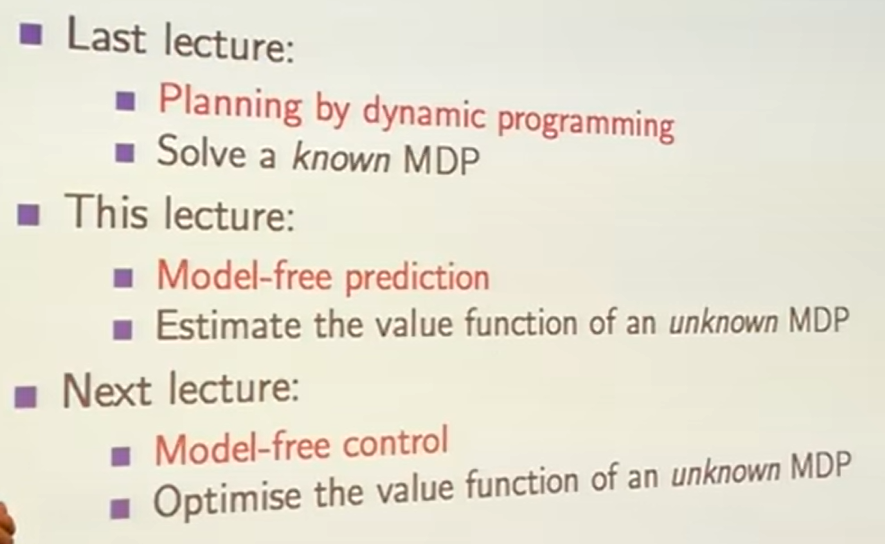
## horizontal line

DP and Monte Carlo Learning & Temporal Difference

21.10.2025



# 

# 

# 

# MDP

**A Markov Decision Process (MDP) is a mathematical framework used to describe how an agent makes decisions in a situation where outcomes are partly random and partly under its control. It’s the foundation of many reinforcement learning (RL) systems.**

🧠 Why MDP Matters in RL

MDPs help formalize decision-making problems where:

* The agent interacts with an environment.
* Each action leads to a new situation (state).
* The agent gets feedback (reward) based on its actions.
* The goal is to **maximize total reward over time**.

🧩 Components of an MDP

An MDP has **five key parts**:

States(S) , Actions(A) , Transition Function(P) , Reward Function(R) , Policy()

🔁 Markov Property

The **Markov property** means:

The next state depends only on the current state and action — not on the full history.

This simplifies learning because the agent doesn’t need to remember everything, just the current situation.

🧠 How MDP Helps in RL

In RL, the agent:

1. Observes the current **state**
2. Chooses an **action** using its **policy**
3. Receives a **reward**
4. Moves to a **new state**
5. Updates its policy to improve future decisions

# Learning by Dynamic Programming in RL

Dynamic Programming (DP) in Reinforcement Learning (RL) is a method where an agent learns the best actions by using a complete model of the environment and solving smaller subproblems step-by-step. It’s like planning ahead using full knowledge of how the world works.

🧠 What Is Dynamic Programming in RL?

Dynamic Programming is a **model-based** approach in RL. That means:

* The agent **knows everything** about the environment — how actions lead to new states and rewards.
* It uses this knowledge to **compute the best strategy** (called a policy) by solving mathematical equations.

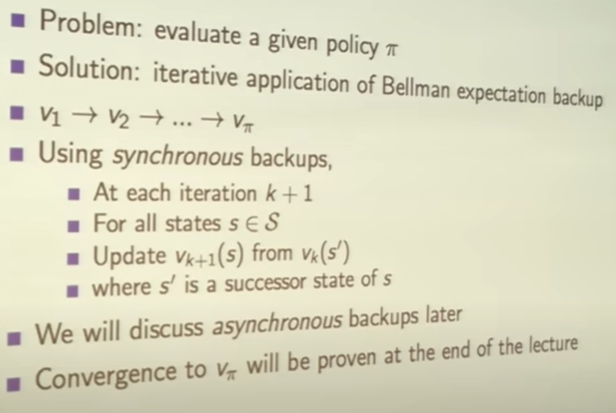
DP is used to:

* **Evaluate policies** (how good a strategy is)
* **Improve policies** (make them better)
* **Find optimal policies** (the best possible strategy)

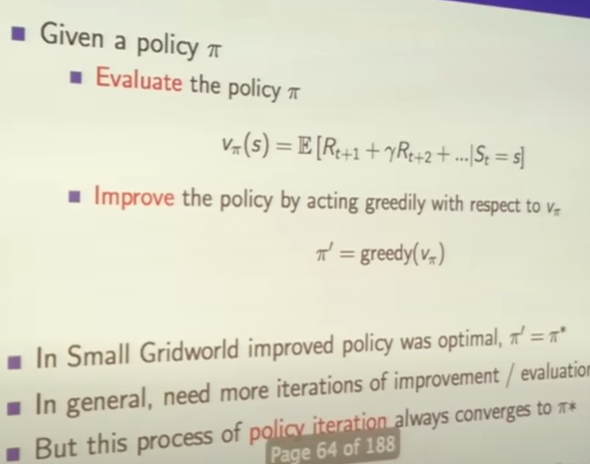
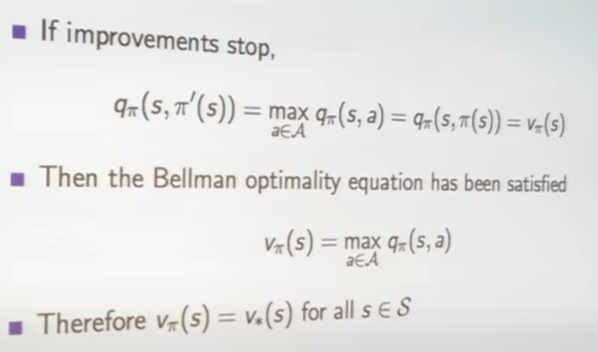
🧩 How Does It Work?

DP breaks the learning process into **smaller steps** and solves them recursively. The two main techniques are:

1. **Policy Evaluation**

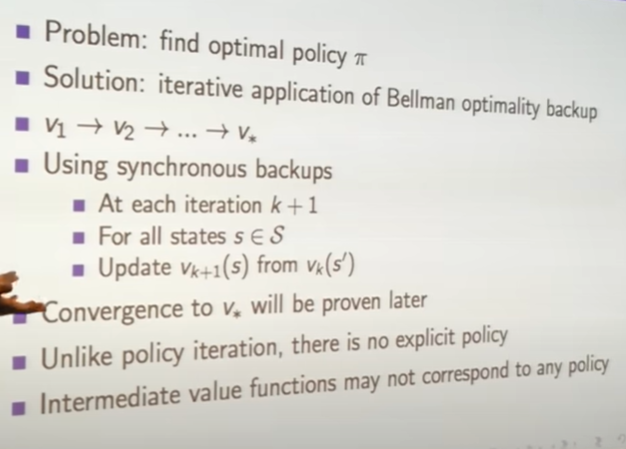
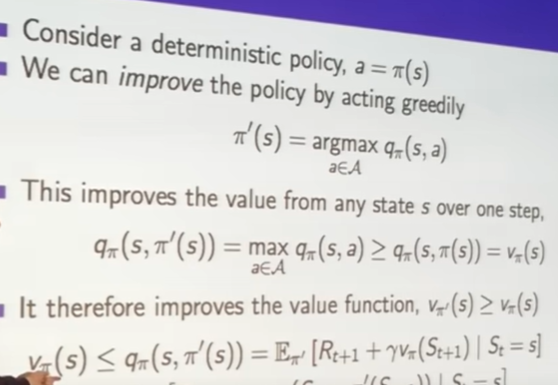
* Calculates how good a given policy is.
* Uses the **Bellman equation** to estimate the value of each state.

2. **Policy Improvement**

* Updates the policy by choosing better actions based on the value estimates.

3. **Value Iteration**

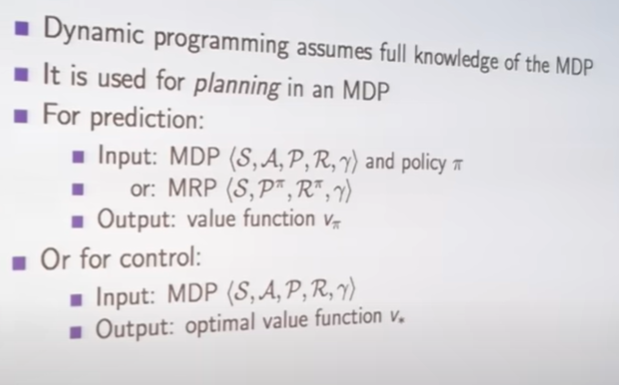
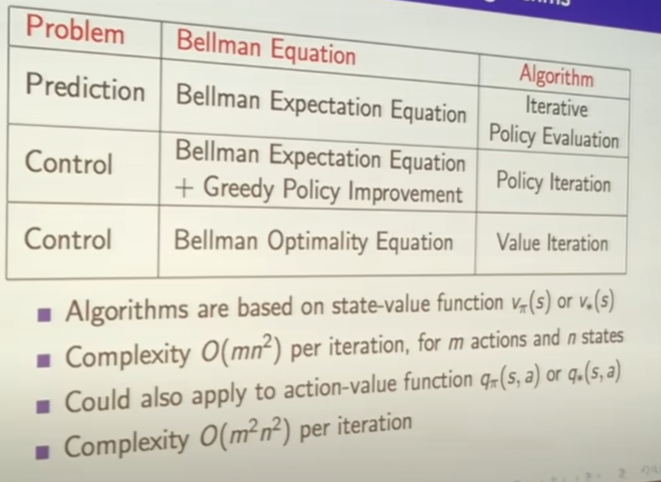
* Combines evaluation and improvement in one loop.
* Keep updating values and policies until they converge to the best solution.

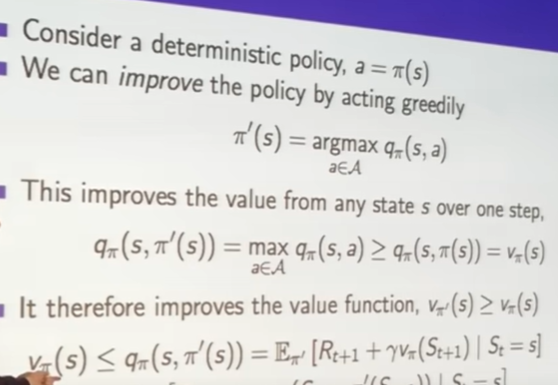


📦 Requirements

To use DP, you need:

* A **complete model** of the environment (called a Markov Decision Process or MDP)
* Knowledge of:
* All possible states
* All possible actions
* Transition probabilities (how likely you move from one state to another)
* Reward function

This makes DP **powerful but limited** — it works best in small, well-defined environments.



# Monte Carlo Learning

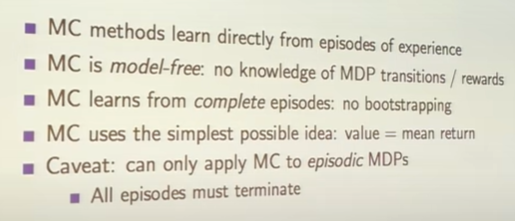
🧠 What Is Monte Carlo Learning?

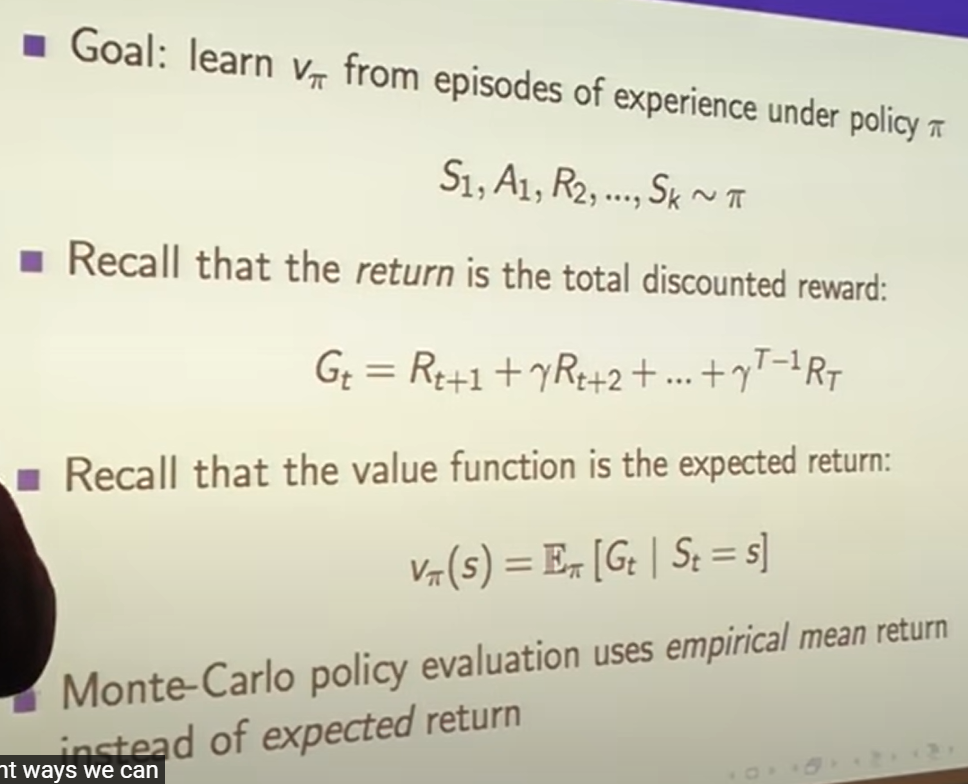
Monte Carlo learning is a method used in **reinforcement learning (RL)** where an agent learns by:

* **Running full episodes** (from start to finish)
* **Observing the total reward** it gets at the end
* **Updating its strategy** based on how good or bad that episode was

It’s named after the **Monte Carlo Casino** because it relies on **randomness and probability**, just like gambling.

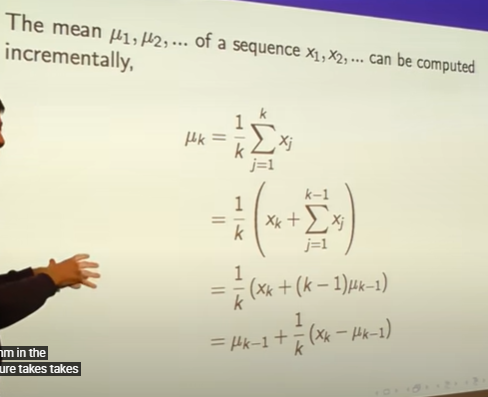
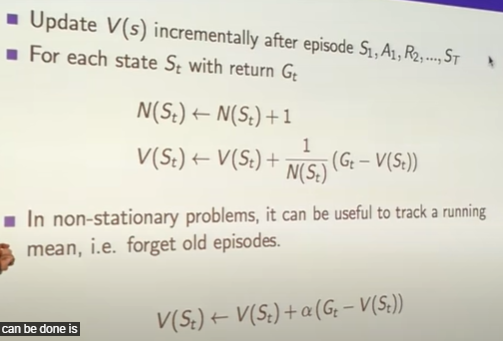
🔁 Key Features

* **No need for a model** of the environment — it learns directly from experience.
* **Works with complete episodes** — it waits until the end before learning.
* **Uses averaging** — it keeps track of how good each action was on average.



Every time-step to that state is visited in an episode.





# Temporal Difference Method

