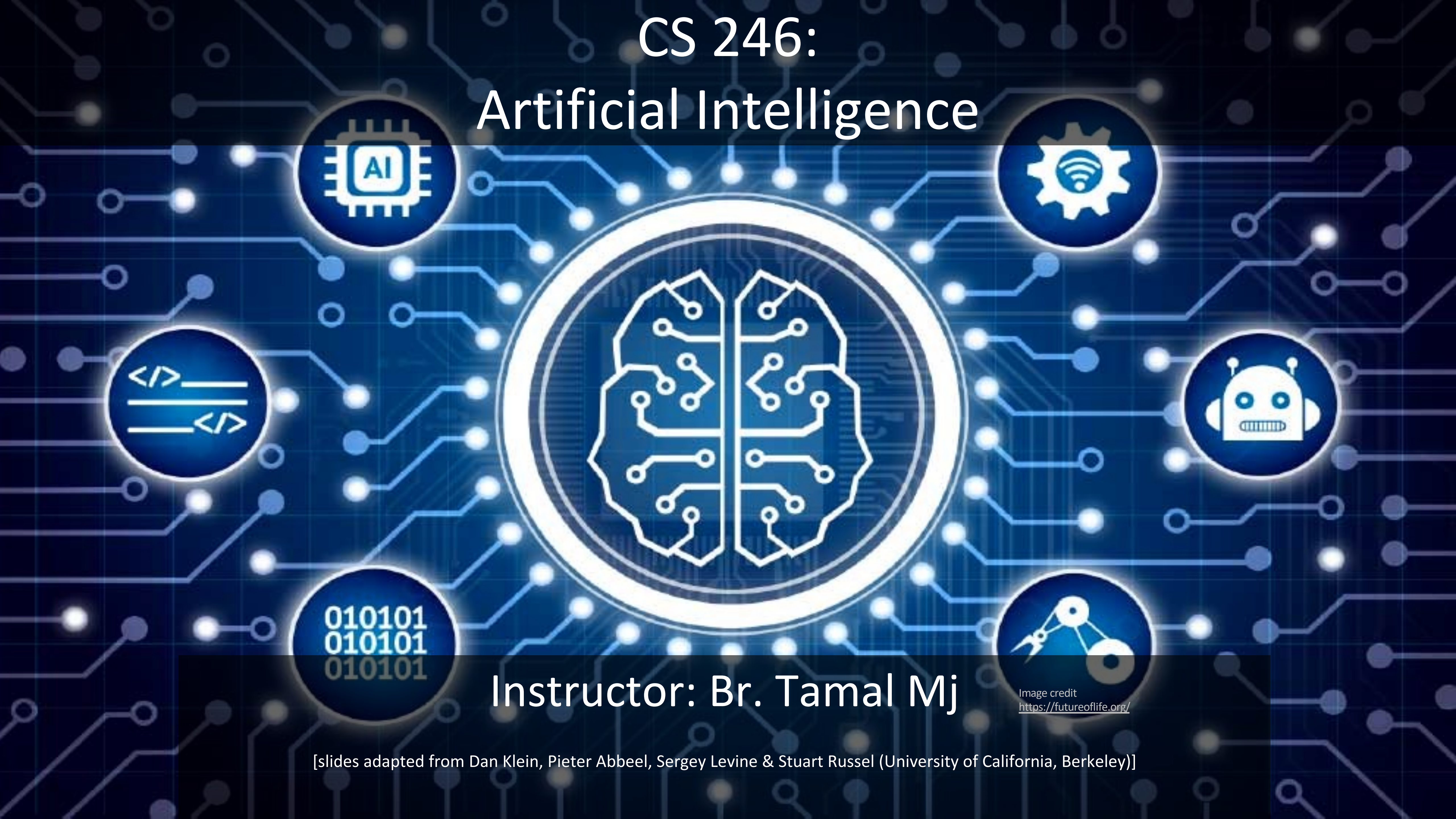
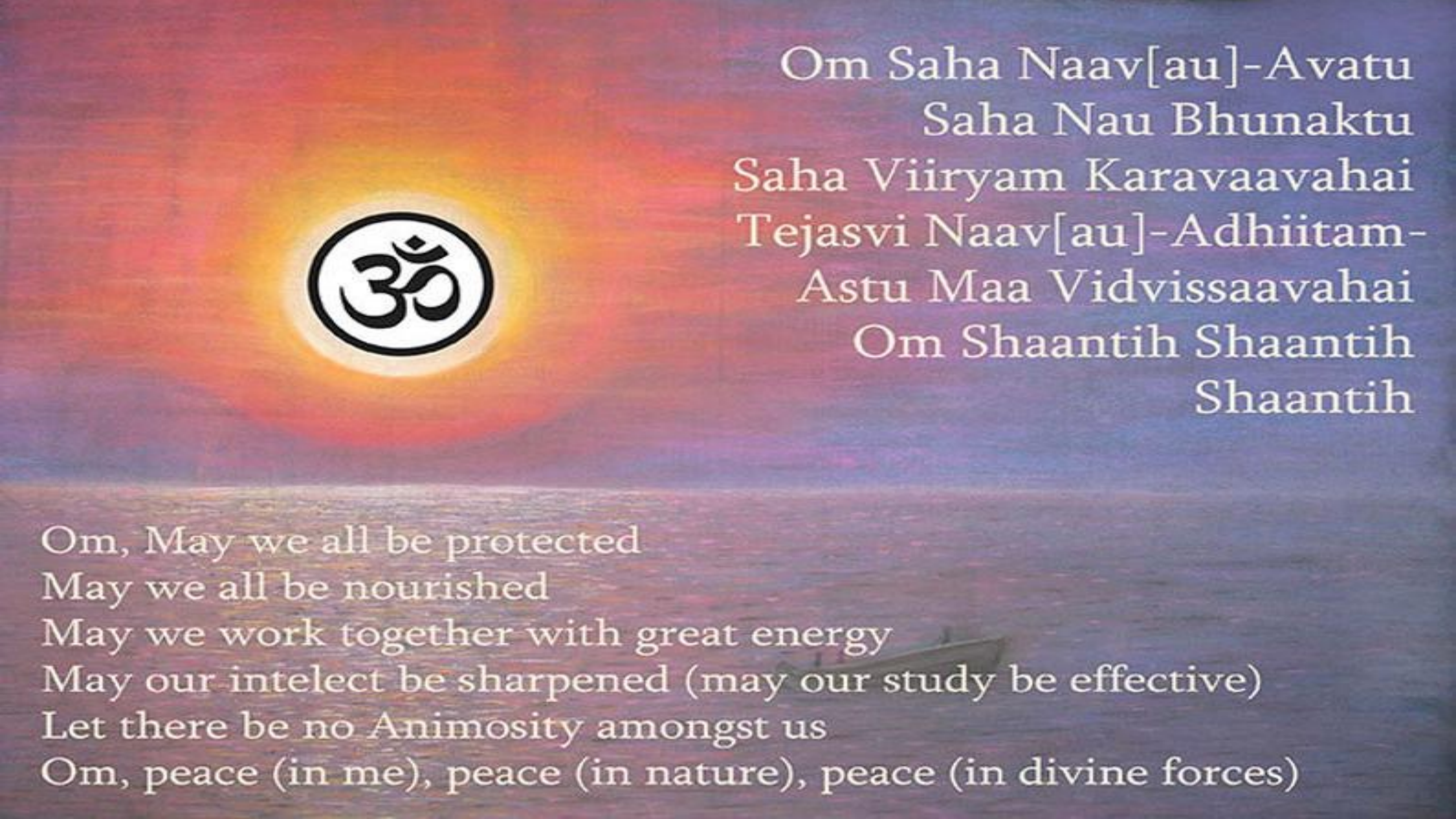
CS 246: Artificial Intelligence



Instructor: Br. Tamal Mj

Image credit https://futureoflife.org/

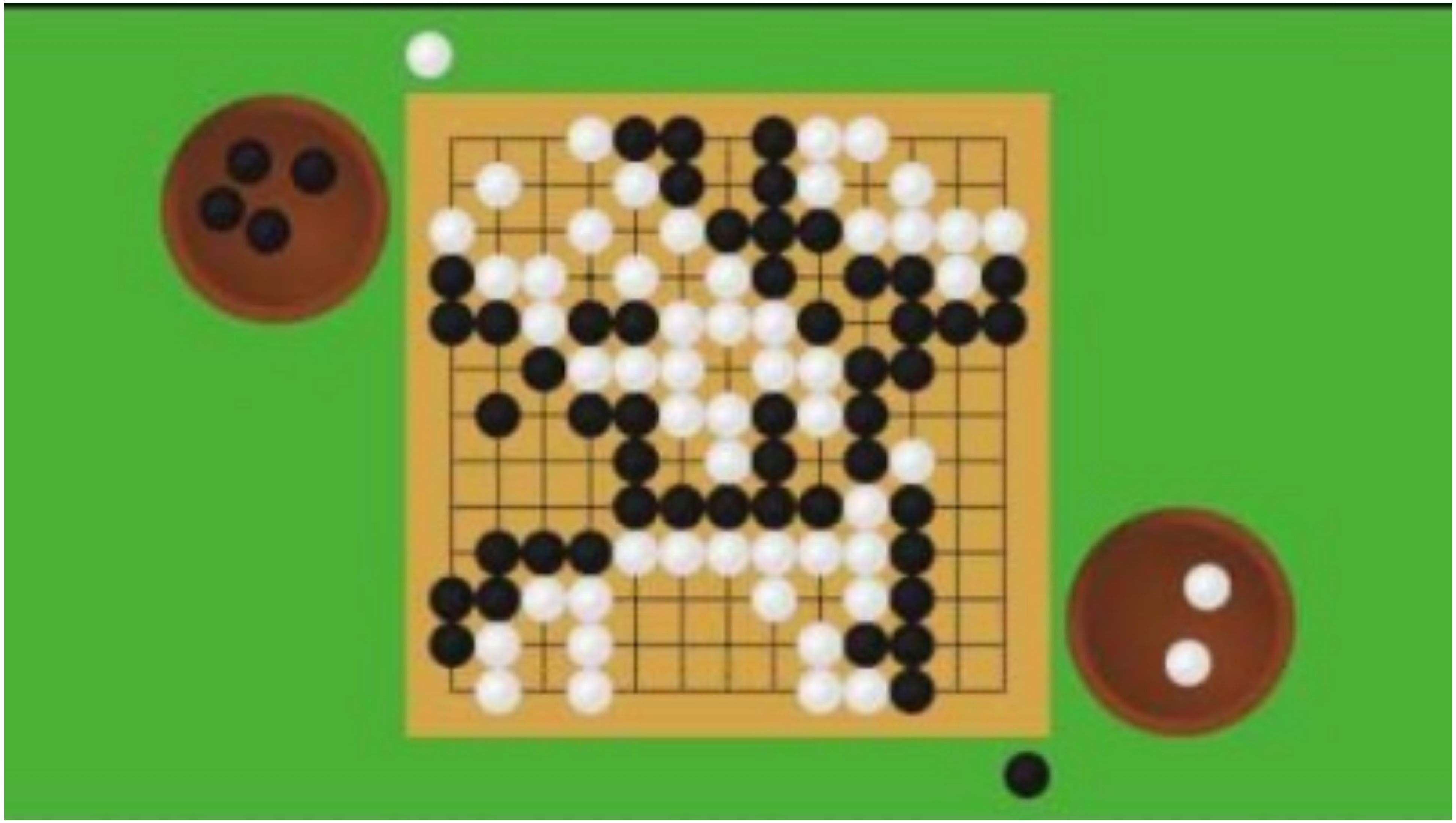
[slides adapted from Dan Klein, Pieter Abbeel, Sergey Levine & Stuart Russel (University of California, Berkeley)]



Om Saha Naav[au]-Avatu
Saha Nau Bhunaktu
Saha Viiryam Karavaavahai
Tejasvi Naav[au]-Adhiitam-
Astu Maa Vidvissaavahai
Om Shaantih Shaantih
Shaantih

Om, May we all be protected
May we all be nourished
May we work together with great energy
May our intelect be sharpened (may our study be effective)
Let there be no Animosity amongst us
Om, peace (in me), peace (in nature), peace (in divine forces)

Go (how to play)



https://www.youtube.com/watch?v=5PTXdR8hLlQ

Go

- Go is the most popular board game in Asia.
- The board is 19x19 and moves are allowed into (almost) every empty square
  - the branching factor starts at 361, which is too daunting for regular alpha–beta search methods.
  - In addition, it is difficult to write an evaluation function because control of territory is often very unpredictable until the endgame.
- Therefore the top programs, such as MOGO, avoid alpha–beta search and instead use Monte Carlo rollouts.

Monte Carlo Tree Search (MCTS)

- **Monte Carlo Tree Search (MCTS)** is a probabilistic, heuristic-driven search algorithm used in AI, combining tree search with reinforcement learning principles.
- **Exploration-Exploitation Trade-off**:
  - **Exploration**: Focuses on the breadth
  - **Exploitation**: Focuses on the depth
- **Balancing Exploration and Exploitation**:
  - MCTS balances these using the **Upper Confidence Bound (UCB)** formula.

Key Concepts of MCTS

1. **Monte Carlo Method**

   This involves using **random simulations** to estimate values or make decisions based on probabilistic outcomes. MCTS uses random simulations to explore the game tree and improve its decision-making.

2. **Tree Search**

   Like other tree search algorithms (e.g., depth-first search, breadth-first search), MCTS builds a **search tree**, where nodes represent game states, and edges represent possible actions from one state to another.

3. **Exploration-Exploitation Trade-off**

Monte Carlo Tree Search (MCTS)

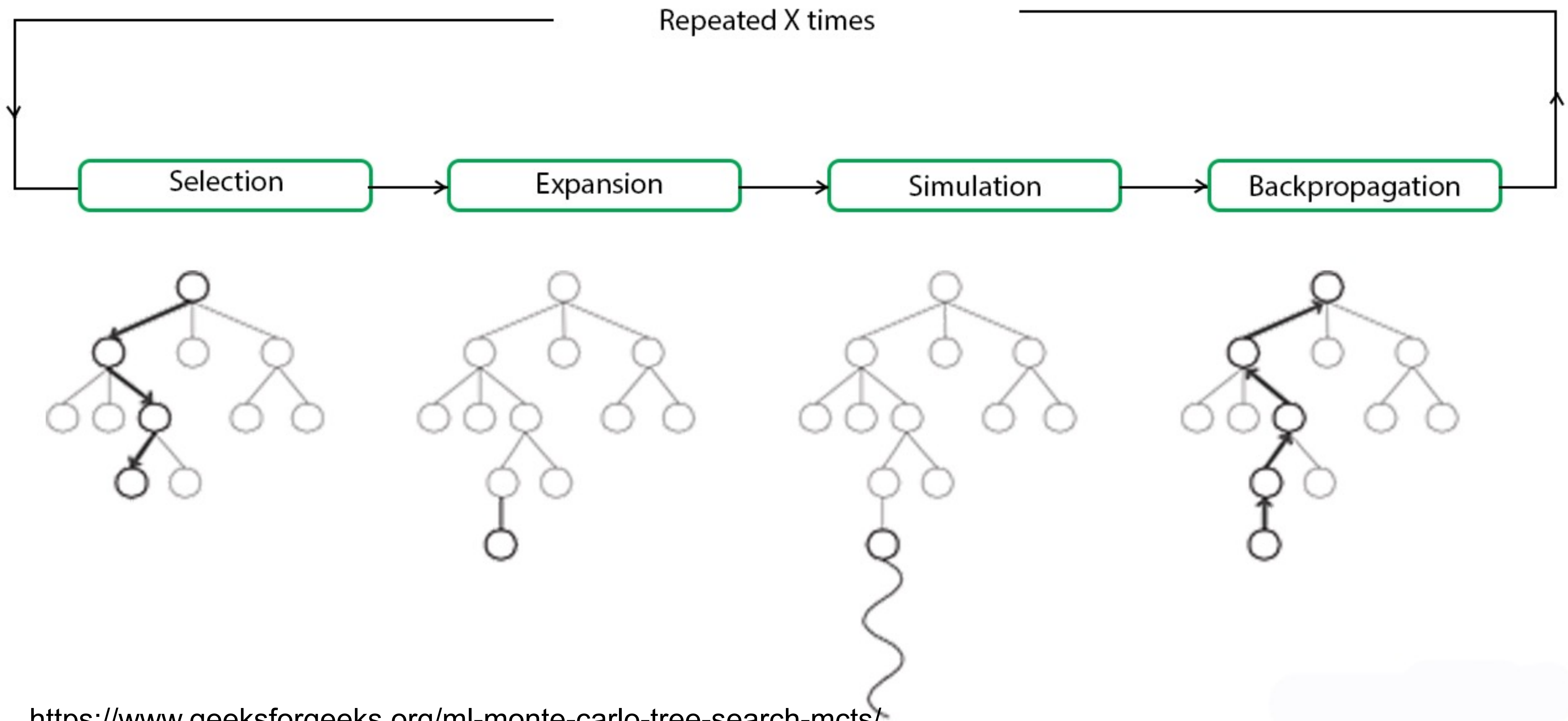
- Methods based on alpha-beta search assume a fixed horizon
  - Pretty hopeless for Go, with b > 300
- MCTS combines two important ideas:
  - ***Evaluation by rollouts*** – play multiple games to termination from a state s (using a simple, fast rollout policy) and count wins and losses
  - ***Selective search*** – explore parts of the tree that will help improve the decision at the root, regardless of depth

Monte Carlo Method

- The Monte Carlo method, which uses random sampling for deterministic problems which are difficult or impossible to solve using other approaches, dates back to the 1940s
- In his 1987 PhD thesis, Bruce Abramson combined minimax search with an *expected-outcome model* based on random game playouts to the end, instead of the usual static evaluation function.

History

- In 2006, inspired by these predecessors,Rémi Coulom described the application of the Monte Carlo method to game-tree search and coined the name Monte Carlo tree search,
- L. Kocsis and Cs. Szepesvári developed the UCT (Upper Confidence bounds applied to Trees) algorithm,[17] and S. Gelly et al. implemented UCT in their program MoGo.
- In 2008, MoGo achieved dan (master) level in 9×9 Go, and the Fuego program began to win against strong amateur players in 9×9 Go.



https://www.geeksforgeeks.org/ml-monte-carlo-tree-search-mcts/

MCTS steps

- Each round of MCTS consists of four steps:
  - *Selection*
  - *Expansion*
  - *Simulation*
  - *Backpropagation*

Step 1: Selection

- Start from root *R* and select successive child nodes until a leaf node *L* is reached.
- The root is the current game state and a leaf is any node that has a potential child from which no simulation (playout) has yet been initiated.

- We will talk about the strategy for selection soon!

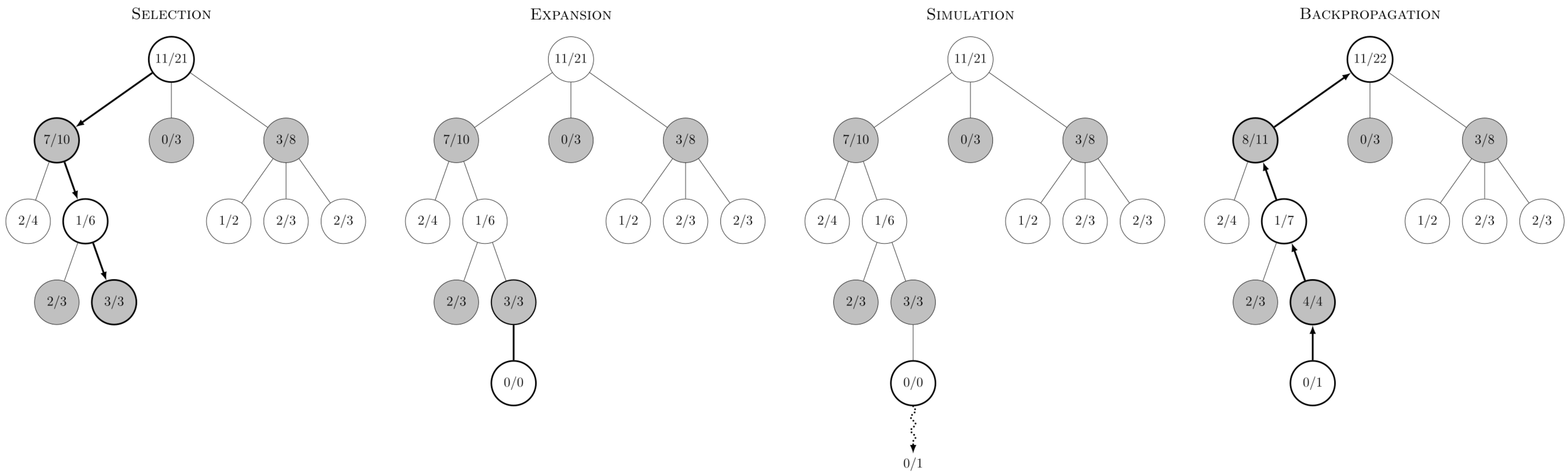
Step 2-3: *Expansion and Simulation*

- *Expansion*:
  - Unless *L* ends the game decisively (e.g. win/loss/draw) for either player, create one (or more) child nodes and choose node *C* from one of them. Child nodes are any valid moves from the game position defined by *L*.
- *Simulation*:
  - Complete one random playout from node *C*.
  - This step is sometimes also called playout or rollout. A playout may be as simple as choosing uniform random moves until the game is decided (for example in chess, the game is won, lost, or drawn).
  - It’s possible to perform multiple simulations with reward accumulated for each simulation

Step 4: Backpropagation

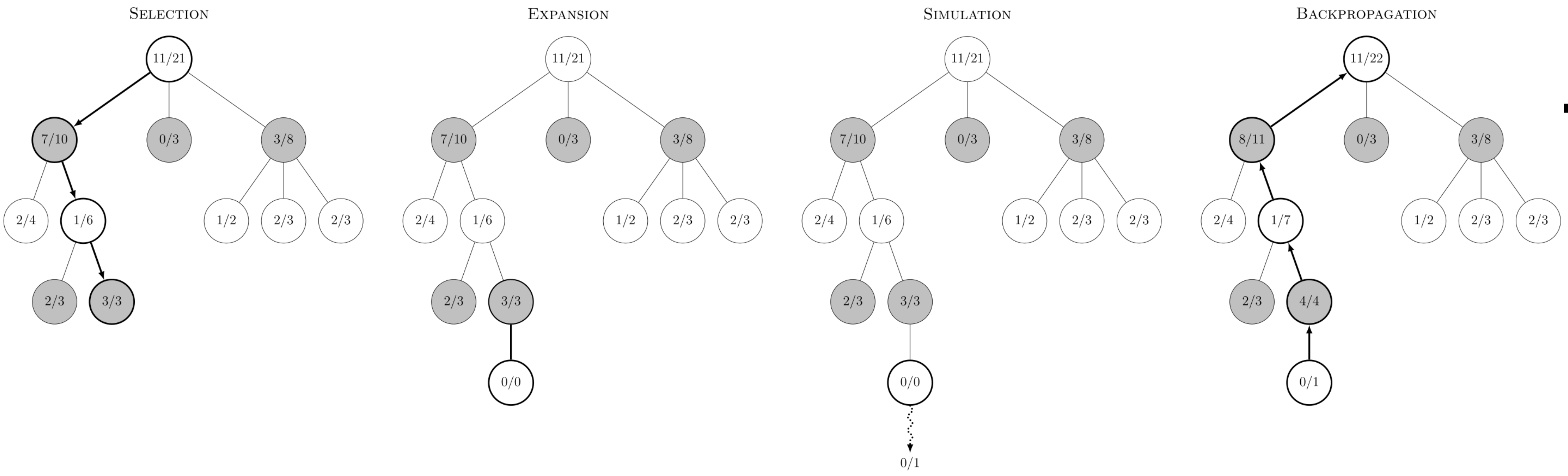
- Use the result of the playout to update information in the nodes on the path from *C* to *R*.

Example: 2-player game



This graph shows the steps involved in one decision, with each node showing the ratio of wins to total playouts from that point in the game tree for the player that the node represents.

In the Selection diagram, black is about to move. The root node shows there are 11 wins out of 21 playouts for white from this position so far. It complements the total of 10/21 black wins shown along the three black nodes under it, each of which represents a possible black move.

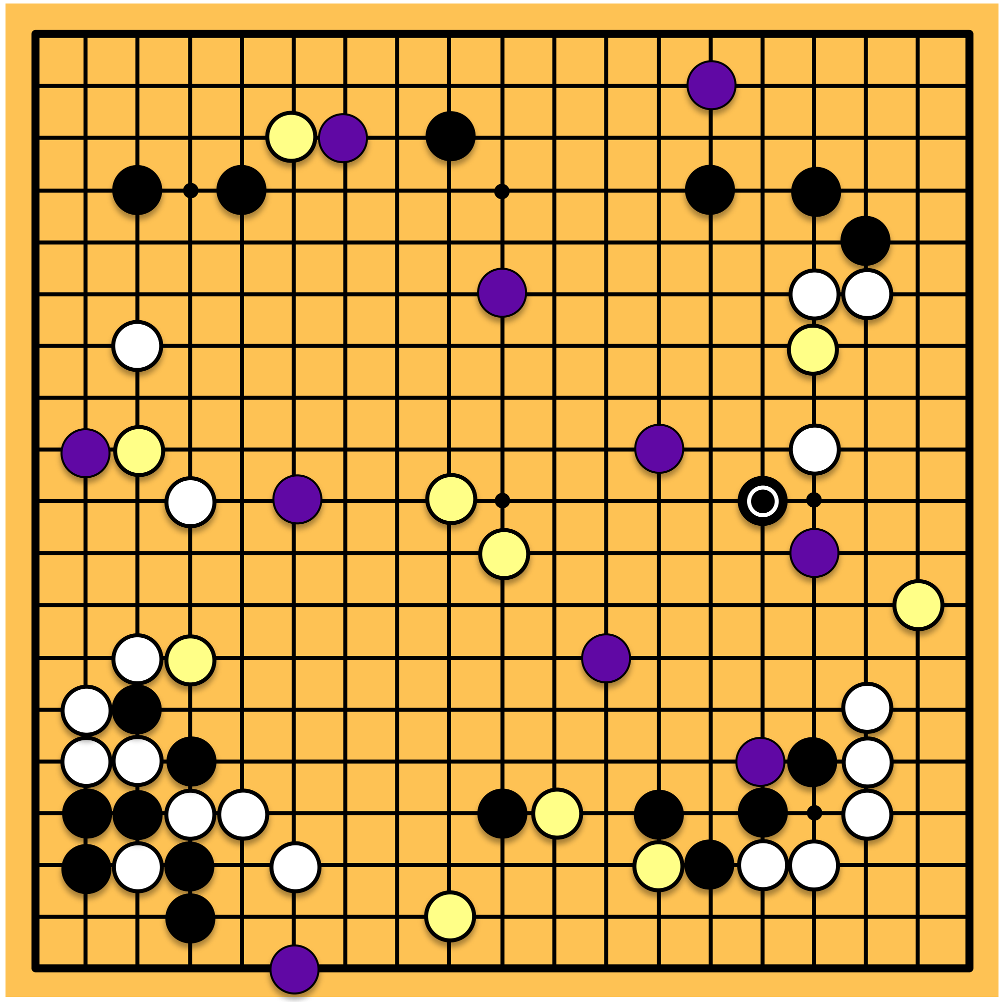


If white loses the simulation:
all nodes along the selection incremented their simulation count (the denominator), but among them only the black nodes were credited with wins (the numerator).
If white wins,the simulation:
all nodes along the selection would still increment their simulation count, but among them only the white nodes would be credited with wins.
This ensures that during selection, each player's choices expand towards the most promising moves for that player, which mirrors the goal of each player to maximize the value of their move.
Rounds of search are repeated as long as the time allotted to a move remains. Then the move with the most simulations made (i.e. the highest denominator) is chosen as the final answer.

Rollouts

- For each rollout:
  - Repeat until terminal:
    - Play a move according to a fixed, fast rollout policy
  - Record the result
- Fraction of wins correlates with the true value of the position!
- Having a "better" rollout policy helps

"Move 37"

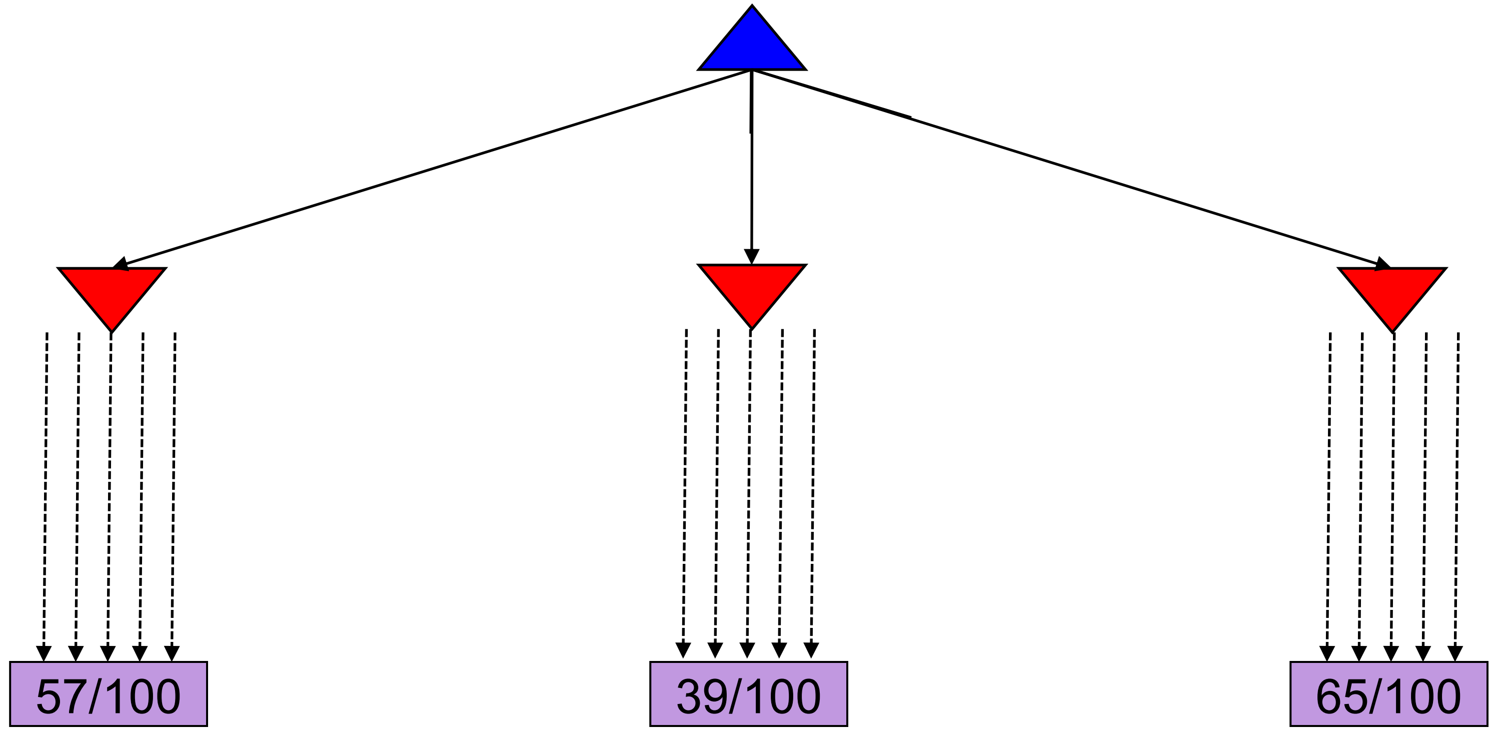


Some Other Considerations

- Considerations:
  - The trick is to decide what moves to make in the course of the rollout.
  - There is no aggressive pruning; all moves are possible.
  - The UCT (upper confidence bounds on trees) method works by making random moves in the first few iterations, and over time guiding the sampling process to prefer moves that have led to wins in previous samples.
  - Some tricks are added, including knowledge-based rules that suggest particular moves whenever a given pattern is detected and limited local search to decide tactical questions.

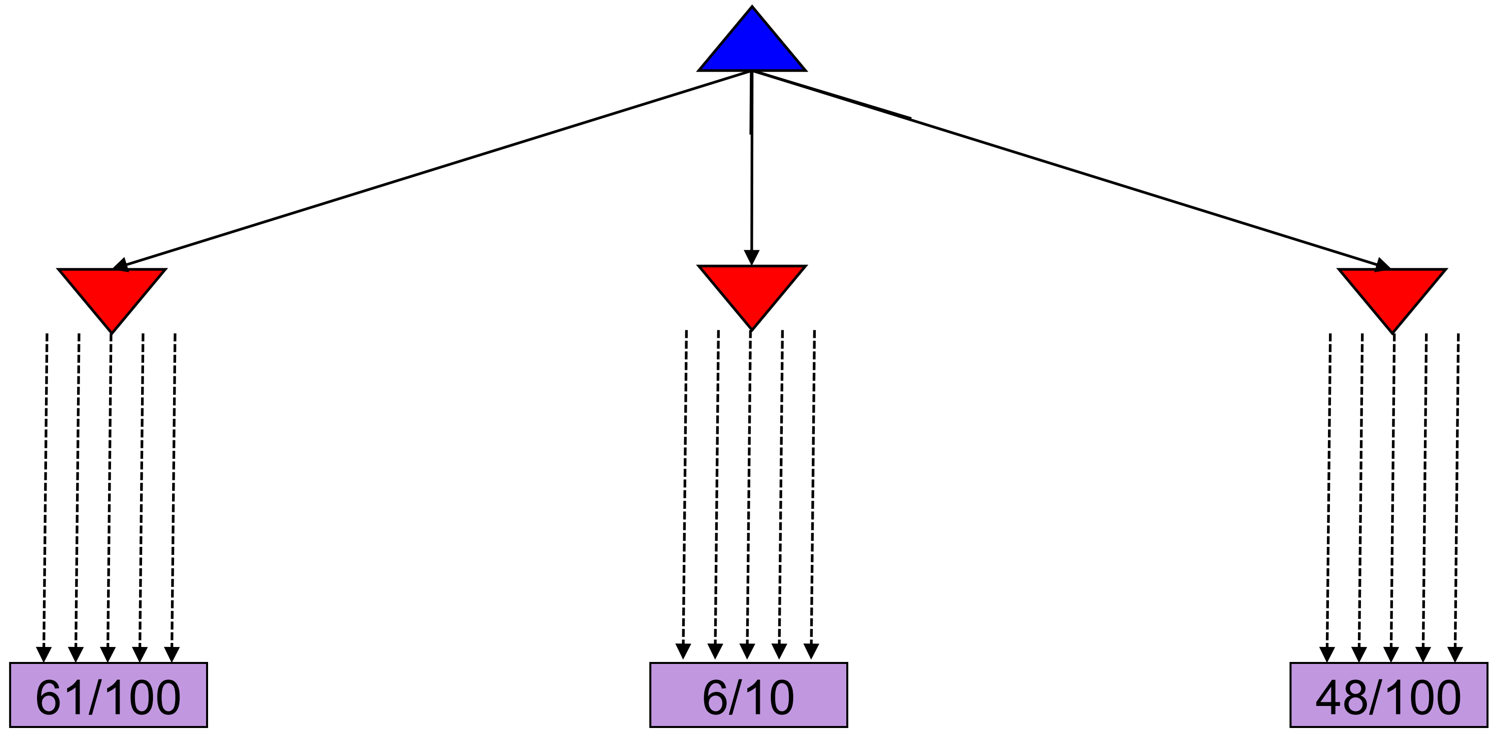
MCTS Version 0

- Do *N* rollouts from each child of the root, record fraction of wins
- Pick the move that gives the best outcome by this metric



MCTS Version 1.0

- Allocate rollouts to more promising nodes
- Allocate rollouts to more uncertain nodes



UCB heuristics

- UCB1 formula combines "promising" and "uncertain":

$$UCB = \frac{W_i}{N_i} + c\sqrt{\frac{\ln N_p}{N_i}},$$

where:

- W_i = Total reward or wins of child node i
- N_i = Number of times node i has been visited
- N_p = Number of times the parent node has been visited
- c = Exploration parameter (higher c favors exploration)

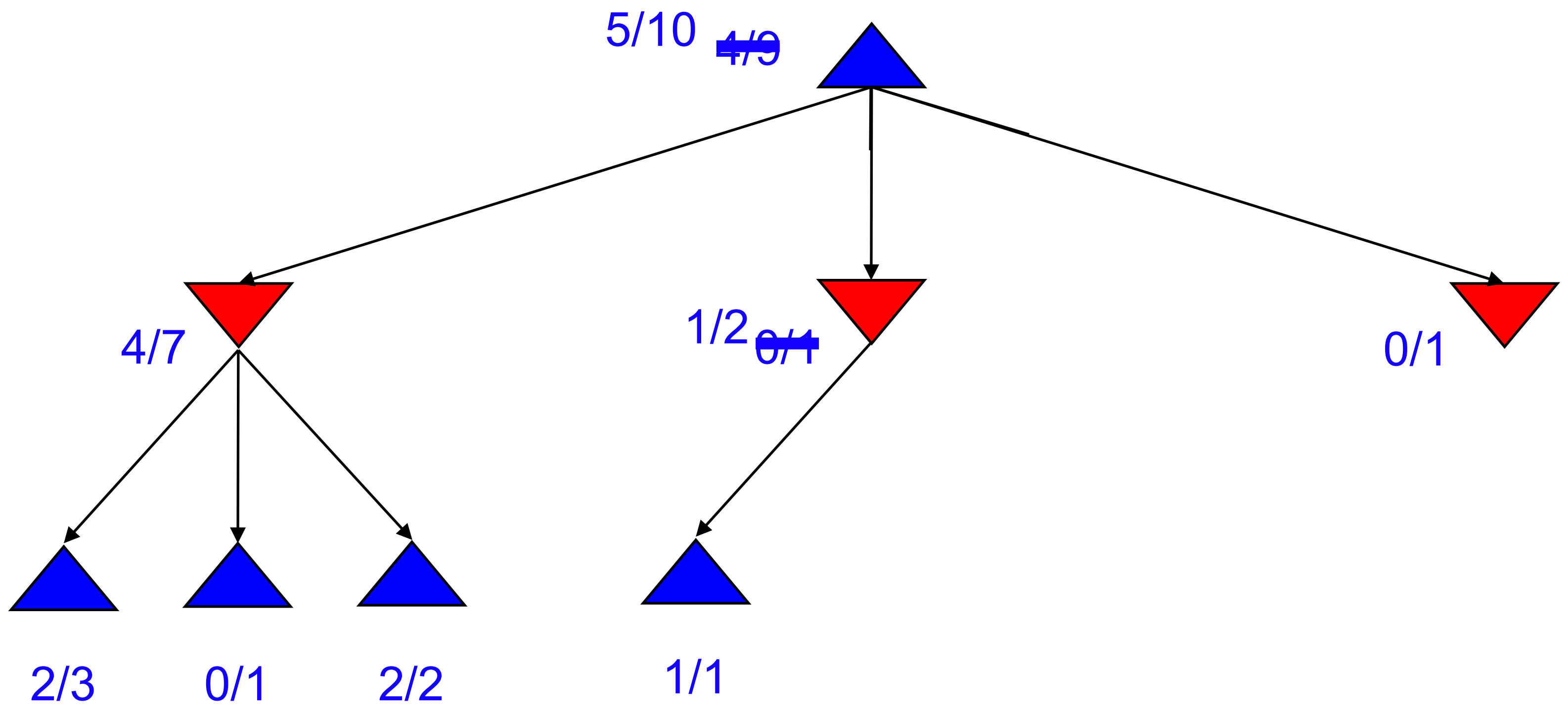
- Exploitation vs Exploration:
  - The first component corresponds to exploitation
    - high for moves with high average win ratio.
  - The second component corresponds to exploration
    - high for moves with few simulations.

MCTS v2 : UCT (Upper Confidence bounds applied to Trees)

- Repeat until out of time:
  - Given the current search tree, recursively apply UCB to choose a path down to a leaf (not fully expanded) node *n*
  - Add a new child *c* to *n* and run a rollout from *c*
  - Update the win counts from *c* back up to the root
- Choose the action leading to the child with highest *N*

Anytime we talk about out MCTS, you can safely assume we are talking about UCT (and not the pure MCTS)

UCT Example



YouTube

Worked out Example:

- https://youtu.be/UXW2yZndl7U

Based on the Book **Reinforcment Leaning: An introduction** by Richard S. Sutton and Andrew G. Barto

CLARIFICATIONS

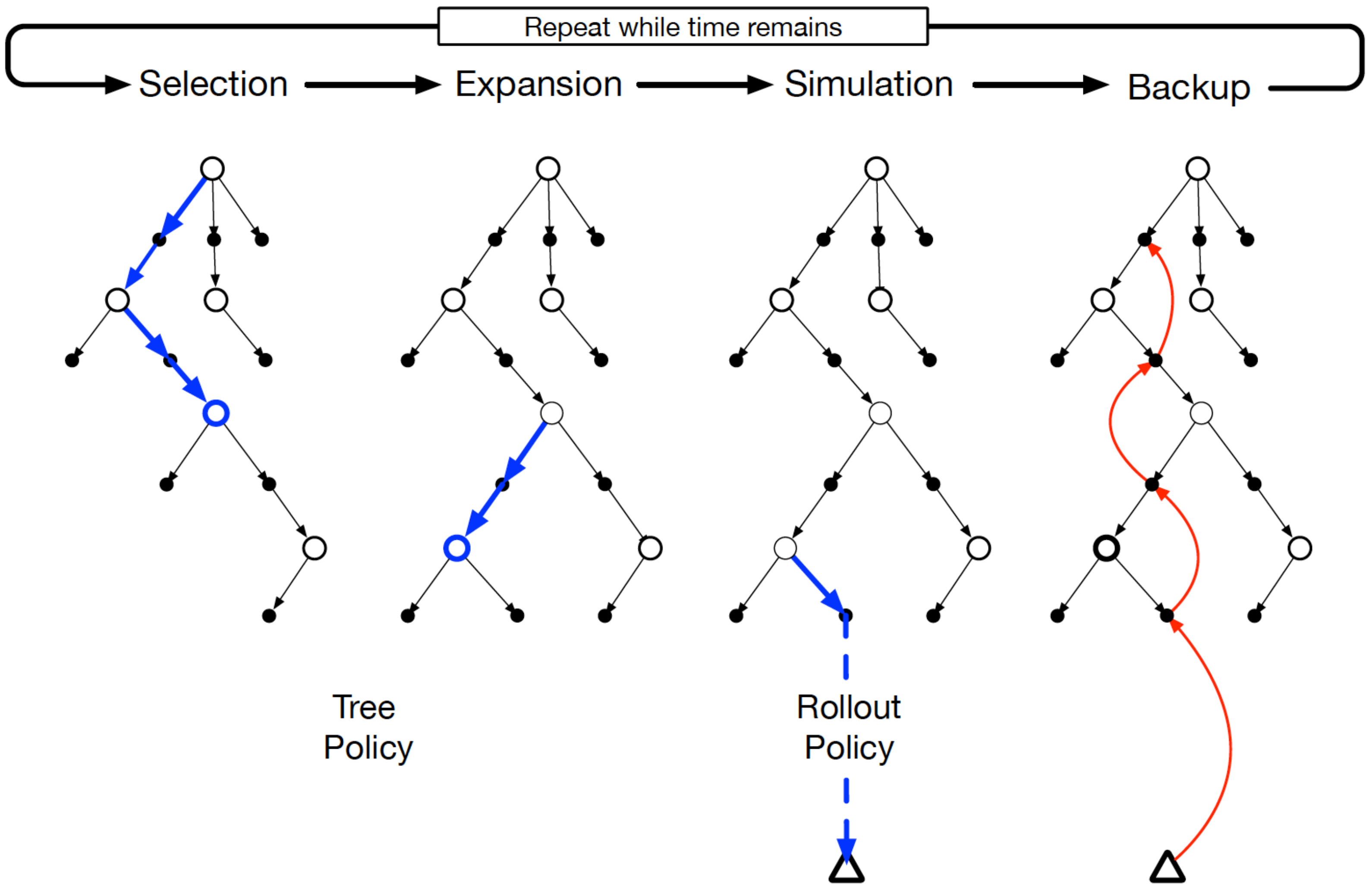


Figure 8.10: Monte Carlo Tree Search. When the environment changes to a new state, MCTS executes as many iterations as possible before an action needs to be selected, incrementally building a tree whose root node represents the current state. Each iteration consists of the four operations **Selection**, **Expansion** (though possibly skipped on some iterations), **Simulation**, and **Backup**, as explained in the text and illustrated by the bold arrows in the trees. Adapted from Chaslot, Bakkes, Szita, and Spronck (2008).

Steps in MCTS

Each iteration of a basic version of MCTS consists of the following four steps as illustrated:

1. Selection: Starting at the root node, a tree policy based on the action values attached to the edges of the tree traverses the tree to select a leaf node.
2. Expansion: On some iterations (depending on details of the application), the tree is expanded from the selected leaf node by adding one or more child nodes reached from the selected node via unexplored actions.
3. Simulation: From the selected node, or from one of its newly-added child nodes (if any), simulation of a complete episode is run with actions selected by the rollout policy. The result is a Monte Carlo trial with actions selected first by the tree policy and beyond the tree by the rollout policy.
4. Backup: The return generated by the simulated episode is backed up to update, or to initialize, the action values attached to the edges of the tree traversed by the tree policy in this iteration of MCTS. No values are saved for the states and actions visited by the rollout policy beyond the tree. Figure 8.10 illustrates this by showing a backup from the terminal state of the simulated trajectory directly to the state–action node in the tree where the rollout policy began (though in general, the entire return over the simulated trajectory is backed up to this state–action node).

Clarification: Selection

- Starting at the root node, a tree policy based on the action values attached to the edges of the tree traverses the tree to select a leaf node

- Important: Tree policy could select actions using an ϵ-greedy or UCB selection rule. There is no hard and fast rule for tree policy selection but we will use UCB1 for most of the cases

- Also note that in the earlier example, some actions are not tried

Clarificaiton: Expansion

- It is possible that Expansion could be skipped on some iterations

Classification: Number of rollouts

- Usually only one per iteration:
  - From the selected node, simulation of a complete episode is run with actions selected by the rollout policy
- Based on the application, it’s possible to have multiple rollouts from the selected node

Clarification: When do we stop?

- MCTS continues executing these four steps, starting each time at the tree’s root node, until
  - no more time is left, or
  - some other computational resource is exhausted

Clarification: Final Selection Criteria

- Finally, an action from the root node (which still represents the current state of the environment) is selected according to some mechanism that depends on the accumulated statistics in the tree; for example,
  - it may be an action having the largest action value of all the actions available from the root state, or
  - the action with the largest visit count to avoid selecting outliers.
  - The latter is the action MCTS actually selects.

Clarification: What happens after the move is played

- After the environment transitions to a new state, MCTS is run again,
  - sometimes starting with a tree of a single root node representing the new state, but
  - often starting with a tree containing any descendants of this node left over from the tree constructed by the previous execution of MCTS; all the remaining nodes are discarded, along with the action values associated with them.

Clarification: Why is there no min or max?????

- "Value" of a node, w_i/n_i, is a weighted ***sum*** of child values!
- Idea: as $n \rightarrow \infty$, the vast majority of rollouts are concentrated in the best child(ren), so weighted average $\rightarrow$ max/min
- Theorem: as $n \rightarrow \infty$ UCT selects the minimax move
  - (but *n* never approaches infinity!)