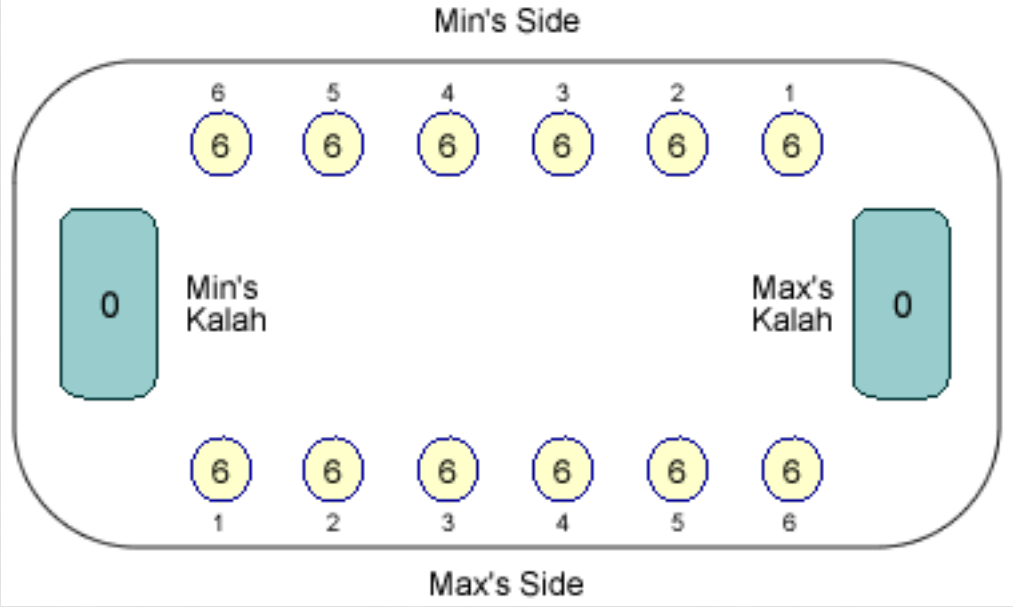
CASE STUDY - KALAH

JEFFREY L. POPYACK

KALAH

Kalah, also known as *Mancala*, *Wari*, or *Owari*, originated in Africa.

- Two players (Max & Min)
- Six *pits* for each player and larger pit (*Kalah*) on their right.



Game begins with each small pit filled with some number *k* of stones (e.g., *k=6*)

Excerpted from “Etudes for Programmers”, Charles Wetherell, Prentice Hall, 1978.

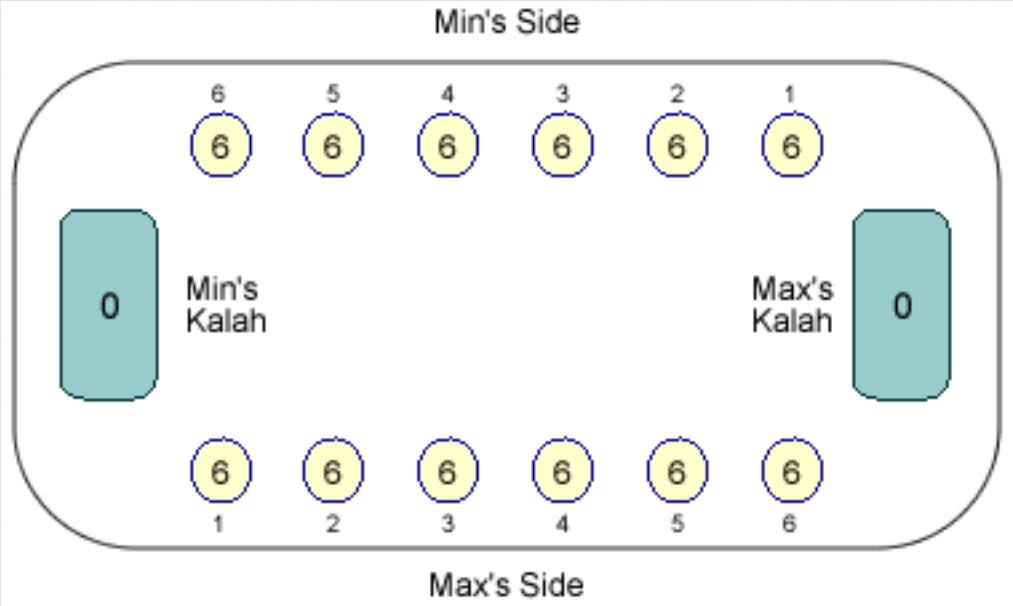
KALAH

On player's turn:

- Remove all stones from one of the player's small pits,
- Sow them into other pits counterclockwise around the board.

Sowing begins to the right of the source pit

- includes player's own Kalah
- includes opponent's small pits,
- does not include opponent's Kalah.



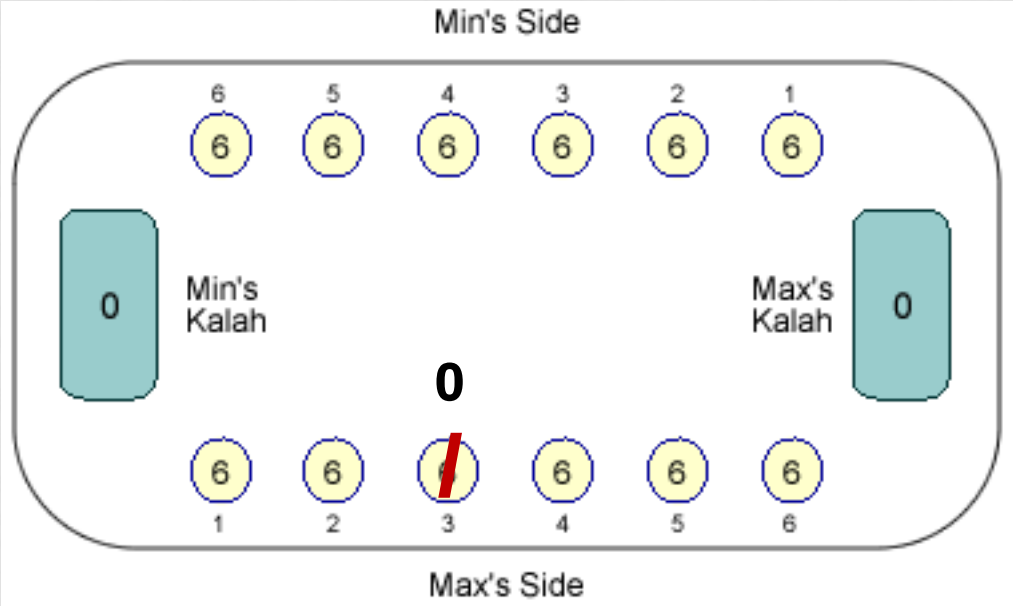
KALAH

On player's turn:

- Remove all stones from one of the player's small pits,
- Sow them into other pits counterclockwise around the board.

Sowing begins to the right of the source pit

- includes player's own Kalah
- includes opponent's small pits,
- does not include opponent's Kalah.



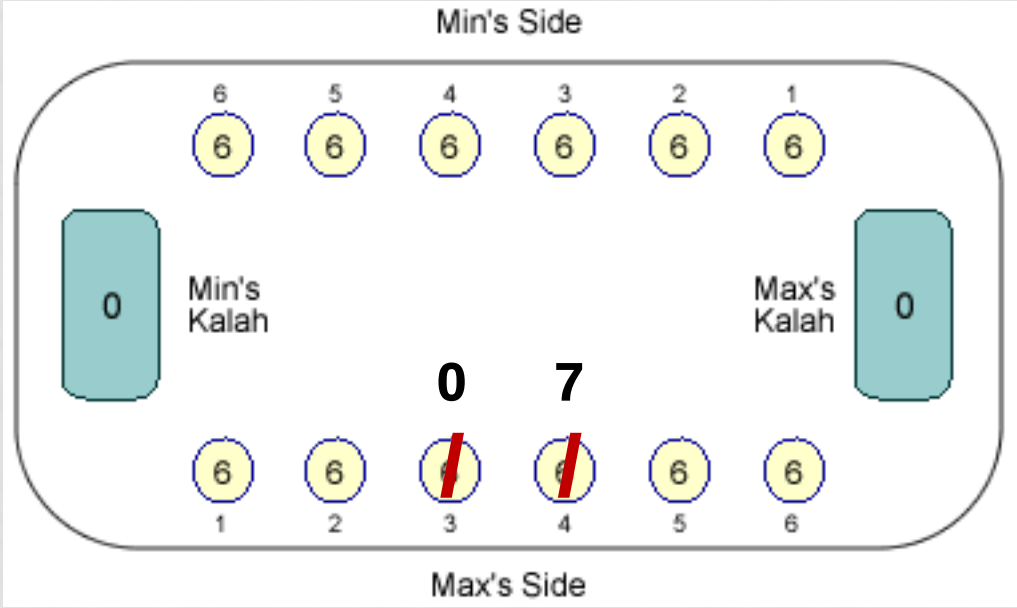
KALAH

On player's turn:

- Remove all stones from one of the player's small pits,
- Sow them into other pits counterclockwise around the board.

Sowing begins to the right of the source pit

- includes player's own Kalah
- includes opponent's small pits,
- does not include opponent's Kalah.



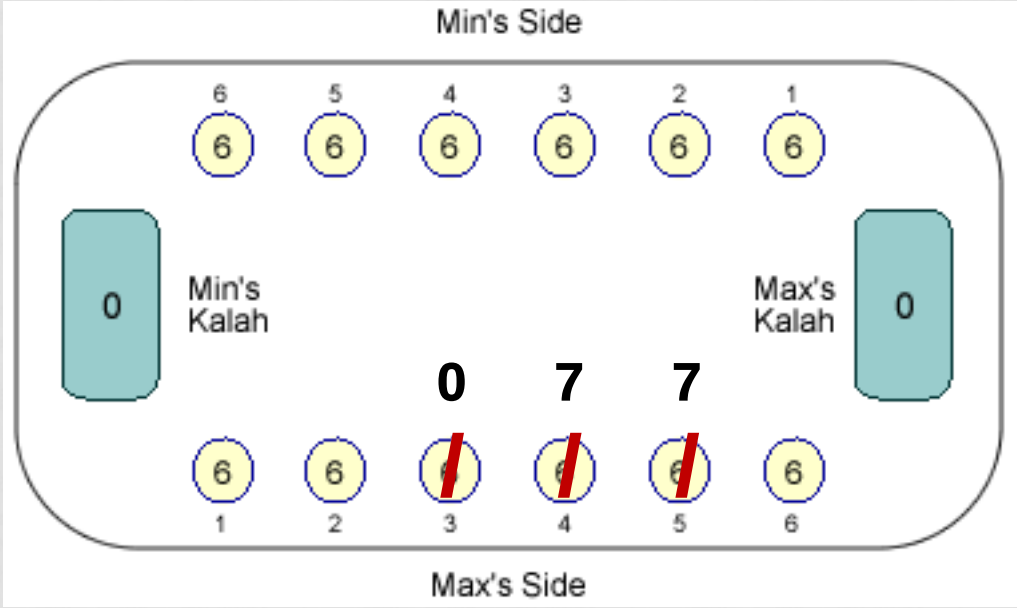
KALAH

On player's turn:

- Remove all stones from one of the player's small pits,
- Sow them into other pits counterclockwise around the board.

Sowing begins to the right of the source pit

- includes player's own Kalah
- includes opponent's small pits,
- does not include opponent's Kalah.



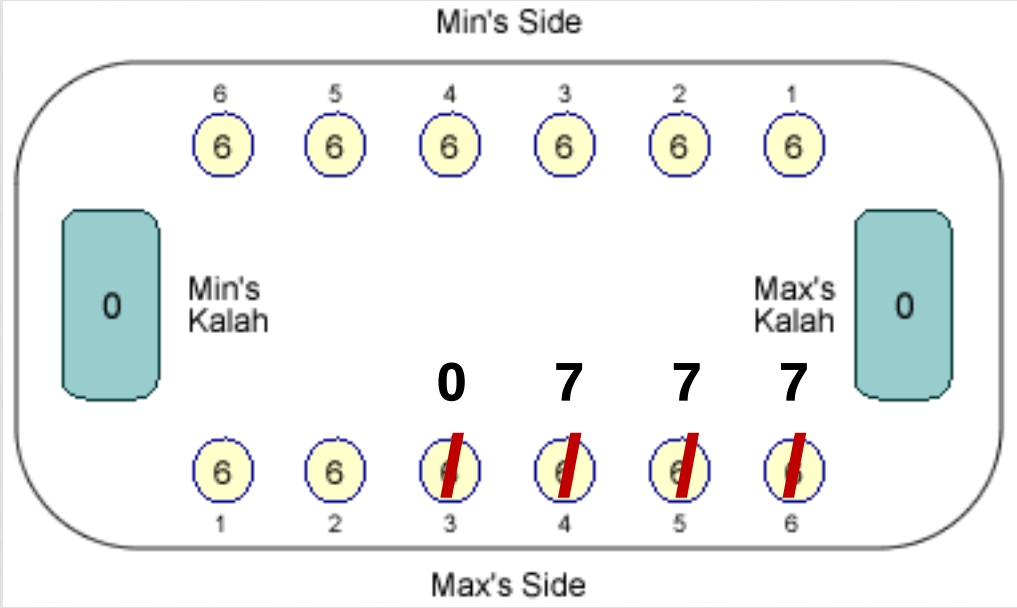
KALAH

On player's turn:

- Remove all stones from one of the player's small pits,
- Sow them into other pits counterclockwise around the board.

Sowing begins to the right of the source pit

- includes player's own Kalah
- includes opponent's small pits,
- does not include opponent's Kalah.



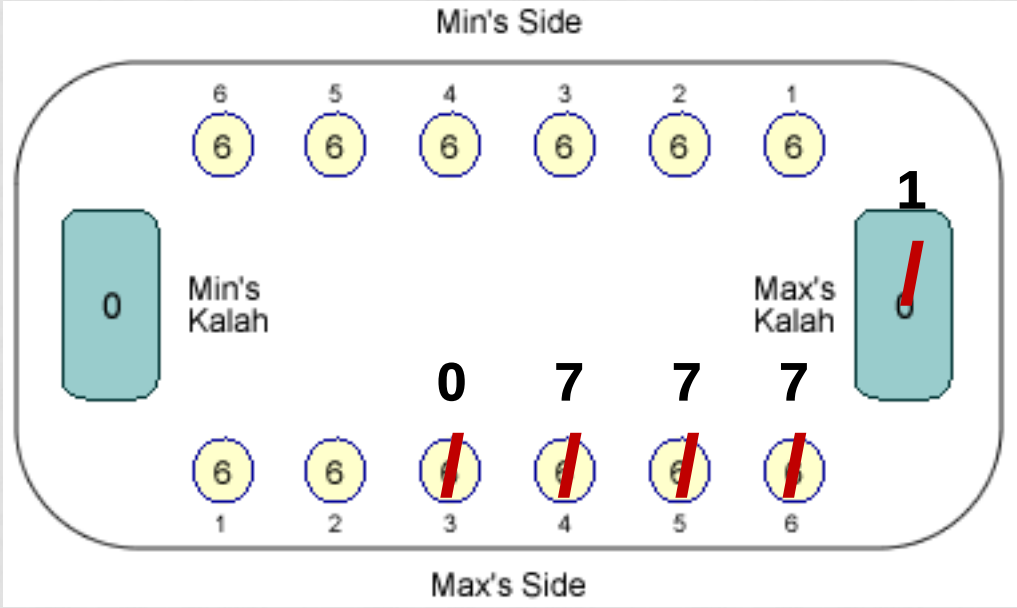
KALAH

On player's turn:

- Remove all stones from one of the player's small pits,
- Sow them into other pits counterclockwise around the board.

Sowing begins to the right of the source pit

- includes player's own Kalah
- includes opponent's small pits,
- does not include opponent's Kalah.



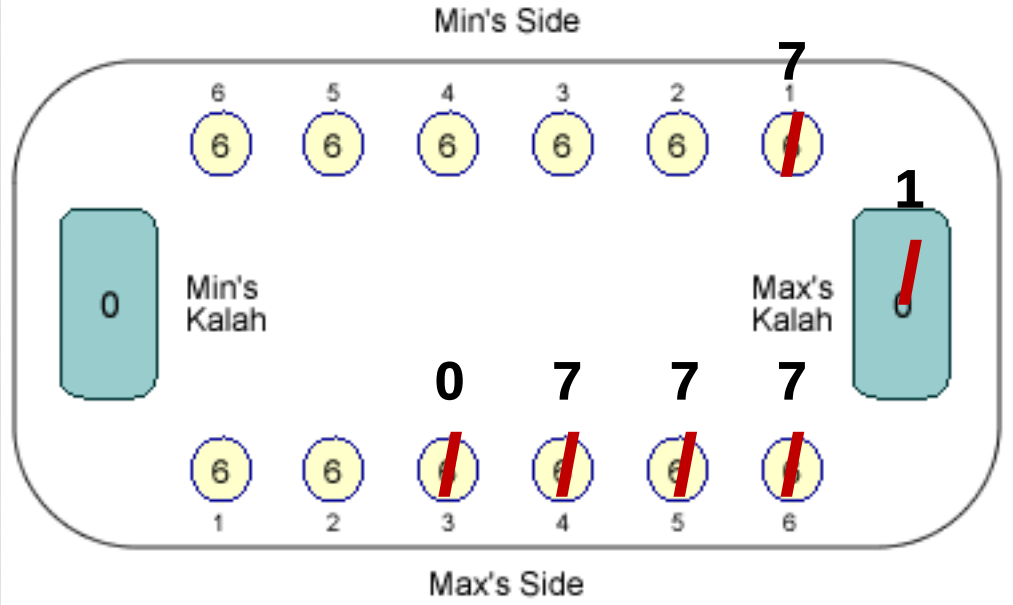
KALAH

On player's turn:

- Remove all stones from one of the player's small pits,
- Sow them into other pits counterclockwise around the board.

Sowing begins to the right of the source pit

- includes player's own Kalah
- includes opponent's small pits,
- does not include opponent's Kalah.



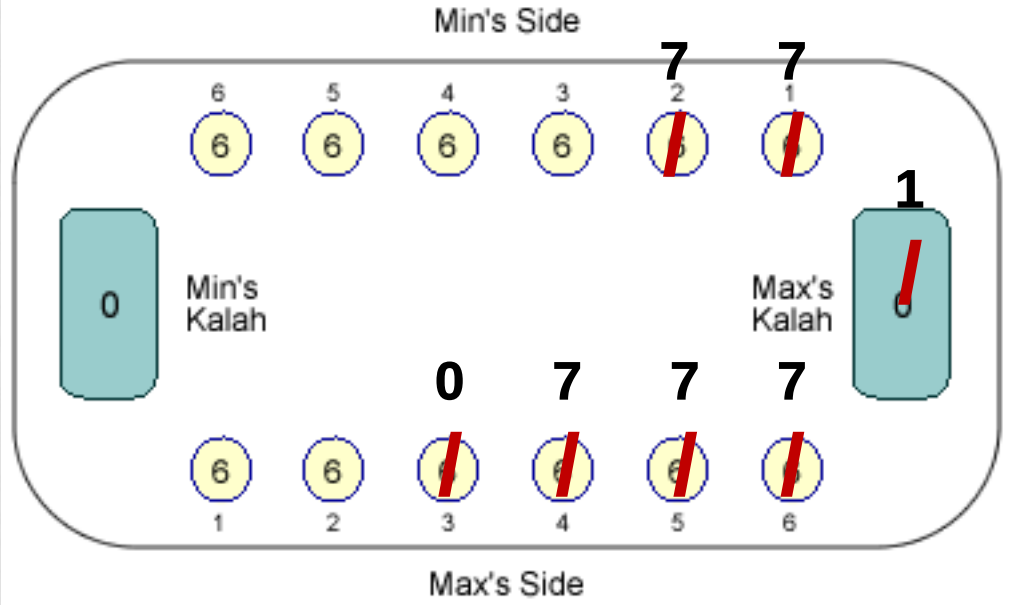
KALAH

On player's turn:

- Remove all stones from one of the player's small pits,
- Sow them into other pits counterclockwise around the board.

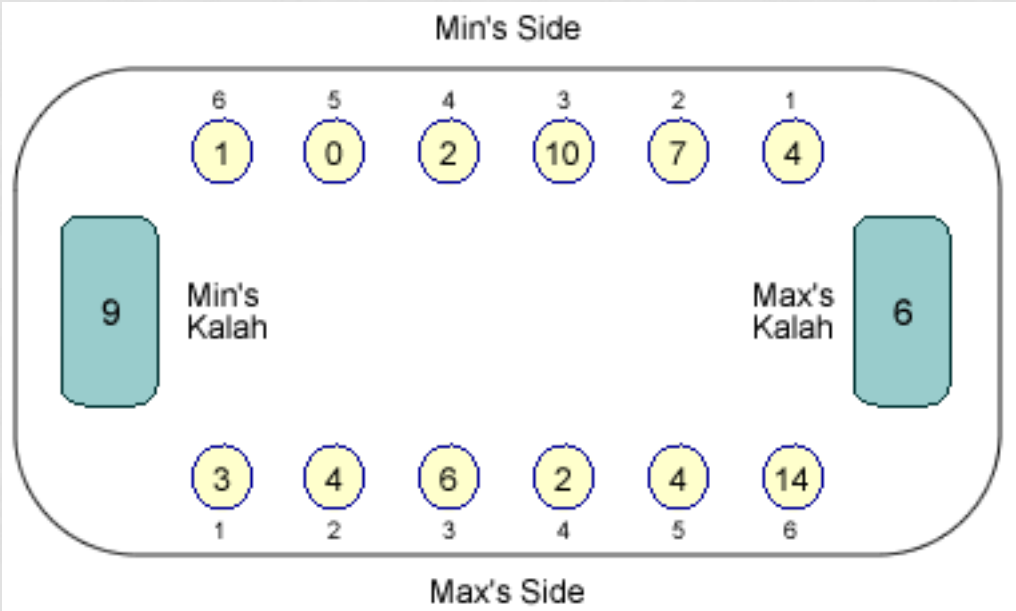
Sowing begins to the right of the source pit

- includes player's own Kalah
- includes opponent's small pits,
- does not include opponent's Kalah.



KALAH – LOOPING MOVE

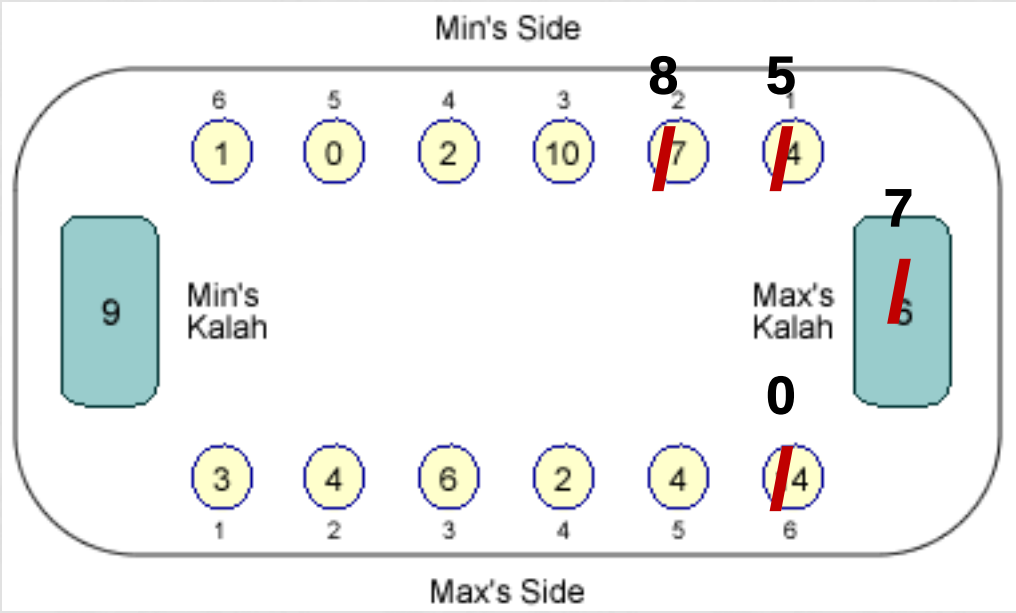
With sufficient number of stones, sowing can continue around the board, back to the player's side. (*Looping Move*)



In this configuration, it is Max's turn to move.
Max will choose to move from Pit 6, performing a Looping Move.

KALAH – LOOPING MOVE

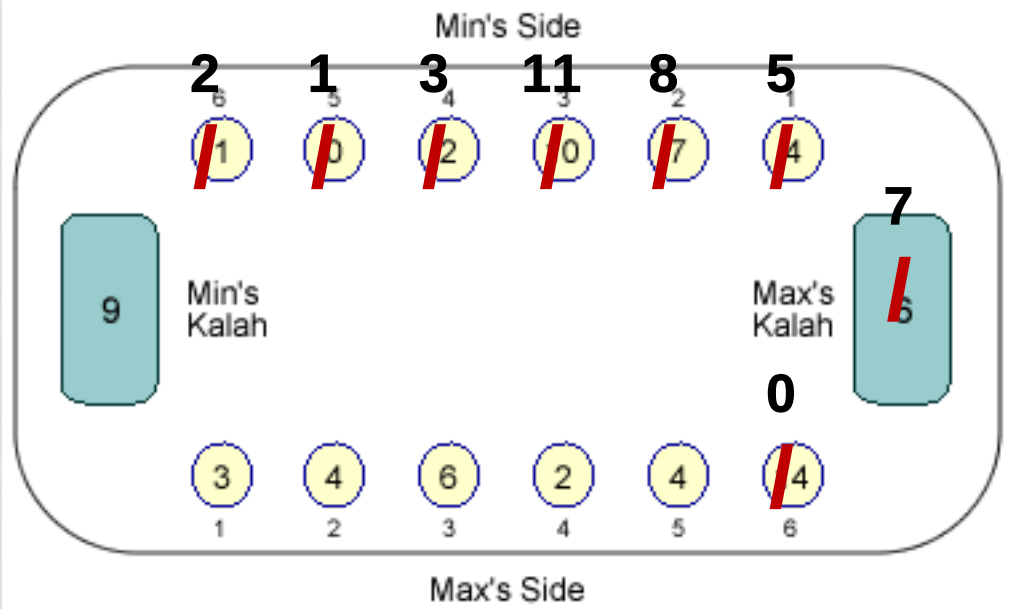
With sufficient number of stones, sowing can continue around the board, back to the player's side. (*Looping Move*)



In this configuration, it is Max's turn to move.
Max will choose to move from Pit 6, performing a Looping Move.

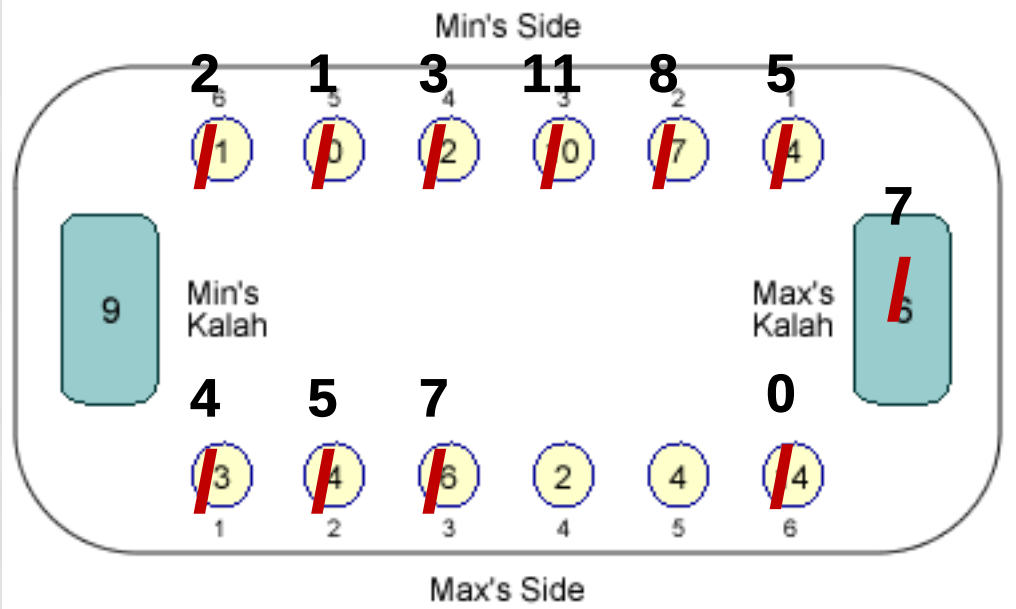
KALAH – LOOPING MOVE

With sufficient number of stones, sowing can continue around the board, back to the player's side. (*Looping Move*)



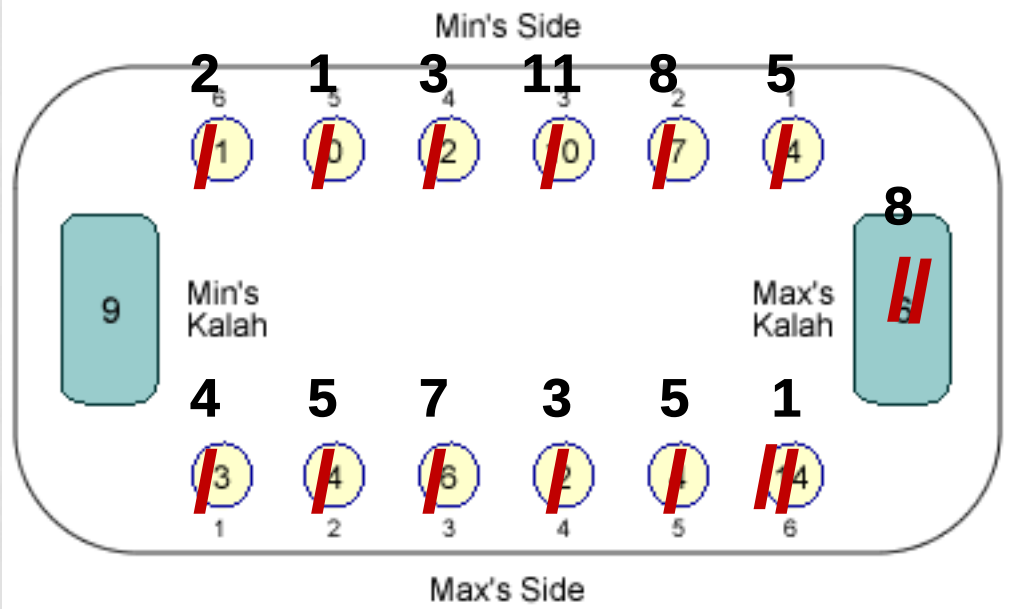
KALAH – LOOPING MOVE

With sufficient number of stones, sowing can continue around the board, back to the player's side. (*Looping Move*)



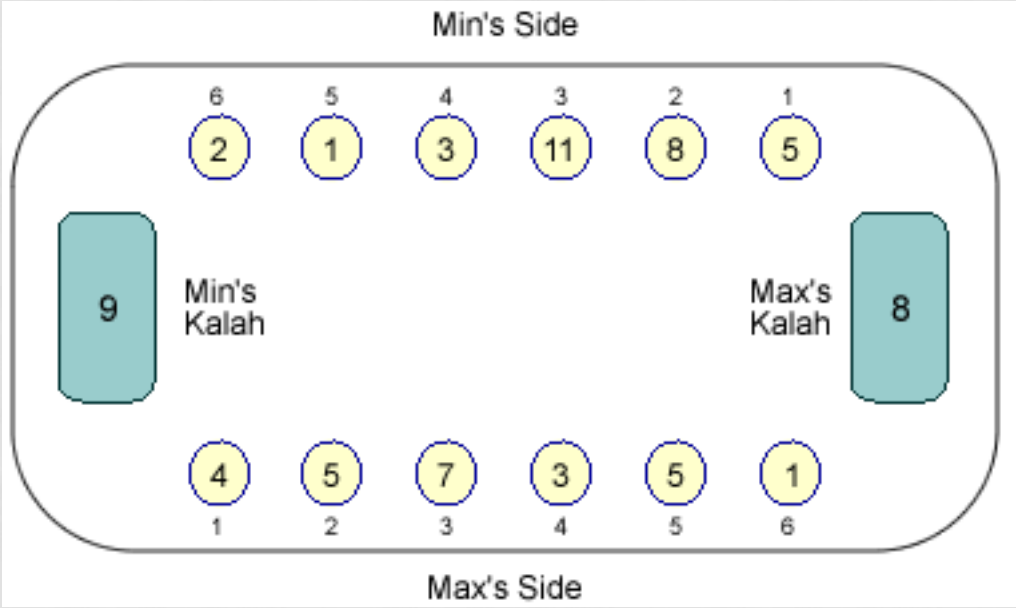
KALAH – LOOPING MOVE

With sufficient number of stones, sowing can continue around the board, back to the player's side. (*Looping Move*)



KALAH – LOOPING MOVE

With sufficient number of stones, sowing can continue around the board, back to the player's side. (*Looping Move*)



Max has completed the Looping Move. Max's Kalah has been sown twice; Min's Kalah has not been sown.

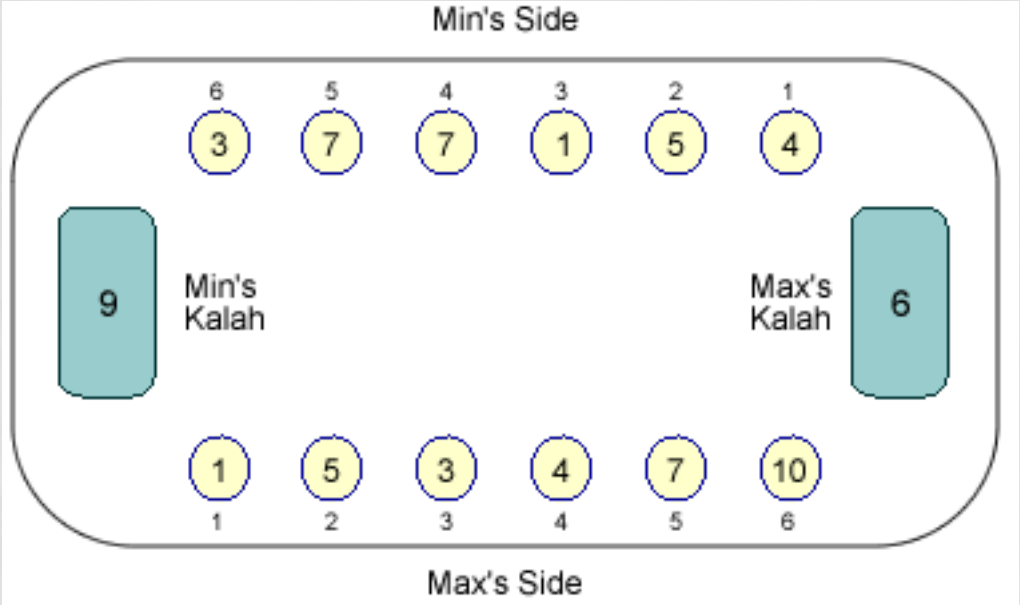
VARIATION #1 – *GO-AGAIN MOVE*

If last stone falls into one of moving player's own small *nonempty* pits;

and

Stones were played on the opponent's side during sowing:

- Stones in final pit are used to start a *Go-Again Move*, just like the original move. A player can have an arbitrarily long chain of go-agains.



Max's turn to move.
Max will choose to move from Pit 6,
performing a Looping Move,
with the last stone sown in Max's Pit 3.

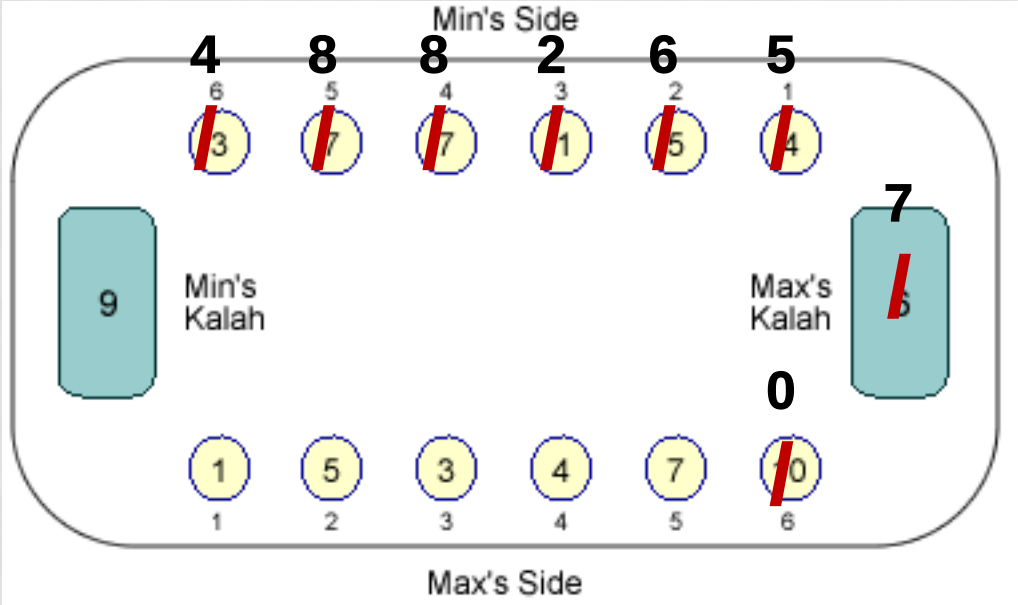
VARIATION #1 – *GO-AGAIN MOVE*

If last stone falls into one of moving player's own small *nonempty* pits;

and

Stones were played on the opponent's side during sowing:

- Stones in final pit are used to start a *Go-Again Move*, just like the original move. A player can have an arbitrarily long chain of go-agains.



Max's turn to move.
Max will choose to move from Pit 6,
performing a Looping Move,
with the last stone sown in Max's Pit 3.

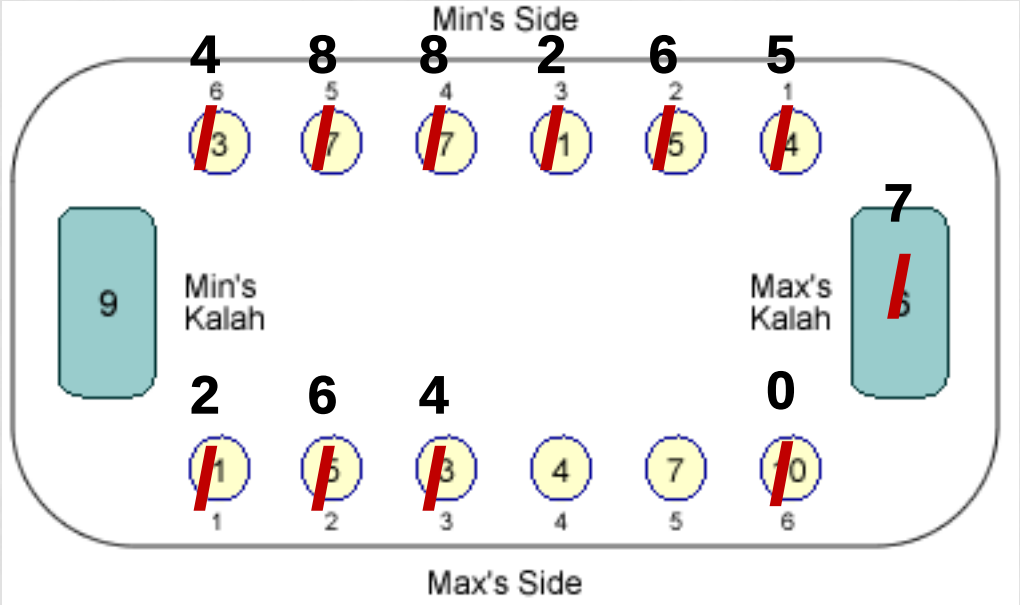
VARIATION #1 – *GO-AGAIN MOVE*

If last stone falls into one of moving player's own small *nonempty* pits;

and

Stones were played on the opponent's side during sowing:

- Stones in final pit are used to start a *Go-Again Move*, just like the original move. A player can have an arbitrarily long chain of go-agains.



Max's turn to move.
Max will choose to move from Pit 6, performing a Looping Move,
with the last stone sown in Max's Pit 3.

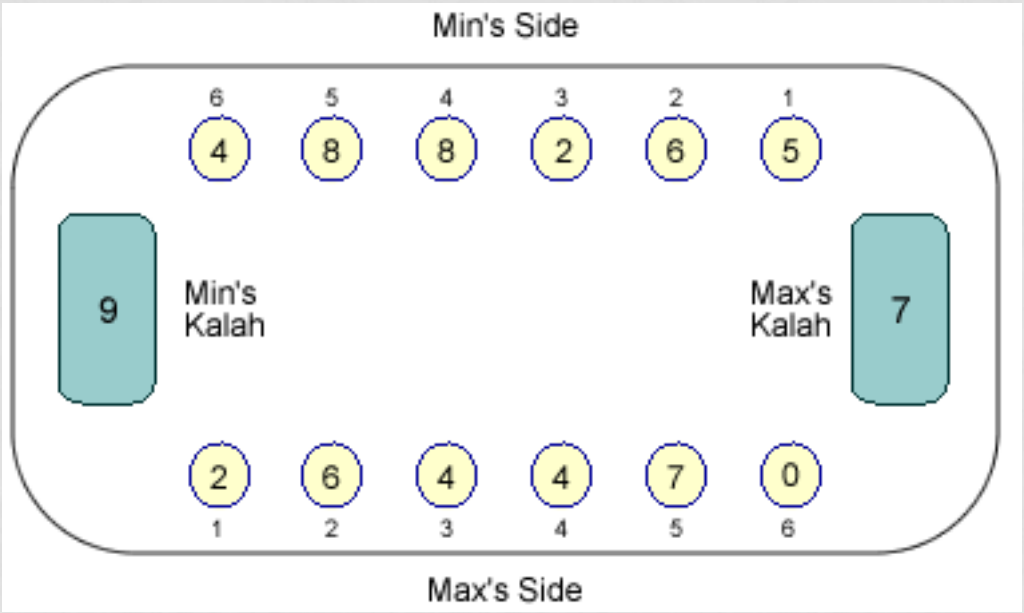
VARIATION #1 – *GO-AGAIN MOVE*

If last stone falls into one of moving player's own small *nonempty* pits;

and

Stones were played on the opponent's side during sowing:

- Stones in final pit are used to start a *Go-Again Move*, just like the original move. A player can have an arbitrarily long chain of go-agains.



Max has completed the Looping Move. Max is permitted to perform a Go-Again move, because a Looping Move caused the last stone to be sown in one of Max's non-empty pits.

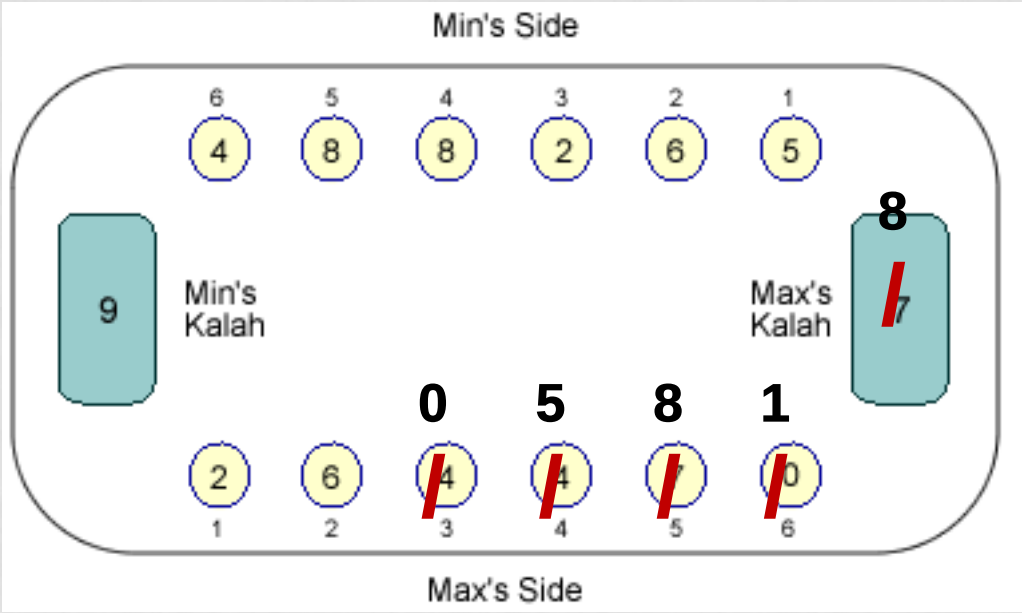
VARIATION #1 – *GO-AGAIN MOVE*

If last stone falls into one of moving player's own small *nonempty* pits;

and

Stones were played on the opponent's side during sowing:

- Stones in final pit are used to start a *Go-Again Move*, just like the original move. A player can have an arbitrarily long chain of go-agains.



Max has completed the Looping Move. Max is permitted to perform a Go-Again move, because a Looping Move caused the last stone to be sown in one of Max's non-empty pits.

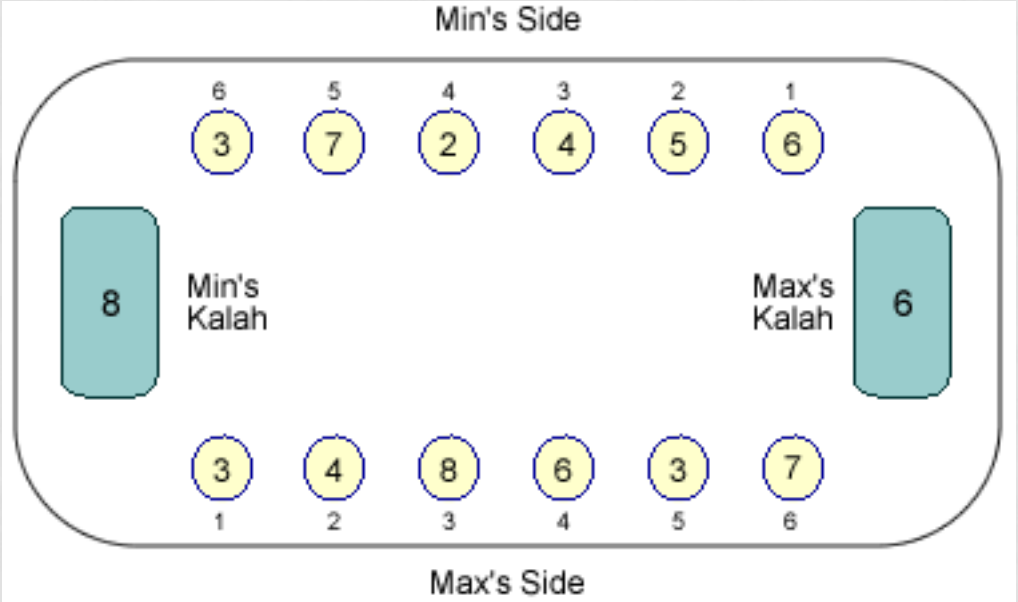
VARIATION #2 – *CAPTURE MOVE*

If final stone falls in one of opponent's pits

and

there are either <u>two or three stones</u> in the pit:

- the stones are *captured*, and placed in moving player's *Kalah*.
- Whenever one pit is captured, the preceding pit may be captured if it contains two or three stones as well.



Max's turn to move.
Max will choose to move from Pit 3. By sowing the last stone in Min's Pit 4, Max will capture the pit, since it will have 3 stones in it.

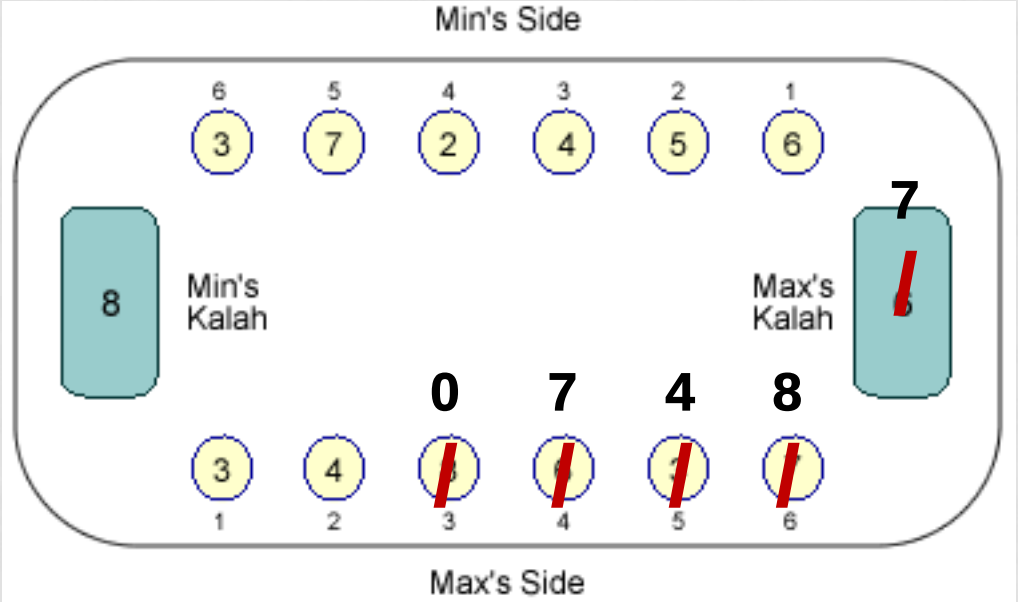
VARIATION #2 – *CAPTURE MOVE*

If final stone falls in one of opponent's pits

and

there are either <u>two or three stones</u> in the pit:

- the stones are *captured*, and placed in moving player's *Kalah*.
- Whenever one pit is captured, the preceding pit may be captured if it contains two or three stones as well.



Max's turn to move.
Max will choose to move from Pit 3. By sowing the last stone in Min's Pit 4, Max will capture the pit, since it will have 3 stones in it.

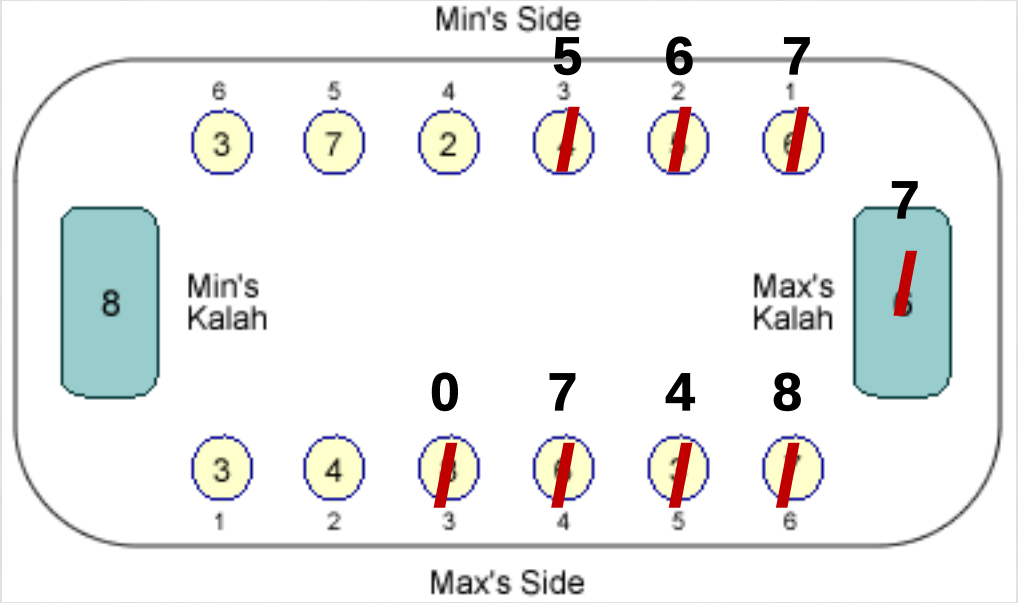
VARIATION #2 – *CAPTURE MOVE*

If final stone falls in one of opponent's pits

and

there are either <u>two or three stones</u> in the pit:

- the stones are *captured*, and placed in moving player's *Kalah*.
- Whenever one pit is captured, the preceding pit may be captured if it contains two or three stones as well.



Max's turn to move.
Max will choose to move from Pit 3. By sowing the last stone in Min's Pit 4, Max will capture the pit, since it will have 3 stones in it.

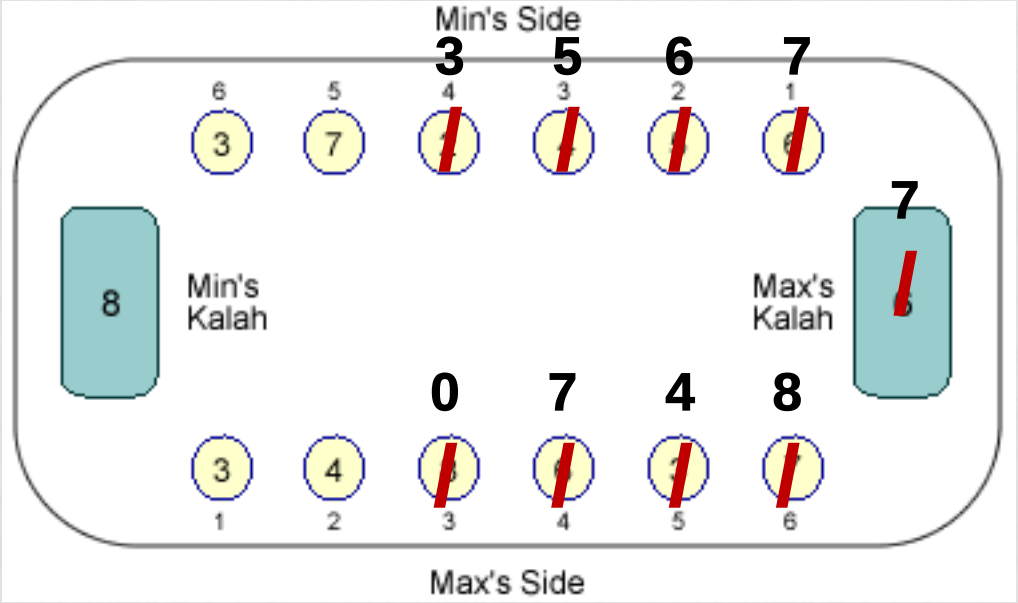
VARIATION #2 – *CAPTURE MOVE*

If final stone falls in one of opponent's pits

and

there are either <u>two or three stones</u> in the pit:

- the stones are *captured*, and placed in moving player's *Kalah*.
- Whenever one pit is captured, the preceding pit may be captured if it contains two or three stones as well.



Max's turn to move.
Max will choose to move from Pit 3. By sowing the last stone in Min's Pit 4, Max will capture the pit, since it will have 3 stones in it.

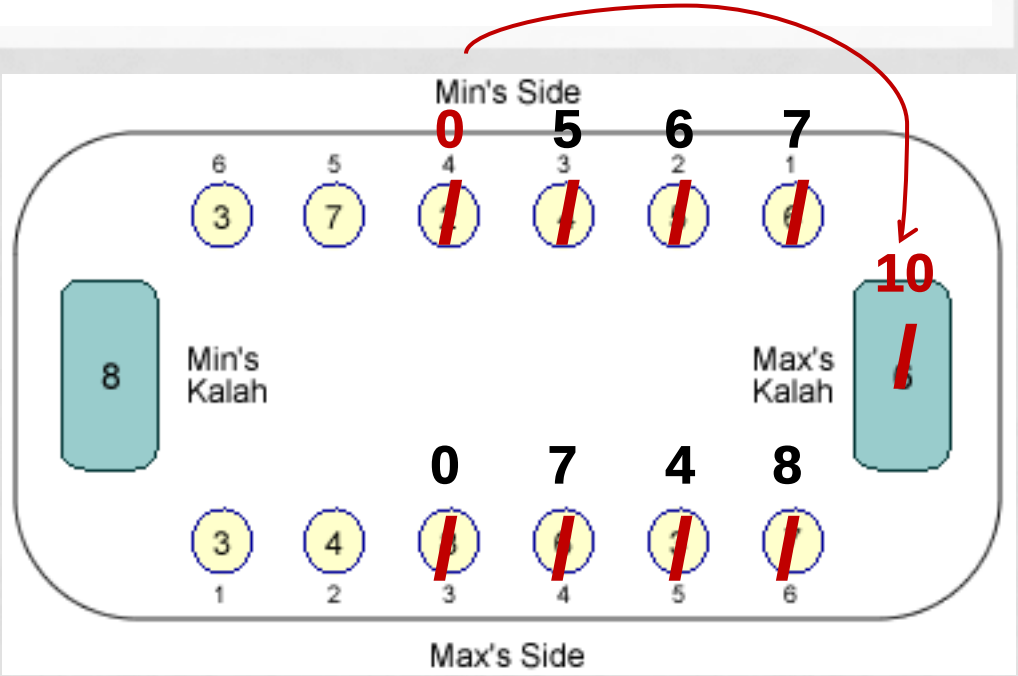
VARIATION #2 – *CAPTURE MOVE*

If final stone falls in one of opponent's pits

and

there are either <u>two or three stones</u> in the pit:

- the stones are *captured*, and placed in moving player's *Kalah*.
- Whenever one pit is captured, the preceding pit may be captured if it contains two or three stones as well.



Max has captured Min's Pit 4. Max's Kalah increased by 4 stones - one from sowing and three from the capture.

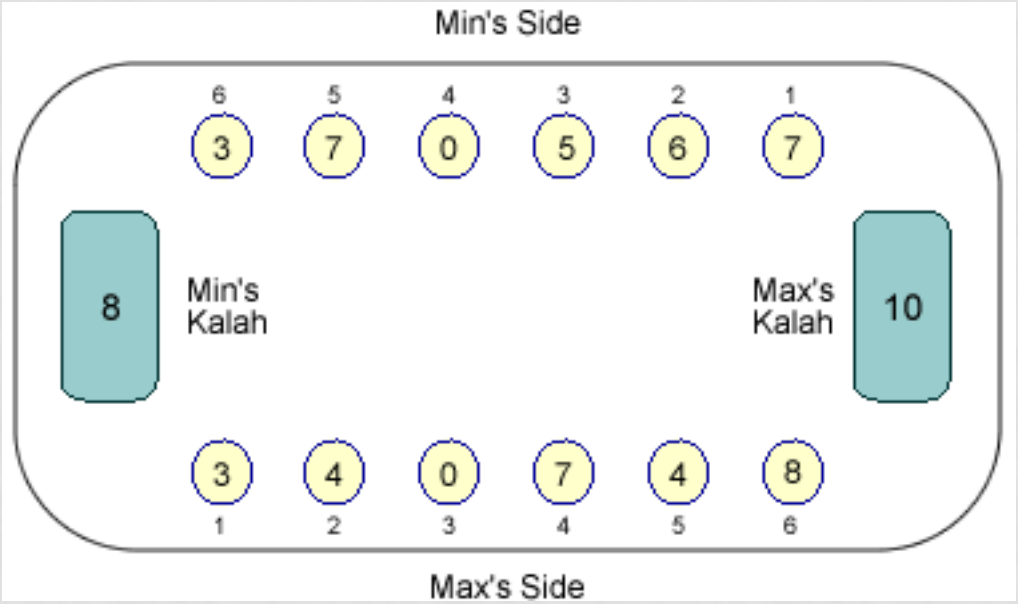
VARIATION #2 – *CAPTURE MOVE*

If final stone falls in one of opponent's pits

and

there are either <u>two or three stones</u> in the pit:

- the stones are *captured*, and placed in moving player's *Kalah*.
- Whenever one pit is captured, the preceding pit may be captured if it contains two or three stones as well.



Max has captured Min's Pit 4. Max's Kalah increased by 4 stones - one from sowing and three from the capture.

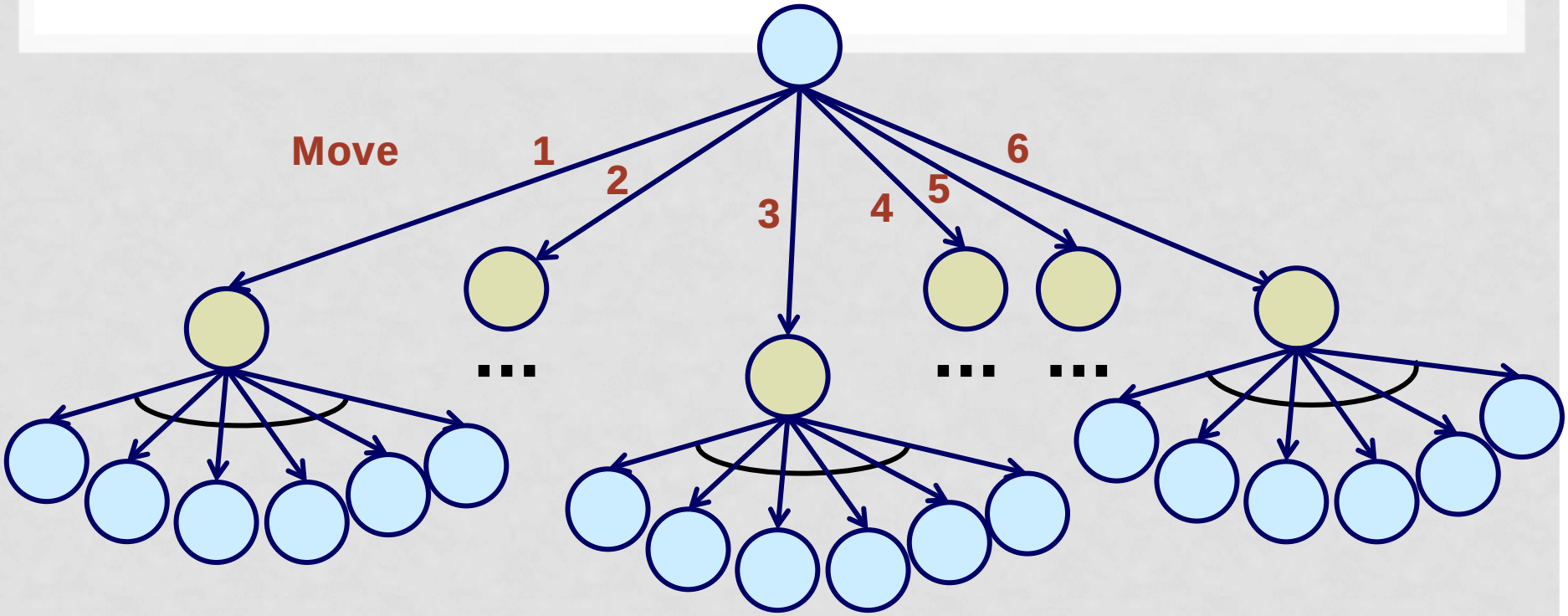
KALAH – GAME OVER

The game is over as soon as more than half the stones are in one player's *Kalah*.

(notice that once a stone enters a Kalah, it can never leave)

Excerpted from "Etudes for Programmers", Charles Wetherell, Prentice Hall, 1978.

MINIMAX AND KALAH



Move *i* : Player removes stones from Pit *i*, and sows them accordingly.
Precondition: Pit *i* is nonempty

For 1 move lookahead, ~42 node expansions, 36 node evaluations.

MINIMAX AND KALAH

IDEA: Let *h(node)* = heuristic that evaluates board quality at bottom level, used by Minimax to backup value

Let *s(node)* = heuristic used to evaluate internal nodes for perceived quality when deciding what order to evaluate them in.

e.g.

h() = (#stones in our Kalah) - (#stones in opponent's Kalah) + α [(#stones in our 6 pits) - (#stones in opponent's pits))

for some α

MINIMAX AND KALAH

Another idea:

As # stones in Kalah increases, perhaps the value should increase more than linearly --

There are 72 total stones in game, so need 36+1=37 to win.

So if we have 13 and opponent has 11, h() = 13-11 = 2.

But if we have 33 and opponent has 31, h() = 33-31 = 2.

Is this value more **urgent** as values get closer to 36?

MINIMAX AND KALAH

Idea for *s(board)* = "rank" $\in$ {0, 1, 2, 3}

Rank(i) = **0** if #stones is not sufficient to cause a wraparound - or a stone in our Kalah
[NumInPit(i) + i $\leq$ 6]

Rank(i) = **2** if #stones causes wraparound to one of our own pits or beyond –
[NumInPit(i) + i > 2*6+1 = 13]
Kalah increases, potential go-again

Rank(i) = **3** if we end up in opponent's pit
potential capture of opponent's pit, more

MINIMAX AND KALAH

Slightly improved ranking system:

Rank(i) = **0** if **[NumInPit(i) + i ≤ 6]** *(stay in our own pits)*

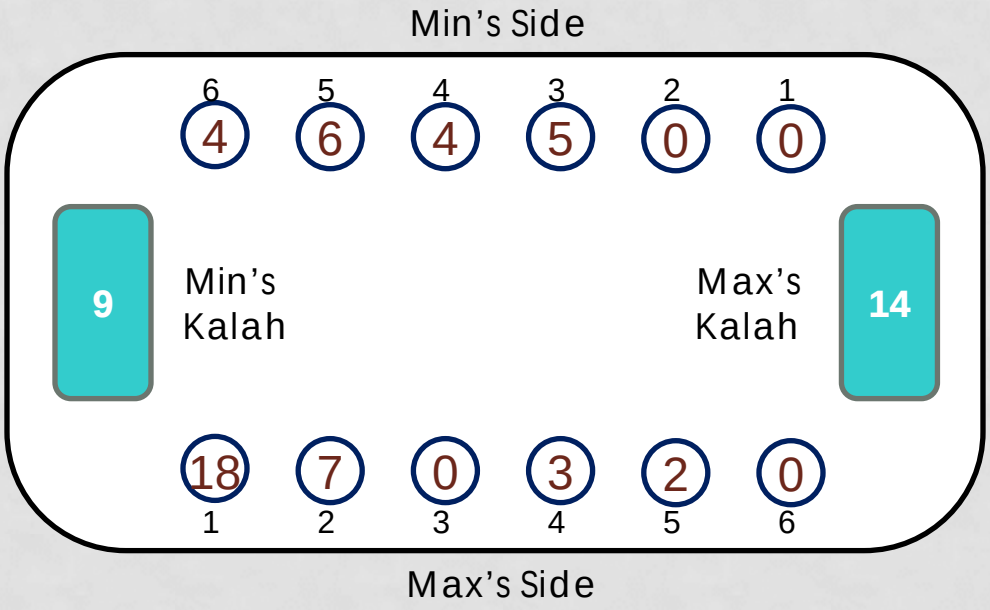
Rank(i) = **1** if **[NumInPit(i) + i = 7]** *(land in our Kalah)*

Rank(i) = **(NumInPit(i) + i)/13** *(int division)*
if **[NumInPit(i)%13 + i ≤ 6]**
*(#stones added to our Kalah if we finish move in one of our own pits – accounts for double wraparounds – will be **1** or **2** or **3**…)*

Rank(i) = **3** otherwise *(land in opponent's pits)*

Recall that the order in which moves are considered is important when doing α-β pruning

EXAMPLE

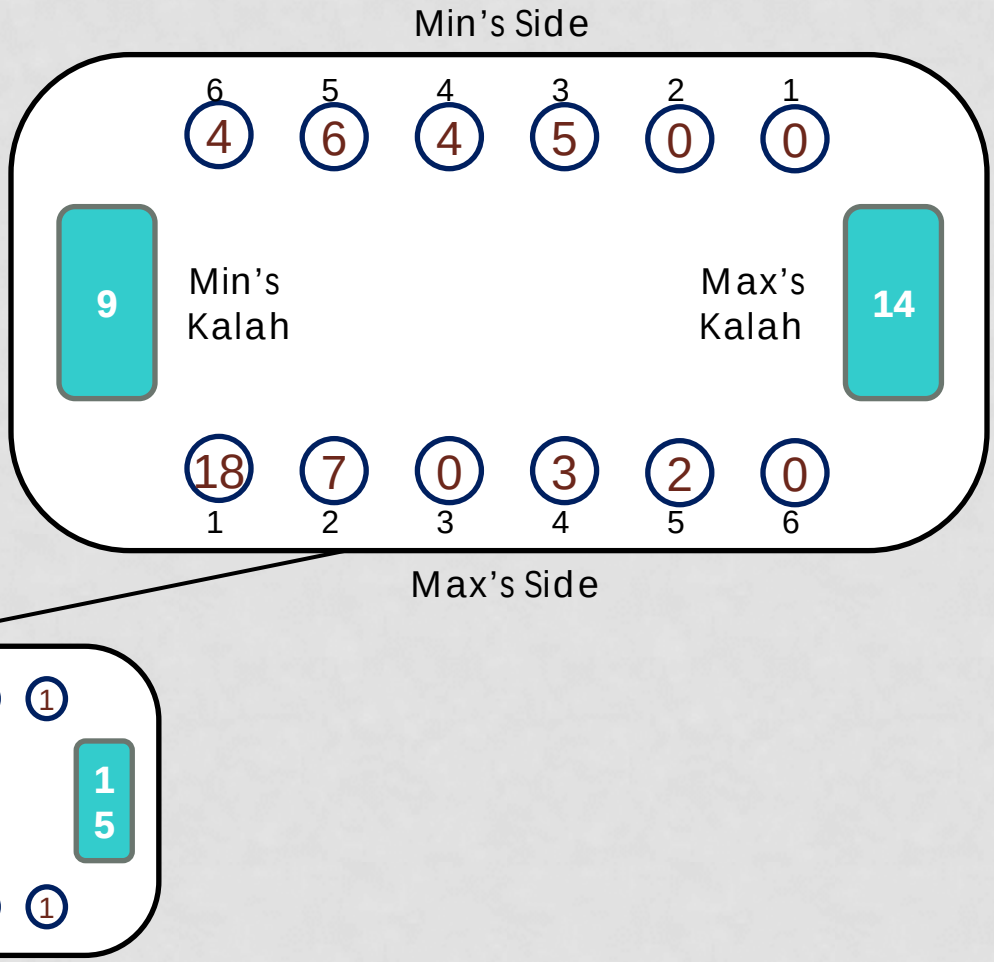


Our choices: **1**, **2**, **4**, **5**

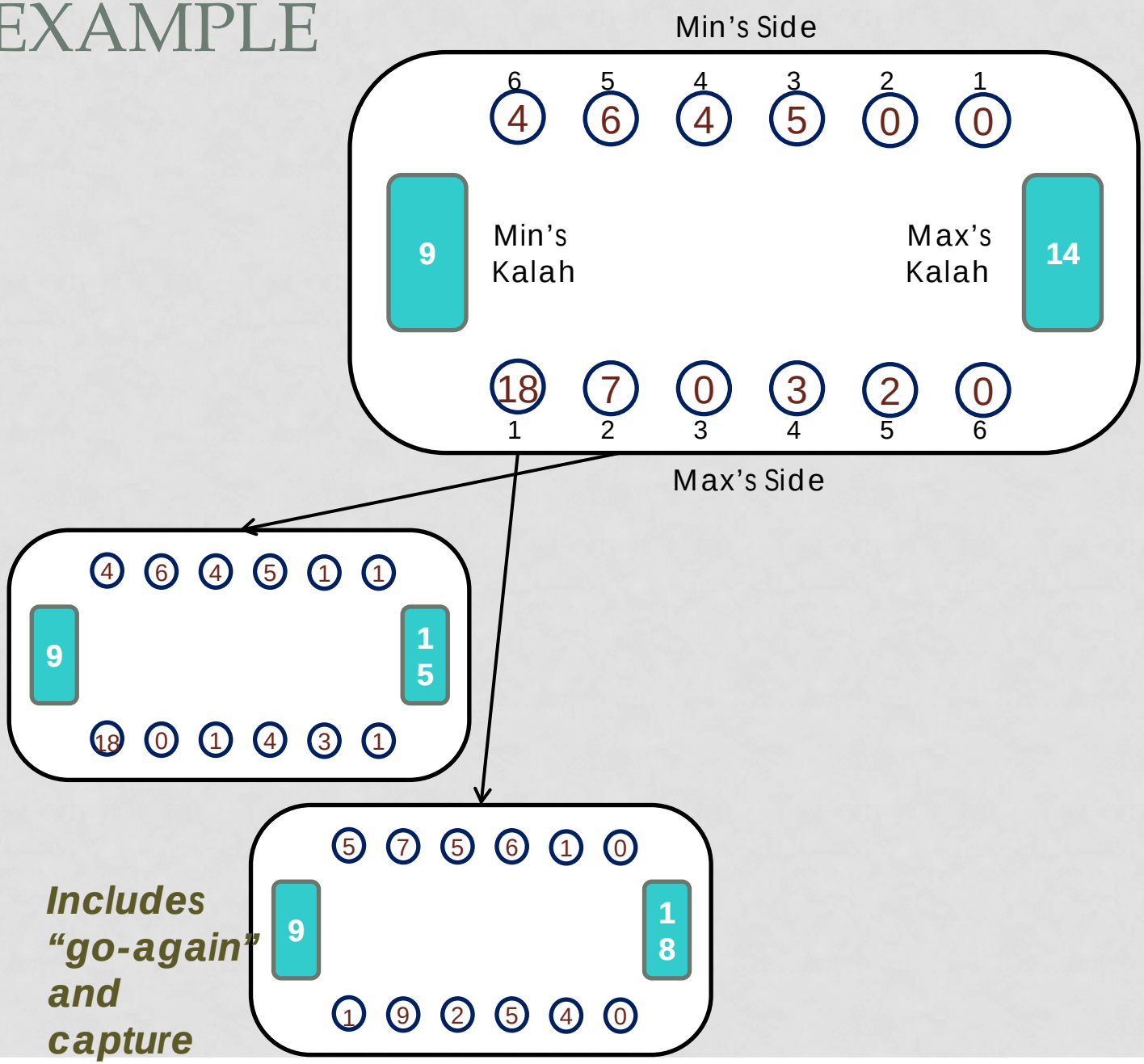
Move(1): **Rank=2**
Move(2): **Rank=3**
Move(4): **Rank=1**
Move(5): **Rank=1**

Evaluate in order **2**, **1**, **4**, **5**

EXAMPLE

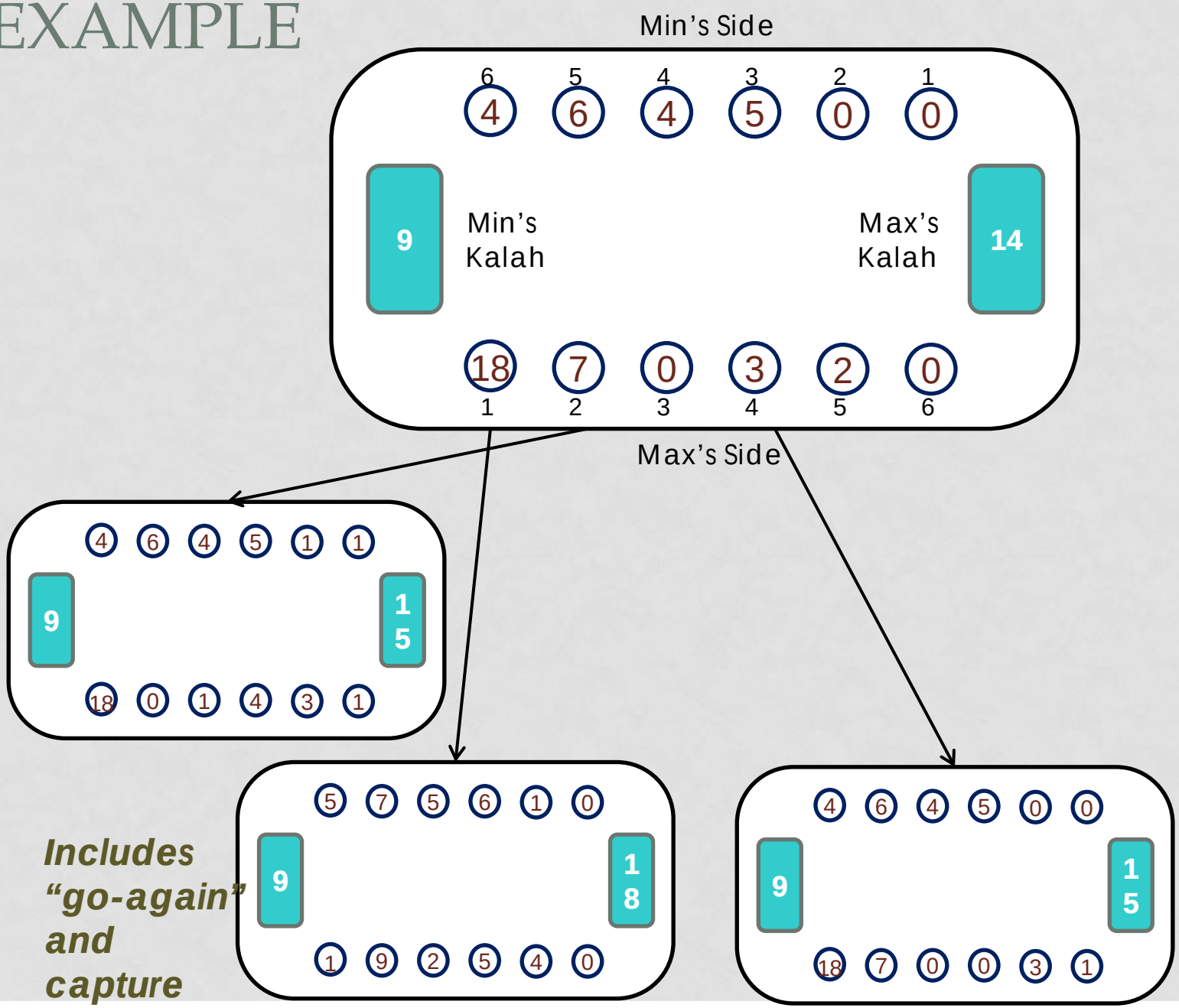

Min's Side
6 5 4 3 2 1
4 6 4 5 0 0
9
Min's Kalah
Max's Kalah
14
18 7 0 3 2 0
1 2 3 4 5 6
Max's Side
4 6 4 5 1 1
9
15
18 0 1 4 3 1

EXAMPLE



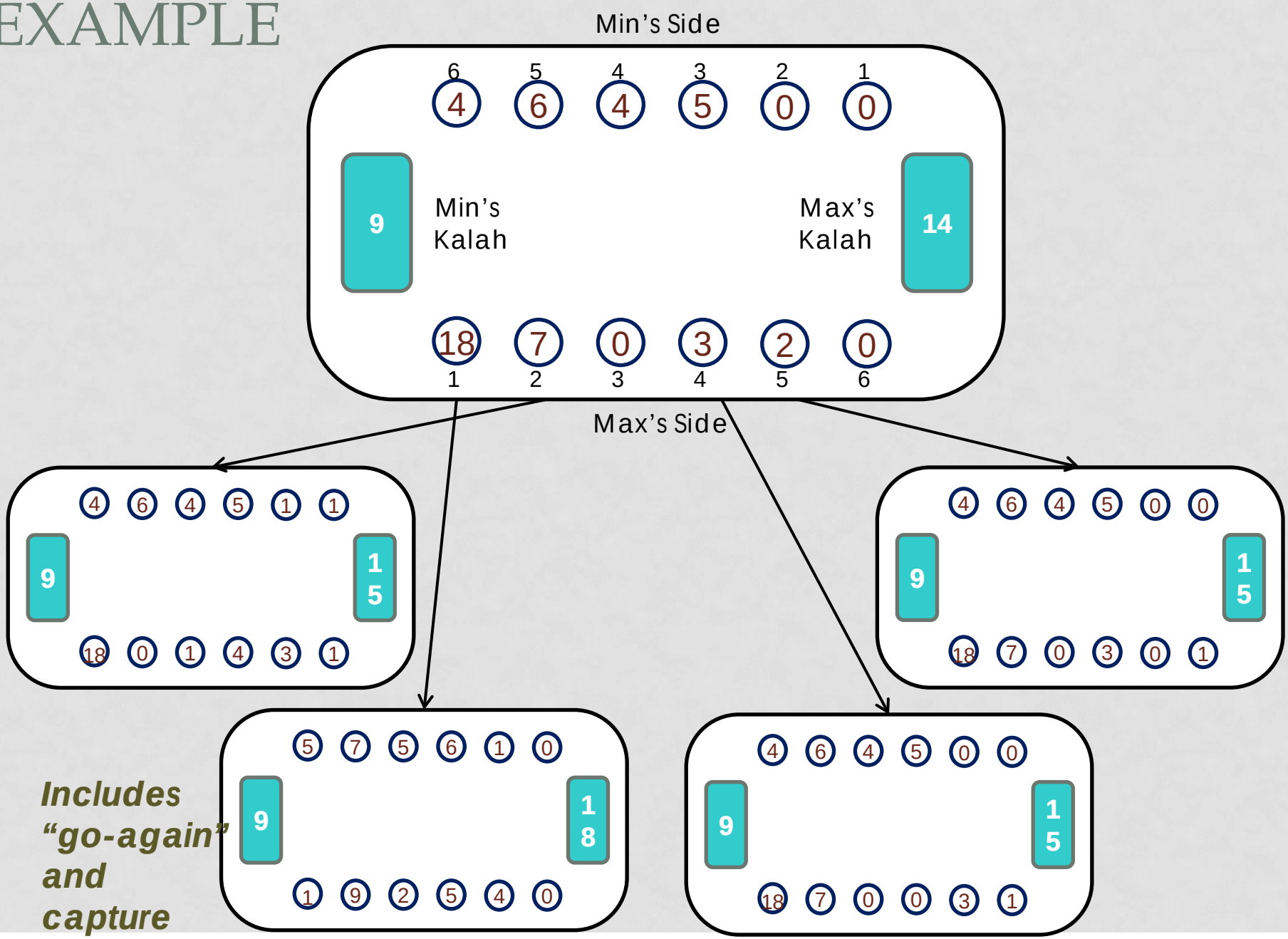
Includes "go-again" and capture

EXAMPLE



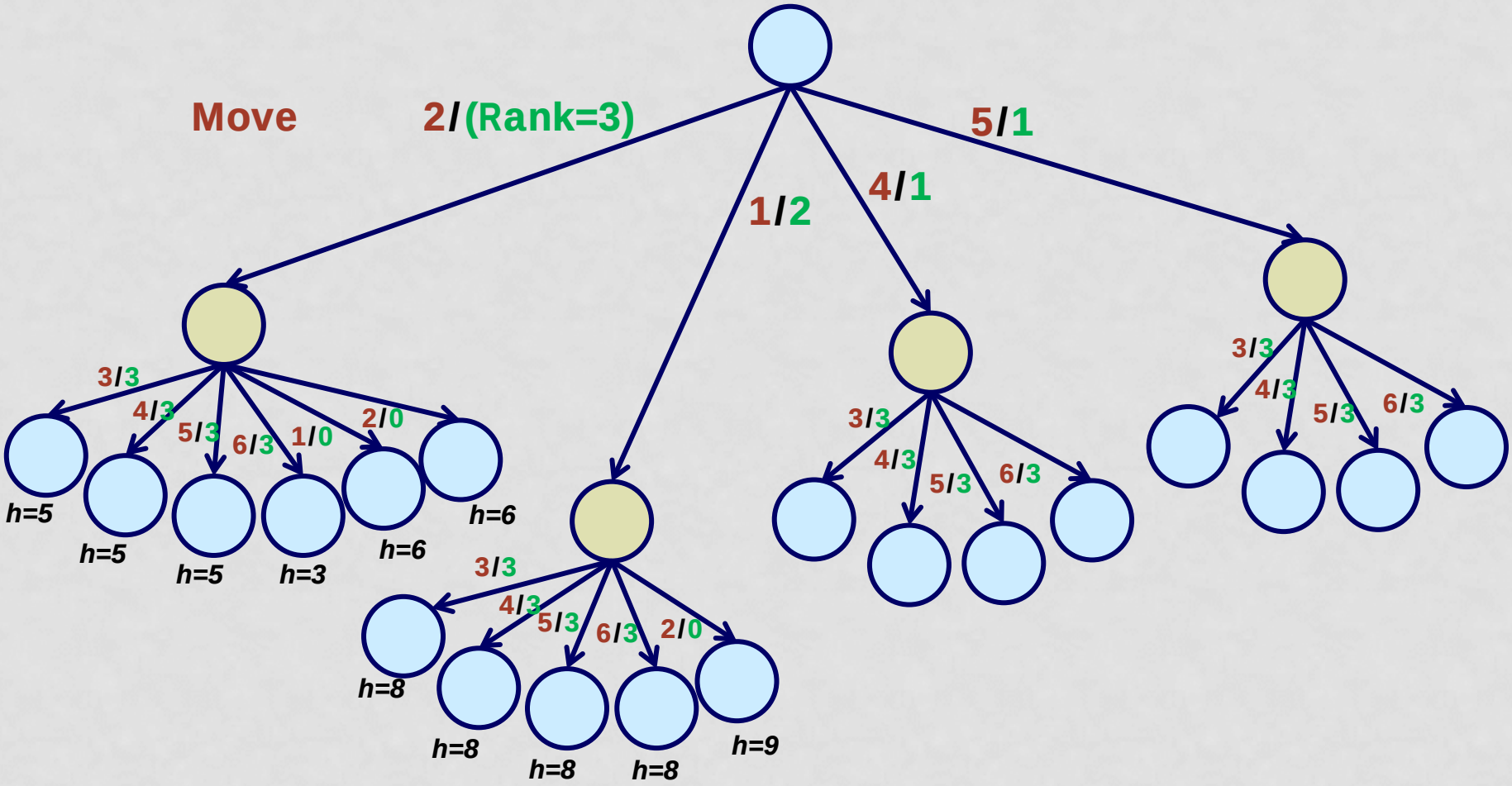
Includes "go-again" and capture

EXAMPLE

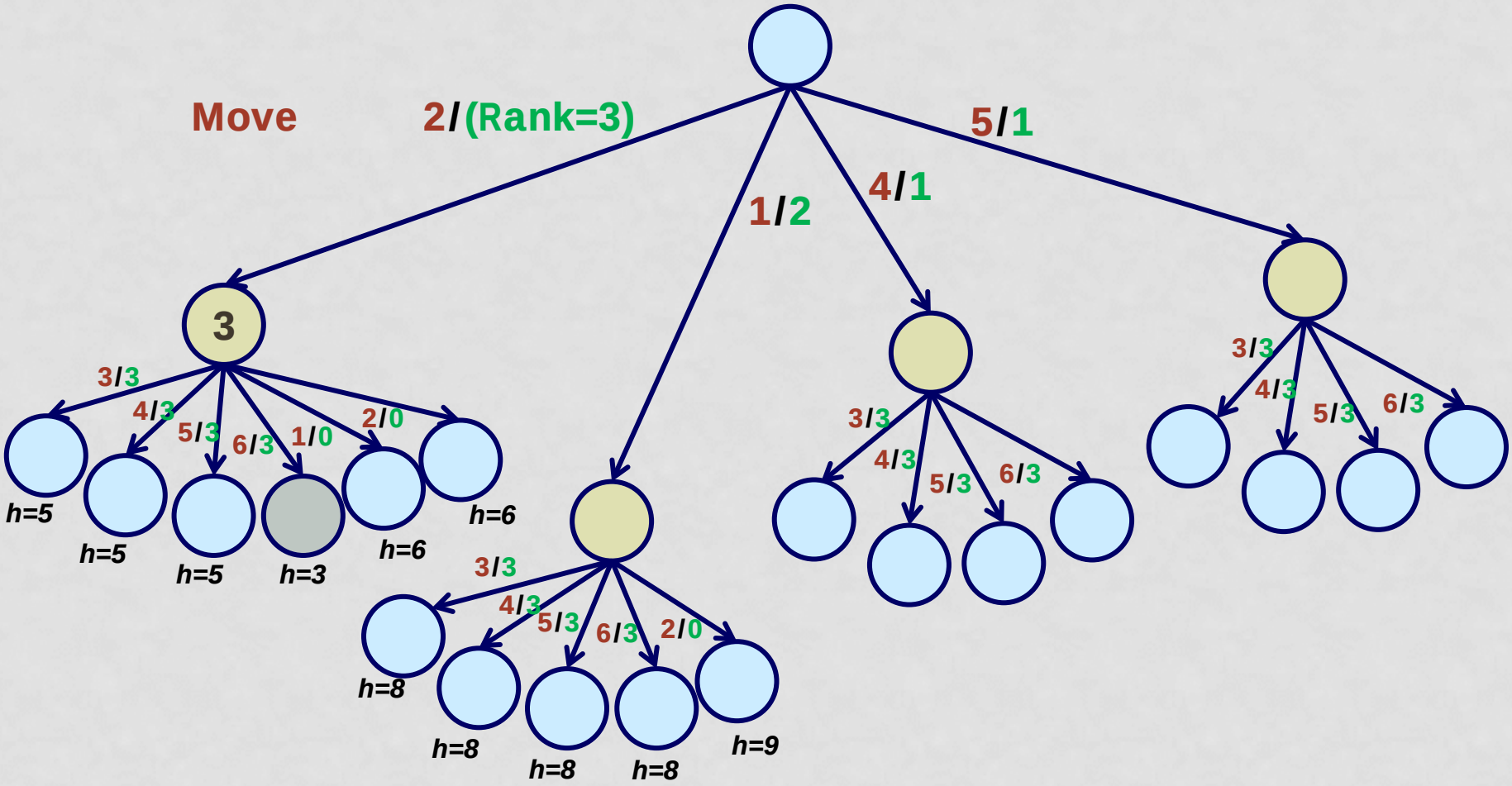


Includes "go-again" and capture

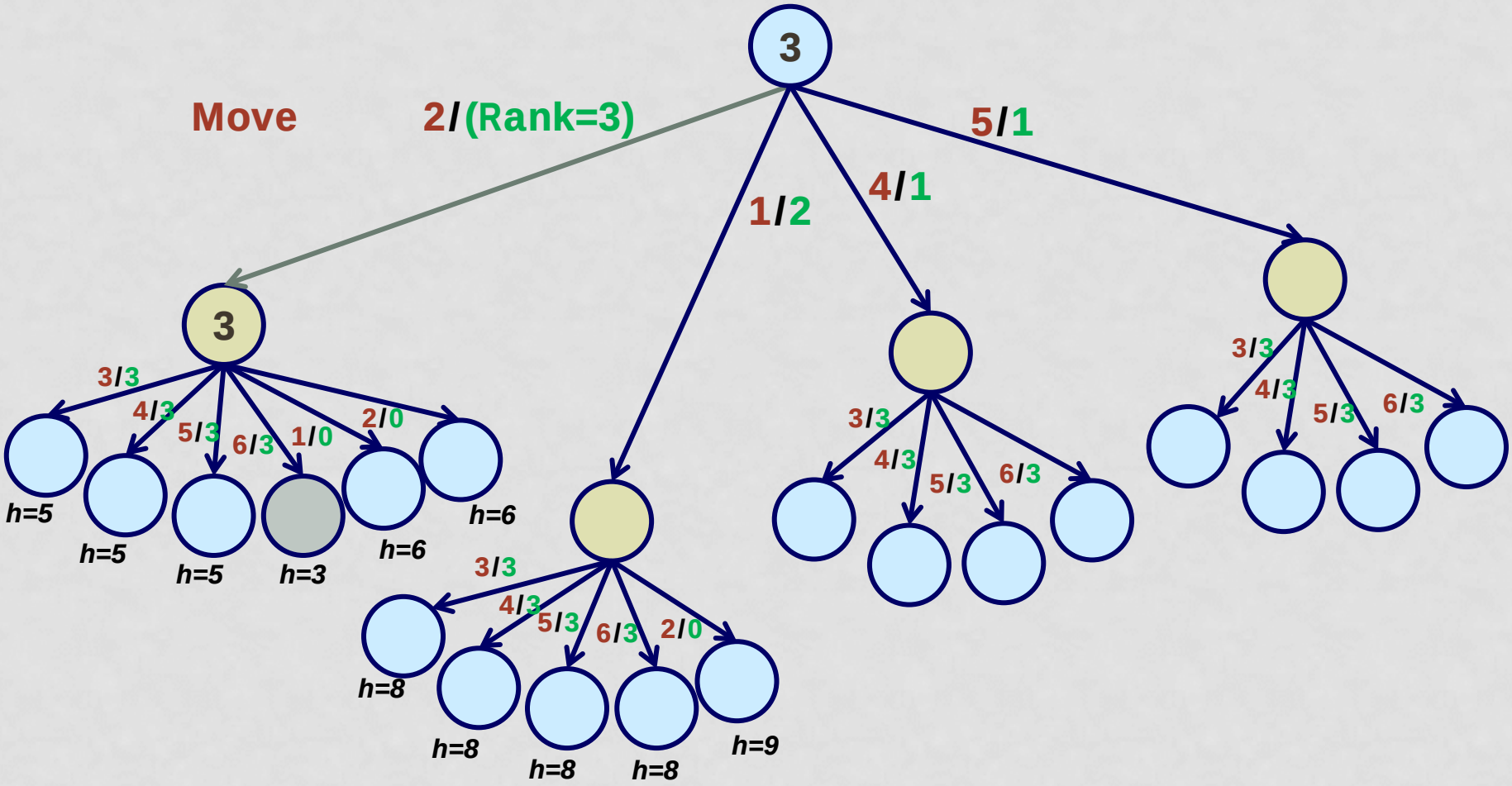
EXAMPLE


Move
2/(Rank=3)
1/2
4/1
5/1
3/3
4/3
5/3
6/3
1/0
2/0
h=5
h=5
h=5
h=3
h=6
h=6
3/3
4/3
5/3
6/3
2/0
h=8
h=8
h=8
h=8
h=9
3/3
4/3
5/3
6/3
3/3
4/3
5/3
6/3

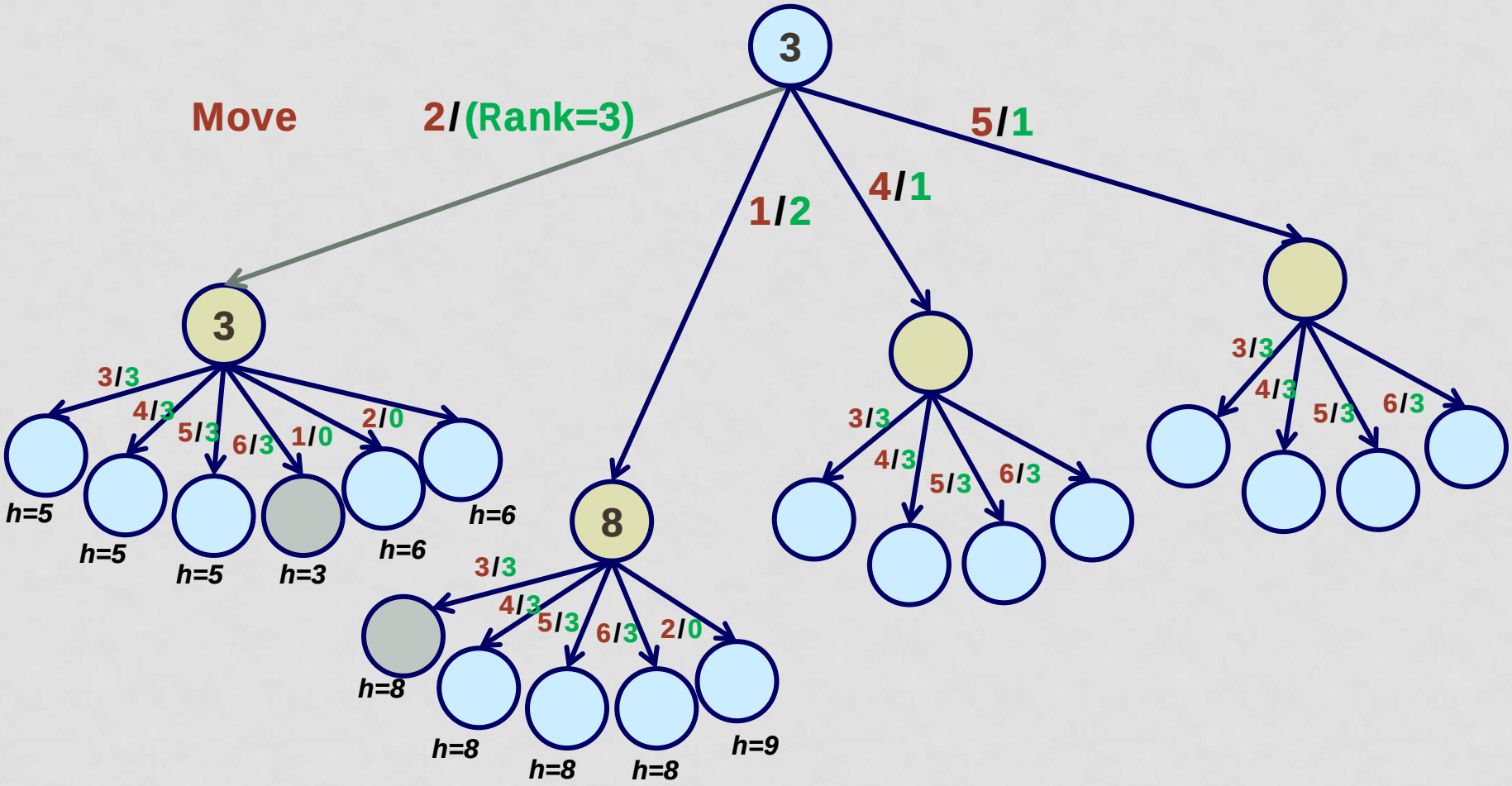
EXAMPLE


Move
2/(Rank=3)
1/2
4/1
5/1
3
3/3
4/3
5/3
6/3
1/0
2/0
h=5
h=5
h=5
h=3
h=6
h=6
3/3
4/3
5/3
6/3
2/0
h=8
h=8
h=8
h=8
h=9
3/3
4/3
5/3
6/3
3/3
4/3
5/3
6/3

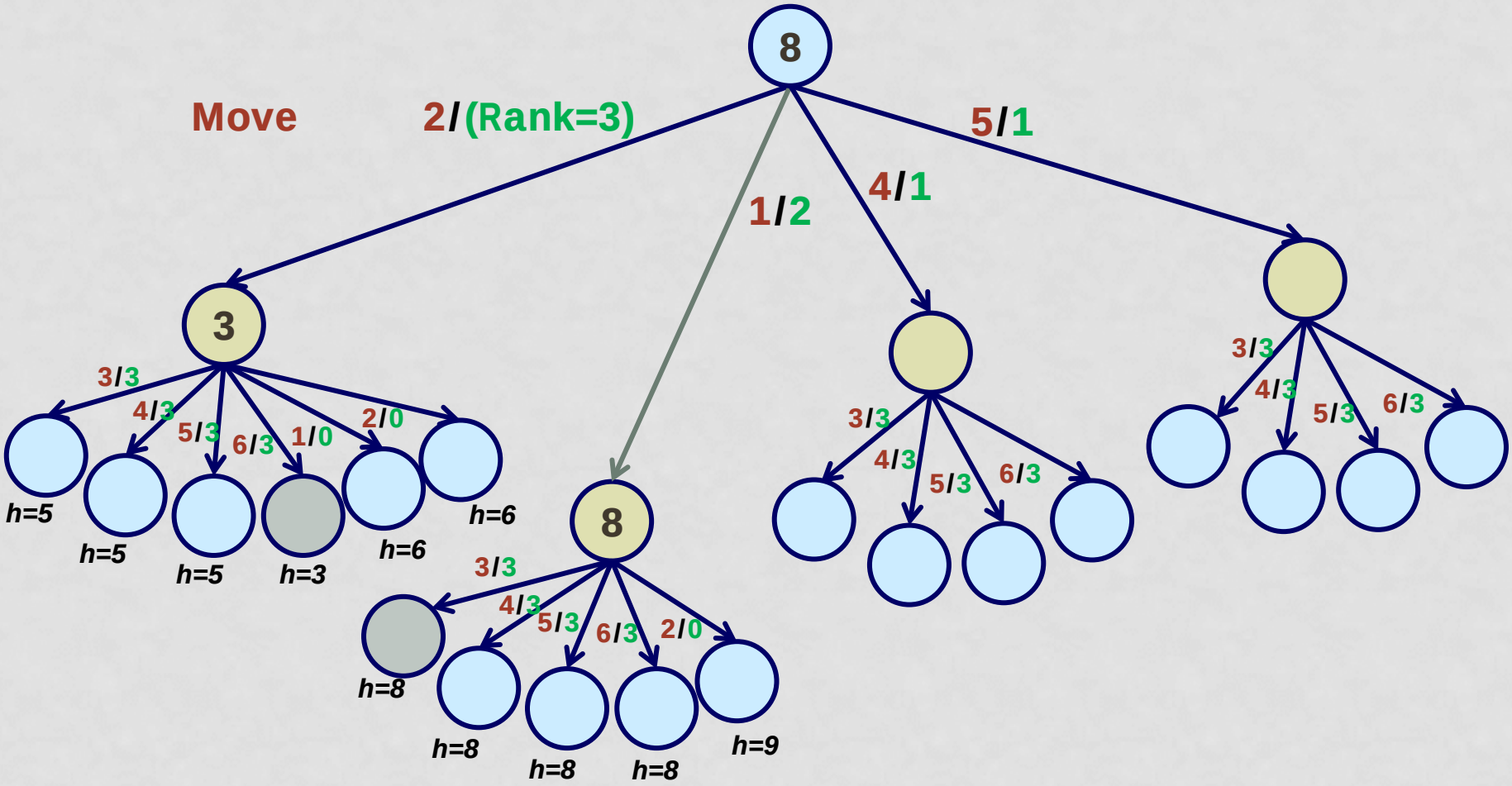
EXAMPLE


3
Move
2/(Rank=3)
1/2
4/1
5/1
3
3/3
4/3
5/3
6/3
1/0
2/0
h=5
h=5
h=5
h=3
h=6
h=6
3/3
4/3
5/3
6/3
3/3
4/3
5/3
6/3
2/0
h=8
h=8
h=8
h=8
h=9

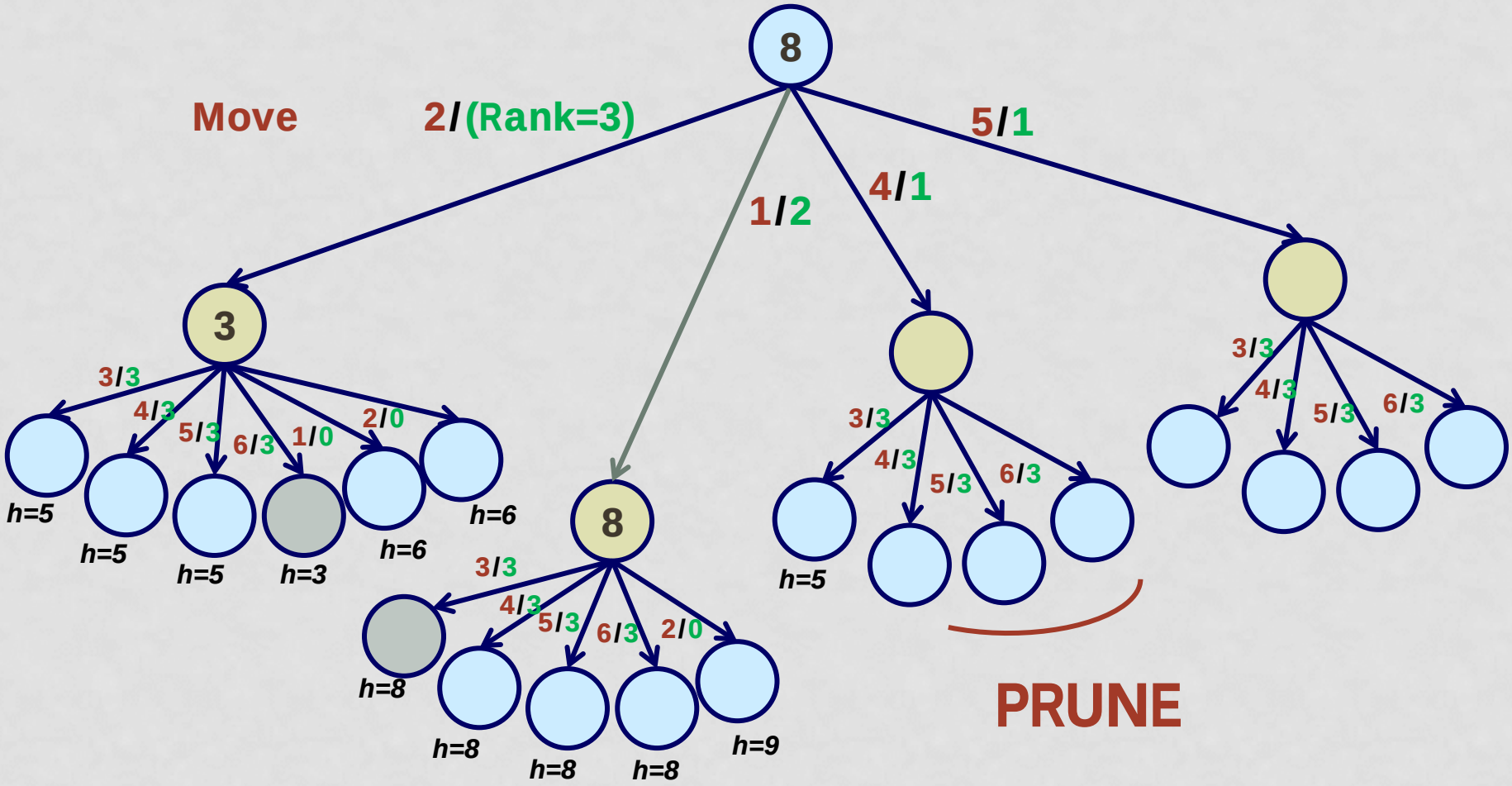
EXAMPLE


3
Move
2/(Rank=3)
1/2
4/1
5/1
3
3/3
4/3
5/3
6/3
1/0
2/0
h=5
h=5
h=5
h=3
h=6
h=6
8
3/3
4/3
5/3
6/3
2/0
h=8
h=8
h=8
h=8
h=9
3/3
4/3
5/3
6/3
3/3
4/3
5/3
6/3

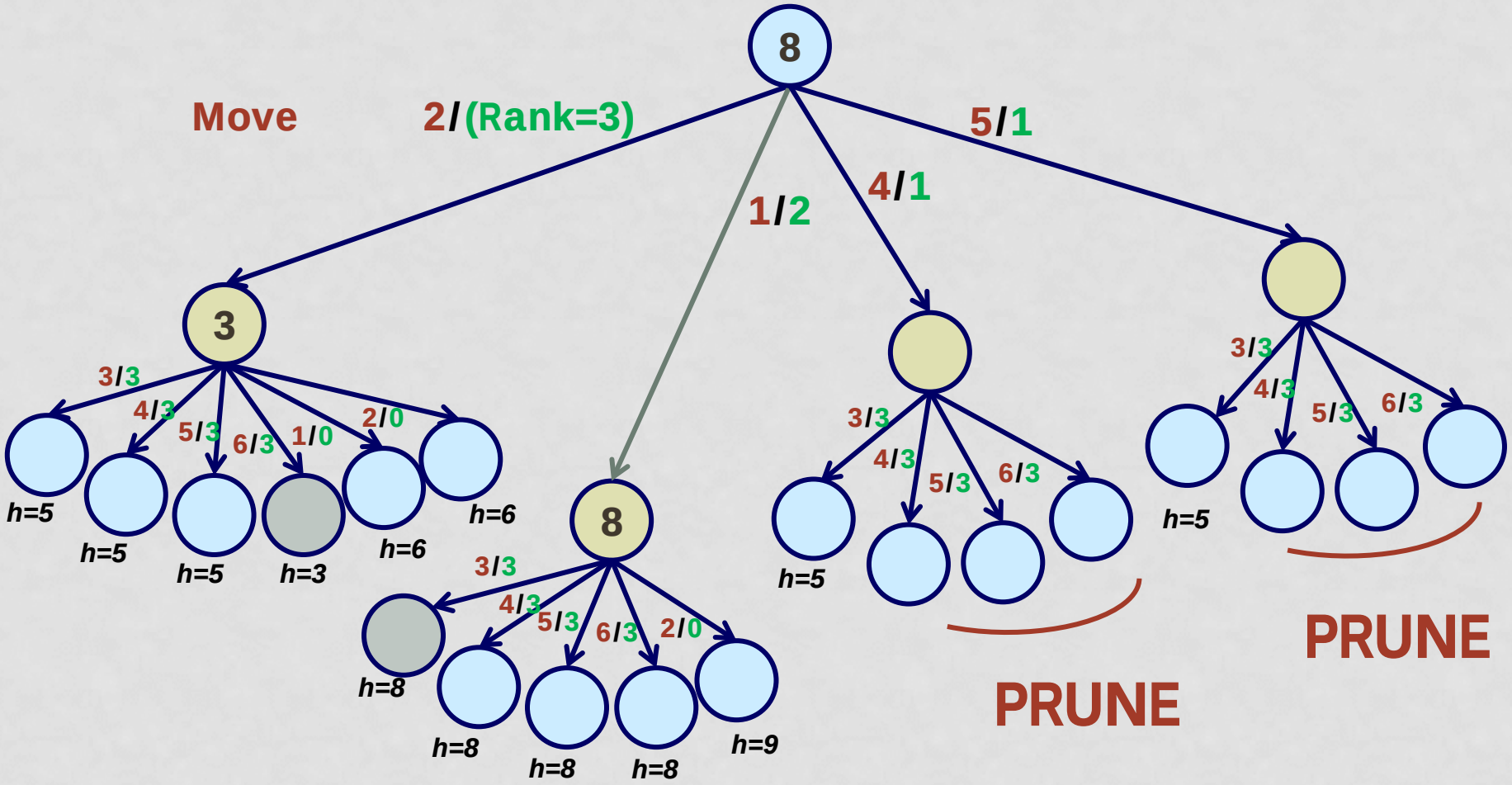
EXAMPLE


8
Move
2/(Rank=3)
1/2
4/1
5/1
3
3/3
4/3
5/3
6/3
1/0
2/0
h=5
h=5
h=5
h=3
h=6
h=6
8
3/3
4/3
5/3
6/3
2/0
h=8
h=8
h=8
h=8
h=9
3/3
4/3
5/3
6/3
3/3
4/3
5/3
6/3

EXAMPLE


8
Move
2/(Rank=3)
1/2
4/1
5/1
3
3/3
4/3
5/3
6/3
1/0
2/0
h=5
h=5
h=5
h=3
h=6
h=6
8
3/3
4/3
5/3
6/3
2/0
h=8
h=8
h=8
h=8
h=9
3/3
4/3
5/3
6/3
h=5
3/3
4/3
5/3
6/3
PRUNE

EXAMPLE


8
Move
2/(Rank=3)
1/2
4/1
5/1
3
3/3
4/3
5/3
6/3
1/0
2/0
h=5
h=5
h=5
h=3
h=6
h=6
8
3/3
4/3
5/3
6/3
2/0
h=8
h=8
h=8
h=8
h=9
3/3
4/3
5/3
6/3
h=5
PRUNE
3/3
4/3
5/3
6/3
h=5
PRUNE

SAMPLE GAME

Level of Recursion=1
(1move per player)

Player Move	Computer Move	Backed Up Value	Nodes Evaluated	Node evals w/Pruning
2	1	-1	41	17
1	2	1	41	17
3	3	1	39	18
4	4	0	40	18
5	6	1	39	23
6	1	1	43	18
1	1	0	41	29
2	2	0	39	18
3	3	0	41	18
4	5	0	40	18
5	6	0	39	18
6	5	0	33	16
2	1	0	29	14
3	4	3	30	24
4	2	4	27	13
5	6	4	23	20
6	3	∞	26	2
		TOTALS	611	301

SAMPLE GAME

Level of Recursion=2
(2 moves per player)

Player Move	Computer Move	Backed Up Value	Nodes Evaluated	Node evals w/Pruning
2	1	-2	1246	375
1	2	1	12898	144
3	4	1	1151	288
4	3	0	1170	343
5	5	1	1107	198
6	6	1	1227	308
1	1	1	1093	217
2	5	3	1022	313
3	4	2	1017	247
4	2	7	682	189
5	5	10	879	219
6	3	14	660	143
3	4	∞	468	84
4	6	∞	131	54
		TOTALS	13142	3122

RESULTS FROM 2 SAMPLE GAMES

Level of recursion = 1

- total # nodes eval (w/o pruning) - 611
- (w/ pruning) - 356
- (w/re-ordering & pruning) - 301
- >50% fewer node evals with this function and α-β pruning

Level of recursion = 2

- total # nodes eval (w/o pruning) - 13142
- (w/ pruning) - 3122
- (w/re-ordering & pruning) - 3002
- >75% fewer nodes