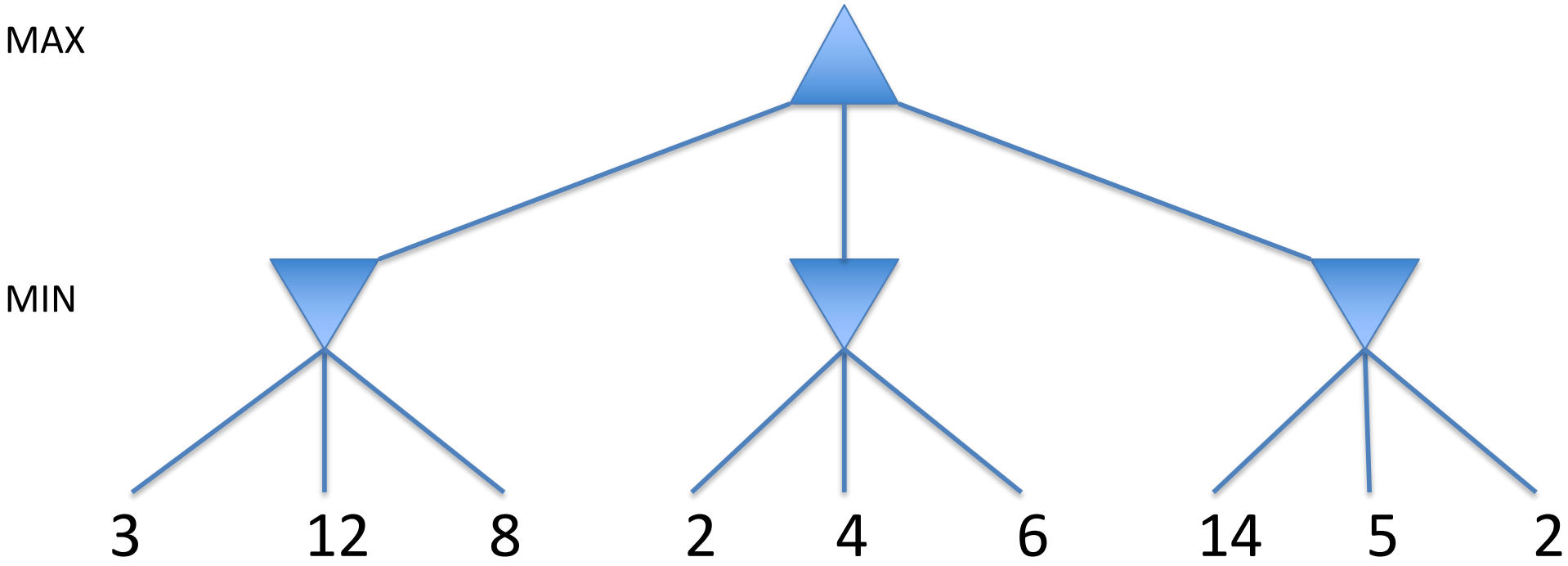
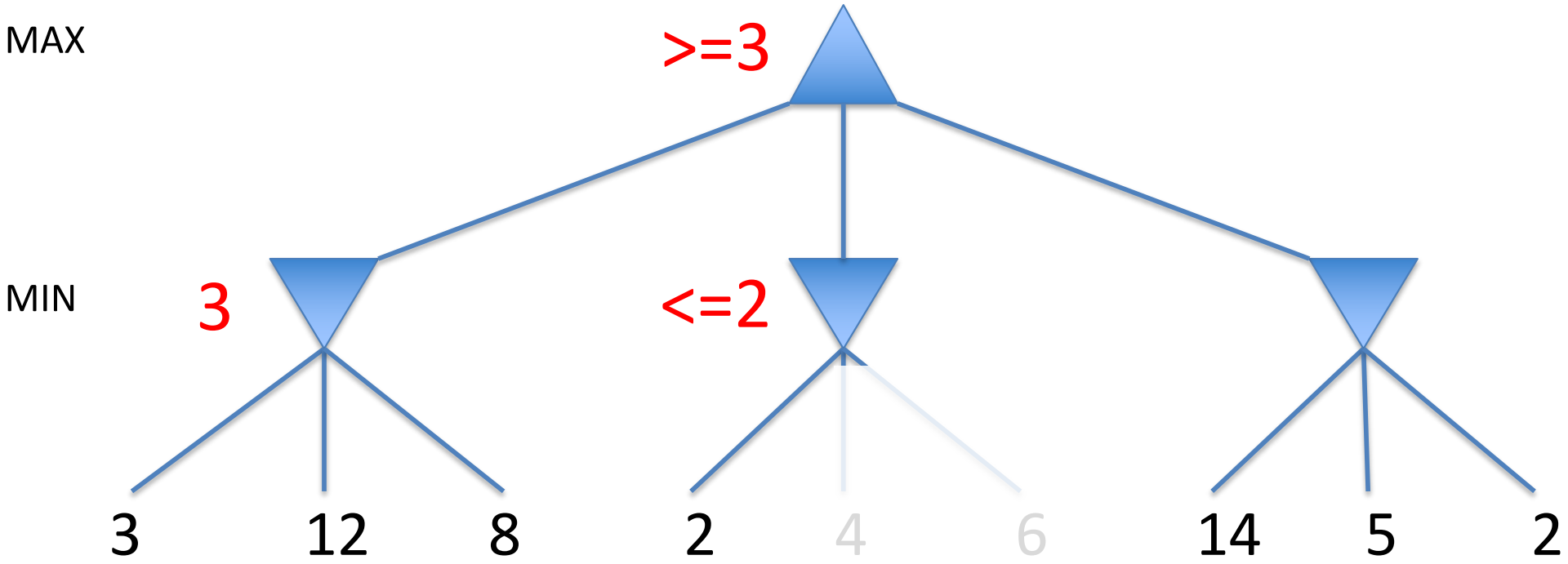
- Problem: minimax takes too long.
- Solution: improve algorithm to ignore parts of the tree that will definitely not be used (assuming both players play optimally).


MAX
MIN
3
12
8
2
4
6
14
5
2


MAX
>=3
MIN
3
<=2
3
12
8
2
4
6
14
5
2

- Idea: for each node, keep track of the range of possible values that minimax could produce for that node.

- If we ever find ourselves at a node that we know will never be selected during (optimal) game play, we can "prune" it (end the recursion on this part of the tree).

- Enhanced version of minimax is called ***minimax with alpha-beta pruning.***

Alpha-beta pruning

- Recall that minimax is a variant of depth-first search. During the algorithm, we will only consider nodes along the path from the root node to the current node.
- At each node in the search, we will maintain two variables:
  - alpha (α) = highest numeric value we've found so far on this path (best move for MAX)
  - beta (β) = lowest numeric value we've found so far on this path (best choice for MIN)

Alpha-beta pruning

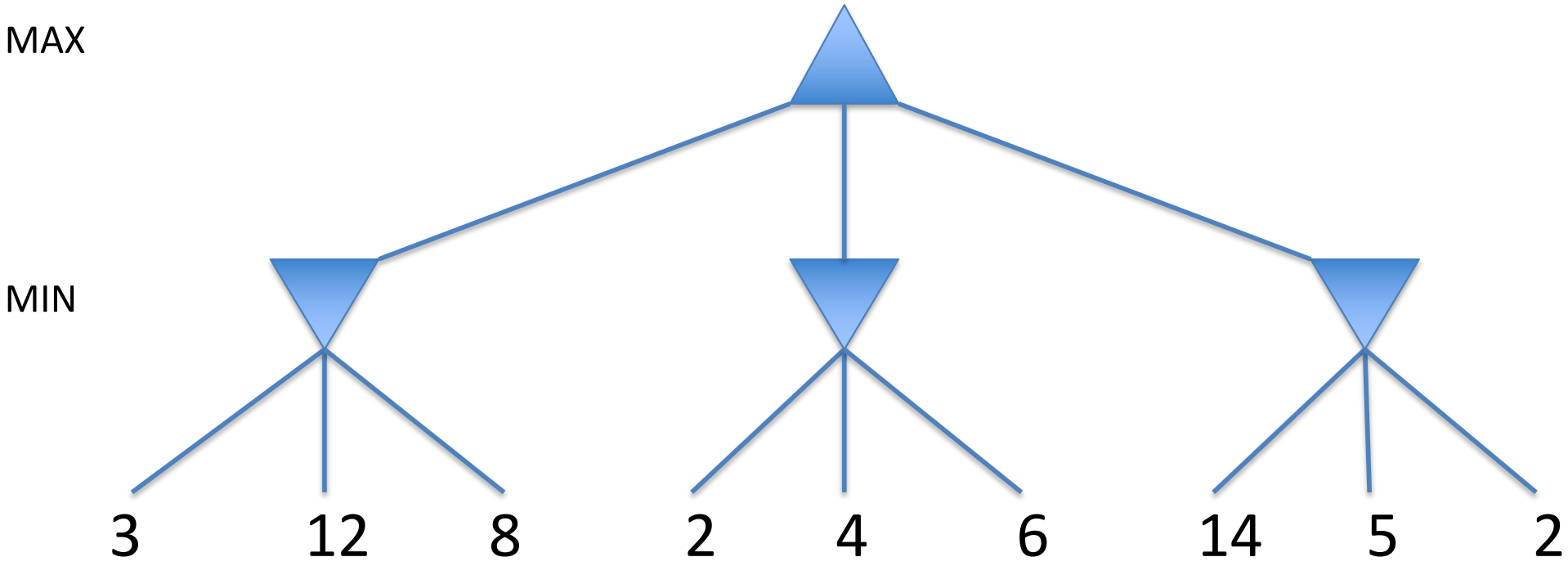
- Alpha and beta are inherited from parent nodes as we recursively descend the tree.

- If at a MAX node, we see a child node that has a value >= than beta, **short-circuit**.
- If at a MIN node, we see a child node that has a value <= than alpha, **short-circuit**.

```
function ALPHA-BETA-SEARCH(game, state) returns an action
    player ← game.TO-MOVE(state)
    value, move ← MAX-VALUE(game, state, −∞, +∞)
    return move

function MAX-VALUE(game, state, α, β) returns a (utility, move) pair
    if game.IS-TERMINAL(state) then return game.UTILITY(state, player), null
    v ← −∞
    for each a in game.ACTIONS(state) do
        v2, a2 ← MIN-VALUE(game, game.RESULT(state, a), α, β)
        if v2 > v then
            v, move ← v2, a
            α ← MAX(α, v)
        if v ≥ β then return v, move
    return v, move

function MIN-VALUE(game, state, α, β) returns a (utility, move) pair
    if game.IS-TERMINAL(state) then return game.UTILITY(state, player), null
    v ← +∞
    for each a in game.ACTIONS(state) do
        v2, a2 ← MAX-VALUE(game, game.RESULT(state, a), α, β)
        if v2 < v then
            v, move ← v2, a
            β ← MIN(β, v)
        if v ≤ α then return v, move
    return v, move
```


MAX
MIN
3
12
8
2
4
6
14
5
2

- The results of alpha-beta depend on the order in which moves are considered among the children of a node.
- If possible, consider better moves first!

```
function ALPHA-BETA-SEARCH(game, state) returns an action
    player ← game.TO-MOVE(state)
    value, move ← MAX-VALUE(game, state, −∞, +∞)
    return move

function MAX-VALUE(game, state, α, β) returns a (utility, move) pair
    if game.IS-TERMINAL(state) then return game.UTILITY(state, player), null
    v ← −∞
    for each a in game.ACTIONS(state) do
        v2, a2 ← MIN-VALUE(game, game.RESULT(state, a), α, β)
        if v2 > v then
            v, move ← v2, a
            α ← MAX(α, v)
        if v ≥ β then return v, move
    return v, move

function MIN-VALUE(game, state, α, β) returns a (utility, move) pair
    if game.IS-TERMINAL(state) then return game.UTILITY(state, player), null
    v ← +∞
    for each a in game.ACTIONS(state) do
        v2, a2 ← MAX-VALUE(game, game.RESULT(state, a), α, β)
        if v2 < v then
            v, move ← v2, a
            β ← MIN(β, v)
        if v ≤ α then return v, move
    return v, move
```