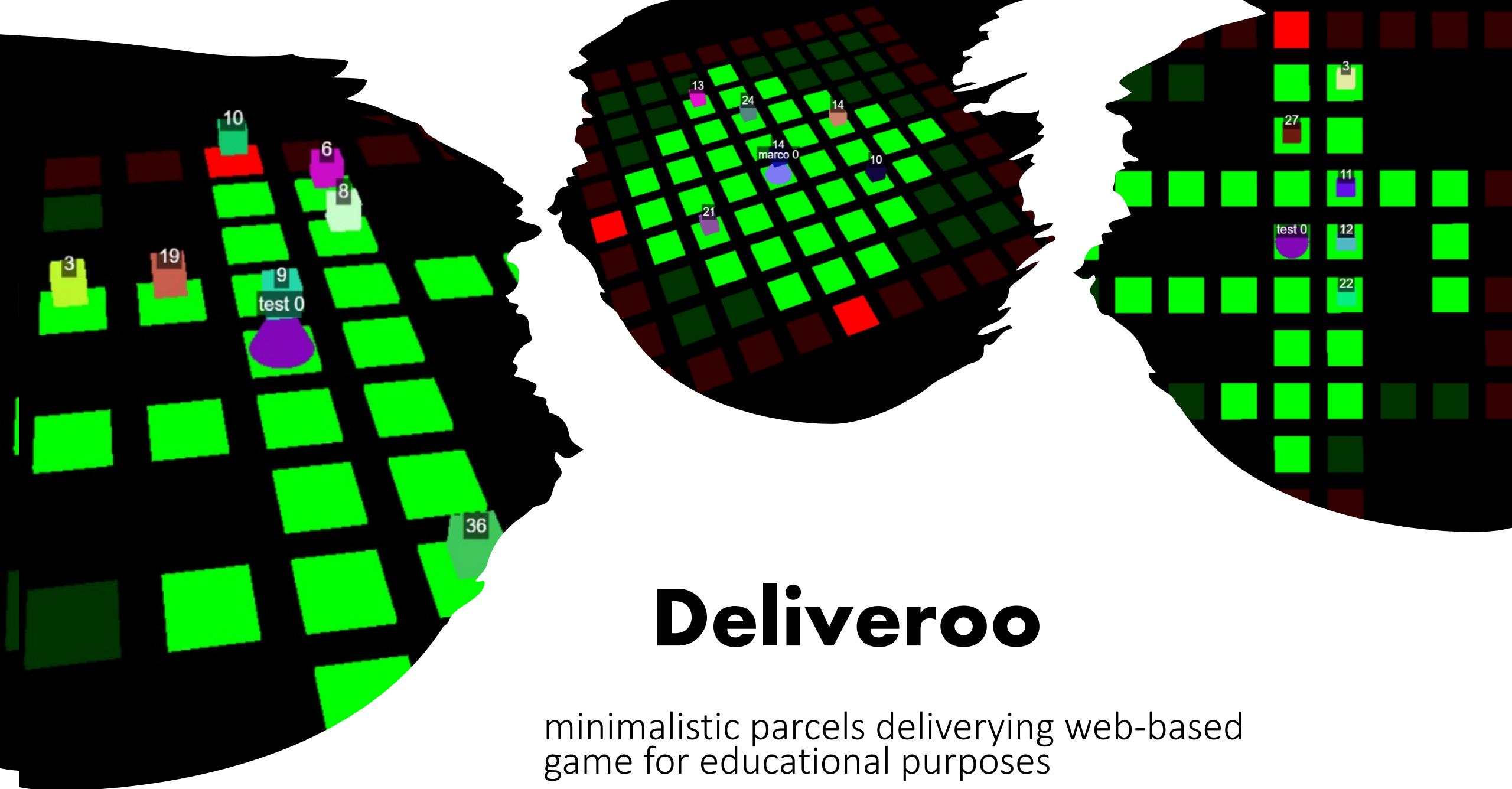
- The agent control loop
- The deliberation process

## Planning

- Exploring the space of alternatives
- Planning in the agent control loop

## Agent-Oriented Methodologies

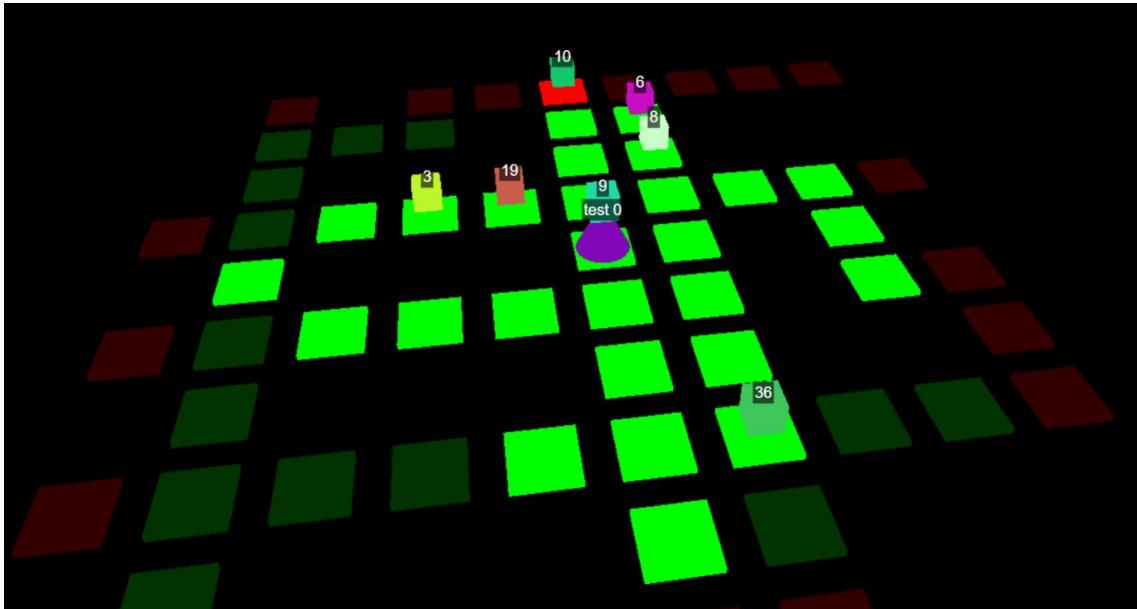
## Agent-oriented development tools



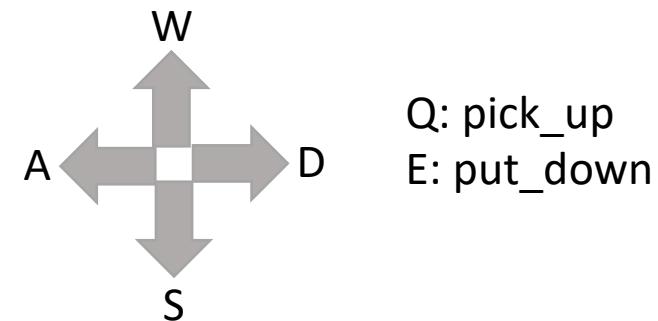
# Deliveroo

minimalistic parcels delivering web-based  
game for educational purposes

# Deliveroo – try it!

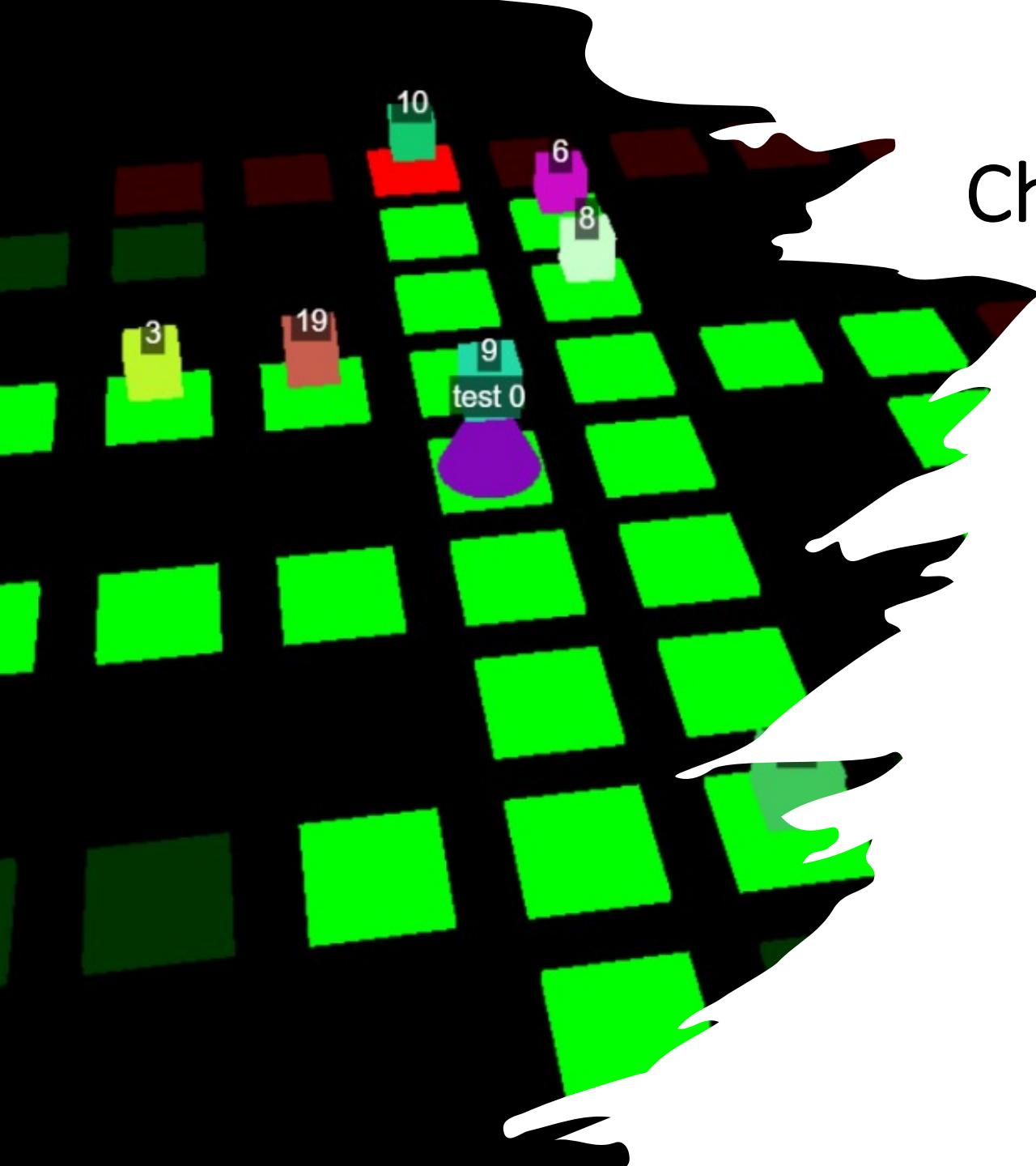


- <https://deliveroojs.onrender.com/>



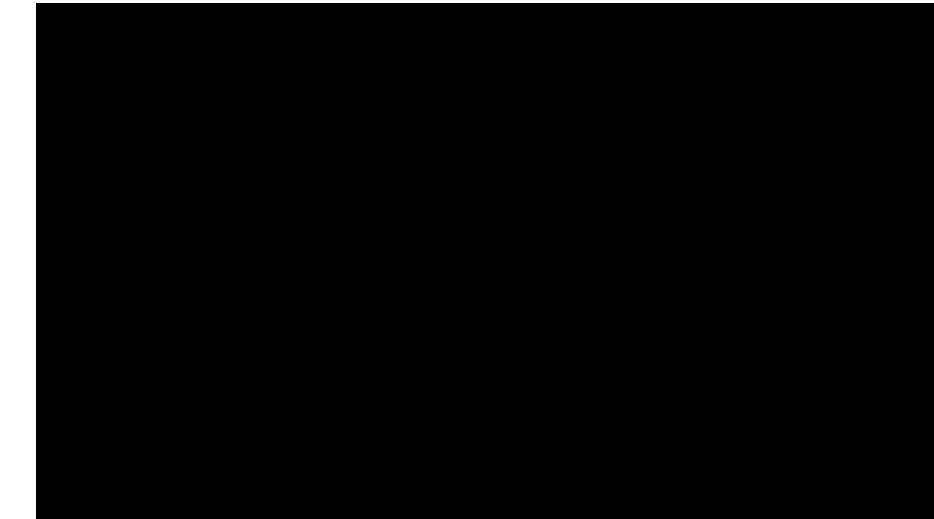
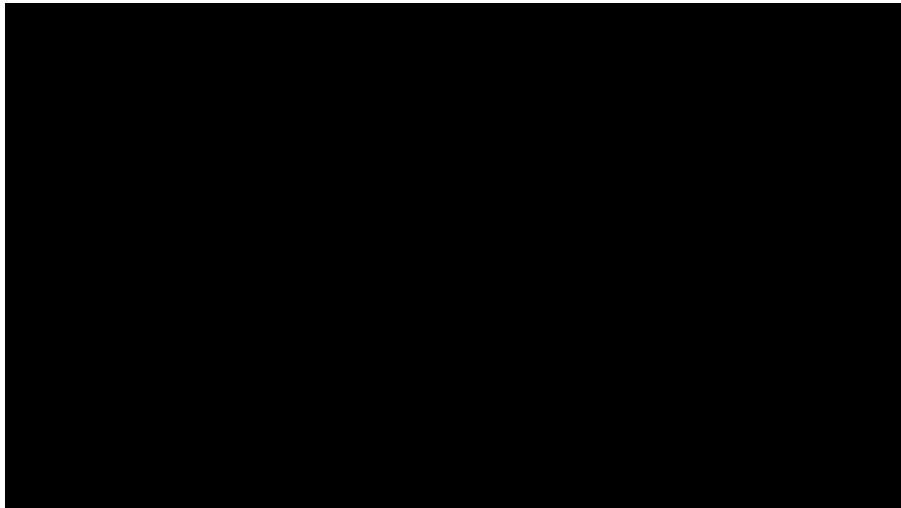
# The objective of the project

- Develop an autonomous software that will play on your behalf
  - earning points by collecting as many parcels as possible and delivering them in the delivery zone
  - Using a BDI architecture
    - Sensing the environment and managing Beliefs
    - Deliberating Intentions
    - Select plans from a library
    - Using an external planner component
    - Execute a plan (actions)
    - Defining strategies and behaviours
    - Replanning and redeliberating



# Challenges and tournaments

- In order to validate your ASA we will organize tournaments where each group will challenge the other groups.
  - Single agent tournament
  - Multi-agent tournament
  - Different level of complexity



# Laboratory

- Hands-on with exercises
- Node.js – javascript
- Planning
  - PDDL
- Project development

→ (Take your laptop with you)



# Exam and prerequisites

**Project:** a scenario will be provided in a couple of weeks

- Groups of 2 students
- First part: single agent
- Second part: team of two collaborative agents
- **Exam**
  - Deliverable
    - **Code + Report** (max 10 pages)
  - Project presentation
- **Prerequisite for the course:** experience in programming
  - Better if you know Javascript (we will provide a Tutorial)



# Books and Material



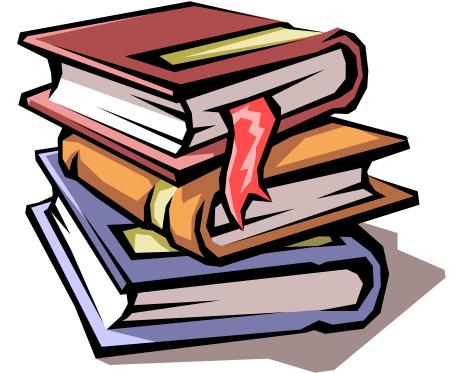
- Given the fact we will try and provide a broad coverage of various topics seems unrealistic an approach in which all the textbooks are adopted.

Therefore:

- We encourage the use of slides, papers, (on-line) material, etc.
- We provide a list of textbooks of which only parts are needed (no need to get all of them, though!)

# Books...

- *Jacques Ferber. Multi-Agent System: An Introduction to Distributed Artificial Intelligence*, Addison Wesley Longman, 1999
- *Michael Wooldridge. An Introduction to Multiagent Systems.* John Wiley & Sons, 2002
- *Paolo Giorgini and Brian Henderson-Sellers. Agent-oriented Methodologies*, Idea Group Inc, 2004.
- *Fabio Bellifemine, Giovanni Caire, Dominic Greenwood. Developing Multi-Agent Systems with JADE.* John Wiley & Son Ltd, 2007
- *G. Weiss. Multiagent systems*, (second edition). MIT Press, 2013.



# Next lectures

- Tuesday 11.30 -13.30 (A110)
- Wednesday 9.30-11.30 (A110)

Lectures will be in presence, but we will stream (zoom) and record them

All info and material is Moodle

- Zoom link for the lectures
- Slides + Video
- ... and much more

# Contacts

Paolo Giorgini - - > [pao.lo.giorgini@unitn.it](mailto:pao.lo.giorgini@unitn.it)

Marco Robol - - > [marco.robol@unitn.it](mailto:marco.robol@unitn.it)



Q/A

# Introduction Agents and Multi-Agent Systems

Autonomous Software Agents

A.A. 2024-2025

**Prof. Paolo Giorgini**

**Dr. Marco Robol**



UNIVERSITY OF TRENTO - Italy

Department of Information  
and Communication Technology

What is a (software) agent?

# New perspective

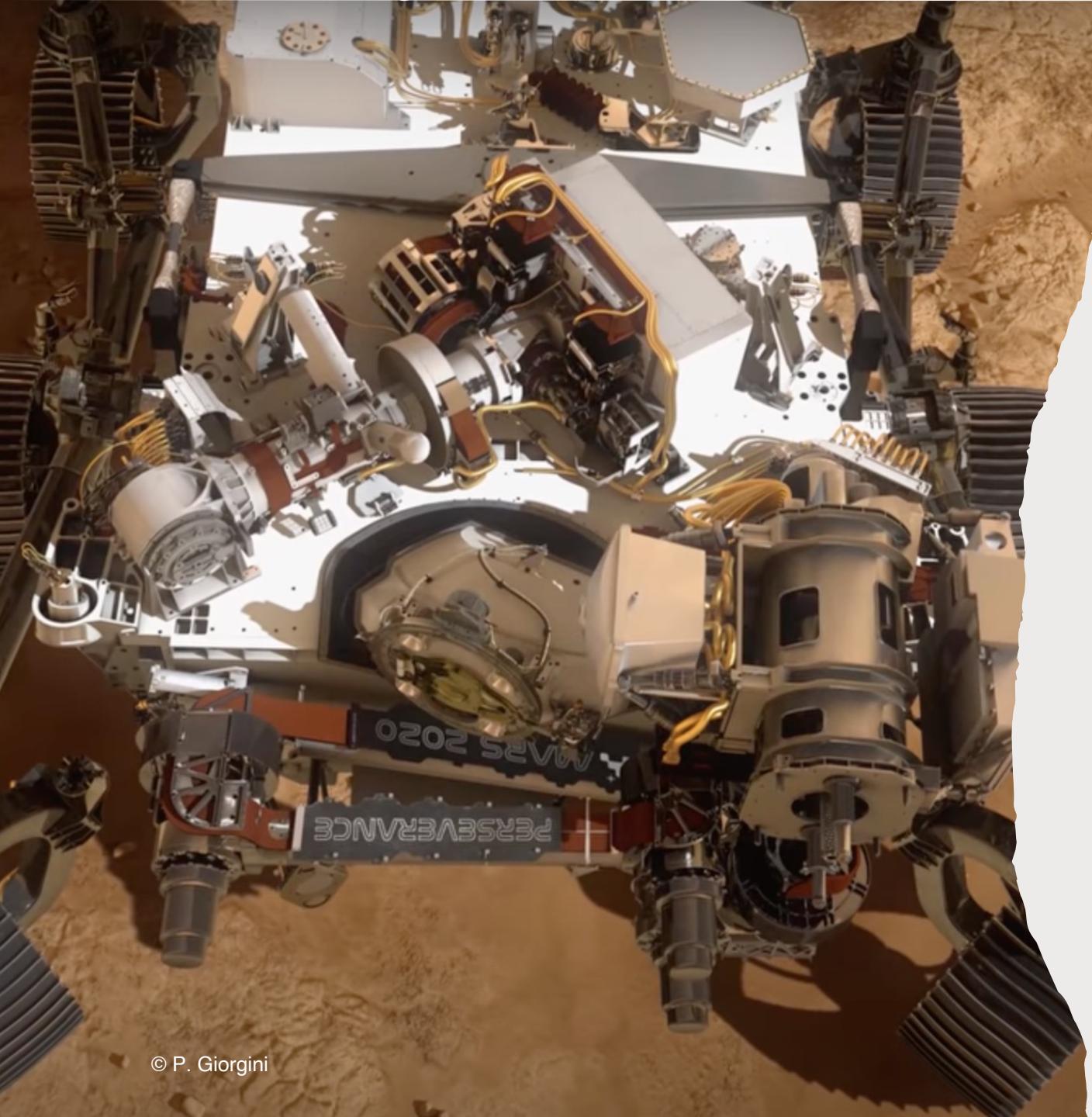
In Java to define the goal in the Object: Agent, we need to specify a specific state of the environment  
↳ In literature GOAL = specific state of the environment

↳ HOW TO DO THIS? translate of what we are sensing in a set of variable, when the configuration of this variables are the same of the set of variables that represents the GOAL, we have achieve the goal

- Computers are not very good at knowing what to do
  - Every action a computer performs must be explicitly anticipated, planned for, and coded by a programmer
- They fail when they encounter a situation that their designers did not anticipate
  - they crash and may cause system crash and loss of life at worst
- Growing interest in developing software that may decide for themselves what they need to do to achieve their objectives
  - able to change their behavior in rapidly changing, unpredictable, or open environments

An important thing for an agent is to find the best representation of the environment.

GOAL-ORIENTED BEHAVIOR OF THE AGENT

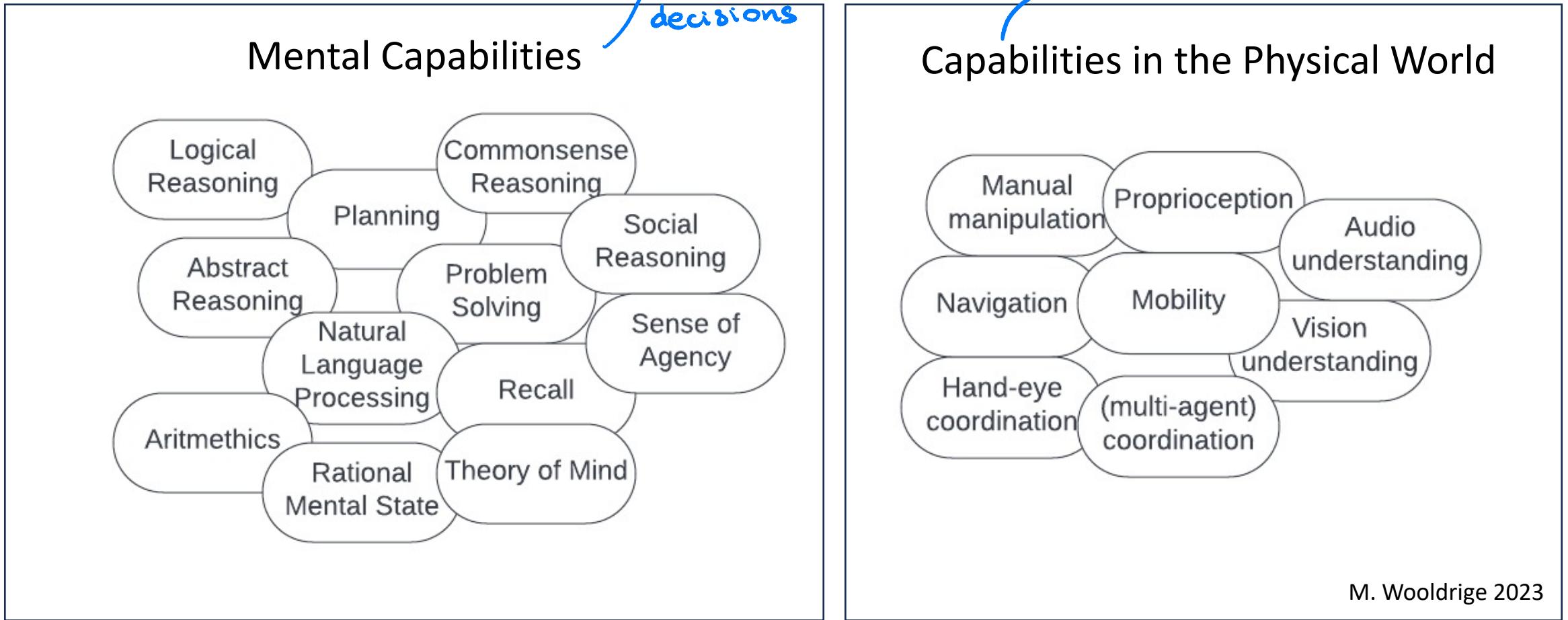


# Examples

- Space probe making its long flight to outer planets
  - Ground crew is required to continually track its progress and decide to deal with unexpected eventualities
  - Very costly and if decisions are required quickly, it is simply not practicable
  - NASA and ESA are interested in the possibility of making probes more autonomous (richer onboard decision-making capabilities and responsibilities)

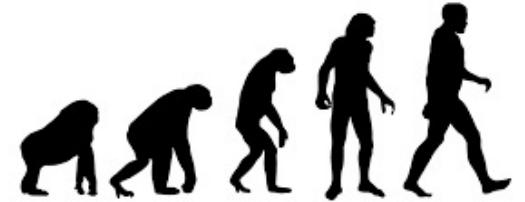
How we can embed all these capabilities in an agent?

# Dimensions of AI



understand which capabilities are important for the system you are developing

# Human Orientation and delegation



- Software systems as human models
  - Software architectures able to conceptualize and implement software in terms of higher-level – more human-oriented – abstractions
    - The object abstraction is not enough
- Delegation and control
  - Unpredictable behavior – with limitations
  - Emergent intelligence and collective behavior
- Socio-technical systems
  - Technical components and human actors are part of the actual system and its design – no conceptual distinction between them
  - Strong interdependency between human actors and software components

# Development is getting harder

Difference between ADAPTATION and EVOLUTION

↓  
change behaviour  
in order to the  
changing of  
environment

↓  
change of  
requirements  
during execution

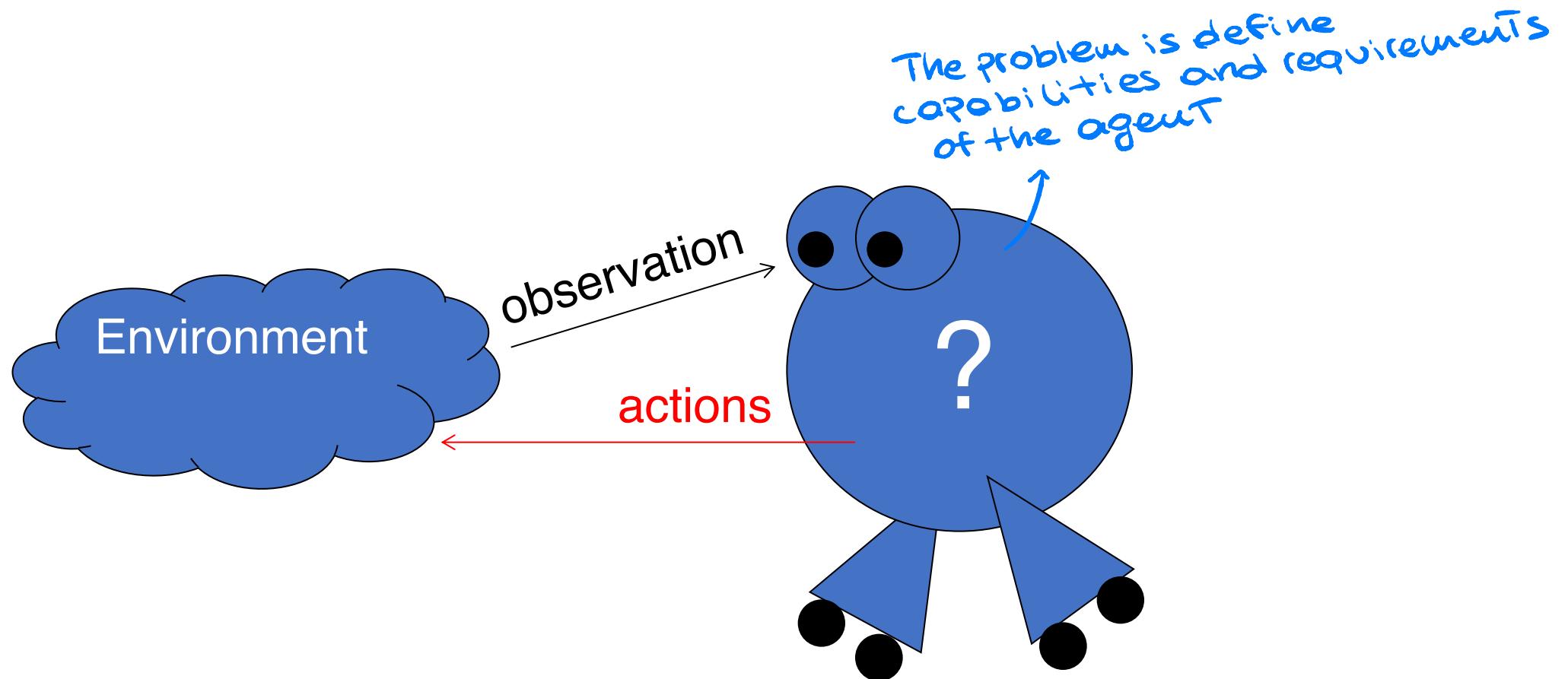
- Very ambitious requirements
  - E.g., Intelligent and human-like software
- Less certain requirements
  - Greater scope for change
  - More interdependent requirements
  - Socio and Organizational constraints
- More challenging environments
  - Greater dynamism
  - Greater openness
- Lack of appropriate design models



# Agent? What's?

- Very controversial definitions for the agent concept
  - "...encapsulated computer system, **situated** in some environment, and capable of **flexible autonomous** action in that environment in order to meet its design objectives" [Wooldridge]
    - **autonomous**: control over internal state and over own behaviour
    - **situated**: experiences environment through sensors and acts through effectors  
*↳ Even when we don't know in advance the environment*
    - **flexible**
      - **reactive**: respond in timely fashion to environmental change
      - **proactive**: act in anticipation of future goals

# An agent its environment



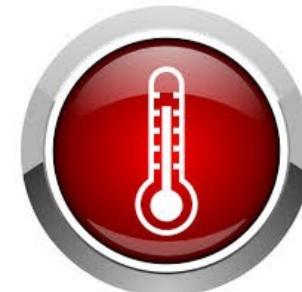
# An example of agent

(neither autonomous or intelligent)

- Any **control system** can be viewed as an agent
  - Thermostat has a sensor for detecting the room temperature
  - Temperature is too low or OK (two possible values)
  - The thermostat can perform `heating_on` or `heating_off`
  - Of course, `heating_on` action cannot guarantee to increase the temperature (partial control)

too cold → `heating_on`

temperature OK → `heating_off`



- What about if windows are open?
  - I should know in advance that there are windows in the room (or doors) that if open can decrease the temperature

- But, what if we don't know about that in advance? Can the software **learn it?**

Learning is very important  
and can be done at different  
levels

???

```
if(temperature==too_cold){  
    heating_on()  
} else {  
    heating_off()  
}
```

```
if(temperature==too_cold && windows_closed){  
    heating_on()  
} else {  
    heating_off()  
}
```

```
if(cond){  
    heating_on()  
} else {  
    heating_off()  
}
```

# Making it a bit more complex

- Thermostat + Automated Windows



Too cold → heating\_on  
Temperature OK → heating\_off



Stale air → open\_windows  
Air ok → close\_windows

```
// Thermostat agent
if(temperature==too_cold){
    heating_on()
} else {
    heating_off()
}

// Windows agent
if(Stale_air) {
    open_windows()
} else {
    close_windows()
}
```

- Does it make sense to keep the heater on when windows are open?
  - Thermostat and Windows should be coordinated

# Solution?

The two agents need to interact and communicate in order to coordinate

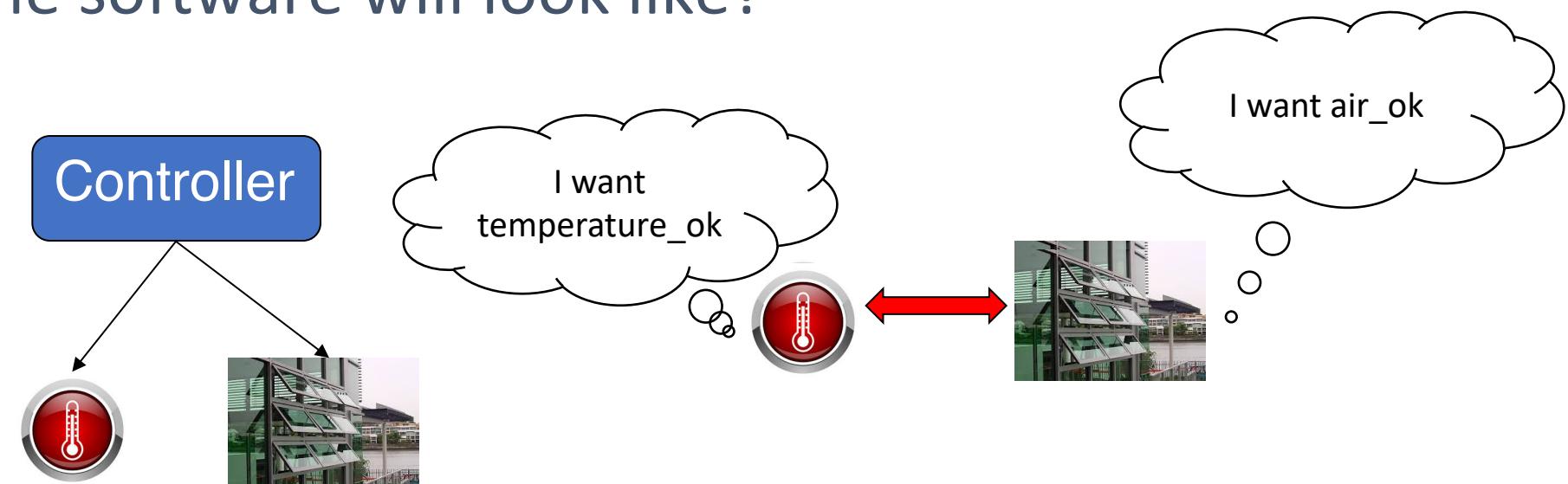
too\_cold ^ windows\_closed → heating\_on

Temperature\_OK → heating\_off

stale\_air ^ heating\_off -> → open\_windows

Air\_ok -> close\_windows

How does the software will look like?



# Consider also this

Can you close the window?

why is she asking me so?

window\_closed → temp\_ok

window\_closed → noise\_ok

Interpretation of an action can give us new knowledge



Could she want to increase the temperature, or is there too much noise in the room?

→ Find the reason behind an action

Abduction - explaining the request

she wants to make true temp\_ok (this means NOT temp\_ok) or she wants to make true noise\_ok (this means NOT noise\_ok)

# Consider also this

Can you close the window?

why is she asking me so?

window\_closed → temp\_ok

window\_closed → noise\_ok



Could she want to increase the temperature,  
or is there too much noise in the room?

*formal reasoning  
in order to be proactive*

NOT temp\_OK → feel\_cold

NOT noise\_ok → NOT follow\_the\_lecture

wearing\_coat → NOT feel\_cold

She is wearing a coat (NOT fell\_cold) so she is asking because she wants follow\_the\_lecture

# In another form

- Receive a request for an action **a**
- Find the **reason** of why of such a request
  - if I Do(**a**)  $\rightarrow$  **b** && **c** will be true
  - this means  $!b \mid\mid !c$
  - $!b \rightarrow d$
  - $!c \rightarrow !e$
  - $g \rightarrow !d$
  - If I know **g** is true ( $!d$ )  $\rightarrow$  she wants to make **e** true
- So what?
  - If she wants to follow the lecture (**e**), is there any “better” way to make this true?

<b>a</b> : close the window
<b>b</b> : temperature is ok
<b>c</b> : noise is ok
<b>d</b> : feel cold
<b>e</b> : follow the lecture
<b>g</b> : wearing a coat

# How to develop an agent like that?

- We should provide knowledge to the agent
  - $\text{Do}(a) \rightarrow b \And c$  expected consequences of an action
  - If we want  $b \And c$  it means  $\neg b \Or \neg c$  -----  
knowledge about the world
  - $\neg b \rightarrow d$
  - $\neg c \rightarrow \neg e$
  - $\neg g \rightarrow \neg d$
  - $g$  -----  
the current state of the world
- And mechanisms to reasoning about that
  - $\neg a \rightarrow e$  producing other knowledge
- Use already known knowledge and produced knowledge to
  - Explore alternative
  - Take a decision

# Flexibility

```
if(obstacle) {  
    avoid(obstacle)  
} else {  
    g = revise_goals()  
    p = plan(g)  
}
```

An agent can act **reactively** as well as **proactively**

- **Reactive** means that the agent reacts in **reasonable time** and in an appropriate way to changes in its environment and to changes in the requirements placed on it
- **Proactive** means that the agent acts with prediction, planning and goal orientation

**Flexibility**, consisting of reactivity and proactivity, is thus the capability to handle possibly unexpected events and simultaneously to act with planning and goal orientation

# Smart Home Assistant

## An example of proactivity

Category	Goal	Proactive Action
<b>Goal-Oriented Actions</b>	Optimize energy usage and enhance user comfort	Adjusts the thermostat based on weather forecasts and the homeowner's daily routine, ensuring the house is warm before the user arrives.
<b>Anticipating User Needs</b>	Improve home security	Automatically locks doors and turns on security lights if no one is home and it's getting dark.
<b>Planning and Scheduling</b>	Reduce energy consumption	Plans to run the washing machine during off-peak electricity hours without user input.
<b>Adapting to Environment</b>	Maintain user comfort	Activates air purifiers if air quality sensors detect poor air inside the home.

# Interactivity

- An agent can **interact** with its environment – especially with human actors and with other agents
- Such interaction can be on a very high level (i.e., they can be markedly communication and knowledge intensive) and they serve the purpose of coordination with third parties, i.e., the coordination of activities and the handling of mutual dependencies
  - Coordination: cooperation / competition
    - E.g., negotiation and conflict resolution in the realm of cooperative planning activities and competitive sales processes
  - Interactivity requires a **precise interface** that normally overshadows all the internals of the agent. Thus, in general interactivity designates all the (higher) social (communicative, cooperative and competitive) capabilities of an agent

## Smart Home Assistant - Interaction

Aspect	Example in Smart Home
<b>Agents Involved</b>	Smart Thermostat, Smart Lighting, Energy Management Agent, User
<b>High-Level Interaction</b>	Energy Management Agent requests 30% energy reduction; Thermostat and Lighting propose actions.
<b>Coordination (Cooperation)</b>	Thermostat and Lighting collaborate to reduce energy while maintaining user comfort.
<b>Coordination (Competition)</b>	Thermostat and Lighting compete for energy credits by optimizing their energy-saving strategies.
<b>Negotiation &amp; Conflict Resolution</b>	User increases heating; Energy Agent negotiates with Thermostat to adjust heating later.
<b>Importance of Interfaces</b>	Agents interact through clear interfaces, abstracting internal processes and ensuring seamless communication.

# Autonomy

- In task processing, an **agent can decide** largely autonomously and without consultation or coordination with third parties (human users or other agents) which activities to execute
- This frequently requires or implicitly assumes that the decisions to be made by the agents are **non-trivial**, i.e., that they might require extensive knowledge processing or that the effects are significant
- An agent has a certain **scope of decision-making**, authorization and freedom of action and so is **subject to control** by third parties only to a restricted degree
- At the bottom line, autonomy implies the ability of an agent to independently handle its own complexity and that of its application and thus especially to relieve its users while protecting their interests

## Smart Home Assistant - Autonomy

Aspect	Example in Smart Home
<b>Agents Involved</b>	Smart Vacuum Cleaner, Smart Thermostat, Smart Lighting
<b>Autonomous Task Processing</b>	Smart Vacuum autonomously schedules cleaning based on user habits and room occupancy without needing external prompts.
<b>Non-Trivial Decision-Making</b>	Smart Thermostat analyzes weather data and user patterns to adjust heating schedules optimally without user input.
<b>Scope of Decision-Making &amp; Authorization</b>	Smart Lighting adjusts brightness and colour temperature throughout the day independently but respects user overrides.
<b>Handling Complexity &amp; User Relief</b>	Agents manage their tasks (e.g., cleaning, heating, lighting) efficiently, reducing user intervention while maintaining comfort and energy savings.

# Further agent attributes

- The most prominent of further attributes include:
  - **Situatedness/embeddedness.** An agent is connected to its environment via close sensory and/or actuator coupling. Thus, it acts and interacts directly in a concrete and socio-technical environment and not only in an abstract model of this environment
  - **Learning capability/adaptivity.** An agent independently optimizes its functionality with respect to the tasks that are assigned to it, which might change over time
  - **Evolution,** change of requirements

# Learning

- What can an agent learn?
  - About the environment
    - Eg. At 5pm all shops are closed
    - Eg. When she asking me to close the door, she wants to tell me something private
  - About itself
    - E.g. I always fail to close the door
    - E.g. it is always better to do one thing at a time
    - E.g., if it is going to rain the goal of organizing a picnic will fail
- How?
  - ML techniques -> Models about
  - By design – dedicated components

# Other attributes

- **Temporal continuity** and **Persistency**: an agent does not simply implement a one-time computation, but acts over a longer period of time
- **Rationality**: an agent acts in the realm of its capability and knowledge as well as possible with respect to fulfilling its task and goals; i.e., it maximizes its chances of success
- **Self-containment**: an agent is a functionally complete and executable entity
- **Inferential capability**: *an agent* acts on abstract task spec., using models of itself, situation, and/or other agents
- **Mobility**: an agent can migrate from one host to another in a self-directed way

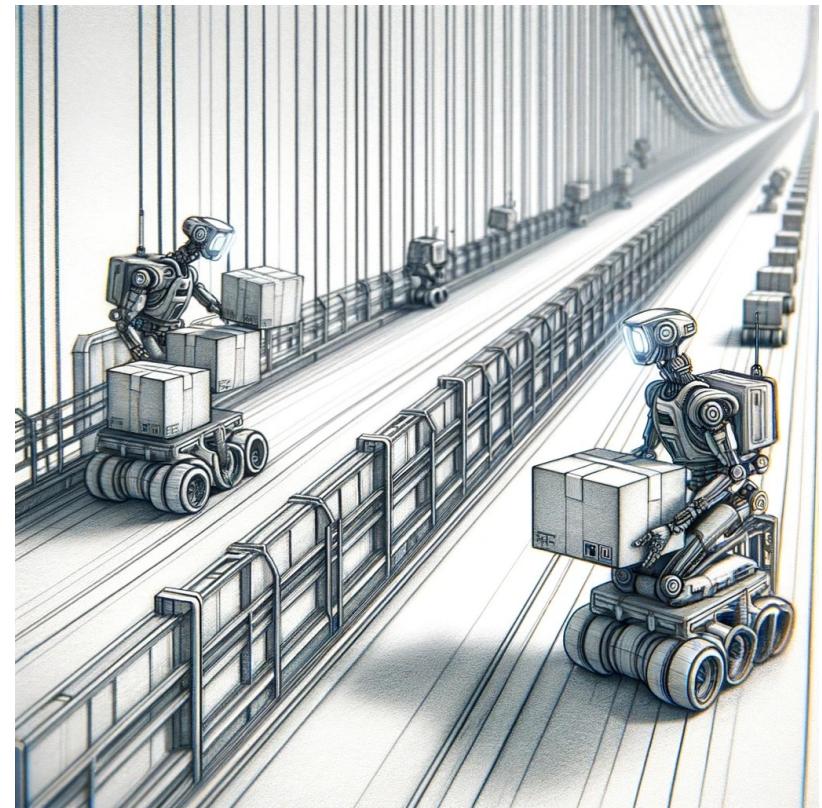
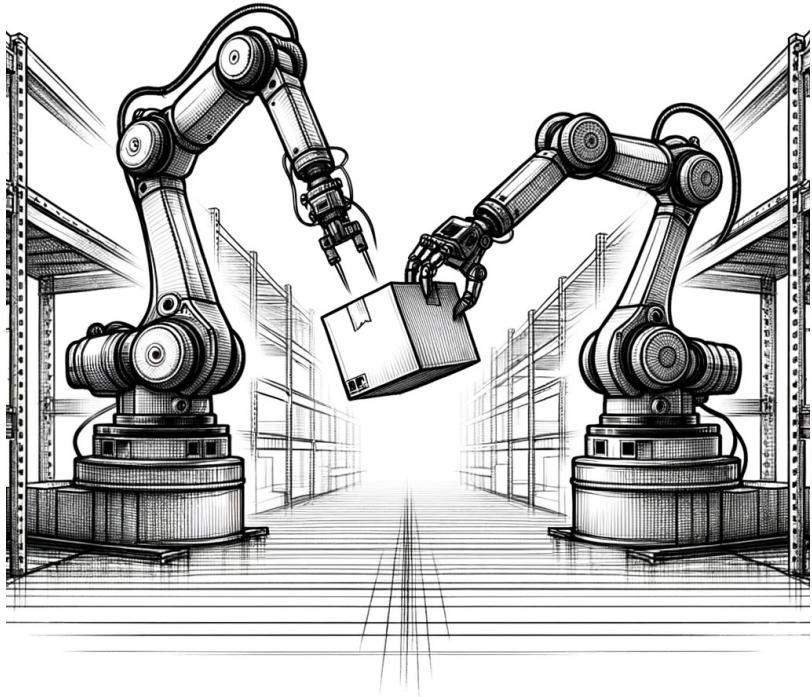
# Role/Goal vs. Task

- We assign “**tasks**” (or services) to software
  - Example: “on-line registration software”
  - “What” and “how” are specified in advance
  - Changes in requirements are not tolerable
- We assign “**roles/goals**” to agents
  - Example: registration agent
  - “What” is specified in advance. “How” is determined dynamically
  - Changes in requirements can be tolerated
- An agent selects and executes tasks at runtime
  - **Goal --> Tasks**
- Higher level of conceptualization

# Multi-agent systems

# More than a single agent

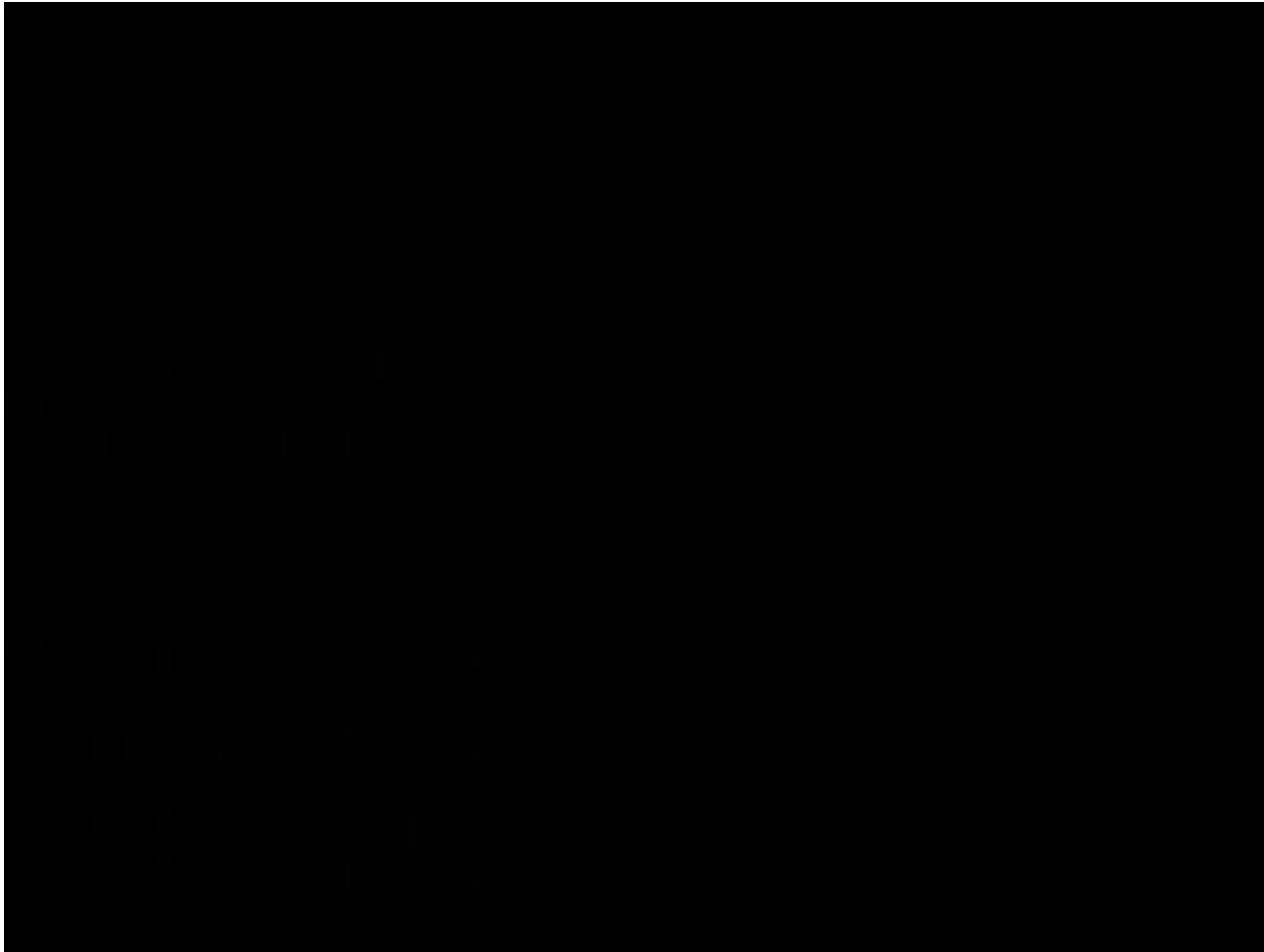
- When a single agent is not enough



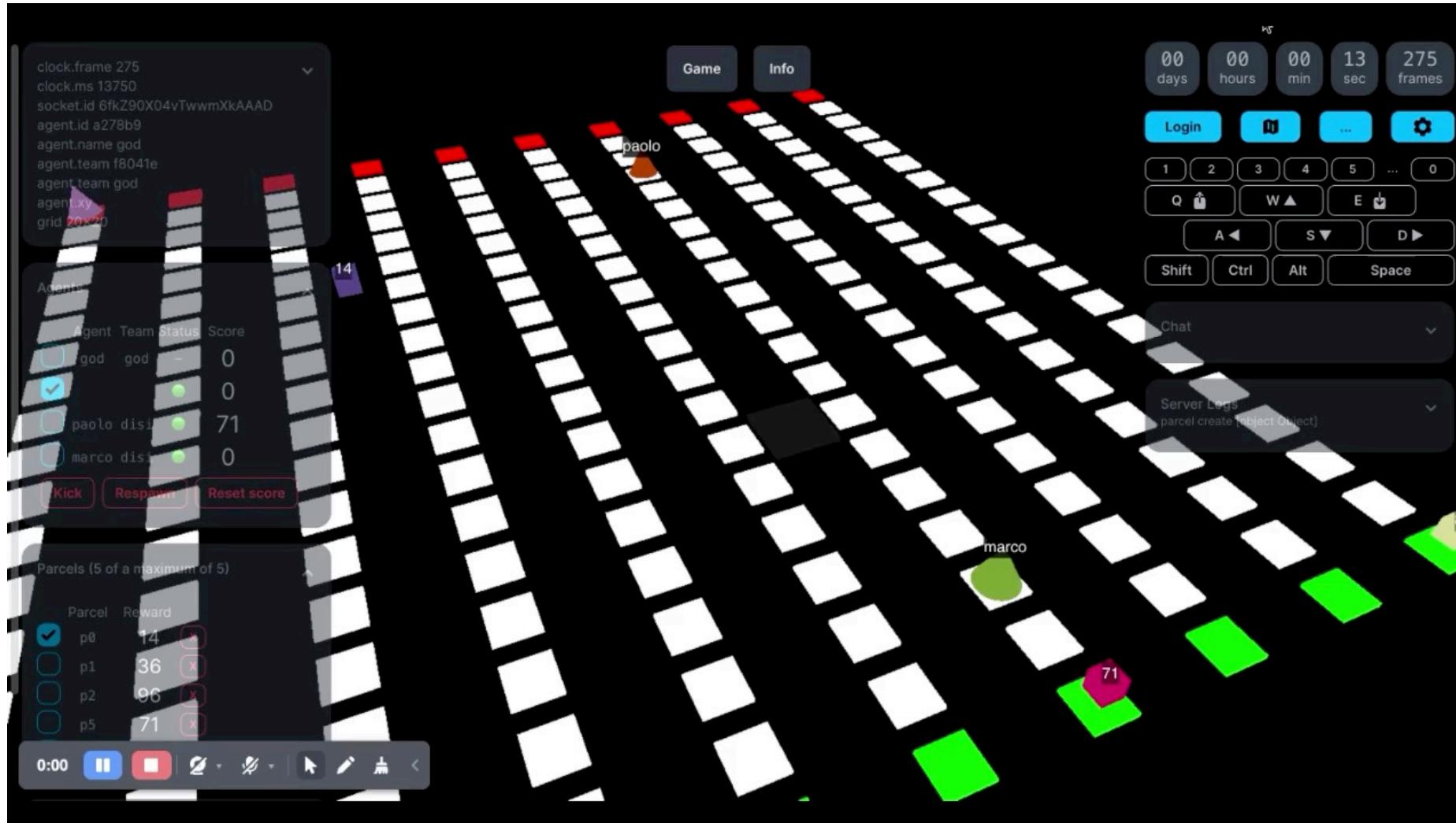
# More than a single agent

- In most cases, a single agent is not enough
  - No such thing as a single agent system (!?)
  - Multiple agents are the norm to represent
    - Decentralization
    - Multiple loci of control
    - Multiple perspectives
    - Competing interests
- A Multi-agent system definition
  - A multi-agent system is one that consists of a number of agents, which interact with one-another
  - In the most general case, agents will be acting on behalf of users with different goals and motivations
  - To successfully interact, they will require the ability to cooperate, coordinate, and negotiate with each other, much as people do

# Robots coordination



# Coordination in Deliveroo.js



# Another definition of MAS

- “... *If a problem domain is particularly complex, large, or unpredictable, then the only way it can reasonably be addressed is to develop a number of functionally specific and (nearly) **modular components** (agents) that are **specialized** at solving a particular problem aspect. When interdependent problems arise, the agents in the system must **coordinate** with one another to ensure that interdependencies are properly managed*” [Katia Sycara]
- Basics of a MAS:
  - **Interaction**
    - Cooperation, Coordination, Competition
    - Communication
  - **Local and organizational interests**
    - Agent objectives vs. organizational (social) objectives
  - **Organizational structure**
    - Overall structure, agents' roles, norms, rules, etc.

# Agent Interactions

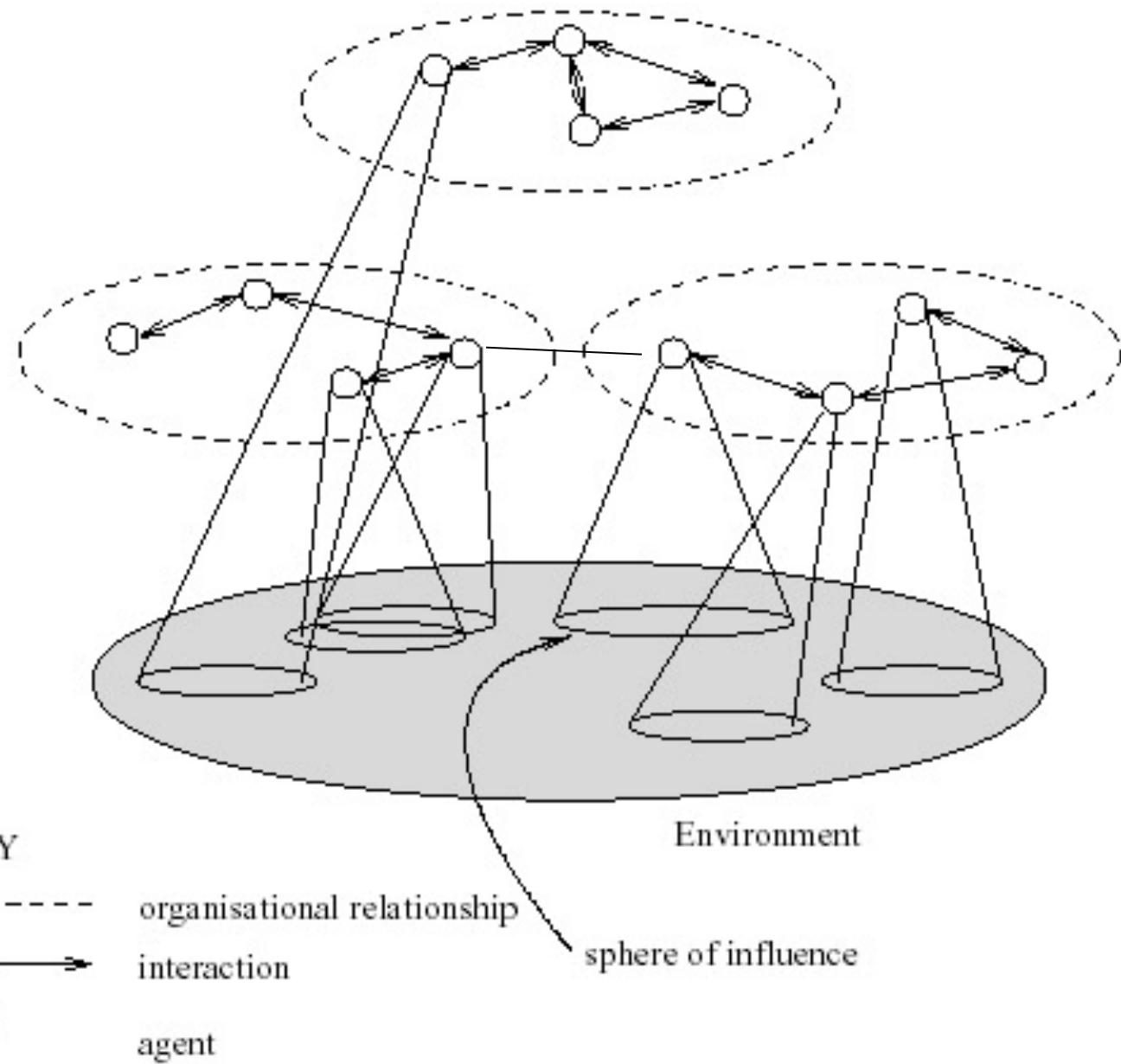
- Interaction between agents is inevitable
  - to achieve individual objectives
  - to manage inter-dependencies
- Conceptualised as taking place at knowledge-level
  - which goals, at what time, by whom, what for
- Agents act to achieve objectives
  - on behalf of individuals/companies
  - part of a wider problem solving initiative
- Agents interact in some organisational setting
  - organisational relationship between agents

# Organisations

- The organisational context
  - Influences agents' behaviour
  - Relationships and rules need to be made explicit
    - Peers, teams, coalitions, institutional norms
- Agents act and interact in organisational institutions
  - They perform particular roles (which may change)
  - They obey particular norms and regulations (which are explicitly stated)
  - Different levels of an organisation
    - E.g., Strategic level, management level, operational level
  - Organisational processes

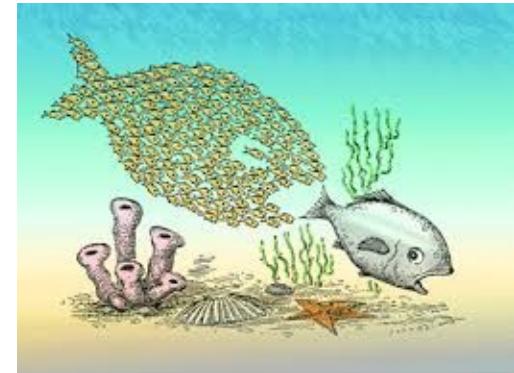
# A canonical view

- A multi-agent system contains a number of agents...
  - ...which interact through communication...
  - ...are able to act in an environment...
  - ...have different “spheres of influence” (which may coincide)...
  - ...will be linked by other (organizational) relationships



# Emergent behaviour

- Multi-agent system behaviour is **not fully predefined**
  - Behavior of a system that is not explicitly described by the behavior of the components of the system and is therefore **unexpected to a designer or observer**
- Result of dynamic interaction among the participants
  - flocking of birds cannot be described by the behavior of individual birds
  - market crashes cannot be explained by "summing up" the behavior of individual investors
- Local objectives generate emergent behaviours
  - Depending on the social/organizational context
  - Depending on the dynamics
  - Openness of the system



# Agent communication (1)

- Basis of interaction and social organization in MAS
- Not just sending messages
  - Communication is carried out by signals between agents
- Significance
  - To be significant there must
    - be an agent that can perceive the message
    - be an interpretive system that can transform the message into meaning



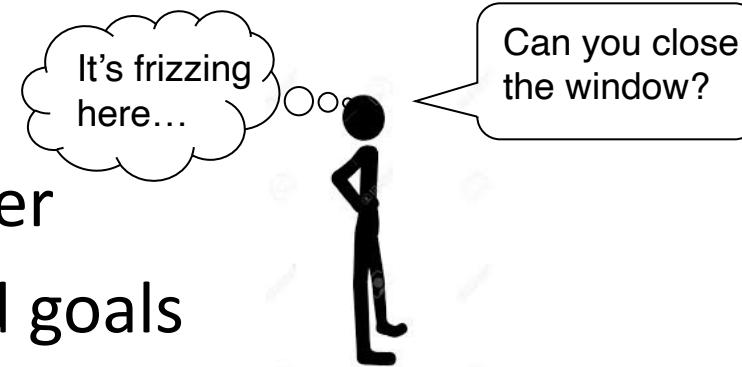
# Agent communication (2)

- communication consists of
  - Sending of information from a **sender** to a **receiver**
  - Encoding/decoding of information in a language
  - Transmission of information on channel or medium
  - A **context** in which the interlocutors (sender and receiver) are placed



# Agent communication (3)

- Characterize the attitude of the sender
- Indicate agent's state, intentions, and goals



- Allow agents to coordinate and make common belief
- Allow order/request & acceptance/refusal
- Send data related to the state of the world
- Verify channel function via acknowledge functions, etc.
- Request for clarification/more detail
- Ensure messages are properly understood



# SE challenges

- Two different problems:
  - How do we build agents capable of independent, autonomous actions, so that they can successfully carry out goals and tasks we delegate to them?
  - How do we build agents that are capable of interacting (cooperating, coordinating, negotiating) with other agents in order to successfully carry out those delegated tasks, especially when the other agents cannot be assumed to share the same interests/goals?
- The first problem is *agent design*, the second is *society/organisational design*
  - micro vs. macro

# Agents' architectures

Autonomous Software Agents

A.A. 2024-2025

**Prof. Paolo Giorgini**

**Dr. Marco Robol**



UNIVERSITY OF TRENTO - Italy

Department of Information  
and Communication Technology

# Internal and Social Architectures

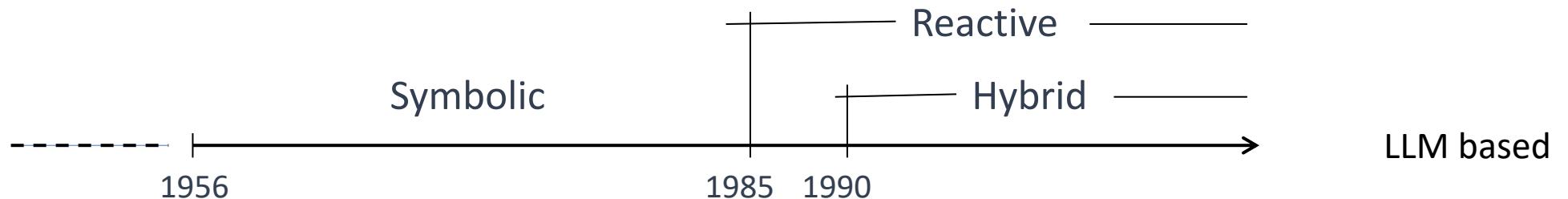
- The internal agent architecture determines the kinds of components an agent consists of and additionally define how these components interact
  - implement an **internal process** (eg. reasoning) that ultimately **leads to agent actions**
- The agent architectures have been devised for describing group structures and behavior
  - provide **concepts on an organizational level**, which allow the **description of structures and behavior** similar to how work is organized within human organizations

# Internal Agent architectures

- Fundamental mechanism underlying the autonomous components that support behaviour in real-world, dynamic and open environments
  - Logic-based (symbolic)
  - Reactive
  - Layered (hybrid)
  - BDI (deliberative)
- Most used
  - Reactive -- hybrid -- deliberative

# Agent Architectures

- Originally (1956–1985), pretty much all agents designed within AI were **symbolic reasoning** agents
  - agents use **explicit logical reasoning** to decide what to do
- Problems with symbolic reasoning led to a reaction against this — the so-called **reactive agents'** movement, 1985–present
- From 1990–present, a number of alternatives proposed: **hybrid** architectures, which attempt to combine the best of reasoning and reactive architectures



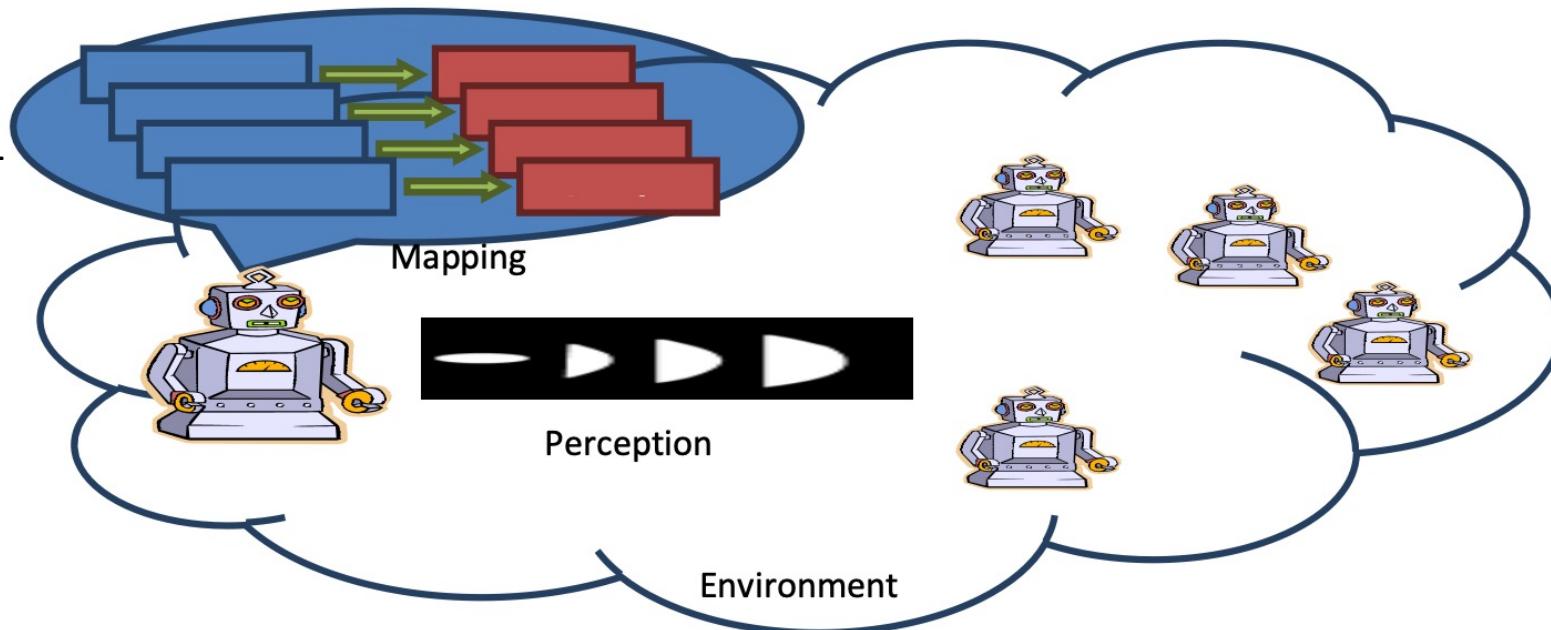
# Reactive architectures(1)

- The ability to perceive changes in the environment and response to it in a timely fashion
- Underpin the importance of **fast reactions to changes** within highly dynamic environments
  - In specific application domains **fast reactions outweigh correct behavior**, which is generated too slowly and might be no longer applicable
  - Reactive agents do **not possess a symbolic representation of the world** and build their decisions on the received percept
- Direct mapping of situation to action
- Realized through a set of **sensors** and **effectors**, where perceptual input is mapped to the effectors to changes in the environment

# Reactive architectures (2)

- Each of the percept situation is mapped into an action which specifically responses to the percept situation

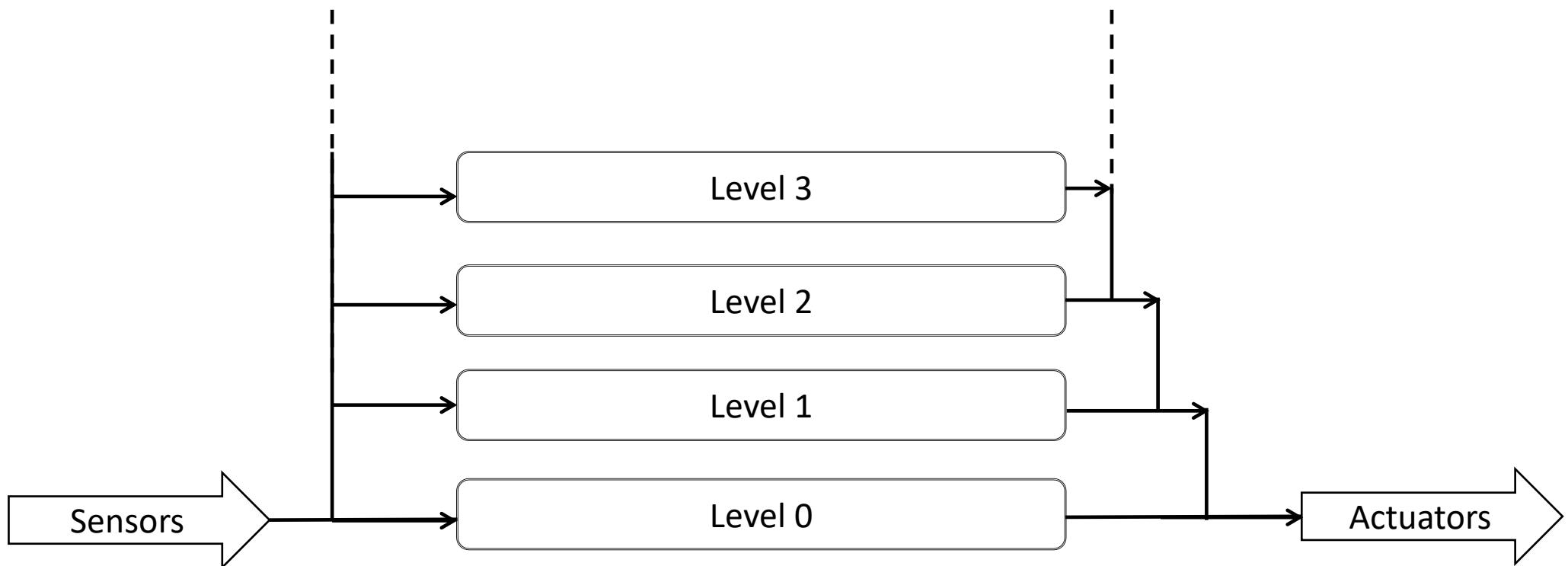
You can also think  
the mapping as a ML  
model



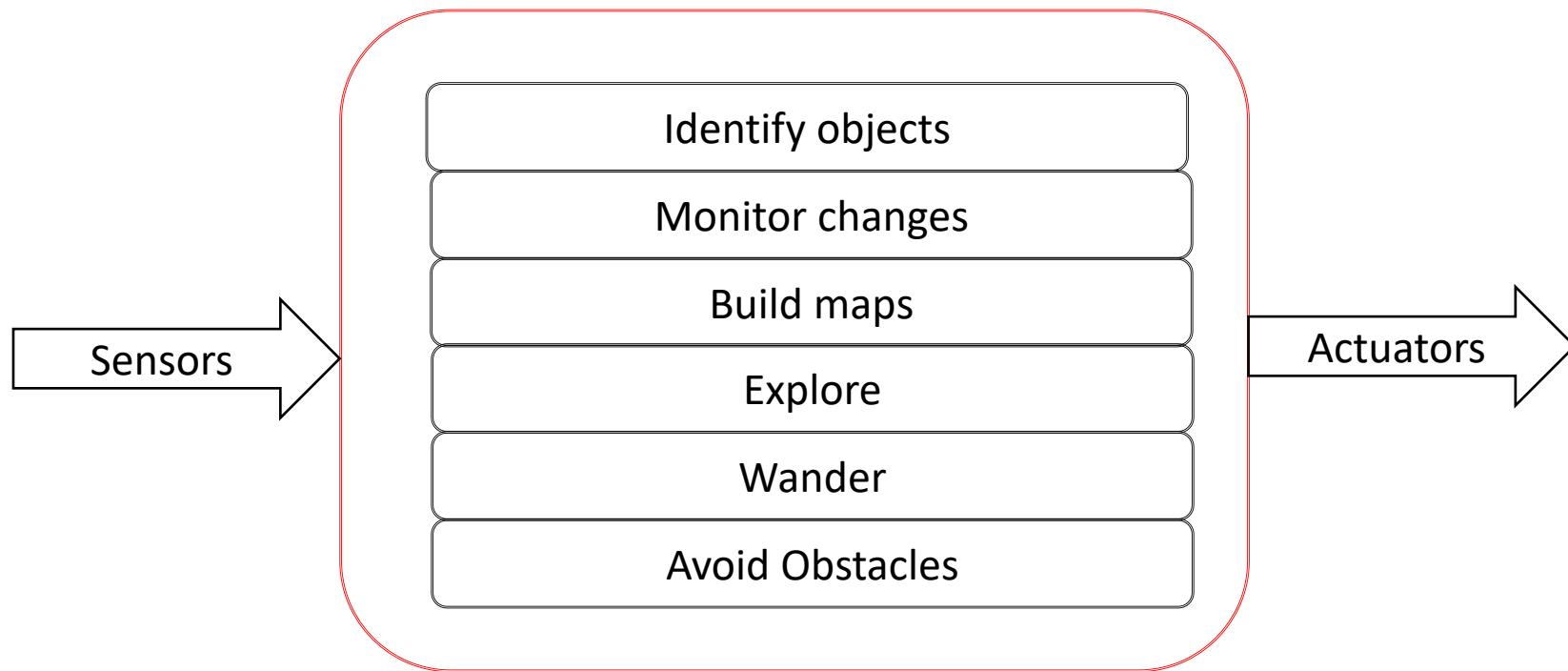
# Reactive Architecture - (Brooks, 1991)

- Brooks has put forward three theses:
  - Intelligent behavior can be generated without explicit representations
  - Intelligent behavior can be generated without explicit abstract reasoning
  - Intelligence is an emergent property of certain complex systems
- Subsumption architecture
  - Hierarchy of task-accomplishing behaviors
  - Each behavior is a rather simple rule-like structure
  - Each behavior ‘competes’ with others to exercise control
  - Lower layers represent more primitive kinds of behavior (such as avoiding obstacles), and have precedence over layers further up the hierarchy
  - The resulting systems are extremely simple

# Reactive Architecture – subsumption



# Reactive Architecture (1)



From Brooks, “A Robust Layered Control System for a Mobile Robot”, 1985

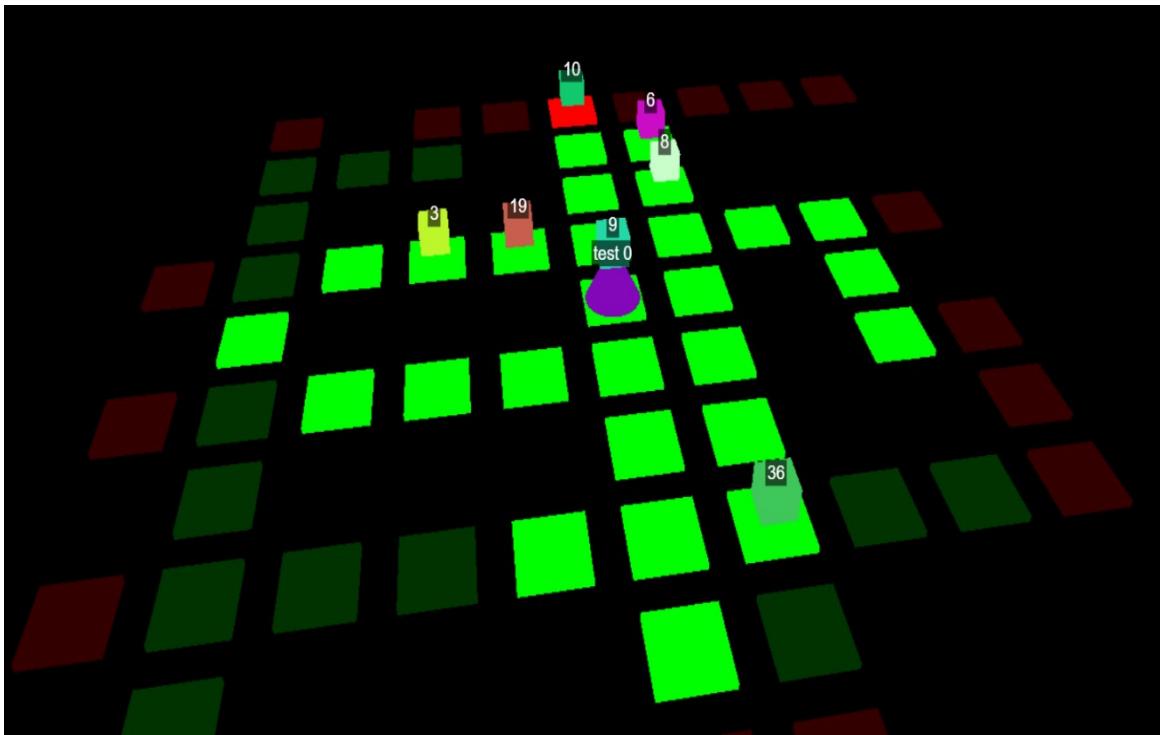
# Explorer Rules

- The lowest-level behavior is obstacle avoidance:
  - (1) **if detect an obstacle then change direction**
- Any samples carried by agents are dropped back at the mother-ship:
  - (2) **if carrying samples and at the base then drop samples**
- Agents carrying samples will return to the mother-ship:
  - (3) **if carrying samples and not at the base then travel to the base**
- Agents will collect samples they find:
  - (4) **if detect a sample then pick sample up**
- An agent with “nothing better to do” will explore randomly:
  - (5) **if true then move randomly**

```
while(run) {
    if (obstacle) { // level 0 (avoid obstacle)
        stop()
        chenge_direction()
    } else if (carry_samp) { // level 1 (if carrying a sample )
        if (at_station) {
            drop_samp()
        } else {
            move_toward_station()
        }
    } else if (samp) { // level 2 (if a sample is founded)
        pick_up()
    } else { // level 3 (if you don't have nothing to do)
        move_randomly()
    }
}
```

What about with a multithread implementation?

# Rules for Deliveroo



Try to define the rules for  
Deliveroo

# Advantages and Limitations of RAs

- Advantages
  - Simplicity, Computational tractability, Robustness against failure
- Limitations
  - Agents **without environment models** must have sufficient information available from local environment
  - If decisions are based on local environment, how does it take into account non-local information ? i.e., it has a “**short-term**” view ← NO concept of memory for RAs
  - How to make reactive **agents that learn** ?
  - Since behavior emerges from component interactions plus environment, it is hard to see **how to engineer specific agents** (no principled methodology exists)
  - It is hard to engineer agents with **large numbers of behaviors** (dynamics of interactions become too complex to understand)

# Logic-based architectures

- Emphasize a symbol-based reasoning process, which requires an agent to posses a **local world view**
- Intelligent behaviors can be generated by a **symbolic representation** of the environment and syntactically **manipulating** this representation
  - Logical formulae (symbolic representation)
  - Logical deduction or **theorem proving** (syntactic manipulation)
- A **theory can explain how the agent should behave**
  - how goals are generated to satisfy its delegated objectives
  - how agent interleaves goal-directed and reactive behavior to achieve these goals
  - The theory (specification) is refined through a series of progressively more concrete stages until finally an implementation is reached

# Symbolic Reasoning Agents

- The classical approach to building agents is to view them as a particular type of **knowledge-based system**
- This paradigm is known as **symbolic AI**
- We define a deliberative agent to be one that:
  - contains an **explicitly** represented, **symbolic model of the world**
  - makes decisions (for example about what actions to perform) via **symbolic reasoning**

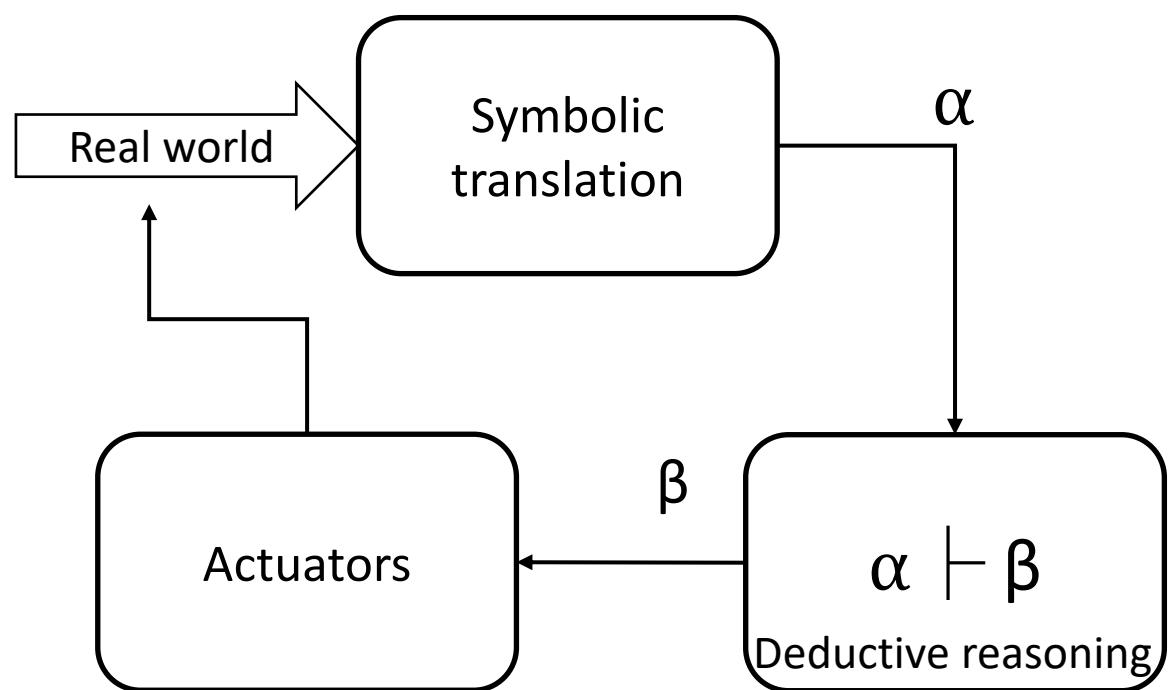
# Symbolic Reasoning Agents

To build an agent in this way, there are two key problems:

- **The transduction problem**
  - real world into an accurate, adequate symbolic description
  - in time for that description to be useful...vision, speech understanding, learning
- **The representation/reasoning problem:**
  - symbolically represent information about complex real-world entities and processes, and how to get agents to reason with this information in time for the results to be useful...knowledge representation, automated reasoning, automatic planning

Underlying problem lies with the **complexity** of symbol manipulation algorithms

- many (most) search-based symbol manipulation algorithms of interest are **highly intractable**



# Deductive Reasoning Agents

- How can an agent decide what to do?
- Basic idea is to use logic to encode a theory stating the *best action to perform in any given situation*
- Let:
  - $\rho$  be this **theory** (typically a set of rules)
  - $\Delta$  be a **logical database** that describes the current state of the world
  - $Ac$  be the **set of actions** the agent can perform
  - $\Delta \vdash_{\rho} \phi$  means that  $\phi$  can be proved from  $\Delta$  using  $\rho$

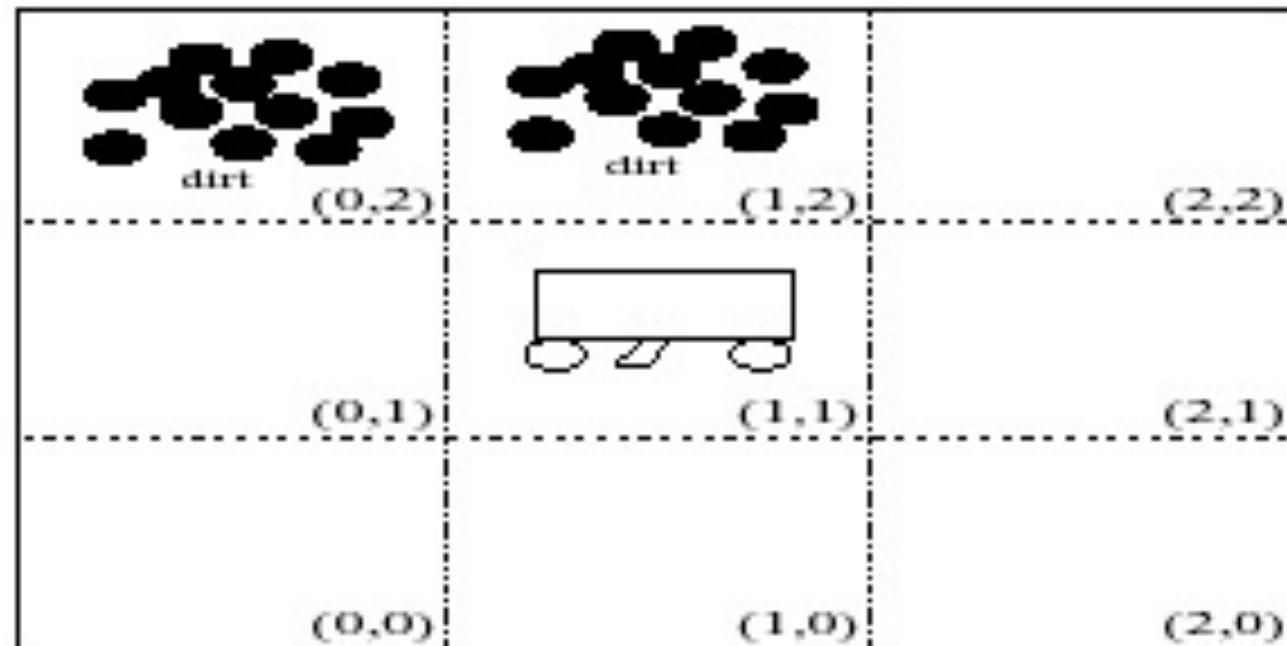
# Deductive Reasoning Agents: action selection

```
/* try to find an action explicitly prescribed */  
for each  $a \in Ac$  do  
    if  $\Delta \vdash_p Do(a)$  then  
        return  $a$   
    end-if  
end-for
```

```
/* try to find an action not excluded */  
for each  $a \in Ac$  do  
    if  $\Delta \not\vdash_p \neg Do(a)$  then  
        return  $a$   
    end-if  
end-for  
return null /* no action found */
```

# Deductive Reasoning Agents

- An example: **The Vacuum World**
  - Goal is for the robot to clear up all dirt



# Deductive Reasoning Agents

- Use 3 *domain predicates* to solve problem:

*In*( $x, y$ )

agent is at ( $x, y$ )

*Dirt*( $x, y$ )

there is dirt at ( $x, y$ )

*Facing*( $d$ )

the agent is facing direction  $d$

- Possible actions:

$Ac = \{turn, forward, suck\}$

... *turn* means “turn right”

# Deductive Reasoning Agents

- Rules  $\rho$  that govern the agent's behavior:
  - $\varphi(\dots) \rightarrow \psi(\dots)$
  - $\varphi$  and  $\psi$  are predicates over some arbitrary list of constants and variables
  - If  $\varphi$  matches against the agent's database, then  $\psi$  can be concluded, with any variables in  $\psi$  instantiated
- Rules
  - Cleaning action rule:  $\text{In}(x, y) \wedge \text{Dirt}(x, y) \rightarrow \text{Do}(\text{suck})$
  - This action will take priority over all other possible behaviors of the agent (such as navigation)
  - In case the rule does not match against the database, the basic action of the agent will be to traverse the world

# Deductive Reasoning Agents

knowledge base

In an agent's database we have facts like

$\text{In}(1, 1) \quad \text{Dirt}(0, 2) \quad \text{Dirt}(1, 2) \quad \text{Facing}(\text{West})$

while formulas like

$\text{In}(x, y) \wedge \text{Dirt}(x, y) \rightarrow \text{Do}(\text{suck})$

are part of the reasoning framework and not in the database

# Example

## States

In  $(x, y)$

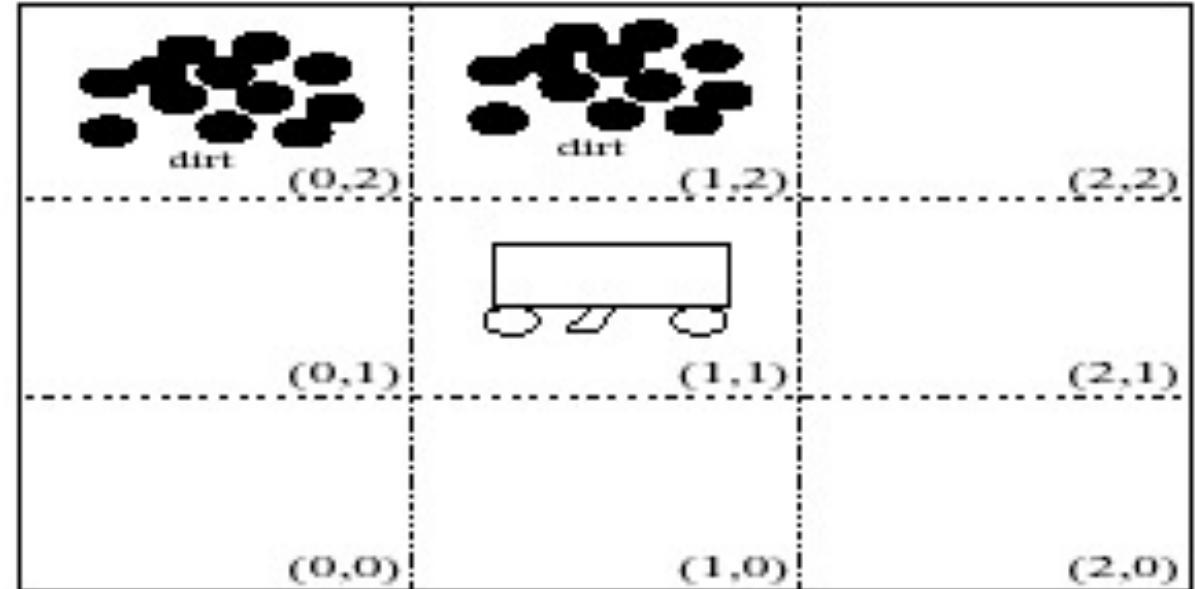
Facing  $(d)$  :  $d = \{ n, s, e, w \}$

Dirt  $(x, y)$

## Actions

DO  $(a)$  :  $a = \{ \text{suck}, \text{rotate}, \text{forward} \}$

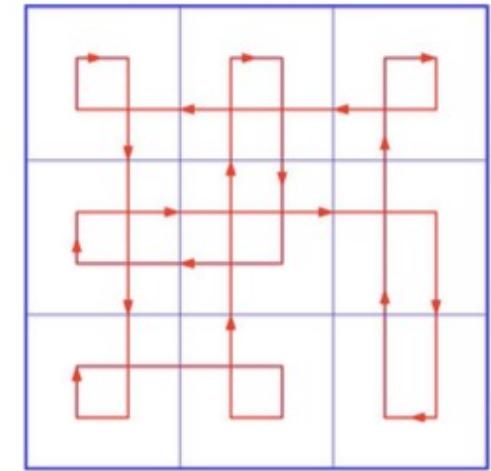
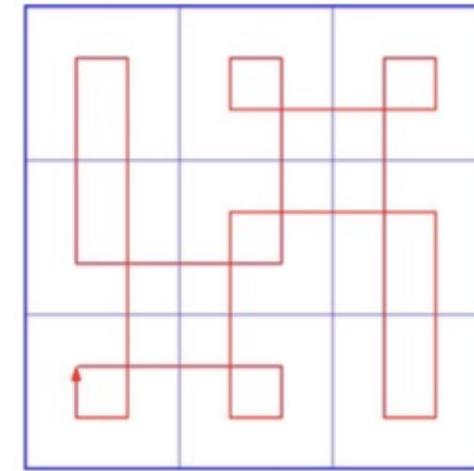
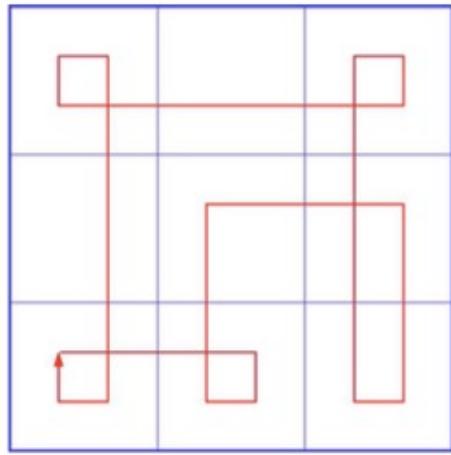
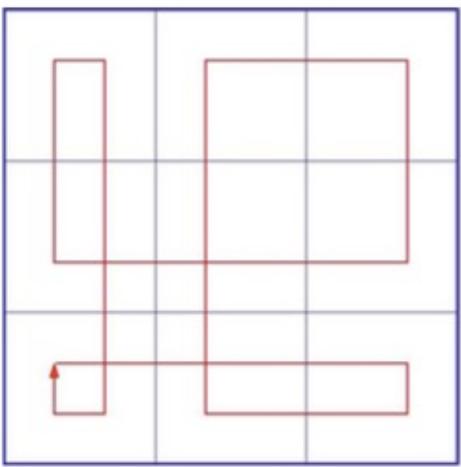
“rotate” means rotate clockwise the direction of moving



# Solution in two steps

- Find a cyclic path that allows the robot to go through all positions
- Write the navigation rules
  - The order will define the priority of the rules

# Possible solutions



# Rules

$In(x, y) \wedge Dirt(x, y) \rightarrow DO(suck)$

$[In(0, 0) \vee In(1, 2)] \wedge \neg Facing(E) \rightarrow DO(turn)$

$[In(2, 0) \vee In(2, 1)] \wedge \neg Facing(W) \rightarrow DO(turn)$

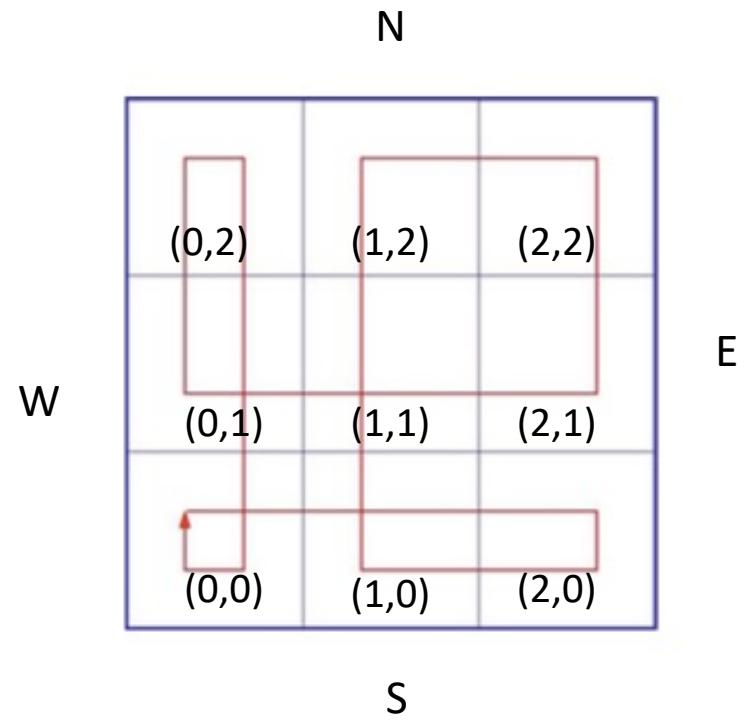
$[In(0, 2) \vee In(2, 2)] \wedge \neg Facing(S) \rightarrow DO(turn)$

$In(1, 0) \wedge [Facing(W) \vee Facing(S)] \rightarrow DO(turn)$

$In(1, 1) \wedge [Facing(E) \vee Facing(S)] \rightarrow DO(turn)$

$In(0, 1) \wedge [Facing(W) \vee Facing(E)] \rightarrow DO(turn)$

$DO(forward)$



# Extended

A new predicate  $\text{free}(x, y, d)$

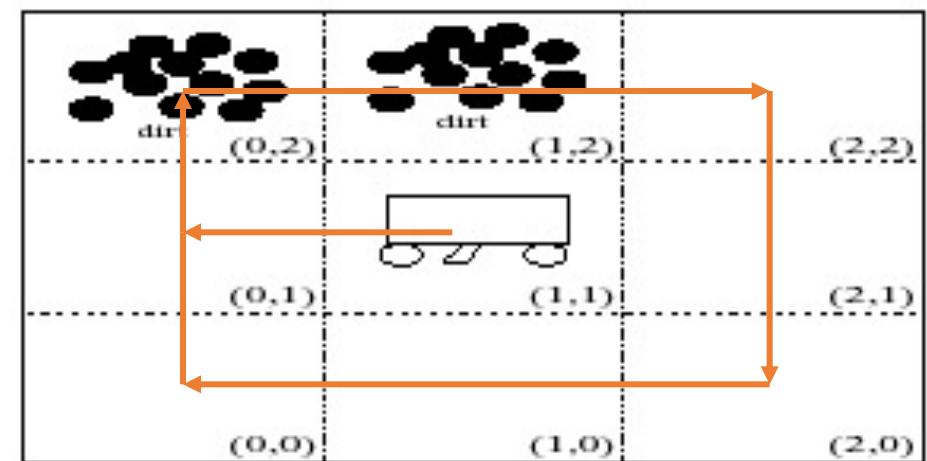
the agent is **free to move** in a given direction  $d$  when it is at position  $(x, y)$

$\text{In}(x, y) \wedge \text{Dirt}(x, y) \rightarrow \text{Do}(\text{suck})$

$\text{free}(x, y, d) \rightarrow \text{Do}(\text{forward})$

$\text{Do}(\text{rotate})$

- Try with memory



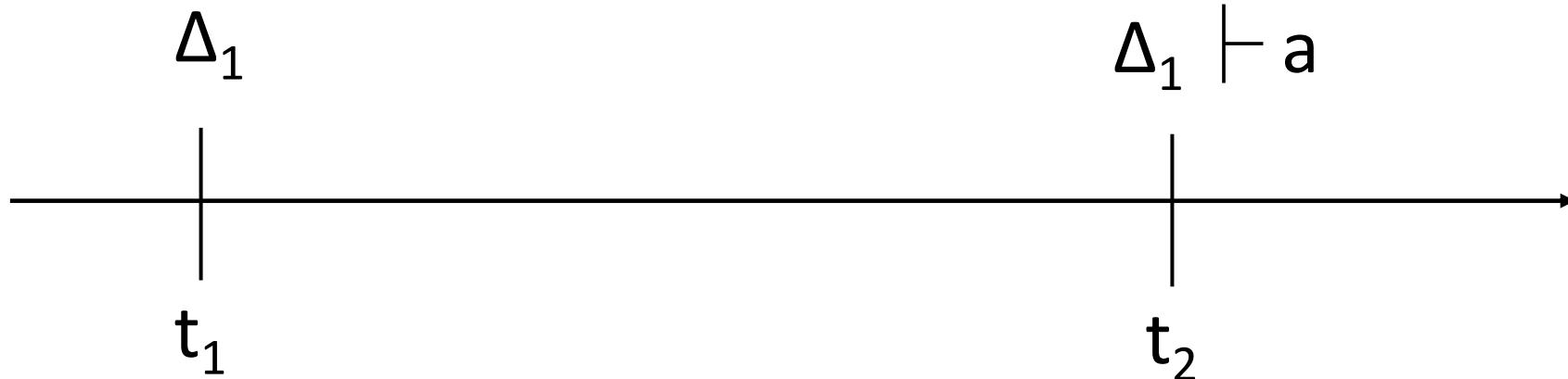
# Problems with DRA

If the time to take de decision  $a$  is longer than the time that the environment take to change, so  $a$  is not longer the optimal action to take. For example, if i decide to take the bus but while i'm taking the decision bus is leaving, at the end the decision i take is not longer applicable.

- If we can prove  $\text{Do}(a)$ , then  $a$  is an optimal action
  - namely, the best action that could be performed when the environment is as described in the database
- Suppose at time  $t_1$  the agent's database is  $\Delta_1$  and  $a$  is the optimal action to be performed. At time  $t_2$ , the agent deduces  $a$  (it took some time to do it) so the agent start to do  $a$
- What happened between  $t_1$  and  $t_2$ ?
  - If  $t_2 - t_1$  is infinitesimal, no problem (decision making was instantaneous)
  - Logic-based reasoning is not instantaneous !

# Calculative rationality

the best action is “a”



Is “a” still the best action to perform?

# Calculative rationality

- An agent is said to enjoy the property of **calculative rationality** if and only if its decision –making apparatus will suggest an action that was optimal **when the agent decision-making process began**.
- Calculative rationality is clearly not acceptable in environment that change faster than the agent make decisions
- Moving away from strictly logical representation languages and complete sets of deduction rules, one can build agents that enjoy respectable performance
  - Losing obviously advantages like simplicity and logical semantics

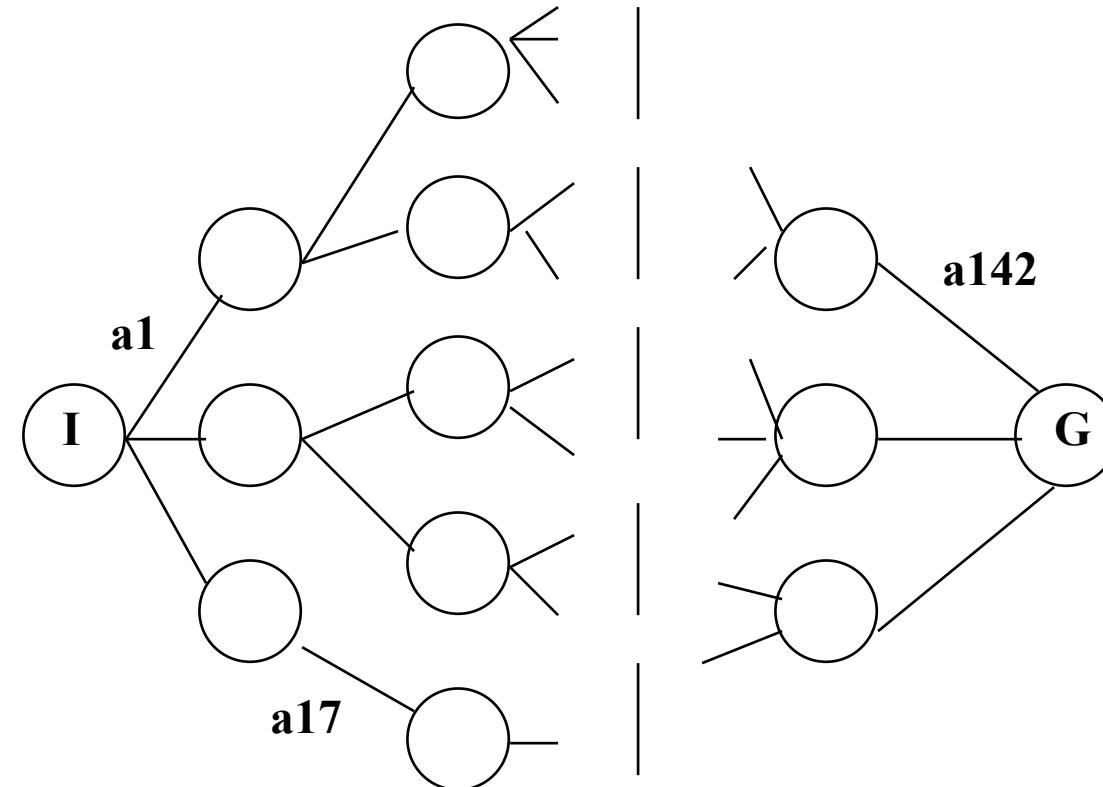
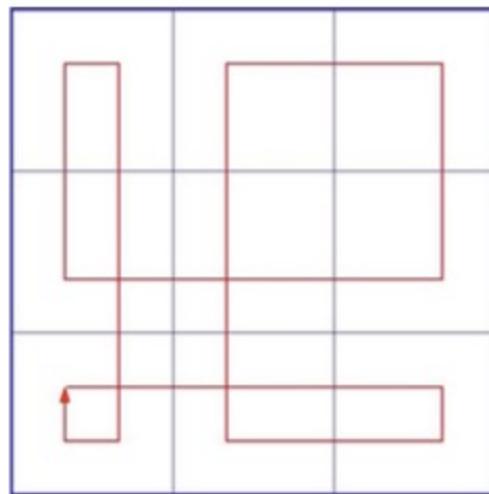
# Other problems

The problem of “translating” raw data provided by the agent’s sensors into an internal symbolic form

- How to convert video camera input to  $Dirt(0, 1)$ ?
- Representing properties of dynamic, real-world is extremely hard (e.g., represent and reason about temporal information)
- Decision making using first-order logic is *undecidable!*
  - Even where we use *propositional* logic, decision making in the worst case means solving NP-complete problems
- Typical solutions:
  - weaken the logic
  - use symbolic, non-logical representations
  - shift the emphasis of reasoning from *run time* to *design time*

# Planning Systems (in general)

- A reasoning agent is one that is able to create a plan of actions
  - find a sequence of actions that transforms an initial state into a goal state



# Planning

- Planning involves issues of both Search and Knowledge Representation
  - Eg., Robot Planning - Planning of speech acts

The Blocks World consists of equal sized blocks on a table

- A robot arm can manipulate the blocks using the actions:

UNSTACK (a, b)  
STACK (a, b)  
PICKUP (a)  
PUTDOWN (a)

# The Blocks World

- The Blocks World (today) consists of equal sized blocks on a table
- A robot arm can manipulate the blocks using the actions:

UNSTACK (a, b)

STACK (a, b)

PICKUP (a)

PUTDOWN (a)

# The Blocks World

- We also use predicates to describe the world:

ON (A, B)

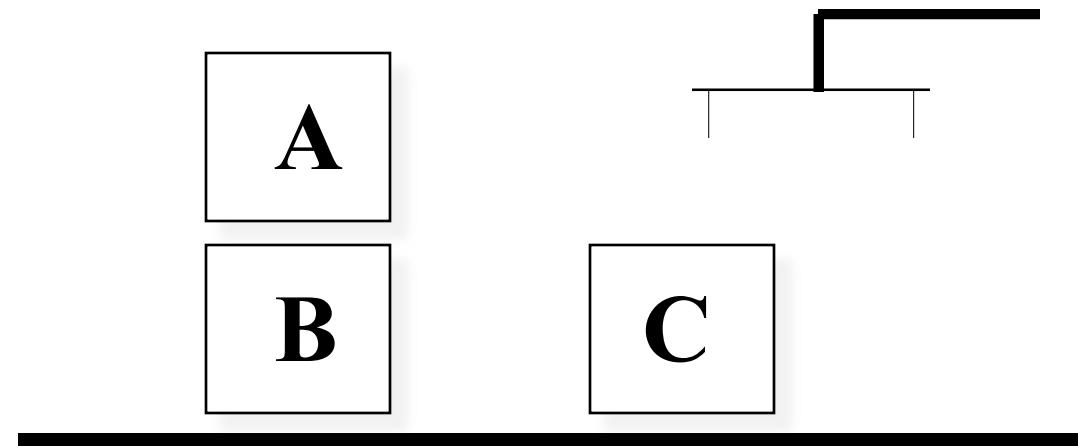
ONTABLE (B)

ONTABLE (C)

CLEAR (A)

CLEAR (C)

ARMEEMPTY



# Formulas to describe facts always True

we can write general logical truths relating the predicates:

$$\exists x \text{ HOLDING}(x) \rightarrow \neg \text{ARMEMPTY}$$

$$\forall x (\text{ONTABLE}(x) \rightarrow \neg \exists y (\text{ON}(x, y)))$$

$$\forall x (\neg \exists y (\text{ON}(y, x)) \rightarrow \text{CLEAR}(x))$$

So...how do we use theorem-proving techniques to construct plans?

# Green's Method

- Add state variables to the predicates, and use a function  $\text{DO}$  that maps actions and states into new states

$\text{DO} : A \times S \rightarrow S$

- Example:

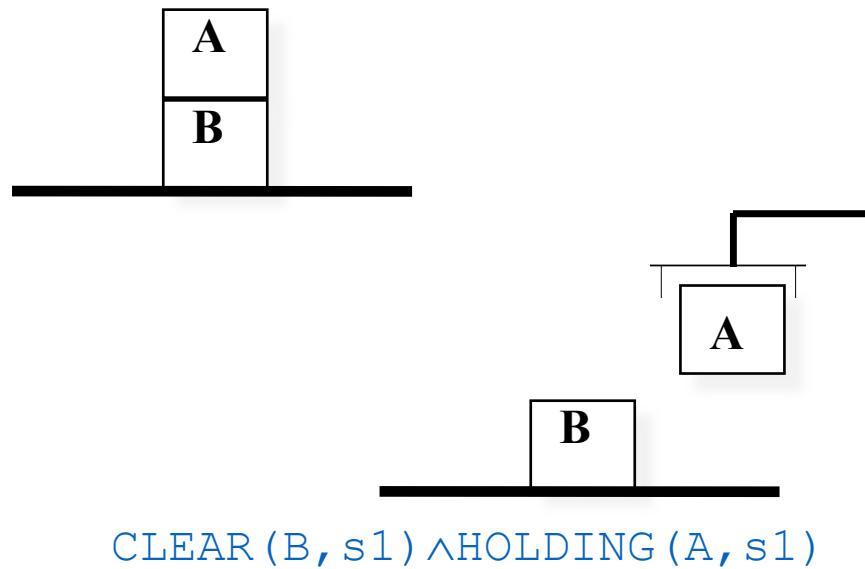
$\text{DO}(\text{UNSTACK}(x, y), s)$  is a new state

# UNSTACK

- To characterize the action UNSTACK we could write:

$$\begin{aligned} \text{CLEAR}(x, s) \wedge \text{ON}(x, y, s) \rightarrow \\ (\text{HOLDING}(x, \text{DO}(\text{UNSTACK}(x, y), s)) \wedge \\ \text{CLEAR}(y, \text{DO}(\text{UNSTACK}(x, y), s))) \end{aligned}$$

- We can prove that if  $s_0$  is

$$\text{ON}(A, B, S_0) \wedge \text{CLEAR}(A, S_0) \text{ then}$$
$$\text{HOLDING}(A, \text{DO}(\text{UNSTACK}(A, B), S_0)) \wedge \\ \text{CLEAR}(B, \text{DO}(\text{UNSTACK}(A, B), S_0))$$


# More Proving

- The proof could proceed further; if we characterize PUTDOWN:

HOLDING (x, s)  $\rightarrow$  ONTABLE (x, DO (PUTDOWN (x), s))

- Then we could prove:

ONTABLE (A,  
DO (PUTDOWN (A),  
DO (UNSTACK (A, B), S0)))

---

$s_1$

---

$s_2$

- The nested actions in this constructive proof give you the plan:
  1. UNSTACK (A, B); 2. PUTDOWN (A)

# More Proving

- if we have in our database:

ON (A, B, S<sub>0</sub>)  $\wedge$  ONTABLE (B, S<sub>0</sub>)  $\wedge$  CLEAR (A, S<sub>0</sub>)

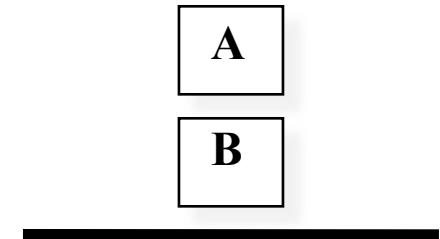
and our goal is

$\exists s: \text{ONTABLE}(A, s)$

we could use theorem proving to find the plan

But how?

ONTABLE (B,  
DO (PUTDOWN (A),  
DO (UNSTACK (A, B) , S<sub>0</sub>)) )

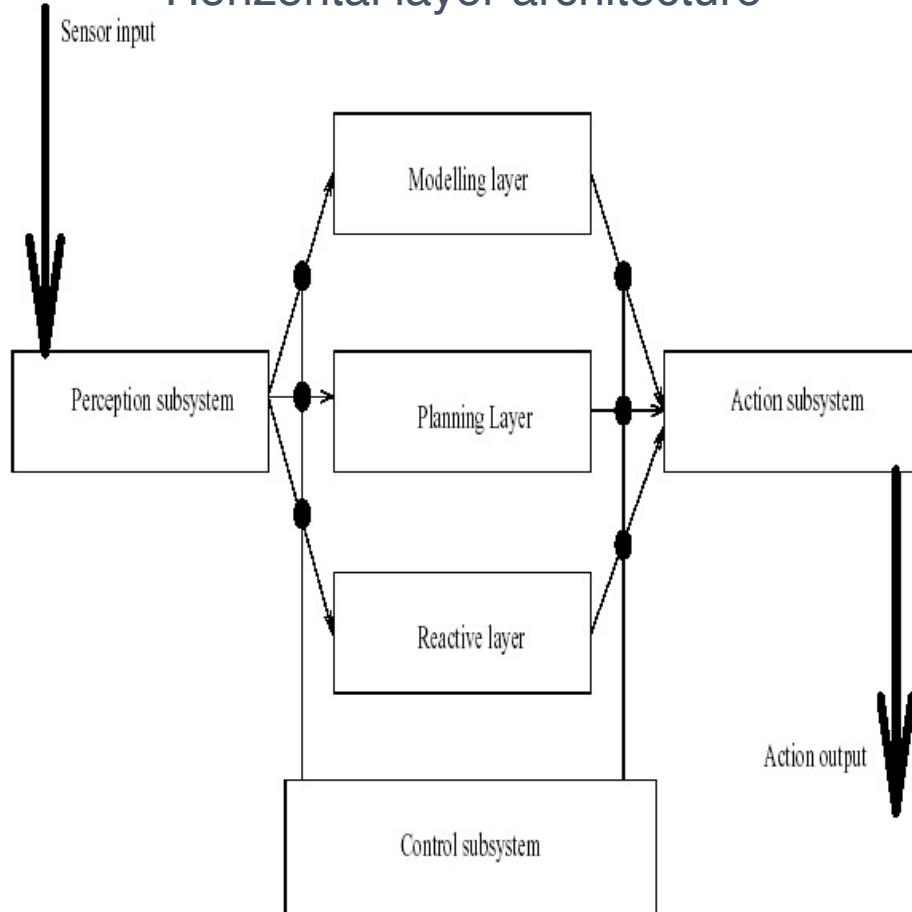


# Hybrid Architectures

- A hybrid architecture is based on two (or more) parts:
  - **deliberative**, containing a symbolic world model, which develops plans and makes decisions in the way proposed by symbolic AI
  - **reactive**, which is capable of reacting to events without complex reasoning (often, the reactive component is given some kind of precedence over the deliberative one)
- This kind of structuring leads naturally to the idea of a **layered architecture** -- the agent's control subsystems are arranged into a hierarchy, with higher layers dealing with information at increasing levels of abstraction
  - **Horizontal layering**: each layer is directly connected to the sensory input and action output. Each layer itself acts like an agent, producing suggestions as to what action to perform.
  - **Vertical layering**: sensory input and action output are each dealt with by at most one layer each

## Ferguson – TOURINGMACHINES

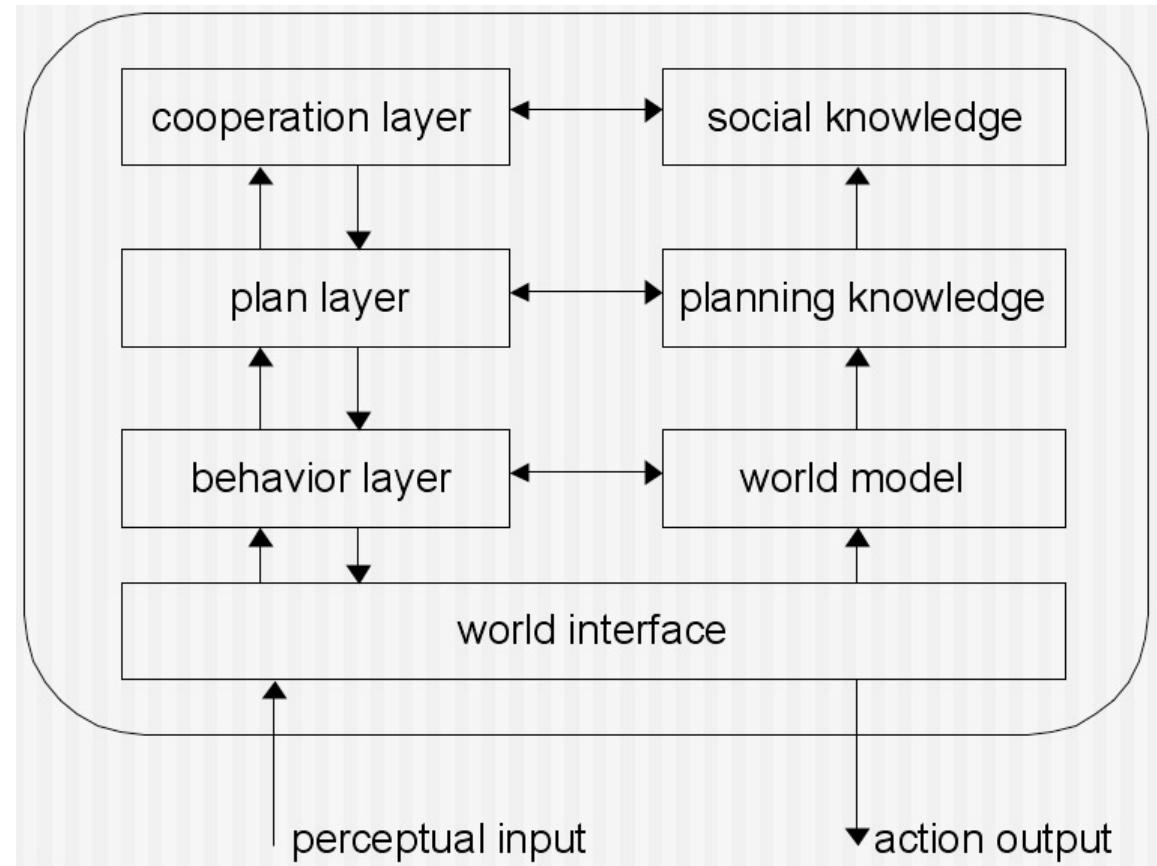
### Horizontal layer architecture



$m^n$  possible interactions  
between  $m$  layers ( $m$  are the  
actions for each of the  $n$   
layers)

## Müller – InteRRaP

### Two-pass Vertically layer architecture



# BDI architecture

- The most popular and used in agent community (Rao and Georgeff, 1995)
- Many implementations
  - PRS (1987), dMARS (1998), JAM (1999), Jack (2001), JADEX (2005)
- A model of practical reasoning (1980s) where:
  - Environment can evolve in many (unpredictable) directions
  - Many potential courses of actions available at any point
  - Many potential objectives at any point
  - Environment can only be sensed locally
  - Environment dynamic - implies resource bound in reasoning!
- Additional assumption: plans!
  - “Plan” used in the sense of a known recipe (“know-how”) rather than a chosen course of action (i.e., not “here” is the plan”)

given a particular state that we want to achieve, we have different set of actions available to try to generate a plan

# BDI: Belief

it can be represented by a database

- Information about the world, the past, ...  
Beliefs are important to take decisions. In order to have consistents beliefs it necessary to do a belief revision and eventually remove inconsistent values
- Why?
  - World is dynamic, so the agent needs to remember the past
  - Agent has a local view, so it needs to remember
  - Agent resource bounded, so store information rather than re-compute
- Since stored information is imperfect, semantics should be beliefs, not knowledge
  - Knowledge (in the context of logics of knowledge of belief) has the property that if agent knows P then P must be true
- Note: in addition to caching beliefs, also cache plans (recipes) for similar reasons

# BDI: (Goal)Desire

Possible goals that the agent can try to achieve  
they can be in conflict with each other

- Desired end state (e.g., “you desire to graduate”)
- Captures why a particular piece of code is/was executing
  - ... useful for recovering from failure (cf. execution cycle)
  - ... (potentially) allows for reasoning about goal interaction

# BDI: Intention

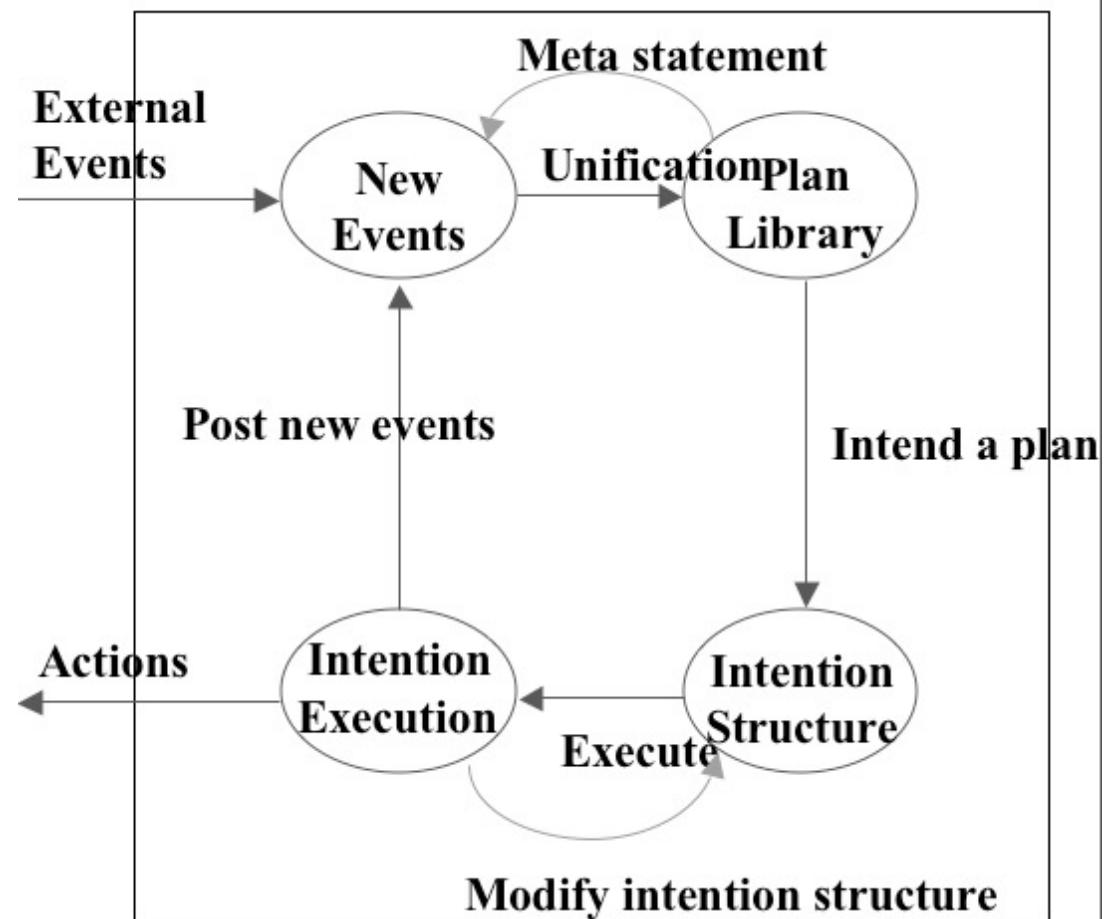
- Question: Are beliefs and desires enough?
- Answer: No, also need intentions
- Intention = selected course of action (plan instances)
- Why?
  - Resource bounded agents in dynamic environment need to plan bit-by-bit while acting
  - Intentions capture the partial plan (in the sense of “course of action”)
  - Intentions can also be used to coordinate agents
- Properties:
  - Intentions lead to action
  - Intentions are persistent by default
  - Intentions should be internally consistent
  - Intentions should be fleshed out by the time they need to be “executed”

# Commitment, re-planning, and options

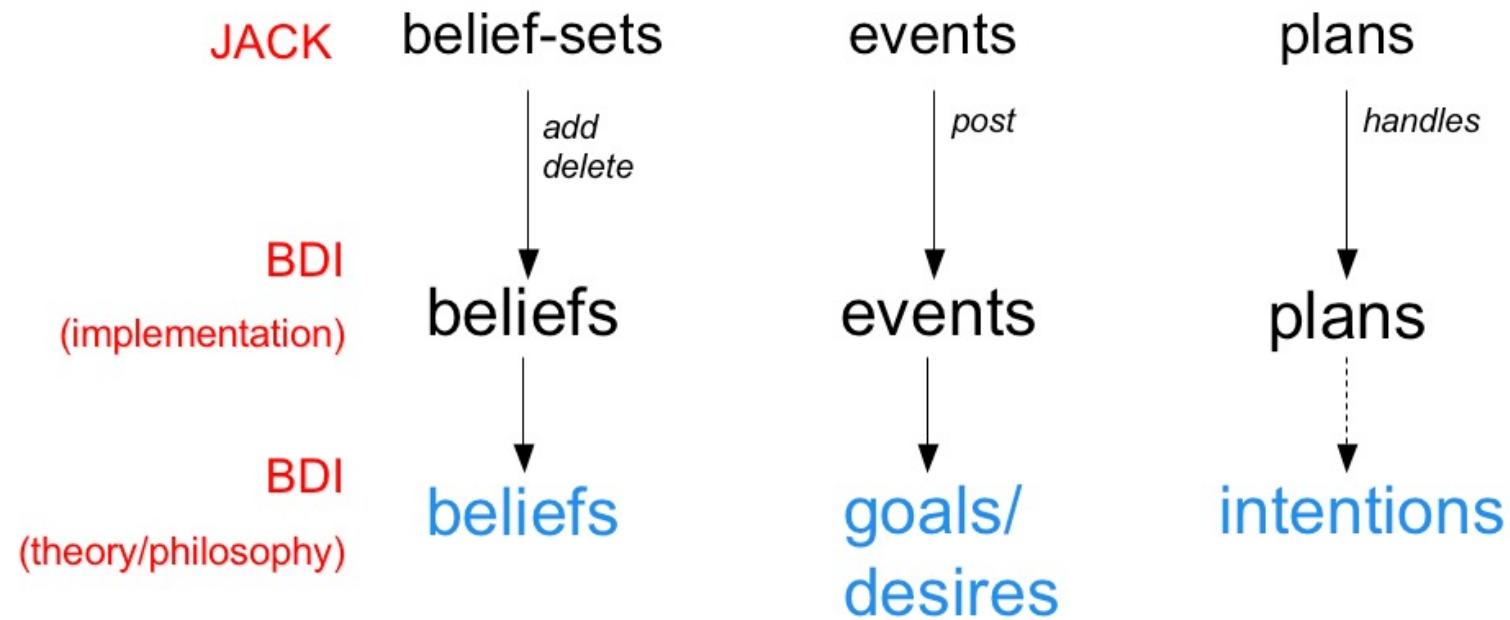
- You are in the middle of following an intention, and the world changes, should you re-plan?
  - Classical decision theory: yes, always
  - Classical computing (no goals): no, never
- Neither is ideal - want mixture!
- A deliberating agent has many available options (too many to consider!), so want to reduce the options to consider
  - Don't consider options that conflict with selected intentions
  - Don't reconsider chosen options (unless there's a good reason)
- When doing further planning, assume that intentions will have been achieved
  - e.g., intend to return library book in the afternoon and go to movies in the evening, since library is at UNITN@Albere, plan to go to movies in the city

# BDI: Implementation in Jack

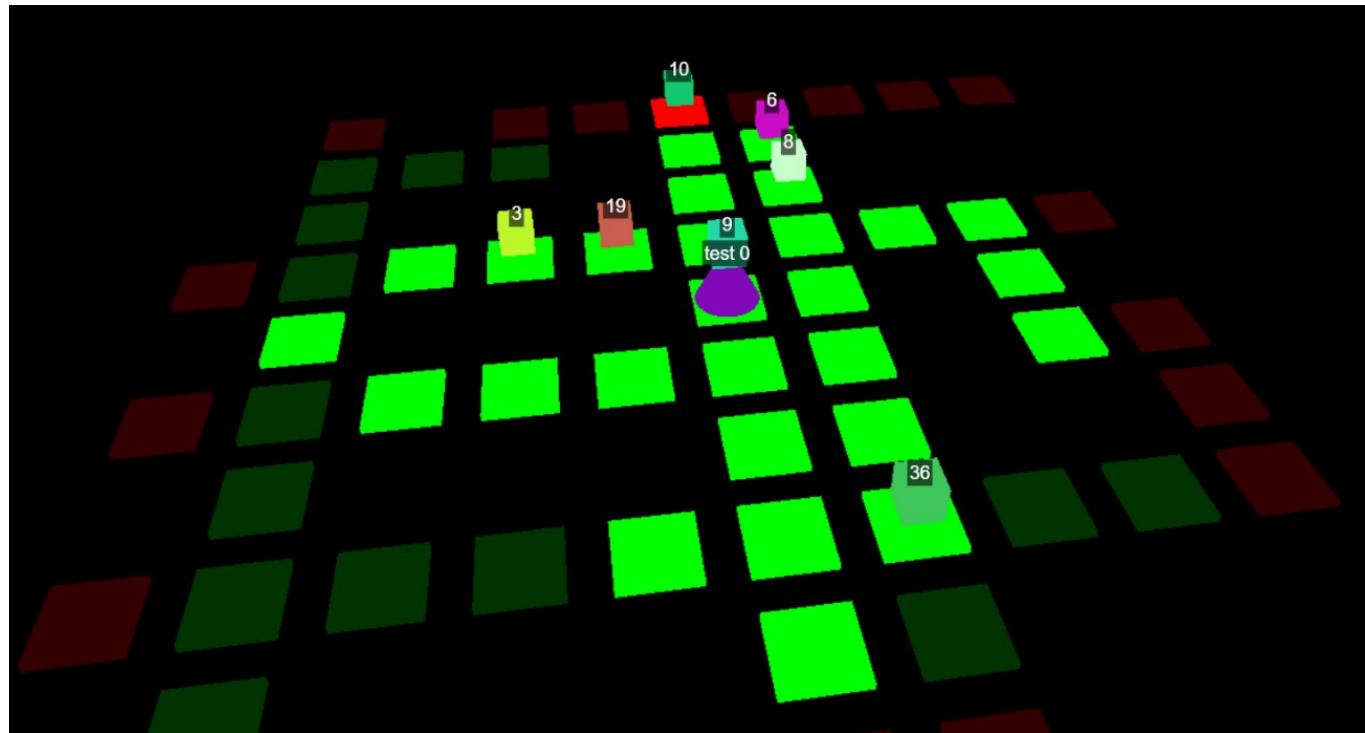
- Architecture vs implementations
- Beliefs - local knowledge base (really database)
- Events - used to model goals
- Plans - predetermined sequences of actions or sub-goals that can accomplish specified tasks
- Intentions - currently running plans



# BDI Theory and JACK implementation



# In the Deliveroo game



What are Beliefs, Desires,  
Goals, Intentions, Plans and  
Actions?

# BDI agents

Autonomous Software Agents

A.A. 2024-2025

**Prof. Paolo Giorgini**

**Dr. Marco Robol**



UNIVERSITY OF TRENTO - Italy

Department of Information  
and Communication Technology

# Practical Reasoning

- Practical reasoning is reasoning **directed towards actions** — the process of figuring out what to do:

“Practical reasoning is a matter of weighing conflicting considerations for and against competing options, where the relevant considerations are provided by what the agent desires/values/cares about and what the agent believes.” (Bratman 1990)

- Example: Alex has received two job offers:

- **Job A:** A high-paying corporate job with long hours and little work-life balance.
- **Job B:** A lower-paying job at a nonprofit organization that aligns with Alex’s passion for social impact and provides a better work-life balance

Alex engages in **practical reasoning** by weighing different considerations:

## Desires/Values/Cares About:

- Alex values financial security and career growth (Job A)
- Alex also deeply cares about making a positive social impact and maintaining a good work-life balance (Job B)

## Beliefs:

- Alex believes that Job A will lead to financial stability but might result in stress and burnout.
- Alex believes that Job B will be more fulfilling but may not provide the same financial security

Alex must now weigh these **conflicting considerations** to make a decision. If financial security is the overriding factor, Alex may choose Job A. If personal fulfillment and work-life balance matter more, Alex may opt for Job B

# Practical vs Theoretical Reasoning

- Practical reasoning is distinguished from *theoretical reasoning* – theoretical reasoning is directed towards beliefs

## Theoretical Reasoning

If I believe that all men are mortal, and If I believe that Socrates is a man, then I will usually conclude that Socrates is mortal

## Practical reasoning

Deciding to catch a bus instead of a train

We can use both: after using theoretical reasoning we have acquired more knowledge to do practical reasoning. And viceversa I can do before practical reasoning (that give me knowledge from experience) and then Theoretical reasoning.

# Practical Reasoning

- Human practical reasoning consists of two activities:
  - **deliberation**  
deciding **what** state of affair we want to achieve
  - **means-ends reasoning**  
deciding **how** to achieve these states of affairs
- Once the goal has been chosen, employ knowledge and reasoning to find out how to reach it
  - The outputs of deliberation are **intentions**

# Intentions in Practical Reasoning

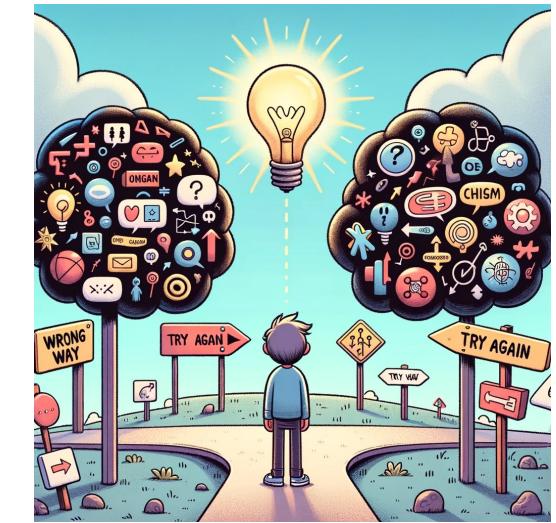
- Intentions pose problems for agents, who need to determine ways of achieving them - *If I have an intention to  $\phi$ , you expect me to devote resources to deciding how to bring about  $\phi$*

Ex: Alice **intends** to run a marathon in six months. This intention creates a **problem** for her: she now needs to figure out **how** to achieve this goal

Since Alice has this intention, others (like her friends or coach) would **expect** her to take concrete steps to make it happen, such as:

- **Planning a training schedule** (e.g., running 4 times a week).
- **Investing in proper running gear** (e.g., buying good running shoes).
- **Adjusting her diet** to build endurance.
- **Finding a coach or training group** for motivation.
- **Setting smaller milestone goals** (e.g., first a 5K, then a 10K).

If Alice **truly** intends to run the marathon, but does nothing to prepare, others might question whether she really meant it.



# Intentions in Practical Reasoning

- Intentions provide a “**filter**” for adopting other intentions, which must not conflict - *If I have an intention to  $\phi$ , you will not expect me to adopt an intention  $\psi$  such that  $\phi$  and  $\psi$  are mutually exclusive*
  - Alice also considers adopting a new intention “**I want to party late every weekend and stay up until 3 AM**”, which conflicts with her marathon training:
    - Late-night partying would **reduce sleep quality**, affecting recovery.
    - Drinking alcohol or unhealthy eating could **hinder her performance**.
    - Staying out late may **interfere with morning training sessions**Since Alice intends to train for the marathon, she **filters out** excessive partying, as both cannot coexist.

# Intentions in Practical Reasoning

- Agents track the success of their intentions, and are inclined to try again if their attempts fail - *If an agent's first attempt to achieve  $\phi$  fails, then all other things being equal, it will try an alternative plan to achieve  $\phi$* 
  - After a few weeks, Alice **fails to complete a long-distance run** due to exhaustion and muscle pain.
  - Alice's goal (running the marathon) **remains unchanged**, she **modifies her approach** and tries again using a different strategy
    - **Adjusting her training plan** (e.g., incorporating more rest days or cross-training).
    - **Improving her nutrition and hydration** to sustain endurance.
    - **Seeking advice from a coach** to correct her running form.
    - **Trying a different running schedule** (e.g., training in the evening instead of the morning).

# Intentions in Practical Reasoning

- Agents believe their intentions are possible - *That is, they believe there is at least some way that the intentions could be brought about*
  - Alice trusts training programs, sees others succeed, and knows she can improve with effort. If she thought it was impossible, she wouldn't commit to it.
- Agents do not believe they will not bring about their intentions - *It would not be rational of me to adopt an intention to  $\phi$  if I believed  $\phi$  was not possible*
  - Alice intends to run a marathon because she doesn't believe she will fail (Eg., She believes proper training will prepare her, she sees no definitive reason why she cannot succeed, she knows challenges may arise but believes she can adapt).
- Under certain circumstances, agents believe they will bring about their intentions - *It would not normally be rational of me to believe that I would bring my intentions about; intentions can fail. Moreover, it does not make sense that if I believe  $\phi$  is inevitable that I would adopt it as an intention*
  - Alice believes she can run the marathon, but she also recognizes that success is not guaranteed. Training injuries, unexpected life events, or sheer exhaustion could still cause her to fail.



# Intentions in Practical Reasoning

- Agents need not intend all the expected side effects of their intentions.

*If I believe  $\phi \rightarrow \psi$  and I intend that  $\phi$ , I do not necessarily intend  $\psi$  also. (Intentions are not closed under implication.)*

- This last problem is known as the **side effect** or **package deal** problem. I may believe that going to the dentist involves pain, and I may also intend to go to the dentist — but this does not imply that I intend to suffer pain!
- Alice **intends to run a marathon**, knowing that this commitment comes with certain **side effects**—some of which she may not desire:
  - She **expects** that intense training may cause soreness, exhaustion, and even occasional injuries
  - She **believes** marathon preparation will require waking up early and sacrificing social outingsshe **does not intend** to suffer pain or give up time with friends—these are merely **side effects** of her main intention



# Intentions in Practical Reasoning

- Notice that intentions are much stronger than mere desires:

*“My desire to play basketball this afternoon is merely a potential influencer of my conduct this afternoon. It must vie with my other relevant desires [ . . . ] before it is settled what I will do. In contrast, once I intend to play basketball this afternoon, the matter is settled: I normally need not continue to weigh the pros and cons. When the afternoon arrives, I will normally just proceed to execute my intentions”* (Bratman, 1990)



# Complications

- Practical reasoning has to be implemented as a computational process.
- **Resource bounds:**
  - available memory;
  - time constraints.
- Implications:
  - computation is an important resource, that impacts on performance;
  - agents cannot deliberate indefinitely: they have to commit to one state of affairs, which may not be optimal

# Intentions

**Definition:** Intentions are the **states** of affairs that an agent has chosen and committed to

- Intentions as *pro-attitudes*: they tend to lead to actions
- Intentions *persist*
- Intentions *constrain* practical reasoning
- Related to beliefs about the future.
- Therefore, intentions *interact* with agent's beliefs and other mental states



# Representation

- Agents keep an explicit representation of B,D,I
- No assumption on how such information is represented

B agent's current beliefs,  $\text{Bel}$  set of all such beliefs ( $B \subseteq \text{Bel}$ )

D agent's current desires,  $\text{Des}$  set of all such desires ( $D \subseteq \text{Des}$ )

I agent's current intentions,  $\text{Int}$  set of all such intentions ( $I \subseteq \text{Int}$ )

# Deliberation

- **Option generation:** an agent generates a set of options among the possible desires:
  - An agent's option generation function maps the agent's current beliefs and intentions to the agent's new desires:

Option:  $\text{Bel} \times \text{Int} \rightarrow \text{Des}$

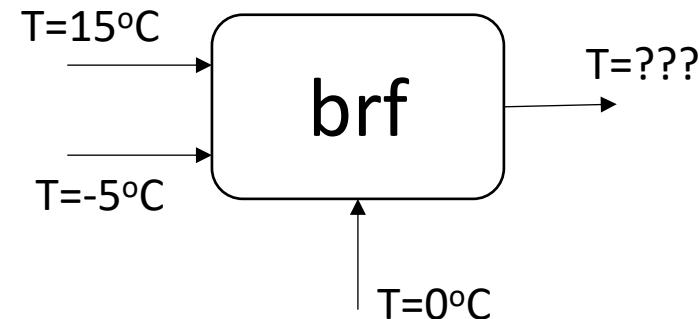
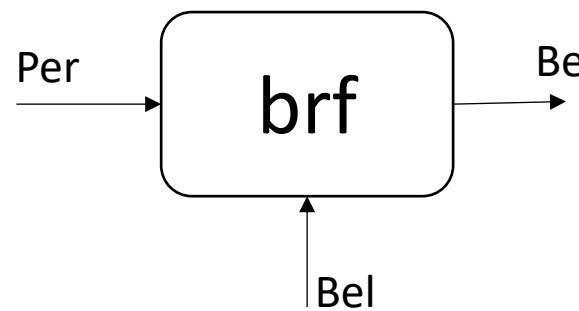
- **Option filtering:** an agent chooses among the possible options, committing to the best:
  - An agent's filtering function maps the agent's current beliefs, desires and intentions to a set of intentions

Filter:  $\text{Bel} \times \text{Des} \times \text{Int} \rightarrow \text{Int}$

# Belief Revision

- An agent updates its current beliefs when it perceives new information from the environment
  - An agent's belief revision function maps the agent's current beliefs and the current percept to the agent's new beliefs

belief revision function  
brf: Bel  $\times$  Per  $\rightarrow$  Bel



# Means-end reasoning

- The process of deciding how to achieve an *end* using the available *means*
- Known in AI as *planning*

Input:

- A *goal, intention or task*: a state of affair that the agent wants to achieve or maintain
- The current *state of the environment*: the agent's beliefs
- A set of *actions* available to the agent.

Output:

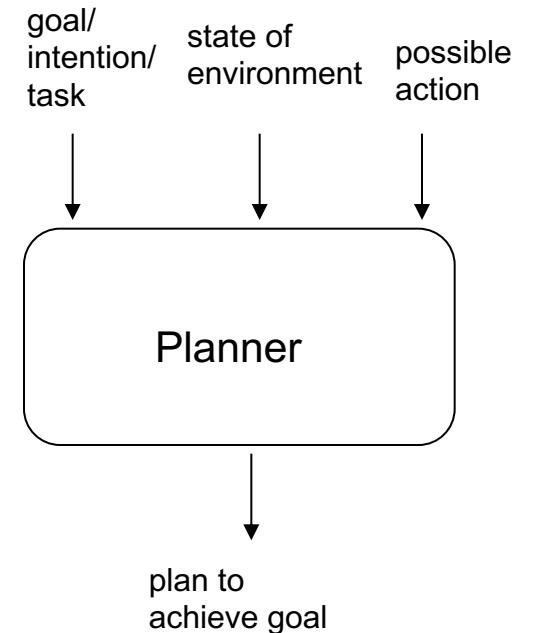
- A sequence of actions, supposed to produce the desired state of affairs.

# What is Means-End Reasoning?

- Basic idea is to give an agent:
  - representation of goal/intention to achieve
  - representation actions it can perform
  - representation of the environment

and have it generate a **plan** to achieve the goal

- *automatic programming*



# Implementing Practical Reasoning Agents

- A first pass at an implementation of a practical reasoning agent:

```
Agent Control Loop Version 1
```

1. while true
2.     observe the world;
3.     update internal world model;
4.     deliberate about what intention to achieve next;
5.     use means-ends reasoning to get a plan for the intention;
6.     execute the plan
7. end while

# Implementing Practical Reasoning Agents

- Let's make the algorithm more formal:

```
Agent Control Loop Version 2
1.  $B := B_0$ ; /* initial beliefs */
2. while true do
3.     get next percept  $\rho$ ;
4.      $B := brf(B, \rho)$ ;
5.      $I := deliberate(B)$ ;
6.      $\pi := plan(B, I)$ ;
7.      $execute(\pi)$ 
8. end while
```

The environment  
does not change

# Deliberation

- How does an agent deliberate?
  - begin by trying to understand what the *options* available to you are
  - **choose between them**, and **commit** to some
- Two distinct functional components:
  - **option generation**  
in which the agent generates a set of possible alternatives;  
Represent option generation via a function, *options*, which takes the agent's current beliefs and current intentions, and from them determines a set of options (= *desires*)
  - **filtering**  
in which the agent chooses between alternatives and commits to achieving them.  
In order to select between competing options, an agent uses a **filter** function.

# Deliberation

In an ideal world,  
an agent would like  
all its desires achieved

Agent Control Loop Version 3

```
1.  
2.    $B := B_0;$   
3.    $I := I_0;$   
4.   while true do  
5.       get next percept  $\rho$ ;  
6.        $B := brf(B, \rho);$   
7.        $D := options(B, I);$   
8.        $I := filter(B, D, I);$   
9.        $\pi := plan(B, I);$   
10.       $execute(\pi)$   
11.  end while
```

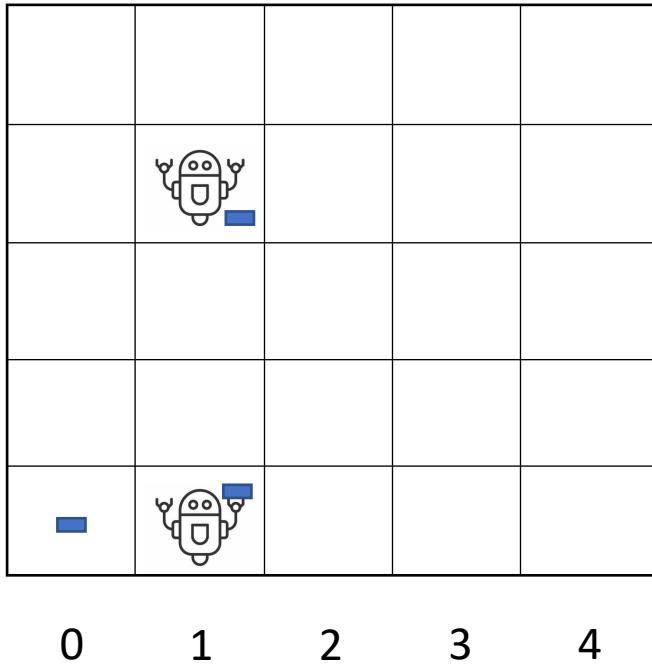
Selects the best  
option for the  
agent to commit to

# Commitment Strategies

- Commitment strategies:

- Blind commitment

A blindly committed agent will continue to maintain an intention until it believes the intention has actually been achieved. Blind commitment is also sometimes referred to as **fanatical** commitment.



### Blind commitment

```
-- t=0
B = {In(1,0), carry(pack_1)}
I = {}
P = {}
Do: Null

-- t=1
B = {In(1,0}, carry(pack_1)}
I = {In(1,3,pack_1)}
P={move(1,1), move(1,2), move(1,3), put_down(pack_1)}
Do: Null

-- t=2
B = {In(1,0), carry(pack_1), In(0,0,pack_2)}
I = {In(1,3,pack_1)}
P = {move(1,2), move(1,3), put_down(pack_1)}
Do: move(1,1)

-- t=3
B = {In(1,1), carry(pack_1), In(0,0,pack_2)}
I = {In(1,3,pack_1)}
P = {move(1,3), put_down(pack_1)}
Do: move(1,2)

-- t=4
B = {In(1,2), carry(pack_1), In(0,0,pack_2)}
I = {In(1,3,pack_1)}
P = {put_down(pack_1)}
Do: move(1,3)
```

```
-- t=0
B = {In(1,0), carry(pack_1)}
I = {}
P = {}
Do: Null

-- t=1
B = {In(1,0}, carry(pack_1)}
I = {In(1,3,pack_1)}
P={move(1,1), move(1,2), move(1,3), put_down(pack_1)}
Do: Null

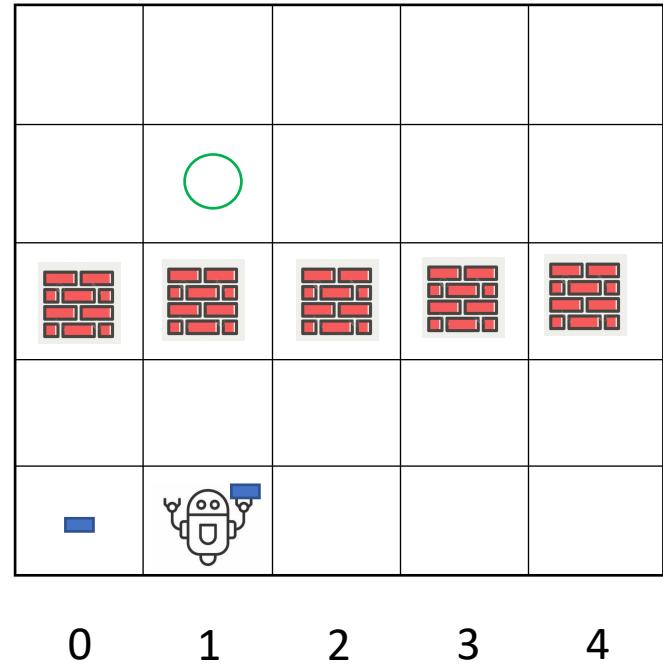
-- t=2
B = {In(1,0), carry(pack_1), In(0,0,pack_2)}
I = {In(1,3,pack_1)}
P = {move(1,2), move(1,3), put_down(pack_1)}
Do: move(1,1)

-- t=3
B = {In(1,1), carry(pack_1), In(0,0,pack_2)}
I = {In(1,3,pack_1)}
P = {move(1,3), put_down(pack_1)}
Do: move(1,2)

-- t=4
B = {In(1,2), carry(pack_1), In(0,0,pack_2)}
I = {In(1,3,pack_1)}
P = {put_down(pack_1)}
Do: move(1,3)

-- t=5
B = {In(1,3), carry(pack_1), In(0,0,pack_2)}
I = {In(1,3,pack_1)}
P = {}
Do: put_down(pack_1)

-- t=6
B = {In(1,3), In(1,3,pack_2), In(0,0,pack_2)}
I = {}
P = {}
Do: Null
```



```
-- t=0
B = {In(1,0), carry(pack_1)}
I = {}
P = {}
Do: Null

-- t=1
B = {In(1,0}, carry(pack_1)}
I = {In(1,3,pack_1)}
P={move(1,1), move(1,2), move(1,3), put_down(pack_1)}
Do: Null

-- t=2
B = {In(1,0), carry(pack_1), In(0,0,pack_2)}
I = {In(1,3,pack_1)}
P = {move(1,2), move(1,3), put_down(pack_1)}
Do: move(1,1)
```

```
-- t=3
B = {In(1,1), carry(pack_1), In(0,0,pack_2)}
I = {In(1,3,pack_1)}
P = {move(1,3), put_down(pack_1)}
Do: move(1,2)
```

```
-- t=4
B = {In(1,1), carry(pack_1), In(0,0,pack_2), block(2)}
I = {In(1,3,pack_1)}
P = {move(1,3), put_down(pack_1)}
Do: move(1,2)
```

- 
- The initial intention is not anymore possible

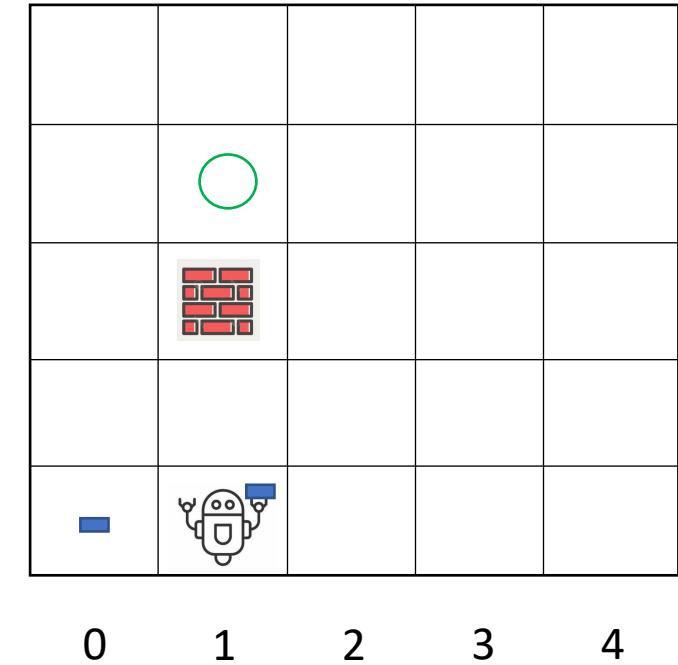
**Blind commitment does not work**

# Commitment Strategies

- An agent has commitment both to **ends** (i.e., the wishes to bring about), and **means** (i.e., the mechanism via which the agent wishes to achieve the state of affairs)
- If a plan goes wrong
  - **Replan** – try another plan keeping the commitment

### Agent Control Loop Version 4

```
1.  
2.    $B := B_0;$   
3.    $I := I_0;$   
4.   while true do  
5.       get next percept  $\rho$ ;  
6.        $B := brf(B, \rho);$   
7.        $D := options(B, I);$   
8.        $I := filter(B, D, I);$   
9.        $\pi := plan(B, I);$   
10.      while not empty( $\pi$ ) do  
11.           $\alpha := hd(\pi);$   
12.          execute( $\alpha$ );  
13.           $\pi := tail(\pi);$   
14.          get next percept  $\rho$ ;  
15.           $B := brf(B, \rho);$   
16.          if not sound( $\pi, I, B$ ) then  
17.               $\pi := plan(B, I)$   
18.          end-if  
19.      end-while  
20.  end-while
```



Replan

```

-- t=0
B = {In(1,0), carry(pack_1)}
I = {}
P = {}
Do: Null

-- t=1
B = {In(1,0}, carry(pack_1)}
I = {In(1,3,pack_1)}
P={move(1,1), move(1,2), move(1,3), put_down(pack_1)}
Do: Null

-- t=2
B = {In(1,0), carry(pack_1), In(0,0,pack_2)}
I = {In(1,3,pack_1)}
P = {move(1,2), move(1,3), put_down(pack_1)}
Do: move(1,1)

-- t=3
B = {In(1,1), carry(pack_1), In(0,0,pack_2)}
I = {In(1,3,pack_1)}
P = {move(1,3), put_down(pack_1)}
Do: move(1,2)           -  
-  
-  
-

-- t=4
B = {In(1,1), carry(pack_1), In(0,0,pack_2), block(1,2)}
I = {In(1,3,pack_1)}
P = {move(0,1), move(0,2), move(0,3), move(1,3), put_down(pack_1)}
Do: Null

```

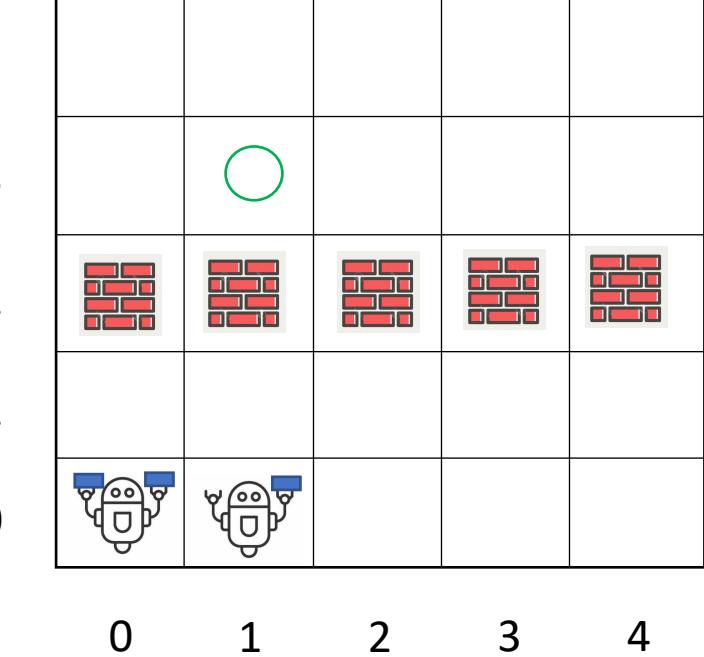
# Commitment Strategies

- Commitment strategies:
  - Still overcommitted to intentions
    - never stops to consider whether or not its intentions are appropriate
  - Modification: stop to determine whether intentions have succeeded or whether they are impossible
  - Single-minded commitment

A single-minded agent will continue to maintain an intention until it believes that either the intention has been achieved, or else that it is no longer possible to achieve the intention

### Agent Control Loop Version 5

```
2.    $B := B_0;$ 
3.    $I := I_0;$ 
4.   while true do
5.       get next percept  $\rho$ ;
6.        $B := brf(B, \rho);$ 
7.        $D := options(B, I);$ 
8.        $I := filter(B, D, I);$ 
9.        $\pi := plan(B, I);$ 
10.      while not empty( $\pi$ )
11.          or succeeded( $I, B$ )
12.          or impossible( $I, B$ )) do
13.               $\alpha := hd(\pi);$ 
14.              execute( $\alpha$ );
15.               $\pi := tail(\pi);$ 
16.              get next percept  $\rho$ ;
17.               $B := brf(B, \rho);$ 
18.              if not sound( $\pi, I, B$ ) then
19.                   $\pi := plan(B, I)$ 
20.              end-if
21.      end-while
22.  end-while
```



## Single-minded commitment

```
-- t=3
B = {In(1,0), carry(pack_1), In(0,0,pack_2)}
I = {In(1,3,pack_1)}
P = {move(1,3), put_down(pack_1)}
Do: move(1,2)
```

```
-- t=4
B = {In(1,1), carry(pack_1), In(0,0,pack_2), block(2)}
I = {}
P = {}
Do: Null
```

```
-- t=0
B = {In(1,0), carry(pack_1)}
I = {}
P = {}
Do: Null

-- t=1
B = {In(1,0}, carry(pack_1)}
I = {In(1,3,pack_1)}
P={move(1,1), move(1,2), move(1,3), put_down(pack_1)}
Do: Null

-- t=2
B = {In(1,0), carry(pack_1), In(0,0,pack_2)}
I = {In(1,3,pack_1)}
P = {move(1,2), move(1,3), put_down(pack_1)}
Do: move(1,1)
```

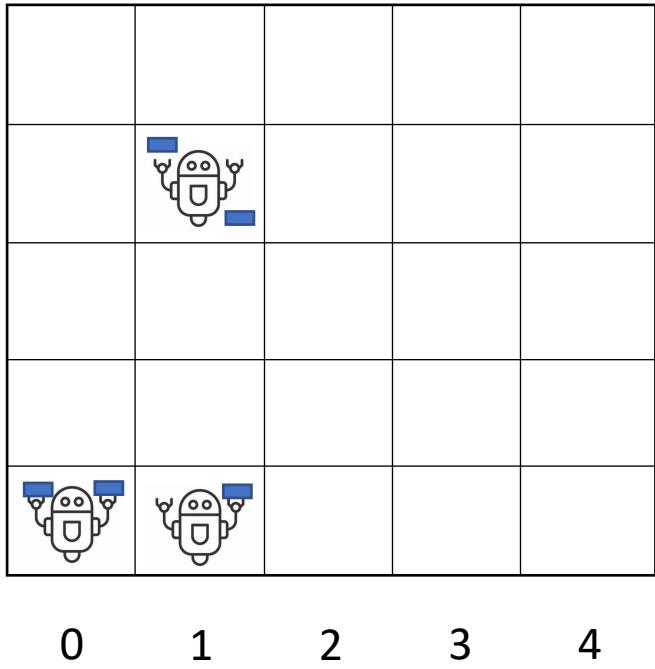
```
-- t=5
B = {In(1,1), carry(pack_1), In(1,0,pack_2)}
I = {carry(pack_2)}
P = {move(0,1), move(0,0), pick_up(pack_2)}
Do: Null
```

# Intention Reconsideration

- Our agent gets to reconsider its intentions once every time around the outer control loop, i.e., when:
  - it has completely executed a plan to achieve its current intentions; or
  - it believes it has achieved its current intentions; or
  - it believes its current intentions are no longer possible.
- This is limited in the way that it permits an agent to *reconsider* its intentions
  - In a competitive environment the agent could decide to `pick_up(pack_2)` before `deliver(pack_1)`
  - What about if `pack_3` appear?
- **Reconsider intentions after executing every action**

### Agent Control Loop Version 6

```
1.  
2.    $B := B_0;$   
3.    $I := I_0;$   
4.   while true do  
5.       get next percept  $\rho$ ;  
6.        $B := brf(B, \rho);$   
7.        $D := options(B, I);$   
8.        $I := filter(B, D, I);$   
9.        $\pi := plan(B, I);$   
10.      while not (empty( $\pi$ )  
           or succeeded( $I, B$ )  
           or impossible( $I, B$ )) do  
11.           $\alpha := hd(\pi);$   
12.          execute( $\alpha$ );  
13.           $\pi := tail(\pi);$   
14.          get next percept  $\rho$ ;  
15.           $B := brf(B, \rho);$   
16.           $D := options(B, I);$   
17.           $I := filter(B, D, I);$   
18.          if not sound( $\pi, I, B$ ) then  
19.               $\pi := plan(B, I)$   
20.          end-if  
21.      end-while  
22.  end-while
```



Intentions  
reconsideration

```
-- t=0
B = {In(1,0), carry(pack_1)}
I = {}
P = {}
Do: Null

-- t=1
B = {In(1,0}, carry(pack_1)}
I = {In(1,3,pack_1)}
P={move(1,1), move(1,2), move(1,3), put_down(pack_1)}
Do: Null

-- t=2
B = {In(1,0), carry(pack_1), In(0,0,pack_2)}
I = {In(1,3,pack_1)}
P = {move(1,2), move(1,3), put_down(pack_1)}
Do: move(1,1)

-- t=3
B = {In(1,1), carry(pack_1), In(0,0,pack_2)}
I = {carry(pack_2), In(1,3,pack_1)}
P= {move(1,0), move(0,0), pick_up(pack_2), move(1,1),
     move(1,2), move(1,3), put_down(pack_1)}
Do: Null

-- t=4
B = {In(1,1), carry(pack_1), In(0,0,pack_2)}
I = {carry(pack_2), In(1,3,pack_1)}
P= {move(0,0), pick_up(pack_2), move(1,1),  move(1,2),
     move(1,3), put_down(pack_1)}
Do: move(1,0).....
```

# Intention Reconsideration

- But intention reconsideration is *costly*!

A dilemma:

- an agent that does not stop to reconsider its intentions sufficiently often will continue attempting to achieve its intentions even after it is clear that they cannot be achieved, or that there is no longer any reason for achieving them
  - an agent that *constantly* reconsiders its intentions may spend insufficient time actually working to achieve them, and hence runs the risk of never actually achieving them
- **Solution:** incorporate an explicit **meta-level control** component, that decides whether or not to reconsider

### Agent Control Loop Version 7

```
1.  
2.    $B := B_0;$   
3.    $I := I_0;$   
4.   while true do  
5.       get next percept  $\rho$ ;  
6.        $B := brf(B, \rho);$   
7.        $D := options(B, I);$   
8.        $I := filter(B, D, I);$   
9.        $\pi := plan(B, I);$   
10.      while not ( $empty(\pi)$   
           or  $succeeded(I, B)$   
           or  $impossible(I, B)$ ) do  
11.           $\alpha := hd(\pi);$   
12.           $execute(\alpha);$   
13.           $\pi := tail(\pi);$   
14.          get next percept  $\rho$ ;  
15.           $B := brf(B, \rho);$   
16.          if reconsider( $I, B$ ) then at the end of first action  
17.               $D := options(B, I);$   
18.               $I := filter(B, D, I);$   
19.          end-if  
20.          if not sound( $\pi, I, B$ ) then  
21.               $\pi := plan(B, I)$   
22.          end-if  
23.      end-while  
24.  end-while
```

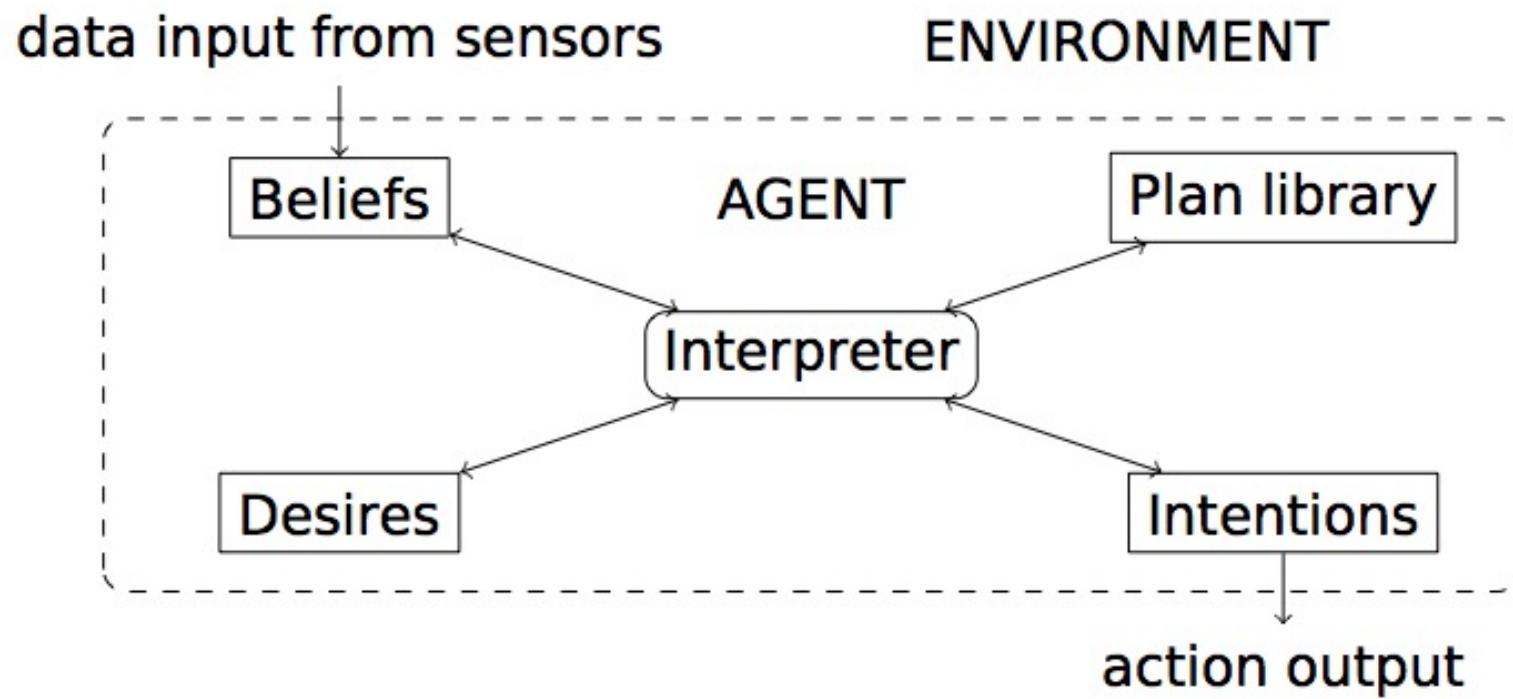
# Optimal Intention Reconsideration

- Examples of reconsideration strategy (Kinny and Georgeff):
  - **bold** agents: never pause to reconsider intentions, and
  - **cautious** agents: stop to reconsider after every action
- **Dynamism** in the environment is represented by the **rate of world change (g)**
- Results (not surprising):
  - If  $g$  is low (i.e., the environment does not change quickly), then bold agents do well compared to cautious ones. This is because cautious ones waste time reconsidering their commitments while bold agents are busy working towards — and achieving — their intentions.
  - If  $g$  is high (i.e., the environment changes frequently), then cautious agents tend to outperform bold agents. This is because they are able to recognize when intentions are doomed, and also to take advantage of serendipitous situations and new opportunities when they arise.

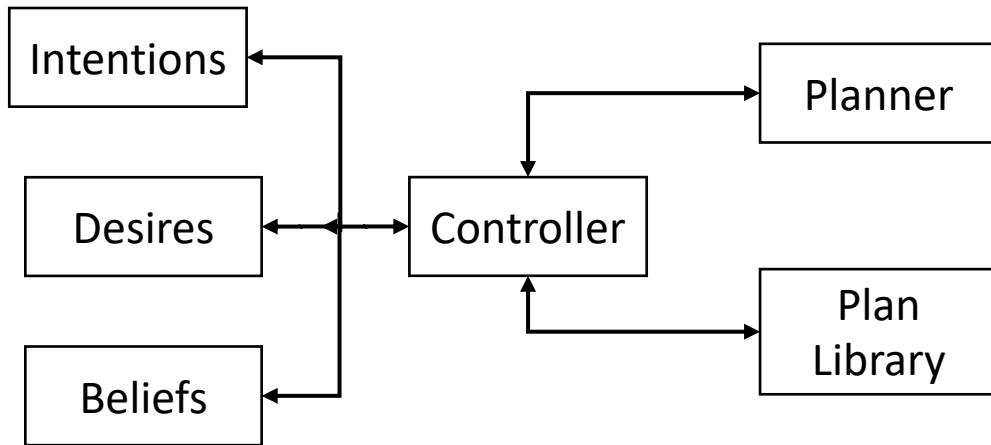
# The Procedural Reasoning System

- Perhaps the first and most influential BDI agent architecture (Georgeff, Lansky)
  - each agent is equipped with a **plan library**, representing that agent's **procedural knowledge**
  - knowledge about the mechanisms that can be used by the agent in order to realize its intentions
- Peculiar management of plans:
  - plans are not built by the agent, but selected from a hand-written library;
  - plans have a precondition (context) and post-condition (goal);
  - The plan's body may contain not only actions, but also goals
  - The options available to an agent are directly determined by the plans an agent has

# PRS architecture



# Combining Procedural and Planning

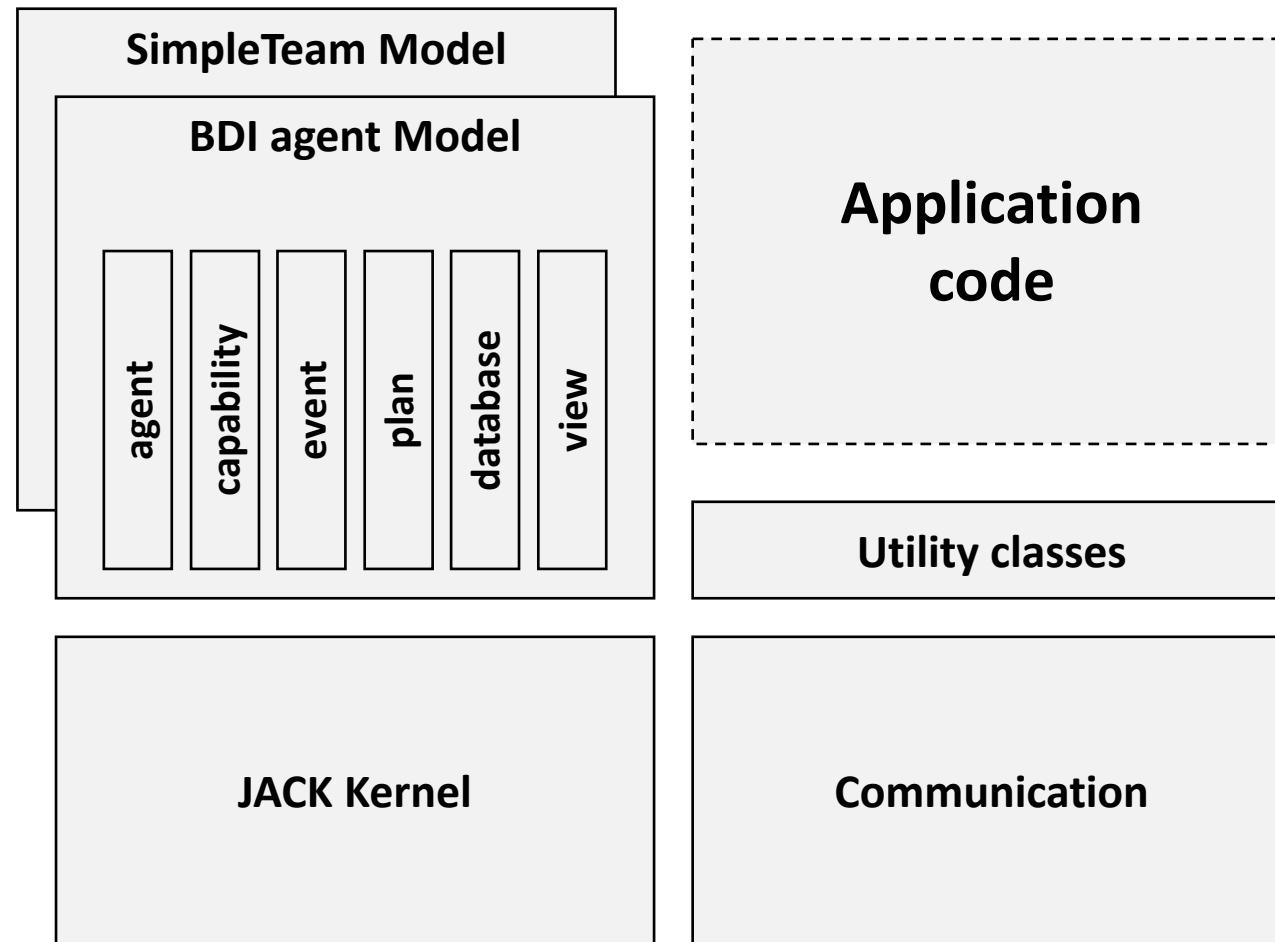


- **Plan Library**
  - Fast (re)planning -> reactive
  - Limited predefined plans
- **Planning**
  - No predefined plans
  - Time consuming

# JACK

- JACK is an agent development environment produced by the Agent Oriented Software Group - first released in 1998 - <http://www-aosgrp.com>
- There are two principles underpinning the development of JACK
  - operates on top of the Java programming language, acting as an extension that provides agent-related concepts
  - based on the Belief-Desire-Intention architecture
- The development environment has three main components:
  - The **JACK Agent Language** is a superset of the Java language, and introduces new semantic and syntactic features, new base classes, interfaces, and methods to deal with agent-oriented concepts
  - The **JACK Compiler** compiles the JACK Agent Language down to pure Java, so that the resulting agents can operate on any Java platform
  - The **JACK Agent Kernel** is the runtime program within which JACK agents operate, and provides the underlying agent functionality that is defined within the JACK Agent Language

# Jack architecture



# JACK - Agents

- Agents schedule actions using the **TaskManager**
- **Plans** are sequences of actions that agents execute on recording an event
  - No automated planning !
- **Events** within the agent architecture are divided into:
  - external events (such as messages from other agents);
  - internal events initiated by the agent itself;
  - and motivations (which are described as goals that the agent wants to achieve)
- **Capabilities** provide a means for structuring a set of reasoning elements into a coherent cluster that can be plugged into agents

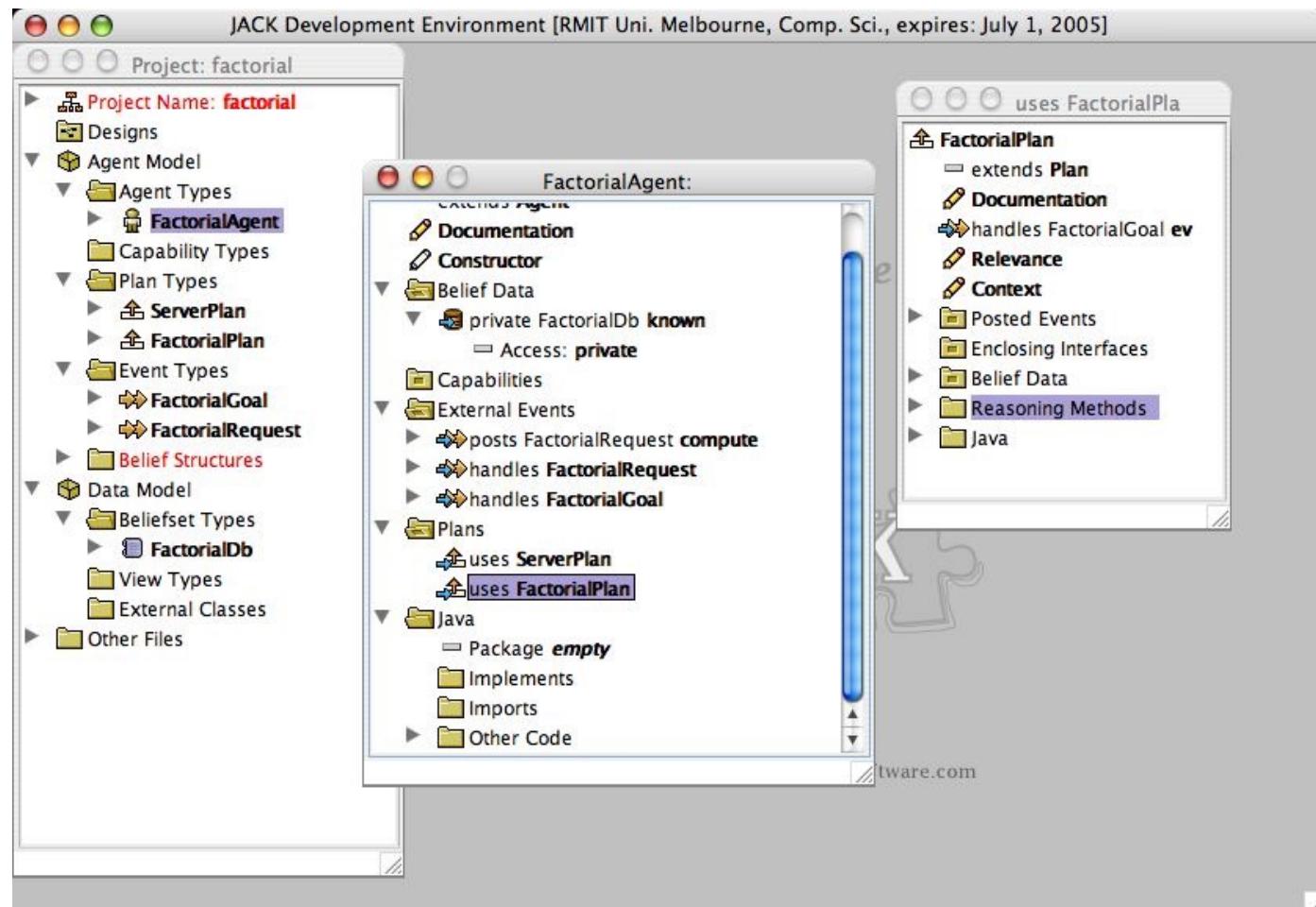
# JACK – Multi-agent systems

- Networking capabilities in JACK are based on UDP over IP, with a thin layer of management on top of that to provide reliable peer-to-peer communication
- Agent communication between agents is handled by the JACK Kernel, which handles the routing of messages and the interface with lower-level networking infrastructure
- A rudimentary Agent Name Server is provided
- FIPA ACL is supported (IEEE standard)

# JACK – Supporting Software

- JACK provides a comprehensive, graphical agent development environment
  - A high-level design tool allows a multi-agent system application to be designed by defining the agents and relationships between them, in a notation similar to UML
  - A plan editor allows plans to be specified as decision diagrams
  - A plan tracing tool and an agent interaction tool allow developers to visualize the monitoring of an application
- An application can be monitored through an Agent Tracing Controller, which allows a developer to choose which agents to trace and provides a visual representation of the agents stepping through their plans

# Jack - JDE



# JAVA Agent DEvelopment Framework

is an open source platform for peer-to-peer agent based applications

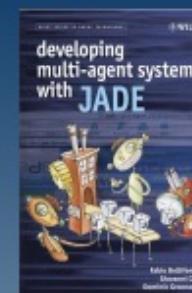
JADE (Java Agent DEvelopment Framework) is a software Framework fully implemented in the Java language. It simplifies the implementation of multi-agent systems through a middle-ware that complies with the [FIPA specifications](#) and through a set of [graphical tools](#) that support the debugging and deployment phases. A JADE-based system can be distributed across machines (which not even need to share the same OS) and the configuration can be controlled via a [remote GUI](#). The configuration can be even changed at run-time by moving agents from one machine to another, as and when required. JADE is completely implemented in Java language and the minimal system requirement is the version 5 of JAVA (the run time environment or the JDK).

Besides the **agent abstraction**, JADE provides a simple yet powerful **task execution** and composition model, peer to peer agent **communication** based on the asynchronous message passing paradigm, a **yellow pages** service supporting publish subscribe discovery mechanism and many other advanced features that facilitates the development of a distributed system.

## Latest news

- JADE 4.6.0 and WADE 3.8.0 have been released  
19/12/2022
- JADE 4.5 and WADE 3.6 have been released  
08/06/2017

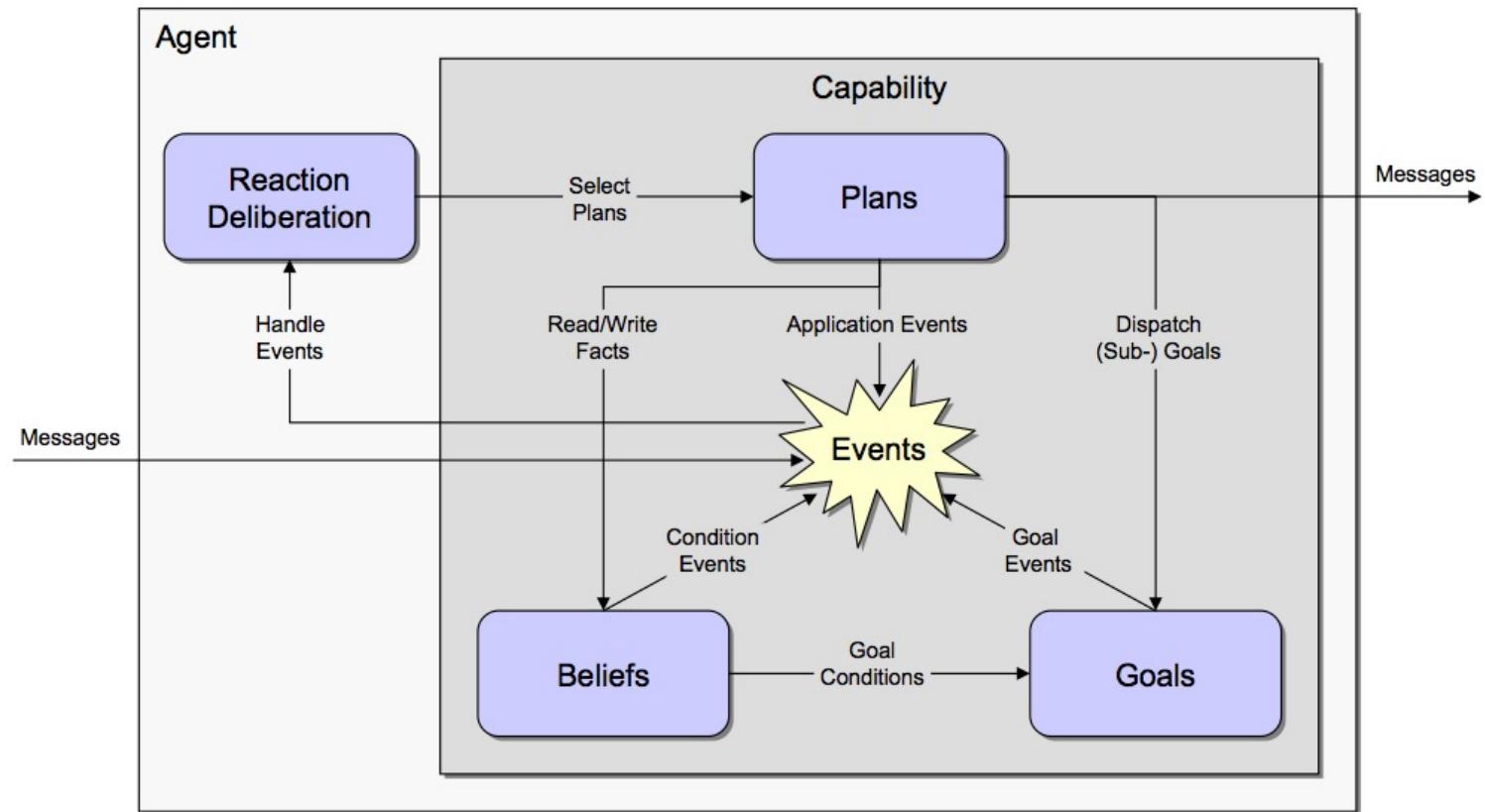
## Jade Book



Developing Multi-Agent Systems with JADE can be ordered from [Wiley](#)

# Jadex

- Jadex is based on JADE
  - Explicit representation of goals allows reasoning about goals



# Jadex goals

- Generic goal types
  - perform (some action)
  - achieve (a specified world state)
  - query (some information)
  - maintain (reestablish a specified world state whenever violated)
- Are strongly typed with
  - name, type, parameters
  - BDI-flags enable non-default goal-processing
- Goal creation/deletion possibilities
  - initial goals for agents
  - goal creation/drop conditions for all goal kinds – top-level / subgoals from within plans

# LLM Agents

Autonomous Software Agents

A.A. 2024-2025

**Prof. Paolo Giorgini**

**Dr. Marco Robol**



UNIVERSITY OF TRENTO - Italy

Department of Information  
and Communication Technology

# Question answering

Q: what is  $1 + 2$ ?



A: 3

Q: Janet's ducks lay 16 eggs per day. She eats three for breakfast every morning and bakes muffins for her friends every day with four. She sells the remainder for \$2 per egg.  
How much does she make every day?



Requires reasoning

Q: who is the latest UK PM?



Requires knowledge

Q: what is the prime factorization of 34324329?



Requires computation

# Code augmentation for computation

Question: In Fibonacci sequence, it follows the rule that each number is equal to the sum of the preceding two numbers. Assuming the first two numbers are 0 and 1, what is the 50th number in Fibonacci sequence?

The first number is 0, the second number is 1, therefore, the third number is  $0+1=1$ . The fourth number is  $1+1=2$ . The fifth number is  $1+2=3$ . The sixth number is  $2+3=5$ . The seventh number is  $3+5=8$ . The eighth number is  $5+8=13$ .  
..... (Skip 1000 tokens)  
The 50th number is 32,432,268,459.

CoT

32,432,268,459



```
length_of_fibonacci_sequence = 50  
fibonacci_sequence = np.zeros(length_of_)  
fibonacci_sequence[0] = 0  
fibonacci_sequence[1] = 1  
for i in range(3, length_of_fibonacci_sequence):  
    fibonacci_sequence[i] = fibonacci_sequence[i-1] +  
        fibonacci_sequence[i-2]  
ans = fibonacci_sequence[-1]
```

PoT



python

12,586,269,025



# Retrieval-augmented generation (RAG) for knowledge

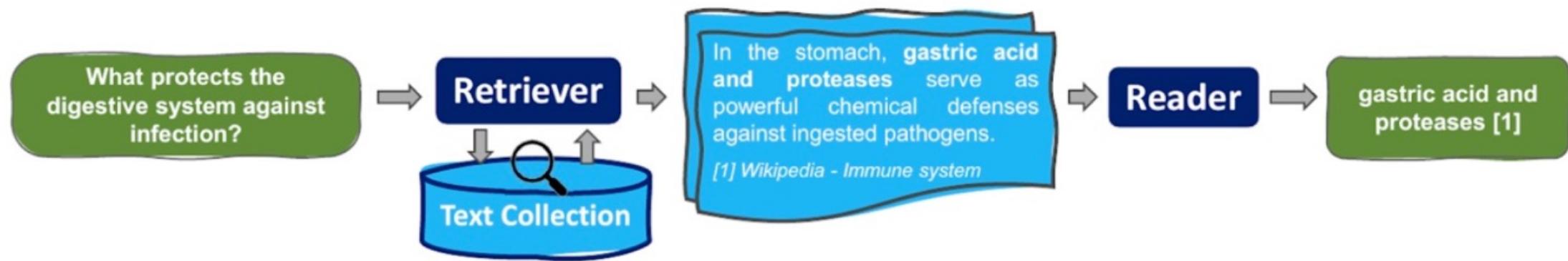


Image: <http://ai.stanford.edu/blog/retrieval-based-NLP/>

# Tool use

- Special tokens to invoke tool calls for
  - Search engine, calculator, etc.
  - Task-specific models (translation)
  - APIs
- Unnatural format requires task/tool-specific fine-tuning
- Multiple tool calls?

**A weather task:**

how hot will it get in NYC today? |**weather** lookup re-gion=NYC |**result** precipitation chance: 10, high temp: 20c, low-temp: 12c |**output** today's high will be 20C

TALM: Tool Augmented Language Models.

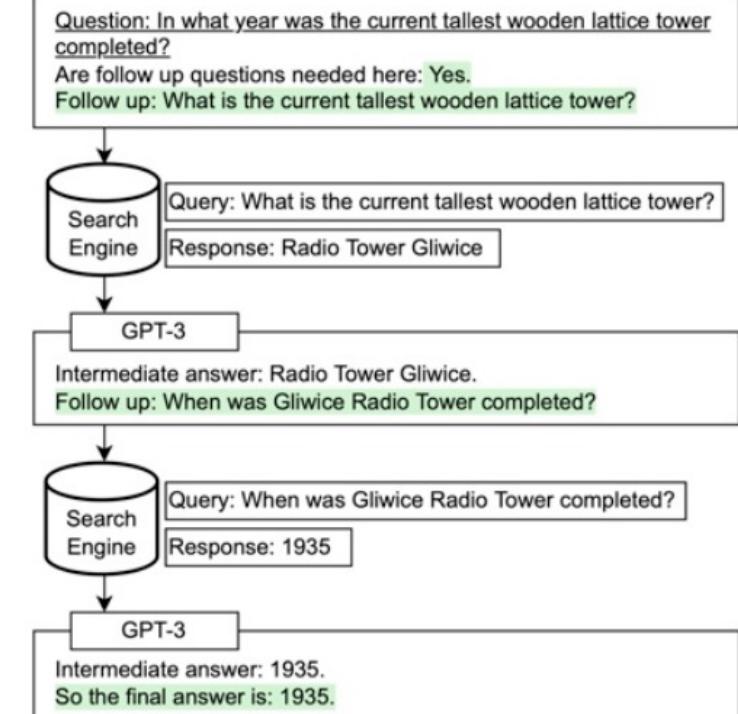
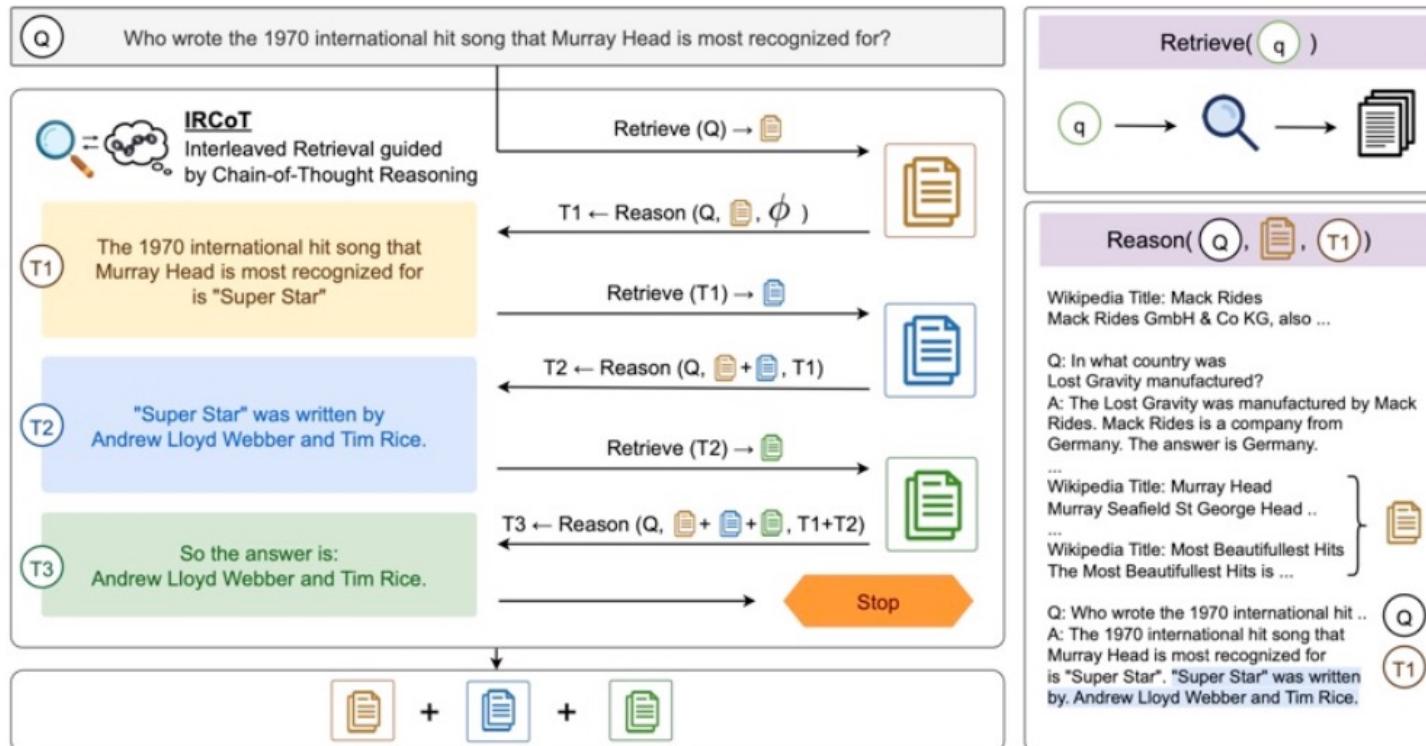
Out of 1400 participants, 400 (or [\[Calculator\(400 / 1400\) → 0.29\]](#) 29%) passed the test.

The name derives from "la tortuga", the Spanish word for [\[MT\("tortuga"\) → turtle\]](#) turtle.

The Brown Act is California's law [\[WikiSearch\("Brown Act"\) → The Ralph M. Brown Act is an act of the California State Legislature that guarantees the public's right to attend and participate in meetings of local legislative bodies.\]](#) that requires legislative bodies, like city councils, to hold their meetings open to the public.

Toolformer: Language Models Can Teach Themselves to Use Tools

# What if both knowledge and reasoning are needed?

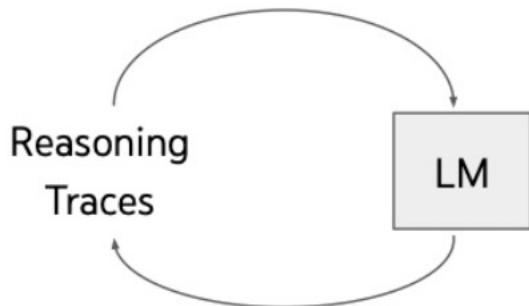


Interleaving Retrieval with Chain-of-Thought Reasoning for Knowledge-Intensive Multi-Step Questions

Measuring and Narrowing the Compositionality Gap in Language Models.

# Reasoning OR acting

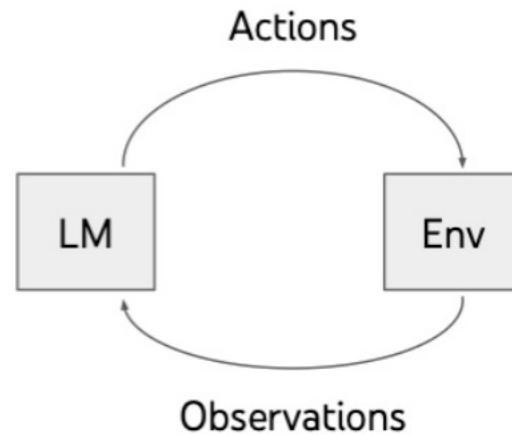
CoT



Flexible and general to augment test-time compute

Lack of external knowledge and tools

RAG/Retrieval/Code/Tool use

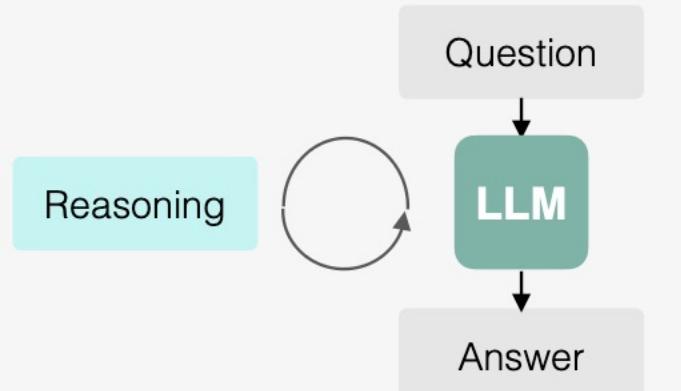


Lack of reasoning

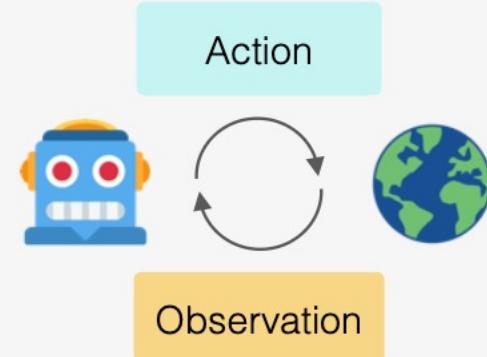
Flexible and general to augment knowledge, computation, feedback, etc.

Retrieval  
Search engine  
Calculator  
Weather API  
Python  
.....

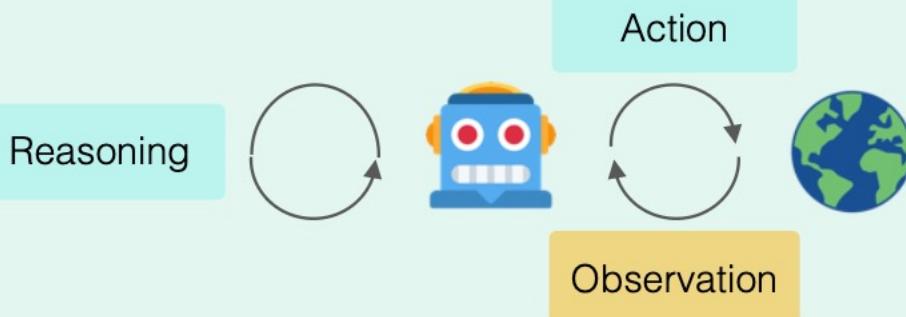
## Reasoning (update internal belief)



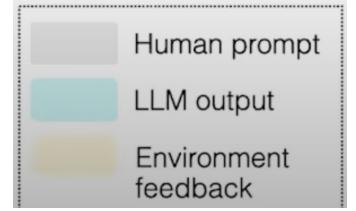
## Acting (obtain external feedback)



**ReAct:** a new paradigm of agents that **reason and act**



- **Synergy** of reasoning and acting
- **Simple** and intuitive to use
- **General** across domains



ReAct supports

- One-shot prompting
- Few-shot prompting
- Fine-tuning

## Example ReAct trajectory

Task: xxxxxxx

LLM

Thought: xxx  
Action: xxx



Observation:  
xxxxxxxx

LLM

Thought: xxx  
Action: xxx

# Zero-shot ReAct prompt

You are an agent that answers questions by using two actions:

- search[query]: Google search the query. You can also use it to calculate math.
- finish[answer]: return the answer

Your generation should have the following format:

**Thought:** to analyze the observation and inform the action.

**Action:** your action

**Question:** If I have seven trillion dollars today, can I buy Apple, Nvidia, and Microsoft? If not, how much more money do I need?

**LLM**

(GPT-4)

**Thought:** I need to find the current market capitalization of Apple, Nvidia, and Microsoft to understand how much it would cost to buy all of them.

**Action:** search["current market capitalization of Apple, Nvidia, and Microsoft"]



(Mar 2024)

**Observation:** "Nvidia's market cap stood at \$1.832 trillion as of about 10 a.m. ET Wednesday, while Alphabet's was at \$1.814 trillion, bringing Nvidia behind just Microsoft (MSFT) at \$3.015 trillion and Apple (AAPL) at \$2.836 trillion."

**Thought:** Now that I have the market cap for all companies, let's add them together and see if \$7 trillion is enough.

**Action:** search["1.832 trillion + 3.015 trillion + 2.836 trillion"]

**Observation:** 7683000000000

**Thought:** The amount of money needed in addition to the \$7 trillion is \$683 billion.

**Action:** finish["You need an additional \$683 billion to be able to buy Apple, Nvidia, and Microsoft."]



LLM

(GPT-4)

Thought: I need to find the current market capitalization of Apple, Nvidia, and Microsoft to understand how much it would cost to buy all of them.

Action: search["current market capitalization of Apple, Nvidia, and Microsoft"]

Observation: nothing found!

(My adversarial observation)

Thought: The search did not give any results. I will need to search for each company's market capitalization individually.

Action: search["current market capitalization of Apple"]

Observation: 'price': 180.75

(My adversarial observation)

Thought: This is the current price of Apple's stock, not its market capitalization. The market cap is calculated by multiplying the share price by the number of outstanding shares. Thus, I need to find the total number of Apple's outstanding shares to calculate its market cap.

Action: search["total number of Apple's outstanding shares"]



(Continues to solve the task)

Synergy: acting support reasoning, reasoning guides acting

# Acting without Reasoning

You are in the middle of a room. Looking quickly around you, you see a cabinet 6, a cabinet 1, a coffee machine 1, a countertop 3, a stove burner 1, and a toaster 1.

**Your task is to:** Put some pepper shaker on a drawer.

Act only

**Act 1:** Go to drawer 1

**Obs 1:** The drawer 1 is closed.

**Act 2:** Open drawer 1

**Obs 2:** You open the drawer 1. The drawer 1 is open. In it, you see a dishponge 2, and a spoon 1.

**Act 3:** Go to sinkbasin 1

**Obs 3:** On the sinkbasin 1, you see a dishponge 3, a spatula 1, and a spoon 2.

**Act 4:** Take peppershaker 1 from sinkbasin 1

**Obs 4:** Nothing happens.

**Act 5:** Take peppershaker 1 from sinkbasin 1

**Obs 5:** Nothing happens.

.....

X

**Cannot explore systematically or incorporate feedback**

# ReAct Enables Systematic Exploration

You are in the middle of a room. Looking quickly around you, you see a cabinet 6, a cabinet 1, a coffee machine 1, a countertop 3, a stove burner 1, and a toaster 1.

**Your task is to:** Put some pepper shaker on a drawer.

ReAct

**Act 1:** **Think**[First I need to find a peppe shaker...more likely to appear in cabinets (1-6), countertops (1-3), ... ]

**Act 2:** Go to cabinet 1

**Obs 2:** On the cabinet 1, you see a vase 2.

(Here the agent goes to cabinet 1, 2, 3, 4, 5, 6, then countertop 1 and 2)

**Act 9:** Go to countertop 3

**Obs 9:** On the countertop 3, you see a apple 1, a bread 1, ..... a pepper shaker 1, and a vase 3.

**Act 10:** Take pepper shaker 1 from countertop 3

**Obs 10:** You pick up the pepper shaker 1 from the countertop 3.

**Act 11:** **Think**[Now I find a pepper shaker 1. Next, I need to put it in/on drawer 1.]

**Act 12:** Go to drawer 1

**Obs 12:** Drawer 1 is closed.

**Act 13:** Open drawer 1

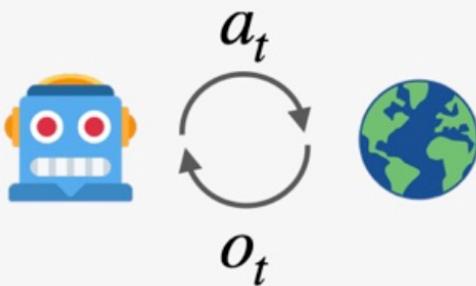
**Obs 13:** You open Drawer 1 ...

**Act 14:** Put pepper shaker 1 in/on drawer 1

**Obs 14:** You put pepper shaker 1 in/on the drawer 1.

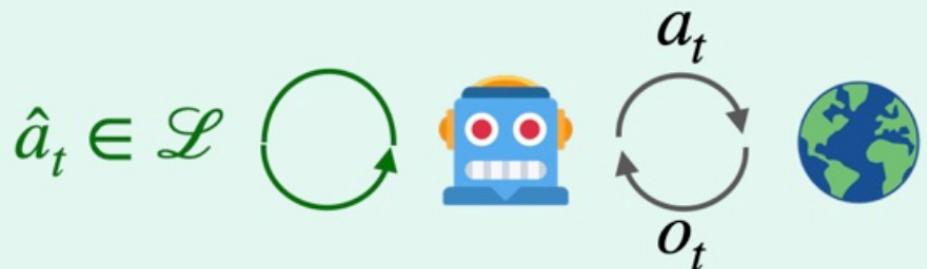


## Traditional agents: action space $A$ defined by the environment



- External feedback  $o_t$
- Agent context  $c_t = (o_1, a_1, o_2, a_2, \dots, o_t)$
- Agent action  $a_t \sim \pi(a | c_t) \in A$

**ReAct:** action space  $\hat{A} = A \cup \mathcal{L}$  augmented by reasoning



- $\hat{a}_t \in \mathcal{L}$  can be any language sequence
- Agent context  $c_{t+1} = (c_t, \hat{a}_t, a_t, o_{t+1})$
- $\hat{a}_t \in \mathcal{L}$  only updates **internal context**

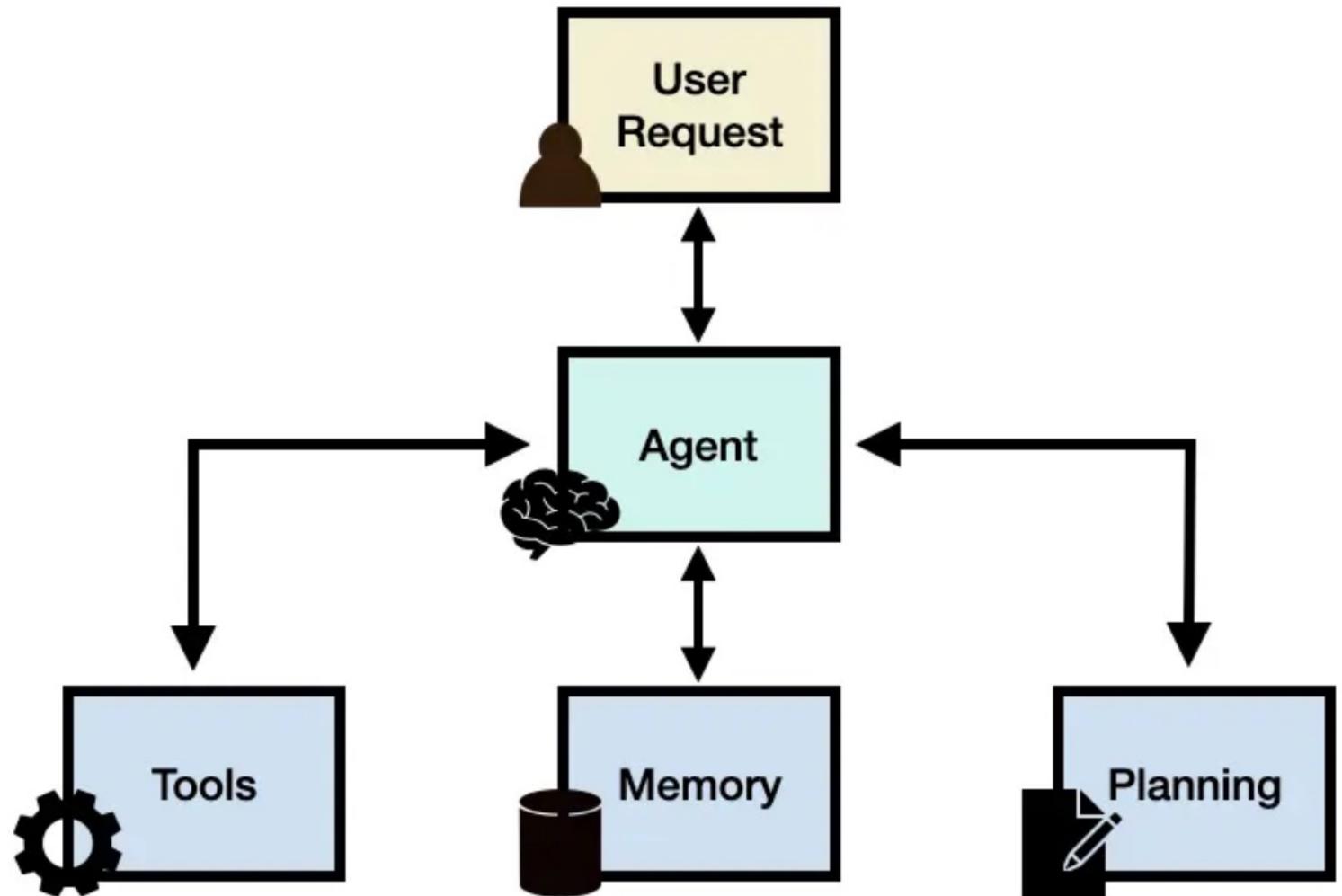
reasoning is an internal action for agents

# What are LLM agents?

- LLM-based agents (LLM agents) are **AI applications** that execute complex tasks using LLMs combined with **modules like planning, memory, and tool usage**
- **Core components**
  - **LLM as the Controller** – Acts as the 'brain' to guide operations
  - **Planning Module** – Breaks down tasks into structured steps
  - **Memory Module** – Stores and retrieves past interactions
  - **Tool Usage Module** – Interacts with external tools and APIs

# How LLM Agents Work

- The LLM processes user input and determines the required steps
- The agent uses memory to maintain context
- Planning ensures systematic execution of tasks
- The agent leverages external tools when necessary



# Agents vs Automation

## LLM Agent

- Automates complex workflows efficiently
- Adapts to various tasks with contextual understanding
- Enhances user interactions with advanced reasoning
- Handles tasks that can be done via a screen, significantly expanding automation potential

## Agents vs Atomation

- **Agents** make decisions (without manual intervention), adapt, and use memory to handle complex tasks
- **Automations are** predefined workflows that follow fixed rules without adaptability or learning
- Agents can handle dynamic and unpredictable scenarios, while automation is limited to repetitive tasks

# An example of agent vs automation

- Automation tools (e.g., Zappi, n8n) execute predefined workflows
  - If an email asks about ASA class times, automation replies and updates a FAQ
  - If something unexpected happens (e.g., wrong course name), automation fails
- An LLM agent
  - Analyses all incoming emails, extract intent, and check personal knowledge
  - Understands company guidelines, past emails, and writes responses in your tone
  - Can draft a reply for review before sending

# More than one agent

- An army of agents, each **specialized** in different tasks
  - Eg.: personal assistant, marketing assistant, finance & accounting agent, software developer
- Agents can work together to execute complex workflows

# Open AI Operator

OpenAI



search  
city  
tGPT  
  
Platform  
Business  
ies  
pany  
's

## Introducing Operator

A research preview of an agent that can use its own browser to perform tasks for you. Available to Pro users in the U.S.

Go to Operator ↗

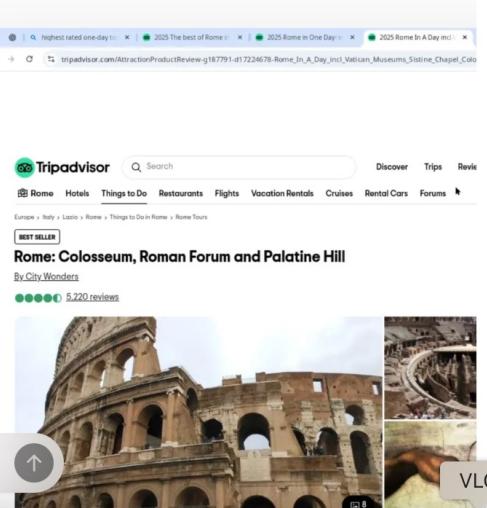
Find and book me the highest rated one-day tour of Rome on TripAdvisor.

I'll search for the highest-rated tour of historic Rome on TripAdvisor. Once I find a suitable option, I'll provide you with the details. Let's begin.

Worked for 2 minutes ^

Navigating to TripAdvisor website  
Selecting "Things to Do" category  
Searching for historic Rome tours  
Closing pop-up, continuing tour search  
Searching for Historic Rome tours  
Exploring all historic Rome tour options  
Closing Colosseum tab, resuming tour search  
Closing tour pop-up, tab afterward  
Exploring options for top-rated tours  
Sorting results by tour ratings  
Exploring filters for top-rated tours

Ask ChatGPT



Using GPT, you can also use an agent  
Instead of just answering questions, the agent takes actions on your behalf

- Example: "Find me a high-rated tour in Rome on TripAdvisor and book it."
- The agent searches, selects the best option, books it, and processes the payment.

# Same with Open Operator

B Open Operator

Close ESC

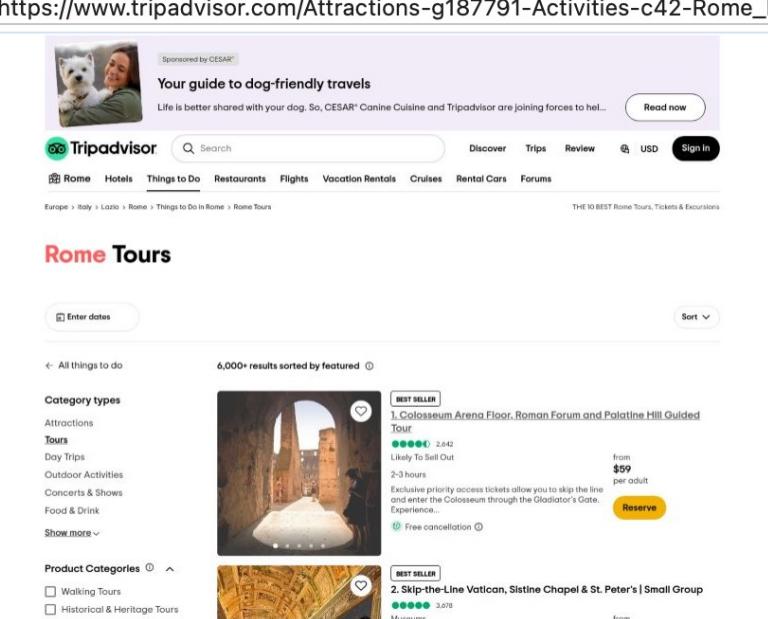
**Goal:**  
book in trip advisor an highest-rated one-day tour in Rome

Step 1 GOTO

Navigating to  
[https://www.tripadvisor.com/Attractions-g187791-Activities-c42-Rome\\_Lazio.html](https://www.tripadvisor.com/Attractions-g187791-Activities-c42-Rome_Lazio.html)

**Reasoning:** TripAdvisor is a well-known platform for booking tours and activities. By going directly to the Rome tours section on TripAdvisor, you can easily find and book the highest-rated one-day tours available in Rome.

← → ⌂ [https://www.tripadvisor.com/Attractions-g187791-Activities-c42-Rome\\_Lazio.html](https://www.tripadvisor.com/Attractions-g187791-Activities-c42-Rome_Lazio.html) ⌂ ⌂



The screenshot shows the TripAdvisor website for Rome Tours. The page title is "Rome Tours". It features a search bar and filters for dates and sorting. Below the search bar, there's a promotional banner for dog-friendly travel. The main content area displays two tours as "BEST SELLER": 1. Colosseum Arena Floor, Roman Forum and Palatine Hill Guided Tour (Rating: 4.5 stars, 2,642 reviews) and 2. Skip-the-Line Vatican, Sistine Chapel & St. Peter's | Small Group (Rating: 4.5 stars, 3,676 reviews). Both tours include images and descriptions of the activities.

# Mindpal

---

Agents platform template-based

## Provide your own instructions & context

Teach your AI agent to understand your business/work context and perform tasks according to your expectations

## Train with your data — any kind of data



## Connect to your favorite tools



## Choose from the best AI models



[mindpal](#)

Features ▾ Tutorials Testimonials Pricing FAQs [✉](#) [YouTube](#) [Facebook](#) [LinkedIn](#)

Get started >

Product of the day  
1st

## Build Your AI Workforce to Automate thousands of tasks

MindPal is the platform to design, deploy, and manage custom AI agents and multi-agent workflows that drive your business forward

[Get started →](#)

[Show me how](#)

mp

# CrewAI

You can set up an enterprise-level infrastructure to develop your army of agents

The screenshot shows the CrewAI website's homepage. At the top, there is a navigation bar with links for Home, Enterprise, Open Source, Ecosystem, Use Cases, Templates, and Blog. To the right of the navigation bar are Log in and Start Enterprise Trial buttons. Below the navigation bar, the main heading reads "The Leading Multi-Agent Platform". A subcopy below the heading states: "Streamline workflows across industries with powerful AI agents. Build and deploy automated workflows using any LLM and cloud platform." There are two call-to-action buttons: "Start Free Trial" and "I Want A Demo". A large red text "100,000,000+" is displayed, followed by the text "Multi-Agent Crews run using CrewAI". To the right of the main content area is a grid of 16 smaller icons, each representing a different AI platform or technology, such as AI, IBM, Box, Salesforce, SAP, and others.

crewai

Home Enterprise Open Source Ecosystem Use Cases Templates Blog

Log in Start Enterprise Trial

## The Leading Multi-Agent Platform

Streamline workflows across industries with powerful AI agents. Build and deploy automated workflows using any LLM and cloud platform.

Start Free Trial I Want A Demo

**100,000,000+**

Multi-Agent Crews run using CrewAI

- AI
- IBM
- Box
- salesforce
- SAP
- infinity
- cloud
- A
- hubspot
- Q

# Lindy

Solutions ▾ Pricing Integrations Log in Get in touch Try for free

## Phone

Manage calls and voicemail intelligently across your entire team.

See all →

**Outbound Phone Call Agent**  
Book consultations with qualified leads instantly.

**Phone Call Assistant**  
An AI assistant for telephoning your contacts.

tomation

## Customer Support

Faster, better, cheaper support with AI.

See all →

Need help using Lindy?



Get in touch Try for free

Solutions ▾ Pricing Integrations Log in Get in touch Try for free

Use cases Integrations

Phone Calls Gmail >

Email Automation Zoom >

Meeting Recording Twilio >

See all templates → See all 234 →

Templates

fill out this form: [docs.google.com](https://docs.google.com)

 Proxy has completed: 1 steps

Steps

Hide all 

● Starting

Could you please provide the specific details for the responses you want to enter into the Google Form? Additionally, if there are any particular instructions or required fields, let me know so I can fill them out correctly. If the form requires login, please confirm if you have access to the necessary credentials.

Create the data to insert



## ASA 2025 - Group Registration

B I U ↲ ✖

Autonomous Software Agents - Unitn 2024/2025

Registration form is intended to be submitted one per group.

Questo modulo raccoglie automaticamente gli indirizzi email di chi risponde [Modifica impostazioni](#)

### Group \*

Assign a short name to your group

Testo risposta breve

### Participant 1

Descrizione (facoltativa)

### Name Surname \*

Participant 1

 **Sign In to Continue**

To fill out this form, you must be signed in.

[Sign In](#)

**Submit**

 **Sign in**

Use your Google Account

Email or phone

Email or phone

**Submit**

 **University of Trento Login**

Username

Username

Password

Password

[Login with SPID](#)

[Login with CIE](#)

**Submit**

 **UNIVERSITY OF TRENTO**

IdP4-Az31  
IT | EN

**Username**

**Password**  
 [Forgot your password?](#)

[@unitn.it](#) [@guest.unitn.it](#)

**Login**

---

Otherwise

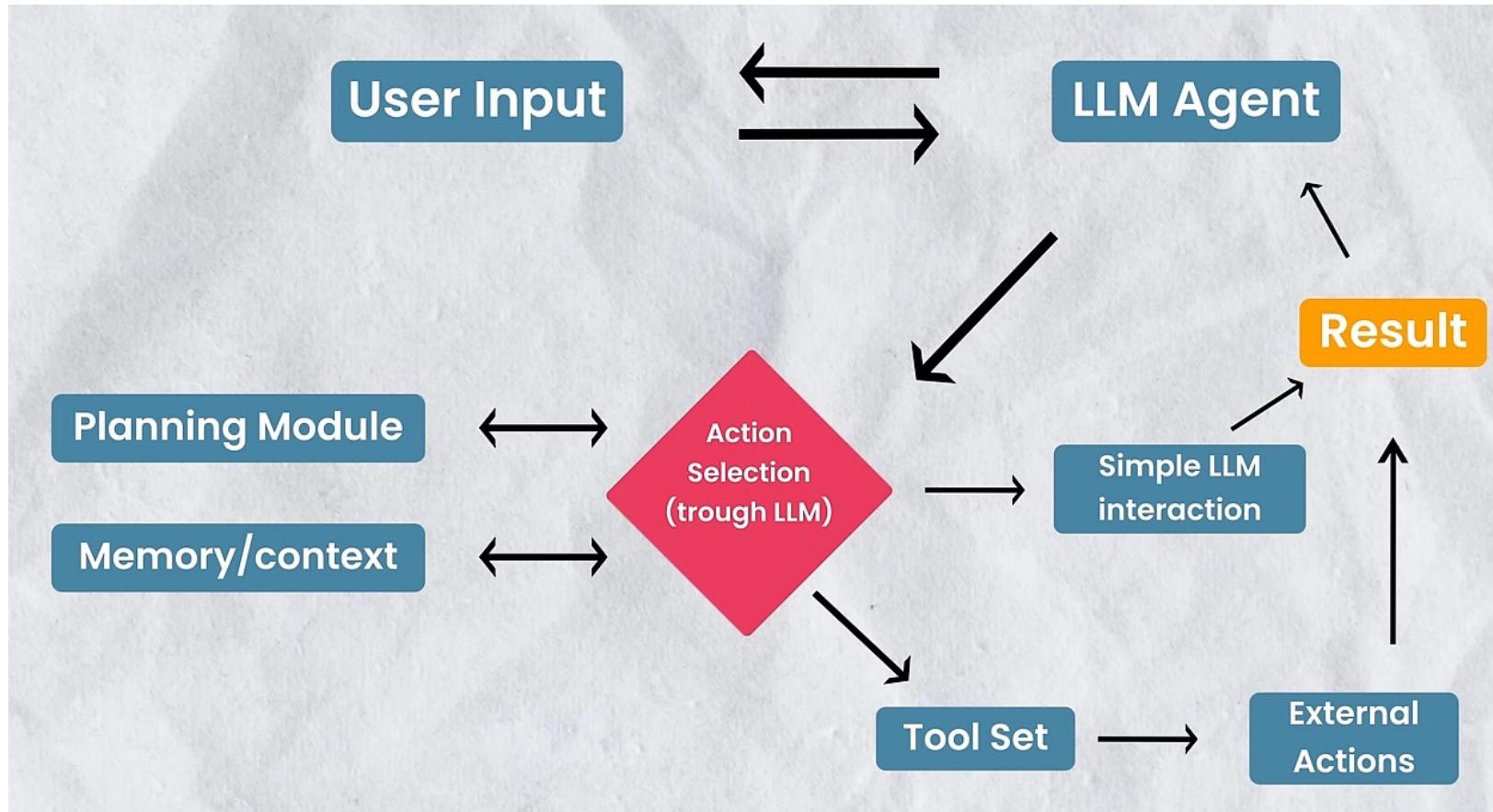
 [Login with SPID](#)

 [Login with CIE](#)

[» Advanced options](#)

[Privacy](#) | [Antiphishing](#) | [Help & Info](#)





# First simple ChatGPT agent

- 
- Create an image
- DALLE (Tool)

ChatGPT

DALL-E

Explore GPTs

Today

New chat

DALL-E

You  
Cute poodle saying happy birthday

DALL-E

DALL-E

DALL-E

HAPPY BIRTHDAY!

HAPPY BIRTHDAY!

Here are two images of a cute poodle celebrating a birthday, just as you described. I hope you find them perfect for your celebration!

# Canva Magic

- Agent integrated in the environment
- The agent understand the user requists and uses Canva tools to develop and answer the user
- Agent limited to all features of the product (canva)



# Replit

Ask to write code for you

- You know what you want but you might not know how to do it
- Replit set the whole development environment
  - Installing libraries
  - Local servers
- Write the code
- You can try the software and you can modify the code

The image shows a white silhouette of a person's head and shoulders facing right, overlaid on a dark-themed user interface. In the top right corner, there is a large, light-colored number '27'. Below it, a code editor window displays Python code for interacting with Google Sheets. A message from 'matt-palmer' is visible. In the bottom half of the image, a terminal window shows a command-line interface with a user named 'amanm3'. Below the terminal, several buttons offer different project templates: 'Demo gallery', 'Beautiful personal blog website' (which is currently selected), 'Bar graph from the data in a .csv', 'Send me a text with the weather', 'Create Repl' (highlighted in blue), 'Create React TypeScript', and 'Create Python'.

```
try:
    except AuthenticationError as e:
        print("Authentication error, please configure the sheet.")
    else:
        # Open the Google Sheet
        sheet = gspread.open(SHEET_NAME)
        worksheet = sheet.worksheet_by_title(WORKSHEET_NAME)

# Skip header
row_num = 2

# Get records
```

Create an app that takes a stock ticker and beautifully visualizes its price history and financials.

amanm3

Demo gallery Beautiful personal blog website Bar graph from the data in a .csv Send me a text with the weather

Create Repl Create React TypeScript Create Python

All Replies Deployments



Devin (the developer) + preview.devin.ai

Expanded Editor

```
Devin's W...
```

```
benchmark.py
```

```
24 PERPLEXITY_HEADERS = {
25     "Authorization": f"Bearer {PERPLEXITY_API_KEY}",
26     "Content-Type": "application/json"
27 }
28
29 # Extended prompt to send to each API to generate more tokens
30 PROMPT = "Please provide a detailed summary of the history of France,
31
32 # Function to measure tokens per second
33 def benchmark_api(endpoint, headers, data):
34     start_time = time.time()
35     response = requests.post(endpoint, headers=headers, json=data)
36     end_time = time.time()
37
38     if response.status_code == 200:
39
40         tokens = len(response.json()['choices'][0]['text'].split())
41         time_taken = end_time - start_time
42         tokens_per_second = tokens / time_taken
43         return tokens_per_second
44
45     else:
46         print(f"Error: {response.status_code}")
47         return None
48
49 # Data payloads for each API
50 replicate_data = {
51     "version": "0xe509c789964a7ea8736978a43525956ef48397be9833abff9fd",
52     "input": {"text": PROMPT, "max_tokens": 512}
53 }
54 together_data = {
55     "model": "mistralai/Mixtral-8x7B-Instruct-v0.1",
56     "messages": [{"role": "user", "content": PROMPT},
57     {"max_tokens": 512
58 }
59
60
61
62 }
```

```
Devin (the developer) + preview.devin.ai
```

Devin's Workspace

Devin

Hey Scott, thanks for the heads up on the model name, and for the updated Perplexity key. I'm currently addressing a couple of errors from the last script run and will incorporate the new key. I'll make sure to use the correct Llama 2 model for Perplexity and get the script working smoothly. I'll reach out with an update as soon as I've made progress. Appreciate your patience!

Updated the perplexity API key in the script with the new key provided.

Scott

ok I found the perplexity model, it was pretty hidden # llama-2-70b-chat

Devin is currently debugging the script to resolve the API response errors and the KeyError text issue.

https://docs.perplexity.ai/reference/post\_chat\_completions



```
default
```

```
<hr><center><script defer s 2de94ce1a686ba8 SCFS3DPRyiWu3Fn n='("rayId":"86 7fb6a25e1d6657 </body> </html>
```

```
Replicate Token Together Tokens Perplexity Token ubuntu@ip-172-31-15-15:~/devin$
```

```
benchmark.py
```

```
home > ubuntu > benchmark.py
1 import os
2 import time
3 import requests
4
5 # API keys
6 TOGETHER_API = os.environ.get("TOGETHER_API")
7 REPLICATE_API = os.environ.get("REPLICATE_API")
8 PERPLEXITY_API = os.environ.get("PERPLEXITY_API")
9
10 # API endpoint URLs
11 API_ENDPOINT_TOGETHER = "https://api.replicate.com/v1/predictions"
12 API_ENDPOINT_REPLICATE = "https://api.replicate.com/v1/predictions"
13 API_ENDPOINT_PERPLEXITY = "https://perplexity.ai/api/v1/completions"
14
15 # Model names
16 MODEL_TOGETHER = "mistralai/Mixtral-8x7B-Instruct-v0.1"
17 MODEL_REPLICATE = "mistralai/Mixtral-8x7B-Instruct-v0.1"
18 MODEL_PERPLEXITY = "llama-2-70b-chat"
19
20 # Prompt
21 PROMPT = "Please provide a detailed summary of the history of France."
22
23 # Headers for API requests
24 PERPLEXITY_HEADERS = {
25     "Authorization": f"Bearer {PERPLEXITY_API_KEY}",
26     "Content-Type": "application/json"
27 }
28
29 # Extended prompt to send to each API to generate more tokens
30 PROMPT = "Please provide a detailed summary of the history of France,
31
32 # Function to measure tokens per second
33 def benchmark_api(endpoint, headers, data):
34     start_time = time.time()
35     response = requests.post(endpoint, headers=headers, json=data)
36     end_time = time.time()
37
38     if response.status_code == 200:
39
40         tokens = len(response.json()['choices'][0]['text'].split())
41         time_taken = end_time - start_time
42         tokens_per_second = tokens / time_taken
43         return tokens_per_second
44
45     else:
46         print(f"Error: {response.status_code}")
47         return None
48
49 # Data payloads for each API
50 replicate_data = {
51     "version": "0xe509c789964a7ea8736978a43525956ef48397be9833abff9fd",
52     "input": {"text": PROMPT, "max_tokens": 512}
53 }
54 together_data = {
55     "model": "mistralai/Mixtral-8x7B-Instruct-v0.1",
56     "messages": [{"role": "user", "content": PROMPT},
57     {"max_tokens": 512
58 }
59
60
61
62 }
```

# Davin

Your software engineer

Devin is waiting to start up...

Hey Devin, I'd like for you to benchmark Llama 2 on three different platforms: Together, Replicate, and Perplexity. Figure out their API formats and write a script to compare them.

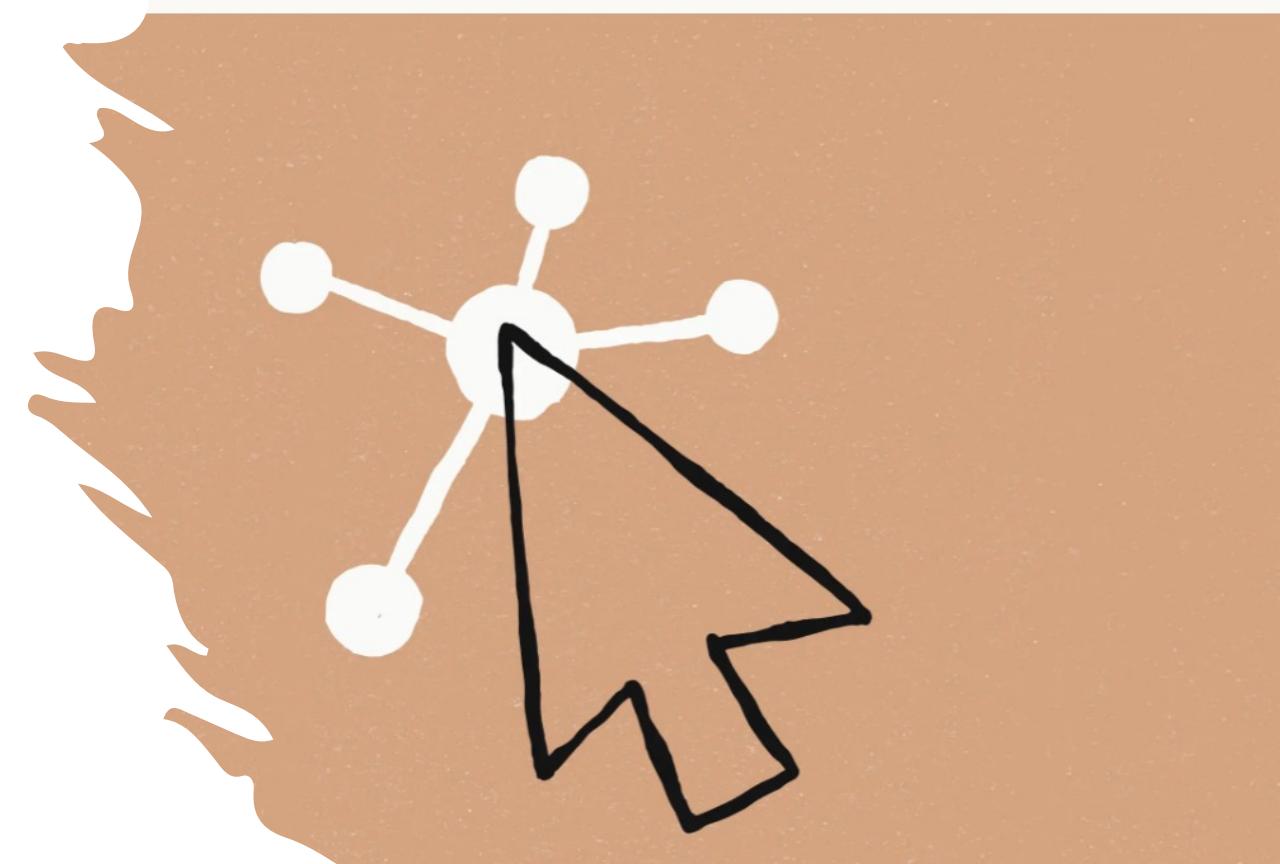
# Claude – Anthropic

- The agent that can do everything we do with a computer.

[Announcements](#) [Product](#)

## a computer use model

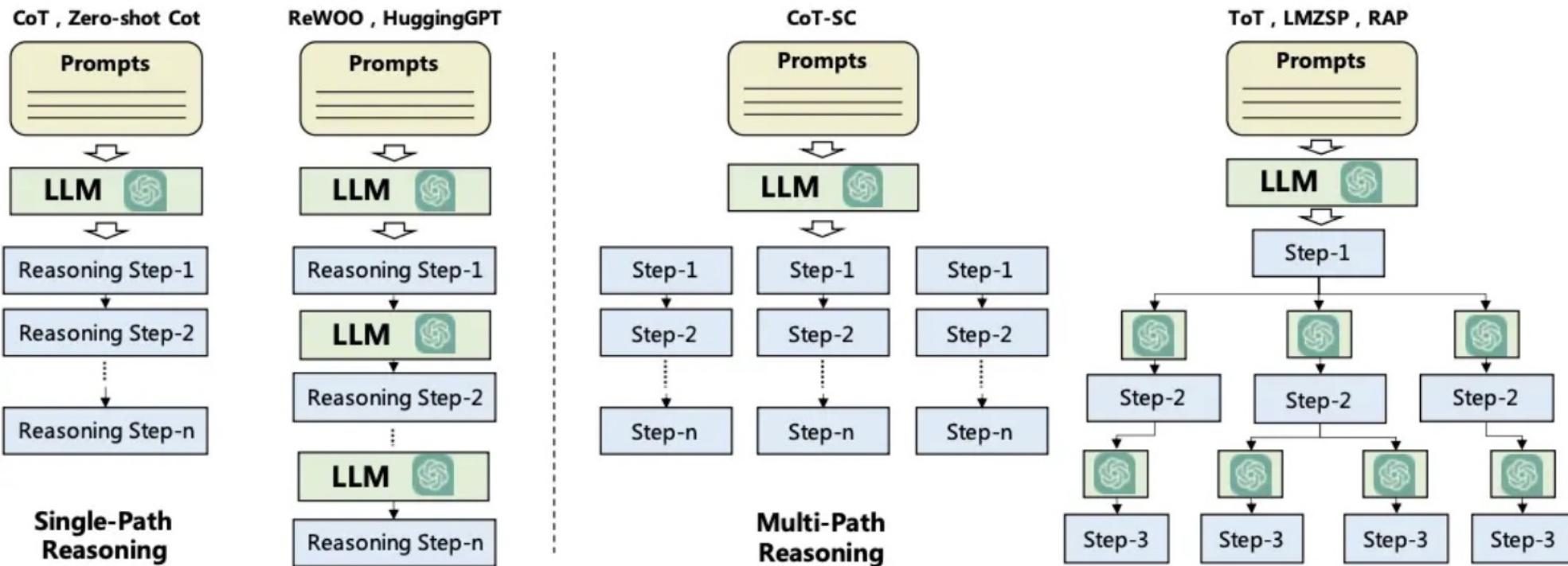
22 Oct 2024 • 7 min read



# Planning Module – without feedback

- Breaks down necessary steps or subtasks to answer user requests
- Helps the agent reason better and find solutions reliably
- Uses an LLM to generate a detailed plan with subtasks
- Techniques for task decomposition:
  - Chain of Thought (Single-path reasoning)
  - Tree of Thoughts (Multi-path reasoning)

# Example comparison from Wang et al., 2023.



# Planning with Feedback

- Traditional planning modules lack feedback, making long-horizon planning difficult
- A **reflection** mechanism allows models to iteratively refine execution plans
  - Iteratively reflect and refine the execution plan based on past actions and observations
- This process corrects mistakes and improves final results
- Essential for complex real-world tasks where trial and error is key.
- Two popular methods:
  - ReAct and Reflexion

# ReAct

- Enables an LLM to solve complex tasks by interleaving three steps (repeated N times):
  - Thought
  - Action
  - Observation
- Receives feedback from the environment as observations
- Other feedback sources include human and model feedback.

## (1d) ReAct (Reason + Act)

**Thought 1:** I need to search Apple Remote and find the program it was originally designed to interact with.

**Act 1:** `Search[Apple Remote]`

**Obs 1:** The Apple Remote is a remote control introduced in October 2005 by Apple ... originally designed to control the Front Row media center program ...

**Thought 2:** Apple Remote was originally designed to control the Front Row media center program. I need to search Front Row next and find what other device can control it.

**Act 2:** `Search[Front Row]`

**Obs 2:** Could not find [Front Row]. Similar: ['Front Row Seat to Earth', 'Front Row Motorsports', 'Front Row (software)', ...]

**Thought 3:** Front Row is not found. I need to search Front Row (software) .

**Act 3:** `Search[Front Row (software)]`

**Obs 3:** Front Row is a discontinued media center software ...

**Thought 4:** Front Row (software) is controlled by an Apple Remote or the keyboard function keys. So the answer is keyboard function keys.

**Act 4:** `Finish[keyboard function keys]`



# Memory

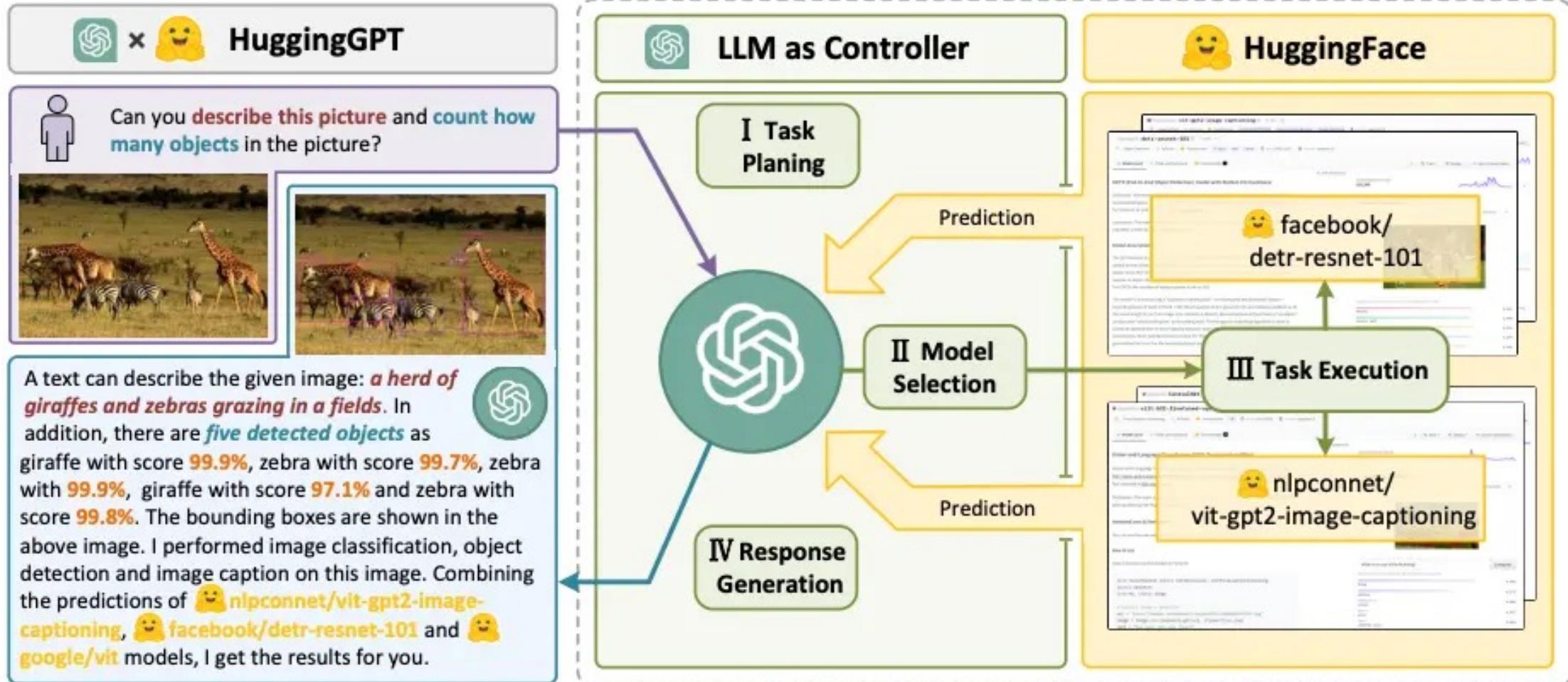
- Stores the agent's internal logs, including:
  - Past thoughts, actions, and observations
  - Interactions between agent and user
  - Planning and memory modules enable to recall past behaviors and plan future actions
- Maintaining relevant context and improving decision-making
  - **Short-term memory**
    - agent's current situation
    - realized via in-context learning - limited by the context window size, making it short and finite
  - **Long-term memory**
    - agent's past behaviours and thoughts for extended periods
    - external vector stores for efficient storage and retrieval
    - Enables scalable access to relevant past information as needed
  - **Hybrid memory**
    - Integrates both short-term memory and long-term memory for long-range reasoning and accumulation of experiences

# Memory Format

- Different memory formats
  - **Representative memory formats**
    - natural language, embeddings, databases, and structured lists
    - can be combined such as in Ghost in the Minecraft that utilizes a key-value structure where the keys are represented by natural language and values are represented by embedding vectors

# Tools

- Enables the agent to interact with external environments
  - Wikipedia Search API, Code Interpreter, Math Engine ...
- Tools could include
  - databases, knowledge bases, and external models.
- The agent uses tools to execute tasks to obtain observations or necessary information to complete subtasks
- Tools are leveraged in different ways by LLMs:
  - MRKL is a framework that combines LLMs with expert modules that are either LLMs or symbolic
  - Toolformer fine-tunes LLMs to use external tool APIs
  - Function Calling - augments LLMs with tool use capability which involves defining a set of tool APIs and providing it to the model as part of a request
  - HuggingGPT - an LLM-powered agent that leverages LLMs as a task planner to connect various existing AI models (based on descriptions) to solve AI tasks.



# Beliefs representation and management

Autonomous Software Agents

A.A. 2024-2025

**Prof. Paolo Giorgini**

**Dr. Marco Robol**

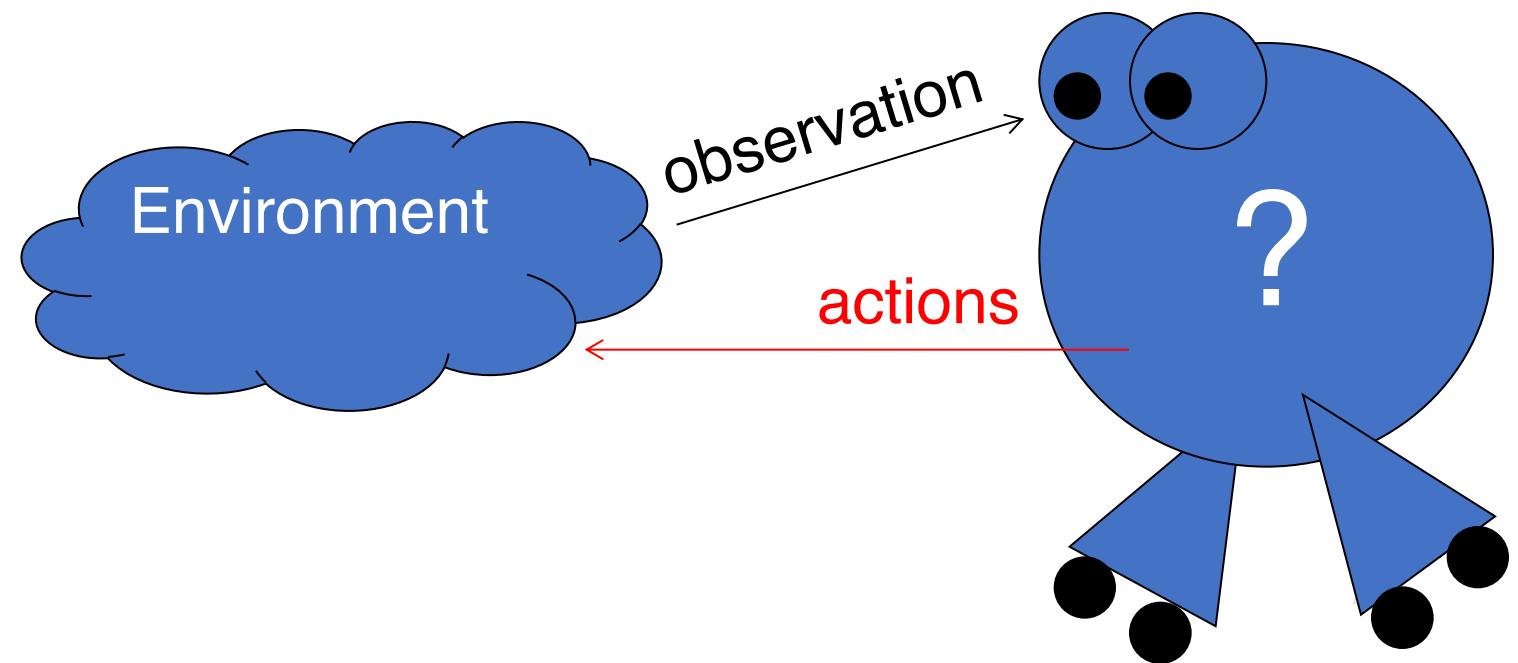


UNIVERSITY OF TRENTO - Italy

Department of Information  
and Communication Technology

# Belief Revision

- An agent revise beliefs in response to new information
  - But which ways of revising beliefs are “OK” and which are not?
- A belief revision theory is meant to provide a general answer, with a sense of “OK” that it specifies



# An example

$\alpha$ : All European swans are white

$\beta$ : The bird caught in the trap is a swan

$\gamma$ : The bird caught in the trap comes from Sweden

$\delta$ : Sweden is part of Europe

If the agent's database is coupled with a program that can compute logical inferences:

$\varepsilon$ : The bird caught in the trap is white

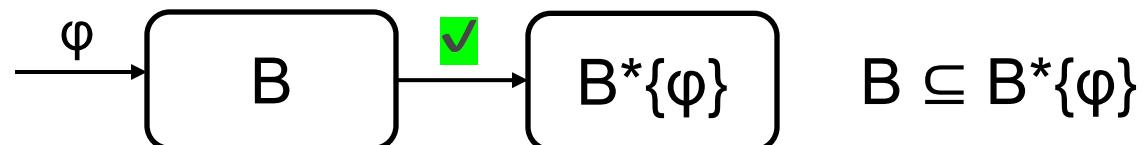
Now: the bird caught in the trap turns out to be black. This means that you want to add the fact  $\neg\varepsilon$ . But then the database becomes *inconsistent*.

# Belief Revision

- To keep the database consistent, you need to *revise* it
  - Some of the beliefs in the original database must be retracted
  - You don't want to give up all of the beliefs since this would be an unnecessary loss of valuable information
  - choose between retracting  $\alpha$ ,  $\beta$ ,  $\gamma$  or  $\delta$
- Problem: beliefs in a database have logical consequences, so which of the consequences to retain and which to retract
- If retract  $\alpha$ ,  $\alpha$  has as logical consequences, among others, the following two:
- $\alpha'$ : *All European swans except the one caught in the trap are white*
- $\alpha''$ : *All European swans except some of the Swedish are white*
- Do you want to keep any of these sentences in the revised database?

# Belief Revision Theory

- A belief revision theory is, roughly, a theory saying which ways of belief revision are OK and which are not
- **Preservation.** If the information that agent A receives at time t is *compatible* with the set of the beliefs that A has right before t, then, right after t, agent A retains all of her beliefs in response to that information
- If  $\phi$  is compatible with B, then B is a subset of  $B * \phi$ , where:
  - B is the set of one's beliefs right before the receipt of new information,
  - $\phi$  is the new information one receives,
  - $B * \phi$  is the set of one's new beliefs in response to new information  $\phi$



# Belief Revision Theory

- **Preservation Thesis (the “Perfect Rationality” Version).** One is perfectly rational only if one has never violated, and would never violate, Preservation.
  - *Example (Three Composers).* The agent initially believes the following about the three composers Verdi, Bizet, and Satie.
    - (a) Verdi is Italian
    - (b) Bizet is French
    - (c) Satie is French
- Then the agent receives this information
- (e) Verdi and Bizet are compatriots

# Example (Three Composers)

- The agent drops her beliefs (a) and (b), and retains the belief in (c) that Satie is French (after all, information (e) has nothing to do with Satie).
- The agent comes to believe the new information (e) that Verdi and Bizet are compatriots, while suspecting that Verdi and Bizet might both be Italian, and that they might both be French.
- At this stage, the agent does not rule out the possibility that Verdi is French (and, hence, a compatriot of Satie).
- So, what she believes at this stage is compatible with the following proposition.
- (f) Verdi and Satie are compatriots

# Example (Three Composers)

- But then she receives a second piece of information, which turns out to be (f). Considering that she started with initial beliefs (a), (b), and (c) and received information (e) and (f), which jointly say that the three composers are compatriots, now she drops her belief (c).
- Information (f) is compatible with what she believes right before receiving this information, and she drops her belief in (c) nonetheless. So, this agent's second revision of beliefs violates Preservation. But there seems nothing in the specification of the scenario that prevents the agent from being perfectly rational. So, this seems to be a counterexample to the Preservation Thesis.

- (a) Verdi is Italian
- (b) Bizet is French
- (c) Satie is French

$$B = \{a, b, c\}$$

*1 Belief Revision (+e) -----*

- (e) Verdi and Bizet are compatriots
- (c) Satie is French

*- (e) not compatible with B*

$$B^* \{e\} = \{c, e\}$$

*2 Belief Revision (+ f) -----*

- (f) Verdi and Satie are compatriots
- (e) Verdi and Bizet are compatriots

*- (f) compatible with  $B^* \{e\}$*

$$B^* \{e\} * \{f\} = \{e, f\}$$

*$\{c, e\} \not\subseteq \{e, f\}$  violates Preservation, but still the agent seems to be perfectly rational*

# Methodological questions

- How are the beliefs in the database ***represented***?
  - Most databases work with elements like *facts* and *rules* as primitive forms of representing information. The code used to represent the beliefs may be more or less closely related to standard logical formalism. A mechanism for belief revision is sensitive to the formalism chosen to represent the beliefs
- What is the relation between the elements explicitly represented in the database and the beliefs that may be ***derived*** from these elements?
  - It depends on the *application area*. The nature of the relation between explicit and implicit beliefs is of crucial importance for how the belief revision process is attacked
- How are the choices concerning what to ***retract*** made?
  - Eg: An agent can decide adopting the minimal change principle or on the base of the importance of the beliefs. In CS, Integrity constraints is a common way of handling the problem

# Three kinds of Belief Changes

In AGM theory - Alchourrón, Gärdenfors, and Makinson (1985)

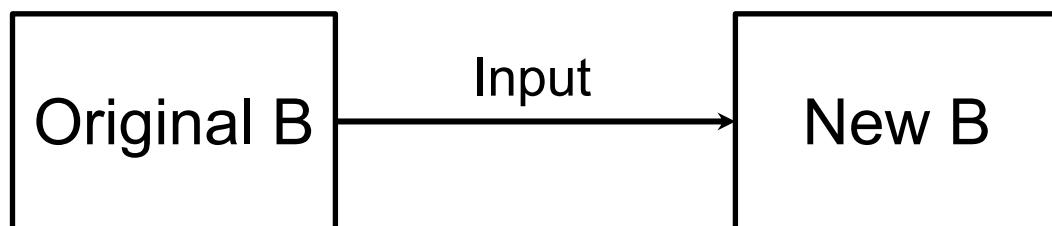
- ***Expansion***: A new belief  $\alpha$  is added to  $B$  together with the logical consequences of the addition (regardless of whether the larger set so formed is consistent). The new belief set is denoted by  $B+\alpha$
- ***Revision***: A new belief that is inconsistent with  $B$  is added, but, in order to maintain consistency of  $B$ , some of the old belief are deleted (  $B\dot{+}\alpha$  )
- ***Contraction***: Some belief in  $B$  is retracted without adding any new belief. In order for the resulting  $B$  to be closed under logical consequences some other sentences from  $B$  must be given up (  $B\dot{-}\alpha$  )

Three possible attitudes for an agent:

Acceptance:  $\alpha \in B$  ( $\alpha$  is true)

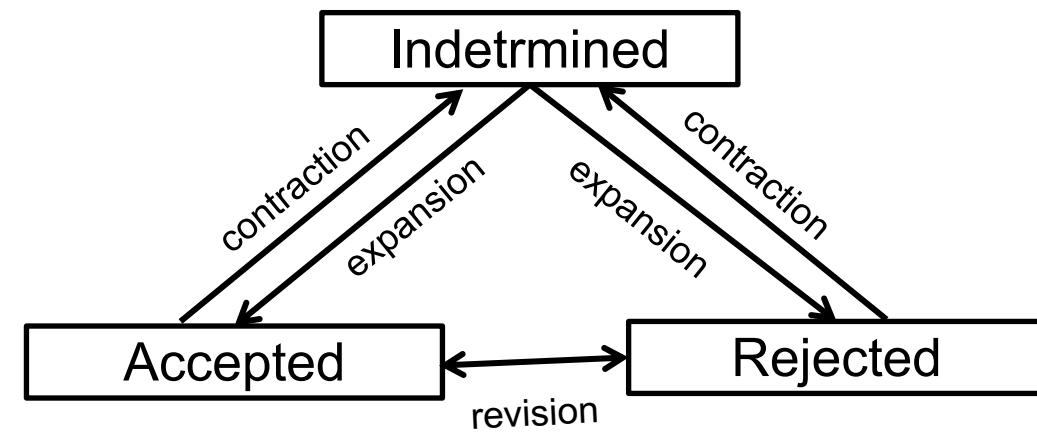
Rejection:  $\neg\alpha \in B$  ( $\alpha$  is false, i.e.  $\neg\alpha$  is true)

Indetermination:  $\alpha \notin B$  nor  $\neg\alpha \in B$



Original  
Epistemic  
state

New  
Epistemic  
state



# Belief Revision functions

- $B+\alpha$  can simply be defined as the logical closure of  $B$  together with  $\alpha$  :  
$$B+\alpha = \{\delta : B \cup \{\alpha\} \vdash \delta\}$$
- Not so easy for revisions (and contractions)
  - There is no purely logical reason for making one choice rather than the other among the beliefs to be retracted, but we have to rely on additional information about these beliefs.
  - There are several ways of specifying the revision  $B+\alpha$ .
  - General properties of a revision function has to be elaborated (i.e., algorithms can be found for computing **revision functions – contraction functions**)

# Minimal change principle + others

- When we change our beliefs, we want to retain as much as possible from our old beliefs – we want to make a **minimal change**
  - Information is in general not gratuitous, and unnecessary losses of information are therefore to be avoided
  - This heuristic criterion may be called the criterion of **informational economy**
- + other criteria: **consistency**, **preference**, **closure**, **primacy of the new information** - not all of them are desirable at all times
- AGM postulates for contraction and revision
- Levi identity:  $B \dot{+} \alpha = (B \dot{-} \neg \alpha) + \alpha$ 
  - (Revision = contraction + expansion)
- Harper identity:  $B \dot{-} \alpha = (B \dot{+} \neg \alpha) \cap B$

# Updates vs. Revisions

- Updating vs Revision - Katsuno and Mendelzon (1992)
  - (1) new belief about a static world – **revision**
    - The information you receive corrects your knowledge about a world that has not changed
    - You need to modify your beliefs to align them with a reality that was already the case, but which you previously misunderstood
  - (2) new belief about changes brought about by some agent – **updating**
    - Your beliefs were correct, but now the world has changed
    - The information you receive describes a change in the world
    - You need to modify your beliefs to reflect a new state of the world

# Example Updating vs Revision 1/2

- In a room there is a table, a book and a magazine, and that either the book is on the table ( $\beta$ ), or ( $\mu$ ) the magazine is on the table, but not both, i.e.,  $(\beta \wedge \neg\mu) \vee (\mu \wedge \neg\beta)$
- A robot is then ordered to put the book on the table, and as a consequence, we learn that  $\beta$ . If we change our beliefs by revision, we should end up in a belief state that contains  $\beta \wedge \neg\mu$  since  $\beta$  is consistent with B
- But, why should we conclude that the magazine is not on the table?
  - Only if you consider the **world to be static (Revision)**, then you are forced to discard  $\mu$ , in order not to contradict your initial knowledge
    - → Your initial knowledge stated that both items cannot be on the table at the same time, so if you now know that  $\beta$  is true, you are compelled to reject  $\mu$

# Example Updating vs Revision 2/2

- Let's imagine that the **world has changed – updating**: the robot placed the book on the table
  - Previously, the magazine might have been on the table
  - Now, after the robot's action, the book is on the table
- In this scenario:
  - There is no need to discard  $\mu$  from the previous beliefs
  - The world has changed, so the initial beliefs no longer have to fully apply
  - Result: you can simply believe that  $\beta$  is true now, without having to infer  $\neg\mu$

Aspect	Revision	Update
Assumption	The world is <b>static</b>	The world has <b>changed</b>
Belief adjustment	You must <b>correct</b> your mistaken beliefs	You must <b>reflect</b> the new state of the world
New information	Reveals what was <b>already true</b>	Reveals what has <b>just changed</b>
Impact on beliefs	Must preserve consistency with prior knowledge	Prior knowledge may no longer apply
Example result	$\beta \wedge \neg\mu$	$\beta$ (no need to reject $\mu$ )

# Belief Revision

Let's make it concrete

# From sensors' data to Belief

- An agent **acquires data** from its sensors
  - E.g., temperature from the thermostat or its position from GPS

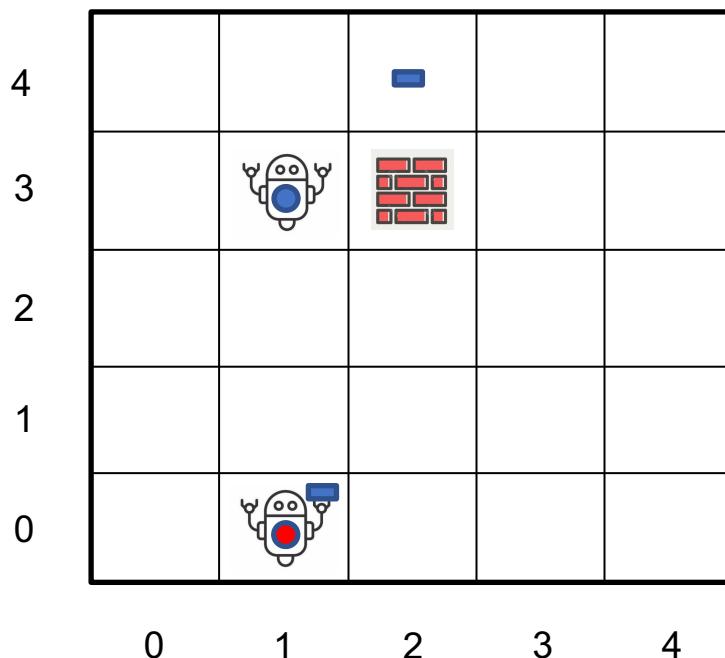
---

Time	Temperature	X	Y
09:00:00	8	3	3
10:00:00	9	3	4
11:00:00	11	3	5
12:00:00	13	4	6

---

- Data can be stored as they are acquired
  - Acquisition time tells us actual values and it draws data evolution

# Data completeness and correctness

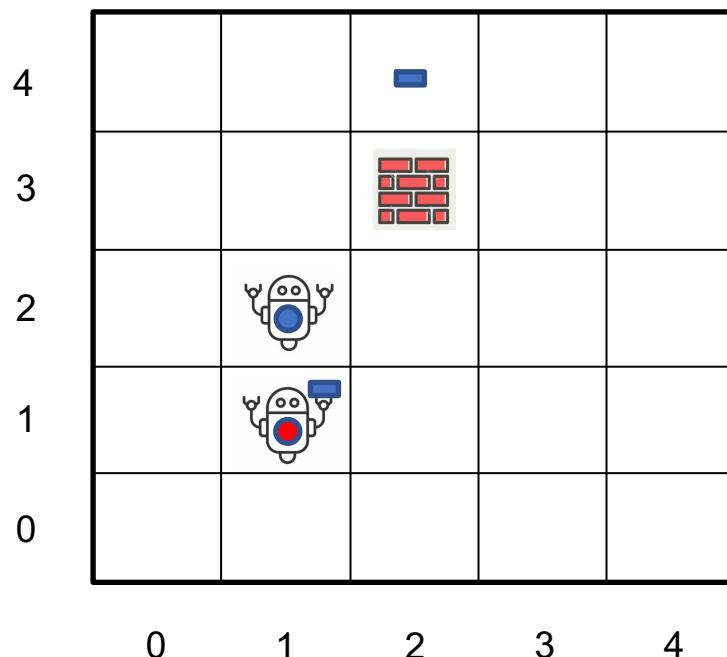


Agent “Ag\_1” (or “Ag\_2”)

Positions				Carry		
Time	Object	X	Y	Time	Agent	Pack
1	Ag_1	1	0	1	Ag_1	Pack_2
1	Ag_2	1	3			
1	Obst_1	2	3			
1	Pack_1	2	4			

$B = \{ \text{In}(Ag\_1, 1, 0), \text{In}(Ag\_2, 1, 3), \text{In}(Obst\_1, 2, 3),$   
 $\text{In}(Pack\_1, 2, 4), \text{carry}(Ag\_1, Pack\_2) \}$

# Data updating



Positions

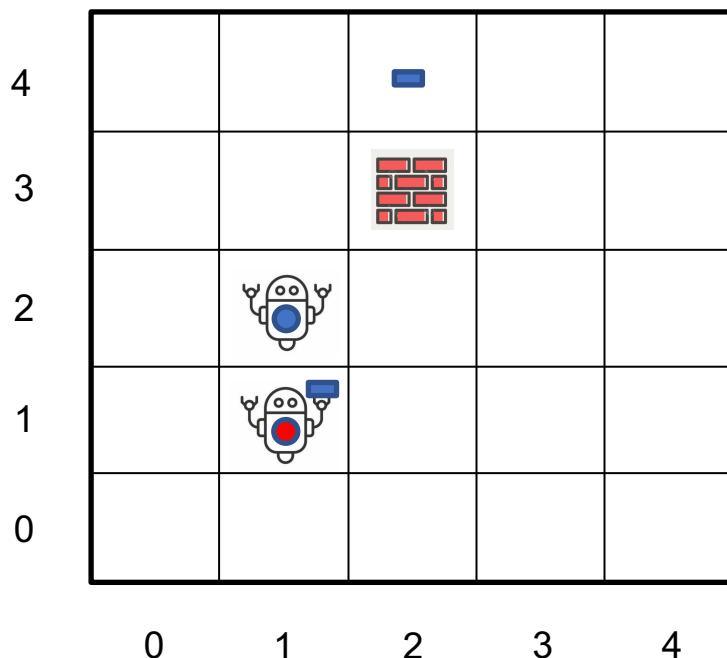
Time	Object	X	Y
1	Ag_1	1	0
1	Ag_2	1	3
1	Obst_1	2	3
1	Pack_1	2	4
2	Ag_1	1	1
2	Ag_2	1	2
2	Obst_1	2	3
2	Pack_1	2	4

Carry

Time	Agent	Pack
1	Ag_1	Pack_2
2	Ag_1	Pack_2

$B = \{ \text{In}(Ag_1, 1, 1), \text{In}(Ag_2, 1, 2), \text{In}(Obst_1, 2, 3),$   
 $\text{In}(Pack_1, 2, 4), \text{carry}(Ag_1, Pack_2) \}$

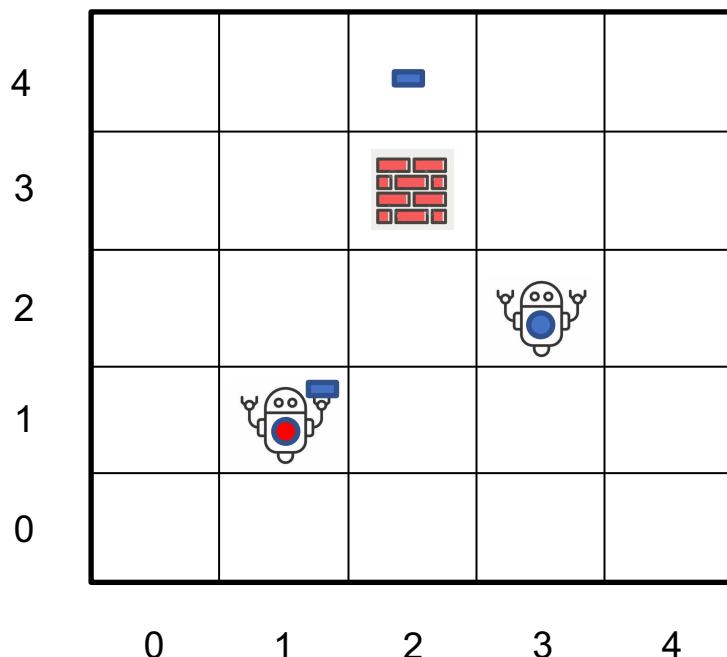
# Derived Beliefs



Positions					Carry		
Time	Object	X	Y	Move	Time	Agent	Pack
1	Ag_1	1	0	No	1	Ag_1	Pack_2
1	Ag_2	1	3	No	2	Ag_1	Pack_2
1	Obst_1	2	3	No			
1	Pack_1	2	4	No			
2	Ag_1	1	1	UP			
2	Ag_2	1	2	DOWN			
2	Obst_1	2	3	No			
2	Pack_1	2	4	No			

$B = \{ \text{In}(Ag_1, 1, 1), \text{In}(Ag_2, 1, 2), \text{In}(Obst_1, 2, 3),$   
 $\text{In}(Pack_1, 2, 4), \text{carry}(Ag_1, Pack_2),$   
 $\text{move}(Ag_1, UP), \text{move}(Ag_2, DOWN) \}$

# Managing Inconsistencies



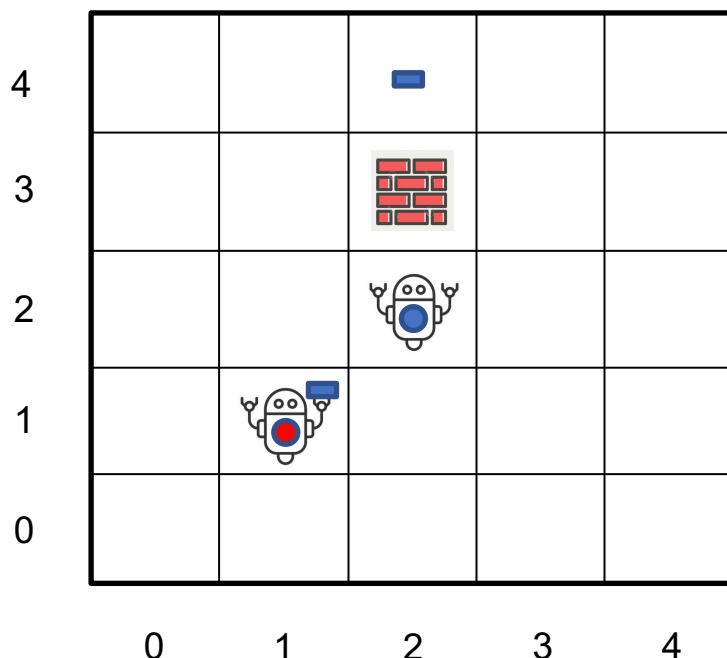
Positions					Carry		
Time	Object	X	Y	Move	Time	Agent	Pack
1	Ag_1	1	0	No	1	Ag_1	Pack_2
1	Ag_2	1	3	No	2	Ag_1	Pack_2
1	Obst_1	2	3	No			
1	Pack_1	2	4	No			
2	Ag_1	1	1	UP			
2	Ag_2	1	2	DOWN			
2	Obst_1	2	3	No			
2	Pack_1	2	4	No			
3	Ag_1	1	1	No			
3	Ag_2	3	2	RIGHT			
3	Obst_1	2	3	No			
3	Pack_1	2	4	No			

Robot cannot move of two tiles in one step

# Inconsistencies

- Inconsistencies can arise for **several reasons**
  - Sensors can send wrong data
  - Predefined rules are not anymore valid (evolution of the environment)
    - Now the robot can move of two tiles
  - Data are provided by other agents
    - They might lie or they could have wrong beliefs
- How much is it **critical** to solve the inconsistency?
  - Can we wait a little bit?
- How to solve them ?
  - Many different ways can be applied
    - `t=2 :In(Ag_2,1,2), move(Ag_2,DOWN)`
    - `t=3 :In(Ag_2,3,2), move(Ag_2,RIGHT)` ----> `t=3 :In(Ag_2,2,2), move(Ag_2,RIGHT)`
- Policies/strategies to solve and manage inconsistencies should be part of the design

# Managing Inconsistencies



Positions

Carry

Time	Object	X	Y	Move	Time	Agent	Pack
1	Ag_1	1	0	No	1	Ag_1	Pack_2
1	Ag_2	1	3	No	2	Ag_1	Pack_2
1	Obst_1	2	3	No			
1	Pack_1	2	4	No			
2	Ag_1	1	1	UP			
2	Ag_2	1	2	DOWN			
2	Obst_1	2	3	No			
2	Pack_1	2	4	No			
3	Ag_1	1	1	No			
3	Ag_2	2	2	RIGHT			
3	Obst_1	2	3	No			
3	Pack_1	2	4	No			

Solving the inconsistency  
(most likely position for Ag\_2?)

# Managing inconsistencies

t=1: In(Ag\_2,1,3)

t=2: In(Ag\_2,1,2), move(Ag\_2,DOWN)

t=3: In(Ag\_2,3,2), move(Ag\_2,RIGHT)

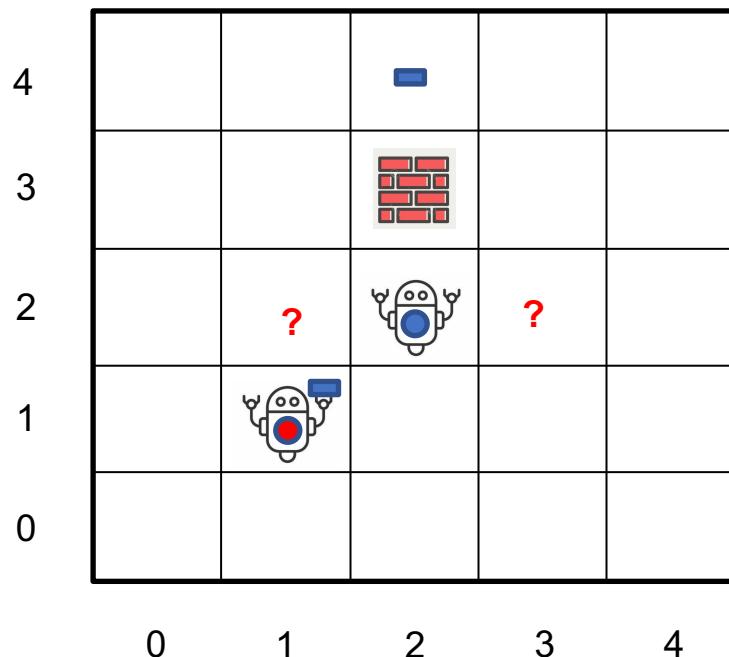
Possible consistent sets:

t=1: In(Ag\_2,1,3)

t=2: In(Ag\_2,1,2), move(Ag\_2,DOWN)

t=3: In(Ag\_2,3,2), move(Ag\_2,RIGHT)

# Another example



t=1: `In(Ag_2, 2, 2)`

t=2: `In(Ag_2, 1, 2), move(Ag_2, LEFT)`

t=3: `In(Ag_2, 3, 2), move(Ag_2, RIGHT)`

Possible consistent sets:

$S_1$

t=1: `In(Ag_2, 2, 2)`

t=2: `In(Ag_2, 1, 2), move(Ag_2, LEFT)`

$S_2$

t=1: `In(Ag_2, 2, 2)`

t=3: `In(Ag_2, 3, 2), move(Ag_2, RIGHT)`

# More on the example

$S_1$

$t=1: \text{In}(\text{Ag\_2}, 2, 2)$   
 $t=2: \text{In}(\text{Ag\_2}, 1, 2), \text{move}(\text{Ag\_2}, \text{LEFT})$

$S_2$

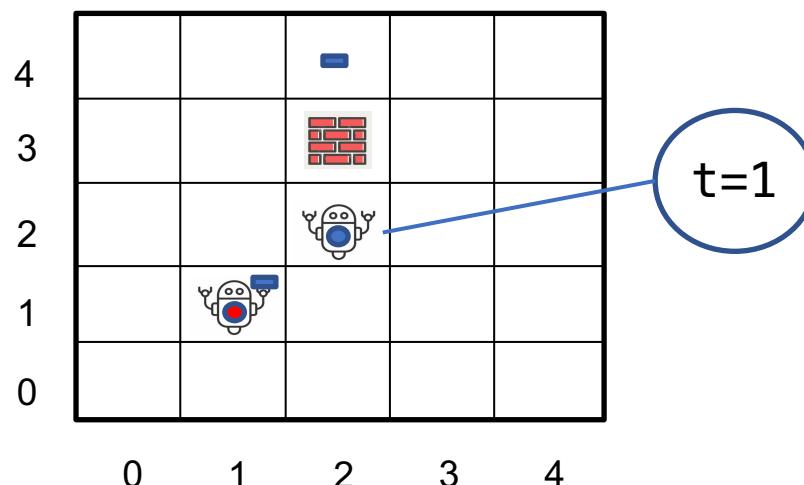
$t=1: \text{In}(\text{Ag\_2}, 2, 2)$   
 $t=3: \text{In}(\text{Ag\_2}, 3, 2), \text{move}(\text{Ag\_2}, \text{RIGHT})$

what about  $t=3$  ?

$t=3: \text{In}(\text{Ag\_2}, 1, 2) \vee \text{In}(\text{Ag\_2}, 0, 2) \vee$   
 $\text{In}(\text{Ag\_2}, 1, 3) \vee \text{In}(\text{Ag\_2}, 2, 2)$

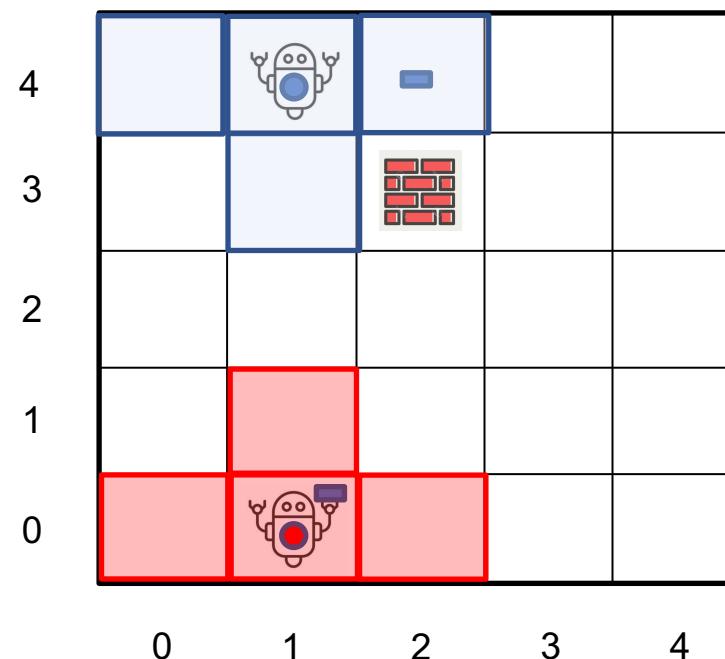
what about  $t=2$  ?

$t=2: \text{In}(\text{Ag\_2}, 2, 2) \vee \text{In}(\text{Ag\_2}, 3, 2)$



After we had choose between  $S_1$  and  $S_2$ , should we update beliefs for  $t=3$  and  $t=2$ , respectively?

# Partial view of the environment



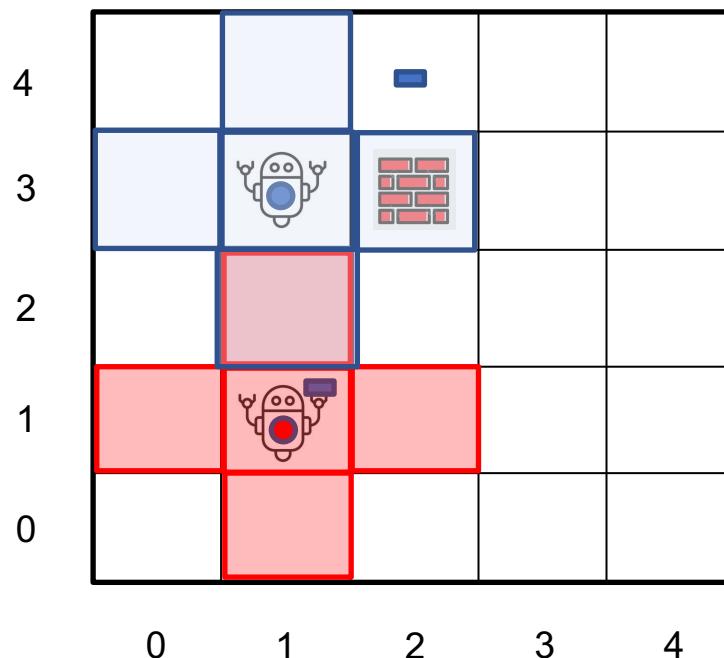
Agent 2

Time	Object	X	Y
1	Ag_2	1	4
1	Pack_1	2	4

Agent 1

Time	Object	X	Y
1	Ag_1	1	0

# Partial view of the environment



**Agent 2**

Time	Object	X	Y
1	Ag_2	1	4
1	Pack_1	2	4
2	Ag_2	1	3
2	Obst_1	2	3

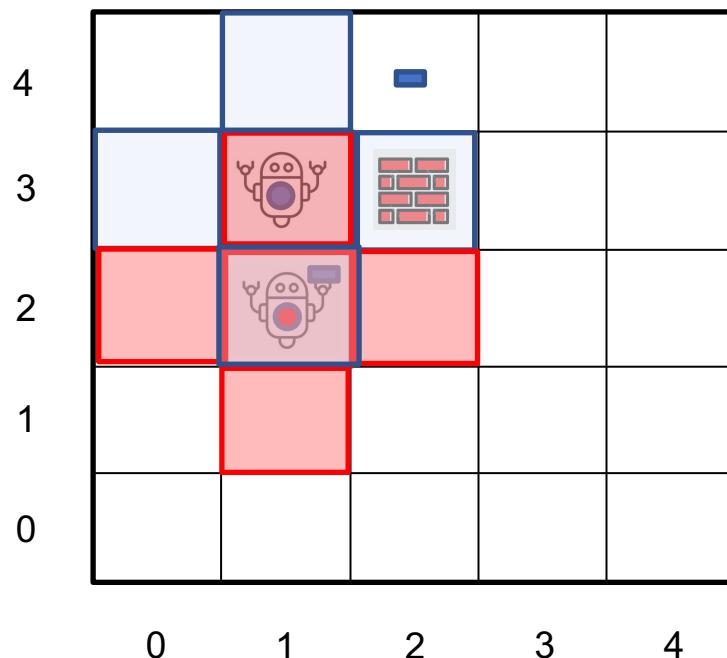
**Agent 1**

Time	Object	X	Y
1	Ag_1	1	0
2	Ag_1	1	1

At time t=2, is Pack\_1 in (2,4)?

- YES ? NO? MAYBE? LIKELY?

# Partial view of the environment



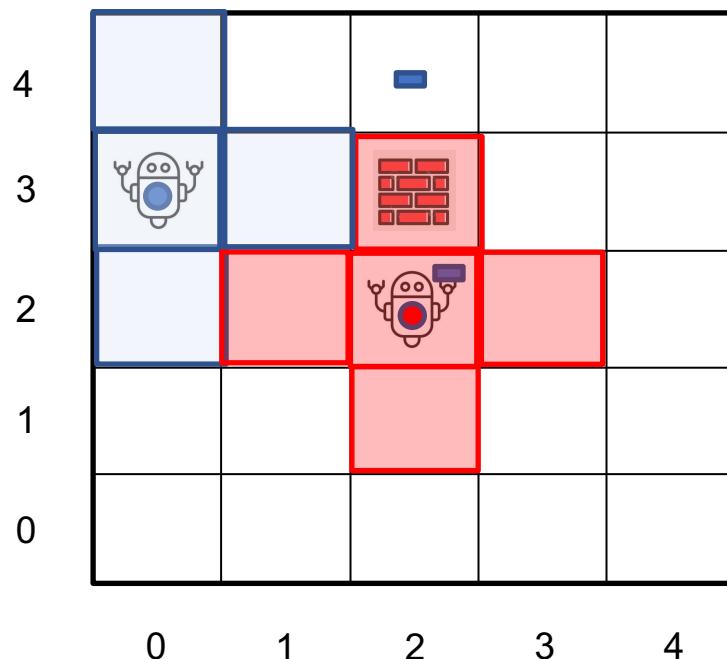
Agent 2

Time	Object	X	Y
1	Ag_2	1	4
1	Pack_1	2	4
2	Obst_1	2	3
2	Ag_2	1	3
3	Obst_1	2	3
3	Ag_2	1	3
3	Ag_1	1	2

Agent 1

Time	Object	X	Y
1	Ag_1	1	0
2	Ag_1	1	1
3	Ag_1	1	2
3	Ag_2	1	3

# Partial view of the environment



Agent 2

Time	Object	X	Y
1	Ag_2	1	4
1	Pack_1	2	4
2	Obst_1	2	3
1	Ag_2	1	3
3	Obst_1	2	3
3	Ag_2	1	3
3	Ag_1	1	2
4	Ag_2	0	3

Agent 1

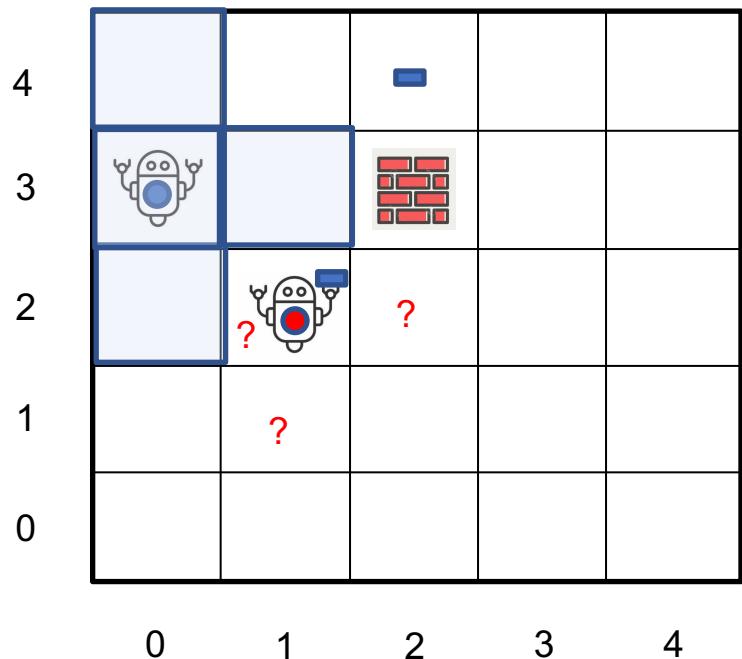
Time	Object	X	Y
1	Ag_1	1	0
2	Ag_1	1	1
3	Ag_1	1	2
3	Ag_2	1	3
4	Ag_1	2	2
4	Obst_1	2	3

They are not anymore there

# As in the previous case

Agent 2				
Time	Object	X	Y	
1	Ag_2	1	4	
1	Pack_1	2	4	
2	Obst_1	2	3	
2	Ag_2	1	3	
3	Obst_1	2	3	
3	Ag_2	1	3	
3	Ag_1	1	2	→
4	Ag_2	0	3	

t=4:  $\text{In}(\text{Ag}_1, 1, 2) \vee \text{In}(\text{Ag}_1, 1, 1) \vee \text{In}(\text{Ag}_1, 2, 2)$



# Beliefs models

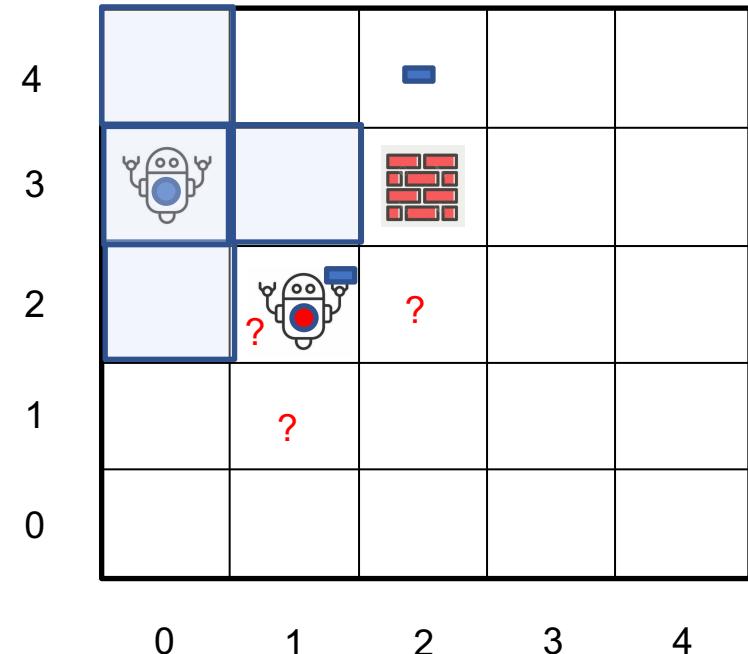
- Several beliefs models can be used
  - **No memory**: only beliefs based on current data
  - **With memory**: beliefs based on current data and keeping true not updated beliefs
  - **With uncertainty**
    - “the probability pack\_1 that I saw long time ago is still in position (x,y) is very low”
    - “the probability obst\_1 is in position (x,y) is 1” (obst\_1 is a wall a nobody can move it)
    - “I saw Ag\_1 going in the direction of Pack\_1, the probability Pack\_1 is in position (x,y) is very low”
    - “Ag\_1 was in position (x,y) and it was moving, the probability it is still there is very low”

# In our example

Agent 2

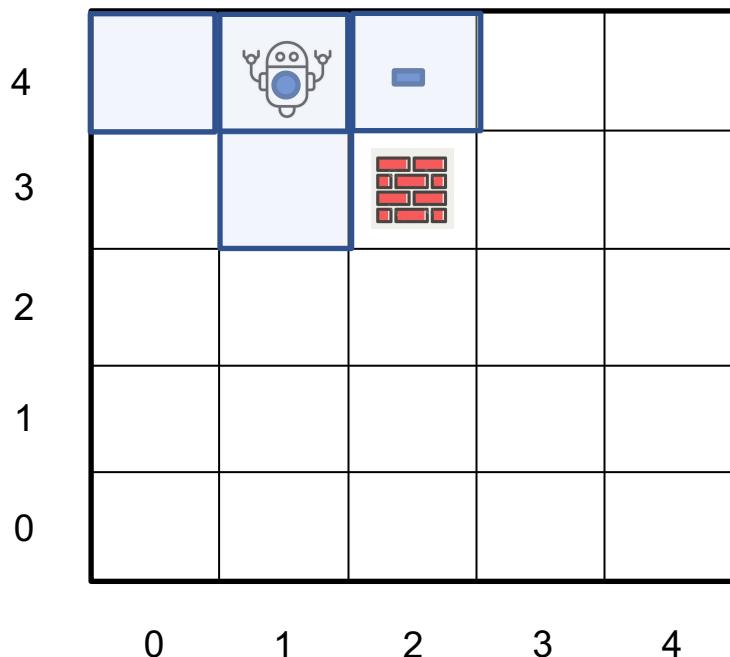
Time	Object	X	Y
1	Ag_2	1	4
1	Pack_1	2	4
2	Obst_1	2	3
2	Ag_2	1	3
3	Obst_1	2	3
3	Ag_2	1	3
3	Ag_1	1	2
4	Ag_2	0	3

$\rightarrow t=4: \text{In}(\text{Ag}_1, 1, 2) \vee \text{In}(\text{Ag}_1, 1, 1) \vee \text{In}(\text{Ag}_1, 2, 2)$



Time	Object	X	Y	Probability
4	Ag_2	0	3	1
4	Ag_1	1	2	0.33
4	Ag_1	1	1	0.33
4	Ag_1	2	2	0.33

# Partial view of the environment



Agent 2

Time	Object	X	Y
1	Ag_2	1	4
1	Pack_1	2	4

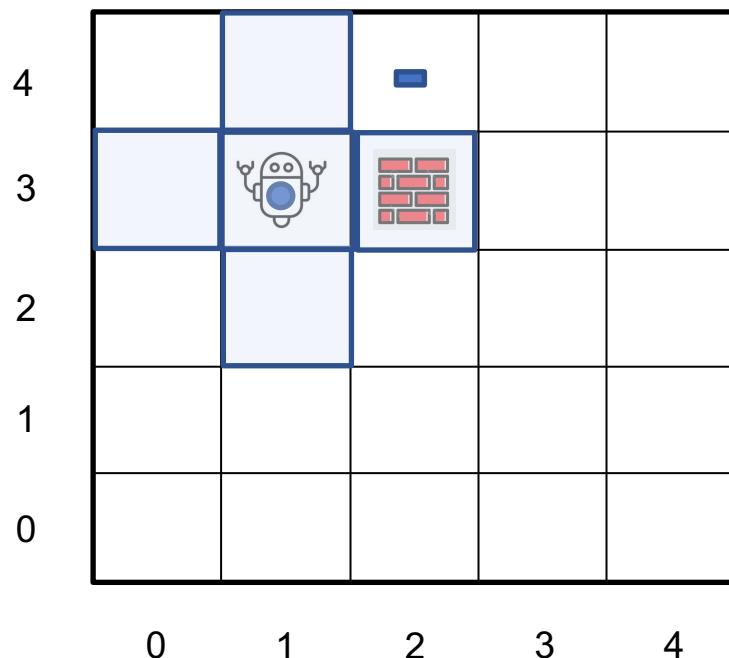
$$P(Pack\_1@2,4) = 0.9$$

Pack\_1 is at (2,4) with confidence  $P(Pack\_1@2,4) = 0.9$

- Why 0.9?
- Agent 2 saw it ( $t=1$ )
- No info suggests it was picked up yet

it's **very likely** it is there

# Partial view of the environment



Agent 2

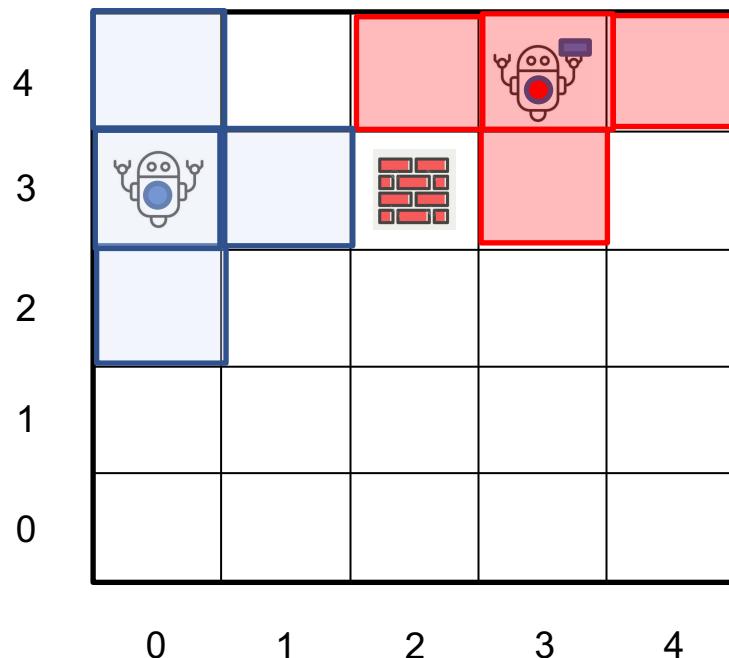
Time	Object	X	Y	
1	Ag_2	1	4	
1	Pack_1	2	4	$P(Pack\_1@2,4) = 0.9$
2	Ag_2	1	3	
2	Obst_1	2	3	$P(Pack\_1@2,3   \text{not seen at } t=2) = 0.5$

Pack\_1 is **not in Agent\_2 view range** and it knows that **other agents may have picked it up**

$$P(P1@2,4 | \text{not seen at } t=2) = 0.5$$

- Partial observability
- Time elapsed since last observation
- Presence of other agents

# Partial view of the environment



Agent 2				
Time	Object	X	Y	
1	Ag_2	1	4	
1	Pack_1	2	4	
2	Obst_1	2	3	
2	Ag_2	1	3	
3	Obst_1	2	3	

Agent 1 enters view of (2,3)

Agent broadcasts:

$\neg \text{In}(P1, 2, 3)$  (Package not found at 2,3)

**Agent A2 revises its belief:**

- Previous:  $P(P1@2, 4) = 0.5$
- New info: A2 did not see Pack\_1
- A2 trusts A1's sensor 80% of the time
- Use **Bayesian-style belief revision**

# Bayesian-style belief revision

$$P(H|E) = \frac{P(E|H) \cdot P(H)}{P(E)}$$

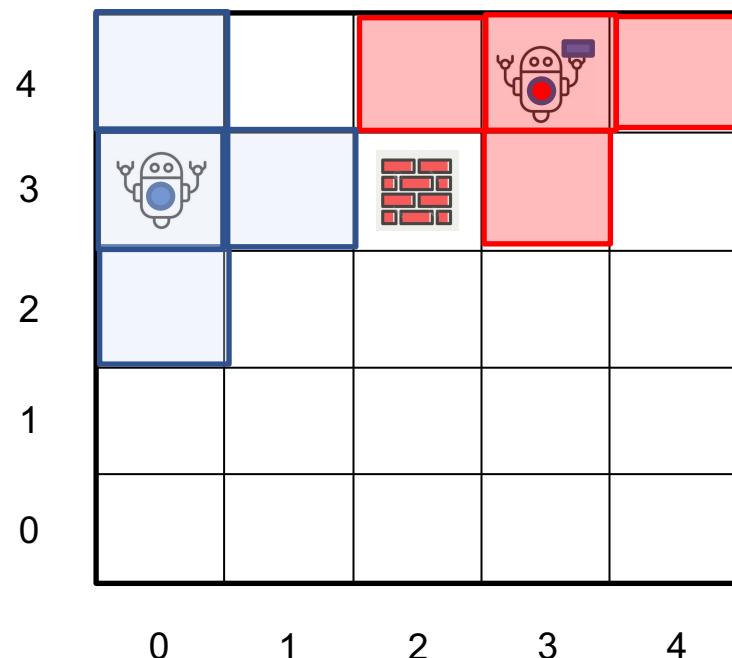
$$P(P1@2,4 | \neg Seen) = P(\neg Seen / P1@2,4) \times P(P1@2,4) / P(\neg Seen)$$

Assume:

- If P1 is at 2,4 → A1 has 80% chance of seeing it →  $P(\neg Seen | P1@2,4) = 0.2$
- If P1 is not at 2,4 →  $P(\neg Seen | \neg P1@2,4) = 1$

$$P(P1@2,4 | \neg Seen) = (0.5 \times 0.2) / [(0.5 \times 0.2) + (0.5 \times 1)] = 0.167$$

# Partial view of the environment



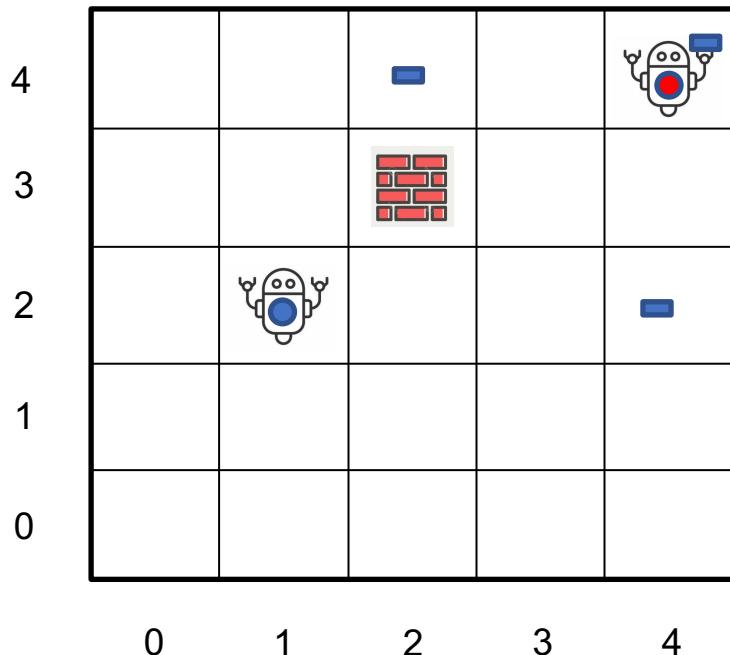
Agent 2

Time	Object	X	Y	P
1	Ag_2	1	4	
1	Pack_2	2	4	0.9
2	Obst_1	2	3	
2	Ag_2	1	3	
2	Pack_2	2	4	0.5
3	Obst_1	2	3	1
3	Ag_2	1	3	
2	Pack_2	2	4	0.167

Agent 2 knows that walls cannot be moved away

Since  $P(P1@2,4) < \text{threshold}$  (e.g., 0.3), A2 does not generate the option to pick P1 up and searches for another package instead

# Partial view of the environment



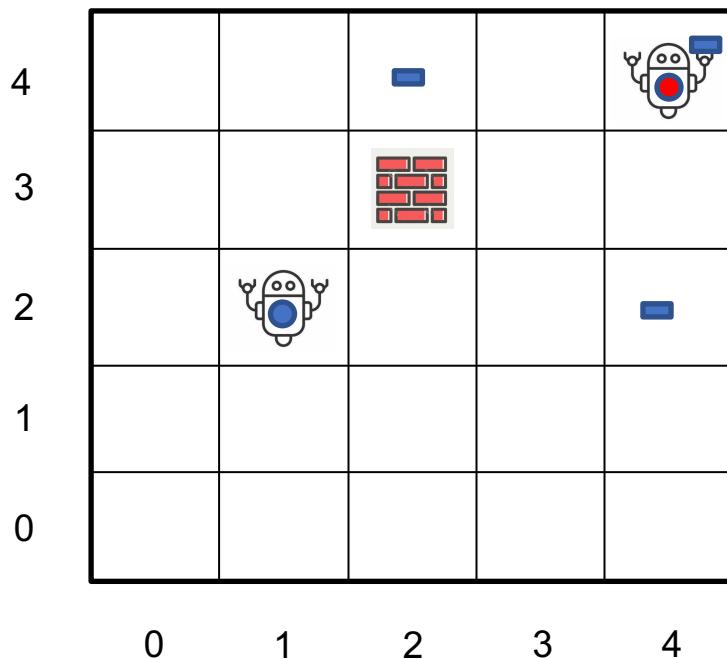
Agent 2

Time	Object	X	Y
1	Ag_2	1	2
1	Pack_1	2	4
1	Pack_2	4	2
1	Ag_1	4	4

Distance Ag\_1, Pack\_1, Pack\_2 = 2

Distance Ag\_2, Pack\_1, Pack\_2 = 3

# More on Belief estimation



Agent 2

Time	Object	X	Y	P
1	Ag_2	1	2	
1	Pack_1	2	4	?
1	Pack_2	4	2	?
1	Ag_1	4	4	

How does **Ag\_2** estimate the probability that **Pack\_1** is **still** at (2,4) and **Pack\_2** is **still** at (4,2) at a future time, given that **Ag\_1** might have moved toward them in the meantime?

# Belief Estimation

- **Belief estimation problem under uncertainty**, based on:
  - Agent **positions**
  - **Time** since last observation ( $t=1$ )
  - **Distance** to targets
  - Risk that another agent has already taken the package

## Manhattan Distances

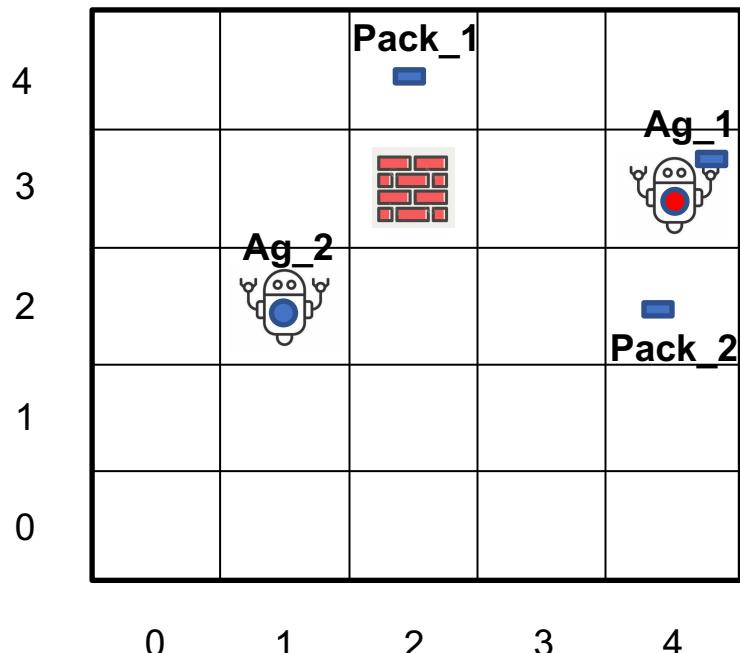
---

Package	Pos	Ag_1	Ag_2
Pack_1	(2,4)	2	3
Pack_2	(4,2)	2	3

---

- **Decay function for uncertainty:**
  - $D(d) = e^{-(\lambda \cdot d)}$
  - where  $d$  is distance from A2, and  $\lambda$  is a decay constant
- **Risk function for A1 stealing the package:**
  - $R = e^{-(\lambda \cdot dA1)}$
  - the closer A1 is, the **higher the risk** of pickup
- Let's pick  $\lambda = 0.3$ 
  - A2's distance: 3  $\rightarrow D = e^{-(0.3 \cdot 3)} \approx 0.406$
  - A1's distance: 2  $\rightarrow P = 1 - e^{-(0.3 \cdot 2)} \approx 1 - 0.549 = 0.451$
  - Final belief (probability P1 is still there):
    - **P(P1 still there) =  $D \times P = 0.406 \times 0.451 \approx 0.183$**
    - **P(P2 still there) =  $P(P1 still there) = 0.183$**
- **A2 has no strong preference** between the two

# Now, consider ...



Manhattan Distances

Package	Pos	Ag_1	Ag_2
Pack_1	(2,4)	3	3
Pack_2	(4,2)	1	3

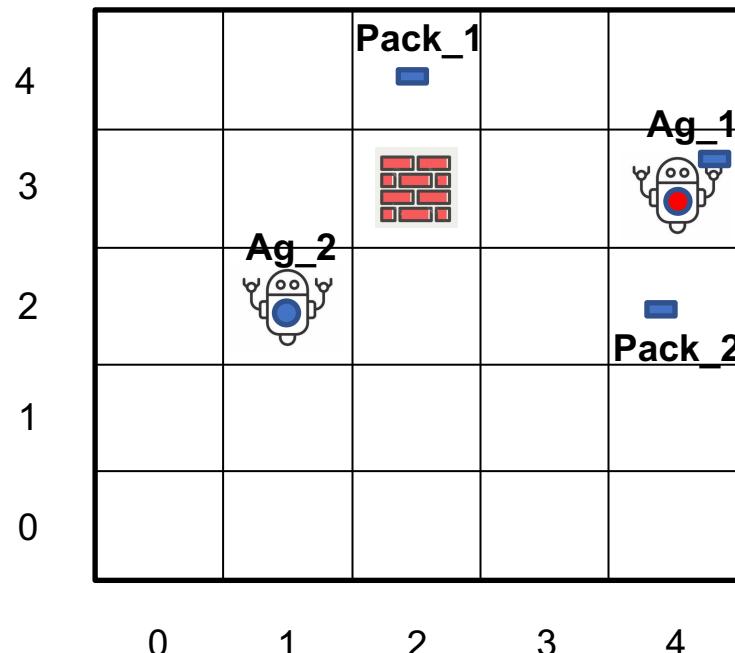
- A1 is **closer** to P2
- Both agents are equally distant from P1

A2's final belief (probability P1 is still there):

- $P(P2 \text{ still there}) = D \times (1-R) = 0.406 \times (1-0.741) \approx 0.105$
- $P(P1 \text{ still there}) = 0.406 \times (1-0.549) = 0.183$

A2 should prioritize going for P1, not P2

# ... and now



Package	\$	Pos	Ag_1	Ag_2
Pack_1	5	(2,4)	3	3
Pack_2	2	(4,2)	1	3

$$P(P2 \text{ still there}) = 0.105$$
$$P(P1 \text{ still there}) = 0.183$$

*Expected Value (EV)= P(available) x Reward*

$$EV(P1)=0.183 \times 5=0.915$$

$$EV(P2)=0.105 \times 2=0.21$$

**A2 should go for P1, because:**

- Even though it's equally distant, and A1 is closer to P2
- P1 offers a much better expected return

**But what about A1? Does it not should go for P1 as well?**

- Let's define the probability that A1 goes for a given package as:

$$P(A1 \rightarrow P_i) = \frac{EV(P_i)}{\sum_j EV(P_j)}$$

$$EV(P1) = 5 ; EV(P2)=2 ; Total =7$$

$$P(A1 \rightarrow P1) = 0.714$$

$$P(A1 \rightarrow P2) = 0.286$$

A2's beliefs

$$P(\text{available}) = D(d_{A2}) \cdot (1 - P(A1 \rightarrow P_i) \cdot e^{-\cdot d_{A1}})$$

$$P(P1) = 0.406 \times (1 - 0.714 \times 0.406) = 0.288$$

$$P(P2) = \dots = 0.32$$

$$\rightarrow EV(P1) = 0.241 \times 5 = 1.445$$

$$EV(P2) = 0.105 \times 2 = 0.64$$

Even after considering that A1 is likely to prioritize P1, the expected value of going for P1 is still higher for A2. Now A2's more informed, realistic belief update that considers A1's rationality.

# Beliefs and Introspective abilities

- What about beliefs concerning my intentions, desires, plans, actions? (**Introspection**)
  - If “I intend G, do I believe I intend to achieve G?”
  - Necessary to reasoning about intentions
    - If there is the opportunity to **pick\_up(Pack\_2)**
    - What about my current intentions? B: {**Intend(pick\_up(Pack\_1))**}
    - Are **Intend(pick\_up(Pack\_1))** and **Intend(pick\_up(Pack\_2))** consistent?
  - Not easy to implement
  - Synchronization between Beliefs – Intentions – Plans
  - Easy to get into self-contradictory reasoning

# Beliefs about other agents' mental states

- Beliefs about **other agents' beliefs**
  - I believe you believe - B:{`believe(Ag_2, In(Pack_1, 2, 4))`}
  - Important for coordination, negotiation, and competition
  - We will see more on agents' communication and the speech act theory
- Beliefs about **other agents' Intentions**
  - I believe you Intend - B:{`intend(Ag_2, pick_up(Pack_1))`}
  - Reasoning about the others' behaviours (coordination, negotiation, and competition)
  - Prediction of others' actions – need to explore possible plans
- Beliefs about **other agents' plans**
  - I believe you have the plan of  
B:{`plan(Ag_2, [move(UP), move(RIGHT), pick_up(Pack_1)])`}
  - Usually related to intentions but not always intentions are known
    - I know it will follow that path, but I don't know why
  - Prediction of others' actions

# For the project

- We do not use logical deduction  
 $B = \{\alpha, \alpha \vdash \beta\}$  means also that  $\beta \in B$
- We only use facts (true/false) and environmental constraints  
(integrety constraint like a DB)

fact:            t:      $\text{In}(\text{Ag\_1}, 1, 2)$

constraint: t:      $\text{In}(\text{Ag\_1}, 1, 2)$

                    t+1:  $\text{In}(\text{Ag\_1}, 1, 2) \vee \text{In}(\text{Ag\_1}, 1, 1) \vee \text{In}(\text{Ag\_1}, 2, 2)$

# Implementing the BDI Control Loop

Autonomous Software Agents

A.A. 2024-2025

**Prof. Paolo Giorgini**

**Dr. Marco Robol**



UNIVERSITY OF TRENTO - Italy

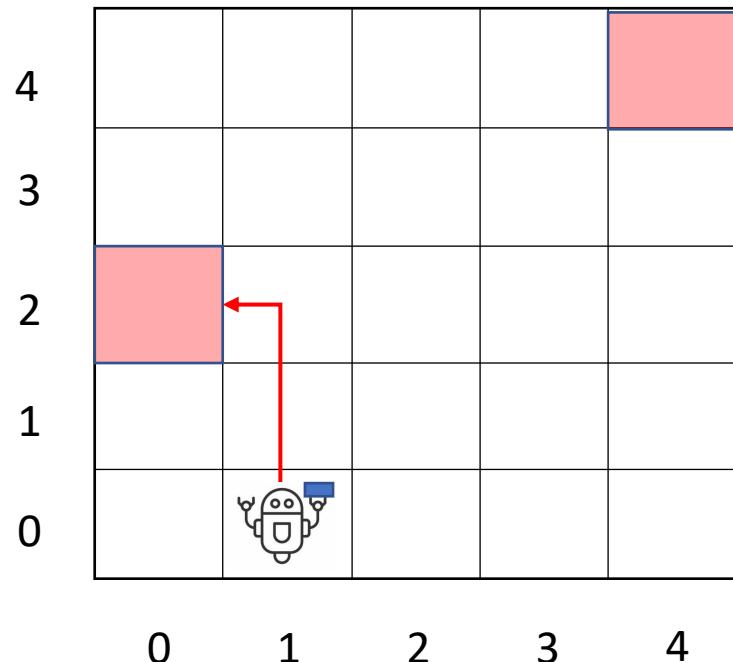
Department of Information  
and Communication Technology

# BDI control loop

- Given a set of beliefs **B**
  1. Decide about possible intentions to adopt
  2. Select new intentions to adopt
  3. Revise the Intention set **I**
  4. Revise and/or select new plans **P** for **I**
  5. Execute plans

Agent Control Loop Version 7

- 1.
2.  $B := B_0;$
3.  $I := I_0;$
4. while true do
5.     get next percept  $\rho$ ;
6.      $B := brf(B, \rho);$
7.      $D := options(B, I);$
8.      $I := filter(B, D, I);$
9.      $\pi := plan(B, I);$
10.    while not ( $empty(\pi)$   
          or  $succeeded(I, B)$   
          or  $impossible(I, B)$ ) do
11.       $\alpha := hd(\pi);$
12.       $execute(\alpha);$
13.       $\pi := tail(\pi);$
14.    get next percept  $\rho$ ;
15.     $B := brf(B, \rho);$
16.    if  $reconsider(I, B)$  then
17.         $D := options(B, I);$
18.         $I := filter(B, D, I);$
19.    end-if
20.    if not  $sound(\pi, I, B)$  then
21.         $\pi := plan(B, I)$
22.    end-if
23. end-while
24. end-while



-- t=0 *set of possible Beliefs*  
 $B = \{in(1,0), carry(pack\_1), del\_zone(0,2), del\_zone(4,4)\}$   
 $I = \{\}$   
 $P = \{\}$   
 Do: Null

**Intention Rules:** if (prec) then Int

if (`carry(Pck)` ^ `del_zone(X,Y)`) { 0 += *set of options* `in(Pck,X,Y)` }

$\rightarrow 0 = \{in(pack\_1, 0, 2), in(pack\_1, 4, 4)\}$  # options  
*↳ 2 possible positions to deliver package*

**Selection:**  $S = select(B, I, 0)$

$\rightarrow S = \{in(pack\_1, 0, 2)\}$ . # selected intentions

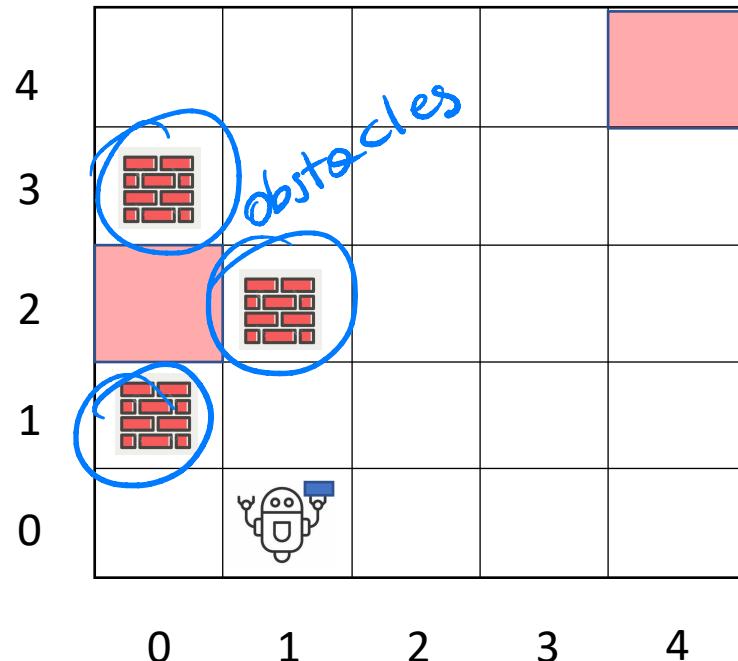
**Intentions Revision:**  $I = I\_revision(B, I, S)$

$\rightarrow I = \{in(pack\_1, 0, 2)\}$  # new intentions

**Plans selection or Planning:**  $P = planning(B, I, P)$

$\rightarrow P = \{p1\}$  # set of plans for I

$\rightarrow p1 = \{move(UP), move(UP), move(LEFT), put\_down(pack\_1)\}$



```
-- t=1
B = {in(1,0), carry(pack_1), del_zone(0,2), del_zone(4,4)}
I = {in(pack_1,0,2)}
P = {{move(UP), move(LEFT), put_down(pack_1)}}
Do: move(UP)
```

### Options generation:

```
if (carry(Pck) ^ del_zone(X,Y)) { 0 += in(Pck,X,Y) }

# again -> in(pack_1,0,2) and in(pack_1,4,4)
# but nothing changed from t=0 so it will be skipped
```

```
-- t=2
B = {in(1,1), carry(pack_1), del_zone(0,2), del_zone(4,4),
      in(block_1,0,1), in(block_2,1,2), in(block_3,0,3)}
I = {in(pack_1,0,2)}
P = {{move(LEFT), put_down(pack_1)}}
Do: move(UP)
```

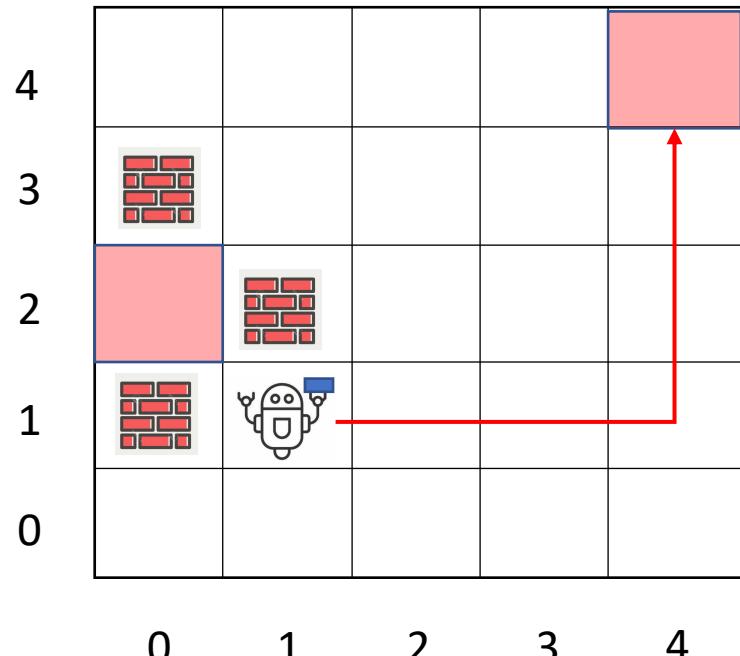
**Action failure:** move(UP)

**Re-Planning:** P = planning (B,I,P)

→ P = {} # no plans for in(pack\_1,0,2)

→ Now the agent have to consider other options

**Intentions Update:**  $I = I\_update(B, I, \text{in}(pack\_1, 0, 2))$



#  $\text{in}(pack\_1, 0, 2)$  is not possible anymore

$\rightarrow I = \{\}$  #  $\text{in}(pack\_1, 0, 2)$  is dropped

**Options generation:**

```
if (carry(Pck) ^ del_zone(X,Y)) { 0 += in(Pck,X,Y) }

# again -> in(pack_1,0,2) and in(pack_1,4,4)
# in this case in(pack_1,0,2) should not be considered
# and only in(pack_1,4,4) will be added

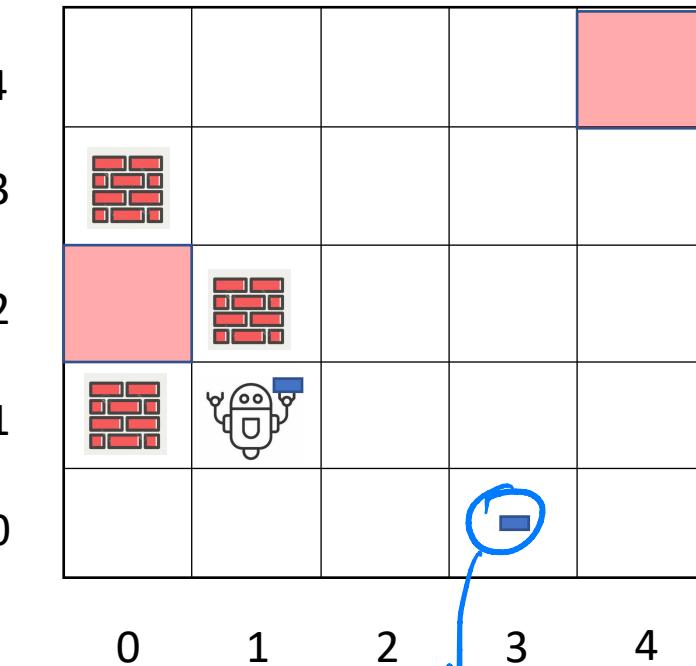
→ 0={in(pack_1,4,4)} # options
```

**Selection:**  $\rightarrow S = \{\text{in}(pack\_1, 4, 4)\}.$

**Intentions Revision:**  $\rightarrow I = \{\text{in}(pack\_1, 4, 4)\}$

**Plan selection:**

$\rightarrow P = \{\{\text{move}(RIGHT), \text{move}(RIGHT), \text{move}(RIGHT), \text{move}(UP),$   
 $\text{move}(UP), \text{move}(UP), \text{put\_down}(pack\_1)\}\}$



New package appears while executing the plan  
Depending on the strategy, the agent decide what to do

```
# with priority on Intentions
# under the hypothesis that Intentions
# cannot run in parallel
```

```
-- t=3
B = {in(1,1), carry(pack_1), del_zone(0,2), del_zone(4,4)
      in(block_1,0,1), in(block_2,1,2), in(block_3,0,3) }
I = {in(pack_1,4,4)}
P = {{move(RIGHT), move(RIGHT), move(UP),
       move(UP), move(UP), put_down(pack_1)}}
Do: move(RIGHT)
```

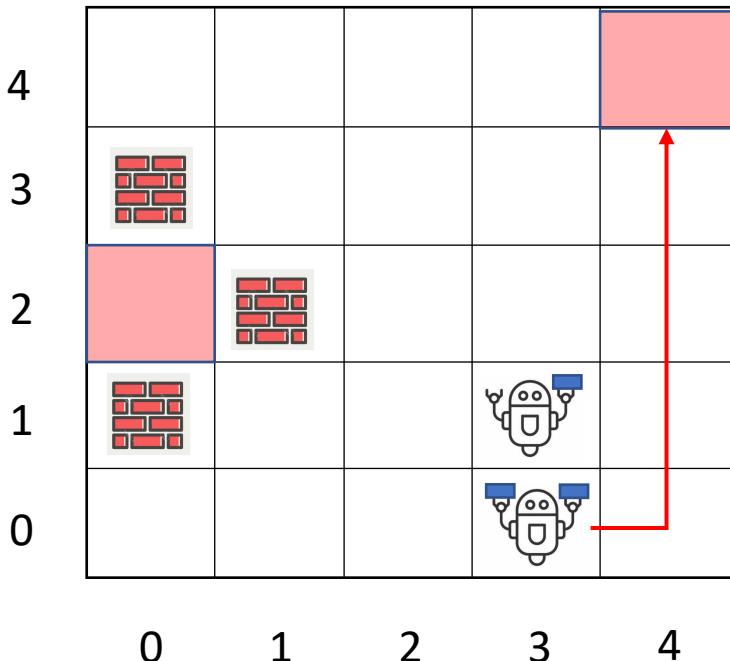
```
-- t=4
B = {in(2,1), carry(pack_1), del_zone(0,2), del_zone(4,4)
      in(block_1,0,1), in(block_2,1,2), in(block_3,0,3) }
I = {in(pack_1,4,4)}
P = {{move(RIGHT), move(UP), move(UP), move(UP),
       put_down(pack_1)}}
Do: move(RIGHT)
```

```
-- t=5
B = {in(3,1), carry(pack_1), del_zone(0,2), del_zone(4,4)
      in(block_1,0,1), in(block_2,1,2), in(block_3,0,3),
      in(pack_2,3,0) }
I = {in(pack_1,4,4)}
P = {{move(UP), move(UP), move(UP), put_down(pack_1)}}
Do: move(RIGHT)
```

**Options generation + selection:** → S={pick\_up(pack\_2)}

**Intentions Revision :** → I={carry(pack\_2), in(pack\_1,4,4)}

**Re-planning:** → I={move(DOWN), pick\_up(pack\_2)}



```

-- t=6
B = {in(3,1), carry(pack_1), del_zone(0,2), del_zone(4,4)
      in(block_1,0,1), in(block_2,1,2), in(block_3,0,3),
      in(pack_2,3,0) }
I = {carry(pack_2), in(pack_1,4,4)}
P = {{pick_up(pack_2)}}
Do: move(DOWN)

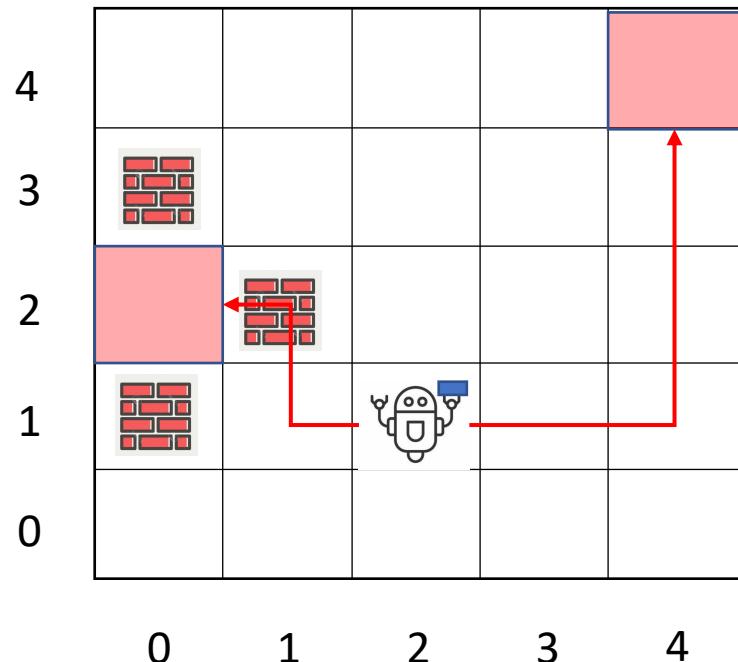
-- t=7
B = {in(3,0), carry(pack_1), del_zone(0,2), del_zone(4,4)
      in(block_1,0,1), in(block_2,1,2), in(block_3,0,3),
      in(pack_2,3,0) }
I = {carry(pack_2), in(pack_1,4,4)}
P = {}
Do: pick_up(pack_2)

-- t=8
B = {in(3,0), carry(pack_1), del_zone(0,2), del_zone(4,4)
      in(block_1,0,1), in(block_2,1,2), in(block_3,0,3),
      in(pack_2,3,0) }
I = {carry(pack_2), in(pack_1,4,4)}
P = {}
Do: NULL

-- t=9
B = {in(3,0), carry(pack_1), del_zone(0,2), del_zone(4,4)
      in(block_1,0,1), in(block_2,1,2), in(block_3,0,3),
      carry(pack_2) }
I = {in(pack_1,4,4)}
P = {{move(UP), move(UP), move(UP), move(UP), put_down(pack_1)}}
Do: move(RIGHT)

```

# Re-Planning for the second intention



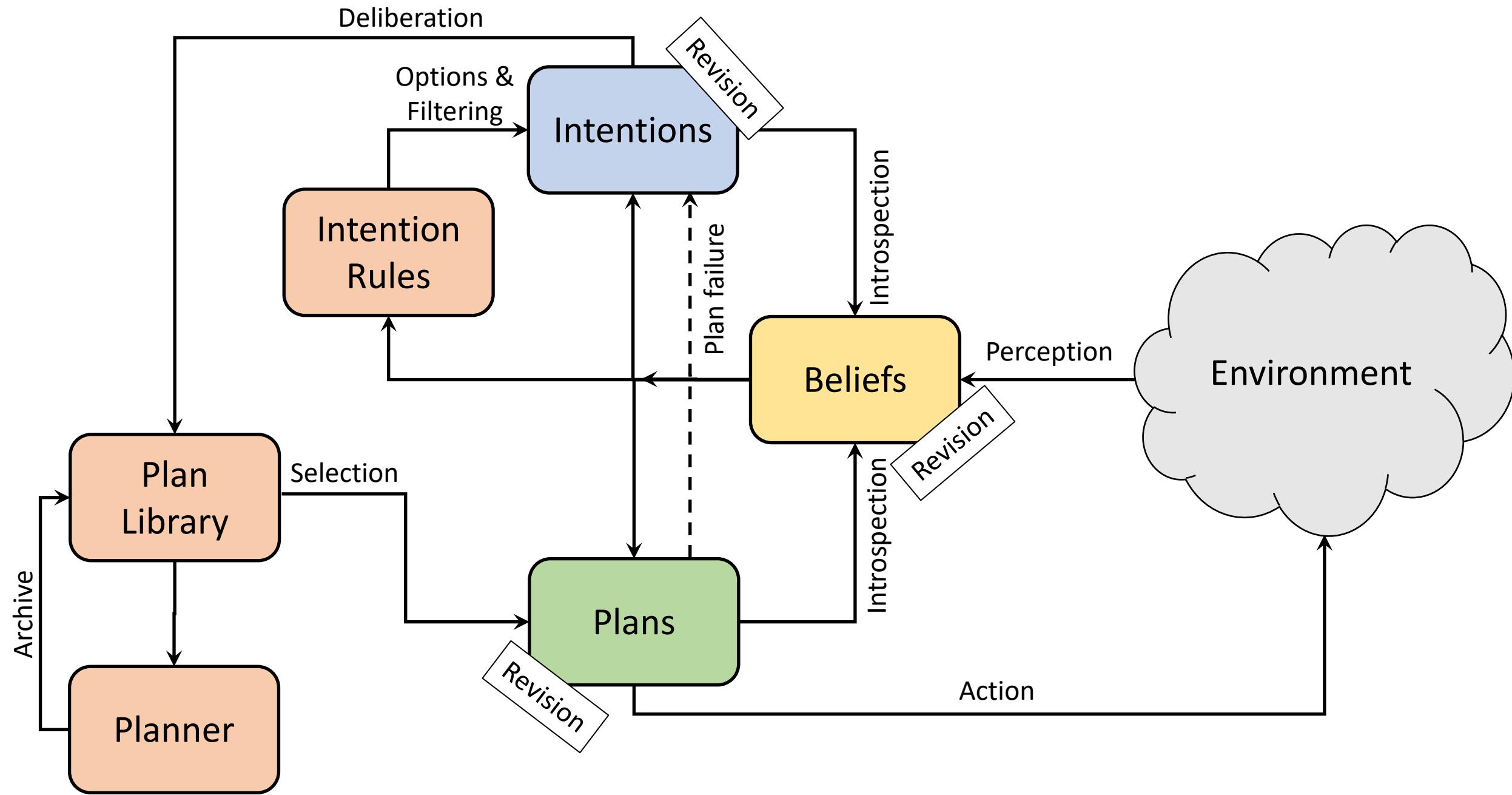
```
-- t=4
B = {in(2,1), carry(pack_1), del_zone(0,2), del_zone(4,4)
     in(block_1,0,1), in(block_3,0,3) }
I = {in(pack_1,4,4)}
P = {{move(RIGHT), move(UP), move(UP), move(UP),
       put_down(pack_1)}}
Do: move(RIGHT)
```

**Options generation + selection:** → S={in(pack\_1,0,2)}

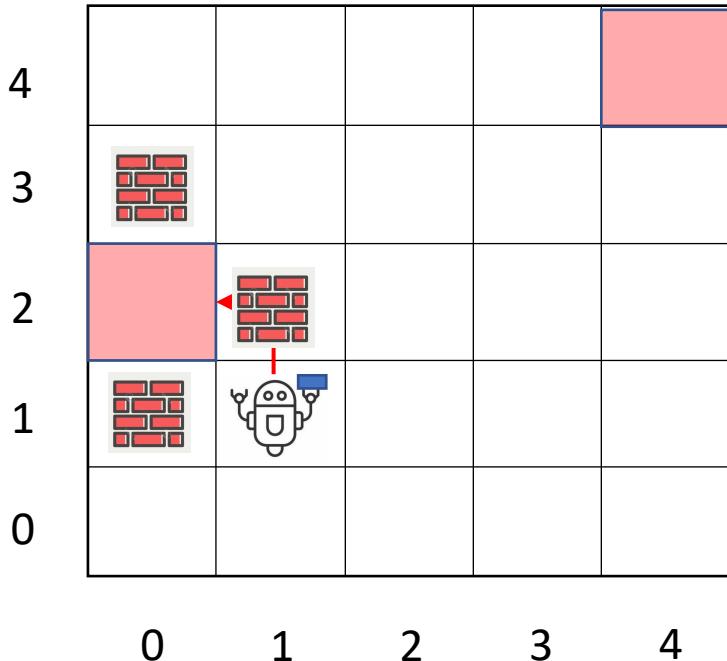
**Intentions Revision :** → I={in(pack\_1,0,2)}

**Re-planning:** → I={move(LEFT), move(UP), move(LEFT),
 put\_down(pack\_q)}

```
-- t=5
B = {in(2,1), carry(pack_1), del_zone(0,2), del_zone(4,4)
     in(block_1,0,1), in(block_3,0,3) }
I = {in(pack_1,0,2)}
P = {move(UP), move(LEFT), put_down(pack_q)}
Do: move(LEFT)
```



# The order of things and timing



```
-- t=0
B = {in(1,1),carry(pack_1),del_zone(0,2), del_zone(4,4)}
I = {}
P = {}
Do: Null
```

**Options + Filtering + Int. Revision + Planning**

What about if in the meanwhile things in the environment change?

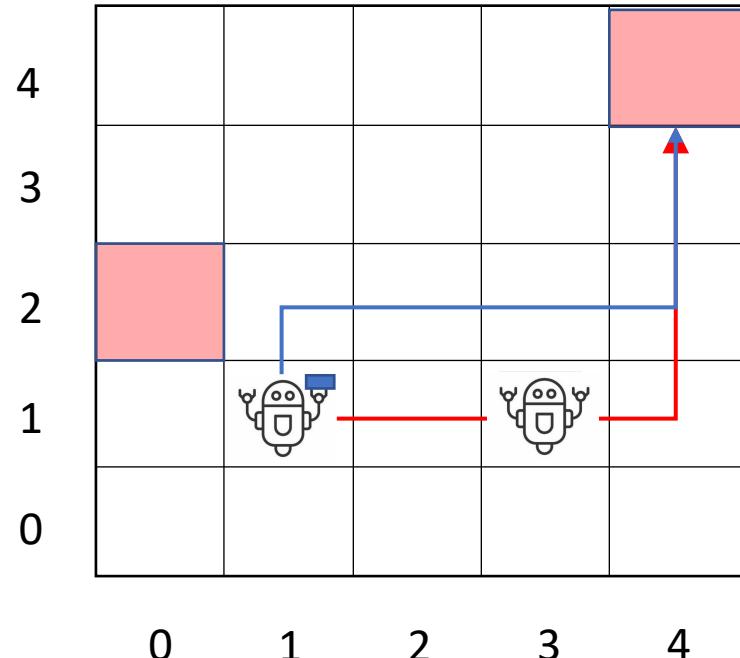
in(block\_1,0,1), in(block\_2,1,2), in(block\_3,0,3)

```
-- t=1
B = {in(1,1),carry(pack_1),del_zone(0,2), del_zone(4,4)
      in(block_1,0,1), in(block_2,1,2), in(block_3,0,3) }
I = {in(pack_1,0,2)}
P = {{move(LEFT),put_down(pack_1)}}
Do: move(UP)
```

# When do we check?

- Much it depends on the application and on the dynamicity of the environment
  - check beliefs at any decision point
    - Option / filtering / deliberation / planning / action
  - check soundness or optimality of the plan before any action
    - Always evaluate the best way to achieve a goal
  - revise intentions before any action
    - This may bring the agent to a schizophrenic behaviour but it could be useful to have agents that are opportunistic

# Be careful



```
-- t=0
B = {in(1,1),carry(pack_1),del_zone(0,2), del_zone(4,4)}
I = {in(pack_1,4,4)}
P = {{move(RIGHT), move(RIGHT),move(UP), move(UP), move(UP),
put_down(pack_1)}}
Do: move(RIGHT)
```

**The plan is not sound anymore: → Re-planning**

```
# since the block will remain in the position (3,1),
# the plan is not sound now and in the future
```

**The plan is not sound anymore: → Re-planning ?**

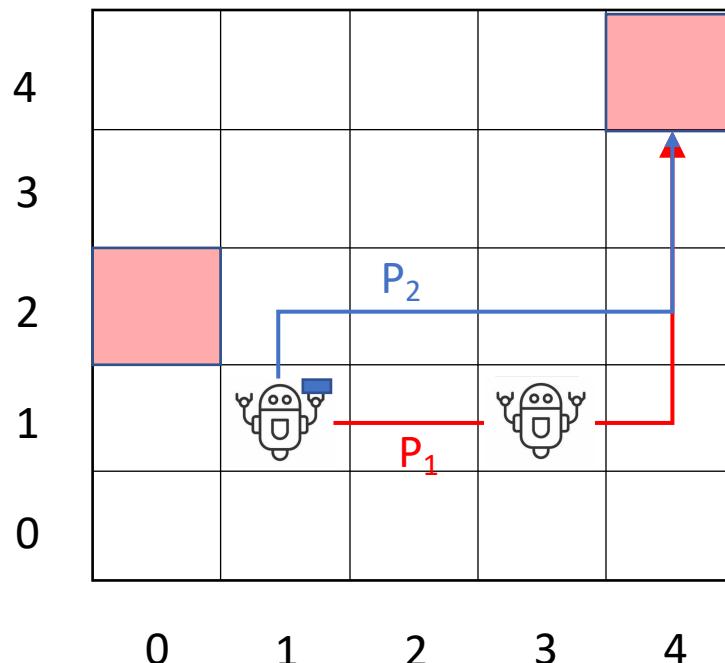
```
# Will the agent remain in the position (3,1)?
# if we replan, it may happen that the new plan will not
# be sound anymore in the next step
```

The agent can distinguish between a block and a moving block

- Different strategies in case of moving blocks

# Moving obstacles

- Consider moving obstacles as permanent obstacles
  - No differences in the behaviour
- Elaborating a probability model for the trajectory of the obstacle
  - The new plan will be selected on the base of the probability that the moving obstacle can make the plan not sound at some point ( $P_1$  vs  $P_2$ )
- Other strategies
  - Eg. , stay away from other agents or techniques based on game theory



# Using the probability model

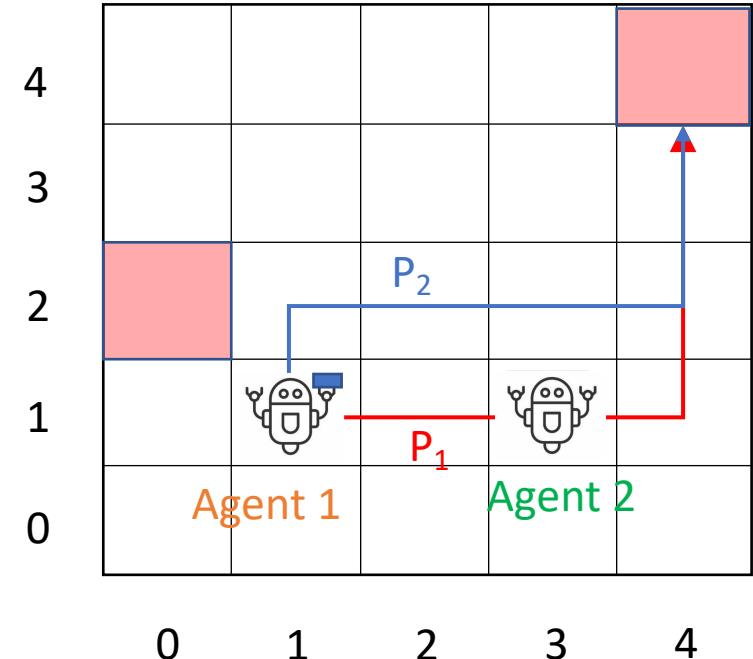
Under the hypothesis of having the same probability that **Agent 1** move in any tile or stay where it is

The probability that **Agent 2** is an obstacle for **Agent 1** executing  $P_1$  is:

- $P(\text{move(LEFT)}) + P(\text{not move}) = 1/5 + 1/5 = 0.4$

The probability that **Agent 2** is an obstacle for **Agent 1** executing  $P_2$  is:

- ....

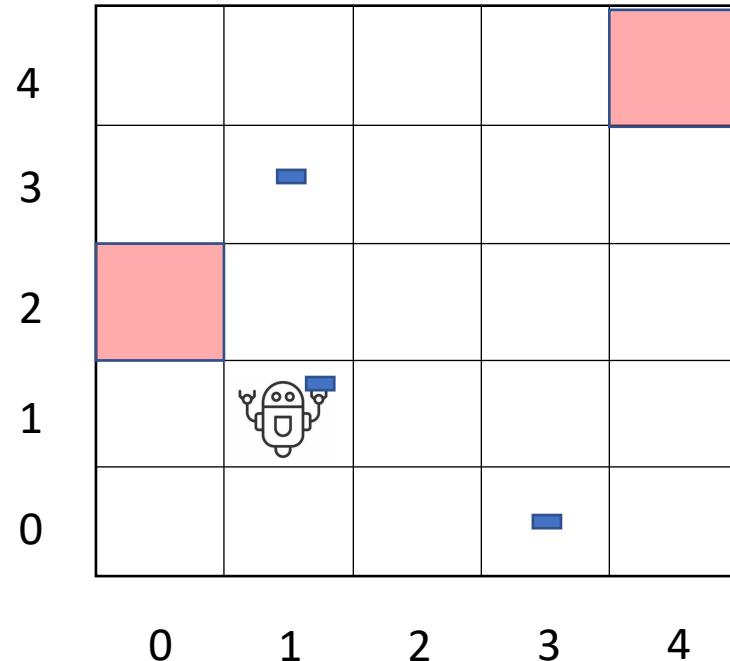


# Intention Revision

# Intention Revision

- The process:  
Beliefs Revision -> Options Generation and Selection -> Intentions Revision
- When does an agent need to revise its intentions?
  - Selected options  $\text{Opt}$  and current Intentions  $I$  are inconsistent
  - Similar to Belief Revision the agent must decide:
    - $\text{Opt} \cup I \rightarrow \text{new } I$
    - Note: Intentions can be achieved in parallel or in a sequential order (priority)  
(the agent must schedule the order of the intentions achievement)

# Consistency → re-scheduling



```
-- t=1
B = {in(1,1), carry(pack_1), del_zone(0,2), del_zone(4,4)}
I = {in(pack_1,0,2)}
P = {{move(UP), move(LEFT), put_down(pack_1)}}
Do: Null
```

```
-- t=2
B = {in(1,2), carry(pack_1), del_zone(0,2), del_zone(4,4)
     in(pack_2,3,0), in(pack_3,1,3)}
I = {in(pack_1,0,2)}
P = {{move(LEFT), put_down(pack_1)}}
Do: move(UP)
```

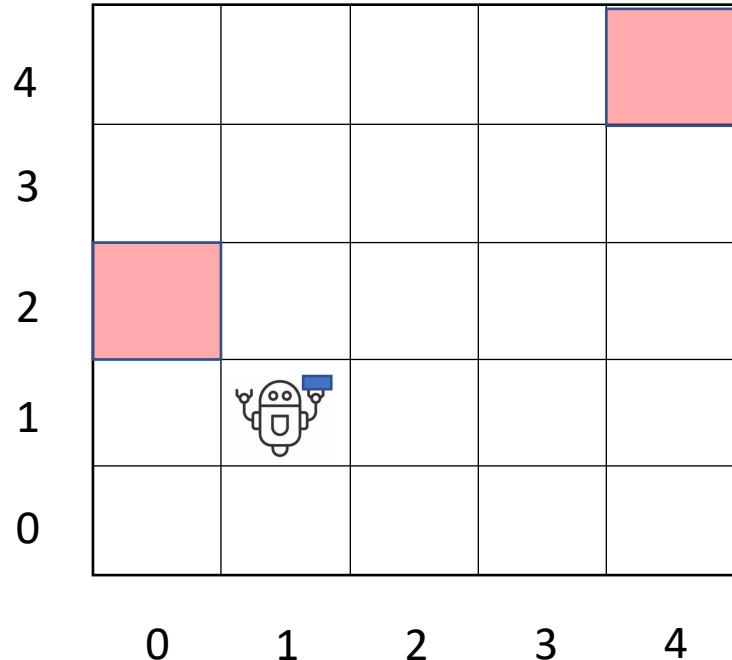
**Options generation:** → S={`pick_up(pack_2)`, `pick_up(pack_3)`}

# the agent can carry max 2 packs

New I = {`in(pack_1,0,2)`, `pick_up(pack_2)`, `pick_up(pack_3)`} or  
I = {`pick_up(pack_2)`, `in(pack_1,0,2)`, `pick_up(pack_3)`} or  
I = {`pick_up(pack_3)`, `in(pack_1,0,2)`, `pick_up(pack_2)`} or  
I = ....

# Re-planning is always needed

# Options and Intentions are inconsistent



```
-- t=1
B = {in(1,1),carry(pack_1), del_zone(4,4)}
I = {in(pack_1,4,4)}
P = {{move(RIGHT), move(RIGHT), move(RIGHT), move(UP),
      move(UP), move(UP), put_down(pack_1)}}
Do: Null

-- t=2
B = {in(2,1),carry(pack_1),del_zone(0,2), del_zone(4,4)}
I = {in(pack_1,4,4)}
P = {{move(RIGHT), move(RIGHT), move(RIGHT), move(UP),
      move(UP), move(UP), put_down(pack_1)}}
Do: move(RIGHT)

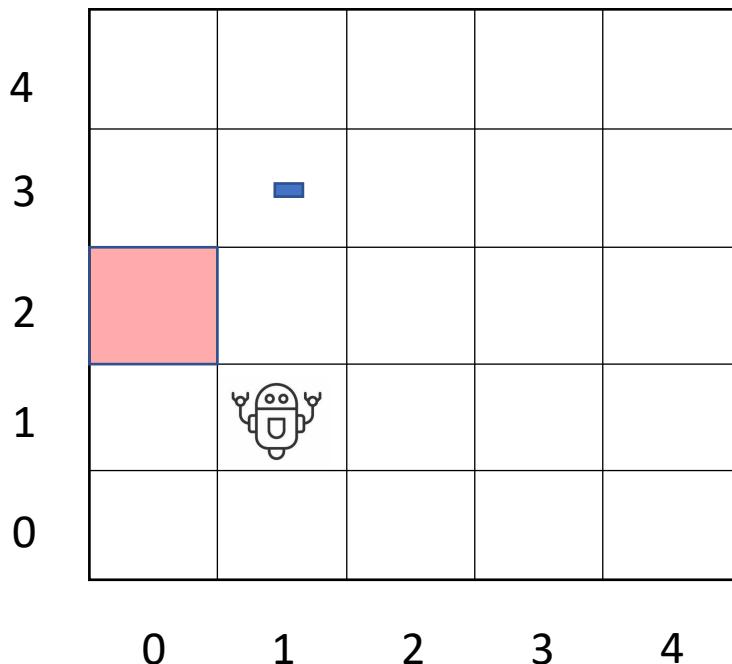
Options generation: → S={in(pack_1,0,2)}

# the option is inconsistent with the current intention

New I = {in(pack_1,0,2)} or
I = {in(pack_1,4,5)}
```

# The problem of consequences

If the agent intends  $\alpha$  and  $\alpha \vdash \beta$



```
-- t=1
B = {in(1,1), in(pack_1,1,3), del_zone(0,2)}
I = {}

# The option is to bring pack_1 in the delivery zone,
# but this means first pick_up the package and then
# delivering it

-- t=2
B = {in(1,1), in(pack_1,1,3), del_zone(0,2)}
I = {carry(pack_1), in(pack_1,1,3)}

# but what about the agent drops the intention to delivery
# pack_1. Does It keep the intention to pick up the package?

# like for beliefs the agent should distinguish between
derived and not derived intentions
```

# Intro to Planning

Autonomous Software Agents

A.A. 2024-2025

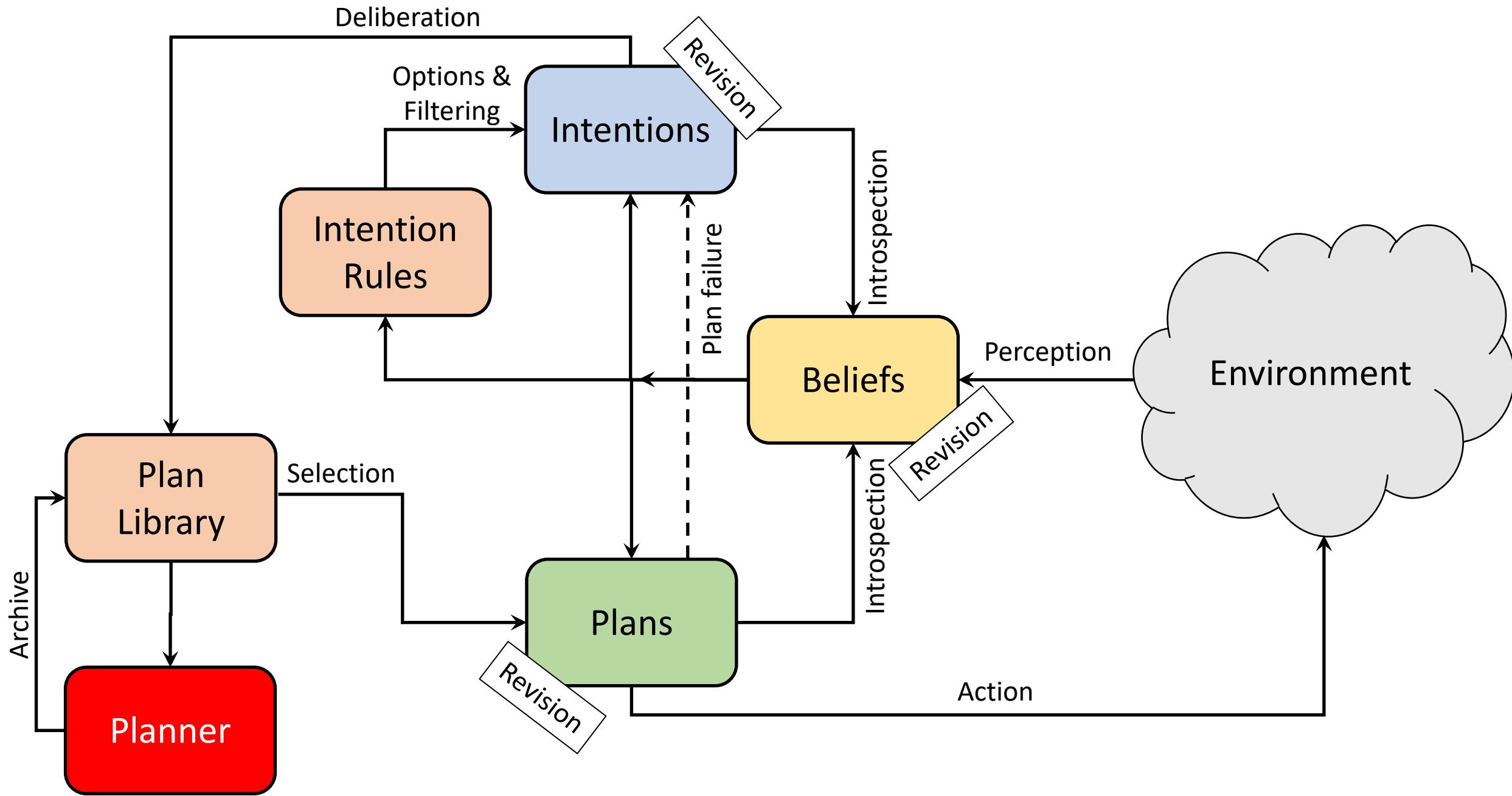
**Prof. Paolo Giorgini**

**Dr. Marco Robol**



UNIVERSITY OF TRENTO - Italy

Department of Information  
and Communication Technology



# What is Planning ?



# Planning as deliberation

An intelligent agent needs planning to decide best actions to do

- Programming-based agent - writing the agent control by **hand**
- Learning-based agent - infer the control by **experience** (Reinforcement Learning)
- Model-based approach - derive the control automatically by **reasoning**

**These approaches are not orthogonal!**

# What is Model-Based Planning?

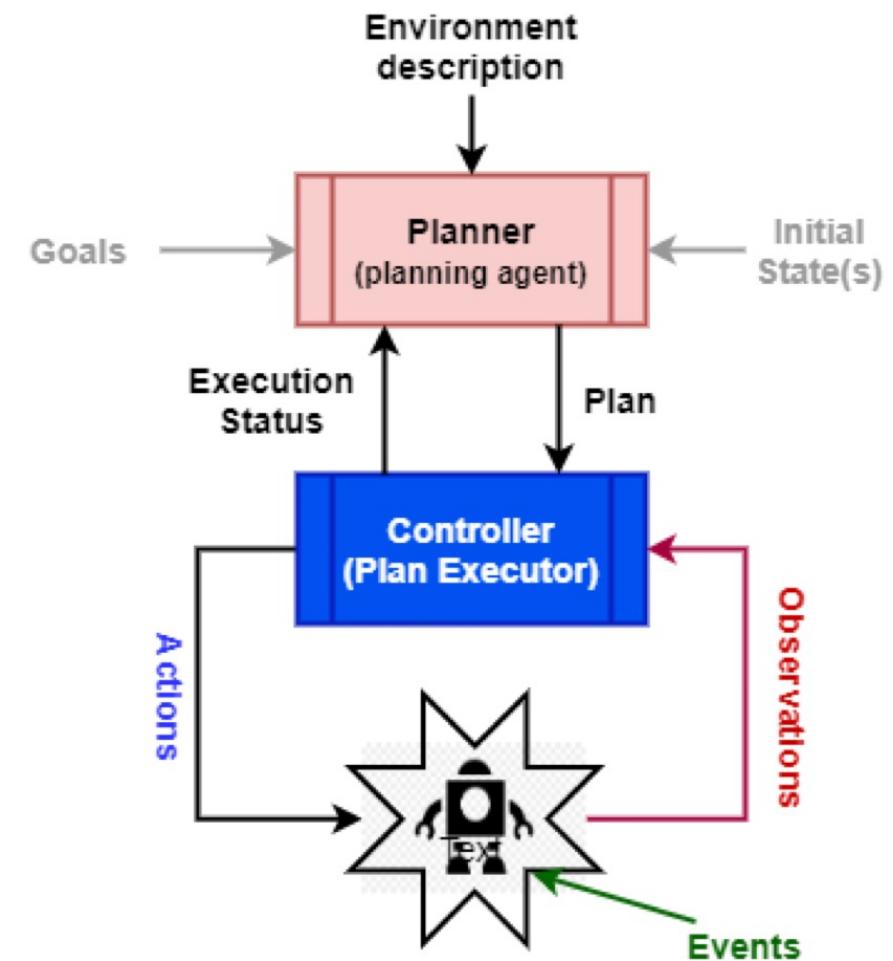
“Planning is the reasoning side of acting. It is an explicit deliberation process that chooses and organises actions, on the basis of their expected outcomes, in order to achieve some objective as best as possible.”

“...planning is the model based approach to action selection...”

“AI Planning is general problem solving”

# What is Model-Based Planning?

- In AI Planning the agent has **access to world dynamic**
  - **actions** describe how the world changes
  - **sensor-model** describe how to update the knowledge of the world
  - **goals** denote what the agent wants
  - **states** capture evolving relevant conditions of the world the agent is operating in
- A **Planner** is a domain independent program that can solve (find a plan for) **all** the planning problems starting from
  - **environment description**
    - including action capabilities
  - **problem description**
    - initial state(s) and goals



# AI Planning Problem

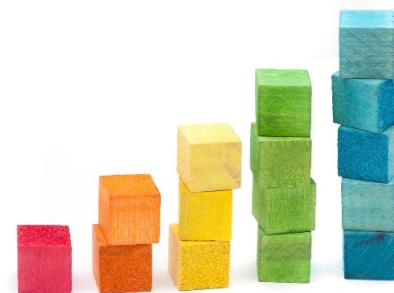
Given:

- A description of (possible) initial state(s)
- A description of desired goal states
- A description of a set of possible actions



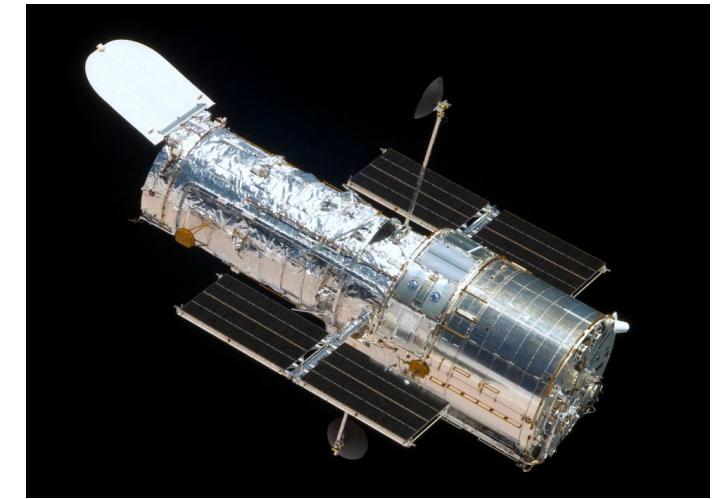
Generate:

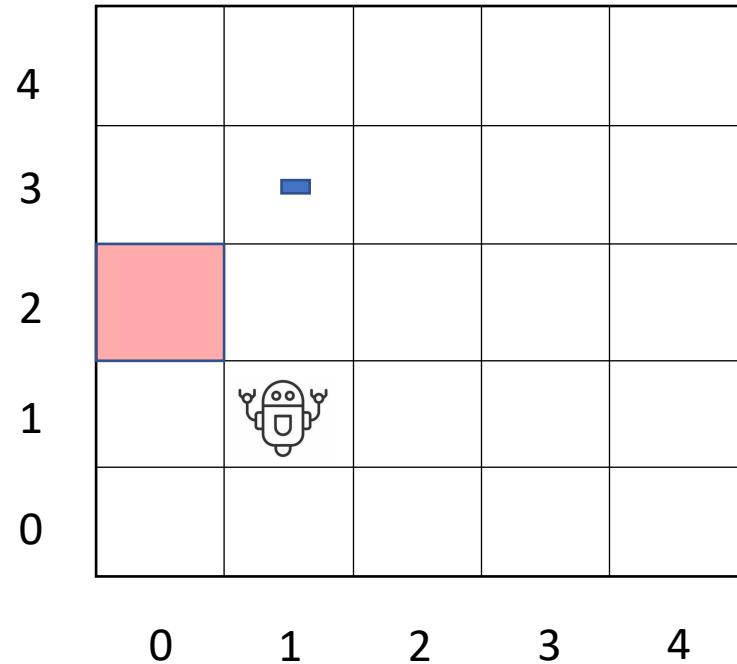
- A sequence of actions that leads to one of the goal states



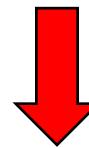
# Where do we use planning?

- Space Missions
- Hubble Telescope
- Logistics
- Games





B = {in(1,1), in(pack\_1,1,3), del\_zone(0,2)}  
I = {del(pack\_1,0,2)}



P = {{move(UP), move(UP), pick\_up(pack\_1),  
move(DOWN), move(LEFT), put\_down(pack\_1),}}

# Why is planning so difficult?

- Uncertainty about effects of actions
- Uncertainty about environment
- Uncertain sensing and perception
- Agents own actions may have bad effects
- Time and resources are limited!
- Preferences
- Complex goals
- Optimality

# Let's simplify things a bit...

- Known initial state
- Deterministic actions
- Simple action representation
- Instantaneous actions
- No deadlines and sufficient resources
- Fully observable
- Single agent /No external events
- No concurrent actions

We introduce constraints and simplification to focus on the reasoning behind planning problems

# Automated planning

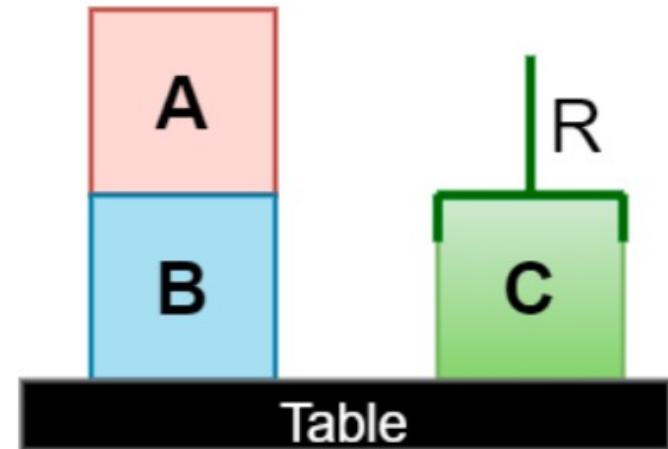
# Planning concepts

Speaking about planning problems we deal with:

- State
- Action
- Goal
- Plan

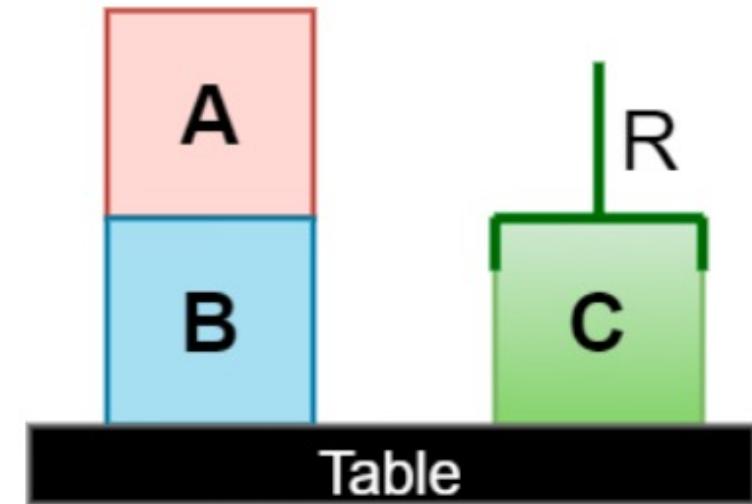
# Example: The Blocks World

- Objects
  - Blocks: A, B, C
  - Table: Table
  - Robot: R
- States: Conjunctions of ground literals built from *predicates*
  - (On A B), (On B Table), (On C Table),  
(Clear A), (Holding R C), (Free R)
- Actions: Operator schemas with variables
  - (Pickup ?R ?A), (Putdown ?R ?B)
- Domain Axioms:
  - 'At most one block on top of another'
  - 'Hand must be empty and block must be clear to pick up'



# What is a state?

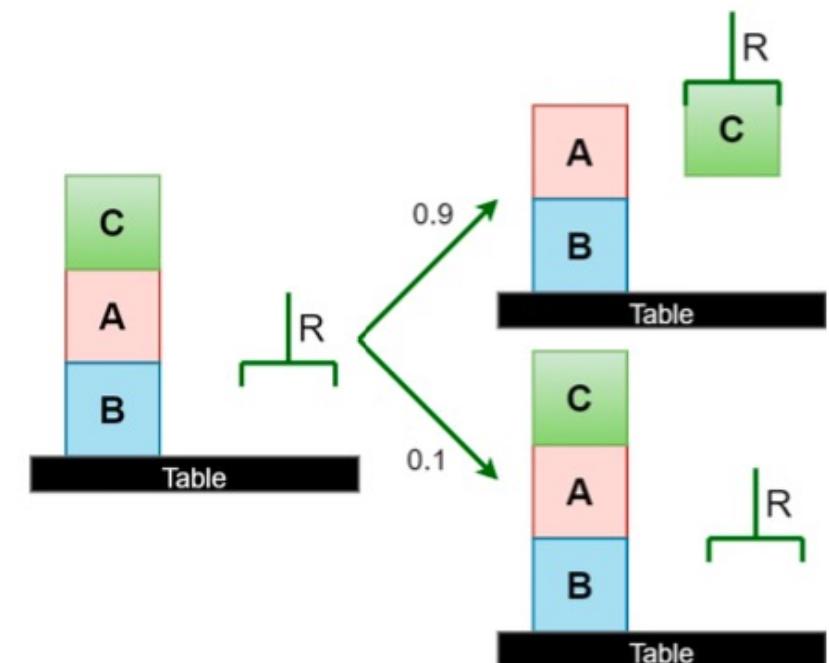
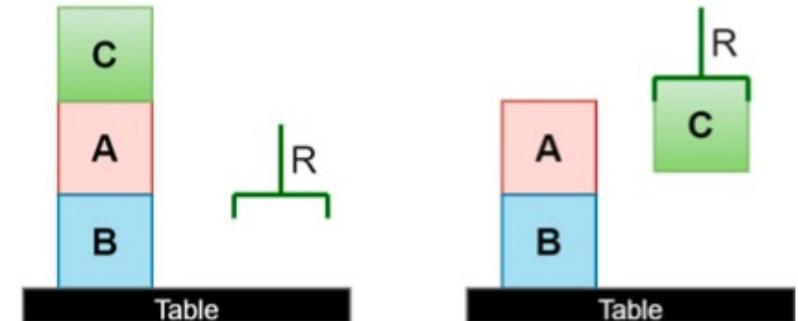
- A complete description of the world.
- Instantiation of all state variables
- Truth assignment of all variables



# What is an action?

Transition from one state to another

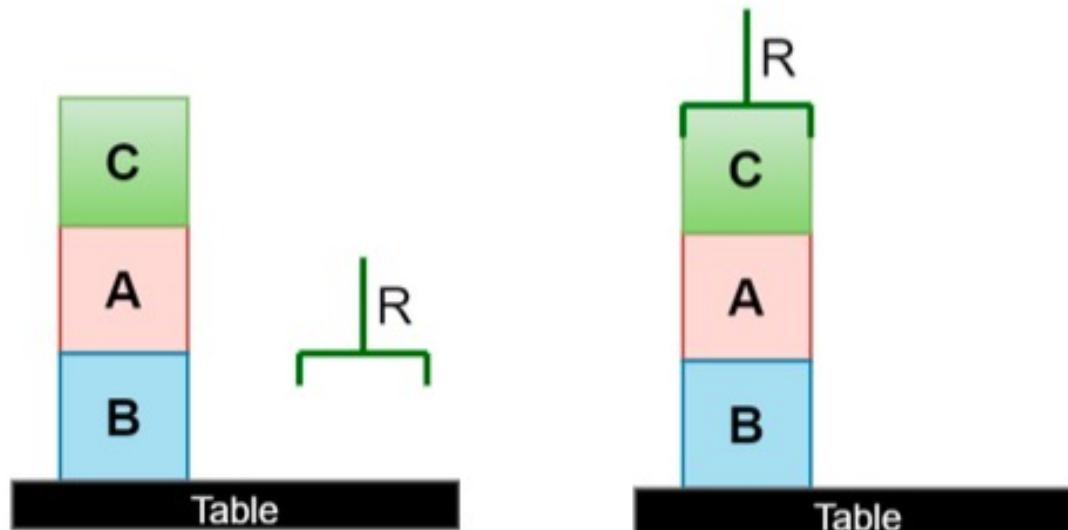
- Some actions may be applicable to some of the states.
  - (Pickup ?R ?X) applicable if (Block ?X), (Clear ?X), (Free ?R)
- Have given effects on the state.
  - Deterministic
  - Non-deterministic
  - Probabilistic



# What is a goal?

A desired set of states

- (On C A), (On A B), (On B Table)
  - (On C A), (On A B), (On B Table), (**Clear C**), (**Free R**)
  - (On C A), (On A B), (On B Table), (**Holding R C**)



# What is a plan?

- A **sequence** of instantiated actions
  - A plan is an ordered list of actions, already instantiated (i.e., with concrete parameters), to be executed one after the other to reach the goal: (pickup a) (stack a b)
- A **set** of instantiated actions
  - Without a strict order, the execution order may be flexible or determined dynamically (the exact ordering isn't important, or actions can happen in parallel or under certain constraints)
- A **tree** of instantiated actions
  - Where each node is an action and branches are alternative outcomes or conditional paths (non-deterministic or partially observable environments - different possible scenarios)
- A **policy** of instantiated actions
  - A mapping from states to actions, rather than a fixed sequence, it tells the agent what to do in any given situation - reinforcement learning or planning under uncertainty (e.g., MDPs, POMDPs)

# How to find a plan?

## Solution Techniques

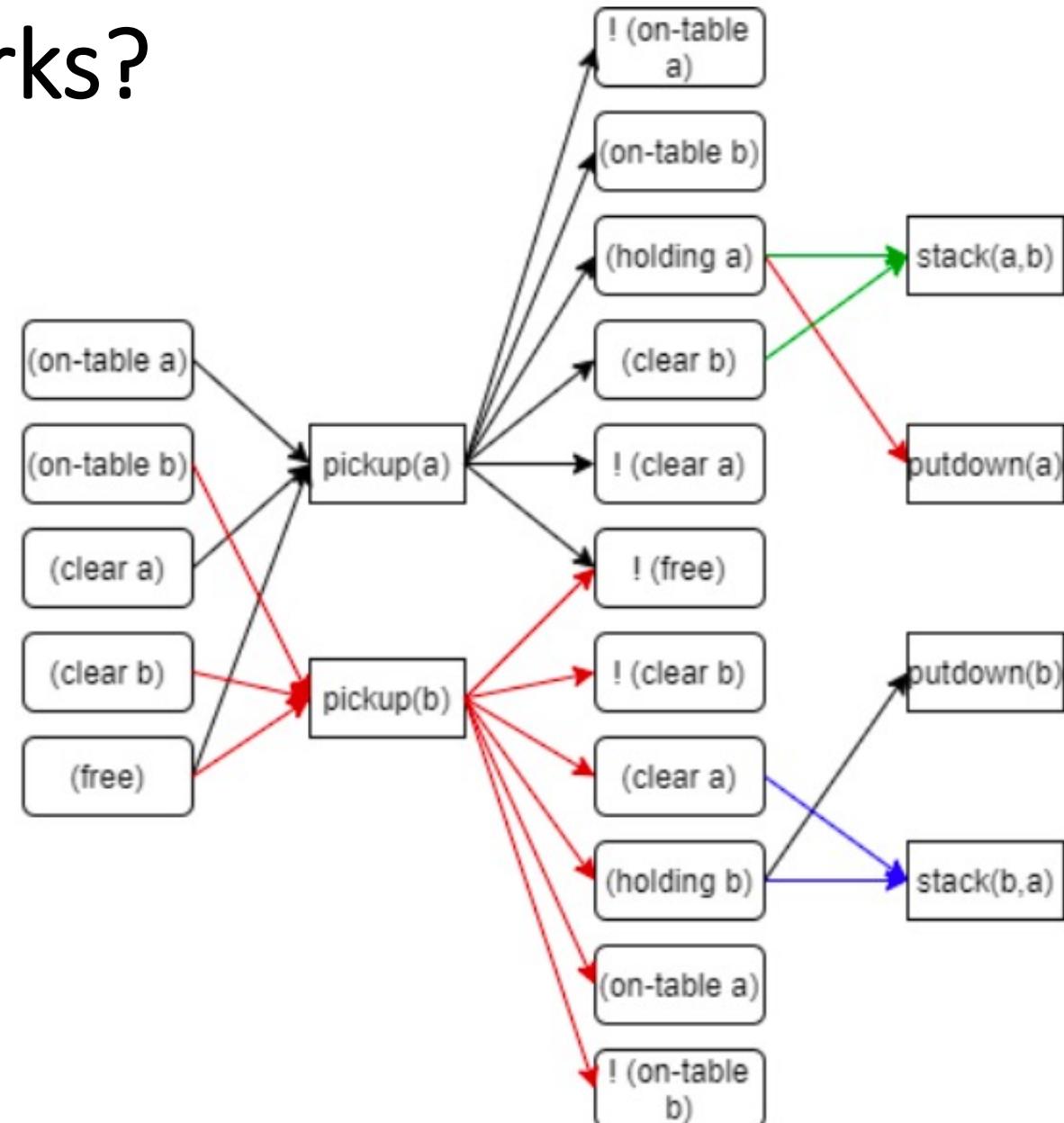
- Graph search
- Heuristic search
- Solve the transitioning matrix

## Complexity

- Exponential in the number of actions
- Polynomial in the number of states

# How does a planner works?

- The planning algorithm creates the **plan graph** exploiting e.g. Graphplan algorithm Blum and Furst [1]
  - Start with initial condition
  - Add actions with satisfied conditions
  - Add all effects of actions at previous levels
  - Add frame conditions
  - Repeat!



# How does it works?

- The planner creates the plan graph exploiting e.g. Graphplan algorithm.
- At each step it checks whether all of its goals exist.
  - if true → there is a chance that the plan graph contains a solution, and the planner has to search for it.
  - if false → the planner has to grow the graph out another level and try again.

The algorithm also performs a pruning phase, in which it finds and marks pairs of actions/propositions that are mutually exclusive.

- A more in-depth look of the Graphplan algorithm (with a visual example) is given at this link <http://bit.ly/MIT-graphplan>.

# STRIPS

In 1971 STRIPS (Stanford Research Institute Problem Solver) was developed as an automated planner. Later, the name STRIPS has been used to refer only to the formal language of the inputs

To actually deal with planning, we use specific language to express the domain and the problem. STRIP is a language to express planning problems in which:

- State is expressed by a set of ground literals, assumed false if not in the set
- Effects of actions described with two sets of ground literals, one to be added and one to be removed from the set describing the world state (add and delete lists)
- No explicit representation of time
- No logical inference rules

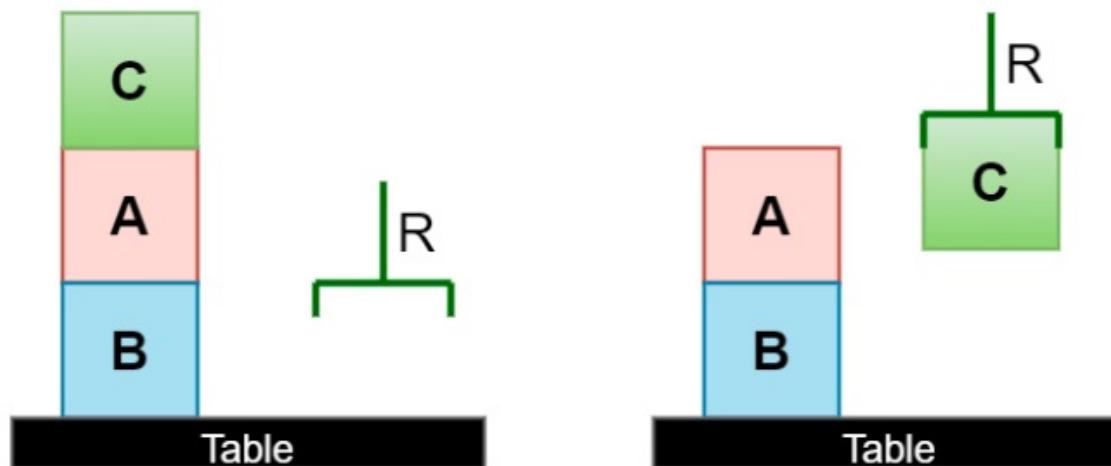
# Something more expressive 1

- Conditional Effects

```
(:action pickup
  :parameters (?X)
  :preconditions (and (BLOCK ?X) (free) (clear ?X) (On ?X ?Y))
  :effects (and (holding ?X) (when (and (BLOCK ?Y)) (and (Clear ?Y))) ; add
                (not (free)) (not (On ?X ?Y))) ; delete
```

- Quantified effects (forall (?x) (when (and (in ?x ?y))...))

- Disjunctive and negated preconditions (or (conn ?x ?y) (not (in ?y ?x)))



# Something more expressive 2

- Functional effects (increment ?x 10)
- Disjunctive effects
- Probabilistic effects
- Duration (actions no more instantaneous)
- External events, agents, concurrent events, etc
- Inference operators

# PDDL

Planning Domain Definition Language : standard encoding language for classical planning

<b>Objects</b>	Things in the world
<b>Predicates</b>	Properties of the objects
<b>Initial state</b>	The state of the world we start in
<b>Goal specification</b>	Things we want to be true
<b>Actions</b>	Ways of changing the state of the world

# How to put pieces together?

It works with two files, splitting content of planning problem in a domain file and a problem file

- A **domain** file for predicates and actions
- A **problem** file for objects, initial states, and goal descriptions

Which one can we re-use?

# PDDL: Domain file

```
(define (domain <domain name>)
  <PDDL directives>          ; specify PDDL language features
  <PDDL code for predicates>
  <PDDL code for actions>
)
```

# The blocks world domain file

```
;; domain file: block-domain-complete.pddl
(define (domain blocksworld)
  (:requirements :strips)

  (:predicates (clear ?x) (on-table ?x) (holding ?x) (on ?x ?y) (free))

  (:action pickup
    :parameters (?ob)
    :precondition (and (clear ?ob) (on-table ?ob) (free))
    :effect (and (holding ?ob)
                  (not (clear ?ob)) (not (on-table ?ob)) (not (free)))))

  (:action putdown
    ...
  ))
```

# Problem file

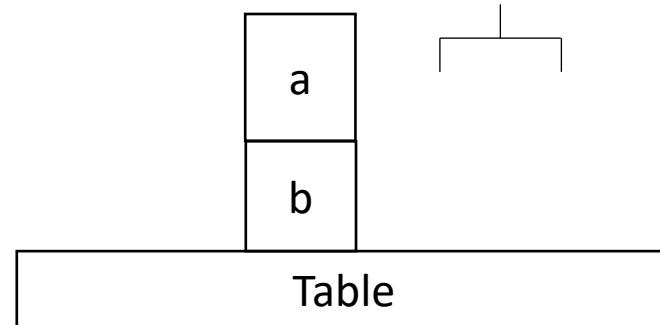
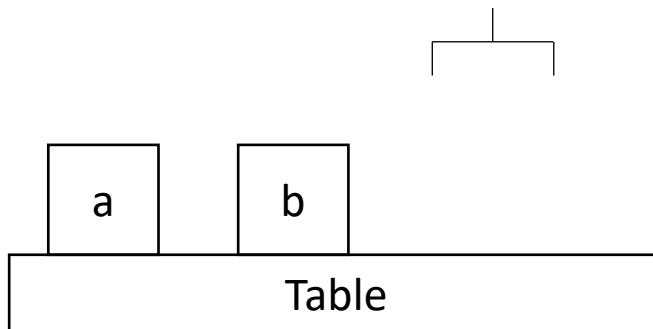
```
(define (problem <problem name>)
  (:domain <domain name>)
  <PDDL code for objects>
  <PDDL code for initial state>
  <PDDL code for goal specification>
)
```

- *<problem name>* is the string for this specific planning problem
- *<domain name>* must match your domain file!

# The blocks world problem file

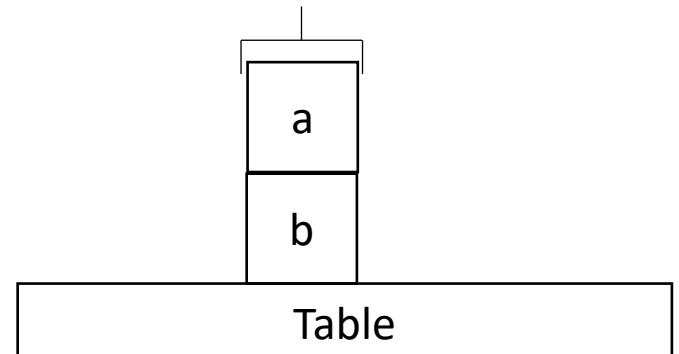
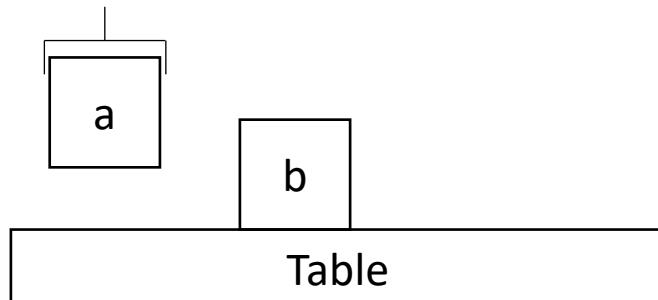
```
;; problem file: blocks-problem1.pddl
```

```
(define (problem blocksworld-prob1)
  (:domain blocksworld)
  (:objects a b)
  (:init (on-table a) (on-table b) (clear a) (clear b) (free))
  (:goal (and (on a b))))
```



# Sample output

```
command_prompt > planner blocks-domain-complete.pddl blocks-problem1.pddl
...
Parsed Domain file blocks-domain-complete.pddl successfully
Parsed Problem file blocks-problem1.pddl successfully
Performing search ...
...
Found a solution
(pickup a)
(stack a b)
Instantiation Time = 0.07sec
Planning Time = 0.03sec
```



# Closed-World Assumption

!!! IMPORTANT !!!

- PDDL adopts the **Closed-World Assumption (CWA)**:
  - Predicates not mentioned in
    - actions' precondition and
    - problems' init

are assumed to be **false**.
- Examples:
  - In pickup's precondition is assumed that `(not (holding ?ob))` holds.
  - In problem's init is assumed that `(not (on a b))`

!!! CWA does not hold for the goal !!!

# GRIPPER Example

Gripper task with four balls: There is a robot that can move between two rooms and pick up or drop balls with either of his two arms. Initially, all balls and the robot are in the first room. We want the balls to be in the second room.

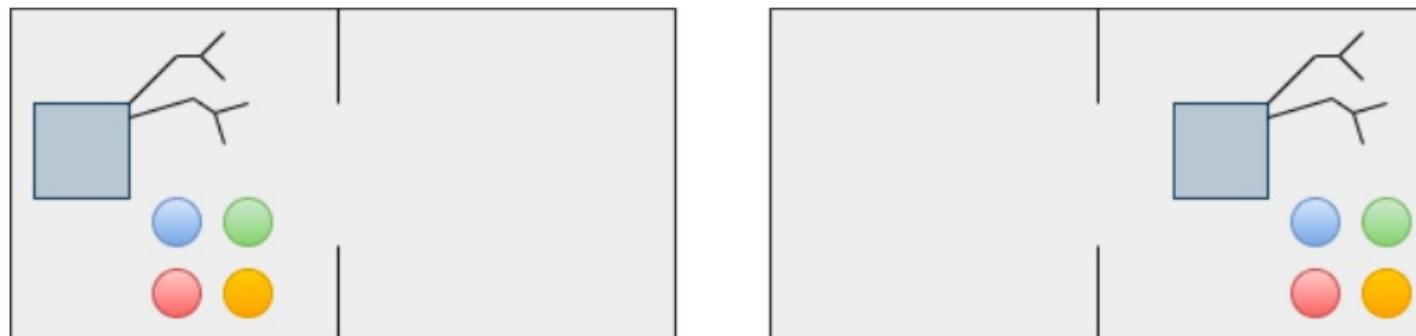
**OBJECTS:** The two rooms, four balls and two robot arms.

**PREDICATES:** Is  $x$  a room? Is  $x$  a ball? Is ball  $x$  inside room  $y$ ? Is robot arm  $x$  empty? [...]

**INITIAL STATE:** All balls and the robot are in the first room. All robot arms are empty. [...]

**GOAL STATES:** All balls must be in the second room.

**ACTIONS:** The robot can move between rooms, pick up a ball or drop a ball.



# Objects

- **Rooms:** *rooma, roomb*
- **Balls:** *ball1, ball2, ball3, ball4*
- **Robot arms:** *left, right*

```
(:objects rooma roomb  
          ball1 ball2 ball3 ball4  
          left right  
)
```

# Predicates

- ROOM(x)
- BALL(x)
- GRIPPER(x)
- at-roddy(x)
- at-ball(x,y)
- free(x), carry(x,y)

```
(:predicates (ROOM ?x) (BALL ?x)  
             (GRIPPER ?x)  
             (at-roddy ?x)  
             (at-ball ?x ?y)  
             (free ?x) (carry ?x ?y)))
```

# Initial state

## INITIAL STATE

ROOM(rooma) and ROOM(roomb) are true. BALL(ball1), ..., BALL(ball4) are true. GRIPPER(left), GRIPPER(right), free(left) and free(right) are true. at-roby(rooma) , at-ball(ball1, rooma), ..., at-ball(ball4, rooma) are true. Everything else is false.

## PDDL

```
(:init (ROOM rooma) (ROOM roomb)
       (BALL ball1) (BALL ball2) (BALL ball3) (BALL ball4)
       (GRIPPER left) (GRIPPER right) (free left) (free right)
       (at-roby rooma)
       (at-ball ball1 rooma) (at-ball ball2 rooma)
       (at-ball ball3 rooma) (at-ball ball4 rooma))
```

# Goal specification

## GOAL SPECIFICATION

at-ball(ball1, roomb), ..., at-ball(ball4, roomb) must be true.

## PDDL

```
:goal (and (at-ball ball1 roomb)
            (at-ball ball2 roomb)
            (at-ball ball3 roomb)
            (at-ball ball4 roomb)))
```

# Movement Action

## ACTION

**DESCRIPTION:** The robot can move from x to y.

**PRECONDITION:** ROOM(x), ROOM(y) and at-roddy(x) are true.

**EFFECT:** at-roddy(y) becomes true. at-roddy(x) becomes false. Everything else does not change.

## PDDL

```
(:action move :parameters (?x ?y)
  :precondition (and (ROOM ?x) (ROOM ?y)
                      (at-roddy ?x))
  :effect (and (at-roddy ?y)
                (not (at-roddy ?x))))
```

# Pick up action

## ACTION

**DESCRIPTION:** The robot can pick up x in y with z.

**PRECONDITION:** ROOM(x), ROOM(y) and at-roddy(x) are true. BALL(x), ROOM(y), GRIPPER(z), at-ball(x,y), at-roddy(y), and free(z) are true.

**EFFECT:** carry(z,x) becomes true. at-ball(x,y) and free(z) become false. Everything else does not change.

## PDDL

```
(:action pick-up :parameters (?x ?y ?z)
  :precondition (and (BALL ?x) (ROOM ?y) (GRIPPER ?z)
                      (at-ball ?x ?y) (at-roddy ?y) (free ?z))
  :effect (and (carry ?z ?x) (not (at-ball ?x ?y))
               (not (free ?z))))
```

# Drop action

## ACTION

**DESCRIPTION:** The robot can drop x in y from z

**PRECONDITION:** BALL(x), ROOM(y), GRIPPER(z), carry(z,x), at-roddy(y)

**EFFECT:** at-ball(x,y) and free(z) become true, carry(z,x) becomes false, everything else does not change

## PDDL

```
(:action drop :parameters (?x ?y ?z)
  :precondition (and (BALL ?x) (ROOM ?y) (GRIPPER ?z)
                      (carry ?z ?x) (at-roddy ?y))
  :effect (and (at-ball ?x ?y) (free ?z)
                (not (carry ?z ?x))))
```

# Example complete

```
(define (domain gripper)
  (:requirements :strips)
  (:predicates (ROOM ?x) (BALL ?x) (GRIPPER ?x) (at-roddy ?x) (at-ball ?x ?y)
               (free ?x) (carry ?x ?y) )
  (:action move
    :parameters (?x ?y)
    :precondition (and (ROOM ?x) (ROOM ?y) (at-roddy ?x))
    :effect (and (at-roddy ?y) (not (at-roddy ?x))) )
  (:action pick-up
    :parameters (?x ?y ?z)
    :precondition (and (BALL ?x) (ROOM ?y) (GRIPPER ?z) (at-ball ?x ?y)
                       (at-roddy ?y) (free ?z))
    :effect (and (carry ?z ?x) (not (at-ball ?x ?y)) (not (free ?z))) )
  (:action drop
    :parameters (?x ?y ?z)
    :precondition (and (BALL ?x) (ROOM ?y) (GRIPPER ?z) (carry ?z ?x) (at-roddy ?y))
    :effect (and (at-ball ?x ?y) (free ?z) (not (carry ?z ?x))) )
)
```

# Example complete

```
(define (problem gripper-prob)
  (:domain gripper)
  (:objects rooma roomb
            ball1 ball2 ball3 ball4
            left right)
  (:init (ROOM rooma) (ROOM roomb)
         (BALL ball1) (BALL ball2) (BALL ball3) (BALL ball4)
         (GRIPPER left) (GRIPPER right) (free left)
         (free right) (at-robbby rooma)
         (at-ball ball1 rooma) (at-ball ball2 rooma)
         (at-ball ball3 rooma) (at-ball ball4 rooma))
  (:goal (and (at-ball ball1 roomb)
              (at-ball ball2 roomb)
              (at-ball ball3 roomb)
              (at-ball ball4 roomb)))
  )
```

# Extras ...1: Typing

```
(define (domain gripper-typed)
  (:requirements :typing)
  (:types room ball gripper)
  (:constants left right - gripper)
  (:predicates (at-roddy ?r - room)
               (at ?b - ball ?r - room)
               (free ?g - gripper)
               (carry ?o - ball ?g - gripper))
  (:action move :parameters (?from ?to - room)
    :precondition (at-roddy ?from)
    :effect (and (at-roddy ?to)
                 (not (at-roddy ?from)))))
```

# Extras ... 2: Type Hierarchy

```
(define (domain logistics-adl)
  (:requirements :adl :domain-axioms)
  (:types physobj - object
         obj vehicle - physobj
         truck airplane - vehicle
         location city - object
         airport - location)
  (:predicates (at ?x - physobj ?l - location)
               (in ?x - obj ?t - vehicle)
               (in-city ?l - location ?c - city)))
```

# Extras...3: Conditional Effects

```
(:action drive-truck
  :parameters (?truck - truck ?loc-from ?loc-to - location ?city - city)
  :precondition (and (at ?truck ?loc-from)
                      (in-city ?loc-from ?city)
                      (in-city ?loc-to ?city))
  :effect (and (at ?truck ?loc-to)
                (not (at ?truck ?loc-from))
                (forall (?x - obj)
                        (when (and (in ?x ?truck))
                            (and (not (at ?x ?loc-from))
                                (at ?x ?loc-to)))))))
```

# Next

- Hands-on Session!
- We will create and solve planning problems
- Download Blackbox planner  
<http://www.cs.rochester.edu/u/kautz/satplan/blackbox/>
- Alternatively use an online planner:  
<http://planning.domains/>  
<http://paas-uom.org/>

# Interaction and Communication

Autonomous Software Agents

A.A. 2024-2025

**Prof. Paolo Giorgini**



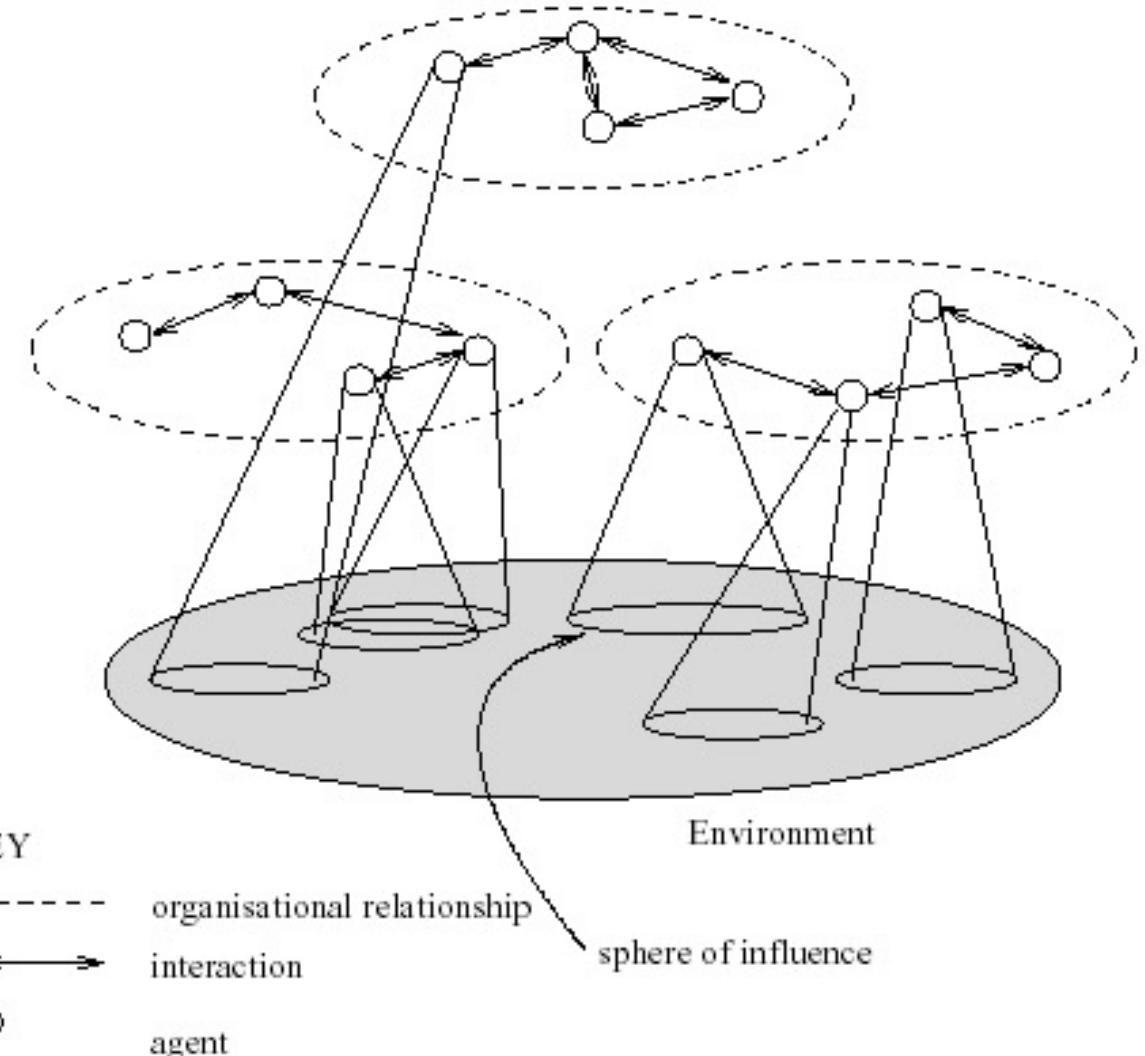
UNIVERSITY OF TRENTO - Italy

Department of Information  
and Communication Technology

# Agents interact one another

A multiagent system contains a number of agents

- which may interact through communication
- are able to act in an environment
- have different “spheres of influence” (which may coincide)
- will be linked by other (organizational) relationships`

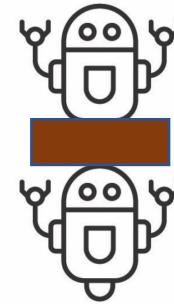
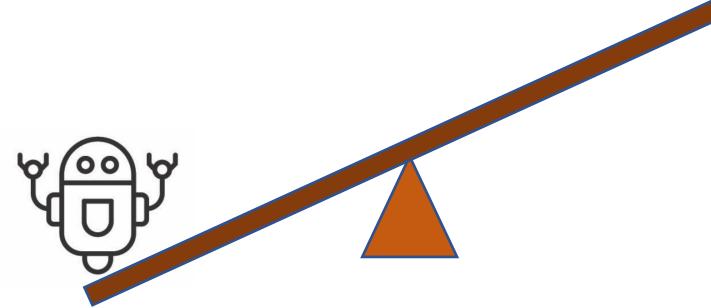


# Agents' interaction



## Physical interaction

- Doing an action in the environment the agent may affect the status of another agent
- An agent can act physically on another agent

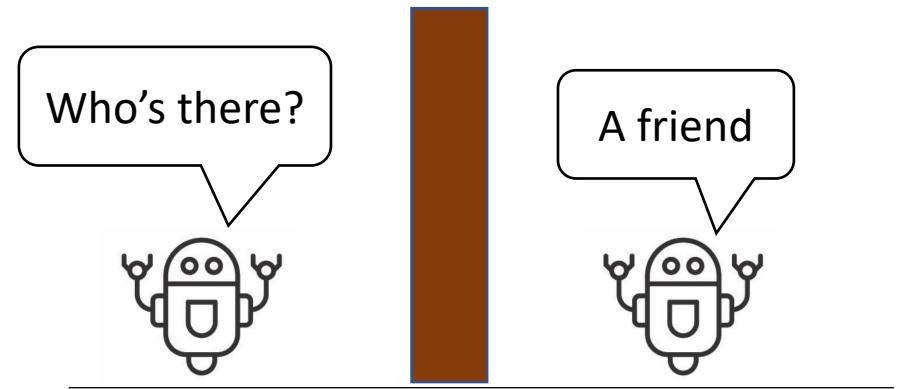


---

# Agents' interaction

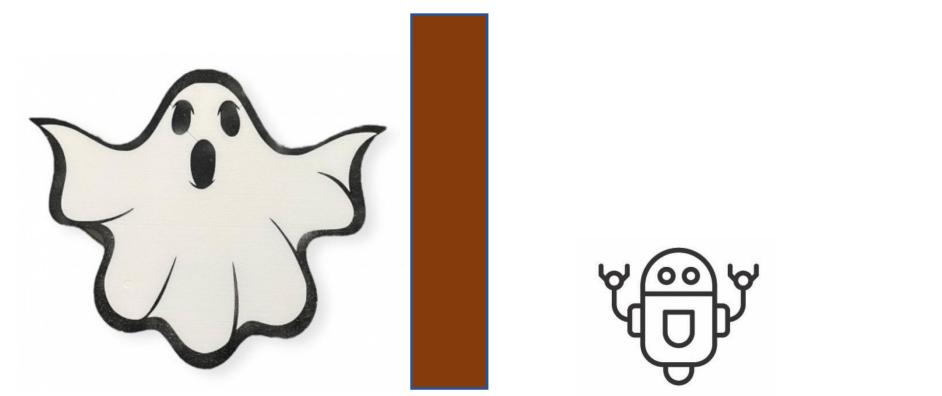
## Communicative Interaction

- Agents interact through communication
- A common communication language



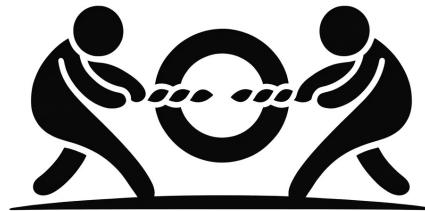
## Other form of interaction

- interactions that influence or modify the behavior of others through actions



# Why should agents interact ?

- They are part of the same environment
- They compete
  - E.g., for the use of resources
  - **Self-interested agents**
- They cooperate
  - E.g., being part of the same system (common overall goal)
    - With different competences and covering different roles
  - **Benevolent agents**



# Benevolent Agents

- If we “own” the whole system, we can design agents to help each other whenever asked – eg. team of cooperative agents
  - agents are *benevolent*: our best interest is their best interest
  - Benevolence simplifies the system design task enormously!
- Problem-solving in benevolent systems is **Cooperative Distributed Problem Solving** (CDPS)
  - subfield of AI focused on how multiple agents or systems can work together to solve complex problems that are **too large or too complex** for a single agent to handle alone
  - agents cooperate by **dividing the problem into smaller parts**, each of which can be solved by different agents with different competences and/or covering different roles

# Cooperative Distributed Problem Solving

- **Decomposition:** the problem is broken down into manageable sub-problems that can be distributed among various agents
- **Coordination:** Agents must coordinate their actions to ensure that their individual solutions align with each other and contribute effectively to solving the overall problem
- **Communication:** Effective communication is crucial in CDPS as it allows agents to share information, results, and updates about their progress. This includes not only data but also plans, goals, and resource availability
- **Integration:** Once individual solutions are developed, they need to be integrated into a final solution that solves the original problem

# Task Sharing and Result Sharing

- Two main modes of cooperative problem solving:
  - **Task sharing**  
components of a task are distributed to component agents
  - **Result sharing**  
information (partial results, etc.) is distributed

# Multi-agent planning

The goal state is reached combining actions of a team of agents

- Elaborating an overall plan
  - Putting together the capabilities (actions) of the agents
- Coordinating agents' activities
- Centralized or distributed point of control
  - A **master agent** responsible to elaborate the plan, dispatch activities to the team, control the flow of activities and monitoring their correct execution
  - The team **negotiate** the plan and coordinate activities one another
- Use of **interaction protocols**

# The Contract Net

- A well-known task-sharing protocol for task allocation is **the contract net**
  - Recognition
  - Announcement
  - Bidding
  - Awarding
  - Execution

# Recognition

- In this stage, an agent recognizes it has a problem it wants help with
- Agent has a goal, and either...
  - realizes **it cannot achieve the goal** in isolation — does not have capability
  - realizes it would **prefer not to achieve the goal** in isolation (typically because of solution quality, deadline, etc.)

# Announcement

- In this stage, the agent with the task sends out an **announcement** of the task which includes a **specification** of the task to be achieved
- Specification must encode:
  - description of task itself (maybe executable)
  - any constraints (e.g., deadlines, quality constraints)
  - meta-task information (e.g., “bids must be submitted by...”)
- The announcement is then **broadcast**

# Bidding

- Agents that receive the announcement decide for themselves whether they wish to **bid** for the task
- Factors:
  - agent must decide whether it is capable of performing the task
  - agent must determine quality constraints & price information (if relevant)
- If they do choose to bid, then they submit a **tender**

# Awarding & Execution

- Agent that sent task announcement must choose between bids & decide who to “award the contract” to
- The result of this process is communicated to agents that submitted a bid
- The successful **contractor** then execute the task
- May involve generating further manager-contractor relationships (e.g., **sub-contracting**)

# Efficiency Modifications

- **Focused addressing** — when general broadcast isn't required
- **Directed contracts** — when manager already knows which node is appropriate
- **Request-response mechanism** — for simple transfer of information without overhead of contracting
- **Node-available message** — reverses initiative of negotiation process

# Example: packages collection

## Announcement

To: \*

From: RED

Type: Task Announcement

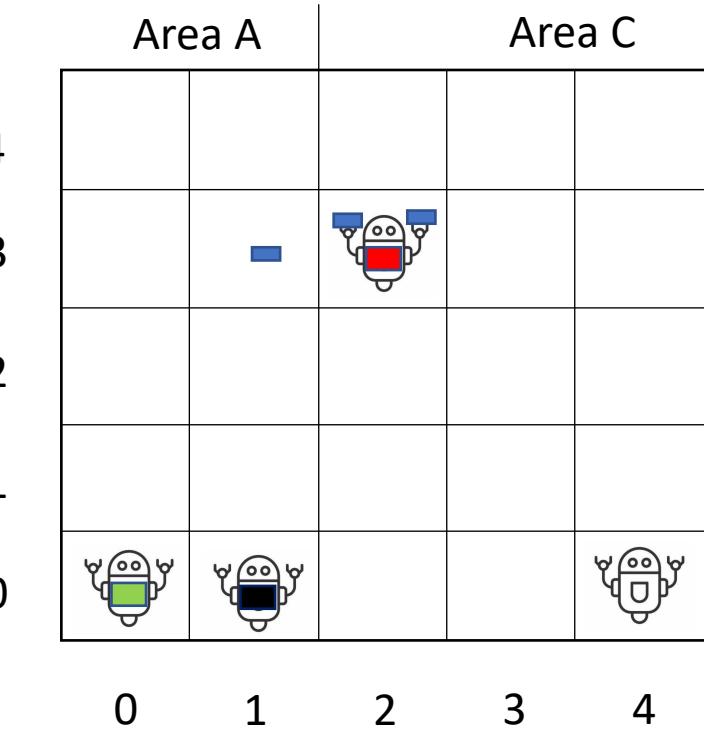
Contract: 07-05-25

Eligibility Specification:  
Must-Have GRIPPER  
Must-be-in Area A

Task Abstraction:  
Task-Type COLLECT  
Position (1,3)  
Area A Specification ((0,0),(1,4))

Bid Specification:  
Position (X,Y)  
Agent Name

Expiration Time:  
07 17:30 MAY 2025



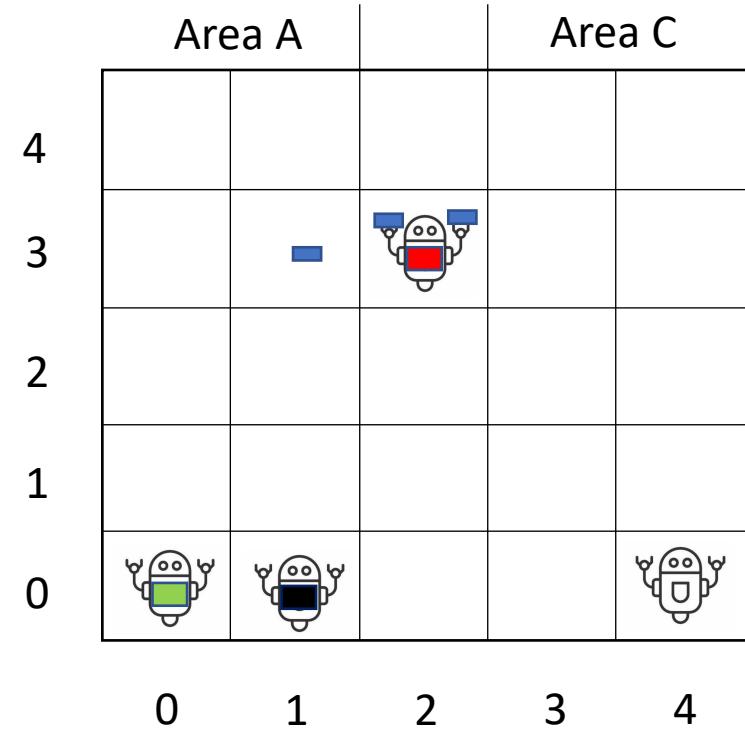
# Example: packages collection

## Bid

To: RED  
From: GREEN  
Type: BID  
Contract: 07-05-25  
Bid Specification:  
Position (0,0)  
Agent GREEN

## Award

To: GREEN  
From: RED  
Type: AWARD  
Contract: 07-05-25  
Task Specification:  
Task-Type COLLECT  
Position (1,3)



# BDI loop for the contract-net 1

## Recognition – Announcement - Awarding

**Intention Rules:** if (prec) then Int

```
if (in(Pck,X,Y) ^ free_gripper(G)) {0 += collect(Pck,X,Y)}  
if (in(Pck,X,Y) ^ not free_gripper(G)) {0 += ask_collect(Pck,X,Y)}  
→ I={ask_collect(Pck,X,Y)} # options → intention
```

**Plans selection or Planning:** P = planning (B,I,P)

```
→ P = {p1} # plan for I  
→ p1={announcement(collect(pack_1)), bids_eval(collect(pack_1)),  
awarding(collect(pack_1))}
```

## Announcement

To: \*  
From: RED  
Type: Task Announcement  
Contract: 23-05-02  
Eligibility Specification:  
Must-Have GRIPPER  
Must-be-in Area A  
Task Abstraction:  
Task-Type COLLECT  
Position (1,3)  
Area A Specification ((0,0),(1,4))  
Bid Specification:  
Position (X,Y)  
Agent Name  
Expiration Time:  
02 1730 MAY 2023

## Award

To: GREEN  
From: RED  
Type: AWARD  
Contract: 23-05-02  
Task Specification:  
Task-Type COLLECT  
Position (1,3)

# BDI loop for the contract-net 2

## Bidding - Execution

### Intention Rules:

```
if (task_ann(Pck,X,Y) ^ free_gripper(G)) {0 += bid(Pck,X,Y)}  
  
if (award(Pck,XY)) {0 += collect(Pck,X,Y)}  
  
→ I={bid(Pck,X,Y)} # options -> intention
```

### Bid

To:	RED
From:	GREEN
Type:	BID
Contract:	23-05-02
Bid Specification:	Position (0,0)
	Agent GREEN

### Plans selection or Planning: P = planning (B,I,P)

```
→ P = {p1} # plan for bid(Pack_1,X,Y)  
  
→ p1={bidding(collect(pack_1))}  
  
→ p2={move(UP), move(UP), move(UP), PICK_UP(pack_1)}) # plan for collect(Pack_1,X,Y)
```

# Self-Interested Agents

- If agents represent individuals or organizations, (the more general case), then we cannot make the benevolence assumption
- Agents will be assumed to act to further their own interests, possibly at expense of others
- Potential for **conflict**
- May complicate the design task enormously

# Utilities and Preferences

- Assume we have just two agents:  $Ag = \{i, j\}$
- Agents are assumed to be **self-interested**:
  - they **have preferences over how the environment is**
- Assume  $\Omega = \{\omega_1, \omega_2, \dots\}$  is the set of “outcomes” that agents have preferences over
- We capture preferences by **utility functions**
$$u_i : \Omega \rightarrow \mathbb{R}$$
$$u_j : \Omega \rightarrow \mathbb{R}$$

- Utility functions lead to **preference orderings** over outcomes:

$\omega >_i \omega'$  means  $u_i(\omega) > u_i(\omega')$

$\omega \geq_i \omega'$  means  $u_i(\omega) \geq u_i(\omega')$

# Utility-based agent

- A measure of how desirable a particular state (goal) is.
- *Utility function* maps a state to a measure of the utility of the state
  - Can describe how "happy" the agent is
- A rational utility-based agent chooses the action that maximizes the expected utility of the action outcomes
  - i.e., the agent expects to derive, on average, given the probabilities and utilities of each outcome.

# Multiagent Encounters

- We need a model of the environment in which these agents will act:
  - agents simultaneously choose an action to perform, and as a result of the actions they select, an outcome in  $\Omega$  will result
  - the **actual** outcome depends on the **combination** of actions
  - assume each agent has just two possible actions that it can perform,  $C$  (“Cooperate”) and  $D$  (“Defect”)
- Environment behavior given by *state transformer function*:

$$\tau : \begin{matrix} \mathcal{A}_C \\ \text{agent } i\text{'s action} \end{matrix} \times \begin{matrix} \mathcal{A}_C \\ \text{agent } j\text{'s action} \end{matrix} \rightarrow \Omega$$

# Multiagent Encounters

- Here is a state transformer function:

$$\tau(D, D) = \omega_1 \quad \tau(D, C) = \omega_2 \quad \tau(C, D) = \omega_3 \quad \tau(C, C) = \omega_4$$

(This environment is sensitive to actions of both agents.)

- Here is another:

$$\tau(D, D) = \omega_1 \quad \tau(D, C) = \omega_1 \quad \tau(C, D) = \omega_1 \quad \tau(C, C) = \omega_1$$

(Neither agent has any influence in this environment.)

- And here is another:

$$\tau(D, D) = \omega_1 \quad \tau(D, C) = \omega_2 \quad \tau(C, D) = \omega_1 \quad \tau(C, C) = \omega_2$$

(This environment is controlled by  $j$ .)

# Rational Action

- Suppose we have the case where **both** agents can influence the outcome, and they have utility functions as follows:

$$\begin{array}{llll} u_i(\omega_1) = 1 & u_i(\omega_2) = 1 & u_i(\omega_3) = 4 & u_i(\omega_4) = 4 \\ u_j(\omega_1) = 1 & u_j(\omega_2) = 4 & u_j(\omega_3) = 1 & u_j(\omega_4) = 4 \end{array}$$

- With a bit of abuse of notation:

$$\begin{array}{llll} u_i(D, D) = 1 & u_i(D, C) = 1 & u_i(C, D) = 4 & u_i(C, C) = 4 \\ u_j(D, D) = 1 & u_j(D, C) = 4 & u_j(C, D) = 1 & u_j(C, C) = 4 \end{array}$$

- Then agent  $i$ 's preferences are:

$$C, C \succeq_i C, D \quad \succ_i \quad D, C \succeq_i D, D$$

- “ $C$ ” is the **rational choice** for  $i$ .  
(Because  $i$  prefers all outcomes that arise through  $C$  over all outcomes that arise through  $D$ .)  
what about  $j$ ?

# Payoff Matrices

- We can characterize the previous scenario in *a payoff matrix*

		<i>i</i>	
		defect	coop
<i>j</i>	defect	1	4
	coop	1	4
		4	4

$$\begin{array}{llll} u_i(D, D) = 1 & u_i(D, C) = 1 & u_i(C, D) = 4 & u_i(C, C) = 4 \\ u_j(D, D) = 1 & u_j(D, C) = 4 & u_j(C, D) = 1 & u_j(C, C) = 4 \end{array}$$

- Agent *i* is the **column player**
- Agent *j* is the **row player**

# Dominant Strategies

- Given any particular strategy  $S$  (either  $C$  or  $D$ ) of agent  $i$ , there will be a number of possible outcomes
- We say  $s_1$  *dominates*  $s_2$  if every outcome possible by  $i$  playing  $s_1$  is preferred over every outcome possible by  $i$  playing  $s_2$
- A rational agent will never play a dominated strategy
- So, in deciding what to do, we can **delete dominated strategies**
- Unfortunately, there isn't always a unique undominated strategy

# Nash Equilibrium

- In general, we will say that two strategies  $s_1$  and  $s_2$  are in Nash equilibrium if:
  - under the assumption that agent  $i$  plays  $s_1$ , agent  $j$  can do no better than play  $s_2$ ; and
  - under the assumption that agent  $j$  plays  $s_2$ , agent  $i$  can do no better than play  $s_1$ .
- Neither agent has any incentive to deviate from a Nash equilibrium
- Unfortunately:
  - *Not every interaction scenario has a Nash equilibrium*
  - *Some interaction scenarios have more than one Nash equilibrium*

# Competitive and Zero-Sum Interactions

- Where preferences of agents are diametrically opposed, we have **strictly competitive** scenarios
- Zero-sum encounters are those where utilities sum to zero:  
$$u_i(\omega) + u_j(\omega) = 0 \quad \text{for all } \omega \in \Omega$$
- Zero sum implies strictly competitive
  - A zero-sum interaction is when one party's gain equals the other party's loss — the sum of their gains and losses is zero.
- Zero sum encounters in real life are very rare ... but people tend to act in many scenarios as if they were zero sum

# The Prisoner's Dilemma

- Two men are collectively charged with a crime and held in separate cells, with no way of meeting or communicating. They are told that:
  - if one confesses and the other does not (C,D), the confessor will be freed, and the other will be jailed for three years
  - if both confess (C,C), then each will be jailed for two years
  - Both prisoners know that if neither confesses (D,D), then they will each be jailed for 1 year

# The Prisoner's Dilemma

Years of prison	Utility
0	4
1	3
2	2
3	1

		i
j	Not Confess	Confess
Not Confess	3 (1 year)	4 (freed)
Confess	1 (3 years)	2 (2 years)

# The Prisoner's Dilemma

- Best strategy is (C,C)
  - Confess ---> from 0 to 2 years
  - Not confess ---> from 1 to 3 years
- The strategy “Not confess” is dominated by “confess”
- Eliminating the dominated strategies, we have that both prisoners confess (and both get 2 years)
- However, the best for each of them is to not confess (1 year)
  - Suppose they both agree before being arrested to not confess
    - If now one of them does not respect the agreement he will be freed
    - So, what to do: confess or not confess?

# The Prisoner's Dilemma

- This apparent paradox is **the fundamental problem of multi-agent interactions**
  - cooperation will not occur in societies of self-interested agents
- Real world examples:
  - nuclear arms reduction ("why don't I keep mine. . .")
- The prisoner's dilemma is **ubiquitous**

# Reasoning about agents' mental state

- For a better interaction agents' may have a local representation of other agents' mental state
  - Can be used to reason about other agents' beliefs and intentions
  - Useful during a negotiation
  - Very challenging problem (e.g., keeping the state updated)
  - Using your own BDI model can be a limitation
    - E.g., Are you sure the other agents' has your intentions rules?
  - The mental state representation can be updated by observation and/or communication

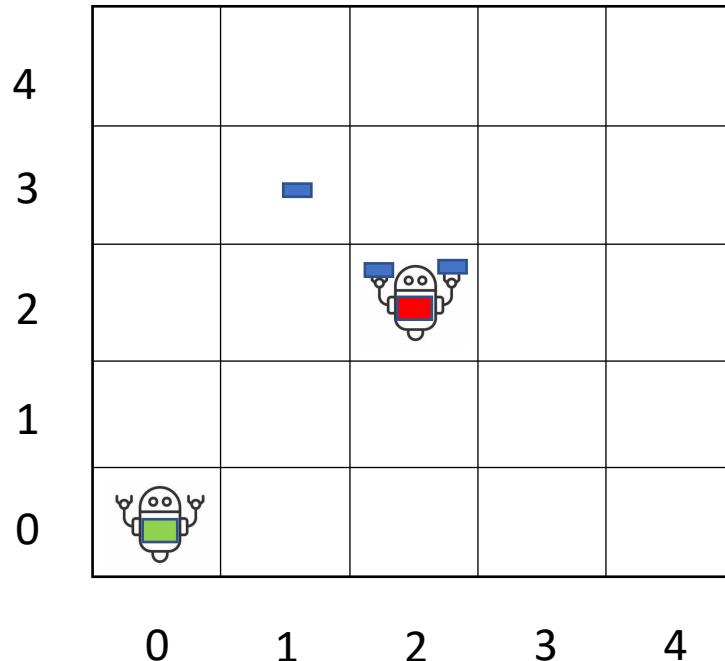
# Example – by observation 1

RED about GREEN

```
-- t=0  
B={}  
I={}  
-- t=2  
B={in(pack_1,1,3)}  
I={pick_up(pack_1)}
```

how to ?

GREEN moves towards the packages, so it has the intention to pick\_up the package and, of course, it believes the package is there



# Example – by observation 1

Observing the execution of P (or part of it),  $\{\text{move(up)}, \text{move(up)}\} \subseteq P$  and from if  $\text{pick\_up(pack)} \rightarrow P$ :

the agent can abduce the goal/intention  $\text{pick\_up(pack)}$

and from if  $\text{in(pack, X, Y)} \rightarrow \text{pick\_up(pack)}$  (intention rule)  
the agent can abduce the belief  $\text{in(pack, X, Y)}$

This is a possible explanation of the agent behaviour

the vice versa is also valid : from beliefs to intentions

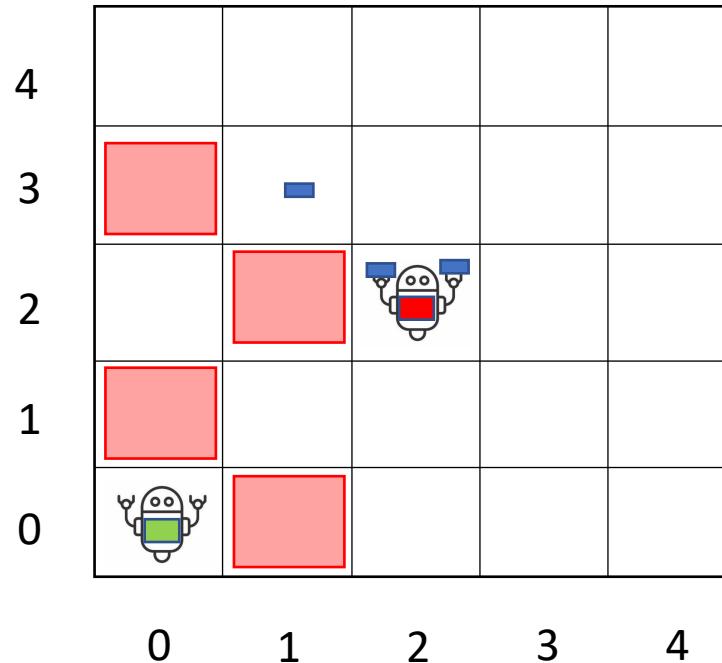
# Example – by observation 2

RED about GREEN

```
-- t=0  
B={}  
I={}  
-- t=2  
B={}  
I={look_for_pack()}
```

how to ?

GREEN moves towards the packages, but since it cannot see pack\_1, it cannot have the intention to pick it up, it is wandering around to find a package



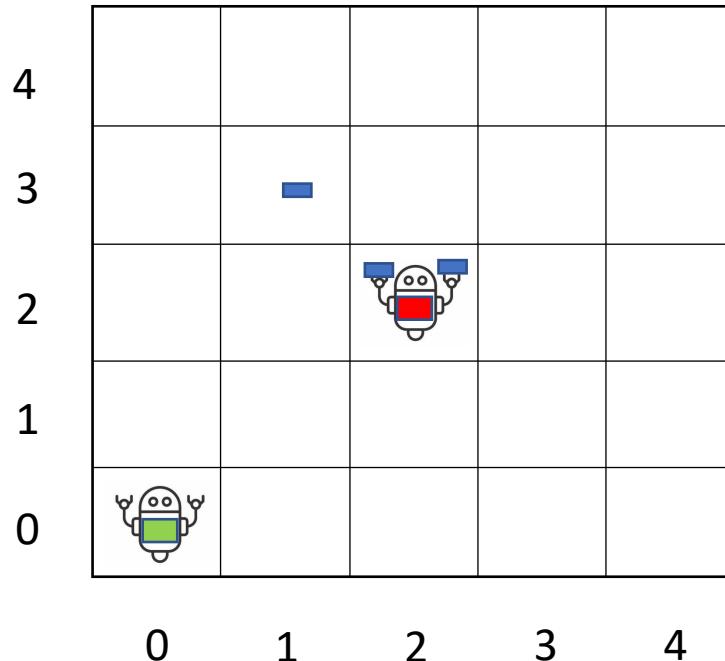
# Example – by observation 3

GREEN about RED

```
-- t=0  
B={}  
I={}  
-- t=1  
B={in(pack_1,1,3)}  
I={}
```

how to ?

RED can see pack\_1, but since it has no free gripper, RED has no intention to pick up the package



# Example – by communication 1

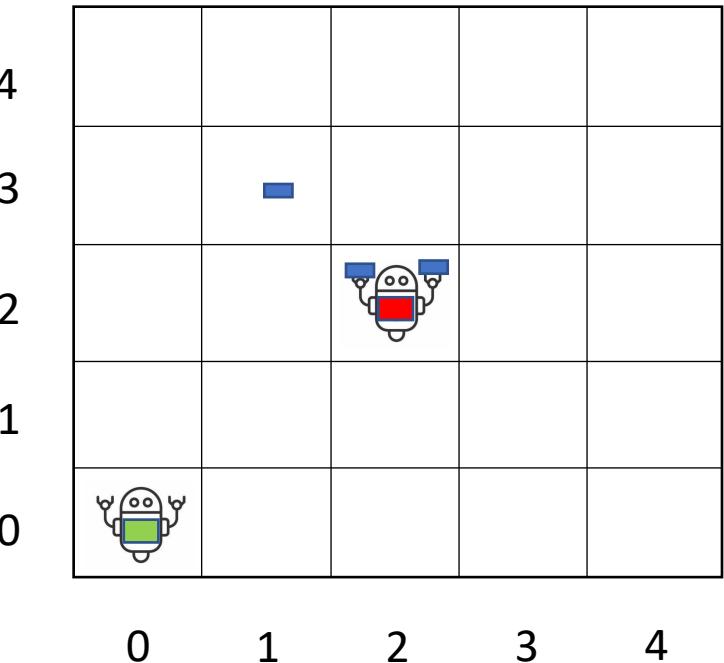
RED about GREEN

```
-- t=0  
B={}  
I={}  
-- t=1 #RED inform GREEN about in(pack_1,1,3)  
B={in(pack_1,1,3)}  
I={pick_up(pack_1)}
```

how to ?

Now GREEN believes pack\_1 is there, so it intends to pick up the package

if in(pack,X,Y) -> pick\_up(pack)



# Note

- It could be the intention of RED to make GREEN believe that `in(pack_1,1,3)`, so GREEN will intend to `pick_up(pack_1)` and then make GREEN to move towards the package
- Communication can be used to change the mental state of the hearer to induce a specific behaviour (injection of an intention)

# Example – by communication 2

RED about GREEN

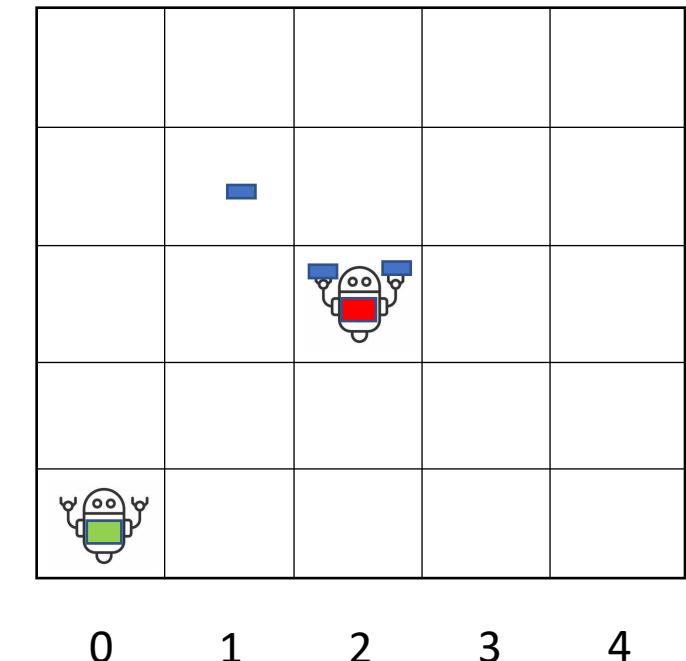
```
-- t=0
B={}
I={}
-- t=1 #RED inform GREEN about pick_up(pack_1)
B={in(pack_1,1,3)}
I={}
```

how to ?

Now GREEN believes RED intends to pick up pack\_1, so there should be pack\_1 and it does not make any sense that GREEN intends to pick up pack\_1

```
if in(pack,X,Y) -> pick_up(pack) # in(pack_1,1,3) for abduction for RED
```

```
if in(pack,X,Y) ^ not pick_up(A,pack) -> pick_up(pack)
```



# Note

Also in this case, it could be a strategy of RED to make GREEN believe that RED intend to pick up pack\_1 , so GREEN will not go to pick up the package

Communication is used as an action of a Plan

- Dissuade GREEN to pick up pack\_1
- Delivery Packs
- Pick up pack\_1

# Agent Communication

- Macro-aspects of intelligent agent technology: those issues relating to the agent **society**, rather than the individual:
  - *communication*  
speech acts; KQML & KIF; FIPA ACL

# Speech Acts

- Most treatments of communication in (multi-) agent systems borrow their inspiration from **speech act theory**
- Speech act theories are **pragmatic** theories of language,
  - i.e., theories of language use: they attempt to account for how language is used by people every day **to achieve their goals and intentions**
- The origin of speech act theories are usually traced to Austin's 1962 book, **How to Do Things with Words**

# Speech Acts

- Austin noticed that some utterances are rather like “physical” actions that appear to **change the state of the world**
- Paradigm examples would be:
  - Declaring war
  - Christening
  - ‘I now pronounce you man and wife’ :-)
- But more generally, **everything** we utter is **uttered with the intention** of satisfying some goal or intention
- A theory of how utterances are used to achieve intentions is a speech act theory

# Different Aspects of Speech Acts

- From “A Dictionary of Philosophical Terms and Names”
- **Locutionary act** the simple speech act of generating sounds that are linked together by grammatical conventions so as to say something meaningful
  - Among English speakers, for example, ‘It is raining’ performs the locutionary act of saying that it is raining, as ‘Grabilistrod zetagflx dapu’ would not”

# Different Aspects of Speech Acts

- **Illocutionary act** the speech act of doing something else
  - offering advice, for example – in the process of uttering meaningful language.
    - Thus, for example, in saying ‘I will repay you this money next week’ one typically performs the illocutionary act of making a promise

# Different Aspects of Speech Acts

- **Perlocutionary act** the speech act of having an effect on those who hear a meaningful utterance
- By telling a ghost story late at night, for example, one may accomplish the cruel perlocutionary act of frightening a child

# Speech Acts

- Searle ('69) identified different types of speech act:
  - **representatives**  
such as *informing*, e.g., 'It is raining'
  - **directives**  
attempts to get the hearer to do something e.g., 'please make the tea'
  - **commisives**  
which commit the speaker to doing something, e.g., 'I promise to...'
  - **expressives**  
whereby a speaker expresses a mental state, e.g., 'thank you!'
  - **declarations**  
such as declaring war or christening

# Speech Acts

- There is some debate about whether this (or any!) typology of speech acts is appropriate
- In general, a speech act can be seen to have two components:
  - a **performative verb**  
(e.g., request, inform, promise, ... )
  - **propositional content**  
(e.g., “the door is closed”)

# Speech Acts

## Consider

performative = **request**

content = “the door is closed”

**speech act** = “please close the door”

performative = **inform**

content = “the door is closed”

**speech act** = “the door is closed!”

performative = **inquire**

content = “the door is closed”

**speech act** = “is the door closed?”

# Plan Based Semantics

- How does one define the semantics of speech acts? When can one say someone has uttered, e.g., a request or an inform?
- Cohen & Perrault (1979) defined semantics of speech acts using the **precondition-delete-add** list formalism of planning research
- Note that a speaker cannot (generally) **force** a hearer to accept some desired mental state
- In other words, there is a separation between the **illocutionary act** and the **perlocutionary act**

# What about inform?

We could adopt that: **inform(s, h,  $\phi$ )**

Pre:

$s$  believes  $\phi$  is true

$s$  intends that  $h$  believes  $f$

Post:

$s$  believes that  $h$  believes  $\phi$

$h$  believes  $\phi$

Other semantics of course can be adopted

# Plan-Based Semantics

- Here is their semantics for **request(s, h, φ)**

pre:

- $s$  believes  $h$  can do  $\phi$   
(you don't ask someone to do something unless you think they can do it)
- $s$  believes  $h$  believe  $h$  can do  $\phi$   
(you don't ask someone unless *they* believe they can do it)
- $s$  believes  $s$  wants  $\phi$   
(you don't ask someone unless you want it!)

post:

- $h$  believes  $s$  believes  $s$  wants  $\phi$   
(the effect is to make them aware of your desire)

# What about inquire?

**Inquire(s,h,φ)**

pre:

- ....

post:

- ....

# Benevolent / Collaborative agents

Inform(RED, GREEN, in(pack\_1, 1, 3))



By Belief Revision GREEN will

$B = \{ \text{in}(\text{pack\_1}, 1, 3) \}$

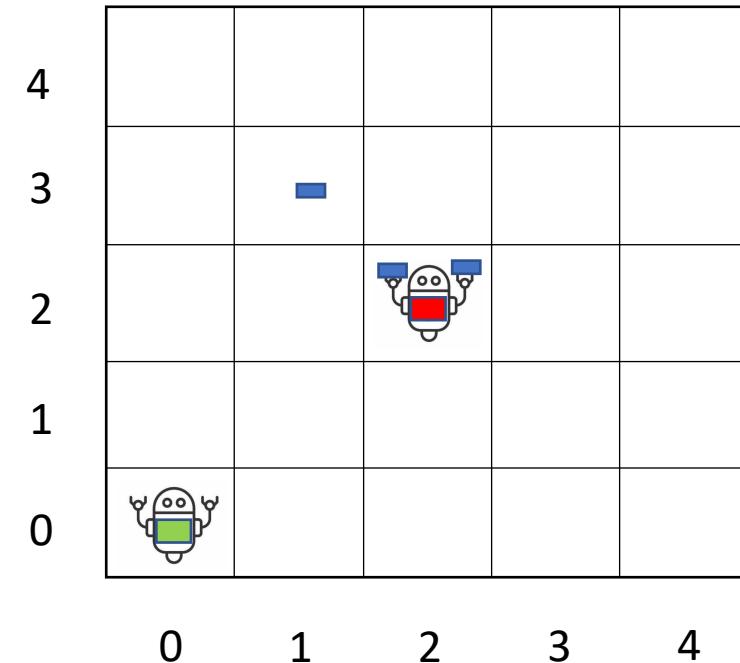
Why did RED inform GREEN ?

By design RED has the following intention rule

if  $\text{in}(\text{pack}, X, Y) \rightarrow \text{inform}(\text{RED}, \text{GREEN}, \text{in}(\text{pack}, X, Y))$

or

if  $\text{in}(\text{pack}, X, Y) \wedge \text{not free(GRIP)} \rightarrow \text{inform}(\text{RED}, \text{GREEN}, \text{in}(\text{pack}, X, Y))$



# Benevolent / Collaborative agents

Request(RED, GREEN, pick\_up(pack\_1, 1, 3))



By intention revision GREEN will

I={pick\_up(pack\_1, 1, 3)}

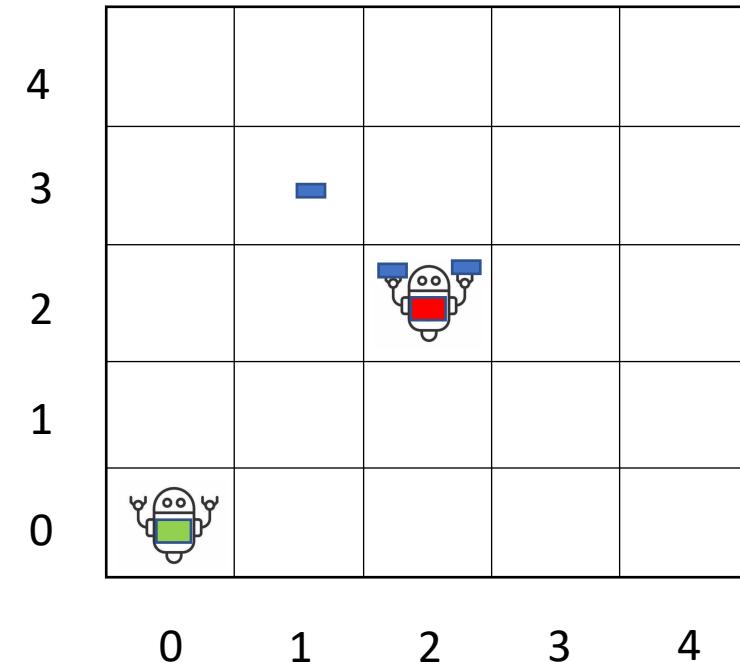
Why did RED request GREEN ?

By design RED has the following intention rule

if in(pack, X, Y) -> request(RED, GREEN, pick\_up(pack, X, Y))

or

if in(pack, X, Y) ^ not free(GRIP) -> request(RED, GREEN, pick\_up(pack, X, Y))



# KQML and KIF

- We now consider *agent communication languages* (ACLs) – standard formats for the exchange of messages
- The best known ACL is KQML, developed by the ARPA knowledge sharing initiative  
KQML is comprised of two parts:
  - the Knowledge Query and Manipulation Language (KQML)
  - the Knowledge Interchange Format (KIF)

# KQML and KIF

- KQML is a language, that defines various acceptable ‘communicative verbs’, or **performatives**

Example performatives:

- ask-if (‘is it true that...’)
  - perform (‘please perform the following action...’)
  - tell (‘it is true that...’)
  - reply (‘the answer is...’)
- KIF is a language for expressing message **content**

# KQML performatives

Category	Reserved performative names
Basic informational performatives	tell, deny, untell, cancel,
Basic query performatives	evaluate, reply, ask-if, ask-about, ask-one, ask-all, sorry
Multi-response query performatives	stream-about, stream-all
Basic effector performatives	achieve, □nachieved
Generator performatives	standby, ready, next, rest, discard, generator
Capability definition performatives	advertise
Notification performatives	subscribe, monitor
Networking performatives	register, unregister, forward, broadcast, pipe, break
Facilitation performatives	broker-one, broker-all, recommend-one, recommend-all, recruit-one, recruit-all

# KIF – Knowledge Interchange Format

Used to state:

- Properties of things in a domain (e.g., “Marco is chairman”)
- Relationships between things in a domain (e.g., “Marco and Paolo are friends”)
- General properties of a domain (e.g., “All students are registered for at least one course”)

# KIF – Knowledge Interchange Format

- “The temperature of room A110 is 21 Celsius”:

```
(= (temperature A110) (scalar 21 Celsius))
```

- “An object is a bachelor if the object is a man and is not married”:

```
(defrelation bachelor (?x) :=  
  (and (man ?x) (not (married ?x))))
```

- “Any individual with the property of being a person also has the property of being a mammal”:

```
(defrelation person (?x) :> (mammal ?x))
```

# KQML and KIF

- In order to be able to communicate, agents must have agreed on a common set of terms
- A formal specification of a set of terms is known as an **ontology**
- The knowledge sharing effort has associated with it a large effort at defining common ontologies — software tools like ontolingua for this purpose
- Example KQML/KIF dialogue...

```
A to B: (ask-if (> (timer pack_1) (timer pack_2)) )  
B to A: (reply true)  
B to A: (inform (= (timer pack_1) 20))  
B to A: (inform (= (timer pack_2) 10))
```

# KQML examples

A query about the price of a share of IBM stock

```
(ask-one
  :content  (TIMER PACK_1 ?timer)
  :receiver Agent_RED
  :language Deliveroo_L
  :ontology Deliveroo_O)

(ask-all
  :content  'timer(PACK_1, [?timer, ?time])'
  :receiver Agent_RED
  :language Deliveroo_LS
  :ontology Deliveroo_OS)
```

# FIPA

- The Foundation for Intelligent Physical Agents (IEEE- FIPA standard) —> the FIPA ACL
- Basic structure is quite similar to KQML:
  - **performative**  
20 performatives in FIPA
  - **housekeeping**  
e.g., sender, etc.
  - **content**  
the actual content of the message

# FIPA - example

```
(inform
  :sender          agent1
  :receiver        agent5
  :content         (price good200 150)
  :language        sl
  :ontology        hpl-auction
)
```

# FIPA-ACL performative set

Accept proposal	Inform If	Refuse
Agree	Inform Ref	Reject Proposal
Cancel	Not Understood	Request
Call for Proposal	Propagate	Request When
Confirm	Propose	Request Whenever
Disconfirm	Proxy	Subscribe
Failure	Query If	
Inform	Query Ref	

# FIPA

performative	passing info	requesting info	negotiation	performing actions	error handling
accept-proposal			x		
agree				x	
cancel		x		x	
cfp			x		
confirm	x				
disconfirm	x				
failure					x
inform	x				
inform-if	x				
inform-ref	x				
not-understood					x
propose			x		
query-if		x			
query-ref		x			
refuse				x	
reject-proposal			x		
request				x	
request-when				x	
request-whenever				x	
subscribe		x			

## FIPA-ACL message elements

Element	Description
Performative	Denotes the type of the communicative act of the ACL message
Sender	Denotes the identity of the sender (the name of the agent) of the message
Receiver	Denotes the identity of the intended recipient of the message
Reply-To	This element indicates that subsequent messages in this conversation thread are to be directed to the agent named in the reply-to-element, instead of the agent named in the sender element
Content	Denotes the content of the message
Language	Denotes the language in which the content element is expressed
Encoding	Denotes the specific encoding of the content language expression
Ontology	Denotes the ontology(s) used to give a meaning to the symbols in the content expression
Protocol	Denotes the interaction protocol that the sending agent is employing with this ACL message
Conversation-id	Introduces an expression (a conversation identifier) which is used to identify the ongoing sequence of communicative acts that together form a conversation
Reply-with	Introduces an expression that will be used by the responding agent to identify this message
In-reply-to	Denotes an expression that references an earlier action to which this message is a reply
Reply-By	Denotes a time and/or date expression which indicates the latest time by which the sending agent would like to have received a reply

# An example of FIPA-ACL

```
(cfp
  :sender (agent-identifier :name BuyerAgent)
  :receiver (set (agent-identifier :name
    SellerAgent))
  :content
    ((action (agent-identifier :name SellerAgent)
      (sell :movie Gladiator))
     (any ?x (and (= ?x (price Gladiator)) (< ?x 20))))
  :ontology movie-ontology
  :language FIPA-SL)
```

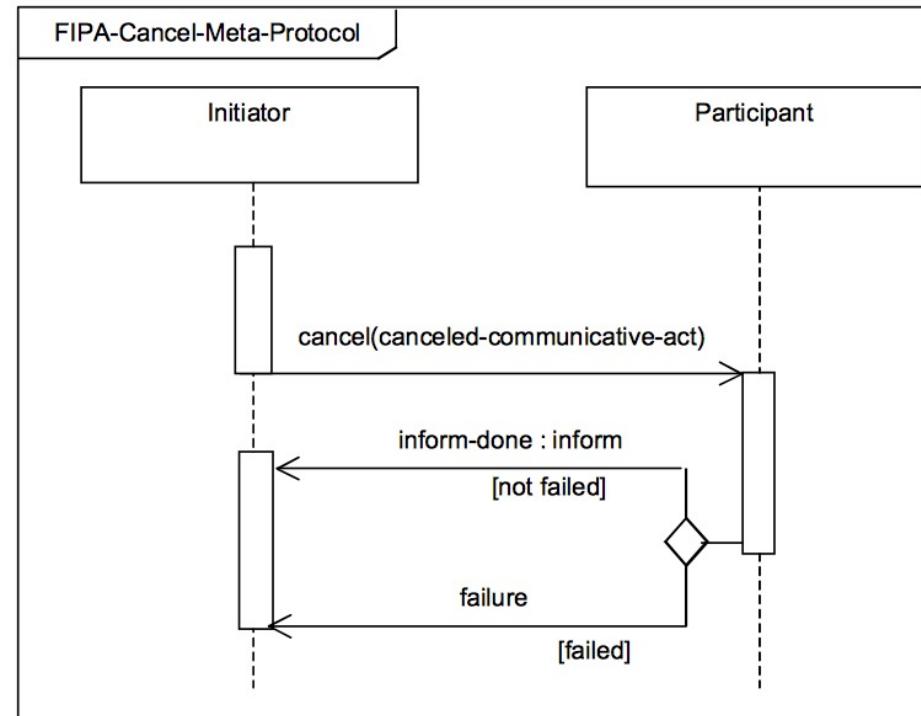
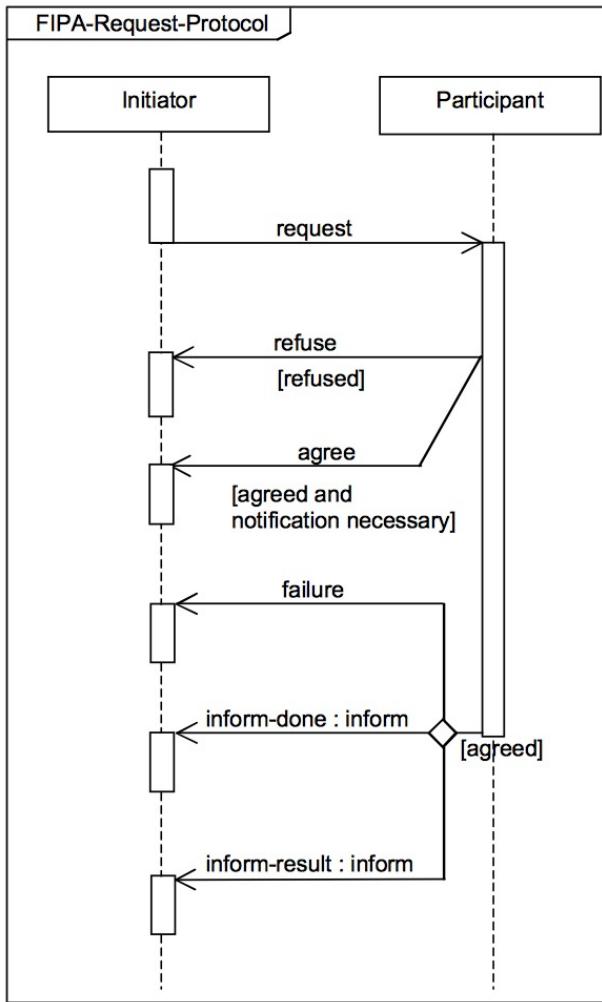
# FIPA Semantic Language

- FIPA Semantic Language (FIPA SL) is a formal language to define the content of the FIPA-ACL.
- FIPA SL can be used to express **objects**, **propositions** and **actions**
  - **Object** expression is used to declare variables and make assertions.
  - **Action** expressions describe some action that is either already performed, intended to be performed in the future or is currently being performed.
  - **Propositions** are used to represent the behavioural aspects of agents like goals, intents, beliefs and uncertainty.
- For example, agents may have persistent goals stated in the form

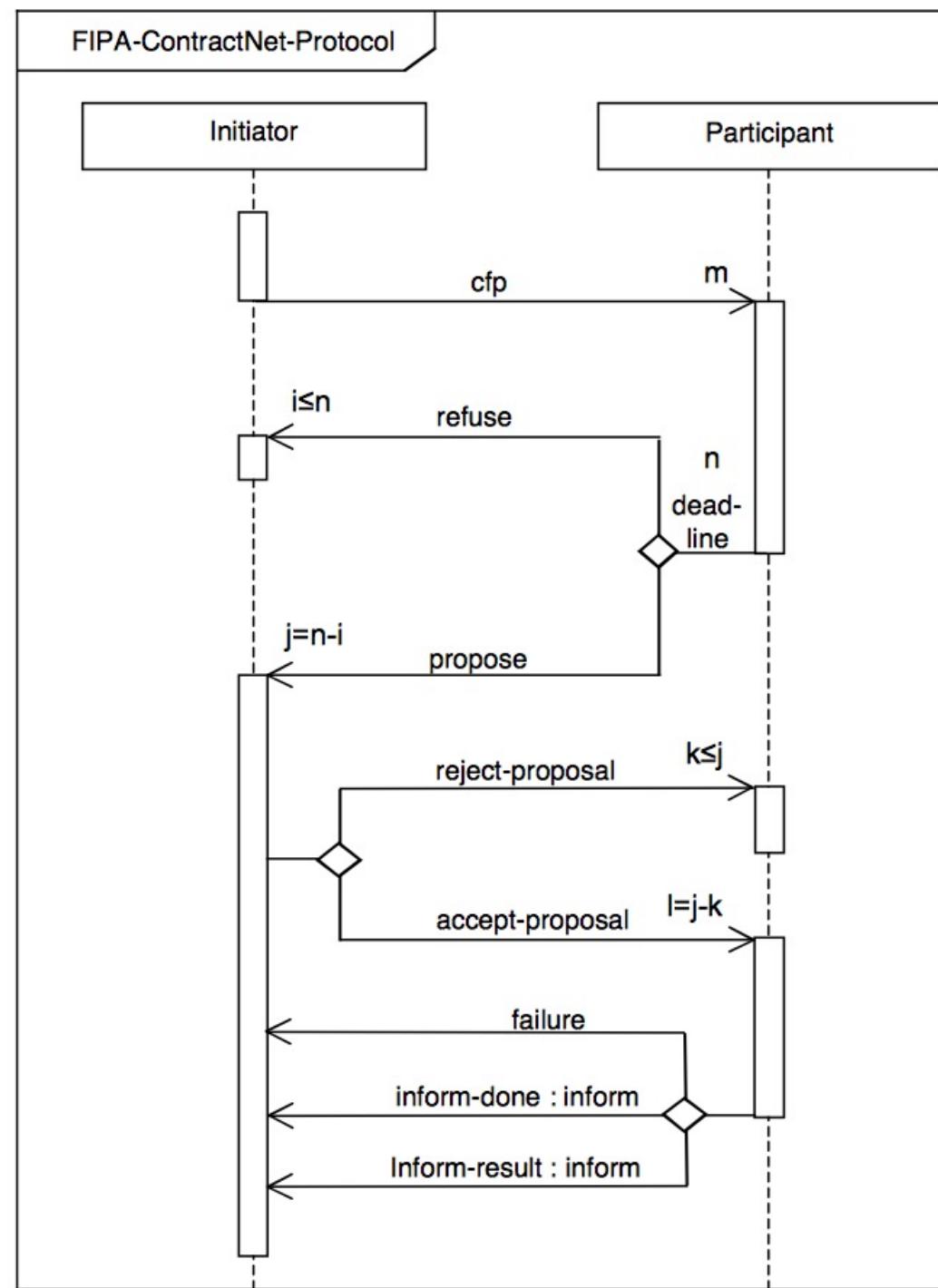
( PG < agent > < expression >)

- This states that an agent holds a persistent goal that expression becomes true but will not necessarily possess a plan to achieve this.

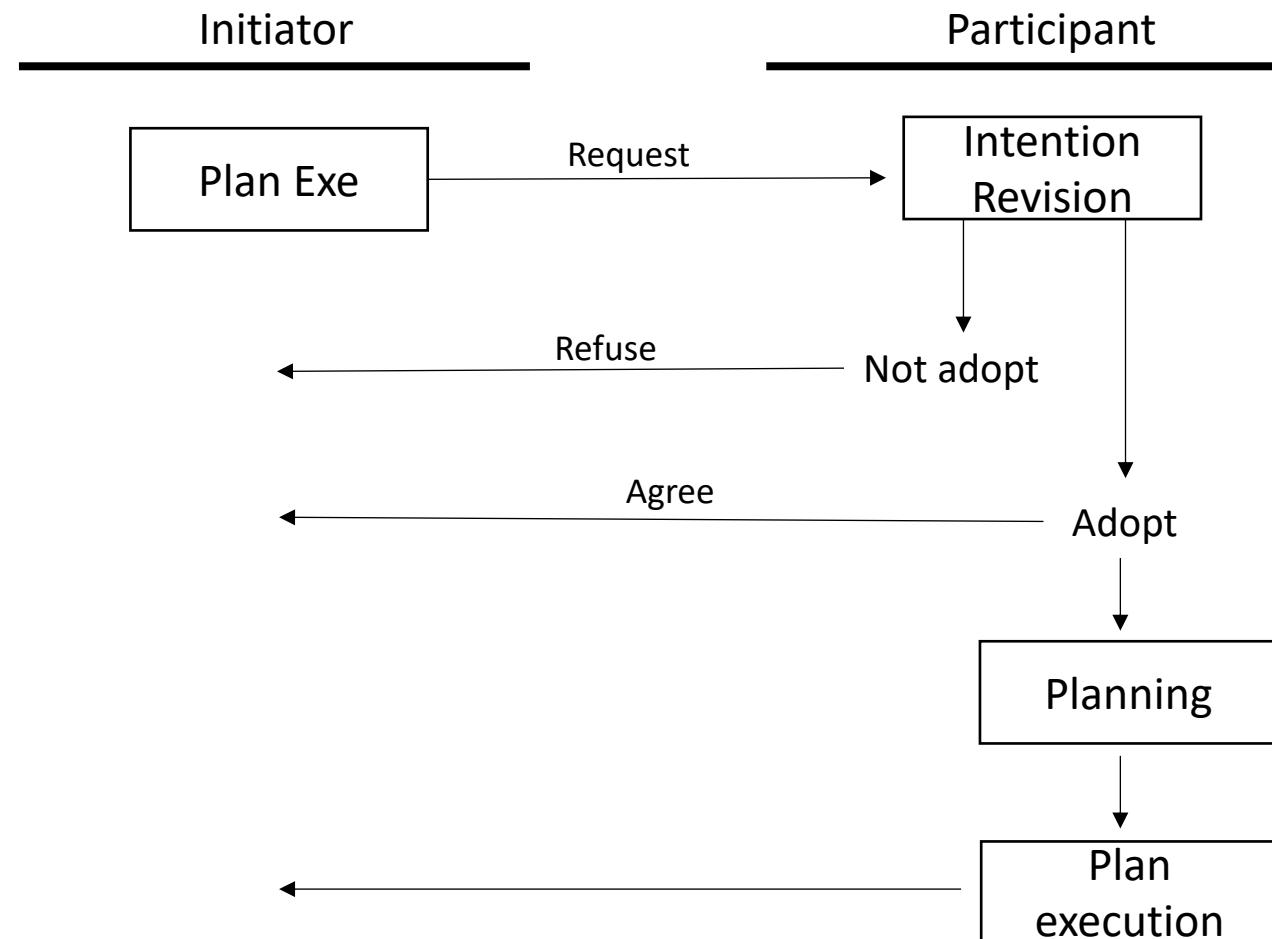
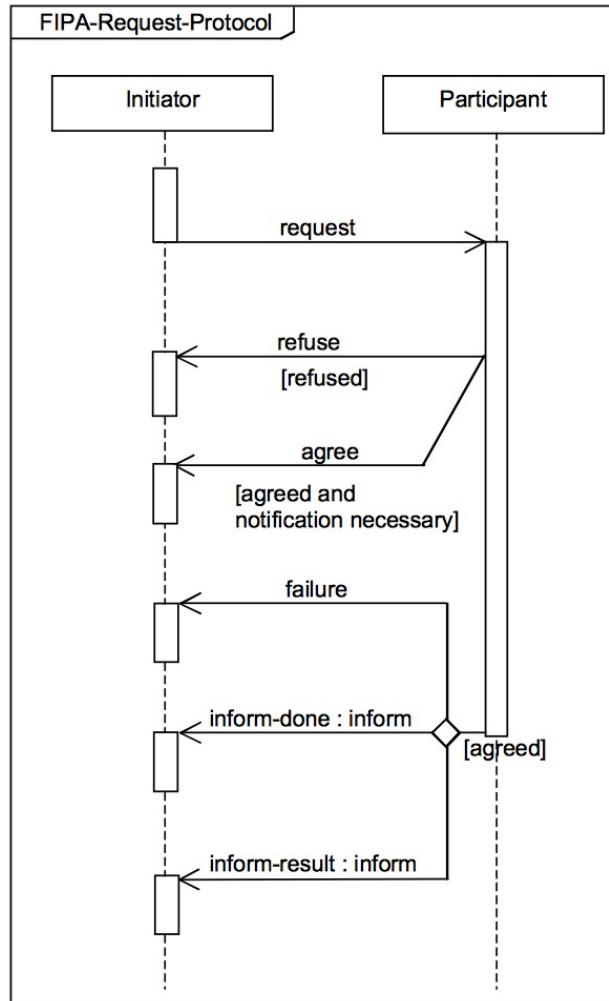
# FIPA REQUEST protocol



# FIPA ContractNet Protocol

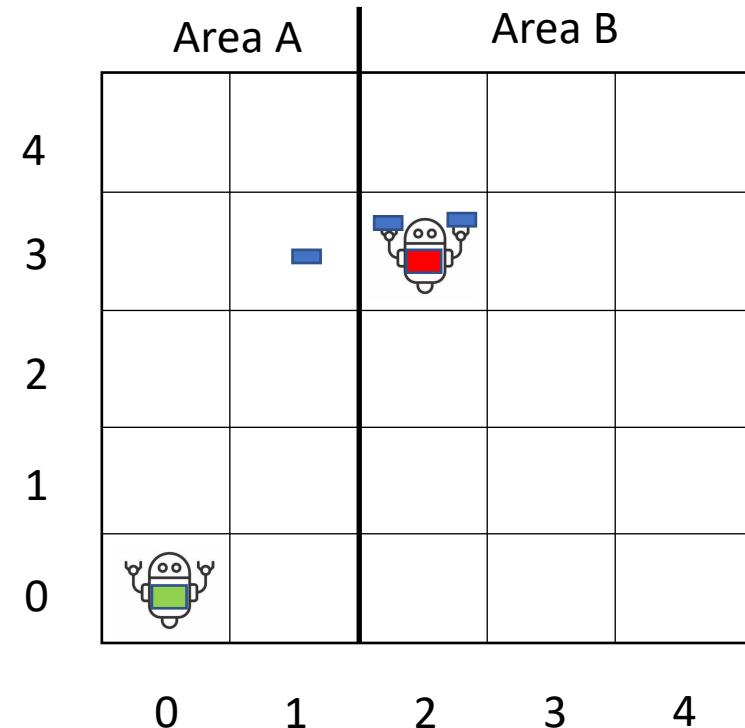


# Implementing a protocol in a BDI agent



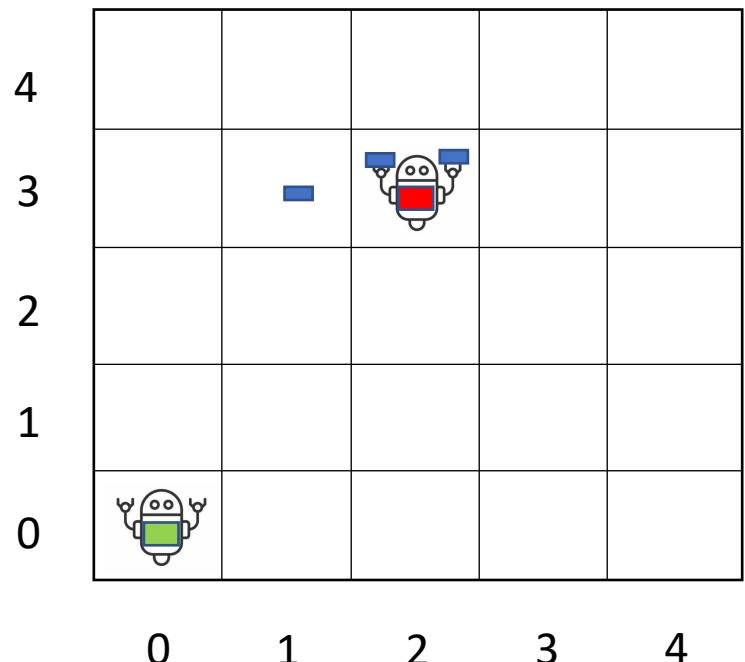
# Strategies of cooperation 1

- Agent **RED** is responsible to collect and deliver packages in AREA B
- Agent **GREEN** is responsible to collect and deliver packages in AREA A
- Agents have the same goals, but they work in separate areas
- No need for communication
- No interaction
- They cooperate in the sense they have a specific role (task) in the environment (overall objective)



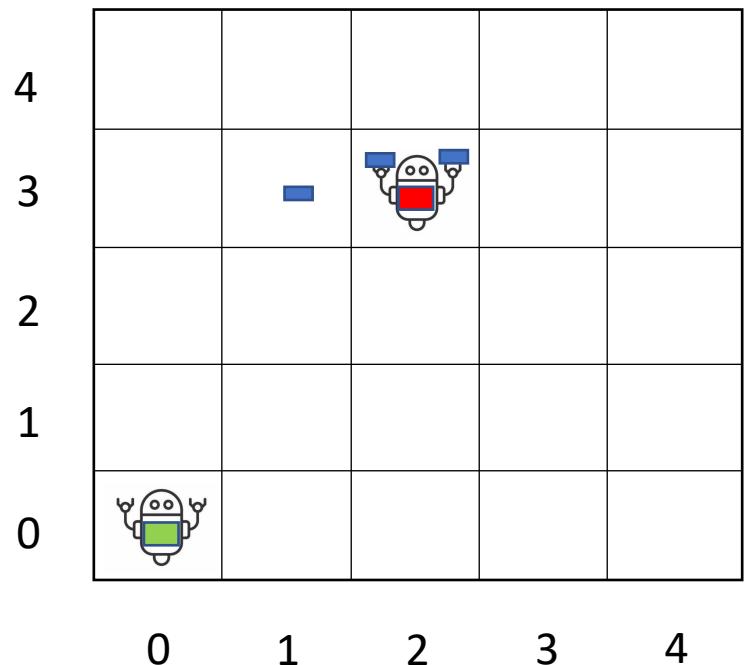
# Strategies of cooperation 2

- No limitations in covering area
- Agents share information
  - RED -> GREEN: `In(1,3,pack_1)`
  - RED -> GREEN: `not_intend(pick_up(pack_1))`
- A black board approach can be used
  - Broadcasting



# Strategies of cooperation 3

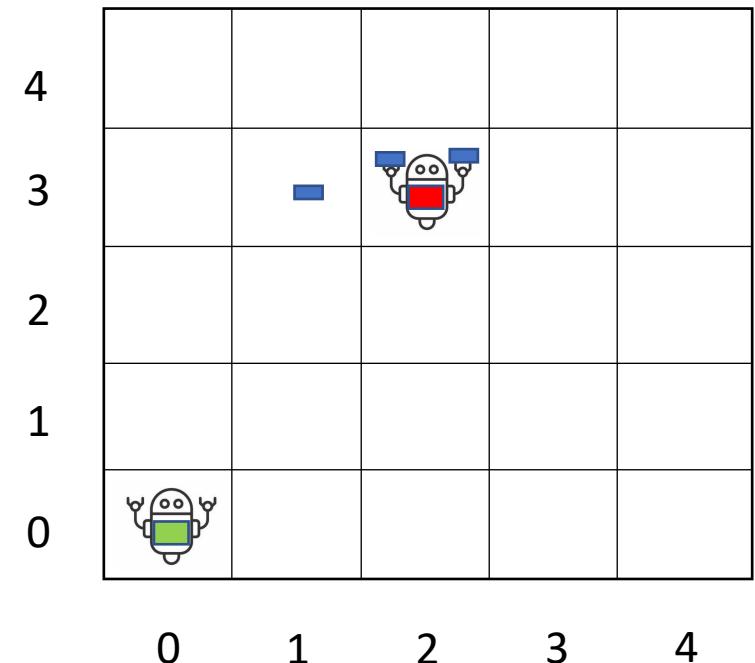
- No limitations in covering area
- Roles: Master and Slave
  - RED -> GREEN: request(`pick_up(pack_1)`)
  - GREEN -> RED: inform(`intend(pick_up(pack_1))`)
- More than two agents could be used
  - E.g., supervisor (master) that assign tasks to slave agents



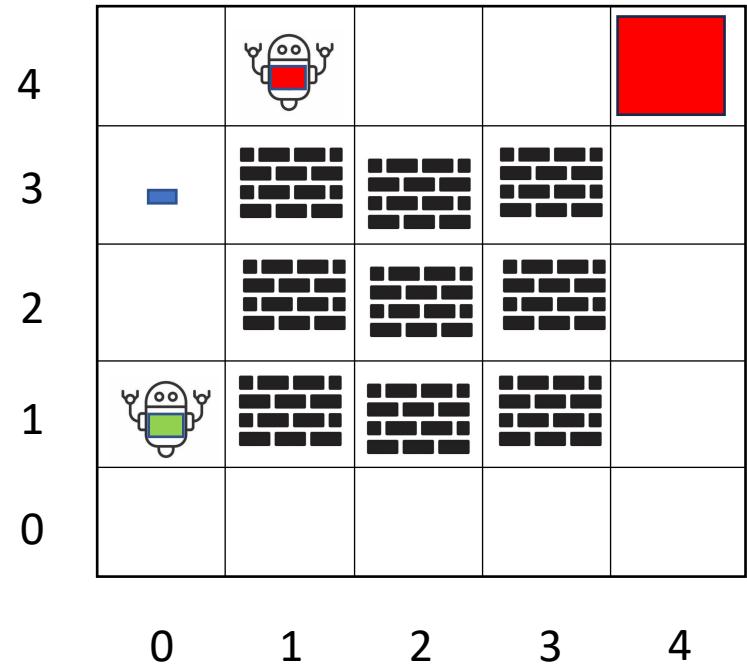
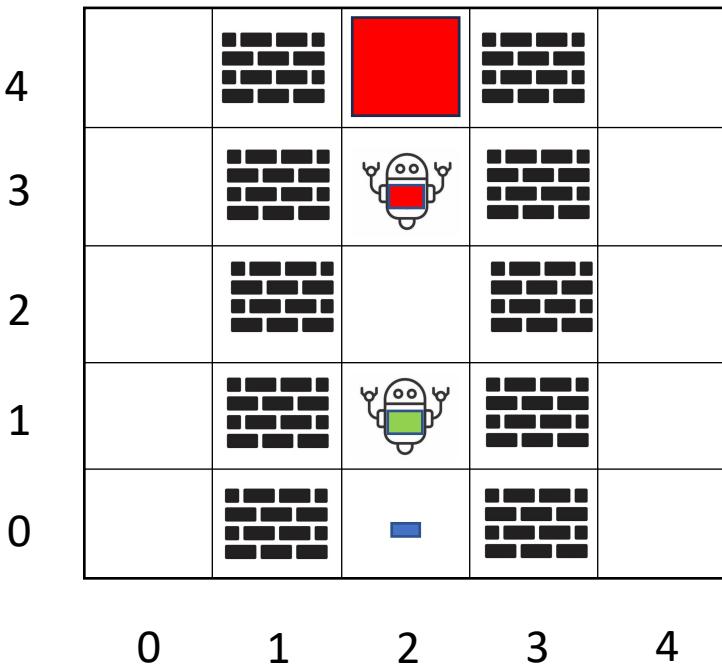
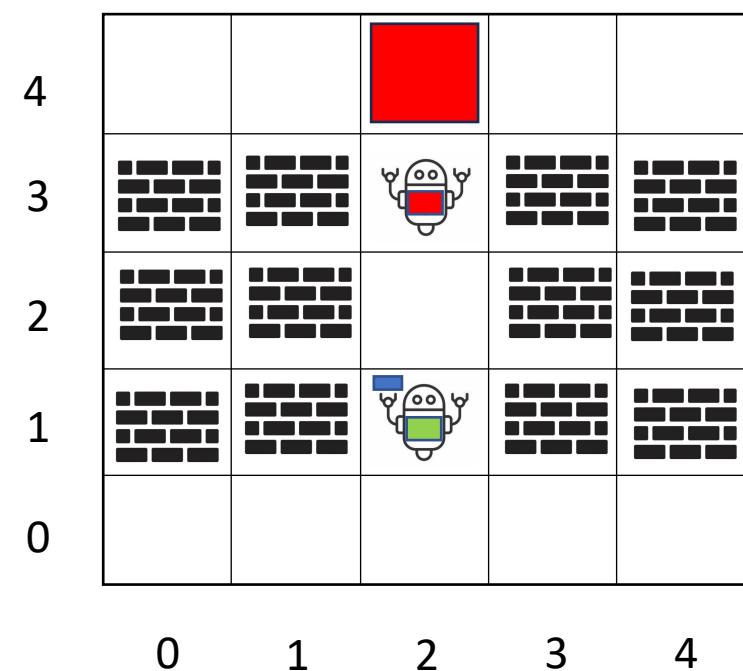
```
if in(pack,X,Y) -> request(RED, GREEN, pick_up(pack,X,Y))
```

# Strategies of cooperation 4 – negotiation

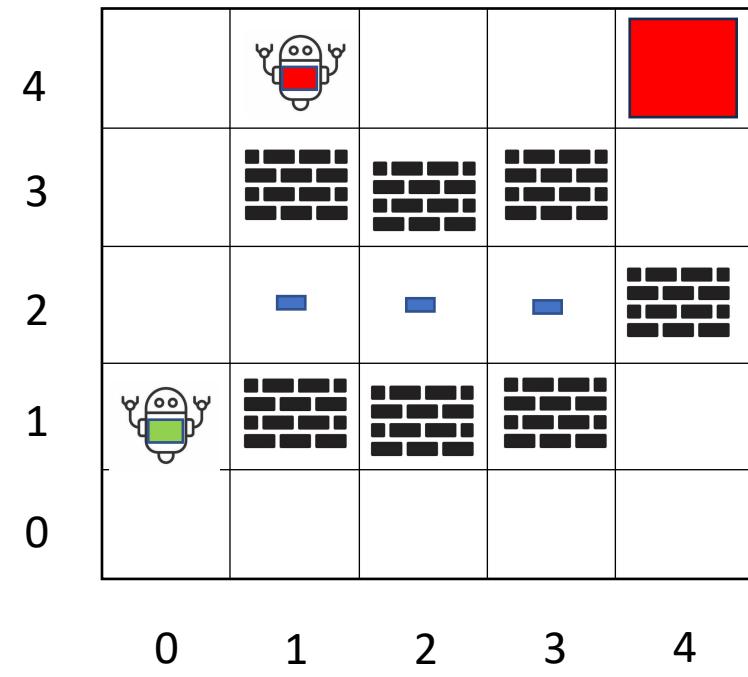
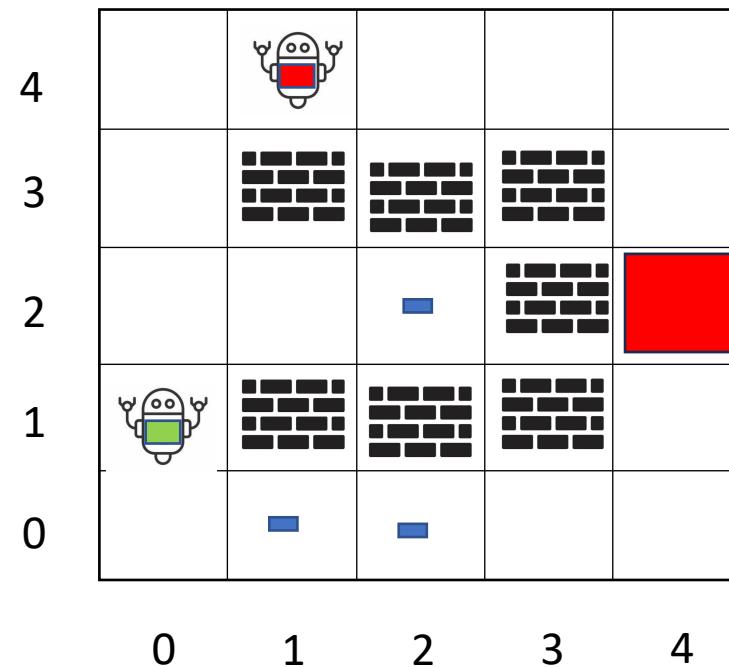
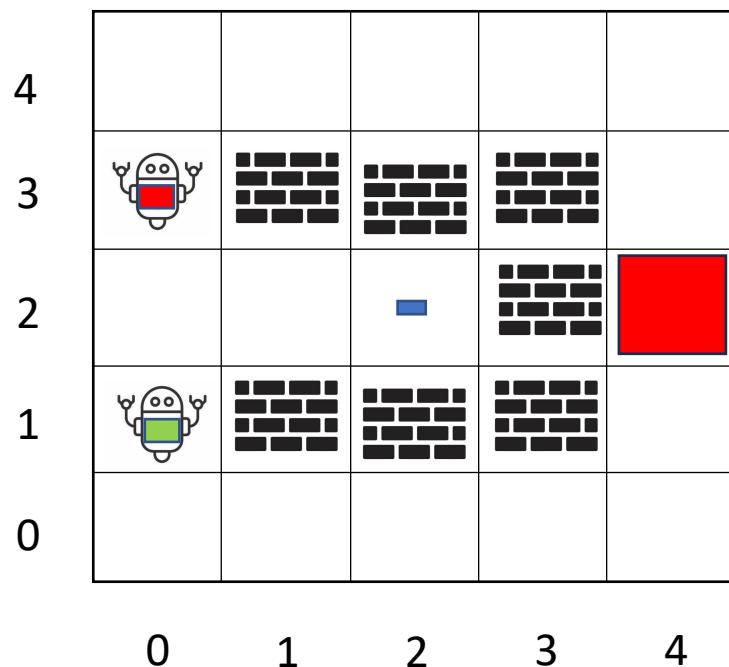
- Negotiation for the fulfilment of the request
  - RED -> GREEN: request(`pick_up(pack_1)`)
  - GREEN -> RED: inform(`pick_up(pack_1), 10s`)
  - RED -> GREEN: accept(`pick_up(pack_1), 10s`)
- Negotiation protocol needs to be designed



# Possible coordination scenarios



... and more



# Strategies for Deliveroo.js

Define your own strategy for Deliveroo.js

- ( $S_1$ ) two agents that exchange information about their mental states to coordinate activities and maximize the overall score
  - E.g., pack\_1 is in position (1,3); if I go to pick up the pack\_1 the score will be X, what about you? ... ok it is better you go.
- ( $S_2$ ) two agents that share information about their mental states and use a multi-agent plan to maximize the overall score
  - E.g., one of the two agents take the responsibility of managing the intentions of both agents and elaborating a multi-agent plan
- ( $S_3$ ) define your own strategy

