Adversarial Search

Rob Platt
Northeastern University

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

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

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

Different types of games

Deterministic / stochastic

Two player / multi player?

Zero-sum / non zero-sum

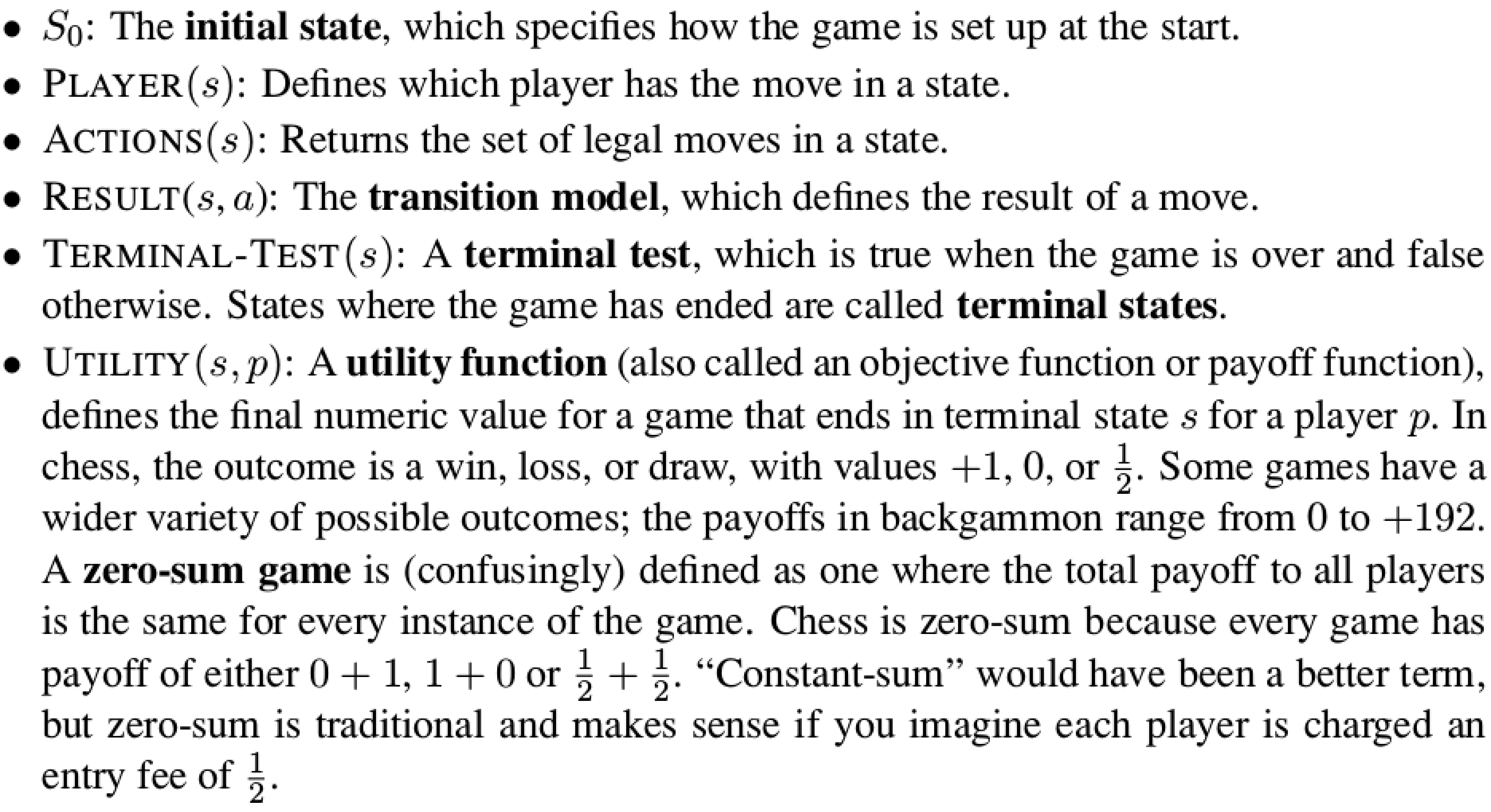
Perfect information / imperfect infor

Zero Sum:
– utilities of all players sum to zero
– pure competition

Non-Zero Sum:
– utility function of each play could be arbitrary
– optimal strategies could involve cooperation

Formalizing a Game

Given:

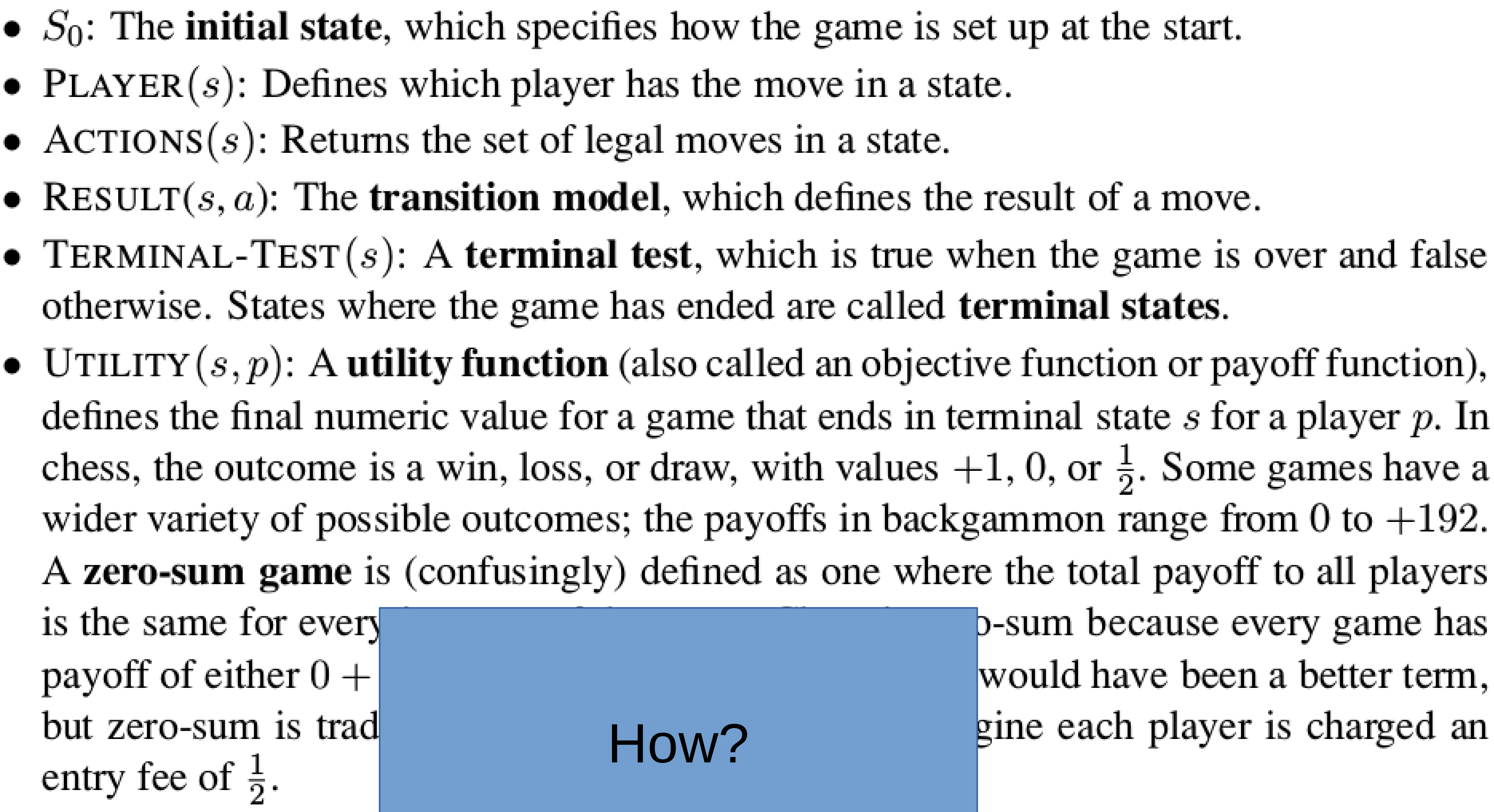


- S_0: The **initial state**, which specifies how the game is set up at the start.
- $\text{PLAYER}(s)$: Defines which player has the move in a state.
- $\text{ACTIONS}(s)$: Returns the set of legal moves in a state.
- $\text{RESULT}(s, a)$: The **transition model**, which defines the result of a move.
- $\text{TERMINAL-TEST}(s)$: A **terminal test**, which is true when the game is over and false otherwise. States where the game has ended are called **terminal states**.
- $\text{UTILITY}(s, p)$: A **utility function** (also called an objective function or payoff function), defines the final numeric value for a game that ends in terminal state s for a player p. In chess, the outcome is a win, loss, or draw, with values $+1$, 0, or $\frac{1}{2}$. Some games have a wider variety of possible outcomes; the payoffs in backgammon range from 0 to $+192$. A **zero-sum game** is (confusingly) defined as one where the total payoff to all players is the same for every instance of the game. Chess is zero-sum because every game has payoff of either $0 + 1$, $1 + 0$ or $\frac{1}{2} + \frac{1}{2}$. "Constant-sum" would have been a better term, but zero-sum is traditional and makes sense if you imagine each player is charged an entry fee of $\frac{1}{2}$.

Calculate a policy: $\pi(s, p) \longleftarrow$ Action that player p should take from state s

Formalizing a Game

Given:



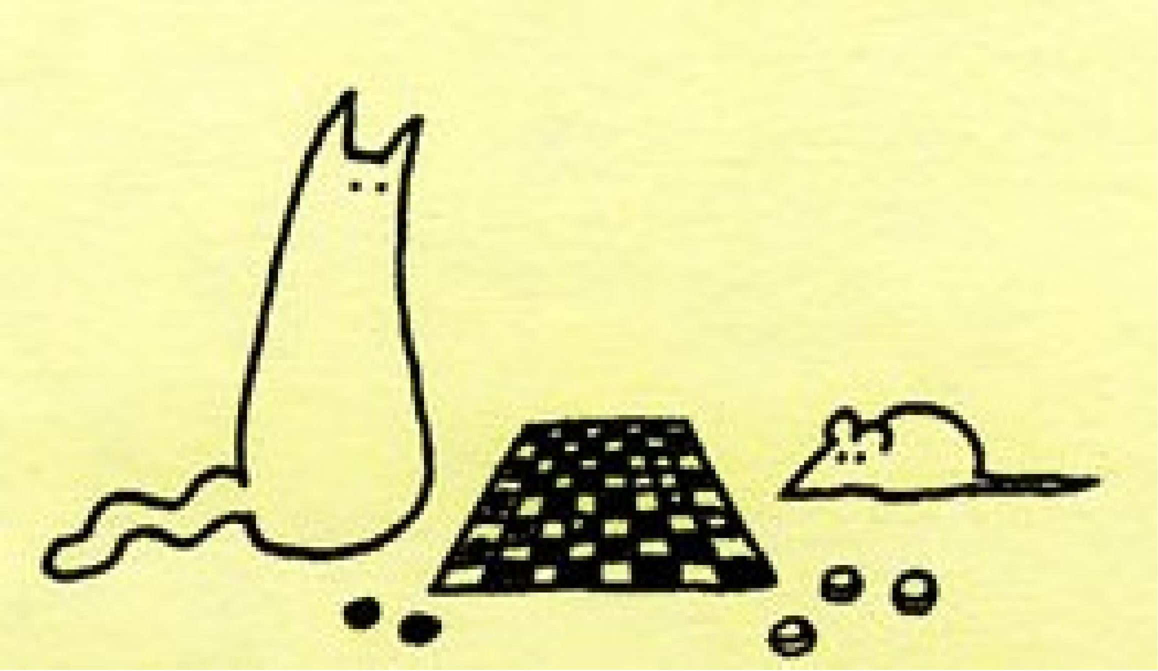
- S_0: The **initial state**, which specifies how the game is set up at the start.
- PLAYER(s): Defines which player has the move in a state.
- ACTIONS(s): Returns the set of legal moves in a state.
- RESULT(s, a): The **transition model**, which defines the result of a move.
- TERMINAL-TEST(s): A **terminal test**, which is true when the game is over and false otherwise. States where the game has ended are called **terminal states**.
- UTILITY(s, p): A **utility function** (also called an objective function or payoff function), defines the final numeric value for a game that ends in terminal state s for a player p. In chess, the outcome is a win, loss, or draw, with values $+1$, 0, or $\frac{1}{2}$. Some games have a wider variety of possible outcomes; the payoffs in backgammon range from 0 to $+192$. A **zero-sum game** is (confusingly) defined as one where the total payoff to all players
  is the same for every
  o-sum because every game has
  payoff of either $0 +$
  would have been a better term,
  but zero-sum is trad
  gine each player is charged an
  entry fee of $\frac{1}{2}$.

How?

Calculate a policy: $\pi(s, p)$

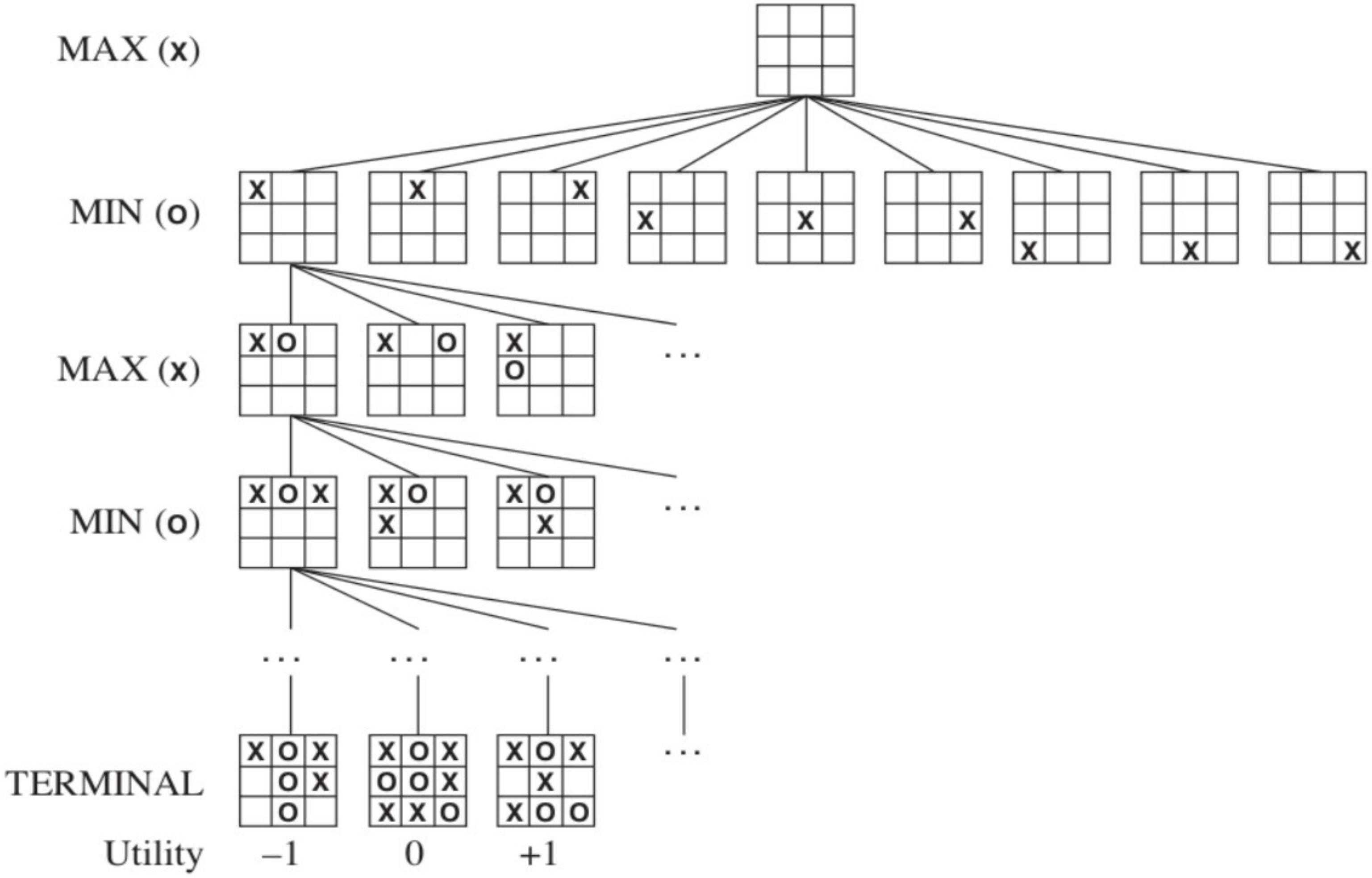
Action that player p
should take from state s

How solve for a policy?

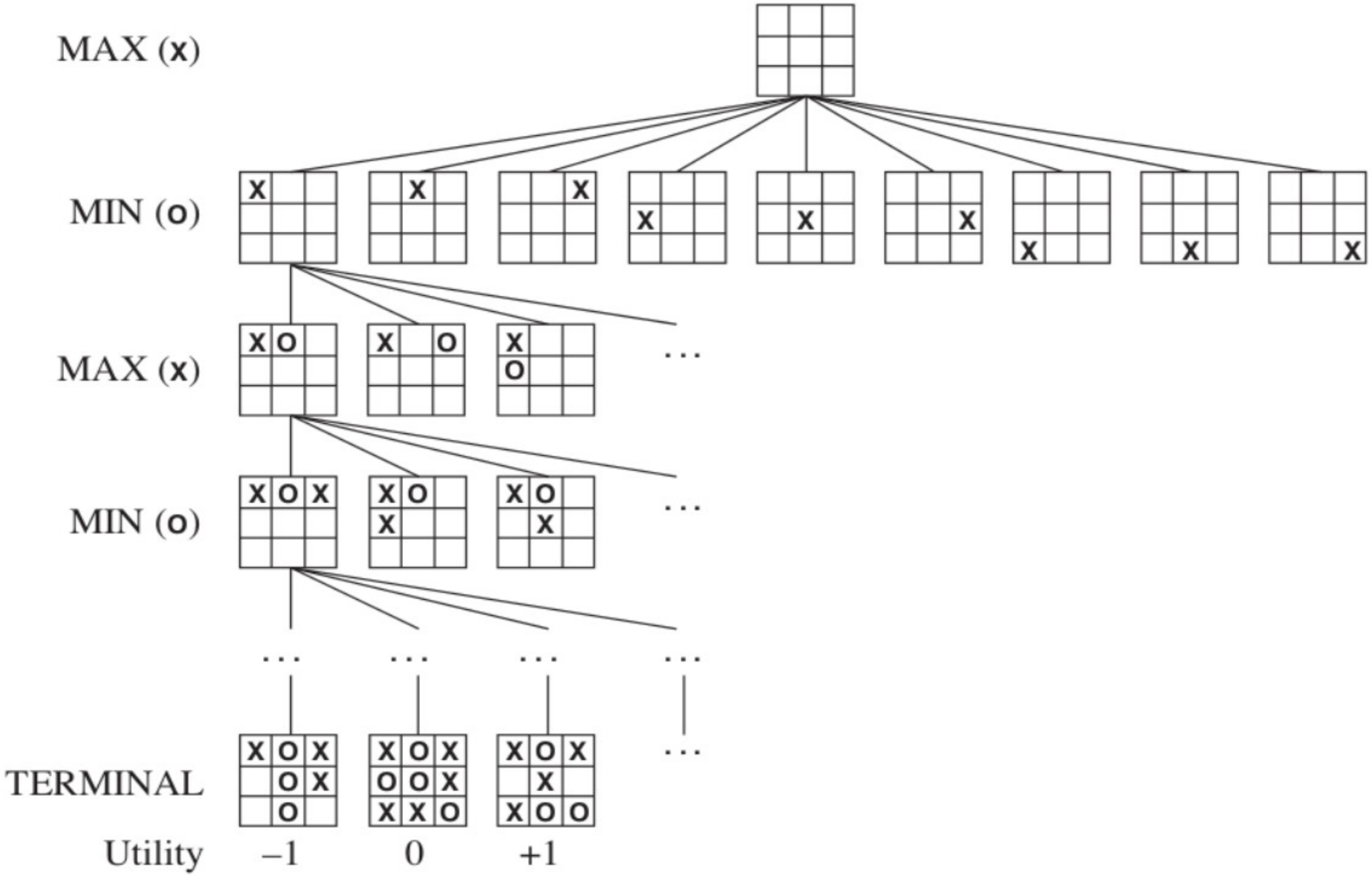


Use adversarial search!
– build a <u>game tree</u>

This is a game tree for tic-tac-toe

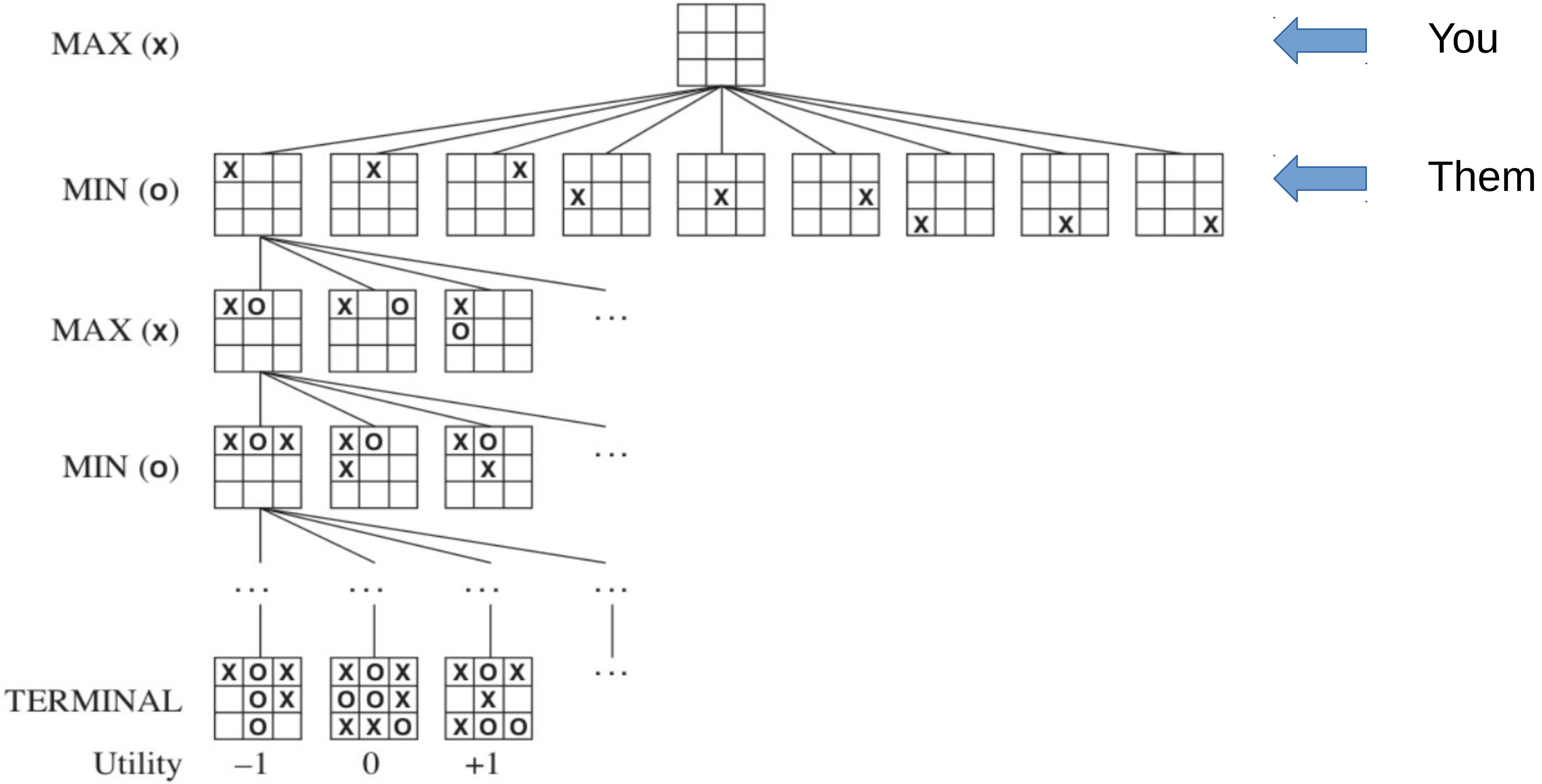


This is a game tree for tic-tac-toe



You

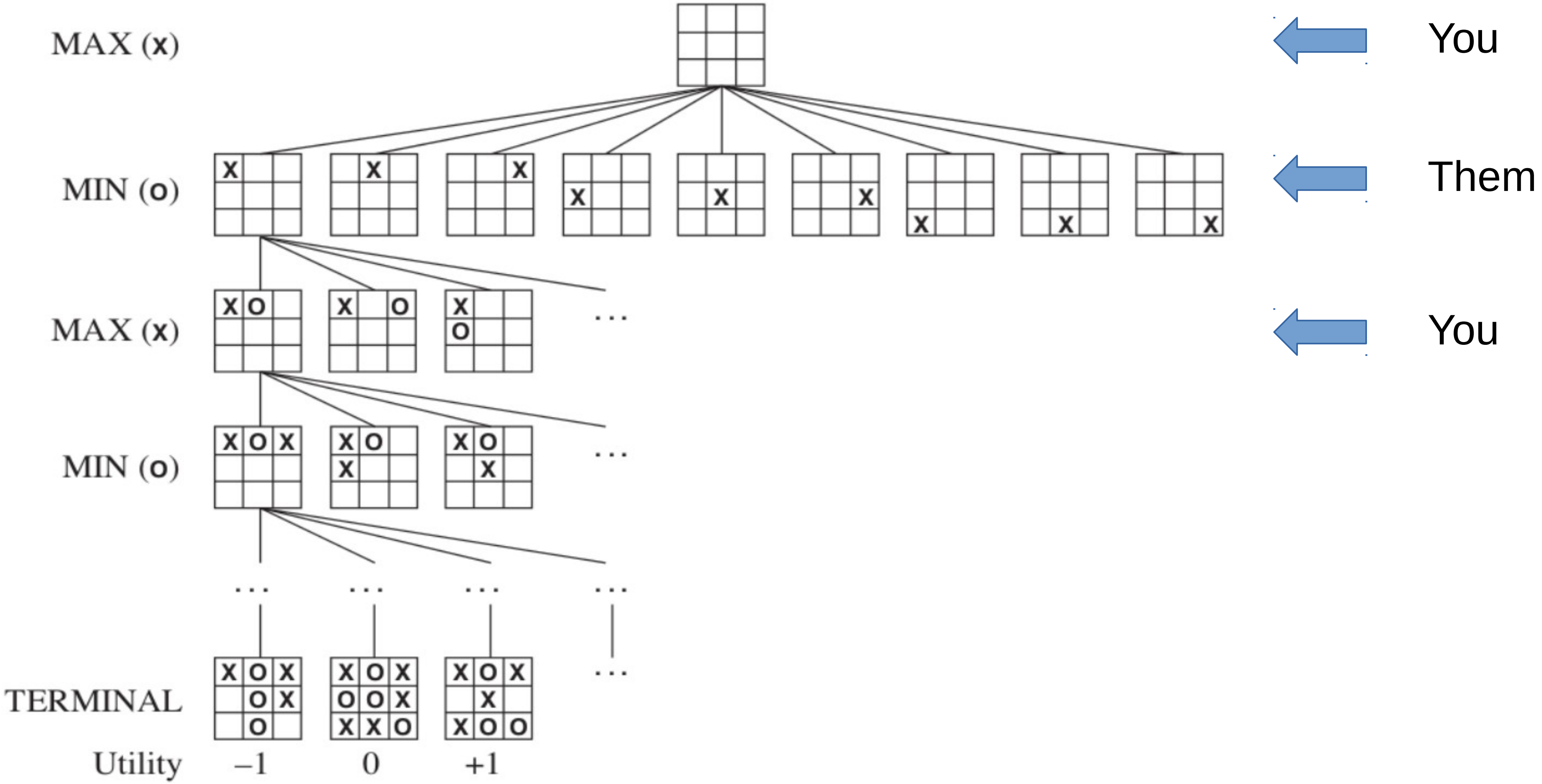
This is a game tree for tic-tac-toe



You

Them

This is a game tree for tic-tac-toe

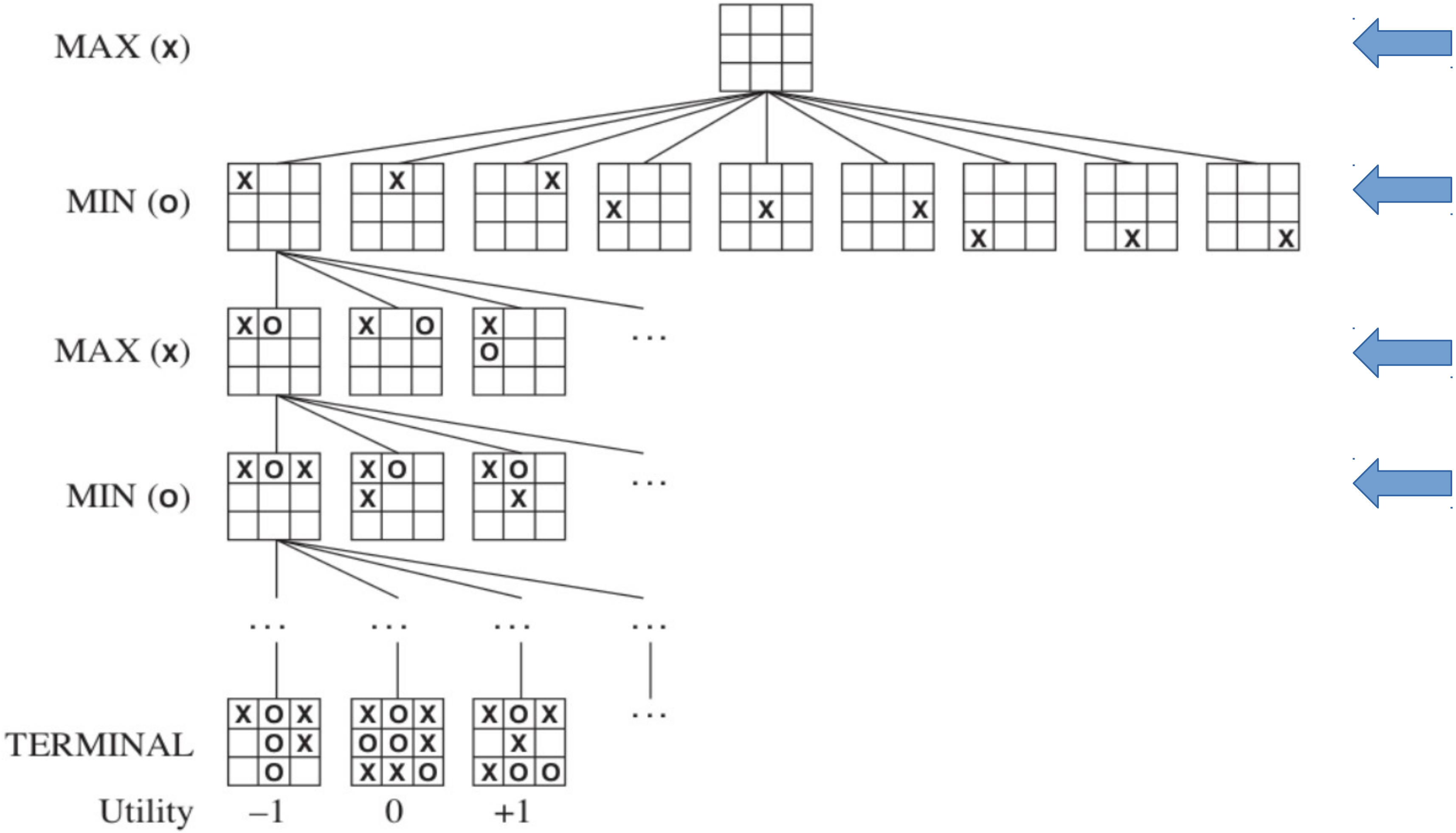


You

Them

You

This is a game tree for tic-tac-toe



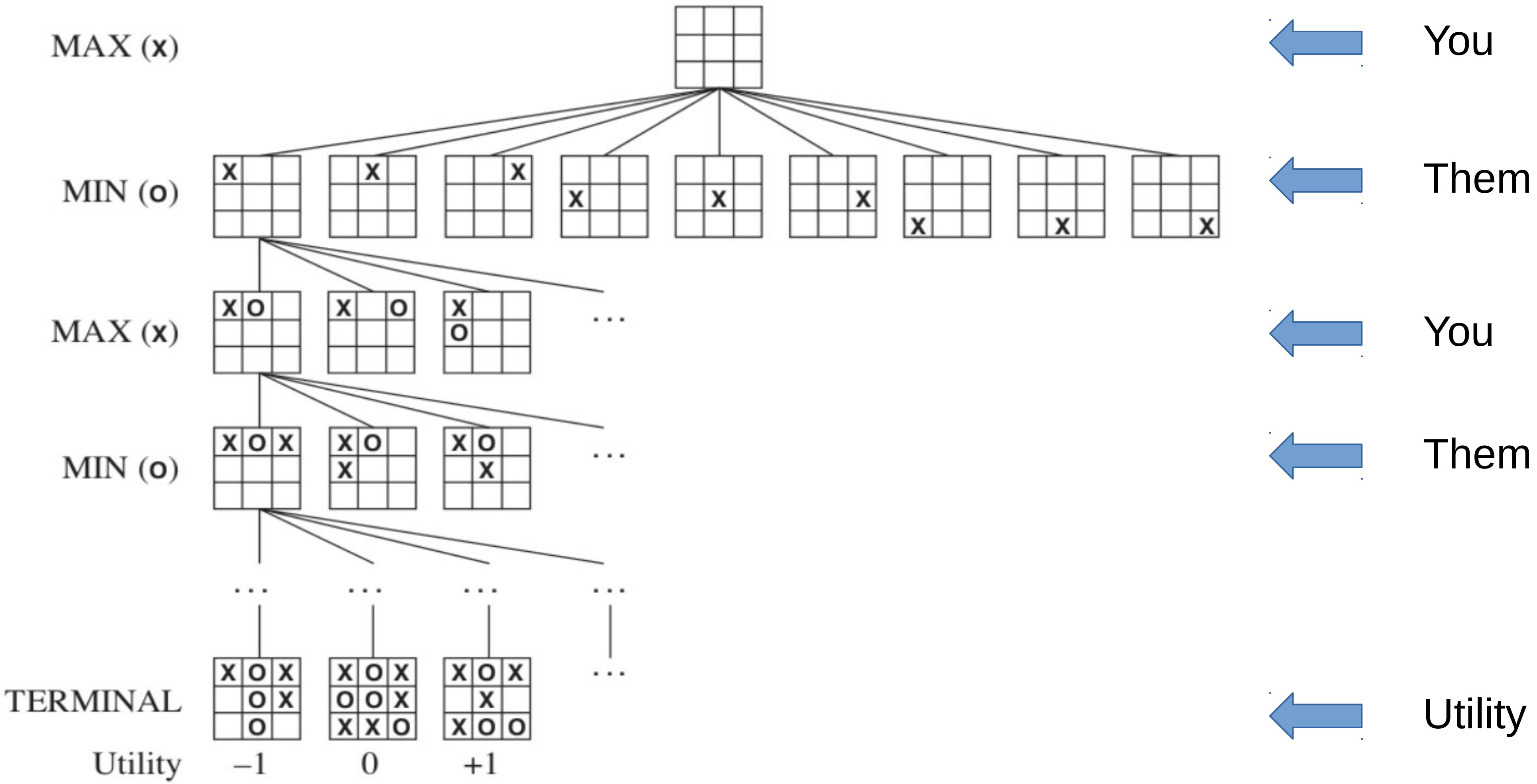
You

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:

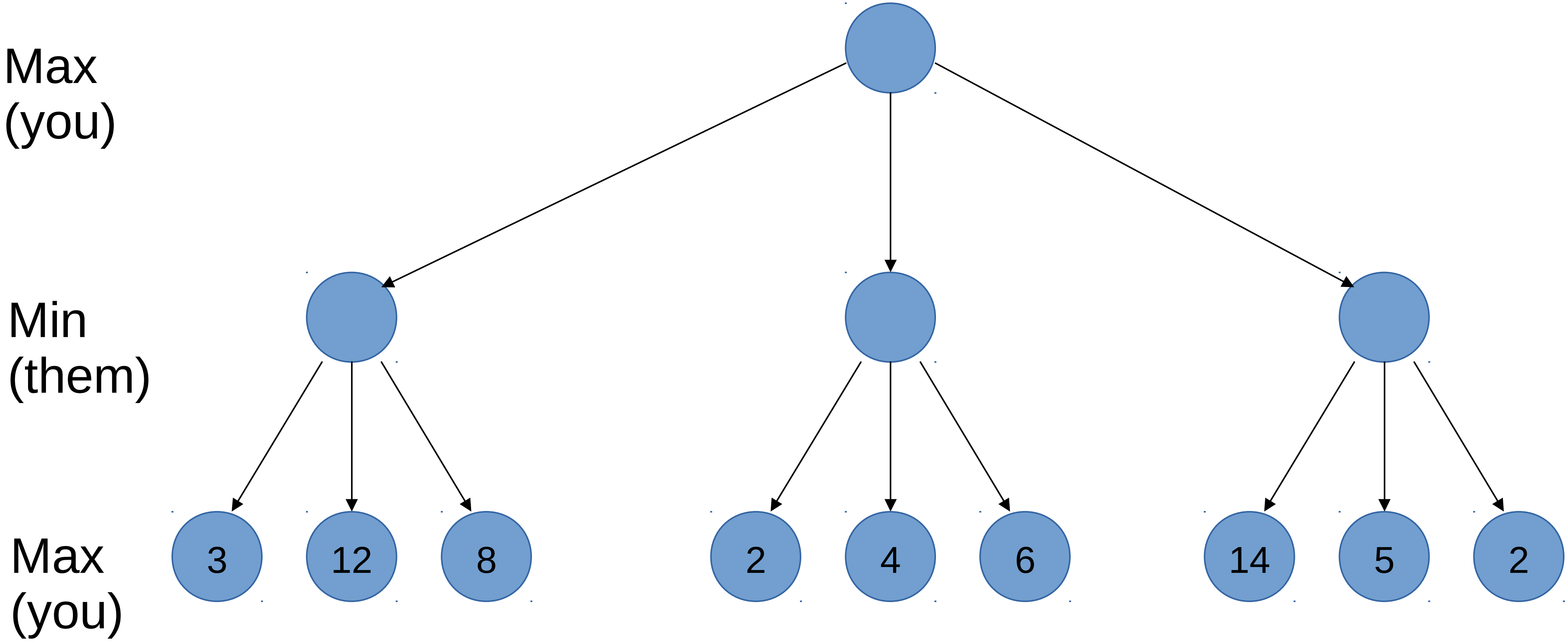
1. you make a move
2. your opponent makes a move
3. game ends

What does the minimax tree look like in this case?

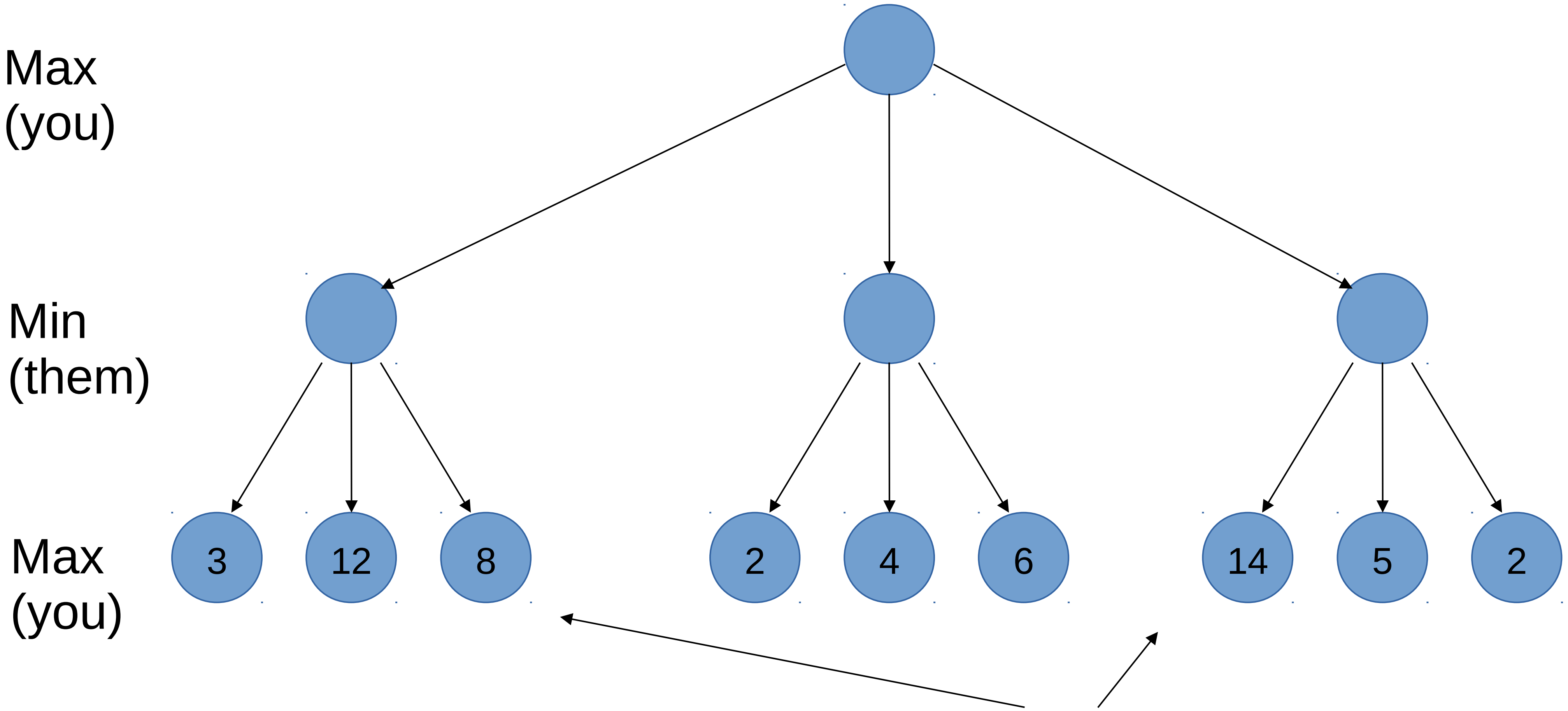
What is Minimax?

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?



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

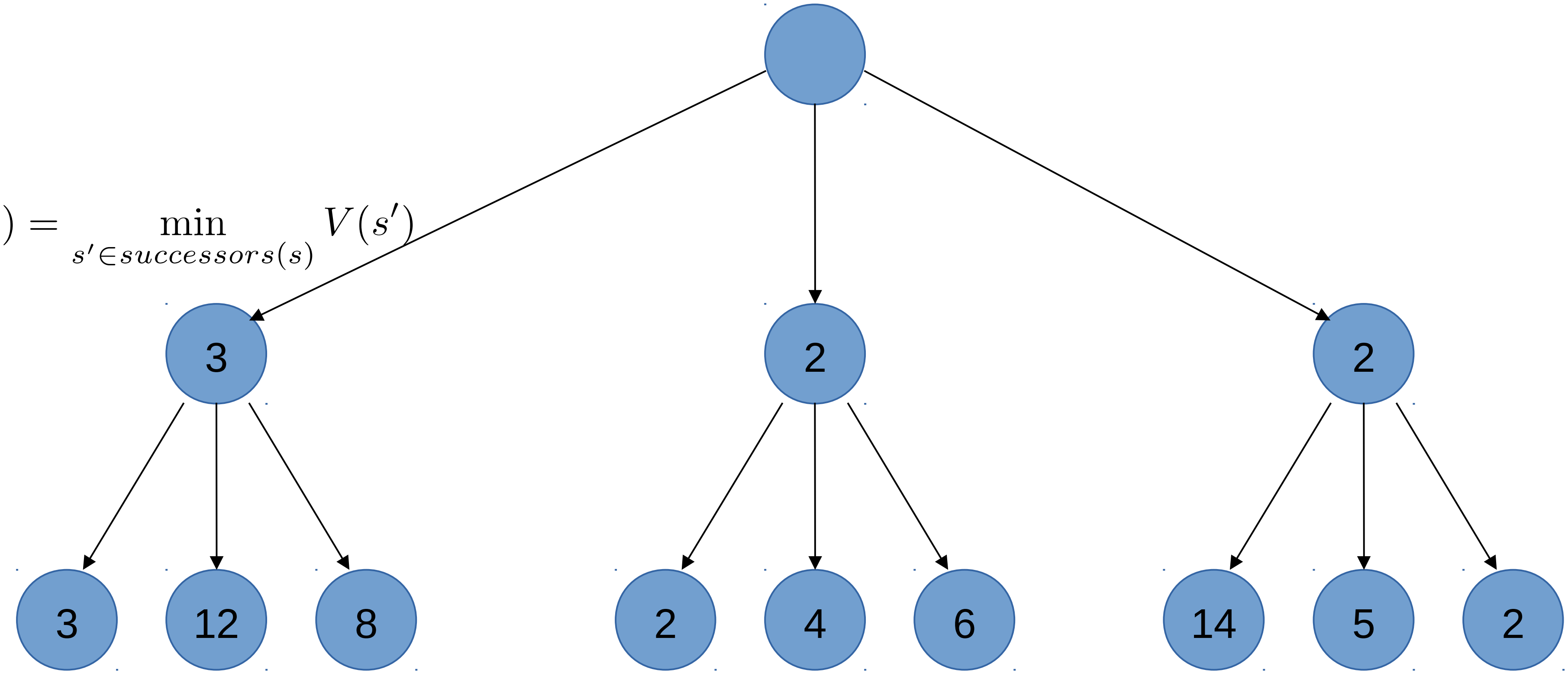
What is Minimax?

Max
(you)

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

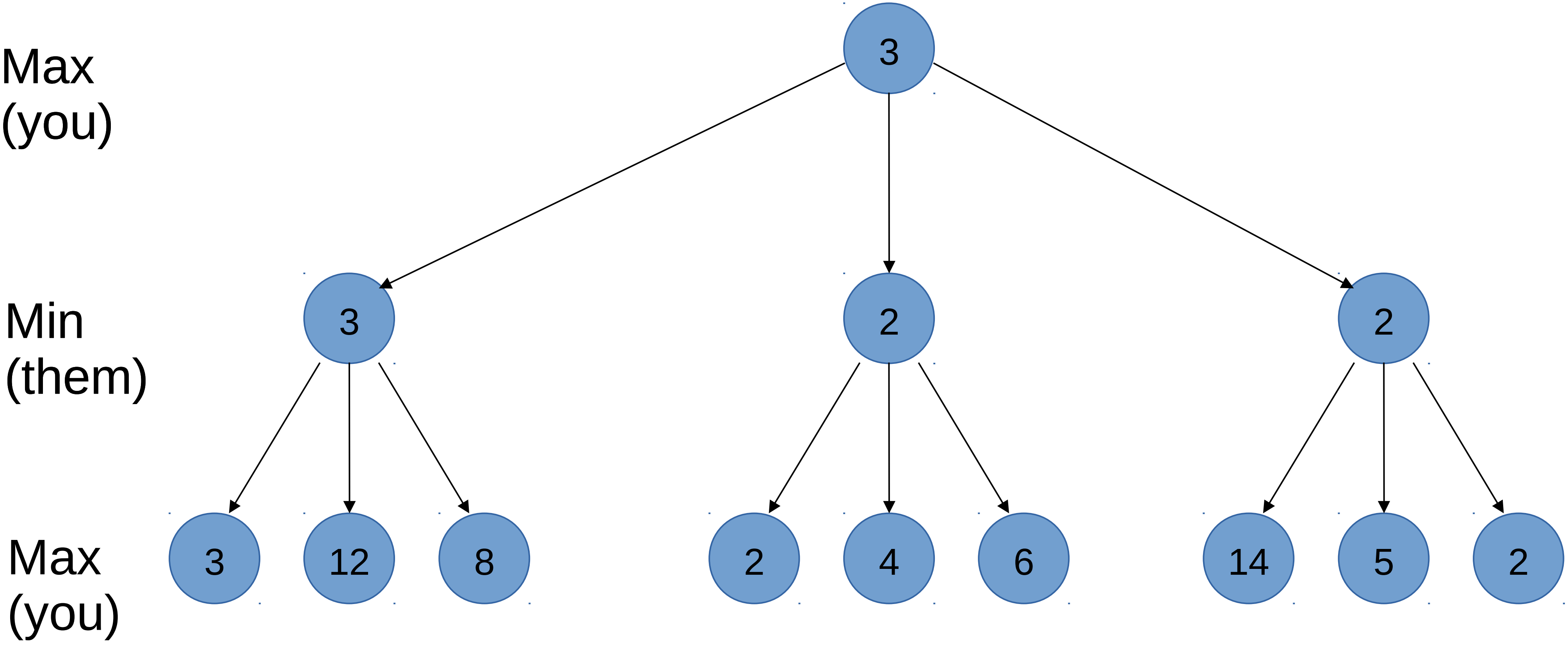
Min
(them)

Max
(you)



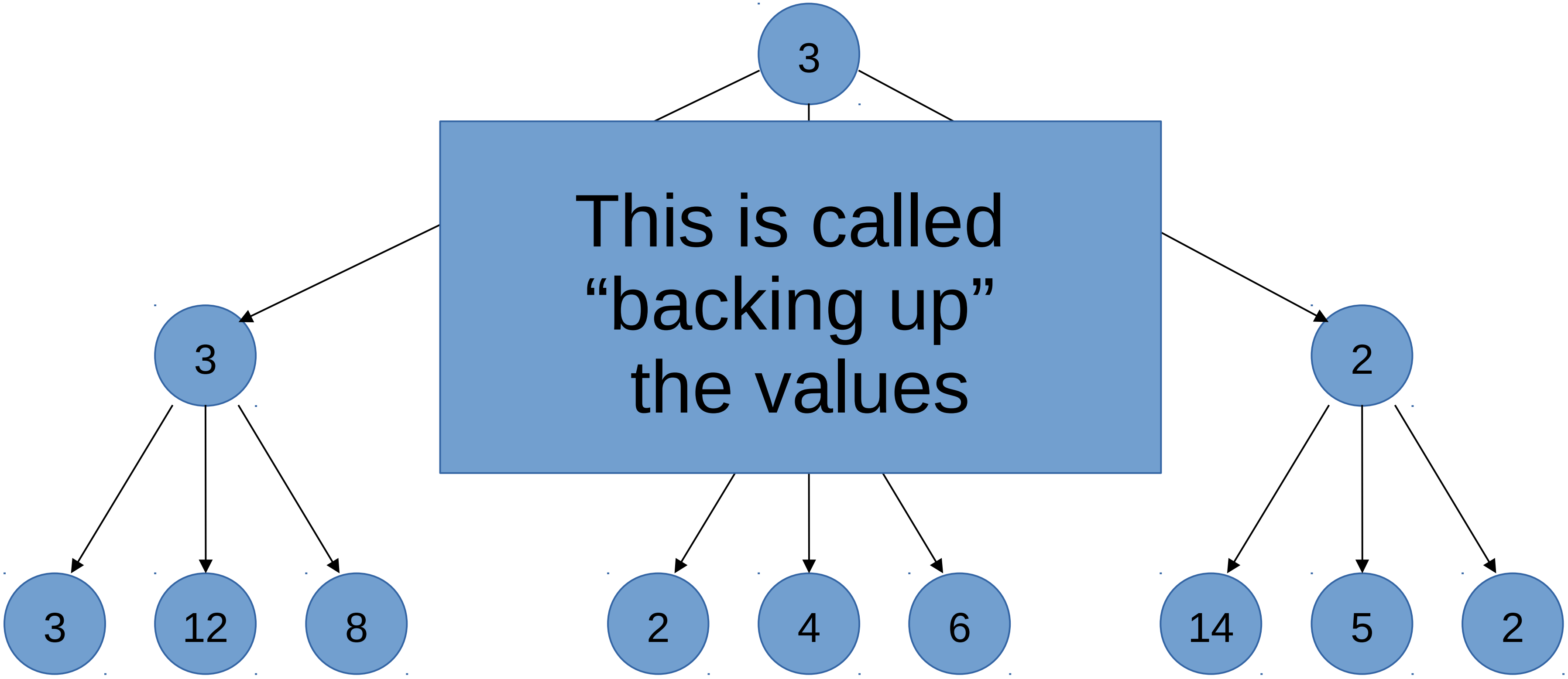
What is Minimax?

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



What is Minimax?

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



Max
(you)

Min
(them)

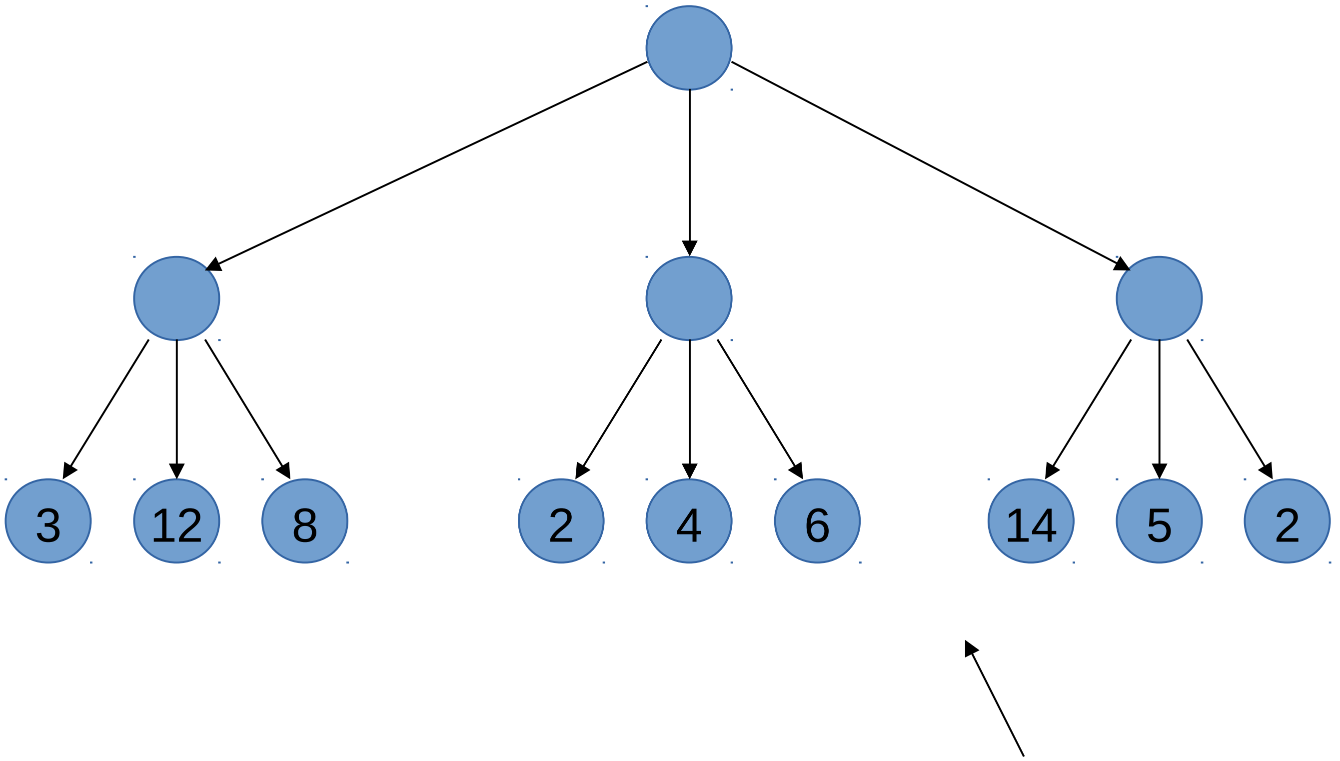
Max
(you)

This is called
“backing up”
the values

Minimax

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

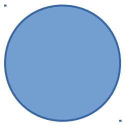
… but, how do we construct the tree?



This tree is already built...

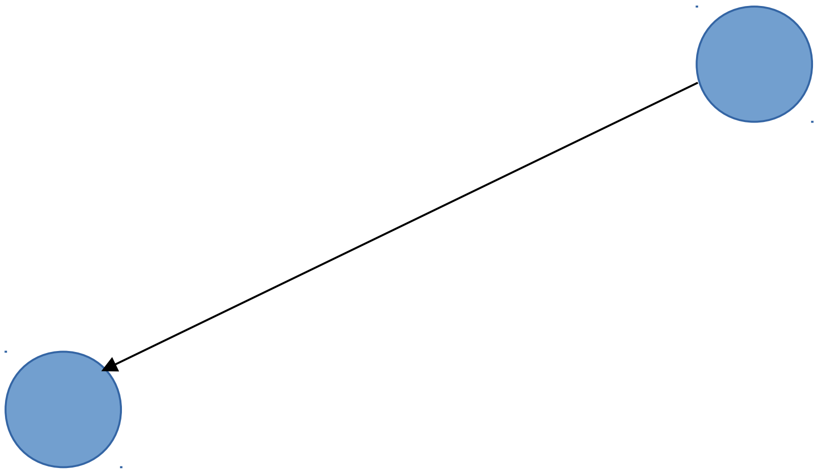
Minimax

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



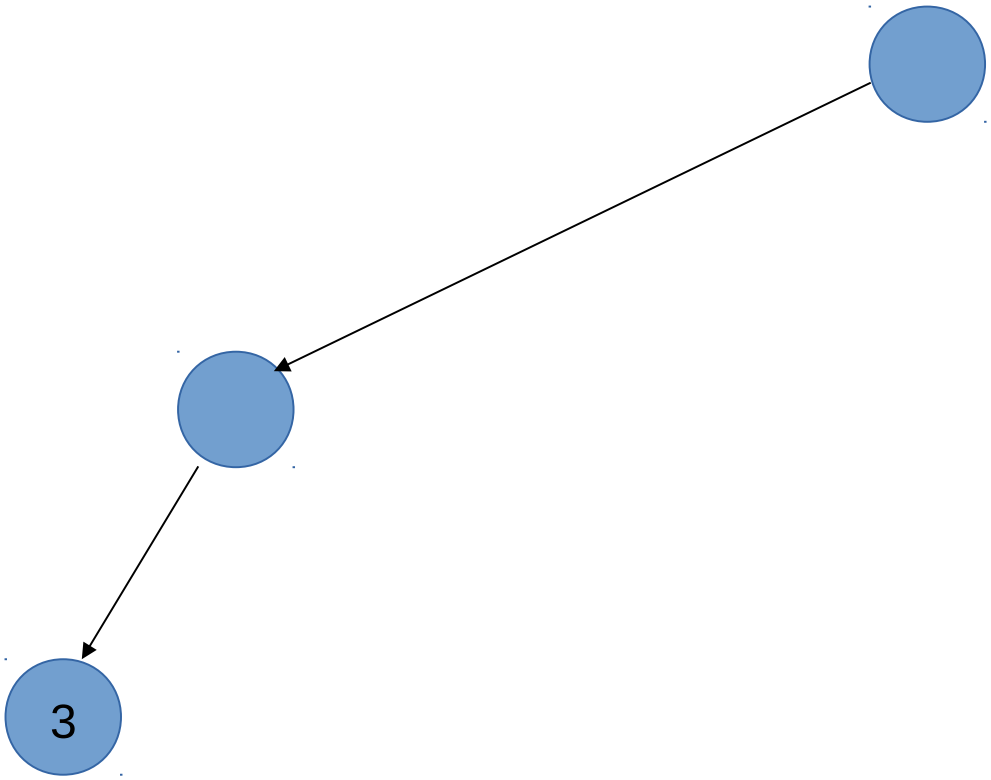
Minimax

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



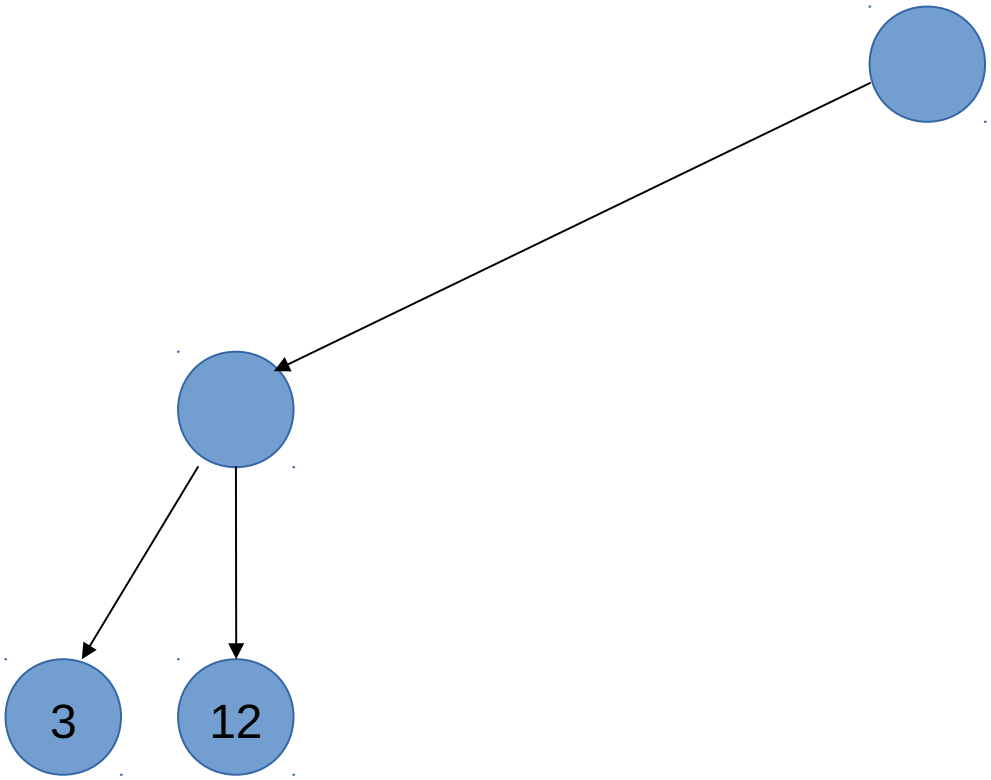
Minimax

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



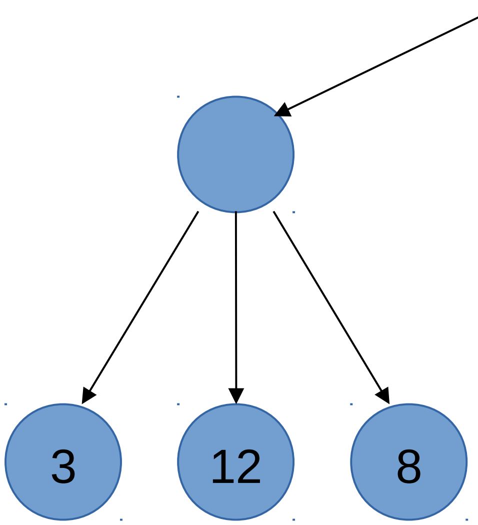
Minimax

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



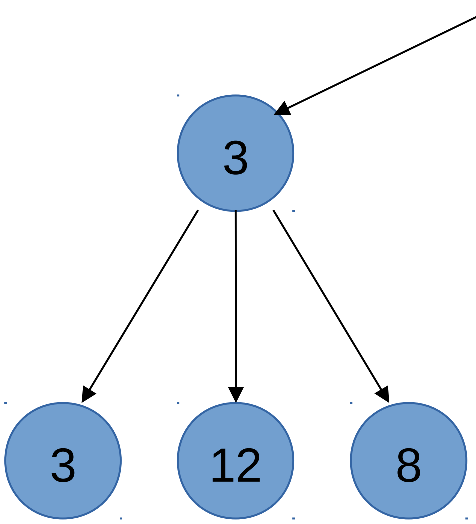
Minimax

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



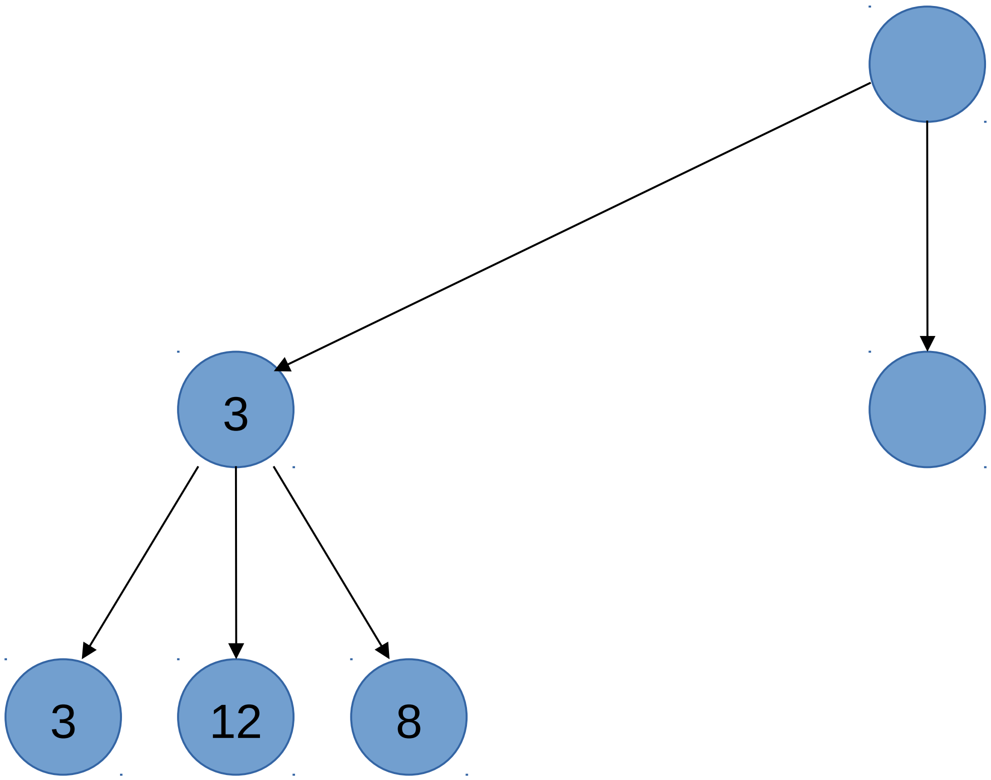
Minimax

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



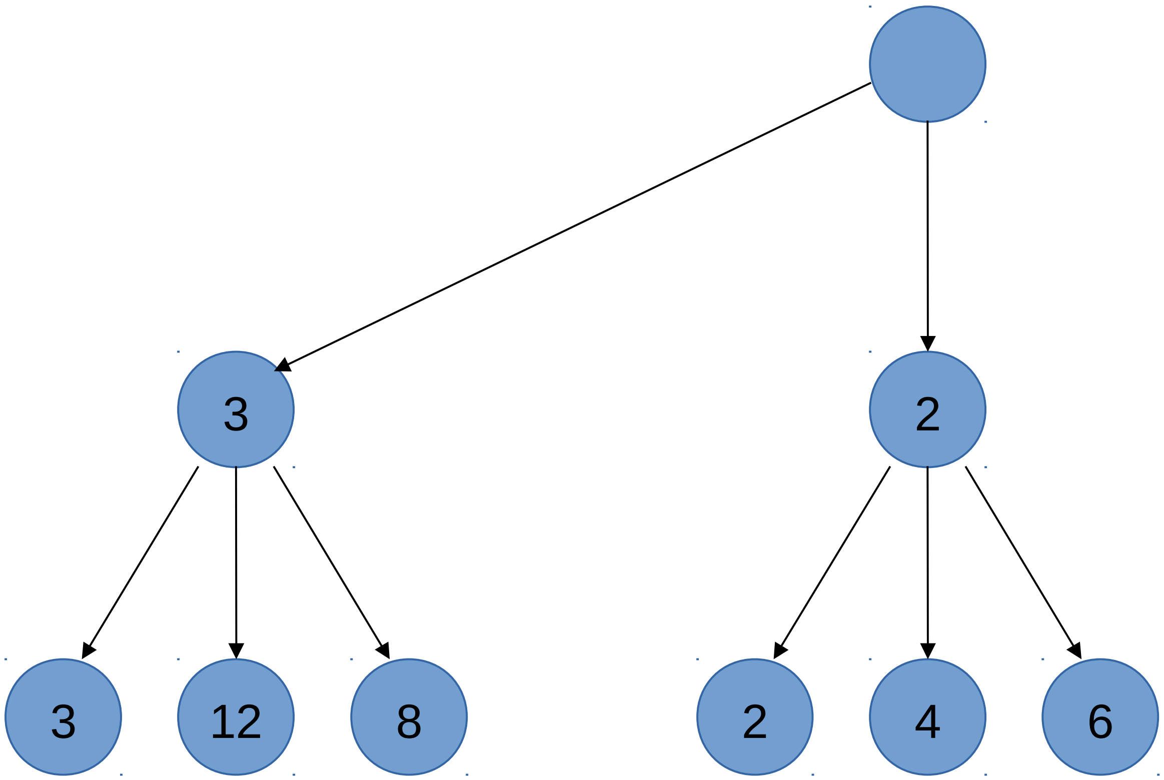
Minimax

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



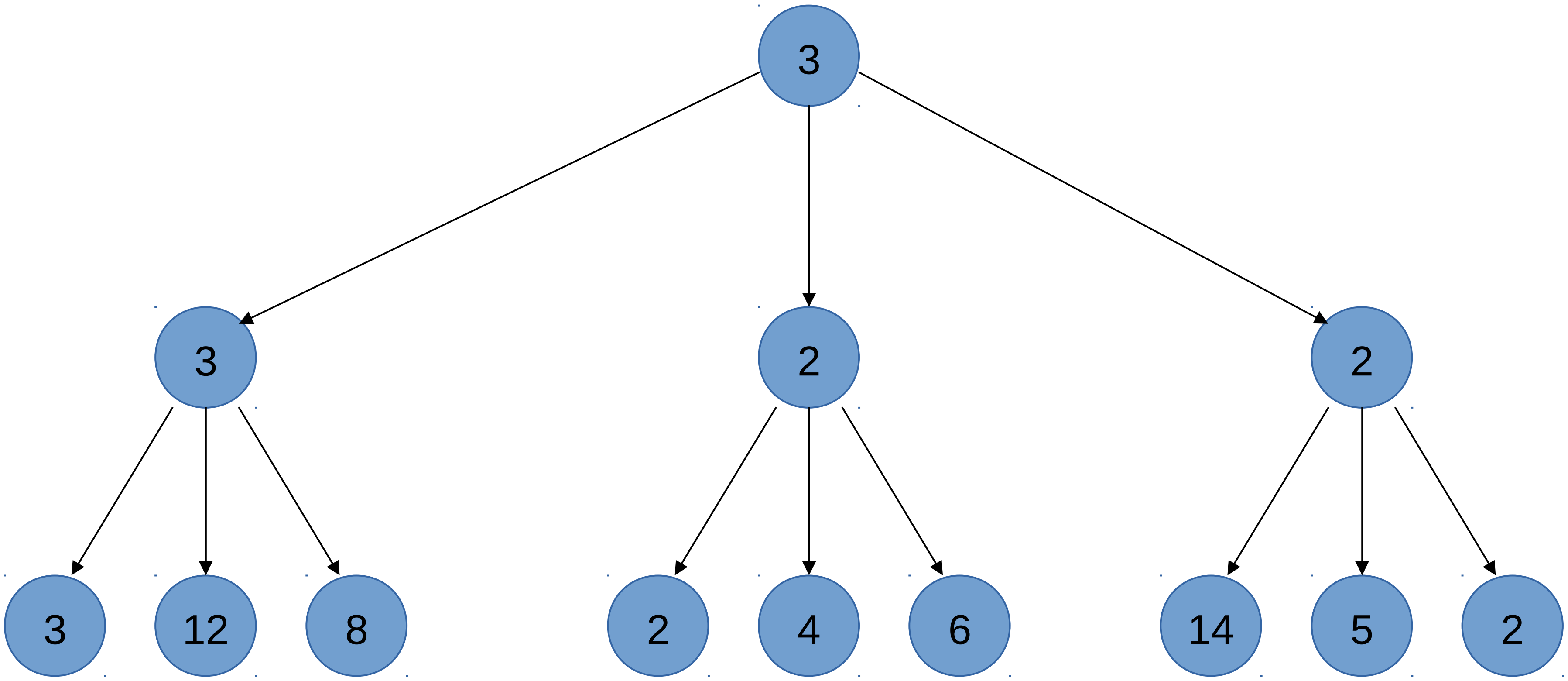
Minimax

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.

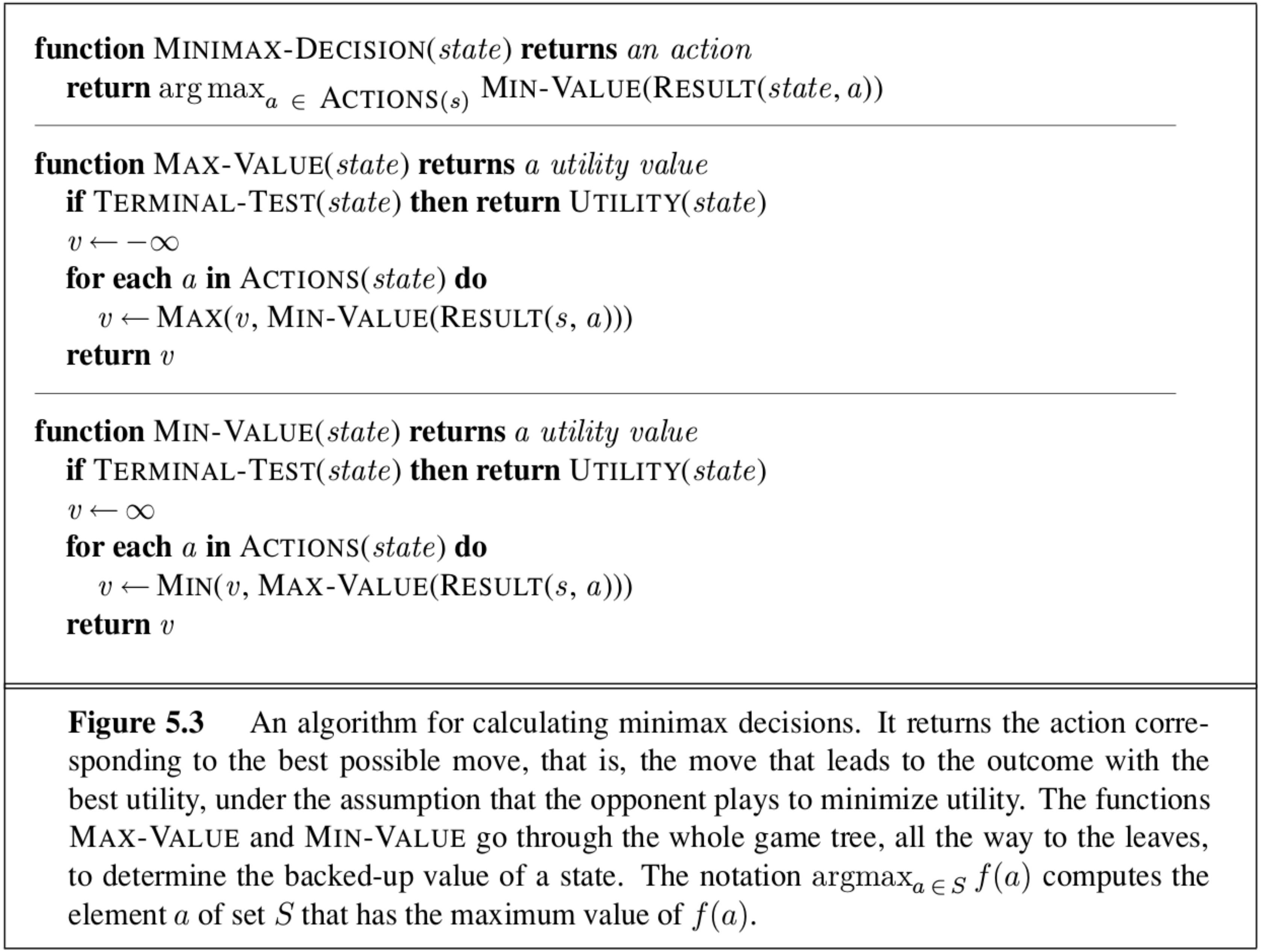


Minimax

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

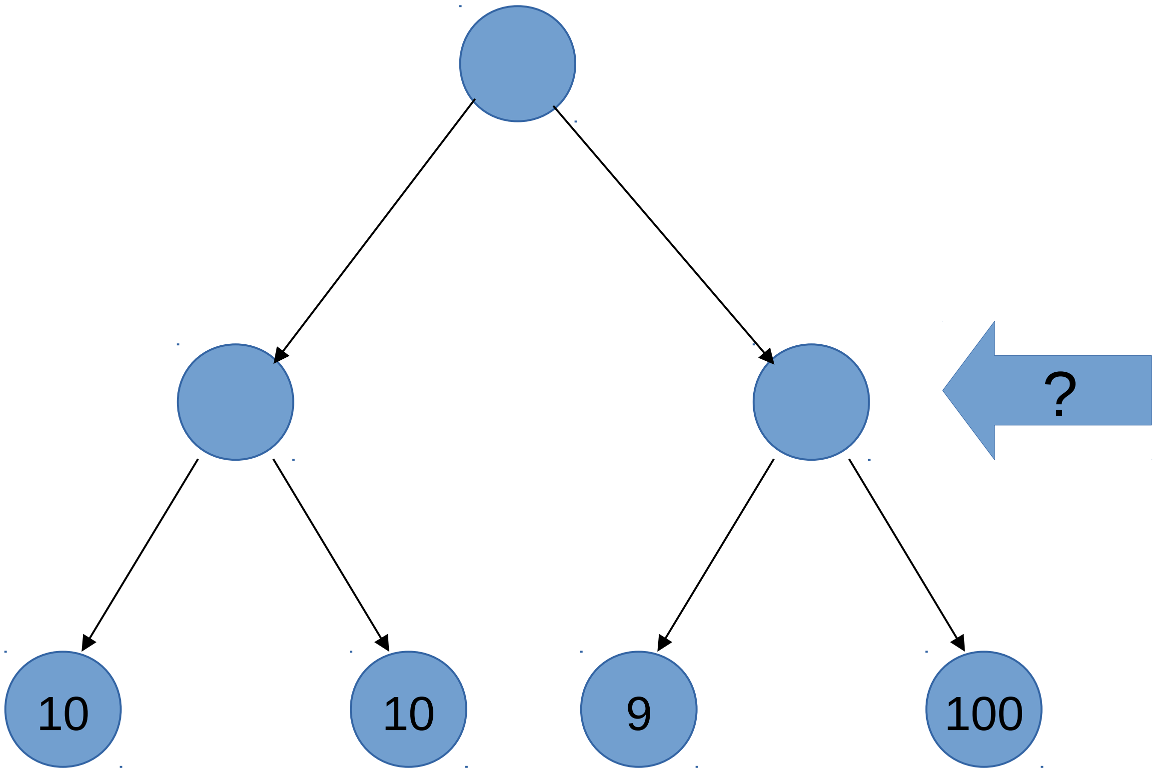


```
function MINIMAX-DECISION(state) returns an action
  return $\arg\max_{a \in \text{ACTIONS}(s)}$ MIN-VALUE(RESULT(state, a))

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

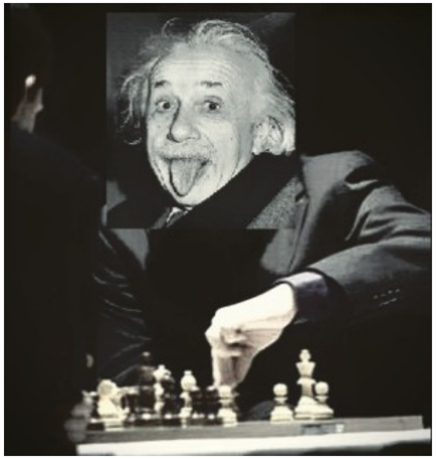
Is it always correct to assume your opponent plays optimally?



Max
(you)

Min
(them)

Max
(you)



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 = $O(bd)$

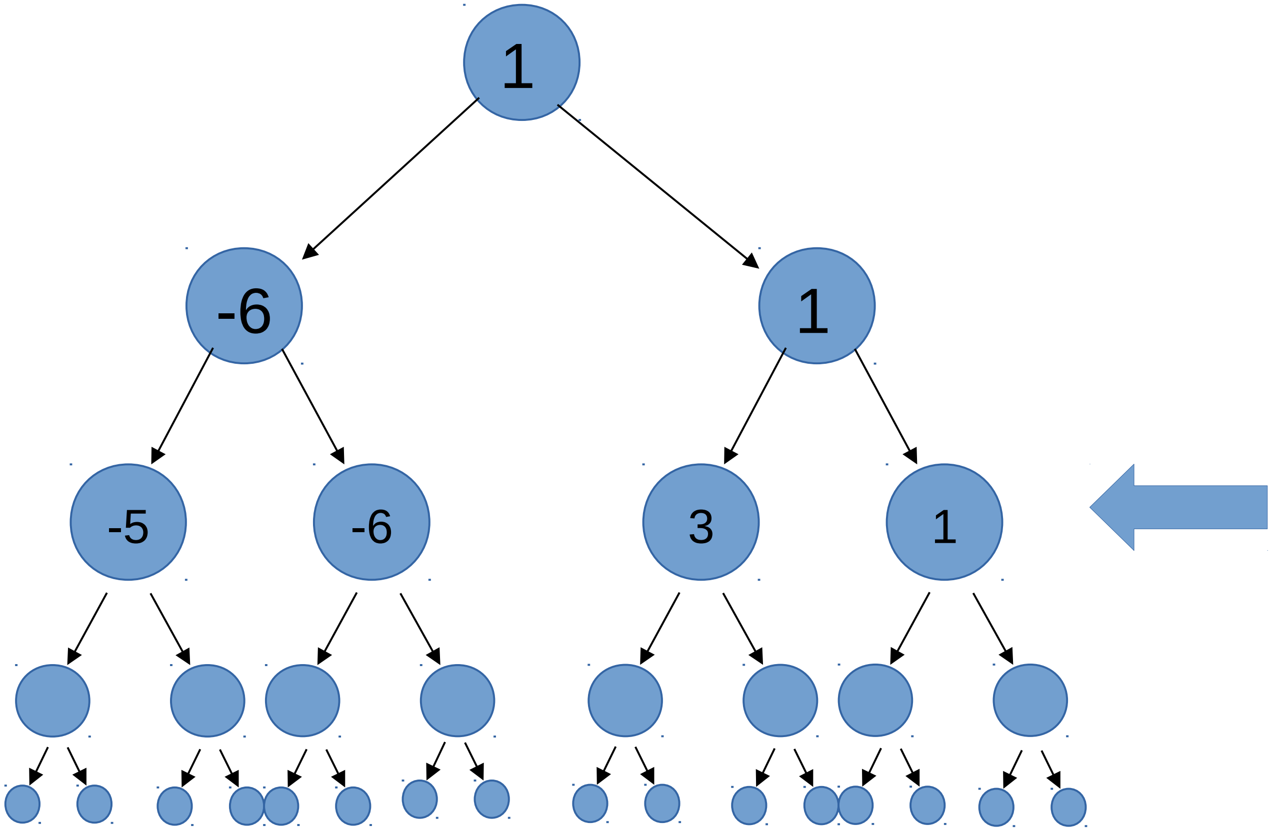
Is it practical? In chess, b=35, d=100

$O(35^{100})$ 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.

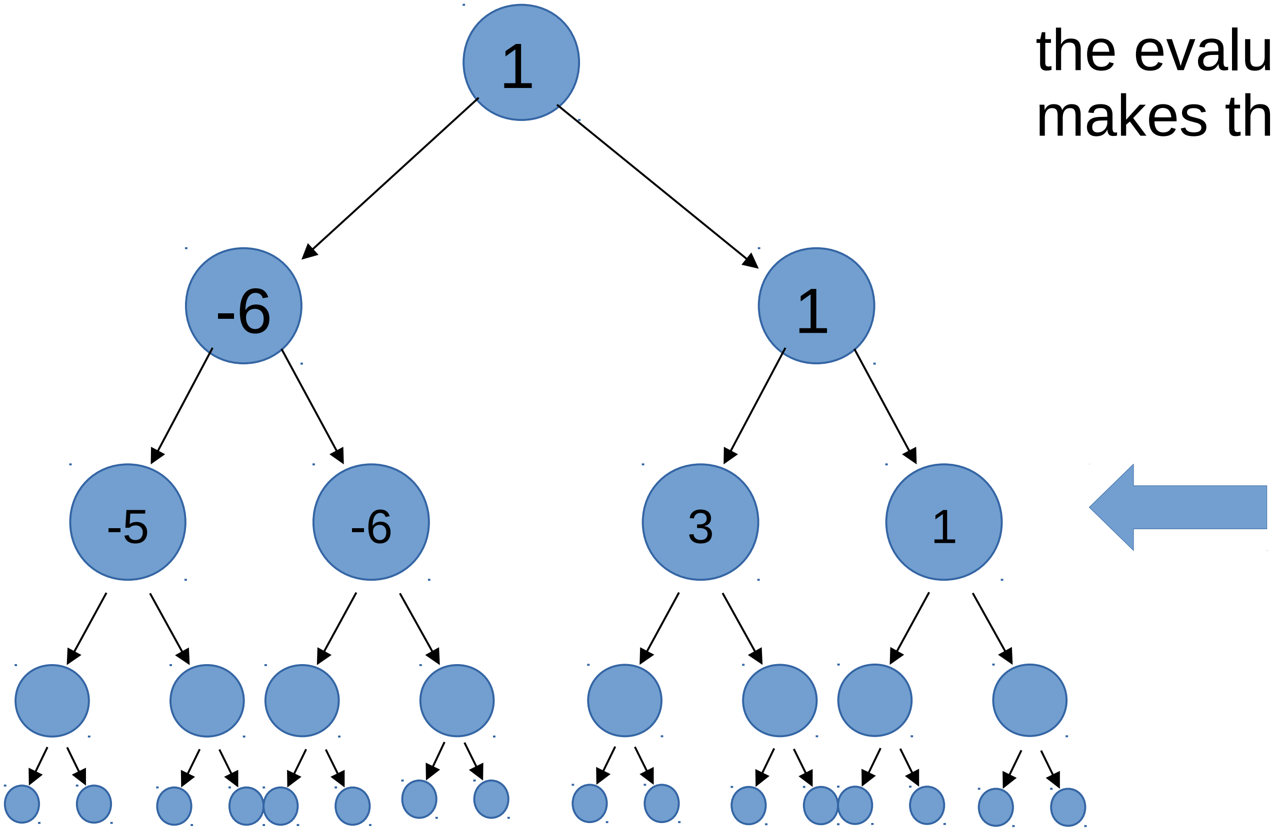


Cut off recursion here

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 off recursion here

Evaluation functions

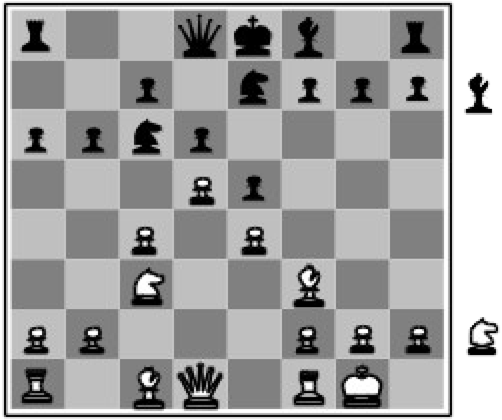
How does the evaluation function make the estimate?

- depends upon domain

For example, in chess, the value of a state might equal the sum of piece values.

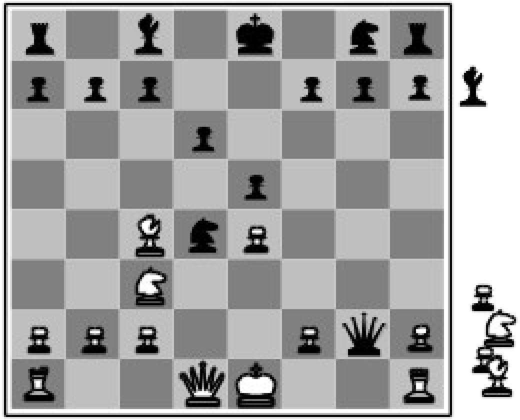
- a pawn counts for 1
- a rook counts for 5
- a knight counts for 3

...



Black to move

White slightly better



White to move

Black winning

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

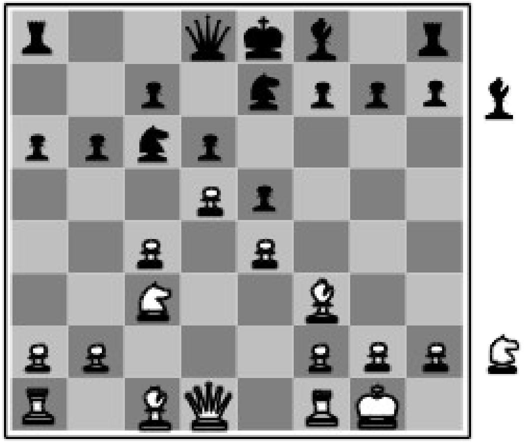
$f_2(s) \equiv$ number of knights on the board

$\vdots$

$w_1 = 1$ ← A pawn counts for 1

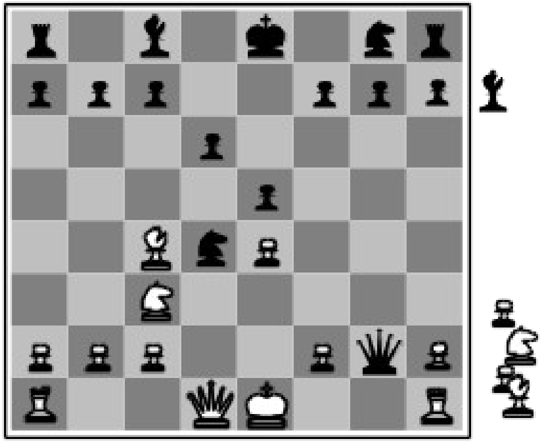
$w_2 = 3$ ← A knight counts for 3

$\vdots$



Black to move

White slightly better



White to move

Black winning

Eval = 3-2.5=0.5

Eval = 3+2.5+1+1-2.5 = 5

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

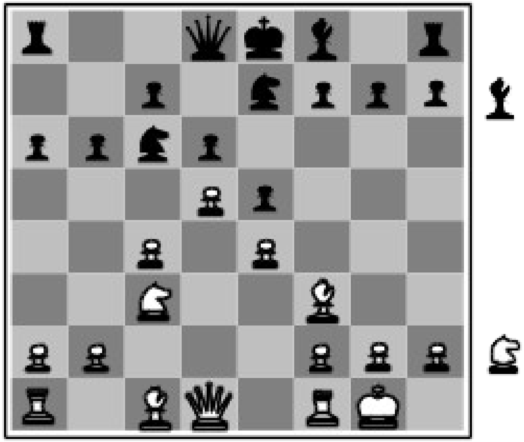
$f_2(s) \equiv$ number of knights on the board

$\vdots$

$w_1 = 1$ ← A pawn counts for 1

$w_2 = 3$ ← A knight counts for 3

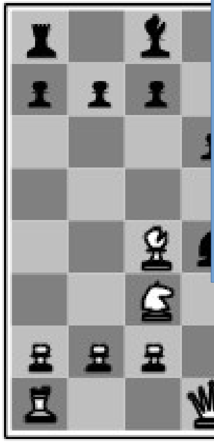
$\vdots$



Black to move

White slightly better

Eval = 3-2.5=0.5



Maybe consider other factors as well?

White to move

Black winning

Eval = 3+2.5+1+1-2.5 = 5

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:

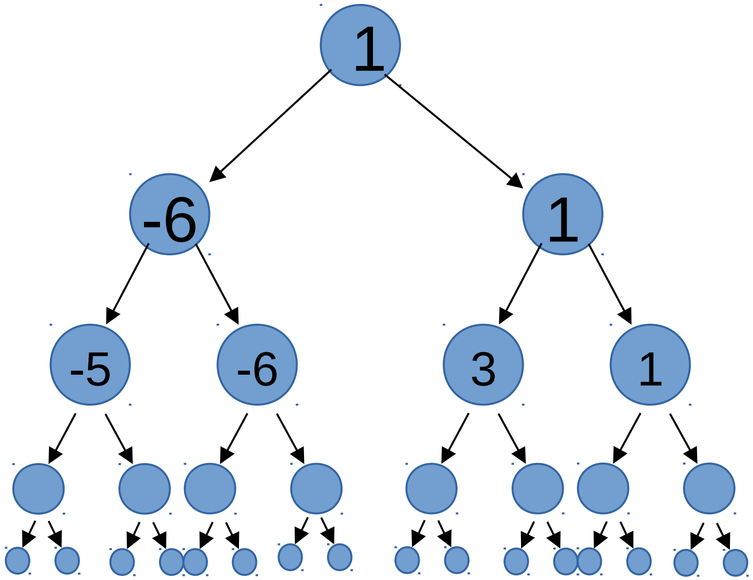
- Suppose we have 100 seconds
- Can explore 10K nodes / sec
- So can check 1M nodes per move

Guarantee of optimal play is gone

More plies makes a BIG difference

Use iterative deepening for an anytime algorithm

At what depth do you run the evaluation function?



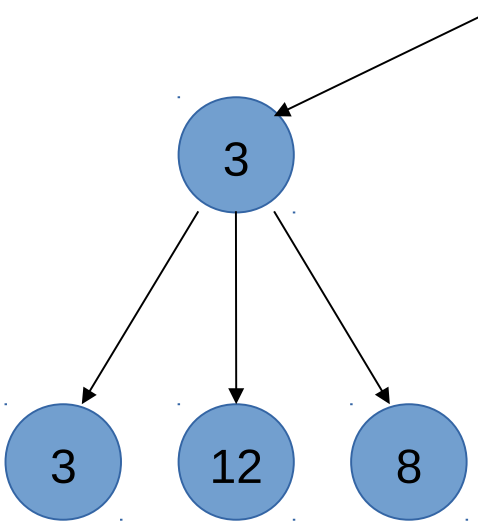
Option 1: cut off search at a fixed depth

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

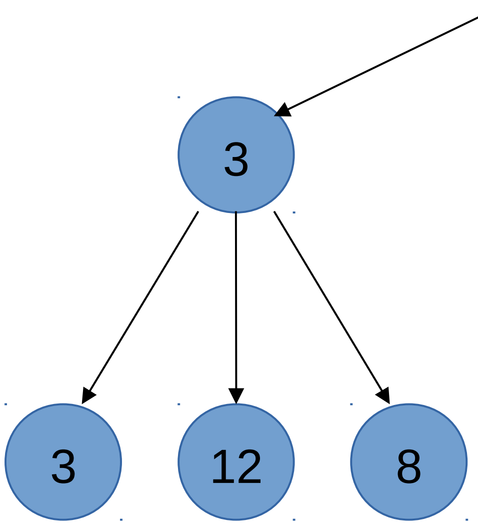
The deeper your threshold, the less the quality of the evaluation function matters...

Alpha/Beta pruning

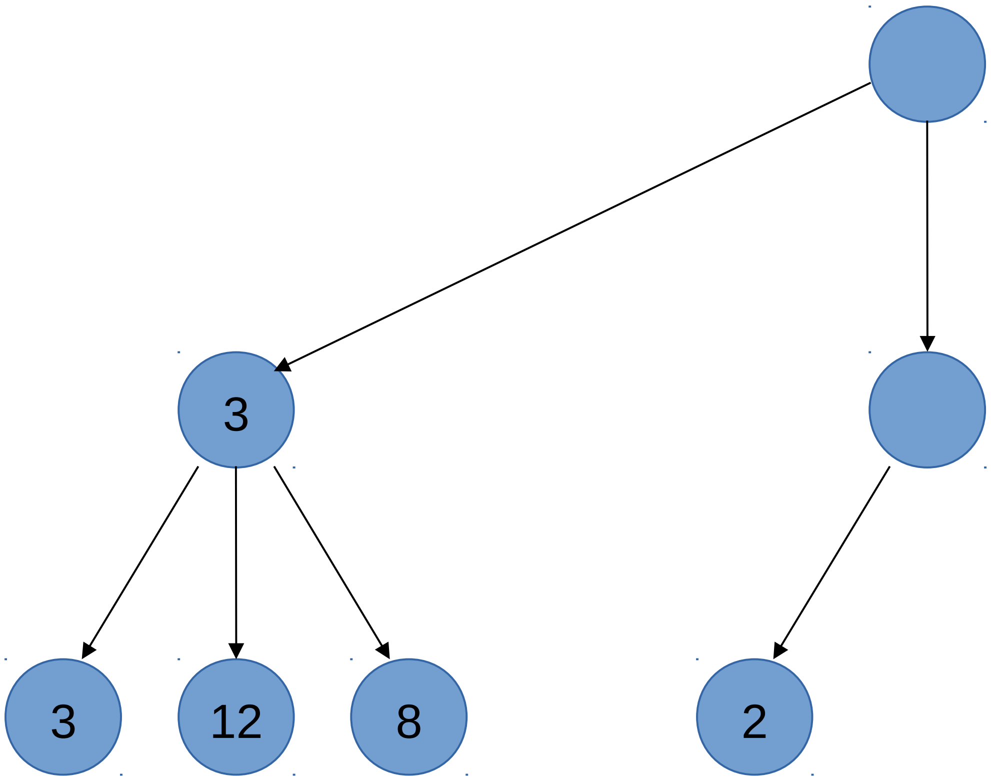
Alpha/Beta pruning



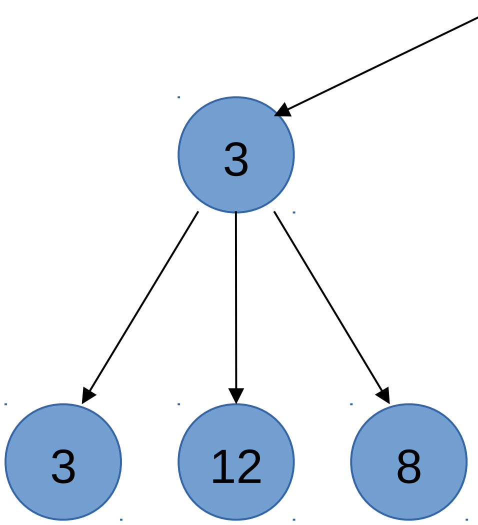
Alpha/Beta pruning



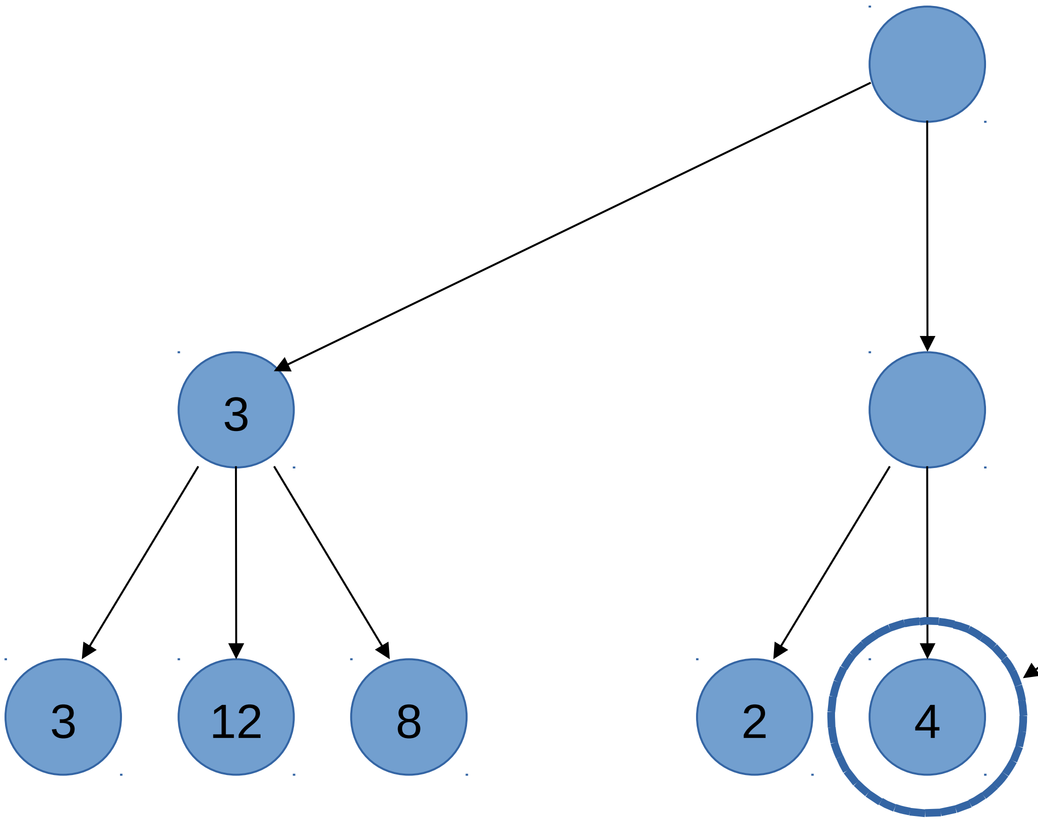
Alpha/Beta pruning



Alpha/Beta pruning

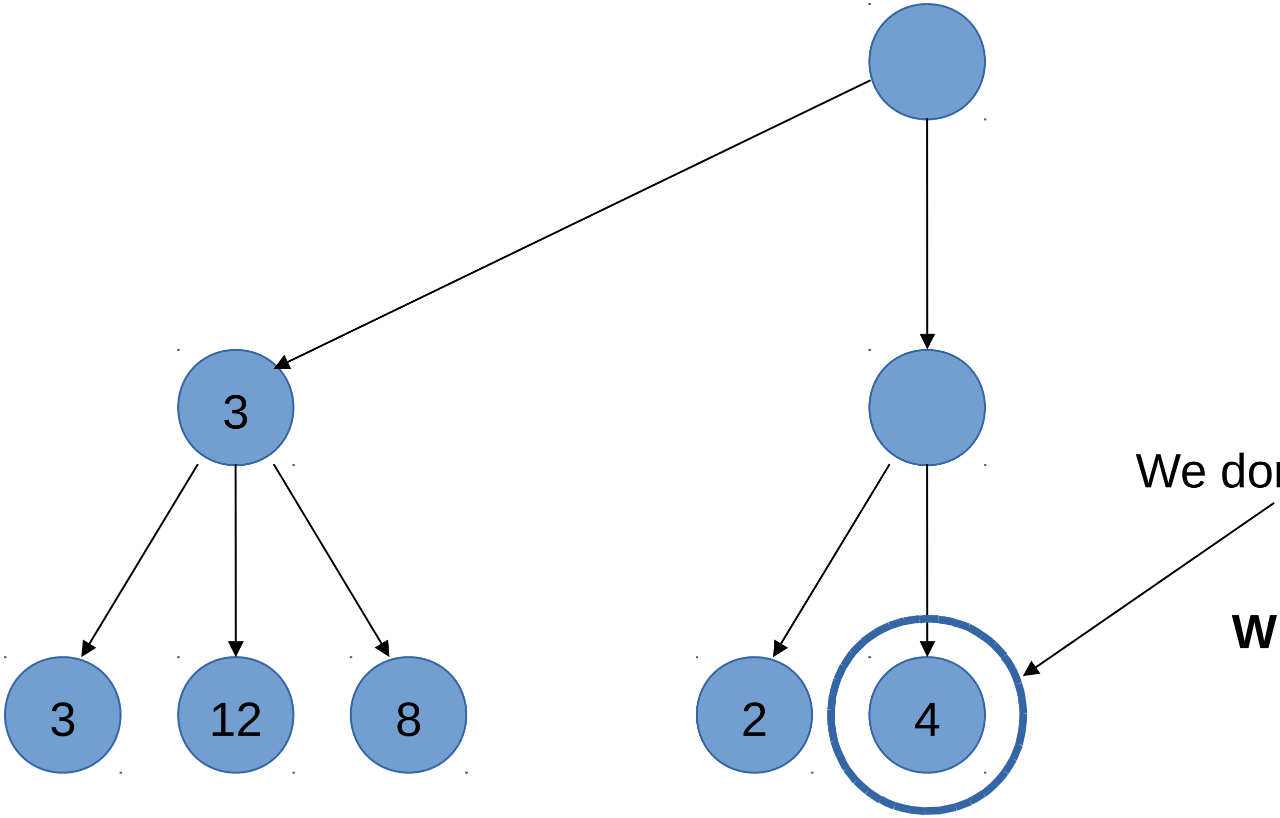


Alpha/Beta pruning



We don't need to expand this node!

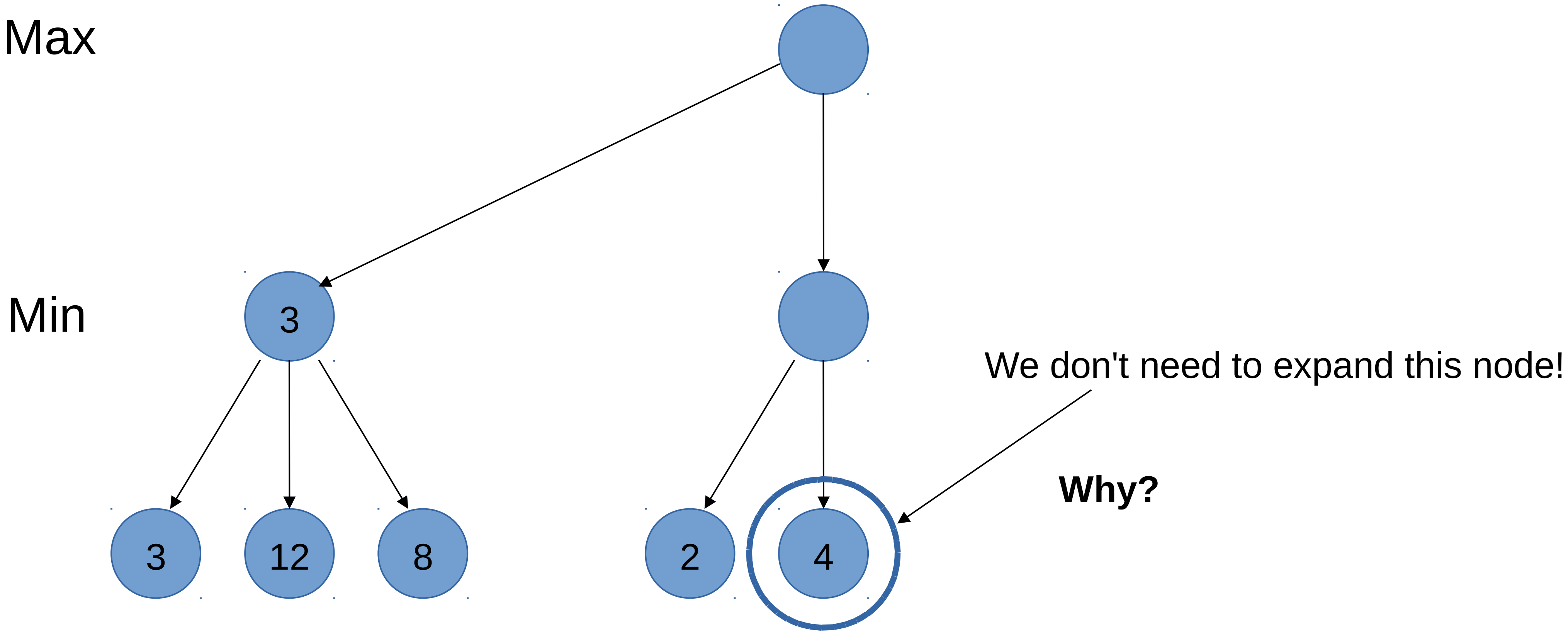
Alpha/Beta pruning



We don't need to expand this node!

Why?

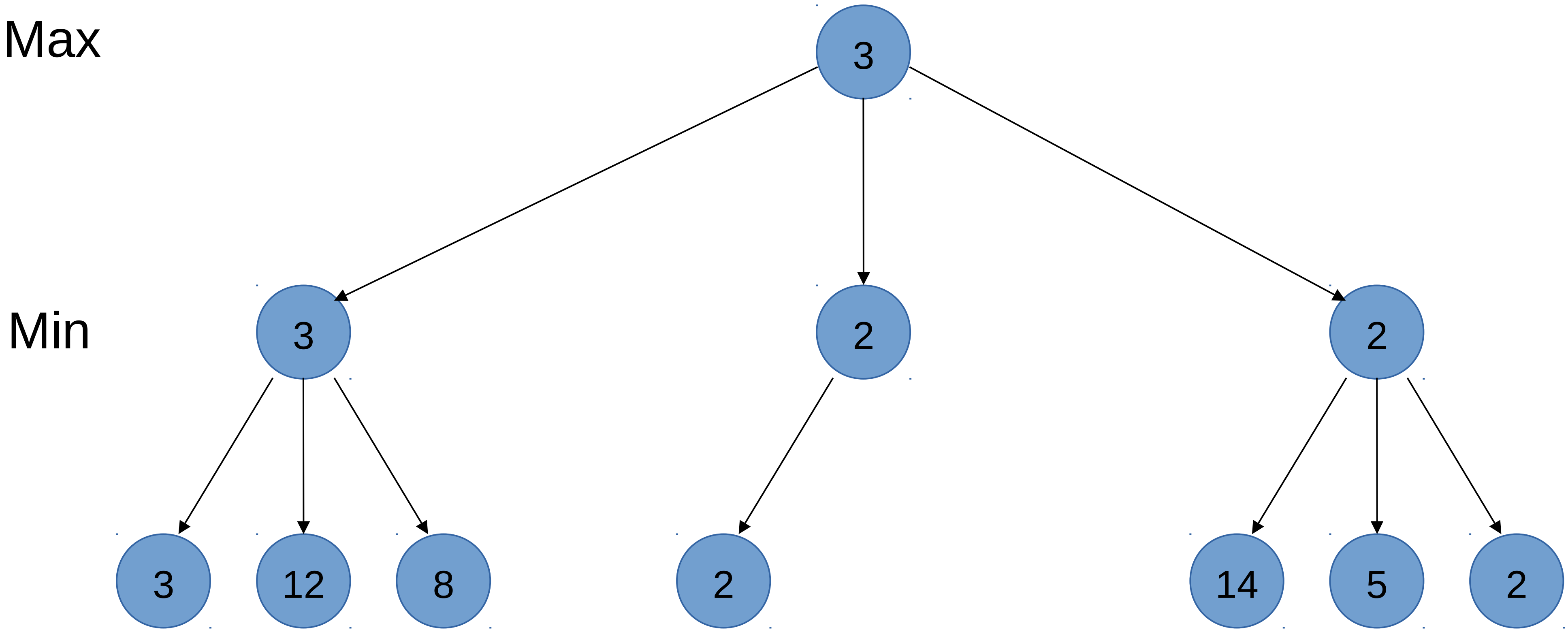
Alpha/Beta pruning



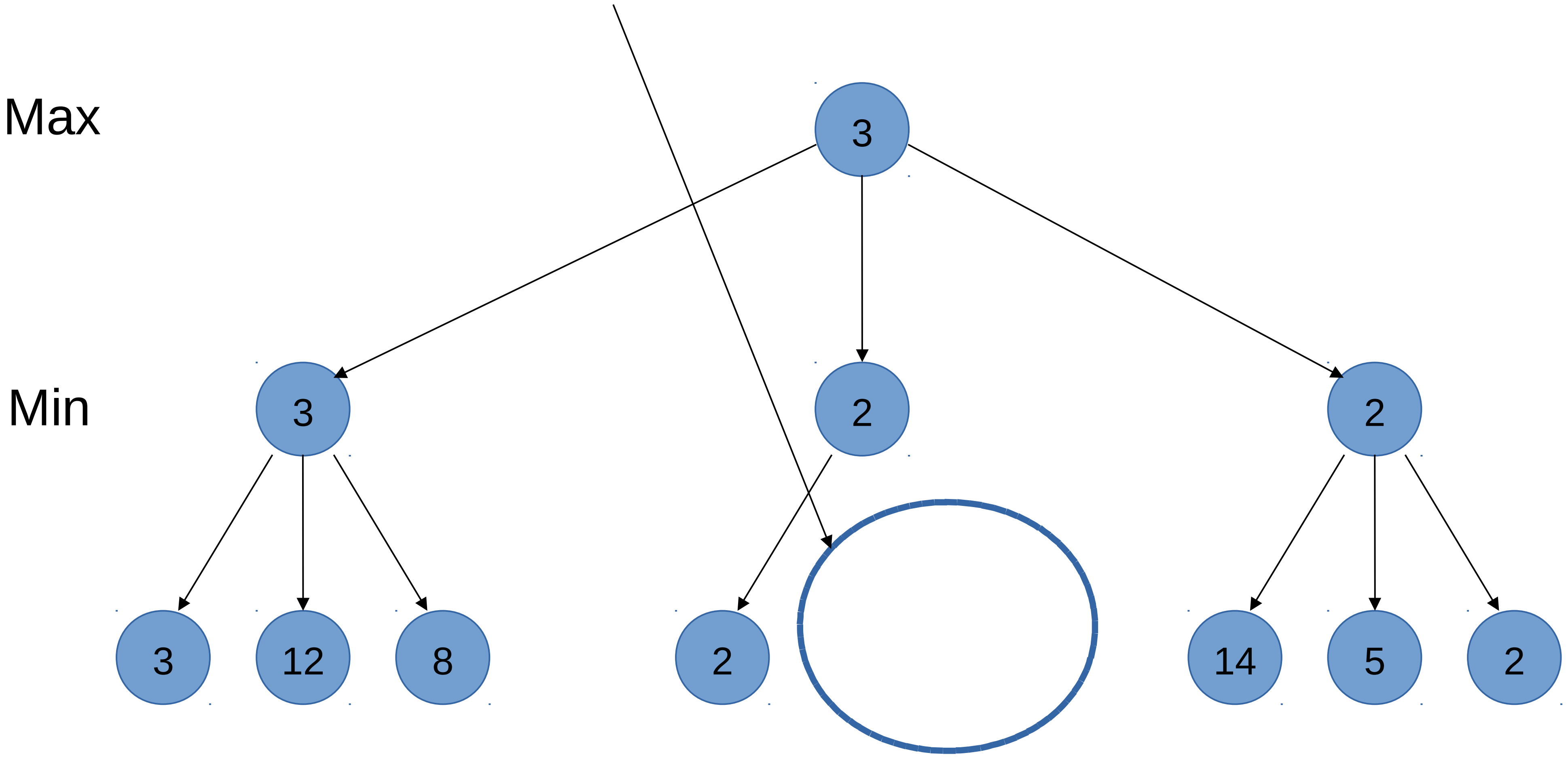
We don't need to expand this node!

Why?

Alpha/Beta pruning



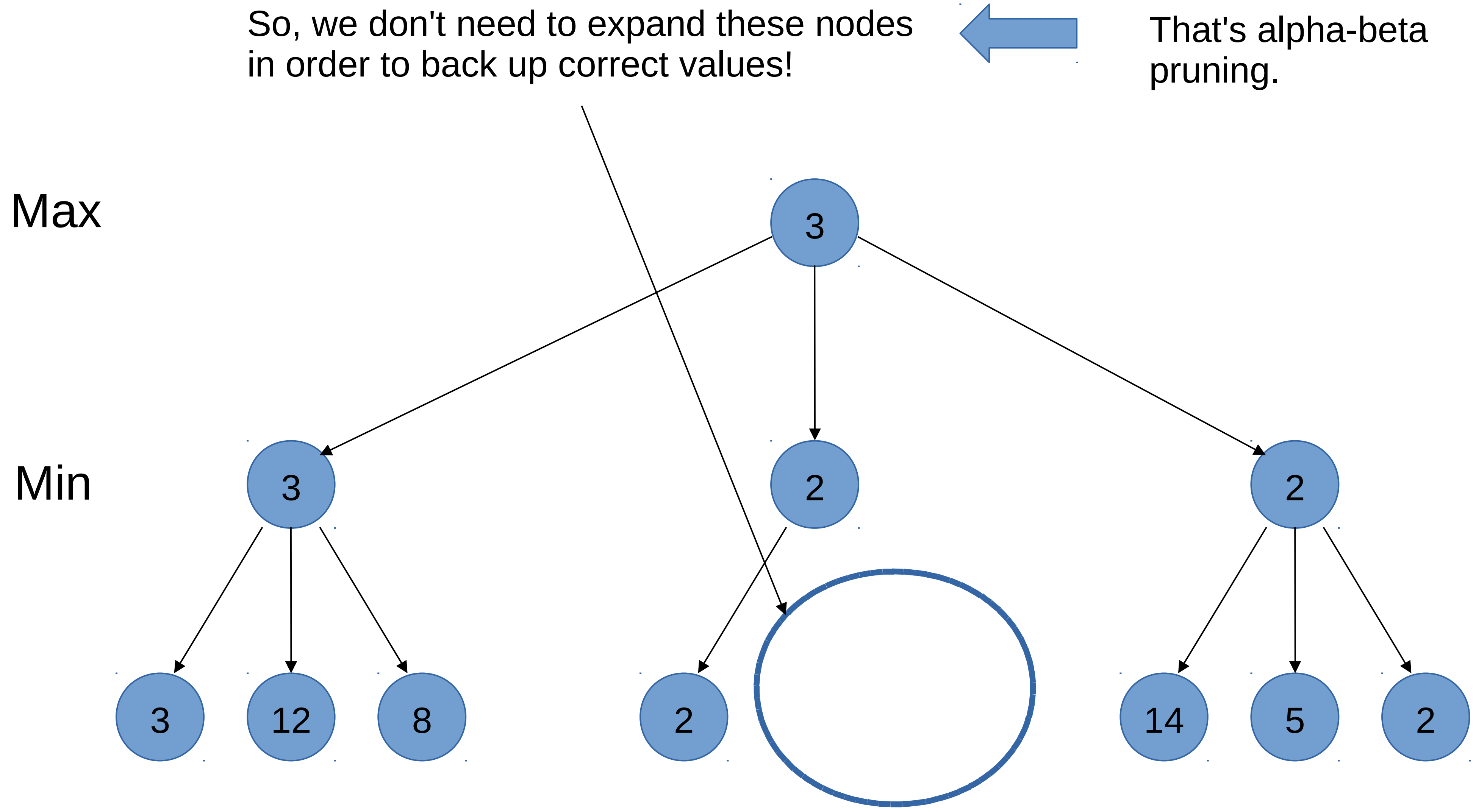
Alpha/Beta pruning


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

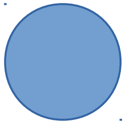
α: MAX’s best option on path to root

β: MIN’s best option on path to root

```
def max-value(state, $\alpha$, $\beta$):
    initialize v = $-\infty$
    for each successor of state:
        v = max(v,
          value(successor, $\alpha$, $\beta$))
        if $v \geq \beta$ return v
        $\alpha$ = max($\alpha$, v)
    return v
```

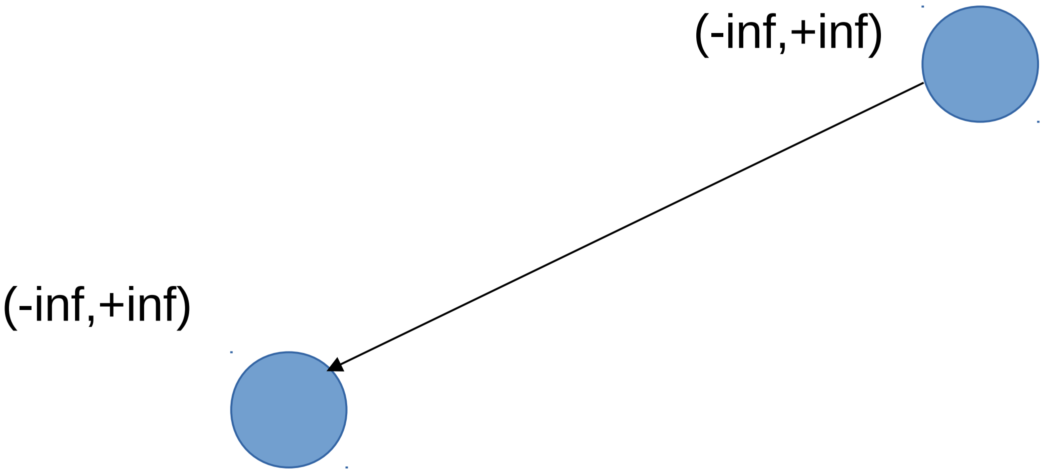
```
def min-value(state , $\alpha$, $\beta$):
    initialize v = $+\infty$
    for each successor of state:
        v = min(v, value(successor,
          $\alpha$, $\beta$))
        if $v \leq \alpha$ return v
        $\beta$ = min($\beta$, v)
    return v
```

Alpha/Beta pruning

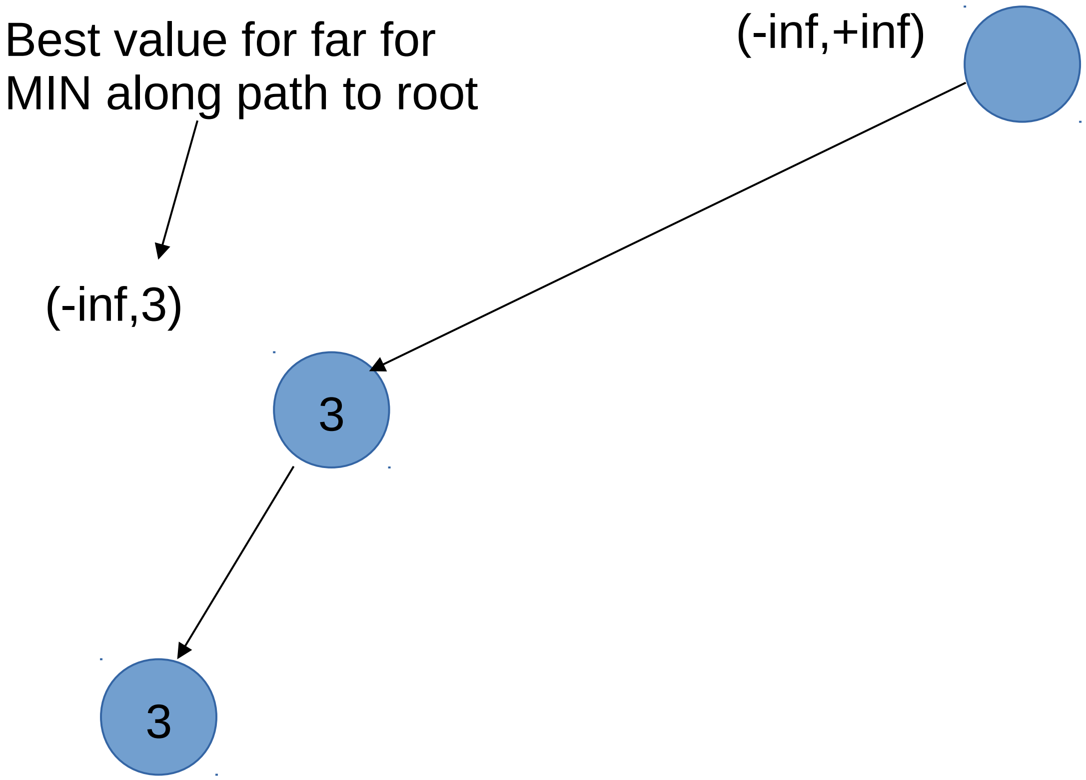


(-inf,+inf)

Alpha/Beta pruning



Alpha/Beta pruning

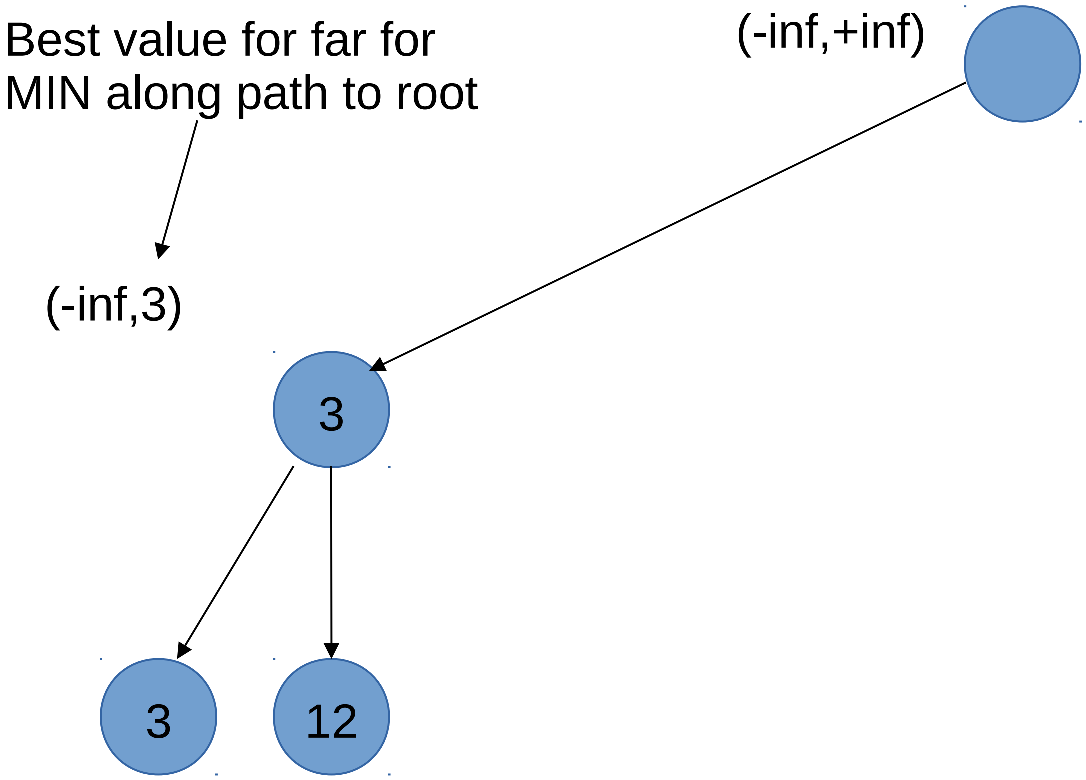


Best value for far for MIN along path to root

(-inf,+inf)

(-inf,3)

Alpha/Beta pruning

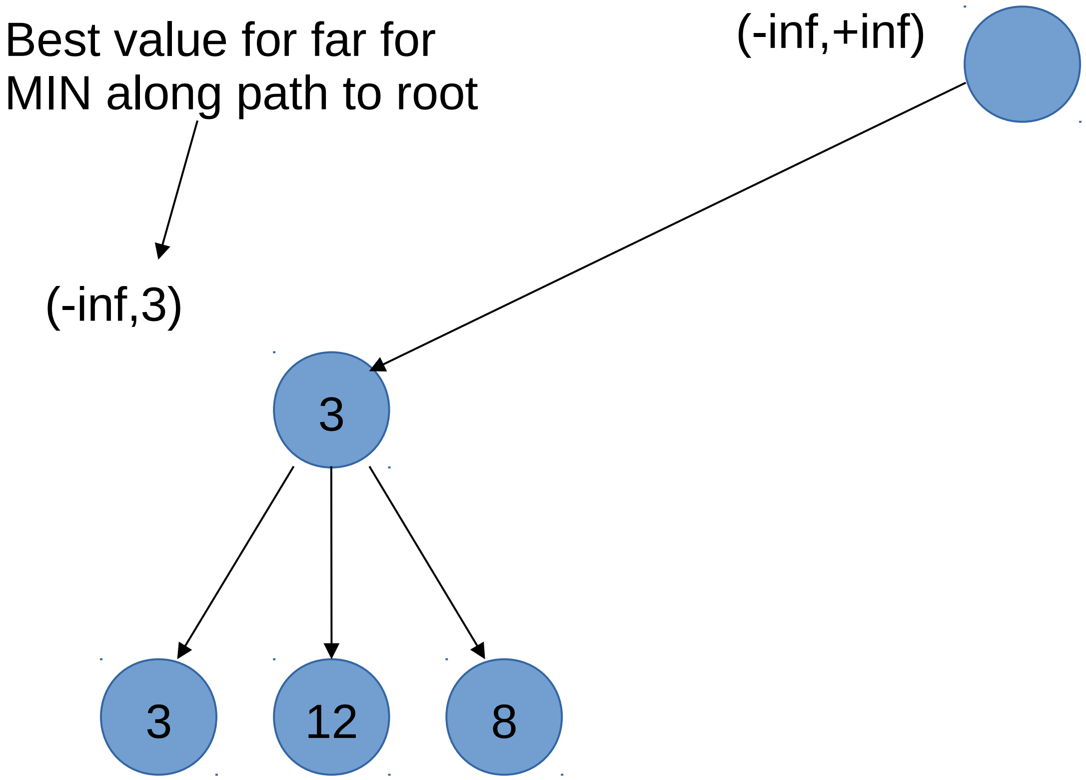


Best value for far for MIN along path to root

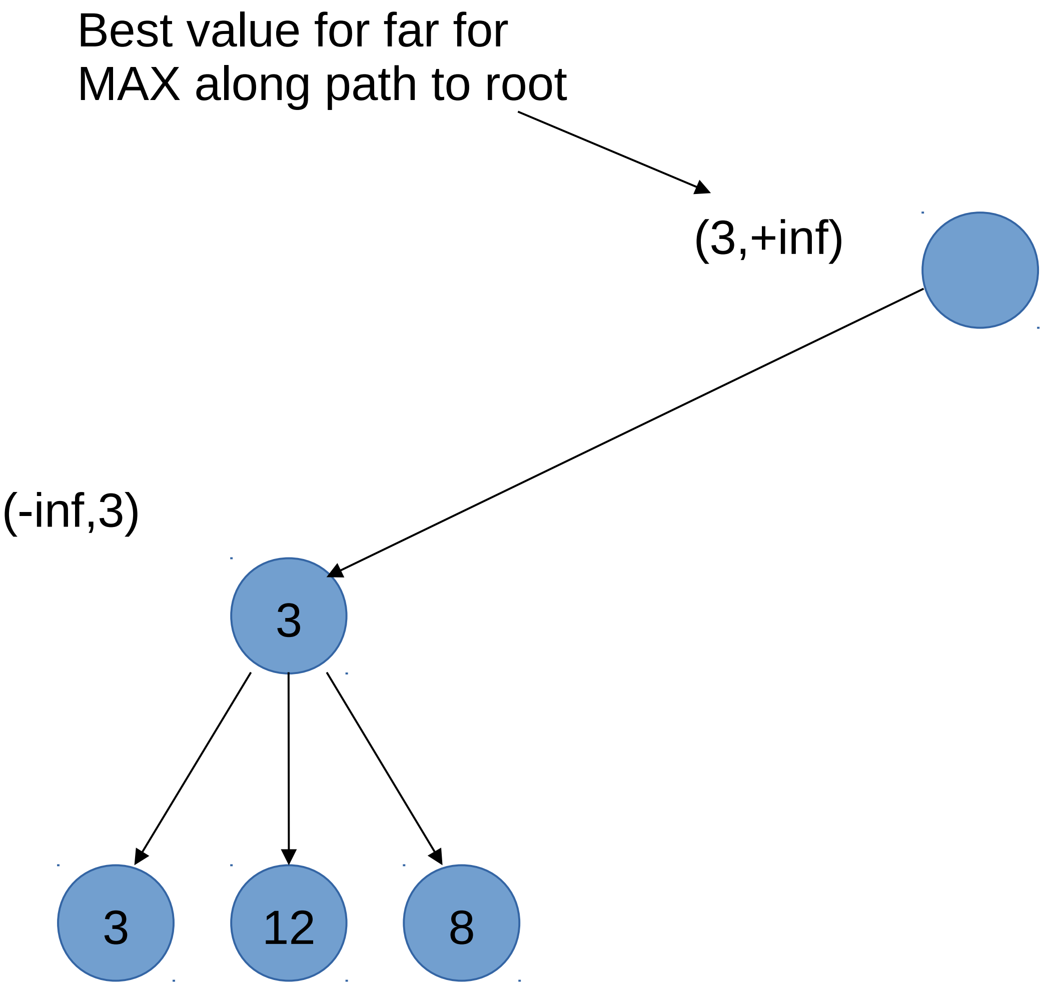
(-inf,+inf)

(-inf,3)

Alpha/Beta pruning


Best value for far for MIN along path to root

Alpha/Beta pruning

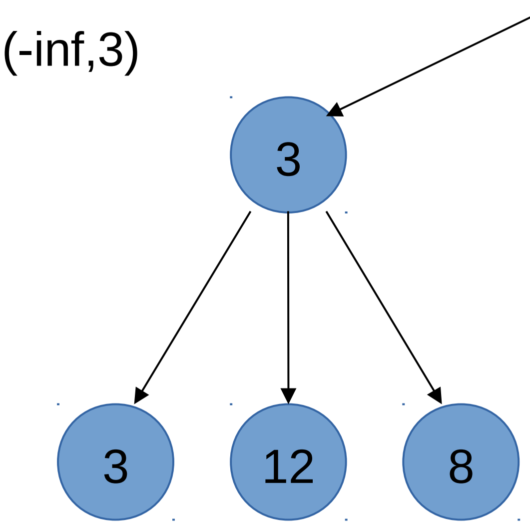


Best value for far for
MAX along path to root

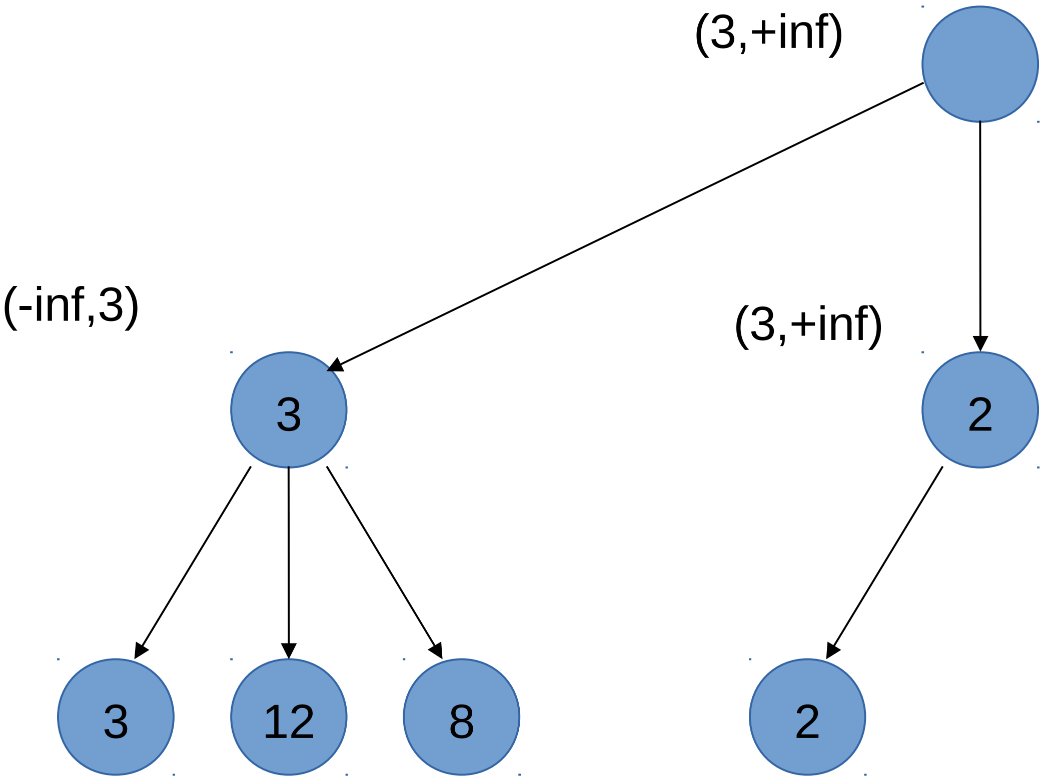
(3,+inf)

(-inf,3)

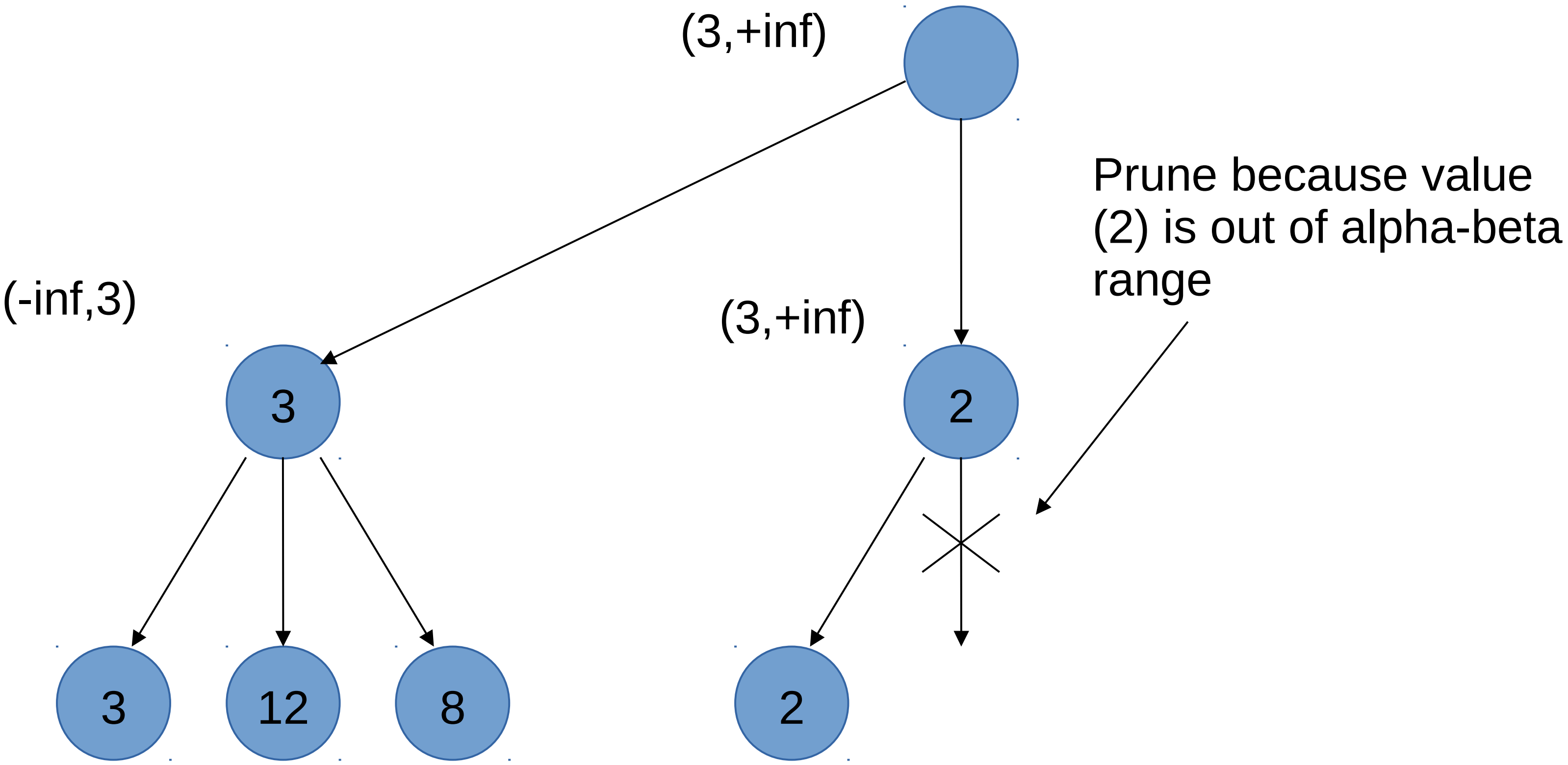
Alpha/Beta pruning



Alpha/Beta pruning

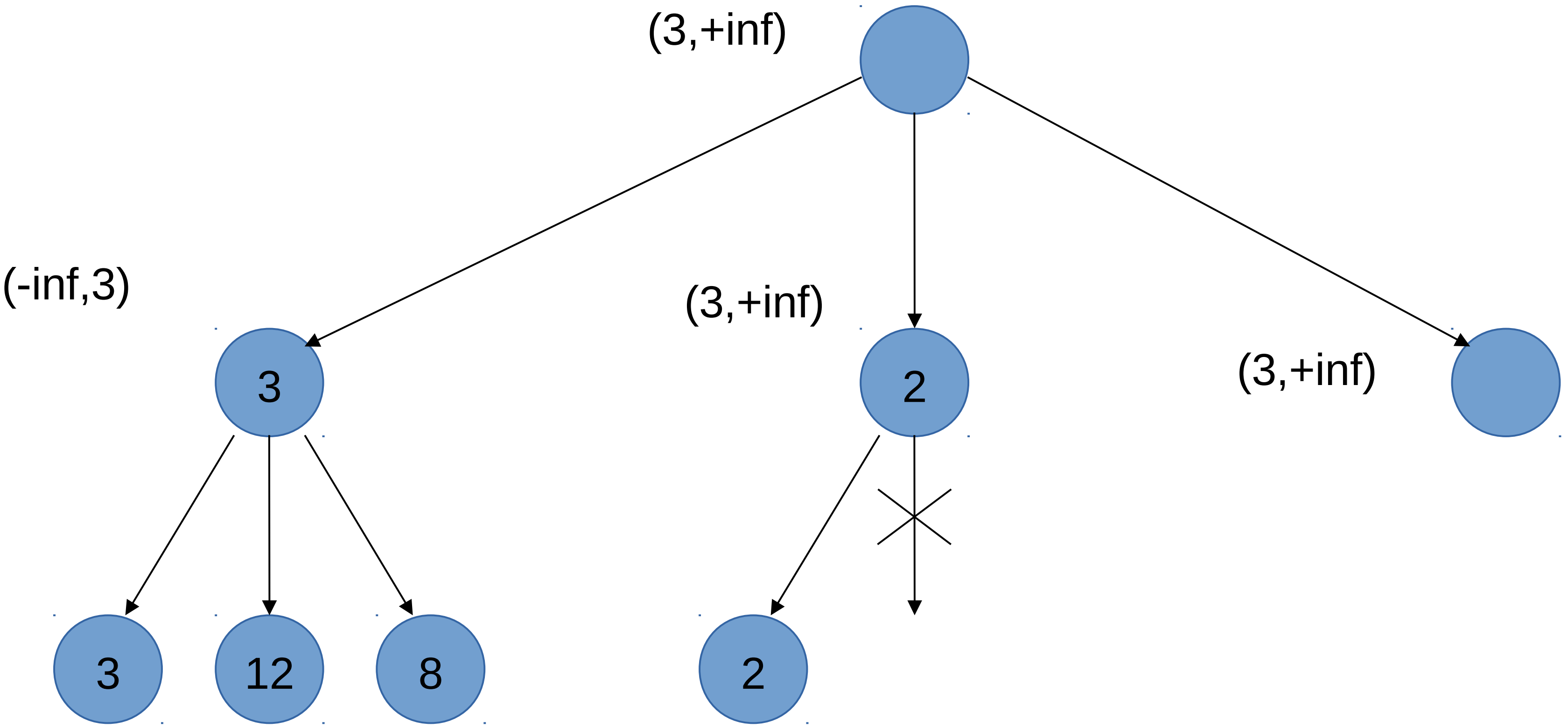


Alpha/Beta pruning

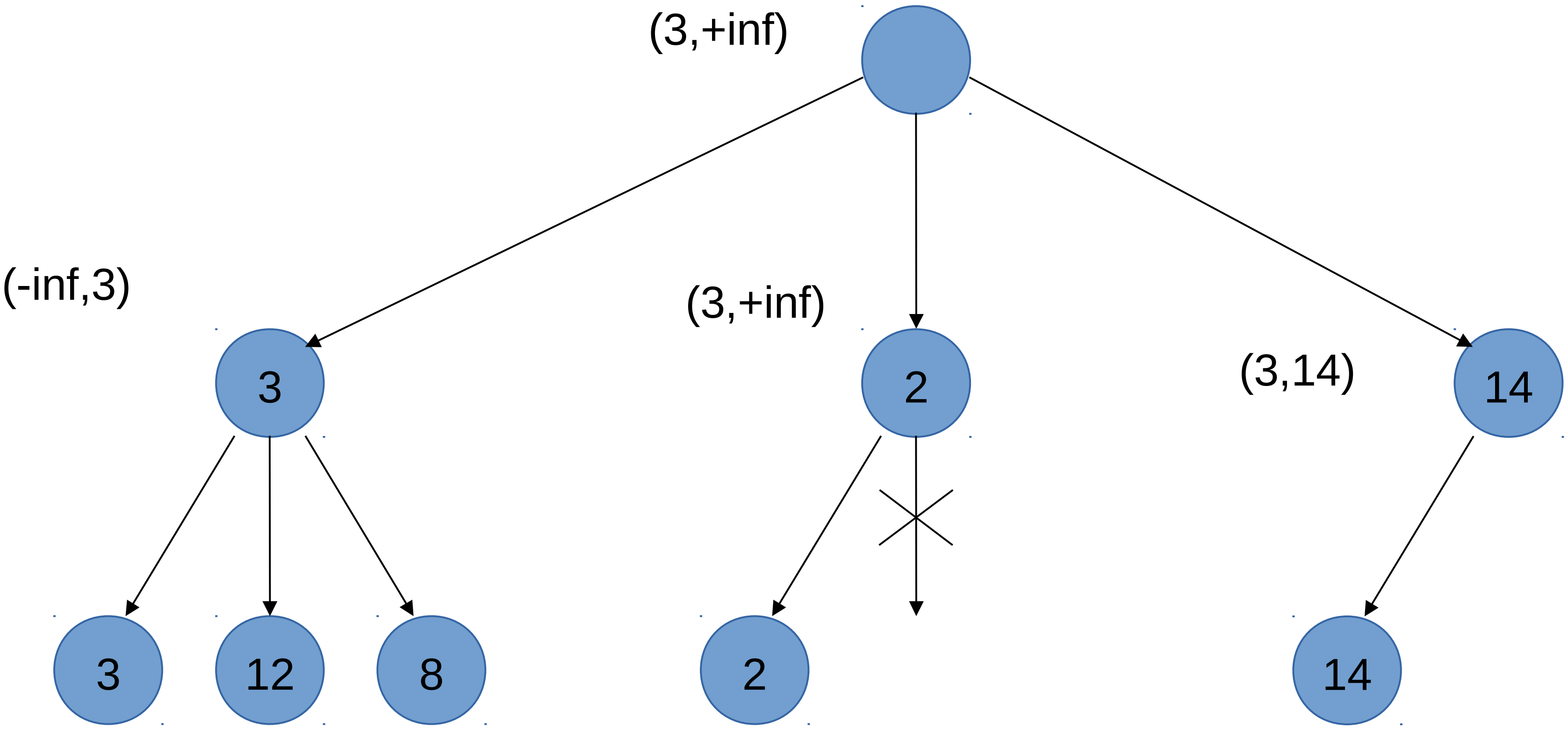


Prune because value (2) is out of alpha-beta range

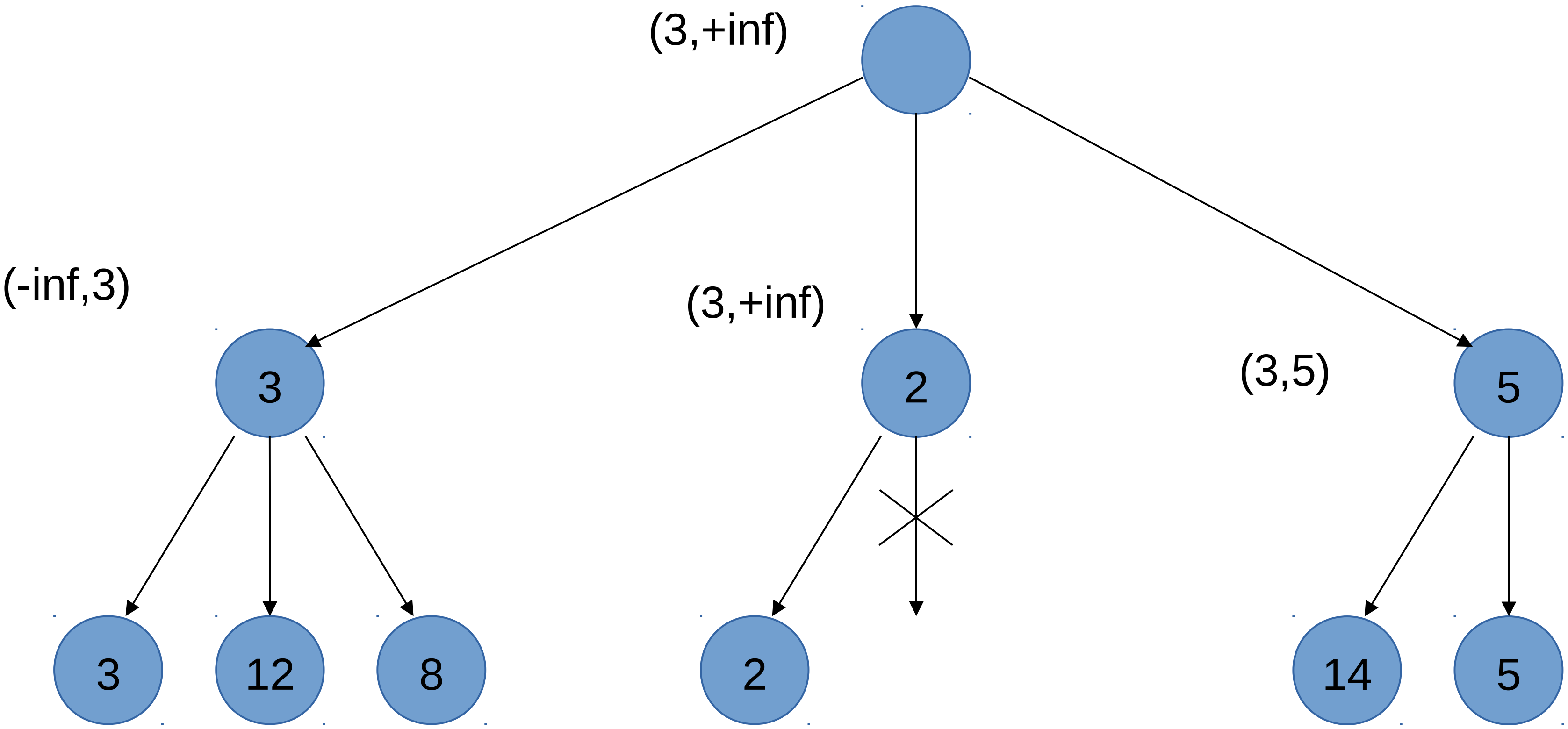
Alpha/Beta pruning



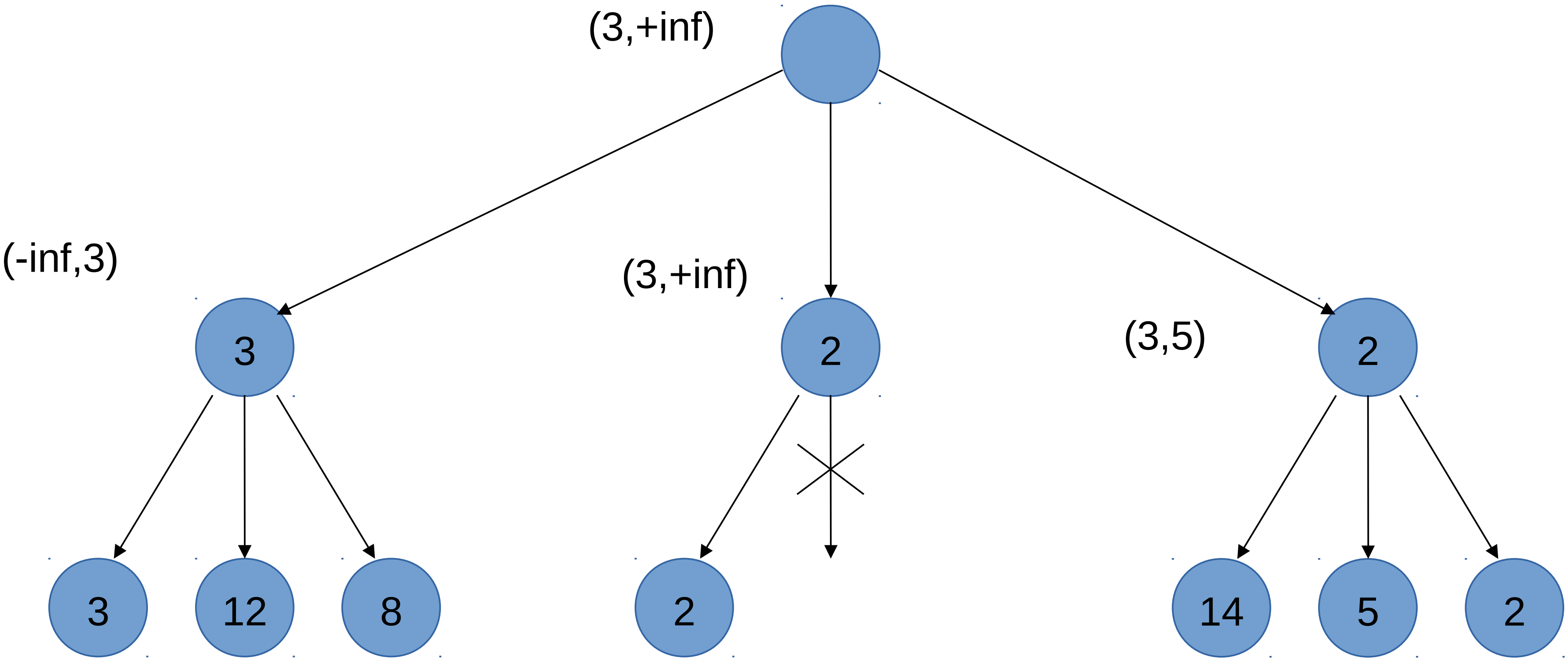
Alpha/Beta pruning



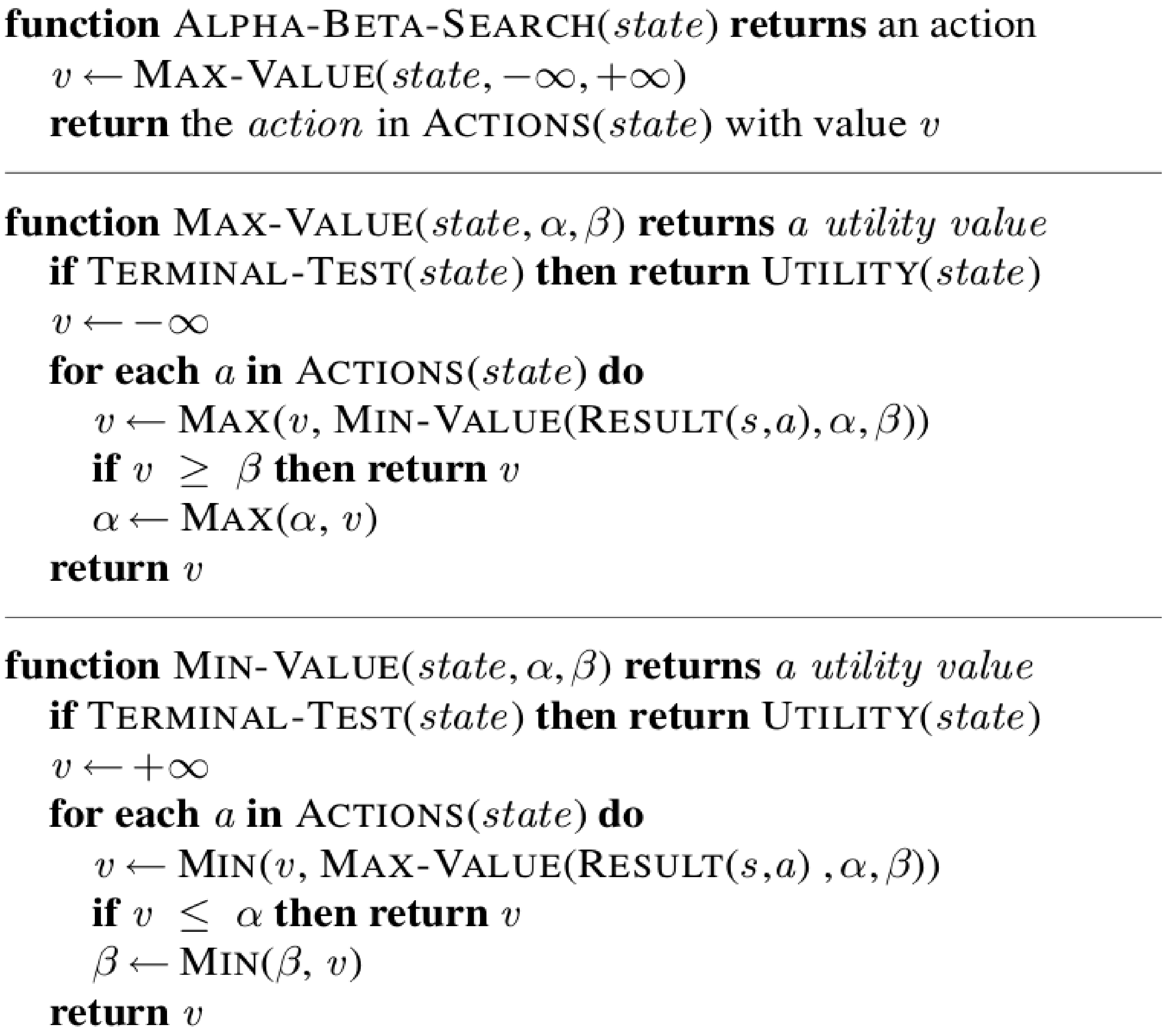
Alpha/Beta pruning



Alpha/Beta pruning



Alpha/Beta algorithm



```
function Alpha-Beta-Search(state) returns an action
    $v \leftarrow$ Max-Value(state, $-\infty, +\infty$)
    return the action in Actions(state) with value $v$

function Max-Value(state, $\alpha, \beta$) 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($s$,$a$), $\alpha, \beta$))
        if $v \geq \beta$ then return $v$
        $\alpha \leftarrow$ Max($\alpha$, $v$)
    return $v$

function Min-Value(state, $\alpha, \beta$) returns a utility value
    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($s$,$a$) ,$\alpha, \beta$))
        if $v \leq \alpha$ then return $v$
        $\beta \leftarrow$ Min($\beta$, $v$)
    return $v$
```

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?

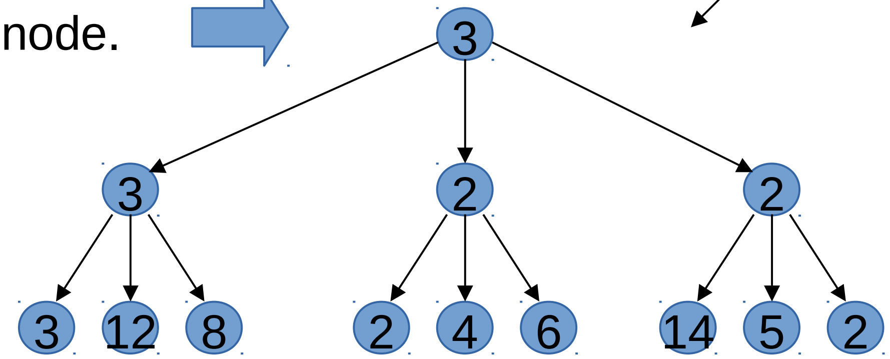
How much does alpha/beta help relative to minimax?

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?

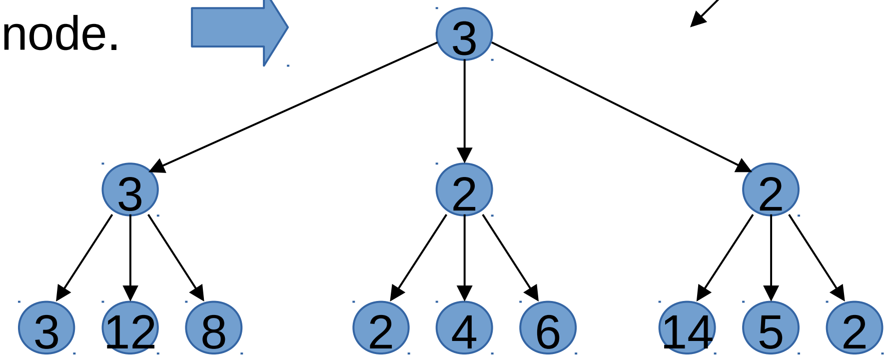
How much does alpha/beta help relative to minimax?

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

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

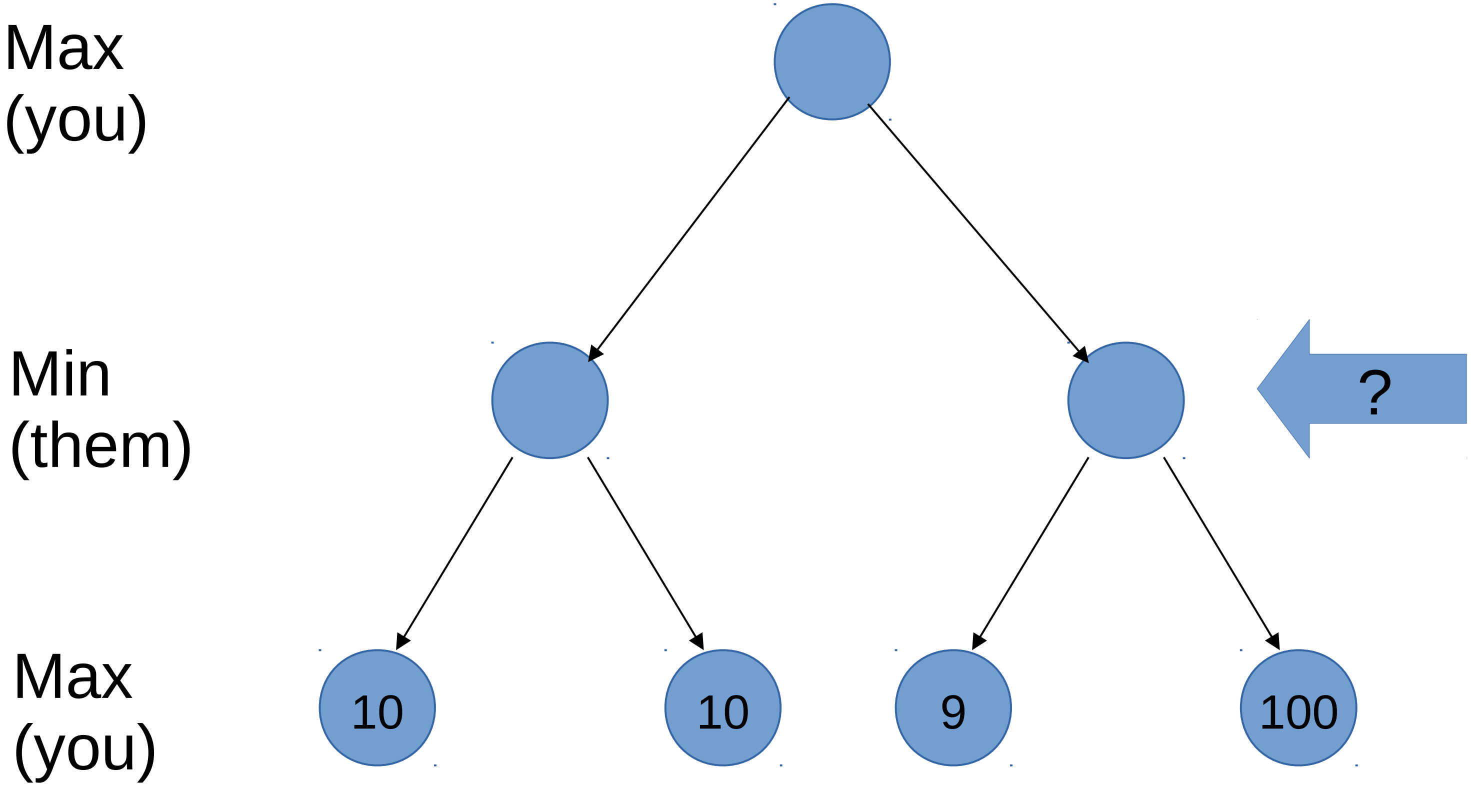
– the improvement w/ alpha/beta depends upon move ordering...

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.

Expectimax



Max
(you)

Min
(them)

Max
(you)

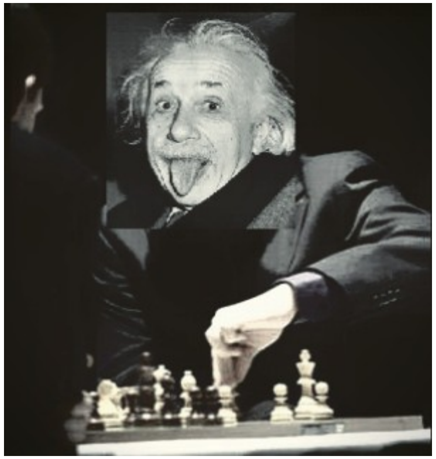
10

10

9

100

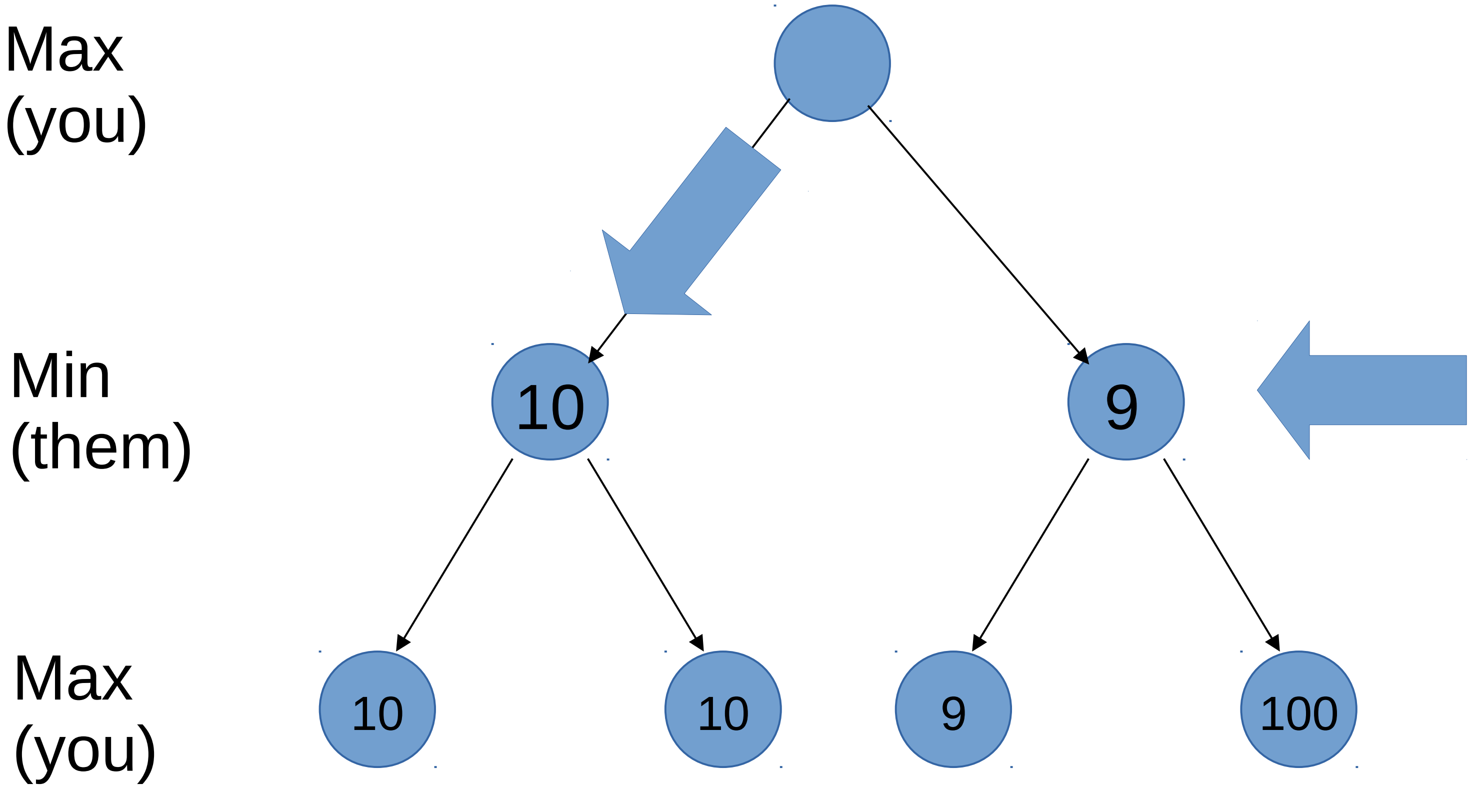
?



What if your opponent does not maximize his/her utility?
– e.g. suppose he/she picks moves uniformly at random?

Expectimax

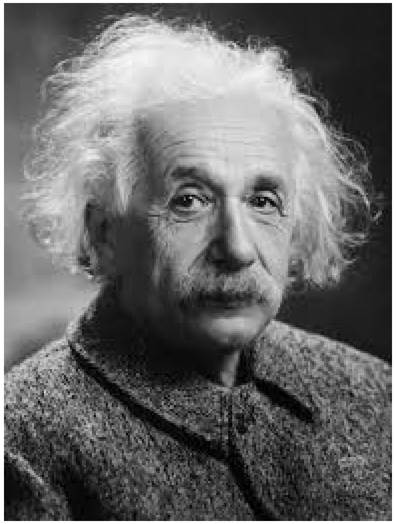
Minimax backup for a rational agent:



Max
(you)

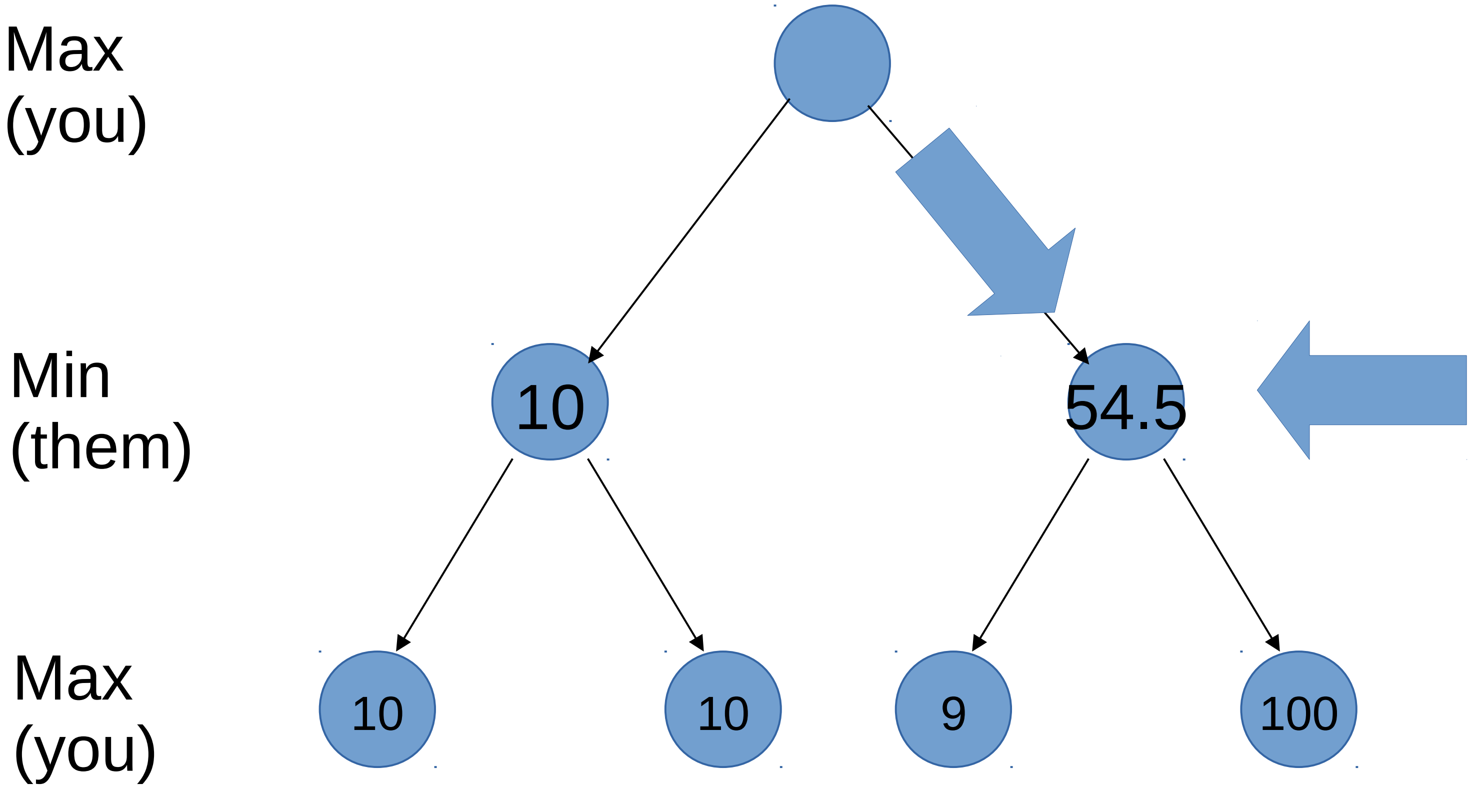
Min
(them)

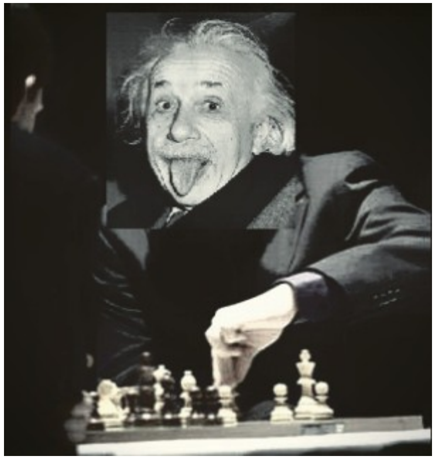
Max
(you)



Expectimax

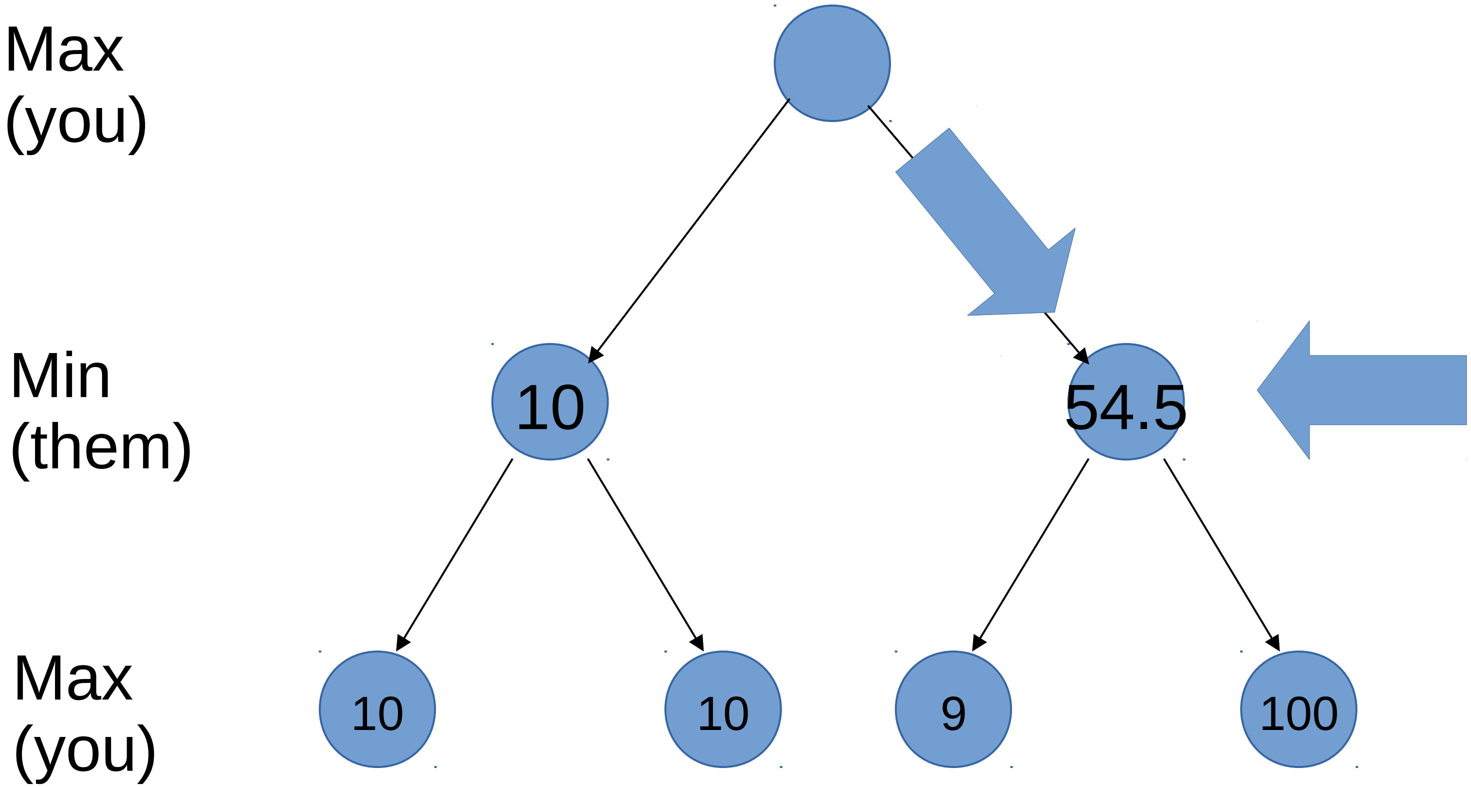
Minimax backup for agent who selects actions uniformly at random:

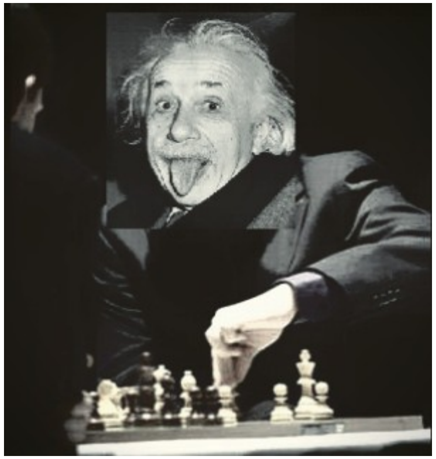




Expectimax

Minimax backup for agent who selects actions uniformly at random:





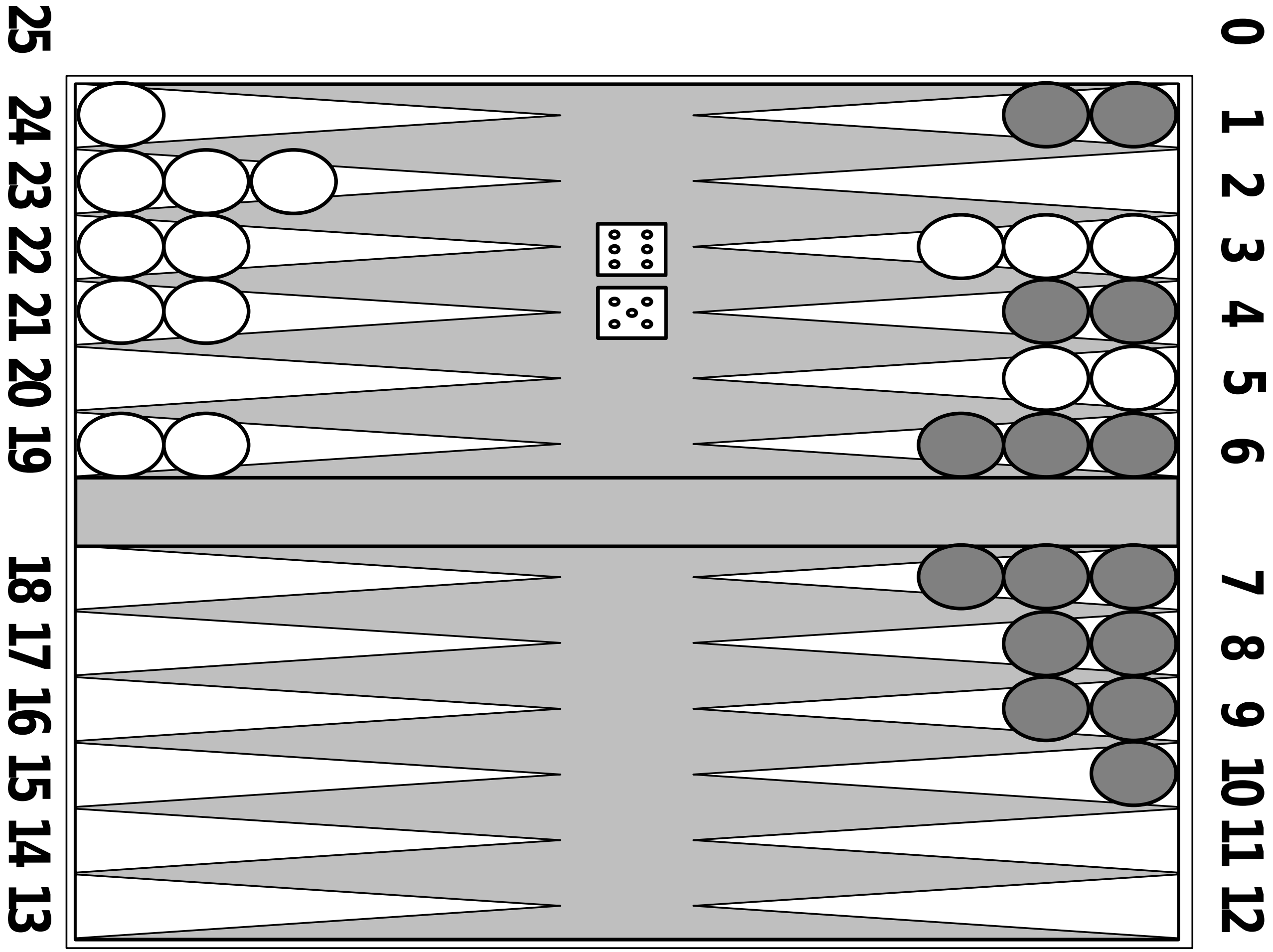
Instead of backing up min values for min-plys, back up the *average*

– could also account for agents who are somewhere in between rational and uniformly random. How?

– later, this idea will be generalized using Markov Decision Processes

Mixing these ideas: Nondeterministic games

Backgammon



Nondeterministic games in general

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

Simplified example with coin-flipping: