

CPS 270: Artificial Intelligence

<http://www.cs.duke.edu/courses/fall08/cps270/>

Two-player, zero-sum, perfect-information

Games

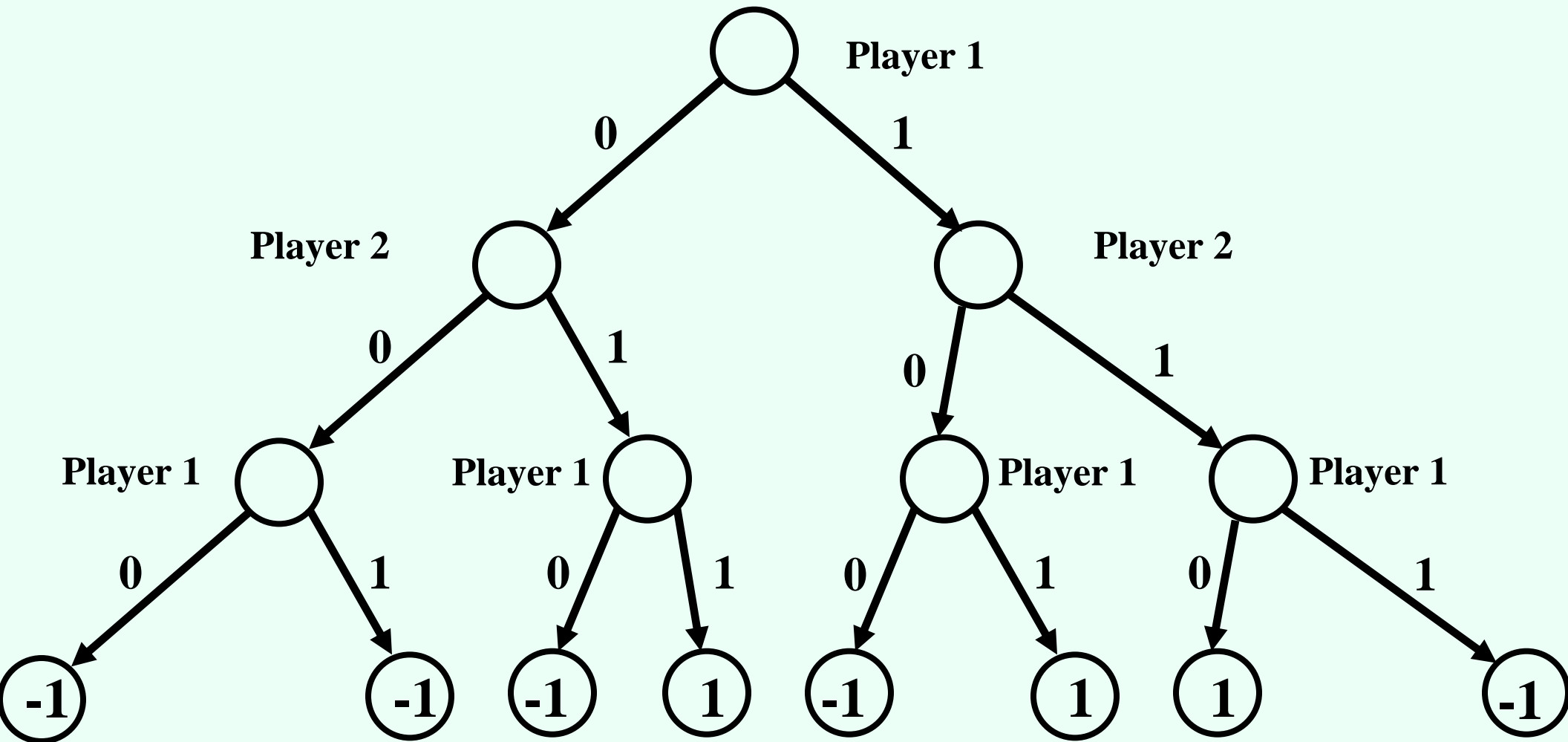
Instructor: Vincent Conitzer

Game playing

- Rich tradition of creating game-playing programs in AI
- Many similarities to search
- Most of the games studied
 - have two players,
 - are **zero-sum**: what one player wins, the other loses
 - have **perfect information**: the entire state of the game is known to both players at all times
- E.g., tic-tac-toe, checkers, chess, Go, backgammon, ...
- Will focus on these for now
- Recently more interest in other games
 - Esp. games without perfect information; e.g., poker
 - Need probability theory, game theory for such games

“Sum to 2” game

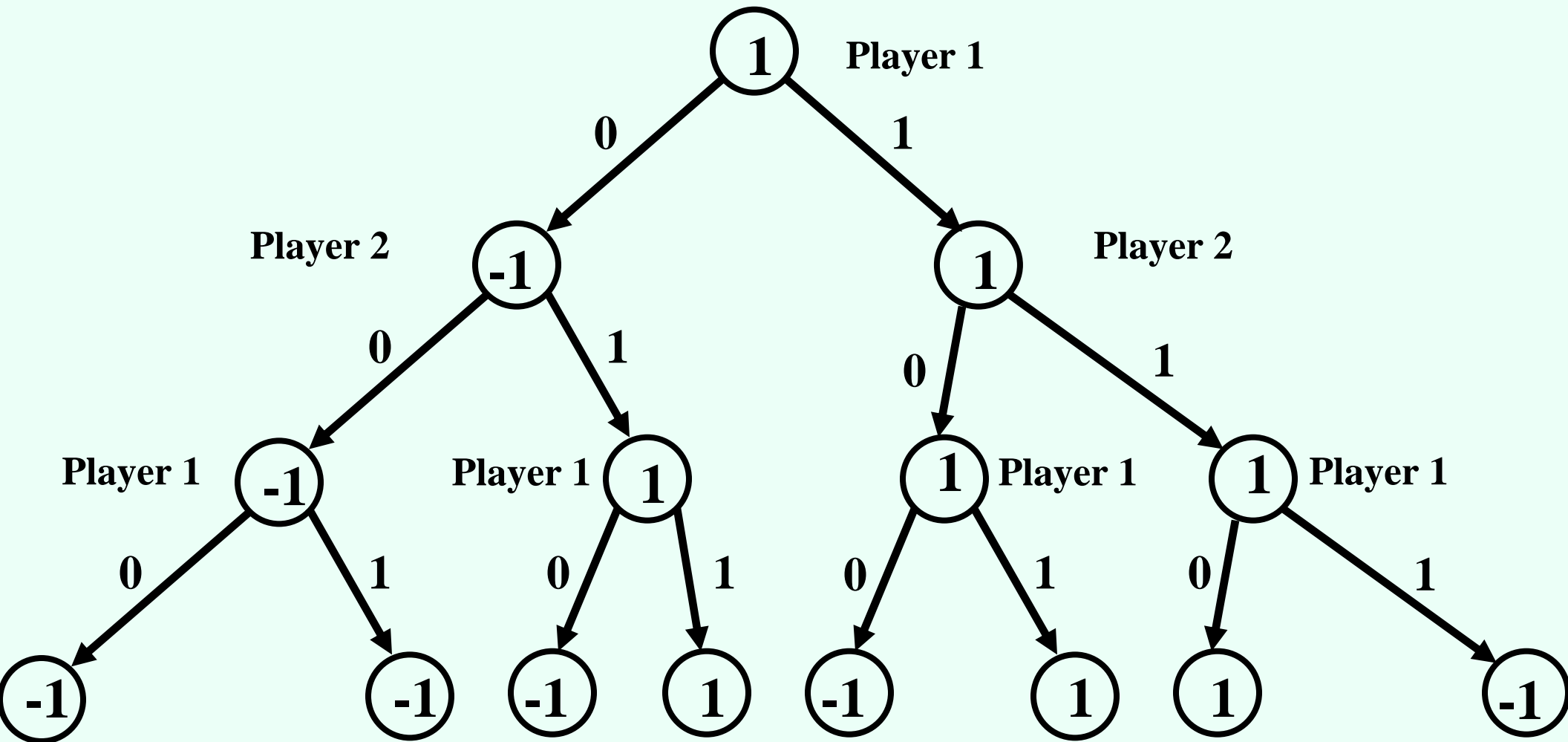
- Player 1 moves, then player 2, finally player 1 again
- Move = 0 or 1
- Player 1 wins if and only if all moves together sum to 2



Player 1's utility is in the leaves; player 2's utility is the negative of this

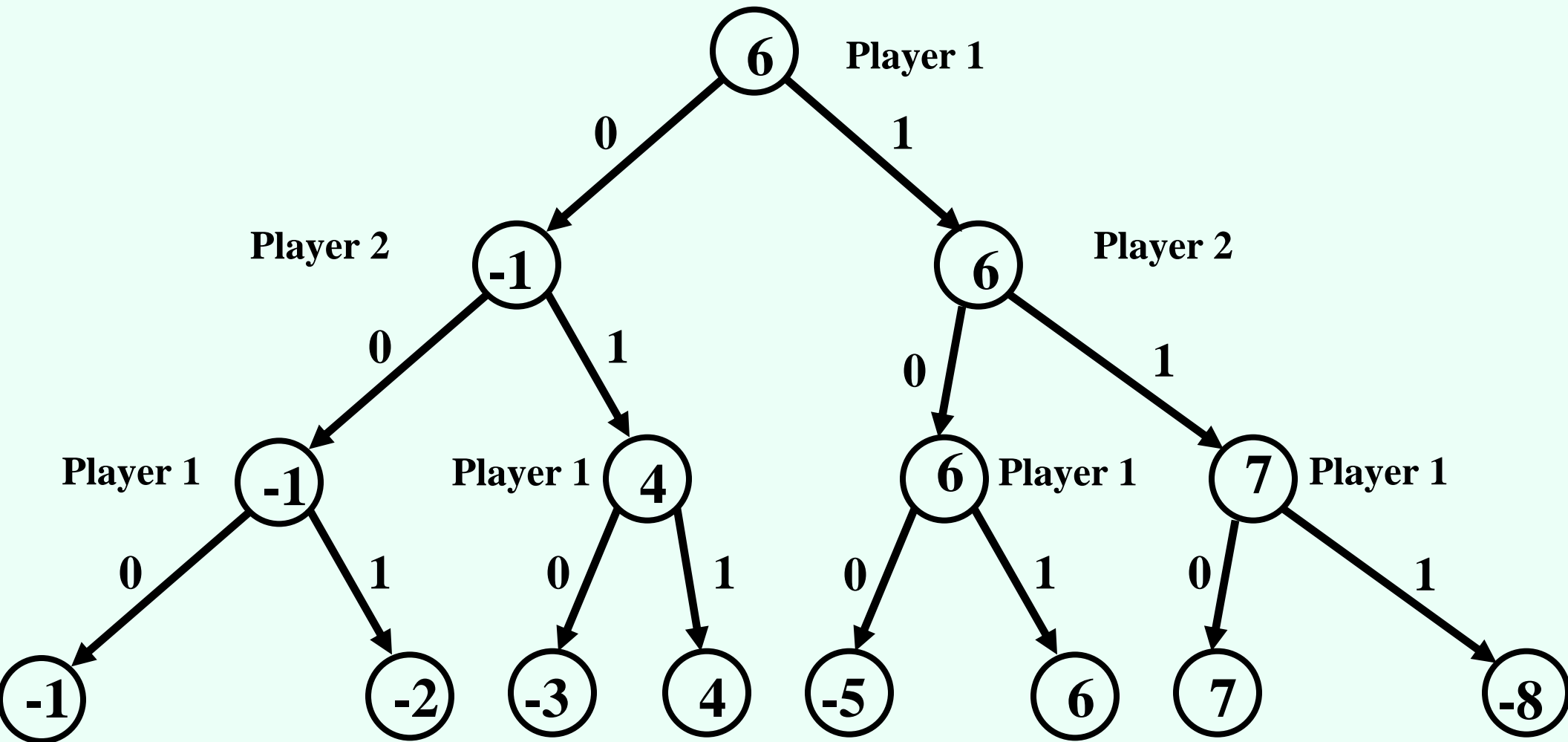
Backward induction (aka. minimax)

- From leaves upward, analyze best decision for player at node, give node a value
 - Once we know values, easy to find optimal action (choose best value)



Modified game

- From leaves upward, analyze best decision for player at node, give node a value



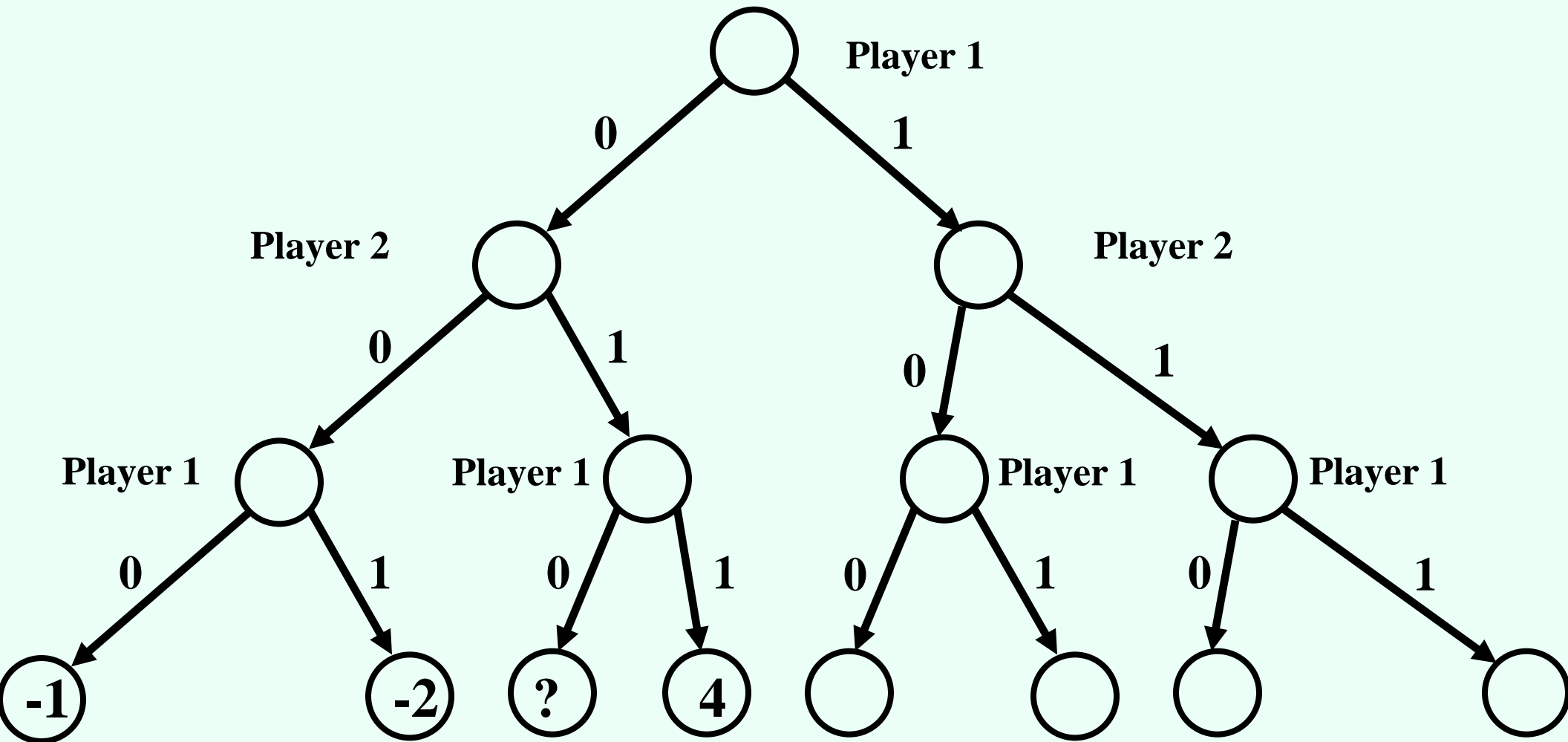
A recursive implementation

- **Value(state)**
- If state is terminal, return its value
- If (player(state) = player 1)
 - $v := -\text{infinity}$
 - For each action
 - $v := \max(v, \text{Value}(\text{successor}(\text{state}, \text{action})))$
 - Return v
- Else
 - $v := \text{infinity}$
 - For each action
 - $v := \min(v, \text{Value}(\text{successor}(\text{state}, \text{action})))$
 - Return v

Space? Time?

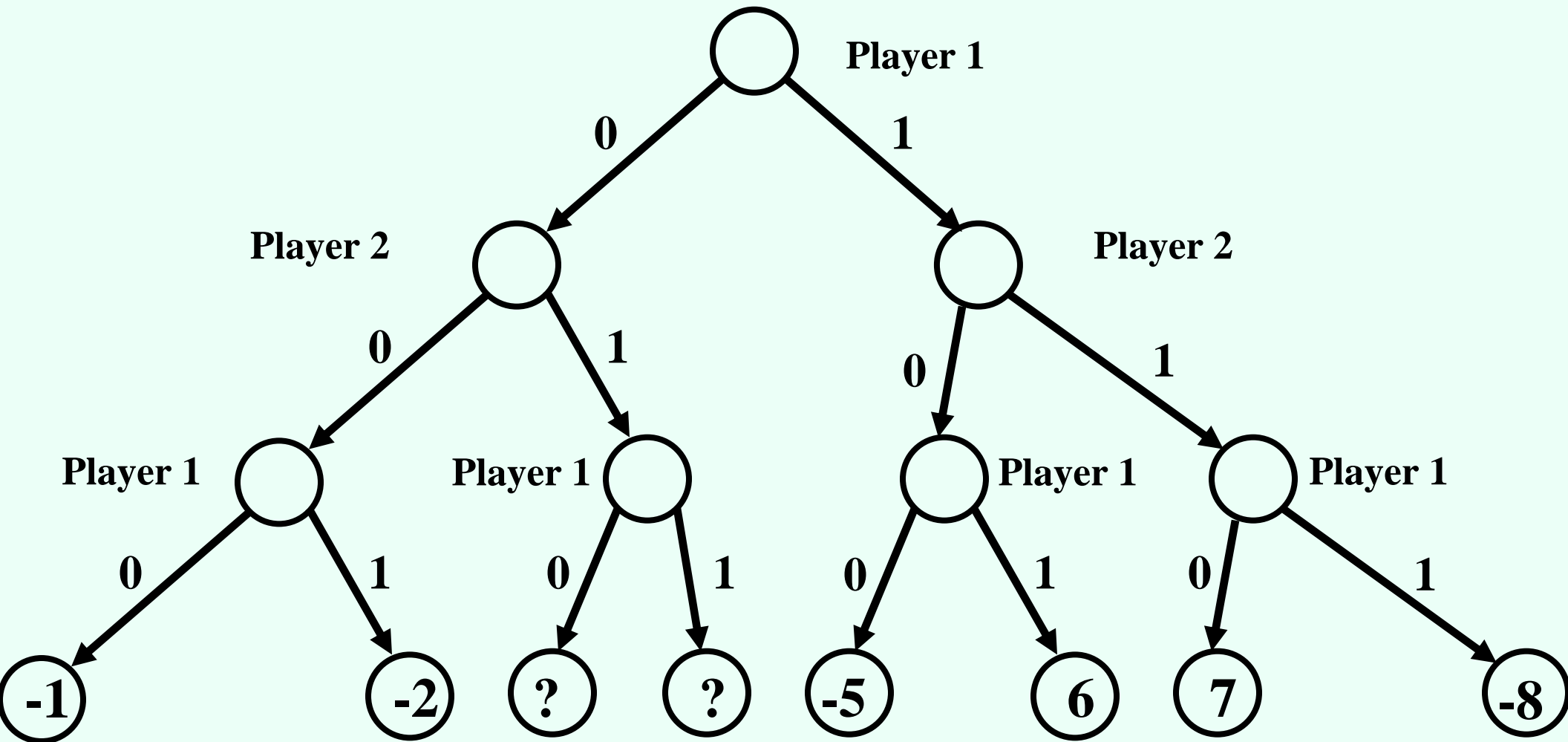
Do we need to see all the leaves?

- Do we need to see the value of the question mark here?



Do we need to see all the leaves?

- Do we need to see the values of the question marks here?

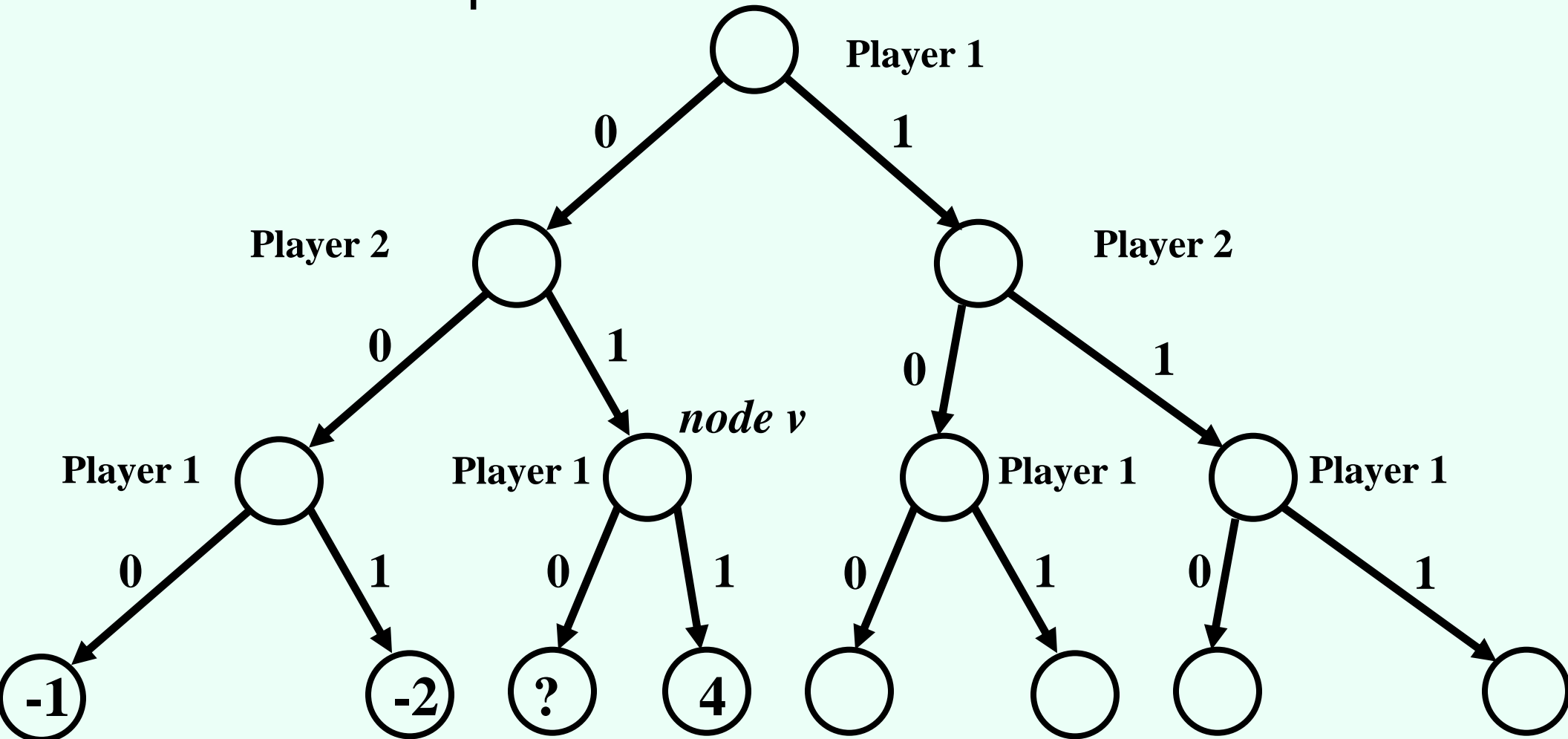


Alpha-beta pruning

- **Pruning** = cutting off parts of the search tree (because you realize you don't need to look at them)
 - When we considered A^* we also pruned large parts of the search tree
- Maintain **alpha** = value of the best option for player 1 along the path so far
- **Beta** = value of the best option for player 2 along the path so far

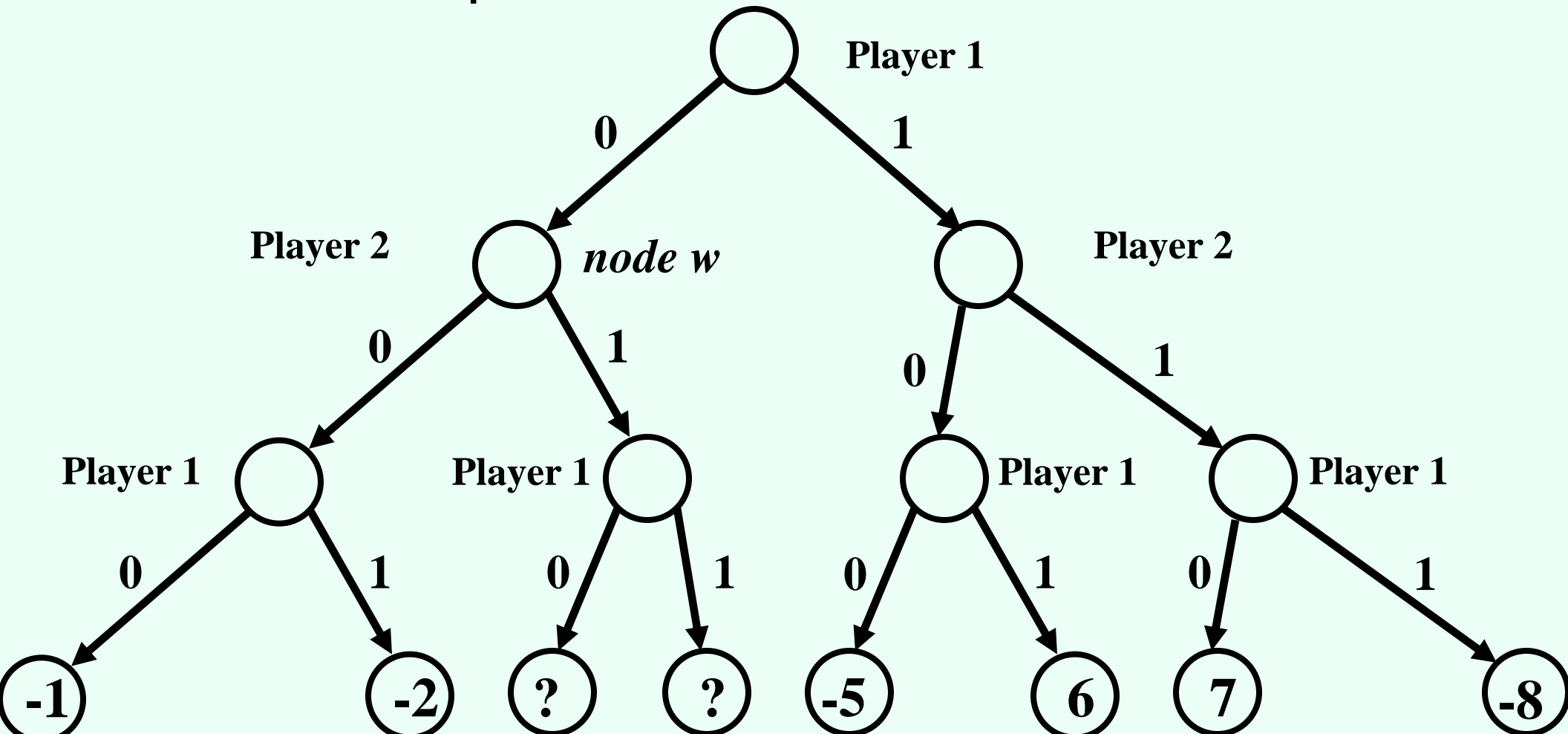
Pruning on beta

- Beta at node v is -1
- We know the value of node v is going to be at least 4 , so the -1 route will be preferred
- No need to explore this node further



Pruning on alpha

- Alpha at node w is 6
- We know the value of node w is going to be at most -1, so the 6 route will be preferred
- No need to explore this node further



Modifying recursive implementation to do alpha-beta pruning

- **Value(state, alpha, beta)**
- If state is terminal, return its value
- If (player(state) = player 1)
 - $v := -\text{infinity}$
 - For each action
 - $v := \max(v, \text{Value}(\text{successor}(\text{state}, \text{action}), \alpha, \beta))$
 - If $v \geq \beta$, return v
 - $\alpha := \max(\alpha, v)$
 - Return v
- Else
 - $v := \text{infinity}$
 - For each action
 - $v := \min(v, \text{Value}(\text{successor}(\text{state}, \text{action}), \alpha, \beta))$
 - If $v \leq \alpha$, return v
 - $\beta := \min(\beta, v)$
 - Return v

Benefits of alpha-beta pruning

- Without pruning, need to examine $O(b^m)$ nodes
- With pruning, depends on which nodes we consider first
- If we choose a random successor, need to examine $O(b^{3m/4})$ nodes
- If we manage to choose the best successor first, need to examine $O(b^{m/2})$ nodes
 - Practical heuristics for choosing next successor to consider get quite close to this
- Can effectively look twice as deep!
 - Difference between reasonable and expert play

Repeated states

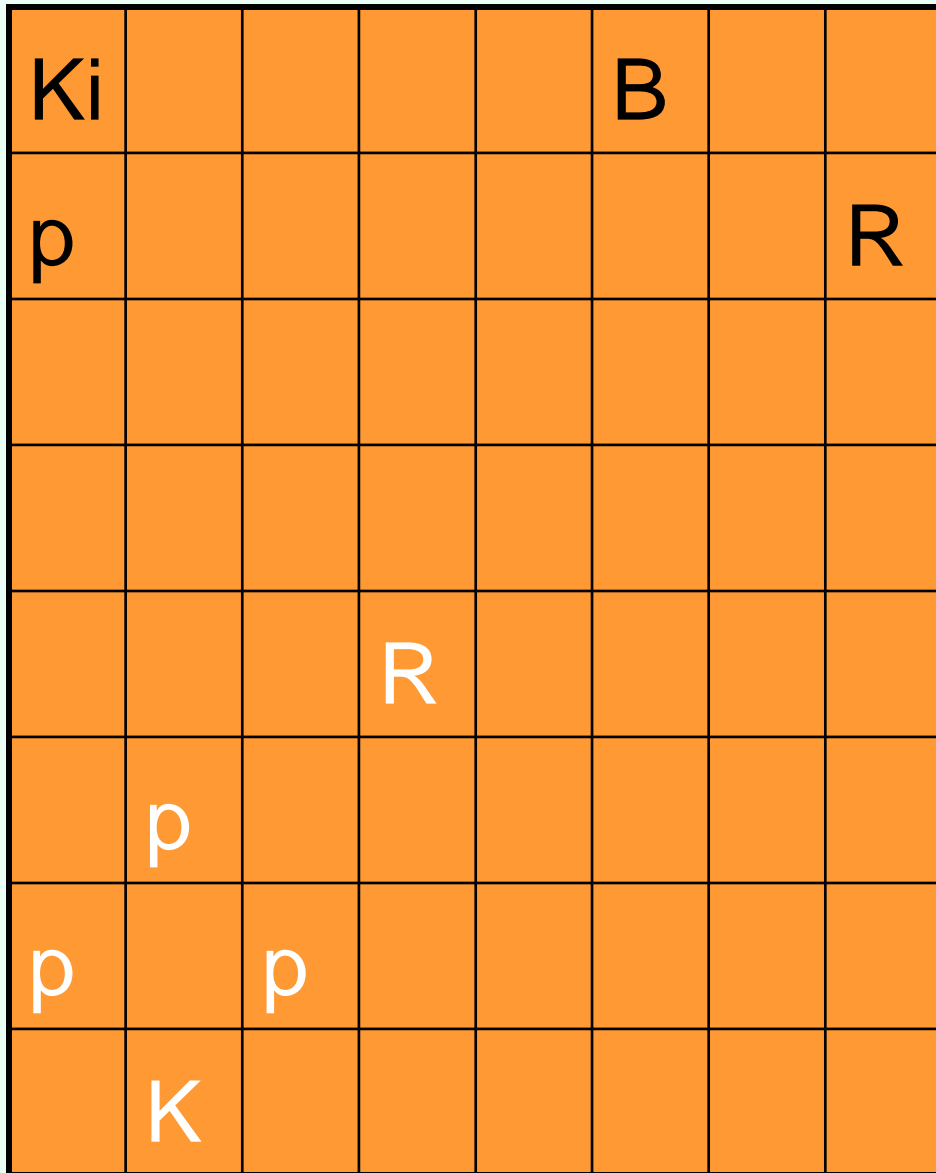
- As in search, multiple sequences of moves may lead to the same state
- Again, can keep track of previously seen states (usually called a **transposition table** in this context)
 - May not want to keep track of **all** previously seen states...

Using evaluation functions

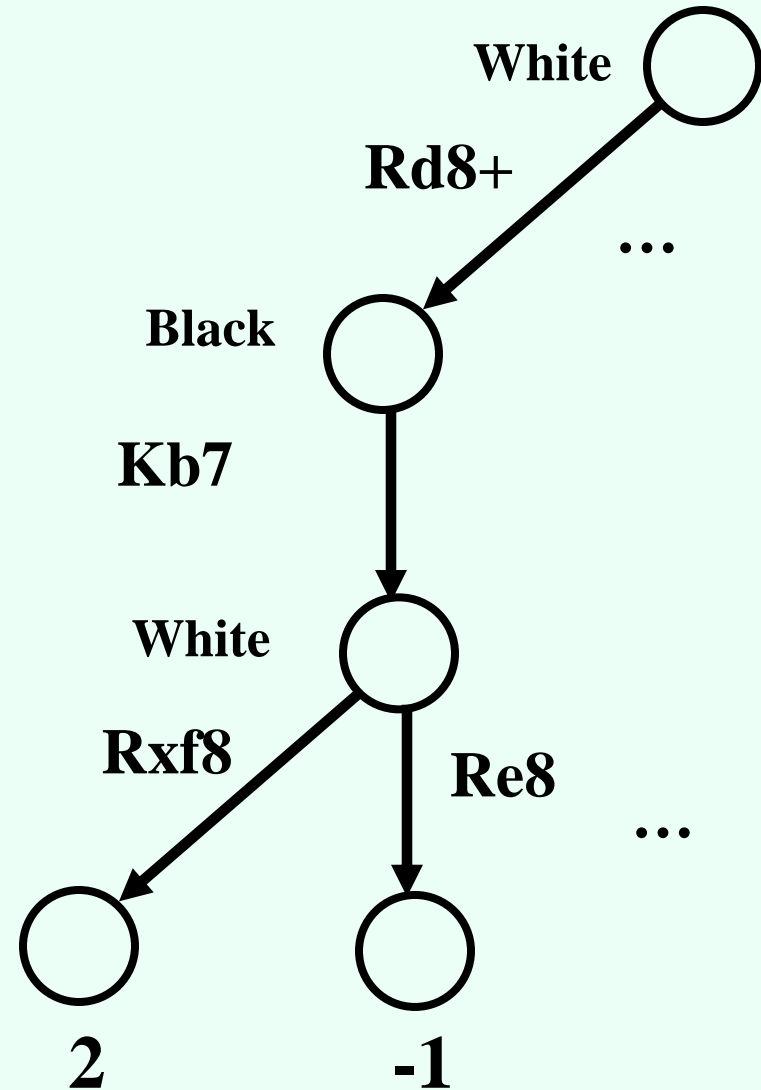
- Most games are too big to solve even with alpha-beta pruning
- Solution: Only look ahead to **limited depth** (nonterminal nodes)
- Evaluate nodes at depth cutoff by a heuristic (aka. **evaluation function**)
- E.g., chess:
 - Material value: queen worth 9 points, rook 5, bishop 3, knight 3, pawn 1
 - Heuristic: difference between players' material values

Chess example

- White to move

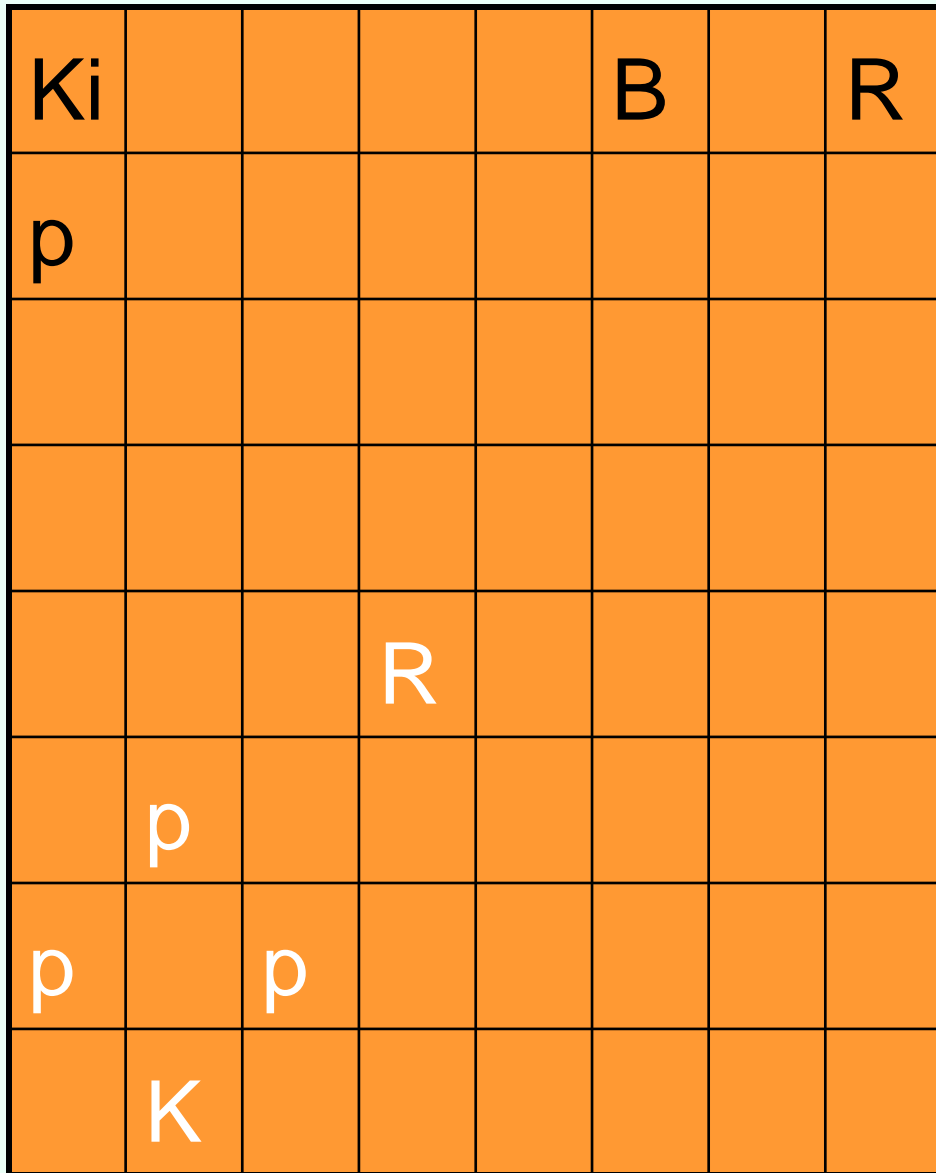


- Depth cutoff: 3 ply
 - Ply = move by one player

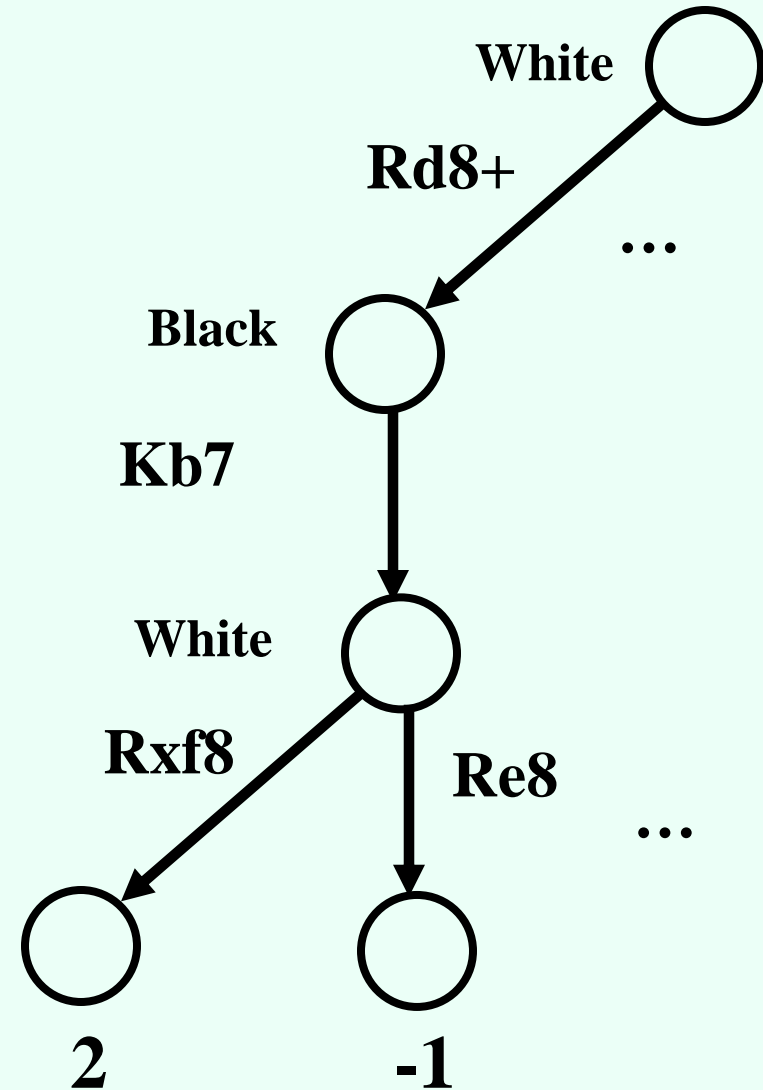


Chess (bad) example

- White to move



- Depth cutoff: 3 ply
 - Ply = move by one player



Depth cutoff obscures fact that white R will be captured

Addressing this problem

- Try to evaluate whether nodes are **quiescent**
 - Quiescent = evaluation function will not change rapidly in near future
 - Only apply evaluation function to quiescent nodes
- If there is an “obvious” move at a state, apply it before applying evaluation function

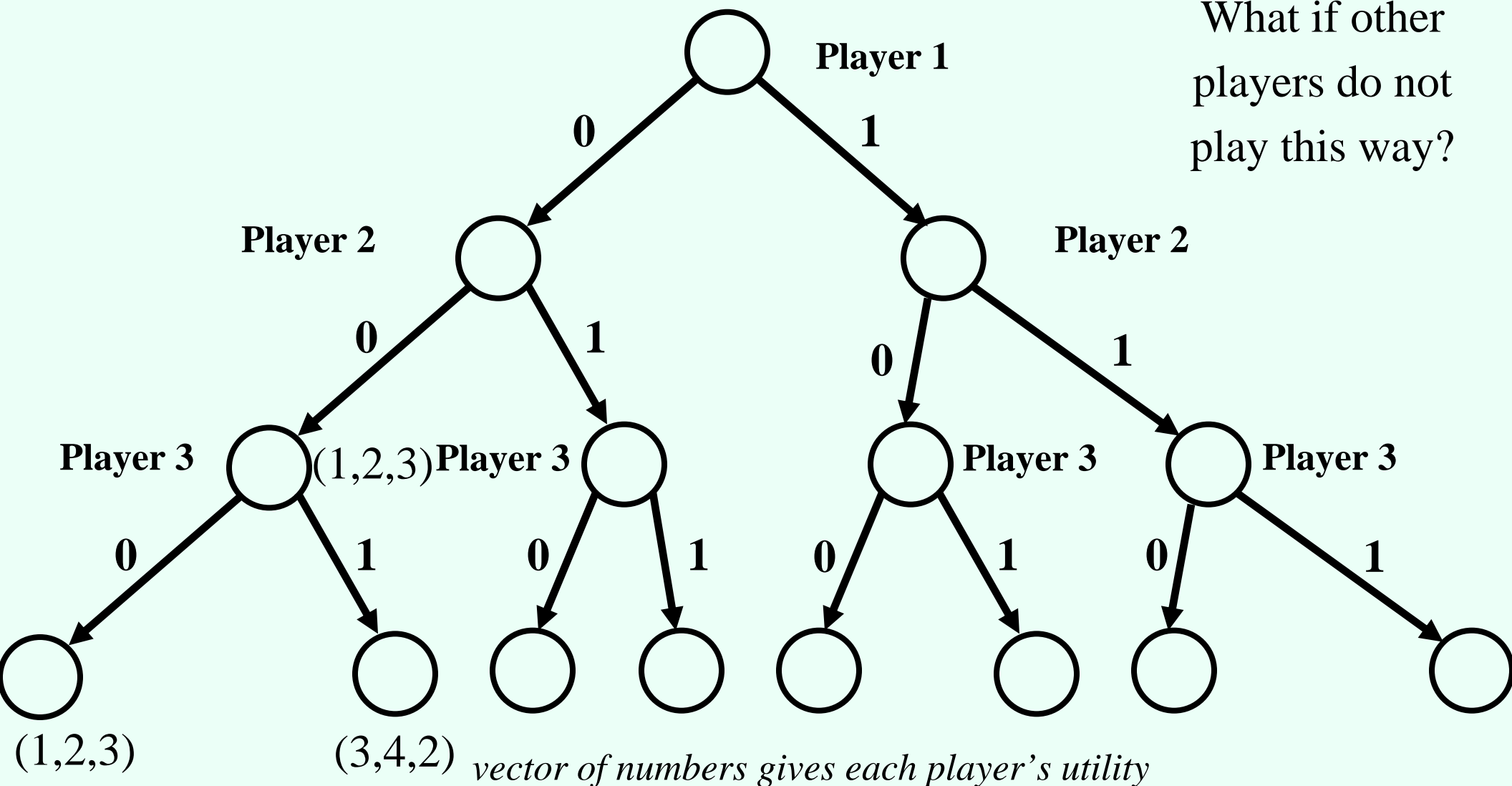
Playing against suboptimal players

- Minimax is optimal against other minimax players
- What about against players that play in some other way?

Many-player, general-sum games of perfect information

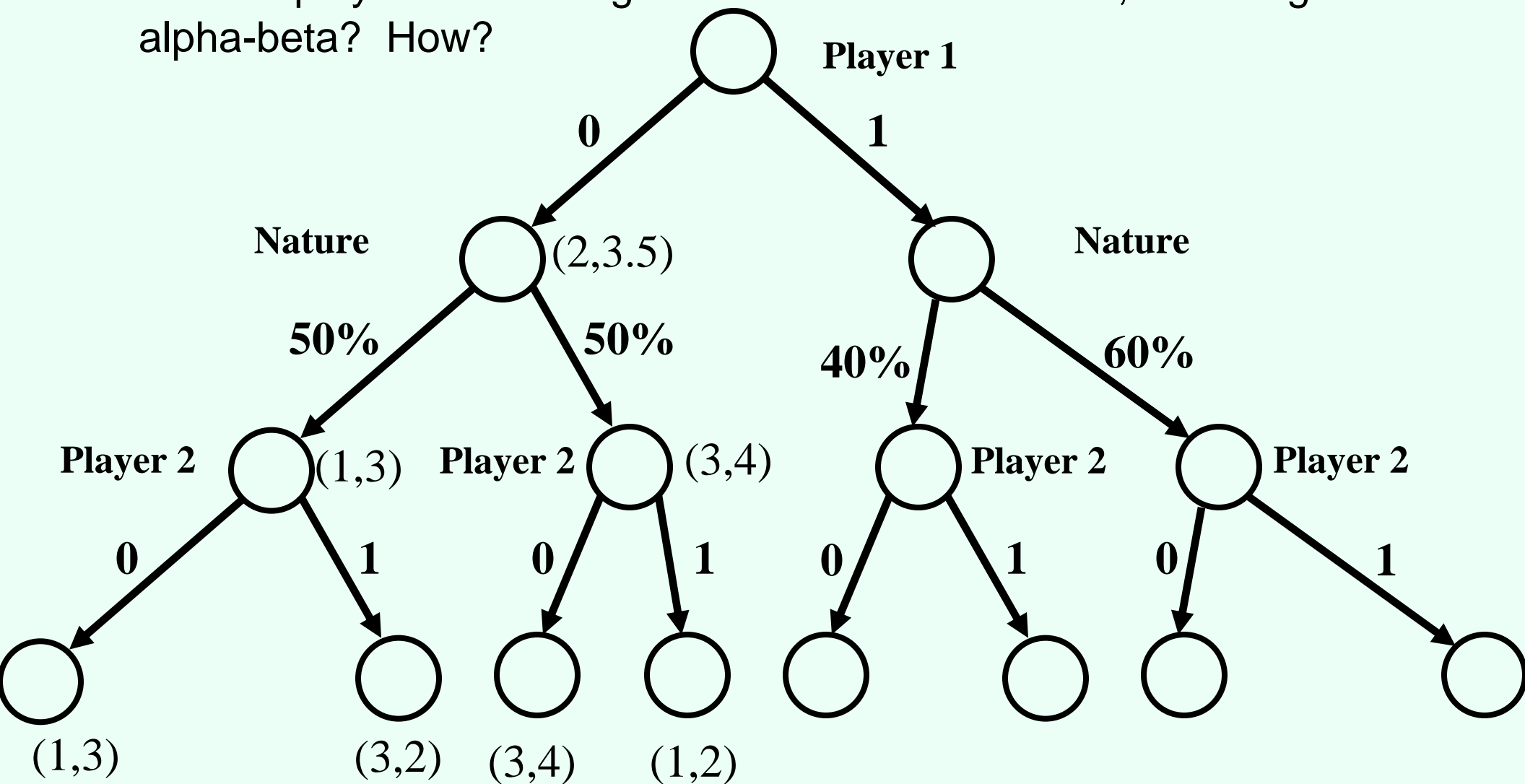
- Basic backward induction still works
 - No longer called minimax

What if other players do not play this way?



Games with random moves by “Nature”

- E.g., games with dice (Nature chooses dice roll)
- Backward induction still works...
 - Evaluation functions now need to be **cardinally** right (not just **ordinally**)
 - For two-player zero-sum games with random moves, can we generalize alpha-beta? How?



Games with imperfect information

- Players cannot necessarily see the whole current state of the game

- Card games

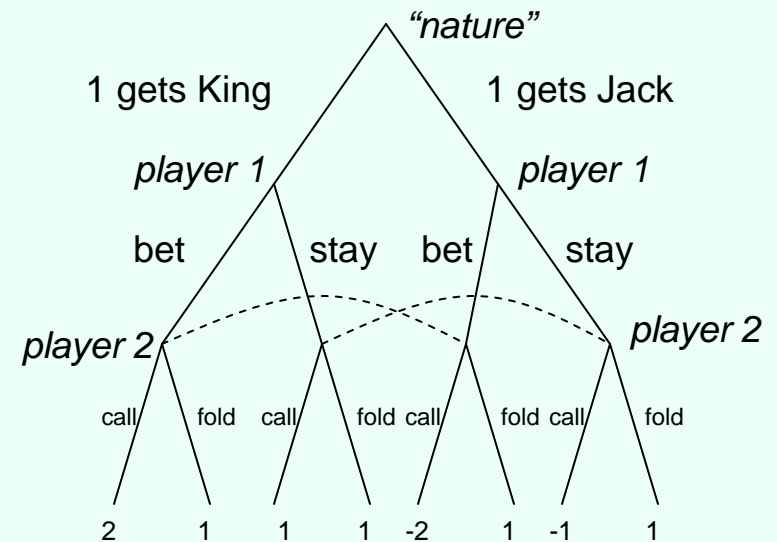
- Ridiculously simple poker game:

- Player 1 receives King (winning) or Jack (losing),

- Player 1 can bet or stay,

- Player 2 can call or fold

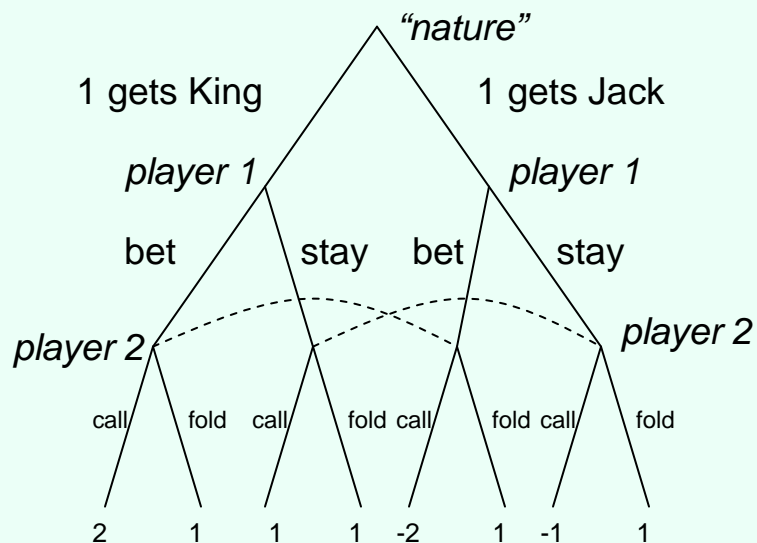
- Dashed lines indicate indistinguishable states



- Backward induction does **not** work, need random strategies for optimality! (more later in course)

Intuition for need of random strategies

- Suppose my strategy is “bet on King, stay on Jack”
 - What will you do?
 - What is your expected utility?
- What if my strategy is “always bet”?
- What if my strategy is “always bet when given King, 10% of the time bet when given Jack”?



The state of the art for some games

- Chess:
 - 1997: IBM Deep Blue defeats Kasparov
 - ... there is still debate about whether computers are really better
- Checkers:
 - Computer world champion since 1994
 - ... there was still debate about whether computers are really better...
 - until 2007: checkers solved **optimally** by computer
- Go:
 - Computers still not very good
 - Branching factor really high
 - Some recent progress
- Poker:
 - Competitive with top humans in some 2-player games
 - 3+ player case much less well-understood

Is this of any value to society?

- Some of the techniques developed for games have found applications in other domains
 - Especially “adversarial” settings
- Real-world strategic situations are usually not two-player, perfect-information, zero-sum, ...
- But **game theory** does not need any of those
- Example application: security scheduling at airports