Solving Continuous MDPs: The Linear Quadratic Regulator (LQR)

Sanjiban Choudhury

Model-based Planning

Step 0: Build a robot

Step 1: Collect data of your robot doing stuff in the world

Step 2: Use data to learn a dynamics model for your robot

Step 3: Plan with the model to compute an optimal policy

Today's Challenge!

Step 0: Build a robot

Step 1: Collect data of your robot doing stuff in the world

Step 2: Use data to learn a dynamics model for your robot

Step 3: Plan with the model to compute an optimal policy

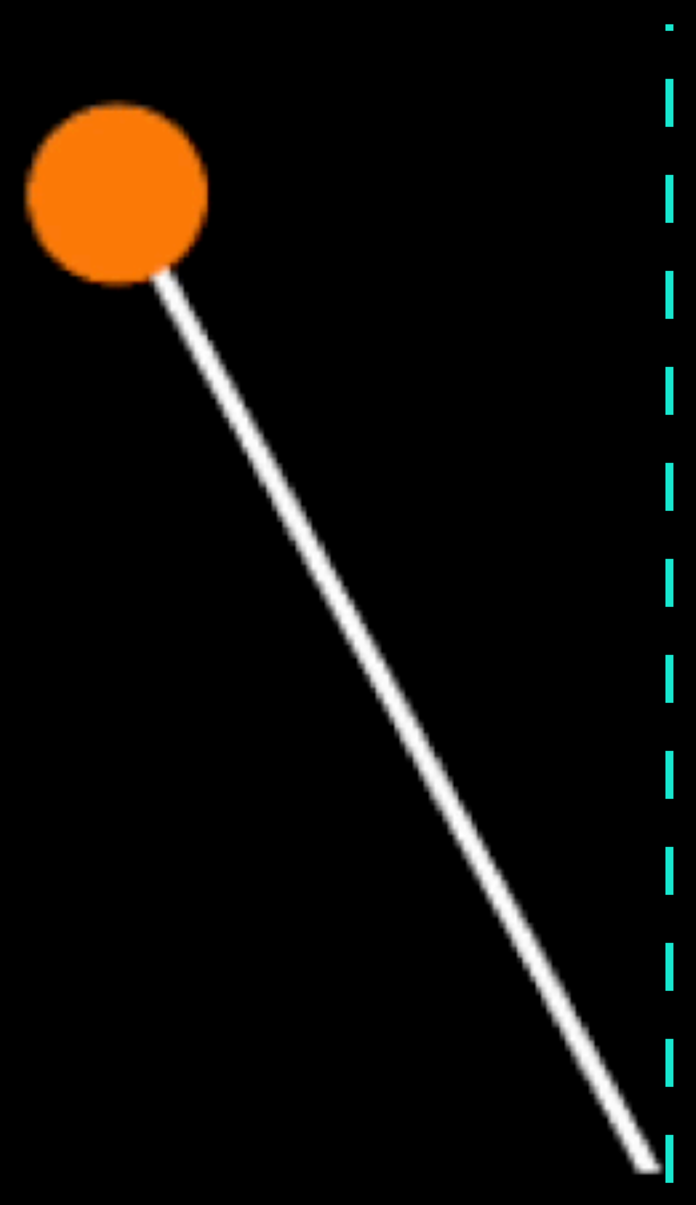
How do we do this for robots with continuous state-actions?


Boston Dynamics

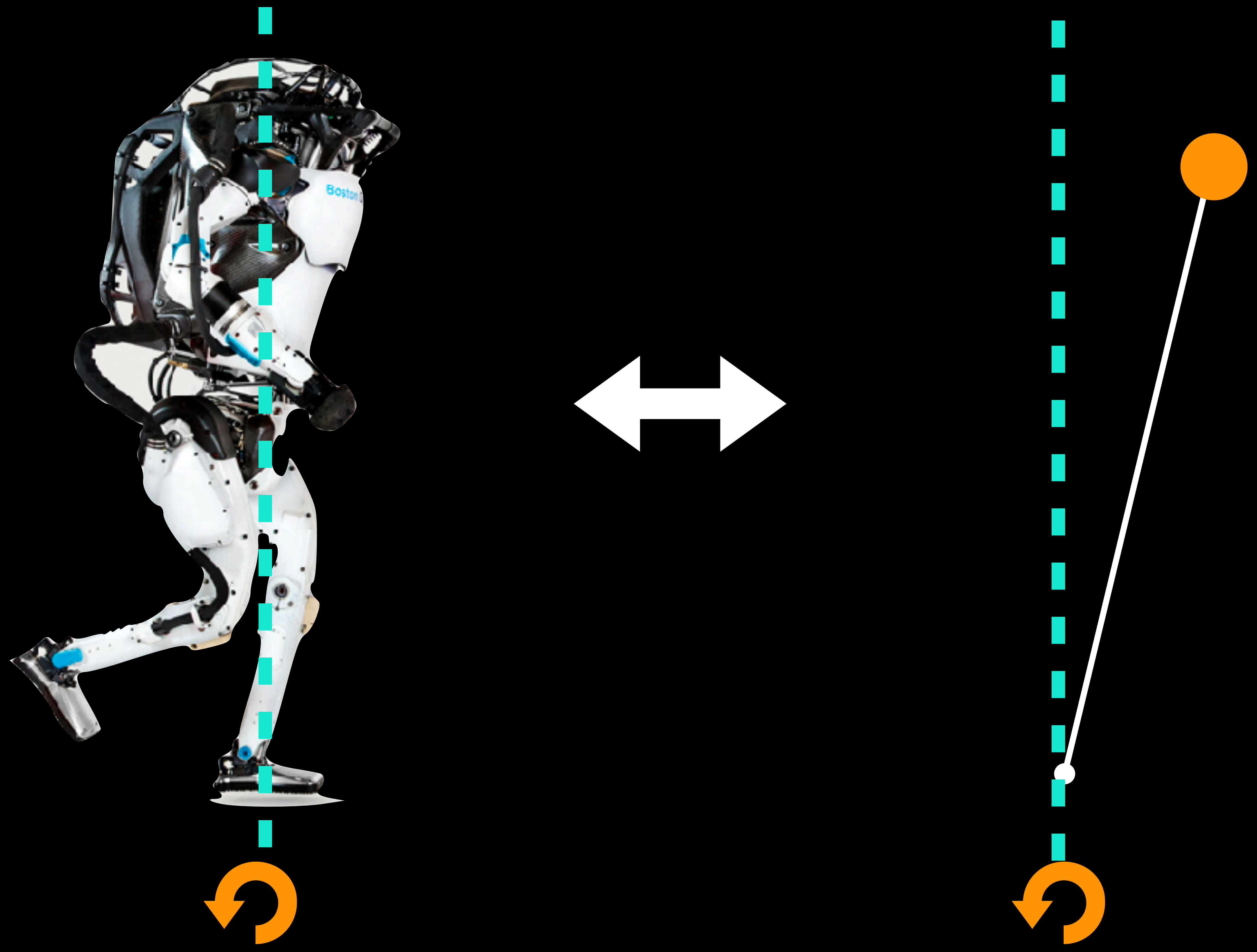
Brainstorm

How do we model the Atlas backflip as a Markov Decision Problem <S, A, C, T>?

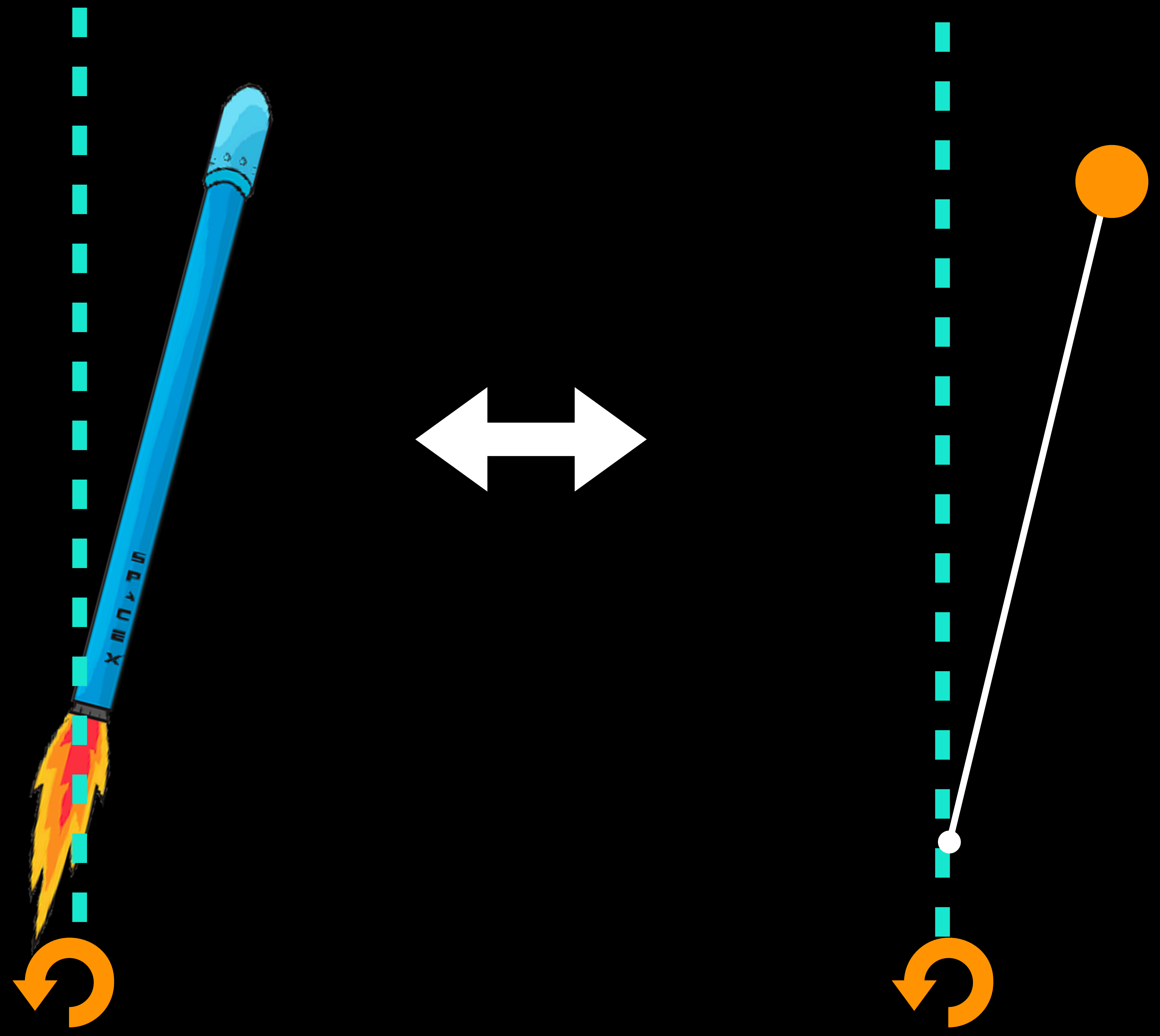
The Inverted Pendulum: A fundamental dynamics model



Humanoid balancing



Rocket landing



Recall: How do we solve a MDP?

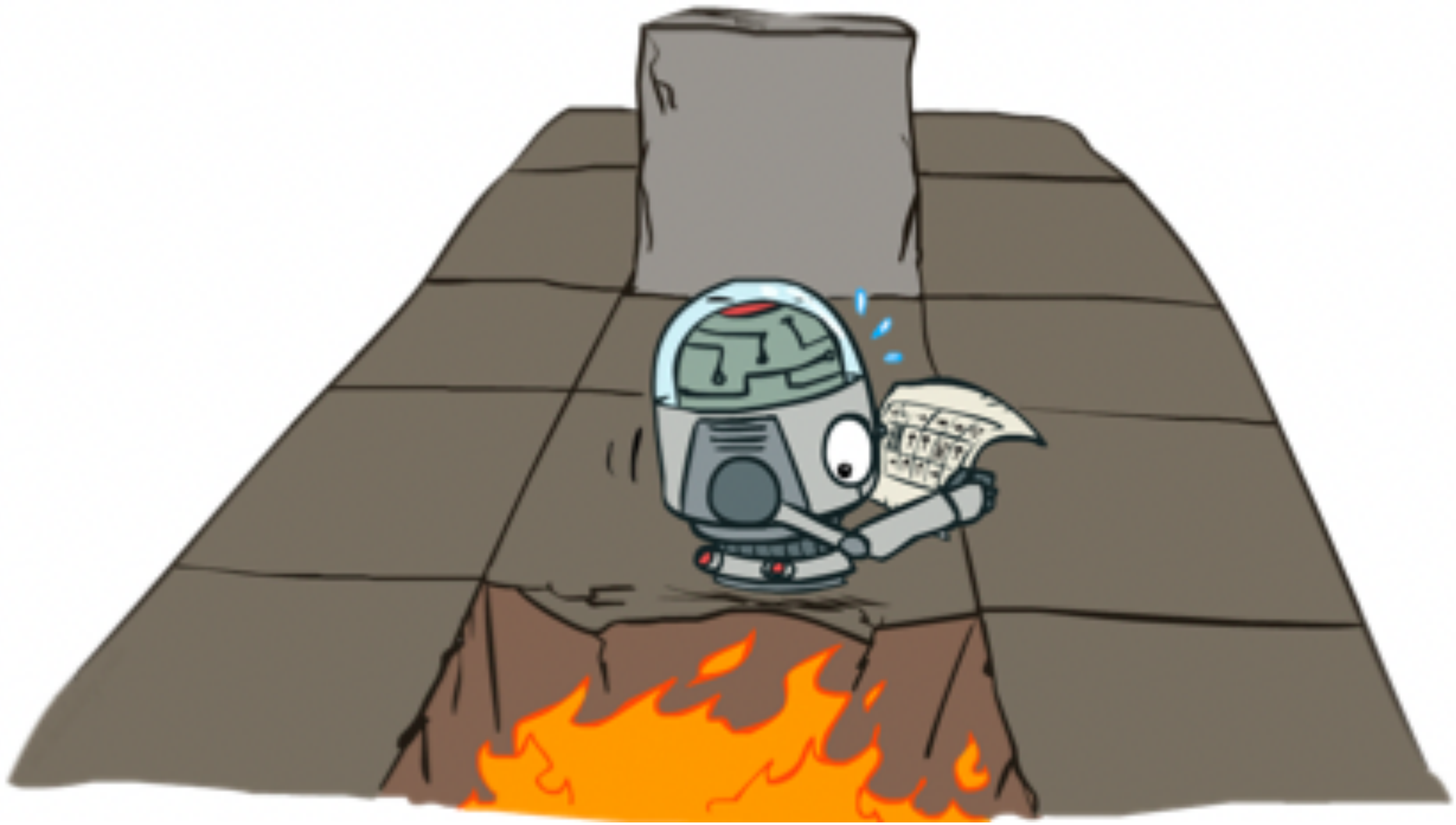


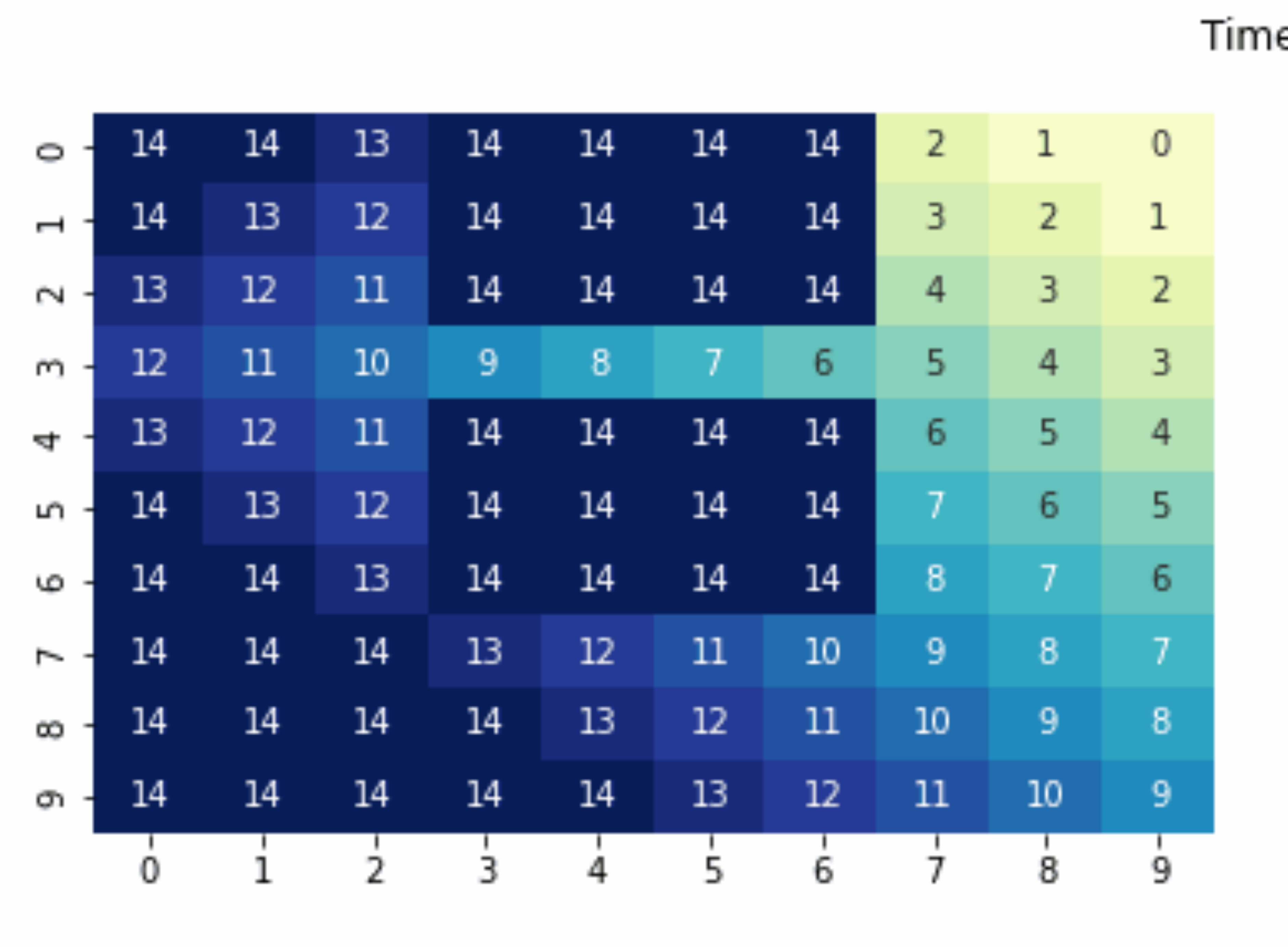
Image courtesy Dan Klein

Value Iteration

Initialize value function at last time-step

$$V^*(s, T-1) = \min_a c(s,a)$$

for $t = T-2, \ldots, 0$



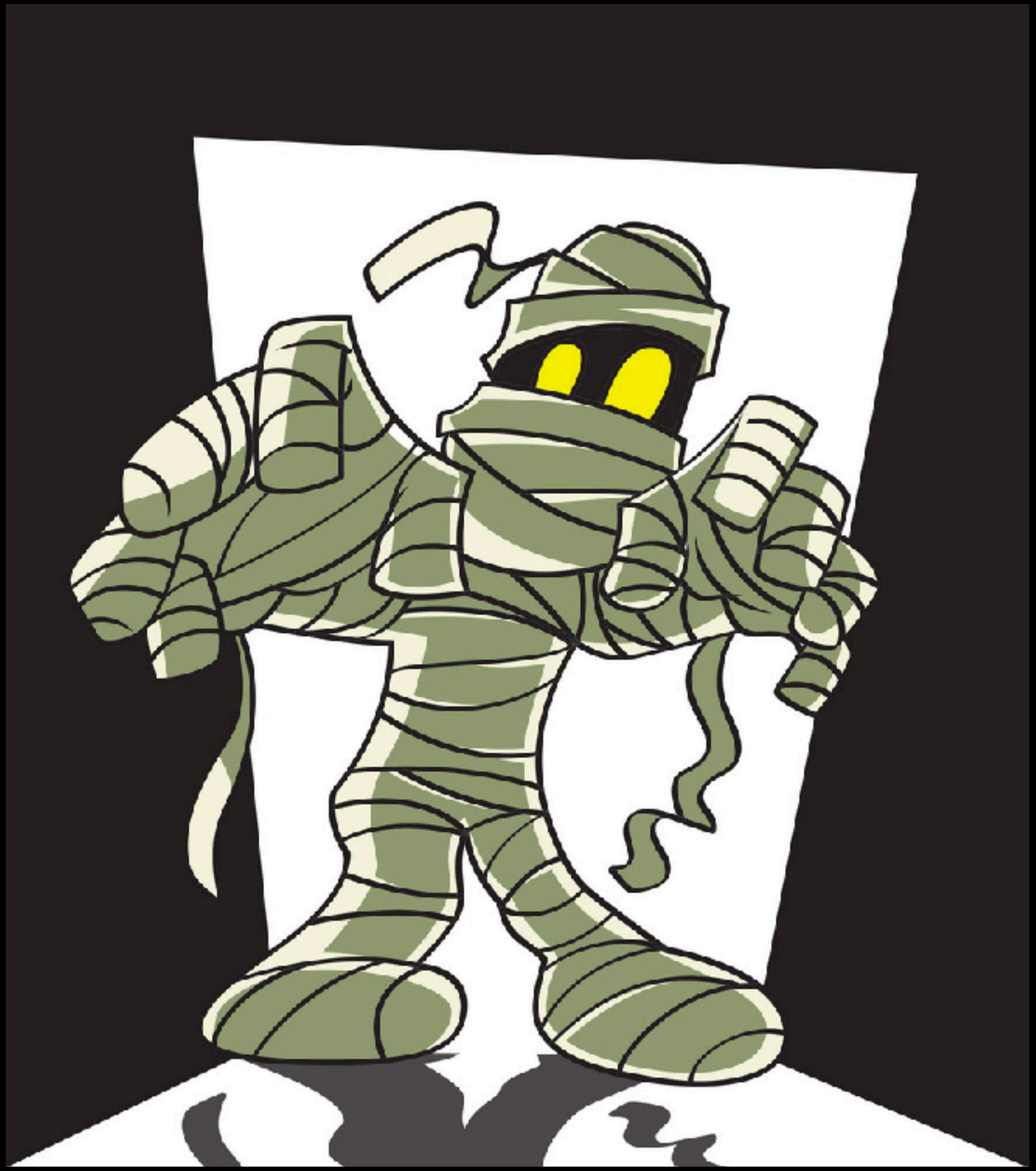
Compute value function at time-step t

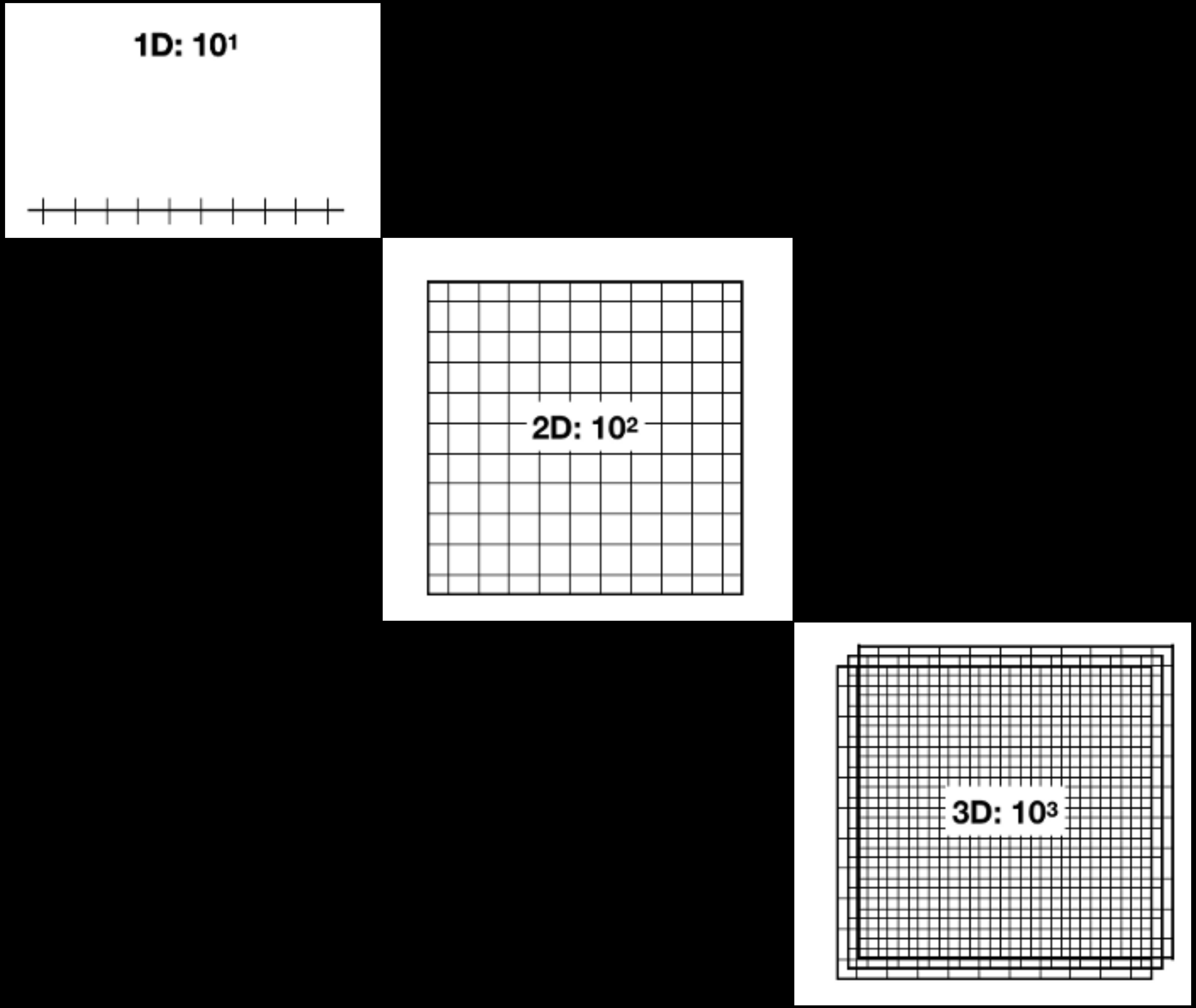
$$V^*(s,t) = \min_a \left[c(s,a) + \gamma \sum_{s'} \mathcal{T}(s' | s,a) V^*(s', t+1) \right]$$

Can we apply value iteration to solve this MDP?

$$V^*(s,t) = \min_a \left[c(s,a) + \gamma \sum_{s'} \mathcal{T}(s' \mid s,a) V^*(s', t+1) \right]$$

THE CURSE OF DIMENSIONALITY





Curse of Dimensionality

We cannot discretize continuous states and actions, because the number of states/action grows exponentially with dimension

We need some approximation or assumptions!

Can we analytically *represent* and *update* $V^*(s, t)$?

$$V^*(s,t) = \min_a \left[c(s,a) + \gamma \sum_{s'} \mathcal{T}(s' \mid s,a) V^*(s', t+1) \right]$$

What class of functions can we use for $\mathcal{T}(s' \mid s, a)$ and $V^*(s', t+1)$?

Can we analytically *represent* and *update* $V^*(s,t)$?

Yes*

$$V^*(s,t) = \min_a \left[c(s,a) + \gamma \sum_{s'} \mathcal{T}(s' \mid s,a) V^*(s', t+1) \right]$$

(Quadratic) (Quadratic) (Linear) (Quadratic)

Linear Quadratic Regulator (LQR)

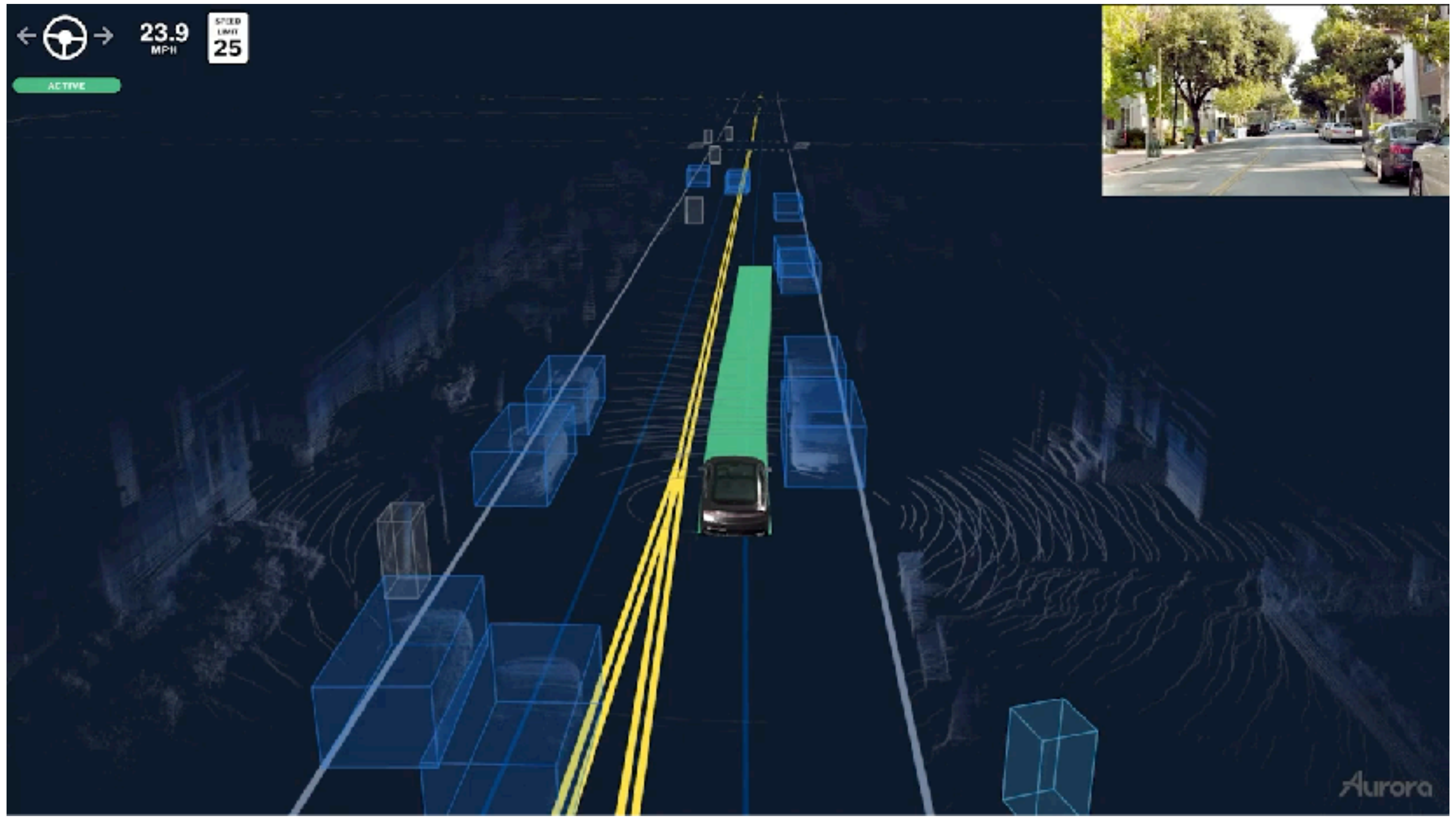
LQR is widely used in real world robotics

But the real world is not linear and quadratic, right?

No, but we can *linearize* dynamics and *quadricize* the costs about some reference

LQR can then be used as a very fast subroutine to compute optimal policy

LQR is widely used in real world robotics



Check out notebook

cs4756_robot_learning / notebooks / inverted_pendulum_lqr.ipynb

jren44 Initial commit a6c9feb · on Jan 18 History

Preview Code Blame Raw

Illustrated Linear Quadratic Regulator

Companion to courses lectures from CS6756: Learning for Robot Decision Making and Chapter 2 of Modern Adaptive Control and Reinforcement Learning.

In [3]:

```
import numpy as np
import autograd.numpy as np
from autograd import grad, jacobian
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation
from matplotlib import rc
from IPython.display import HTML, Image
from matplotlib.patches import Circle
rc('animation', html='jshtml')
```

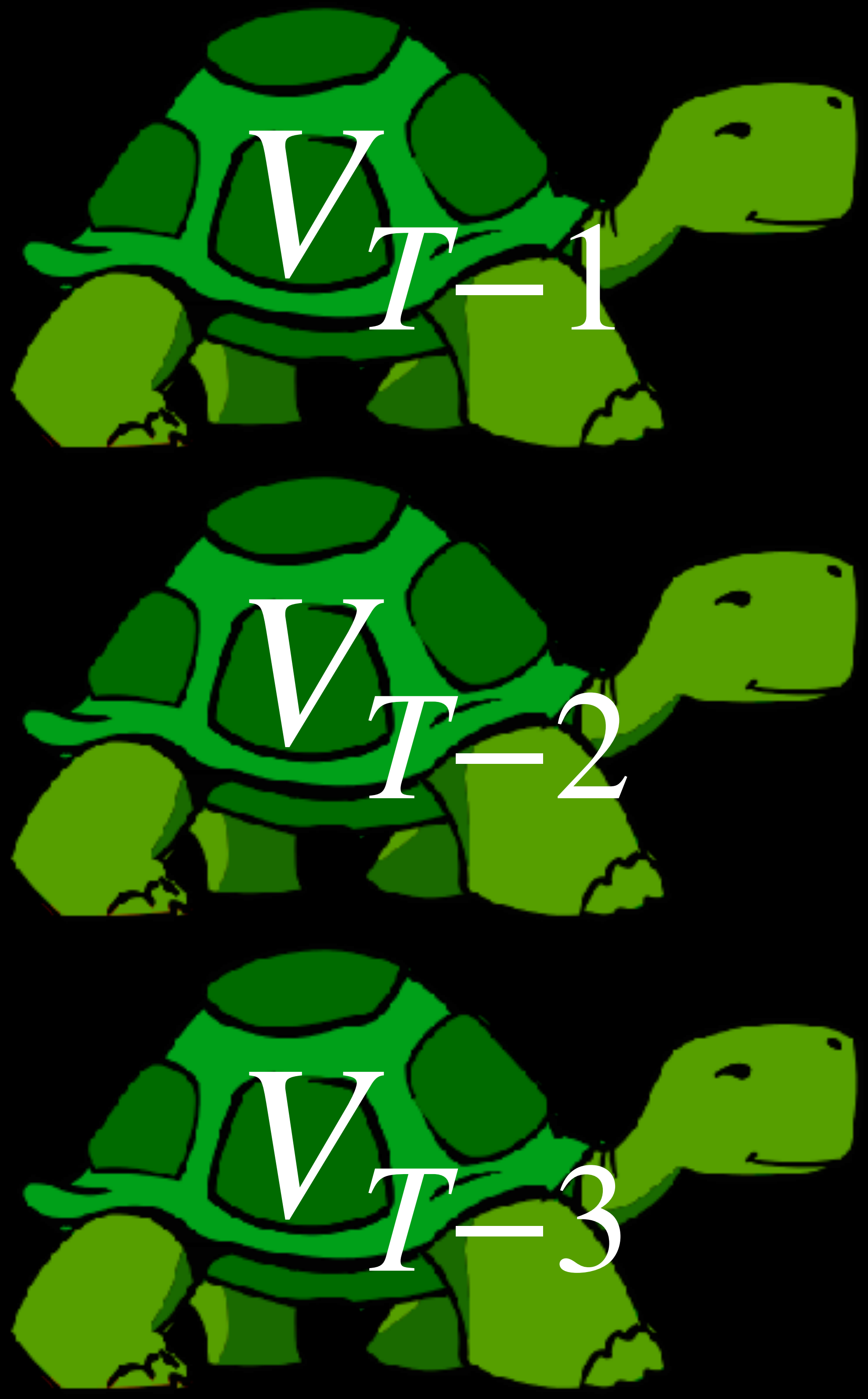
Dynamics of an Inverted Pendulum

Let’s formalize!

It’s quadratics all the way down!

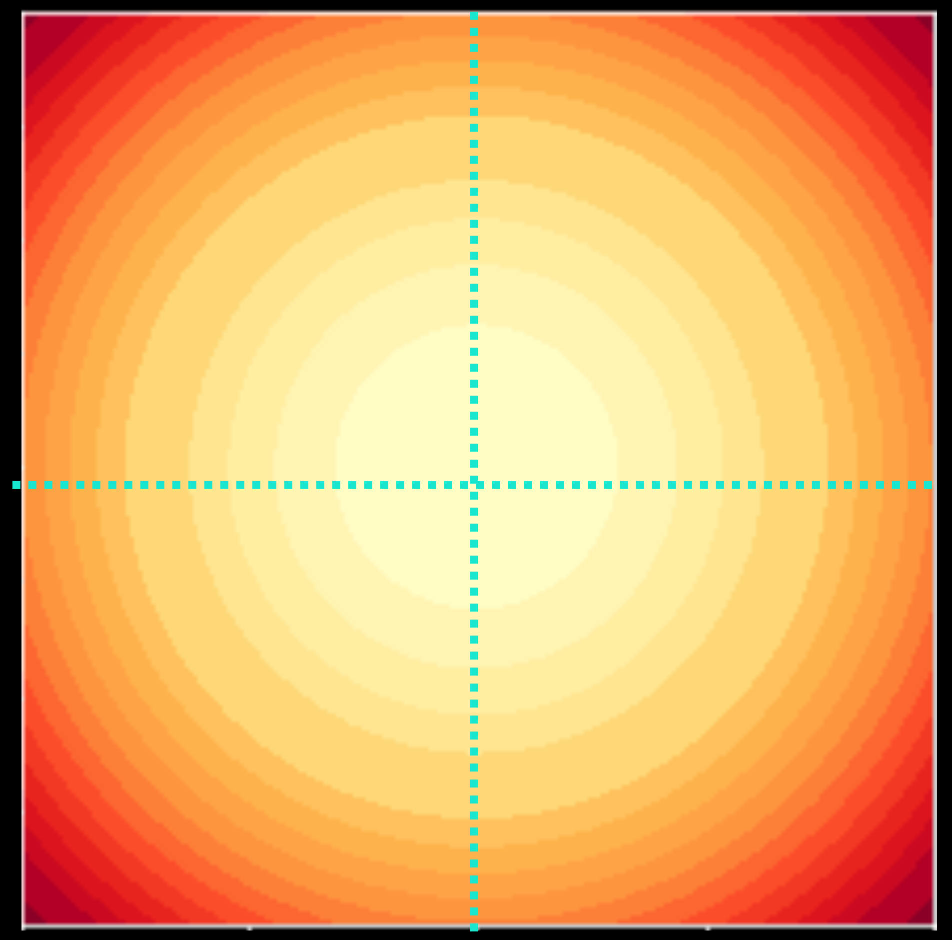
$$K_t = (R + B^T V_{t+1} B)^{-1} B^T V_{t+1} A$$

$$V_t = Q + K_t^T R K_t + (A + BK_t)^T V_{t+1} (A + BK_t)$$



The LQR Algorithm

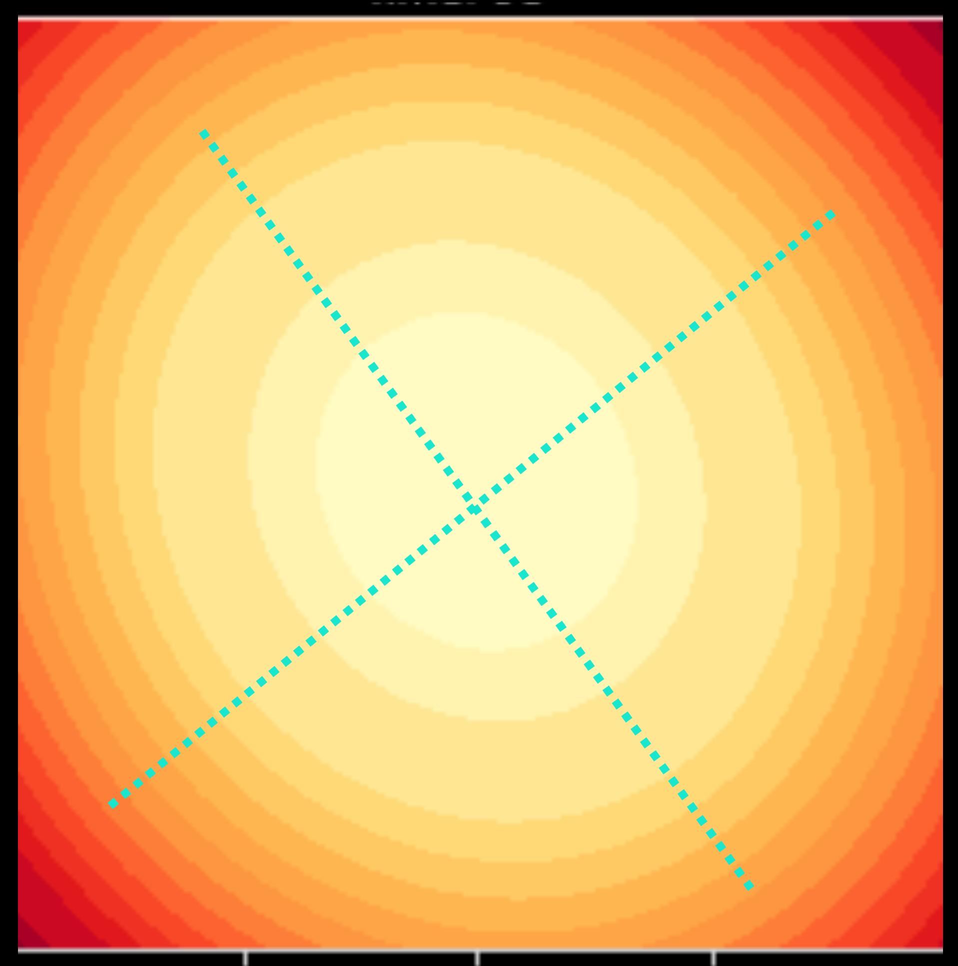
Initialize $V_T = Q$



For t = T-1, ..., 1

Compute gain matrix

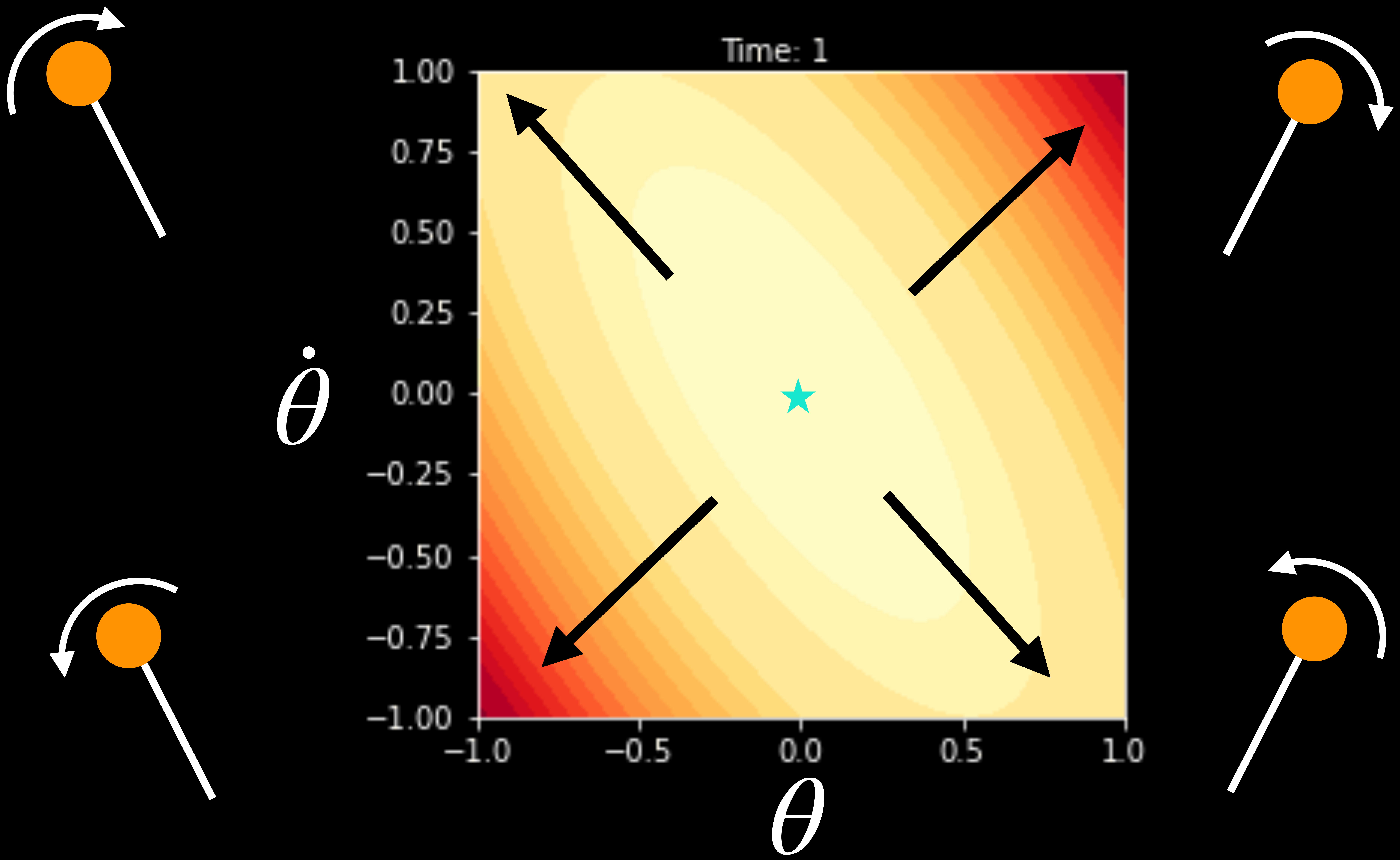
$$K_t = (R + B^T V_{t+1} B)^{-1} B^T V_{t+1} A$$



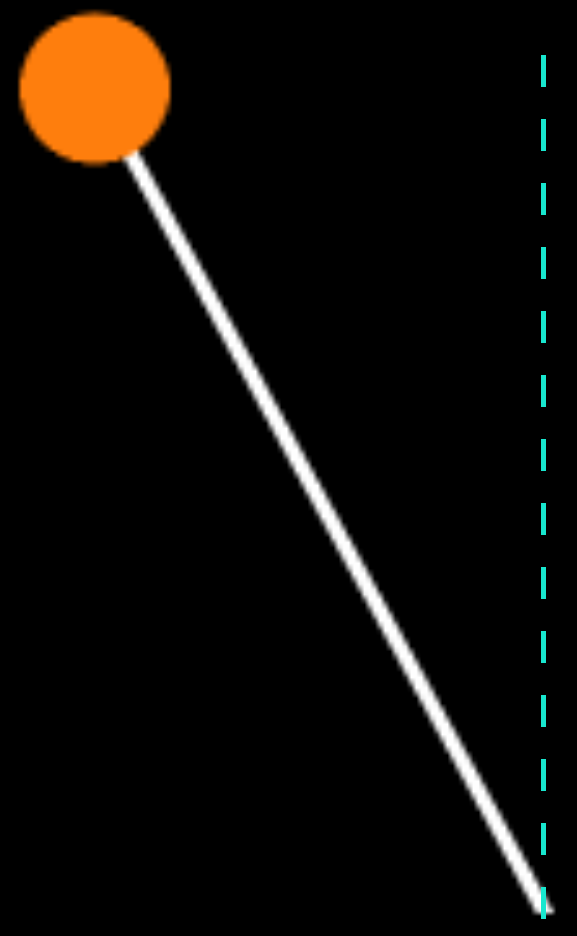
Update value

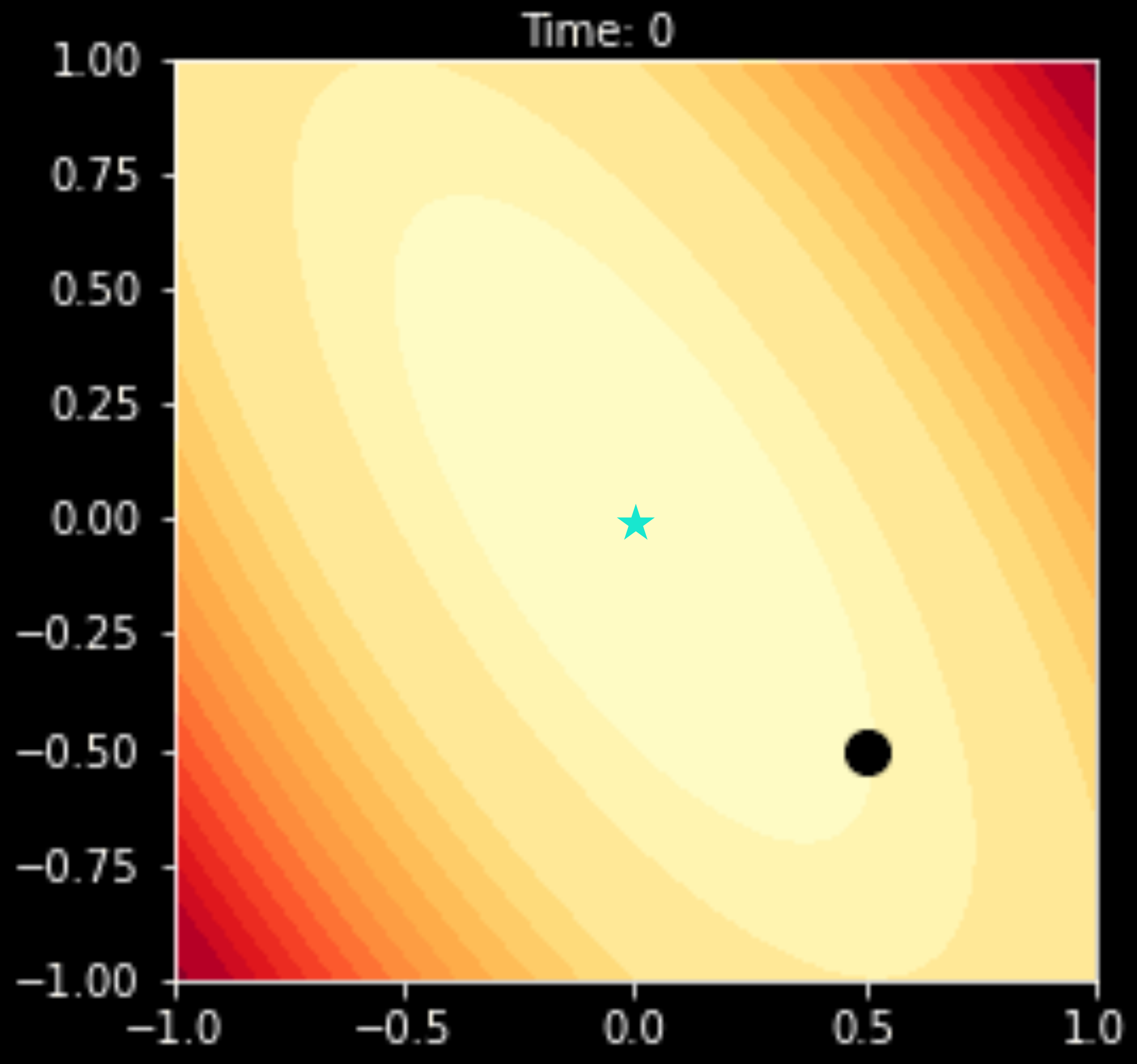
$$V_t = Q + K_t^T R K_t + (A + BK_t)^T V_{t+1} (A + BK_t)$$

Value Iteration for Inverted Pendulum

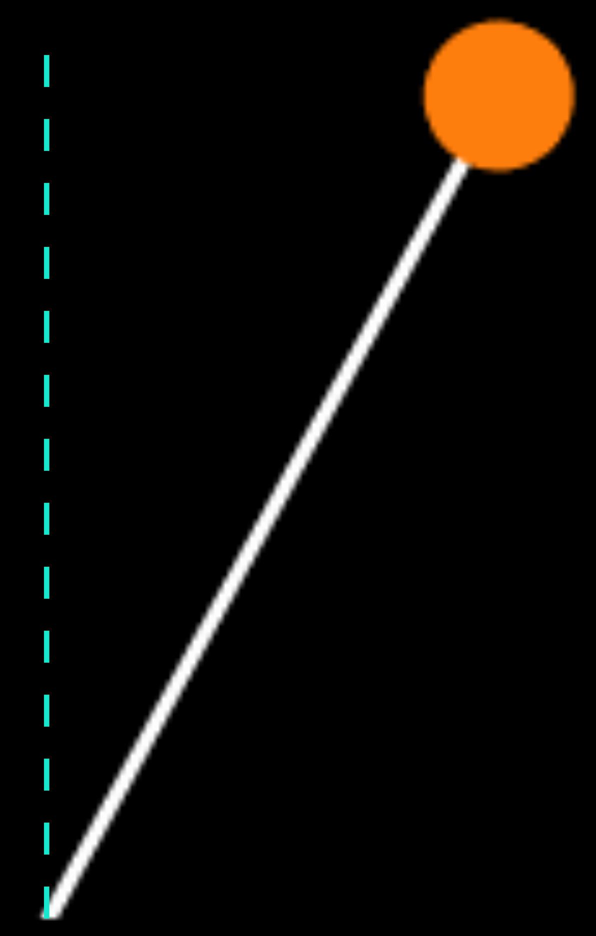


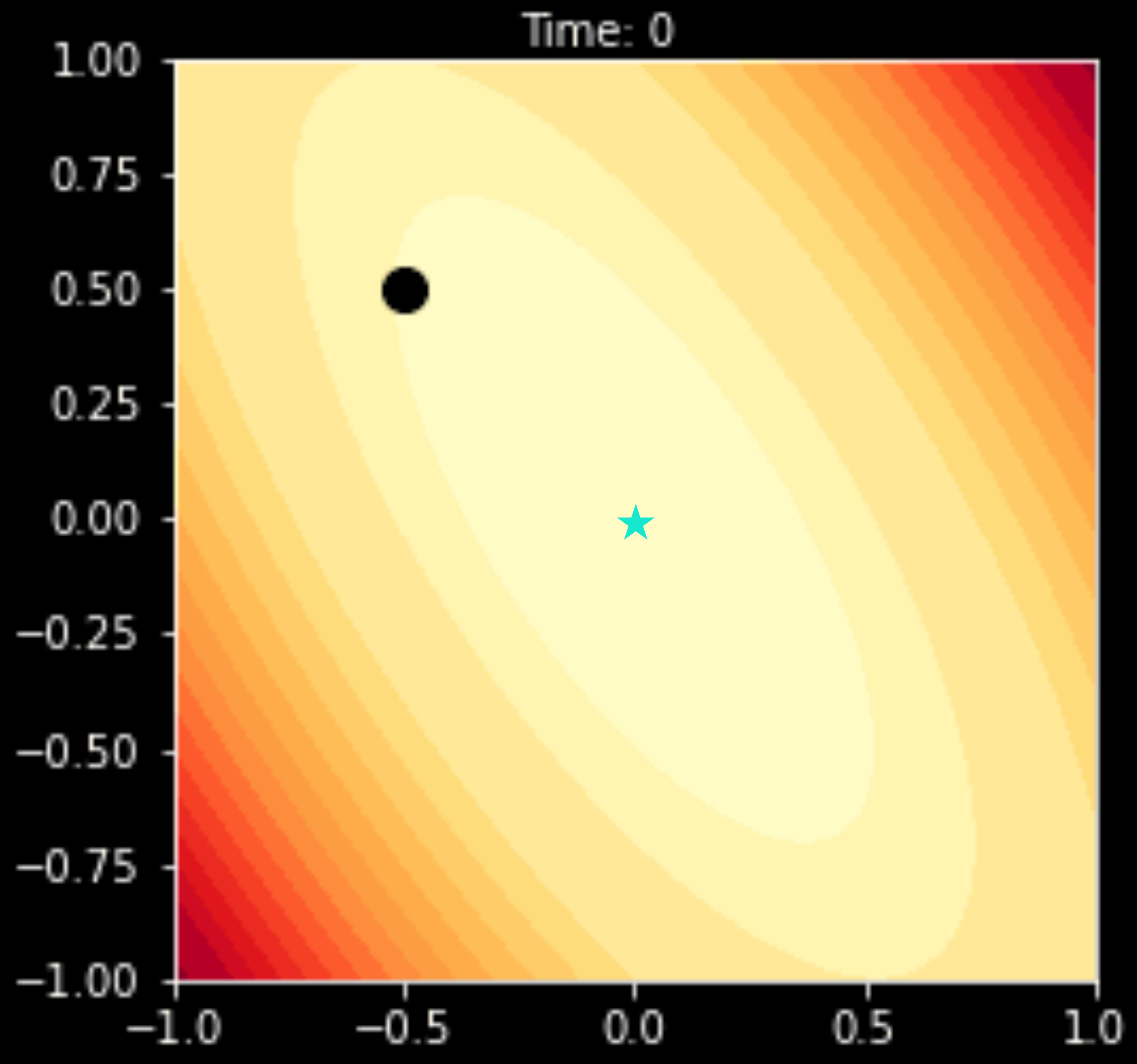
An Easy Starting Point



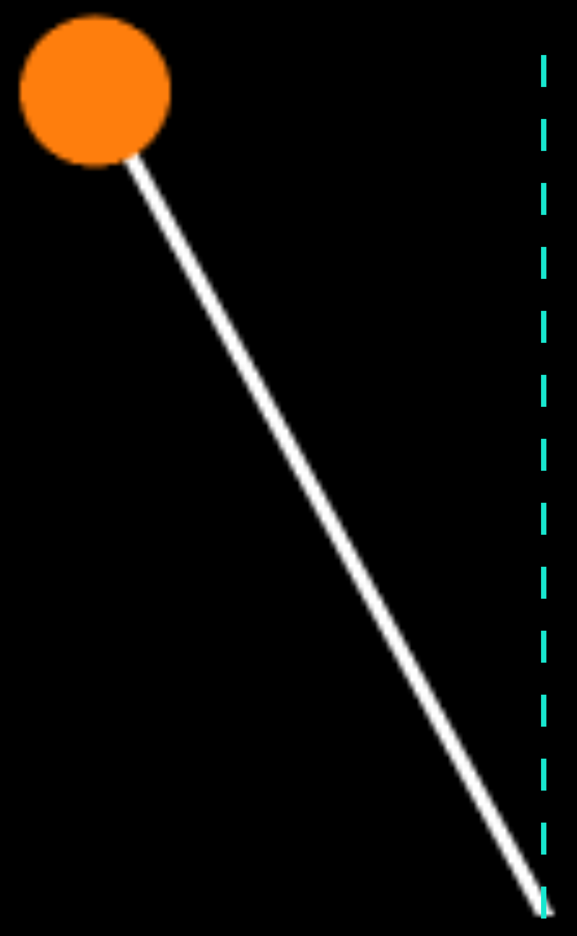


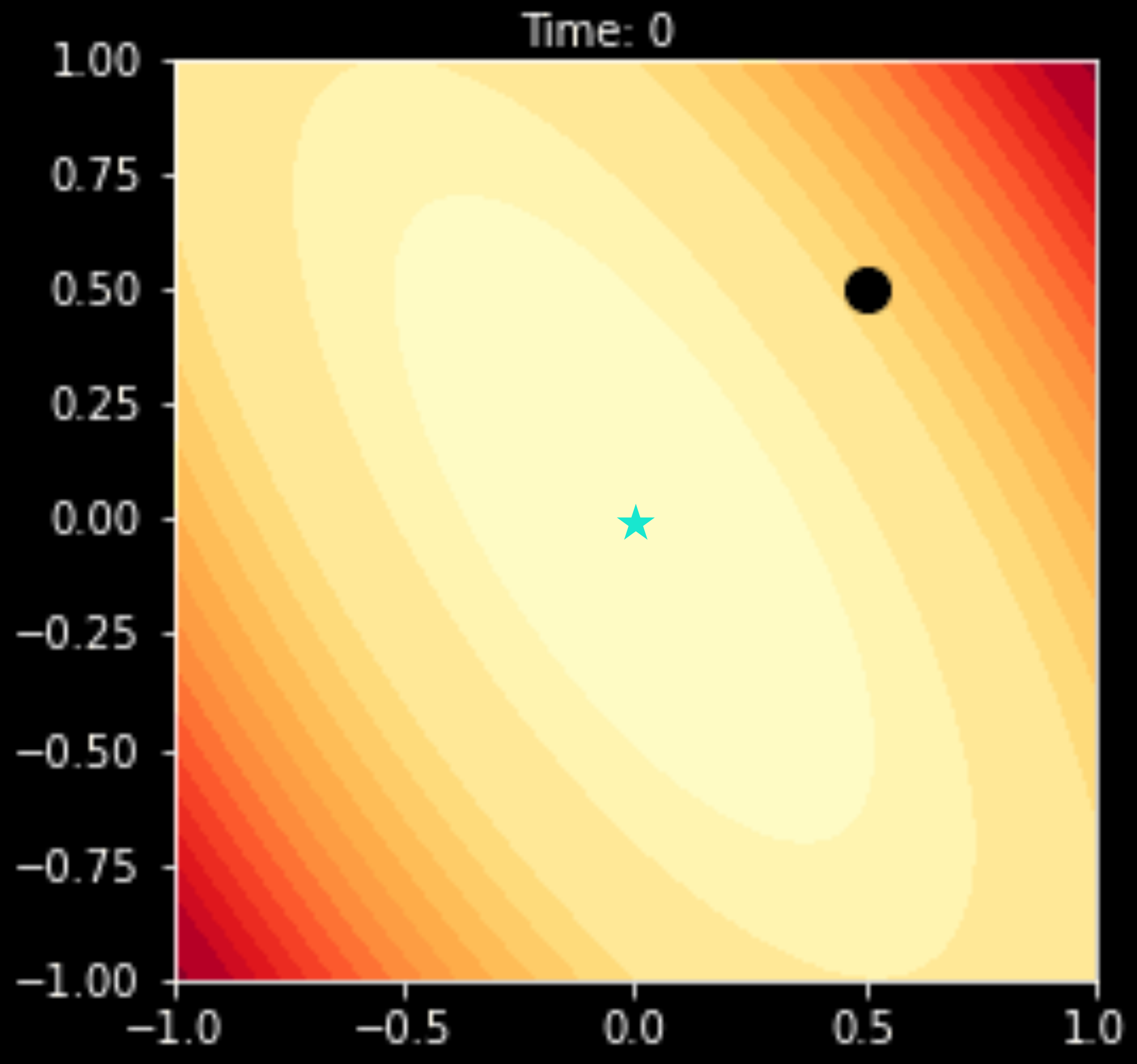
Another Easy Starting Point



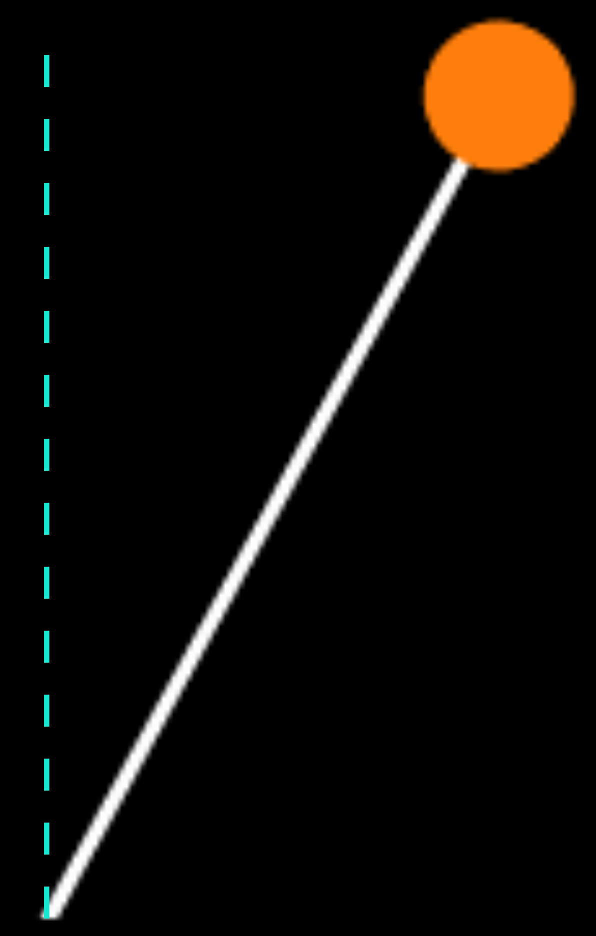


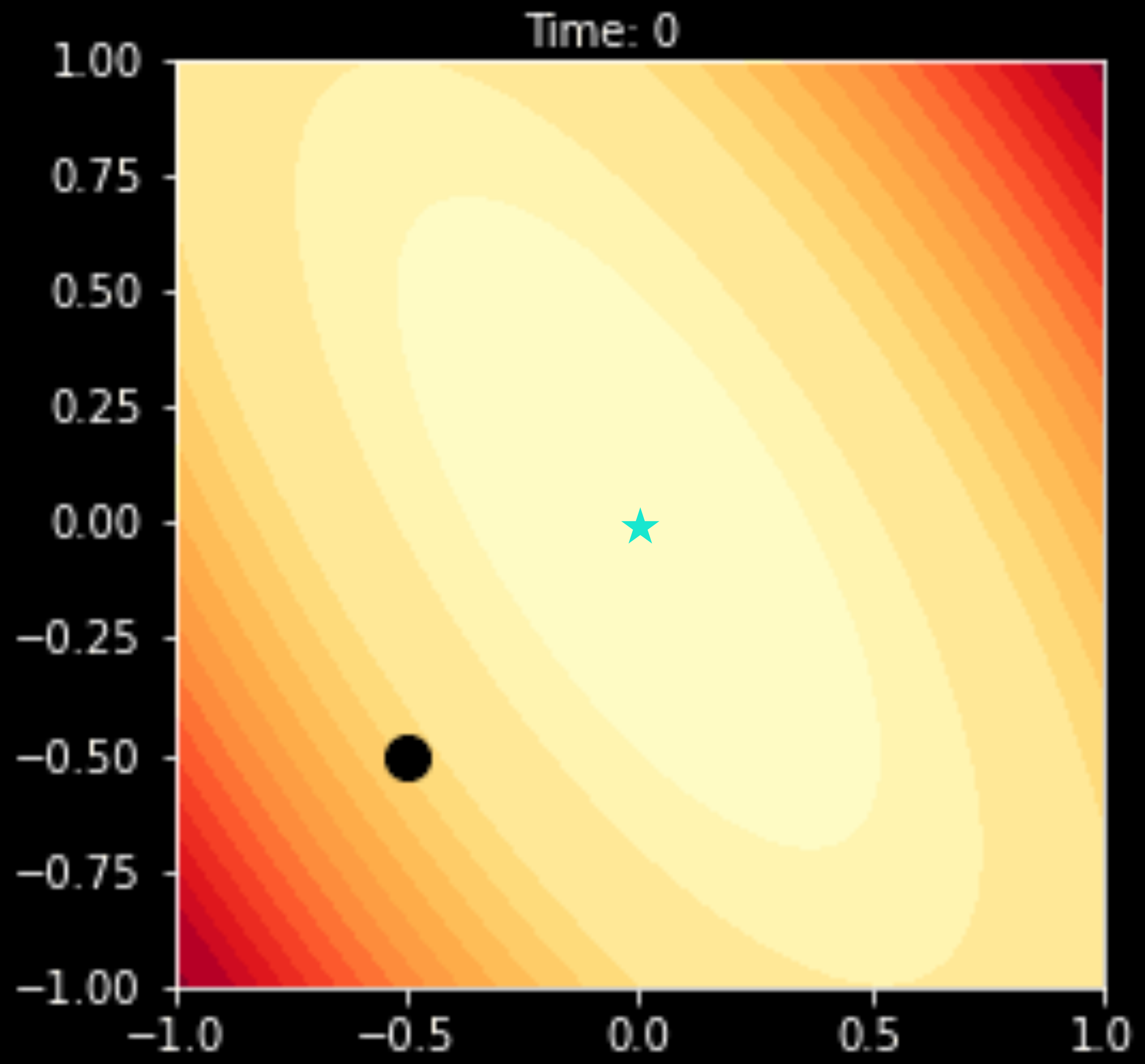
A Hard Starting Point



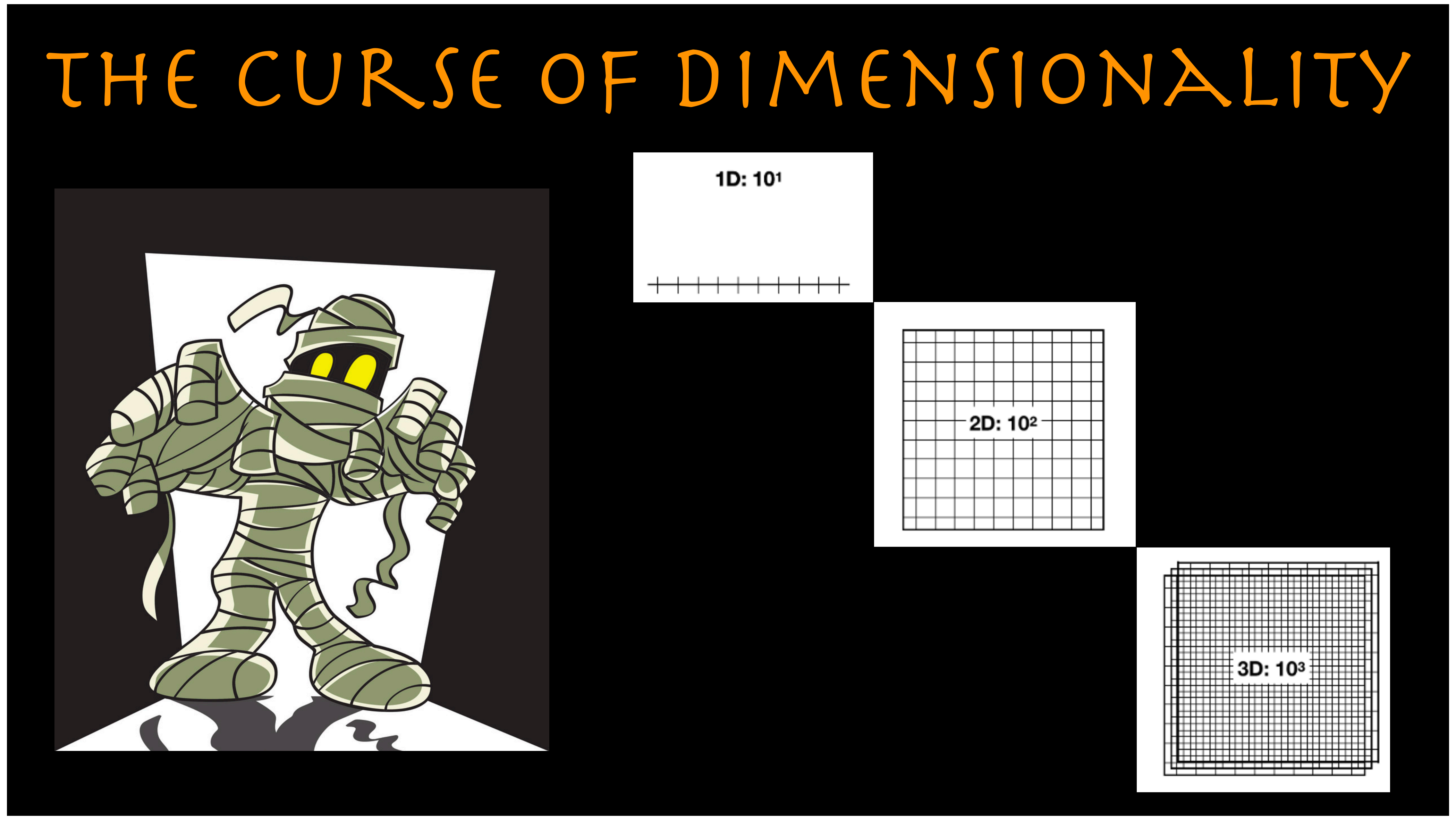


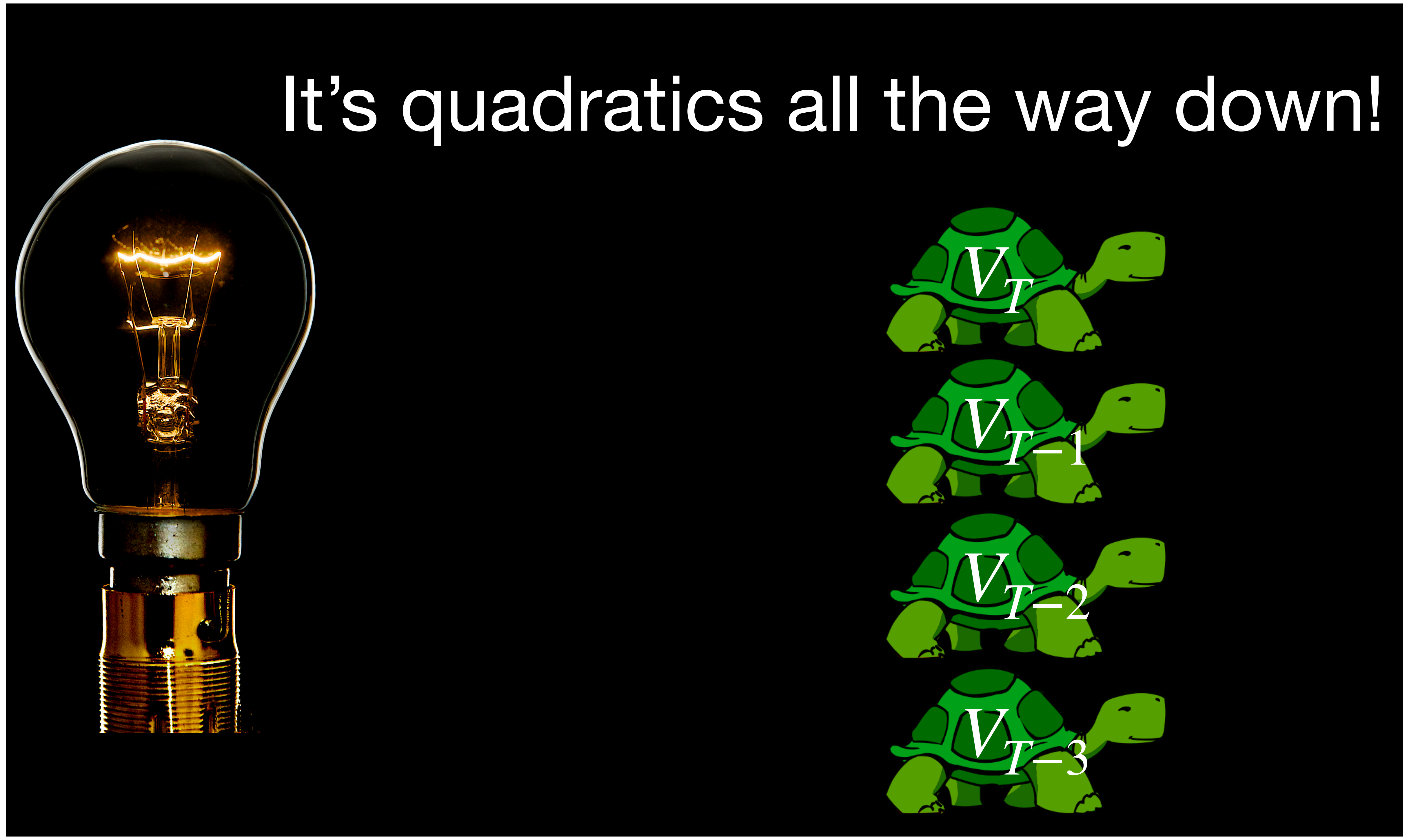
Another Hard Starting Point

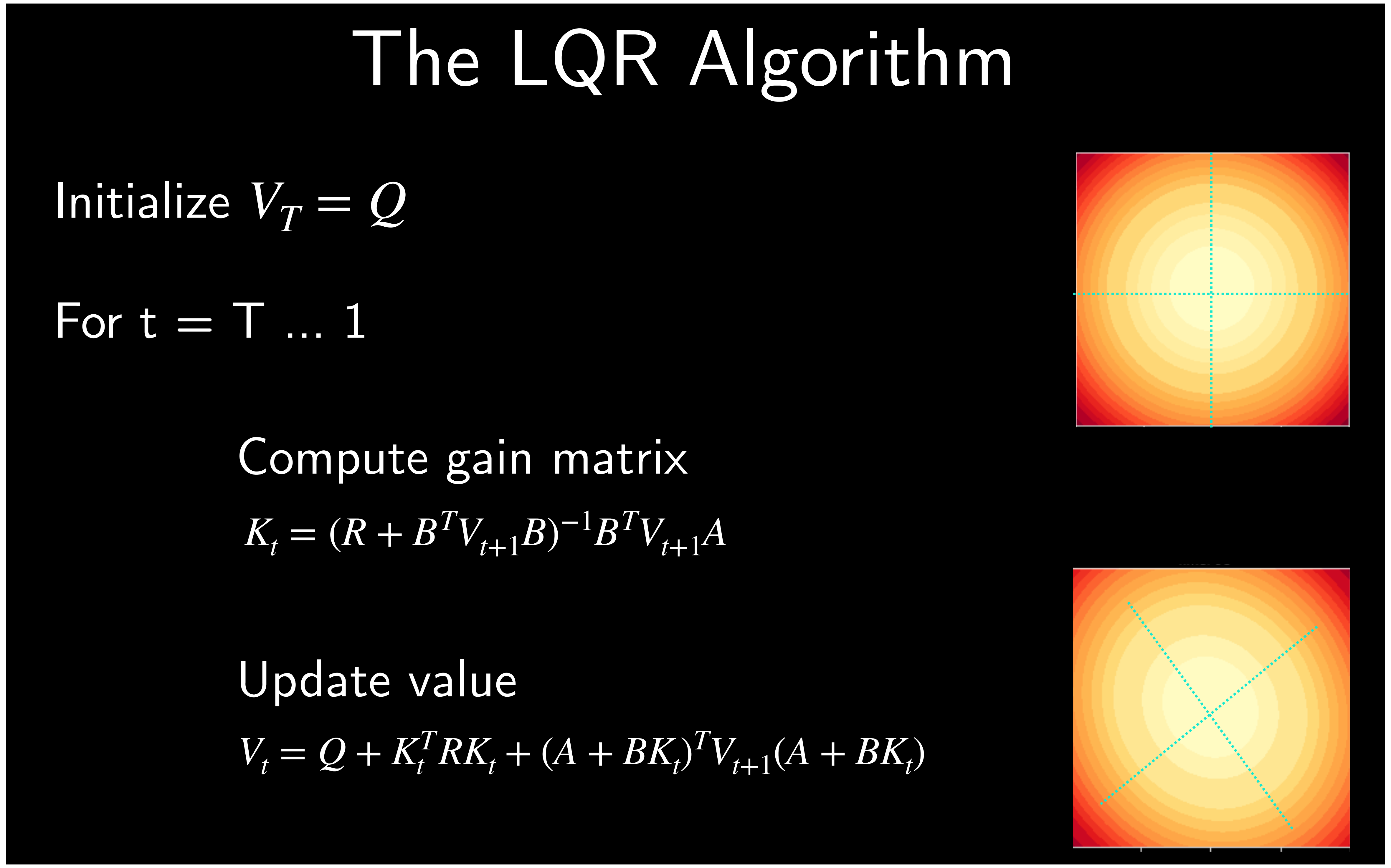




tl;dr


THE CURSE OF DIMENSIONALITY
1D: 10^1
2D: 10^2
3D: 10^3


It's quadratics all the way down!
V_T
V_{T-1}
V_{T-2}
V_{T-3}


The LQR Algorithm
Initialize $V_T = Q$
For t = T … 1
Compute gain matrix
$K_t = (R + B^T V_{t+1} B)^{-1} B^T V_{t+1} A$
Update value
$V_t = Q + K_t^T R K_t + (A + BK_t)^T V_{t+1} (A + BK_t)$