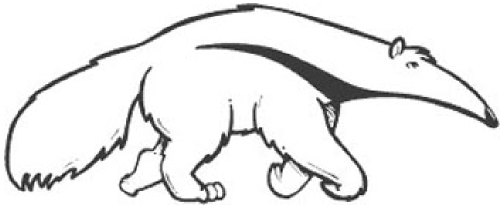
Games & Adversarial Search B: Alpha-Beta Pruning and MCTS

Introduction to Artificial Intelligence

Prof. Richard Lathrop



<u>Read Beforehand:</u> R&N 5.3; Optional: 5.5+

BREN:ICS
INFORMATION AND COMPUTER SCIENCES

UNIVERSITY of CALIFORNIA IRVINE

Alpha-Beta pruning

- Exploit the “fact” of an adversary

- Bad = not better than we already know we can get elsewhere
- If a position is provably bad
  - It’s NO USE expending search effort to find out just how bad it is
- If the adversary can force a bad position
  - It’s NO USE searching to find the good positions the adversary won’t let you achieve anyway

- Contrast normal search:
  - ANY node might be a winner, so ALL nodes must be considered.
  - A* avoids this through heuristics that transmit your knowledge.
  - Alpha-Beta pruning avoids this through exploiting the adversary.

Pruning with Alpha/Beta

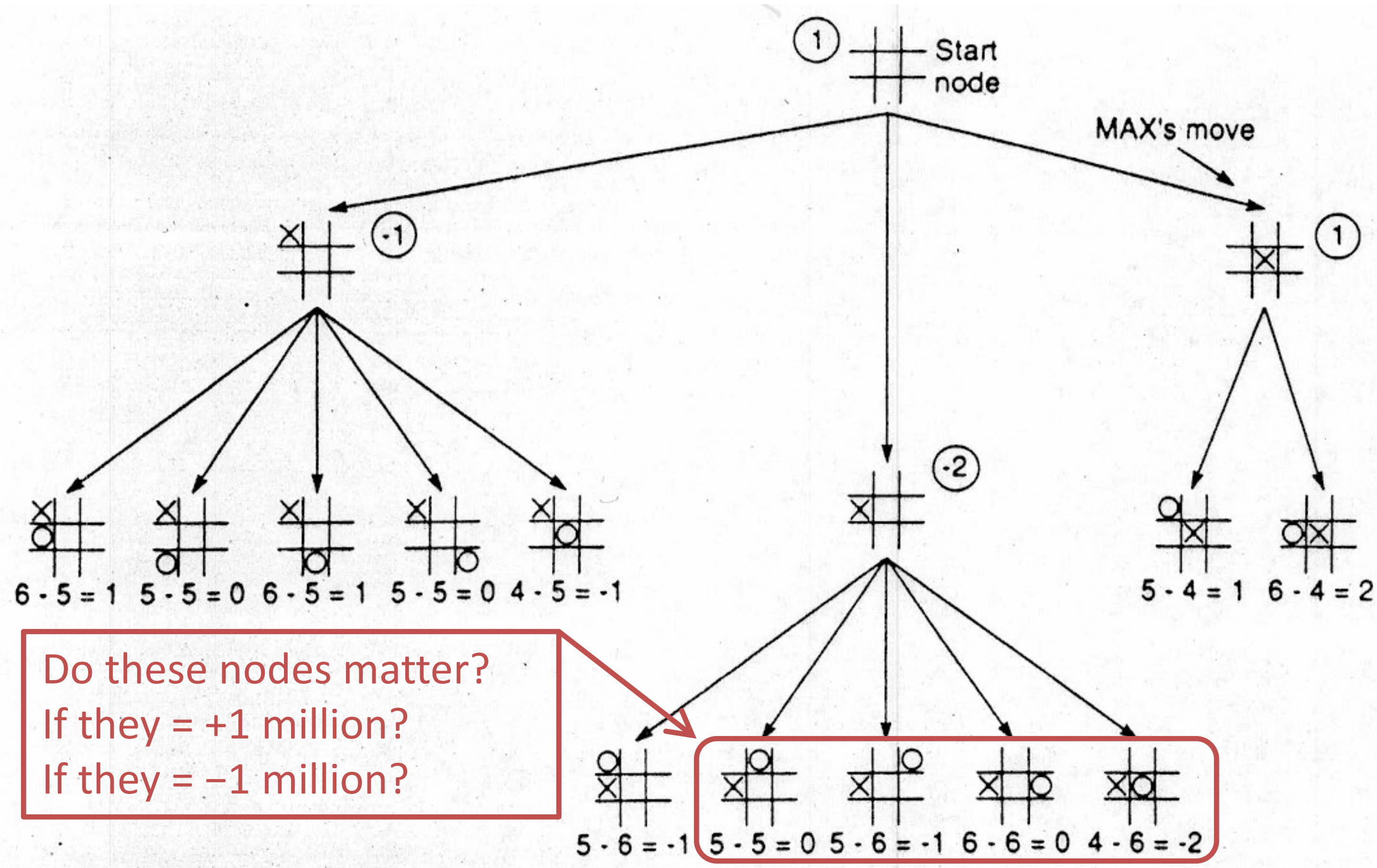
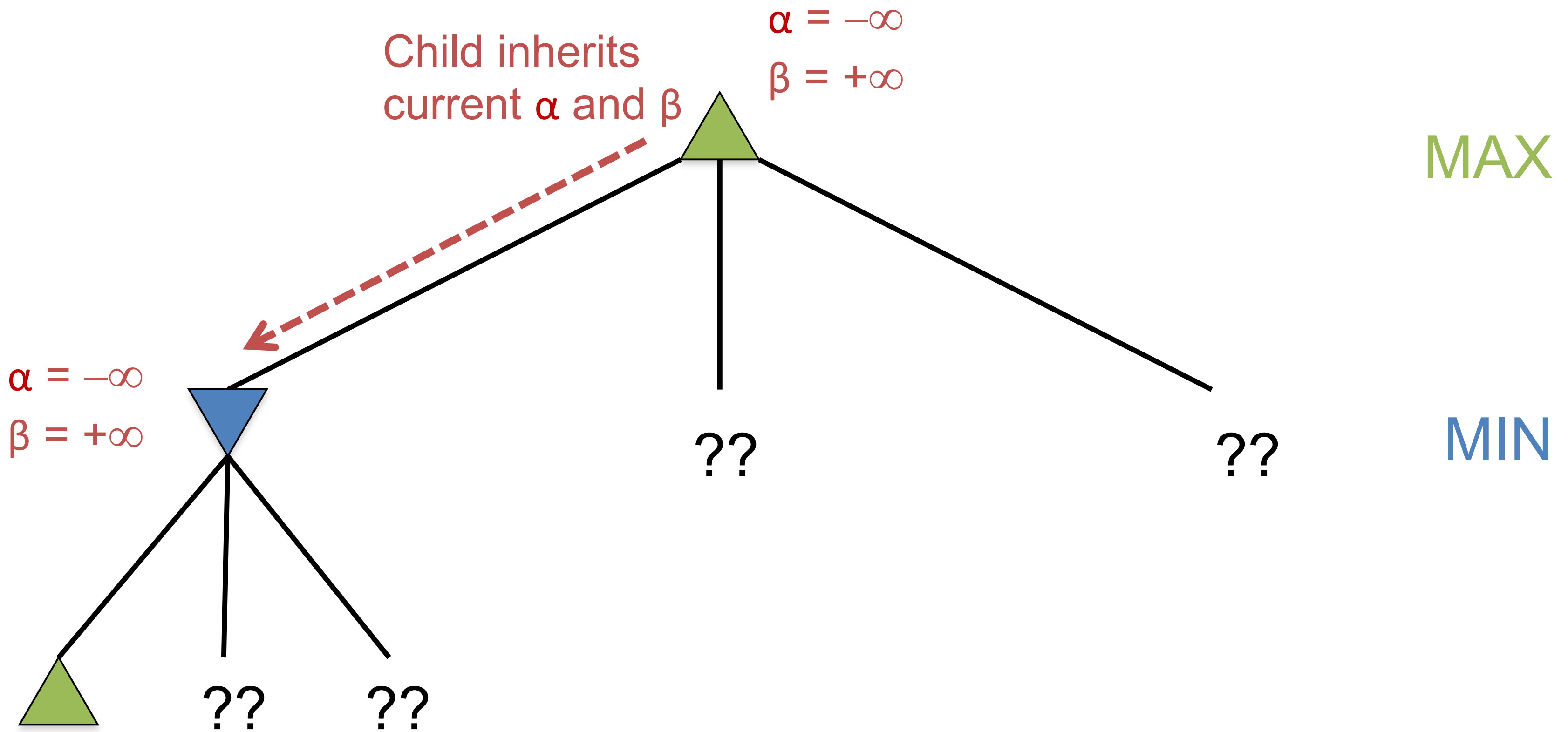


Figure 4.17 Two-ply minimax applied to the opening move of tic-tac-toe.

Alpha-Beta Example

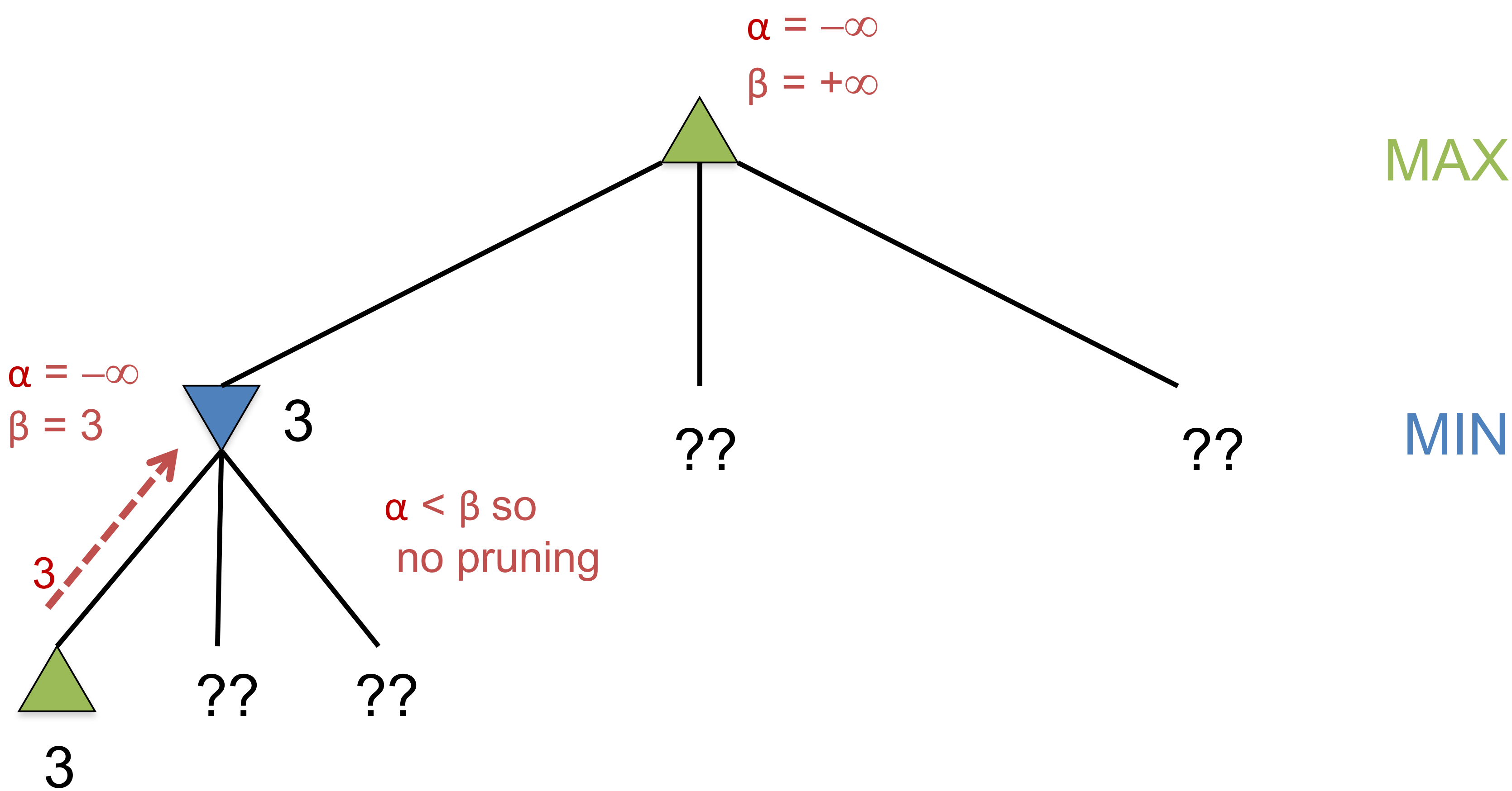
Initially, possibilities are unknown: range ($\alpha = -\infty$, $\beta = +\infty$)

Do a depth-first search to the first leaf.



Alpha-Beta Example

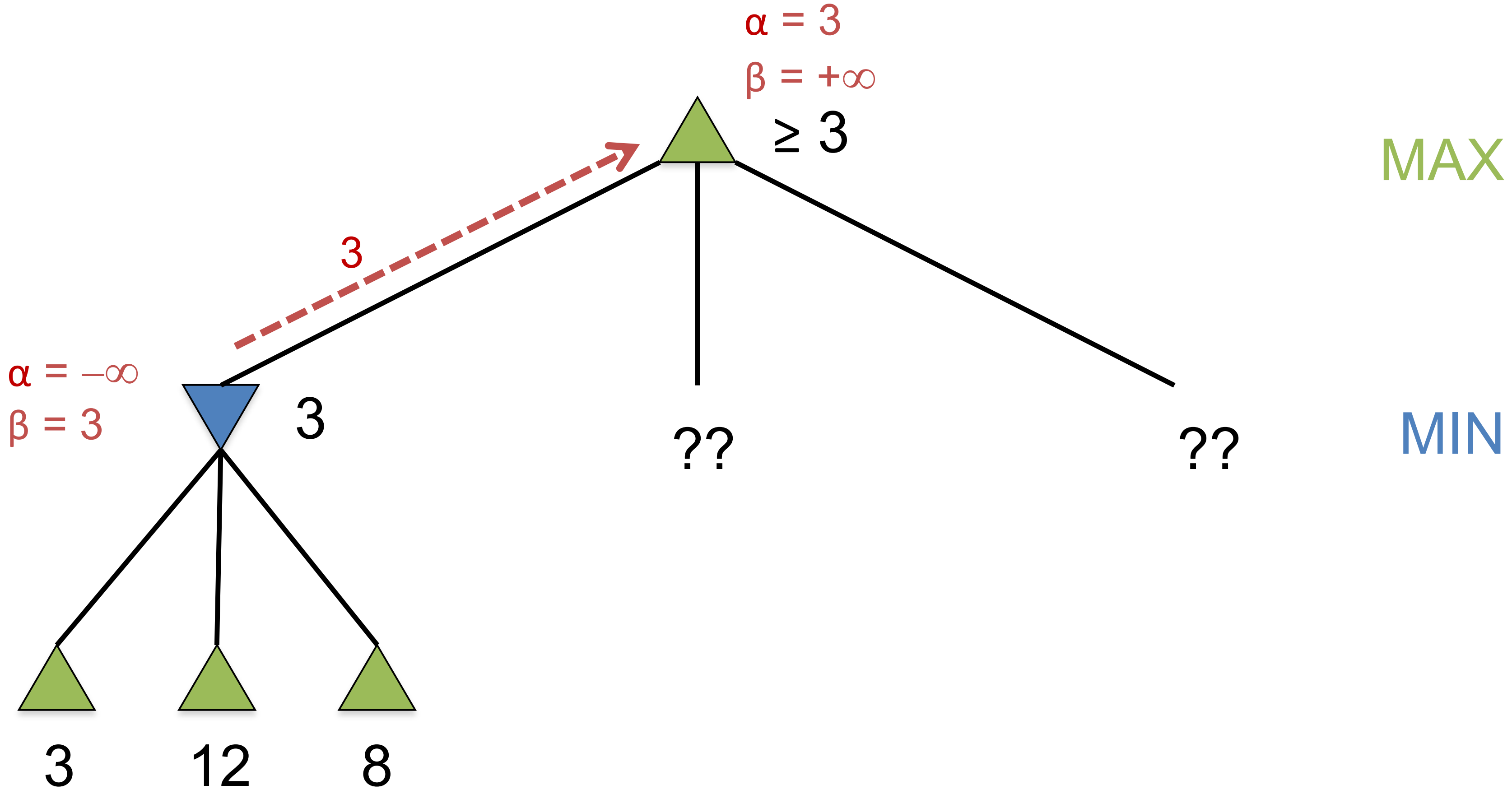
See the first leaf, after MIN's move: MIN updates β



Alpha-Beta Example

See remaining leaves; value is known

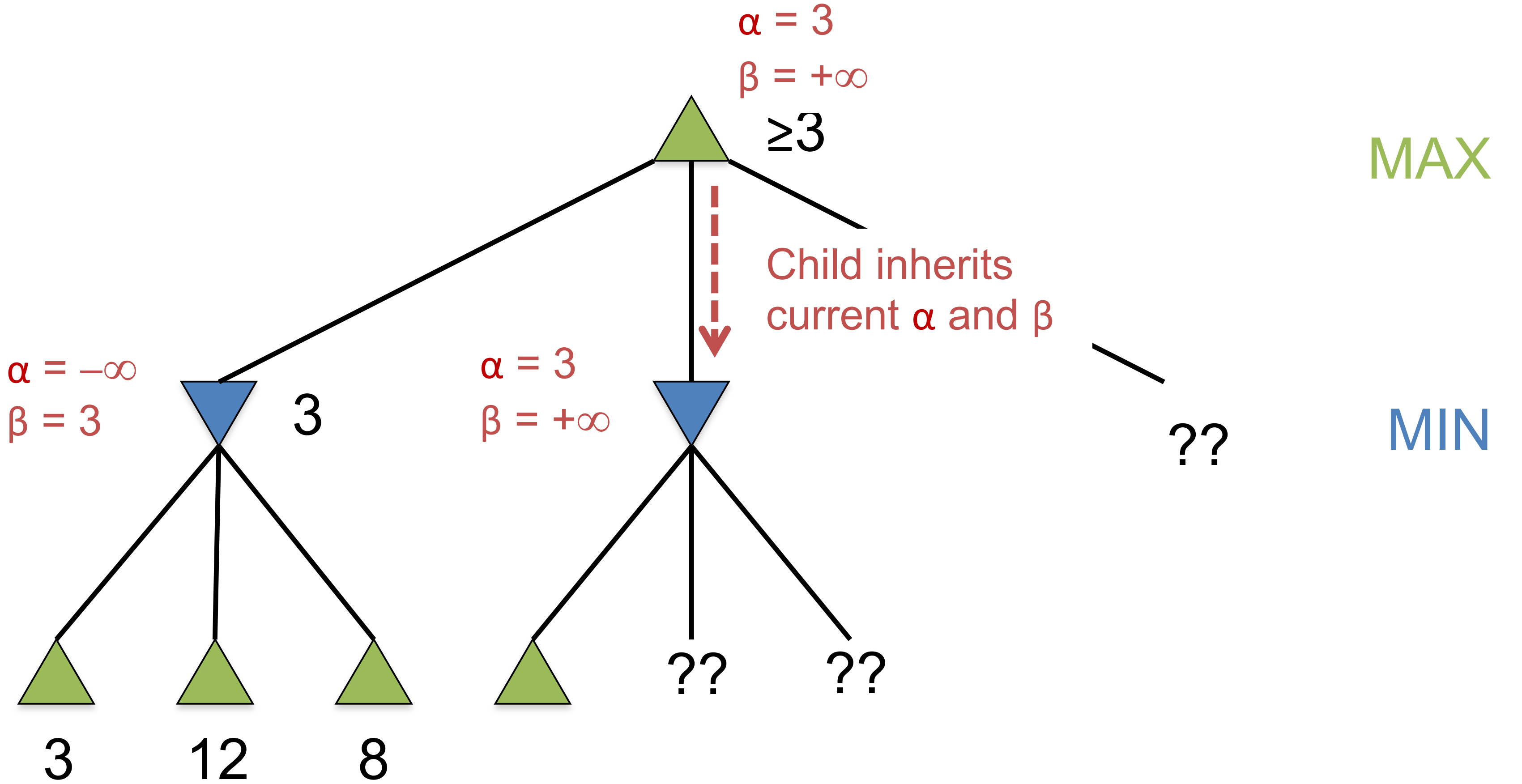
Pass outcome to caller; MAX updates α



Alpha-Beta Example

Continue depth-first search to next leaf.

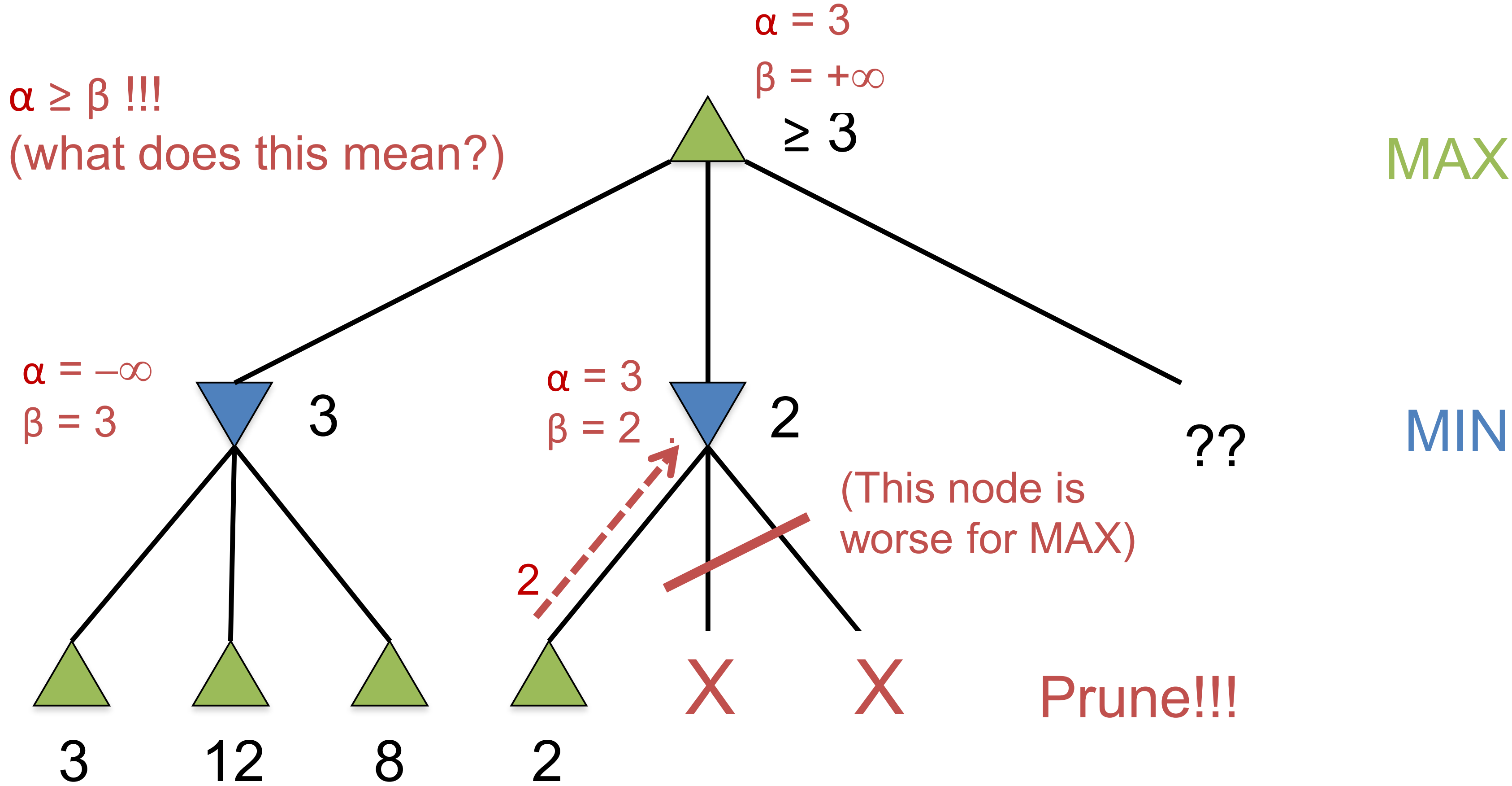
Pass α, β to descendants



Alpha-Beta Example

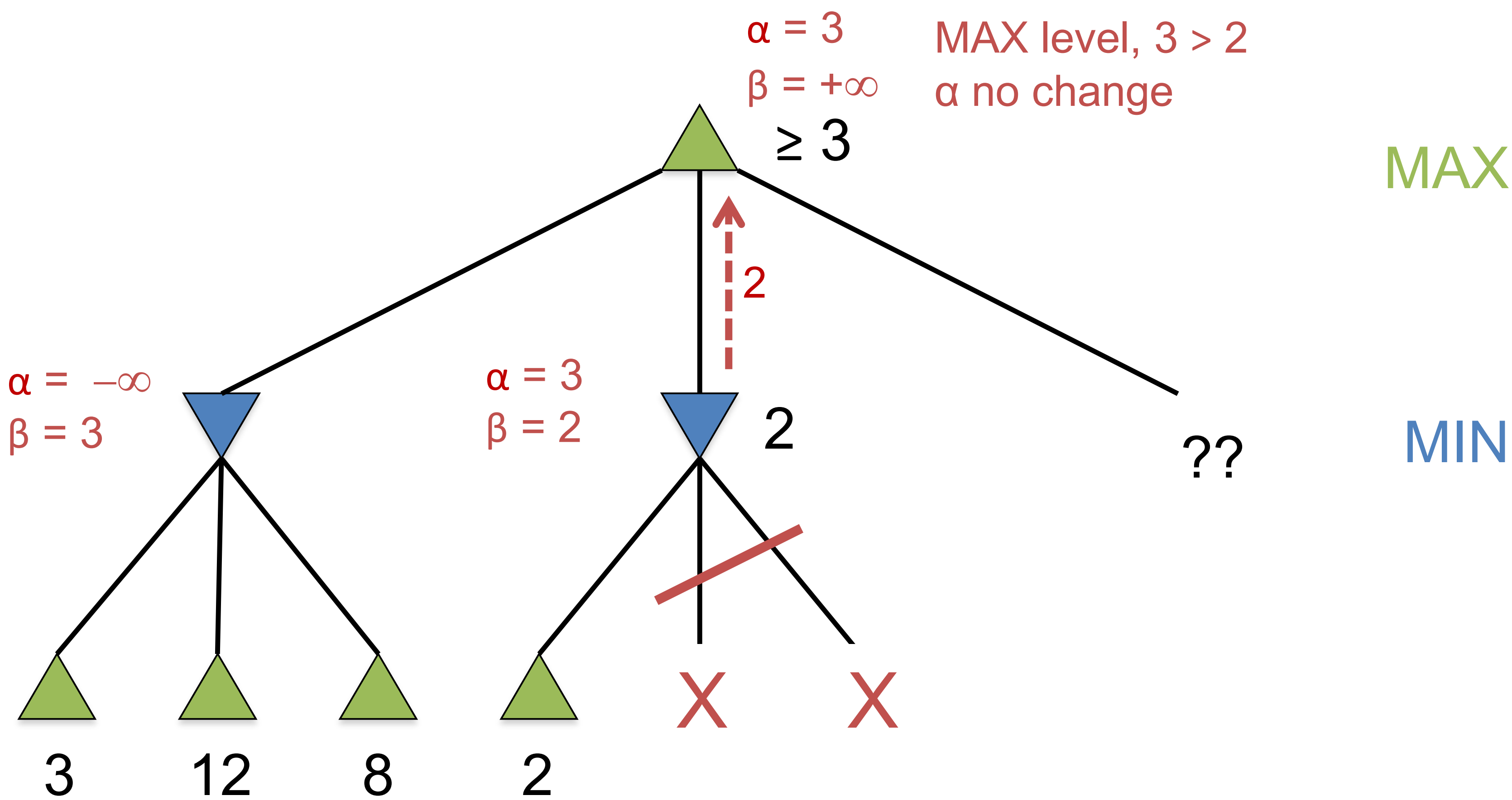
Observe leaf value; MIN's level; MIN updates β

Prune – play will never reach the other nodes!



Alpha-Beta Example

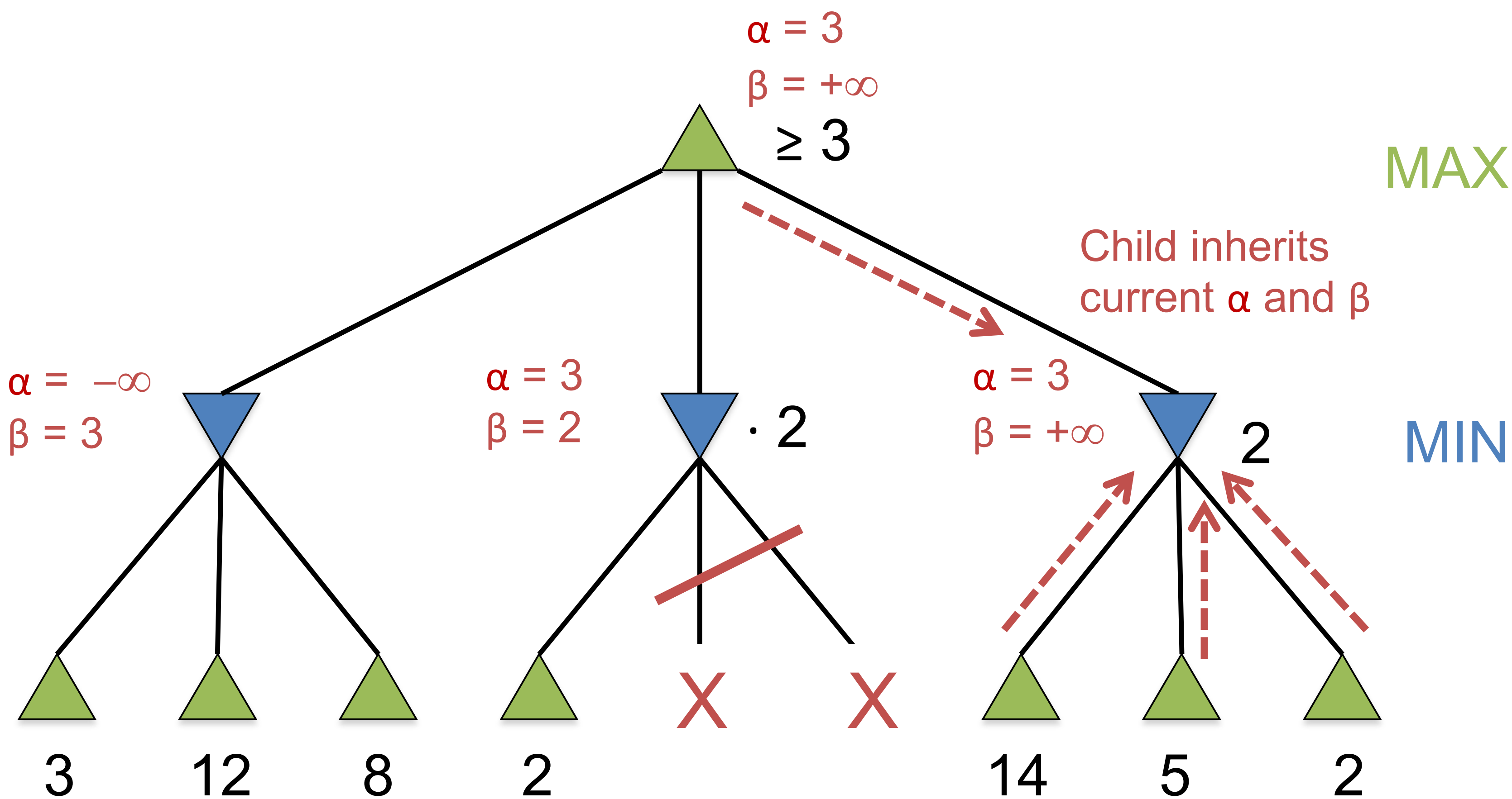
Pass outcome to caller & update caller:



Alpha-Beta Example

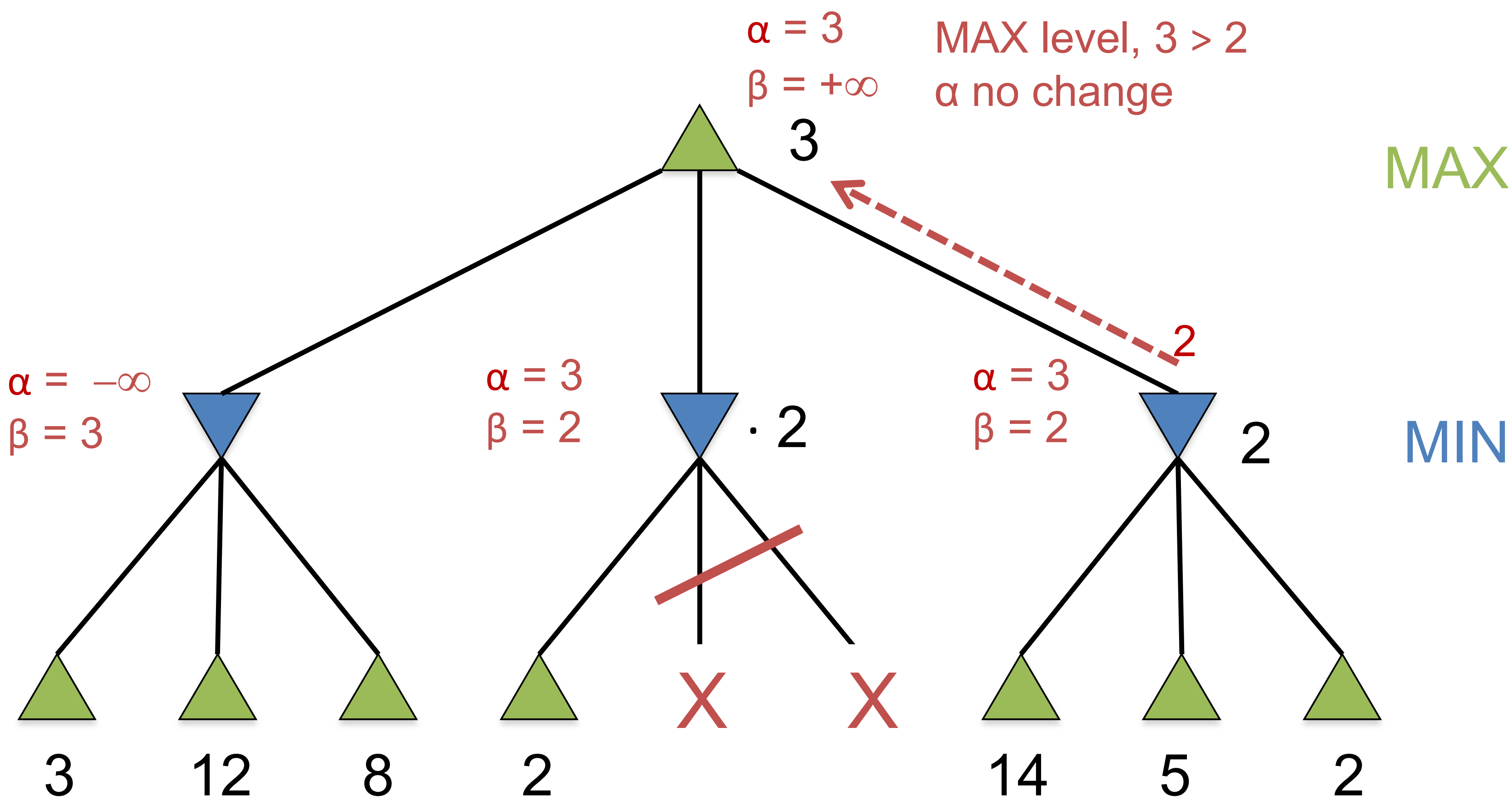
Continue depth-first exploration…

No pruning here; value is not resolved until final leaf.



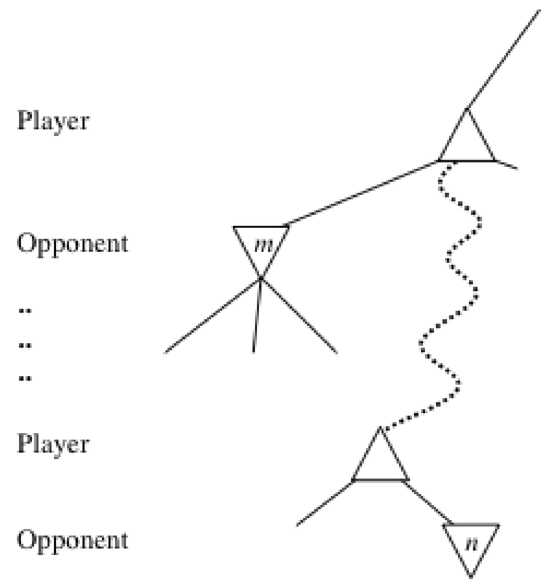
Alpha-Beta Example

Pass outcome to caller & update caller.
Value at the root is resolved.



General alpha-beta pruning

- Consider a node n in the tree:
- If player has a better choice at
  - Parent node of n
  - Or, any choice further up!
- Then n is never reached in play



- So:
  - When that much is known about n, it can be pruned

Recursive α-β pruning: R&N Fig. 5.7

```
function ALPHA-BETA-SEARCH(state) returns an action
   v ← MAX-VALUE(state, −∞, +∞)
   return the action in ACTIONS(state) with value v
```

Simple stub to call recursion functions
Initialize alpha, beta; get best value
Score each action; return best action

```
function MAX-VALUE(state, α, β) returns a utility value
   if CUTOFF-TEST(state) then return EVAL(state)
   v ← −∞
   for each a in ACTIONS(state) do
      v ← MAX(v, MIN-VALUE(RESULT(state,a), α, β))
      if v ≥ β then return v
      α ← MAX(α, v)
   return v
```

If Cutoff reached, return Eval heuristic
Otherwise, find our best child:
If our options become too good, our min ancestor will never let us come this way, so prune now & return best value so far
Finally, return the best value we found

```
function MIN-VALUE(state, α, β) returns a utility value
   if CUTOFF-TEST(state) then return EVAL(state)
   v ← +∞
   for each a in ACTIONS(state) do
      v ← MIN(v, MAX-VALUE(RESULT(state,a), α, β))
      if v ≤ α then return v
      β ← MIN(β, v)
   return v
```

If Cutoff reached, return Eval heuristic
Otherwise, find our worst child:
If our options become too bad, our max ancestor will never let us come this way, so prune now & return worst value so far
Finally, return the worst value we found

Figure 5.7 The alpha–beta search algorithm. Notice that these routines are the same as the MINIMAX functions in Figure 5.3, except for the two lines in each of MIN-VALUE and MAX-VALUE that maintain α and β (and the bookkeeping to pass these parameters along).

Recursive α-β pruning variant: Prune when α ≥ β

```
function ALPHA-BETA-SEARCH(state) returns an action
   v ← MAX-VALUE(state, −∞, +∞)
   return the action in ACTIONS(state) with value v

function MAX-VALUE(state, α, β) returns a utility value
   if CUTOFF-TEST(state) then return EVAL(state)
   v ← −∞
   for each a in ACTIONS(state) do
      v ← MAX(v, MIN-VALUE(RESULT(state,a), α, β))
      α ← MAX(α, v)
      if α ≥ β then return v
   return v

function MIN-VALUE(state, α, β) returns a utility value
   if CUTOFF-TEST(state) then return EVAL(state)
   v ← +∞
   for each a in ACTIONS(state) do
      v ← MIN(v, MAX-VALUE(RESULT(state,a), α, β))
      β ← MIN(β, v)
      if α ≥ β then return v
   return v
```

This variant has a conceptually simpler pruning rule (α ≥ β), but when pruning occurs it makes one extra call to MAX(). Both variants yield the same pruning behavior, and **both are considered correct on tests.**

Effectiveness of α-β Search

- Worst-Case
  - Branches are ordered so that no pruning takes place. In this case alpha-beta gives no improvement over exhaustive search

- Best-Case
  - Each player's best move is the left-most alternative (i.e., evaluated first)
  - In practice, performance is closer to best rather than worst-case

- In practice often get $O(b^{(d/2)})$ rather than $O(b^d)$
  - This is the same as having a branching factor of sqrt(b),
    - since $(\mathrm{sqrt}(b))^d = b^{(d/2)}$ (i.e., we have effectively gone from b to square root of b)
  - In chess go from b ~ 35 to b ~ 6
    - permiting much deeper search in the same amount of time

Iterative deepening

- In real games, there is usually a time limit T to make a move

- How do we take this into account?
- Minimax cannot use “partial” results with any confidence, unless the full tree has been searched
  - Conservative: set small depth limit to guarantee finding a move in time < T
  - But, we may finish early – could do more search!

- Added benefit with Alpha-Beta Pruning:
  - Remember node values found at the previous depth limit
  - Sort current nodes so that each player’s best move is left-most child
  - Likely to yield good Alpha-Beta Pruning => better, faster search
  - Only a heuristic: node values will change with the deeper search
  - Usually works well in practice

Comments on alpha-beta pruning

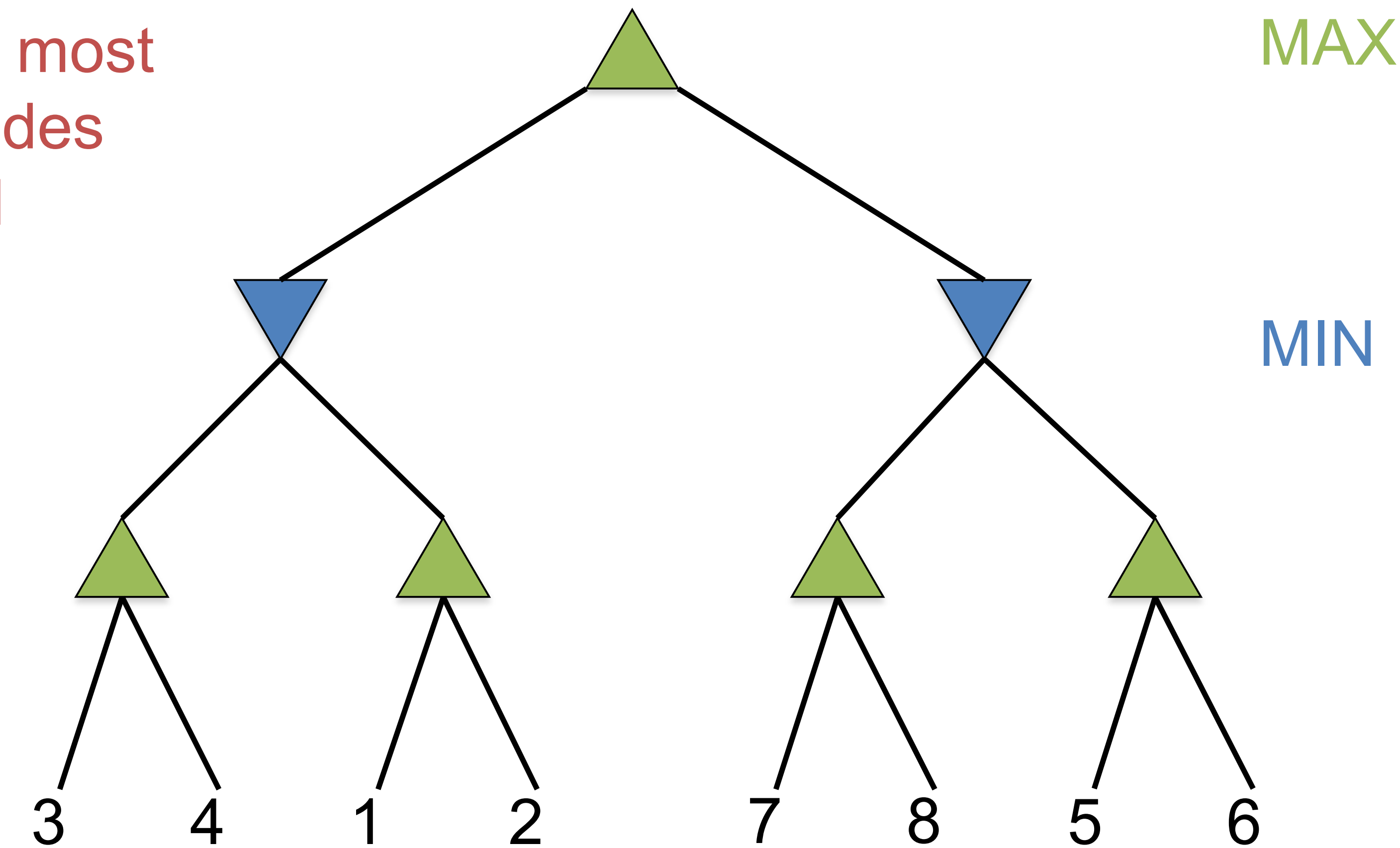
- Pruning does not affect final results
- Entire subtrees can be pruned
- Good move ordering improves pruning
  - Order nodes so player’s best moves are checked first
- Repeated states are still possible
  - Store them in memory = transposition table

Iterative deepening reordering

Which leaves can be pruned?

None!

because the most favorable nodes are explored last…

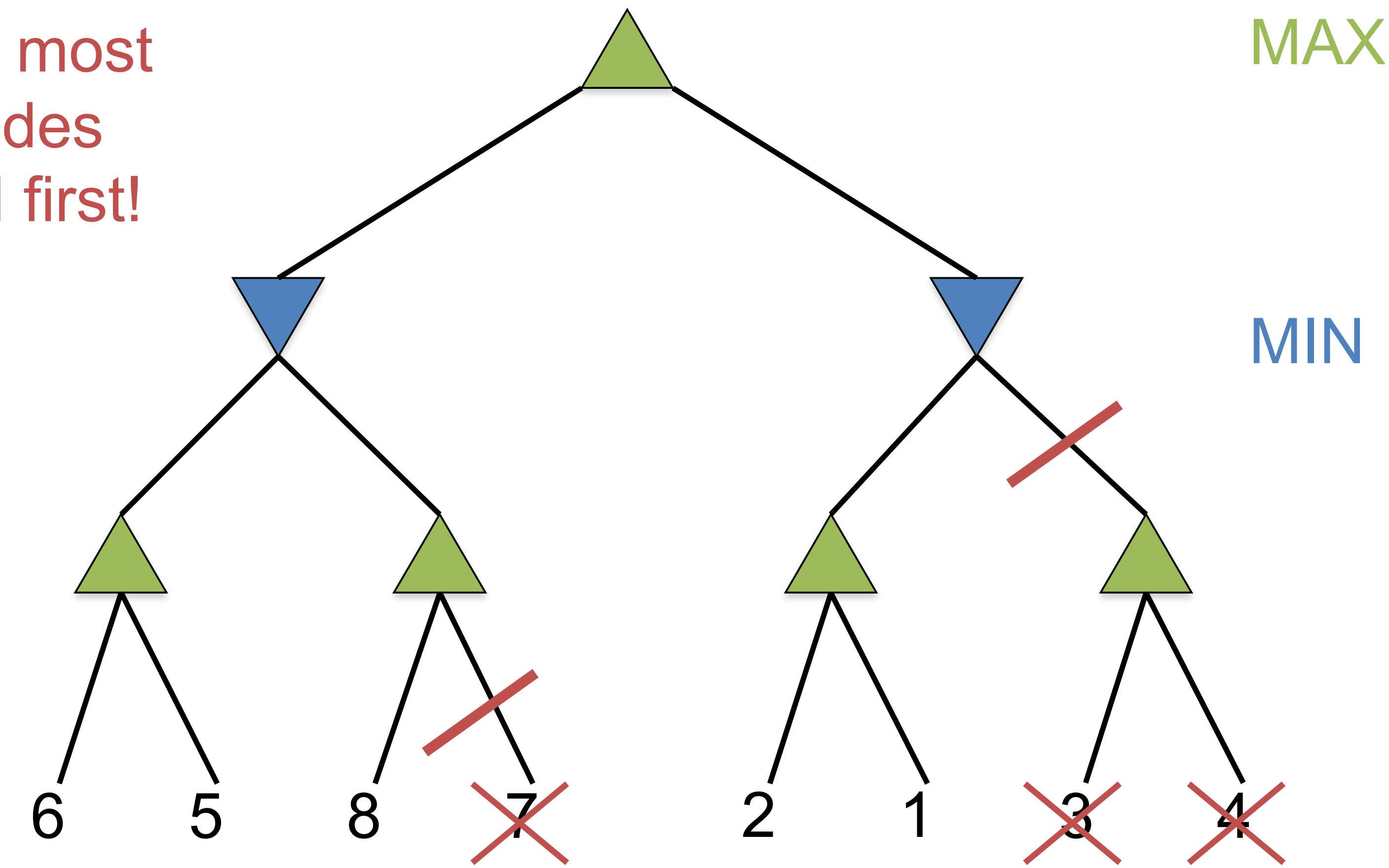


Iterative deepening reordering

Different exploration order: now which leaves can be pruned?

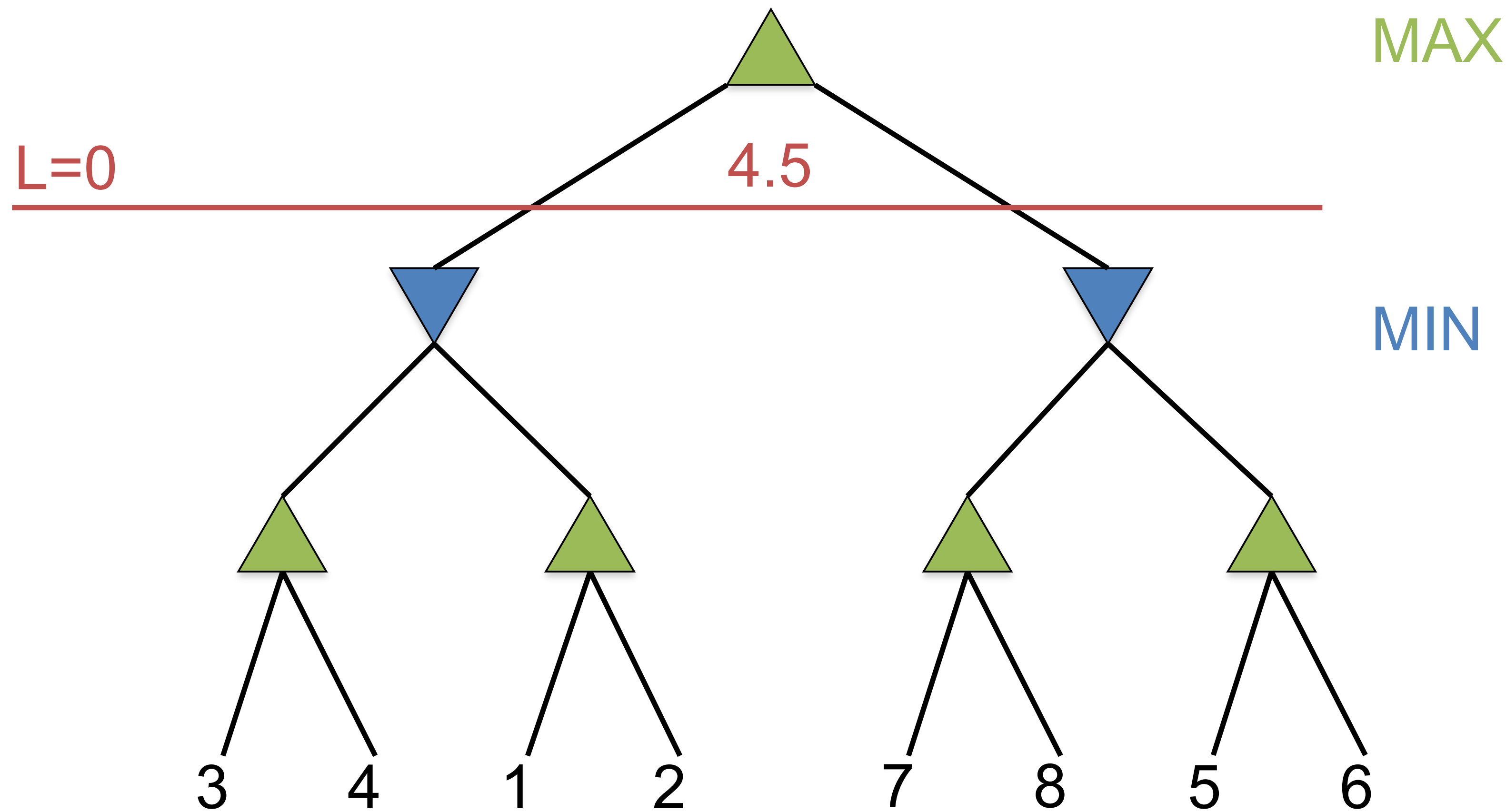
Lots!

because the most favorable nodes are explored first!



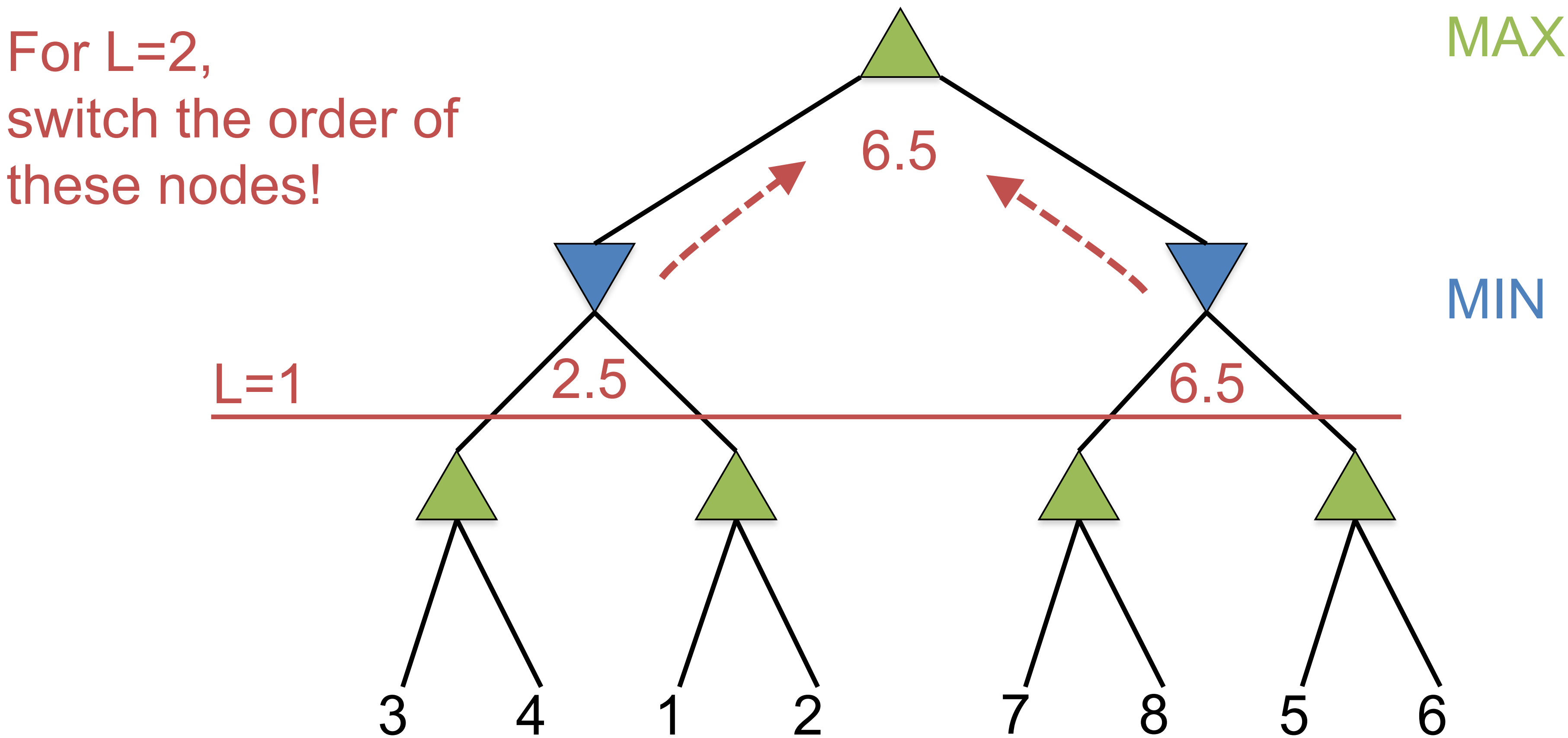
Iterative deepening reordering

Order with no pruning; use iterative deepening approach.
Assume node score is the average of leaf values below.



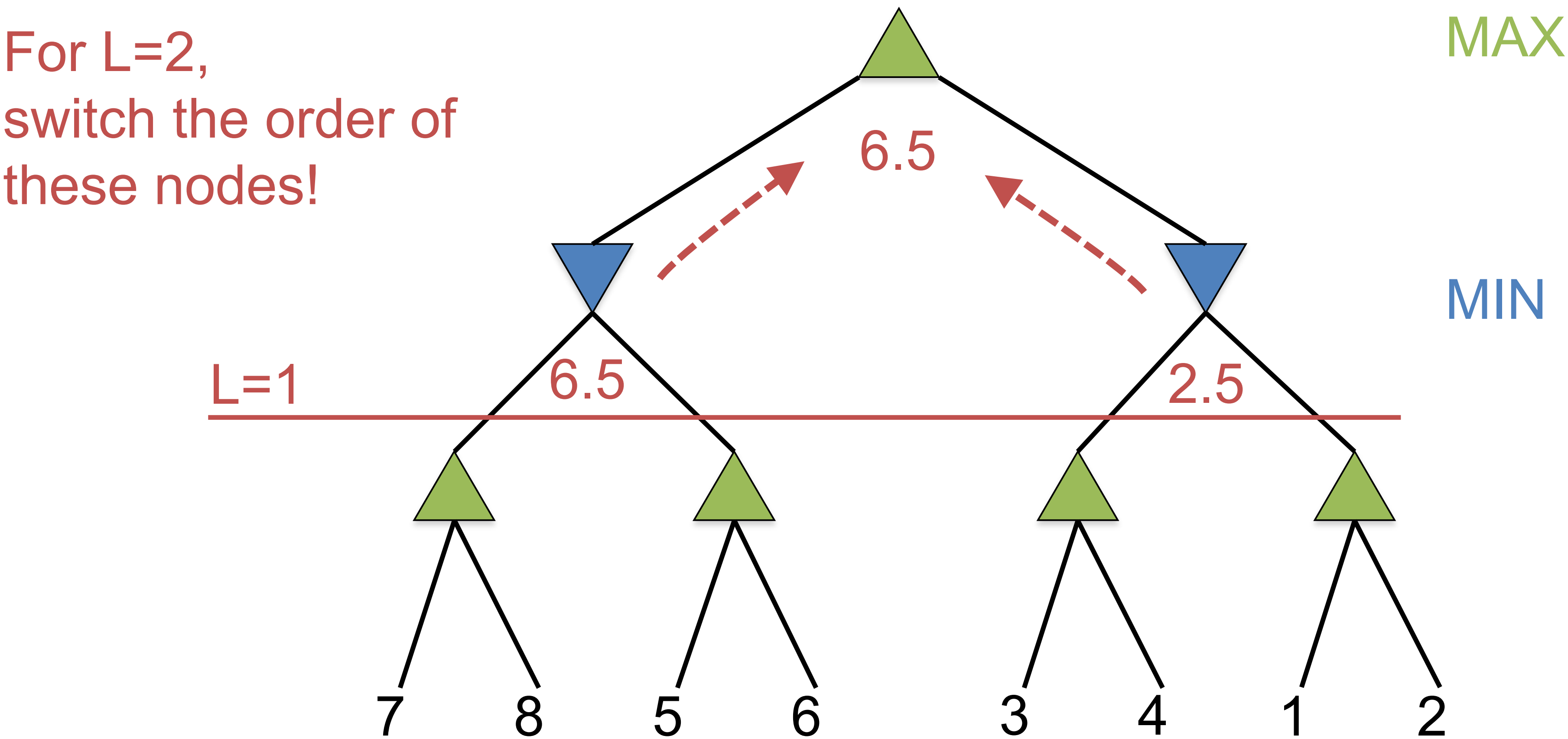
Iterative deepening reordering

Order with no pruning; use iterative deepening approach.
Assume node score is the average of leaf values below.



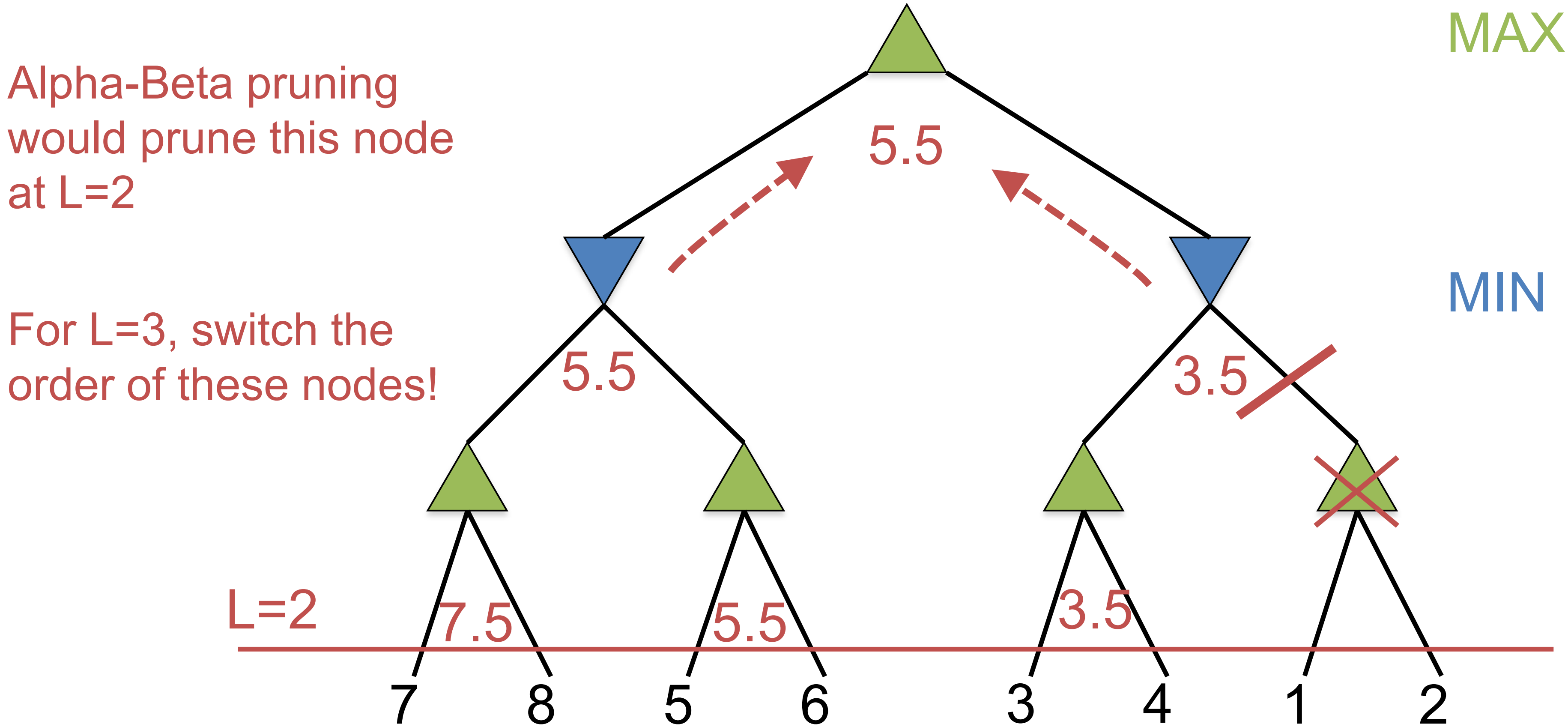
Iterative deepening reordering

Order with no pruning; use iterative deepening approach.
Assume node score is the average of leaf values below.



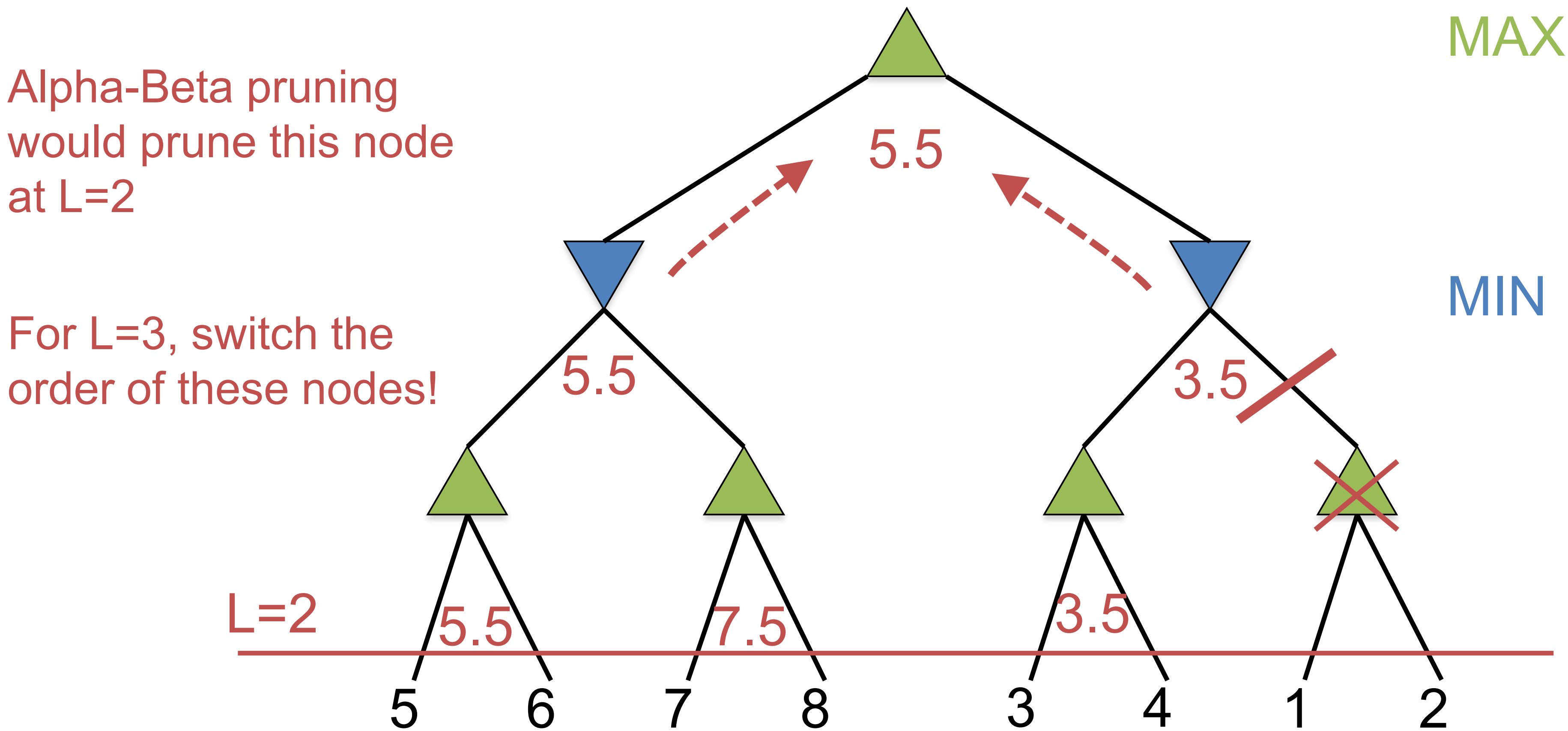
Iterative deepening reordering

Order with no pruning; use iterative deepening approach.
Assume node score is the average of leaf values below.



Iterative deepening reordering

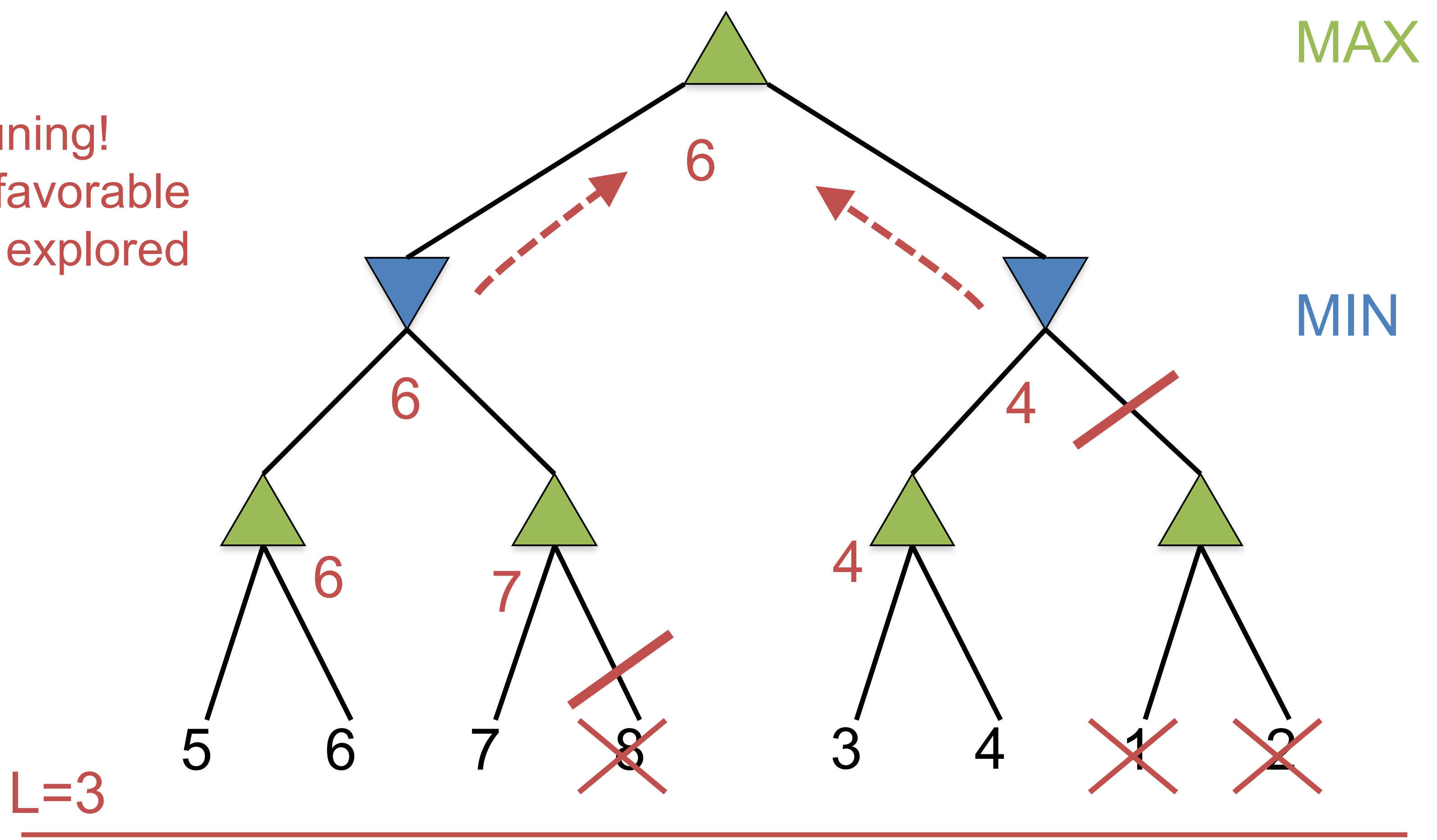
Order with no pruning; use iterative deepening approach.
Assume node score is the average of leaf values below.



Iterative deepening reordering

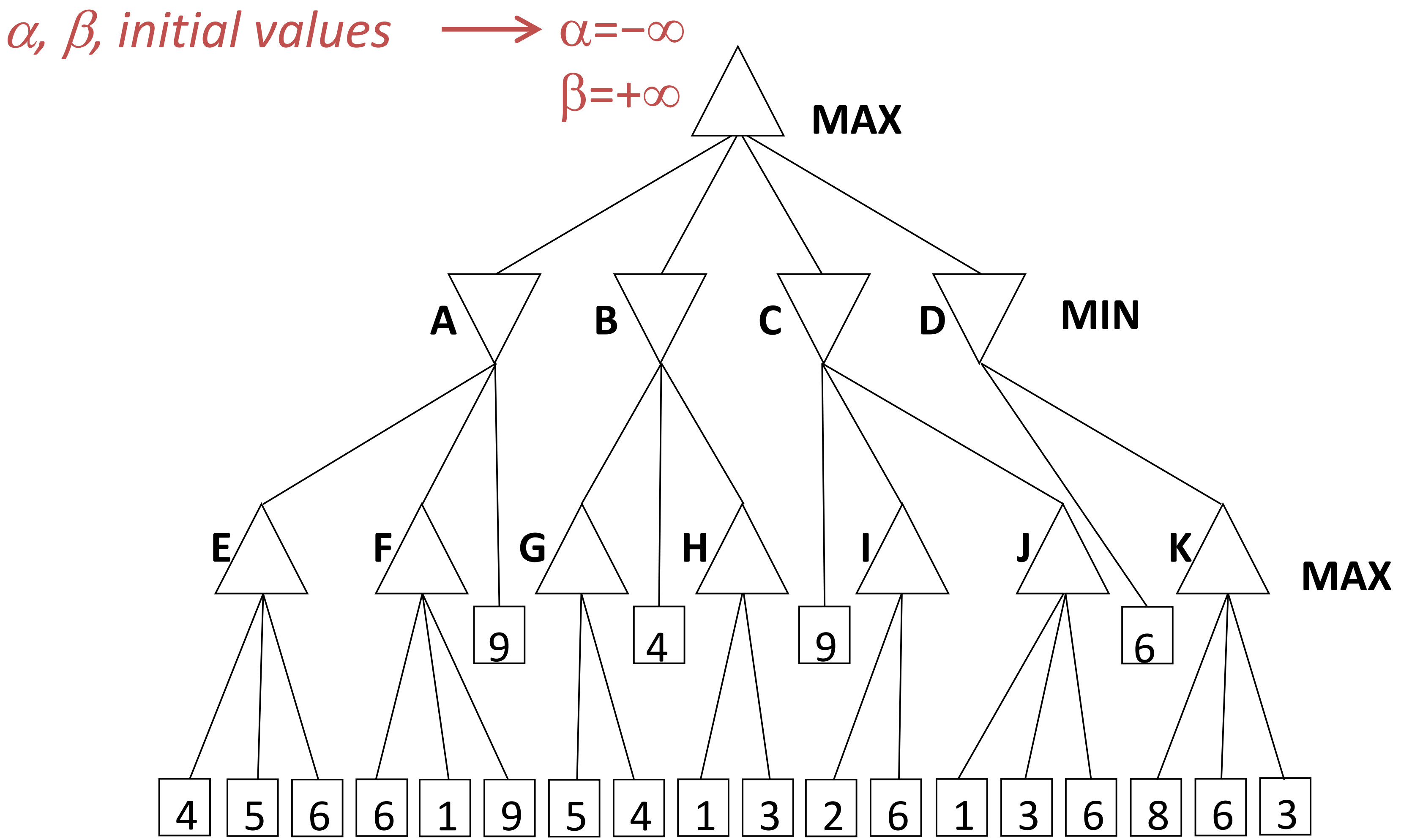
Order with no pruning; use iterative deepening approach.
Assume node score is the average of leaf values below.

Lots of pruning!
The most favorable nodes are explored earlier.



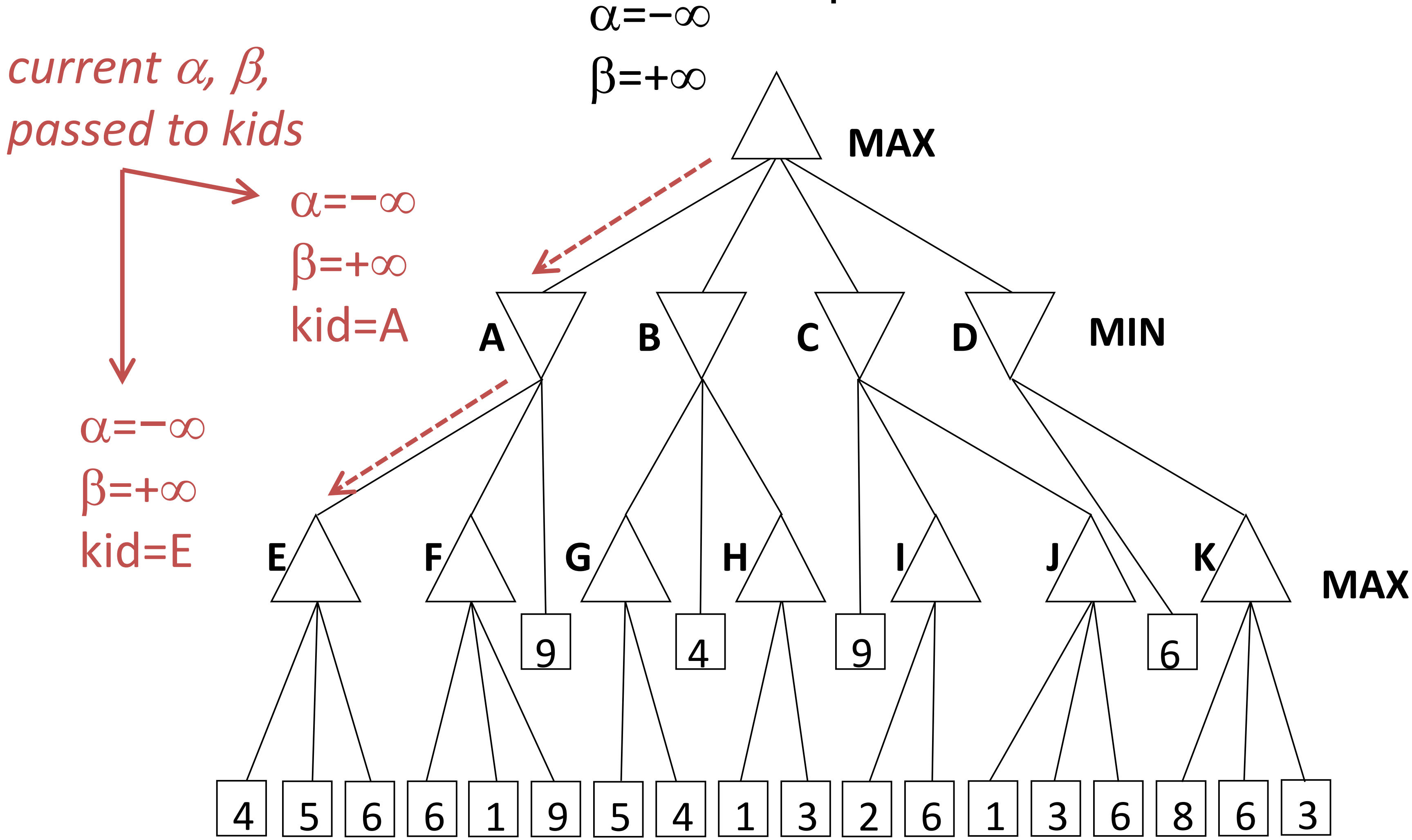
Longer Alpha-Beta Example

Branch nodes are labeled A..K for easy discussion

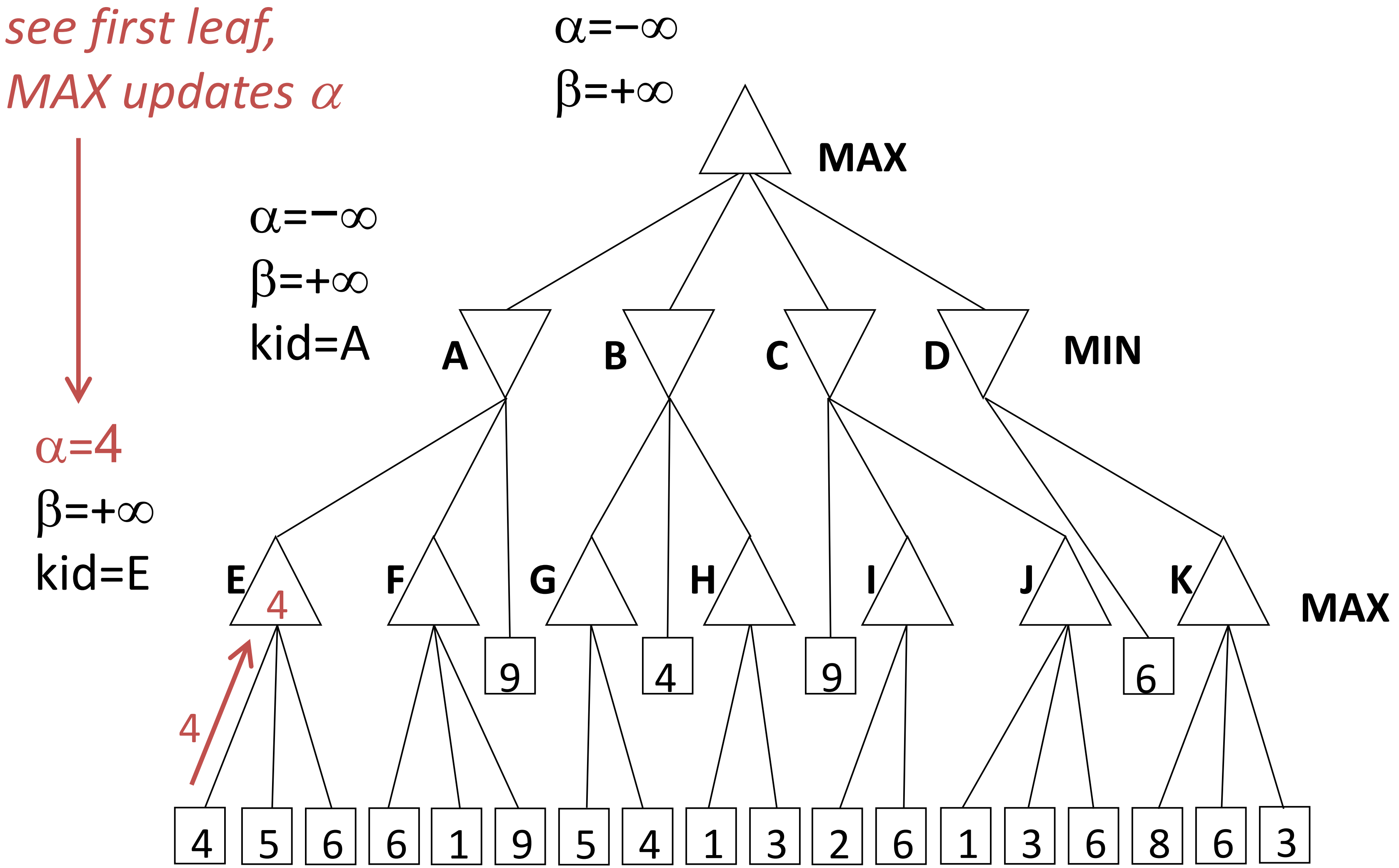


Longer Alpha-Beta Example

Note that cut-off occurs at different depths…



Longer Alpha-Beta Example

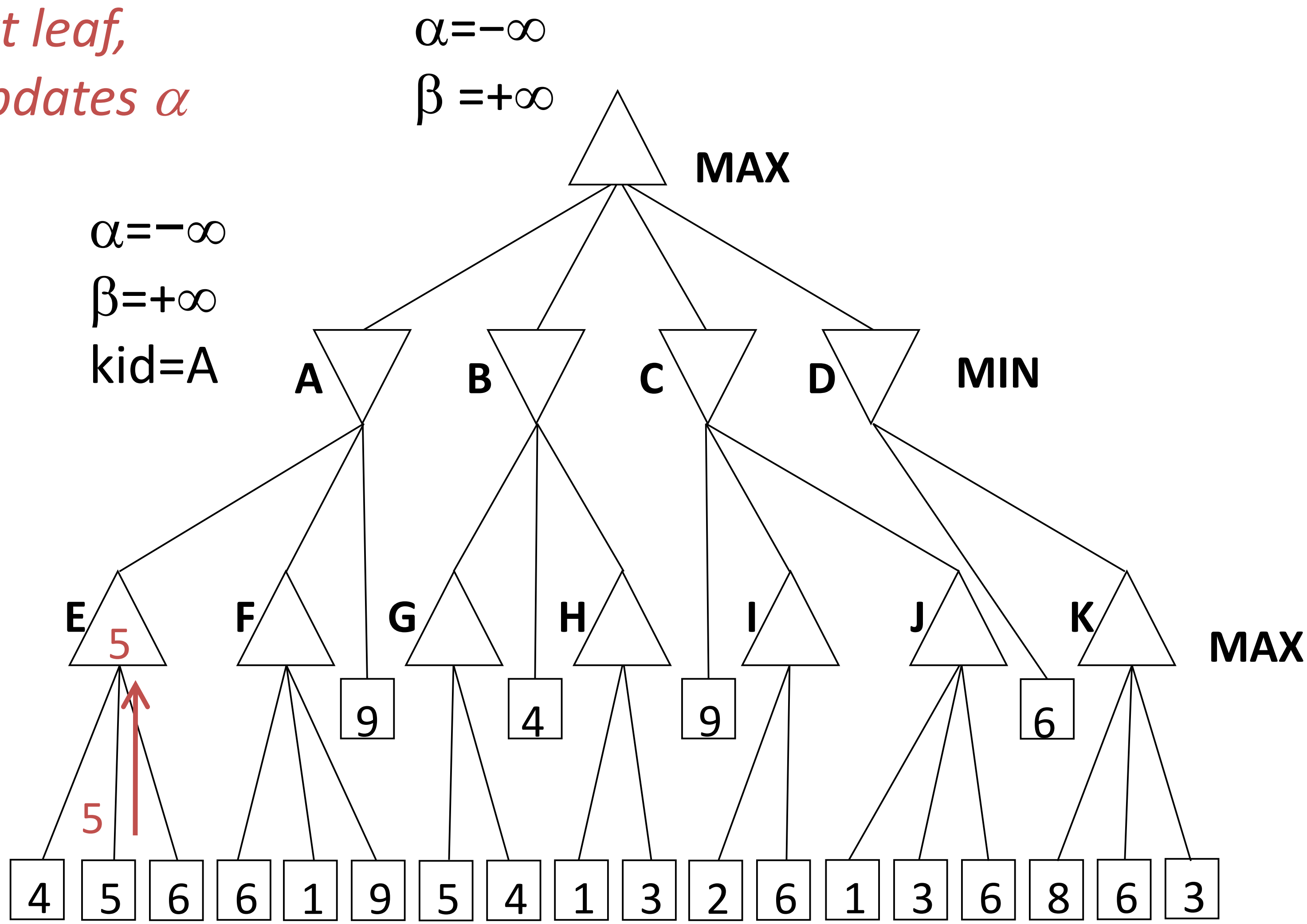


We also are running MiniMax search and recording node values within the triangles, without explicit comment.

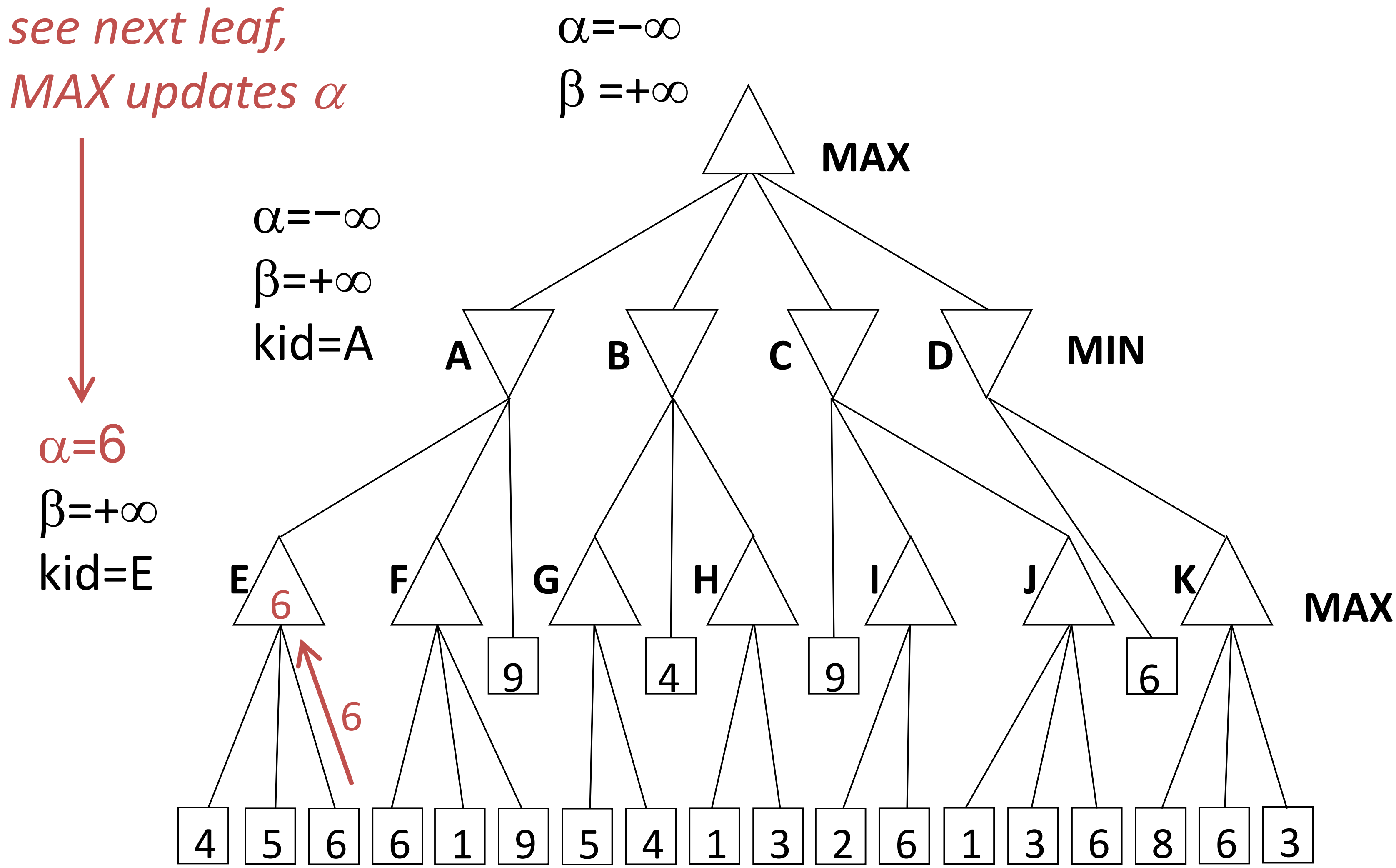
Longer Alpha-Beta Example

see next leaf,
MAX updates α

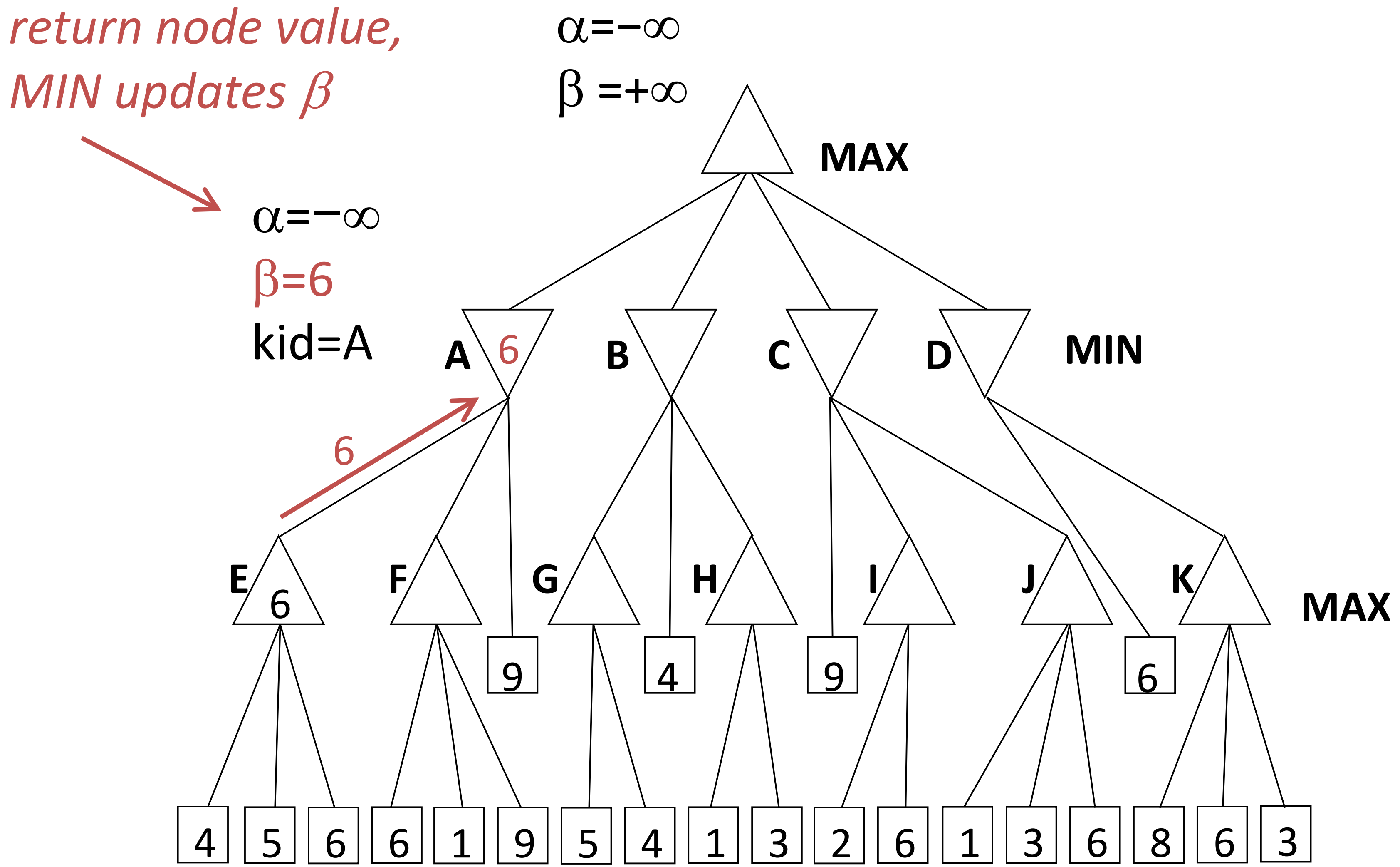
$\alpha=5$
$\beta=+\infty$
kid=E



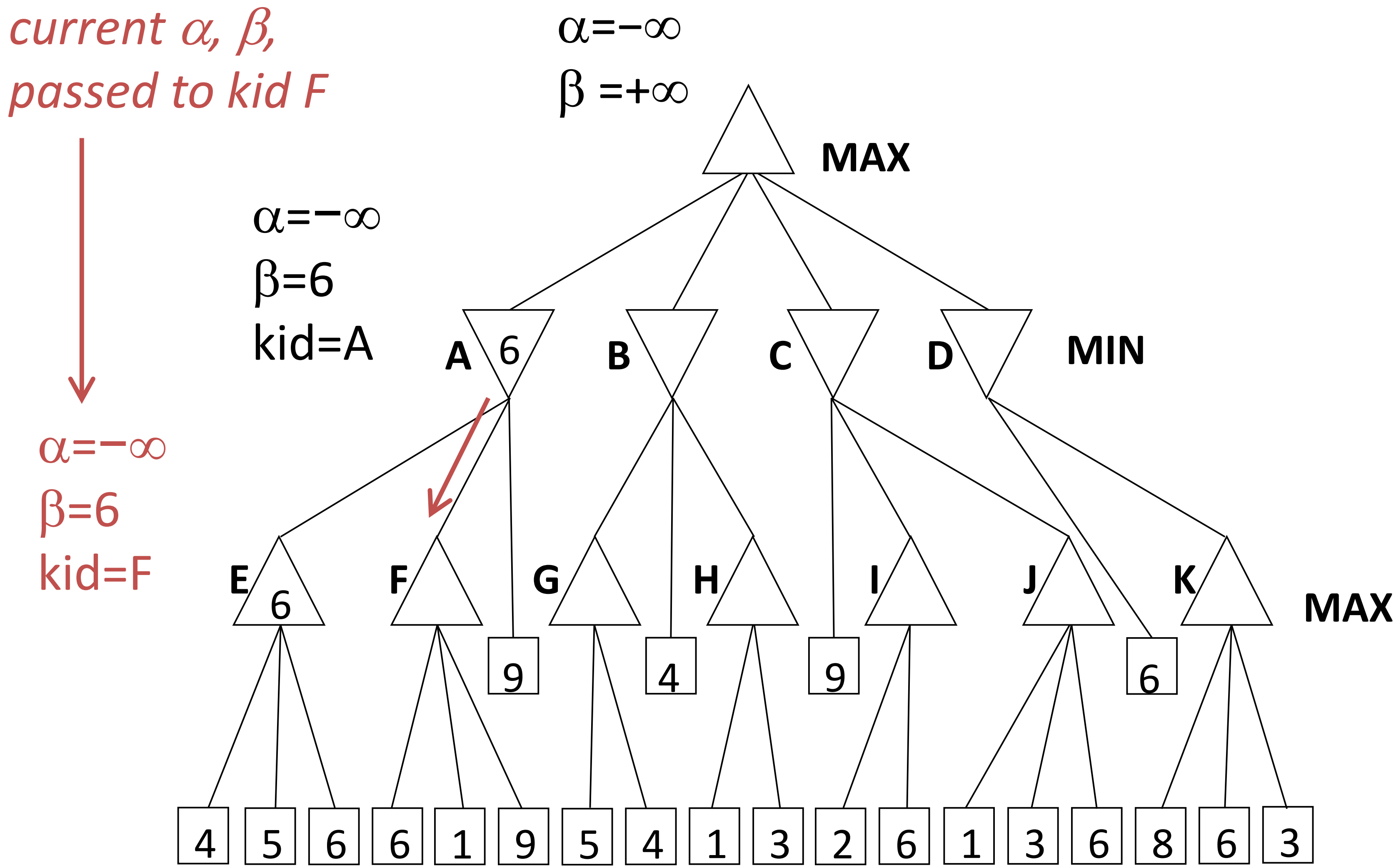
Longer Alpha-Beta Example



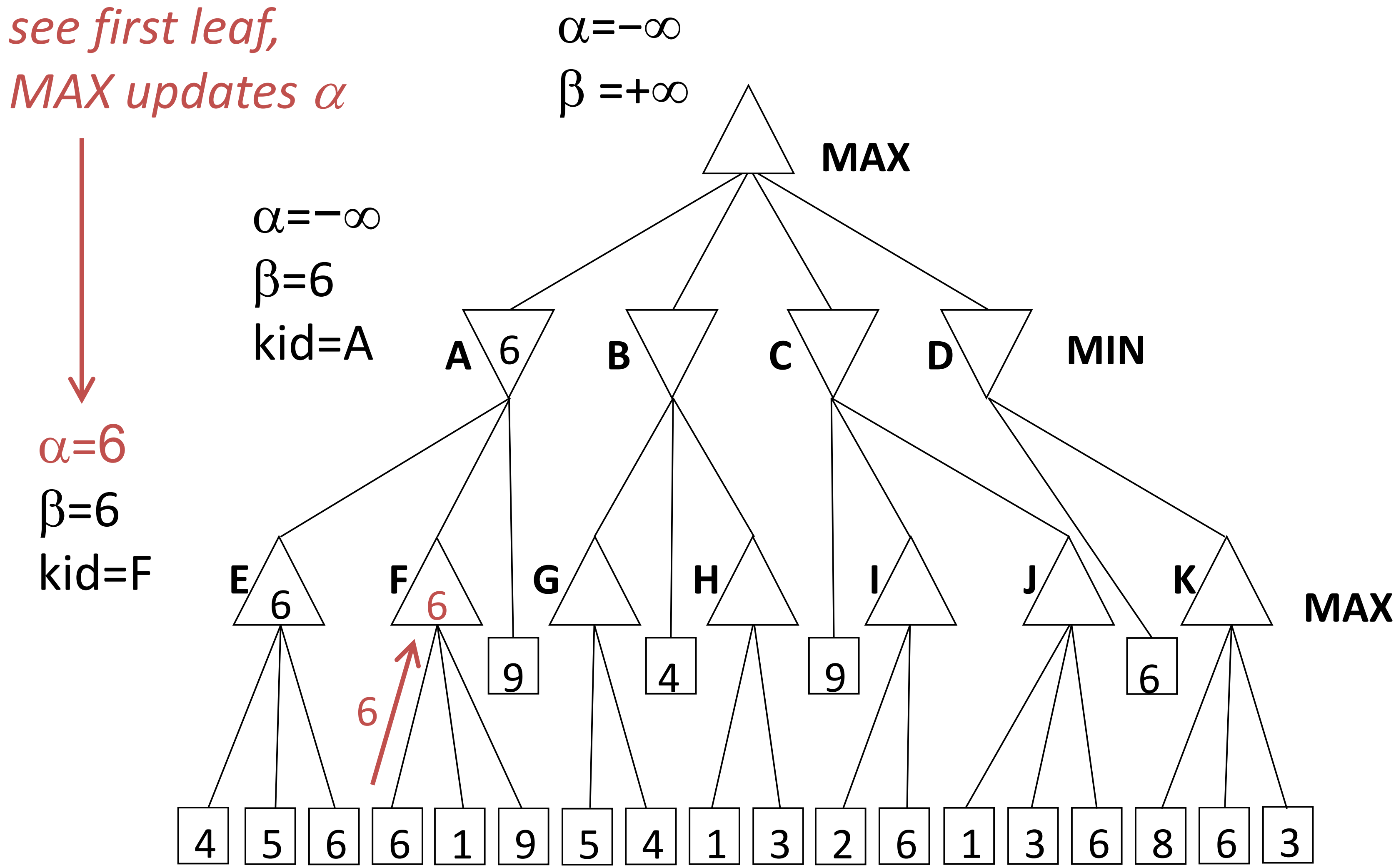
Longer Alpha-Beta Example



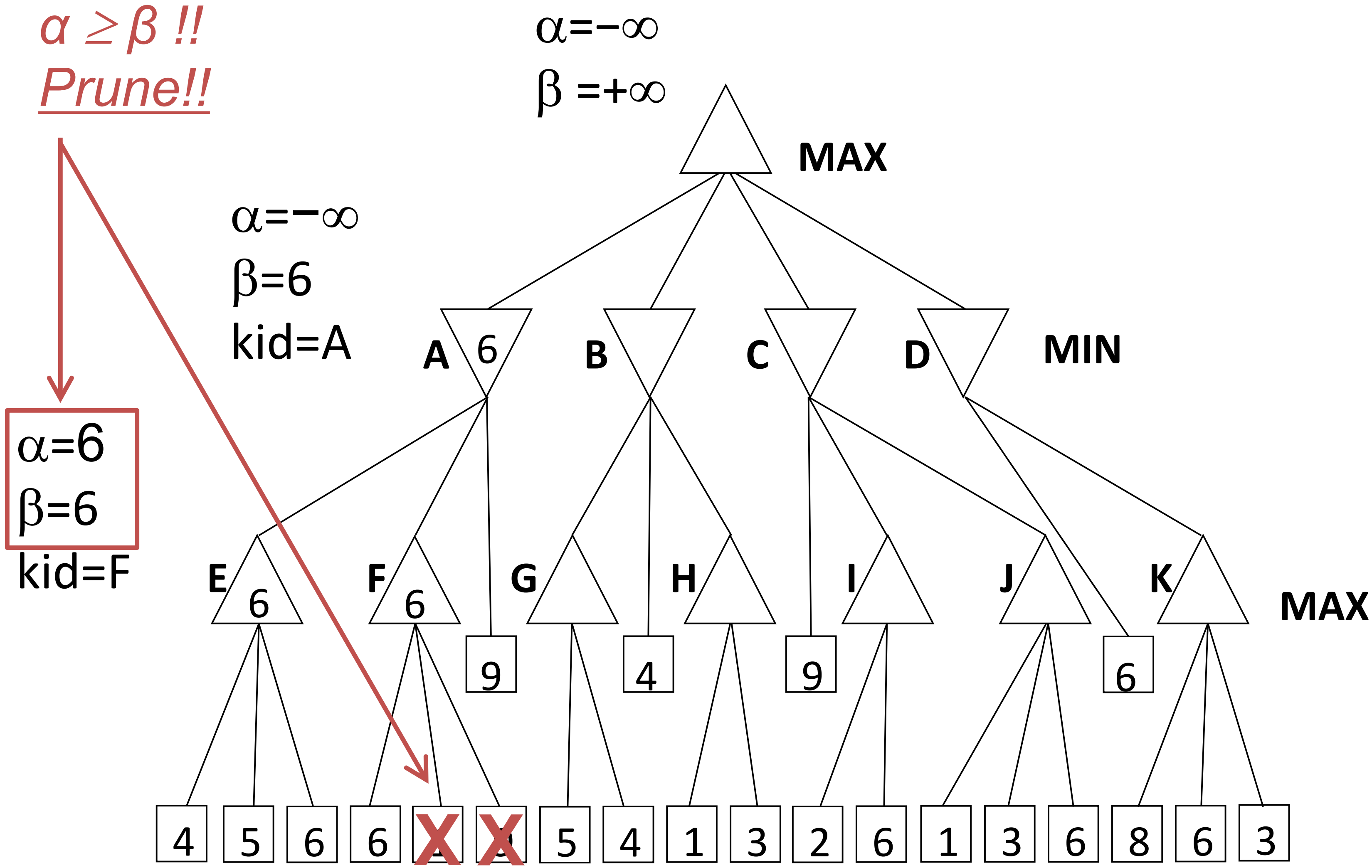
Longer Alpha-Beta Example



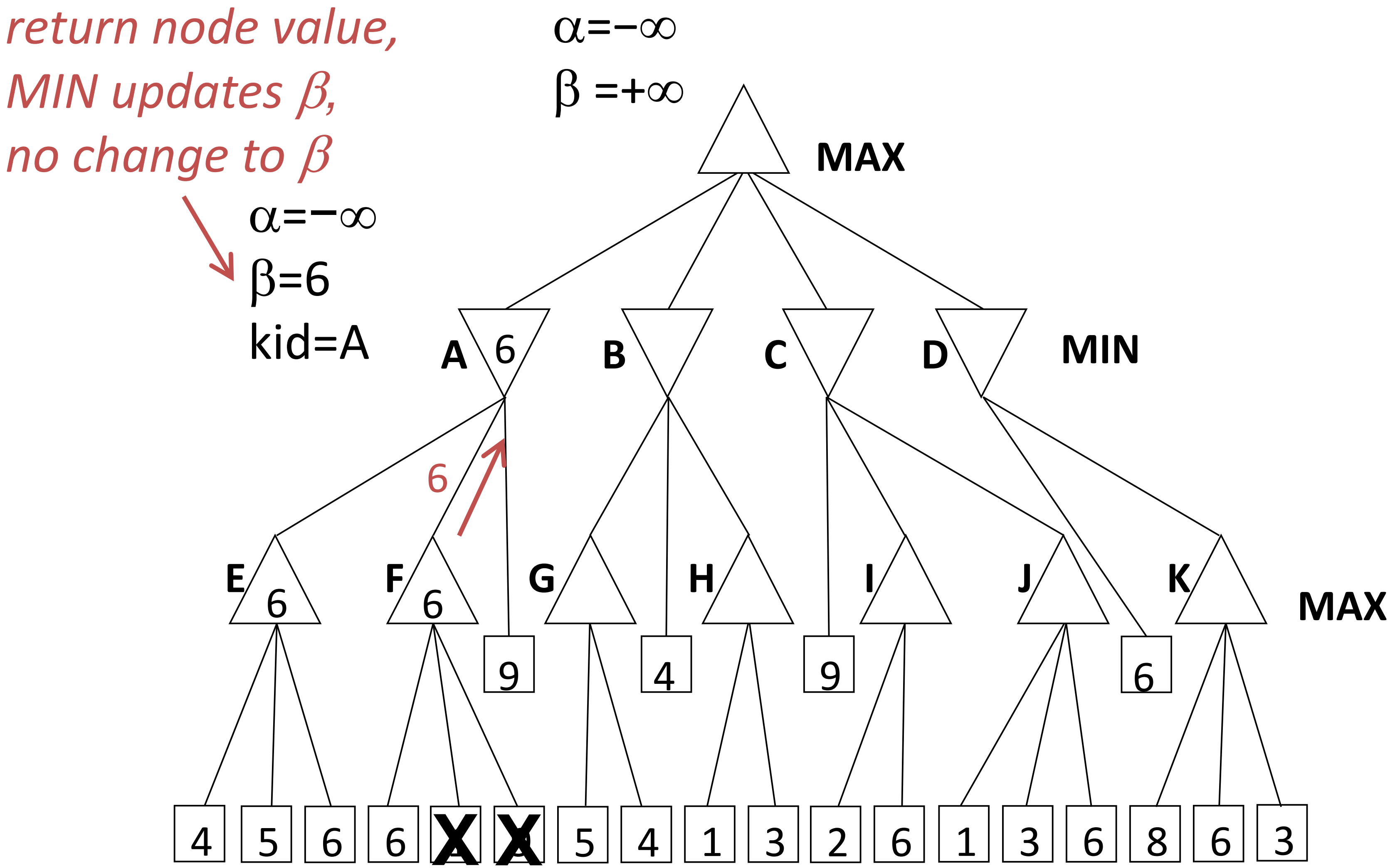
Longer Alpha-Beta Example



Longer Alpha-Beta Example

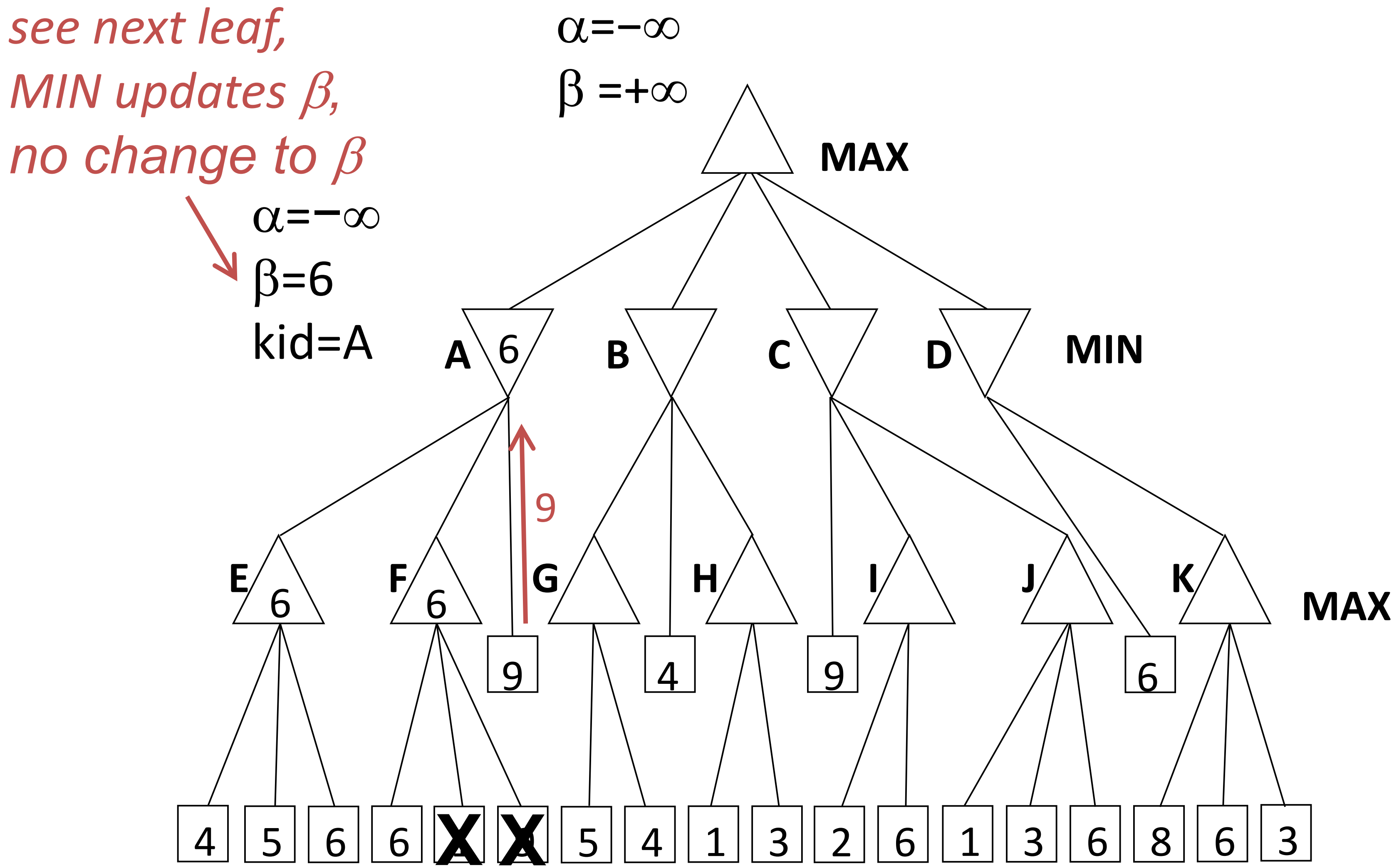


Longer Alpha-Beta Example



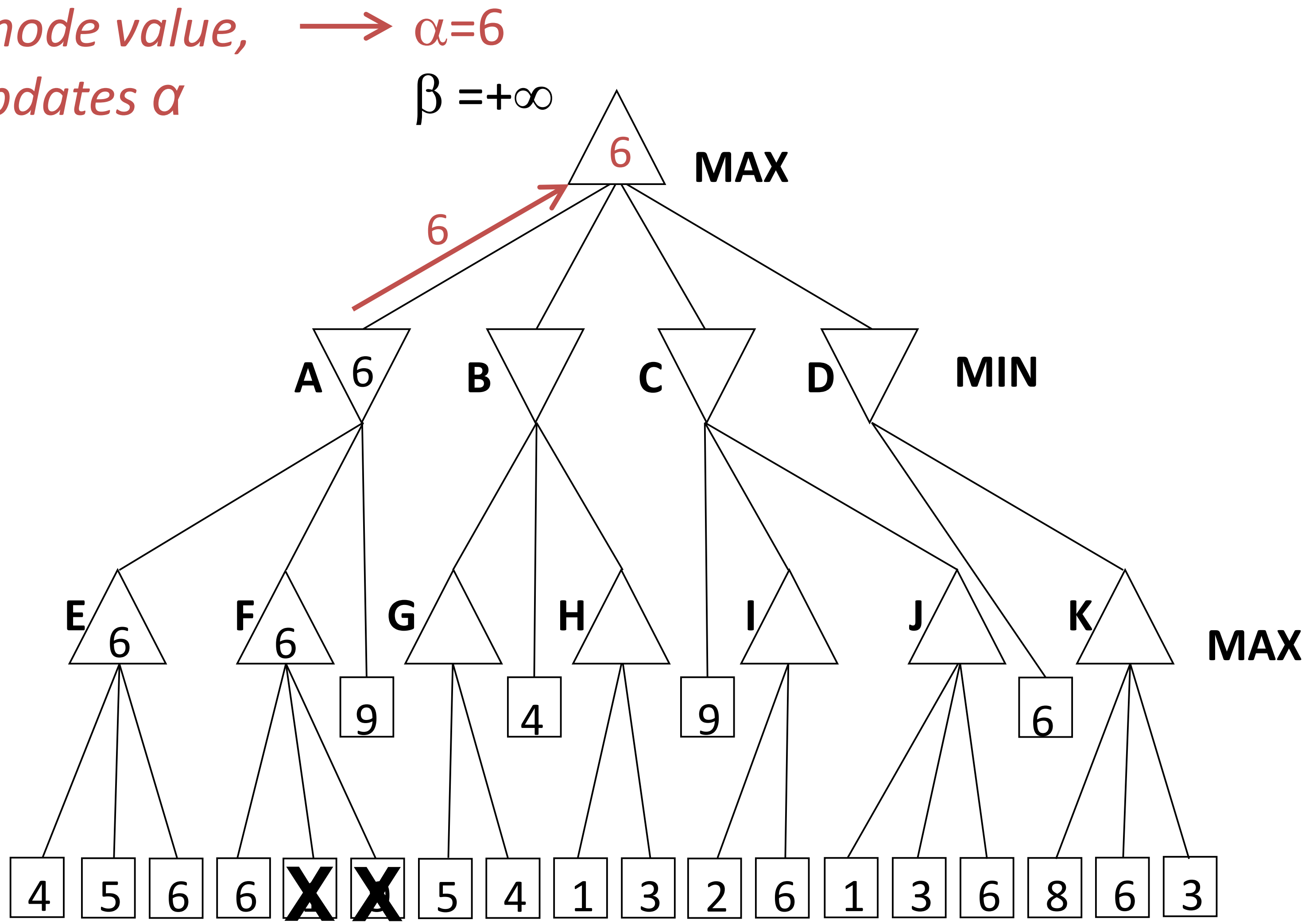
If we had continued searching at node F, we would see the 9 from its third leaf. Our returned value would be 9 instead of 6. But at A, MIN would choose E(=6) instead of F(=9). Internal values may change; root values do not.

Longer Alpha-Beta Example

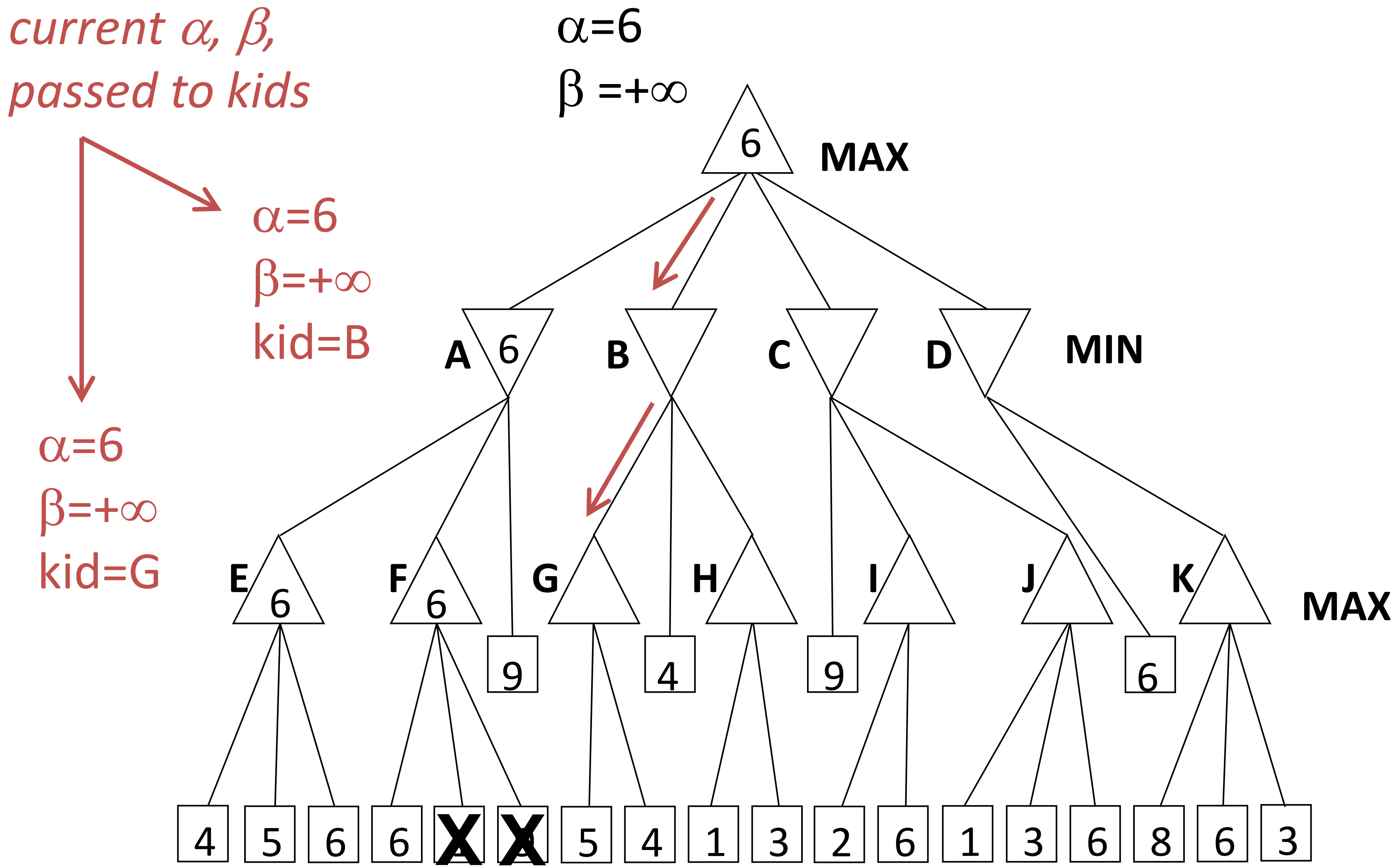


Longer Alpha-Beta Example

return node value,
MAX updates α

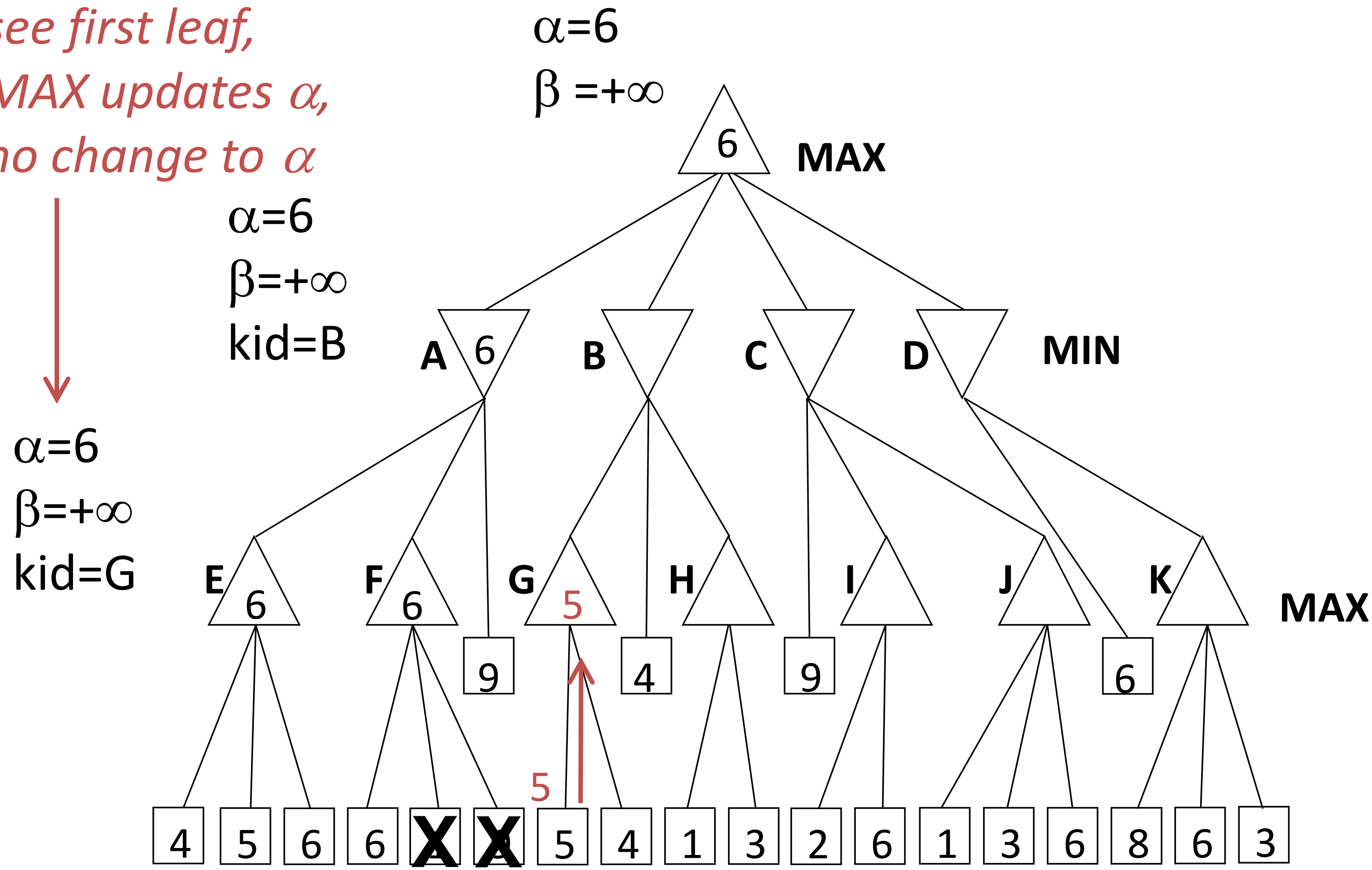


Longer Alpha-Beta Example



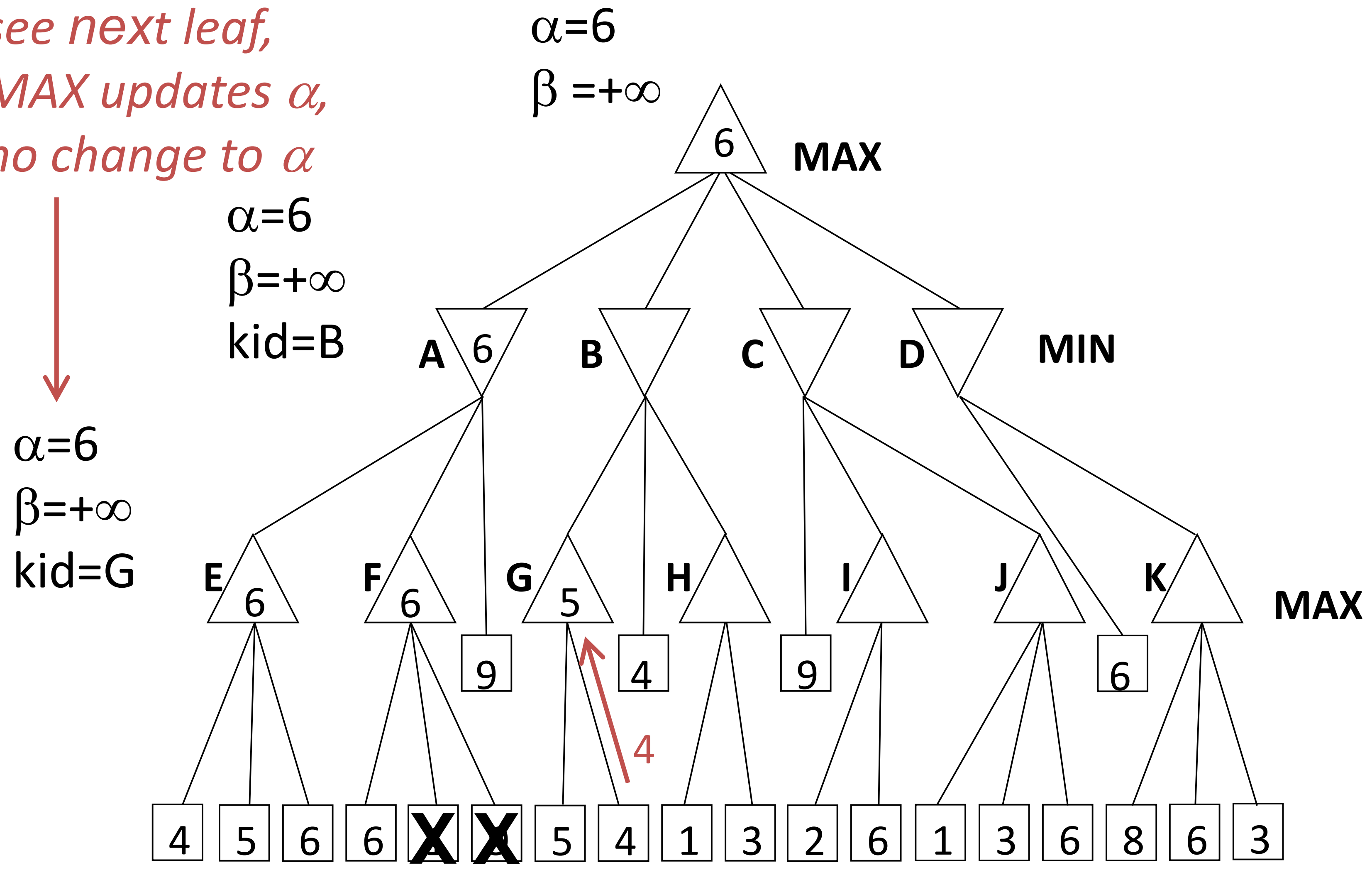
Longer Alpha-Beta Example

see first leaf,
MAX updates α,
no change to α



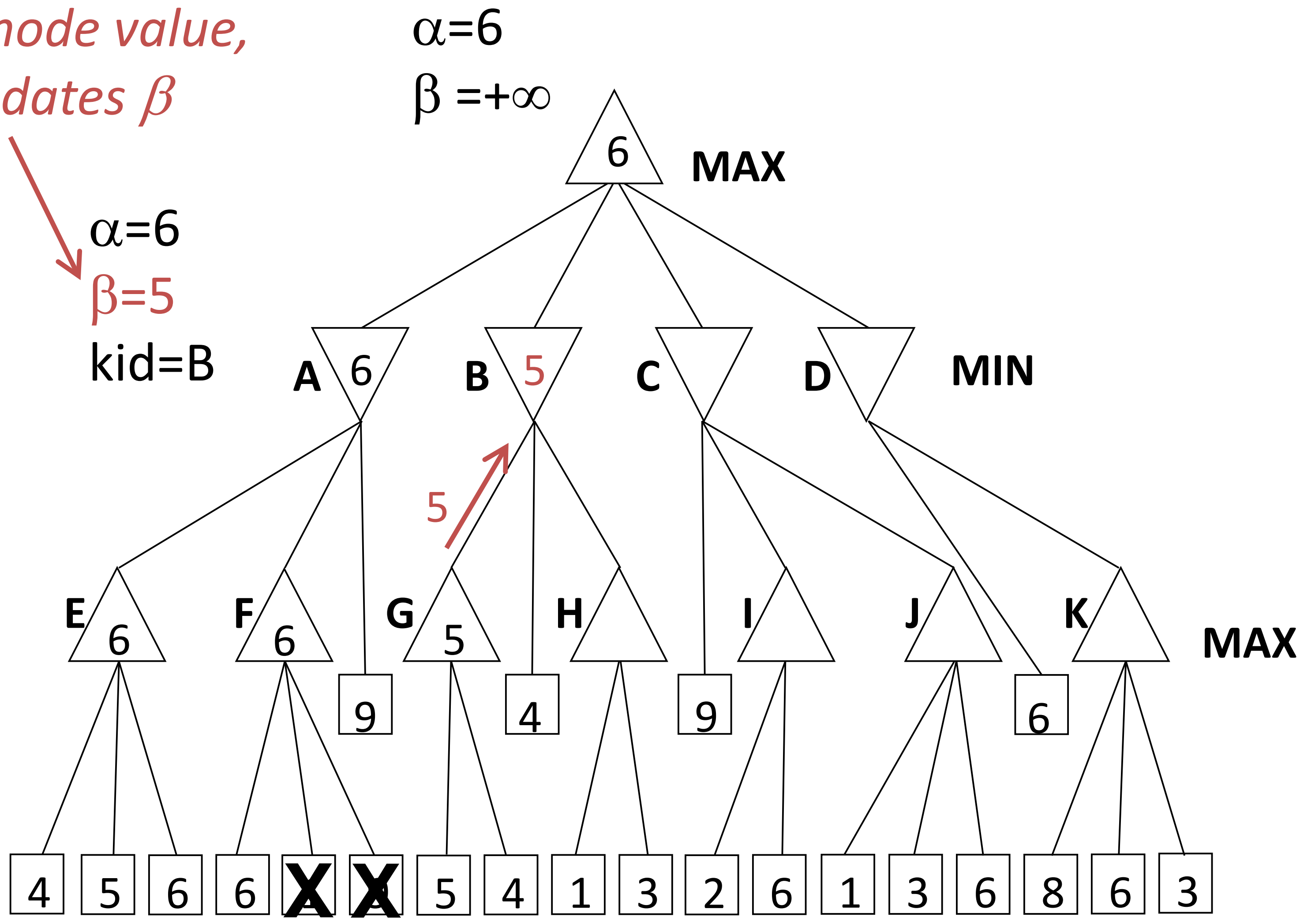
Longer Alpha-Beta Example

see next leaf,
MAX updates α,
no change to α

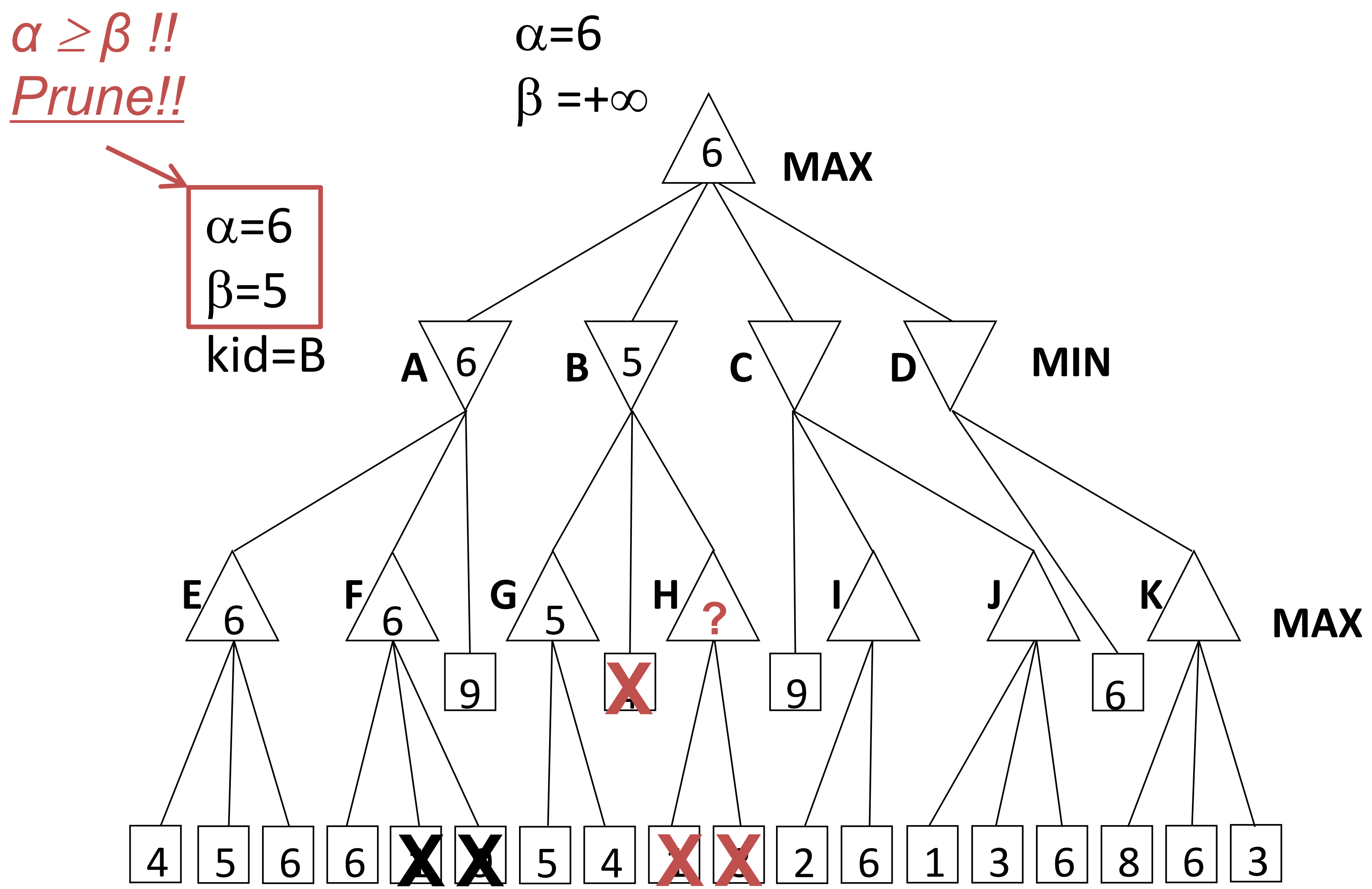


Longer Alpha-Beta Example

return node value,
MIN updates β



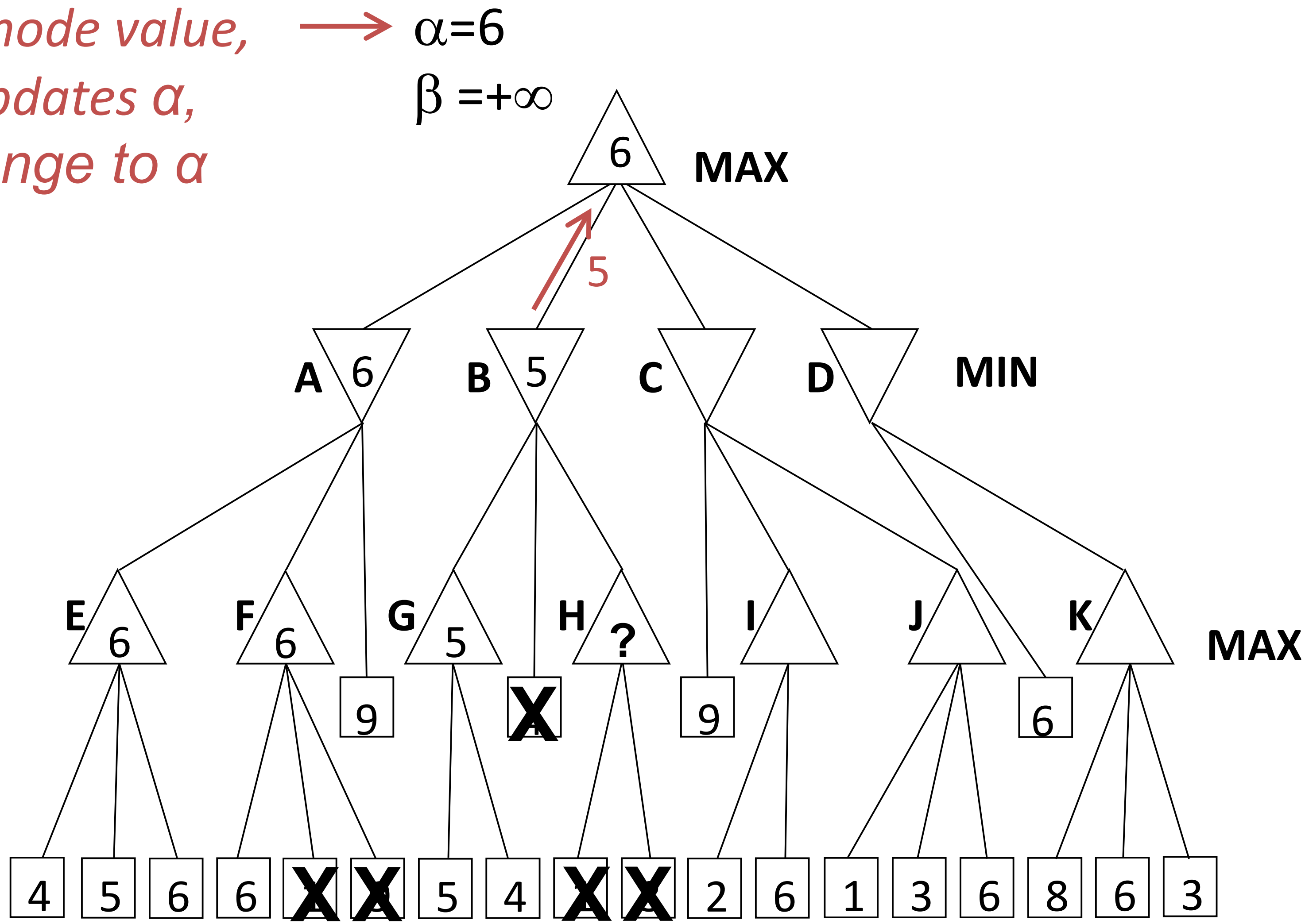
Longer Alpha-Beta Example



Note that we never find out, what is the node value of H? But we have proven it doesn't matter, so we don't care.

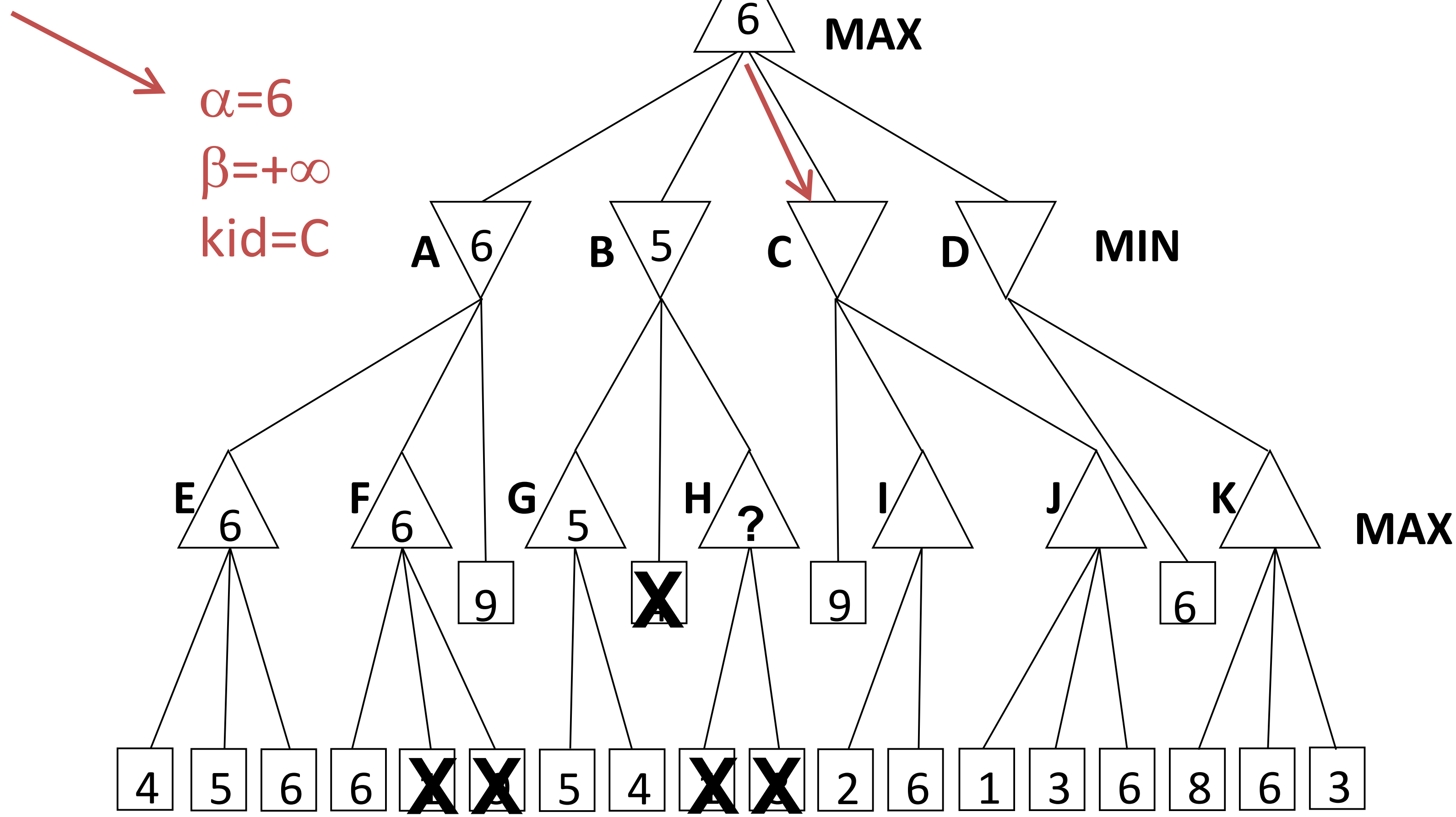
Longer Alpha-Beta Example

return node value,
MAX updates α,
no change to α



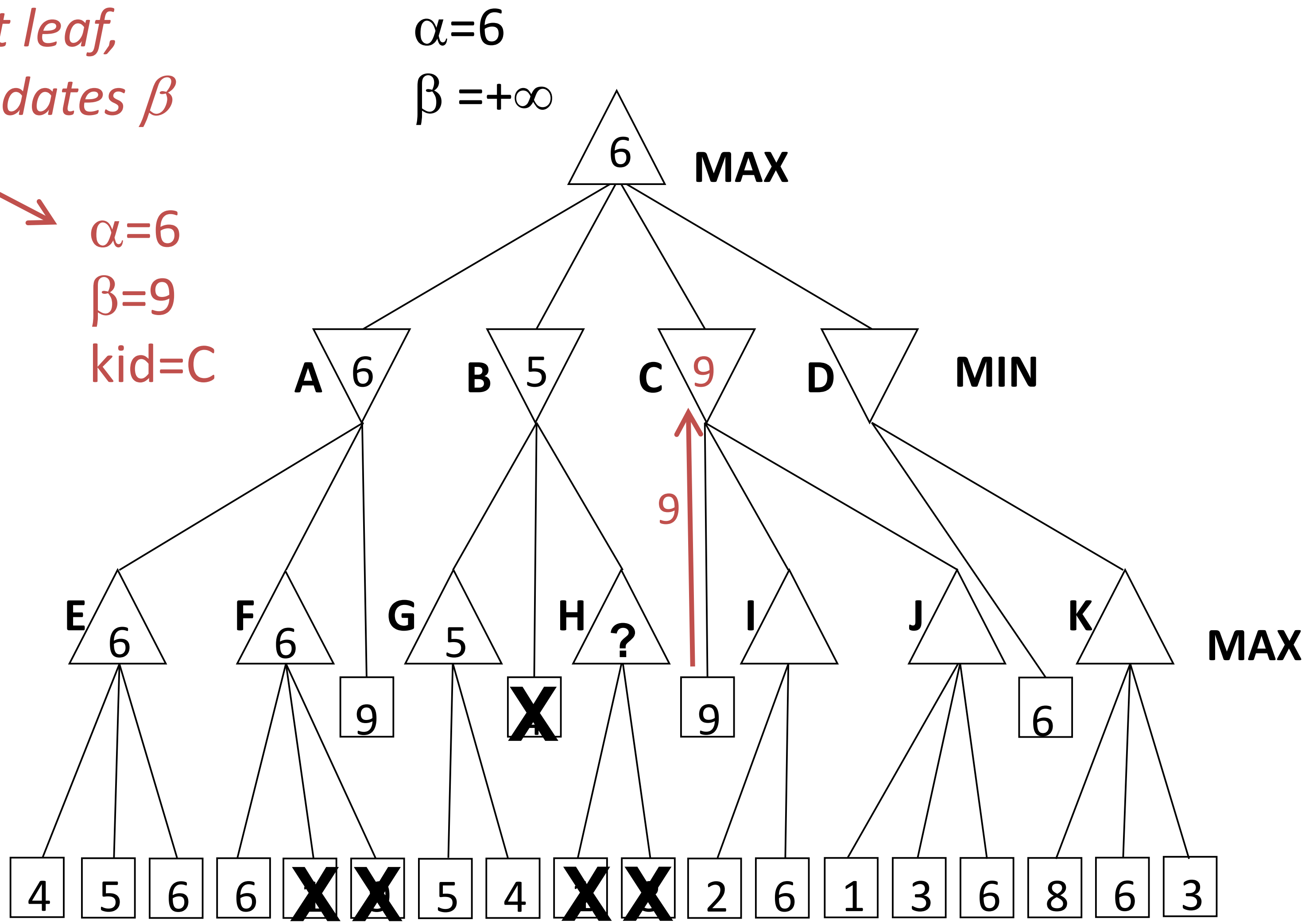
Longer Alpha-Beta Example

current α, β, passed to kid=C



Longer Alpha-Beta Example

see first leaf,
MIN updates β

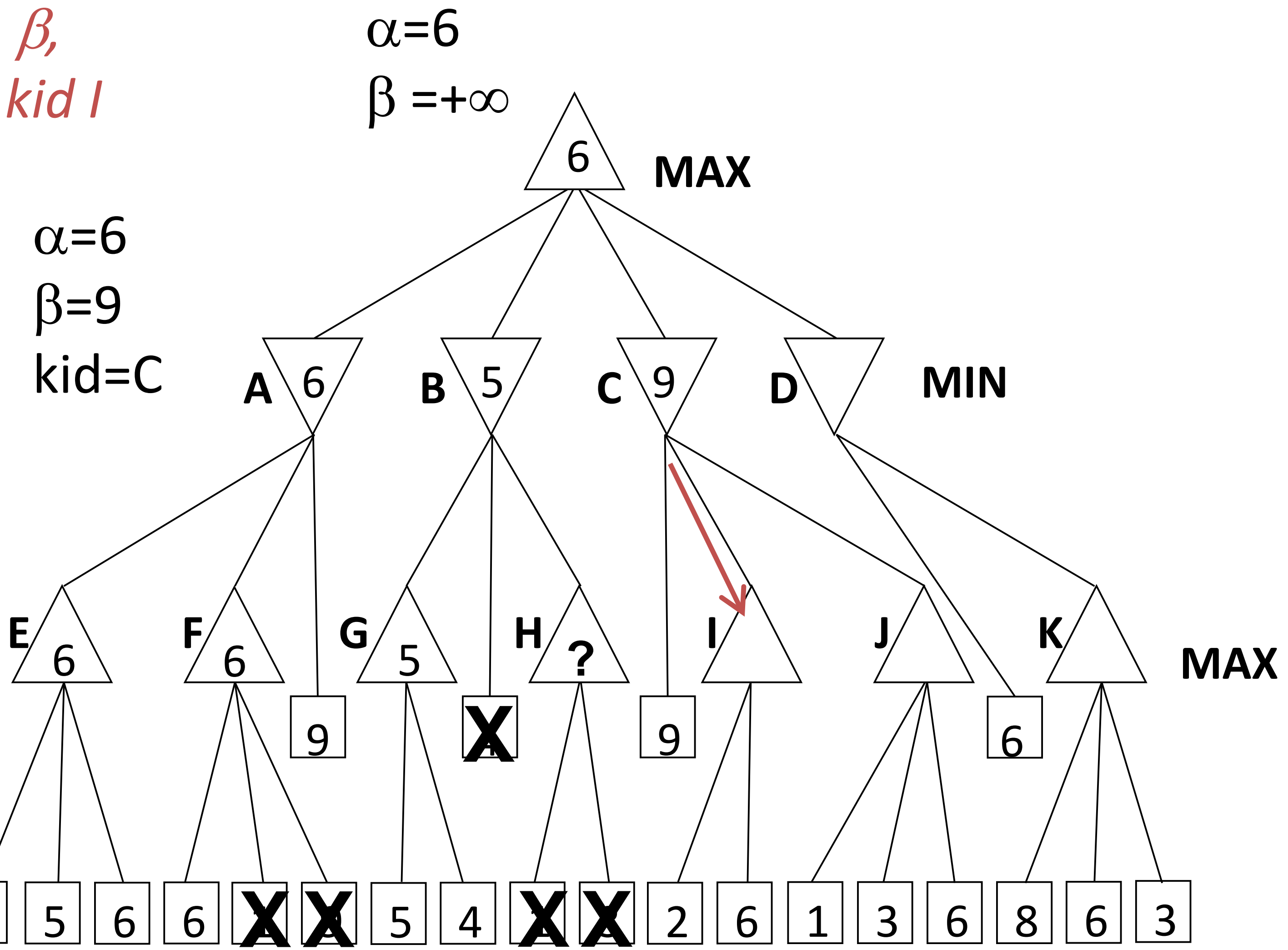


Longer Alpha-Beta Example

current α, β, passed to kid I

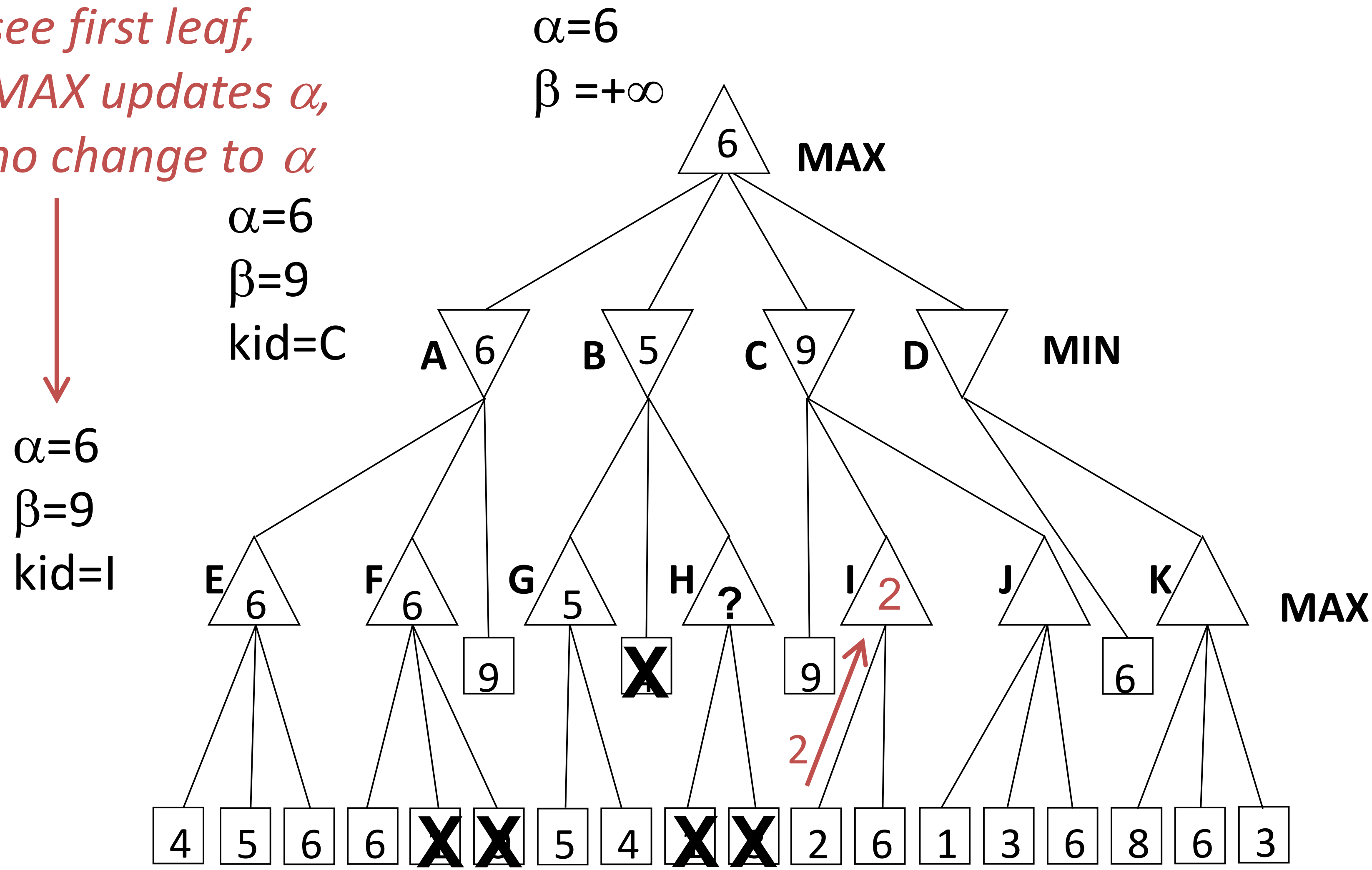
↓

α=6
β=9
kid=I



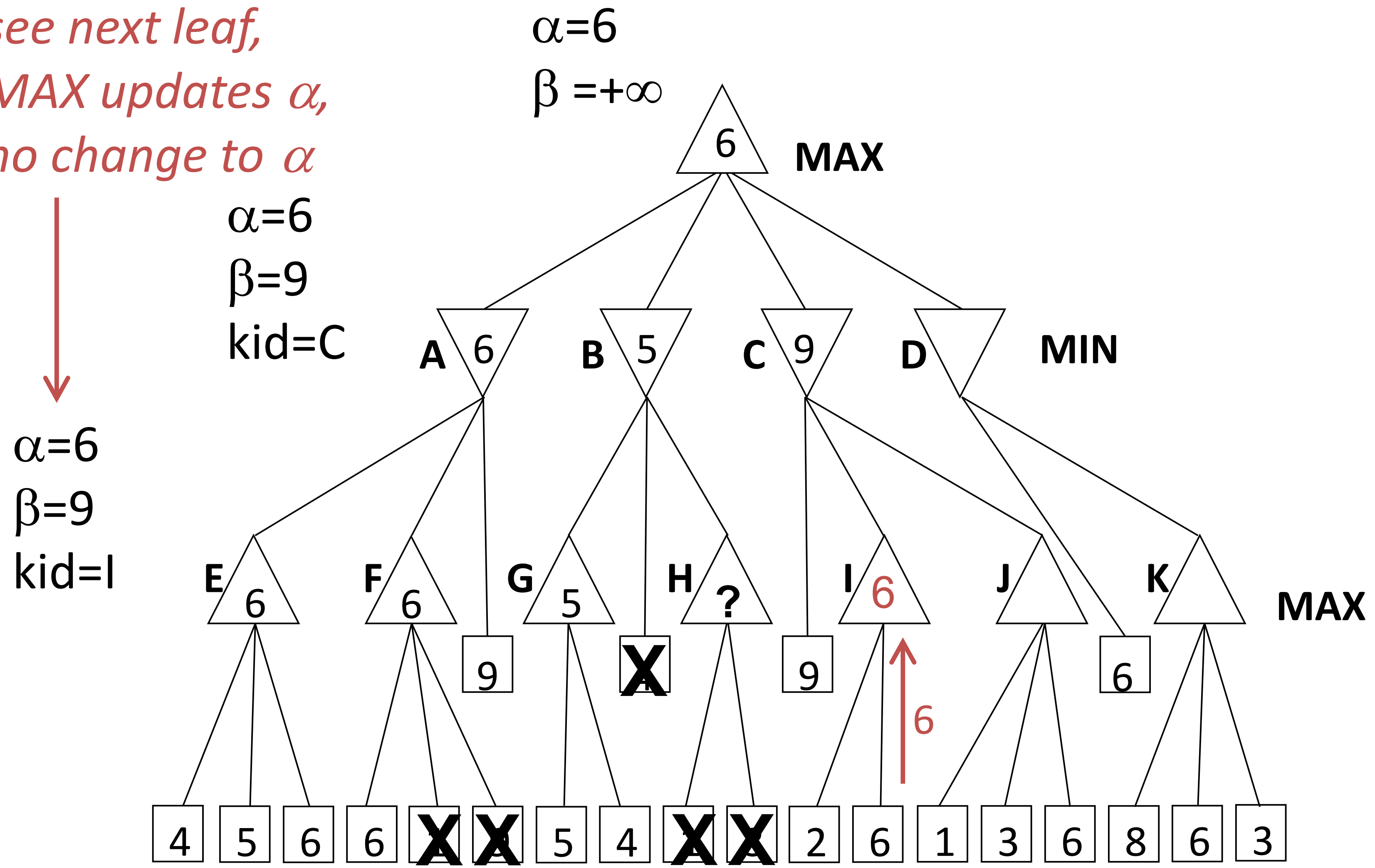
Longer Alpha-Beta Example

see first leaf,
MAX updates α,
no change to α



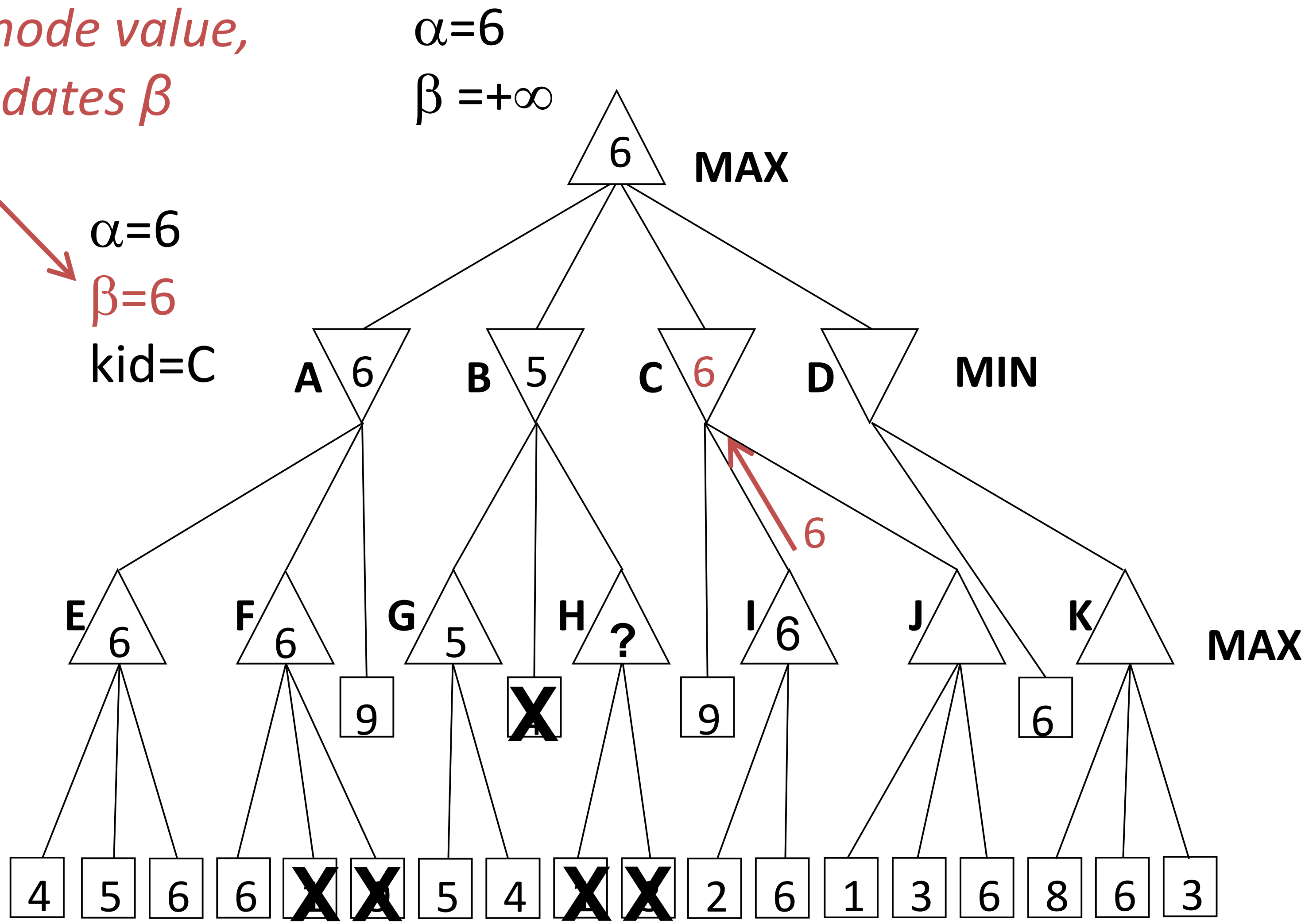
Longer Alpha-Beta Example

see next leaf,
MAX updates α,
no change to α

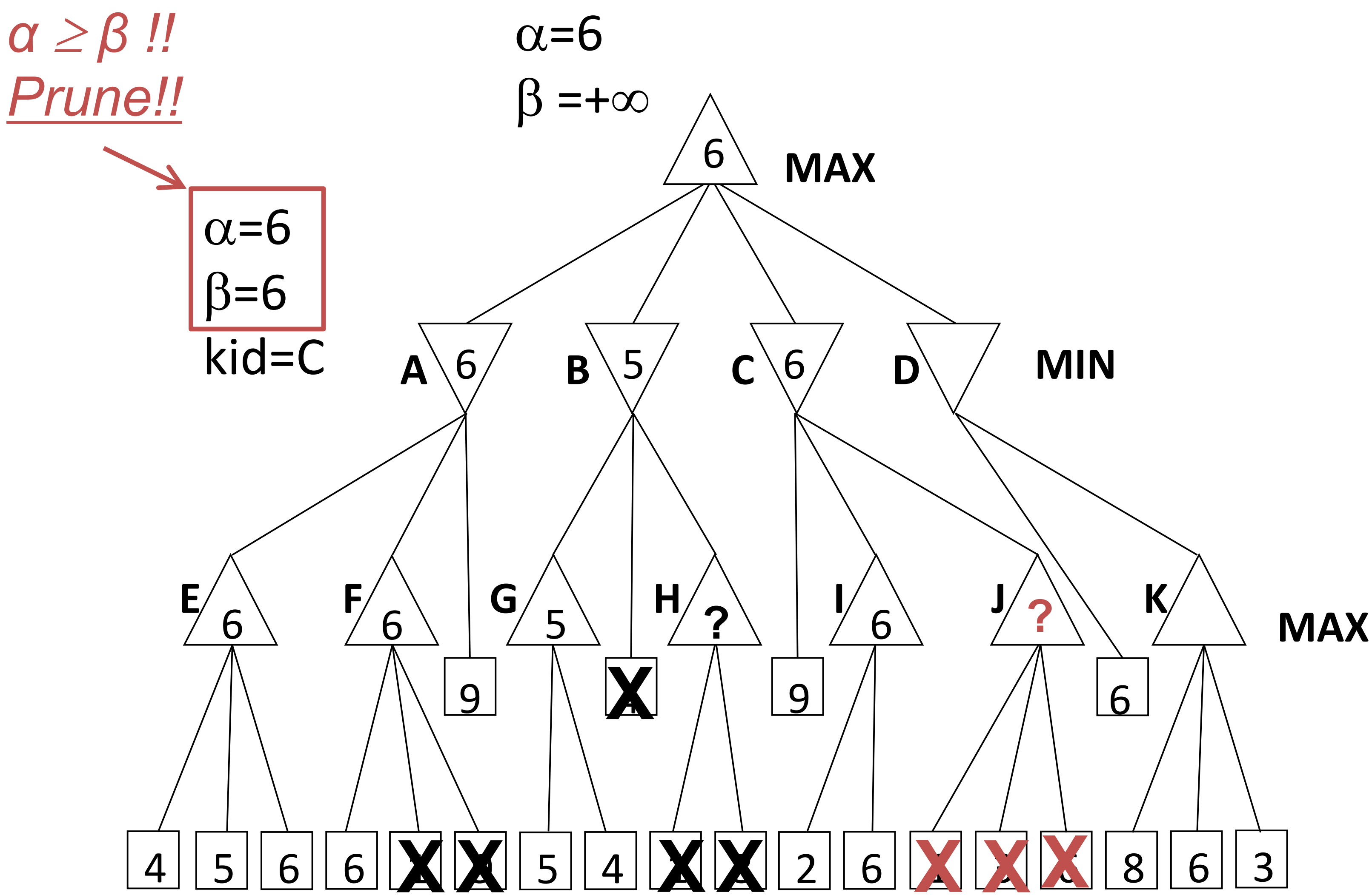


Longer Alpha-Beta Example

return node value, MIN updates β

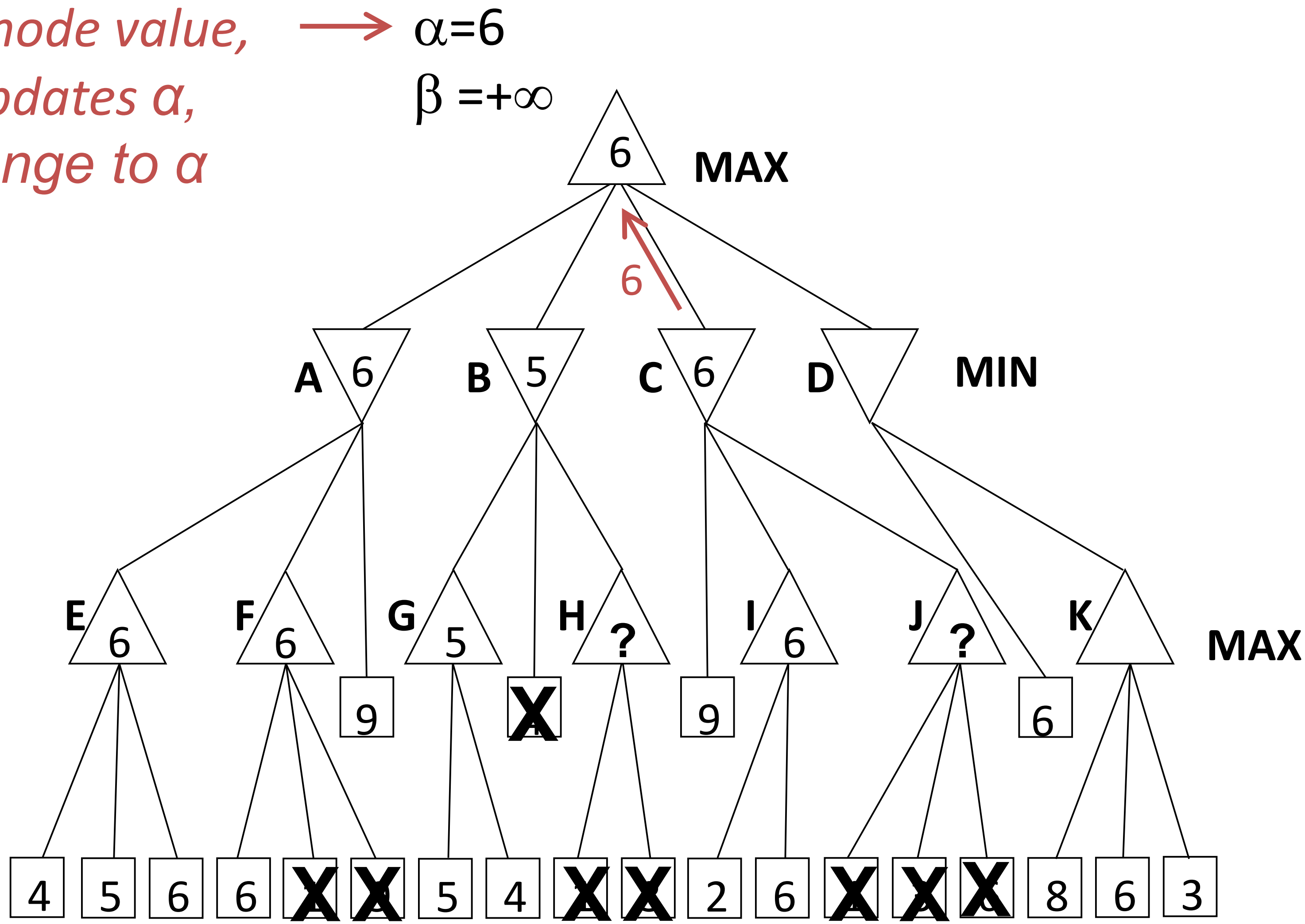


Longer Alpha-Beta Example

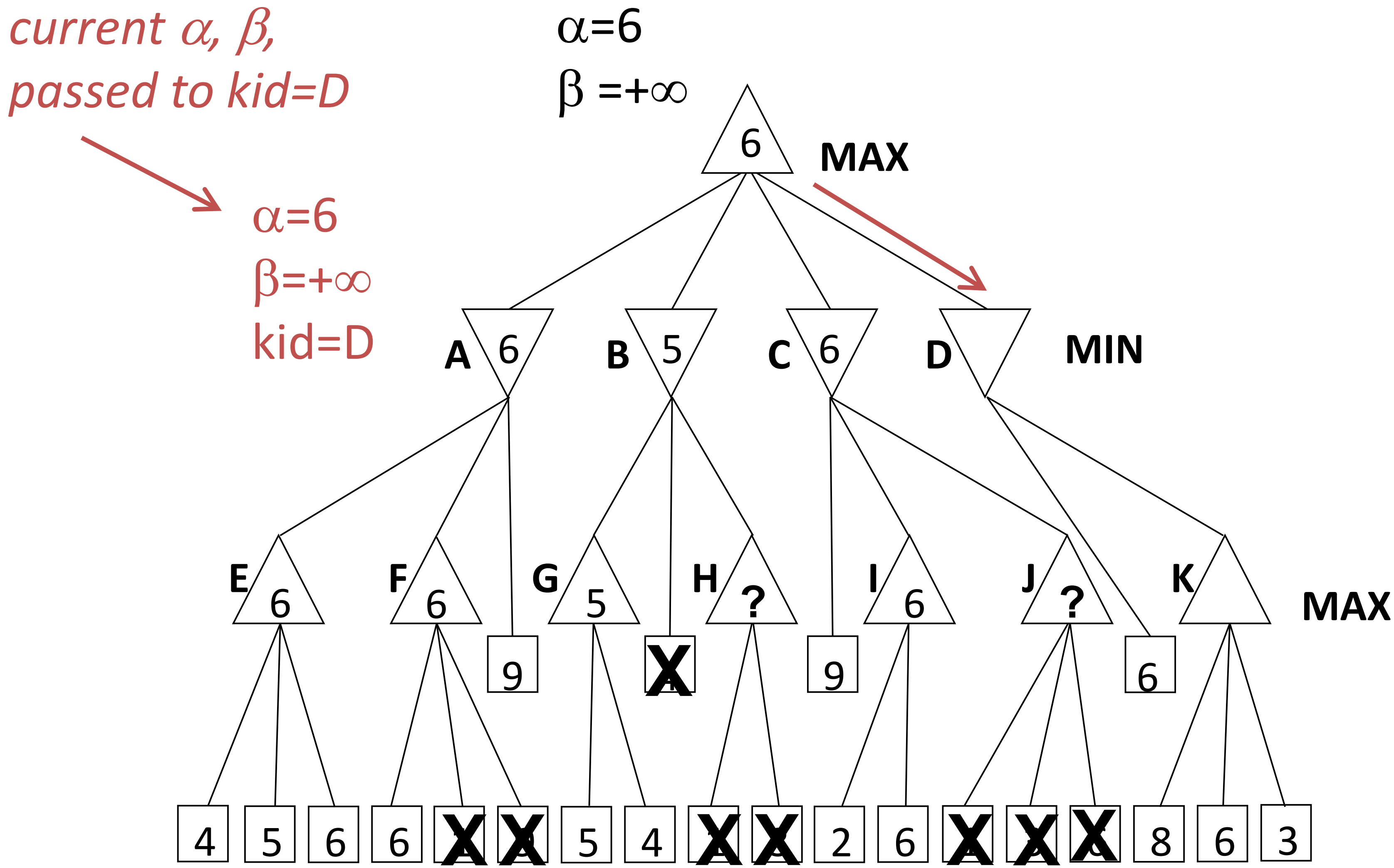


Longer Alpha-Beta Example

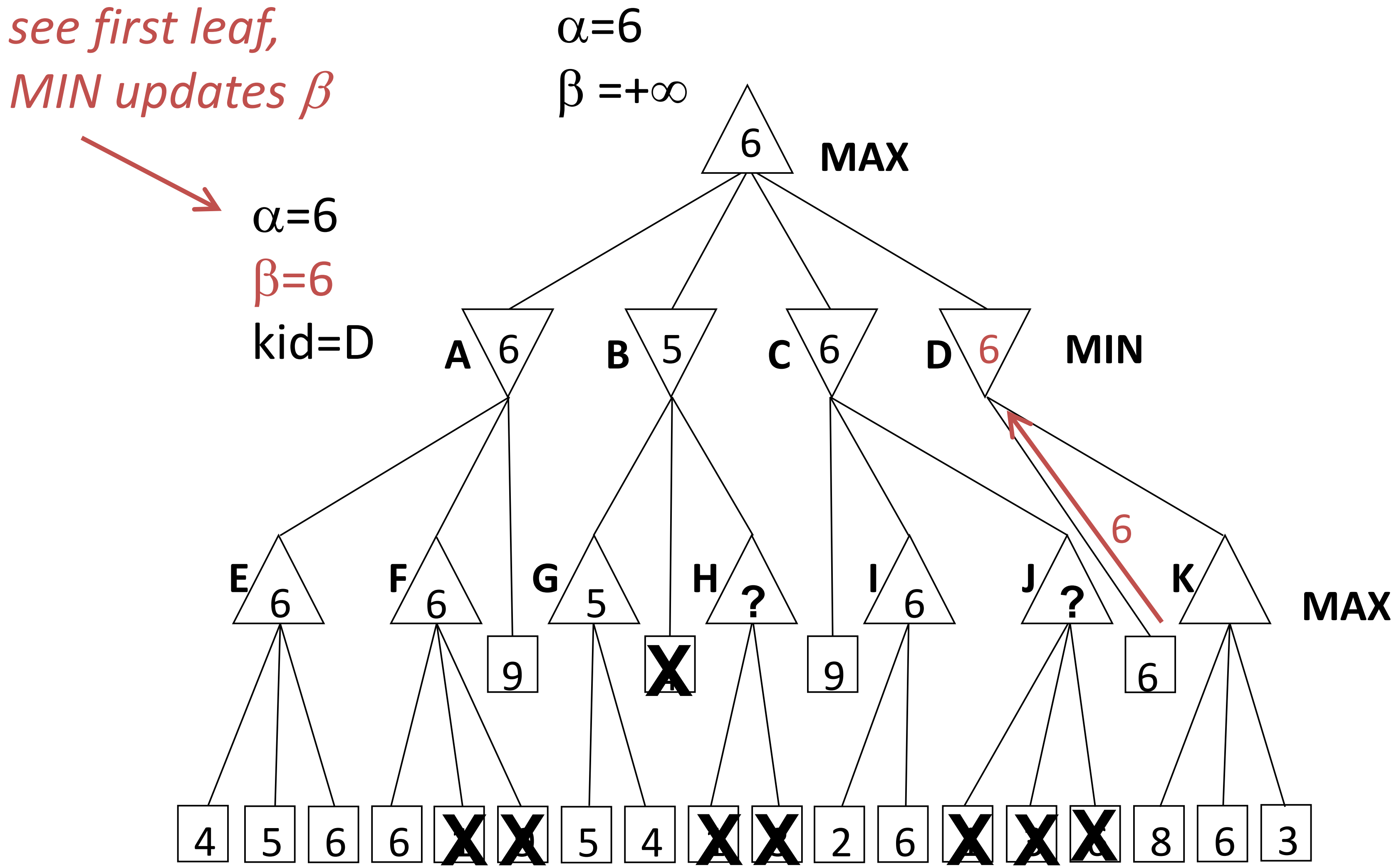
return node value,
MAX updates α,
no change to α



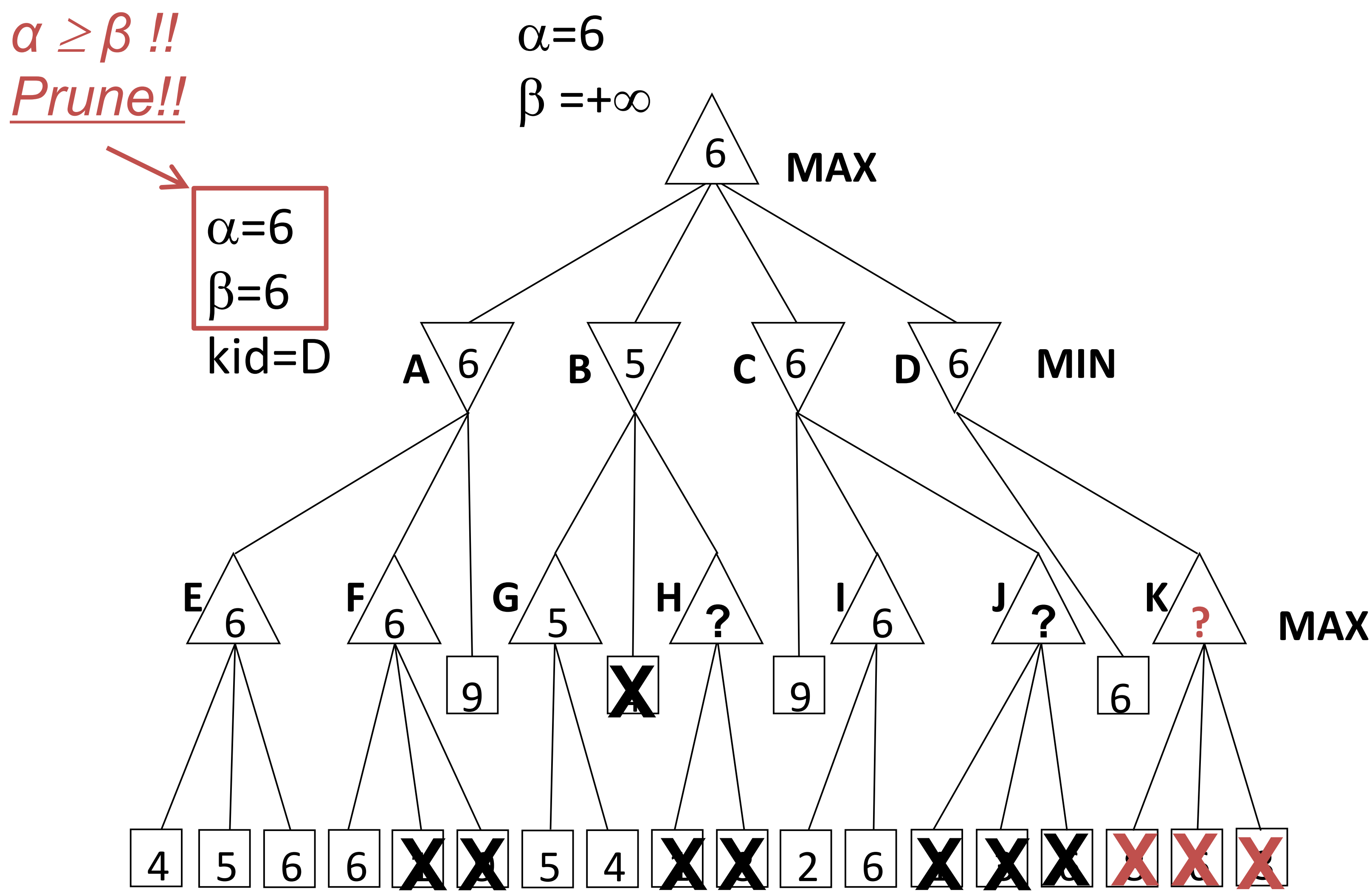
Longer Alpha-Beta Example



Longer Alpha-Beta Example



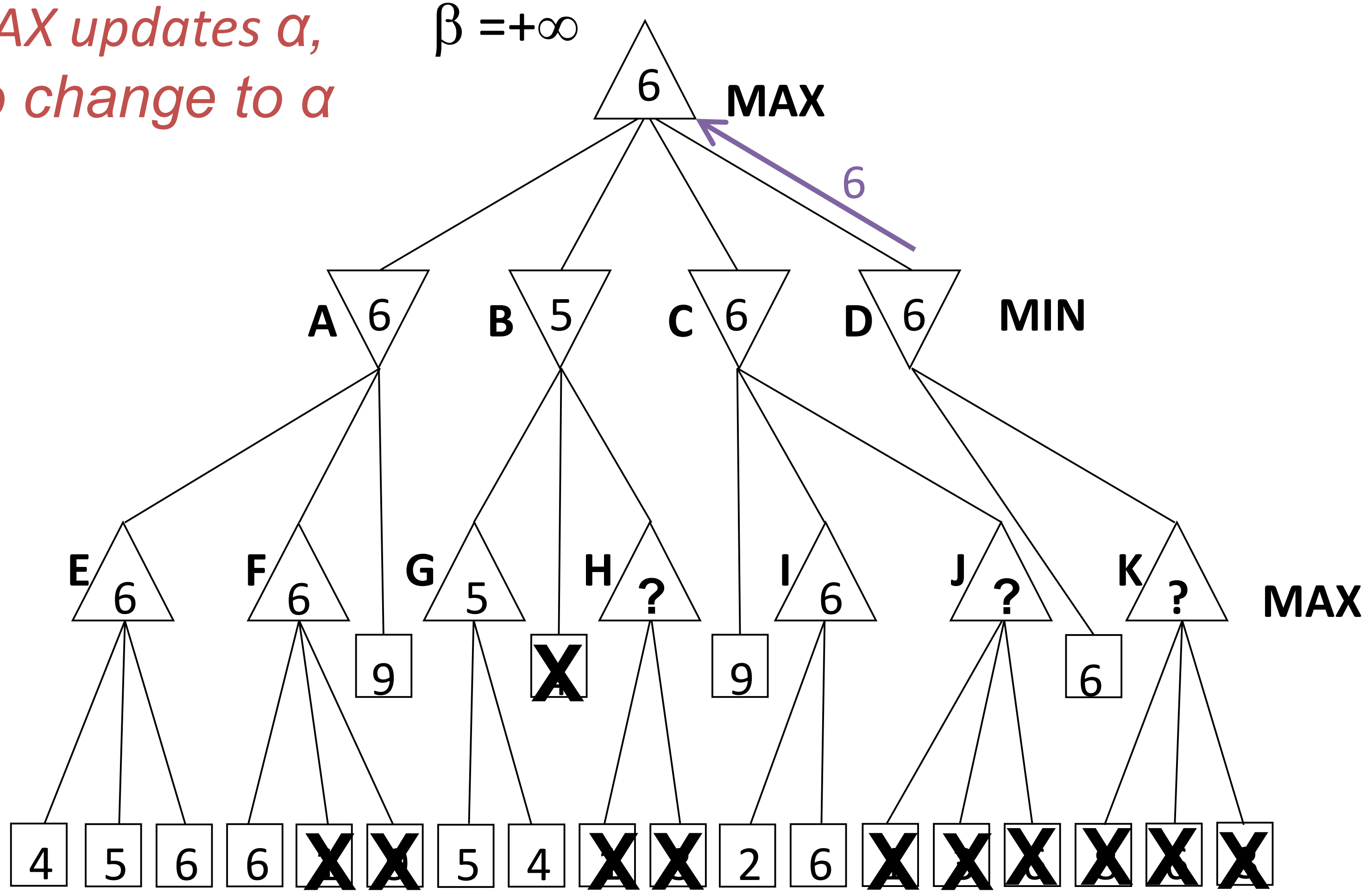
Longer Alpha-Beta Example



Alpha-Beta Example #2

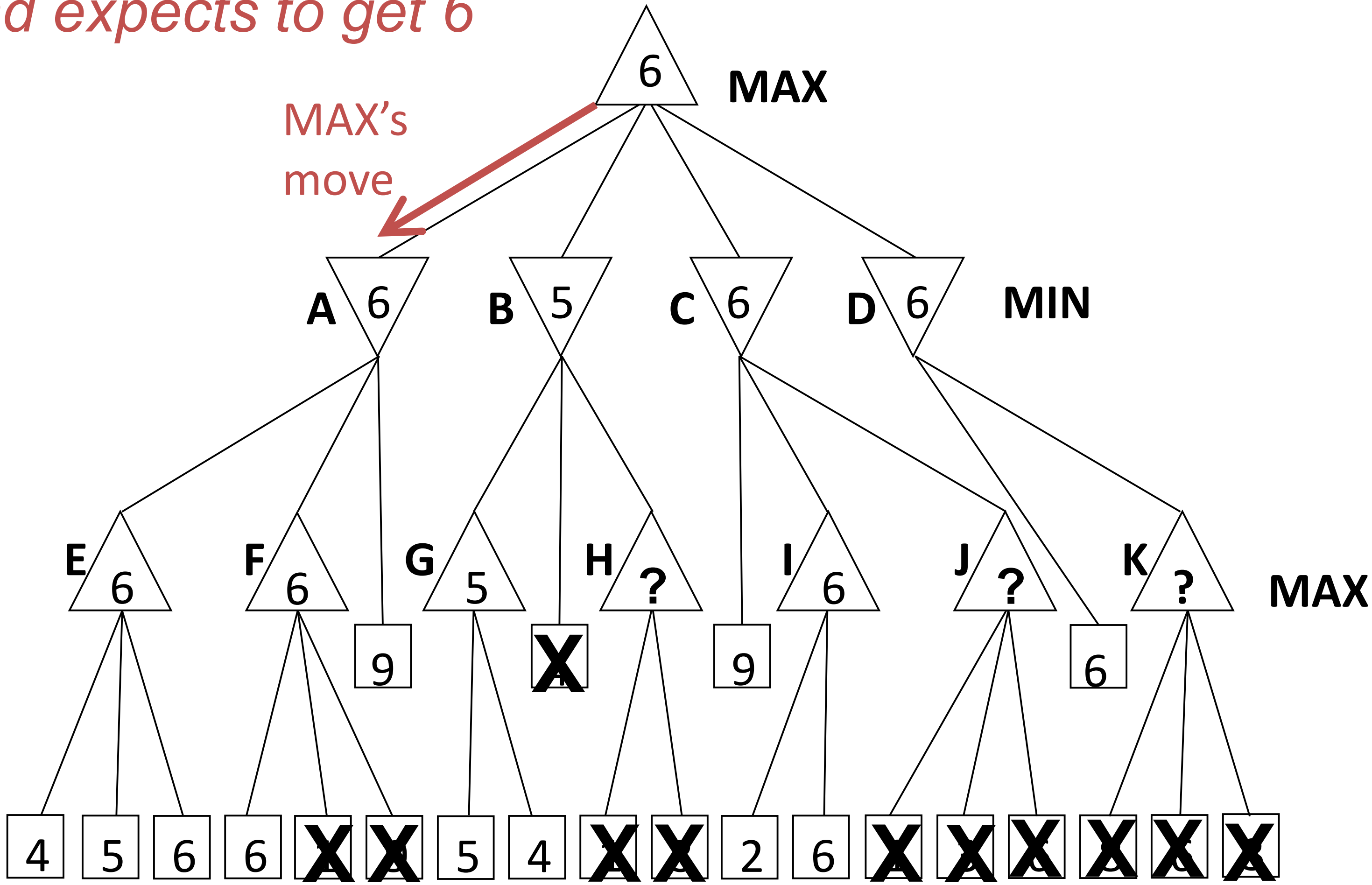
return node value,
MAX updates α,
no change to α

α=6
β =+∞



Alpha-Beta Example #2

MAX moves to A,
and expects to get 6



Although we may have changed some internal branch node return values, the final root action and expected outcome are identical to if we had not done alpha-beta pruning. Internal values may change; root values do not.

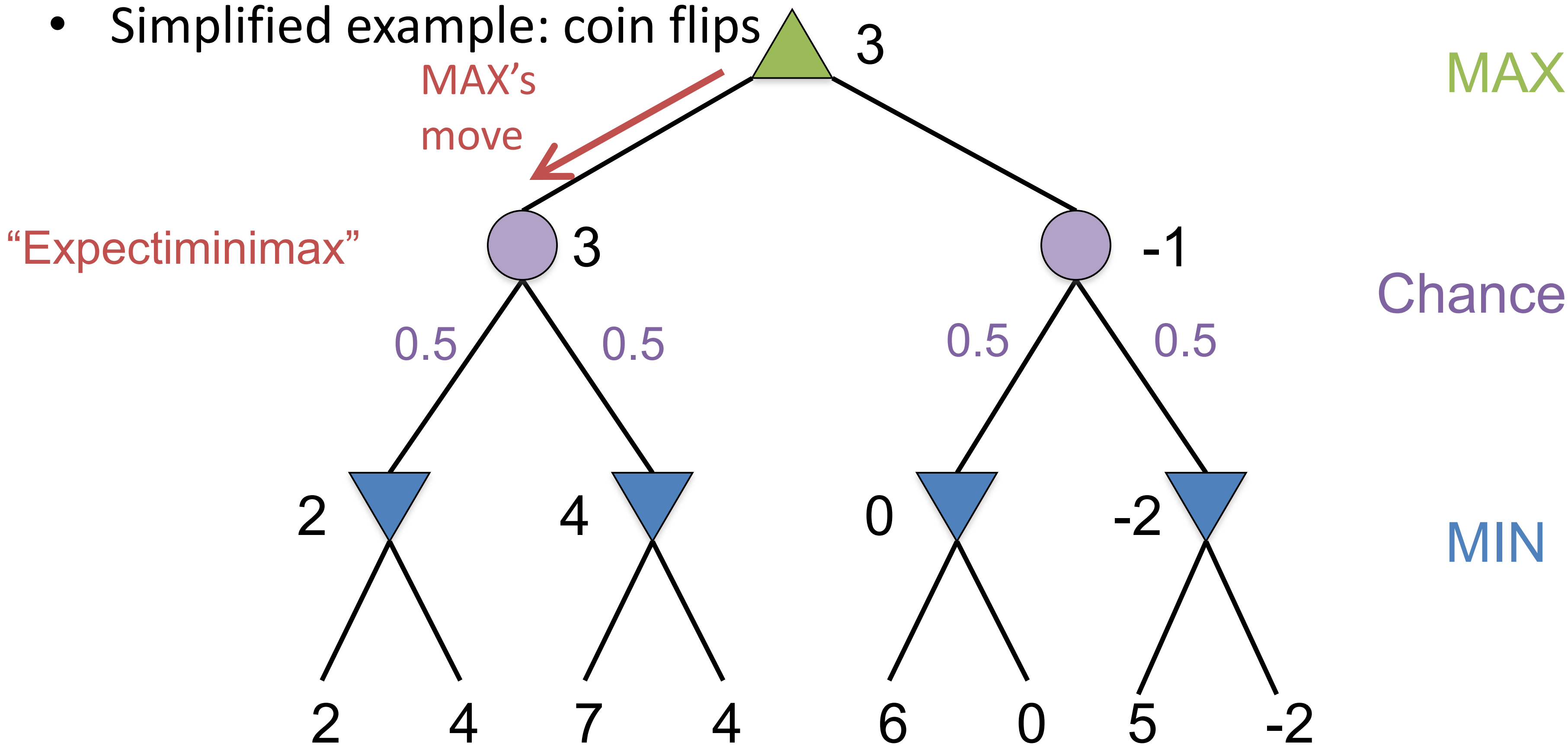
Nondeterministic games

- Ex: Backgammon
  - Roll dice to determine how far to move (random)
  - Player selects which checkers to move (strategy)

https://commons.wikimedia.org/wiki/File:Backgammon_lg.jpg

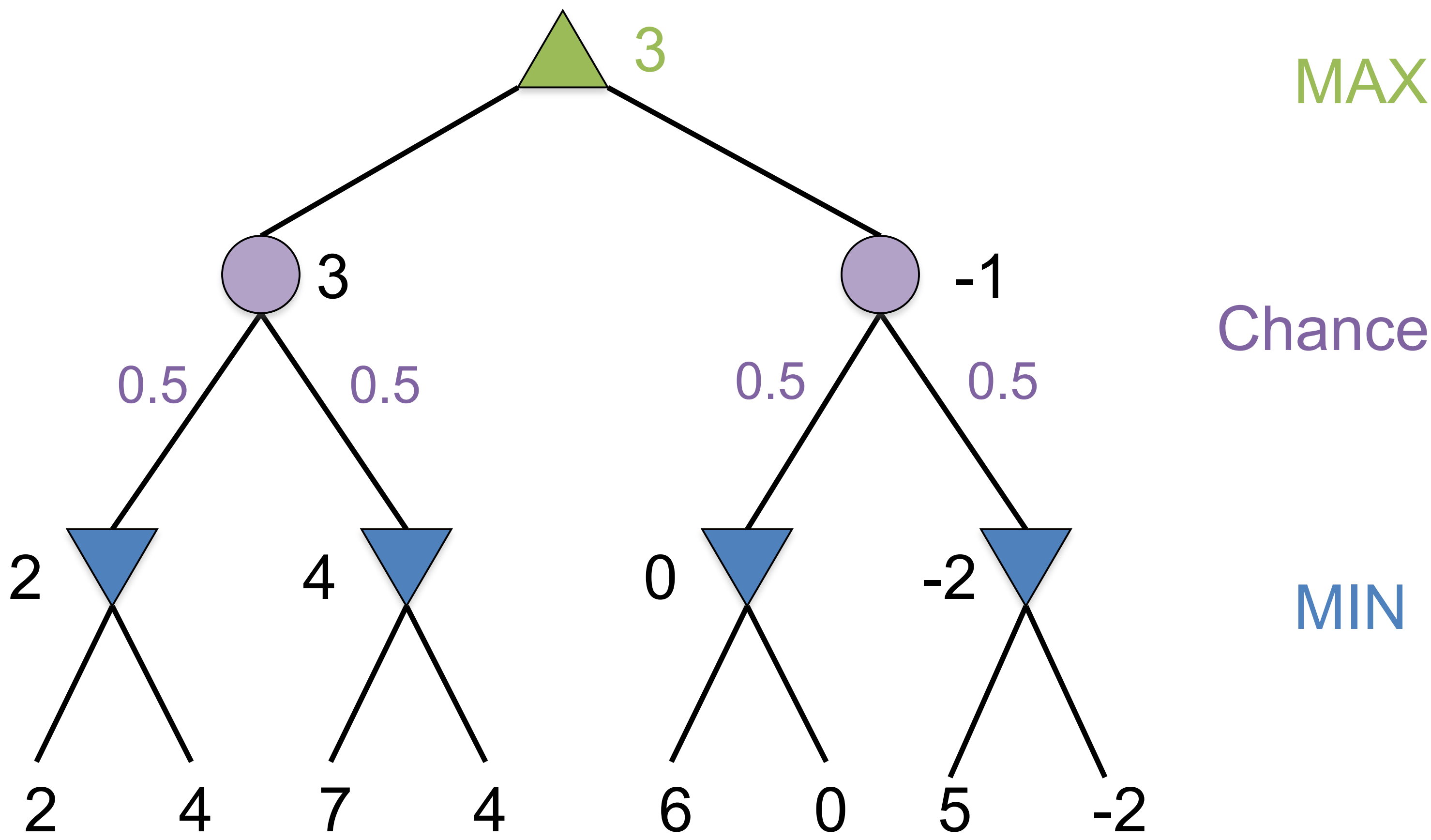
Nondeterministic games

- Chance (random effects) due to dice, card shuffle, …
- Chance nodes: expectation (weighted average) of successors
- Simplified example: coin flips



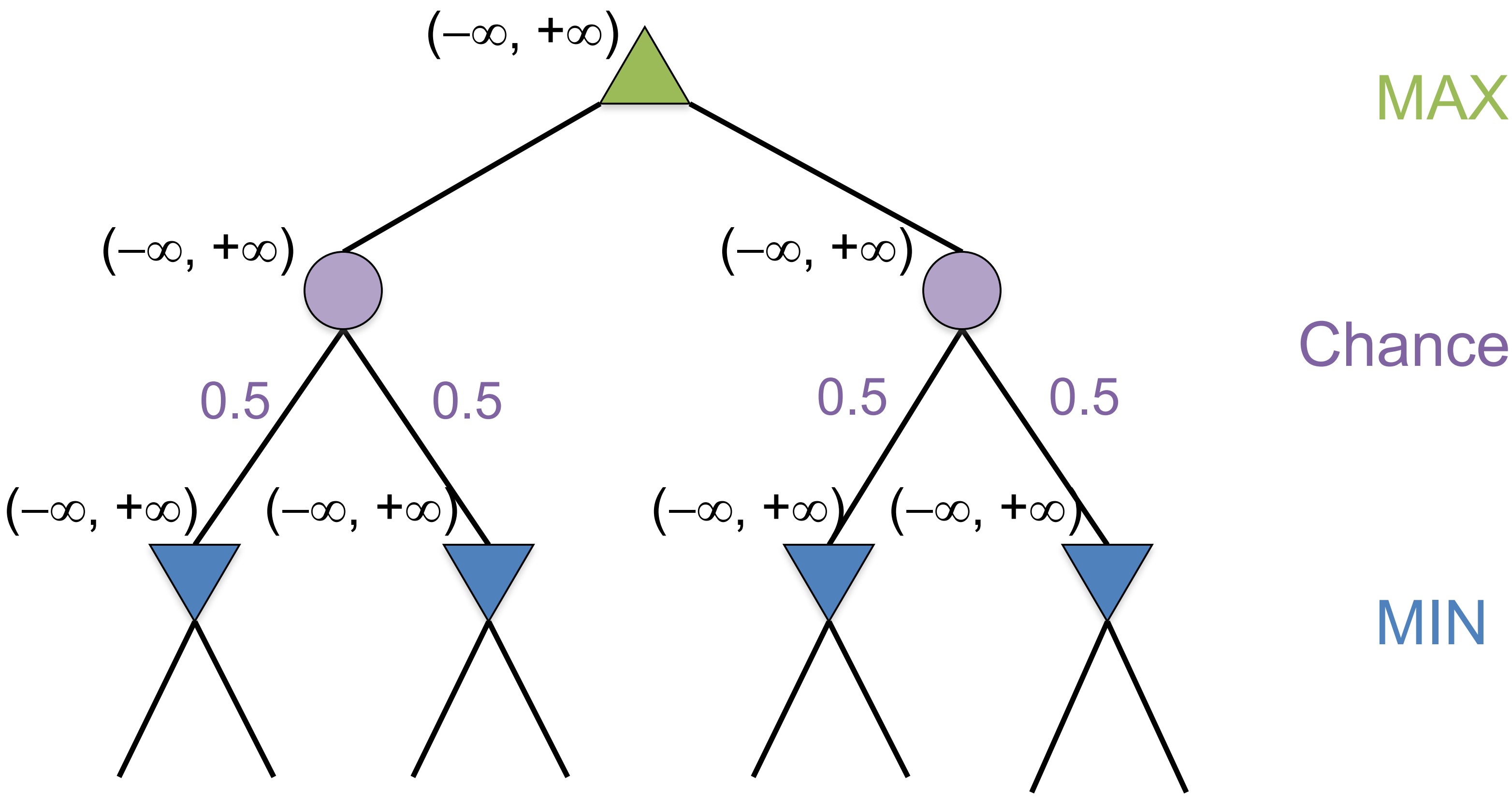
Pruning in nondeterministic games

- Can still apply a form of alpha-beta pruning



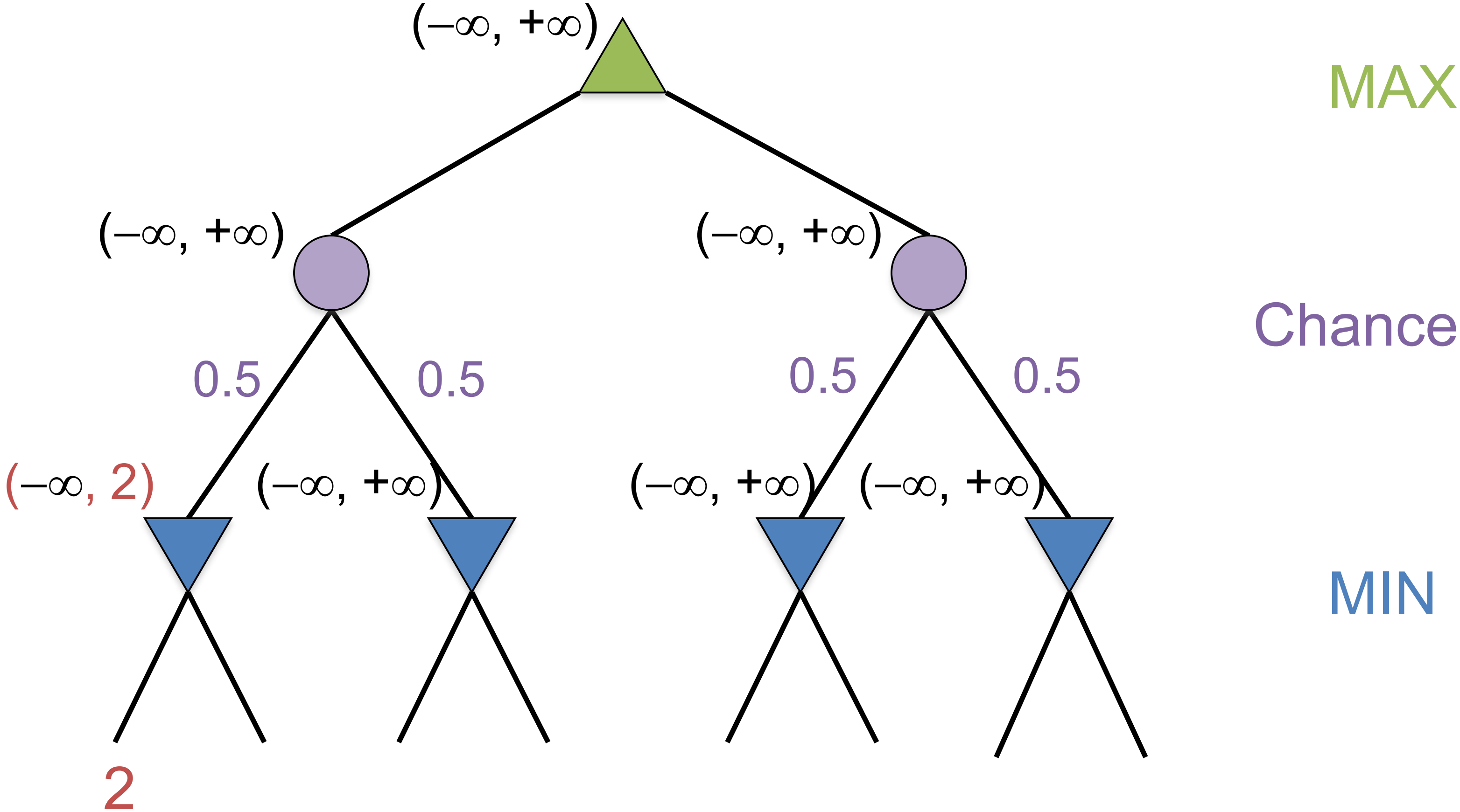
Pruning in nondeterministic games

- Can still apply a form of alpha-beta pruning



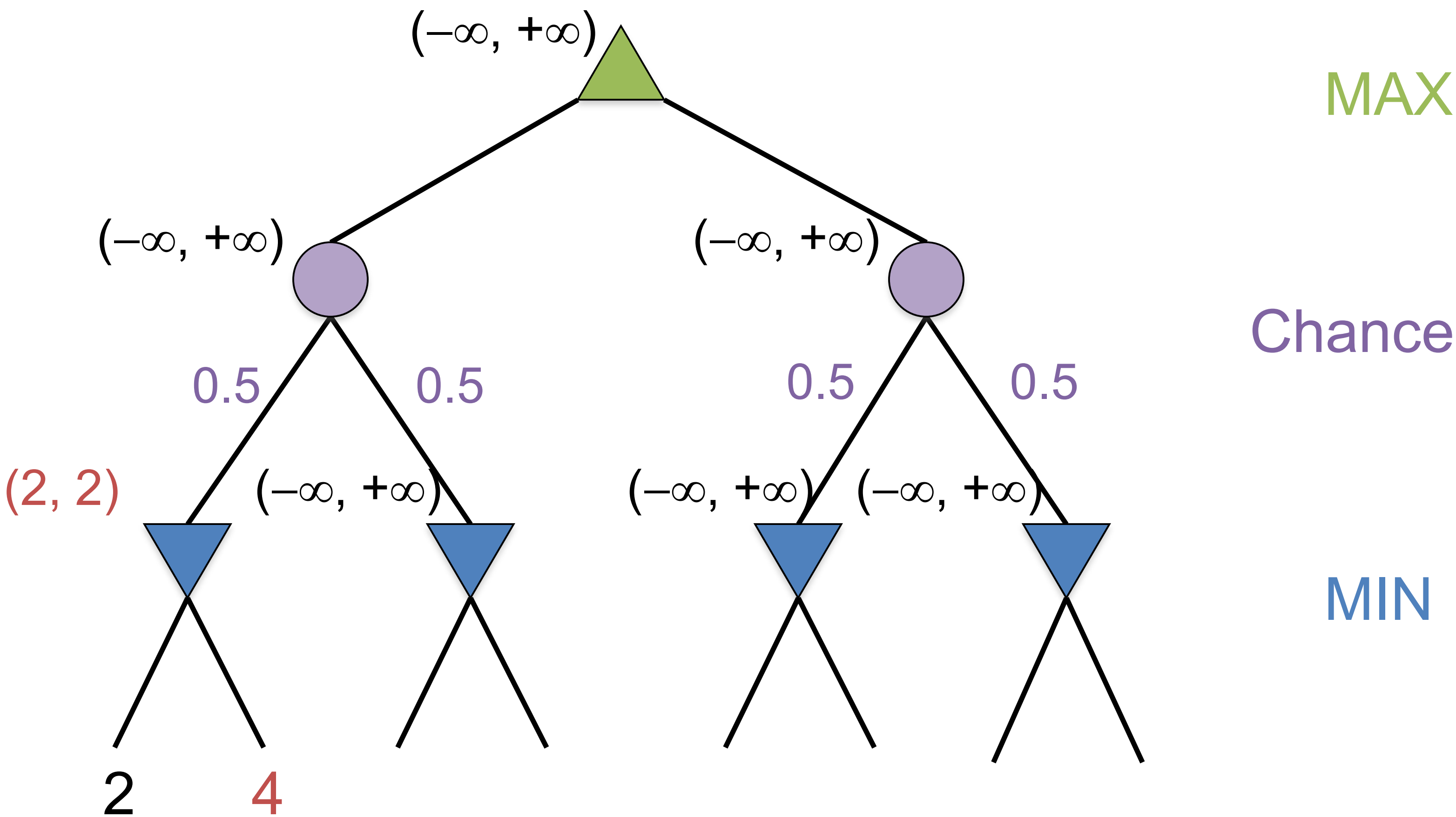
Pruning in nondeterministic games

- Can still apply a form of alpha-beta pruning



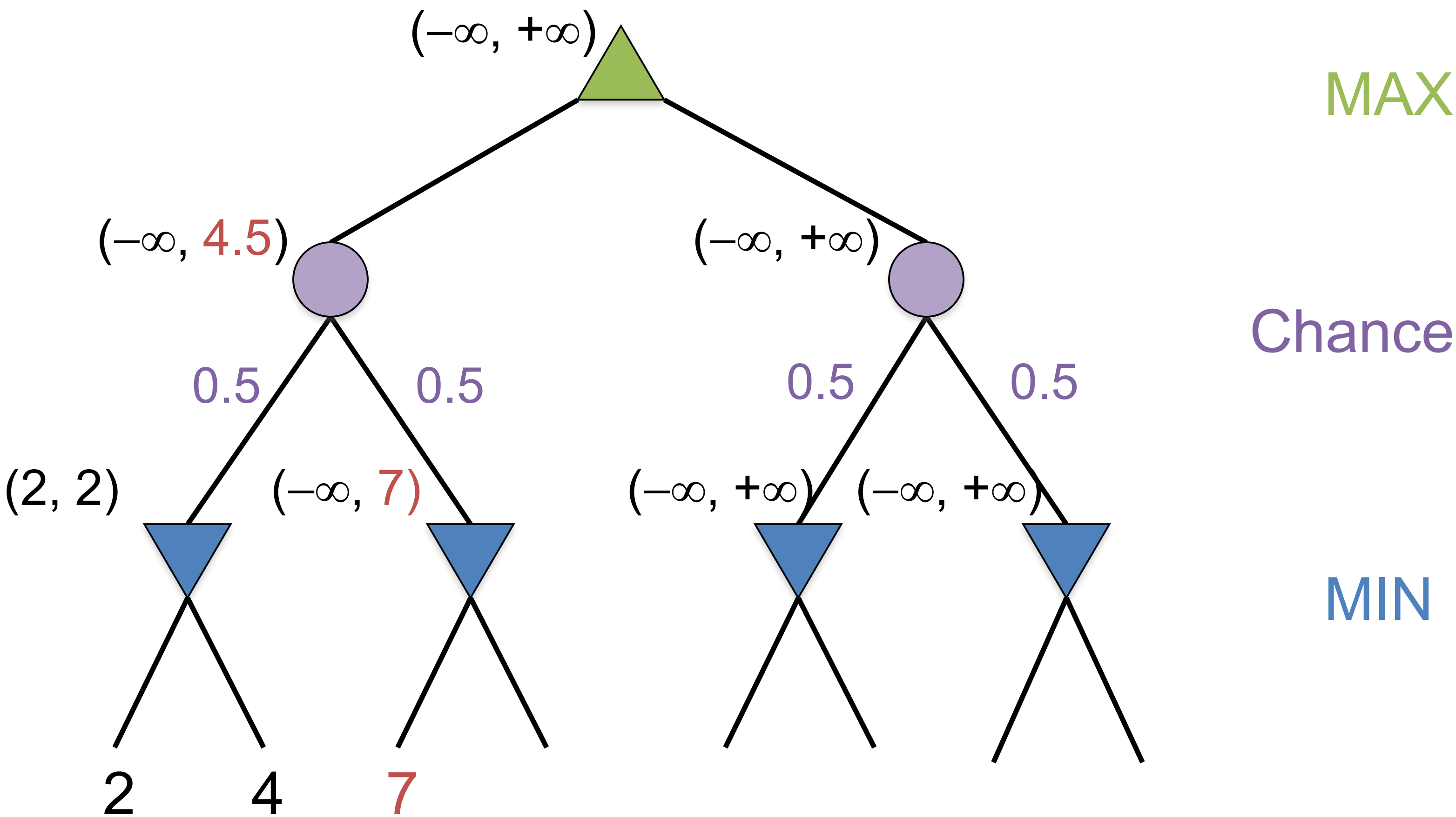
Pruning in nondeterministic games

- Can still apply a form of alpha-beta pruning



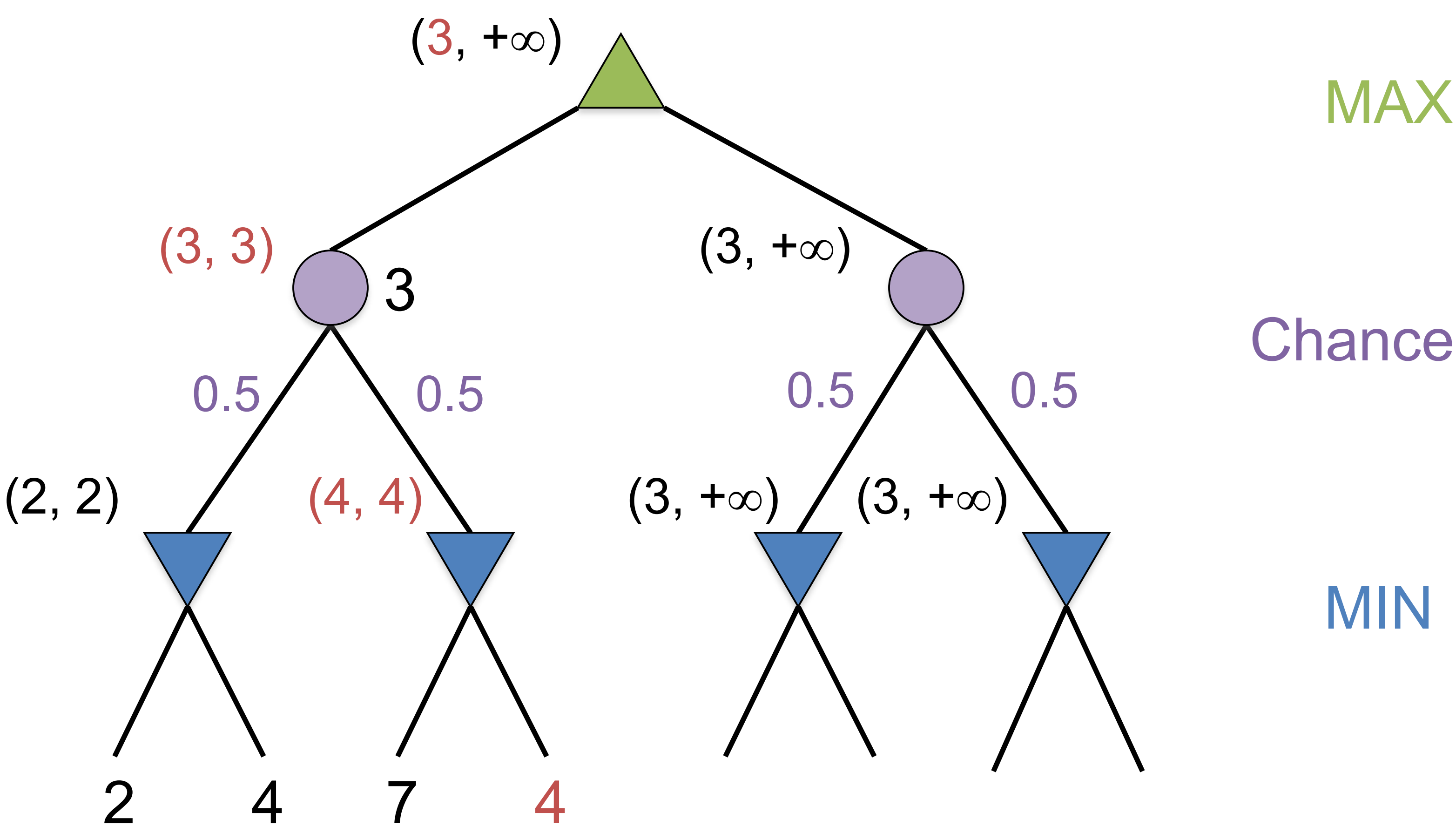
Pruning in nondeterministic games

- Can still apply a form of alpha-beta pruning



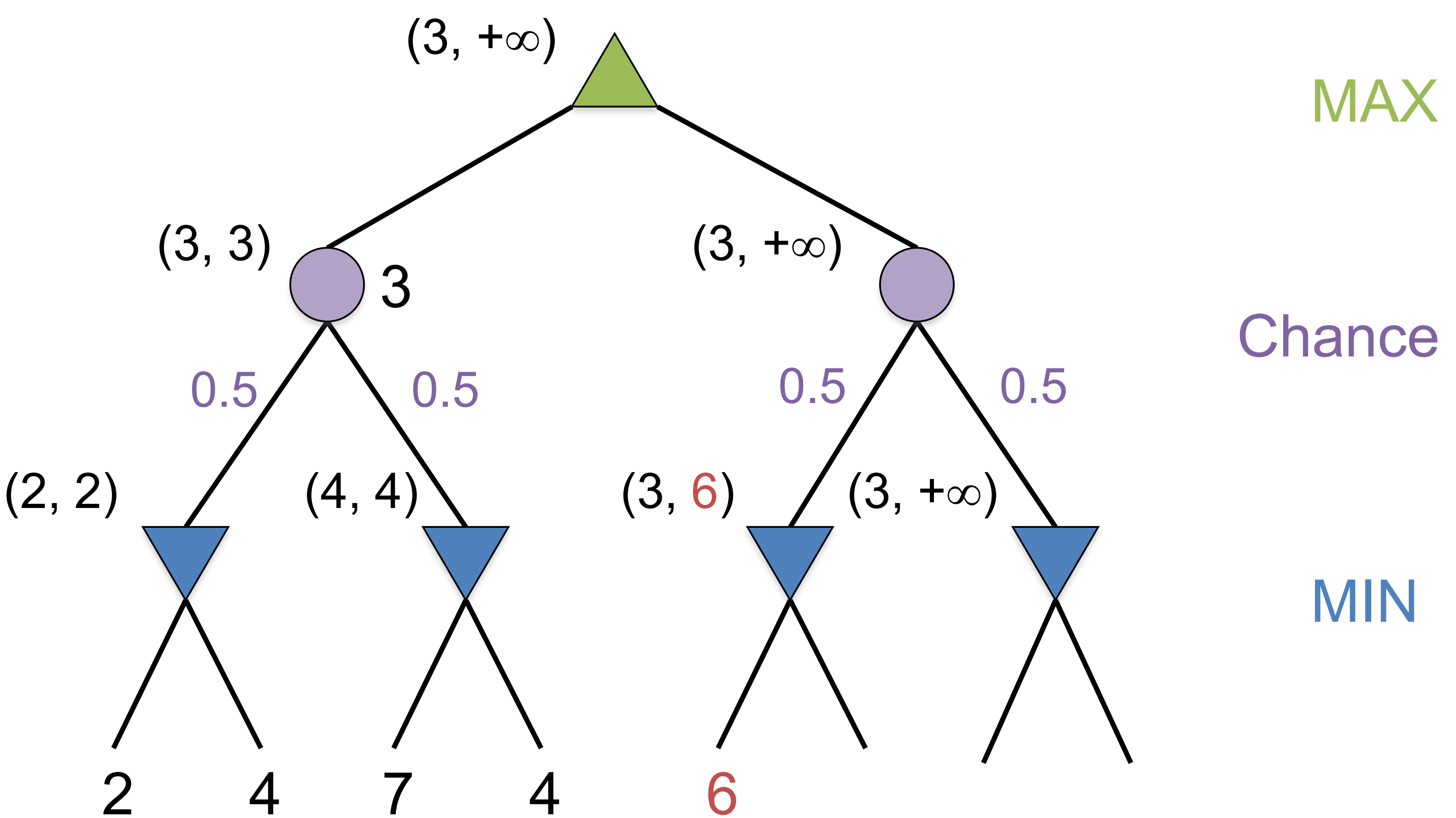
Pruning in nondeterministic games

- Can still apply a form of alpha-beta pruning



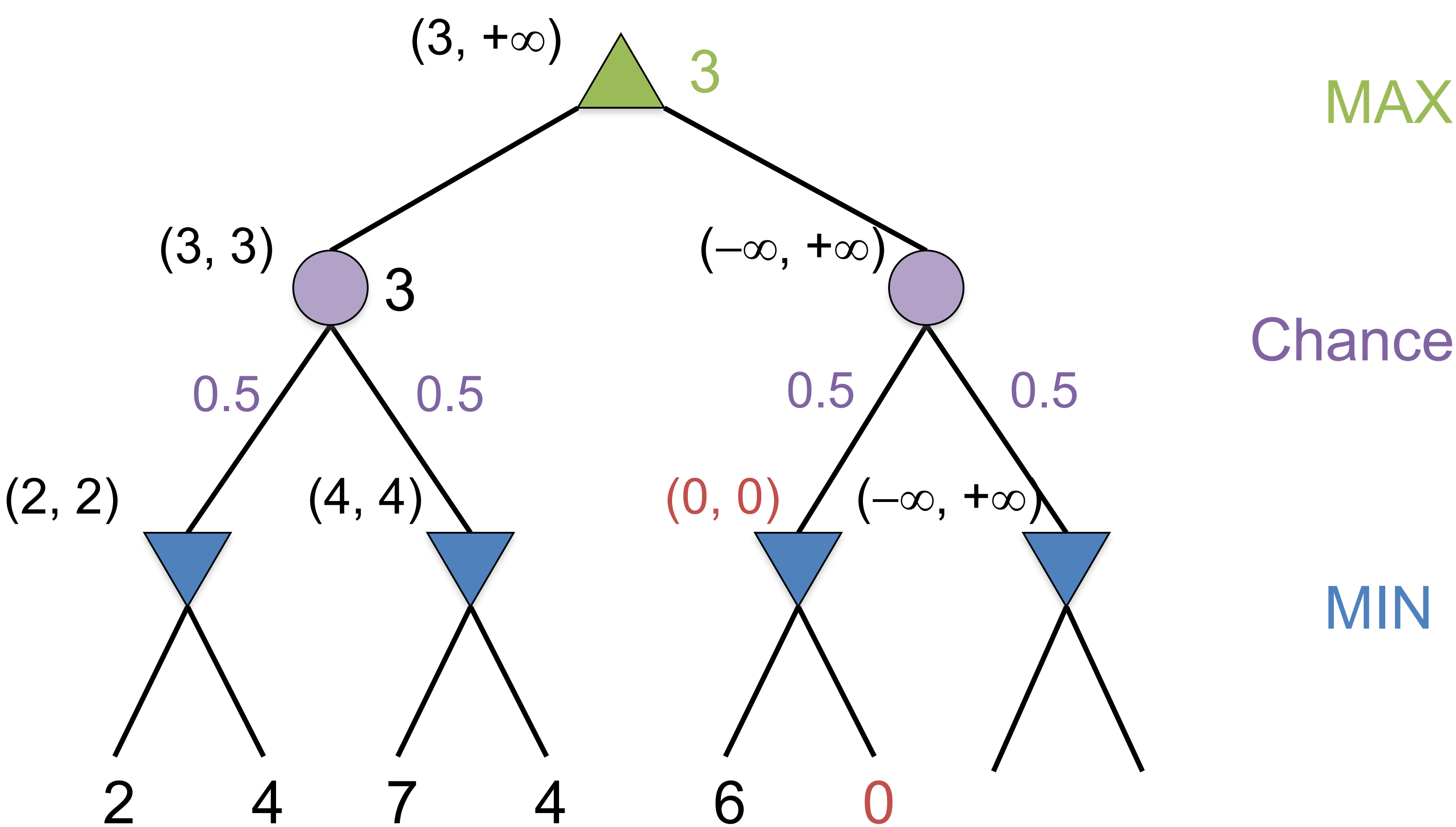
Pruning in nondeterministic games

- Can still apply a form of alpha-beta pruning



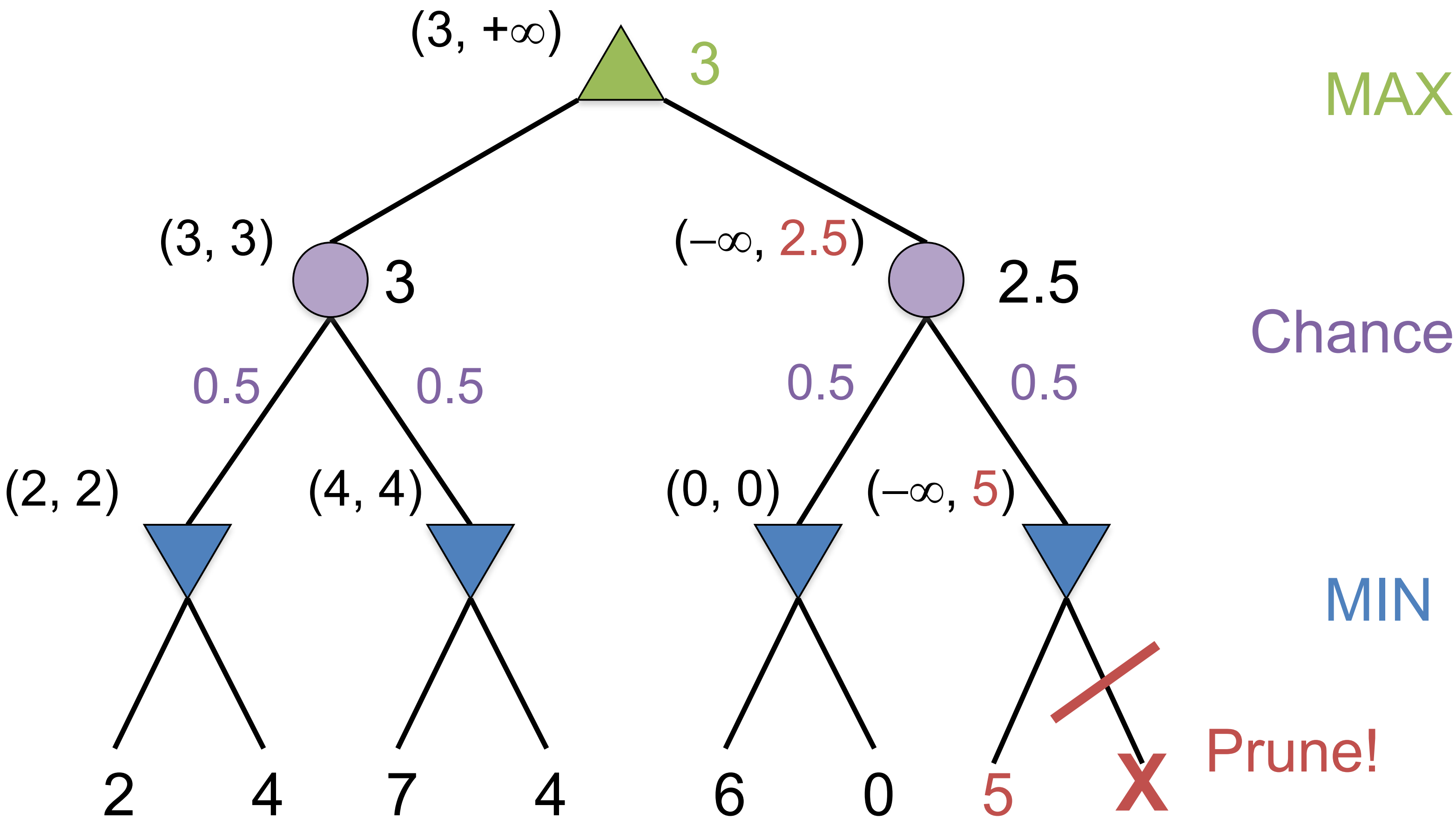
Pruning in nondeterministic games

- Can still apply a form of alpha-beta pruning



Pruning in nondeterministic games

- Can still apply a form of alpha-beta pruning

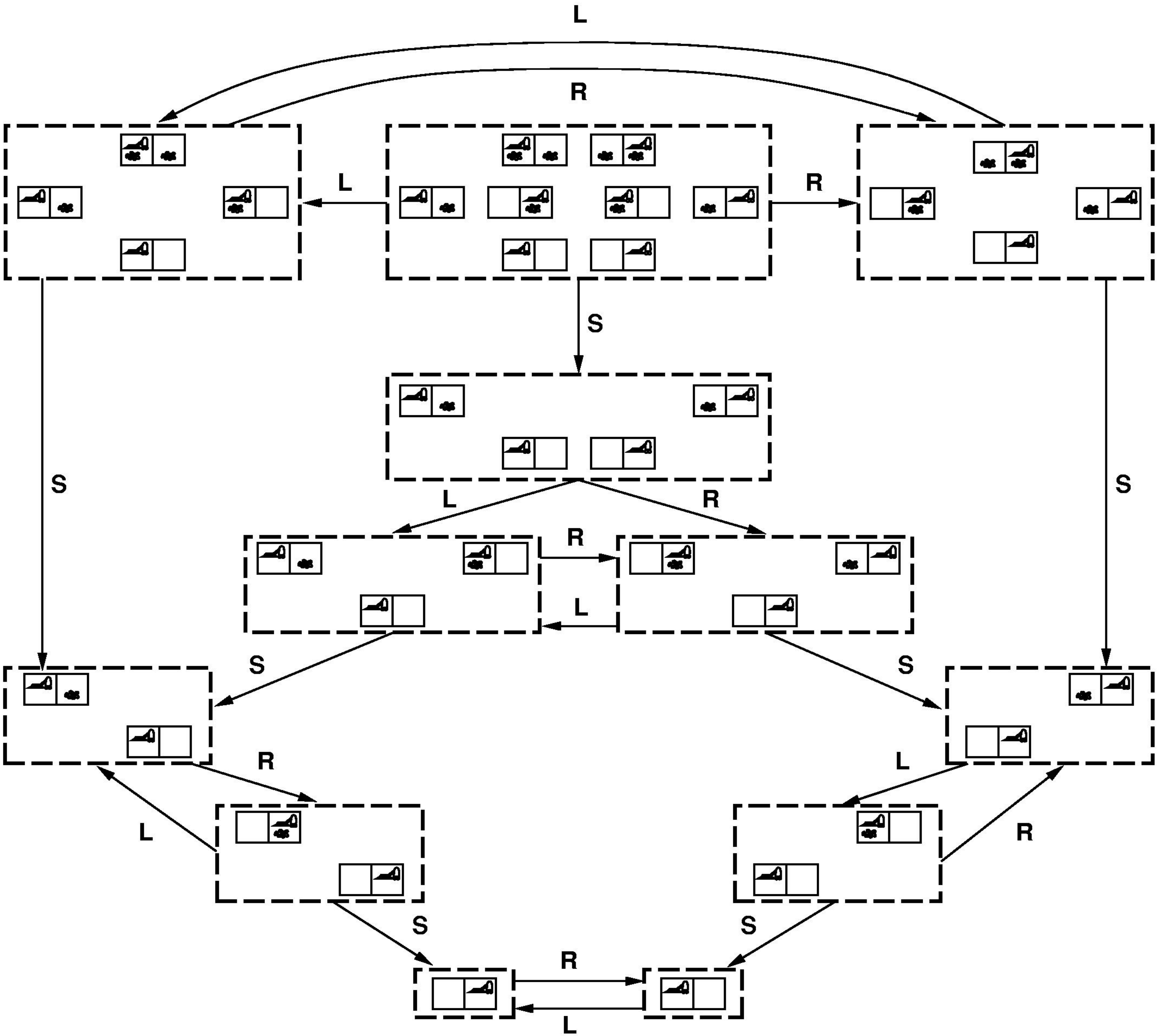


Partially observable games

- R&N Chapter 5.6 – "The fog of war"
- Background: R&N, Chapter 4.3-4
  - Searching with Nondeterministic Actions/Partial Observations

- Search through Belief States (see Fig. 4.14)
  - Agent's current belief about which states it might be in, given the sequence of actions & percepts to that point
- Actions(b) = ?? Union? Intersection?
  - Tricky: an action legal in one state may be illegal in another
  - Is an illegal action a NO-OP? or the end of the world?
- Transition Model:
  - Result(b,a) = { s' : s' = Result(s, a) and s is a state in b }
- Goaltest(b) = every state in b is a goal state

Belief States for Unobservable Vacuum World



Partially observable games

- R&N Chapter 5.6
- Player's current node is a belief state
- Player's move (action) generates child belief state
- Opponent's move is replaced by Percepts(s)
  - Each possible percept leads to the belief state that is consistent with that percept
- Strategy = a move for every possible percept sequence
- Minimax returns the worst state in the belief state

- <u>Many more complications and possibilities!!</u>
  - Opponent may select a move that is not optimal, but instead minimizes the information transmitted, or confuses the opponent
  - May not be reasonable to consider ALL moves; open P-QR3??
- **See R&N, Chapter 5.6, for more info**

The State of Play

- Checkers:
  - Chinook ended 40-year-reign of human world champion Marion Tinsley in 1994.
- Chess:
  - Deep Blue defeated human world champion Garry Kasparov in a six-game match in 1997.
- Othello:
  - human champions refuse to compete against computers:
    they are too good.
- Go:
  - AlphaGo recently (3/2016) beat 9^{th} dan Lee Sedol
  - b > 300 (!); full game tree has > 10^760 leaf nodes (!!)
- See (e.g.) http://www.cs.ualberta.ca/~games/ for more info

High branching factors

- What can we do when the search tree is too large?
  - Example: Go (b = 50 to 300+ moves per state)
  - Heuristic state evaluation (score a partial game)
- Where does this heuristic come from?
  - Hand designed
  - Machine learning on historical game patterns
  - Monte Carlo methods – play random games

Monte Carlo heuristic scoring

- Idea: play out the game randomly, and use the results as a score
  - Easy to generate & score lots of random games
  - May use 1000s of games for a node
- The basis of Monte Carlo tree search algorithms…

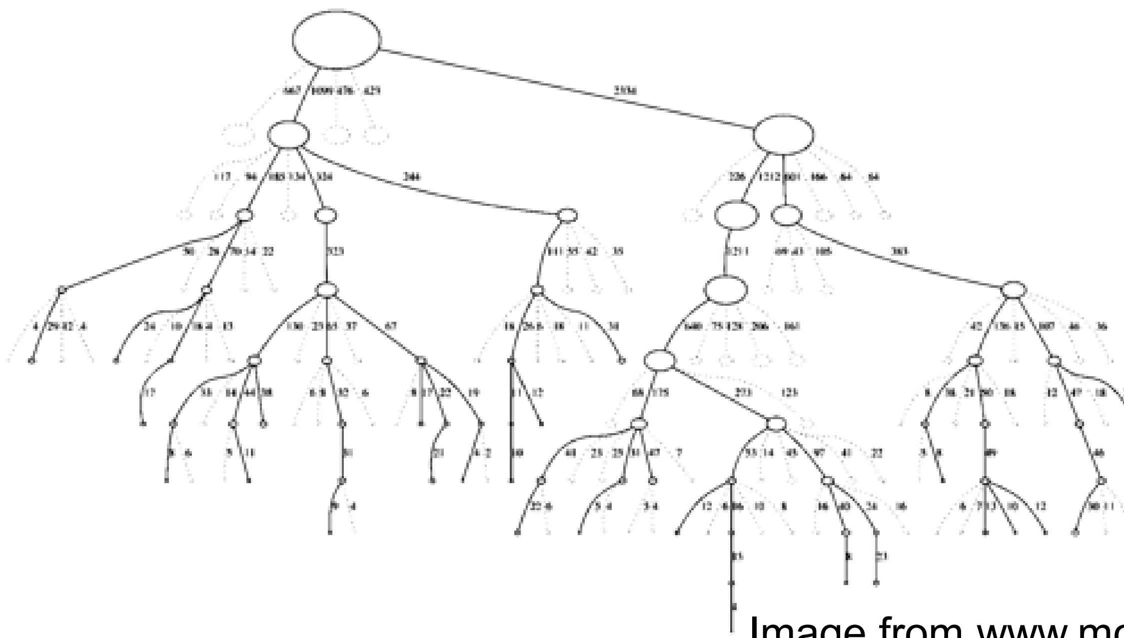


Image from www.mcts.ai

Monte Carlo Tree Search

- Should we explore the whole (top of) the tree?
  - Some moves are obviously not good…
  - Should spend time exploring / scoring promising ones
- This is a *multi-armed bandit* (MAB) problem:
- Want to spend our time on good moves
- Which moves have high payout?
  - Hard to tell – random…
- *Explore* vs. *exploit* tradeoff

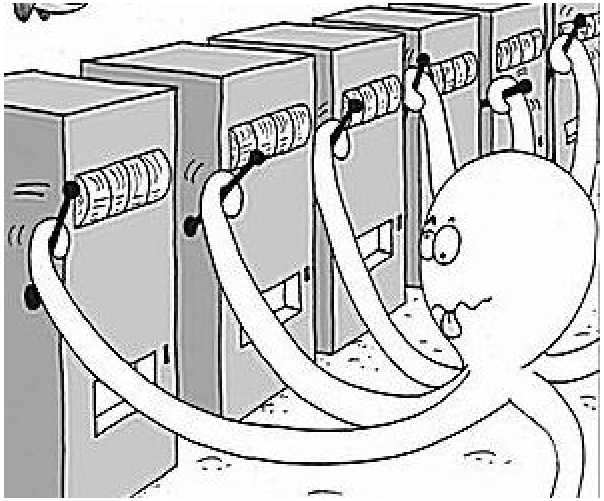
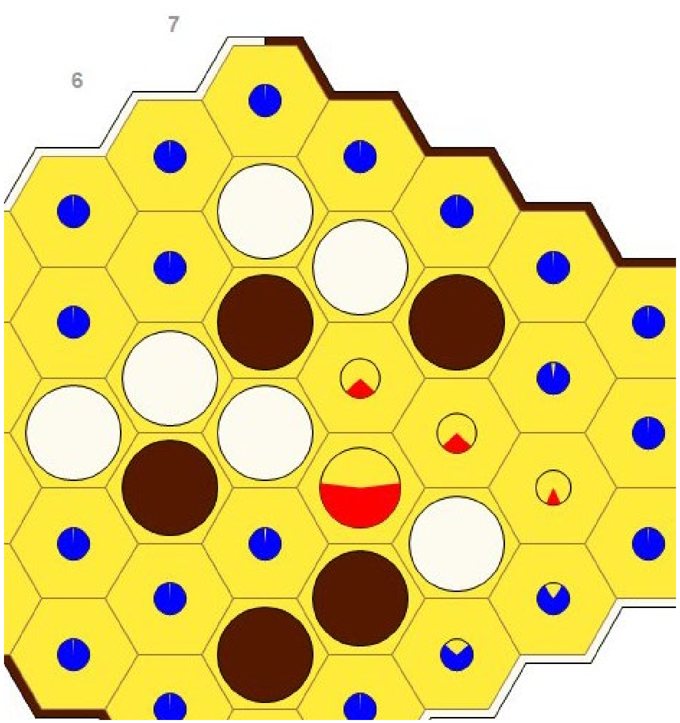
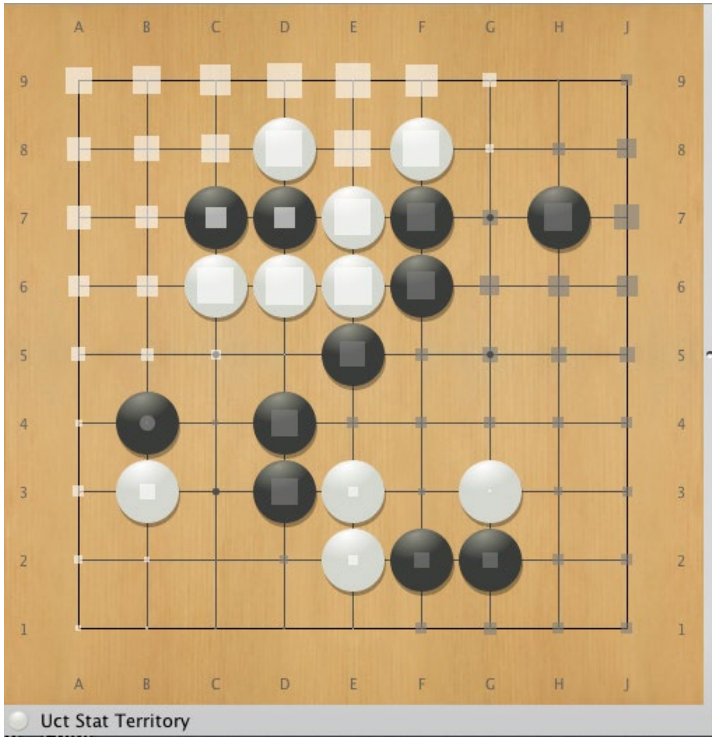


Image from Microsoft Research

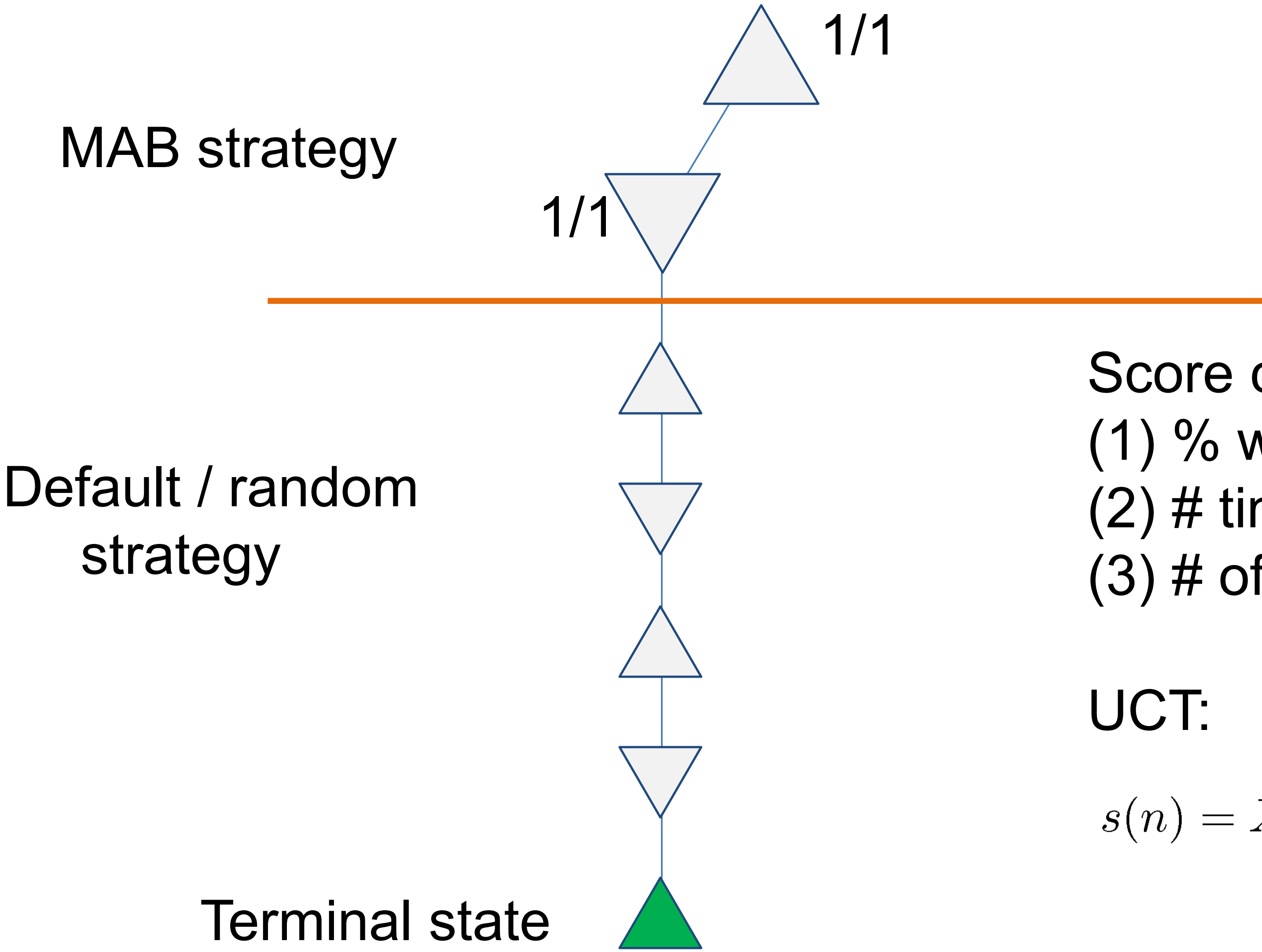
Visualizing MCTS

- At each level of the tree, keep track of
  - Number of times we’ve explored a path
  - Number of times we won
- Follow winning (from max/min perspective) strategies more often, but also explore others





MCTS

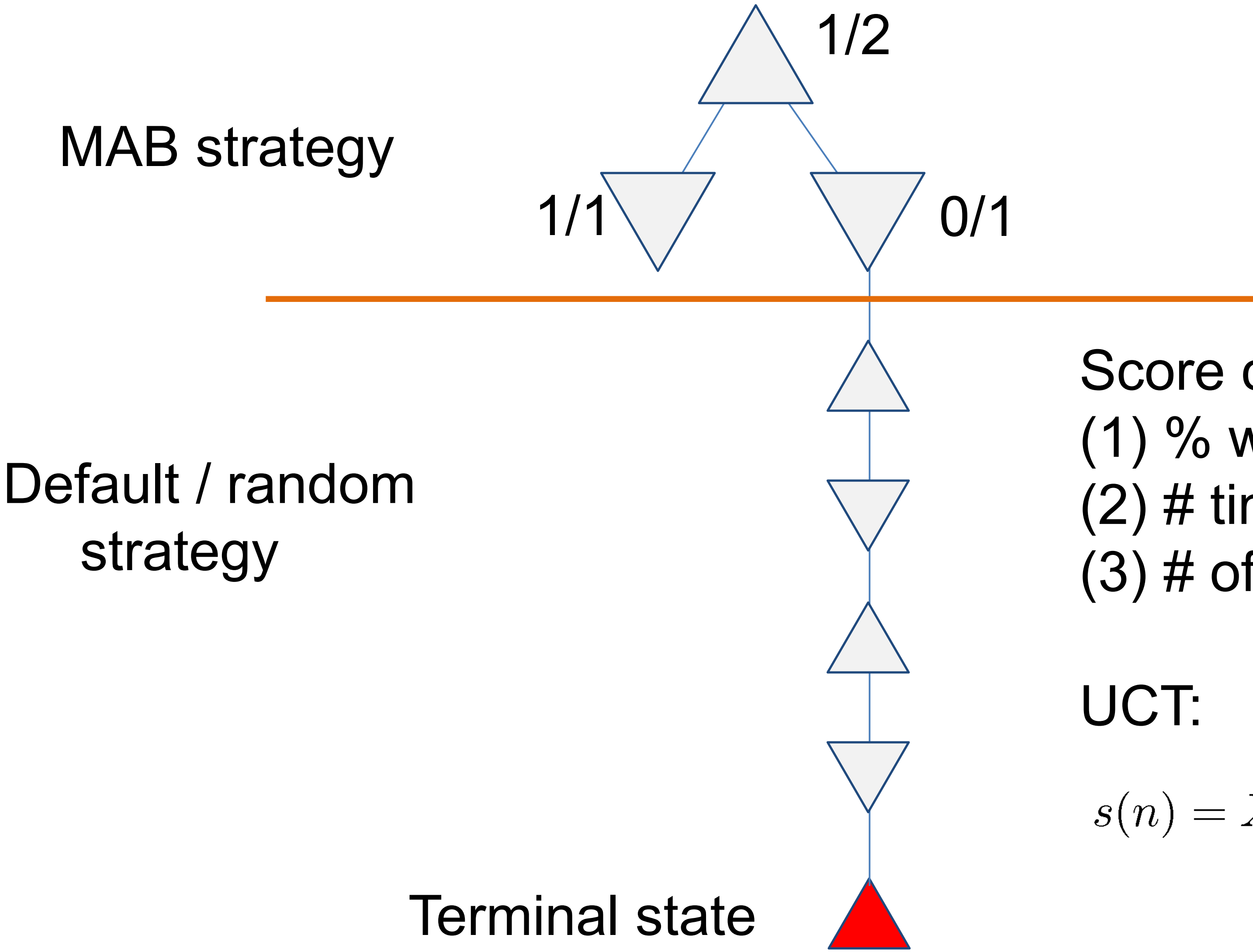


Score consists of
(1) % wins
(2) # times tried
(3) # of steps total

UCT:

$$s(n) = \bar{X}_n \pm \sqrt{\frac{\log n}{t(n)}}$$

MCTS

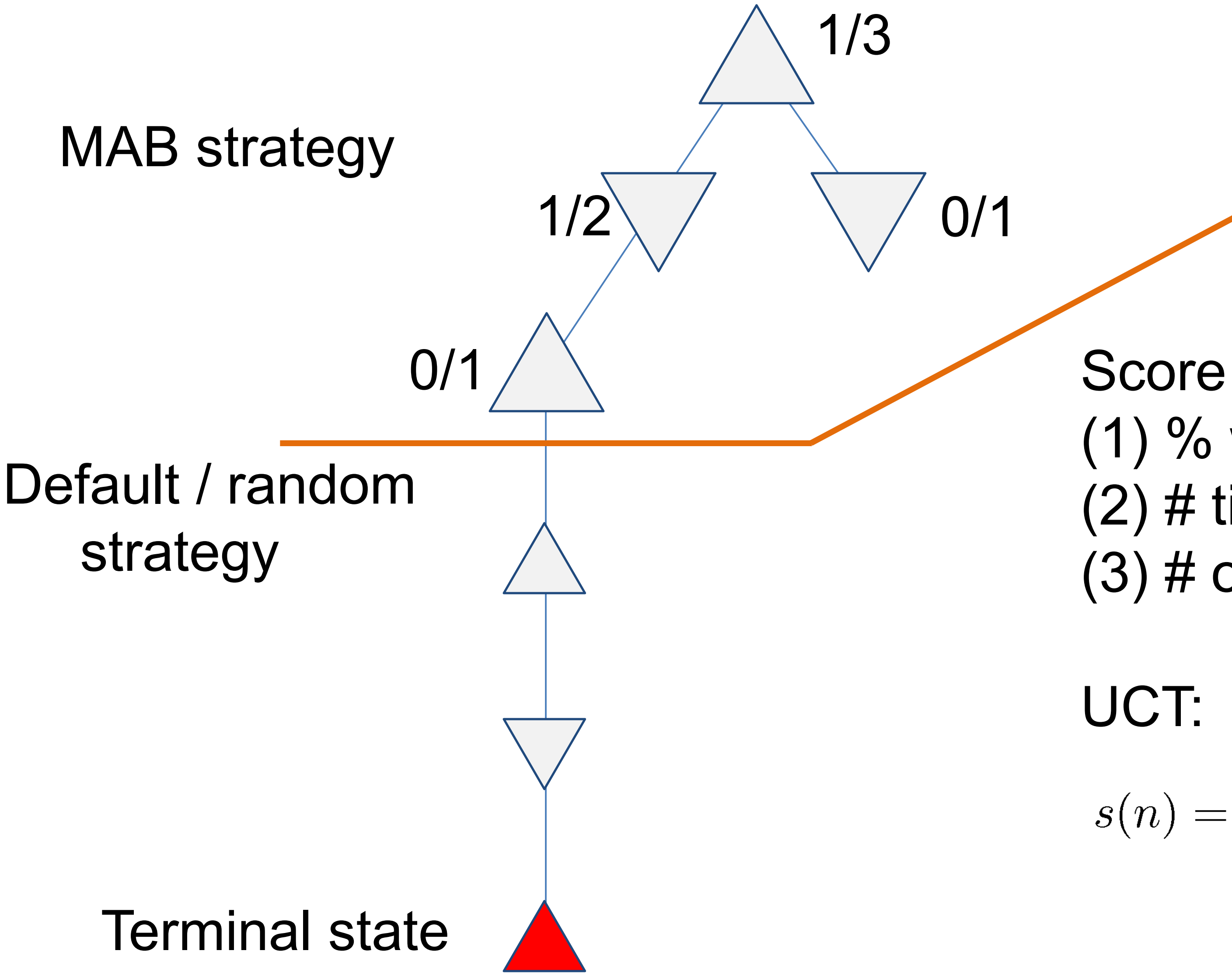


Score consists of
(1) % wins
(2) # times tried
(3) # of steps total

UCT:

$$s(n) = \bar{X}_n \pm \sqrt{\frac{\log n}{t(n)}}$$

MCTS



Score consists of
(1) % wins
(2) # times tried
(3) # of steps total

UCT:

$$s(n) = \bar{X}_n \pm \sqrt{\frac{\log n}{t(n)}}$$

Summary

- Game playing is best modeled as a search problem
- Game trees represent alternate computer/opponent moves
- Evaluation functions estimate the quality of a given board configuration for the Max player.
- Minimax is a procedure which chooses moves by assuming that the opponent will always choose the move which is best for them
- Alpha-Beta is a procedure which can prune large parts of the search tree and allow search to go deeper
- For many well-known games, computer algorithms based on heuristic search match or out-perform human world experts.