Agents that Search Together: Adversarial Search

Russell and Norvig: Chapter 5

CSE 240: Winter 2023

Lecture 5

Announcements

- Assignment 1 is due WEDNESDAY at 5pm
- Quiz 1 on Thursday: will open at 11:25am (after class on Thursday)
  - Due Friday at 5pm.
  - Open book, open note
  - 30 minutes
  - Time added for DRC.

My Meta-Learning Tips: Quizzes

- Manage stress through relaxation and focus.
  - Relaxation: Take three deep breathes when quiz starts. If you find yourself getting anxious – breathe! It works!!! Be prepared, have notes, review, etc.
  - Focus: Manage time. Be a smart test taker, ALWAYS put down/choose something. Do the easy questions first. Limit distractions.

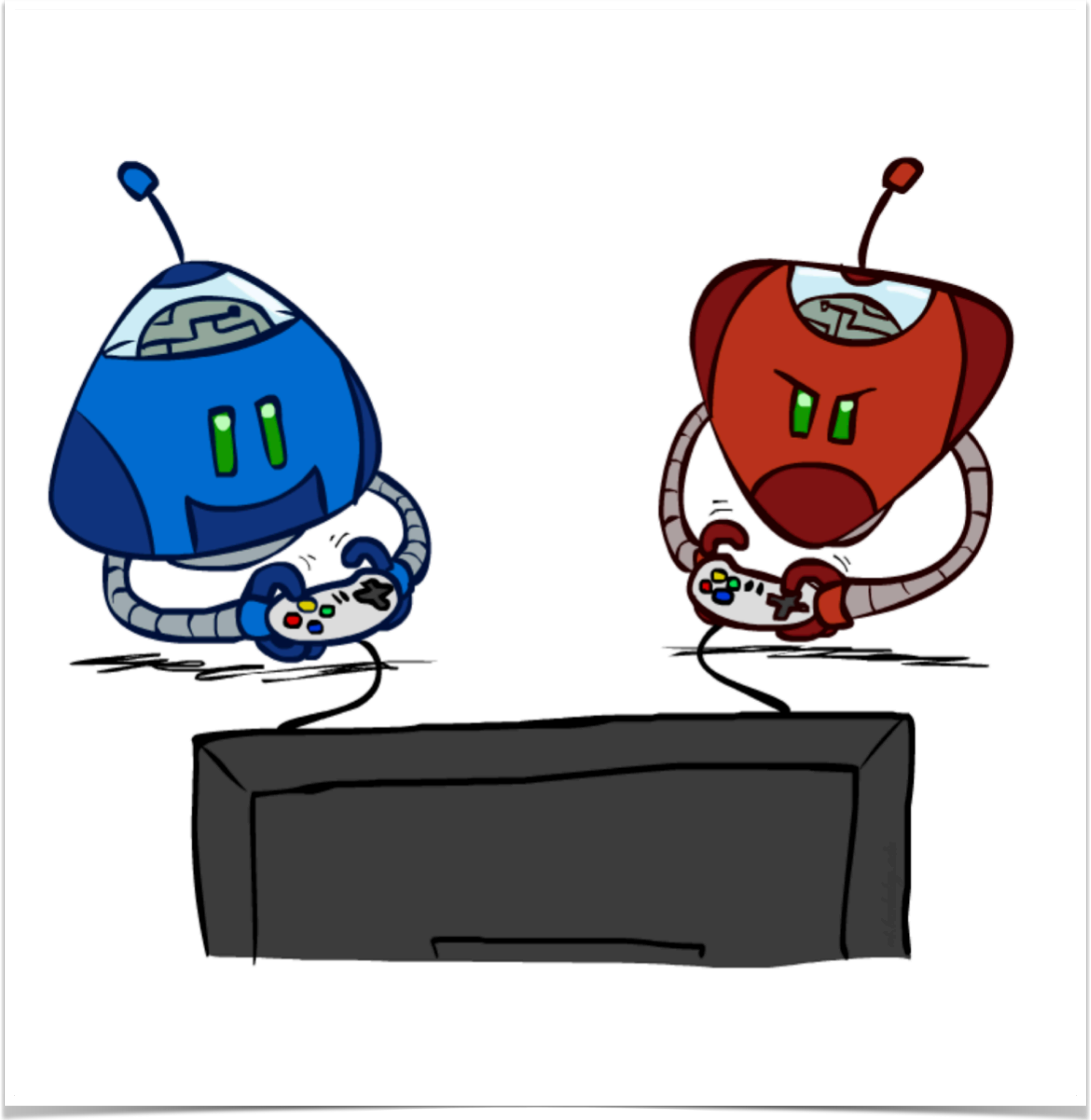
Agenda/Schedule

Week 3

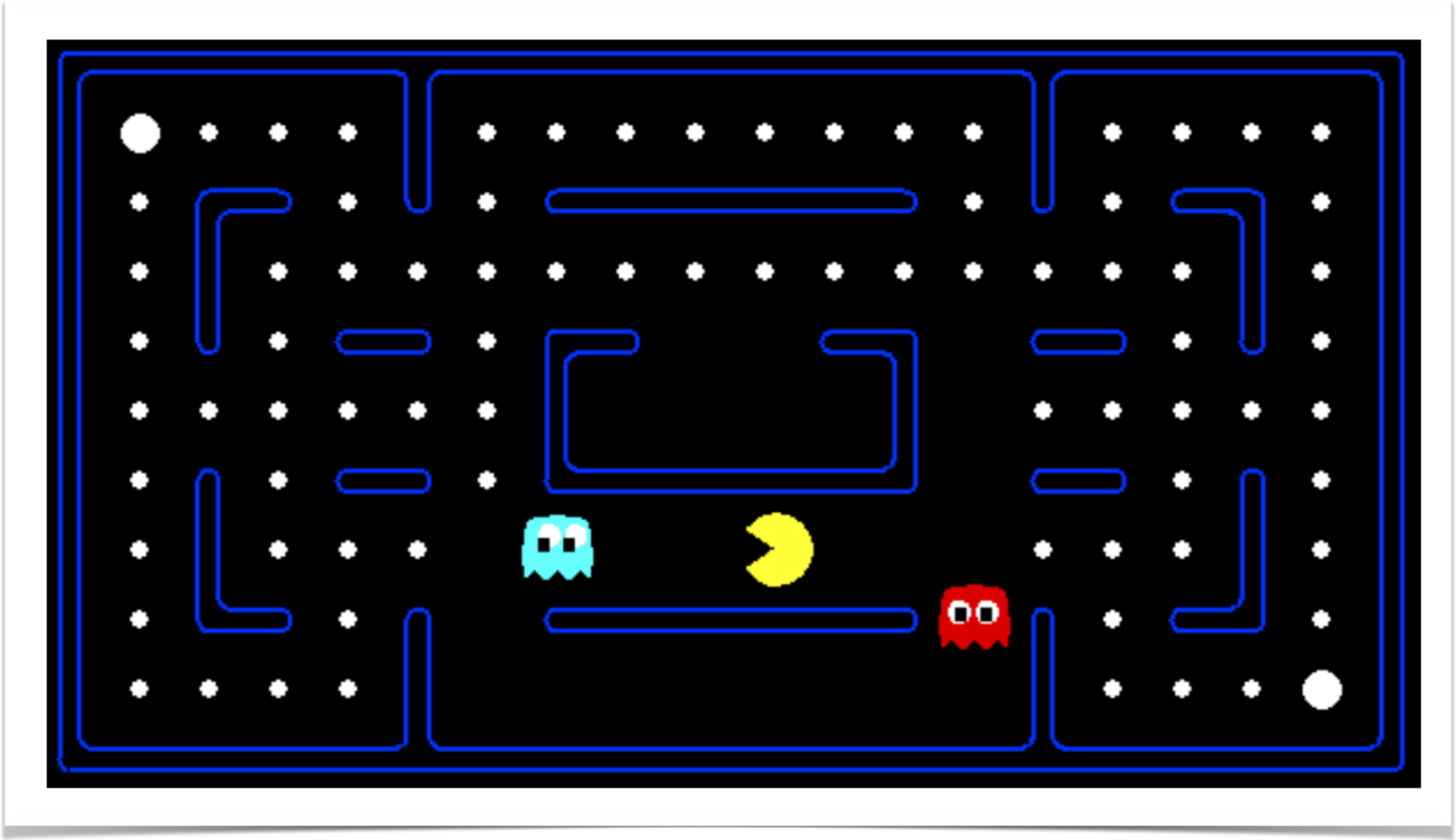
- Game theory
  - Adversarial games
    - Minimax algorithm and alpha-beta pruning
  - Stochastic games
    - Expectimax search algorithm

Today

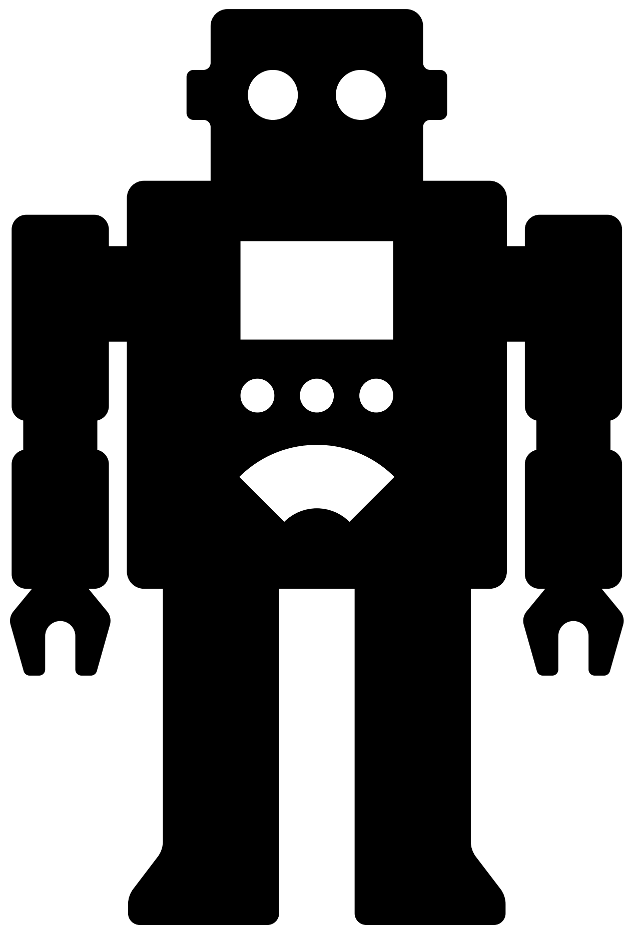
- Adversarial search
    - Or search with other agent



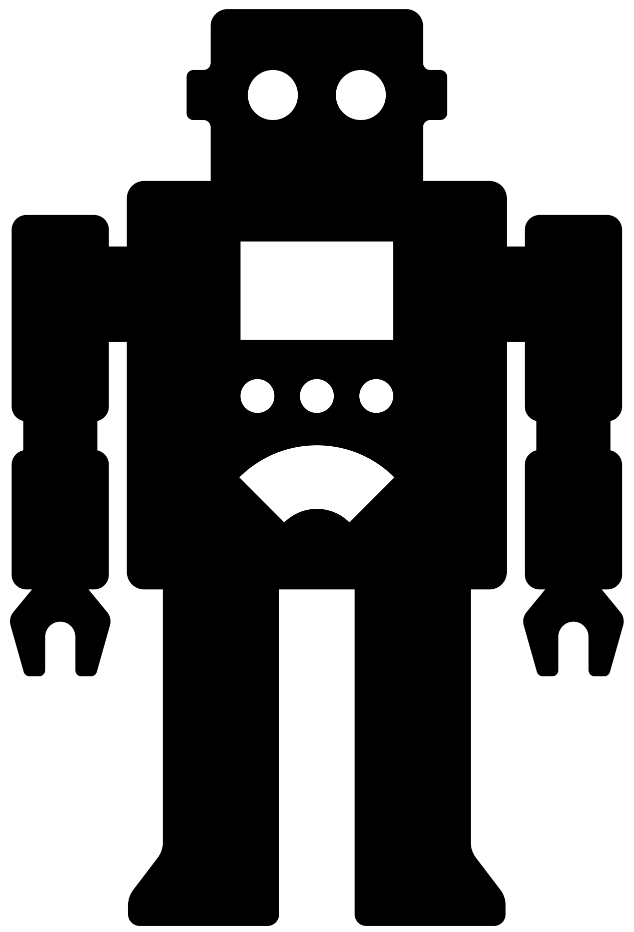
Behavior Based on Computation

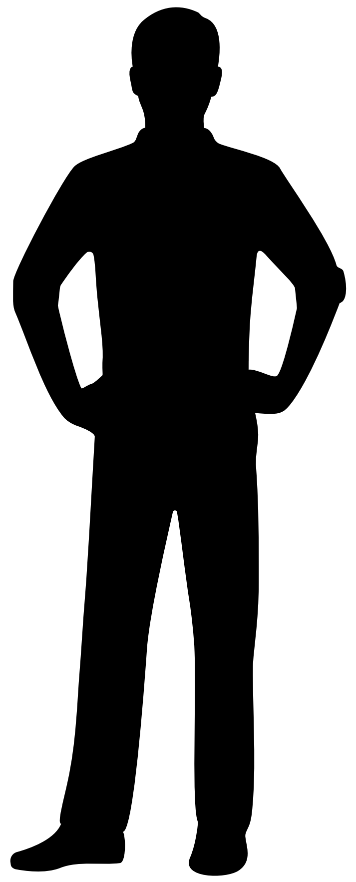


Agents Getting Along With Other Agents



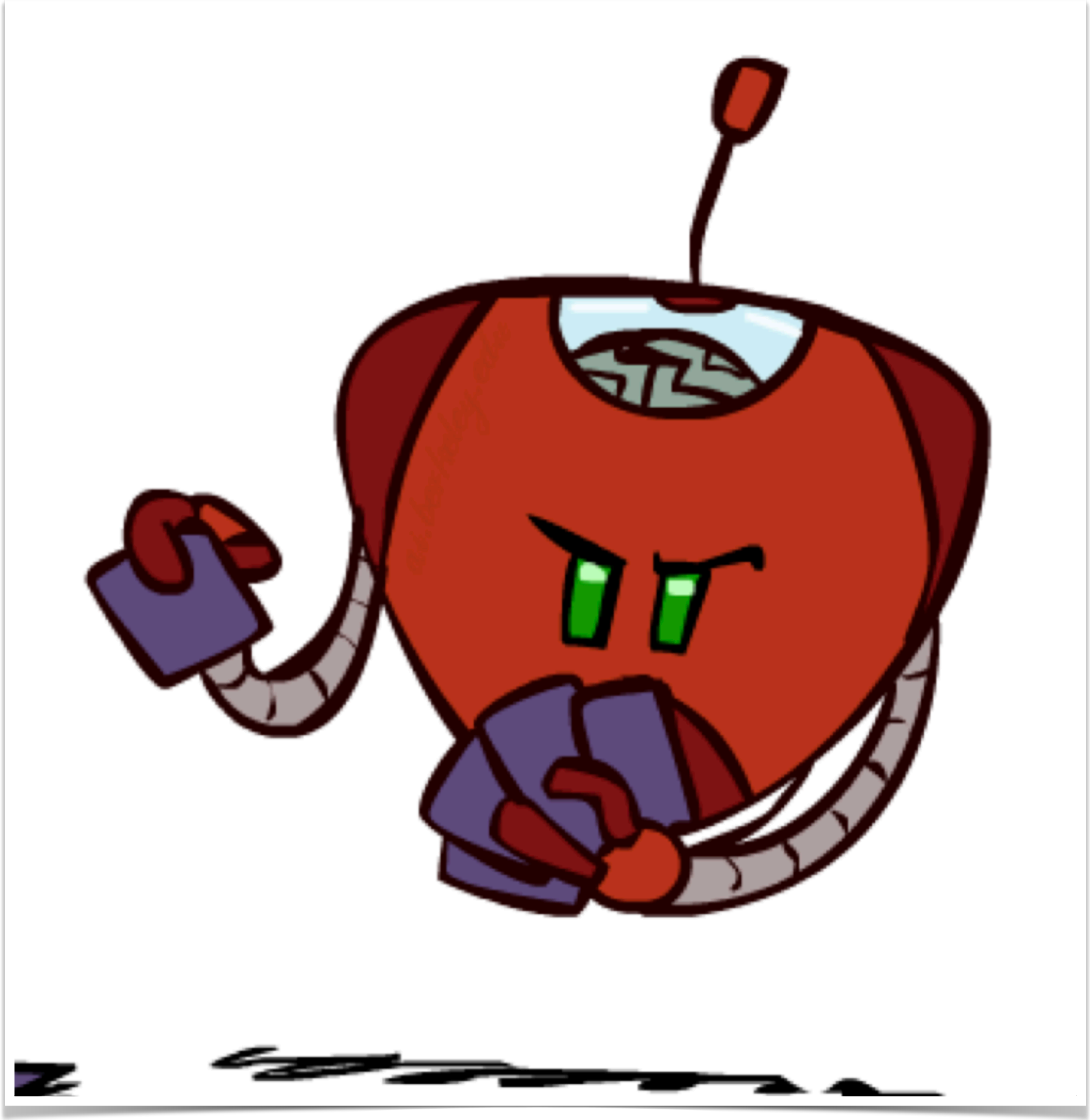
Agents Getting Along With Humans





Types of Game

- Many different kinds of games!
- Types
  - Deterministic of stochastic?
  - One, two or more players?
  - Zero sum?
  - Perfect information (can you see the state)?



Types of Games

- General Games
  - Agents have independent utilities (values on outcomes)
  - Cooperation, indifference, competition, and more are all possible
    - We don’t make AI to act in isolation, it should: 1) work around people and 2) help people
    - That means that every AI agents needs to solve a game.

- Zero-Sum Games
  - Agents have opposite utilities (values on outcomes)
  - Think of a single value that one maximizes and the other minimizes
  - Adversarial, pure competition

Primary Assumptions

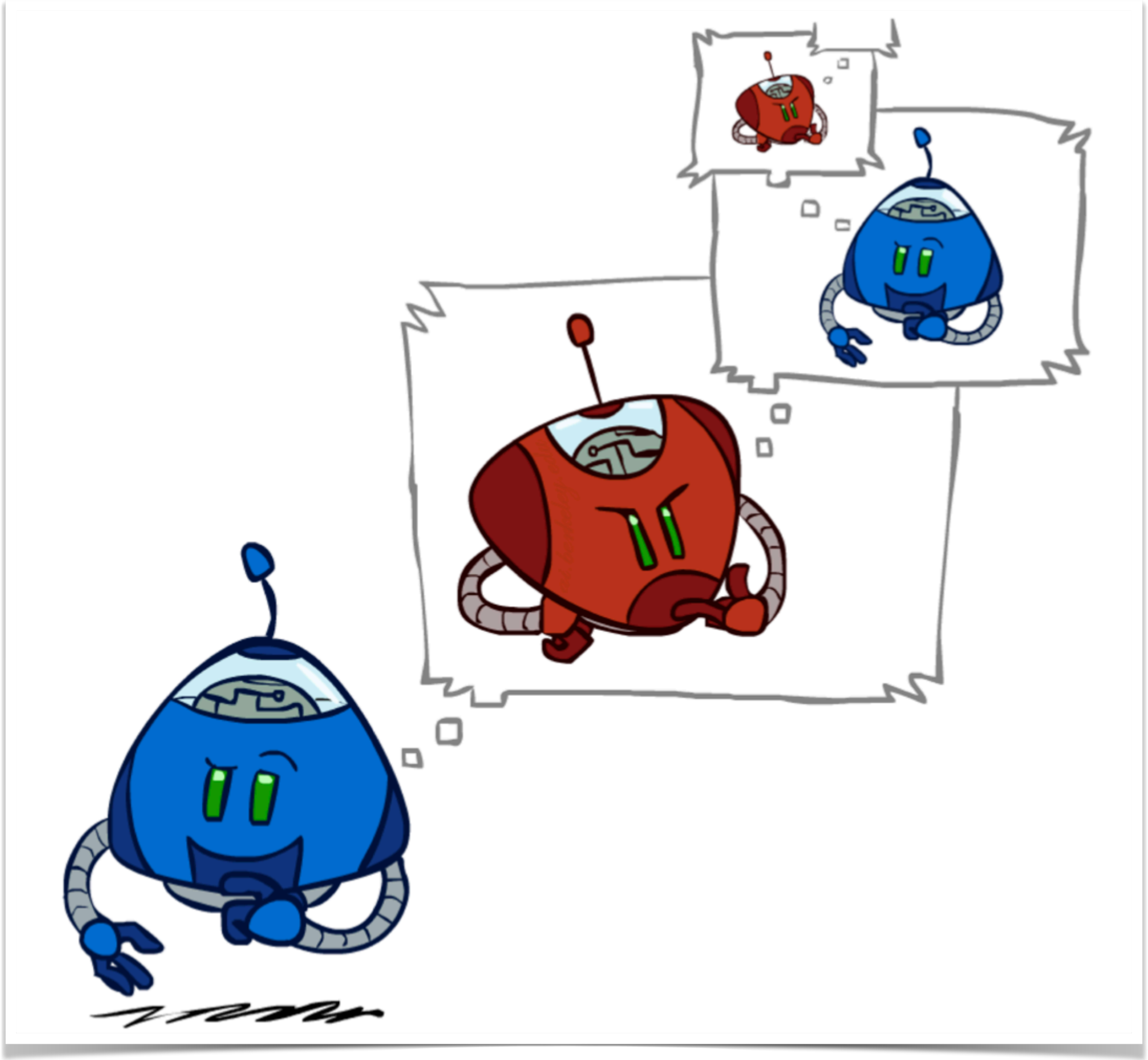
- We start with there games:
  - Two players
  - Turn taking -> agents act alternately
  - Zero sum -> agents’ goals are in conflict
  - Deterministic
  - Perfect information -> fully-observable

Deterministic Games with Terminal Utilities

- Many possible formalizations, one is:
  - States: S (start at s_0)
  - Players: $P = \{1,...,N\}$ (usually take turns)
  - Actions: A (may depend on player/state)
  - Transition Function: $S \times A \to S$
  - Terminal Test $S \to \{t, f\}$
  - Terminal Utilities: $S \times P \to R$
- Solution for a player is a policy: $S \to A$

Adversarial Games

Adversarial Search



Cost Utility

- No longer minimizing cost!
- Agent now wants to maximize its score/utility!

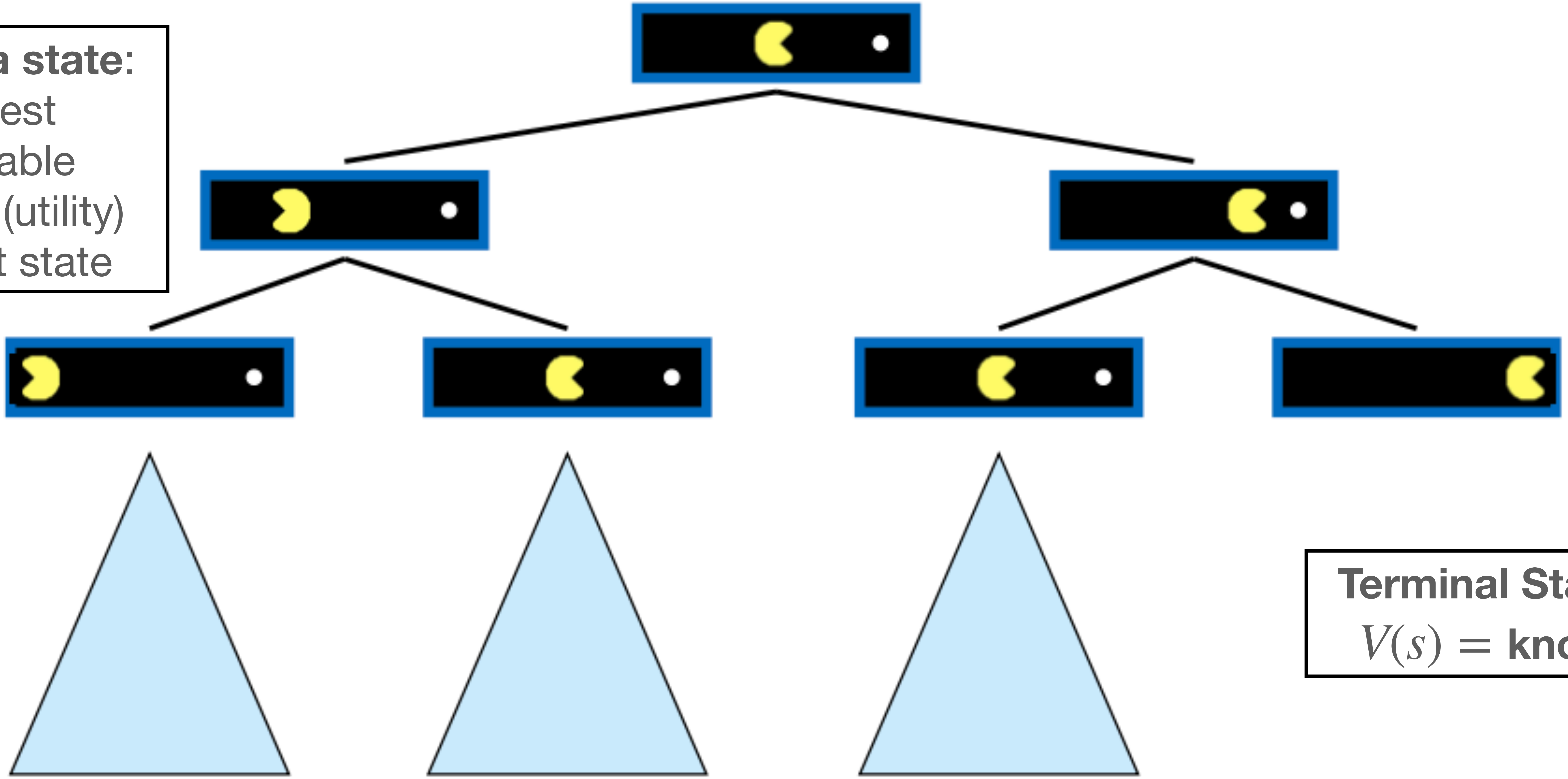
Single-Agent Trees

Single-Agent Trees

Single-Agent Trees

Non-terminal states:
$V(s) = \max_{s' \in children(s)} V(s')$

Value of a state: The best achievable outcome (utility) from that state

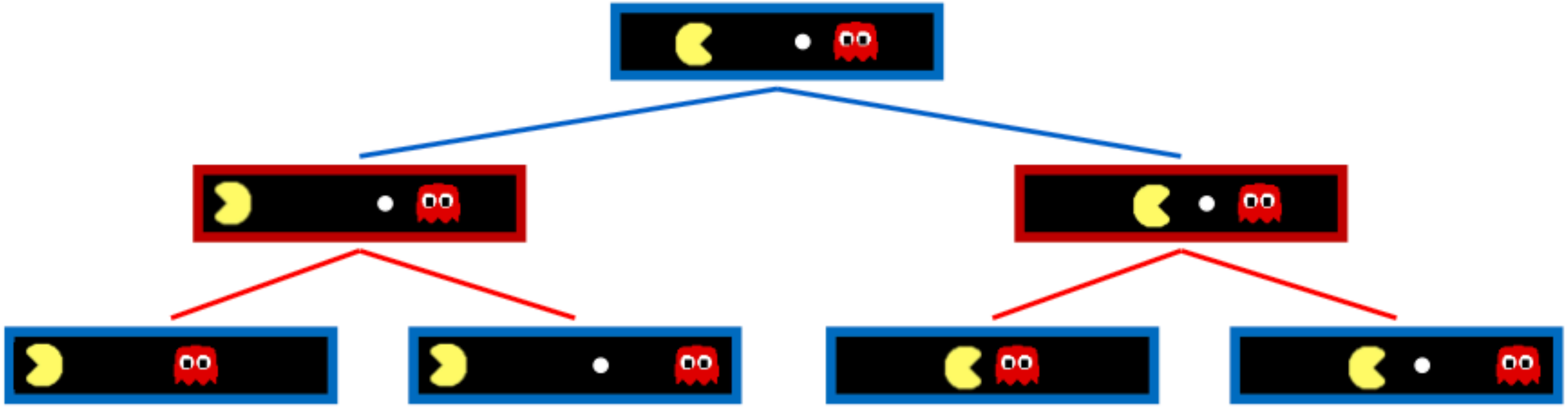


Terminal States:
$V(s) =$ **known**

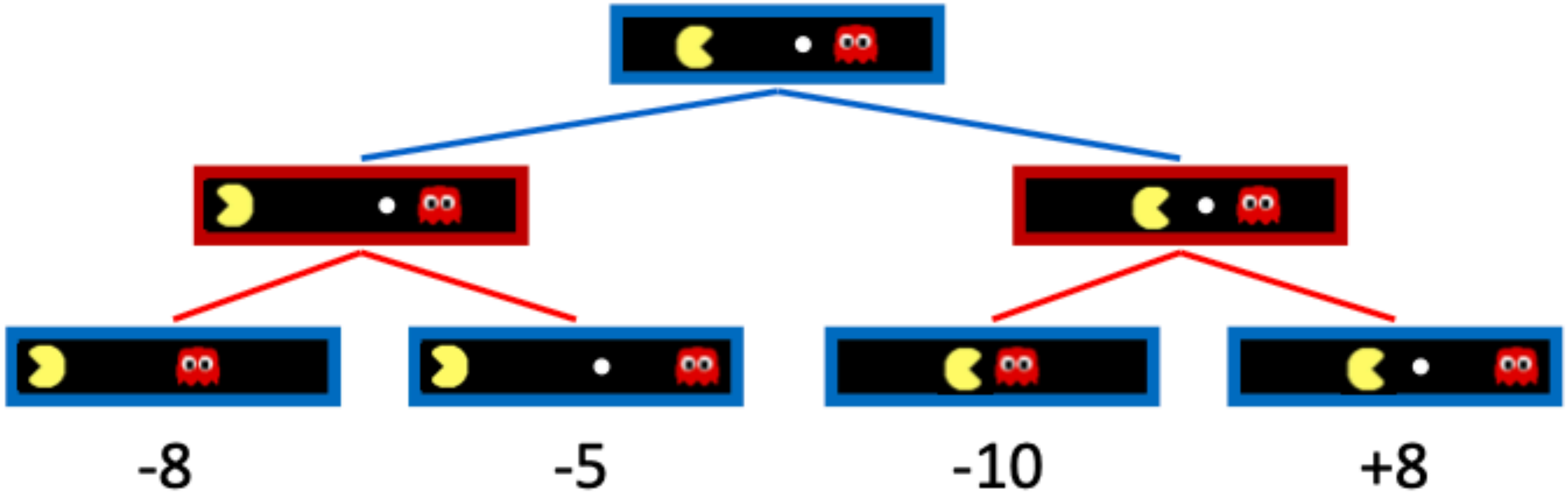
Adversarial Game Trees

Adversarial Game Trees

Adversarial Game Trees



Minimax Values

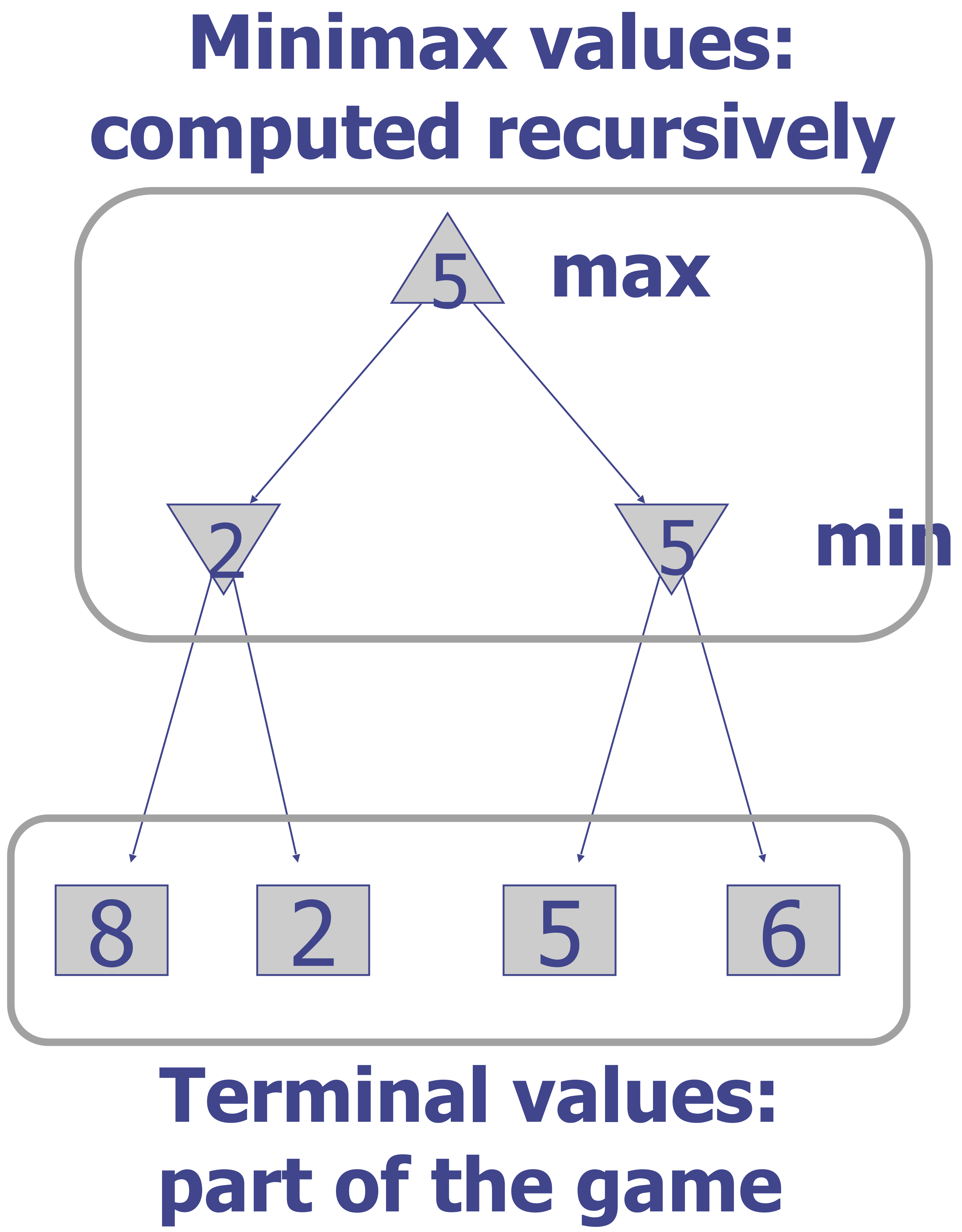


Terminal States:

$V(s) = \text{known}$

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
  - Each node has a minimax value: best achievable utility against a rational adversary



Optimal strategies

- Find the **strategy** for MAX assuming an infallible MIN opponent.
- Assumption: Both players play optimally !!
- Given a game tree, the optimal strategy can be determined by using the minimax value of each node:

MINIMAX-VALUE(n)=

UTILITY(n)	If n is a terminal
$\max_{s \in \text{successors}(n)}$ MINIMAX-VALUE(s)	If n is a max node
$\min_{s \in \text{successors}(n)}$ MINIMAX-VALUE(s)	If n is a min node

Computing Minimax Values

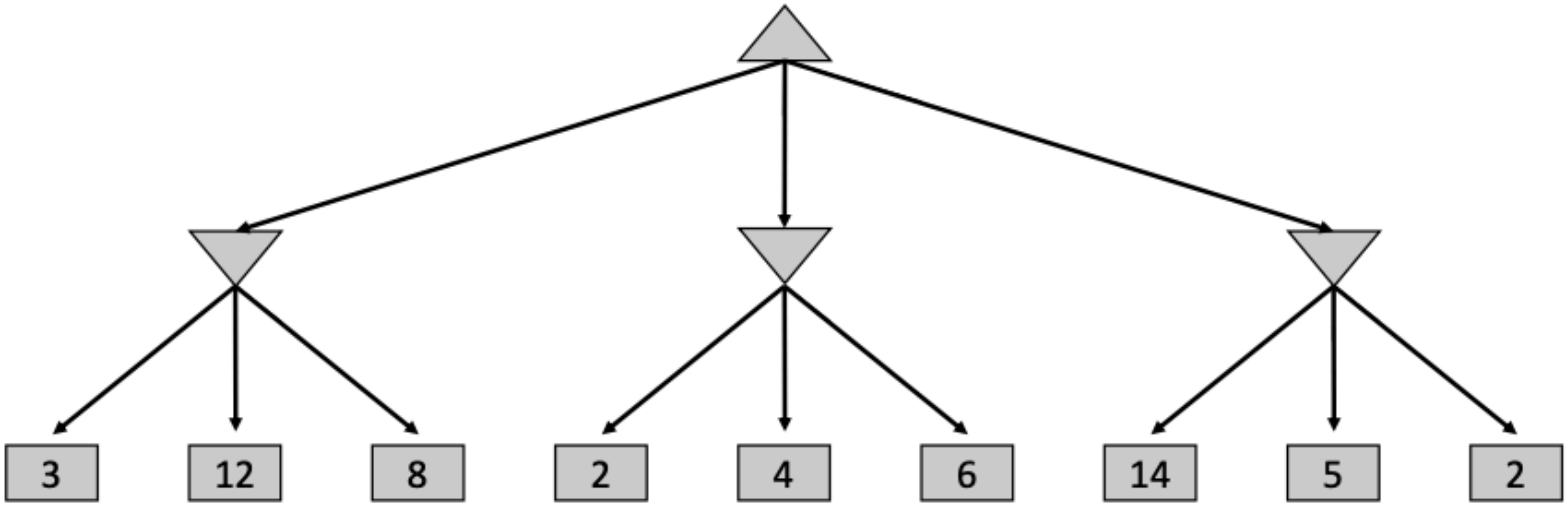
- Two recursive functions:
  - max-value maxes the values of successors
  - min-value mins the values of successors

- 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)

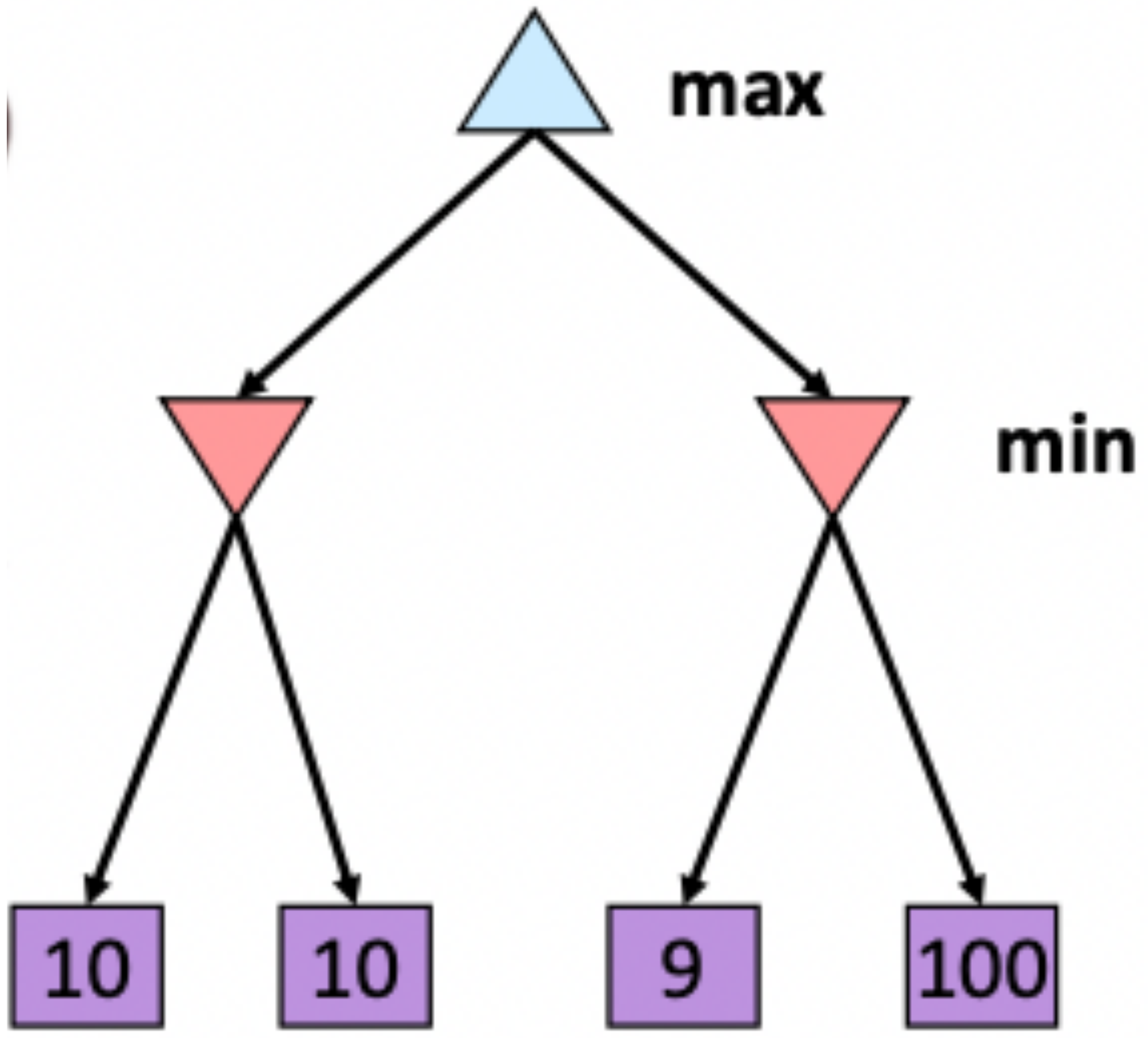
```
def max-value(state):
    Initialize max = -∞
    For each successor of state:
        Compute value(successor)
        Update max accordingly
    Return max
```

```
def min-value(state):
    Initialize min = ∞
    For each successor of state:
        Compute value(successor)
        Update min accordingly
    Return min
```

CE 5: Minimax



Minimax Properties



Optimal against a perfect player. Otherwise?

Properties of Minimax

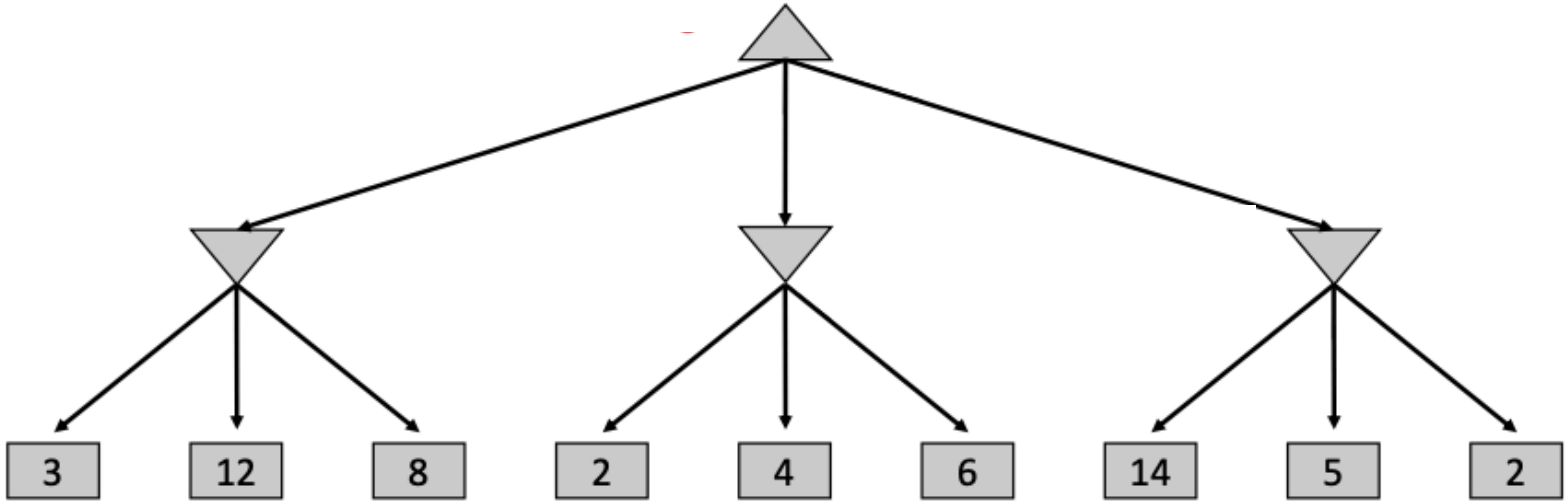
Criterion	Minimax
Complete?	yes, ☺
Optimal?	yes, ☺
Time	$O(b^m)$, ☹
Space	$O(bm)$, ☺

in theory...

For chess, b~35, m~100, exact solution is completely infeasible

Pruning

Minimax Example

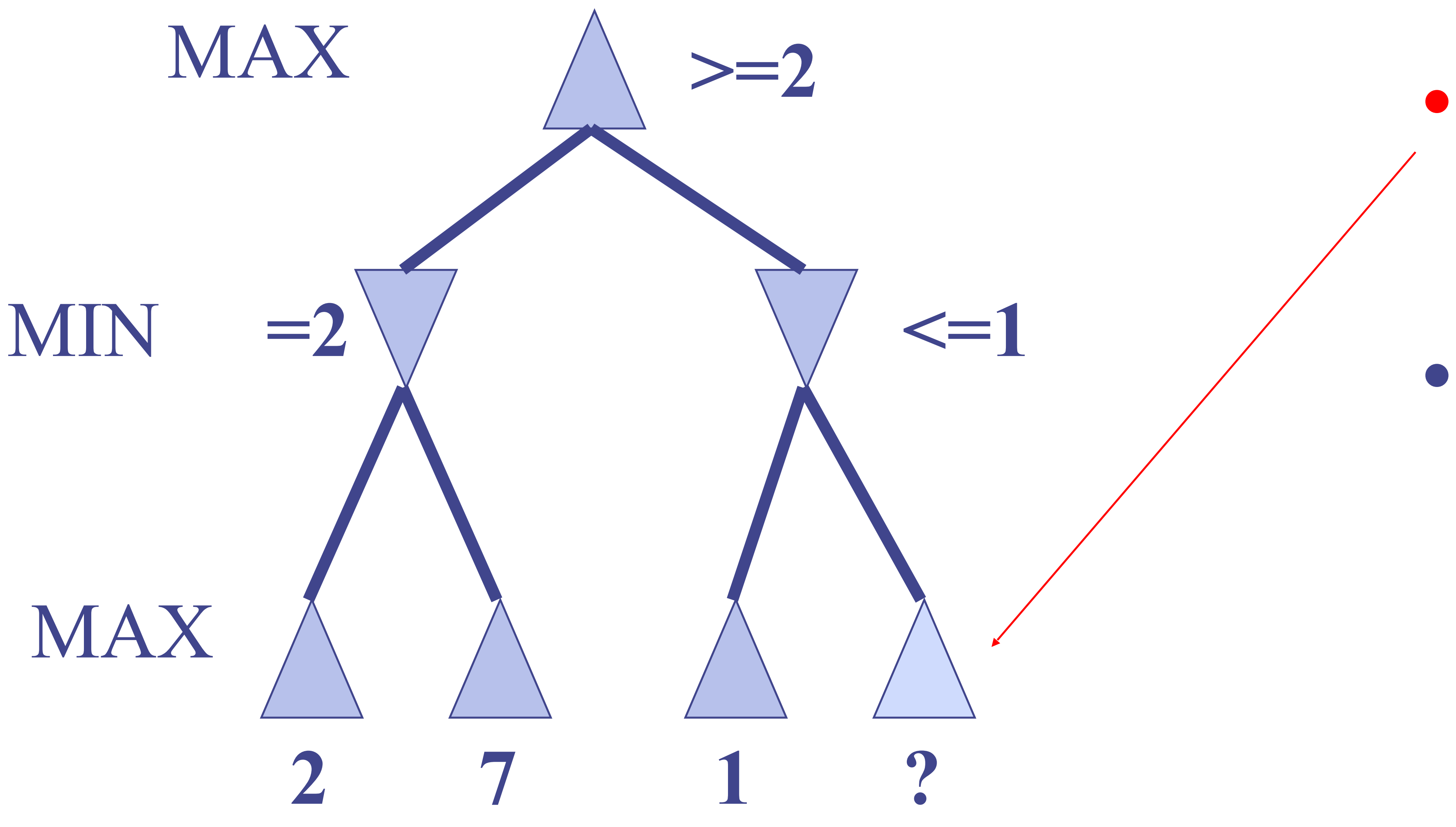


Problem of minimax search

- Number of games states is exponential to the number of moves.
  - Solution: Do not examine every node
  - Alpha-beta pruning
    - Remove branches that do not influence final decision

Alpha-beta pruning

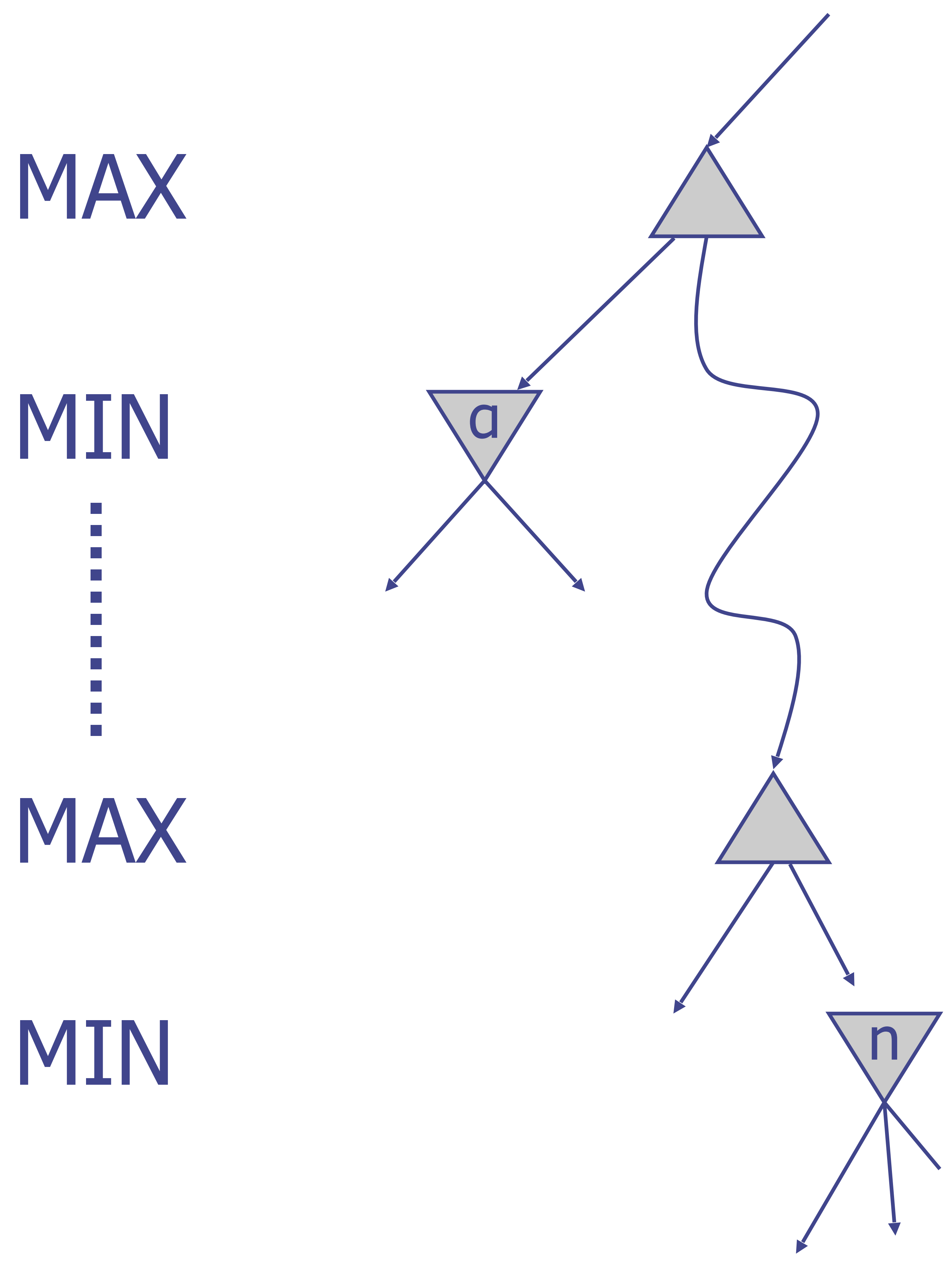
- We can improve on the performance of the minimax algorithm through **alpha-beta pruning**
- Basic idea: “If you have an idea that is surely bad, don't take the time to see how truly awful it is.” -- Patrick Winston



- We don’t need to compute the value at this node.
- No matter what it is, it can’t affect the value of the root node.

Alpha-Beta Pruning

- General configuration
  - We’re computing the MIN-VALUE at n
  - We’re looping over n’s children
  - n’s value estimate is dropping
  - α is the best value that MAX can get at any choice point along the current path
  - If n becomes worse than α, MAX will avoid it, so can stop considering n’s other children
  - Define β similarly for MIN



Alpha Beta Implementation

α: MAX's best option on path to root
β: MIN's best option on path to root

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

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

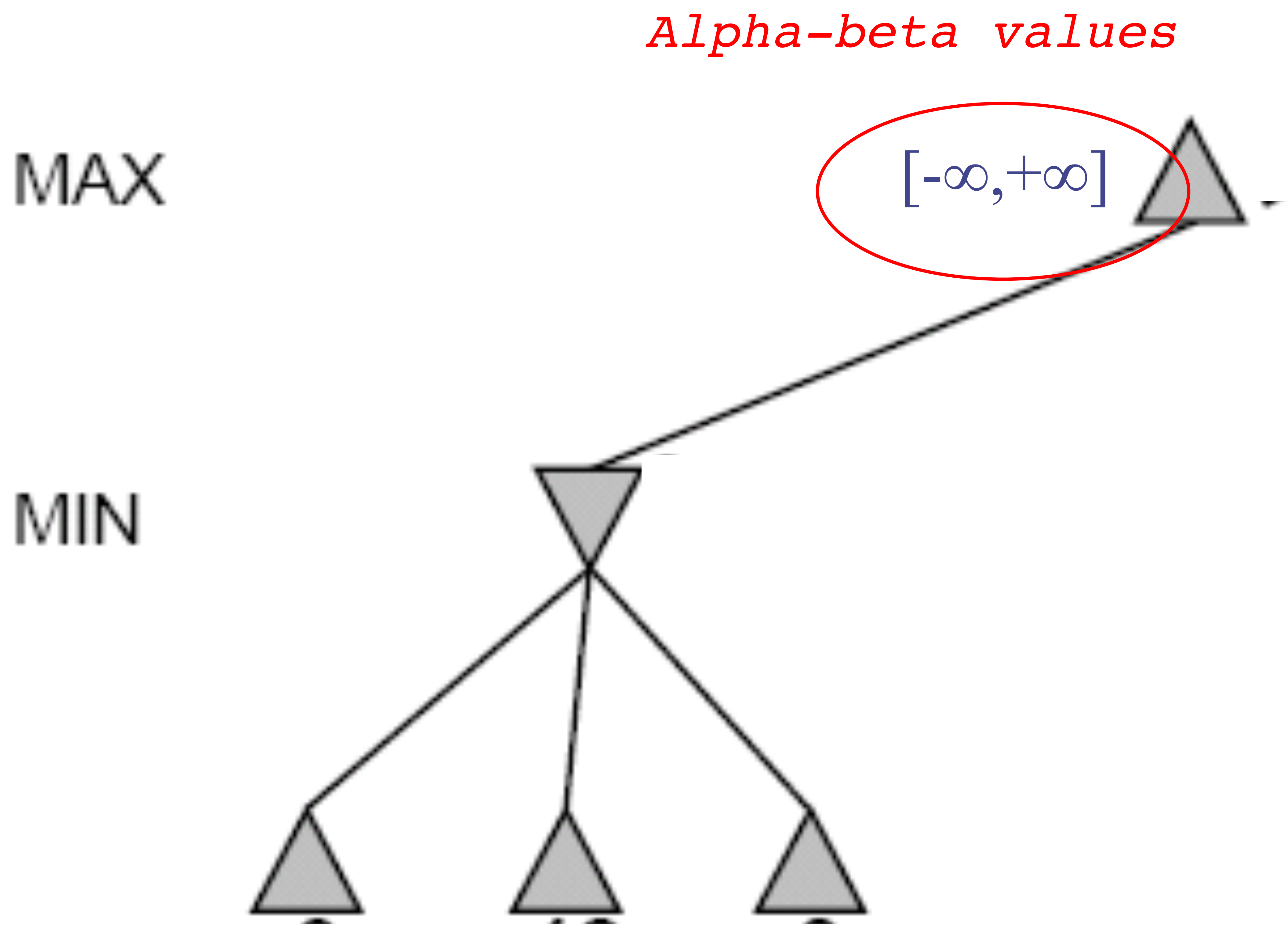
Alpha Beta Pruning Properties

- This pruning has no effect on minimax value computed for the root!
- Values of intermediate nodes might be wrong
  - Important: children of the root may have the wrong value
  - So the most naive version won't let you do action section
- Good child ordering improves effectiveness of pruning
- With "perfect ordering"
  - Time complexity drops to $O(b^{m/2})$
  - Doubles solvable depth!
  - Full search of, e.g., chess, is still hopeless

α: best already explored path along path to root for maximer
β: best already explored path along path to root for minimizer

Alpha-Beta Example

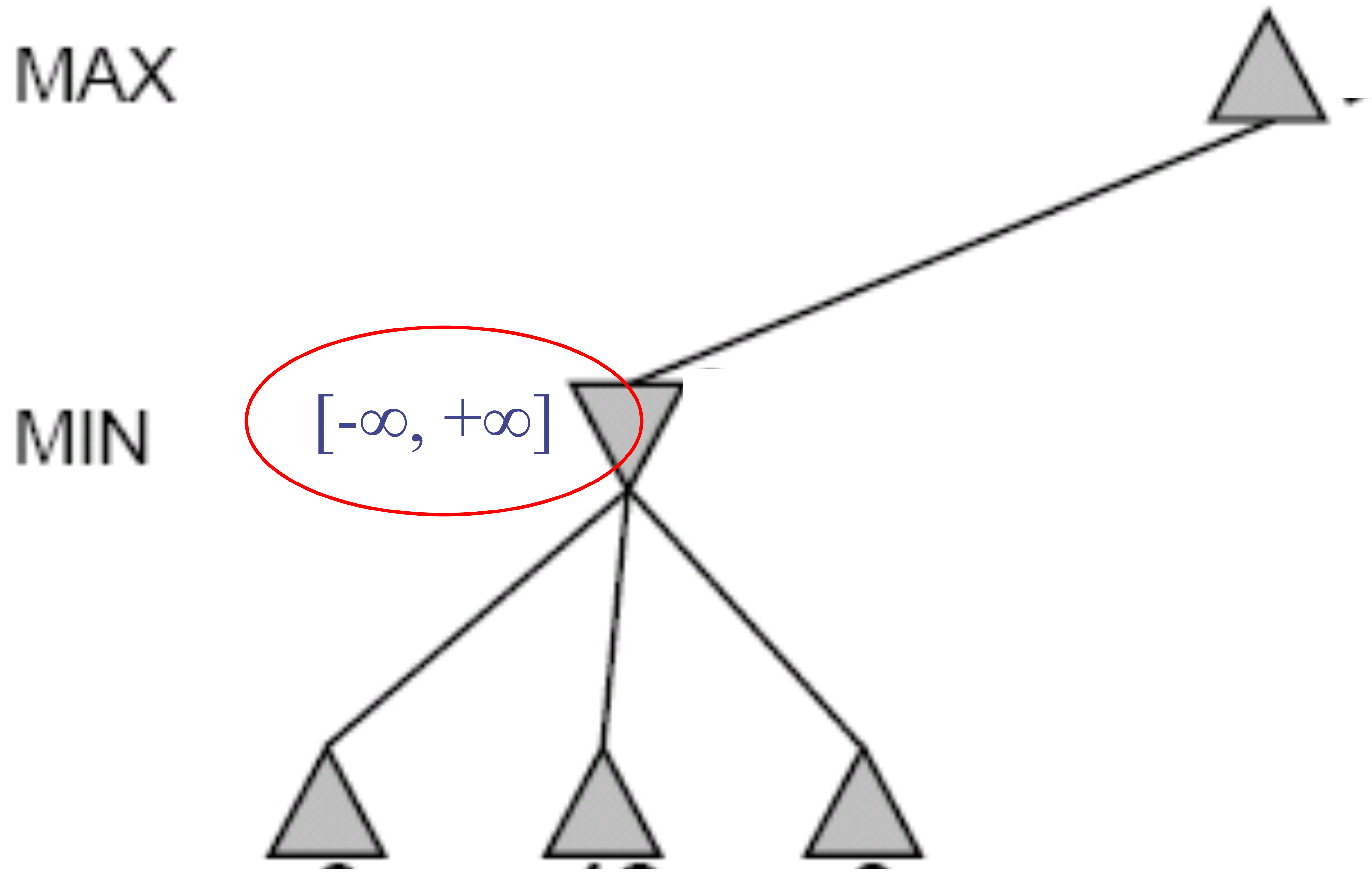
Do DF-search until first leaf



α: best already explored path along path to root for maximer
β: best already explored path along path to root for minimizer

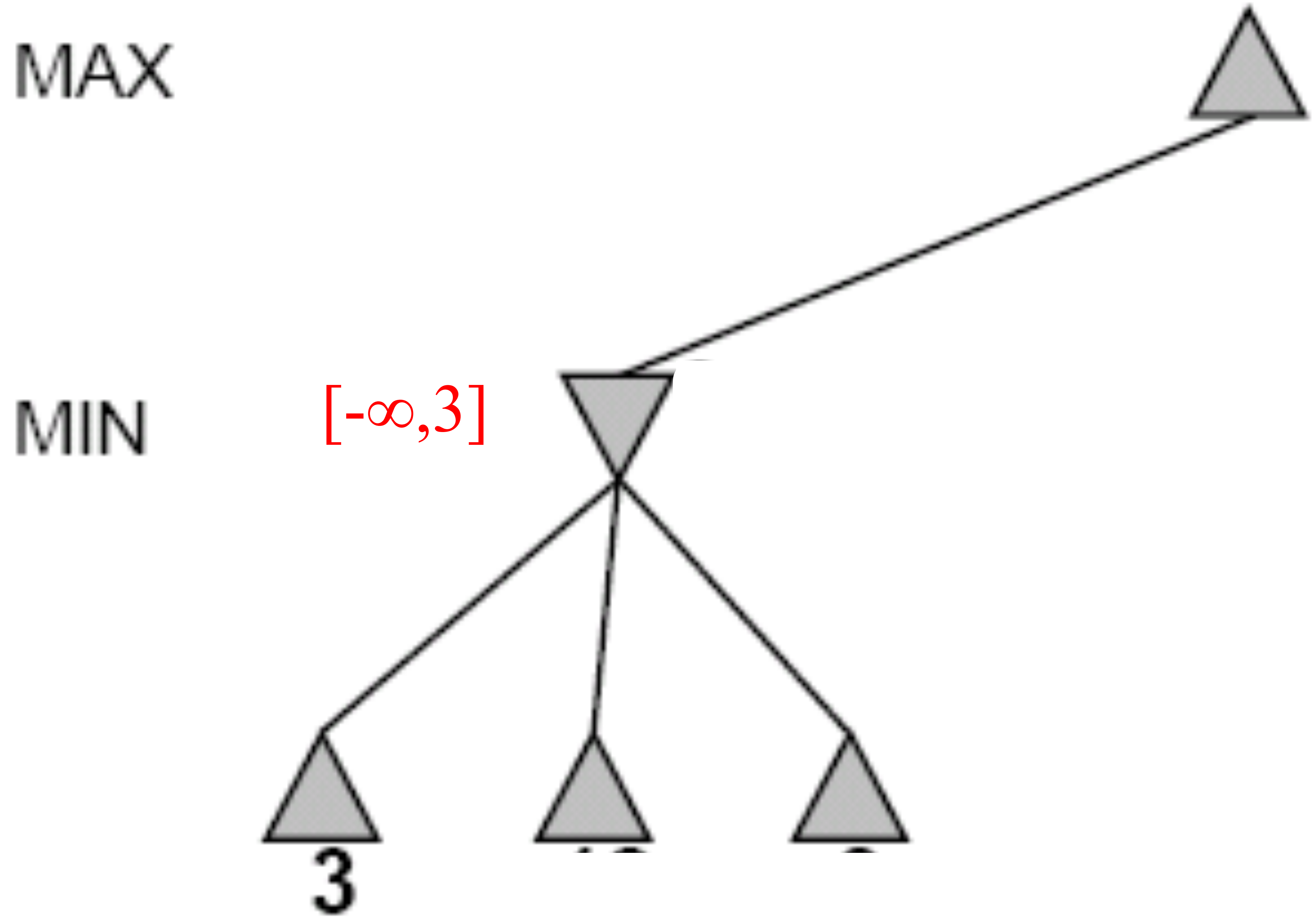
Alpha-Beta Example

Do DF-search until first leaf



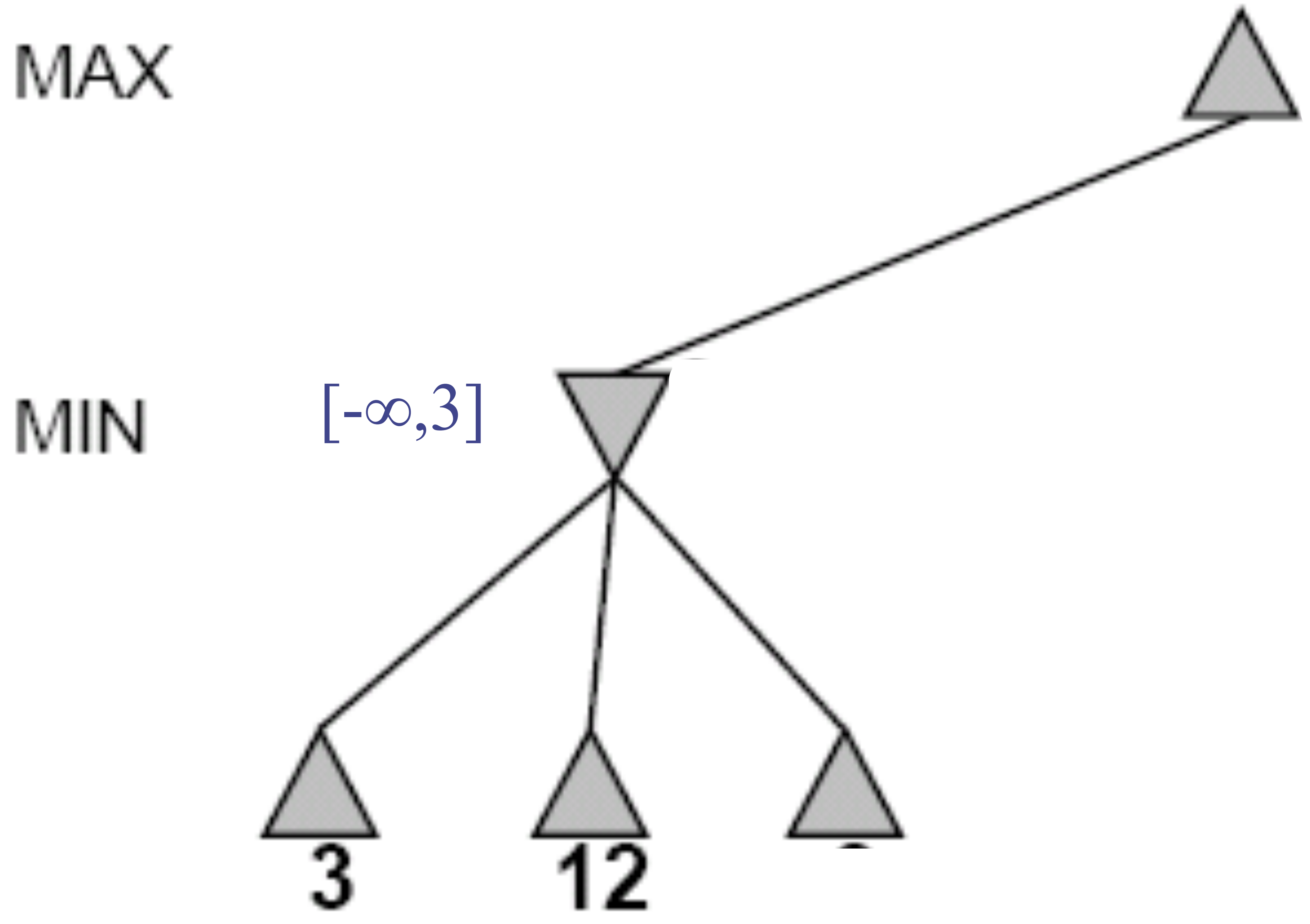
α: best already explored path along path to root for maximer
β: best already explored path along path to root for minimizer

Alpha-Beta Example (continued)



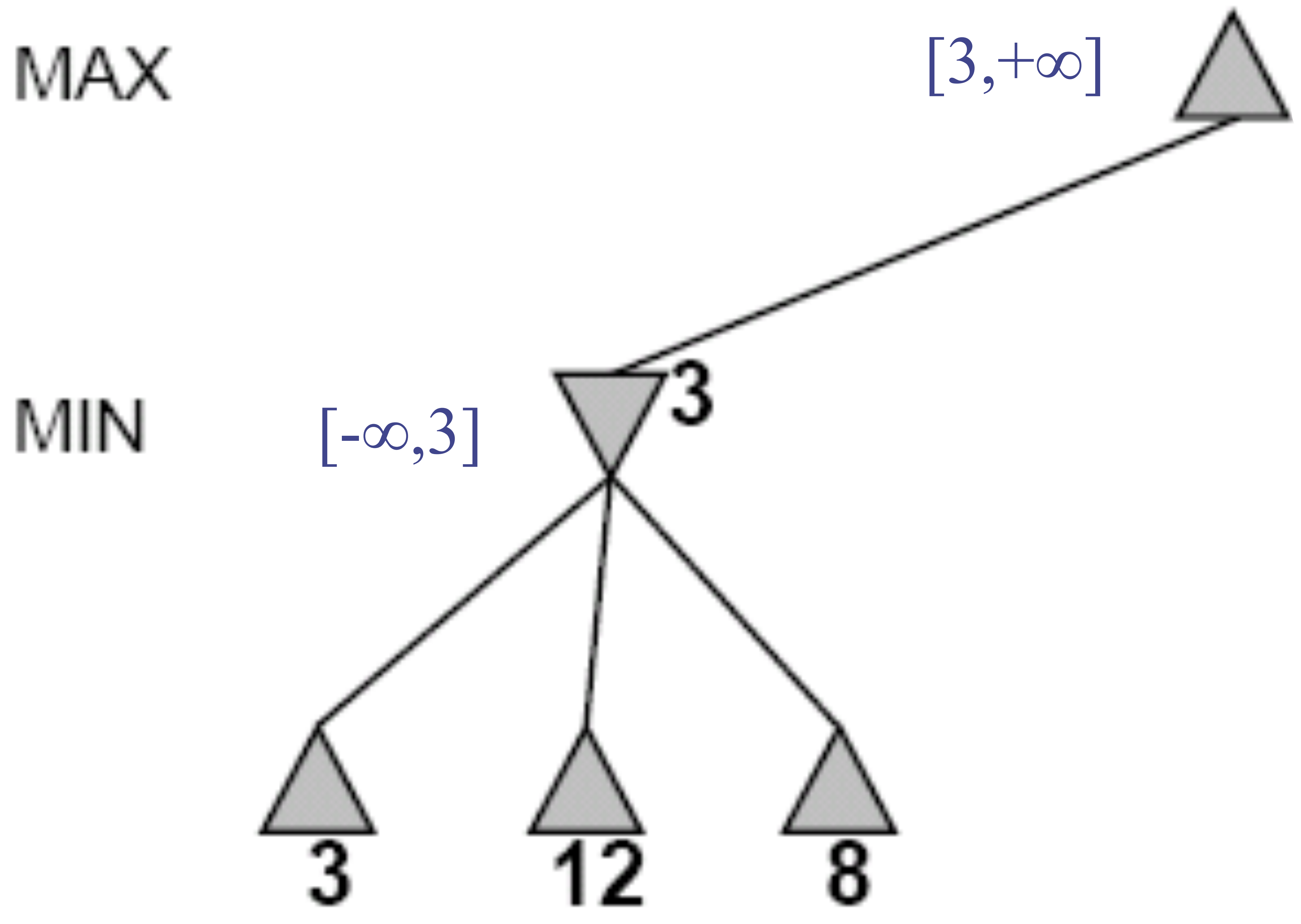
α: best already explored path along path to root for maximer
β: best already explored path along path to root for minimizer

Alpha-Beta Example (continued)



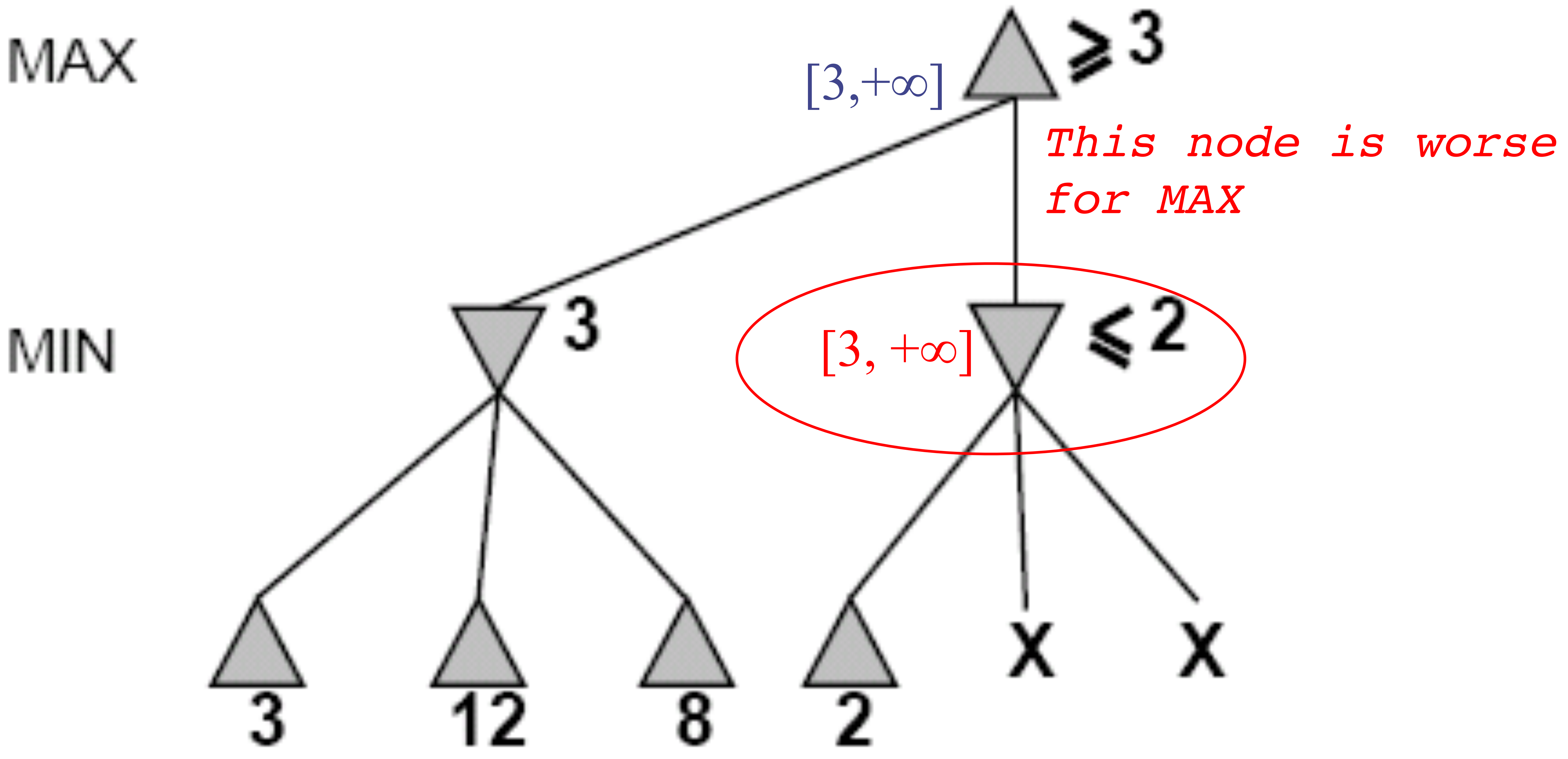
α: best already explored path along path to root for maximer
β: best already explored path along path to root for minimizer

Alpha-Beta Example (continued)



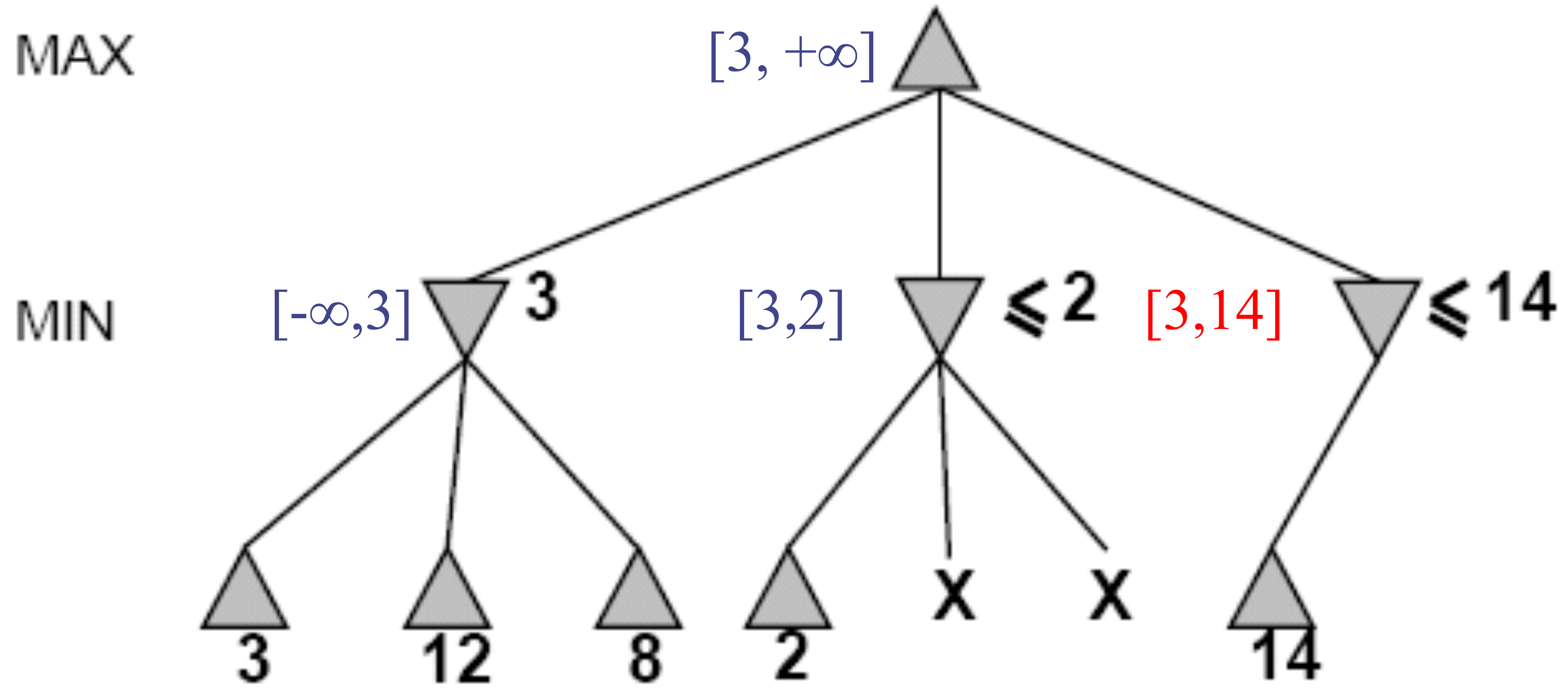
α: best already explored path along path to root for maximer
β: best already explored path along path to root for minimizer

Alpha-Beta Example (continued)



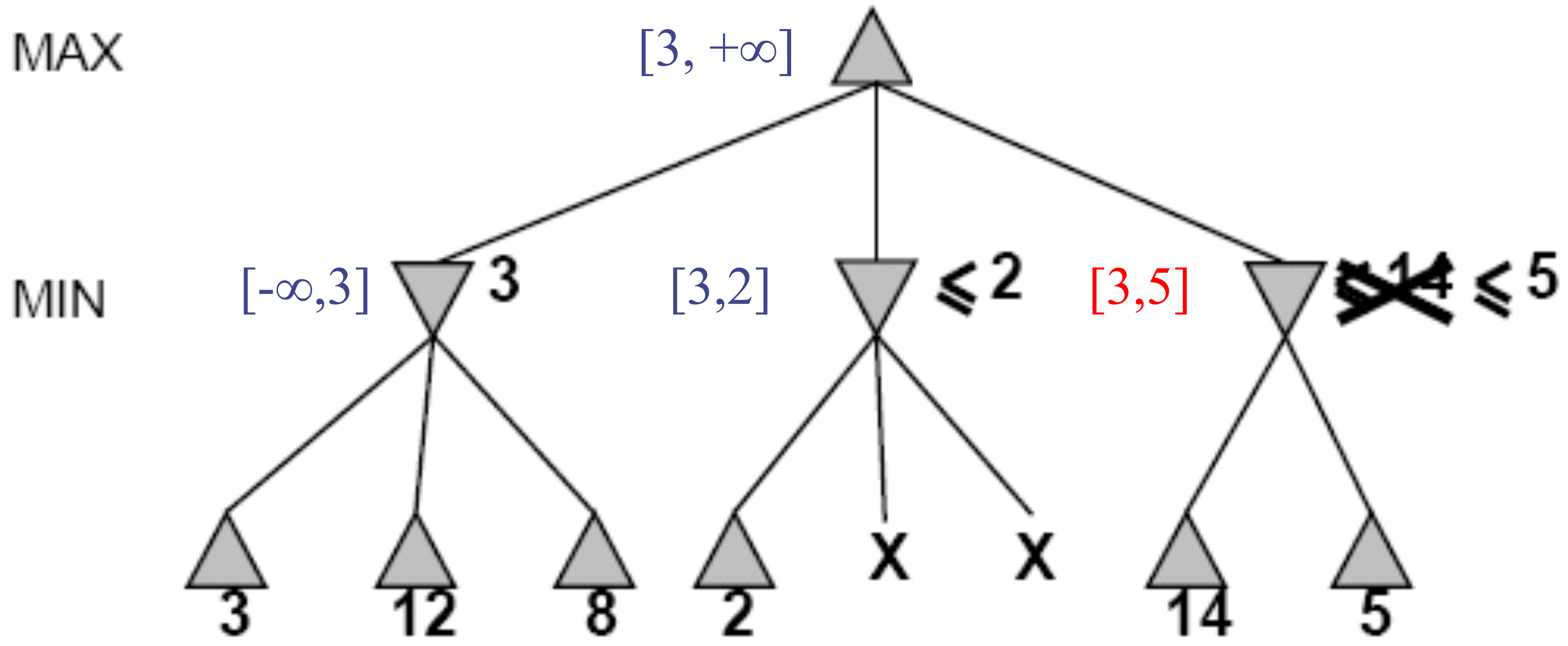
α: best already explored path along path to root for maximer
β: best already explored path along path to root for minimizer

Alpha-Beta Example (continued)



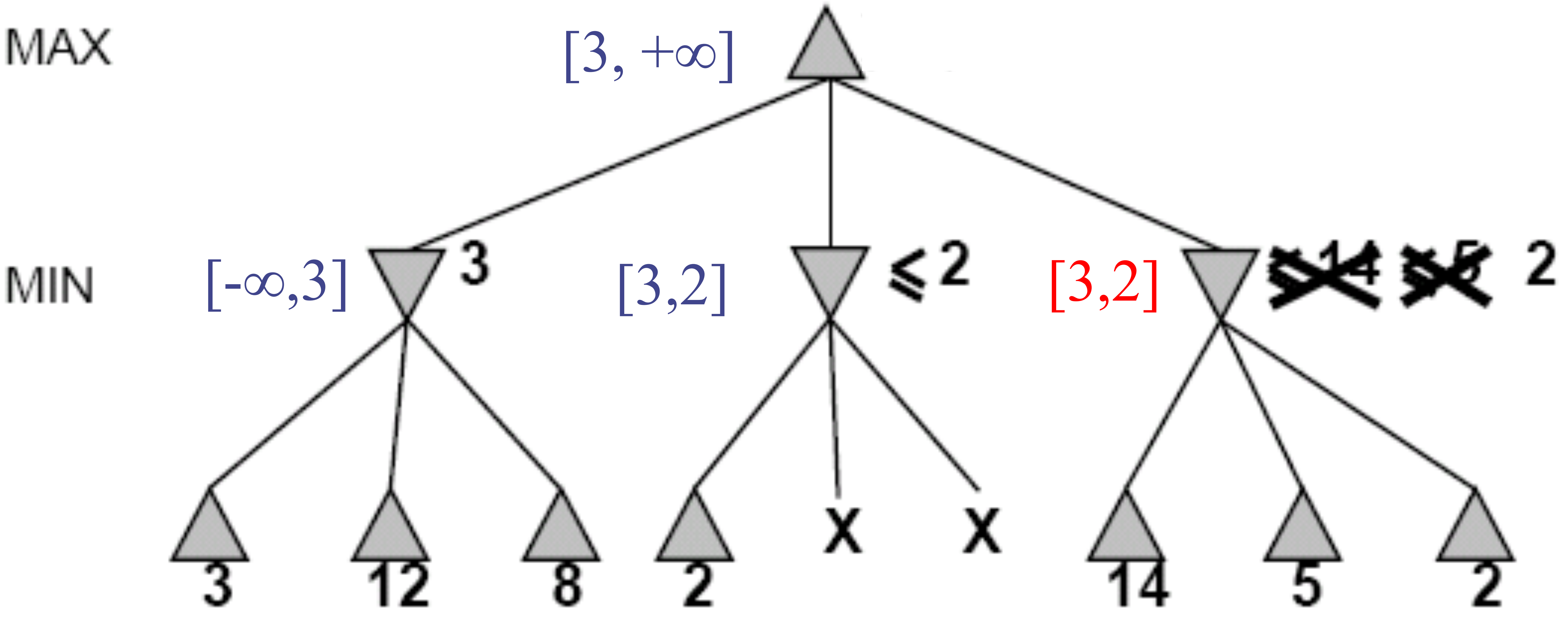
α: best already explored path along path to root for maximer
β: best already explored path along path to root for minimizer

Alpha-Beta Example (continued)



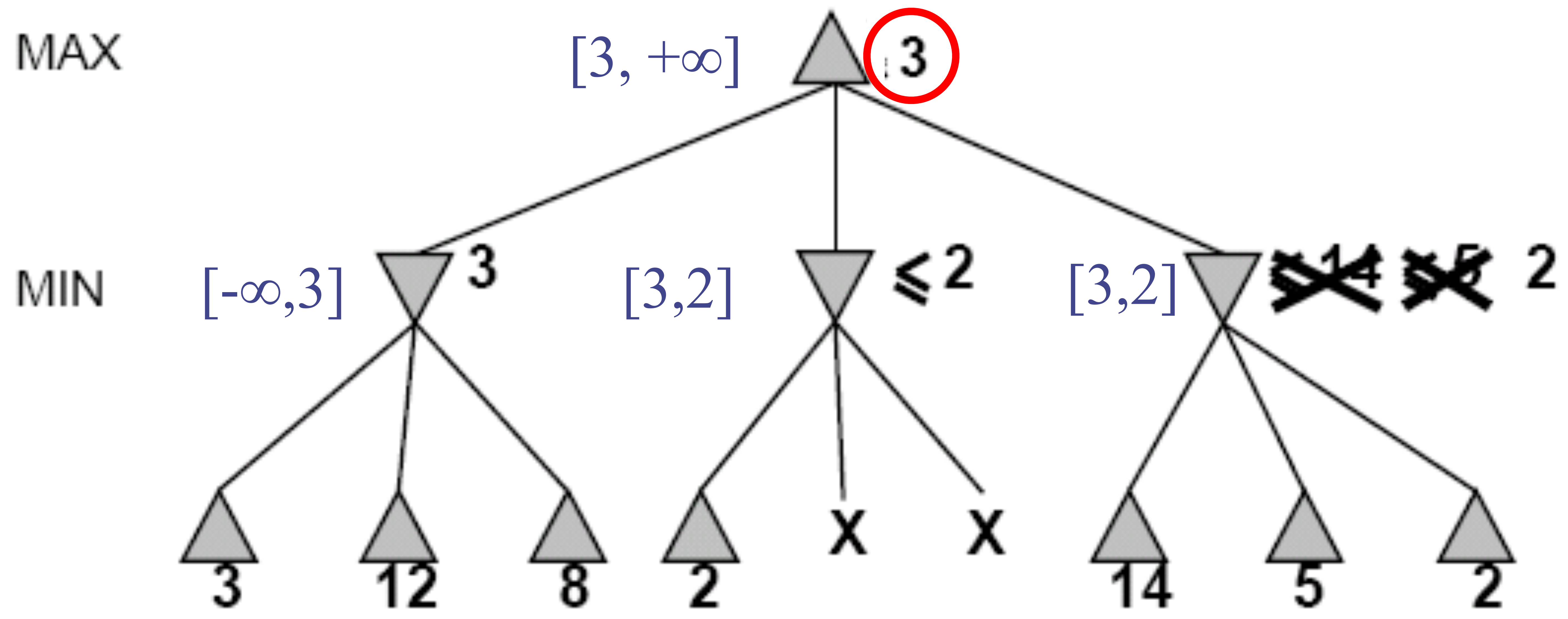
α: best already explored path along path to root for maximer
β: best already explored path along path to root for minimizer

Alpha-Beta Example (continued)

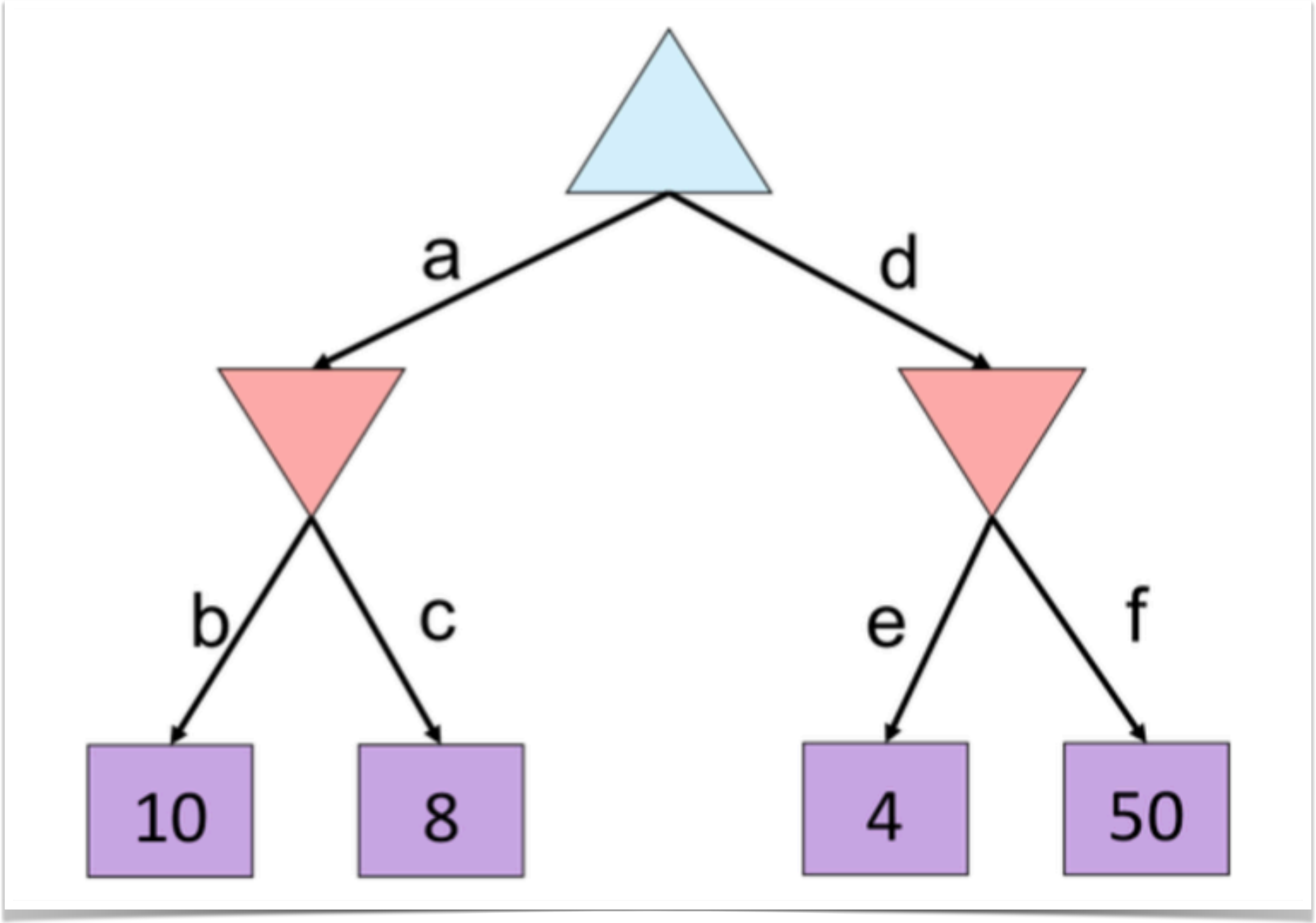


α: best already explored path along path to root for maximer
β: best already explored path along path to root for minimizer

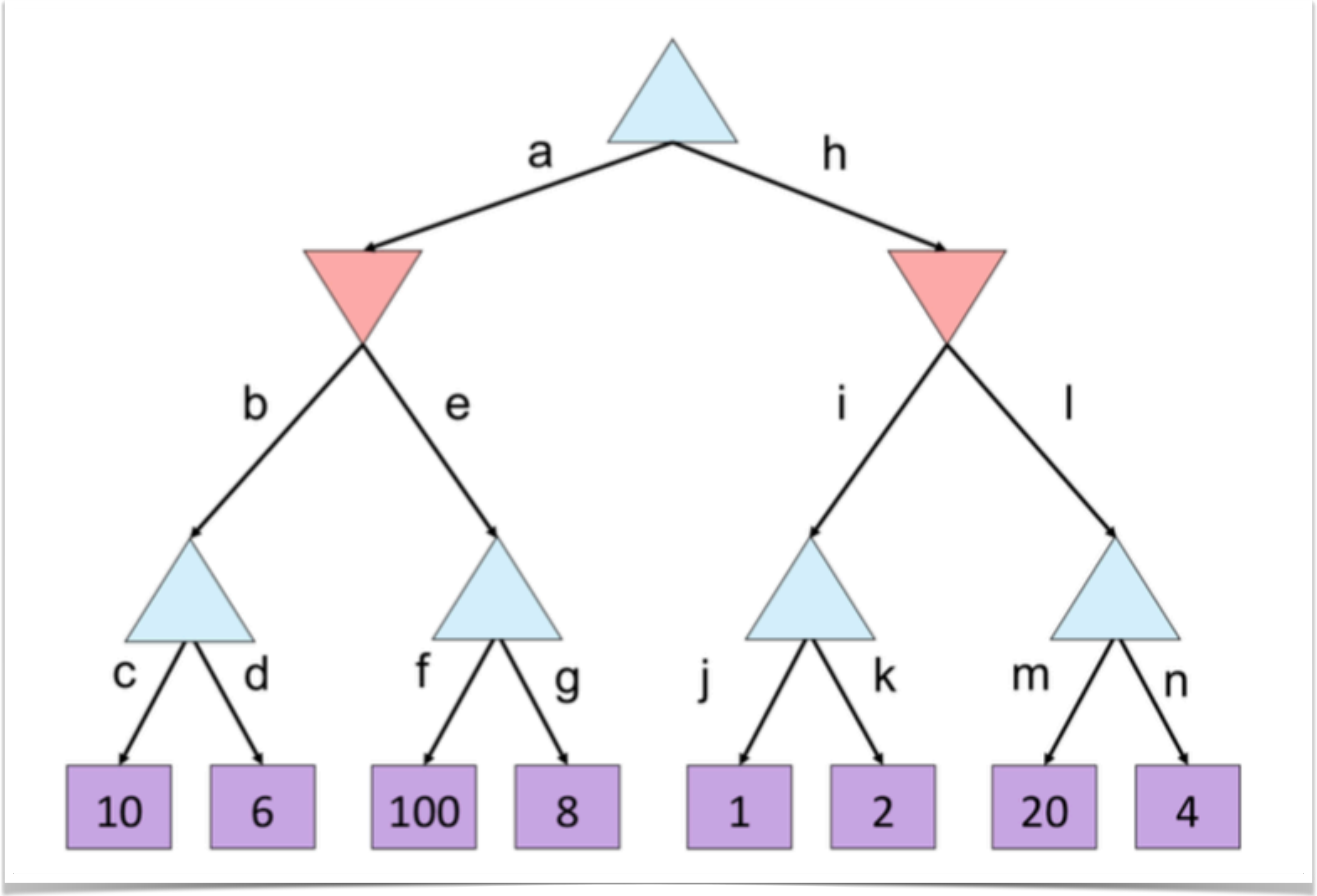
Alpha-Beta Example (continued)



Alpha-Beta Practice 1



Alpha-Beta Practice 2



Recap

- Game theory
  - Adversarial games
    - Minimax algorithm and alpha-beta pruning
- Next lecture
  - Stochastic games
    - Expectimax search algorithm