Section 4:
MDPs and Reinforcement Learning

Will Deaderick, Chris Waites
CS 221 - Autumn 2019

Overview of today's section:

- Recap various policy learning algorithms (and when to use them) at a high-level
- SARSA vs. Q-Learning intuition
- Introduce deep reinforcement learning
- See deep RL in action on an Atari game!
- Cool RL extensions

Markov Decision Processes (MDPs)

A MDP is a search problem where **transitions are random** and instead of minimizing cost, we are **maximizing reward**.

Definition: Markov decision process

States: the set of states

$s_{\text{start}} \in \text{States}$: starting state

$\text{Actions}(s)$: possible actions from state s

$T(s, a, s')$: probability of s' if take action a in state s

$\text{Reward}(s, a, s')$: reward for the transition (s, a, s')

$\text{IsEnd}(s)$: whether at end of game

$0 \leq \gamma \leq 1$: discount factor (default: 1)

Reward Specification

Reward Specification

- For the majority of MDP applications, transition dynamics come from some real world process (e.g. stock market, traffic, plain old physics)

Reward Specification

- For the majority of MDP applications, transition dynamics come from some real world process (e.g. stock market, traffic, plain old physics)

- But rewards are chosen by **you** - this is more art than science.

Reward Specification

- For the majority of MDP applications, transition dynamics come from some real world process (e.g. stock market, traffic, plain old physics)

- But rewards are chosen by **you** - this is more art than science.

Reward sparsity

Reward Specification

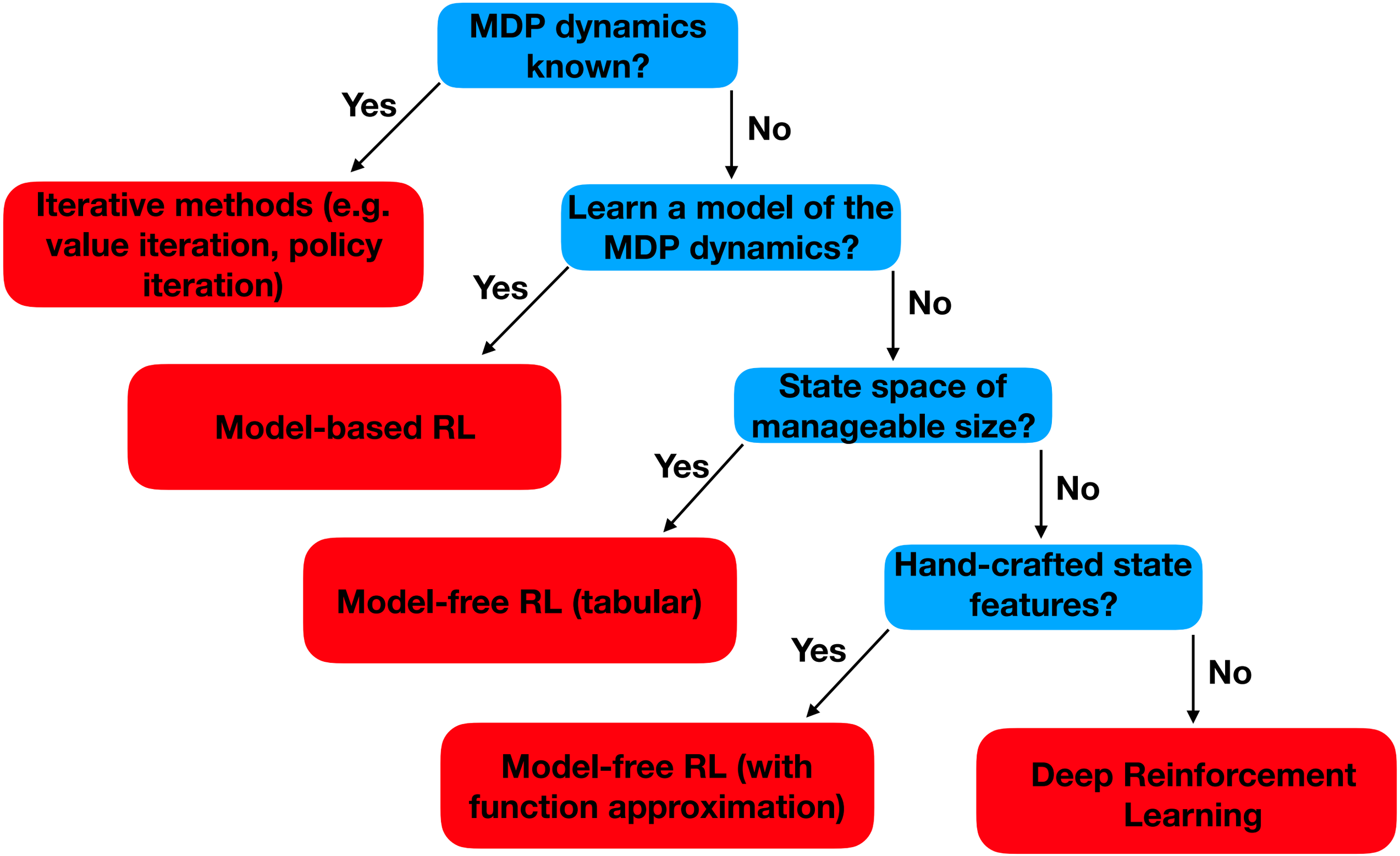
- For the majority of MDP applications, transition dynamics come from some real world process (e.g. stock market, traffic, plain old physics)

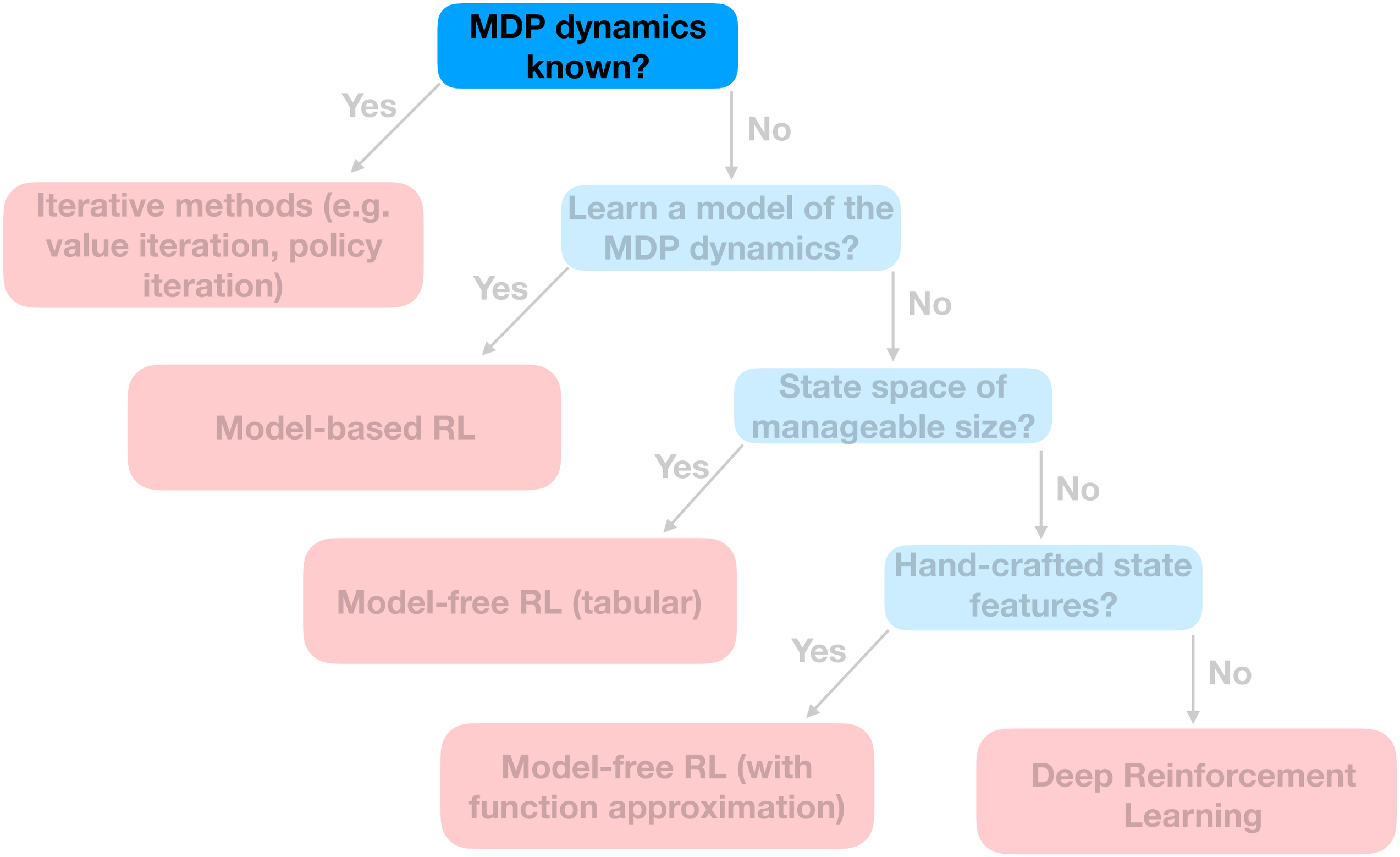
- But rewards are chosen by **you** - this is more art than science.

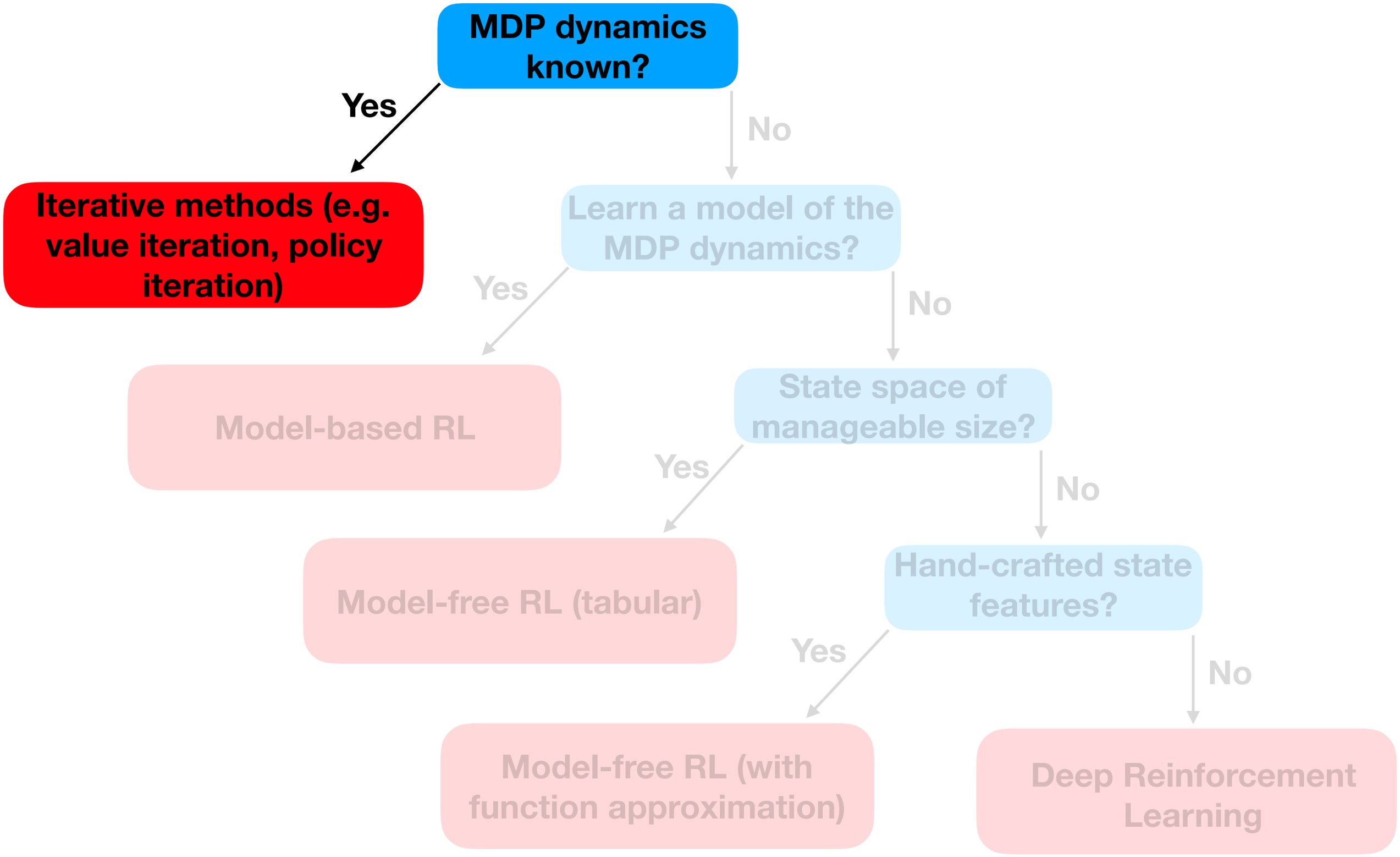
Reward sparsity

Goal alignment

Learning policies in MDPs


MDP dynamics known?
Yes
No
Iterative methods (e.g. value iteration, policy iteration)
Learn a model of the MDP dynamics?
Yes
No
Model-based RL
State space of manageable size?
Yes
No
Model-free RL (tabular)
Hand-crafted state features?
Yes
No
Model-free RL (with function approximation)
Deep Reinforcement Learning


MDP dynamics known?
Yes
No
Iterative methods (e.g. value iteration, policy iteration)
Learn a model of the MDP dynamics?
Yes
No
Model-based RL
State space of manageable size?
Yes
No
Model-free RL (tabular)
Hand-crafted state features?
Yes
No
Model-free RL (with function approximation)
Deep Reinforcement Learning


MDP dynamics known?
Yes
No
Iterative methods (e.g. value iteration, policy iteration)
Learn a model of the MDP dynamics?
Yes
No
Model-based RL
State space of manageable size?
Yes
No
Model-free RL (tabular)
Hand-crafted state features?
Yes
No
Model-free RL (with function approximation)
Deep Reinforcement Learning

Iterative methods for when dynamics are known

- Pseudocode:

1. Policy Evaluation: $\pi \rightarrow V_\pi$
2. Policy Improvement: $\mathrm{V}_\pi \rightarrow \pi'$

} **Repeat until convergence to optimal policy**

Iterative methods for when dynamics are known

- Pseudocode:

1. Policy Evaluation: $\pi \rightarrow V_\pi$
2. Policy Improvement: $V_\pi \rightarrow \pi'$

} **Repeat until convergence to optimal policy**

“Generalized Policy Iteration”

(See Sutton & Barto Ch. 4)

Iterative methods for when dynamics are known

- Pseudocode:

    1. Policy Evaluation: iterate the value equations
    2. Policy Improvement: act optimally with respect to value estimate

Iterative methods for when dynamics are known

- Pseudocode:

    1. Policy Evaluation: iterate the value equations
    2. Policy Improvement: act optimally with respect to value estimate

Iterative methods differ only in how they specify and interleave these two steps!

Iterative methods for when dynamics are known

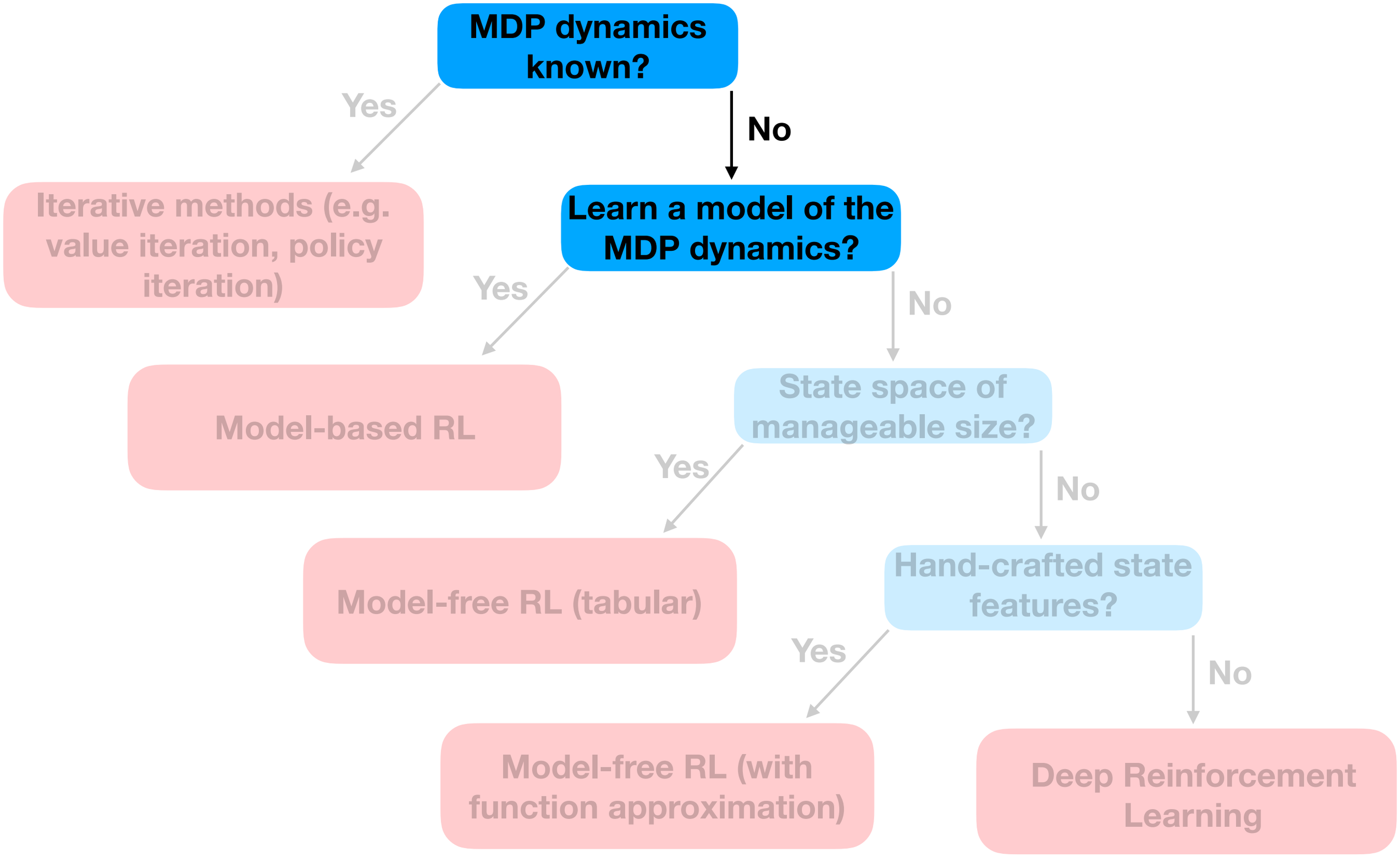
- Pseudocode:

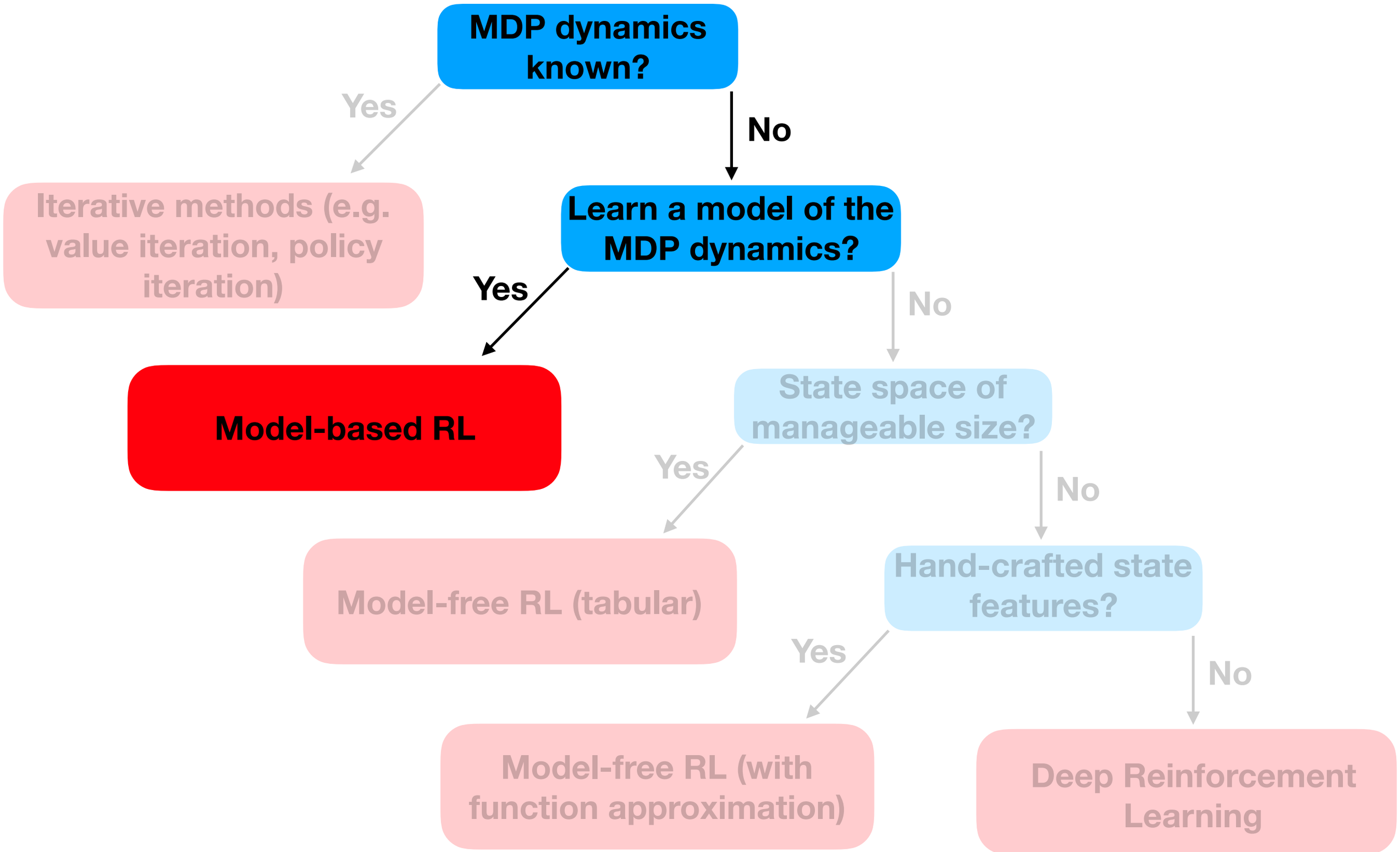
    1. Policy Evaluation: iterate the value equations
    2. Policy Improvement: act optimally with respect to value estimate

Example: Value Iteration

$$V_{\text{opt}}^{(t)}(s) \leftarrow \max_{a \in \text{Actions}(s)} \sum_{s'} T(s, a, s')[\text{Reward}(s, a, s') + \gamma V_{\text{opt}}^{(t-1)}(s')]$$

Improvement **Evaluation**


MDP dynamics known?
Yes
No
Iterative methods (e.g. value iteration, policy iteration)
Learn a model of the MDP dynamics?
Yes
No
Model-based RL
State space of manageable size?
Yes
No
Model-free RL (tabular)
Hand-crafted state features?
Yes
No
Model-free RL (with function approximation)
Deep Reinforcement Learning


MDP dynamics known?
Yes
No
Iterative methods (e.g. value iteration, policy iteration)
Learn a model of the MDP dynamics?
Yes
No
Model-based RL
State space of manageable size?
Yes
No
Model-free RL (tabular)
Hand-crafted state features?
Yes
No
Model-free RL (with function approximation)
Deep Reinforcement Learning

Model-based RL

$$\hat{T}(s,a,s') = \frac{\#\text{ times } (s,a,s') \text{ occurs}}{\#\text{ times } (s,a) \text{ occurs}}$$

$$\widehat{\text{Reward}}(s,a,s') = \text{average of } r \text{ in } (s,a,r,s')$$

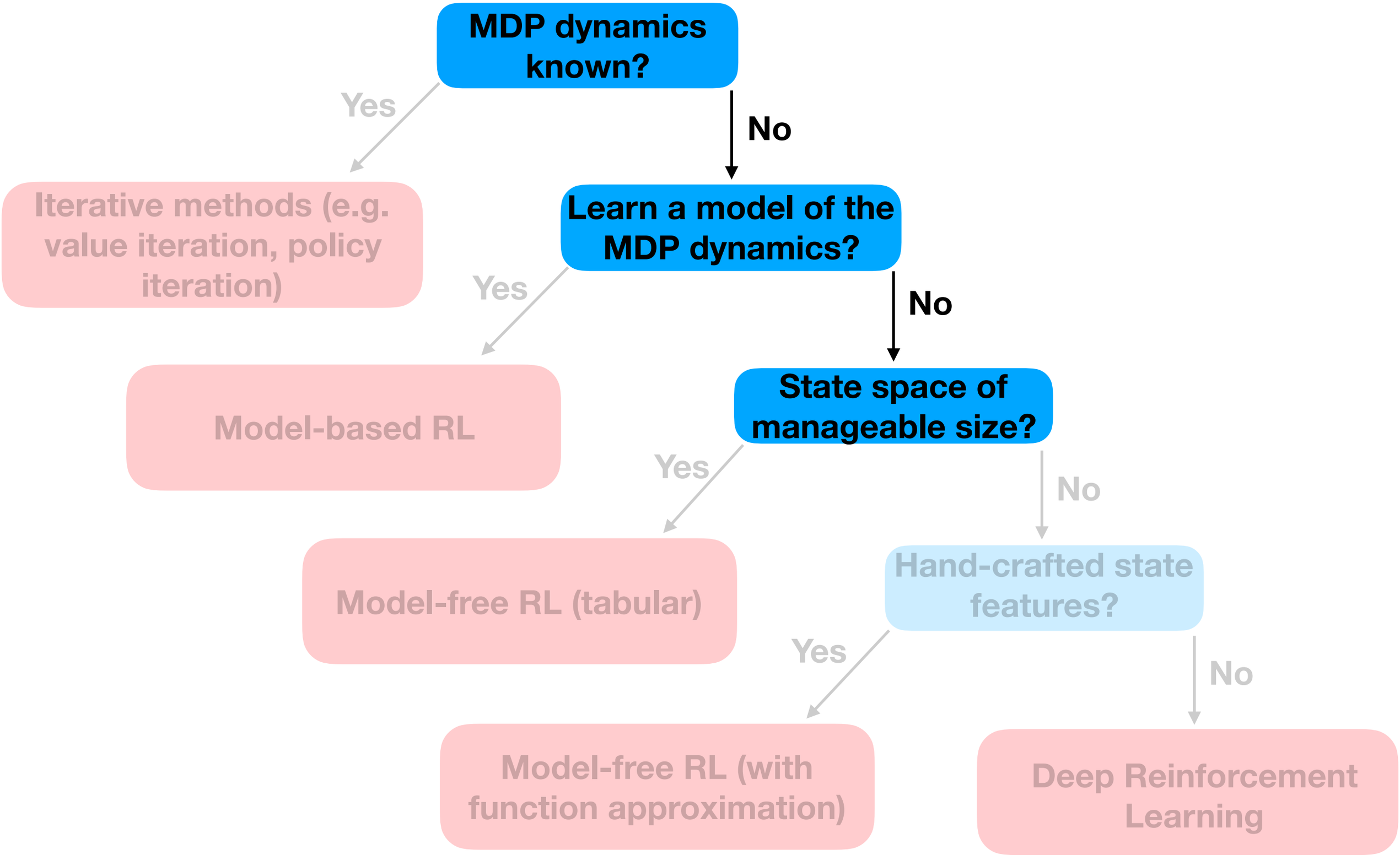
Model-based RL

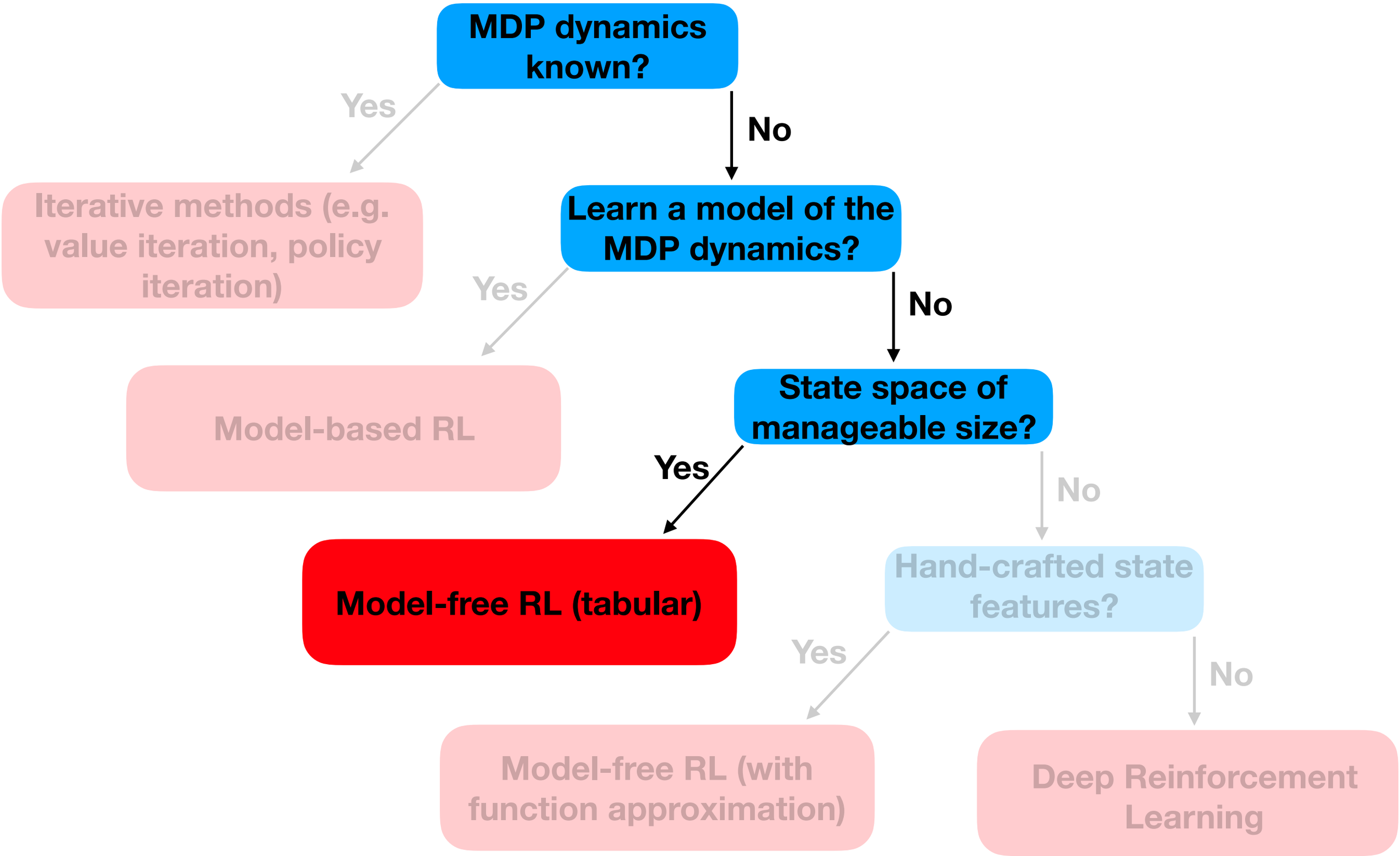
$$\hat{T}(s,a,s') = \frac{\#\text{ times } (s,a,s') \text{ occurs}}{\#\text{ times } (s,a) \text{ occurs}}$$

$$\widehat{\text{Reward}}(s,a,s') = \text{average of } r \text{ in } (s,a,r,s')$$

Now we have an estimate of the MDP dynamics - use an iterative method as if our estimate were exact!

But if our estimate is bad…


MDP dynamics known?
Yes
No
Iterative methods (e.g. value iteration, policy iteration)
Learn a model of the MDP dynamics?
Yes
No
Model-based RL
State space of manageable size?
Yes
No
Model-free RL (tabular)
Hand-crafted state features?
Yes
No
Model-free RL (with function approximation)
Deep Reinforcement Learning


MDP dynamics known?
Yes
No
Iterative methods (e.g. value iteration, policy iteration)
Learn a model of the MDP dynamics?
Yes
No
Model-based RL
State space of manageable size?
Yes
No
Model-free RL (tabular)
Hand-crafted state features?
Yes
No
Model-free RL (with function approximation)
Deep Reinforcement Learning

Model-free RL (tabular)

Starting in state s_0, apply policy π

s_0

$a_1 \sim \pi(s_0)$

$s_1 \sim T(s_0, a_1, .)$

$r_1 \sim P_R(s_0, a_1, s_1)$

$a_2 \sim \pi(s_1)$

$s_2 \sim T(s_1, a_2, .)$

$r_2 \sim P_R(s_1, a_2, s_2)$

$\vdots$

Model-free RL (tabular)

- For each (s, a) pair, maintain a value $\hat{Q}(s, a)$
- Whenever we encounter (s, a), update $Q(s, a)$ towards a "target" value computed using the rewards that follow
- Model-free RL algorithms primarily differ based on how they compute these "target" values.

Starting in state s_0, apply policy π

s_0

$a_1 \sim \pi(s_0)$

$s_1 \sim T(s_0, a_1, .)$

$r_1 \sim P_R(s_0, a_1, s_1)$

$a_2 \sim \pi(s_1)$

$s_2 \sim T(s_1, a_2, .)$

$r_2 \sim P_R(s_1, a_2, s_2)$

$\vdots$

Examples of target values

- **<u>Monte Carlo</u>:**
  - Sum of all (discounted) rewards that follow (s, a)
- **<u>SARSA,</u>** for transition (s, a, r, s', a') :
  - $r + \gamma \hat{Q}(s', a')$
- **<u>Q-Learning</u>:**
  - $r + \gamma \max_{a'} \hat{Q}(s', a')$

How are we selecting actions?

- Popular choice: **epsilon-greedy**

$$\pi(s) = \begin{cases} \text{argmax}_a \hat{Q}(s,a) & \text{with prob } 1-\epsilon \\ \text{random from Actions(s)} & \text{with prob } \epsilon \end{cases}$$

How are we selecting actions?

- Popular choice: **epsilon-greedy**

$$\pi(s) = \begin{cases} \text{argmax}_a \hat{Q}(s,a) & \text{with prob } 1-\epsilon \\ \text{random from Actions(s)} & \text{with prob } \epsilon \end{cases}$$

- Policy -> Experience -> Value Function -> Policy -> …

How are we selecting actions?

- Popular choice: **epsilon-greedy**

$$\pi(s) = \begin{cases} \text{argmax}_a \hat{Q}(s,a) & \text{with prob } 1-\epsilon \\ \text{random from Actions(s)} & \text{with prob } \epsilon \end{cases}$$

- Policy -> Experience -> Value Function -> Policy -> …

“Generalized Policy Iteration”

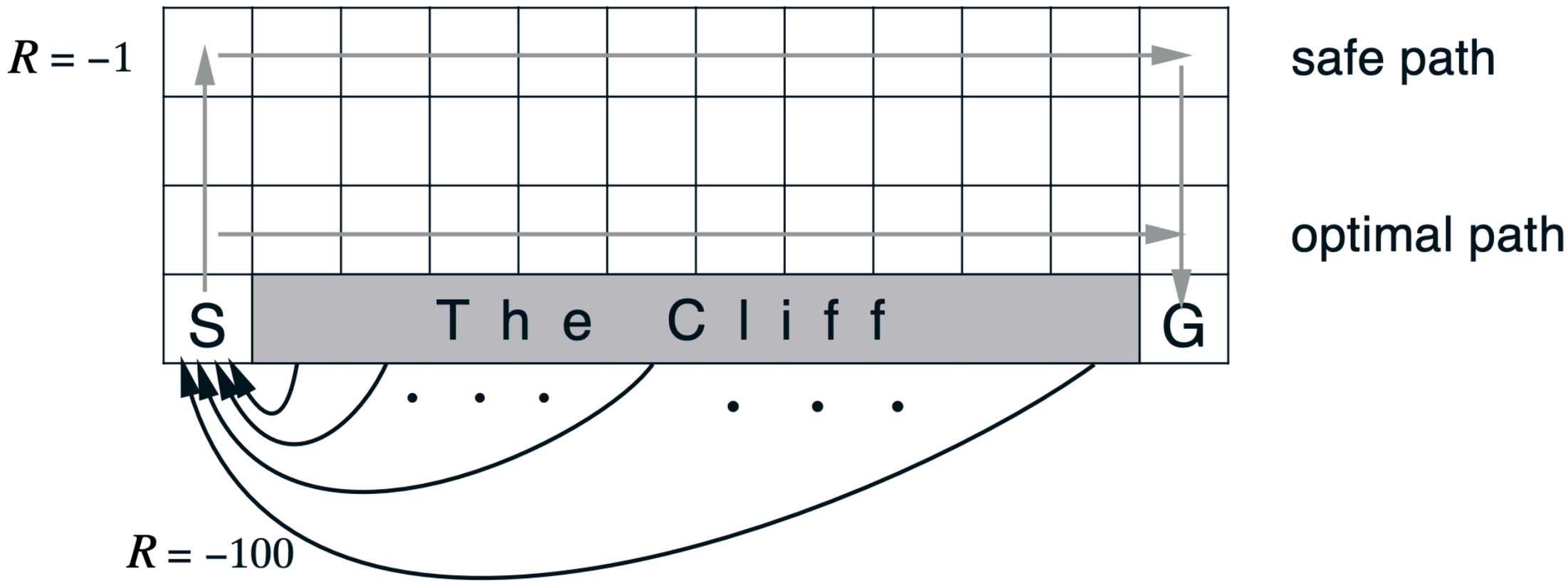
Examples of target values

- **<u>Monte Carlo</u>:**
  - Sum of all (discounted) rewards that follow (s, a)
- **<u>SARSA,</u>** for transition (s, a, r, s', a') :
  - $r + \gamma \hat{Q}(s', a')$
- **<u>Q-Learning</u>:**
  - $r + \gamma \max_{a'} \hat{Q}(s', a')$

Many other forms of target values exist!

(See Sutton & Barto Ch. 5-7)

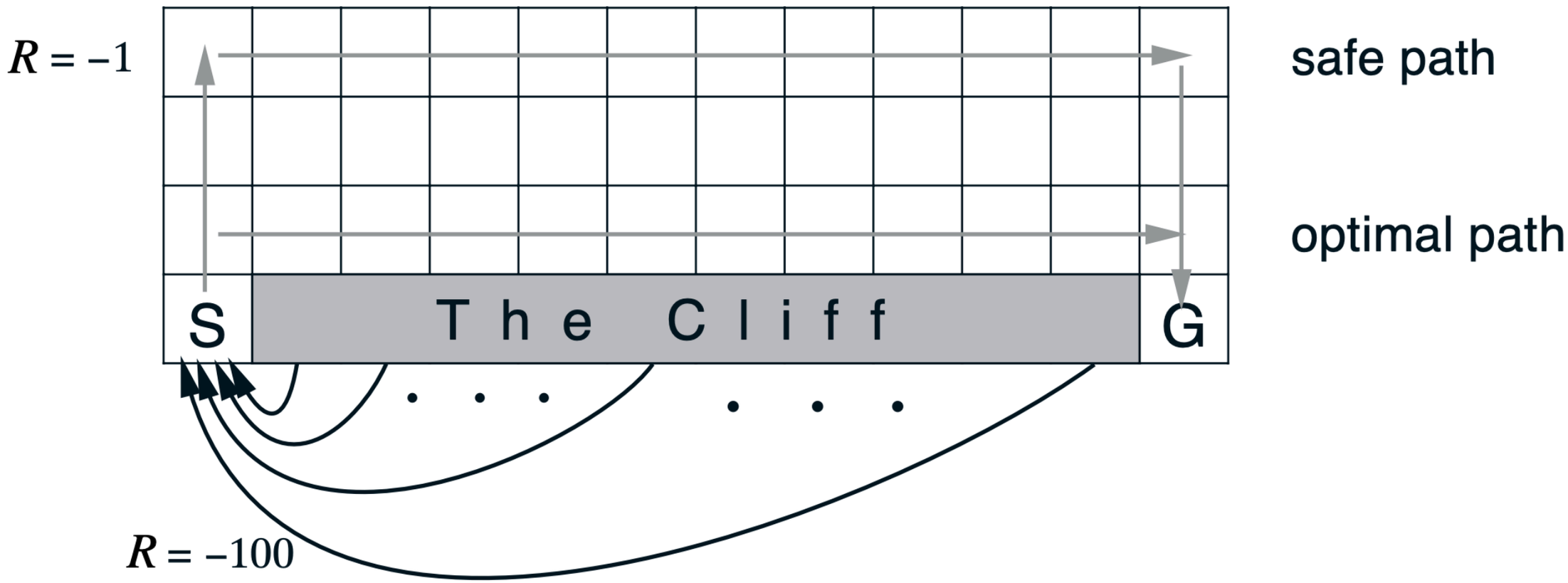
SARSA v. Q-Learning



- **Assume both SARSA and Q-learning use an epsilon-greedy policy**

Sutton and Barto. Reinforcement Learning: An Introduction.

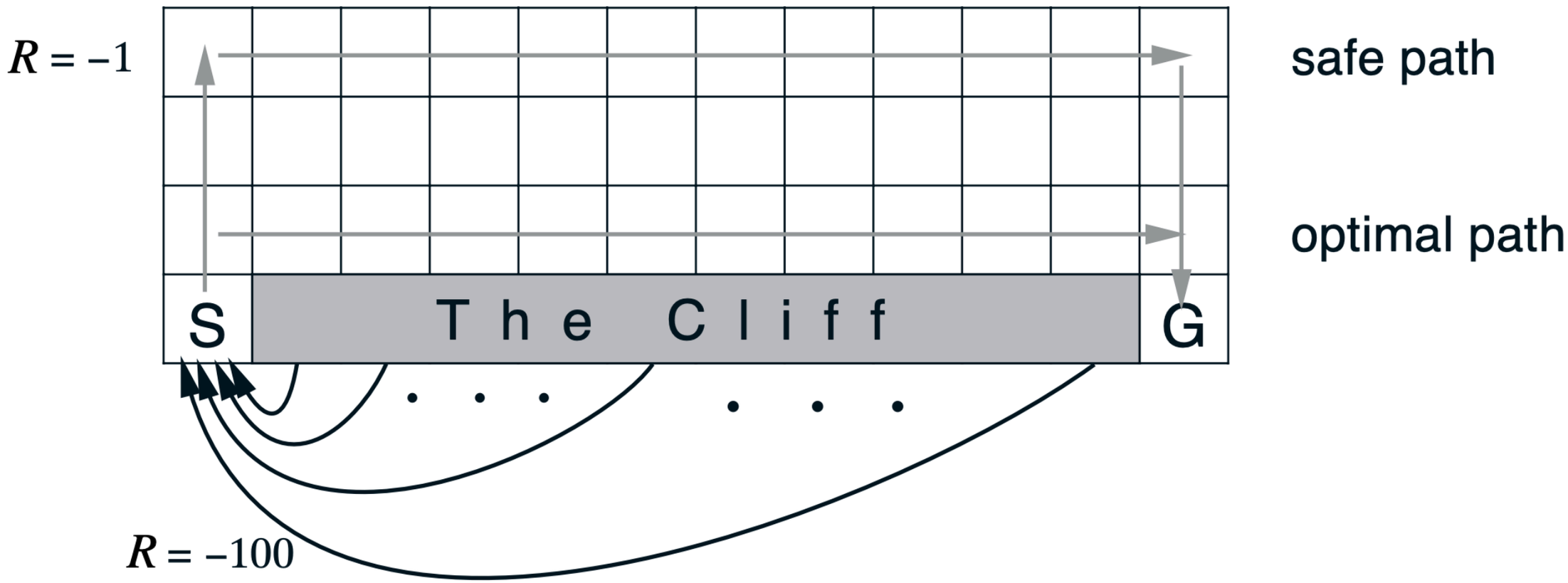
SARSA v. Q-Learning



- **Assume both SARSA and Q-learning use an epsilon-greedy policy**
- **SARSA outperforms Q-Learning in terms of online performance**

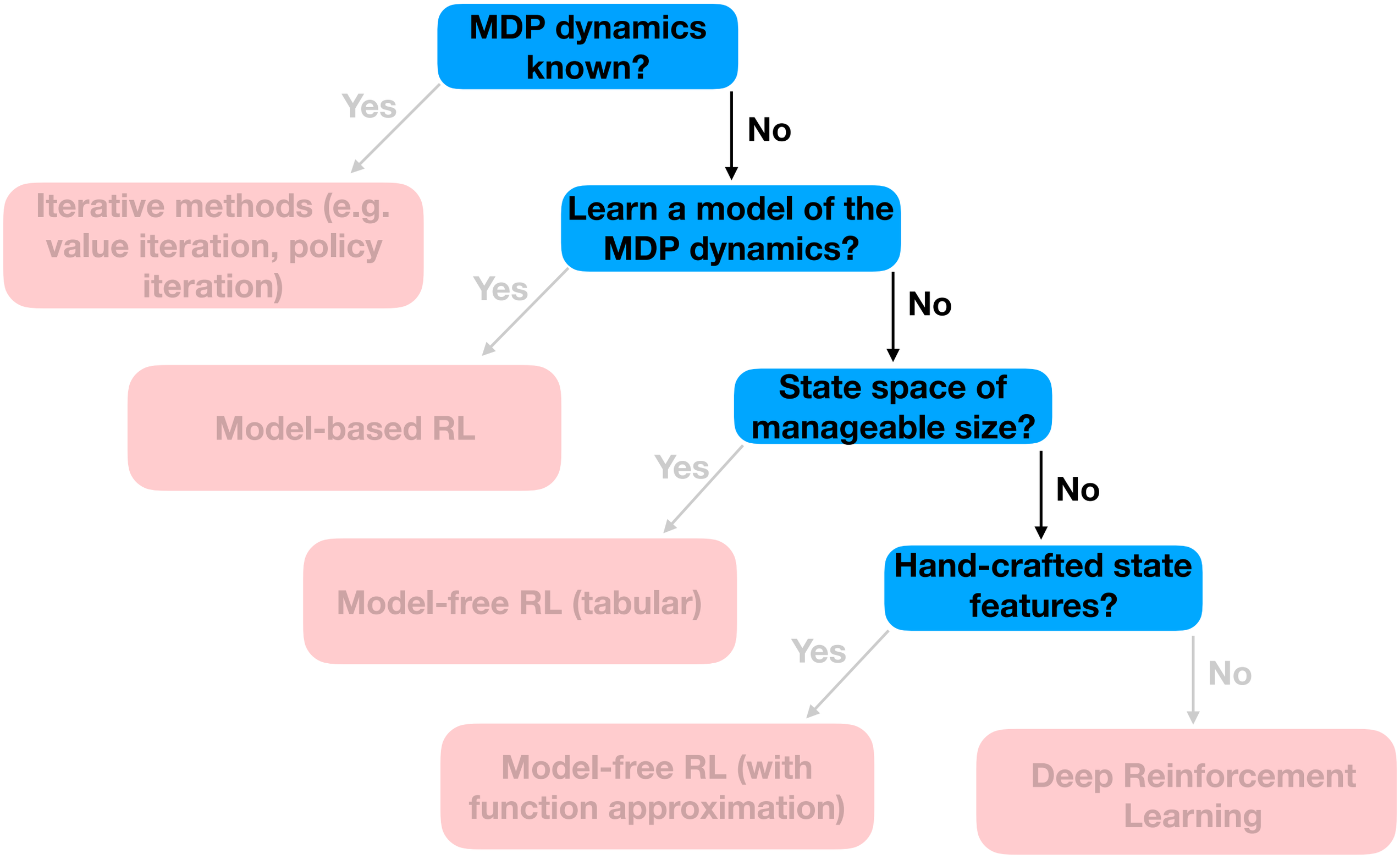
Sutton and Barto. Reinforcement Learning: An Introduction.

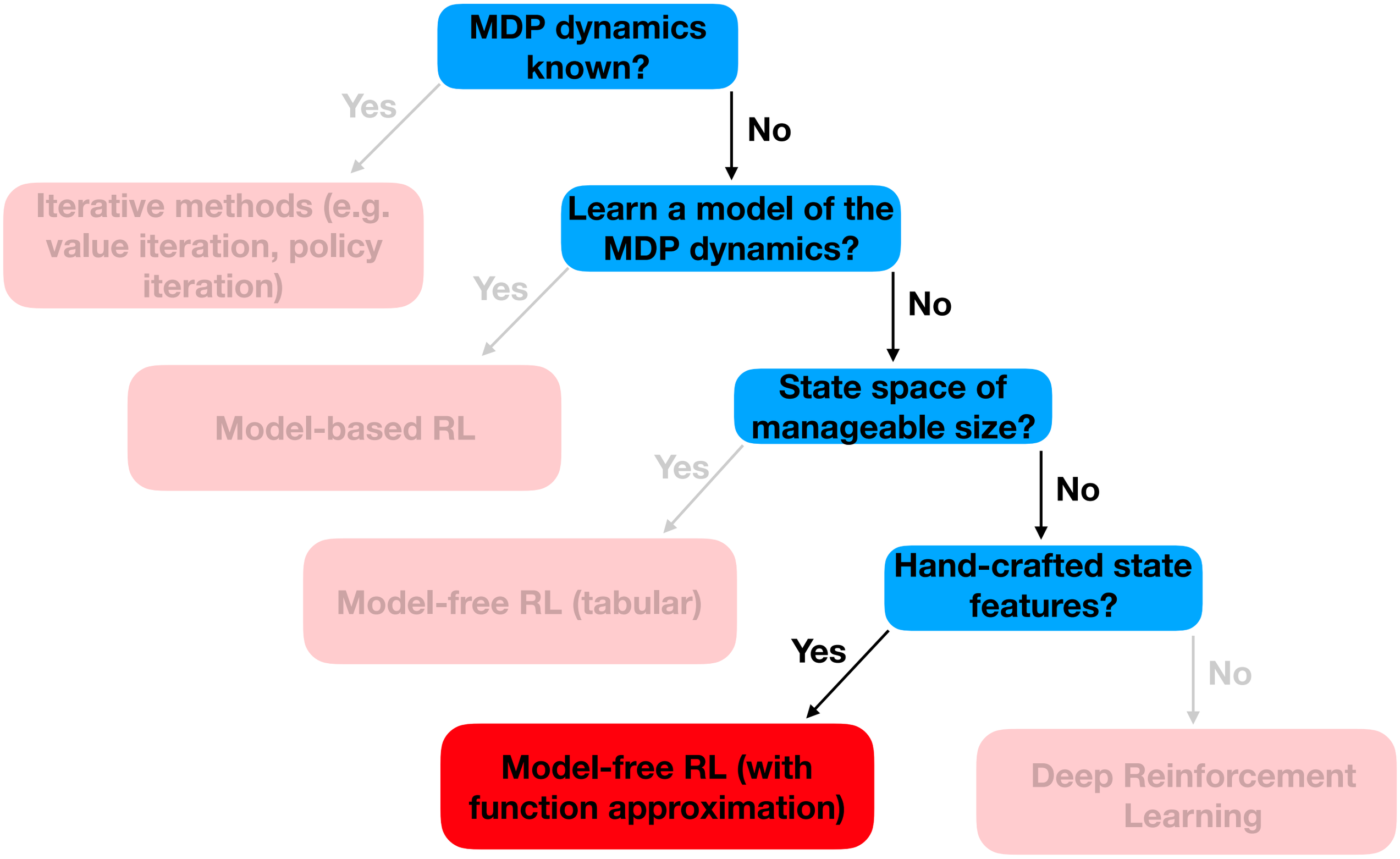
SARSA v. Q-Learning



- **Assume both SARSA and Q-learning use an epsilon-greedy policy**
- **SARSA outperforms Q-Learning in terms of online performance**
- **Maybe epsilon-greedy isn't the smartest way to explore…**

Sutton and Barto. Reinforcement Learning: An Introduction.


MDP dynamics known?
Yes
No
Iterative methods (e.g. value iteration, policy iteration)
Learn a model of the MDP dynamics?
Yes
No
Model-based RL
State space of manageable size?
Yes
No
Model-free RL (tabular)
Hand-crafted state features?
Yes
No
Model-free RL (with function approximation)
Deep Reinforcement Learning


MDP dynamics known?
Yes
No
Iterative methods (e.g. value iteration, policy iteration)
Learn a model of the MDP dynamics?
Yes
No
Model-based RL
State space of manageable size?
Yes
No
Model-free RL (tabular)
Hand-crafted state features?
Yes
No
Model-free RL (with function approximation)
Deep Reinforcement Learning

Function Approximation

- Need to share value estimate information across states based on some notion of state similarity

- Use the same strategy from supervised learning - encode states to feature vectors, and learn a model with weights

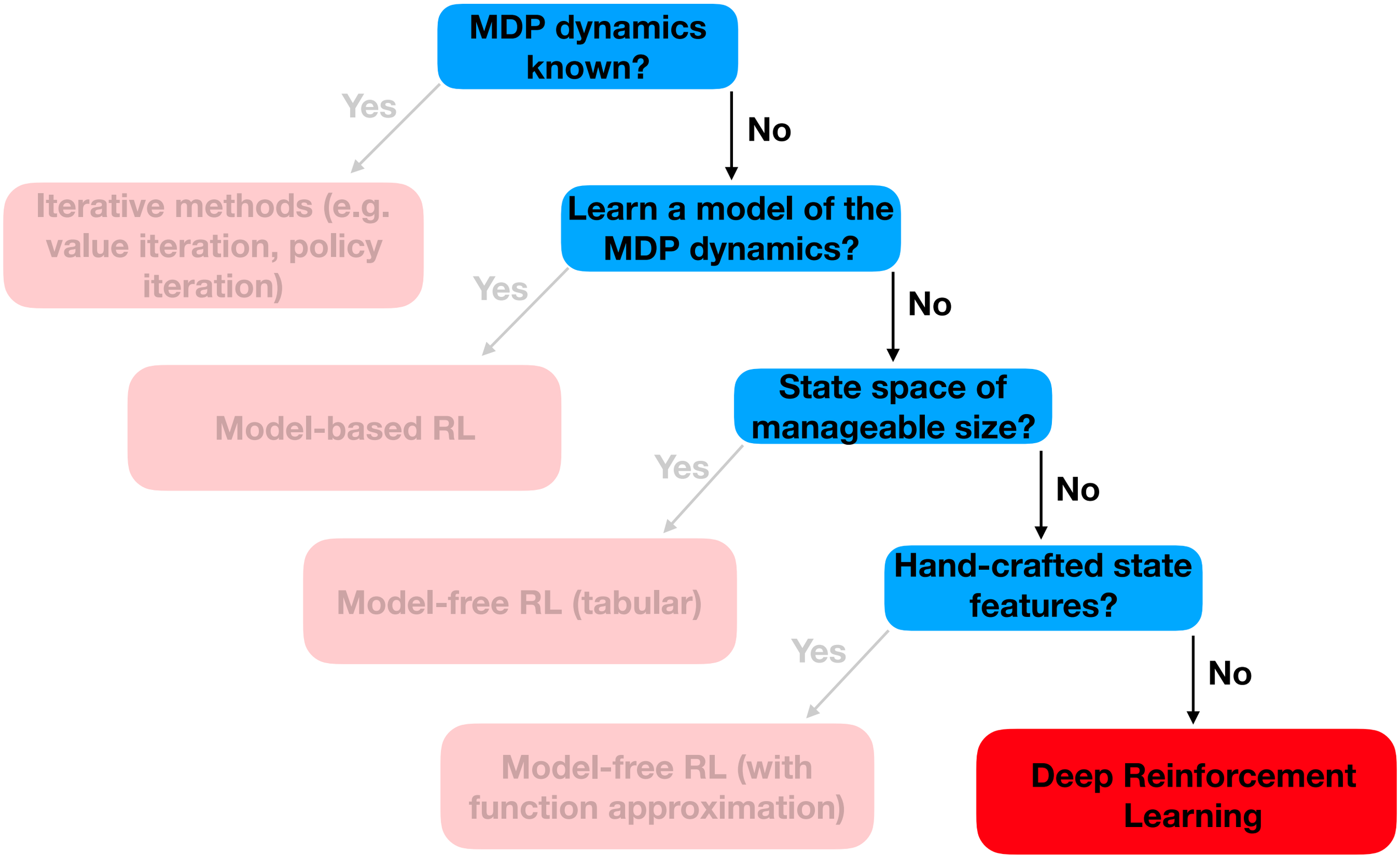
Function Approximation

- Need to share value estimate information across states based on some notion of state similarity

- Use the same strategy from supervised learning - encode states to feature vectors, and learn a model with weights

- Updating value estimates towards target values can now be done using gradient descent:

$$\min_{\mathbf{w}} \sum_{(s,a,r,s',a')} (\hat{Q}(s,a;\mathbf{w}) - \text{target})^2$$


MDP dynamics known?
Yes
No
Iterative methods (e.g. value iteration, policy iteration)
Learn a model of the MDP dynamics?
Yes
No
Model-based RL
State space of manageable size?
Yes
No
Model-free RL (tabular)
Hand-crafted state features?
Yes
No
Model-free RL (with function approximation)
Deep Reinforcement Learning

RL application: Breakout!

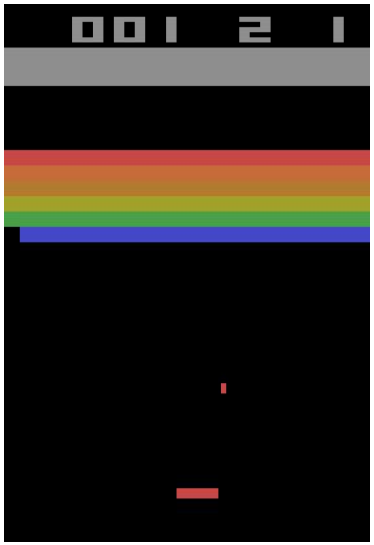
Breakout Game Description

Formally:

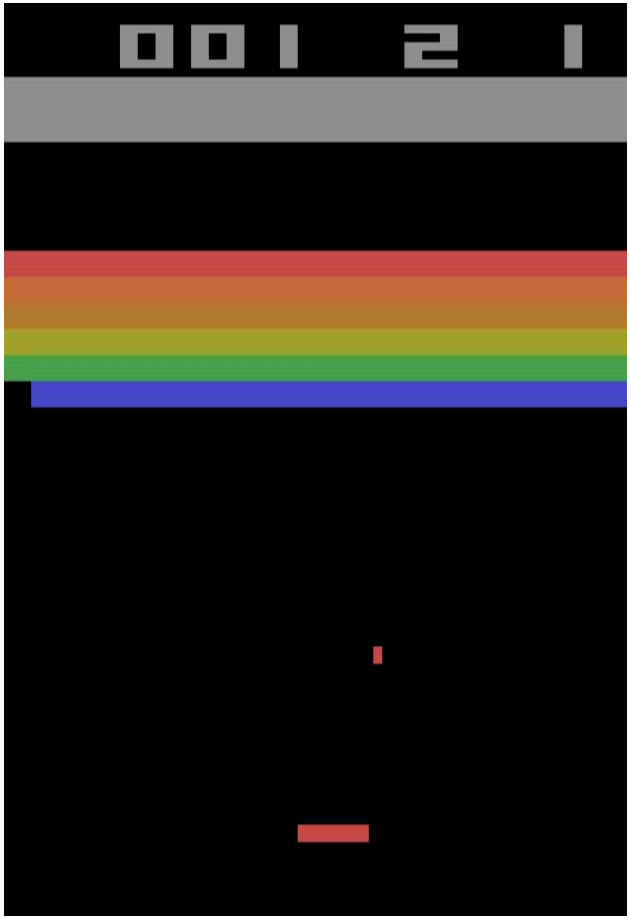
- *Actions*
  - move_paddle_left
  - move_paddle_right
  - do_not_move_paddle
- *Rewards*
  - If ball hits brick, reward = 1
  - Otherwise, reward = 0
- *End condition*
  - If ball falls off the screen, game ends

Can we learn to control an agent directly from sensory input?

In Breakout, sensory input would be a game screen frame.



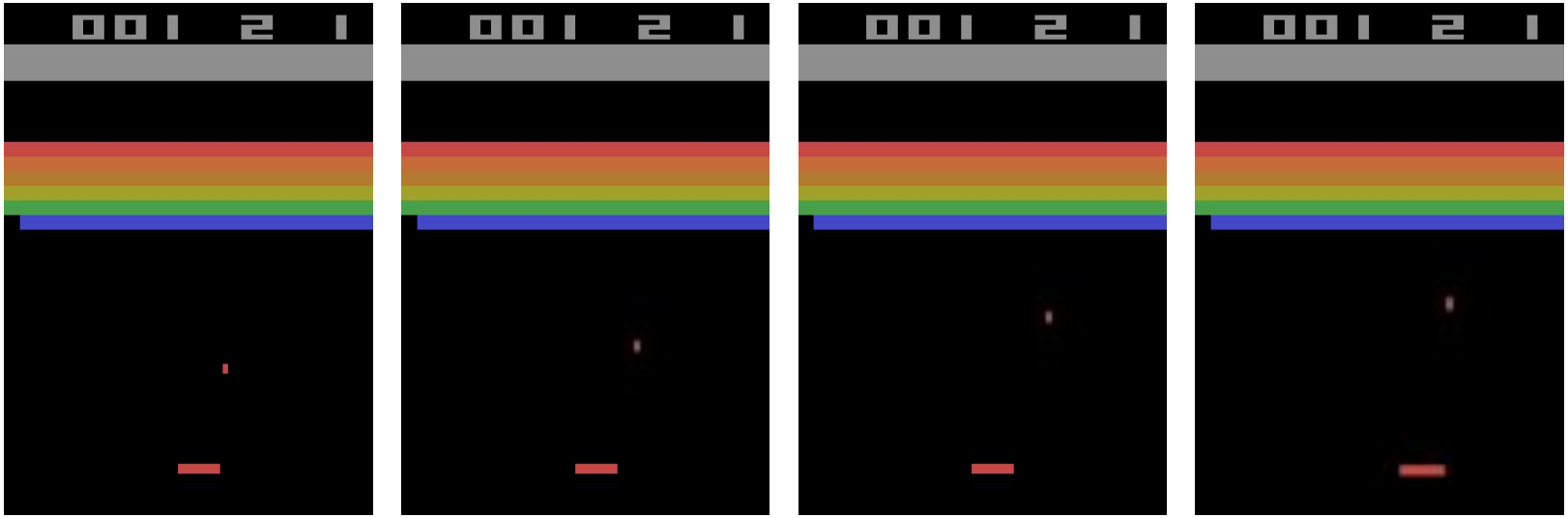
Finding a state representation



Consider this frame.

- Can you capture information like direction of the ball?
- Can you capture velocity?

Use a small number of consecutive frames for each state.



PROBLEM!!!

$$\#\ \text{states} \approx 256^{84x84X4}$$

(assume 84x84 pixels per screenshot, where each pixel can take on 256 values, and 4 screenshots per state)

Use function approximation!

featurize our state space!

1st try: hand-designing features Φ(s, a)

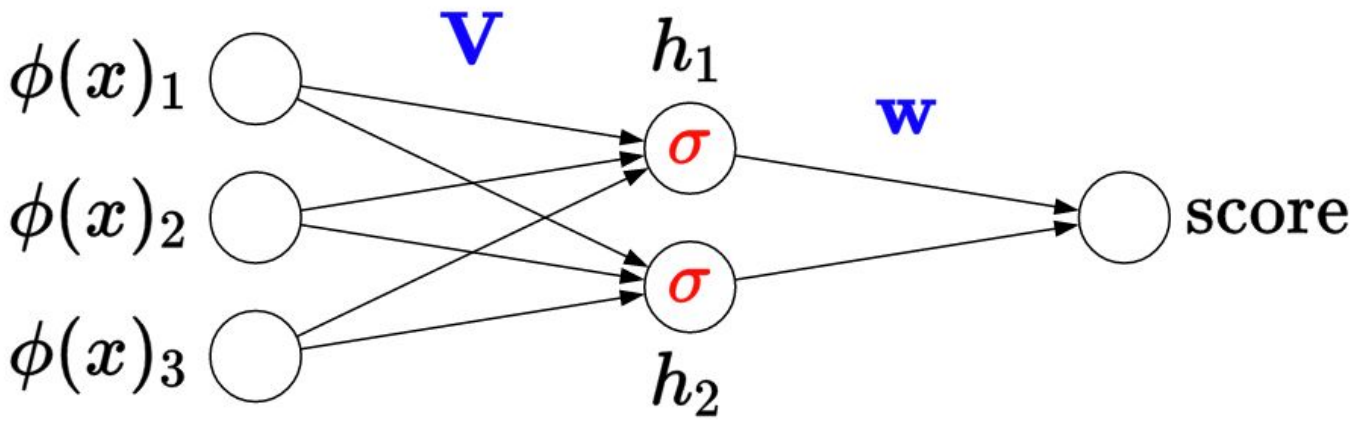
- Performance depends on the quality of features **Φ(s, a)**
- **Not generalized**
- Doesn't scale well with game complexity

→ ***Handcrafting features is very difficult!***

What if we automatically learned features from the pixels?

Deep Neural Nets (Review)

Neural network:



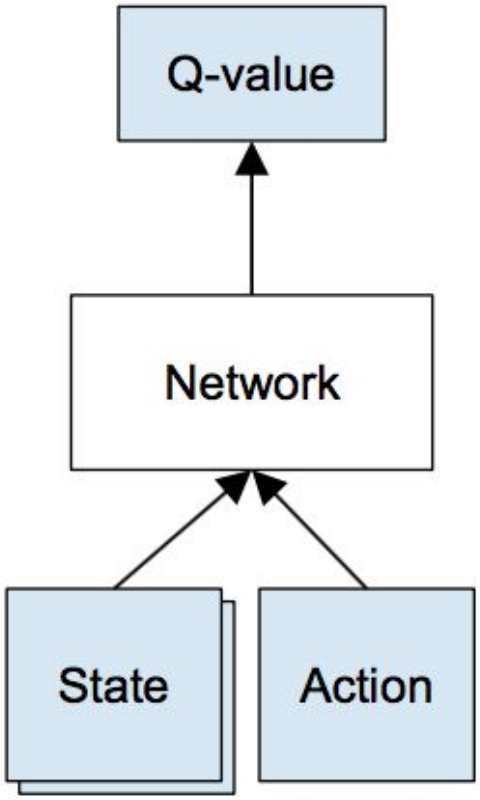
Intermediate hidden units:

$$h_j = \sigma(\mathbf{v}_j \cdot \phi(x)) \quad \sigma(z) = (1 + e^{-z})^{-1}$$

Output:

$$\text{score} = \mathbf{w} \cdot \mathbf{h}$$

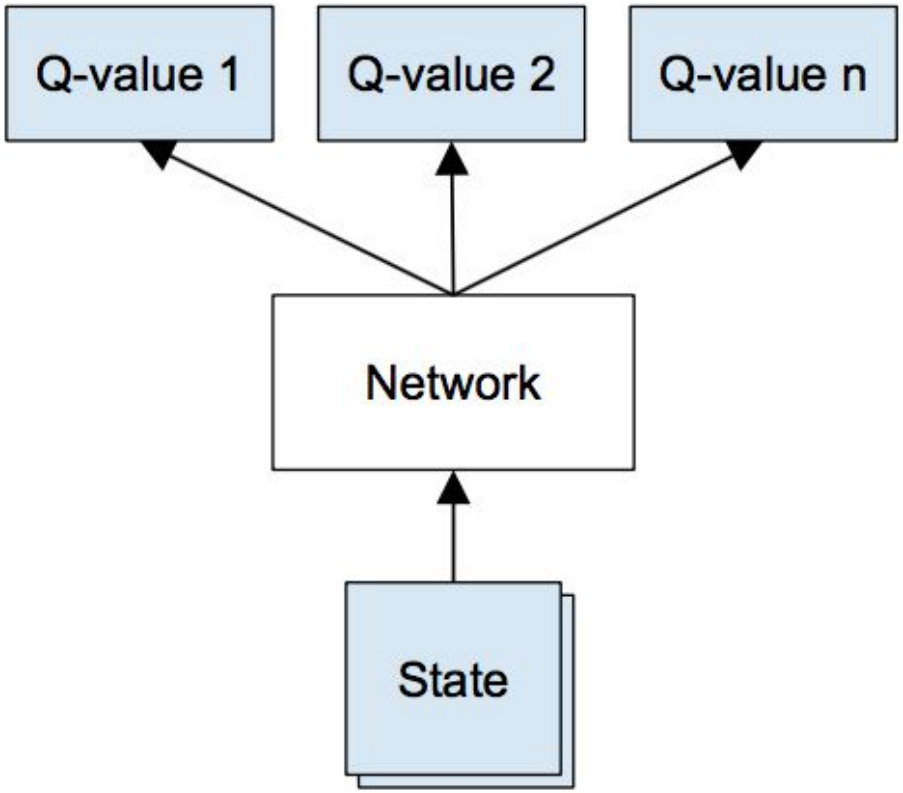
Neural Networks as Q(s, a) approximators



- Input (s,a) pair to neural network.
- Neural network predicts the Q-value for (s,a) pair

Can we make this even more efficient?

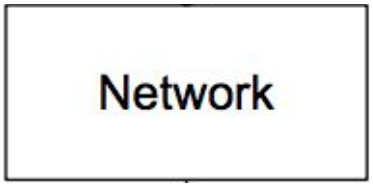
Neural Networks as Q(s, a) approximators



- State is the only input into the neural network.
- Network outputs a Q-value for every possible action.
- Action corresponding to the highest Q-value is chosen.

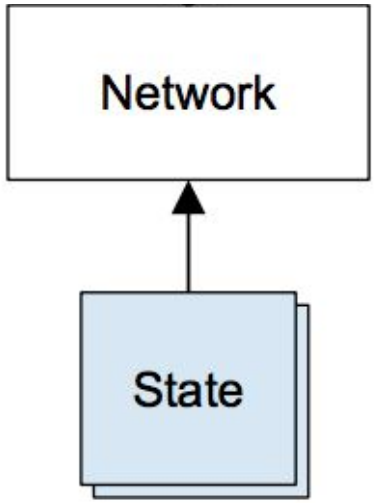
A single network to predict Q(s,a) for all possible states and actions!

Training Deep-Q-Networks (DQN)



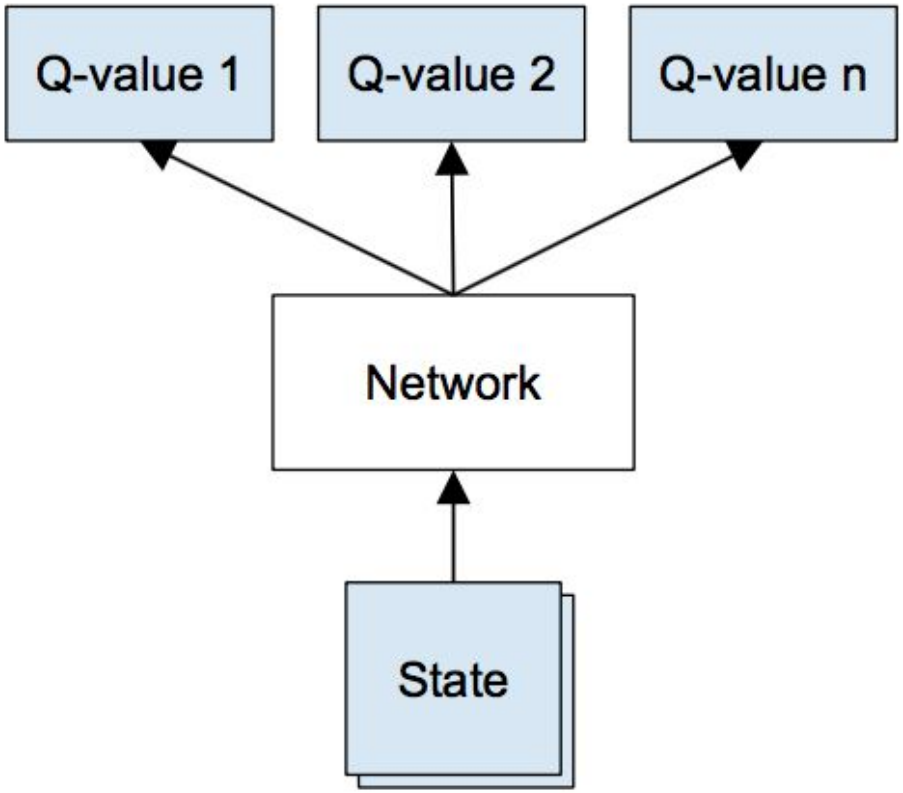
Initialize weights randomly!

Training Deep-Q-Networks (DQN)



States are passed as input.

Training Deep-Q-Networks (DQN)

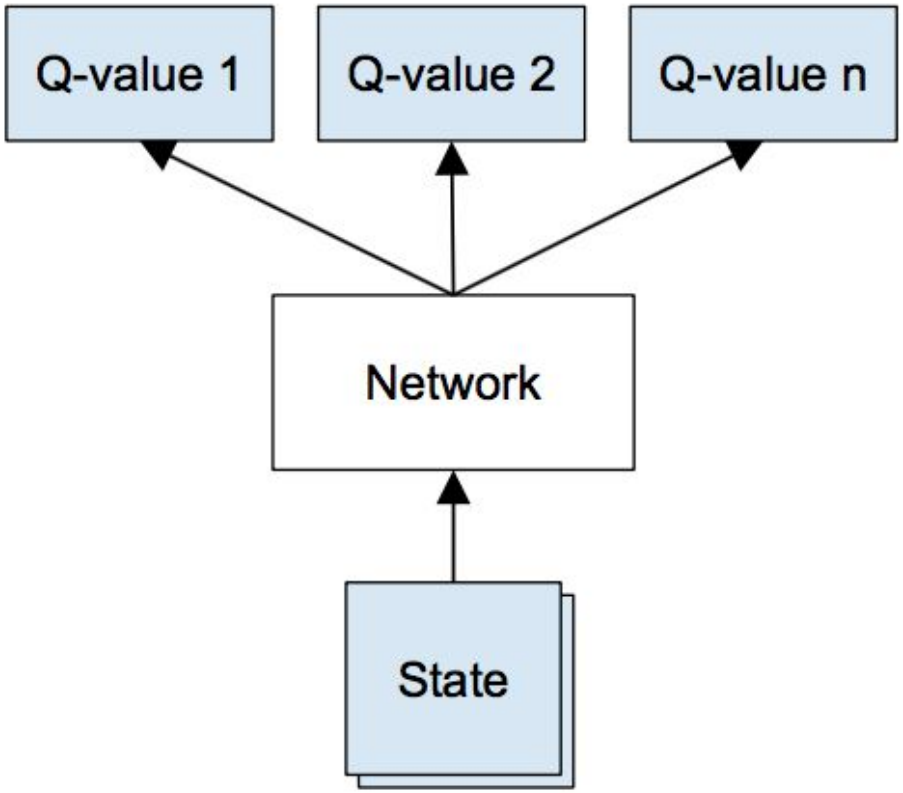


- Network outputs Q-values for each possible action.
- Action space for Breakout:

  [left, right, no-op]

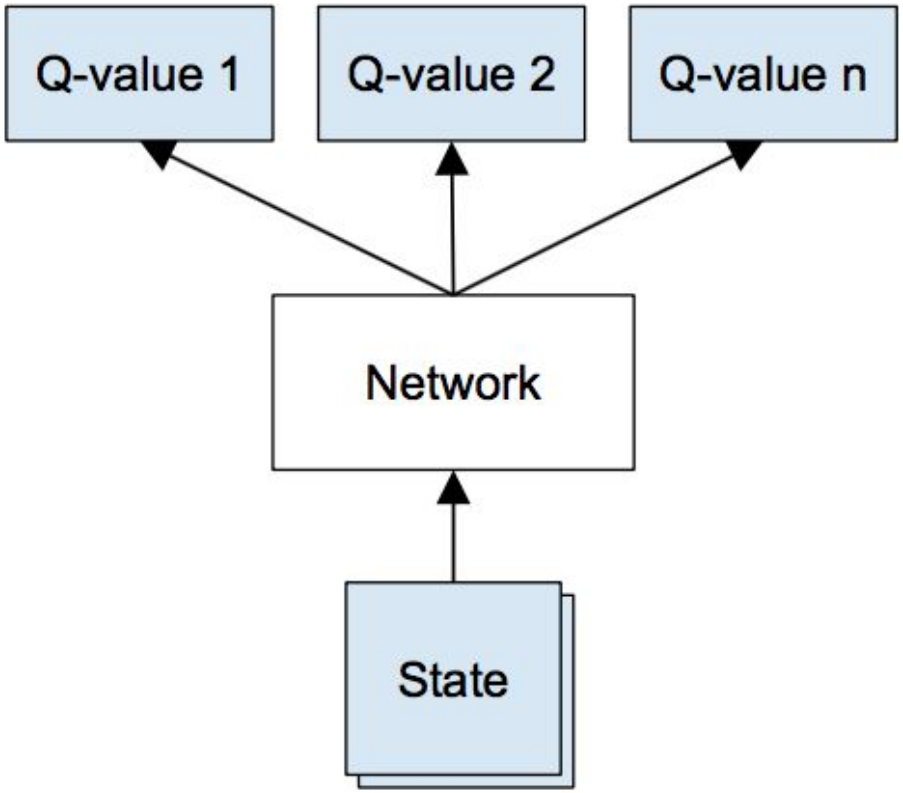
- Since initial weights are random, Q-values are random.

Training Deep-Q-Networks (DQN)



- Execute action that maximizes Q-value.
- Environment may or may not react to that action with a reward.
- Obtain next state.

Training Deep-Q-Networks (DQN)



- Run gradient descent on Q-learning loss.

Training Deep-Q-Networks

- Initialize weights randomly.
- Loop:
  - Obtain current state (s)
  - Run Neural Network on s to obtain Q-value for every action.
  - Execute action (a) that maximizes Q-value.
  - Obtain reward (r) and new state (s').
  - Perform gradient descent on Q-learning loss using (s, a, r, s')

$$\min_{\mathbf{w}} \sum_{(s,a,r,s')} (\underbrace{\hat{Q}_{\text{opt}}(s,a;\mathbf{w})}_{\text{prediction}} - \underbrace{(r+\gamma \hat{V}_{\text{opt}}(s'))}_{\text{target}})^2$$

Training Deep-Q-Networks - Additional Considerations

- Initialize weights randomly
- Initialize memory (D) with capacity N
- Loop:
  - Obtain current state (s)
  - Run Neural Network on s to obtain Q-value for every action
  - With probability ε, execute random action (a)
  - Otherwise, execute action (a) that maximizes Q-value
  - Obtain reward (r) and new state (s')
  - Store (s, a, r, s') in D
  - Randomly sample $(s, a, r, s')_D$ from D
  - Perform gradient descent on Q-learning loss using $(s, a, r, s')_D$

Let’s watch Deep RL in action

https://www.youtube.com/watch?v=V1eYniJ0Rnk

Sources

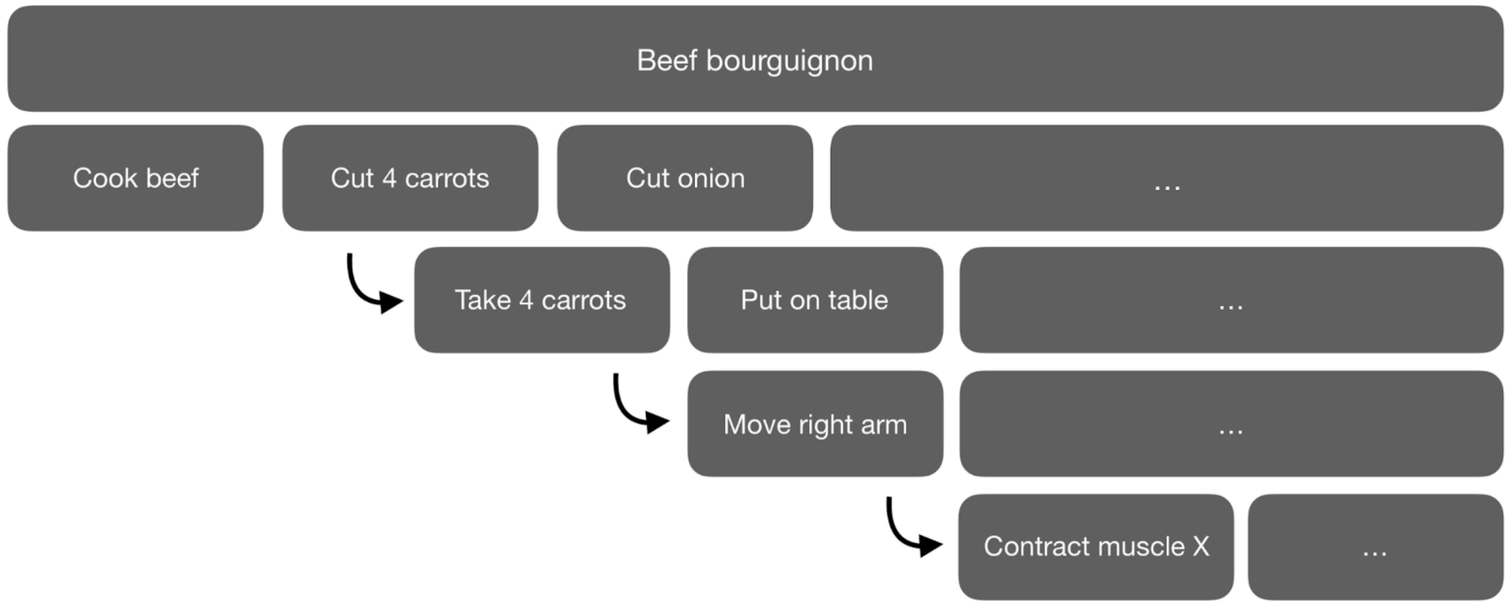
[1] Mnih, Volodymyr, et al. "Human-level control through deep reinforcement learning." Nature 518.7540 (2015): 529-533.

[2] Mnih, Volodymyr, et al. "Playing atari with deep reinforcement learning." arXiv preprint arXiv:1312.5602 (2013).

Different flavors of RL

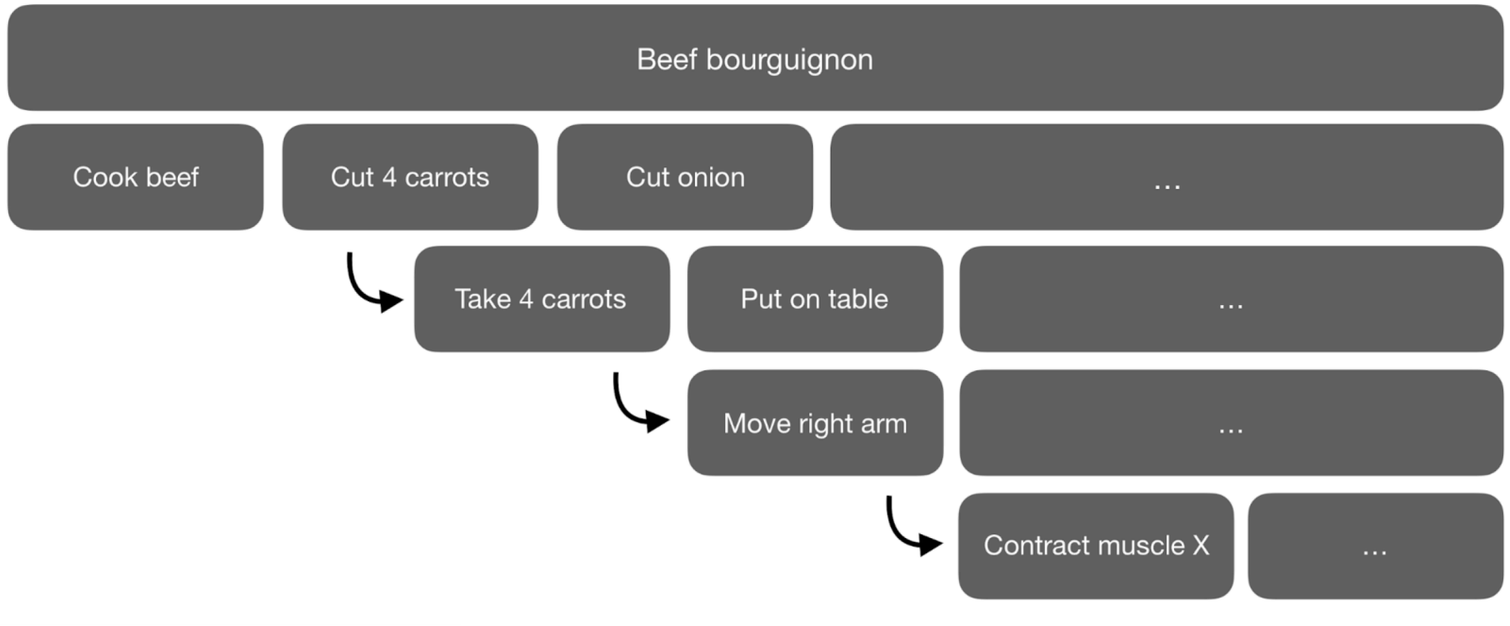
Different flavors of RL

- **Hierarchical RL**: Compose low-level policies to form higher-level policies

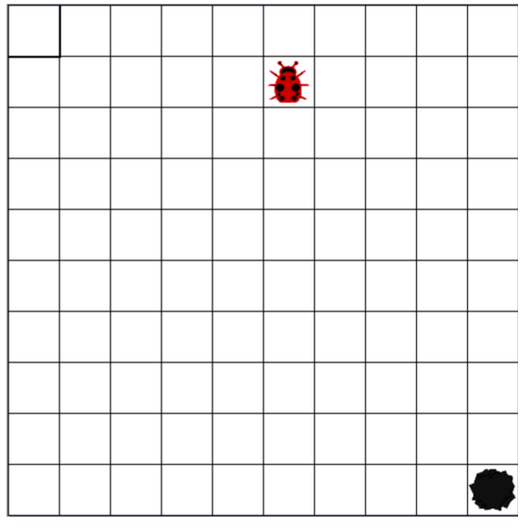


Different flavors of RL

- **Hierarchical RL**: Compose low-level policies to form higher-level policies

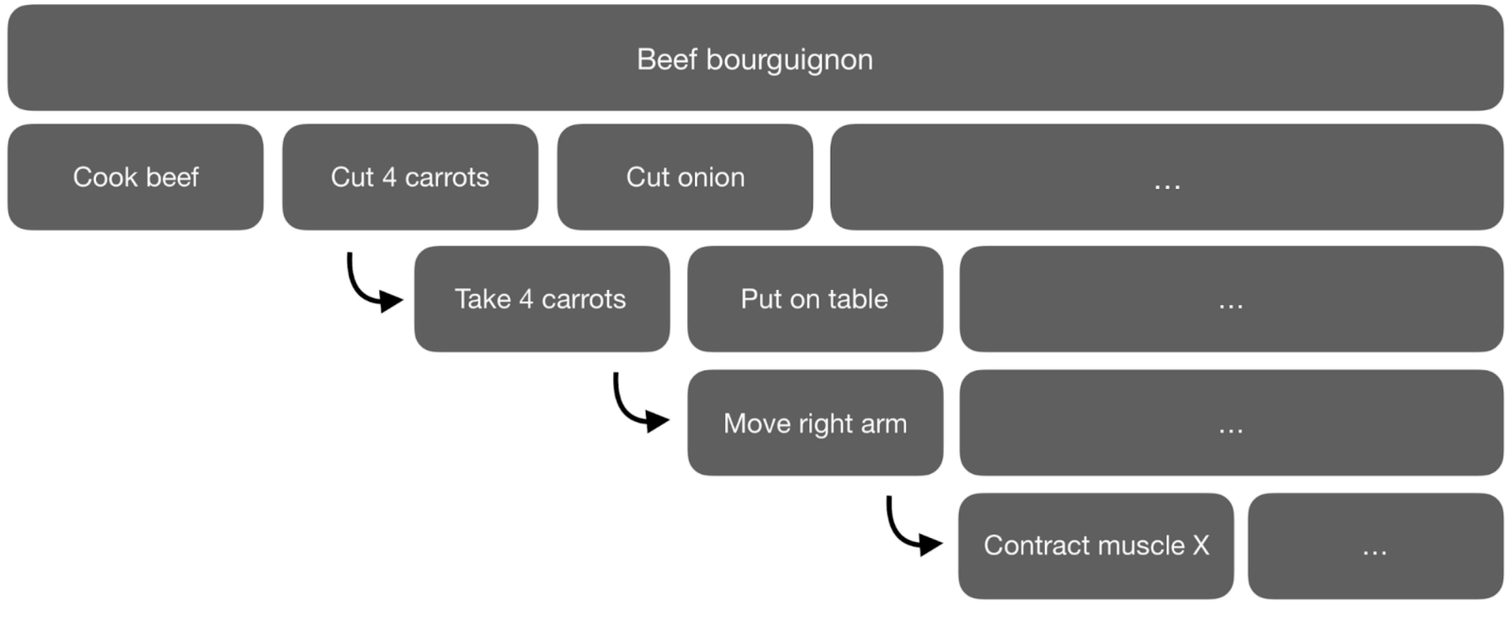


- **Meta RL**: Train an agent which can quickly adapt to small changes in the MDP

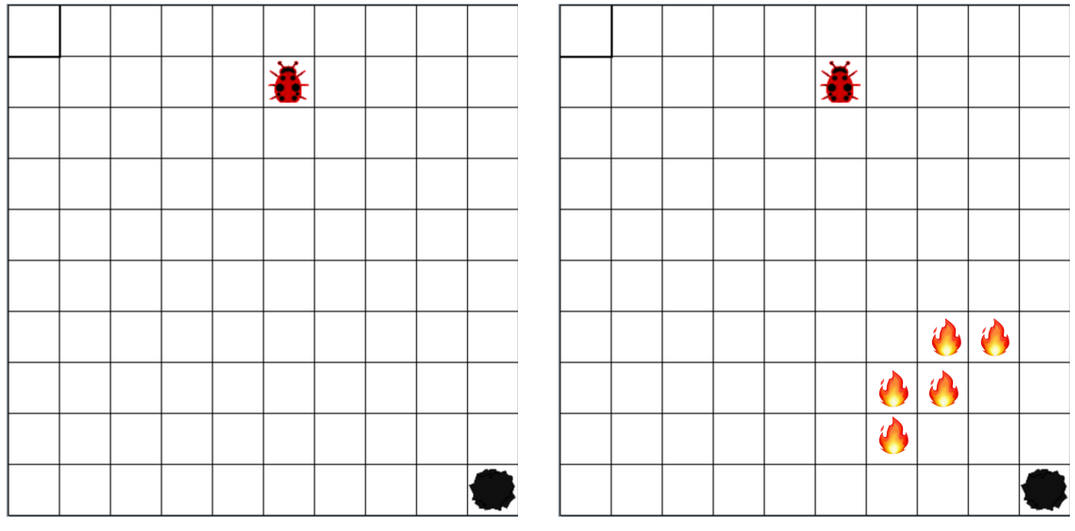


Different flavors of RL

- **Hierarchical RL**: Compose low-level policies to form higher-level policies

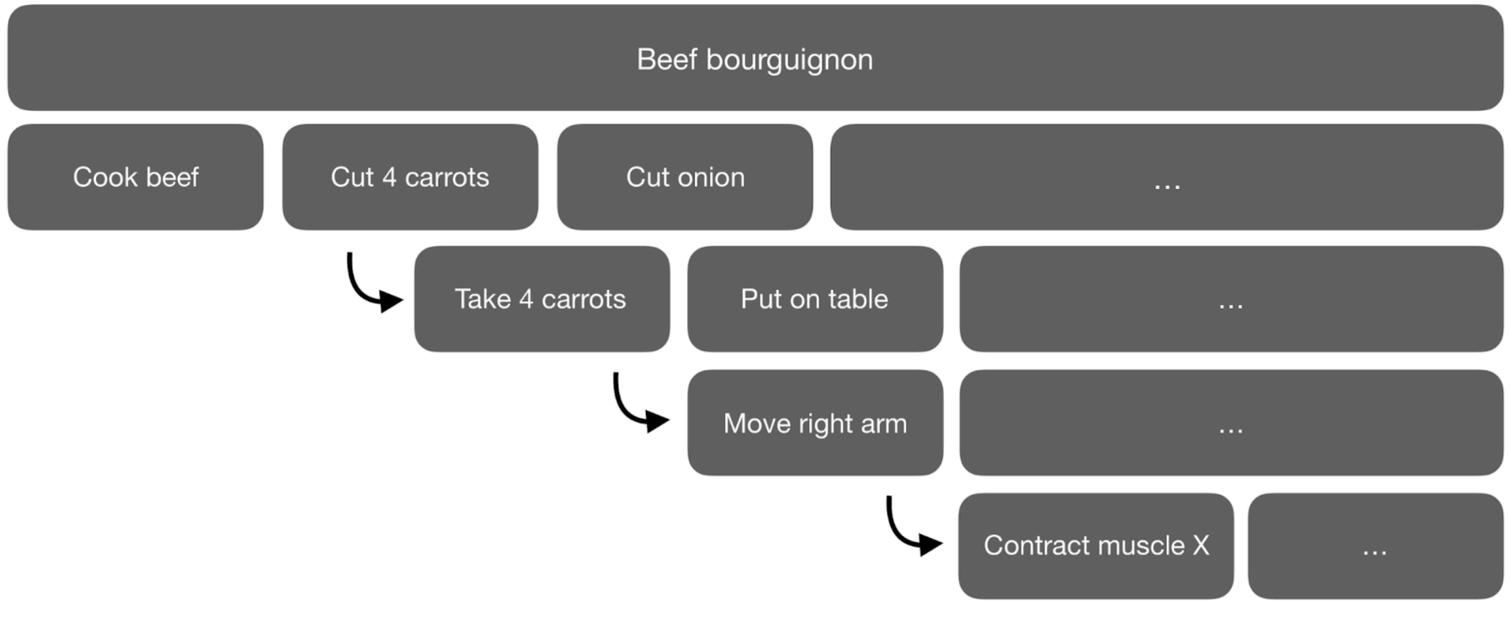


- **Meta RL**: Train an agent which can quickly adapt to small changes in the MDP

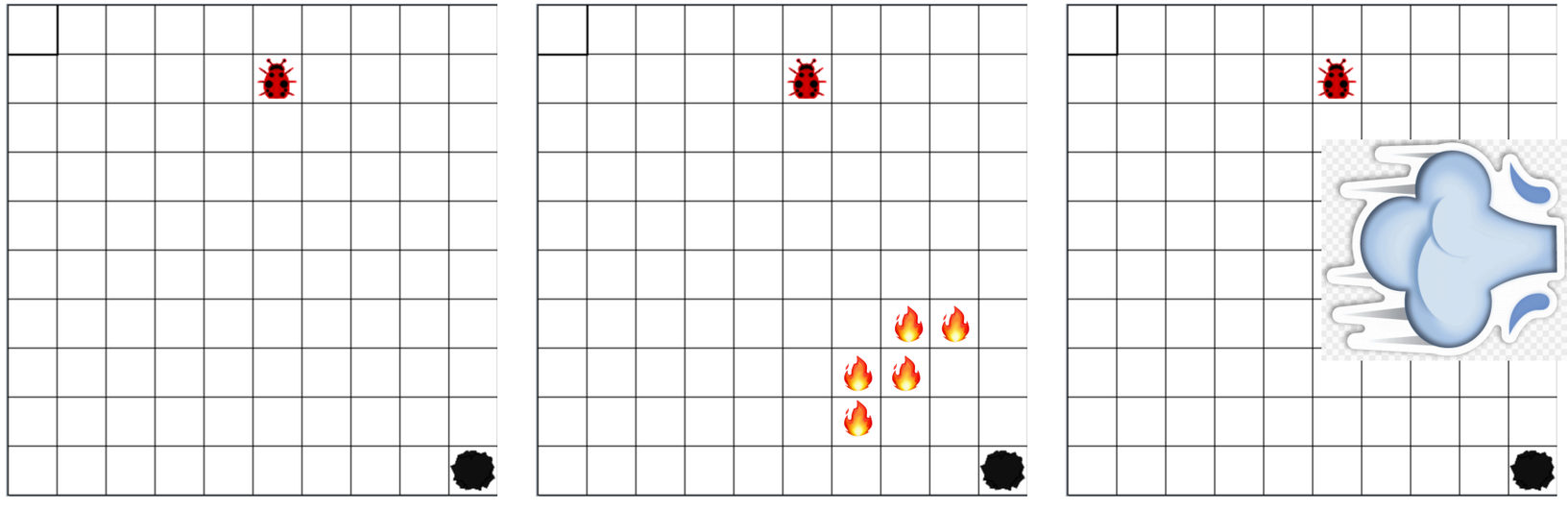


Different flavors of RL

- **Hierarchical RL**: Compose low-level policies to form higher-level policies

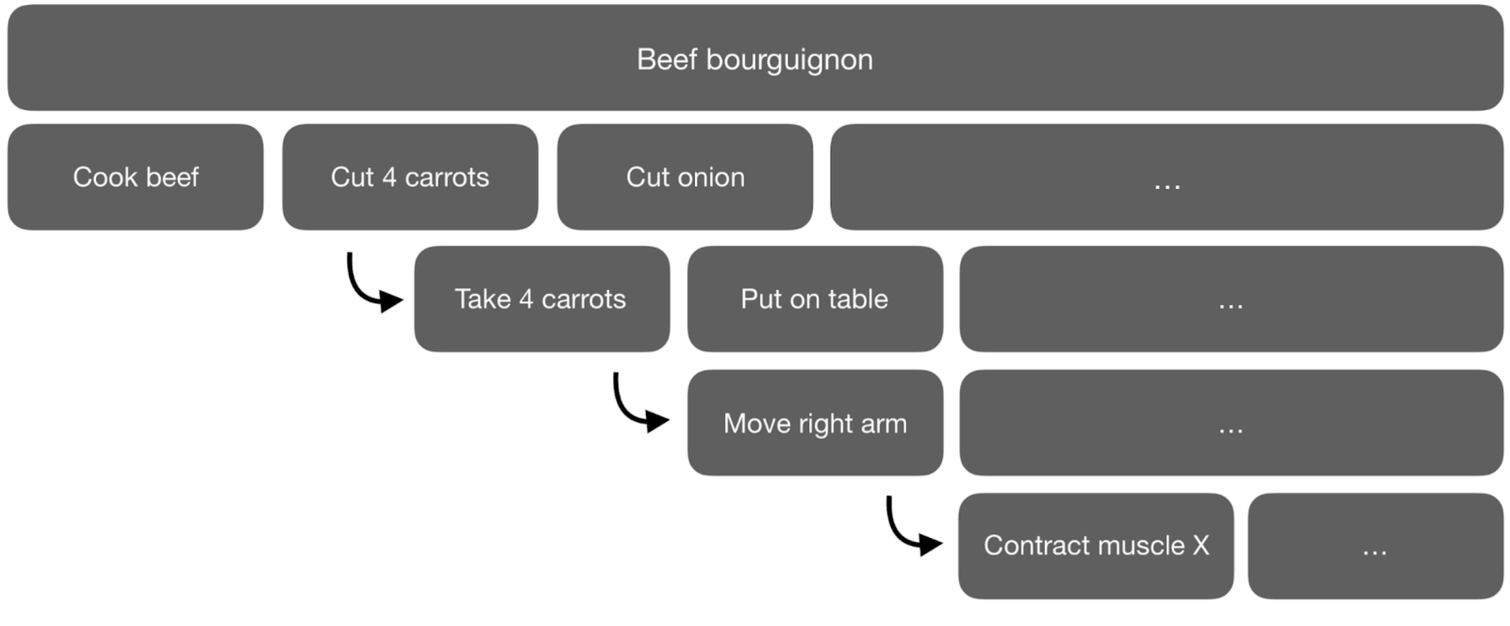


- **Meta RL**: Train an agent which can quickly adapt to small changes in the MDP

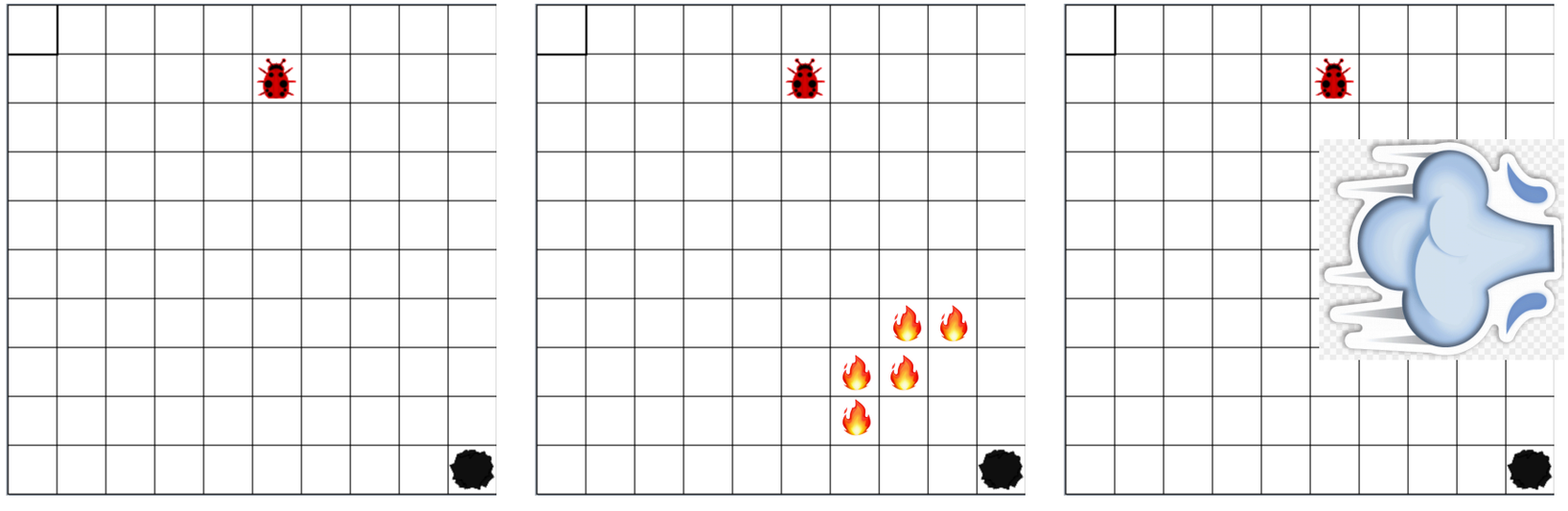


Different flavors of RL

- **Hierarchical RL**: Compose low-level policies to form higher-level policies



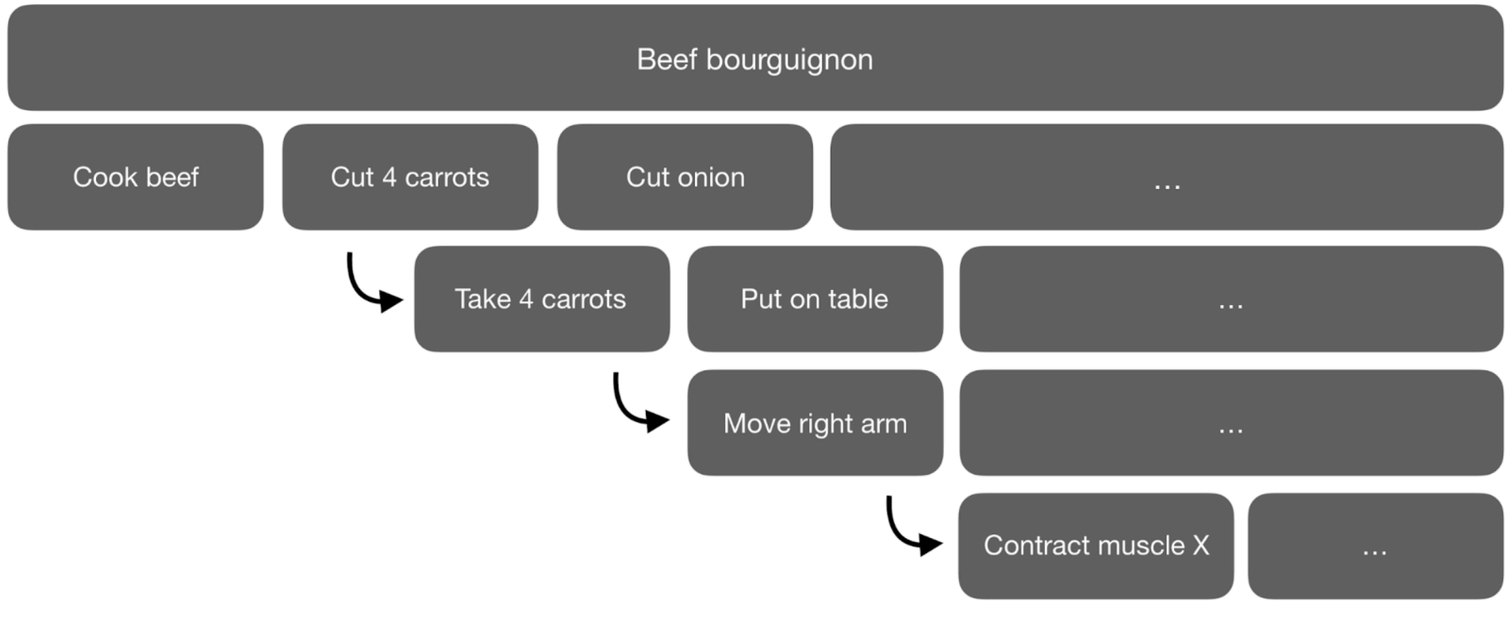
- **Meta RL**: Train an agent which can quickly adapt to small changes in the MDP



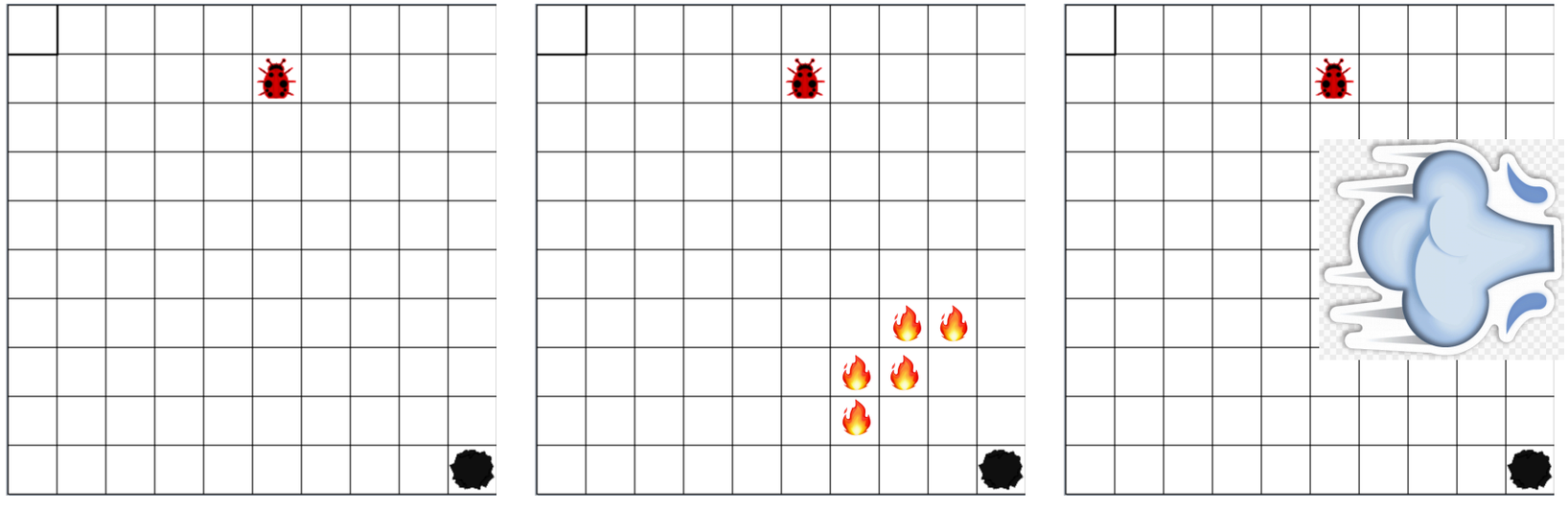
- **Inverse RL**: Infer agent preferences given experience

Different flavors of RL

- **Hierarchical RL**: Compose low-level policies to form higher-level policies



- **Meta RL**: Train an agent which can quickly adapt to small changes in the MDP



- **Inverse RL**: Infer agent preferences given experience

Reward —RL—> Behavior

Behavior —IRL—> Reward