

# Course 8

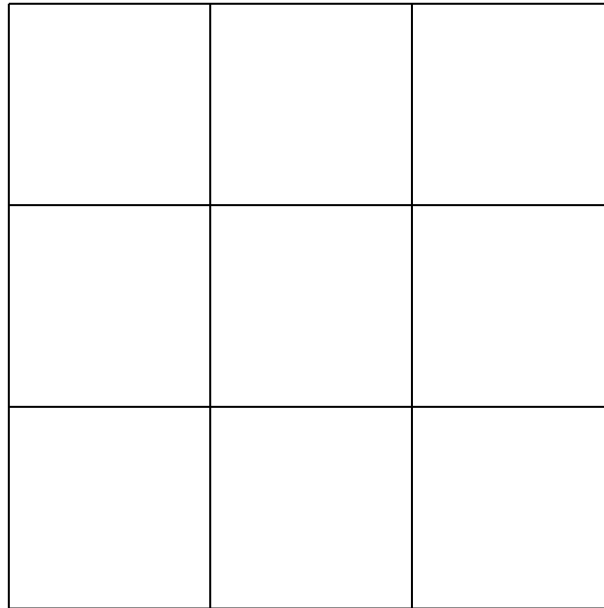
## Games

# Rules of the game

- Two players: MAX and MIN
- Both have as goal to win the game
- Only one can win or else it will be a draw
- In the initial modeling there is no chance (but it can be simulated)
- Examples:
  - chess
  - checkers
  - tic-tac-toe
  - ...

# The tic-tac-toe game

MAX plays with Xs



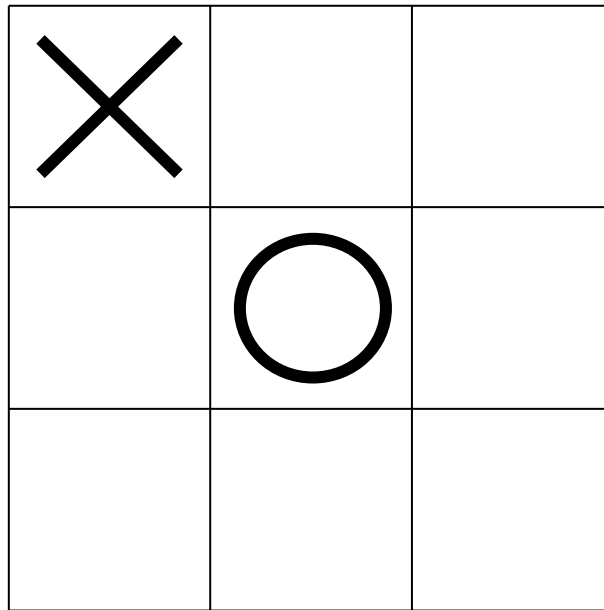
MIN plays with Os

# The tic-tac-toe game

MAX

X		

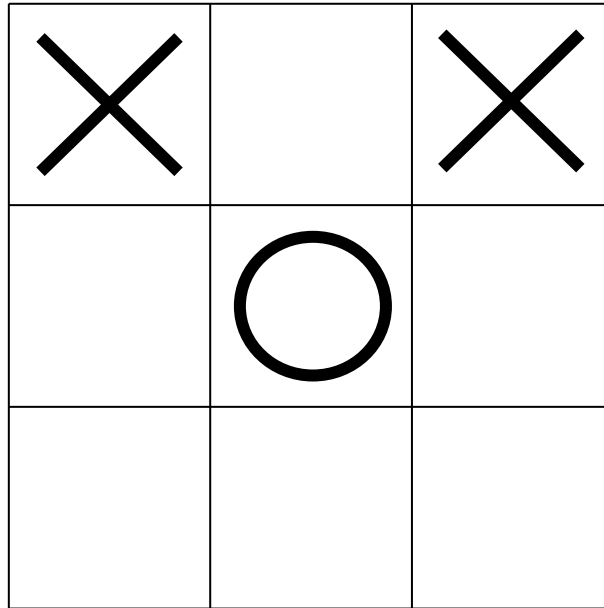
# The tic-tac-toe game



MIN

# The tic-tac-toe game

MAX



# The tic-tac-toe game

X	O	X
	O	

MIN

# The tic-tac-toe game

MAX

X	O	X
	O	
	X	



# The tic-tac-toe game

X	O	X
	O	O
	X	

MIN

# The tic-tac-toe game

MAX

X	O	X
X	O	O
	X	

# The tic-tac-toe game

X	O	X
X	O	O
O	X	

MIN

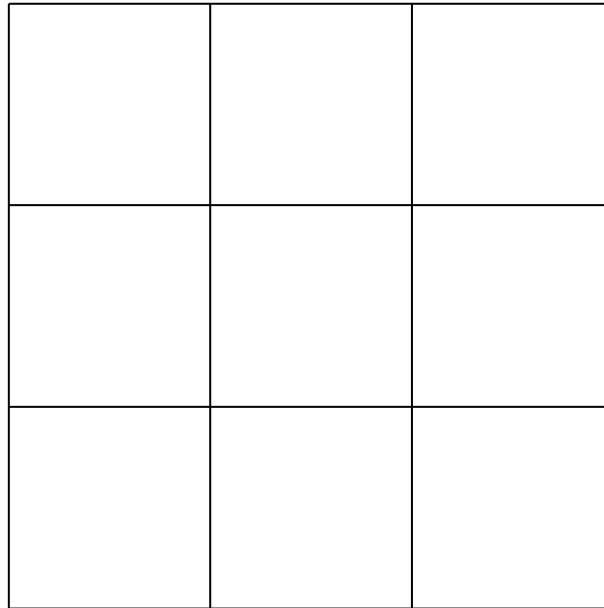
# The tic-tac-toe game

MAX

X	O	X
X	O	O
O	X	X

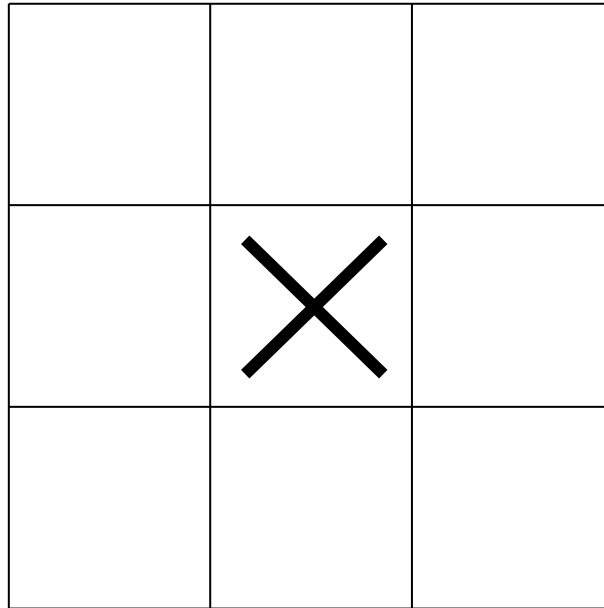
Draw!

# The tic-tac-toe game

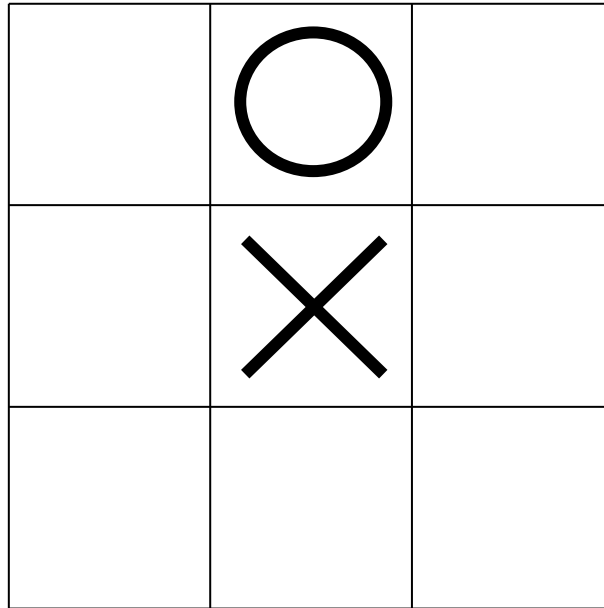


# The tic-tac-toe game

MAX



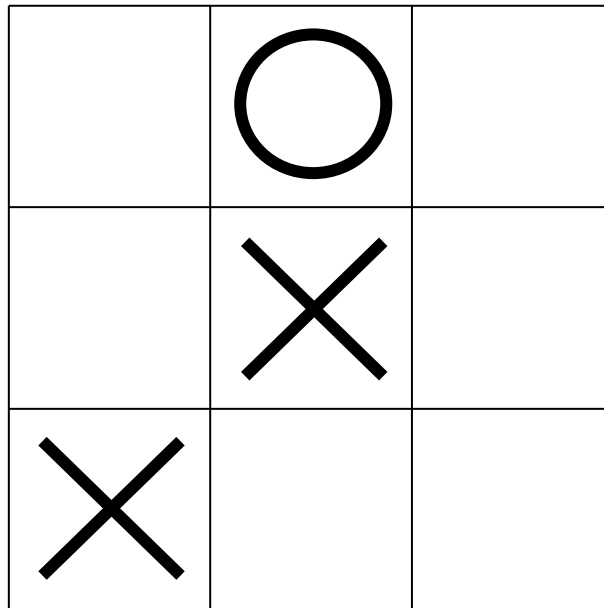
# The tic-tac-toe game



MIN

# The tic-tac-toe game

MAX





# The tic-tac-toe game

	○	○
	×	
×		

MIN

# The tic-tac-toe game

MAX

X	O	O
	X	
X		

# The tic-tac-toe game

X	O	O
	X	
X		O

MIN

# The tic-tac-toe game

MAX

X	O	O
X	X	
X		O

# The tic-tac-toe game

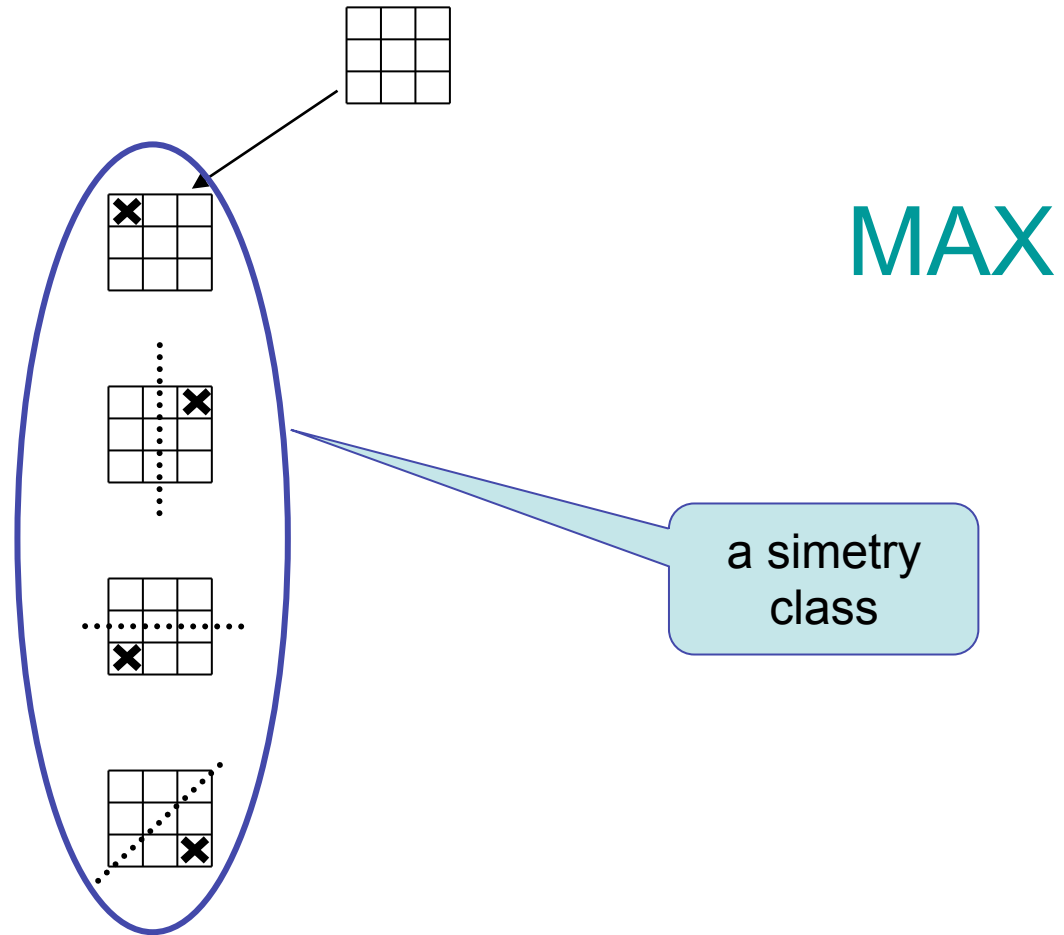
MAX wins

X	O	O
X	X	
X		O

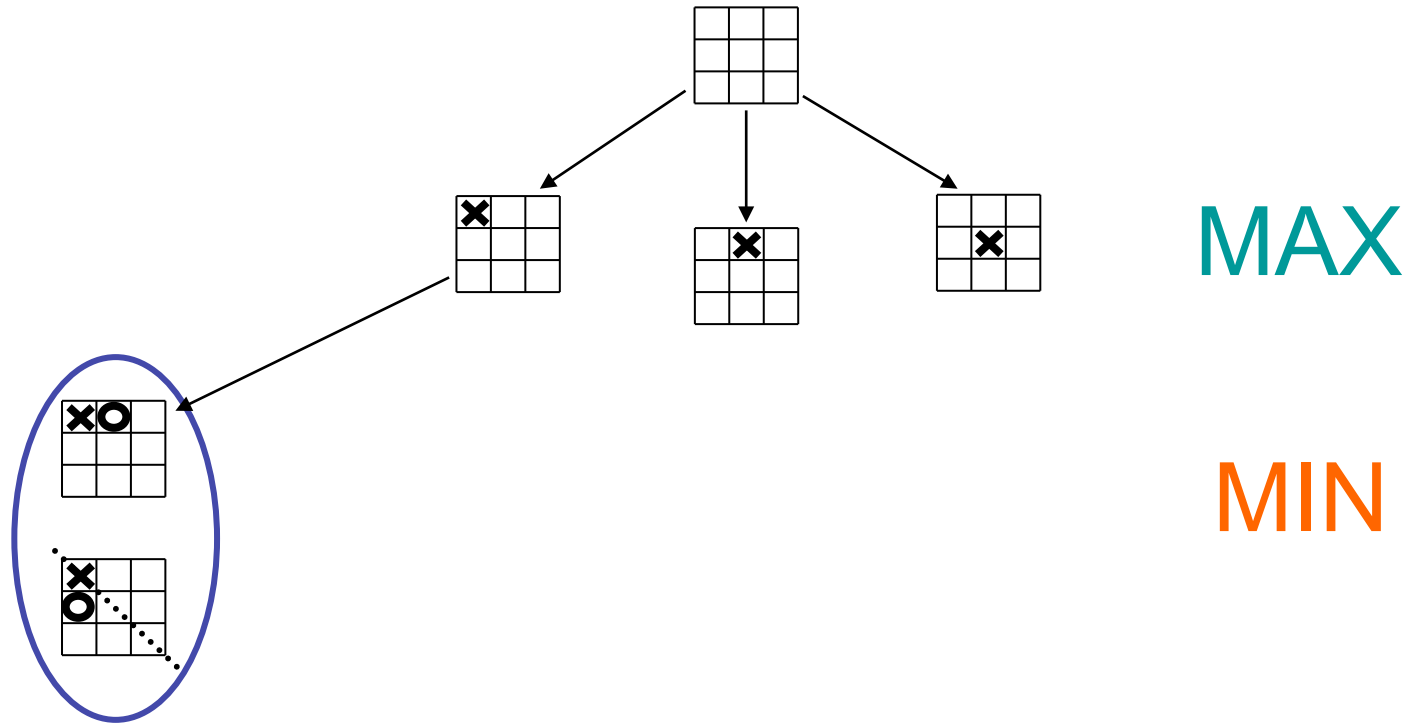
# Representation as an AI problem

1. Problem versus instance
2. The state space:
  - a state: the position on the board of the signs between two moves
  - the size of the space:  $3^9$
3. Representing a state:
  - a 3x3 matrix
4. Representing a transition
  - algorithmical (in the present approach)
5. How is it controlled the evolution of the game?
  - the MIN-MAX method
  - the ALPHA-BETA method

# The game tree

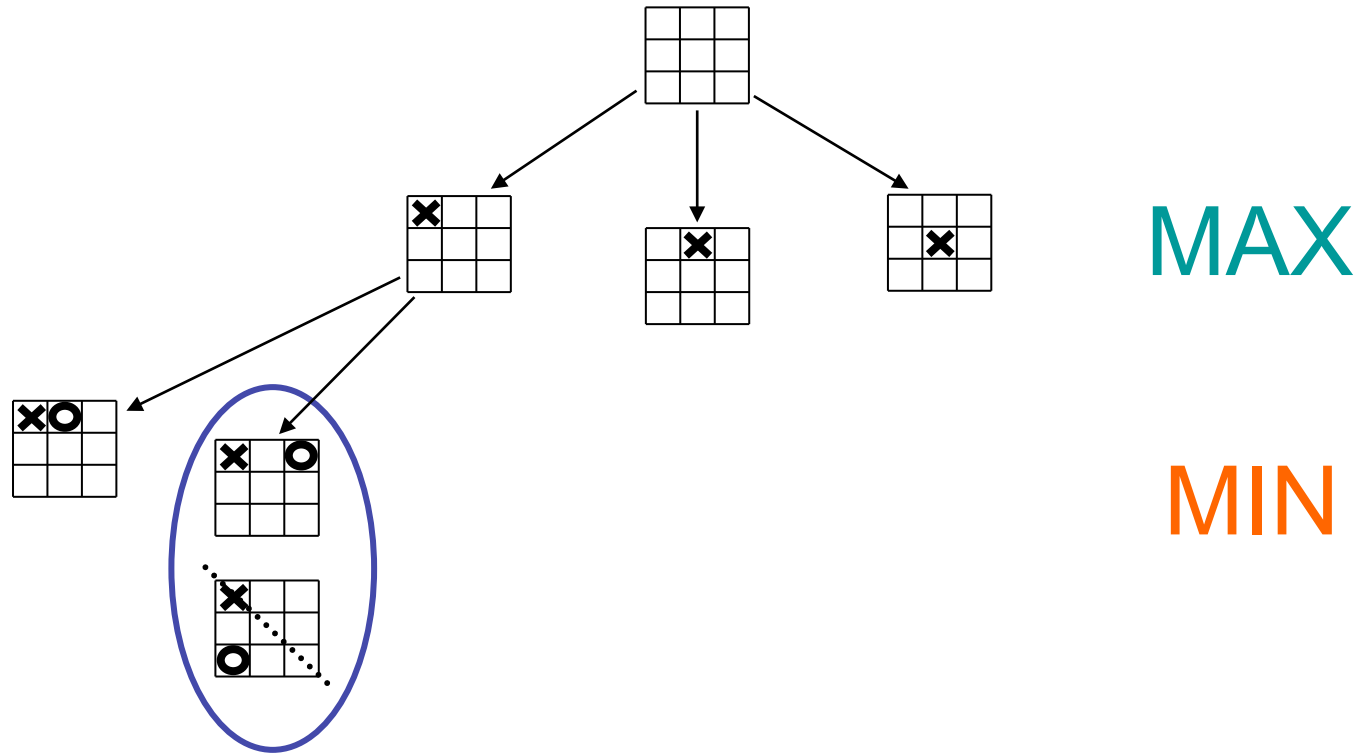


# The game tree

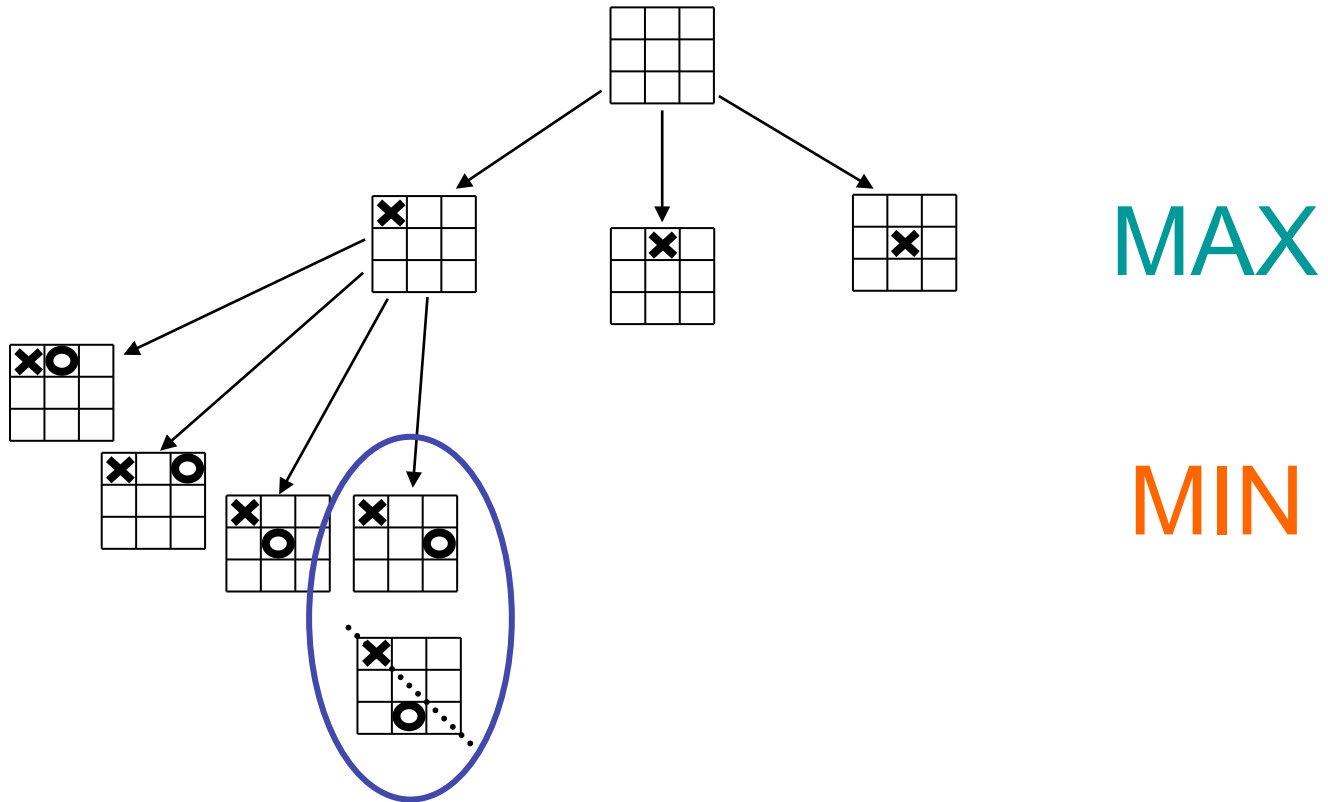




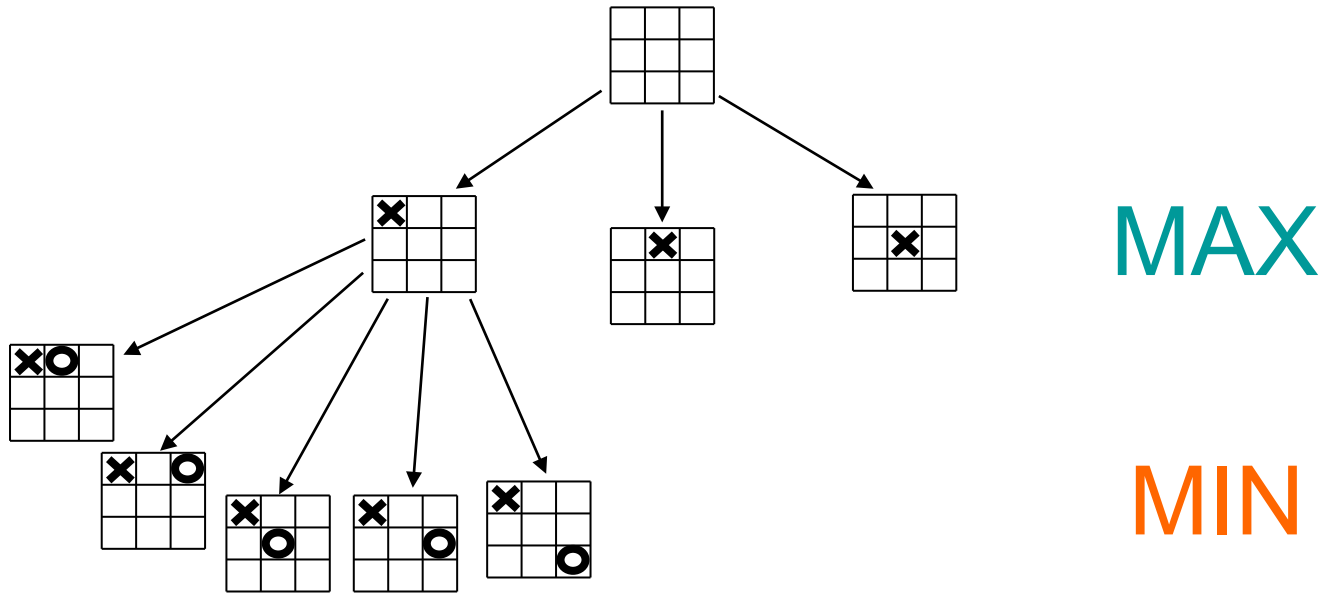
# The game tree



# The game tree



# The game tree



# The value of a state

MAX wins:  $+\infty$

X	O	
	X	O
O		X

# The value of a state

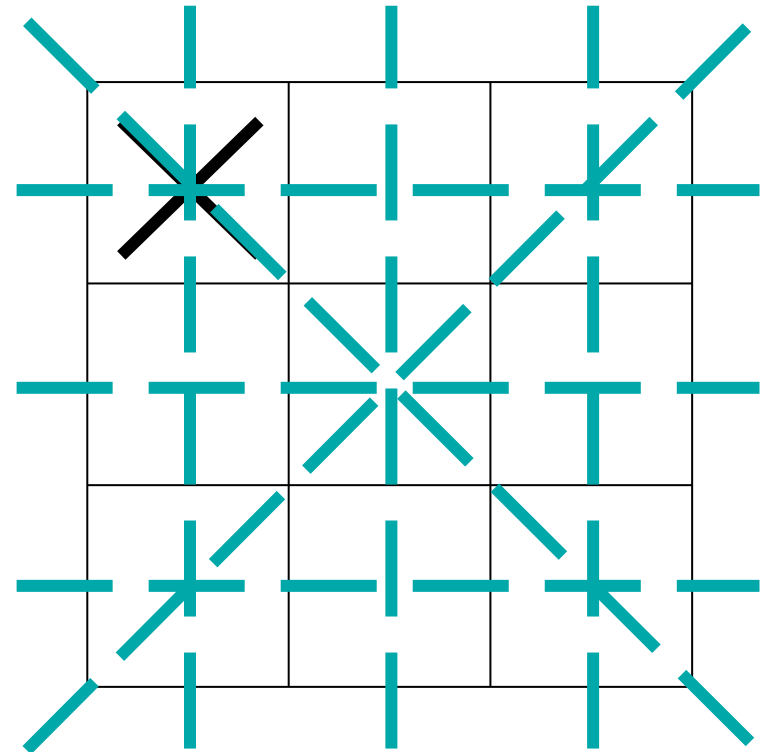
MIN wins:  $-\infty$

X	X	O
	O	
O		X

# Evaluating a state

A state is better if it opens up more win possibilities until the end of the game.

An example of an evaluation function:  
*the value of the state is the difference between the number of lines that MAX could fill in till the end of the game and those that MIN could fill in till the end of the game.*

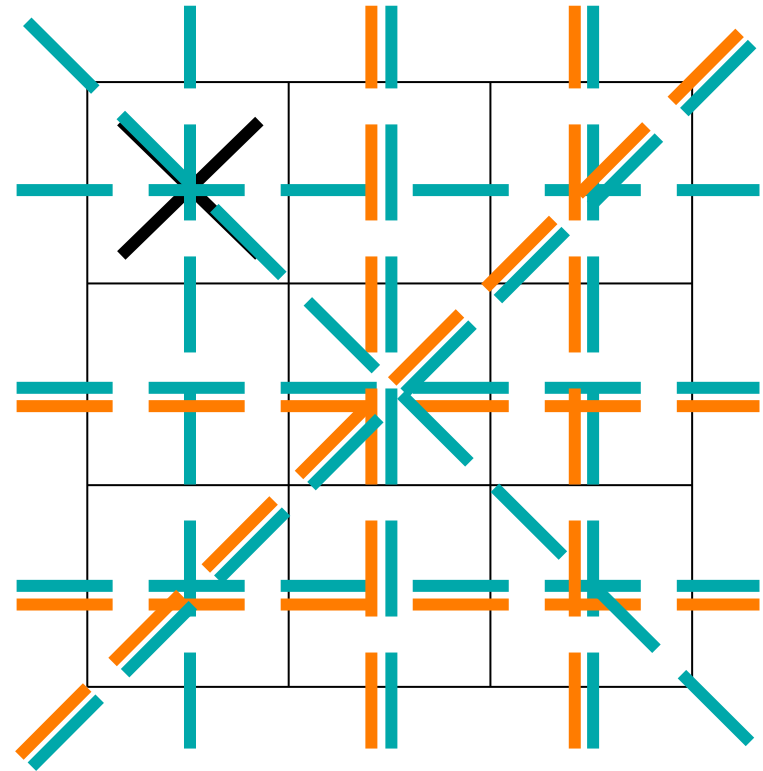


8

# Evaluating a state

A state is better if it opens up more win possibilities until the end of the game.

An example of an evaluation function:  
*the value of the state is the difference between the number of lines that MAX could fill in till the end of the game and those that MIN could fill in till the end of the game.*



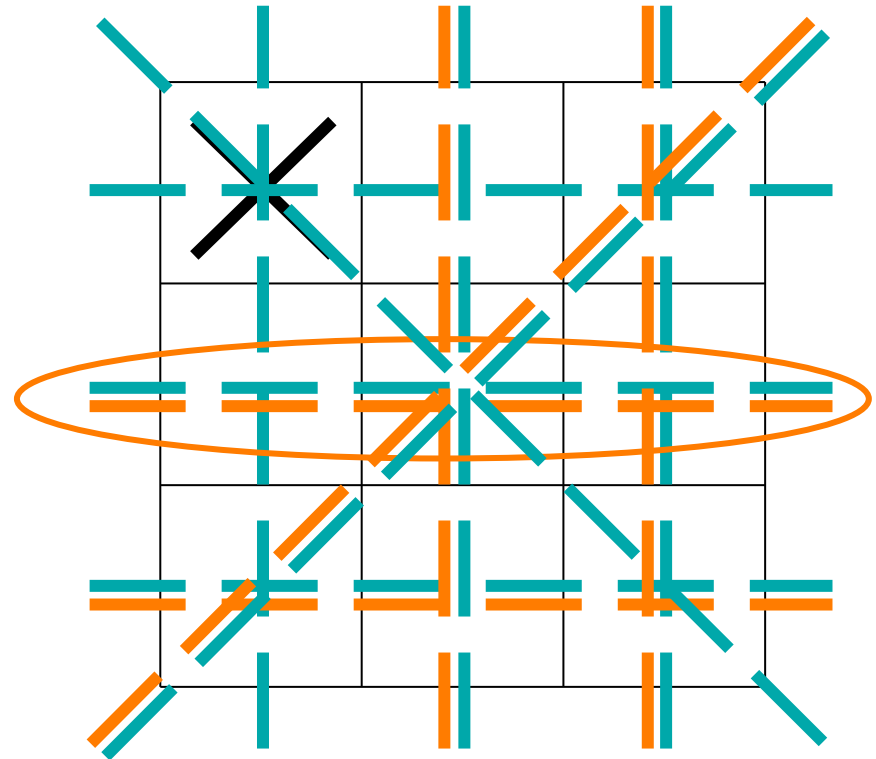
$$8 - 5 = 3$$

# Evaluating a state

A state is better if it opens up more win possibilities until the end of the game.

An example of an evaluation function:  
*the value of the state is the difference between the number of lines that MAX could fill in till the end of the game and those that MIN could fill in till the end of the game.*

Lines having no sign could be taken by any of the players...



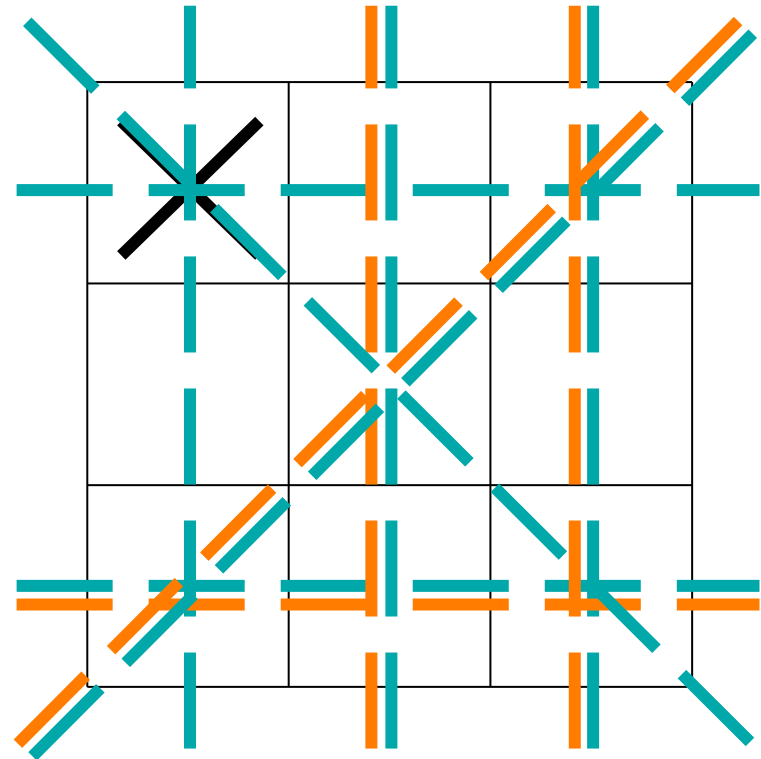


# Evaluating a state

A state is better if it opens up more win possibilities until the end of the game.

An example of an evaluation function:  
*the value of the state is the difference between the number of lines that MAX could fill in till the end of the game and those that MIN could fill in till the end of the game.*

Lines having no sign could be taken by any of the players...

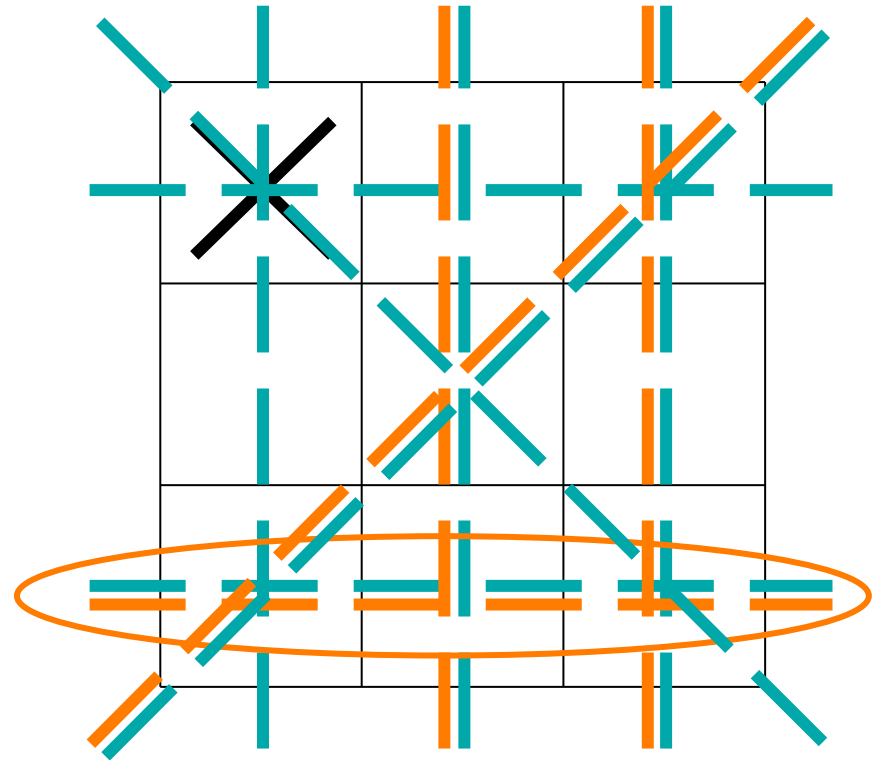


# Evaluating a state

A state is better if it opens up more win possibilities until the end of the game.

An example of an evaluation function:  
*the value of the state is the difference between the number of lines that MAX could fill in till the end of the game and those that MIN could fill in till the end of the game.*

Lines having no sign could be taken by any of the players...

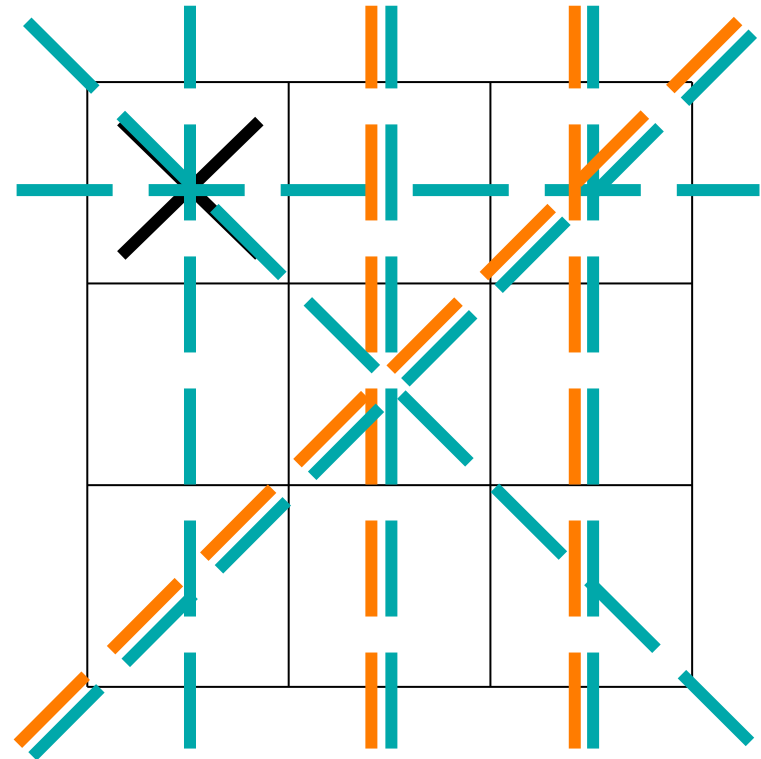


# Evaluating a state

A state is better if it opens up more win possibilities until the end of the game.

An example of an evaluation function:  
*the value of the state is the difference between the number of lines that MAX could fill in till the end of the game and those that MIN could fill in till the end of the game.*

Lines having no sign could be taken by any of the players...

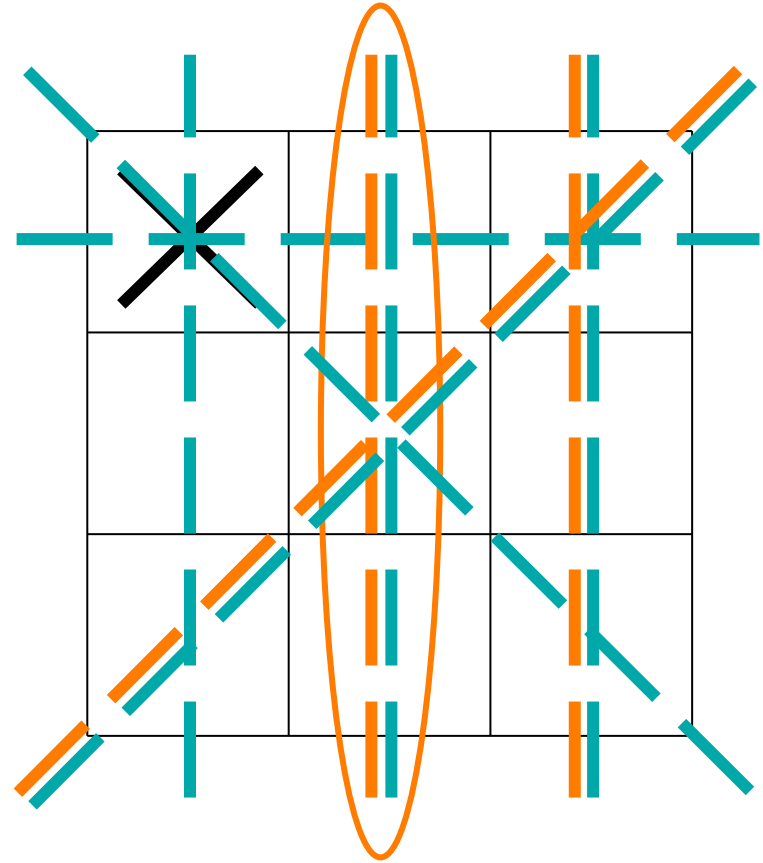


# Evaluating a state

A state is better if it opens up more win possibilities until the end of the game.

An example of an evaluation function:  
*the value of the state is the difference between the number of lines that MAX could fill in till the end of the game and those that MIN could fill in till the end of the game.*

Lines having no sign could be taken by any of the players...



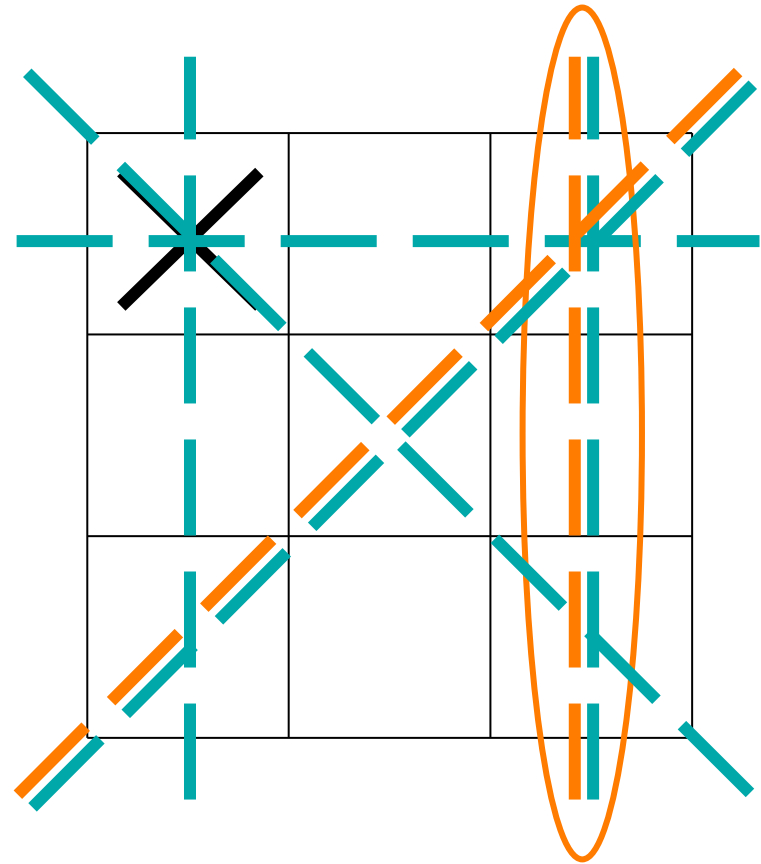


# Evaluating a state

A state is better if it opens up more win possibilities until the end of the game.

An example of an evaluation function:  
*the value of the state is the difference between the number of lines that MAX could fill in till the end of the game and those that MIN could fill in till the end of the game.*

Lines having no sign could be taken by any of the players...

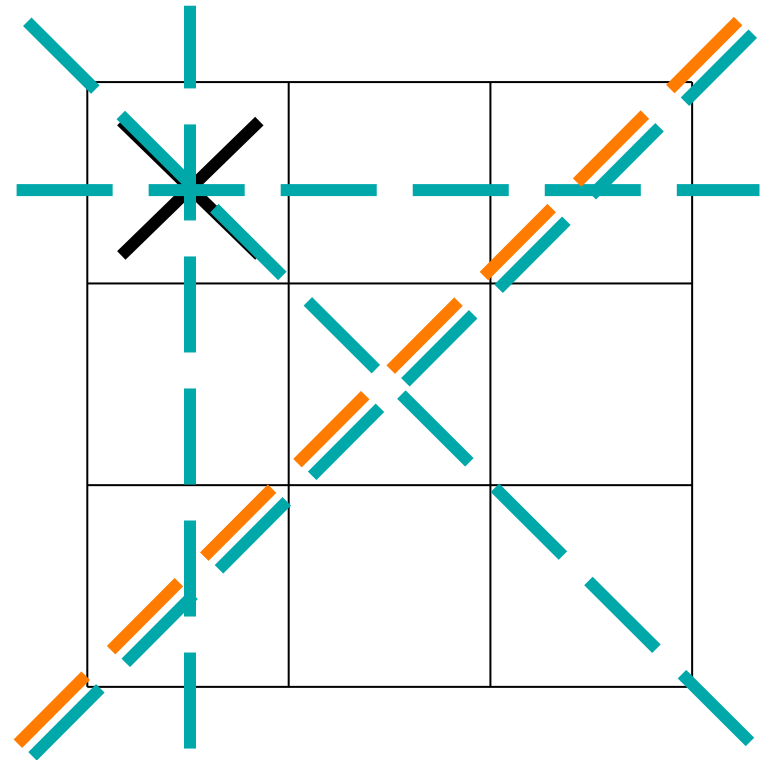


# Evaluating a state

A state is better if it opens up more win possibilities until the end of the game.

An example of an evaluation function:  
*the value of the state is the difference between the number of lines that MAX could fill in till the end of the game and those that MIN could fill in till the end of the game.*

Lines having no sign could be taken by any of the players...

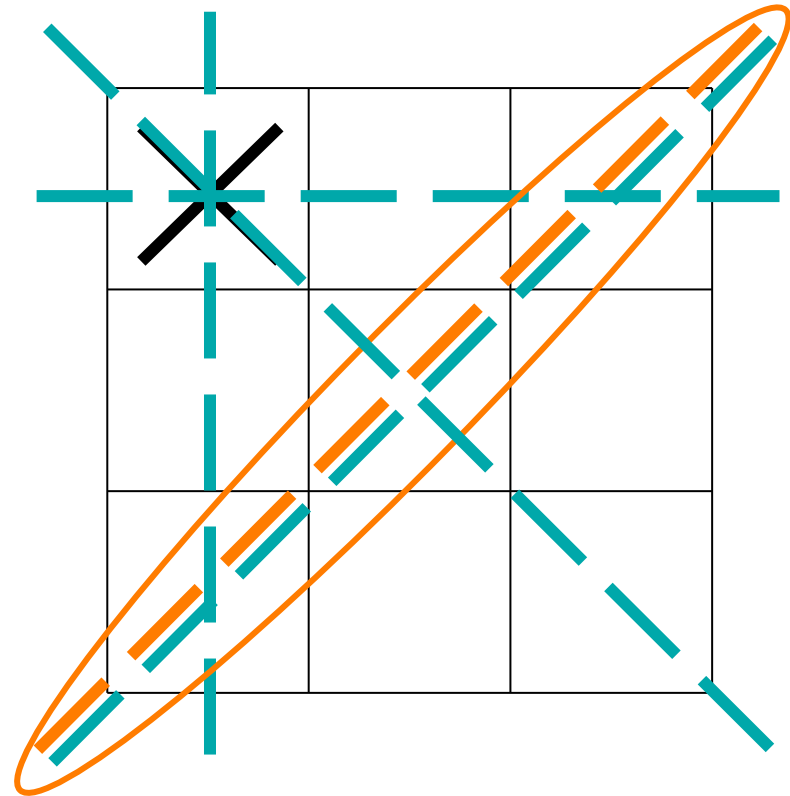


# Evaluating a state

A state is better if it opens up more win possibilities until the end of the game.

An example of an evaluation function:  
*the value of the state is the difference between the number of lines that MAX could fill in till the end of the game and those that MIN could fill in till the end of the game.*

Lines having no sign could be taken by any of the players...



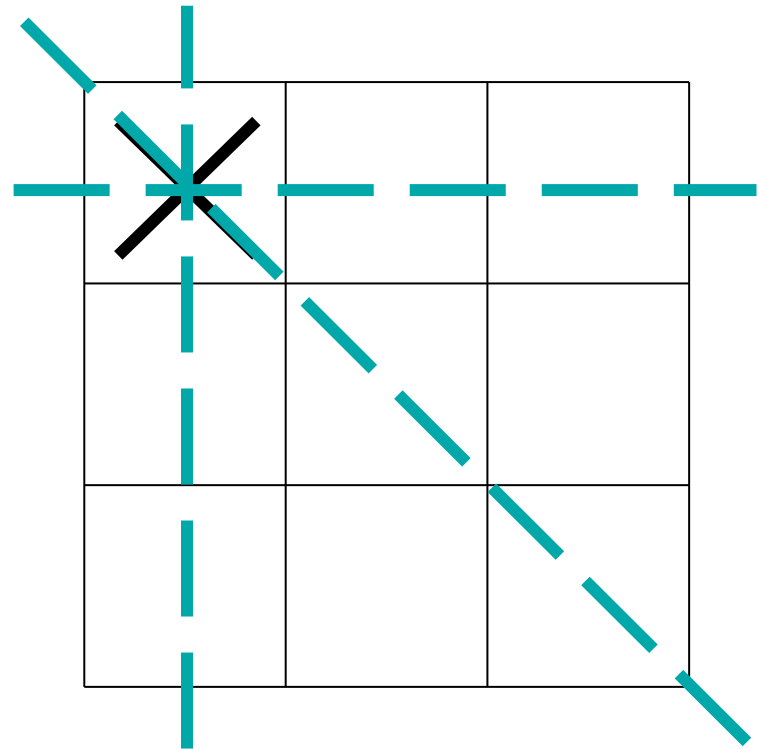


# Evaluating a state

A state is better if it opens up more win possibilities until the end of the game.

An example of an evaluation function:  
*the value of the state is the difference between the number of lines that MAX could fill in till the end of the game and those that MIN could fill in till the end of the game.*

Lines having no sign could be taken by any of the players...

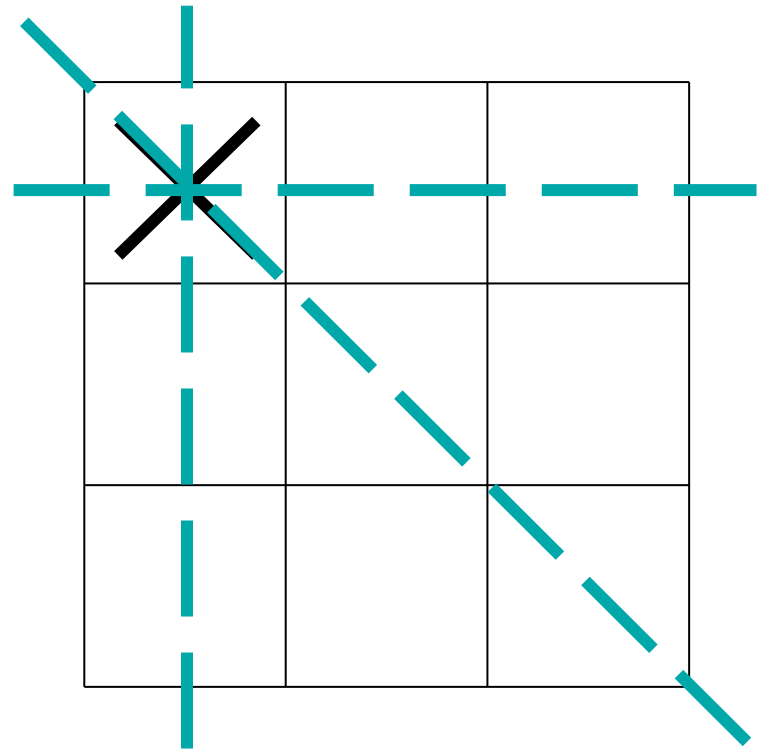


# Evaluating a state

A state is better if it opens up more win possibilities until the end of the game.

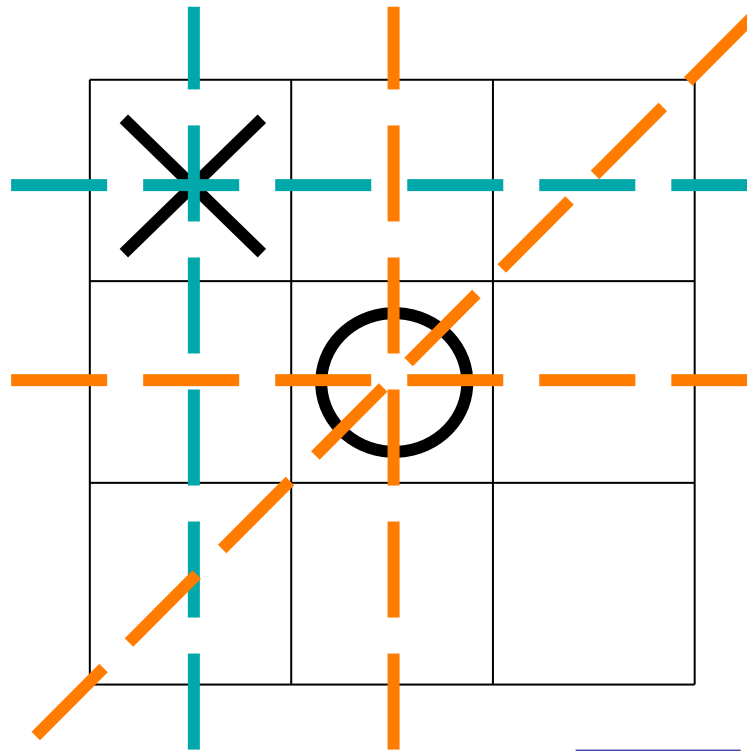
An example of an evaluation function:  
*the value of the state is the difference between the number of lines that MAX could fill in till the end of the game and those that MIN could fill in till the end of the game.*

Lines having no sign could be taken by any of the players...



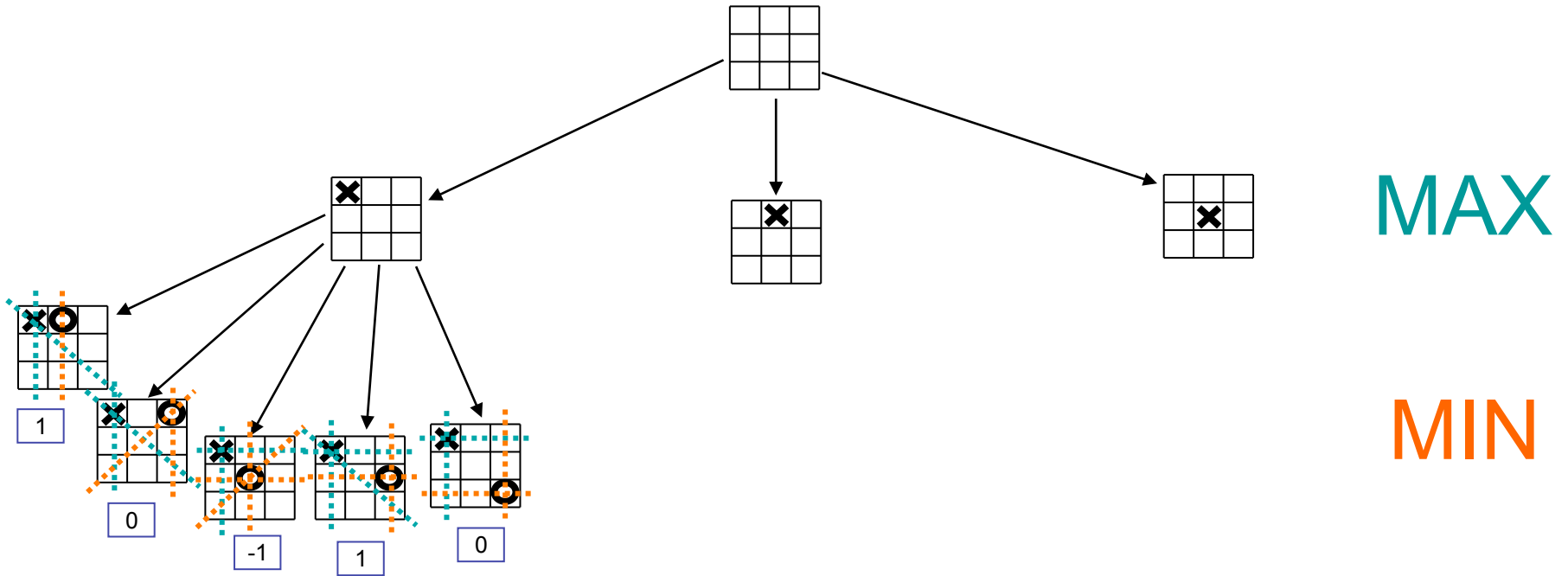
$$3 - 0 = 3$$

# Evaluating a state

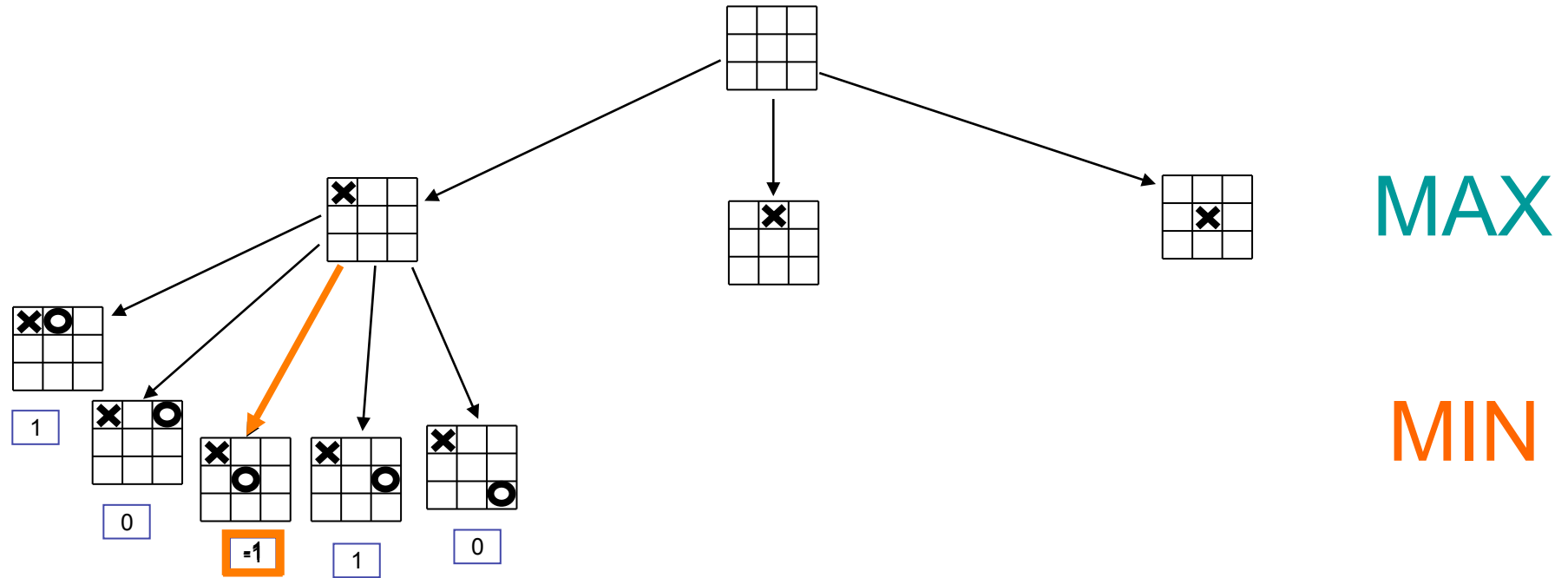


$$2 - 3 = -1$$

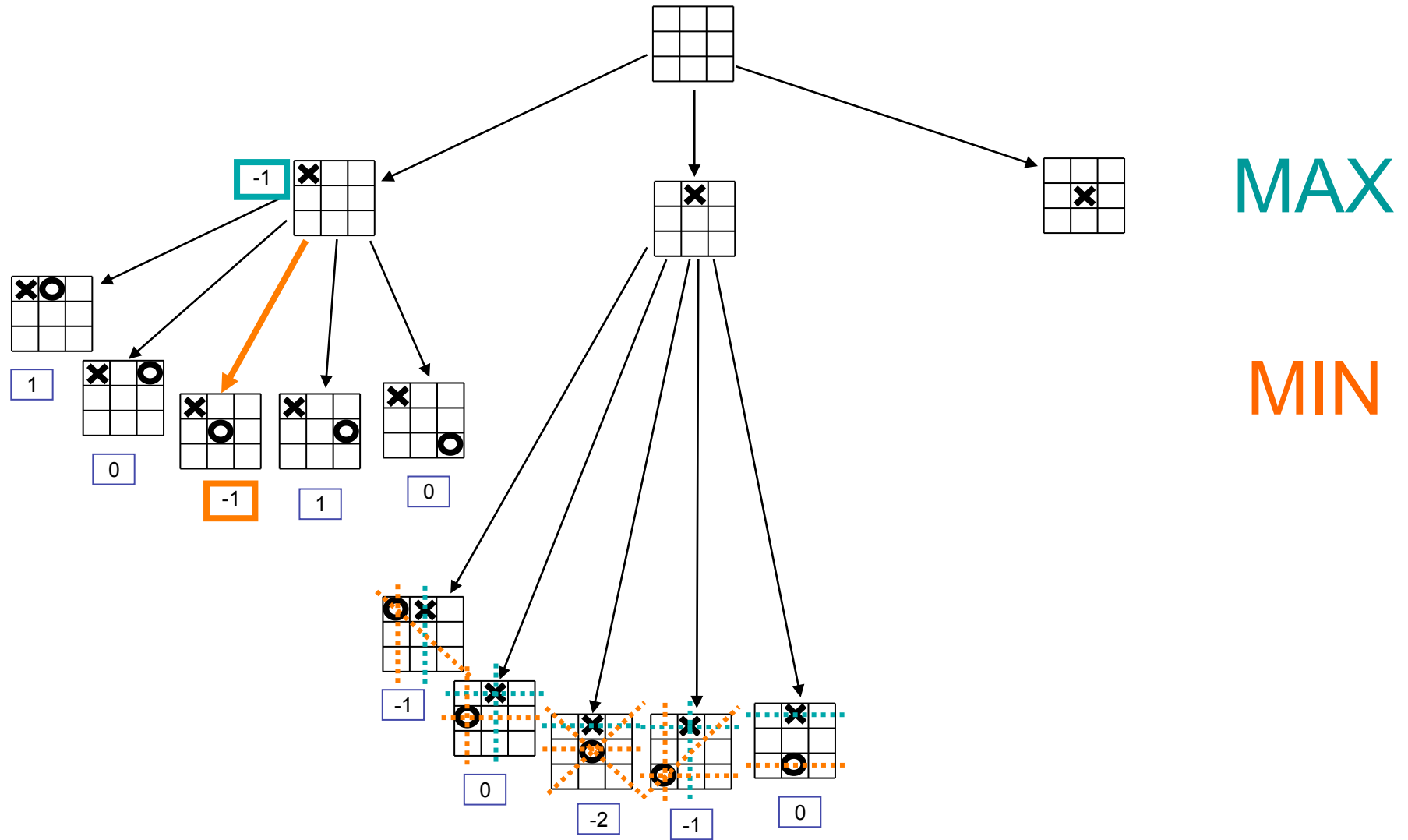
# Evaluation: bottom-up



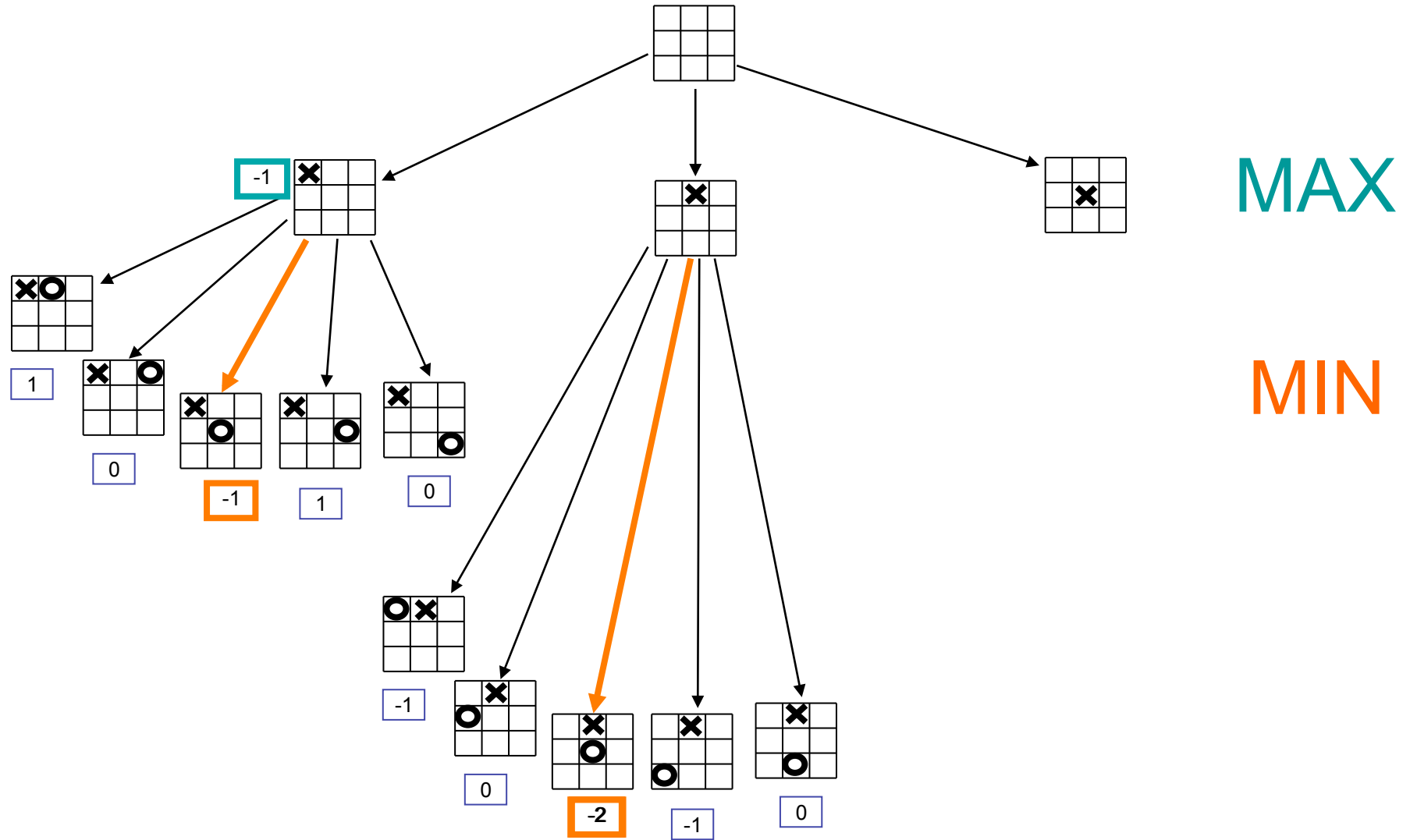
# Evaluation: bottom-up



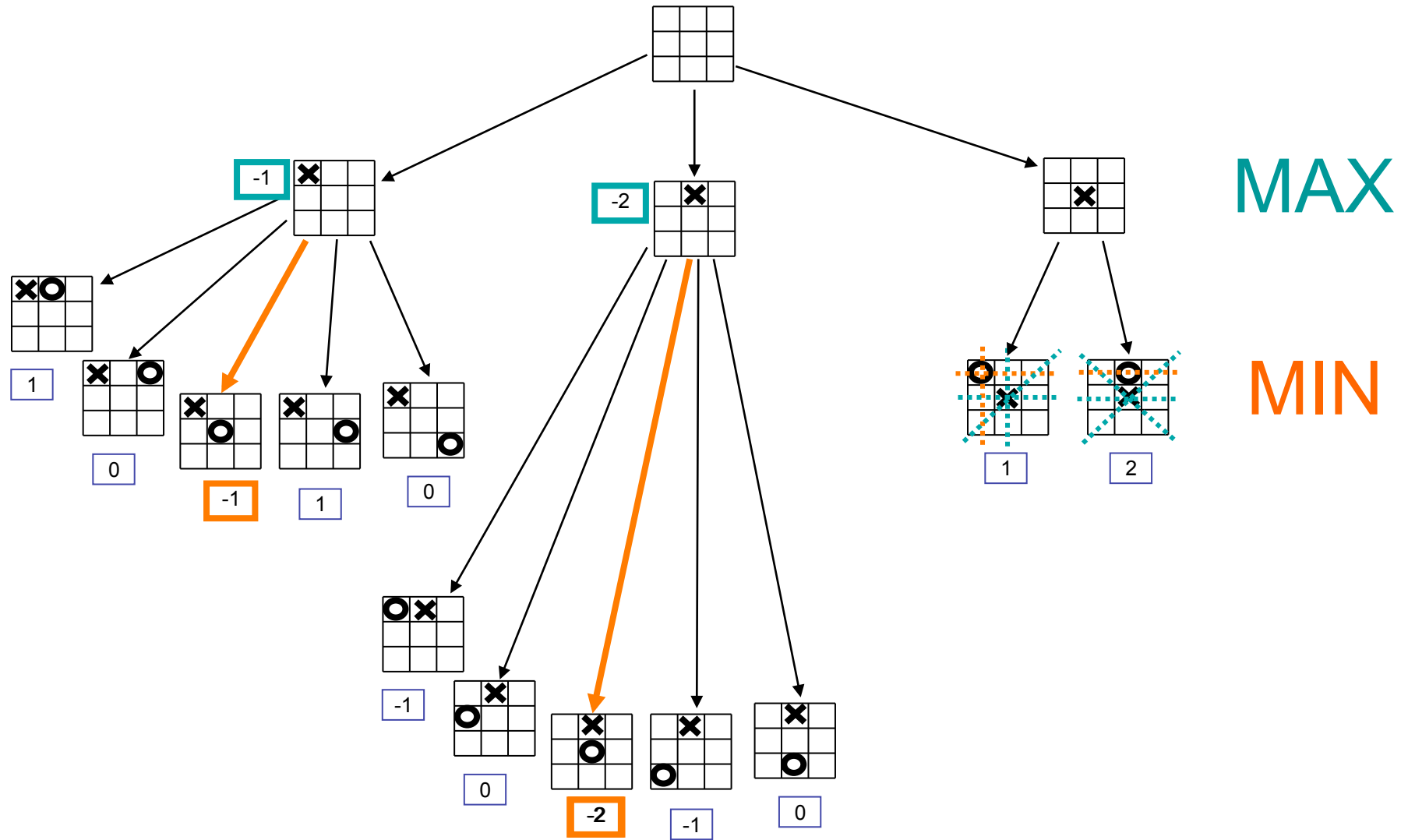
# Evaluation: bottom-up



# Evaluation: bottom-up

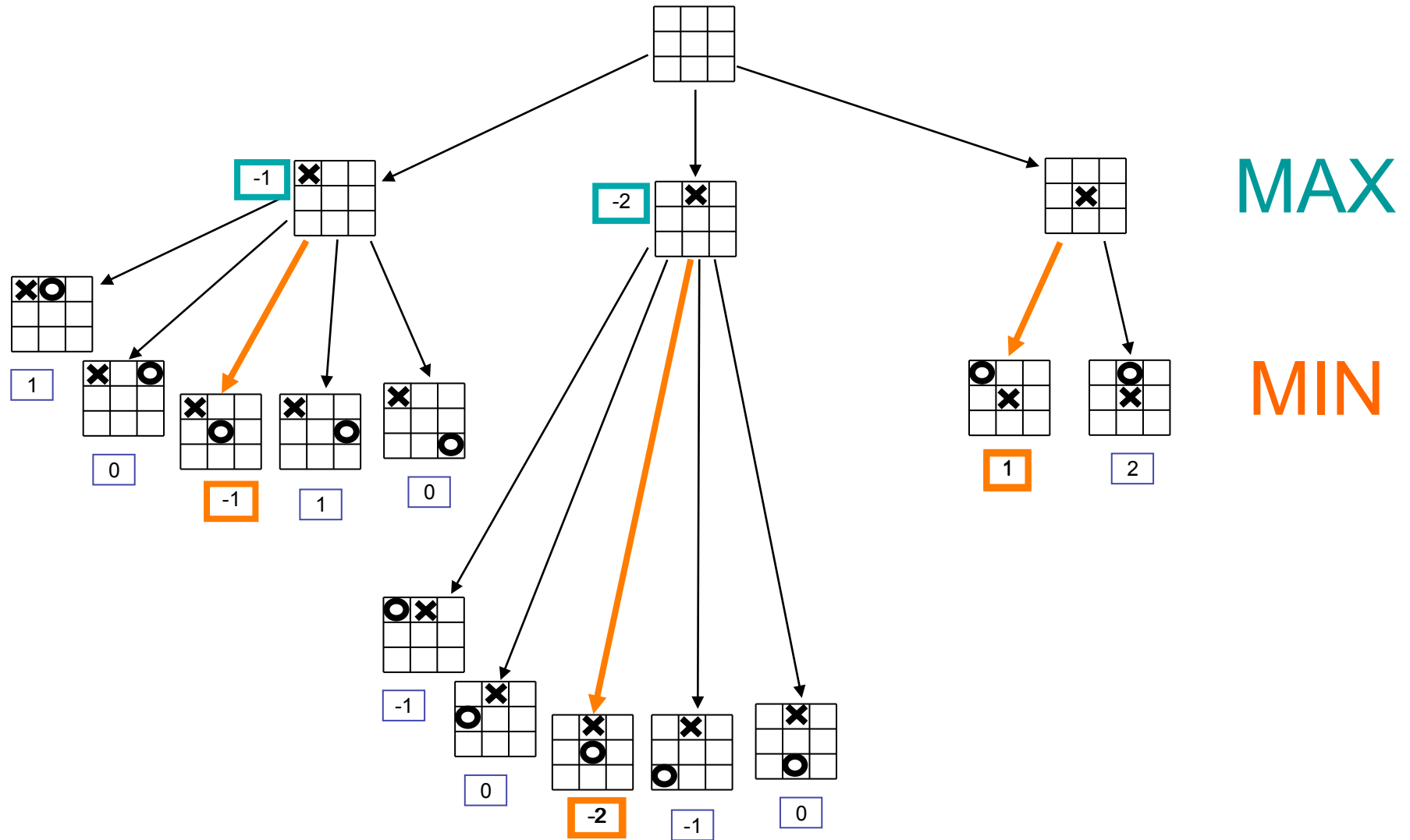


# Evaluation: bottom-up

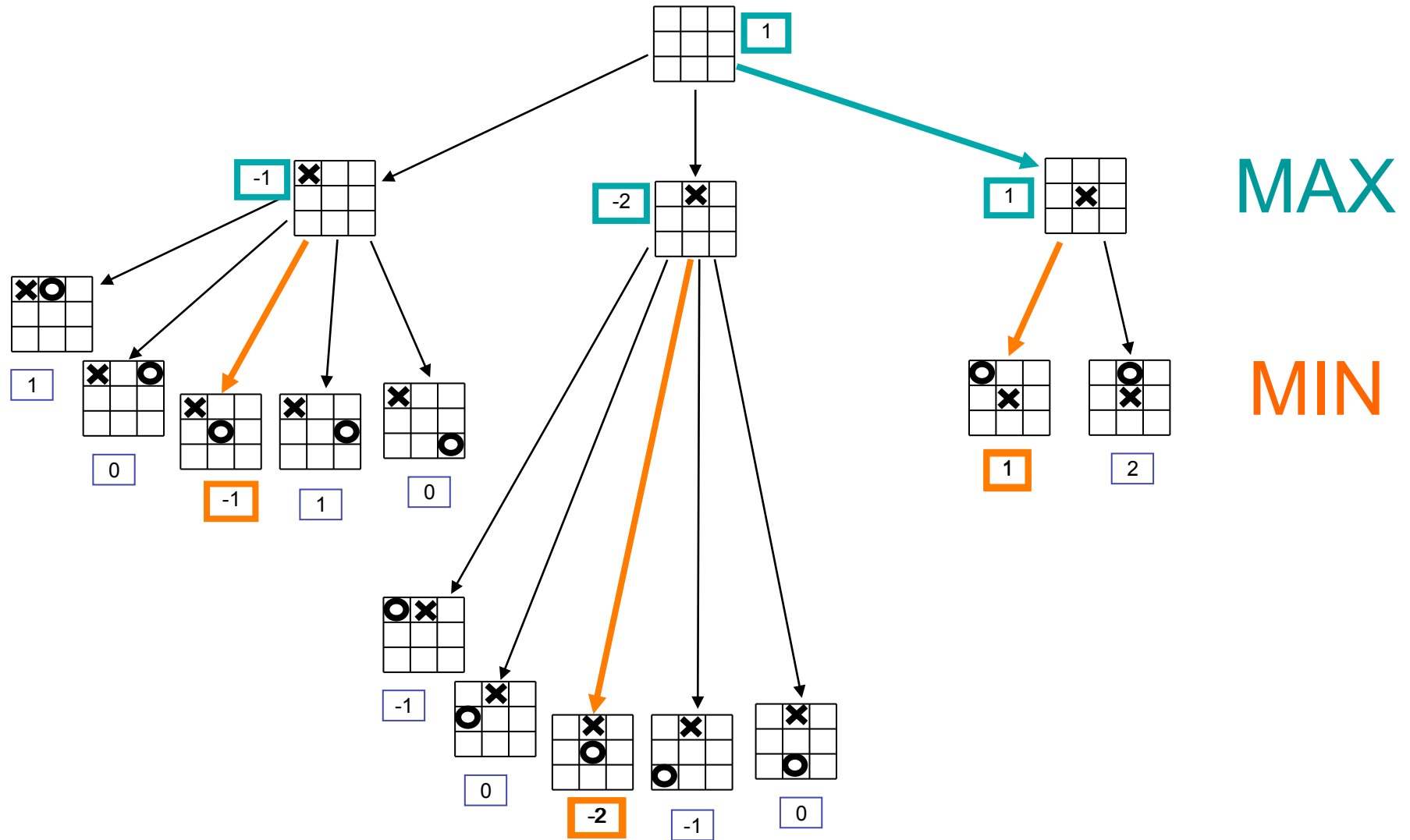




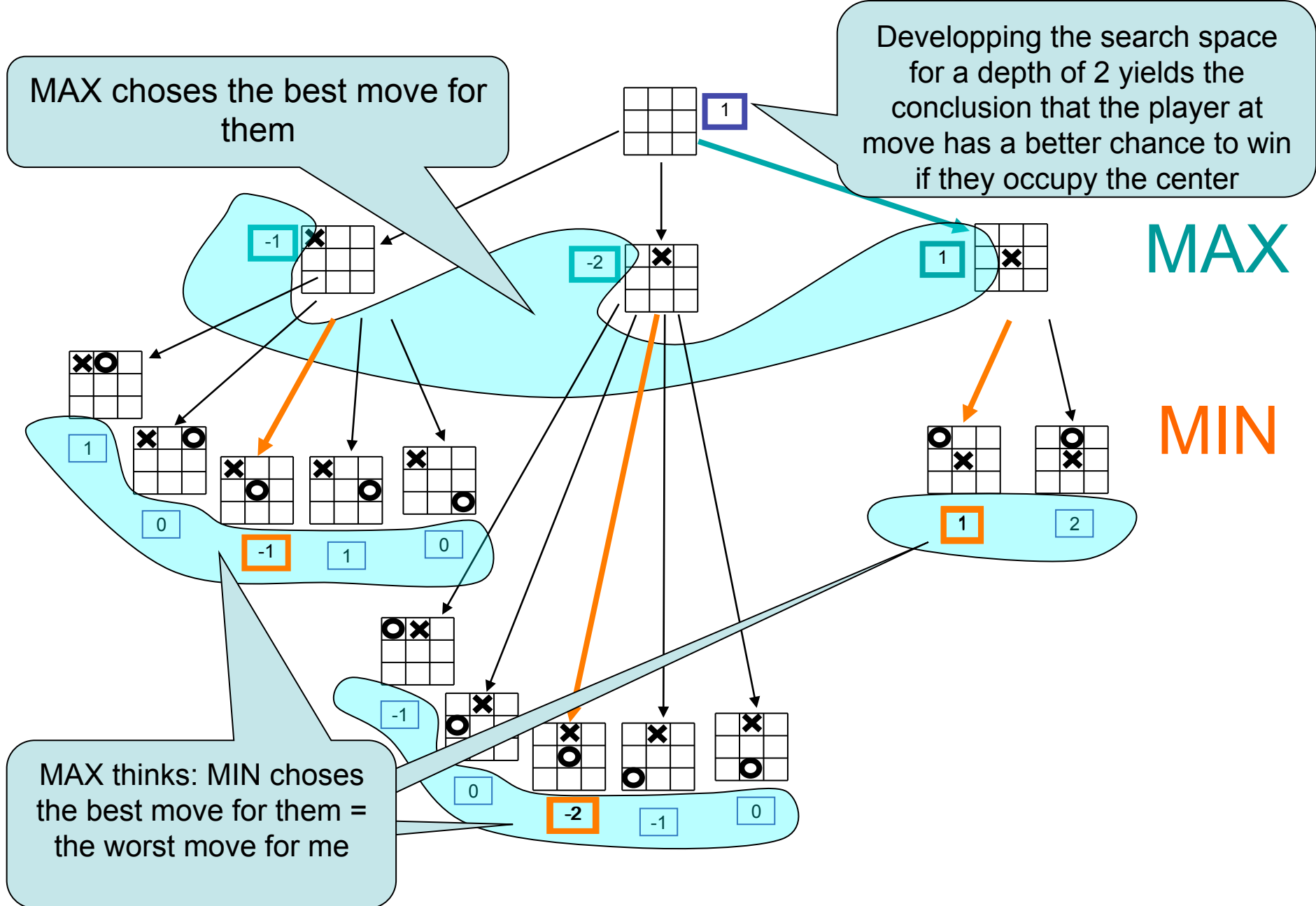
# Evaluation: bottom-up



# Evaluation: bottom-up



# Evaluation: bottom-up




# The MIN-MAX method

```
function min-max(state, player, depth)  
begin  
  if (depth = 0) then return score(state);  
  val = worst(player);  
  while (there are still states to generate) begin  
    generate a state -> s;  
    val <- back-up-compare(val, min-max(s, not(player), depth-1), player);  
    // the following instruction reduces the search space in case a win is reached in one of the generated states:  
    if (val = -worst(player)) return(val);  
  end  
  return(val);  
end
```

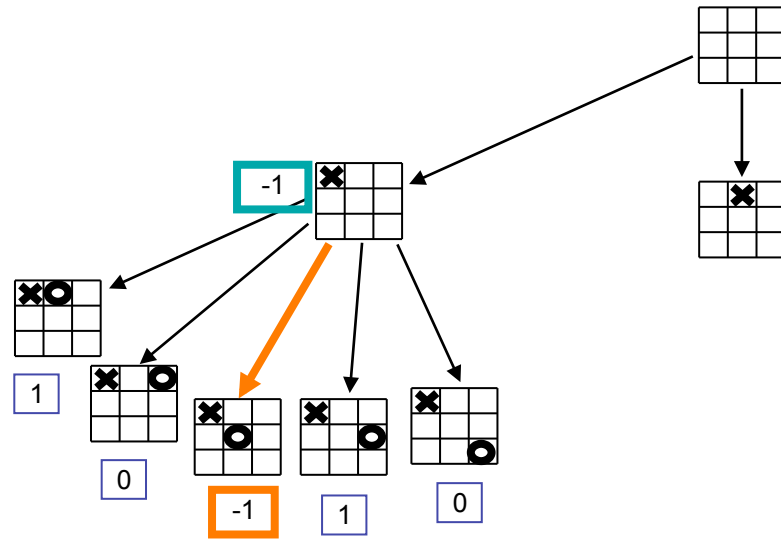
```
function worst(player)  
begin  
  if player = MAX then return  $-\infty$ ;  
  else return  $+\infty$ ;  
end
```

```
function back-up-compare(val1, val2, player)  
begin  
  if player = MAX then return max(val1, val2);  
  else return min(val1, val2);  
end
```

The initial call:

min-max(, MAX, 2)

# Evaluation: bottom-up



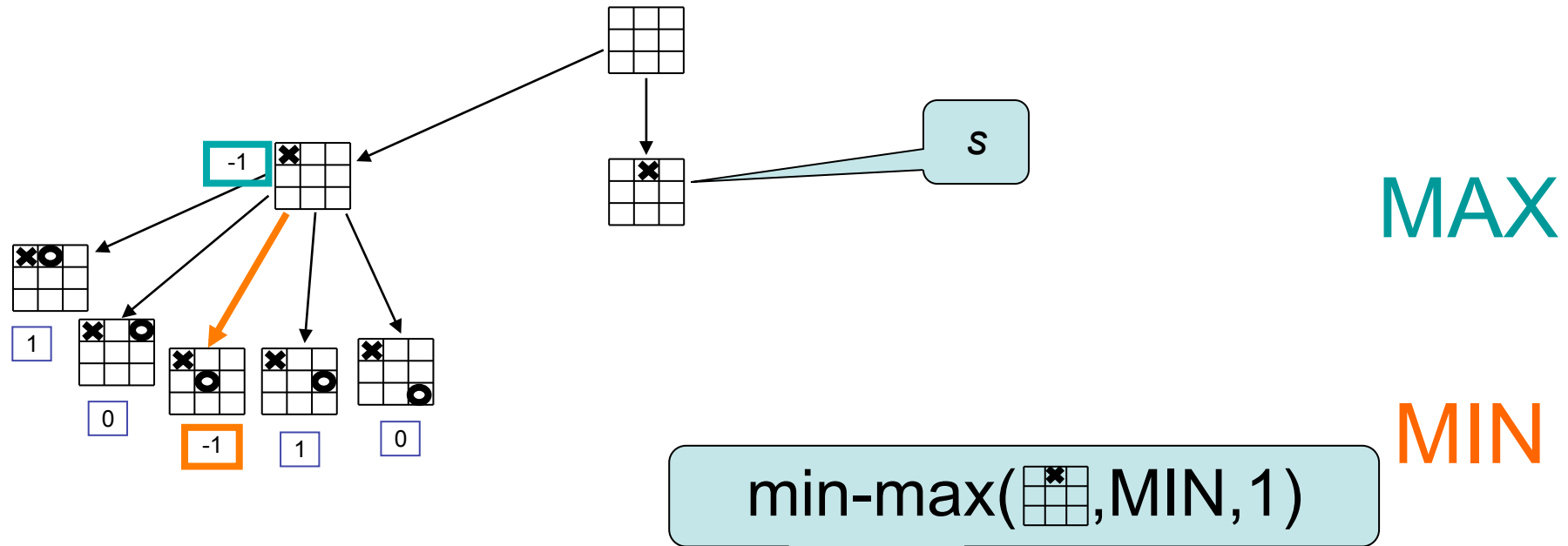
MAX

MIN

```
val=-1; player = MAX; depth=1;
```

```
while (there are still states to generate) begin  
  generate a state -> s;  
  ...  
end
```

# Evaluation: bottom-up

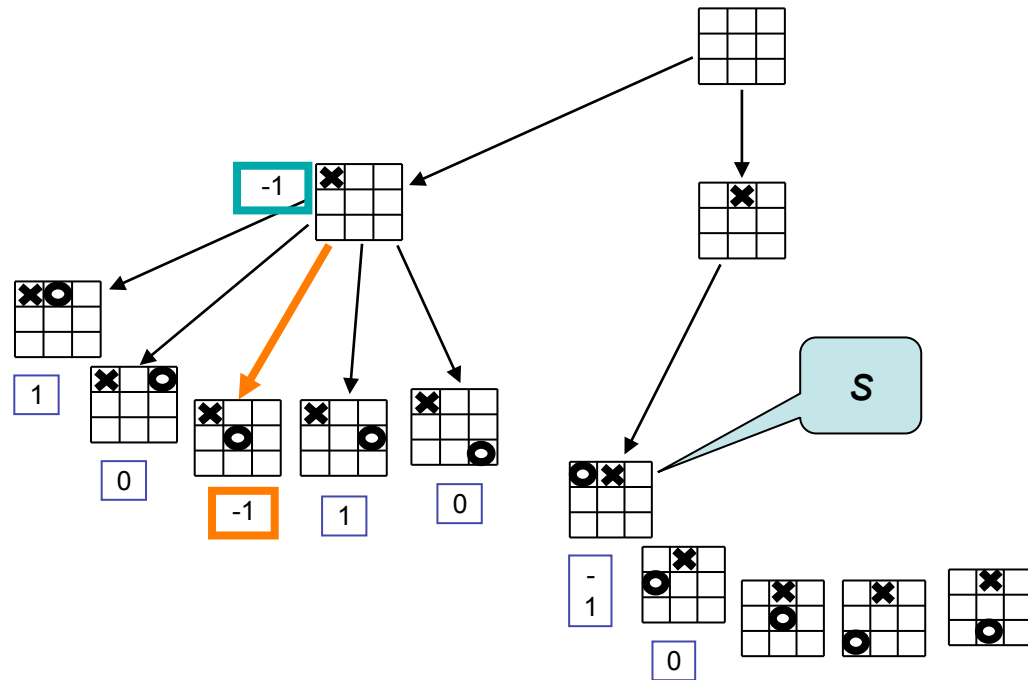


`val=-1; player = MAX; depth=2;`

```

while (there are still states to generate) begin
  generate a state -> s;
  val <- back-up-compare(val, min-max(s, not(player), depth-1), player);
  if (val = -worst(player)) return(val);
end
  
```

# Evaluation: bottom-up



MAX

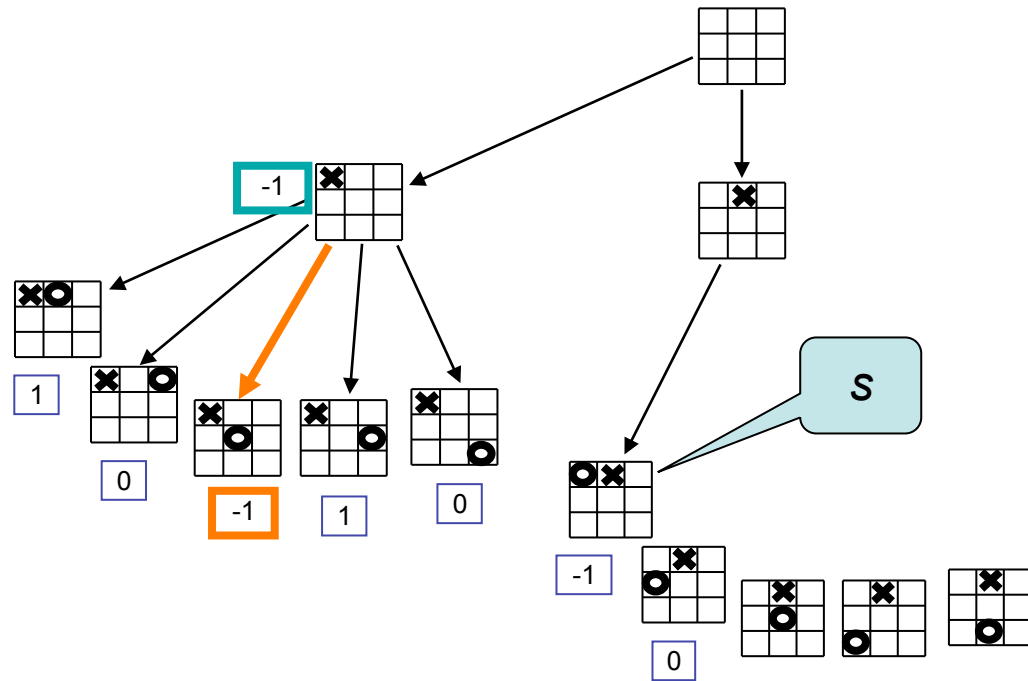
MIN

$\text{min-max}(\begin{array}{|c|c|c|} \hline & & \\ \hline & & \\ \hline & & \\ \hline \end{array}, \text{MIN}, 1)$

```

val = worst(player);  $\longrightarrow$  val= $\infty$ 
while (there are still states to generate) begin
  generate a state  $\rightarrow$  s;
  val <- back-up-compare(val, min-max(s, not(player), depth-1), player);
  if (val = -worst(player)) return(val);
end
  
```

# Evaluation: bottom-up



MAX

MIN

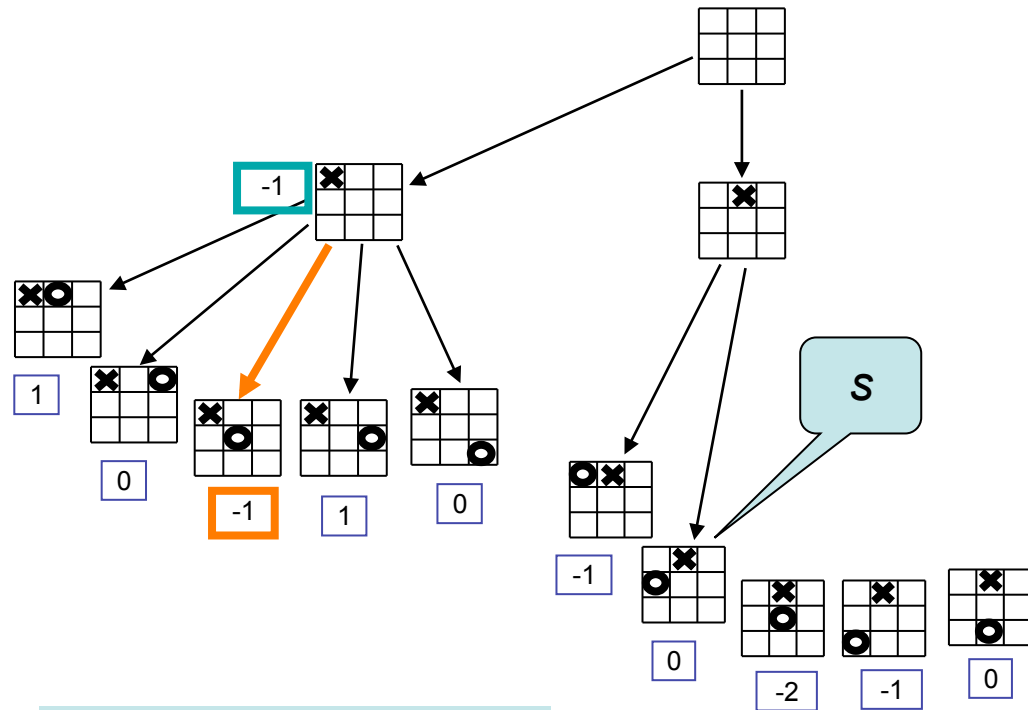
$\text{min-max}(\begin{array}{|c|c|c|} \hline \text{O} & \text{X} & \\ \hline & & \\ \hline & & \\ \hline \end{array}, \text{MAX}, 0)$

if ( $\text{depth} = 0$ ) then return  $\text{score}(\text{state})$ ;





# Evaluation: bottom-up



MAX

MIN

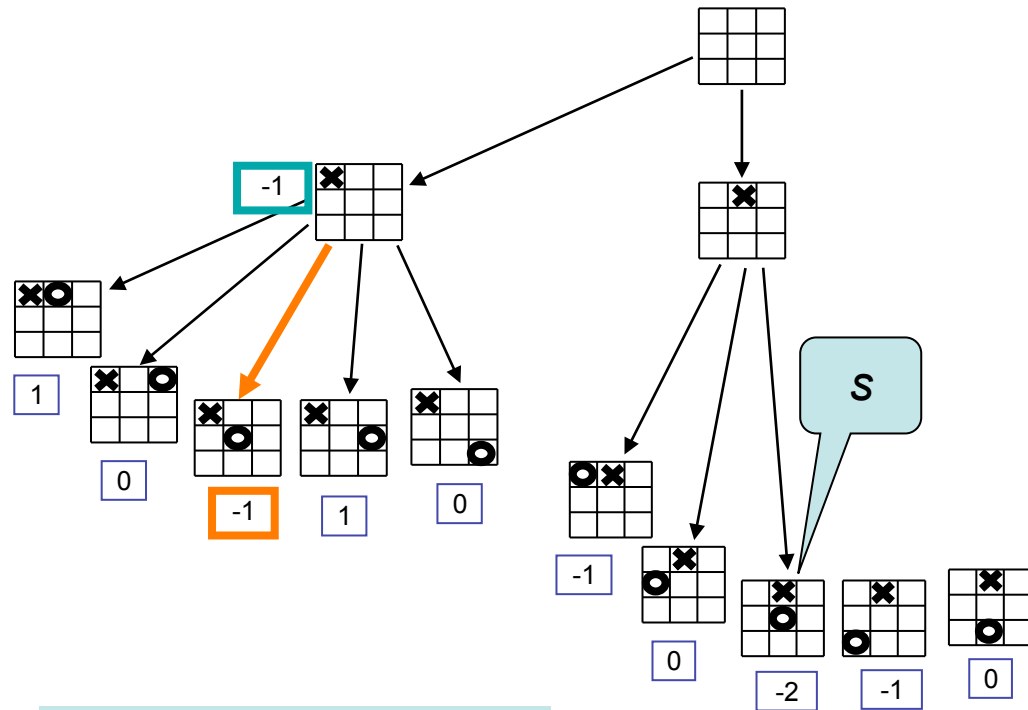
`min-max( , MIN, 1)`

`val = ∞; player = MIN;`

`val <- back-up-compare(val, -1, player)`  
`if (val = -worst(player)) return(val);`  
`end`

-1

# Evaluation: bottom-up



MAX

MIN

min-max( , MIN, 1)

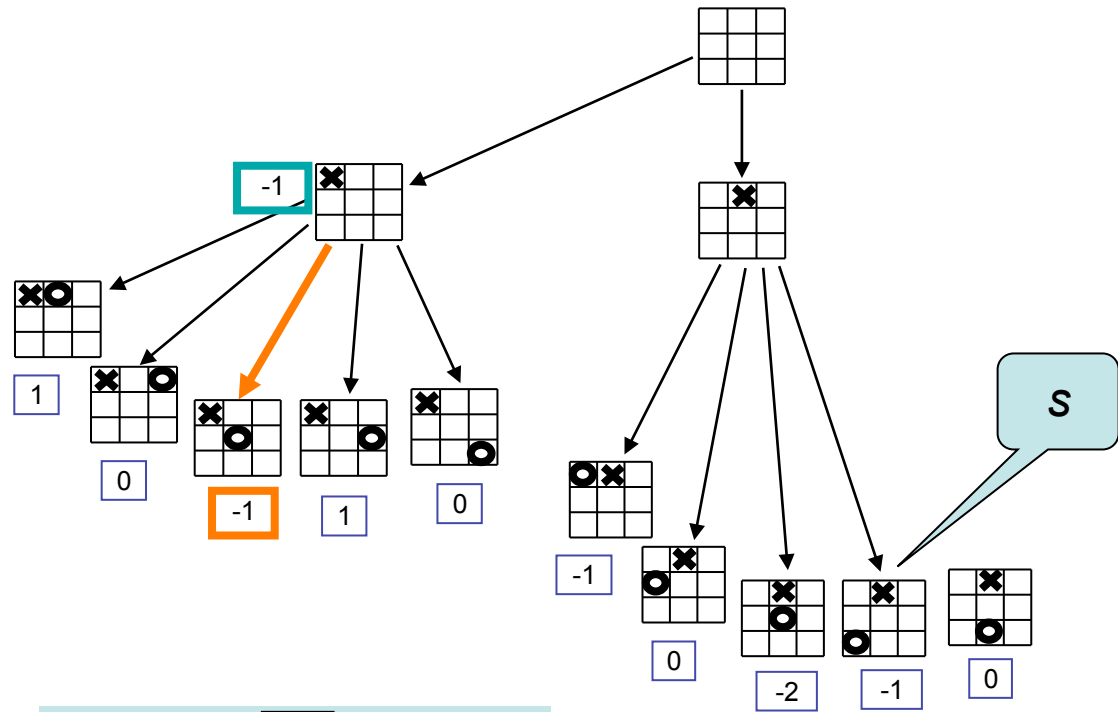
val = ∞; player = MIN;

```

val <- back-up-compare(val, -1, player)
if (val = -worst(player)) return(val);
end
    
```

~~-1~~ -2

# Evaluation: bottom-up



MAX

MIN

min-max( , MIN, 1)

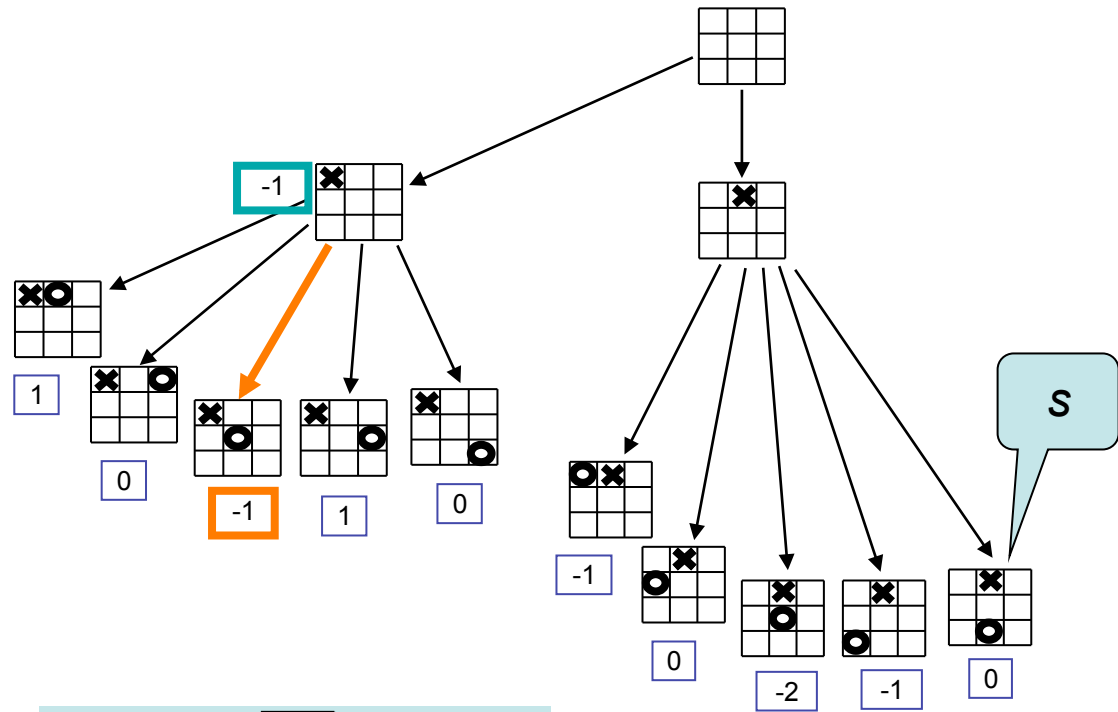
val = ∞; player = MIN;

```

val <- back-up-compare(val, -1, player)
if (val = -worst(player)) return(val);
end
    
```

~~-1~~ -2

# Evaluation: bottom-up



MAX

MIN

min-max( , MIN, 1)

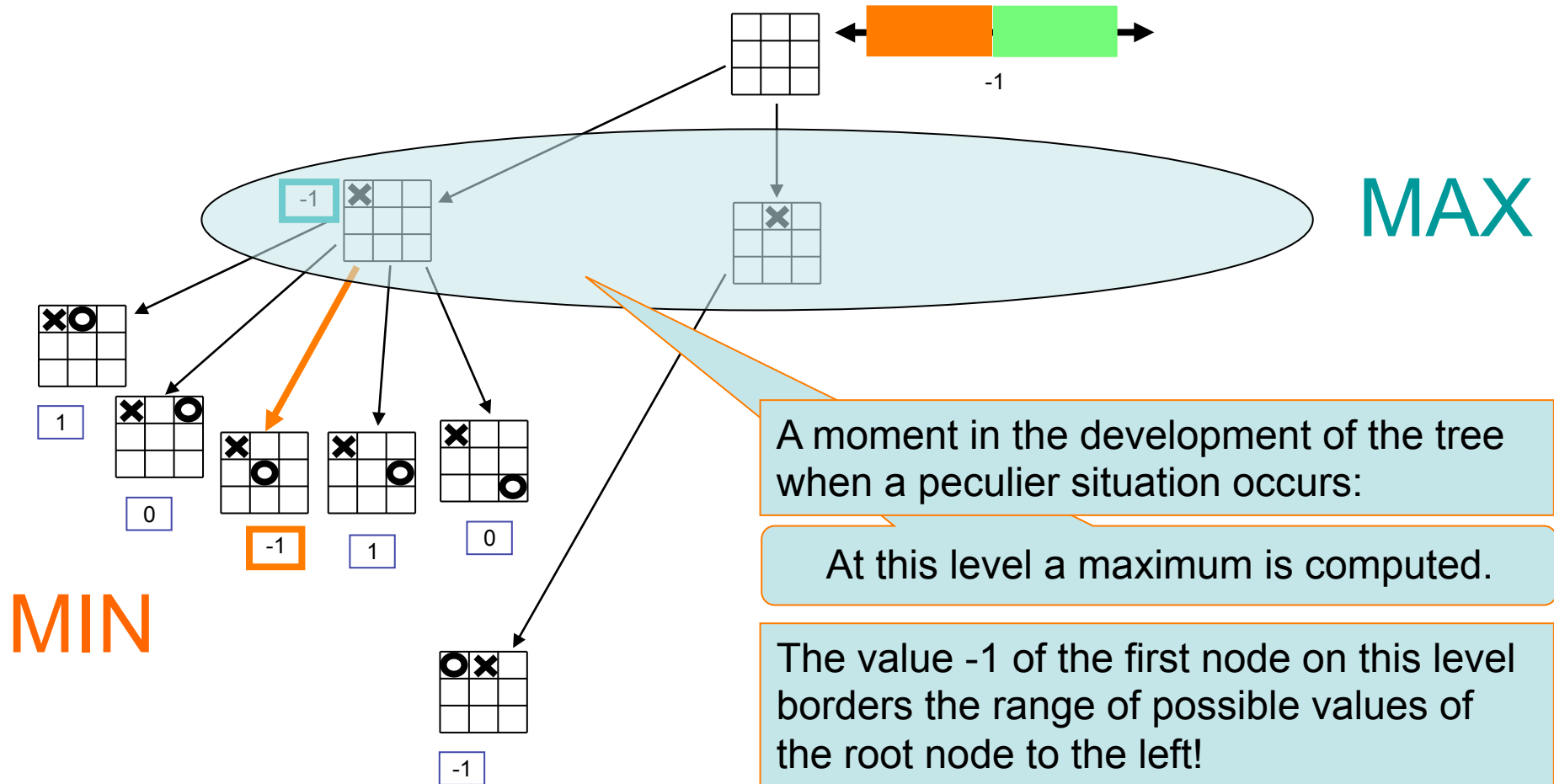
val = ∞; player = MIN;

```

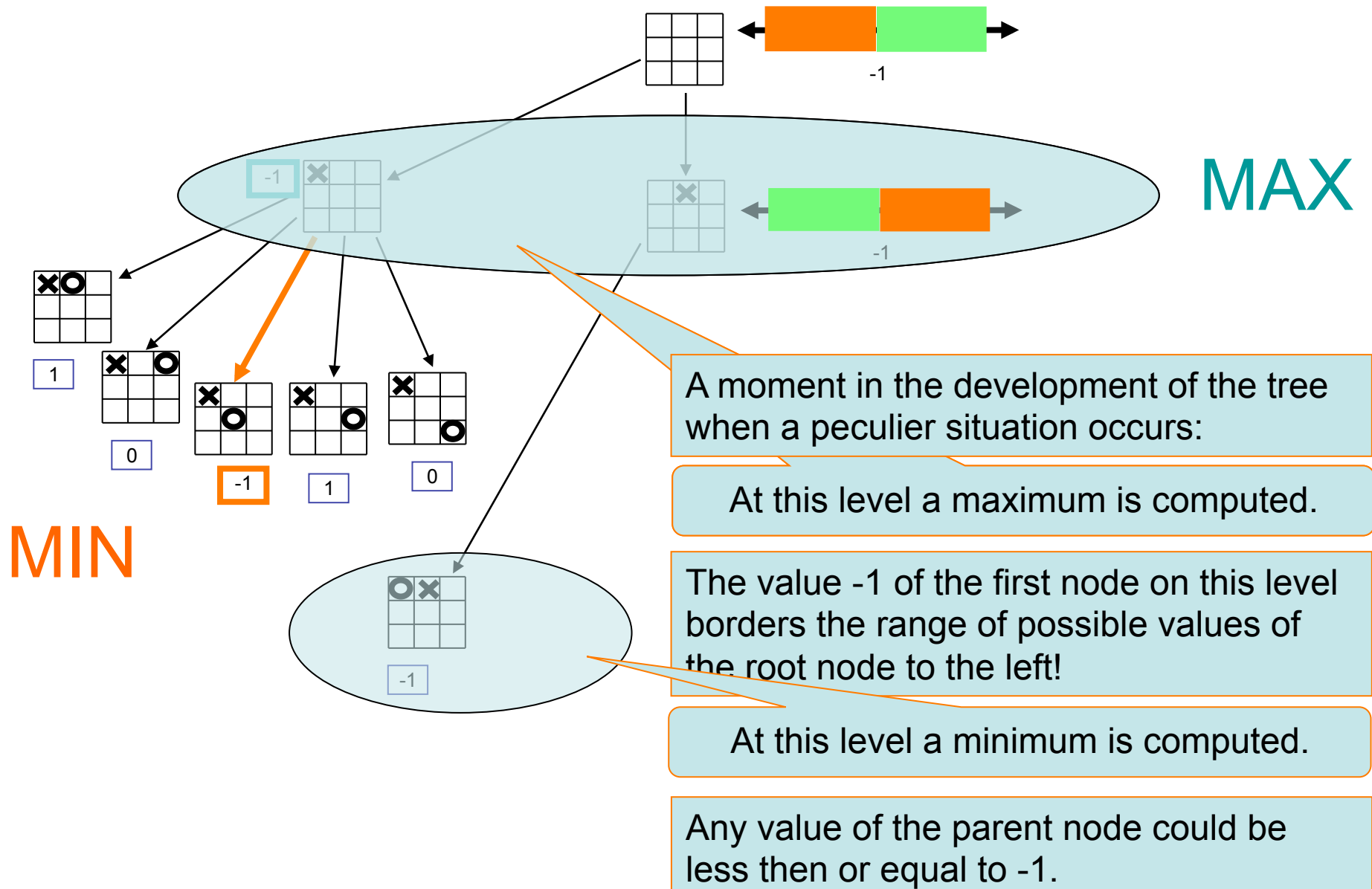
val <- back-up-compare(val, -1, player)
if (val = -worst(player)) return(val);
end
    
```

~~-1~~ -2

# The Alpha-Beta method

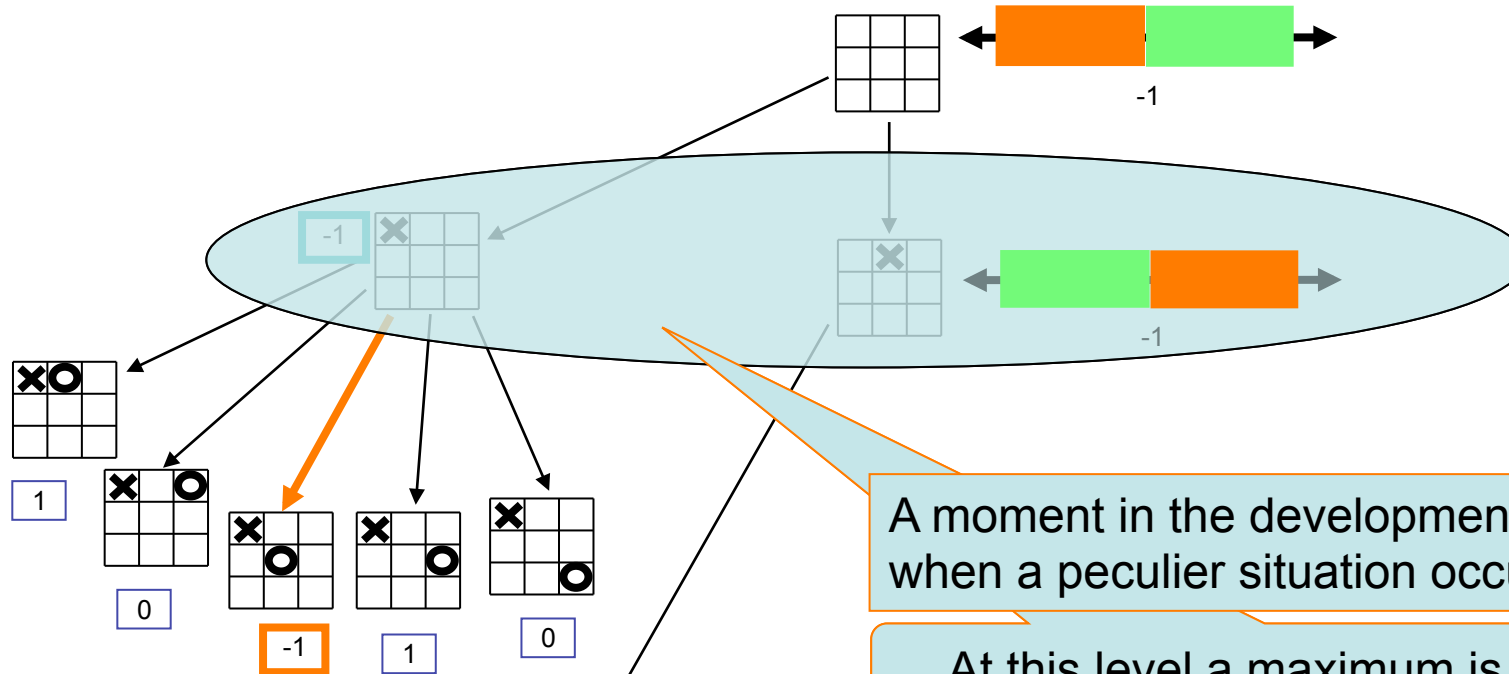


# The Alpha-Beta method



# The Alpha-Beta method

MAX



MIN

The generation can now be stopped!

It can not influence any longer the value of the root node.

A moment in the development of the tree when a peculiar situation occurs:

At this level a maximum is computed.

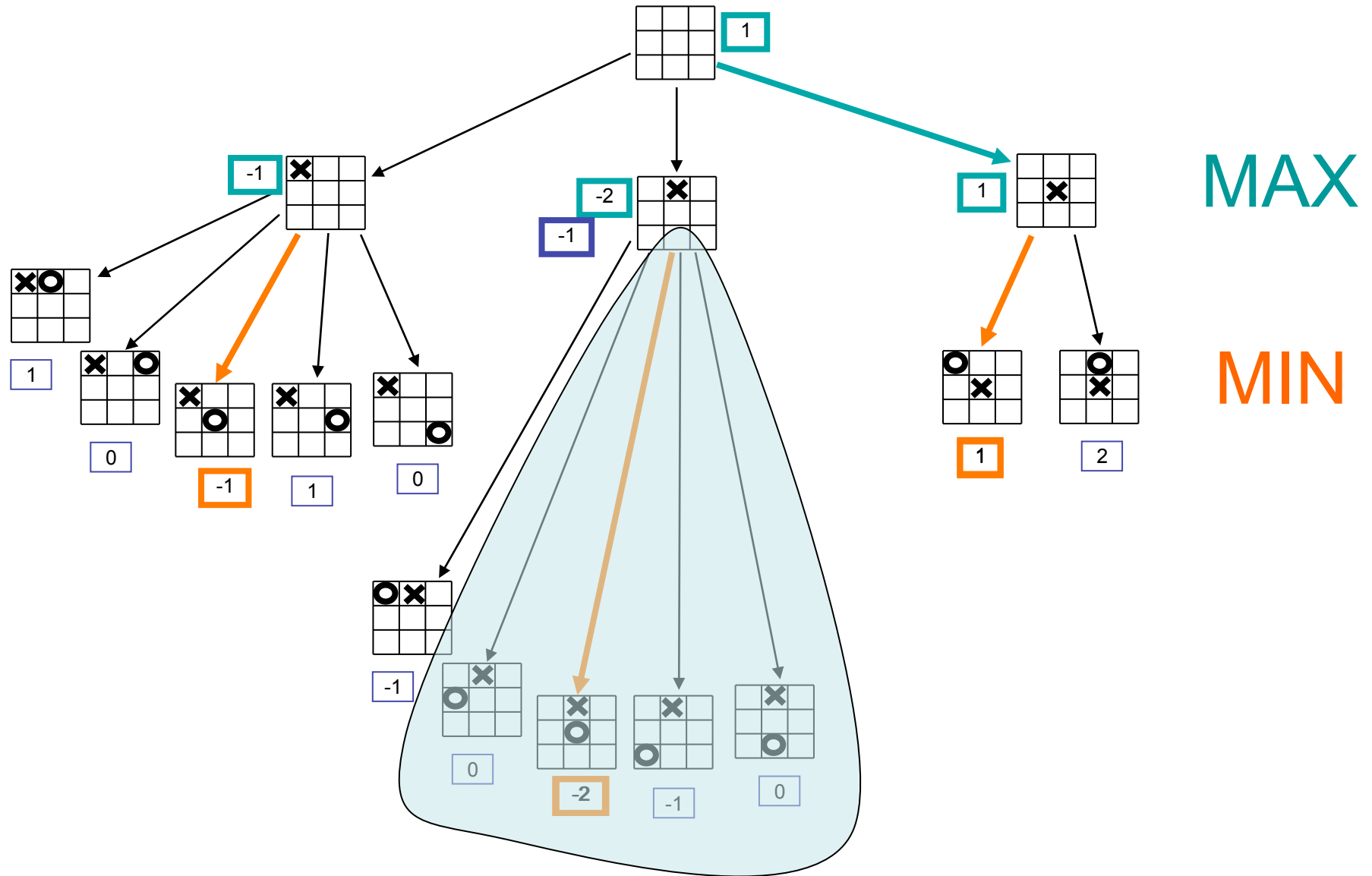
The value -1 of the first node on this level borders the range of possible values of the root node to the left!

At this level a minimum is computed.

Any value of the parent node could be less then or equal to -1.



# The Alpha-Beta method





# The Alpha-Beta method

```
function alpha-beta(state, player, depth)
```

```
begin
```

```
  if (depth = 0) then return score(state);
```

```
  val = worst(player);
```

```
  while (there are still states to generate) begin
```

```
    generate a state -> s;
```

```
    newval <- alpha-beta(s, not(player), depth-1);
```

```
    if player=MAX & newval ≤ val then return(newval);
```

```
    else if player=MIN & newval ≥ val then return(newval);
```

```
    else val ← back-up-compare(val, min-max(s, not(player), depth-1), player);
```

```
    // the following instruction reduces the search space in case a win is reached in one of the generated states:
```

```
    if (val = -worst(player)) return(val);
```

```
  end
```

```
  return(val);
```

```
end
```

```
function worst(player)
```

```
begin
```

```
  if player = MAX then return  $-\infty$ ;
```

```
  else return  $+\infty$ ;
```

```
end
```

```
function back-up-compare(val1, val2, player)
```


```
begin
```

```
  if player = MAX then return max(val1, val2);
```

```
  else return min(val1, val2);
```

```
end
```

The initial call:

alpha-beta (, MAX, 2)