Artificial Intelligence DT8012

Adversarial search

Chapter 6, AIMA 2nd ed

Chapter 5, AIMA 3rd ed

This presentation owes a lot to V. Pavlovic @ Rutgers, who borrowed from J. D. Skrentny, who in turn borrowed from C. Dyer,...

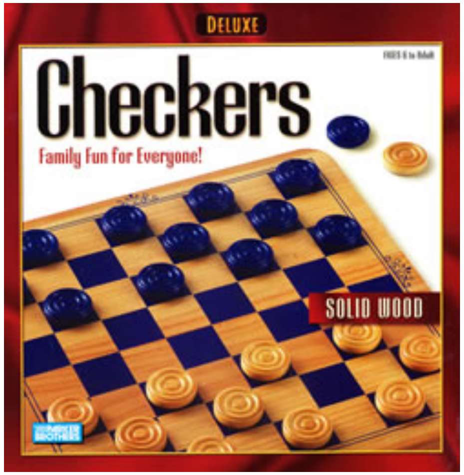
Adversarial search

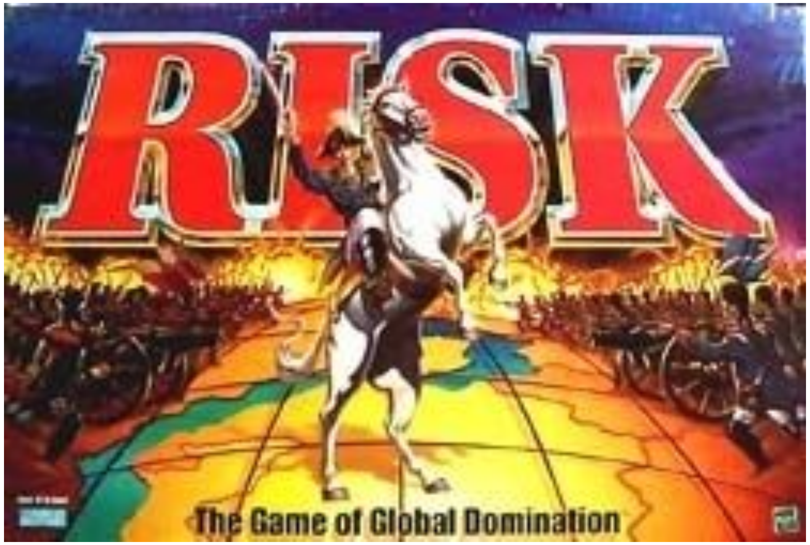
- At least two agents and a competitive environment: Games, economies.

- Games and AI:
  - Generally considered to require intelligence (to win)
  - Have to evolve in real-time
  - Well-defined and limited environment

Board games

© Thierry Dichtenmuller





Games & AI

	Deterministic	Chance
perfect info	**Checkers, Chess, Go, Othello**	**Backgammon, Monopoly**
imperfect info		**Bridge, Poker, Scrabble**

Games and search

Traditional search: single agent, searches for its well-being, unobstructed

Games: search against an opponent

Example: two player board game (chess, checkers, tic-tac-toe,…)

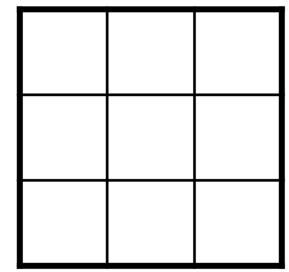
Board configuration: unique arrangement of "pieces‟

Representing board games as goal-directed search problem (states = board configurations):

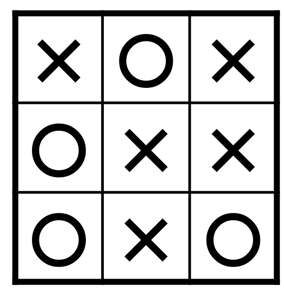
- **Initial state:** Current board configuration
- **Successor function:** Legal moves
- **Goal state:** Winning/terminal board configuration
- **Utility function:** Value of final state

Example: Tic-tac-toe

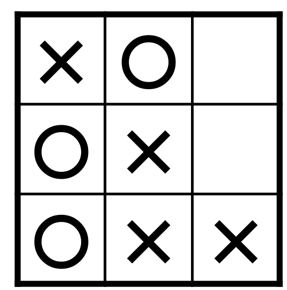
- **Initial state:** 3×3 empty table.
- **Successor function:** Players take turns marking × or O in the table cells.
- **Goal state:** When all the table cells are filled or when either player has three symbols in a row.
- **Utility function:** +1 for three in a row, -1 if the opponent has three in a row, 0 if the table is filled and no-one has three symbols in a row.



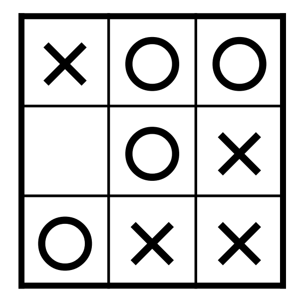
Initial state



Goal state
Utility = 0



Goal state
Utility = +1



Goal state
Utility = -1

The minimax principle

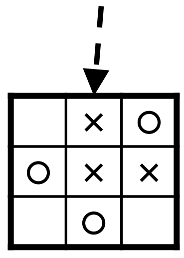
Assume the opponent plays to win and always makes the best possible move.

The **minimax value** for a node = the utility for you of being in that state, assuming that both players (you and the opponent) play optimally from there on to the end.

Terminology:
MAX = you, MIN = the opponent.

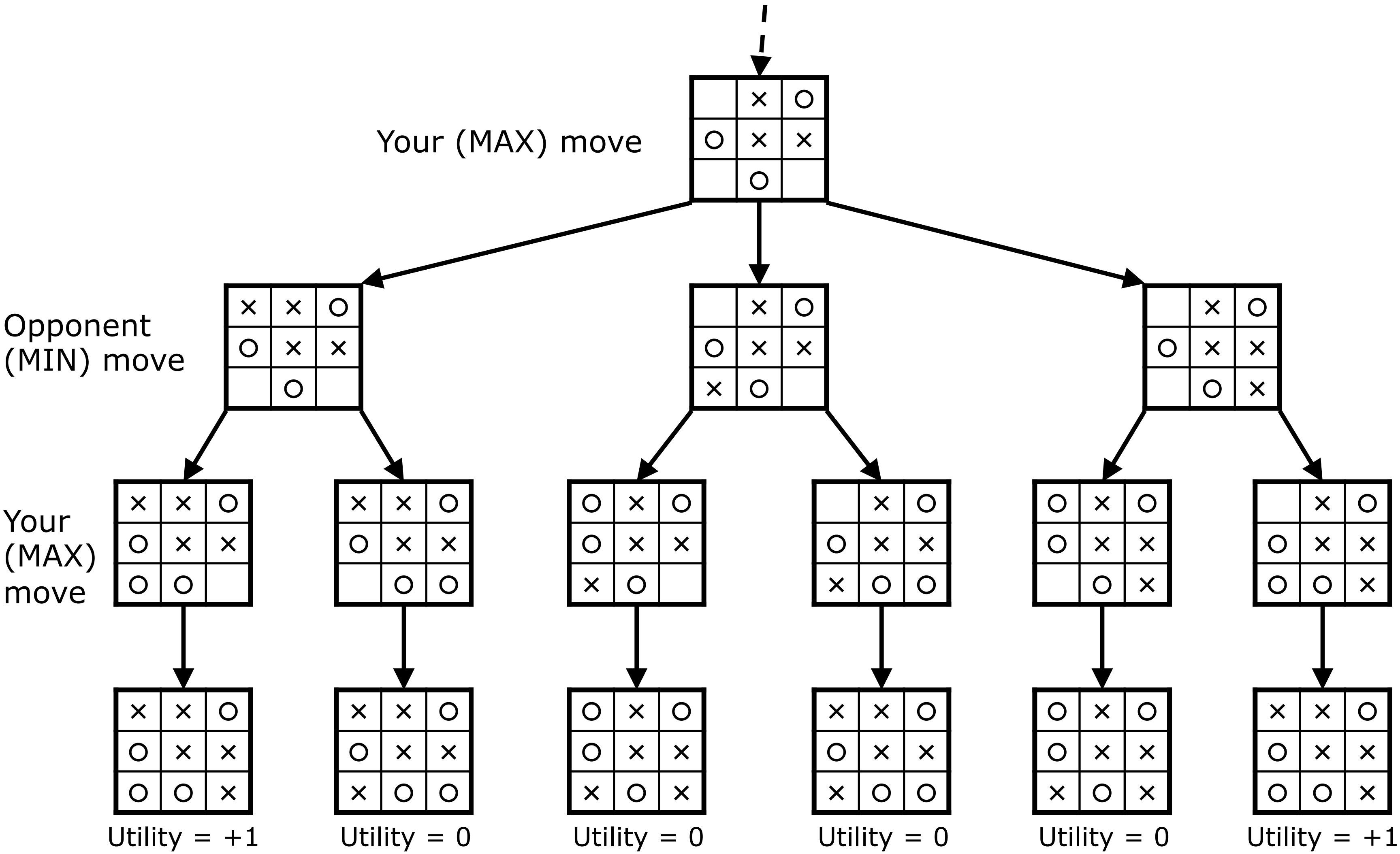
Example: Tic-tac-toe

Your (MAX) move
(×)

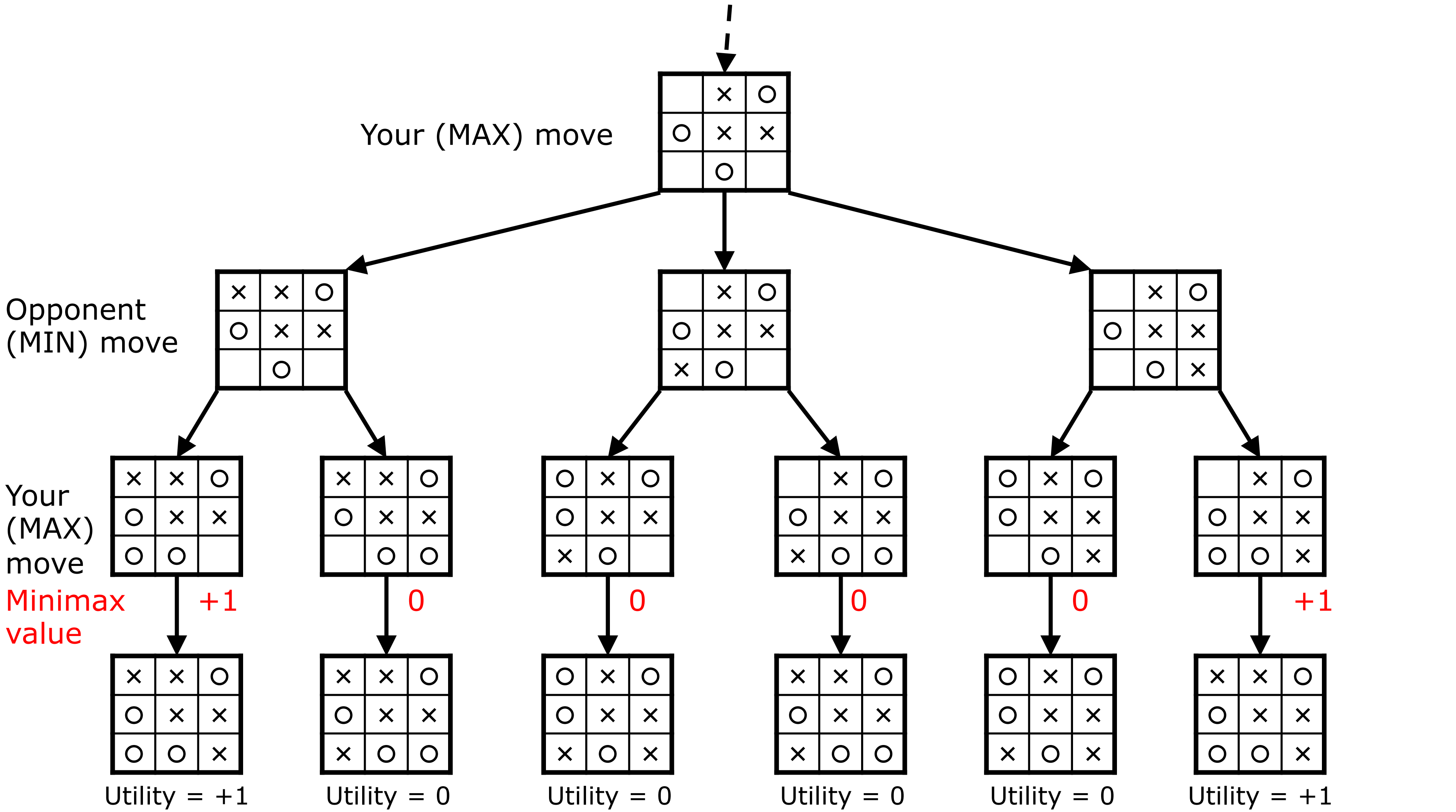


Assignment: Expand this tree to the end of the game.

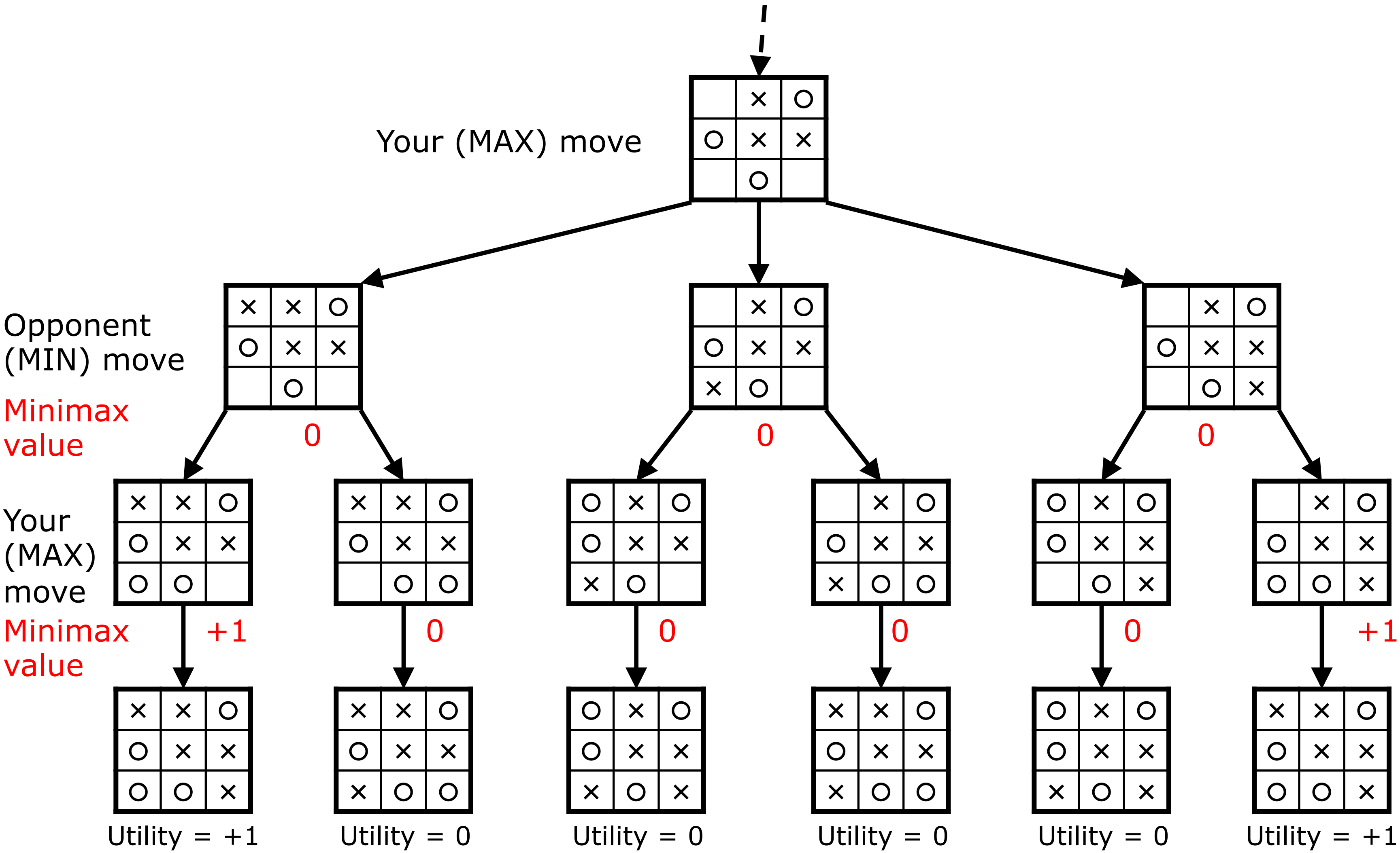
Example: Tic-tac-toe



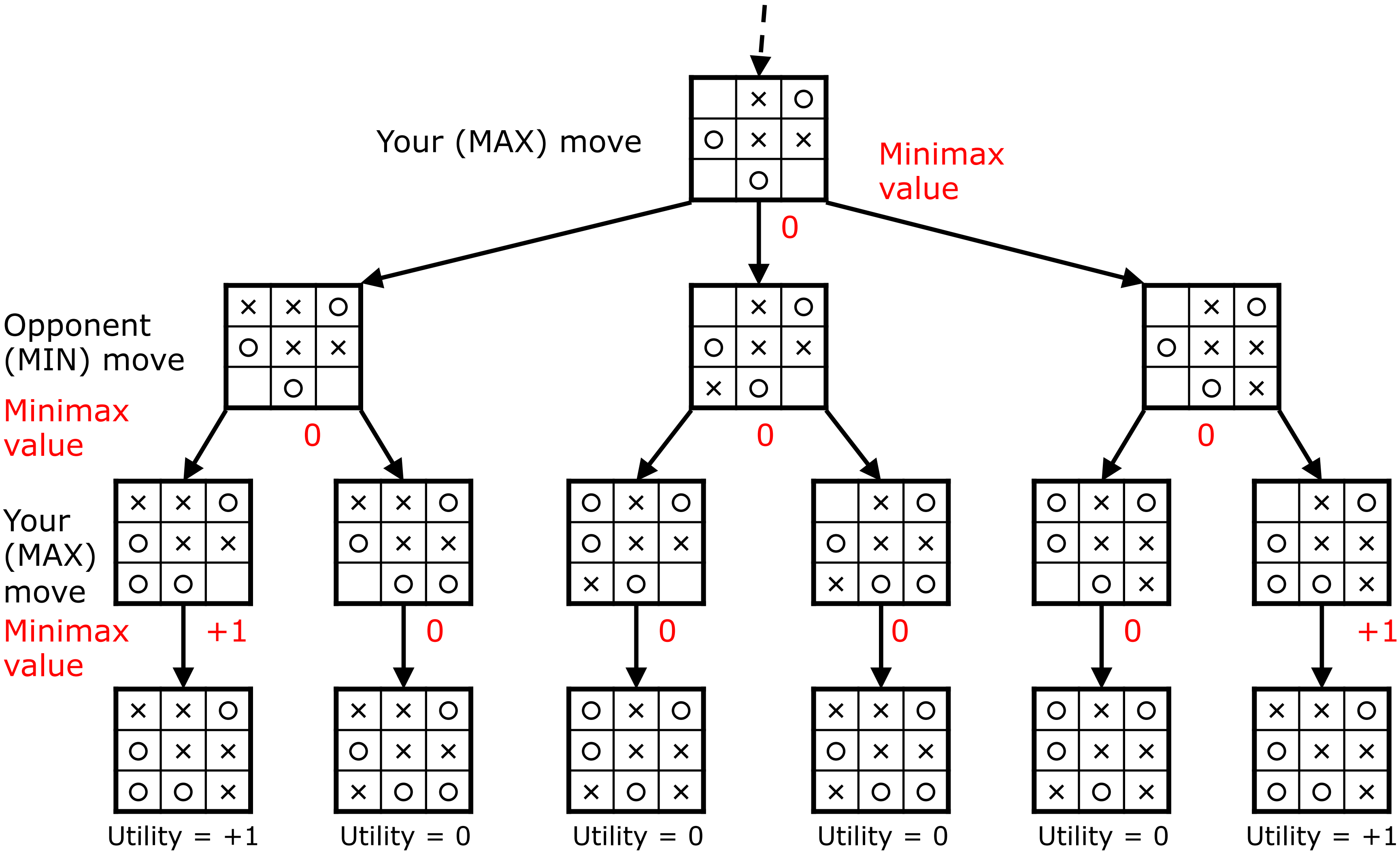
Example: Tic-tac-toe



Example: Tic-tac-toe



Example: Tic-tac-toe



The minimax value

Minimax value for node n =

$$\begin{cases} \text{Utility}(n) & \text{If } n \text{ is a terminal node} \\ \text{Max(Minimax-values of successors)} & \text{If } n \text{ is a MAX node} \\ \text{Min(Minimax-values of successors)} & \text{If } n \text{ is a MIN node} \end{cases}$$

High utility favours you (MAX), therefore choose move with highest utility

Low utility favours the opponent (MIN), therefore choose move with lowest utility

The minimax algorithm

1. Start with utilities of terminal nodes
2. Propagate them back to root node by choosing the **minimax** strategy

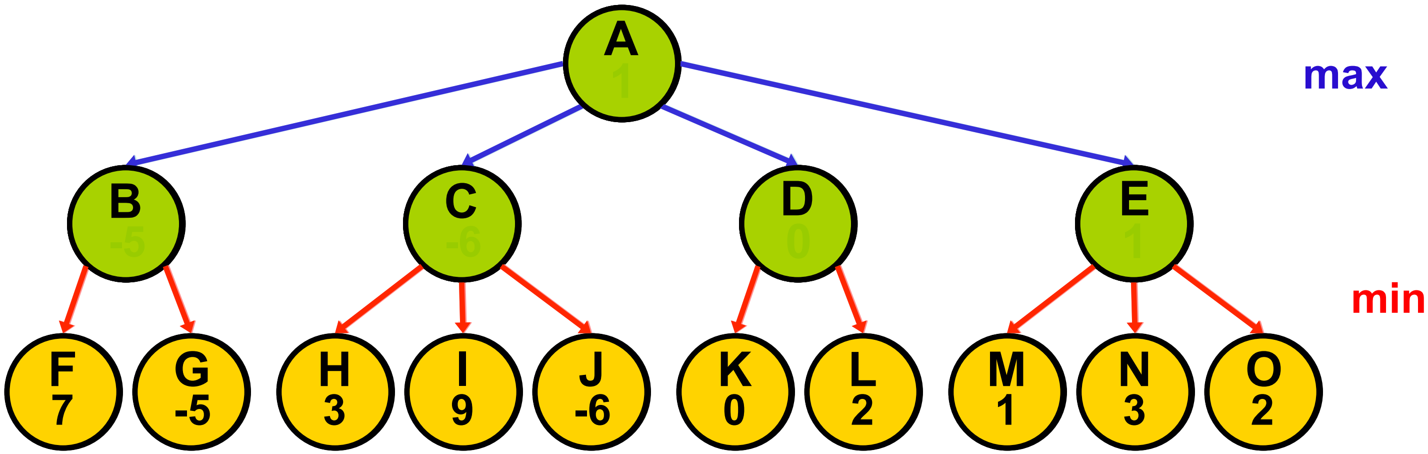


Figure borrowed from V. Pavlovic

The minimax algorithm

1. Start with utilities of terminal nodes
2. Propagate them back to root node by choosing the **minimax** strategy

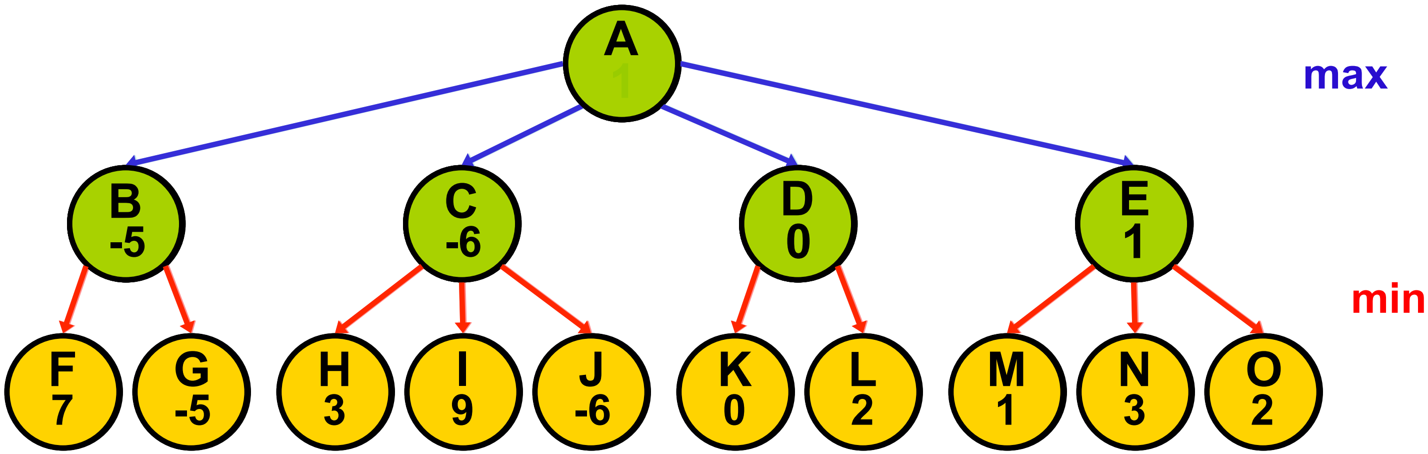


Figure borrowed from V. Pavlovic

The minimax algorithm

1. Start with utilities of terminal nodes
2. Propagate them back to root node by choosing the **minimax** strategy

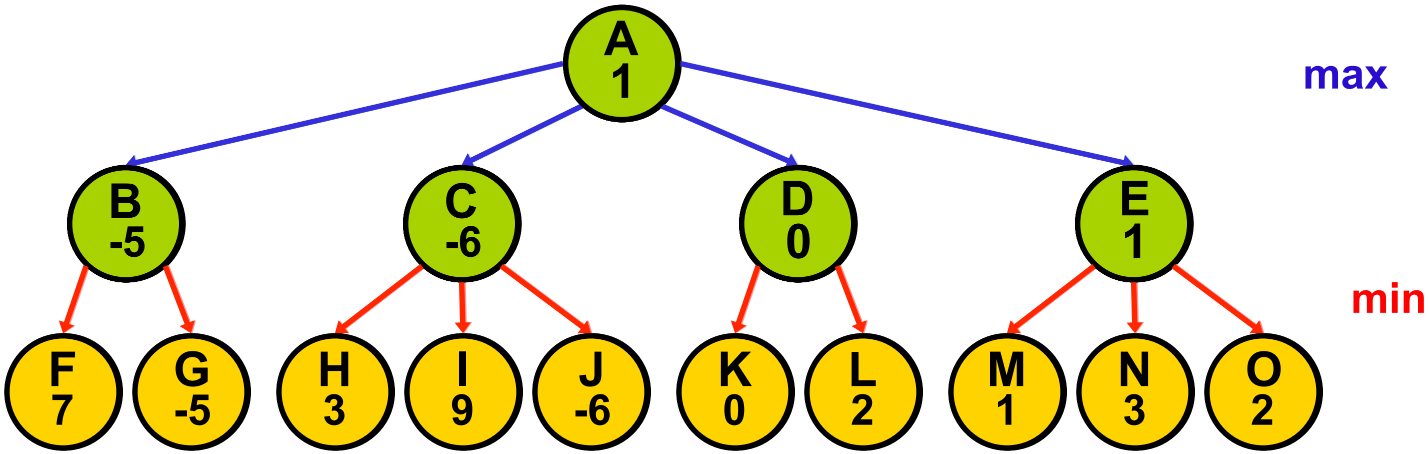


Figure borrowed from V. Pavlovic

Complexity of minimax algorithm

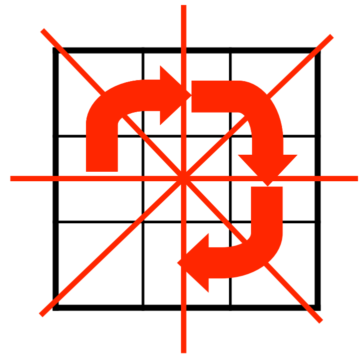
- A depth-first search
  - Time complexity $O(b^d)$
  - Space complexity $O(bd)$

- Time complexity <u>impossible</u> in real games (with time constraints) except in very simple games (e.g. tic-tac-toe)

Strategies to improve minimax

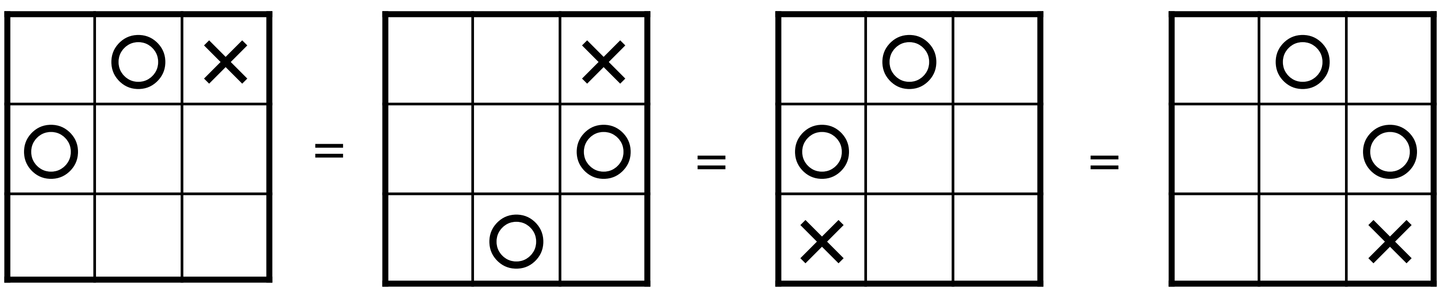
1. Remove redundant search paths
   - symmetries
2. Remove uninteresting search paths
   - alpha-beta pruning
3. Cut the search short before goal
   - Evaluation functions
4. Book moves

1. Remove redundant paths



Tic-tac-toe has mirror symmetries

and rotational symmetries



First three levels of the tic-tac-toe state space reduced by symmetry

3 states
(instead of 9)

12 states
(instead of
8·9 = 72)

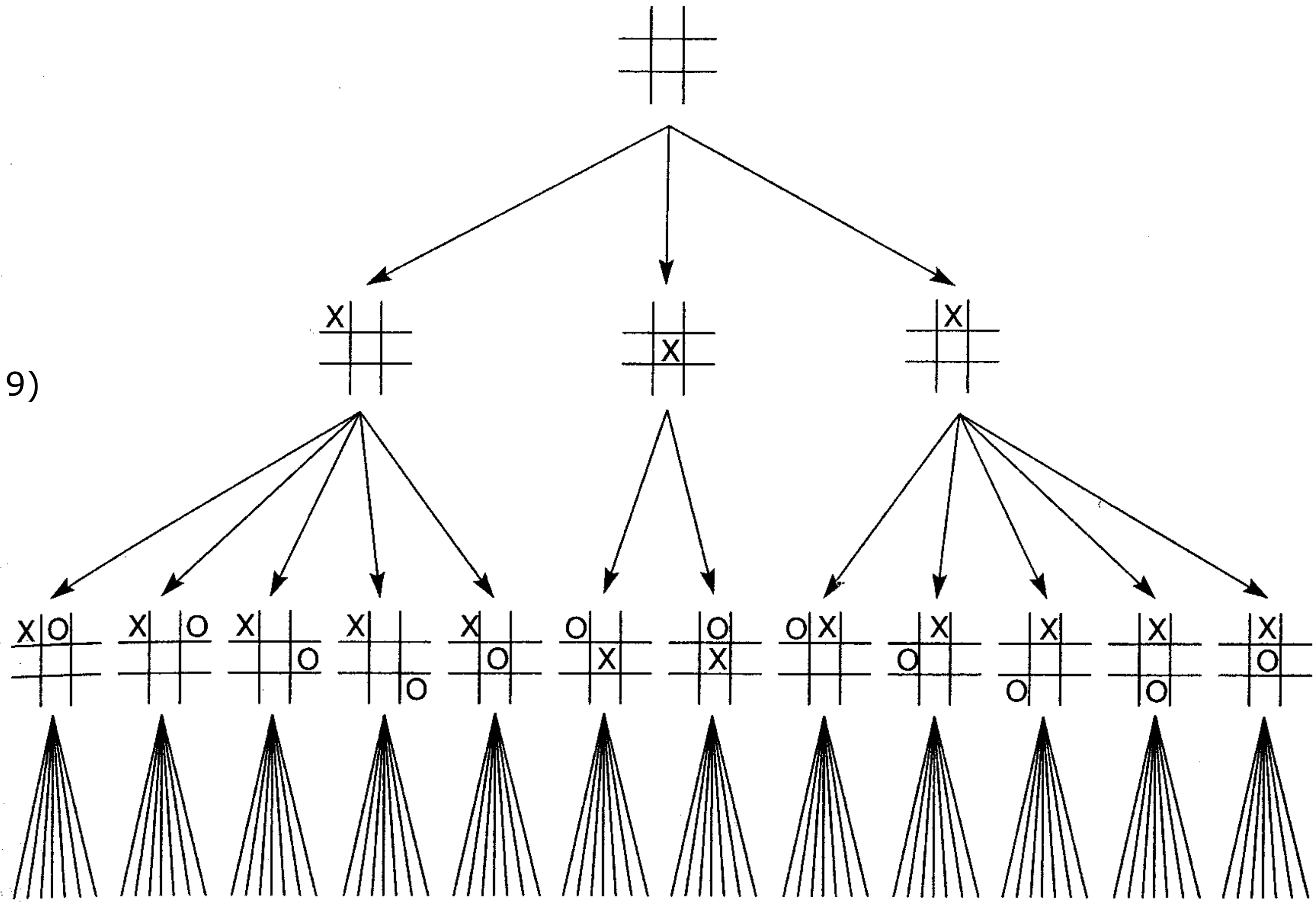


Image from G. F. Luger, ”Artificial Intelligence”, 2002

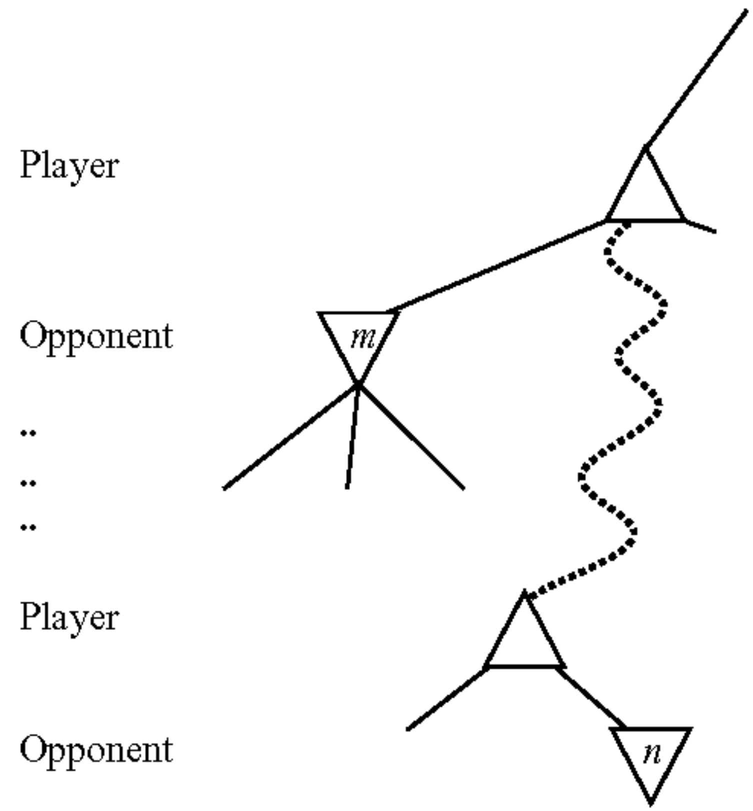
2. Remove uninteresting paths

If the player has a better choice m at n's parent node, or at any node further up, then node n will never be reached.

Prune the entire path below node m's parent node (except for the path that m belongs to, and paths that are equal to this path).

Minimax is depth-first → keep track of highest (α) and lowest (β) values so far.

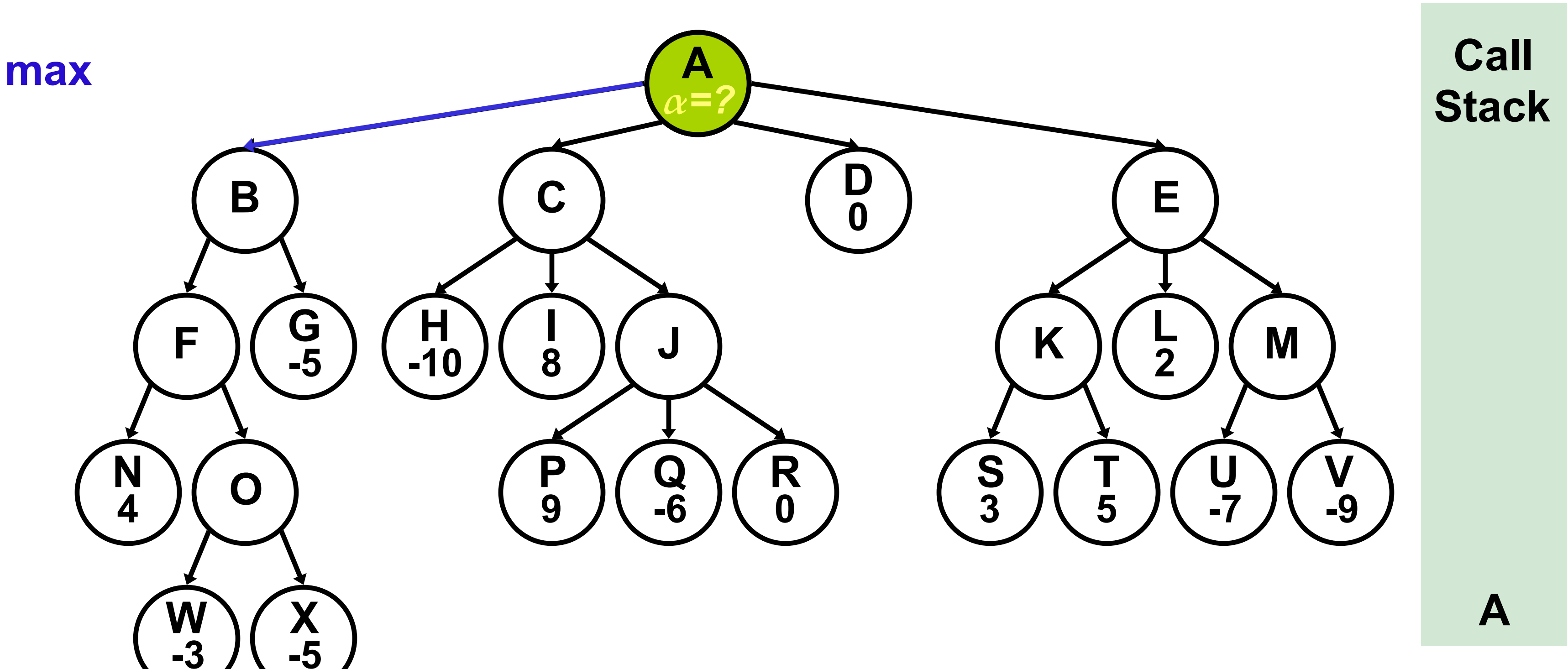
Called alpha-beta pruning.



Alpha-Beta Example

`minimax(A,0,4)`

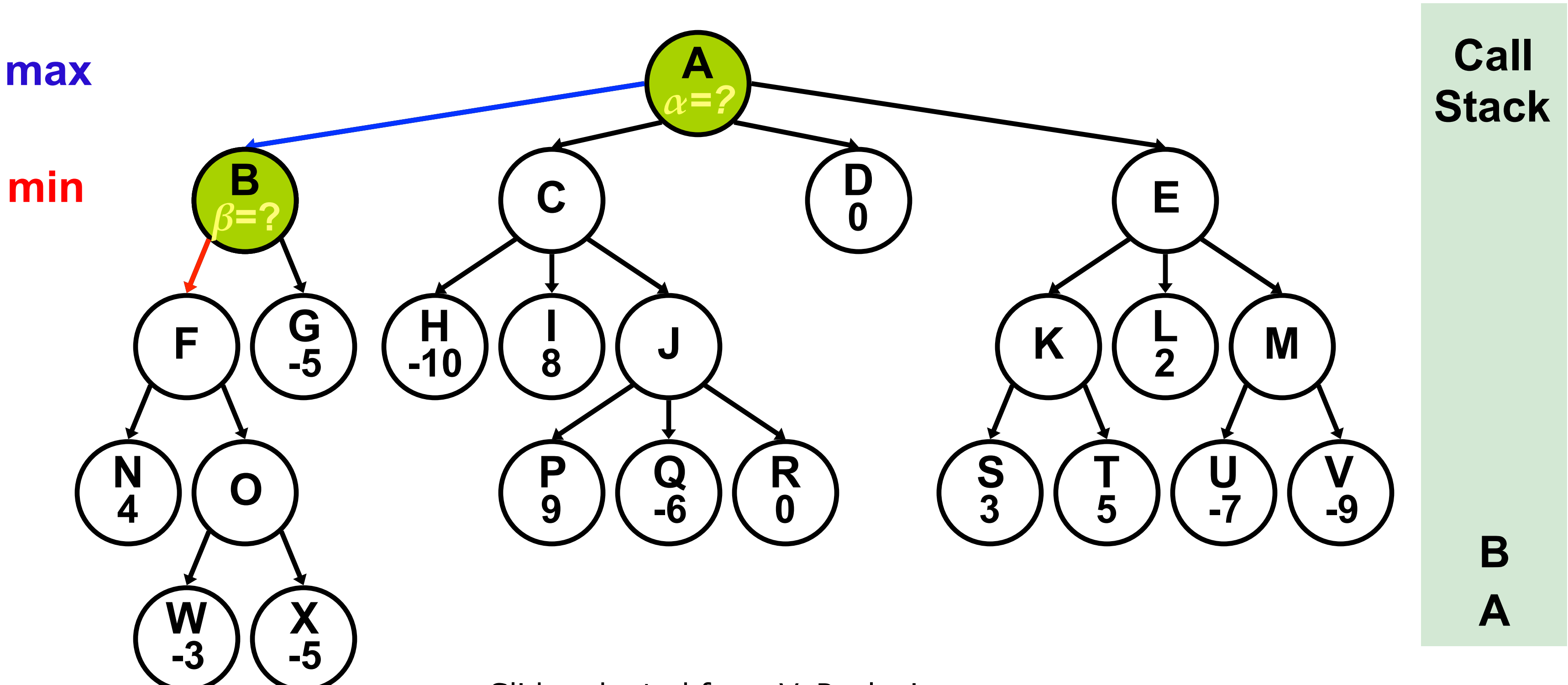
minimax(node, level, depth limit)



Slide adapted from V. Pavlovic

Alpha-Beta Example

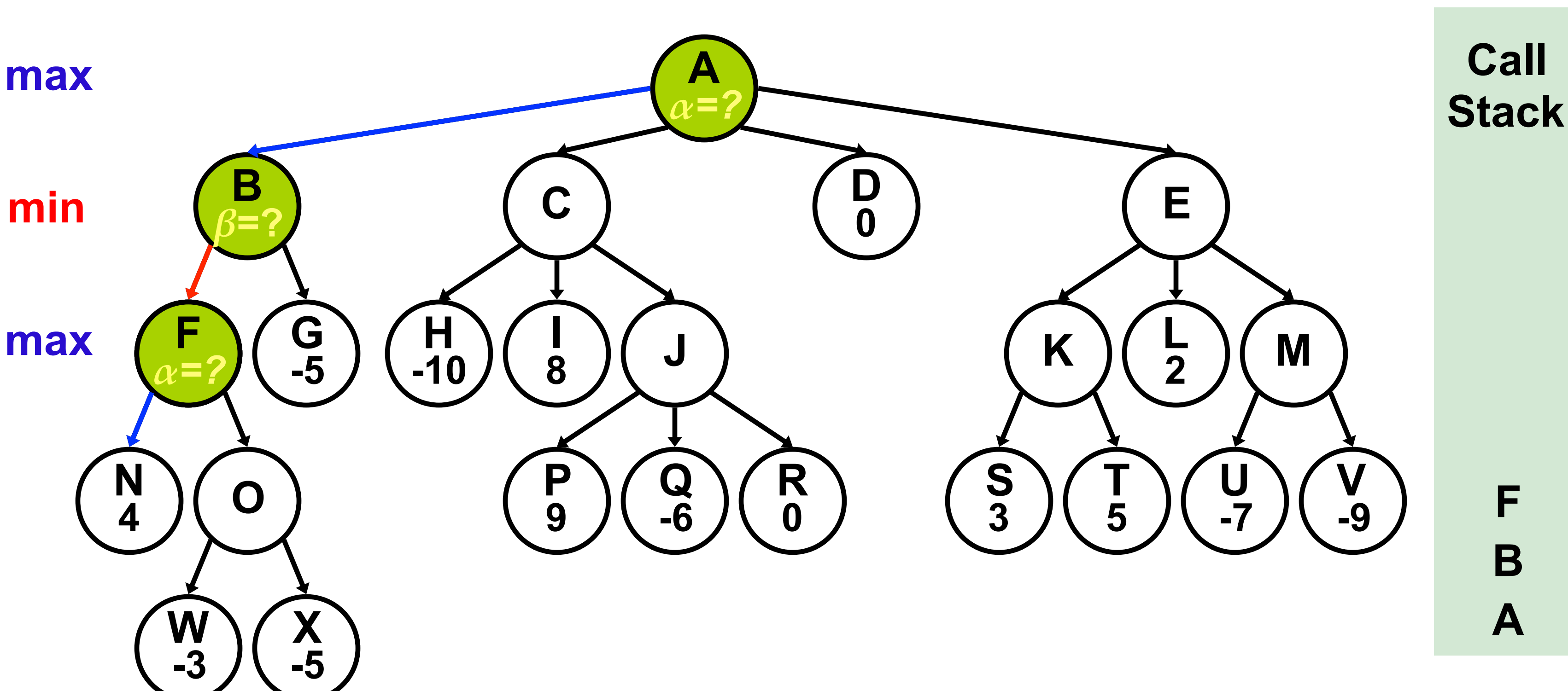
`minimax(B,1,4)`



Slide adapted from V. Pavlovic

Alpha-Beta Example

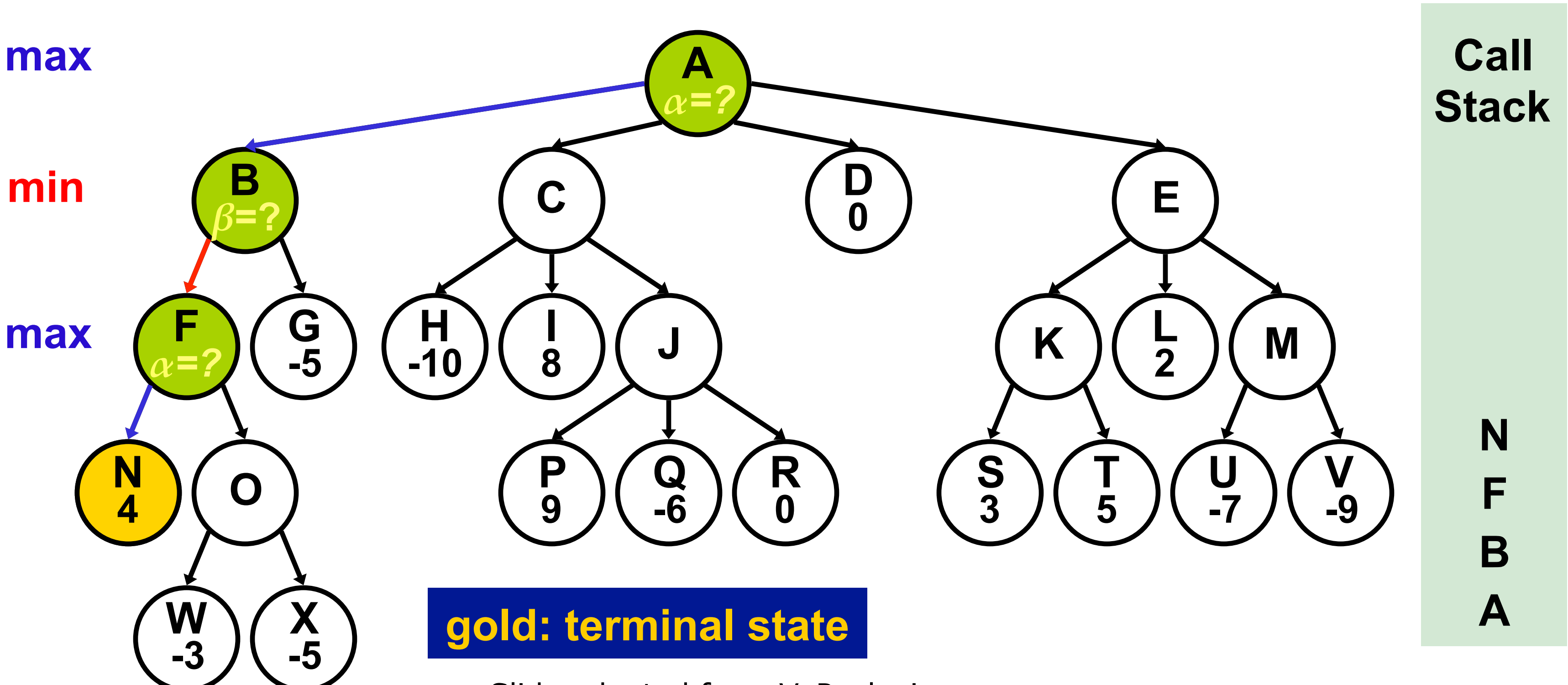
`minimax(F,2,4)`



Slide adapted from V. Pavlovic

Alpha-Beta Example

minimax(N,3,4)

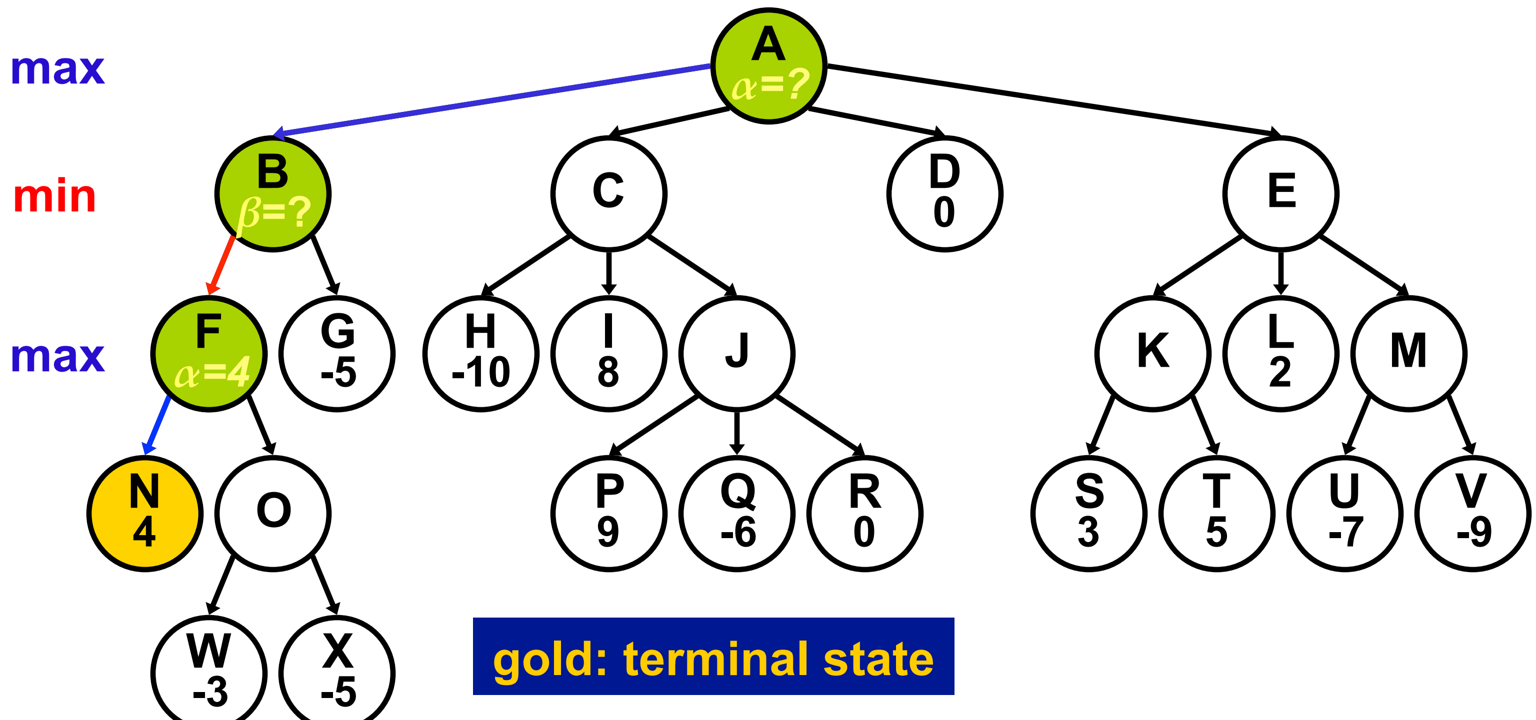


Slide adapted from V. Pavlovic

Alpha-Beta Example

`minimax(F,2,4)` **is returned to**

`alpha = 4`, maximum seen so far



Call Stack

F

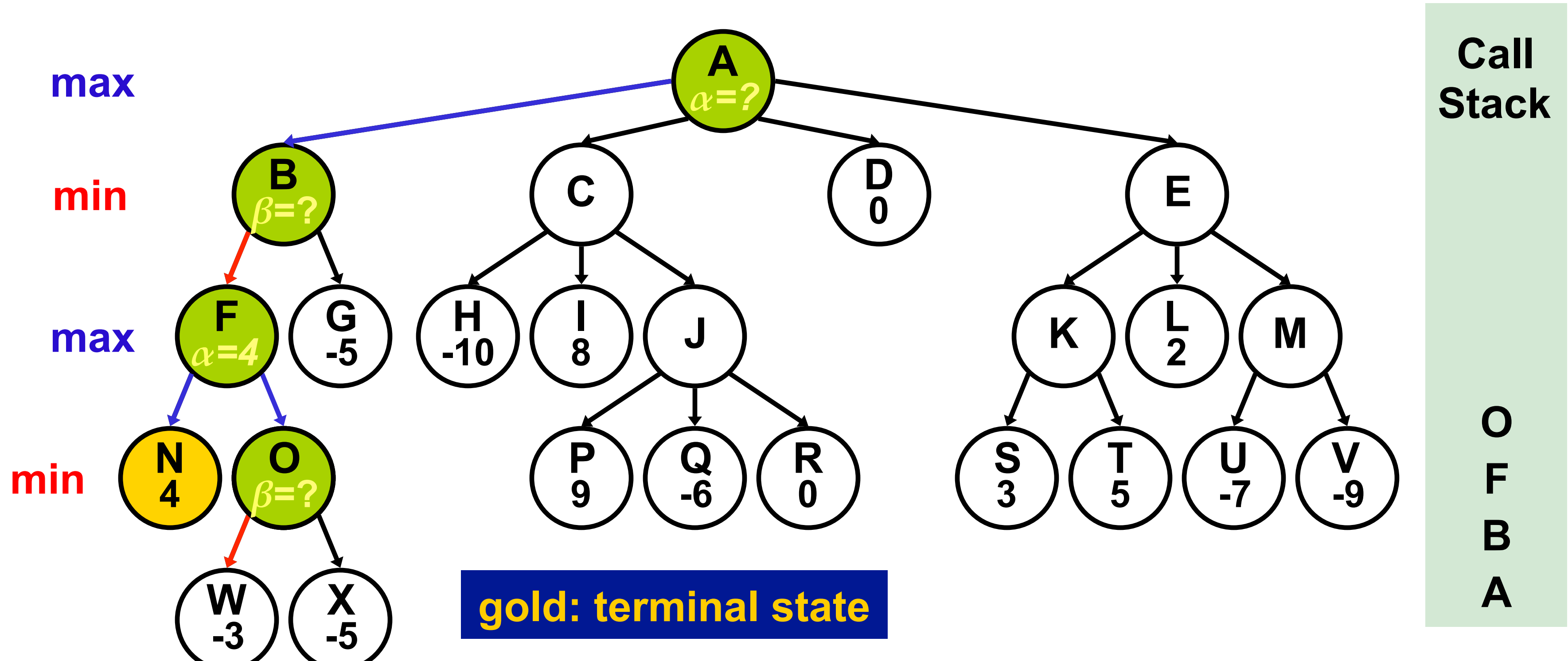
B

A

Slide adapted from V. Pavlovic

Alpha-Beta Example

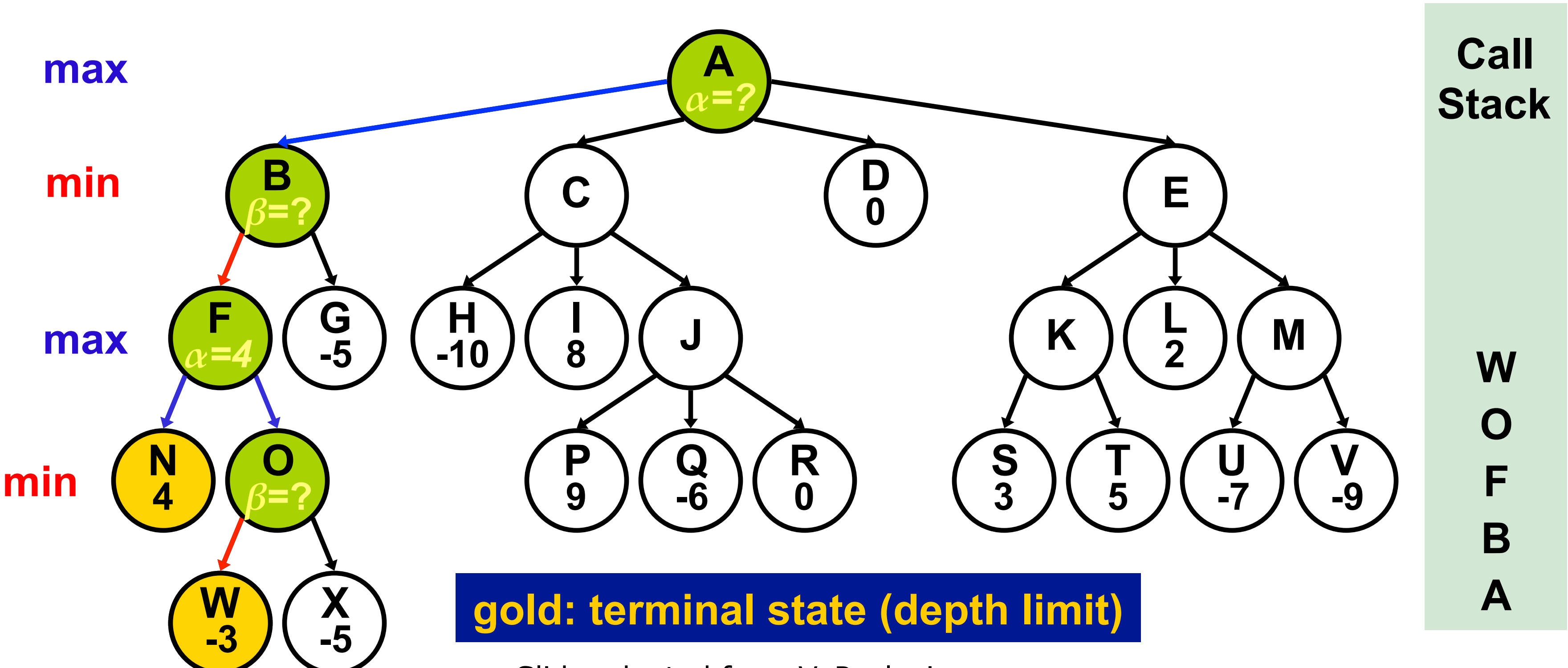
`minimax(O,3,4)`



Slide adapted from V. Pavlovic

Alpha-Beta Example

`minimax(W,4,4)`

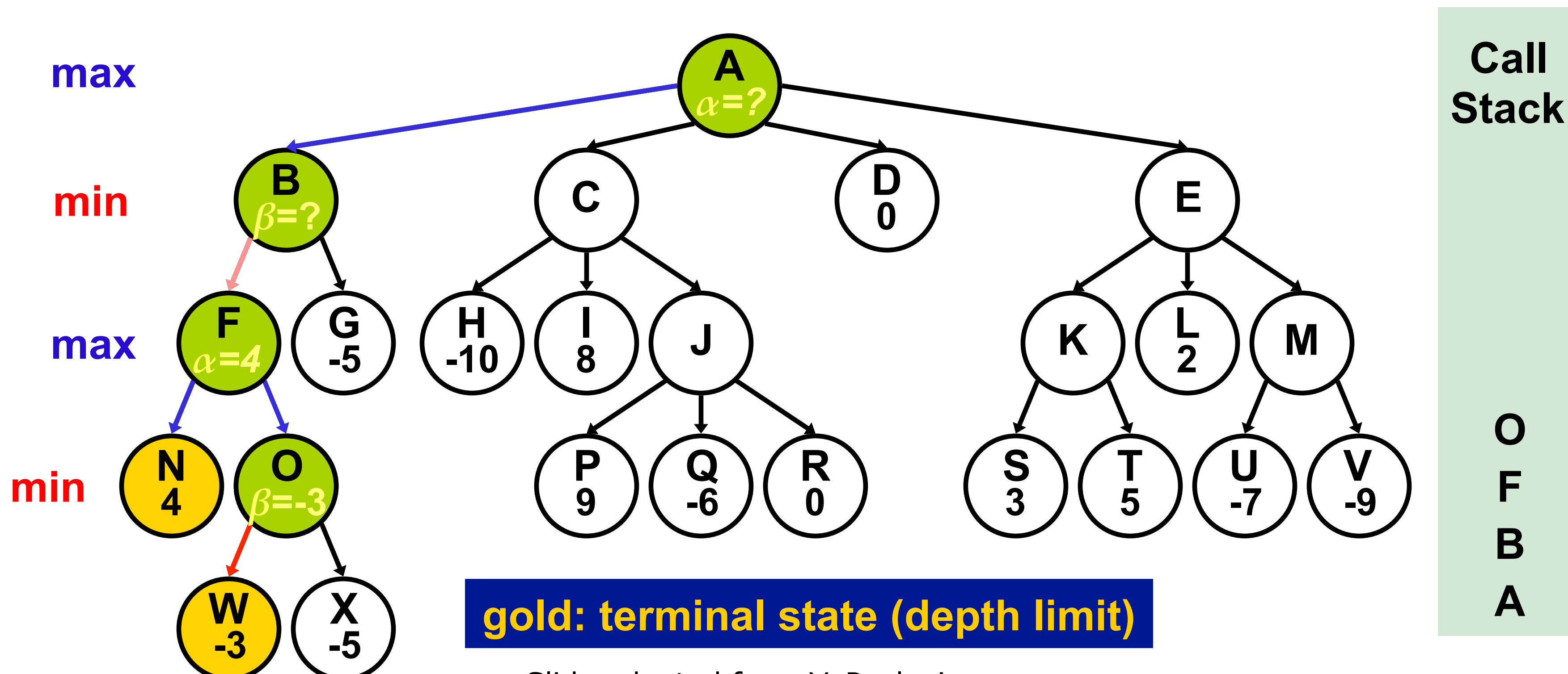


Slide adapted from V. Pavlovic

Alpha-Beta Example

`minimax(O,3,4)` **is returned to**

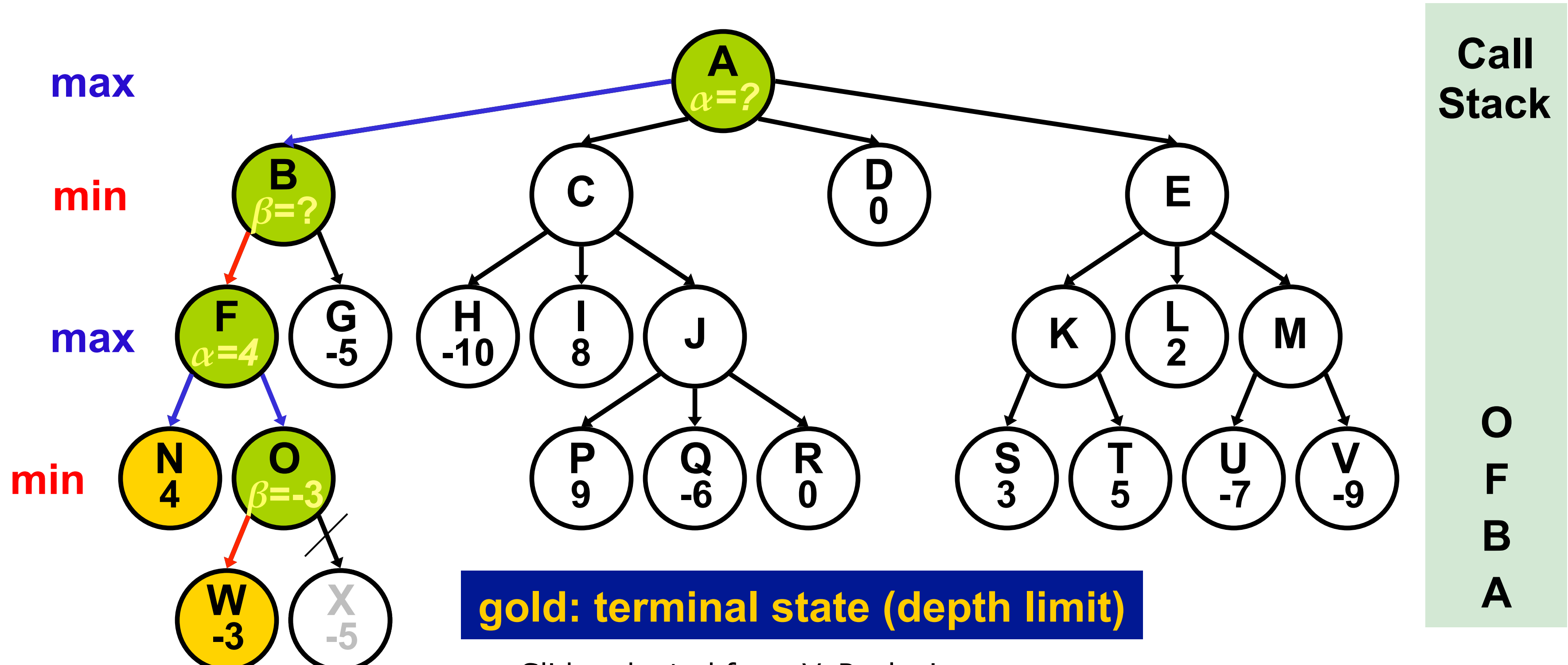
`beta = -3`, minimum seen so far



Slide adapted from V. Pavlovic

Alpha-Beta Example

O's beta (-3) < F's alpha (4): Stop expanding O (alpha cut-off)



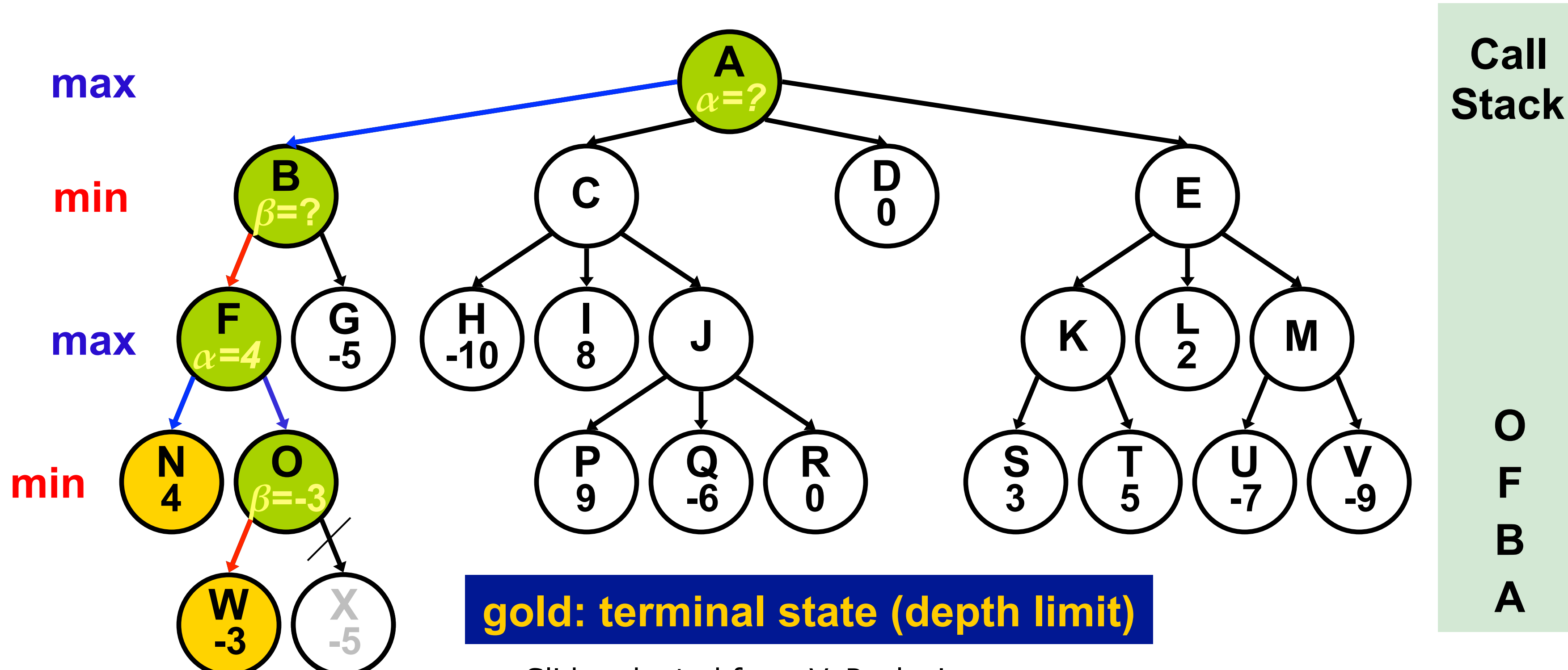
Slide adapted from V. Pavlovic

Alpha-Beta Example

Why?

Smart opponent selects W or worse → O's upper bound is –3

So MAX shouldn't select O:-3 since N:4 is better

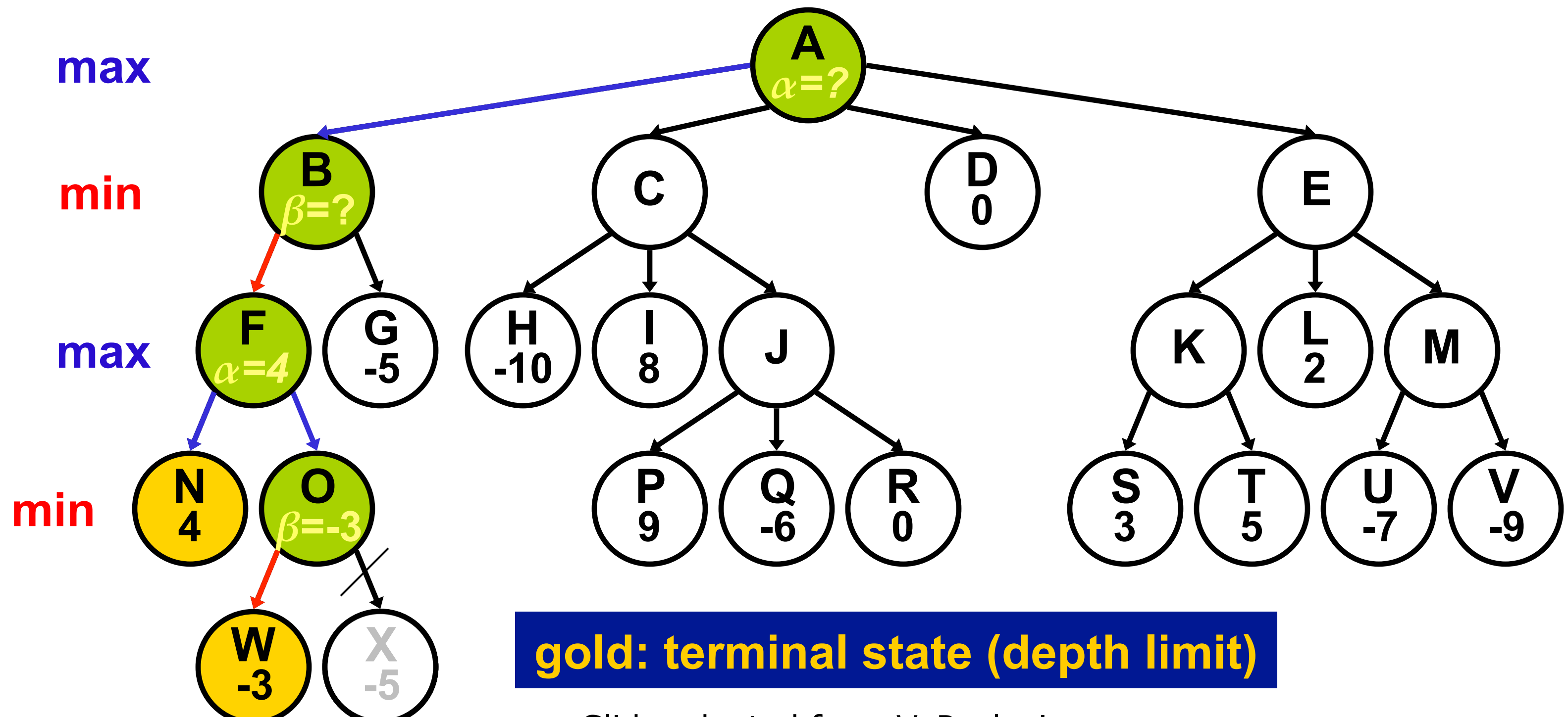


Slide adapted from V. Pavlovic

Alpha-Beta Example

`minimax(F,2,4)` **is returned to**

alpha not changed (maximizing)



Call Stack

F

B

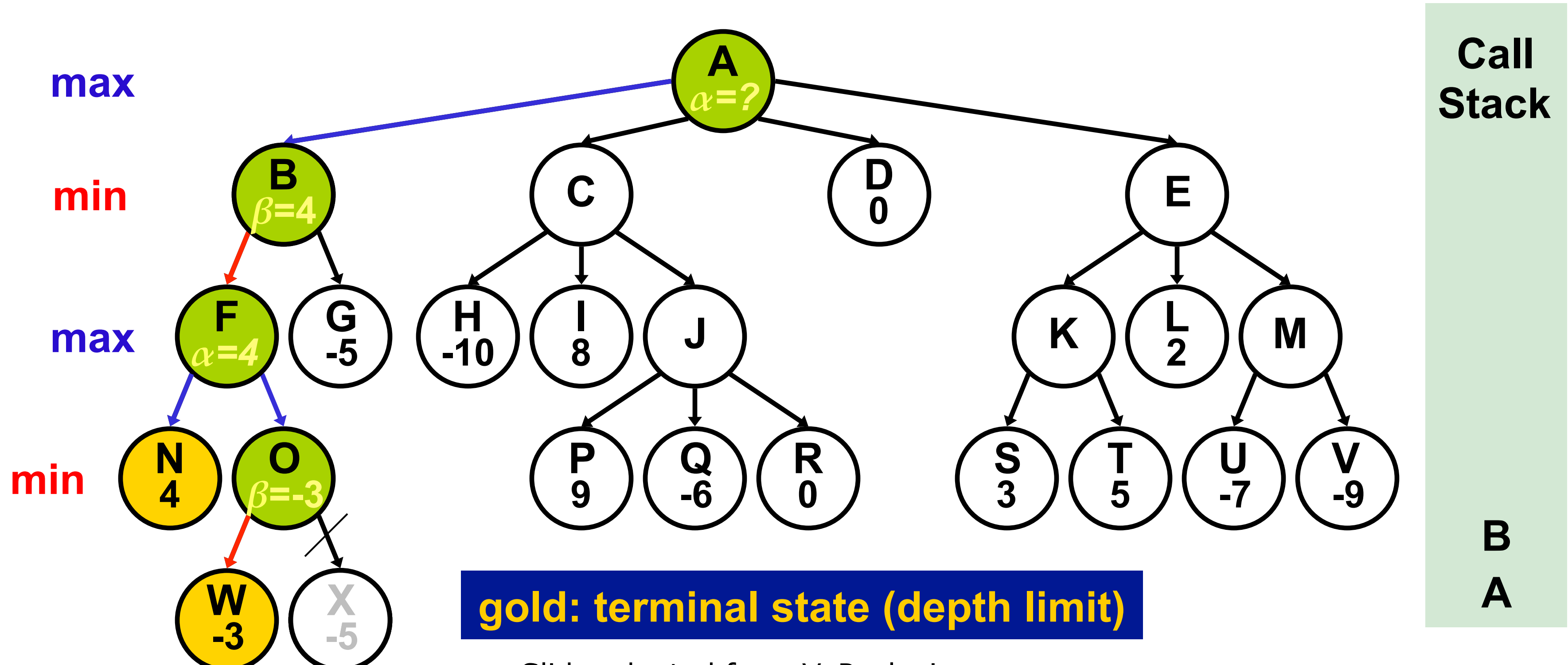
A

Slide adapted from V. Pavlovic

Alpha-Beta Example

`minimax(B,1,4)` is returned to

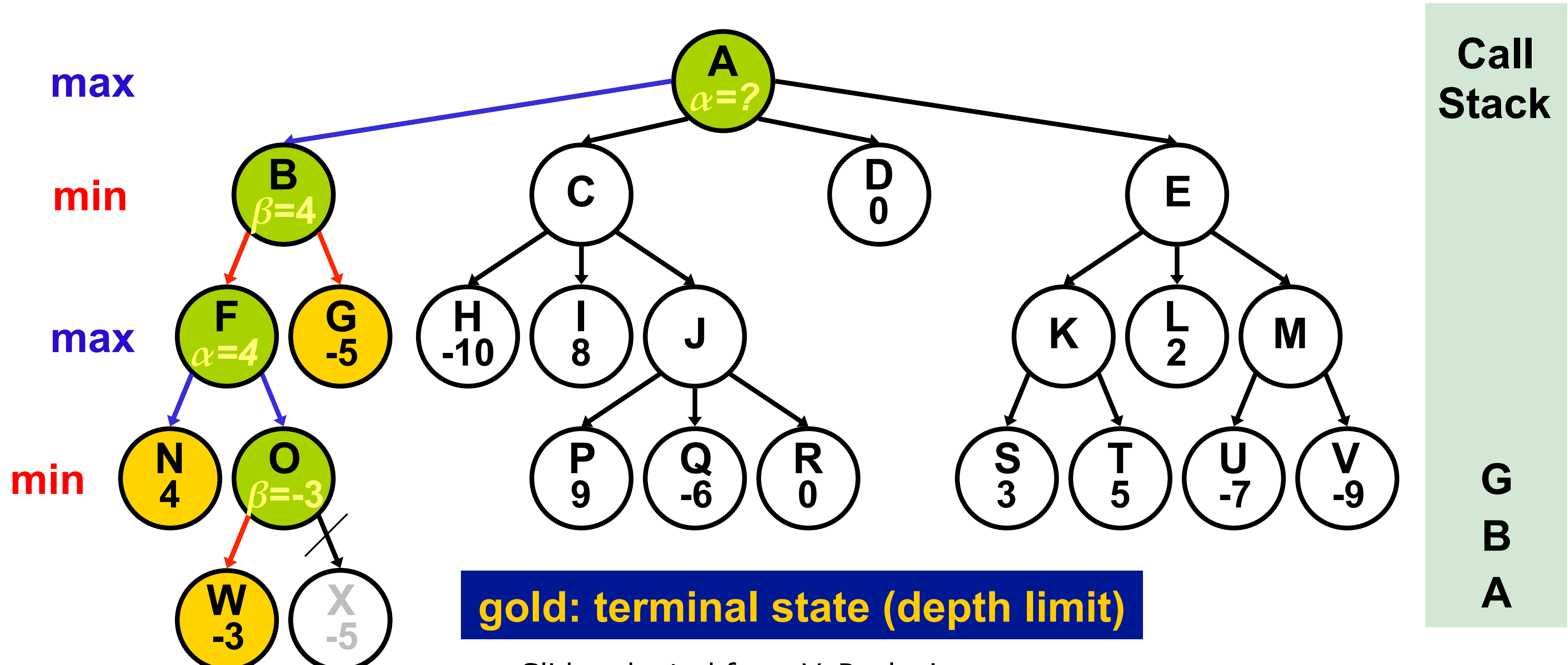
`beta = 4`, minimum seen so far



Slide adapted from V. Pavlovic

Alpha-Beta Example

minimax(G,2,4)

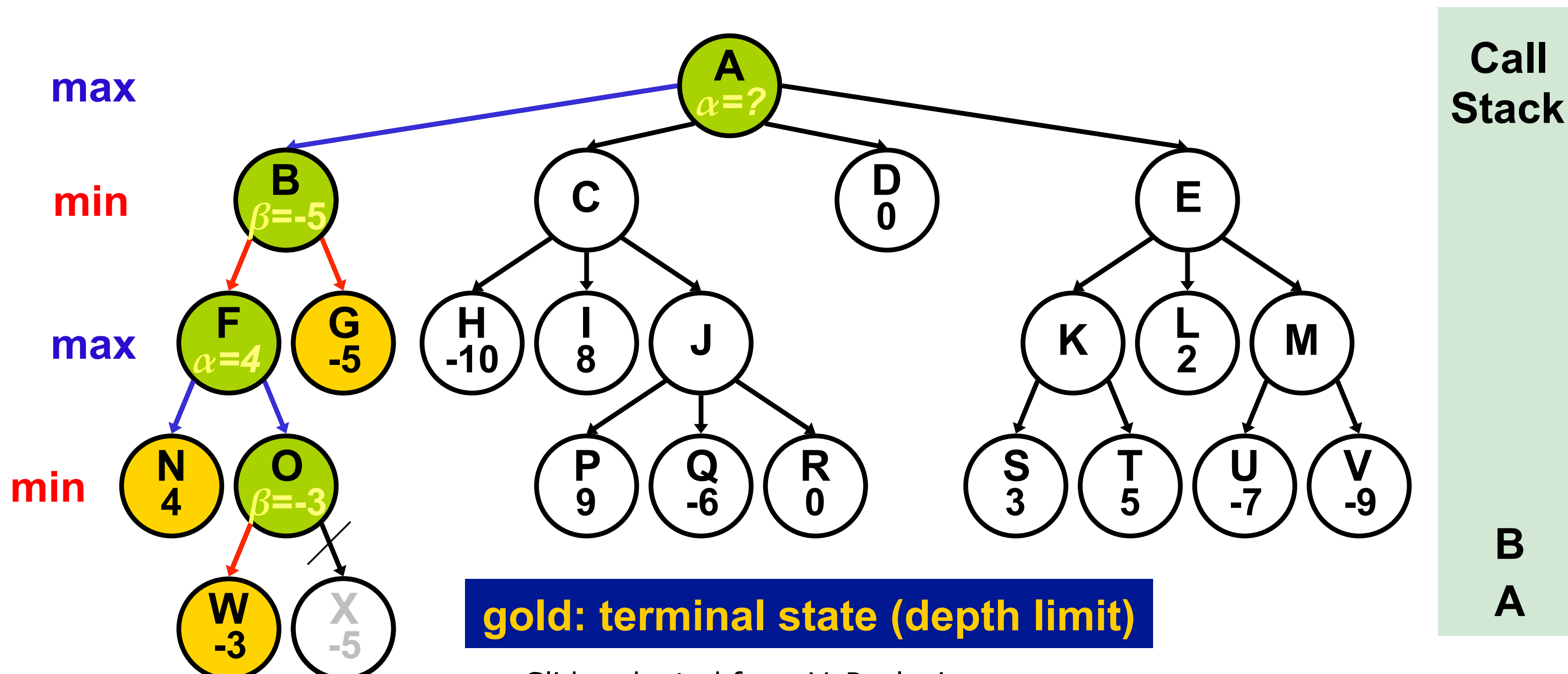


Slide adapted from V. Pavlovic

Alpha-Beta Example

`minimax(B,1,4)` **is returned to**

`beta = -5`, minimum seen so far

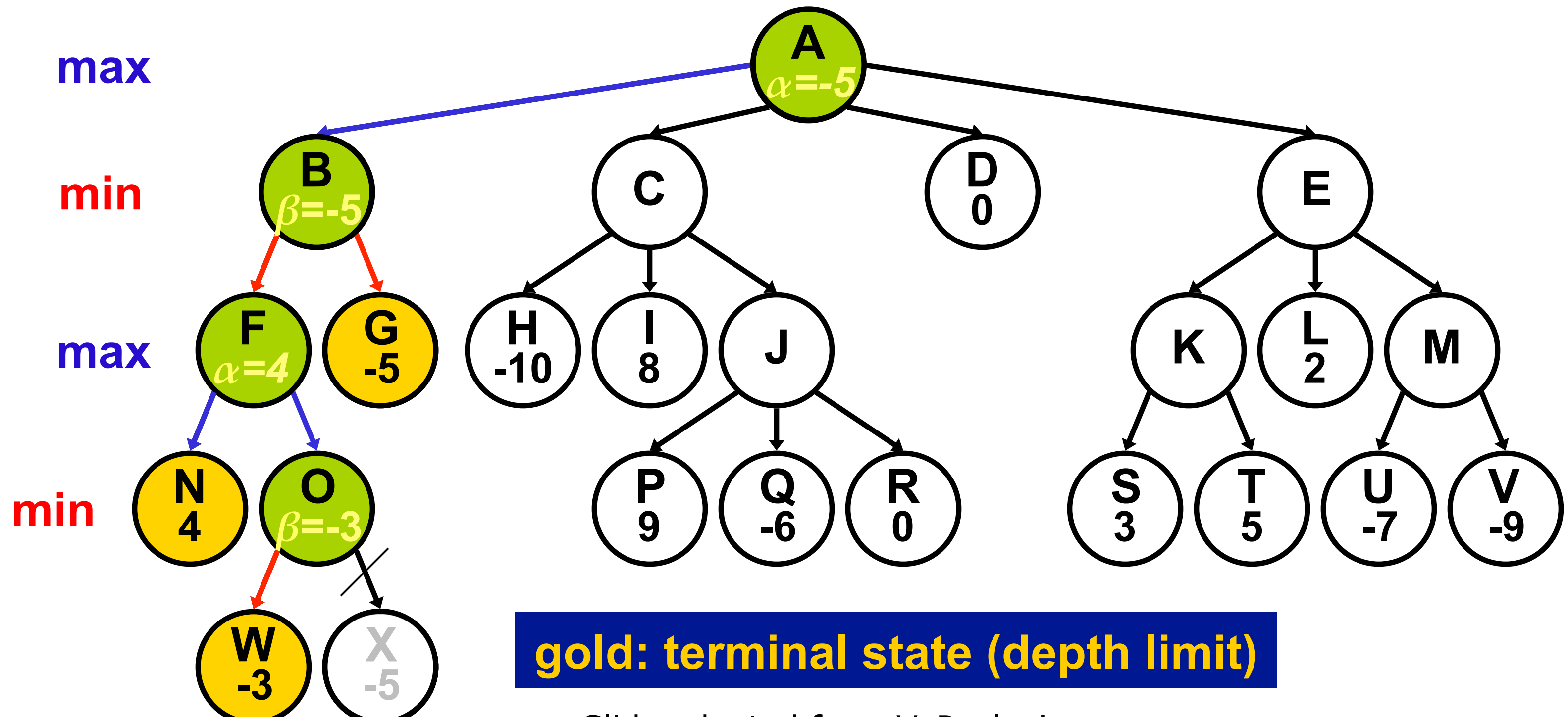


Slide adapted from V. Pavlovic

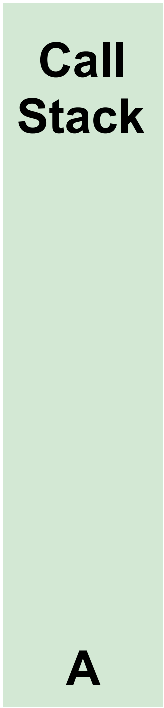
Alpha-Beta Example

`minimax(A,0,4)` **is returned to**

`alpha = -5`, maximum seen so far



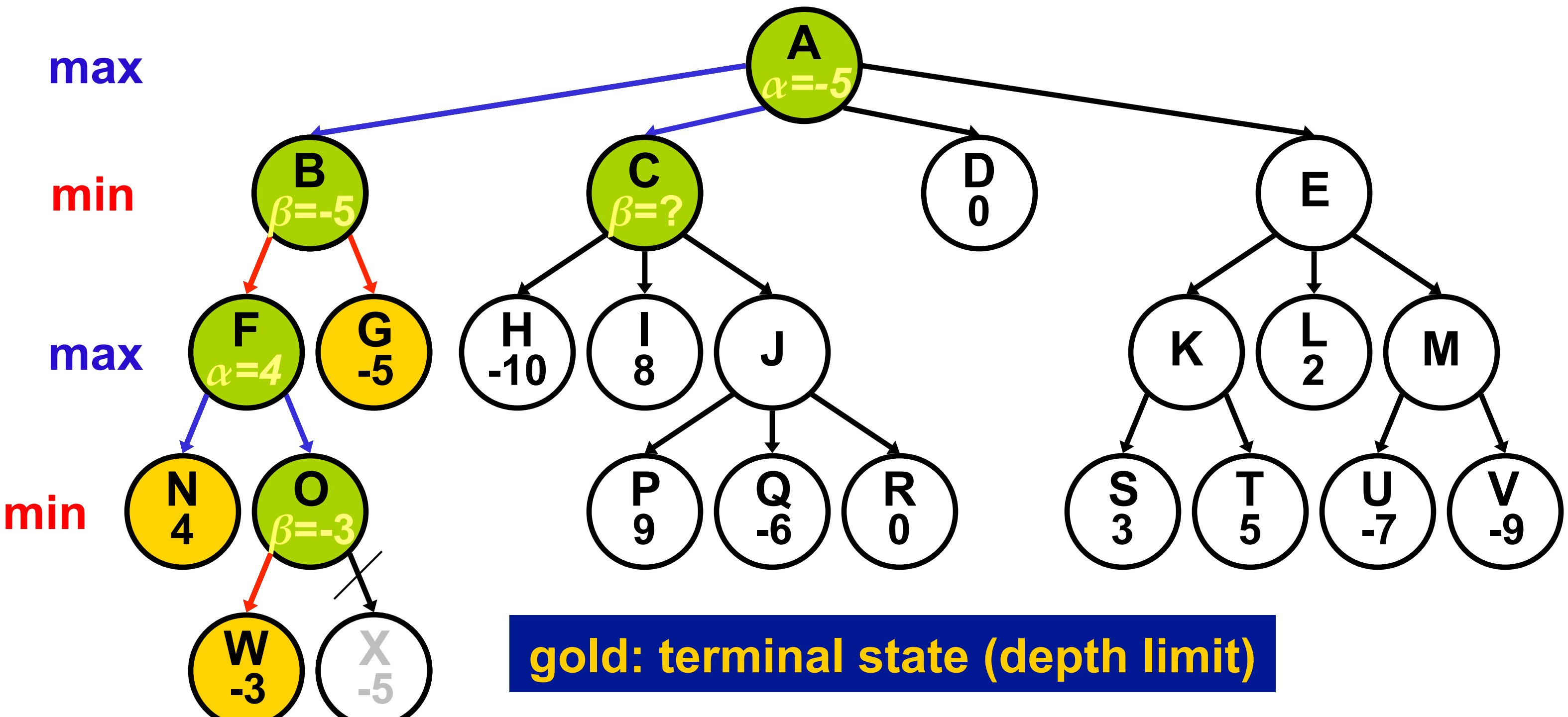
gold: terminal state (depth limit)



Slide adapted from V. Pavlovic

Alpha-Beta Example

`minimax(C,1,4)`



Call Stack

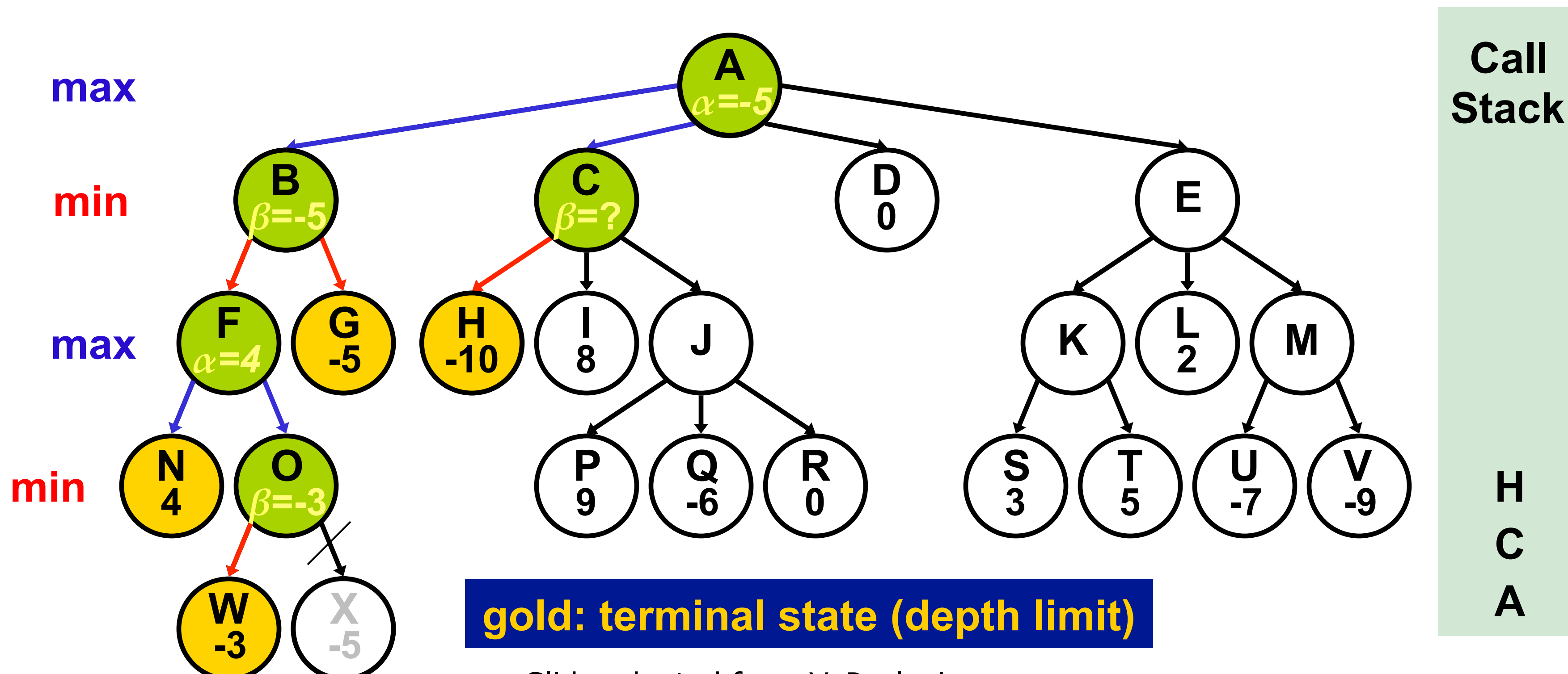
C

A

Slide adapted from V. Pavlovic

Alpha-Beta Example

`minimax(H,2,4)`

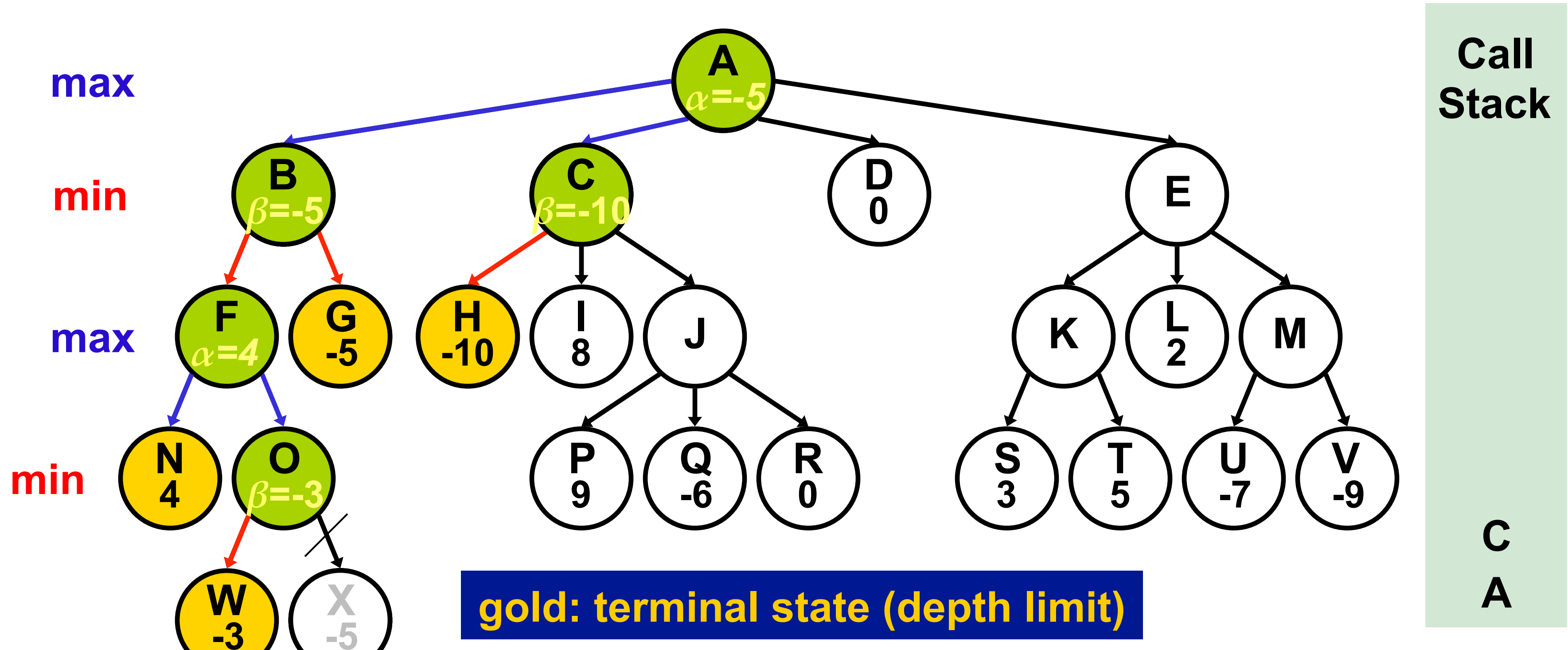


Slide adapted from V. Pavlovic

Alpha-Beta Example

`minimax(C,1,4)` **is returned to**

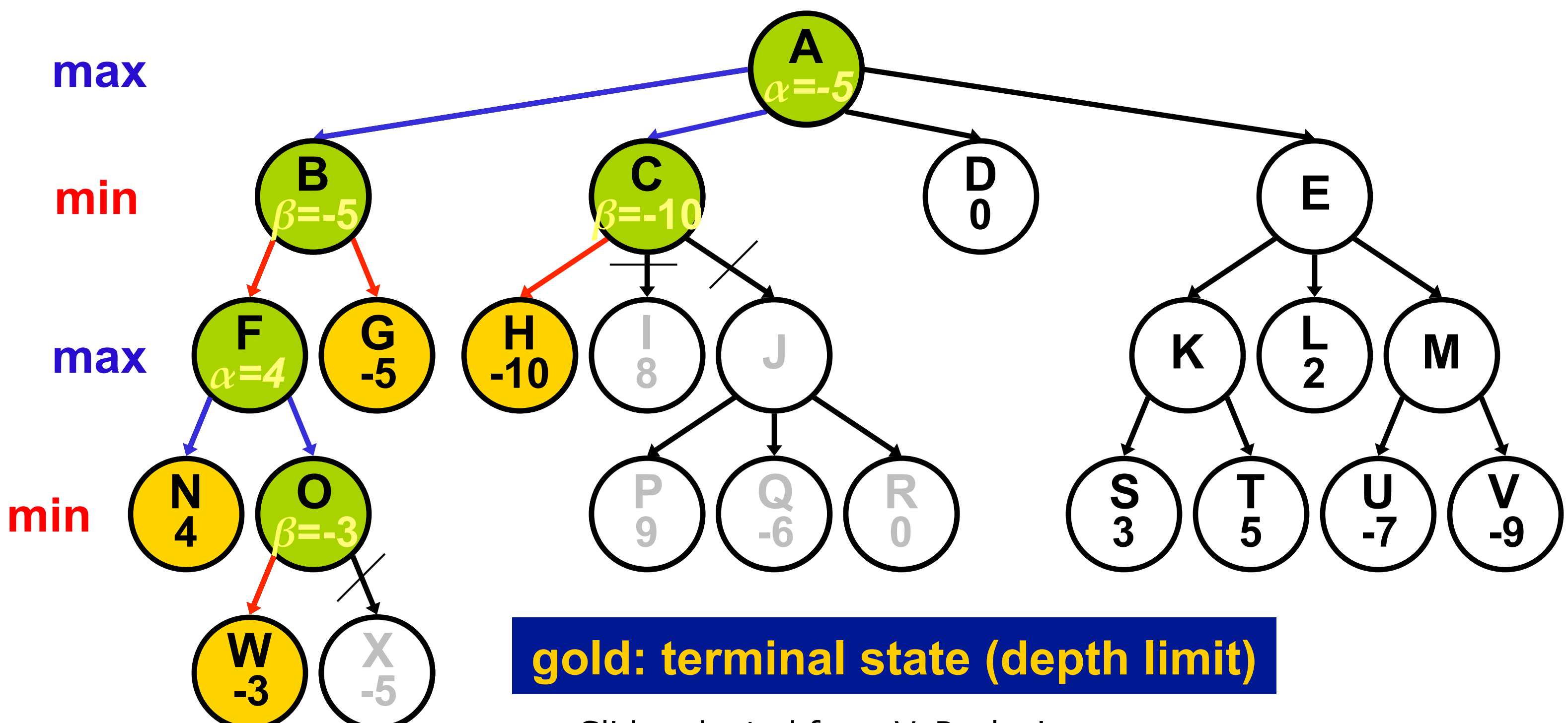
`beta = -10`, minimum seen so far



Slide adapted from V. Pavlovic

Alpha-Beta Example

C's beta (-10) < A's alpha (-5): Stop expanding C (alpha cut-off)



Call Stack

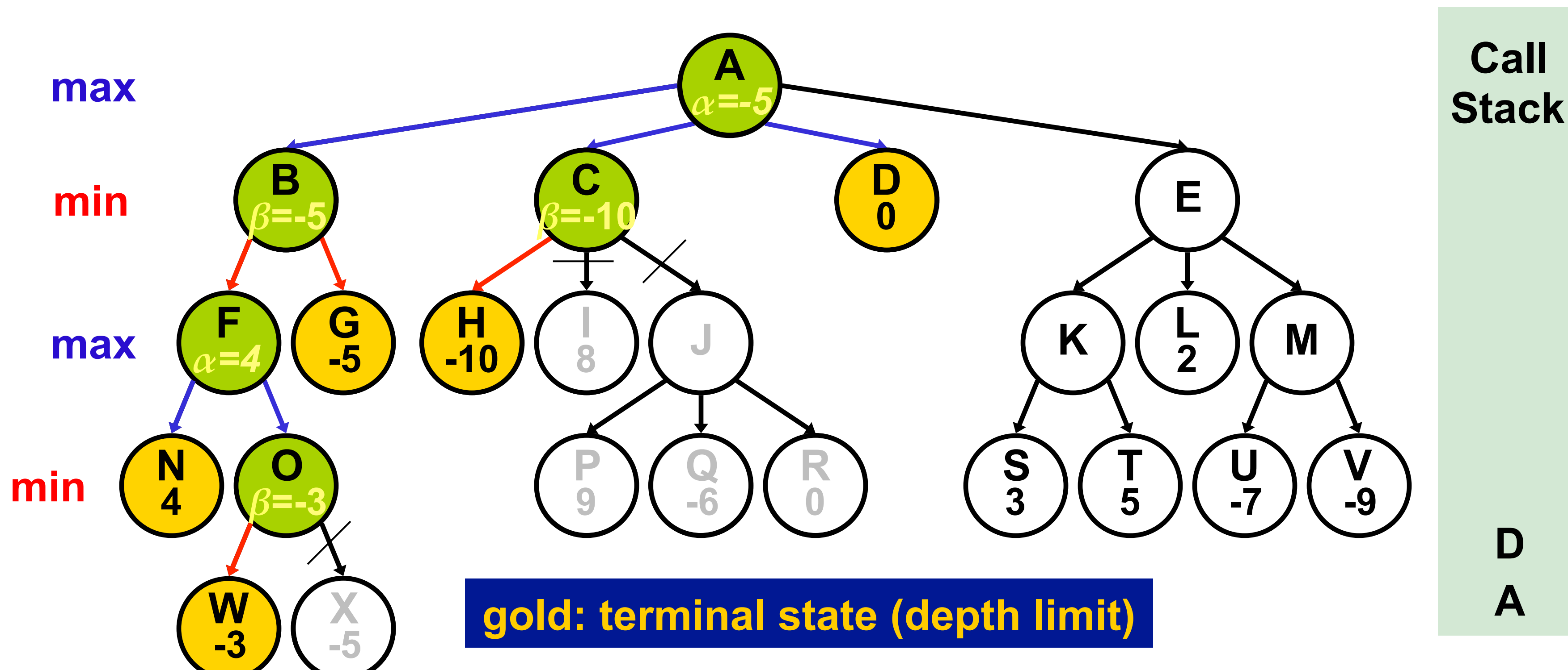
C

A

Slide adapted from V. Pavlovic

Alpha-Beta Example

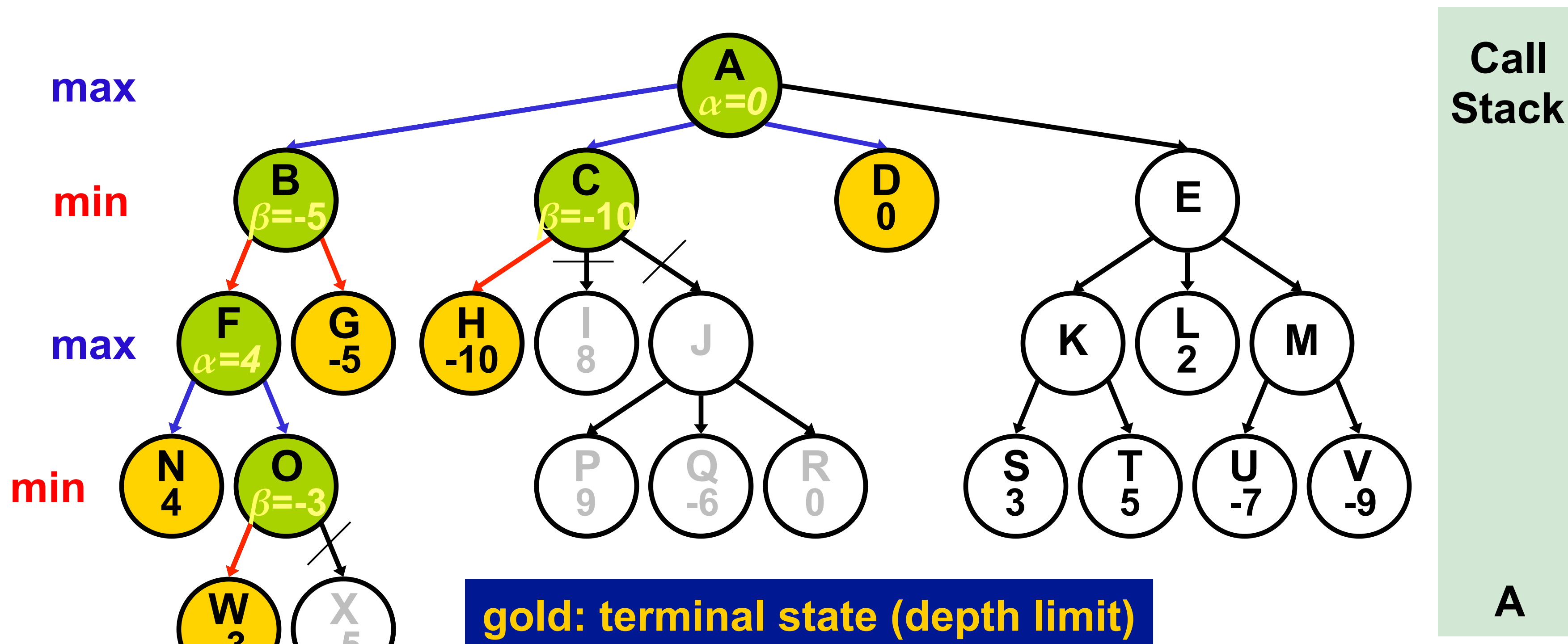
```
minimax(D,1,4)
```



Slide adapted from V. Pavlovic

Alpha-Beta Example

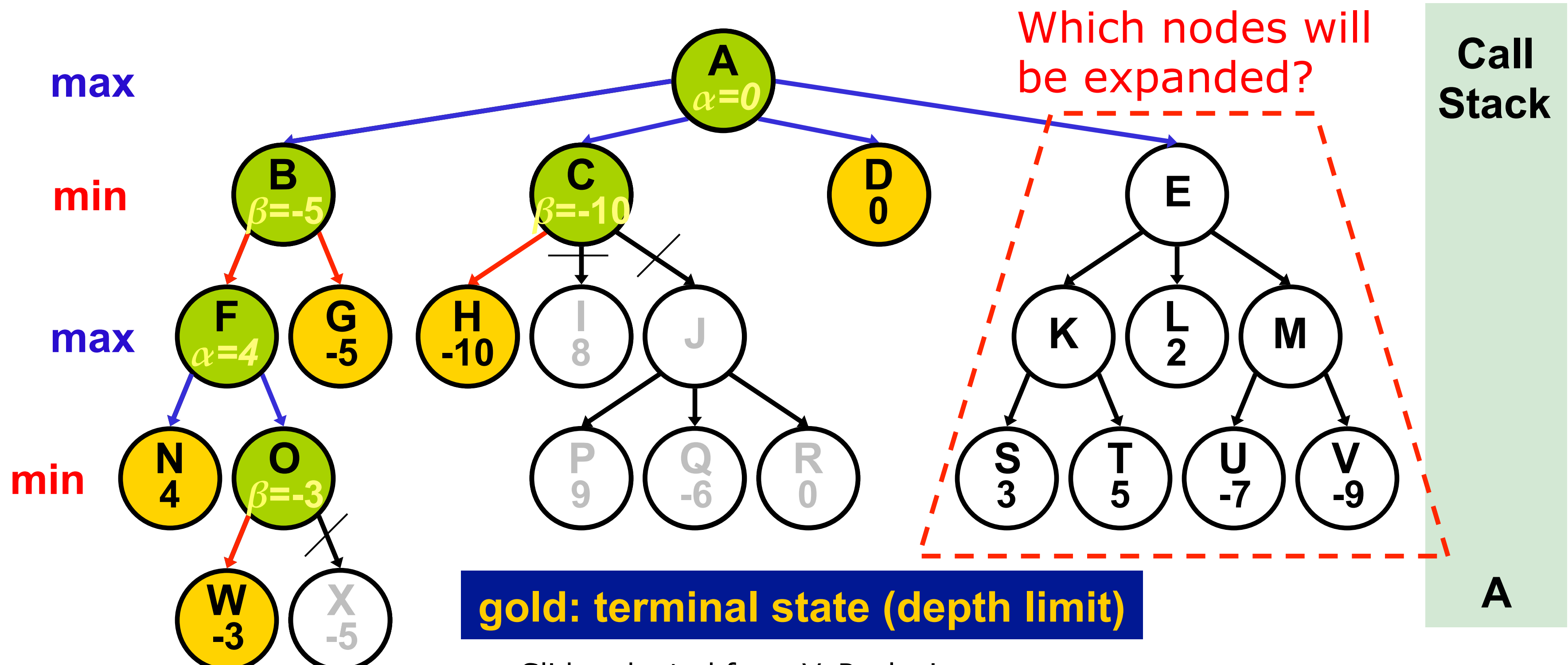
`minimax(D,1,4)` **is returned to**



Slide adapted from V. Pavlovic

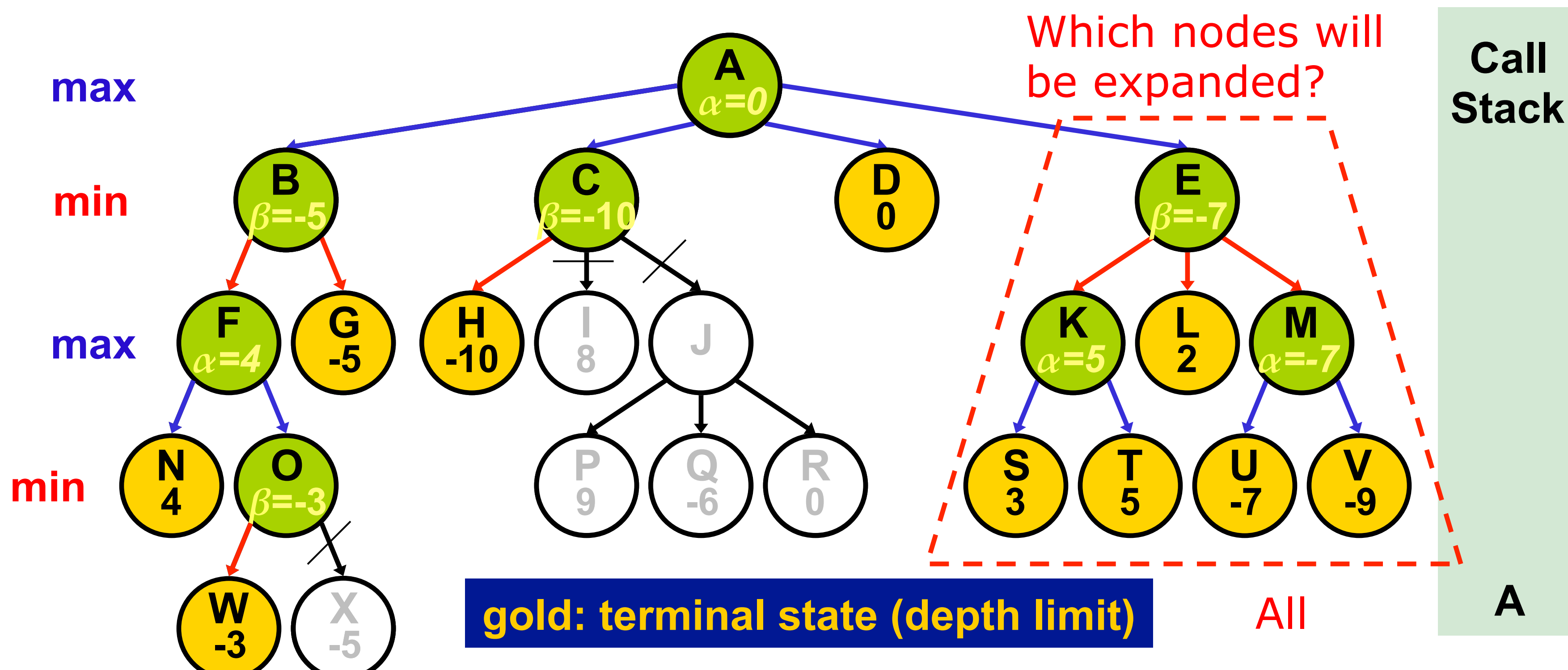
Alpha-Beta Example

`minimax(D,1,4)` **is returned to**



Slide adapted from V. Pavlovic

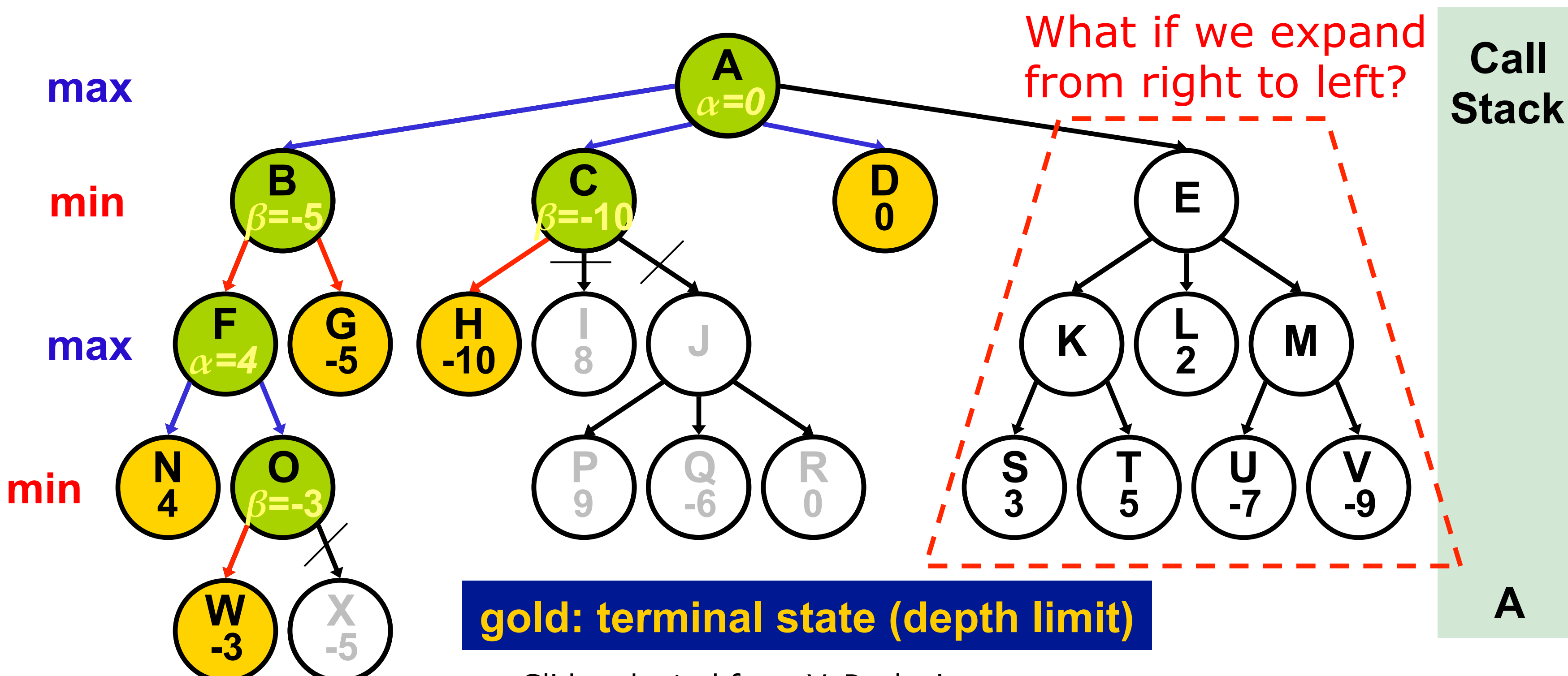
Alpha-Beta Example



Slide adapted from V. Pavlovic

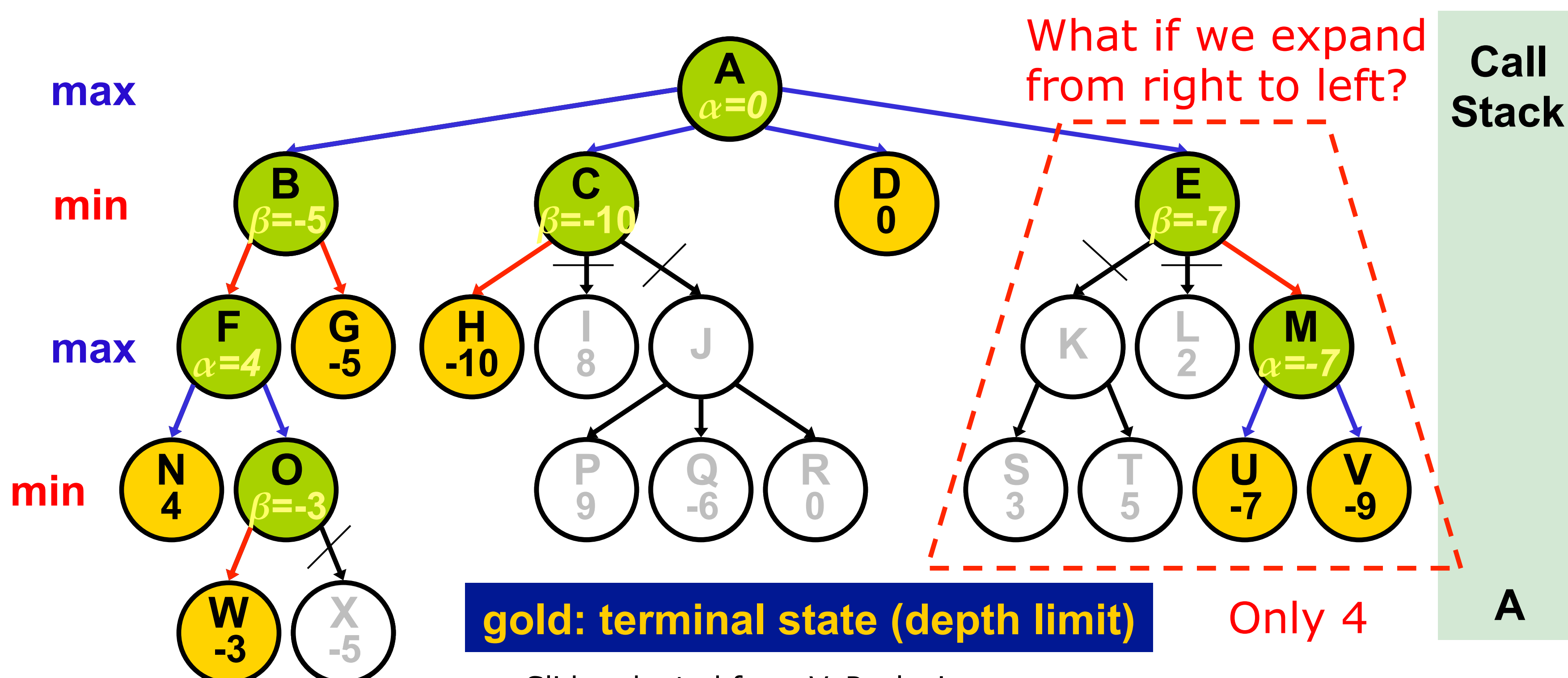
Alpha-Beta Example

`minimax(D,1,4)` **is returned to**



Slide adapted from V. Pavlovic

Alpha-Beta Example



Slide adapted from V. Pavlovic

Alpha-Beta pruning rule

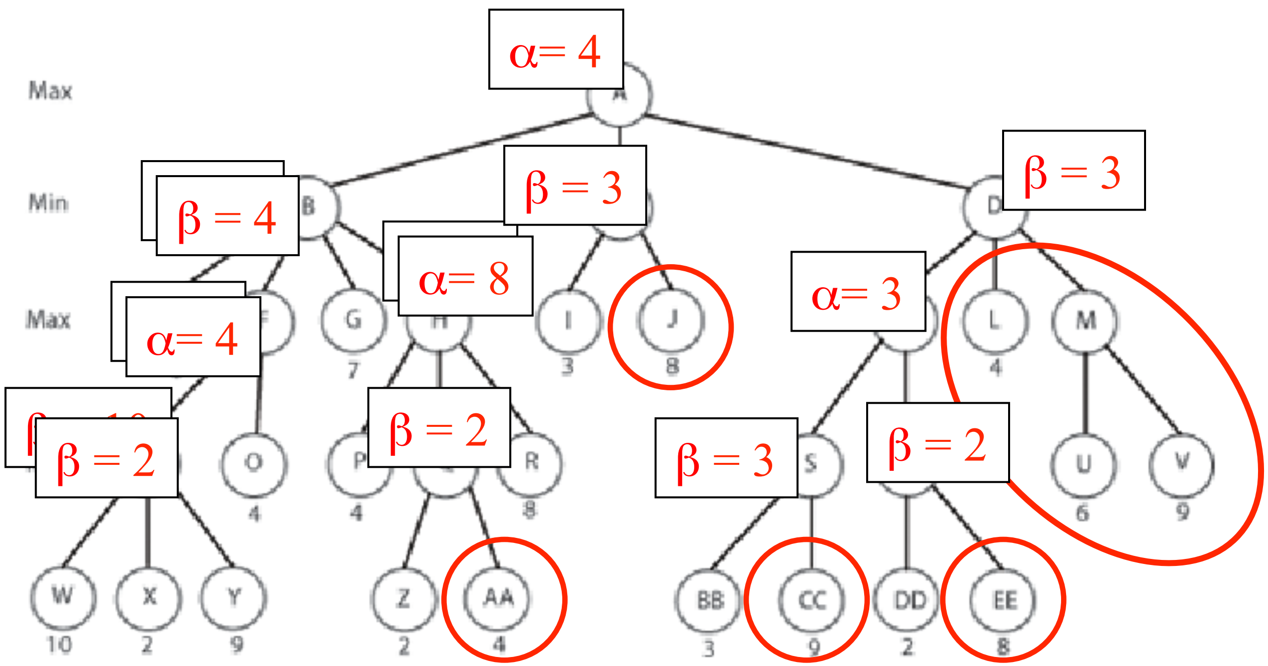
Stop expanding

max node n if $\alpha(n) > \beta$ higher in the tree

min node n if $\beta(n) < \alpha$ higher in the tree

Alpha-Beta pruning rule

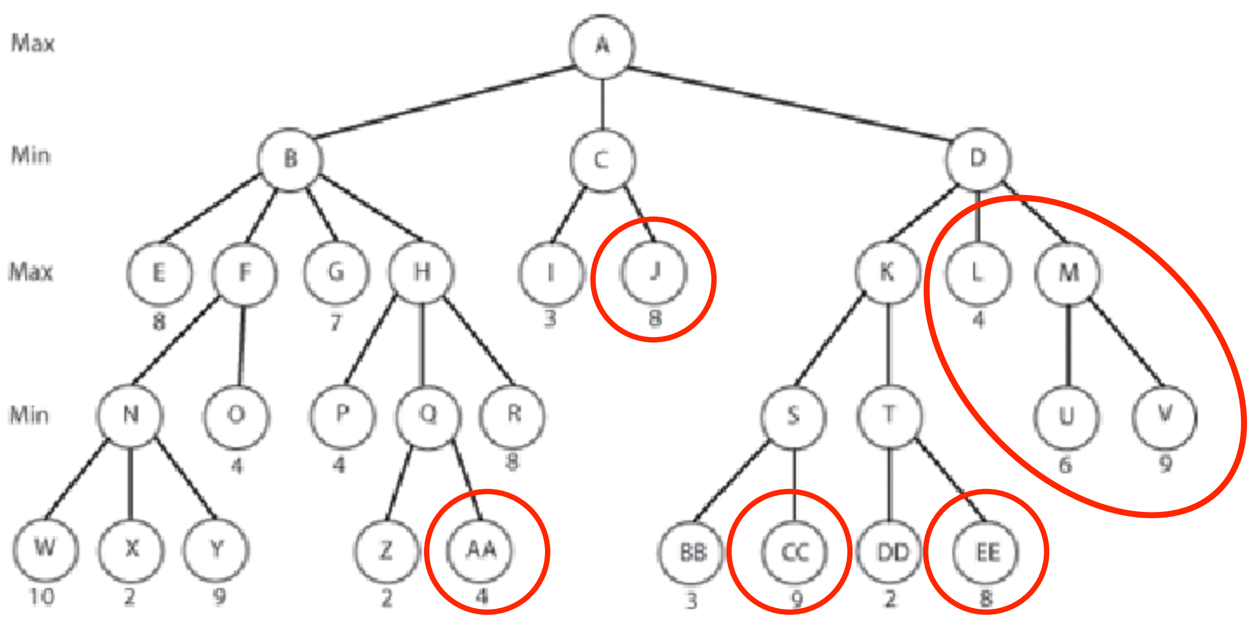
Stop expanding
- max node n if $\alpha(n) > \beta$ higher in the tree
- min node n if $\beta(n) < \alpha$ higher in the tree



Which nodes will not be expanded when expanding from left to right?

Alpha-Beta pruning rule

Stop expanding
  max node n if $\alpha(n) > \beta$ higher in the tree
  min node n if $\beta(n) < \alpha$ higher in the tree



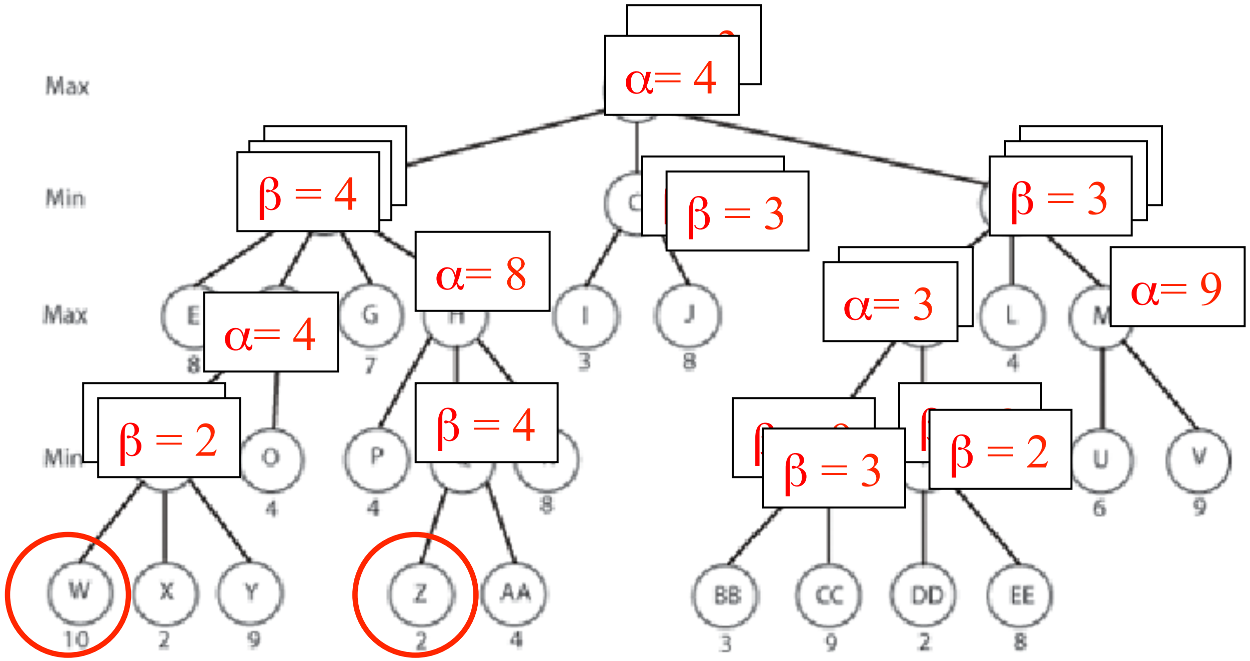
Which nodes will not be expanded when expanding from left to right?

Alpha-Beta pruning rule

Stop expanding

max node n if $\alpha(n) > \beta$ higher in the tree

min node n if $\beta(n) < \alpha$ higher in the tree



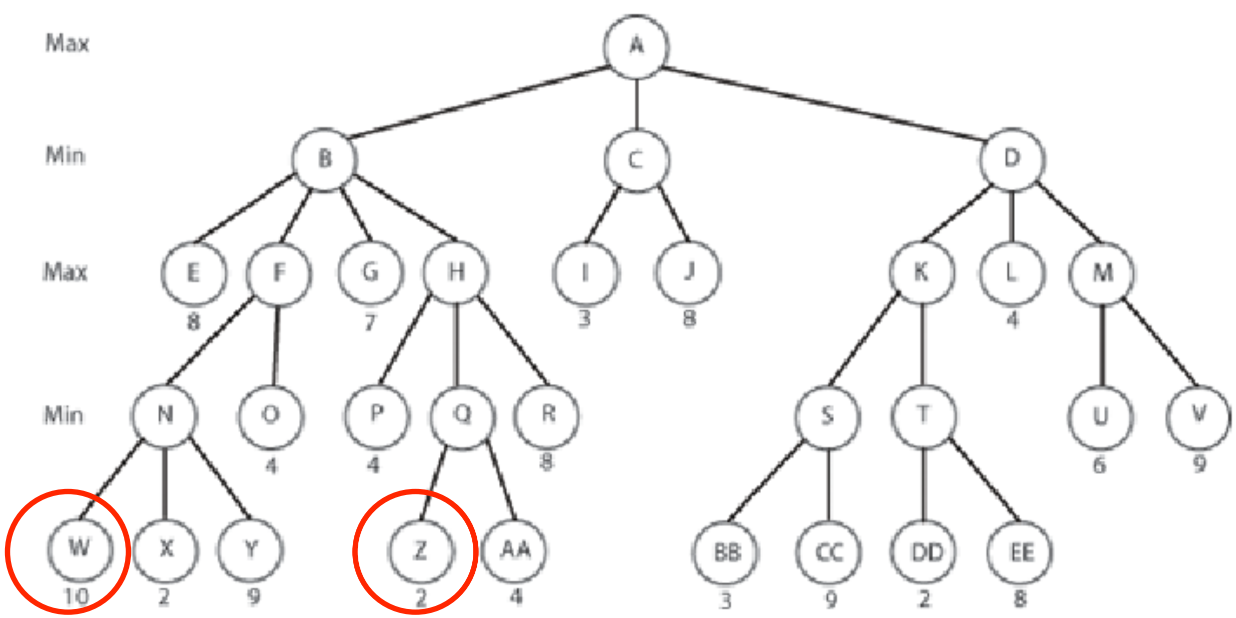
Which nodes will not be expanded when expanding from right to left?

Alpha-Beta pruning rule

Stop expanding

max node n if $\alpha(n) > \beta$ higher in the tree

min node n if $\beta(n) < \alpha$ higher in the tree



Which nodes will not be expanded when expanding from right to left?

3. Cut the search short

- Use depth-limit and estimate utility for non-terminal nodes (evaluation function)
  - Static board evaluation (SBE)
  - Must be easy to compute

Example, chess:

$$SBE = \alpha\text{"Material Balance"} + \beta\text{"Center Control"} + \gamma\ldots$$

Material balance = value of white pieces – value of black pieces, where pawn = +1, knight & bishop = +3, rook = +5, queen = +9, king = ?

The parameters (α,β,γ,...) can be learned (adjusted) from experience.

http://en.wikipedia.org/wiki/Computer_chess

Leaf evaluation

For most chess positions, computers cannot look ahead to all final possible positions. Instead, they must look ahead a few plies and then evaluate the final board position. The algorithm that evaluates final board positions is termed the "evaluation function", and these algorithms are often vastly different between different chess programs.

Nearly all evaluation functions evaluate positions in units and at the least consider material value. Thus, they will count up the amount of material on the board for each side (where a pawn is worth exactly 1 point, a knight is worth 3 points, a bishop is worth 3 points, a rook is worth 5 points and a queen is worth 9 points). The king is impossible to value since its loss causes the loss of the game. For the purposes of programming chess computers, however, it is often assigned a value of appr. 200 points.

Evaluation functions take many other factors into account, however, such as pawn structure, the fact that doubled bishops are usually worth more, centralized pieces are worth more, and so on. The protection of kings is usually considered, as well as the phase of the game (opening, middle or endgame).

Evaluation function

$$f(n) = w_1 F_1(n) + w_2 F_2(n) + \ldots + w_M F_M(n)$$

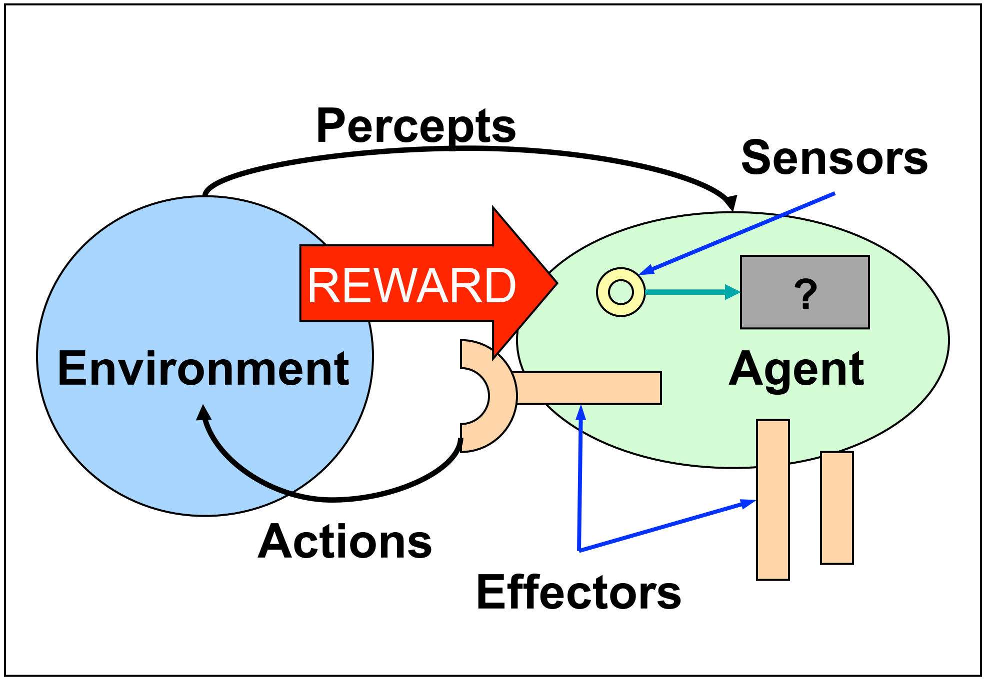
- Here w_i are weighting factors and F_i are features (for position n), e.g. number of pawns, knights, control over central squares, etc.
- Assumes independence (that features are additive and don't interact)

Evaluation function: Deep Fritz Chess

- Employ a ”null move” strategy: MAX is allowed two moves (MIN does not move at all in between).
  - If the evaluation function after these two steps is not high – then don’t search further along this path.
  - Saves time (doesn’t generate any MIN move and cuts off many useless searches)

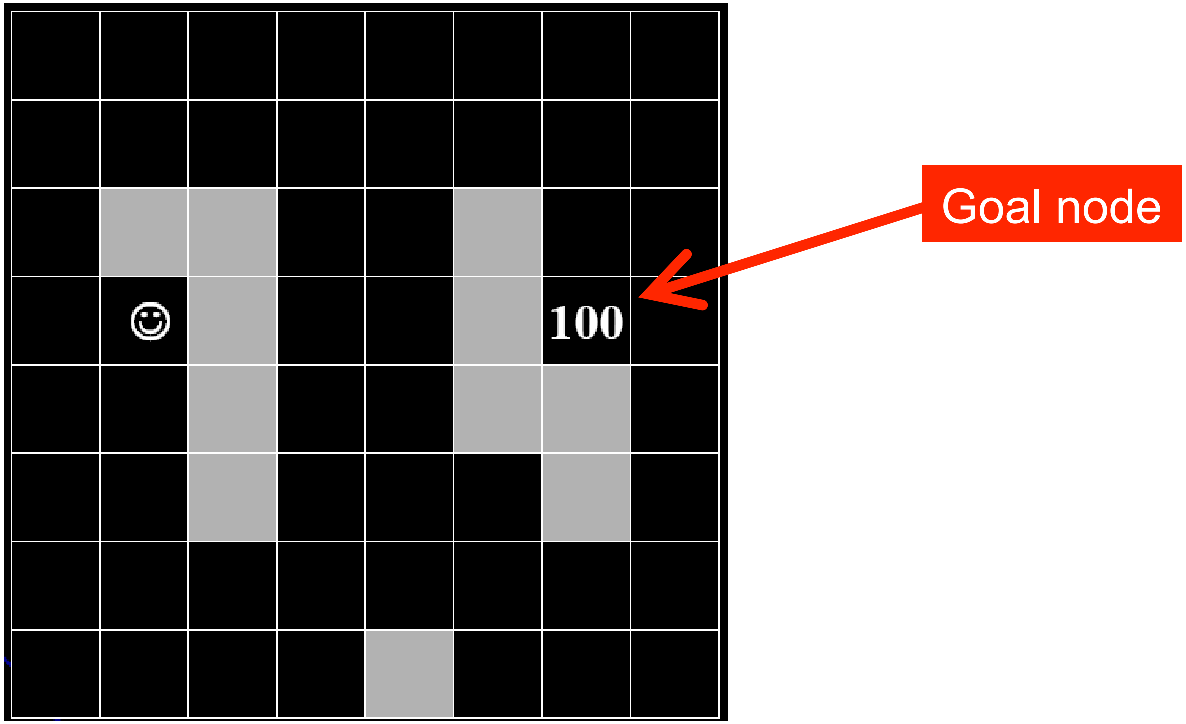
Reinforcement learning

- A method to learn an evaluation function (e.g. For Chess: learn the weights w_i).
- Reinforcement learning is about receiving feedback from the environment (occasionally) and updating the values when this happens.



Reinforcement learning example

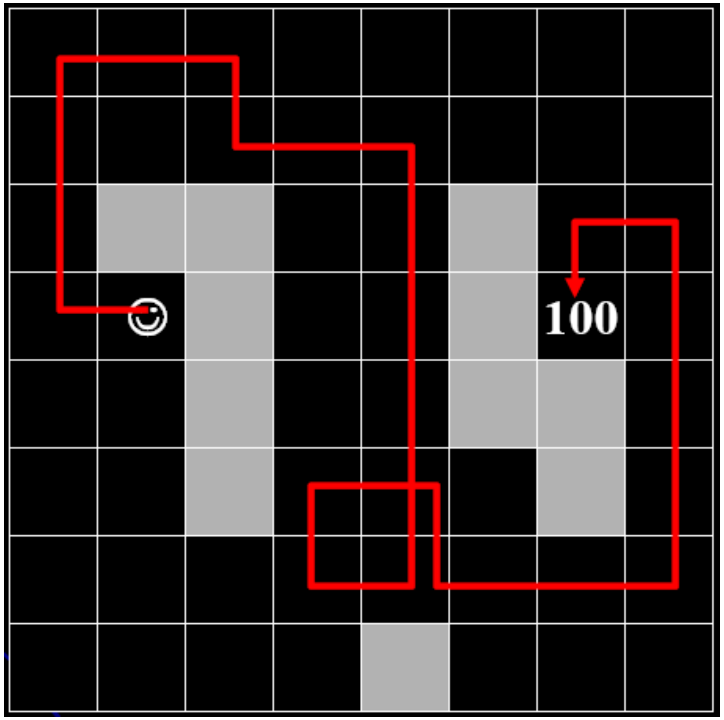
Robot learning to navigate in a maze



Example borrowed from Matthias R. Brust, Univ. Luxemburg

Reinforcement learning example

Robot learning to navigate in a maze

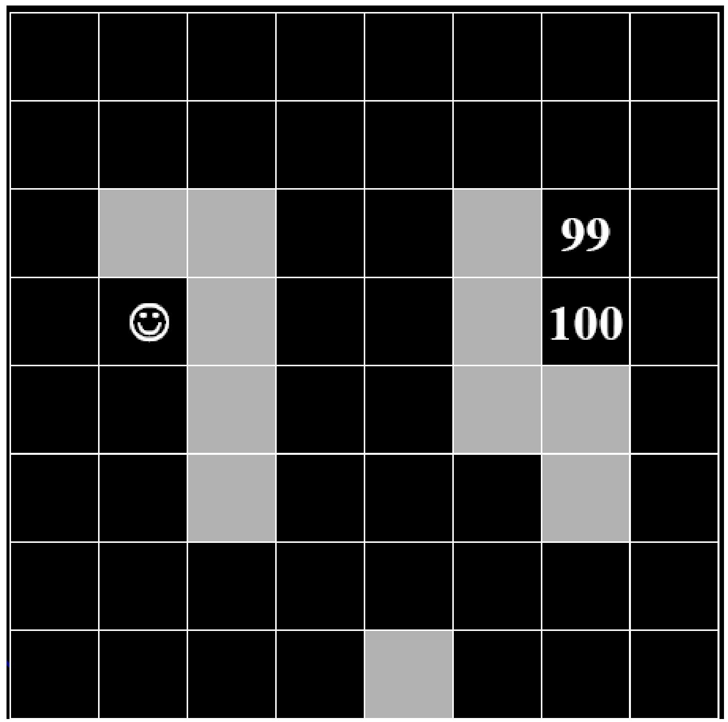


Generate a path at random and run until the goal node is reached.

Example borrowed from Matthias R. Brust, Univ. Luxemburg

Reinforcement learning example

Robot learning to navigate in a maze

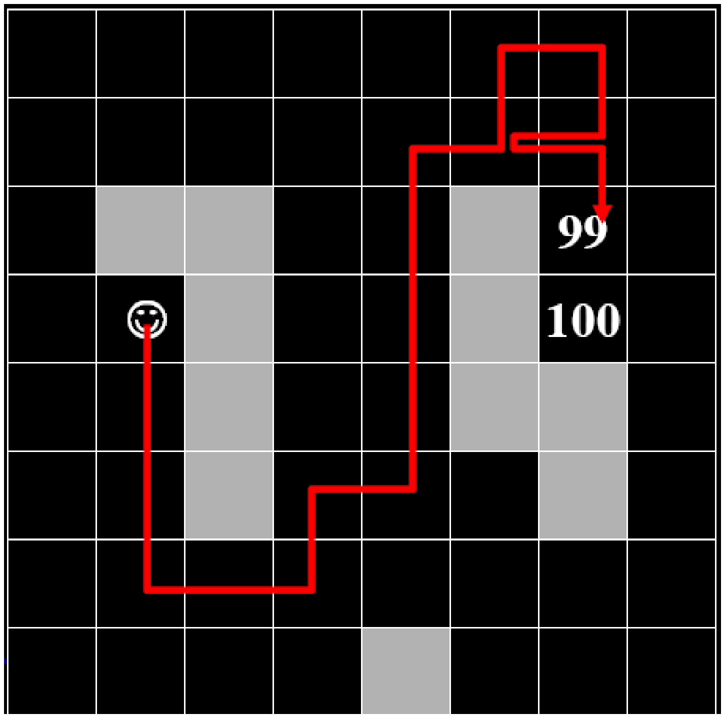


Assign a bit of the goal node's utility value
to the next last square (the square just before
we reached the goal node).

Example borrowed from Matthias R. Brust, Univ. Luxemburg

Reinforcement learning example

Robot learning to navigate in a maze

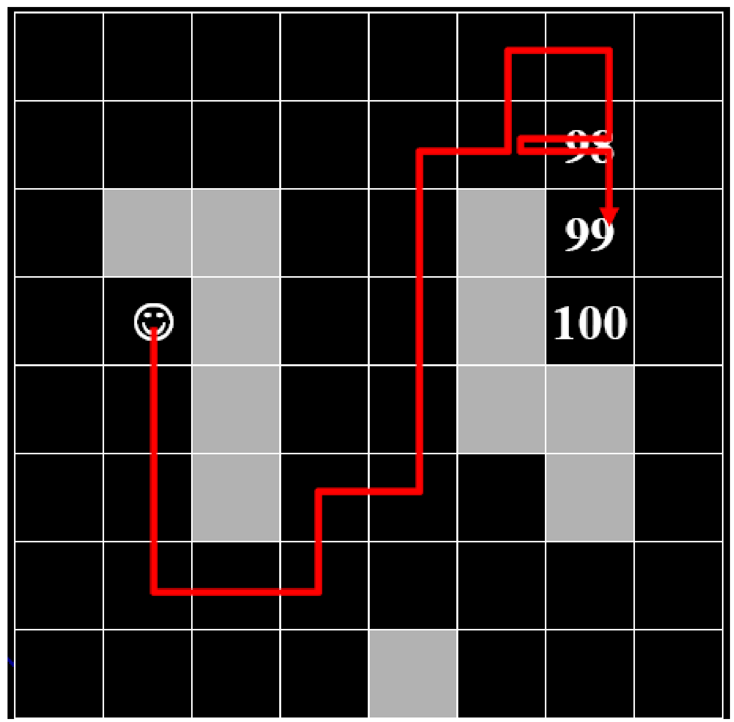


Generate a new random path and run
until a square with utility value is encountered.

Example borrowed from Matthias R. Brust, Univ. Luxemburg

Reinforcement learning example

Robot learning to navigate in a maze



Assign a bit of the utility value to the next last square...etc.

Example borrowed from Matthias R. Brust, Univ. Luxemburg

Reinforcement learning example

Robot learning to navigate in a maze

91	92	93	94	95	96	97	96
92	93	94	95	96	97	98	97
91			94	95		99	98
90	☺		93	94		100	99
89	88		92	93			98
88	89		91	92	93		97
89	90	91	92	93	94	95	96
88	89	90	91		93	94	95

After some (a long) time do we have utility estimates of all squares.

Example borrowed from Matthias R. Brust, Univ. Luxemburg

KnightCap (1997)

http://samba.org/KnightCap/

- Uses reinforcement learning to learn an evaluation function for Chess.
- Initial values for pieces:
  - 1 for a pawn
  - 4 for a knight
  - 4 for a bishop
  - 6 for a rook
  - 12 for a queen
- After self-learning:
  - 1 for a pawn
  - 3 for a knight
  - 3 for a bishop
  - 5 for a rook
  - 9 for a queen

Position (control, number of pieces attacking king) features crucial

4. Book moves

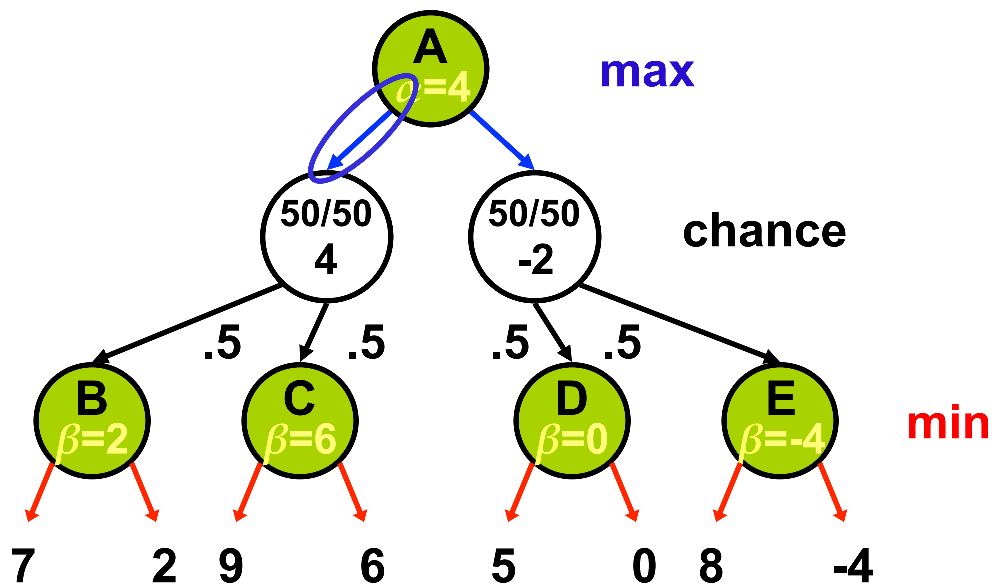
- Build a database (look-up table) of endgames, openings, etc.
- Use this instead of minimax when possible.

Games with chance

- Dice games, card games,...
- Extend the minimax tree with chance layers.

Compute the expected value over outcomes.

Select move with the highest expected value.



Animation adapted from V. Pavlovic