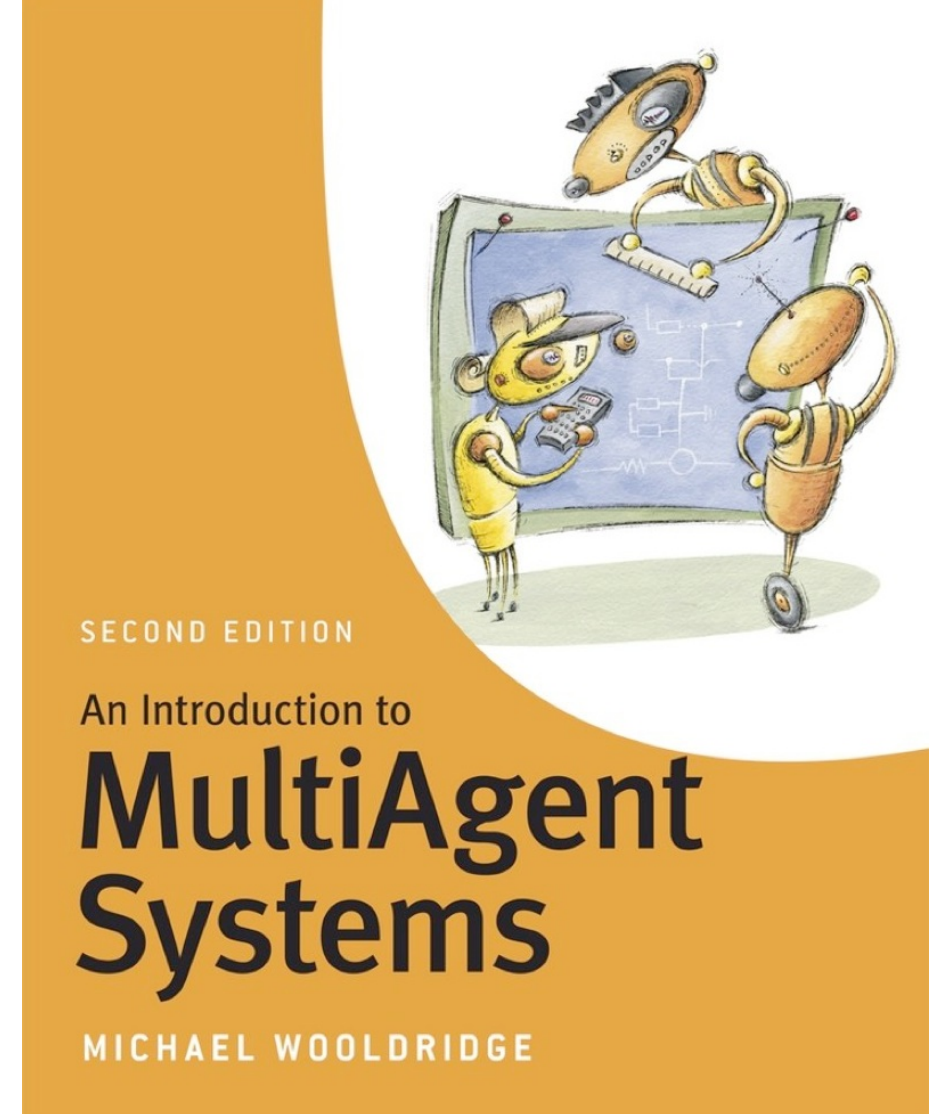


# COMP310

# MultiAgent Systems

## Chapter 2 - Intelligent Agents



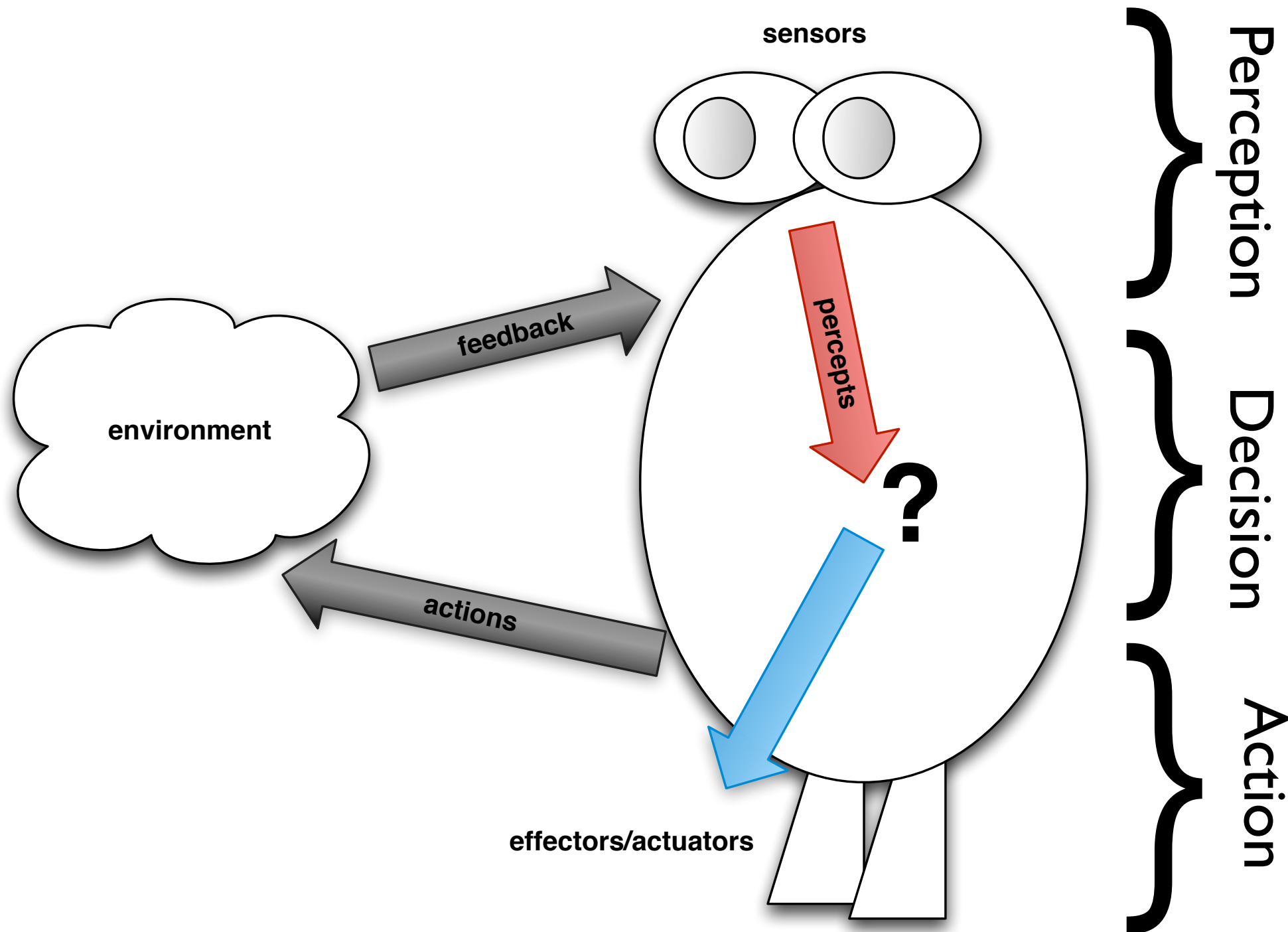
# What is an Agent?

- The main point about agents is they are **autonomous**: capable independent action.
- Thus:

“...An agent is a computer system that is *situated* in some *environment*, and that is capable of *autonomous action* in that environment in order to meet its delegated objectives...”

- It is all about decisions
  - An agent has to choose **what** action to perform.
  - An agent has to decide **when** to perform an action.

# Agent and Environment



# Autonomy

- There is a **spectrum** of autonomy



Simple Machines  
(no autonomy)

People  
(full autonomy)

- Autonomy is adjustable
  - Decisions handed to a higher authority when this is beneficial

# Simple (Uninteresting) Agents

- Thermostat
  - *delegated goal* is maintain room temperature
  - *actions* are heat on/off
- UNIX biff program
  - *delegated goal* is monitor for incoming email and flag it
  - *actions* are GUI actions.
- They are trivial because the **decision making** they do is trivial.



# Intelligent Agents

- We typically think of an intelligent agent as exhibiting 3 types of behaviour:
  - Pro-active (goal-driven);
  - Reactive (environment aware)
  - Social Ability.

# Proactiveness

- Reacting to an environment is easy
  - e.g., *stimulus* → *response rules*
- But we generally want agents to **do things for us**.
  - Hence **goal directed behaviour**.
- **Pro-activeness** = generating and attempting to achieve goals; not driven solely by events; taking the initiative.
  - Also: recognising opportunities.

# Reactivity

- If a program's environment is guaranteed to be fixed, a program can just execute blindly.
  - The real world is not like that: most environments are **dynamic** and information is **incomplete**.
- Software is hard to build for dynamic domains: program must take into account possibility of failure
  - ask itself whether it is worth executing!
- A **reactive** system is one that maintains an ongoing interaction with its environment, and responds to changes that occur in it (in time for the response to be useful).



# Social Ability

- The real world is a **multi**-agent environment: we cannot go around attempting to achieve goals without taking others into account.
  - Some goals can only be achieved by interacting with others.
  - Similarly for many computer environments: witness the INTERNET.
- **Social ability** in agents is the ability to interact with other agents (and possibly humans) via **cooperation**, **coordination**, and **negotiation**.
  - At the very least, it means the ability to communicate...

# Social Ability: Cooperation

- Cooperation is ***working together as a team to achieve a shared goal.***
- Often prompted either by the fact that no one agent can achieve the goal alone, or that cooperation will obtain a better result (e.g., get result faster).

# Social Ability: Coordination

- Coordination is ***managing the interdependencies between activities.***
- For example, if there is a non-sharable resource that you want to use and I want to use, then we need to coordinate.

# Social Ability: Negotiation

- Negotiation is the ability to reach agreements on matters of common interest.
- For example: *You have one TV in your house; you want to watch a movie, your housemate wants to watch football.*
- A possible deal: *watch football tonight, and a movie tomorrow.*
- Typically involves offer and counter-offer, with compromises made by participants.

# Some Other Properties...

- **Mobility**

- The ability of an agent to move. For software agents this movement is around an electronic network.

- **Veracity**

- Whether an agent will knowingly communicate false information.

- **Benevolence**

- Whether agents have conflicting goals, and thus whether they are inherently helpful.

- **Rationality**

- Whether an agent will act in order to achieve its goals, and will not deliberately act so as to prevent its goals being achieved.

- **Learning/adaption**

- Whether agents improve performance over time.

# Agents and Objects

- **Are agents just objects by another name?**
- Object:
  - encapsulates some state;
  - communicates via message passing;
  - has methods, corresponding to operations that may be performed on this state.

“...Agents are objects with **attitude**...”

# Differences between Agents and Objects

- **Agents are autonomous:**

- agents embody stronger notion of autonomy than objects, and in particular, they decide for themselves whether or not to perform an action on request from another agent;

- **Agents are smart:**

- capable of flexible (reactive, pro-active, social) behaviour – the standard object-oriented model has nothing to say about such types of behaviour;

- **Agents are active:**

- not passive service providers; a multi-agent system is inherently multi-threaded, in that each agent is assumed to have at least one thread of active control.

Objects do it because they **have to!**

Objects do it for **free!**

Agents do it because they **want to!**

Agents do it for **money!**

# Aren't agents just expert systems by another name?

- Expert systems typically disembodied 'expertise' about some (abstract) domain of discourse.
- agents are ***situated in an environment***:
  - MYCIN is not aware of the world — only information obtained is by asking the user questions.
- agents act:
  - MYCIN does not operate on patients.

**MYCIN** is an example of an Expert System that knows about blood diseases in humans.

It has a wealth of knowledge about blood diseases, in the form of rules.

A doctor can obtain expert advice about blood diseases by giving MYCIN facts, answering questions, and posing queries.



# Intelligent Agents and AI

- Aren't agents just the AI project?  
Isn't building an agent what AI is all about?
- AI aims to build systems that can (ultimately) understand natural language, recognise and understand scenes, use common sense, think creatively, etc — all of which are very hard.
- So, don't we need to solve all of AI to build an agent...?

# Intelligent Agents and AI

- When building an agent, we simply want a system that can choose the right action to perform, typically in a limited domain.
- We **do not** have to solve **all** the problems of AI to build a useful agent:

**“...a little intelligence goes a long way!..”**

- Oren Etzioni, speaking about the commercial experience of NETBOT, Inc:

*“...We made our agents dumber and dumber and dumber ... until finally they made money...”*

# Properties of Environments

- Since agents are in close contact with their environment, the properties of the environment affect agents.
  - Also have a big effect on those of us who build agents.
- Common to categorise environments along some different dimensions.
  - Fully observable vs partially observable
  - Deterministic vs non-deterministic
  - Episodic vs non-episodic
  - Static vs dynamic
  - Discrete vs continuous

# Properties of Environments

- Fully observable vs partially observable.
  - An accessible or **fully observable** environment is one in which the agent can obtain complete, accurate, up-to-date information about the environment's state.
  - Most moderately complex environments (including, for example, the everyday physical world and the Internet) are inaccessible, or **partially observable**.
  - The more accessible an environment is, the simpler it is to build agents to operate in it.

# Properties of Environments

- Deterministic vs non-deterministic.
  - A deterministic environment is one in which any action has a single guaranteed effect — there is no uncertainty about the state that will result from performing an action.
  - The physical world can to all intents and purposes be regarded as non-deterministic.
  - We'll follow *Russell and Norvig* in calling environments *stochastic* if we quantify the non-determinism using probability theory.
  - Non-deterministic environments present greater problems for the agent designer.

# Properties of Environments

- Episodic vs non-episodic.
  - In an episodic environment, the performance of an agent is dependent on a number of discrete episodes, with no link between the performance of an agent in different scenarios.
    - An example of an episodic environment would be an assembly line where an agent had to spot defective parts.
  - Episodic environments are simpler from the agent developer's perspective because the agent can decide what action to perform based only on the current episode — it need not reason about the interactions between this and future episodes.
  - Environments that are not episodic are called either **non-episodic** or **sequential**. Here the current decision affects future decisions.
    - Driving a car is sequential.

# Properties of Environments

- Static vs dynamic.
  - A **static** environment is one that can be assumed to remain unchanged except by the performance of actions by the agent.
  - A **dynamic** environment is one that has other processes operating on it, and which hence changes in ways beyond the agent's control.
  - The physical world is a highly dynamic environment.
  - One reason an environment may be dynamic is the presence of other agents.

# Properties of Environments

- Discrete vs continuous.
  - An environment is discrete if there are a fixed, finite number of actions and percepts in it.
  - *Russell and Norvig* give a chess game as an example of a discrete environment, and taxi driving as an example of a continuous one.





# Agents as Intentional Systems

- When explaining human activity, it is often useful to make statements such as the following:
  - Janine took her umbrella because she **believed** it was going to rain.
  - Michael worked hard because he **wanted** to possess a PhD.
- These statements make use of a **folk psychology**, by which human behavior is predicted and explained through the attribution of **attitudes**
  - e.g. *believing, wanting, hoping, fearing ...*
- The attitudes employed in such folk psychological descriptions are called the **intentional** notions.

# Dennett on Intentional Systems

- The philosopher Daniel Dennett coined the term **intentional system** to describe entities:

*“... whose behaviour can be predicted by the method of attributing belief, desires and rational acumen...”*

- Dennett identifies different ‘**grades**’ of intentional system:

*“...A **first-order** intentional system has beliefs and desires (etc.) but **no beliefs and desires about** beliefs and desires...”*

*...A **second-order** intentional system is more sophisticated; it has beliefs and desires (and no doubt other intentional states) about beliefs and desires (and other intentional states) — both those of others and its own...”*

- Is it legitimate or useful to attribute beliefs, desires, and so on, to computer systems?

# McCarthy on Intentional Systems

- John McCarthy argued that there are occasions when the **intentional stance** is appropriate:

*“...To ascribe **beliefs, free will, intentions, consciousness, abilities, or wants** to a machine is legitimate when such an ascription expresses the same information about the machine that it expresses about a person. It is **useful** when the ascription helps us understand the structure of the machine, its past or future behaviour, or how to repair or improve it. It is perhaps never **logically required** even for humans, but expressing reasonably briefly what is actually known about the state of the machine in a particular situation may require mental qualities or qualities isomorphic to them.*

*Theories of belief, knowledge and wanting can be constructed for machines in a simpler setting than for humans, and later applied to humans. Ascription of mental qualities is **most straightforward** for machines of known structure such as thermostats and computer operating systems, but is most useful when applied to entities whose structure is incompletely known ...”*

# What can be described with the intentional stance?

- As it turns out, more or less anything can... consider a light switch:

*“... It is perfectly coherent to treat a light switch as a (very cooperative) agent with the capability of transmitting current at will, who invariably transmits current when it believes that we want it transmitted and not otherwise; flicking the switch is simply our way of communicating our desires ...” (Yoav Shoham)*



- But most adults would find such a description absurd!
  - Why is this?

# What can be described with the intentional stance?

- The answer seems to be that while the intentional stance description is consistent:

*“... it does not buy us anything, since we essentially understand the mechanism sufficiently to have a simpler, mechanistic description of its behaviour ...” (Yoav Shoham)*

- Put crudely, the more we know about a system, the less we need to rely on animistic, intentional explanations of its behaviour.
- But with very complex systems, a mechanistic, explanation of its behaviour may not be practicable.
  - **As computer systems become ever more complex, we need more powerful abstractions and metaphors to explain their operation — low level explanations become impractical.**
  - **The intentional stance is such an abstraction.**

# Agents as Intentional Systems

- So agent theorists start from the (strong) view of agents as intentional systems: one whose simplest consistent description requires the intentional stance.
- This **intentional stance** is an **abstraction tool**...
  - ... a convenient way of talking about complex systems, which allows us to predict and explain their behaviour without having to understand how the mechanism actually works.
- Most important developments in computing are based on new **abstractions**:
  - procedural abstraction, abstract data types, objects, etc
- Agents, and agents as intentional systems, represent a further, and increasingly powerful abstraction.

*So why not use the intentional stance as an abstraction tool in computing — to explain, understand, and, crucially, **program** computer systems, through the notion of “agents”?*

# Agents as Intentional Systems

- There are other arguments in favour of this idea...
  1. Characterising Agents
    - It provides us with a familiar, non-technical way of understanding and explaining agents.
  2. Nested Representations
    - It gives us the potential to specify systems that include representations of other systems.
    - It is widely accepted that such nested representations are essential for agents that must cooperate with other agents.
    - “If you think that Agent B knows *x*, then move to location *L*”.

## *North by Northwest*



Eve Kendell **knows** that Roger Thornhill is working for the FBI. Eve **believes** that Philip Vandamm **suspects** that she is helping Roger. This, in turn, leads Eve to **believe** that Philip **thinks** she is working for the FBI (which is true). By pretending to shoot Roger, Eve **hopes** to convince Philip that she is not working for the FBI

# Agents as Intentional Systems

- There are other arguments in favour of this idea...

## 3. Post-Declarative Systems

- In **procedural programming**, we say exactly what a system should do;
- In **declarative programming**, we state something that we want to achieve, give the system general info about the relationships between objects, and let a built-in control mechanism (e.g., goal-directed theorem proving) figure out what to do;
- With agents, we give a **high-level description of the delegated goal**, and let the control mechanism figure out what to do, knowing that it will act in accordance with some built-in theory of rational agency.



# An aside...

- We find that researchers from a more mainstream computing discipline have adopted a similar set of ideas in **knowledge based protocols**.
- The idea: when constructing protocols, one often encounters reasoning such as the following:

If      process  $i$  **knows** process  $j$   
          **has received** message  $m_1$

Then process  $i$  **should send**  
          process  $j$  the message  $m_2$ .

# Abstract Architectures for Agents

- Assume the world may be in any of a finite set  $E$  of discrete, instantaneous **states**:

$$E = \{e, e', \dots\}$$

- Agents are assumed to have a repertoire of possible actions available to them, which transform the state of the world.

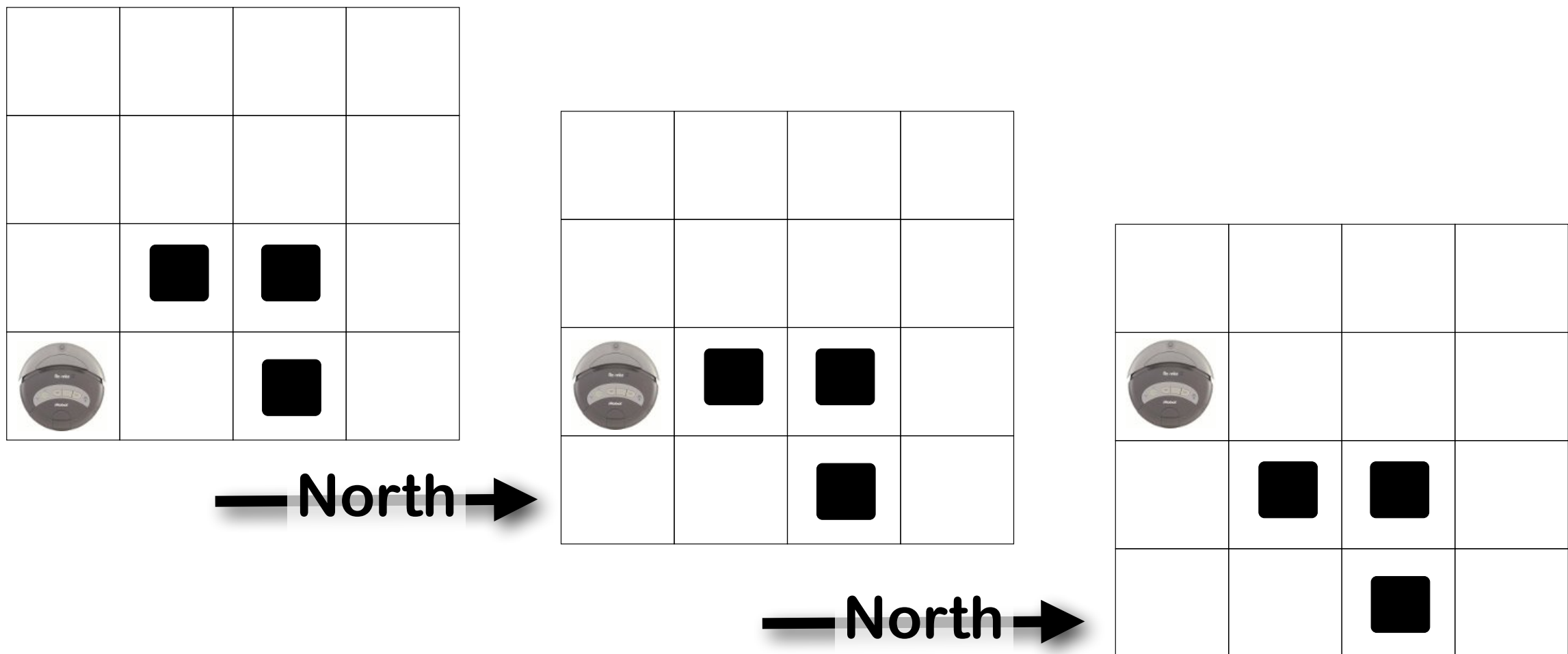
$$Ac = \{\alpha, \alpha', \dots\}$$

- Actions can be non-deterministic, but only one state ever results from an action.
- A **run**,  $r$ , of an agent in an environment is a sequence of interleaved world states and actions:

$$r : e_0 \xrightarrow{\alpha_0} e_1 \xrightarrow{\alpha_1} e_2 \xrightarrow{\alpha_2} e_3 \xrightarrow{\alpha_3} \dots \xrightarrow{\alpha_{u-1}} e_u$$

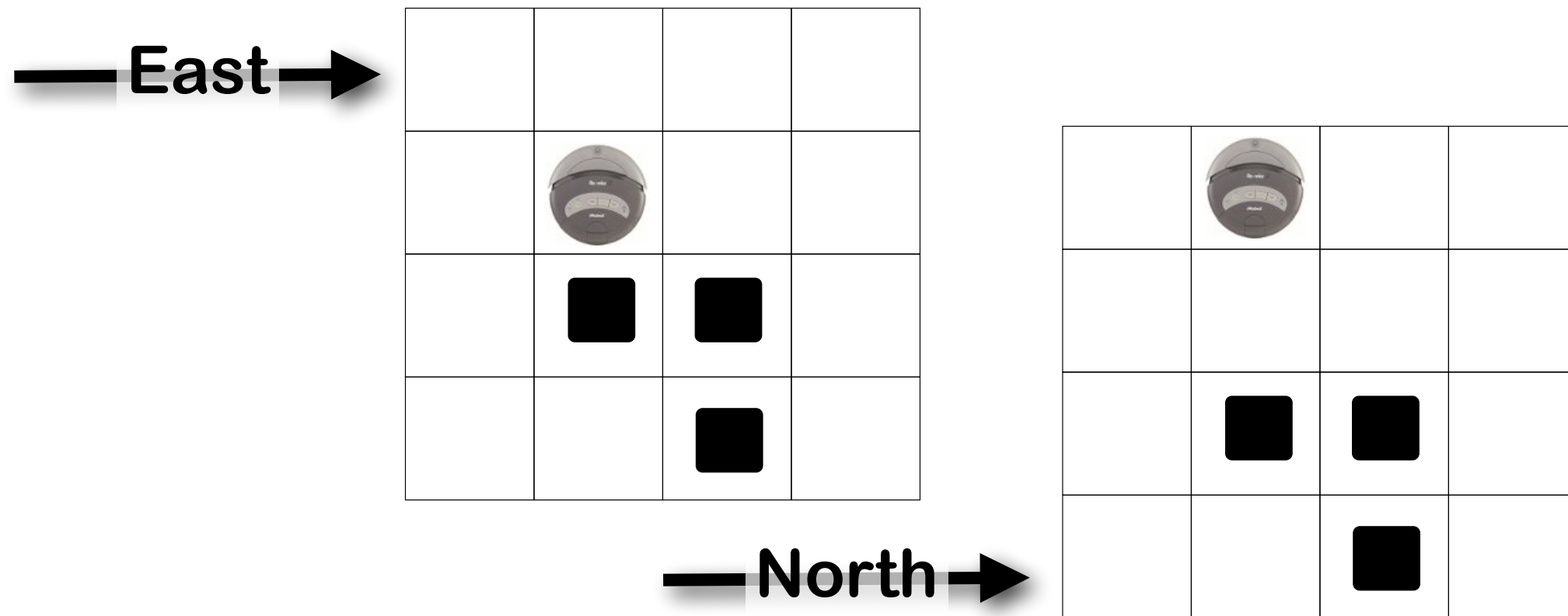
# Abstract Architectures for Agents (I)

- When actions are deterministic each state has only one possible successor.
- A run would look something like the following:

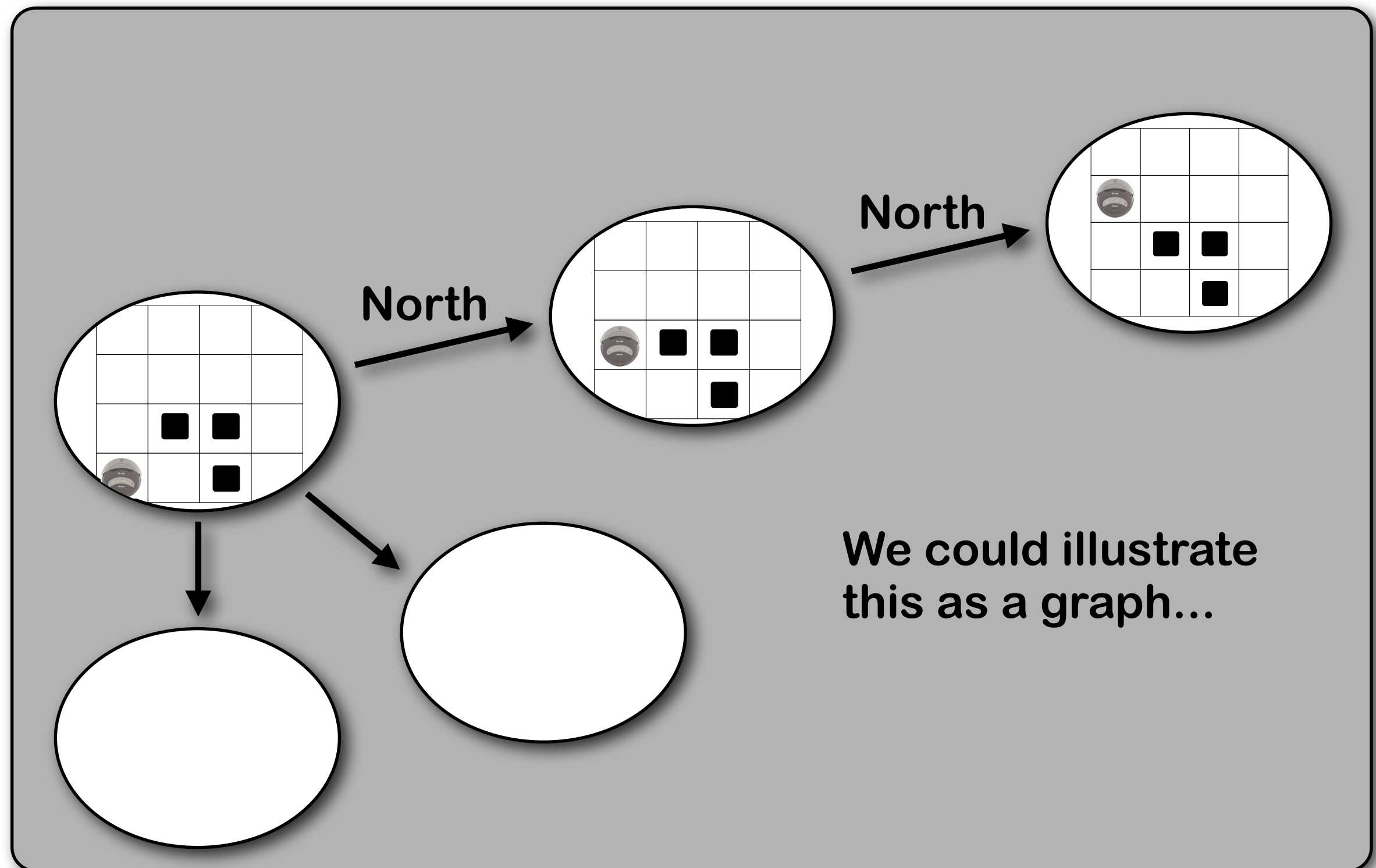


# Abstract Architectures for Agents (2)

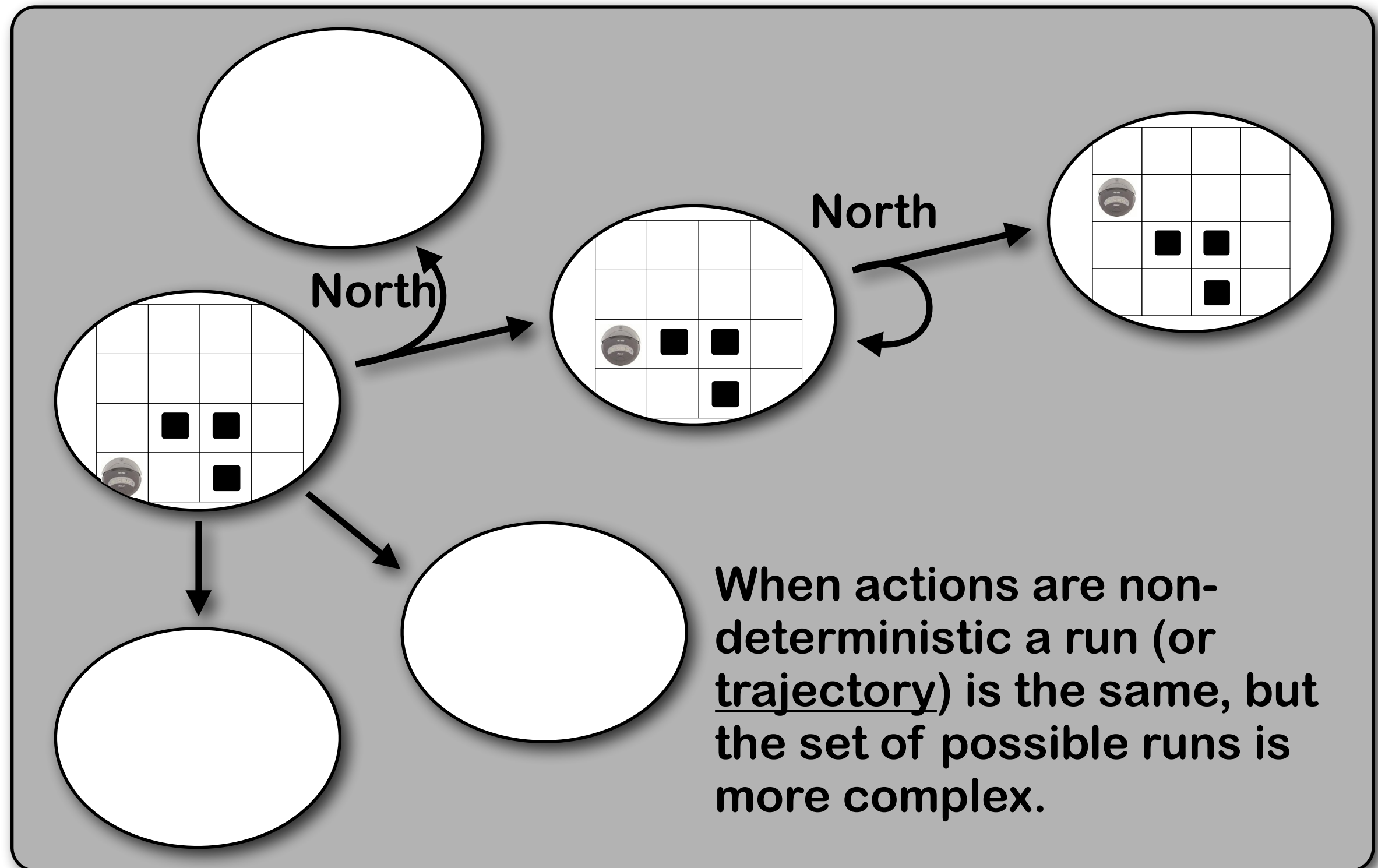
- When actions are deterministic each state has only one possible successor.
- A run would look something like the following:



# Abstract Architectures for Agents



# Abstract Architectures for Agents



# Runs

- In fact it is more complex still, because all of the runs we pictured start from the same state.
- Let:

$\mathcal{R}$  be the set of all such possible finite sequences (over  $E$  and  $Ac$ );  
 $\mathcal{R}^{Ac}$  be the subset of these that end with an action; and  
 $\mathcal{R}^E$  be the subset of these that end with a state.

- We will use  $r, r', \dots$  to stand for the members of  $\mathcal{R}$
- These sets of runs contain **all** runs from **all** starting states.

# Environments

- A state transformer function represents behaviour of the environment:

$$\tau : \mathcal{R}^{Ac} \rightarrow \wp(E)$$

- Note that environments are...
  - history dependent.
  - non-deterministic.
- If  $\tau(r) = \emptyset$  there are no possible successor states to  $r$ , so we say the run has ended. (“Game over.”)
- An environment  $Env$  is then a triple  $Env = \langle E, e_0, \tau \rangle$  where  $E$  is set of states,  $e_0 \in E$  is initial state; and  $\tau$  is state transformer function.



# Agents

- We can think of an agent as being a function which maps runs to actions:

$$Ag : \mathcal{R}^E \rightarrow Ac$$

- Thus an agent makes a decision about what action to perform based on the history of the system that it has witnessed to date.
- Let  $Ag$  be the set of all agents.

# Systems

- A **system** is a pair containing an *agent* and an *environment*.
- Any system will have associated with it a set of possible runs; we denote the set of runs of agent  $Ag$  in environment  $Env$  by:

$$\mathcal{R}(Ag, Env)$$

- Assume  $\mathcal{R}(Ag, Env)$  contains only runs that have ended.

# Systems

Formally, a sequence

$$(e_0, \alpha_0, e_1, \alpha_1, e_2, \dots)$$

represents a run of an agent  $Ag$  in environment  $Env = \langle E, e_0, \tau \rangle$  if:

1.  $e_0$  is the initial state of  $Env$

2.  $\alpha_0 = Ag(e_0)$ ; and

3. for  $u > 0$ ,

$$\begin{array}{ll} e_u & \in \tau((e_0, \alpha_0, \dots, \alpha_{u-1})) \quad \text{and} \\ \alpha_u & = Ag((e_0, \alpha_0, \dots, e_u)) \end{array}$$

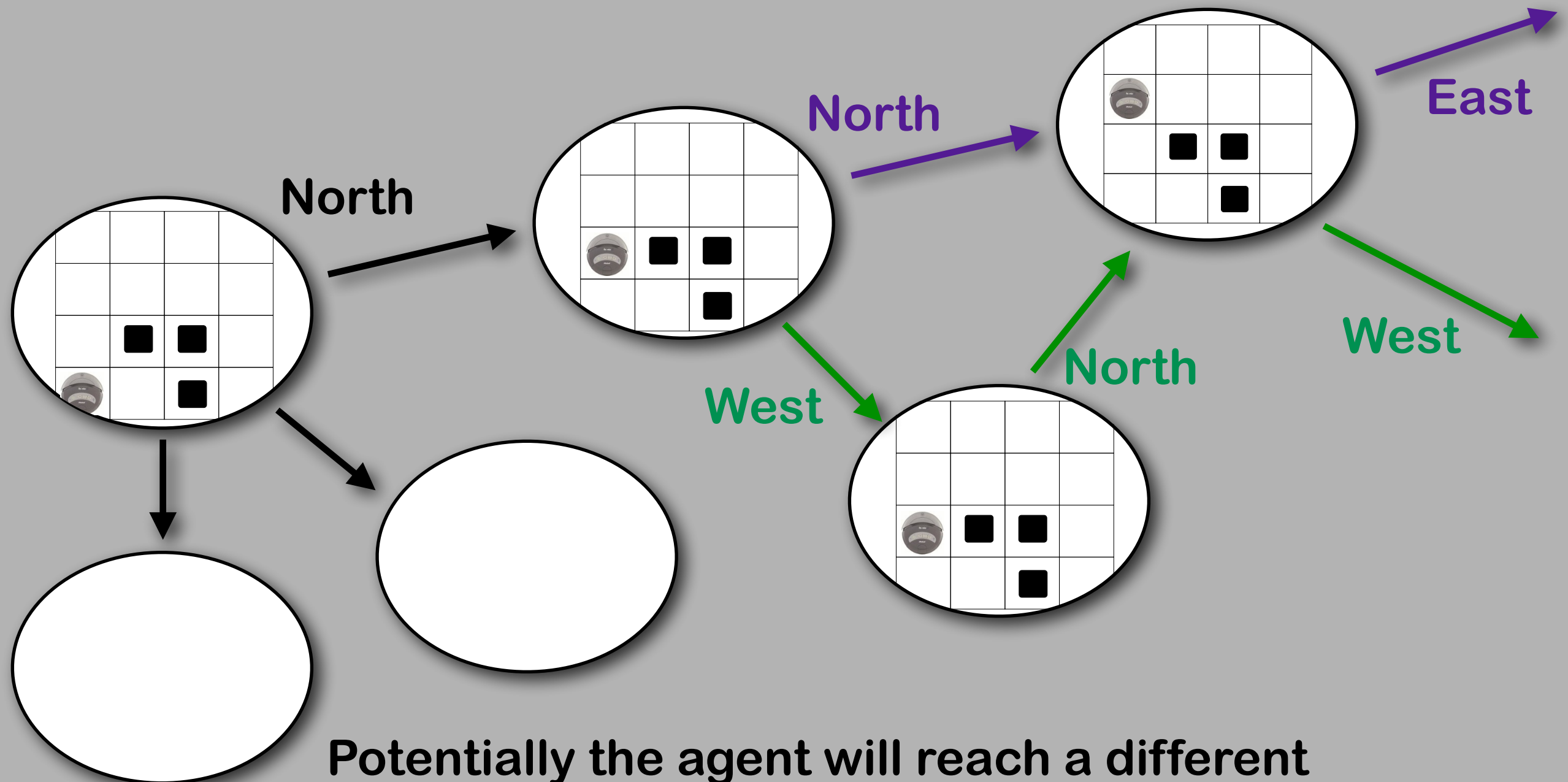
# Why the notation?

- Well, it allows us to get a precise handle on some ideas about agents.
  - For example, we can tell when two agents are the same.
- Of course, there are different meanings for “same”. Here is one specific one.

Two agents are said to be *behaviorally equivalent* with respect to *Env* iff  $\mathcal{R}(Ag_1, Env) = \mathcal{R}(Ag_2, Env)$ .

- We won't be able to tell two such agents apart by watching what they do.

# Deliberative Agents



Potentially the agent will reach a different decision when it reaches the same state by different routes.

# Purely Reactive Agents

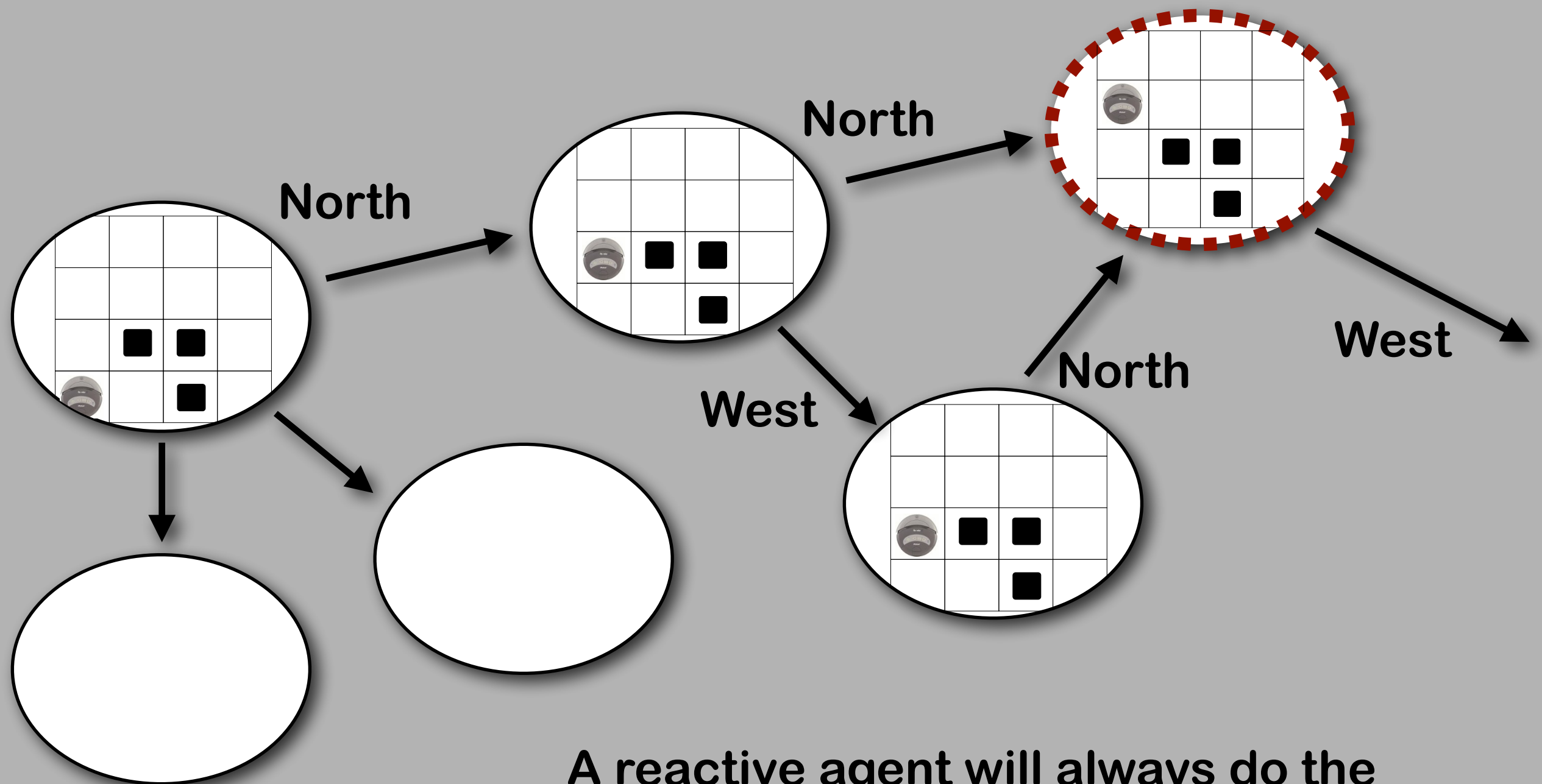
- Some agents decide what to do without reference to their history — they base their decision making entirely on the present, with no reference at all to the past.
- We call such agents ***purely reactive***:

$$action : E \rightarrow Ac$$

- A thermostat is a purely reactive agent.

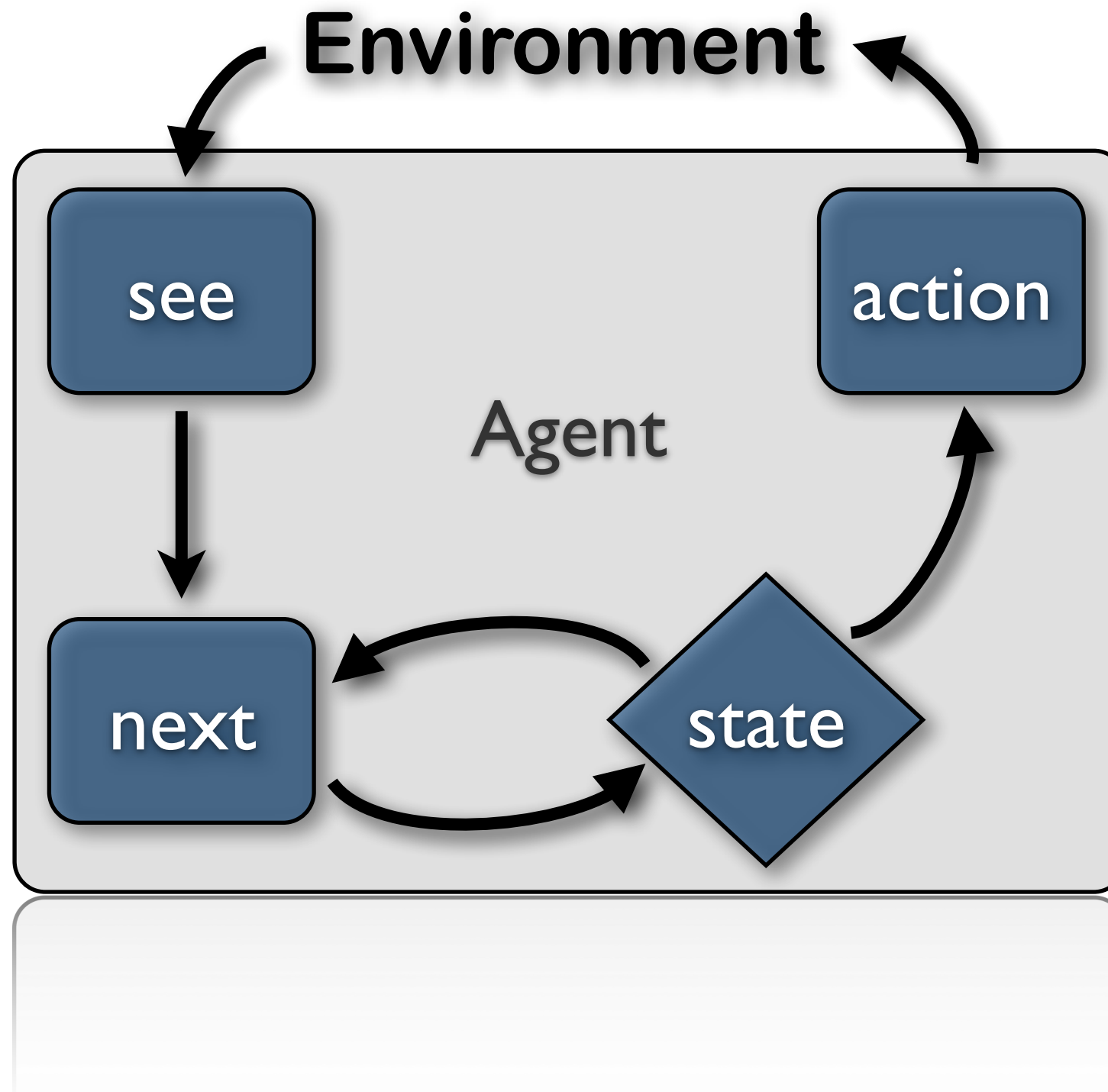
$$action(e) = \begin{cases} \text{off} & \text{if } e = \text{temperature OK} \\ \text{on} & \text{otherwise.} \end{cases}$$

# Reactive Agents



A reactive agent will always do the same thing in the same state.

# Agents with State



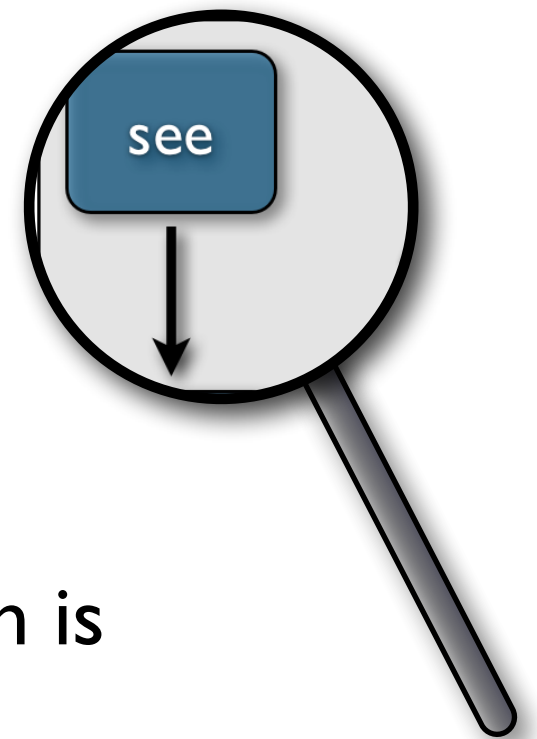


# Perception

- The see function is the agent's ability to observe its environment, whereas the action function represents the agent's decision making process.
- **Output** of the *see* function is a **percept**:

$$see : E \rightarrow Per$$

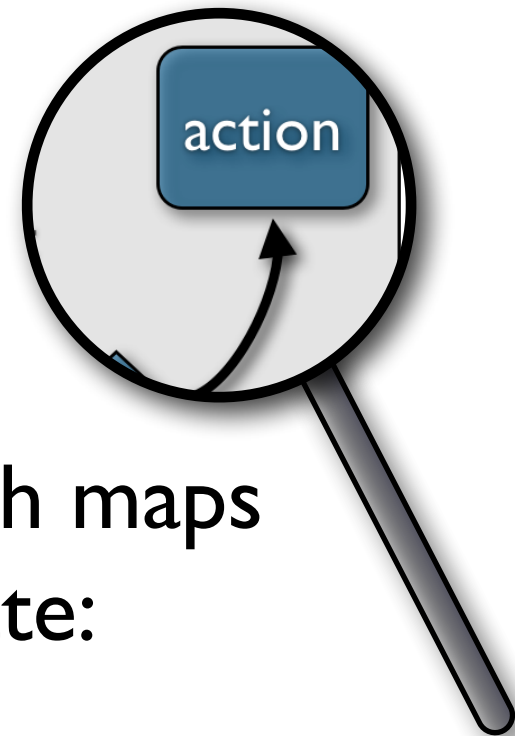
- ...which maps environment states to percepts.
- The agent has some internal data structure, which is typically used to record information about the environment state and history.
- Let  $I$  be the set of all internal states of the agent.



# Actions and Next State Functions

- The action-selection function *action* is now defined as a mapping from internal states to actions:

$$action : I \rightarrow Ac$$



- An additional function *next* is introduced, which maps an internal state and percept to an internal state:

$$next : I \times Per \rightarrow I$$



- This says how the agent updates its view of the world when it **gets a new percept**.

# Agent Control Loop

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

# Tasks for Agents

- We build agents in order to carry out **tasks** for us.
- The task must be **specified** by us...
- But we want to tell agents what to do **without** telling them how to do it.
- How can we make this happen???

# Utility functions

- One idea: associated **rewards** with states that we want agents to bring about.
- We associate **utilities** with individual states — the task of the agent is then to bring about states that maximise utility.
- A task specification is then a function which associates a real number with every environment state:

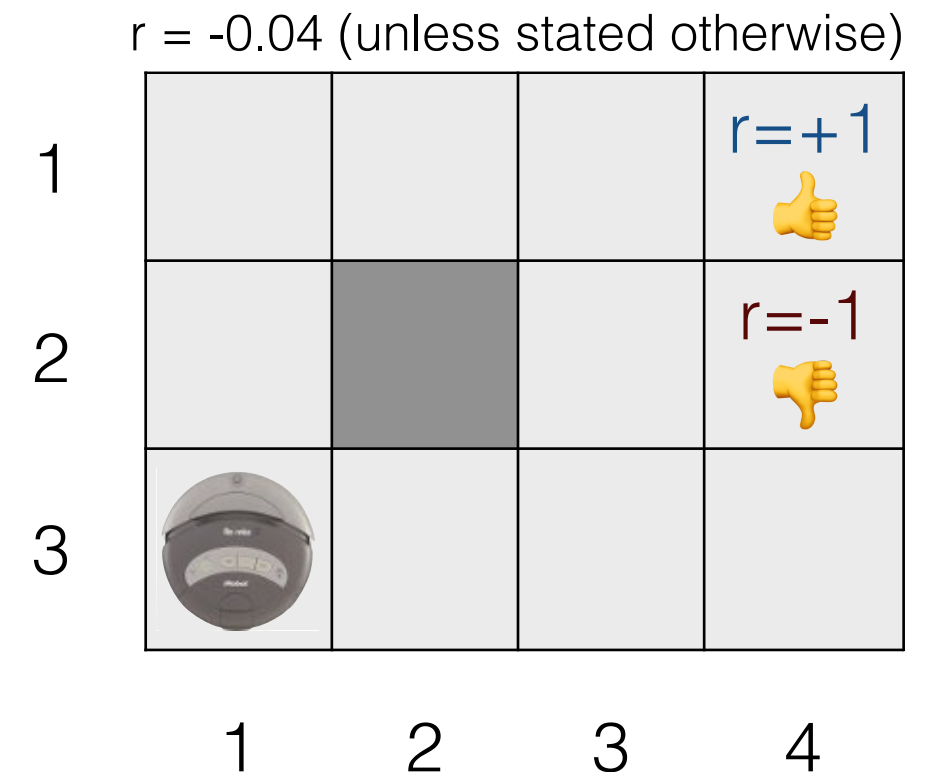
$$u : E \rightarrow \mathbb{R}$$

# Local Utility Functions

- But what is the value of a run...
  - minimum utility of state on run?
  - maximum utility of state on run?
  - sum of utilities of states on run?
  - average?
- Disadvantage:
  - difficult to specify a **long term** view when assigning utilities to individual states.
- One possibility:
  - a **discount** for states later on. This is what we do in **reinforcement learning**.

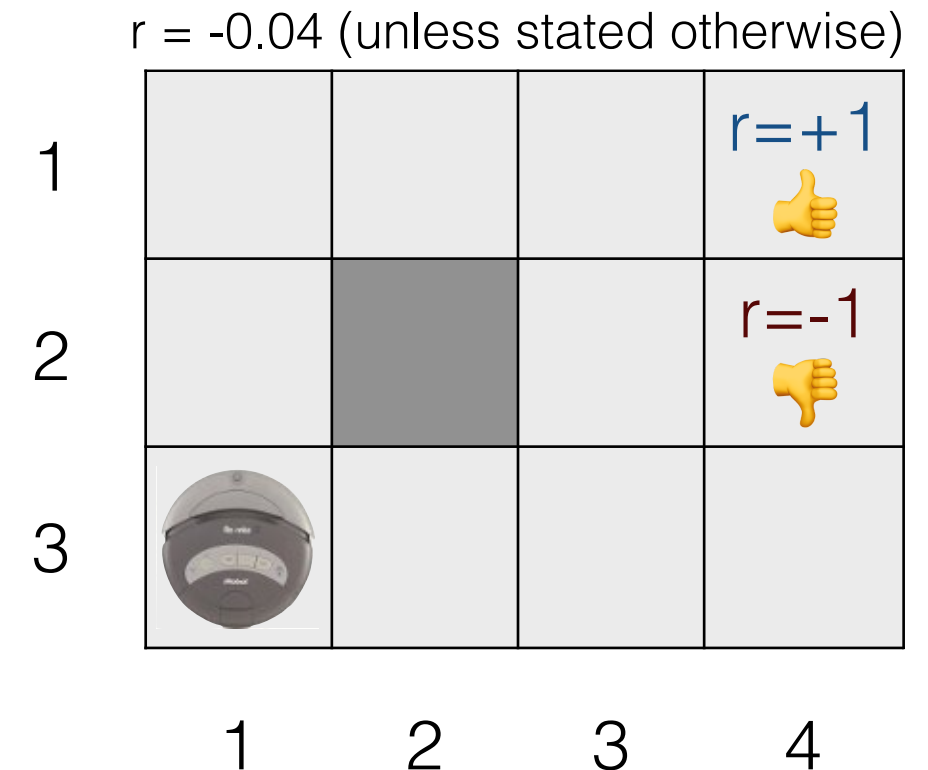
# Example of local utility function

- Goal is to select actions to maximise **future rewards**
  - Each action results in moving to a state with some assigned reward
  - Allocation of that reward may be immediate or delayed (e.g. until the end of the run)
    - It may be better to sacrifice immediate reward to gain more long-term reward
- We can illustrate with a simple 4x3 environment
  - What actions maximise the reward?



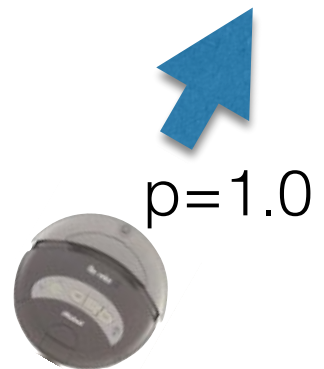
# Example of local utility function

- Assume environment was **deterministic**
  - Optimal Solution is:
    - [Up, Up, Right, Right, Right]
  - **Additive Reward** is:
    - $r = (-0.04 \times 4) + 1.0$
    - $r = 1.0 - 0.16 = 0.84$
- i.e. the utility gained is the sum of the rewards received
  - The negative (-0.04) reward incentivises the agent to reach its goal asap.



## Deterministic Environment

Agent is guaranteed to be in the intended cell (i.e. probability = 1.0)






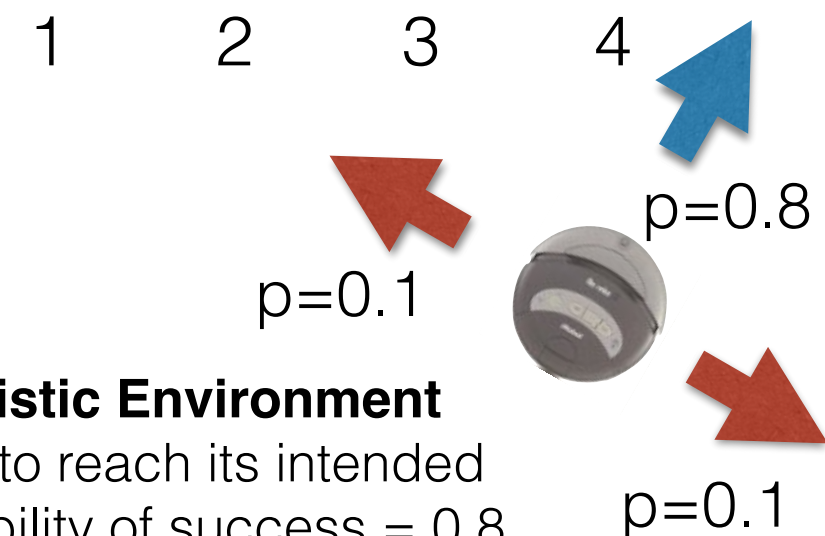
# Sequential Decision Making

## Returning to our earlier example

- Assume environment was **non-deterministic**
  - Optimal Solution is:
    - [Up, Up, Right, Right, Right]
  - Probability of reaching the goal if successful:
    - $p = 0.8 \times 5 = 0.32768$
  - Could also reach the goal accidentally by going the wrong way round:
    - $p = (0.1 \times 4) \times 0.8 = 0.00008$
  - Final probability of reaching the goal:  $p = 0.32776$
- Utility gained depends on the route taken
  - We will see later how to compute this...
    - Reinforcement Learning builds upon this type of model

$r = -0.04$  (unless stated otherwise)

1				$r = +1$ 👍
2				$r = -1$ 👎
3				



### Non-Deterministic Environment

Agent may fail to reach its intended cell (i.e. probability of success = 0.8, but may move sideways with  $p=0.1$  in each direction)

# Utilities over Runs

- Another possibility: assigns a utility not to individual states, but to runs themselves:

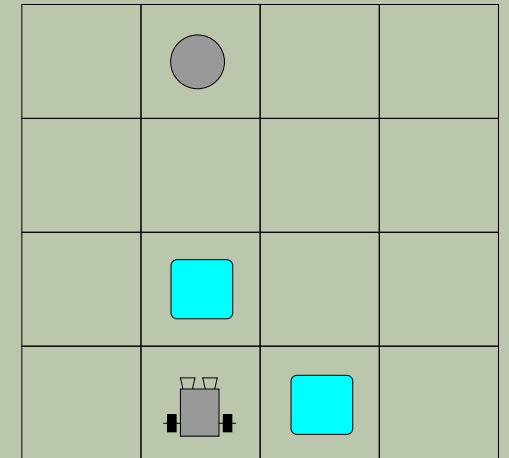
$$u : \mathcal{R} \rightarrow \mathbb{R}$$

- Such an approach takes an inherently **long term** view.
- Other variations:
  - incorporate probabilities of different states emerging.
- To see where utilities might come from, let's look at an example.

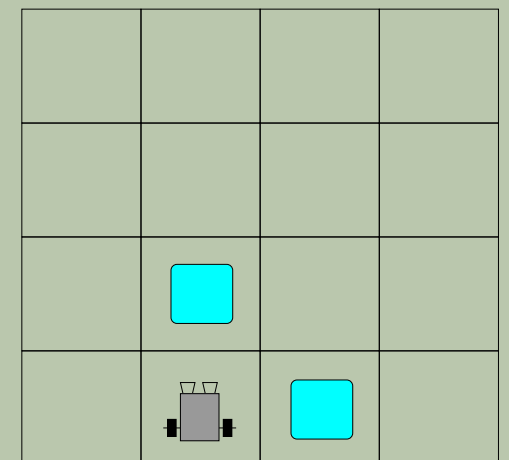
# Utility in the Tileworld

- Simulated two dimensional grid environment on which there are agents, tiles, obstacles, and holes.
- An agent can move in four directions, up, down, left, or right, and if it is located next to a tile, it can push it.
- Holes have to be filled up with tiles by the agent. An agent scores points by filling holes with tiles, with the aim being to fill as many holes as possible.
- TILEWORLD changes with the random appearance and disappearance of holes.

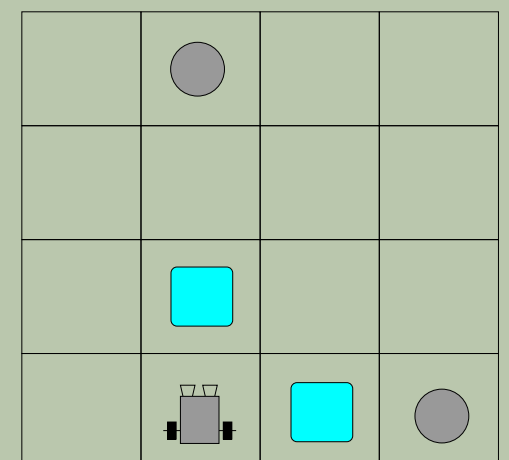
*The agent starts to push a tile towards the hole.*



*But then the hole disappears!!!*



*Later, a much more convenient hole appears (bottom right)*



# Utilities in the Tileworld

- Utilities are associated over runs, so that more holes filled is a higher utility.
- Utility function defined as follows:

$$u(r) \hat{=} \frac{\text{number of holes filled in } r}{\text{number of holes that appeared in } r}$$

- Thus:
  - if agent fills **all** holes, utility = 1.
  - if agent fills **no** holes, utility = 0.
- TILEWORLD captures the need for **reactivity** and for the advantages of exploiting opportunities.

# Expected Utility

- To denote probability that run  $r$  occurs when agent  $Ag$  is placed in environment  $Env$ , we can write:

$$P(r \mid Ag, Env)$$

- In a non-deterministic environment, for example, this can be computed from the probability of each step.

For a run  $r = (e_0, \alpha_0, e_1, \alpha_1, e_2, \dots)$ :

$$P(r \mid Ag, Env) = P(e_1, \mid e_0, \alpha_0)P(e_2 \mid e_1, \alpha_1) \dots$$

and clearly:

$$\sum_{r \in \mathcal{R}(Ag, Env)} P(r \mid Ag, Env) = 1.$$

# Expected Utility

- The expected utility ( $EU$ ) of agent  $Ag$  in environment  $Env$  (given  $P, u$ ), is then:

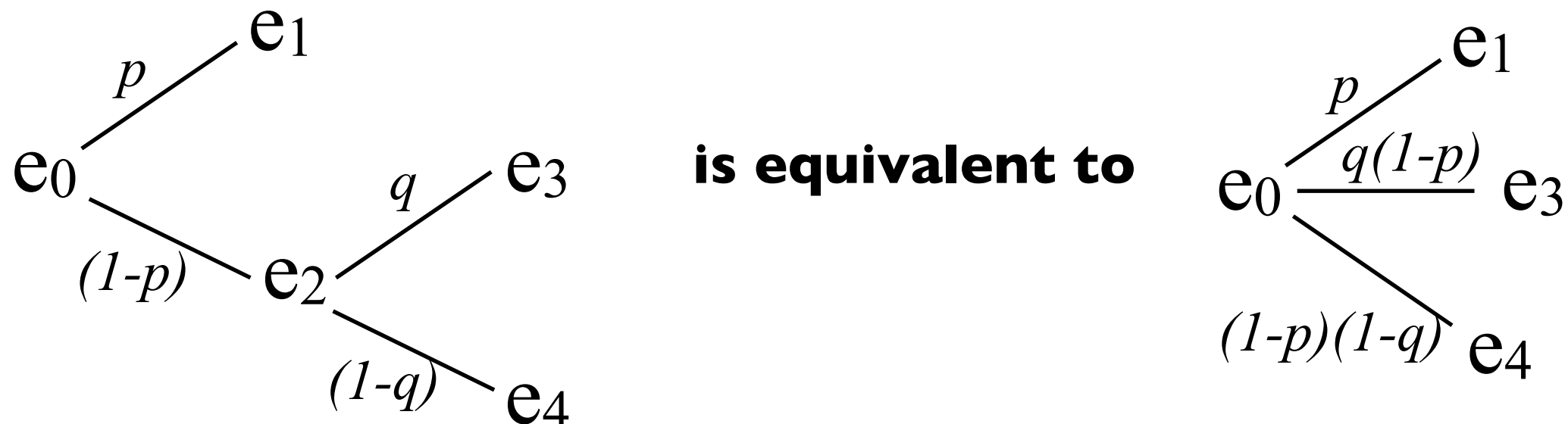
$$EU(Ag, Env) = \sum_{r \in \mathcal{R}(Ag, Env)} u(r)P(r \mid Ag, Env).$$

- That is, for each run we compute the utility and multiply it by the probability of the run.
- The expected utility is then the sum of all of these.

# Expected Utility

- The probability of a run can be determined from individual actions within a run
- Using the decomposability axiom from Utility Theory

*“... Compound lotteries can be reduced to simpler ones using the law of probability. Known as the “no fun in gambling” as two consecutive lotteries can be compressed into a single equivalent lottery...”*



# Optimal Agents

- The optimal agent  $Ag_{opt}$  in an environment  $Env$  is the one that maximizes expected utility:

$$Ag_{opt} = \arg \max_{Ag \in \mathcal{AG}} EU(Ag, Env)$$

- Of course, the fact that an agent is optimal does not mean that it **will** be best; only that **on average**, we can expect it to do best.



# Example I

Consider the environment  $Env_1 = \langle E, e_0, \tau \rangle$  defined as follows:

$$E = \{e_0, e_1, e_2, e_3, e_4, e_5\}$$

$$\tau(e_0 \xrightarrow{\alpha_0}) = \{e_1, e_2\}$$

$$\tau(e_0 \xrightarrow{\alpha_1}) = \{e_3, e_4, e_5\}$$

There are two agents possible with respect to this environment:

$$Ag_1(e_0) = \alpha_0$$

$$Ag_2(e_0) = \alpha_1$$

The probabilities of the various runs are as follows:

$$P(e_0 \xrightarrow{\alpha_0} e_1 \mid Ag_1, Env_1) = 0.4$$

$$P(e_0 \xrightarrow{\alpha_0} e_2 \mid Ag_1, Env_1) = 0.6$$

$$P(e_0 \xrightarrow{\alpha_1} e_3 \mid Ag_2, Env_1) = 0.1$$

$$P(e_0 \xrightarrow{\alpha_1} e_4 \mid Ag_2, Env_1) = 0.2$$

$$P(e_0 \xrightarrow{\alpha_1} e_5 \mid Ag_2, Env_1) = 0.7$$

Assume the utility function  $u_1$  is defined as follows:

$$u_1(e_0 \xrightarrow{\alpha_0} e_1) = 8$$

$$u_1(e_0 \xrightarrow{\alpha_0} e_2) = 11$$

$$u_1(e_0 \xrightarrow{\alpha_1} e_3) = 70$$

$$u_1(e_0 \xrightarrow{\alpha_1} e_4) = 9$$

$$u_1(e_0 \xrightarrow{\alpha_1} e_5) = 10$$

What are the expected utilities of the agents for this utility function?

# Example 1 Solution

Given the utility function  $u_1$  in the question, we have two transition functions defined as  $\tau(e_0 \xrightarrow{\alpha_0}) = \{e_1, e_2, e_3\}$ , and  $\tau(e_0 \xrightarrow{\alpha_1}) = \{e_4, e_5, e_6\}$ . The probabilities of the various runs (two for the first agent and three for the second) is given in the question, along with the probability of each run occurring. Given the definition of the utility function  $u_1$ , the *expected utilities* of agents  $Ag_0$  and  $Ag_1$  in environment  $Env$  can be calculated using:

$$EU(Ag, Env) = \sum_{r \in \mathcal{R}(Ag, Env)} u(r)P(r|Ag, Env).$$

This is equivalent to calculating the sum of the product of each utility for a run ending in some state with the probability of performing that run; i.e.

- Utility of  $Ag_0 = (0.4 \times 8) + (0.6 \times 11) = 9.8$
- Utility of  $Ag_1 = (0.1 \times 70) + (0.2 \times 9) + (0.7 \times 10) = 15.8$

Therefore agent  $Ag_1$  is optimal.

# Example 2

Consider the environment  $Env_1 = \langle E, e_0, \tau \rangle$  defined as follows:

$$E = \{e_0, e_1, e_2, e_3, e_4, e_5\}$$

$$\tau(e_0 \xrightarrow{\alpha_0}) = \{e_1, e_2\}$$

$$\tau(e_1 \xrightarrow{\alpha_1}) = \{e_3\}$$

$$\tau(e_2 \xrightarrow{\alpha_2}) = \{e_4, e_5\}$$

There are two agents,  $Ag_1$  and  $Ag_2$ , with respect to this environment:

$$\begin{array}{l|l} Ag_1(e_0) = \alpha_0 & Ag_2(e_0) = \alpha_0 \\ Ag_1(e_1) = \alpha_1 & Ag_2(e_2) = \alpha_2 \end{array}$$

The probabilities of the various runs are as follows:

$$P(e_0 \xrightarrow{\alpha_0} e_1 \mid Ag_1, Env_1) = 0.5$$

$$P(e_0 \xrightarrow{\alpha_0} e_2 \mid Ag_1, Env_1) = 0.5$$

$$P(e_1 \xrightarrow{\alpha_1} e_3 \mid Ag_1, Env_1) = 1.0$$

$$P(e_0 \xrightarrow{\alpha_0} e_1 \mid Ag_2, Env_1) = 0.1$$

$$P(e_0 \xrightarrow{\alpha_0} e_2 \mid Ag_2, Env_1) = 0.9$$

$$P(e_2 \xrightarrow{\alpha_2} e_4 \mid Ag_2, Env_1) = 0.4$$

$$P(e_2 \xrightarrow{\alpha_2} e_5 \mid Ag_2, Env_1) = 0.6$$

Assume the utility function  $u_1$  is defined as follows:

$$u_1(e_0 \xrightarrow{\alpha_0} e_1) = 4$$

$$u_1(e_0 \xrightarrow{\alpha_0} e_2) = 3$$

$$u_1(e_1 \xrightarrow{\alpha_1} e_3) = 7$$

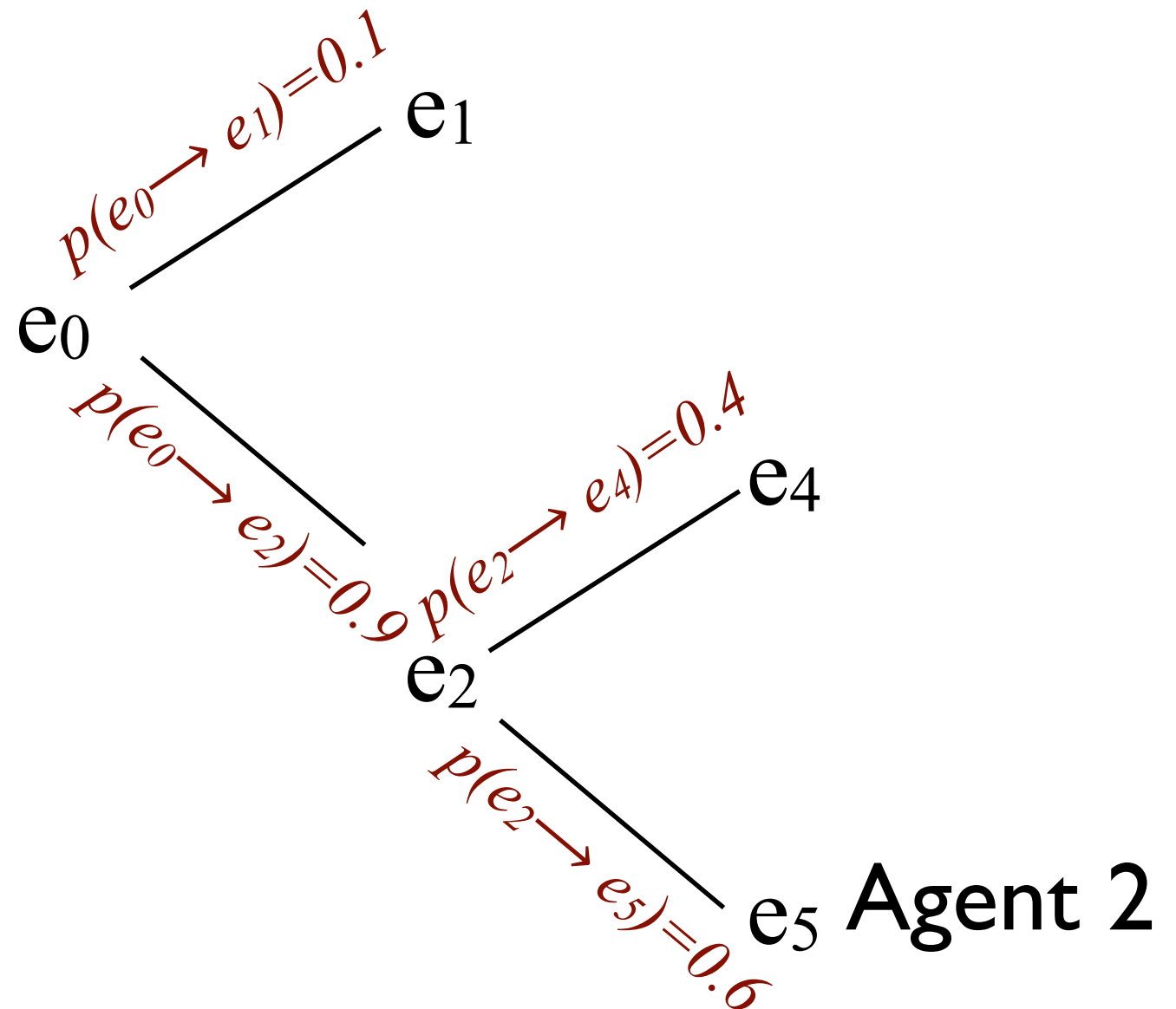
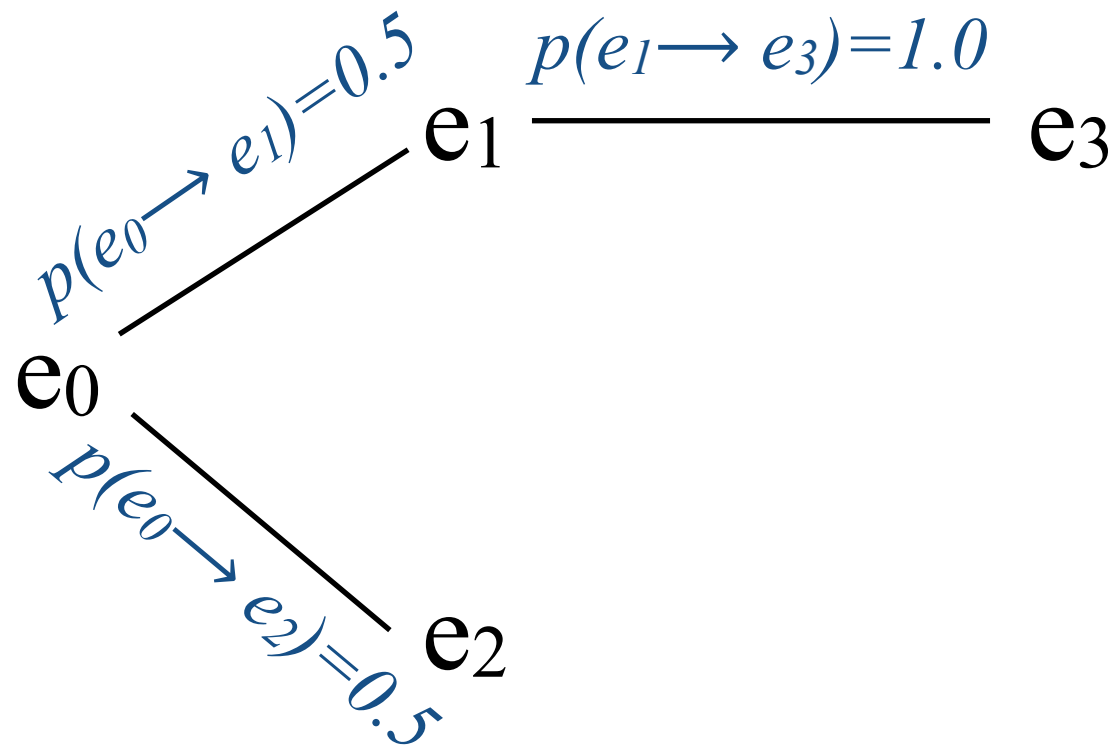
$$u_1(e_2 \xrightarrow{\alpha_2} e_4) = 3$$

$$u_1(e_2 \xrightarrow{\alpha_2} e_5) = 2$$

What are the expected utilities of the agents for this utility function?

# Example 2 solution

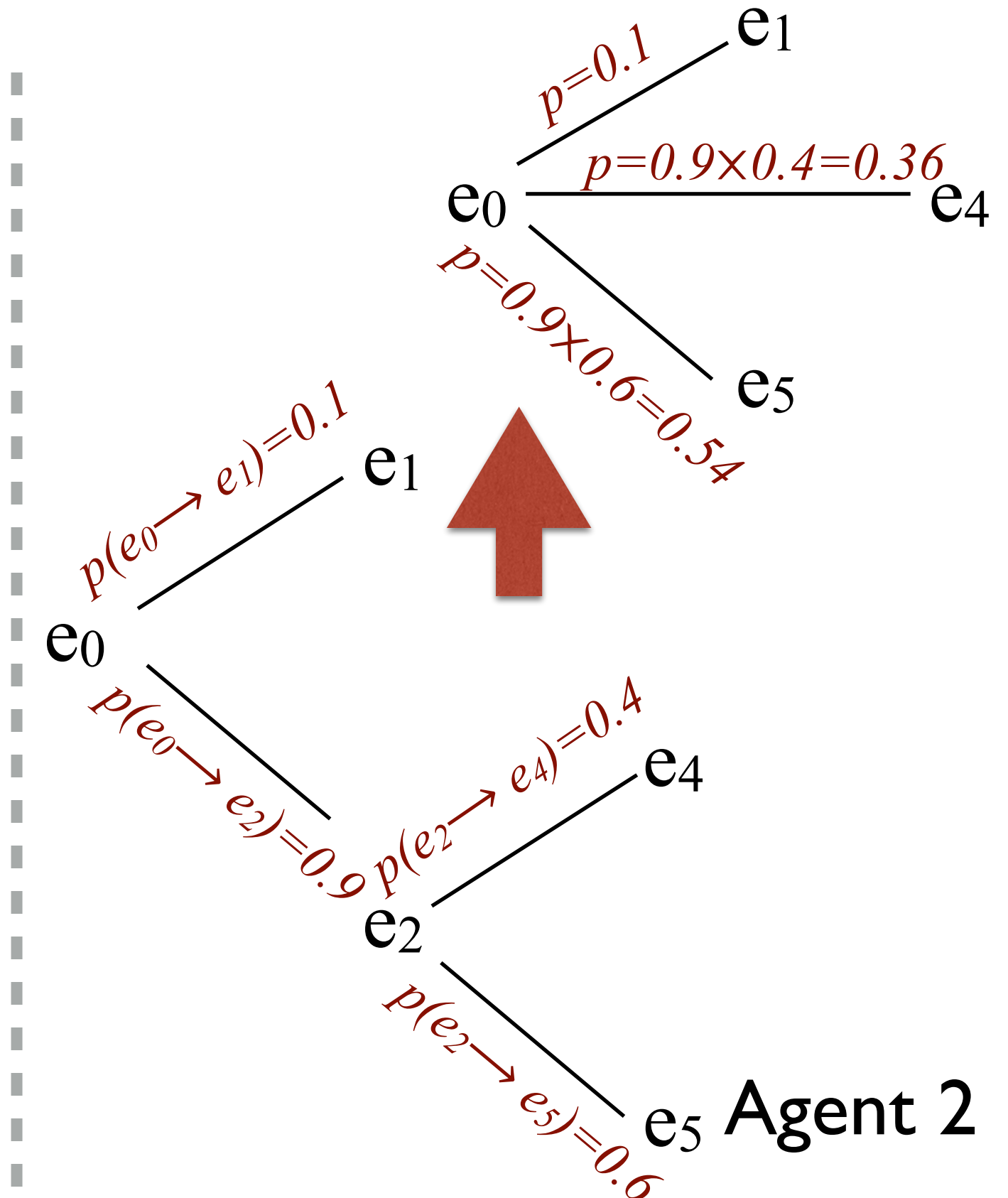
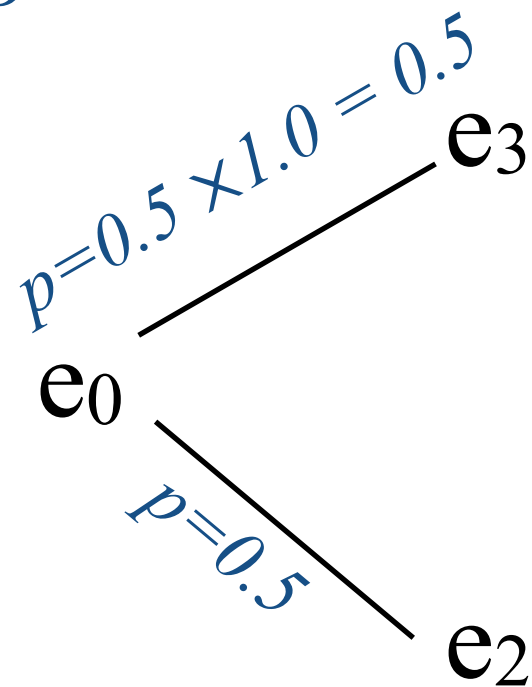
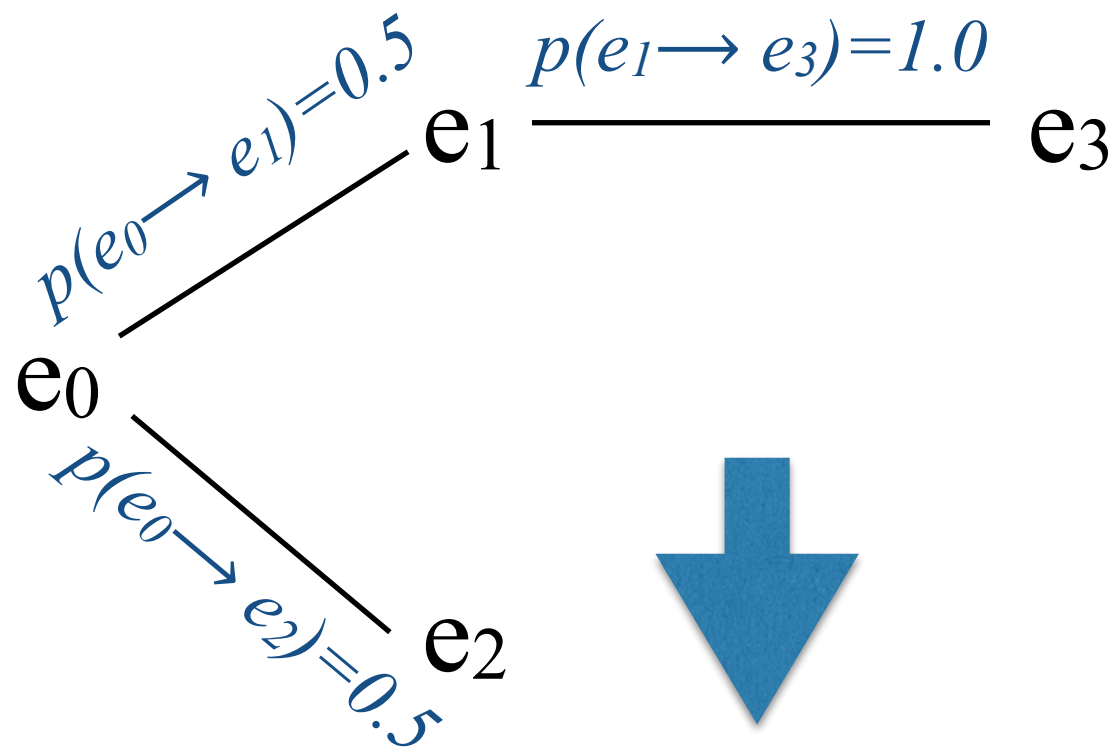
Agent 1



Agent 2

# Example 2 solution

Agent 1

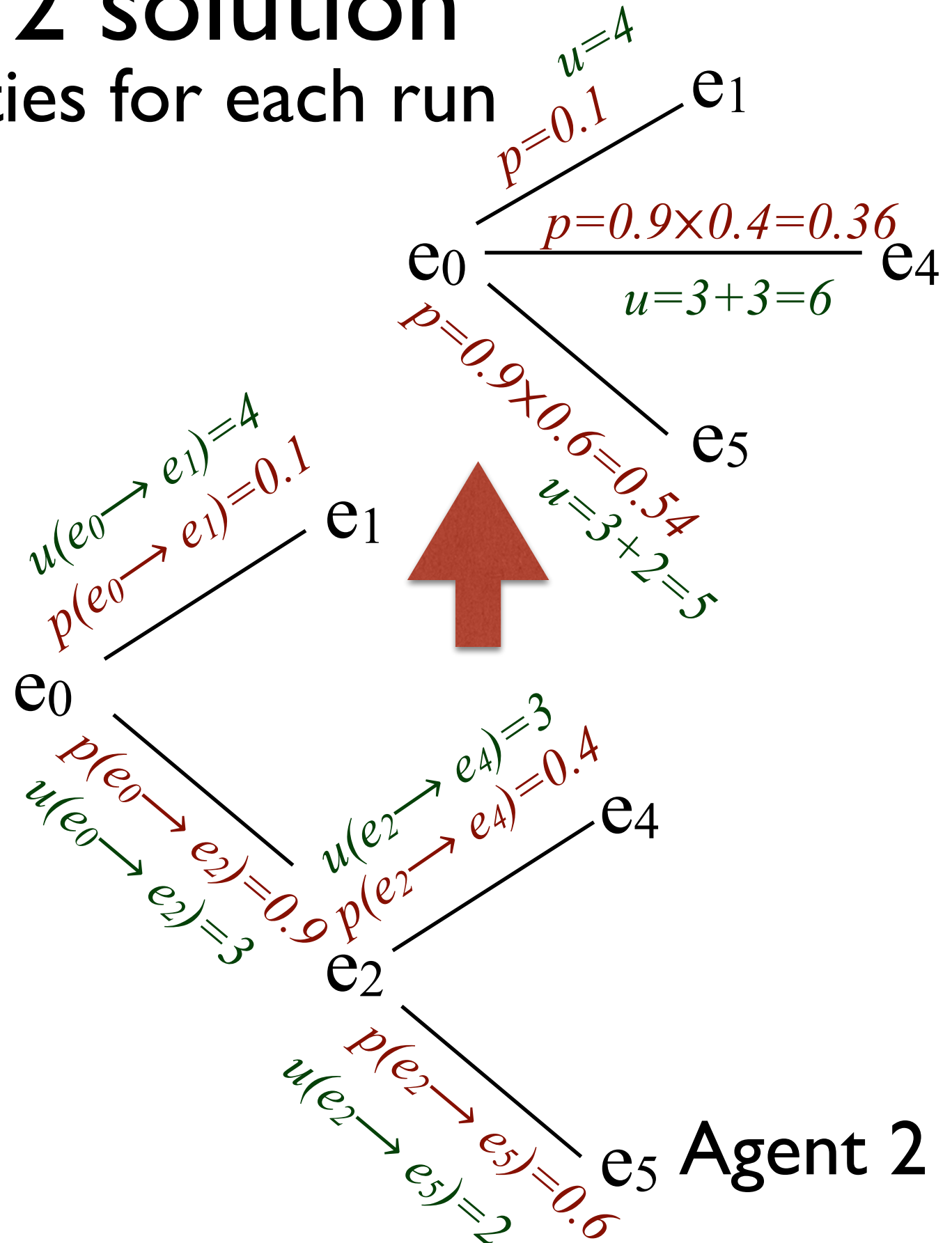
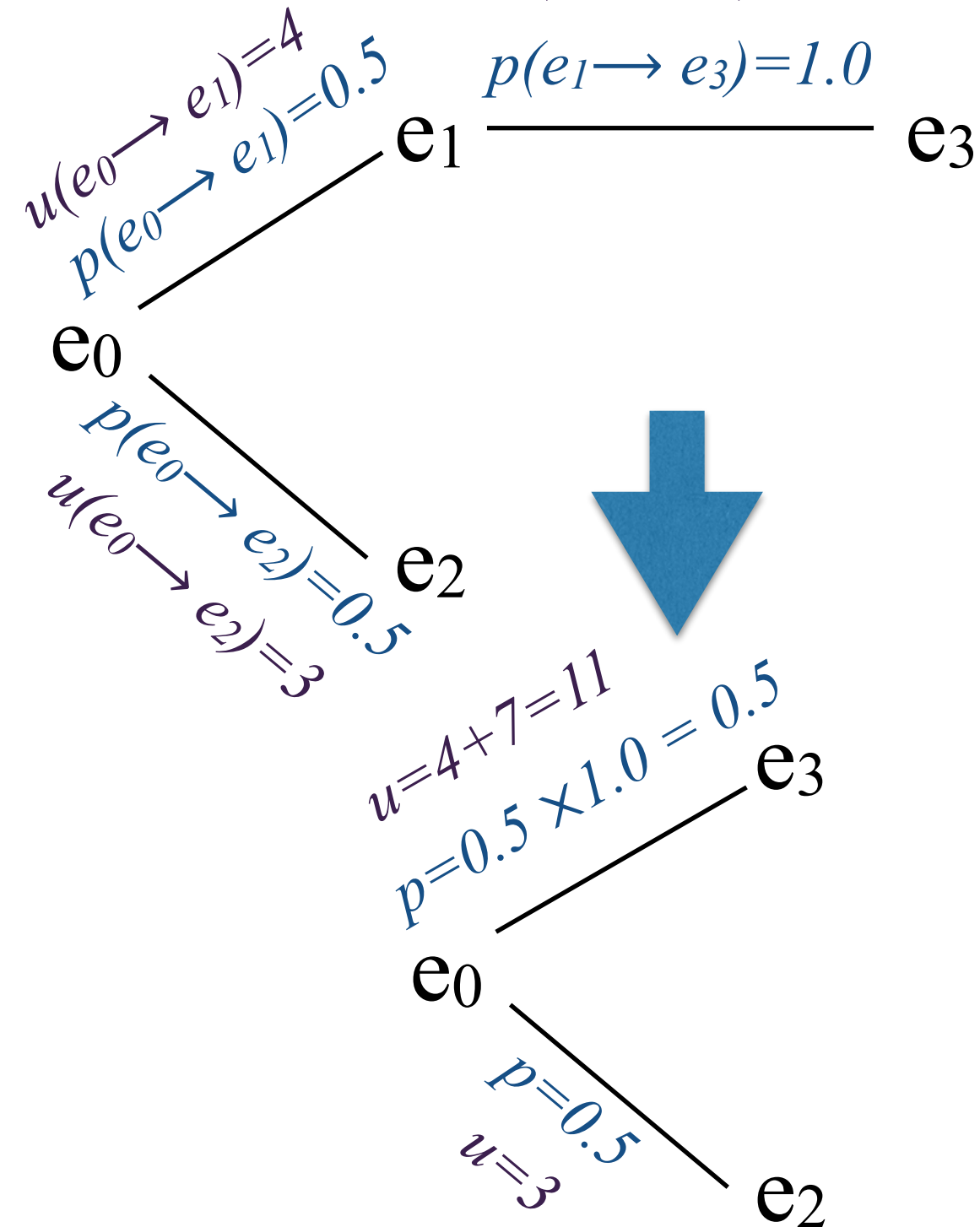


Agent 2

# Example 2 solution

Find sum of utilities for each run

Agent 1



Agent 2

# Example 2 solution

	Run	Utility	Probability
Agent 1	$e_0 \rightarrow e_3$	$u=11$	$p=0.5$
	$e_0 \rightarrow e_2$	$u=3$	$p=0.5$
Agent 2	$e_0 \rightarrow e_1$	$u=4$	$p=0.1$
	$e_0 \rightarrow e_4$	$u=6$	$p=0.36$
	$e_0 \rightarrow e_5$	$u=5$	$p=0.54$

$$Ag_1 = (11 \times 0.5) + (3 \times 0.5) = 5.5 + 1.5 = 7$$

$$\begin{aligned} Ag_2 &= (4 \times 0.1) + (6 \times 0.36) + (5 \times 0.54) \\ &= 0.4 + 2.16 + 2.7 = 5.26 \end{aligned}$$

# Bounded Optimal Agents

- Some agents cannot be implemented on some computers
  - The number of actions possible on an environment (and consequently the number of states) may be so big that it may need more than available memory to implement.
- We can therefore constrain our agent set to include only those agents that can be implemented on machine  $m$ :

$$\mathcal{AG}_m = \{Ag \mid Ag \in \mathcal{AG} \text{ and } Ag \text{ can be implemented on } m\}.$$

- The bounded optimal agent,  $Ag_{bopt}$ , with respect to  $m$  is then...

$$Ag_{bopt} = \arg \max_{Ag \in \mathcal{AG}_m} EU(Ag, Env)$$



# Predicate Task Specifications

- A special case of assigning utilities to histories is to assign 0 (false) or 1 (true) to a run.
  - If a run is assigned 1, then the agent **succeeds** on that run, otherwise it **fails**.
- Call these ***predicate task specifications***.
- Denote predicate task specification by  $\Psi$ :

$$\Psi : \mathcal{R} \rightarrow \{0, 1\}$$

# Task Environments

- A task environment is a pair  $\langle Env, \Psi \rangle$ , where  $Env$  is an environment, and the task specification  $\Psi$  is defined by:

$$\Psi : \mathcal{R} \rightarrow \{0, 1\}$$

- Let the set of all task environments be defined by:

$$\mathcal{TE}$$

- A task environment specifies:
  - the properties of the system the agent will inhabit;
  - the criteria by which an agent will be judged to have either failed or succeeded.

# Task Environments

- To denote set of all runs of the agent  $Ag$  in environment  $Env$  that satisfy  $\Psi$ , we write:

$$\mathcal{R}_{\Psi}(Ag, Env) = \{r \mid r \in \mathcal{R}(Ag, Env) \text{ and } \Psi(r) = 1\}.$$

- We then say that an agent  $Ag$  succeeds in task environment  $\langle Env, \Psi \rangle$  if

$$\mathcal{R}_{\Psi}(Ag, Env) = \mathcal{R}(Ag, Env)$$

- In other words, an agent succeeds if every run satisfies the specification of the agent.

We could also write this as:

$\forall r \in \mathcal{R}(Ag, Env), \text{ we have } \Psi(r) = 1$

**However**, this is a bit **pessimistic**: if the agent fails on a single run, we say it has failed overall.

A more **optimistic** idea of success is:

$\exists r \in \mathcal{R}(Ag, Env), \text{ we have } \Psi(r) = 1$   
which counts an agent as successful as soon as it completes a single successful run.

# The Probability of Success

- If the environment is non-deterministic, the  $\tau$  returns a **set** of possible states.
- We can define a probability distribution across the set of states.
- Let  $P(r \mid Ag, Env)$  denote probability that run  $r$  occurs if agent  $Ag$  is placed in environment  $Env$ .
- Then the probability  $P(\Psi \mid Ag, Env)$  that  $\Psi$  is satisfied by  $Ag$  in  $Env$  would then simply be:

$$P(\Psi \mid Ag, Env) = \sum_{r \in \mathcal{R}_{\Psi}(Ag, Env)} P(r \mid Ag, Env)$$

# Achievement and Maintenance Tasks

- The idea of a predicate task specification is admittedly abstract.
- It generalises two common types of tasks: **achievement tasks** and **maintenance tasks**:
  1. **Achievement tasks** Are those of the form “achieve state of affairs  $\varphi$ ”.
  2. **Maintenance tasks** Are those of the form “maintain state of affairs  $\psi$ ”.

# Achievement and Maintenance Tasks

- An **achievement task** is specified by a set  $G$  of “good” or “goal” states:  $G \subseteq E$ .
  - The agent succeeds if it is guaranteed to bring about at least one of these states (we don’t care which, as all are considered good).
  - The agent **succeeds** if it can **force the environment** into one of the goal states  $g \in G$ .
- A **maintenance goal** is specified by a set  $B$  of “bad” states:  $B \subseteq E$ .
  - The agent succeeds in a particular environment if it manages to avoid all states in  $B$  — if it never performs actions which result in any state in  $B$  occurring.
  - In terms of games, the agent **succeeds** in a maintenance task if it ensures that it is **never forced into** one of the fail states  $b \in B$ .

# Summary

- This chapter has looked in detail at what constitutes an intelligent agent.
  - We looked at the properties of an intelligent agent and the properties of the environments in which it may operate.
  - We introduced the intentional stance and discussed its use.
  - We looked at abstract architectures for agents of different kinds; and
  - Finally we discussed what kinds of task an agent might need to carry out.
- In the next chapter, we will start to look at how one might program an agent using deductive reasoning.

## *Class Reading (Chapter 2):*

*"Is it an Agent, or Just a Program?: A Taxonomy for Autonomous Agents", Stan Franklin and Art Graesser. ECAI '96 Proceedings of the Workshop on Intelligent Agents III, Agent Theories, Architectures, and Languages. pp 21-35*

*This paper informally discusses various different notions of agency. The focus of the discussion might be on a comparison with the discussions in this chapter*