EECS 391
Intro to AI

Optimal Game Play, Minimax, α-β Pruning

L6 Tue Sep 19, 2017

Key concepts today

- types of games
- game trees - for representing the state space of a game
- minimax algorithm - for optimal game play
- alpha-beta pruning - to reduce the size of the search space

Types of games

We'll focus on these.

	deterministic	chance
perfect information	chess checkers tic tac toe go	backgammon monopoly
imperfect information	battleship mastermind	bridge poker scrabble

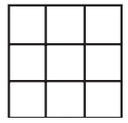
How do we formalize game playing?

Game playing is very much like a search problem:

- **states**:
  - initial state, board state, next player
- **result**:
  - list of legal moves: move/state pairs or transition model
- **terminal test**:
  - Is the game over?
- **utility function**:
  - numerical value of terminal states
  - +1, -1, or 0 for won, loose or draw
  - "zero-sum" game: one player's loss is another's gain
    (technically: total payoff of all players is constant)
  - Could be a score (or payoff), e.g. backgammon

Representing game play with game trees

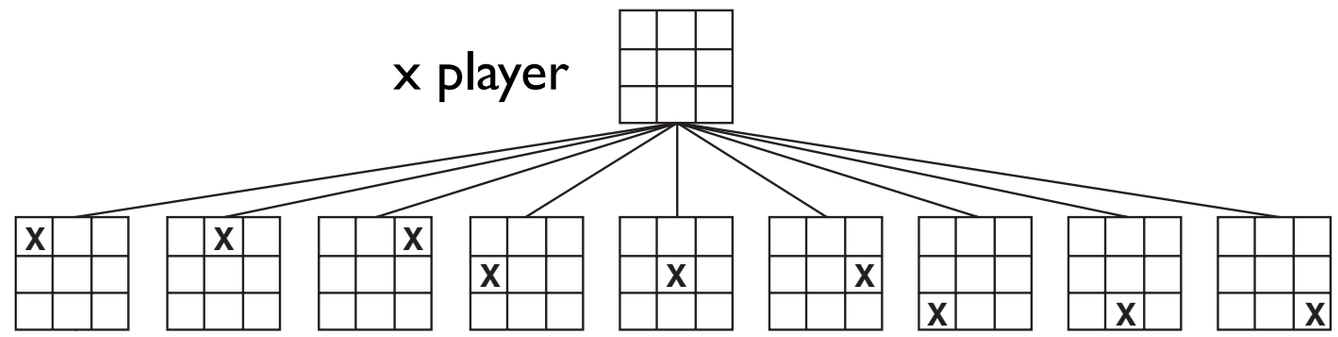
- Top node is initial state
- each level lists the available moves (or results) from each game state
- each player tries to maximize their utility (or win the game)

x player 

Representing game play with game trees

- Top node is initial state
- each level lists the available moves (or results) from each game state
- each player tries to maximize their utility (or win the game)

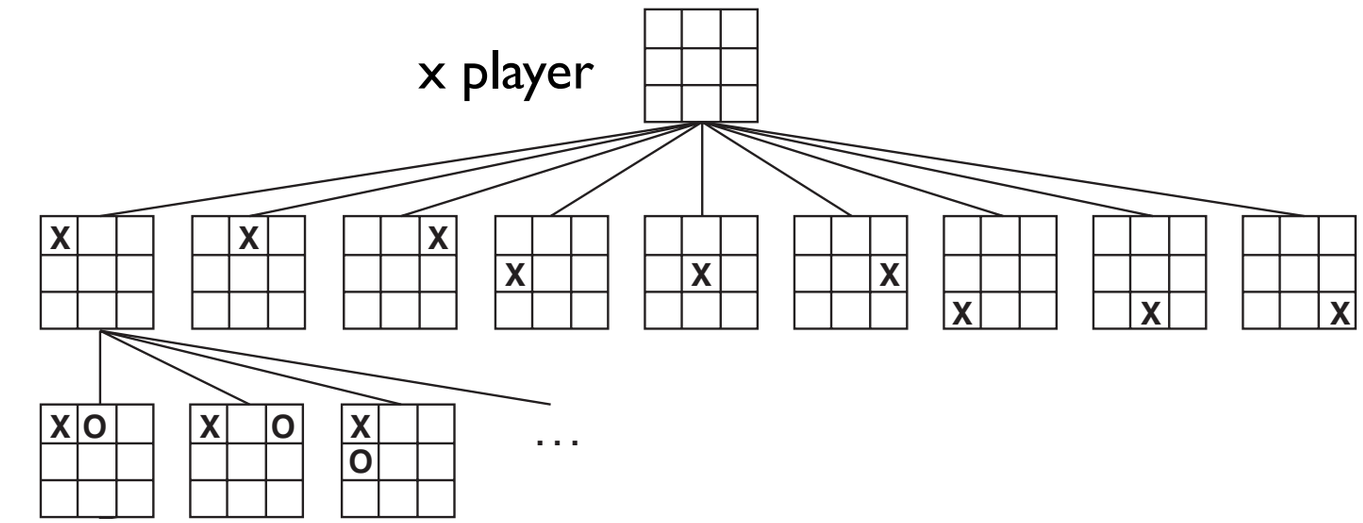
Each level of the tree (i.e. a move by one player) is called a "ply".



Representing game play with game trees

- Top node is initial state
- each level lists the available moves (or results) from each game state
- each player tries to maximize their utility (or win the game)

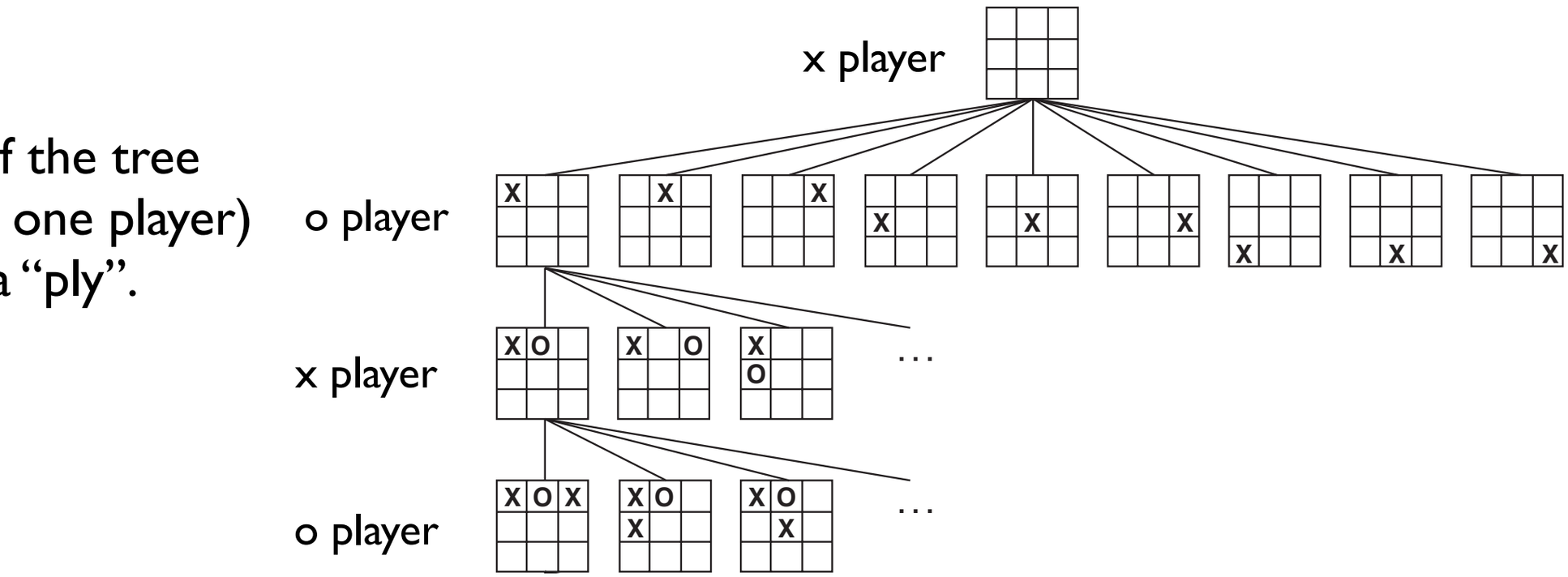
Each level of the tree (i.e. a move by one player) is called a "ply".



Representing game play with game trees

- Top node is initial state
- each level lists the available moves (or results) from each game state
- each player tries to maximize their utility (or win the game)

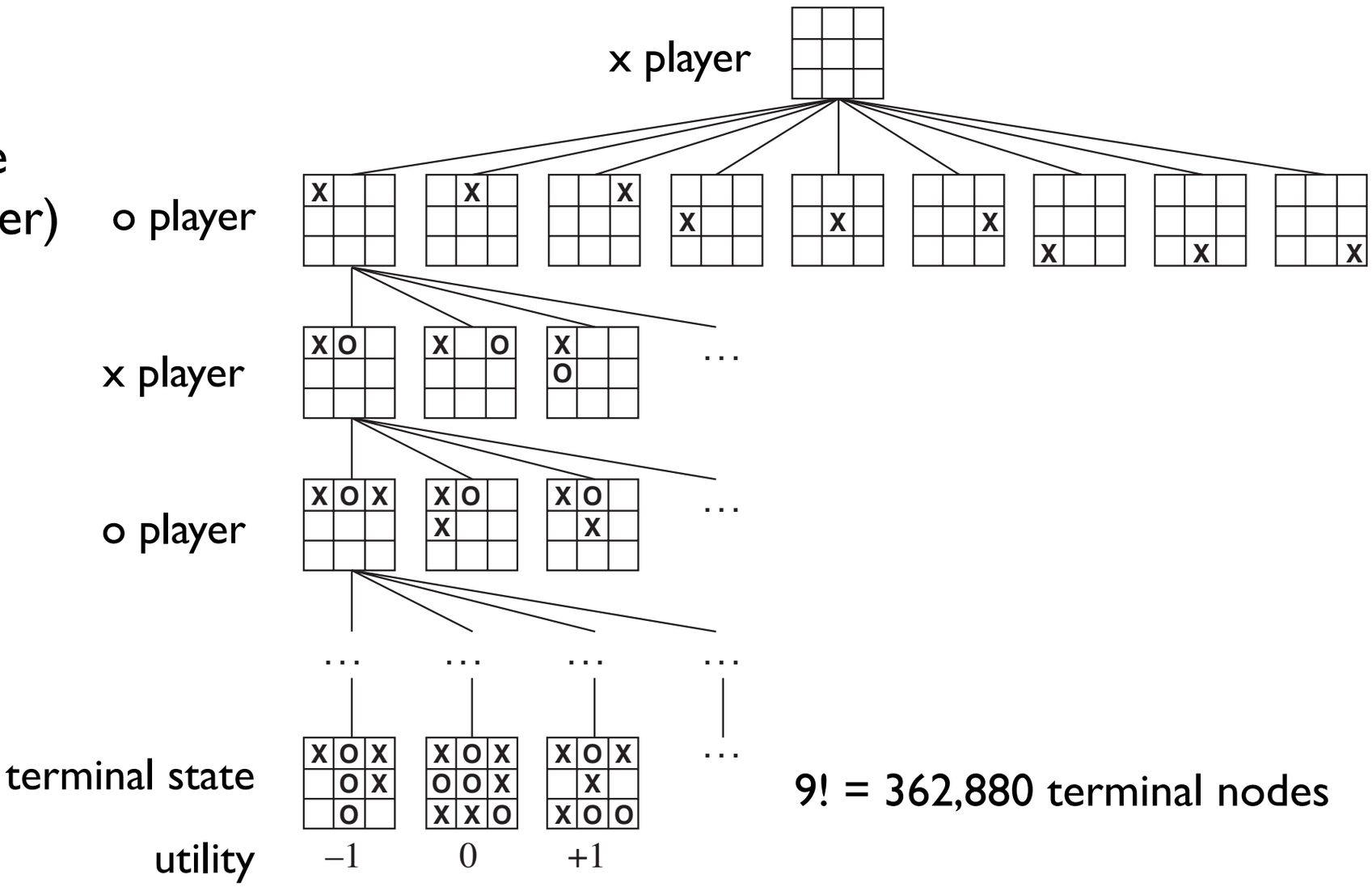
Each level of the tree (i.e. a move by one player) is called a "ply".



Representing game play with game trees

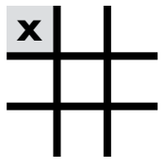
- Top node is initial state
- each level lists the available moves (or results) from each game state
- each player tries to maximize their utility (or win the game)

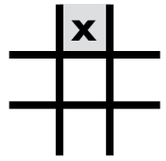
Each level of the tree (i.e. a move by one player) is called a "ply".



9! = 362,880 terminal nodes

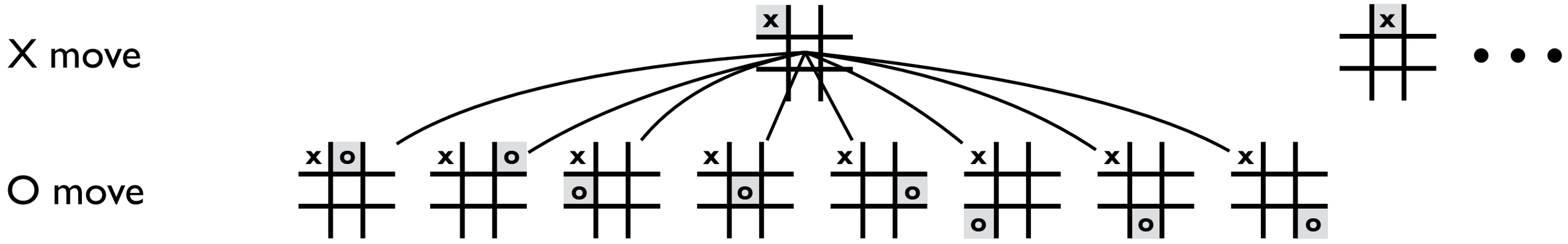
X move



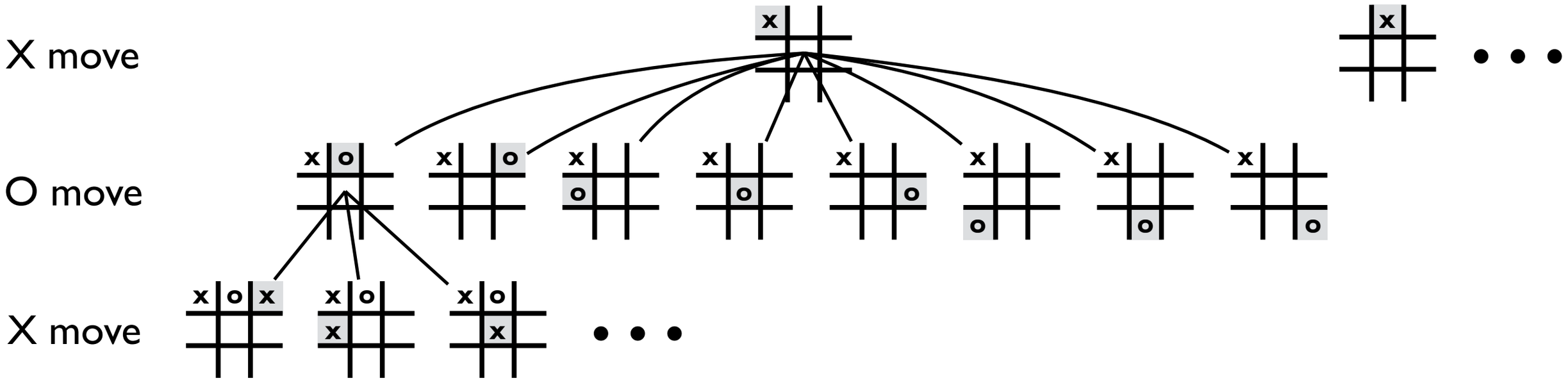


• • •

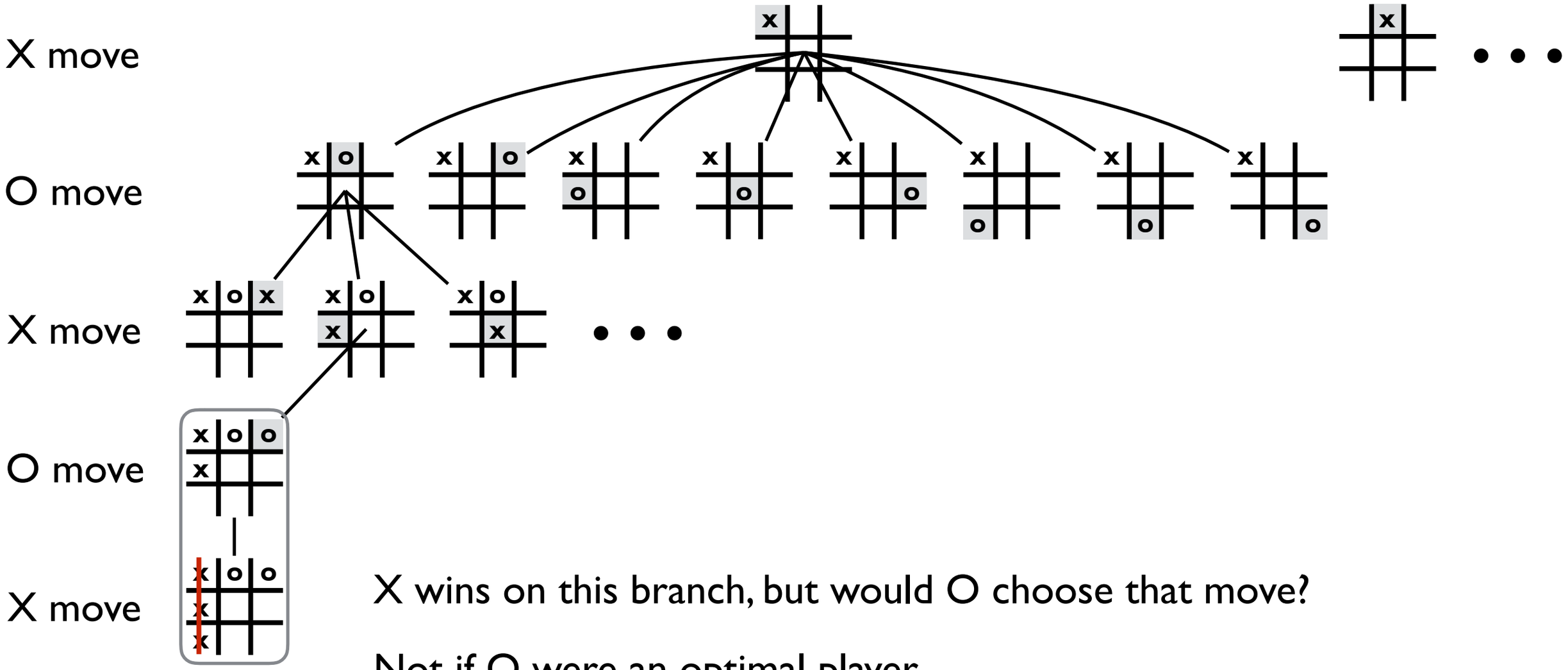
Initial moves start different game trees.
Each board position is a state in the search space.



The X player considers moves the O player might do from a given position.



Then looks deeper into the tree from each potential move by the O player.

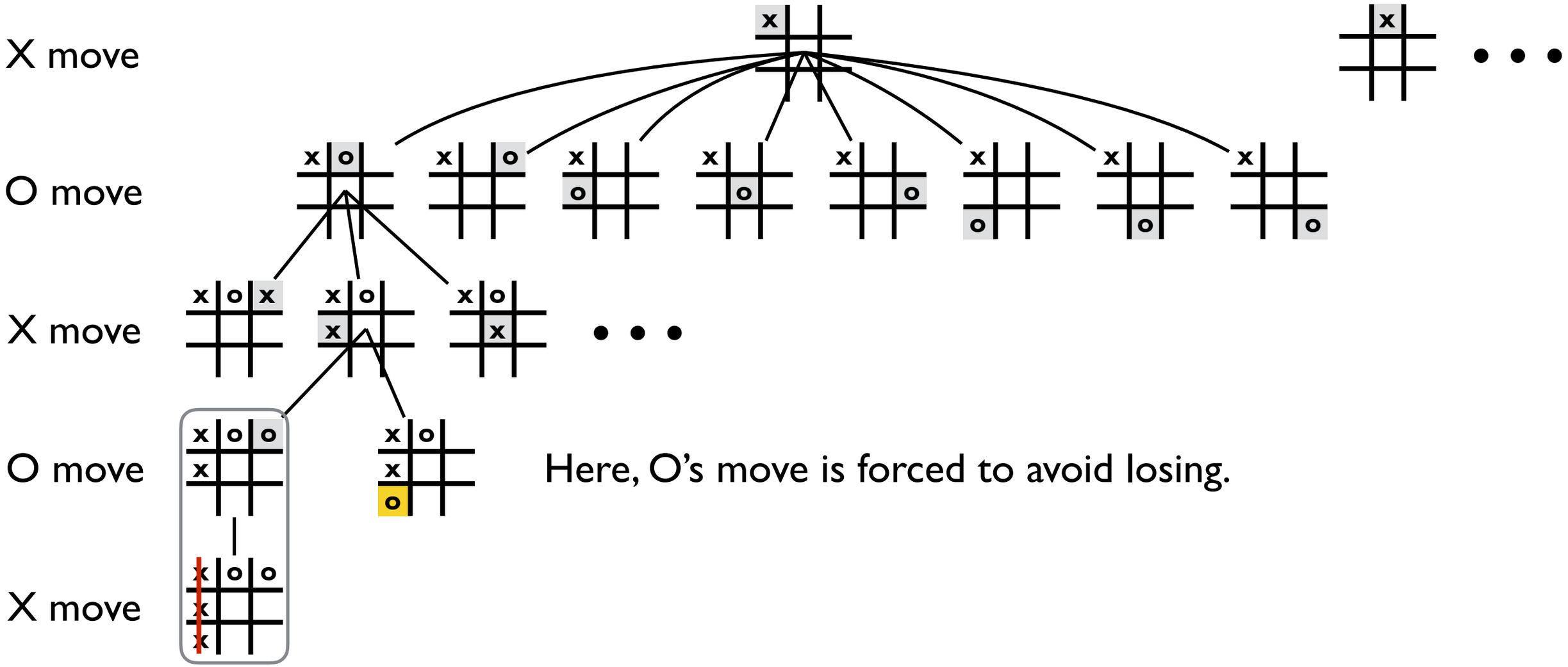


X wins on this branch, but would O choose that move?

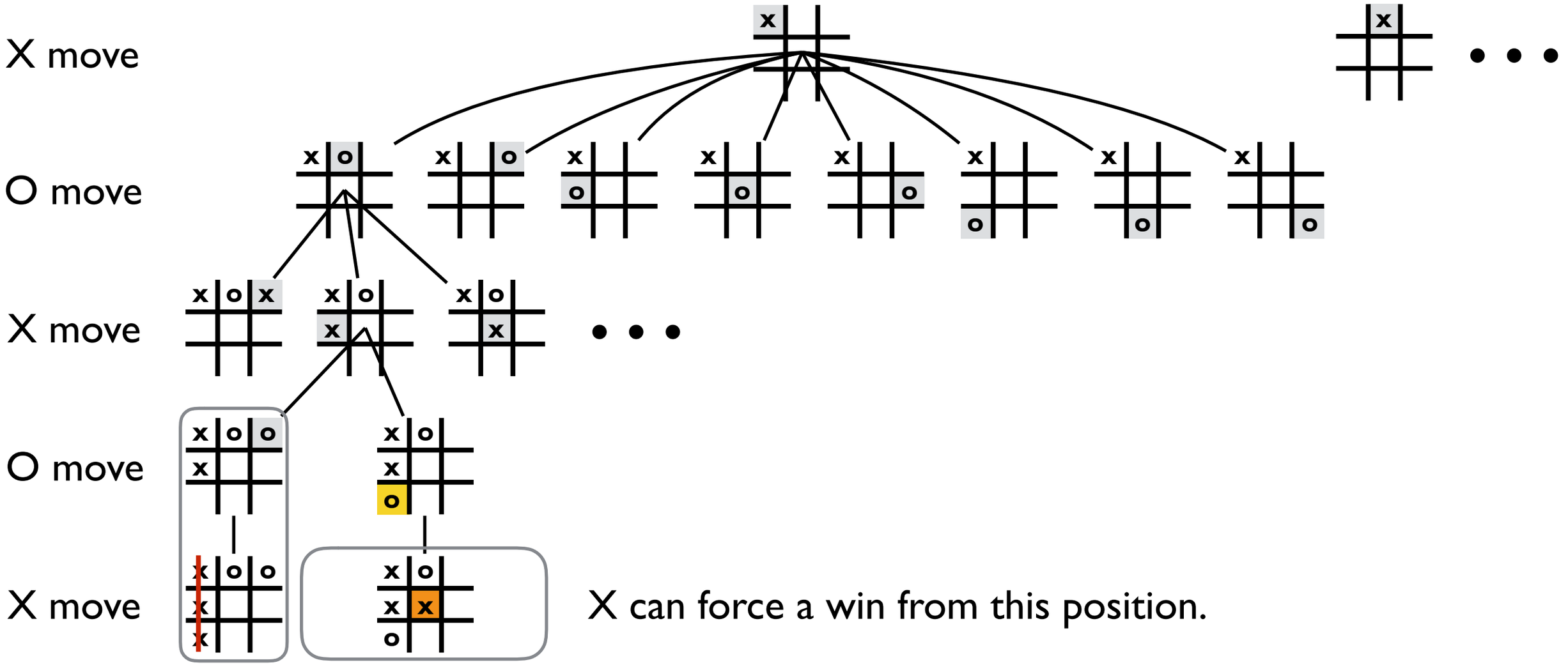
Not if O were an optimal player.

X must assume that O will play optimally.

(Unless there is evidence to the contrary,
but then X has to have a model to predict
how O will play.)

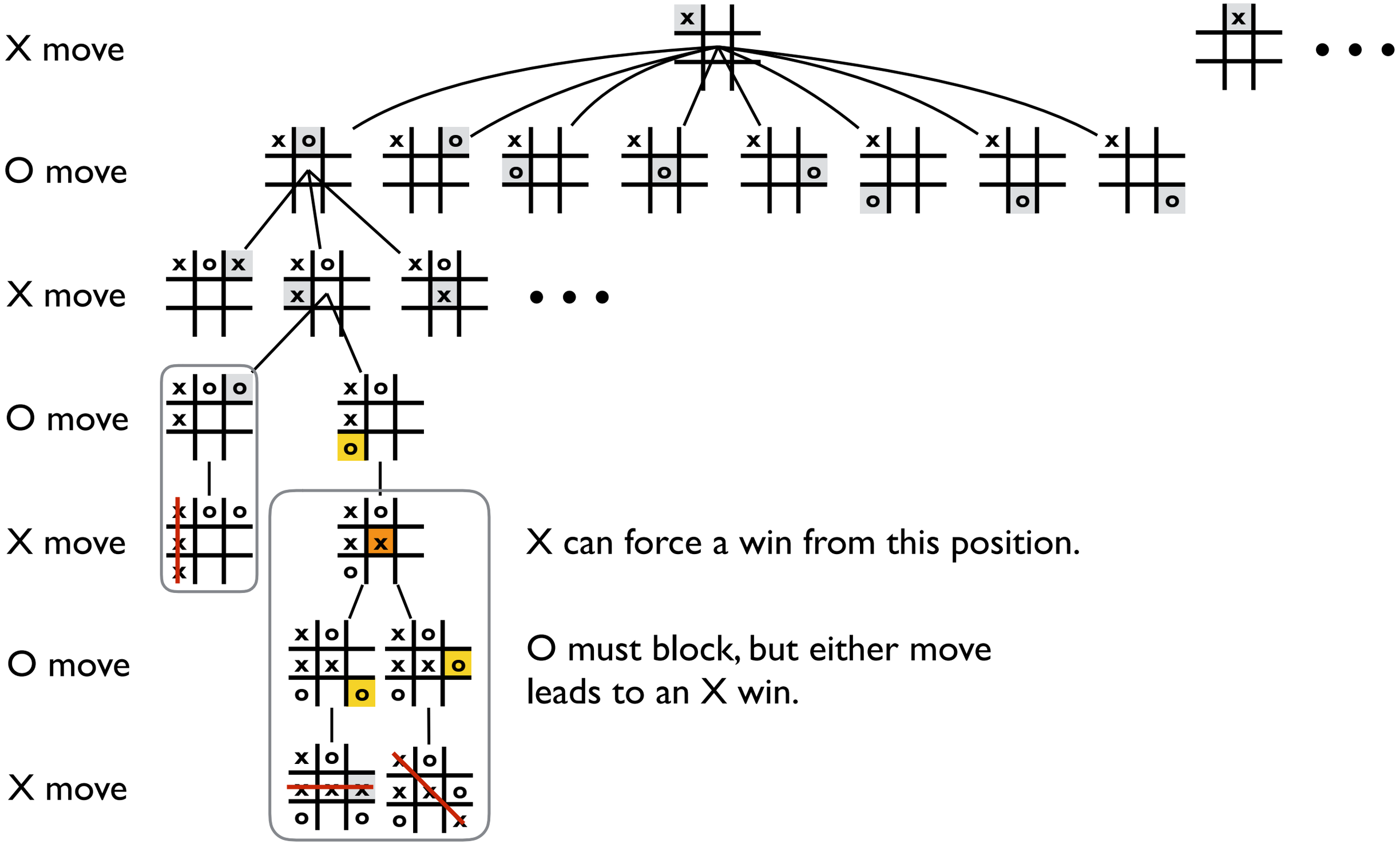


Here, O's move is forced to avoid losing.



X can force a win from this position.

O must block, but either move leads to an X win.



X can force a win from this position.

O must block, but either move leads to an X win.

X move

O move

X move

O move

X move

O move

X move

O move

If X doesn't play optimally,
O can force a win.

X move

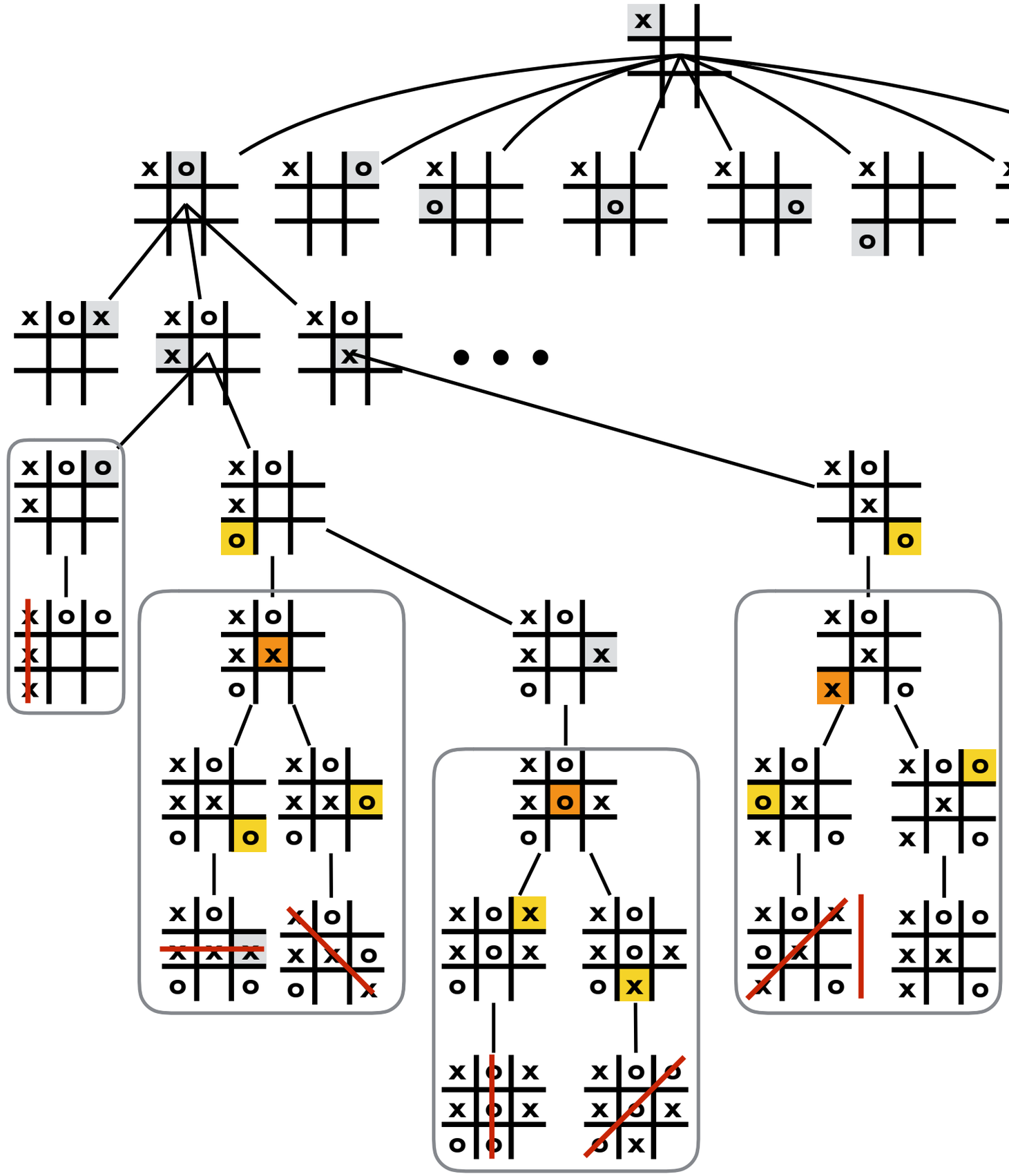
O move

X move

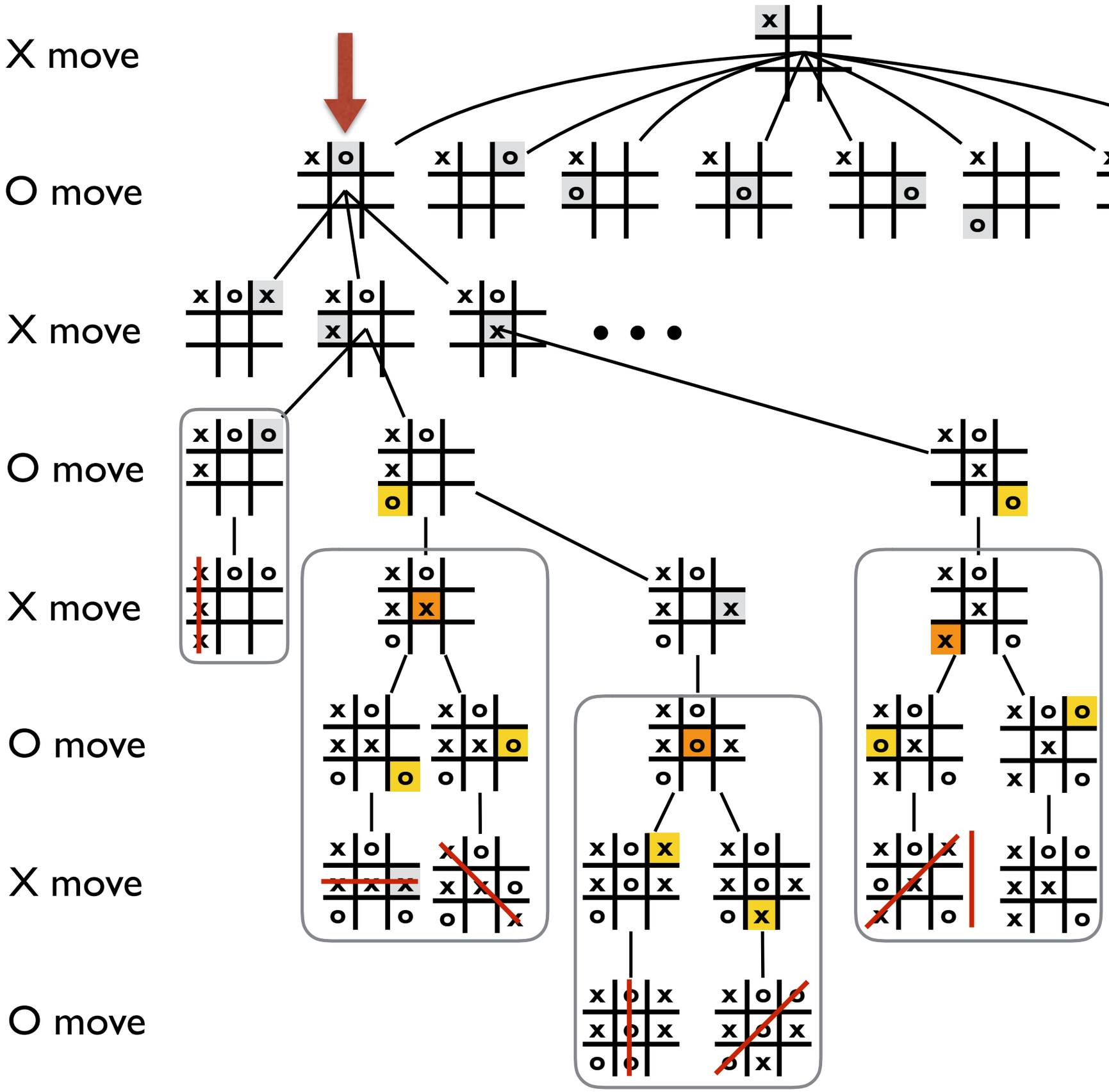
O move

X move

O move



Another position from which X can force a win.



Another position from which X can force a win.

But, these positions all assume O played (1,2), which isn't optimal.

Why?

Because X can then force a win.

X move

O move

X move

O move

X move

O move

X move

This O move forces a draw.

A game tree for pennies

- Pennies game:
  - stack of pennies
  - each player divides one of the stacks into two *unequal* stacks
  - game ends when every stack contains one or two pennies
  - first player who cannot play looses
- How do you represent the game state?
- How do we choose the best moves?
- What's the optimal strategy for searching the tree?

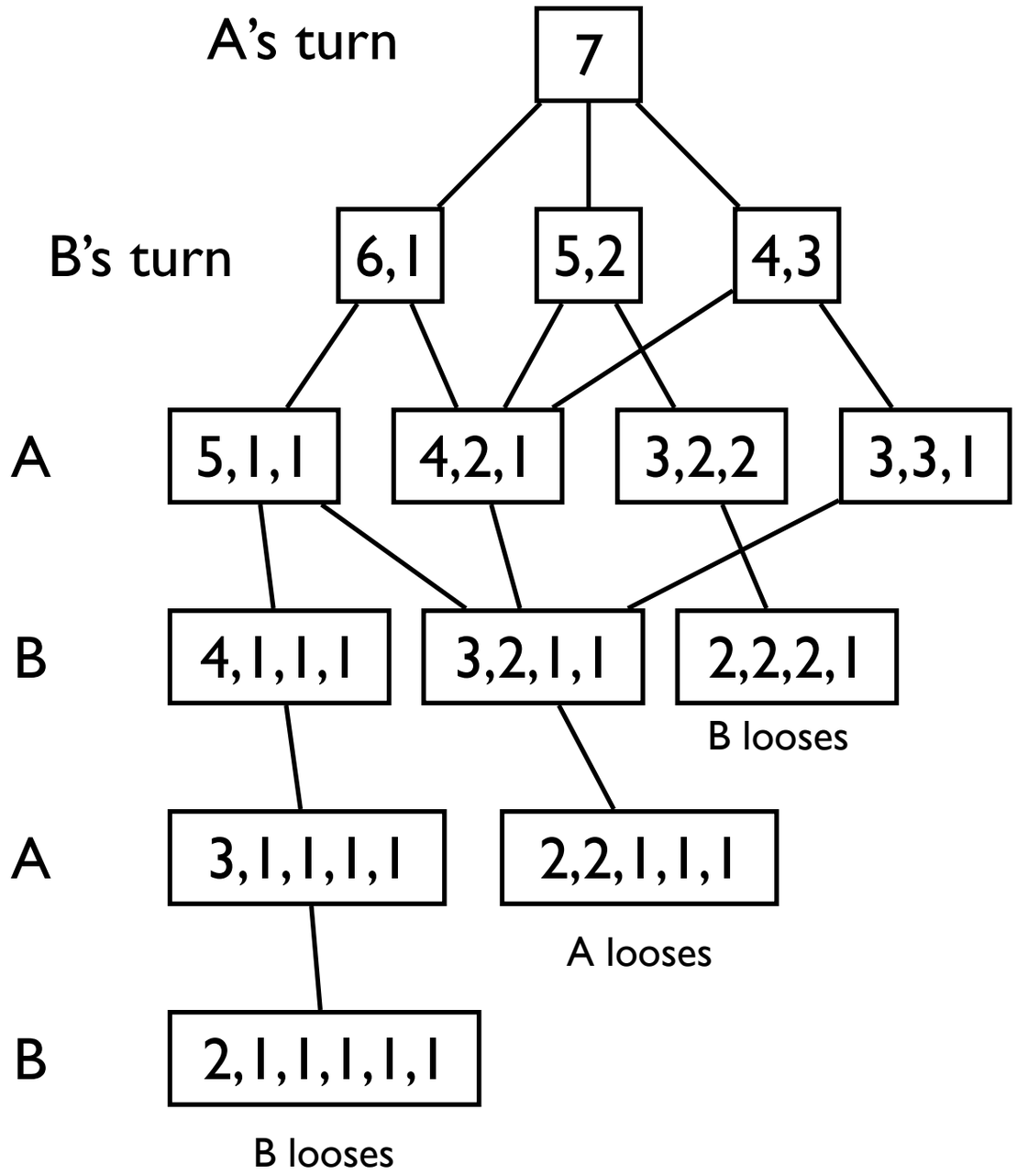
A game tree for pennies

- Pennies game:
  - stack of pennies
  - each player divides one of the stacks into two *unequal* stacks
  - game ends when every stack contains one or two pennies
  - first player who cannot play looses
- How do you represent the game state?
- How do we choose the best moves?
- What's the optimal strategy for searching the tree?

In class exercise:
Draw the game tree.

A game tree for pennies

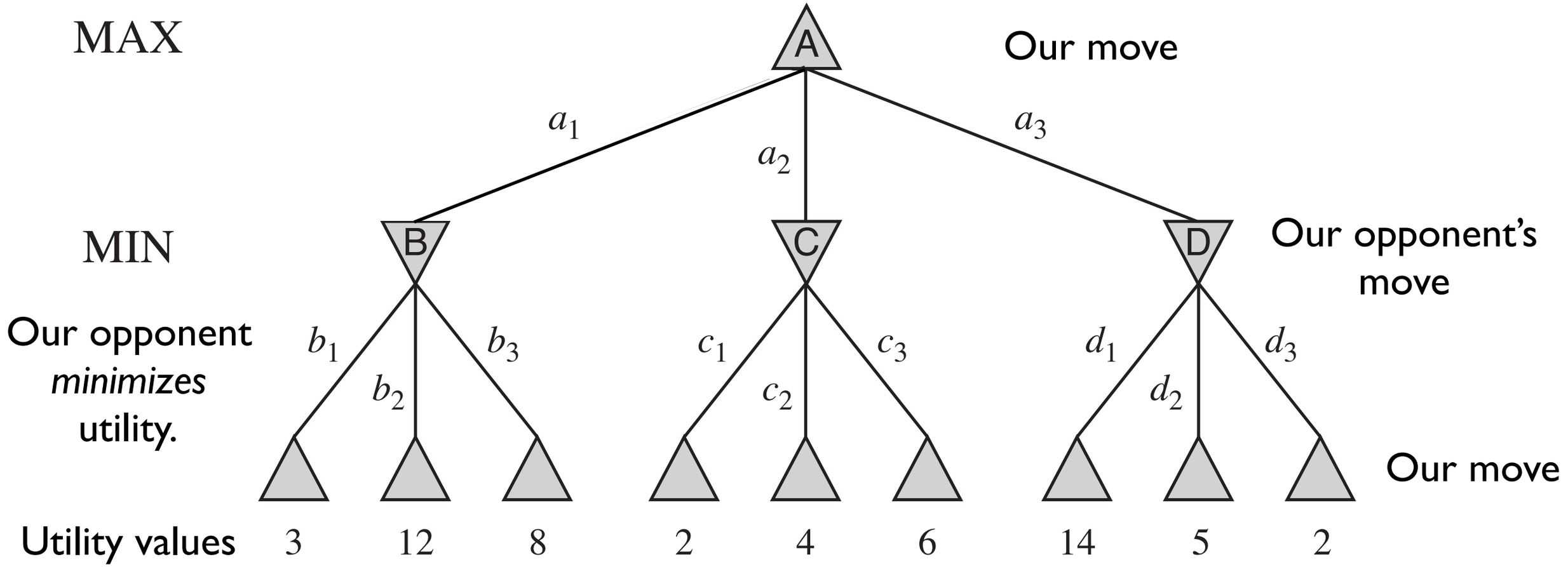
- Pennies game:
  - stack of pennies
  - each player divides one of the stacks into two *unequal* stacks
  - game ends when every stack contains one or two pennies
  - first player who cannot play looses
- How do you represent the game state?
- How do we choose the best moves?
- What's the optimal strategy for searching the tree?



Minimax: perfect play for deterministic 2-player games

- Idea: Chose move to position (or state) with highest minimax value.
  Best achievable payoff against optimal player
- minimax-val(n) =
  - Utility(n) if terminal state
  - max s ∈ result(n) if n is max node
  - min s ∈ result(n) if n is min node

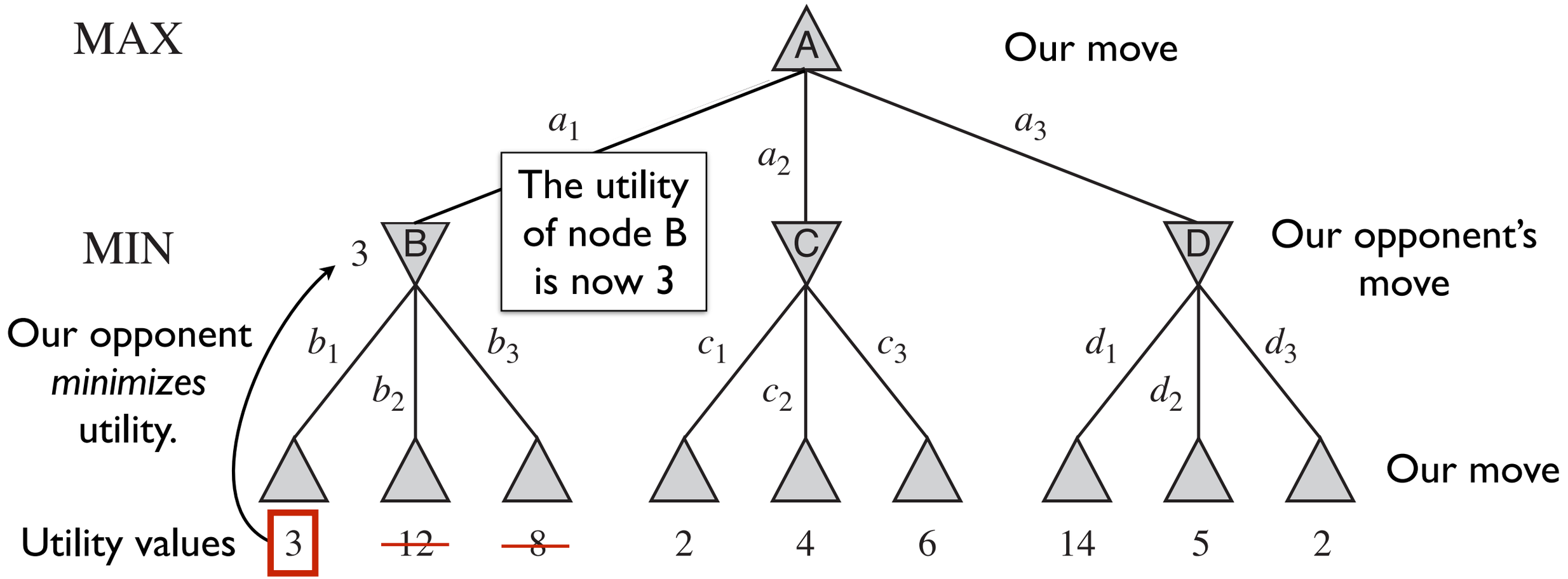
Utilities are computed recursively from terminal states.



Minimax: perfect play for deterministic 2-player games

- Idea: Chose move to position (or state) with highest minimax value.
  Best achievable payoff against optimal player
- minimax-val(n) =
  - Utility(n) if terminal state
  - max s ∈ result(n) if n is max node
  - min s ∈ result(n) if n is min node

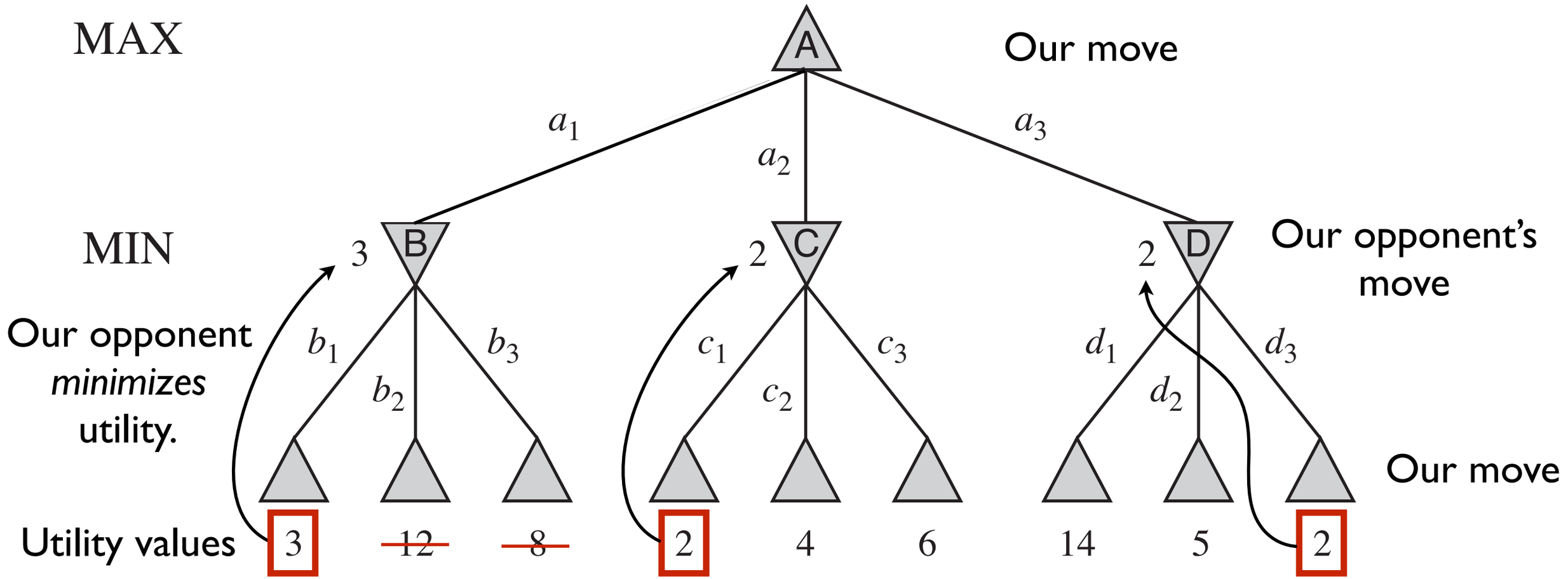
Utilities are computed recursively from terminal states.



Minimax: perfect play for deterministic 2-player games

- Idea: Chose move to position (or state) with highest minimax value.
  Best achievable payoff against optimal player
- minimax-val(n) =
  - Utility(n) if terminal state
  - max s ∈ result(n) if n is max node
  - min s ∈ result(n) if n is min node

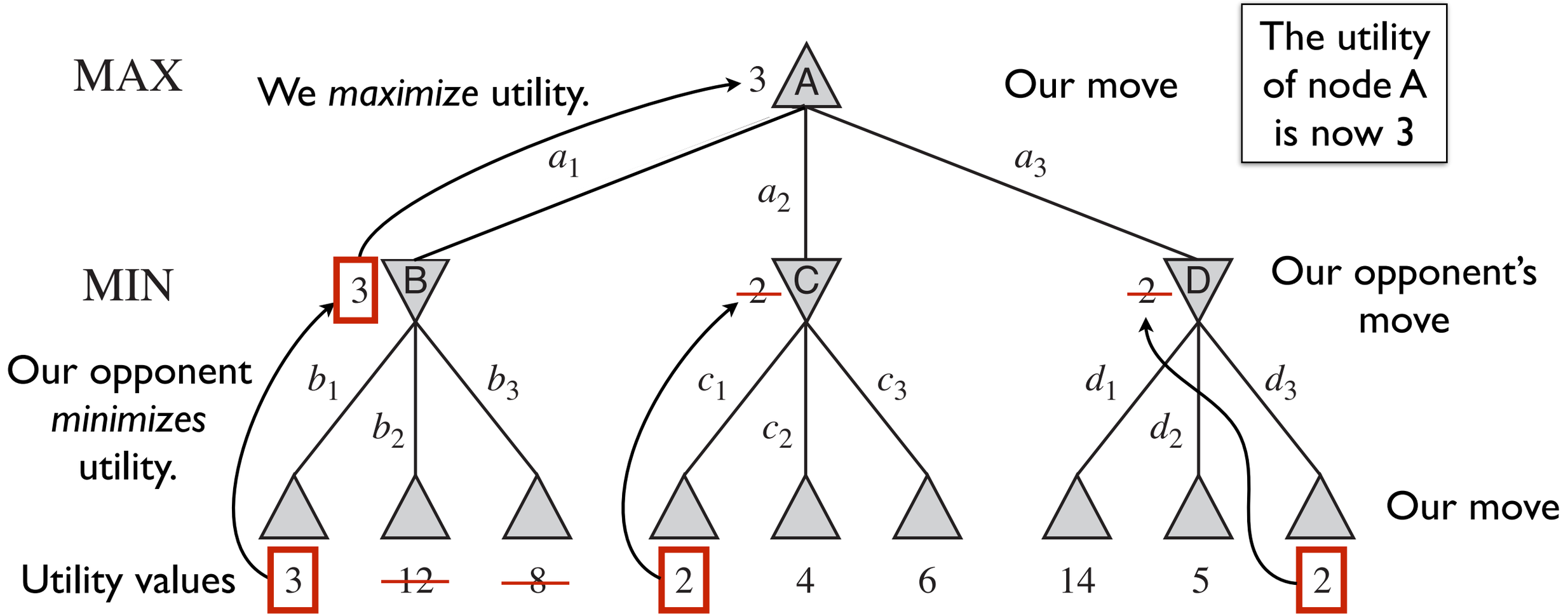
Utilities are computed recursively from terminal states.



Minimax: perfect play for deterministic 2-player games

- Idea: Chose move to position (or state) with highest minimax value.
  Best achievable payoff against optimal player
- minimax-val(n) =
  - Utility(n) if terminal state
  - max s ∈ result(n) if n is max node
  - min s ∈ result(n) if n is min node

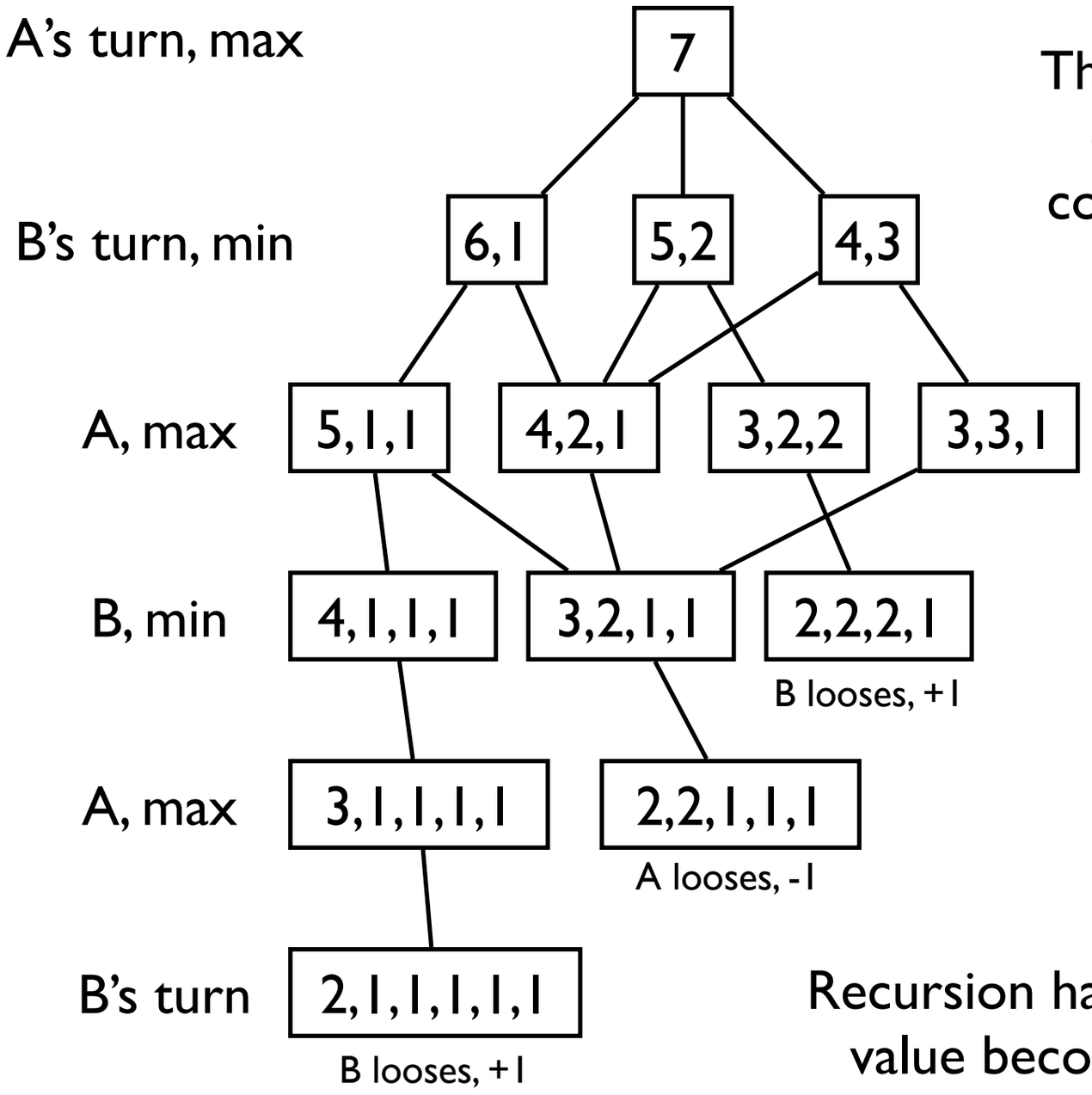
Utilities are computed recursively from terminal states.



General minimax algorithm

- The minimax is defined recursively.
- Base case is terminal nodes.
- Minimax(s) =
  - Utility(s) if Terminal-test(s)
  - max $_{a \in actions(s)}$ Minimax(Result(s,a)) if Player(s) = max
  - min $_{a \in actions(s)}$ Minimax(Result(s,a)) if Player(s) = min

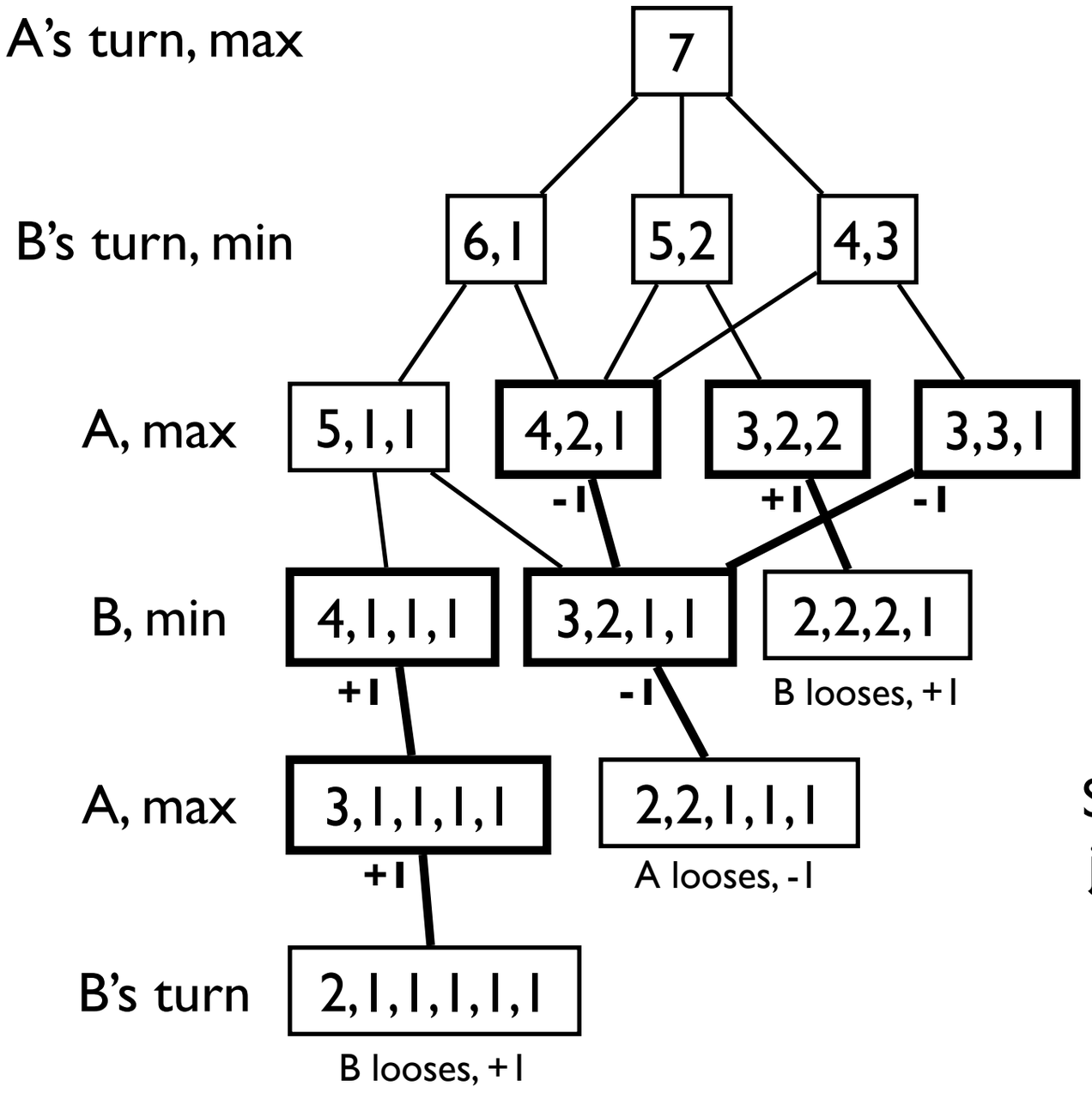
Minimax in the penny pile game



The minimax value for each game state is computed recursively.

Recursion halts when a utility value becomes known, i.e. when the game ends.

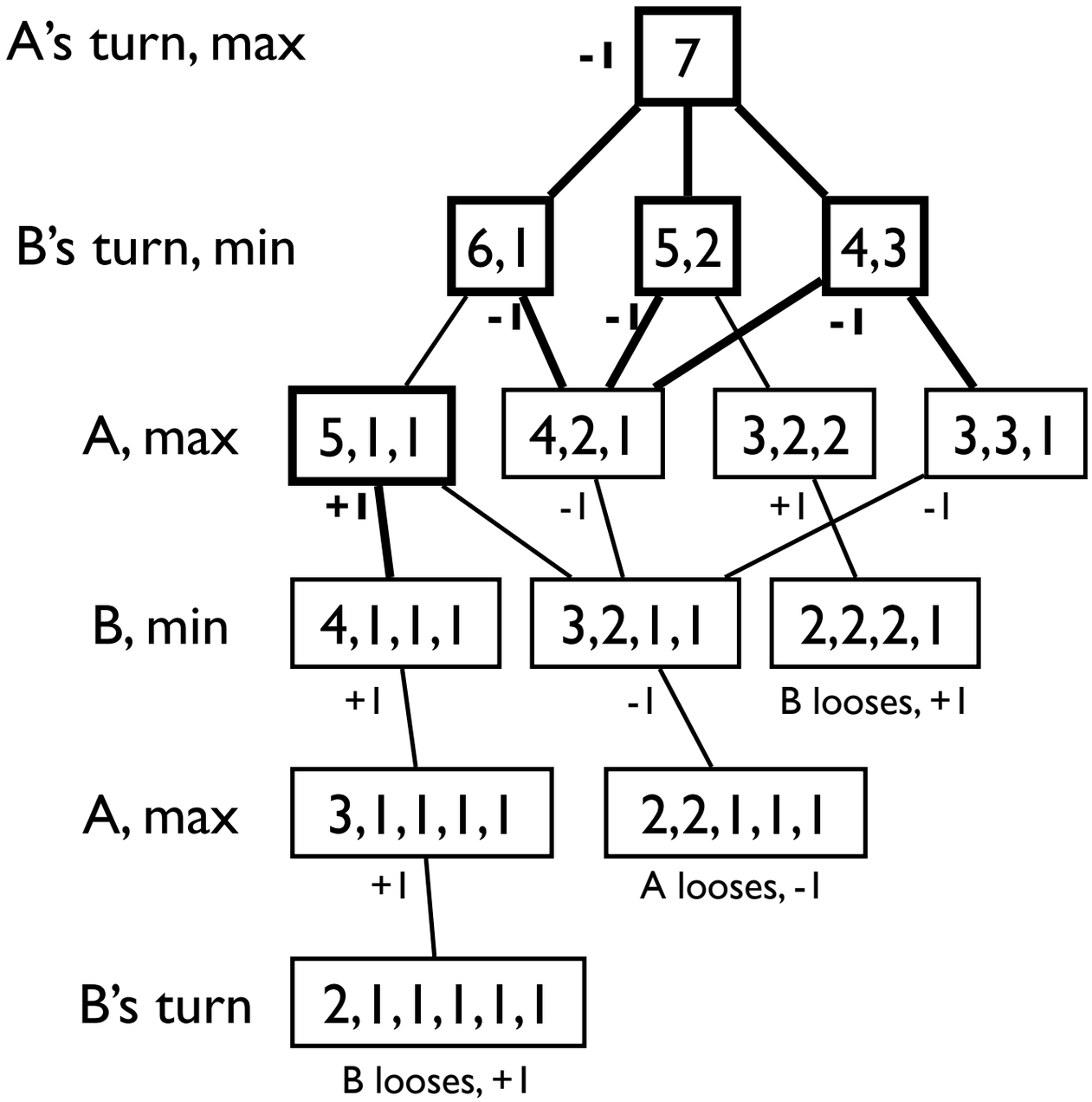
Minimax in the penny pile game



Each of these game states have only one legal move

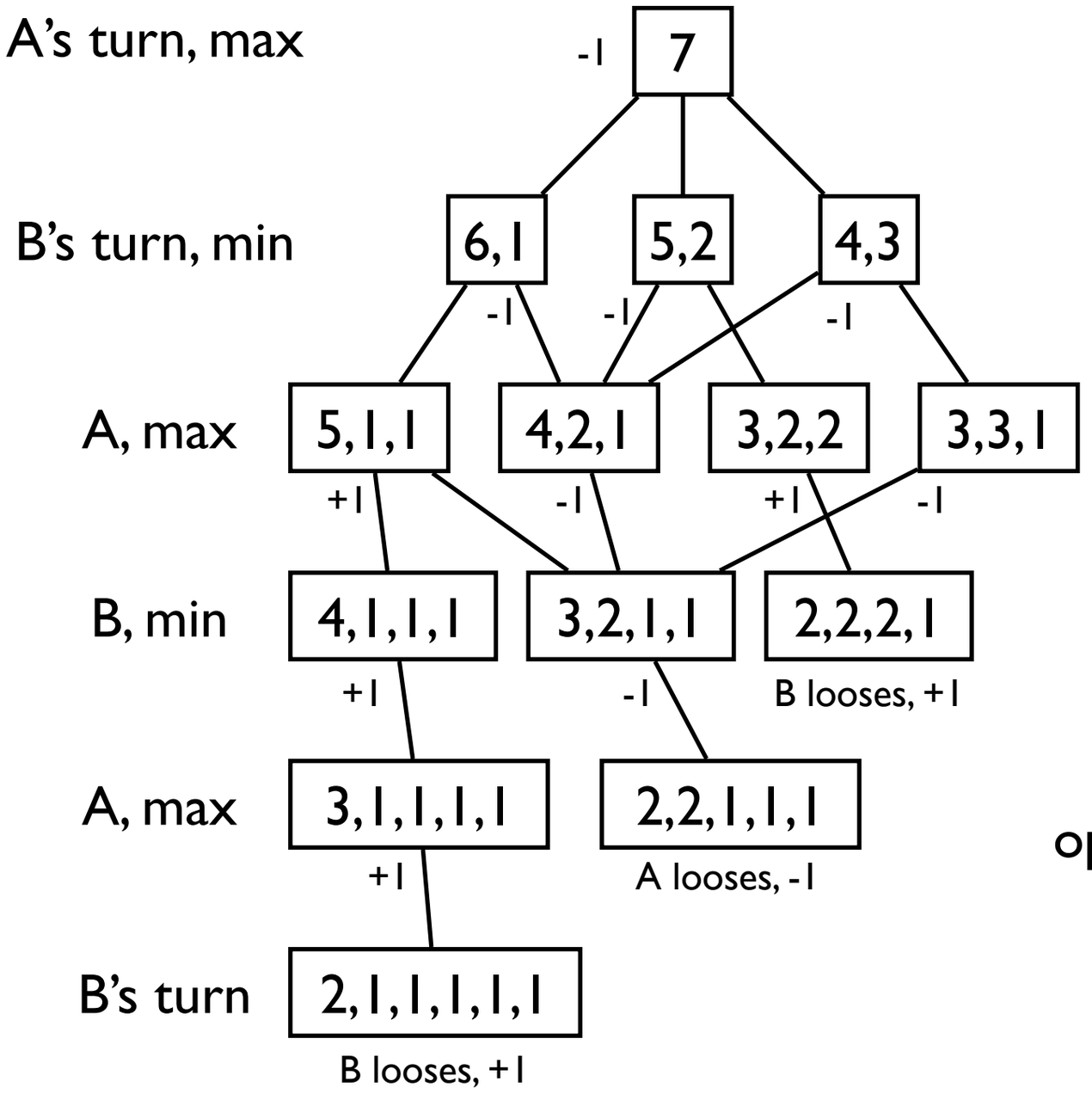
Singleton branches just propagate the utility value up: max(a) = a

Minimax in the penny pile game



The rest take the max or min of the available utilities.

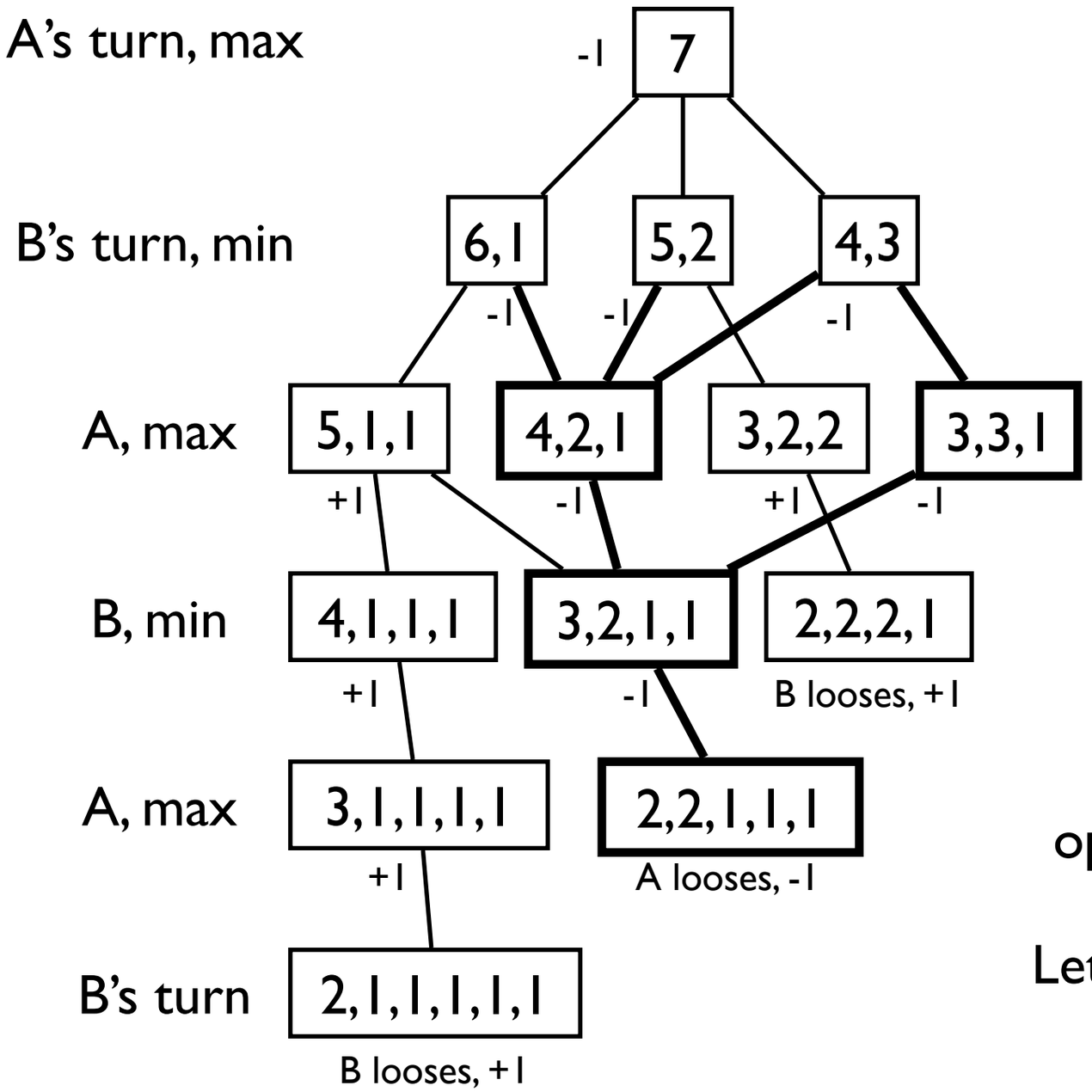
Minimax in the penny pile game



The rest take the max or min of the available utilities.

So, what's the optimal strategy?

Minimax in the penny pile game



The rest take the max or min of the available utilities.

So, what's the optimal strategy?

Let your opponent move first!

B can always force a win.

Properties of the minimax algorithm

- Complete?

- Optimal?

- Time complexity?

- Space complexity?

- Tic-tac-toe?
- Chess?

- Do we need to explore every path?

Properties of the minimax algorithm

- Complete? Yes.

- Optimal? Yes, if opponent is optimal. (If not? Even better)

- Time complexity? $O(b^m)$ b=branching factor, #legal moves
  m=depth of tree

- Space complexity? O(bm) depth-first exploration
  O(m) if generating one successor at a time

- Tic-tac-toe?
- Chess?

- Do we need to explore every path?

Properties of the minimax algorithm

- Complete? Yes.

- Optimal? Yes, if opponent is optimal. (If not? Even better)

- Time complexity? $O(b^m)$ b=branching factor, #legal moves
  m=depth of tree

- Space complexity? O(bm) depth-first exploration
  O(m) if generating one successor at a time

- Tic-tac-toe? b=9, m=9 9^9 = 387 million
- Chess?

- Do we need to explore every path?

Properties of the minimax algorithm

- Complete? Yes.

- Optimal? Yes, if opponent is optimal. (If not? Even better)

- Time complexity? $O(b^m)$ b=branching factor, #legal moves
  m=depth of tree

- Space complexity? O(bm) depth-first exploration
  O(m) if generating one successor at a time

- Tic-tac-toe? b=9, m=9 9^9 = 387 million
- Chess? $b \approx 35$, $m \approx 100$ $O(35^{100})$

- Do we need to explore every path?

Properties of the minimax algorithm

- Complete? Yes.

- Optimal? Yes, if opponent is optimal. (If not? Even better)

- Time complexity? $O(b^m)$ b=branching factor, #legal moves
  m=depth of tree

- Space complexity? O(bm) depth-first exploration
  O(m) if generating one successor at a time

- Tic-tac-toe? b=9, m=9 9^9 = 387 million
- Chess? $b \approx 35, m \approx 100$ $O(35^{100})$

- Do we need to explore every path? No.

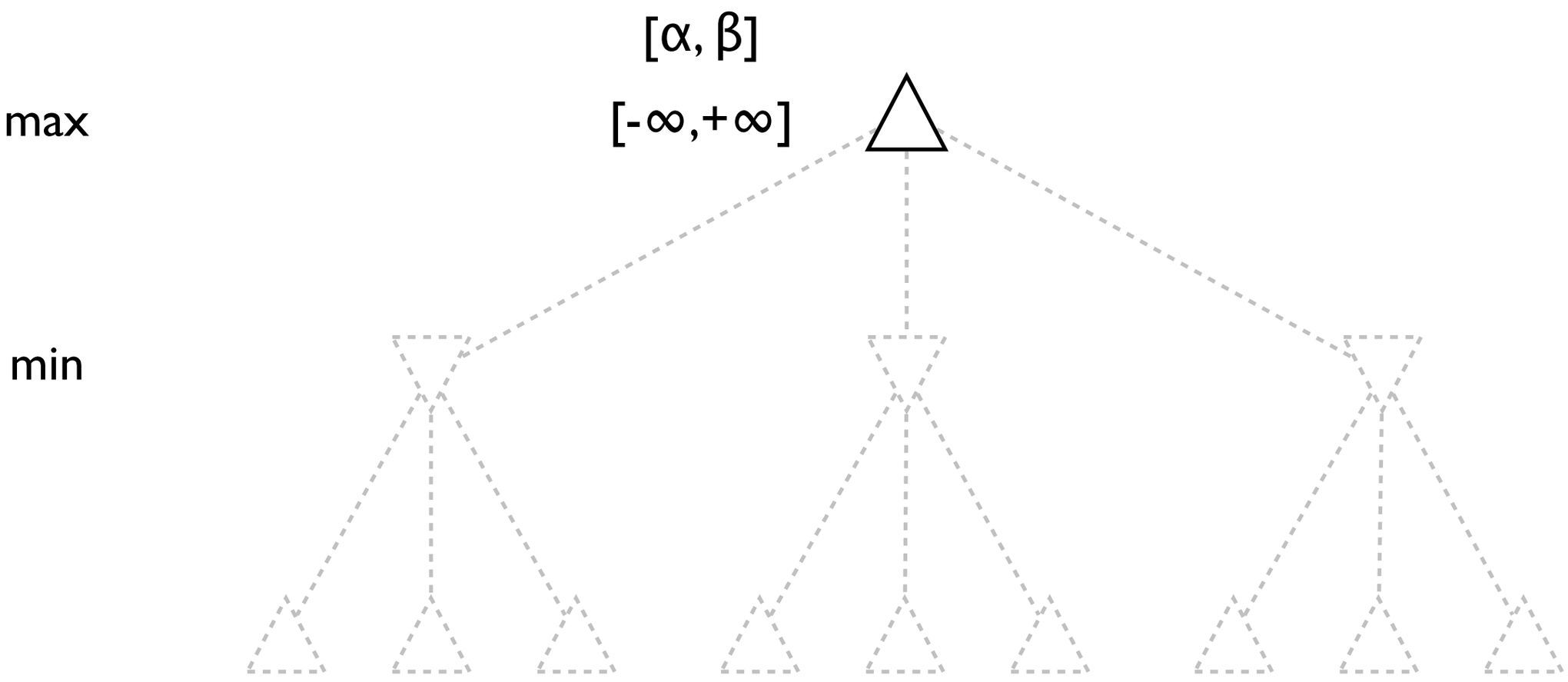
Some branches can be *pruned*.

Alpha-beta pruning

- Idea: prune branches that can't influence final decision
- alpha-beta pruning keeps track of the range of possible values:

  α = max's best choice (highest val.) so far at any point along path
  β = min's best choice (lowest val.) so far at any point along path

- alpha-beta doesn't search: it "*prunes*" branches that don't affect current estimates

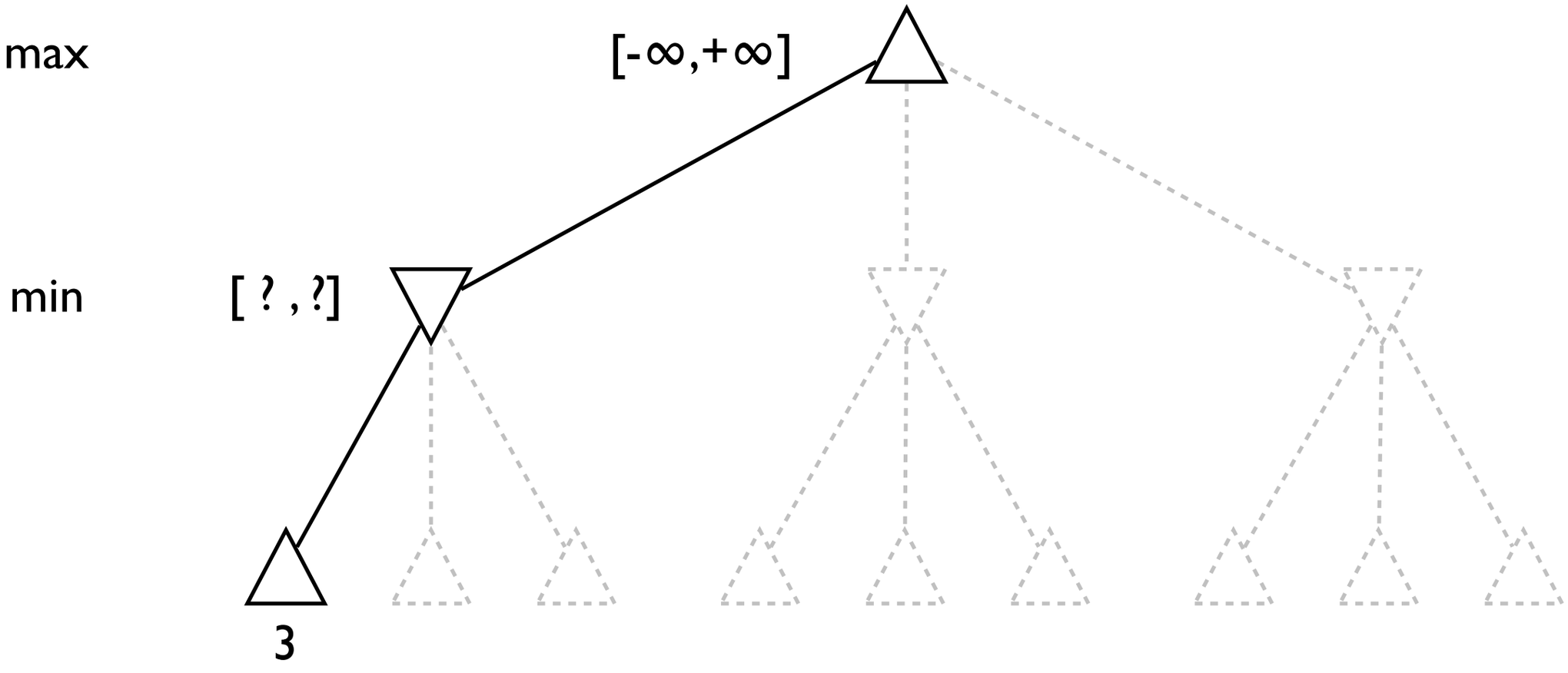


Alpha-beta pruning

- Idea: prune branches that can't influence final decision
- alpha-beta pruning keeps track of the range of possible values:

  α = max's best choice (highest val.) so far at any point along path
  β = min's best choice (lowest val.) so far at any point along path

- alpha-beta doesn't search: it "*prunes*" branches that don't affect current estimates

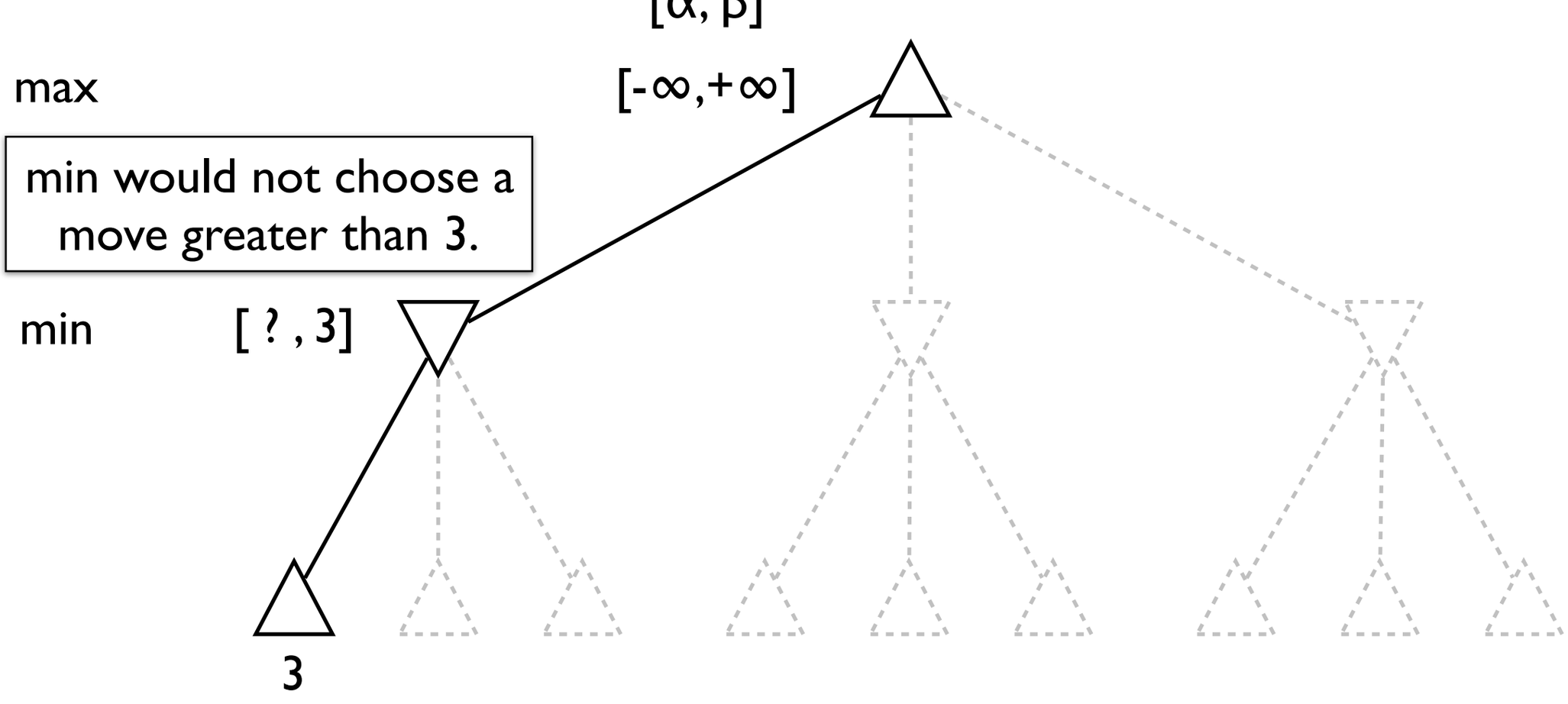


Alpha-beta pruning

- Idea: prune branches that can't influence final decision
- alpha-beta pruning keeps track of the range of possible values:

  α = max's best choice (highest val.) so far at any point along path
  β = min's best choice (lowest val.) so far at any point along path

- alpha-beta doesn't search: it "*prunes*" branches that don't affect current estimates

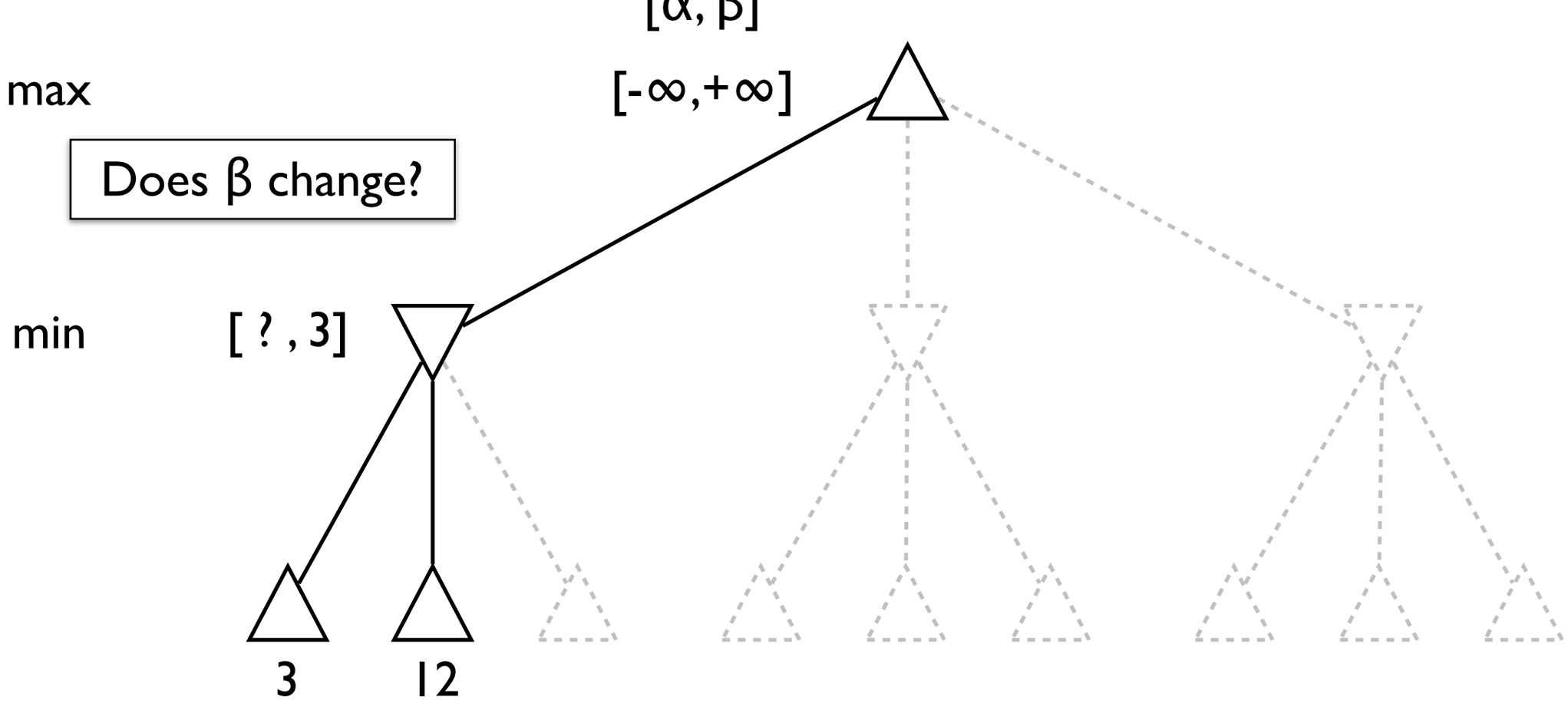


Alpha-beta pruning

- Idea: prune branches that can't influence final decision
- alpha-beta pruning keeps track of the range of possible values:

  α = max's best choice (highest val.) so far at any point along path
  β = min's best choice (lowest val.) so far at any point along path

- alpha-beta doesn't search: it "*prunes*" branches that don't affect current estimates

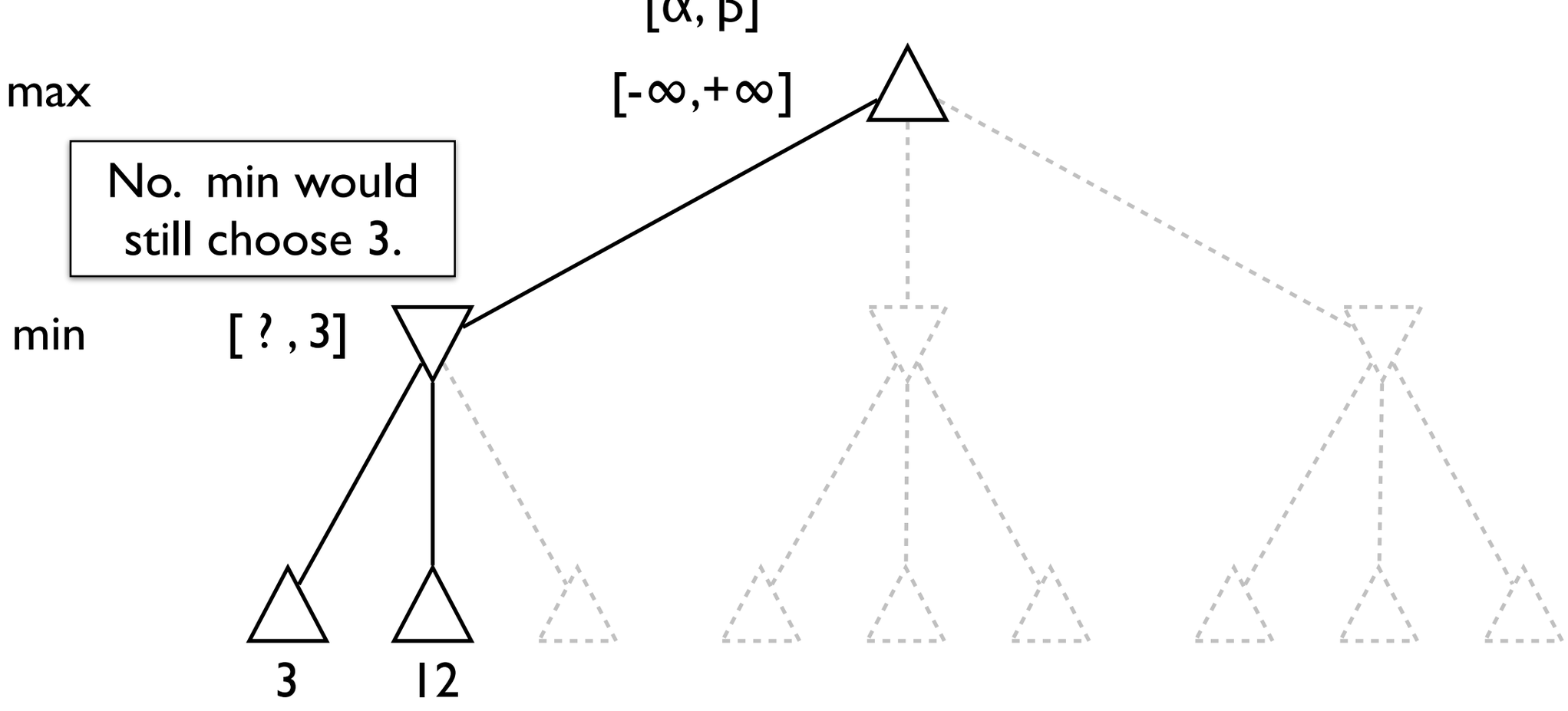


Alpha-beta pruning

- Idea: prune branches that can't influence final decision
- alpha-beta pruning keeps track of the range of possible values:

  α = max's best choice (highest val.) so far at any point along path
  β = min's best choice (lowest val.) so far at any point along path

- alpha-beta doesn't search: it "*prunes*" branches that don't affect current estimates

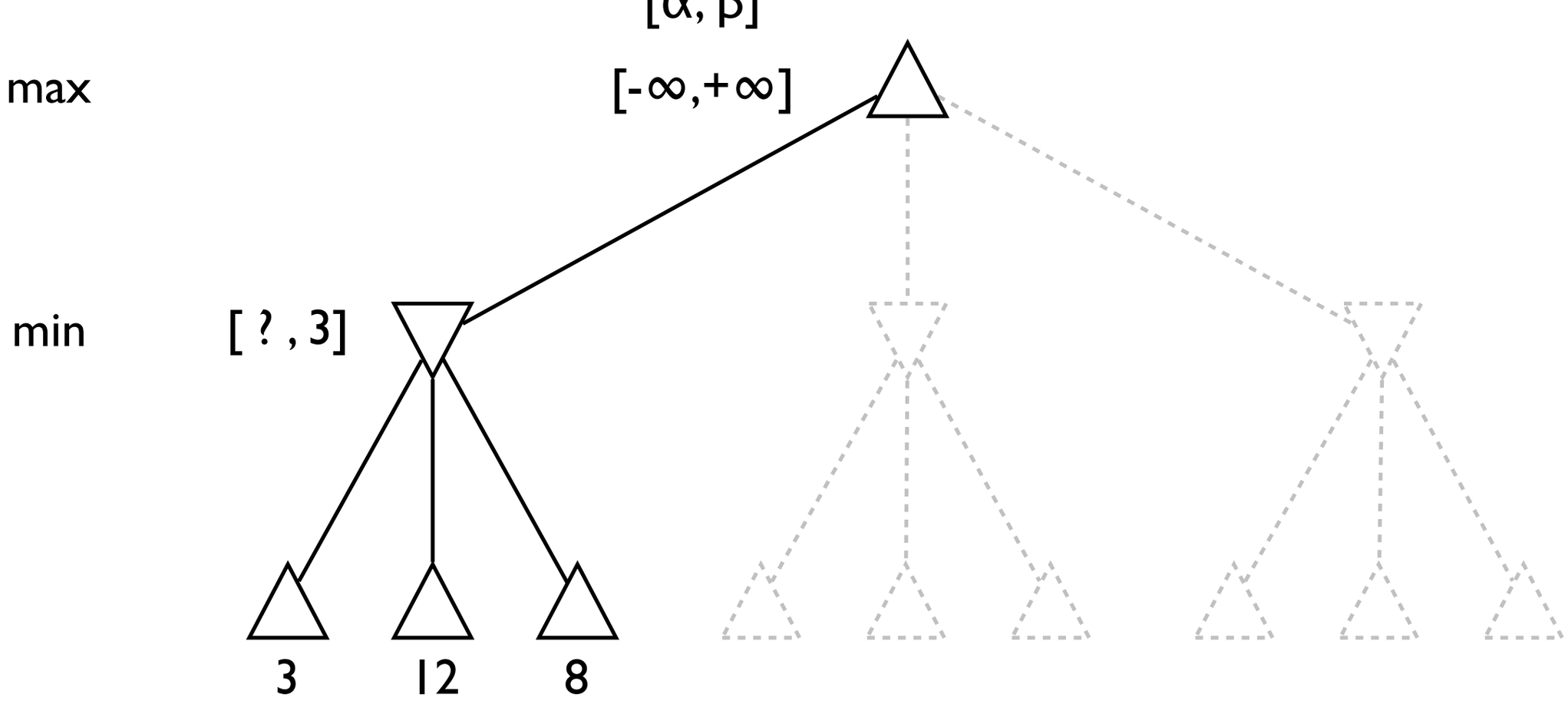


Alpha-beta pruning

- Idea: prune branches that can't influence final decision
- alpha-beta pruning keeps track of the range of possible values:

  α = max's best choice (highest val.) so far at any point along path
  β = min's best choice (lowest val.) so far at any point along path

- alpha-beta doesn't search: it "*prunes*" branches that don't affect current estimates

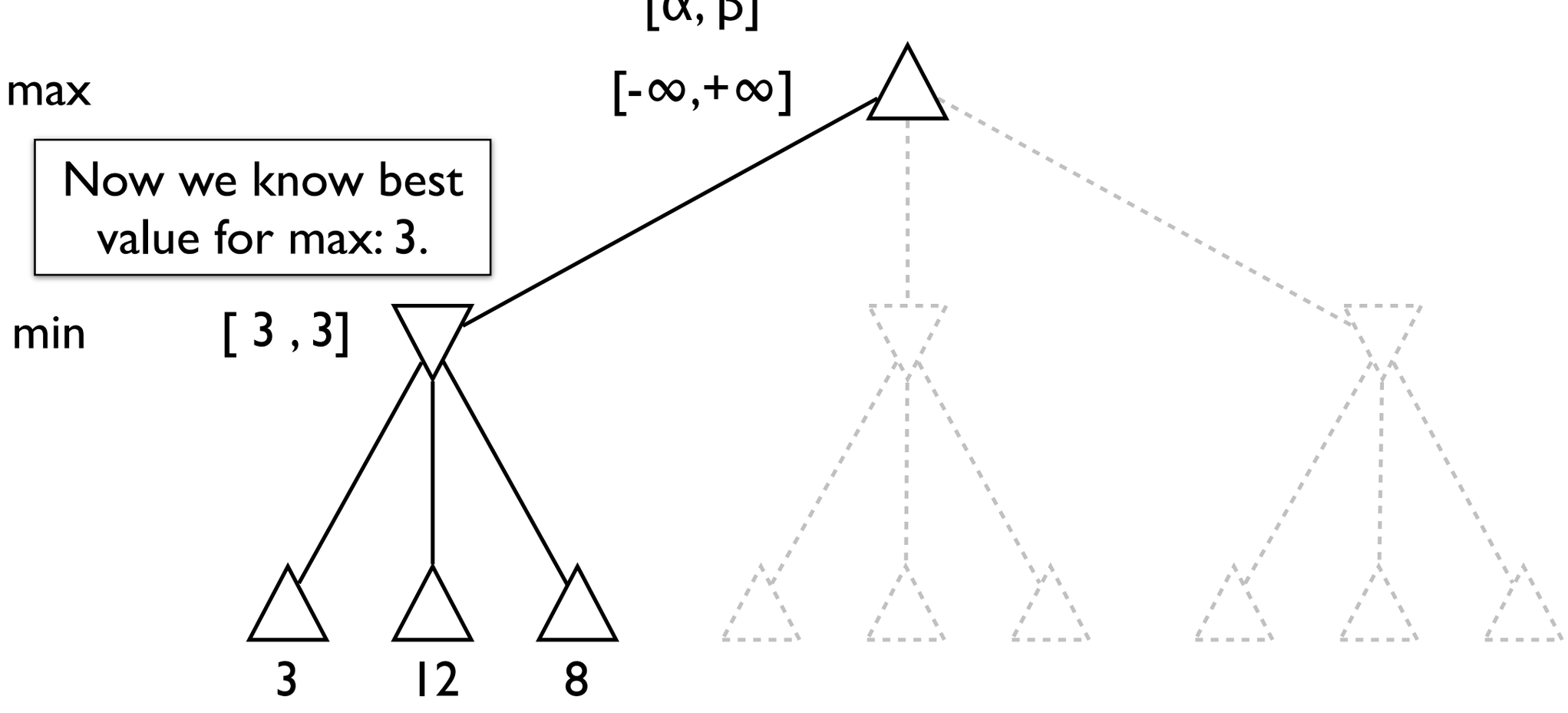


Alpha-beta pruning

- Idea: prune branches that can't influence final decision
- alpha-beta pruning keeps track of the range of possible values:

  α = max's best choice (highest val.) so far at any point along path
  β = min's best choice (lowest val.) so far at any point along path

- alpha-beta doesn't search: it "*prunes*" branches that don't affect current estimates

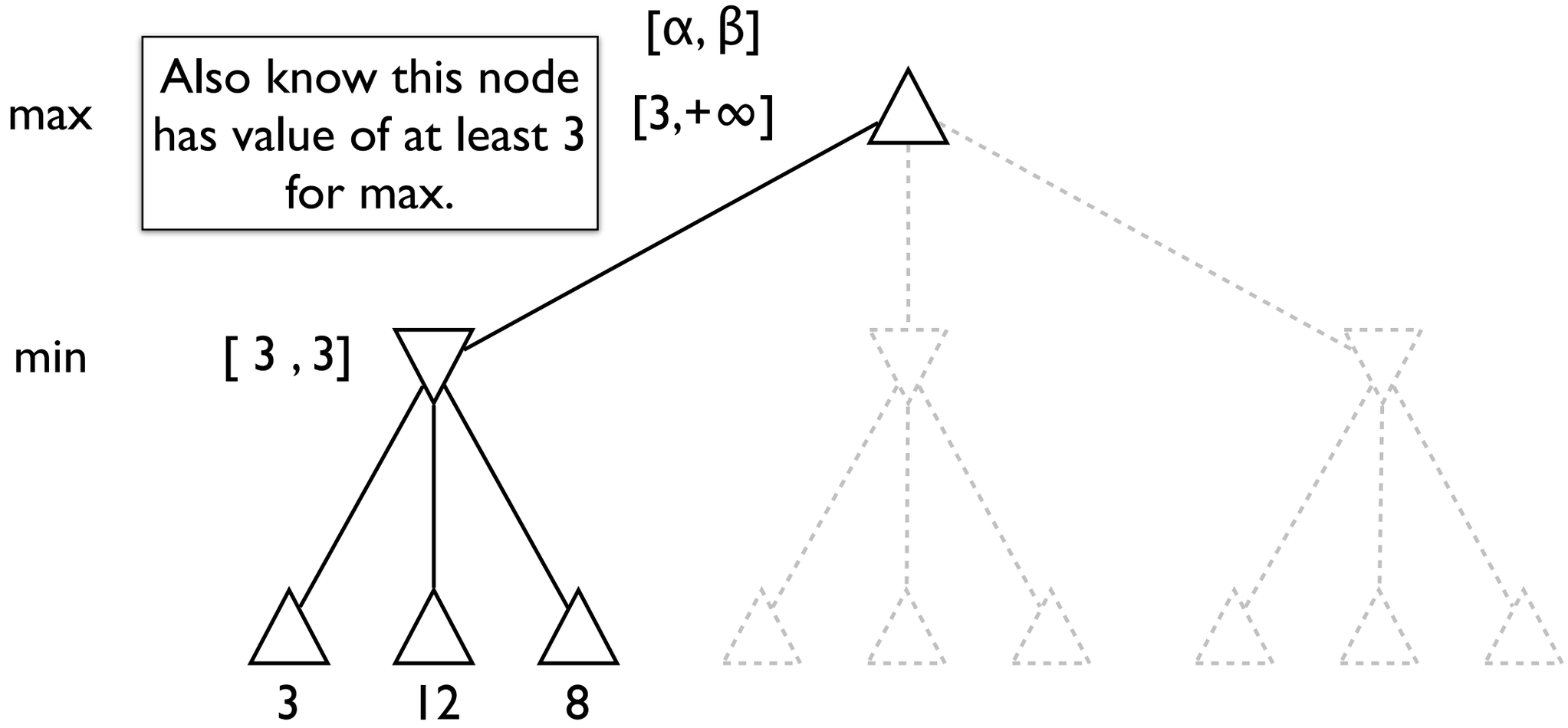


Alpha-beta pruning

- Idea: prune branches that can't influence final decision
- alpha-beta pruning keeps track of the range of possible values:

  α = max's best choice (highest val.) so far at any point along path
  β = min's best choice (lowest val.) so far at any point along path

- alpha-beta doesn't search: it "*prunes*" branches that don't affect current estimates

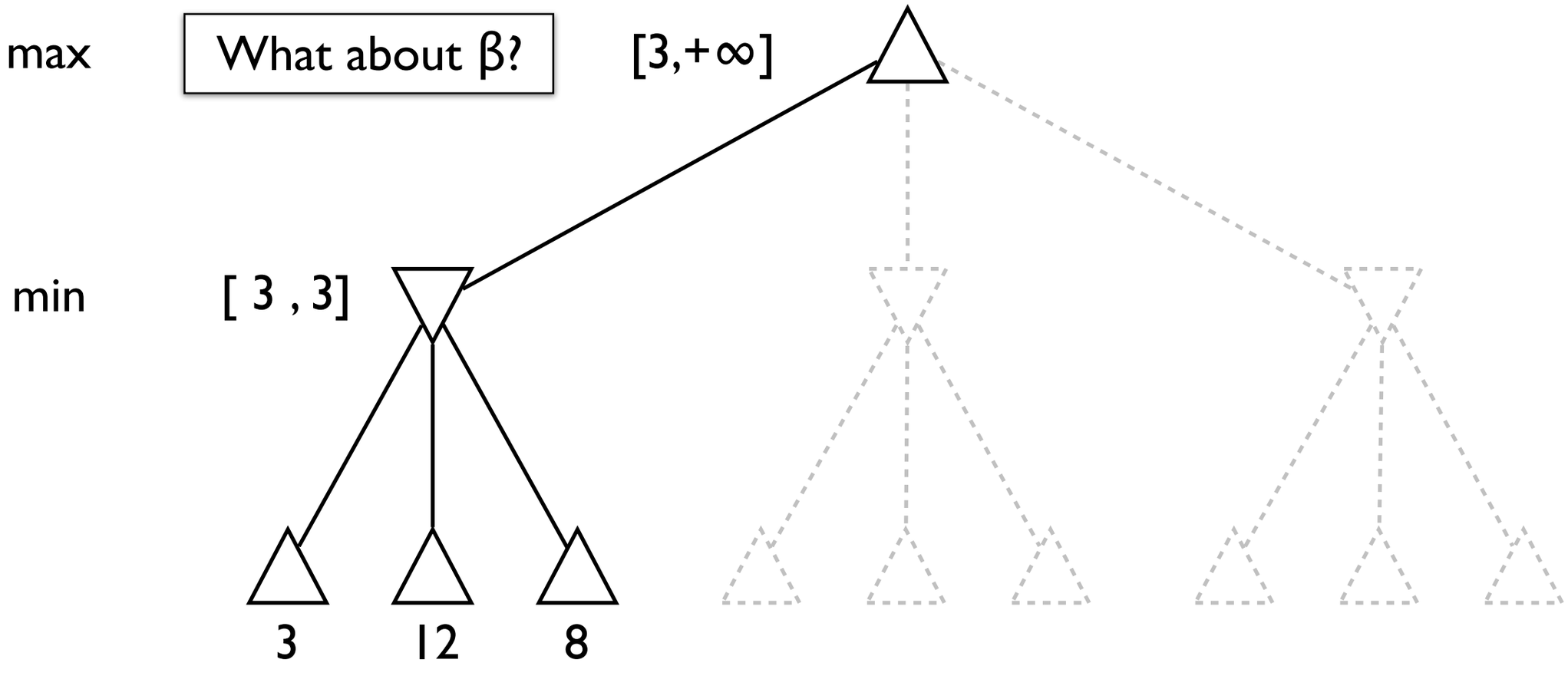


Alpha-beta pruning

- Idea: prune branches that can't influence final decision
- alpha-beta pruning keeps track of the range of possible values:

  α = max's best choice (highest val.) so far at any point along path
  β = min's best choice (lowest val.) so far at any point along path

- alpha-beta doesn't search: it "*prunes*" branches that don't affect current estimates

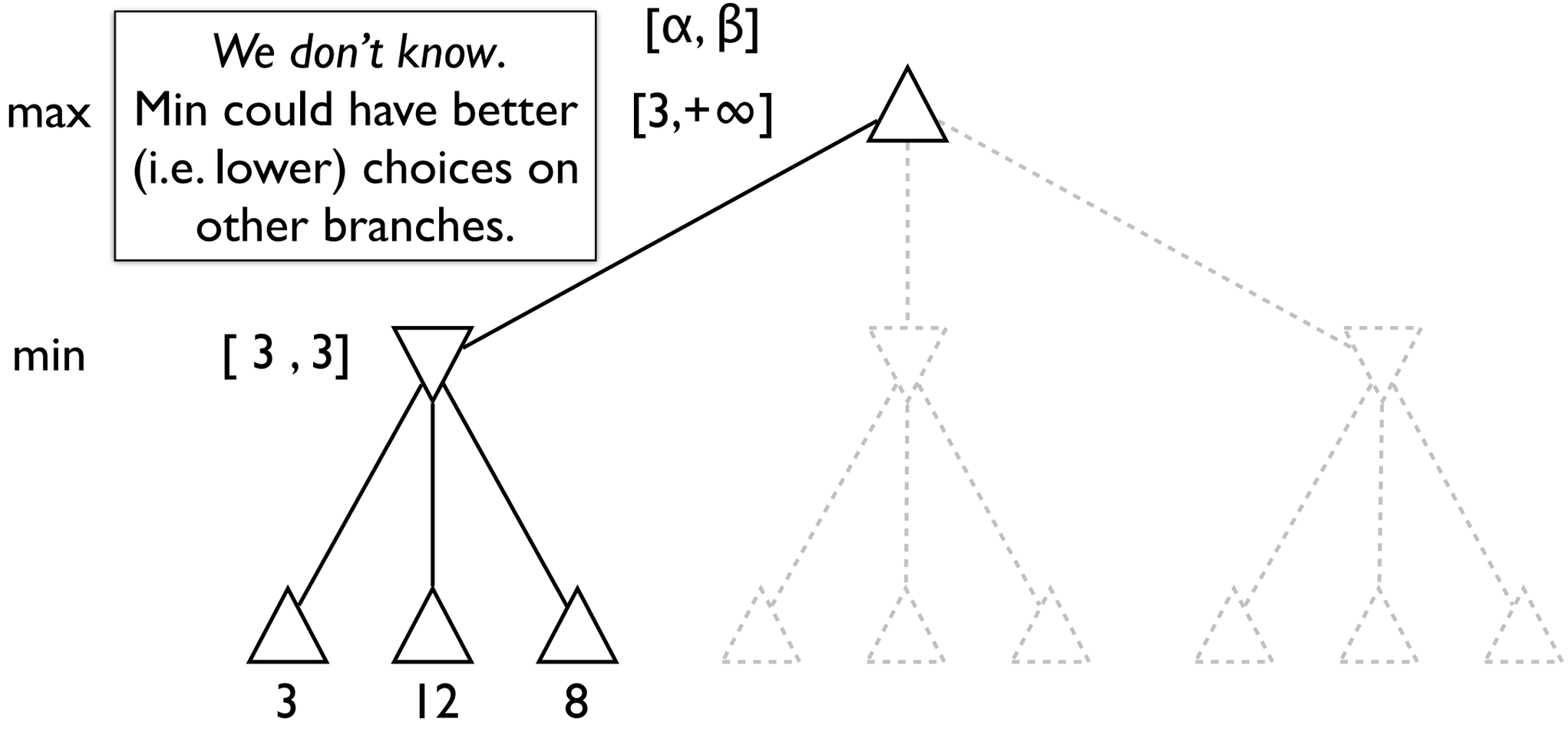


Alpha-beta pruning

- Idea: prune branches that can't influence final decision
- alpha-beta pruning keeps track of the range of possible values:

  α = max's best choice (highest val.) so far at any point along path
  β = min's best choice (lowest val.) so far at any point along path

- alpha-beta doesn't search: it "*prunes*" branches that don't affect current estimates

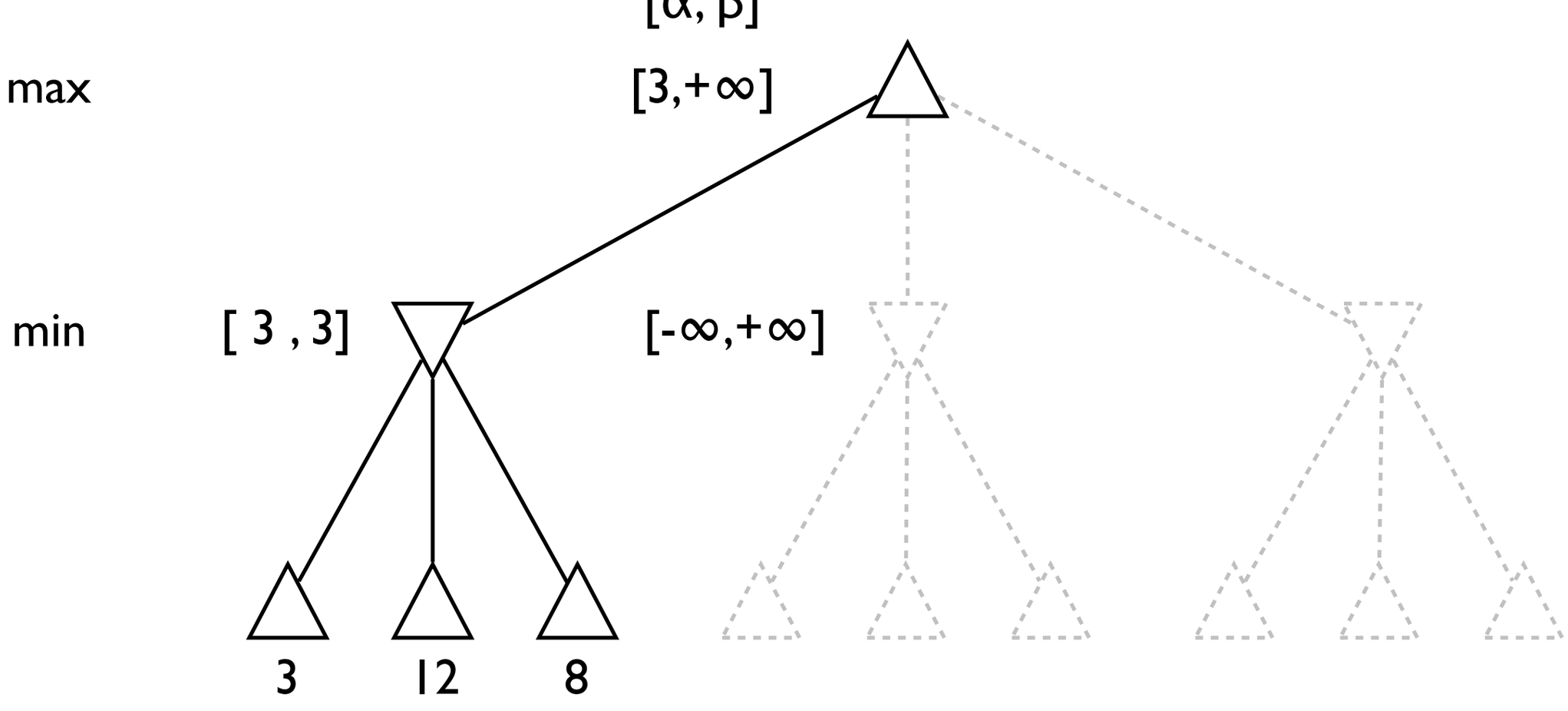


Alpha-beta pruning

- Idea: prune branches that can't influence final decision
- alpha-beta pruning keeps track of the range of possible values:

  α = max's best choice (highest val.) so far at any point along path
  β = min's best choice (lowest val.) so far at any point along path

- alpha-beta doesn't search: it "*prunes*" branches that don't affect current estimates

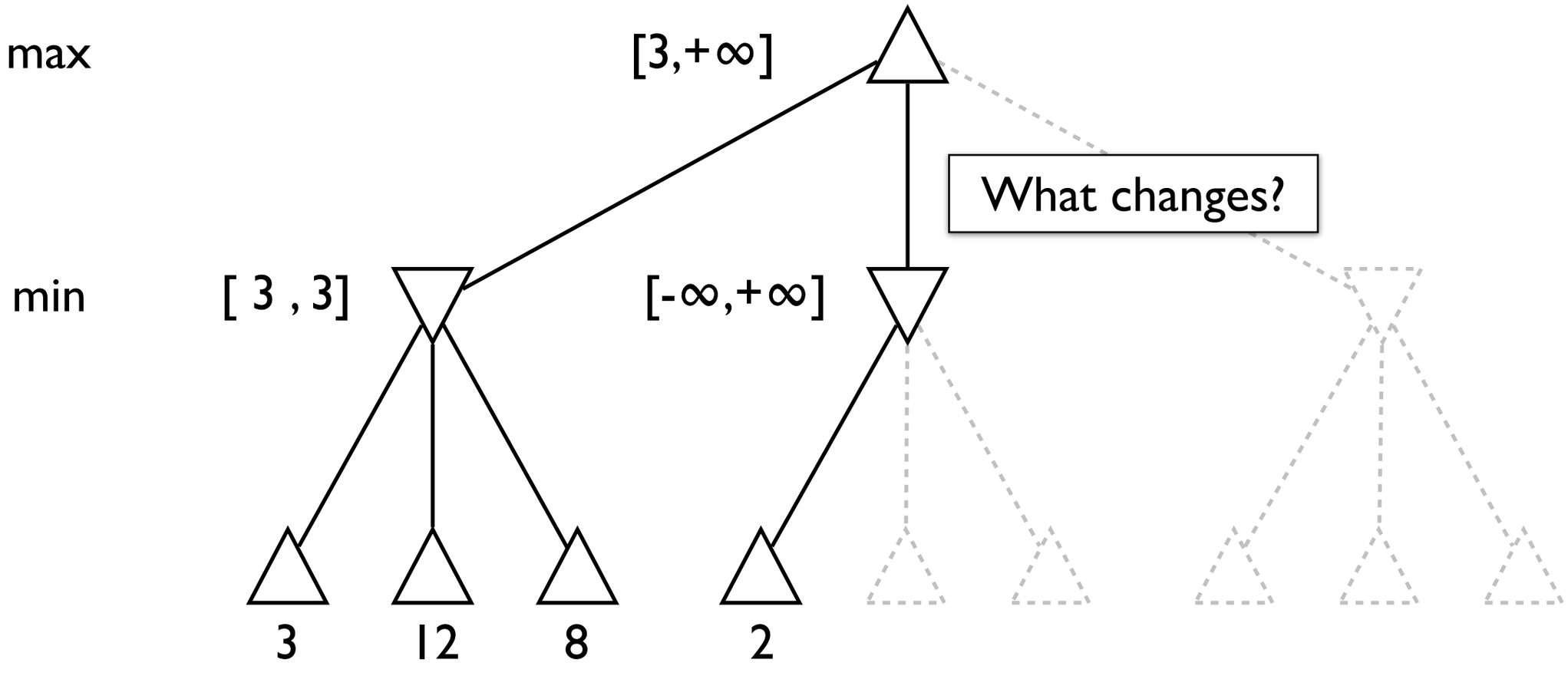


Alpha-beta pruning

- Idea: prune branches that can't influence final decision
- alpha-beta pruning keeps track of the range of possible values:

  α = max's best choice (highest val.) so far at any point along path
  β = min's best choice (lowest val.) so far at any point along path

- alpha-beta doesn't search: it "*prunes*" branches that don't affect current estimates

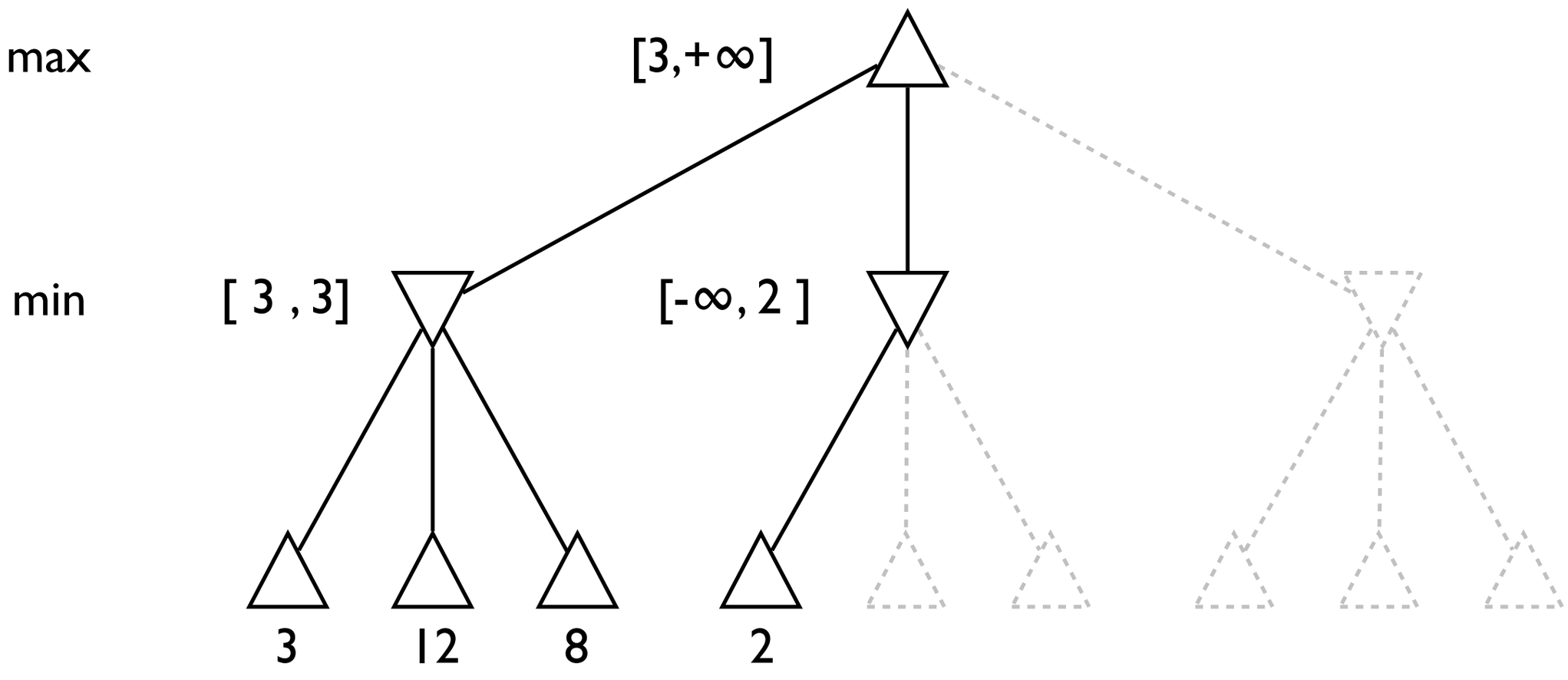


Alpha-beta pruning

- Idea: prune branches that can't influence final decision
- alpha-beta pruning keeps track of the range of possible values:

  α = max's best choice (highest val.) so far at any point along path
  β = min's best choice (lowest val.) so far at any point along path

- alpha-beta doesn't search: it "*prunes*" branches that don't affect current estimates

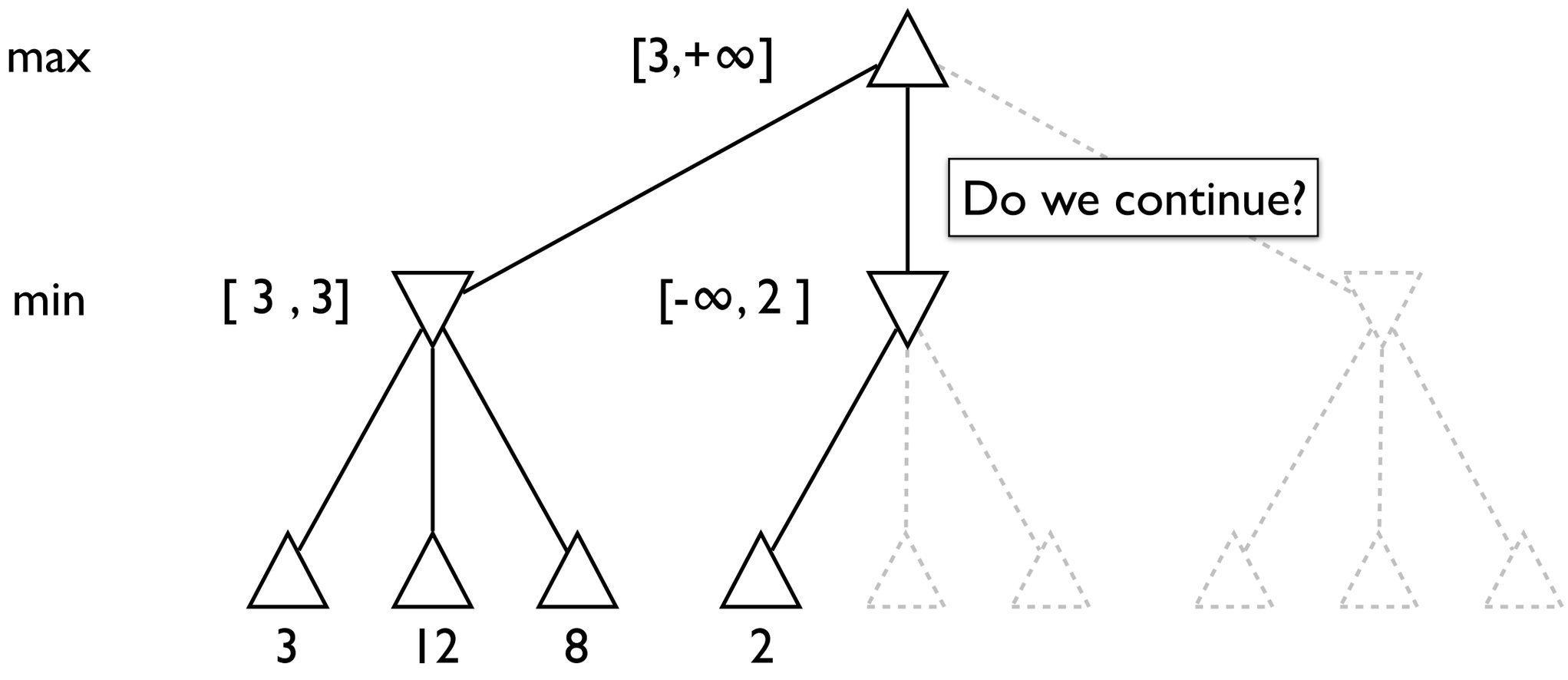


Alpha-beta pruning

- Idea: prune branches that can't influence final decision
- alpha-beta pruning keeps track of the range of possible values:

  α = max's best choice (highest val.) so far at any point along path
  β = min's best choice (lowest val.) so far at any point along path

- alpha-beta doesn't search: it "*prunes*" branches that don't affect current estimates

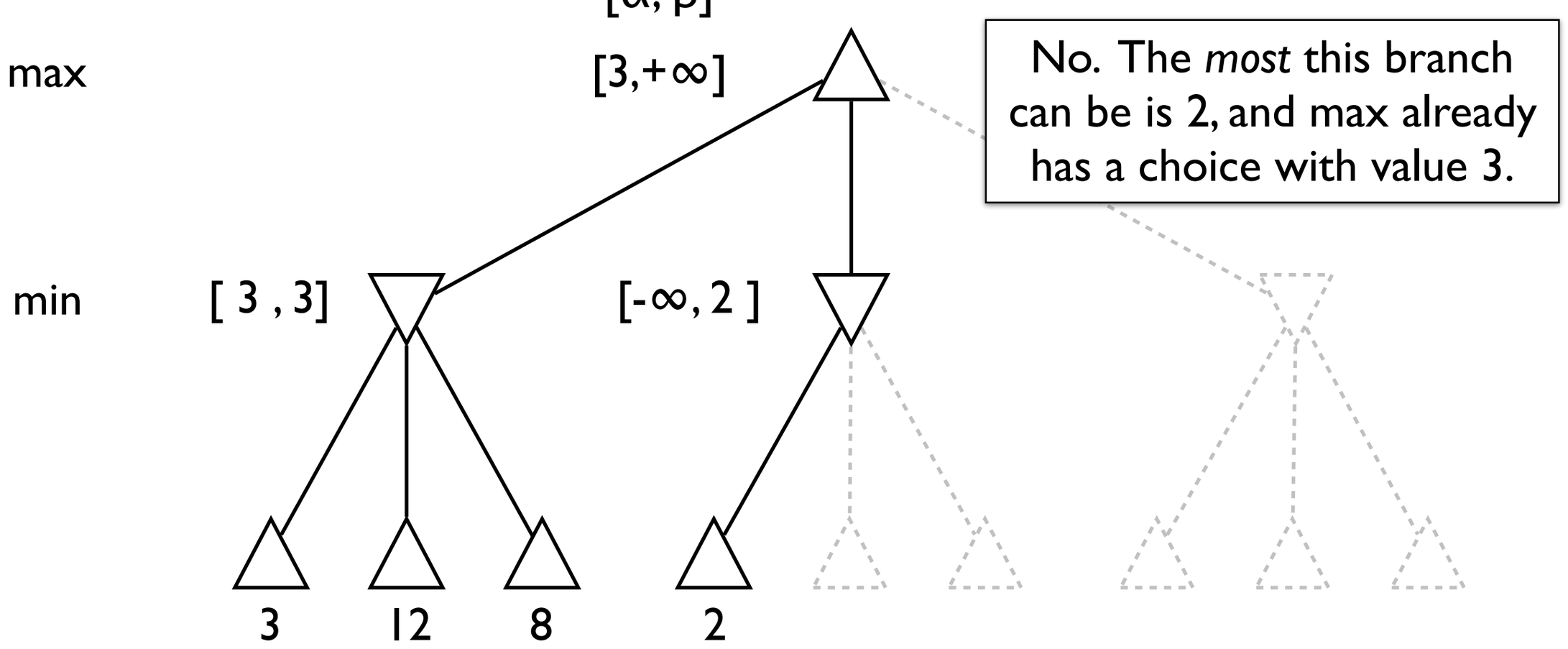


Alpha-beta pruning

- Idea: prune branches that can't influence final decision
- alpha-beta pruning keeps track of the range of possible values:

  α = max's best choice (highest val.) so far at any point along path
  β = min's best choice (lowest val.) so far at any point along path

- alpha-beta doesn't search: it "*prunes*" branches that don't affect current estimates

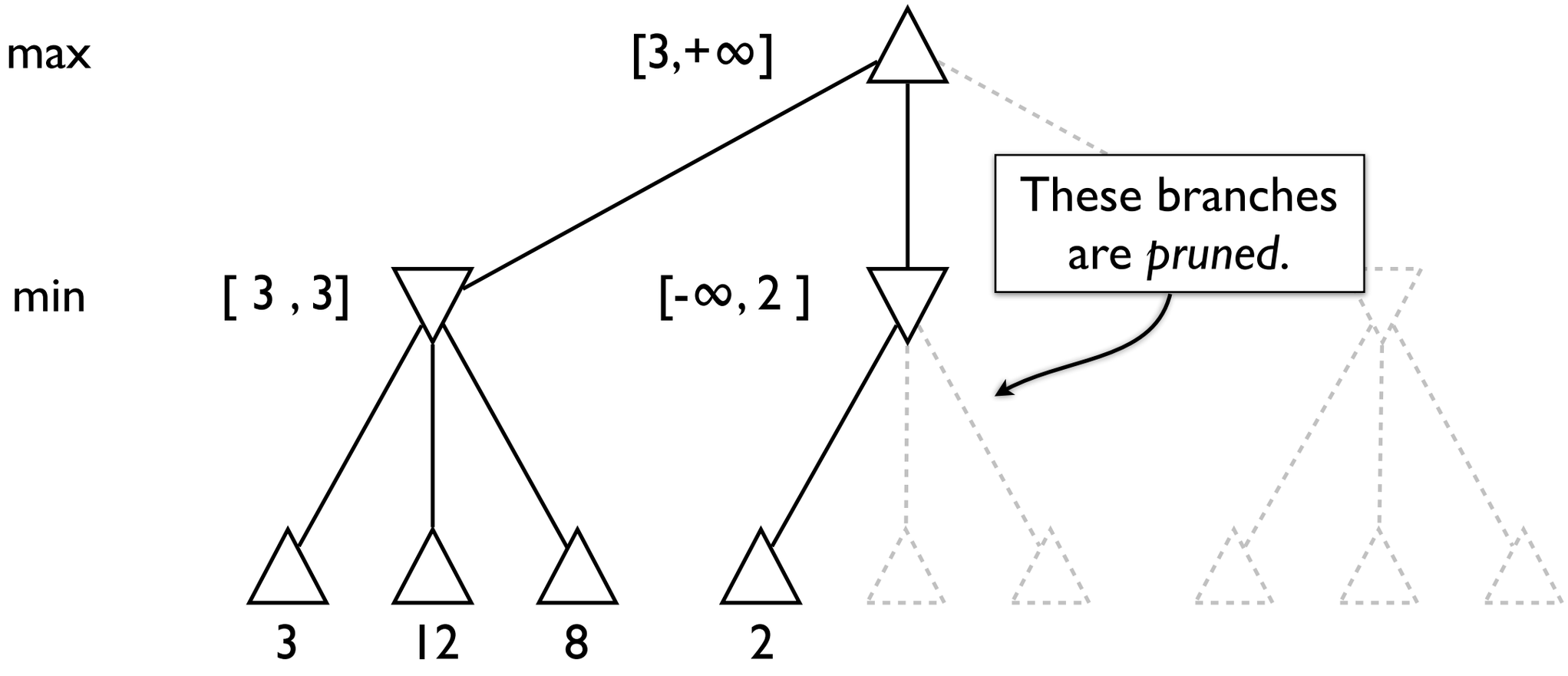


Alpha-beta pruning

- Idea: prune branches that can't influence final decision
- alpha-beta pruning keeps track of the range of possible values:

  α = max's best choice (highest val.) so far at any point along path
  β = min's best choice (lowest val.) so far at any point along path

- alpha-beta doesn't search: it "*prunes*" branches that don't affect current estimates

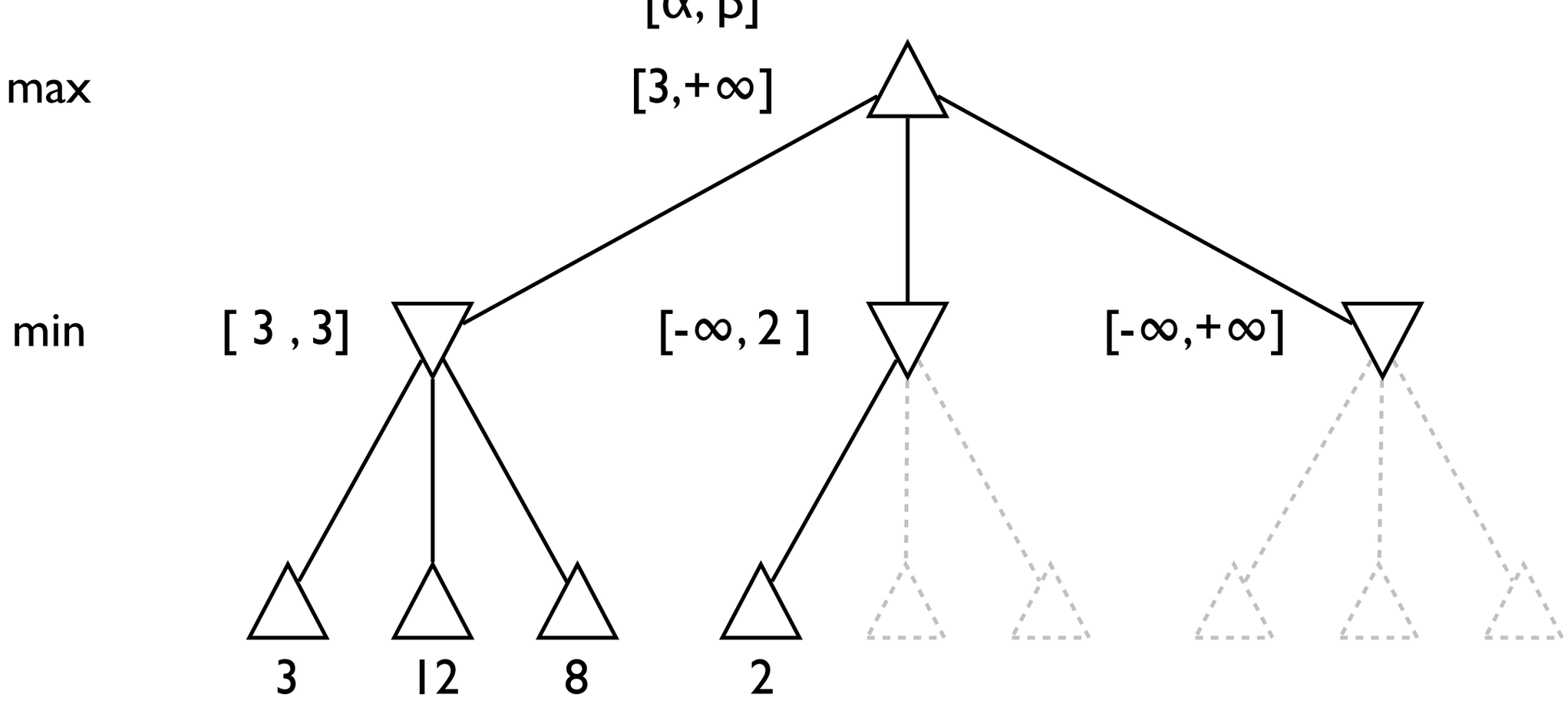


Alpha-beta pruning

- Idea: prune branches that can't influence final decision
- alpha-beta pruning keeps track of the range of possible values:

  α = max's best choice (highest val.) so far at any point along path
  β = min's best choice (lowest val.) so far at any point along path

- alpha-beta doesn't search: it "*prunes*" branches that don't affect current estimates

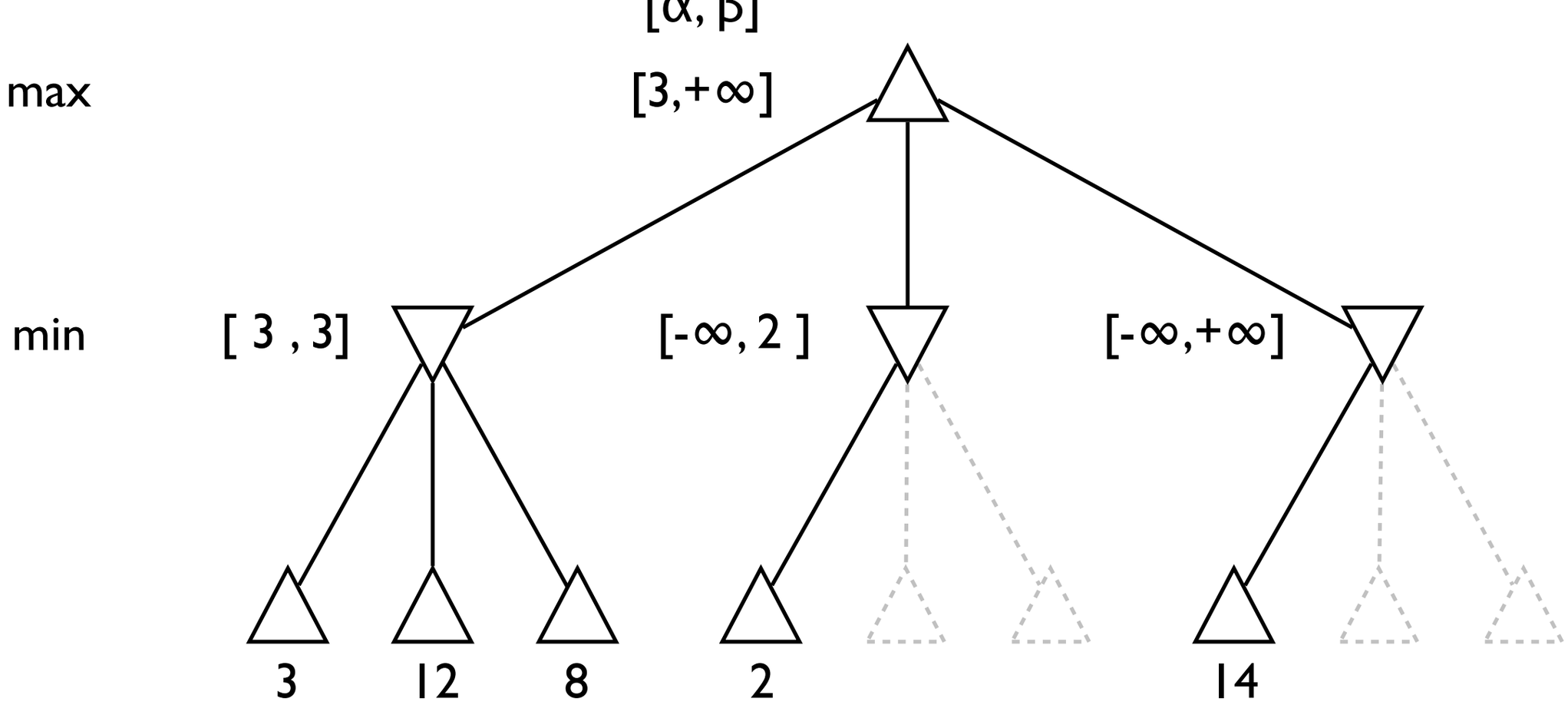


Alpha-beta pruning

- Idea: prune branches that can't influence final decision
- alpha-beta pruning keeps track of the range of possible values:

  α = max's best choice (highest val.) so far at any point along path
  β = min's best choice (lowest val.) so far at any point along path

- alpha-beta doesn't search: it "*prunes*" branches that don't affect current estimates

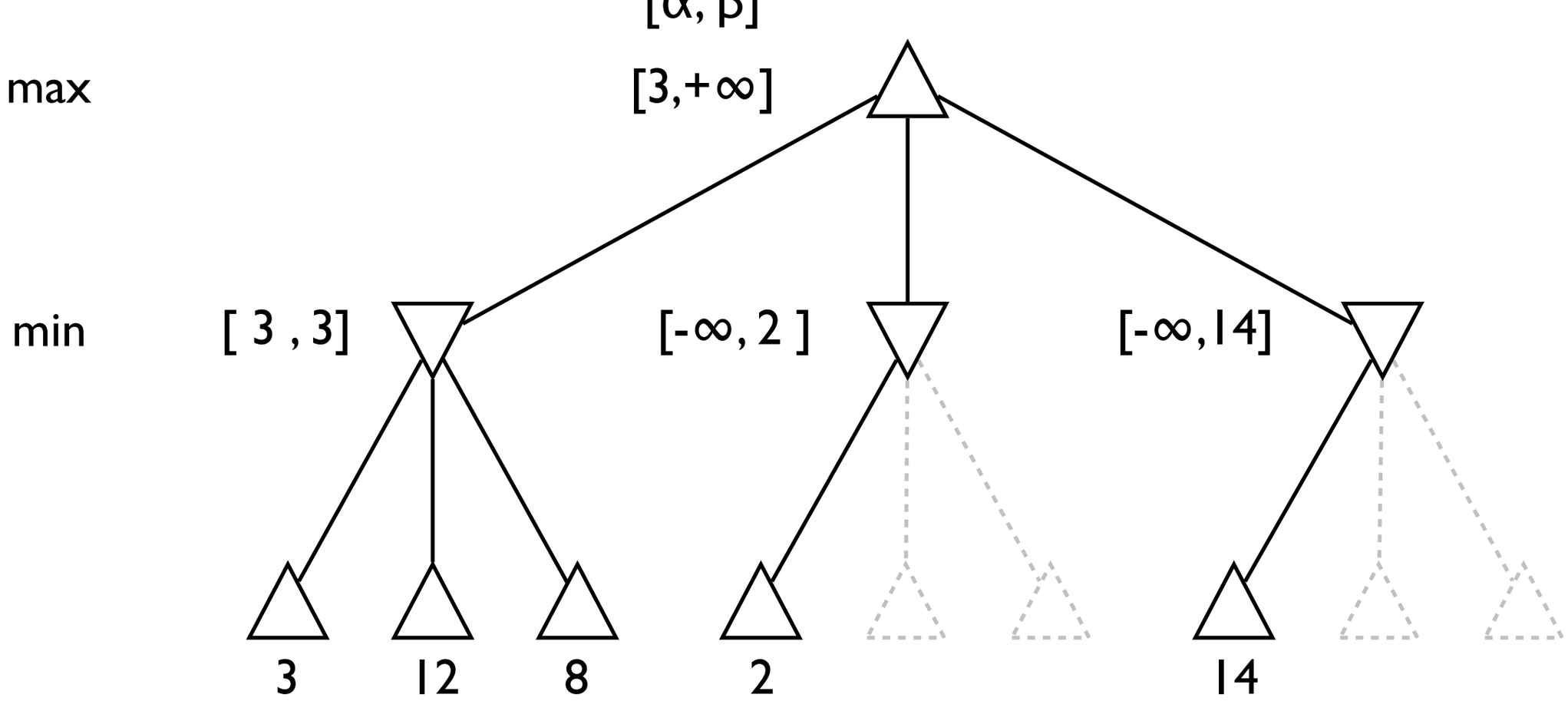


Alpha-beta pruning

- Idea: prune branches that can't influence final decision
- alpha-beta pruning keeps track of the range of possible values:

  α = max's best choice (highest val.) so far at any point along path
  β = min's best choice (lowest val.) so far at any point along path

- alpha-beta doesn't search: it "*prunes*" branches that don't affect current estimates

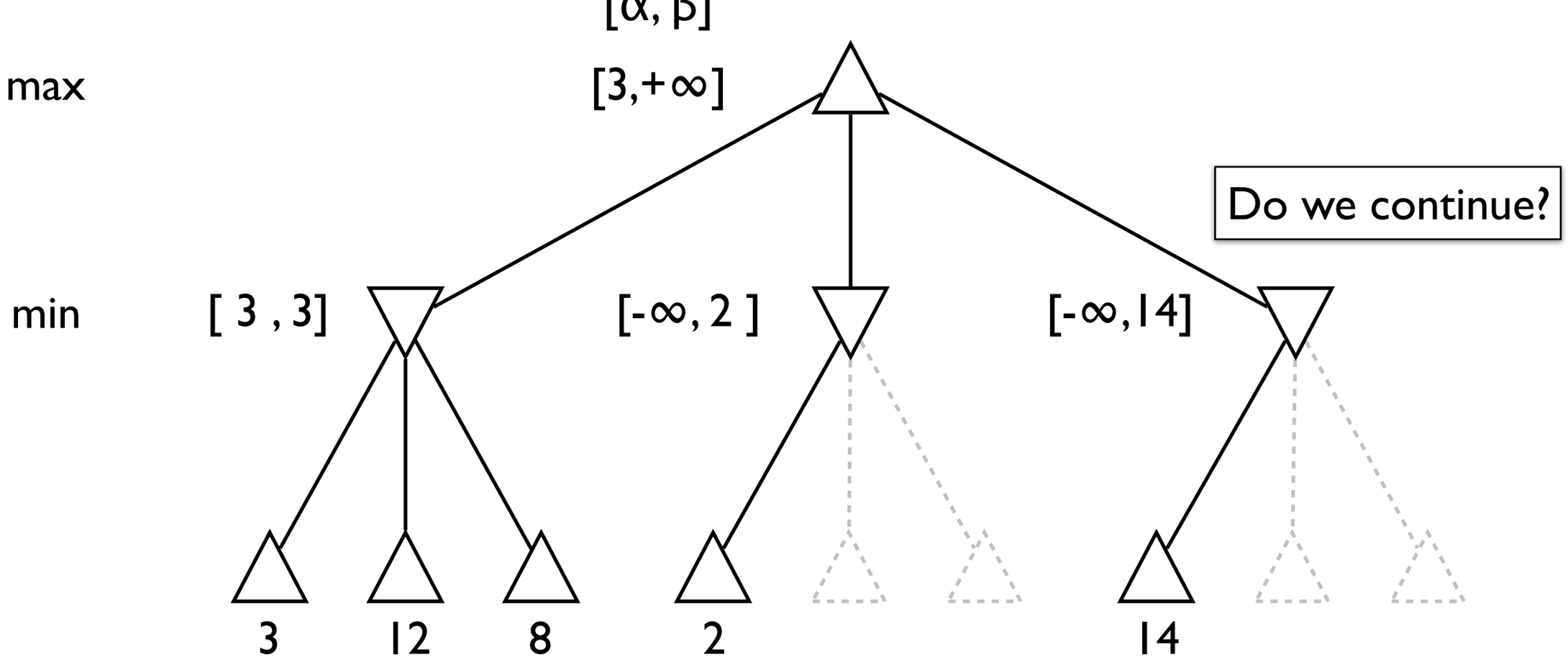


Alpha-beta pruning

- Idea: prune branches that can't influence final decision
- alpha-beta pruning keeps track of the range of possible values:

  α = max's best choice (highest val.) so far at any point along path
  β = min's best choice (lowest val.) so far at any point along path

- alpha-beta doesn't search: it "*prunes*" branches that don't affect current estimates

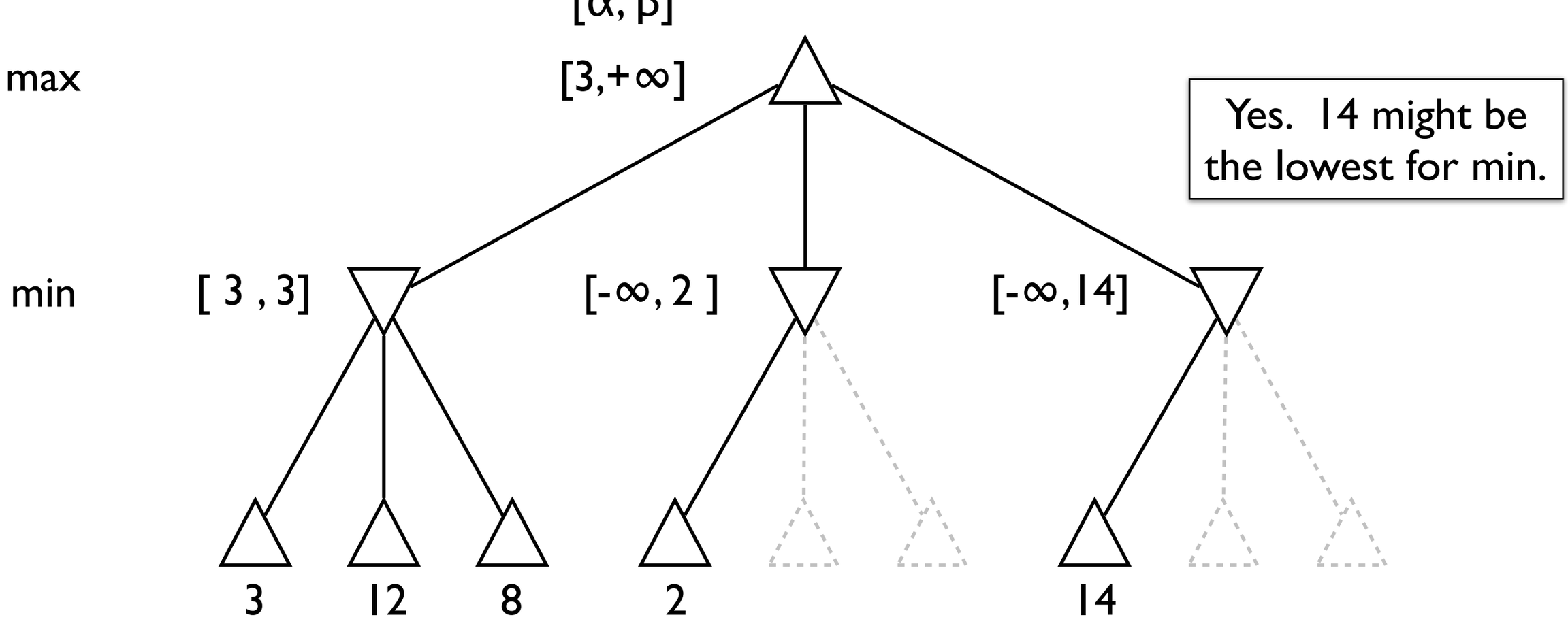


Alpha-beta pruning

- Idea: prune branches that can't influence final decision
- alpha-beta pruning keeps track of the range of possible values:

  α = max's best choice (highest val.) so far at any point along path
  β = min's best choice (lowest val.) so far at any point along path

- alpha-beta doesn't search: it "*prunes*" branches that don't affect current estimates

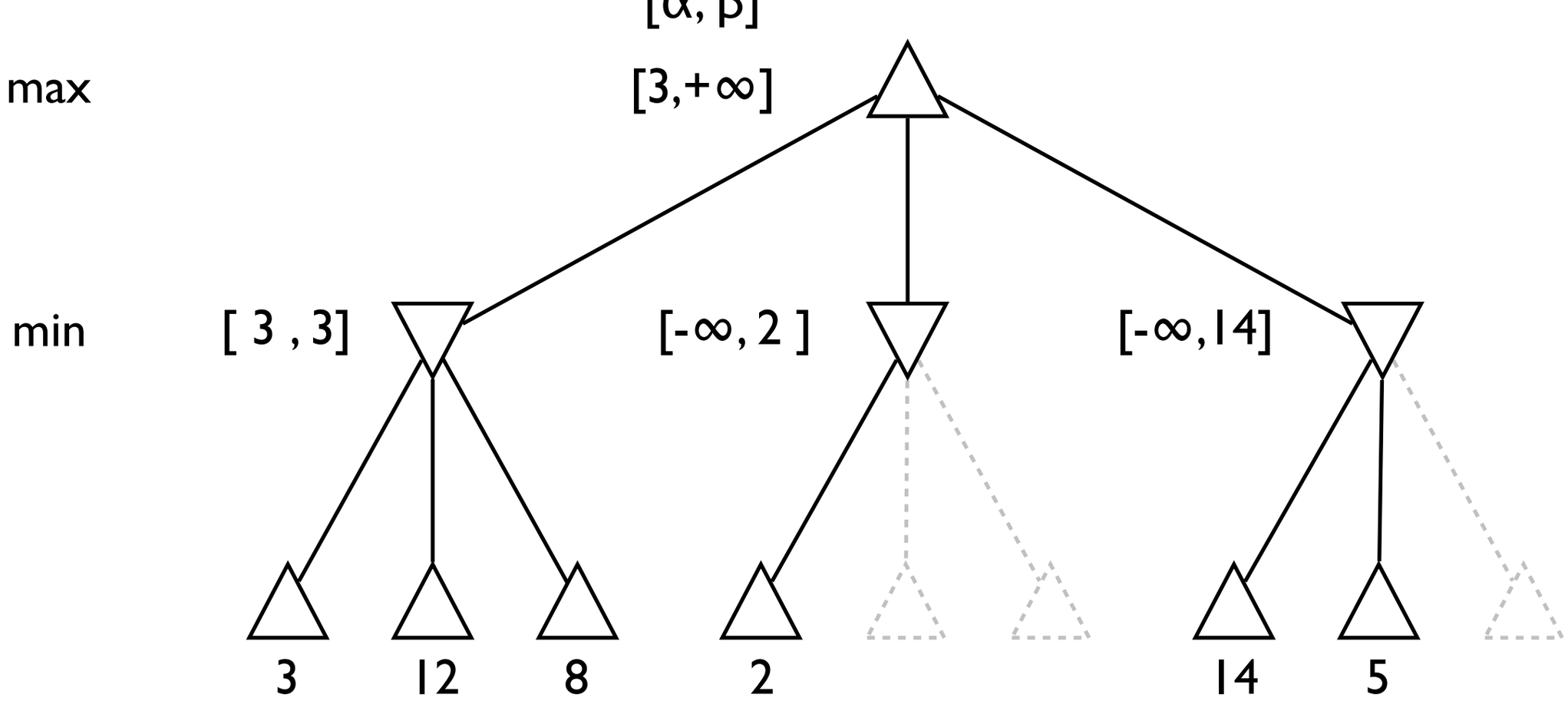


Alpha-beta pruning

- Idea: prune branches that can't influence final decision
- alpha-beta pruning keeps track of the range of possible values:

  α = max's best choice (highest val.) so far at any point along path
  β = min's best choice (lowest val.) so far at any point along path

- alpha-beta doesn't search: it "*prunes*" branches that don't affect current estimates

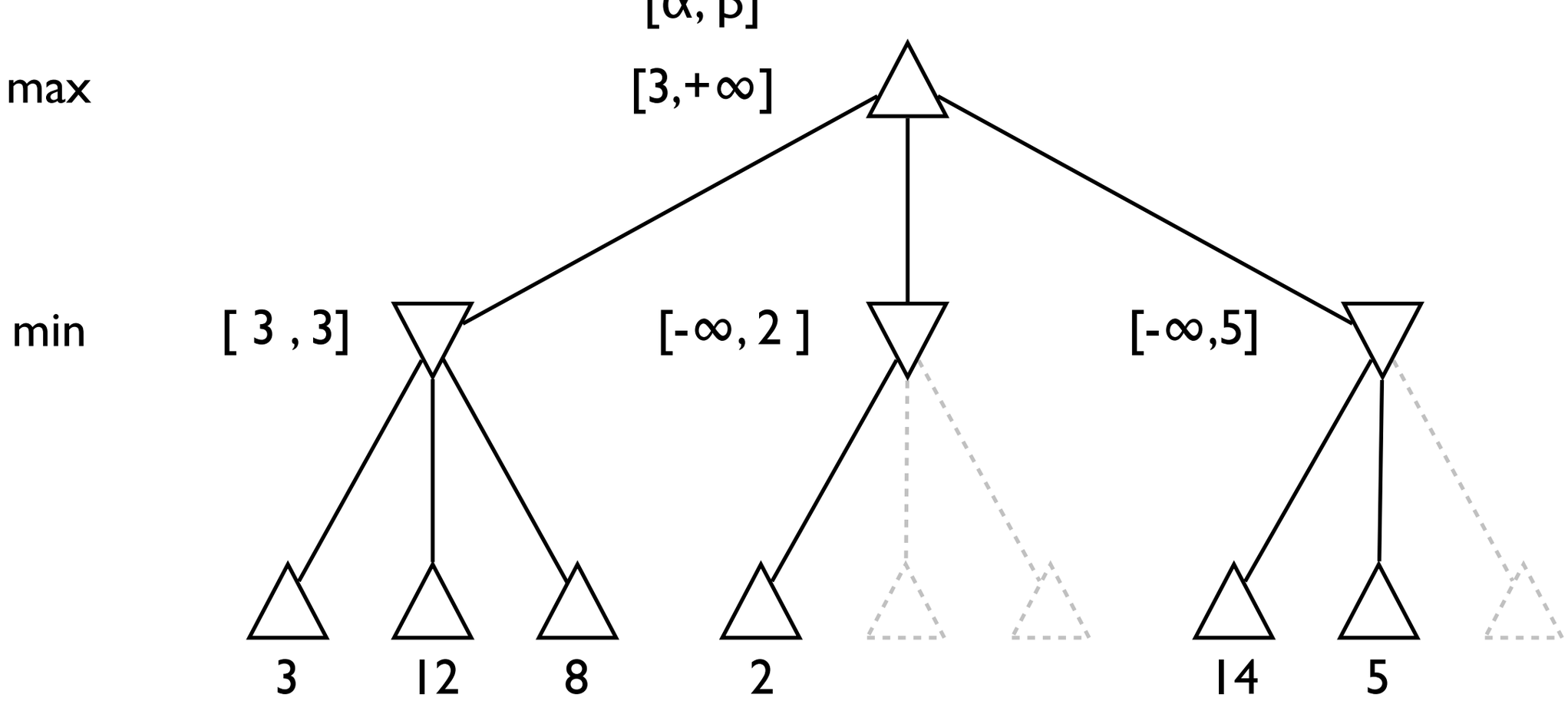


Alpha-beta pruning

- Idea: prune branches that can't influence final decision
- alpha-beta pruning keeps track of the range of possible values:

  α = max's best choice (highest val.) so far at any point along path
  β = min's best choice (lowest val.) so far at any point along path

- alpha-beta doesn't search: it "*prunes*" branches that don't affect current estimates

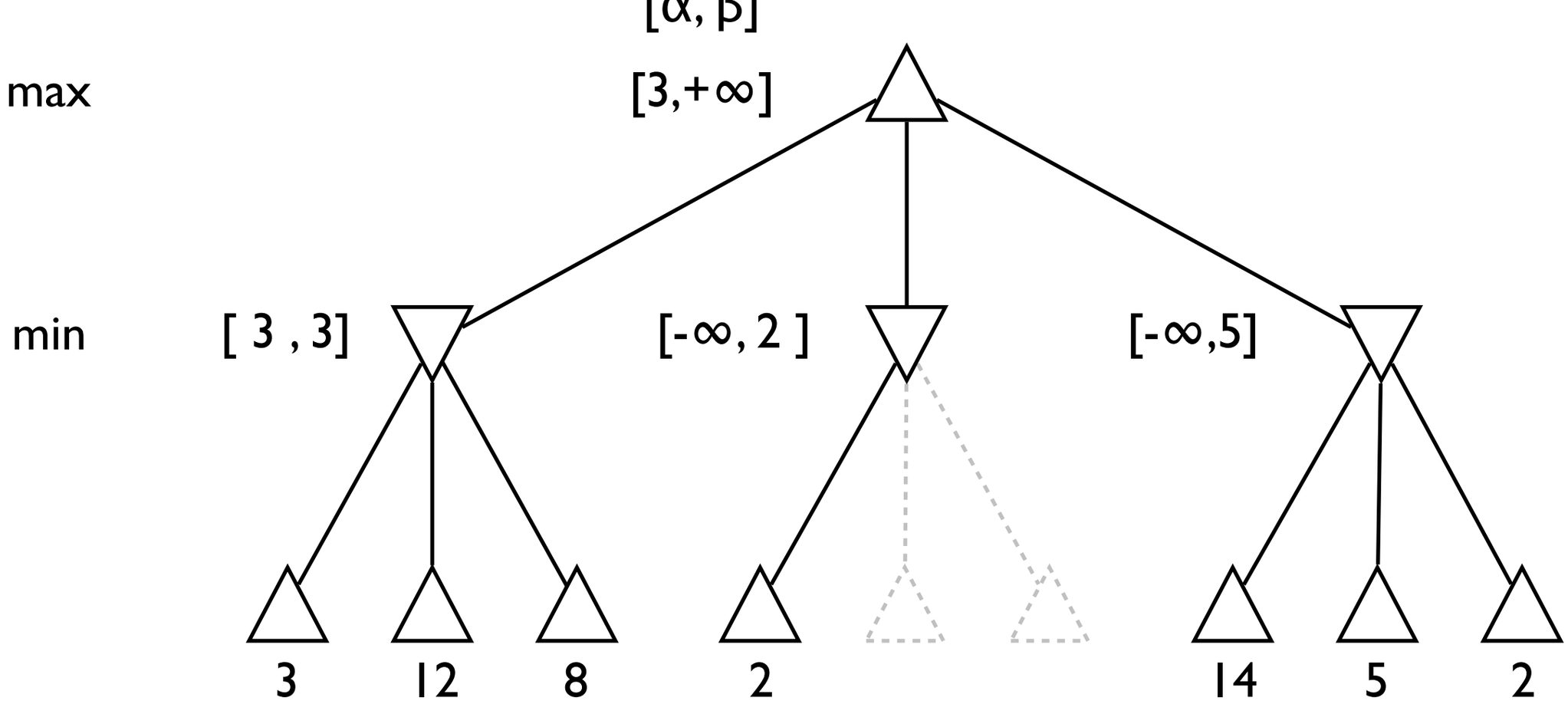


Alpha-beta pruning

- Idea: prune branches that can't influence final decision
- alpha-beta pruning keeps track of the range of possible values:

  α = max's best choice (highest val.) so far at any point along path
  β = min's best choice (lowest val.) so far at any point along path

- alpha-beta doesn't search: it "*prunes*" branches that don't affect current estimates

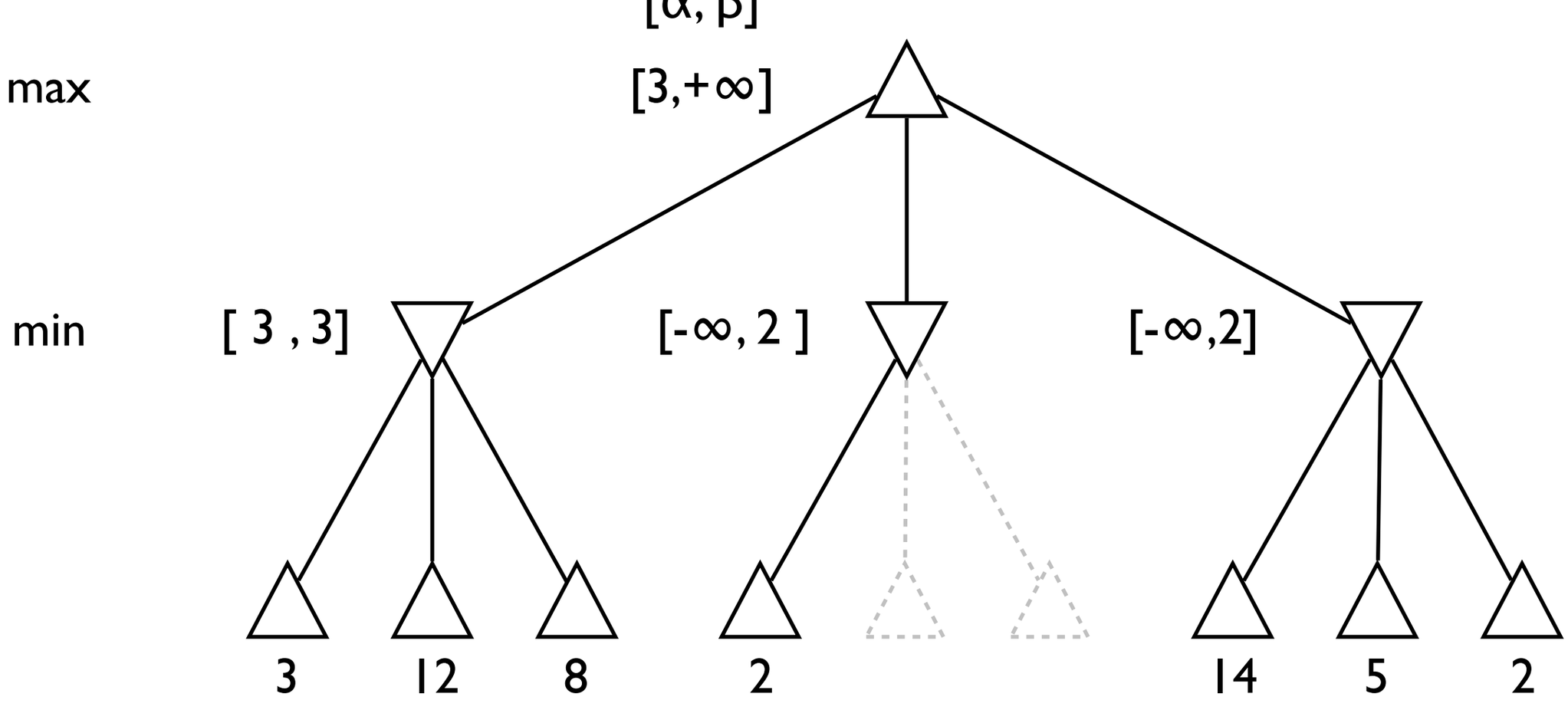


Alpha-beta pruning

- Idea: prune branches that can't influence final decision
- alpha-beta pruning keeps track of the range of possible values:

  α = max's best choice (highest val.) so far at any point along path
  β = min's best choice (lowest val.) so far at any point along path

- alpha-beta doesn't search: it "*prunes*" branches that don't affect current estimates

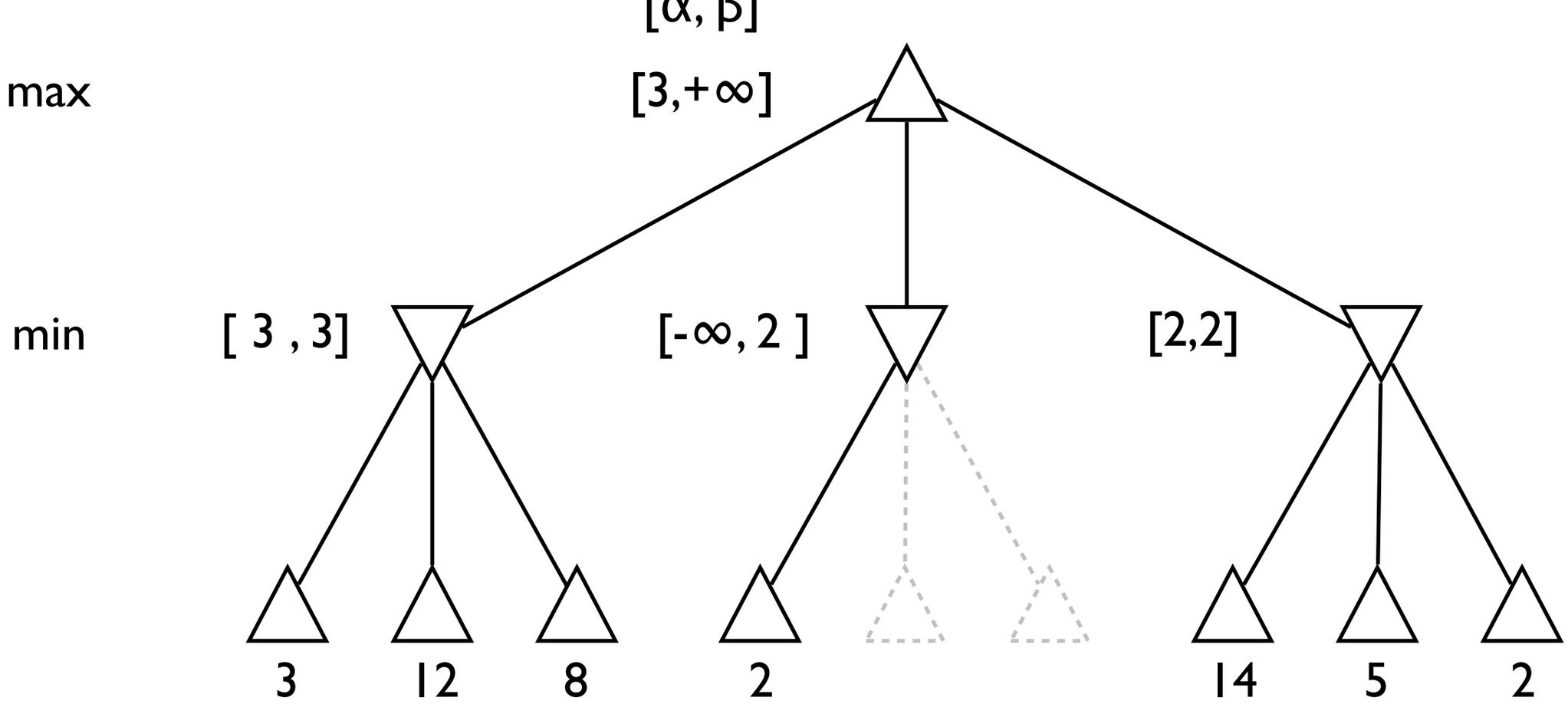


Alpha-beta pruning

- Idea: prune branches that can't influence final decision
- alpha-beta pruning keeps track of the range of possible values:

  α = max's best choice (highest val.) so far at any point along path
  β = min's best choice (lowest val.) so far at any point along path

- alpha-beta doesn't search: it "*prunes*" branches that don't affect current estimates

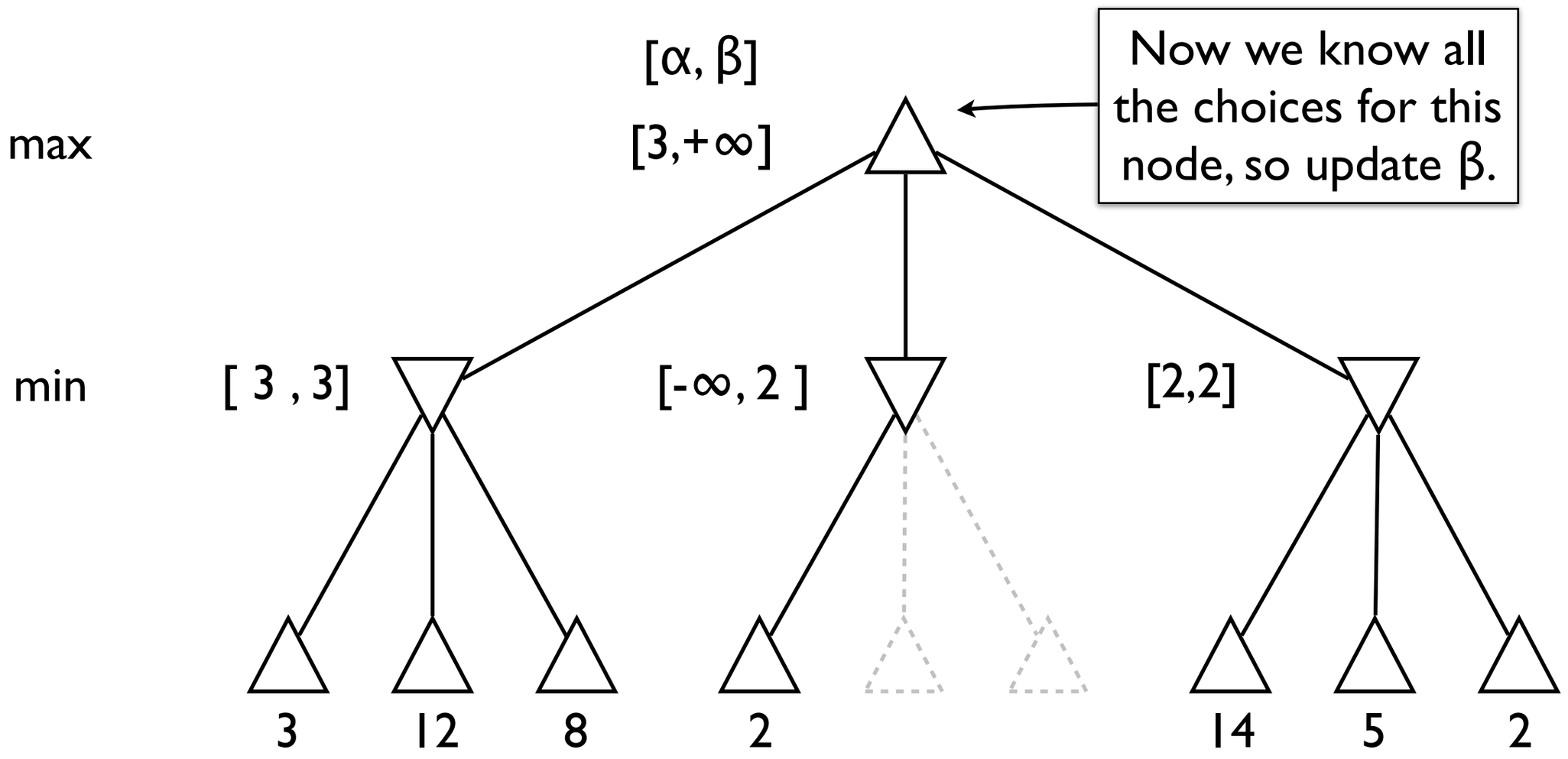


Alpha-beta pruning

- Idea: prune branches that can't influence final decision
- alpha-beta pruning keeps track of the range of possible values:

  α = max's best choice (highest val.) so far at any point along path
  β = min's best choice (lowest val.) so far at any point along path

- alpha-beta doesn't search: it "*prunes*" branches that don't affect current estimates

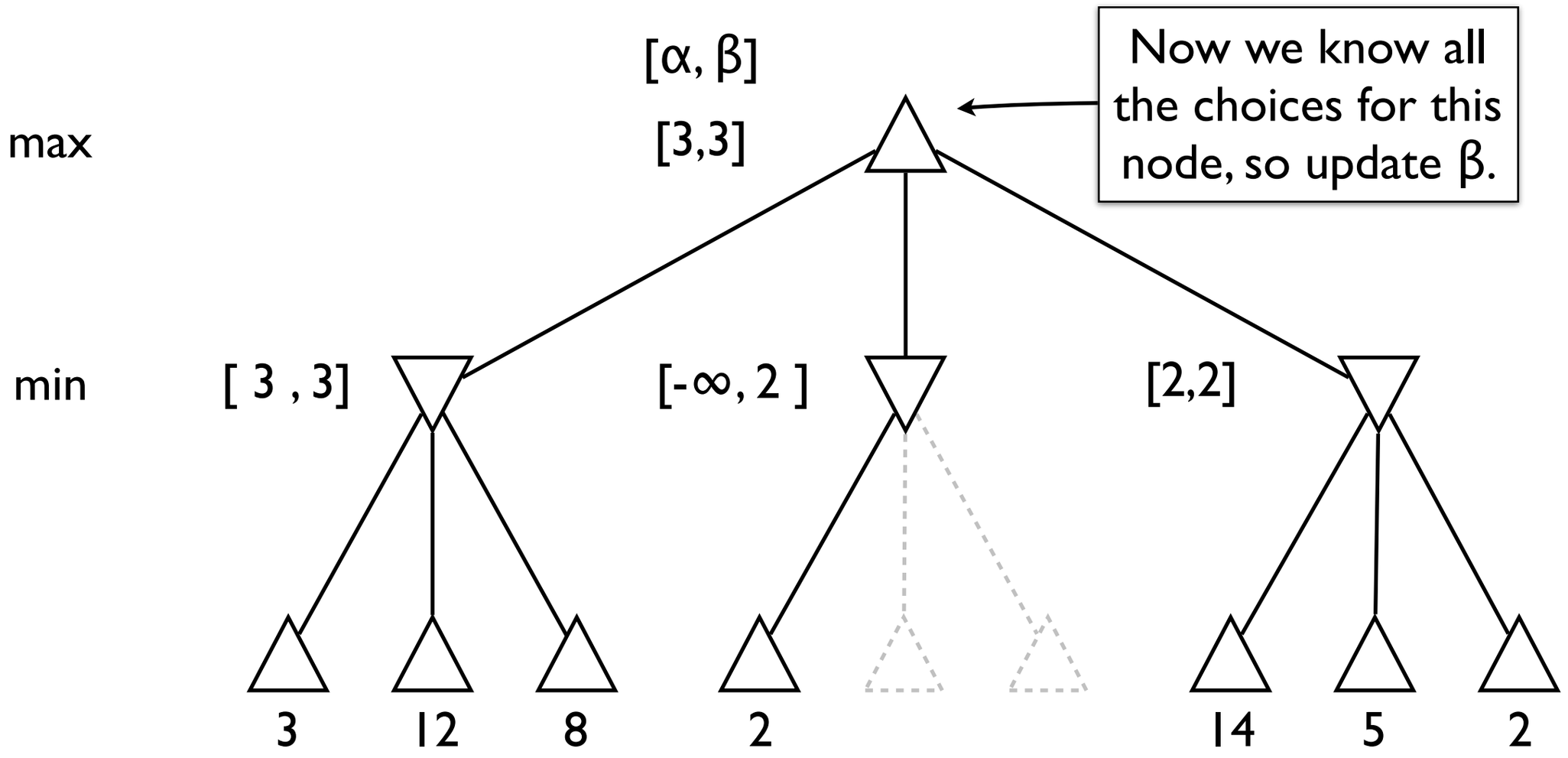


Alpha-beta pruning

- Idea: prune branches that can't influence final decision
- alpha-beta pruning keeps track of the range of possible values:

  α = max's best choice (highest val.) so far at any point along path
  β = min's best choice (lowest val.) so far at any point along path

- alpha-beta doesn't search: it "*prunes*" branches that don't affect current estimates

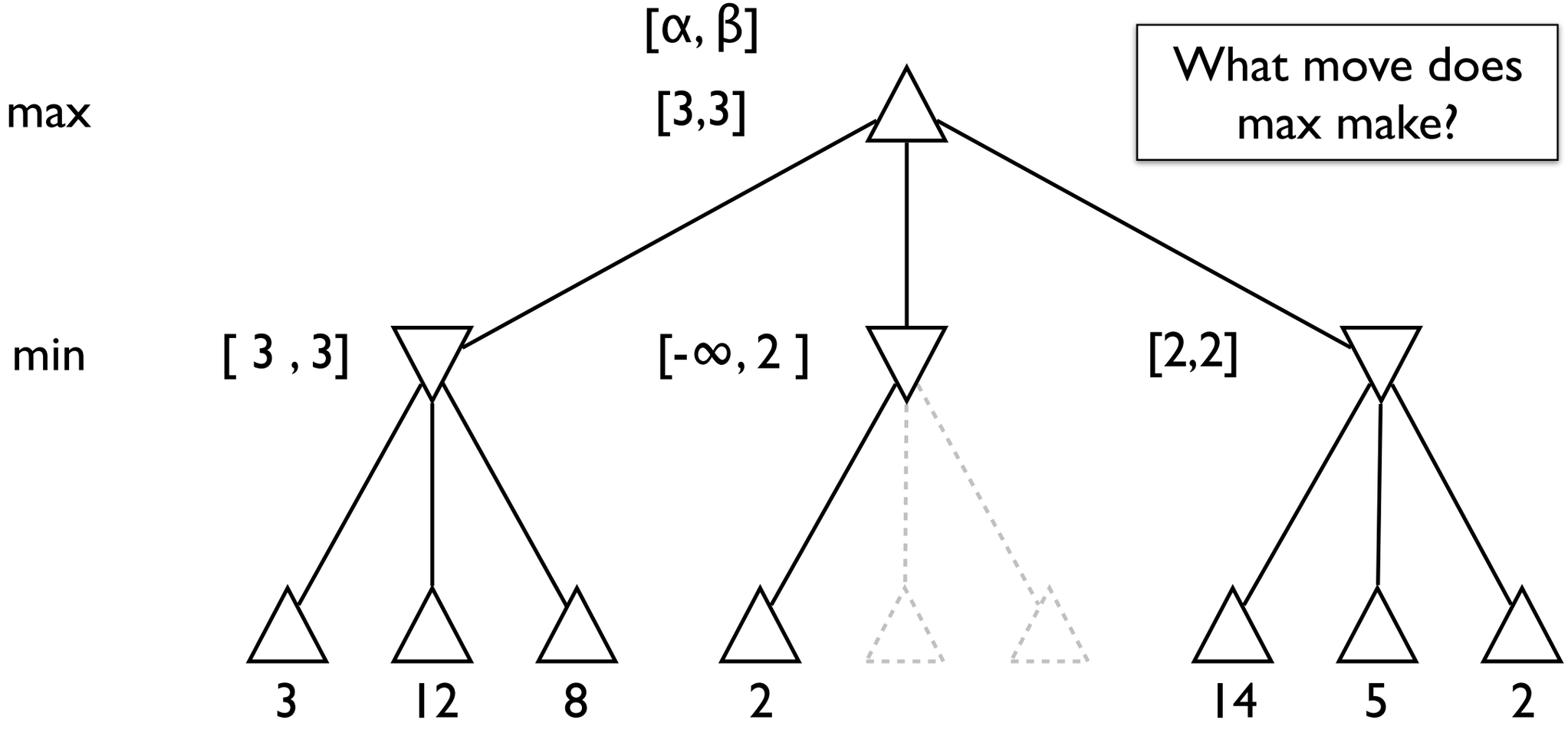


Alpha-beta pruning

- Idea: prune branches that can't influence final decision
- alpha-beta pruning keeps track of the range of possible values:

  α = max's best choice (highest val.) so far at any point along path
  β = min's best choice (lowest val.) so far at any point along path

- alpha-beta doesn't search: it "*prunes*" branches that don't affect current estimates



Alpha-beta pruning

- Idea: prune branches that can't influence final decision
- alpha-beta pruning keeps track of the range of possible values:

  α = max's best choice (highest val.) so far at any point along path
  β = min's best choice (lowest val.) so far at any point along path

- alpha-beta doesn't search: it "*prunes*" branches that don't affect current estimates