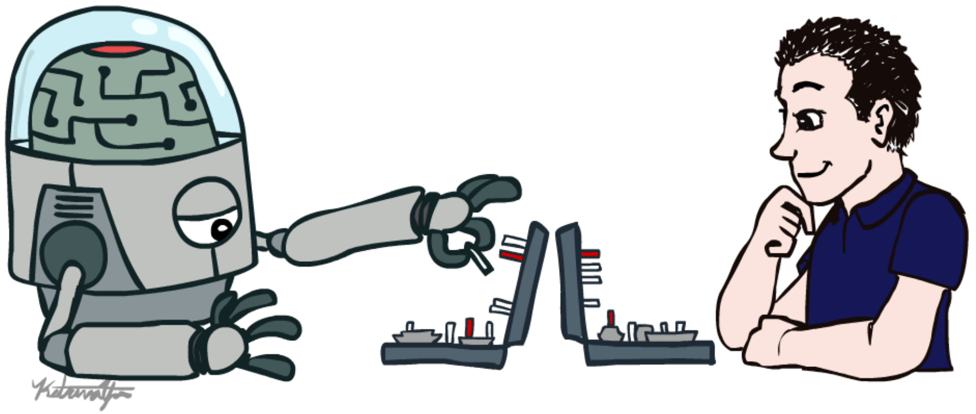
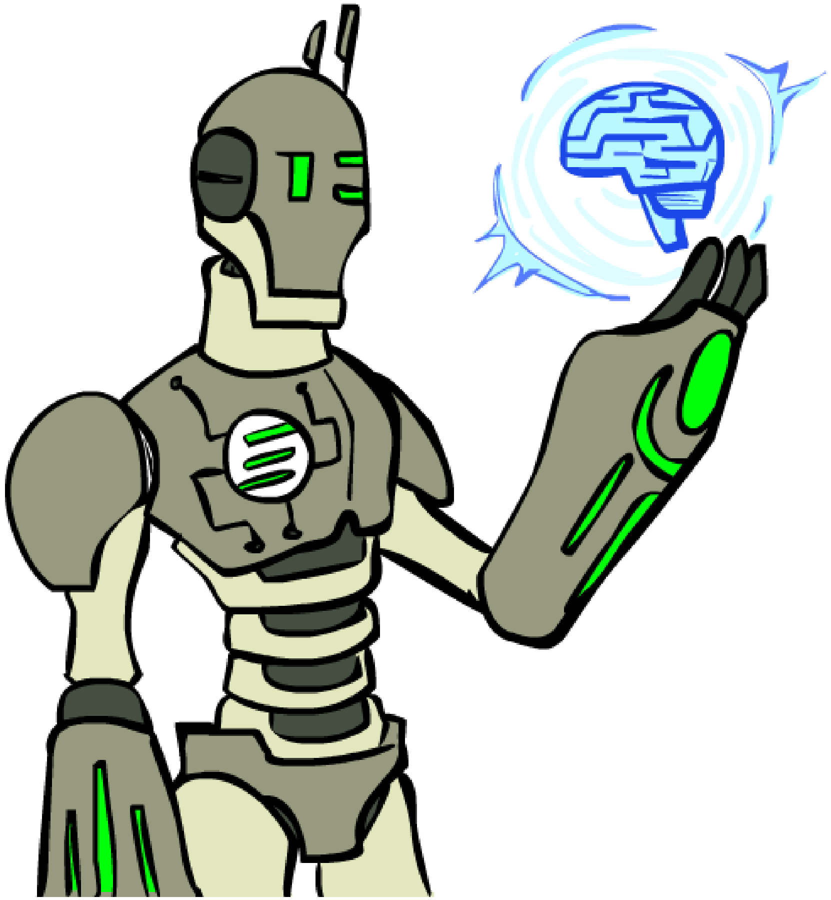
Lecture 05



Ashis Kumar Chanda
chanda@rowan.edu

Rowan University

Today

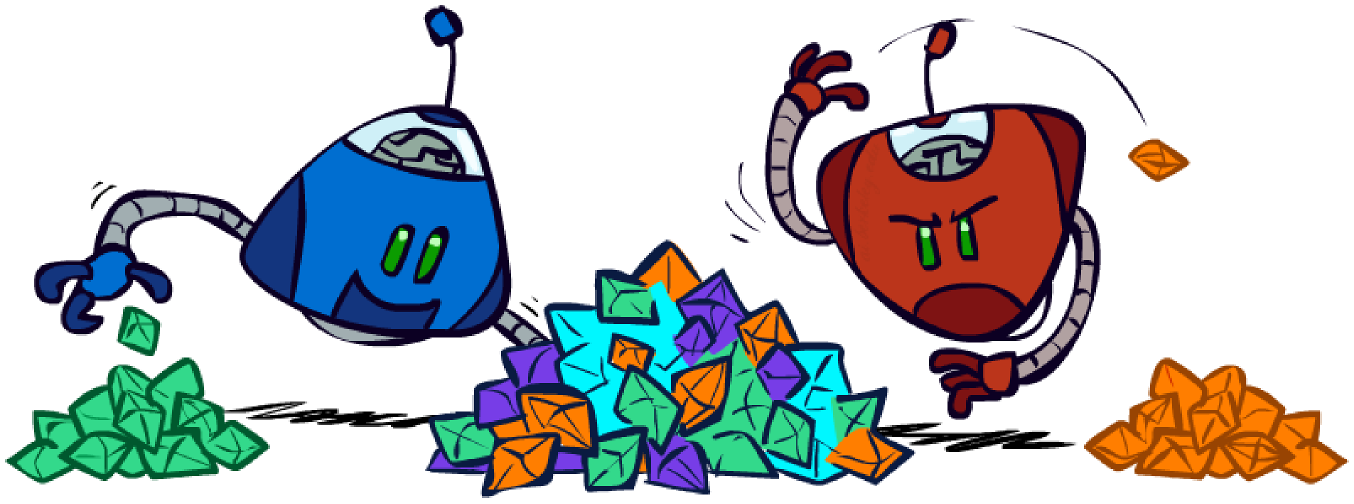


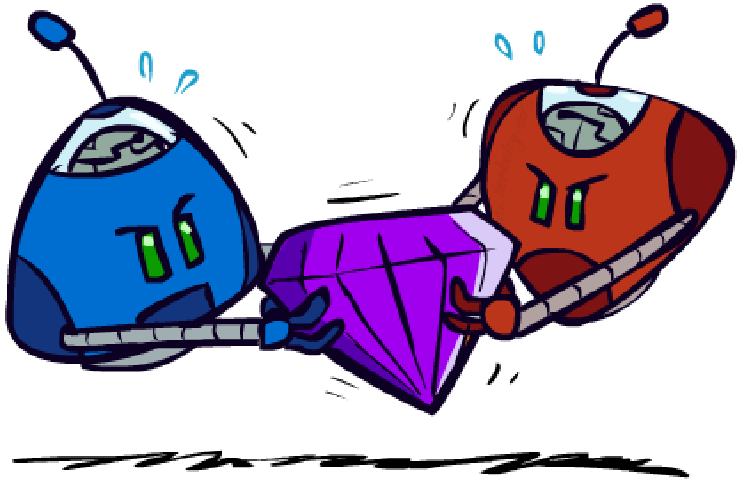
- Minimax algorithm
- Alpha-beta pruning

Introduction

- In previous classes, we learned different searching methods with and without knowledge.
- We had a specific cost for each node in a search tree and goal was fixed.
- Let's consider what happens when we have opponent.
- And the opponent is also rational.

Types of Games





- General Games
  - Agents have independent utilities (values on outcomes)
  - Cooperation, indifference, competition, and more are all possible
- Zero-Sum Games
  - Agents have opposite utilities (values on outcomes)
  - Lets us think of a single value that one maximizes and the other minimizes

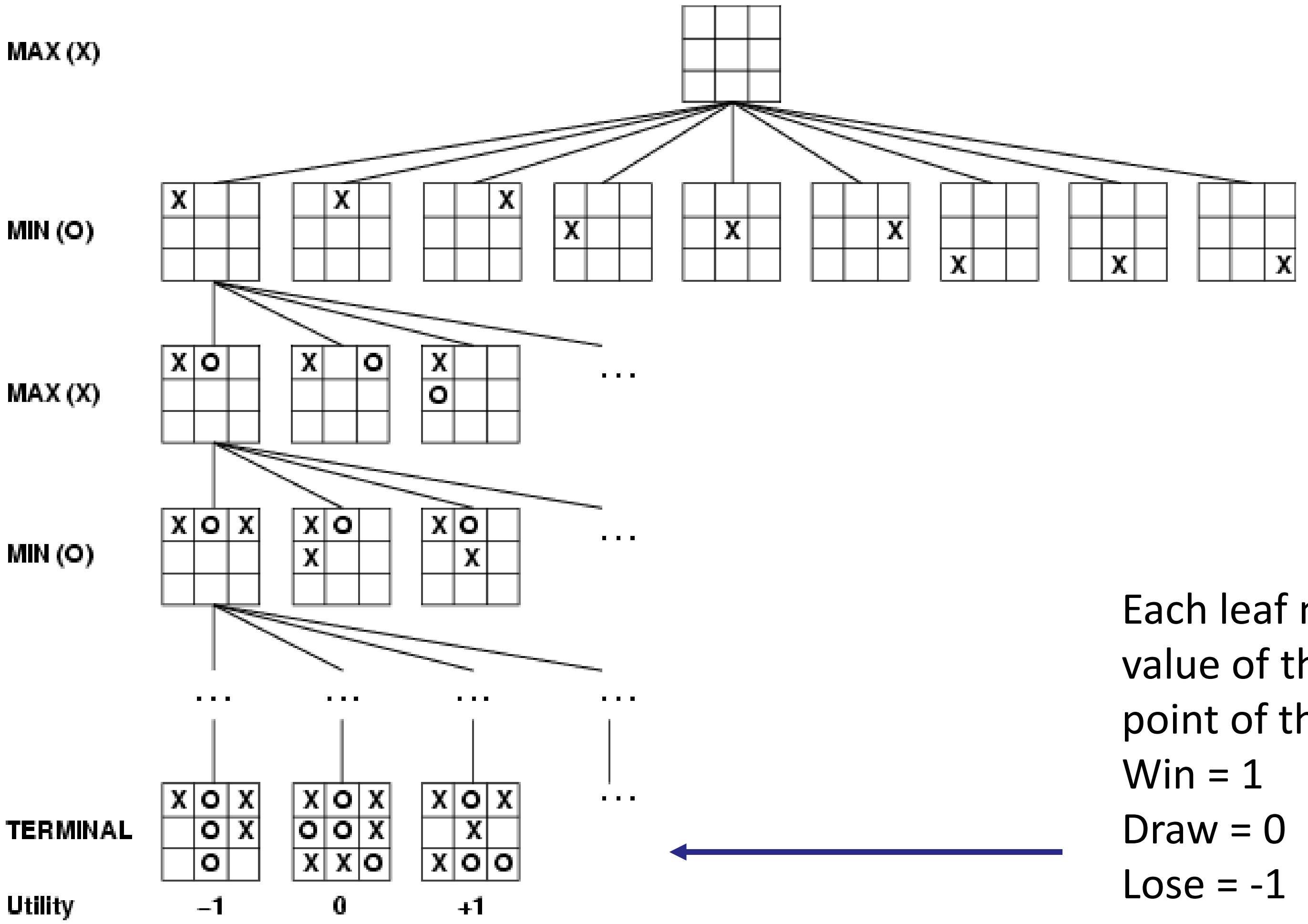
Adversarial Search

- In a multi-agent environment, we can take two types of actions:
  - Maximizing utility (without having to predict the action of any individual agent)
  - Explicitly model the adversarial agents with the techniques of adversarial game-tree search.
- "Unpredictable" opponent → specifying a move for every possible opponent's reply.

Games as Adversarial Search

- Let's define a game as a search tree, where
  - Move = Action
  - Board position = State or Node
  - Terminal states = Where the game has ended
  - Utility function = Defines the final numeric value to player p when the game ends in terminal state
- Special feature:
  - Two players, adversarial

Two Player Game Example: Tic-tac-toe



Each leaf node indicates the **utility** value of the terminal state from the point of the view of MAX.

Win = 1

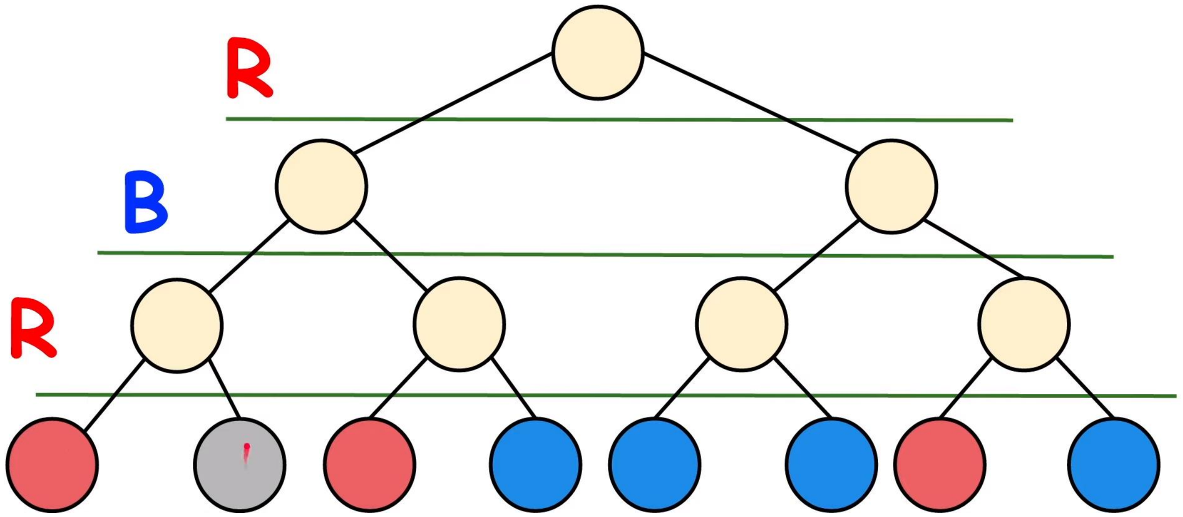
Draw = 0

Lose = -1

A game tree for the game of tic-tac-toe. The top node is the initial state, and MAX moves first, placing an X in an empty square.

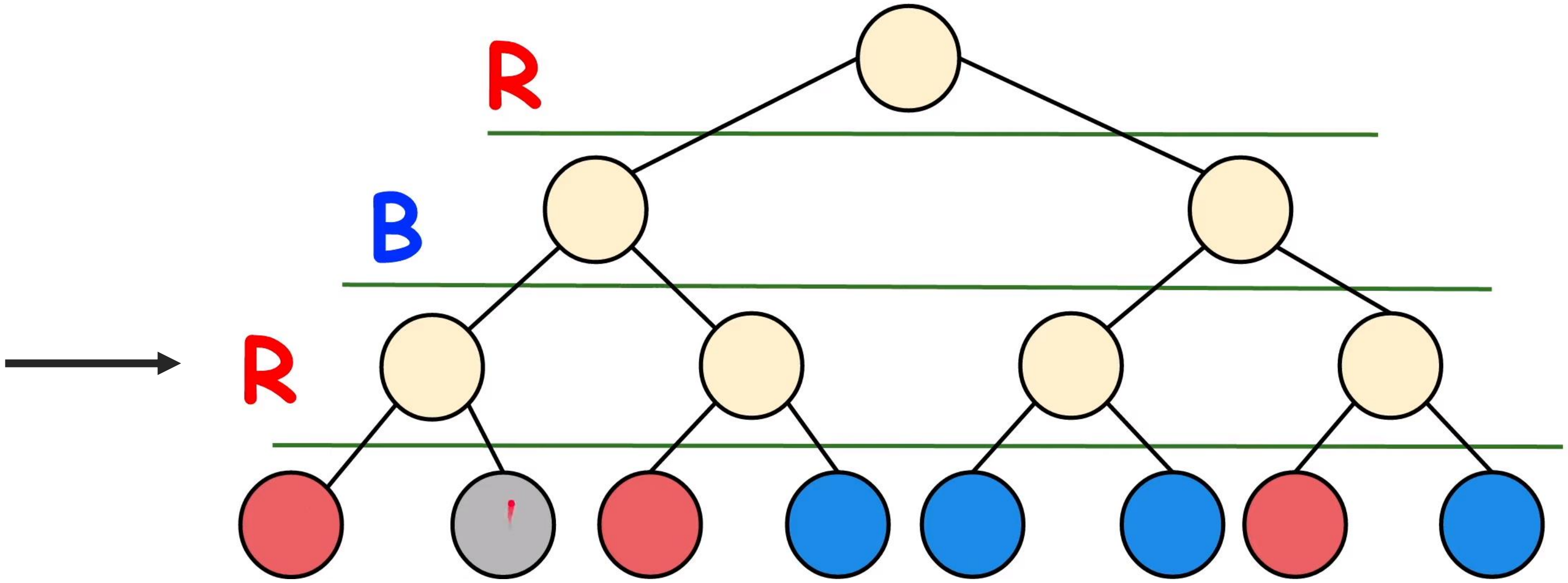
Two Player Game Example: Chess

- Suppose, we have two opponents in a chess game, **Red** and **Blue**.
- **Red** moves first.
- We can think about the game moves like the following tree where the leaf nodes are terminal states.

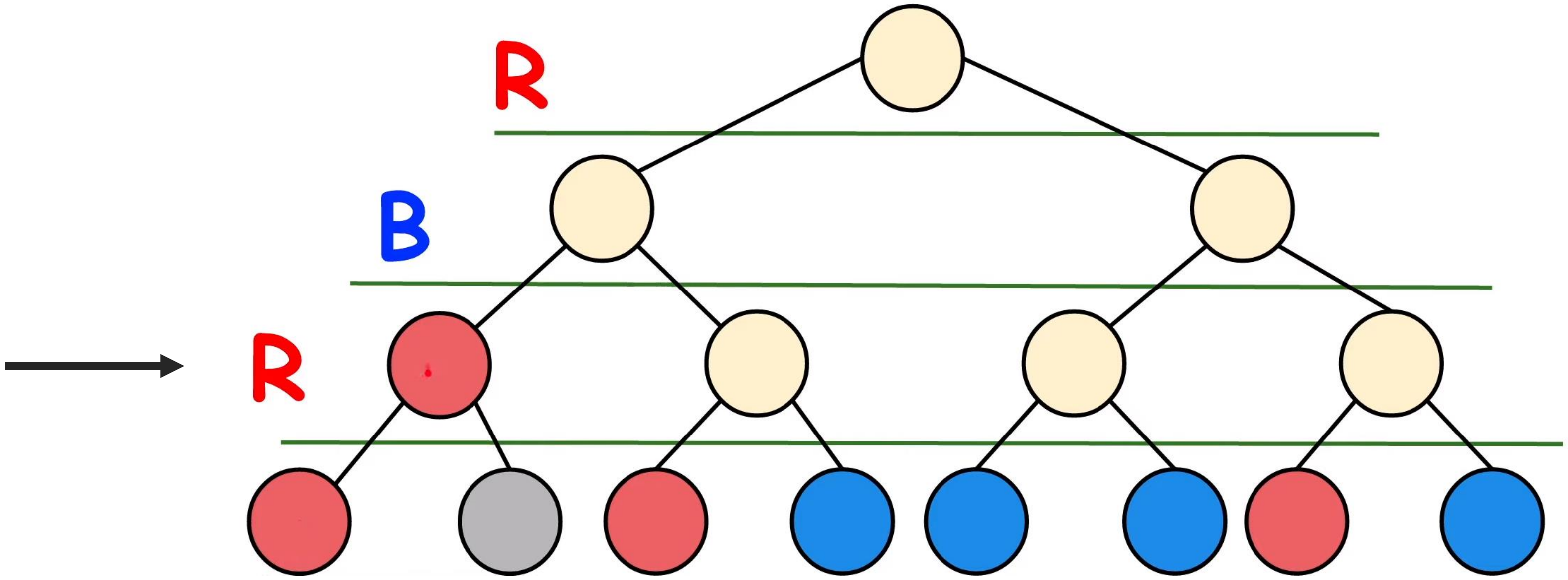


- Red nodes are the winning state for Red.
- Blue nodes are the winning states for blue.
- Gray nodes are draw states.
- Red will try to reach at final red states, and so, we need to think from the **bottom**.

Two Player Game Example: Chess

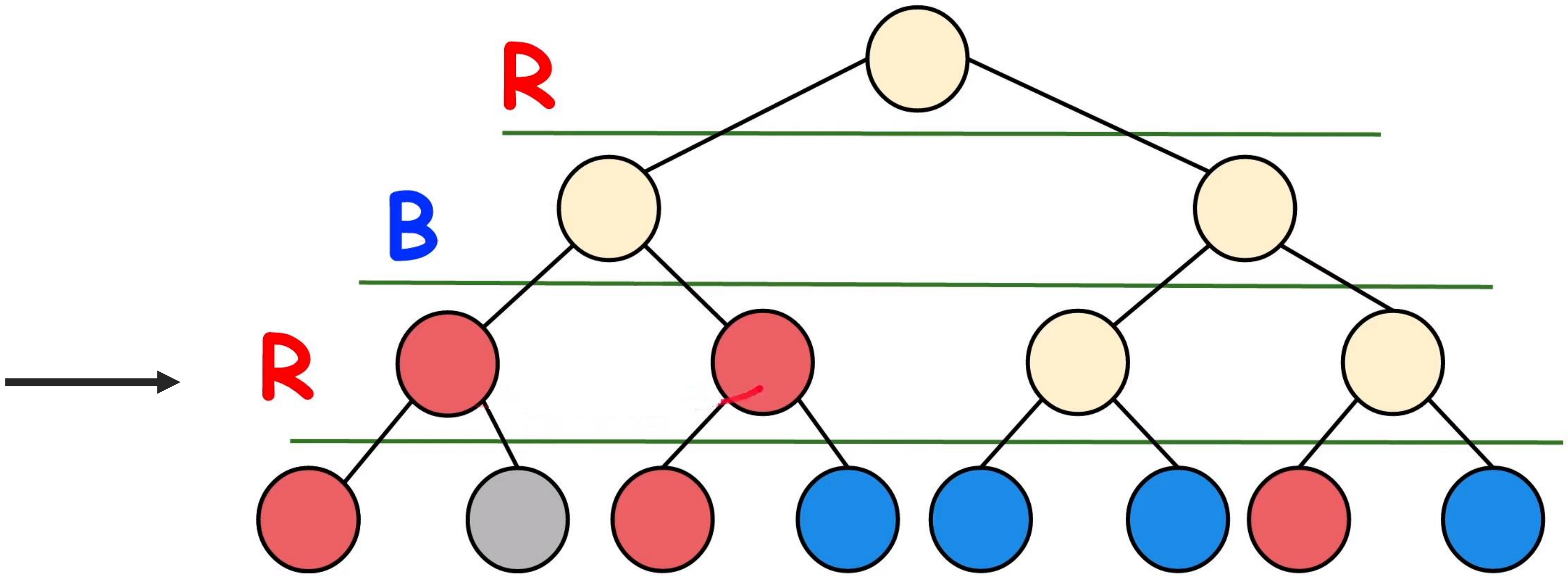


Two Player Game Example: Chess



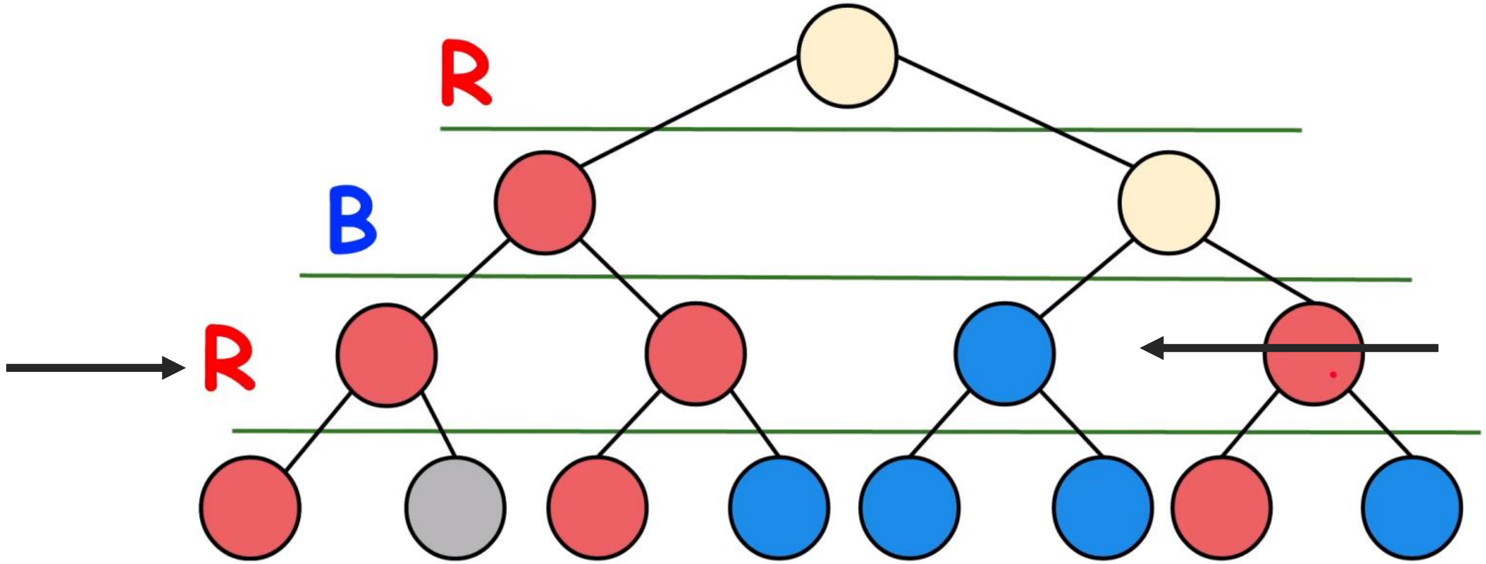
Red will try to reach at final red states

Two Player Game Example: Chess



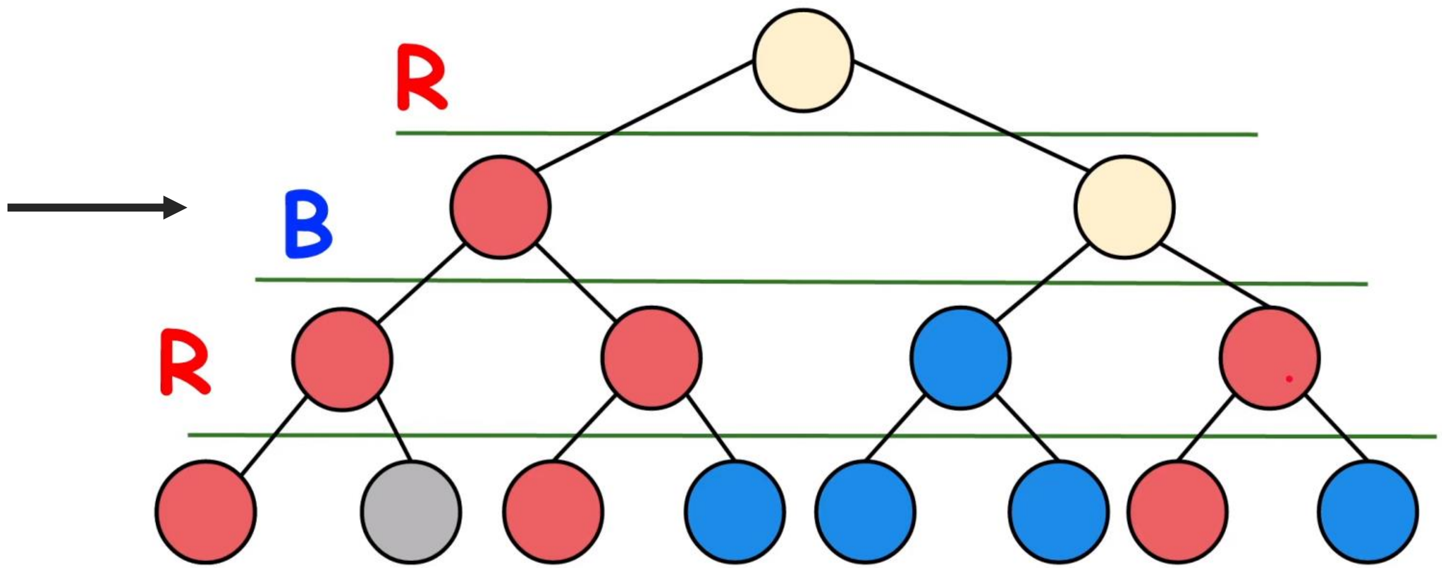
Red will try to reach at final red states

Two Player Game Example: Chess



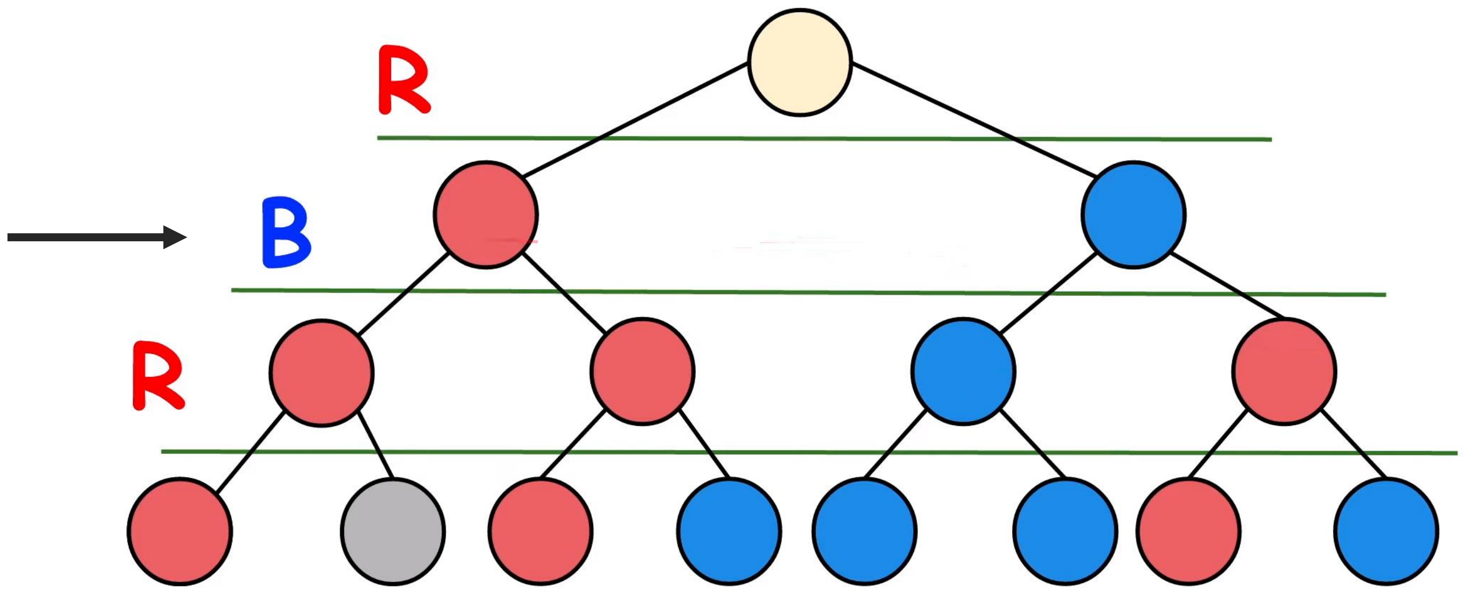
Here, Red has no other options, it will pick blue.

Two Player Game Example: Chess



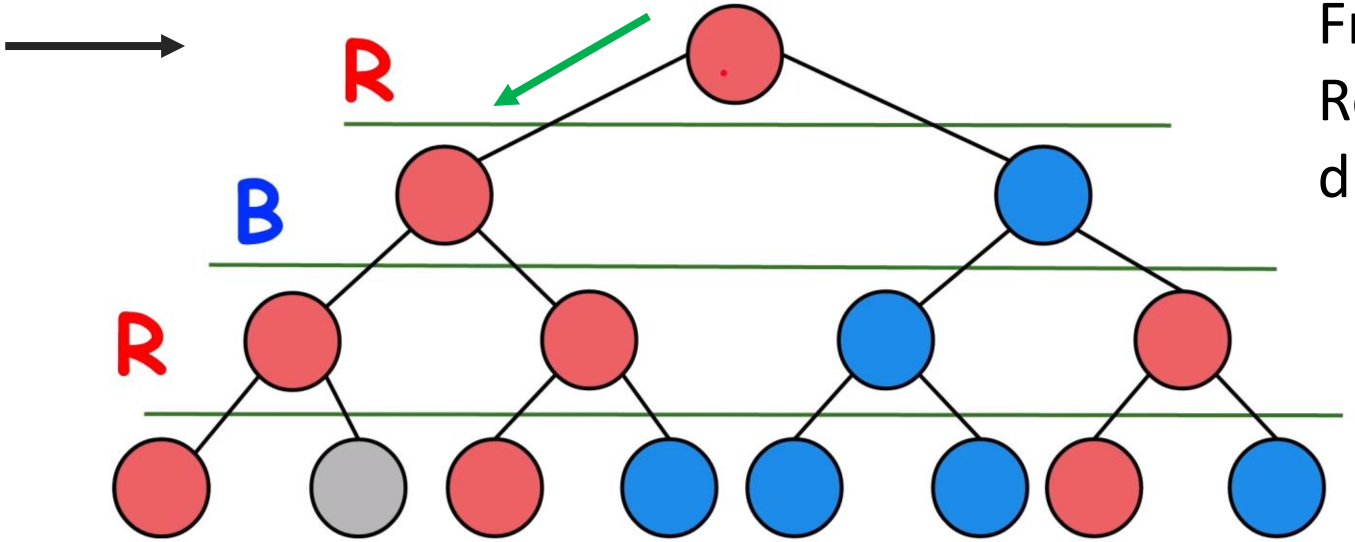
Blue will try to reach at final blue states

Two Player Game Example: Chess



Blue will try to reach at final blue states

Two Player Game Example: Chess

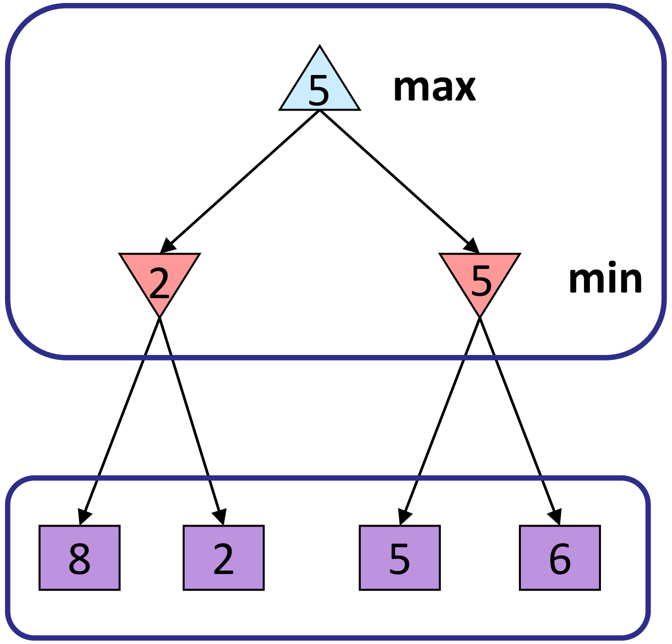


From the initial state, Red will select the left direction.

Adversarial Search (Minimax)

- Deterministic, zero-sum games:
  - Tic-tac-toe, chess, checkers
  - One player maximizes result
  - The other minimizes result
- Minimax search:
  - A state-space search tree
  - Players alternate turns
  - Compute each node's minimax value: the best achievable utility against a rational (optimal) adversary

Minimax values:
computed recursively



Terminal values:
part of the game

Minimax Procedure

- If at limit of search, compute static value
  - Relative to player
- If minimizing level, do minimax
  - Report minimum
- If maximizing level, do minimax
  - Report maximum

Two Player Games

- We have two players, Max and Min.
- Max always moves first.
- Min is the opponent.


Max Vs Min

We have

- An initial state.
- A set of actions.
- A terminal test (which tells us when the game is over).
- A utility function (evaluation function).

- The utility function is like the heuristic function we have seen in the past, except it evaluates a node in terms of how good it is for each player.

Minimax Example

- Idea: choose a move to a position with the highest minimax value = best achievable payoff against a rational opponent.

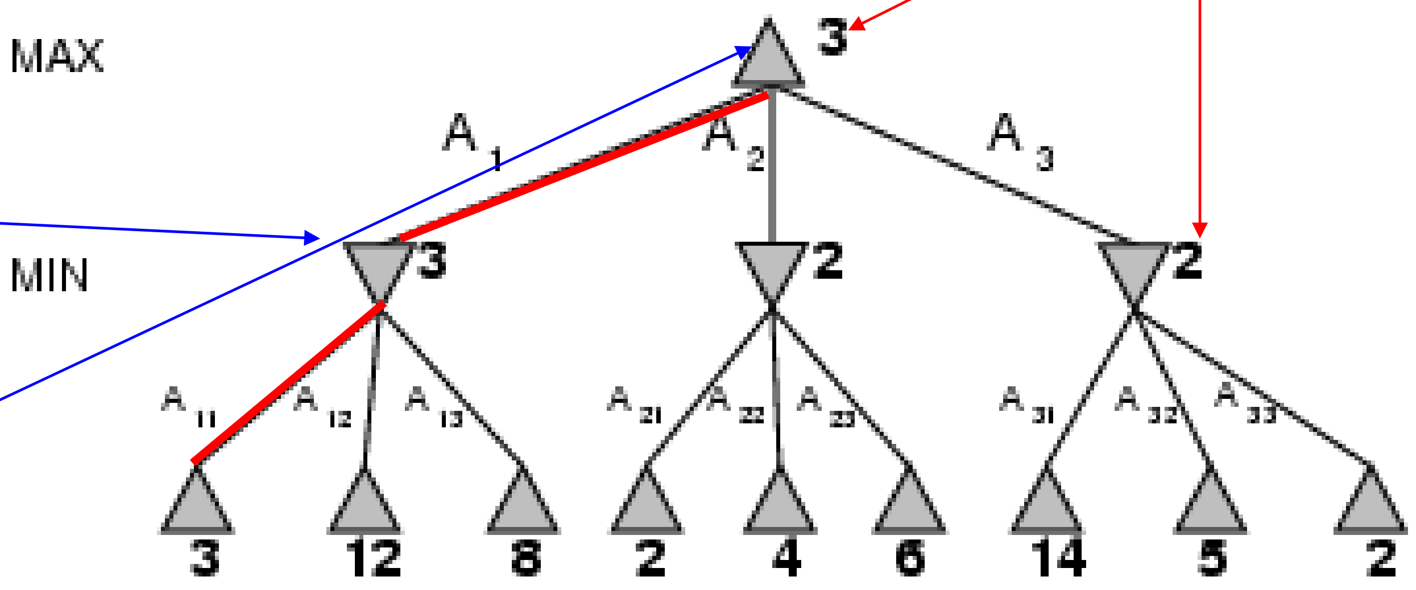
Example: deterministic 2-ply game:

Minimax value is computed bottom up:
-Leaf values are given.

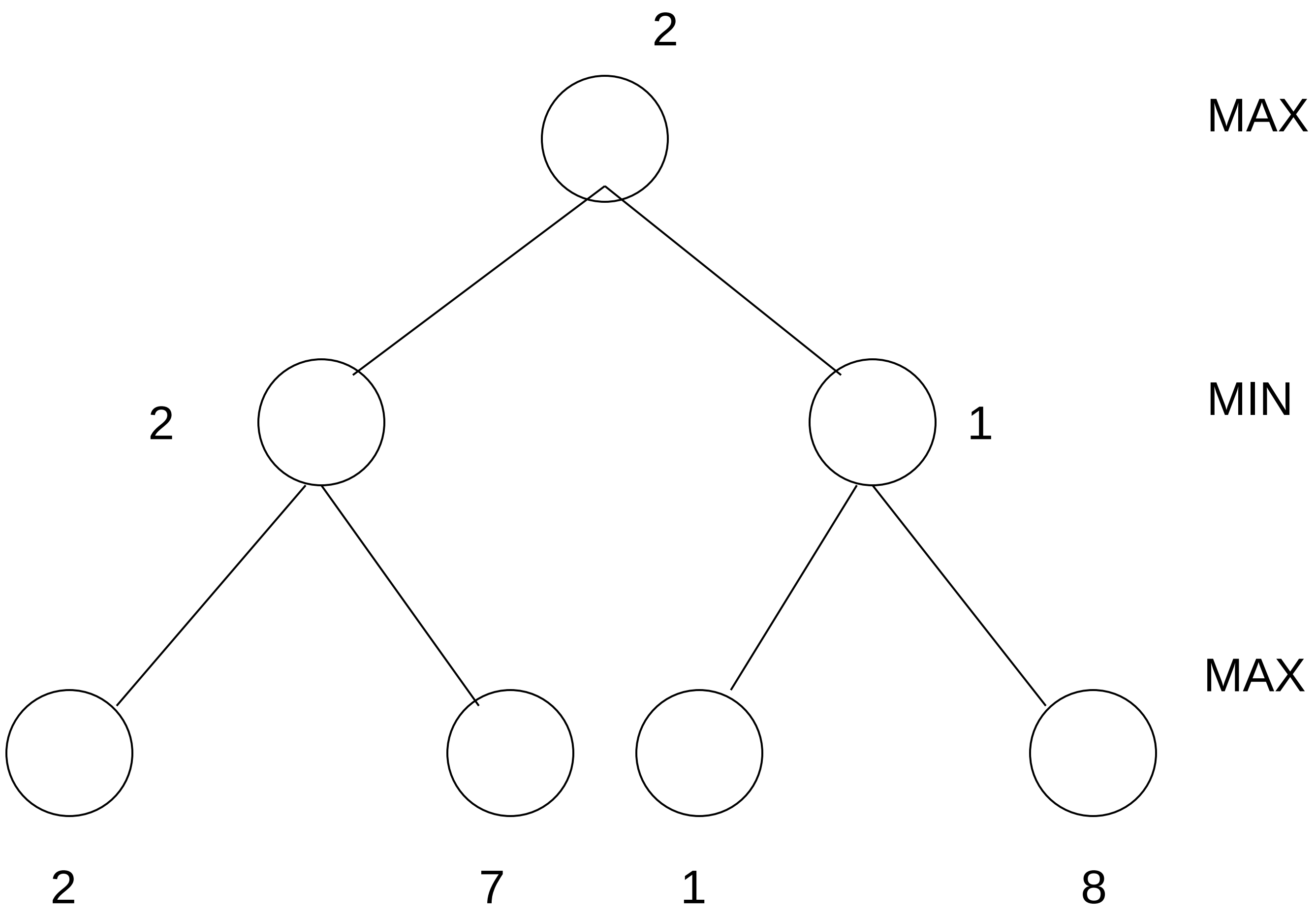
-3 is the best outcome
 for MIN in this branch.

-3 is the best outcome
 for MAX in this game.

-We explore this tree in
 depth-first manner.



Minimax Example



Minimax Implementation (Dispatch)

```
def value(state):
    if the state is a terminal state: return the state's utility
    if the next agent is MAX: return max-value(state)
    if the next agent is MIN: return min-value(state)
```

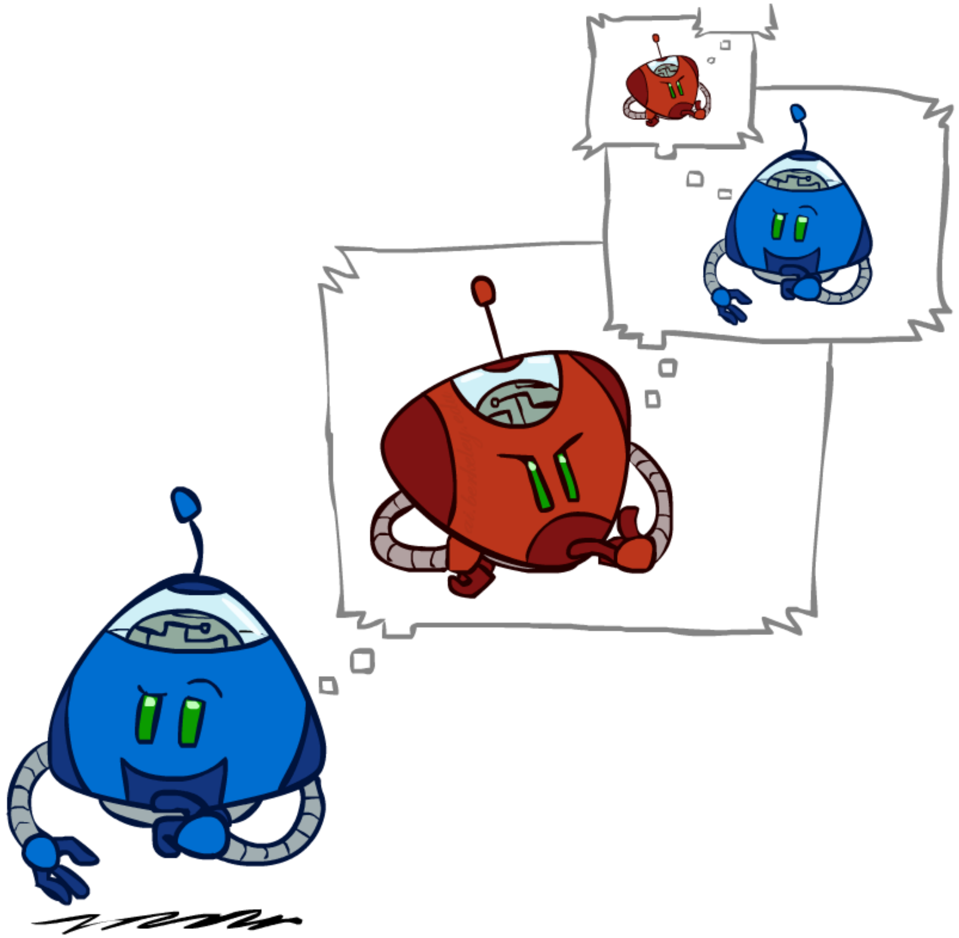
```
def max-value(state):
    initialize v = -∞
    for each successor of state:
        v = max(v, value(successor))
    return v
```

```
def min-value(state):
    initialize v = +∞
    for each successor of state:
        v = min(v, value(successor))
    return v
```

Minimax Efficiency

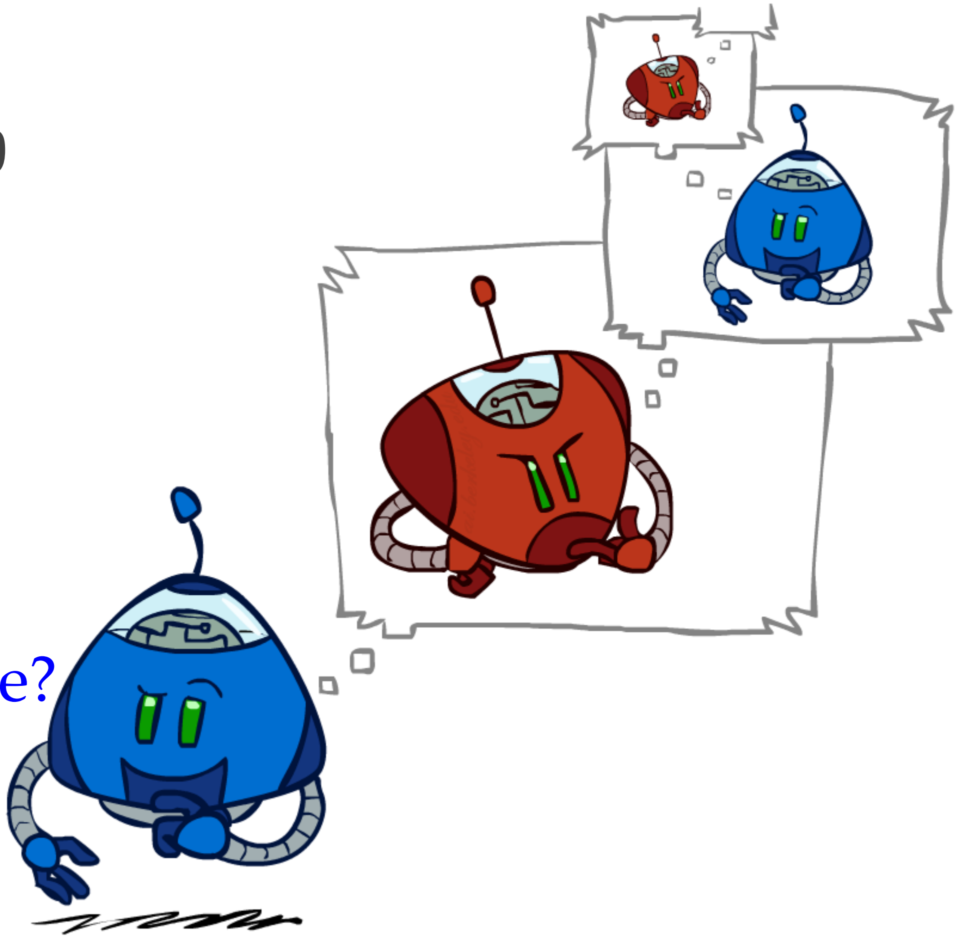
- How efficient is minimax?
  - Just like (exhaustive) DFS
  - Time: $O(b^m)$
  - Space: $O(bm)$

- The number of game states is exponential in the depth of the tree.

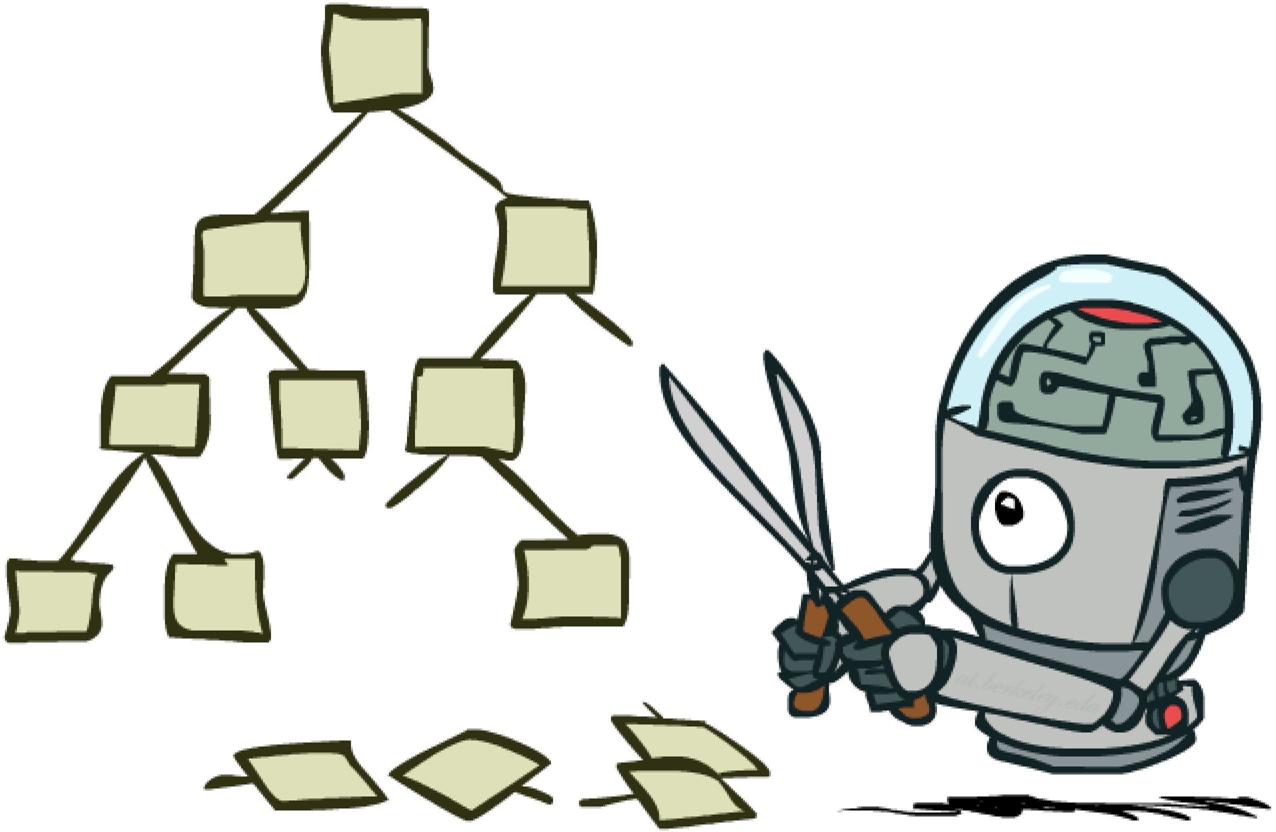


Minimax Efficiency

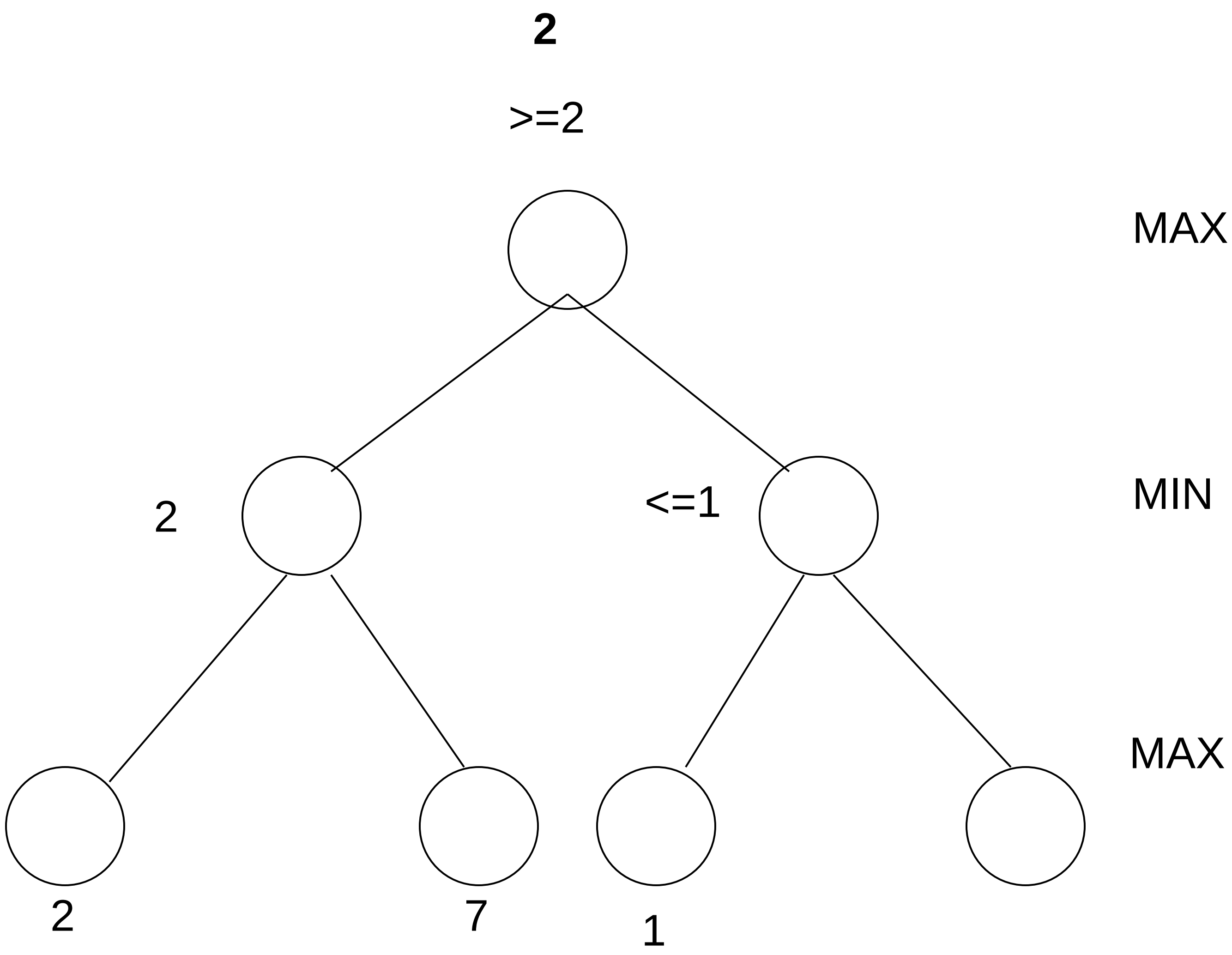
- Example:
  - For tic-tac-toe game, we have 9! = 362,880 terminal nodes.
- Example:
  - For chess, $b \approx 35$, $m \approx 100$
  - Exact solution is completely infeasible
  - But, do we need to explore the whole tree?



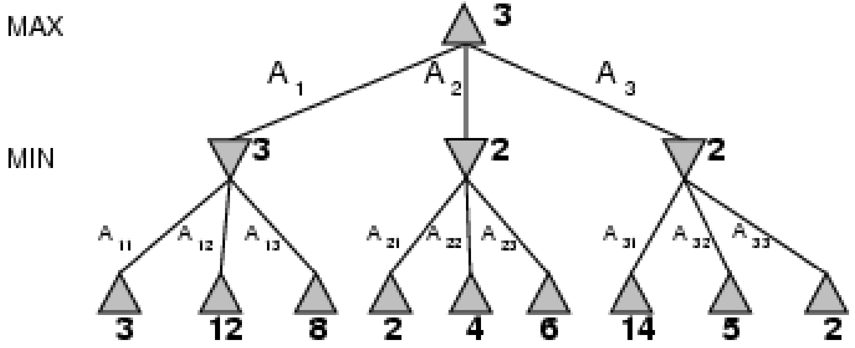
Game Tree Pruning

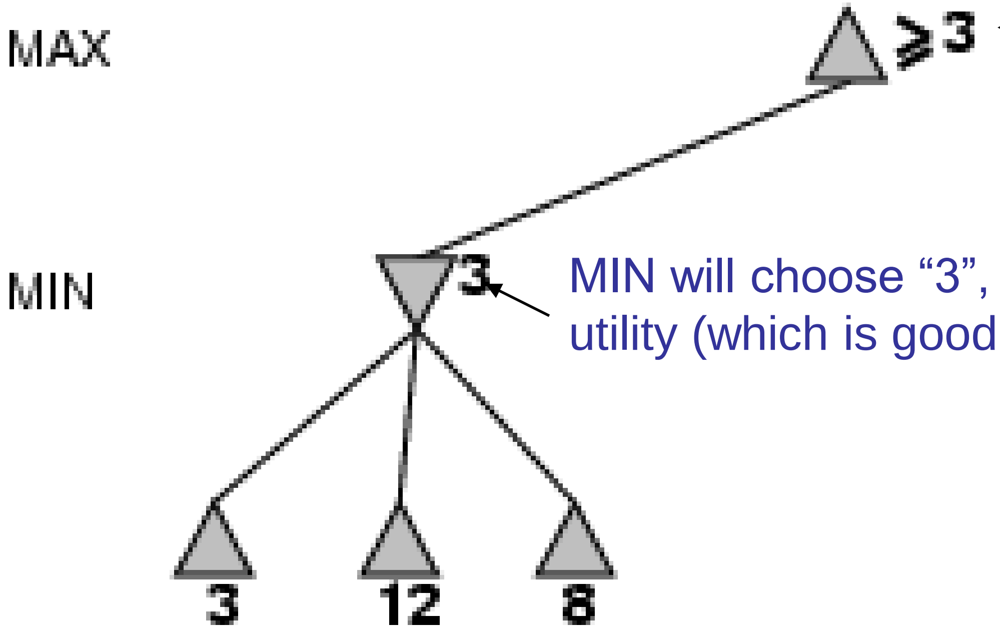


Simple α-β Example



α-β pruning example

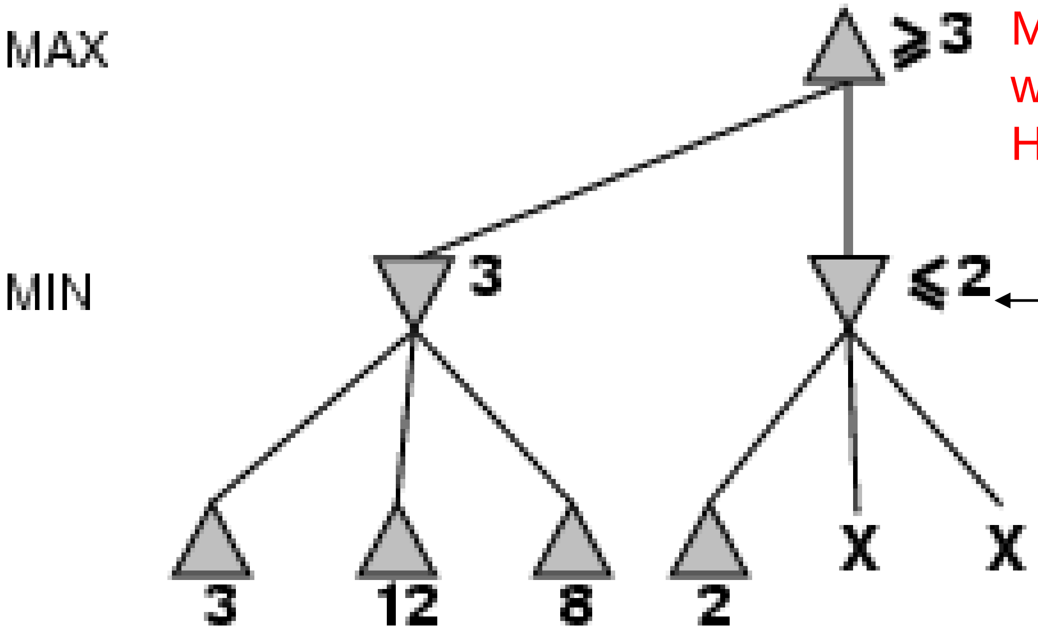




MAX knows that it can at least get “3” by playing this branch

MIN will choose “3”, because it minimizes the utility (which is good for MIN)

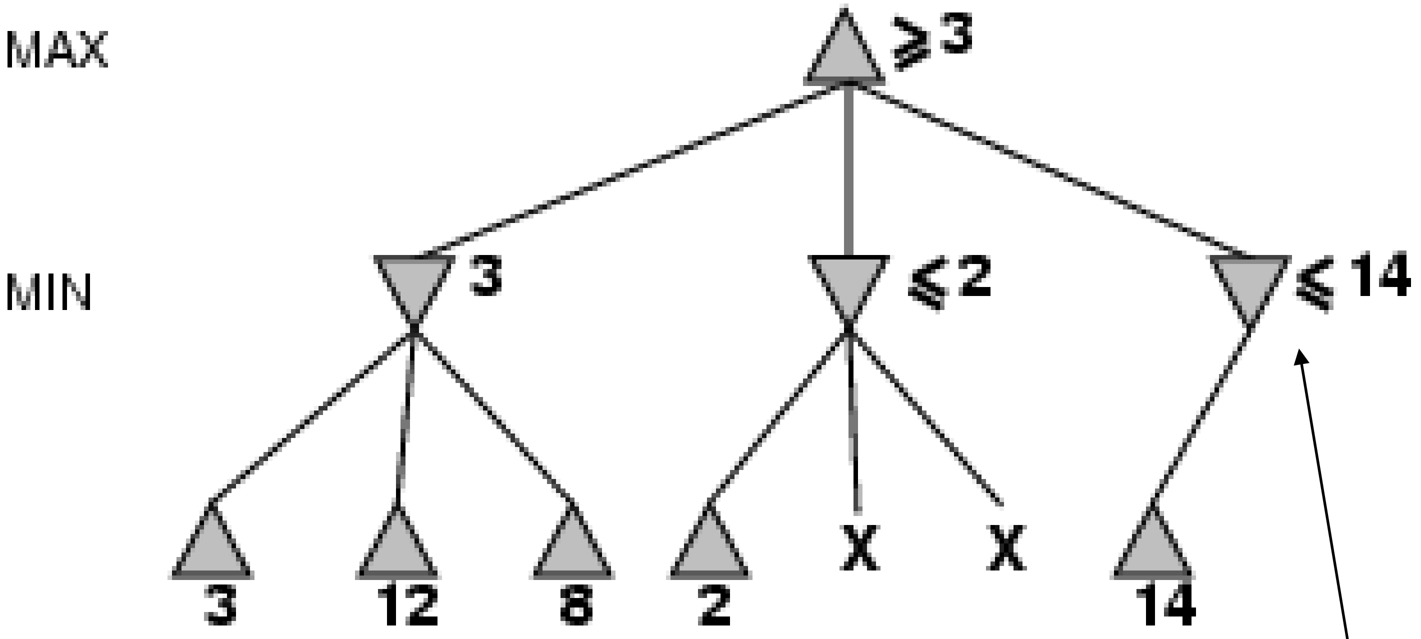
α-β pruning example



MAX knows that the new branch will never be better than 2 for him. He can *ignore* it.

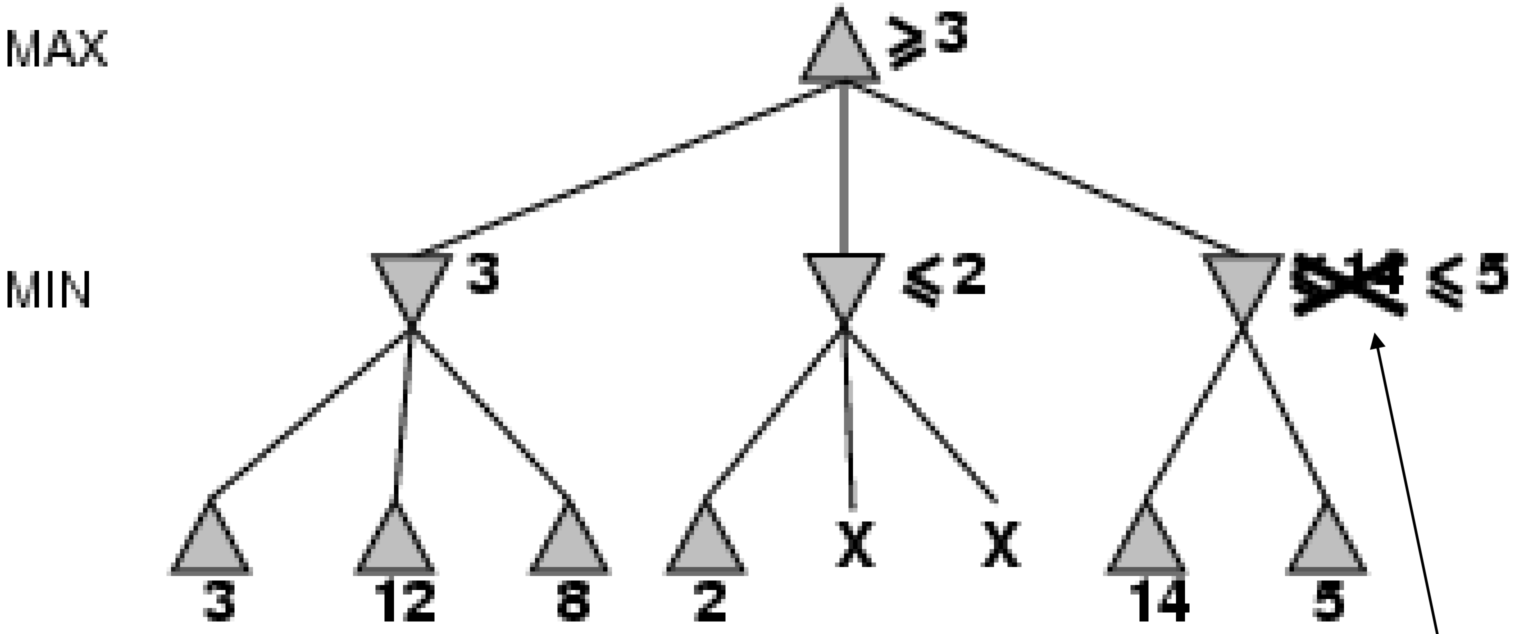
MIN can certainly do as good as 2, but maybe better (= smaller)

α-β pruning example



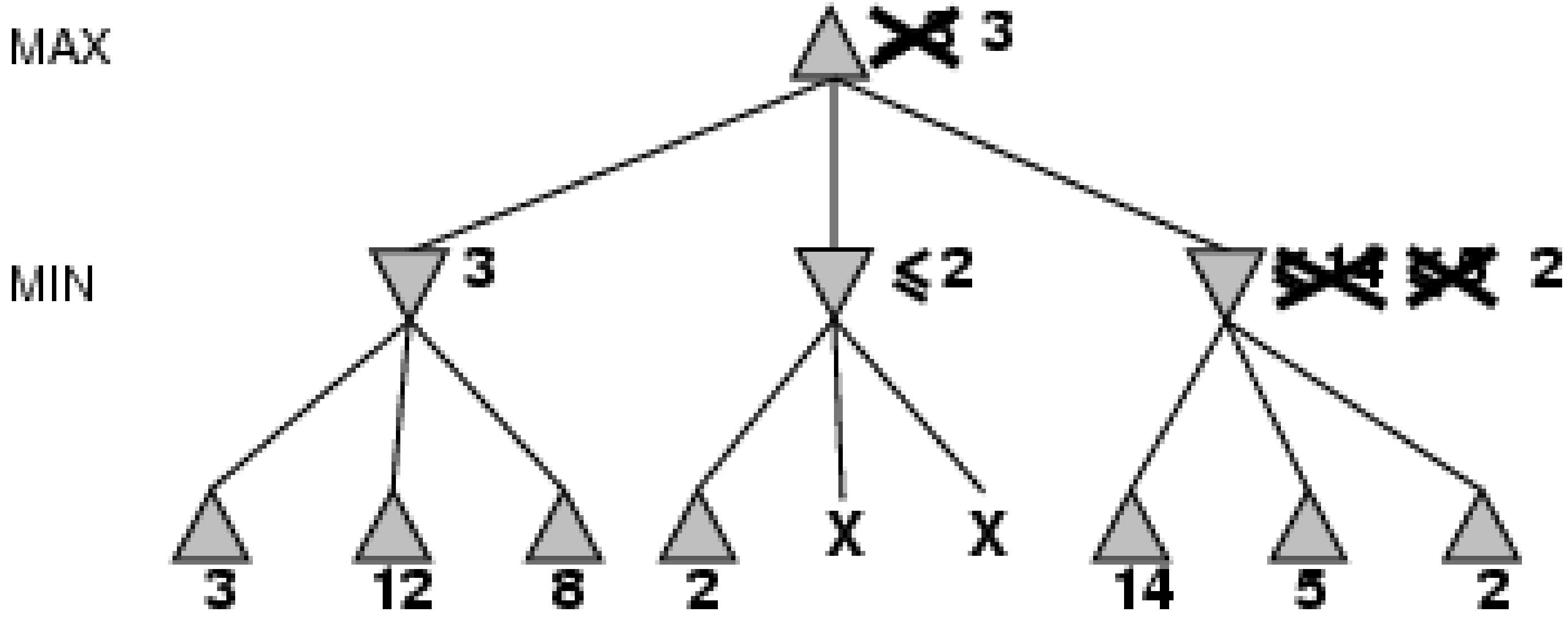
MIN will do at least as good as 14 in this branch (which is very good for MAX!) so MAX will want to explore this branch more.

α-β pruning example



MIN will do at least as good as 5 in this branch (which is still good for MAX) so MAX will want to explore this branch more.

α-β pruning example



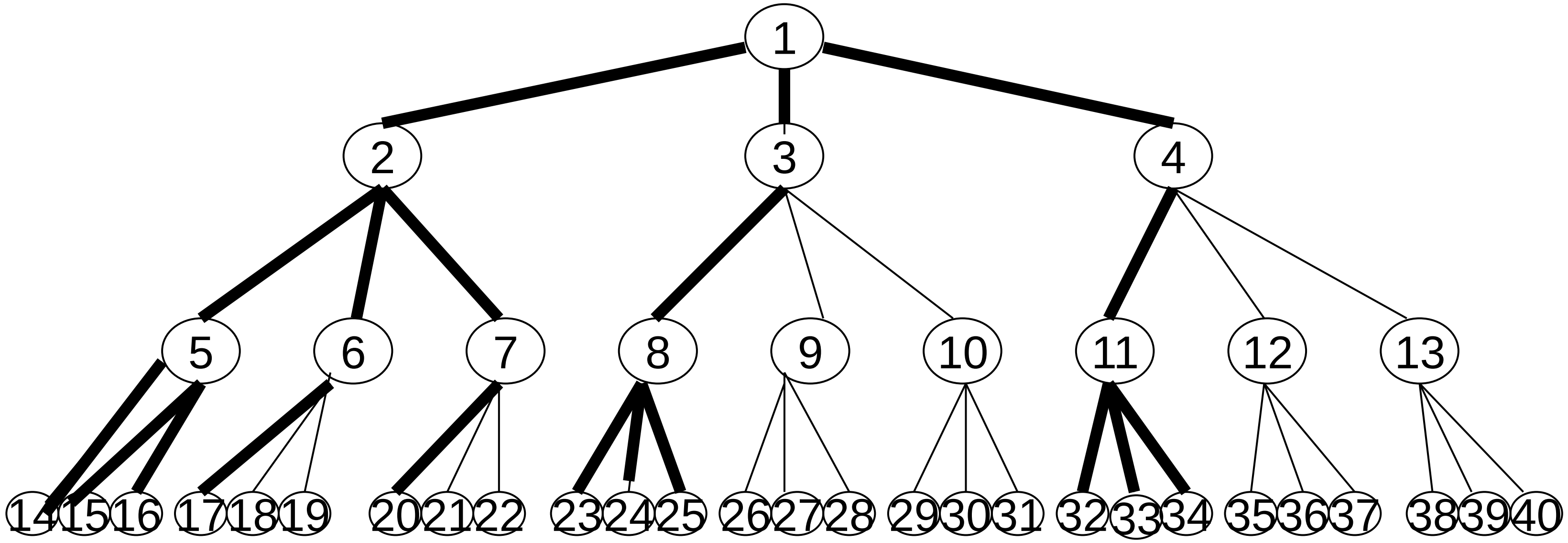
pity (for MAX): MIN will be able
to play this last branch and get 2. This
is worse than 3, so MAX will play 3.

α-β pruning

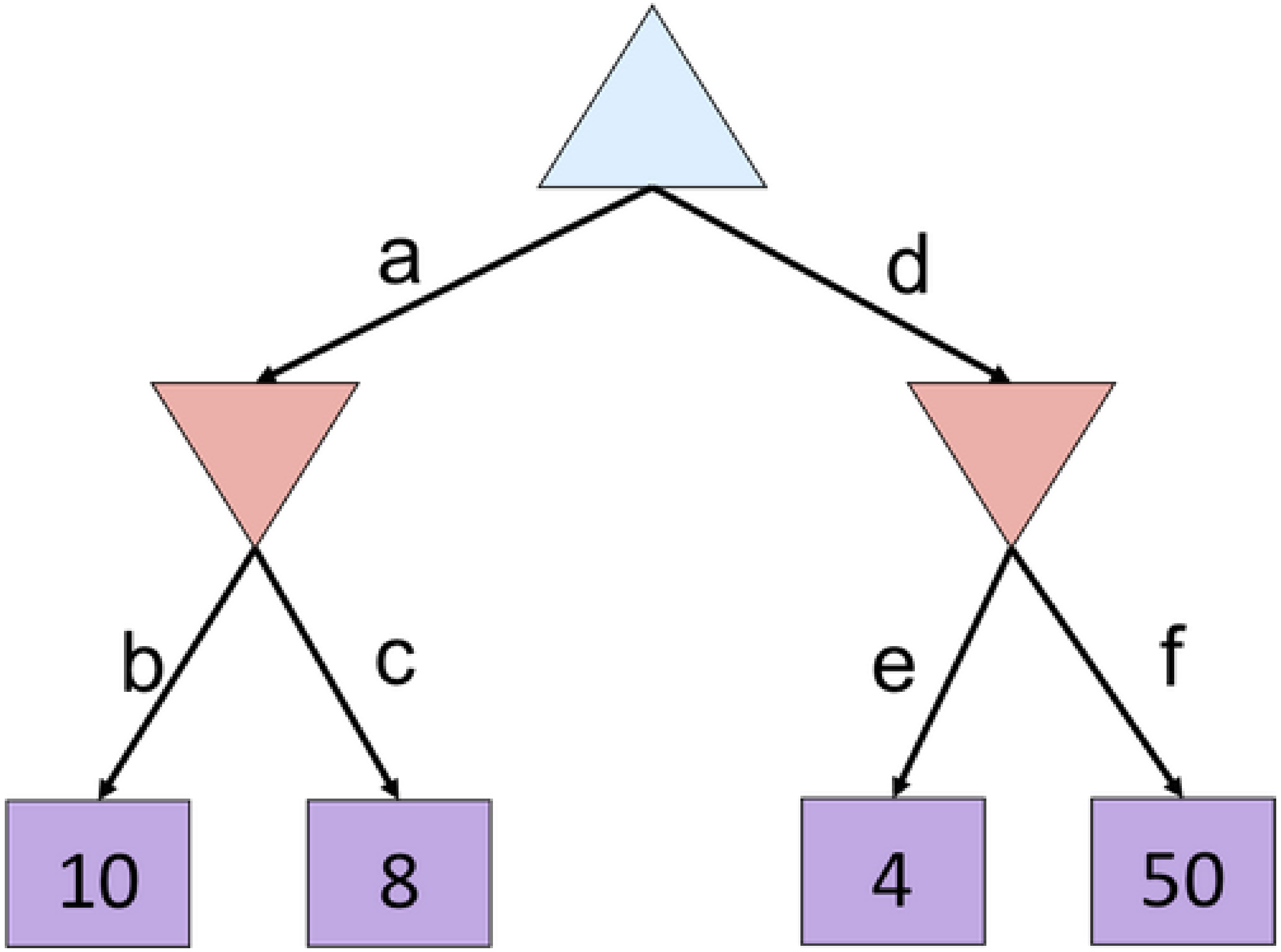
- Why we call it as α-β pruning?
- The name come from the two extra parameters in MaxValue(state, α, β).
- α = The **highest value** choice we have found so far at choice point along the path for MAX
- β = The **lowest value** choice we have found so far at choice point along the path for MIN.

α-β Pruning best and worst case

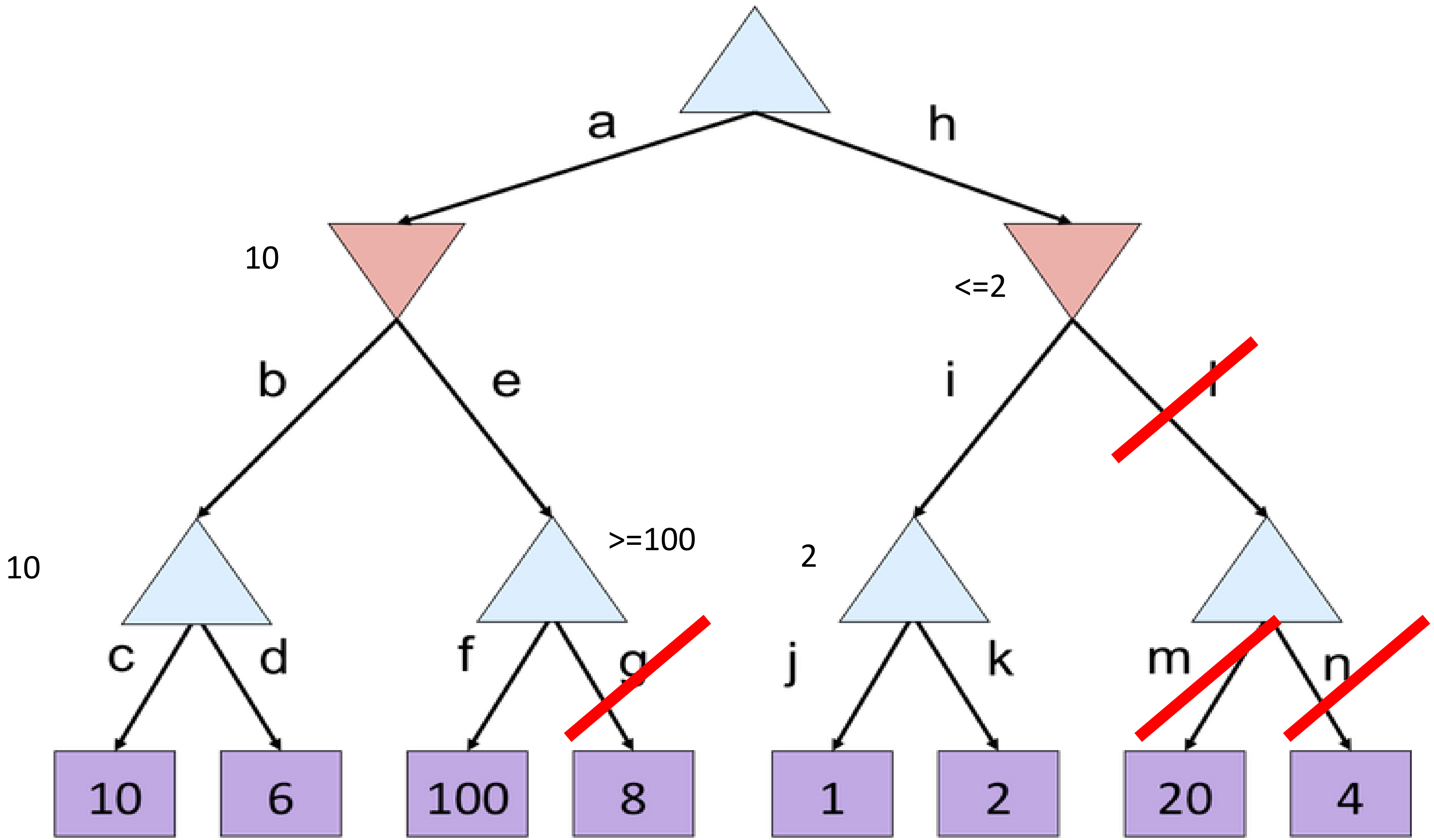
- Worst case:
  - Bad ordering: Alpha-beta prunes NO nodes
- Best case:
  - Assume cooperative oracle orders nodes
    - Best value on left



Alpha-Beta Quiz



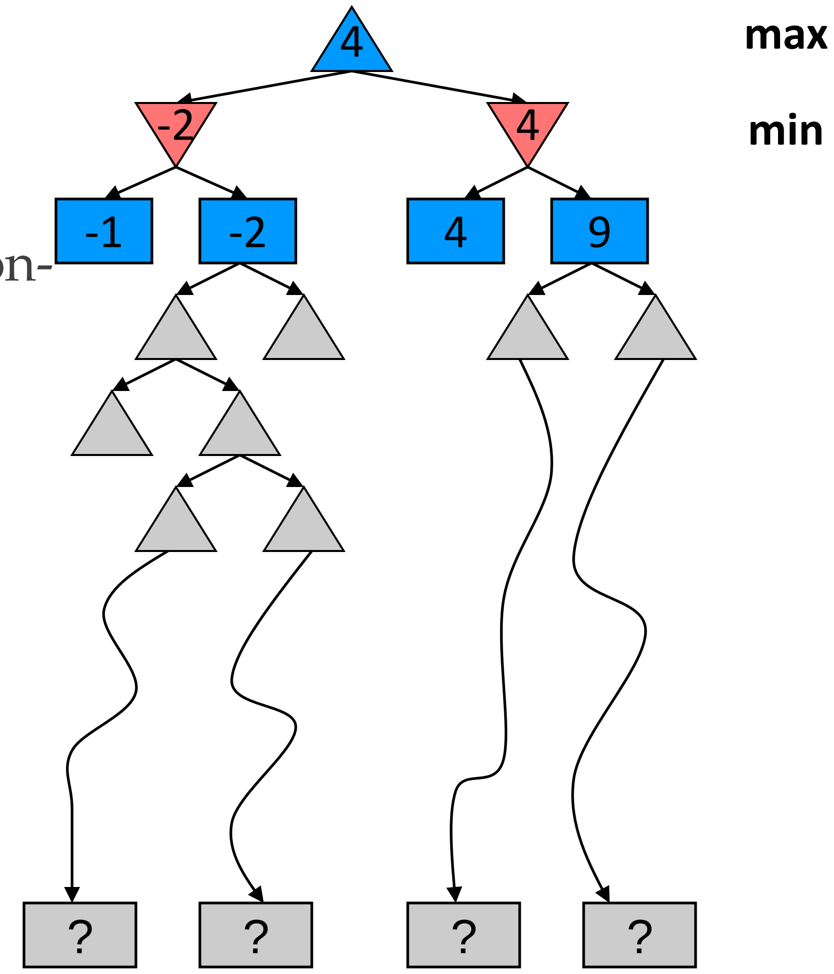
Alpha-Beta Quiz 2



Resource Limits

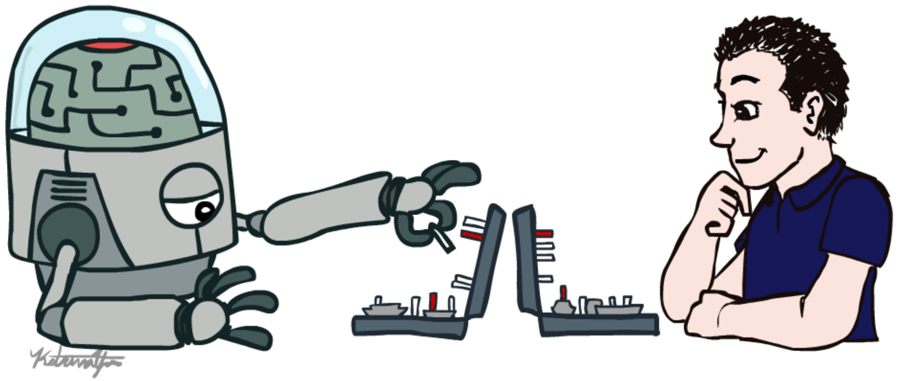
Resource Limits

- Problem: In realistic games, cannot search to leaves!
- Solution: Depth-limited search
  - Instead, search only to a limited depth in the tree
  - Replace terminal utilities with an evaluation function for non-terminal positions
- Example:
  - Suppose we have 100 seconds, can explore 10K nodes / sec
  - So can check 1M nodes per move
  - α-β reaches about depth 8 – decent chess program
- Guarantee of optimal play is gone
- More plies makes a BIG difference
- Use iterative deepening for an anytime algorithm



Next class?

- Propositional logic
- First order logic
- Quantifier



Thanks!