Markov Decision Processes

Matt Nassar
CPC Zurich
9/25/2021

Learning to select actions

Learning to select actions

Learning to select actions

Learning to select actions

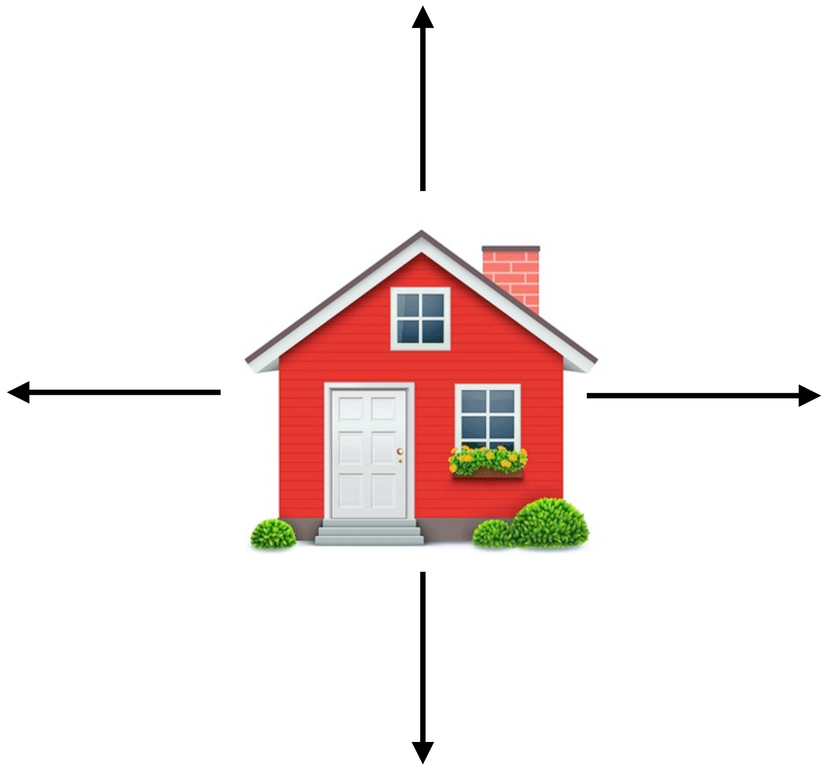
Or a more modern (video game) equivalent?

Learning to select actions

A slightly older — but similar— problem…

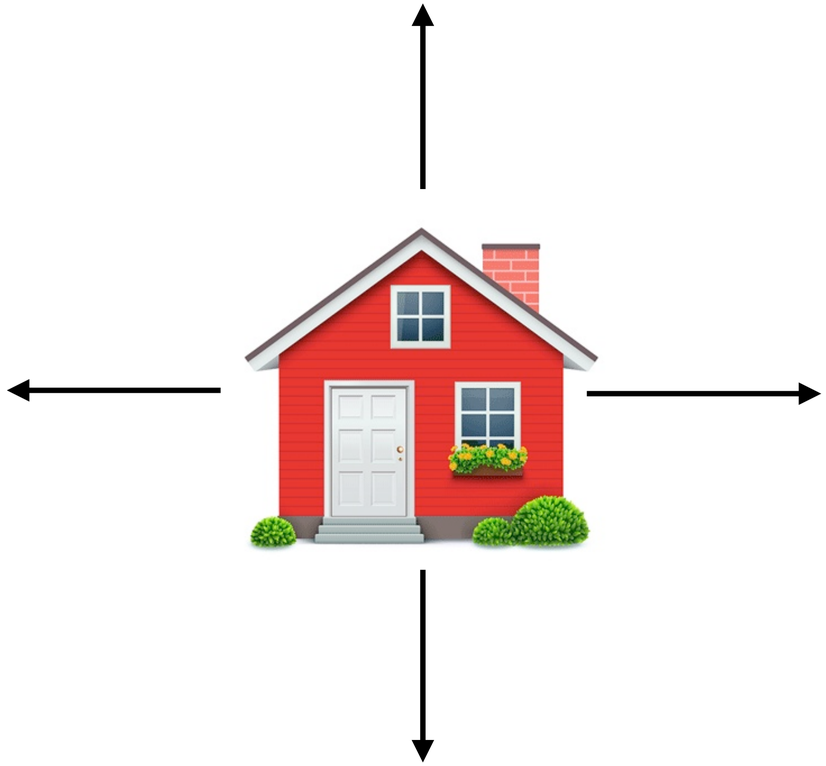
Learning to select actions

Learning to select actions



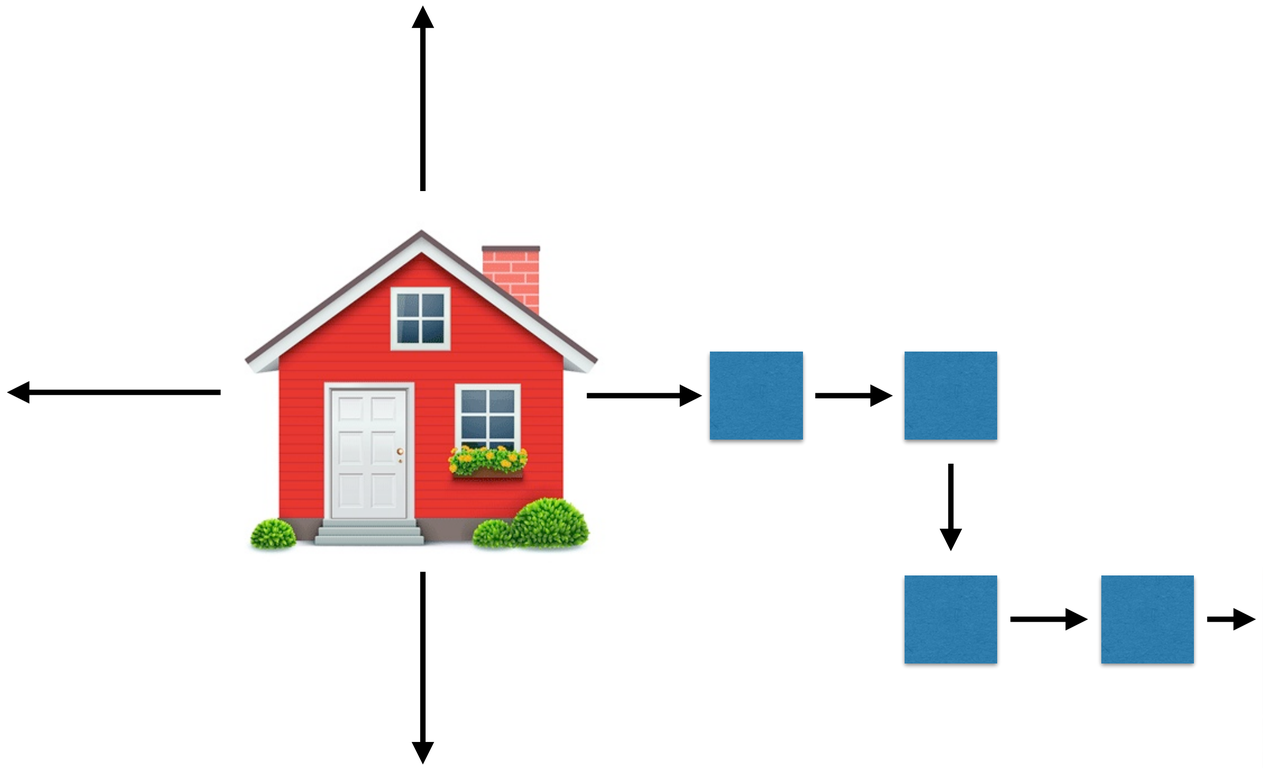
Which way should I go?

Learning to select actions



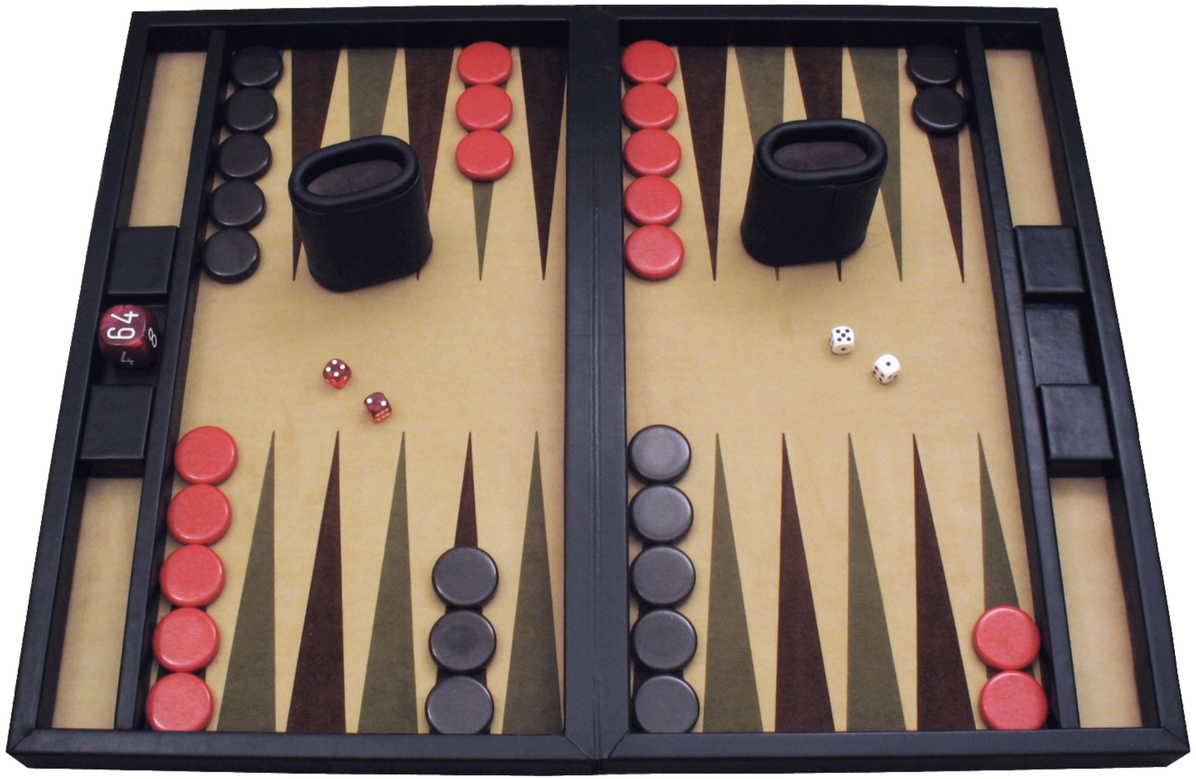
Which way should I go?

Learning to select actions



Which way should I go?

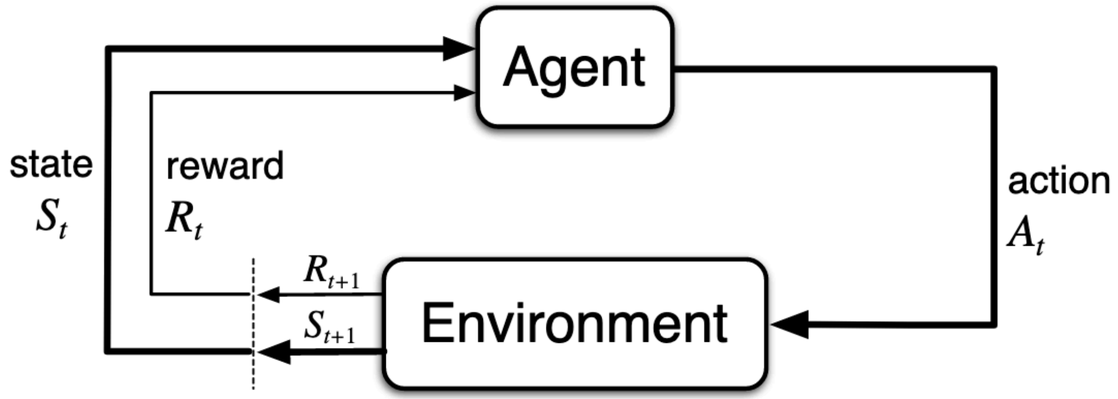
Learning to select actions



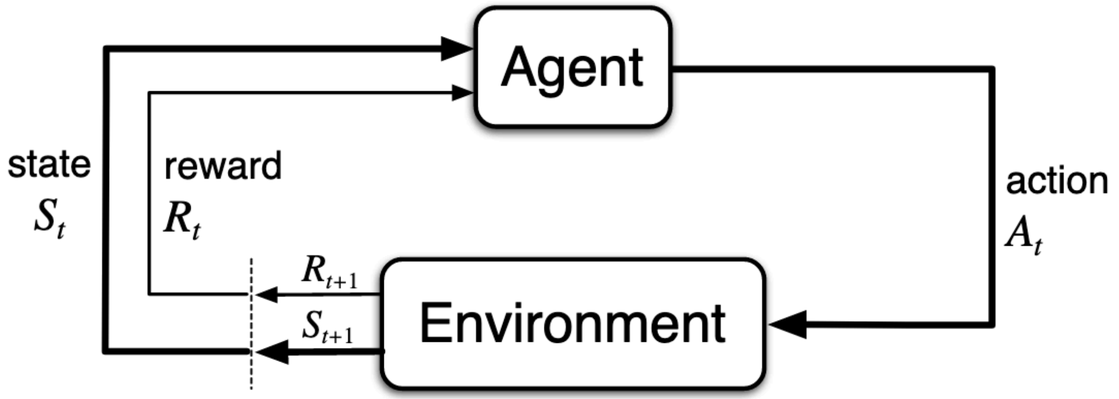
What move should I make?

Markov decision processes

Markov decision processes

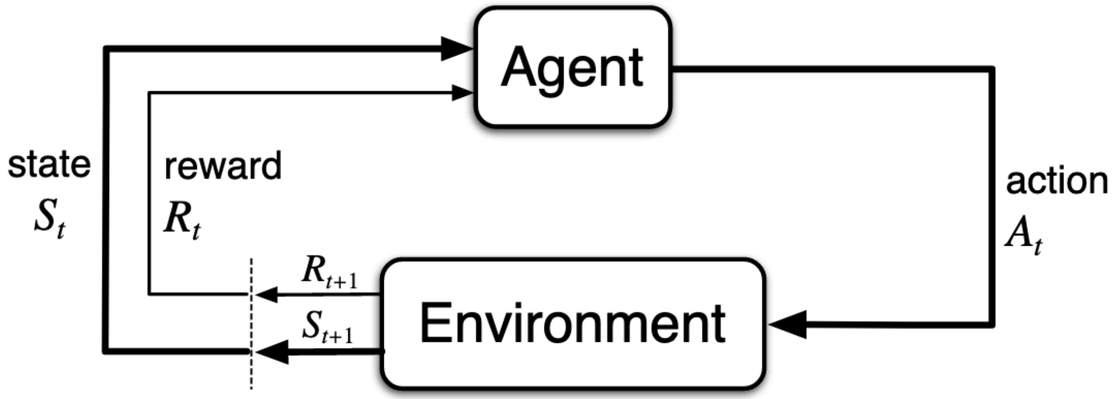


Markov decision processes



- S is a set of states (state space)
- A is a set of actions (action space)
- p(s',r|s,a) is a state transition/reward function

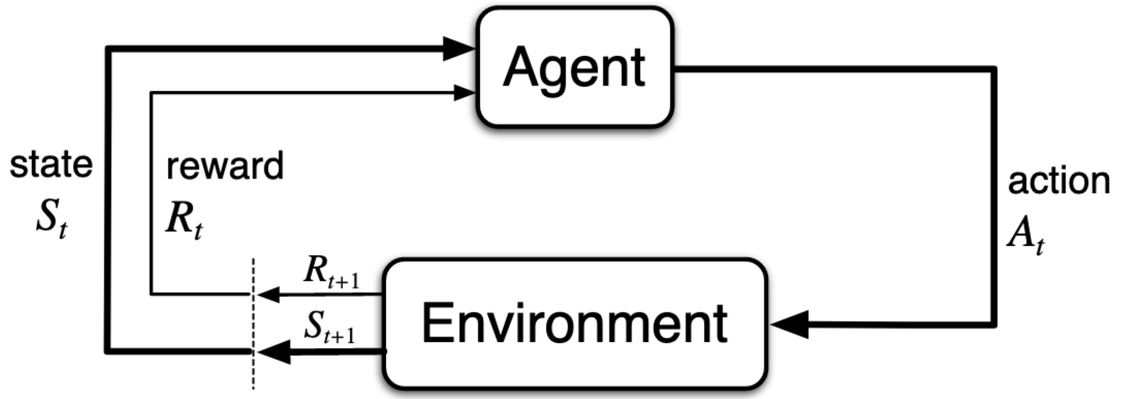
Markov decision processes



- S is a set of states (state space)
- A is a set of actions (action space)
- p(s',r|s,a) is a state transition/reward function

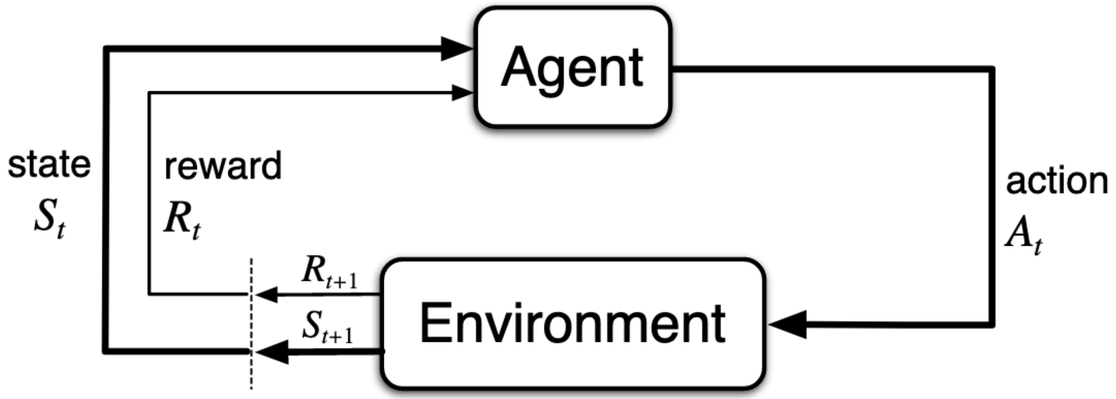
Probability function mapping current state/action onto successor states and rewards

Markov decision processes



- S is a set of states (state space)
- A is a set of actions (action space)
- p(s'|s,a) is a state transition function
- R = p(r|s,a) is a reward function

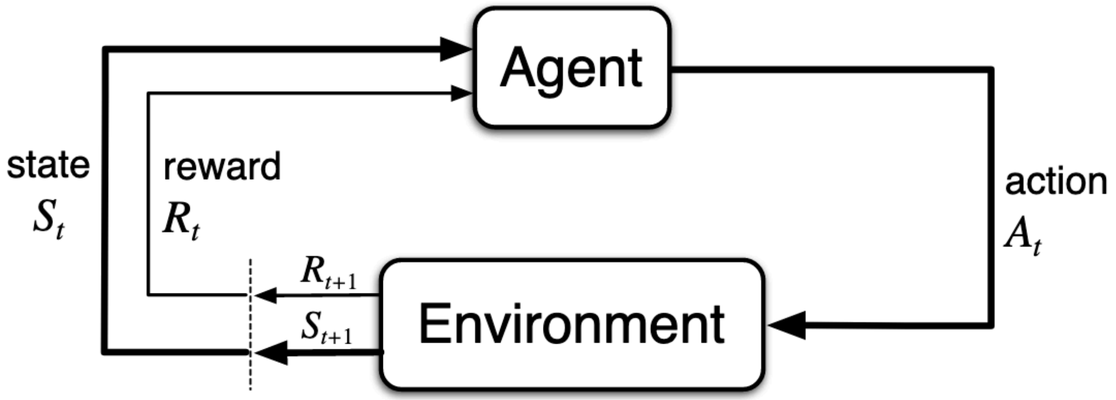
Markov decision processes



- S is a set of states (state space)
- A is a set of actions (action space)
- p(s'|s,a) is a state transition function
- R = p(r|s,a) is a reward function

Goal:
Find a "policy" mapping states to actions
That yields high long run rewards

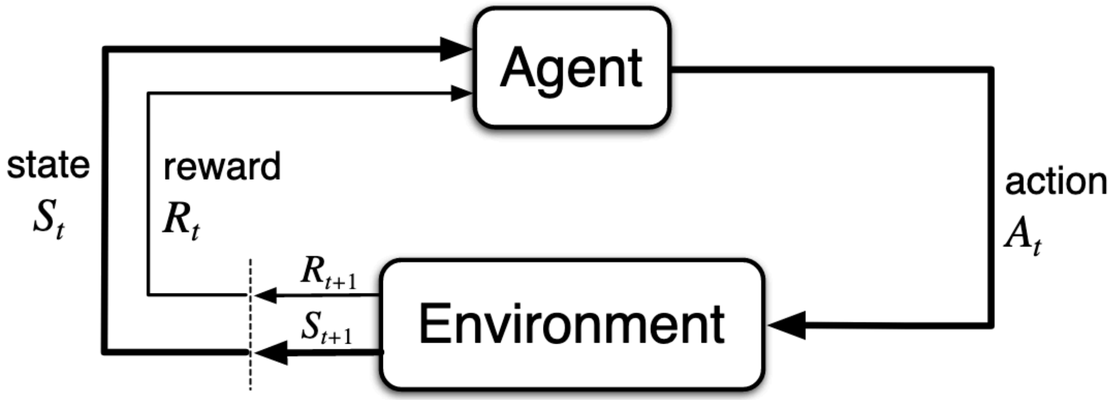
Markov decision processes



- S is a set of states (state space)
- A is a set of actions (action space)
- p(s'|s,a) is a state transition function
- R = p(r|s,a) is a reward function

?

Markov decision processes



- S is a set of states (state space)
- A is a set of actions (action space)
- p(s'|s,a) is a state transition function
- R = p(r|s,a) is a reward function

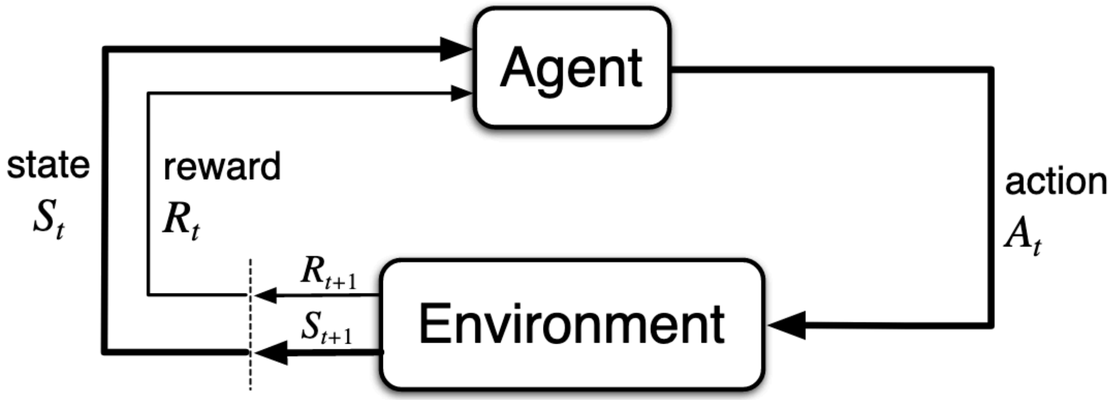
Just one state = standing in front of bandits

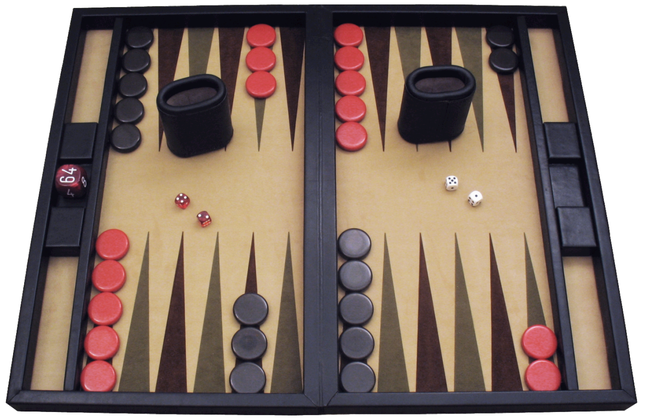
A = { pull arm 1, pull arm 2, don't pull any arm}

Always stay in same state….

p(rla) = payout function of bandit

Markov decision processes





- S is a set of states (state space)
- A is a set of actions (action space)
- p(s'|s,a) is a state transition function
- R = p(r|s,a) is a reward function

All possible board positions (LARGE!)

All legal moves

Updated board positions after move

{0, Win, Lose}

So… how do we get a good policy?

So… how do we get a good policy?

Goal:
Find a “policy” mapping states to actions
That yields high long run rewards

So… how do we get a good policy?

Bellman equation:

$$v_\pi(s) \quad = \sum_a \pi(a|s) \sum_{s',r} p(s', r \,|\, s, a) \Big[r + \gamma v_\pi(s') \Big], \quad \text{for all } s \in \mathcal{S},$$

So… how do we get a good policy?

Bellman equation:

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r \,|\, s, a) \Big[r + \gamma v_\pi(s') \Big], \quad \text{for all } s \in \mathcal{S},$$

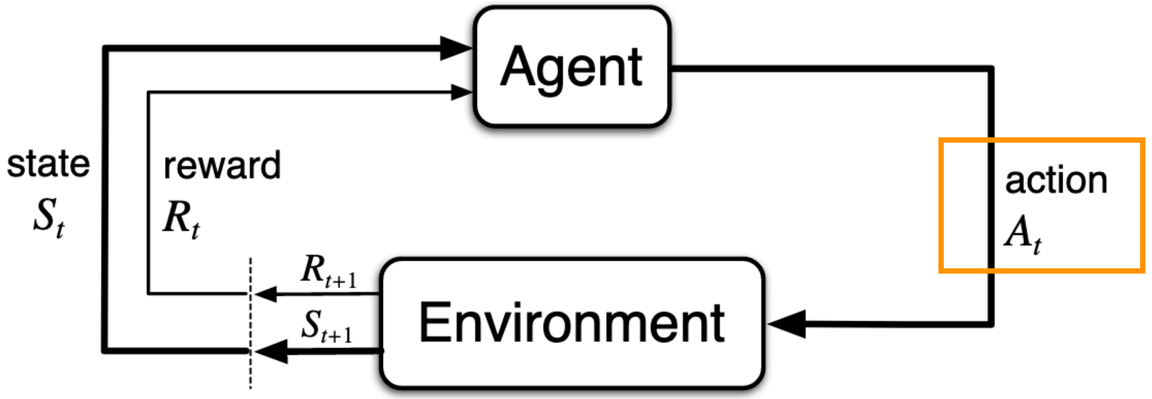
Value of state S under policy pi

So… how do we get a good policy?

Bellman equation:

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)\Big[r + \gamma v_\pi(s')\Big], \quad \text{for all } s \in \mathcal{S},$$

Mixture over actions that will be taken from this state
Using policy Pi

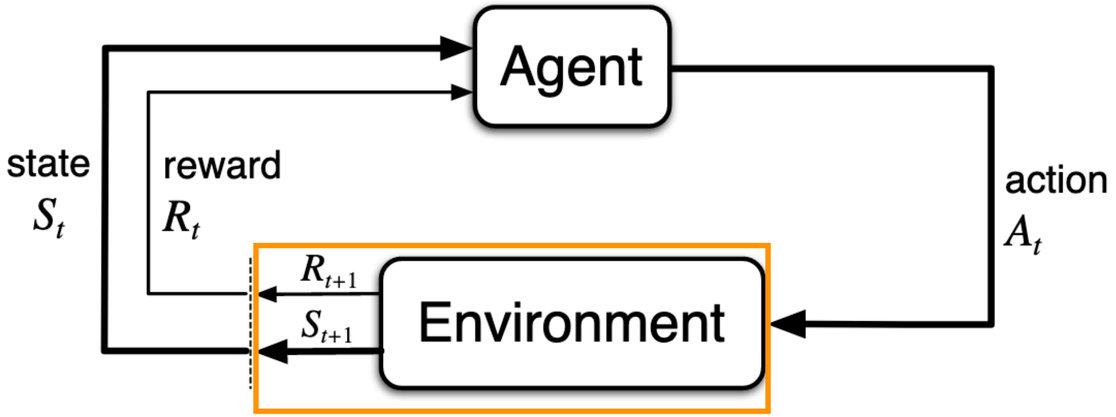


So… how do we get a good policy?

Bellman equation:

$$v_\pi(s) \quad = \sum_a \pi(a|s) \sum_{s',r} p(s', r \,|\, s, a) \Big[r + \gamma v_\pi(s') \Big], \quad \text{for all } s \in \mathcal{S},$$

Probability of each combination of successor state and reward if I take a given action from this state…



So… how do we get a good policy?

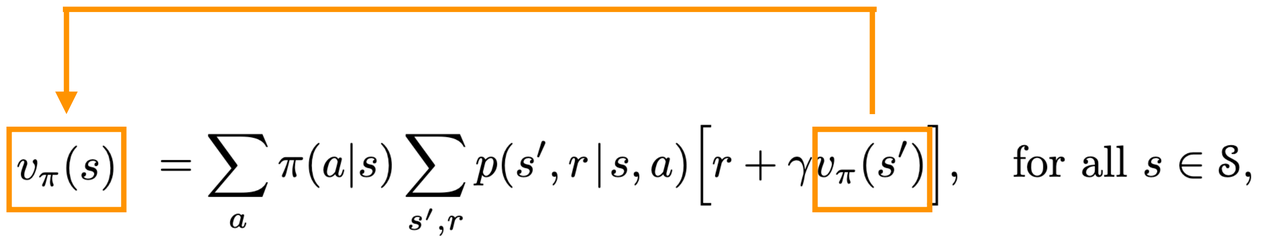
Bellman equation:

$$v_\pi(s) \quad = \sum_a \pi(a|s) \sum_{s',r} p(s', r \,|\, s, a) \Big[r + \gamma v_\pi(s') \Big], \quad \text{for all } s \in \mathcal{S},$$

Reward received plus the “discounted” value of the successor state

So… how do we get a good policy?

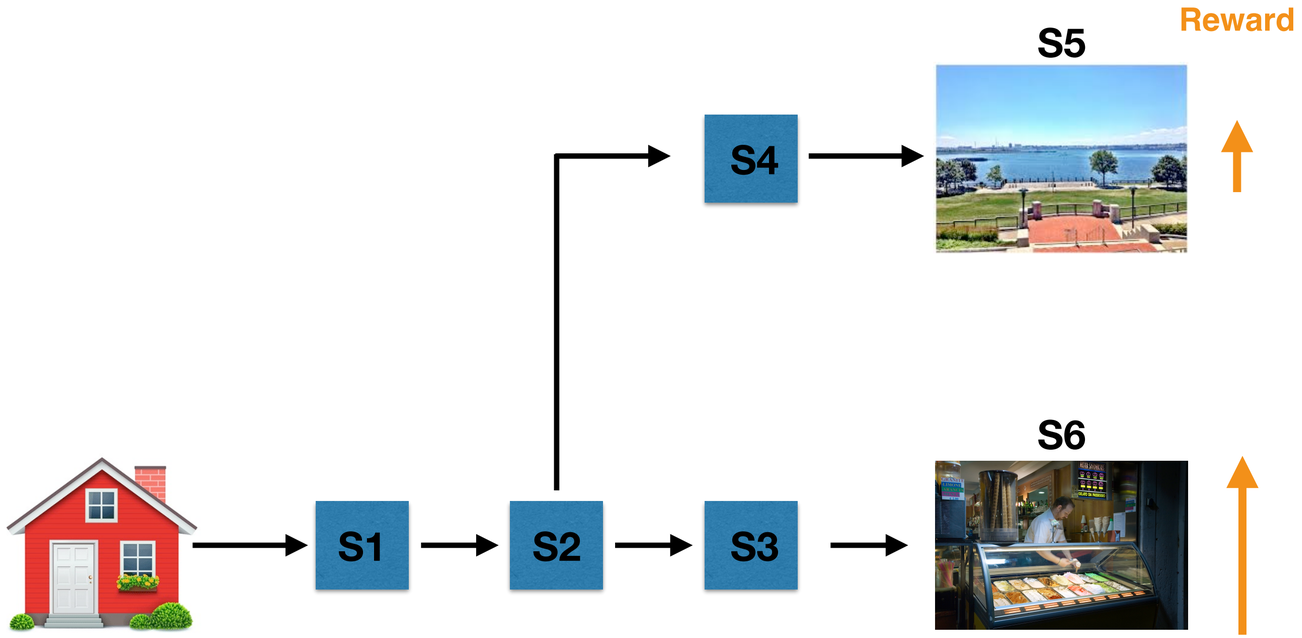
Bellman equation:



$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r \,|\, s, a) \Big[r + \gamma v_\pi(s') \Big], \quad \text{for all } s \in \mathcal{S},$$

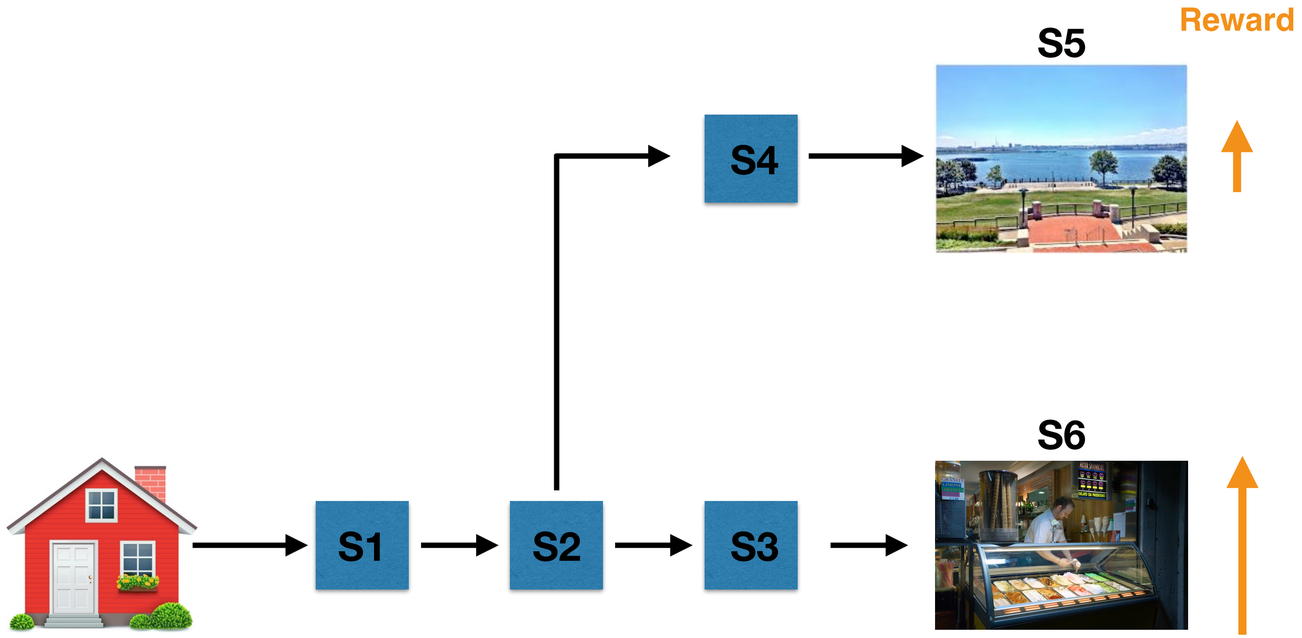
Recursive relationship allows us to “back up”
Value from successor states to current state

Backward planning with a model



$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r \,|\, s, a) \Big[r + \gamma v_\pi(s') \Big], \quad \text{for all } s \in \mathcal{S},$$

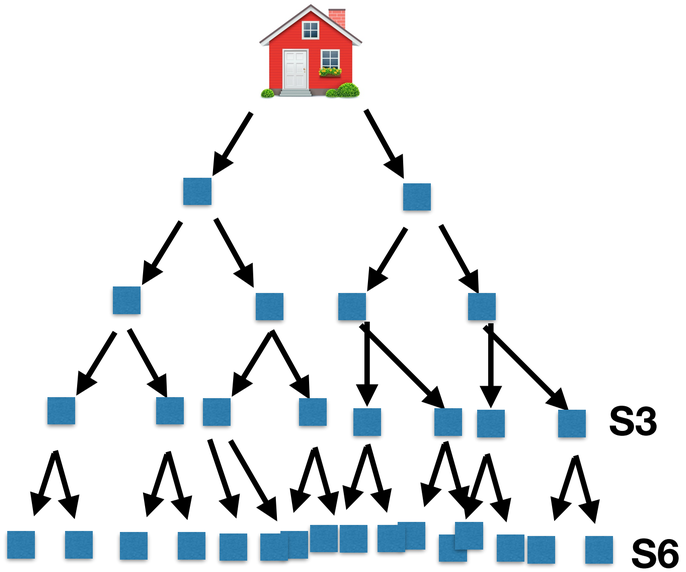
Backward planning with a model

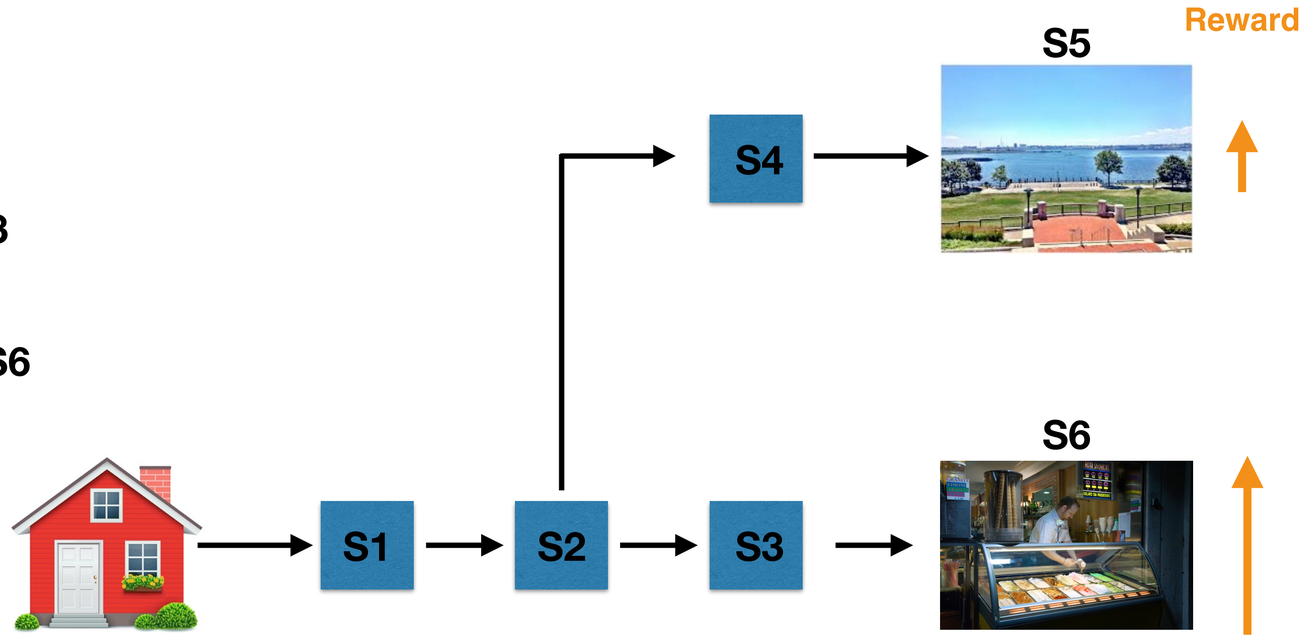


Model of environment/ cognitive map

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)\Big[r + \gamma v_\pi(s')\Big], \quad \text{for all } s \in \mathcal{S},$$

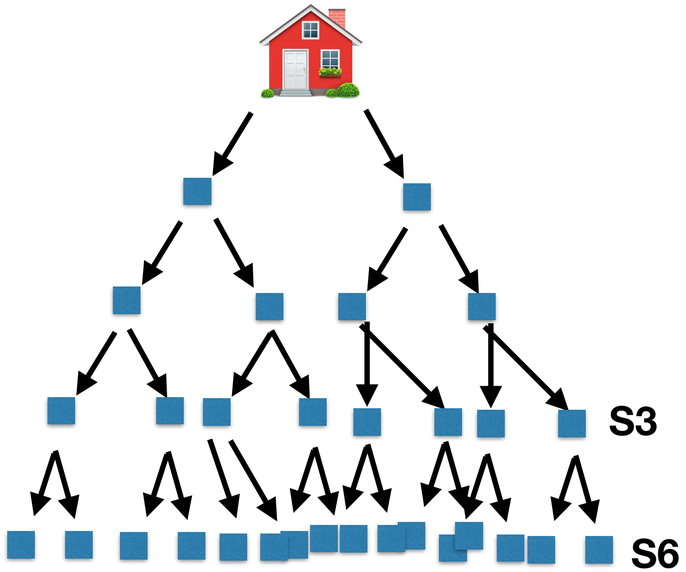
Backward planning with a model

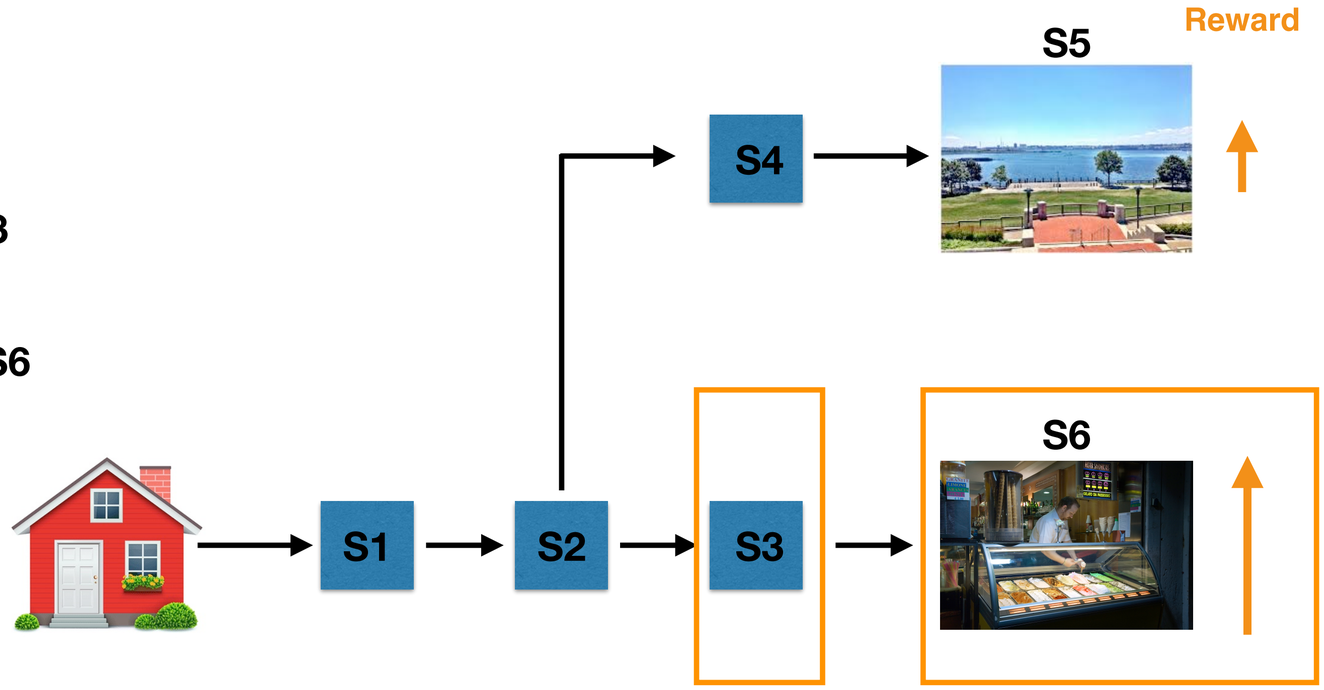




$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r \,|\, s, a) \Big[r + \gamma v_\pi(s') \Big], \quad \text{for all } s \in \mathcal{S},$$

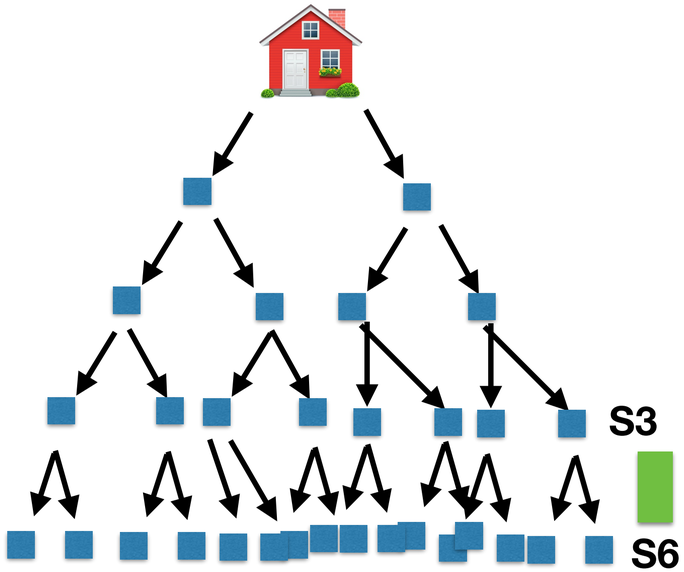
Backward planning with a model

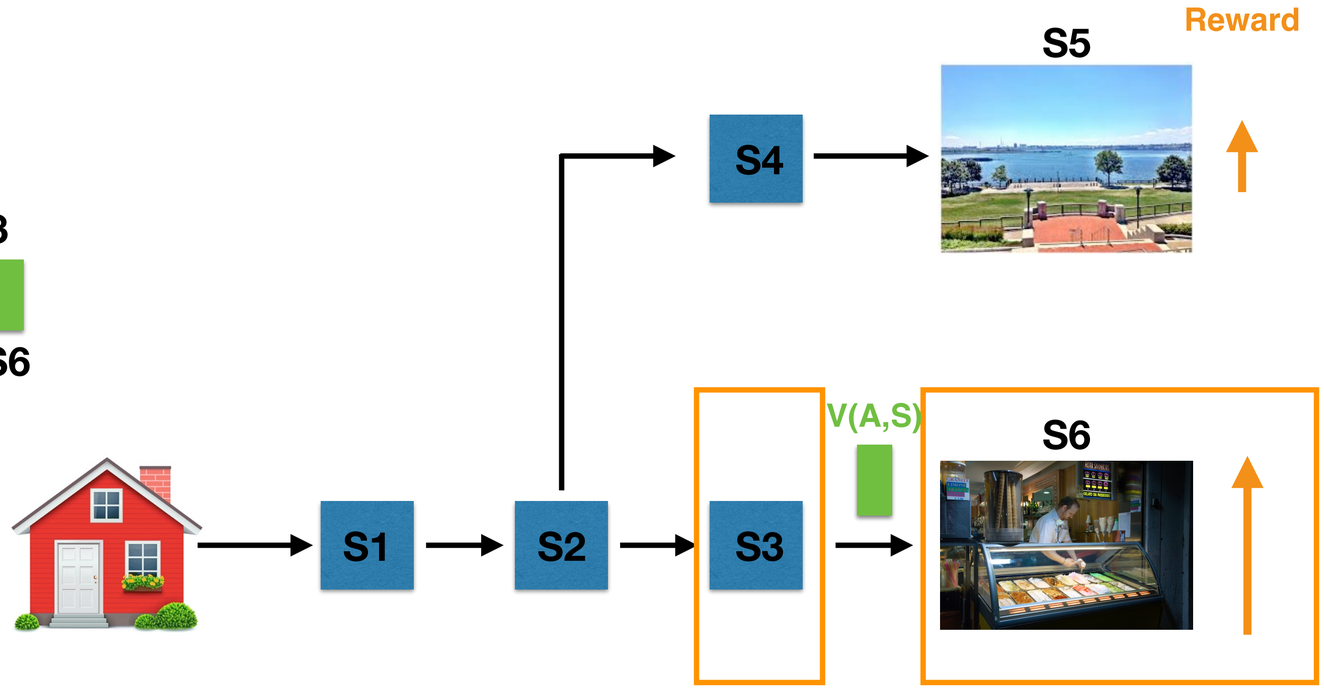




$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r \,|\, s, a) \Big[r + \gamma v_\pi(s') \Big], \quad \text{for all } s \in \mathcal{S},$$

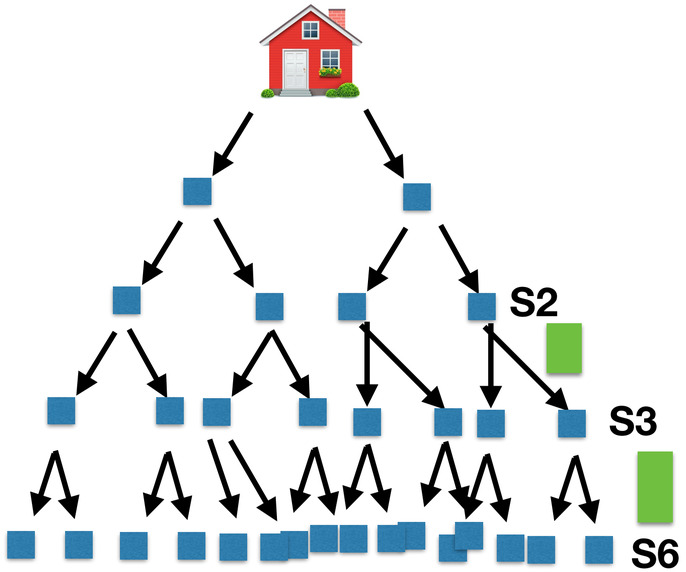
Backward planning with a model

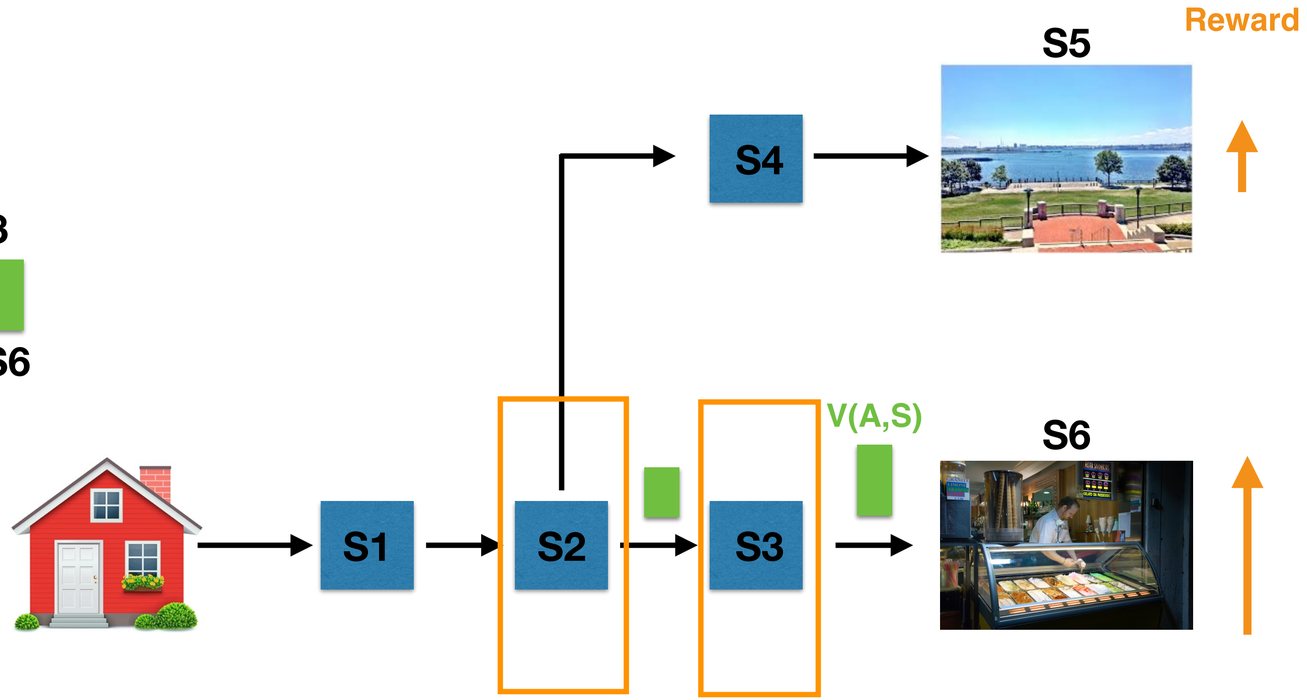




$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)\Big[r + \gamma v_\pi(s')\Big], \quad \text{for all } s \in \mathcal{S},$$

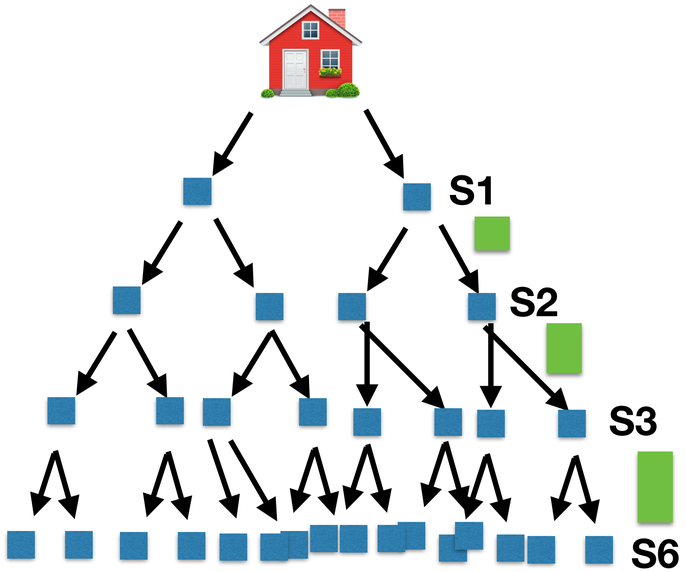
Backward planning with a model

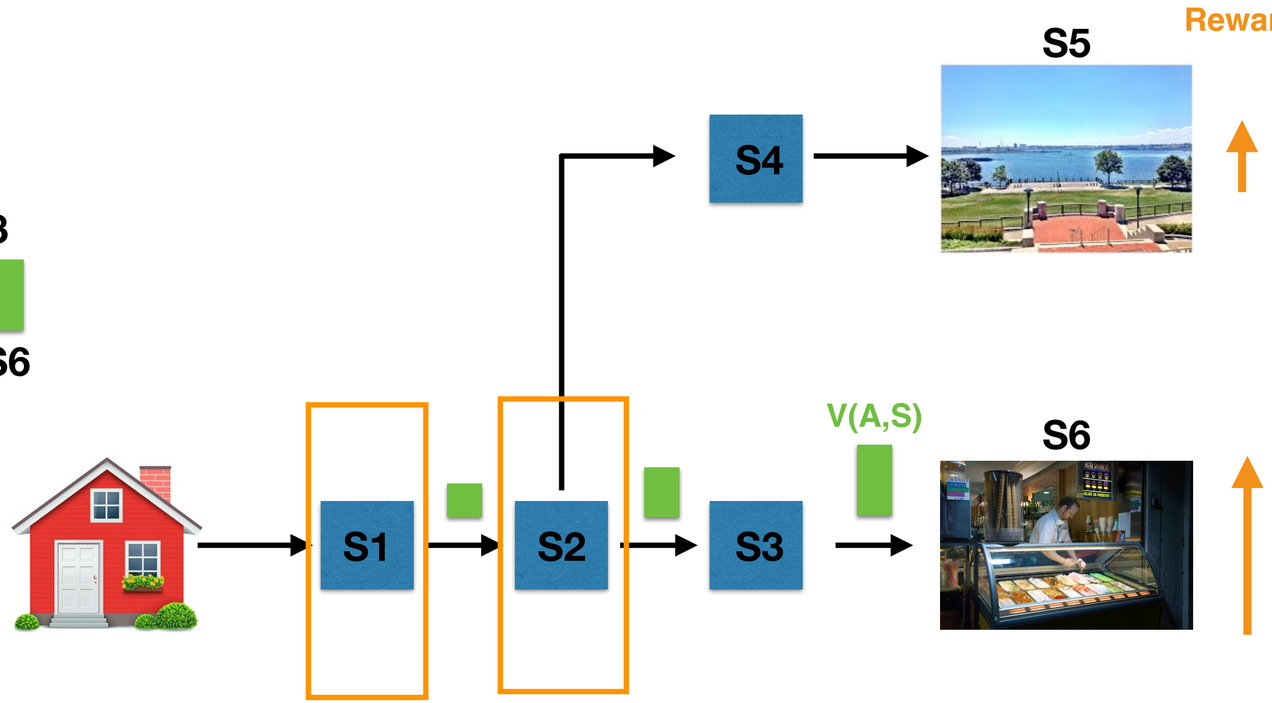




$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)\Big[r + \gamma v_\pi(s')\Big], \quad \text{for all } s \in \mathcal{S},$$

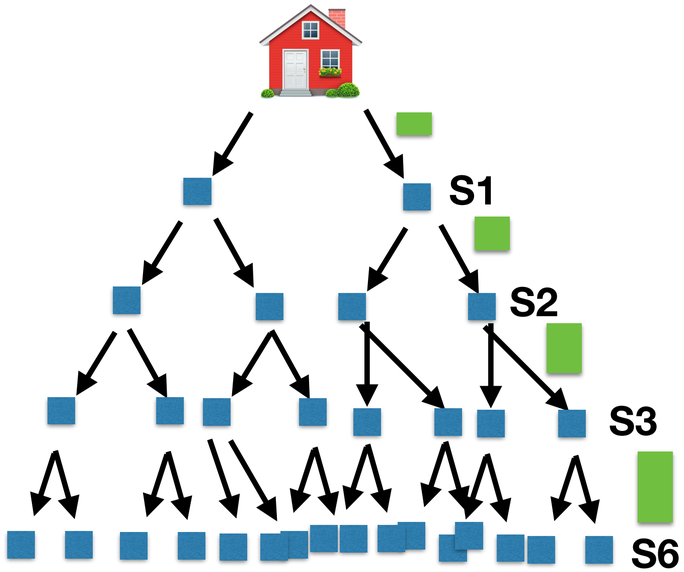
Backward planning with a model

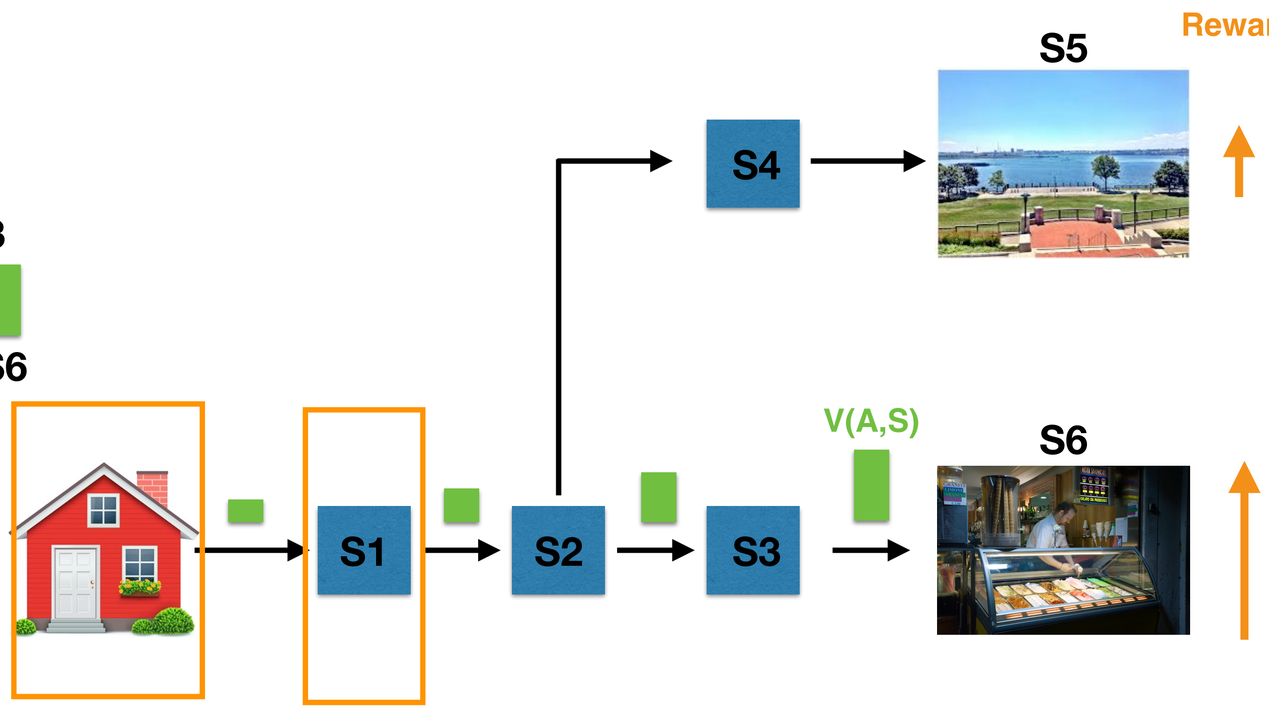




$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)\Big[r + \gamma v_\pi(s')\Big], \quad \text{for all } s \in \mathcal{S},$$

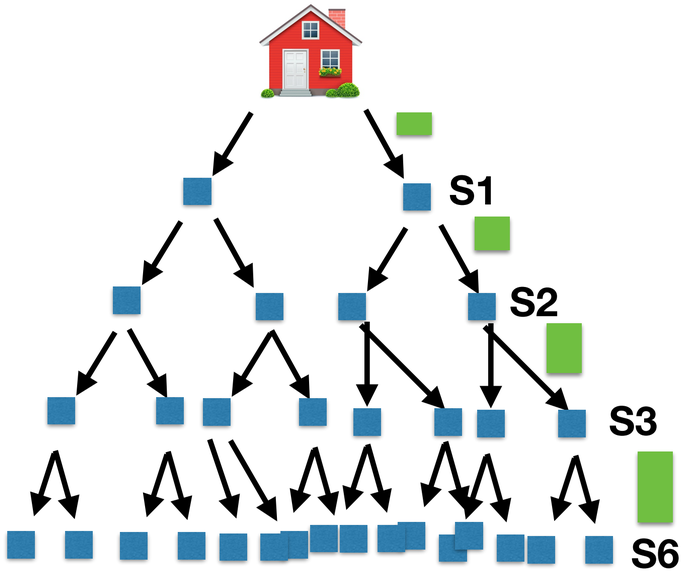
Backward planning with a model



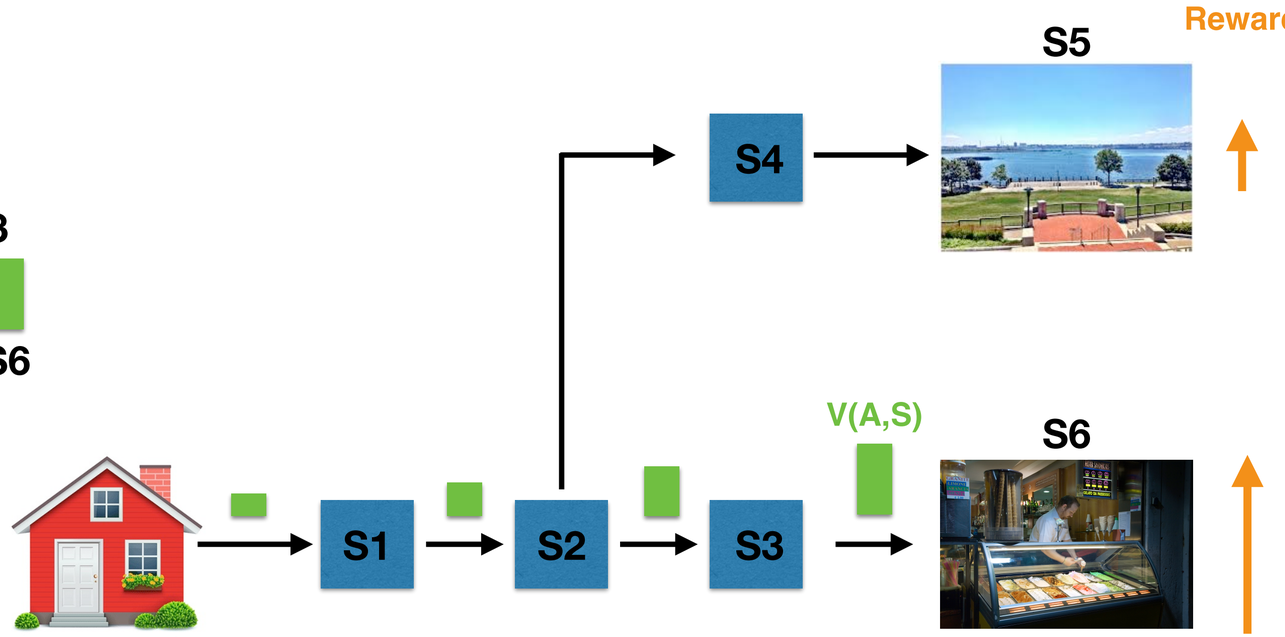


$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)\Big[r + \gamma v_\pi(s')\Big], \quad \text{for all } s \in \mathcal{S},$$

Backward planning with a model

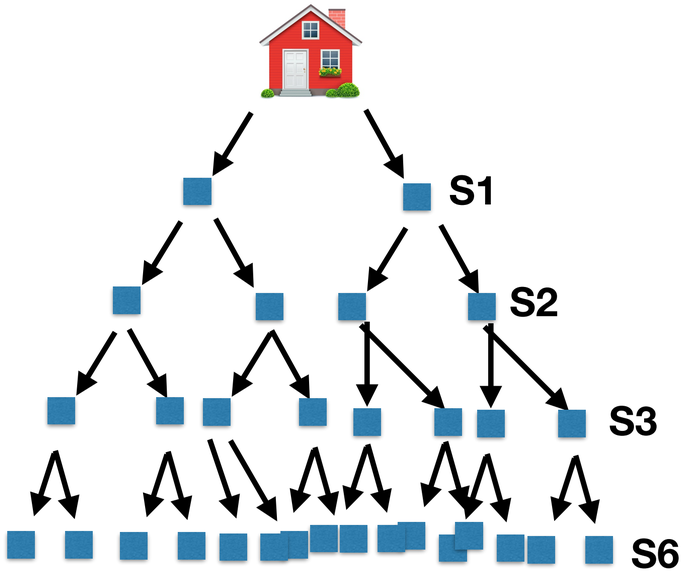


Pros: Flexible & Accurate

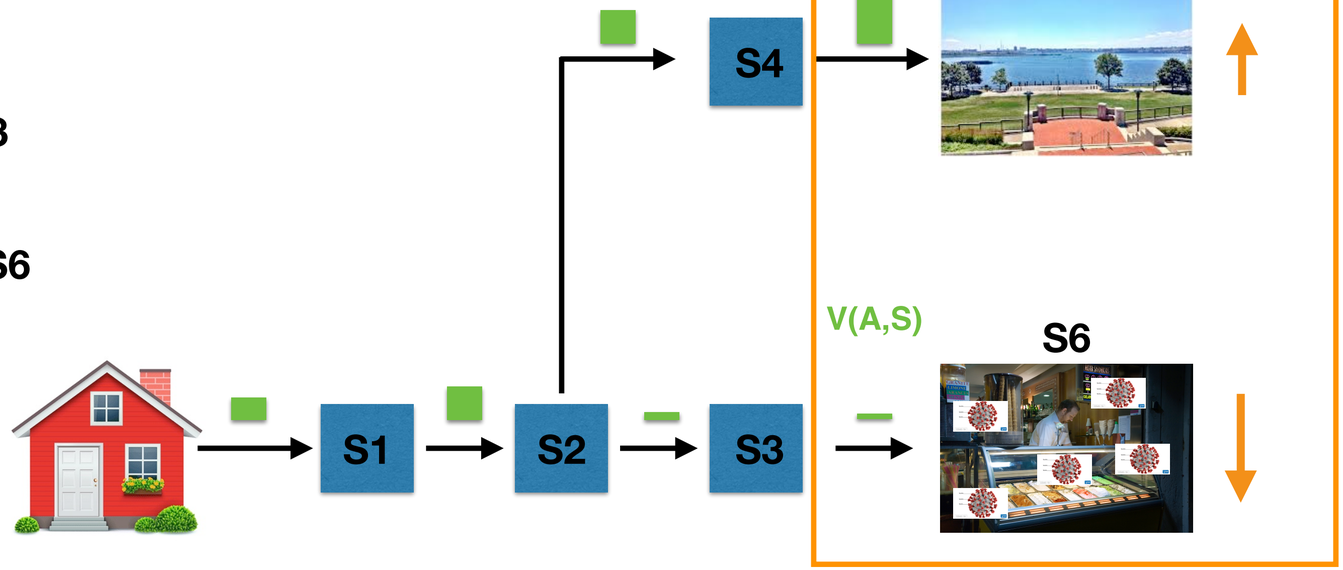


$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)\Big[r + \gamma v_\pi(s')\Big], \quad \text{for all } s \in \mathcal{S},$$

Backward planning with a model



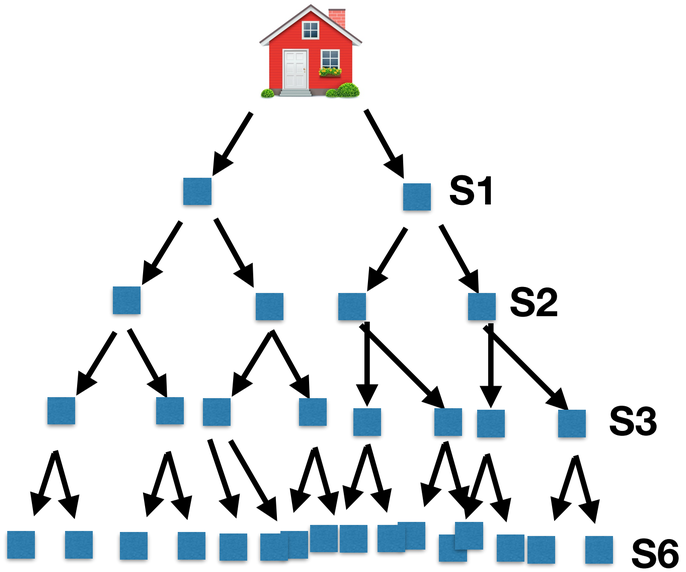
Pros: Flexible & Accurate

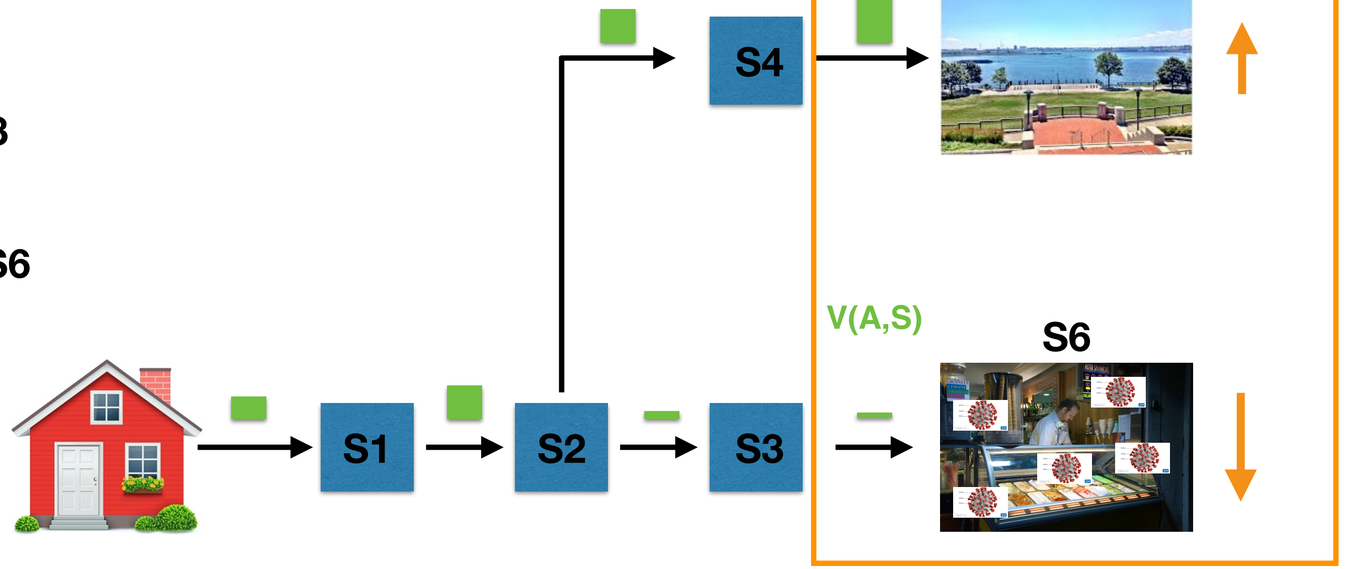


Flexible reevaluation
In response to
Change in reward
Contingencies

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r \,|\, s, a) \Big[r + \gamma v_\pi(s') \Big], \quad \text{for all } s \in \mathcal{S},$$

Backward planning with a model





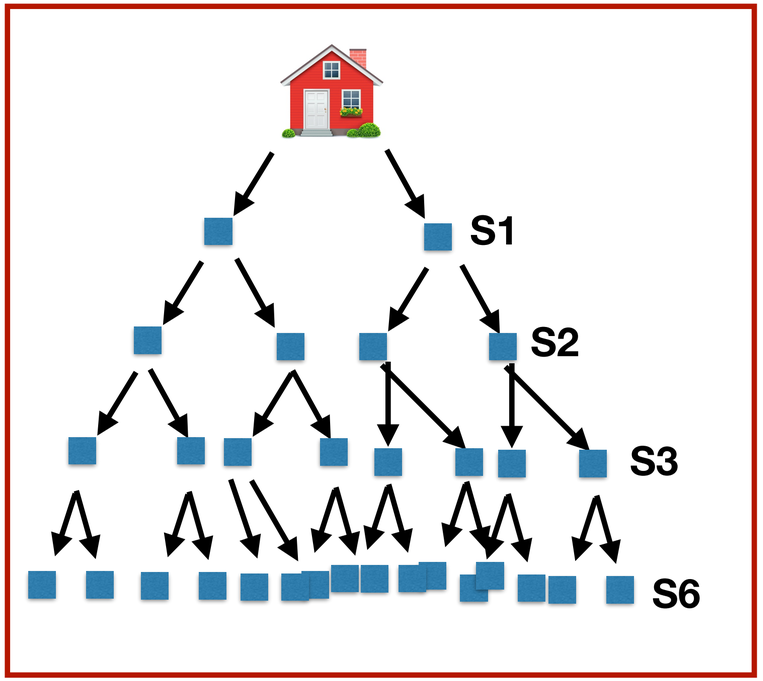
Flexible reevaluation
In response to
Change in reward
Contingencies

Pros: Flexible & Accurate

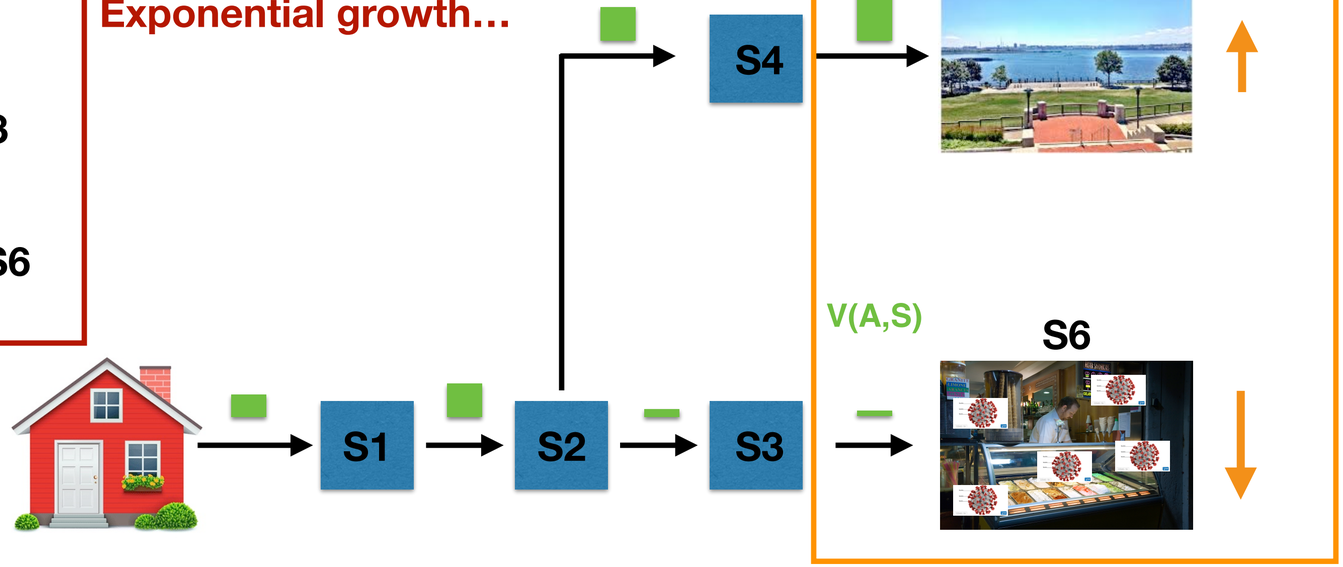
Cons: Computationally expensive!!!

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r \,|\, s, a) \Big[r + \gamma v_\pi(s') \Big], \quad \text{for all } s \in \mathcal{S},$$

Backward planning with a model



Exponential growth…



Flexible reevaluation
In response to
Change in reward
Contingencies

Pros: Flexible & Accurate

Cons: Computationally expensive!!!

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)\Big[r + \gamma v_\pi(s')\Big], \quad \text{for all } s \in \mathcal{S},$$

Reinforcement learning: Cheap solutions to MDPs that don’t require a model

Temporal difference learning

Temporal difference learning

Temporal difference learning:

$$V(A,S) \leftarrow V(A,S) + \alpha(r(A) + \gamma V(A,S') - V(A,S))$$

Temporal difference learning

Updated action value
In a given state

$$V(A,S) \leftarrow V(A,S) + \alpha(r(A) + \gamma V(A,S') - V(A,S))$$

Temporal difference learning

Depends on previous Action value

$$V(A,S) \leftarrow V(A,S) + \alpha(r(A) + \gamma V(A,S') - V(A,S))$$

Temporal difference learning

How much better the reward was
Than what was expected…

$$V(A,S) \leftarrow V(A,S) + \alpha(r(A) + \gamma V(A,S') - V(A,S))$$

Temporal difference learning

But we also give an action some credit for getting us to a good state

$$V(A,S) \leftarrow V(A,S) + \alpha(r(A) + \boxed{\gamma V(A,S')} - V(A,S))$$

Temporal difference learning

But we discount the value of future states, relative to current rewards

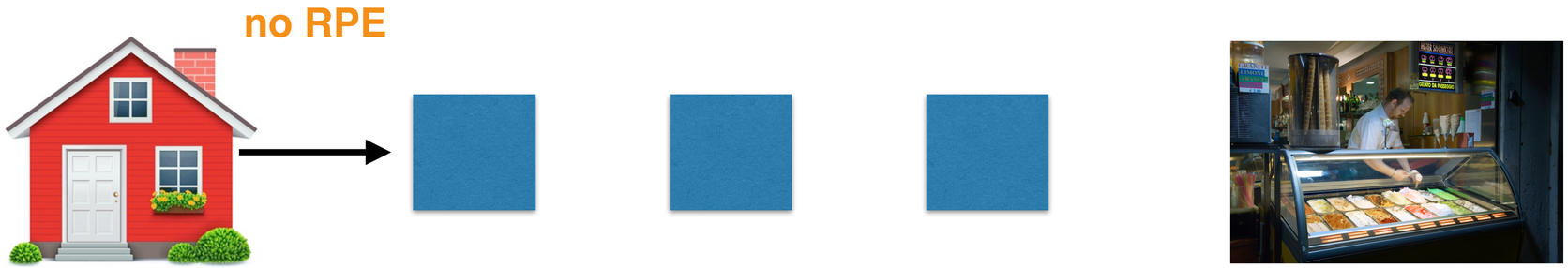
$$V(A,S) \leftarrow V(A,S) + \alpha(r(A) + \gamma V(A,S') - V(A,S))$$

Temporal difference learning

Now this entire term is referred to as the reward prediction error

$$V(A,S) \leftarrow V(A,S) + \alpha(r(A) + \gamma V(A,S') - V(A,S))$$

Temporal difference learning

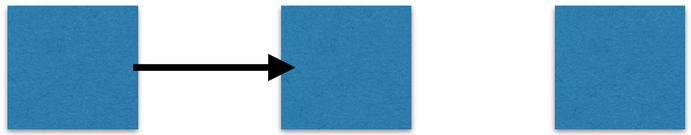


RPE

$$V(A,S) \leftarrow V(A,S) + \alpha(r(A) + \gamma V(A,S') - V(A,S))$$

Temporal difference learning

no RPE

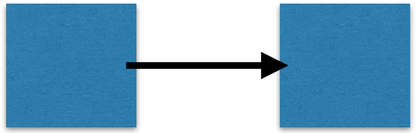


RPE

$$V(A,S) \leftarrow V(A,S) + \alpha(r(A) + \gamma V(A,S') - V(A,S))$$

Temporal difference learning

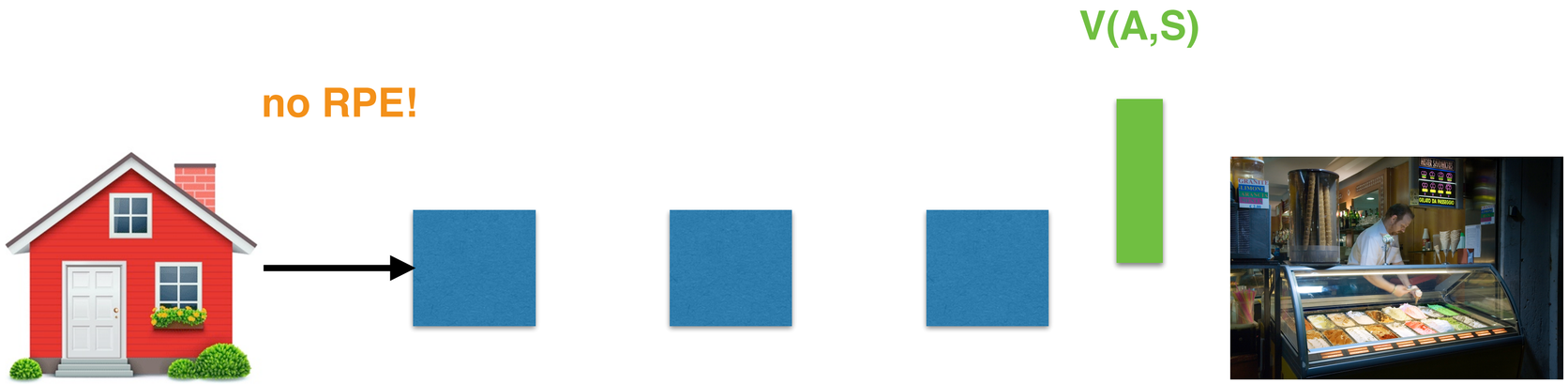
no RPE



RPE

$$V(A,S) \leftarrow V(A,S) + \alpha(r(A) + \gamma V(A,S') - V(A,S))$$

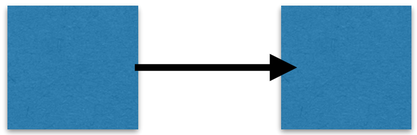
Temporal difference learning



RPE

$$V(A,S) \leftarrow V(A,S) + \alpha(r(A) + \gamma V(A,S') - V(A,S))$$

Temporal difference learning



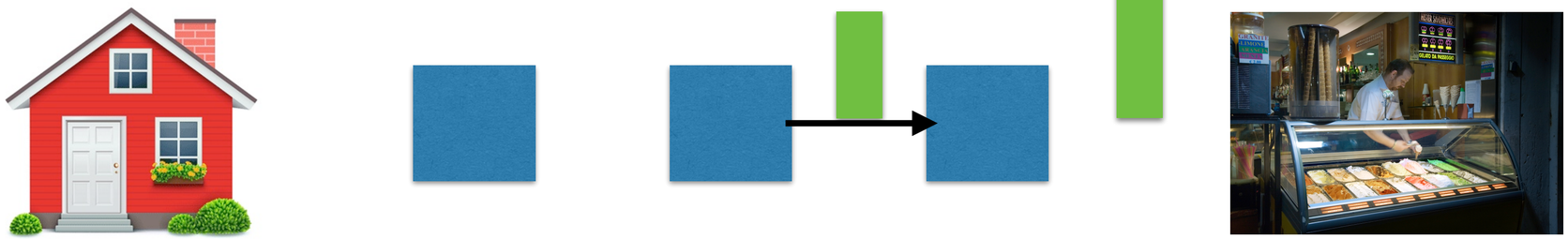
no RPE!

V(A,S)

RPE

$$V(A,S) \leftarrow V(A,S) + \alpha(r(A) + \gamma V(A,S') - V(A,S))$$

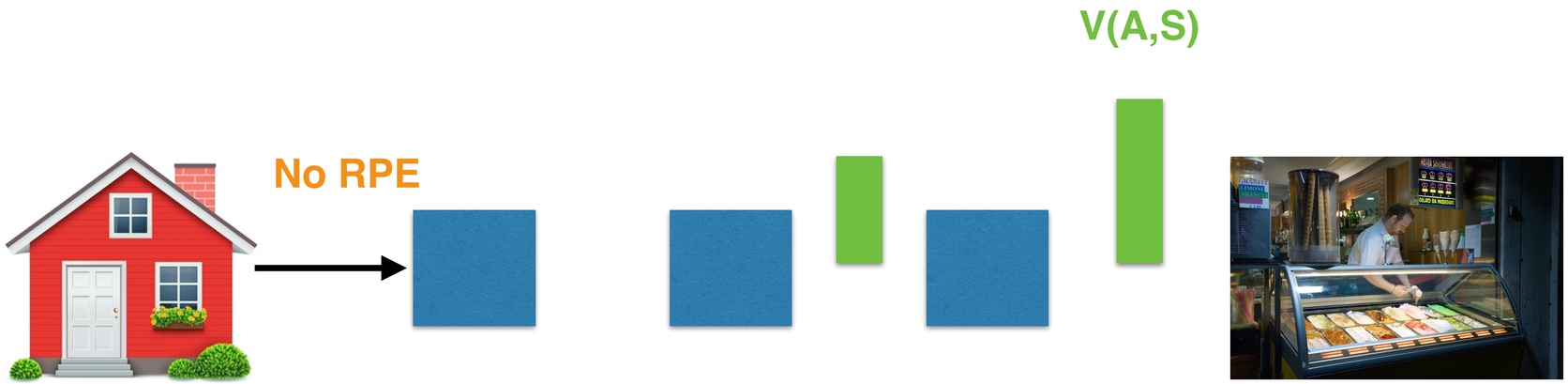
Temporal difference learning



RPE

$$V(A,S) \leftarrow V(A,S) + \alpha(r(A) + \gamma V(A,S') - V(A,S))$$

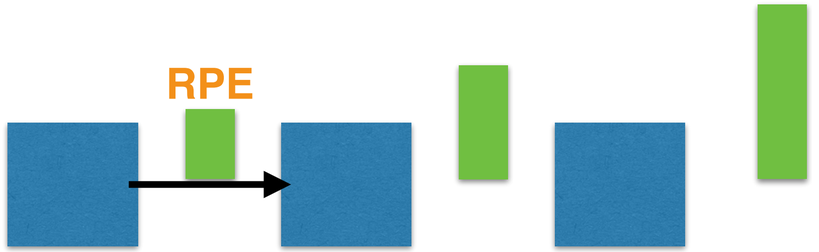
Temporal difference learning



RPE

$$V(A,S) \leftarrow V(A,S) + \alpha(r(A) + \gamma V(A,S') - V(A,S))$$

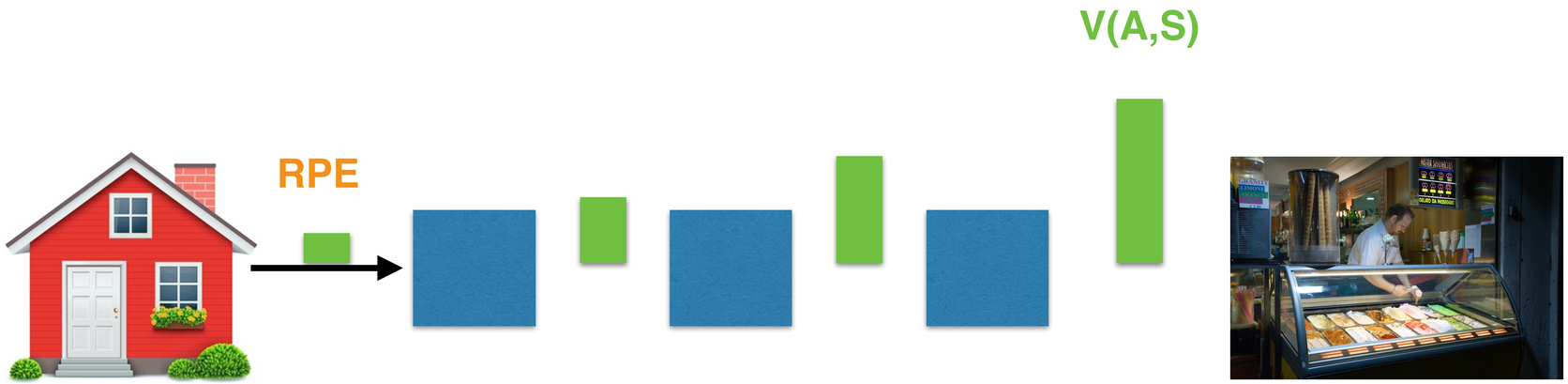
Temporal difference learning



RPE

$$V(A,S) \leftarrow V(A,S) + \alpha(r(A) + \gamma V(A,S') - V(A,S))$$

Temporal difference learning



RPE

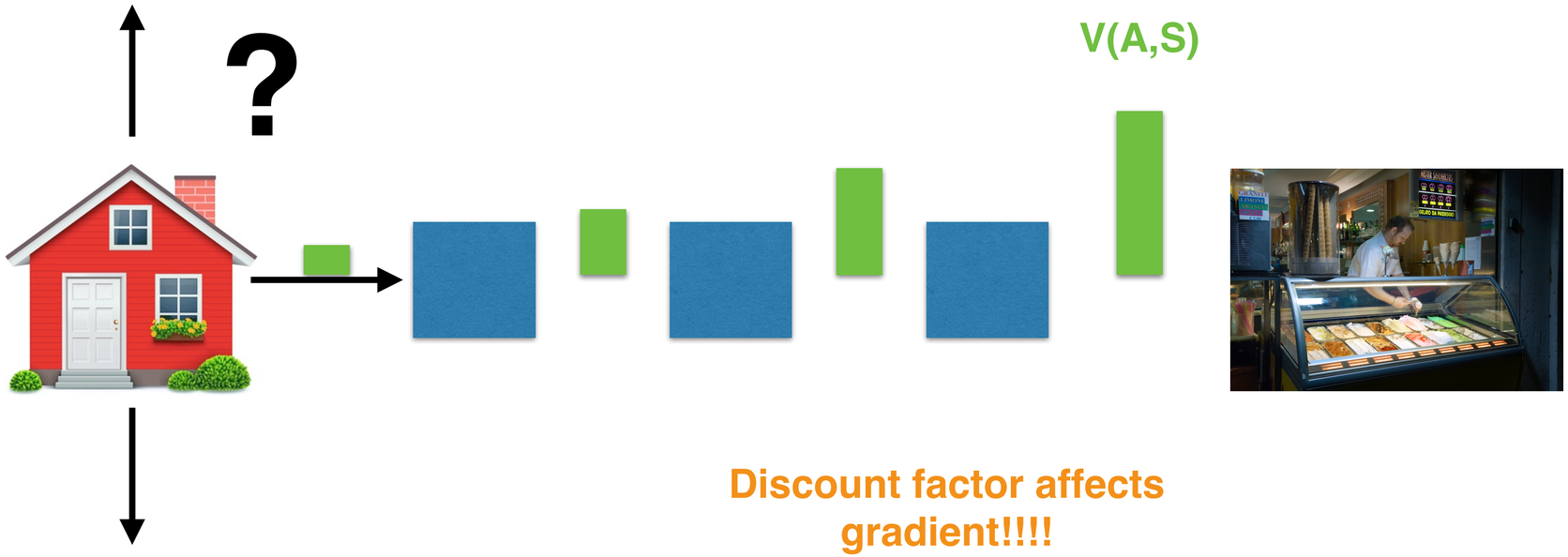
$$V(A,S) \leftarrow V(A,S) + \alpha(r(A) + \gamma V(A,S') - V(A,S))$$

Temporal difference learning

RPE

$$V(A,S) \leftarrow V(A,S) + \alpha(r(A) + \gamma V(A,S') - V(A,S))$$

Temporal difference learning



Discount factor affects gradient!!!!

$$V(A,S) \leftarrow V(A,S) + \alpha(r(A) + \gamma V(A,S') - V(A,S))$$

Temporal difference learning

Discount factor = 1

$$V(A,S) \leftarrow V(A,S) + \alpha(r(A) + \gamma V(A,S') - V(A,S))$$

PROBLEM: now all states have same value… no more gradient to follow!!!!

Temporal difference learning

Discount factor = 0.5

$$V(A,S) \leftarrow V(A,S) + \alpha(r(A) + \gamma V(A,S') - V(A,S))$$

Better… value goes TWICE as high each step along the path!

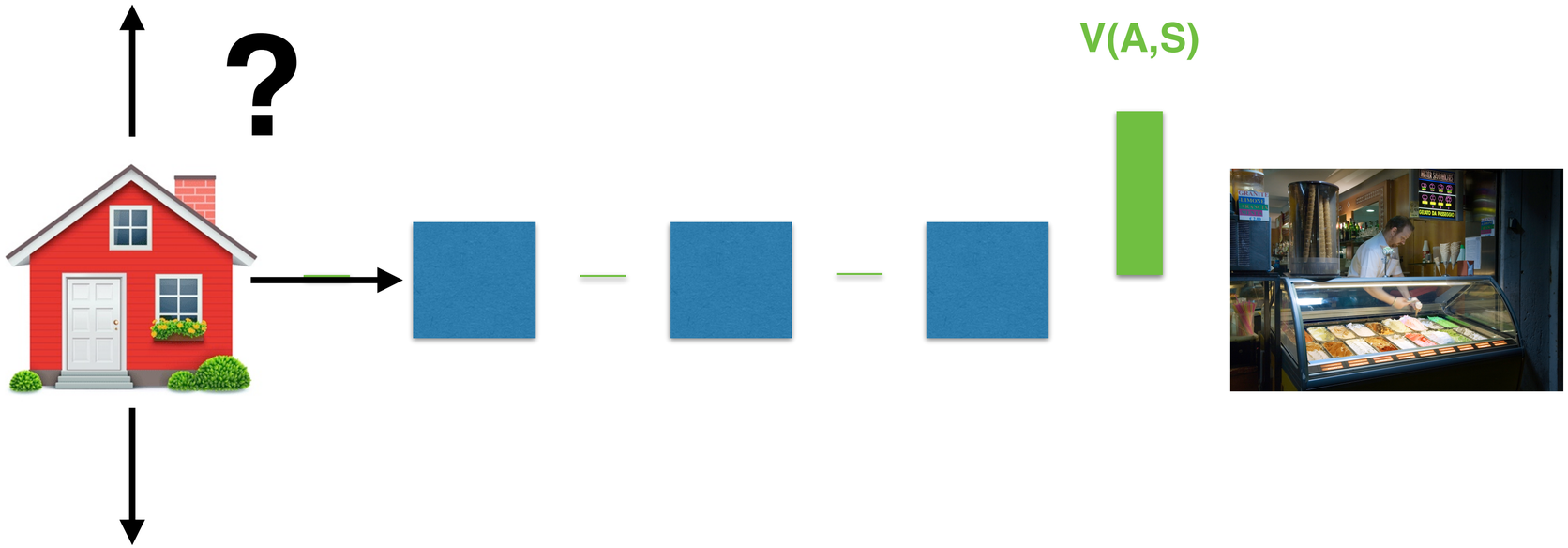
Temporal difference learning

Discount factor = 0

$$V(A,S) \leftarrow V(A,S) + \alpha(r(A) + \gamma V(A,S') - V(A,S))$$

When discount factor goes to zero, model no longer gives credit for getting to a "good state" and only gives credit for primary rewards.

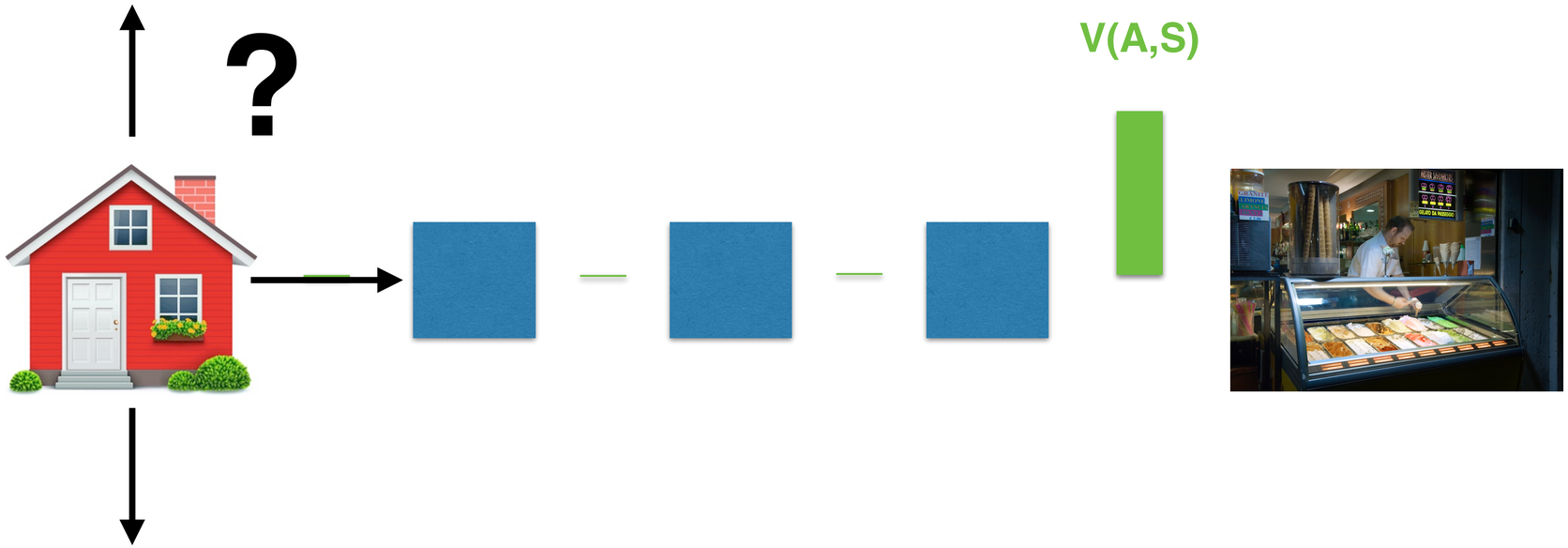
Temporal difference learning



$$V(A,S) \leftarrow V(A,S) + \alpha(r(A) + \gamma V(A,S') - V(A,S))$$

Pros: computationally cheap

Temporal difference learning

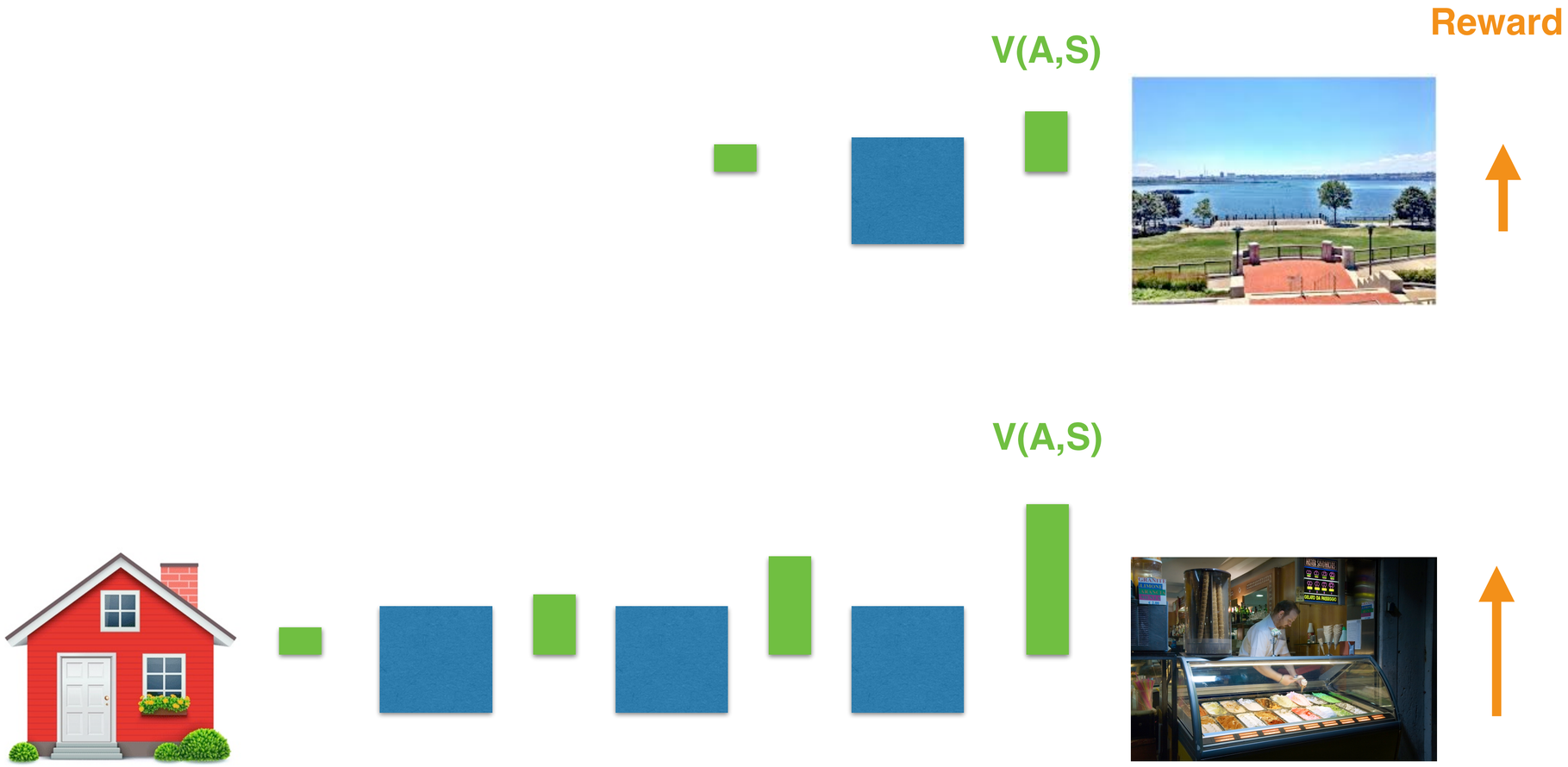


$$V(A,S) \leftarrow V(A,S) + \alpha(r(A) + \gamma V(A,S') - V(A,S))$$

Pros: computationally cheap

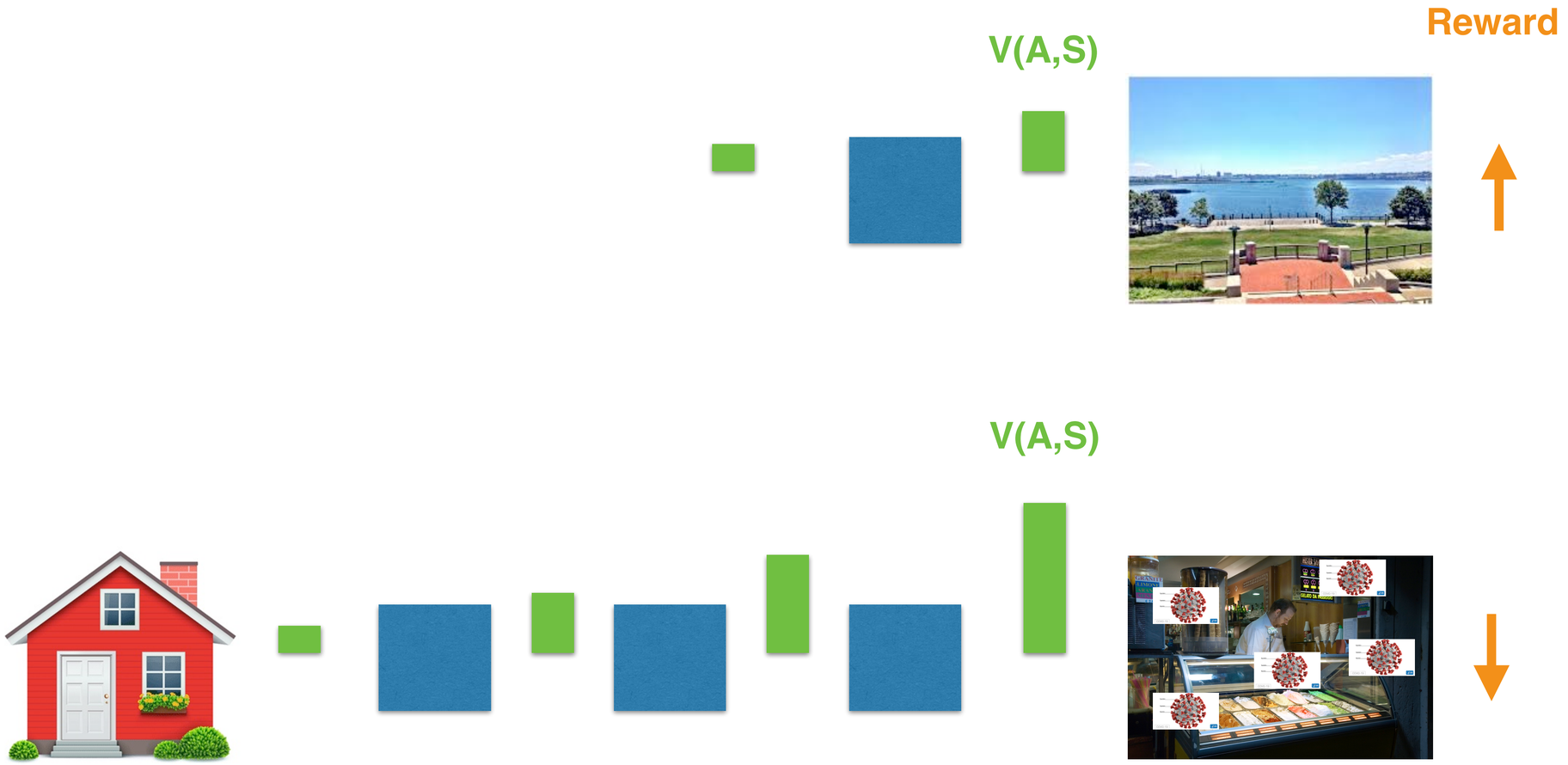
Cons…

Model Free RL only knows SA values…



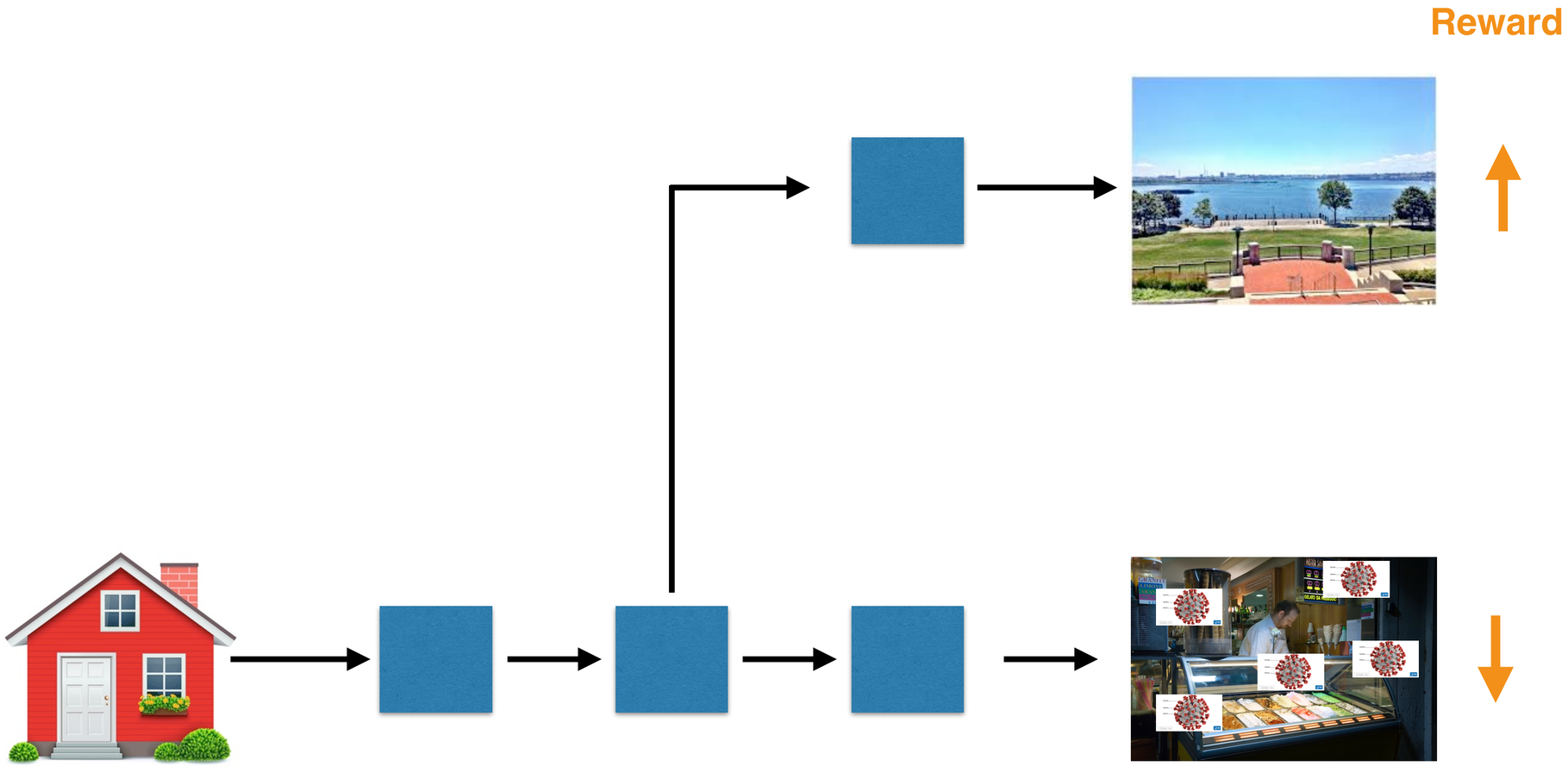
$$V(A,S) \leftarrow V(A,S) + \alpha(r(A) + \gamma V(A,S') - V(A,S))$$

Model Free RL only knows SA values…



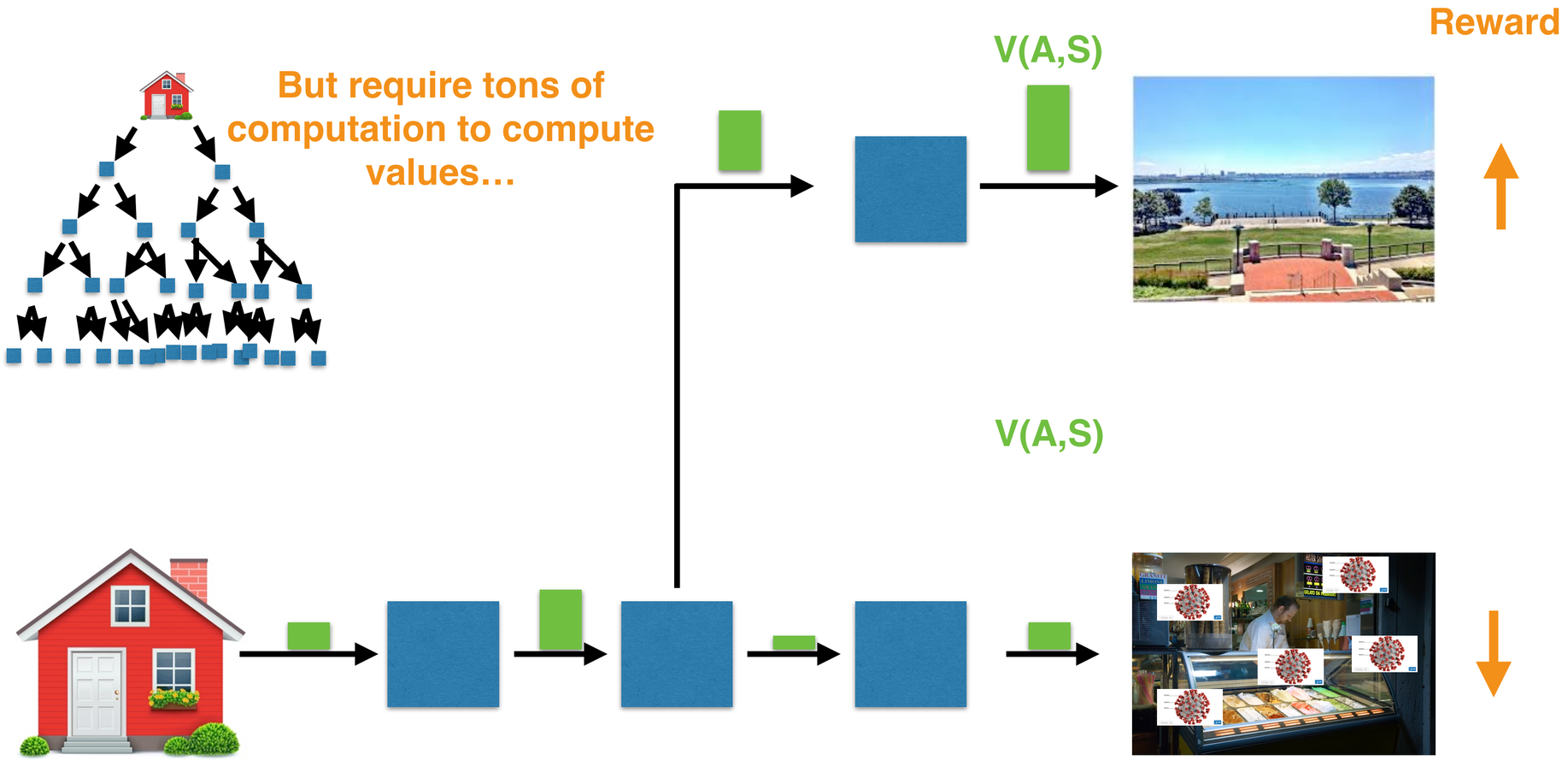
$$V(A,S) \leftarrow V(A,S) + \alpha(r(A) + \gamma V(A,S') - V(A,S))$$

Model Based methods learn/use transition structure



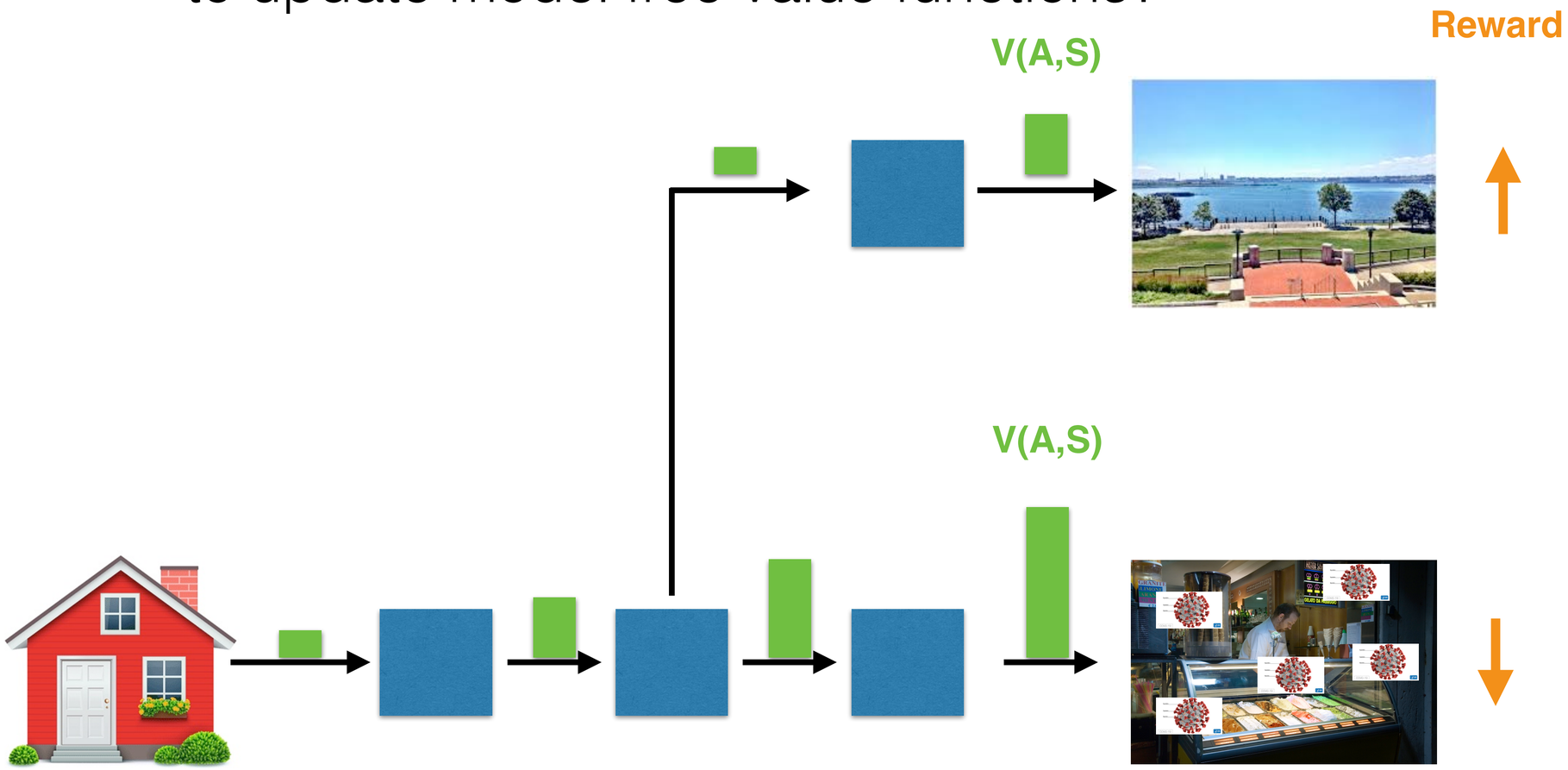
$$V(A,S) \leftarrow V(A,S) + \alpha(r(A) + \gamma V(A,S') - V(A,S))$$

Model Based methods learn/use transition structure



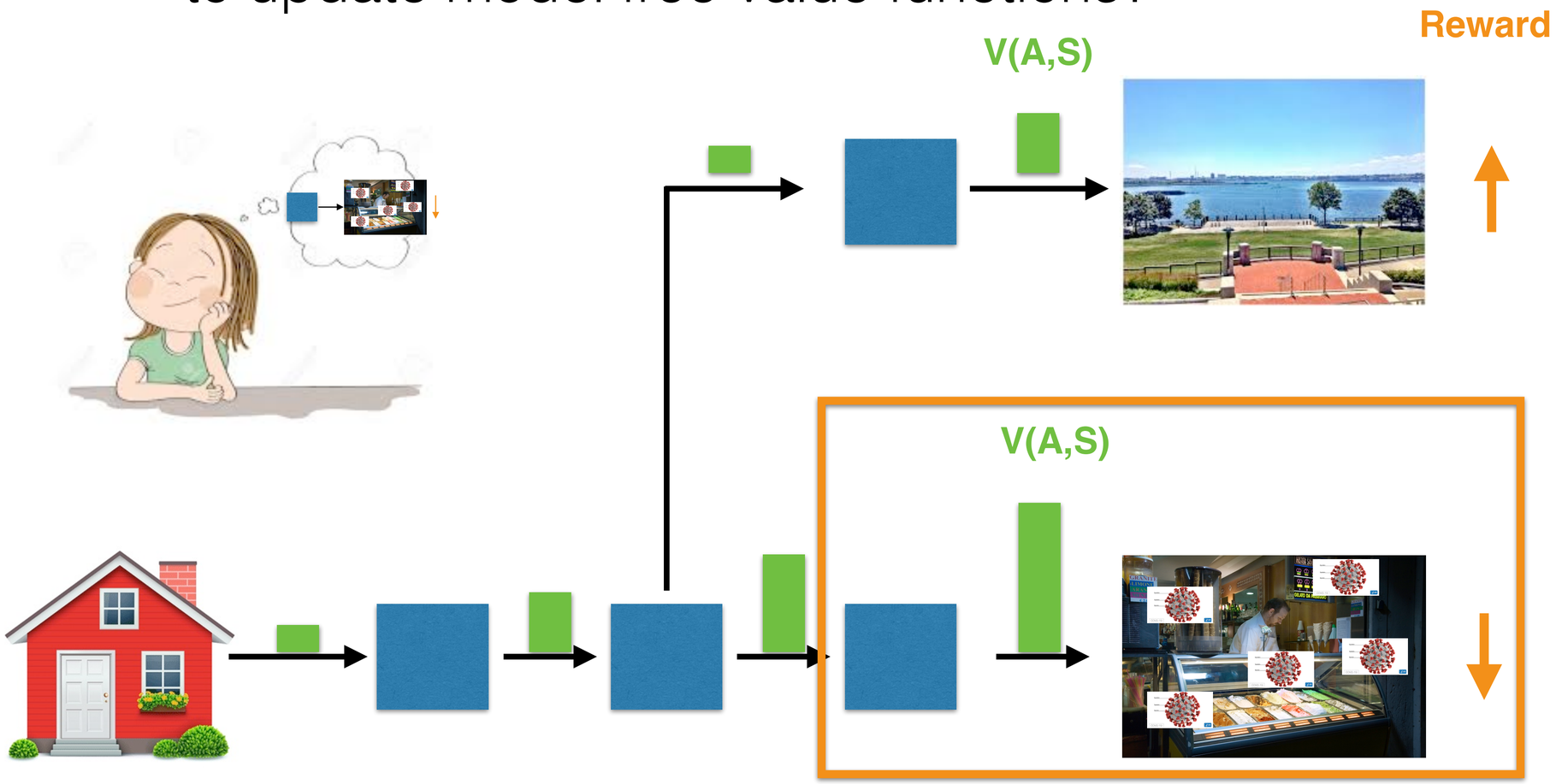
$$V(A,S) \leftarrow V(A,S) + \alpha(r(A) + \gamma V(A,S') - V(A,S))$$

What if we SIMULATE experiences from model to update model free value functions?



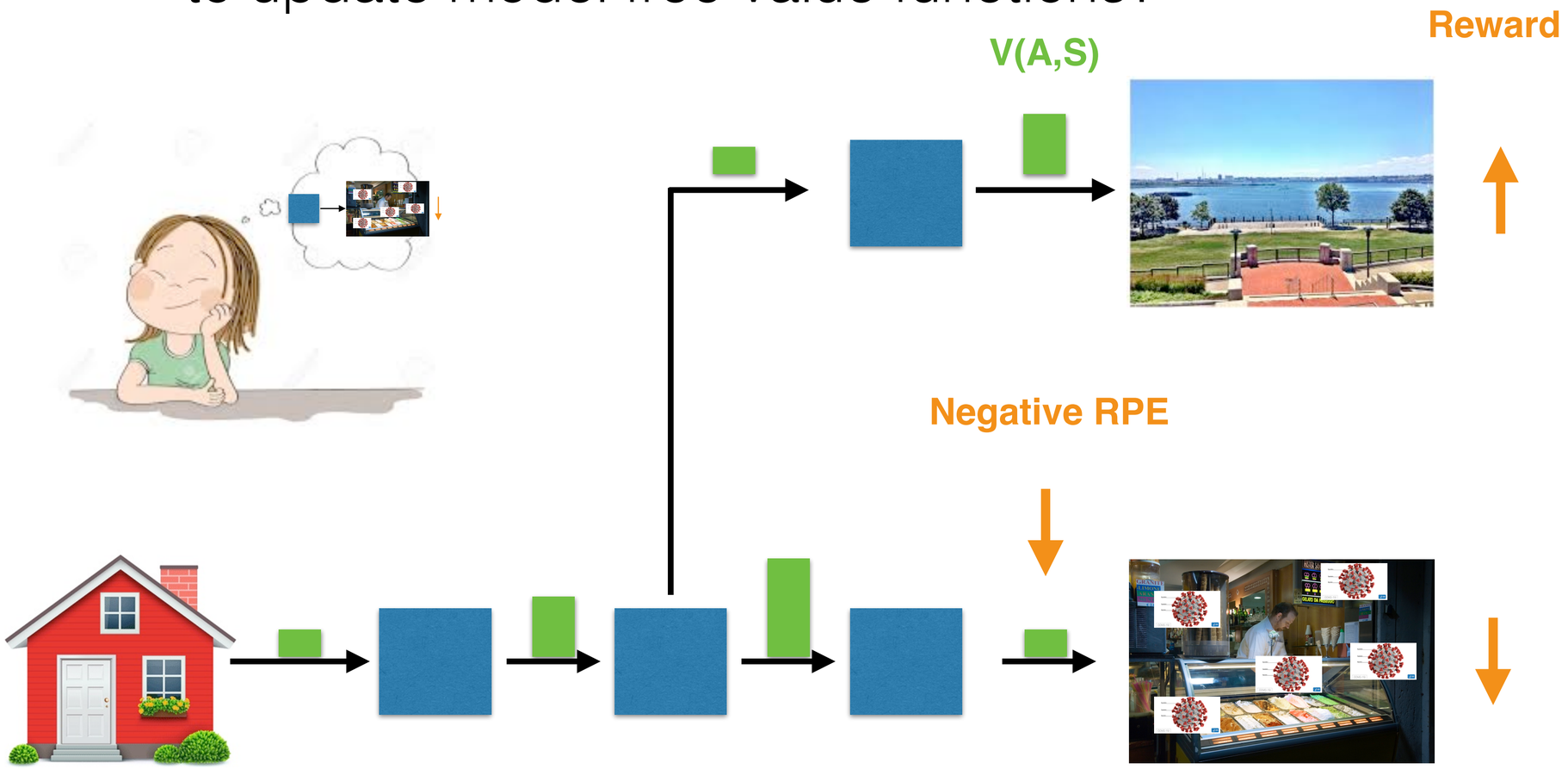
$$V(A,S) \leftarrow V(A,S) + \alpha(r(A) + \gamma V(A,S') - V(A,S))$$

What if we SIMULATE experiences from model to update model free value functions?



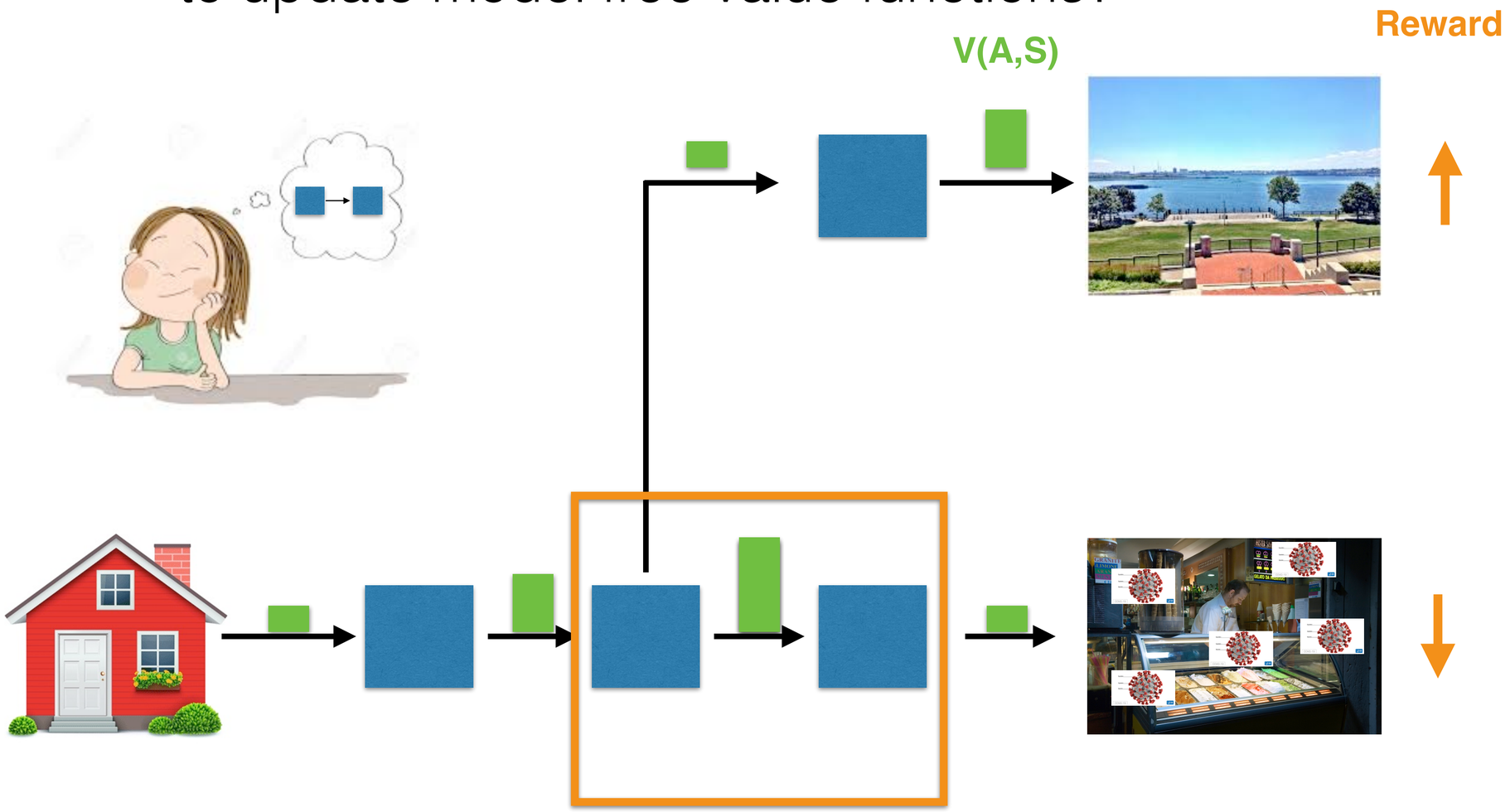
$$V(A,S) \leftarrow V(A,S) + \alpha(r(A) + \gamma V(A,S') - V(A,S))$$

What if we SIMULATE experiences from model to update model free value functions?



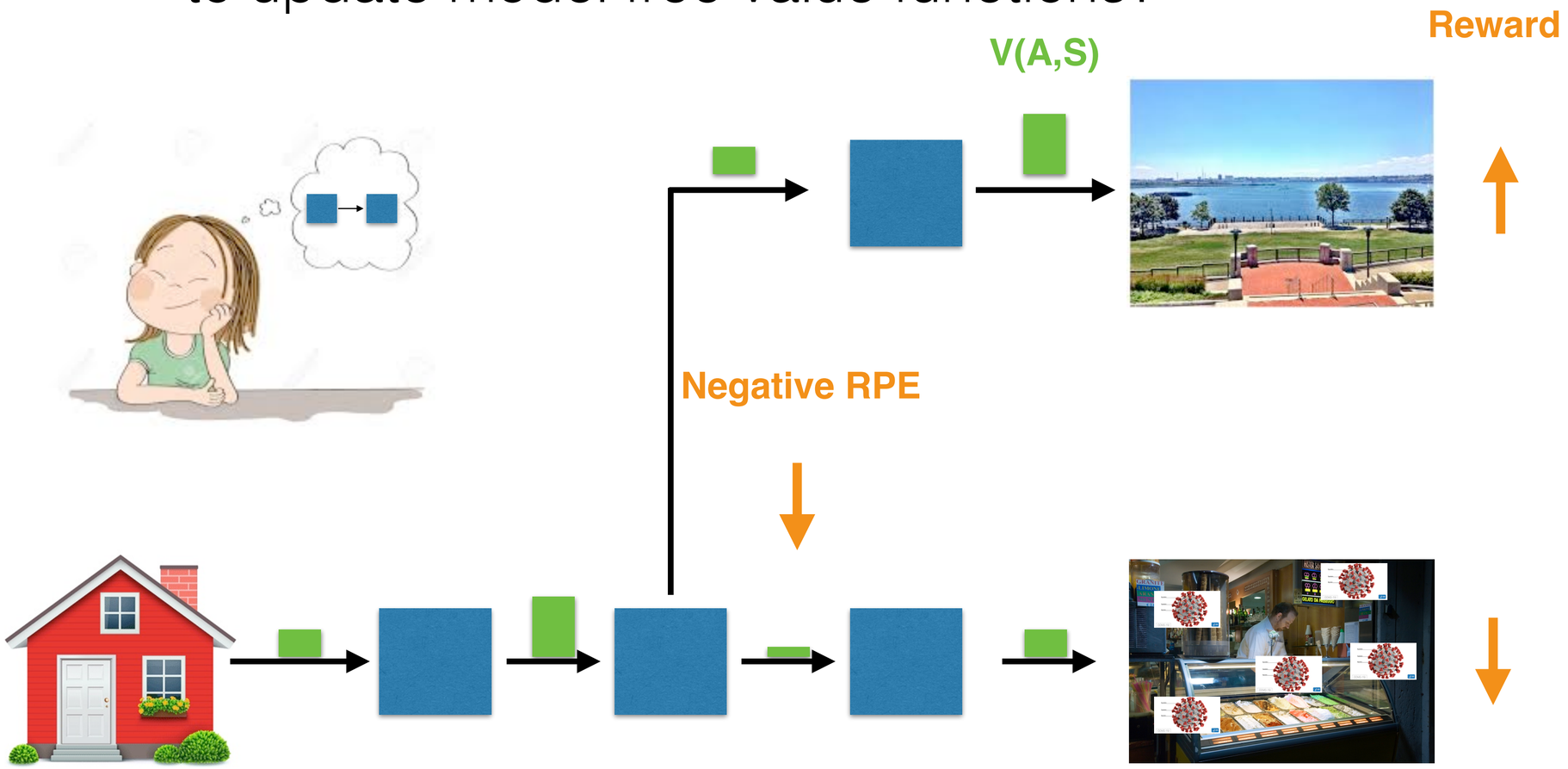
$$V(A,S) \leftarrow V(A,S) + \alpha(r(A) + \gamma V(A,S') - V(A,S))$$

What if we SIMULATE experiences from model to update model free value functions?



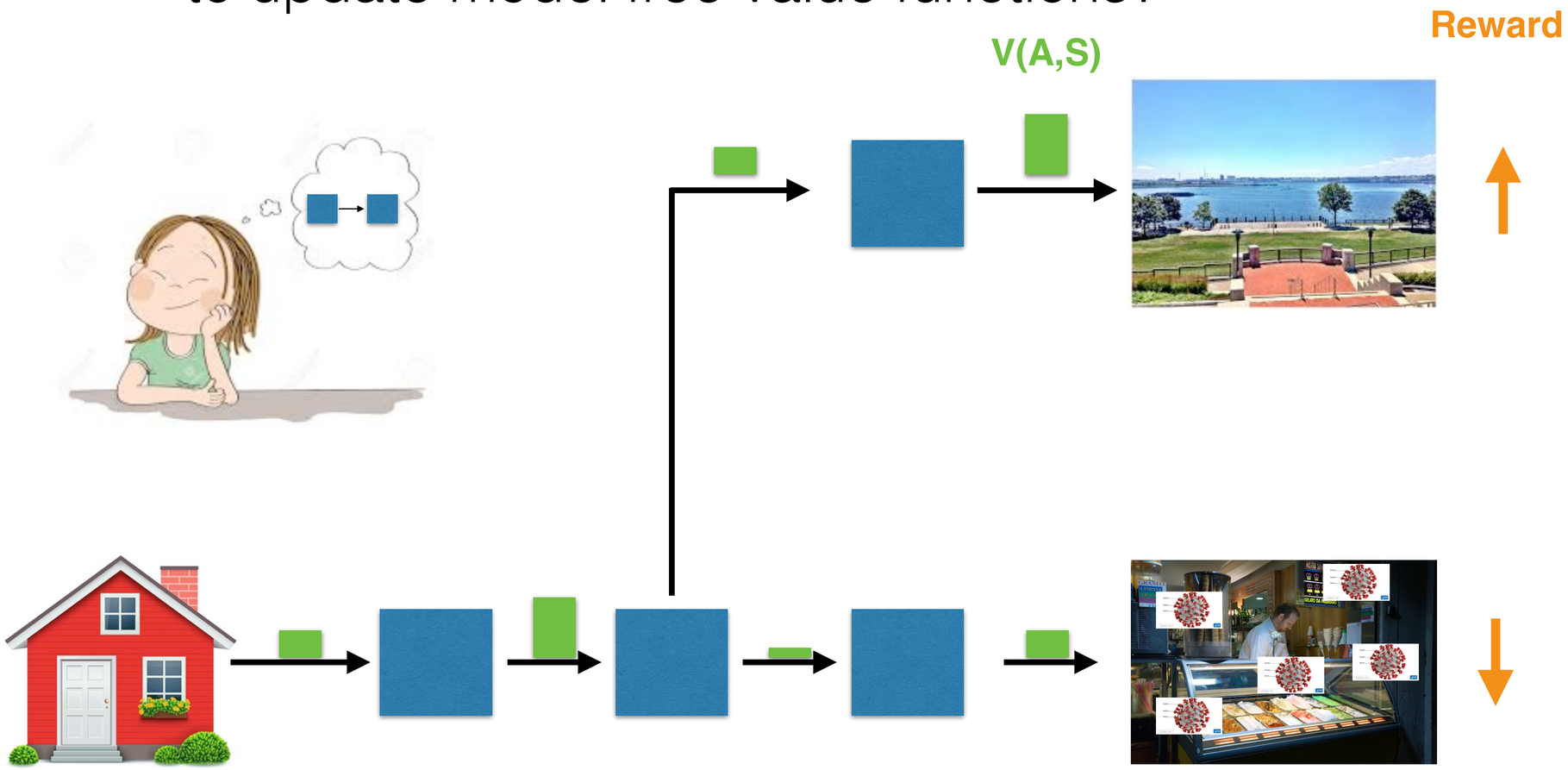
$$V(A,S) \leftarrow V(A,S) + \alpha(r(A) + \gamma V(A,S') - V(A,S))$$

What if we SIMULATE experiences from model to update model free value functions?



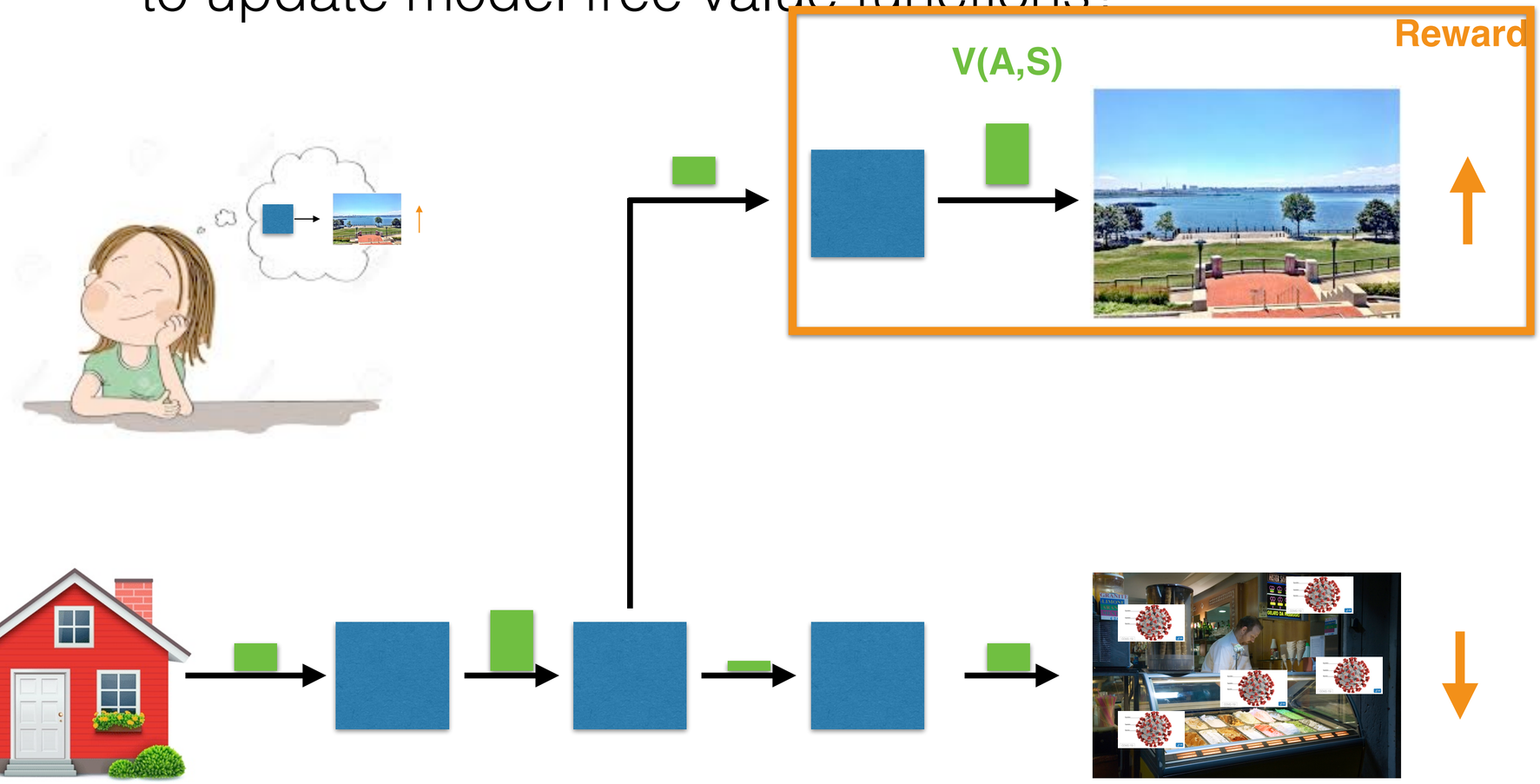
$$V(A,S) \leftarrow V(A,S) + \alpha(r(A) + \gamma V(A,S') - V(A,S))$$

What if we SIMULATE experiences from model to update model free value functions?



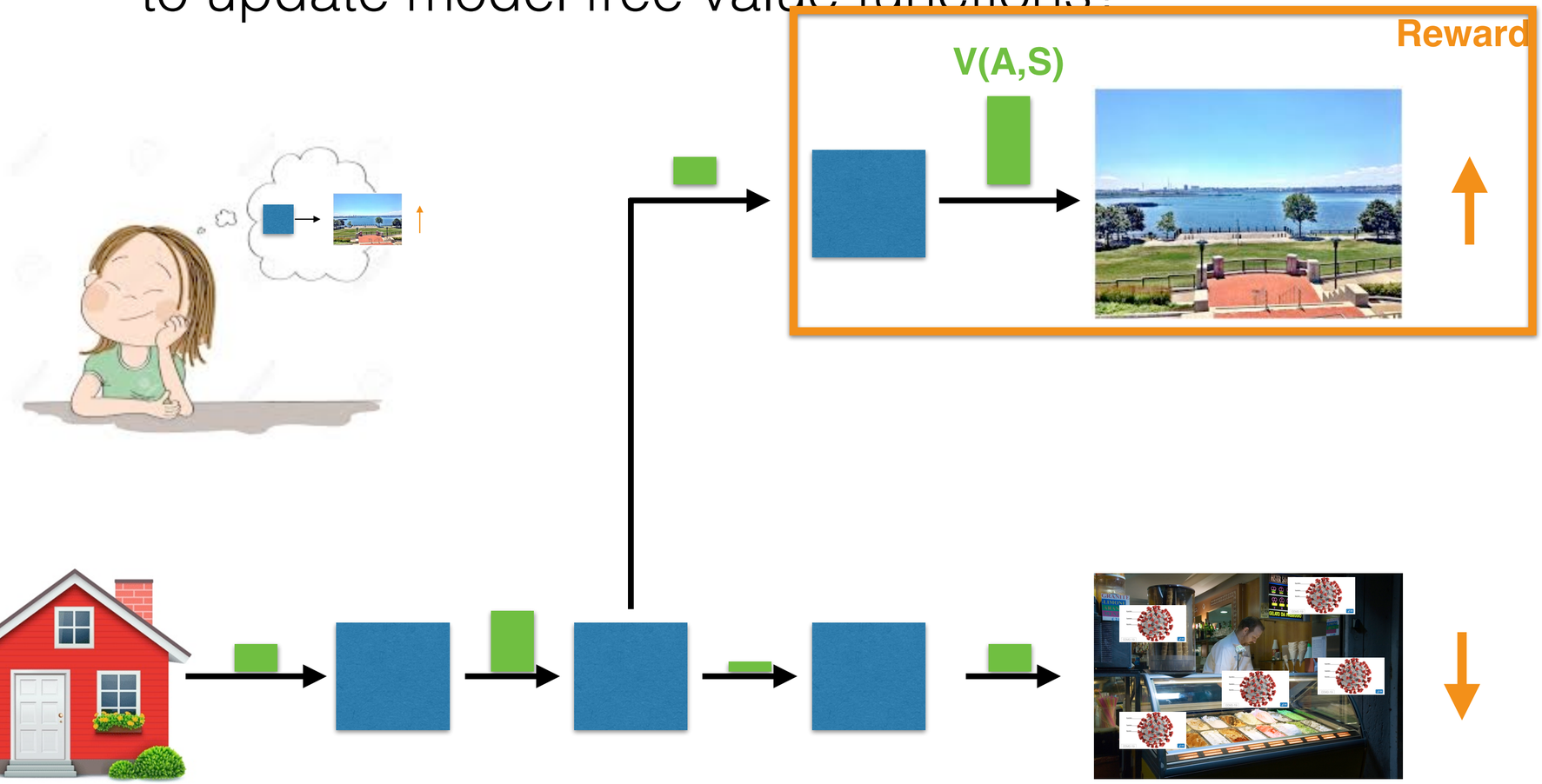
$$V(A,S) \leftarrow V(A,S) + \alpha(r(A) + \gamma V(A,S') - V(A,S))$$

What if we SIMULATE experiences from model to update model free value functions?



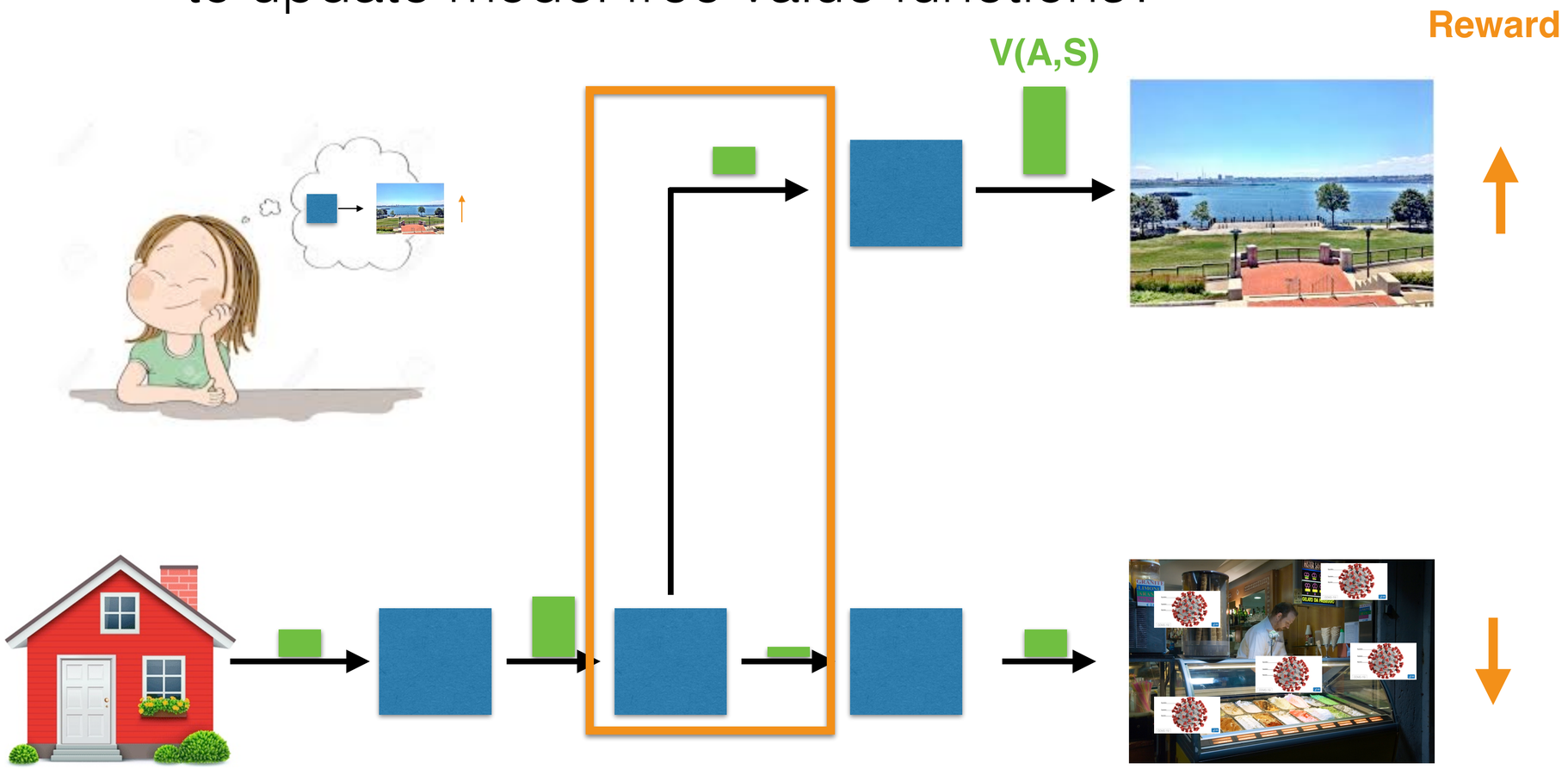
$$V(A,S) \leftarrow V(A,S) + \alpha(r(A) + \gamma V(A,S') - V(A,S))$$

What if we SIMULATE experiences from model to update model free value functions?



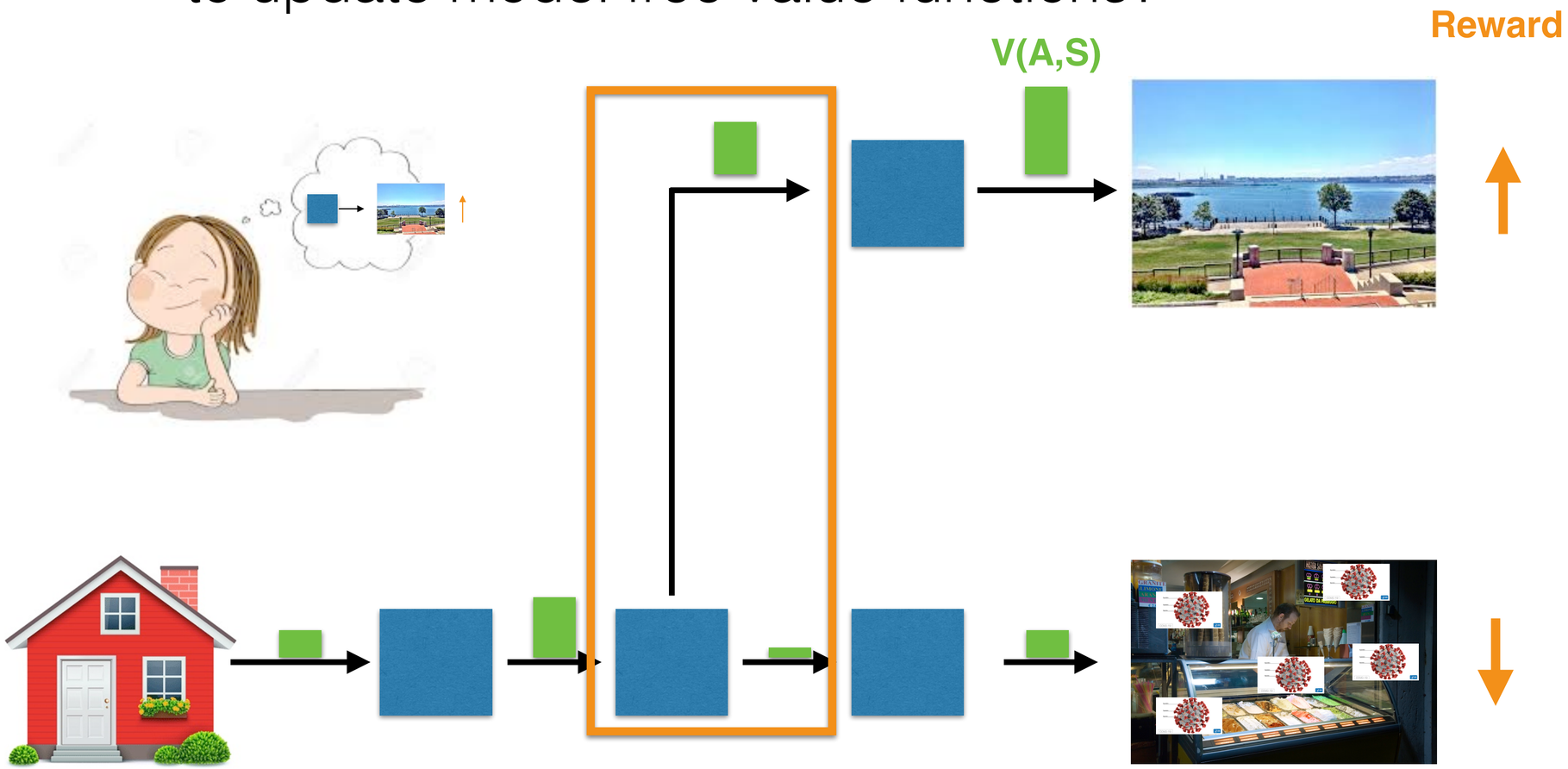
$$V(A,S) \leftarrow V(A,S) + \alpha(r(A) + \gamma V(A,S') - V(A,S))$$

What if we SIMULATE experiences from model to update model free value functions?



$$V(A,S) \leftarrow V(A,S) + \alpha(r(A) + \gamma V(A,S') - V(A,S))$$

What if we SIMULATE experiences from model to update model free value functions?



$$V(A,S) \leftarrow V(A,S) + \alpha(r(A) + \gamma V(A,S') - V(A,S))$$

Dyna-Q — integrating planning, acting, and learning

Tabular Dyna-Q

Initialize $Q(s,a)$ and $Model(s,a)$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$

Do forever:

(a) $S \leftarrow$ current (nonterminal) state

(b) $A \leftarrow \epsilon$-greedy(S,Q)

(c) Execute action A; observe resultant reward, R, and state, S'

(d) $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$

Q learning

Sutton and Barto

Dyna-Q — integrating planning, acting, and learning

Tabular Dyna-Q

Initialize $Q(s,a)$ and $Model(s,a)$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$

Do forever:

(a) $S \leftarrow$ current (nonterminal) state

(b) $A \leftarrow \epsilon$-greedy(S,Q)

(c) Execute action A; observe resultant reward, R, and state, S'

(d) $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$

(e) $Model(S,A) \leftarrow R, S'$ (assuming deterministic environment)

Q learning

Learn model of state transitions

Sutton and Barto

Dyna-Q — integrating planning, acting, and learning

Tabular Dyna-Q

Initialize $Q(s,a)$ and $Model(s,a)$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$

Do forever:

(a) $S \leftarrow$ current (nonterminal) state

(b) $A \leftarrow \epsilon$-greedy(S,Q)

(c) Execute action A; observe resultant reward, R, and state, S'

(d) $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$

(e) $Model(S,A) \leftarrow R, S'$ (assuming deterministic environment)

(f) Repeat n times:

$S \leftarrow$ random previously observed state

$A \leftarrow$ random action previously taken in S

$R, S' \leftarrow Model(S,A)$

$Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$

Q learning

Learn model of state transitions

+ simulate a bunch of state transitions and update state action values

Sutton and Barto

Summary

- MDPs provide a flexible framework to formalize sequential decision problems
- Solving an MDP requires finding a policy that yields high long run rewards
- There are many strategies to solve an MDP, and these strategies often differ in computational cost/flexibility
- Planning/model based learning uses model of environment to compute action values — accurate, but costly
- Model free RL learns action values, but doesn't know where its actions will take it — cheap, but inflexible

Questions?