COMP329 Robotics & Autonomous Systems

Lectures 2/3

Dr Terry R. Payne
Department of Computer Science

UNIVERSITY OF LIVERPOOL

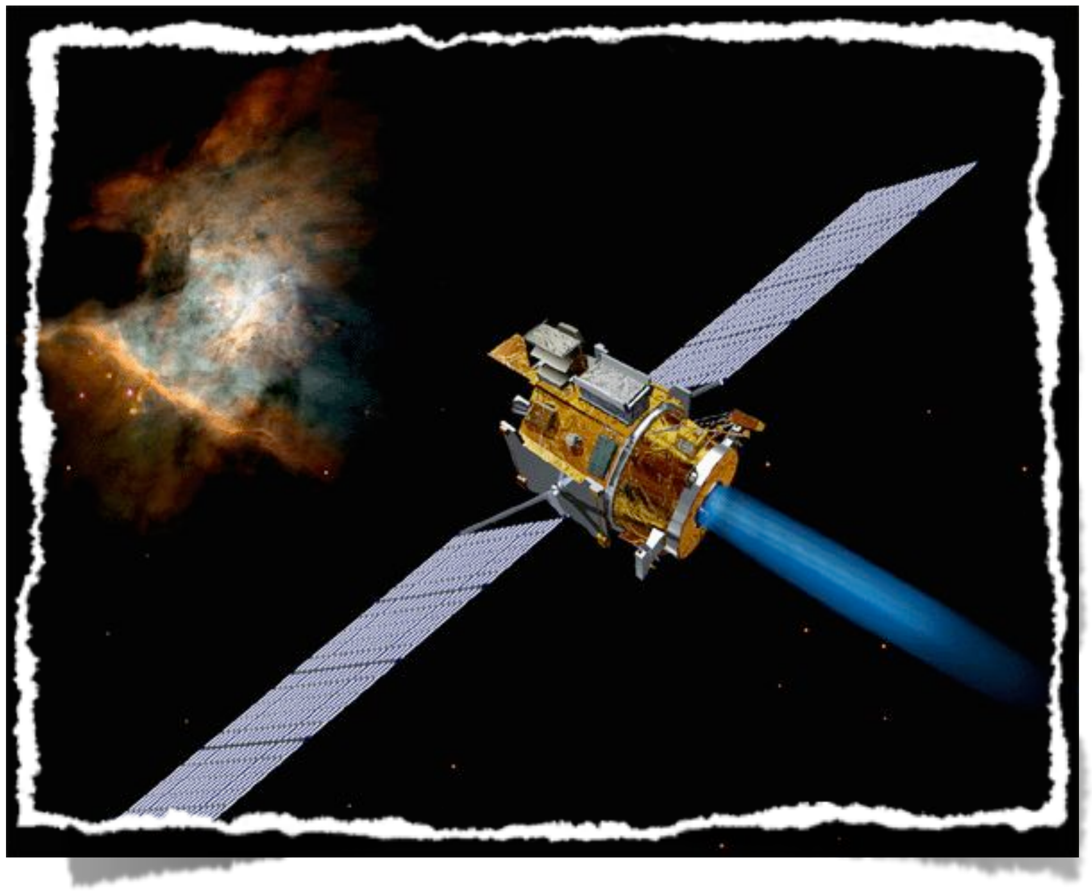
Autonomous Agents

- An agent is…
  - …a computer system that is capable of independent (autonomous) action on behalf of its user or owner (figuring out what needs to be done to satisfy design objectives, rather than constantly being told).

- Systems like Deep Space 1 and the Autonomous Asteroid Exploration Project show that it is possible to do this!

A Vision: Autonomous Space Probes

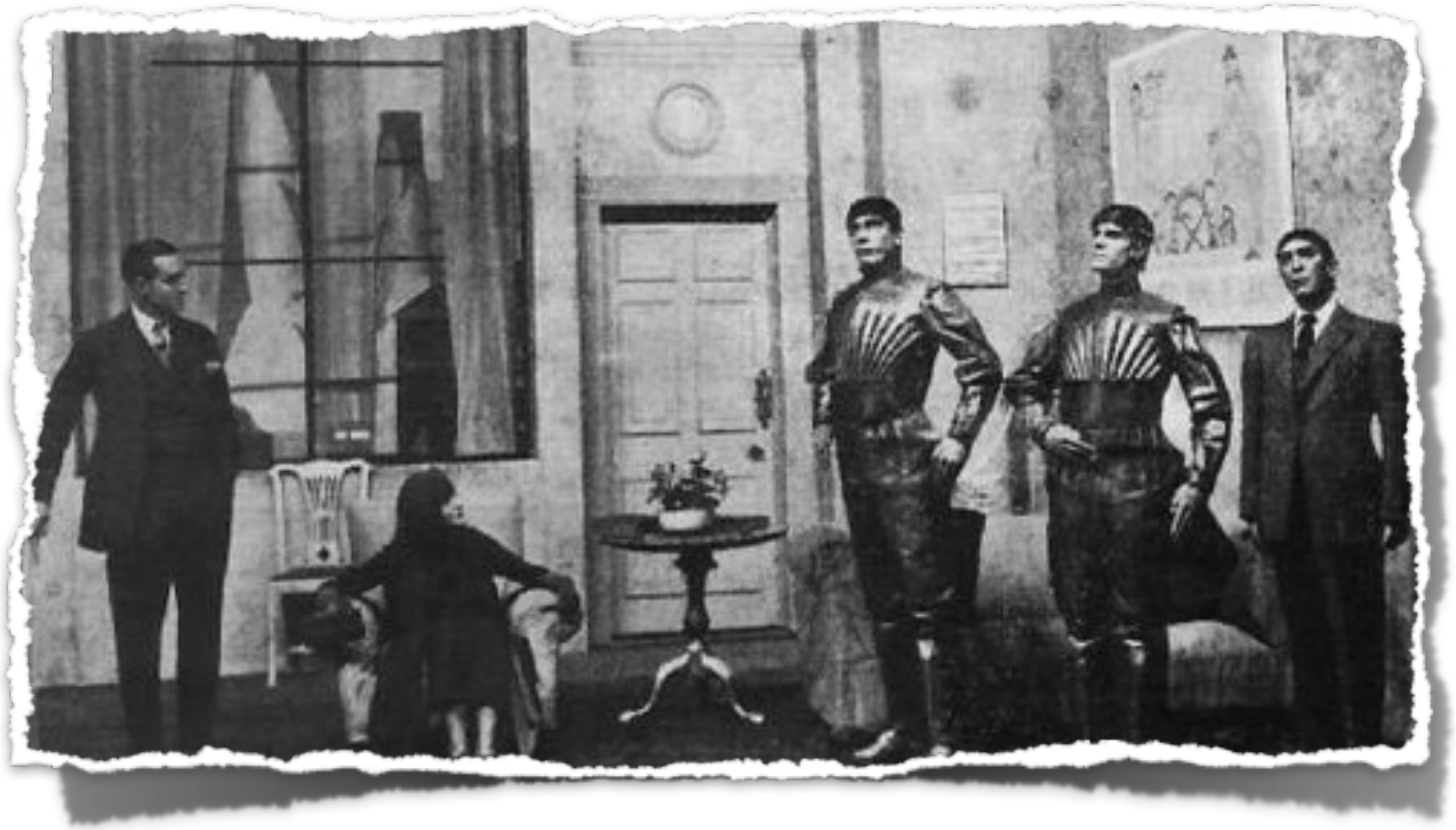
- When a space probe makes its long flight from Earth to the outer planets, a ground crew is usually required to continually track its progress, and decide how to deal with unexpected eventualities.
  - This is costly and, if decisions are required ***quickly***, it is simply not practicable.
  - For these reasons, organisations like NASA are seriously investigating the possibility of making probes more autonomous — giving them richer decision making capabilities and responsibilities.

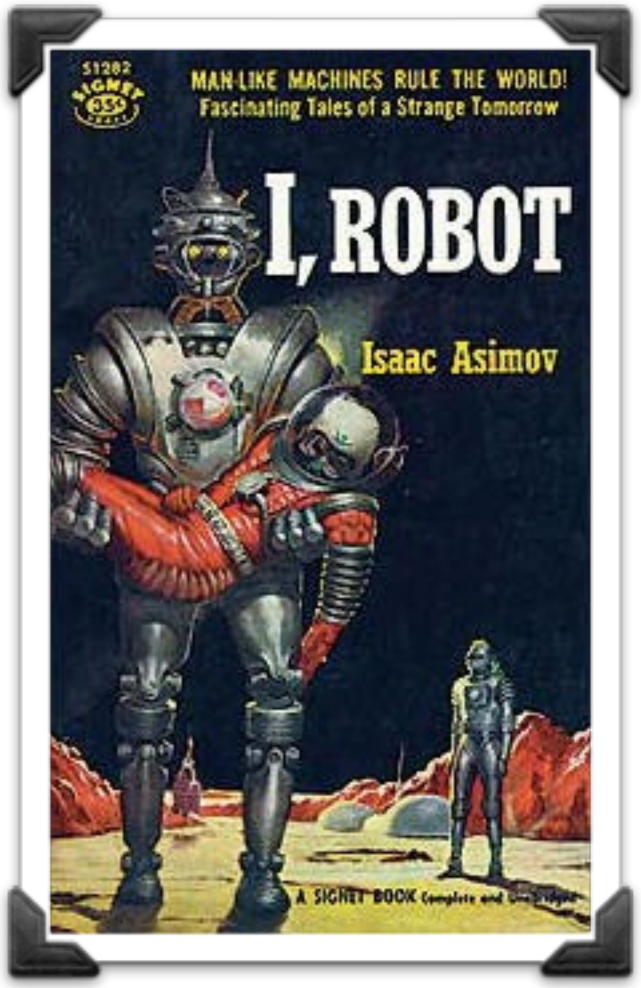


- *This is not fiction: NASA's DS1 did this years ago!*

So what are Robots?

The word "robot" was first used in Karel Capek's play "Rossum's Universal Robots" in 1920




MAN-LIKE MACHINES RULE THE WORLD!
Fascinating Tales of a Strange Tomorrow
I, ROBOT
Isaac Asimov
A SIGNET BOOK

Isaac Asimov coined the term "robotics" in 1942.

Original Source: M. Wooldridge, S.Parsons, D.Grossi - updated by Terry Payne, Autumn 2016

So what are Robots?

"…a programmable, multifunction manipulator designed to move material, parts, tools, or specialized devices through variable programmed motions for the performance of a variety of tasks…"

Robot Institute of America (1980)

"…[a] physical agent that performs tasks by manipulating the physical world…" Russell and Norvig (2003).

Robots, Teleoperation and Autonomy

- Many autonomous vehicles are not really autonomous
  - They are teleoperated.

- Autonomous Robots make their **own** decisions

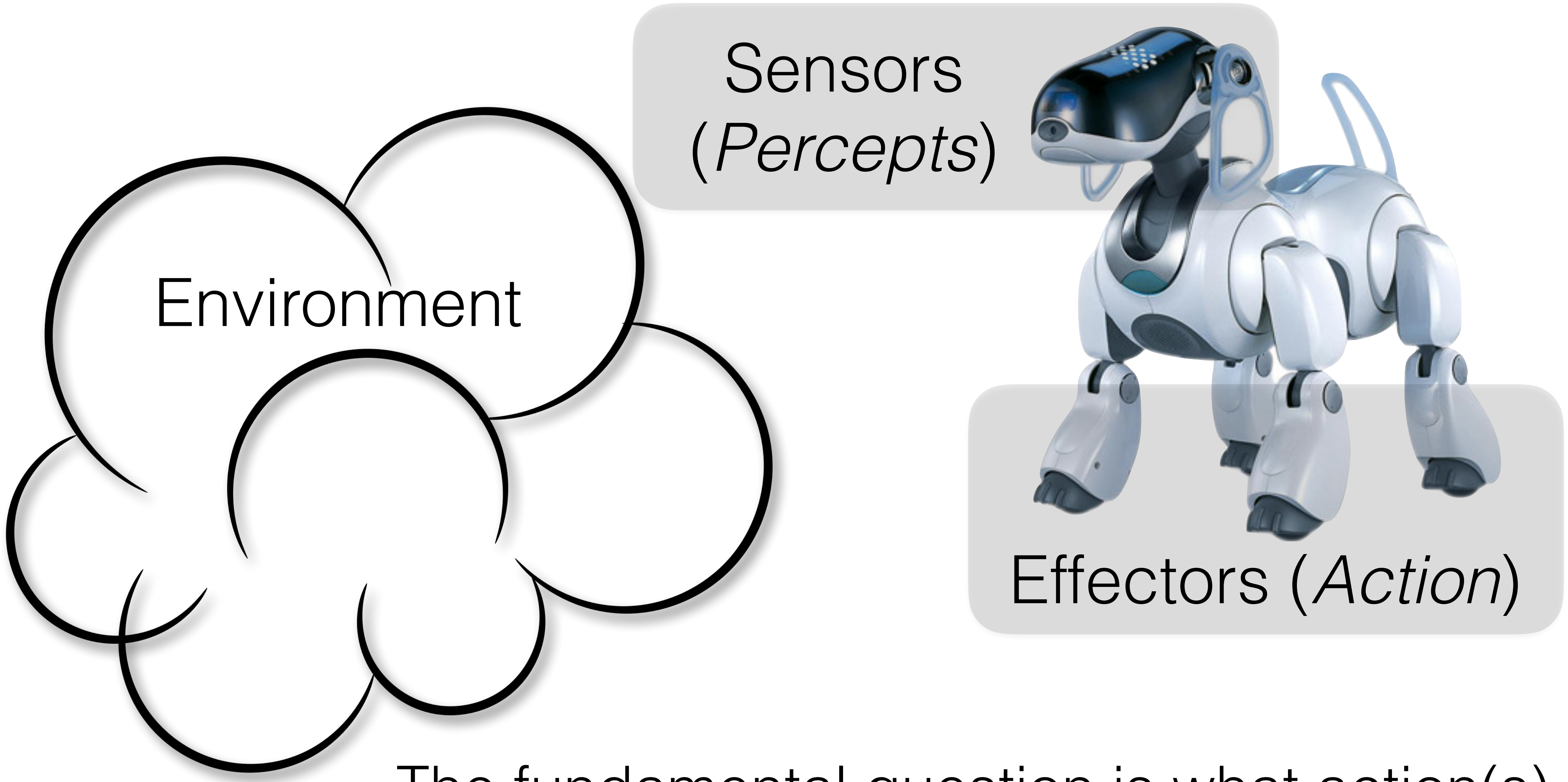
What is an Agent?

- 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.

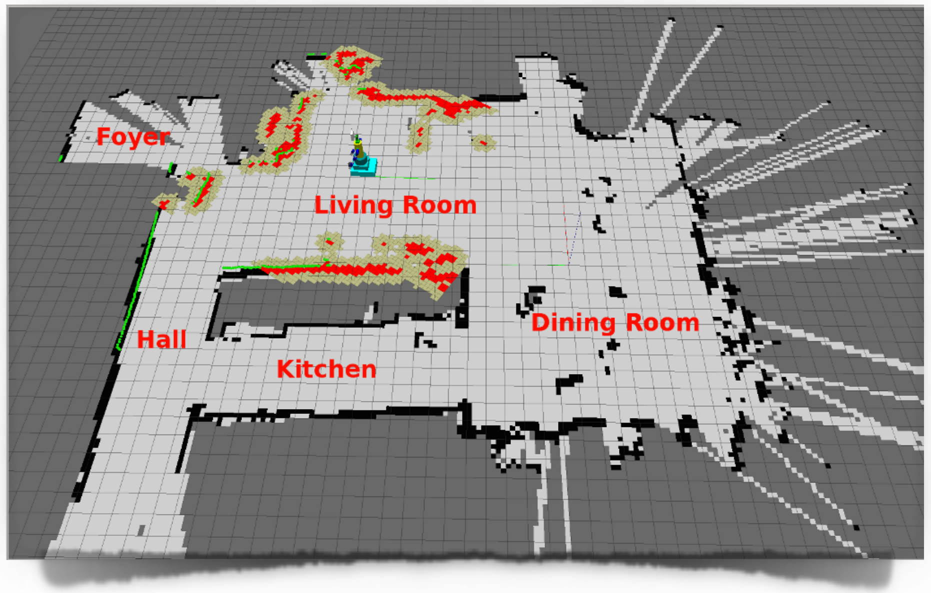
Agent and Environment

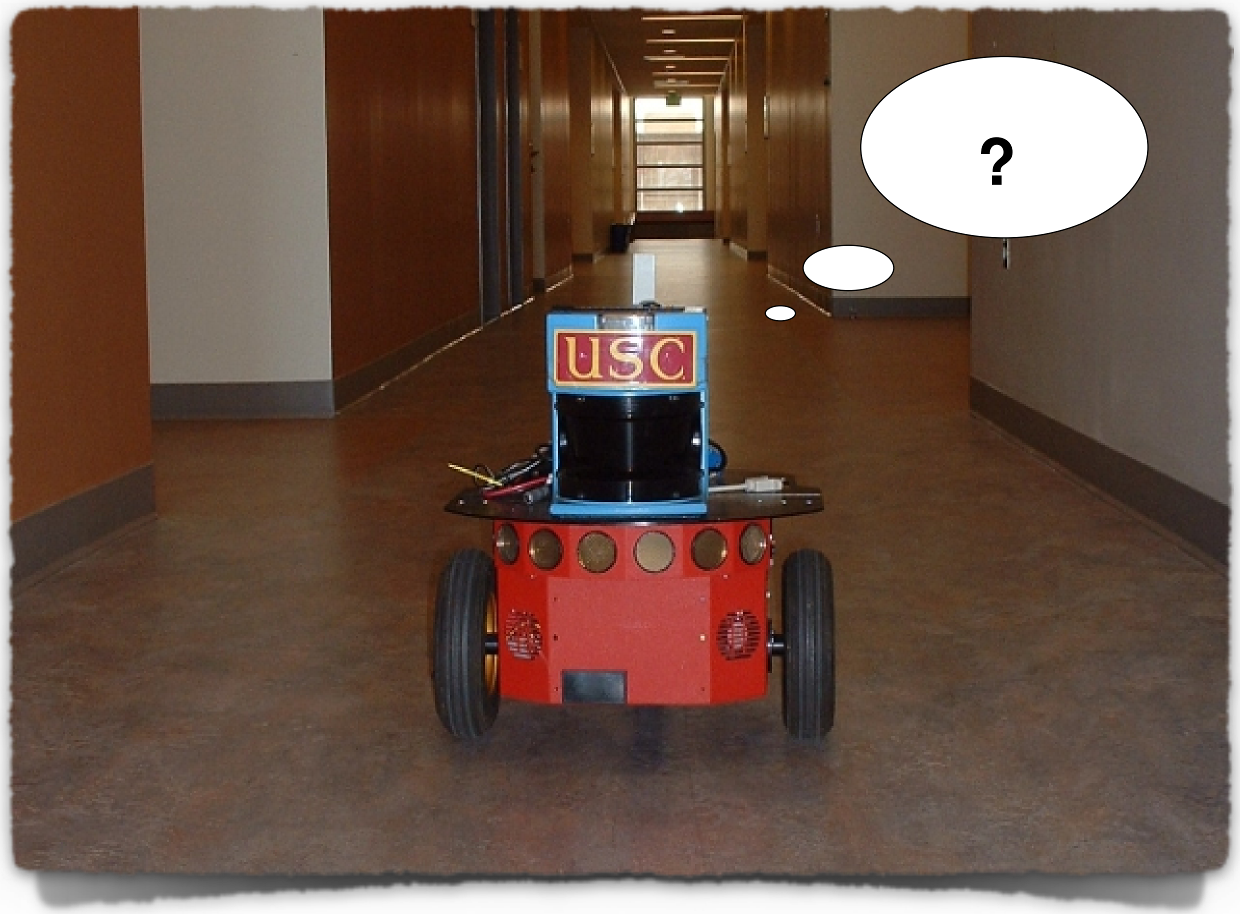


The fundamental question is what action(s) to take for a given state of the environment

What is mobile robotics?

- In mobile robotics this becomes three questions:



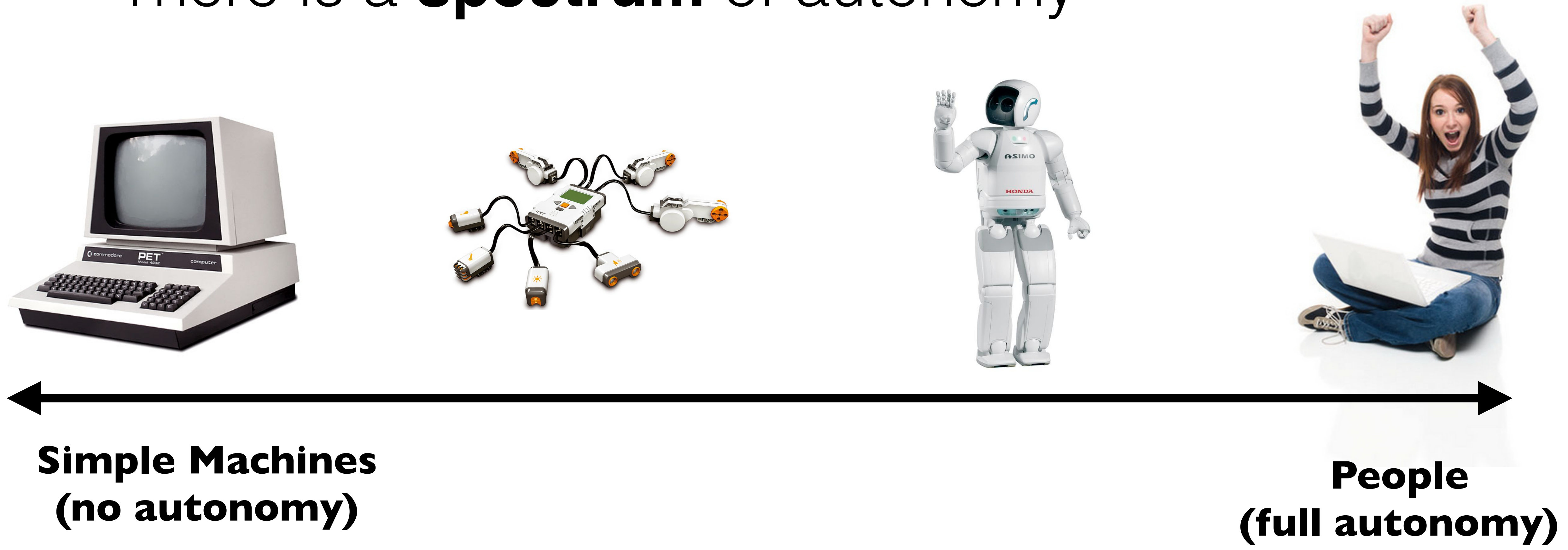


  - Where am I ?
  - Where am I going ?
  - How do I get there ?

- The robotics part of this course is about answering those questions.

Autonomy

- There is a **spectrum** of 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 as 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. . .

Abstract Architectures for Agents

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

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

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

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

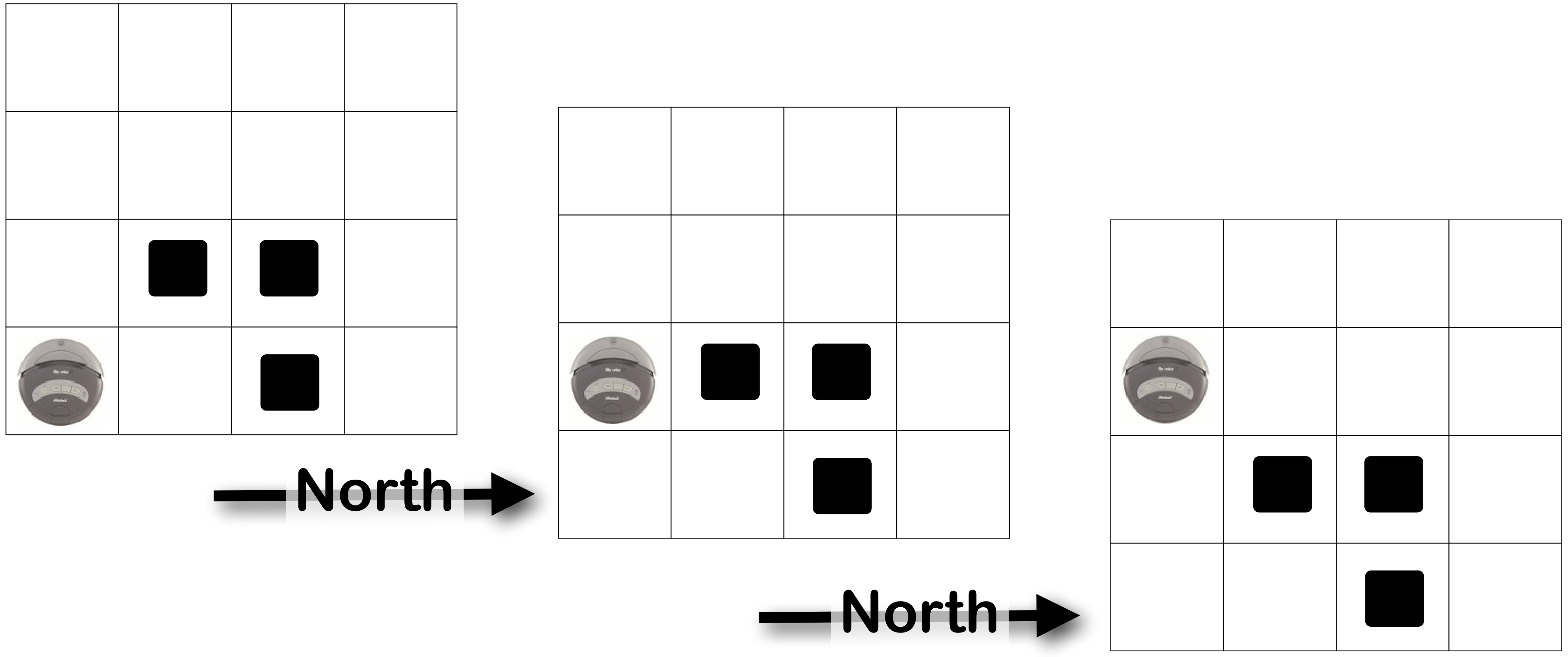
  - Actions can be non-deterministic, but only one state ever results from and action.

- A ***run***, r, of an agent in an environment is a sequence of interleaved world states and actions:

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

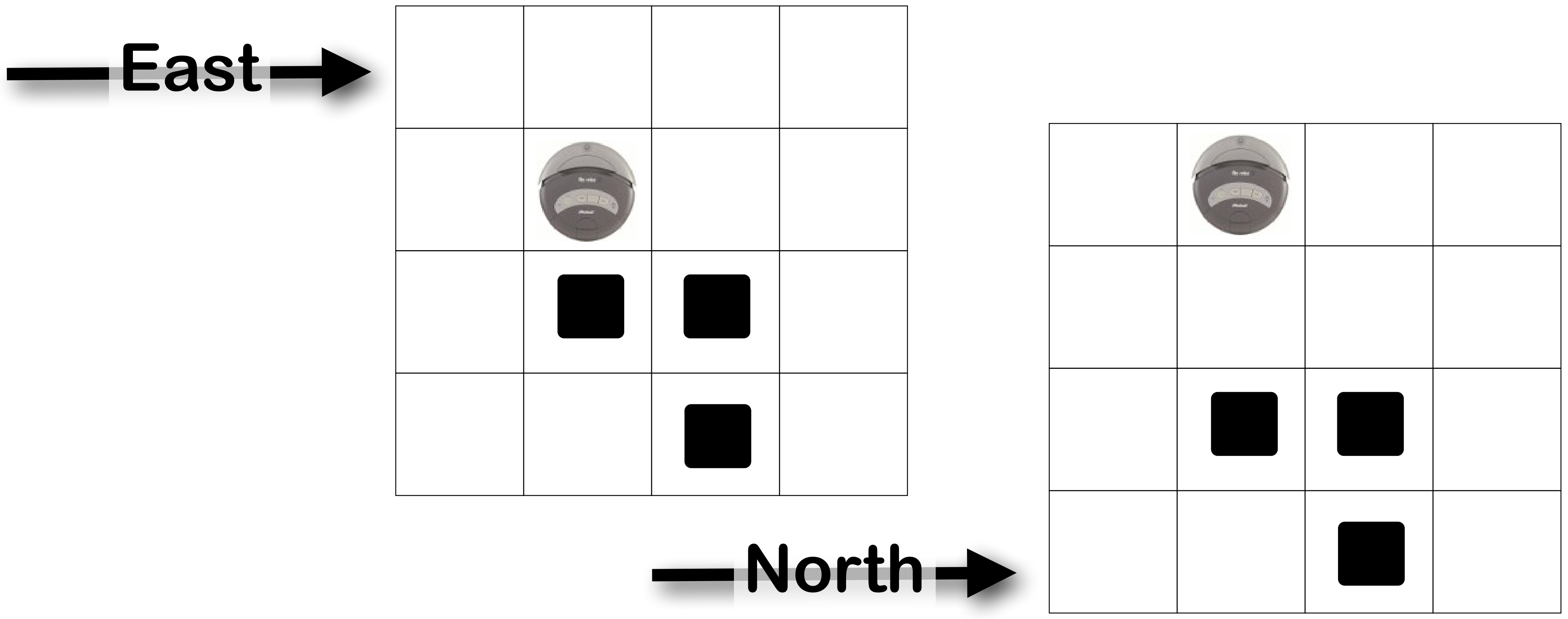
Abstract Architectures for Agents (1)

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

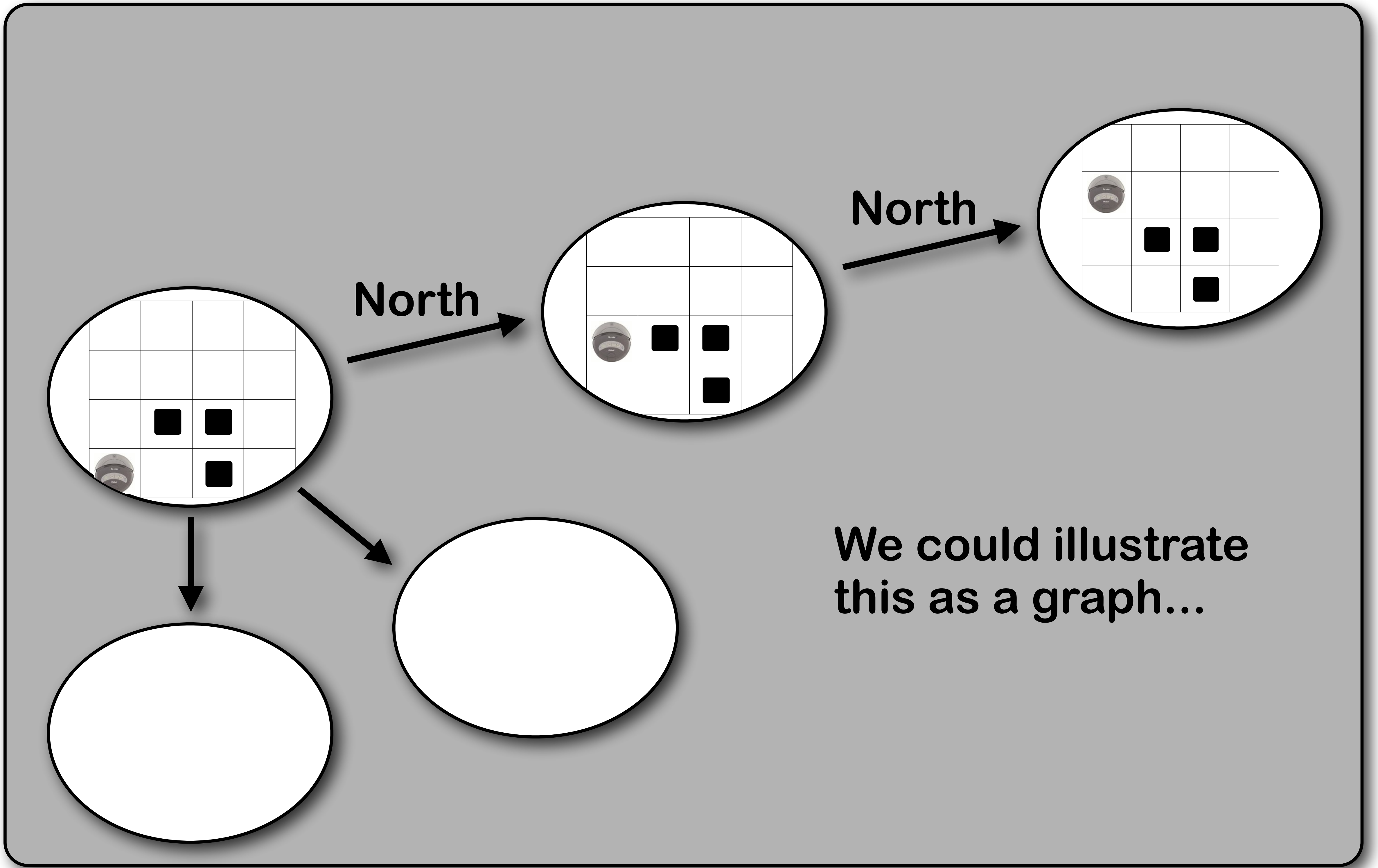


Abstract Architectures for Agents (2)

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

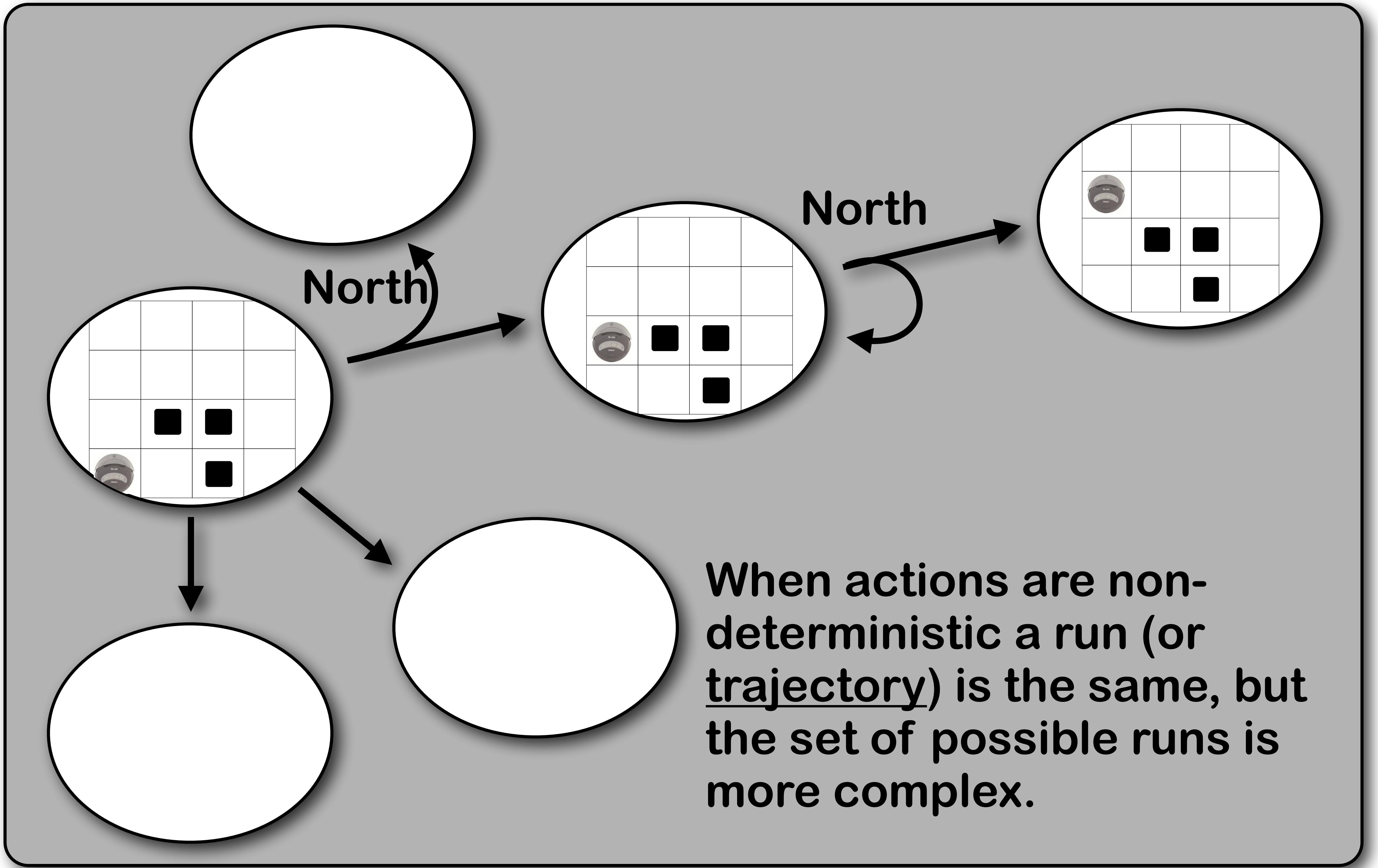


Abstract Architectures for Agents



We could illustrate this as a graph...

Abstract Architectures for Agents



When actions are non-deterministic a run (or trajectory) is the same, but the set of possible runs is more complex.

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',... 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:

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

- 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 τ is state transformer function.

Agents

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

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

- 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

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

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 $u > 0$,

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

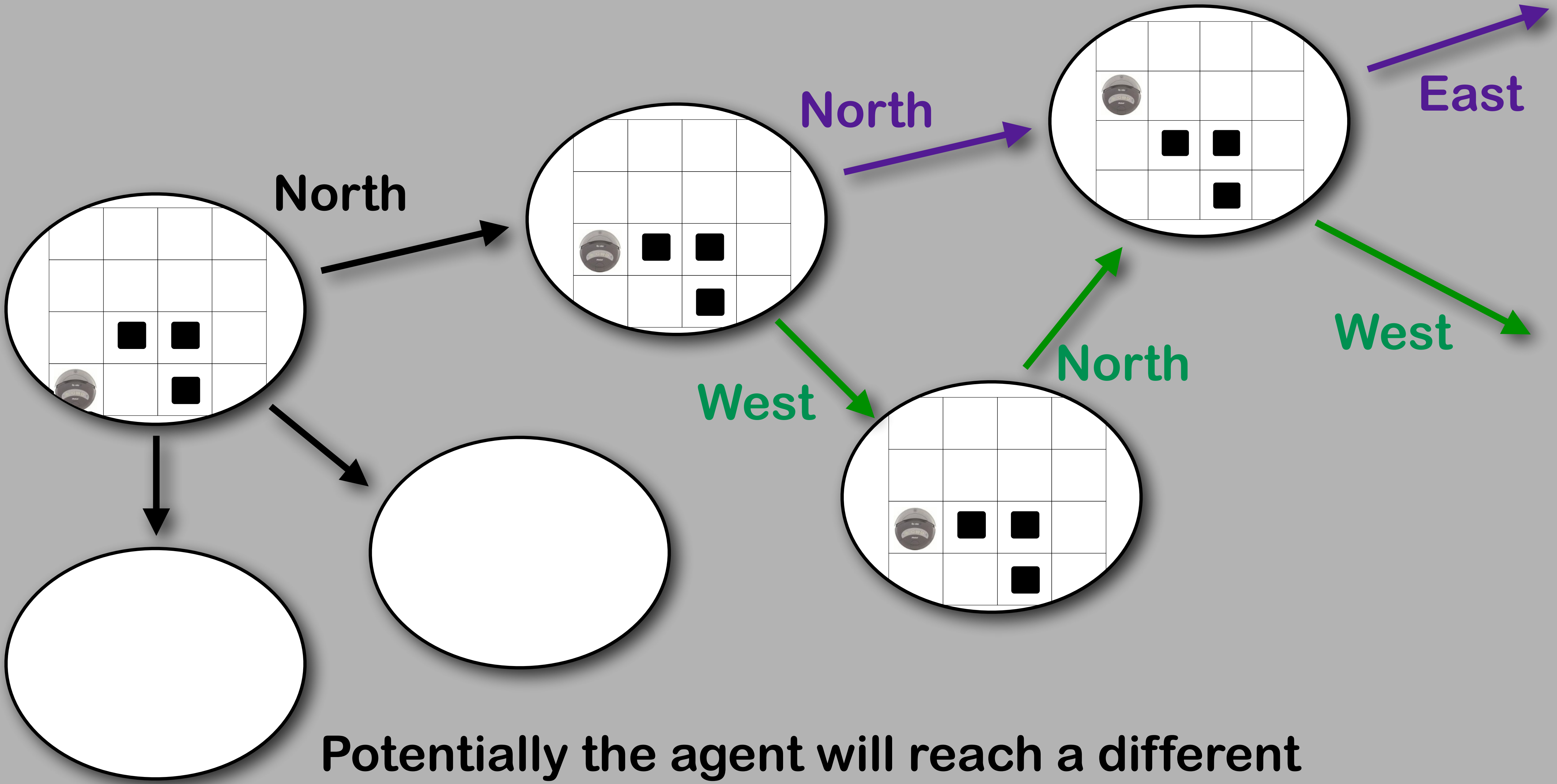
Why the notation?

- 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

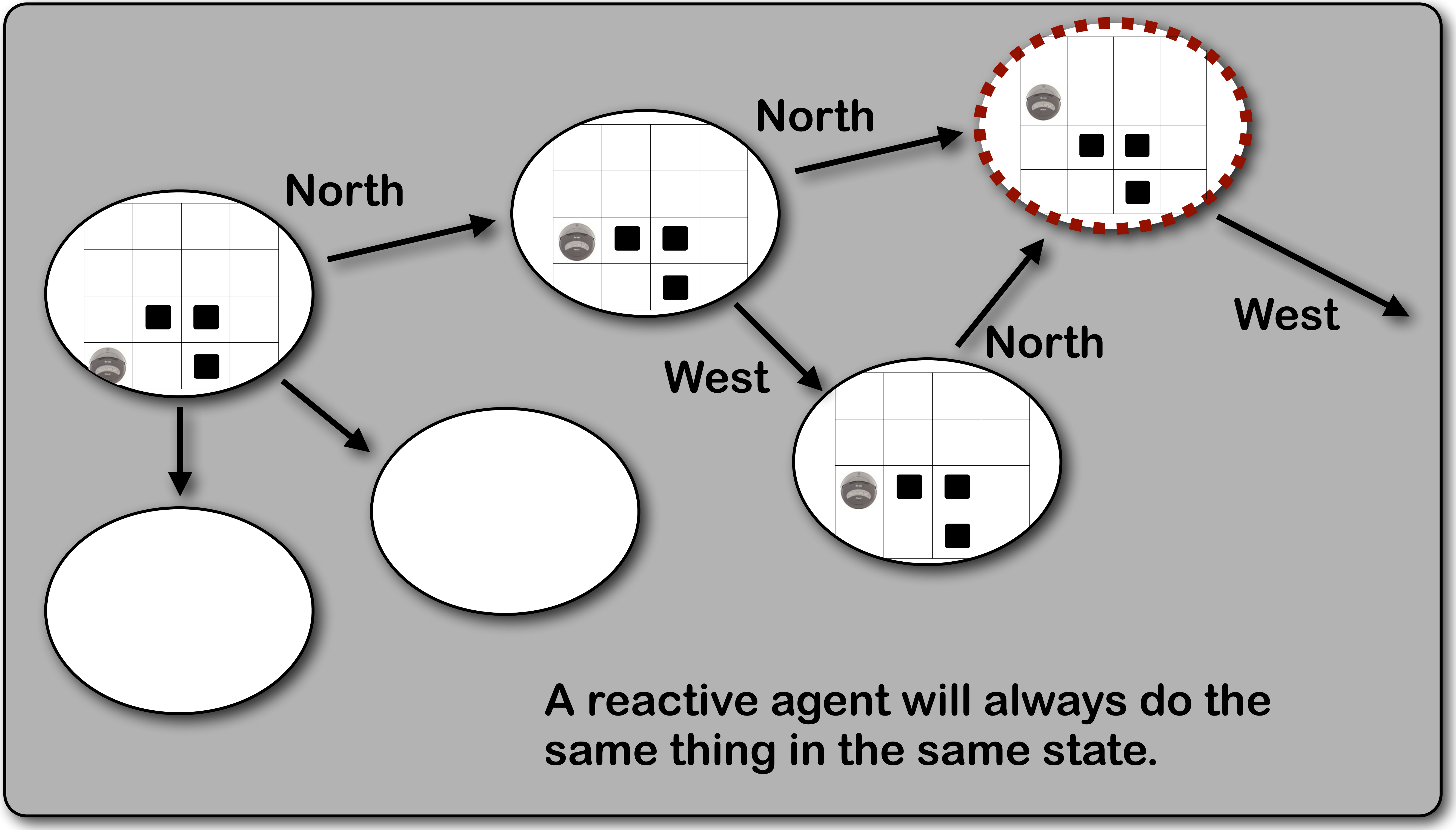
- Some agents decide what to do without reference to their history — they base their decision making entirely on the present, with no reference at all to the past.
- We call such agents ***purely reactive***:

$$action : E \to Ac$$

- A thermostat is a purely reactive agent.

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

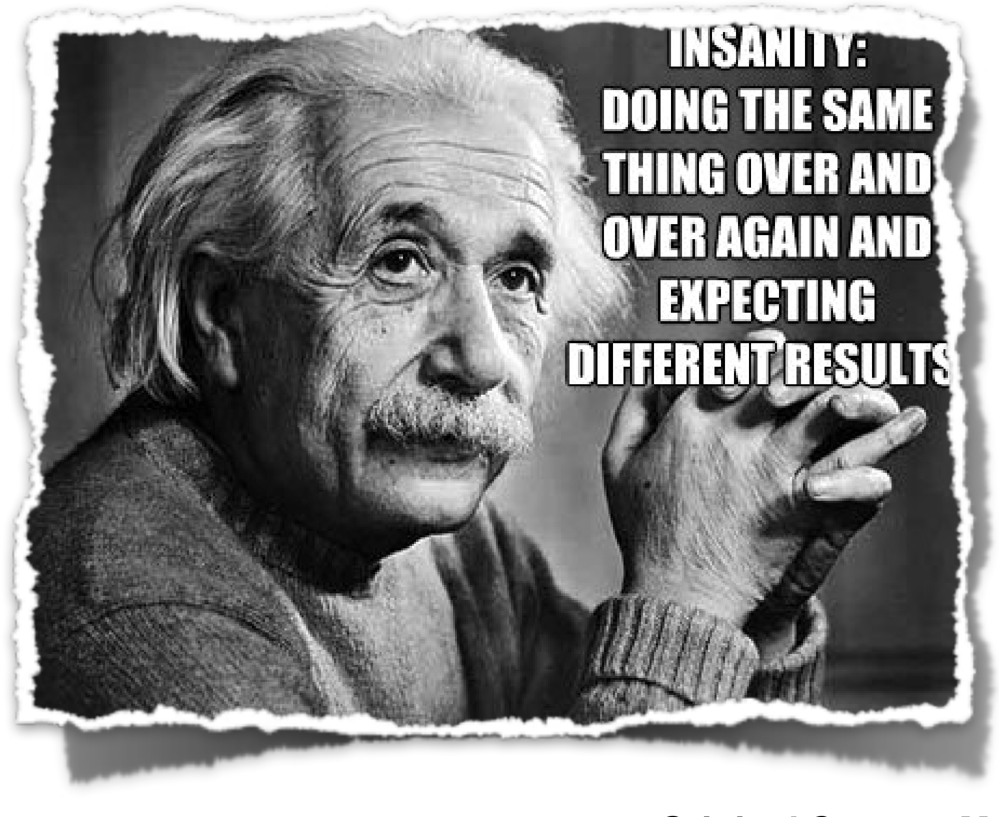
Reactive Agents



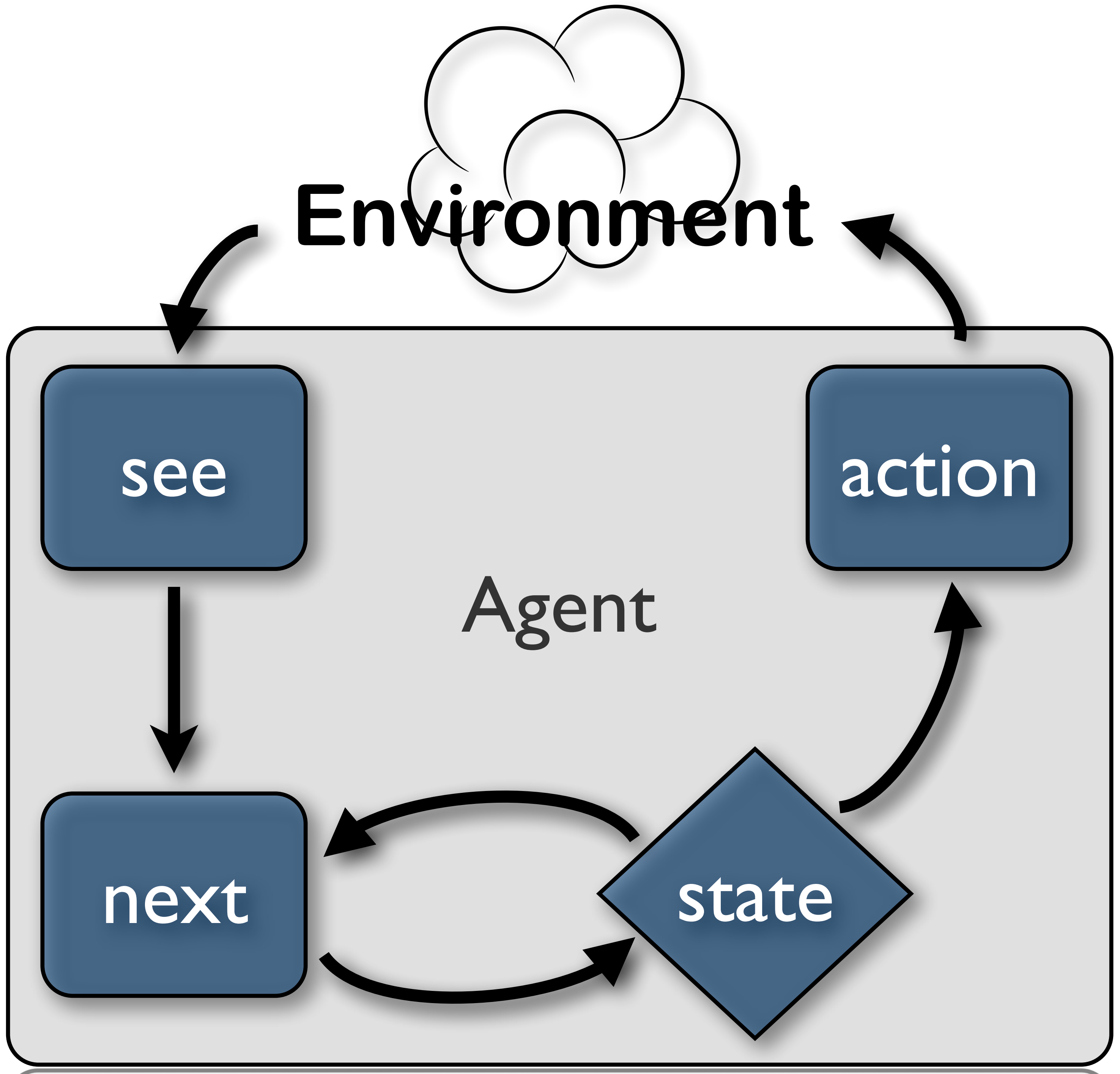
A reactive agent will always do the same thing in the same state.

Purely Reactive Robots

- A simple reactive program for a robot might be:
  - *Drive forward until you bump into something. Then, turn to the right. Repeat.*



Agents with State

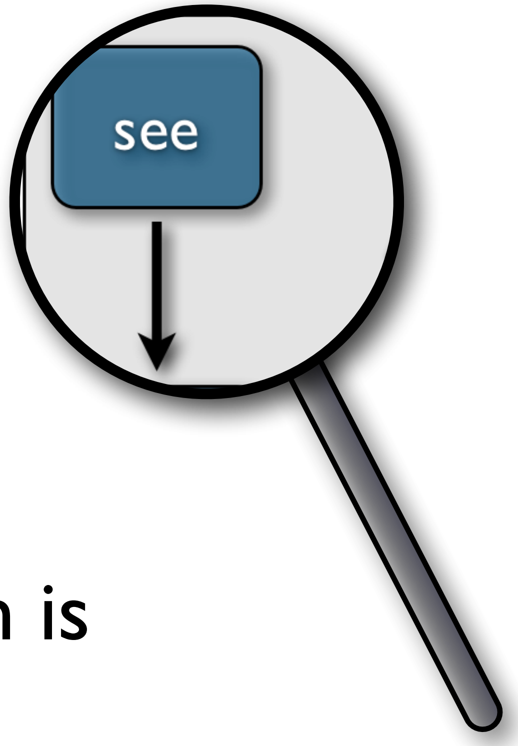


Perception

- 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**:

$$see : E \to Per$$

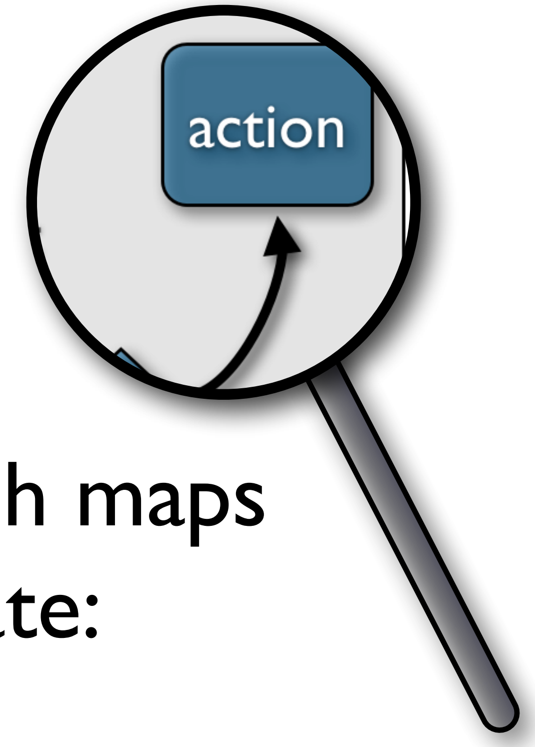
  - ...which maps environment states to percepts.



- 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:



$$action : I \to Ac$$

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

$$next : I \times Per \to I$$

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

Agents with State

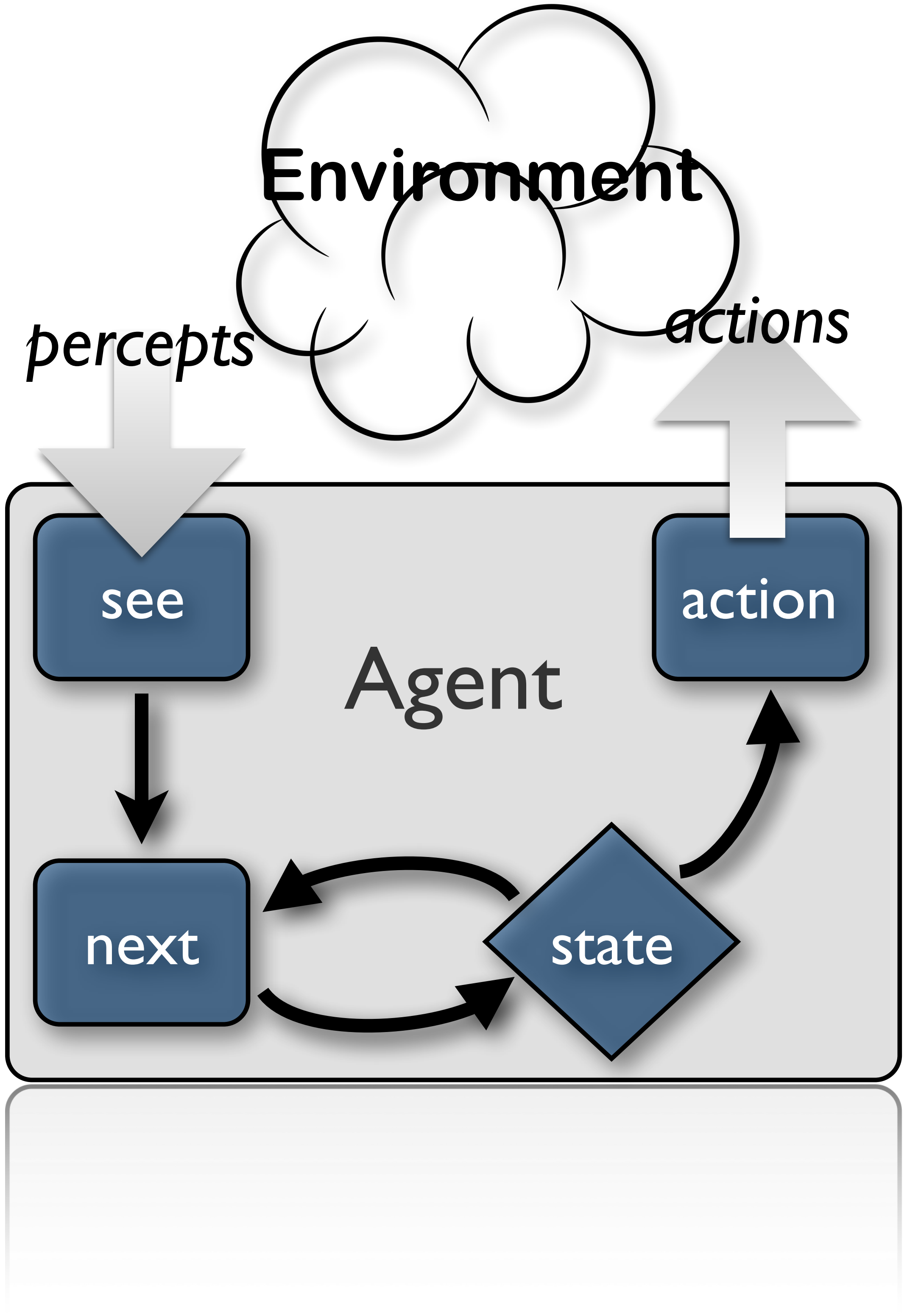
1. Agent starts in some initial internal state i_0.
2. Observes its environment state e, and generates a percept $see(e)$.
3. Internal state of the agent is then updated via $next$ function, becoming $next(i_0, see(e))$.
4. The action selected by the agent is $action(next(i_0, see(e)))$.

   This action is then performed.
5. Goto (2).

A Robot with state

- *per* is a bool that indicates "against an object"
- *i* is an integer, "against object for i steps".
- *see* updates *per* each step, indicating if the robot is against an object.
- *next* is as follows:

$$next(i) = \begin{cases} \text{i+1} & \text{if } per = \text{true} \\ 0 & \text{otherwise.} \end{cases}$$

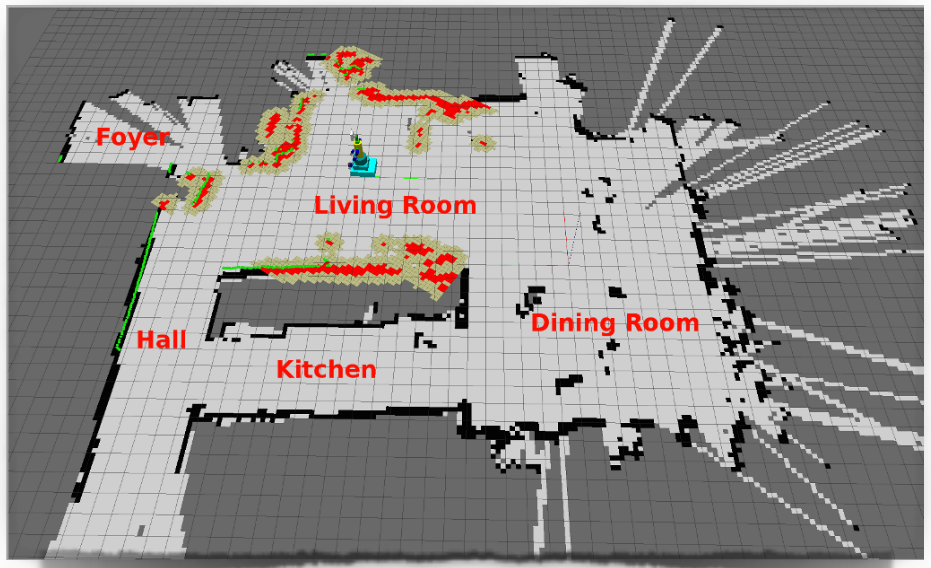


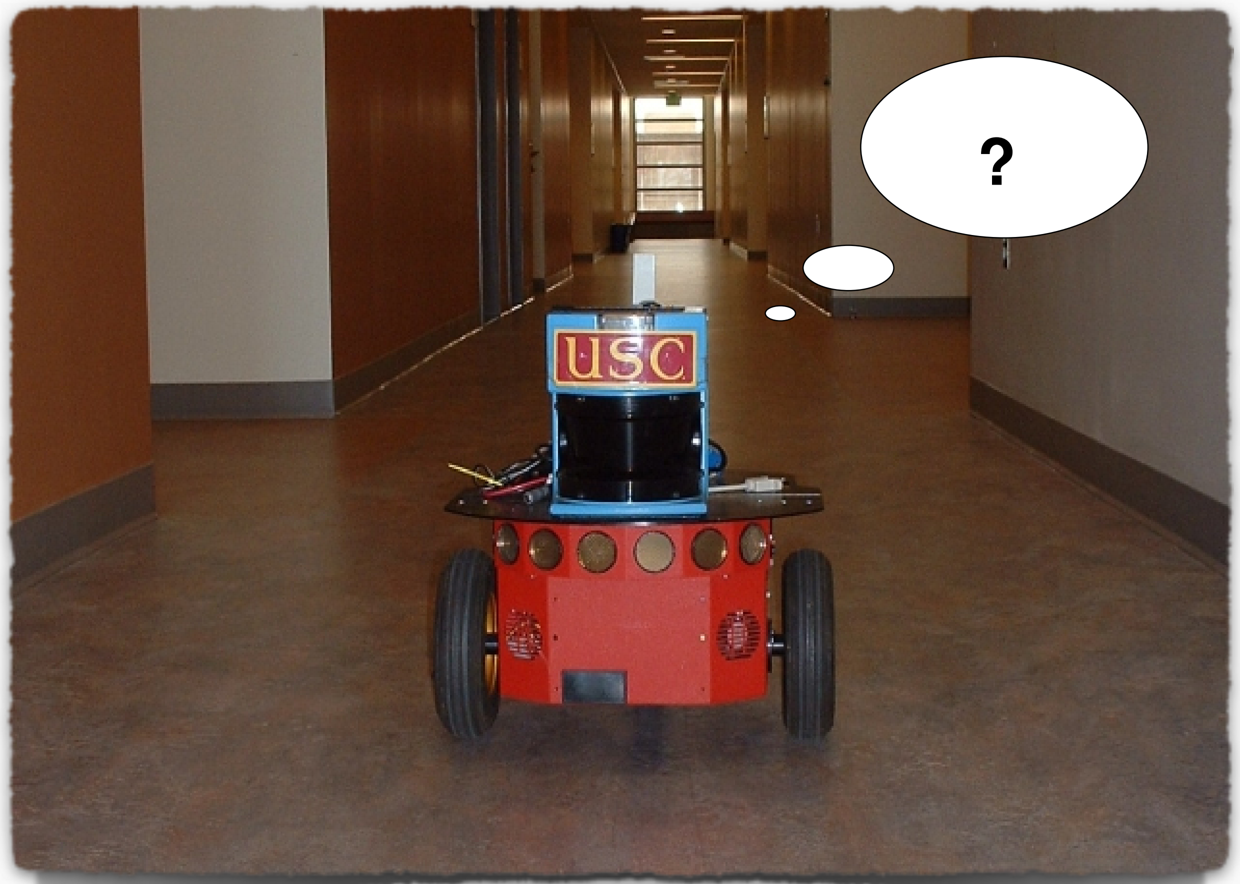
A Robot with state

- Now the robot can take more sophisticated action.
    - For example, backing up if it cannot turn away from the wall immediately.
- This is an example of a common situation in robotics.
- **Trading memory and computation for sensing.**

What is mobile robotics?

- Last time we boiled the challenges of mobile robotics down to:



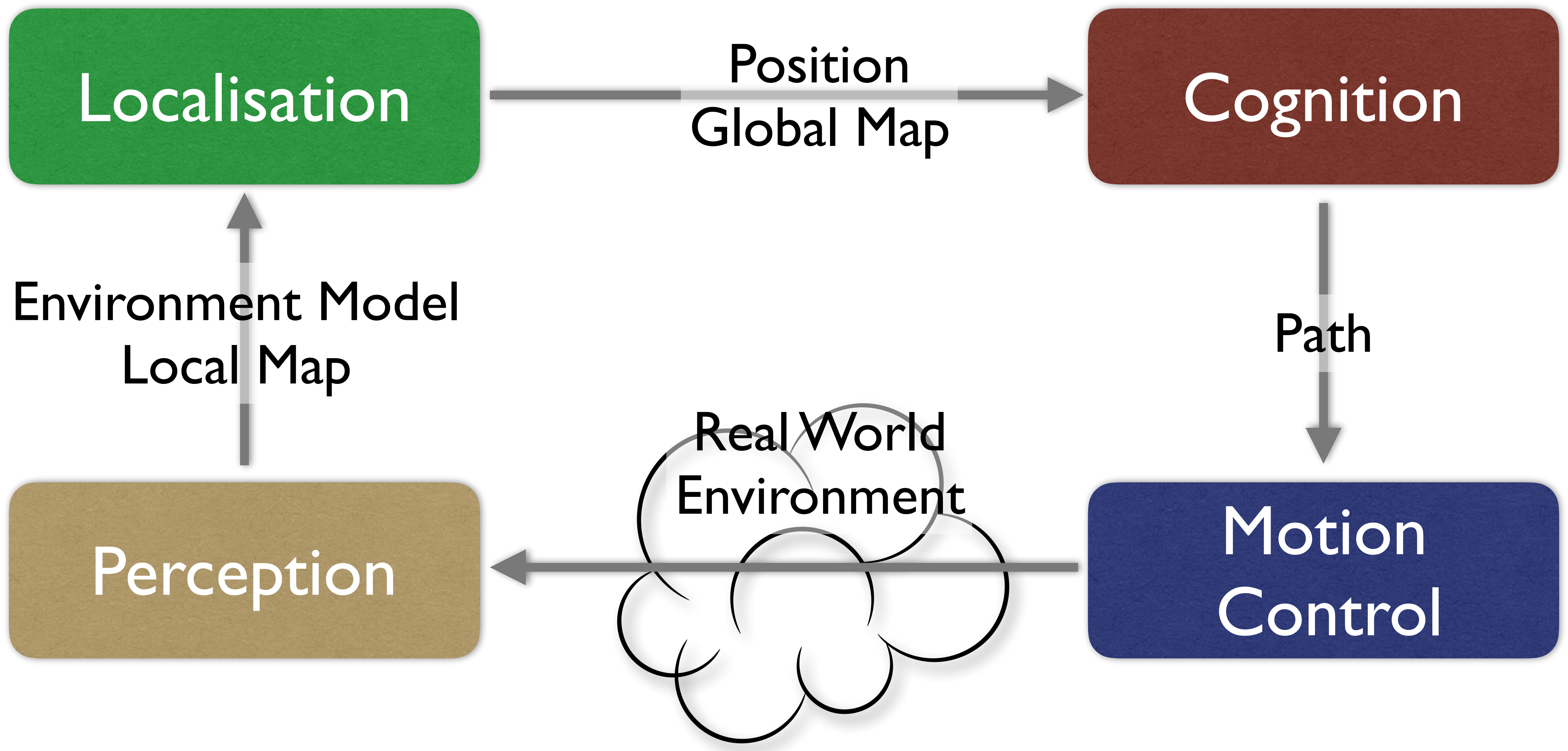


  - Where am I ?
  - Where am I going ?
  - How do I get there ?
- Now we'll start talking about how to answer these questions.

The pieces we need

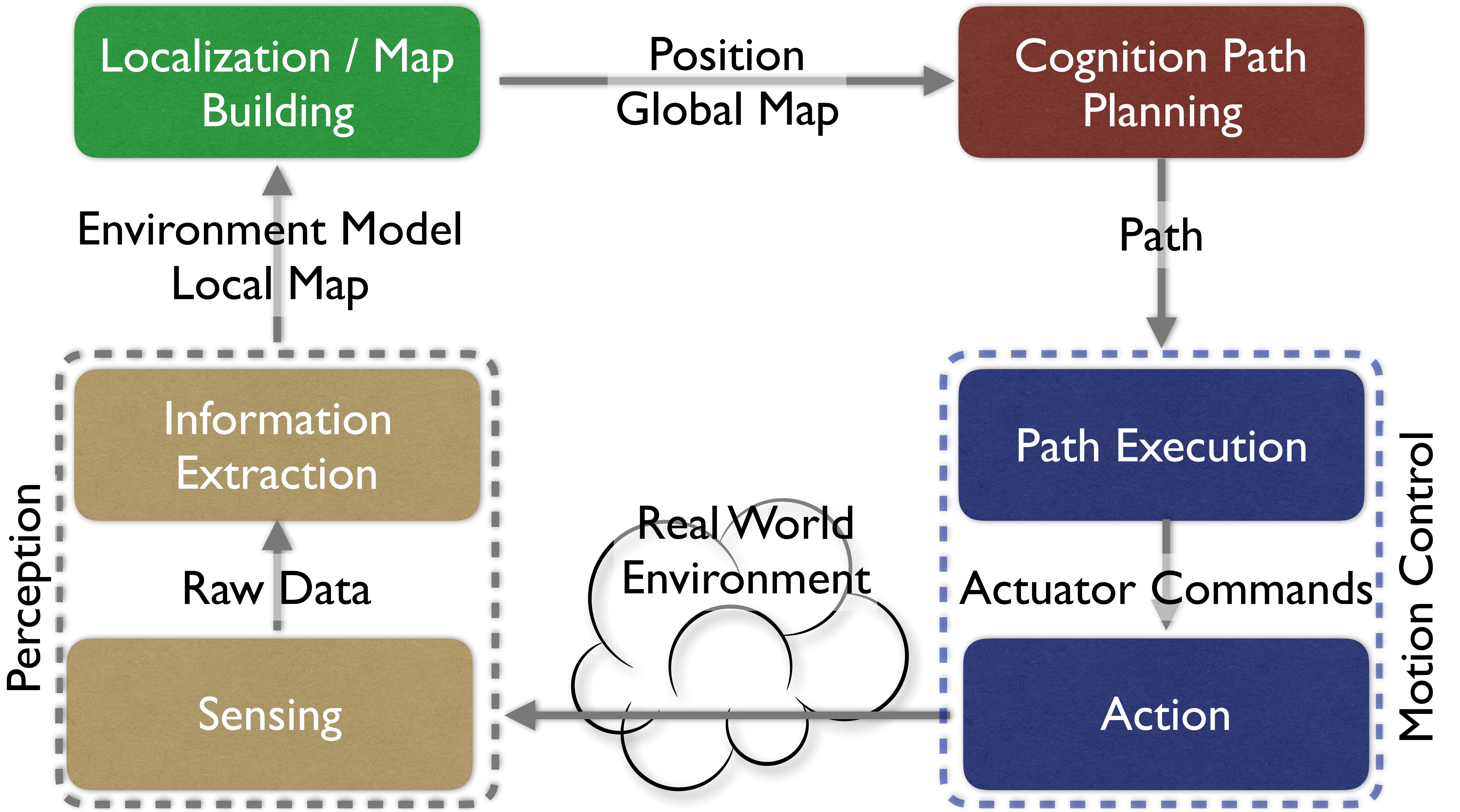
- ***Locomotion*** and ***Kinematics***
  - How to make the robot move, tradeoff between manoeverability and ease of control.
- ***Perception***
  - How to make the robot ``see''. Dealing with uncertainty in sensor input and changing environment. Tradeoff between cost, data and computation.
- ***Localisation*** and ***Mapping***
  - Establish the robot's position, and an idea of what it's environment looks like.
- ***Planning*** and ***navigation***
  - How the robot can find a route to its goal, how it can follow the route.

General control architecture



Original Source: M. Wooldridge, S.Parsons, D.Grossi - updated by Terry Payne, Autumn 2016

General control architecture



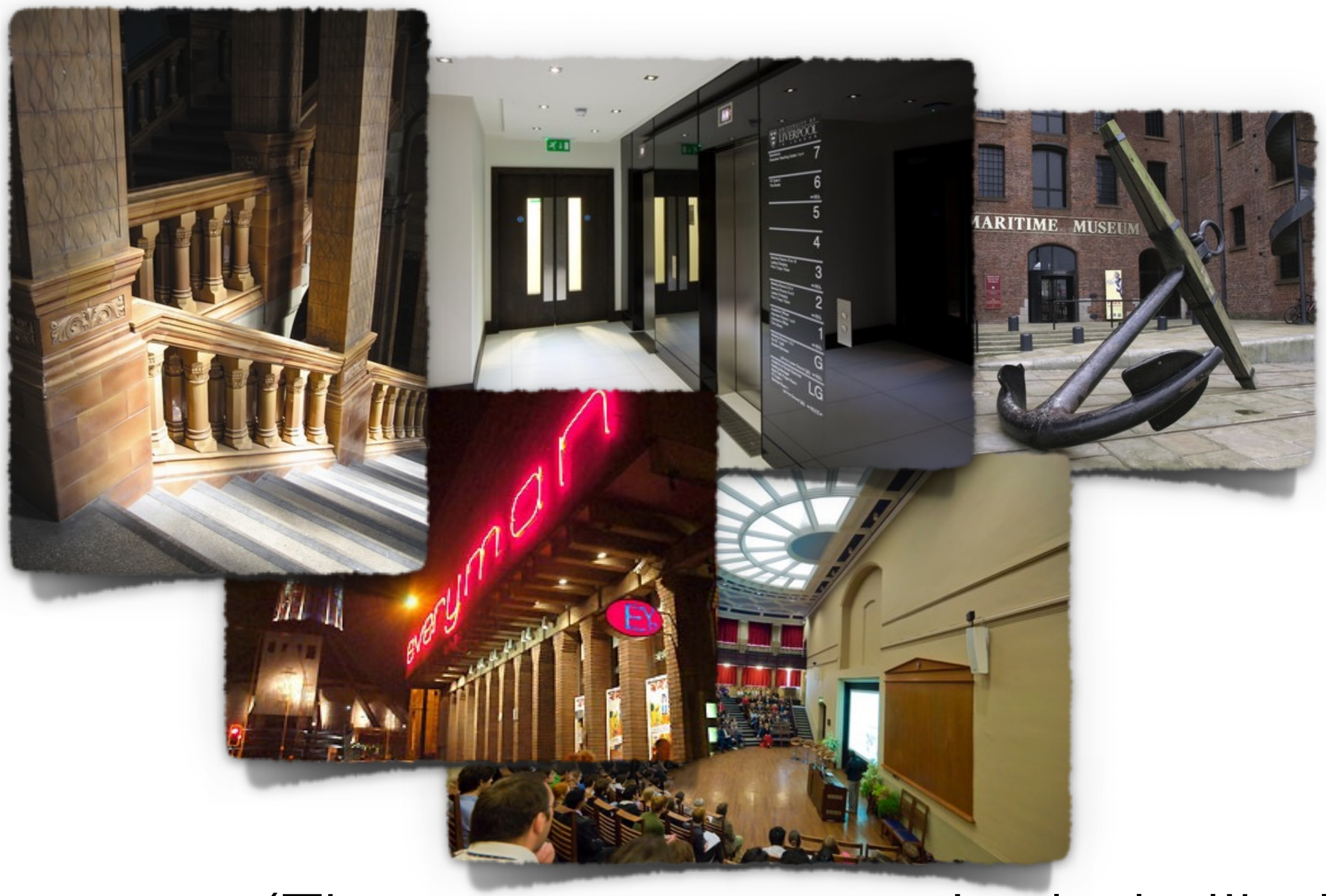
Original Source: M. Wooldridge, S.Parsons, D.Grossi - updated by Terry Payne, Autumn 2016

What makes it particularly hard

- Changing environment.
  - Things change.
  - Things get in the way.
- No compact model available.
  - How do you represent this all?
- Many sources of uncertainty.
  - All information comes from sensors which have errors.
  - The process of extracting useful information from sensors has errors

The basic operations

- We start with what the robot can ``see''.



- There are several forms this might take, but it will depend on:
- What sensors the robot has
- What features can be extracted.

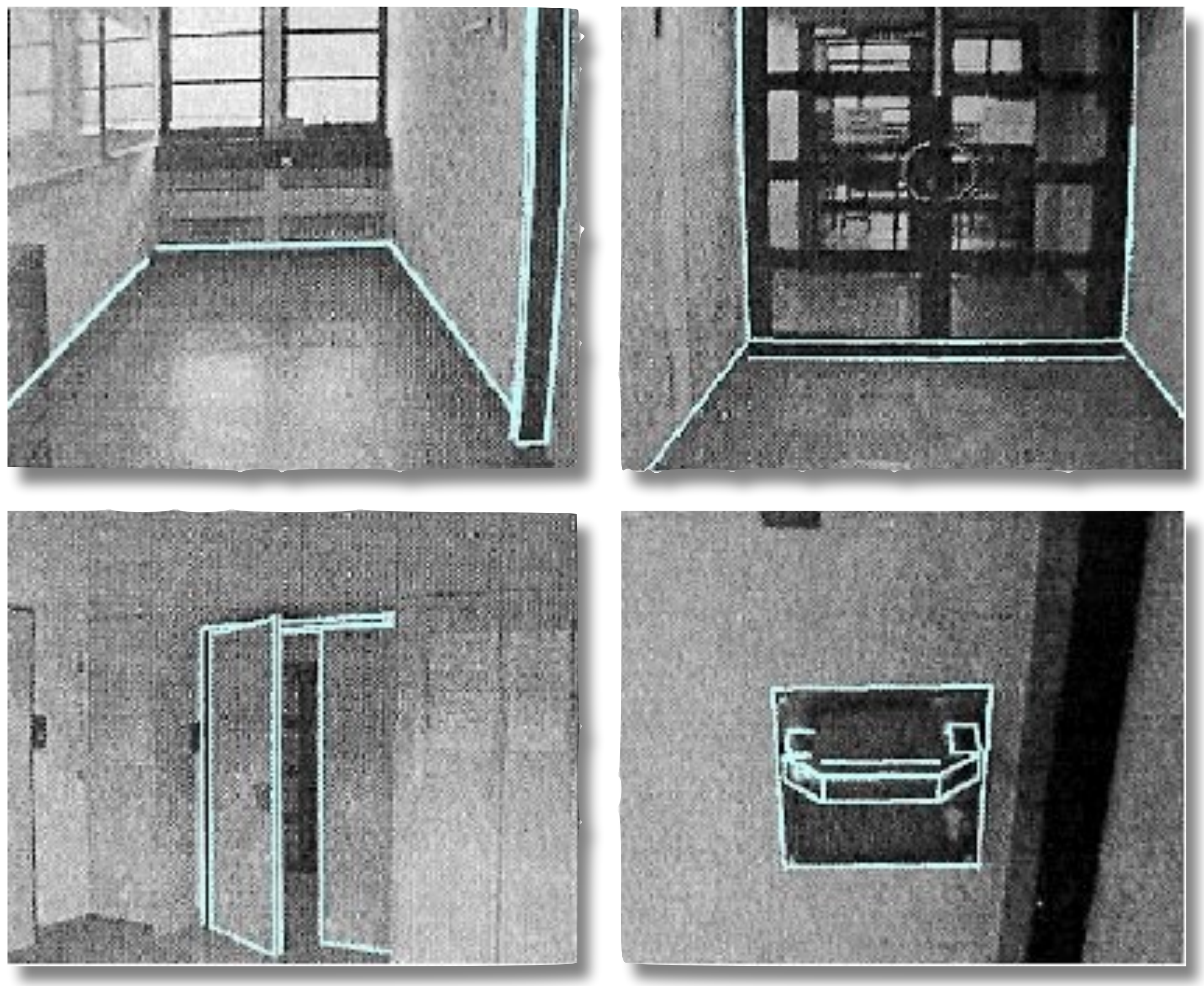
- (These are not a particularly likely set of features.)

Mapping features

- A ***map*** then says, for example, how these features sit relative to one another.

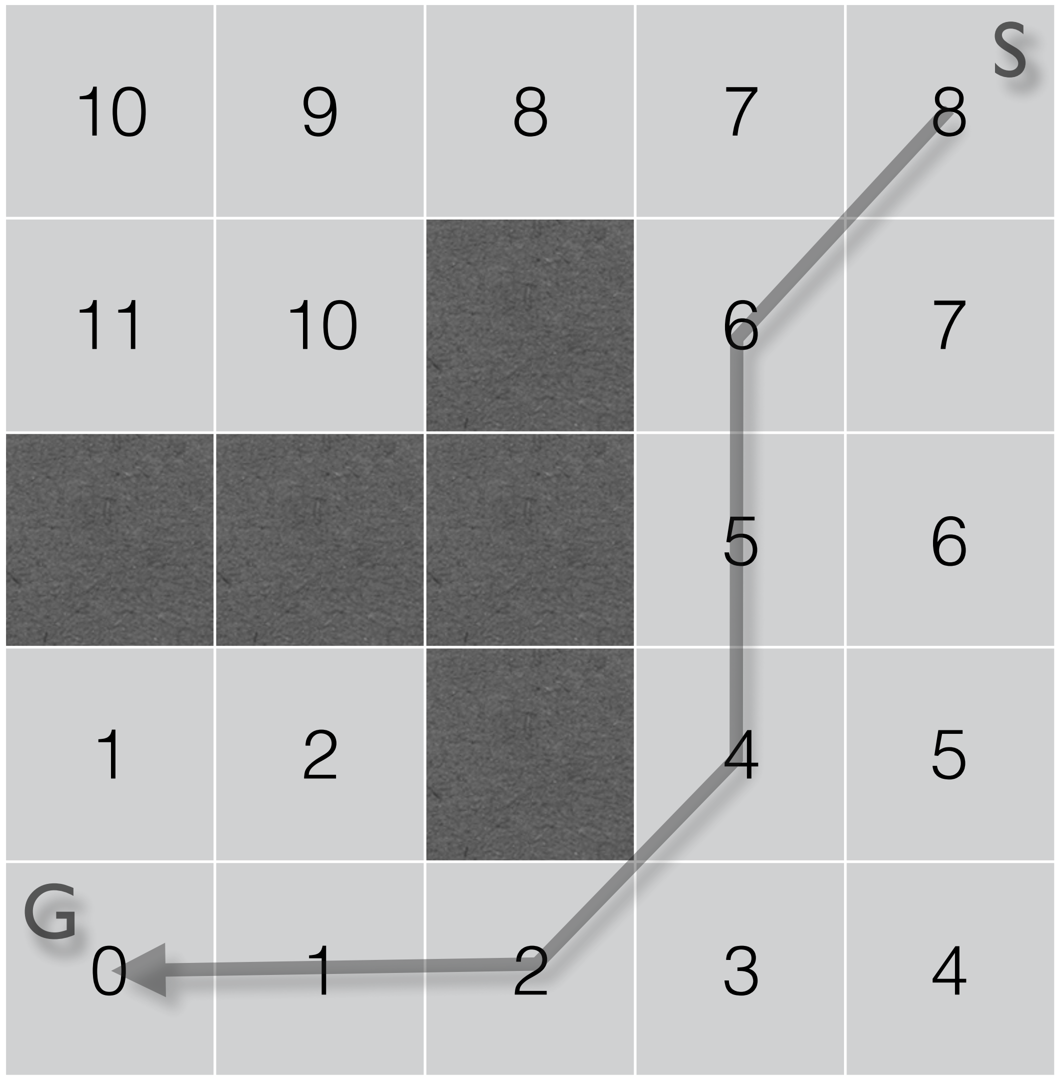
Localisation

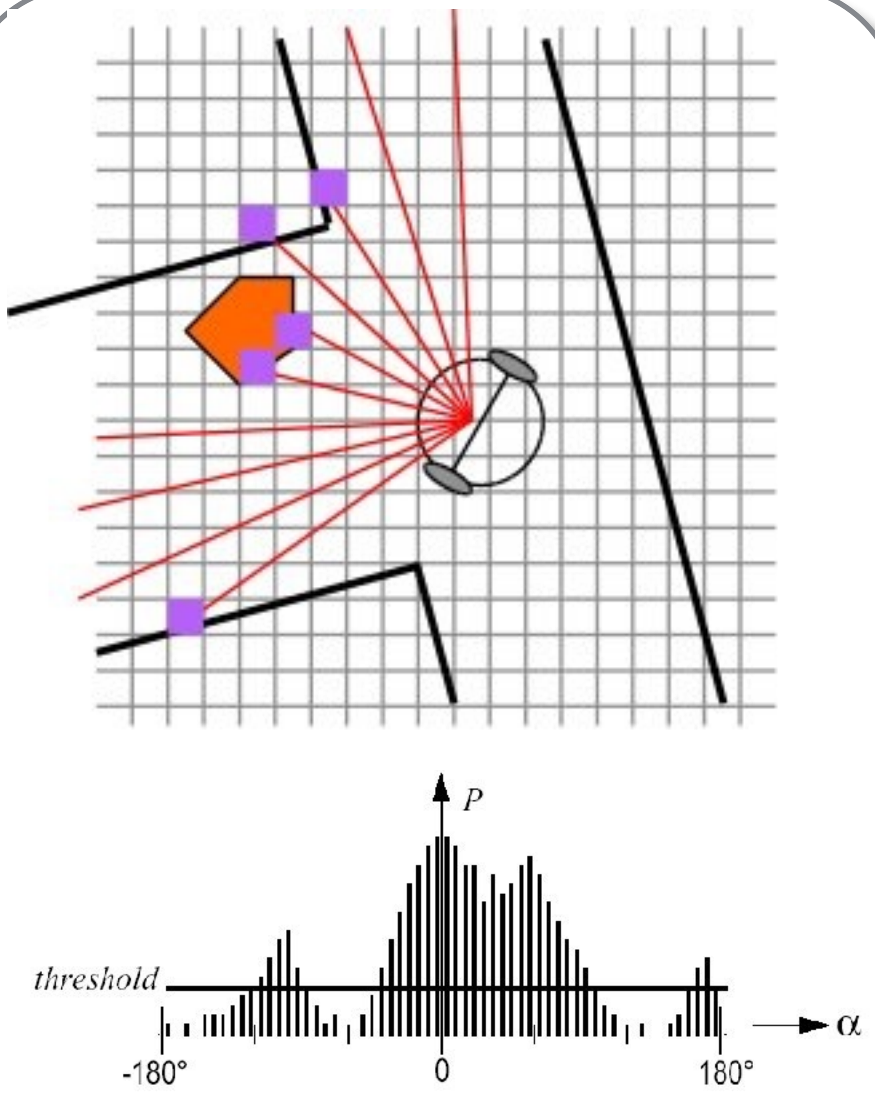
- A robot ***localises*** by identifying features and the position in the map from which it could see them.



Lanser et al (1996)

Navigatio



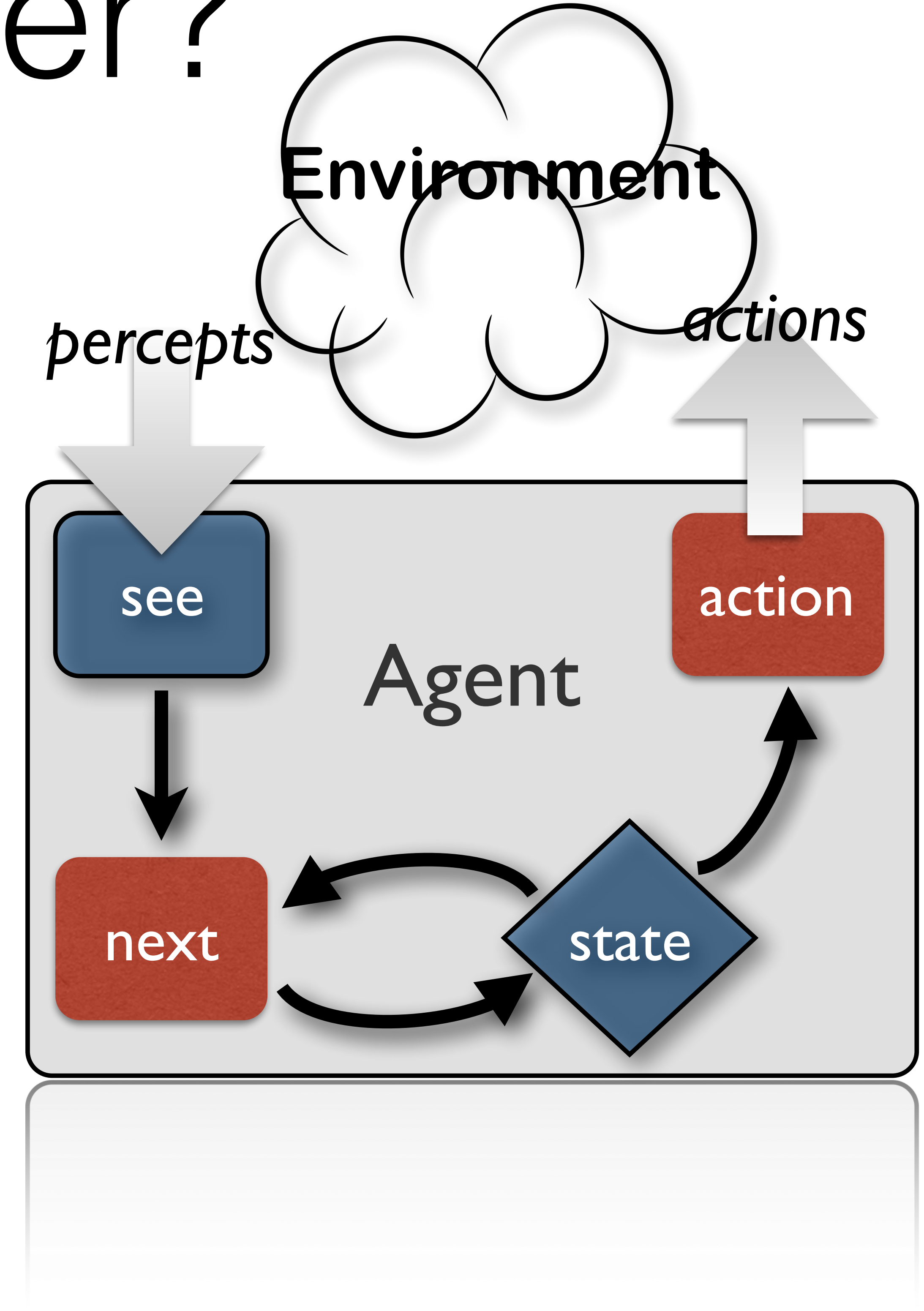


…avoiding things that get in the way!

- Navigation is then a combination of finding a path through the map…

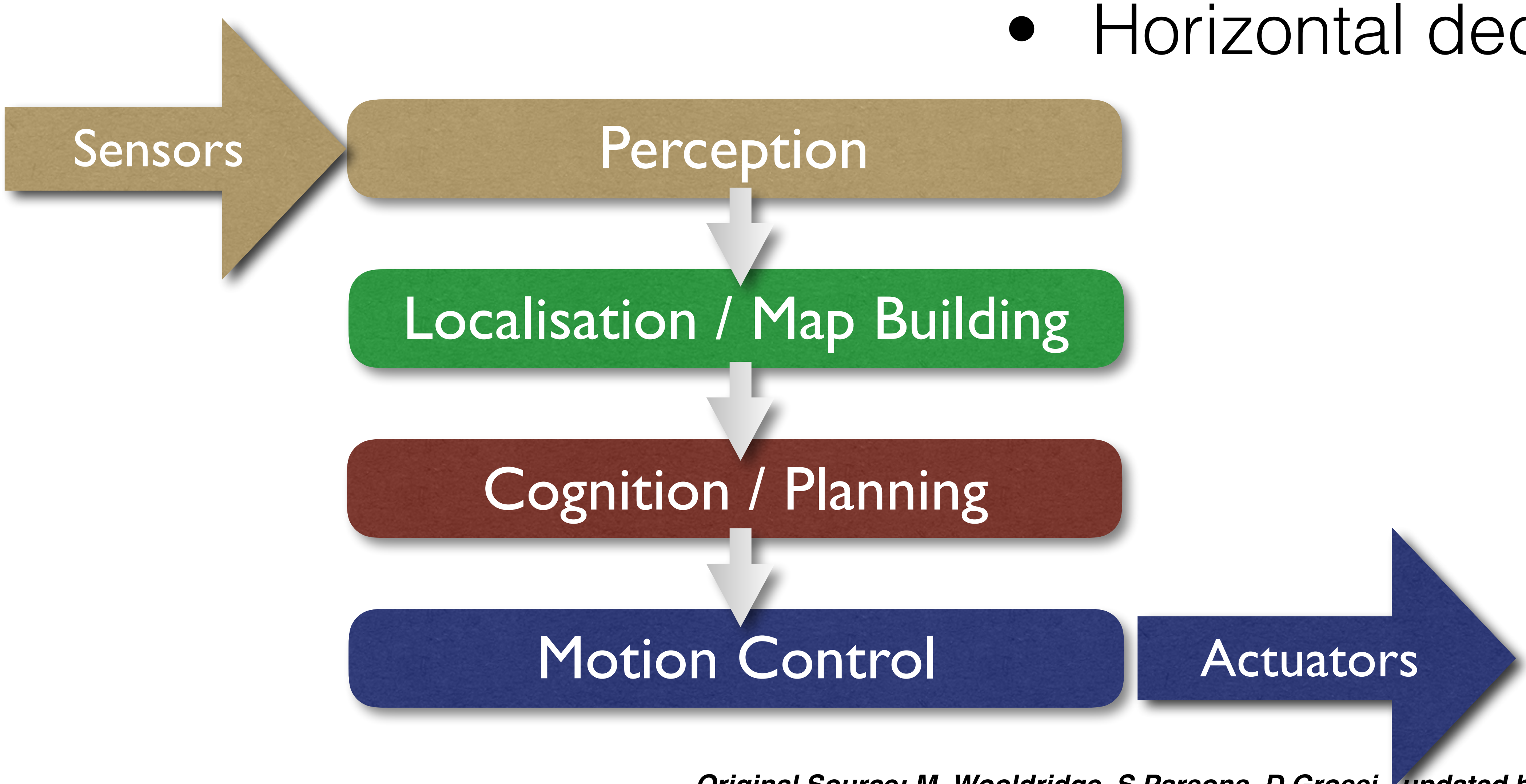
How do we put these together?

- A system architecture specifies how these pieces fit together.
- Consider these to be refinements of the "agent with state" from above.
    - Breaking down *next* and *action* into additional pieces.
    - Adding in new aspects of state *i*.



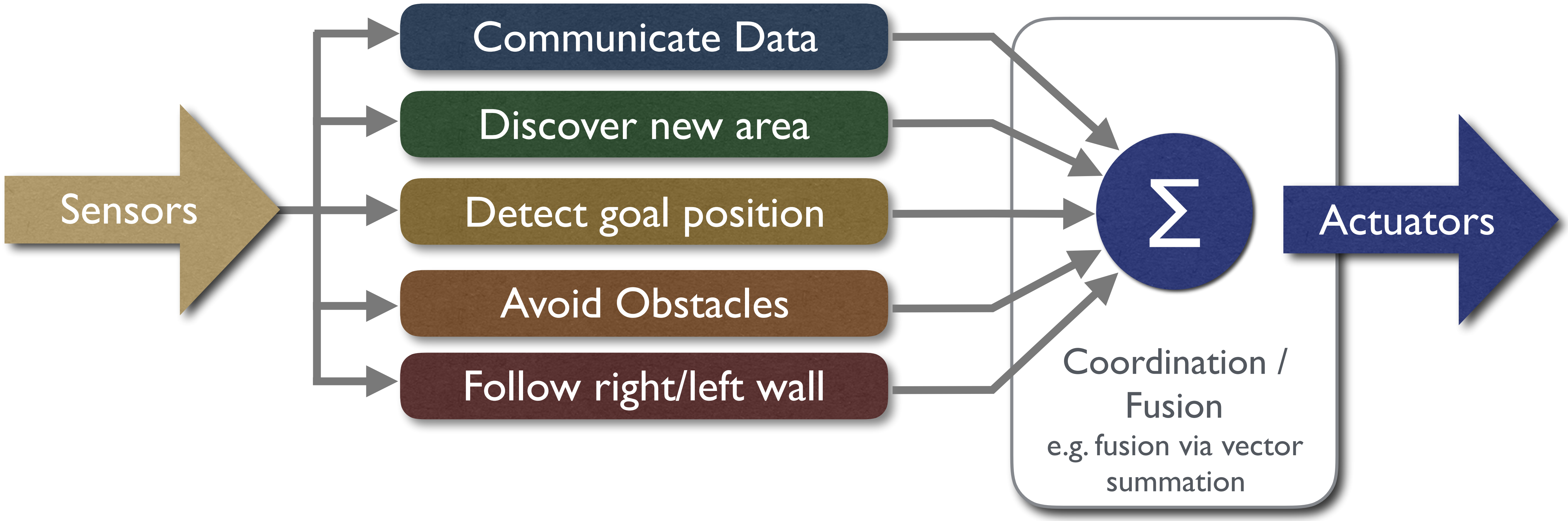
Approach: Classical/ Deliberative

- Complete modelling
  - Function based
  - Horizontal decomposition



Approach: Behaviour Based

- Sparse or no modelling
- Behaviour based
- Vertical decomposition
- Bottom up



Original Source: M. Wooldridge, S.Parsons, D.Grossi - updated by Terry Payne, Autumn 2016

Approach: Hybrid

- A combination of the previous two approaches.
- Exactly the best way to combine them nobody knows.
- Typical approach is:
  - Let ``lower level'' pieces be behaviour based
    - Localisation
    - Obstacle avoidance
    - Data collection
  - Let more ``cognitive'' pieces be deliberative
    - Planning
    - Map building

Summary

- Last time we talked about what the main challenges of mobile robotics are.
    - These lectures started to describe how we can meet these challenges.
    - We covered the main things we need to be able to autonomously control a robot.
    - Along the way we looked at how notions of agency --- and what this means for autonomy --- can help.

- In the next lecture, we will start to look at Lego EV3 components and the Lejos environment