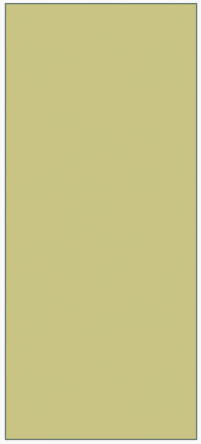
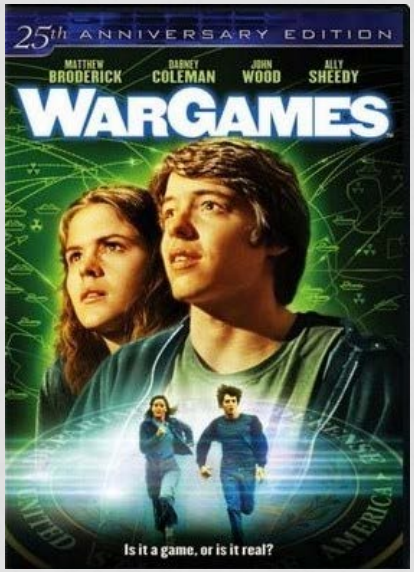
# ADVERSARIAL SEARCH

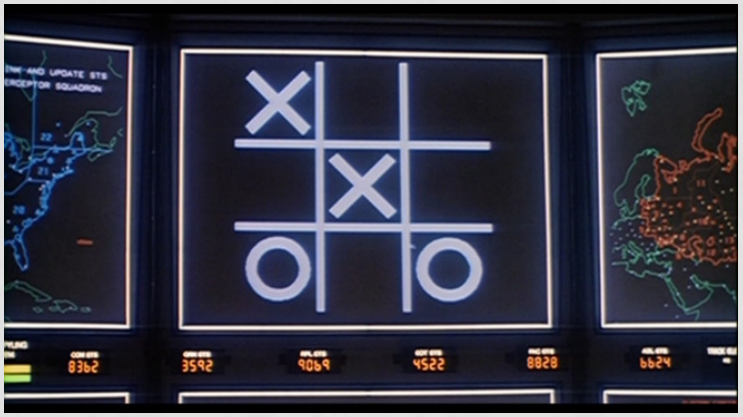
JEFFREY L. POPYACK



# SHALL WE PLAY A GAME?

# SHALL WE PLAY A GAME?

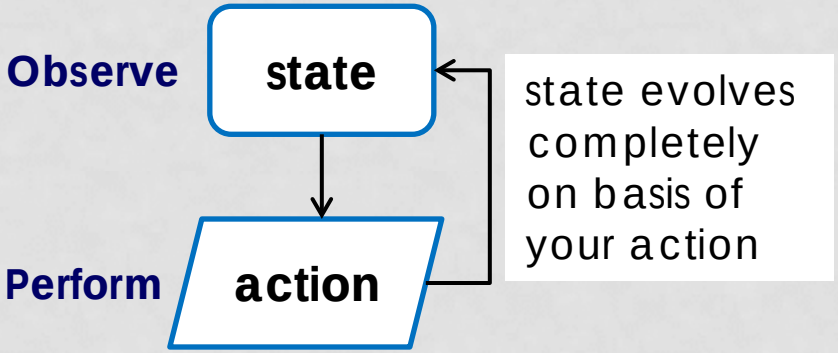





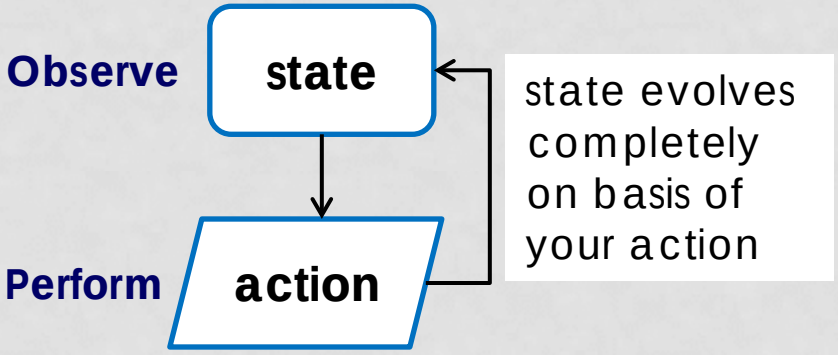

# SHALL WE PLAY A GAME?

**Search Methods So Far:**

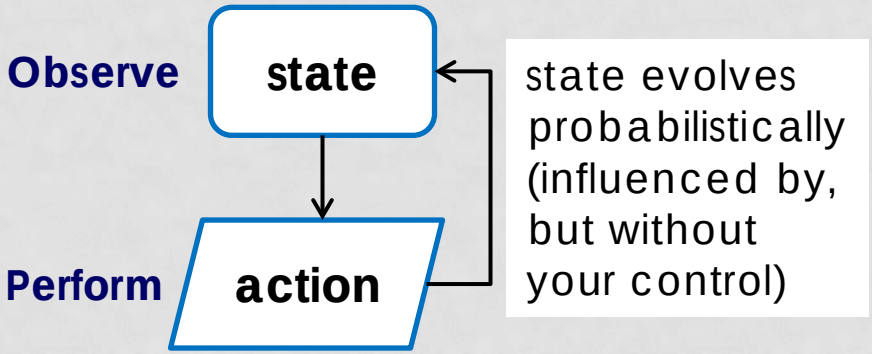


# SHALL WE PLAY A GAME?

**Search Methods So Far:**

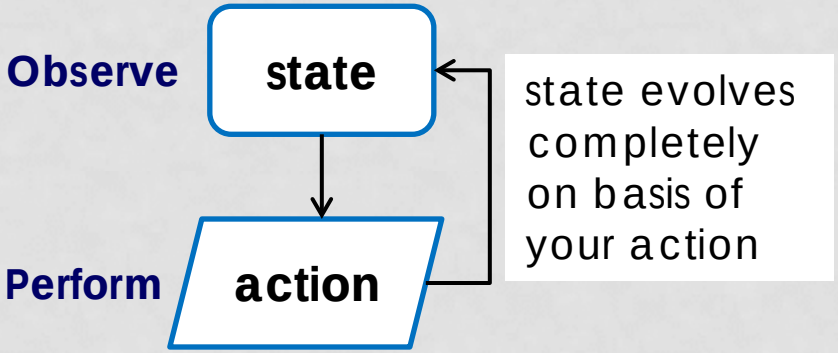



**Stochastic System:**

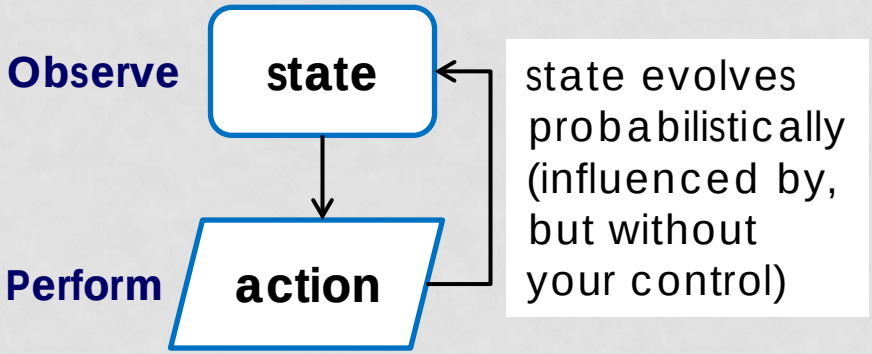


# SHALL WE PLAY A GAME?

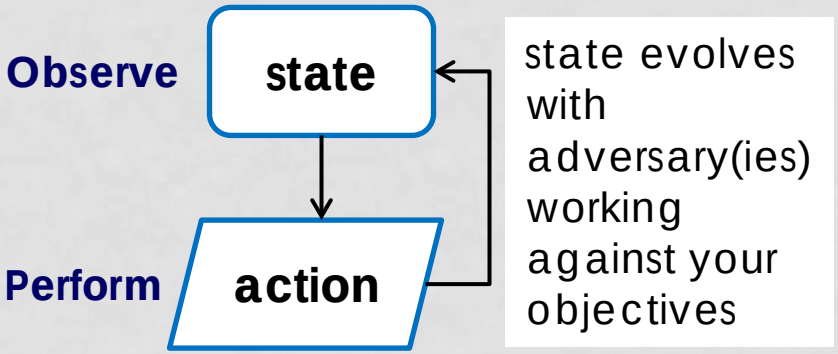
**Search Methods So Far:**




**Stochastic System:**

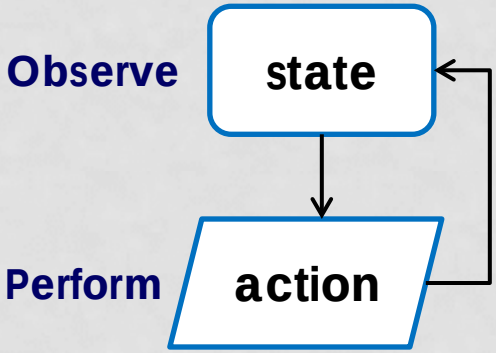



**Adversarial Search:**



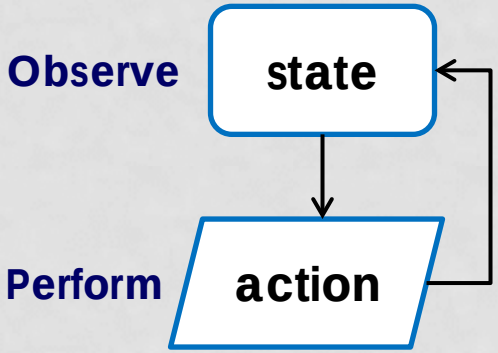
# SHALL WE PLAY A GAME?

## Search Methods So Far:



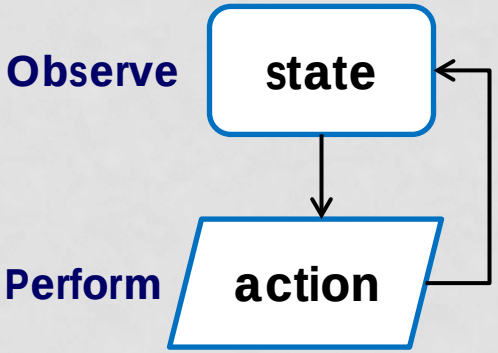

state evolves completely on basis of your action

## Stochastic System:



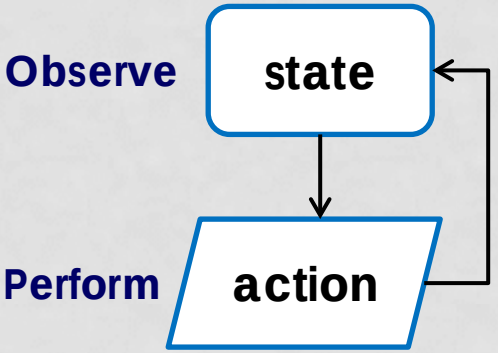

state evolves probabilistically (influenced by, but without your control)

## Adversarial Search:




state evolves with adversary(ies) working against your objectives

## Stochastic Adversarial Search:



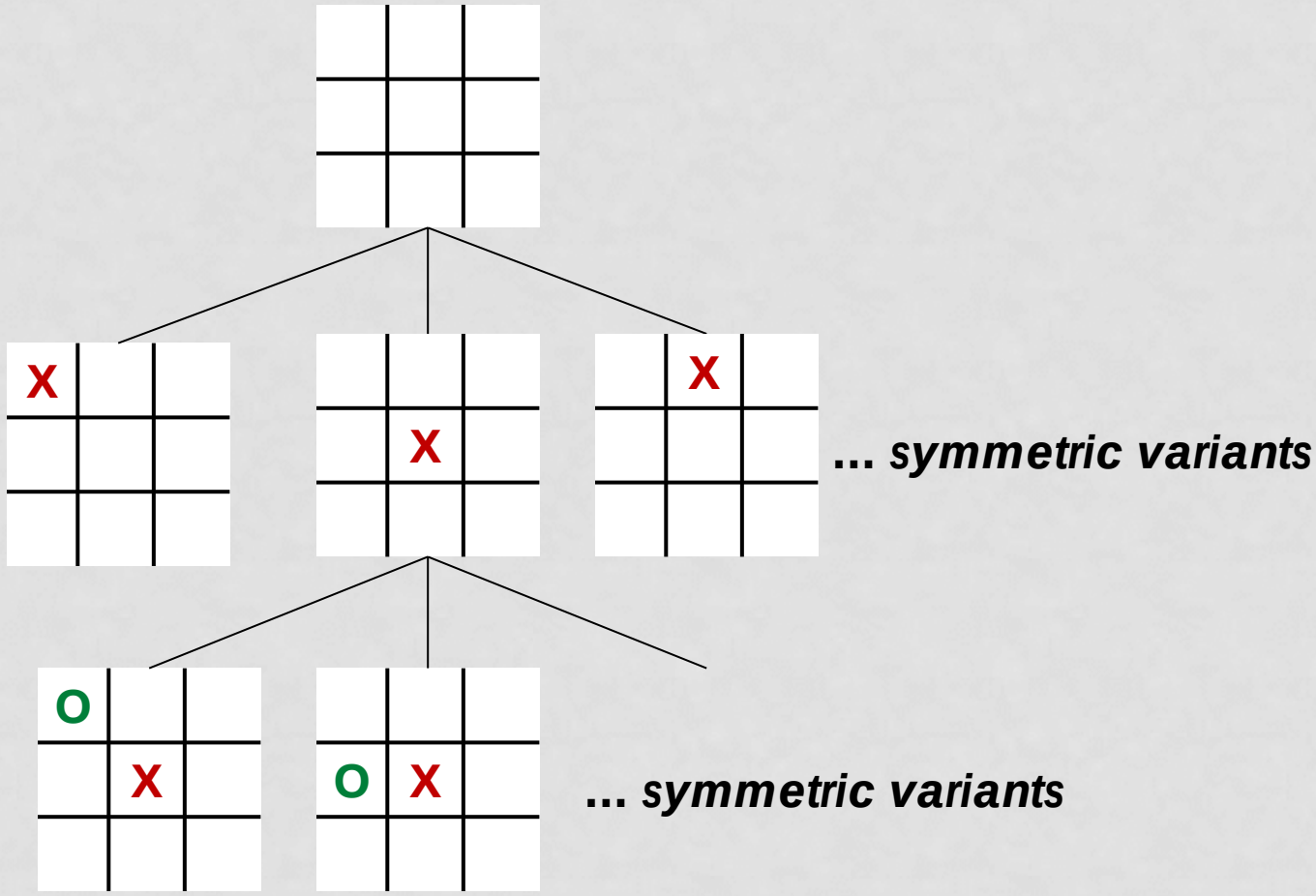

adversary(ies) make moves, **PLUS** rules of chance:

# EXAMPLES

| Deterministic, Single player | Stochastic |
| --- | --- |
| 8-Puzzle<br>Pegboard<br>… | Roulette<br>… |

| Adversarial | Adversial, Stochastic |
| --- | --- |
| Checkers<br>Chess<br>… | Backgammon<br>… |

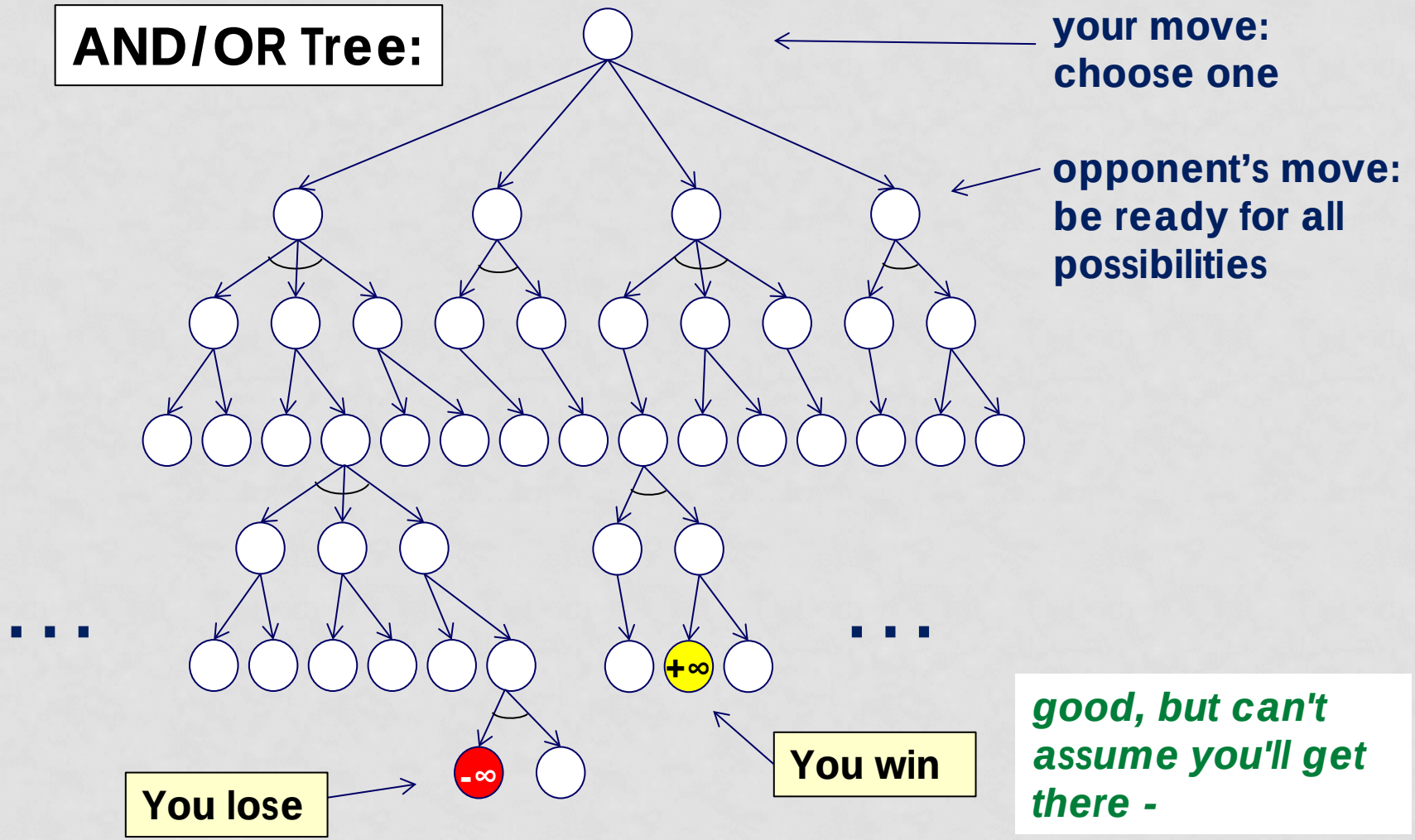
# EXAMPLES: TIC-TAC-TOE




**9! Possible Games (many cannot happen, e.g.,**

| | | |
|---|---|---|
| X | | O |
| X | | O |
| X | | O |

# EXAMPLES: TIC-TAC-TOE

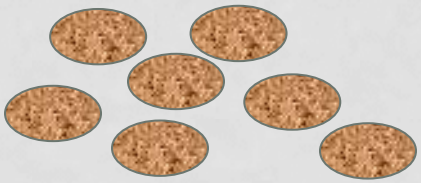


# THE GAME OF NIM

**K** players
**n** stones in a pile
On each turn, player can take
1,2, …, **m** stones

Player who takes last stone loses!

For instance,
**K** = 2
**n** = 7
**m** = 3

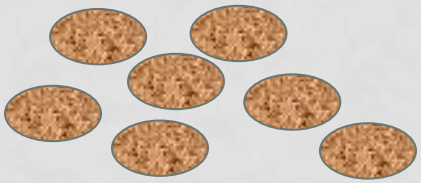


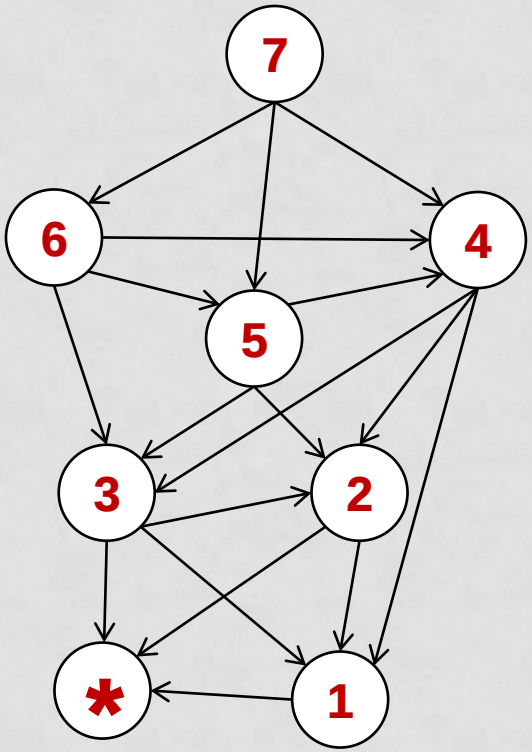
# THE GAME OF NIM

**K** players
**n** stones in a pile
On each turn, player can take
1,2, …, **m** stones

Player who takes last stone loses!

For instance,
**K** = 2
**n** = 7
**m** = 3





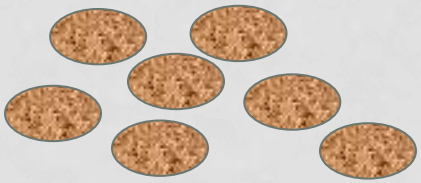

***Intense study reveals*** : person who goes first should choose 2 stones
Suppose we start with **n** = 8?

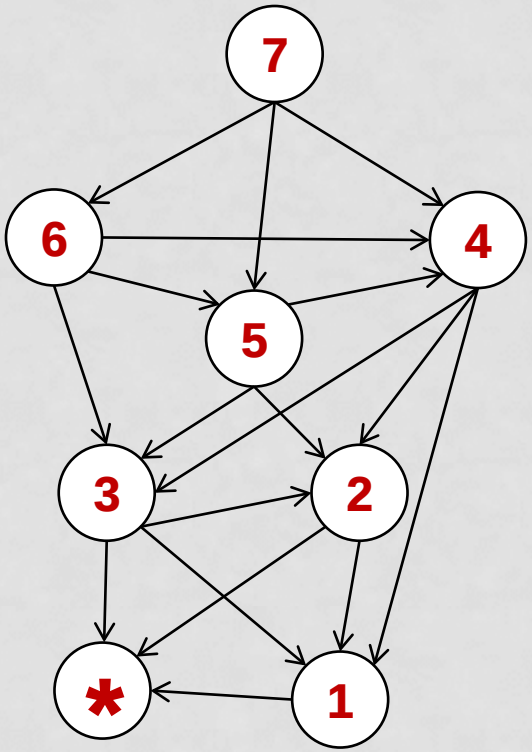
# THE GAME OF NIM

**K** players
**n** stones in a pile
On each turn, player can take
1,2, …, **m** stones

Player who takes last stone loses!

For instance,
**K** = 2
**n** = 7
**m** = 3





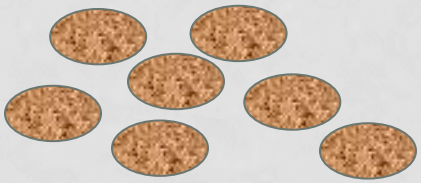

Intense study reveals : person who goes first should choose 2 stones
Suppose we start with **n** = 8? **(choose 3)** with **n** = 9?

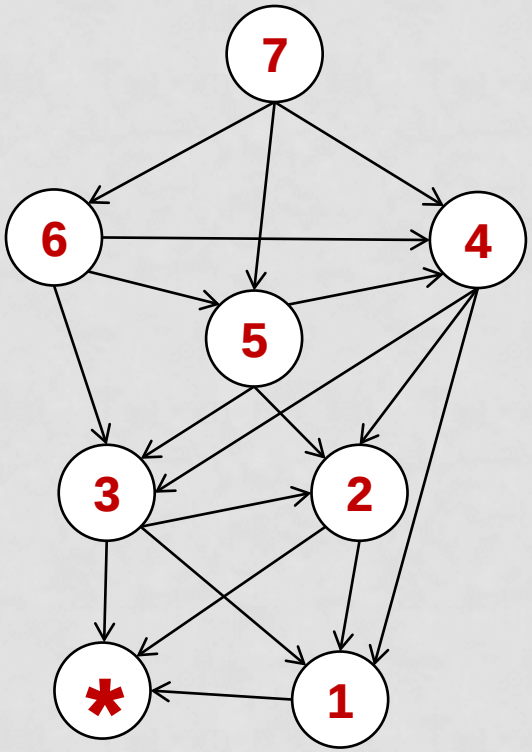
# THE GAME OF NIM

**K** players
**n** stones in a pile
On each turn, player can take
1,2, …, **m** stones

Player who takes last stone loses!

For instance,
**K** = 2
**n** = 7
**m** = 3



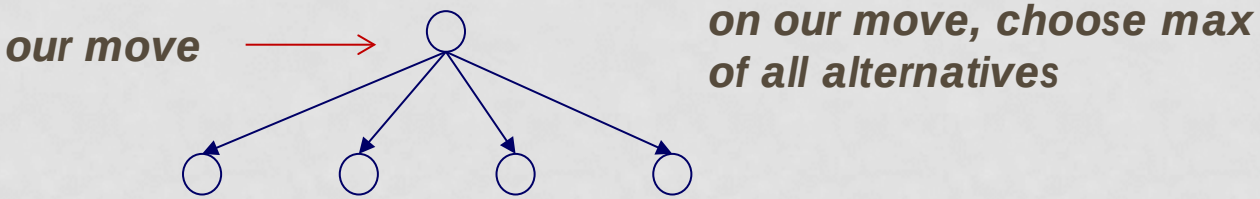



Intense study reveals : person who goes first should choose 2 stones
Suppose we start with **n** = 8? **(choose 3)** with **n** = 9 ? **(no good answer)**
Suppose we start with **n** = 10?

# MINIMAX

- You are trying to **MAXIMIZE** some utility function
- Opponent is trying to **MINIMIZE** the same function

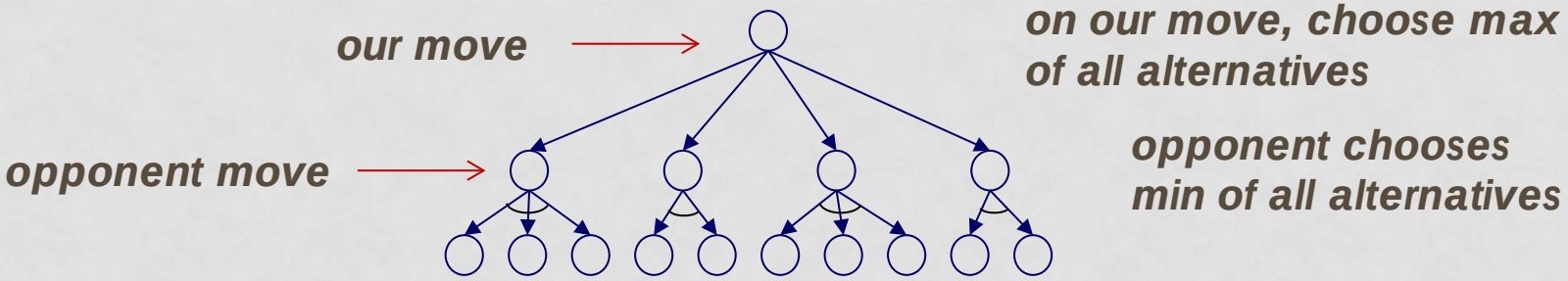
We use a 3-move lookahead:



# MINIMAX

- You are trying to **MAXIMIZE** some utility function
- Opponent is trying to **MINIMIZE** the same function

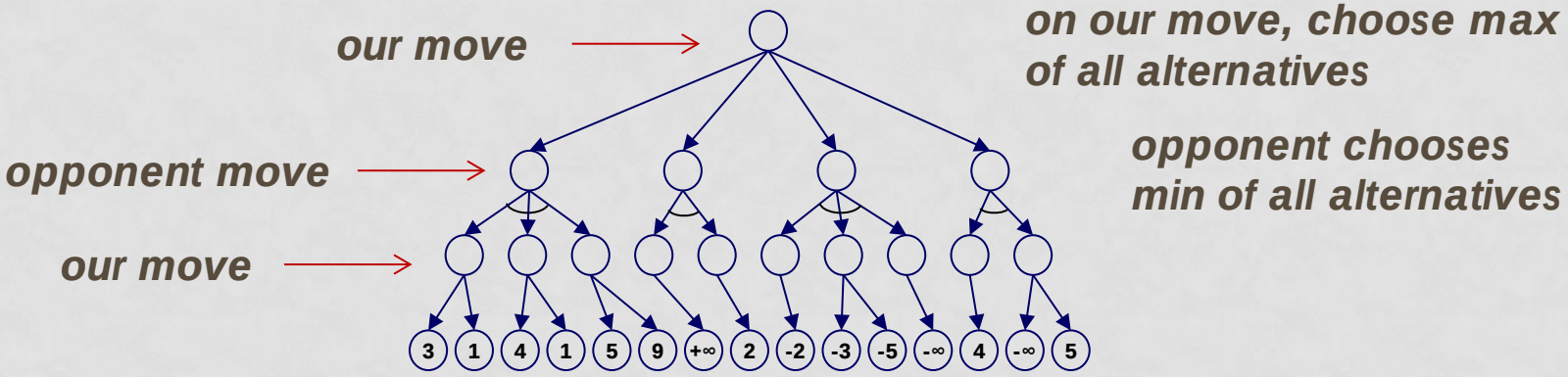
We use a 3-move lookahead:



# MINIMAX

- You are trying to **MAXIMIZE** some utility function
- Opponent is trying to **MINIMIZE** the same function

We use a 3-move lookahead:

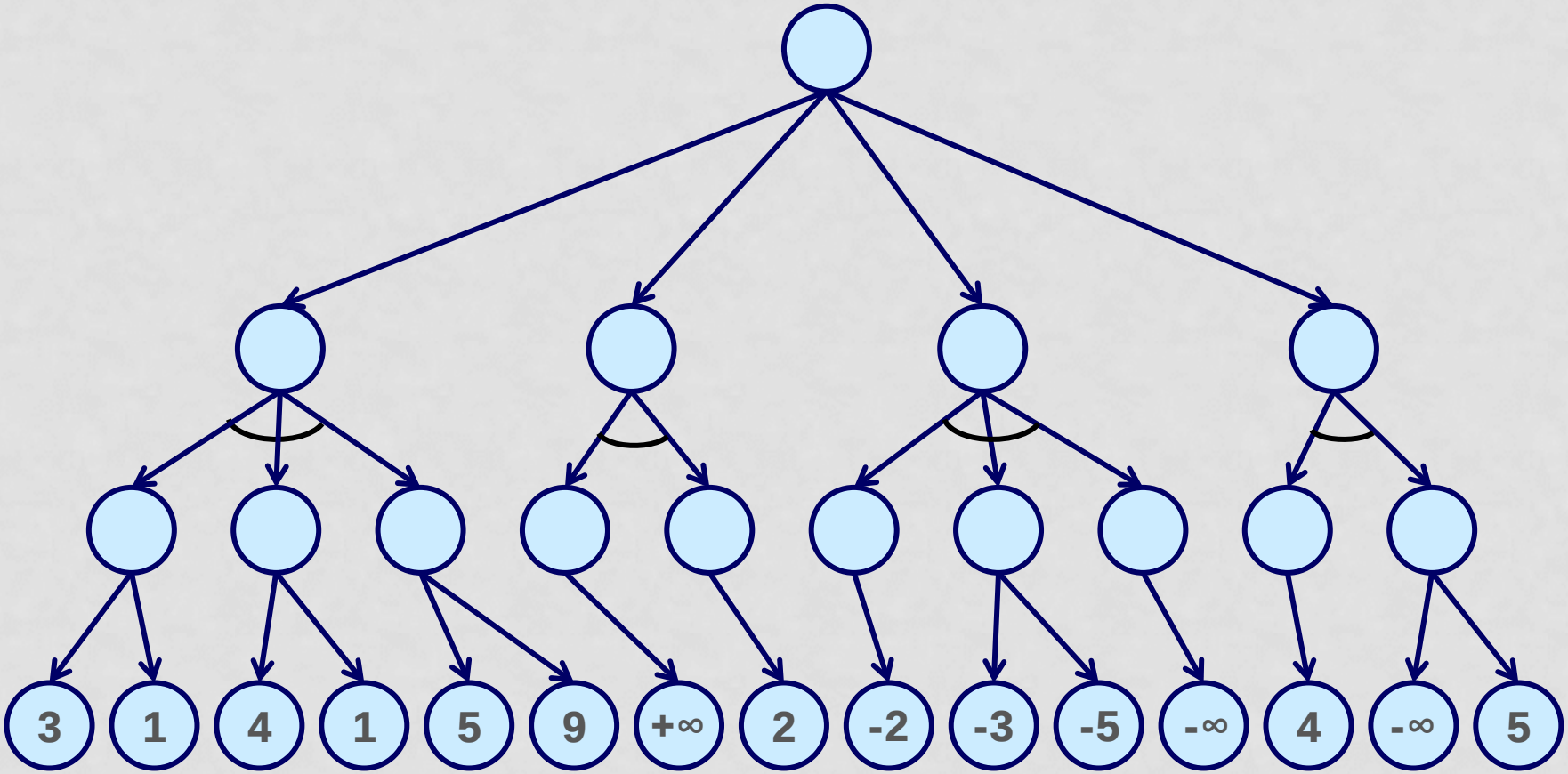


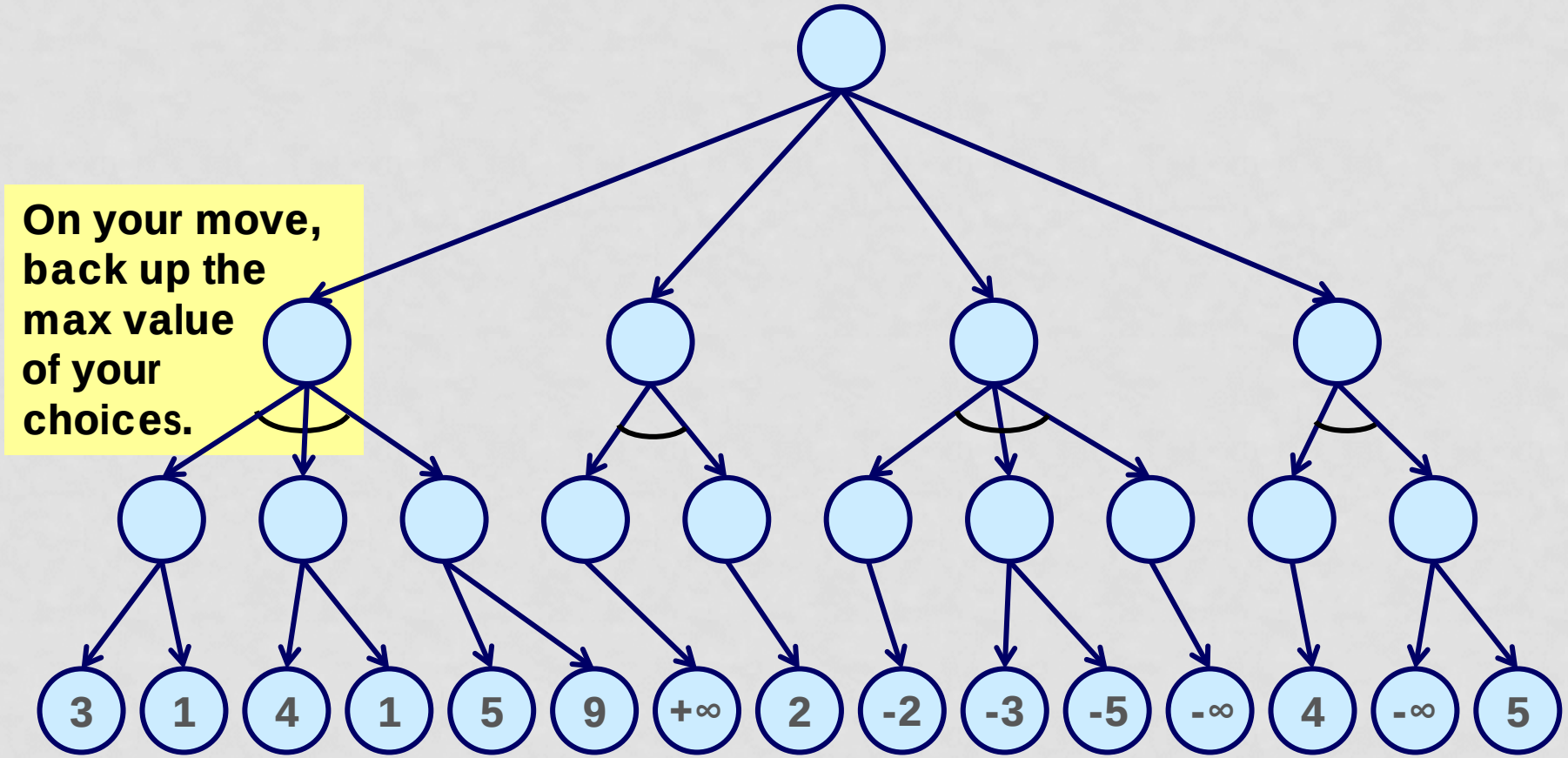

Employ an evaluation function (heuristic) at bottom level, based on state of system, including whose turn it is.

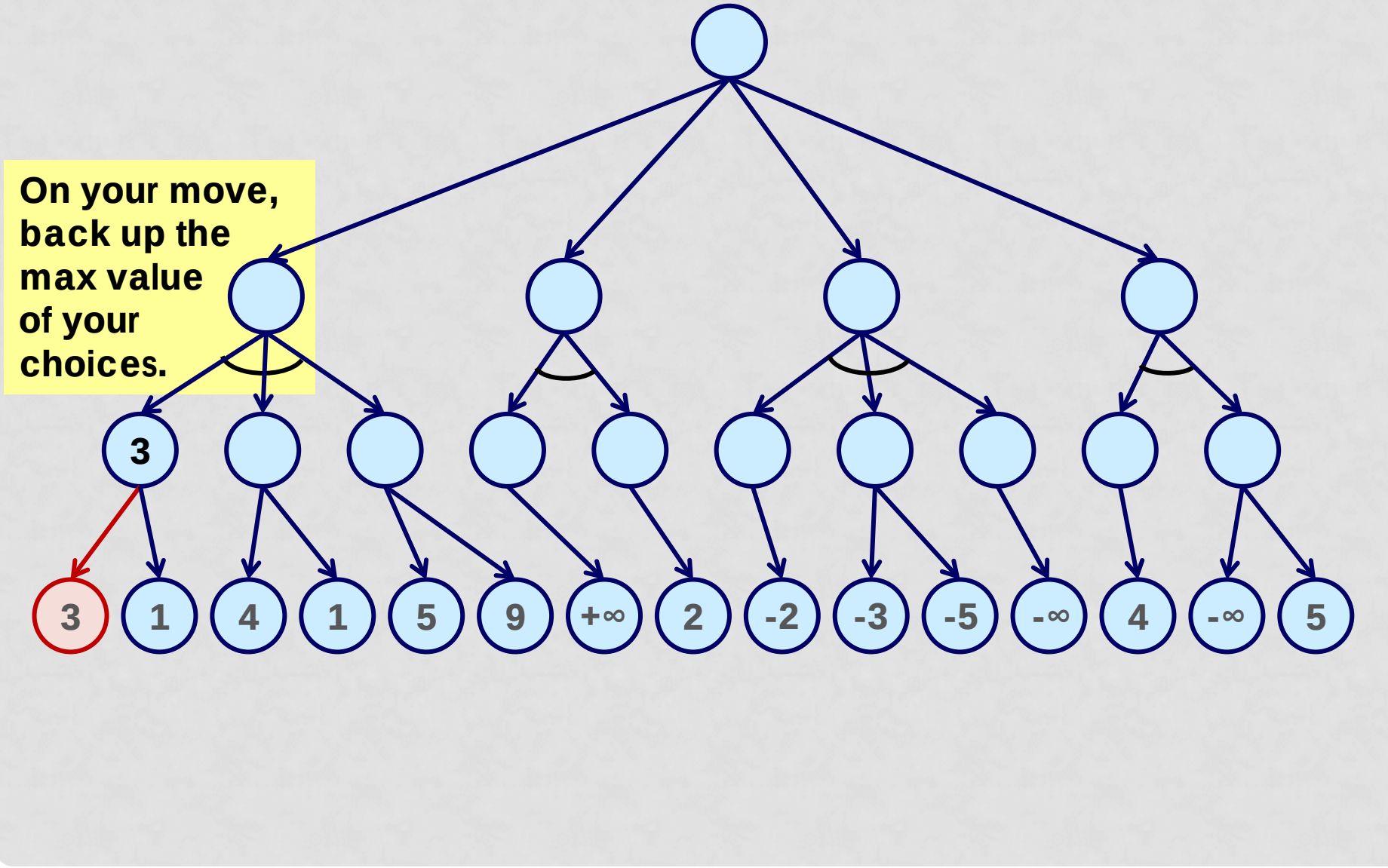
***"Back up" values to root***

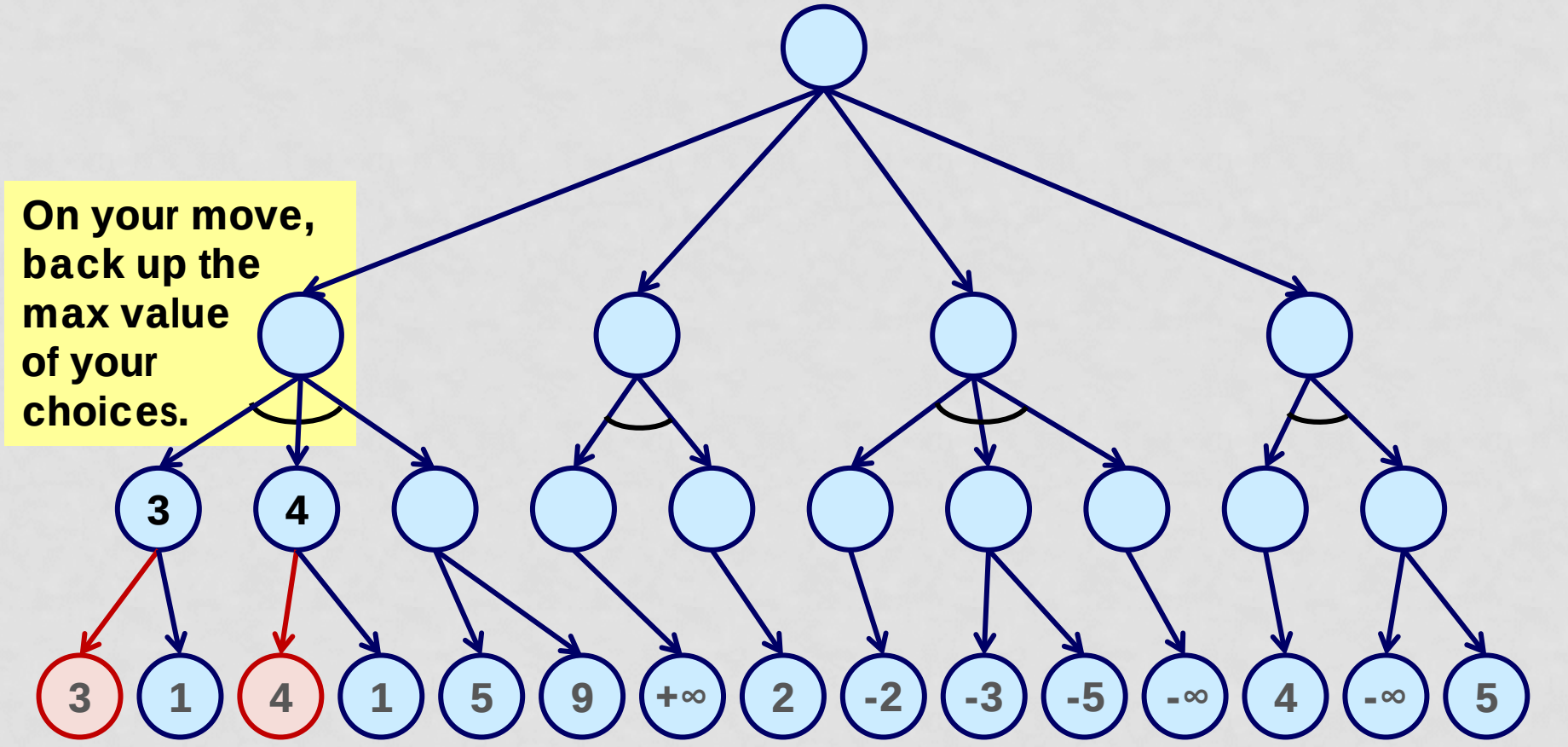
**+∞ = WIN**

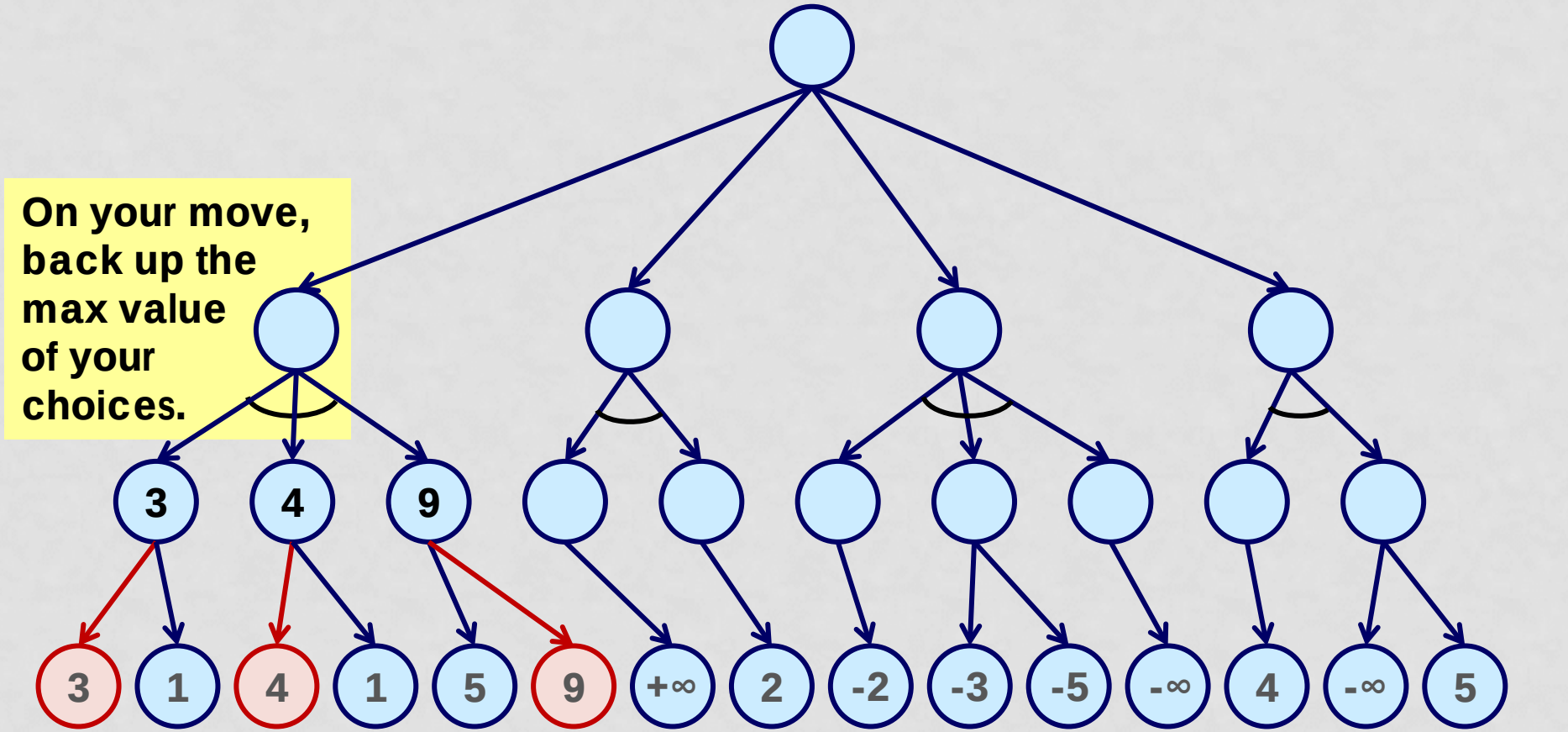
**-∞ = LOSS**

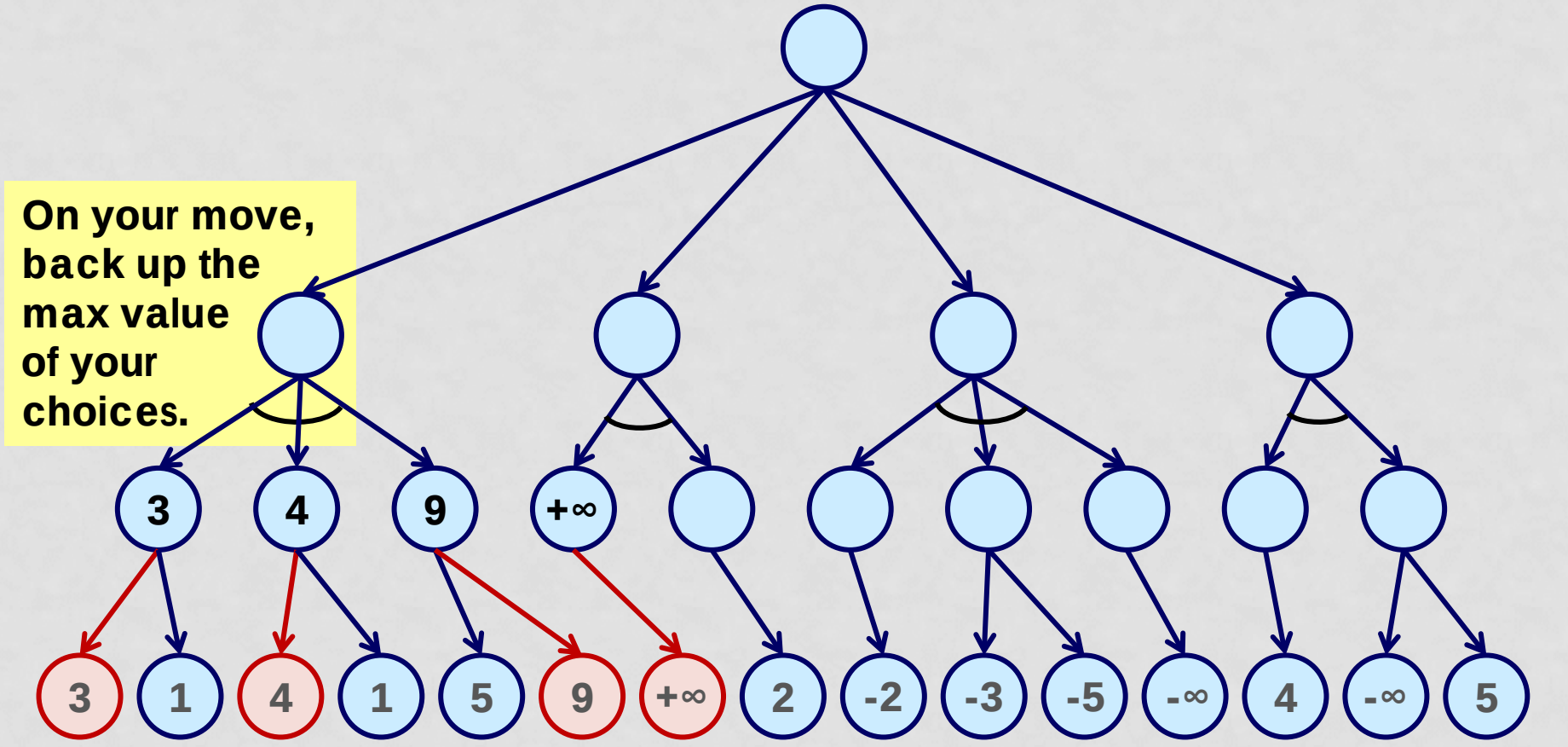

3
1
4
1
5
9
+∞
2
-2
-3
-5
-∞
4
-∞
5

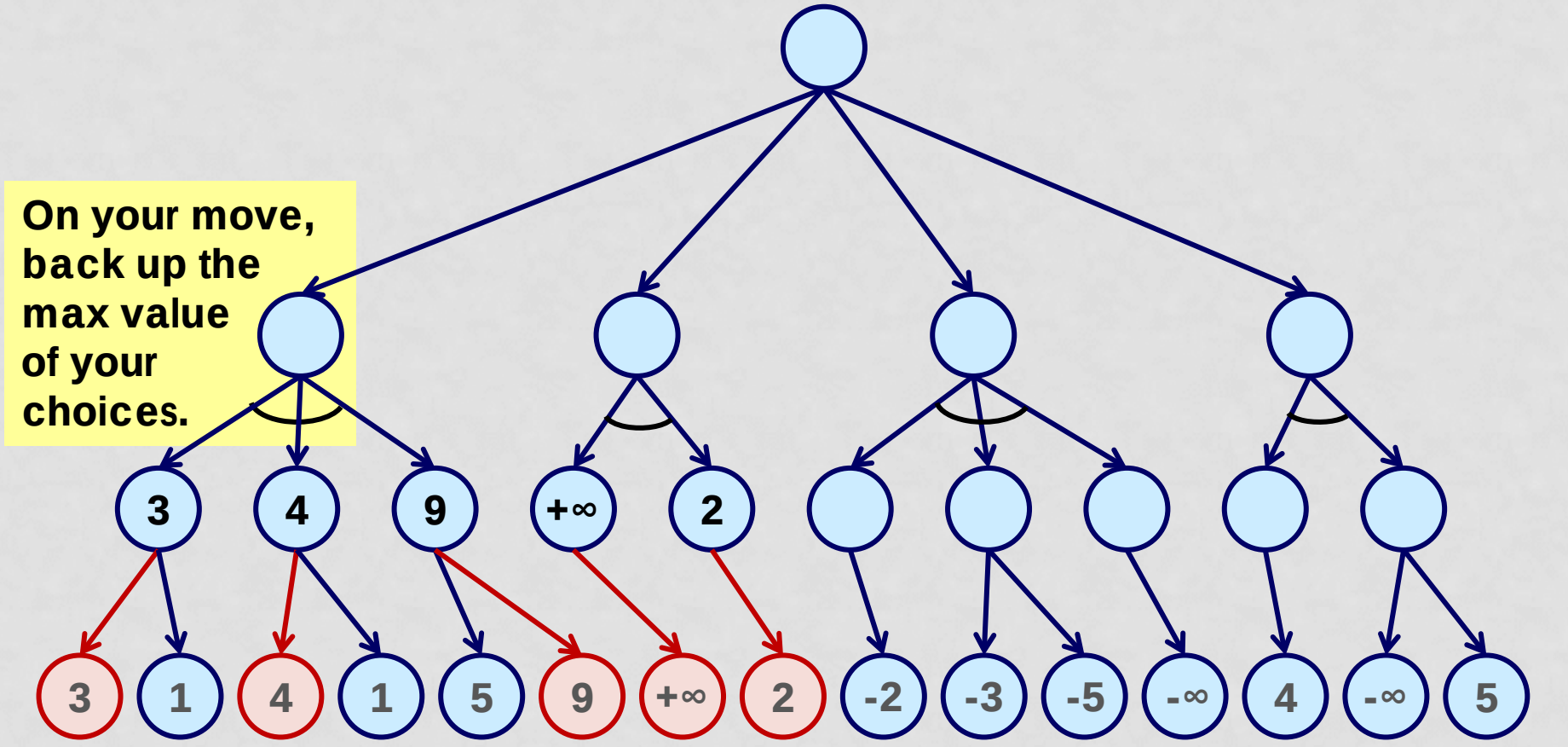

On your move, back up the max value of your choices.
3
1
4
1
5
9
+∞
2
-2
-3
-5
-∞
4
-∞
5

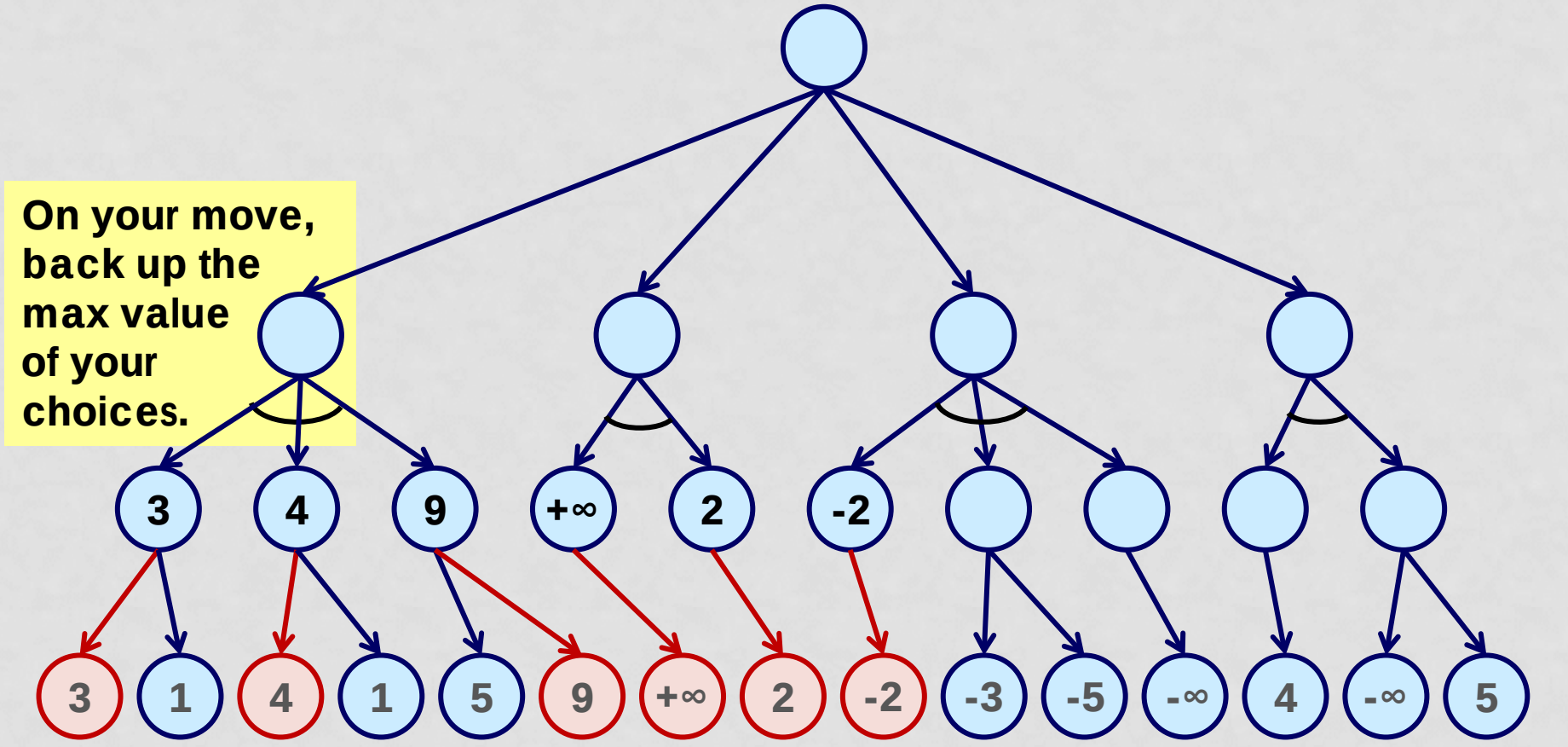

On your move, back up the max value of your choices.
3
3
1
4
1
5
9
+∞
2
-2
-3
-5
-∞
4
-∞
5

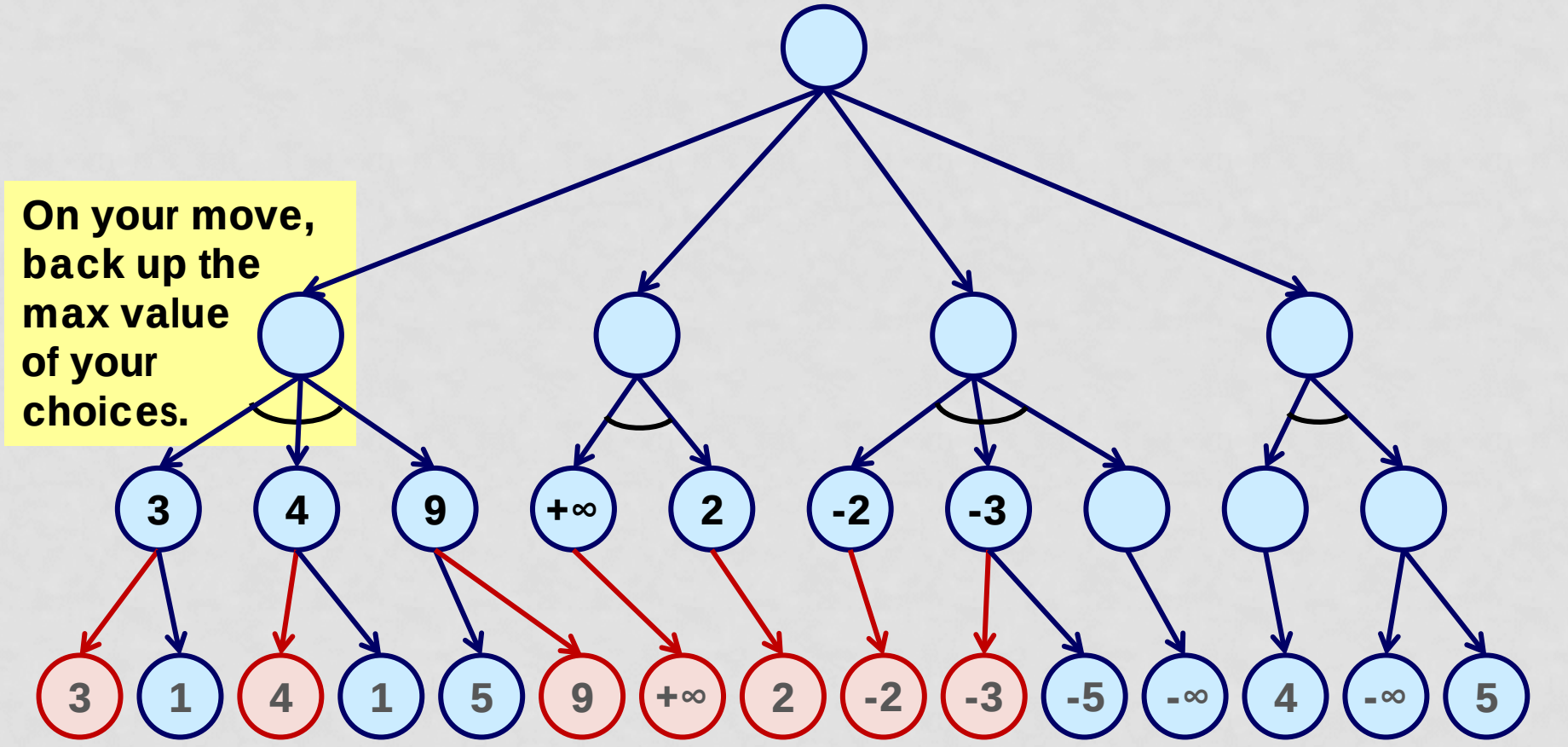

On your move, back up the max value of your choices.
3
4
3
1
4
1
5
9
+∞
2
-2
-3
-5
-∞
4
-∞
5

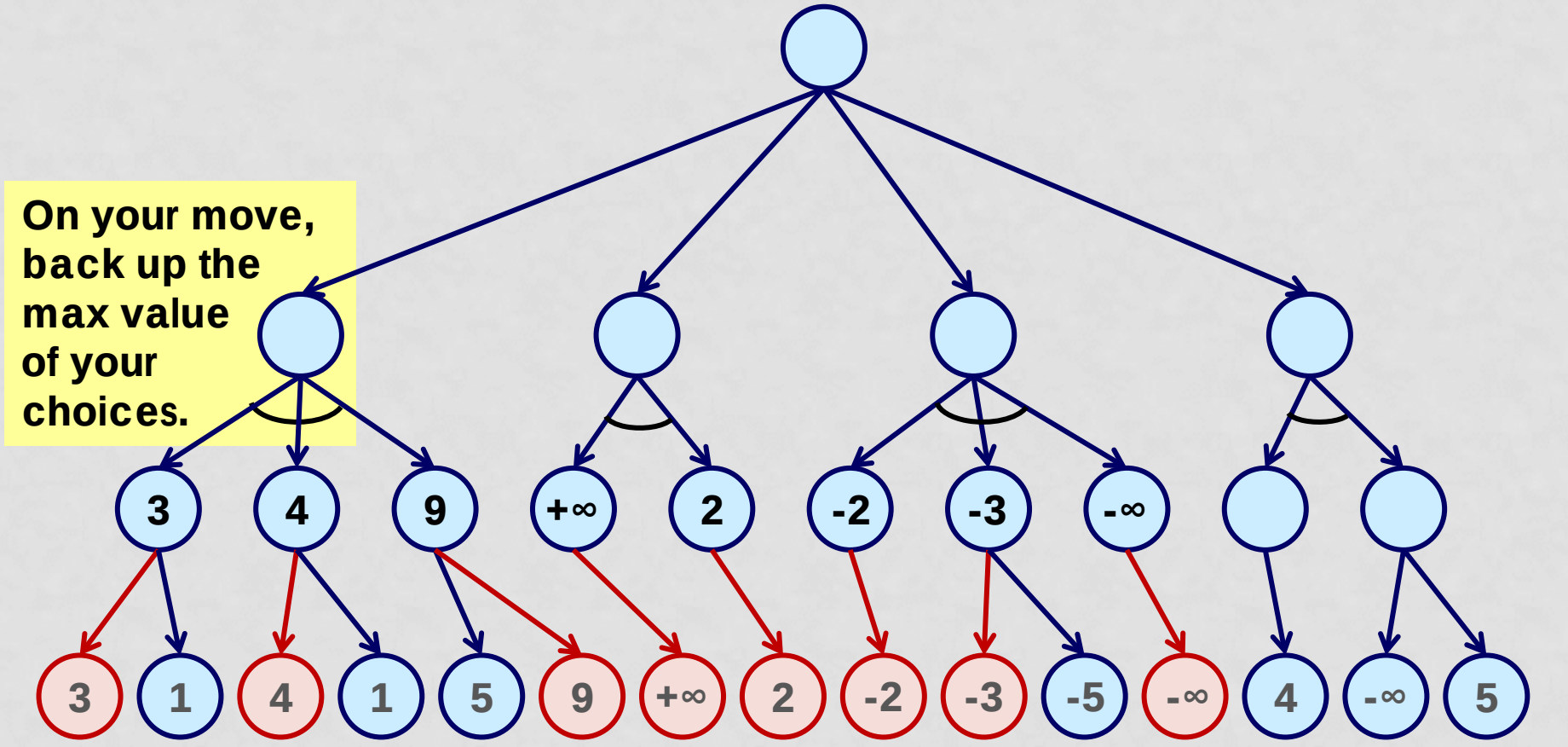

On your move, back up the max value of your choices.
3
4
9
3
1
4
1
5
9
+∞
2
-2
-3
-5
-∞
4
-∞
5

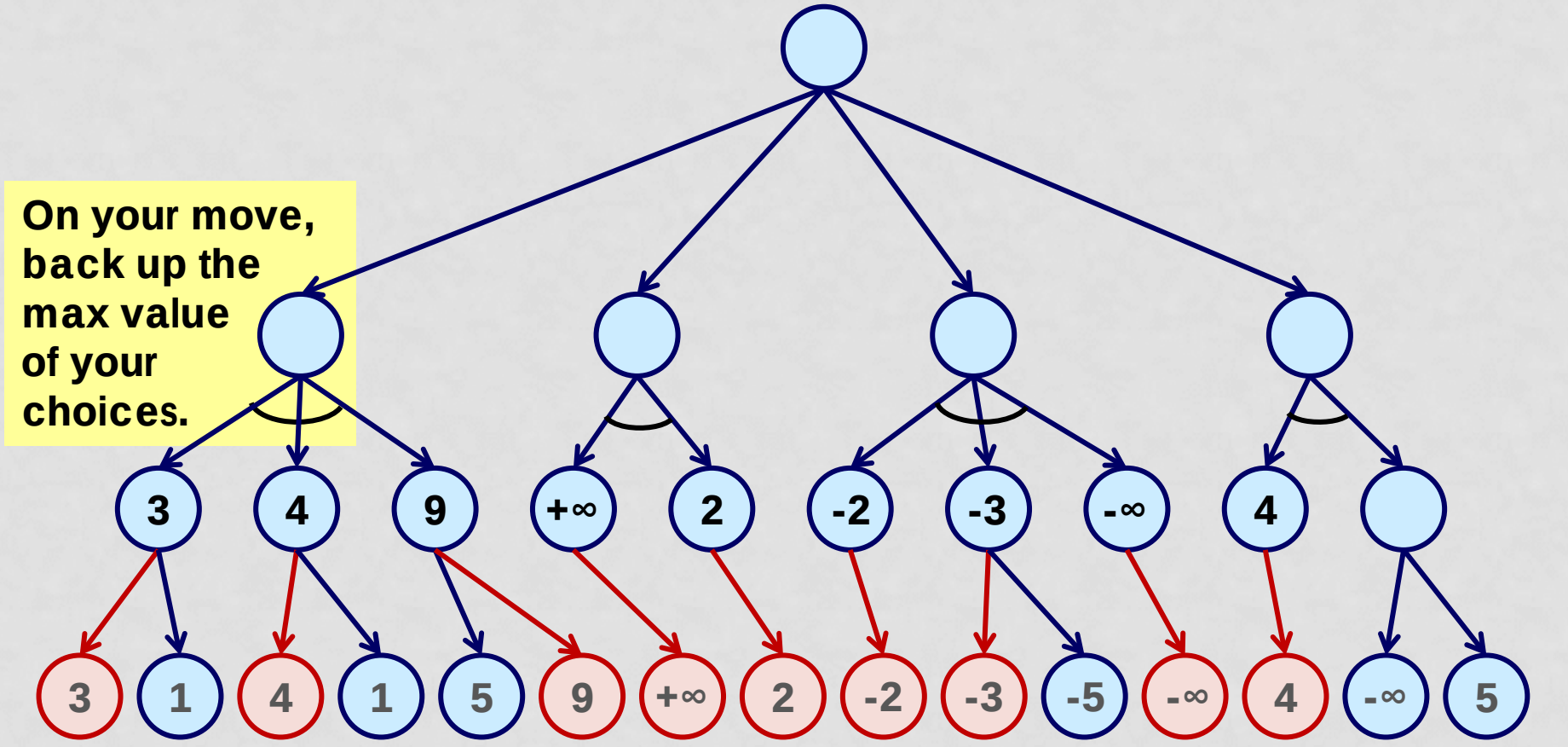

On your move, back up the max value of your choices.
3
4
9
+∞
3
1
4
1
5
9
+∞
2
-2
-3
-5
-∞
4
-∞
5

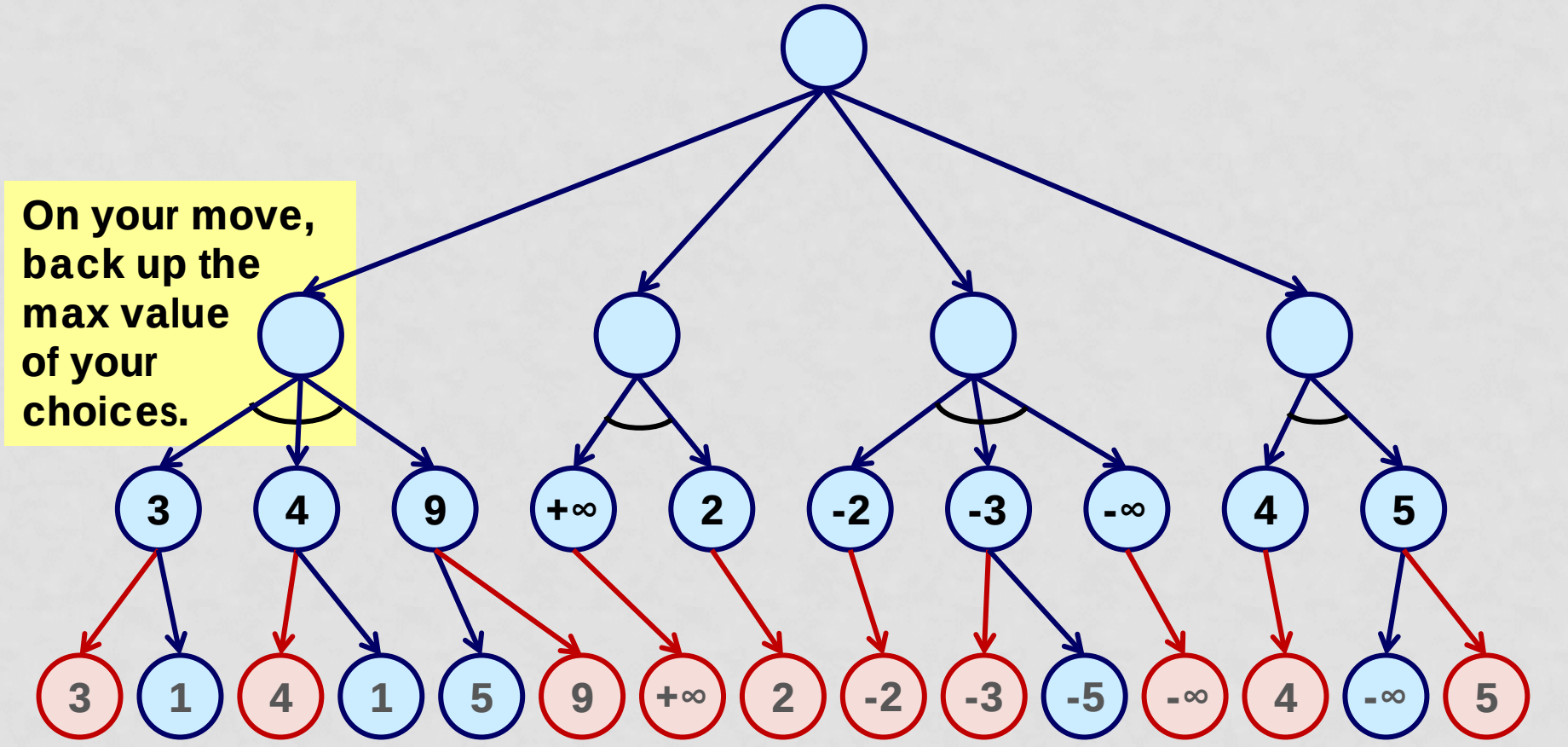

On your move, back up the max value of your choices.
3
4
9
+∞
2
3
1
4
1
5
9
+∞
2
-2
-3
-5
-∞
4
-∞
5


On your move, back up the max value of your choices.
3
4
9
+∞
2
-2
3
1
4
1
5
9
+∞
2
-2
-3
-5
-∞
4
-∞
5

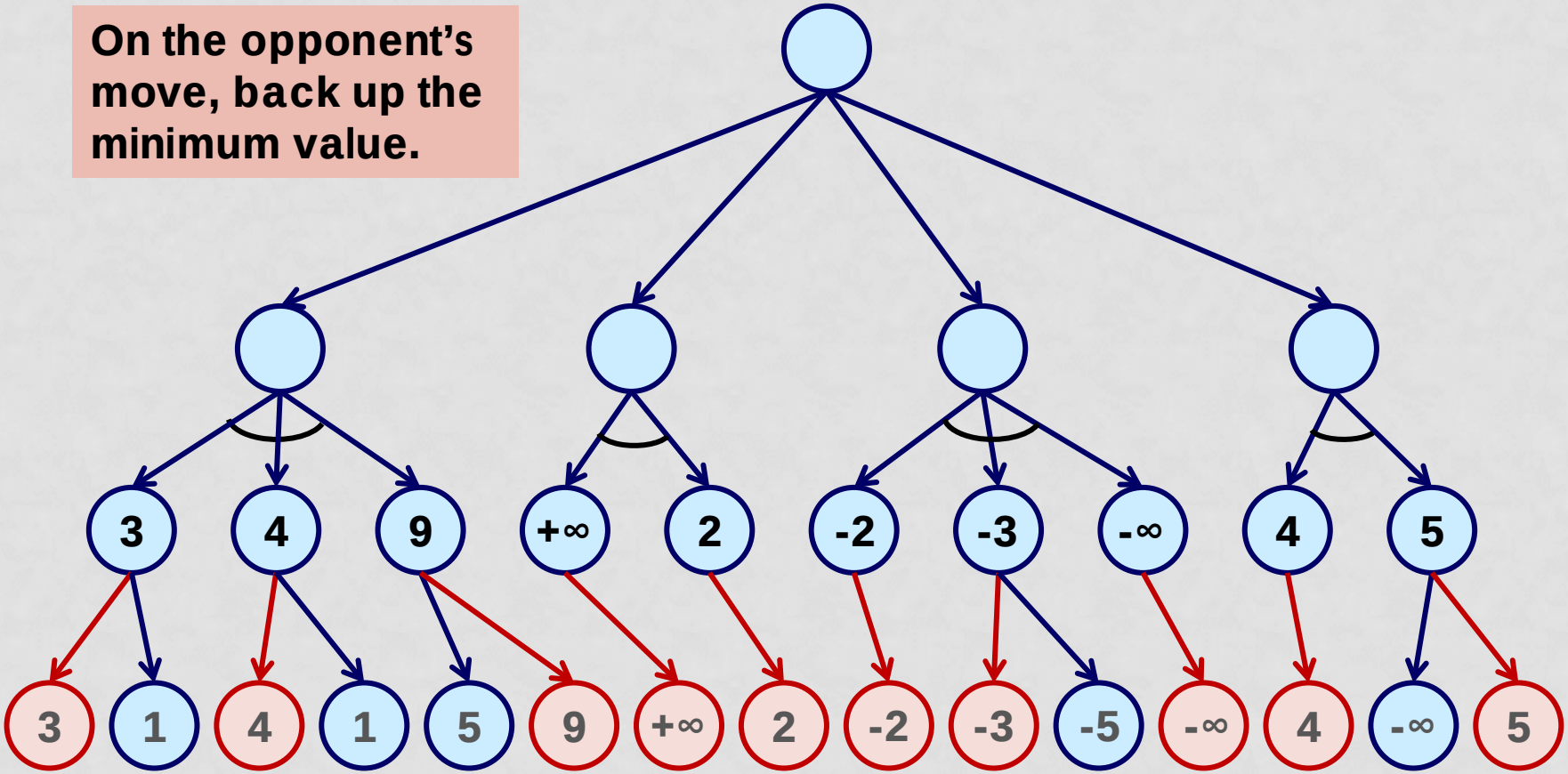

On your move, back up the max value of your choices.
3
4
9
+∞
2
-2
-3
3
1
4
1
5
9
+∞
2
-2
-3
-5
-∞
4
-∞
5


On your move, back up the max value of your choices.
3
4
9
+∞
2
-2
-3
-∞
3
1
4
1
5
9
+∞
2
-2
-3
-5
-∞
4
-∞
5

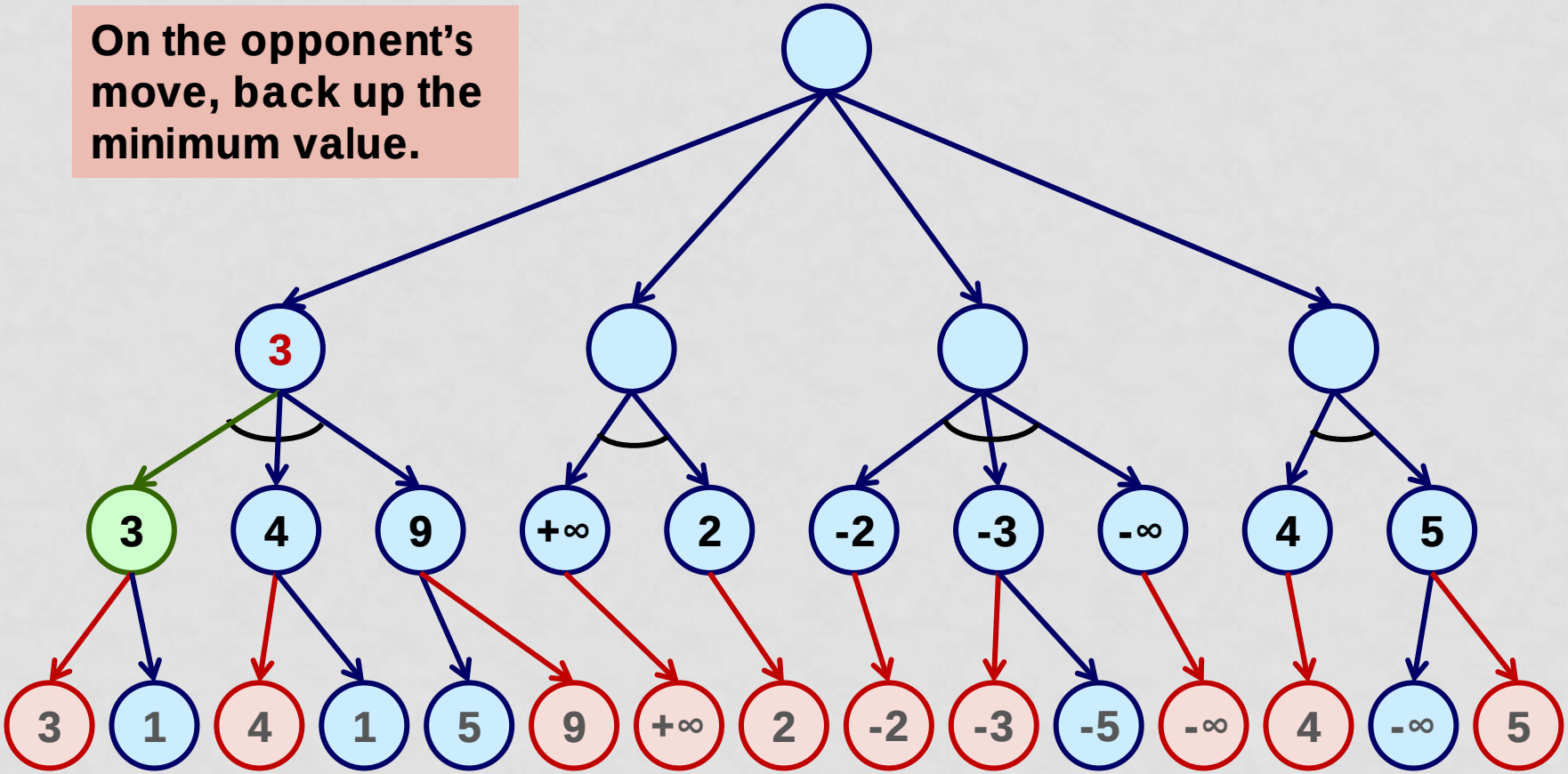

On your move, back up the max value of your choices.
3
4
9
+∞
2
-2
-3
-∞
4
3
1
4
1
5
9
+∞
2
-2
-3
-5
-∞
4
-∞
5


On your move, back up the max value of your choices.
3
4
9
+∞
2
-2
-3
-∞
4
5
3
1
4
1
5
9
+∞
2
-2
-3
-5
-∞
4
-∞
5

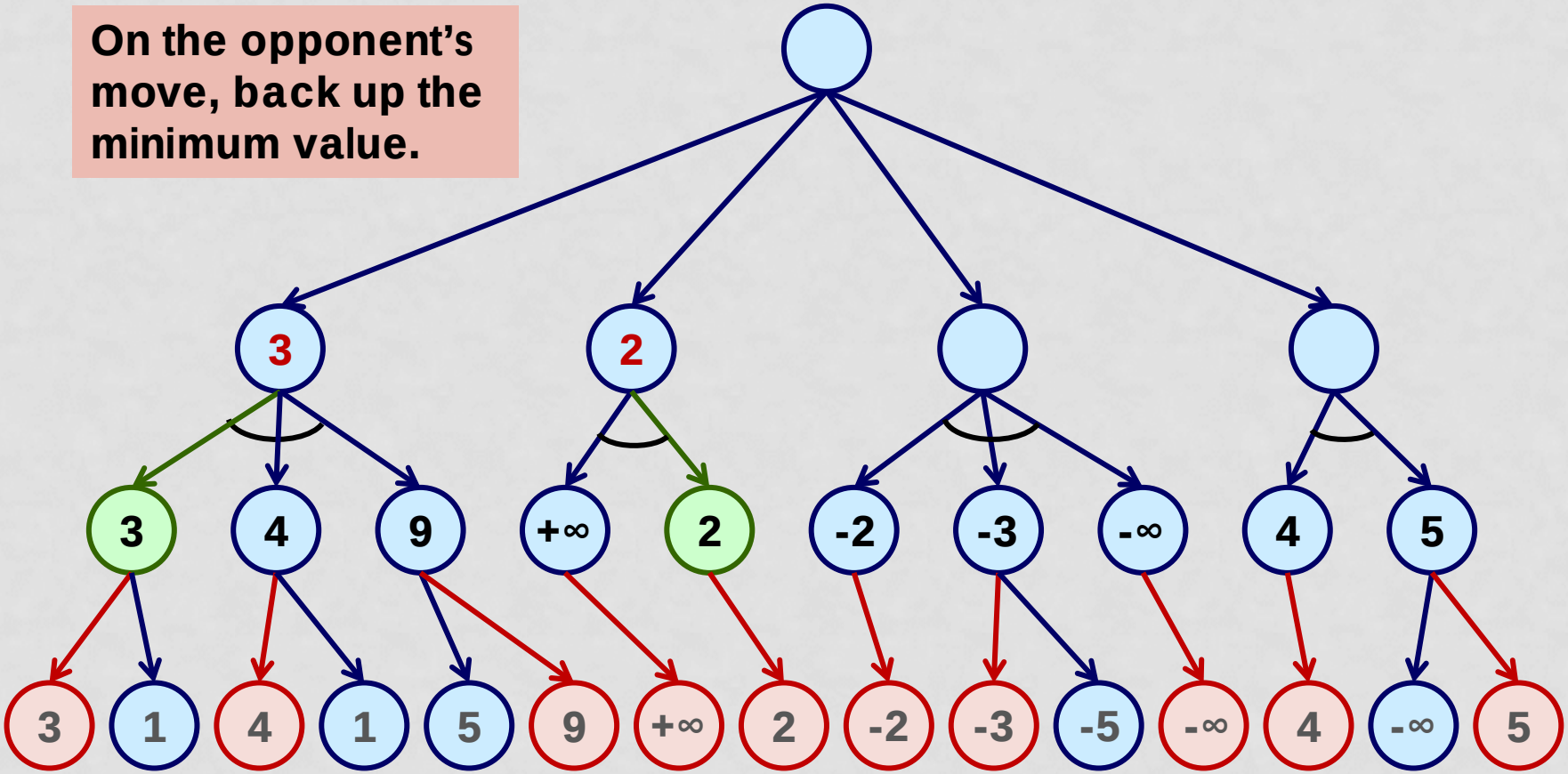
On the opponent's move, back up the minimum value.



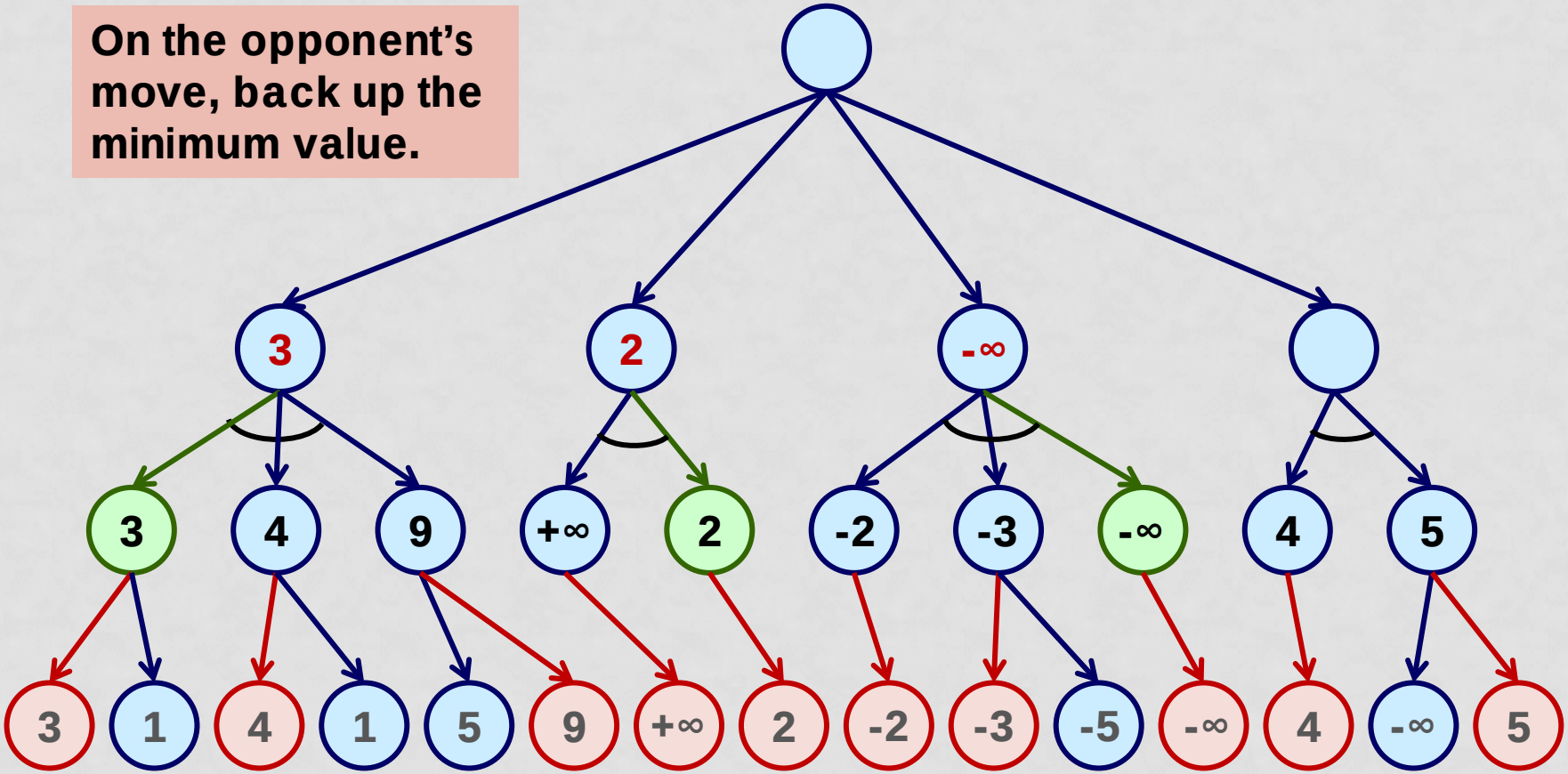
**On the opponent's move, back up the minimum value.**



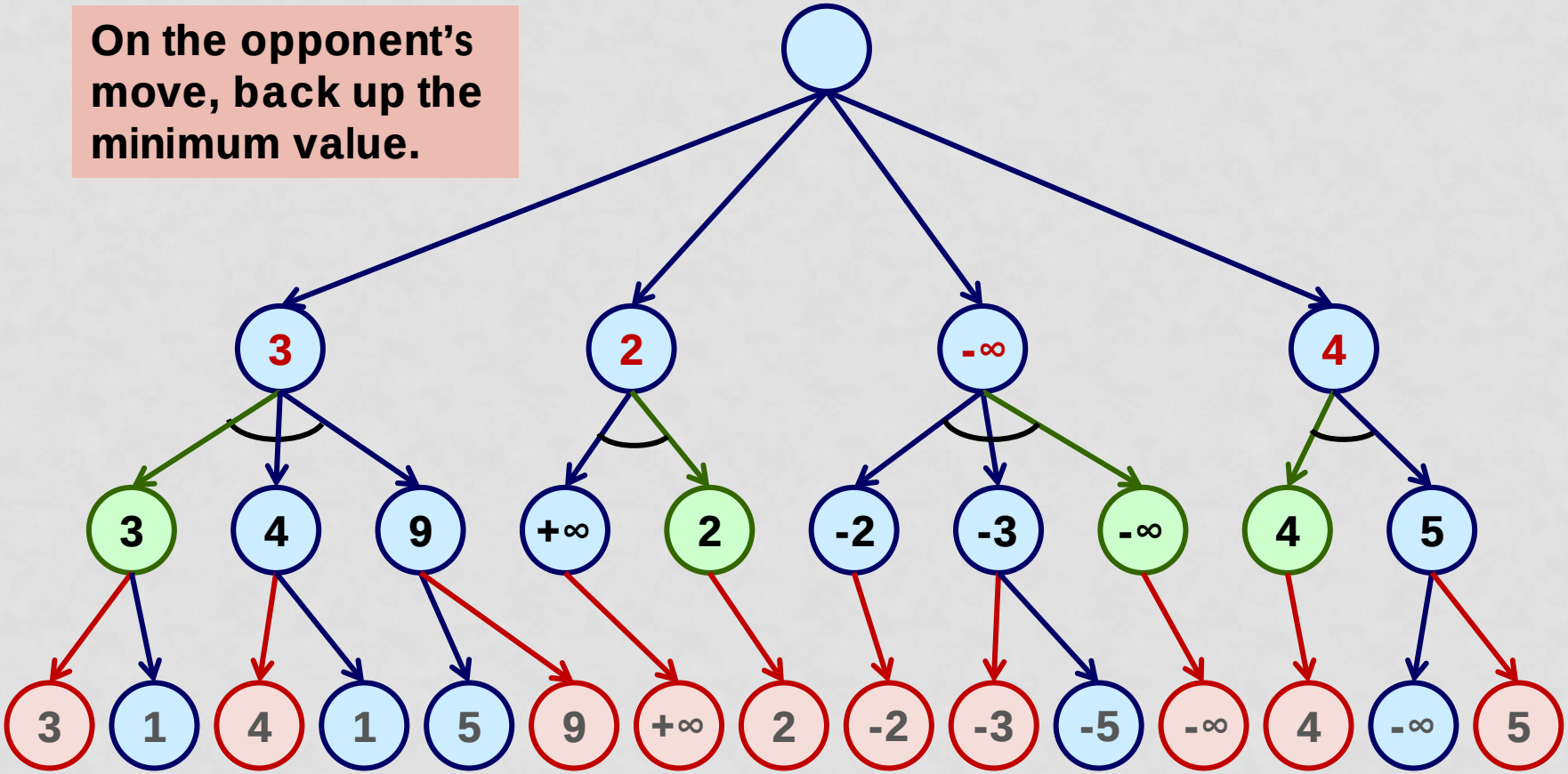
On the opponent's move, back up the minimum value.



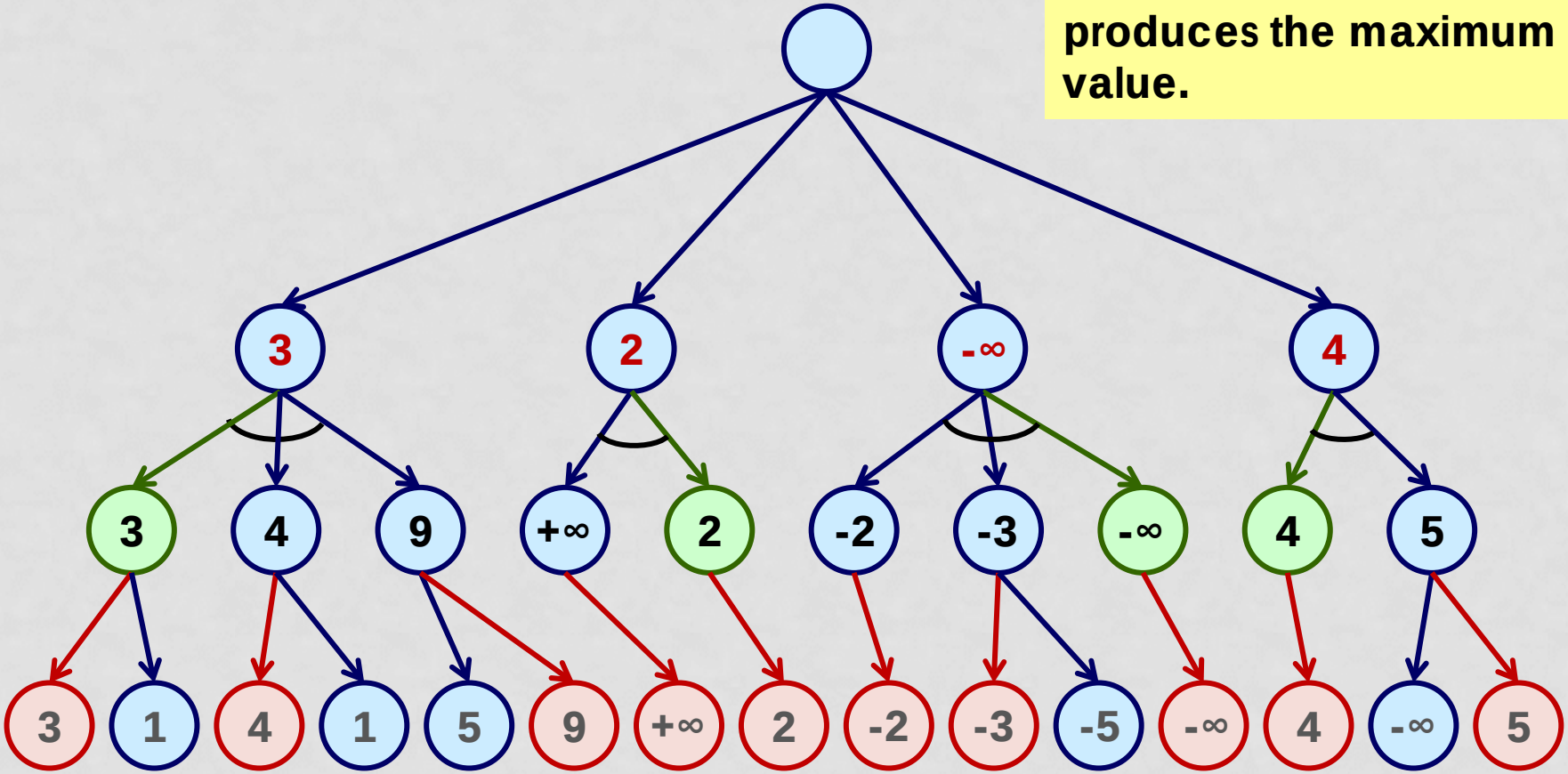
On the opponent's move, back up the minimum value.



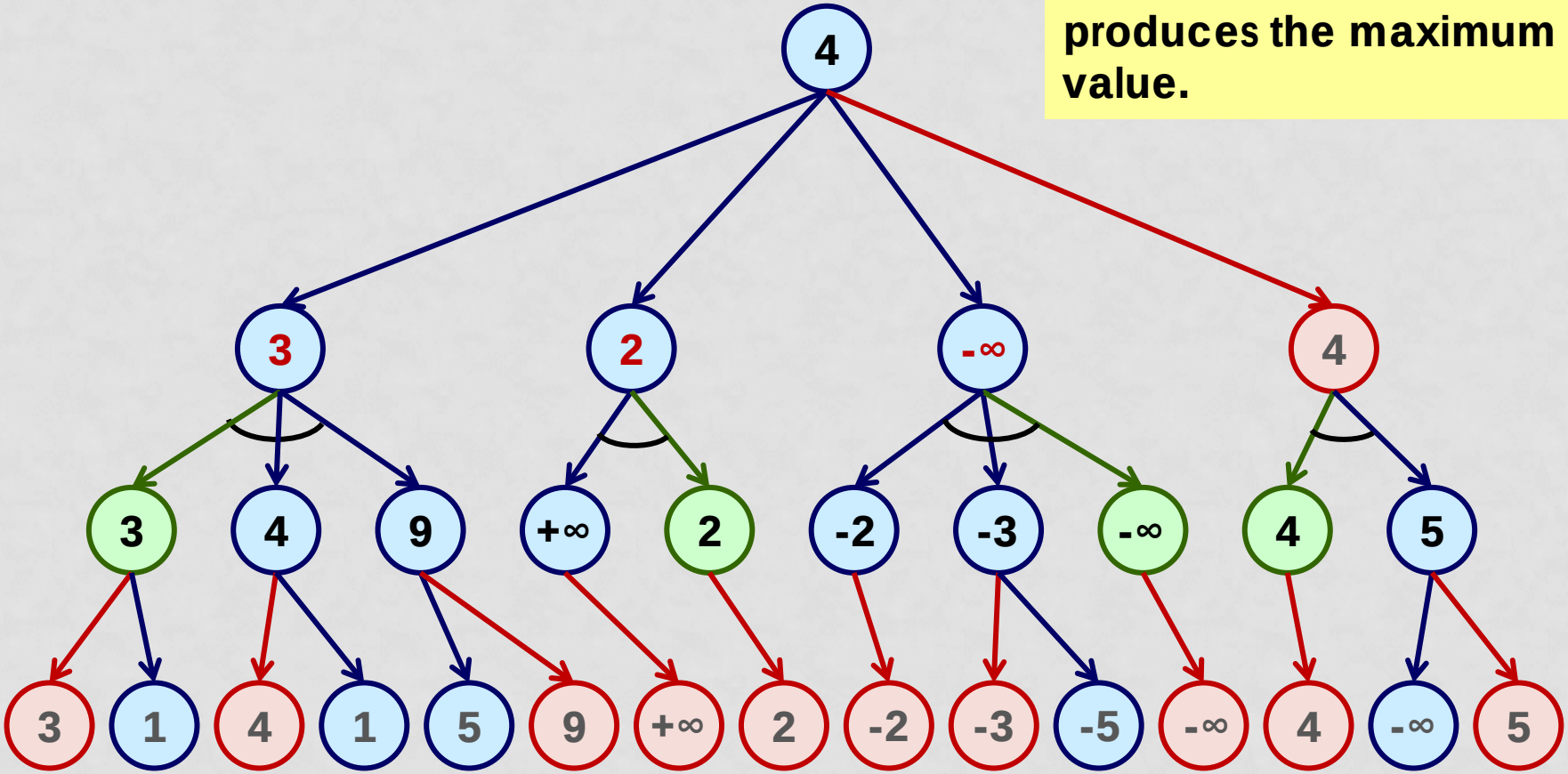
On the opponent's move, back up the minimum value.



On your move, back up the maximum value. Make the move that produces the maximum value.



On your move, back up the maximum value. Make the move that produces the maximum value.



```
// Given current state, return a
// move and "backed up value“
// Version 0: no lookahead; choose random move
```

## Minimax( state )

```
move = random m in MyMoves(state)
nextState ← applyMove(move,state)

val  ← h(nextState)

return { move, val }
```

∞ = value of win

-∞ = value of loss

**h()** is evaluation function based on game state, which includes who's turn it is

**val** is your value

**move** is move that produced **Max**

```
// Given current state, look ahead and return a
// move and "backed up value“
// Version 1: 1-move lookahead (yours)
Minimax( state )

Max ← -(∞+1)
for each move m in MyMoves(state)
    nextState ← applyMove(m,state)

    val  ← h(nextState)

    if val > Max
       Max ← val
       move ← m

return { move, Max }
```

∞ = value of win

-∞ = value of loss

**h()** is evaluation function "*killer heuristic*" - no further look-ahead (hope heuristic is good)

**Max** is your value [maximum of all choices]

**move** is move that produced **Max**

```
// Given current state, look ahead and return a
// move and "backed up value“
// Version 1: 1-move lookahead (yours)
Minimax( state )

  Max ← -(∞+1)
  for each move m in MyMoves(state)
     nextState ← applyMove(m,state)

     val  ← h(nextState)

     if val > Max
        Max ← val
        move ← m

  return { move, Max }
```

∞ = value of win

-∞ = value of loss

**h()** is evaluation function

"*killer heuristic*" - no further look-ahead (hope heuristic is good)

**Max** is your value
[maximum of all choices]

**move** is move that produced **Max**

```
// Given current state, look ahead and return a
// move and "backed up value“
// Version 2: 2-move lookahead (yours + opponent's)
Minimax( state )

Max ← -(∞+1)
for each move m in MyMoves(state)
   nextState ← applyMove(m,state)

   Min ← +(∞+1)

   for each move n in opponentMoves(nextState)
      newState ← applyMove(n, nextState)
      if h(newState) < Min
         Min ← h(newState)

   if Min > Max
      Max ← Min
      move ← m

return { move, Max }
```

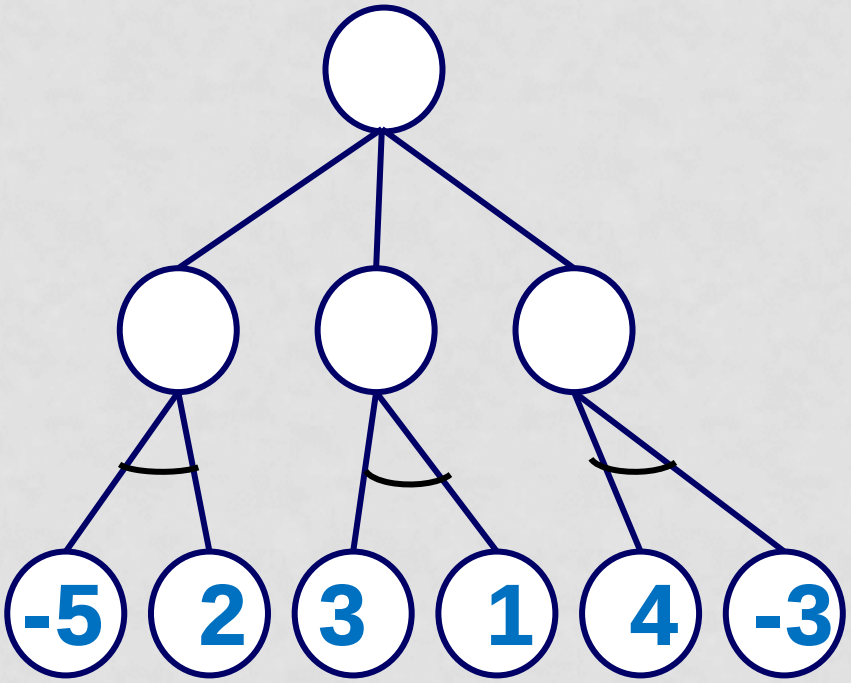
∞ = value of win
-∞ = value of loss

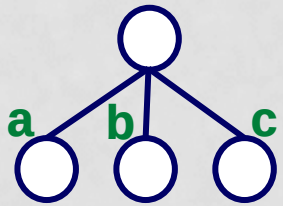
opponent chooses a move with Min value

**h()** is evaluation function
**Min** is opponent's value
[minimum of all choices]
**Max** is your value
[maximum of all choices]
**move** is move that produced **Max**

# EXAMPLE

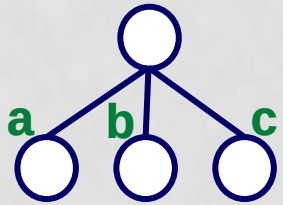
**Applying Minimax with Depth=2 to a sample problem:**





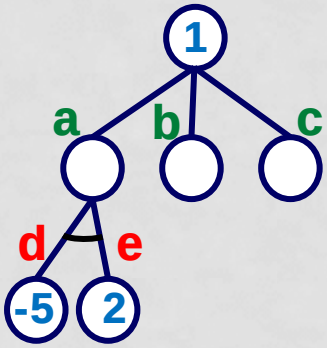

Max = -(∞+1) = -10001
My moves = {a, b, c}

∞ = 10000
Move a results in state A, etc.




Max = -(∞+1) = -10001
My moves = {a, b, c}

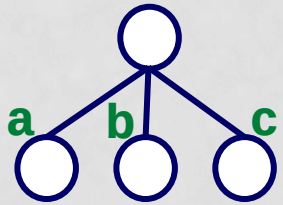
∞ = 10000
Move a results in state A, etc.




nextState = A
Min = (∞+1) = 10001
OpponentMoves= { d, e }

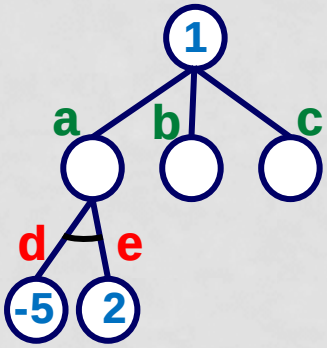
h(D) = -5 < Min (= 10001)?? Yes => Min = -5
h(E) =2 < Min ? NO

Done looking at opp. moves (value = -5)




Max = -(∞+1) = -10001
My moves = {a, b, c}

∞ = 10000
Move a results in state A, etc.

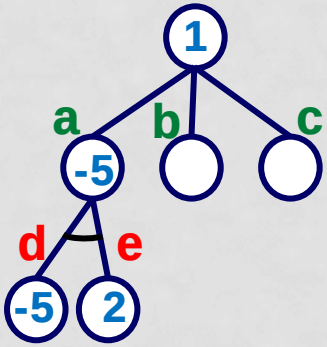



nextState = A
Min = (∞+1) = 10001
OpponentMoves= { d, e }

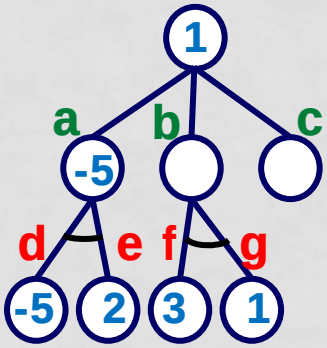
h(D) = -5 < Min (= 10001)?? Yes => Min = -5
h(E) =2 < Min ? NO

Done looking at opp. moves (value = -5)

if(Min>Max) YES
Max = Min (= -5)
move = a




first move (a) has backed-up value of -5




Max = -(∞+1) = -10001
My moves = {a, b, c}

∞ = 10000
Move a results in state A, etc.

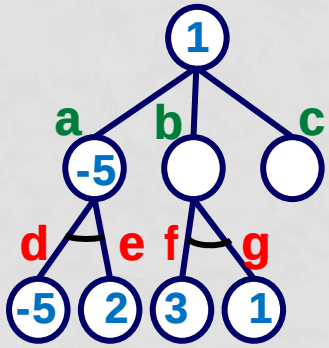
nextState = B
Min = (∞+1) = 10001
OpponentMoves = { f, g }

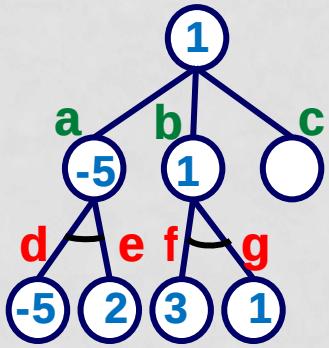
h(F)= 3 < Min (= 10001)? YES => Min = 3
h(G)= 1 < Min (=3)? YES => Min = 1

Done looking at opp. moves (value = 1)

Max = -(∞+1) = -10001
My moves = {a, b, c}

∞ = 10000
Move a results in state A, etc.



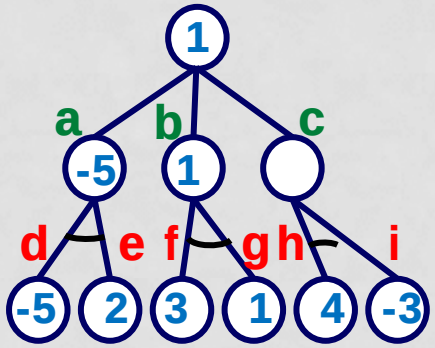




nextState = B
Min = (∞+1) = 10001
OpponentMoves = { f, g }

h(F)= 3 < Min (= 10001)? YES => Min = 3
h(G)= 1 < Min (=3)? YES => Min = 1

Done looking at opp. moves (value = 1)

if(Min>Max) YES
Max = Min (= 1)
move = b




Max = -(∞+1) = -10001
My moves = {a, b, c}

∞ = 10000
Move a results in state A, etc.

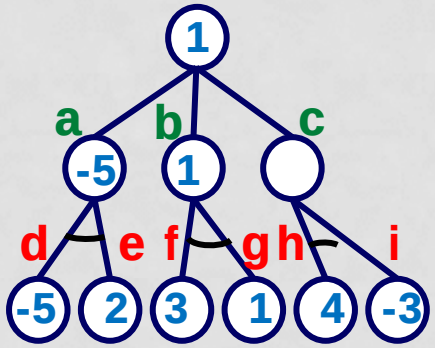
nextState = C
Min = (∞+1) = 10001
OpponentMoves = { h, i }

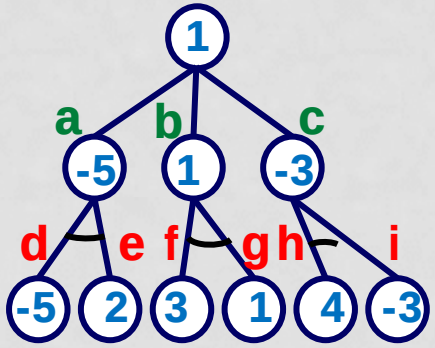
h(H)= 4 < Min (= 10001)? YES => Min = 4
h(I)= -3 < Min (=4)? YES => Min = -3

Done looking at opp. moves (value = -3)

Max = -(∞+1) = -10001
My moves = {a, b, c}

∞ = 10000
Move a results in state A, etc.




nextState = C
Min = (∞+1) = 10001
OpponentMoves = { h, i }

h(H)= 4 < Min (= 10001)? YES => Min = 4
h(I)= -3 < Min (=4)? YES => Min = -3

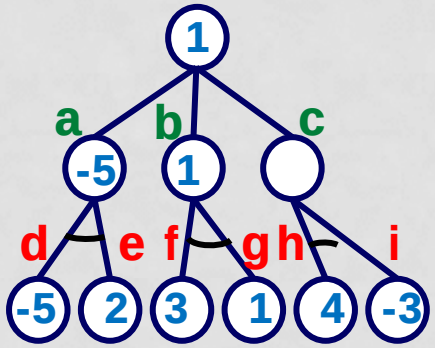
Done looking at opp. moves (value = -3)

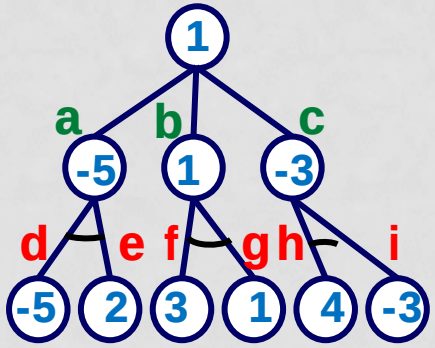
if(Min>Max) NO

third move(c) has a backed-up value of -3; do not prefer this move

Max = -(∞+1) = -10001
My moves = {a, b, c}

∞ = 10000
Move a results in state A, etc.




nextState = C
Min = (∞+1) = 10001
OpponentMoves = { h, i }

h(H)= 4 < Min (= 10001)? YES => Min = 4
h(I)= -3 < Min (=4)? YES => Min = -3

Done looking at opp. moves (value = -3)

if(Min>Max) NO

third move(c) has a backed-up value of -3; do not prefer this move

Return Max (= 1)
move = b