Linköping University

Automated Planning

Planning under Uncertainty

Jonas Kvarnström
Department of Computer and Information Science
Linköping University

jonas.kvarnstrom@liu.se – 2019

Restricted State Transition System

- Recall the classical **state transition system** $\Sigma = (S, A, \gamma)$
  - $S = \{ s_0, s_1, \dots \}$: Finite set of **world states**
  - $A = \{ a_0, a_1, \dots \}$: Finite set of **actions**
  - $\gamma: S \times A \rightarrow 2^S$: **State transition function**, where $|\gamma(s, a)| \leq 1$
    - If $\gamma(s, a) = \{s'\}$,
      then whenever you are in state *s*,
      you can execute action *a*
      and you end up in state *s'*
    - If $\gamma(s, a) = \emptyset$ (the empty set),
      then *a* cannot be executed in *s*

$S = \{ s_0, s_1, \dots \}$

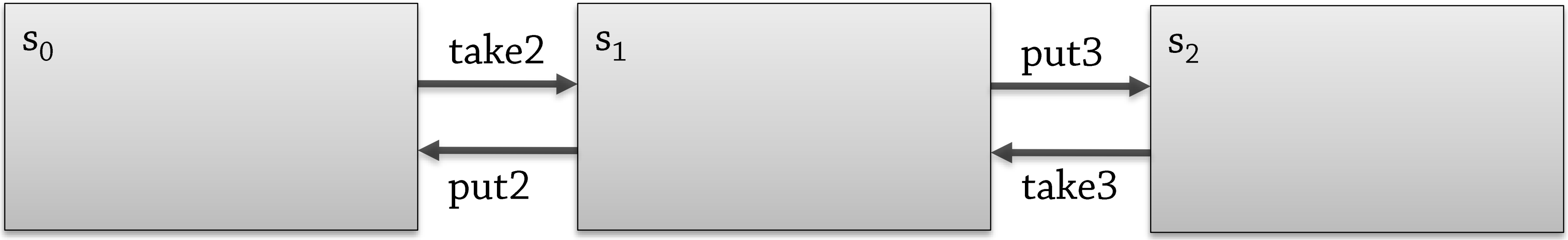
$A = \{ \text{take1}, \text{put1}, \dots \}$

$\gamma: S \times A \rightarrow 2^S$

$\gamma(s_0, \text{take2}) = \{ s_1 \}$

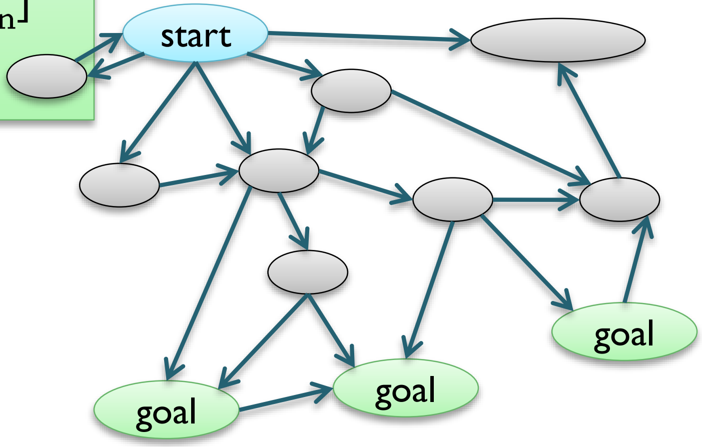
$\gamma(s_1, \text{take2}) = \emptyset$

Often we also add a cost function:
c: $S \times A \rightarrow \mathbb{R}$



Classical Planning Problem

- Recall the **classical planning problem**
  - Let $\Sigma = (S, A, \gamma)$ be a state transition system satisfying the assumptions A0 to A7 (called a **restricted** state transition system in the book)
  - Let $s_0 \in S$ be the **initial state**
  - Let $S_g \subseteq S$ be the **set of goal states**
  - Then, find a **sequence** of **transitions** labeled with actions $[a_1, a_2, \ldots, a_n]$ that can be applied starting at s_0 resulting in a **sequence** of **states** $[s_1, s_2, \ldots, s_n]$ such that $s_n \in S_g$



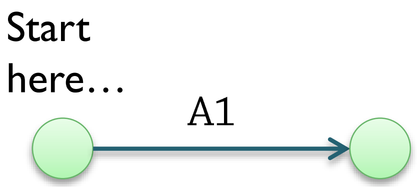
Planning with Complete Information

- This assumes we know in advance:
  - The state of the world when **plan execution** starts
  - The **outcome** of any action, given the state where it is executed
    - State + action ➔ unique resulting state

- Solution exists ➔ **Unconditional** solution exists

Planning

Model says: we end up in this specific state!



Execution

No new information can be relevant (at least in theory!)

Just follow the unconditional plan…

Multiple Outcomes

- In reality, actions may have **multiple outcomes**
  - Some outcomes can indicate **faulty / imperfect execution**
    - **pick-up(object)**
      Intended outcome: carrying(object) is true
      Unintended outcome: carrying(object) is false
    - **move(100,100)**
      Intended outcome: xpos(robot)=100
      Unintended outcome: xpos(robot) != 100
    - **jump-with-parachute**
      Intended outcome: alive is true
      Unintended outcome: alive is false
  - Some outcomes are more **random**,
    but clearly **desirable / undesirable**
    - Pick a present at random – do I get the one I longed for?
    - Toss a coin – do I win?
  - Sometimes we have **no clear idea** what is desirable
    - Outcome will affect how we can continue,
      but in less predictable ways

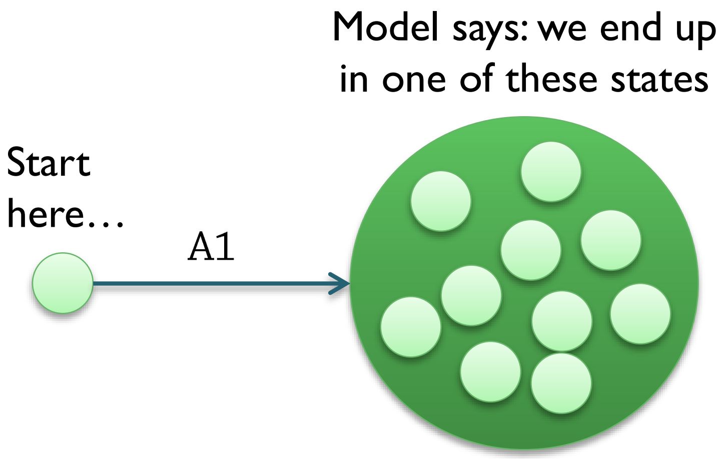
To a *planner*, there is generally no difference between these cases!

Non-Deterministic Planning

Nondeterministic Planning

- **Nondeterministic** planning:
  - $S = \{ s_0, s_1, \ldots \}$: Finite set of **world states**
  - $A = \{ a_0, a_1, \ldots \}$: Finite set of **actions**
  - $\gamma\colon S \times A \rightarrow 2^S$: **State transition function**, where $|\gamma(s, a)|$ **is finite**

Planning



Model says: we end up in one of these states

Start here…

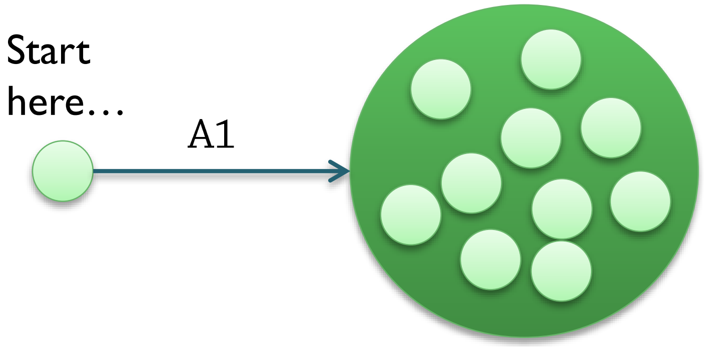
Execution

Will we find out more when we execute?

FOND Planning

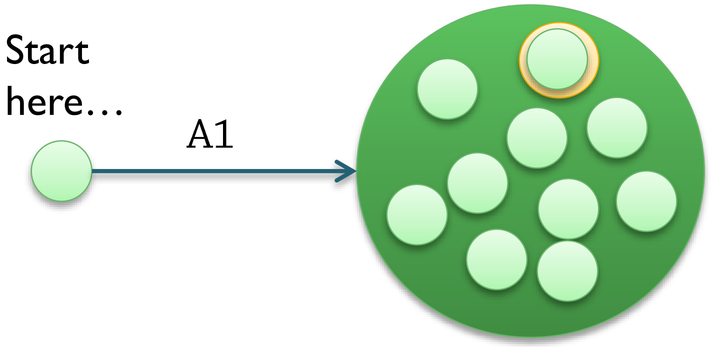
- FOND: **<u>Fully Observable</u>** Non-Deterministic
  - After executing an action, sensors determine exactly which state we are in

Planning



Model says: we end up in **one of** these states

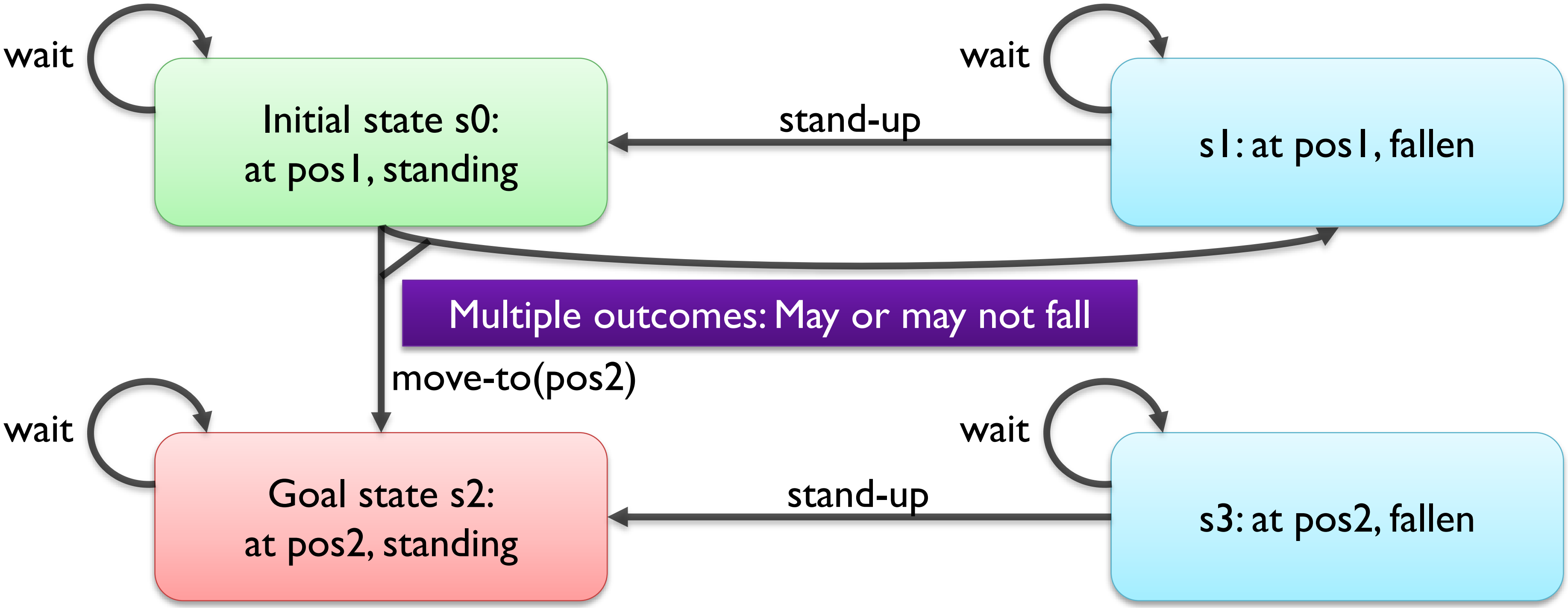
Execution



Sensors say: we are in **this** state!

FOND Planning: Plan Structure (1)

- Example state transition system:



wait

Initial state s0:
at pos1, standing

stand-up

wait

s1: at pos1, fallen

Multiple outcomes: May or may not fall

move-to(pos2)

wait

Goal state s2:
at pos2, standing

stand-up

wait

s3: at pos2, fallen

- **<u>Intuitive strategy</u>**:

```
while (not in s2) {
    move-to(pos2);
    if (fallen) stand-up;
}
```

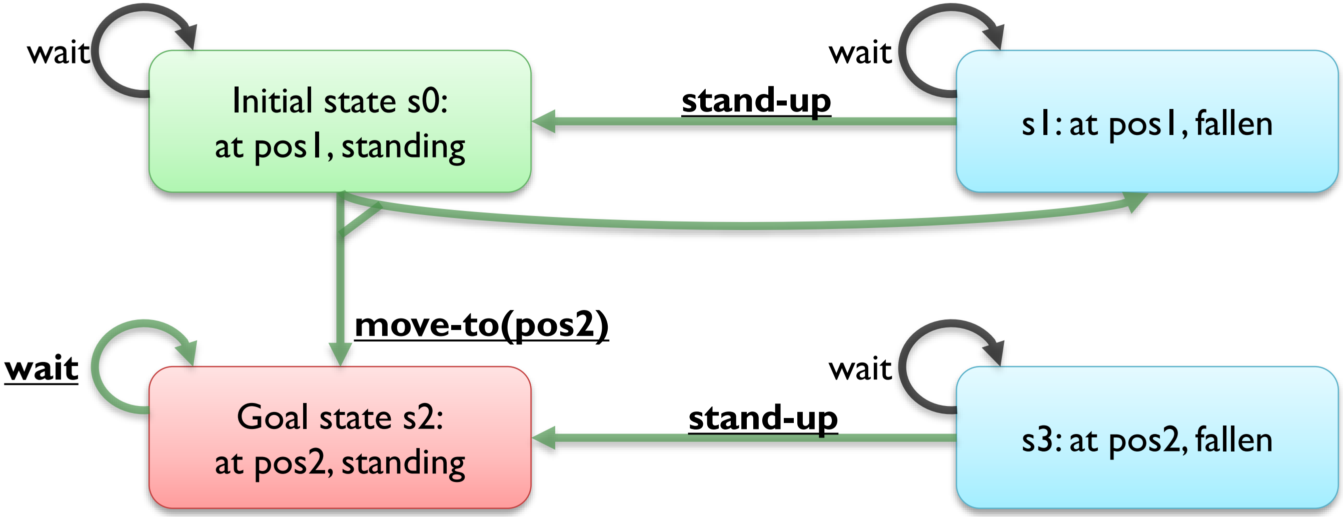
FOND ➔ The action to execute should depend on the current state, which depends on previous outcomes

There may be no upper bound on how many actions we may have to execute!

FOND Planning: Plan Structure (2)

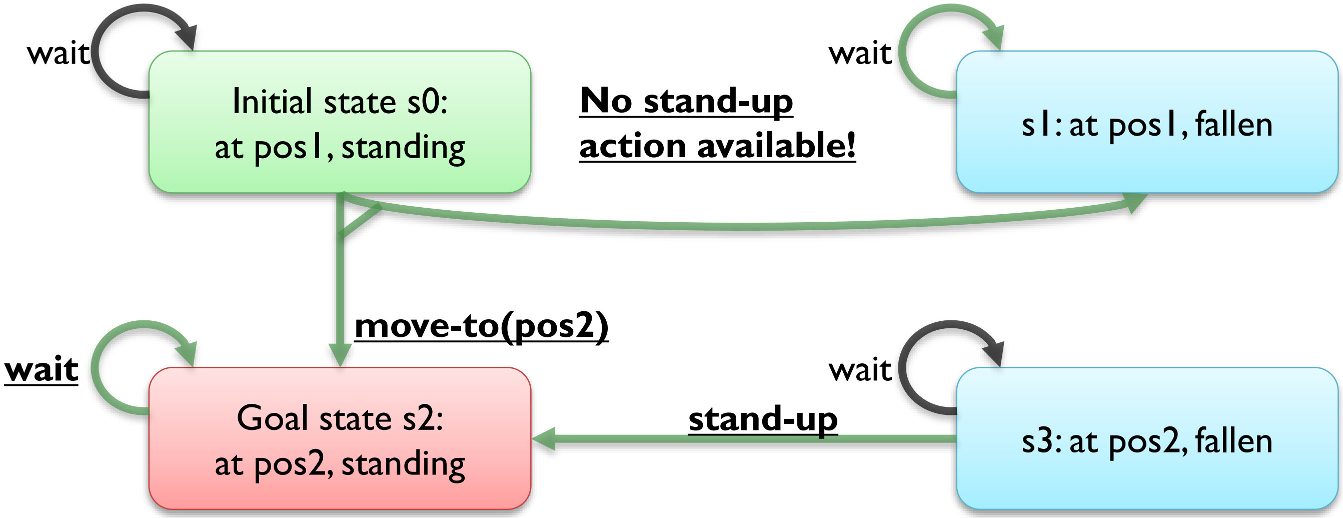
- Examples of formal plan structures:
  - **Conditional plans** (with if/then/else statements)
  - **Policies** $\pi : S \to A$
    - Defining, **for each state**, which action to execute **whenever** we end up there
    - $\pi(s0)$ = move-to(pos2)
    - $\pi(s1)$ = stand-up
    - $\pi(s2)$ = wait
    - $\pi(s3)$ = stand-up

Or at least, for every state
that is *reachable* from the possible initial states
(➔ A policy can be a *partial function*)



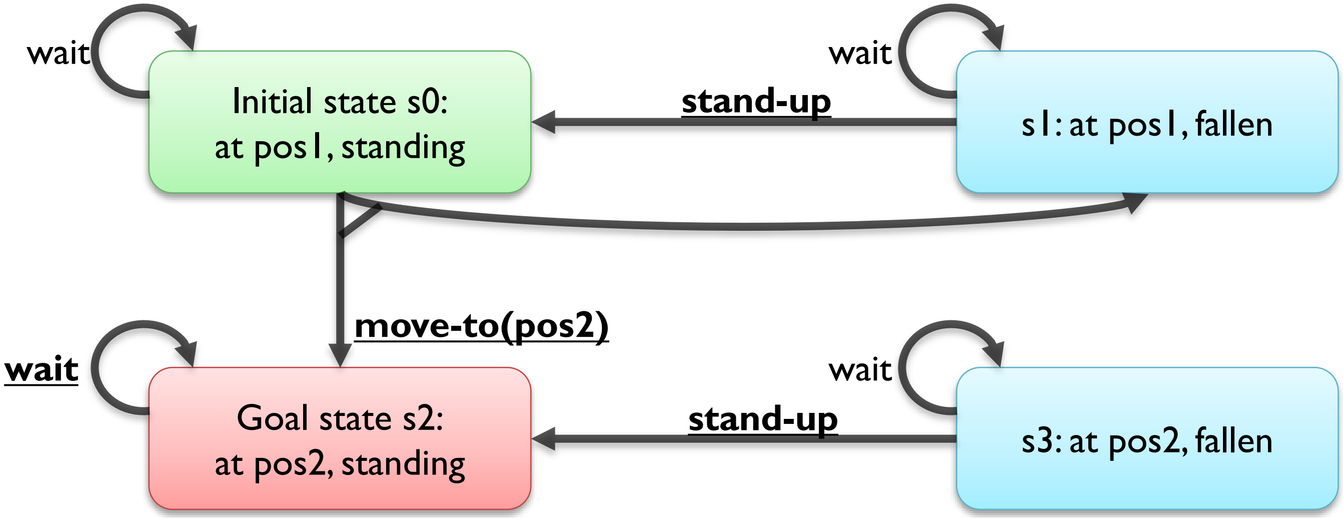
Solution Types 1

- Assume our **objective** is still to **reach a state** in S_g
  - And then remain there (executing "wait" actions forever)
    - A policy never terminates…
  - A **weak solution**:
    For *some* outcomes, the goal is reached in a finite number of steps
    - $\pi(s0)$ = move-to(pos2)
    - $\pi(s1)$ = wait
    - $\pi(s2)$ = wait
    - $\pi(s3)$ = stand-up



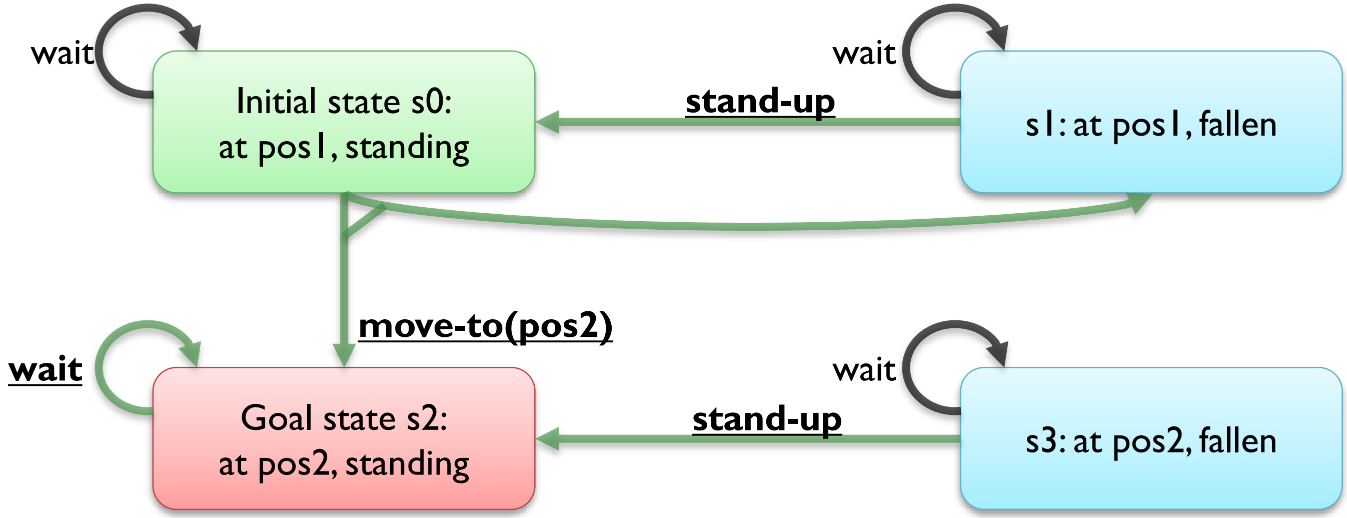
Solution Types 2

- Assume our **objective** is still to **reach a state** in S_g
  - A **strong** solution:
    For *every* outcome, the goal is reached in a finite number of steps
    - Not possible for this example problem
    - Could fall *every time*



Solution Types 3

- Assume our **objective** is still to **reach a state** in S_g
  - A **strong cyclic** solution *will* reach a goal state in a finite number of steps *given a fairness assumption*:
    Informally, "if we **can** exit a loop, we eventually **will**"
    - $\pi(s0)$ = move-to(pos2)
    - $\pi(s1)$ = stand-up
    - $\pi(s2)$ = wait
    - $\pi(s3)$ = stand-up

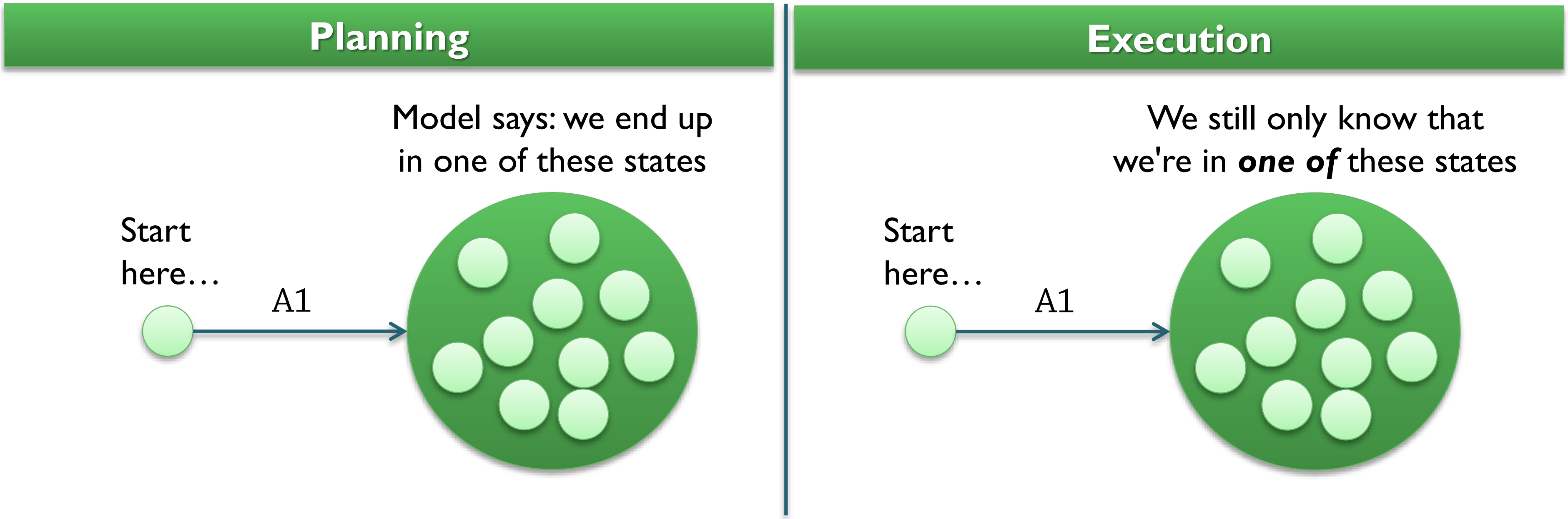


Solutions and Costs

- The **cost** of a **FOND policy** is undefined
  - We don't know in advance which actions we must execute
  - *And* we have no estimate of how *likely* different outcomes are

NOND Planning

- NOND: **Non-Observable** Non-Deterministic
  - Also called ***conformant*** *non-deterministic*
  - *Only* predictions can guide us – **no sensors to use during execution**
  - May still give sufficient information for solving a problem



Planning

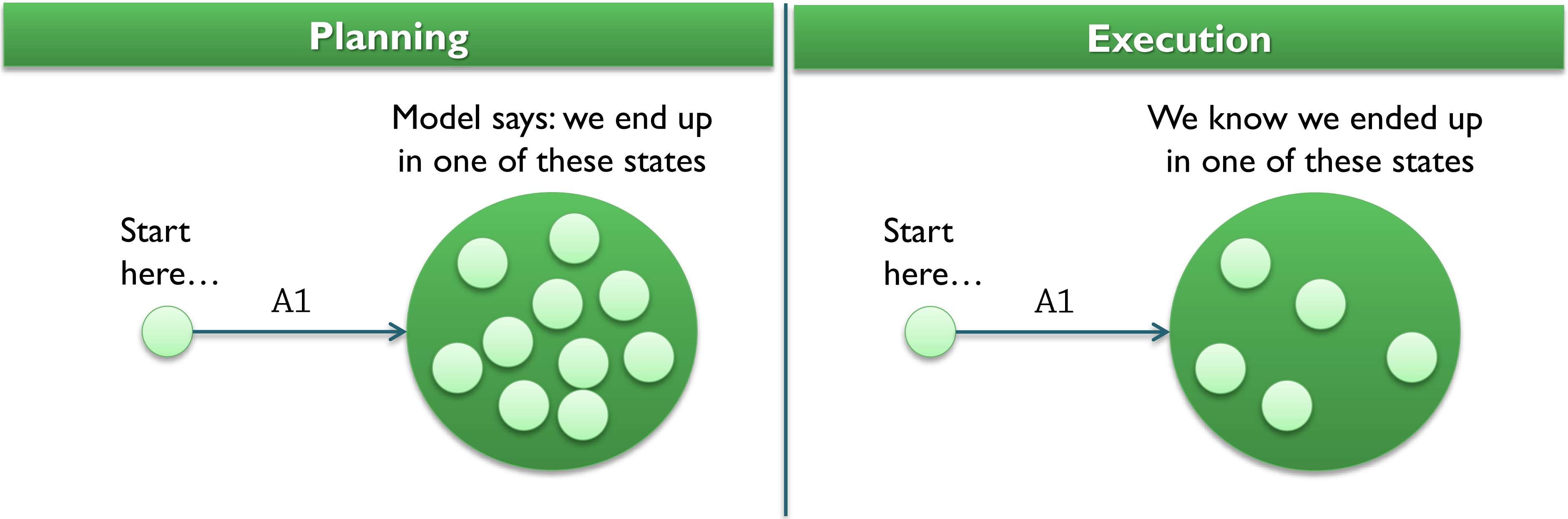
Model says: we end up in one of these states

Execution

We still only know that we're in ***one of*** these states

POND Planning

- POND: **Partially Observable** Non-Deterministic



Planning

Model says: we end up in one of these states

Execution

We know we ended up in one of these states

Overview

	Non-Observable: No information gained after action	**Fully Observable: Exact outcome known after action**	**Partially Observable: Some information gained after action**
Deterministic: Exact outcome known in advance	**Classical planning** (possibly with extensions) Information dimension is meaningless!		
Non-deterministic: Multiple outcomes, no probabilities	**NOND**: Conformant Planning	**FOND**: Conditional (Contingent) Planning	**POND**: Partially Observable, Non-Deterministic

We will not discuss non-deterministic planning algorithms!

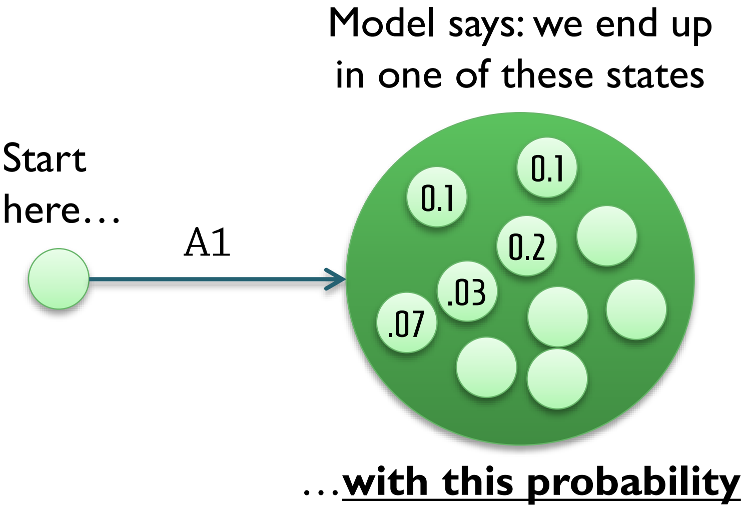
Probabilistic Planning:
Defining the World as a Stochastic System

Stochastic Systems

- **Probabilistic planning** uses a **stochastic system** $\Sigma = (S, A, P)$
  - $S = \{ s_0, s_1, \ldots \}$: Finite set of **world states**
  - $A = \{ a_0, a_1, \ldots \}$: Finite set of **actions**
  - $P(s, a, s')$: Given that we are in *s* and execute *a*, the **probability** of ending up in *s'*
  - For every state s and action a, we have $\sum_{s' \in S} P(s, a, s') = 1$: The world gives us 100% probability of ending up in *some* state

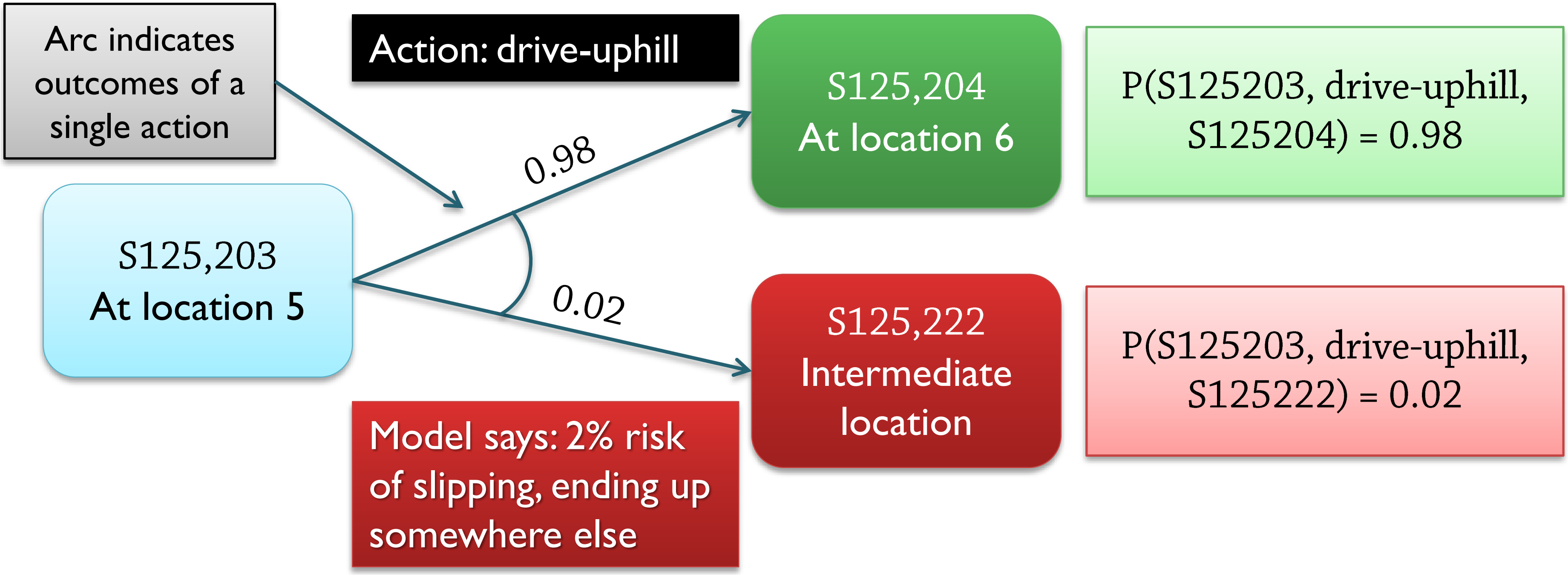
Replaces γ

Planning



Stochastic Systems (2)

Example with "desirable outcome"



Arc indicates outcomes of a single action

Action: drive-uphill

S125,203
At location 5

S125,204
At location 6

P(S125203, drive-uphill, S125204) = 0.98

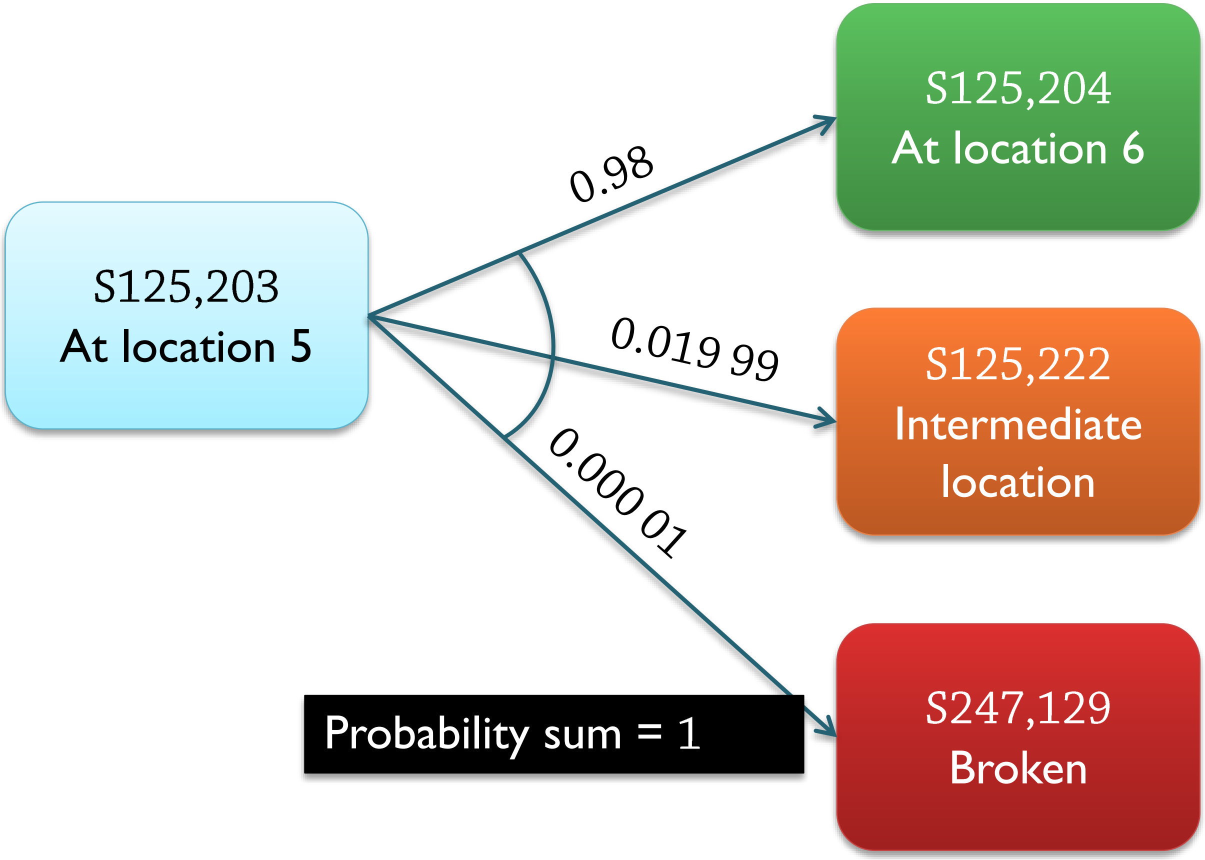
S125,222
Intermediate location

P(S125203, drive-uphill, S125222) = 0.02

Model says: 2% risk of slipping, ending up somewhere else

Stochastic Systems (3)

- May have very **unlikely** outcomes…

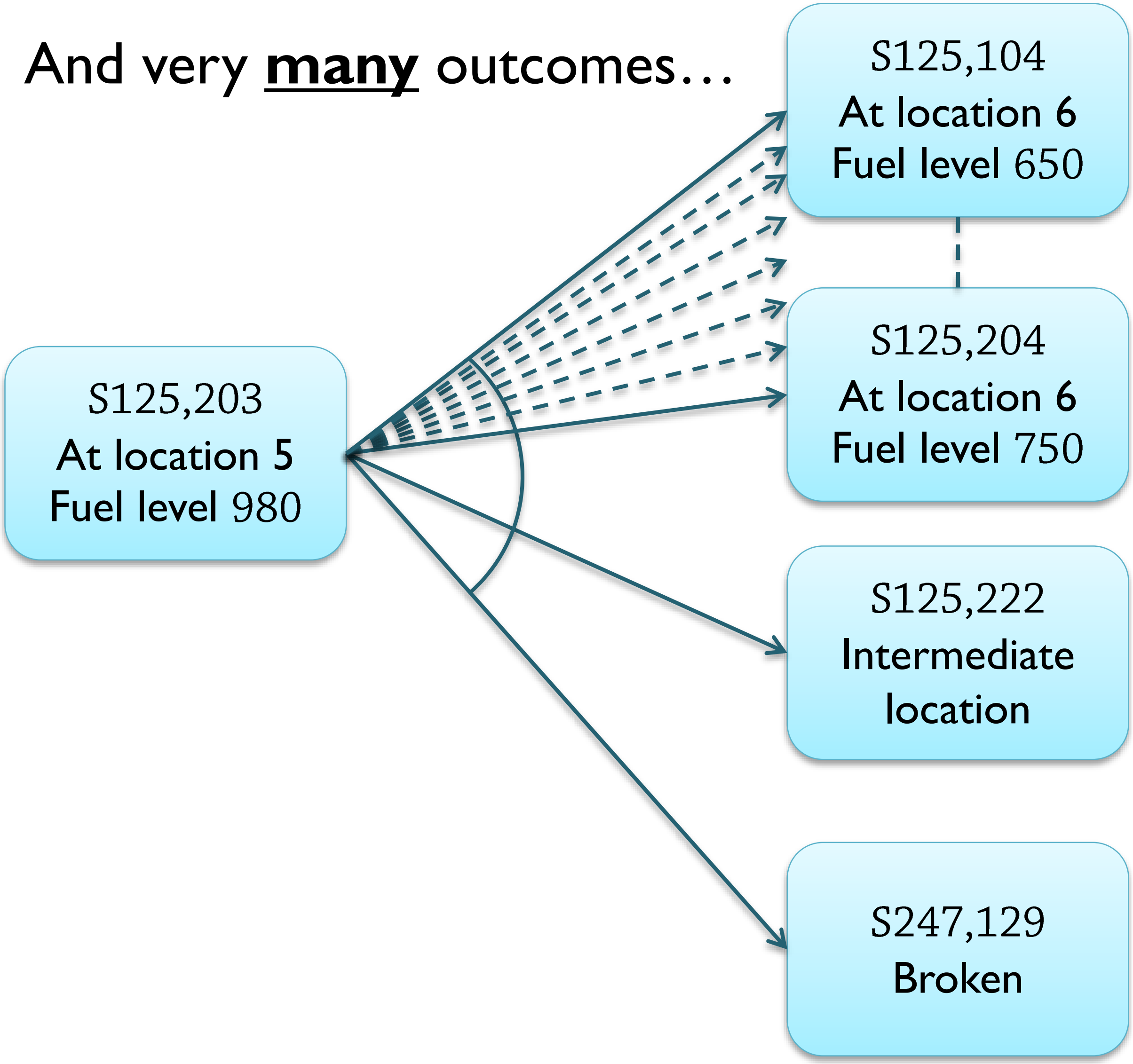


Probability sum = 1

Very unlikely, but may still be important to consider, if it has great impact on goal achievement!

Stochastic Systems (4)

- And very **many** outcomes…



S125,203
At location 5
Fuel level 980

S125,104
At location 6
Fuel level 650

S125,204
At location 6
Fuel level 750

S125,222
Intermediate location

S247,129
Broken

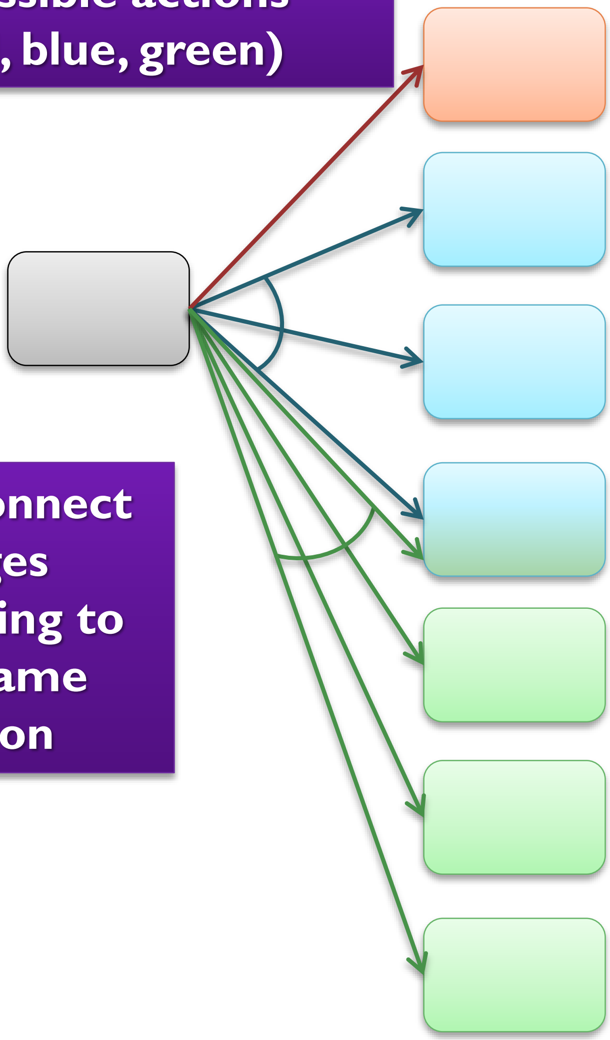
Uncertain how much fuel will be consumed

As always, one state for every **combination** of properties

Stochastic Systems (5)

- Like before, often **many executable actions** in every state

3 possible actions (red, blue, green)



Probability sum = 1 (certain outcome)

Probability sum = 1 (three possible outcomes of A2)

Probability sum = 1 (three possible outcomes of A3)

Arcs connect edges belonging to the same action

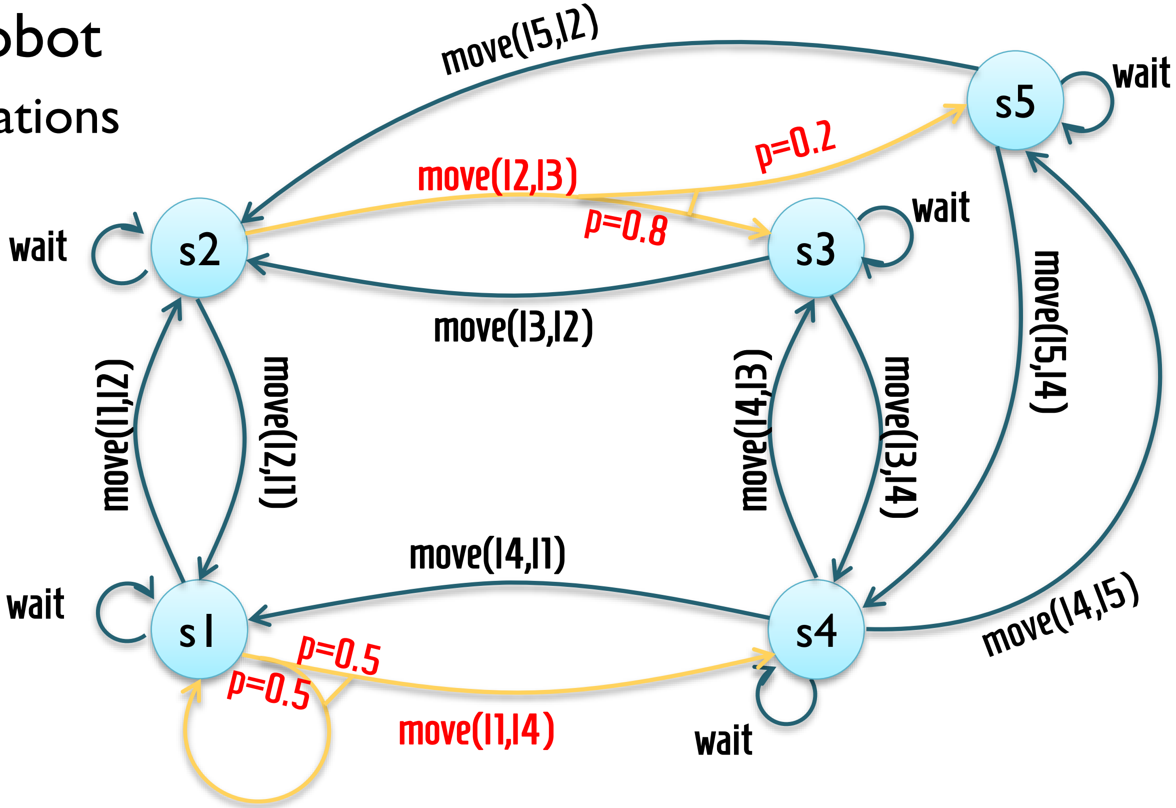
We choose the **action**…

Nature chooses the **outcome**, so we must be prepared for *all* of them!

Searching the state space yields an AND/OR tree

Stochastic System Example

- **Example**: A single robot
  - Moving between 5 locations
  - For simplicity, states correspond directly to locations
    - s1: at(r1, l1)
    - s2: at(r1, l2)
    - s3: at(r1, l3)
    - s4: at(r1, l4)
    - s5: at(r1, l5)



- Some transitions are **deterministic**, some are **stochastic**
  - Trying to move from l2 to l3: You may end up at l5 instead (20% risk)
  - Trying to move from l1 to l4: You may stay where you are instead (50% risk)

Overview

	Non-Observable: **No information gained after action**	**Fully Observable:** **Exact outcome known after action**	**Partially Observable:** **Some information gained after action**
Deterministic: **Exact outcome known in advance**	**Classical planning** (possibly with extensions) Information dimension is meaningless!		
Non-deterministic: **Multiple outcomes, no probabilities**	**NOND**: Conformant Planning	**FOND**: Conditional (Contingent) Planning	**POND**: Partially Observable, Non-Deterministic
Probabilistic: **Multiple outcomes with probabilities**	Probabilistic Conformant Planning (Non-observable MDPs: Special case of POMDPs)	Probabilistic Conditional Planning Stochastic Shortest Path Problems Markov Decision Processes (MDPs)	Partially Observable MDPs (POMDPs)

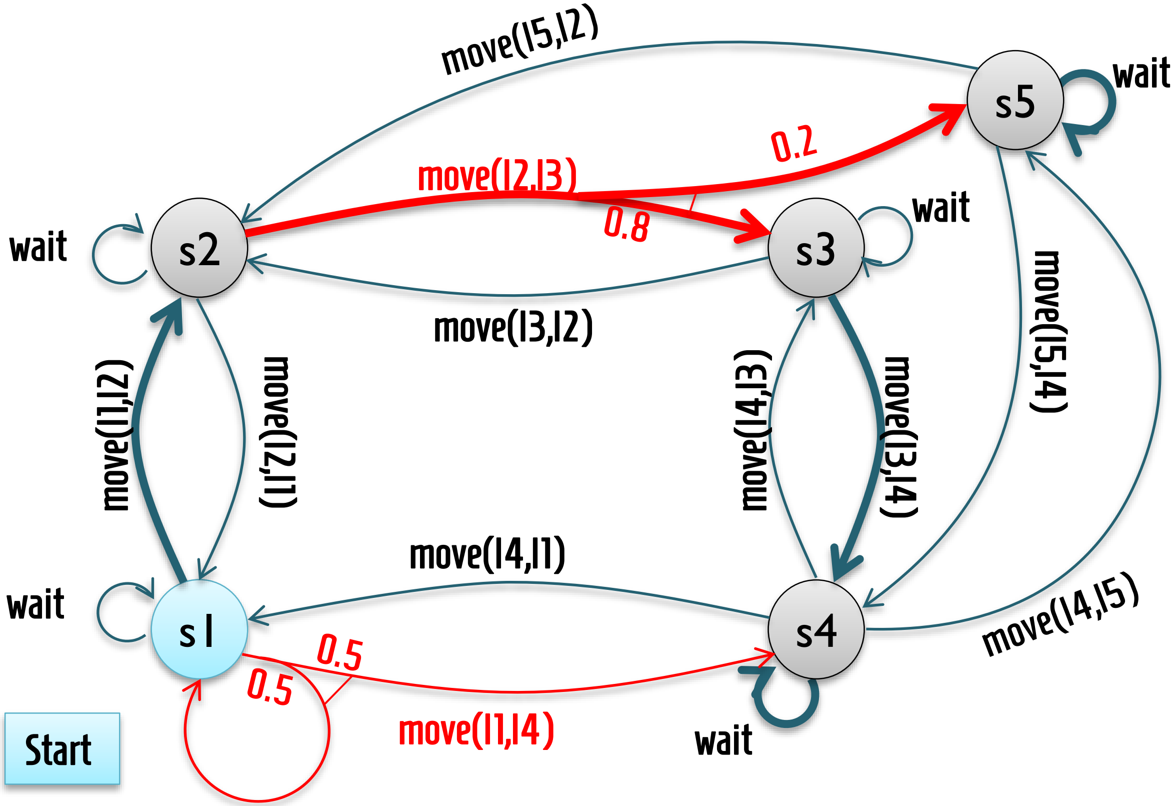
To be discussed now!

Fully Observable Probabilistic Planning: Policies and Histories

Important concepts,
before we define the planning problem itself!

Policy Example 1

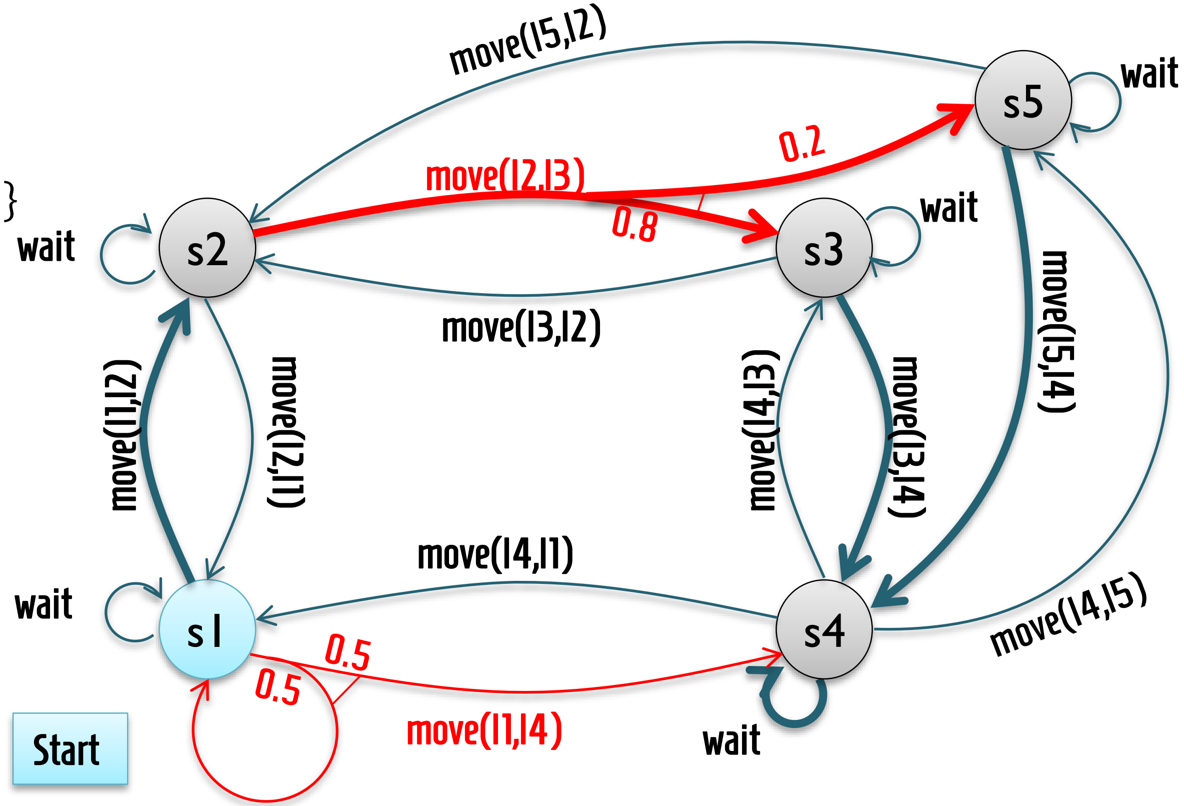
- Example 1
  - π1 = { (s1, move(l1,l2)),
    (s2, move(l2,l3)),
    (s3, move(l3,l4)),
    (s4, wait),
    (s5, wait)}



Reaches s4 or s5, waits there infinitely many times

Policy Example 2

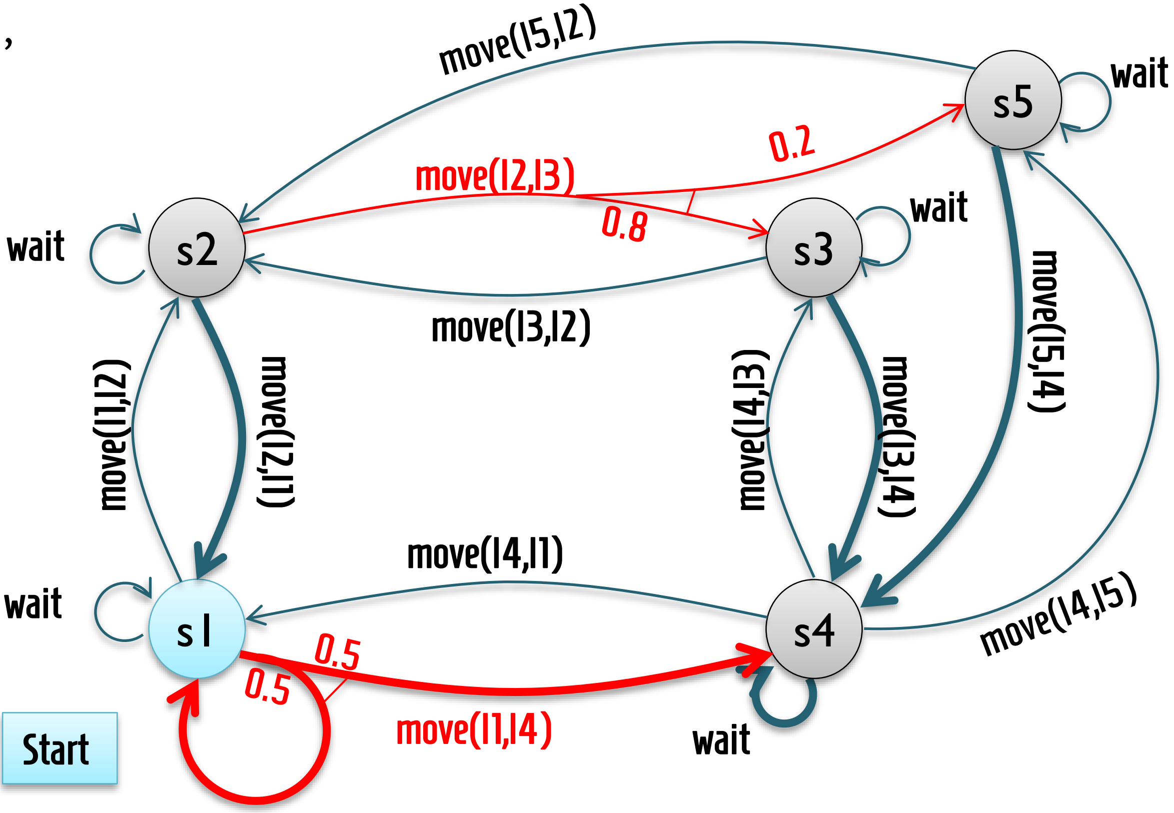
- Example 2
  - π2 = { (s1, move(l1,l2)),
    (s2, move(l2,l3)),
    (s3, move(l3,l4)),
    (s4, wait),
    (s5, **move(l5,l4)**)}



Always reaches state s4, waits there infinitely many times

Policy Example 3

- Example 3
  - π3 = { **(s1, move(l1,l4))**,
    (s2, move(l2,l1)),
    (s3, move(l3,l4)),
    (s4, wait),
    (s5, move(l5,l4)}



Reaches state s4 with 100% probability "in the limit"
(it *could* happen that you never reach s4, but the probability is 0)

Policies and Histories

- The **outcome** of sequentially executing a policy:
  - A **state sequence**, called a **history**
  - Infinite, since policies do not terminate
  - $h = \langle s_0, s_1, s_2, s_3, s_4, \dots \rangle$

s_0 (*index* zero): **Variable** used in histories, etc

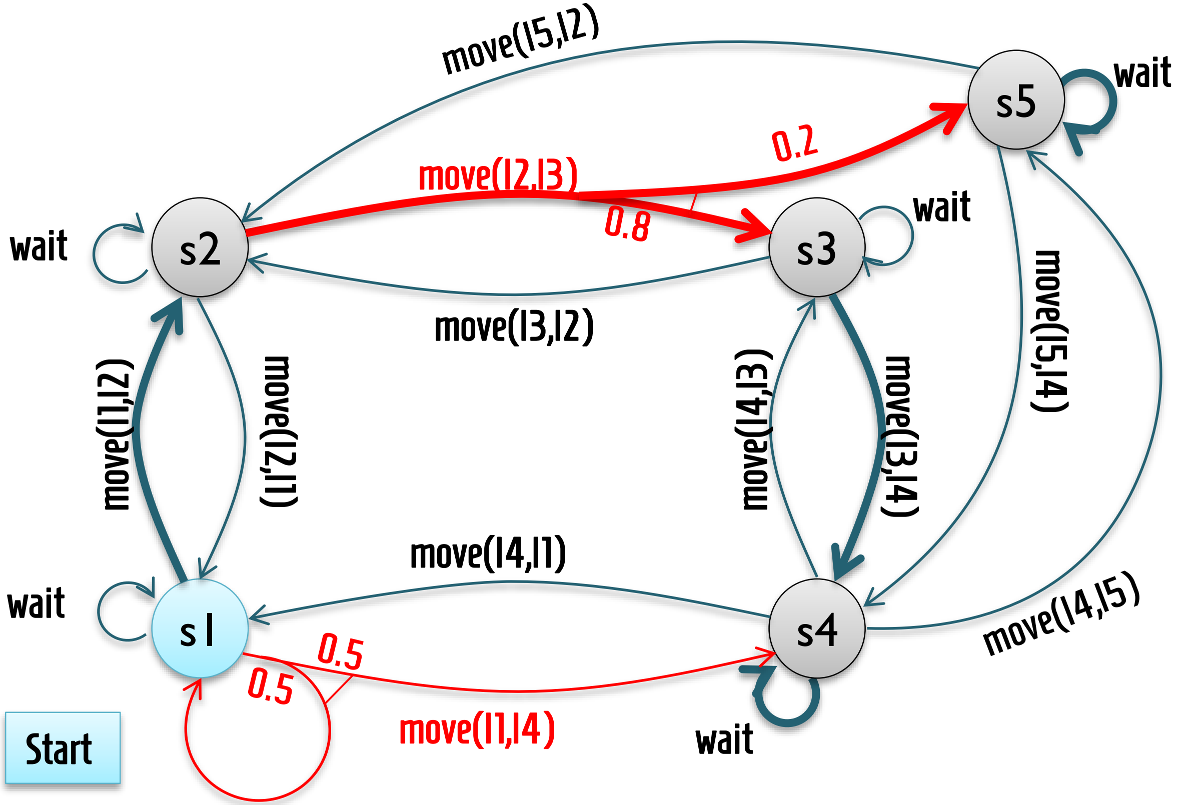
s0: **concrete** state name used in diagrams

We may have s_0 = s27

- For **classical** planning:
  - A plan yields a **single** history (last state repeated infinitely), known in advance
- For **probabilistic** planning:
  - We may not know the **initial state** with certainty
    - For every state s, there will be a **probability** $P(s)$ that we **begin** in the state s
  - **Actions** can have multiple outcomes
  - → A policy can yield **many** different histories
    - Which one? Gradually discovered at execution time!

History Example 1

- Example 1
  - $\pi1$ = { (s1, move(l1,l2)),
    (s2, move(l2,l3)),
    (s3, move(l3,l4)),
    (s4, wait),
    (s5, wait)}



- Even if we only consider starting in $s1$: Two possible histories
  - $h_1 = \langle s1, s2, s3, s4, s4, \ldots \rangle$ – Reached $s4$, waits indefinitely
    $h_2 = \langle s1, s2, s5, s5 \ldots \rangle$ – Reached $s5$, waits indefinitely

How probable are these histories?

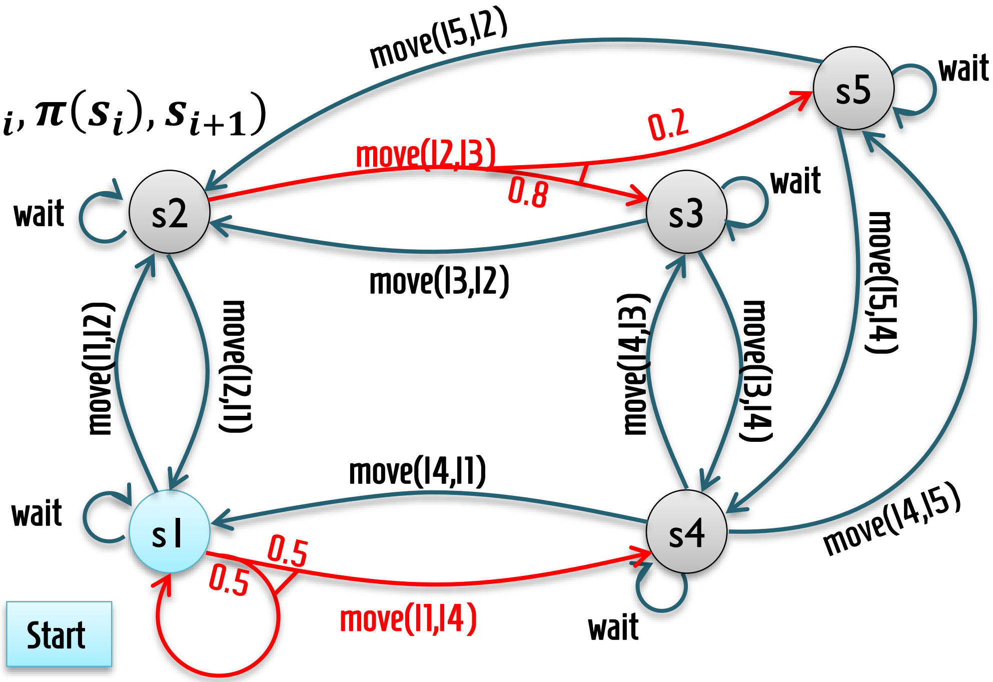
Probabilities: Initial States, Transitions

- Each policy has a **probability distribution over histories/outcomes**
  - With unknown initial state:
    - $P(\langle s_0, s_1, s_2, s_3, \ldots \rangle \mid \pi) =$

$$P(s_0) \cdot \prod_{i \geq 0} P(s_i, \pi(s_i), s_{i+1})$$

Probability of starting in this specific s_0

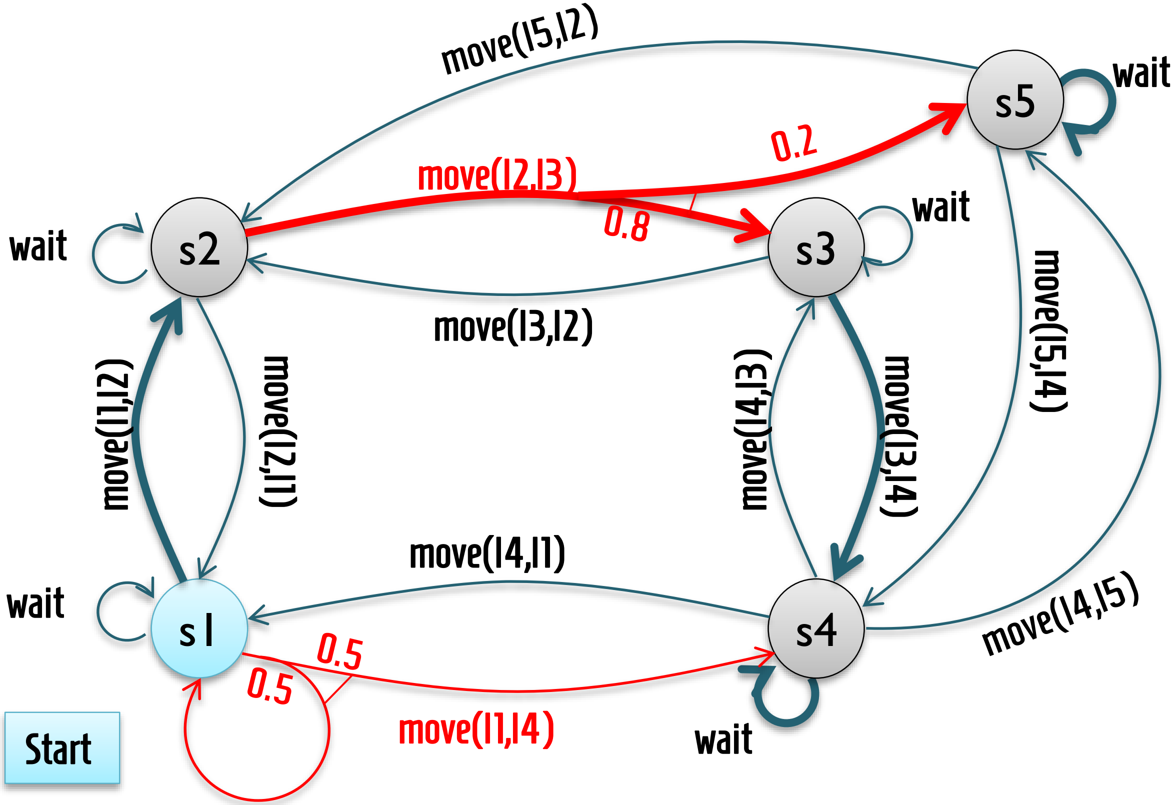
Probabilities for each required state transition



- The book:
  - Assumes you start in a known state s_0
  - So all histories start with the same state
  - $P(\langle s_0, s_1, s_2, s_3, \ldots \rangle \mid \pi) = \prod_{i \geq 0} P(s_i, \pi(s_i), s_{i+1})$ **if** s_0 is the known initial state
    $P(\langle s_0, s_1, s_2, s_3, \ldots \rangle \mid \pi) = 0$ **if** s_0 is any other state

History Example 1

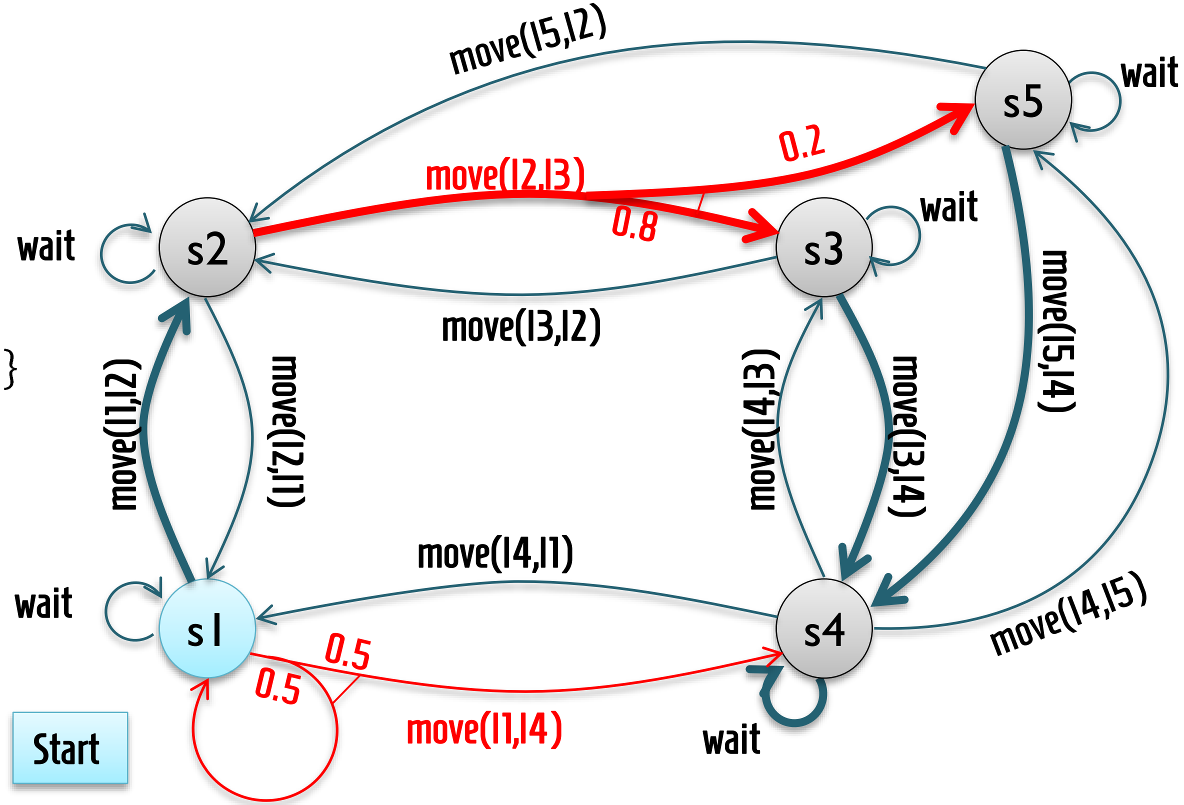
- Example 1
  - $\pi1$ = { (s1, move(l1,l2)),
    (s2, move(l2,l3)),
    (s3, move(l3,l4)),
    (s4, wait),
    (s5, wait)}



- Two possible histories, if $P(s1) = 1$:
  - $h_1 = \langle$s1, s2, s3, s4, s4, … $\rangle$ – $P(h_1 \mid \pi_1) = 1 \times 1 \times 0.8 \times 1 \times \ldots = 0.8$
    $h_2 = \langle$s1, s2, s5, s5 … $\rangle$ – $P(h_2 \mid \pi_1) = 1 \times 1 \times 0.2 \times 1 \times \ldots = 0.2$
    – $P(h \mid \pi_1) = 1 \times 0 = 0$ for all other h

History Example 2

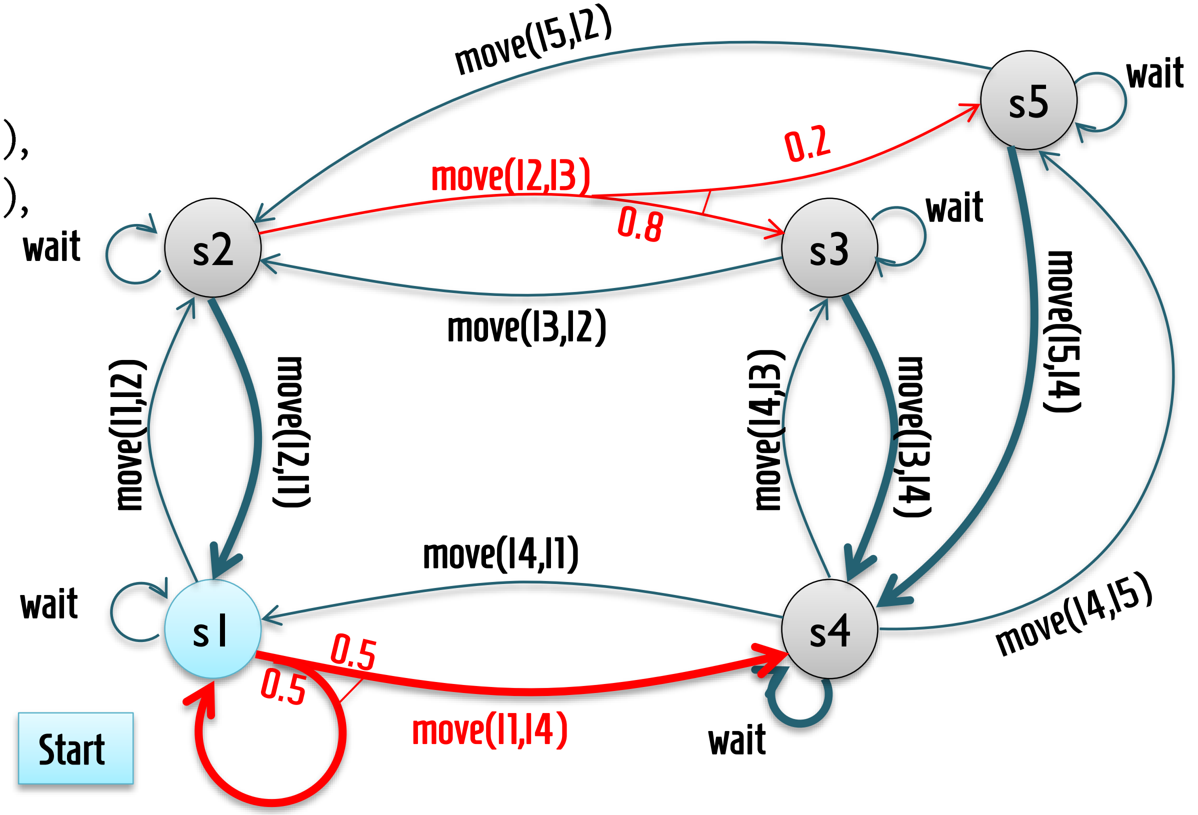
- Example 2
  - $\pi 2$ = { (s1, move(l1,l2)),
    (s2, move(l2,l3)),
    (s3, move(l3,l4)),
    (s4, wait),
    (s5, **move(l5,l4)**)}



- $h_1 = \langle s1, s2, s3, \boxed{s4, s4, \ldots} \rangle$ $\quad P(h_1 \mid \pi_2) = 1 \times 1 \times 0.8 \times 1 \times \ldots = 0.8$
  $h_3 = \langle s1, s2, s5, \boxed{s4, s4, \ldots} \rangle$ $\quad P(h_3 \mid \pi_2) = 1 \times 1 \times 0.2 \times 1 \times \ldots = 0.2$
  $P(h \mid \pi_2) = 1 \times 0$ for all other h

History Example 3

- Example 3
  - $\pi 3$ = { **(s1, move(l1,l4))**,
    (s2, move(l2,l1)),
    (s3, move(l3,l4)),
    (s4, wait),
    (s5, move(l5,l4)}



- $h_4 = \langle s1, s4, s4, \ldots \rangle$ $\quad P(h_4 \mid \pi_3) = 0.5 \times 1 \times 1 \times 1 \times 1 \times \ldots = 0.5$

  $h_5 = \langle s1, s1, s4, s4, \ldots \rangle$ $\quad P(h_5 \mid \pi_3) = 0.5 \times 0.5 \times 1 \times 1 \times 1 \times \ldots = 0.25$

  $h_6 = \langle s1, s1, s1, s4, s4, \ldots \rangle$ $\quad P(h_6 \mid \pi_3) = 0.5 \times 0.5 \times 0.5 \times 1 \times 1 \times \ldots = 0.125$

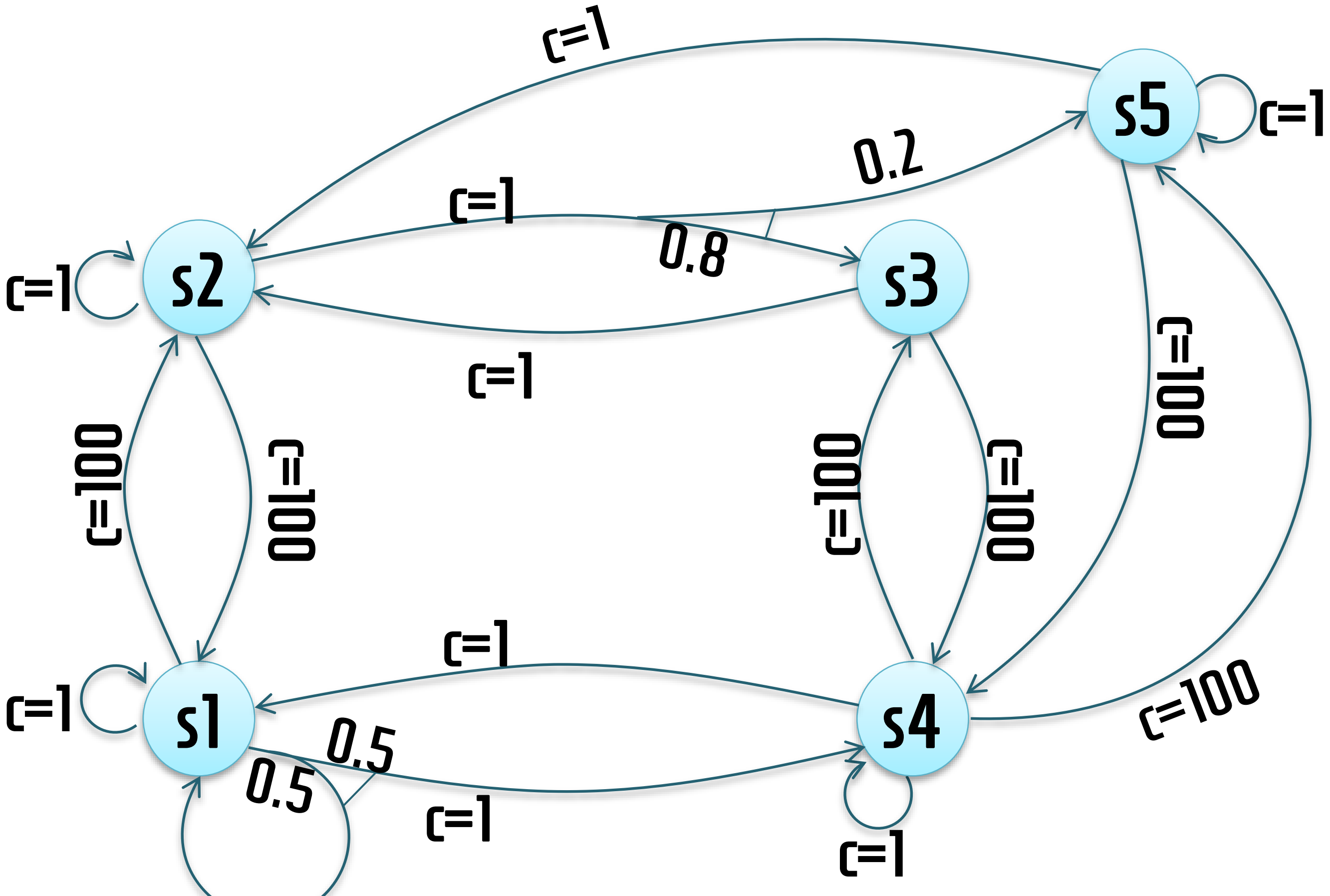
  …

  $h_\infty = \langle s1, s1, s1, s1, s1, s1, \ldots \rangle$ $\quad P(h_\infty \mid \pi_3) = 0.5 \times 0.5 \times 0.5 \times 0.5 \times 0.5 \times \ldots = 0$

Costs and Expected Costs

Cost of an Action

- Part of the specification: A **cost function** $c(s,a)$
  - Representing the known cost of executing a in state s
  - $c(s,a) = 1$ for each “horizontal” action
  - $c(s,a) = 100$ for each “vertical” action: Far away, difficult, …
  - $c(s, wait) = 1$



Cost of a History

- Assume as given:
  - A policy π
  - An outcome, an infinite history $h = \langle s_0, s_1, \ldots \rangle$ resulting from executing π

  - We can then calculate the **cost of execution** for the given **history / outcome**:

$$C(h|\pi) = \sum_{i \geq 0} c\big(s_i, \pi(s_i)\big)$$

Given what happened,
this is how much it cost us!

”Cost of history given policy”:
Using the same actions *in different states* → different cost!
Using *other actions* to reach the same states → different cost!

Expected Cost of a Policy

- We want to choose a good = ”cheap” **policy**
  - Actual cost depends on outcome, which we **can’t** choose
  - For **each** possible history (outcome), we can calculate:
    - The *probability* that the history will occur
    - The resulting *cost*
  - So: calculate the statistically **expected cost** (~"average" cost) for the entire **policy**:

$$E_C(\pi) = \sum_{h \in \{\text{all possible histories for } \pi\}} P(h|\pi)C(h|\pi)$$

  - Later, we will calculate costs without the need to explicitly find all histories – examples then!

Stochastic Shortest Path Problems

Stochastic Shortest Path Problem

- Closest to *classical* planning: **Stochastic Shortest Path Problem**
  - Let $\Sigma = (S, A, P)$ be a stochastic system
  - Let $c: (S, A) \to R$ be a cost function
  - Let $s_0 \in S$ be an **initial state**
  - Let $S_g \subseteq S$ be a **set of goal states**

  - Then, find a **policy of minimal *expected* cost**
    that can be applied starting at s_0
    and that **reaches** a state in S_g with probability 1

Stochastic outcomes →
only expected costs can be calculated

Probability 1: "Infinitely unlikely"
that we don't reach a goal state

SSPP: Termination?

- But **policies never terminate**!
  - Even in a goal state, $\pi(s)$ specifies an action to execute
  - Histories are infinitely long
  - → Cost calculations include infinitely many actions!

- **Why** define policies this way, when we *do* want to stop at the goal?
  - We are using more general "machinery"
    that is ***also*** used for ***non-terminating*** execution!

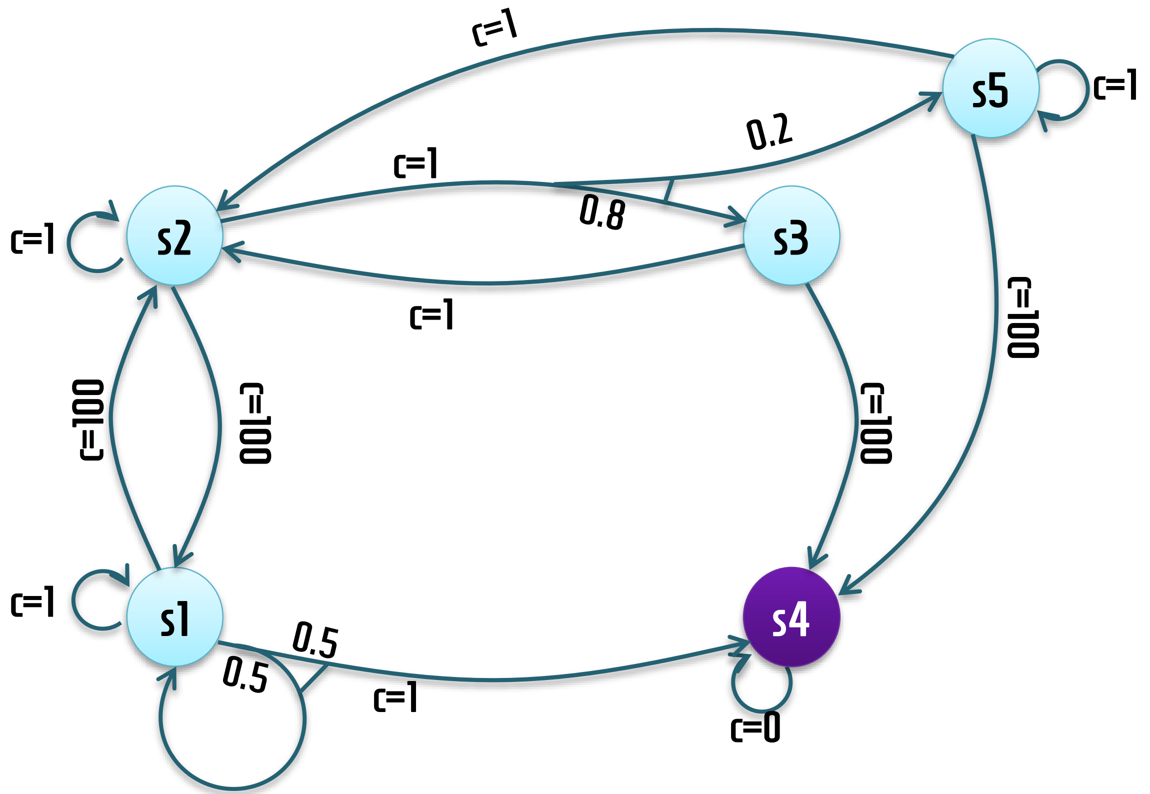
SSPP: Absorbing Goal State

- How to **solve** the problem?
  - Make every goal state g **absorbing** – state s4 below
    - For every action a,

      $P(g, a, g) = 1$ ➔ returns to the same goal state (we'll stop anyway)

      $c(g, a) = 0$ ➔ no more cost accumulates
  - Solve the problem using *general* methods, generate a policy

- How to **execute**?
  - Follow the policy
  - When you reach a goal state, stop!



Utility Functions and SSPP

- **The SSPP:**
  - Strictly positive action cost (>0) except in goal states (=0)
  - If infinite history h **visits a goal state**, it consists of:
    - Finitely many actions of finite positive cost
    - Followed by infinitely many actions *of cost 0*
    - ➔ Finite total cost
  - If infinite history h *does not visit a goal state*:
    - Infinitely many actions of strictly positive cost
    - ➔ Infinite total cost
  - If *any* history that does not visit a goal state has *non-zero* probability:

$$E_C(\pi) = \sum_{h \in \{\text{all possible histories for } \pi\}} P(h|\pi)C(h|\pi) = \infty$$

Policy π
has finite expected cost
➔
π visits a goal state
with probability 1
➔
π solves the SSPP

Stochastic Shortest Path Problems: Domain Examples

Action Representations and PPDDL

- **Action representations**:
  - The book only deals with the **underlying semantics**: "Unstructured" probability distribution $P(s, a, s')$
  - Several "convenient" representations possible, such as Bayes networks, probabilistic operators

- **Probabilistic PDDL**: new constructs for effects, initial state
  - (probabilistic p_1 e_1 … p_k e_k)
    - Effect e_1 takes place with probability p_1, etc.
    - **Sum** of probabilities = $s \leq 1$ ($s < 1$➔ with probability $1 - s$, nothing happens)

Tire World

- Tire may go flat – good idea to load a spare from the start…

```
(:action move-car :parameters (?from - location ?to - location)
    :precondition (and (vehicle-at ?from) (road ?from ?to)
                       (not (flattire)))
    :effect (and (vehicle-at ?to) (not (vehicle-at ?from))
           (increase (cost) 1)  (probabilistic .15 (flattire))))
```

```
(:action changetire
    :precondition (and (vehicle-has-spare) (flattire))
    :effect (and (increase (cost) 1)
                       (not (vehicle-has-spare))    (not (flattire))))
```

```
(:action loadspare :parameters (?loc - location)
    :precondition (and (vehicle-at ?loc) (spare-at ?loc)
                              (not (vehicle-has-spare)))
    :effect (and (vehicle-has-spare) (not (spare-at ?loc))
                         (increase (cost) 1)))
```

```
(:action callAAA
    :precondition (flattire)
    :effect  (and (increase (cost) 100)
                       (not (flattire))))
```

Variation of SSPP: Achieve a goal, be at X, *at minimum expected cost*

You can bring <u>one</u> spare tire, but what if you need more?

Some locations provide spare tires – affects where you should go in the road network

Spares have a cost, but you may still want to load one to handle *potential* flat tires

Can manage without a spare, but then you must call the AAA (*tow truck*) which is expensive

SSPP variations

- A **variation** of the Stochastic Shortest Path Problem:
  - Let $\Sigma = (S, A, P)$ be a stochastic system
  - Let $s_0 \in S$ be an **initial state**
  - Let $S_g \subseteq S$ be a **set of goal states**
  - (Ignore the cost function)
  - Then, find a **policy** (not "*of minimal expected cost*")
    that can be applied starting at s_0
    and that **reaches** a state in S_g with **maximum probability**

Representation Example: PPDDL

Bomb-and-toilet problem

```
(define (domain bomb-and-toilet)
        (:requirements :conditional-effects :probabilistic-effects)
        (:predicates (bomb-in-package ?pkg) (toilet-clogged) (bomb-defused))
        (:action dunk-package
            :parameters (?pkg)
            :effect (and
                (when (bomb-in-package ?pkg) (bomb-defused))
                (probabilistic 0.05 (toilet-clogged)))))
```

First, a "standard" effect

5% chance of toilet-clogged, 95% chance of no effect

```
(define (problem bomb-and-toilet)
        (:domain bomb-and-toilet)
        (:requirements :negative-preconditions)
        (:objects package1 package2)
        (:init (probabilistic   0.5 (bomb-in-package package1)
                                0.5 (bomb-in-package package2)))
        (:goal (and (bomb-defused) (not (toilet-clogged)))))
```

Probabilistic initial state

Goal – no plan guarantees satisfaction; might maximize probability

Ladder

-
  ```
  (define (problem climber-problem)
    (:domain climber)
    (:init (on-roof) (alive) (ladder-on-ground))
    (:goal (and (on-ground) (alive))))
  ```

-
  ```
  (define (domain climber)
    (:requirements :typing :strips :probabilistic-effects)
    (:predicates (on-roof) (on-ground)
                 (ladder-raised) (ladder-on-ground) (alive))
  ```

-
  ```
  (:action climb-without-ladder :parameters ()
     :precondition (and (on-roof) (alive))
     :effect (and (not (on-roof))  (on-ground)
                     (probabilistic 0.4 (not (alive)))))
  ```

-
  ```
  (:action climb-with-ladder :parameters ()
     :precondition (and (on-roof) (alive) (ladder-raised))
     :effect (and (not (on-roof)) (on-ground)))
  ```

-
  ```
  (:action call-for-help :parameters ()
     :precondition (and (on-roof) (alive) (ladder-on-ground))
     :effect (and (not (ladder-on-ground)) (ladder-raised))))
  ```

;; Sylvie Thiébaux + Iain Little

You are **stuck on a roof** because the ladder you climbed up on fell down.

There are plenty of people around; if you call out for help **someone will certaintly lift the ladder up** again.

Or you can try the **climb down without it**.

You aren't a very good climber though, so there is a 40% chance that you will fall and **break your neck** if you go it alone.

What do you do?

Exploding Blocks World

- When you **stack/putdown** an **undetonated** block:
  - 30% probability that it detonates, **destroying** what is **below** it

    -
      ```
      (:action put-down-block-on-table
                   :parameters (?b - block)
                   :precondition (and (holding ?b) (not (destroyed-table)))
                   :effect (and (not (holding ?b))
                                (ontable ?b)
                                (when (not (detonated ?b))
                                      (probabilistic .3 (and (detonated ?b)
                                                             (destroyed-table))
                                ))))
      ```
  - **Solutions use unneeded blocks as potential "sacrifices"**
    - Repeat placing required blocks there
      until they detonate, destroying the unneeded blocks
    - Ordering is important: Some unneeded blocks are *not clear*, must be freed
    - Strategy of *replanning after unexpected events* won't work: Needed blocks are gone!
    - *https://www.aaai.org/Papers/JAIR/Vol24/JAIR-2421.pdf*

Beyond SSPP:
Rewards for Indefinite Execution

Generalizating from the SSPP

- We have defined the **Stochastic Shortest Path Problem**
  - Similar to the classical planning problem,
    but adapted to probabilistic outcomes

- But policies allow *indefinite execution*
  - No predetermined termination criterion – go on "forever"
  - Can we **exploit** this fact to **generalize** from SSPPs?

Yes – remove the goal states, assume no termination

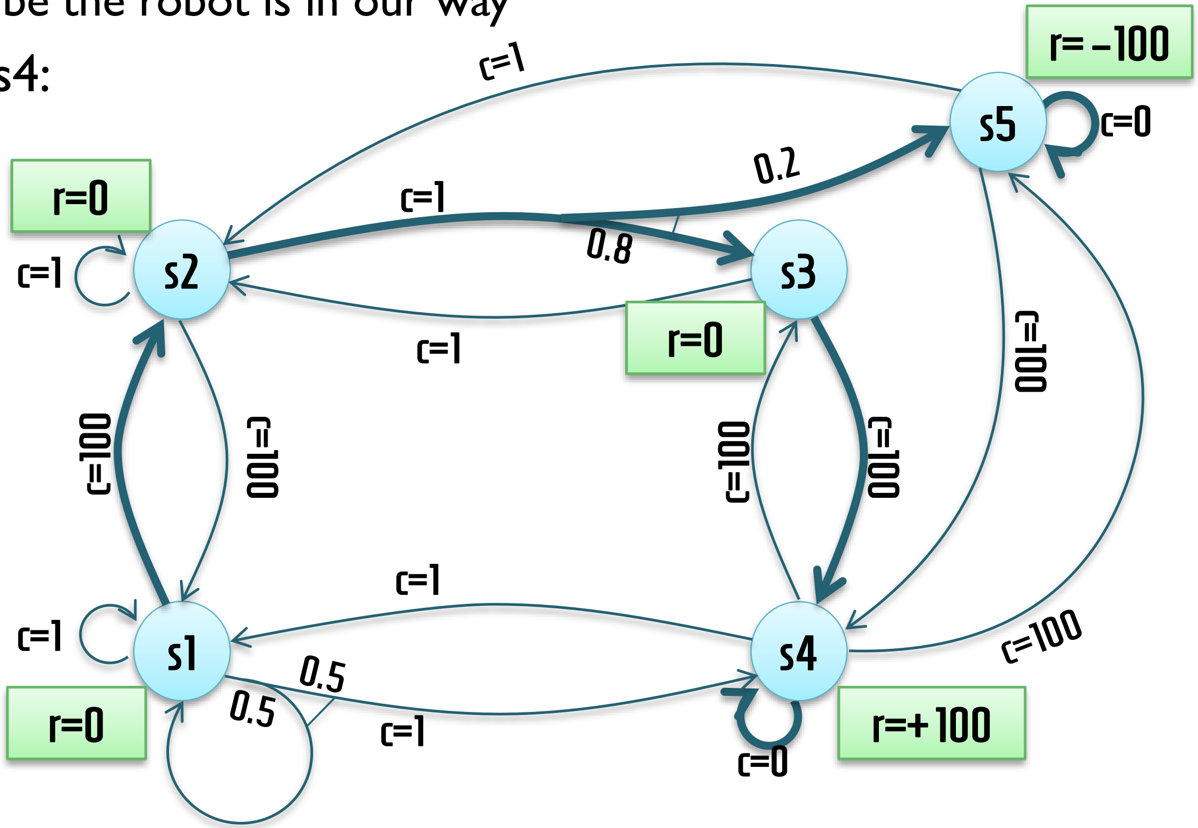
But without goal states, what is the objective?

Goals → Rewards

- How to determine what's a **good** policy?
  - Introduce ***rewards*** that can be ***accumulated*** during execution!
  - **Reward function** $R(s, a, s')$
    - Reward gained for **being** in s, **executing** action a and **ending up** in s'
    - Can be negative!

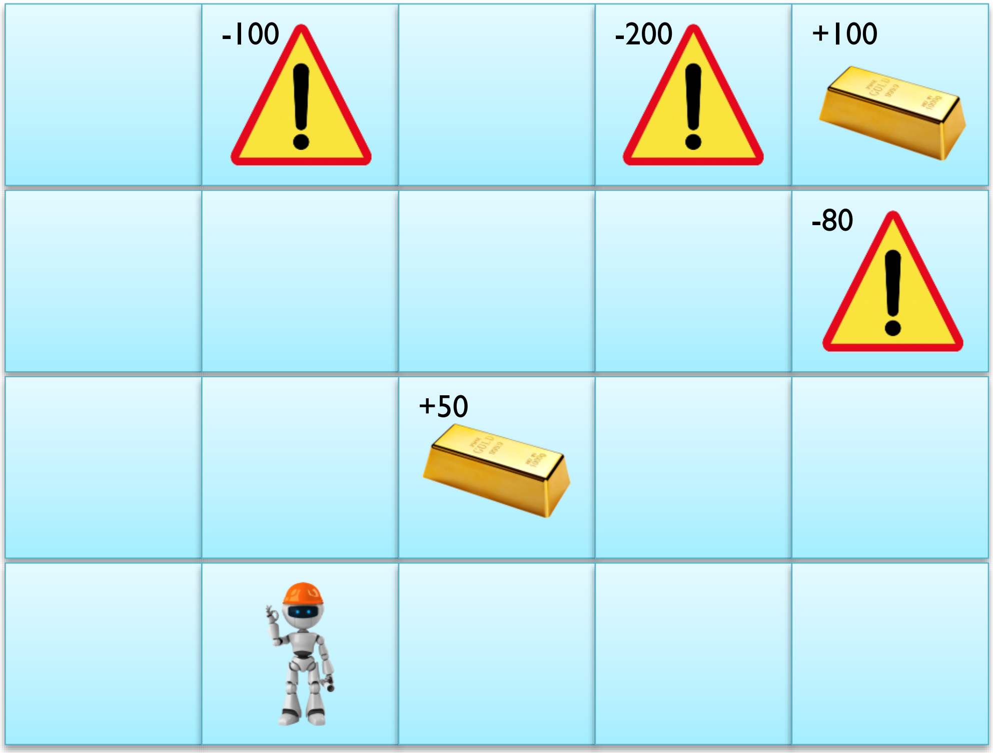
Rewards: Robot Navigation

- Example:
  - The robot does not "want to reach s4"
  - It wants to ***execute actions to gain rewards***
  - Every time step it is in s5:
    - Negative reward – maybe the robot is in our way
  - Every time step it is in s4:
    - Positive reward – maybe it helps us and "gets a salary"



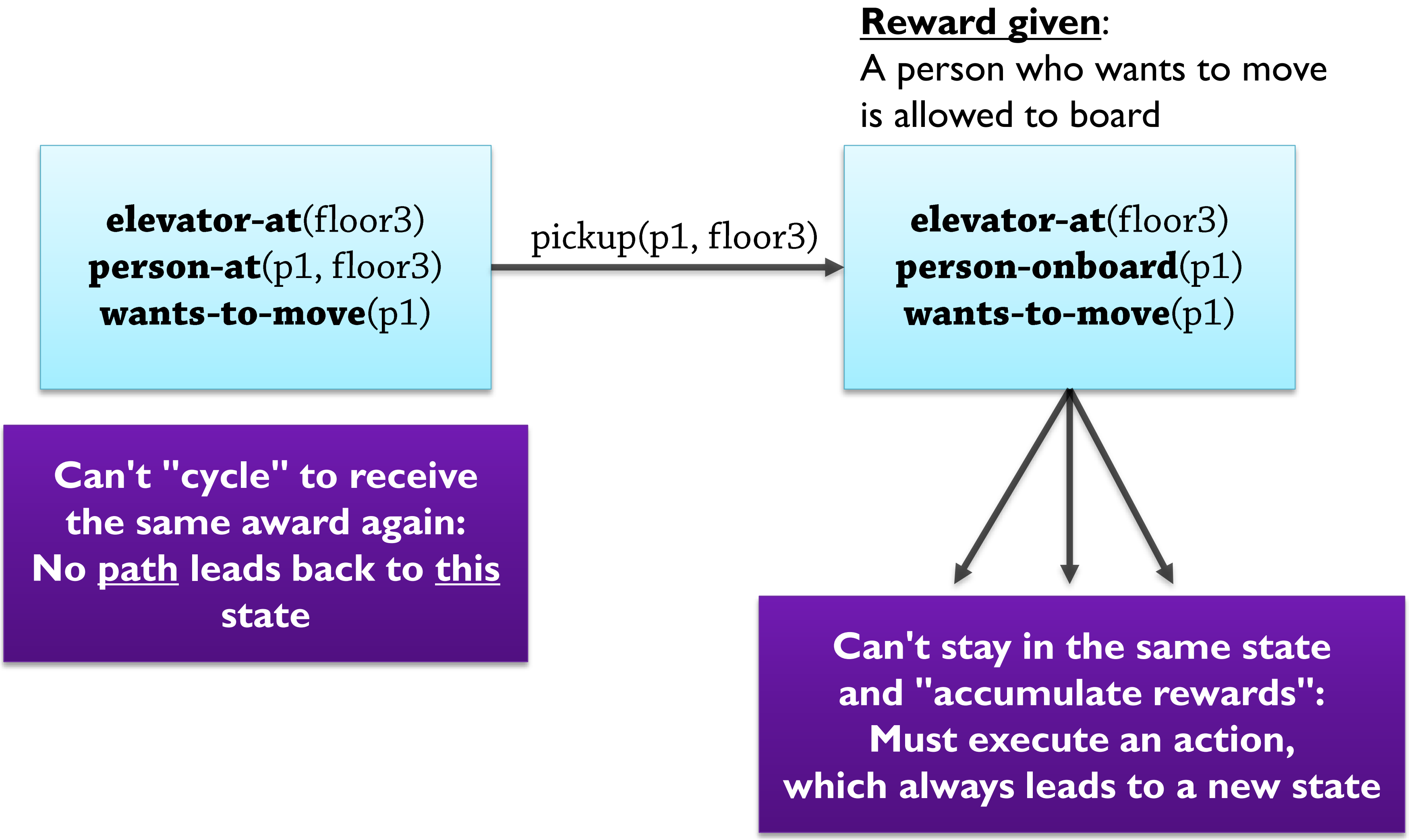
Rewards: Grid World

- Example: **Grid World**
  - **Actions**: North, South, West, East, NorthWest, …
    - Associated with a cost
    - 90% probability of doing what you want
    - 10% probability of moving to another cell
  - **Rewards** in some cells
    - $R(s, a, s') = +100$ for transitions where you end up in the top right cell
  - **Danger** in some cells
    - $R(s, a, s') = -200$ for transitions where you end up in the neighbor cell
  - The same action *may* give $+100$, *may* give -200!



States, not Locations

- **Important**: States != locations



Reward given:
A person who wants to move
is allowed to board

elevator-at(floor3)
person-at(p1, floor3)
wants-to-move(p1)

pickup(p1, floor3)

elevator-at(floor3)
person-onboard(p1)
wants-to-move(p1)

Can't "cycle" to receive the same award again: No path leads back to this state

Can't stay in the same state and "accumulate rewards": Must execute an action, which always leads to a new state

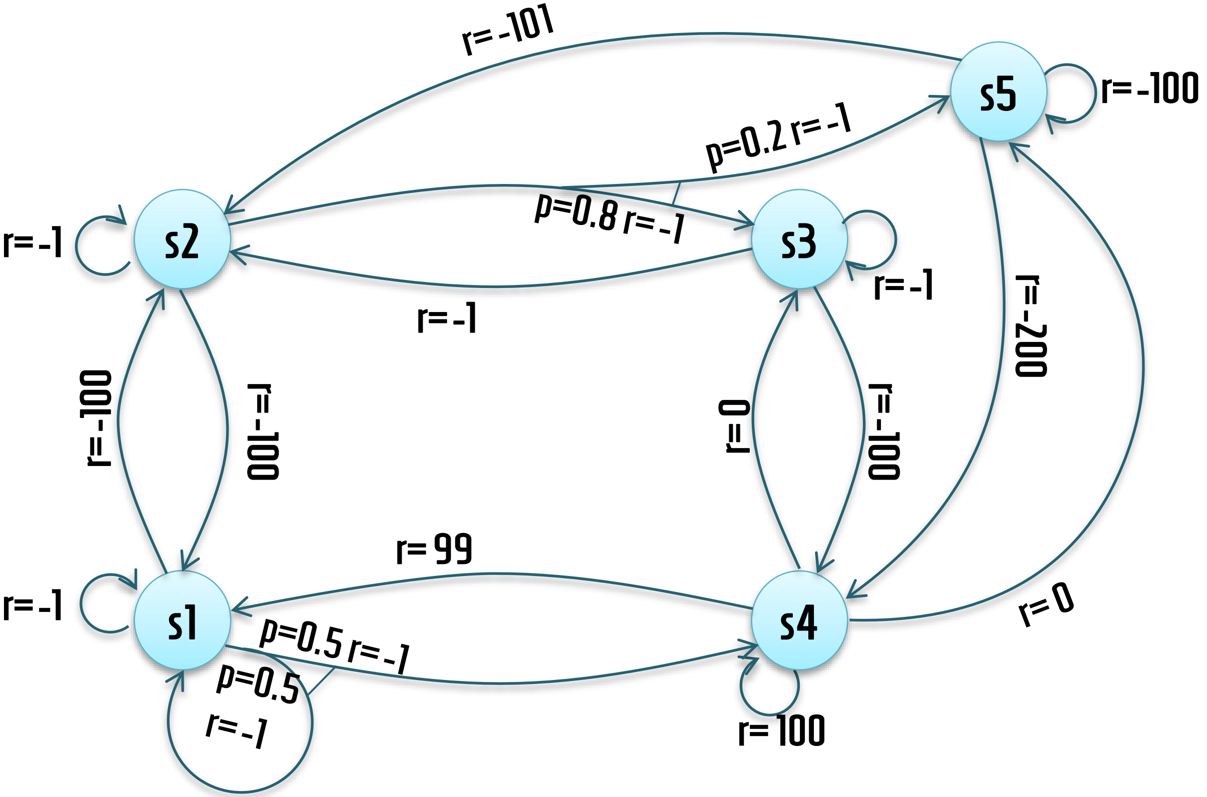
Simplification

- To simplify formulas, **include the cost in the reward**!
  - Decrease each $R(s_i, \pi(s_i), s_{i+1})$ by $C(s_i, \pi(s_i))$

C(s0, takeoff) = 80
R(s0, takeoff, s1) = 200
R(s0, takeoff, s2) = –100

↓

R(s0, takeoff, s1) = 120
R(s0, takeoff, s2) = –180



Utility Functions and Discount Factors

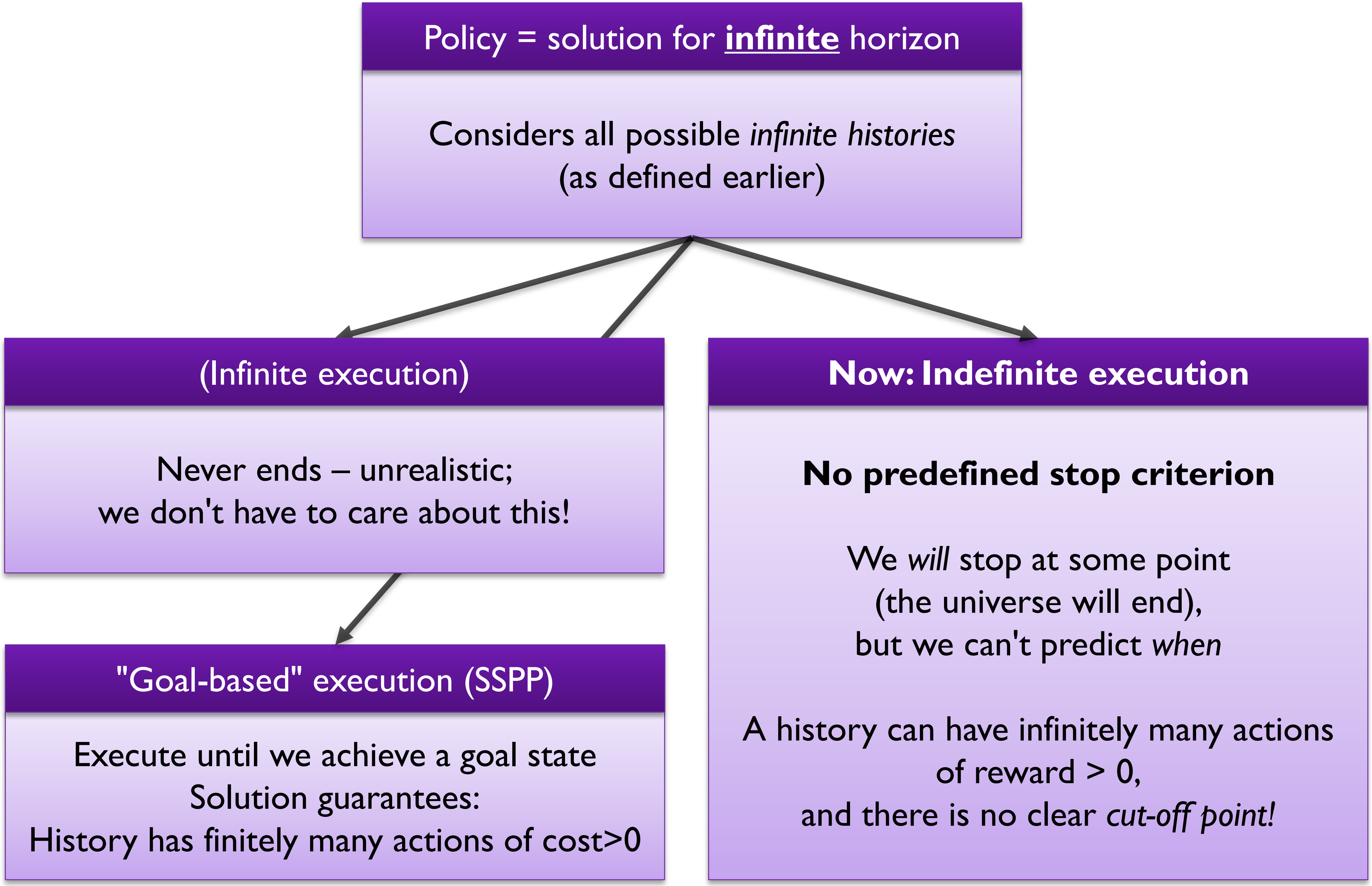
Utility Functions

- Cost→reward, cost function → **utility function**
  - Suppose a policy has one particular outcome
    ➔ results in one particular **history** (state sequence)
  - How ”useful / valuable" is **this** outcome to us? What is our **reward**?
- First: **Un-discounted utility**
  - $h = \langle s_0, s_1, \ldots \rangle$ ➔ $V(h|\pi) = \sum_{i \geq 0} R(s_i, \pi(s_i), s_{i+1})$

Un-discounted utility
of history *h*
given policy π

The **reward**
for step *i*

Utility in a Context



Policy = solution for infinite horizon

Considers all possible *infinite histories*
(as defined earlier)

(Infinite execution)

Never ends – unrealistic;
we don't have to care about this!

"Goal-based" execution (SSPP)

Execute until we achieve a goal state
Solution guarantees:
History has finitely many actions of cost>0

Now: Indefinite execution

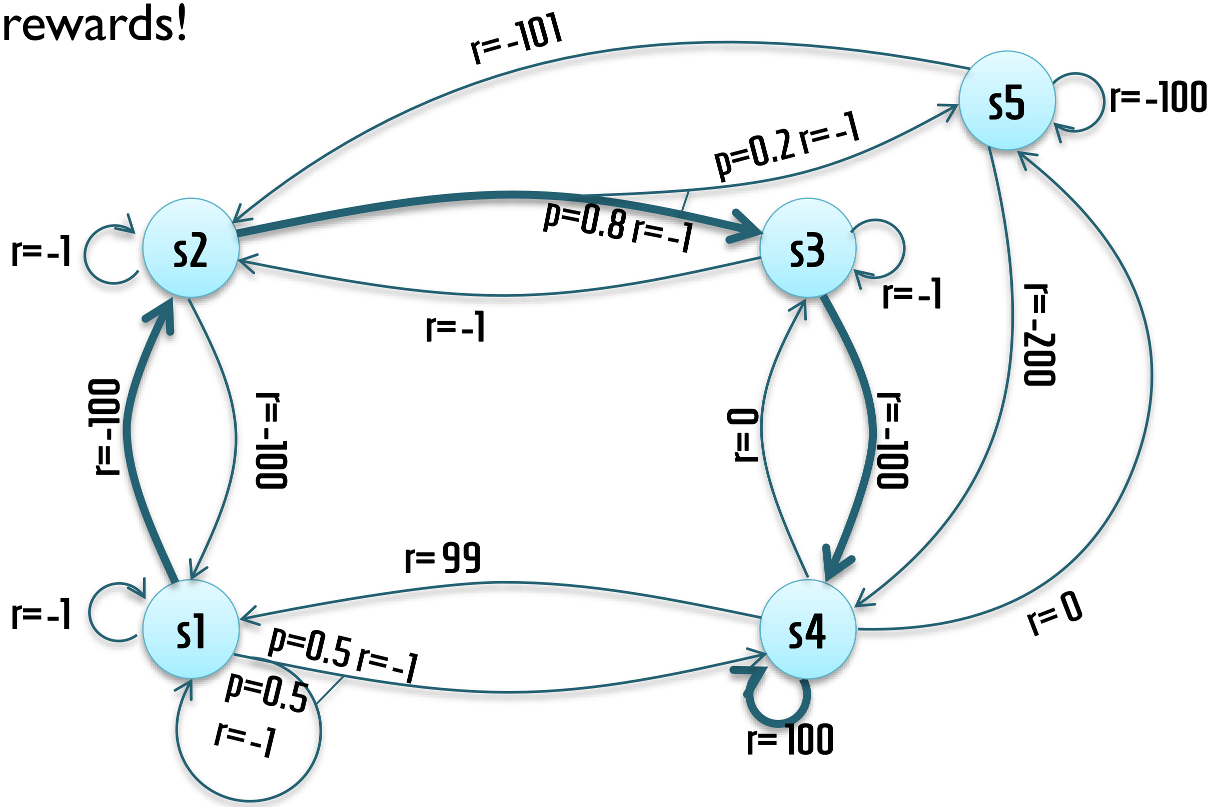
No predefined stop criterion

We *will* stop at some point
(the universe will end),
but we can't predict *when*

A history can have infinitely many actions
of reward > 0,
and there is no clear *cut-off point!*

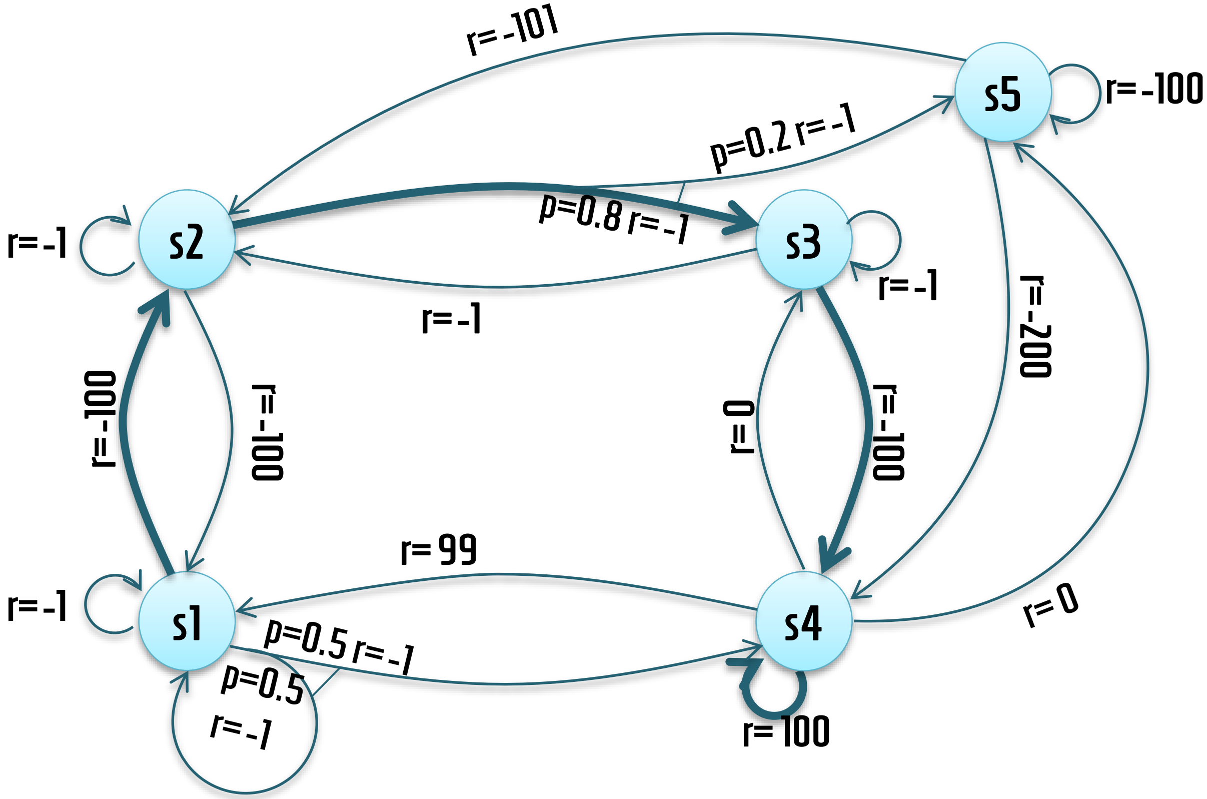
Infinite Undiscounted Utility

- Leads to problems:
  - π_1 could result in $h_1 = \langle s1, s2, s3, s4, s4, \ldots \rangle$
  - Using undiscounted utility:
    $V(h_1 \mid \pi_1) = (-100) + (-1) + (-100) + 100 + 100 + 100 + 100 + 100 + \ldots$
  - Stays at $s4$ forever, executing "wait"
    ➔ **infinite** amount of rewards!



Infinite Undiscounted Utility (2)

- **What's the problem**, given that we "like" being in state s4?
  - We can't distinguish between different ways of getting there!
    - s1→s2→s3→s4: -201 + ∞ = ∞
    - s1→s2→s1→s2→s3→s4: -401 + ∞ = ∞
    - Both appear equally good…



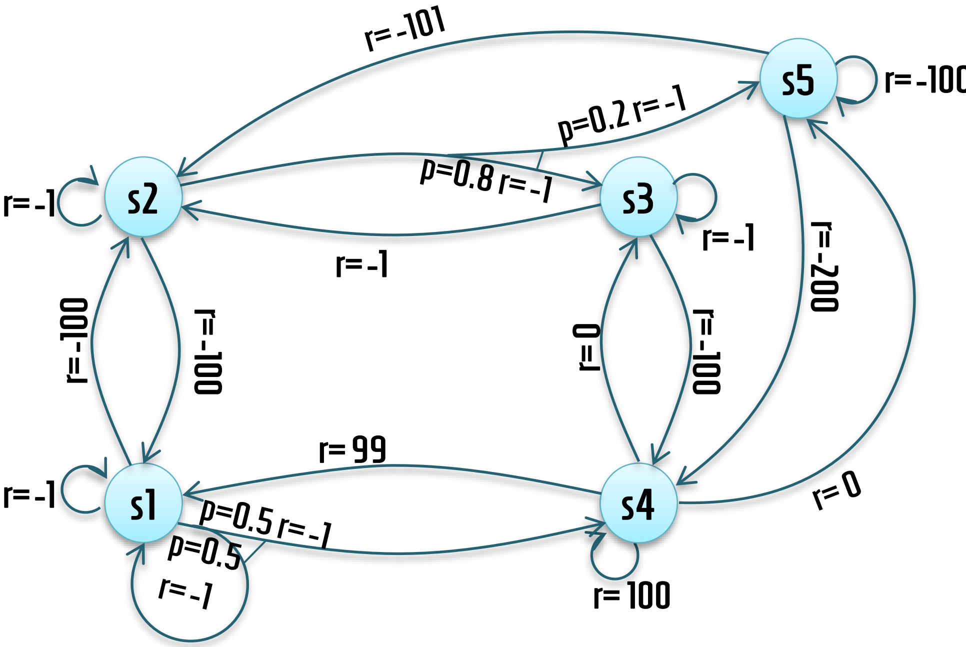
Discounted Utility

- Solution: Use a **discount factor**, γ, with $0 \leq \gamma \leq 1$
  - To avoid infinite utilities $V(\dots)$
  - To model "impatience":
    rewards and costs far in the **future** are **less important** to us

- **Discounted utility** of a history:
  - $V(h|\pi) = \sum_{i \geq 0} \gamma^i R(s_i, \pi(s_i), s_{i+1})$
  - Distant rewards/costs
    have **less influence**
  - **Convergence** (finite results)
    is guaranteed if $0 \leq \gamma < 1$

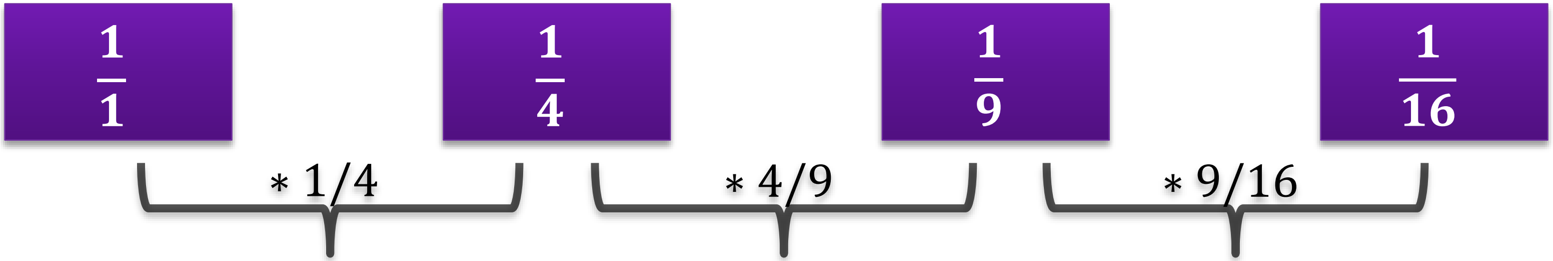
Examples will use $\gamma = 0.9$

Only to simplify formulas!
Should choose carefully…

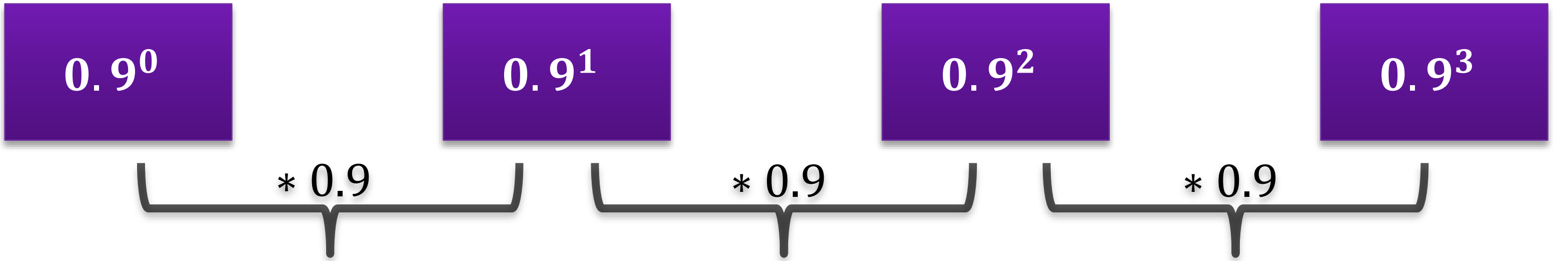


Discounted Utility (2)

- Why γ^i and not (for example) $\frac{1}{i^2}$?



Different factors → need a unique ***starting point*** to know where you are



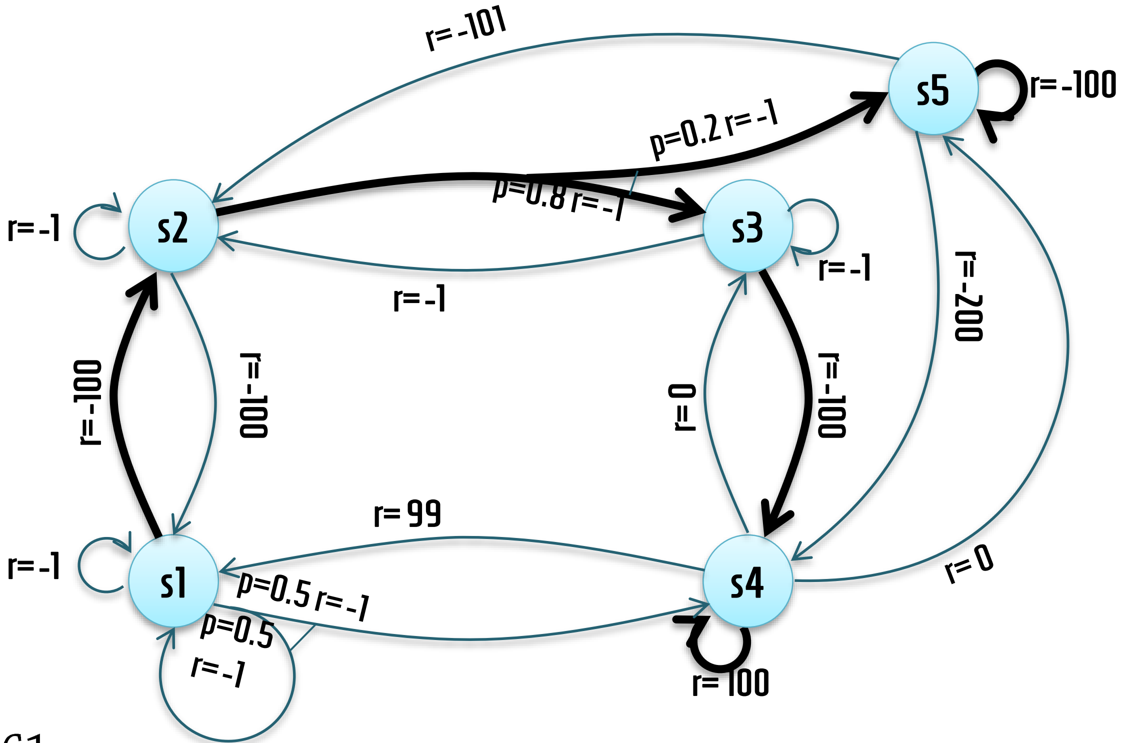
Same factor → doesn't matter where you start

→ The best action to take doesn't depend on how many actions you already took

Example

$\pi_1 = \{(s1, move(l1,l2)),$
$(s2, move(l2,l3)),$
$(s3, move(l3,l4)),$
$(s4, wait),$
$(s5, wait)\}$

Given that we start in s1,
π_1 can lead to only **two** histories:
80% chance of history h1,
20% chance of history h2



$\gamma = 0.9$

Factors 1, 0.9, 0.81, 0.729, 0.6561…

$h_1 = \langle s1, s2, s3, s4, s4, \ldots \rangle$

$V(h_1 \mid \pi_1) = .9^0(-100) + .9^1(-1) + .9^2(-100) + .9^3\,100 + .9^4\,100 + \ldots = 547.9$

$h_2 = \langle s1, s2, s5, s5 \ldots \rangle$

$V(h_2 \mid \pi_1) = .9^0(-100) + .9^1(-1) + .9^2(-100) + .9^3(-100) + \ldots = -910.1$

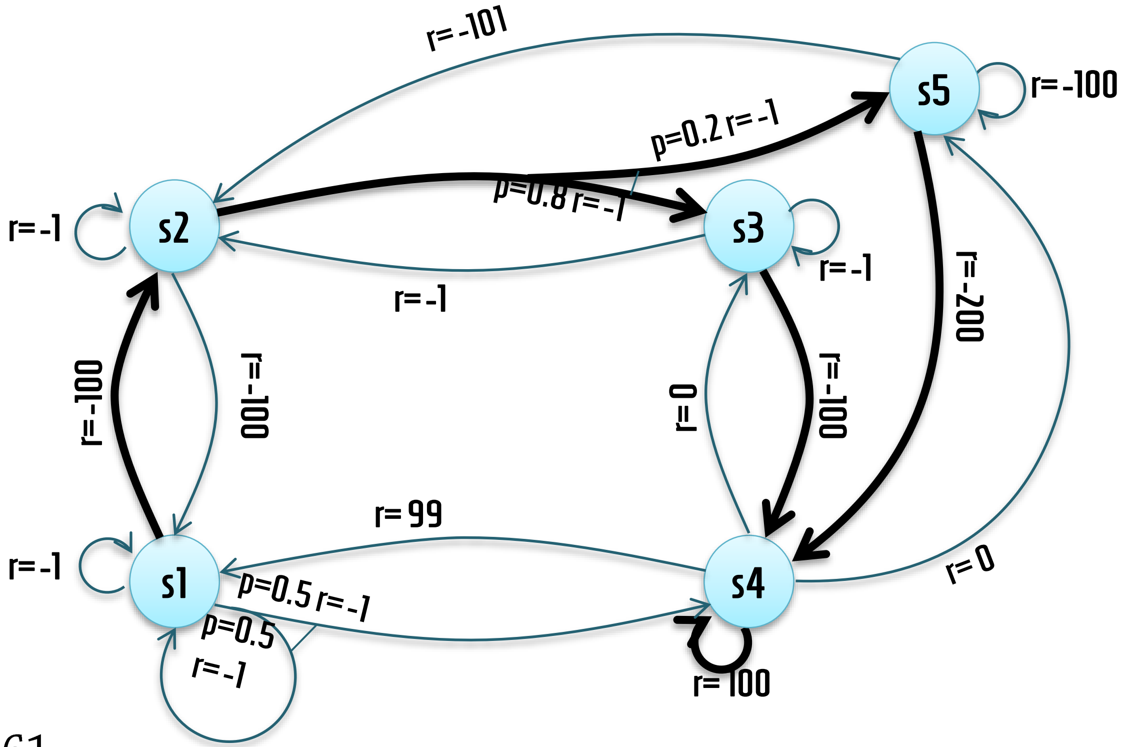
$E(\pi_1) = 0.8 * \boxed{547.9} + 0.2\,\boxed{(-910.1)} = 256.3$

We expect a reward of 256.3 on average

Example

π_2 = {(s1, move(l1,l2)),
(s2, move(l2,l3)),
(s3, move(l3,l4)),
(s4, wait),
(s5, **move(l5,l4)**)}

Given that we start in s1,
also **two** different histories…
80% chance of history h1,
20% chance of history h2



$\gamma = 0.9$

Factors 1, 0.9, 0.81, 0.729, 0.6561…

$h_1 = \langle$s1, s2, s3, s4, s4, … $\rangle$

$V(h_1 \mid \pi_1) = .9^0(100) + .9^1(-1) + .9^2(-100) + .9^3\,100 + .9^4\,100 + \ldots = 547.9$

$h_2 = \langle$s1, s2, s5, s4, s4, … $\rangle$

$V(h_2 \mid \pi_1) = .9^0(-100) + .9^1(-1) + .9^2(-200) + .9^3 100 + \ldots = 466.9$

$E(\pi_2) = 0.8 * \boxed{547.9} + 0.2\,\boxed{(466.9)} = 531{,}7$

Expected reward 531,7 (π_1 gave 256.3)

Fully Observable Probabilistic Planning: Markov Decision Processes

Overview

- **Markov Decision Processes**
  - Underlying world model: **Stochastic system**
  - Plan representation: **Policy** – which action to perform in **any** state
  - Goal representation: **Utility function** defining "solution quality"
  - Planning problem: **Optimization**: Maximize **expected** utility

Why "Markov"?

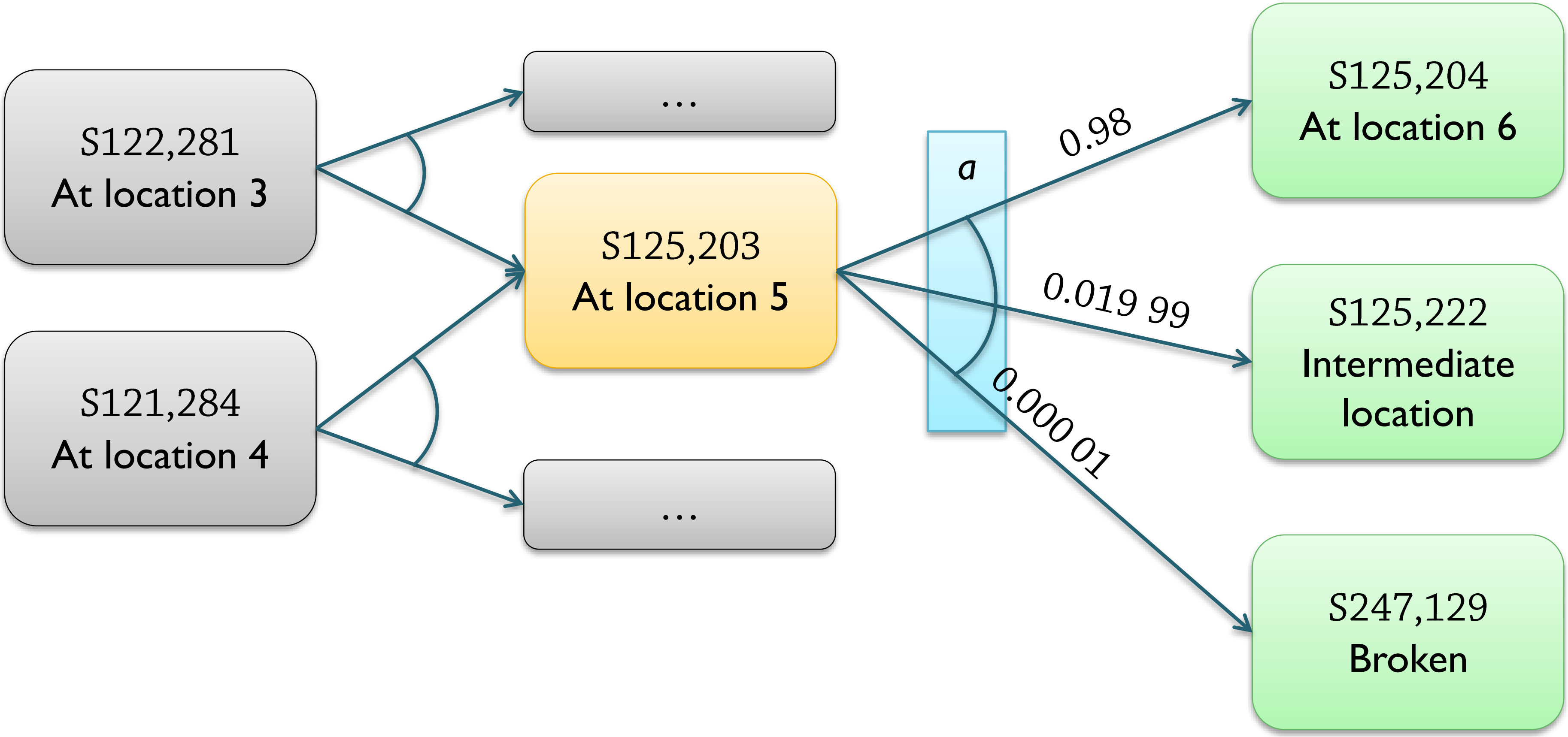
Markov Property (1)

- If a stochastic process has the **Markov Property**:
  - It is **memoryless**
  - The future of the process can be predicted equally well if we use only its *current state* or if we use *its entire history*
- This is part of the definition!
  - $P(s, a, s')$ is the **probability** of ending up in *s'* when we **are in *s*** and **execute *a***

Nothing else matters!

А. А. Марков (1886).

Markov Property (2)



We don't need to know the states we visited before…

Only the current state

and the action…

…To find out where we may end up, with which prob.

Remembering the Past

- Essential distinction:

Previous **states** in the **history sequence**:	What happened at **earlier timepoints**:
Cannot affect the transition function	Can partly be *encoded* into the *current state* **Can** affect the transition function

- Example:
  - If you **have visited the lectures**, you **are more likely to pass the exam**
    - Add a **visitedLectures** predicate / variable, representing *in this state* what you *did in the past*
  - This information is **encoded and stored** in the **current state**
    - State space doubles in size (and here we often treat *every state* separately!)
    - We only have a finite number of states ➔ can't encode an *unbounded* history

Policies and Expected Utilities: Expectations Revisited

Expected Utility

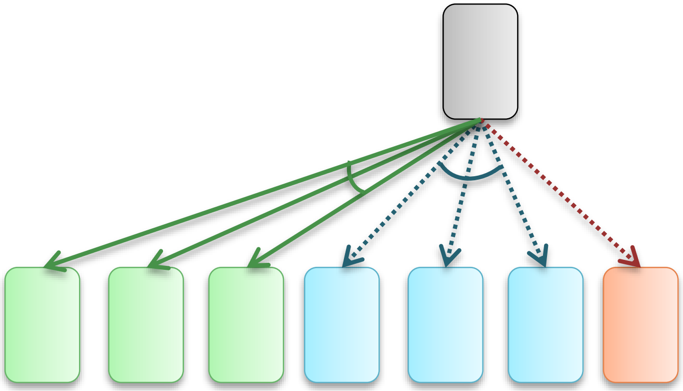
- Expected utility – similar to expected cost:

  - We know the utility of each **history**, of each **outcome**
    - But we can only *decide* a policy

  - Each outcome has a **probability**
    - So we can calculate an **expected** ("average") utility for the policy: $E(\pi)$

Expected Utility 2

- A **policy** selects actions; the **world** chooses the outcome



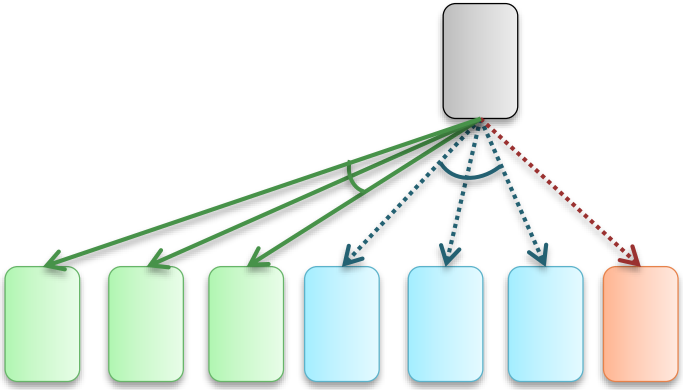
If the policy chooses the green action, the world selects one of these outcomes

Action blue ➔ world selects outcome

Action red ➔ one possible outcome

Expected Utility 3

- We must consider *all possible outcomes / histories* but not *all possible choices*



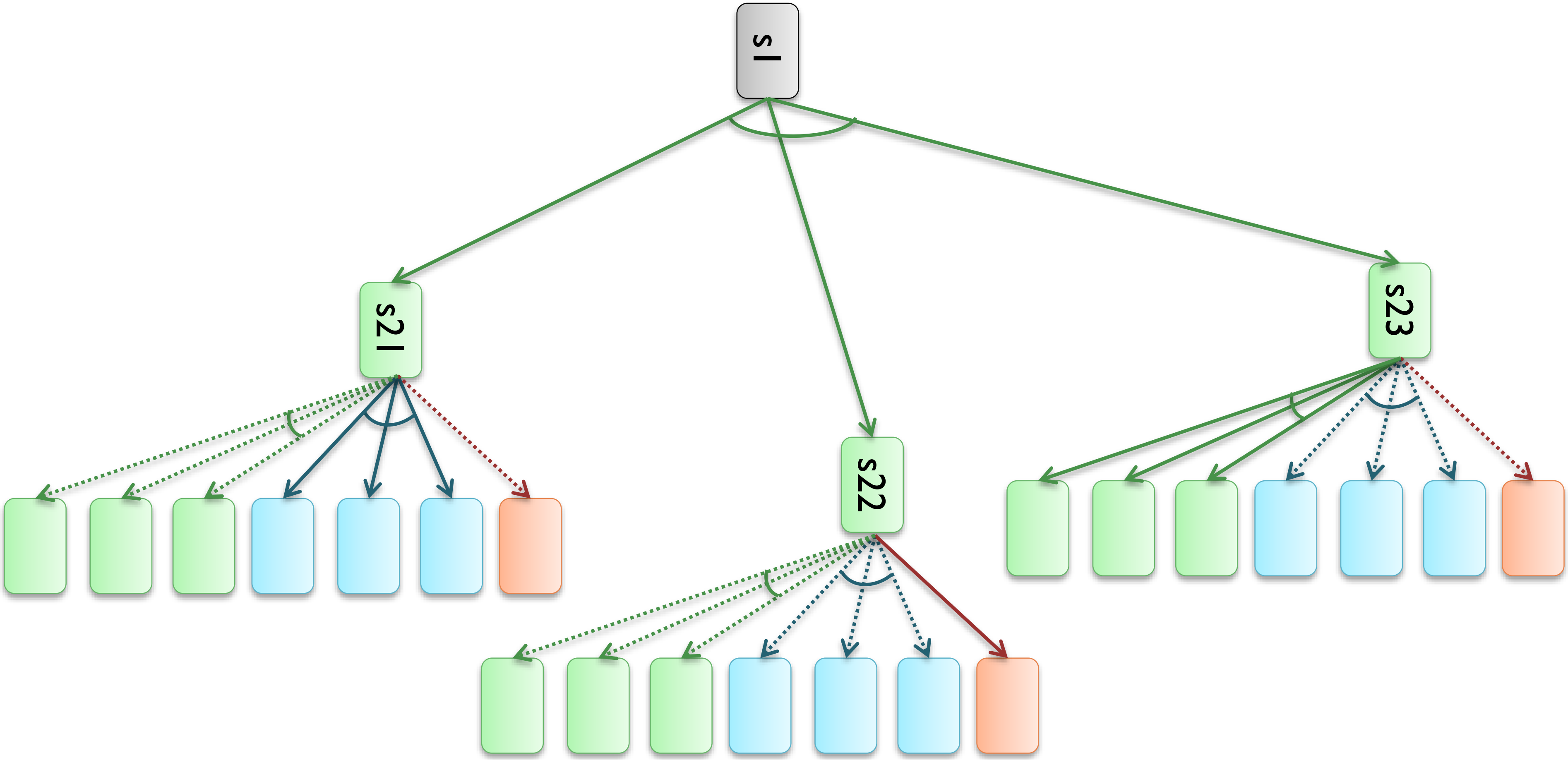
Suppose the policy chooses green action

These outcomes must be handled!

Irrelevant to us

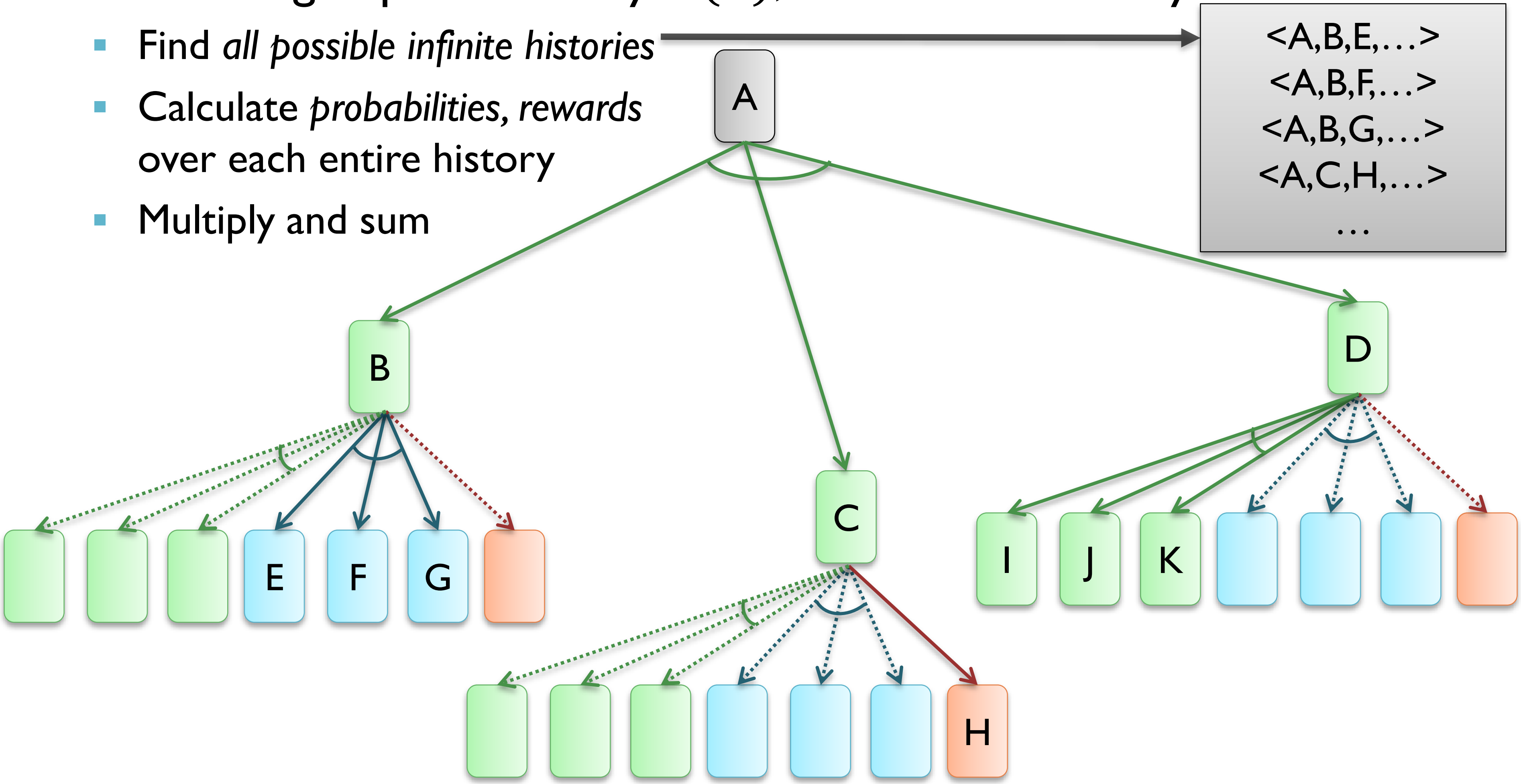
Expected Utility 4

- In the next step the policy again makes a choice
  - Use $\pi(s21), \pi(s22)$ or $\pi(s23)$ depending on where you are



Expected Utility 4

- Calculating expected utility $E(\pi)$, method 1: "History-based"
  - Find *all possible infinite histories*
  - Calculate *probabilities, rewards* over each entire history
  - Multiply and sum



<A,B,E,…>
<A,B,F,…>
<A,B,G,…>
<A,C,H,…>
…

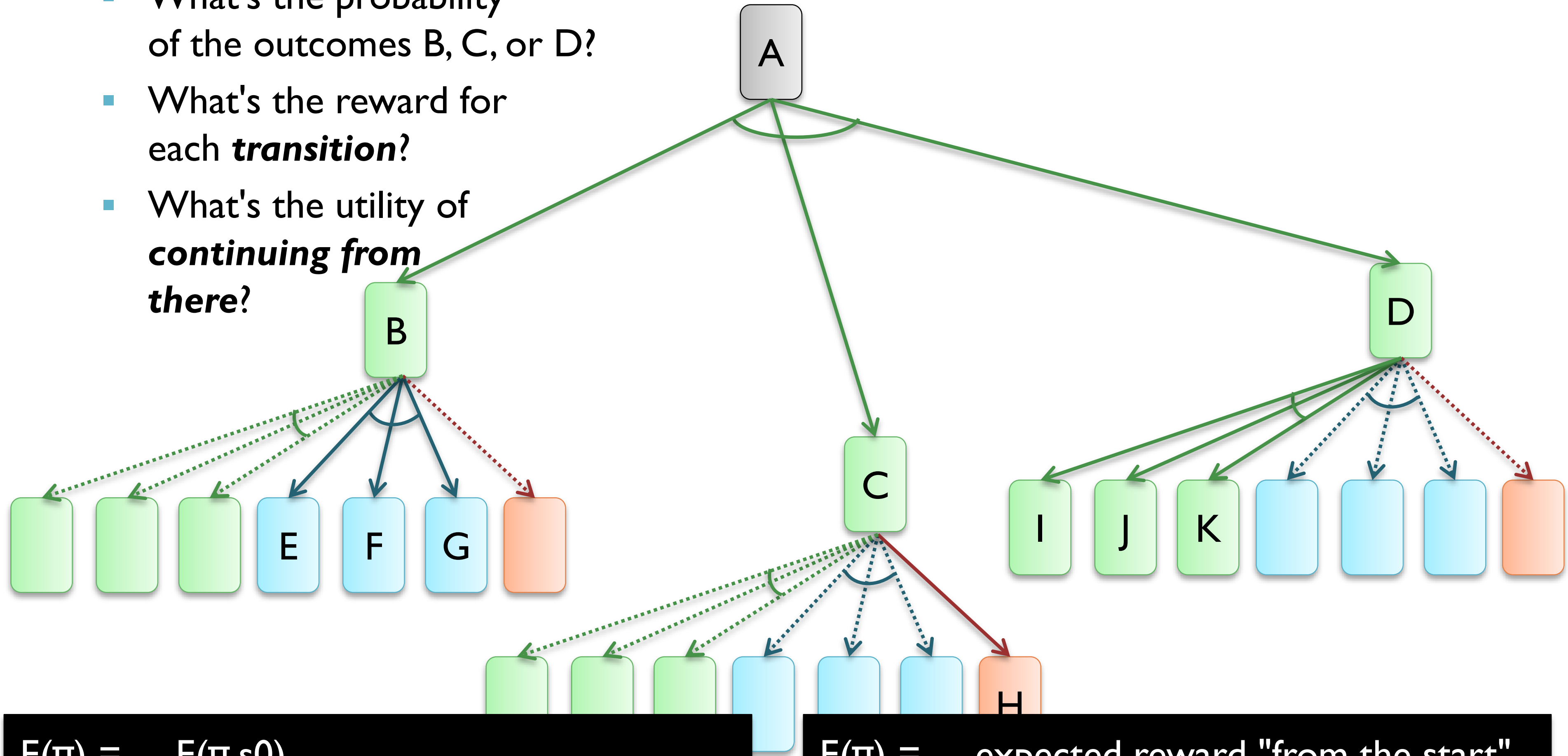
$E(\pi) = \sum_h P(h \mid \pi)\, V(h \mid \pi)$

where $V(h \mid \pi) = \sum_{i \geq 0} \gamma^i R(s_i, \pi(s_i), s_{i+1})$

Simple conceptually
Less useful for calculations

Expected Utility 5

- Calculating expected utility, method 2: **Recursive**
  - What's the probability of the outcomes B, C, or D?
  - What's the reward for each ***transition***?
  - What's the utility of ***continuing from there***?



$E(\pi) = E(\pi, s0)$

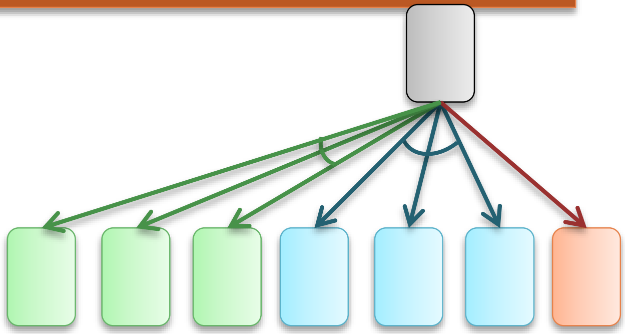
$E(\pi, s) = \sum_{s' \in S} P(s, \pi(s), s') * (R(s, \pi(s), s') + \gamma E(\pi, s'))$

$E(\pi) =$ expected reward "from the start"

$E(\pi, s) =$ "continuing after having reached s"

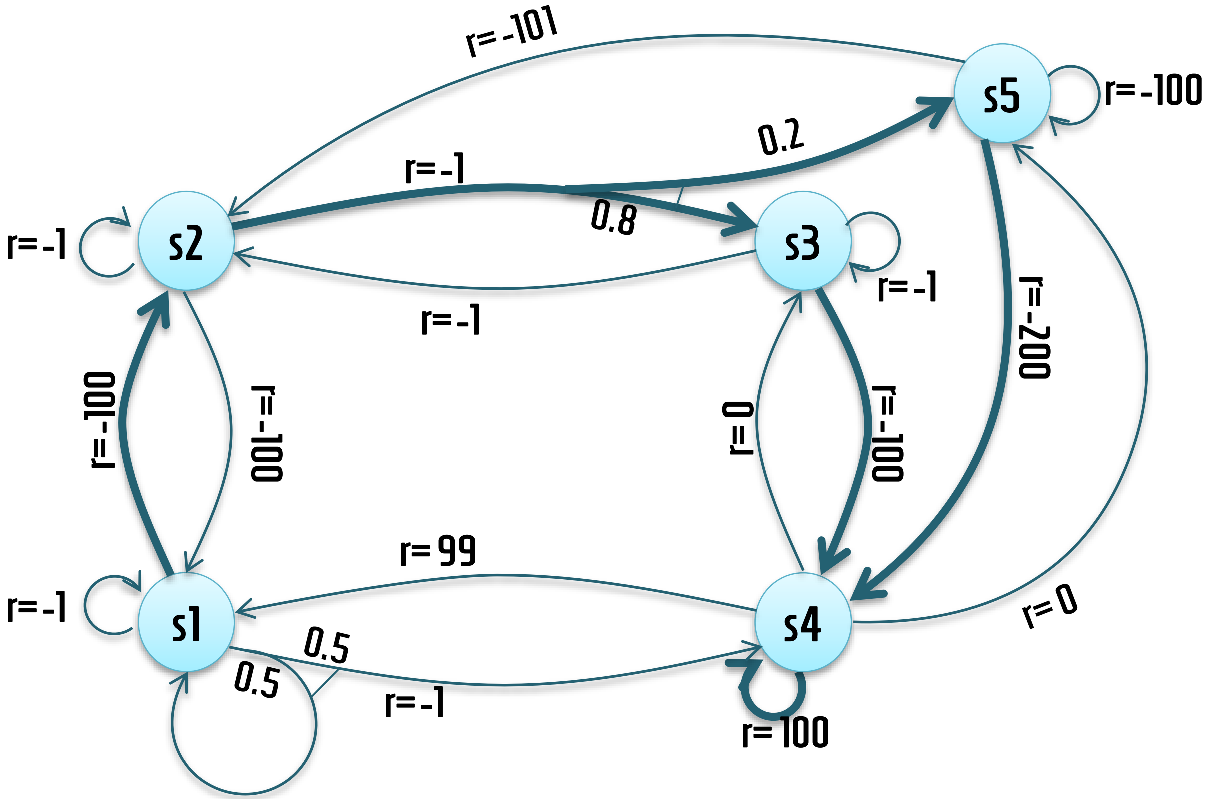
Expected Utility 6: "Step-Based"

- If π is a policy, then
  - E(π,s) = $\sum_{s' \in S}$ P(s, π(s), s') * (R(s, π(s), s') + γ E(π,s'))
  - The expected utility of continuing to execute π after having reached s
  - Is the sum, for all possible states $s' \in S$ that you might end up in,
    - of the probability $P(s, \pi(s), s')$ of actually ending up in that state given the action $\pi(s)$ chosen by the policy, times
    - the reward you get for this transition
    - plus the discount factor times the expected utility $E(\pi, s')$ of continuing π from the new state s'



Example 1

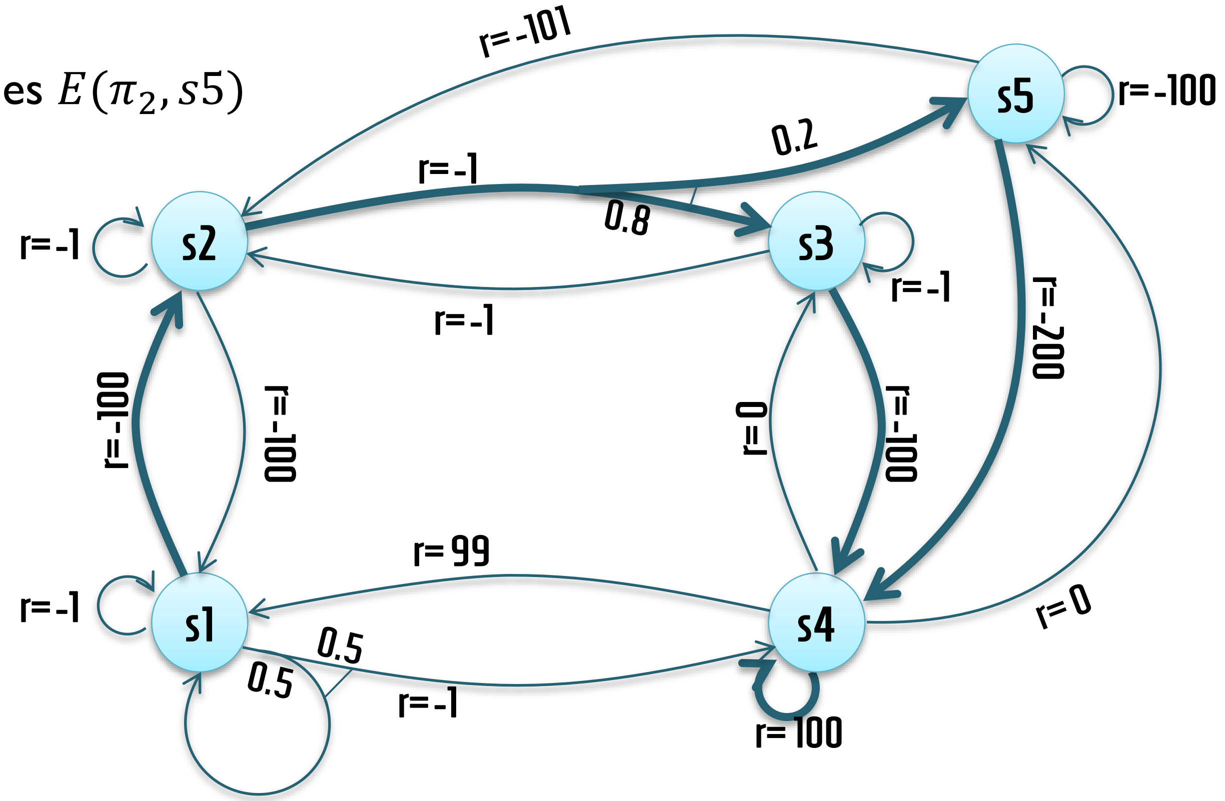
- $E(\pi_2, s1)$ = The expected reward of executing π_2 starting in **s1**:
  - Ending up in s2: 100% probability times
    - Reward -100
    - Discount factor γ times $E(\pi_2, s2)$



π_2 = {(s1, move(l1,l2)),
(s2, move(l2,l3)),
(s3, move(l3,l4)),
(s4, wait),
(s5, move(l5,l4)}

Example 2

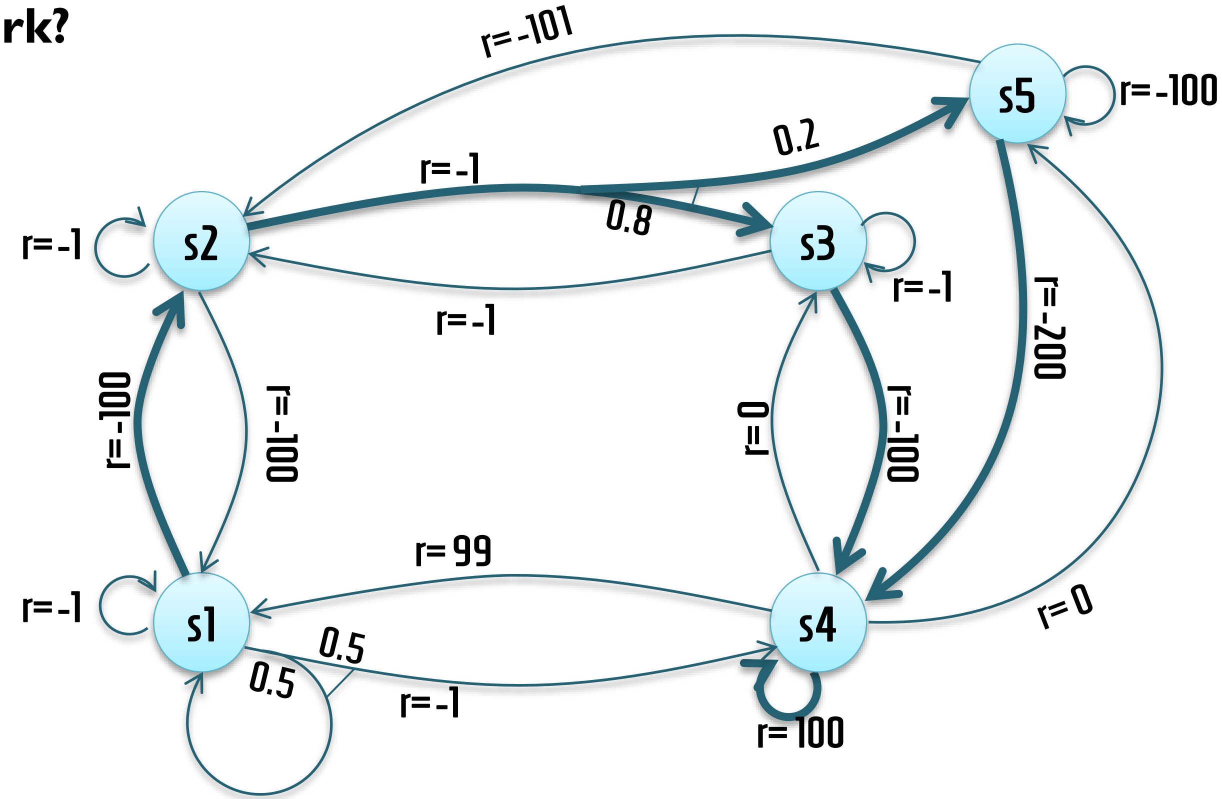
- $E(\pi_2, s2)$ = the expected utility of executing π_2 starting in **s2**:
  - Ending up in $s3$: 80% probability times
    - Reward -1
    - Discount factor γ times $E(\pi_2, s3)$
  - Ending up in $s5$: 20% probability times
    - Reward -1
    - Discount factor γ times $E(\pi_2, s5)$



π_2 = {(s1, move(l1,l2)),
(s2, move(l2,l3)),
(s3, move(l3,l4)),
(s4, wait),
(s5, move(l5,l4)}

Recursive?

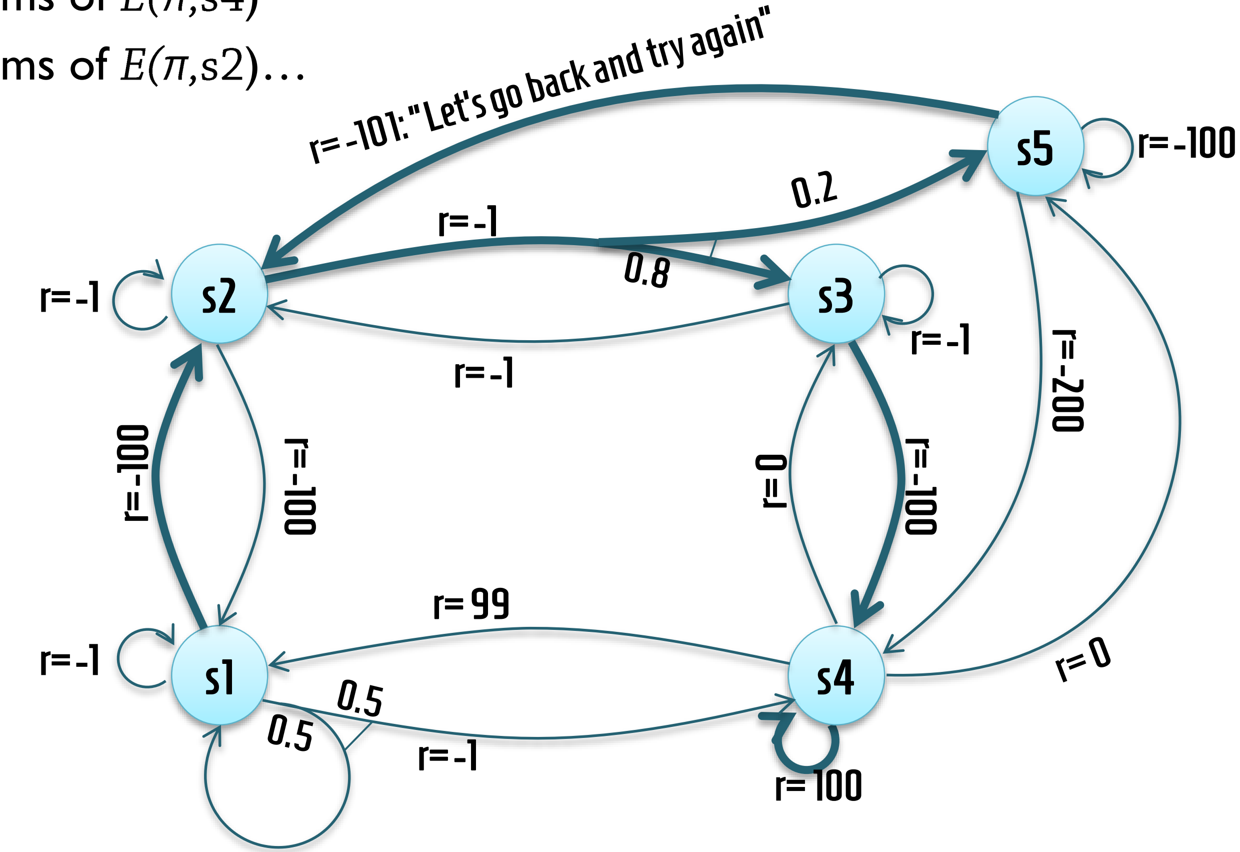
- Seems like we could easily calculate this **recursively**!
  - $E(\pi_2, s1)$
    - defined in terms of $E(\pi_2,s2)$
      - defined in terms of $E(\pi_2,s3)$ and $E(\pi_2,s5)$
        - …
  - Just continue until you reach the end!
  - **Why doesn't this work?**



π_2 = {(s1, move(l1,l2)),
(s2, move(l2,l3)),
(s3, move(l3,l4)),
(s4, wait),
(s5, move(l5,l4)}

Not Recursive!

- **There isn't always an "end"**!
  - Modified example below is a valid policy π (different action in s5)
    - $E(\pi, s1)$ defined in terms of $E(\pi, s2)$
    - $E(\pi, s2)$ defined in terms of $E(\pi, s3)$ and $E(\pi, s5)$
    - $E(\pi, s3)$ defined in terms of $E(\pi, s4)$
    - $E(\pi, s5)$ defined in terms of $E(\pi, s2)$…



Equation System

- If π is a policy, then
  - $E(\pi,s) = \sum_{s' \in S} P(s, \pi(s), s') * (R(s, \pi(s), s') + \gamma E(\pi,s'))$
  - The expected utility of continuing to execute π after having reached s
  - Is the sum, for all possible states s' ∈ S that you might end up in,
    - of the probability P(s, π(s), s') of actually ending up in that state
      given the action π(s) chosen by the policy, times
    - the reward you get for this transition
    - plus the discount factor
      times the expected utility E(π,s') of continuing π from the new state s'

This is an **<u>equation system</u>**: |S| equations, |S| variables!

Requires different solution methods…

MDPs part 2:
Finding Solutions

Optimality and
Bellman's Principle of Optimality

Repetition: Utility

- Let us first revisit the definition of **utility**
  - We can define the **actual utility** given an **outcome**, a history
    - Given any history $\langle s_0, s_1, \dots \rangle$:

$$V(\langle s_0, s_1, \dots \rangle | \pi) = \sum_{i \geq 0} \gamma^i \, R(s_i, \pi(s_i), s_{i+1})$$

Value of a history

Discounted rewards claimed

What happens in the future is *less* important ➔ expected **utility** != expected sum of **rewards**

  - We can define the **expected utility** using the given probability distribution:
    - Given that we start in state s:

$$E(\pi, s) = \sum_{\langle s_0, s_1, \dots \rangle} \left(P(\langle s_0, s_1, \dots \rangle \, | s_0 = s) \sum_{i \geq 0} \gamma^i \, R(s_i, \pi(s_i), s_{i+1}) \right)$$

All possible histories

P(that entire history, when starting in *s*)

Discounted reward for that entire history

    - As we saw, we can also **rewrite this recursively**!
      Given that we start in state s:

$$E(\pi, s) = \sum_{s' \in S} P(s, \pi(s), s') \cdot \left(R(s, \pi(s), s') + \gamma E(\pi, s') \right)$$

All possible next states s'

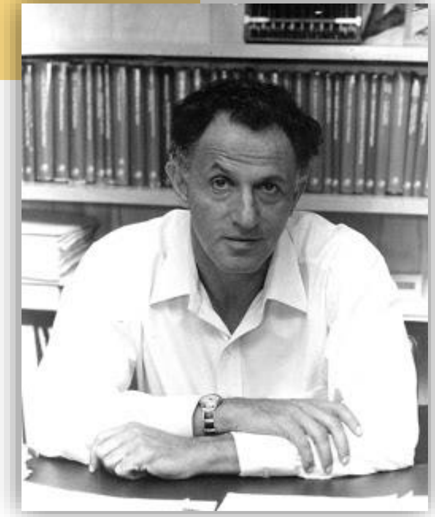
P(first step leads to s')

Immediate reward + discounted reward of continuing from s'

Maximizing Expected Utility

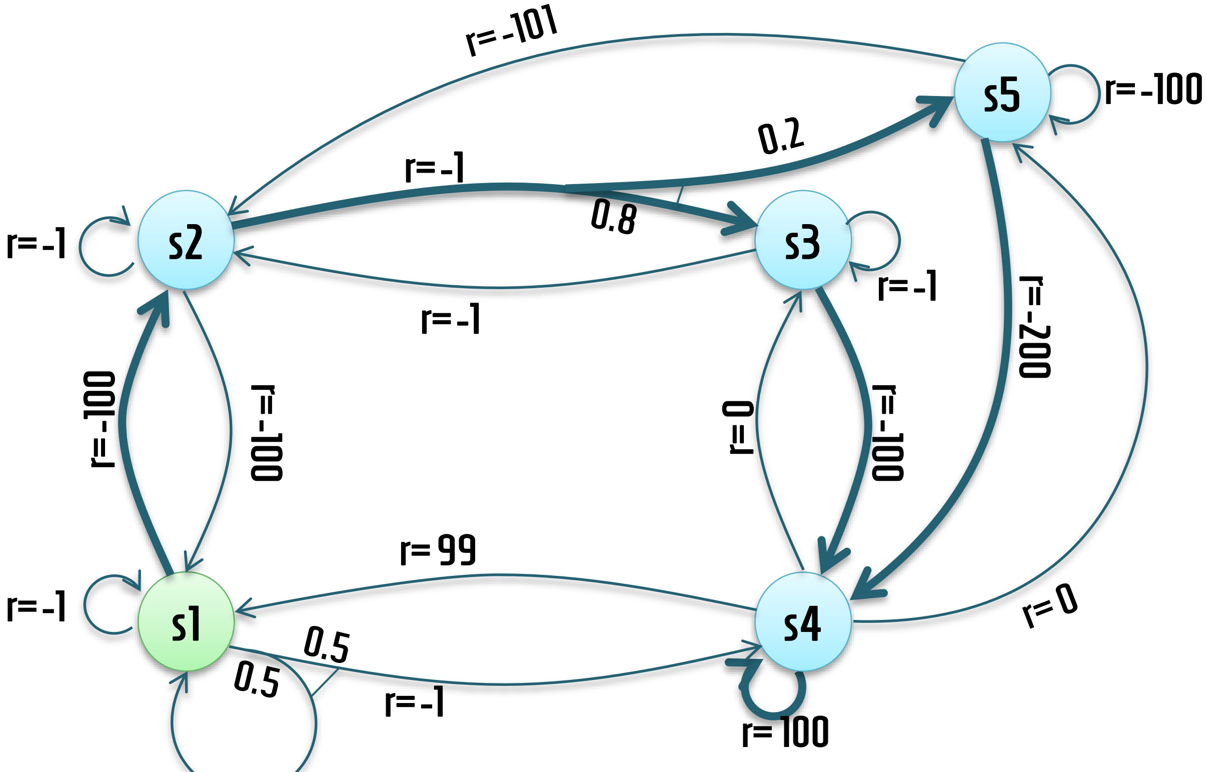
- Suppose that:
  - We know the **initial state** s_0
  - We want a **policy** π^* that **maximizes expected utility**: $E(\pi^*, s_0)$
  - How do we find one?

- Bellman's **Principle of Optimality**:
  - An **optimal policy** has the property that whatever the initial state and initial decision are, the **remaining decisions must constitute an optimal policy** with regard to the state resulting from the first decision!
  - Richard Ernest Bellman, 1920-1984



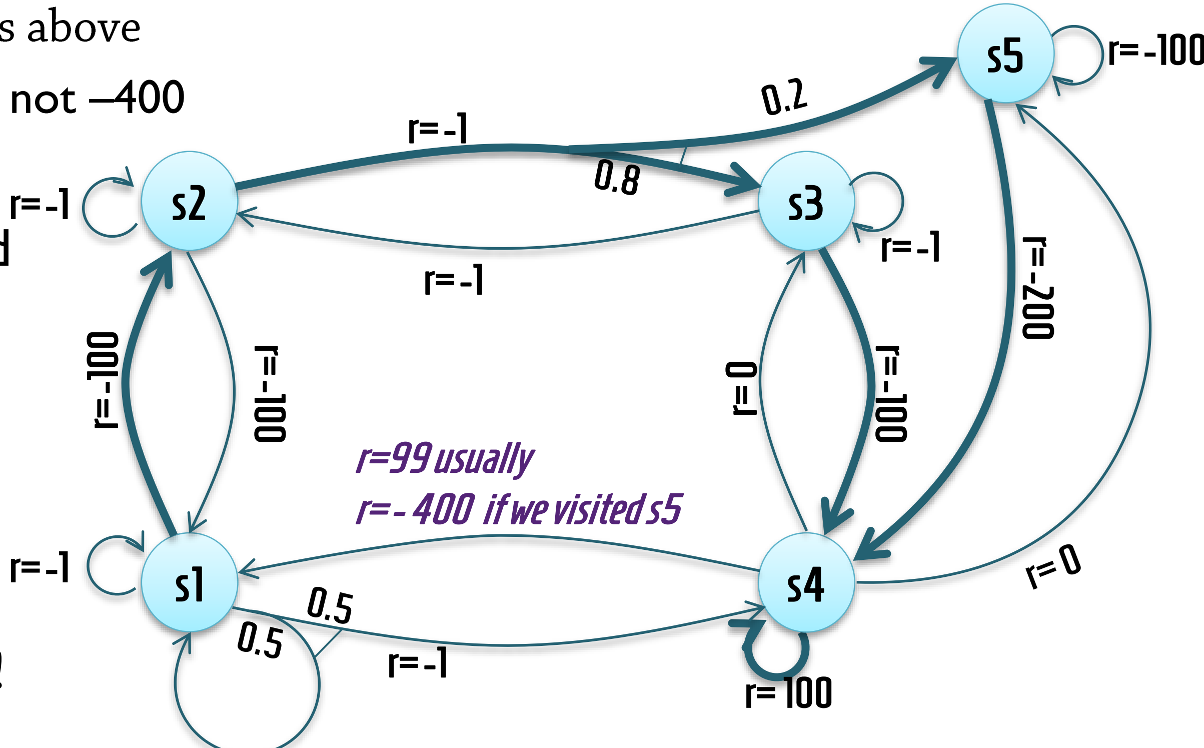
Principle of Optimality: Example

- Suppose we start in $s1$
- Suppose π^* is optimal **starting in $s1$**
  - It maximizes $E(\pi^*, s1)$: Expected utility starting in $s1$
- Suppose that $\pi^*(s1) = \text{move(l1,l2)}$, so that the next state must be $s2$
- Then π^* must also be optimal **starting in $s2$**!
  - Must maximize $E(\pi^*, s2)$: Expected utility starting in $s2$



Principle of Optimality (2)

- Sounds obvious? Depends on the **Markov Property**!
  - Suppose **rewards** depended on **which states you had visited before**
  - To go s5 → s4 → s1:
    - Use move(l5,l4) and move(l4,l1)
    - Reward –200 + –400 = –600
  - To go s4 → s1 *without* having visited s5:
    - Use move(l4,l1), same as above
    - Reward for this step: 99, not –400
  - ➔ Optimal action would have to take history into account
  - This can't happen in an MDP: **Markovian**!

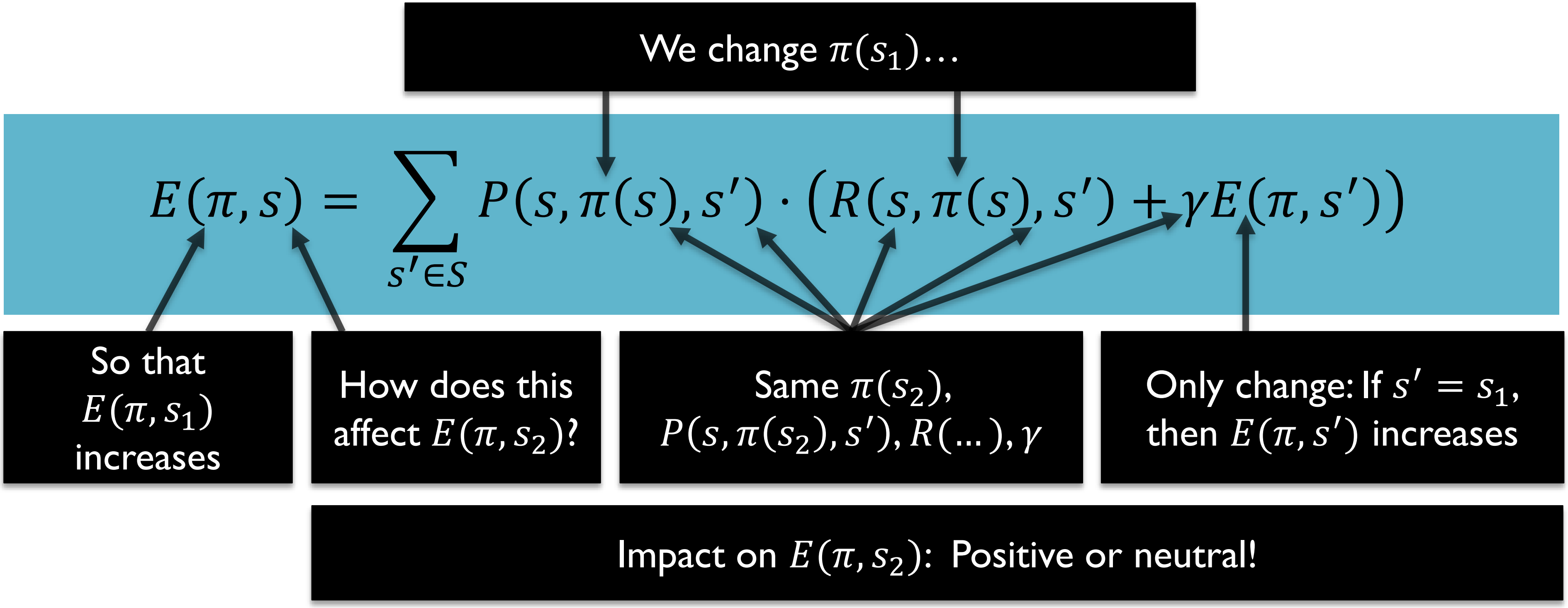


Consequences (1)

- To find an optimal policy π^*:
  - No need to know the initial state s_0 in advance:
    We can find a policy that is **optimal for all initial states**
  - **Definition:**
    An optimal policy π^* **maximizes expected utility for all states**:
    For all states s and alternative policies π,
    $$E(\pi^*, s) \geq E(\pi, s)$$
  - **Definition**:
    A **solution** to an MDP is an **optimal policy**!

Consequences (2)

- Suppose I have a **non-optimal** policy π
  - I select an arbitrary state s
  - I make a **local improvement**:
    Change $\pi(s)$, selecting another action that increases $\mathrm{E}(\pi, s)$
  - This cannot make anything worse:
    Cannot decrease $\mathrm{E}(\pi, s')$ for **any** s'!



We change $\pi(s_1)$…

$$E(\pi, s) = \sum_{s' \in S} P(s, \pi(s), s') \cdot \left(R(s, \pi(s), s') + \gamma E(\pi, s')\right)$$

So that $E(\pi, s_1)$ increases

How does this affect $E(\pi, s_2)$?

Same $\pi(s_2)$, $P(s, \pi(s_2), s')$, $R(\dots)$, γ

Only change: If $s' = s_1$, then $E(\pi, s')$ increases

Impact on $E(\pi, s_2)$: Positive or neutral!

Consequences (3)

- Also:
  - Every global improvement **can be reached** through such local improvements (no need to first make the policy worse, then better)
- → We can **find optimal solutions** through **local** improvements
  - No need to "think **globally**"

Finding a Solution (Optimal Policy): Algorithm 1, Policy Iteration

Simplification

- In many presentations (and our current example), rewards **do not depend on the outcome s'**!

$$E(\pi, s) = \sum_{s' \in S} P(s, \pi(s), s') \cdot \big(R(s, \pi(s), s') + \gamma E(\pi, s')\big)$$

➔

$$E(\pi, s) = R\big(s, \pi(s)\big) + \sum_{s' \in S} P(s, \pi(s), s') \cdot \gamma E(\pi, s')$$

Policy Iteration

- First algorithm: **Policy iteration**
  - General idea:
    - Start out with an **initial policy**, maybe randomly chosen
    - Calculate the **expected utility** of executing that policy from each state
    - **Update** the policy by making a **local** decision **for each state** : ”Which action should my **improved** policy choose in this state, given the expected utility of the **current** policy?”
    - Iterate until convergence (until the policy no longer changes)

Preliminaries 1: Single-step policy changes

- Preliminaries:
  - Suppose I have a policy π, with an expected utility:

$$E(\pi, s) = R(s, \pi(s)) + \sum_{s' \in S} P(s, \pi(s), s') \cdot \gamma E(\pi, s')$$

  - Suppose I change the decision in the **<u>first step</u>**, and keep the policy for everything else!
  - Expected utility of this procedure:

$$Q(\pi, s, a) \;=\; R(s, a) + \sum_{s' \in S} P(s, a, s') \cdot \gamma E(\pi, s')$$

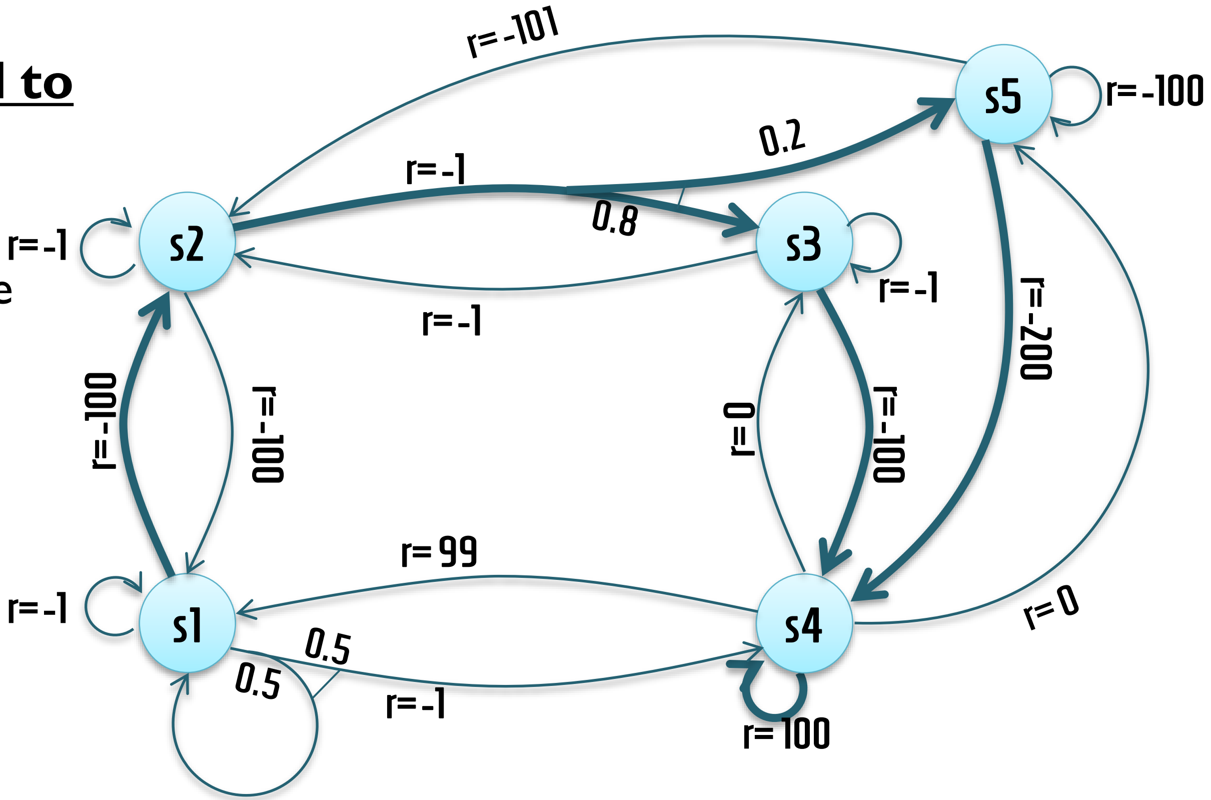
    - $Q(\pi, s, a)$ is the expected utility of π in a state s if we **<u>start</u>** by executing the given action a, but we use the **<u>policy</u>** π from then onward

<u>Note:</u> $E(\pi, s) = Q(\pi, s, \pi(s))$: *"What if we first did what the policy said, and then continued using the policy?"*

<u>Why</u>?
This tells us if we have a potential *improvement*, without solving a full equation system!

Preliminaries 2: Example

- Example: $E(\pi, s1)$
  - The expected utility of following π
  - Starting in s1, beginning with move(l1,l2)
- $Q(\pi, s1, \text{move}(l1, l4))$
  - The expected utility of first executing move(l1, l4) from s1, *then* following policy π
- **Does not correspond to any possible policy!**
  - If move(l1,l4) returns you to state s1, then the next action is move(s1,s2)!



Preliminaries 3

- Suppose you have an **optimal** policy π^*
  - Then, because of the principle of optimality:
    - In every state, the **local** choice made by the policy is **locally** optimal
    - For all states s,

$$E(\pi^*, s) = \max_{a \in A} Q(\pi^*, s, a)$$

- **Yields the modification step of policy iteration**!
  - We **have** a possibly non-optimal policy π,
    want to create an improved policy π'
  - For every state s, set

$$\pi'(s) := \arg\max_{a \in A} Q(\pi, s, a)$$

But what if there was an **even better** choice,
which we don't see now because of our single step modification (Q)?

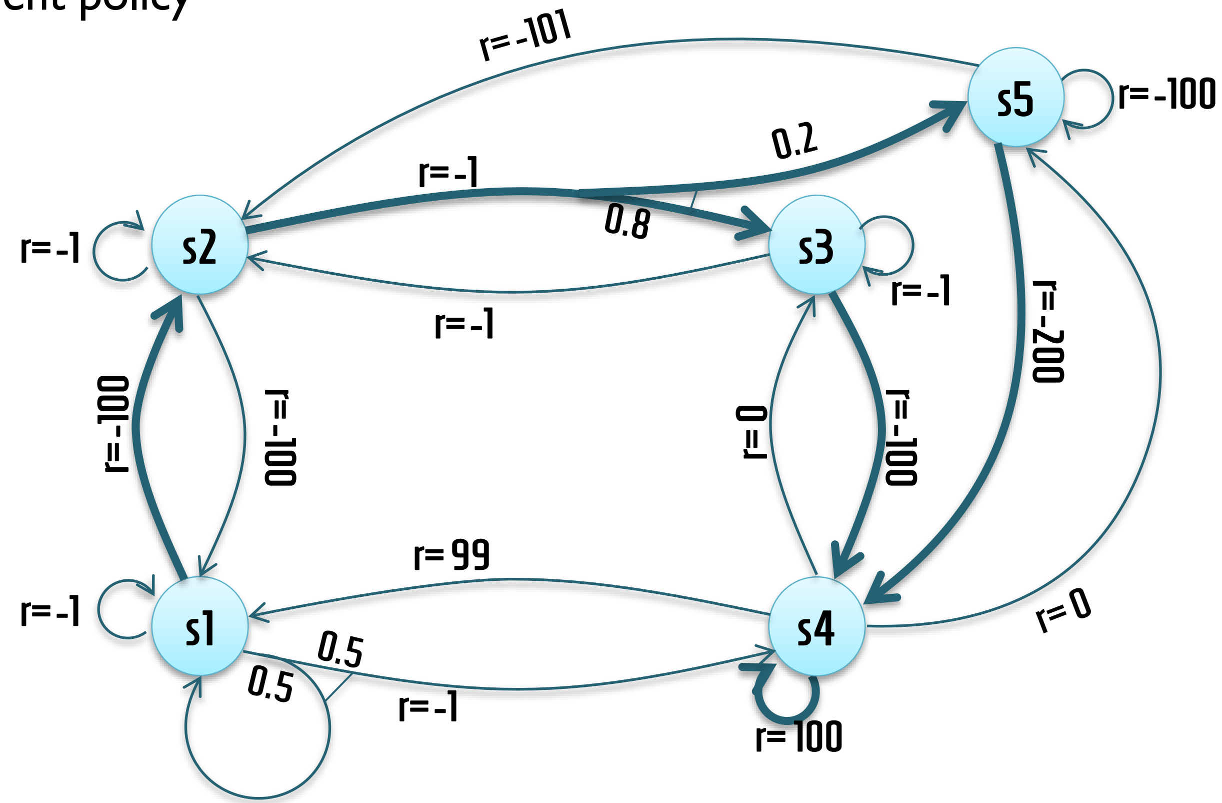
That's OK: We still have an *improvement*,
which cannot prevent *future* improvements

Preliminaries 4

- Example: $E(\pi, s1)$
  - The expected utility of following the current policy
  - Starting in s1, beginning with move(l1,l2)
- $Q(\pi, s1, move(l1, l4))$
  - The expected utility of first trying to move from l1 to l4, then following the current policy

If doing move(l1,l4) first
has a greater expected utility,
we should **modify**
the current policy:

$\pi'(s1) :=$ move(l1,l4)



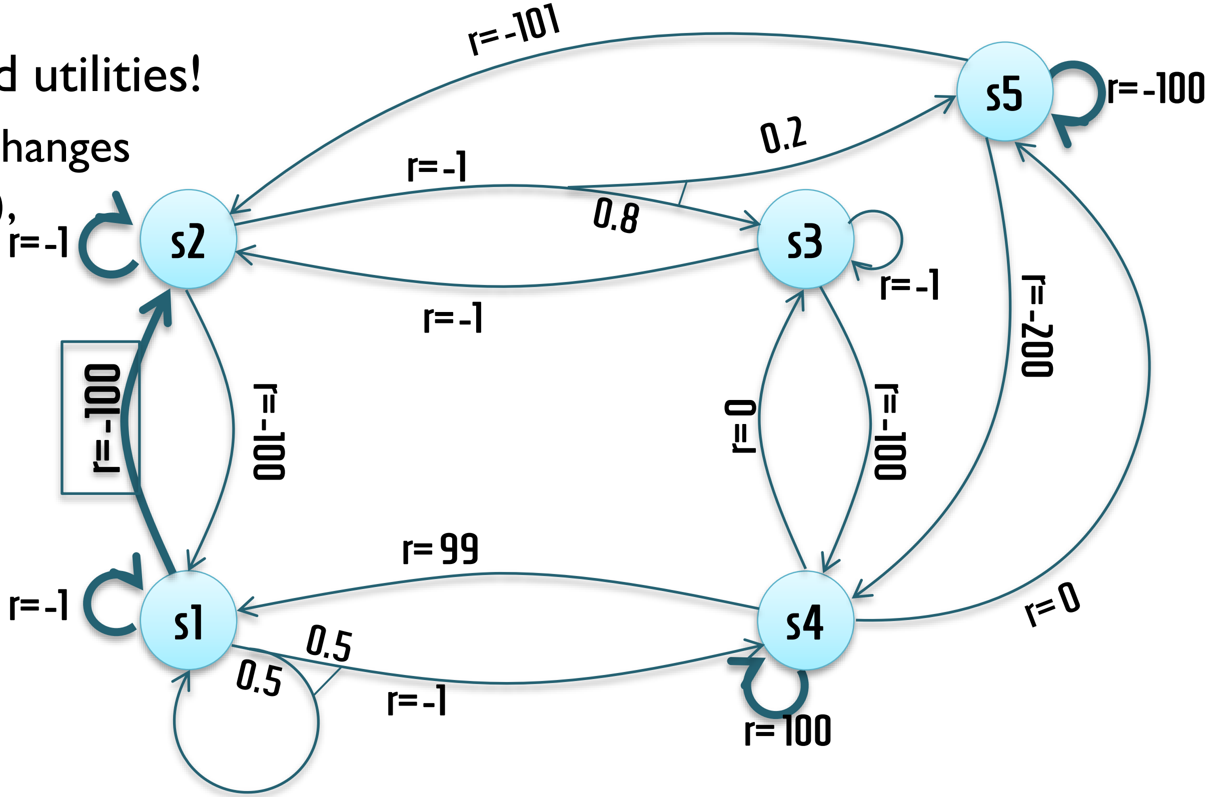
First Iteration

Policy Iteration 1: Initial Policy π_1

- Policy iteration requires an **initial policy**
  - Let's start by choosing "wait" in every state
  - Let's set a discount factor: $\gamma = 0.9$
    - Easy to use in calculations on these slides, but in reality we might use a larger factor (we're not **that** short-sighted!)

$$\pi_1 = \{(s1, wait), (s2, wait), (s3, wait), (s4, wait), (s5, wait)\}$$

- Need to know expected utilities!
  - Because we will make changes according to $Q(\pi_1, s, a)$, which depends on $\sum_{s' \in S} P(s, a, s')\, E(\pi_1, s')$



Policy Iteration 2: Expected Utility for π_1

- Calculate expected utilities for the **current** policy π_1
  - Simple: Chosen transitions are deterministic ***and*** return to the same state!
    - $E(\pi,s) = R(s, \pi(s)) + \gamma \sum_{s' \in S} P(s, \pi(s), s')\, E(\pi,s')$
    - $E(\pi 1,s1) = R(s1, \text{wait}) + \gamma\, E(\pi 1,s1) = -1 + 0.9\, E(\pi 1,s1)$
    - $E(\pi 1,s2) = R(s2, \text{wait}) + \gamma\, E(\pi 1,s2) = -1 + 0.9\, E(\pi 1,s2)$
    - $E(\pi 1,s3) = R(s3, \text{wait}) + \gamma\, E(\pi 1,s3) = -1 + 0.9\, E(\pi 1,s3)$
    - $E(\pi 1,s4) = R(s4, \text{wait}) + \gamma\, E(\pi 1,s4) = +100 + 0.9\, E(\pi 1,s4)$
    - $E(\pi 1,s5) = R(s5, \text{wait}) + \gamma\, E(\pi 1,s5) = -100 + 0.9\, E(\pi 1,s5)$
  - Simple equations to solve:
    - $0.1E(\pi 1,s1) = -1$ ➔ $E(\pi 1,s1) = -10$
    - $0.1E(\pi 1,s2) = -1$ ➔ $E(\pi 1,s2) = -10$
    - $0.1E(\pi 1,s3) = -1$ ➔ $E(\pi 1,s3) = -10$
    - $0.1E(\pi 1,s4) = +100$ ➔ $E(\pi 1,s4) = +1000$
    - $0.1E(\pi 1,s5) = -100$ ➔ $E(\pi 1,s5) = -1000$

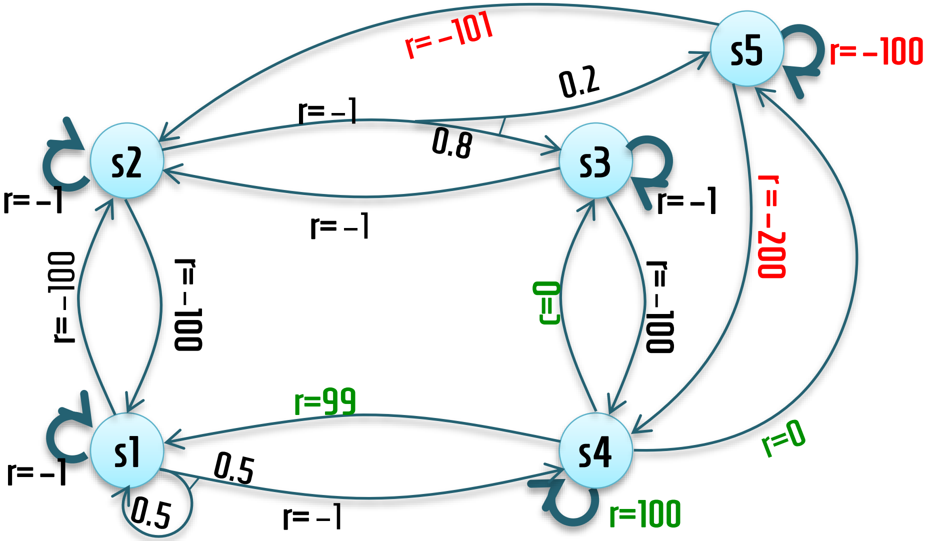
Given this policy π_1:

High rewards if we start in s4,
high costs if we start in s5

Policy Iteration 3: Update 1a

What is the best **local** modification according to the **expected utilities** of the **current** policy?

$E(\pi_1, s1) = -10$
$E(\pi_1, s2) = -10$
$E(\pi_1, s3) = -10$
$E(\pi_1, s4) = +1000$
$E(\pi_1, s5) = -1000$



- For every state s:
  - Let $\pi_2(s) = \text{argmax}_{a \in A}\, Q(\pi_1, s, a)$
  - That is, find the action *a* that maximizes $R(s, a) + \gamma \sum_{s' \in S} P(s, a, s')\ E(\pi 1, s')$
    - s1: wait / move(l1,l2) / move(l1,l4)

s1: wait	-1	$+ 0.9 *$	-10	$= -10$
move(l1,l2)	-100	$+ 0.9 *$	-10	$= -109$
move(l1,l4)	-1	$+ 0.9 *$	$(0.5 * -10 + 0.5 * 1000)$	$= +444{,}5$

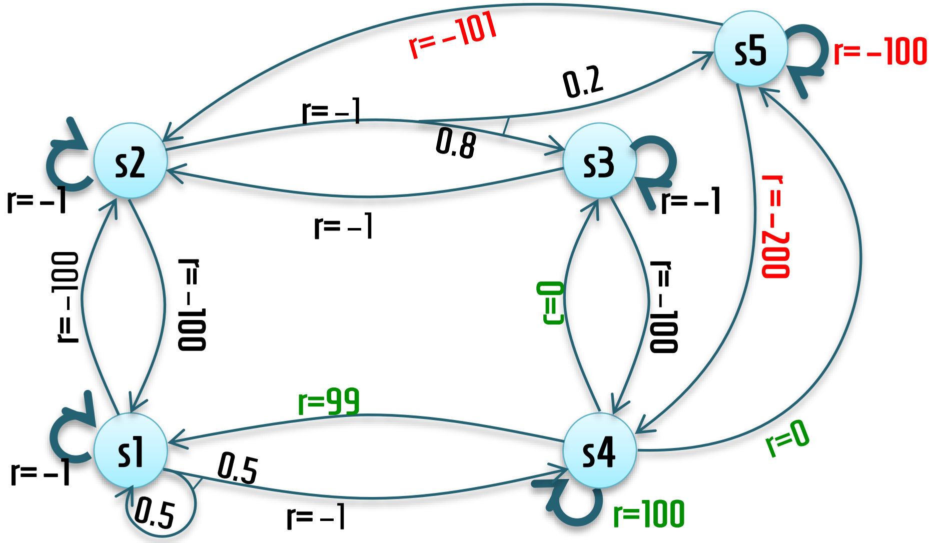
Best improvement

  - These are not the **true** expected utilities for starting in state $s1$!
    - Only correct if we locally change the **first** action to execute then *go on to use the previous policy* (in this case, always waiting)!
    - But they can be proven to yield good guidance, as long as you apply the improvements repeatedly (as policy iteration does)

Policy Iteration 4: Update 1b

What is the best **local** modification according to the **expected utilities** of the **current** policy?

$E(\pi_1, s1) = -10$
$E(\pi_1, s2) = -10$
$E(\pi_1, s3) = -10$
$E(\pi_1, s4) = +1000$
$E(\pi_1, s5) = -1000$

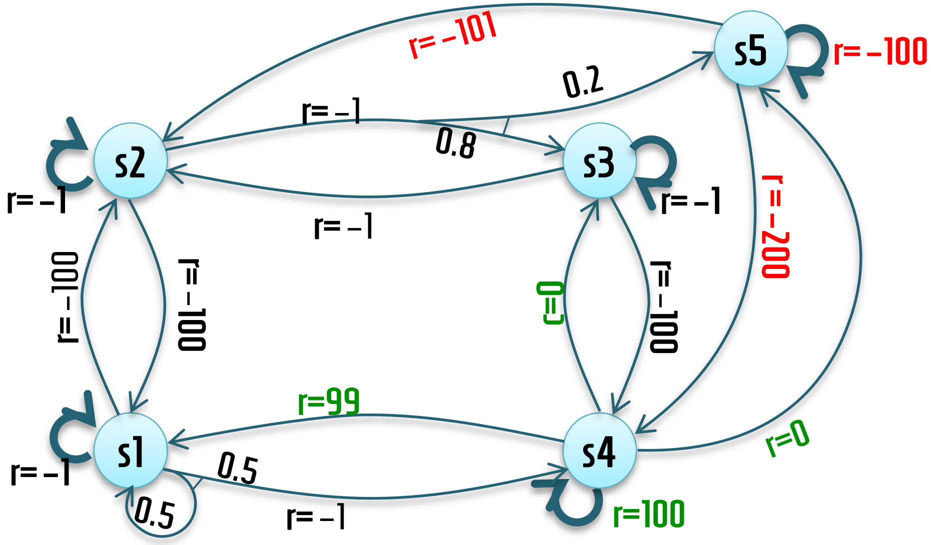


- For every state s:
  - Let $\pi_2(s) = \text{argmax}_{a \in A} Q(\pi 1, s, a)$
  - That is, find the action *a* that maximizes $R(s, a) + \gamma \sum_{s' \in S} P(s, a, s')\ E(\pi 1, s')$
    - s2: wait $-1 + 0.9 * -10 = -10$
      move(l2,l1) $-100 + 0.9 * -10 = -109$
      move(l2,l3) $-1 + 0.9 * (0.8*-10 + 0.2*-1000) = -188{,}2$

Policy Iteration 5: Update 1c

What is the best **local** modification according to the **expected utilities** of the **current** policy?

$E(\pi_1, s1)$	$= -10$
$E(\pi_1, s2)$	$= -10$
$E(\pi_1, s3)$	$= -10$
$E(\pi_1, s4)$	$= +1000$
$E(\pi_1, s5)$	$= -1000$



- For every state s:
  - Let $\pi_2(s) = \mathrm{argmax}_{a \in A}\, Q(\pi 1, s, a)$
  - That is, find the action *a* that maximizes $R(s, a) + \gamma \sum_{s' \in S} P(s, a, s')\ \ E(\pi 1, s')$
    - s3: wait
    - s4: wait
    - s5: wait

s3: wait	−1	+ 0.9 *	−10	= −10
move(l3,l2)	−1	+ 0.9 *	−10	= −10
move(l3,l4)	−100	+ 0.9 *	+1000	= +800
s4: **wait**	+100	+ 0.9 *	+1000	= +1000
move(l4,l1)	+99	+ 0.9 *	−10	= +90
…				
s5: wait	−100	+ 0.9 *	−1000	= −1000
move(l5,l2)	−101	+ 0.9 *	−10	= −110
move(l5,l4)	−200	+ 0.9 *	+1000	= +700

Second Iteration

Policy Iteration 6: Second Policy

- This results in a **new policy**

π_1 = {(s1, wait),	E(π1,s1) =–10
(s2, wait),	E(π1,s2) = –10
(s3, wait),	E(π1,s3) = –10
(s4, wait),	E(π1,s4) =+ 1000
(s5, wait)}	E(π1,s5) = –1000

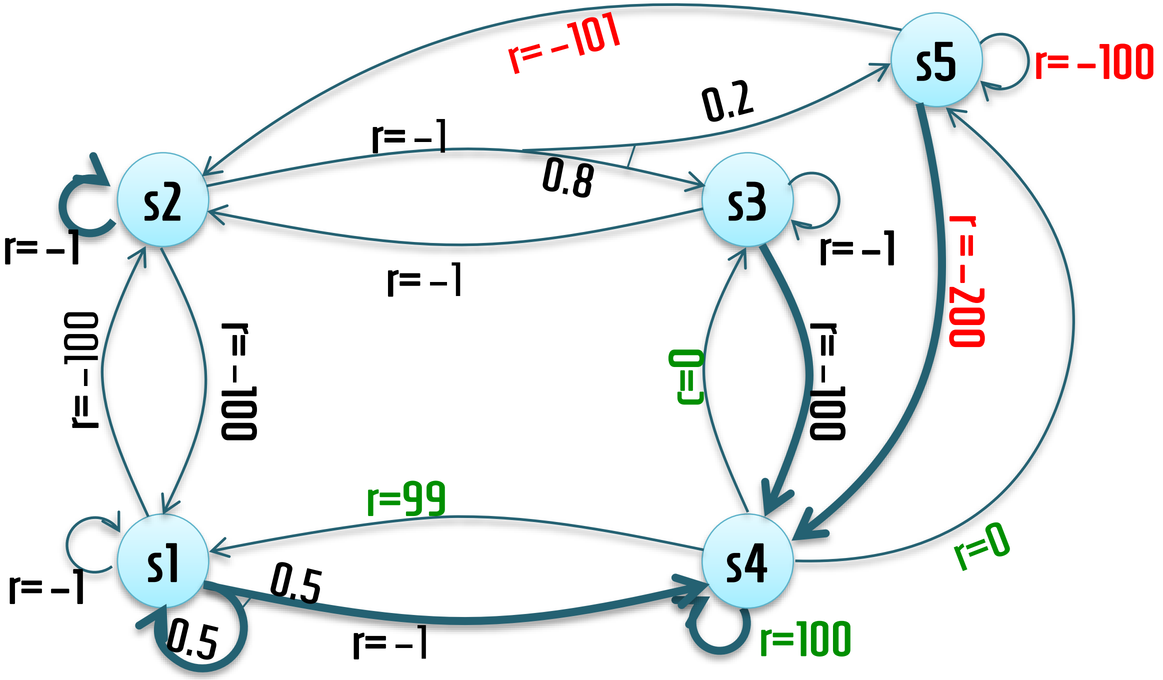
π_2 = { (s1, move(l1,l4),	>= + **444,5**
(s2, wait),	>= –10
(s3, move(l3,l4)),	>= + 800
(s4, wait),	>= + 1000
(s5, move(l5,l4))}	>= + 700

Utilities based on *one* modified action, then following π_1 (can't decrease!)

Now we have made use of earlier indications that s4 seems to be a good state

→ Try to go there from s1 / s3 / s5!

No change in s2 yet…



Policy Iteration 7: Expected Utilities for π_2

- Calculate **true** expected utilities for the **new** policy π_2
  - $E(\pi2,s1) = R(s1, move(l1,l4)) + \gamma \ldots = -1 + 0.9\,(0.5E(\pi2,s1) + 0.5E(\pi2,s4))$
  - $E(\pi2,s2) = R(s2, wait) + \gamma E(\pi2,s2) = -1 + 0.9\,E(\pi2,s2)$
  - $E(\pi2,s3) = R(s3, move(l3,l4)) + \gamma E(\pi2,s4) = -100 + 0.9\,E(\pi2,s4)$
  - $E(\pi2,s4) = R(s4, wait) + \gamma E(\pi2,s4) = +100 + 0.9\,E(\pi2,s4)$
  - $E(\pi2,s5) = R(s5, move(l5,l4)) + \gamma E(\pi2,s4) = -200 + 0.9\,E(\pi2,s4)$
- Equations to solve:
  - $0.1E(\pi2,s2) = -1$ ➔ $E(\pi2,s2) = -10$
  - $0.1E(\pi2,s4) = +100$ ➔ $E(\pi2,s4) = +1000$
  - $E(\pi2,s3) = -100 + 0.9E(\pi2,s4) = -100 + 0.9^*1000 = +800$ ➔ $E(\pi2,s3) = +800$
  - $E(\pi2,s5) = -200 + 0.9E(\pi2,s4) = -200 + 0.9^*1000 = +700$ ➔ $E(\pi2,s5) = +700$
  - $E(\pi2,s1) = -1 + 0.45 * E(\pi2,s1) + 0.45 * E(\pi2,s4)$ ➔ ➔ $E(\pi2,s1) = +816{,}36$
    $0.55\,E(\pi2,s1) = -1 + 0.45 * E(\pi2,s4)$ ➔
    $0.55\,E(\pi2,s1) = -1 + 450$ ➔
    $0.55\,E(\pi2,s1) = +449$ ➔
    $E(\pi2,s1) = +816{,}3636\ldots$

π_2 = {(s1, move(l1,l4)),
(s2, wait),
(s3, move(l3,l4)),
(s4, wait),
(s5, move(l5,l4))}

Policy Iteration 8: Second Policy

- Now we have the **true** expected utilities of the second policy…

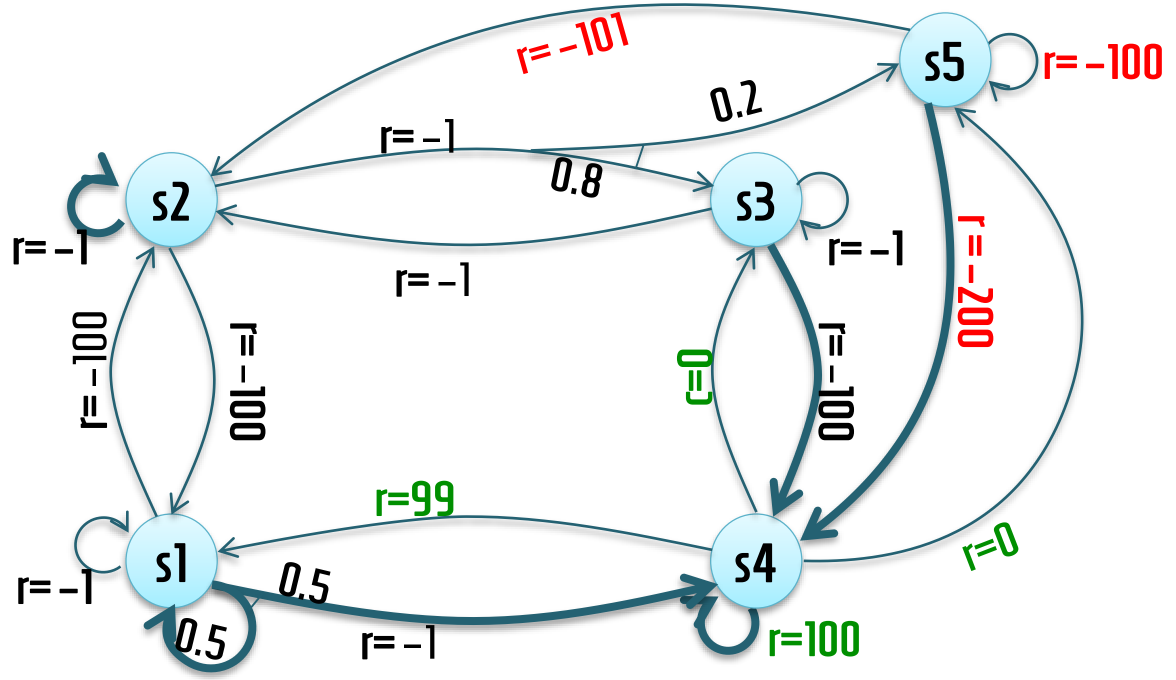
π_1 = {(s1, wait),	E(π1,s1)=–10
(s2, wait),	E(π1,s2)=–10
(s3, wait),	E(π1,s3)=–10
(s4, wait),	E(π1,s4)=+1000
(s5, wait)}	E(π1,s5)=–1000

π_2 = { (s1, move(l1,l4)),	>= +**444,5**	E(π2,s1) = +816,36
(s2, wait),	>= –10	E(π2,s2) = – 10
(s3, move(l3,l4)),	>= +800	E(π2,s3) = +800
(s4, wait),	>= +1000	E(π2,s4) = +1000
(s5, move(l5,l4))}	>= +700	E(π2,s5) = +700

S5 wasn't so bad after all,
since you can reach s4
in a single step!

S1 / s3 are even better.

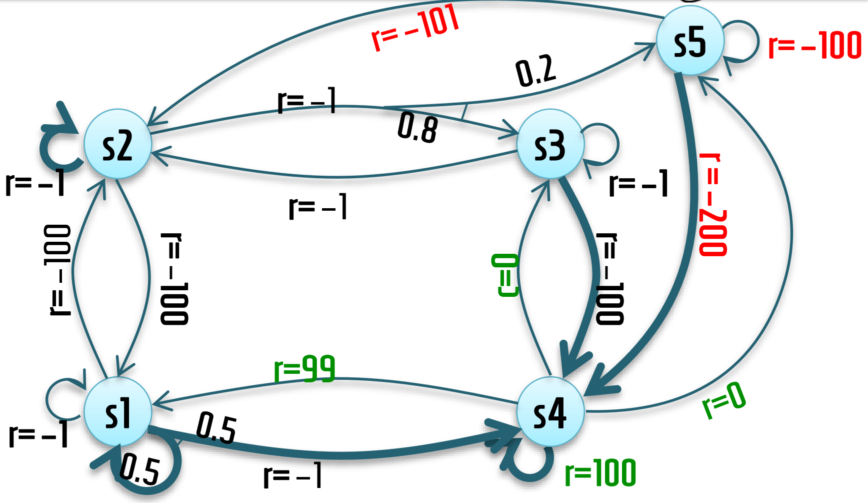
S2 *seems* much worse
in comparison,
since the benefits of s4
haven't "propagated" that far.



Policy Iteration 9: Update 2a

What is the best **local** modification according to the **expected utilities** of the **current** policy?

$E(\pi2,s1) = +816,36$
$E(\pi2,s2) = -10$
$E(\pi2,s3) = +800$
$E(\pi2,s4) = +1000$
$E(\pi2,s5) = +700$



- For every state s:
  - Let $\pi_3(s) = \operatorname{argmax}_{a \in A} Q(\pi_2,s,a)$
  - That is, find the action *a* that maximizes $R(s, a) + \gamma \sum_{s' \in S} P(s, a, s') \; E(\pi_2,s')$

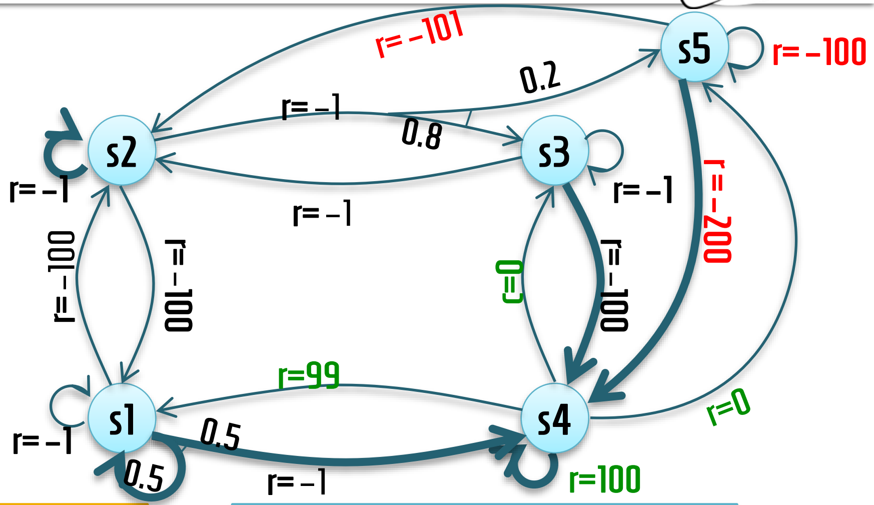
s1: wait	–1	+ 0.9 * 816,36	= +733,72
move(l1,l2)	–100	+ 0.9 * –10	= –109
move(l1,l4) **Seems best – chosen!**	–1	+ 0.9 * (.5*1000+.5*816.36)	= +816,36
s2: wait	–1	+ 0.9 * –10	= –10
move(l2,l1)	–100	+ 0.9 * 816,36	= +634,72
move(l2,l3)	–1	+ 0.9 * (0.8*800 + 0.2*700)	= +701

Now we will change the action taken at s2, since we have the expected utilities for reachable states s1, s3, s5… have increased

Policy Iteration 10: Update 2b

What is the best **local** modification according to the **expected utilities** of the **current** policy?

E(π2,s1) = +816,36
E(π2,s2) = –10
E(π2,s3) = +800
E(π2,s4) = +1000
E(π2,s5) = +700

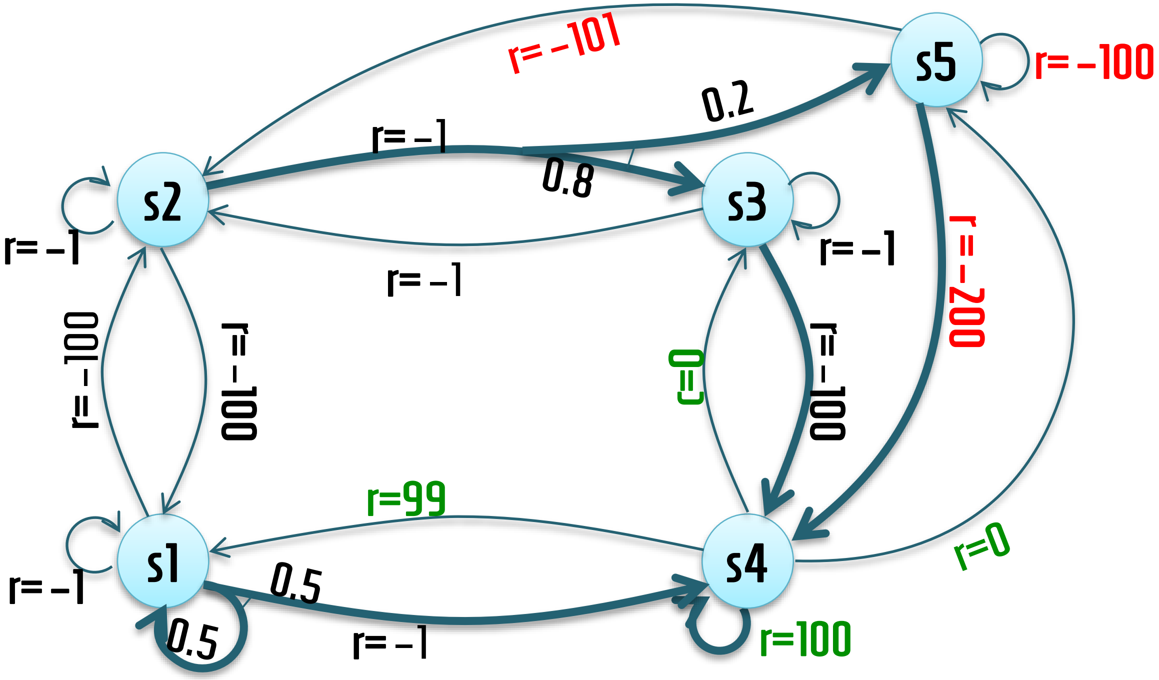


- For every state s:
  - Let $\pi_3(s) = \text{argmax}_{a \in A}\, Q(\pi_2,s,a)$
  - That is, find the action *a* that maximizes $R(s, a) + \gamma \sum_{s' \in S} P(s, a, s')\ \ E(\pi_2,s')$
    - s3: wait –1 + 0.9 * 800 = +719
      move(l3,l2) –1 + 0.9 * –10 = –10
      move(l3,l4) –100 + 0.9 * 1000 = +800
    - s4: wait +100+ 0.9 * 1000 = +1000
      move(l4,l1) +99 + 0.9 * 816,36 = +833,72
      …
    - s5: wait –100 + 0.9 * 700 = +530
      move(l5,l2) –101 + 0.9 * –10 = –110
      move(l5,l4) –200 + 0.9 * –1000 = +700

Policy Iteration 11: Third Policy

- This results in a **new policy** π$_3$
  - **True expected utilities** are updated by solving an equation system
  - The algorithm will iterate once more
  - No changes will be made to the policy
  - ➔ Termination with optimal policy!

π_3 = {(s1, move(l1,l4),
(s2, move(l2,l3)),
(s3, move(l3,l4)),
(s4, wait),
(s5, move(l5,l4))}



Policy Iteration Algorithm

Policy Iteration 12: Algorithm

- **Policy iteration** is a way to find an optimal policy π^*
  - Start with an **arbitrary** initial policy π_1. Then, for $i = 1, 2, \ldots$
    - Compute expected utilities $E(\pi_i, s)$ for every s by **solving a system of equations**

Find utilities according to current policy

- System: For all s, $E(\pi_i, s) = Q(\pi_i, s, \pi_i(s))$
  $= R(s, \pi_i(s)) + \gamma \sum_{s' \in S} P(s, \pi_i(s), s') E(\pi_i, s')$
- Result: The **expected utilities** of the "current" policy in **every** state s

- Not a simple recursive calculation – the state graph is generally cyclic!

- Compute an improved policy π_{i+1} "locally" for every s

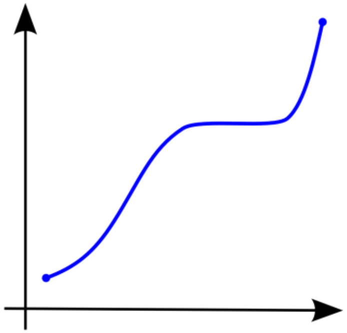
Find best local improvements

- $\pi_{i+1}(s) := \operatorname{argmax}_{a \in A} Q(\pi_i, s, a)$
  $= \operatorname{argmax}_{a \in A} R(s, a) + \gamma \sum_{s' \in S} P(s, a, s') E(\pi_i, s')$
- **Best action** in **any** given state s given expected utilities of **old** policy π_i

- If $\pi_{i+1} = \pi_i$ then exit
  - No local improvement possible,
    so the solution is optimal
- Otherwise
  - This is a new policy π_{i+1} – with **new** expected utilities!
  - Iterate, calculate **those** utilities, …

Convergence

- **Converges** in a finite number of iterations!
  - We change which action to execute
    if this **improves expected (pseudo-)utility** for this state
    - This can sometimes increase,
      and **never decrease**, the utility of the policy in other states!
    - So utilities are **monotonically improving**
      and we only have to consider a finite number of policies



- In general:
  - May take **many** iterations
  - Each iteration involved can be slow
  - Mainly because of the need to **solve a large equation system**!

Avoiding Equation Systems

Avoiding Equation Systems

- Plain policy iteration:
  - In every iteration i we have a policy π_i, want its expected utilities $E(\pi_i, s)$
  - Can use an **equation system** or **iterate until convergence**:
    - $E_{i,\,0}(\pi_i, s) = 0$ for all s

Finite horizon:
Exact expected utility for 0 steps

    - Then iterate for j=0, 1, 2, … and for all states s:

$$E_{i,\,j+1}(\pi_i, s) = R(s, \pi_i(s)) + \gamma \left(\sum_{s' \in S} P(s, \pi_i(s), s')\, E_{i,j}\,(\pi_i, s') \right)$$

Definite reward

Prob. of outcome

Reward from *prev.* iteration

Exact exp. utility for 1 step, 2 steps, 3 steps, …

  - Will *converge* in the limit ($j \to \infty$)
    - $\gamma < 1$ ➔ steps sufficiently far into the future are almost irrelevant
    - Stop when $E_{i,j+1}$ is **very close** to $E_{i,j}$ – then we're *close* to $E(\pi_i, s)$

Avoiding Equation Systems (2)

- Finally, the *approximated* utility function $E_{i,n}$ determines the best actions to use
  - Previously:

$$\pi_{i+1}(s) = \arg\max_{a \in A} Q(\pi_i, s, a)$$

$$= \arg\max_{a \in A} \left(R(s,a) + \gamma \sum_{s' \in S} P(s,a,s') E(\pi_i, s) \right)$$

True expected cost

  - Approximated:

$$\pi_{i+1}(s) = \arg\max_{a \in A} \left(R(s,a) + \gamma \sum_{s' \in S} P(s,a,s') E_{i,n}(\pi_i, s) \right)$$

Approximate expected cost

Finding a Solution (Optimal Policy): Algorithm 2, Value Iteration

Value Iteration (1)

- Another algorithm: **Value iteration** – no policy used!
  - What's the max expected utility of executing **0 steps** starting in any state?
    - No rewards, no costs
    - For all states $s \in S$, set $V_0(s) = 0$
  - What's the max expected utility of executing **1 step** starting in any state?
    - Choose one action;
      max utility of executing remaining 0 actions in resulting state is known

$$V_1(s) = \max_{a \in A}\left(R(s,a) + \gamma \sum_{s' \in S} P(s,a,s')\, V_0(s) \right)$$

Value Iteration (2)

- Long formulas again…
  - Let’s abbreviate this…
    - $V_1(s) = \max_{a \in A} \left(R(s,a) + \gamma\ \sum_{s' \in S} P(s,a,s')\ V_0(s)\right)$
  - By defining some ***non-standard*** notation:
    - $Q(V_i, s, a) = R(s,a) + \gamma\ \sum_{s' \in S} P(s,a,s')\ V_i(s)$
  - So that:
    - $V_1(s) = \max_{a \in A} Q(V_0, s, a)$
  - Then what's the max expected utility of executing ***$j + 1$* steps**?
    - Choose one action;
      max utility of executing remaining *j* actions in resulting state is known

$$V_{j+1}(s) = \max_{a \in A}\ Q\left(V_j, s, a\right) = \max_{a \in A}\left(R(s,a) + \gamma \sum_{s' \in S} P(s,a,s')\ V_j\ (s)\right)$$

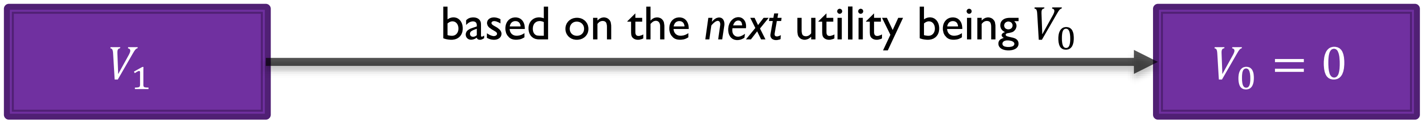
Maximizes **expected finite-horizon utility**

Value Iteration (3)

- Notice: In essence, we find actions in inverse order
  - Best expected utility with a horizon of zero steps?

$V_0 = 0$

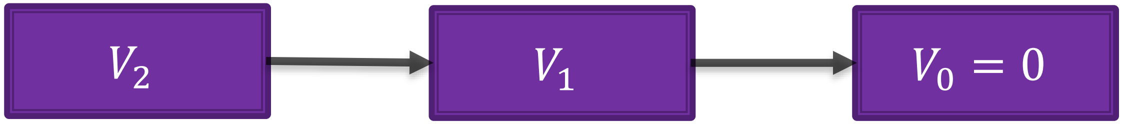
  - One step?


V_1

Maximize V_1: Choose an action based on the *next* utility being V_0

$V_0 = 0$

  - Two steps?


V_2

V_1

$V_0 = 0$

Value Iteration (4)

- Notice: $V_j(s)$ is **not** the expected value of a **policy**
  - For a given state s, a policy π always uses the **same** action $\pi(s)$, but value iteration **chooses** an action separately for every step
    - Based on **different information** each time: Iteration j+1 based on iteration j

$$V_{j+1}(s) = \max_{a \in A}\Big(Q\big(V_j, s, a\big)\Big)$$

$$V_{j+1}(s) = \max_{a \in A}\left(R(s,a) + \gamma \sum_{s' \in S} P(s,a,s')\, V_j\,(s)\right)$$

    - Iterations j and j $+\ 1$ could use *different* actions for the same state s

Value Iteration (5)

- **Expected finite-horizon utility:**
  - $V_{j+1}(s) = \max_{a\in A}\Big(Q\big(V_j, s, a\big) = R(s,a) + \gamma \sum_{s'\in S} P(s,a,s')\, V_j\,(s)\Big)$
  - Corresponds to **best possible action choice** in **each step** given that you **will** execute **exactly j+1 actions**
- As $j \rightarrow \infty$:
  - **Converges** towards the **expected utility** of an **optimal policy** for **infinite execution**
    - Will converge **faster** if $V_0(s)$ is close to the true value function
    - **Will** actually converge regardless of the initial value of $V_0(s)$, **despite** not corresponding to a policy
  - **Intuition**: As $j \rightarrow \infty$, the discount factor ensures…
    - Unconsidered actions in the distant future become irrelevant
    - As the value function converges, the implicit action choices will converge

Value Iteration (6)

- **Expected finite-horizon utility:**
  - $V_{j+1}(s) = \max_{a \in A}\Big(Q\big(V_j, s, a\big) = R(s,a) + \gamma\ \sum_{s' \in S} P(s,a,s')\, V_j\,(s)\Big)$
  - Policy extraction from a value function V_k:
    for all s,

$$\pi(s) = \arg\max_{a \in A}\big(Q(V_k, s, a)\big)$$

$$\pi(s) = \arg\max_{a \in A}\left(R(s,a) + \gamma \sum_{s' \in S} P(s,a,s')\, V_k\,(s') \right)$$

Value Iteration (7)

- Main difference:
  - With policy iteration
    - Find a **policy**
    - Find *exact* expected **utilities** for infinite steps using this policy (expensive, but gives the *best* possible basis for improvement)
      - Use these to generate a new **policy**
    - *Throw away* the old utilities,
      find *exact* expected **utilities** for infinite steps using the *new* policy
      - Use these to generate a new **policy**
    - …

  - With value iteration, if $V_0(s) = 0$ for all s:
    - Find *exact* expected **utilities** for 0 steps; *implicitly* defines a policy
    - Find *exact* expected **utilities** for 1 step; *implicitly* defines a policy
    - Find *exact* expected **utilities** for 2 steps; *implicitly* defines a policy
    - …

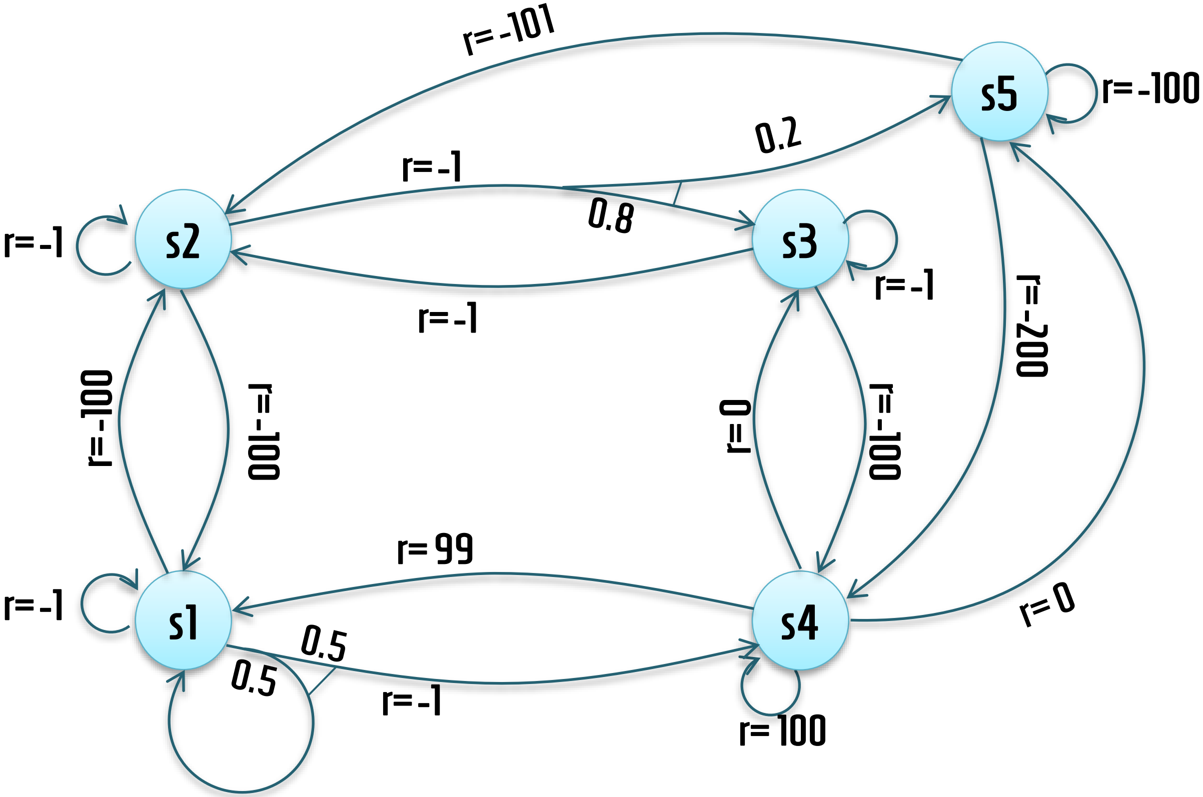
Value Iteration Example

Value Iteration Horizon:
0 actions

VI Example 1: Initial Guess V_0

- Value iteration requires an **initial value function**
  - Let's start with $V_0(s) = 0$ for each s
    - Expected utility of executing zero steps

V0(s1) = 0
V0(s2) = 0
V0(s3) = 0
V0(s4) = 0
V0(s5) = 0

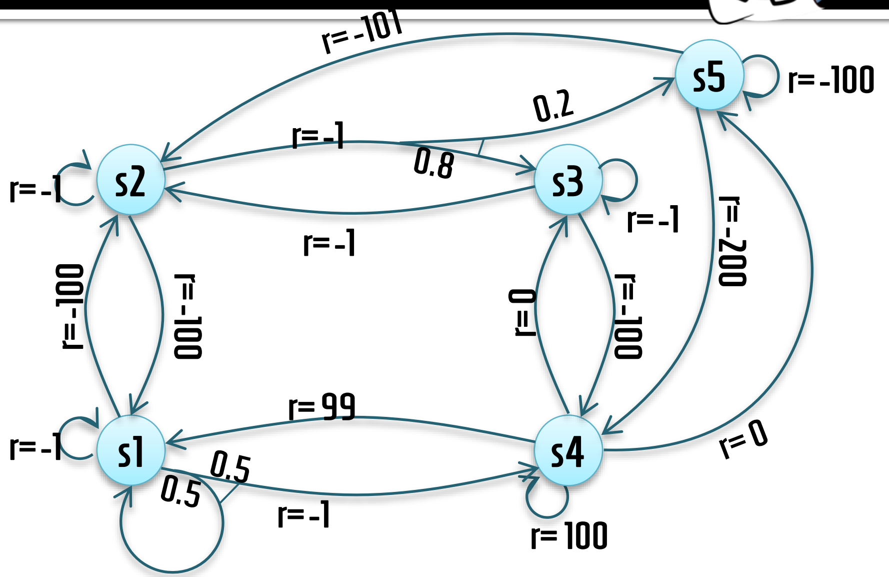


Value Iteration Horizon:
0 actions ➔ 1 action

VI Example 2: Update 1a

What is the (expected) best **first action** for each state if we then *continue* according to V_0?

$V_0(s1) = 0$
$V_0(s2) = 0$
$V_0(s3) = 0$
$V_0(s4) = 0$
$V_0(s5) = 0$



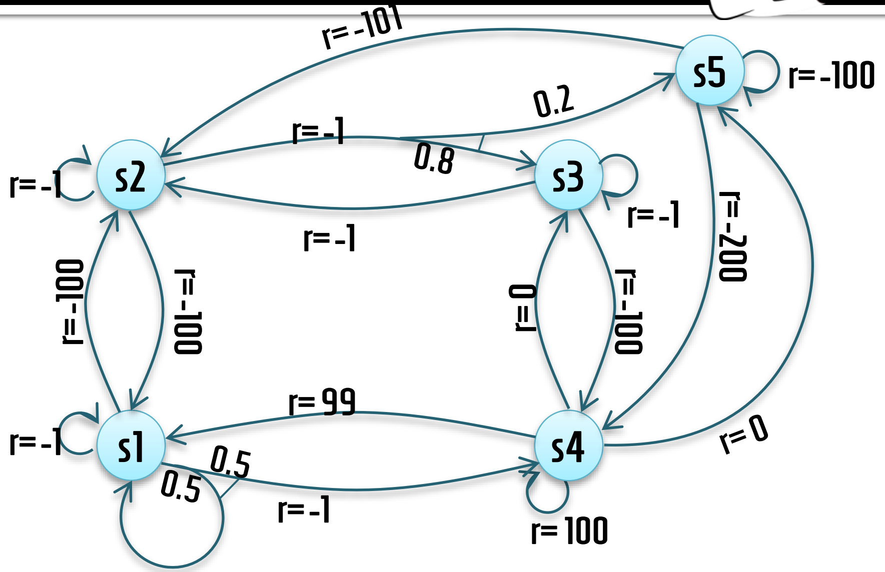
- For every state s:
  - **PI**: find $a \in A$ maximizing $Q(\pi_1, s, a) = R(s,a) + \gamma \Sigma_{s' \in S} P(s,a,s') E(\pi_1, s')$
  - **VI**: find $a \in A$ maximizing $Q(V_0, s, a) = \boxed{R(s,a)} + \gamma \boxed{\Sigma_{s' \in S} P(s,a,s') V_0(s')}$

s1:	wait	$-1 + 0.9 * 0$	$= -1$
	move(l1,l2)	$-100 + 0.9 * 0$	$= -100$
	move(l1,l4)	$-1 + 0.9 * (0.5*0 + 0.5*0)$	$= -1$
s2:	wait	$-1 + 0.9 * 0$	$= -1$
	move(l2,l1)	$-100 + 0.9 * 0$	$= -100$
	move(l2,l3)	$-1 + 0.9 * (0.8*0 + 0.2*0)$	$= -1$

VI Example 3: Update 1b

What is the (expected) best **first action** for each state if we then *continue* according to V_0?

V0(s1) = 0
V0(s2) = 0
V0(s3) = 0
V0(s4) = 0
V0(s5) = 0



- For every state s:
  - **VI**: find $a \in A$ maximizing $Q(V_0, s, a) = R(s,a) + \gamma \Sigma_{s' \in S} P(s,a,s') V_0(s')$
    - s3: wait $-1 + 0.9 * 0 = -1$
      move(l3,l2) $-1 + 0.9 * 0 = -1$
      move(l3,l4) $-100 + 0.9 * 0 = -100$
    - s4: wait $+100 + 0.9 * 0 = +100$
      move(l4,l1) $+99 + 0.9 * 0 = +99$
      …
    - s5: wait $-100 + 0.9 * 0 = -100$
      move(l5,l2) $-101 + 0.9 * 0 = -101$
      move(l5,l4) $-200 + 0.9 * 0 = -200$

VI Example 4: V_1

- This results in a **new value function**
  - **Finite horizon:** $V_1(s)$ is the (actual) expected utility of executing 1 action, making the best choices at all steps
  - **Infinite horizon**: $V_1(s)$ is our current approximation of the (actual) expected utility of following the best possible policy forever

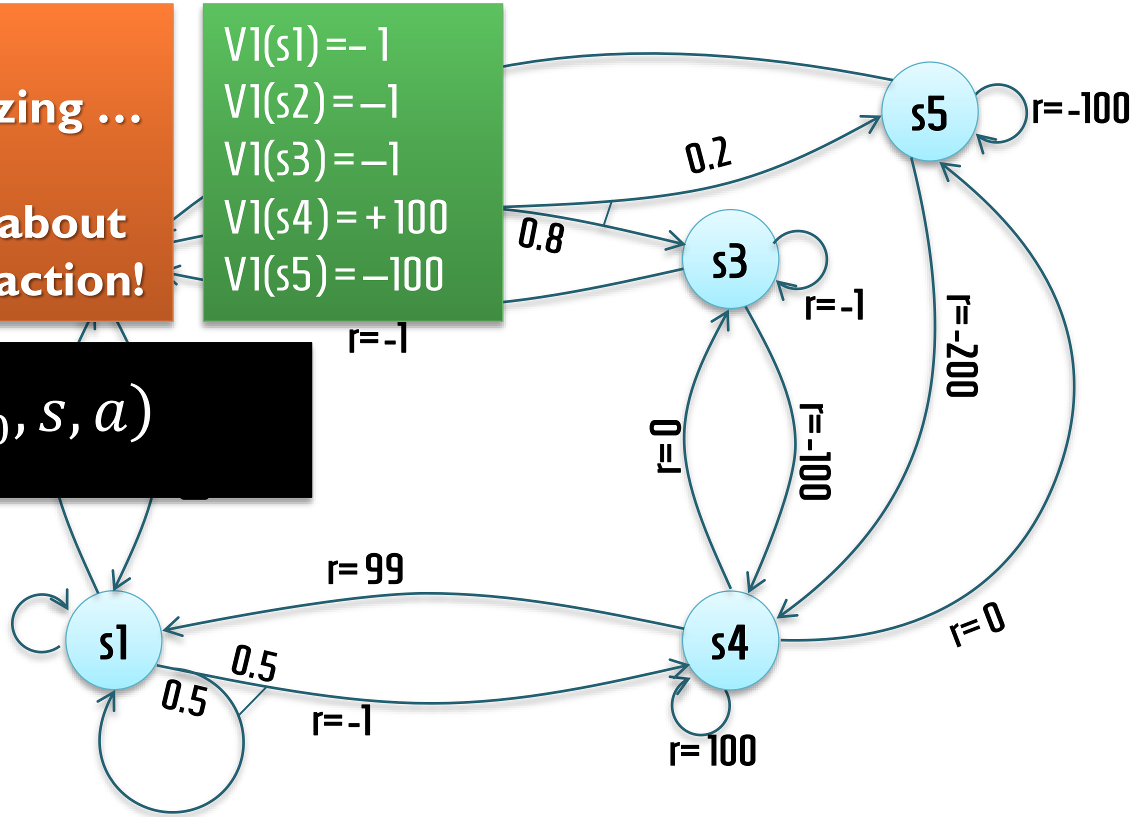
V0(s1) = 0
V0(s2) = 0
V0(s3) = 0
V0(s4) = 0
V0(s5) = 0

We said:
find $a \in A$ maximizing …

But we only care about the value, not the action!

V1(s1) =– 1
V1(s2) = –1
V1(s3) = –1
V1(s4) = + 100
V1(s5) = –100

$$V_1(s) = \max_{a \in A} Q(V_0, s, a)$$



VI Example 5: Policy

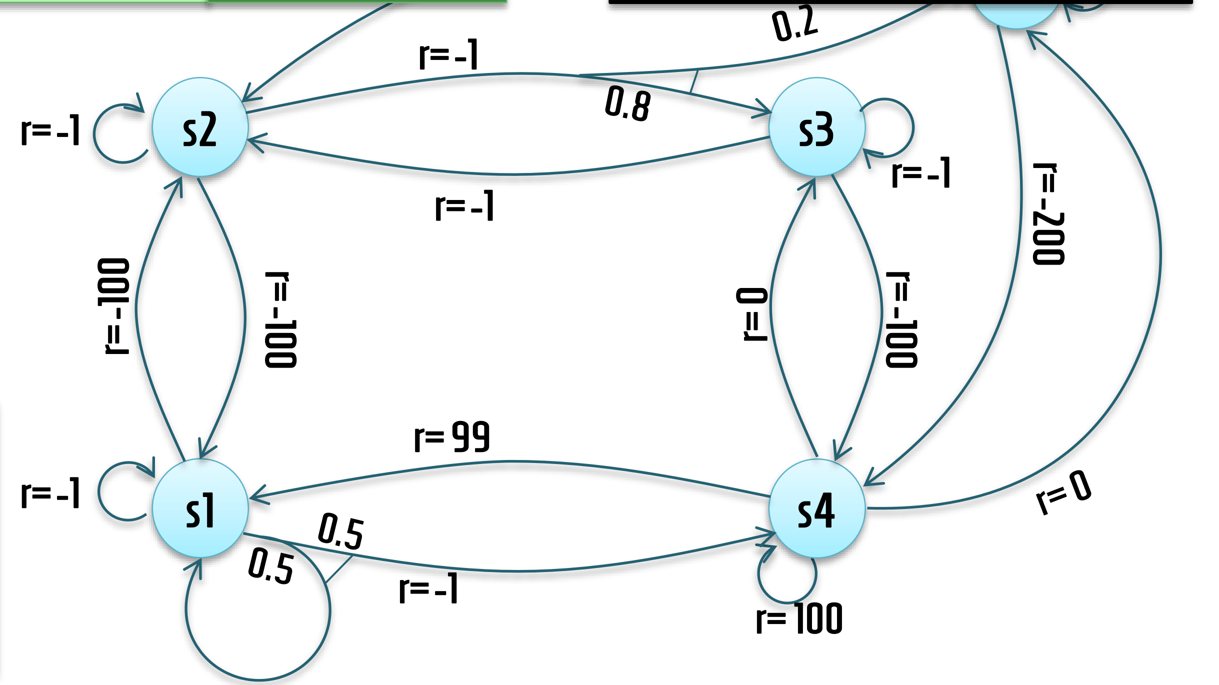
- If we **stopped** value iteration here, we could **extract** a policy π_1
  - $\pi_1(s) = \arg\max_{a \in A}\big(Q(V_1, s, a)\big)$

V0(s1) = 0
V0(s2) = 0
V0(s3) = 0
V0(s4) = 0
V0(s5) = 0

π_1 = { (s1, wait),
(s2, wait),
(s3, move(l3,l2)),
(s4, wait),
(s5, wait)}

V1(s1) =- 1
V1(s2) = –1
V1(s3) = –1
V1(s4) = + 100
V1(s5) = –100

Best expected utility for executing 1 action!
For *infinite* execution, E(π1,s1) = -10,
but this is not calculated…

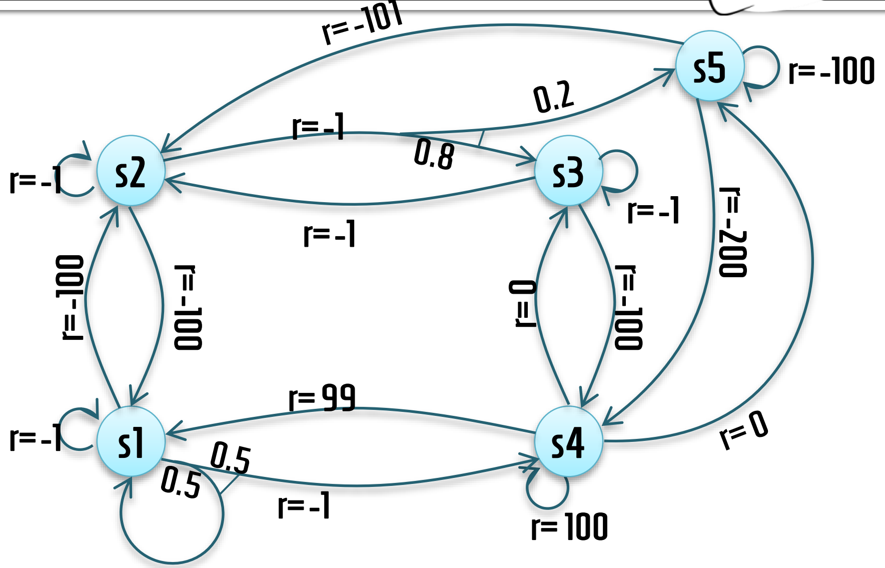


Instead we continue with the next iteration…

Value Iteration Horizon:
1 action → 2 actions

VI Example 6: Update 2a

What is the (expected) best **first action** for each state if we then *continue* according to V_1?	V1(s1) = –1 V1(s2) = –1 V1(s3) = –1 V1(s4) = +100 V1(s5) = –100



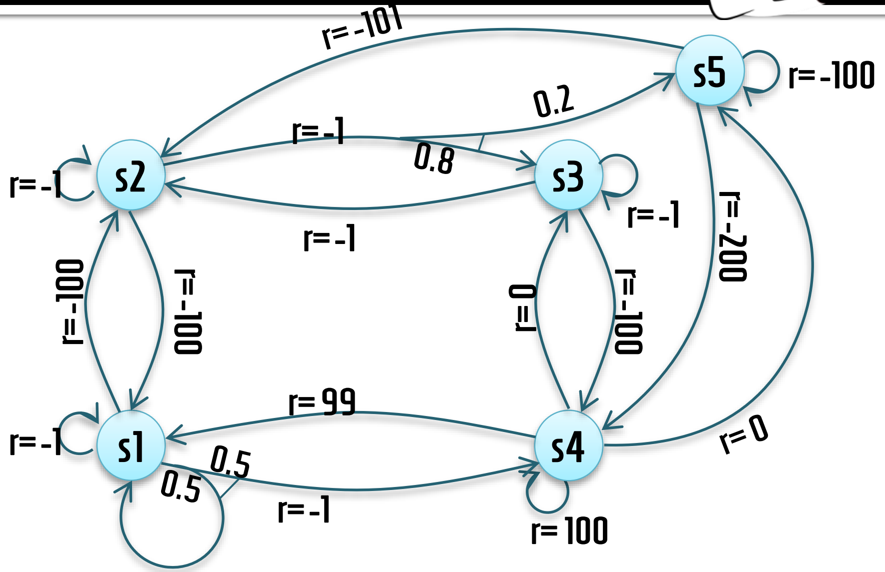
- For every state s:
  - **PI**: $a \in A$ maximizing $Q(\pi_k, s, a) = R(s,a) + \gamma \Sigma_{s' \in S} P(s,a,s') \boldsymbol{E}(\boldsymbol{\pi_k}, \boldsymbol{s'})$
  - **VI**: $a \in A$ maximizing $Q(V_{k-1}, s, a) = R(s,a) + \gamma \Sigma_{s' \in S} P(s,a,s') \boldsymbol{V_{k-1}}(\boldsymbol{s'})$

s1:	wait	–1 + 0.9	* –1	= –1.9
	move(l1,l2)	–100 + 0.9	* –1	= –100.9
	move(l1,l4)	–1 + 0.9	* (0.5*–1 + 0.5*100)	= +43,55
s2:	wait	–1 + 0.9	* –1	= –1.9
	move(l2,l1)	–100 + 0.9	* –1	= –100.9
	move(l2,l3)	–1 + 0.9	* (0.8*–1 + 0.2*–1)	= –1.9

VI Example 7: Update 2b

What is the (expected) best **first action** for each state if we then *continue* according to V_1?

V1(s1) = –1
V1(s2) = –1
V1(s3) = –1
V1(s4) = +100
V1(s5) = –100



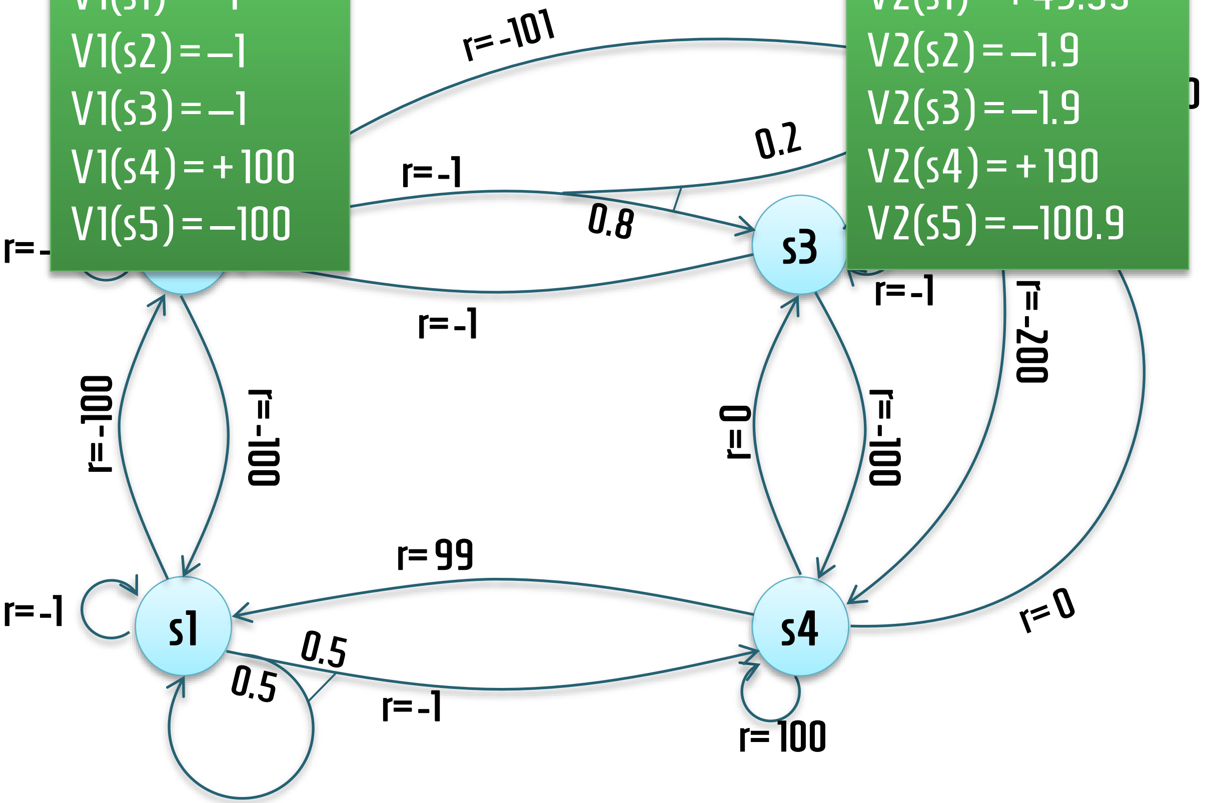
- For every state s:
  - **VI**: $a \in A$ maximizing $Q(V_{k-1}, s, a) = \boxed{R(s,a)} + \gamma \boxed{\Sigma_{s' \in S} P(s,a,s') \boldsymbol{V_{k-1}(s')}}$

s3:	wait	-1	$+ 0.9 * -1$	$= -1.9$
	move(l3,l2)	-1	$+ 0.9 * -1$	$= -1.9$
	move(l3,l4)	-100	$+ 0.9 * +100$	$= -10$
s4:	wait	$+100$	$+ 0.9 * +100$	$= +190$
	move(l4,l1)	$+99$	$+ 0.9 * -1$	$= +98.1$
	…			
s5:	wait	-100	$+ 0.9 * -1$	$= -100.9$
	move(l5,l2)	-101	$+ 0.9 * -1$	$= -101.9$
	move(l5,l4)	-200	$+ 0.9 * +100$	$= -110$

VI Example 8: V_2

- This results in another **new value function**
  - **Finite horizon:** $V_2(s)$ is the (actual) expected utility of executing 2 actions, making the best choices at all steps
  - **Infinite horizon**: $V_2(s)$ is our current approximation of the (actual) expected utility of following the best possible policy forever

V0(s1) = 0
V0(s2) = 0
V0(s3) = 0
V0(s4) = 0
V0(s5) = 0



V1(s1) = –1
V1(s2) = –1
V1(s3) = –1
V1(s4) = +100
V1(s5) = –100

V2(s1) = +43.55
V2(s2) = –1.9
V2(s3) = –1.9
V2(s4) = +190
V2(s5) = –100.9

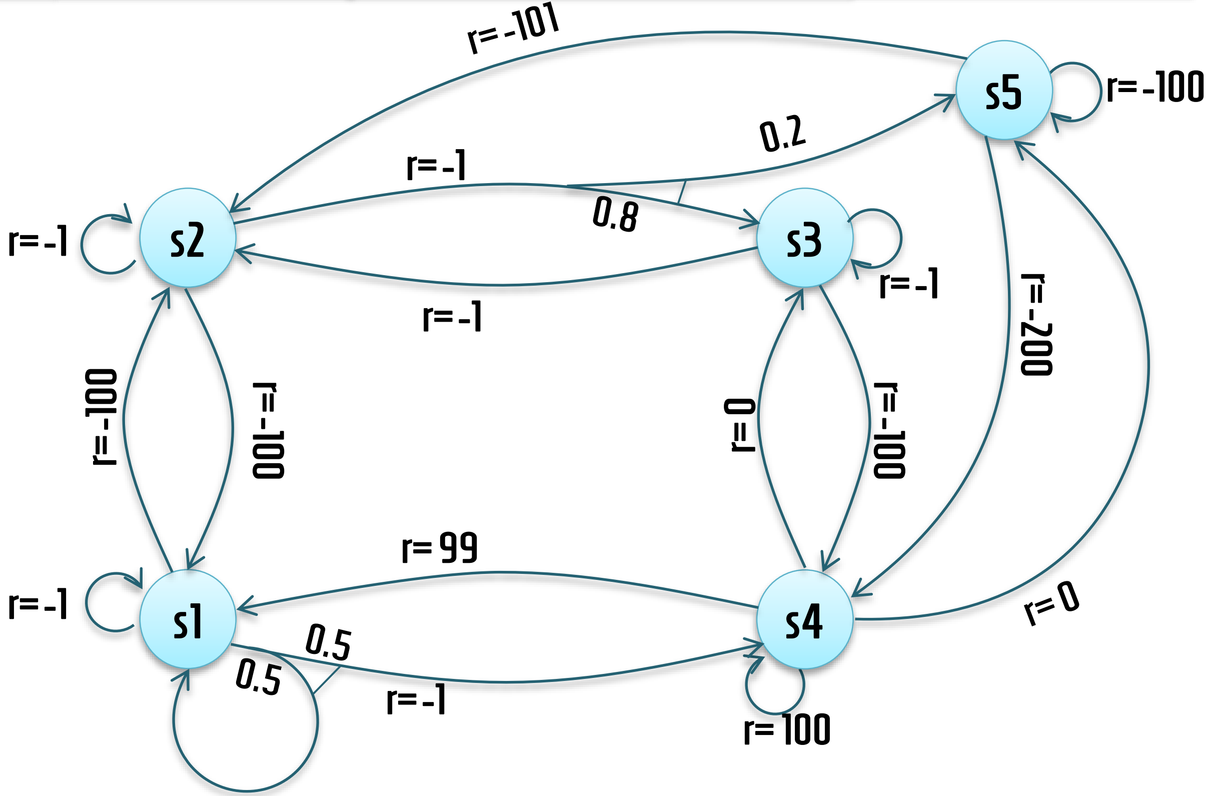
VI Example 9: Policy

- Now we have a new implicit policy

V0(s1) = 0	π_1 = { (s1, wait),	V1(s1) = –1	π_2 = { (s1, move(l1,l4)),	V2(s1) = +43.55
V0(s2) = 0	(s2, wait),	V1(s2) = –1	(s2, wait),	V2(s2) = –1.9
V0(s3) = 0	(s3, move(l3,l2)),	V1(s3) = –1	(s3, wait),	V2(s3) = –1.9
V0(s4) = 0	(s4, wait),	V1(s4) = +100	(s4, wait),	V2(s4) = +190
V0(s5) = 0	(s5, wait)}	V1(s5) = –100	(s5, wait)}	V2(s5) = –100.9

Again, V_2 doesn't represent the true (infinite) expected utility of π_2

It is the true expected utility of taking the "best" action (whatever that is!) two steps in a row



Analysis

Differences

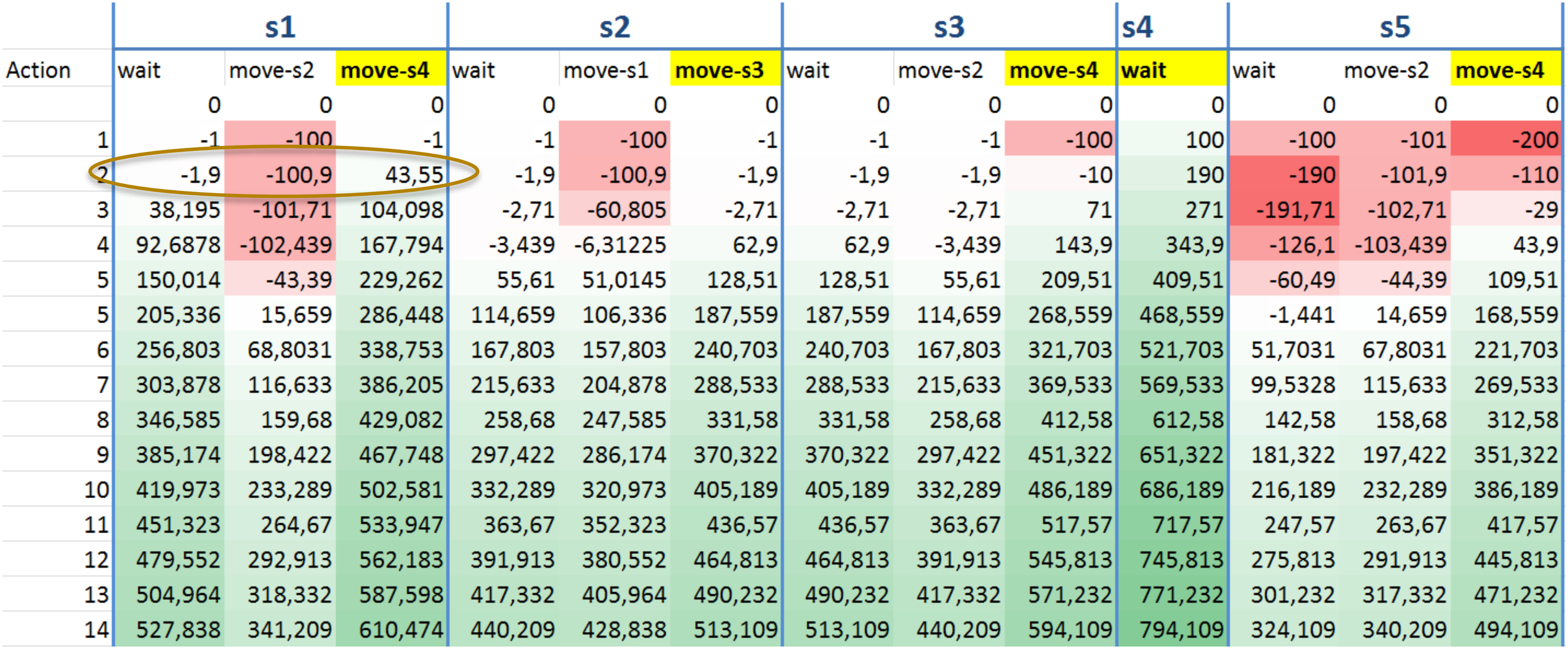
- **Significant differences** from policy iteration
  - Less accurate basis for action selection
    - Based on **finite horizon utility**, which incrementally **approximates** the true infinite horizon utility
    - ➔ Requires a larger *number* of iterations, but each iteration is *cheaper*
  - The **implicit policy** does not necessarily change in each iteration
    - May first have to iterate *n* times, incrementally improving approximations
    - **Then** another action suddenly seems better in some state
  - ➔ Need a new termination condition!
    - Cannot terminate just because the policy does not change…

Illustration

- Illustration below
  - Notice that we already calculated rows 1 and 2
    - s1: wait $-1 + 0.9 * -1$ $= -1.9$

      move(l1,l2) $-100 + 0.9 * -1$ $= -100.9$

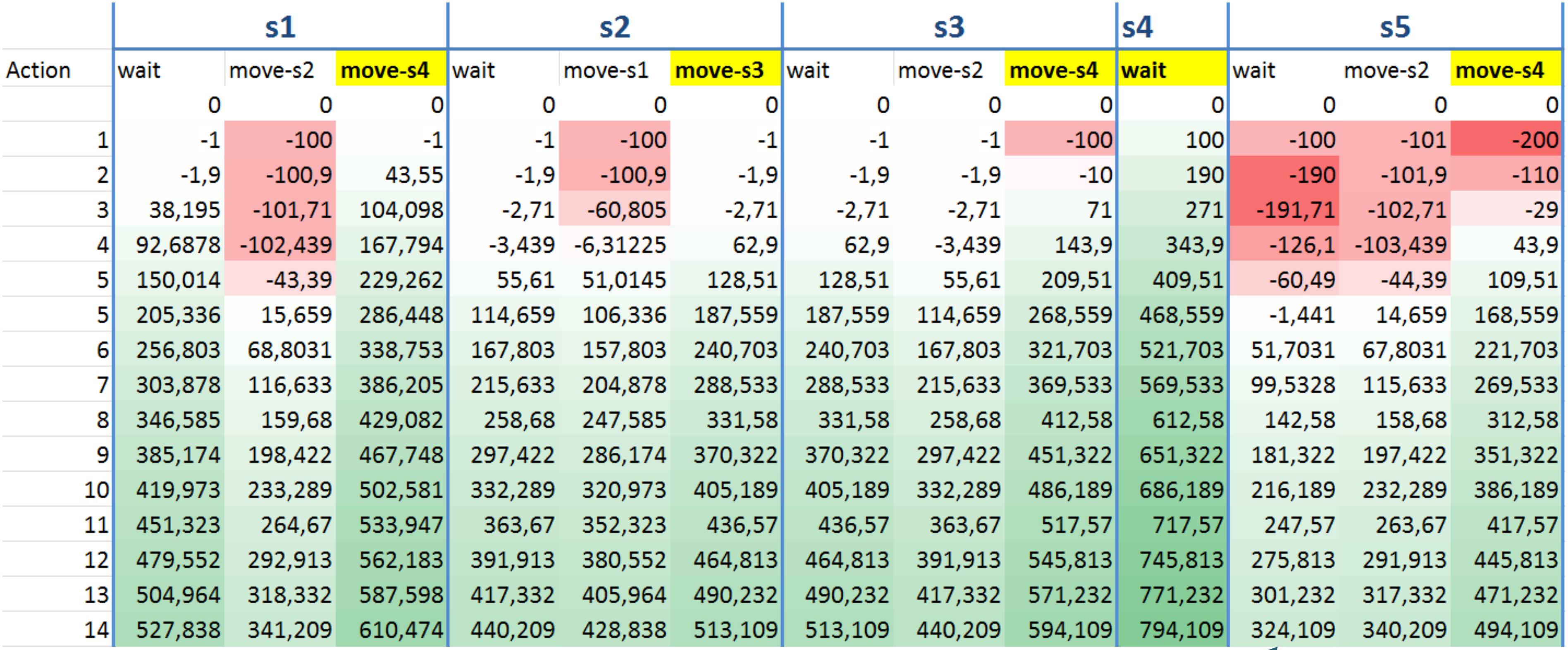
      move(l1,l4) $-1 + 0.9 * (0.5*-1 + 0.5*+100)$ $= +43,55$



	s1			s2			s3			s4	s5		
Action	wait	move-s2	move-s4	wait	move-s1	move-s3	wait	move-s2	move-s4	wait	wait	move-s2	move-s4
	0	0	0	0	0	0	0	0	0	0	0	0	0
1	-1	-100	-1	-1	-100	-1	-1	-1	-100	100	-100	-101	-200
2	-1,9	-100,9	43,55	-1,9	-100,9	-1,9	-1,9	-1,9	-10	190	-190	-101,9	-110
3	38,195	-101,71	104,098	-2,71	-60,805	-2,71	-2,71	-2,71	71	271	-191,71	-102,71	-29
4	92,6878	-102,439	167,794	-3,439	-6,31225	62,9	62,9	-3,439	143,9	343,9	-126,1	-103,439	43,9
5	150,014	-43,39	229,262	55,61	51,0145	128,51	128,51	55,61	209,51	409,51	-60,49	-44,39	109,51
5	205,336	15,659	286,448	114,659	106,336	187,559	187,559	114,659	268,559	468,559	-1,441	14,659	168,559
6	256,803	68,8031	338,753	167,803	157,803	240,703	240,703	167,803	321,703	521,703	51,7031	67,8031	221,703
7	303,878	116,633	386,205	215,633	204,878	288,533	288,533	215,633	369,533	569,533	99,5328	115,633	269,533
8	346,585	159,68	429,082	258,68	247,585	331,58	331,58	258,68	412,58	612,58	142,58	158,68	312,58
9	385,174	198,422	467,748	297,422	286,174	370,322	370,322	297,422	451,322	651,322	181,322	197,422	351,322
10	419,973	233,289	502,581	332,289	320,973	405,189	405,189	332,289	486,189	686,189	216,189	232,289	386,189
11	451,323	264,67	533,947	363,67	352,323	436,57	436,57	363,67	517,57	717,57	247,57	263,67	417,57
12	479,552	292,913	562,183	391,913	380,552	464,813	464,813	391,913	545,813	745,813	275,813	291,913	445,813
13	504,964	318,332	587,598	417,332	405,964	490,232	490,232	417,332	571,232	771,232	301,232	317,332	471,232
14	527,838	341,209	610,474	440,209	428,838	513,109	513,109	440,209	594,109	794,109	324,109	340,209	494,109

Illustration

- Remember, these are **finite horizon** utilities!

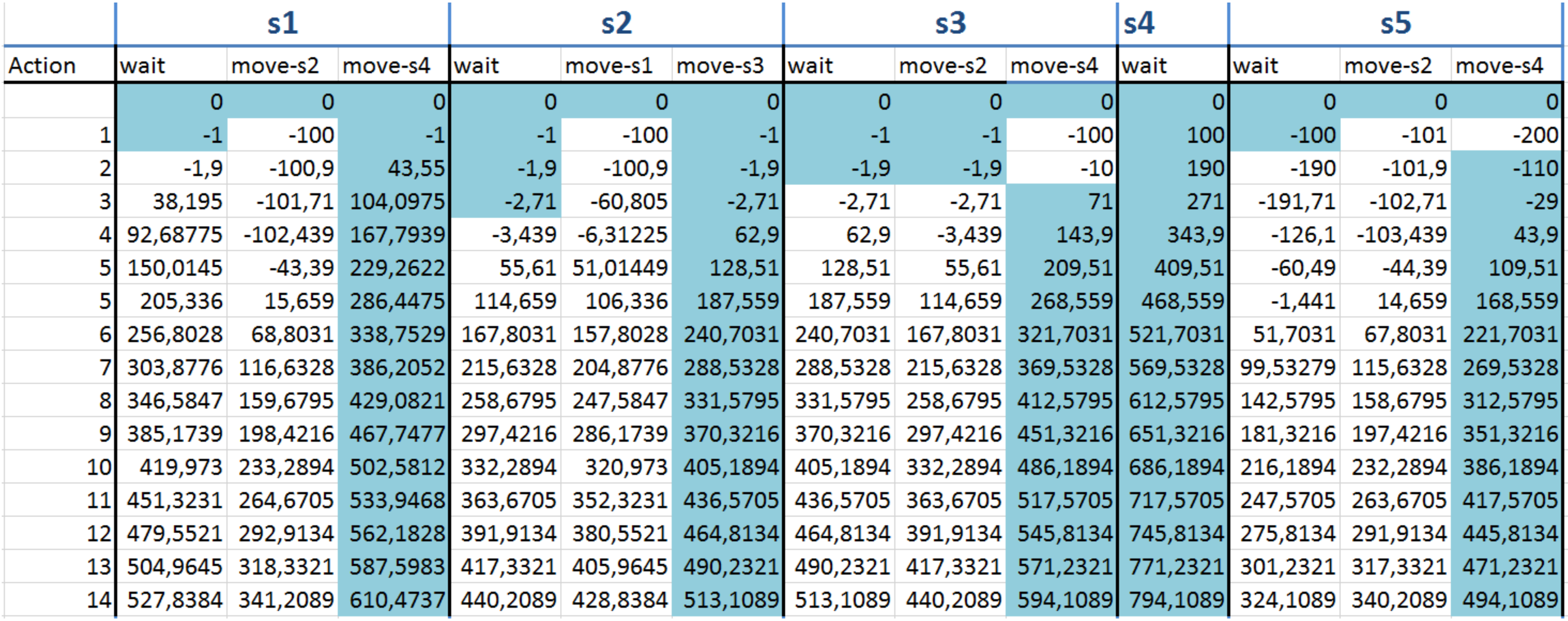


	s1			s2			s3			s4	s5		
Action	wait	move-s2	**move-s4**	wait	move-s1	**move-s3**	wait	move-s2	**move-s4**	**wait**	wait	move-s2	**move-s4**
	0	0	0	0	0	0	0	0	0	0	0	0	0
1	-1	-100	-1	-1	-100	-1	-1	-1	-100	100	-100	-101	-200
2	-1,9	-100,9	43,55	-1,9	-100,9	-1,9	-1,9	-1,9	-10	190	-190	-101,9	-110
3	38,195	-101,71	104,098	-2,71	-60,805	-2,71	-2,71	-2,71	71	271	-191,71	-102,71	-29
4	92,6878	-102,439	167,794	-3,439	-6,31225	62,9	62,9	-3,439	143,9	343,9	-126,1	-103,439	43,9
5	150,014	-43,39	229,262	55,61	51,0145	128,51	128,51	55,61	209,51	409,51	-60,49	-44,39	109,51
5	205,336	15,659	286,448	114,659	106,336	187,559	187,559	114,659	268,559	468,559	-1,441	14,659	168,559
6	256,803	68,8031	338,753	167,803	157,803	240,703	240,703	167,803	321,703	521,703	51,7031	67,8031	221,703
7	303,878	116,633	386,205	215,633	204,878	288,533	288,533	215,633	369,533	569,533	99,5328	115,633	269,533
8	346,585	159,68	429,082	258,68	247,585	331,58	331,58	258,68	412,58	612,58	142,58	158,68	312,58
9	385,174	198,422	467,748	297,422	286,174	370,322	370,322	297,422	451,322	651,322	181,322	197,422	351,322
10	419,973	233,289	502,581	332,289	320,973	405,189	405,189	332,289	486,189	686,189	216,189	232,289	386,189
11	451,323	264,67	533,947	363,67	352,323	436,57	436,57	363,67	517,57	717,57	247,57	263,67	417,57
12	479,552	292,913	562,183	391,913	380,552	464,813	464,813	391,913	545,813	745,813	275,813	291,913	445,813
13	504,964	318,332	587,598	417,332	405,964	490,232	490,232	417,332	571,232	771,232	301,232	317,332	471,232
14	527,838	341,209	610,474	440,209	428,838	513,109	513,109	440,209	594,109	794,109	324,109	340,209	494,109

324.109 = reward of waiting **once** in s5,
then making the best **finite horizon** decisions for 14 steps,
under the assumption that you will then do **nothing**!

Illustration

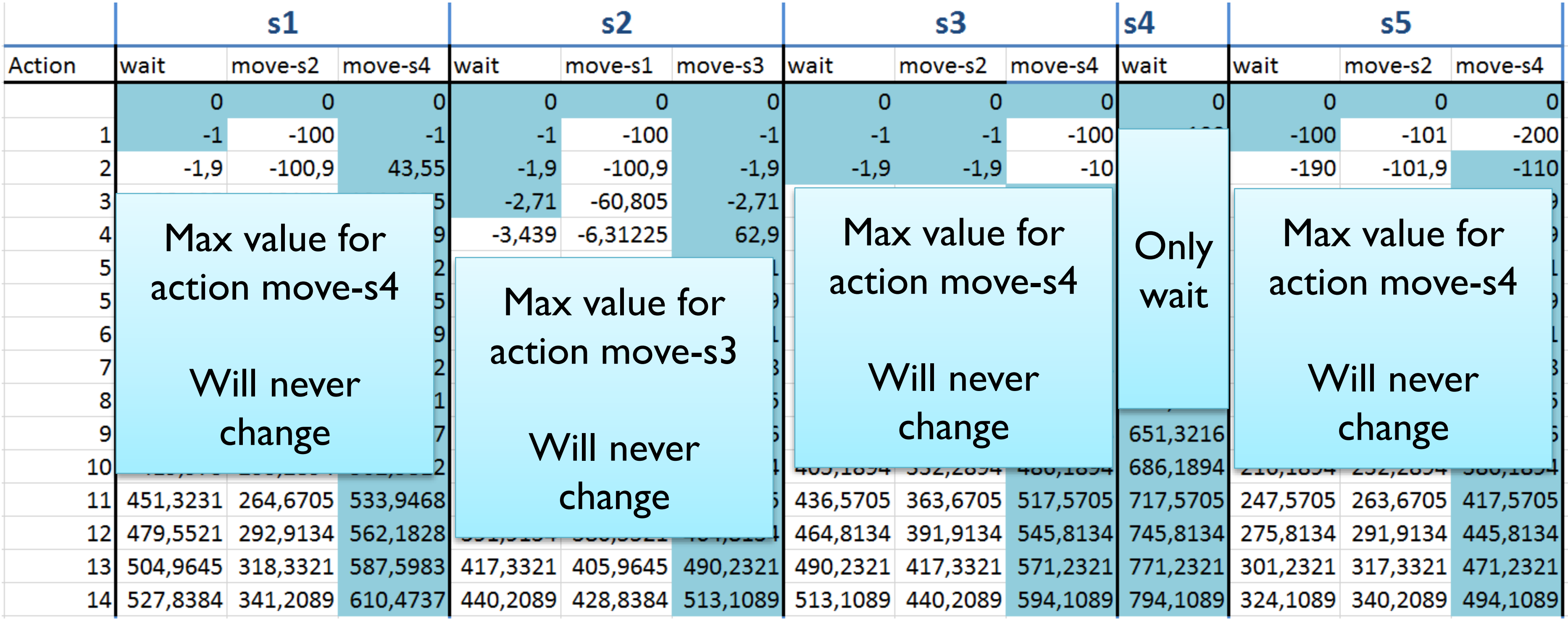
- The policy implicit in the value function changes incrementally…
  - Blue highlight: Optimal action choices in each step
  - Sometimes multiple choices are optimal!



	s1			s2			s3			s4	s5		
Action	wait	move-s2	move-s4	wait	move-s1	move-s3	wait	move-s2	move-s4	wait	wait	move-s2	move-s4
	0	0	0	0	0	0	0	0	0	0	0	0	0
1	-1	-100	-1	-1	-100	-1	-1	-1	-100	100	-100	-101	-200
2	-1,9	-100,9	43,55	-1,9	-100,9	-1,9	-1,9	-1,9	-10	190	-190	-101,9	-110
3	38,195	-101,71	104,0975	-2,71	-60,805	-2,71	-2,71	-2,71	71	271	-191,71	-102,71	-29
4	92,68775	-102,439	167,7939	-3,439	-6,31225	62,9	62,9	-3,439	143,9	343,9	-126,1	-103,439	43,9
5	150,0145	-43,39	229,2622	55,61	51,01449	128,51	128,51	55,61	209,51	409,51	-60,49	-44,39	109,51
5	205,336	15,659	286,4475	114,659	106,336	187,559	187,559	114,659	268,559	468,559	-1,441	14,659	168,559
6	256,8028	68,8031	338,7529	167,8031	157,8028	240,7031	240,7031	167,8031	321,7031	521,7031	51,7031	67,8031	221,7031
7	303,8776	116,6328	386,2052	215,6328	204,8776	288,5328	288,5328	215,6328	369,5328	569,5328	99,53279	115,6328	269,5328
8	346,5847	159,6795	429,0821	258,6795	247,5847	331,5795	331,5795	258,6795	412,5795	612,5795	142,5795	158,6795	312,5795
9	385,1739	198,4216	467,7477	297,4216	286,1739	370,3216	370,3216	297,4216	451,3216	651,3216	181,3216	197,4216	351,3216
10	419,973	233,2894	502,5812	332,2894	320,973	405,1894	405,1894	332,2894	486,1894	686,1894	216,1894	232,2894	386,1894
11	451,3231	264,6705	533,9468	363,6705	352,3231	436,5705	436,5705	363,6705	517,5705	717,5705	247,5705	263,6705	417,5705
12	479,5521	292,9134	562,1828	391,9134	380,5521	464,8134	464,8134	391,9134	545,8134	745,8134	275,8134	291,9134	445,8134
13	504,9645	318,3321	587,5983	417,3321	405,9645	490,2321	490,2321	417,3321	571,2321	771,2321	301,2321	317,3321	471,2321
14	527,8384	341,2089	610,4737	440,2089	428,8384	513,1089	513,1089	440,2089	594,1089	794,1089	324,1089	340,2089	494,1089

Illustration

- At some point we reach the final recommendation/policy:



	s1			s2			s3			s4	s5		
Action	wait	move-s2	move-s4	wait	move-s1	move-s3	wait	move-s2	move-s4	wait	wait	move-s2	move-s4
	0	0	0	0	0	0	0	0	0	0	0	0	0
1	-1	-100	-1	-1	-100	-1	-1	-1	-100		-100	-101	-200
2	-1,9	-100,9	43,55	-1,9	-100,9	-1,9	-1,9	-1,9	-10		-190	-101,9	-110
3				-2,71	-60,805	-2,71							
4				-3,439	-6,31225	62,9							
5													
5													
6													
7													
8													
9										651,3216			
10										686,1894			
11	451,3231	264,6705	533,9468				436,5705	363,6705	517,5705	717,5705	247,5705	263,6705	417,5705
12	479,5521	292,9134	562,1828				464,8134	391,9134	545,8134	745,8134	275,8134	291,9134	445,8134
13	504,9645	318,3321	587,5983	417,3321	405,9645	490,2321	490,2321	417,3321	571,2321	771,2321	301,2321	317,3321	471,2321
14	527,8384	341,2089	610,4737	440,2089	428,8384	513,1089	513,1089	440,2089	594,1089	794,1089	324,1089	340,2089	494,1089

Max value for action move-s4

Will never change

Max value for action move-s3

Will never change

Max value for action move-s4

Will never change

Only wait

Max value for action move-s4

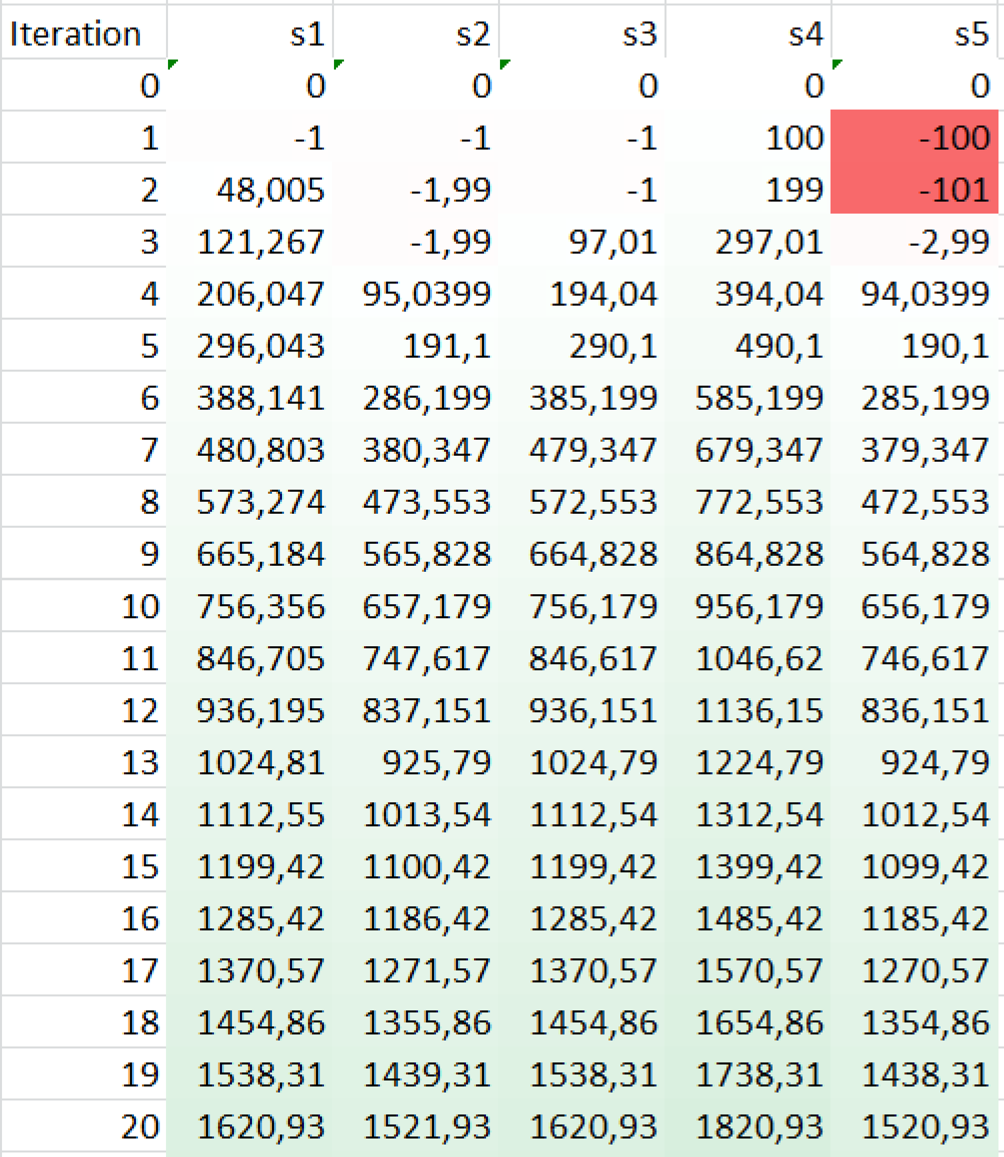
Will never change

Optimal infinite horizon policy corresponds to iteration 4

Can't be seen directly in rows 0-4:
We don't know how the approximation will change
Maybe one action will soon "overtake" another!

Different Discount Factors

- Suppose discount factor is 0.99 instead
  - Illustration, only showing **best** finite horizon utility (for the best action choice) at each iteration
  - Much slower convergence
    - Change at step 20: 2% ➔ 5%
    - Change at step 50: 0.07% ➔ 1.63%
  - Care more about the future ➔ need to consider many more steps!



Iteration	s1	s2	s3	s4	s5
0	0	0	0	0	0
1	-1	-1	-1	100	-100
2	48,005	-1,99	-1	199	-101
3	121,267	-1,99	97,01	297,01	-2,99
4	206,047	95,0399	194,04	394,04	94,0399
5	296,043	191,1	290,1	490,1	190,1
6	388,141	286,199	385,199	585,199	285,199
7	480,803	380,347	479,347	679,347	379,347
8	573,274	473,553	572,553	772,553	472,553
9	665,184	565,828	664,828	864,828	564,828
10	756,356	657,179	756,179	956,179	656,179
11	846,705	747,617	846,617	1046,62	746,617
12	936,195	837,151	936,151	1136,15	836,151
13	1024,81	925,79	1024,79	1224,79	924,79
14	1112,55	1013,54	1112,54	1312,54	1012,54
15	1199,42	1100,42	1199,42	1399,42	1099,42
16	1285,42	1186,42	1285,42	1485,42	1185,42
17	1370,57	1271,57	1370,57	1570,57	1270,57
18	1454,86	1355,86	1454,86	1654,86	1354,86
19	1538,31	1439,31	1538,31	1738,31	1438,31
20	1620,93	1521,93	1620,93	1820,93	1520,93

How Many Iterations?

- We can find bounds!
  - Let ε be the greatest change in pseudo-utility between two iterations:

$$\epsilon = \max_{s \in S} |V_{new}(s) - V_{old}(s)|$$

  - Then if we extract a policy π from V_{new}, we have a bound:

$$\max_{s \in S} |E(\pi, s) - E(\pi^*, s)| < 2\epsilon\gamma/(1-\gamma)$$

    - For every state, the reward of π is at most $2\epsilon\gamma/(1-\gamma)$ from the reward of an optimal policy

		Discount factor γ				
		0,5	0,9	0,95	0,99	0,999
Maximum absolute difference ϵ between two iterations	0,001	0,002	0,018	0,038	0,198	1,998
	0,01	0,02	0,18	0,38	1,98	19,98
	0,1	0,2	1,8	3,8	19,8	199,8
	1	2	18	38	198	1998
	5	10	90	190	990	9990
	10	20	180	380	1980	19980
	100	200	1800	3800	19800	199800

How Many Iterations? Discount 0.90

Quit after 2 iterations ➔ $V_2(s1)=43$.
Guarantee: By using the corresponding policy π_2, we lose at most 1620 compared to π^*.

Iteration	s1	s2	s3	s4	s5	Greatest change	Possible diff from optimal policy
0	0	0	0	0	0		
1	-1	-1	-1	100	-100	100	1800
2	43,55	-1,9	-1,9	190	-110	90	1620
3	104,0975	-2,71	71	271	-29	81	1458
4	167,7939	62,9	143,9	343,9	43,9	72,9	1312,2
5	229,2622	128,51	209,51	409,51	109,51	65,61	1180,98
6	286,4475	187,559	268,559	468,559	168,559	59,049	1062,882
7	338,7529	240,7031	321,7031	521,7031	221,7031	53,1441	956,5938
8	386,2052	288,5328	369,5328	569,5328	269,5328	47,82969	860,9344
9	429,0821	331,5795	412,5795	612,5795	312,5795	43,04672	774,841
10	467,7477	370,3216	451,3216	651,3216	351,3216	38,74205	697,3569
20	694,787	597,4233	678,4233	878,4233	578,4233	13,50852	243,1533
30	773,9725	676,6088	757,6088	957,6088	657,6088	4,710129	84,78232
40	801,5828	704,2191	785,2191	985,2191	685,2191	1,64232	29,56177
50	811,2099	713,8462	794,8462	994,8462	694,8462	0,572642	10,30755
60	814,5666	717,203	798,203	998,203	698,203	0,199668	3,594021
70	815,7371	718,3734	799,3734	999,3734	699,3734	0,06962	1,253157
80	816,1452	718,7815	799,7815	999,7815	699,7815	0,024275	0,436949
90	816,2875	718,9238	799,9238	999,9238	699,9238	0,008464	0,152355
100	816,3371	718,9734	799,9734	999,9734	699,9734	0,002951	0,053123

Bounds are incrementally tightened!

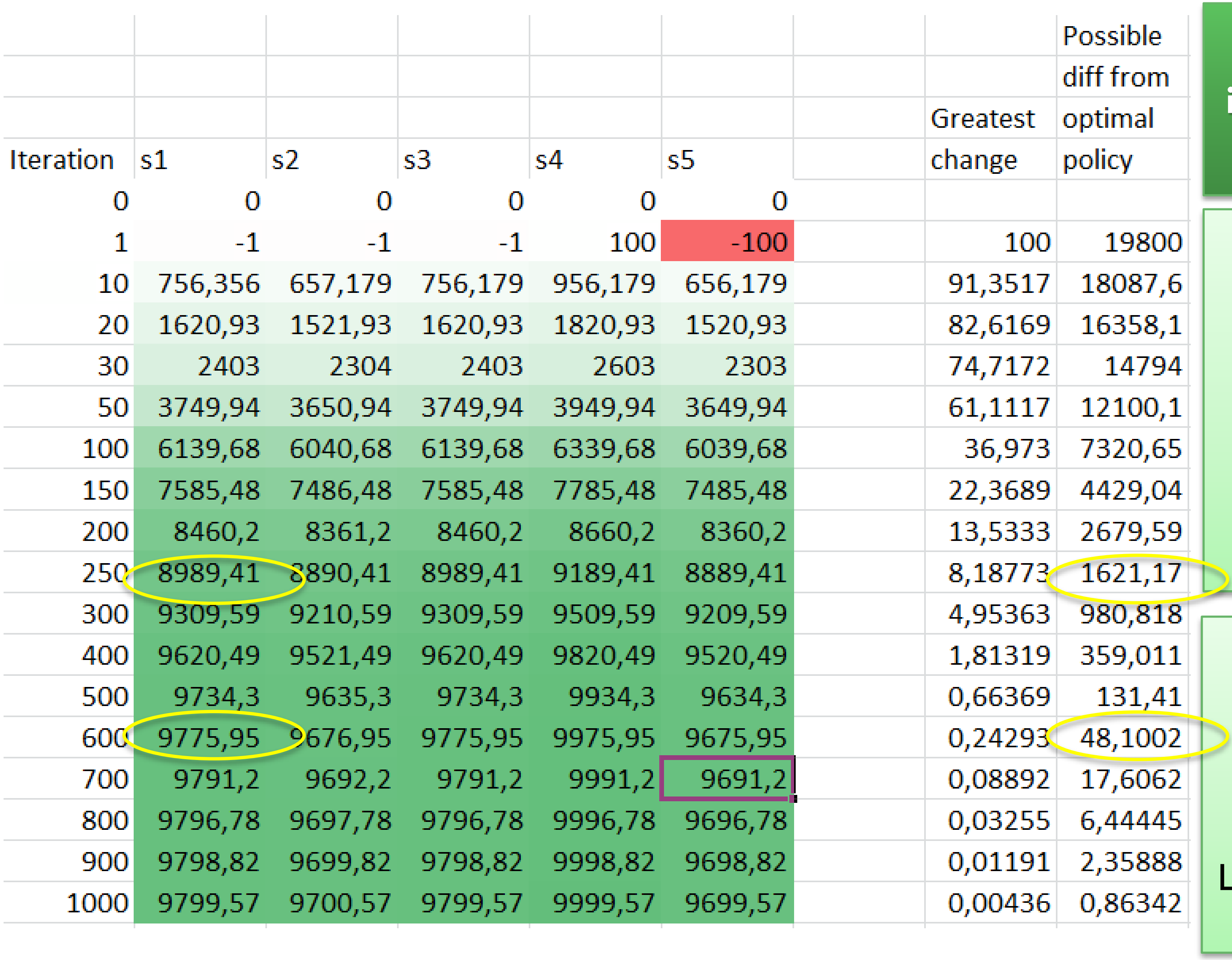
Quit after 10 iterations?

Guarantee: Lose at most 697 by using the corresponding policy π_{10}.

Quit after 50 iterations?

New guarantee: Lose at most 10 by using π_{50} (actually, $\pi_{50} = \pi_{10}$)

How Many Iterations? Discount 0.99



Iteration	s1	s2	s3	s4	s5		Greatest change	Possible diff from optimal policy
0	0	0	0	0	0			
1	-1	-1	-1	100	-100		100	19800
10	756,356	657,179	756,179	956,179	656,179		91,3517	18087,6
20	1620,93	1521,93	1620,93	1820,93	1520,93		82,6169	16358,1
30	2403	2304	2403	2603	2303		74,7172	14794
50	3749,94	3650,94	3749,94	3949,94	3649,94		61,1117	12100,1
100	6139,68	6040,68	6139,68	6339,68	6039,68		36,973	7320,65
150	7585,48	7486,48	7585,48	7785,48	7485,48		22,3689	4429,04
200	8460,2	8361,2	8460,2	8660,2	8360,2		13,5333	2679,59
250	8989,41	8890,41	8989,41	9189,41	8889,41		8,18773	1621,17
300	9309,59	9210,59	9309,59	9509,59	9209,59		4,95363	980,818
400	9620,49	9521,49	9620,49	9820,49	9520,49		1,81319	359,011
500	9734,3	9635,3	9734,3	9934,3	9634,3		0,66369	131,41
600	9775,95	9676,95	9775,95	9975,95	9675,95		0,24293	48,1002
700	9791,2	9692,2	9791,2	9991,2	9691,2		0,08892	17,6062
800	9796,78	9697,78	9796,78	9996,78	9696,78		0,03255	6,44445
900	9798,82	9699,82	9798,82	9998,82	9698,82		0,01191	2,35888
1000	9799,57	9700,57	9799,57	9999,57	9699,57		0,00436	0,86342

Bounds are incrementally tightened!

Quit after 250 iterations?

Guarantee: Lose at most 1621.

Quit after 600 iterations?

Guarantee: Lose at most 48.

Value Iteration

- Convergence?
  - On an acyclic graph, the values converge in finitely many iterations
  - On a cyclic graph, **<u>value</u>** convergence can take infinitely many iterations
  - That's why $\varepsilon > 0$ is needed

Comparison

Policy Iteration

Start: $\forall s.\pi_1(s) =$ some arbitrary action

Better guesses ➔ faster convergence

Loop: For steps $i = 1, 2, 3, \ldots$

Compute true expected utilities $E(\pi_i, s)$:

Notation: Define

$$Q(\pi_i, s, a) = R(s,a) + \gamma \sum_{s' \in S} P(s,a,s') \boldsymbol{E(\pi_i, s')}$$

Circular definition, harder

```
Step 1:
    // Solve an equation system
    solve_eq($\forall s. E(\pi_i, s) = Q(\pi_i, s, \pi_i(s))$)
    // Represents true expected costs
    // for infinite execution of $\pi_i$
Step 2:
    // Compute $\pi_{i+1}$
    for all states $s \in S$:
        $\pi_{i+1}(s) = \arg\max_{a \in A} Q(\pi_i, s, a)$
```

π_{i+1} based on true exp. utility for π_i

// Optimal yet?

If $\pi_{i+1} = \pi_i$ **then stop**

Value Iteration given $\epsilon > 0$

Start: $\forall s. V_0(s) =$ some arbitrary utility

Better guesses ➔ faster convergence

$V_0(s) = 0$ ➔ true finite horizon utility, 0 steps

Loop: For steps $i = 1, 2, 3, \ldots$

Compute expected finite-horizon utilities $V_i(s)$:

Notation: Define

$$Q(V_{i-1}, s, a) = R(s,a) + \gamma \sum_{s' \in S} P(s,a,s') \boldsymbol{V_{i-1}(s')}$$

New def. of $Q()$

No circularity

```
Step 1:
    // Use the old $i-1$ step rewards
    for all states $s \in S$:
        $V_i(s) = \max_{a \in A} Q(V_{i-1}, s, a)$
```

V_i based on finite horizon utility from V_{i-1}

// Good enough yet?

If $\max_{s \in S} |V_i(s) - V_{i-1}(s)| < \epsilon$ **then stop**

```
Finishing:
    // Compute $\pi$
    for all states $s \in S$:
        $\pi(s) = \arg\max_{a \in A} Q(V_{last}, s, a)$
```

π_i based on expected finite horizon utility

Discussion

- Both algorithms **terminate** in a **polynomial** number of iterations
  - (Assuming $\epsilon > 0$ for VI)
  - But the variable in the polynomial is ***the number of states***
  - Need to examine the **entire state space** in each iteration

- ➔ Requires significant time and space
  - Probabilistic planning is **EXPTIME-complete**,
    even for set-theoretic planning
    - (Like propositional logic: Simplified – no variables, no parameters)

  - Methods exist for **reducing the search space**,
    and for **approximating** optimal solutions

Value Iteration given $\epsilon > 0$

Start: $\forall s. V_0(s)$ = some arbitrary reward

Better guesses ➔ faster convergence
$V_0(s) = 0$ ➔ true finite horizon reward, 0 steps

```
Loop: For steps $i = 1, 2, 3, \ldots$
    Compute pseudo-utilities $V_i(s)$:
    Notation: Define
        $Q(V_{i-1}, s, a) = R(s,a) + \gamma \sum_{s' \in S} P(s,a,s') V_{i-1}(s')$
    Step 1:
        // Use the old $i-1$ step rewards
        for all states $s \in S$:
            $V_i(s) = \max_{a \in A} Q(V_{i-1}, s, a)$
    // Good enough yet?
    If $\max_{s \in S} |V_i(s) - V_{i-1}(s)| < \epsilon$ then stop
Finishing:
    // Compute $\pi$
    for all states $s \in S$:
        $\pi(s) = \arg\max_{a \in A} Q(V_{last}, s, a)$
```

New def. of $Q()$

No circularity!

V_i based on finite horizon rewards from V_{i-1}

π_i based on finite horizon rewards

Can **prioritize** some states, visit them more often!
For example, states "close to" significant changes in V…

Partial Observability

Overview

	Non-Observable: No information gained after action	**Fully Observable:** Exact outcome known after action	**Partially Observable:** Some information gained after action
Deterministic: Exact outcome known in advance	**Classical planning** (possibly with extensions) Information dimension is meaningless!		
Non-deterministic: Multiple outcomes, no probabilities	**NOND**: Conformant Planning	**FOND**: Conditional (Contingent) Planning	**POND**: Partially Observable, Non-Deterministic
Probabilistic: Multiple outcomes with probabilities	Probabilistic Conformant Planning (Non-observable MDPs: Special case of POMDPs)	Probabilistic Conditional Planning Stochastic Shortest Path Problems Markov Decision Processes (MDPs)	Partially Observable MDPs (POMDPs)

- In general:
  - Full information is the easiest
  - Partial information is the hardest!