Spring School on Control Theory and Reinforcement Learning

CWI Research Semester Programme on Control Theory and Reinforcement Learning: Connections and Challenges

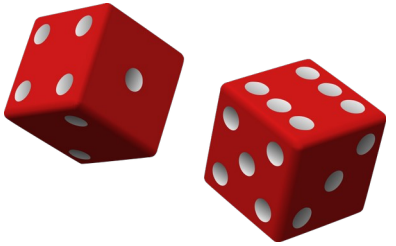
Markov decision processes, Generalized policy iteration, partial observability, abstractions

Frans A. Oliehoek

Part 1: Recap of stochastic sequential decision making problems (“MDPs”)

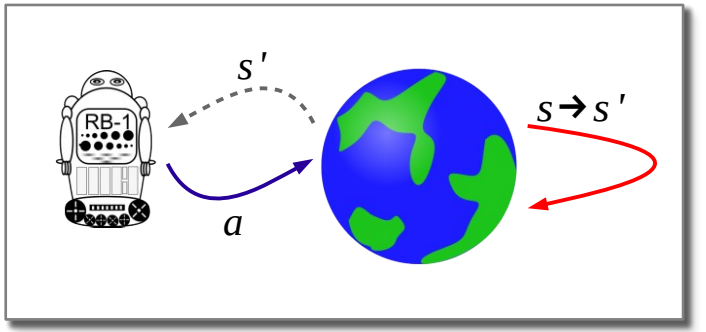
Sequential stochastic problems

- Many problems have uncertain components
  - in fastest route: traffic
  - in inventory management: sales
  - in control of a robot:
    noise in actuators, wheel slip, etc.
  - in maintenance:
    wear and tear of components



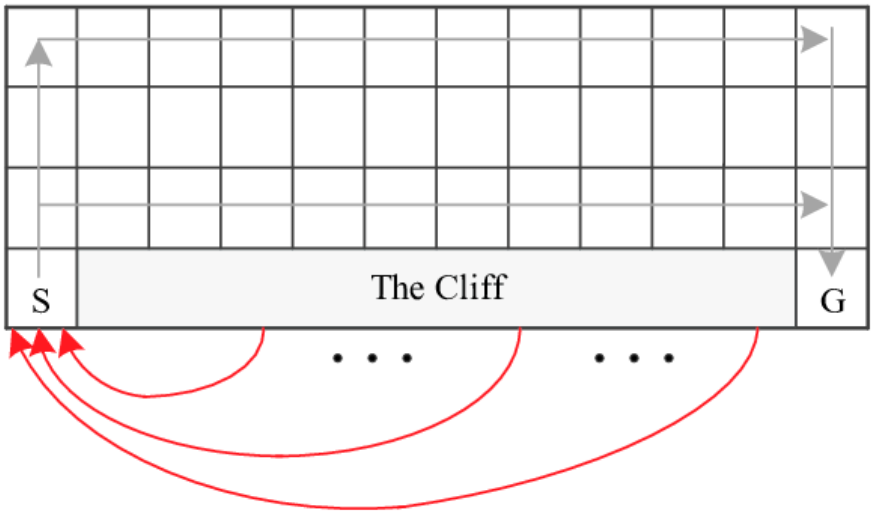
Sequential stochastic problems

- main idea:
  - discrete time steps t=0,1,2,…
  - “world” is in some state
  - and changes stochastically



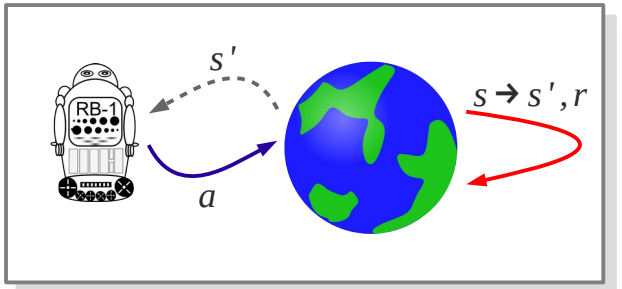
- The **horizon** (*T*) of the problem: how many steps
  - can be finite or infinite

Example: walk the cliff



- Actions:
  - Up, Down, Left, Right
  - can accidentally move to other direction
- States: Location
  - Starting location (S)
  - Terminal States
    - G: Agent is ‘reset’ at S
    - C: Agent is ‘reset’ at S
- Between resets: 1 “episode”
- Preferences with “rewards”
  - 10 for getting to the goal (G)
  - -100 for walking/falling into the cliff (C)

Policies



- So how should such policies π look like…?
  - clearly need to condition on states…
  - but they could depend on histories…?
  - they could randomize…?

- For the problems we will consider **
  it is a **mapping from state to action**: π(s) = a
  - Randomization and history are not needed
  - But a policy is a **feedback plan,** not an 'open loop' plan

**under some technical conditions

Stochastic changes of the world

- State transition function T(s,a,s')
  - Defines the world's reactions to the agent's actions
- **Markov assumption**:

  Effect of actions depends only on current state:

  $P(s_t \mid s_{t-1}\ a_{t-1}\ s_{t-2}\ a_{t-2}\ ....) = P(s_t \mid s_{t-1}\ a_{t-1}\ )$

It is a very strong assumption!

→ state has enough information to predict the future

- everything I need to know can be observed
- do not need to remember anything…

Goal ('optimality criterion')

- Optimality criterion: which policy π is best?
- Commonly: optimize long-term (sum of) rewards ('**return**'):
  - finite horizon task: $G = R_1 + R_2 + \ldots + R_T$
  - continuing task: $G = R_1 + \gamma R_2 + \gamma^2 R_3 + \ldots$
    - discount factor γ in [0,1)

- Expected return, often called **value**:
  $V(\pi) = E[\, G \mid \text{policy} = \pi \,]$

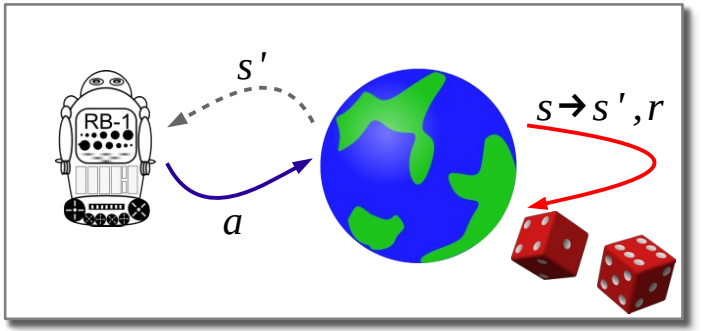
Value Functions

- We will want to express rewards **from stage *t* onwards**:
  - episodic / finite horizon**:** $G_t = R_{t+1} + R_{t+2} + \ldots + R_T$
  - continuing / infinite horizon: $G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} \ldots$

- Now, **expected (discounted) return**:
  - when acting according to policy π
  - is expressed by the **value function** of π: $v_\pi(s) = E_\pi [G_t \mid S_t=s]$
    (“cost-to-go”)

Putting it together: MDPs

- A Markov decision process (MDP) *M*=<*S,A,T,R*>
  - S – set of world states s
  - A – set of actions a
  - T – transition function
    - specifies p(s'|s,a)
    - enables: outcome uncertainty
  - R – reward function
    - Task encoded by rewards R(s,a,s')
    - also R(s,a) or R(s')



Sutton&Barto V2: combine both into a single function p(s',r|s,a)

- makes explicit stochastic rewards
- can convert this to deterministic reward function: take expectation (cf. p49 SBv2)

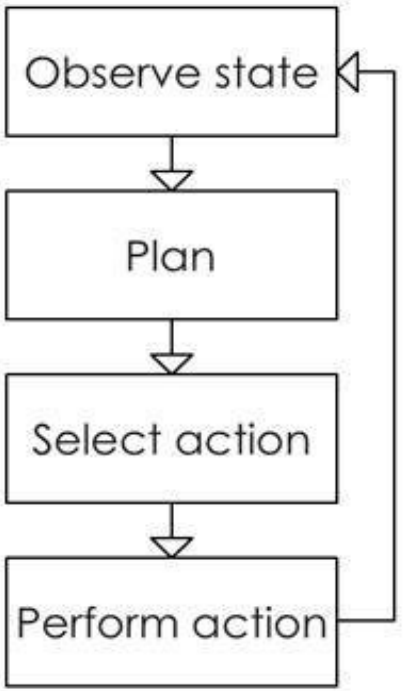
- Optimality criterion: expected (discounted) return

Part 2: finite horizon dynamic programming (on trees)

Planning for a finite horizon

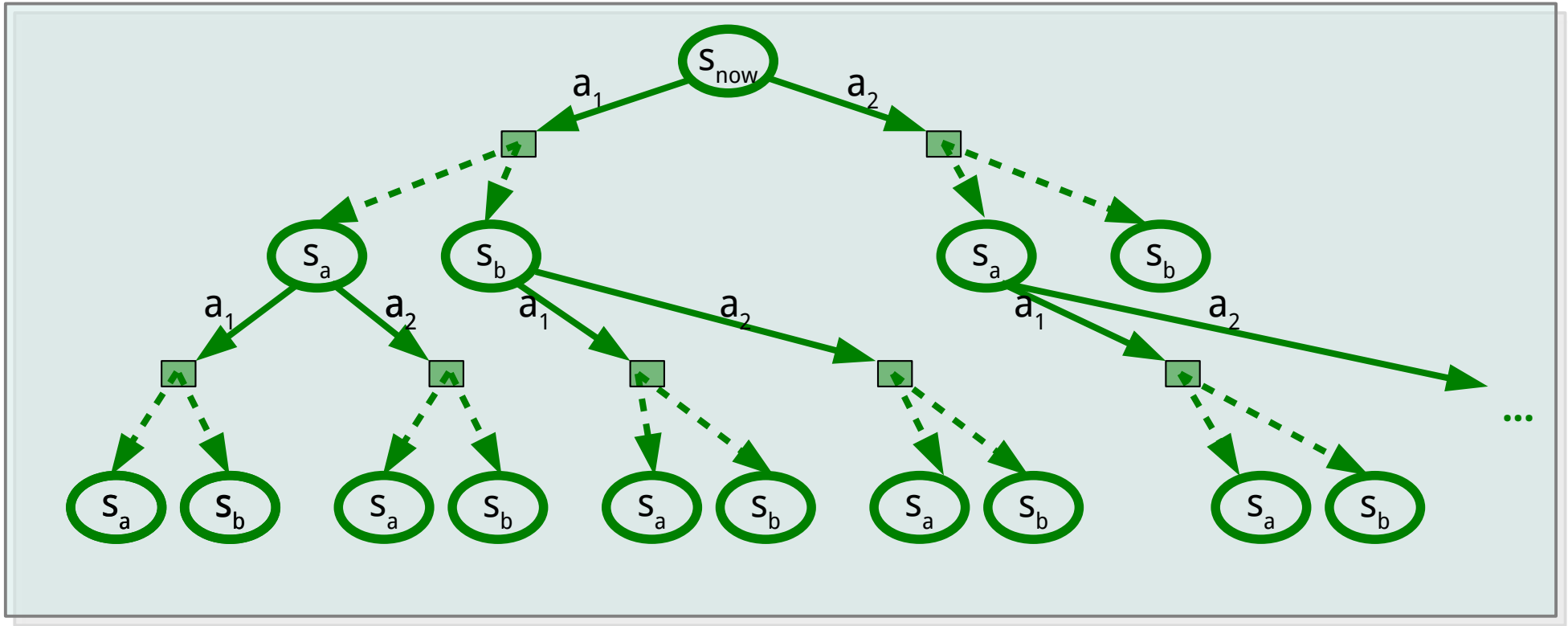
- We will only plan for T time steps
  - Finite horizon problem, or
  - online (“lookahead”) planning

- Just apply “maximum expected utility”
  - but exploit temporal structure in computation (“dynamic programming”)



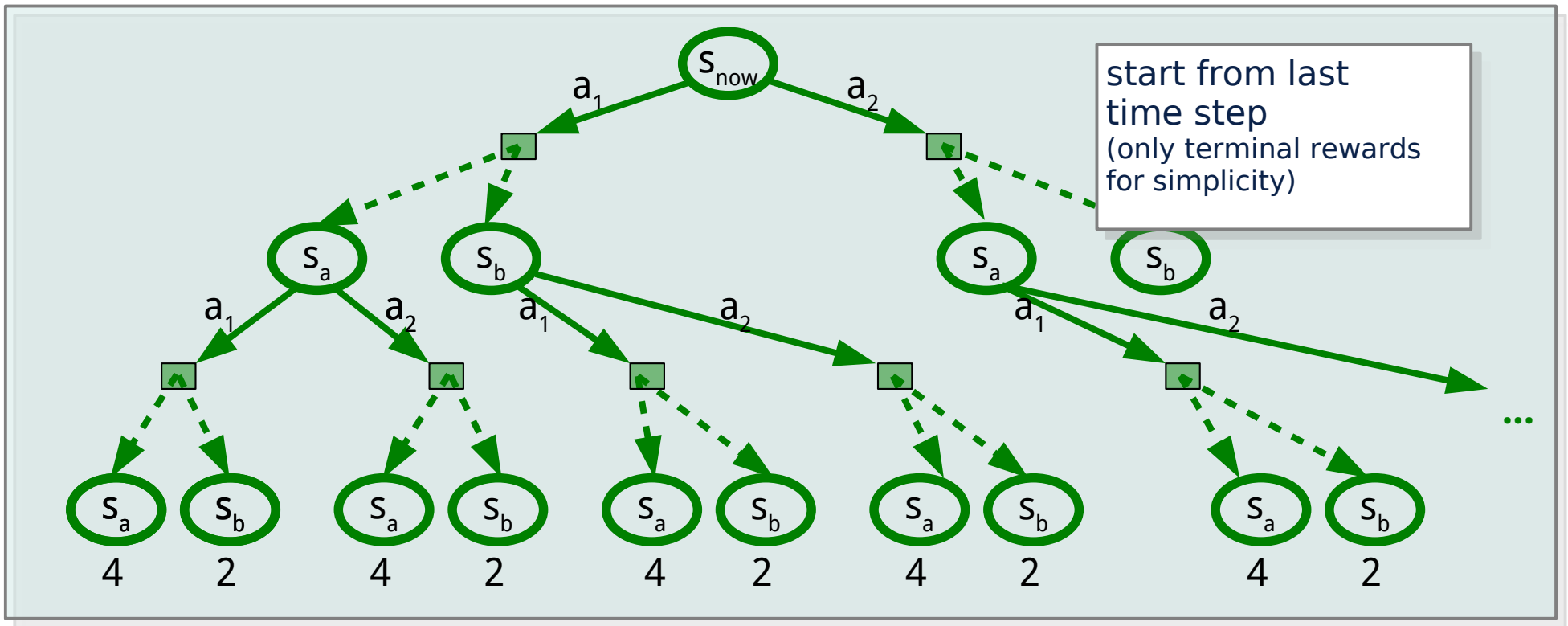
Dynamic Programming

- Construct a plan for *T* time steps into the future



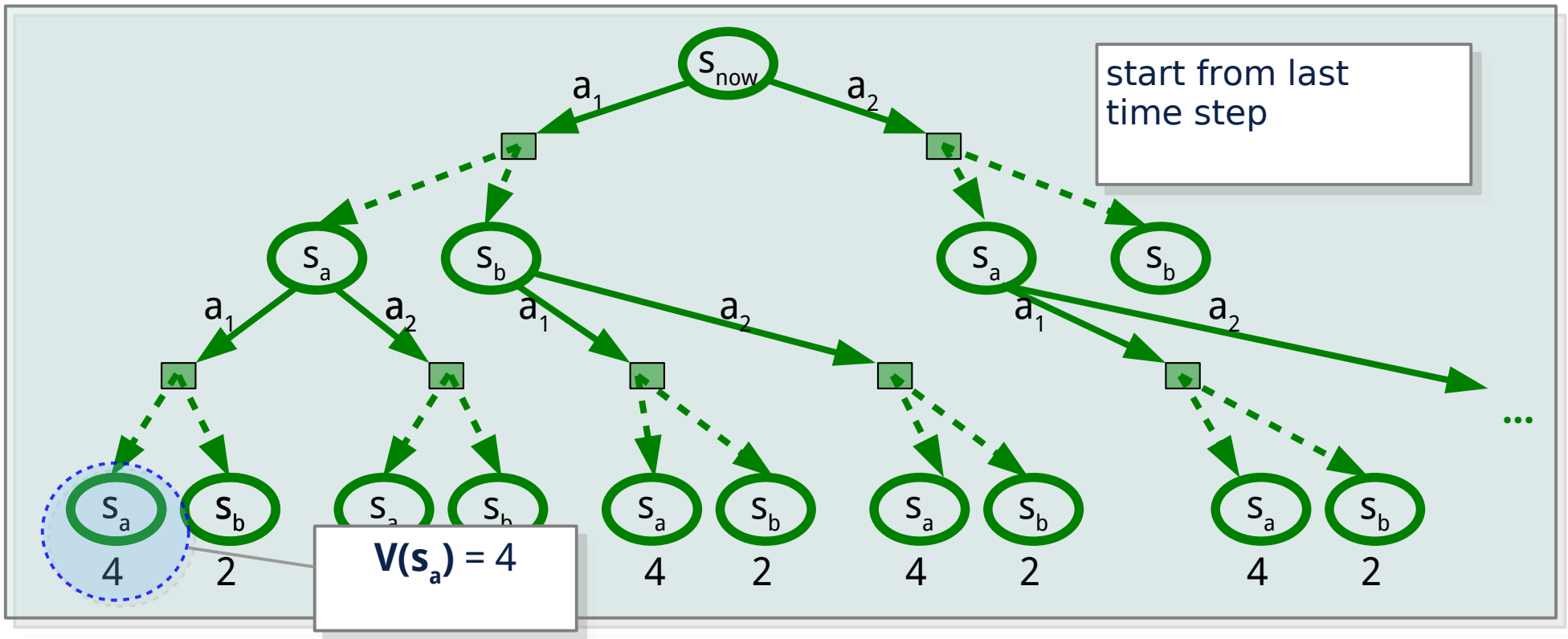
Dynamic Programming

- Construct a plan for *T* time steps into the future



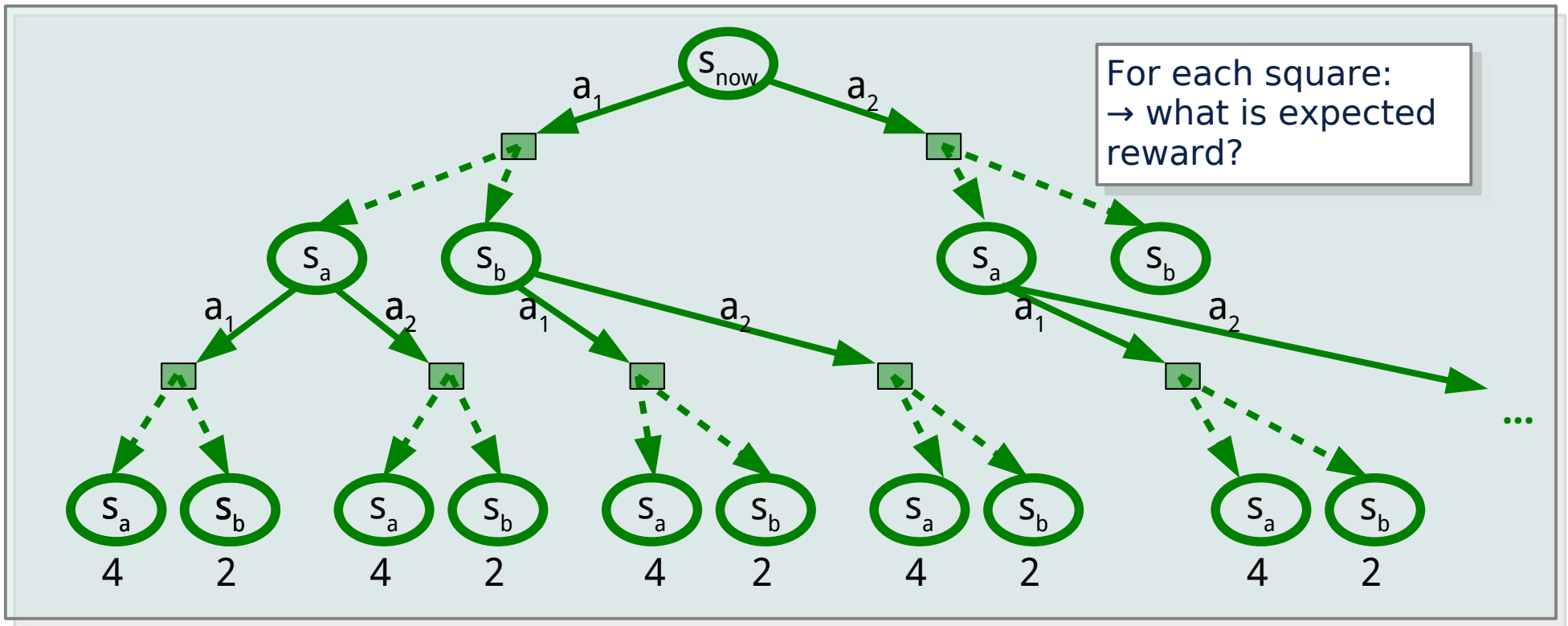
Dynamic Programming

- Construct a plan for *T* time steps into the future



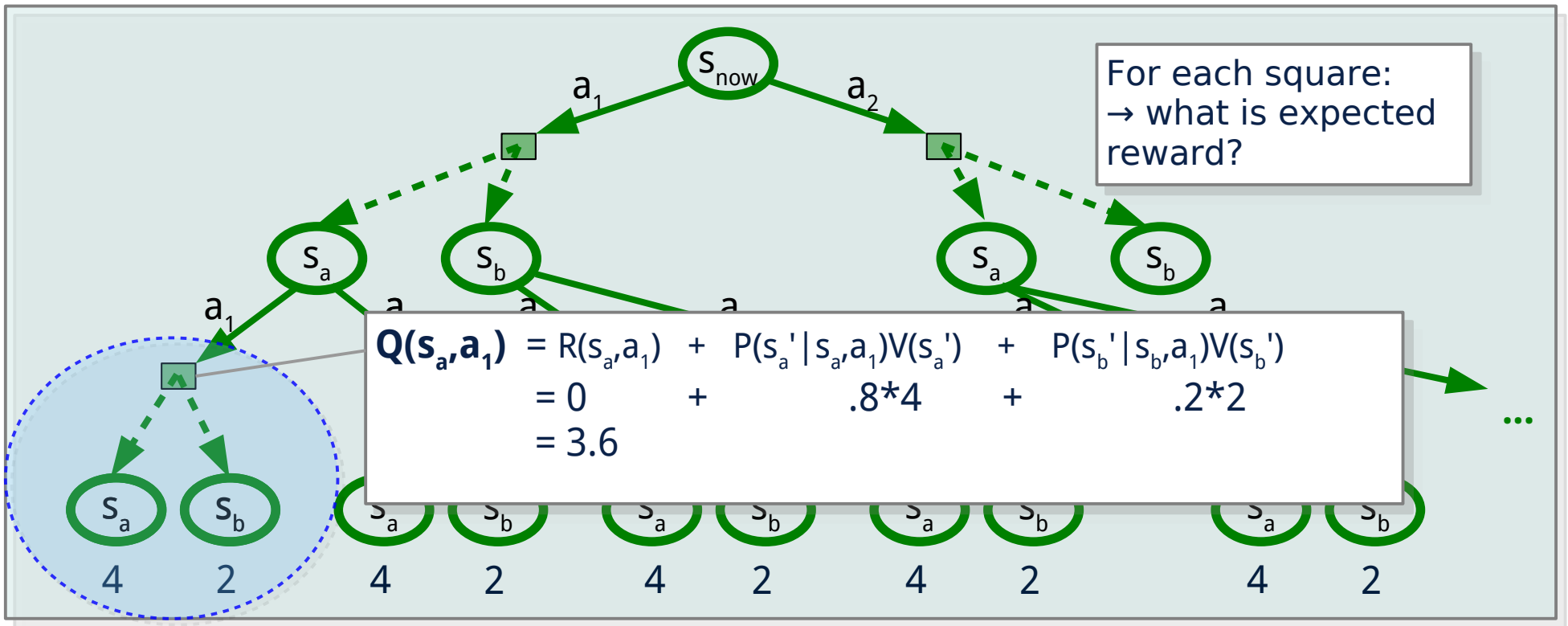
Dynamic Programming

- Construct a plan for *T* time steps into the future



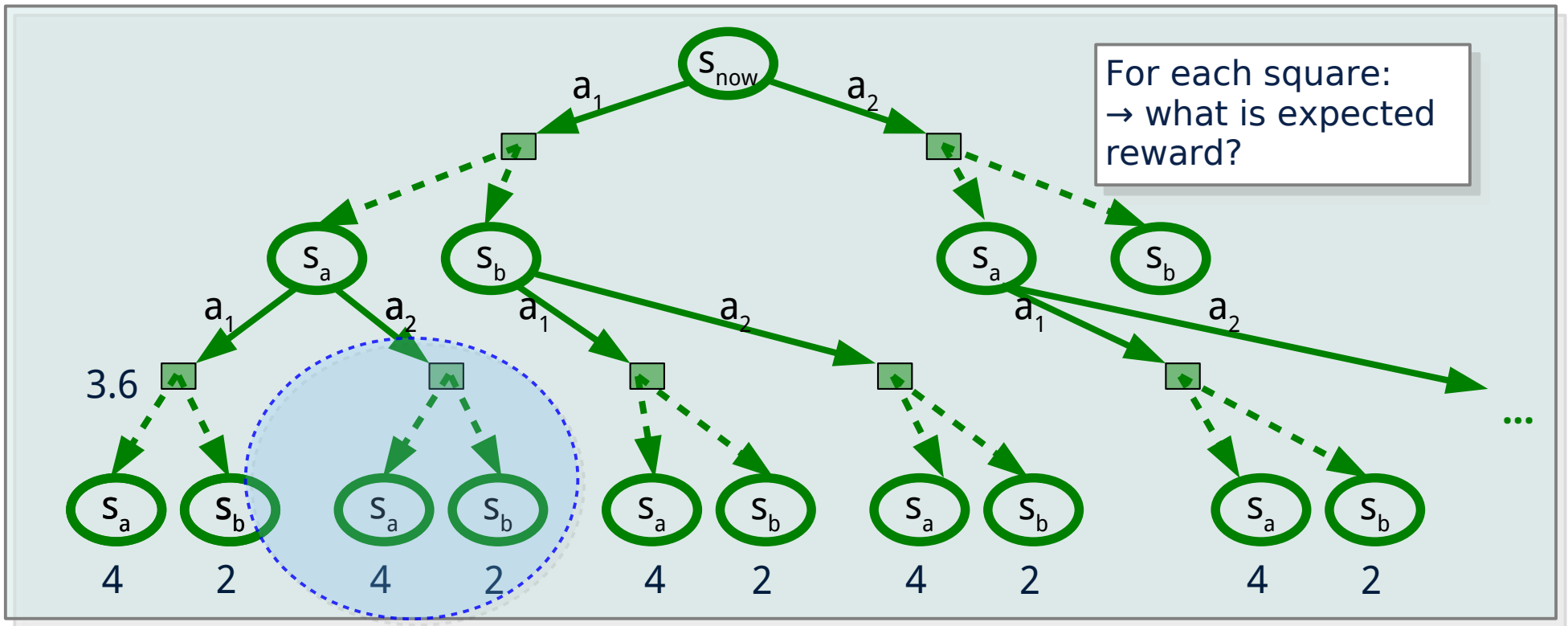
Dynamic Programming

- Construct a plan for *T* time steps into the future



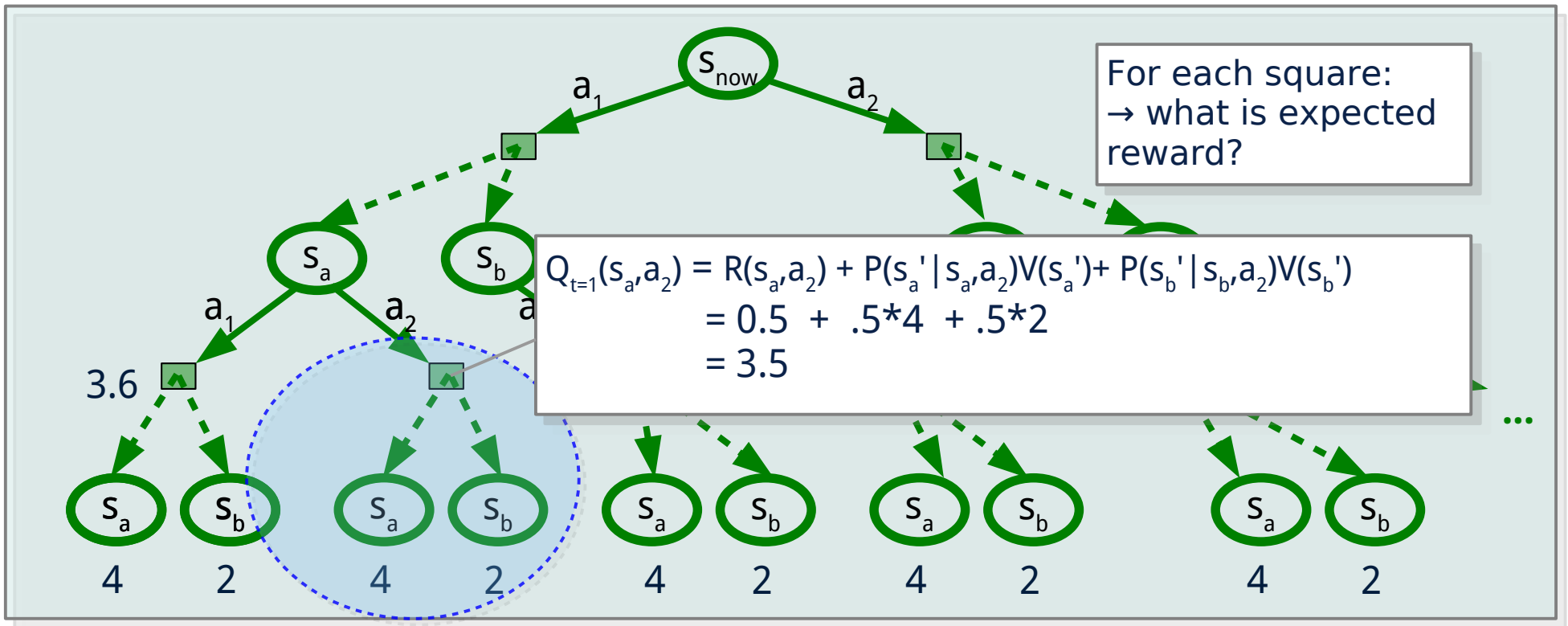
Dynamic Programming

- Construct a plan for *T* time steps into the future



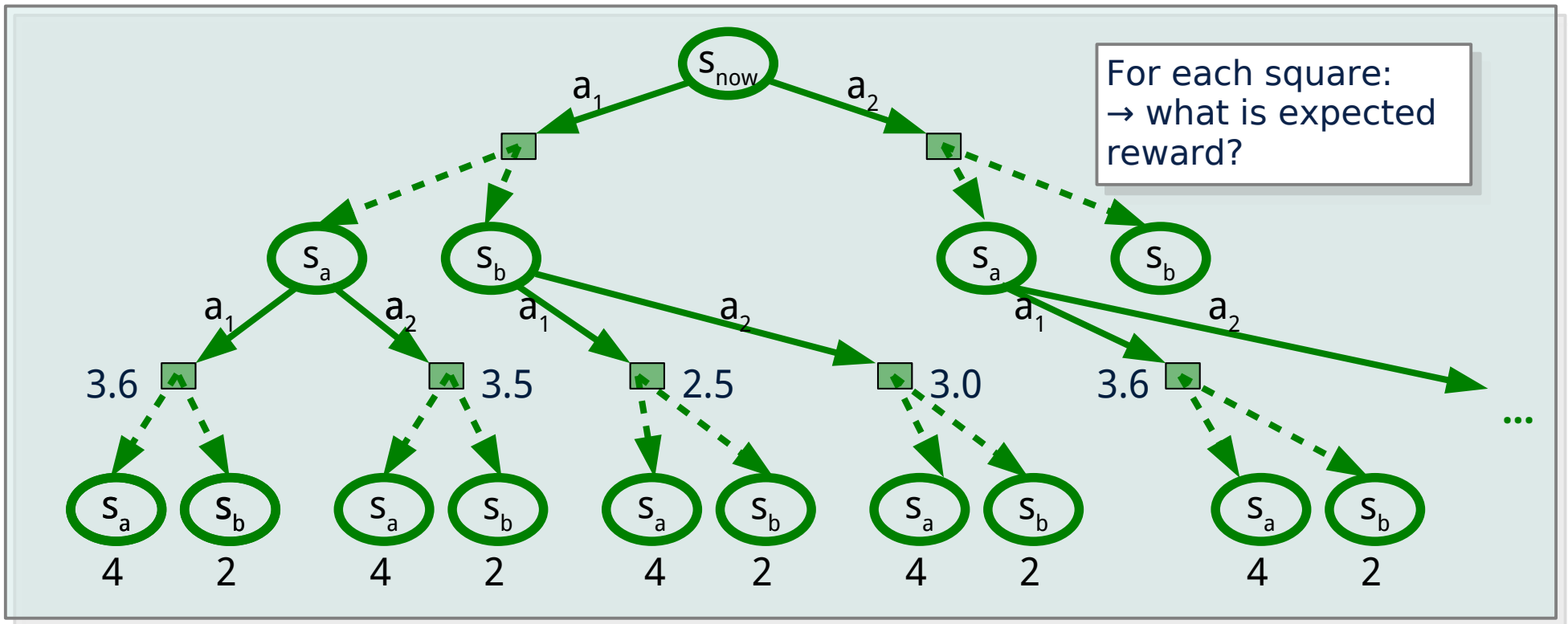
Dynamic Programming

- Construct a plan for *T* time steps into the future



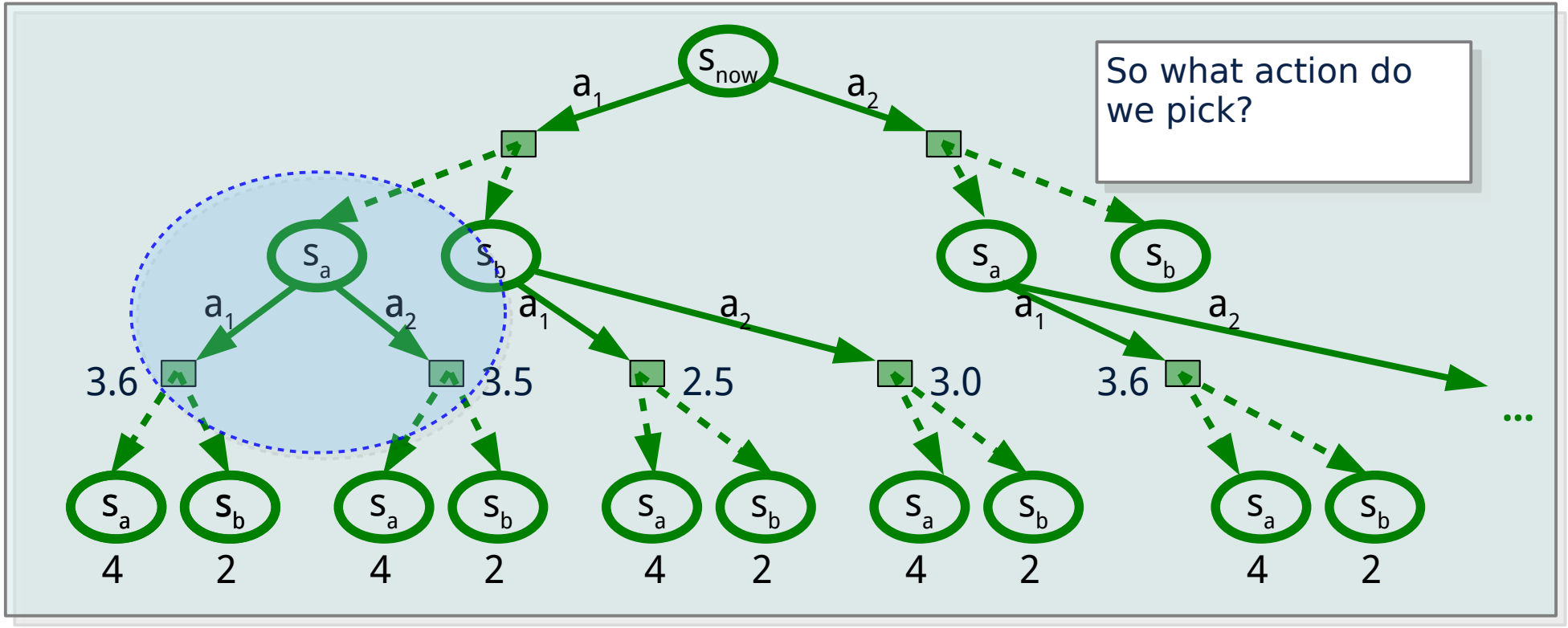
Dynamic Programming

- Construct a plan for *T* time steps into the future



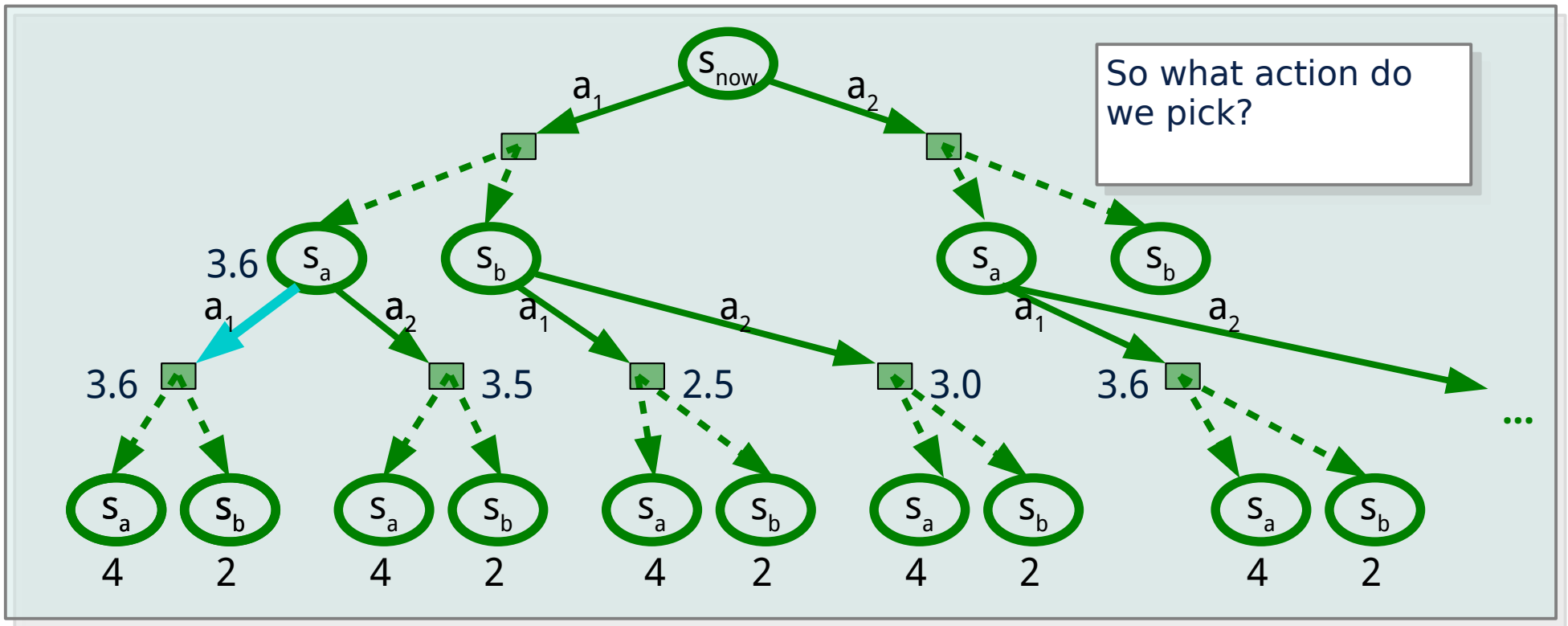
Dynamic Programming

- Construct a plan for *T* time steps into the future



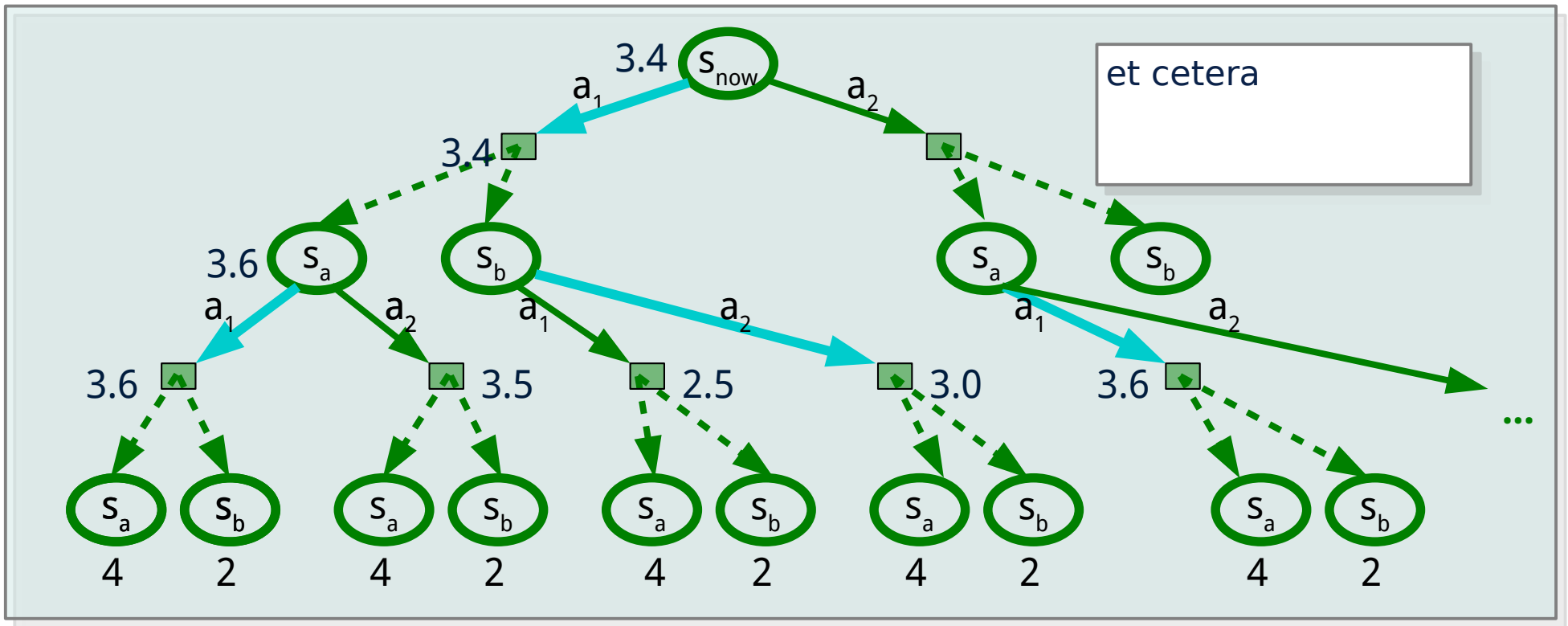
Dynamic Programming

- Construct a plan for *T* time steps into the future



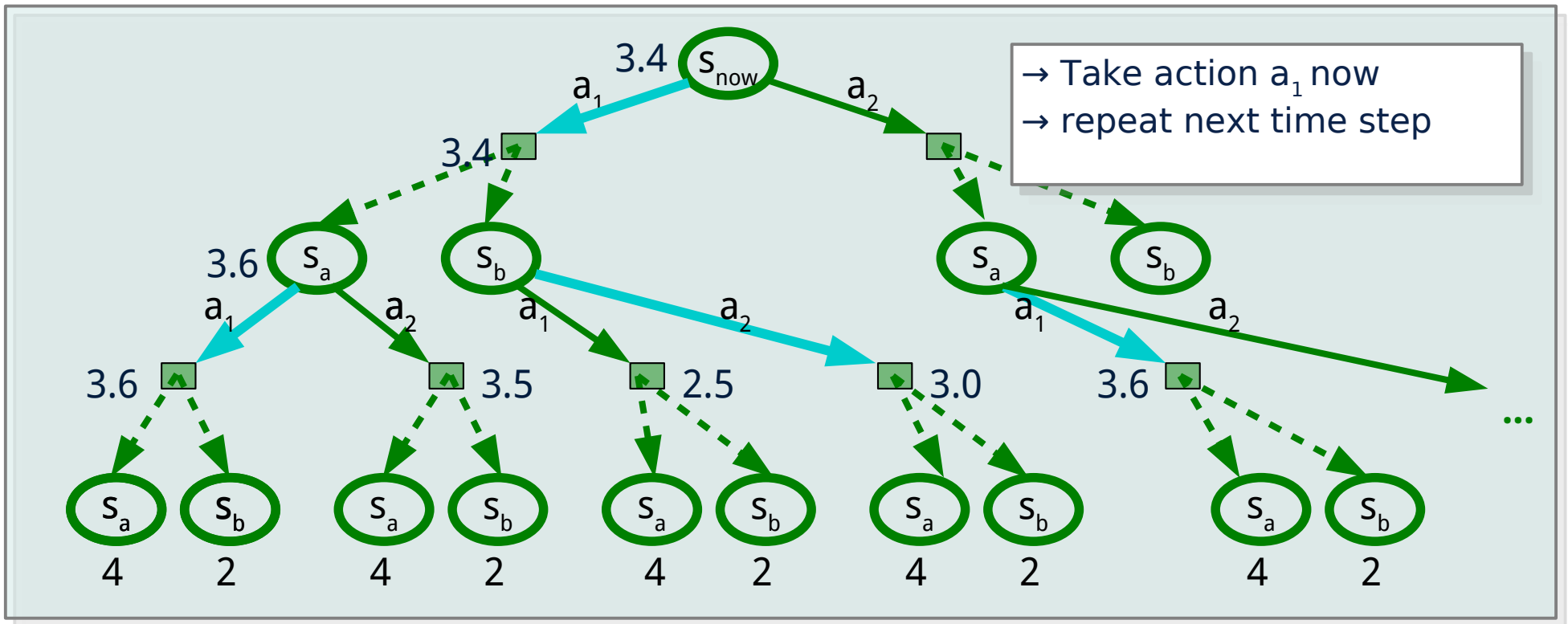
Dynamic Programming

- Construct a plan for *T* time steps into the future



Dynamic Programming

- Construct a plan for *T* time steps into the future

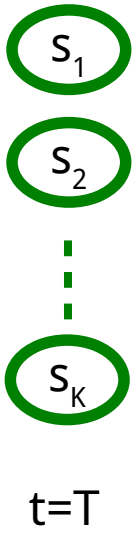


Dynamic Programming – limitations

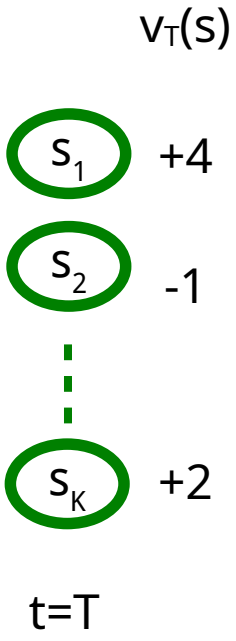
- Problem: trees get huge… → not practical
  - One solution: sampling (e.g., MCTS)

- But first… how about caching?

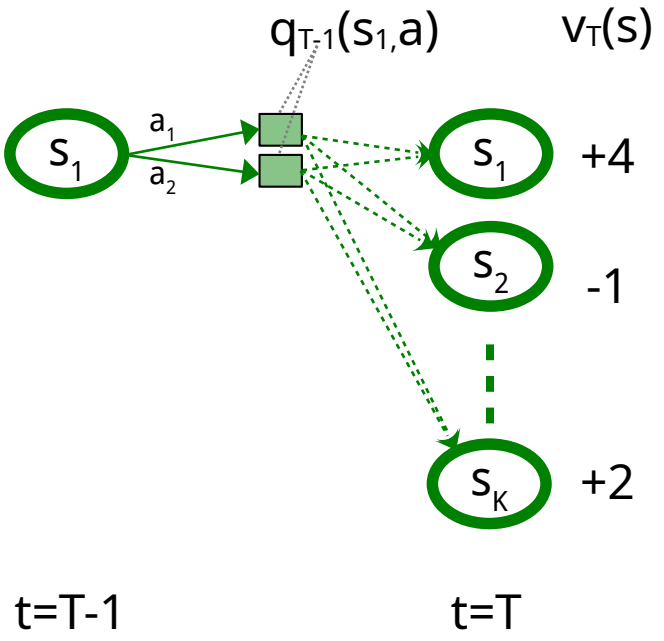
Avoiding trees completely. Use DAGs.



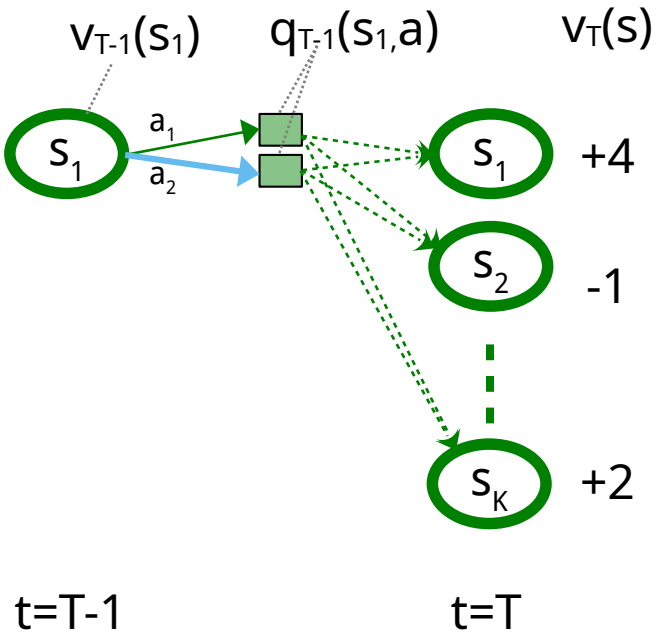
Avoiding trees completely. Use DAGs.



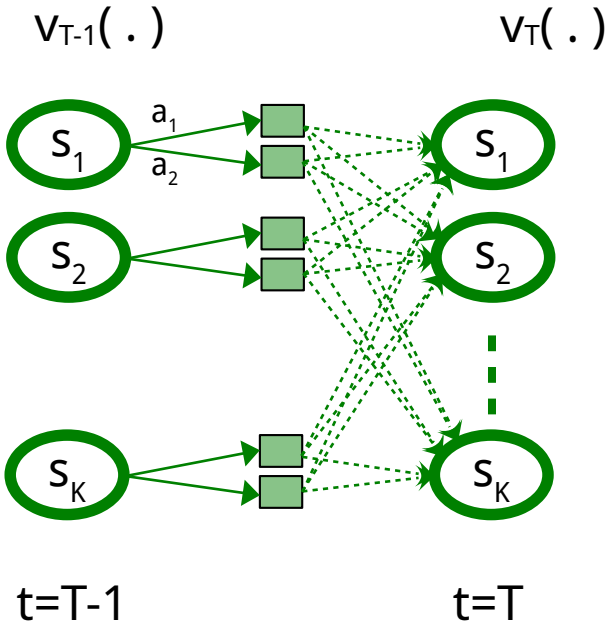
Avoiding trees completely. Use DAGs.



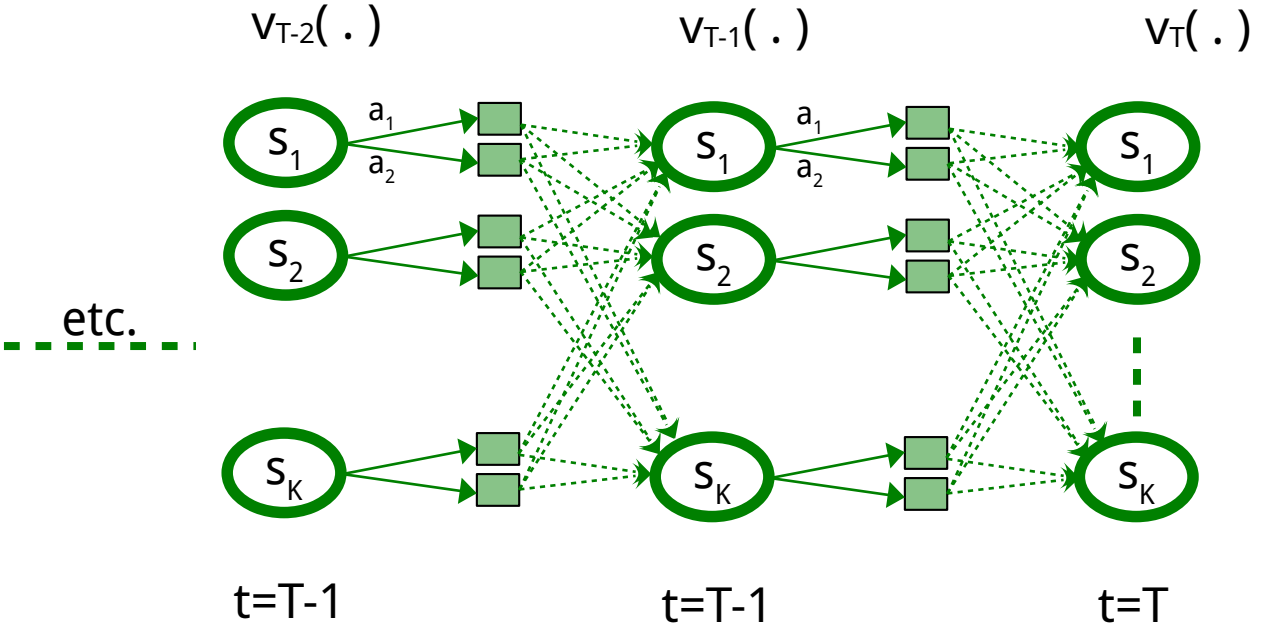
Avoiding trees completely. Use DAGs.



Avoiding trees completely. Use DAGs.



Avoiding trees completely. Use DAGs.



Summarizing so far...

- MDPs formalize decision making in stochastic environment

- For finite horizon it is easy to use dynamic programming: exploit temporal structure
  - DP on tree of trajectories, great given infinite computation
  - In general: use DAGs compute from t=T back to 0

Quiz

- Which are correct?
  - DP exploits temporal structure
  - DP on trees depends on the Markov property
  - DP on DAGs is linear in the horizon
  - DP is linear in the number of states

Next: Infinite horizon problems?

- Given an MDP <*S,A,T,R*> with **discounted returns**
  - **compute** a policy π that optimizes value $V(\pi)$

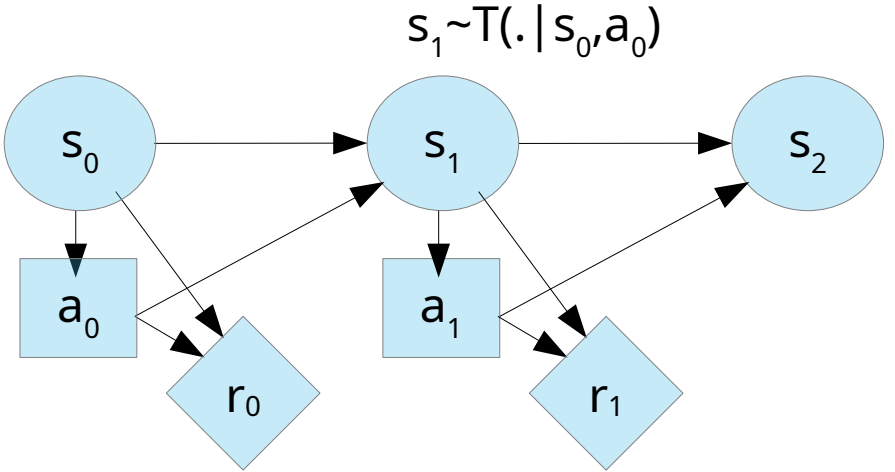
- (Generalized) policy iteration:
  1) pick an arbitrary π
  2) compute its value function $v_\pi(s)$
  3) use the value function to find a better policy π'
  4) $\pi \leftarrow \pi'$, goto 2).

Part 3: the value of a policy

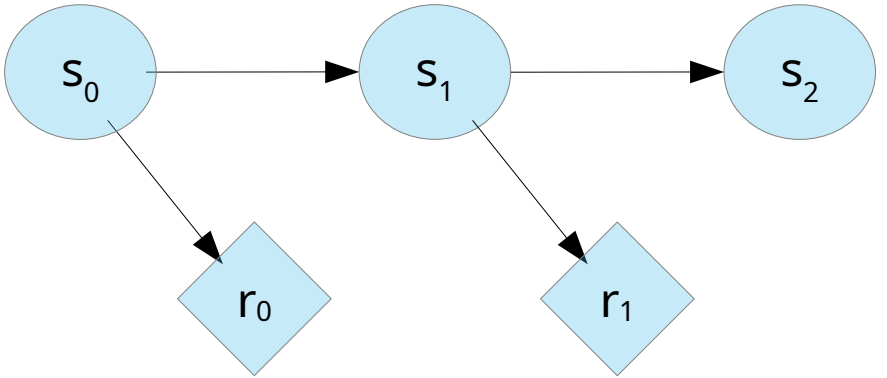
We will focus on a **fixed (arbitrary) policy π**

Fixing the Policy

- An MDP graphically



- Fixing the policy…
  - agent will always select $a=\pi(s)$
  - Induces a "Markov reward process"



So what is the value of a particular policy?

- How can we compute $v_\pi(s) = E_\pi [G_t \mid S_t=s]$?
- It satisfies the **Bellman equation**:

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma v_\pi(s')]$$

- How to solve? How to do "**policy evaluation**" ?
- Two options:
  - linear system of |S| equations, with |S| unknowns.
  - iterative policy evaluation

So what is the value of a particular policy?

- How can we compute
- It satisfies the **Bellm**

$$v_\pi(s) = \sum_a \pi$$

- How to solve? How to
- Two options:
  - linear system of |S| eq
  - iterative policy evaluat

Different forms of the Bellman equation.

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)[r + \gamma v_\pi(s')] \qquad \text{SBv2}$$

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s'} p(s'|s,a)[r(s,a,s') + \gamma v_\pi(s')] \qquad \text{r(s,a,s')}$$

$$v_\pi(s) = \sum_a \pi(a|s)[r(s,a) + \sum_{s'} p(s'|s,a) \gamma v_\pi(s')] \qquad \text{r(s,a)}$$

$$v_\pi(s) = r(s,\pi(s)) + \sum_{s'} p(s'|s,\pi(s)) \gamma v_\pi(s') \qquad \text{determ. policy}$$

all of these lead to the same result.

Iterative Policy Evaluation (IPE)

- Given that we know what the value function *is*… **how can we compute it?**
- Steps:
  - 1. fix the policy π to evaluate (let's assume deterministic)
  - 2. Take the Bellman equation…

$$v_\pi(s) = r(s, \pi(s)) + \sum_{s'} p(s'|s, \pi(s))\, \gamma\, v_\pi(s')$$

← use variant that is most convenient

… and turn it into an update equation:

$$v_{k+1}(s) := r(s, \pi(s)) + \sum_{s'} p(s'|s, \pi(s))\, \gamma\, v_k(s')$$

  - 3. Initialize, $v_0(s)=0$, for all s
  - 4. Apply the update equation, in iterations, to all state

So, we compute a sequence

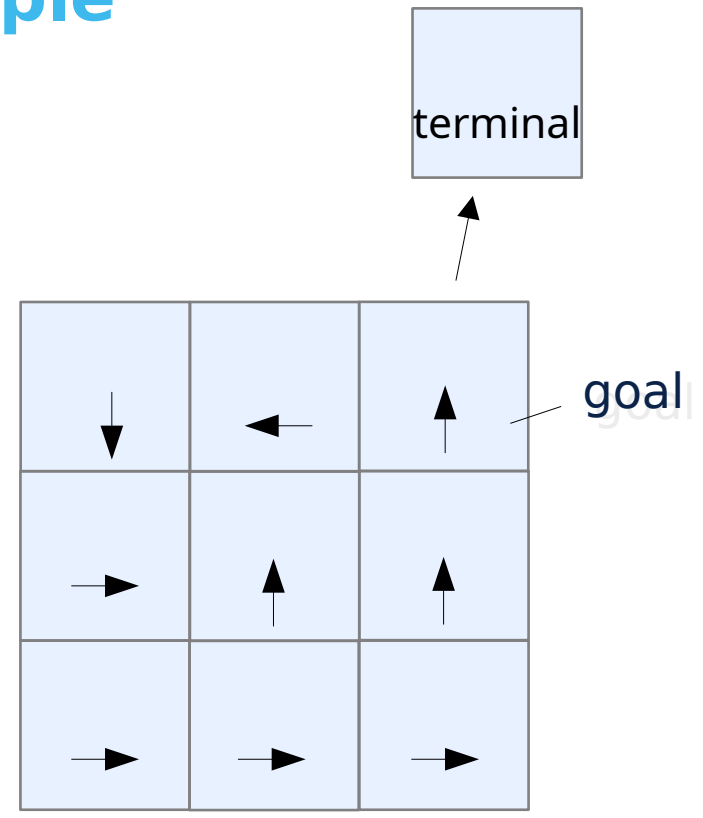
$(v_0, v_1, v_2, \ldots)$

of "k-steps-to-go" value functions

Policy Evaluation Example

- A little maze:
  - transitions:

    N,E,S,W, deterministic movements
    P(s' = terminal | s=goal, a=*)=1
  - rewards:
    R(s=Goal,a=*) = +10
    R(s=terminal, a=*) = 0
    R(s=*, a=*) = -1 (otherwise)
  - discount γ=0.9

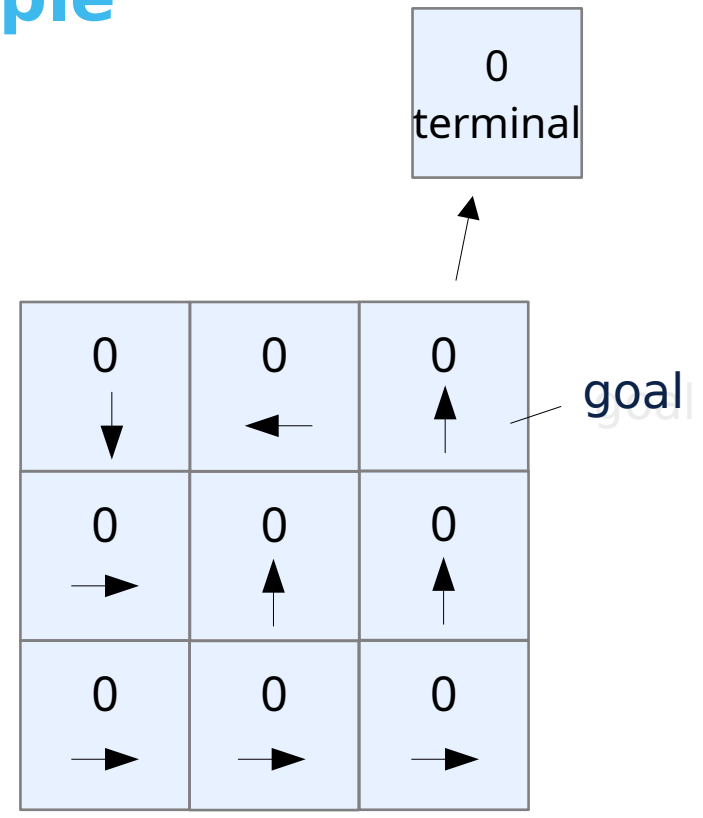


Policy Evaluation Example

- A little maze:
  - transitions:

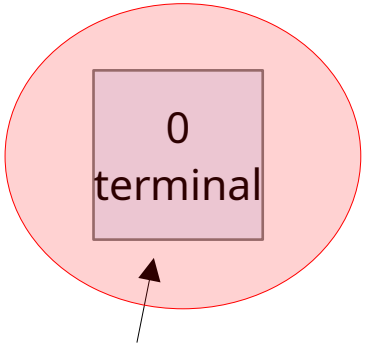
    N,E,S,W, deterministic movements
    P(s' = terminal | s=goal, a=*)=1
  - rewards:
    R(s=Goal,a=*) = +10
    R(s=terminal, a=*) = 0
    R(s=*, a=*) = -1 (otherwise)
  - discount γ=0.9

Initialize $v_0(s) = 0$



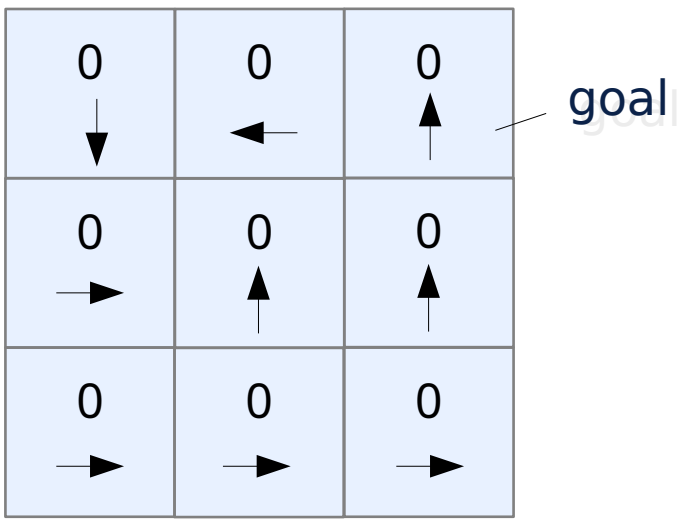
Policy Evaluation Example

v_1(terminal) = 0



- A little maze:
  - transitions:

    N,E,S,W, deterministic movements
    P(s' = terminal | s=goal, a=*)=1
  - rewards:
    R(s=Goal,a=*) = +10
    R(s=terminal, a=*) = 0
    R(s=*, a=*) = -1 (otherwise)
  - discount γ=0.9



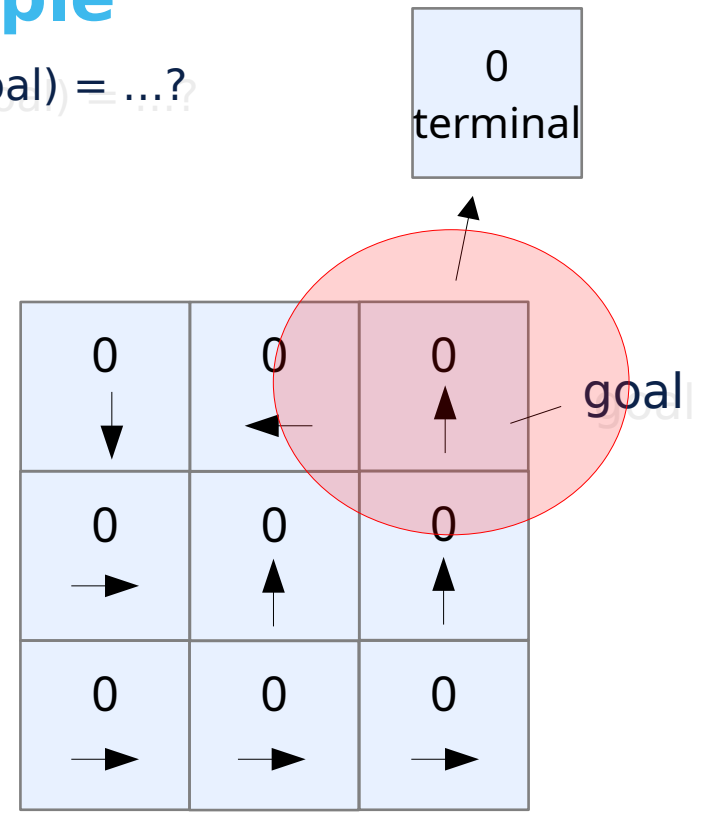
$$v_1(s) := r(s,\pi(s)) + \sum_{s'} p(s'|s,\pi(s))\,\gamma\, v_0(s')$$

Policy Evaluation Example

- A little maze:
  - transitions:

    N,E,S,W, deterministic movements
    P(s' = terminal | s=goal, a=*)=1
  - rewards:
    R(s=Goal,a=*) = +10
    R(s=terminal, a=*) = 0
    R(s=*, a=*) = -1 (otherwise)
  - discount γ=0.9

$v_1(s) := r(s, \pi(s)) + \sum_{s'} p(s'|s, \pi(s)) \gamma v_0(s')$

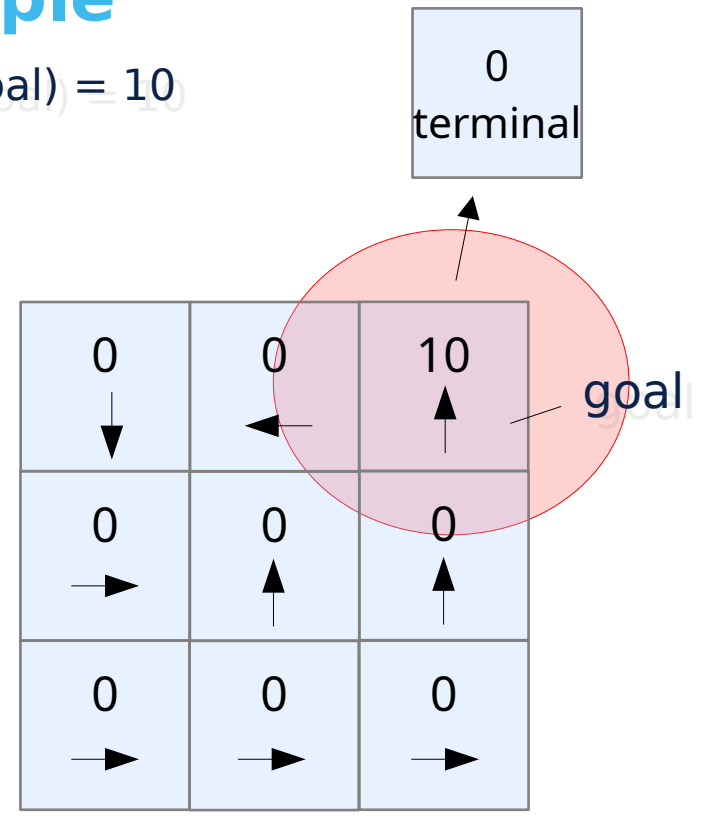


Policy Evaluation Example

- A little maze:
  - transitions:

    N,E,S,W, deterministic movements
    P(s' = terminal | s=goal, a=*)=1
  - rewards:
    R(s=Goal,a=*) = +10
    R(s=terminal, a=*) = 0
    R(s=*, a=*) = -1 (otherwise)
  - discount γ=0.9

$$v_1(s) := r(s, \pi(s)) + \sum_{s'} p(s'|s, \pi(s)) \gamma v_0(s')$$

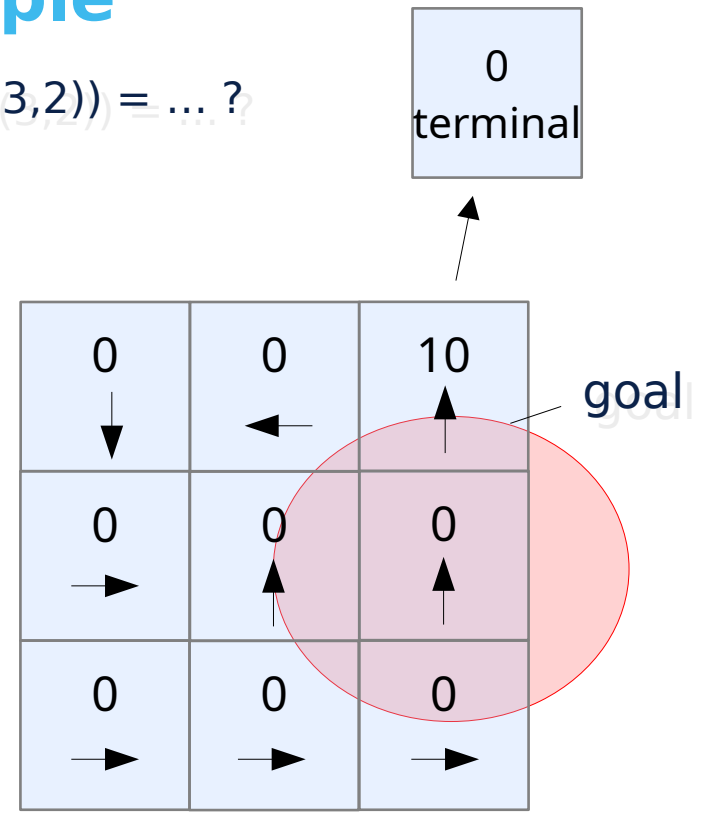


Policy Evaluation Example

$v_1((3,2)) = \ldots ?$

- A little maze:
  - transitions:

    N,E,S,W, deterministic movements
    P(s' = terminal | s=goal, a=*)=1
  - rewards:
    R(s=Goal,a=*) = +10
    R(s=terminal, a=*) = 0
    R(s=*, a=*) = -1 (otherwise)
  - discount γ=0.9



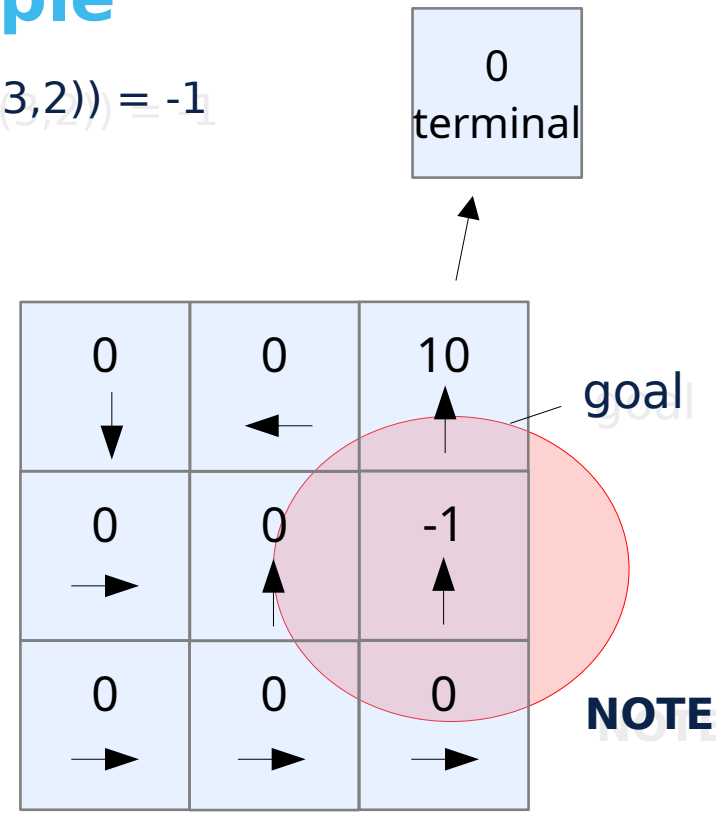
$$v_1(s) := r(s,\pi(s)) + \sum_{s'} p(s'|s,\pi(s))\,\gamma\, v_0(s')$$

Policy Evaluation Example

- A little maze:
  - transitions:

    N,E,S,W, deterministic movements
    P(s' = terminal | s=goal, a=*)=1
  - rewards:
    R(s=Goal,a=*) = +10
    R(s=terminal, a=*) = 0
    R(s=*, a=*) = -1 (otherwise)
  - discount γ=0.9

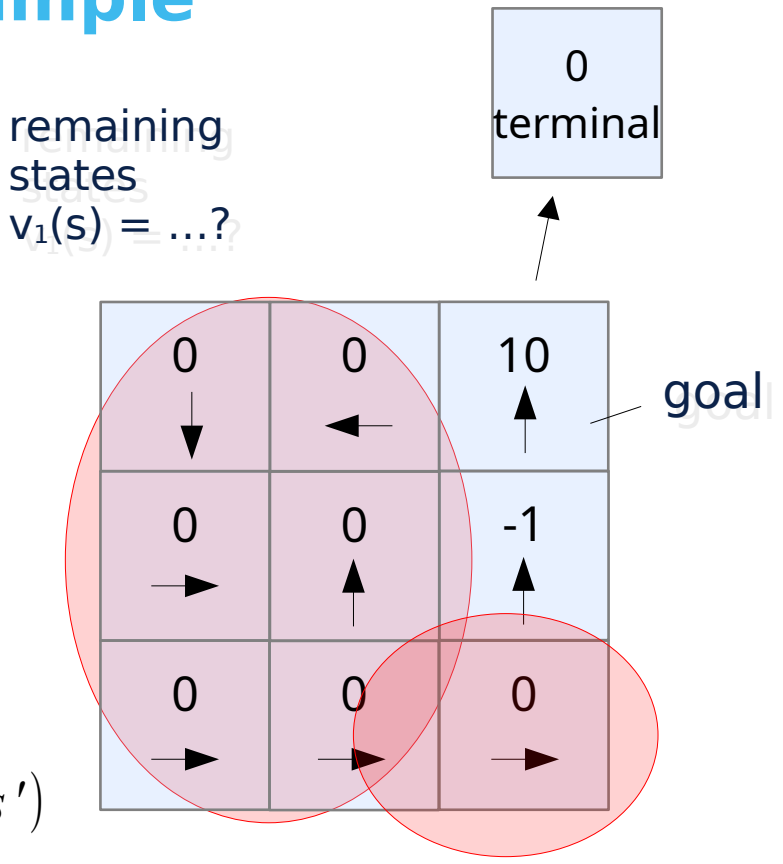
$$v_1(s) := r(s, \pi(s)) + \sum_{s'} p(s'|s, \pi(s)) \gamma v_0(s')$$



Policy Evaluation Example

- A little maze:
  - transitions:

    N,E,S,W, deterministic movements
    P(s' = terminal | s=goal, a=*)=1
  - rewards:
    R(s=Goal,a=*) = +10
    R(s=terminal, a=*) = 0
    R(s=*, a=*) = -1 (otherwise)
  - discount γ=0.9

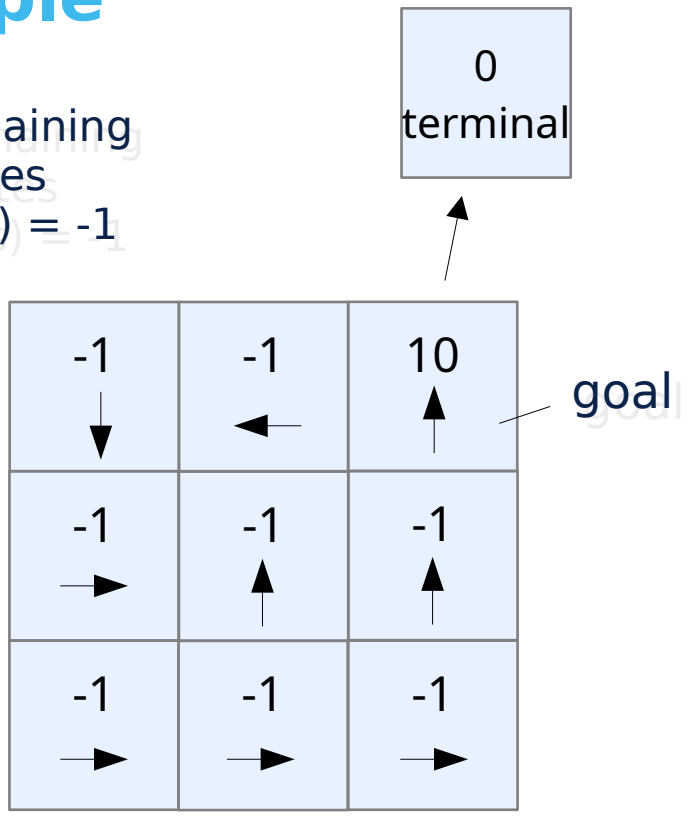


$$v_1(s) := r(s, \pi(s)) + \sum_{s'} p(s'|s, \pi(s)) \gamma v_0(s')$$

Policy Evaluation Example

- A little maze:
  - transitions:

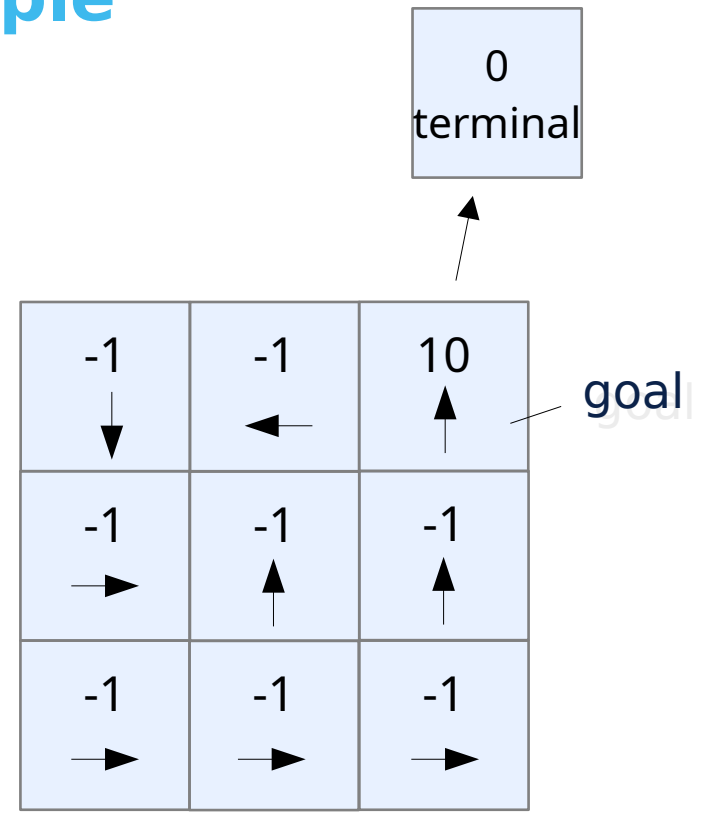
    N,E,S,W, deterministic movements
    P(s' = terminal | s=goal, a=*)=1
  - rewards:
    R(s=Goal,a=*) = +10
    R(s=terminal, a=*) = 0
    R(s=*, a=*) = -1 (otherwise)
  - discount γ=0.9



$$v_1(s) := r(s, \pi(s)) + \sum_{s'} p(s'|s, \pi(s)) \gamma v_0(s')$$

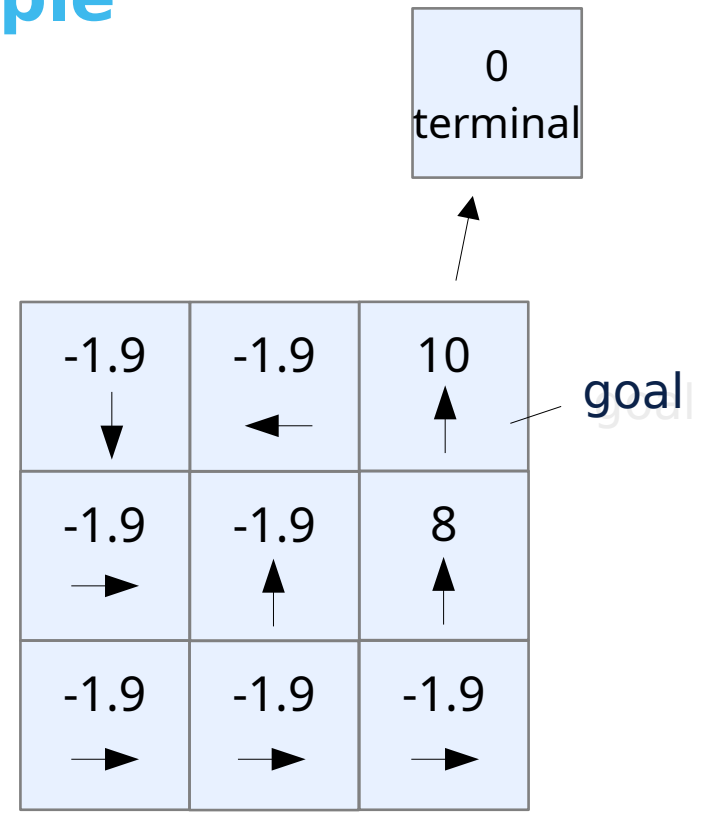
Policy Evaluation Example

Have now computed v_1, lets move to v_2...



Policy Evaluation Example

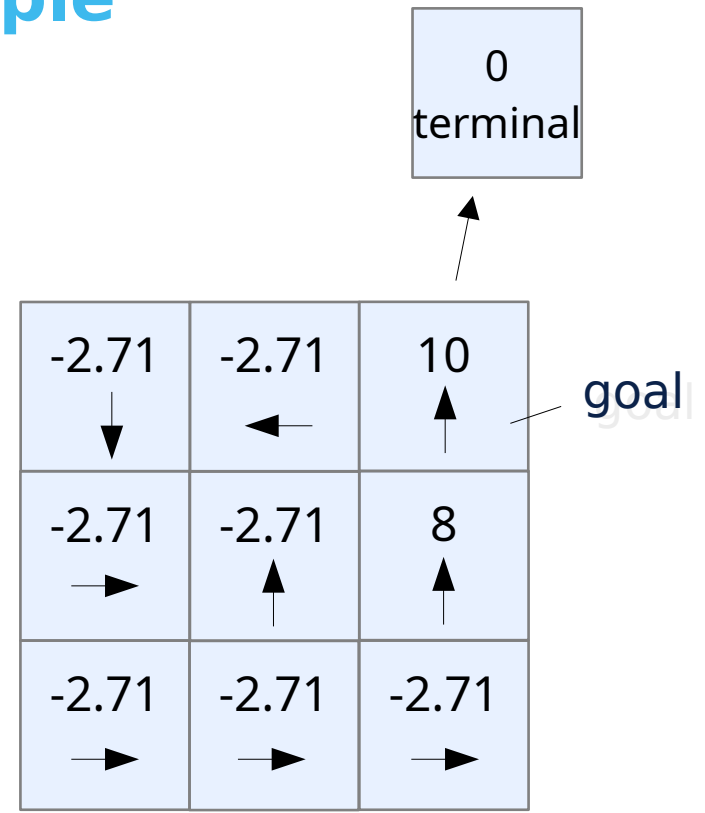
have now computed v_2
(the 2-steps-to-go value function)



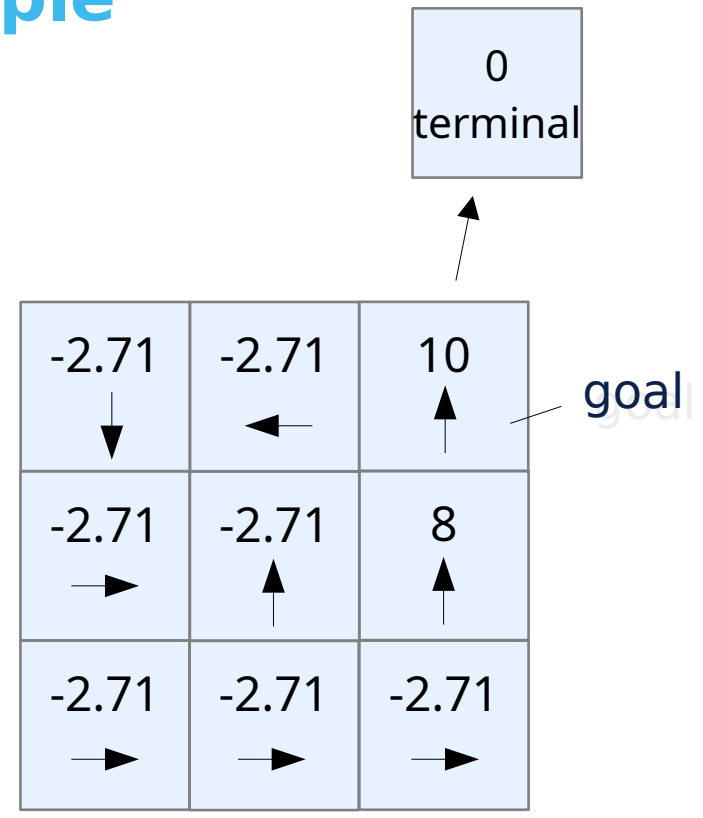
Policy Evaluation Example

What does this converge to…?

v_3 is this:



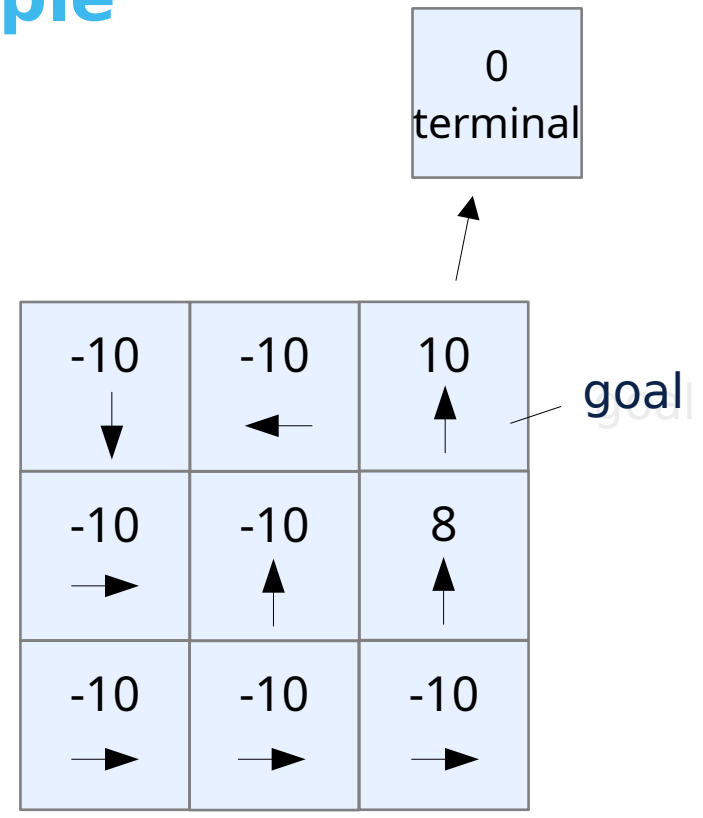
Policy Evaluation Example



What does this converge to…?

$$\sum_{t=0}^{\infty} \gamma^t r = \frac{r}{1-\gamma} = \frac{-1}{1-0.9} = -10$$

Policy Evaluation Example



$v_\pi = v_\infty$ is this:

Implementational issues

- As you saw, we did “parallel updates”
  - V_k is the ***k-steps-to-go* value function**
    (for following π)
  - but requires 2 arrays to implement

- Can also implement in 1 array
  - do updates “in place”
  - also converges, **and can be faster!**
  - but V_k will no longer correspondence to k-steps-to-go value

Part 4: Computing an Optimal Policy

(Generalized) policy iteration

to compute an optimal policy π^*

Policy Iteration

- 2 steps:
  - policy evaluation: compute $v_\pi(s)$
  - policy improvement: update $\pi \rightarrow \pi'$
- By alternating these, converge to optimal policy π^*

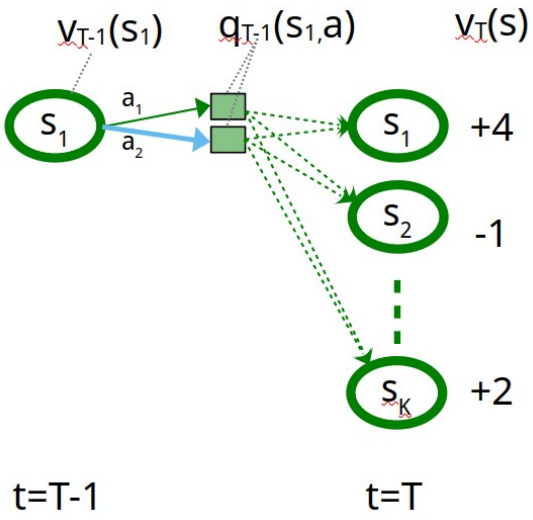
Policy Improvement – 1

- When we have computed $v_\pi(s)$…
  …we want to use that to **improve** the policy!

- Let's define the **action-value function**:

$$q_\pi(s,a) = \sum_{s',r} p(s',r|s,a)[r + \gamma v_\pi(s')]$$

  - expected value when selecting *a* at *s*, and following π afterwards



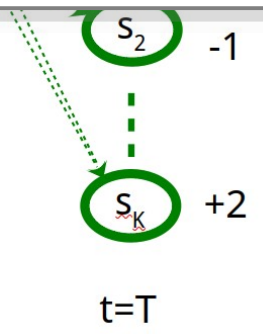
Policy Improven

different forms of v_π → different forms of q_π !

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)[r + \gamma v_\pi(s')]$$

- When we have comp
  ...we want to use tha
- Let's define the **actio**

$$q_\pi(s,a) = \sum_{s',r} p(s',r|s,a)[r + \gamma v_\pi(s')]$$



- expected value when selecting *a* at *s*, and following π afterwards

Policy Improvement – 2

- Now, given $q_\pi(s,a)$, we can improve the policy…

  …by being **greedy**:

  forall s: $\pi'(s) \leftarrow \max_a q_\pi(s,a)$

- Then repeat:
  - policy evaluation
  - policy improvement

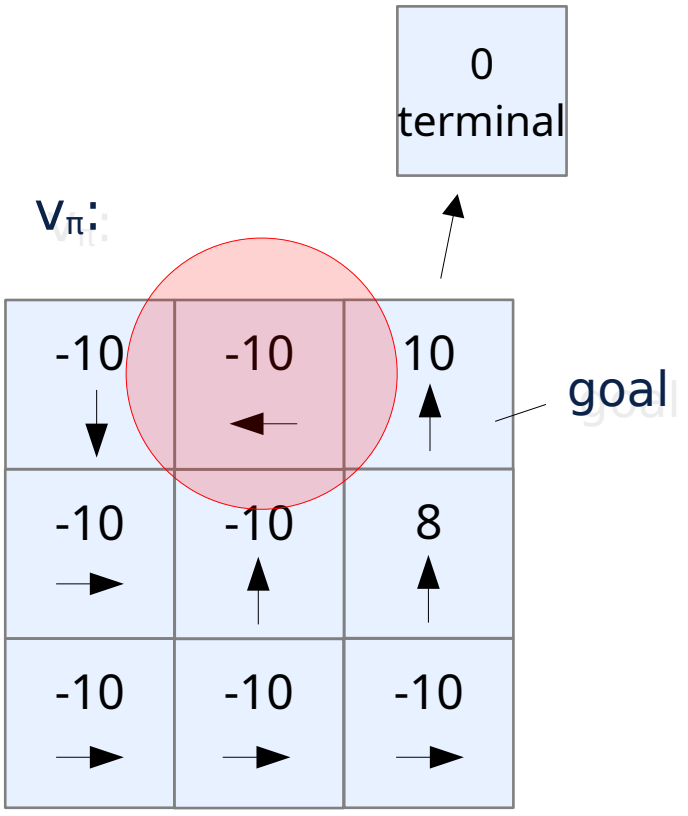
  → called **policy iteration**

Policy Improvement Example

- Continuing with our little maze:
  - convenient Q-value function:

$$q_\pi(s,a) = r(s,a) + \gamma \sum_{s'} p(s'|s,a) v_\pi(s')$$

```
q(s(2,1), N) = -1 +  .9 * -10  = -10
q(s(2,1), E) = -1 +  .9 * +10  =  +8
q(s(2,1), S) = -1 +  .9 * -10  = -10
q(s(2,1), W) = -1 +  .9 * -10  = -10
```

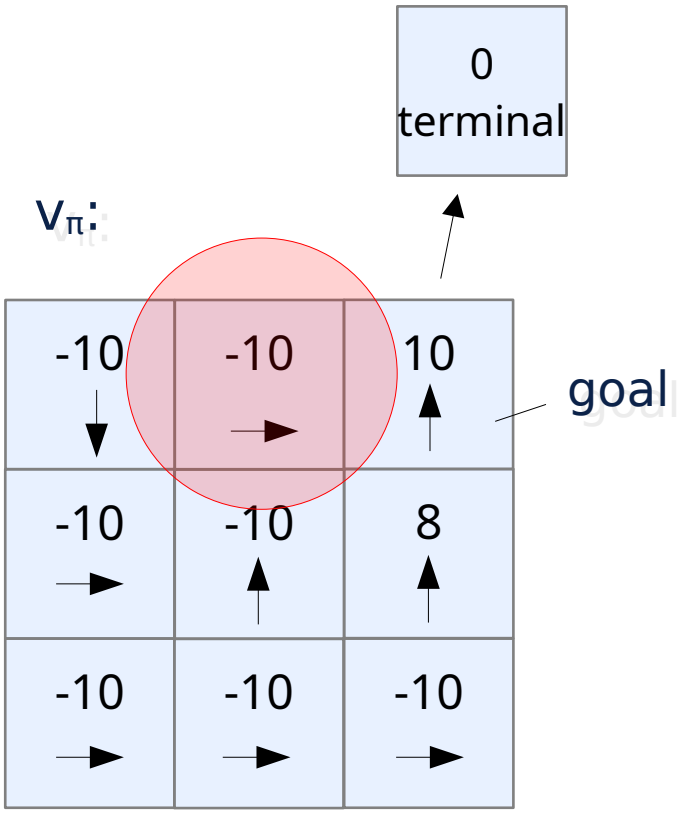


Policy Improvement Example

- Continuing with our little maze:
  - convenient Q-value function:

$$q_\pi(s,a) = r(s,a) + \gamma \sum_{s'} p(s'|s,a)\, v_\pi(s')$$

```
q(s(2,1), N) = -1 +  .9 * -10  = -10
q(s(2,1), E) = -1 +  .9 * +10  =  +8
q(s(2,1), S) = -1 +  .9 * -10  = -10
q(s(2,1), W) = -1 +  .9 * -10  = -10
```

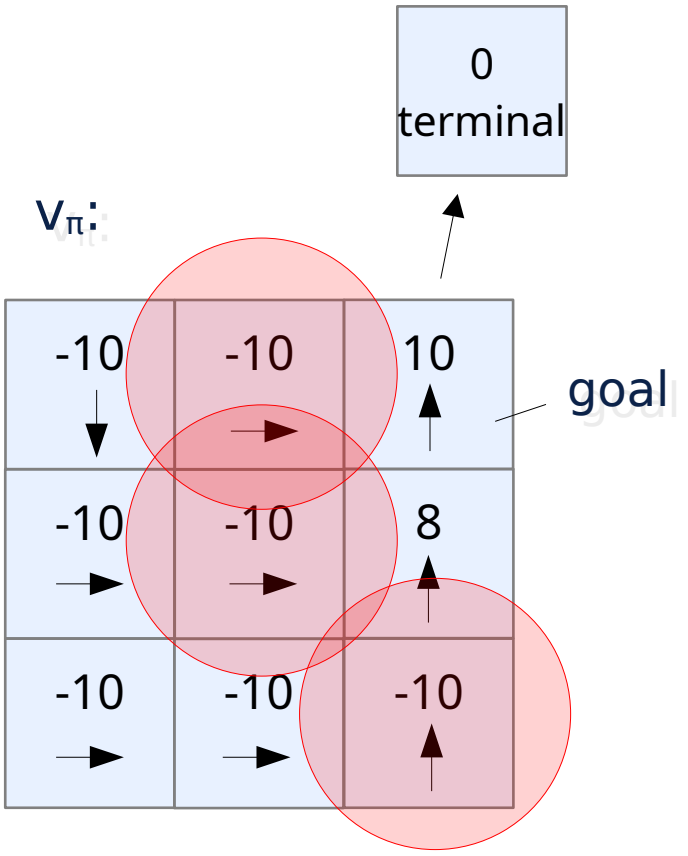


Policy Improvement Example

- Continuing with our little maze:
  - convenient Q-value function:

$$q_\pi(s,a) = r(s,a) + \gamma \sum_{s'} p(s'|s,a) v_\pi(s')$$

Other updates...

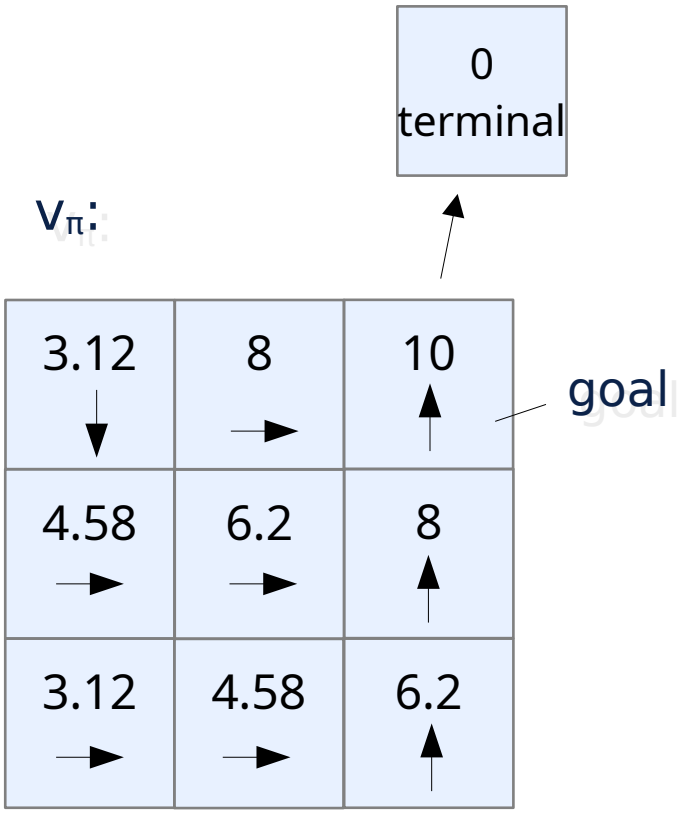


Policy Improvement Example

- Continuing with our little maze:
  - convenient Q-value function:

$$q_\pi(s,a) = r(s,a) + \gamma \sum_{s'} p(s'|s,a) v_\pi(s')$$

then: compute value v_π of this new policy, etc.



Optimal policies & (action-) value functions

- So does this converge…?

Optimal policies & (action-) value functions

- So does this converge…?
- Yes! Converges to unique optimal value functions
  - given by **Bellman optimality equations**:

$$v_*(s) = max_a q_*(s,a)$$

$$q_*(s,a) = \sum_{s',r} p(s',r|s,a)[r + \gamma v_*(s')]$$

- There can be multiple optimal policies
  - they share the same optimal value function

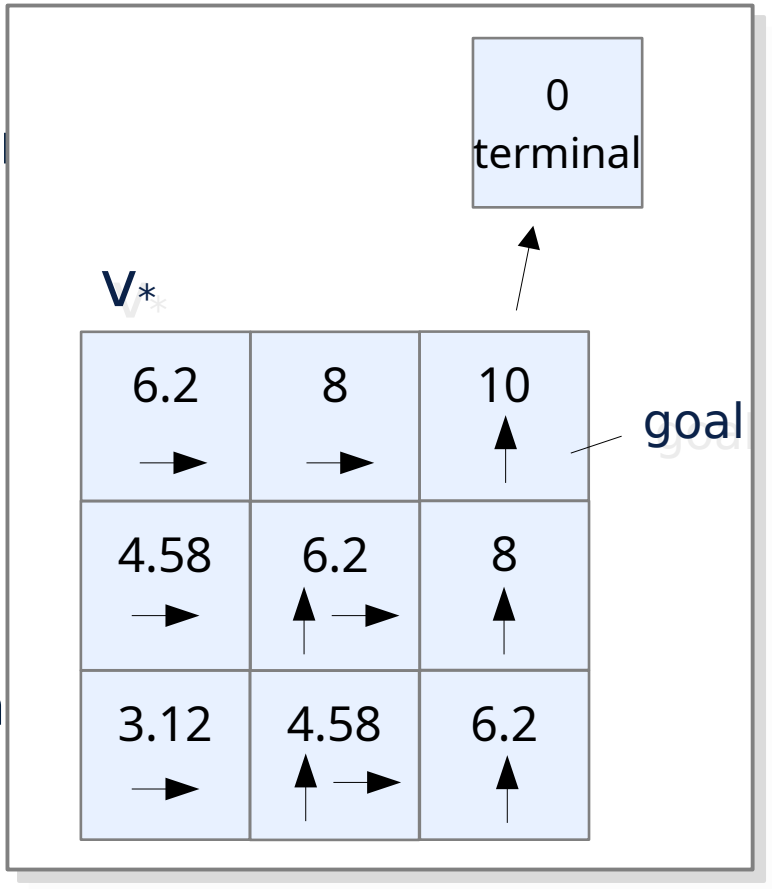
Optimal policies & (action-) value functions

- So does this converge…?
- Yes! Converges to unique optimal value fu
  - given by **Bellman optimality equations**:

$$v_*(s) = max_a q_*(s,a)$$

$$q_*(s,a) = \sum_{s',r} p(s',r|s,a)[r + \gamma v_*(s')]$$

- There can be multiple optimal policies
  - they share the same optimal value function



Generalized Policy Iteration & Value Iteration

- Is it needed to run policy evaluation until convergence…?
  → No..! Can do a few iterations of IPE, and then do policy improvement. Still works.
- Leads to "**generalized policy iteration**": can approximate both policy evaluation and improvement

- In the extreme: **value iteration**
  - does only 1 IPE iteration
  - It combines IPE and policy improvement in a single update rule:

$$v(s) \leftarrow max_a \{ \sum_{s',r} p(s'|s,a)[r(s,a,s') + \gamma v(s')] \}$$

  - repeatedly sweep through state space, until convergence

Summary so far

- Many problems are stochastic… → need feedback plans
- MDPs model these problems
  - key component: Markov assumption

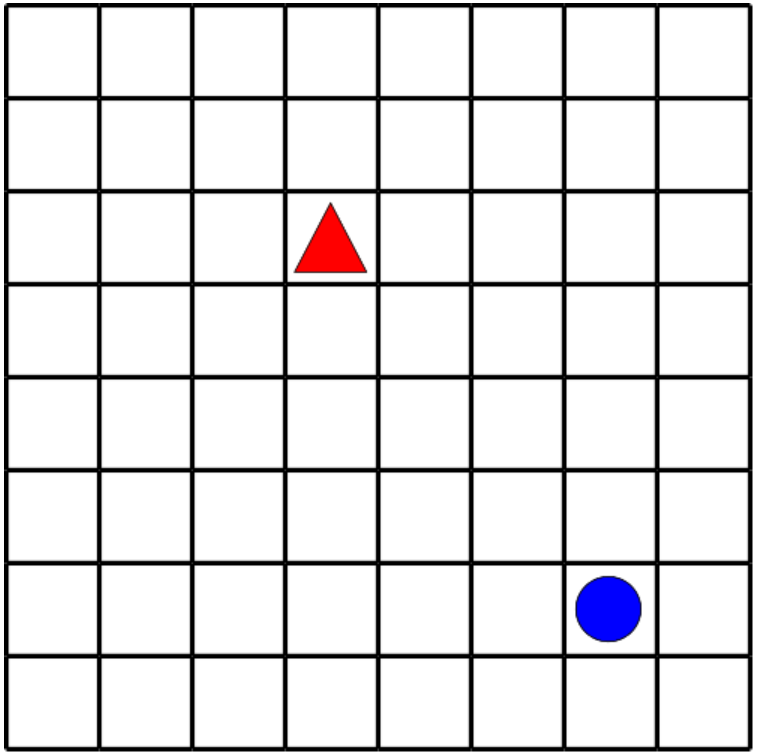
- Planning aka dynamic programming:
  - Finite horizon: DP over a tree / DAG
  - Infinite horizon: (generalized) policy iteration
    - policy evaluation ↔ policy improvement

Part 5: What if we cannot observe the state?

Images by the U.S. National Park Service in public domain

Example: Predator-prey MDP

- We are the blue round predator
  - A={left, right, up, down}
- Prey moving stochastically
  - independent of us. (why important?)
- States…?
  - relative positions.
  - E.g.: current state s=(-3,4)
  - (assumes “wrap around”)

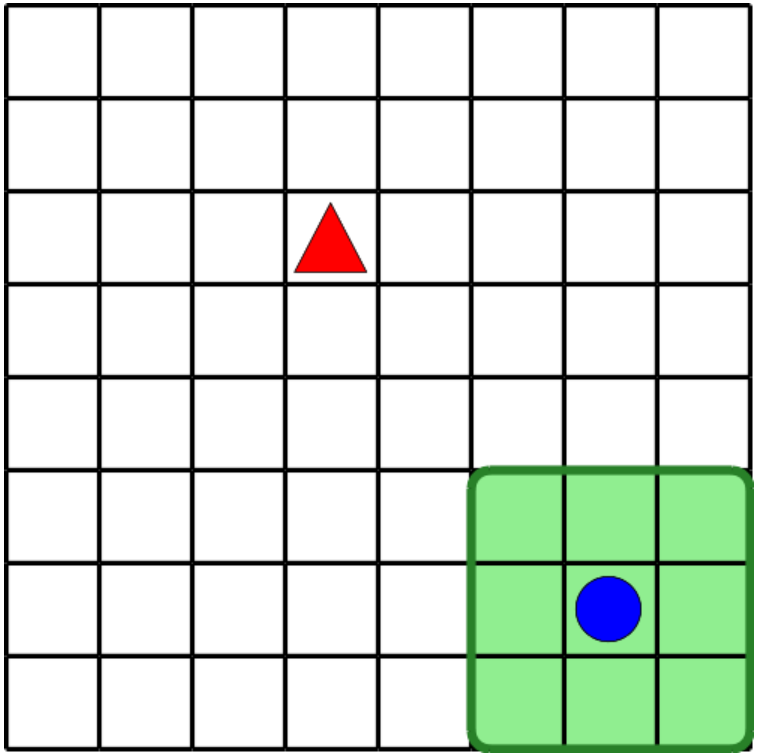


Example: Predator-prey MDP

- But now we have limited range…
- State?
  - still s=(-3,4)

- But what does the agent observe?
- current observation…?

o=’none’

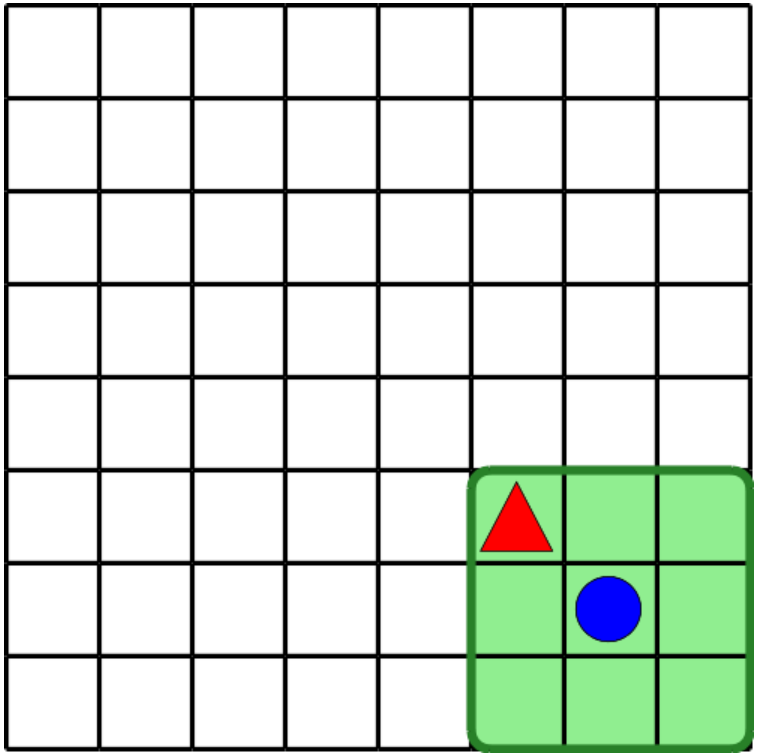


Example: Predator-prey MDP

- But now we have limited range…
- State?
  - still s=(-3,4)

- But what does the agent observe?
- current observation…?

o=(-1,1)



Types of Partial Observability

- Noise
  - Sensors have measurement errors.
  - Sensor (or other part of the agent) can fail.

- Perceptual aliasing
  - When multiple situations can't be discriminated.
  - I.e., multiple states give the same observation.
    - e.g. what is behind a wall?

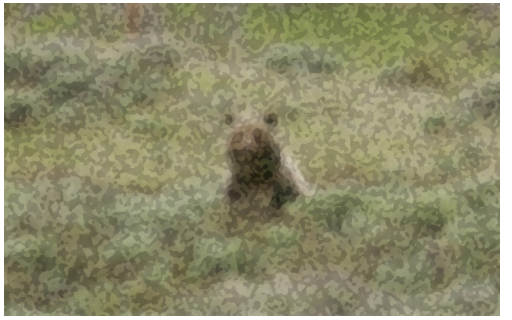
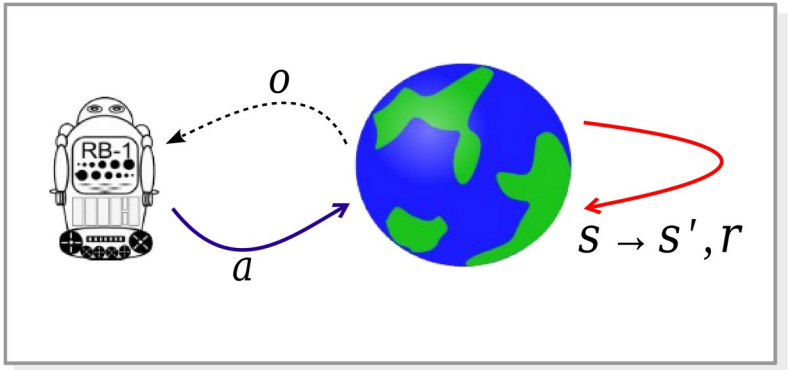


Image by the U.S. National Park Service in public domain.
Illustration from OpenClipart is licensed under CC0 1.0.

Formal model: POMDP

- A Markov decision process (MDP) $M=<S,A,P_T,R>$
  - S – set of world states s
  - A – set of actions a
  - P_T – transition function, P(s'|s,a)
  - R – reward function, R(s,a)



- A partially observable MDP (POMDP), $M=<S,A,O,P_T,P_O,R>$
  - O – set of observations o
  - P_O – observation function, P(o|a, s')

- Optimality criterion typically expected (discounted) return

Policies in P.O. environments

- Now given that the agent only gets some observations…
  - what policy should he follow?
  - How does such a policy even look like?
- No Markovian signal (i.e. the state) directly available to the agent…
  - → In general: should use all information!
  - → i.e. **full history of actions and observations** $h_t=(a_0,o_1,a_1,\ldots,a_{t-1},o_t)$
  - → deterministic policies: observation histori

Why not just using observations?

- Could be very bad!
  - randomization can help
  - and history can help more!

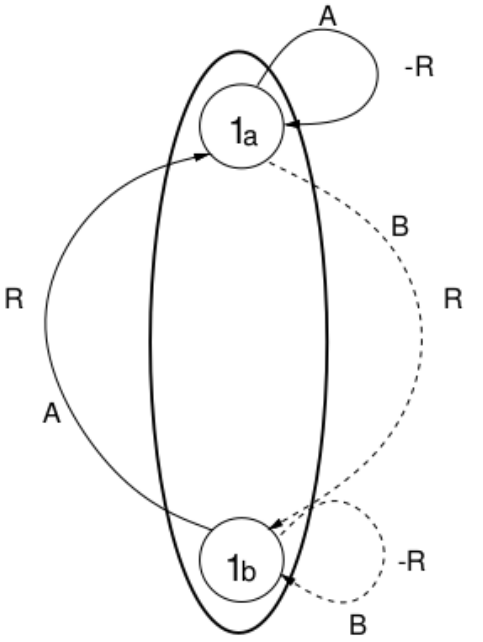


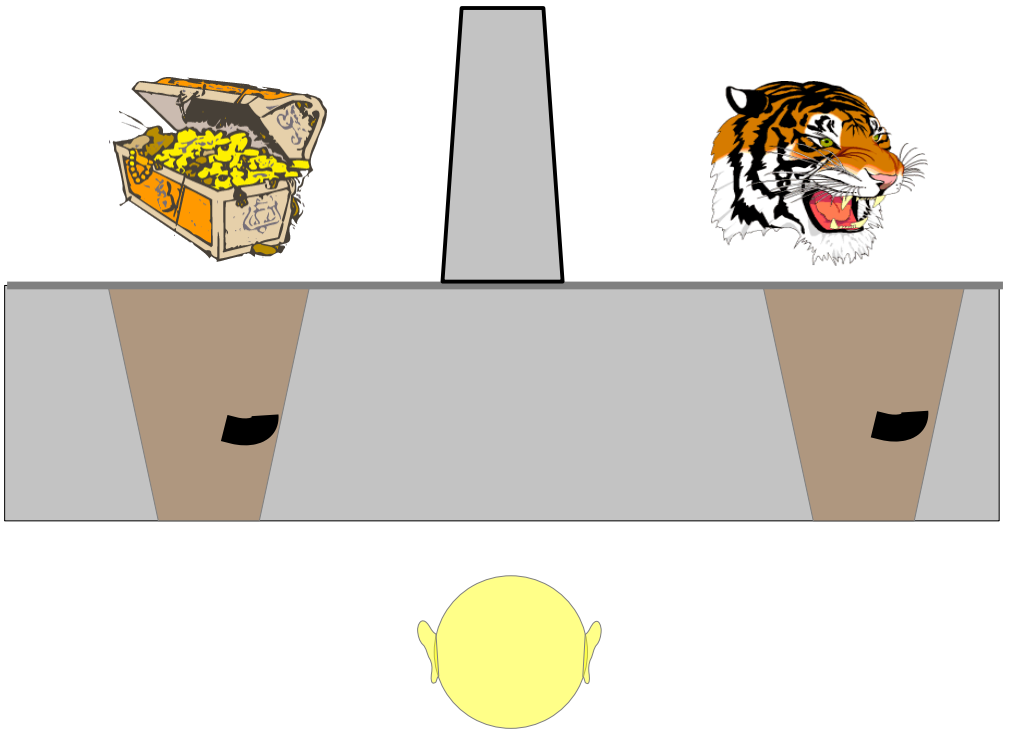
Figure 1: Need for Stochastic Policies. This figure shows a POMDP for which the optimal stationary policy is stochastic. The underlying MDP has 2 states and 2 actions A and B. The payoff for each transition, R or $-R$, is labeled along the transition. The agent sees only one observation. The ellipse around the states $1a$ and $1b$ indicate that both states yield the same observation. This figure is used to prove Facts 1 to 4.

Example from:

Singh, Satinder P., Tommi Jaakkola, and Michael I. Jordan. "Learning without state-estimation in partially observable Markovian decision processes." Machine Learning Proceedings 1994. Morgan Kaufmann, 1994. 284-292.

The Tiger Problem

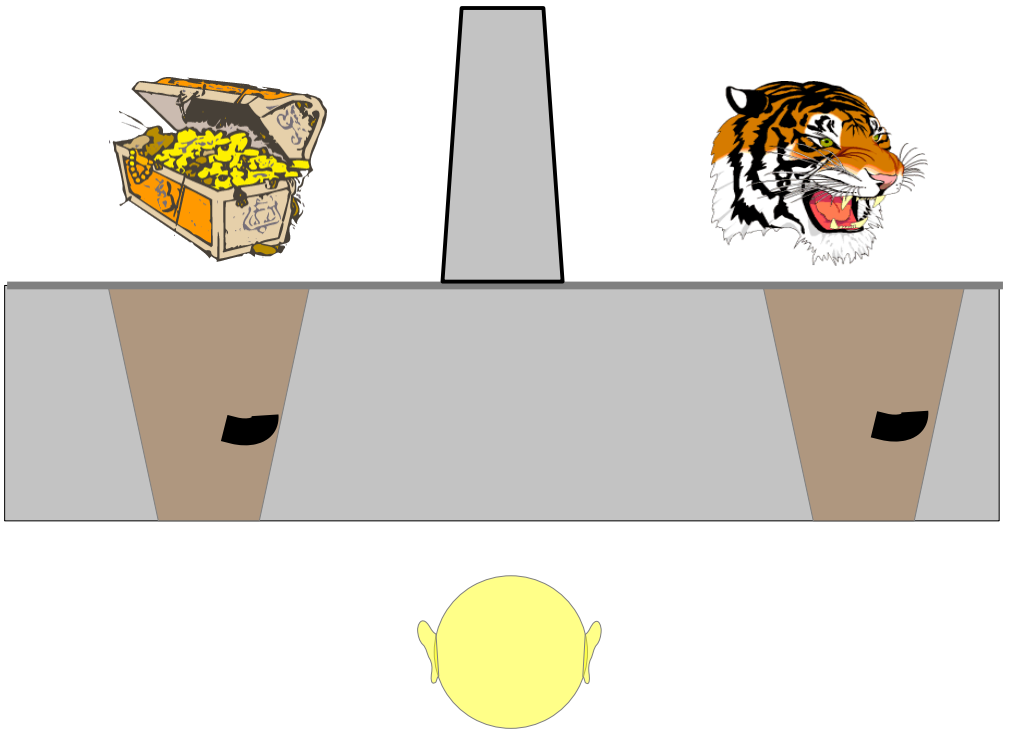
- **States**: left / right (50% prob.)
- **Actions**: Open left, open right, listen
- **Transitions**: static, but opening resets.
- **Observation**: Hear left, Hear right
  - correct 85% of the time.
  - P(HearLeft | Listen, State=left) = 0.85
  - P(HearRight | Listen, State=left) = 0.15
- **Rewards**:
  - correct door +10,
  - wrong door -100
  - listen -1



The Tiger Problem

- **So… when do you open the door?**
  - At the beginning?
  - After HL ?
  - After HL, HL ?
  - After HL, HL, HL ?

  (note: assuming “Listen” so far…!)



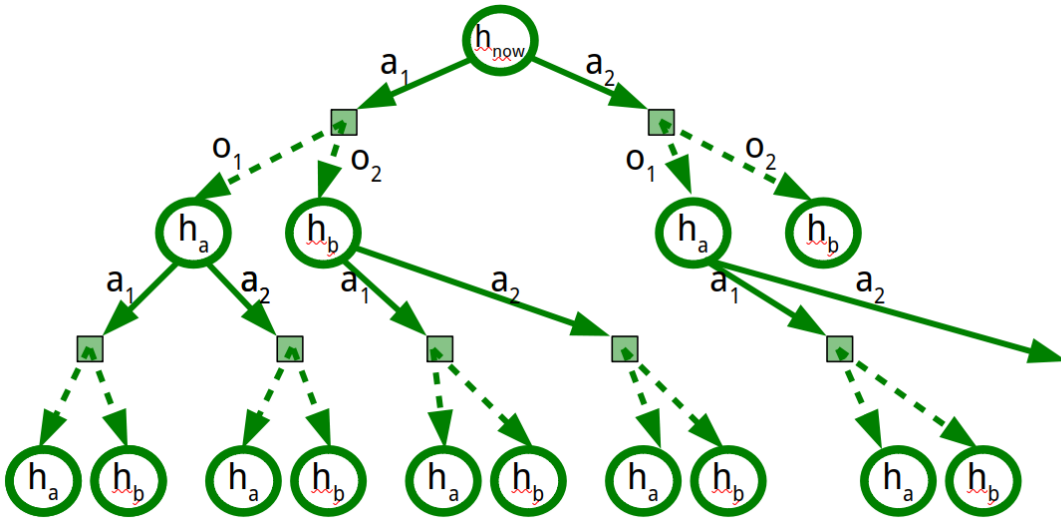
Value of histories

- We act based on histories $h_t=(a_0,o_1,a_1,\ldots o_{t-1},a_{t-1},o_t)$
  - histories take the role of states…
- Indeed, can define a **history MDP** !
  - $M_{HistMDP}=<H,A,T,R>$
- And this leads to value functions:
  - $Q(h,a) = R(h,a) + \Sigma_o P(h'=<h,a,o> | h,a) V(h')$
  - $V(h') = max_{a'} Q(h',a')$

- How are R(h,a) and T(h'|h,a) defined?
  → expectations over the underlying states!

Solving POMDPs

- So we have found a recipe for dealing with POMDPs!
- For a finite horizon:
  - generate the (look-ahead) tree of all action-observation histories
  - perform dynamic programming on this tree
- Problem: too many action-observation histories!



From histories → beliefs

- One would hope: not every history needs different treatment…?
- In the end, it is the states that determine the rewards.
- Suppose for horizon T, we are at the last time step t=T-1…
  - $Q(h_{T-1}, a) = R(h_{T-1}, a) = \Sigma_s P(s \mid h_{T-1}) R(s, a)$

From histories → beliefs

- One would hope: not every history needs different treatment…?
- In the end, it is the states that determine the rewards.
- Suppose for horizon T, we are at the last time step t=T-1…
  - $Q(h_{T-1}, a) = R(h_{T-1}, a) = \Sigma_s P(s \mid h_{T-1}) R(s, a)$

- posterior prob. over states,
- called **belief**, also b(s)
- sufficient to define the value for T-1

Beliefs are ‘sufficient statistics’

- Turns out, that beliefs $b(s) \triangleq P(s \mid h_t)$
  are sufficient to define the value functions for all stages t

- I.e, we can write
  - $Q(b,a) = R(b,a) + \Sigma\, P(b' \mid b, a)\, V(b')$
  - $V(b') = \max_{a'} Q(b',a')$

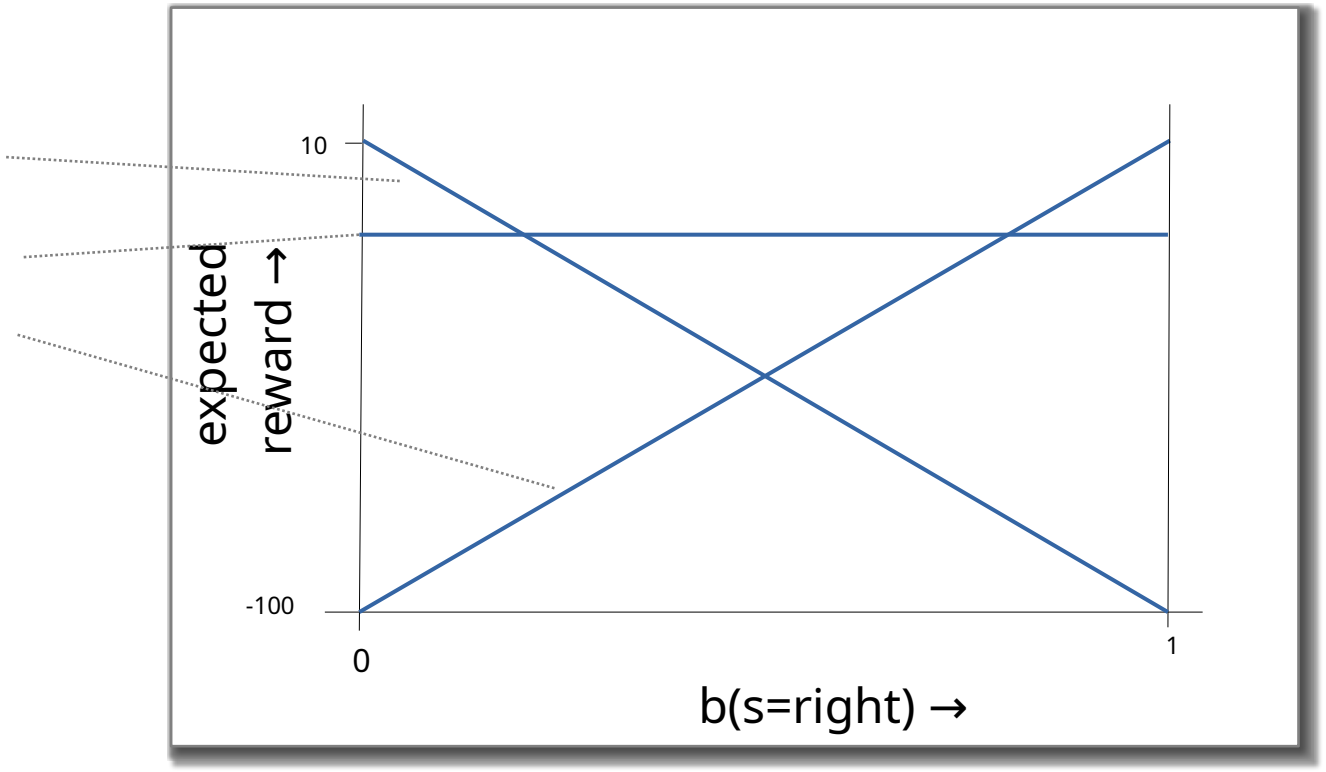
Using beliefs to solve POMDPs

- OK, so now what?
  - Can define “belief MDP”, tree of (reachable) beliefs
  - Q: how does that help…?
- A: many histories can correspond to the same belief!
  - tree → DAG (directed acyclic graph)

- Alternative: plan for the continuum of all possible beliefs…!

DP by Exploiting the PWLC property

- Rewards are vectors
  - R(., OR) = [10, -100]
  - R(., Li) = [-1, 1]
  - R(., OL) = [-100, 10]
- Can perform DP with these vectors
  - E.g. [Spaan 2012]



MTJ Spaan. Partially observable Markov decision processes. Reinforcement learning: State-of-the-art, 387-414, 2012

POMDP Summary

- Many problems are partially observable
  - cannot assume that observations are Markov
- Solutions
  - use histories
  - use beliefs
- Solution approaches:
  - Discrete belief state DP: trees, DAGs
  - Continuous belief state DP: exploit PWLC structure

Part 6: What if we do not want to observe the full state?

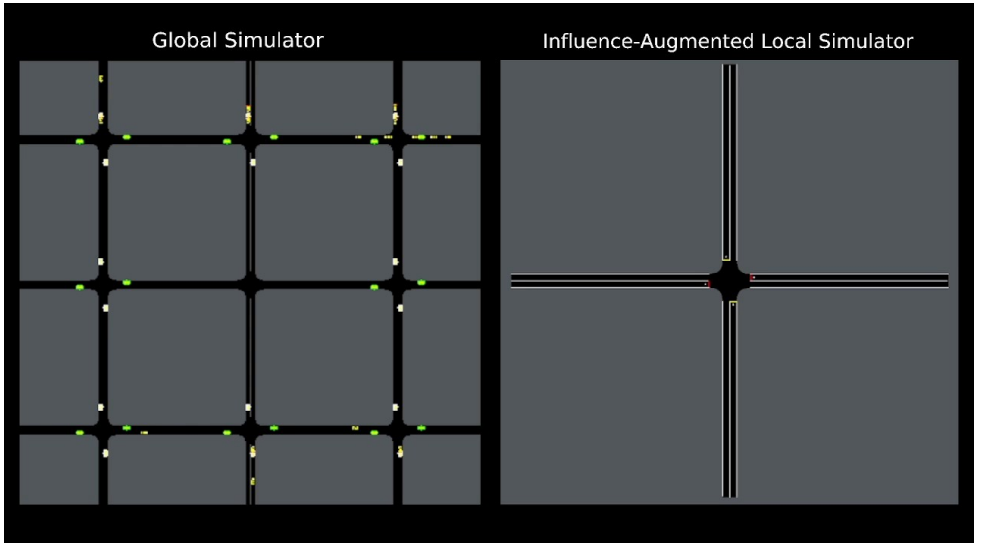
→ Abstraction

Illustration by Nik on Unsplash

Even MDPs are usually difficult...

- Real world problems have **huge state spaces**...

  → can we make abstractions?



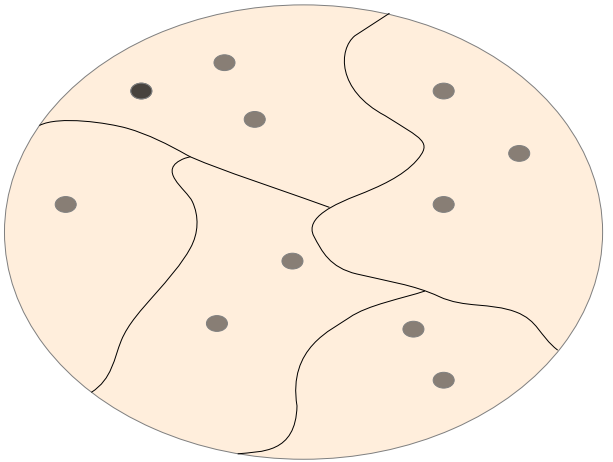
- Specifically, we consider **state abstractions**
  - function $\varphi(s)$ that maps state s → abstract state φ

Suau, Miguel, et al. "Distributed influence-augmented local simulators for parallel MARL in large networked systems." Advances in Neural Information Processing Systems 35 (2022): 28305-28318.

Abstractions partition the state space

- Abstract state φ = cluster of states
  - What are good abstractions?
  - how to cluster…?
- Different types of abstractions:
  - φ_0 — identity (i.e., no abstraction)
  - φ_m — model irrelevance, preserve R,T
    - φ_Π — Q^Π irrelevance (for all $\pi \in \Pi$), preserves Q-values
    - φ_{Q^*} — Q* irrelevance, preserves all optimal Q-values
    - φ_{a^*} — a* irrelevance, preserve Q(., a*)
    - φ_{π^*} — π* irrelevance, preserves optimal action

coarser



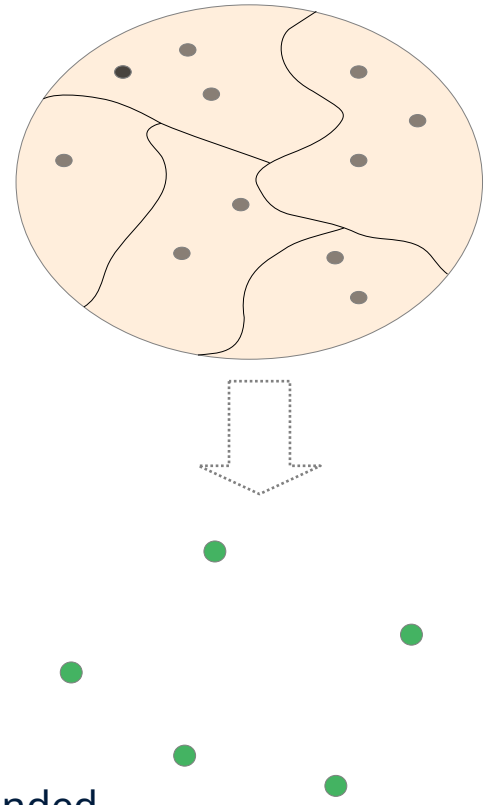
- Hierarchy:

$$\varphi_0 \text{ ISA } \varphi_m \text{ ISA } \varphi_\Pi \text{ ISA } \varphi_{Q^*} \text{ ISA } \varphi_{Q^*} \text{ ISA } \varphi_{a^*} \text{ ISA } \varphi_{\pi^*}$$

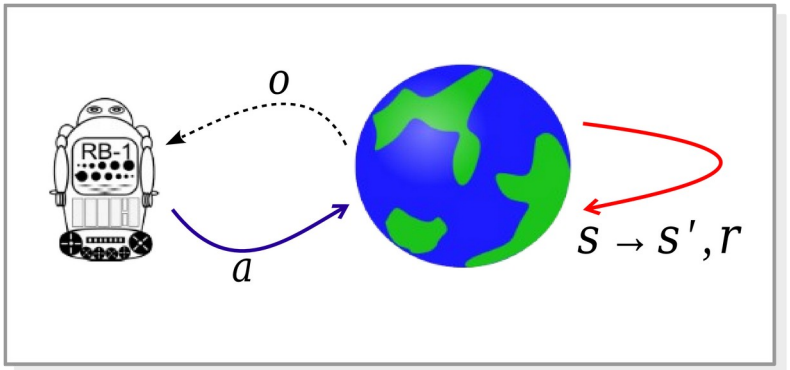
, Lihong, Thomas J. Walsh, and Michael L. Littman. "Towards a unified theory of state abstraction for MDPs." AI&M 1.2 (2006): 3.

Abstract MDPs

- Given and MDP and some φ….
  ….can create an **abstract MDP:**
- Weighting function $\omega_\varphi(s)$
  - specifies the assumed state probabilities
  - for each abstract state φ
- Transitions:
  $T(\varphi' \mid \varphi, a) = \Sigma_{s' \in \varphi'} \Sigma_{s \in \varphi} T(s' \mid s, a)\ \omega_\varphi(s)$
- Rewards:
  $R(\varphi, a) = \Sigma_{s \in \varphi} R(s, a)\ \omega_\varphi(s)$
- Under some assumptions (‘ε-model similarity abstraction’): value loss bounded.



Abstraction as a POMDP



- Abstraction can be thought of as a POMDP!
  - abstract states are observations: $\varphi \leftrightarrow o$
  - myopic decisions in these POMDPs can be good!

- When entering φ, there is a distribution over states
  - there **is** a true belief, that depends on history $h_t=(\varphi_0,a_0,\ldots,a_{t-1},\varphi_t)$

- $\omega_\varphi(s)$ approximates that belief
  - in a non-history dependent way

    → an Abstract MDP is an MDP

    → an Abstract MDP can be constructed and used for planning, **it can not be 'experienced'**

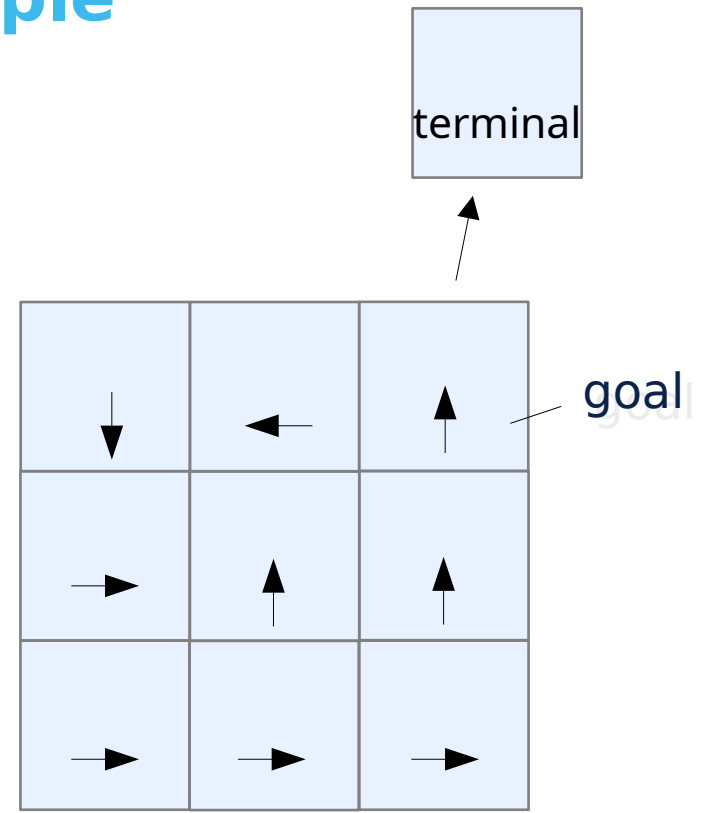
Conclusions

- Many problems are sequential and stochastic → model as MDPs
- ‘Solving’ MDPs
  - finite horizon: ‘plain’ dynamic progamming
  - infinite horizon: (generalized) policy iteration
- Partial observable problems… → POMDPs
- Large problems: state abstraction…
  - are making the problem a POMDP!

Policy Evaluation Example

- A little maze:
  - transitions:

    N,E,S,W, deterministic movements
    P(s' = terminal | s=goal, a=*)=1
  - rewards:
    R(s=Goal,a=*) = +10
    R(s=terminal, a=*) = 0
    R(s=*, a=*) = -1 (otherwise)
  - discount γ=0.9

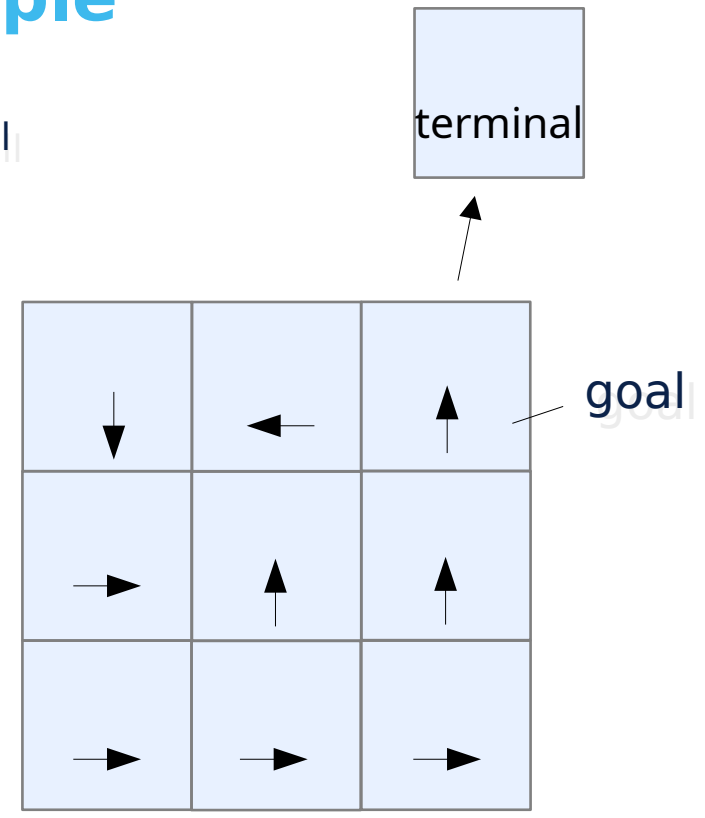


Policy Evaluation Example

- A little maze:
  - transitions:

    N,E,S,W, deterministic movements
    P(s' = terminal | s=goal, a=*)=1
  - rewards:
    R(s=Goal,a=*) = +10
    R(s=terminal, a=*) = 0
    R(s=*, a=*) = -1 (otherwise)
  - discount γ=0.9

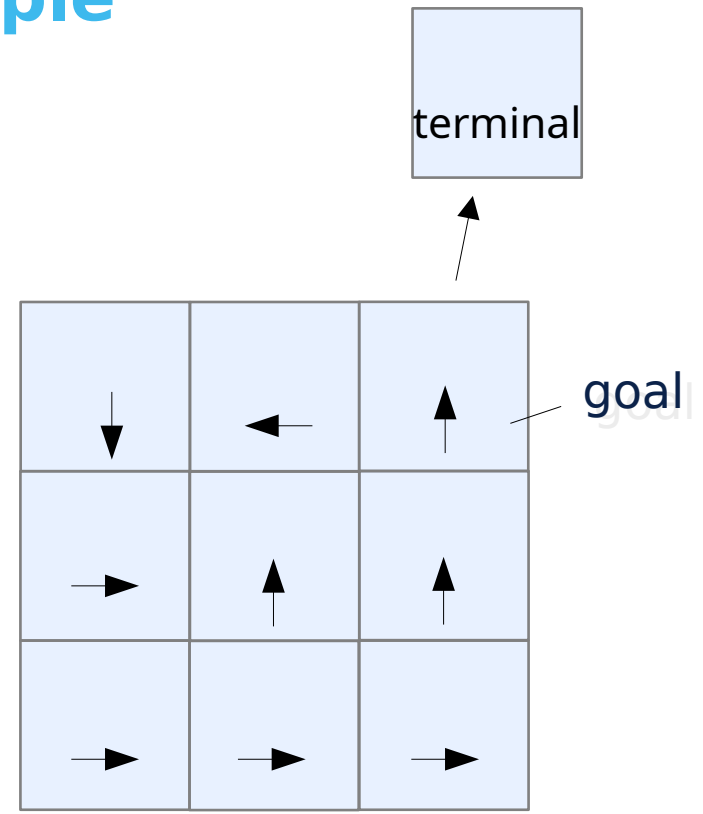
when hitting a wall
we stay in place



Policy Evaluation Example

- A little maze:
  - transitions:

    N,E,S,W, deterministic movements
    P(s' = terminal | s=goal, a=*)=1
  - rewards:
    R(s=Goal,a=*) = +10
    R(s=terminal, a=*) = 0
    R(s=*, a=*) = -1 (otherwise)
  - discount γ=0.9

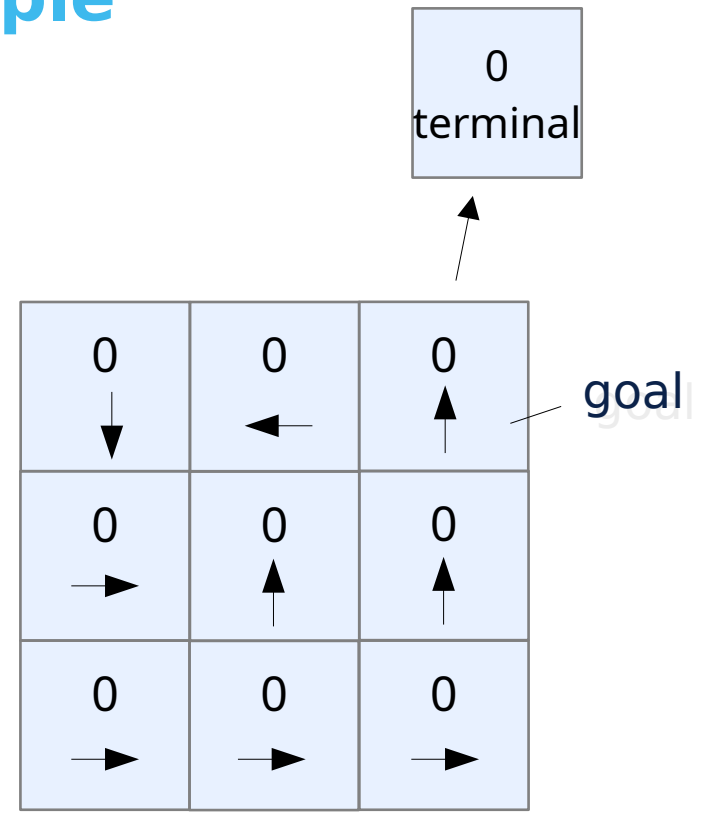


First step…?

Policy Evaluation Example

- A little maze:
  - transitions:

    N,E,S,W, deterministic movements
    P(s' = terminal | s=goal, a=*)=1
  - rewards:
    R(s=Goal,a=*) = +10
    R(s=terminal, a=*) = 0
    R(s=*, a=*) = -1 (otherwise)
  - discount γ=0.9



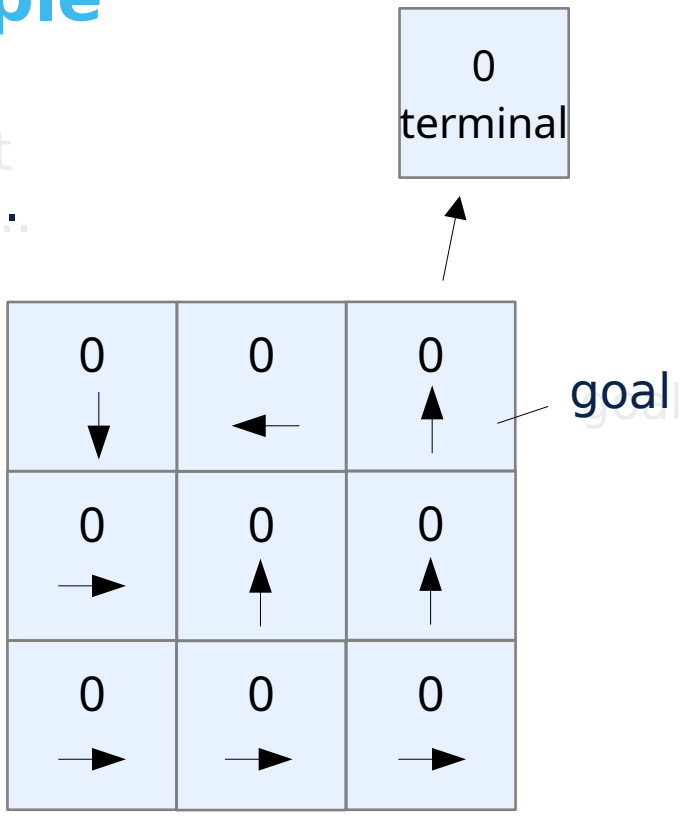
Initialize $v_0(s) = 0$

Policy Evaluation Example

- A little maze:
  - transitions:

    N,E,S,W, deterministic movements
    P(s' = terminal | s=goal, a=*)=1
  - rewards:
    R(s=Goal,a=*) = +10
    R(s=terminal, a=*) = 0
    R(s=*, a=*) = -1 (otherwise)
  - discount γ=0.9

Let's start updating...



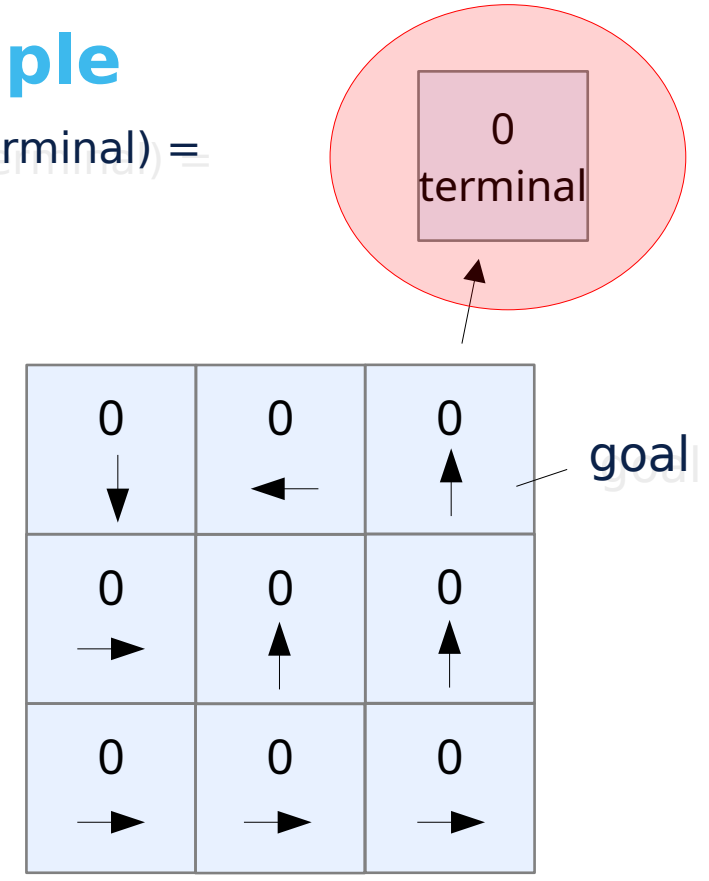
$$v_1(s) := r(s, \pi(s)) + \sum_{s'} p(s'|s, \pi(s)) \gamma v_0(s')$$

Policy Evaluation Example

- A little maze:
  - transitions:

    N,E,S,W, deterministic movements
    P(s' = terminal | s=goal, a=*)=1
  - rewards:
    R(s=Goal,a=*) = +10
    R(s=terminal, a=*) = 0
    R(s=*, a=*) = -1 (otherwise)
  - discount γ=0.9

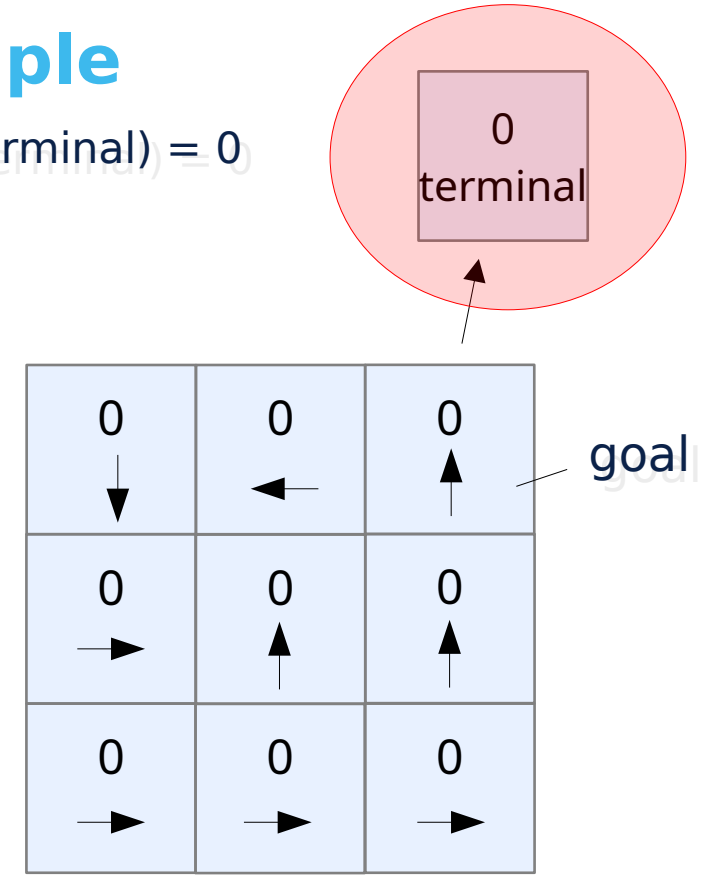
v_1(terminal) = … ?



$$v_1(s) := r(s,\pi(s)) + \sum_{s'} p(s'|s,\pi(s))\,\gamma\, v_0(s')$$

Policy Evaluation Example

v_1(terminal) = 0



- A little maze:
  - transitions:

    N,E,S,W, deterministic movements
    P(s' = terminal | s=goal, a=*)=1
  - rewards:
    R(s=Goal,a=*) = +10
    R(s=terminal, a=*) = 0
    R(s=*, a=*) = -1 (otherwise)
  - discount γ=0.9

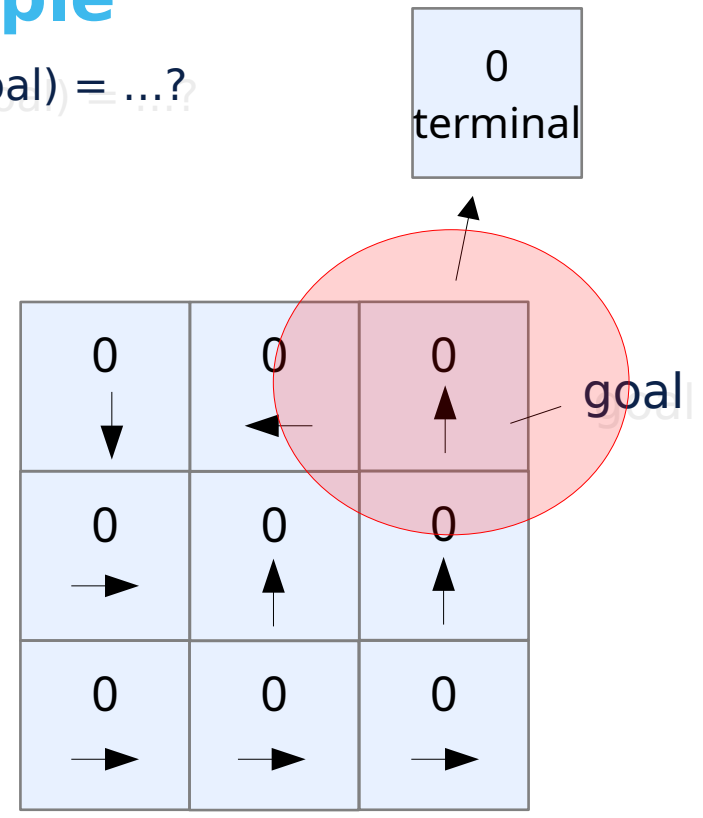
$$v_1(s) := r(s, \pi(s)) + \sum_{s'} p(s'|s, \pi(s)) \gamma v_0(s')$$

Policy Evaluation Example

$v_1(goal) = ...?$

- A little maze:
  - transitions:

    N,E,S,W, deterministic movements
    P(s' = terminal | s=goal, a=*)=1
  - rewards:
    R(s=Goal,a=*) = +10
    R(s=terminal, a=*) = 0
    R(s=*, a=*) = -1 (otherwise)
  - discount γ=0.9

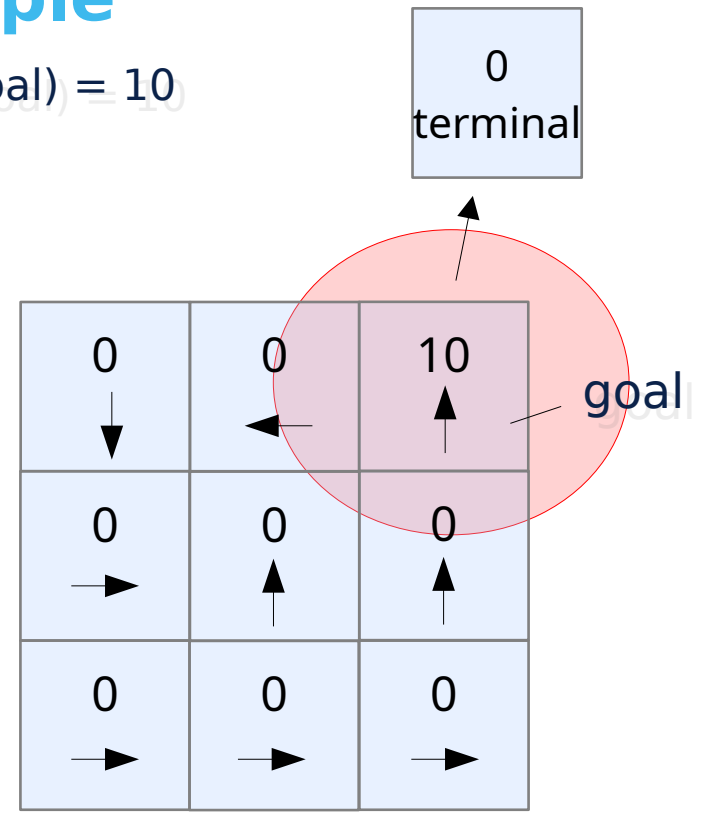


$$v_1(s) := r(s, \pi(s)) + \sum_{s'} p(s'|s, \pi(s)) \gamma v_0(s')$$

Policy Evaluation Example

- A little maze:
  - transitions:

    N,E,S,W, deterministic movements
    P(s' = terminal | s=goal, a=*)=1
  - rewards:
    R(s=Goal,a=*) = +10
    R(s=terminal, a=*) = 0
    R(s=*, a=*) = -1 (otherwise)
  - discount γ=0.9



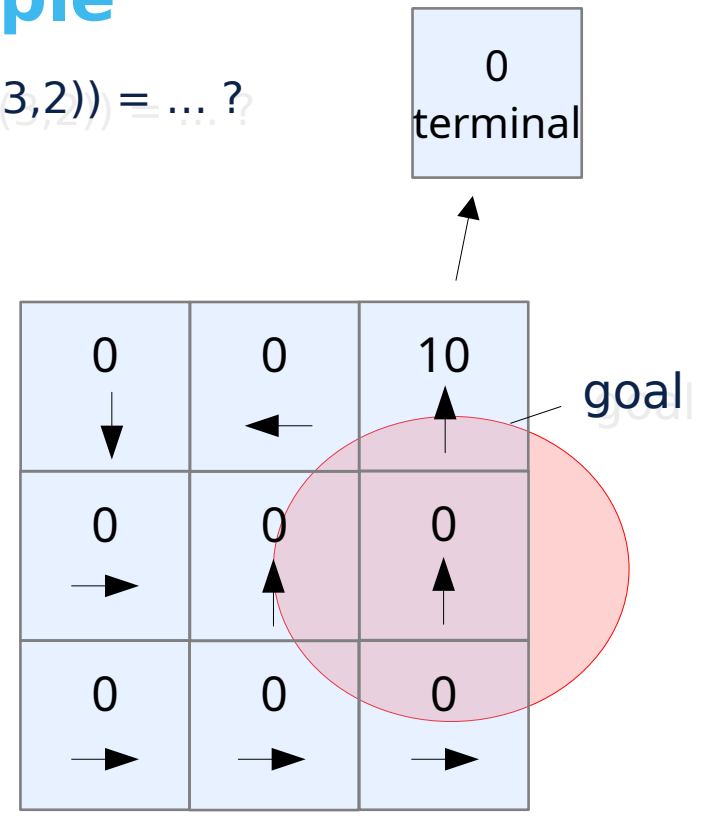
$$v_1(s) := r(s, \pi(s)) + \sum_{s'} p(s'|s, \pi(s)) \gamma v_0(s')$$

Policy Evaluation Example

$v_1((3,2)) = \ldots ?$

- A little maze:
  - transitions:

    N,E,S,W, deterministic movements
    P(s' = terminal | s=goal, a=*)=1
  - rewards:
    R(s=Goal,a=*) = +10
    R(s=terminal, a=*) = 0
    R(s=*, a=*) = -1 (otherwise)
  - discount γ=0.9



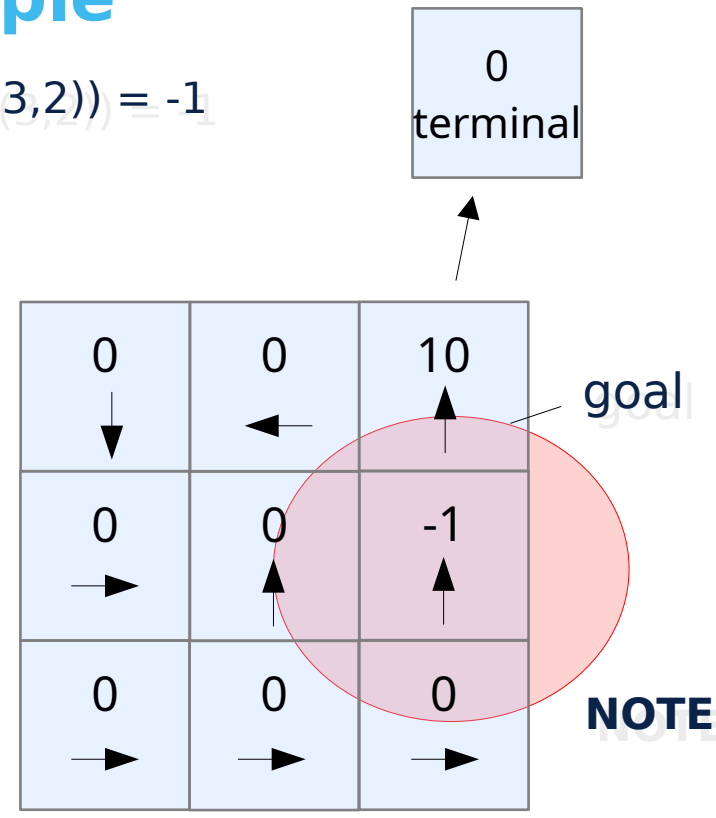
$$v_1(s) := r(s, \pi(s)) + \sum_{s'} p(s'|s, \pi(s)) \gamma v_0(s')$$

Policy Evaluation Example

- A little maze:
  - transitions:

    N,E,S,W, deterministic movements
    P(s' = terminal | s=goal, a=*)=1
  - rewards:
    R(s=Goal,a=*) = +10
    R(s=terminal, a=*) = 0
    R(s=*, a=*) = -1 (otherwise)
  - discount γ=0.9

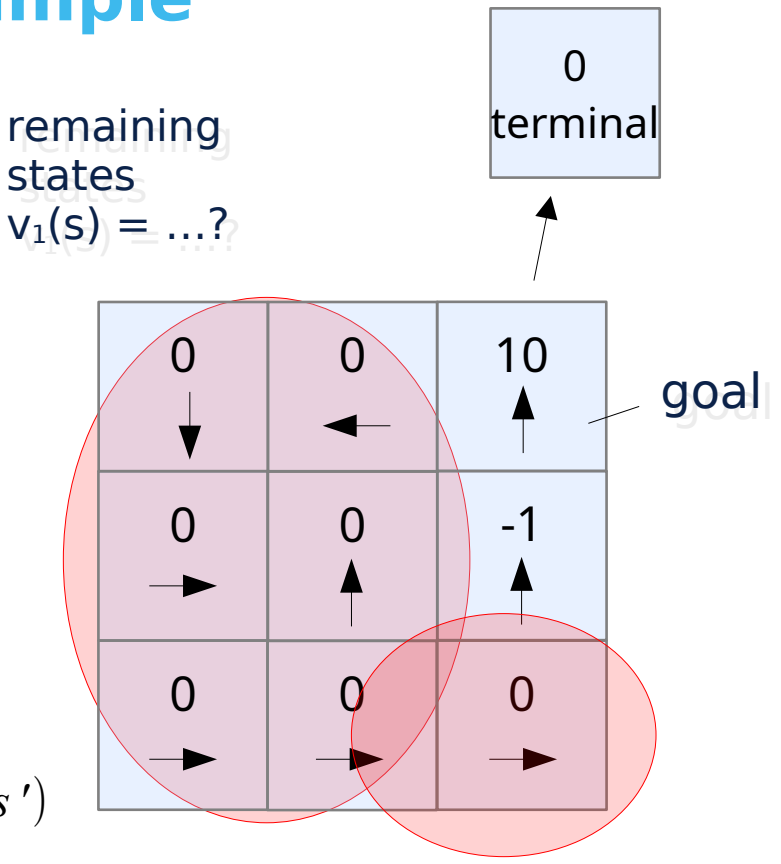
$$v_1(s) := r(s, \pi(s)) + \sum_{s'} p(s'|s, \pi(s)) \gamma v_0(s')$$



Policy Evaluation Example

- A little maze:
  - transitions:

    N,E,S,W, deterministic movements
    P(s' = terminal | s=goal, a=*)=1
  - rewards:
    R(s=Goal,a=*) = +10
    R(s=terminal, a=*) = 0
    R(s=*, a=*) = -1 (otherwise)
  - discount γ=0.9

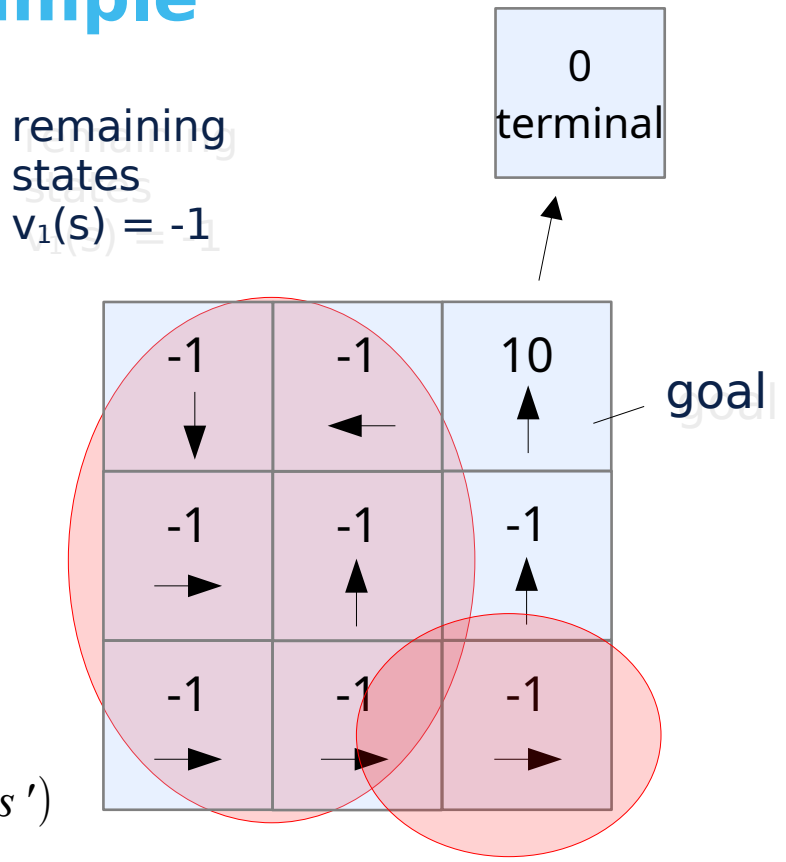


$$v_1(s) := r(s, \pi(s)) + \sum_{s'} p(s'|s, \pi(s))\, \gamma\, v_0(s')$$

Policy Evaluation Example

- A little maze:
  - transitions:

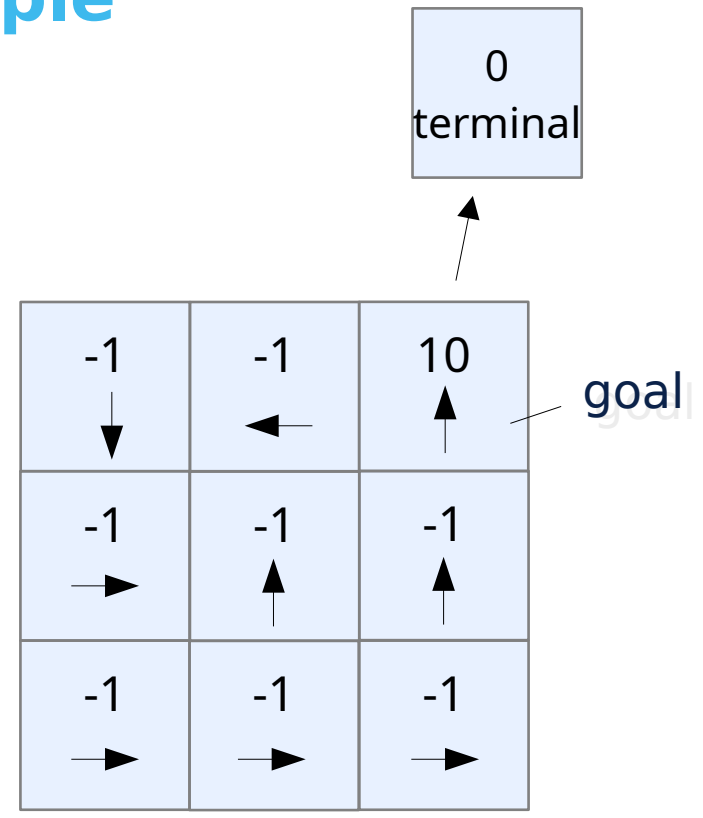
    N,E,S,W, deterministic movements
    P(s' = terminal | s=goal, a=*)=1
  - rewards:
    R(s=Goal,a=*) = +10
    R(s=terminal, a=*) = 0
    R(s=*, a=*) = -1 (otherwise)
  - discount γ=0.9



$$v_1(s) := r(s, \pi(s)) + \sum_{s'} p(s'|s, \pi(s))\, \gamma\, v_0(s')$$

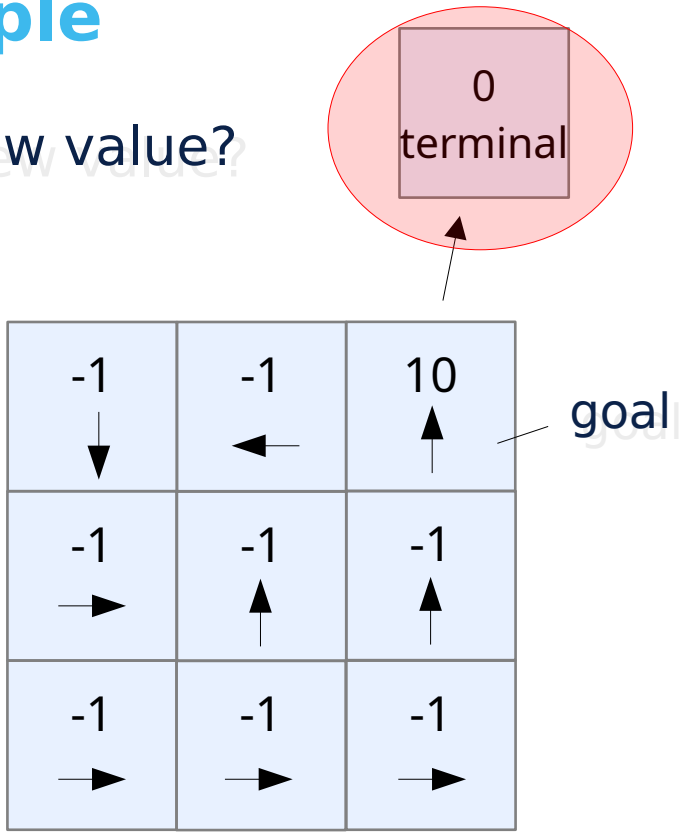
Policy Evaluation Example

Have now computed v_1, lets move to v_2...



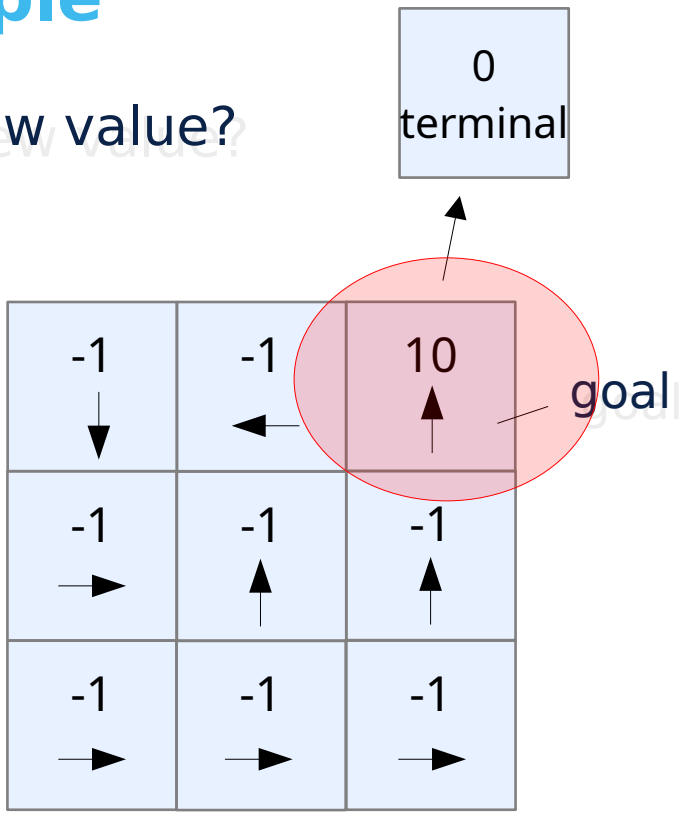
Policy Evaluation Example

new value?



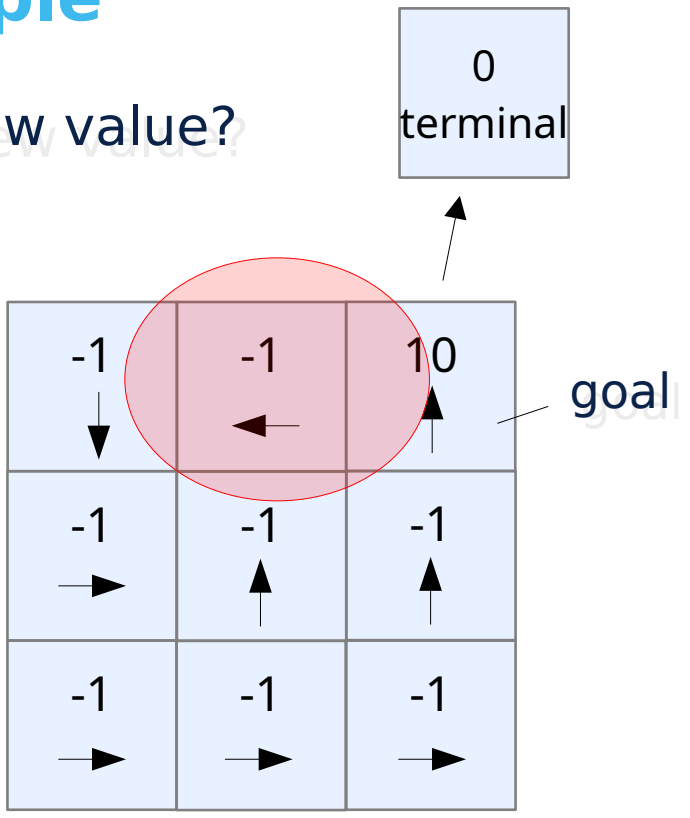
$$v_2(s) := r(s, \pi(s)) + \sum_{s'} p(s'|s, \pi(s)) \gamma v_1(s')$$

Policy Evaluation Example



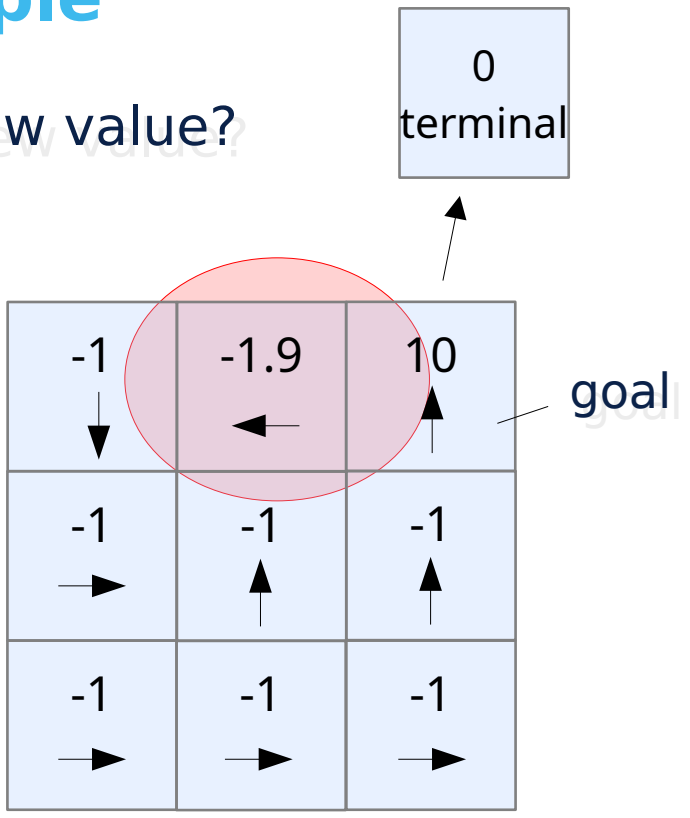
$$v_2(s) := r(s, \pi(s)) + \sum_{s'} p(s'|s, \pi(s)) \gamma v_1(s')$$

Policy Evaluation Example



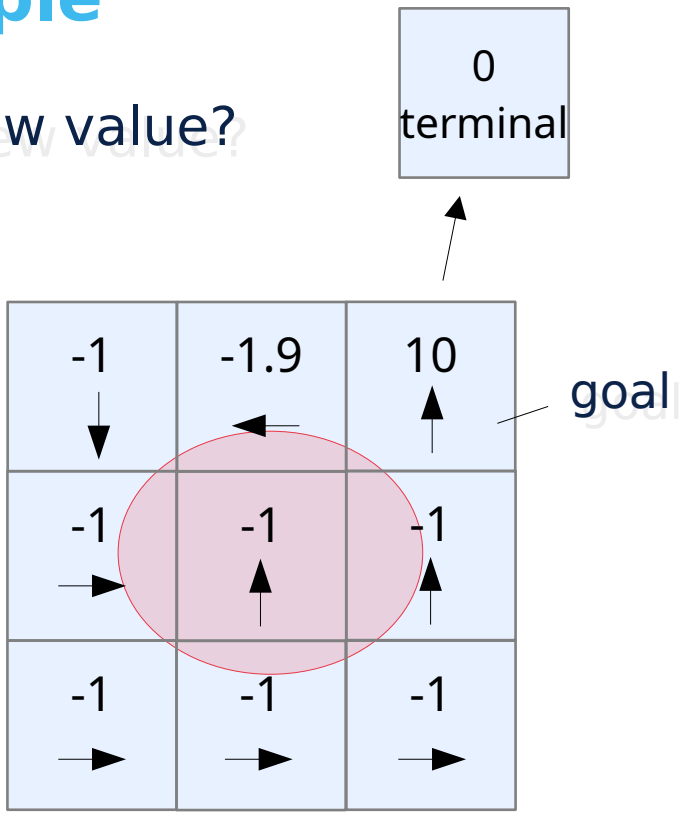
$$v_2(s) := r(s, \pi(s)) + \sum_{s'} p(s'|s, \pi(s)) \gamma v_1(s')$$

Policy Evaluation Example



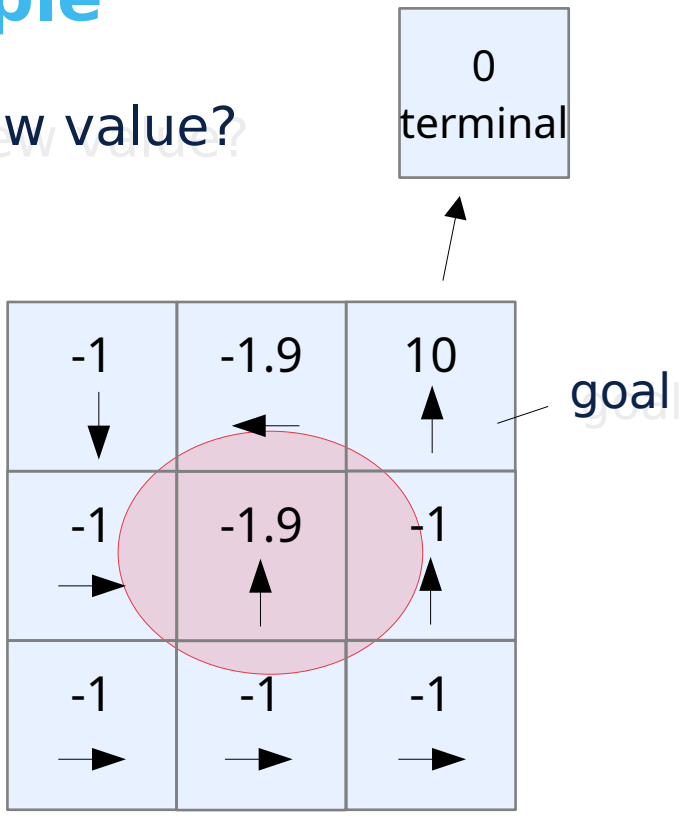
$$v_2(s) := r(s, \pi(s)) + \sum_{s'} p(s'|s, \pi(s)) \gamma v_1(s')$$

Policy Evaluation Example



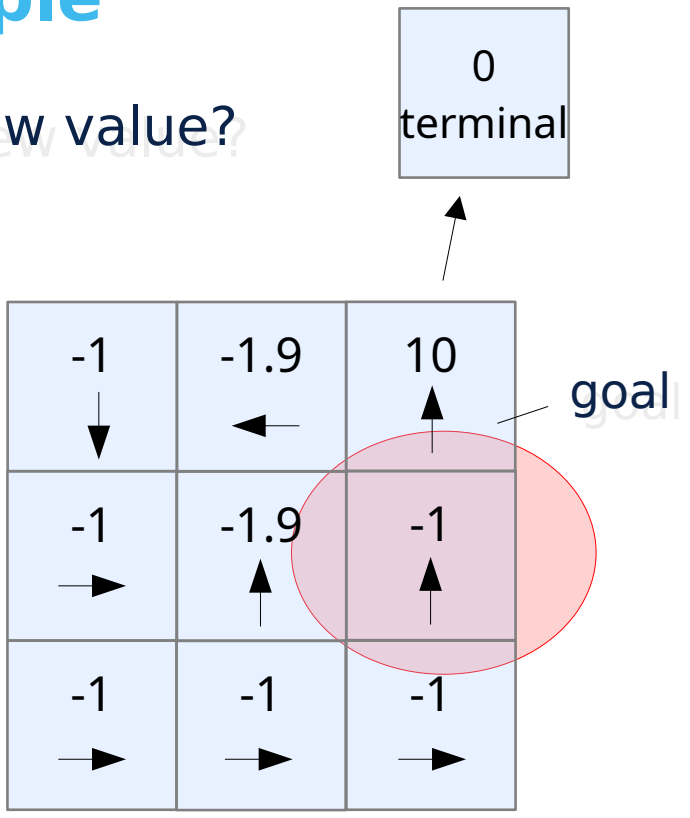
$$v_2(s) := r(s, \pi(s)) + \sum_{s'} p(s'|s, \pi(s)) \gamma v_1(s')$$

Policy Evaluation Example



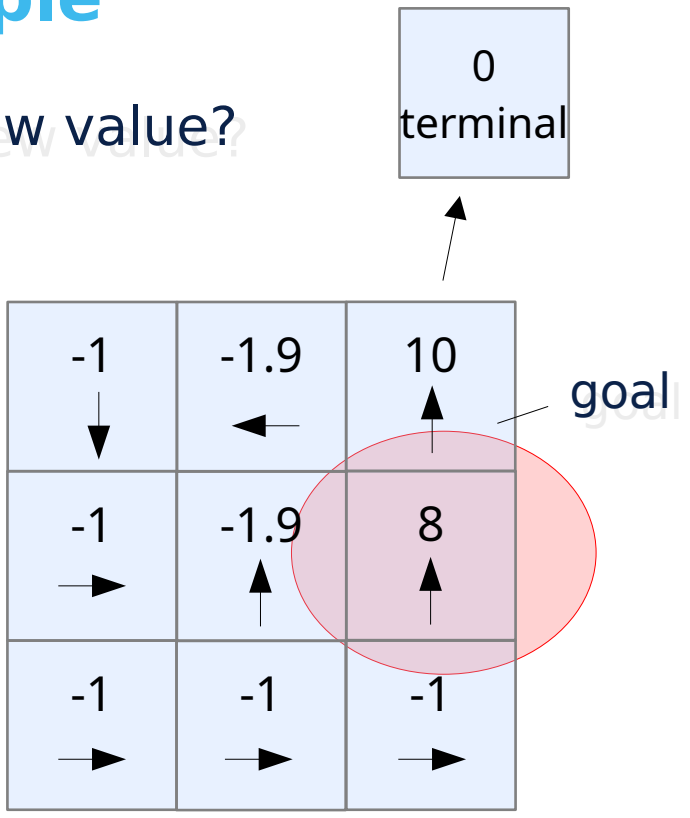
$$v_2(s) := r(s, \pi(s)) + \sum_{s'} p(s'|s, \pi(s)) \gamma v_1(s')$$

Policy Evaluation Example



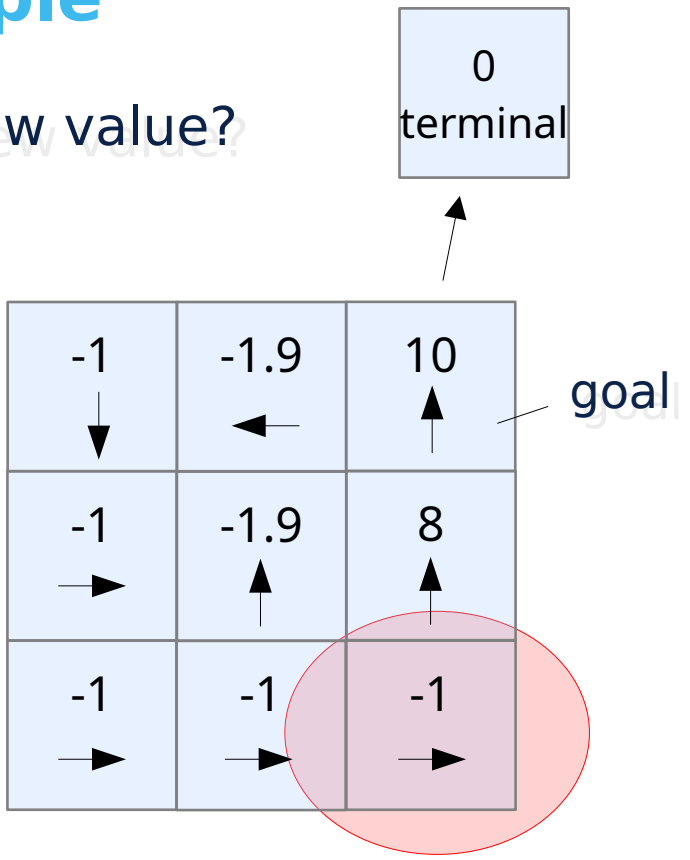
$$v_2(s) := r(s, \pi(s)) + \sum_{s'} p(s'|s, \pi(s)) \gamma v_1(s')$$

Policy Evaluation Example



$$v_2(s) := r(s, \pi(s)) + \sum_{s'} p(s'|s, \pi(s)) \gamma v_1(s')$$

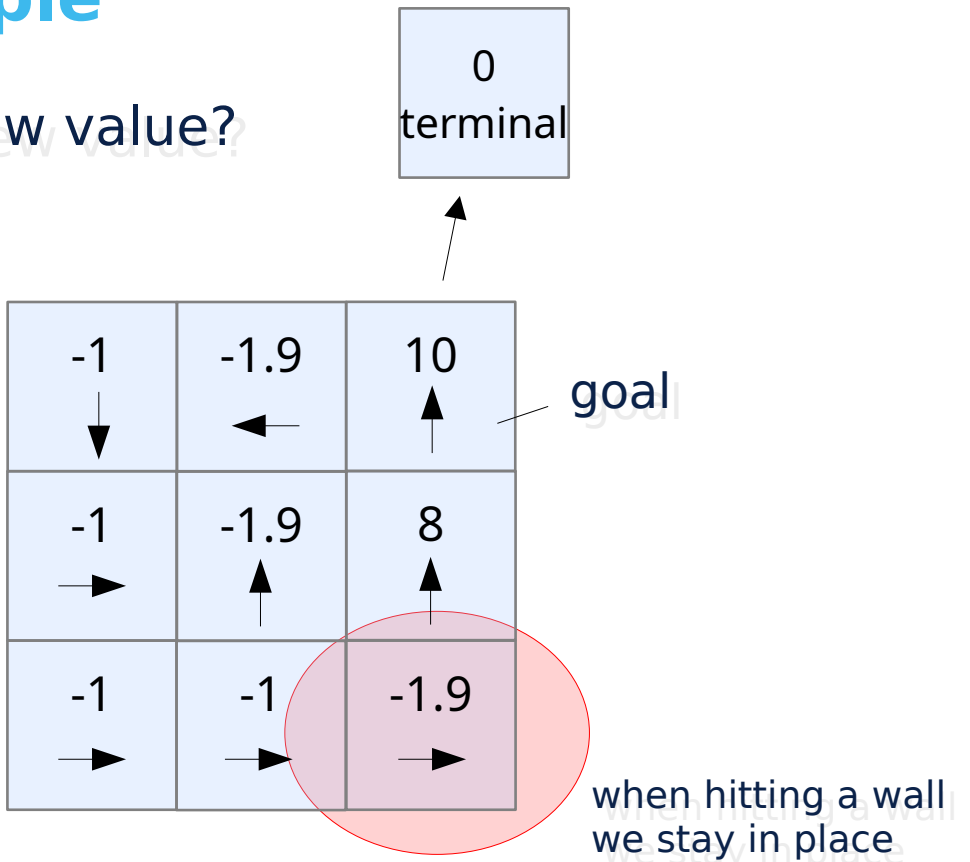
Policy Evaluation Example



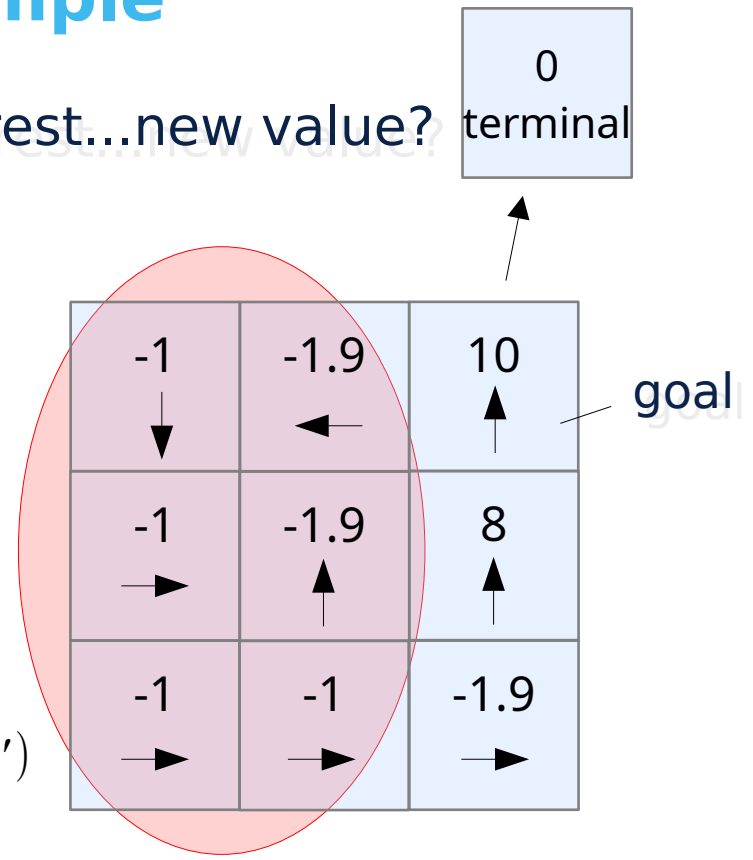
$$v_2(s) := r(s, \pi(s)) + \sum_{s'} p(s'|s, \pi(s)) \gamma v_1(s')$$

Policy Evaluation Example

$$v_2(s) := r(s, \pi(s)) + \sum_{s'} p(s'|s, \pi(s)) \gamma v_1(s')$$



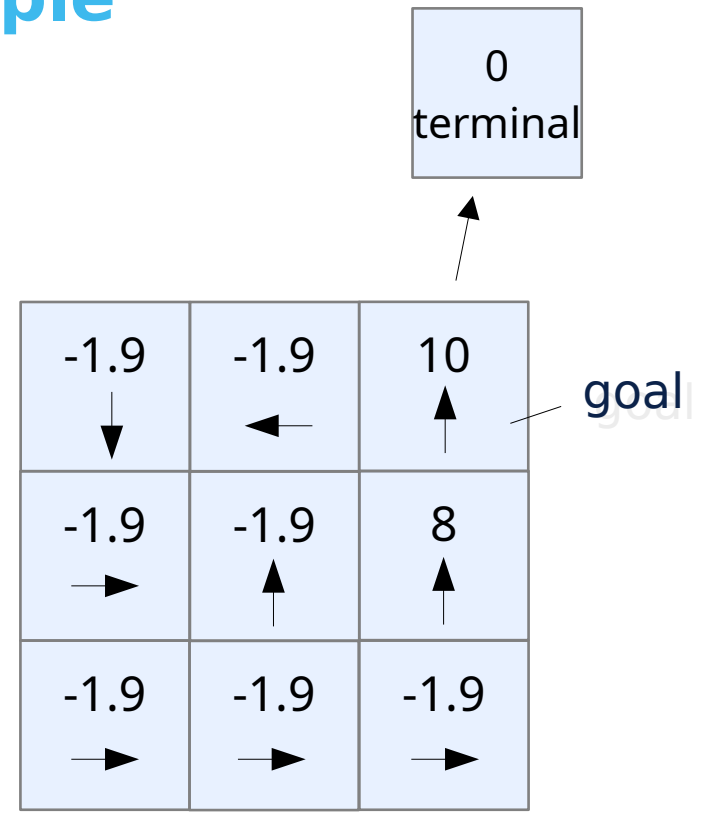
Policy Evaluation Example



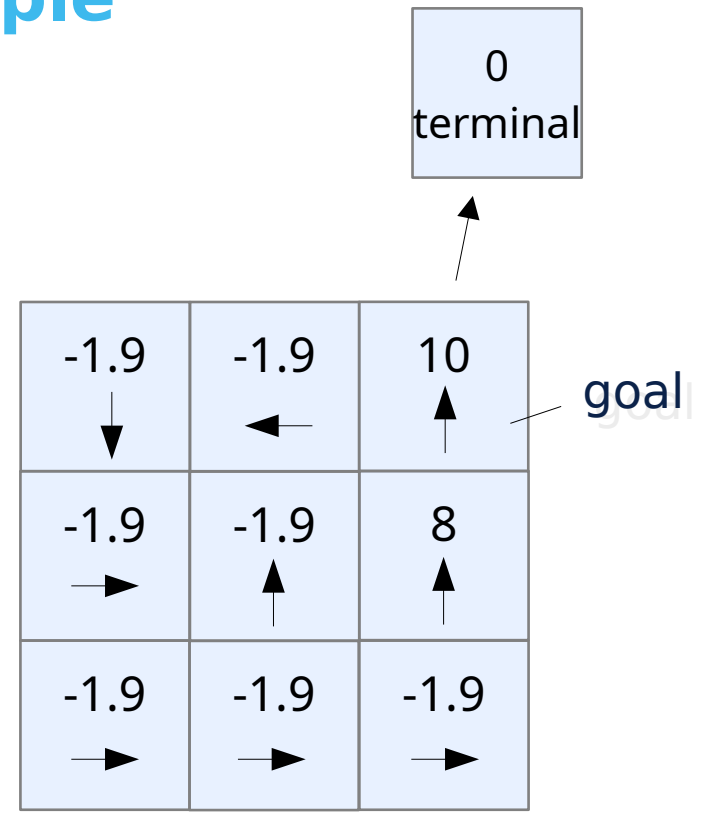
$$v_2(s) := r(s, \pi(s)) + \sum_{s'} p(s'|s, \pi(s)) \gamma v_1(s')$$

Policy Evaluation Example

have now computed v_2
(the 2-steps-to-go value function)

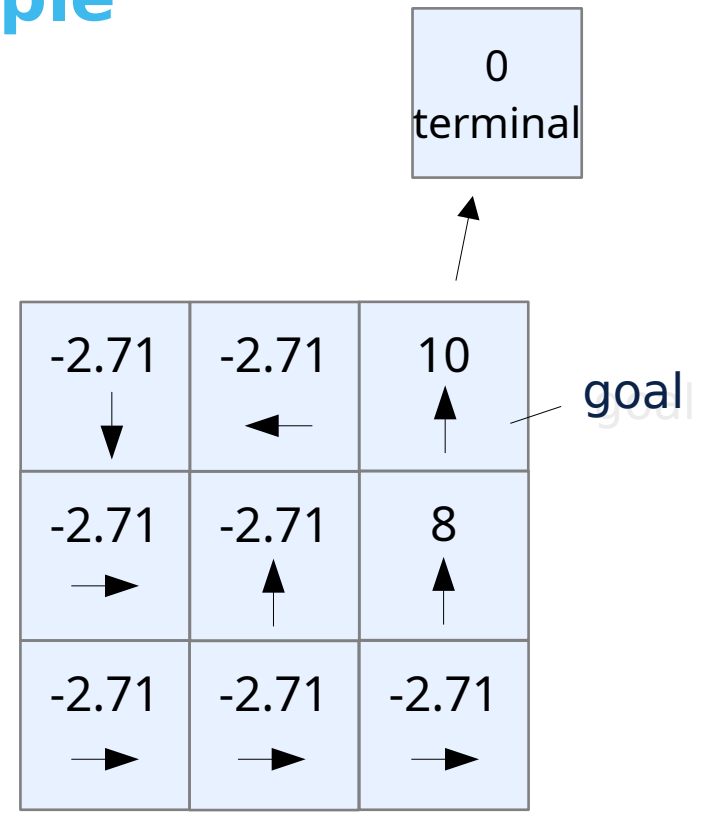


Policy Evaluation Example



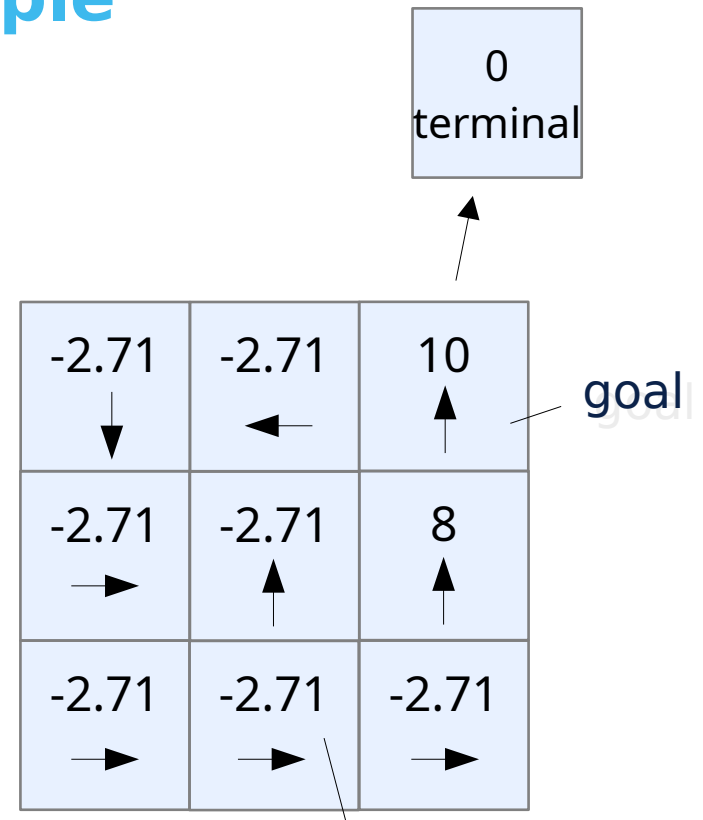
how about v_3...?

Policy Evaluation Example



v_3 is this:

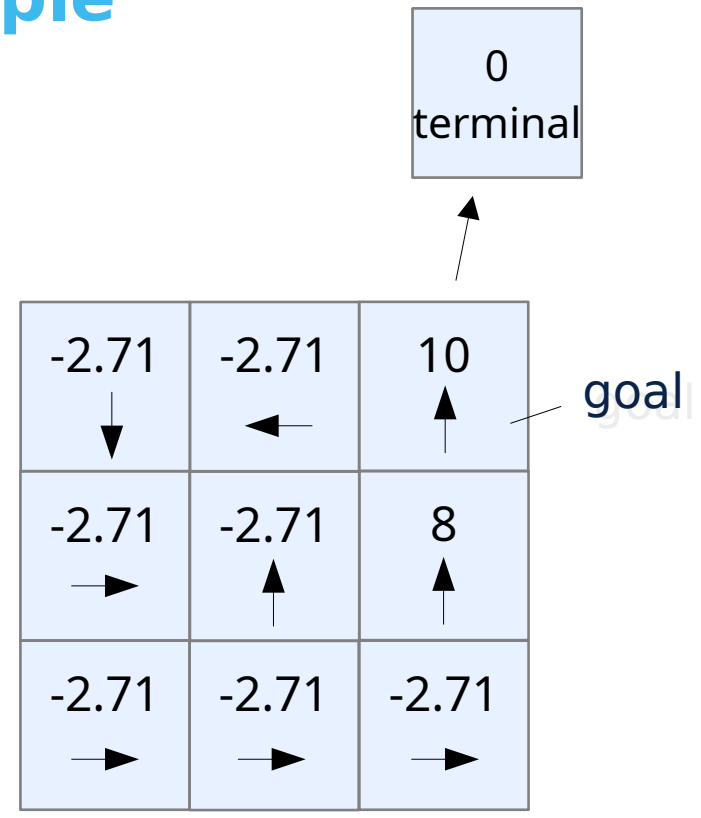
Policy Evaluation Example



v_3 is this:

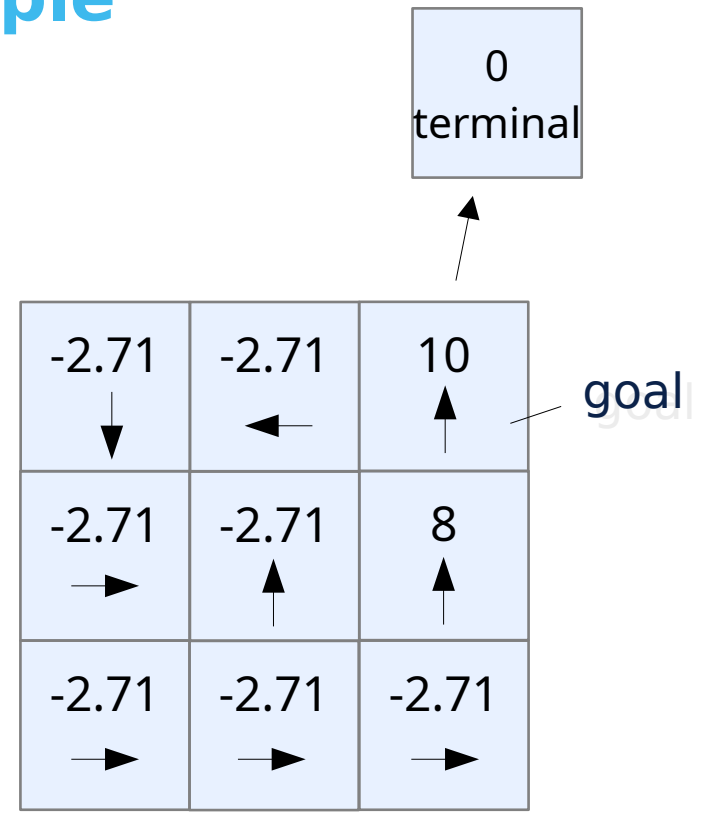
$-1 + .9 * (-1) + .9^2 * (-1) = -2.71$

Policy Evaluation Example



What does this converge to…?

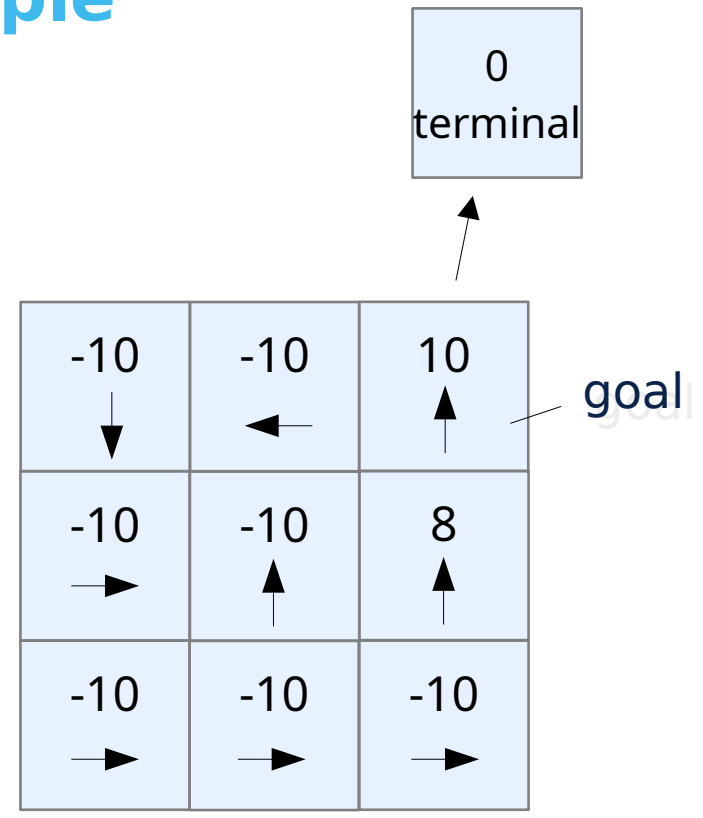
Policy Evaluation Example



What does this converge to…?

$$\sum_{t=0}^{\infty} \gamma^t r = \frac{r}{1-\gamma} = \frac{-1}{1-0.9} = -10$$

Policy Evaluation Example



$v_\pi = v_\infty$ is this: