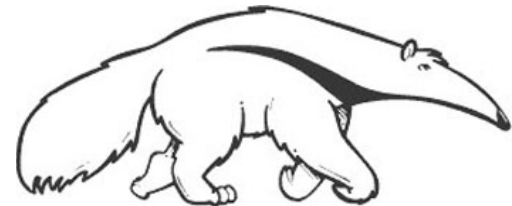


Games & Adversarial Search B: Alpha-Beta Pruning and MCTS

CS171, Winter Quarter, 2019

Introduction to Artificial Intelligence

Prof. Richard Lathrop



Read Beforehand: R&N 5.3; Optional: 5.5+

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

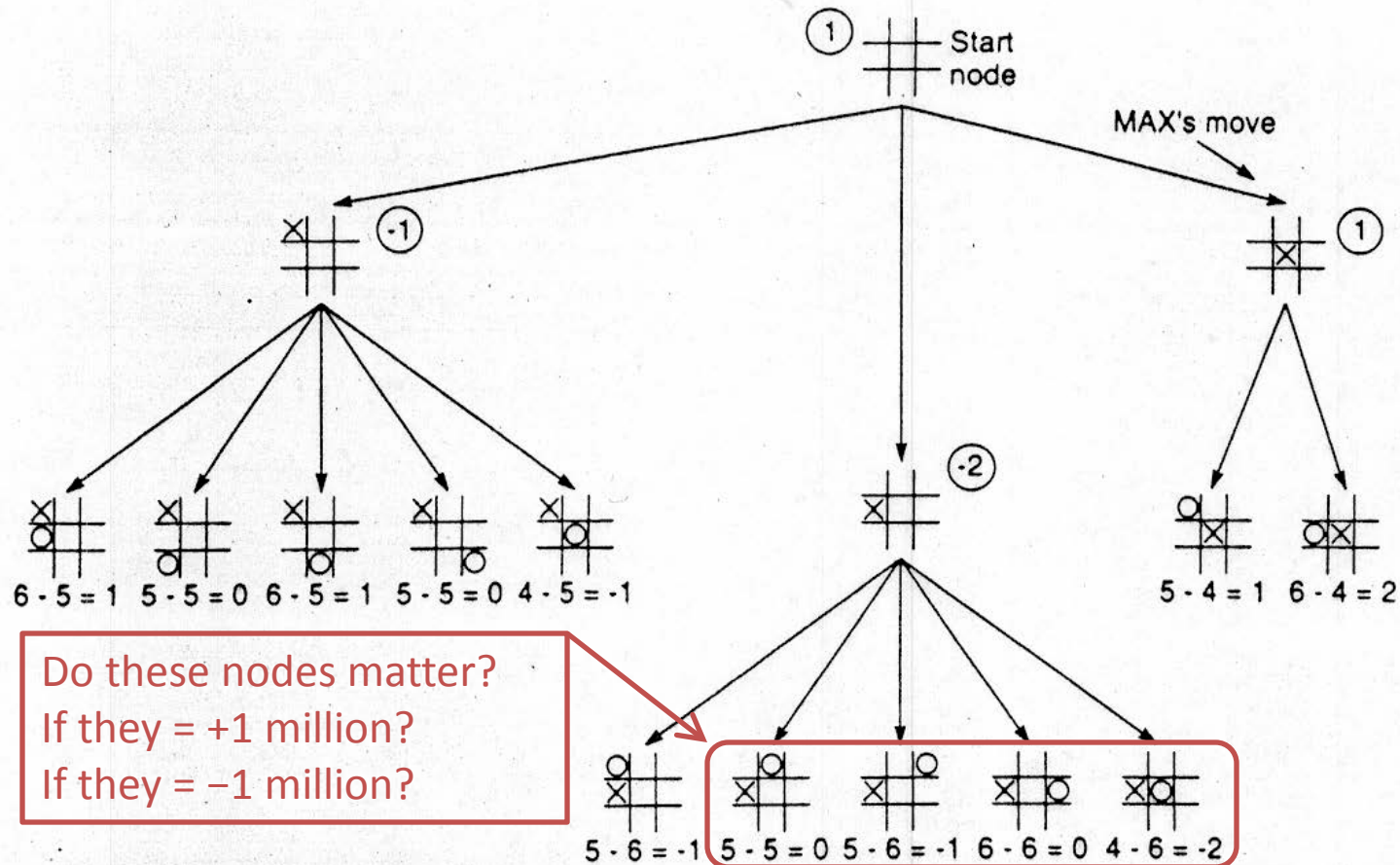
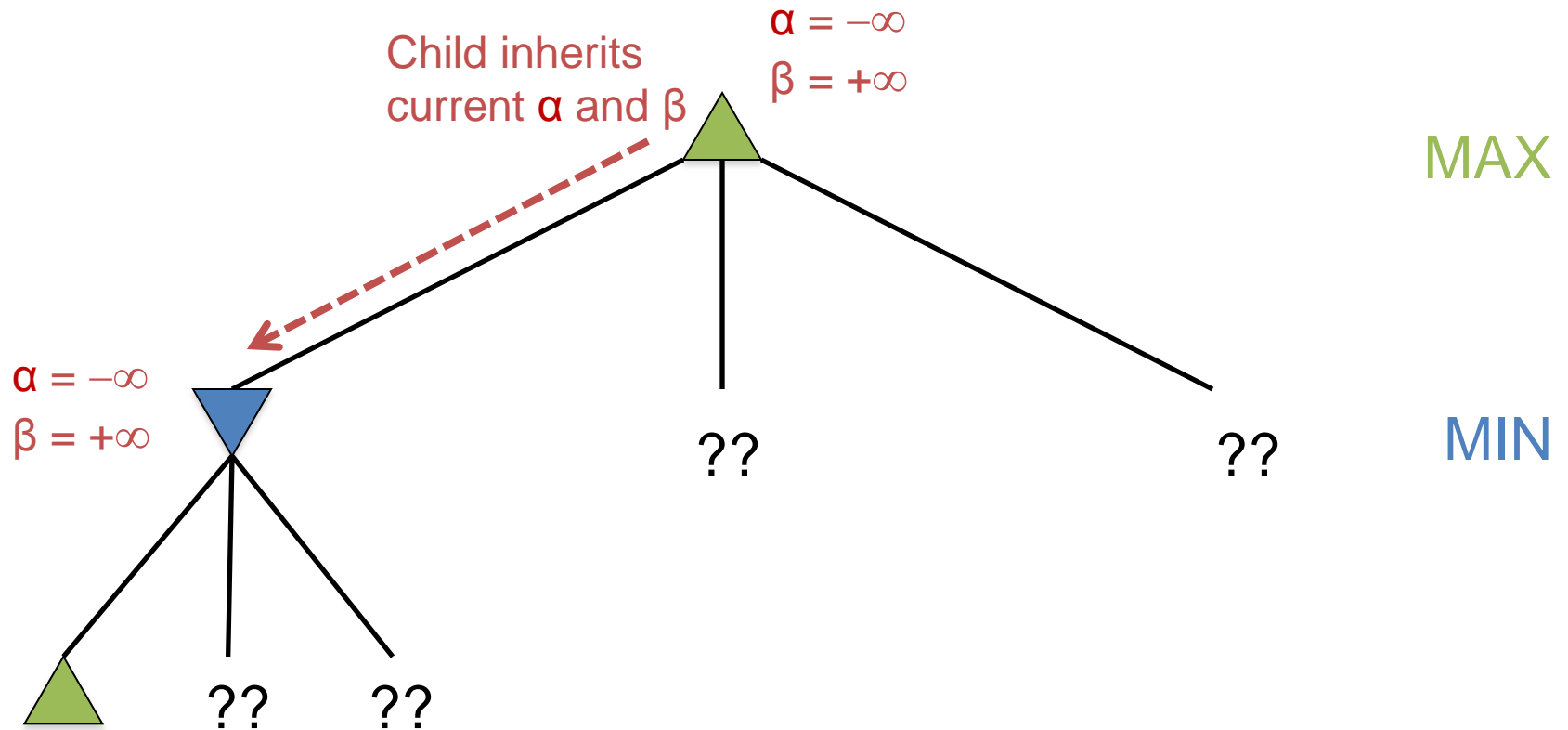


Figure 4.17 Two-ply minimax applied to the opening move of tic-tac-toe.

Alpha-Beta Example

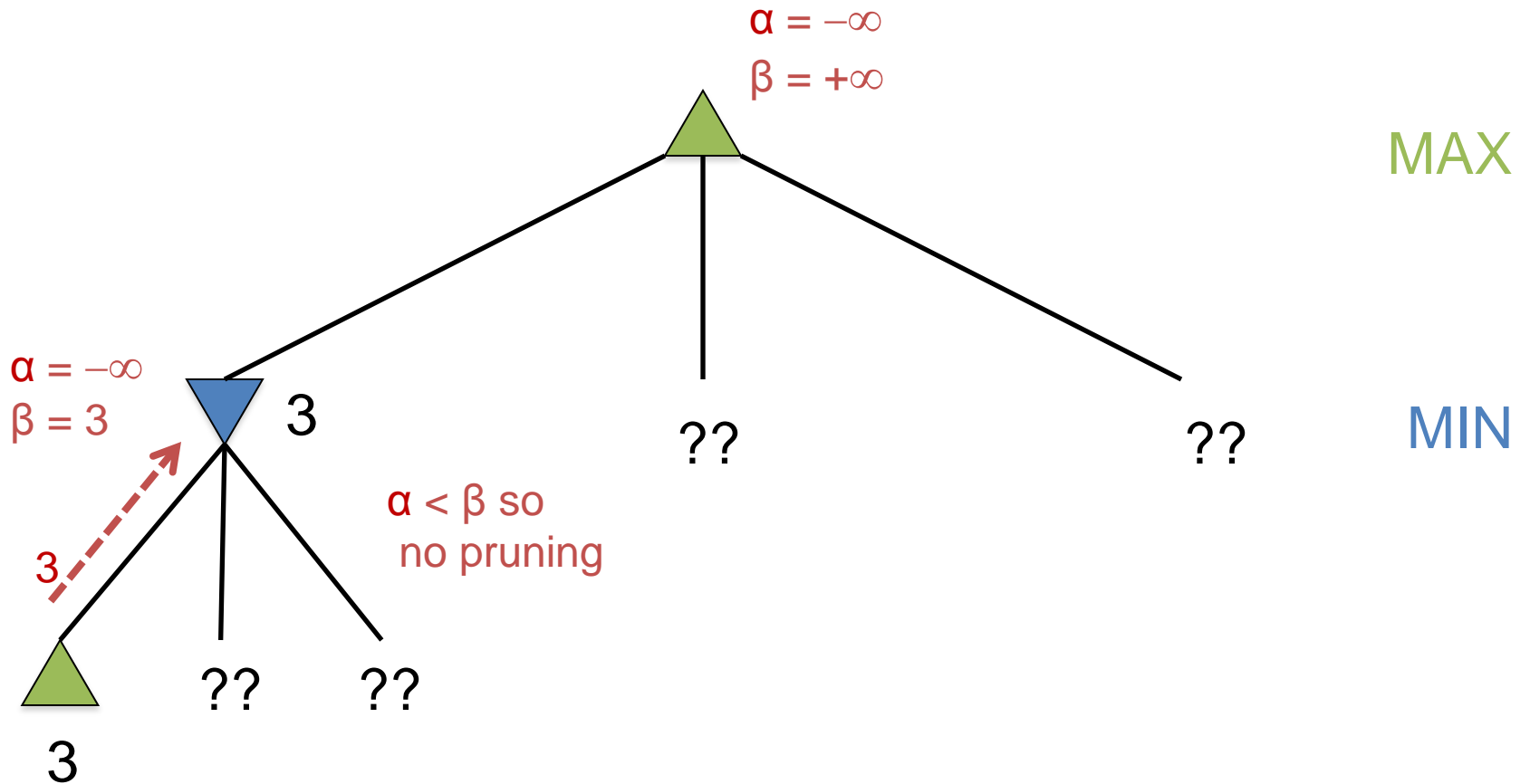
Initially, possibilities are unknown: range ($\alpha = -\infty$, $\beta = +\infty$)

Do a depth-first search to the first leaf.



Alpha-Beta Example

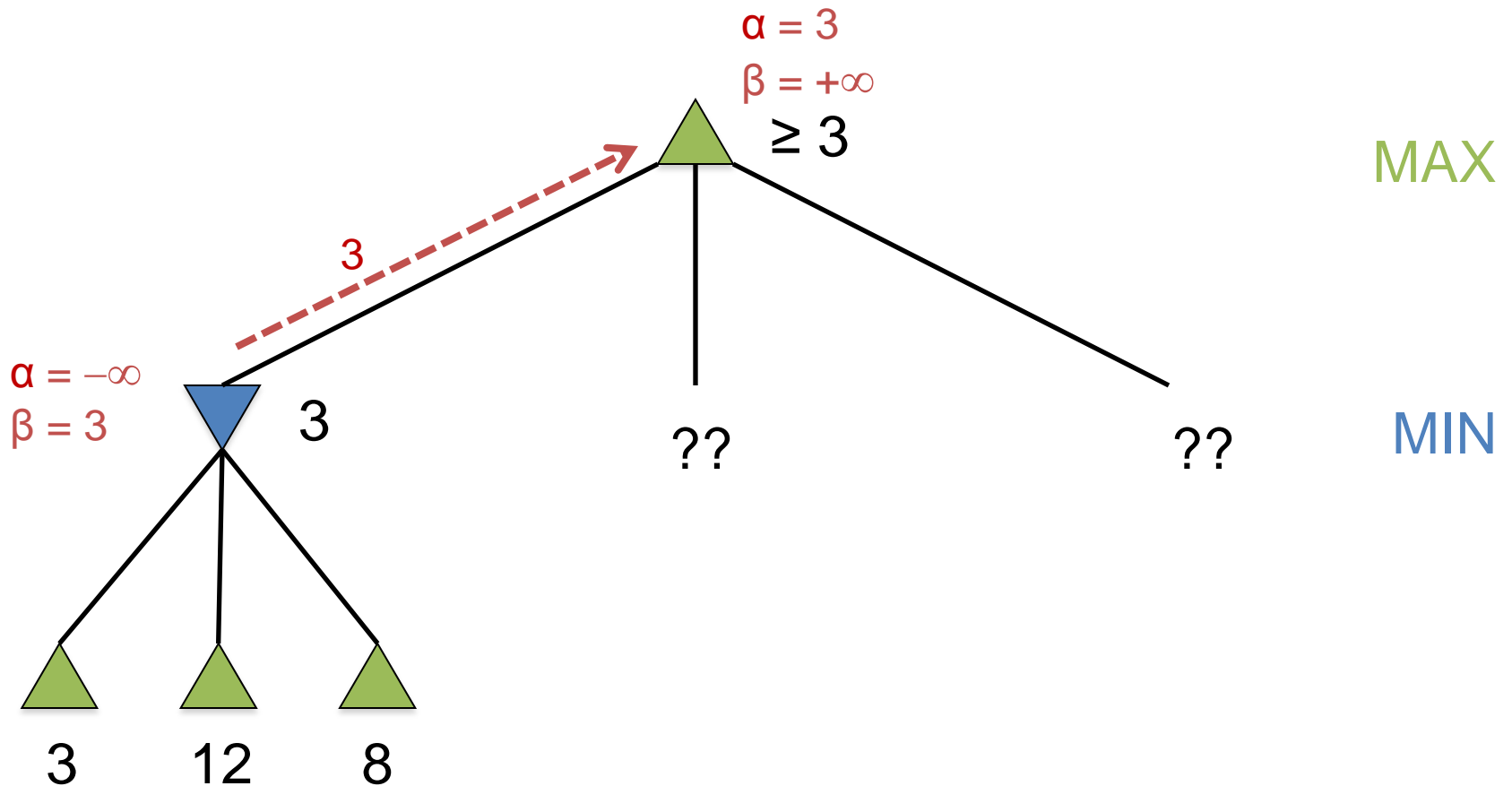
See the first leaf, after MIN's move: MIN updates β



Alpha-Beta Example

See remaining leaves; value is known

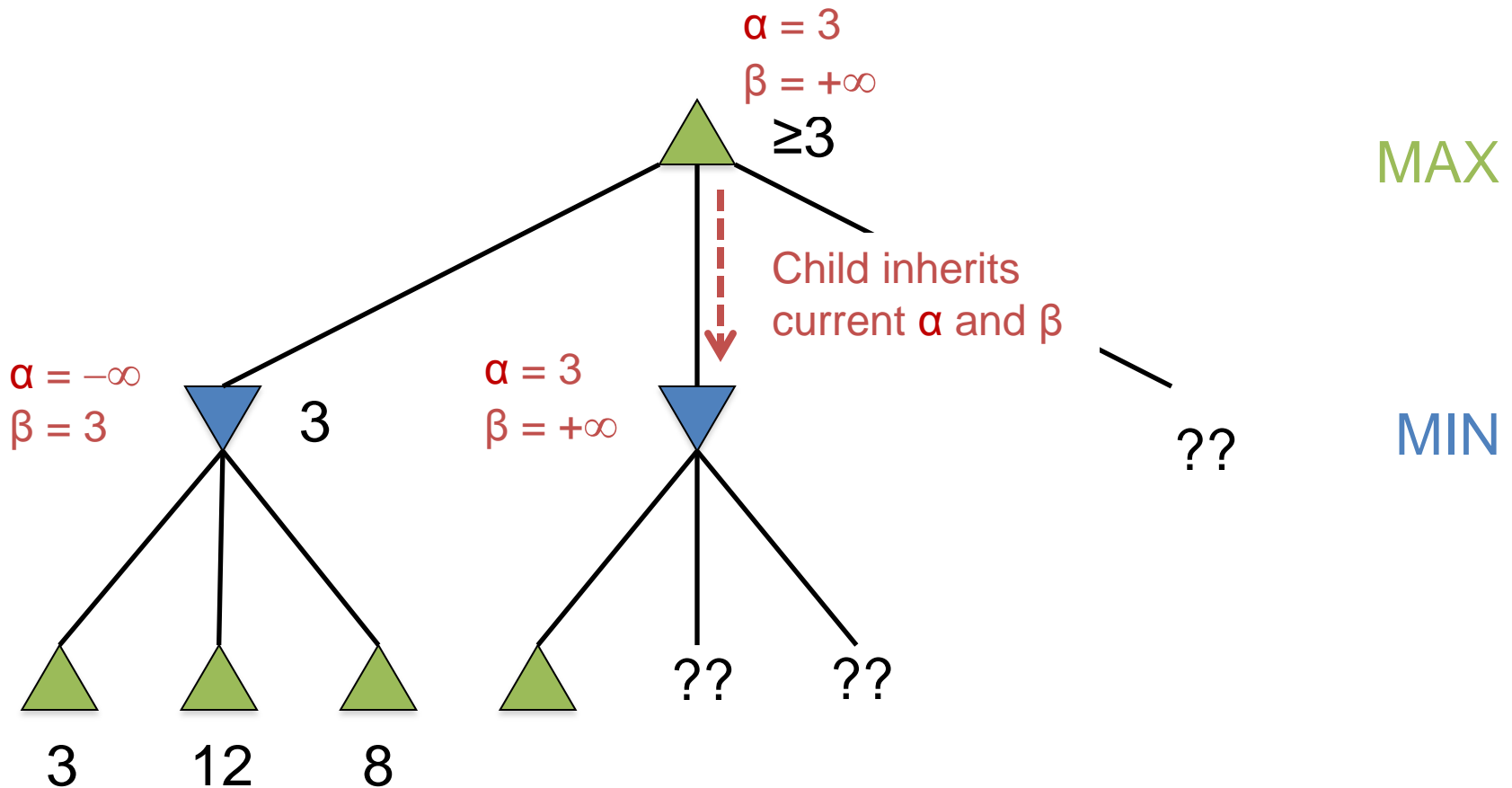
Pass outcome to caller; MAX updates α



Alpha-Beta Example

Continue depth-first search to next leaf.

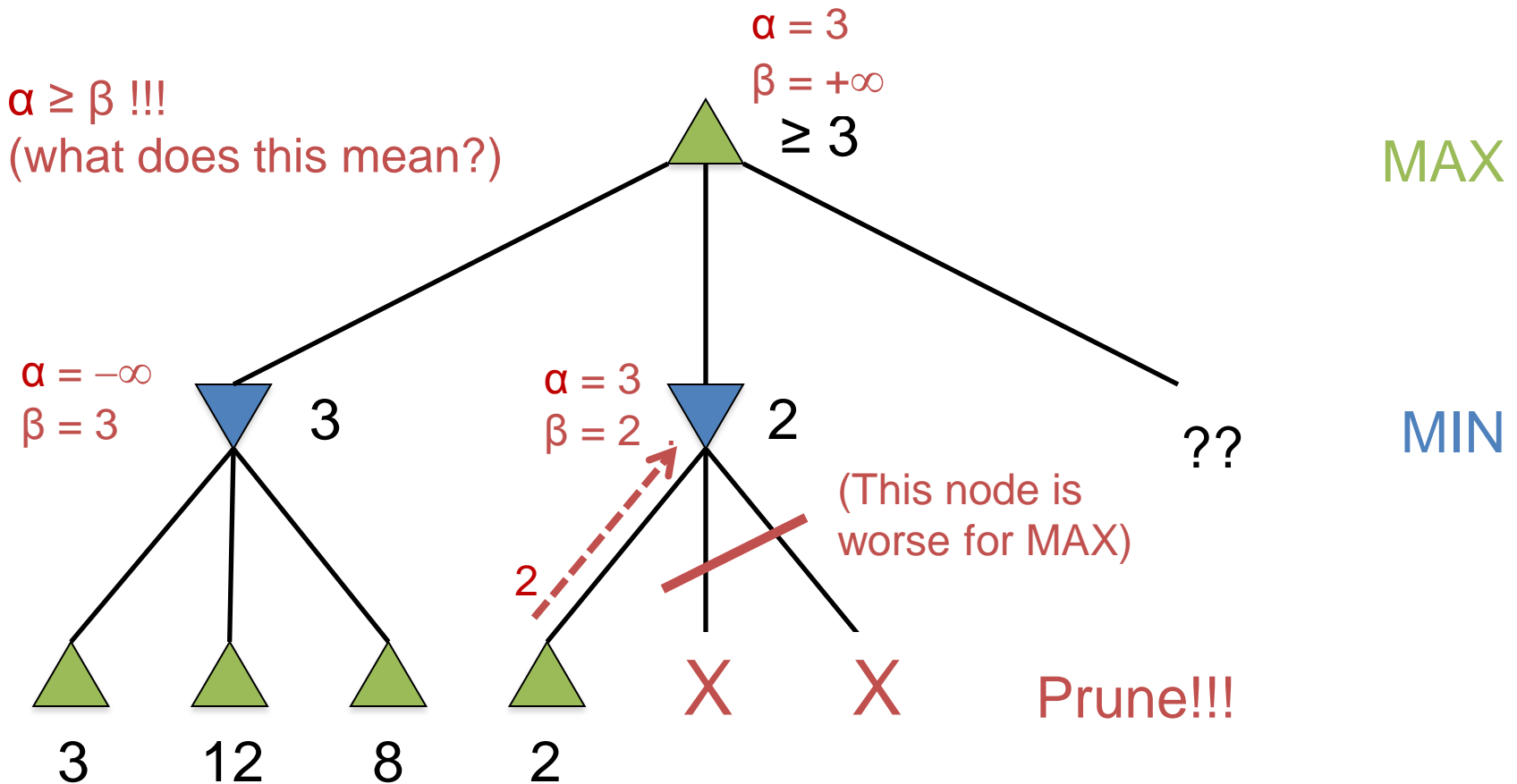
Pass α , β to descendants



Alpha-Beta Example

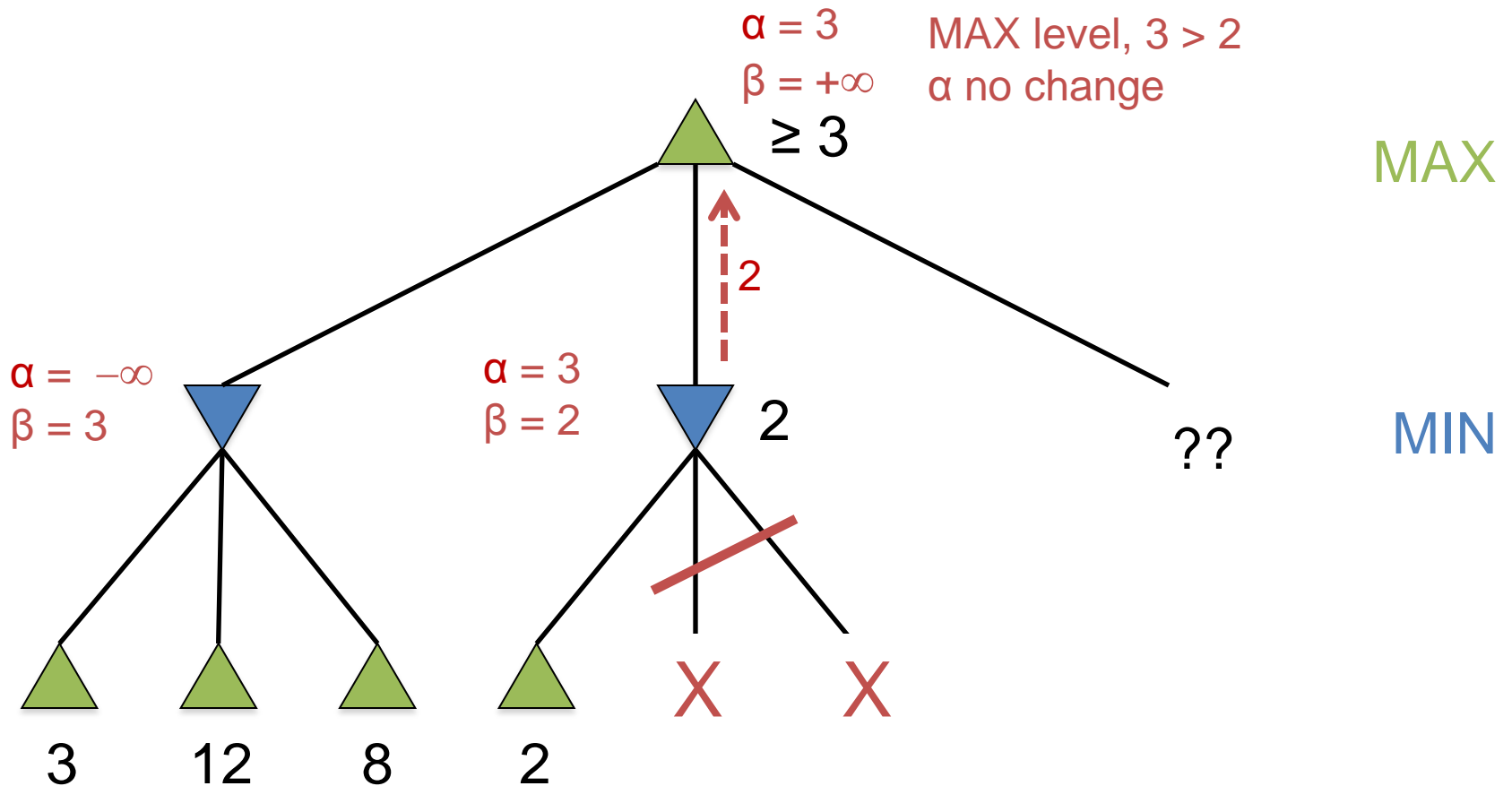
Observe leaf value; MIN's level; MIN updates β

Prune – play will never reach the other nodes!



Alpha-Beta Example

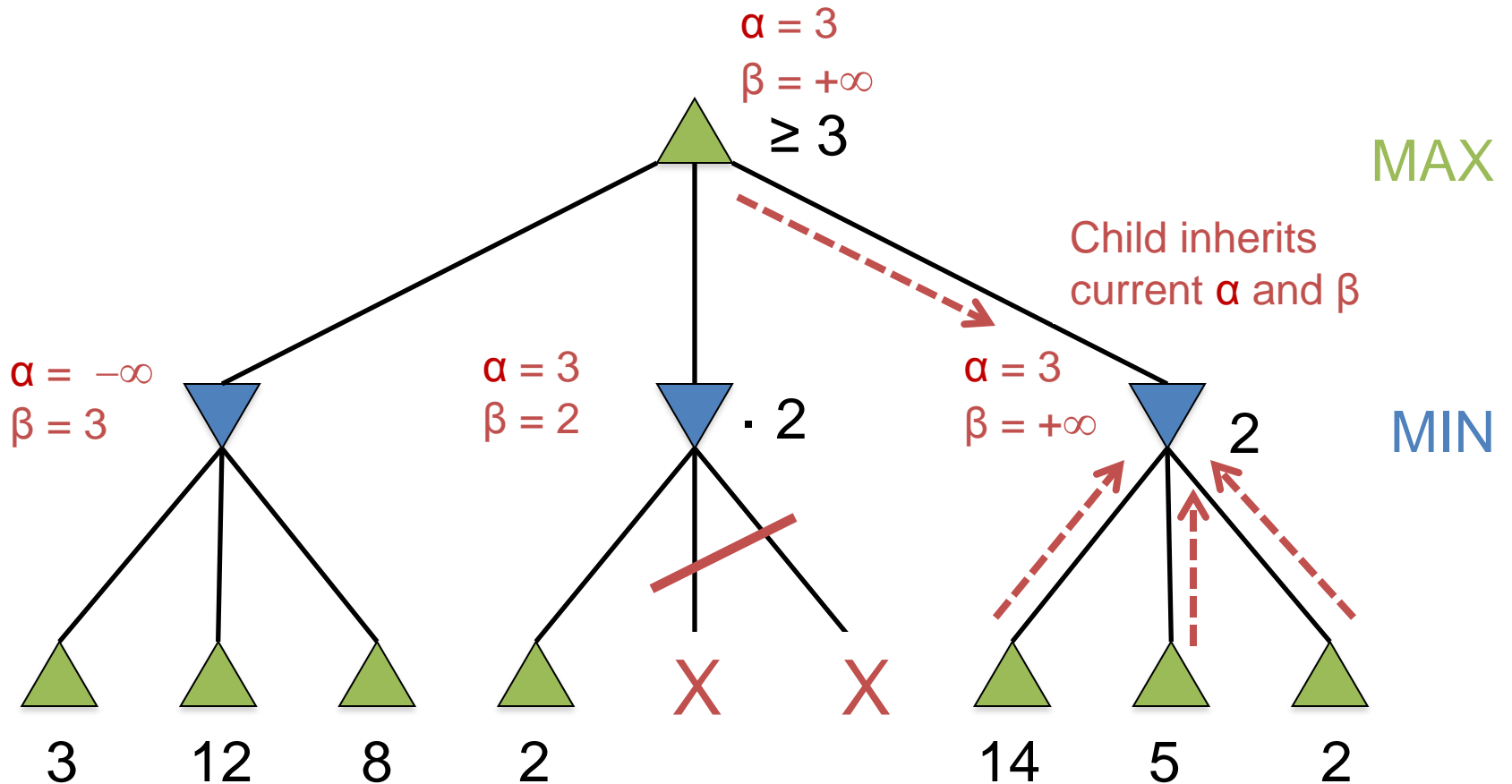
Pass outcome to caller & update caller:



Alpha-Beta Example

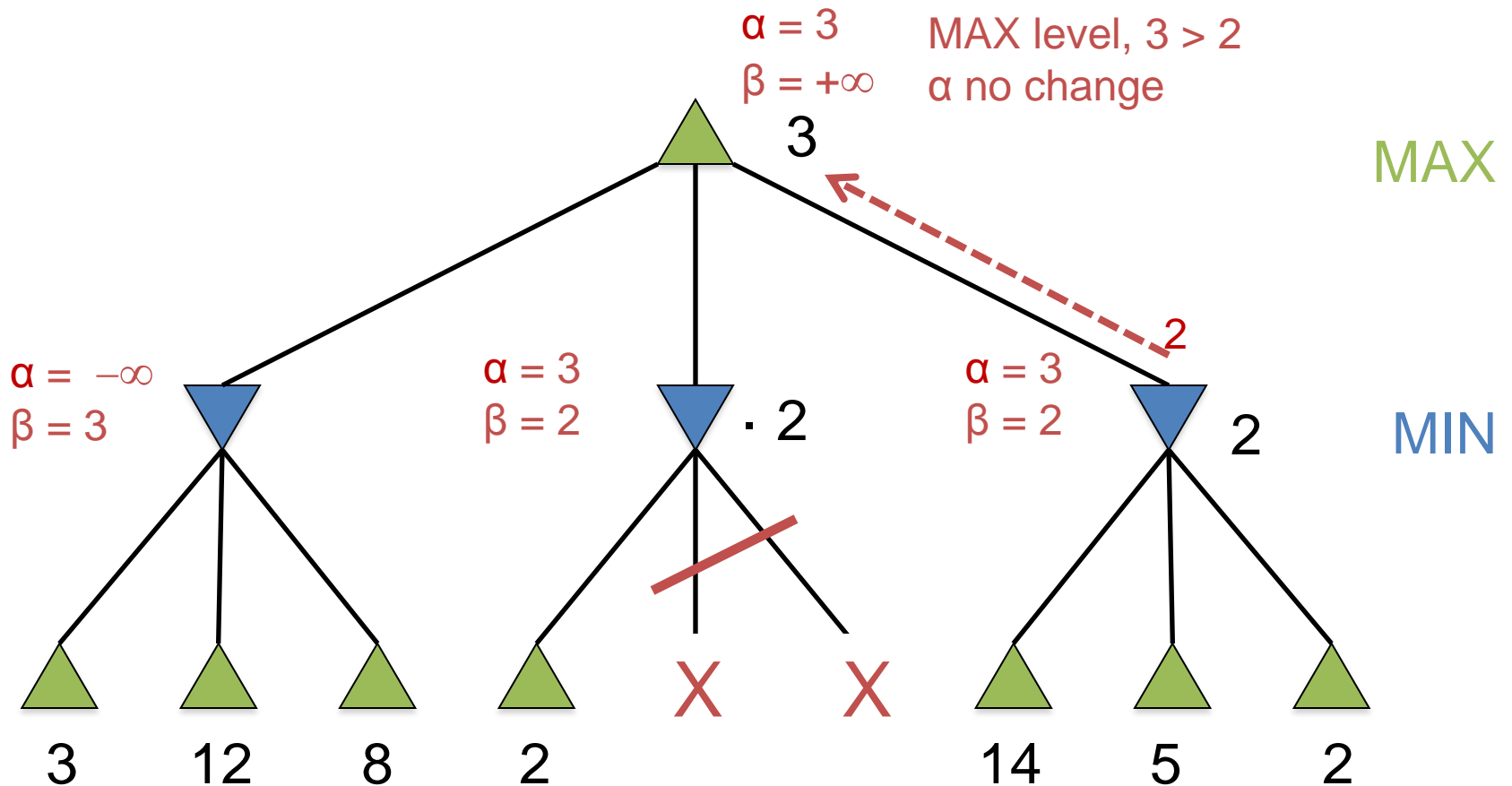
Continue depth-first exploration...

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



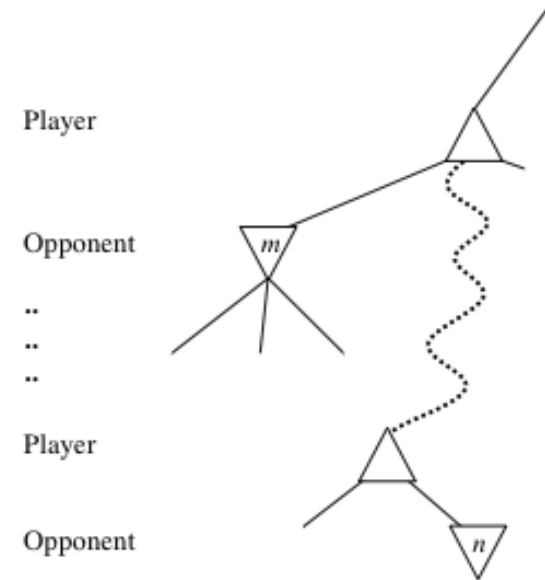
Alpha-Beta Example

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 (expands on Fig. 5.7)

Alpha-Beta-Search(state)

```
alpha = -infty, beta = +infty, act = None
for each a in Actions(state) do
    val = Min-Value( Result(state, a), alpha, beta )
    if ( val > alpha ) then alpha = val, act = a
return act
```

Simple stub to call recursion functions
Initialize alpha, beta; no move found
Score each action; update alpha & best action

MaxValue(state, al, be)

```
if (Cutoff(state)) then return Eval(state)
val = -infty
for each a in Actions(state) do
    val = max( val, MinValue( Result(state, a), al, be ) )
    if ( val ≥ be ) then return val
    al = max( al, val )
return val
```

If Cutoff reached, return Eval heuristic
Otherwise, find our best child:
If our options are too good, our min
ancestor will never let us come this way
Otherwise return the best we can find

MinValue(state, al, be)

```
if (Cutoff(state)) then return Eval(state)
val = +infty
for each a in Actions(state) do
    val = min( val, MaxValue( Result(state, a), al, be ) )
    if ( val ≤ al ) then return val
    be = min( be, val )
return be
```

If Cutoff reached, return Eval heuristic
Otherwise, find the worst child:
If our options are too bad, our max
ancestor will never let us come this way
Otherwise return the worst we can find

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 $(\sqrt{b})^d = b^{(d/2)}$ (i.e., we have effectively gone from b to square root of b)
 - In chess go from $b \sim 35$ to $b \sim 6$
 - permitting 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 \Rightarrow better, faster search
 - Only a heuristic: node values will change with the deeper search
 - Usually works well in practice

Comments on alpha-beta pruning

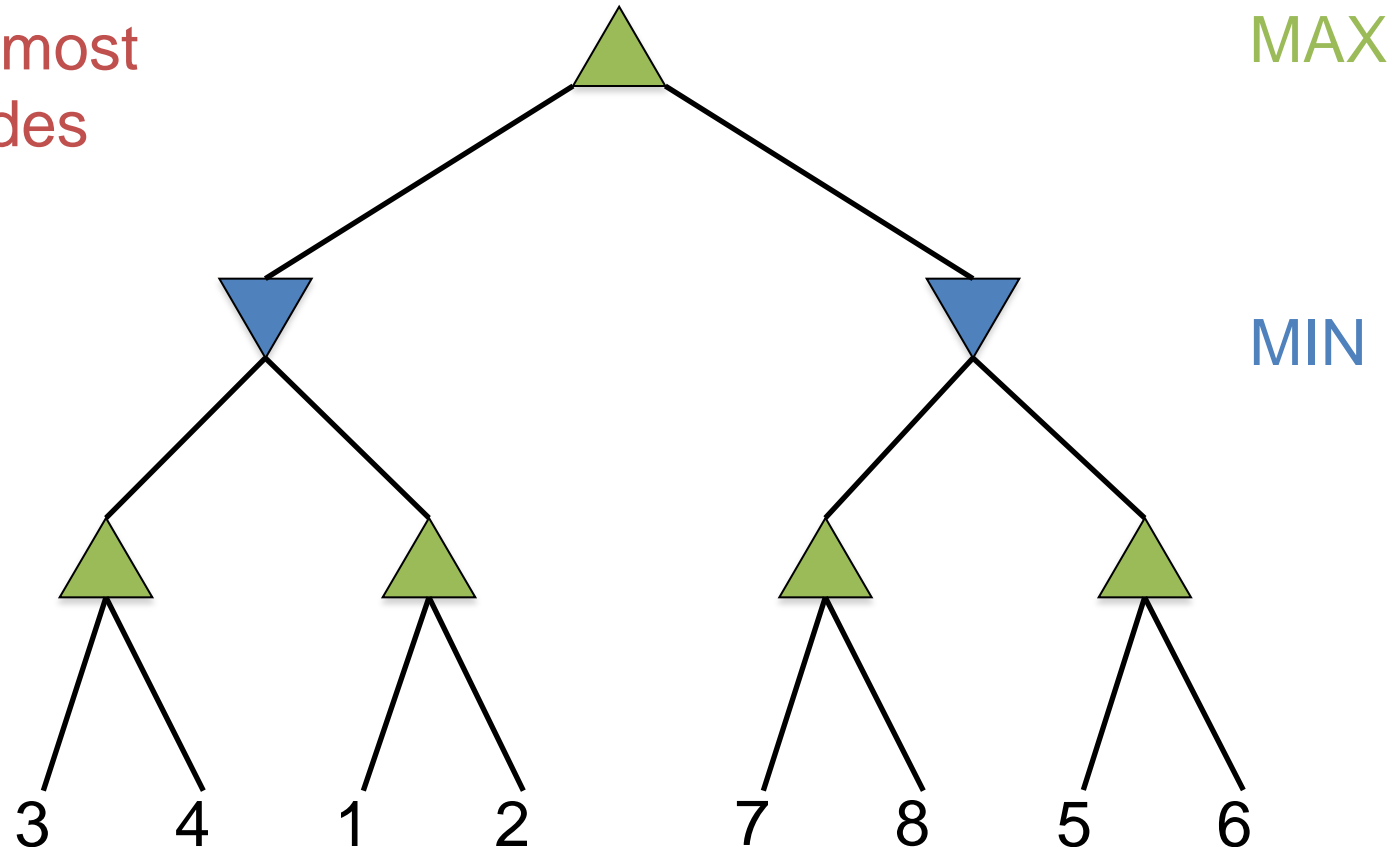
- Pruning does not affect final results
- Entire subtrees can be pruned
- Good move ordering improves pruning
 - Order nodes so player's best moves are checked first
- Repeated states are still possible
 - Store them in memory = transposition table

Iterative deepening reordering

Which leaves can be pruned?

None!

because the most favorable nodes are explored last...

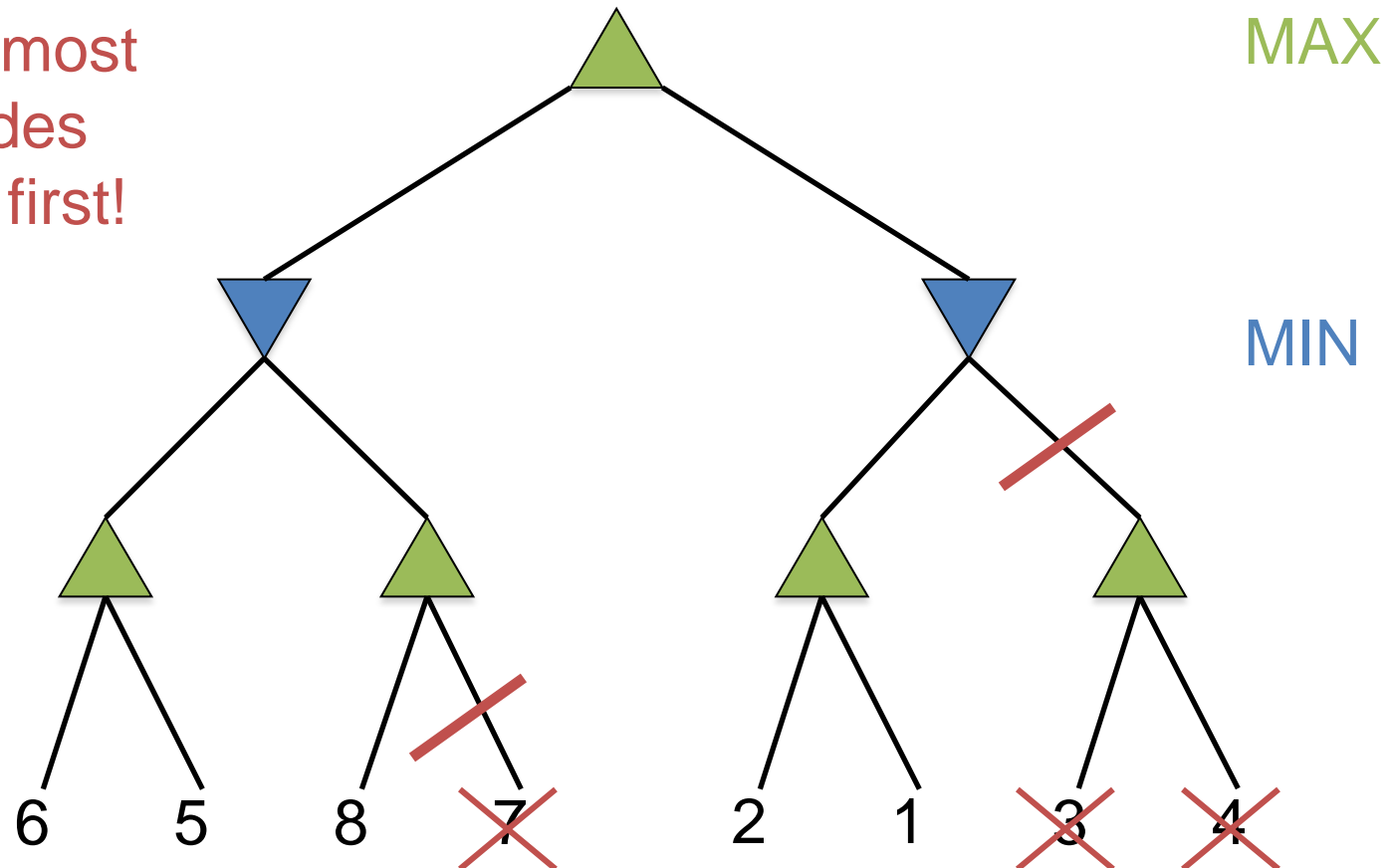


Iterative deepening reordering

Different exploration order: now which leaves can be pruned?

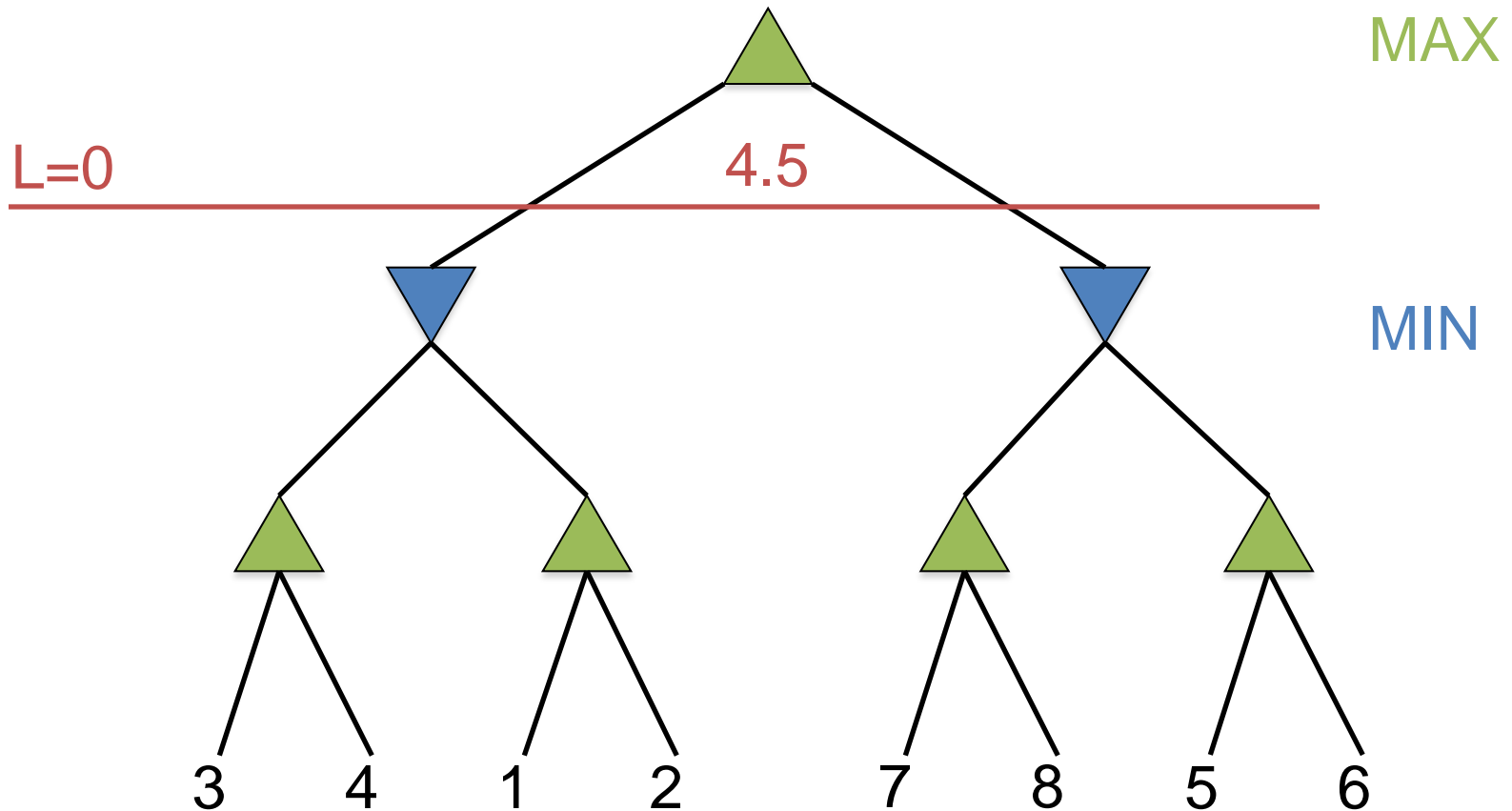
Lots!

because the most favorable nodes are explored first!



Iterative deepening reordering

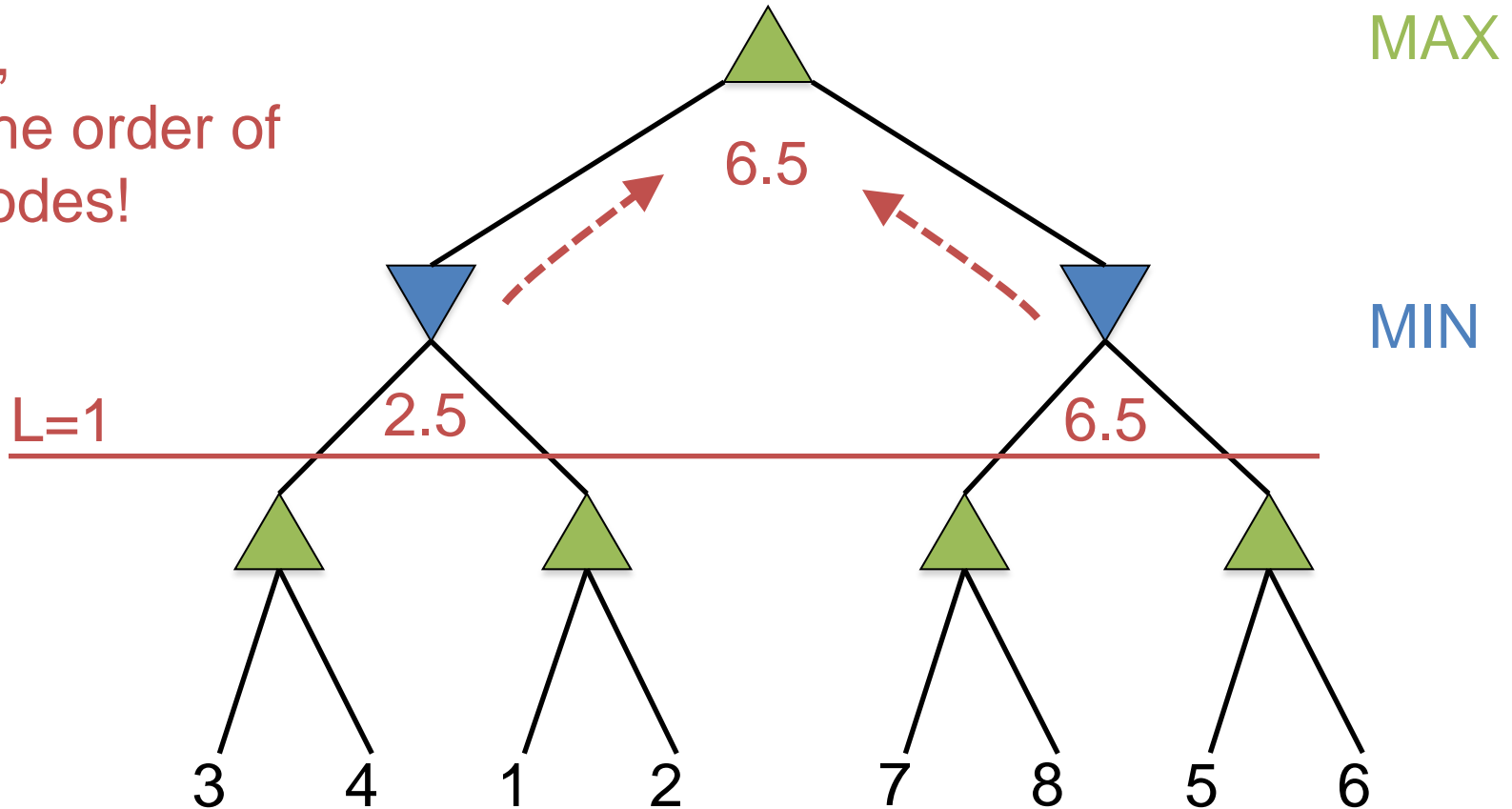
Order with no pruning; use iterative deepening approach.
Assume node score is the average of leaf values below.



Iterative deepening reordering

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

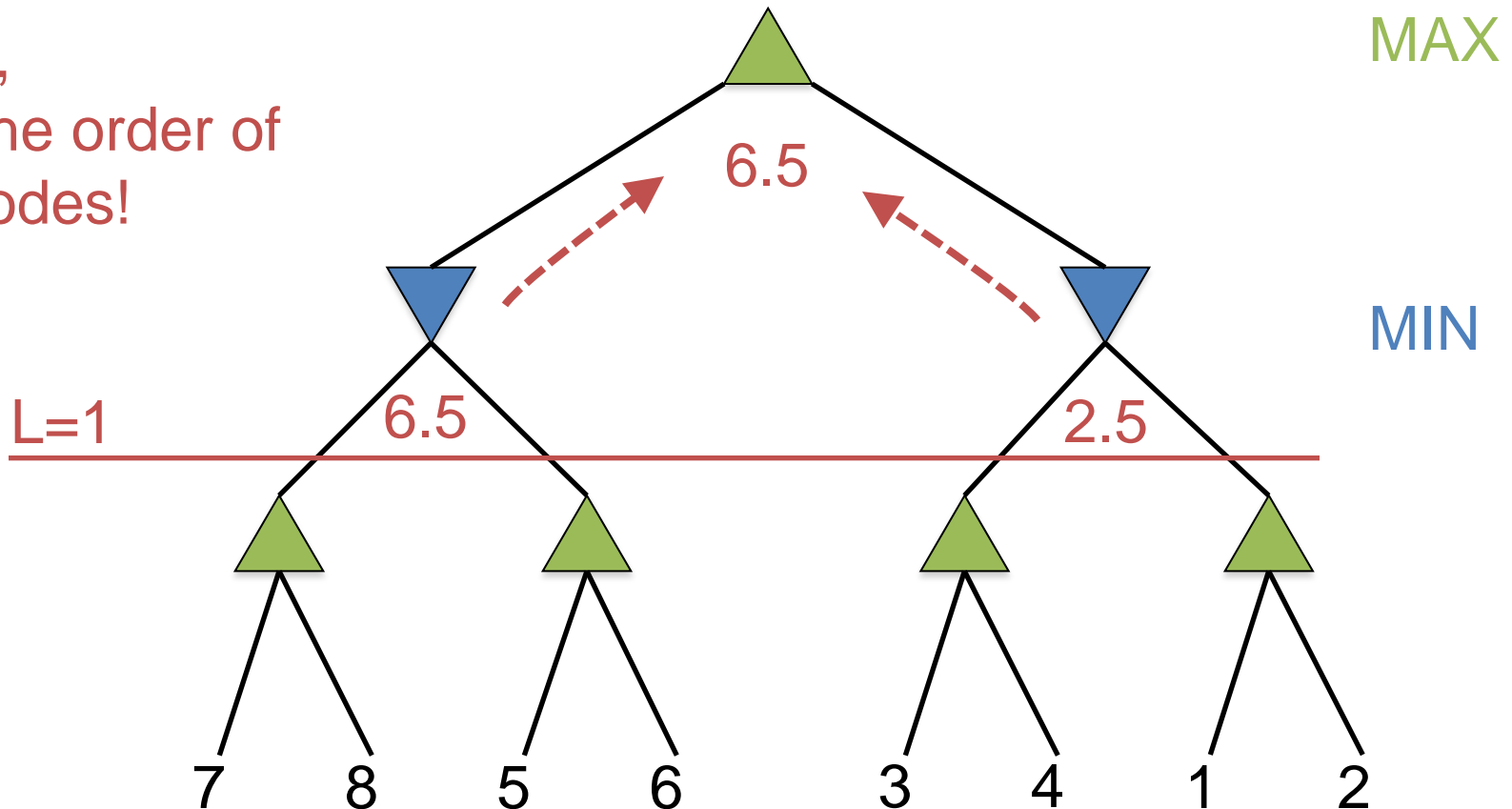
For $L=2$,
switch the order of
these nodes!



Iterative deepening reordering

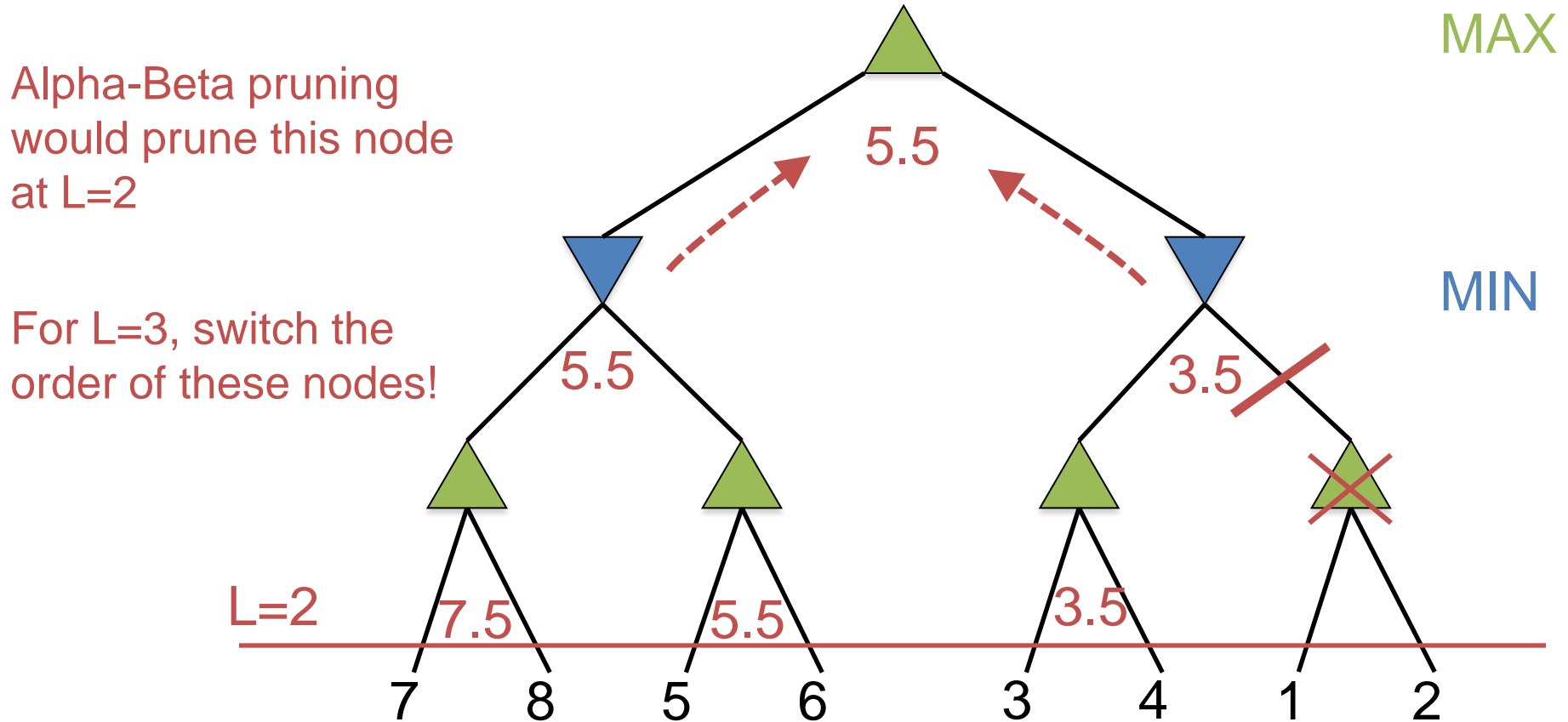
Order with no pruning; use iterative deepening approach.
Assume node score is the average of leaf values below.

For $L=2$,
switch the order of
these nodes!



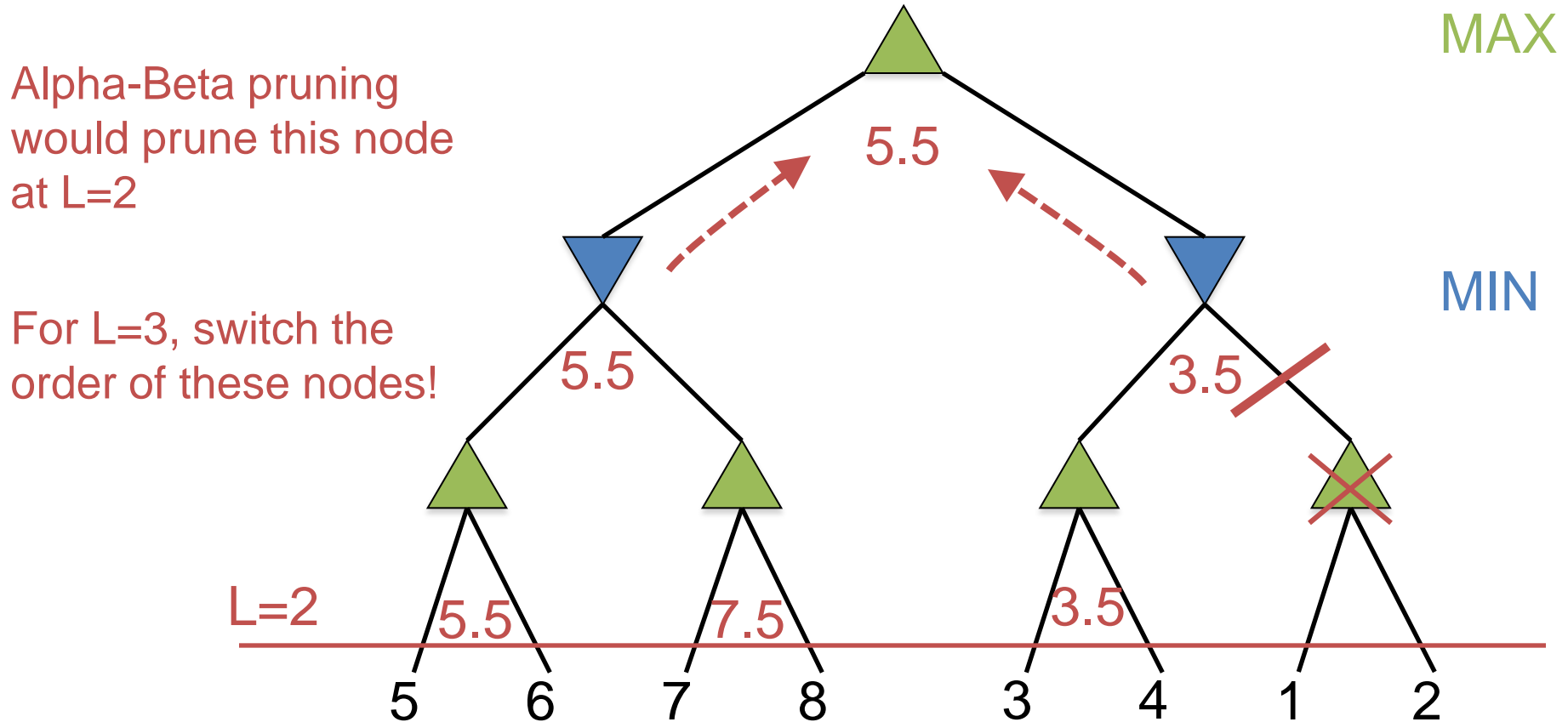
Iterative deepening reordering

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



Iterative deepening reordering

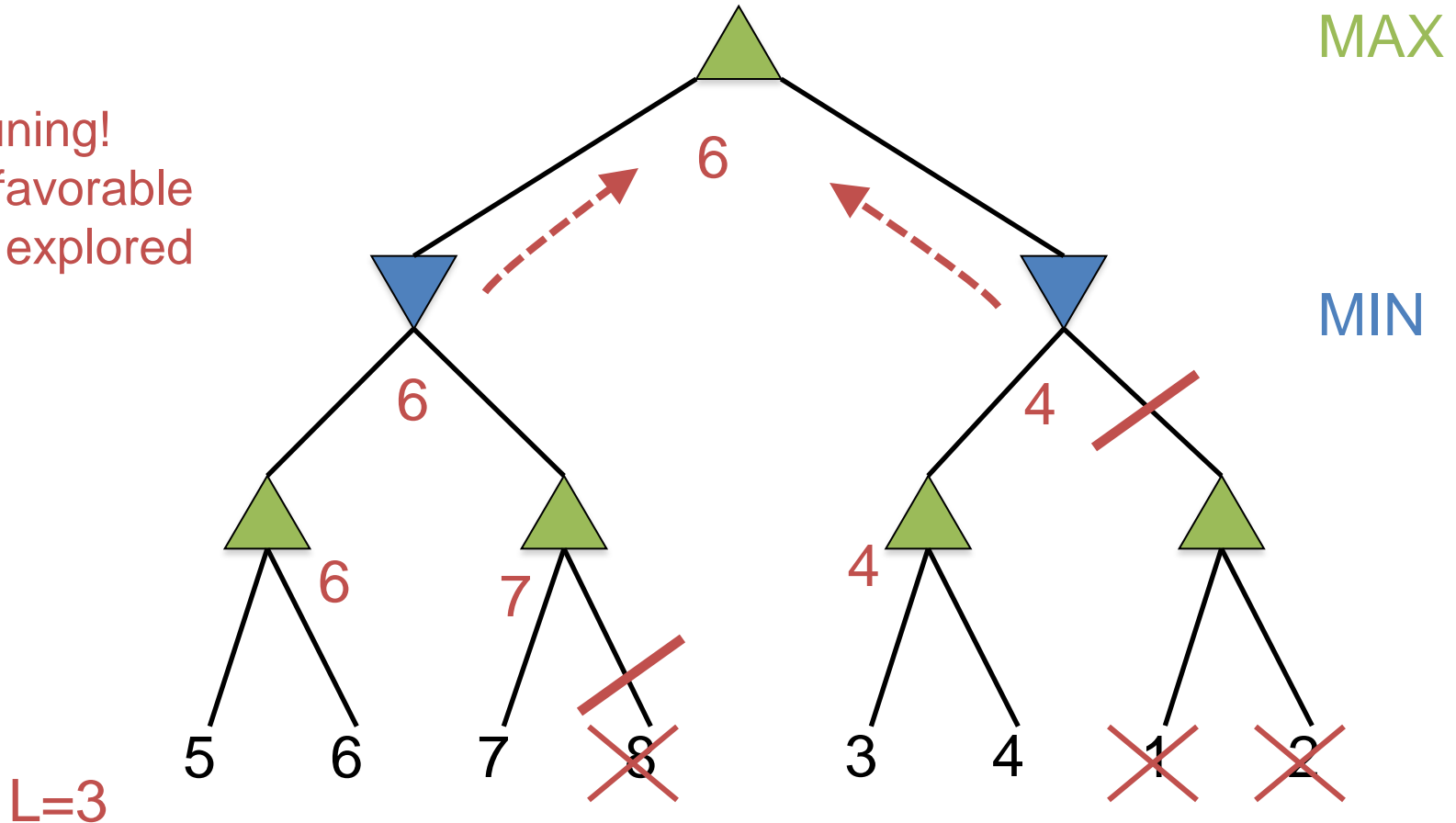
Order with no pruning; use iterative deepening approach.
Assume node score is the average of leaf values below.



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.

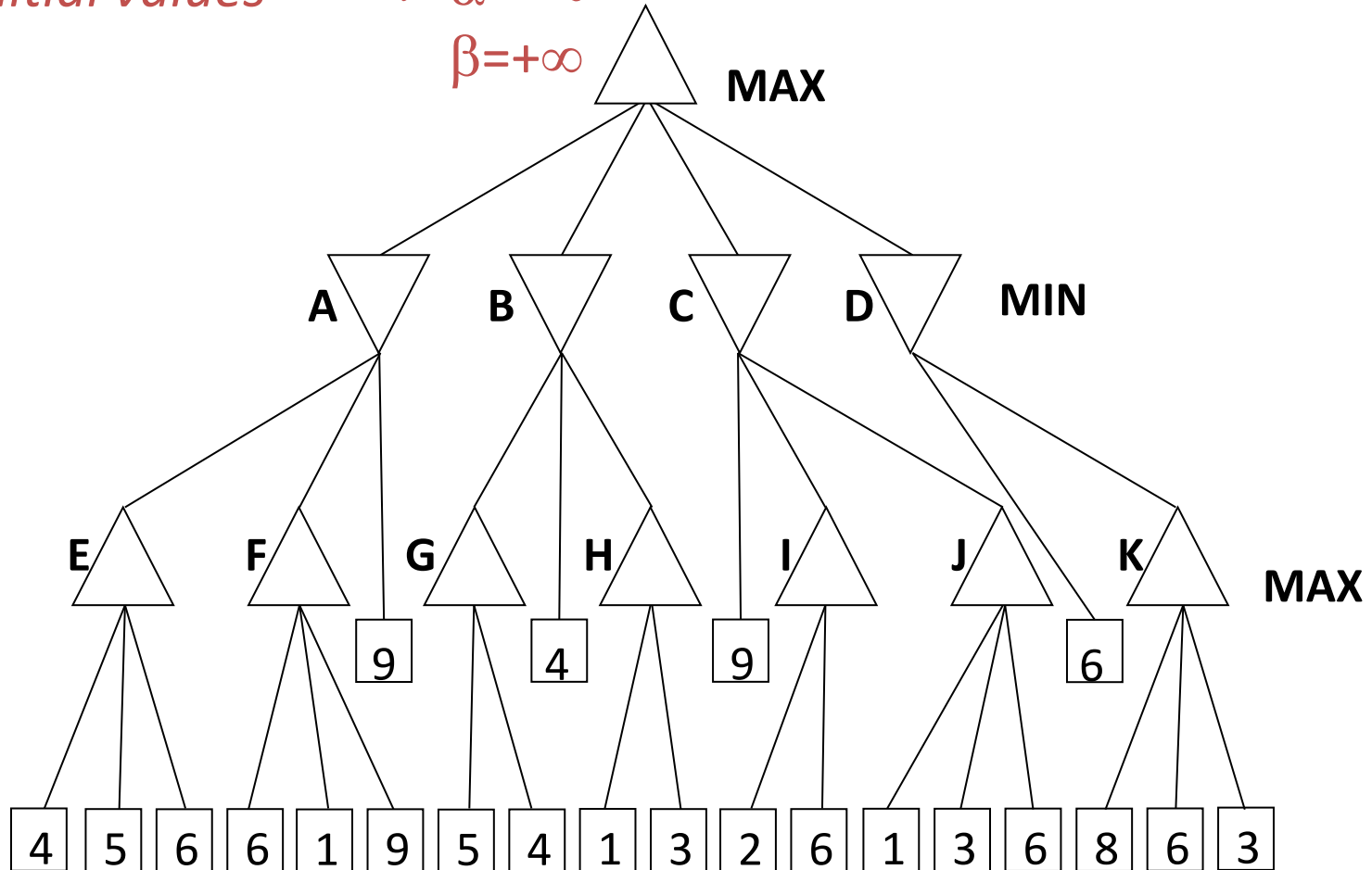


Longer Alpha-Beta Example

Branch nodes are labeled A..K for easy discussion

α, β , initial values $\longrightarrow \alpha = -\infty$

$\beta = +\infty$



Longer Alpha-Beta Example

Note that cut-off occurs at different depths...

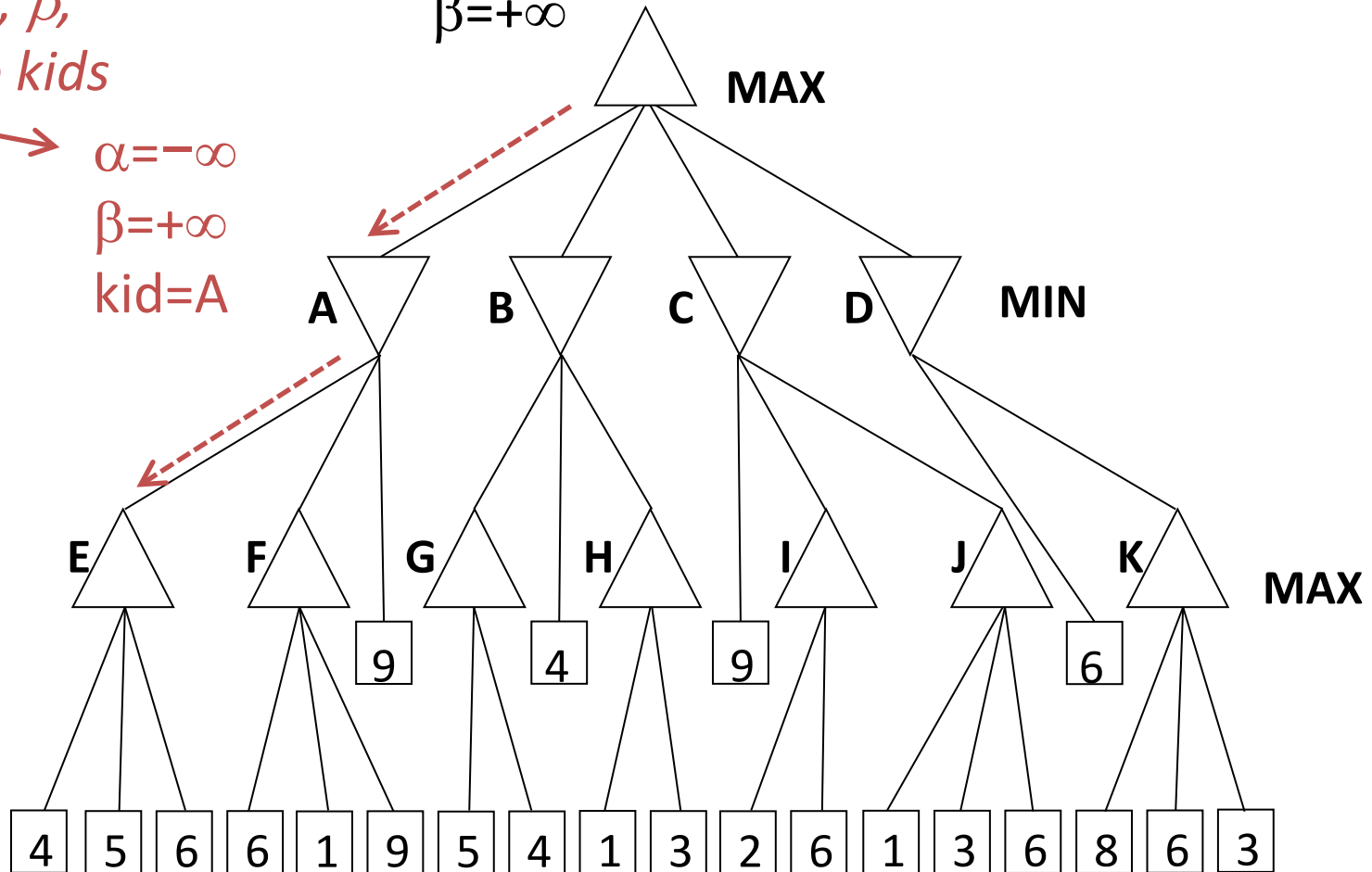
$\alpha = -\infty$

$\beta = +\infty$

*current α, β ,
passed to kids*

$\alpha = -\infty$
 $\beta = +\infty$
kid=A

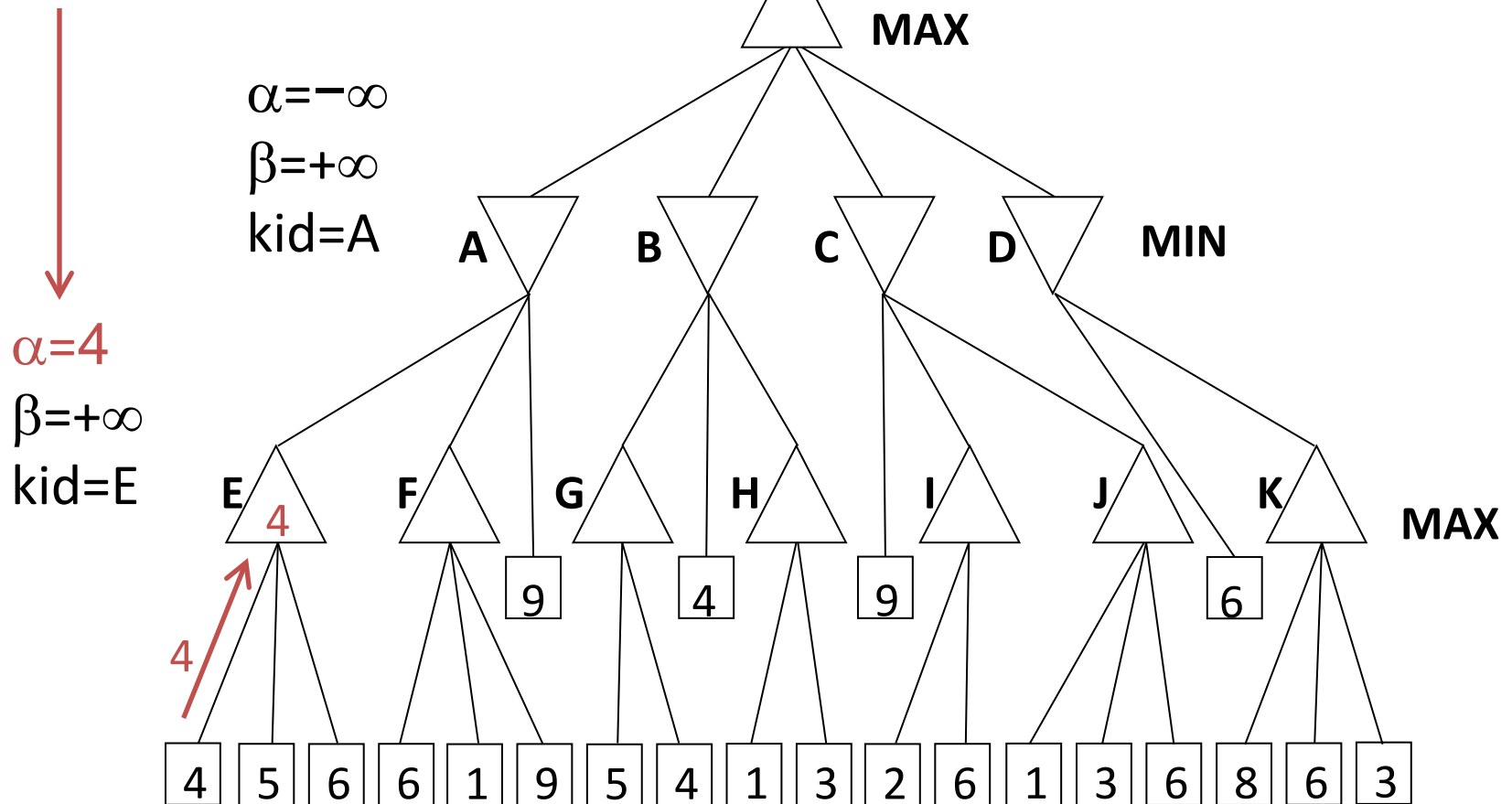
$\alpha = -\infty$
 $\beta = +\infty$
kid=E



Longer Alpha-Beta Example

*see first leaf,
MAX updates α*

$\alpha = -\infty$
 $\beta = +\infty$



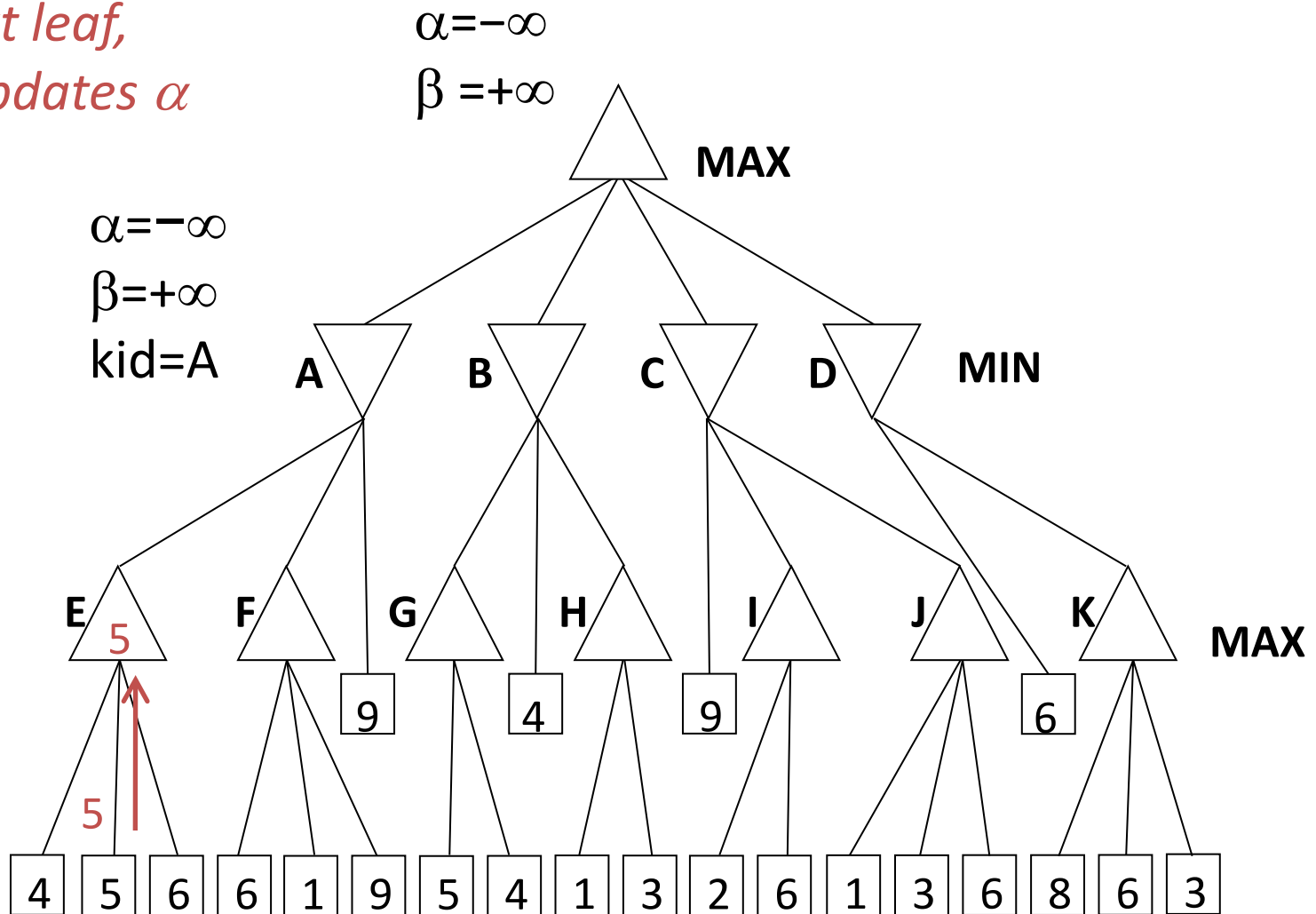
We also are running MiniMax search and recording node values within the triangles, without explicit comment.

Longer Alpha-Beta Example

*see next leaf,
MAX updates α*



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

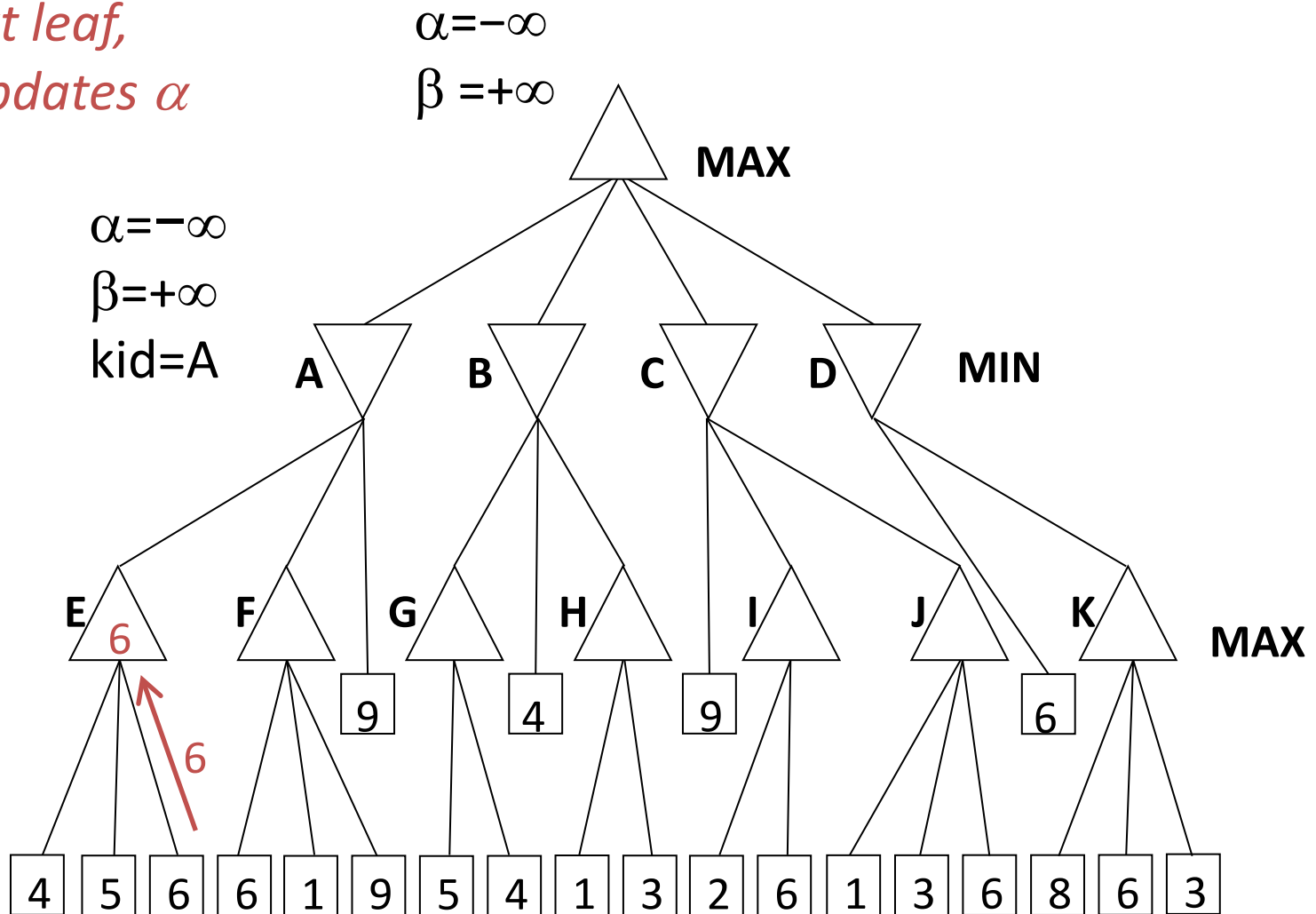


Longer Alpha-Beta Example

*see next leaf,
MAX updates α*



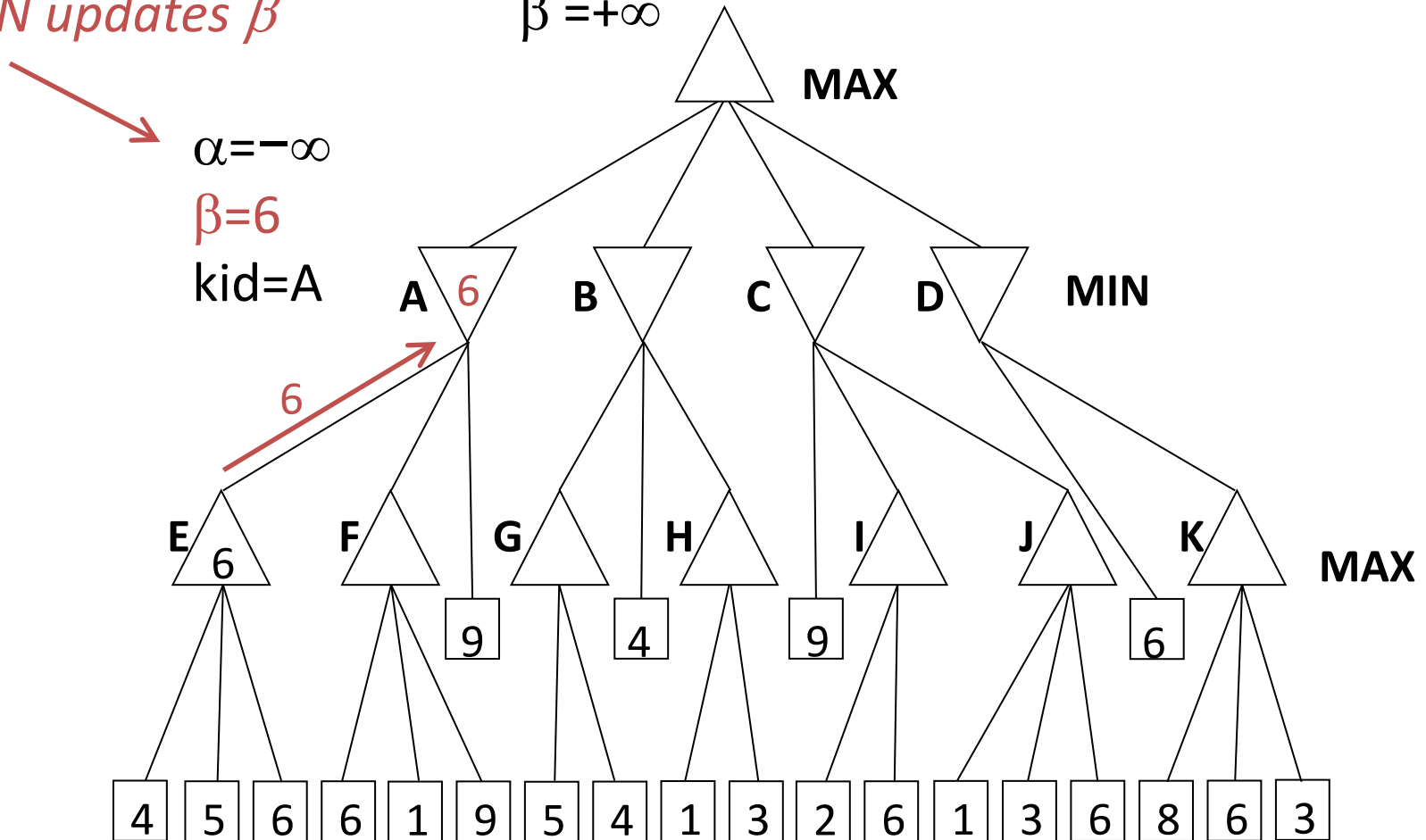
$\alpha=6$
 $\beta=+\infty$
kid=E



Longer Alpha-Beta Example

*return node value,
MIN updates β*

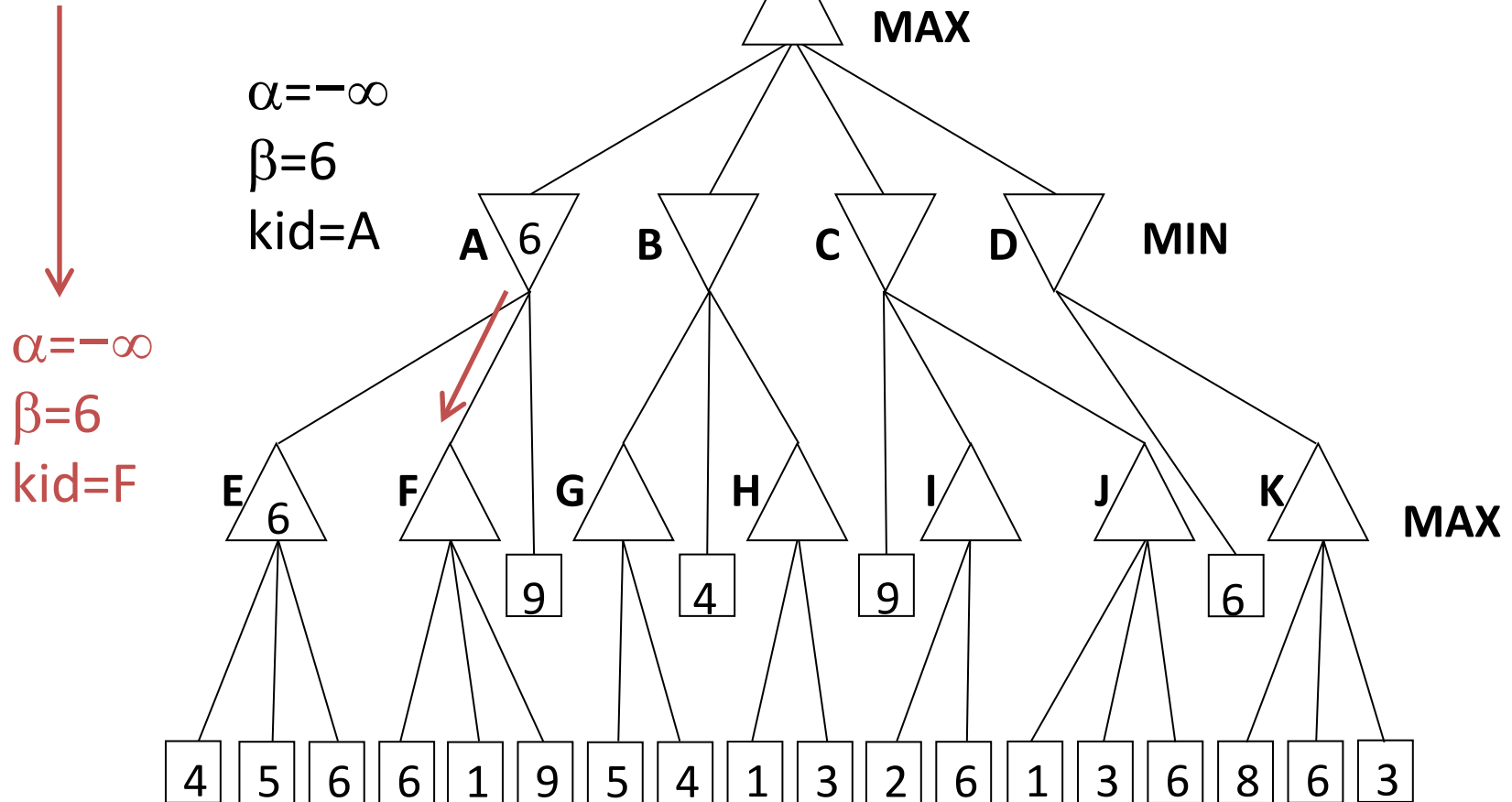
$\alpha = -\infty$
 $\beta = +\infty$



Longer Alpha-Beta Example

*current $\alpha, \beta,$
passed to kid F*

$\alpha = -\infty$
 $\beta = +\infty$



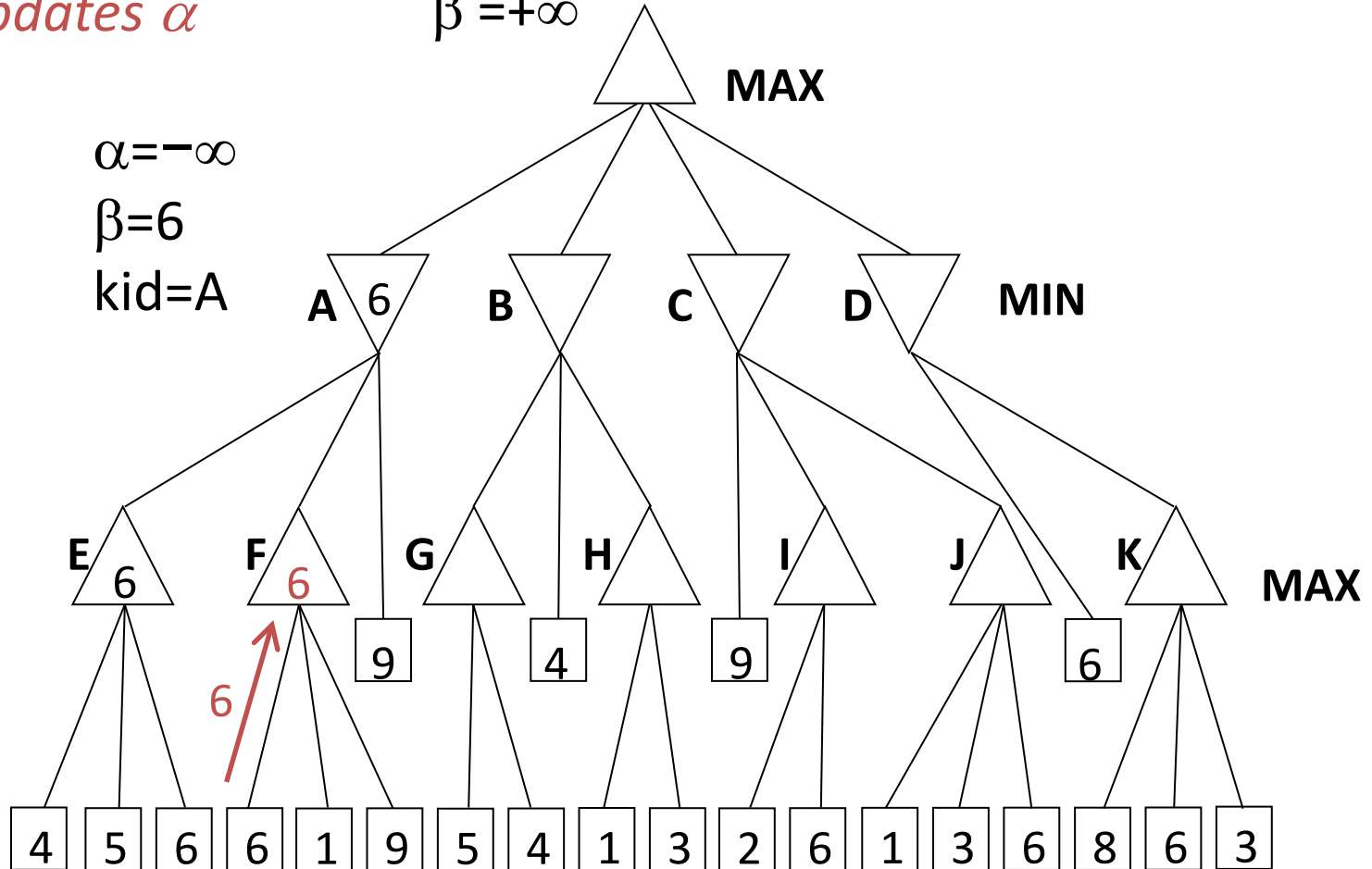
Longer Alpha-Beta Example

*see first leaf,
MAX updates α*

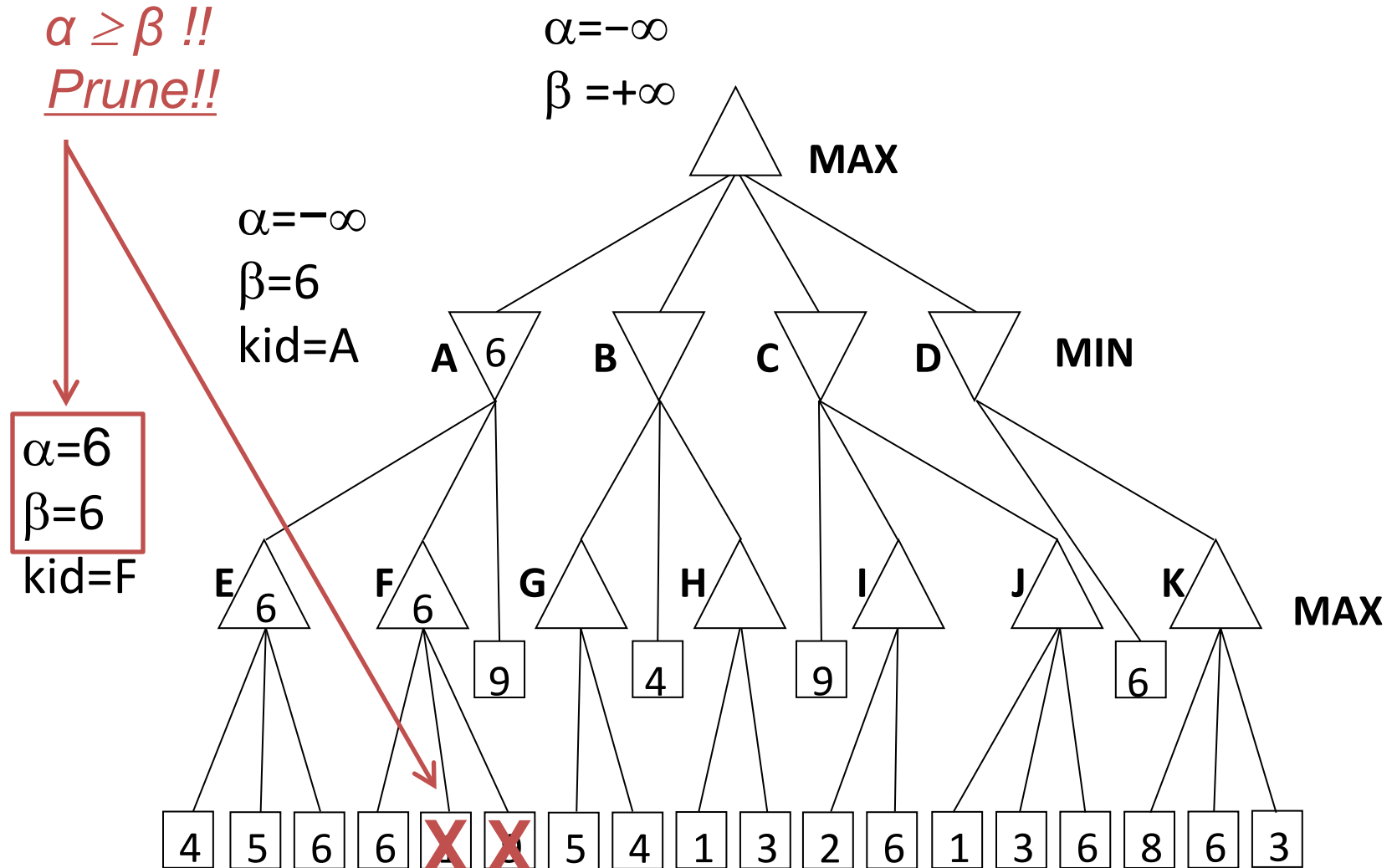


$\alpha=6$
 $\beta=6$
kid=F

$\alpha=-\infty$
 $\beta=+\infty$



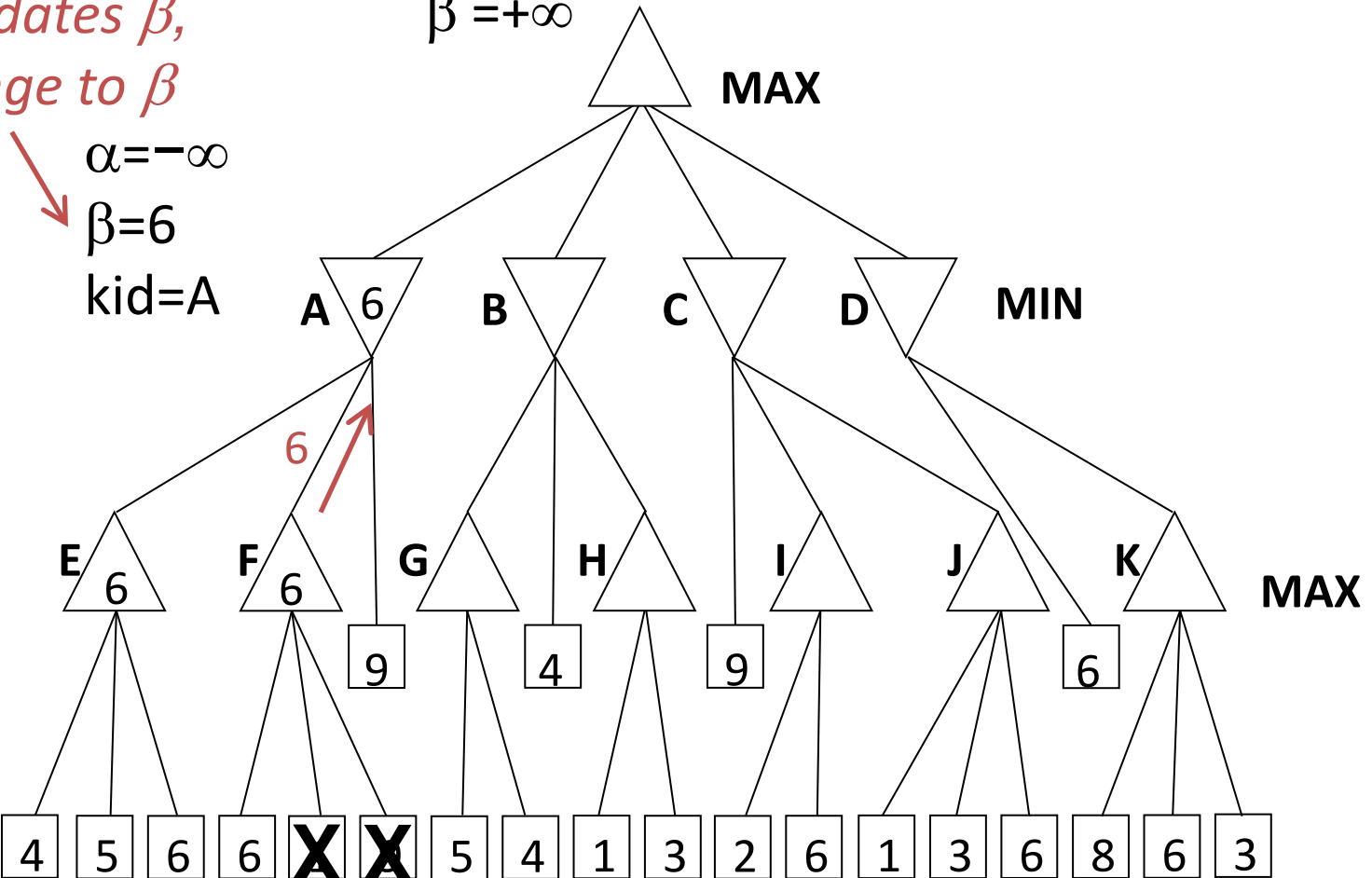
Longer Alpha-Beta Example



Longer Alpha-Beta Example

*return node value,
MIN updates β ,
no change to β*

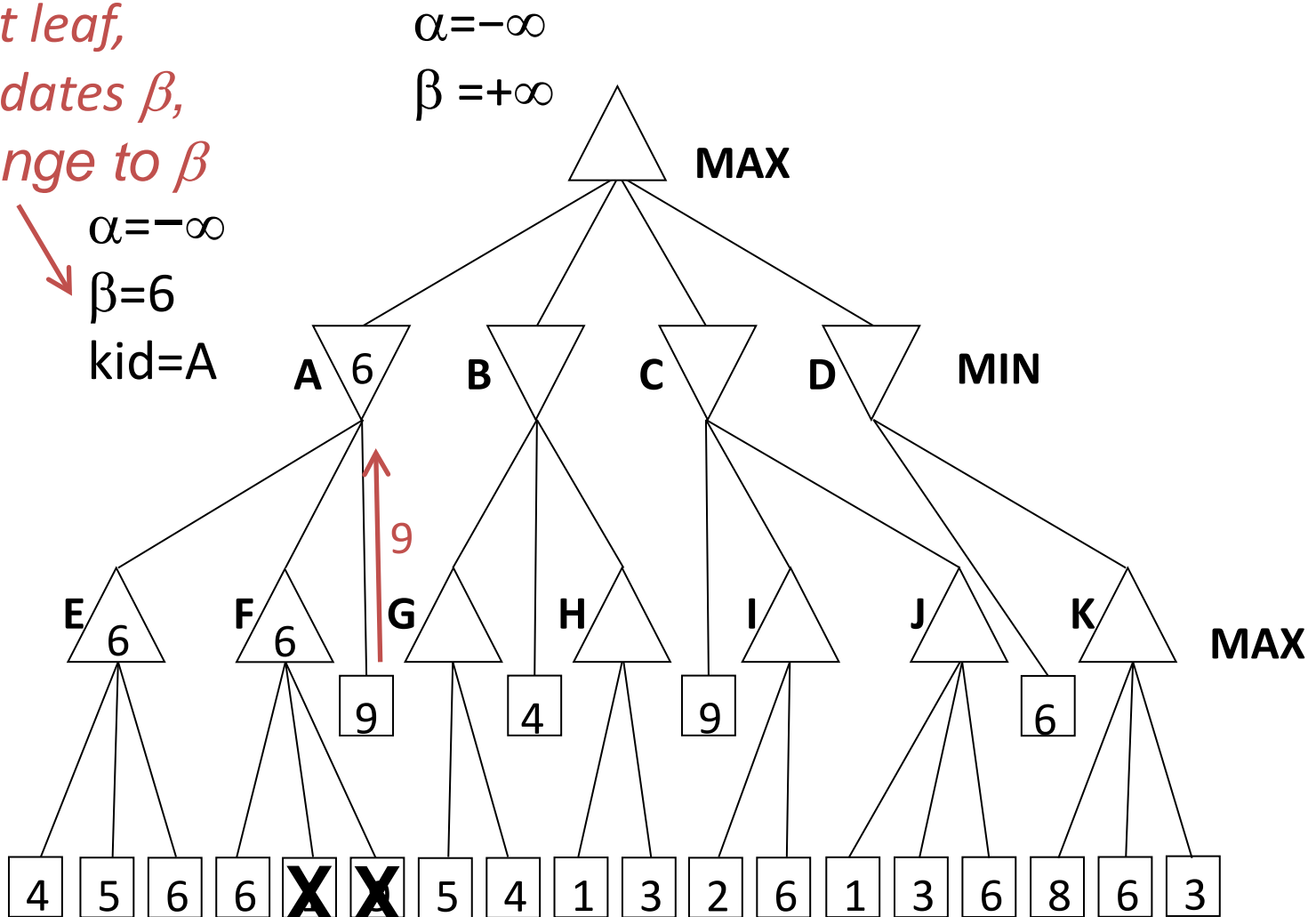
$\alpha = -\infty$
 $\beta = +\infty$



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

*see next leaf,
MIN updates β ,
no change to β*

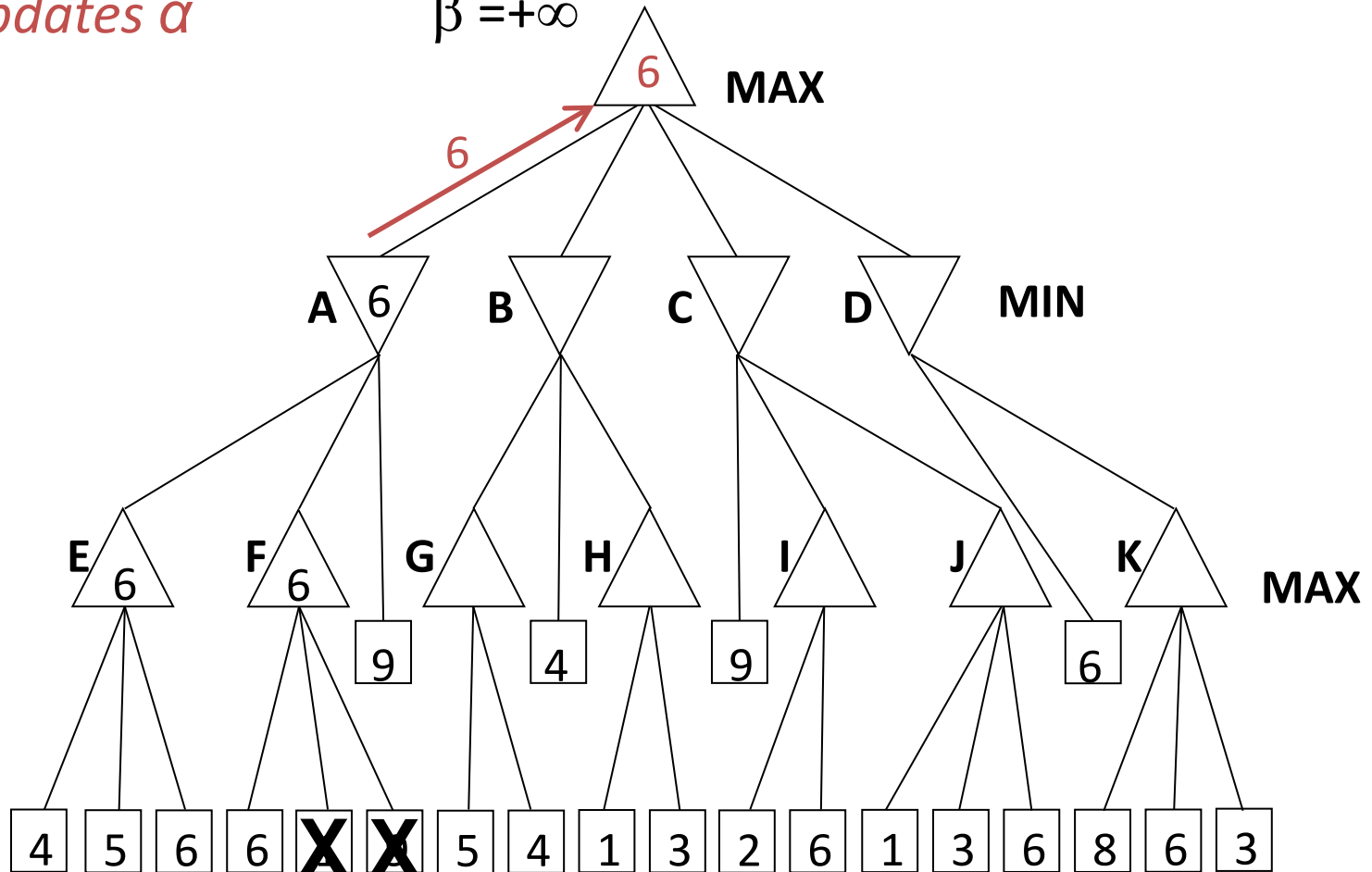


Longer Alpha-Beta Example

return node value, $\rightarrow \alpha=6$

MAX updates α

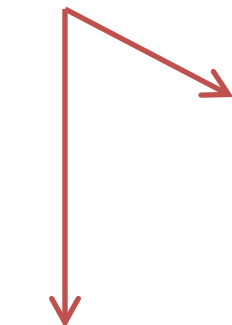
$\beta = +\infty$



Longer Alpha-Beta Example

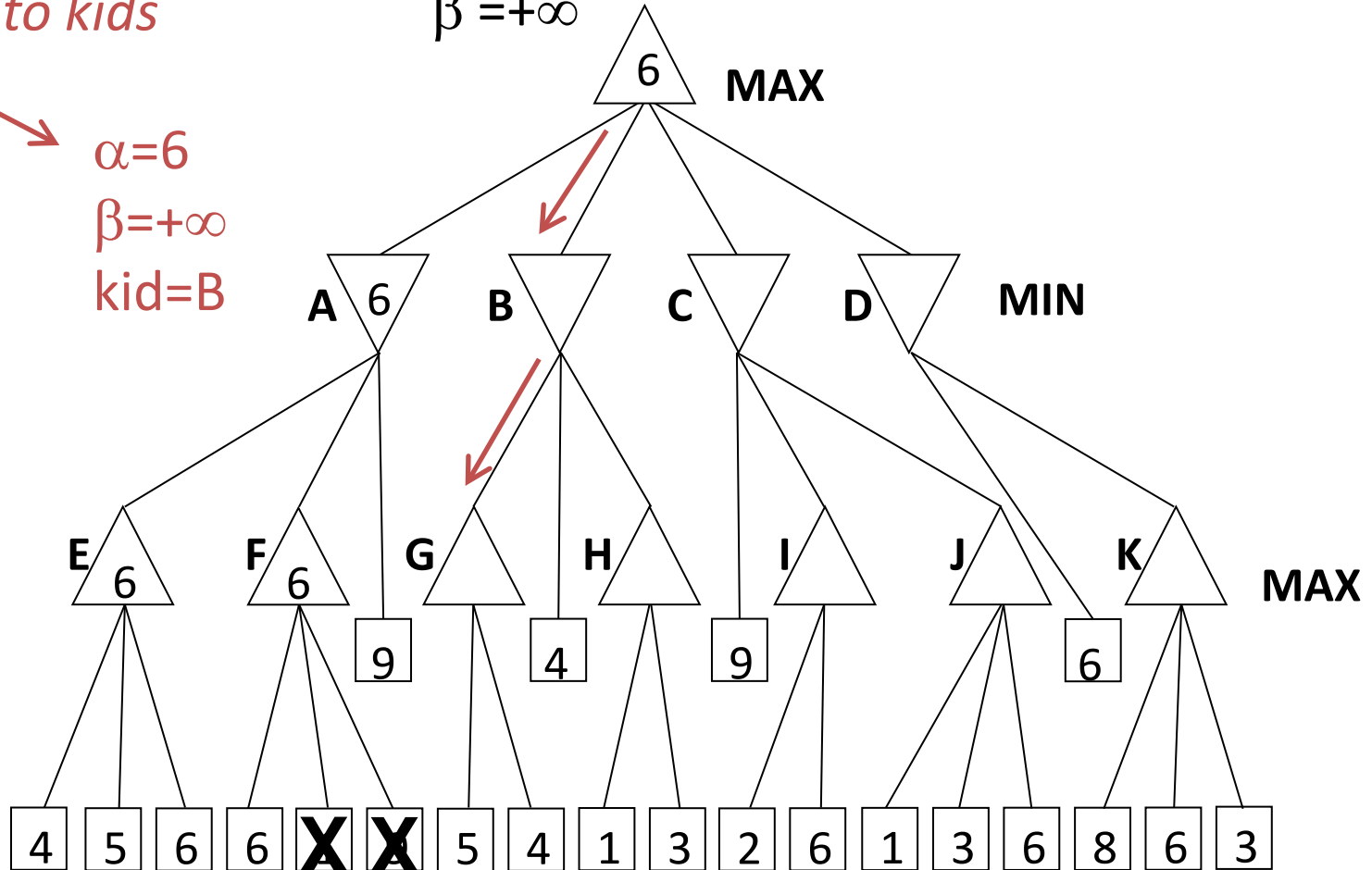
*current α , β ,
passed to kids*

$\alpha=6$
 $\beta=+\infty$



$\alpha=6$
 $\beta=+\infty$
kid=B

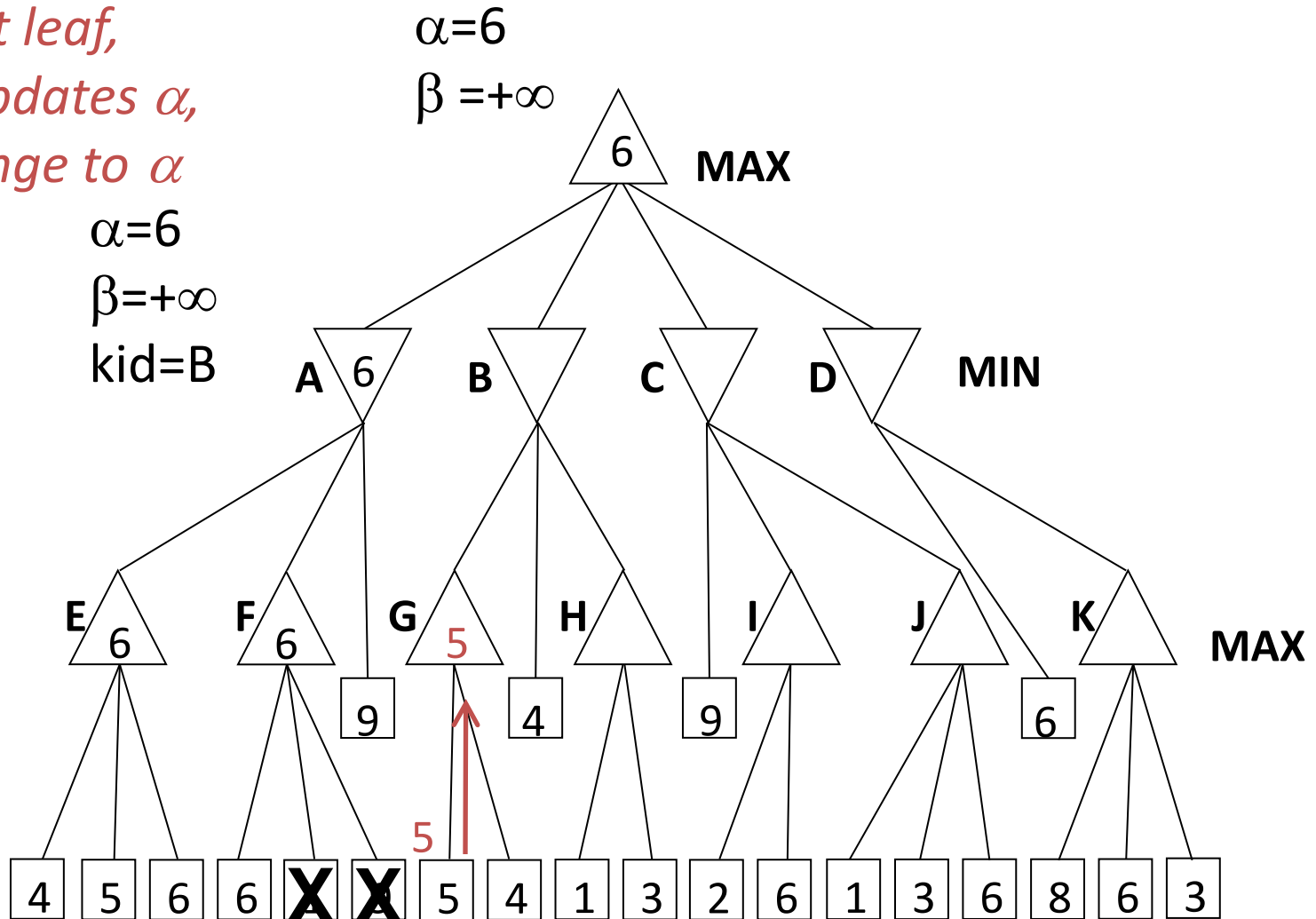
$\alpha=6$
 $\beta=+\infty$
kid=G



Longer Alpha-Beta Example

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

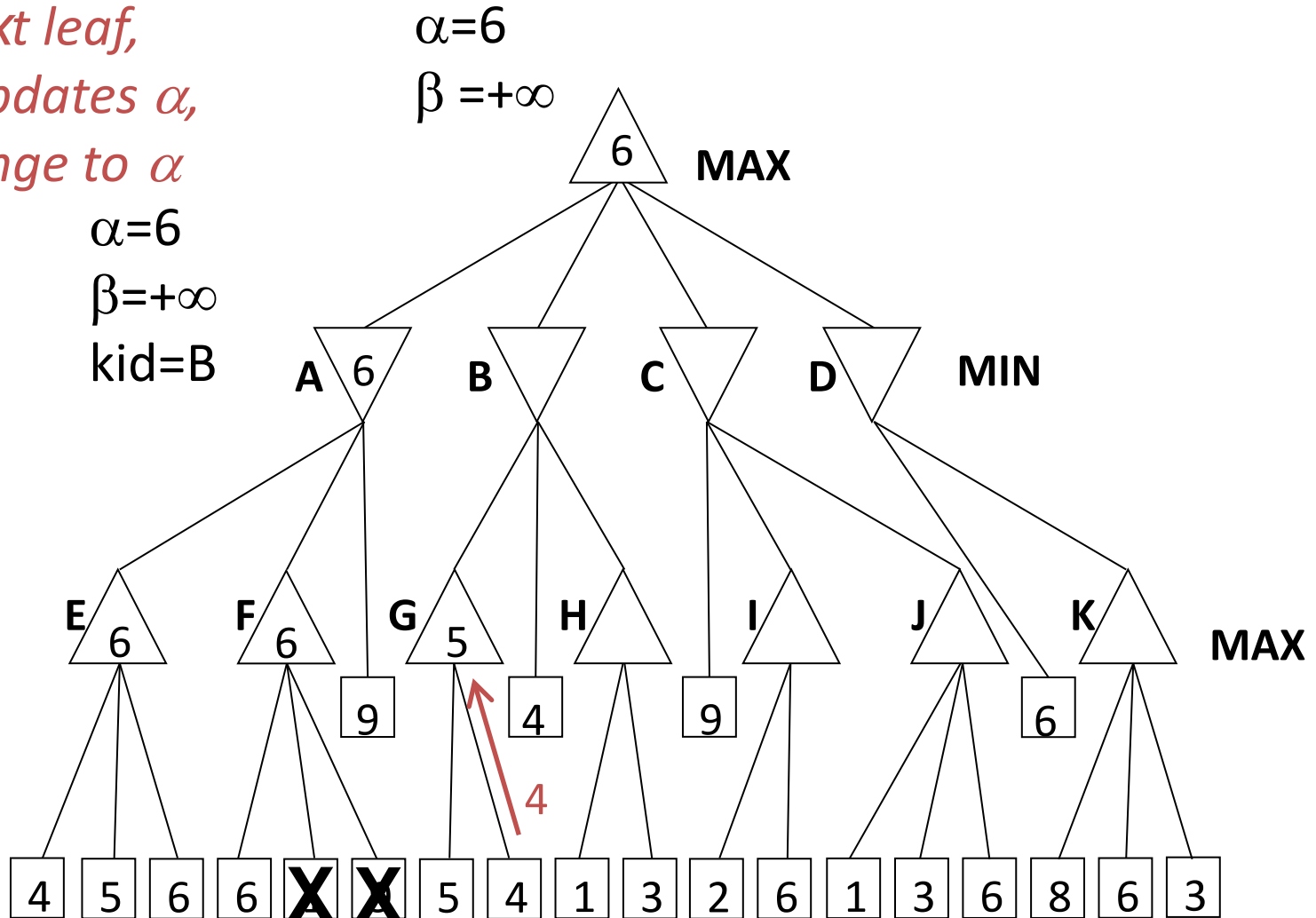
$\alpha=6$
 $\beta=+\infty$
kid=G



Longer Alpha-Beta Example

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

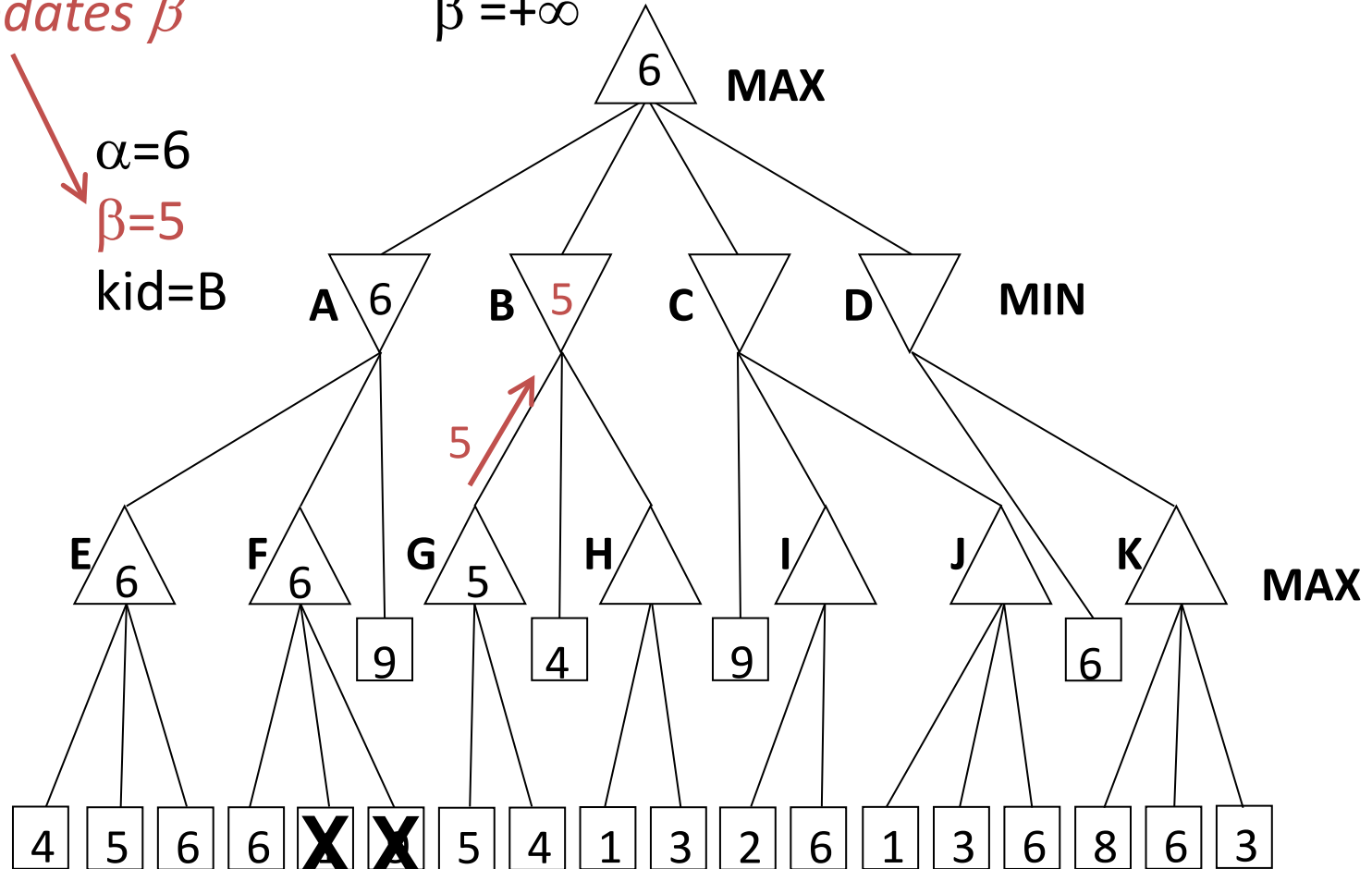
$\alpha=6$
 $\beta=+\infty$
kid=G



Longer Alpha-Beta Example

*return node value,
MIN updates β*

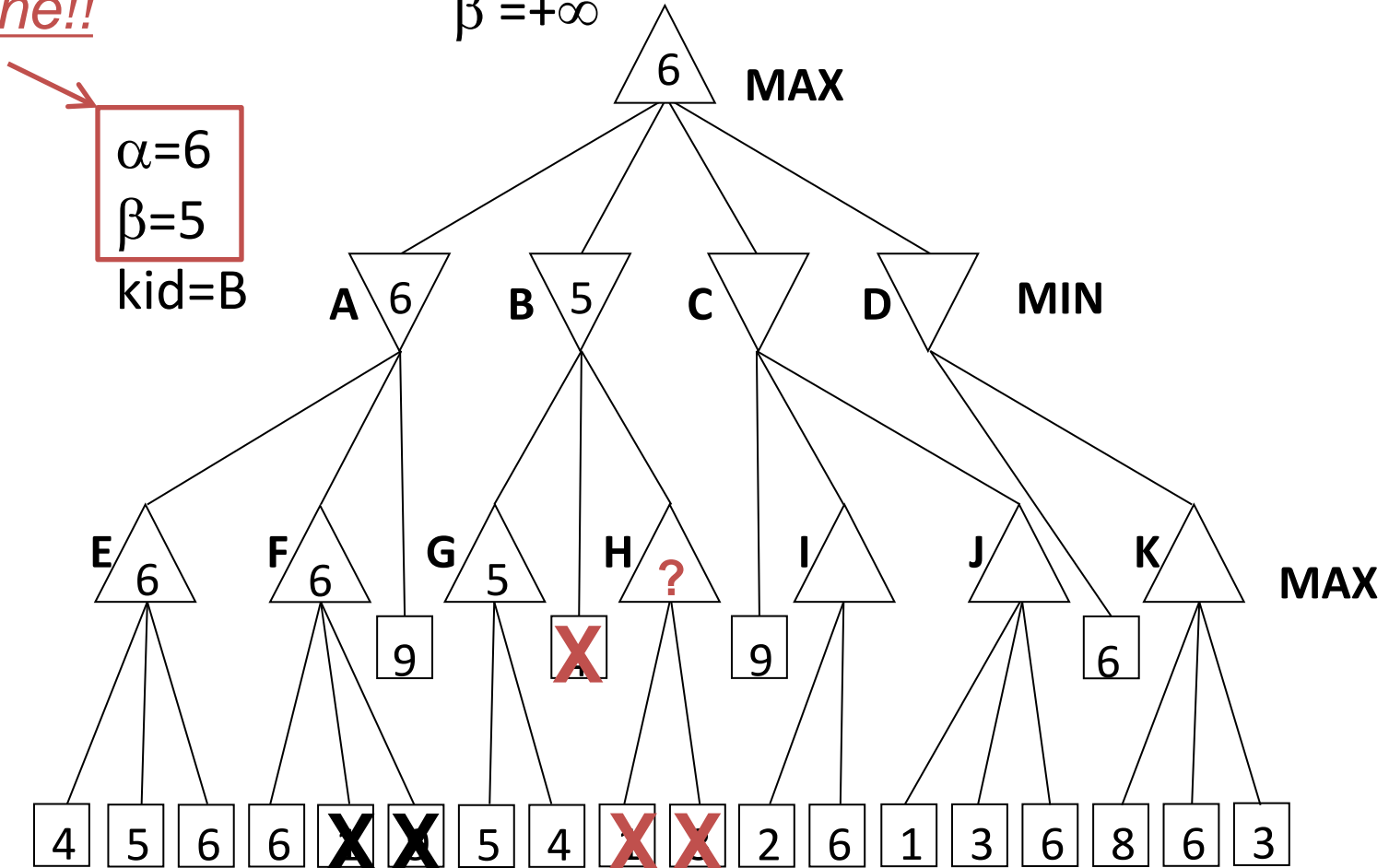
$\alpha=6$
 $\beta=+\infty$



Longer Alpha-Beta Example

$\alpha \geq \beta$!!
Prune!!

$\alpha=6$
 $\beta = +\infty$



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.

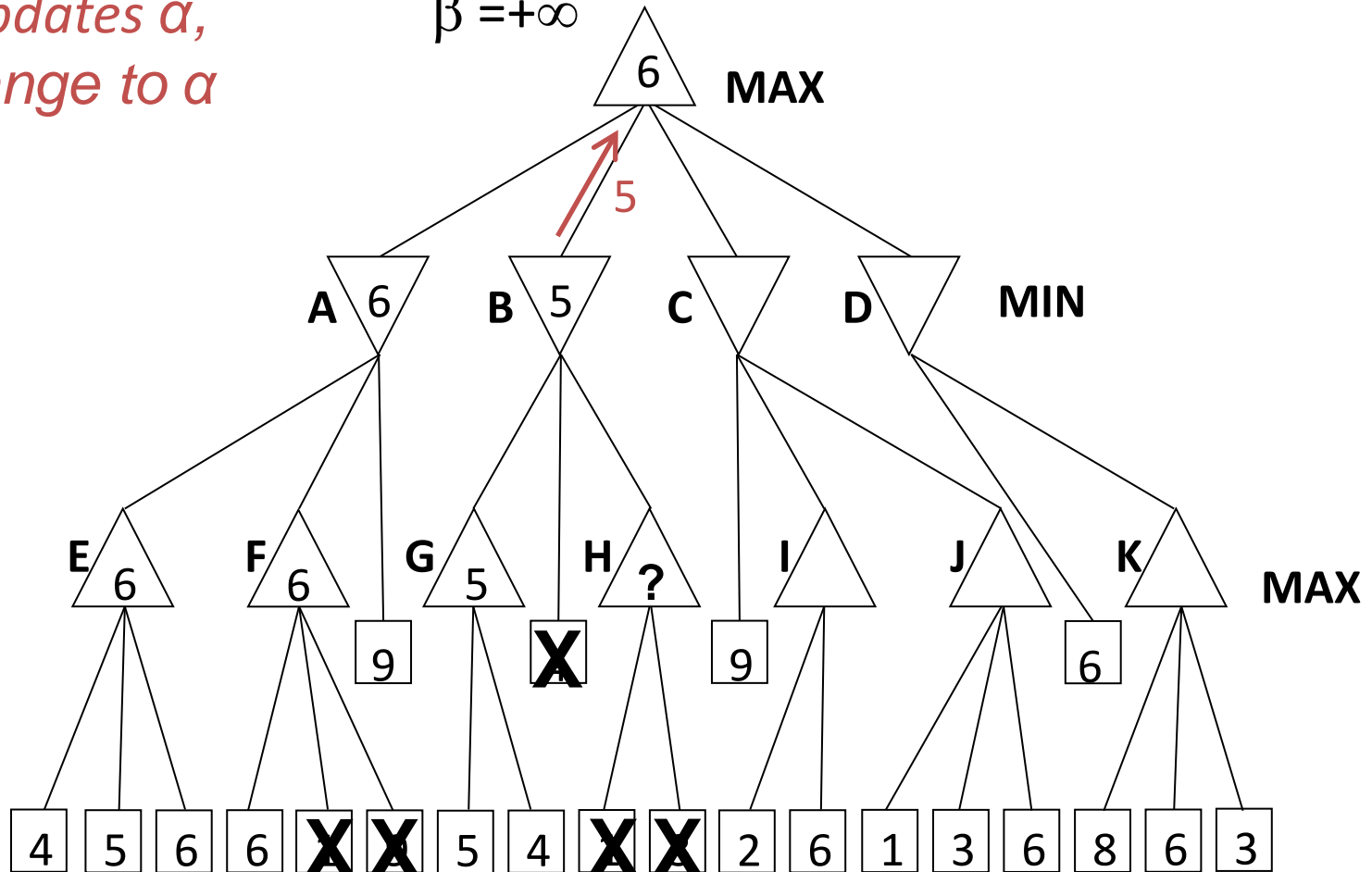
Longer Alpha-Beta Example

return node value, $\longrightarrow \alpha=6$

MAX updates α ,

no change to α

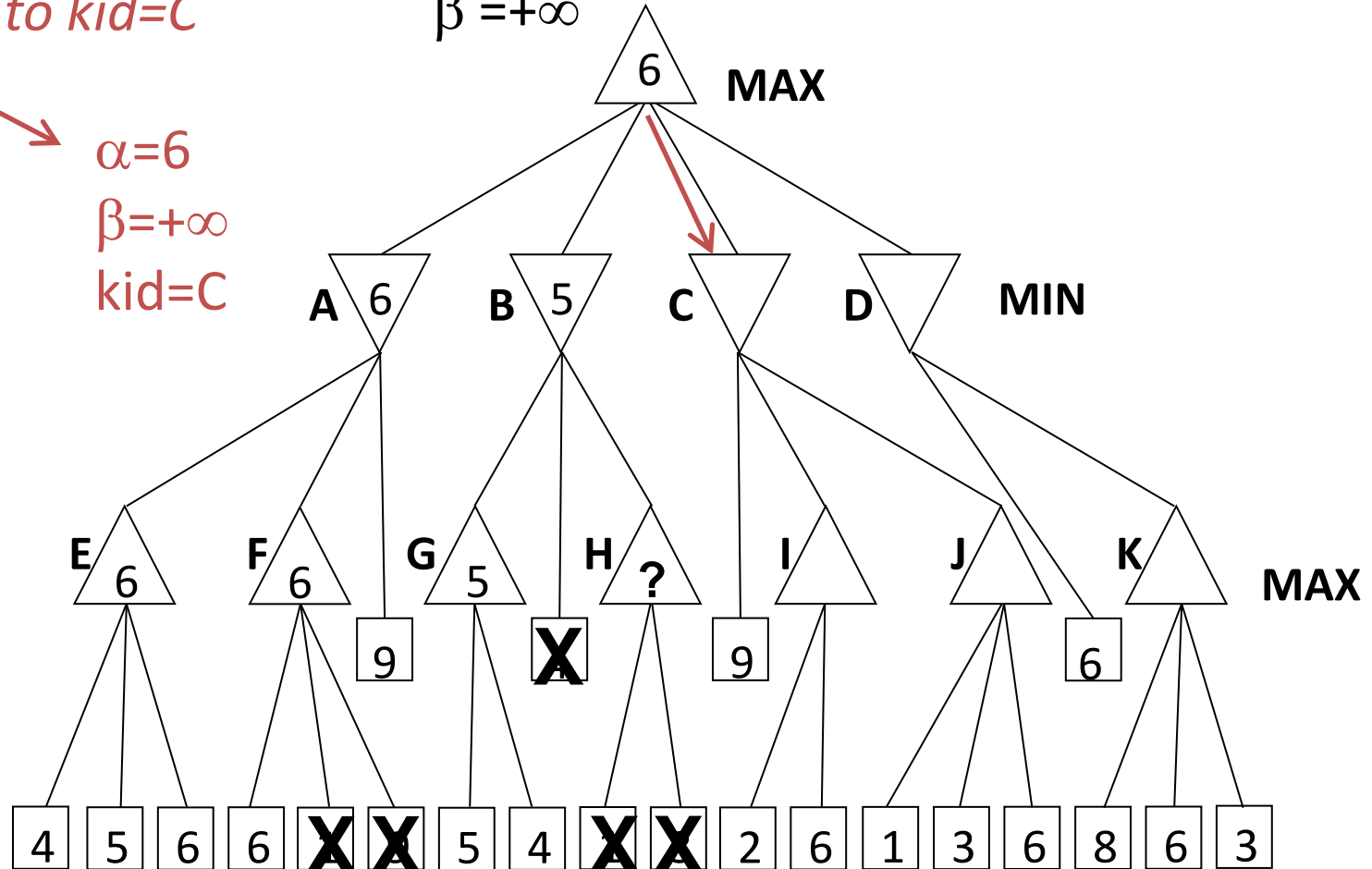
$\beta = +\infty$



Longer Alpha-Beta Example

*current α , β ,
passed to kid=C*

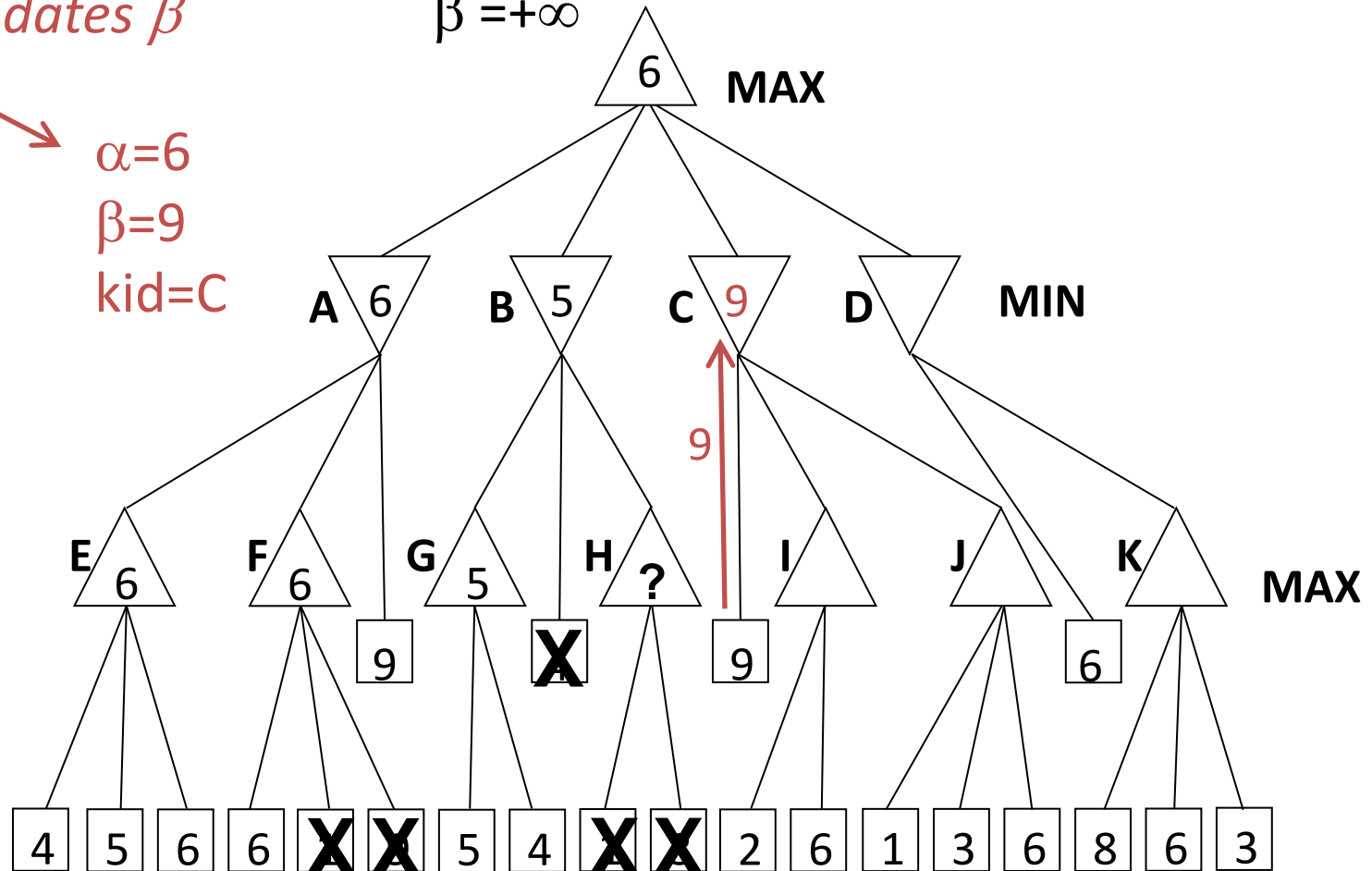
$\alpha=6$
 $\beta=+\infty$



Longer Alpha-Beta Example

*see first leaf,
MIN updates β*

$\alpha=6$
 $\beta=+\infty$



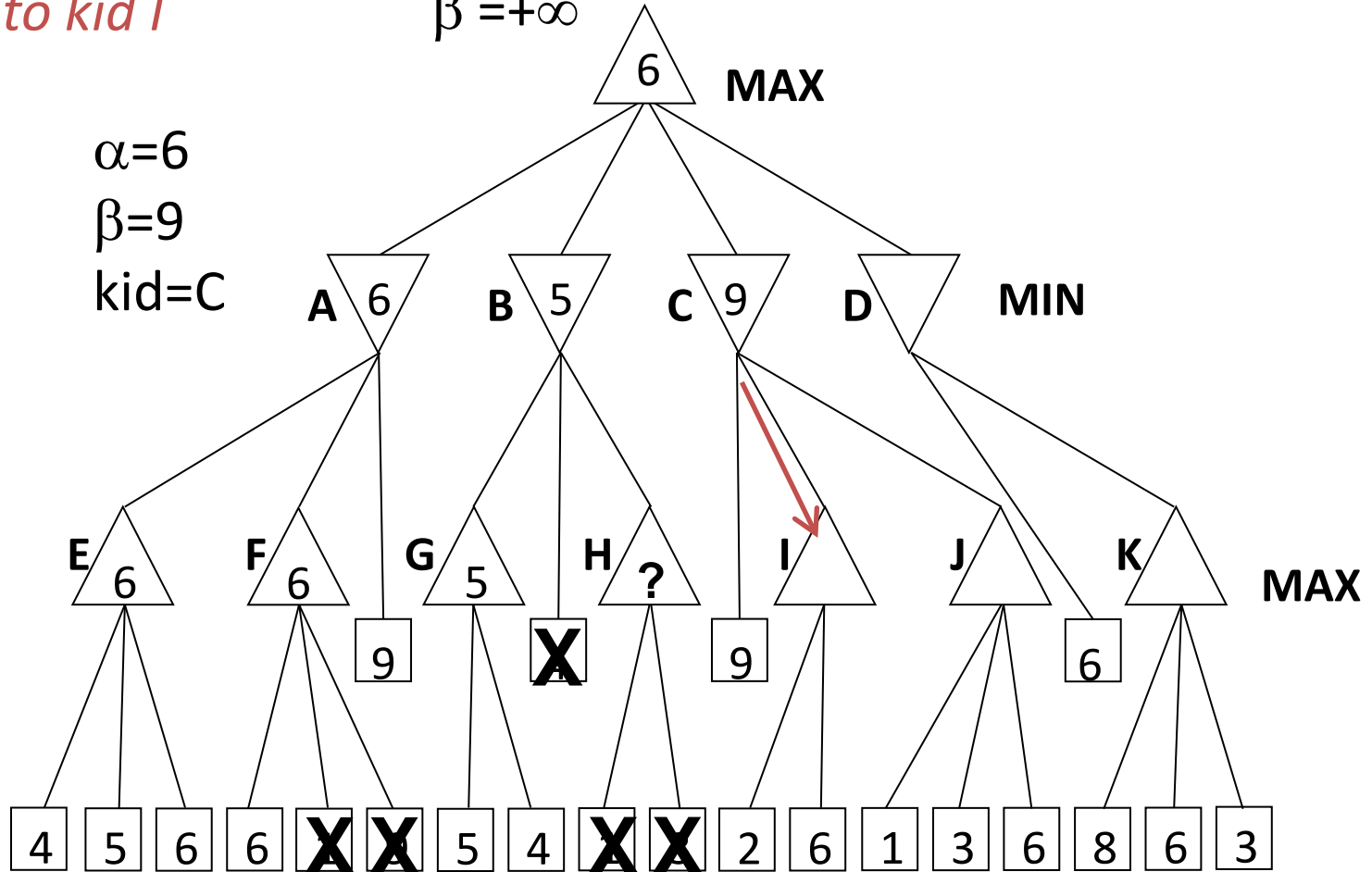
Longer Alpha-Beta Example

*current α , β ,
passed to kid I*

$\alpha=6$
 $\beta=+\infty$



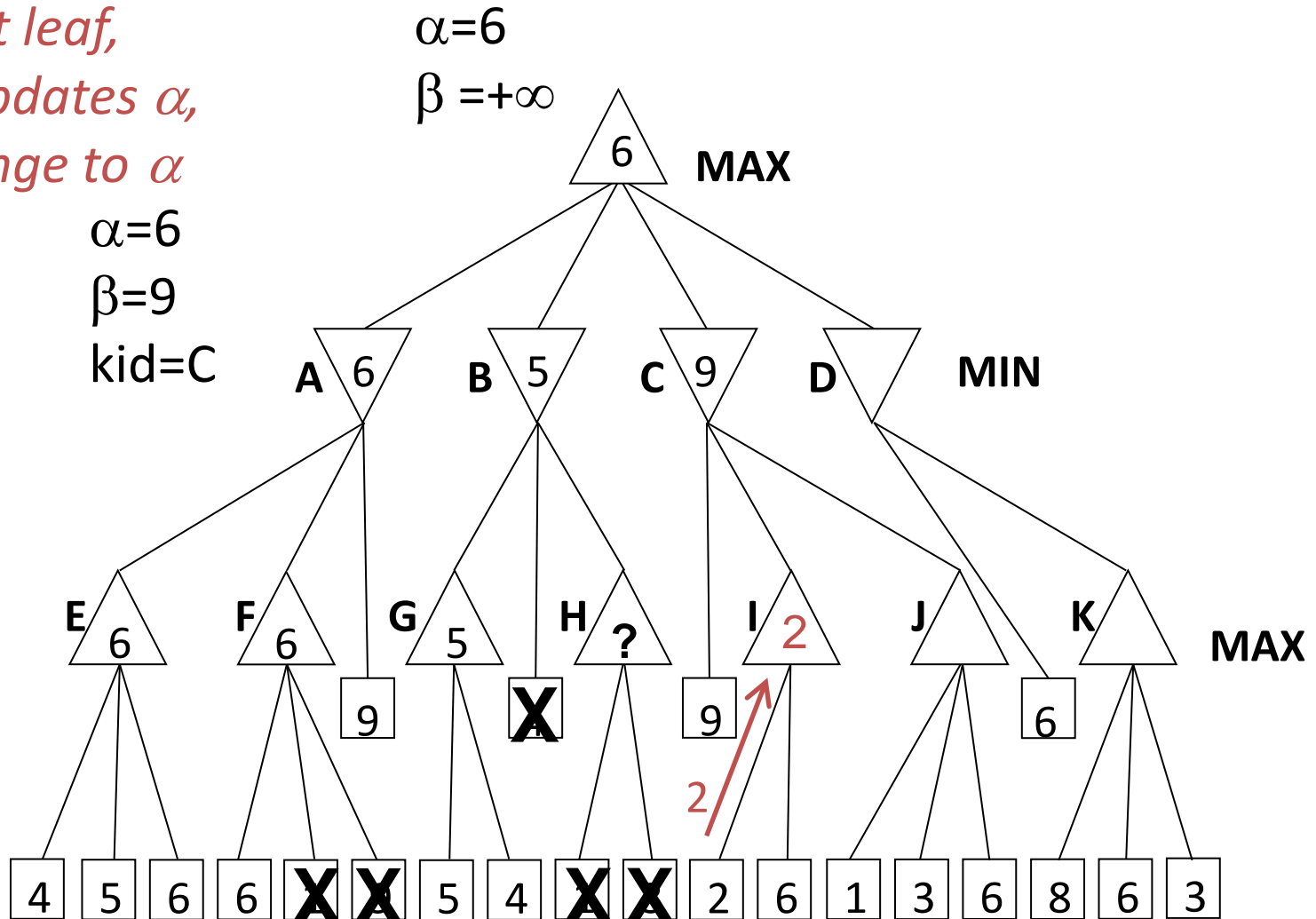
*$\alpha=6$
 $\beta=9$
kid=I*



Longer Alpha-Beta Example

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

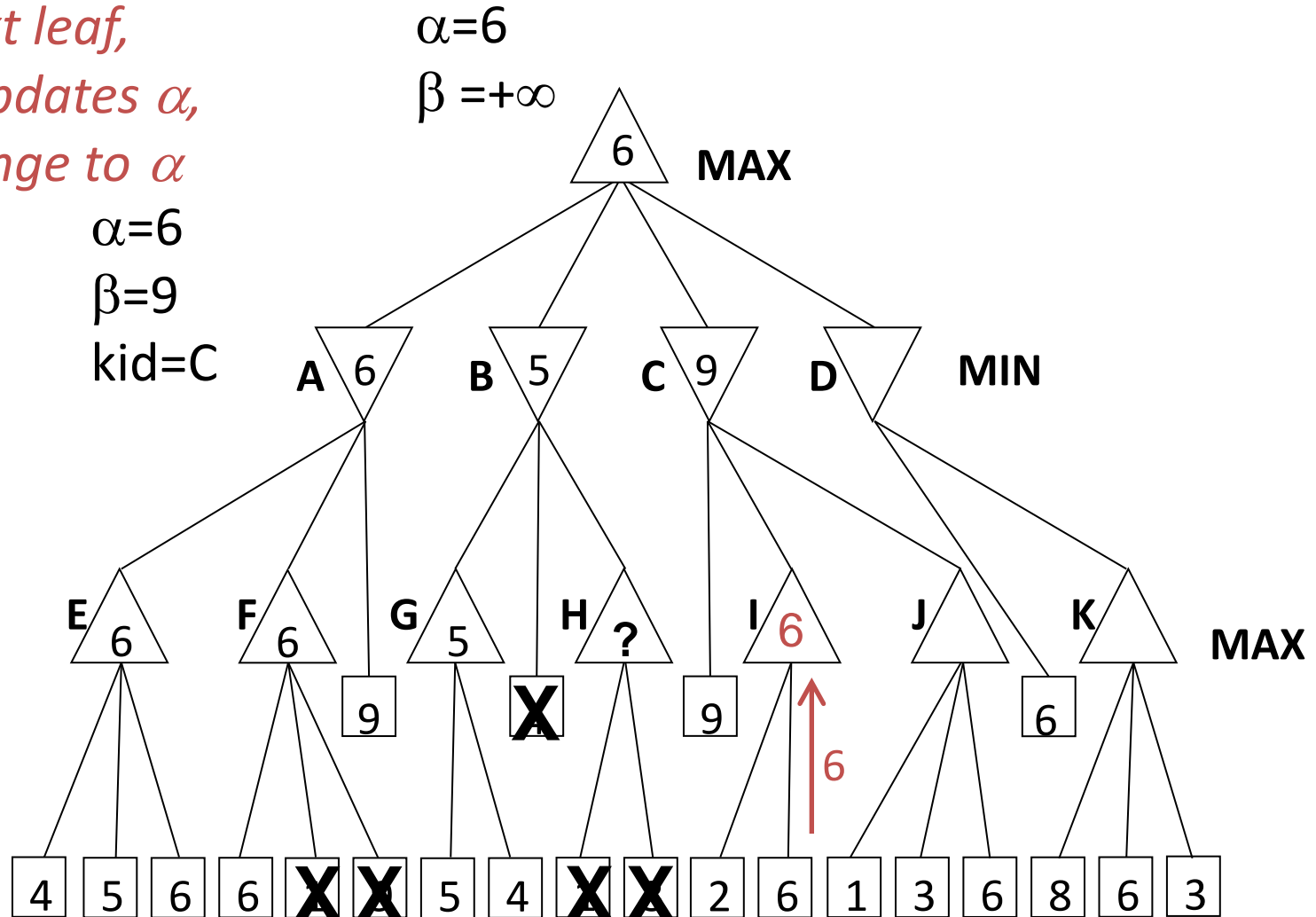
$\alpha=6$
 $\beta=9$
kid=1



Longer Alpha-Beta Example

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

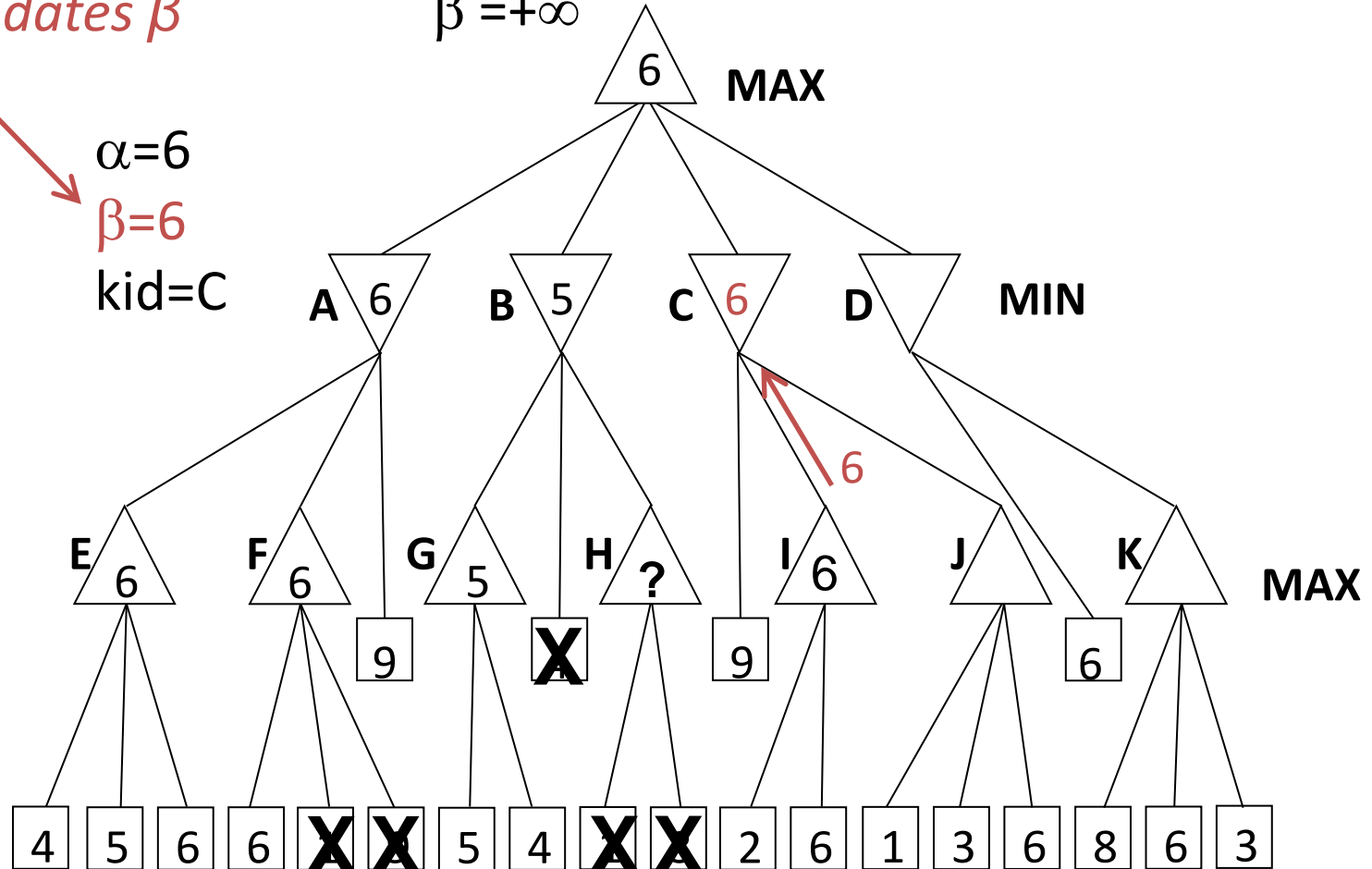
$\alpha=6$
 $\beta=9$
kid=l



Longer Alpha-Beta Example

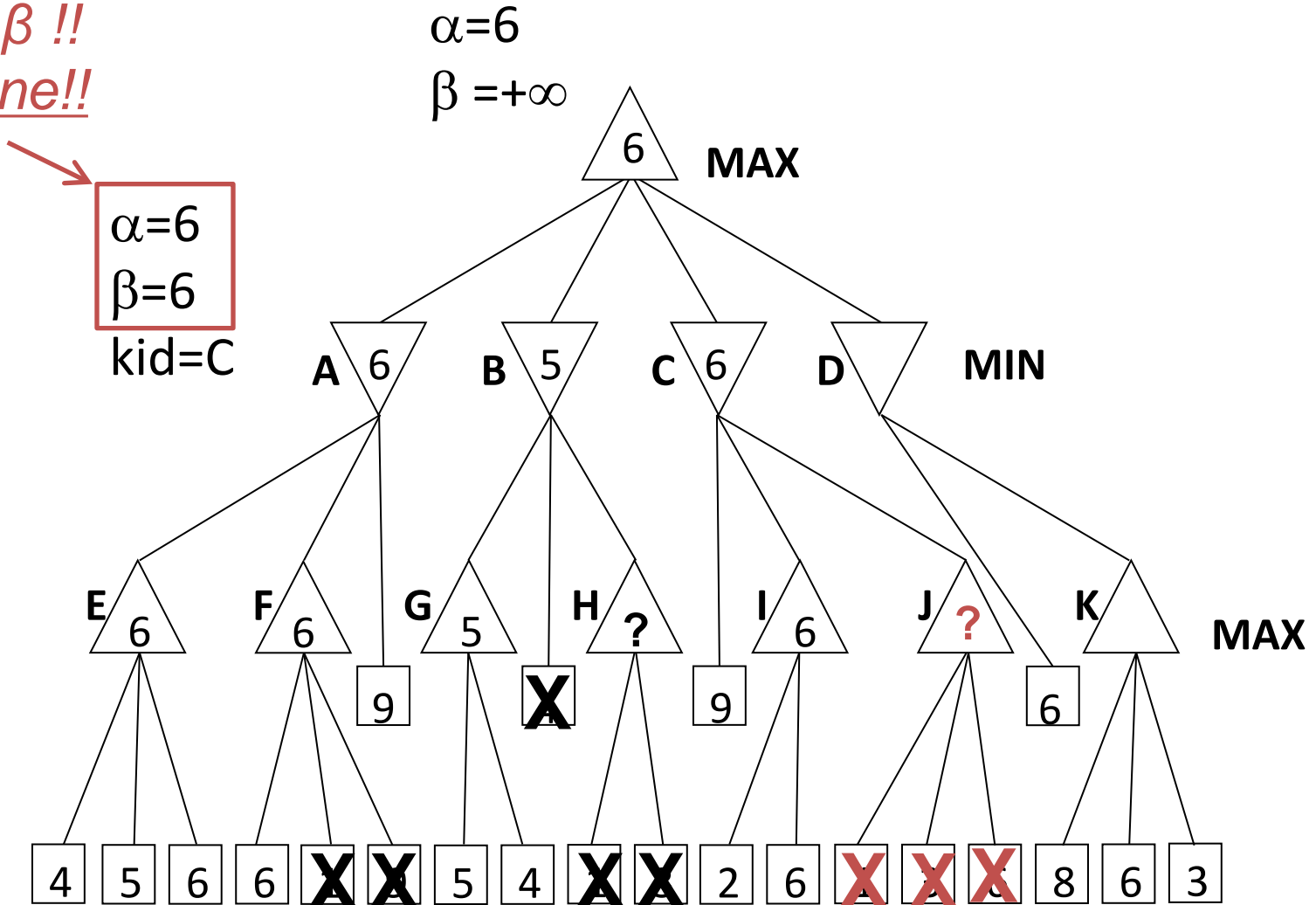
*return node value,
MIN updates β*

$\alpha=6$
 $\beta=+\infty$



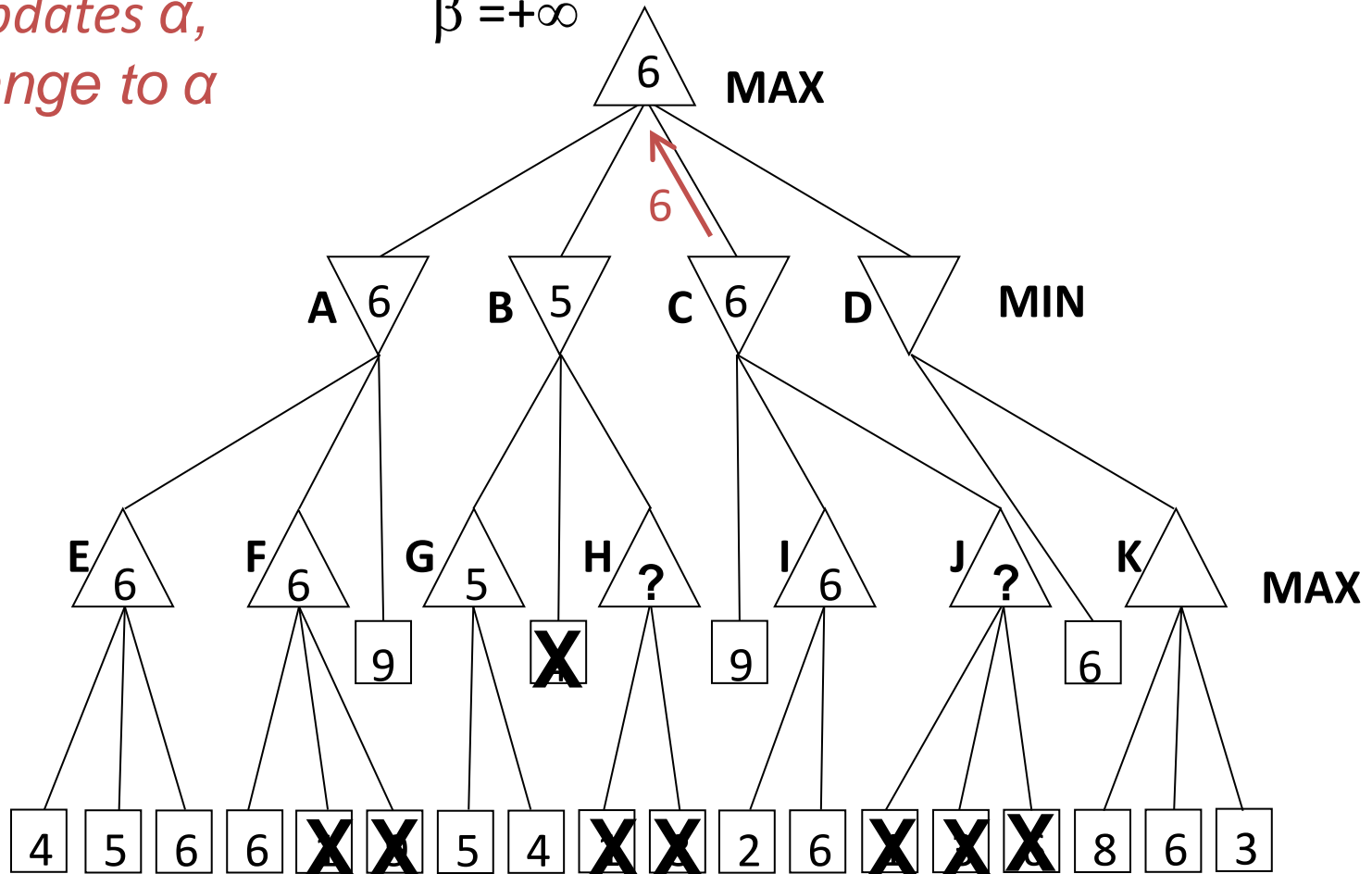
Longer Alpha-Beta Example

$\alpha \geq \beta$!!
Prune!!



Longer Alpha-Beta Example

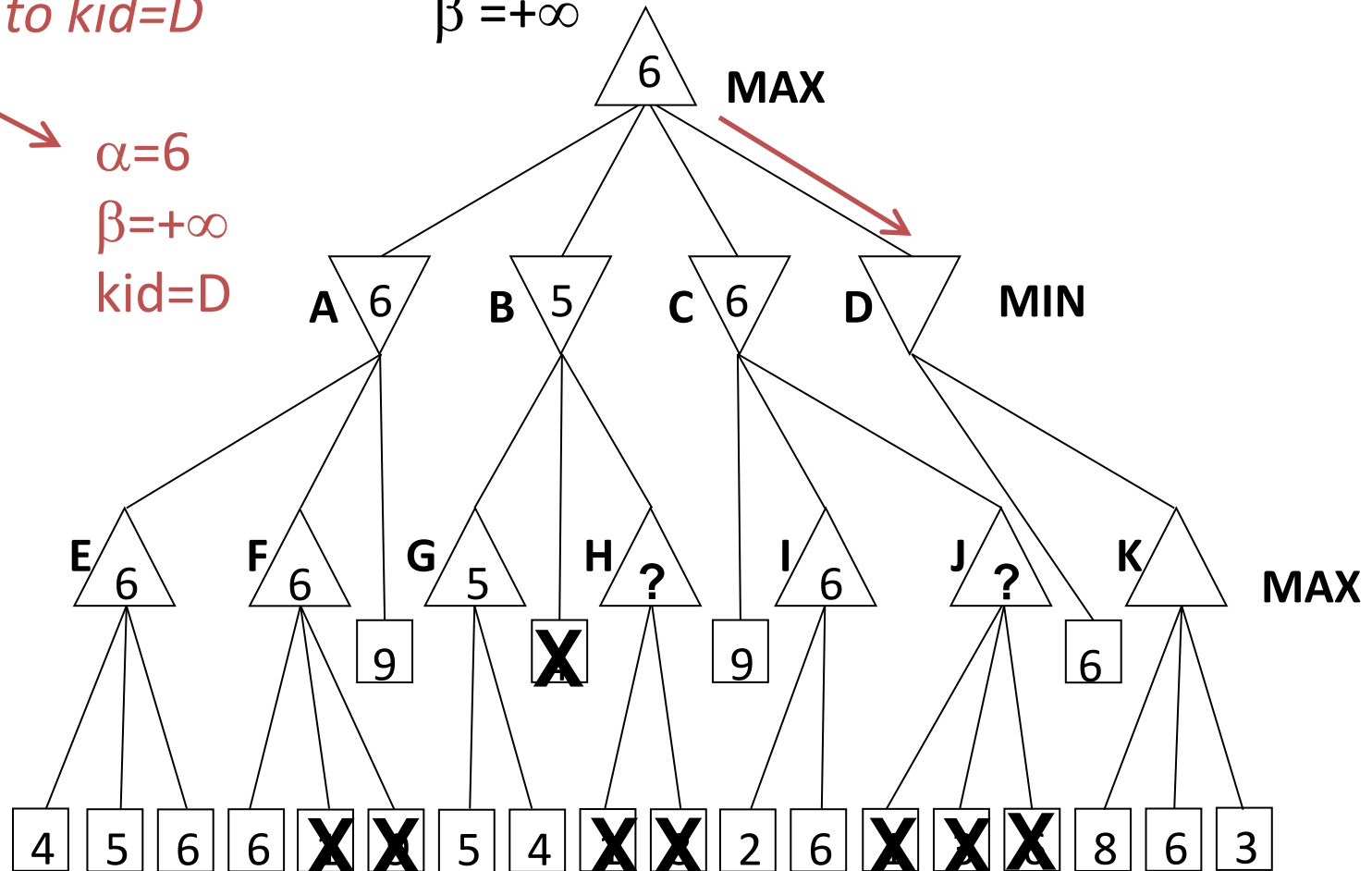
return node value, $\longrightarrow \alpha=6$
MAX updates α ,
no change to α $\beta = +\infty$



Longer Alpha-Beta Example

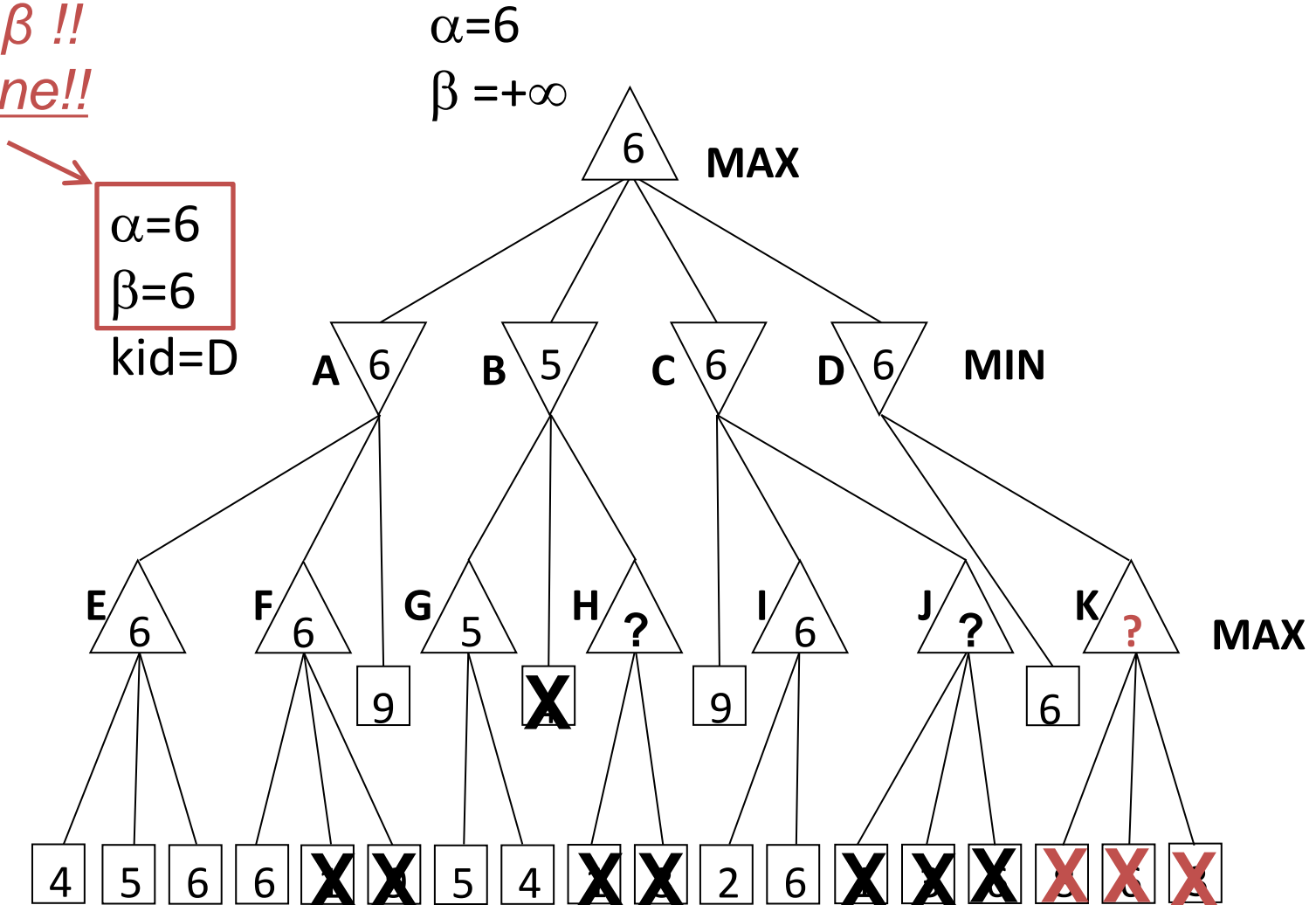
*current α , β ,
passed to kid=D*

$\alpha=6$
 $\beta=+\infty$



Longer Alpha-Beta Example

$\alpha \geq \beta$!!
Prune!!

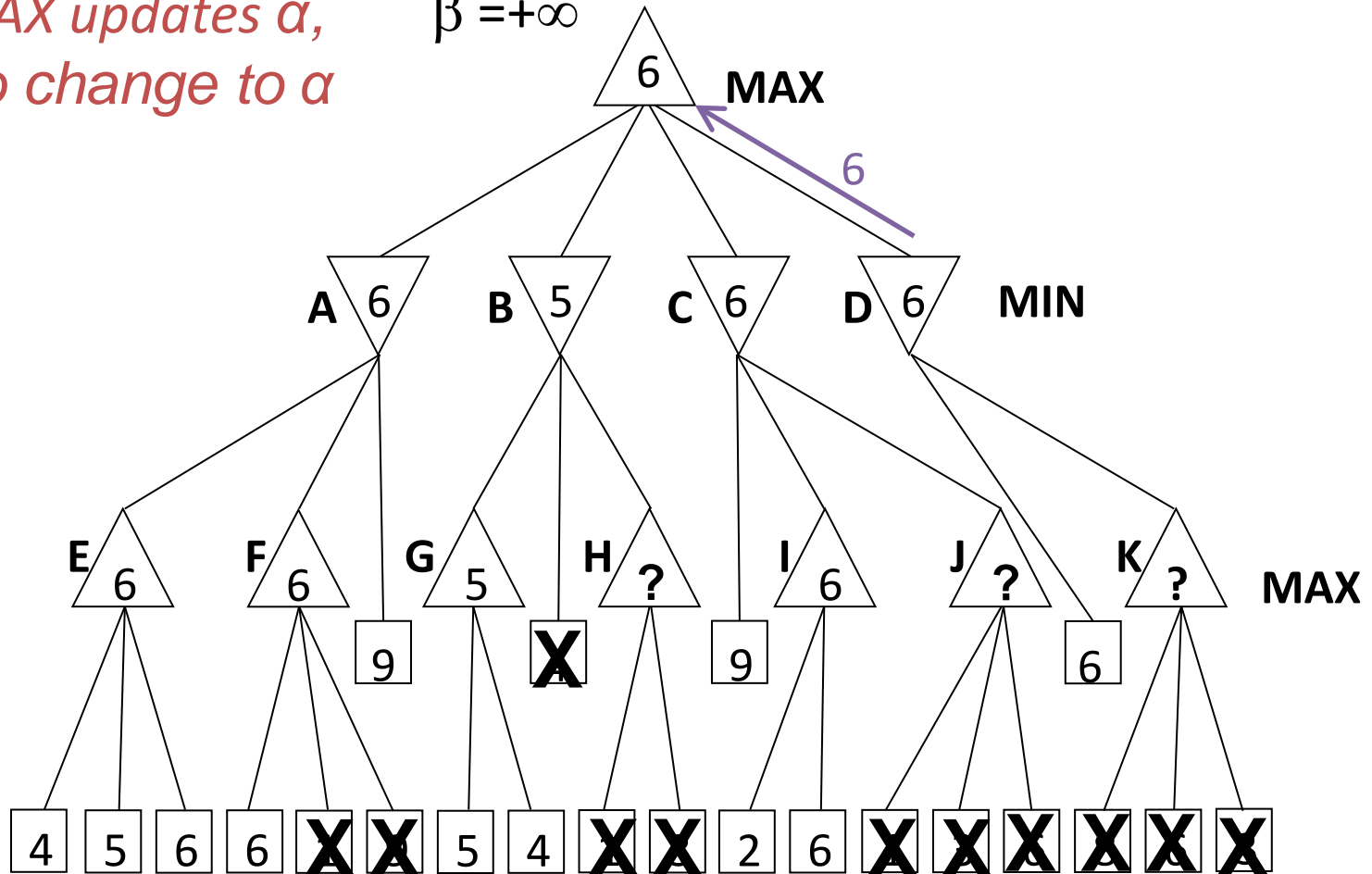


Alpha-Beta Example #2

return node value, $\alpha=6$

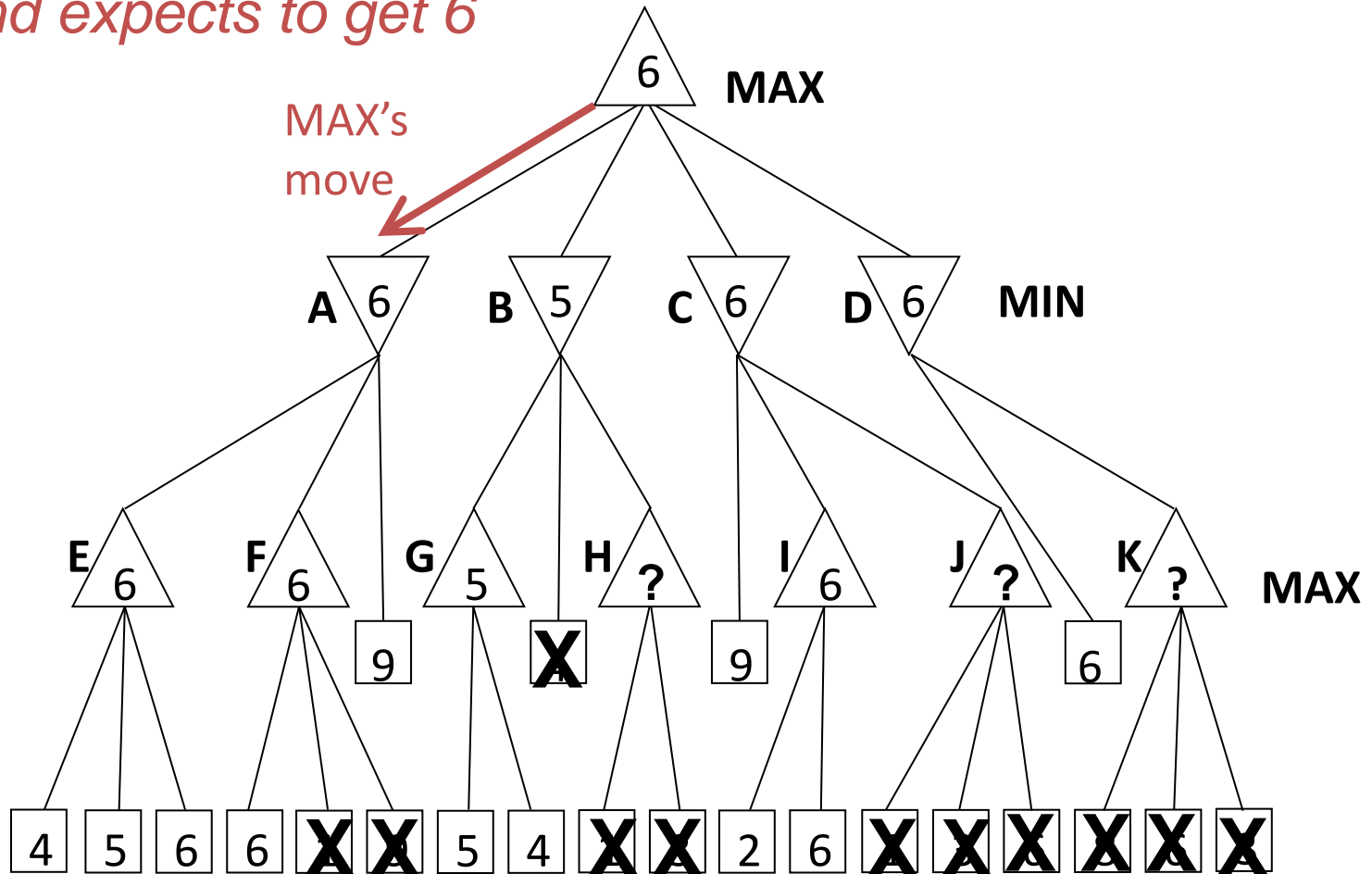
MAX updates α , $\beta = +\infty$

no change to α



Alpha-Beta Example #2

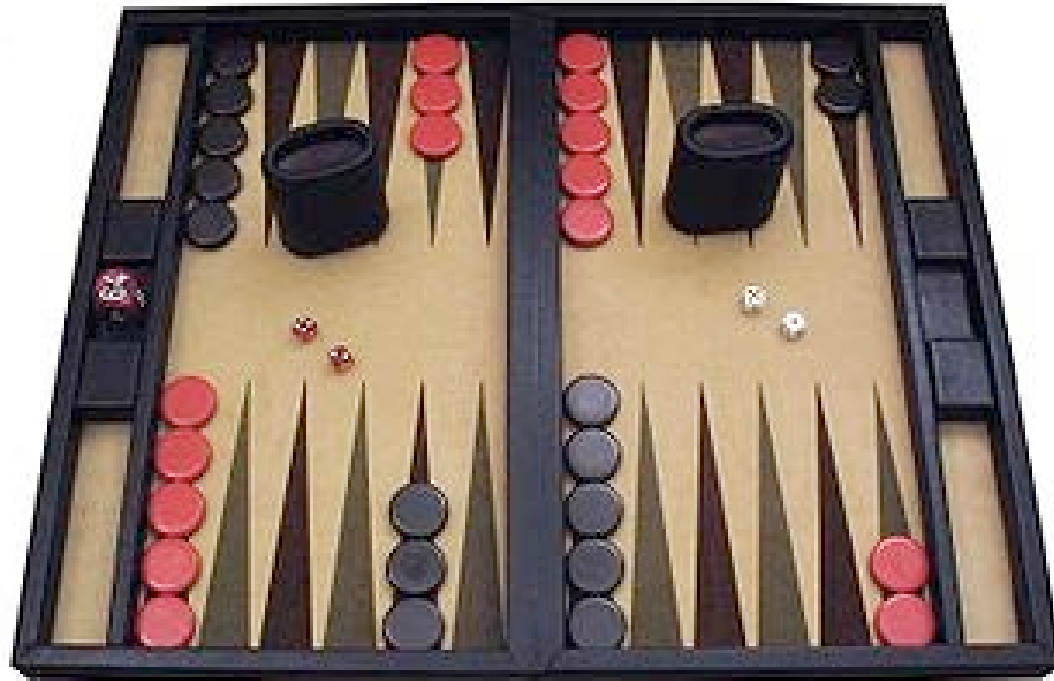
*MAX moves to A,
and expects to get 6*



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.

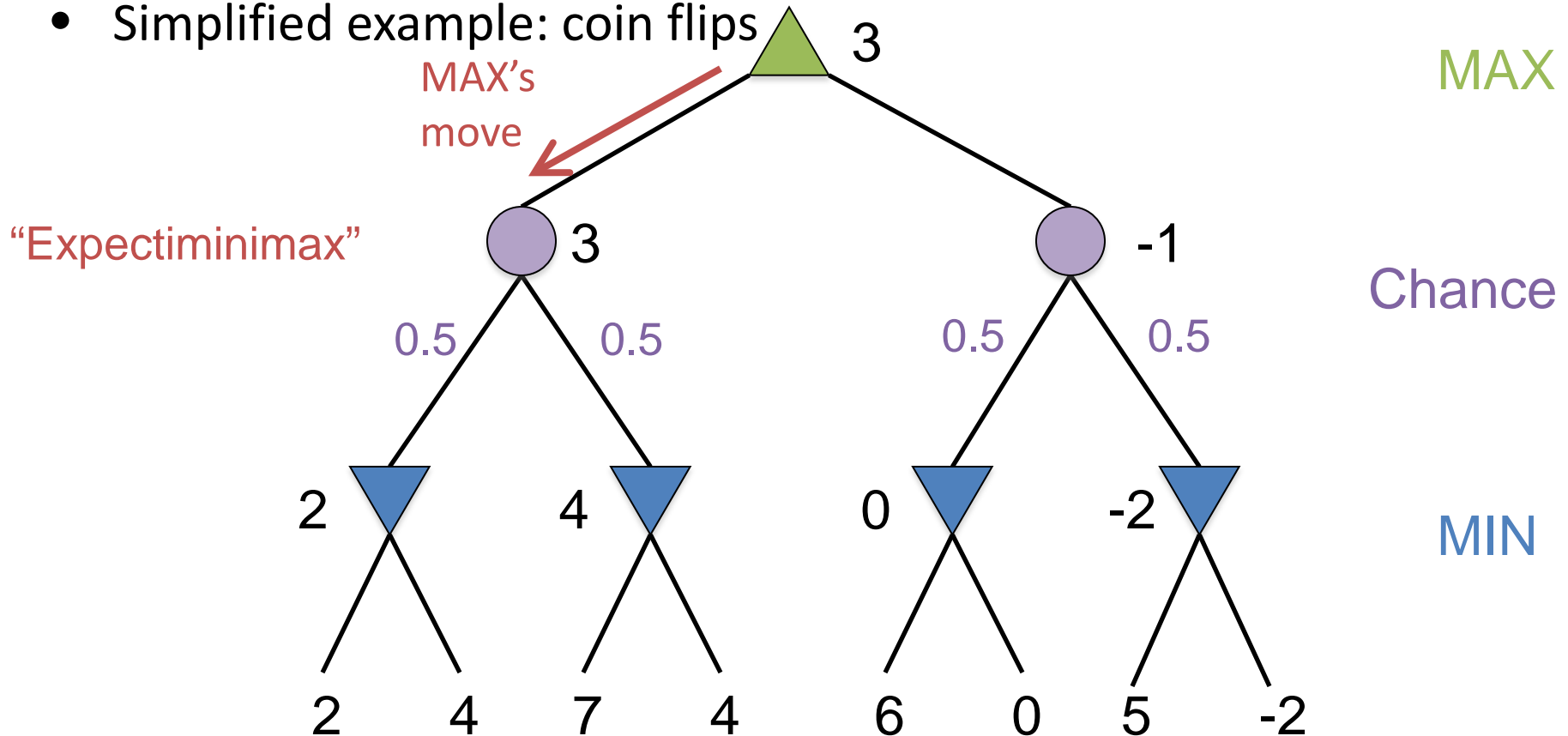
Nondeterministic games

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



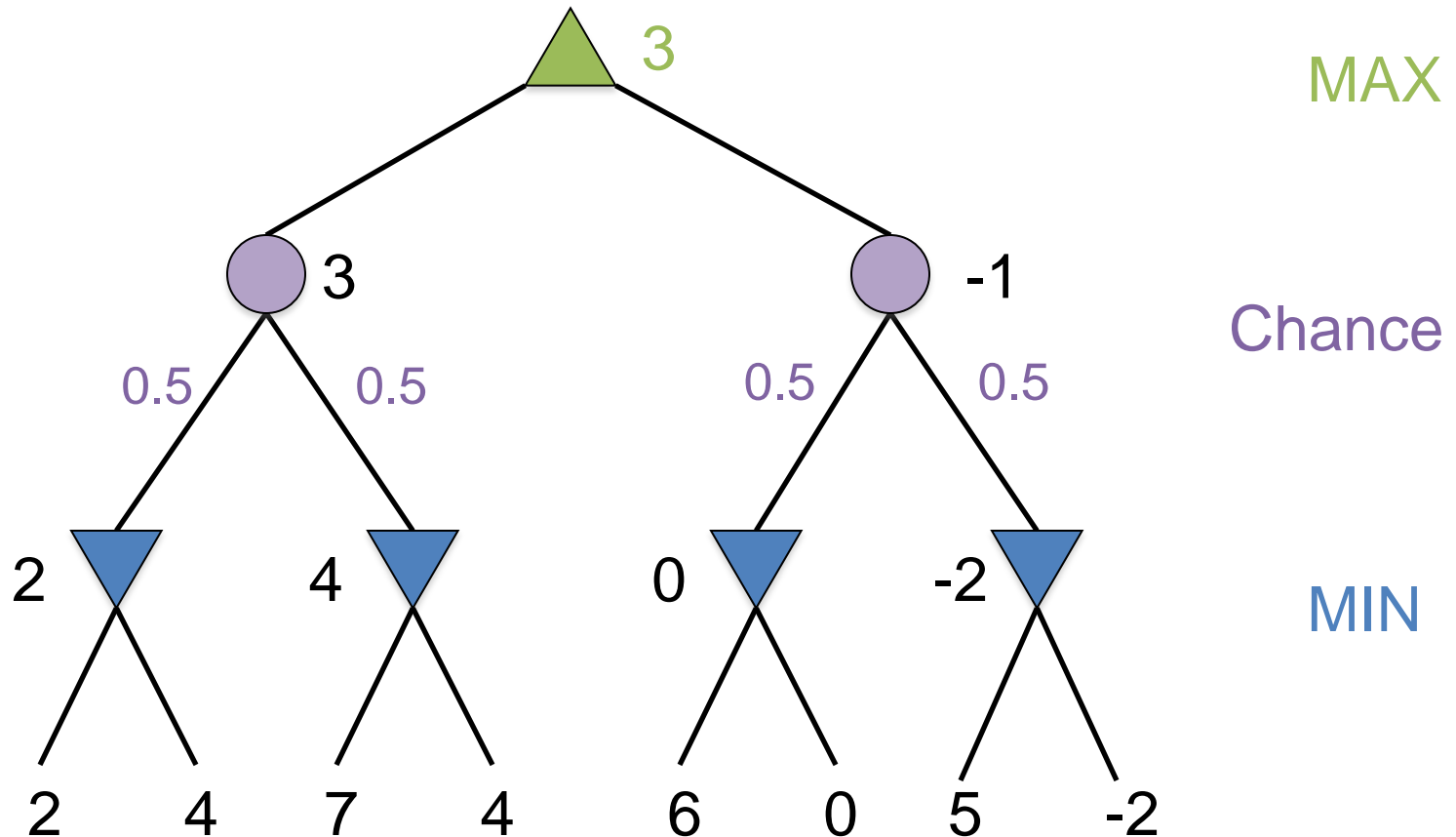
Nondeterministic games

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



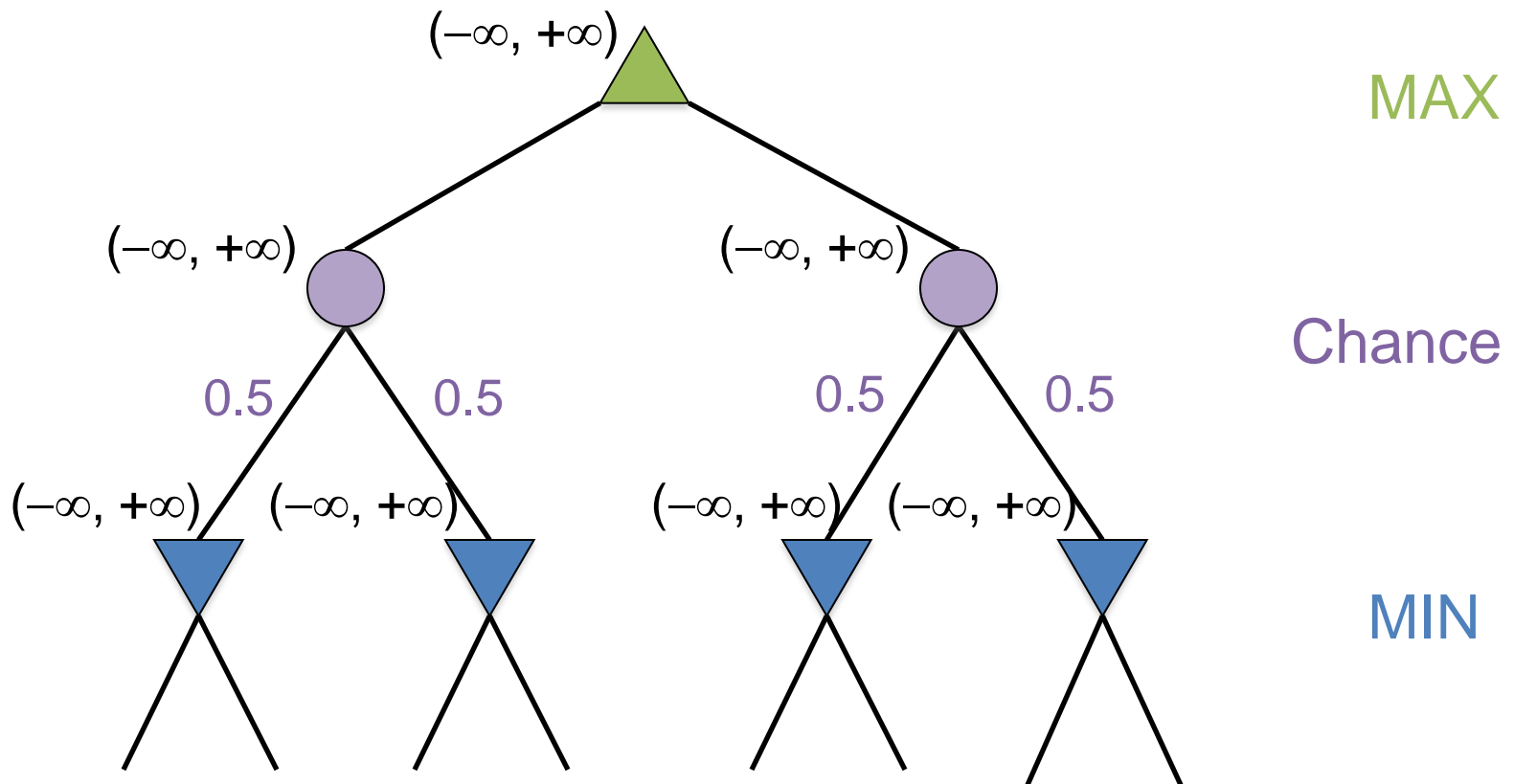
Pruning in nondeterministic games

- Can still apply a form of alpha-beta pruning



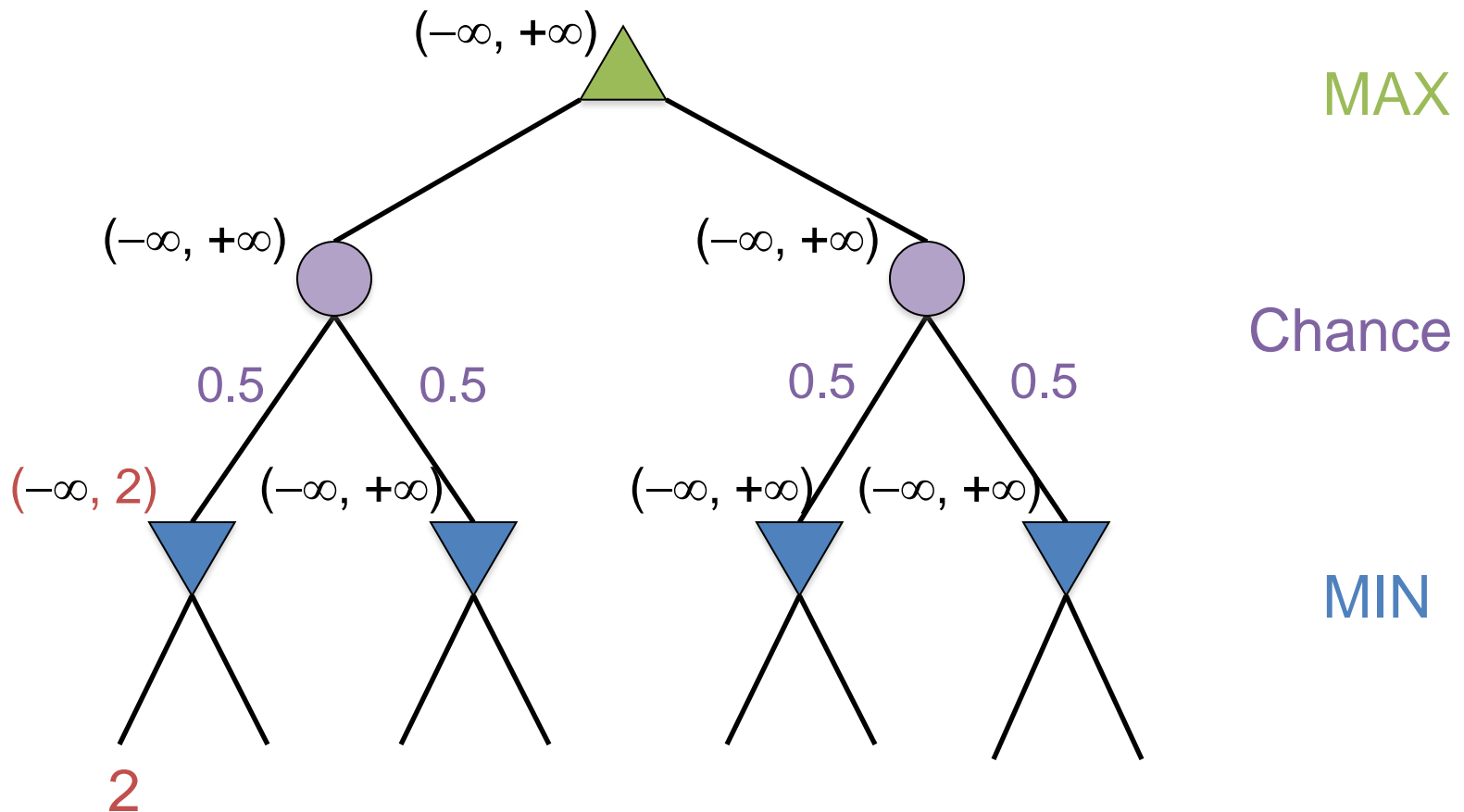
Pruning in nondeterministic games

- Can still apply a form of alpha-beta pruning



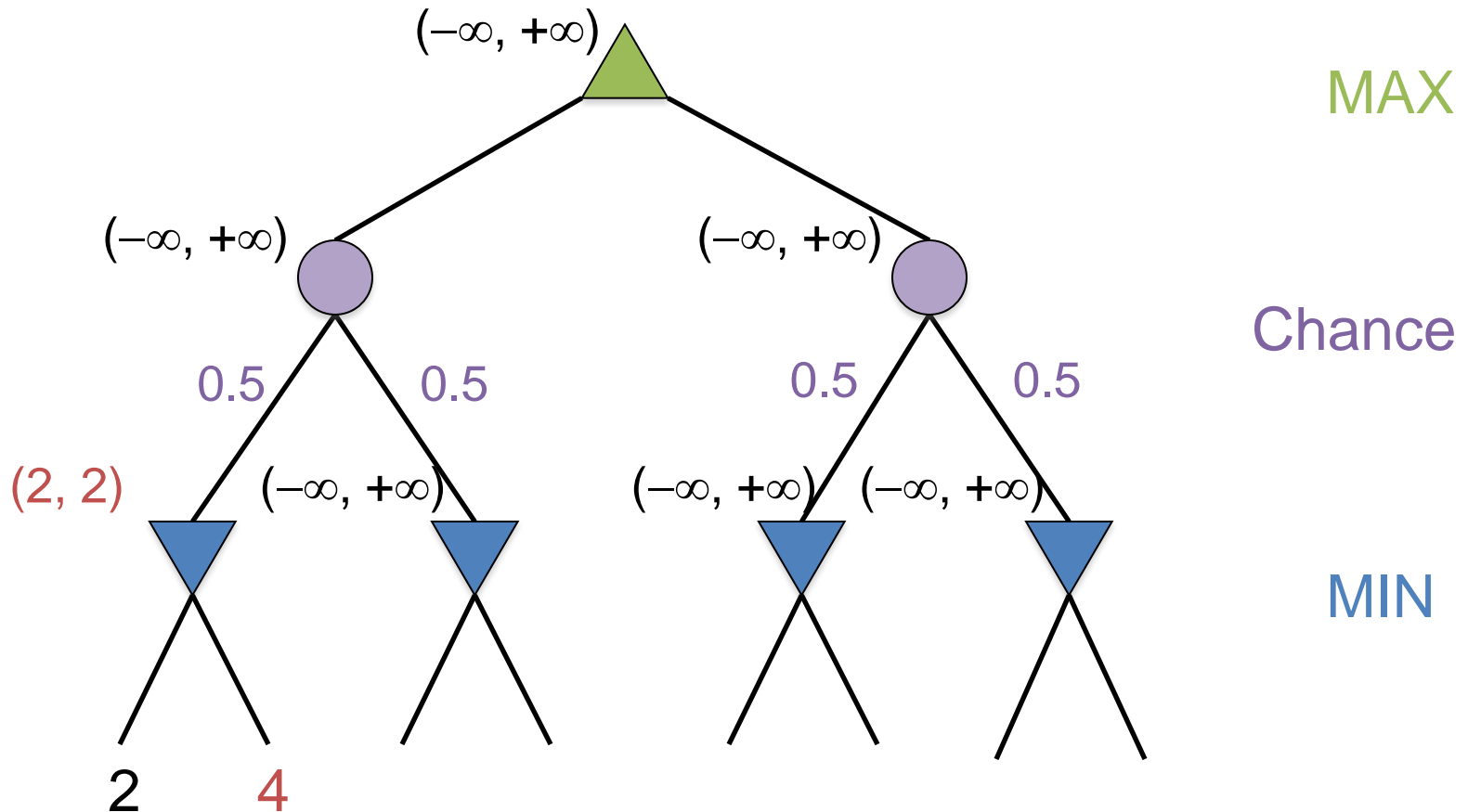
Pruning in nondeterministic games

- Can still apply a form of alpha-beta pruning



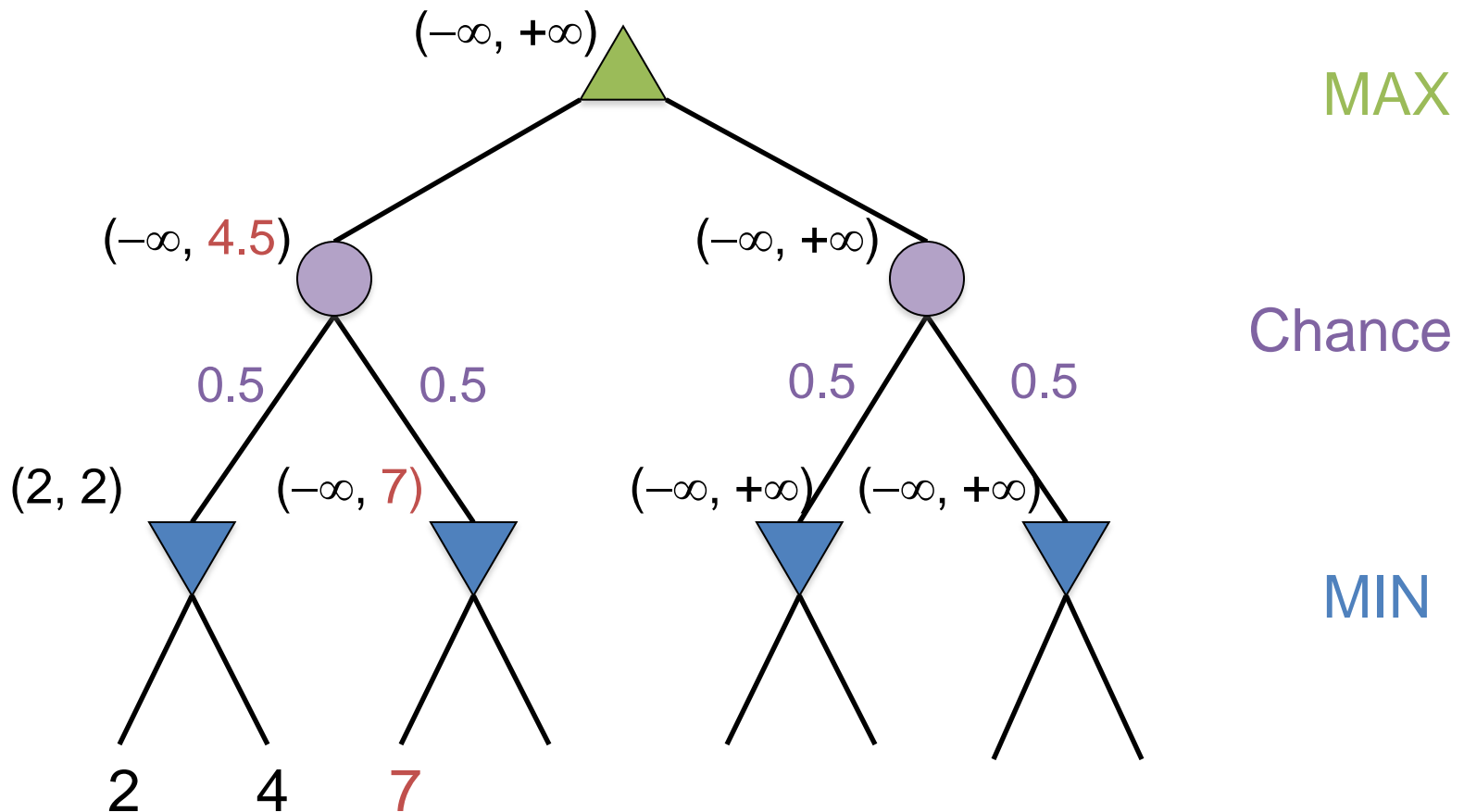
Pruning in nondeterministic games

- Can still apply a form of alpha-beta pruning



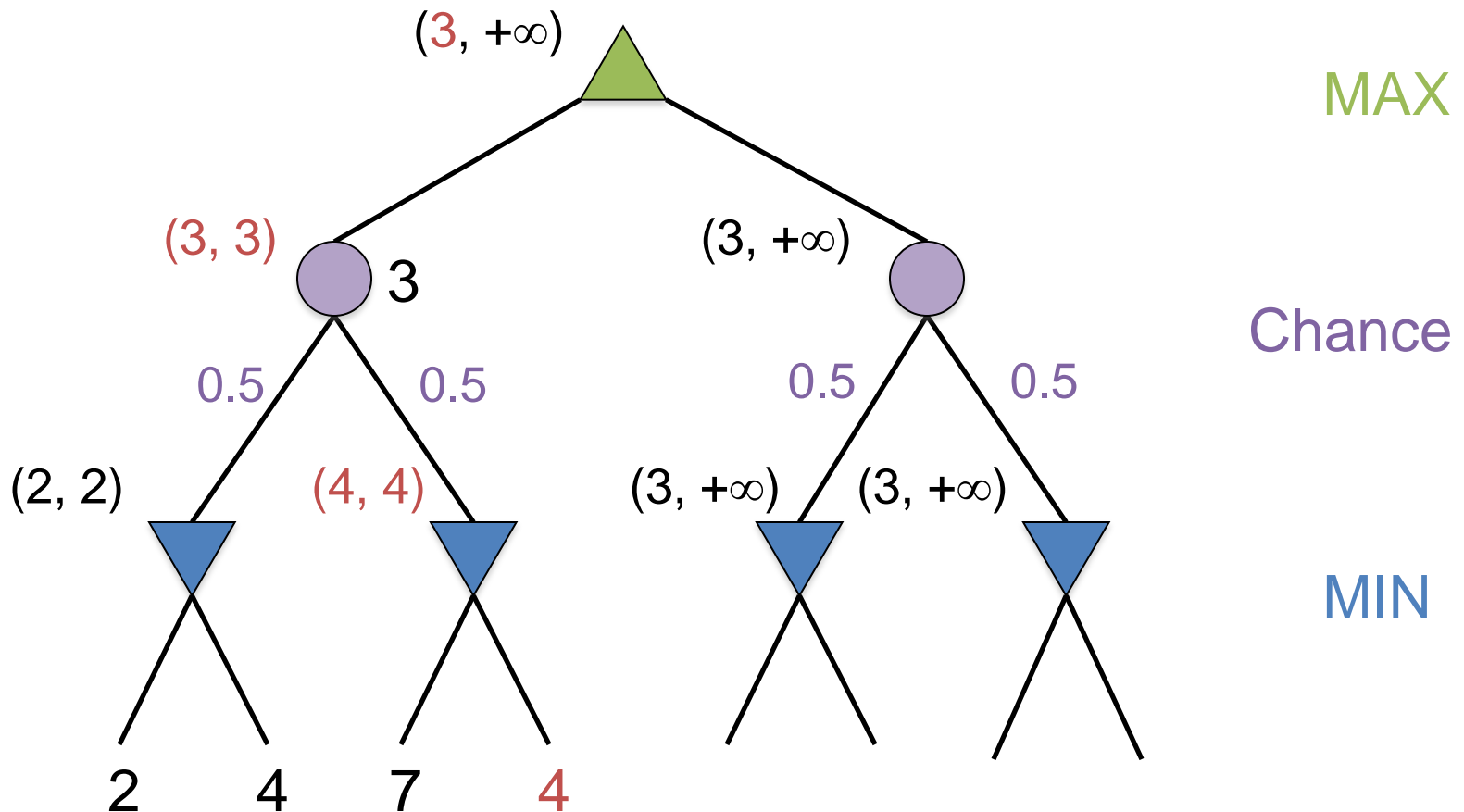
Pruning in nondeterministic games

- Can still apply a form of alpha-beta pruning



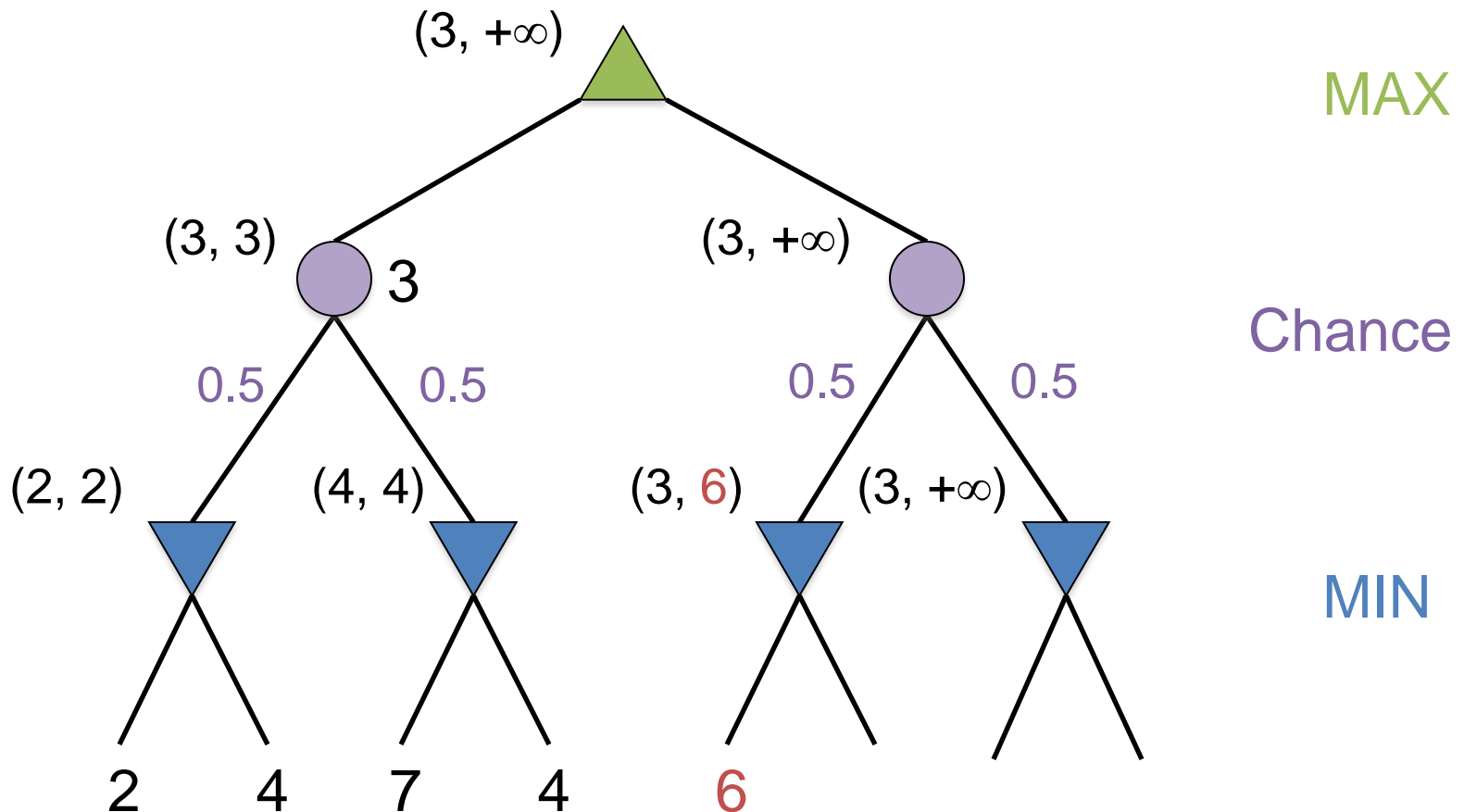
Pruning in nondeterministic games

- Can still apply a form of alpha-beta pruning



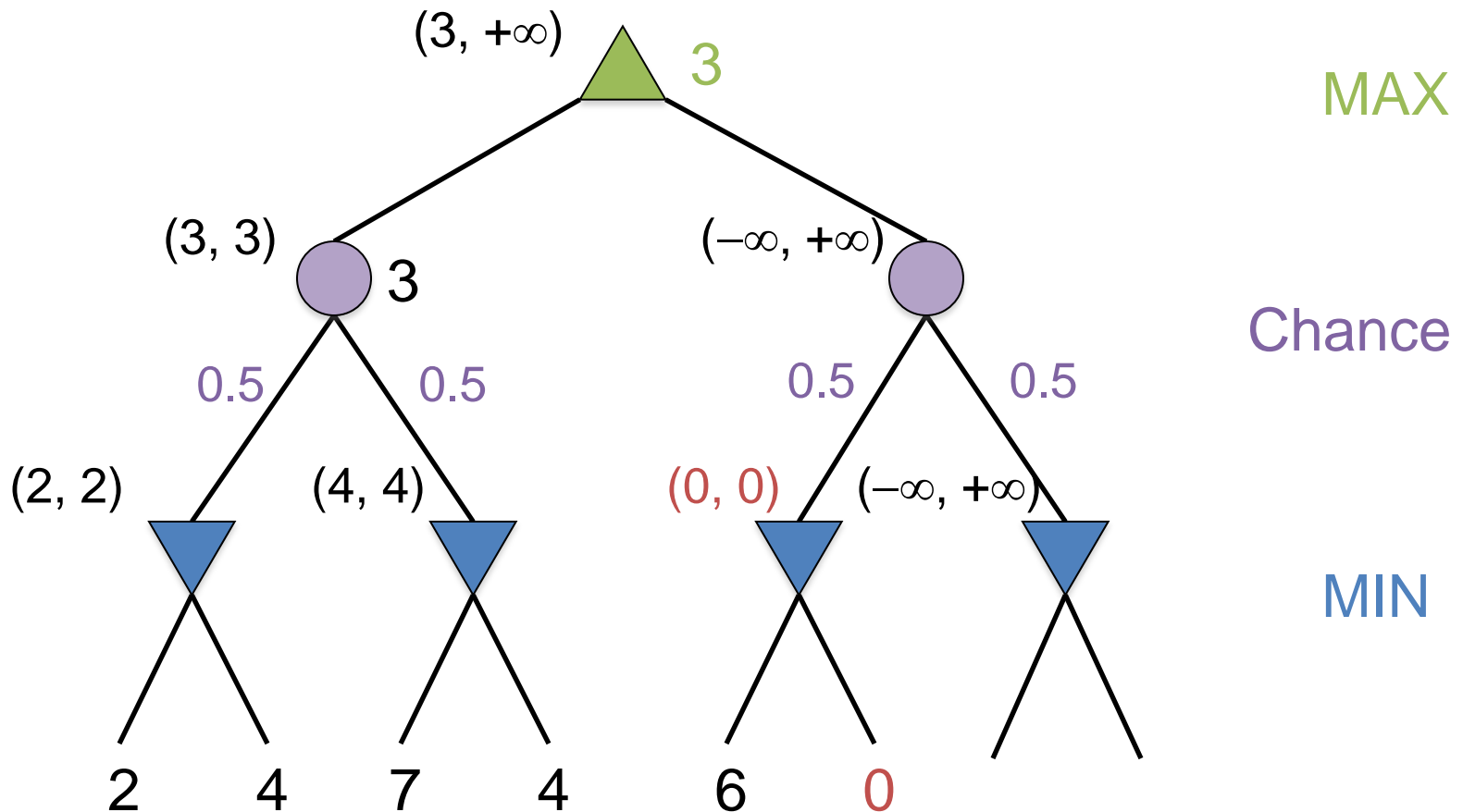
Pruning in nondeterministic games

- Can still apply a form of alpha-beta pruning



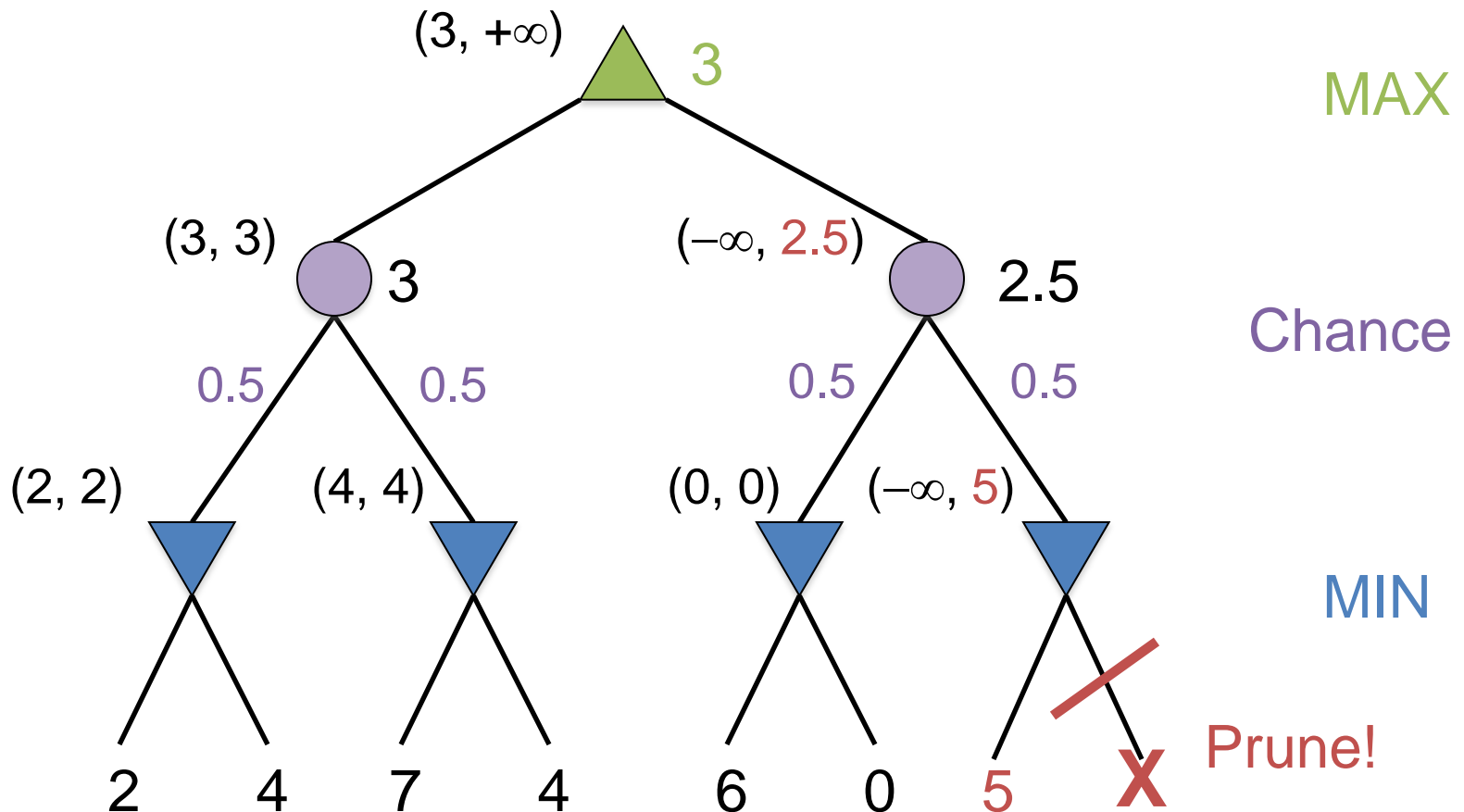
Pruning in nondeterministic games

- Can still apply a form of alpha-beta pruning



Pruning in nondeterministic games

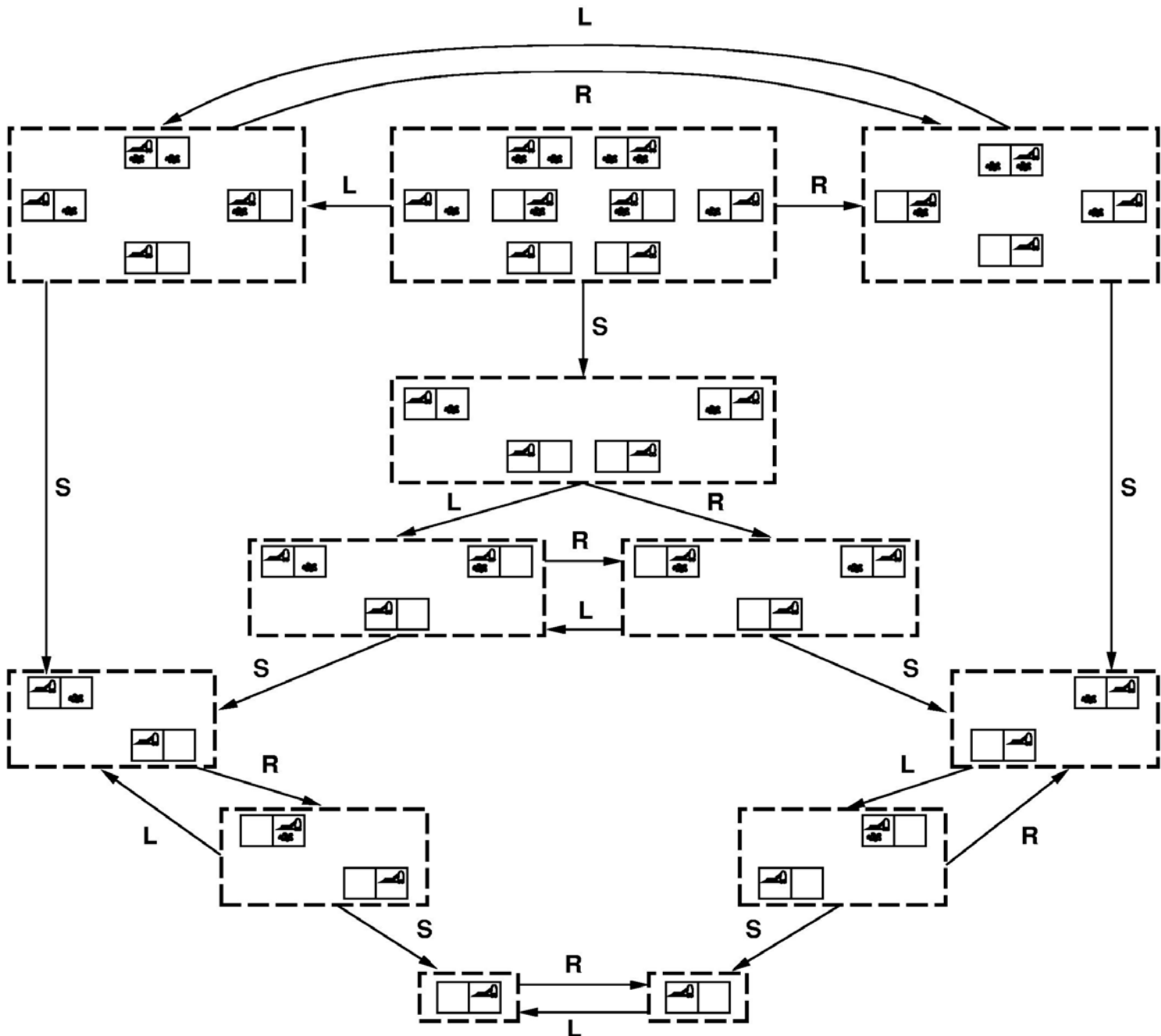
- Can still apply a form of alpha-beta pruning



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
- $\text{Actions}(b) = ??$ 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:
 - $\text{Result}(b,a) = \{ s' : s' = \text{Result}(s, a) \text{ and } s \text{ is a state in } b \}$
- $\text{Goaltest}(b) =$ 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
- Many more complications and possibilities!!
 - 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 9th 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

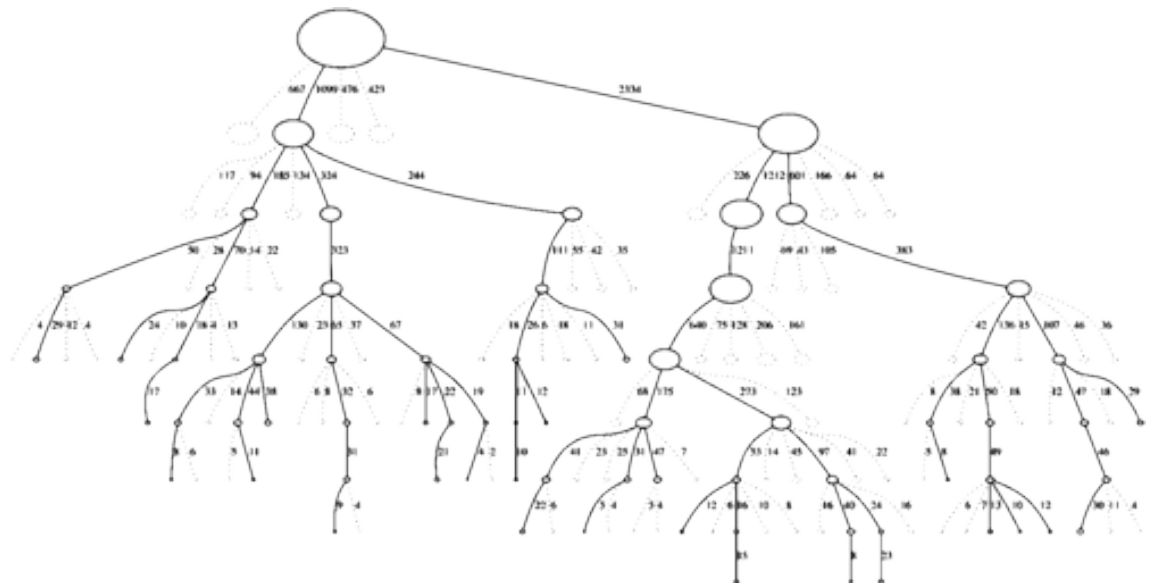
High branching factors

- 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...



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

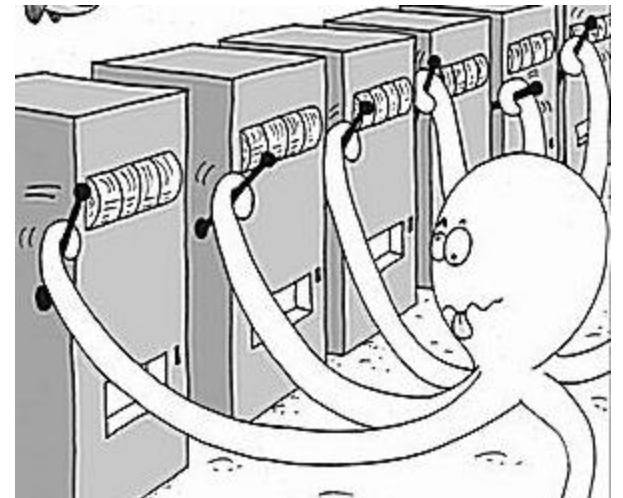
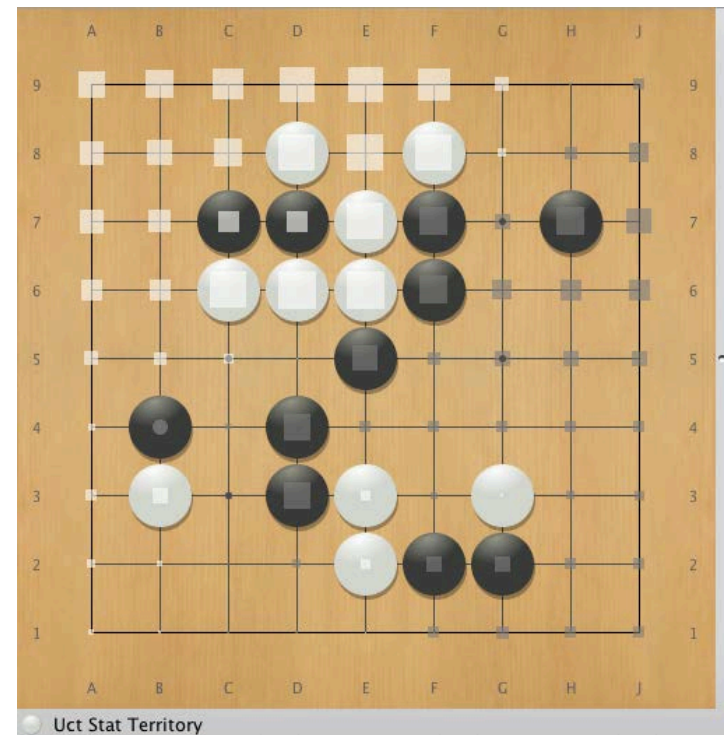
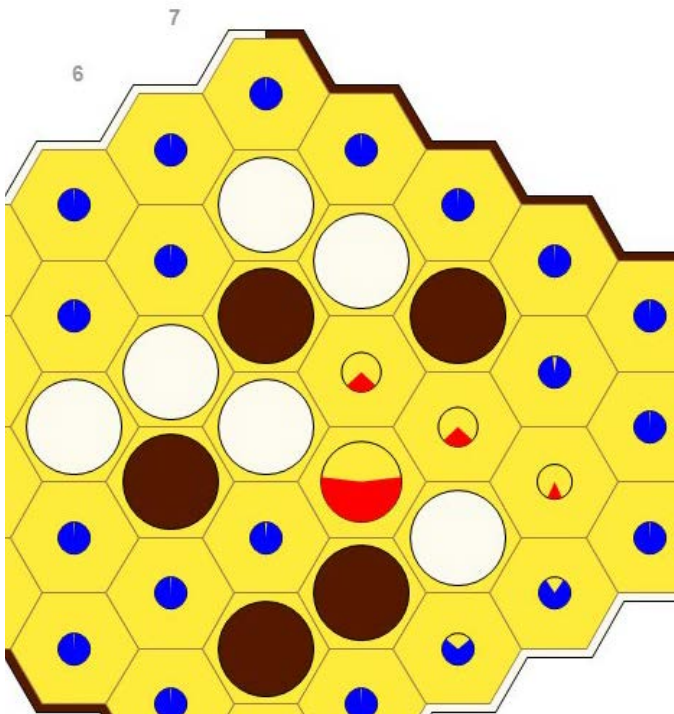


Image from Microsoft Research

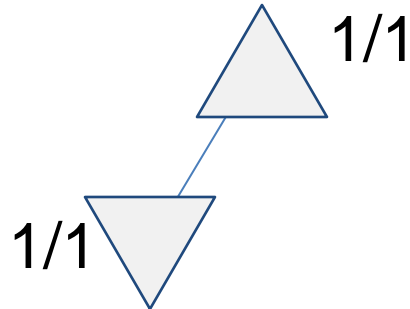
Visualizing MCTS

- 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

MAB strategy



Default / random strategy



Terminal state

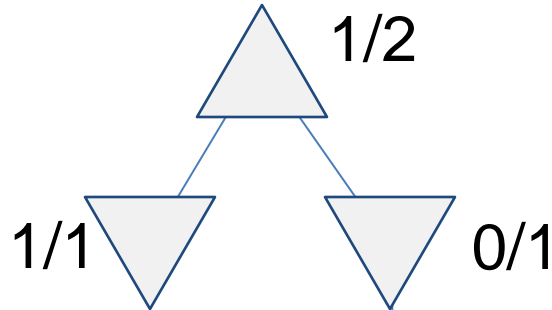
Score consists of
(1) % wins
(2) # times tried
(3) # of steps total

UCT:

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

MCTS

MAB strategy



Default / random strategy



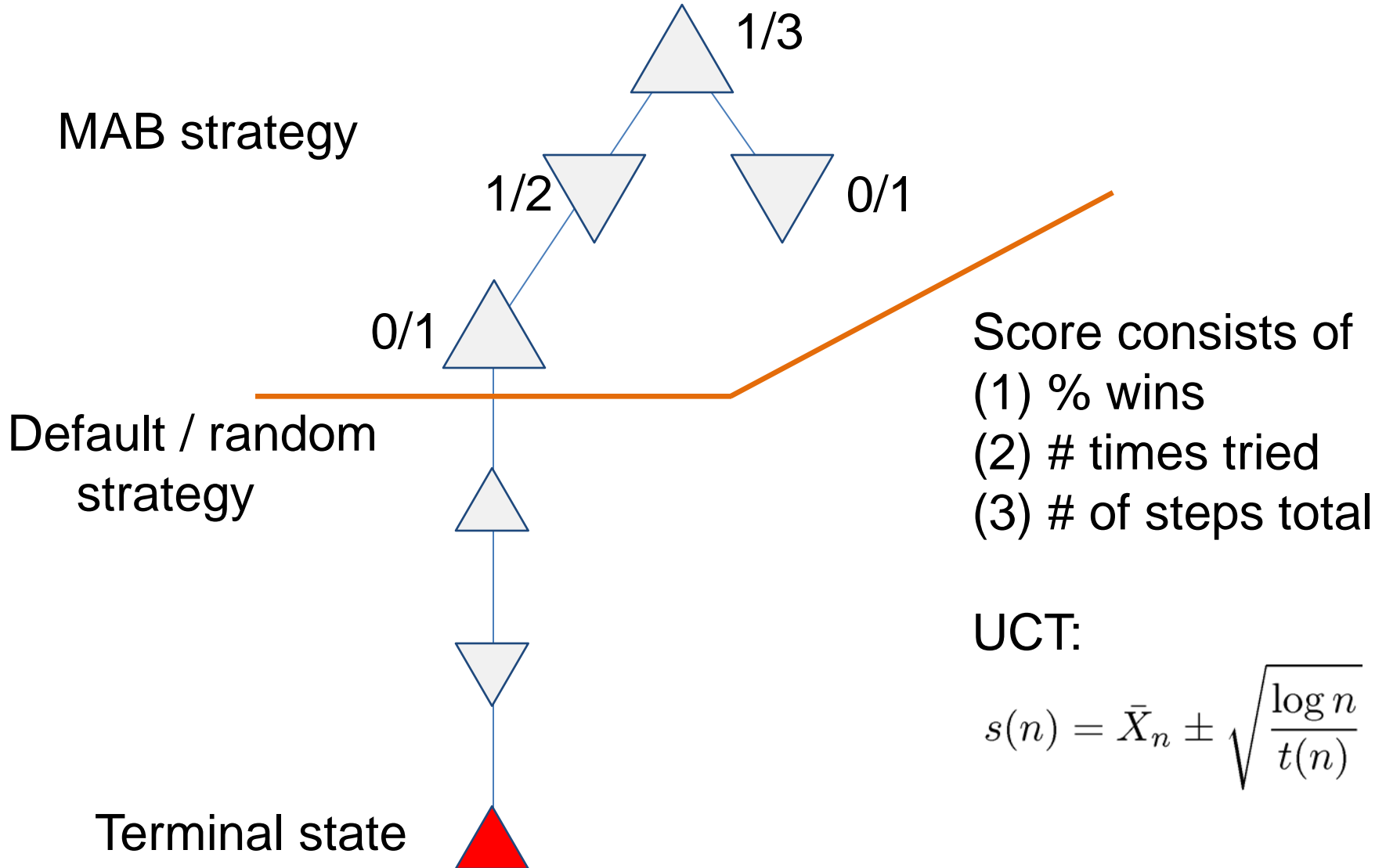
Terminal state

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



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.