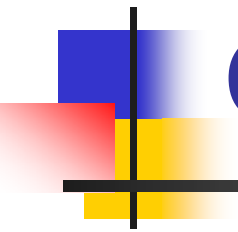


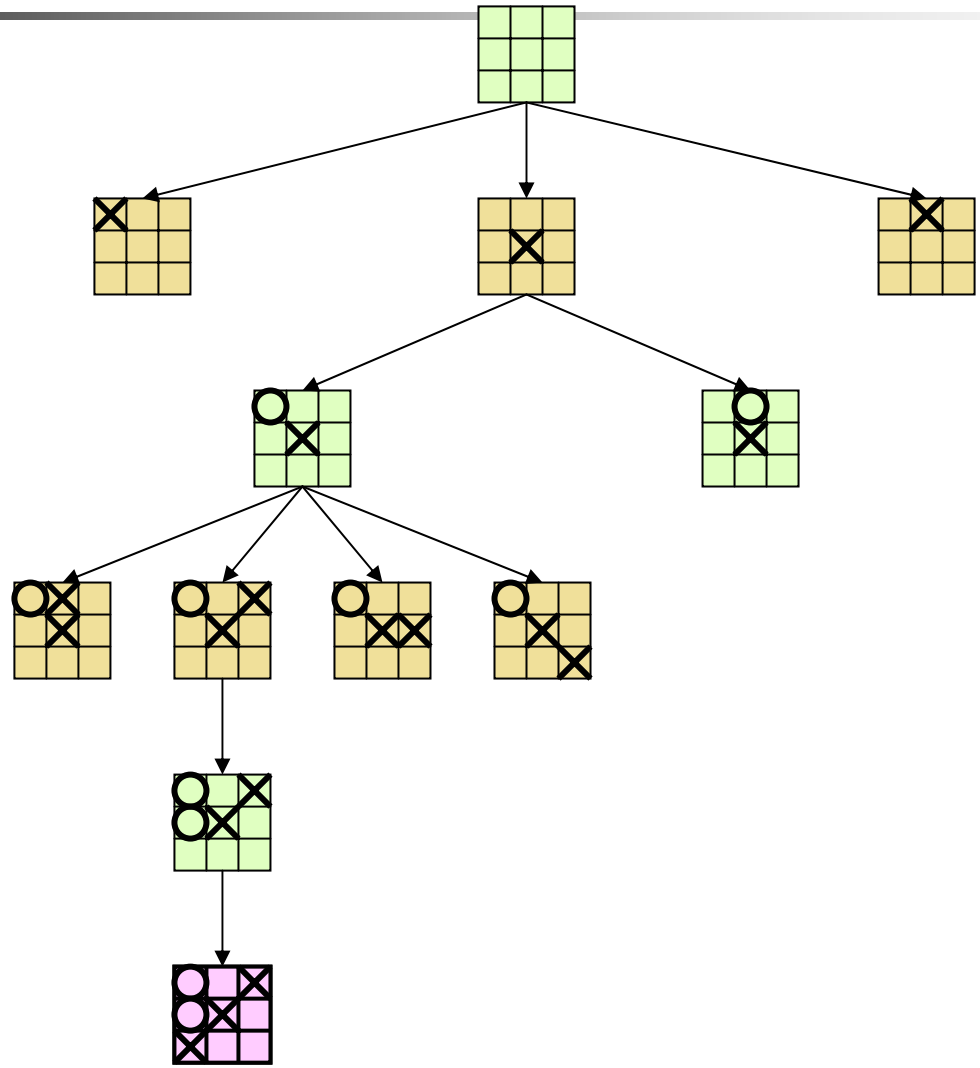
Game Tree Search

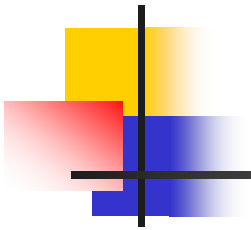
based on D Lin and Jean-Claude Latombe's notes

Types of Games

	deterministic	chance
perfect information	chess, checkers, go, othello	backgammon monopoly
imperfect information		bridge, poker, scrabble nuclear war

Partial Game Search Tree for Tic-Tac-Toe





But… in general the search tree is too big to make it possible to reach the terminal states!

Examples:

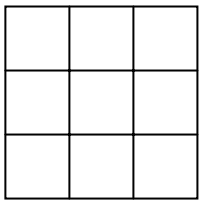
- Checkers: $\sim 10^{40}$ nodes
- Chess: $\sim 10^{120}$ nodes

Evaluation Function of a State

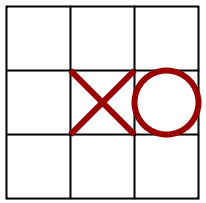
- e(s) = $+\infty$ if s is a win for MAX
- e(s) = $-\infty$ if s is a win for MIN
- e(s) = a measure of how "favorable" is s for MAX
  - > 0 if s is considered favorable to MAX
  - < 0 otherwise

Example: Tic-Tac-Toe

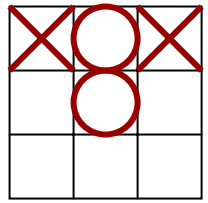
e(s) = number of rows, columns, and diagonals open for MAX
- number of rows, columns, and diagonals open for MIN



8-8 = 0



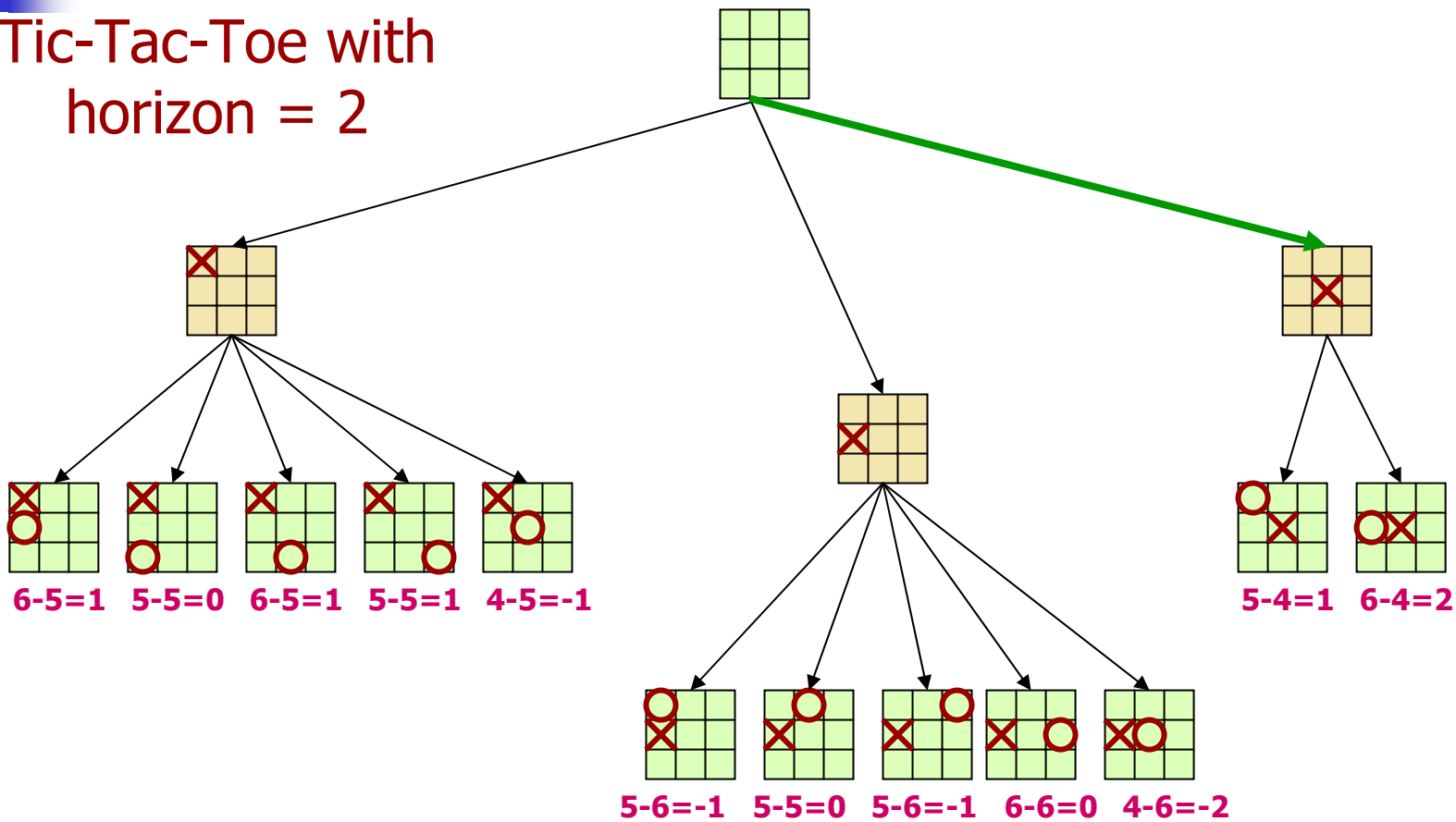
6-4 = 2

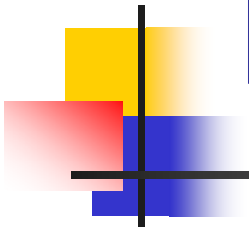


3-3 = 0

Example

Tic-Tac-Toe with horizon = 2

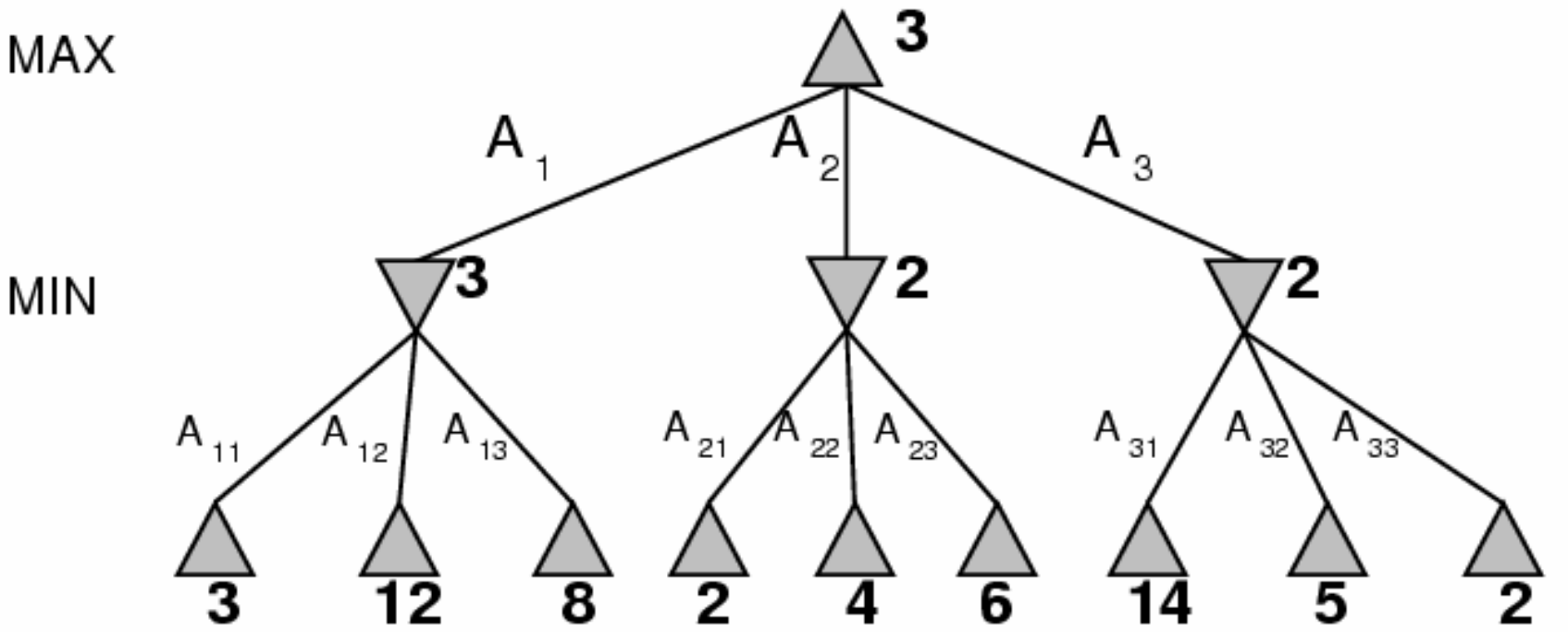




Minimax

- Perfect play for deterministic, perfect-information games
- Idea: choose move leading to position with highest minimax value
  - best achievable payoff against best play

Example: a 2-ply game



Minimax Algorithm

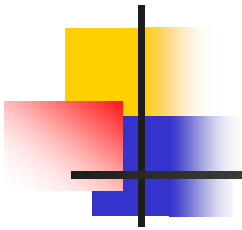
function Minimax-Decision(*game*) **returns** *an operator*

```
for each op in Operators[game] do
    Value[op] ← Minimax-Value(Apply(op, game), game)
end
return the op with the highest Value[op]
```

---

function Minimax-Value(*state*, *game*) **returns** *a utility value*

```
if Terminal-Test[game](state) then
    return Utility[game](state)
else if max is to move in state then
    return the highest Minimax-Value of Successors(state)
else
    return the lowest Minimax-Value of Successors(state)
```



Minimax

- Does it work in practice?

$$b^m = 10^6, \quad b = 35 \quad \Rightarrow \quad m = 4$$

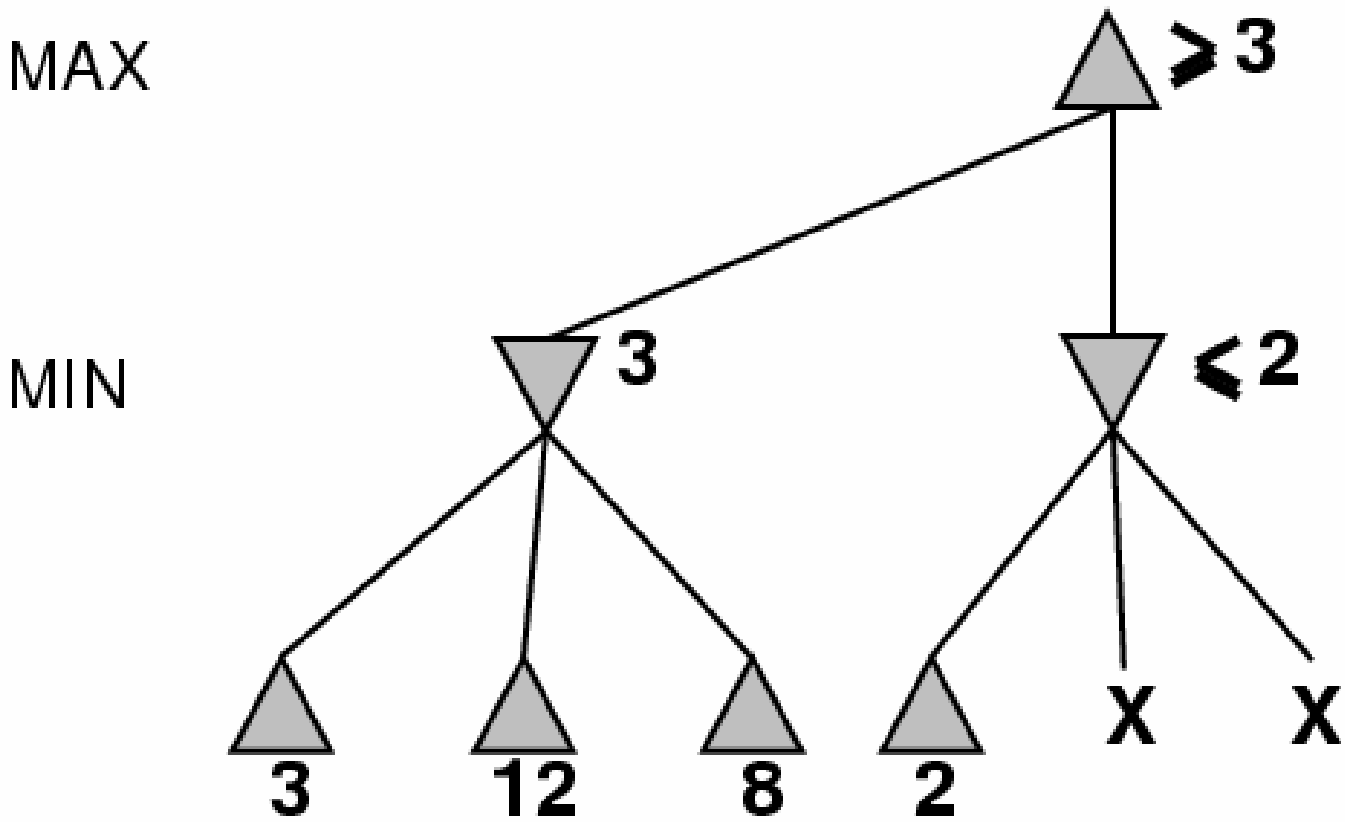
4-ply lookahead is a hopeless chess player!

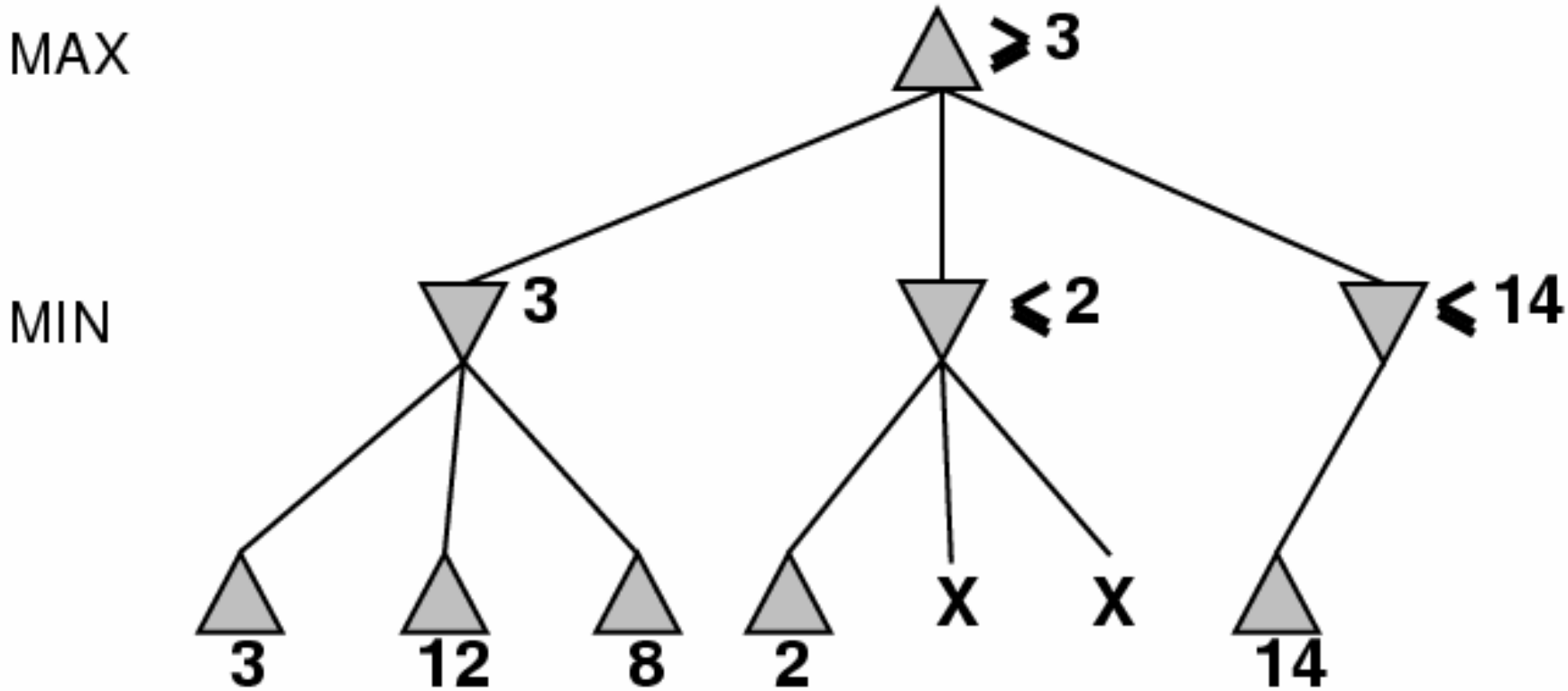
- 4-ply $\approx$ human novice

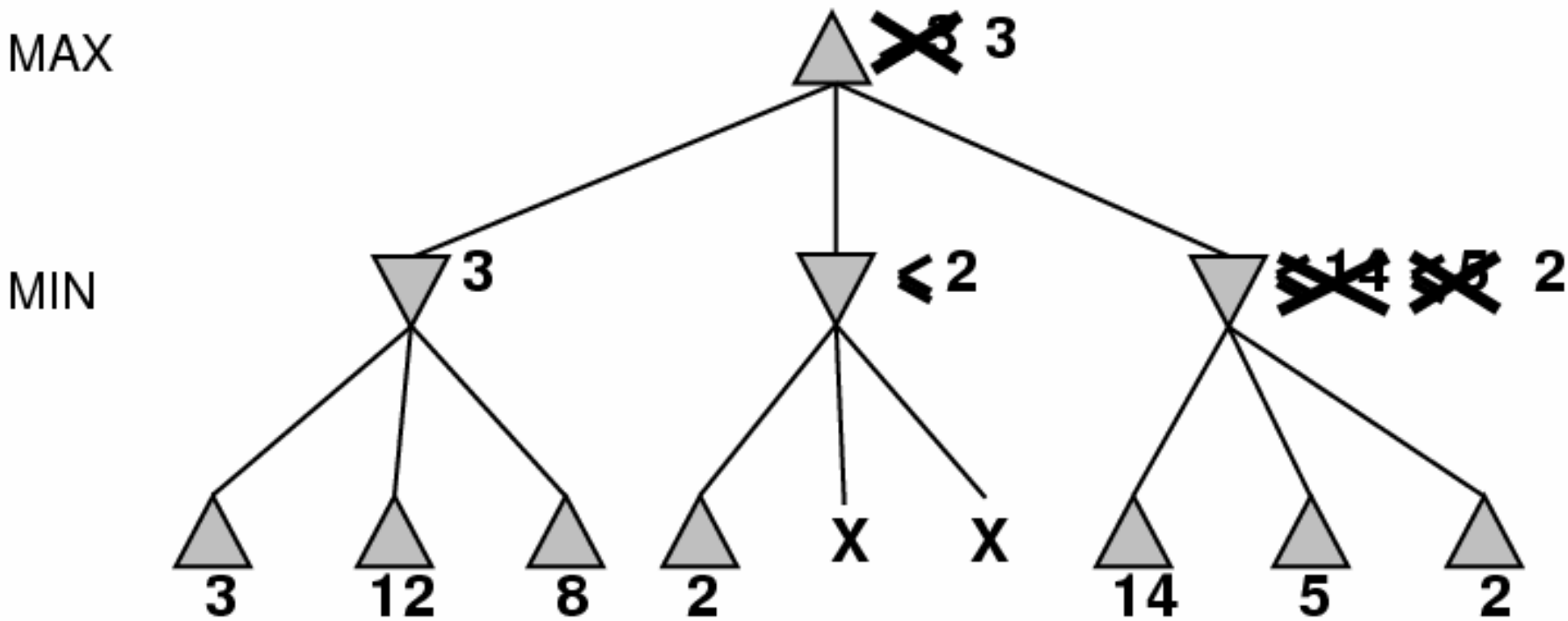
  8-ply $\approx$ typical PC, human master

  12-ply $\approx$ Deep Blue, Kasparov

α-β Pruning



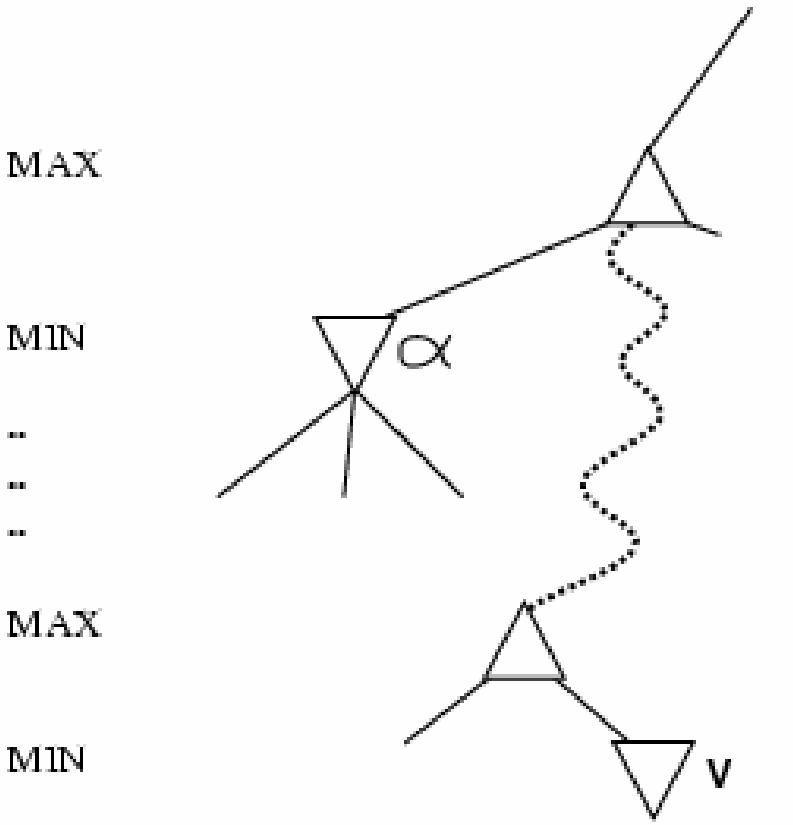

MAX
MIN
≥ 3
3
≤ 2
≤ 14
3
12
8
2
X
X
14


MAX
MIN
3
3
2
2
14
5
2
3
12
8
2
X
X
14
5
2

Properties of α-β Search

- Pruning does not affect final result
- Good move ordering improves effectiveness of pruning
- With "perfect ordering":
  - time complexity = $O(b^{m/2})$
  - doubles depth of search
  - can easily reach depth 8
    - play good chess!
- Shows value of "metareasoning":
  - Reasoning about which computations are relevant

Why is it called α-β?



- α = best value (for max) found so far, off the current path
- If V is worse than α, max will avoid it
  - prune that branch
- Define β similarly for min