

Reinforcement Learning Introduction

William Uther
NICTA/UNSW

William.Uther@nicta.com.au

Some diagrams from Rich Sutton's course slides:
CMP499/609 Reinforcement Learning in AI
<http://rlai.cs.ualberta.ca/RLAI/RLAICourse/RLAICourse2007.html>

Some diagrams/slides courtesy Scott Sanner @
NICTA/ANU

Afternoon Schedule

<i>Time</i>	<i>Topic</i>
<i>2:30-3:30</i>	<i>Overview / Introduction</i>
<i>3:50-4:50</i>	<i>Standard Solution Techniques</i>
<i>5:00-6:00</i>	<i>Function Approximation</i>

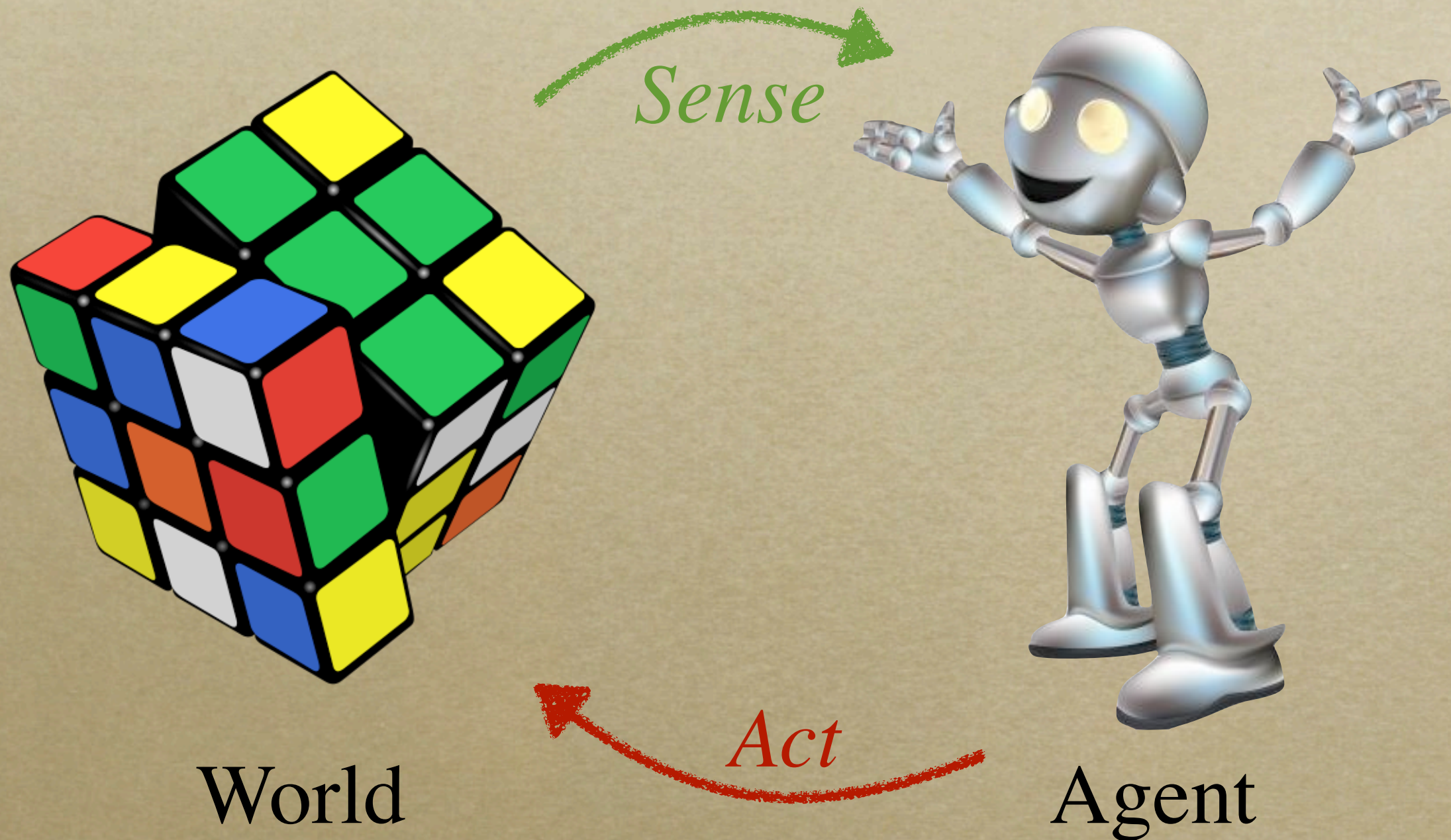
Overview Overview :)

- What is Reinforcement Learning?
- Common Optimisation Criteria
- Basic Dynamic Programming Solutions

What is Reinforcement Learning?

- RL is a class of **problems**,
 - Not a class of solutions/techniques
- We want an agent to
 - **learn** how to behave through interaction with its environment

What is Reinforcement Learning?



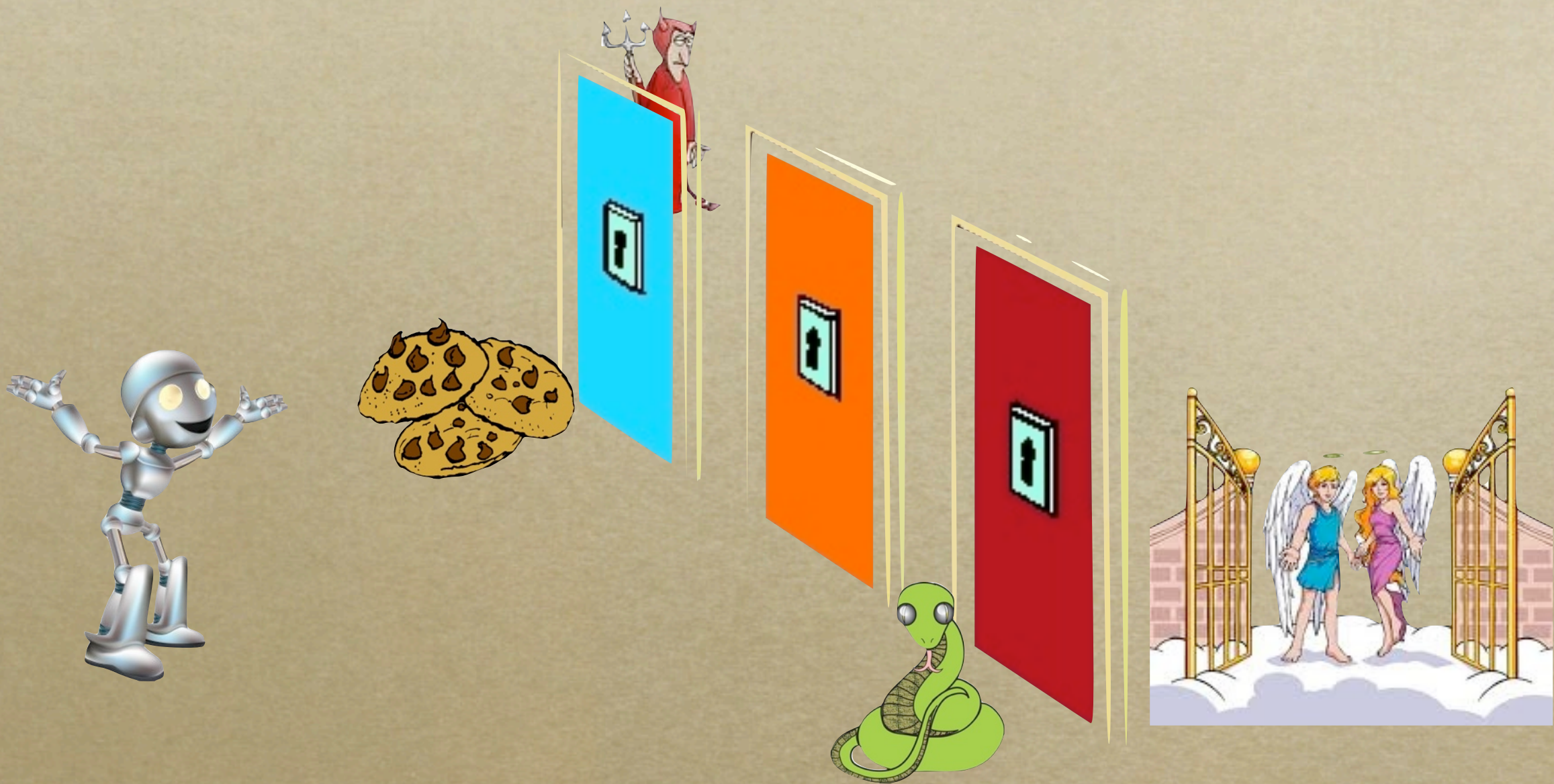
What is Reinforcement Learning?

- We want an agent to
 - **learn** how to behave
 - with minimal prior knowledge
 - (we want it to learn)
- so how do we communicate with it to specify the goal?

What is Reinforcement Learning?

- Train it with rewards - minimal goal language
- Learn how to behave
 - through interaction with the environment
 - to maximise reward received

Self Control?



What is Reinforcement Learning?

- Learn how to behave
 - through interaction with the environment
 - to maximise reward received
 - over the longer term

Formalities : Defining the world

- Partially Observable Markov Decision Processes (POMDPs)

$$\langle \mathcal{S}, \mathcal{A}, \mathcal{O}, T, \Omega, R \rangle$$

$$\mathcal{S} = \{s_1, \dots, s_{|\mathcal{S}|}\} \quad \mathcal{A} = \{a_1, \dots, a_{|\mathcal{A}|}\} \quad \mathcal{O} = \{o_1, \dots, o_{|\mathcal{O}|}\}$$

$$T : \mathcal{S} \times \mathcal{A} \rightarrow (\mathcal{S} \rightarrow [0, 1])$$

$$\Omega : \mathcal{S} \rightarrow (\mathcal{O} \rightarrow [0, 1])$$

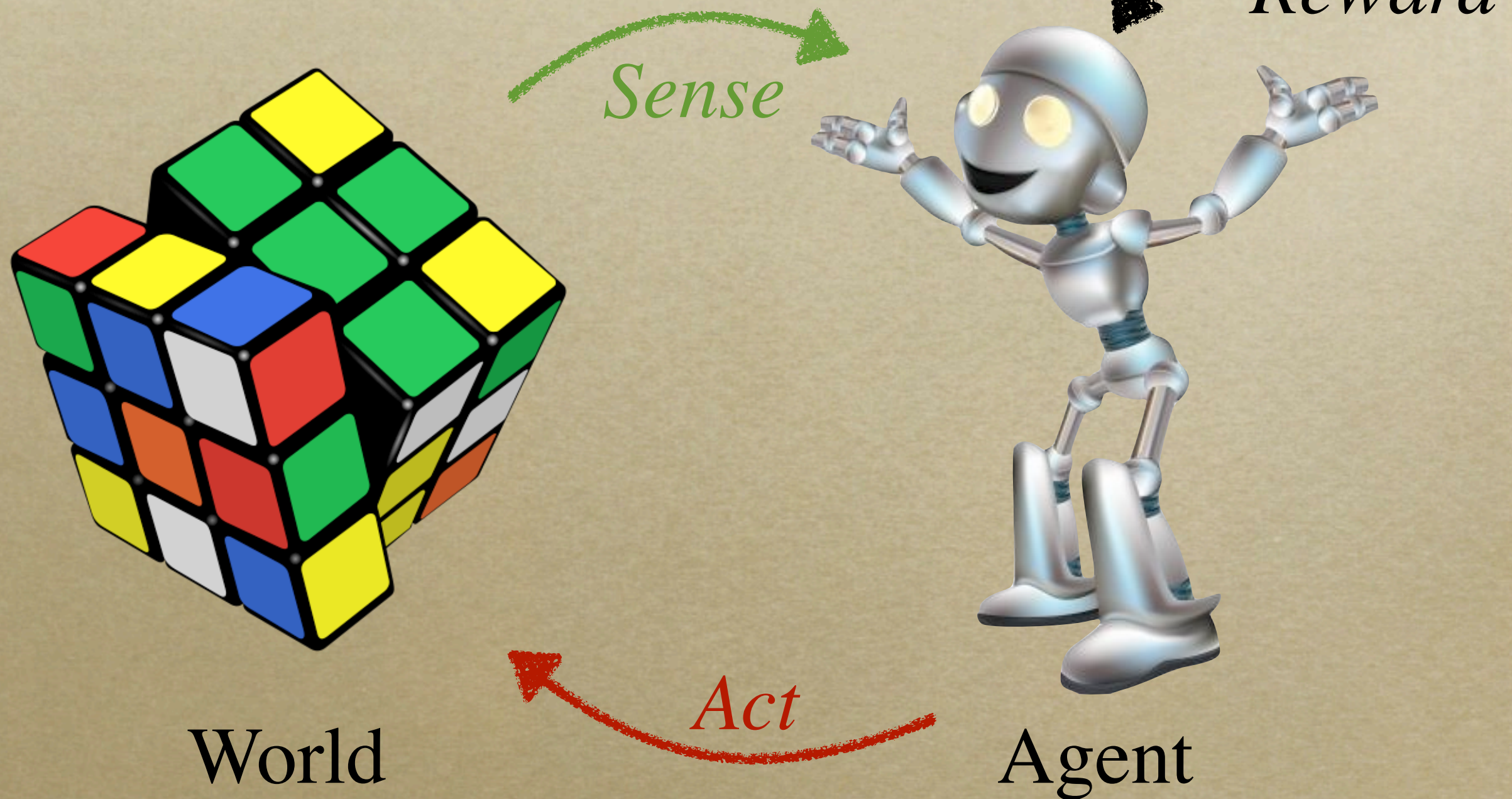
$$T_{s,a}(s') = P(s'|s, a)$$

$$\Omega_s(o) = P(o|s)$$

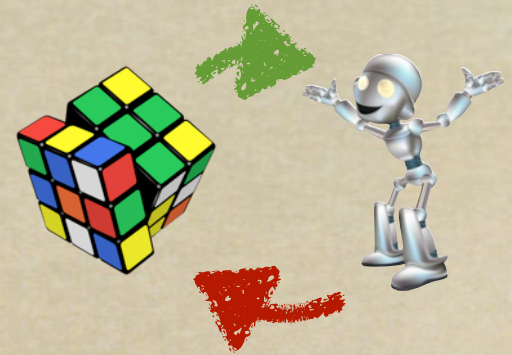
$$R : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$$

$$R(s, a) = E[\text{immediate reward for } s, a]$$

Example: Rubick's Cube



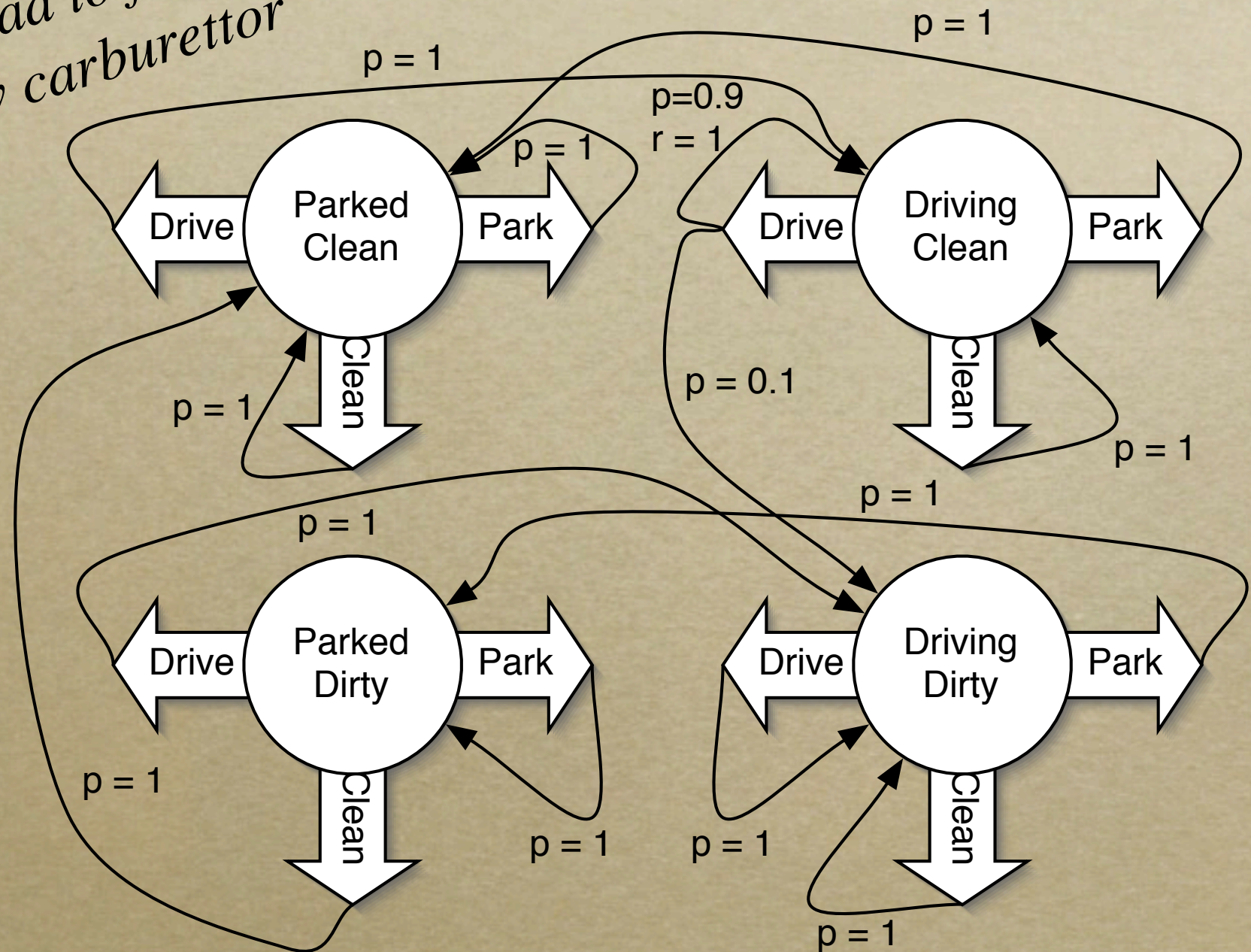
Example: Rubick's Cube



- States: Many - Each particular cube setup
- Actions: 6 - Rotate each face 90°
- Observations: Many - Same as states
- T/Ω Function: Hard to represent compactly (flat)
- R Function:
 - +1 at goal, 0 elsewhere?
 - 0 at goal, -1 elsewhere?

Example 2: Car Driving

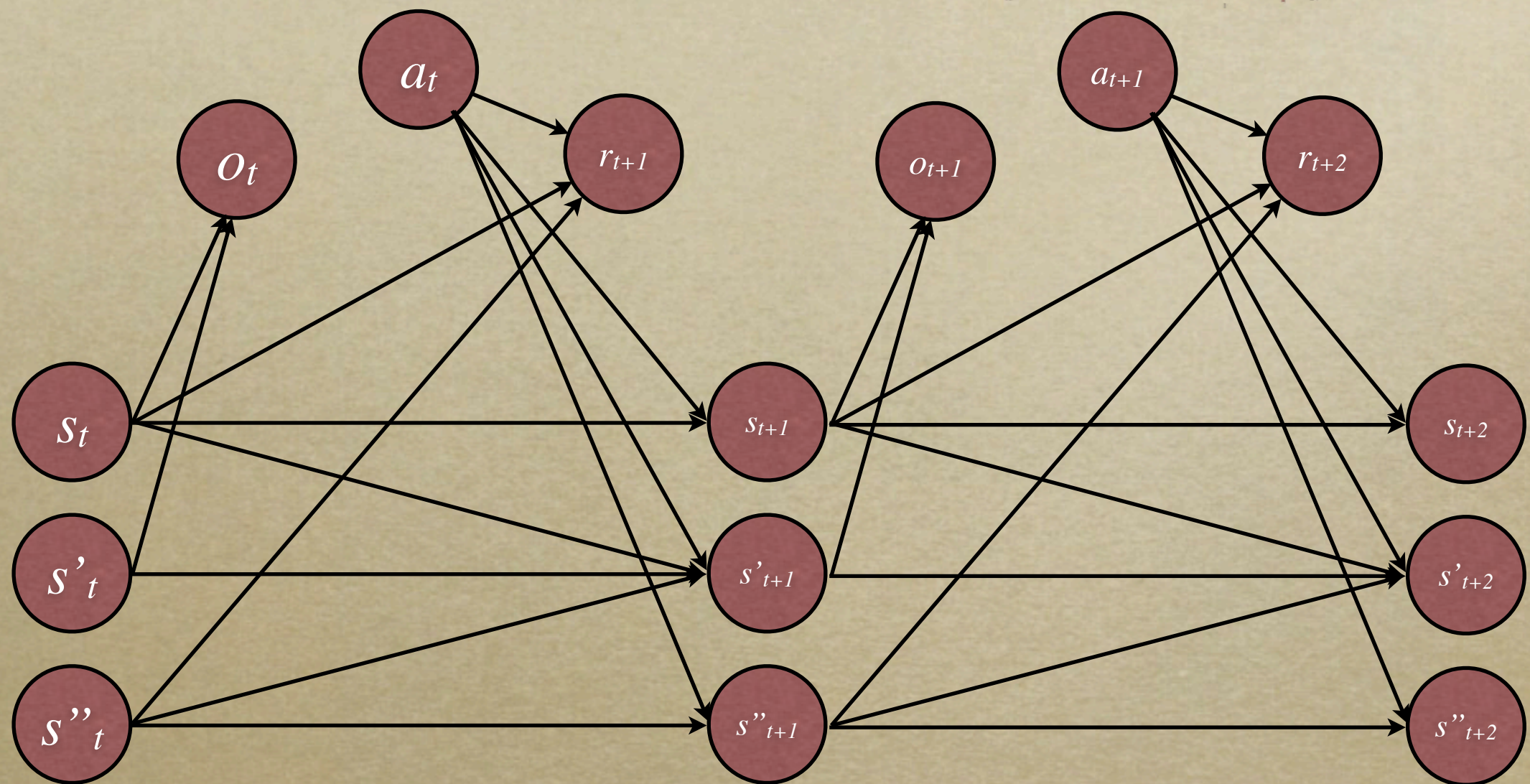
*“Told my girlfriend I had to forget ‘er,
Gotta buy me a new carburettor”*



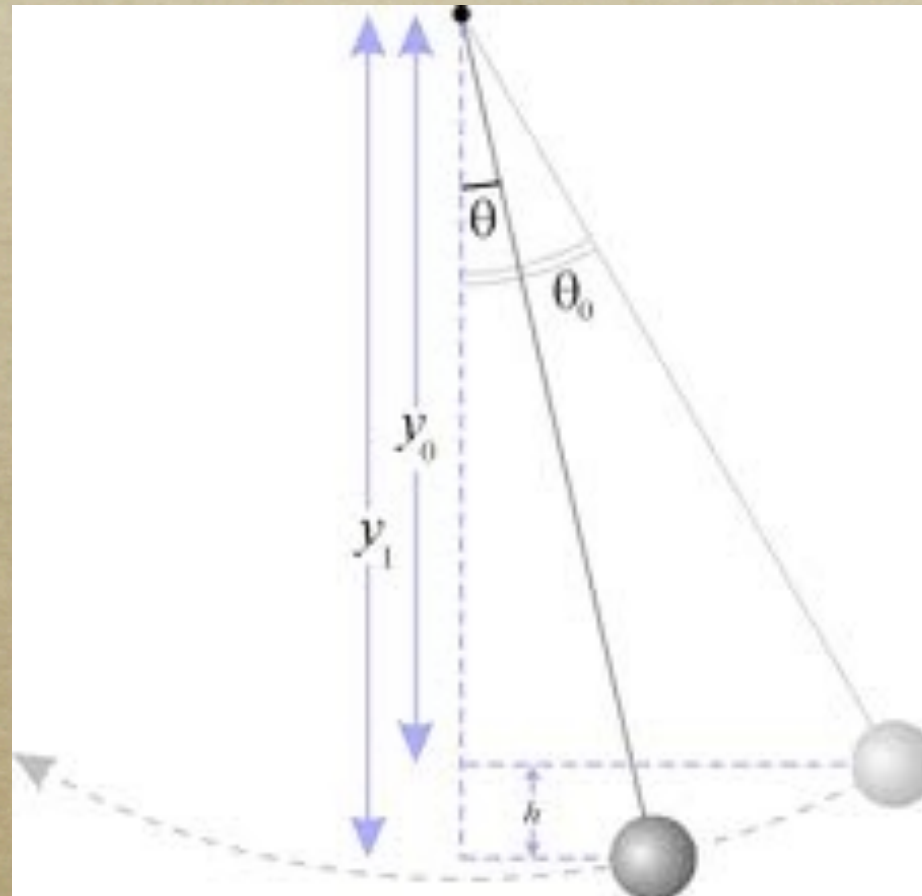
Aside: Markov vs. Fully Obs.

- A 'Markov' state representation makes the future behaviour of the system conditionally independent of the past given the current state
 - e.g. Ω , T and R are functions of s_t , not s_{t-1} or earlier
- Non-Markov state spaces and Partially Observable spaces are related and often confused

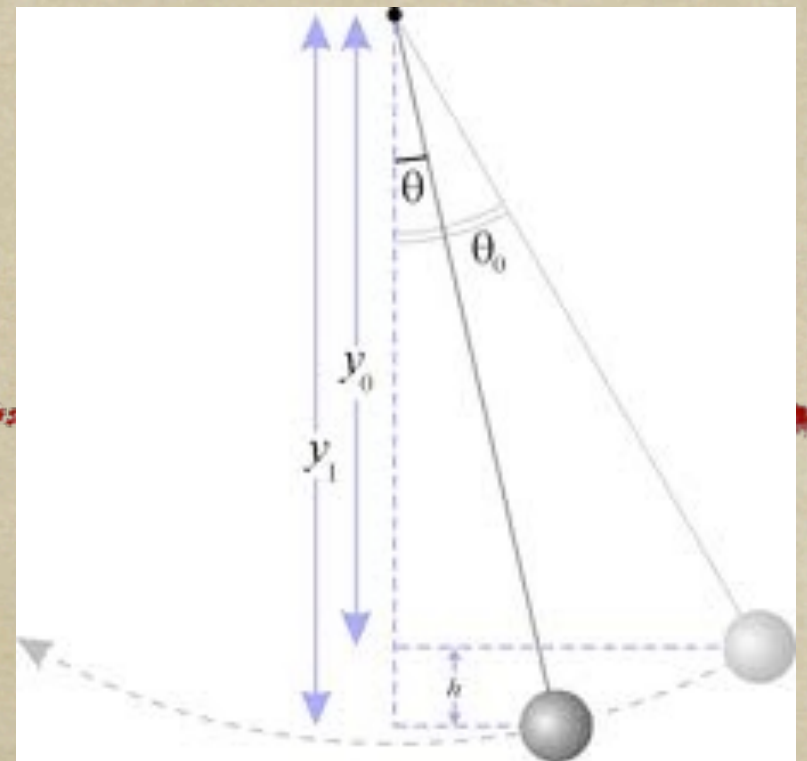
As a graphical model



Example: Pendulum

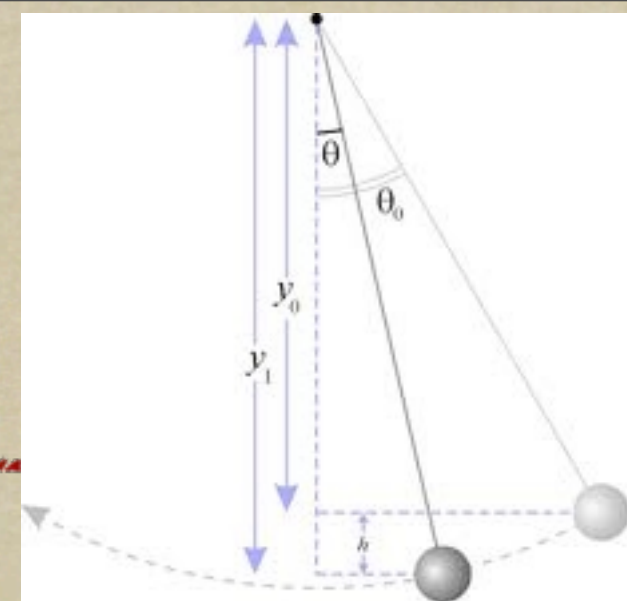


Example: Pendulum



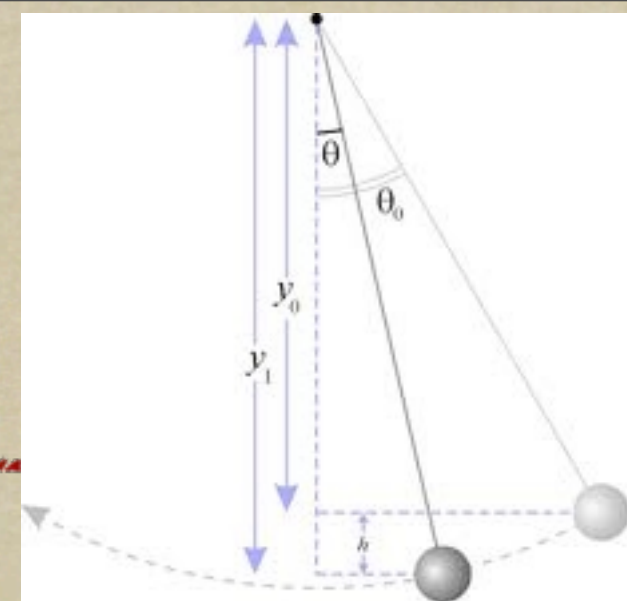
<i>Non-Markov</i>	<i>Markov</i>
<i>State: θ</i>	<i>State: $\theta, \dot{\theta}$</i>

Example: Pendulum



	<i>Non-Markov</i>	<i>Markov</i>
<i>Fully Observable</i>	<i>State:</i> θ <i>Obs:</i> θ	<i>State:</i> $\theta, \dot{\theta}$ <i>Obs:</i> $\theta, \dot{\theta}$
<i>Partially Observable</i>	<i>State:</i> θ <i>Obs:</i> $\theta > 0?$	<i>State:</i> $\theta, \dot{\theta}$ <i>Obs:</i> θ

Example: Pendulum



	<i>Non-Markov</i>	<i>Markov</i>	<i>n-Markov</i>
<i>Fully Observable</i>	State: θ Obs: θ	State: $\theta, \dot{\theta}$ Obs: $\theta, \dot{\theta}$	State: θ_{t-1}, θ_t Obs: θ_{t-1}, θ_t
<i>Partially Observable</i>	State: θ Obs: $\theta > 0?$	State: $\theta, \dot{\theta}$ Obs: θ	State: θ_{t-1}, θ_t Obs: θ_t

What do we need to learn?

- Assume that S , O , A and time are discrete
- Assume sensor and actuator designers tell us O and A
- Do we know S (the size (and structure) of the world)?
- Do we know T and Ω (the way the world behaves)?
- Do we know R (the goal we're trying to achieve)?
- Is Ω invertible? (i.e. Can we infer the state from a single observation, making the world 'fully observable'?)

Classes of Problems

	<i>Fully Observable</i>	<i>Partially Observable</i>
<i>Everything known & Deterministic</i>	<i>Traditional Planning</i>	
<i>Complete model known</i>	<i>MDP Solving / Stochastic Planning</i>	<i>POMDP Solving / Planning</i>
<i>S,A,O known, T, R partial</i>	<i>Traditional Reinforcement Learning</i>	<i>SLAM</i>
<i>S,A,O known, T, R unknown</i>		<i>POMDP Learning</i>
<i>A, O known, S, T, R unknown</i>	<i>Universal Artificial Intelligence</i>	

Reinforcement Learning

- S, A, O (or supersets) known
- Ω known and invertible ('Fully Observable')
- T, R Unknown
- And we want to know how to act!

BUT

Let's think about the 'acting' without the
learning first...

Stochastic Planning

Stochastic Planning

- Everything is known
- Ω is invertible (i.e. ‘Fully Observable’)
- Want to choose actions to maximise ‘long term reward’
 - but how do we formalise that...

Policies

- In general use entire history[†] to decide the next action:

$$\pi : (\mathcal{O} \times \mathcal{A})^* \times \mathcal{O} \rightarrow \mathcal{A}$$

- In the fully observable case, use the state:

$$\pi : \mathcal{S} \rightarrow \mathcal{A}$$

[†] There are ways to compress the information in the agent's history - I'm not going to talk about them.

e.g. GridWorld Policy

→	→	→	→	→	→	→	→	→	😊
→	→	↓	↑	←	↑	→	↓	↑	↑
→	←	↑	↓	↑	←	↓	↑	←	↑
↑	↓	←	↑	→	↑	↑	→	↓	↑
←	→	↓	→	↓	→	↓	↑	→	↑
↑	←	→	→	↑	↑	←	→	↑	↑
←	↑	↑	↑	←	↓	→	←	↑	↑
→	←	→	↓	↓	↑	→	↓	→	↑
←	→	↑	→	↓	→	↑	→	←	↑
→	→	←	↑	←	↑	→	↑	→	↑

Optimality Criteria

- A good policy “maximises long term reward”...
 - What do we mean by that?

Infinite Horizon, Undiscounted

$$\text{Goodness} = \sum_{t=t_0}^{\infty} r_t$$

$$1 + 1 + 1 + \dots < 2 + 2 + 2 + \dots$$

$$\infty < \infty \quad ?!?$$

Only usable if you can guarantee termination

Note: for ease of notation, I'm dropping expectation symbols

Fixed Finite Horizon

$$\text{goodness} = \sum_{t=t_0}^n r_t$$

- No longer diverges
- Only plans until time n , after that, anything could happen

Finite Receding Horizon

$$\text{goodness} = \sum_{t=t_0}^{t_0+n} r_t$$

- Always plans n steps into the future

Discounted Sum of Rewards



*Most
Common*

$$\text{goodness} = \sum_{t=t_0}^{\infty} \gamma^t r_t$$

- No longer a sharp cliff in lookahead
- Justifications for γ :
 - $(1-\gamma)$ chance of dying each step
 - $(1-\gamma)$ inflation rate
 - It's just a mathematical trick

Average Reward

$$\text{goodness} = \lim_{n \rightarrow \infty} \frac{1}{n+1} \sum_{t=t_0}^{t_0+n} r_t$$

- Average reward in limit as range of average grows
- A.K.A. Gain
- Related to Sensitive Discount Optimality:

$$\lim_{\gamma \rightarrow 1} \operatorname{argmax}_{\pi} \sum_{t=t_0}^{\infty} \gamma^t r_t$$

Average Reward Laziness

- As long as the long run is the same, why optimise the short term?
- Often a second ‘Bias’ optimality criterion is introduced
- There is an infinite sequence of increasingly stringent optimality criteria that can be applied

What people use

- Average reward is theoretically nice, but practically difficult
- Undiscounted systems are used with known fixed horizons
- Discounted reward is the standard

State-Value Functions

- Define the ‘value’ of each state as the expected sum discounted reward for starting in that state and following the given policy for n steps:

$$\begin{aligned} V_n^\pi(s) &= \sum_{t=0}^n \gamma^t r_t \\ &= r_0 + \gamma^1 r_1 + \gamma^2 r_2 + \dots \\ &= r_0 + \gamma[r_1 + \gamma[r_2 + \dots \end{aligned}$$

Recursive Formulation (Bellman)

$$\begin{aligned} V_n^\pi(s) &= \sum_{t=0}^n \gamma^t r_t \\ &= r_0 + \gamma[r_1 + \gamma[r_2 + \dots \\ &= R(s, \pi(s)) + \gamma \mathbf{E}_{s' \in \mathcal{S}} V_{n-1}^\pi(s') \\ &= R(s, \pi(s)) + \gamma \sum_{s' \in \mathcal{S}} T_{s, \pi(s)}(s') V_{n-1}^\pi(s'), \text{ where} \end{aligned}$$

$$V_0^\pi(s) = 0$$

$$V^\pi(s) = \lim_{n \rightarrow \infty} V_n^\pi(s)$$

State-Action Formulation

- The State-Action value function defines the expected sum of discounted reward of choosing action a in state s once, and following the given policy from then onwards:

$$\begin{aligned} Q_n^\pi(s, a) &= R(s, a) + \gamma \mathbb{E}_{s' \in \mathcal{S}} V_{n-1}^\pi(s') \\ &= R(s, a) + \gamma \sum_{s' \in \mathcal{S}} T_{s,a}(s') V_{n-1}^\pi(s'), \text{ where} \end{aligned}$$

$$Q_0^\pi(s, a) = 0$$

$$V_n^\pi(s) = Q_n^\pi(s, \pi(s))$$

Value Determination

- For a given policy, V and Q are linear

$$V^\pi(s) = R(s, \pi(s)) + \gamma \sum_{s' \in \mathcal{S}} T_{s, \pi(s)}(s') V^\pi(s')$$

$$\vec{R} = (I - \gamma T) \vec{V}$$

$$Q^\pi(s, a) = R(s, a) + \gamma \sum_{s' \in \mathcal{S}} T_{s, a}(s') Q^\pi(s', \pi(s'))$$

*Fixed point can be found using
linear algebra / matrix inversion.*

Greedy Policies

- Given a Q function, or V, T, R functions find a policy:

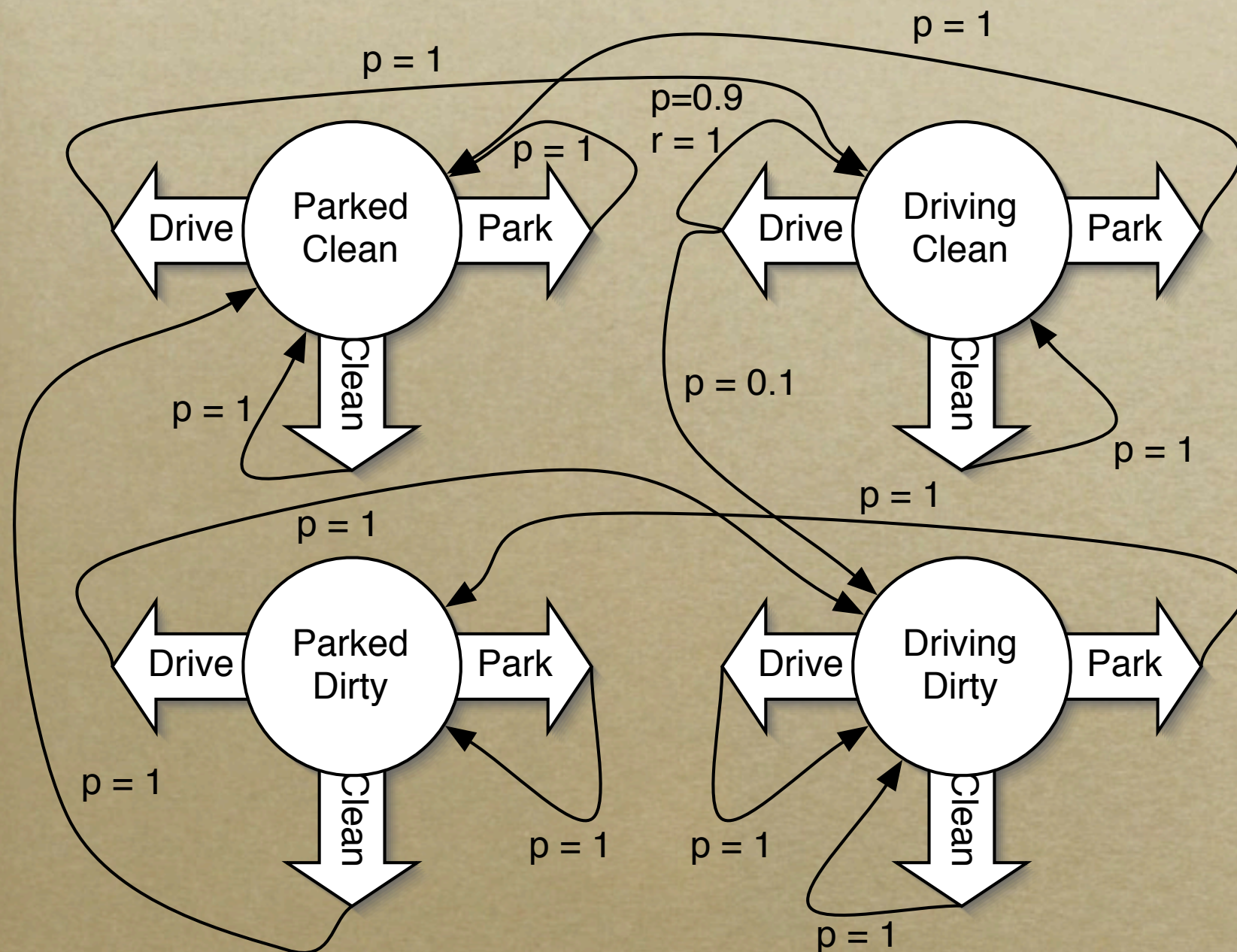
$$\pi(s) = \operatorname{argmax}_a Q(s, a), \text{ or}$$

$$\pi(s) = \operatorname{argmax}_a R(s, a) + \gamma \sum_{s' \in \mathcal{S}} T_{s,a}(s') V(s')$$

Policy Iteration

- Choose any initial policy
- Loop
 - Find the policy's value function
 - Find the greedy policy for that value function

Car Driver Example



		<i>s</i>			
		<i>PC</i>	<i>DC</i>	<i>PD</i>	<i>DD</i>
<i>s'</i>	<i>PC</i>	0	0	0	0
	<i>DC</i>	1	0.9	0	0
	<i>PD</i>	0	0	0	0
	<i>DD</i>	0	0.1	1	1

		<i>s</i>			
		<i>PC</i>	<i>DC</i>	<i>PD</i>	<i>DD</i>
<i>s'</i>	<i>PC</i>	1	1	0	0
	<i>DC</i>	0	0	0	0
	<i>PD</i>	0	0	1	1
	<i>DD</i>	0	0	0	0

		<i>s</i>			
		<i>PC</i>	<i>DC</i>	<i>PD</i>	<i>DD</i>
<i>s'</i>	<i>PC</i>	1	0	1	0
	<i>DC</i>	0	1	0	0
	<i>PD</i>	0	0	0	0
	<i>DD</i>	0	0	0	1

Policy Iteration Example

<i>PC</i>	<i>D</i>
<i>DC</i>	<i>D</i>
<i>PD</i>	<i>D</i>
<i>DD</i>	<i>D</i>

	<i>PC</i>	<i>DC</i>	<i>PD</i>	<i>DD</i>
<i>D</i>	4.7	5.3	0	0
<i>P</i>	4.3	4.3	0	0
<i>C</i>	4.3	4.7	4.3	0

<i>PC</i>	<i>D</i>
<i>DC</i>	<i>D</i>
<i>PD</i>	<i>C</i>
<i>DD</i>	<i>D</i>

	<i>PC</i>	<i>DC</i>	<i>PD</i>	<i>DD</i>
<i>D</i>	4.7	5.3	0	0
<i>P</i>	4.3	4.3	3.8	3.8
<i>C</i>	4.3	4.7	4.3	0

<i>PC</i>	<i>D</i>
<i>DC</i>	<i>D</i>
<i>PD</i>	<i>C</i>
<i>DD</i>	<i>P</i>

	<i>PC</i>	<i>DC</i>	<i>PD</i>	<i>DD</i>
<i>D</i>	7.2	8.0	5.3	5.3
<i>P</i>	6.5	6.5	5.9	5.9
<i>C</i>	6.5	7.2	6.5	5.3

Update order irrelevant

- Repeat often enough for all s , a each of:

$$Q(s, a) := R(s, a) + \gamma \sum_{s' \in \mathcal{S}} T_{s,a}(s') V(s')$$

$$V(s) := Q(s, \pi(s))$$

$$\pi(s) := \operatorname{argmax}_a Q(s, a)$$

Prioritised Sweeping

- Use a Priority Queue to update nodes in a smart order
- On each update, track $|\Delta V|$, and add it to the priority of all upstream states
- Reset priority to 0 when a node is pulled from the queue and updated



Break Time

Standard Techniques

- But what about the Learning!?!
 - Exploration/Exploitation
 - Model based vs Model free techniques
 - On-Policy vs Off Policy techniques
- Other solution methods
 - State space and Plan space searches
- Combinations
 - TD(λ) and VaPs

Learning a Model

- Move about the world
 - Record $\langle s, a, s', r \rangle$ for each transition
- Finding T, R given some data is an incremental supervised learning problem
 - Want distribution, not classification
- Use what you learn from others this week

e.g. Driving Clean Car

- Driving Clean, Drive, Driving Clean, Drive, Driving Dirty, Park, Parked Dirty, Clean, Parked Clean, Drive, Driving Clean
- Driving Clean, Drive \Rightarrow Driving Clean
- Driving Clean, Drive \Rightarrow Driving Dirty
- Driving Dirty, Park \Rightarrow Parked Dirty
- Parked Dirty, Clean \Rightarrow Parked Clean
- Parked Clean, Drive \Rightarrow Driving Clean

Exploration/Exploitation

- You're an agent, and you've explored the world a little bit, how do you act?
- Choose the action that is best given what you know?
- Choose an action that will help you learn more?

Example: n-armed bandit

- You have a row of n poker machines
- Each machine is free to play, and has a stationary individual distribution over payouts for a single game
- How do you act so as to maximise the amount of money won over time?

Exploration/Exploitation

- ϵ -greedy: Exploit ϵ , Random $(1-\epsilon)$
- Boltzmann: $P(a|s) = \frac{e^{\frac{Q(s,a)}{\tau}}}{\sum_{a' \in \mathcal{A}} e^{\frac{Q(s,a')}{\tau}}}$
- Optimistic prior
- Fully Bayesian Solution

Model Free Learning

Stochastic Gradient Descent

$$Q(s, a) := R(s, a) + \gamma \sum_{s' \in \mathcal{S}} T_{s,a}(s') V(s')$$

$$Q(s, a) := \mathbb{E}(r) + \gamma \mathbb{E}_{s' \in \mathcal{S}} V(s')$$

$$Q(s, a) \rightsquigarrow [r + \gamma V(s')]$$

$$Q(s, a) := (1 - \alpha) Q(s, a) + \alpha [r + \gamma V(s')]$$

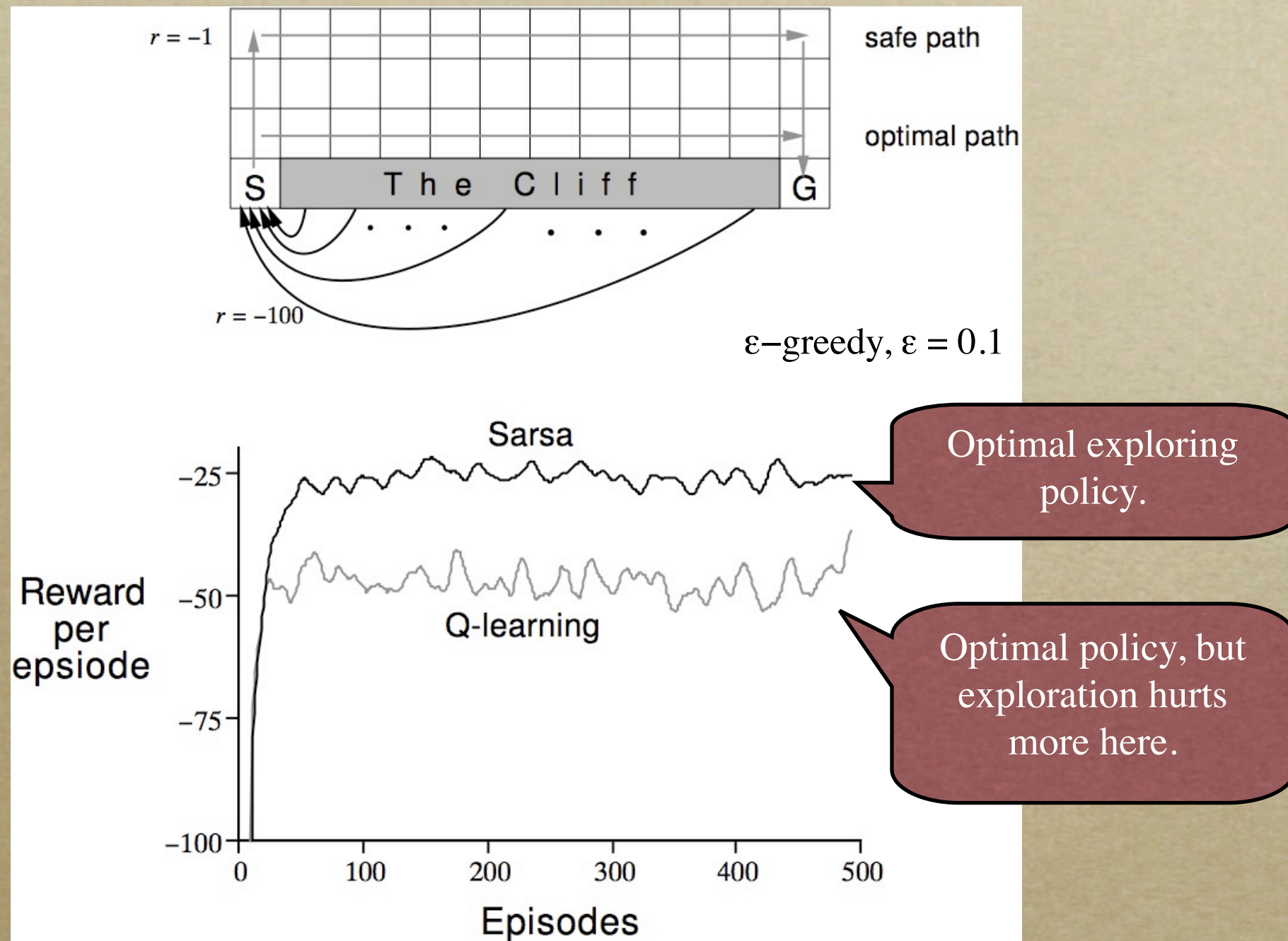
$$Q(s, a) := Q(s, a) + \alpha [(r + \gamma V(s')) - Q(s, a)]$$

On-policy vs Off-policy learning

- Q-Learning: Learns the optimal policy regardless of where samples start
- What if we want to learn the value of a particular policy?
- SARSA: (learn from $\langle s, a, r, s', a' \rangle$)

$$Q(s, a) := (1 - \alpha)Q(s, a) + \alpha [r + \gamma Q(s', a')]$$

Cliffwalking



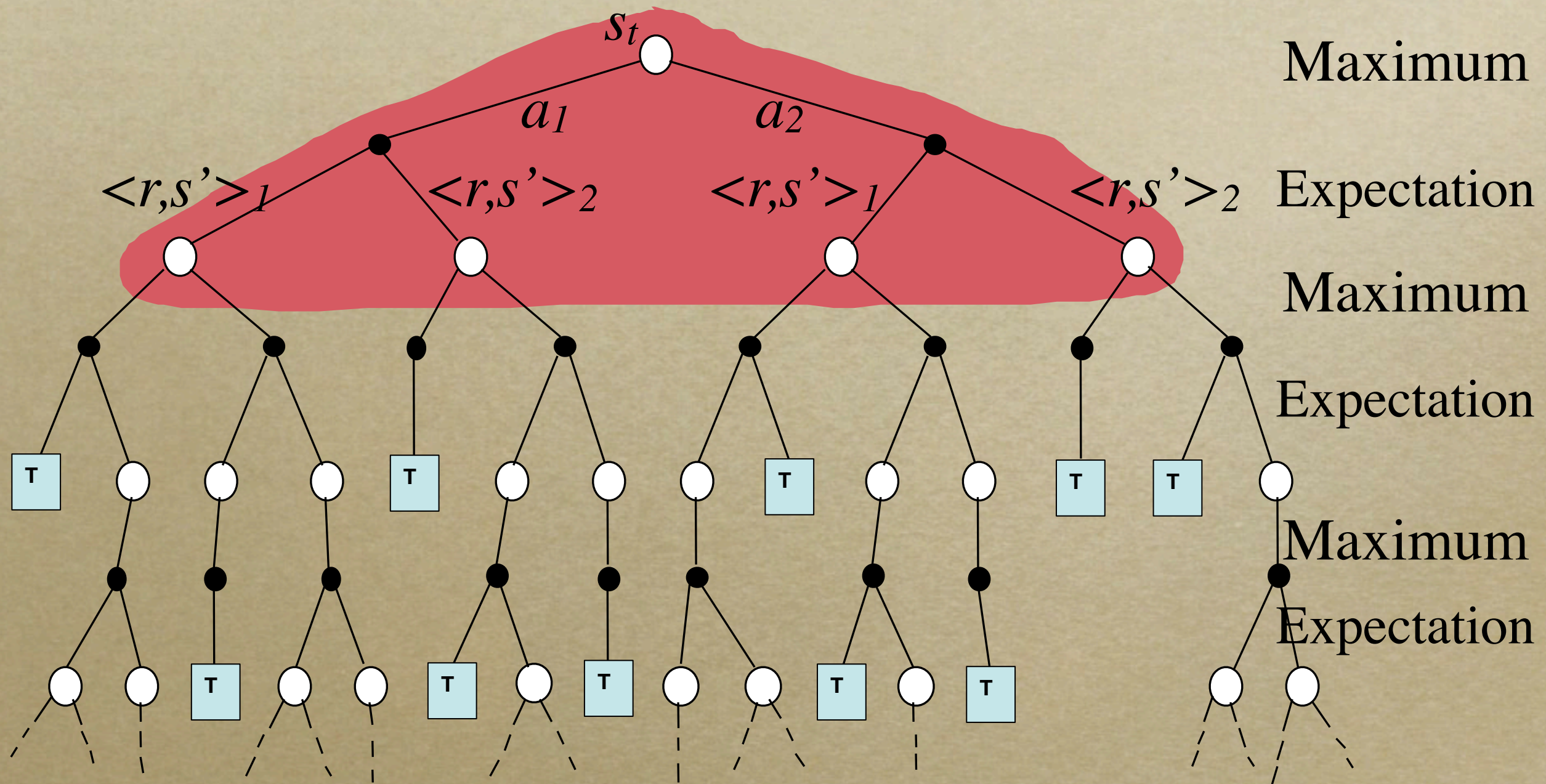
State Space vs. Plan Space

- Both map to tree (graph) search
- They assign different meanings to the nodes and edges

	<i>State Space</i>	<i>Plan Space</i>
<i>Node</i>	<i>State</i>	<i>(Partial) Plan / Policy</i>
<i>Edge</i>	<i>Action/State transition</i>	<i>Plan Alteration</i>
<i>Goal</i>	<i>'Path' through tree</i>	<i>Good node in tree</i>

Stochastic State Space Search

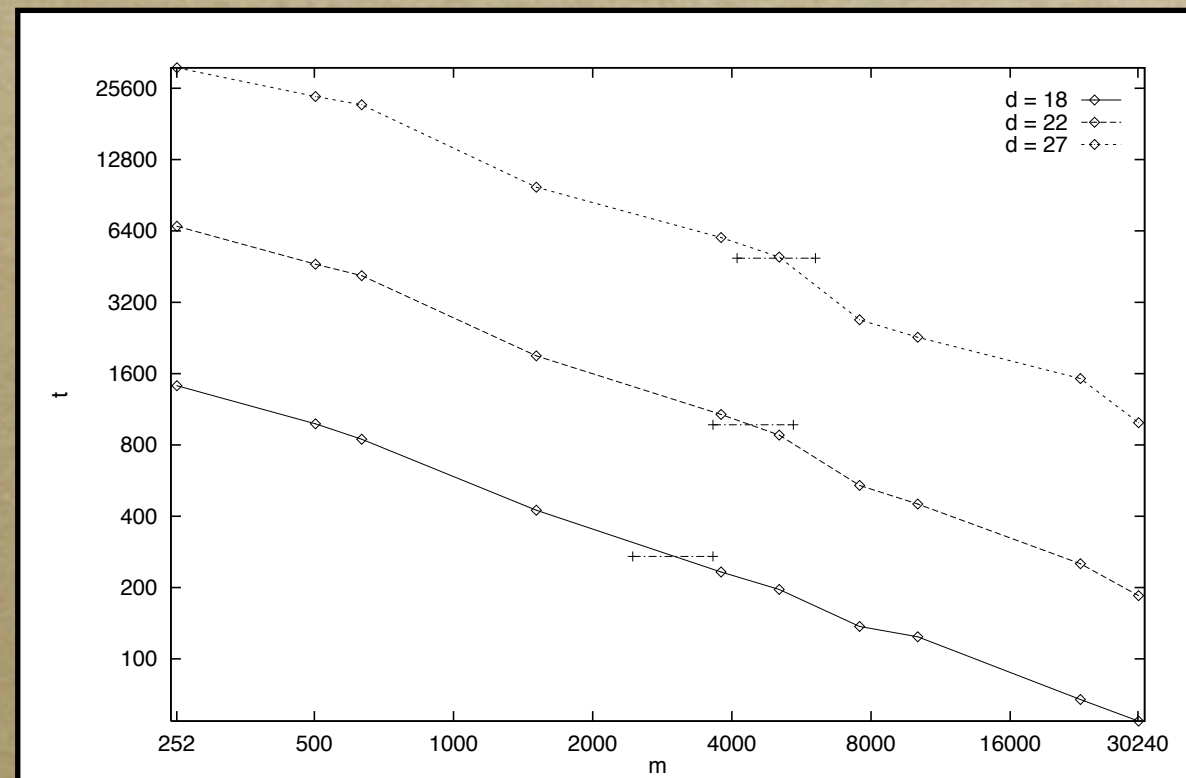
Expectimax Search



Similar to Dynamic Programming, but forwards

Memory Search Tradeoff

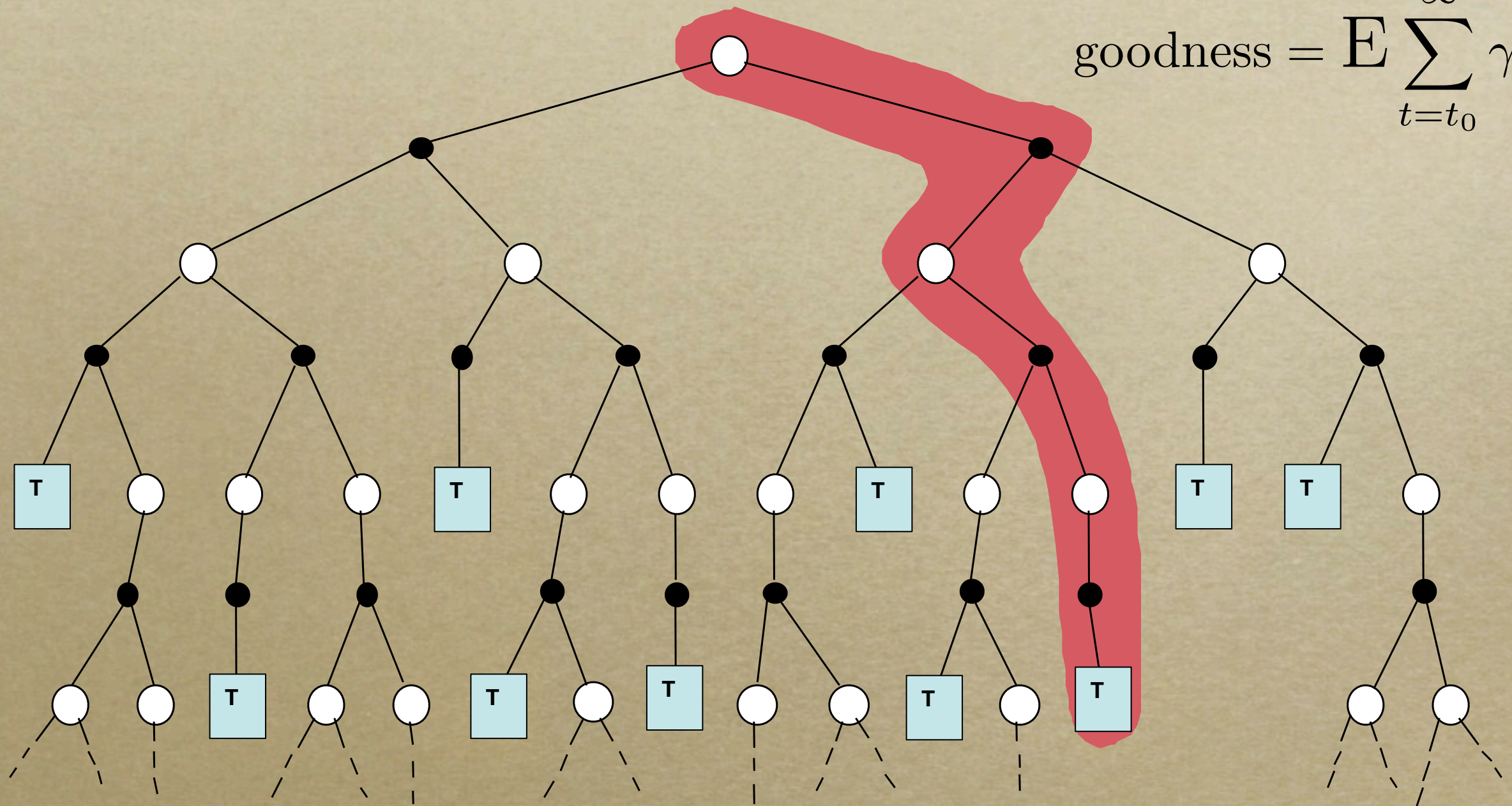
- Find $V(s)$, then store a reduced version: index by only the top n bits of the state
- Use A^* in a deterministic domain, and see how much search is needed



Monte-Carlo State Space Search

UCT

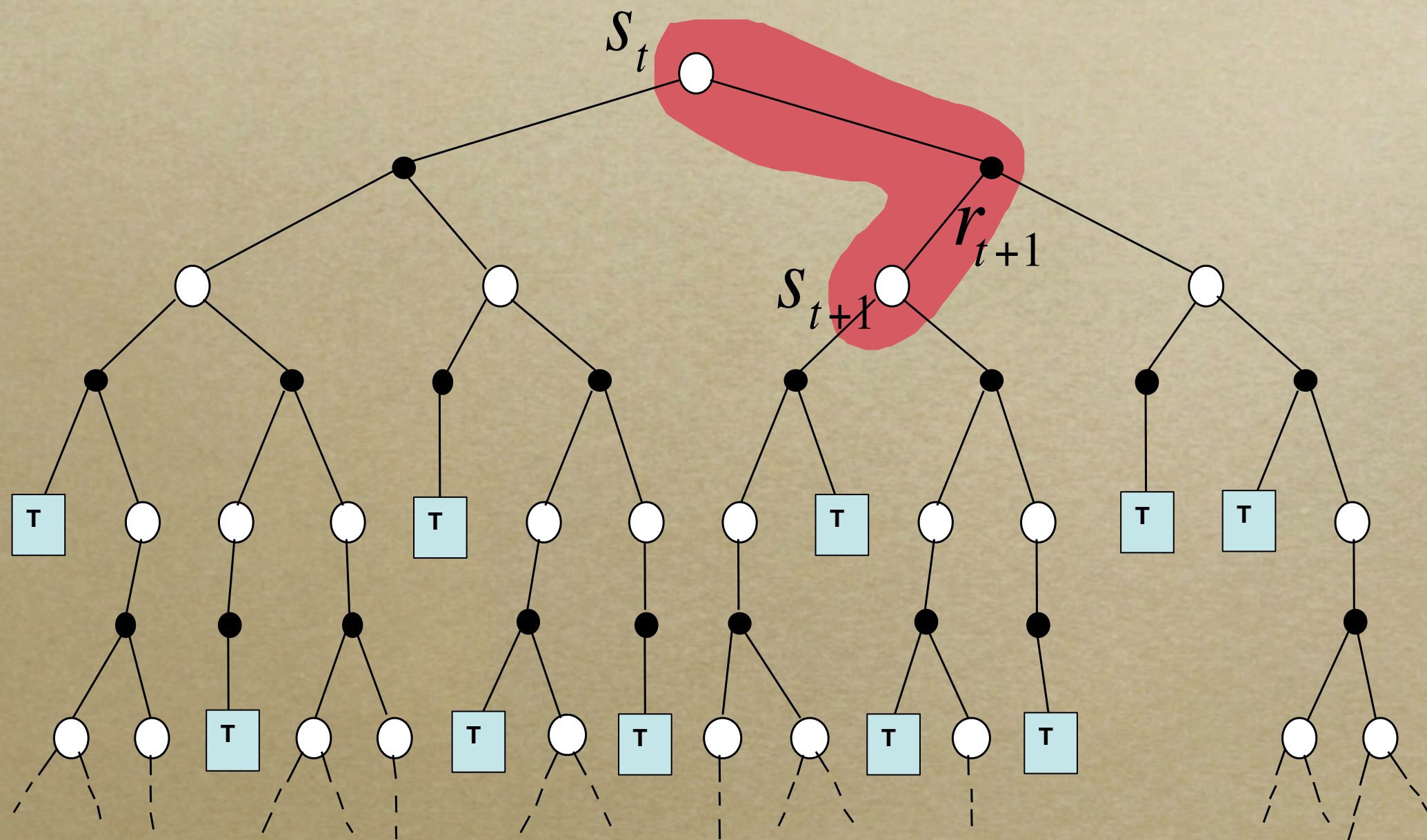
$$\text{goodness} = \mathbb{E} \sum_{t=t_0}^{\infty} \gamma^t r_t$$



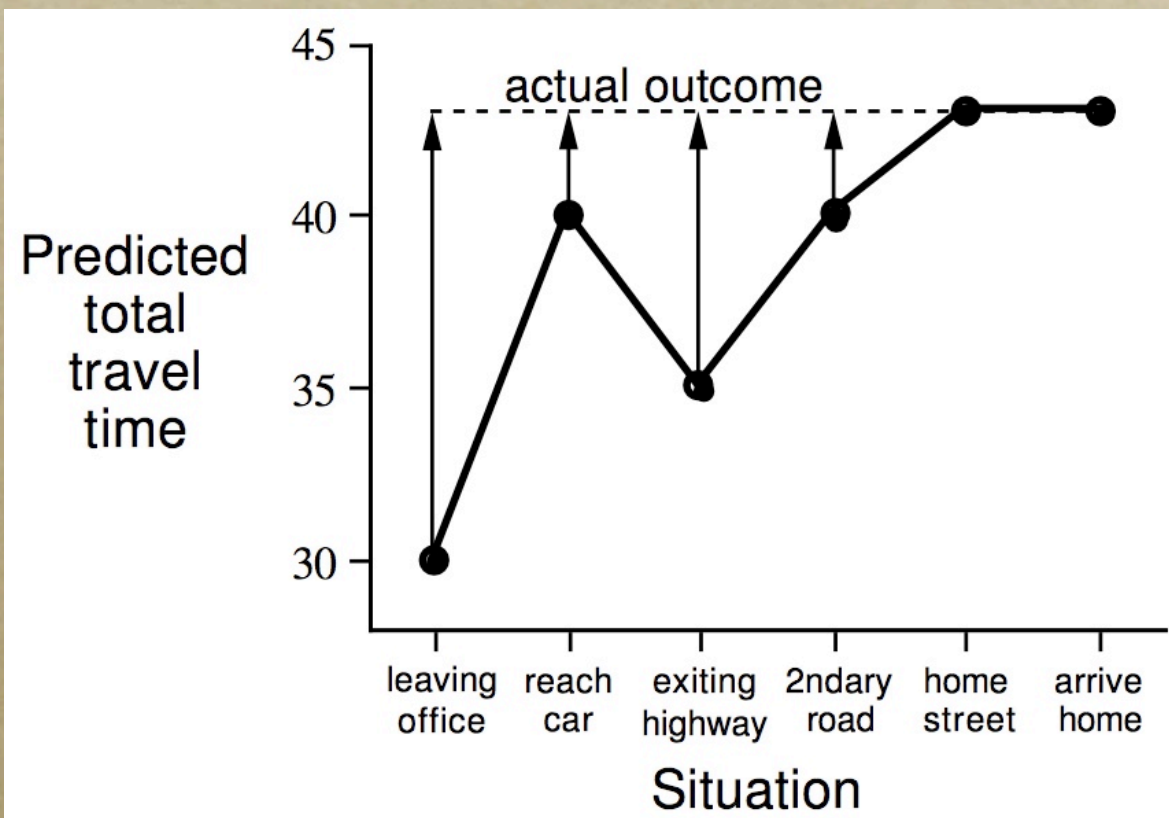
Use Monte-Carlo methods to estimate a value function

Temporal Differencing Update

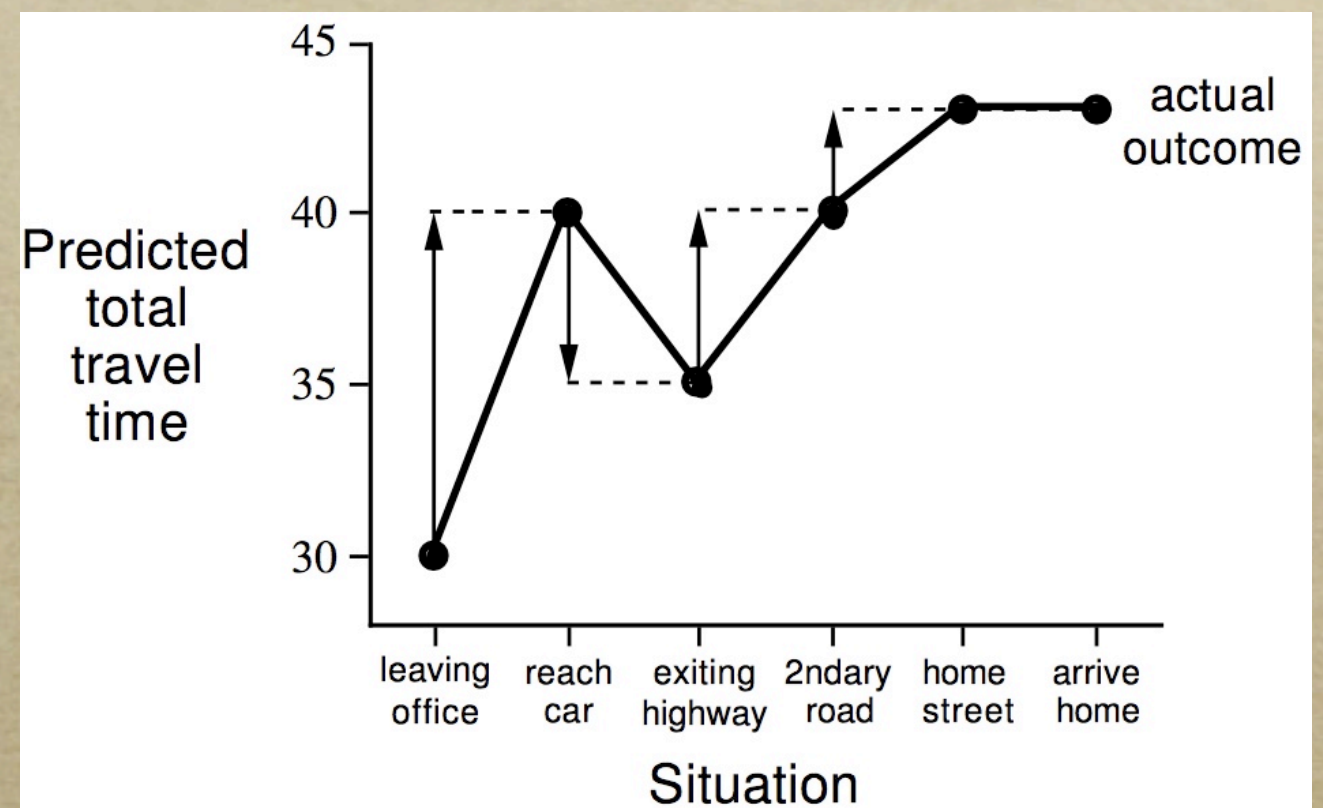
$$V(s_t) \leftarrow V(s_t) + \alpha [(r_{t+1} + \gamma V(s_{t+1})) - V(s_t)]$$



Monte-Carlo vs Temporal Diff.

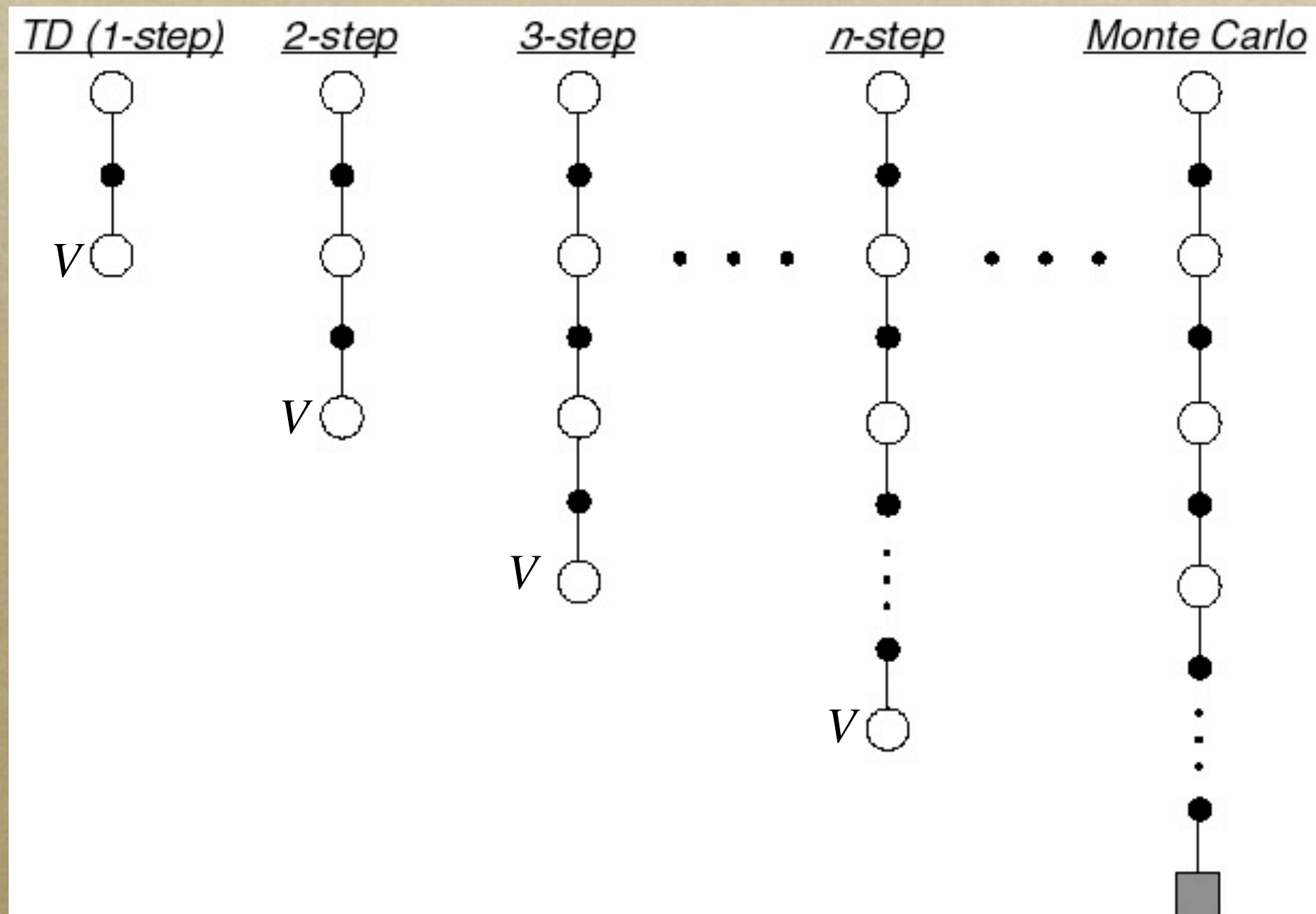


Monte-Carlo: Move towards final estimate



TD: Move towards next state estimate

Merging Monte-Carlo and TD



Forward View of TD(λ)

- TD(λ) is a method for averaging all n-step backups

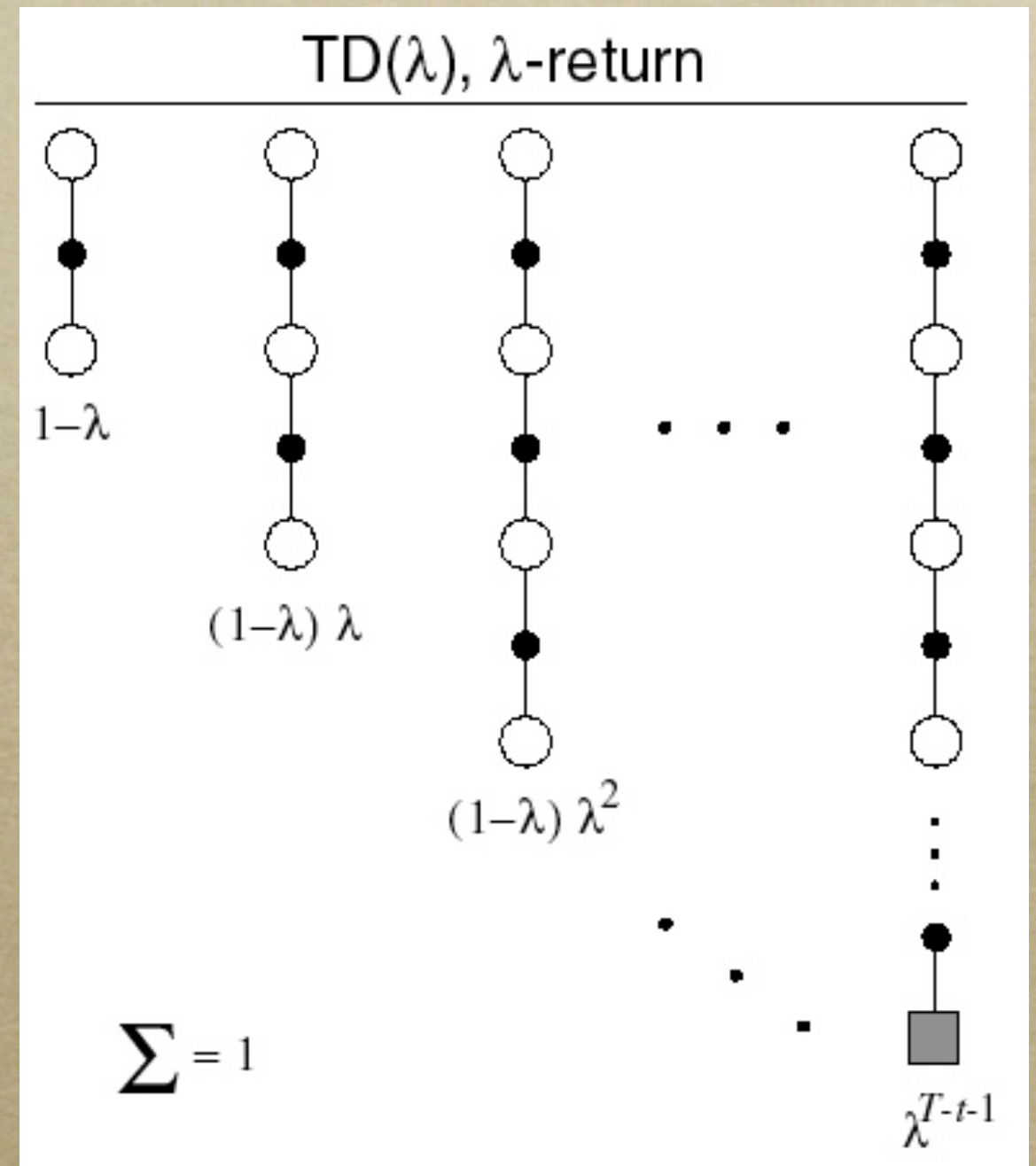
- weight by λ^{n-1}
(time since visitation)

– λ -return:

$$R_t^\lambda = (1 - \lambda) \sum_{n=1}^{\infty} \lambda^{n-1} R_t^{(n)}$$

- Backup using λ -return:

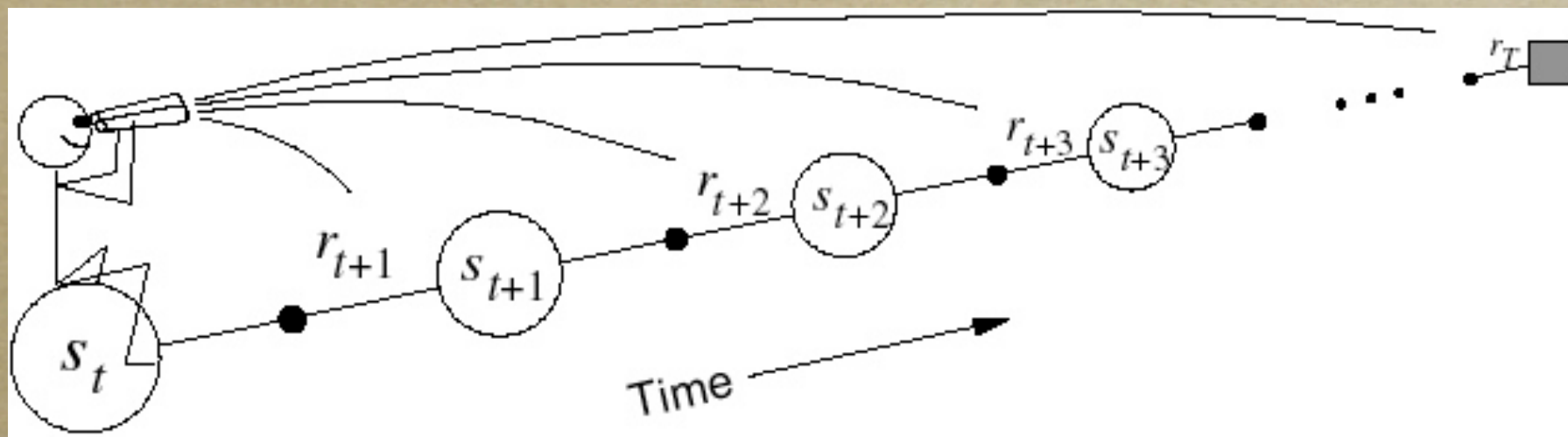
$$\Delta V_t(s_t) = \alpha [R_t^\lambda - V_t(s_t)]$$



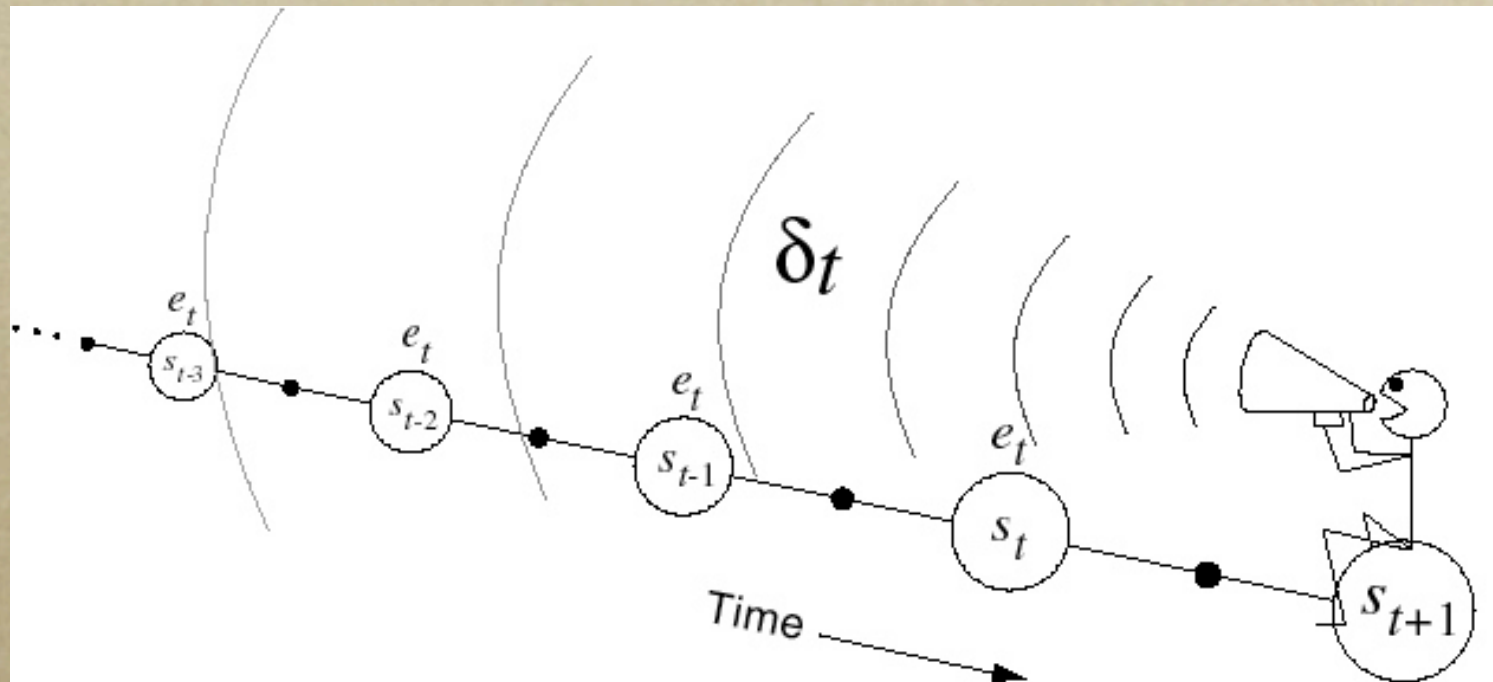
What happens when $\lambda=1$, $\lambda=0$?

Forward View of TD(λ)

- Look forward from each state to determine update from future states and rewards:



Backward View



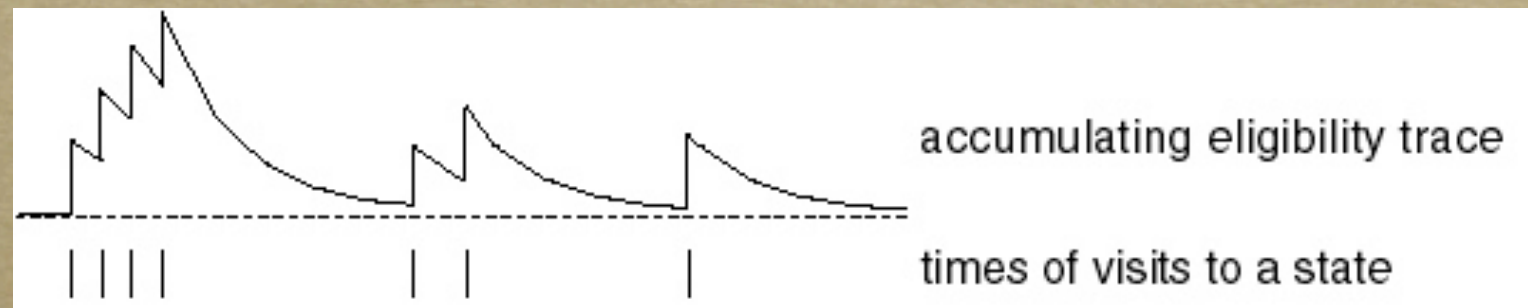
$$\delta_t = r_{t+1} + \gamma V_t(s_{t+1}) - V_t(s_t)$$

- Shout δ_t backwards over time
- The strength of your voice decreases with temporal distance by $\gamma\lambda$

Backward View of TD(λ)

- The forward view was for theory
- The backward view is for mechanism
- New variable called *eligibility trace* $e_t(s) \in \mathbb{R}^+$
 - On each step, decay all traces by $\gamma\lambda$ and increment the trace for the current state by 1
 - Accumulating trace

$$e_t(s) = \begin{cases} \gamma\lambda e_{t-1}(s) & \text{if } s \neq s_t \\ \gamma\lambda e_{t-1}(s) + 1 & \text{if } s = s_t \end{cases}$$



On-line Tabular TD(λ)

Initialize $V(s)$ arbitrarily

Repeat (for each episode) :

$e(s) = 0$, for all $s \in S$

Initialize s

Repeat (for each step of episode) :

$a \leftarrow$ action given by π for s

Take action a , observe reward, r , and next state s'

$\delta \leftarrow r + \gamma V(s') - V(s)$

$e(s) \leftarrow e(s) + 1$

For all s :

$V(s) \leftarrow V(s) + \alpha \delta e(s)$

$e(s) \leftarrow \gamma \lambda e(s)$

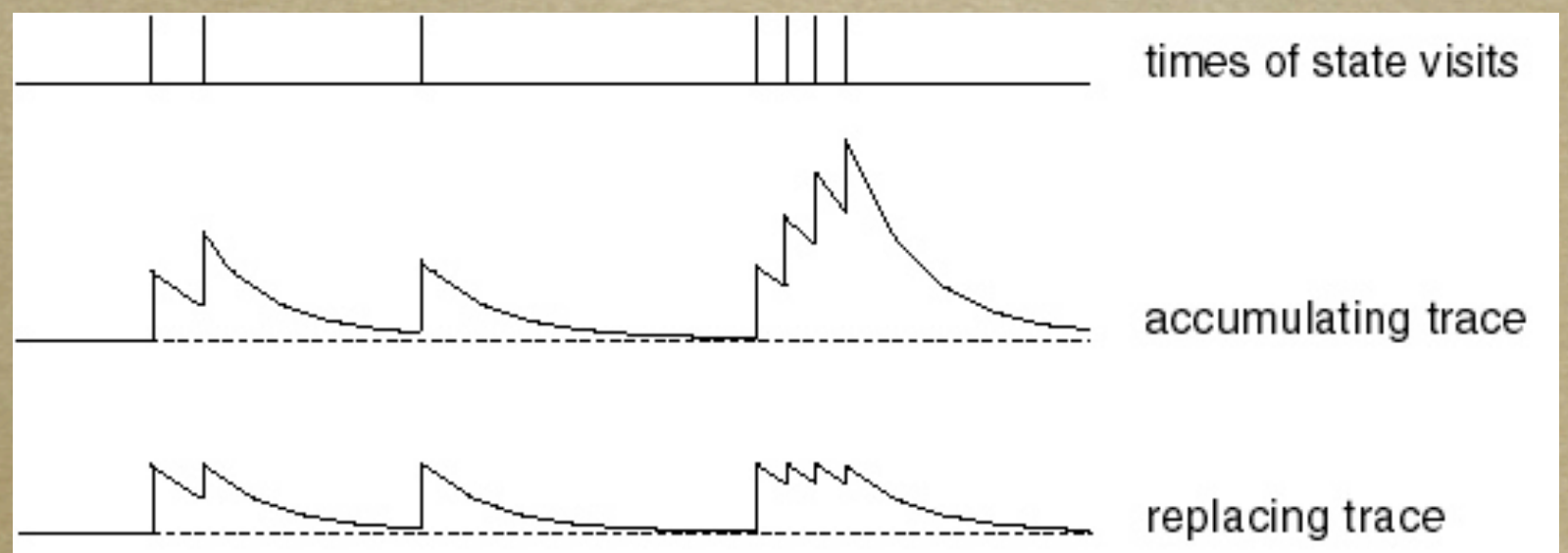
$s \leftarrow s'$

Until s is terminal

Replacing Traces

- Using accumulating traces, frequently visited states can have eligibilities greater than 1
 - This can be a problem for convergence
- *Replacing traces*: Instead of adding 1 when you visit a state, set that trace to 1

$$e_t(s) = \begin{cases} \gamma \lambda e_{t-1}(s) & \text{if } s \neq s_t \\ 1 & \text{if } s = s_t \end{cases}$$



Plan Space Search

- Similar to Monte-Carlo, but only keep one value (root of the tree)
- Because of discounting, can cut off samples at 300-1000 nodes
- Use gradient ascent in policy space to maximise reward

Learning to Walk

- Use corners of foot trajectory to define policy space
 - $3D * 4 \text{ pts} * 2 \text{ (front/back legs)} = 24D$
- Each point in this space defines a behaviour
- Measure behaviour quality walking across the RoboCup field

Learning to Walk



Learning to Walk



Learning to Walk



Policy Search

- Will only find a local optimum
- A differentiable policy can allow gradient based optimisation techniques
 - A stochastic policy is differentiable

$$\pi : S \rightarrow (A \rightarrow [0, 1])$$

VaPS: Value And Policy Search

- TD gives us a way to update a value function for a policy
- If we use Boltzmann exploration, then a value function can also describe a stochastic, differentiable policy:

$$P(a|s) = \frac{e^{\frac{Q(s,a)}{\tau}}}{\sum_{a' \in \mathcal{A}} e^{\frac{Q(s,a')}{\tau}}}$$

- Use weighted combination



Break Time

Function Approximation in RL

- Curse of Dimensionality
 - Forget convergence: too many states to visit them all even once
- Generalise across ‘similar’ states
- Use a representation with fewer free parameters: function approximation

Function Approximation

- $V(s) = f(\Theta, s)$
- Θ represents the parameter vector of the function
- Often want derivative vector of f :

$$\nabla f = \left(\frac{\partial f}{\partial \theta_1}, \dots, \frac{\partial f}{\partial \theta_n} \right)$$

Linear Function Approximation

- Assume we have a set of basis functions on the state:

$$\Phi(s) = \langle \phi_1(s), \dots, \phi_n(s) \rangle$$

- Combine basis functions linearly

$$f(s) = \Theta \cdot \Phi(s)$$

- Derivative is simple

Requirements

- Approach must converge
 - not diverge or oscillate
- Approach should converge to a reasonable solution
 - Generally a local optimum
 - Need one overall measure of ‘goodness’

Note: Just because value determination converges does not mean that you can find a control policy!

Difficulties for convergence

- T & R learned from data
 - Standard Supervised Learning
- TD/recursive methods
 - V defined in terms of V
 - Updates Non-stationary
 - Feedback loops

Optimality Criteria

- Average reward:
 - Optimise the gain
- Discounted reward:
 - Pick a probability distribution over the states
 - Optimise expected V

Types of value update

- Linear Equations

$$V^\pi(s) = R(s, \pi(s)) + \gamma \sum_{s' \in \mathcal{S}} T_{s, \pi(s)}(s') V^\pi(s')$$

- Recursive Assignment

$$V(s, \pi(s)) := R(s, \pi(s)) + \gamma \sum_{s' \in \mathcal{S}} T_{s, \pi(s)}(s') V(s')$$

- Temporal Differencing

$$V(s_t) \leftarrow V(s_t) + \alpha [(r_{t+1} + \gamma V(s_{t+1})) - V(s_t)]$$

- Monte Carlo

$$V(s) = \mathbb{E} \sum_{t=0}^{\infty} \gamma^t r_t$$

Sum Squared Error

$$V^\pi(s) = R(s, \pi(s)) + \gamma \sum_{s' \in \mathcal{S}} T_{s, \pi(s)}(s') V^\pi(s')$$

- Replace equality with min square error

$$Err^2 = \left(V^\pi(s) - \left[R(s, \pi(s)) + \gamma \sum_{s' \in \mathcal{S}} T_{s, \pi(s)}(s') V^\pi(s') \right] \right)^2$$

- Differentiate and use a gradient method

Monte-Carlo

$$V(s) = \mathbb{E} \sum_{t=0}^{\infty} \gamma^t r_t$$

- Already covered this with policy gradient methods
- Converges to a local optimum

Approximate TD Methods

$$V(s_t) + = \alpha [(r_{t+1} + \gamma V(s_{t+1})) - V(s_t)]$$

- Long history of convergence issues
- Still the method that everyone wants to use
- Convert “ $+ = \alpha \text{ err}$ ” to gradient delta

$$\Theta + = \alpha \nabla V(s) [(r_{t+1} + \gamma V(s_{t+1})) - V(s_t)]$$

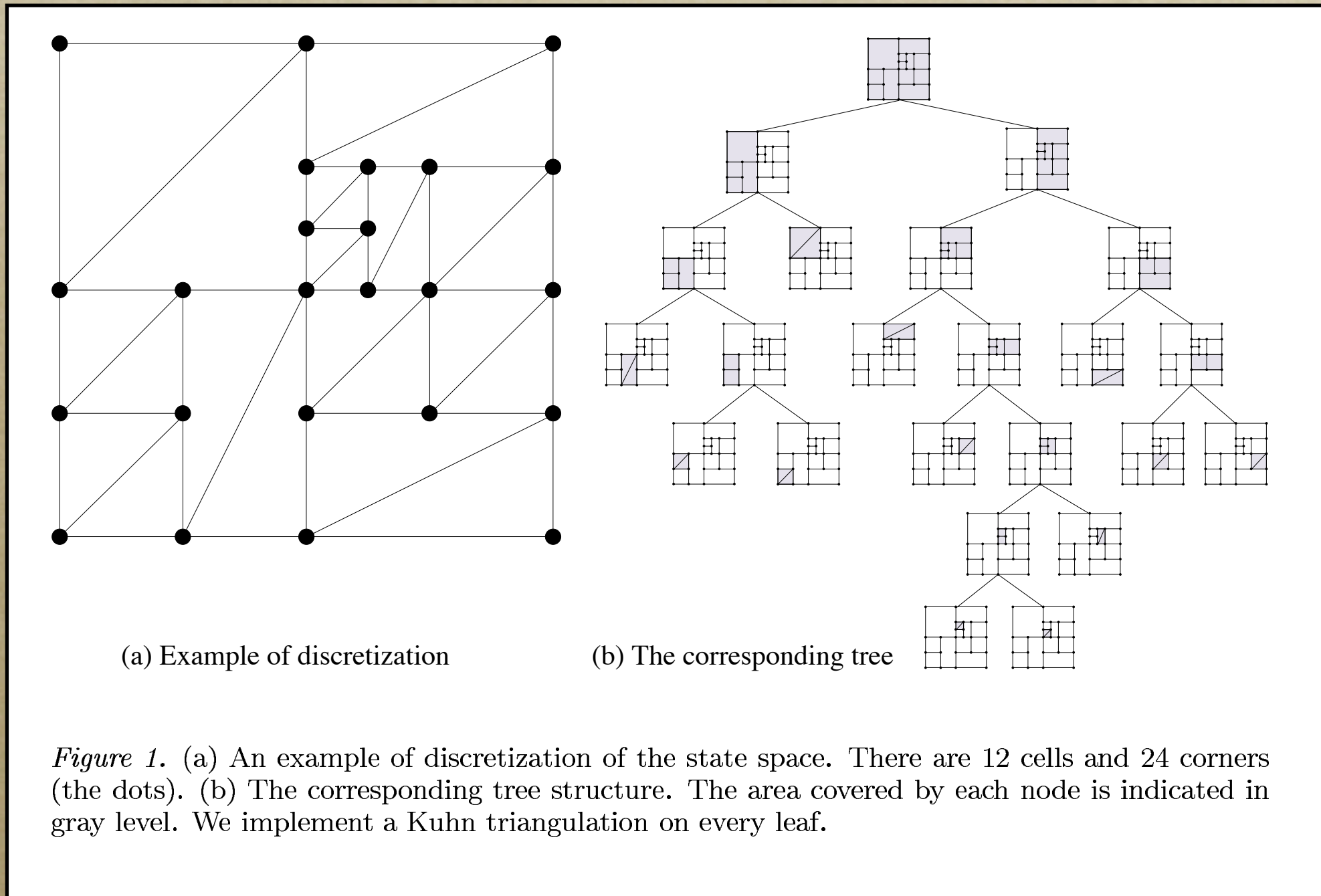
Convergence of Approx TD

- Linear approximation
 - Converges if you follow complete trajectories
- Non-linear approximation
 - Simple method given does not always converge
 - Convergent method now known
 - It's complex

Convergence of TD

- Non-linear TD also shown to converge for contraction mappings
 - Doesn't require updating in trajectories
 - Includes most averaging approximators
 - Includes linear interpolation, not extrapolation

Variable Resolution Techniques



Mountain Car Problem

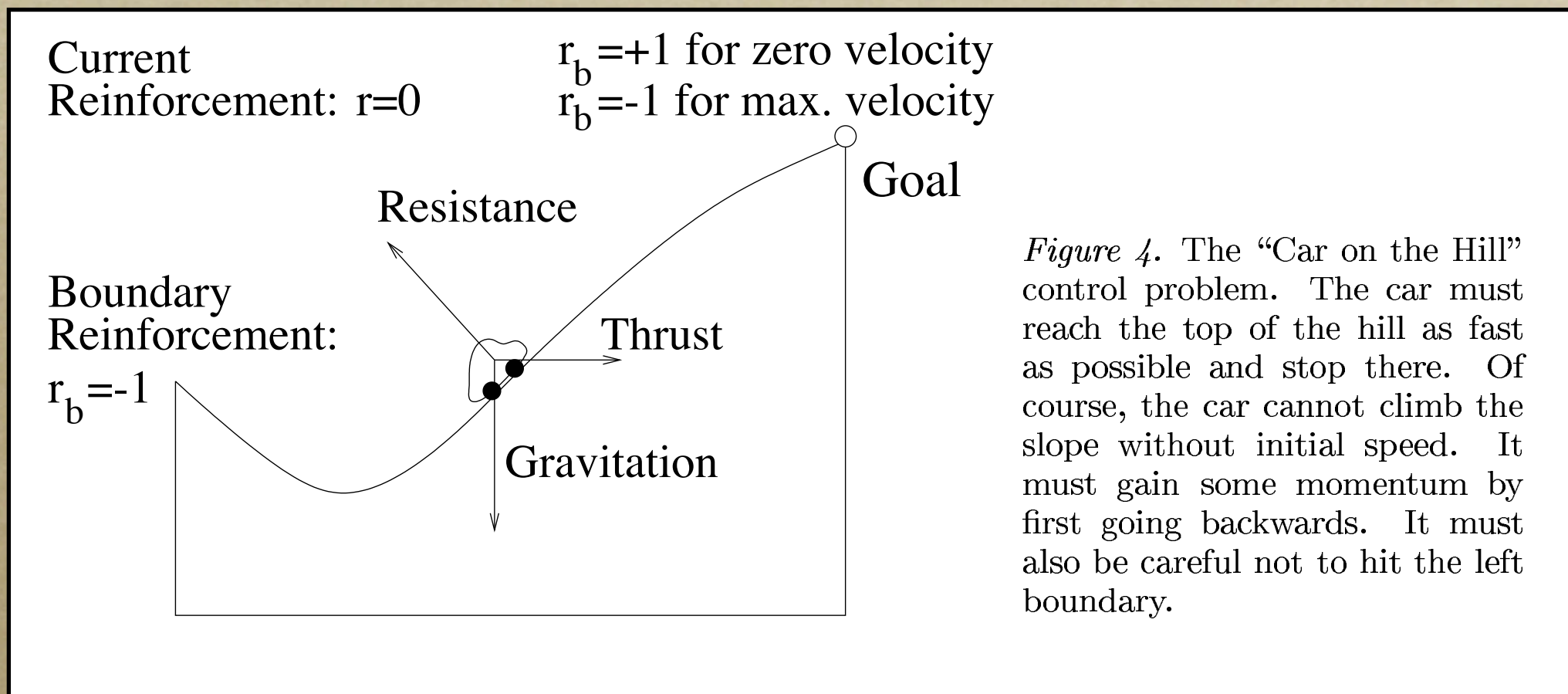


Figure 4. The “Car on the Hill” control problem. The car must reach the top of the hill as fast as possible and stop there. Of course, the car cannot climb the slope without initial speed. It must gain some momentum by first going backwards. It must also be careful not to hit the left boundary.

Mountain Car True Solution

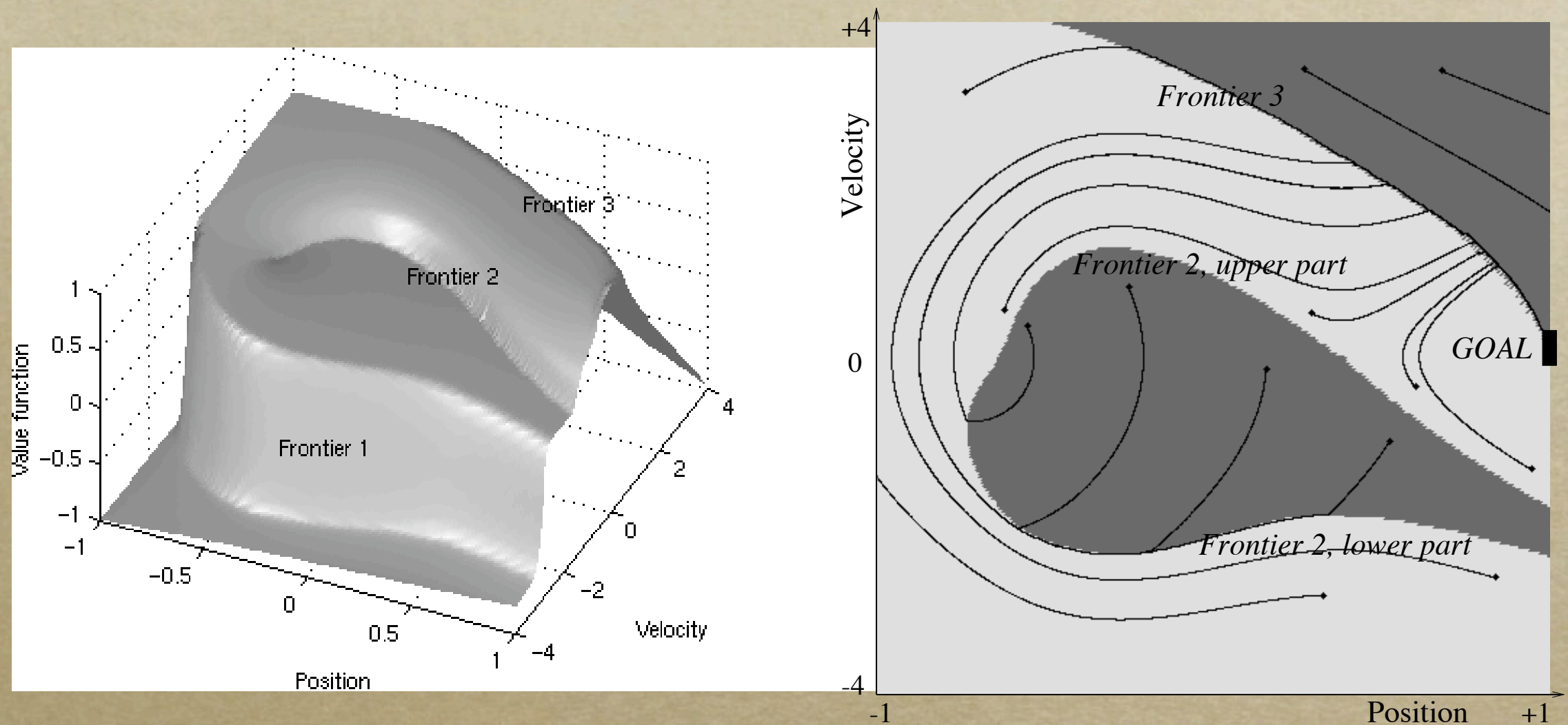
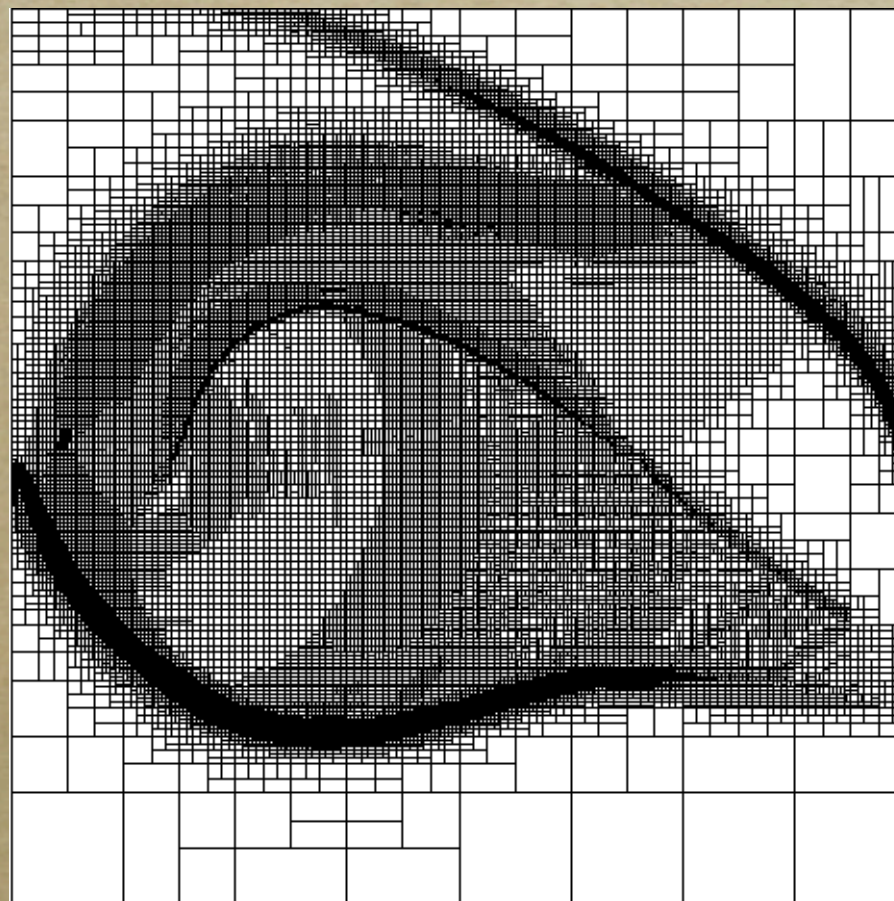


Figure 5. The value function of the Car-on-the-Hill problem obtained by a regular grid of 257 by 257 = 66049 states. The Frontier 1 (white line) illustrates the discontinuity of the VF, the Frontiers 2 and 3 (black lines) stands where there is a transition of the optimal control.

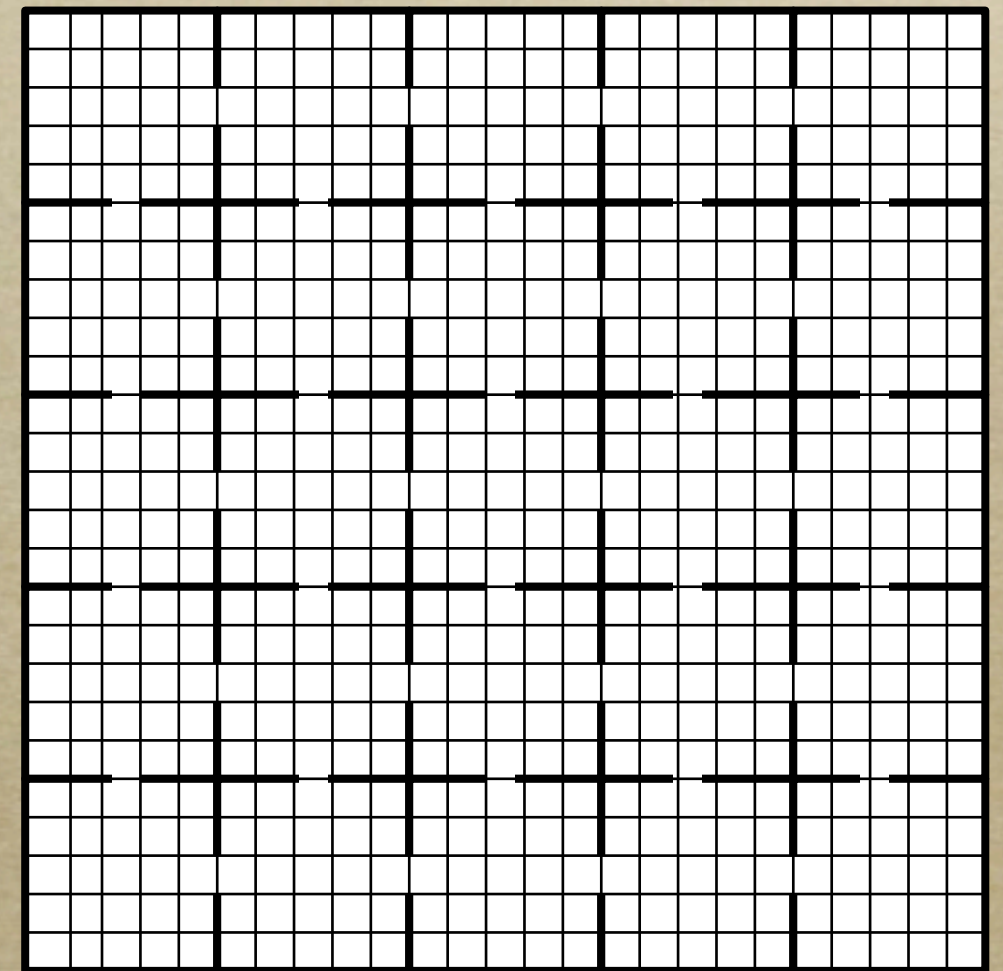
Mountain Car Discretisation

- Split when best action differs across a state
- Split when this state has high variance and would influence a state where the best action changes

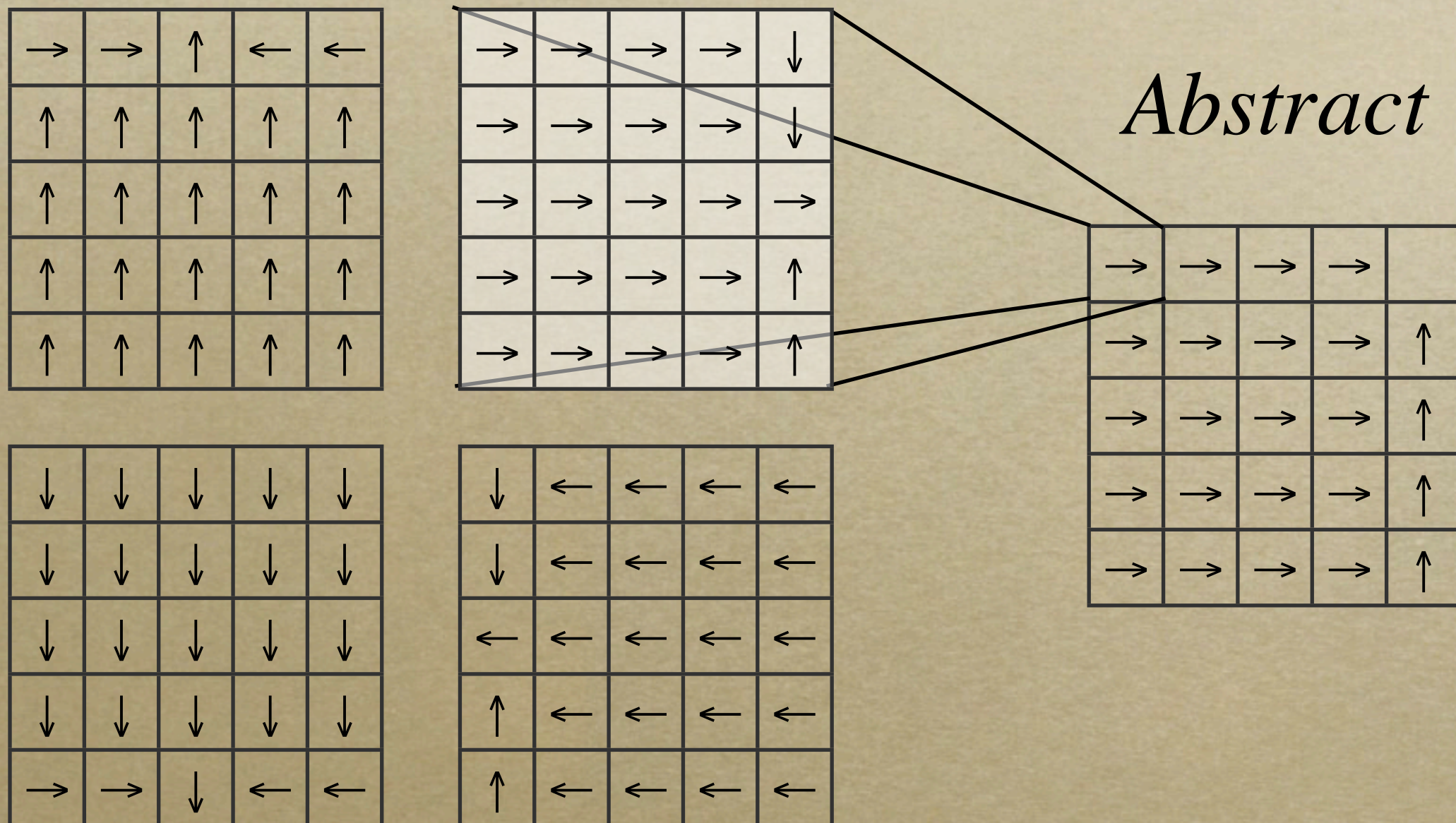


Hierarchical RL: Example

- Consider a 5 by 5 grid of 5 by 5 rooms
- Behaviour within a room is repeated for different rooms



Hierarchical RL



Hierarchical RL

3	2	1	2	3
4	3	2	3	4
5	4	3	4	5
6	5	4	5	6
7	6	5	6	7

7	6	5	4	3
6	5	4	3	2
5	4	3	2	1
6	5	4	3	2
7	6	5	4	3

Abstract

17	12	7	2	0
22	17	12	7	2
27	22	17	12	7
32	27	22	17	12
37	32	27	22	17

7	6	5	6	7
6	5	4	5	6
5	4	3	4	5
4	3	2	3	4
3	2	1	2	3

3	4	5	6	7
2	3	4	5	6
1	2	3	4	5
2	3	4	5	6
3	4	5	6	7

*Value functions
decompose nicely
using cost-to-goal*