AI in Mathematics
Lecture 13
Reinforcement Learning

Bar-Ilan University
Nebius Academy | Stevens Institute of Technology

June 24, 2025

About This Course

~~1 week: Intro~~

~~2 weeks: Classic ML~~

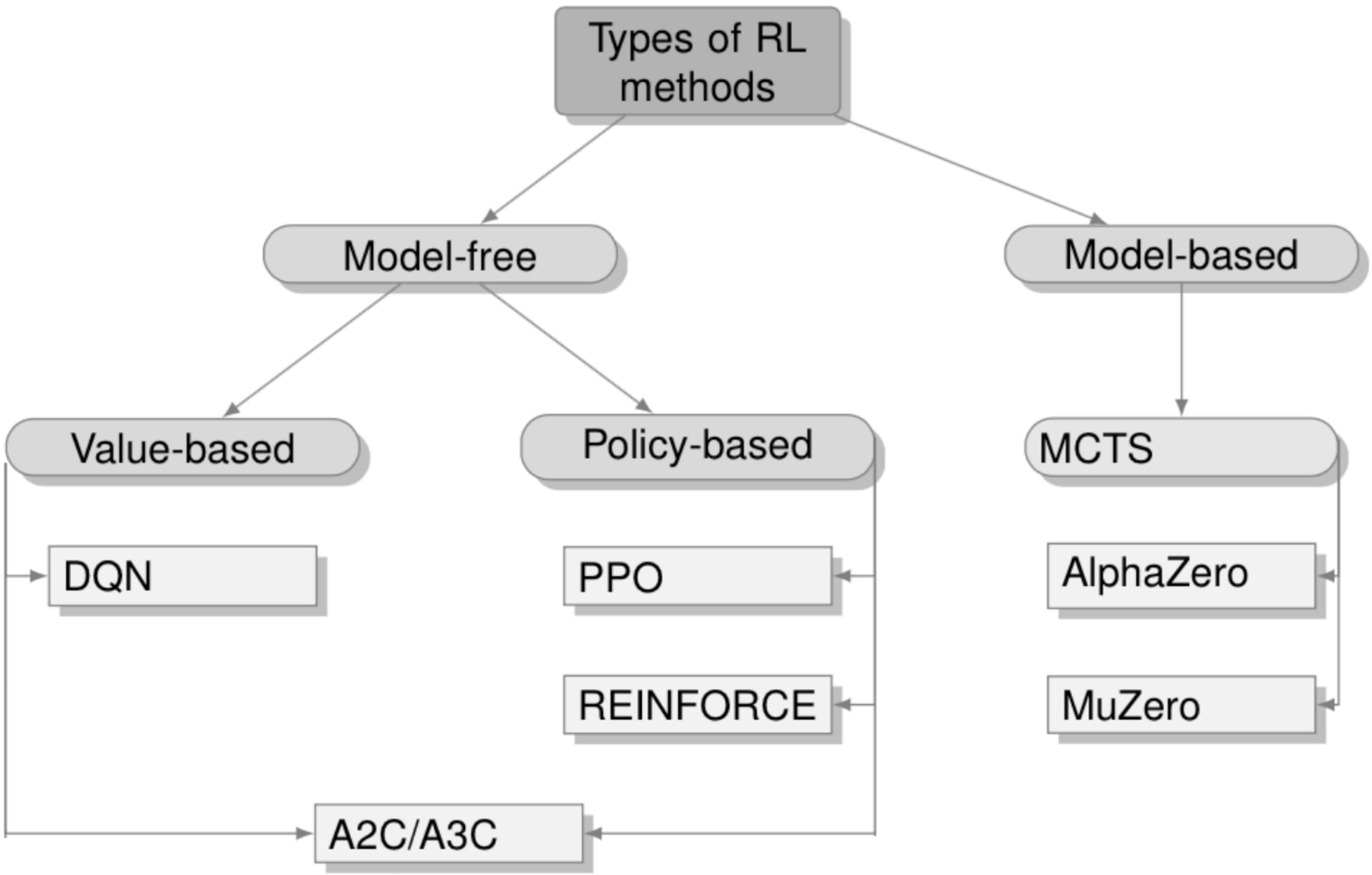
~~2 weeks: Deep Learning in Mathematics~~

~~4 weeks: Math as an NLP problem (LLMs etc.)~~

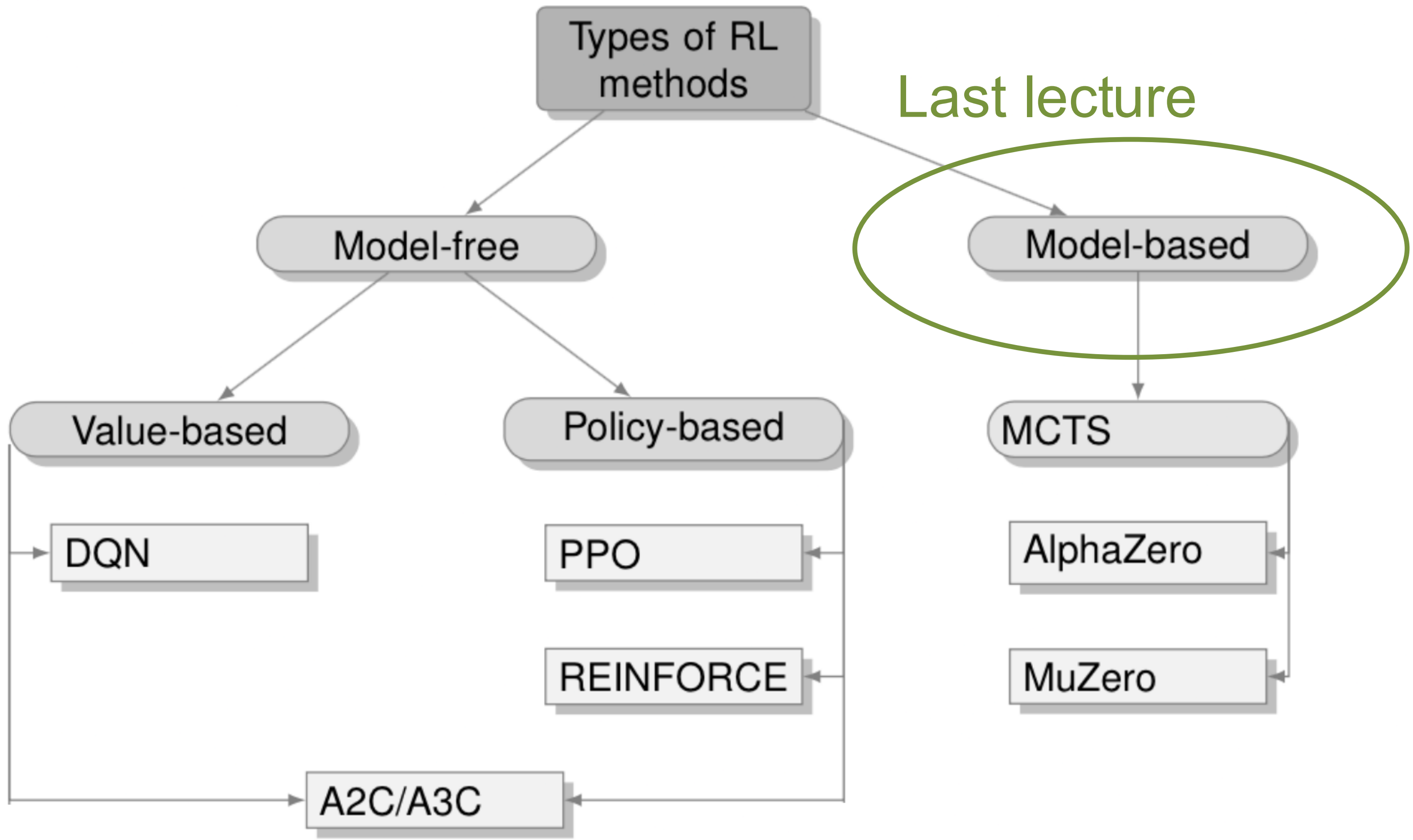
4 weeks: Reinforcement Learning (RL) in Math

Last lecture!!!

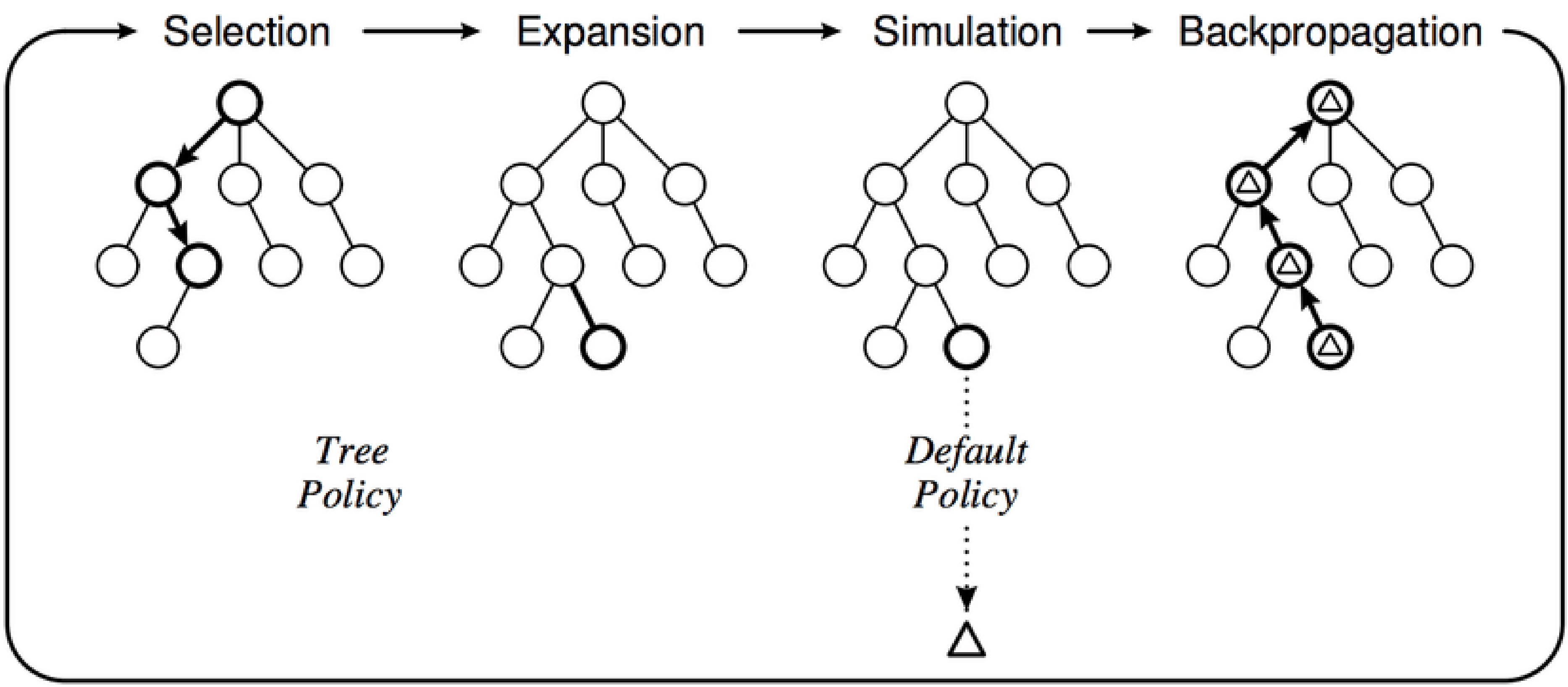
RL Algorithms



RL Algorithms



Monte Carlo Tree Search (MCTS)



Idea

The Challenge:

- Traversing the full action tree is exponential in depth — impossible in most real-world problems.
- We want to explore without visiting every node.

What we need:

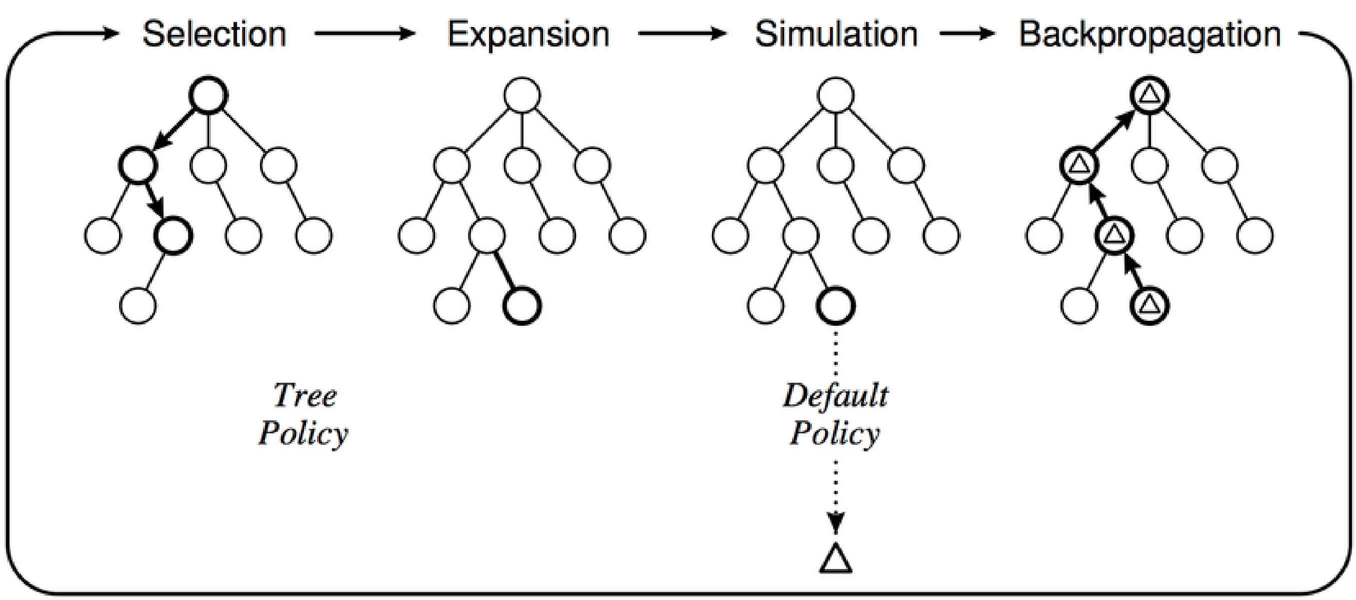
- A method that selectively explores the tree
- Visits the most promising nodes more often
- Still guarantees that every part of the tree is explored on average

MCTS

What is MCTS?

- Tree search guided by simulation
- Builds tree incrementally using Monte Carlo samples
- Repeats 4 steps:

Select → **Expand** → **Simulate** → **Backpropagate**



Selection

- Use UCB1 to pick child:

$$score(s,a) = \frac{Q(s,a)}{N(s,a)} + c \cdot \sqrt{\frac{\ln N(s)}{N(s,a)}}$$

Balance exploration & exploitation:

Where:
$Q(s,a)$: cumulative reward from action a
$N(s,a)$: number of times action a has been tried
$N(s)$: total number of visits to state s
c: exploration constant (tunable)

Expansion

If all children of this vertex have been tried:

- MCTS does not expand anything at this level.
- Uses the UCB1 formula to select one of the existing child nodes.

If we arrive at a state s that is already in the tree, but not fully expanded. We choose **one unvisited action** a from this state and apply it:

- Simulate the environment to reach next state s'.
- Add s' as a new child node in the tree.

Simulation

We reach a new node s’ that has just been added to the tree. From this node, we simulate a full trajectory.

We use a **default policy**, often:

- A **random policy** (uniformly choose legal actions)
- Or a **simple heuristic policy** (e.g., favor central moves in a game)

This policy should be **fast and lightweight.**

Backpropagation

After a simulation finishes, we have a result: a scalar reward R.

We then walk back up the tree and for each node–action pair (s, a) along this path, we update:

$Q(s, a) \mathrel{+}= R$ cumulative reward from action,

$N(s, a) \mathrel{+}= 1$ number of times action a has been tried.

Final Action Selection

We've run MCTS for **N simulations**, building a partial search tree rooted at the root state.

1. Highest visit count (most common in practice)

$$a^* = \text{argmax}_a\, N(s_0, a)$$

Reflects the action that MCTS has the most confidence in.

2. Highest average reward (less robust, more greedy)

$$a^* = \text{argmax}_a \frac{Q(s_0, a)}{N(s_0, a)}$$

Picks the action with the best estimated value, regardless of visit count.

Monte Carlo Tree Search (MCTS)