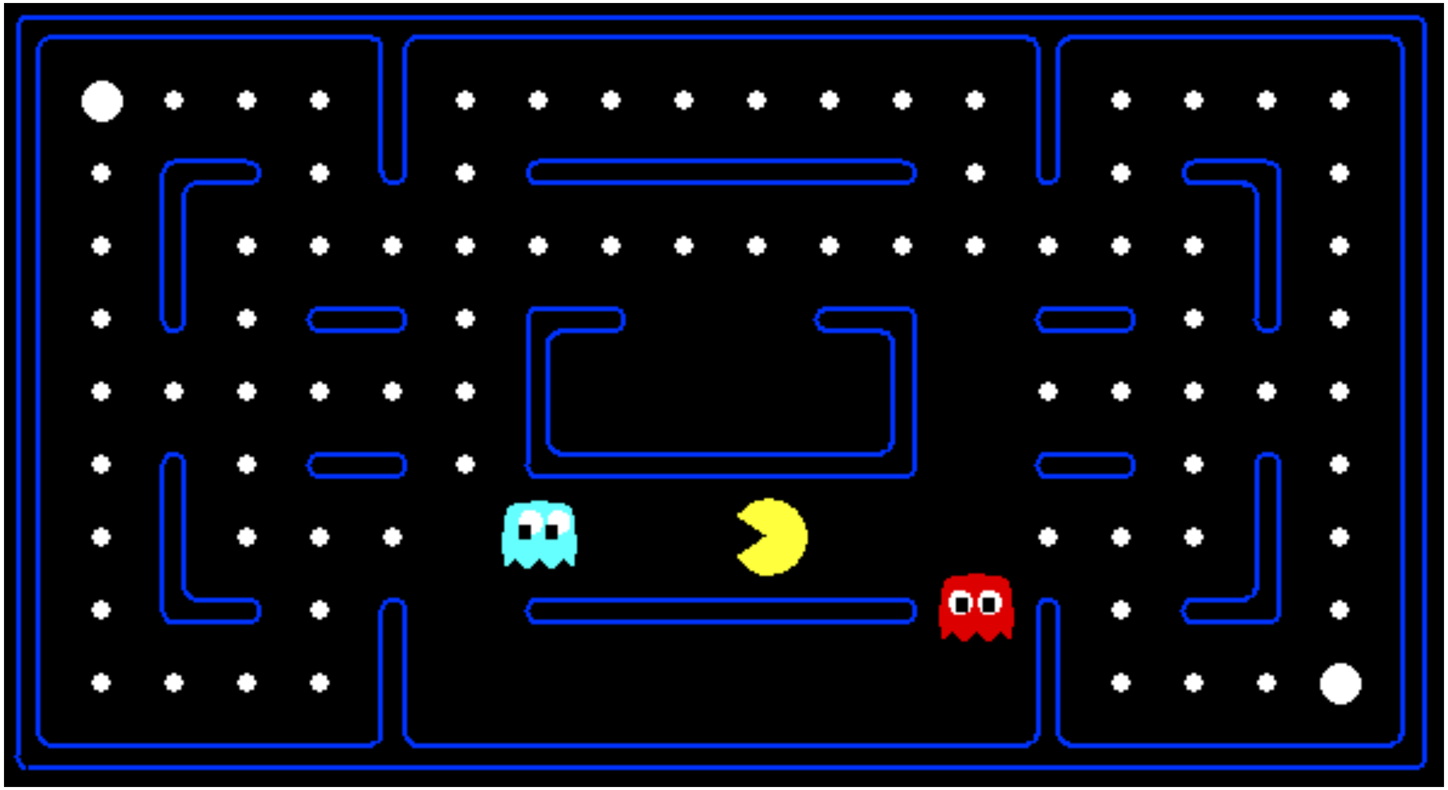
Adversarial Search

Chris Amato
Northeastern University

Some images and slides are used from: Rob Platt, CS188 UC Berkeley, AIMA

Adversarial search



How should Pac-Man move when there are ghosts?

What is adversarial search?

<u>Adversarial search</u>: planning used to play a game such as chess or checkers

– algorithms are similar to graph search except that we plan under the assumption that our opponent will maximize his own advantage...

Some types of games

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

Some types of games

Chess	Solved/unsolved?
Checkers	Solved/unsolved?
Tic-tac-toe	Solved/unsolved?
Go	Solved/unsolved?

Outcome of game can be predicted
from any initial state assuming
both players play perfectly

Examples of adversarial search

Chess	Unsolved
Checkers	Solved
Tic-tac-toe	Solved
Go	Unsolved

Outcome of game can be predicted
from any initial state assuming
both players play perfectly

Examples of adversarial search

Chess	Unsolved	~10^40 states
Checkers	Solved	~10^20 states
Tic-tac-toe	Solved	Less than 9!=362k states
Go	Unsolved	?

Outcome of game can be predicted
from any initial state assuming
both players play perfectly

Different types of games

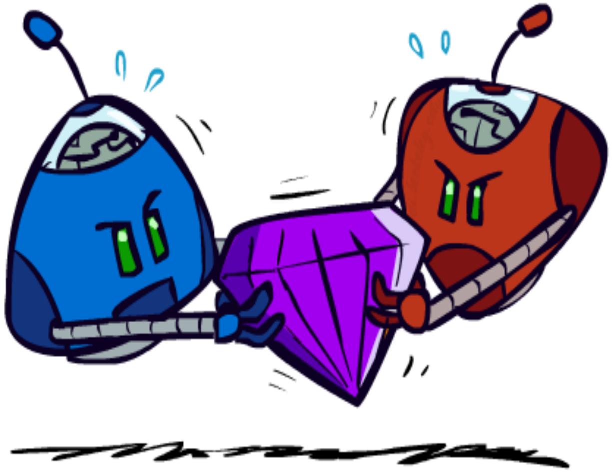
Deterministic / stochastic

Two player / multi player?

Zero-sum / non zero-sum

Perfect information / imperfect information

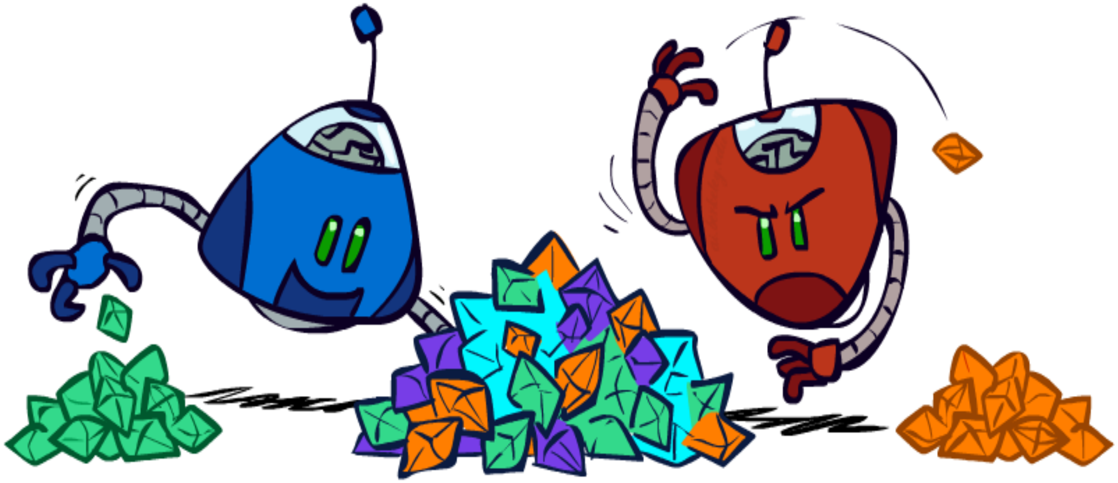
What is a zero-sum game?



Zero-Sum Games

- Agents have opposite utilities (values on outcomes)
- Lets us think of a single value that one maximizes and the other minimizes (for two player game

  $U_A = -U_B$)
- Adversarial, pure competition



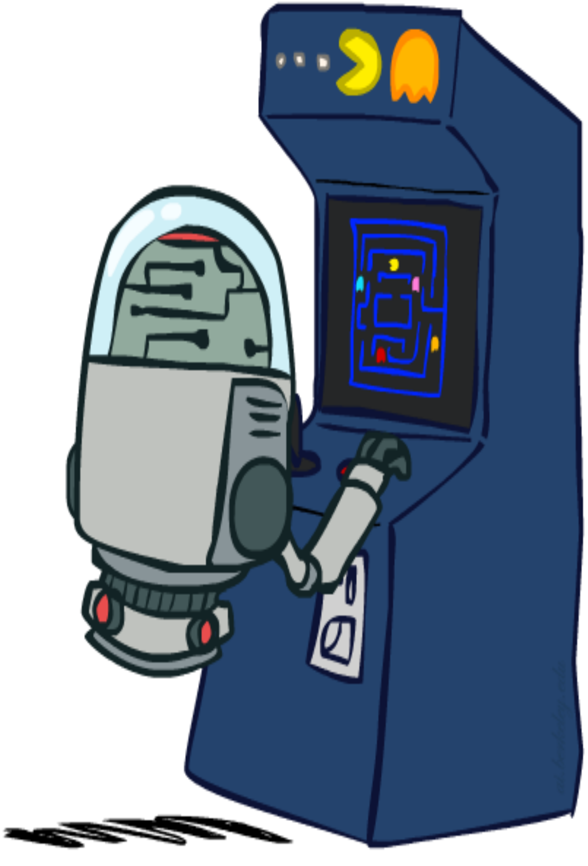
General Games

- Agents have independent utilities (values on outcomes)
- Cooperation, indifference, competition, and more are all possible
- More later on non-zero-sum games

Deterministic games

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: $S \times A \rightarrow S$
- Terminal Test: $S \rightarrow \{t,f\}$
- Terminal Utilities: $S \times P \rightarrow R$



Solution for a player is a policy: $S \rightarrow A$

Deterministic games

Many possible formalizations, one is:

States: S (start at s_0)

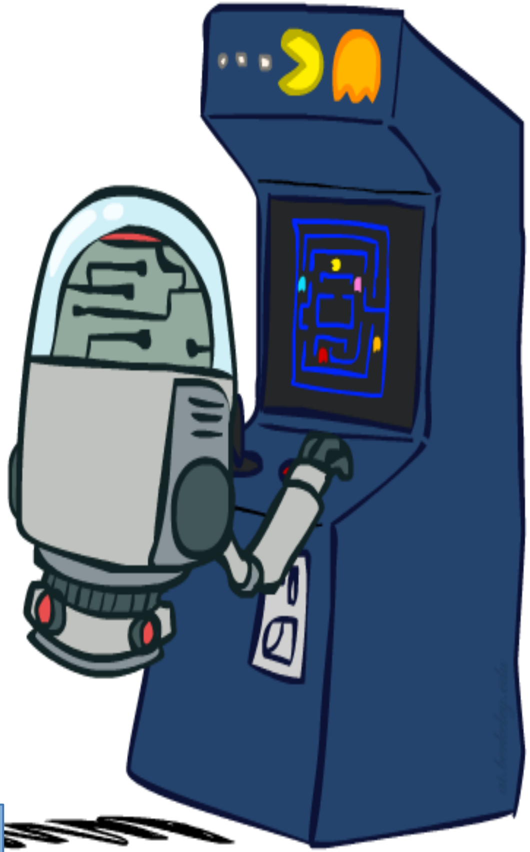
Players: $P=\{1...N\}$ (usually take turns)

Actions: A (may depend on player / state)

Transition Function: $S \times A \rightarrow S$

Terminal Test: $S \rightarrow \{t,f\}$

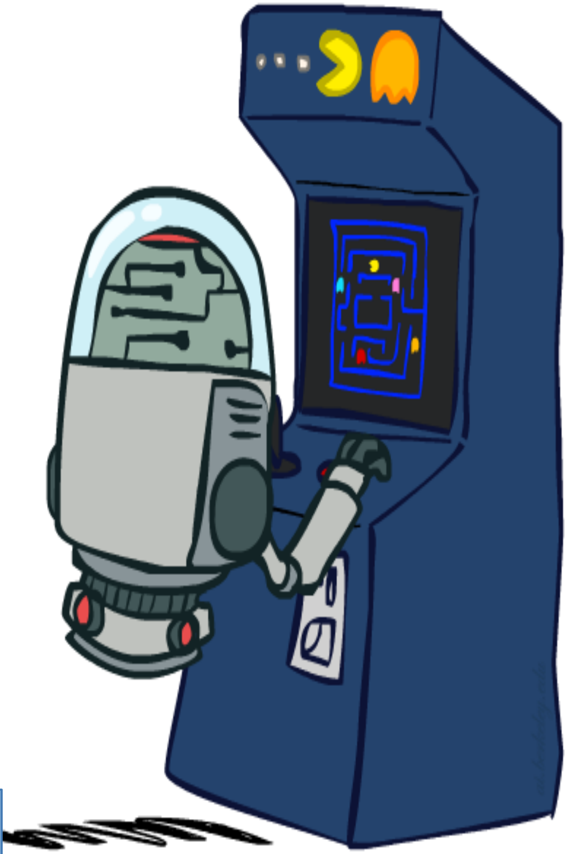
Terminal Utilities: $S \times P \rightarrow R$



Solution for a pla

How is this similar/different to the definition of a standard search problem?

Deterministic games



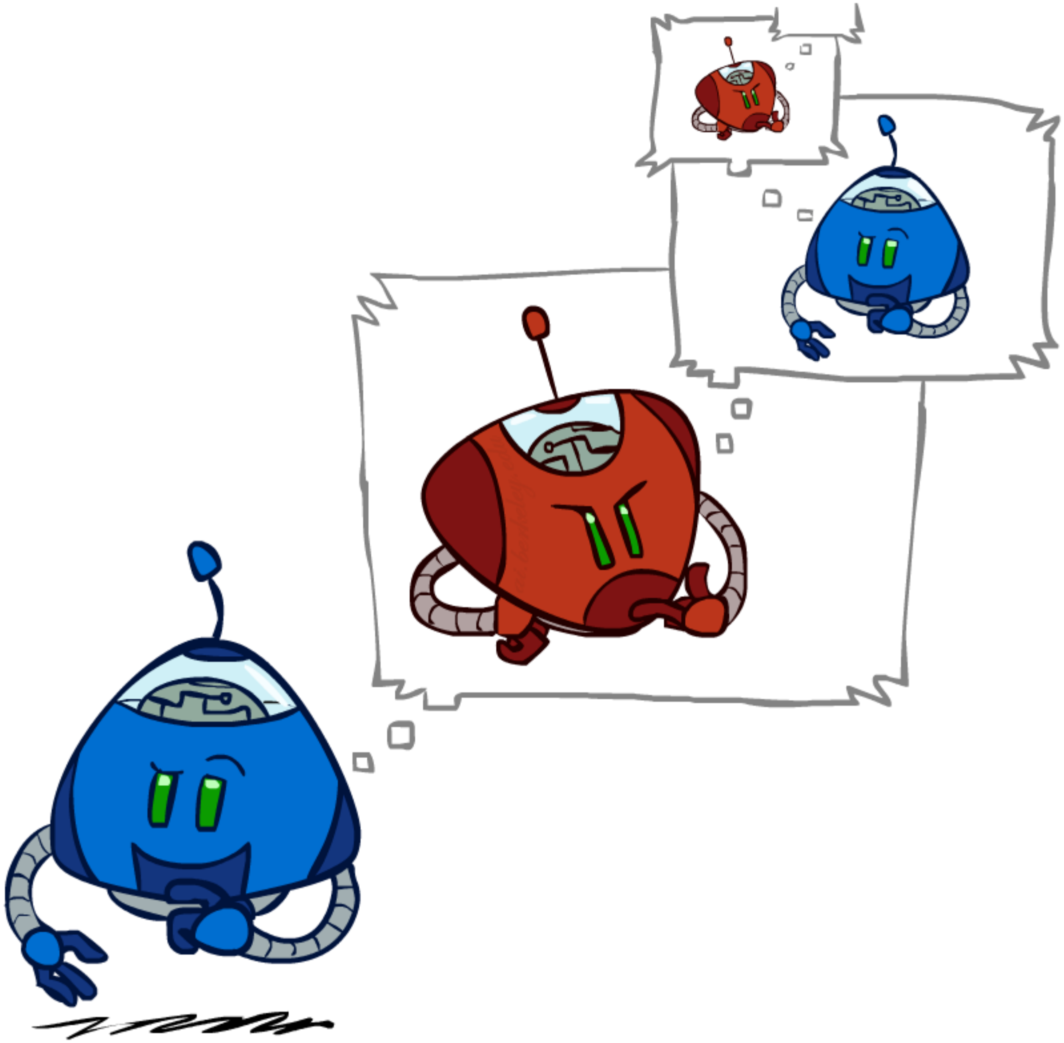
Many possible formalizations, one is:

- 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: $S \times P \to R$

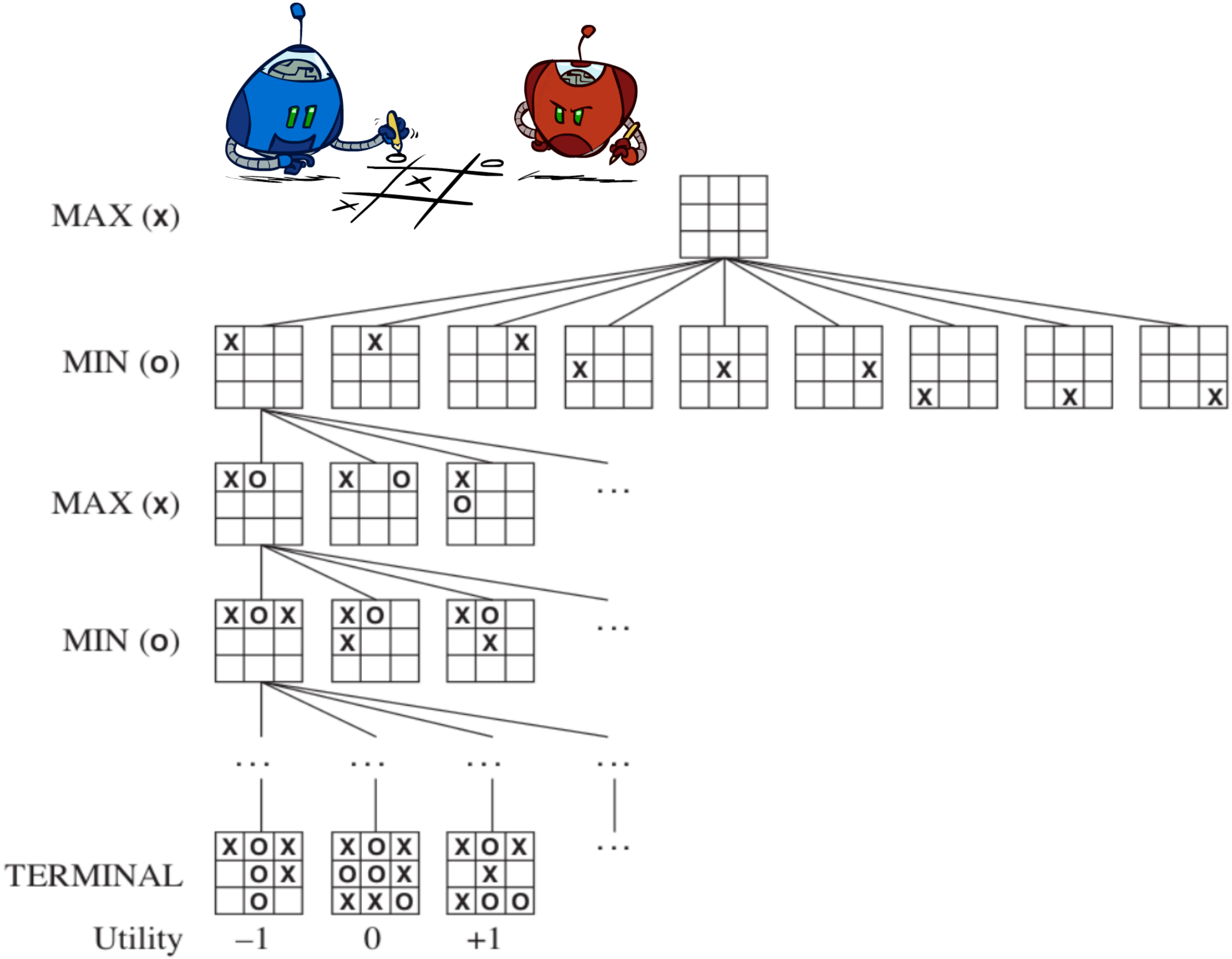
Solution for a play

How do we solve this problem?

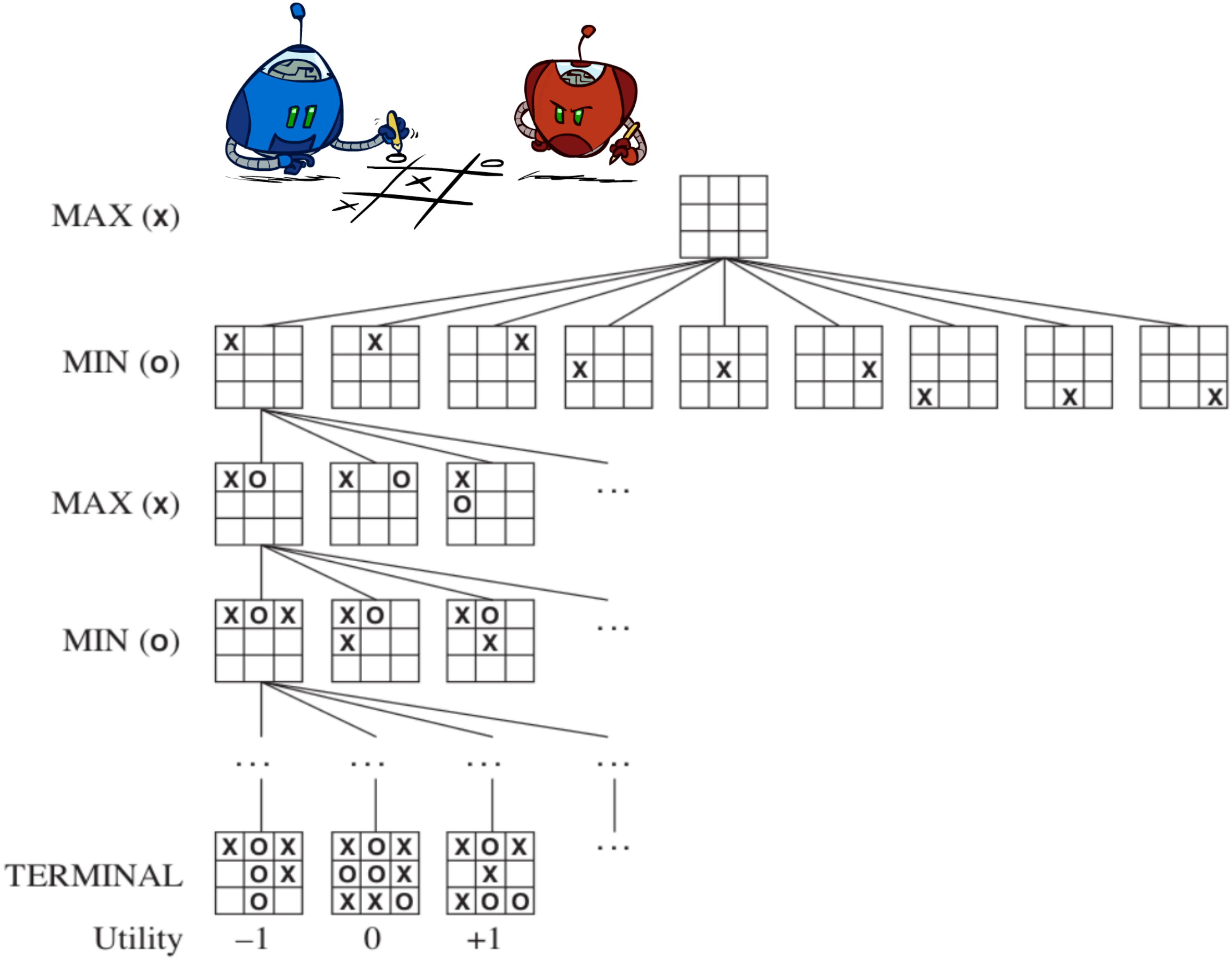
Adversarial search



This is a game tree for tic-tac-toe

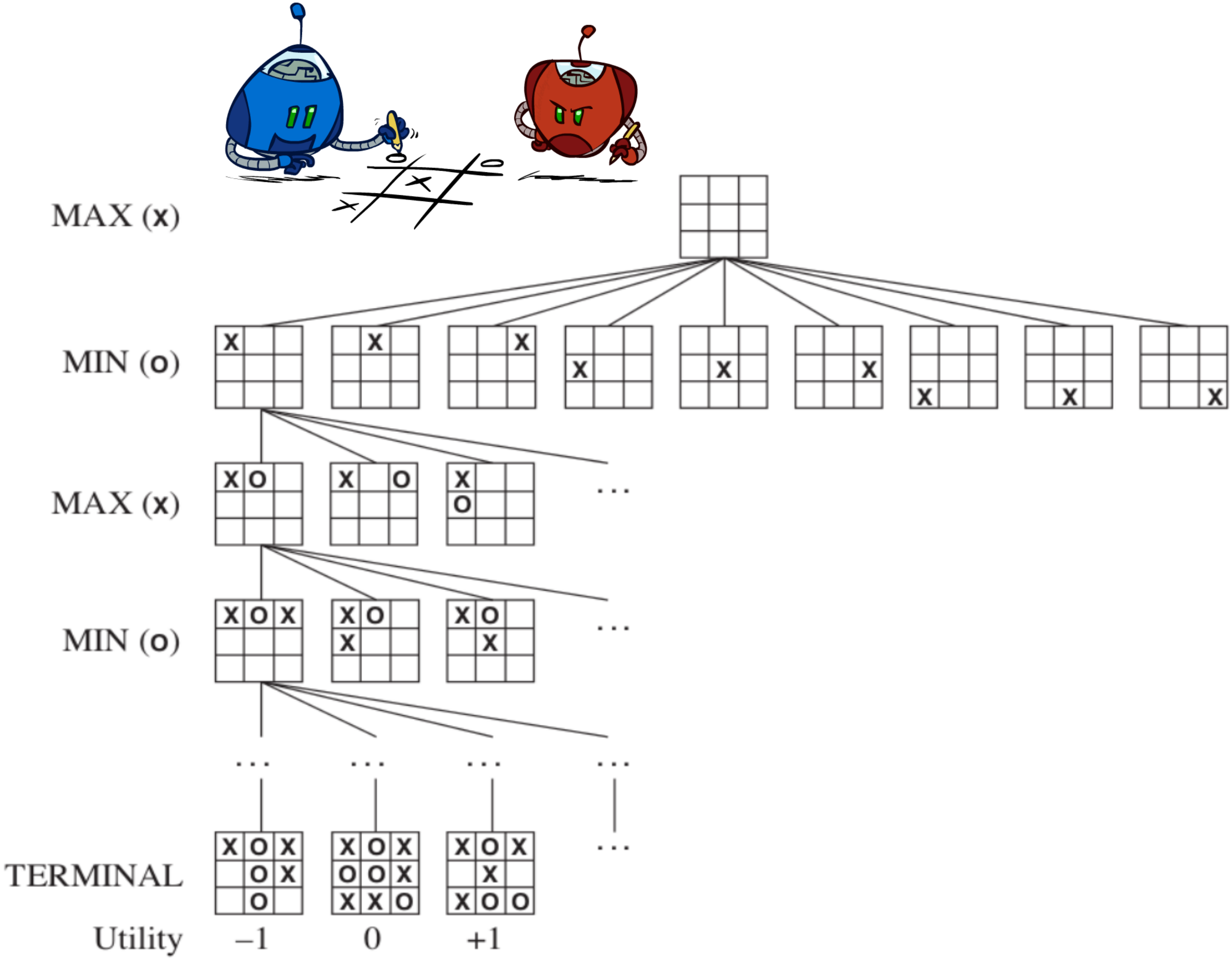


This is a game tree for tic-tac-toe



You

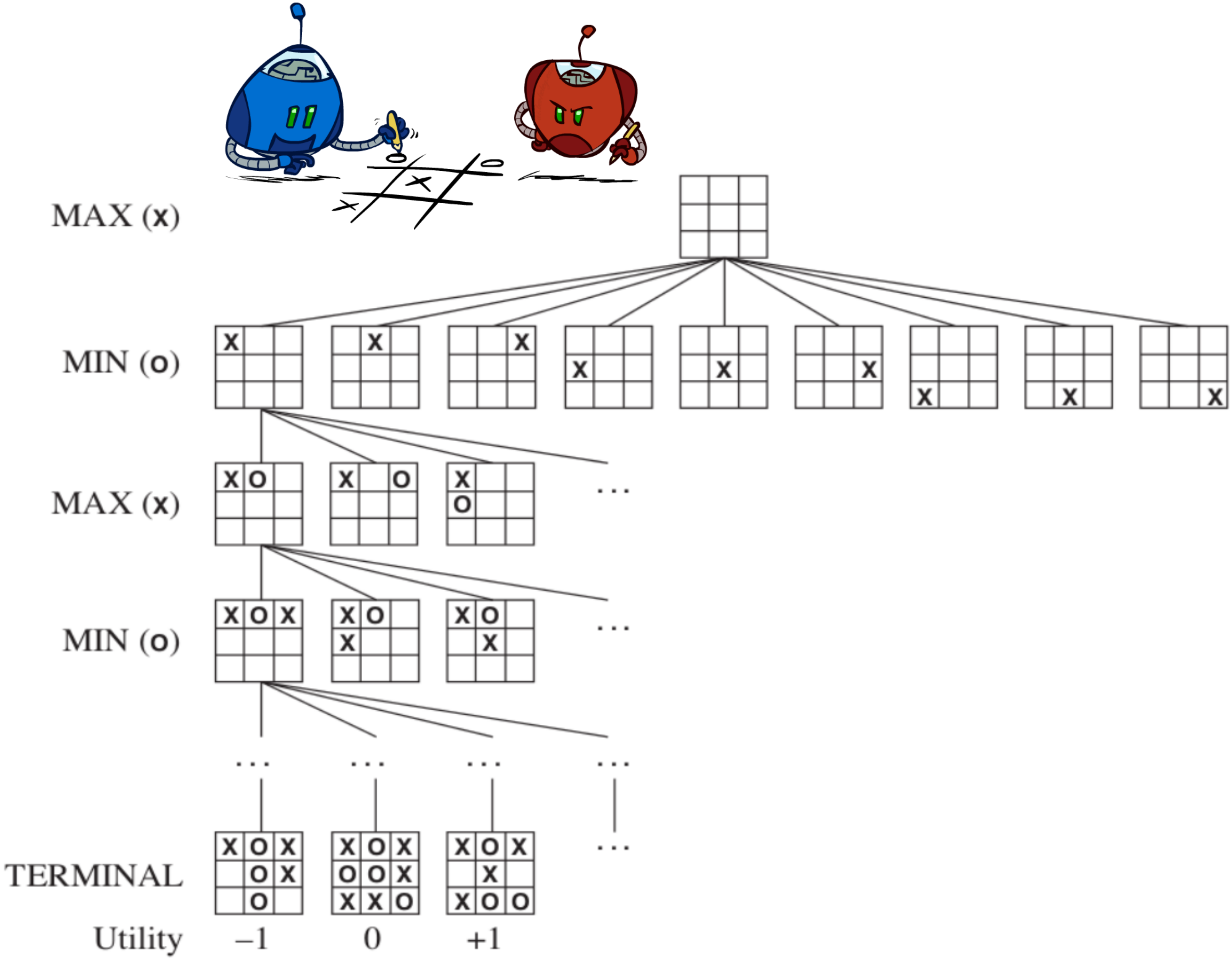
This is a game tree for tic-tac-toe



You

Them

This is a game tree for tic-tac-toe

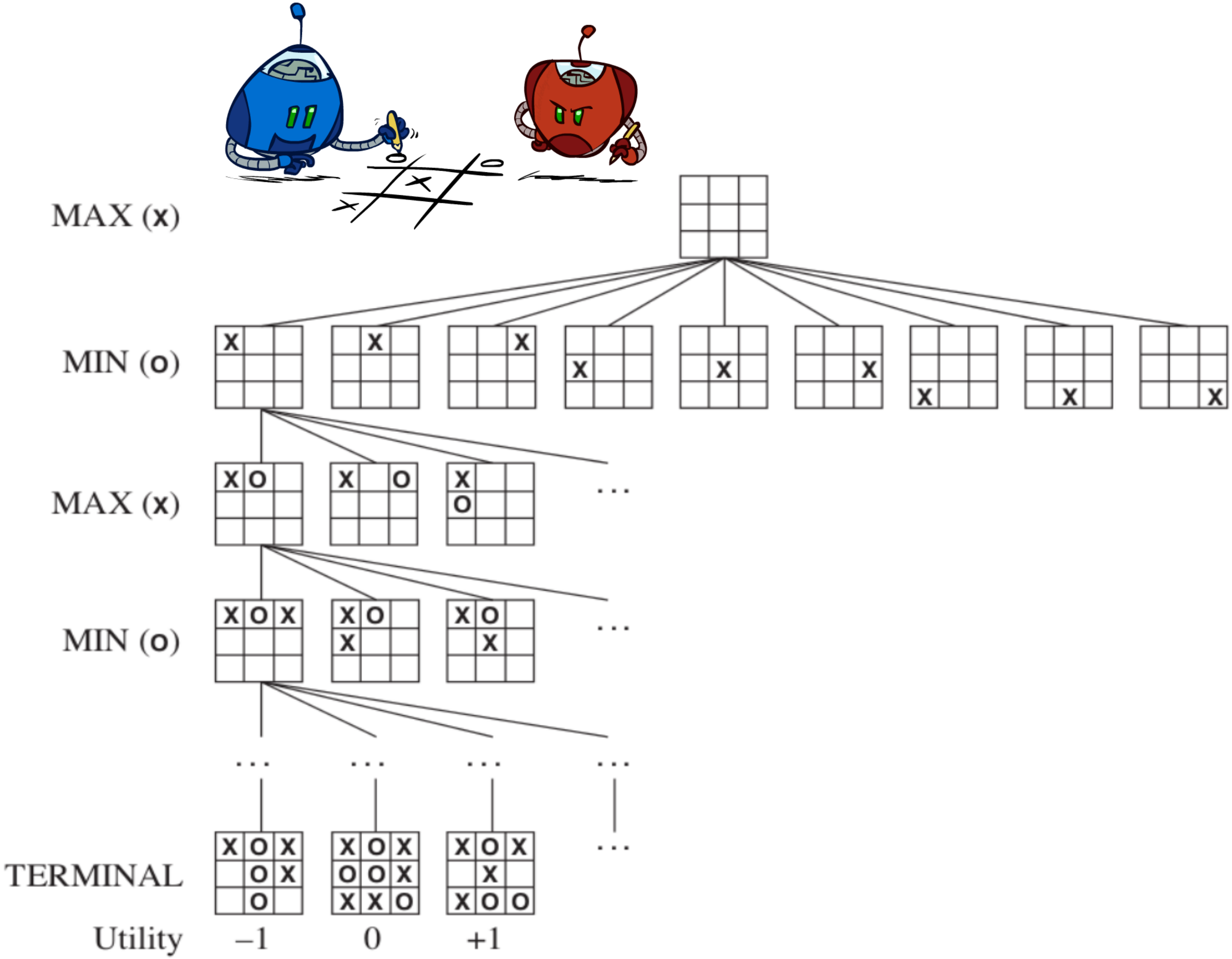


You

Them

You

This is a game tree for tic-tac-toe



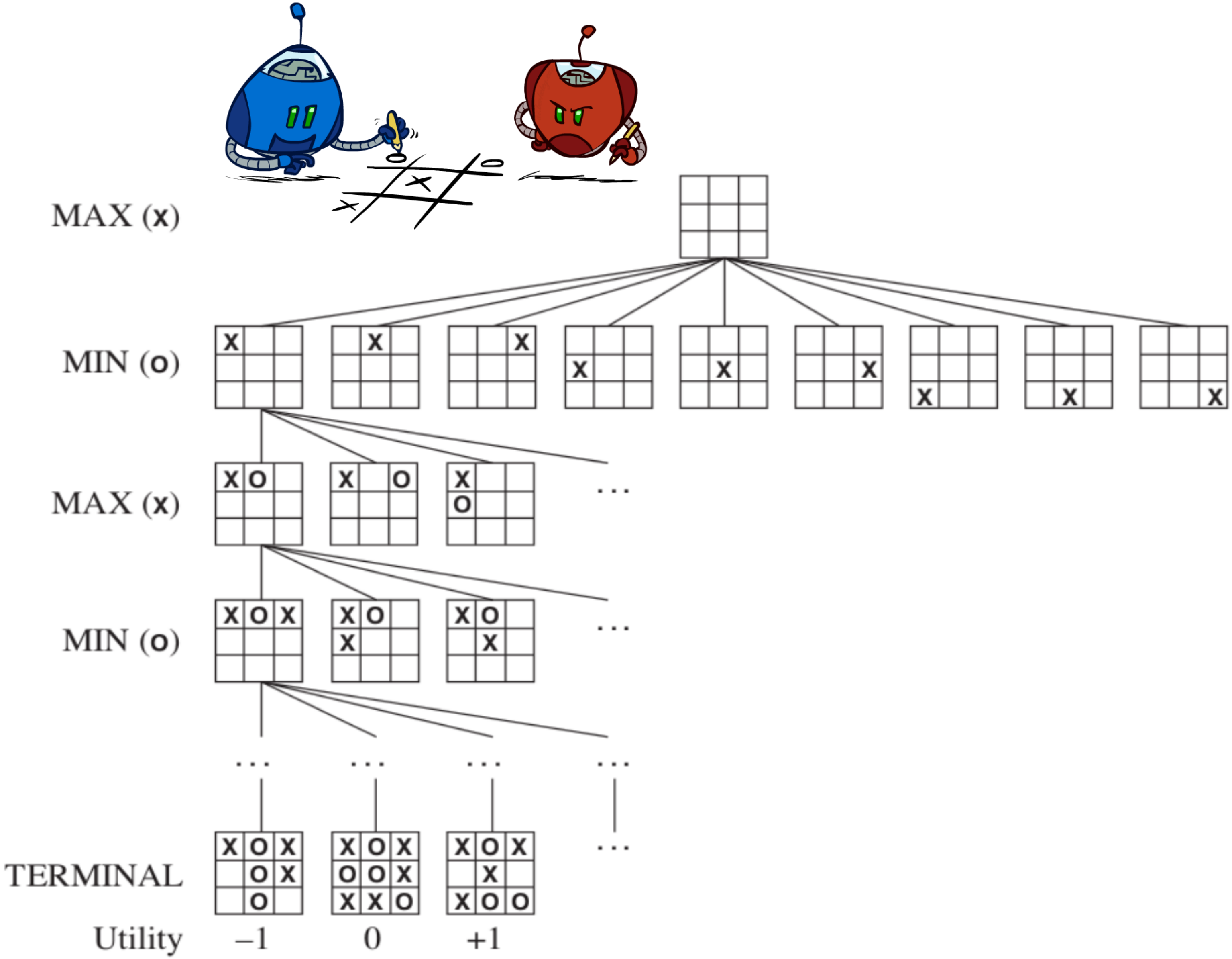
You

Them

You

Them

This is a game tree for tic-tac-toe



You

Them

You

Them

Utility

What is Minimax?

Consider a simple game:

1. you make a move
2. your opponent makes a move
3. game ends

What is Minimax?

Consider a simple game:

1. you make a move
2. your opponent makes a move
3. game ends

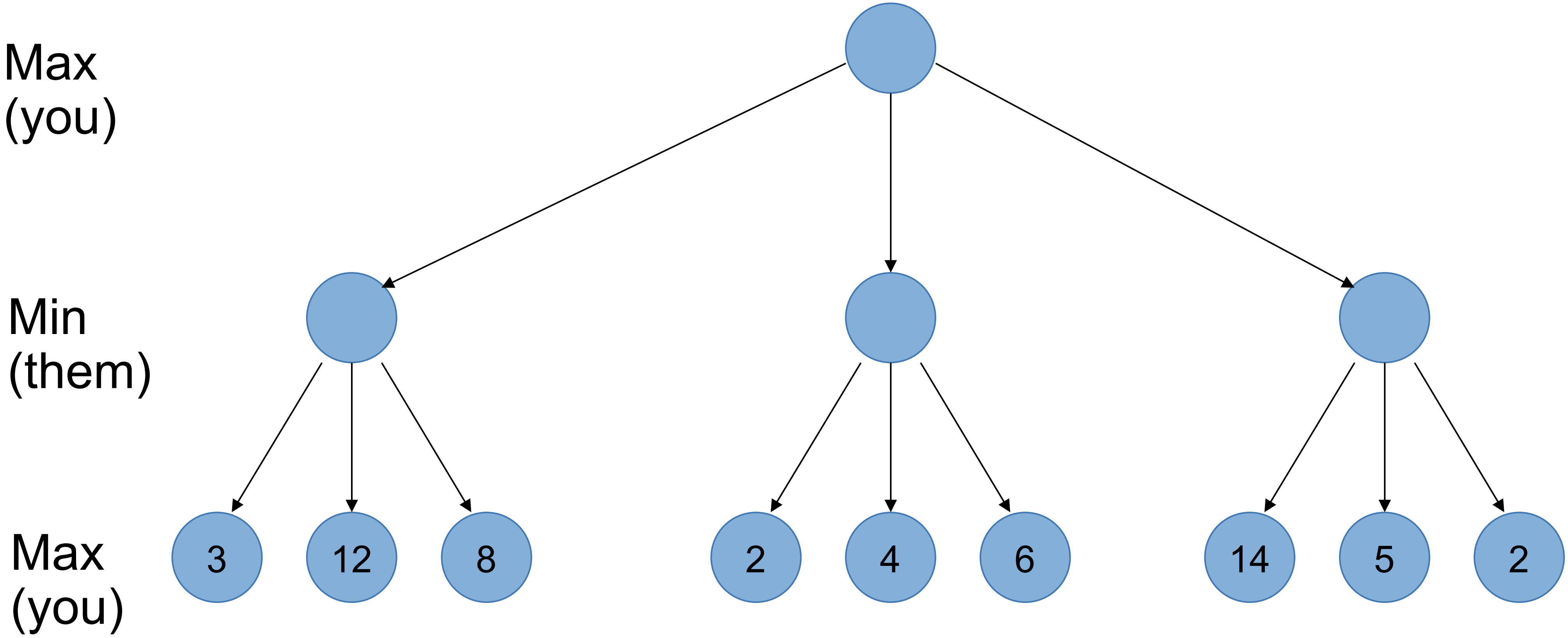
What does the minimax tree look like in this case?

What is Minimax?

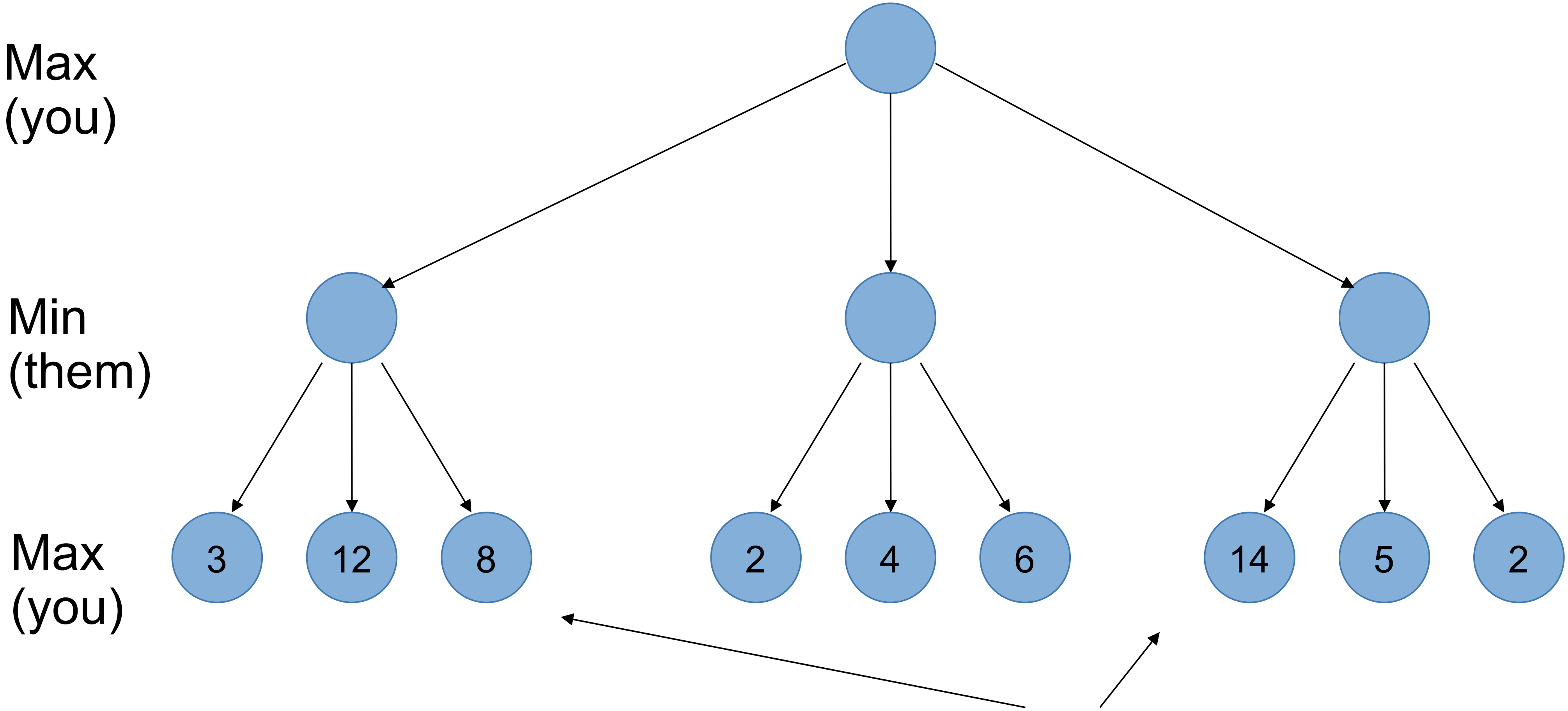
Consider a simple game:

1. you make a move
2. your opponent makes a move
3. game ends

What does the minimax tree look like in this case?



What is Minimax?



Max
(you)

Min
(them)

Max
(you)

These are terminal utilities
– assume we know what
these values are

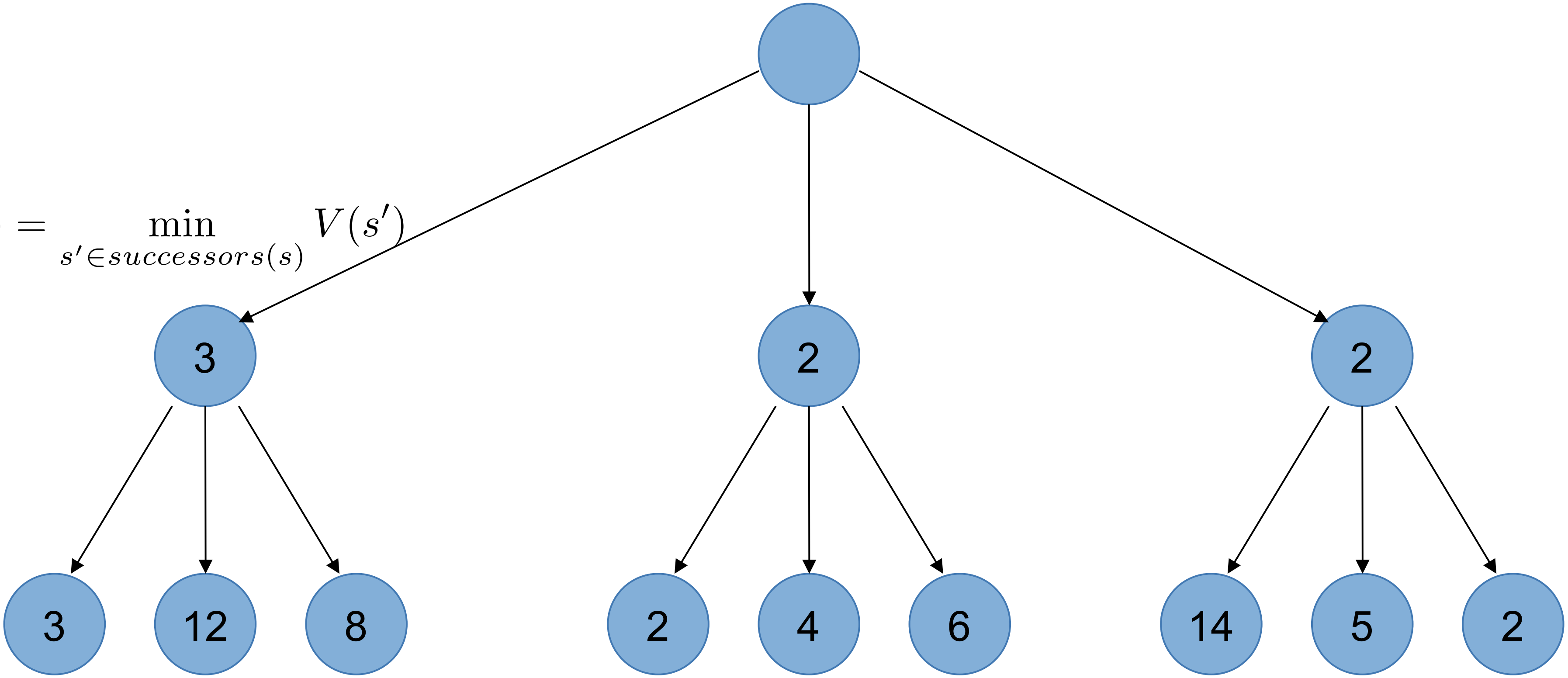
What is Minimax?

Max
(you)

$$V(s) = \min_{s' \in successors(s)} V(s')$$

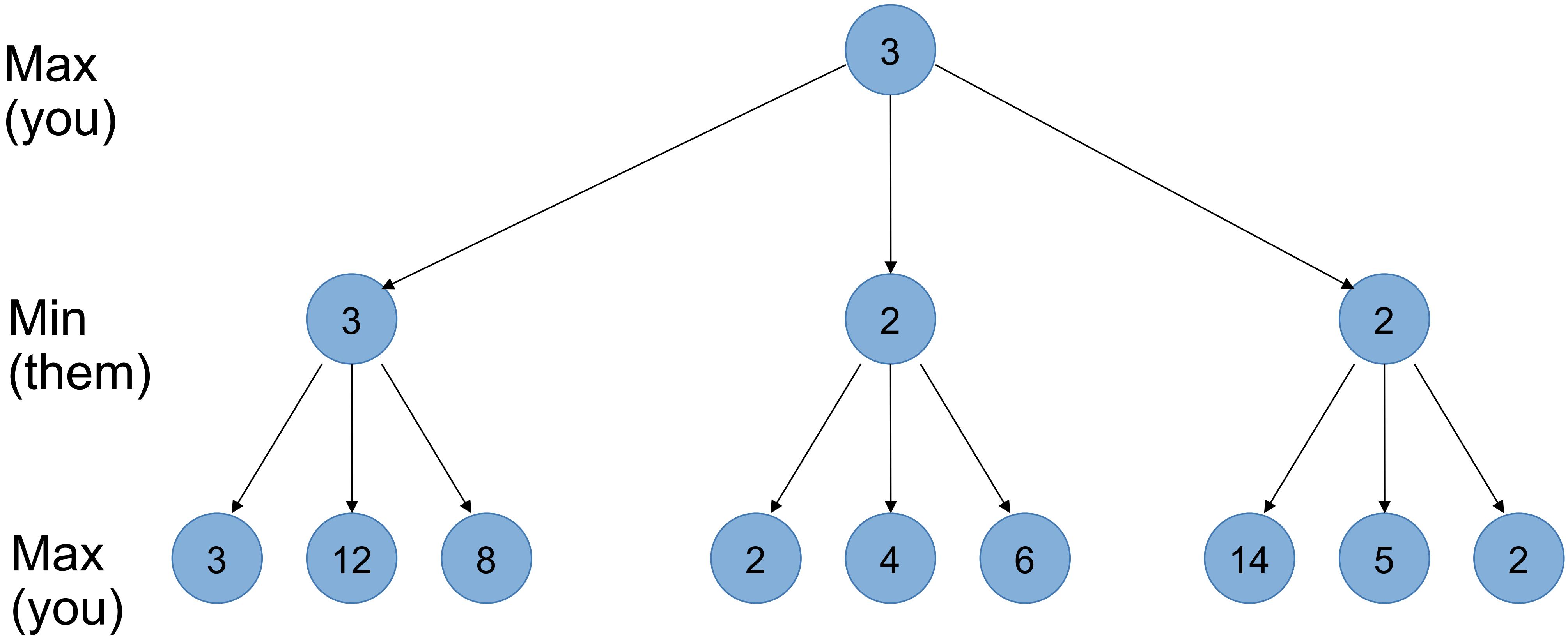
Min
(them)

Max
(you)



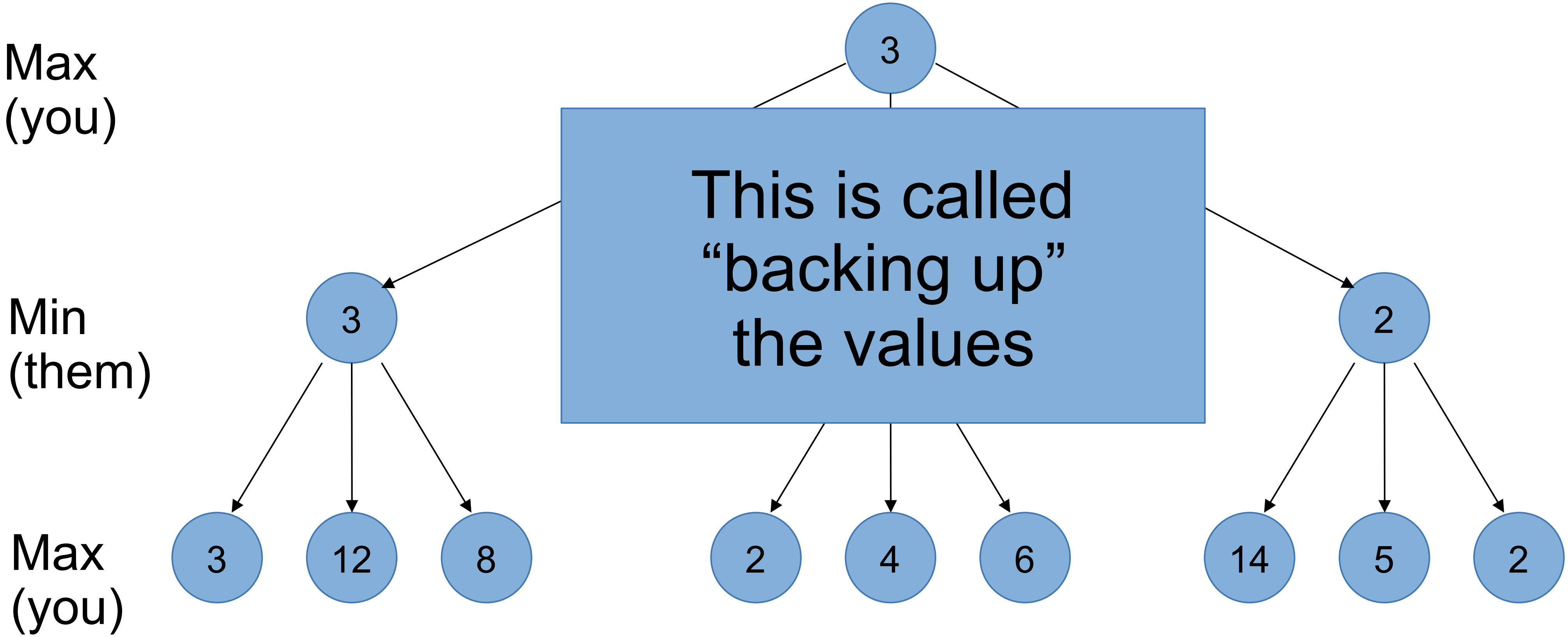
What is Minimax?

$$V(s) = \max_{s' \in successors(s)} V(s')$$



What is Minimax?

$$V(s) = \max_{s' \in successors(s)} V(s')$$



This is called
“backing up”
the values

Minimax

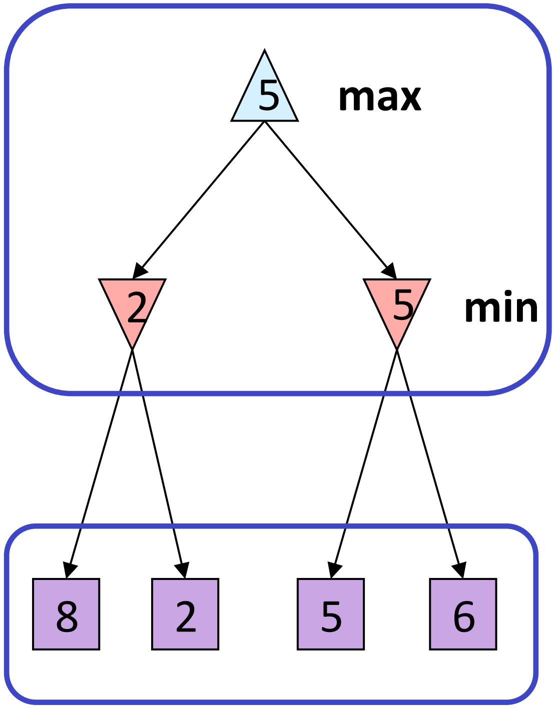
Deterministic, zero-sum games:

- 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

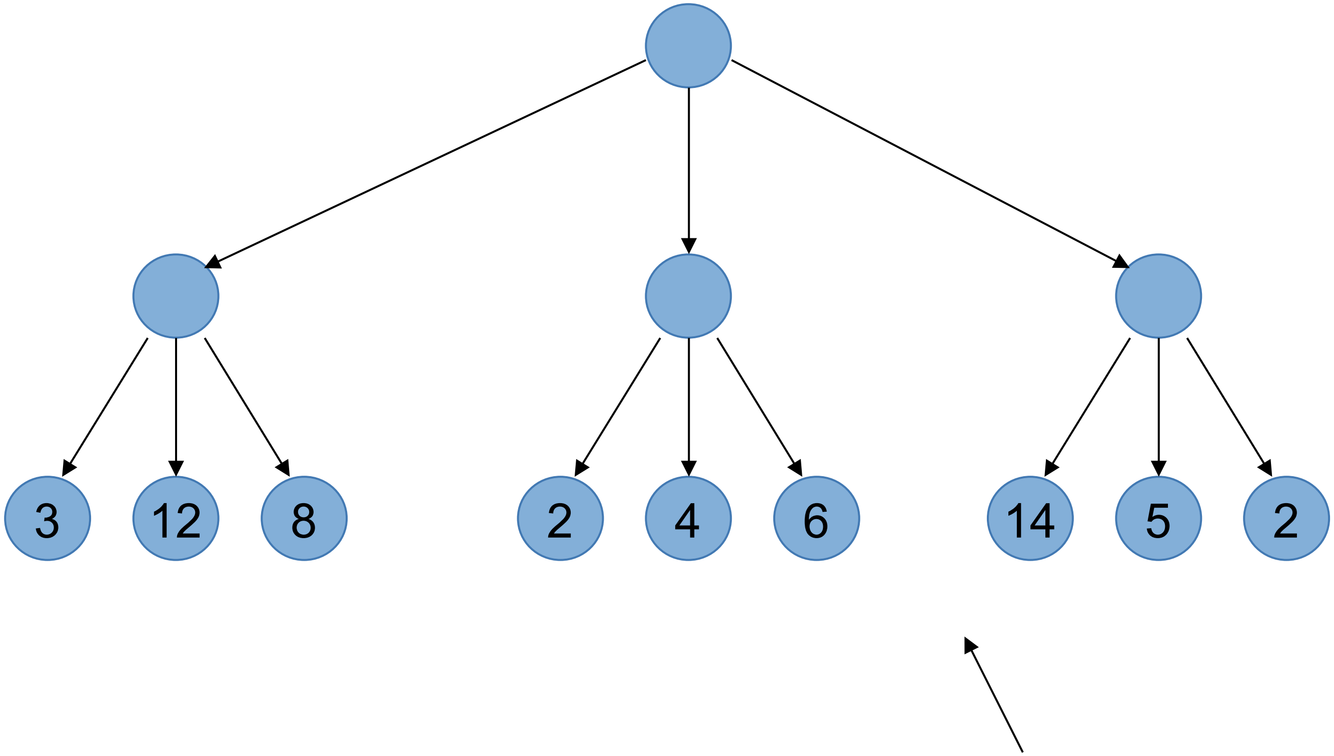


Terminal values: part of the game

Minimax

Okay – so we know how to back up values ...

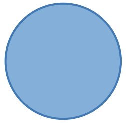
… but, how do we construct the tree?



This tree is already built...

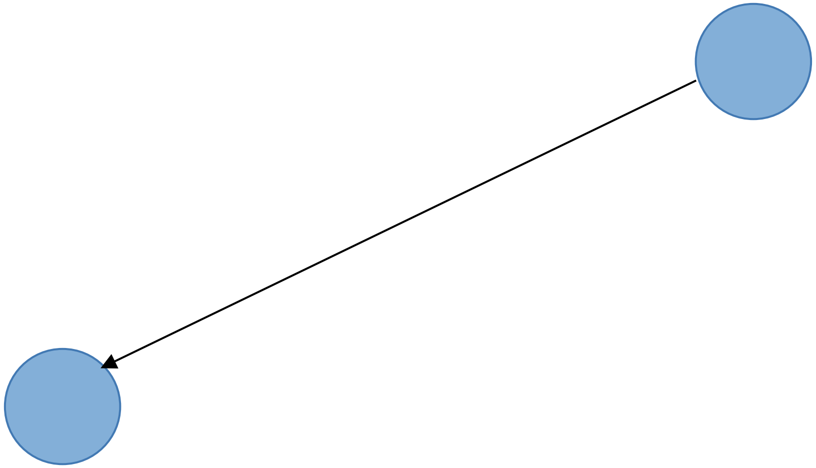
Minimax

Notice that we only get utilities at the *bottom* of the tree …
– therefore, DFS makes sense.



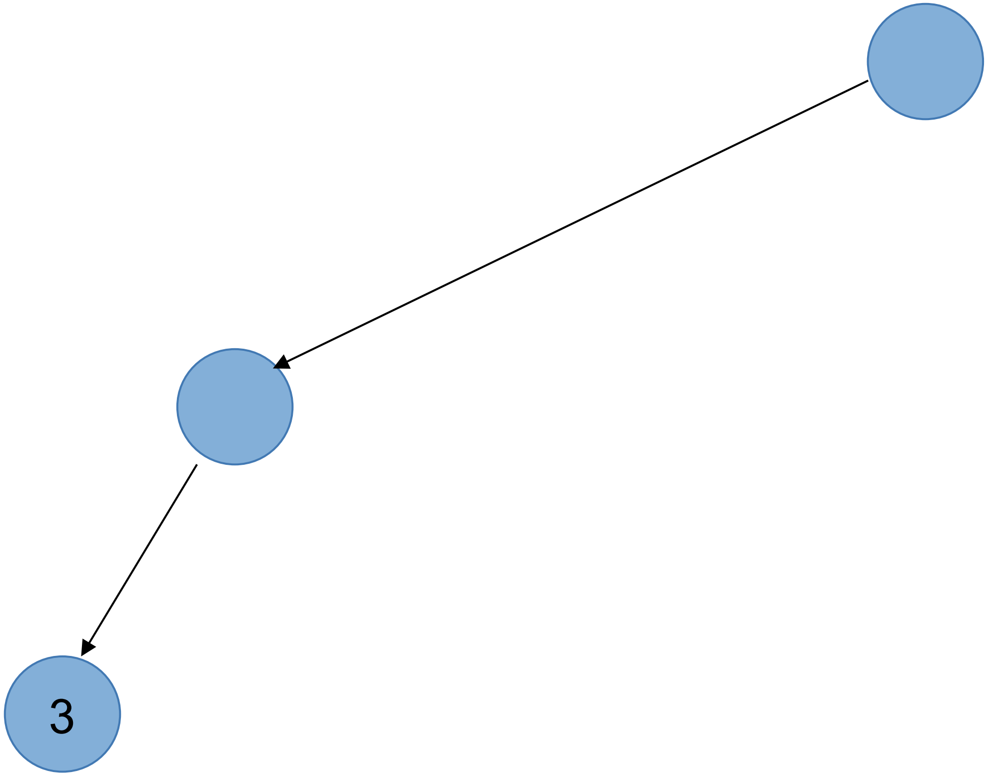
Minimax

Notice that we only get utilities at the *bottom* of the tree …
– therefore, DFS makes sense.



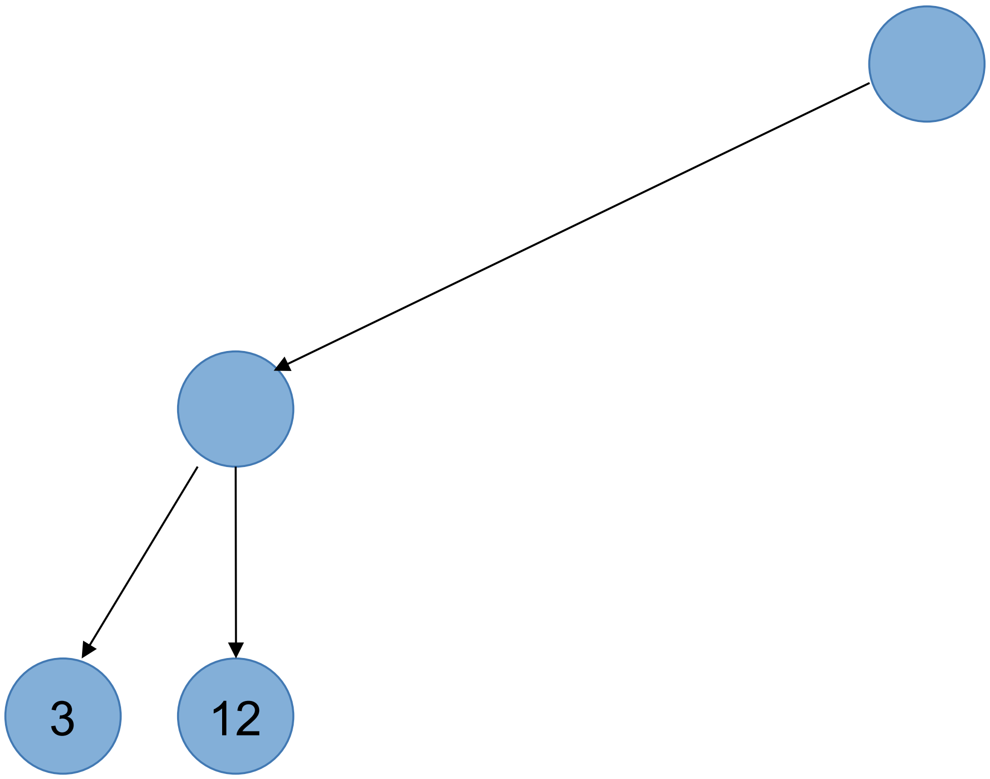
Minimax

Notice that we only get utilities at the *bottom* of the tree …
– therefore, DFS makes sense.



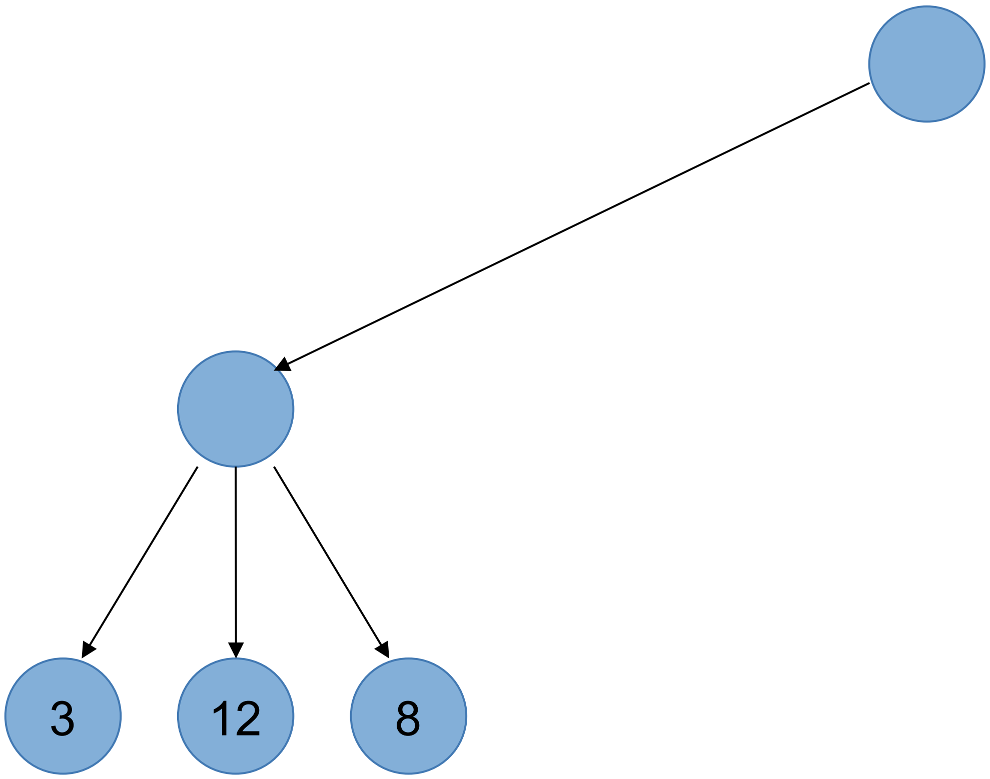
Minimax

Notice that we only get utilities at the *bottom* of the tree … – therefore, DFS makes sense.



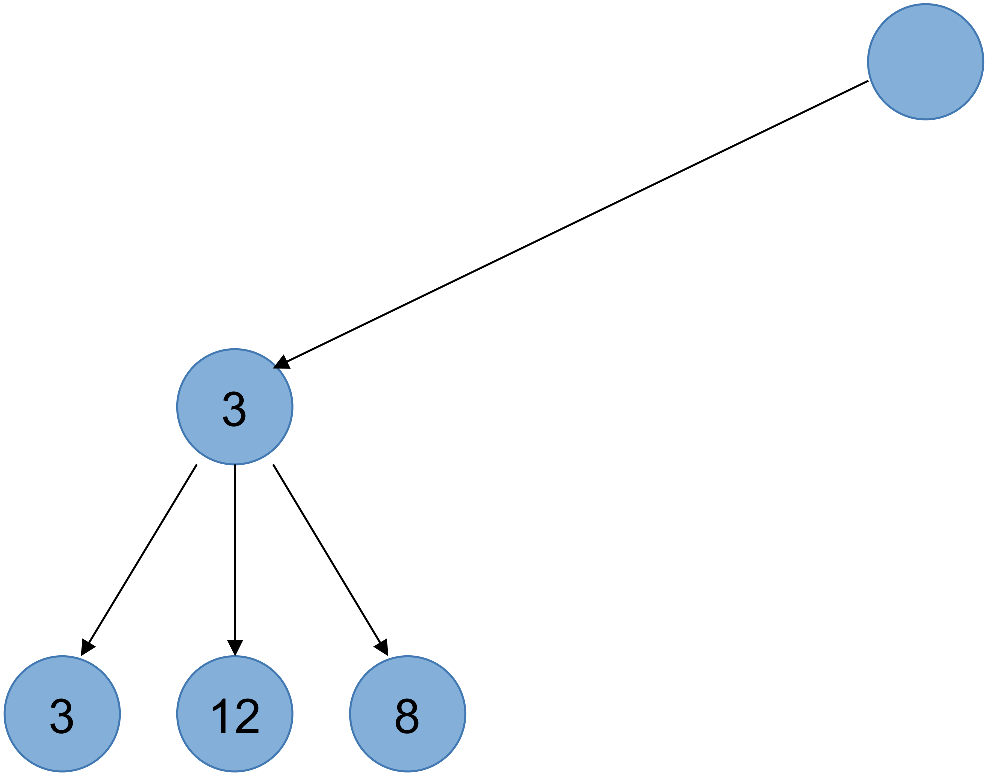
Minimax

Notice that we only get utilities at the *bottom* of the tree … – therefore, DFS makes sense.



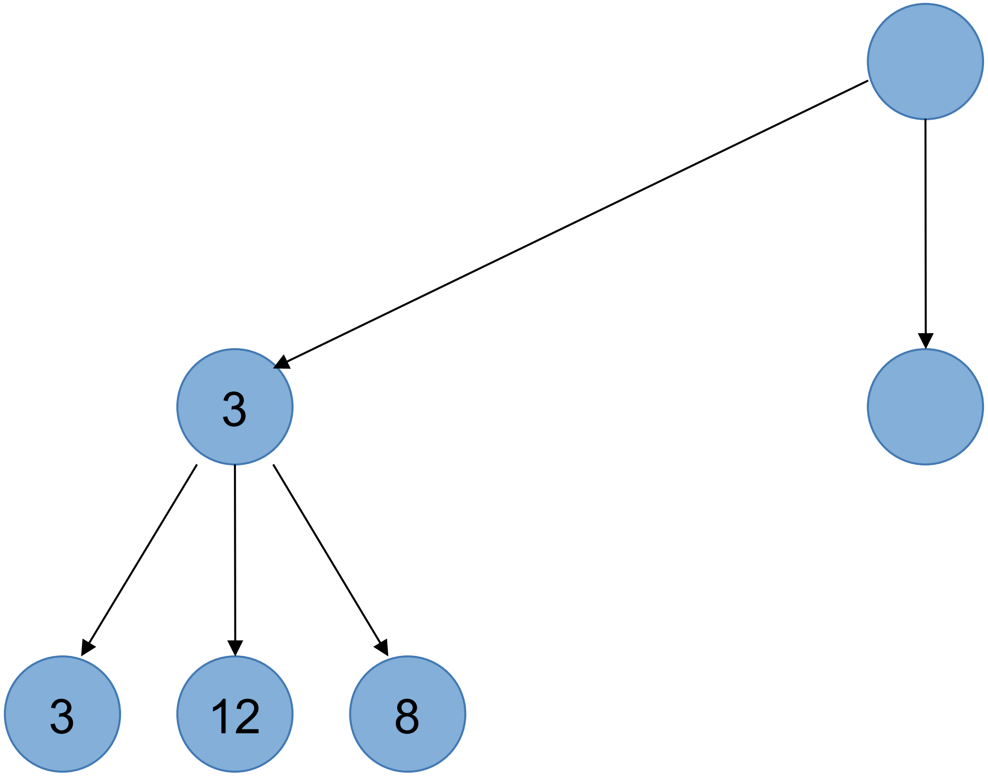
Minimax

Notice that we only get utilities at the *bottom* of the tree …
– therefore, DFS makes sense.



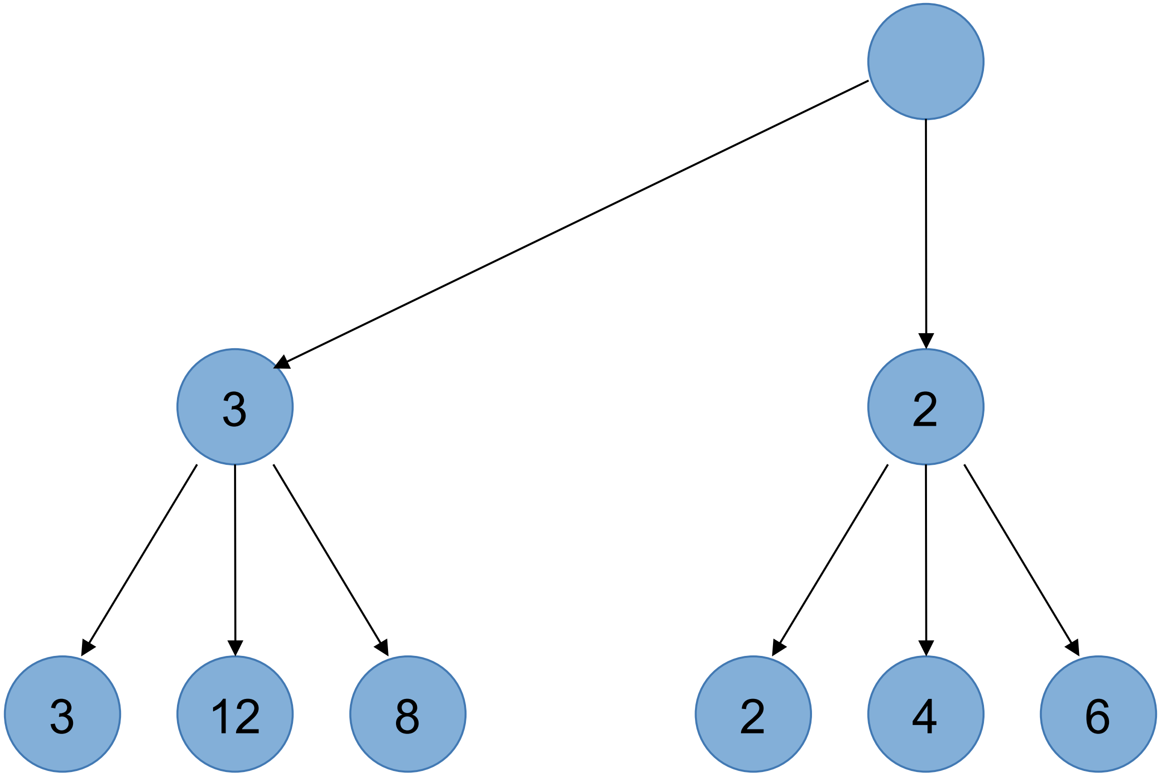
Minimax

Notice that we only get utilities at the *bottom* of the tree …
– therefore, DFS makes sense.



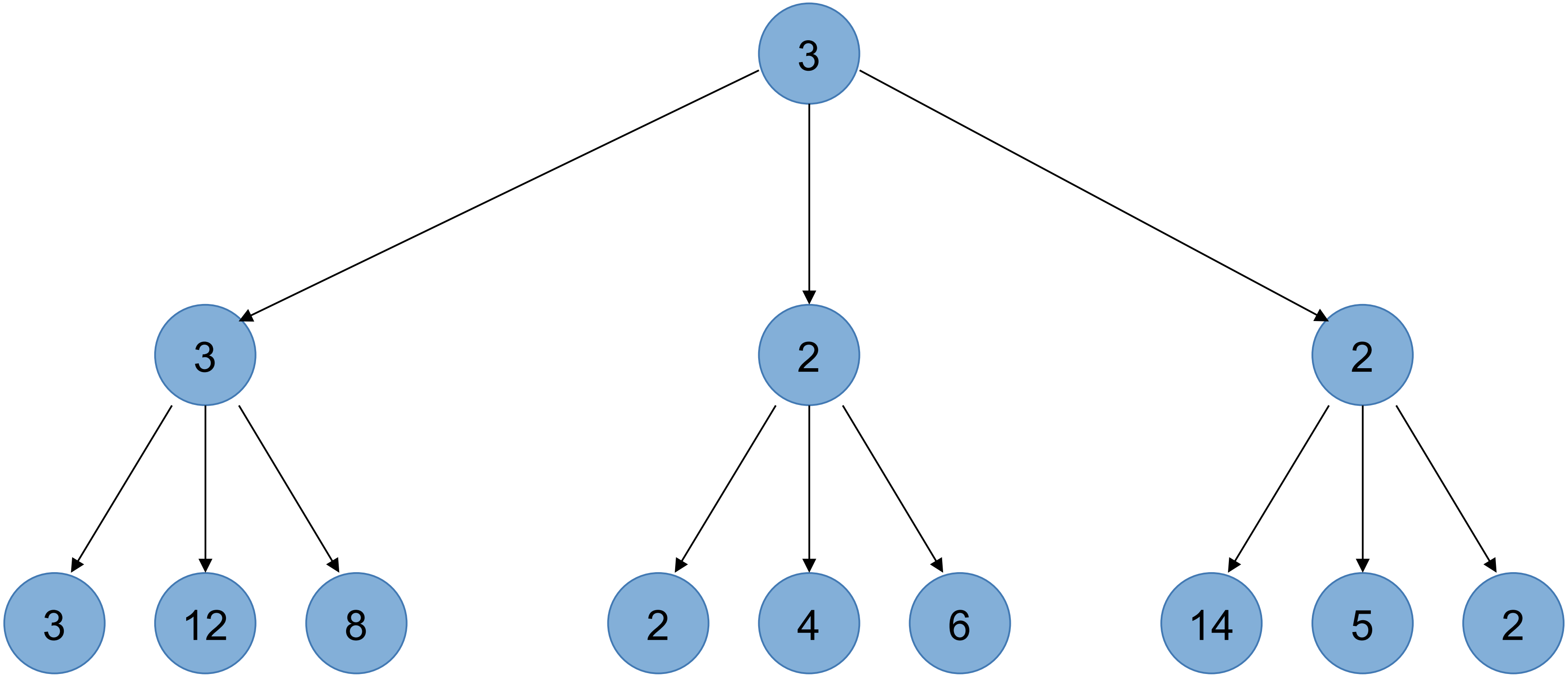
Minimax

Notice that we only get utilities at the *bottom* of the tree …
– therefore, DFS makes sense.



Minimax

Notice that we only get utilities at the *bottom* of the tree …
– therefore, DFS makes sense.



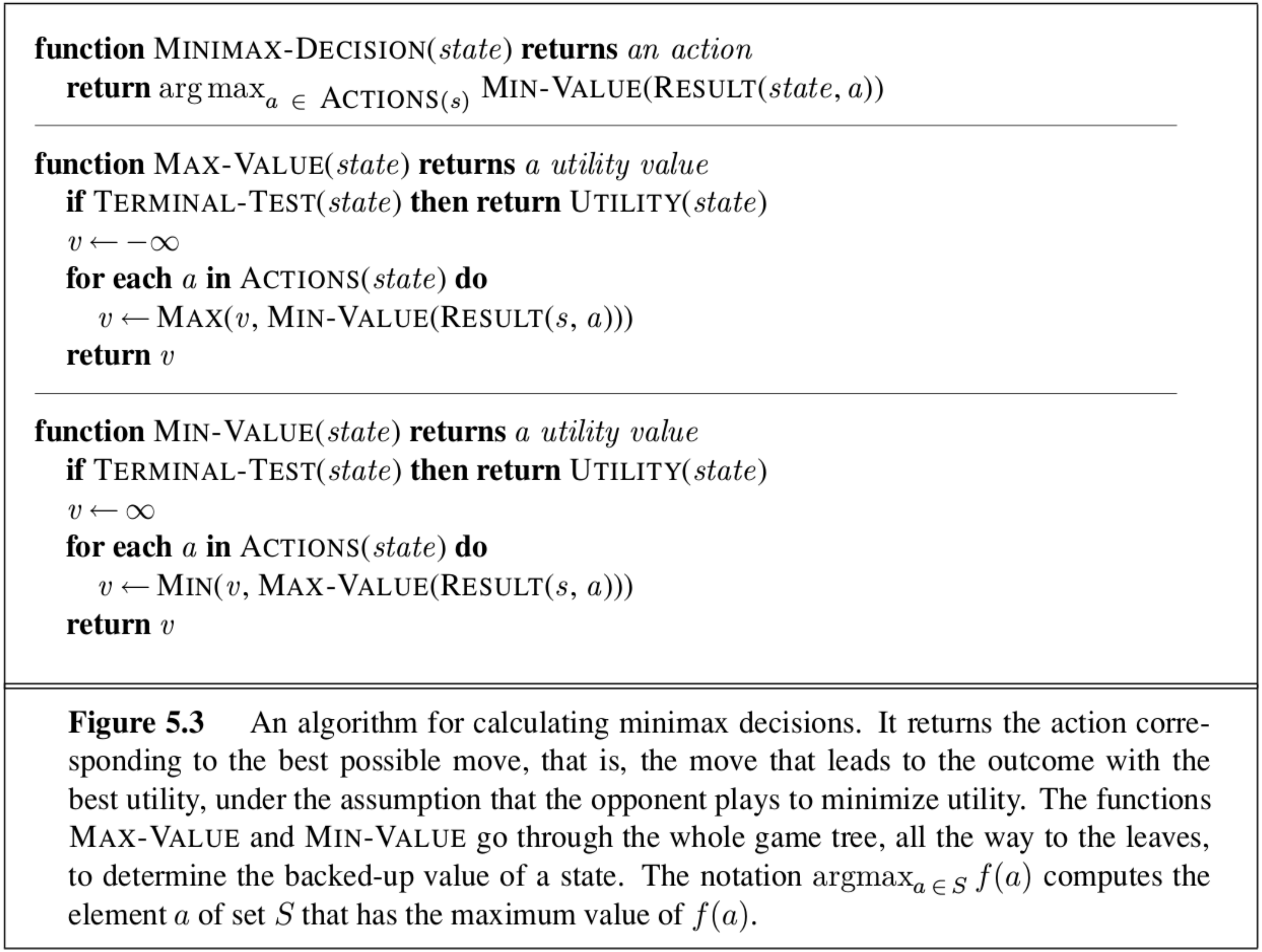
Minimax

Notice that we only get utilities at the *bottom* of the tree …

– therefore, DFS makes sense.

– since most games have forward progress, the distinction between tree search and graph search is less important

Minimax

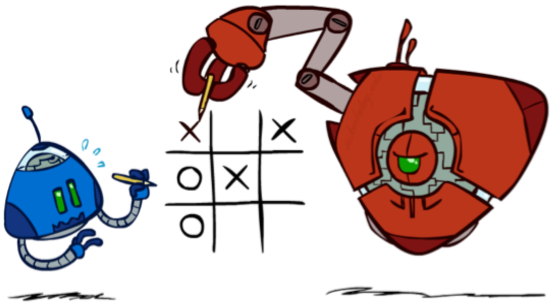


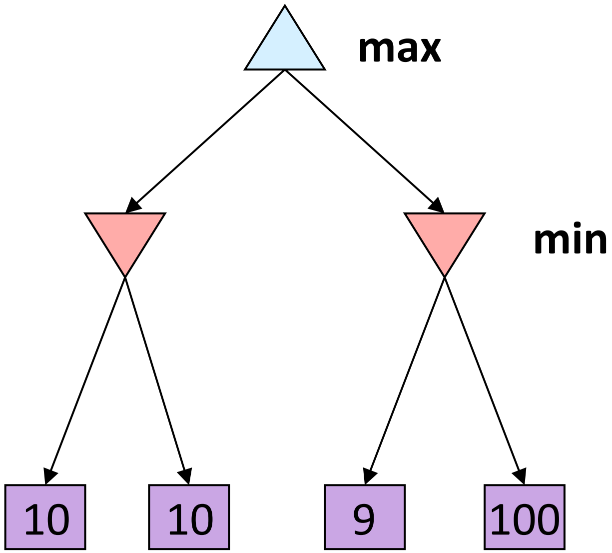
```
function MINIMAX-DECISION($state$) returns an action
  return $\arg\max_{a \in \text{ACTIONS}(s)}$ MIN-VALUE(RESULT($state$, $a$))

function MAX-VALUE($state$) returns a utility value
  if TERMINAL-TEST($state$) then return UTILITY($state$)
  $v \leftarrow -\infty$
  for each $a$ in ACTIONS($state$) do
    $v \leftarrow$ MAX($v$, MIN-VALUE(RESULT($s$, $a$)))
  return $v$

function MIN-VALUE($state$) returns a utility value
  if TERMINAL-TEST($state$) then return UTILITY($state$)
  $v \leftarrow \infty$
  for each $a$ in ACTIONS($state$) do
    $v \leftarrow$ MIN($v$, MAX-VALUE(RESULT($s$, $a$)))
  return $v$
```

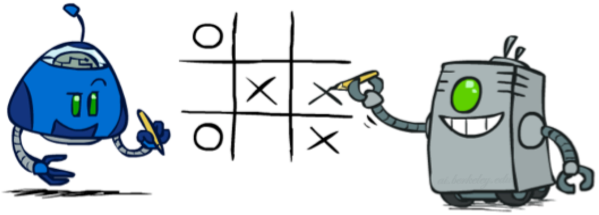
Figure 5.3 An algorithm for calculating minimax decisions. It returns the action corresponding to the best possible move, that is, the move that leads to the outcome with the best utility, under the assumption that the opponent plays to minimize utility. The functions MAX-VALUE and MIN-VALUE go through the whole game tree, all the way to the leaves, to determine the backed-up value of a state. The notation $\text{argmax}_{a \in S} f(a)$ computes the element a of set S that has the maximum value of $f(a)$.

Minimax properties

Is it always correct to assume your opponent plays optimally?







Minimax properties

Is minimax optimal? Is it complete?

Minimax properties

Is minimax optimal? Is it complete?

Time complexity = ?

Space complexity = ?

Minimax properties

Is minimax optimal? Is it complete?

Time complexity = $O(b^d)$

Space complexity = $O(bd)$

Minimax properties

Is minimax optimal? Is it complete?

Time complexity = $O(b^d)$

Space complexity = $O(bd)$

Is it practical? In chess, b=35, d=100

Minimax properties

Is minimax optimal? Is it complete?

Time complexity = $O(b^d)$

Space complexity = $O(bd)$

Is it practical? In chess, b=35, d=100

$O(35^{100})$ is a big number...

Minimax properties

Is minimax optimal? Is it complete?

Time complexity = $O(b^d)$

Space complexity = $O(bd)$

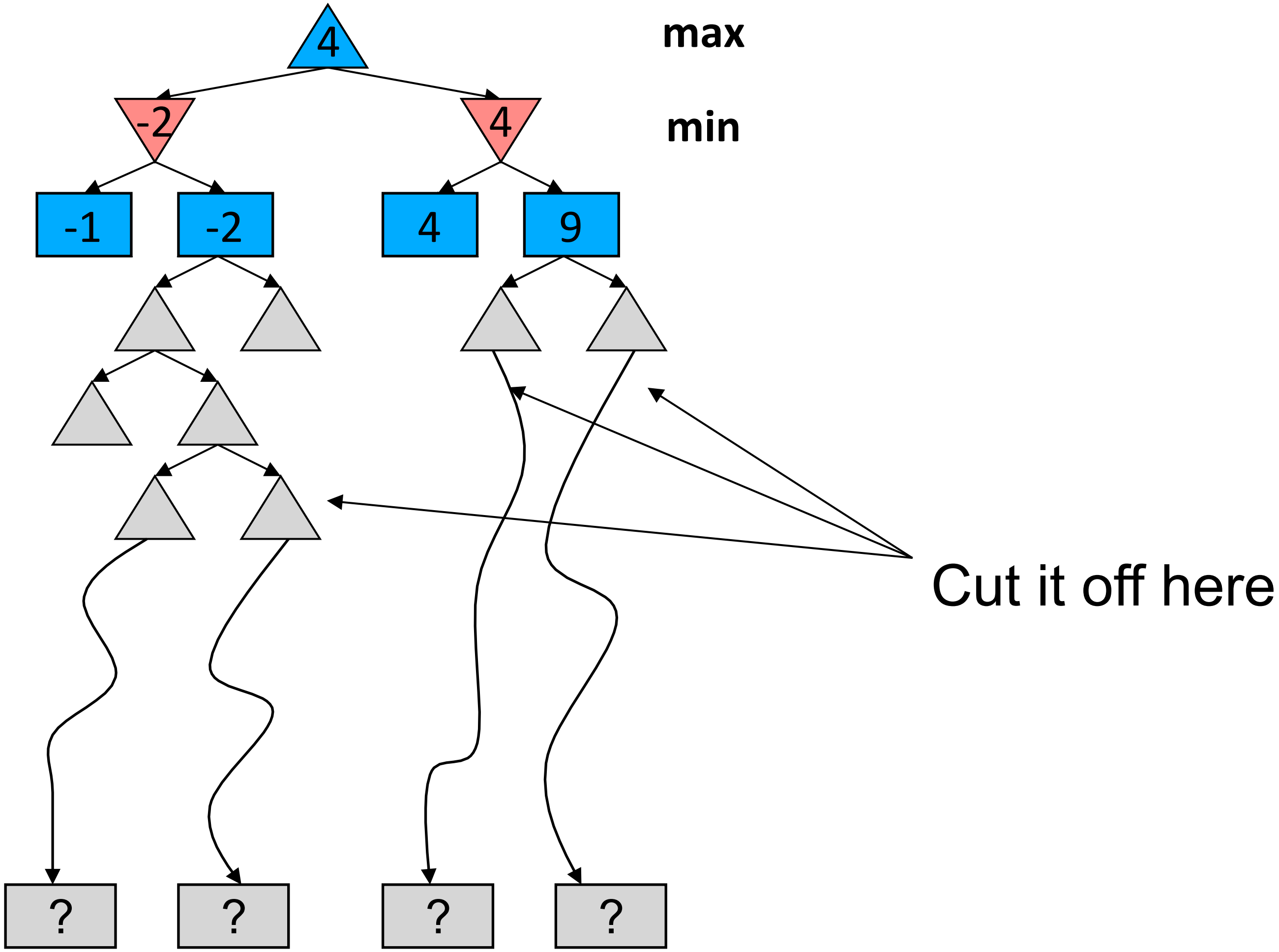
Is it practical? In chess, b=35, d=100

$O(35^{100})$ is a big number...

So what can we do?

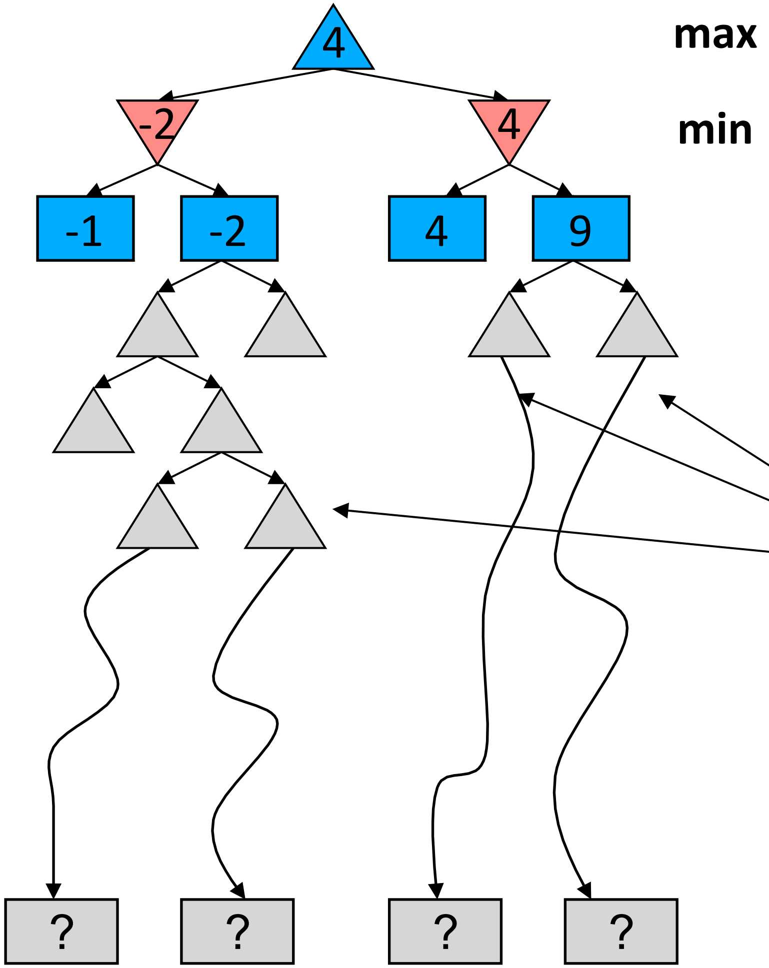
Evaluation functions

Key idea: cut off search at a certain depth and give the corresponding nodes an estimated value.



Evaluation functions

Key idea: cut off search at a certain depth and give the corresponding nodes an estimated value.

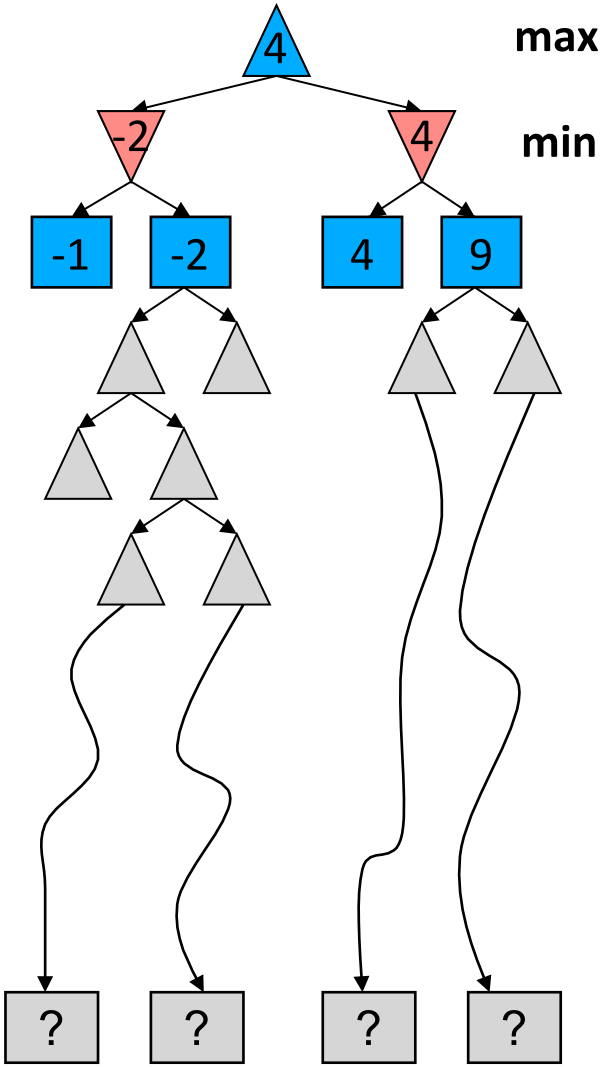


the evaluation function makes this estimate.

Cut it off here

Evaluation functions

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

Guarantee of optimal play is gone

More plies makes a BIG difference

Use iterative deepening for an anytime algorithm

Evaluation functions

How does the evaluation function make the estimate?
- depends upon domain

For example, in chess, the value of a state might equal the sum of piece values.
- a pawn counts for 1
- a rook counts for 5
- a knight counts for 3

...

A weighted linear evaluation function

$$eval(s) = w_1 f_1(s) + \cdots + w_n f_n(s)$$

$f_1(s) \equiv$ number of pawns on the board

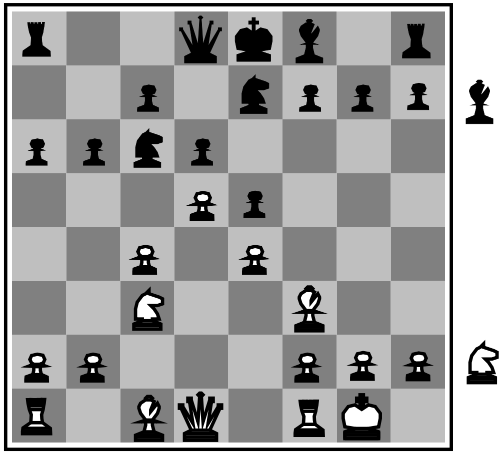
$f_2(s) \equiv$ number of knights on the board

$\vdots$

$w_1 = 1$ ← A pawn counts for 1

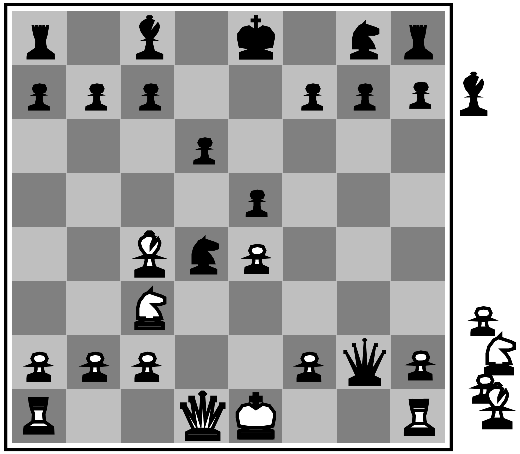
$w_2 = 3$ ← A knight counts for 3

$\vdots$



Black to move

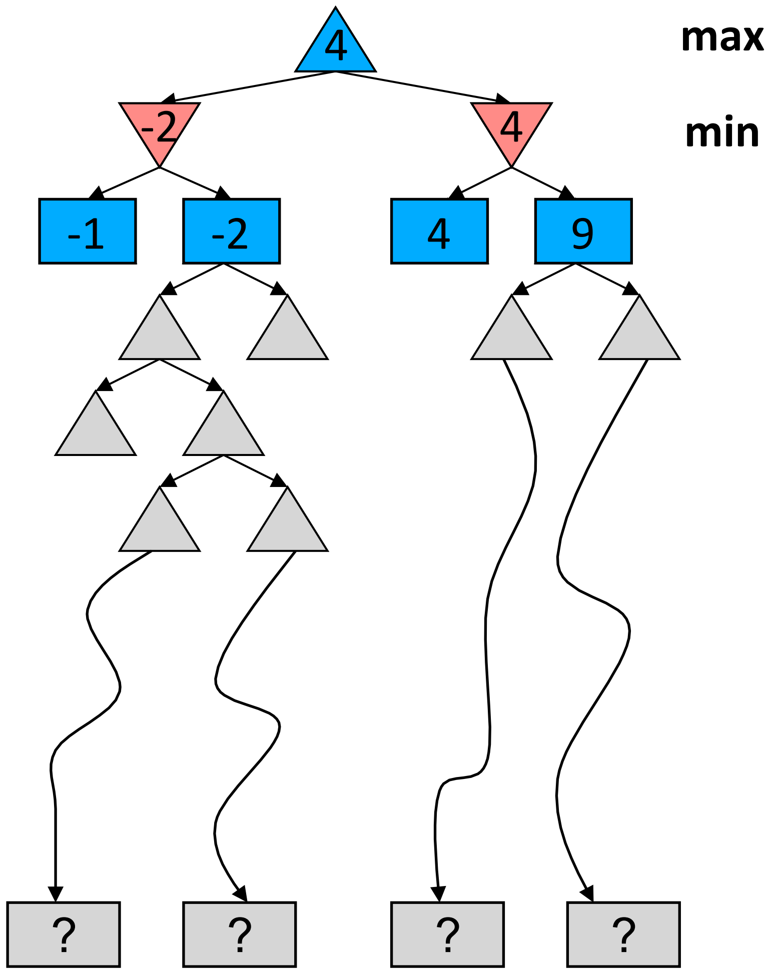
White slightly better



White to move

Black winning

At what depth do you run the evaluation function?

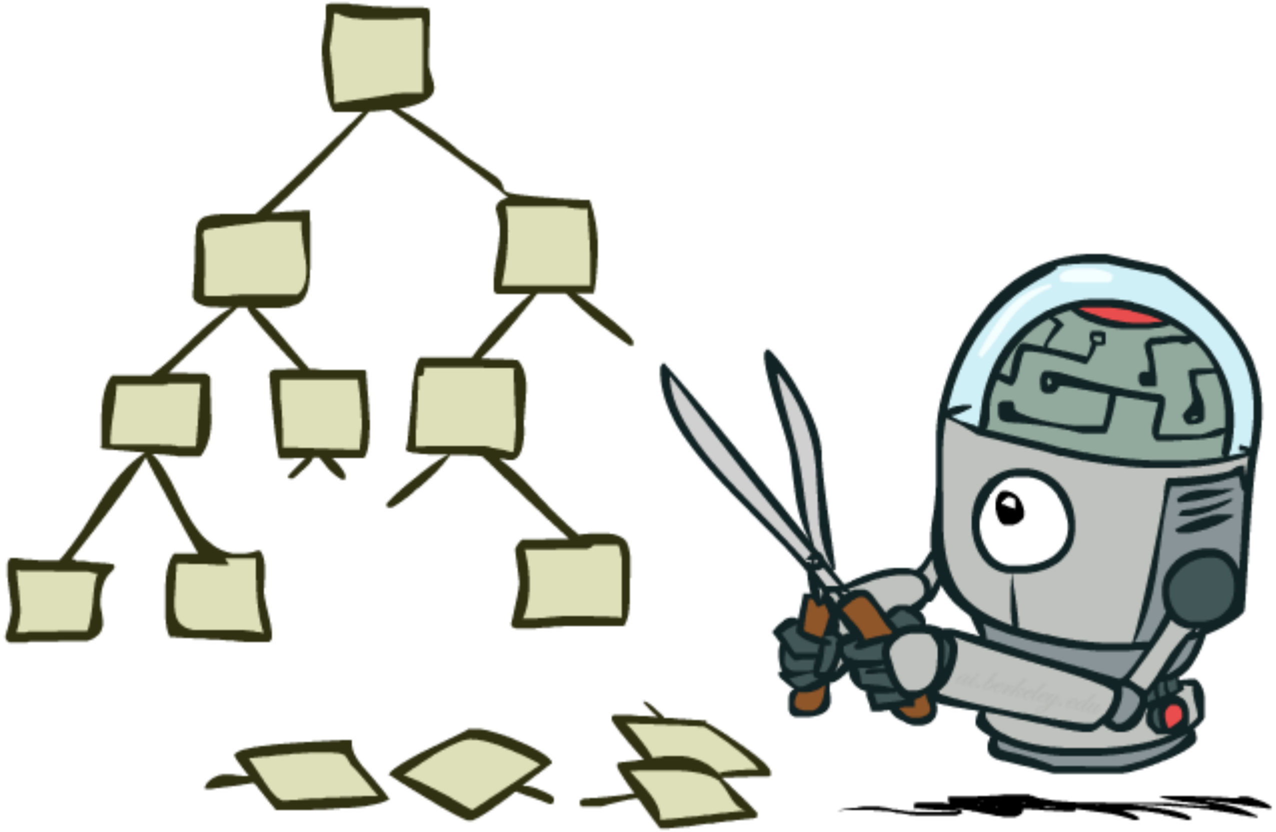


Option 1: cut off search at a fixed depth

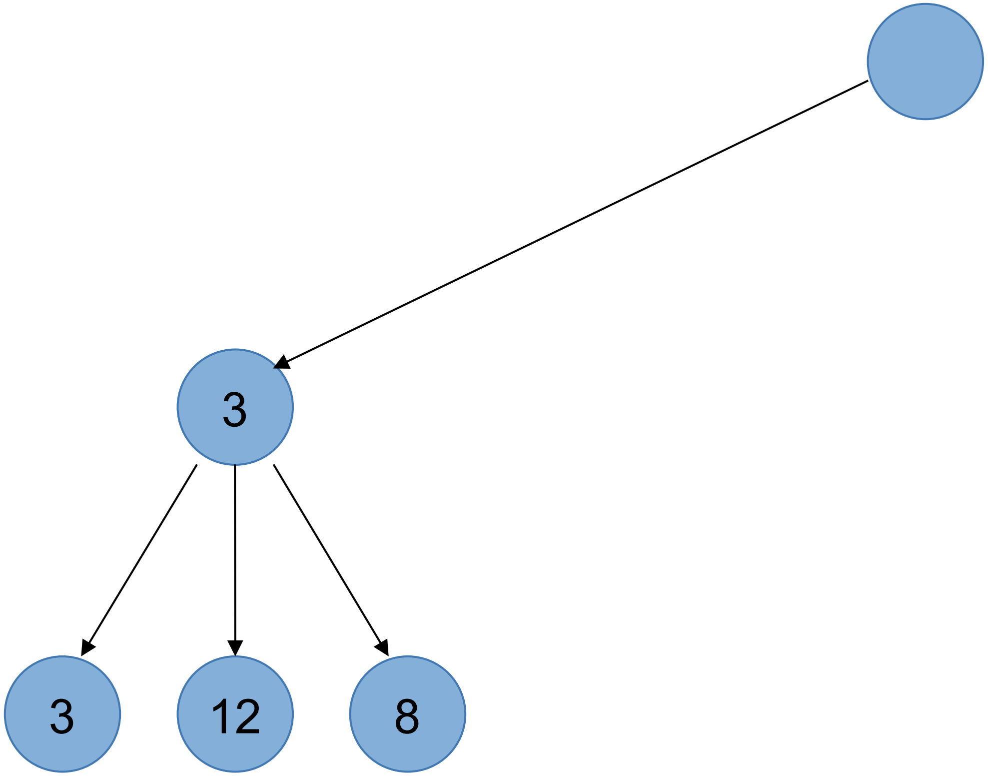
Option 2: cut off search at particular states deeper than a certain threshold

The deeper your threshold, the less the quality of the evaluation function matters...

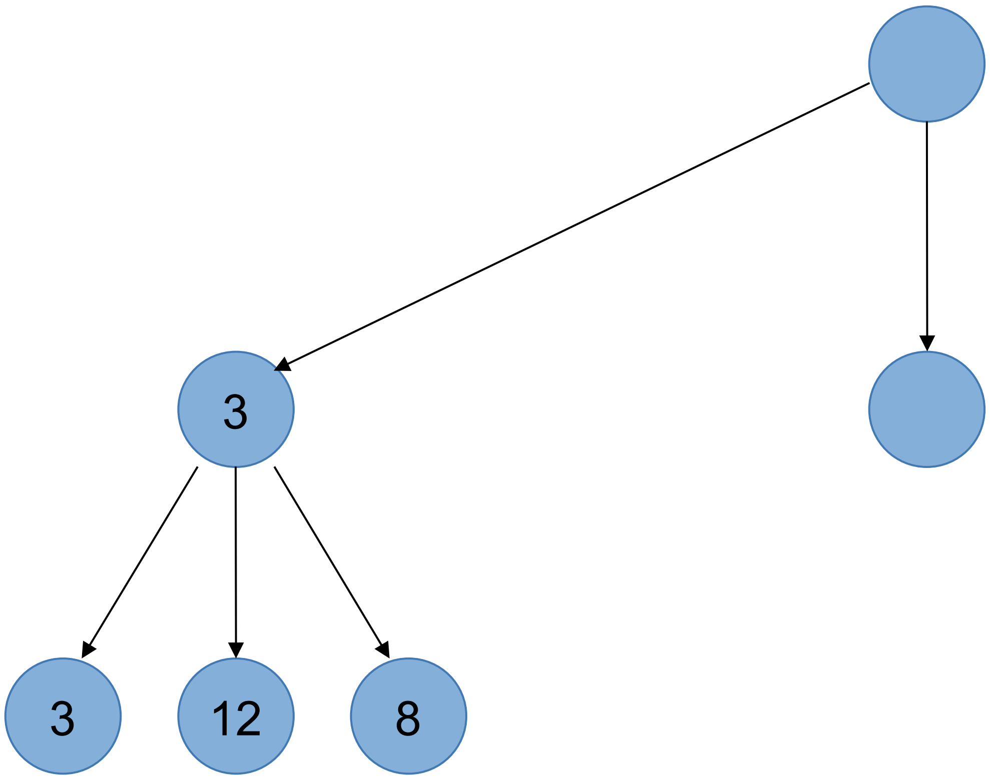
Alpha/Beta pruning



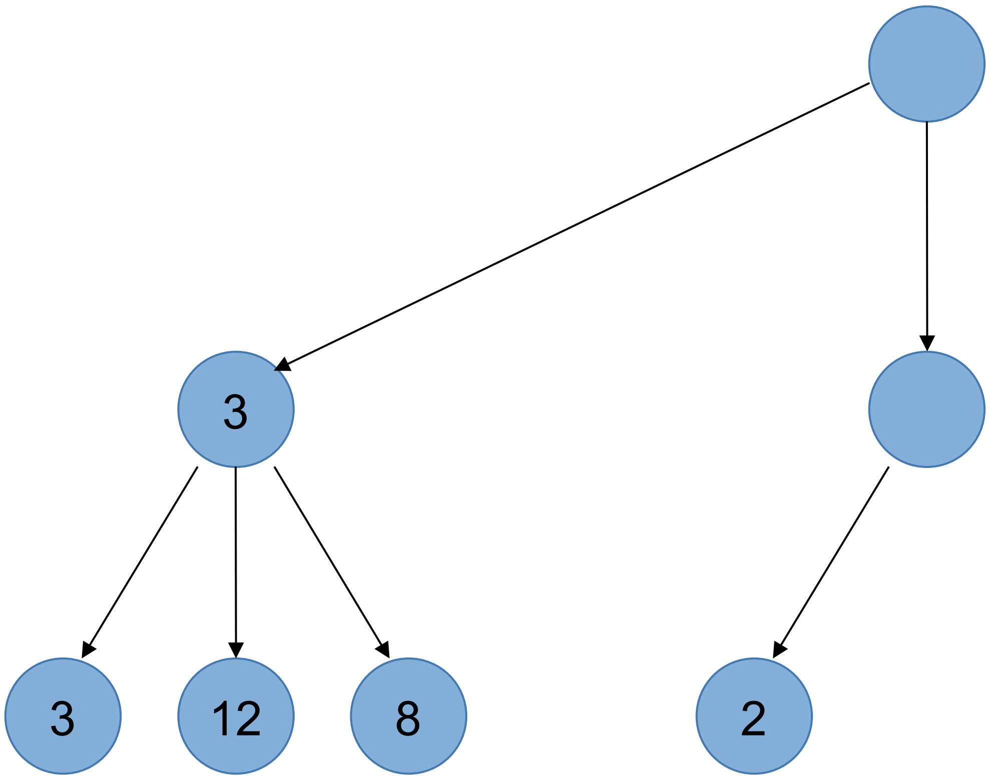
Alpha/Beta pruning



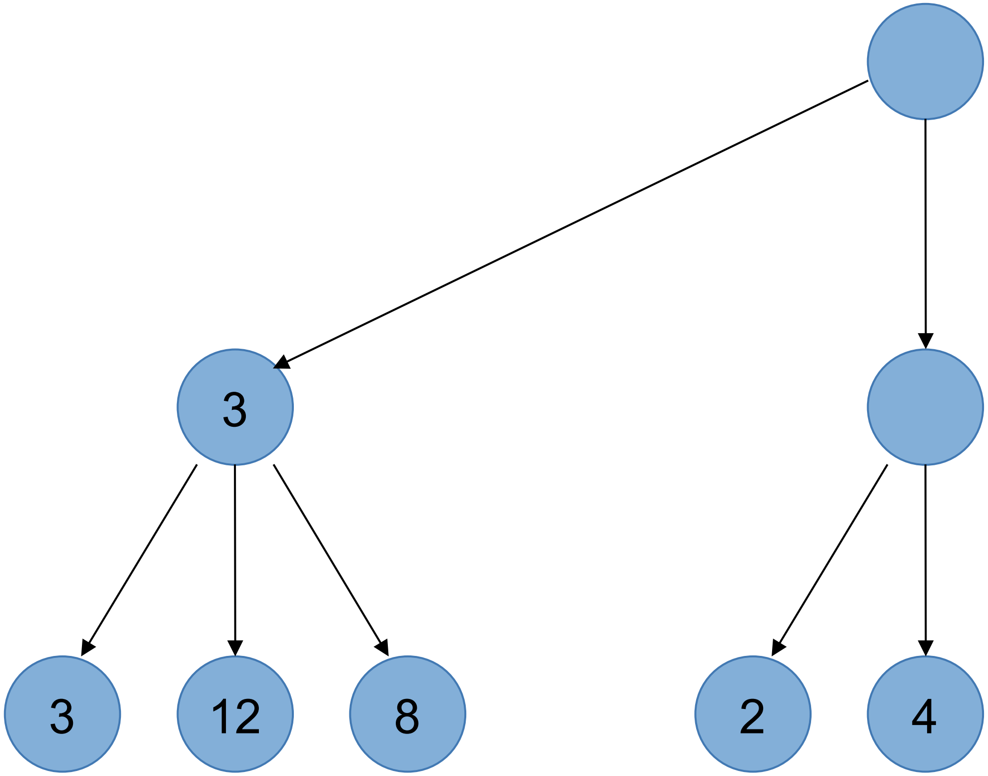
Alpha/Beta pruning



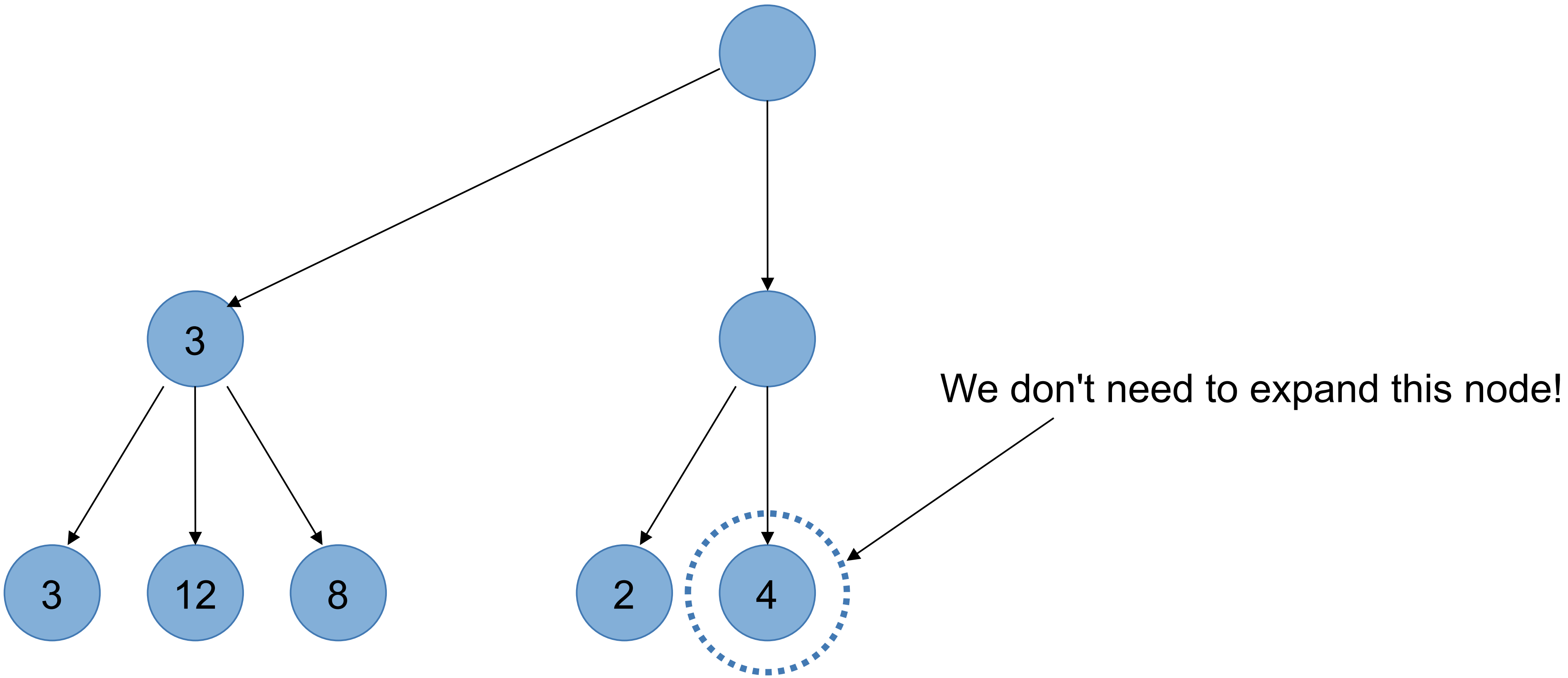
Alpha/Beta pruning



Alpha/Beta pruning

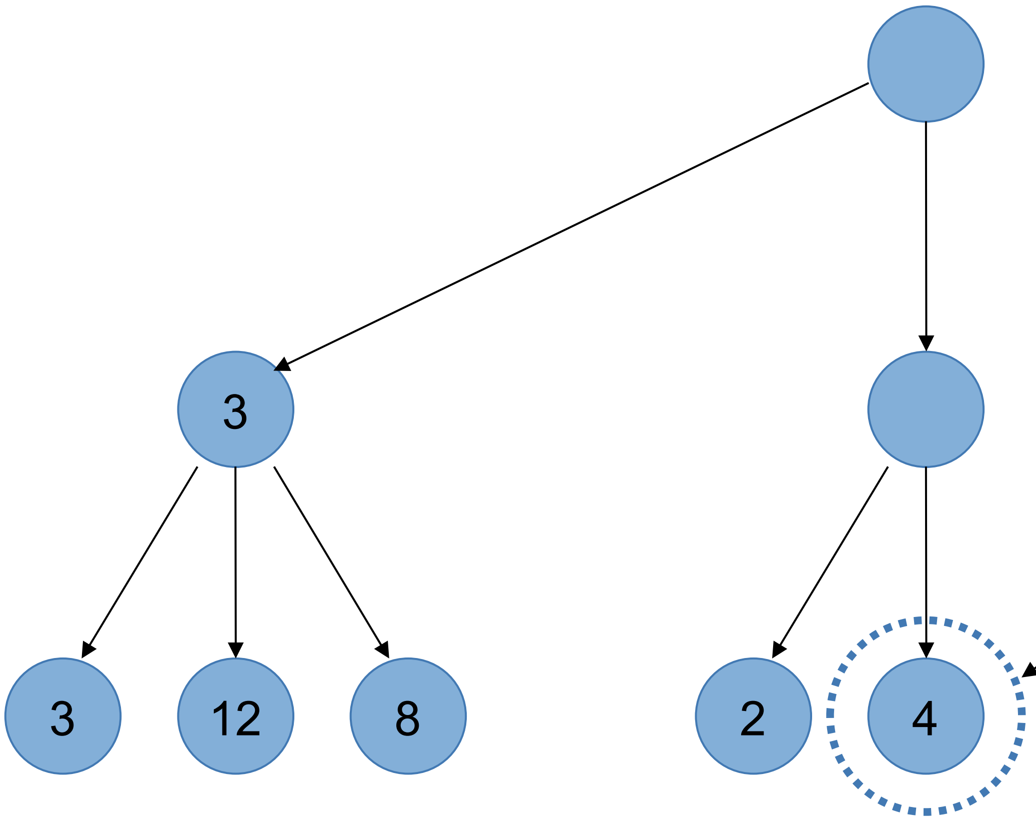


Alpha/Beta pruning



We don't need to expand this node!

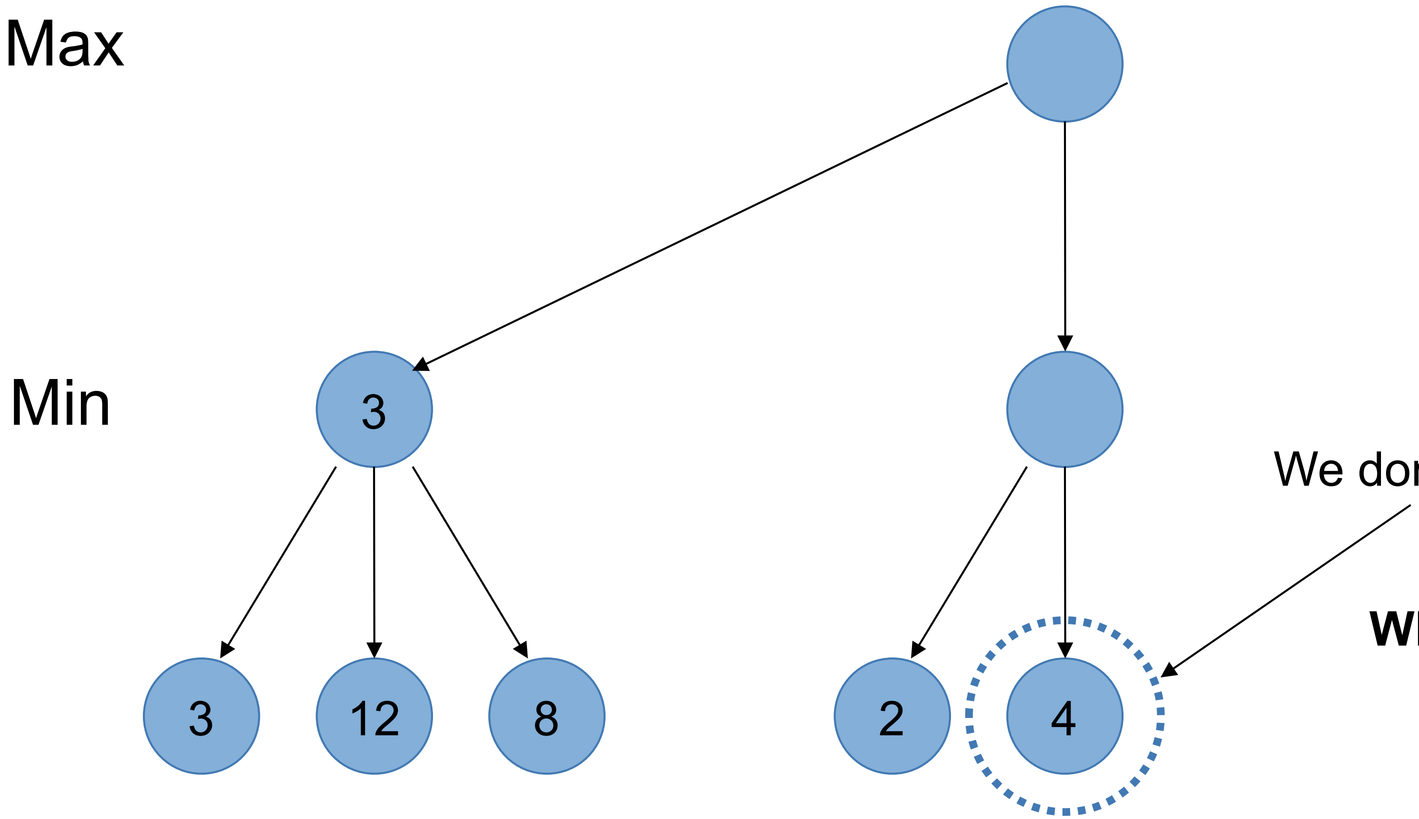
Alpha/Beta pruning



We don't need to expand this node!

Why?

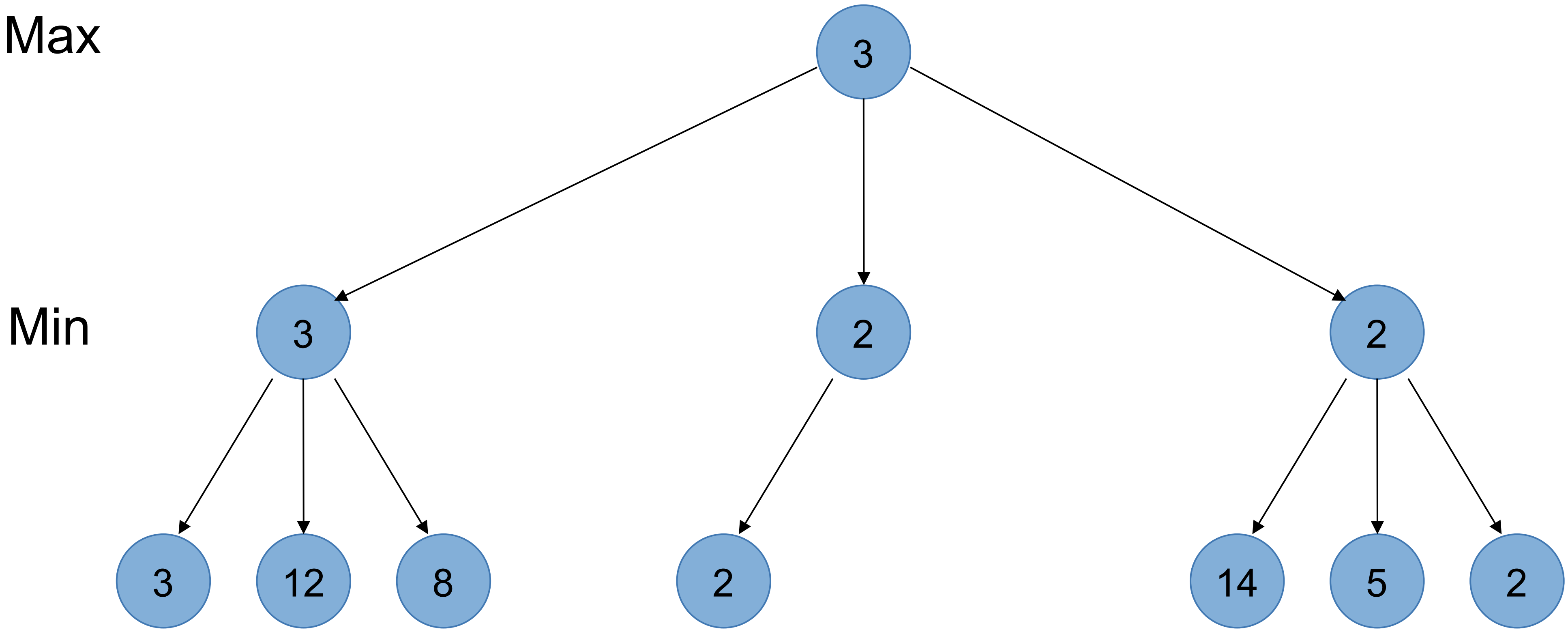
Alpha/Beta pruning



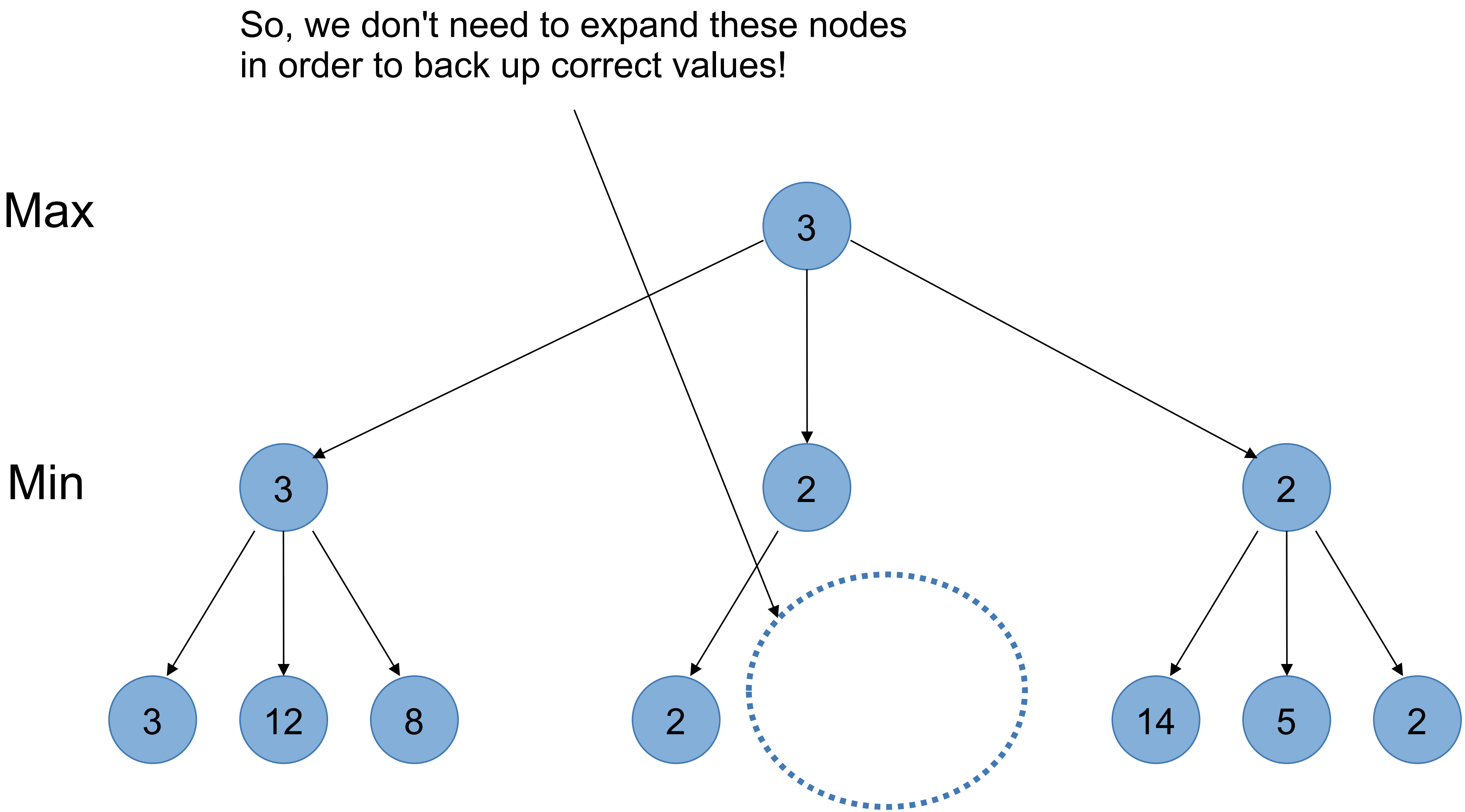
We don't need to expand this node!

Why?

Alpha/Beta pruning

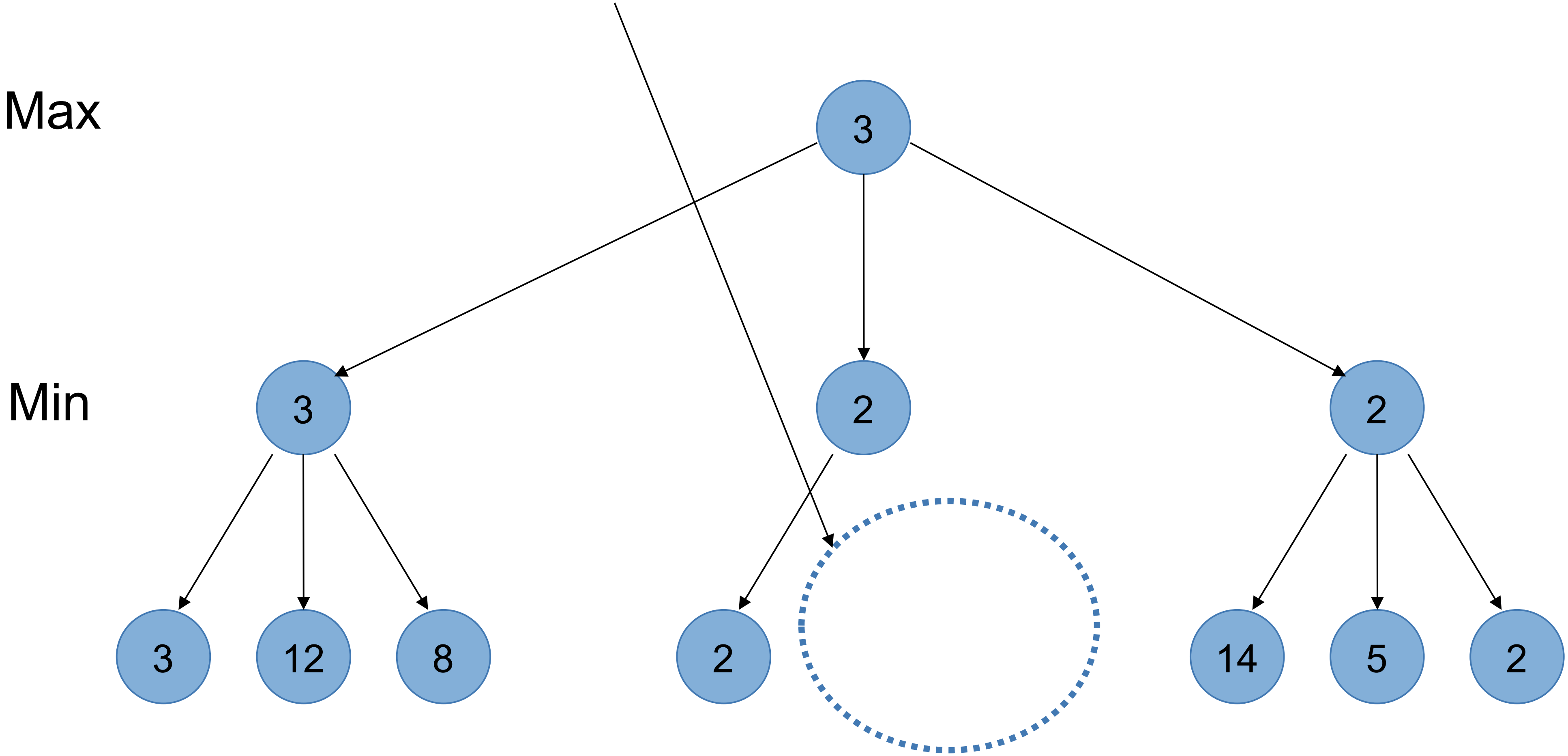


Alpha/Beta pruning



So, we don't need to expand these nodes in order to back up correct values!

Alpha/Beta pruning



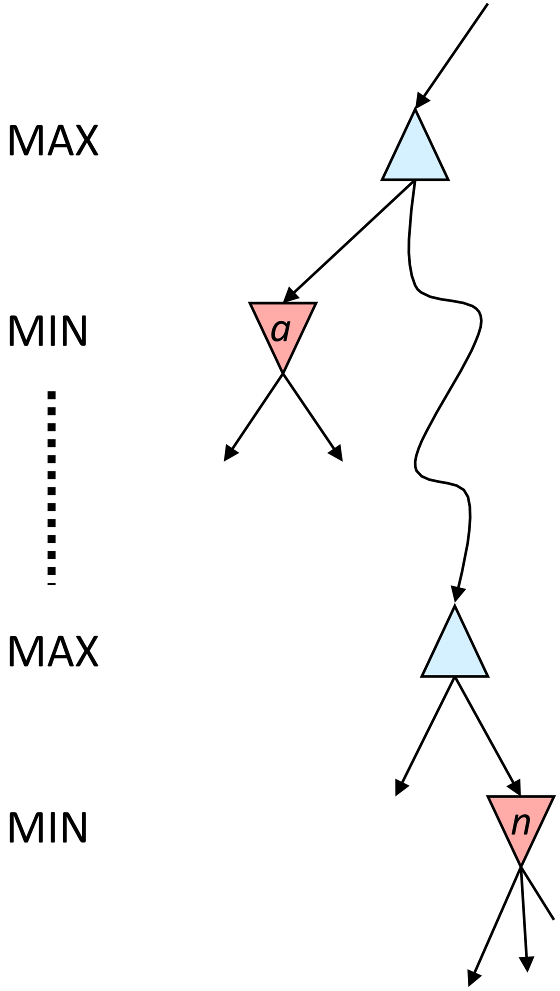
So, we don't need to expand these nodes in order to back up correct values!

That's alpha-beta pruning.

Alpha/Beta pruning: algorithm idea

General configuration (MIN version)

- We’re computing the MIN-VALUE at some node *n*
- We’re looping over *n*’s children
- *n*’s estimate of the childrens’ min is dropping
- Who cares about *n*’s value? MAX
- Let *a* be the best value that MAX can get at any choice point along the current path from the root
- If *n* becomes worse than *a*, MAX will avoid it, so we can stop considering *n*’s other children (it’s already bad enough that it won’t be played)



MAX version is symmetric

Alpha/Beta pruning: algorithm

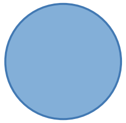
α: MAX's best option on path to root

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
```

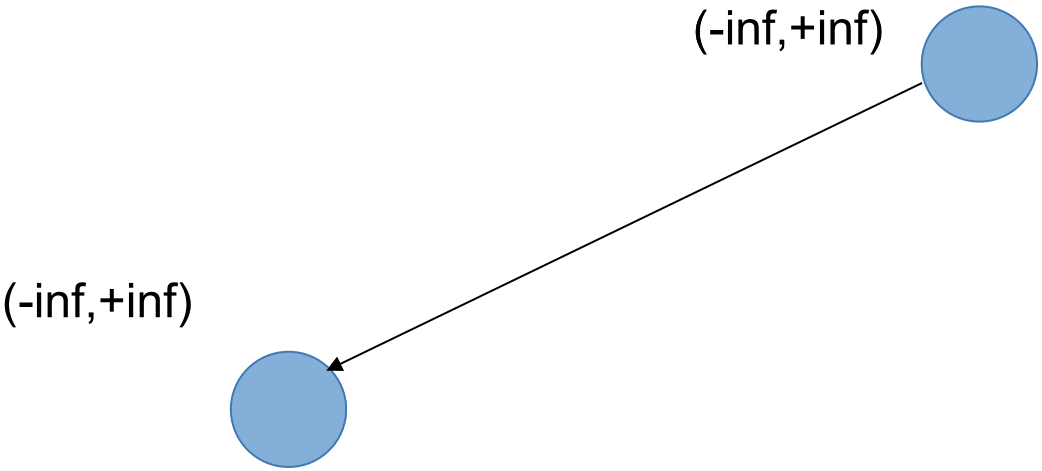
```
def min-value(state , α, β):
    initialize v = +∞
    for each successor of state:
        v = min(v, value(successor, α, β))
        if v ≤ α return v
        β = min(β, v)
    return v
```

Alpha/Beta pruning

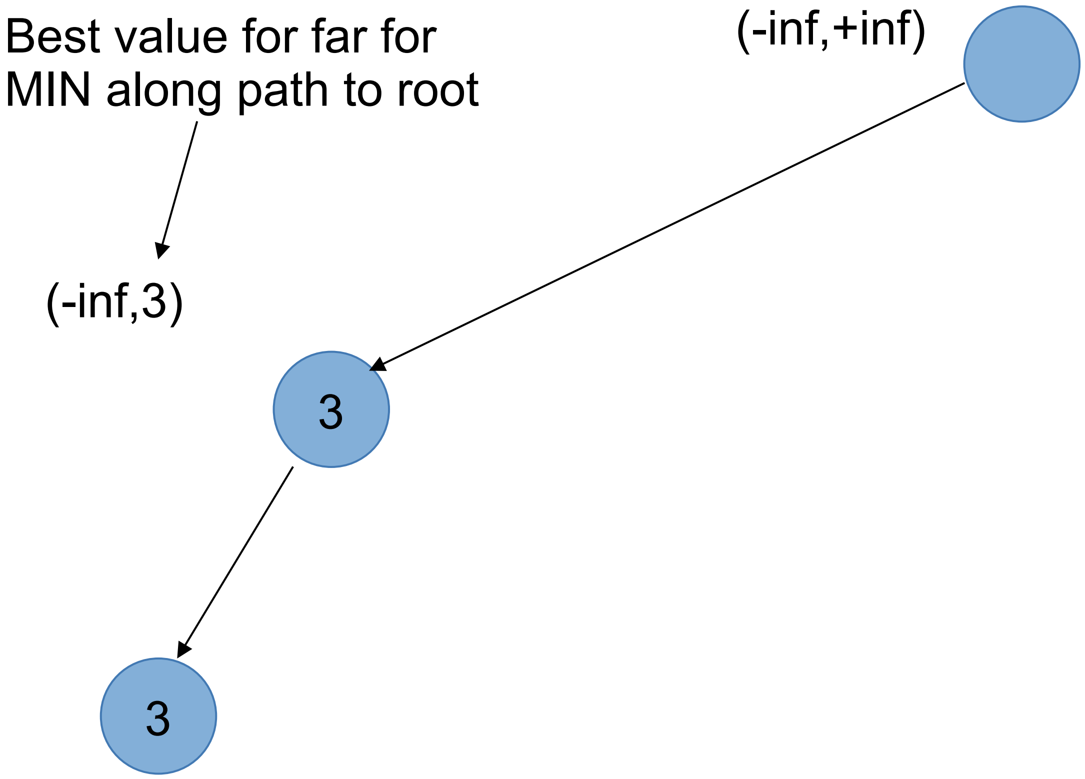


(-inf,+inf)

Alpha/Beta pruning

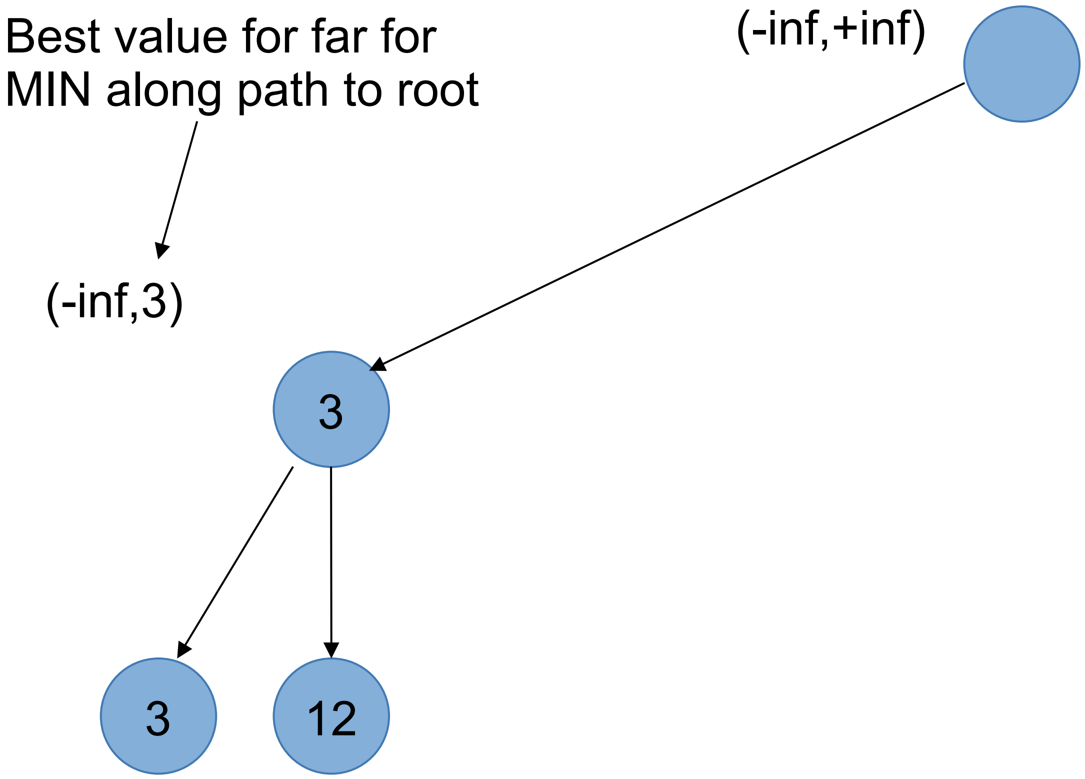


Alpha/Beta pruning

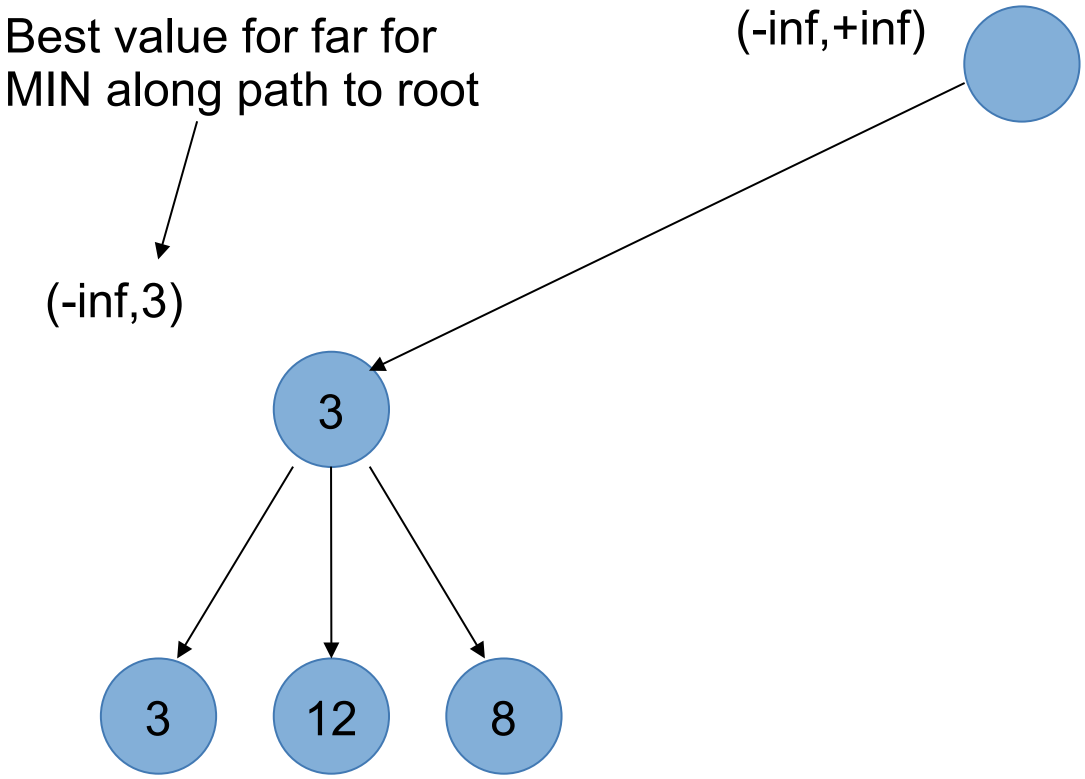


Best value for far for MIN along path to root

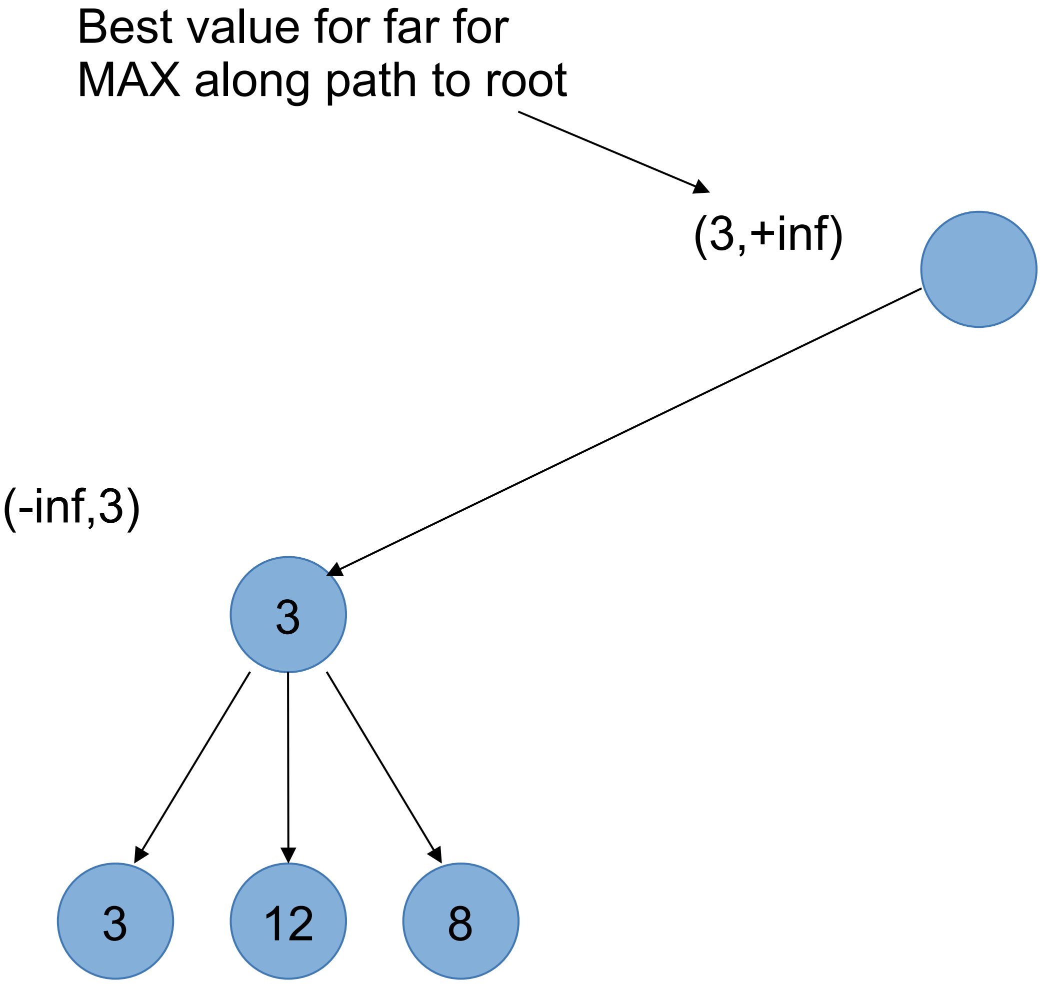
Alpha/Beta pruning


Best value for far for MIN along path to root

Alpha/Beta pruning

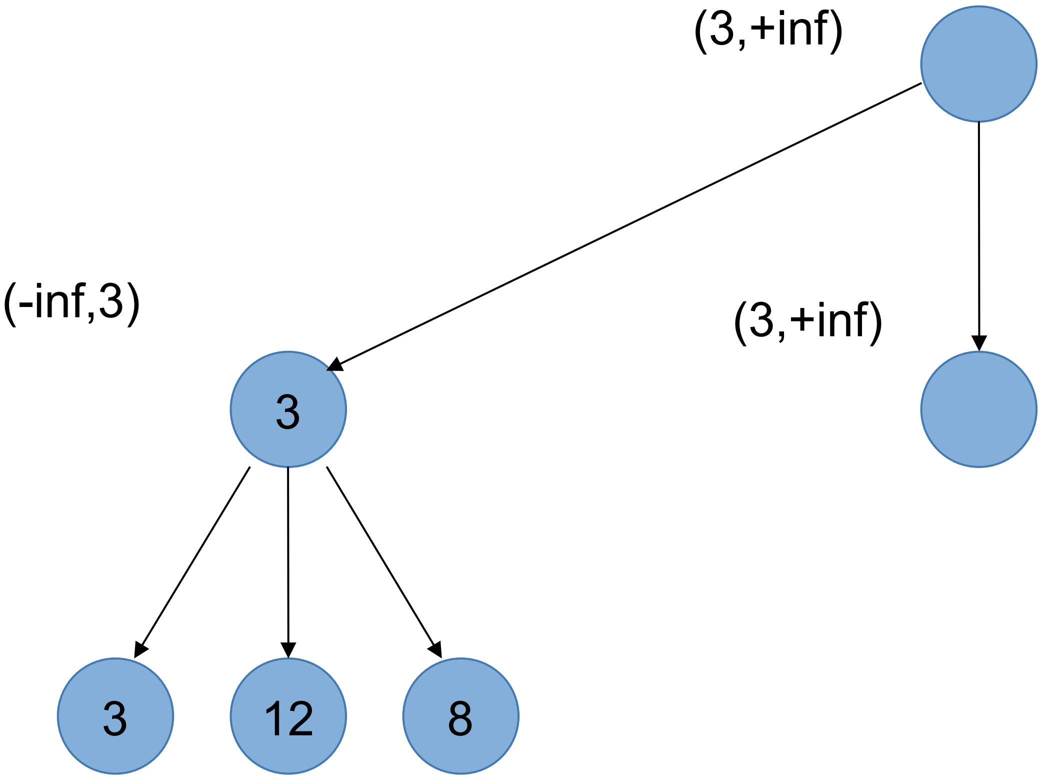


Alpha/Beta pruning

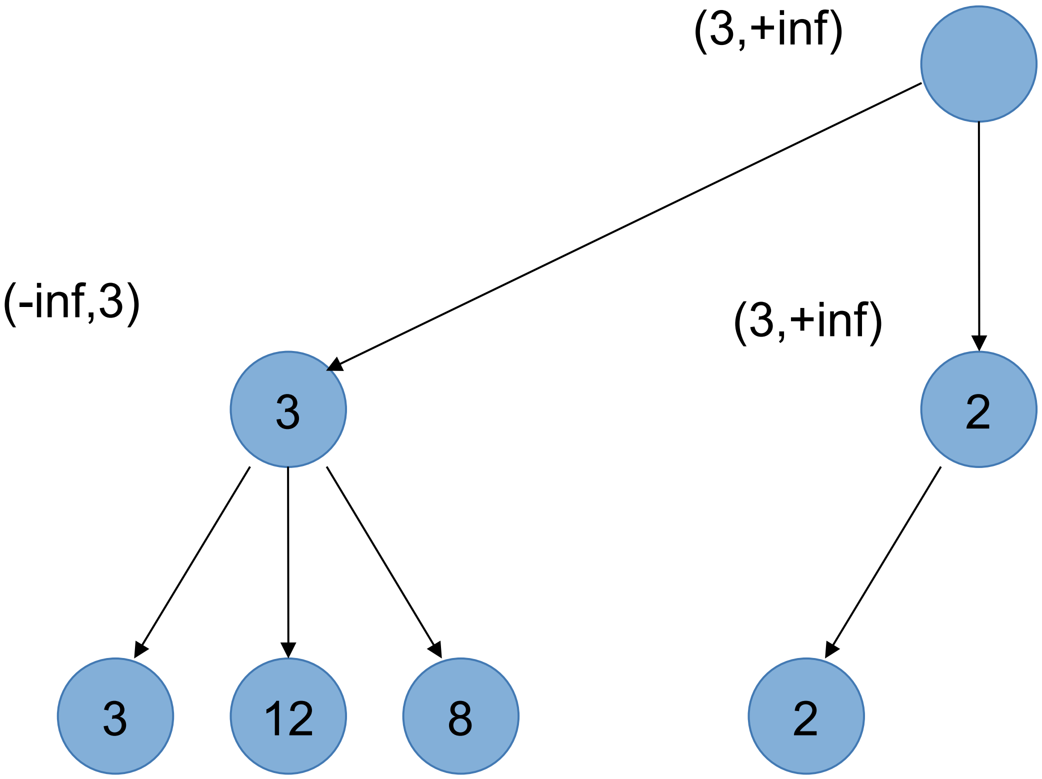


Best value for far for MAX along path to root

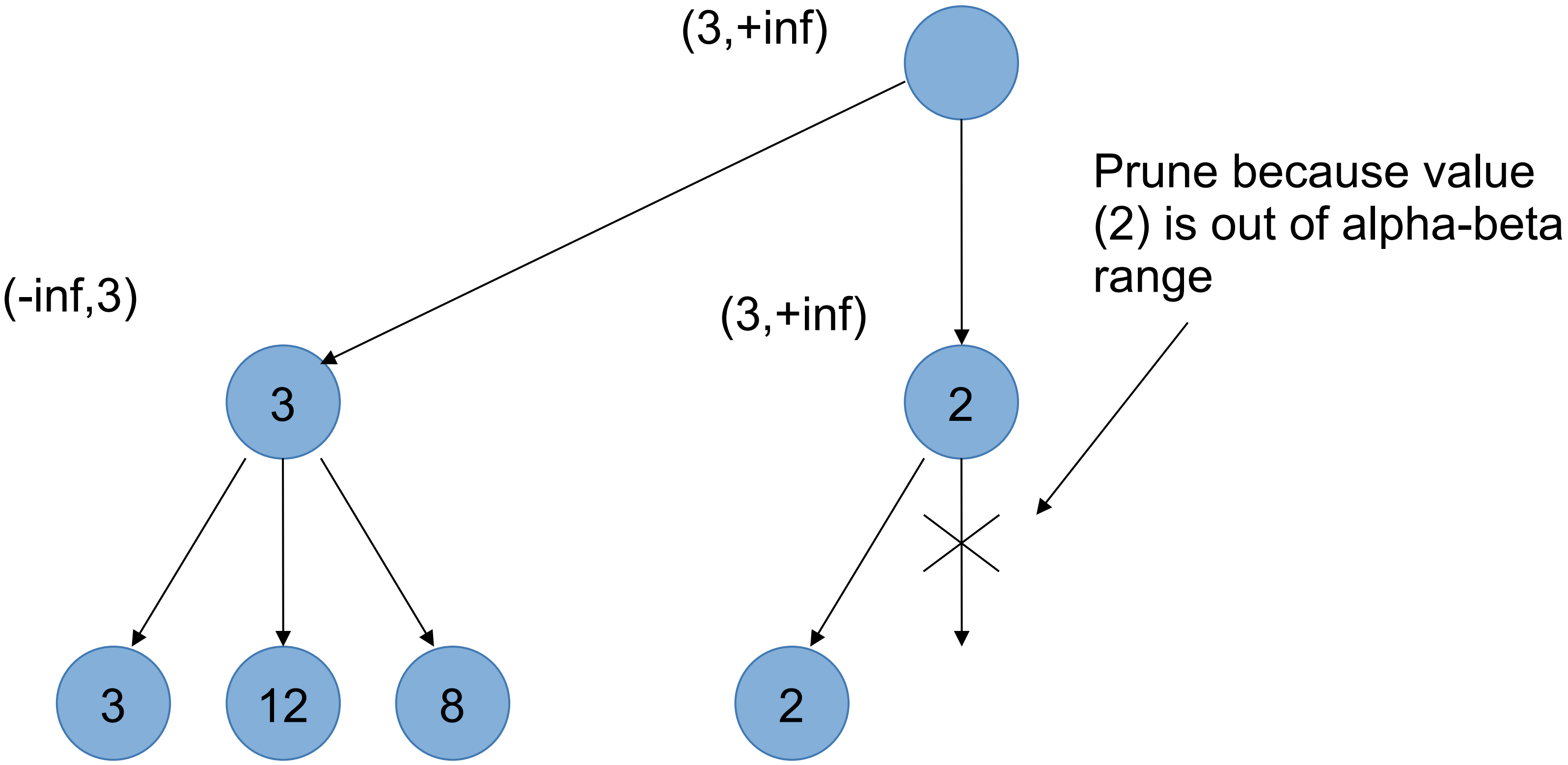
Alpha/Beta pruning



Alpha/Beta pruning

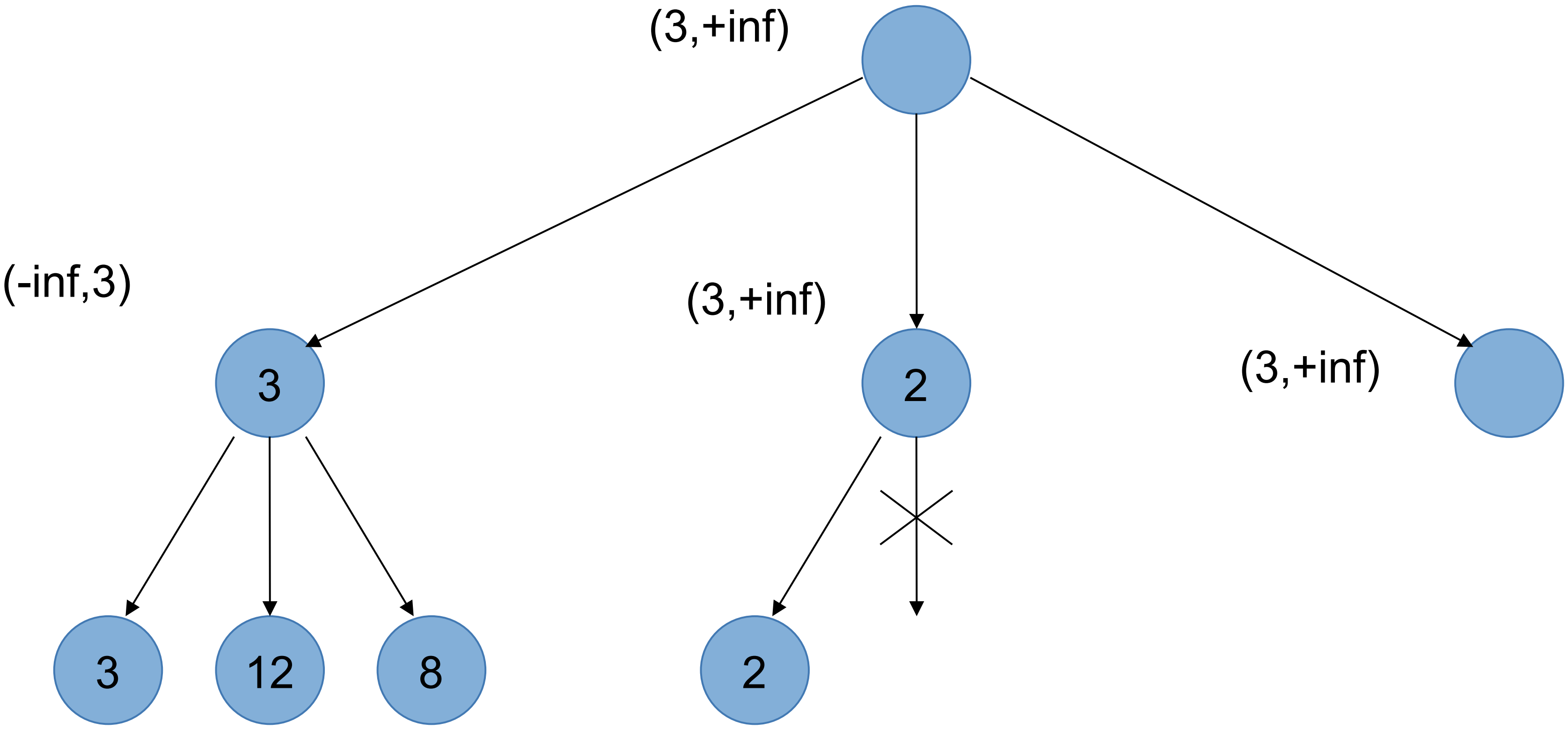


Alpha/Beta pruning

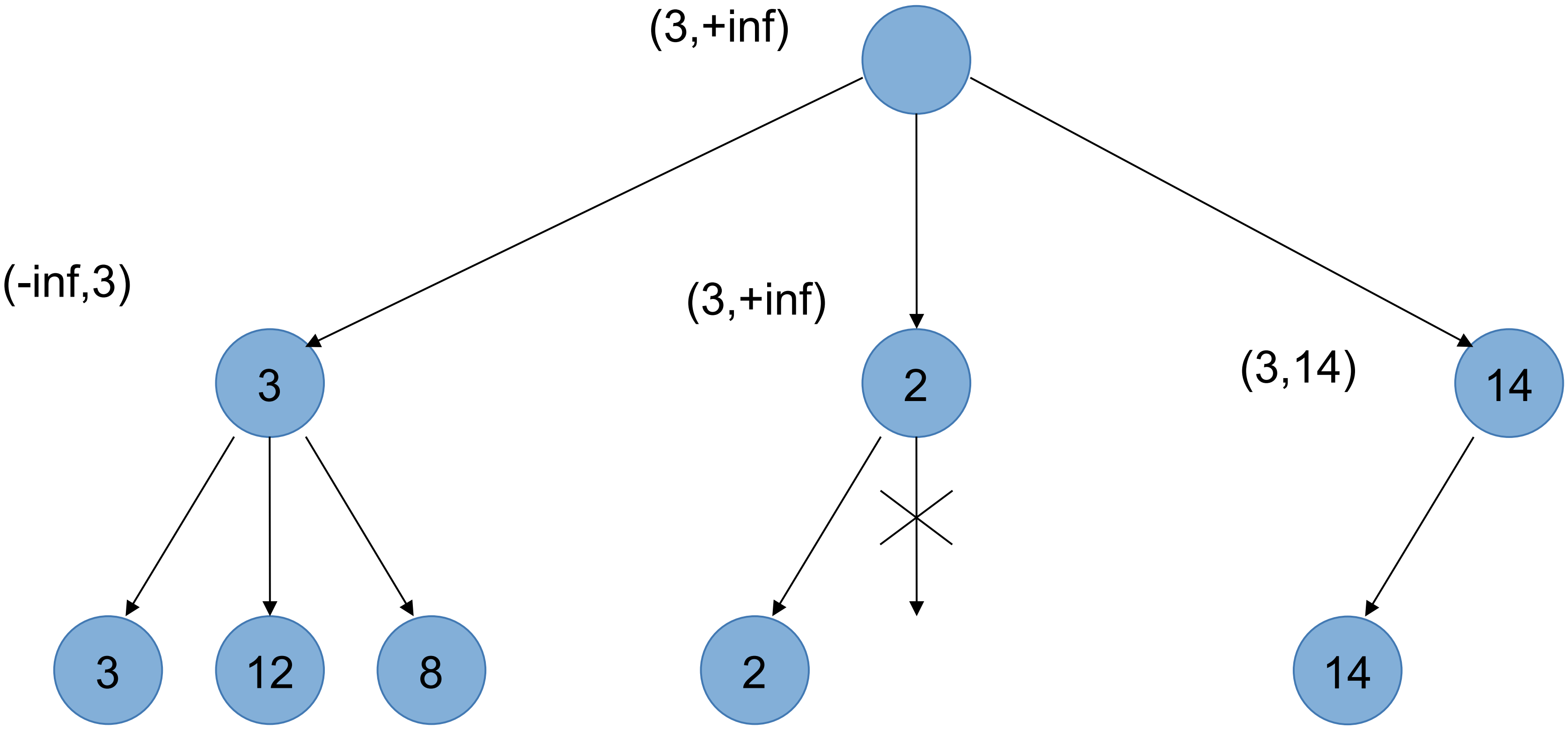


Prune because value (2) is out of alpha-beta range

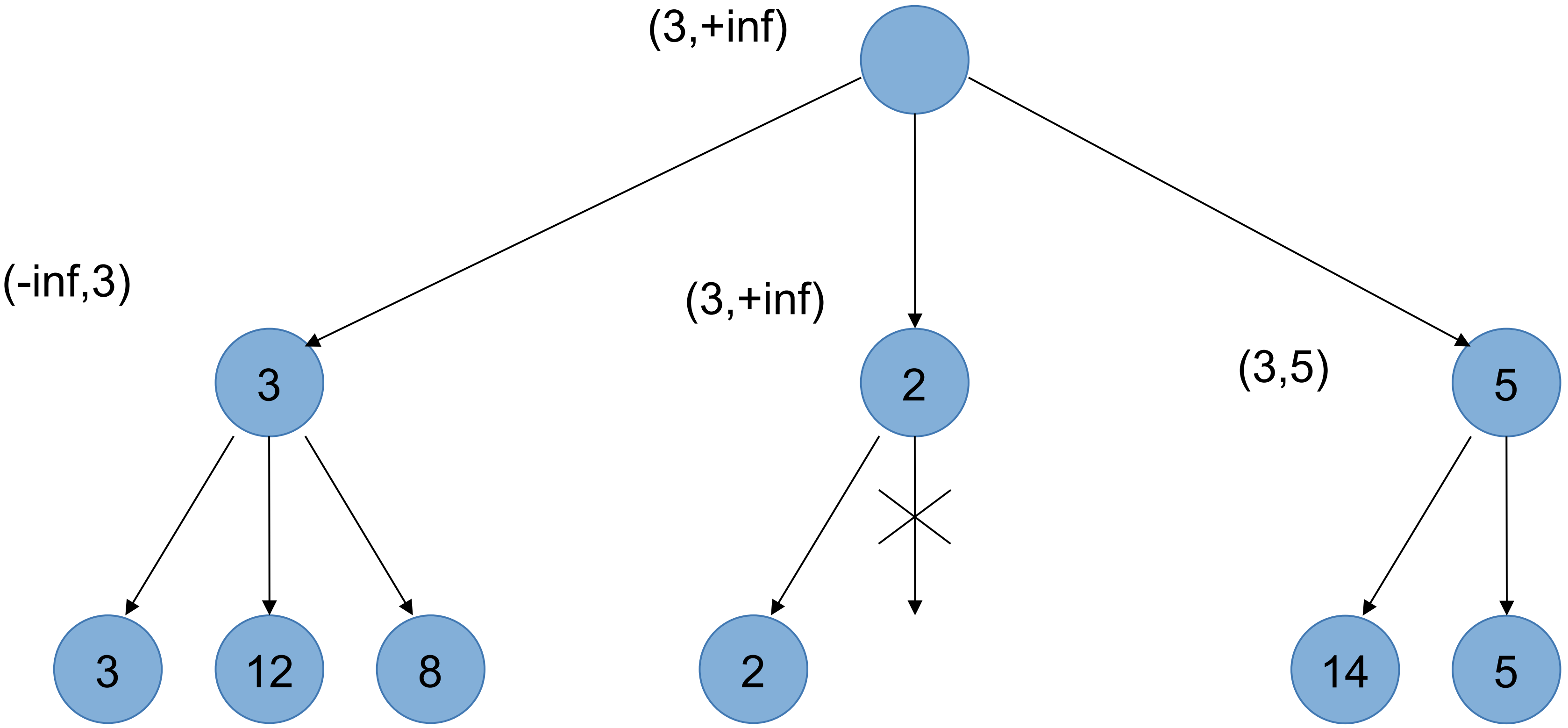
Alpha/Beta pruning



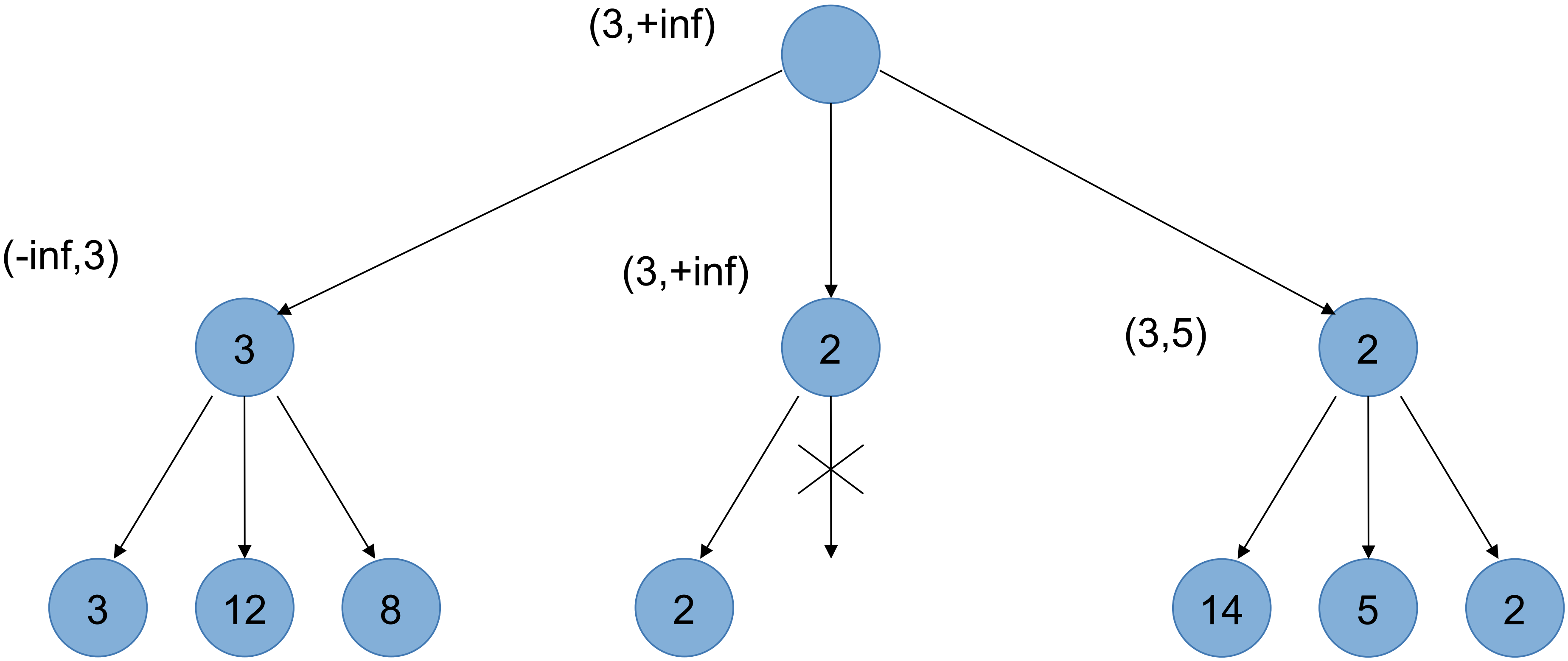
Alpha/Beta pruning



Alpha/Beta pruning



Alpha/Beta pruning



Alpha/Beta pruning: algorithm

```
function ALPHA-BETA-DECISION(state) returns an action
    return the $a$ in ACTIONS(state) maximizing MIN-VALUE(RESULT($a$, state))
```

```
function MAX-VALUE(state, $\alpha$, $\beta$) returns a utility value
    inputs: state, current state in game
            $\alpha$, the value of the best alternative for MAX along the path to state
            $\beta$, the value of the best alternative for MIN along the path to state

    if TERMINAL-TEST(state) then return UTILITY(state)
    $v \leftarrow -\infty$
    for $a$, $s$ in SUCCESSORS(state) do
        $v \leftarrow$ MAX($v$, MIN-VALUE($s$, $\alpha$, $\beta$))
        if $v \geq \beta$ then return $v$
        $\alpha \leftarrow$ MAX($\alpha$, $v$)
    return $v$
```

```
function MIN-VALUE(state, $\alpha$, $\beta$) returns a utility value
    same as MAX-VALUE but with roles of $\alpha$, $\beta$ reversed
```

Alpha/Beta properties

Is it complete?

Alpha/Beta properties

Is it complete?

How much does alpha/beta help relative to minimax?

Minimax time complexity = $O(b^m)$

Alpha/beta time complexity >= $O(b^{\frac{m}{2}})$

– the improvement w/ alpha/beta depends upon move ordering...

Alpha/Beta properties

Is it complete?

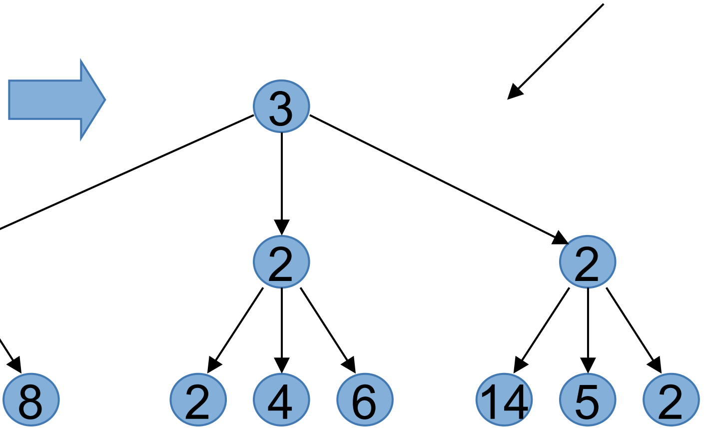
How much does alpha/beta help relative to minimax?

Minimax time complexity = $O(b^m)$

Alpha/beta time complexity >= $O(b^{\frac{m}{2}})$

– the improvement w/ alpha/beta depends upon move ordering...

The order in which we expand a node.



Alpha/Beta properties

Is it complete?

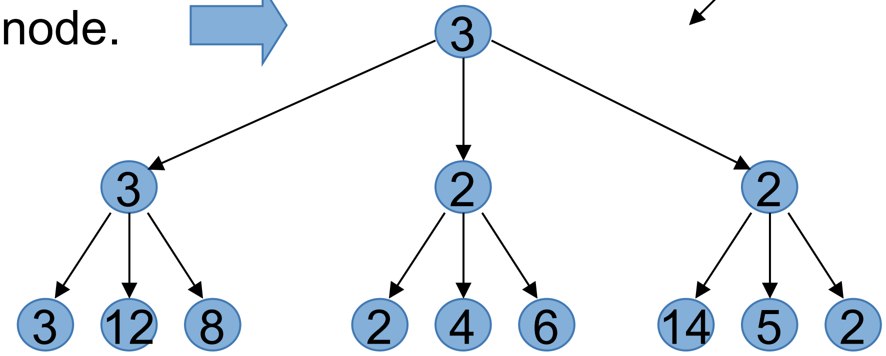
How much does alpha/beta help relative to minimax?

Minimax time complexity = $O(b^m)$

Alpha/beta time complexity >= $O(b^{\frac{m}{2}})$

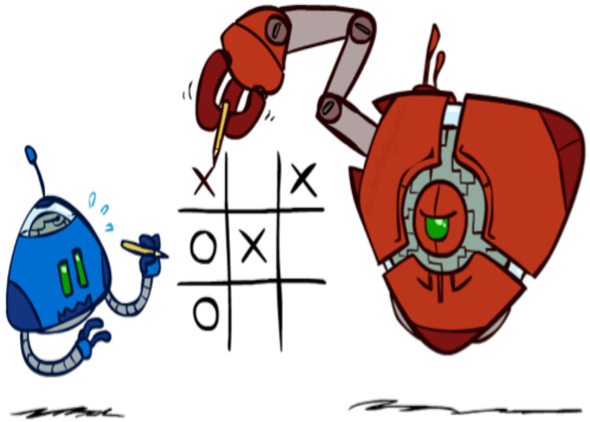
– the improvement w/ alpha/beta depends upon move ordering...

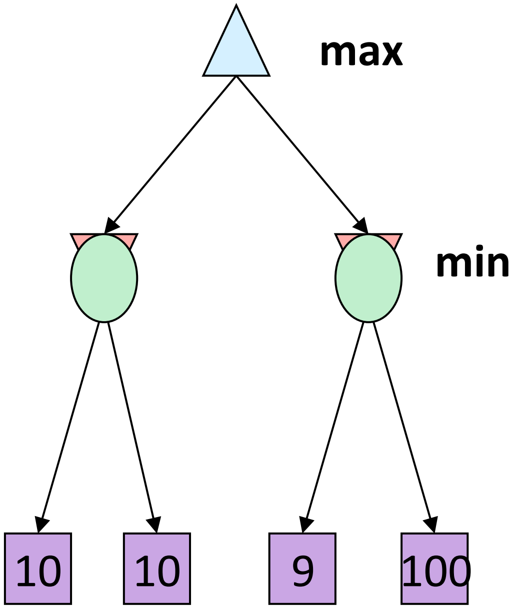
The order in which we expand a node.

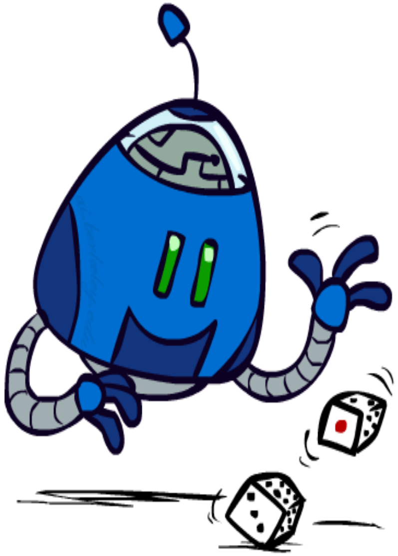


How to choose move ordering? Use IDS.
– on each iteration of IDS, use prior run to inform ordering of next node expansions.

Worst-Case vs. Average Case





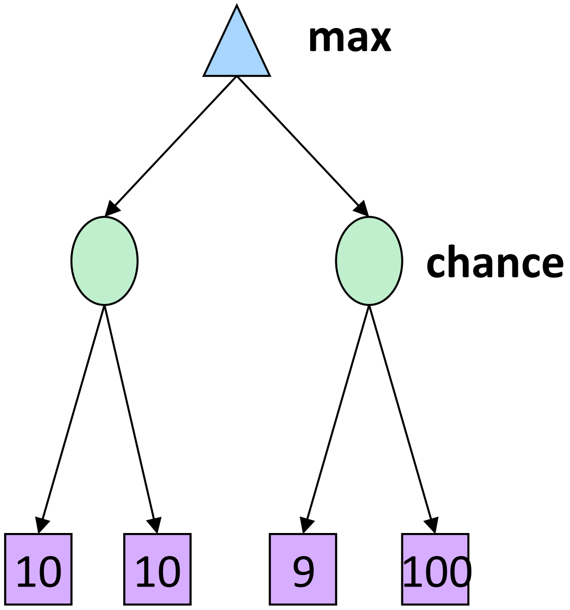


Idea: Uncertain outcomes controlled by chance, not an adversary!

Expectimax search

Why wouldn't we know the result of an action?

- Explicit randomness: rolling dice
- Unpredictable opponents: the ghosts respond randomly
- Actions can fail: when moving a robot, wheels may slip



Values should now reflect average-case (expectimax) outcomes, not worst-case (minimax) outcomes

Expectimax search: compute the average score under optimal play

- Max nodes as in minimax search
- Chance nodes are like min nodes but the outcome is uncertain
- Calculate their expected utilities
- I.e. take weighted average (expectation) of children

Later, we'll learn how to formalize the underlying uncertain-result problems as Markov Decision Processes

Expectimax demo (min)

Expectimax demo (exp)

Expectimax pseudocode

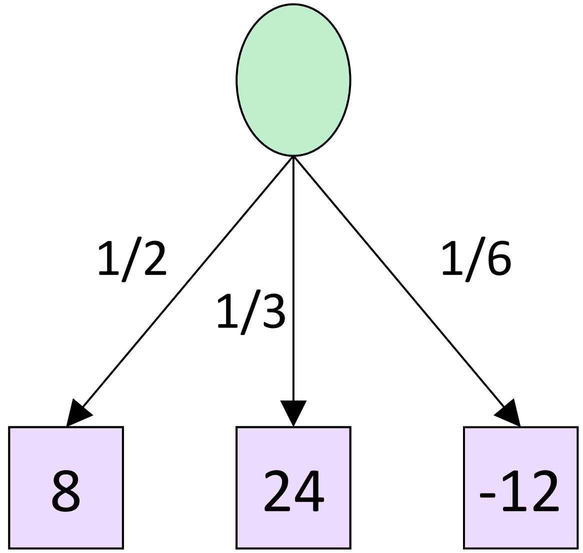
```
def value(state):
    if the state is a terminal state: return the state’s utility
    if the next agent is MAX: return max-value(state)
    if the next agent is EXP: return exp-value(state)
```

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

```
def exp-value(state):
    initialize v = 0
    for each successor of state:
        p = probability(successor)
        v += p * value(successor)
    return v
```

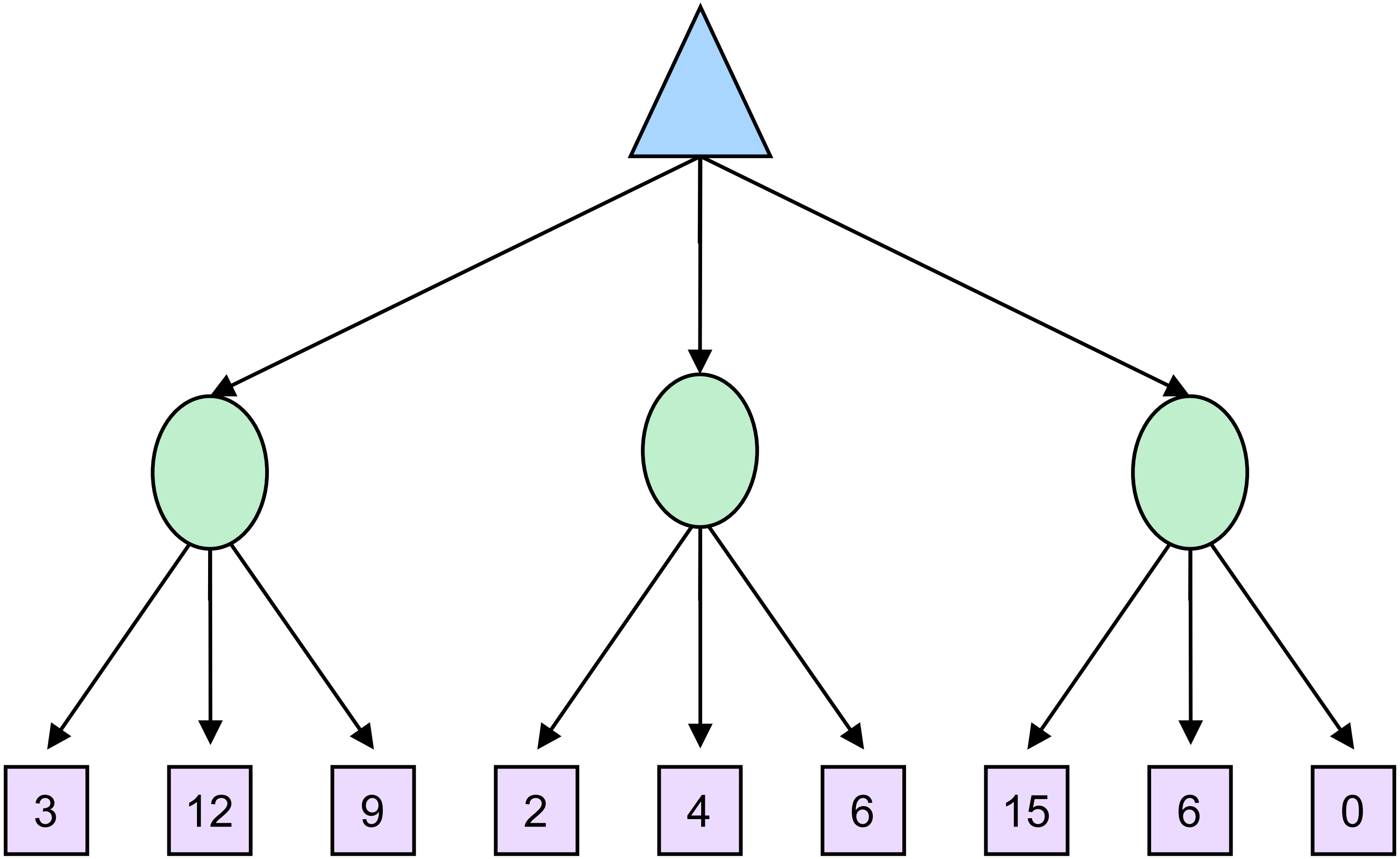
Expectimax pseudocode

```
def exp-value(state):
    initialize v = 0
    for each successor of state:
        p = probability(successor)
        v += p * value(successor)
    return v
```

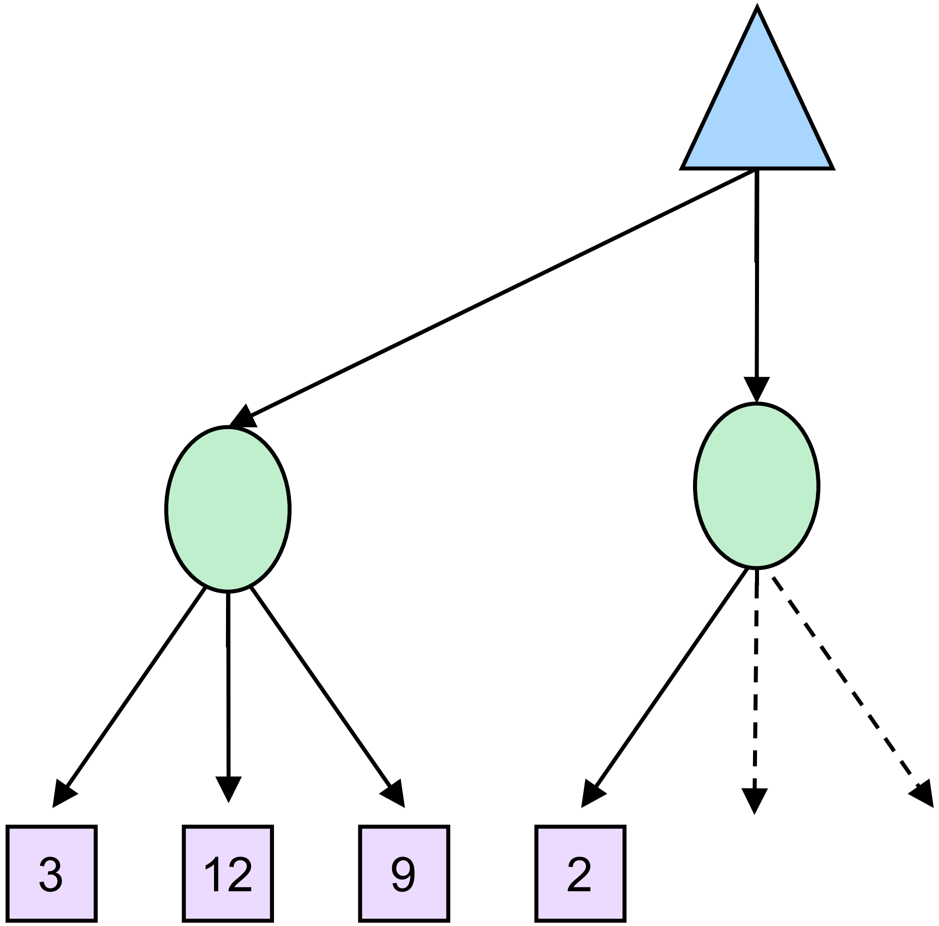


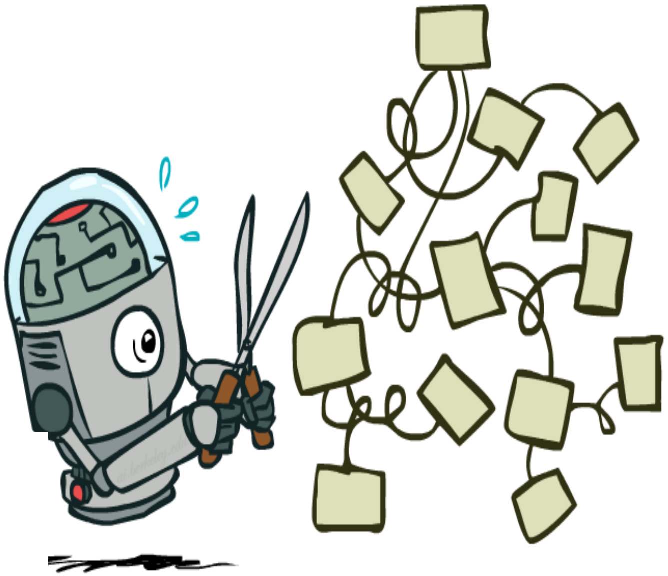
$v = (1/2)\,(8) + (1/3)\,(24) + (1/6)\,(-12) = 10$

Expectimax example

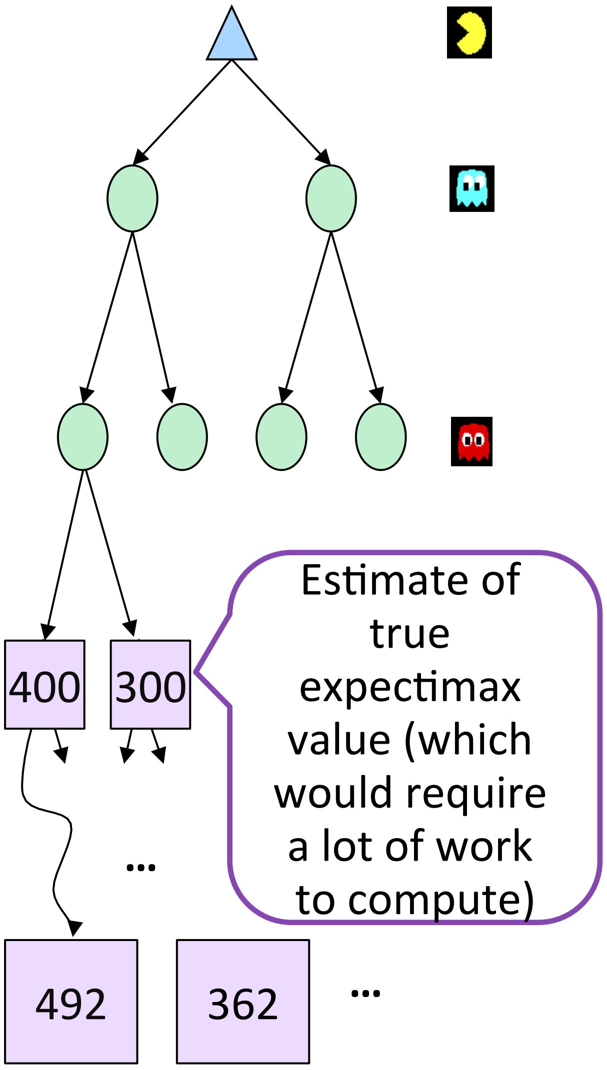


Expectimax pruning?





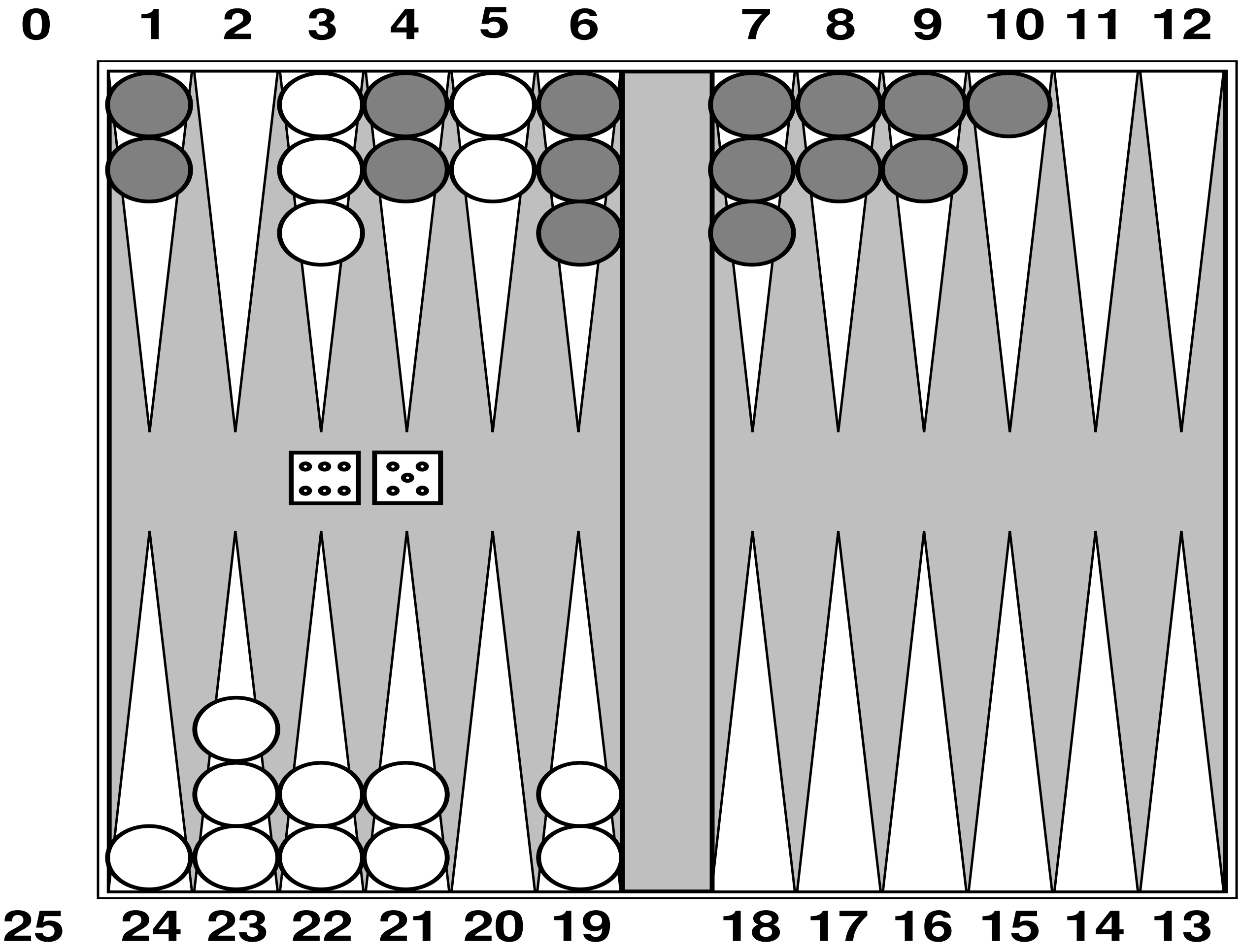
Depth-limited expectimax



Estimate of true expectimax value (which would require a lot of work to compute)

Mixing these ideas: Nondeterministic games

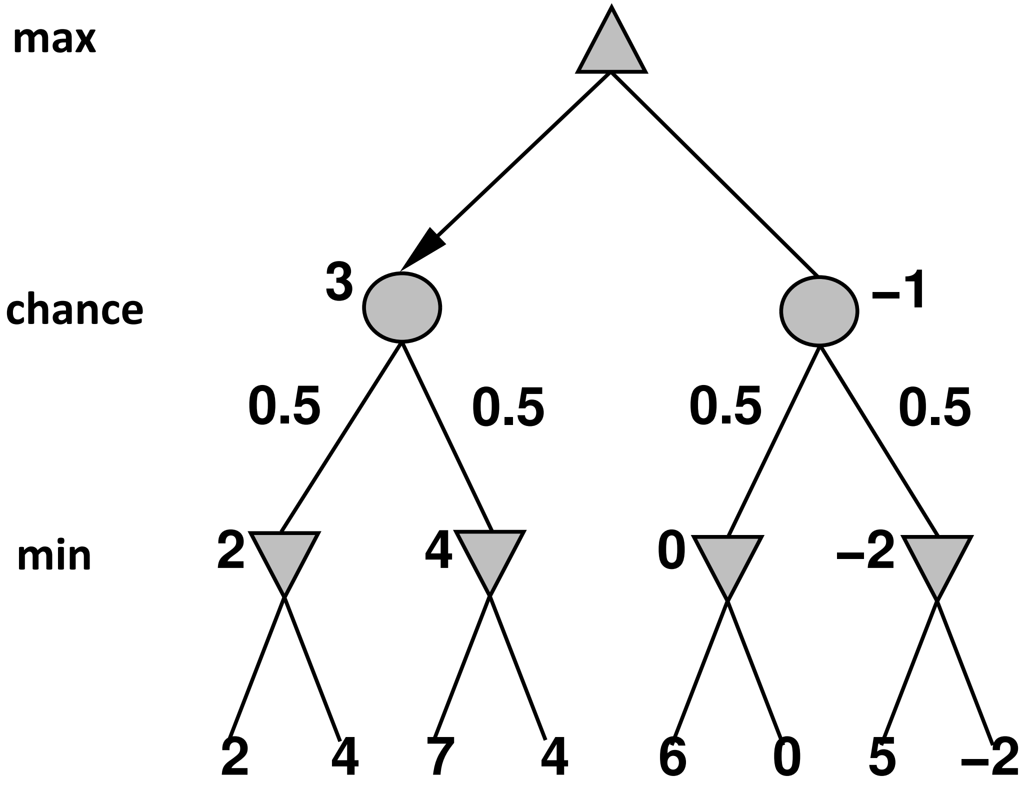
Backgammon



Nondeterministic games in general

In nondeterministic games, chance introduced by dice, card-shuffling

Simplified example with coin-flipping:



Algorithm for nondeterministic games

Expectiminimax gives perfect play

Just like Minimax, except we must also handle chance nodes:

```
...

if state is a Max node then
    return the highest ExpectiMinimax-Value of Successors(state)

if state is a Min node then
    return the lowest ExpectiMinimax-Value of Successors(state)

if state is a chance node then
    return average of ExpectiMinimax-Value of Successors(state)

...
```

Nondeterministic games in practice

Dice rolls increase *b*: 21 possible rolls with 2 dice

Backgammon ≈ 20 legal moves

depth 4=$20\times(21\times20)^3 \approx 1.2\times10^9$

As depth increases, probability of reaching a given node shrinks

⇒ value of lookahead is diminished

α–β pruning is much less effective

TDGammon uses depth-2 search + very good Eval ≈ world-champion level

Adversarial search: summary

Games are fun to work on! (and dangerous)

They illustrate several important points about AI

- perfection is unattainable ⇒ must approximate
- good idea to think about what to think about
- uncertainty constrains the assignment of values to states
- optimal decisions depend on information state, not real state

Games are to AI as grand prix racing is to automobile design