


Chapter 5

Adversarial Search



Jane Hsu
National Taiwan University

Acknowledgements: This presentation is created by Jane hsu based on the lecture slides from *The Artificial Intelligence: A Modern Approach* by Russell & Norvig, a PowerPoint version by Han-Shen Huang, as well as various materials from the web.

In which we examine the problems that arise when we try to plan ahead in a world where other agents are planning against us.



Turing and Chess-Playing Programs

- Alan Turing talked with Jack Good at Bletchley Park about the so-called chess-playing programs.
 - They got the idea of *searching decision trees* for the *best* move.
- Later (during WWII), he talked with a young student, Donald Michie, about the prospect of machines “learning.”
- After 1945, he often used chess-playing as an example of what a computer could do.
- In his 1946 report on the possibilities of a computer, Turing made his first reference to machine “intelligence” in connection with chess-playing.
- In 1948, Turing met Donald Michie again and competed with him in writing a simple chess-playing algorithm.

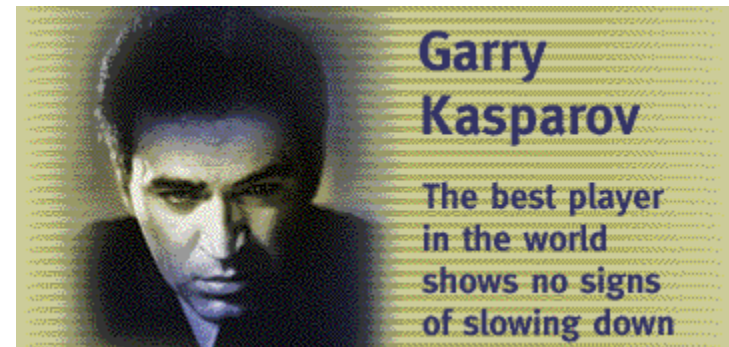
A Bit More History

- Computer chess programs
 - An early program for IBM mainframes in 1958
 - Another one for \$10-million Cray supercomputers by the 1970s
 - Commercial programs contributed to the demand for the first personal computers in the 1980s.
- John McCarthy, a leading AI expert and Stanford emeritus professor, built a machine that defeated a Soviet computer chess team in 1965.

“The Soviets had a better program but a worse computer!”
- David Levy, a British grandmaster, won a series of bets against chess computers in the 1970s and 1980s that inspired programmers to refine their machines
- The IBM team (Murray Campbell, FH Hsu etc.) built Deep Blue.

The Historical Match [1997]

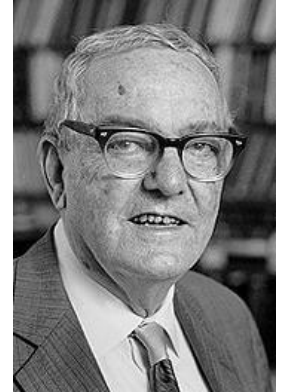
- Kasparov vs. Deep Blue
- The Match
 - May 3~11, 1997
 - Deep Blue won in 6 games



- From a different view

No Triumph for AI

- In 1957, the AI pioneer Herbert Simon predicted that a machine would be chess champion of the world within 10 years.
- He was off by three decades.
- More importantly, however, his prediction of how computers would solve chess proved to be entirely wrong — to artificial intelligence's enduring **chagrin**.



苦惱

Differences

1. Chess positions per second: up to 200,000,000 vs. 3
2. Amount of chess knowledge vs. amount of calculation ability
3. Sense of feeling and intuition
4. Guidance of five IBM research scientists and one international grandmaster vs. personal coach Yuri Dokhoian
5. Learning and adaptation

Differences (continued)

- ❑ Human frailties: forgetfulness, distraction, intimidation, fatigue, boredom and loss of concentration.
- ❑ Task-specific
- ❑ Changes by development team vs. self-modification
- ❑ evaluating its opponent's weaknesses
- ❑ Exhaustive vs. selective search through the possible positions

Outline

Alpha-Beta Pruning

MiniMax Search

Faster and Faster

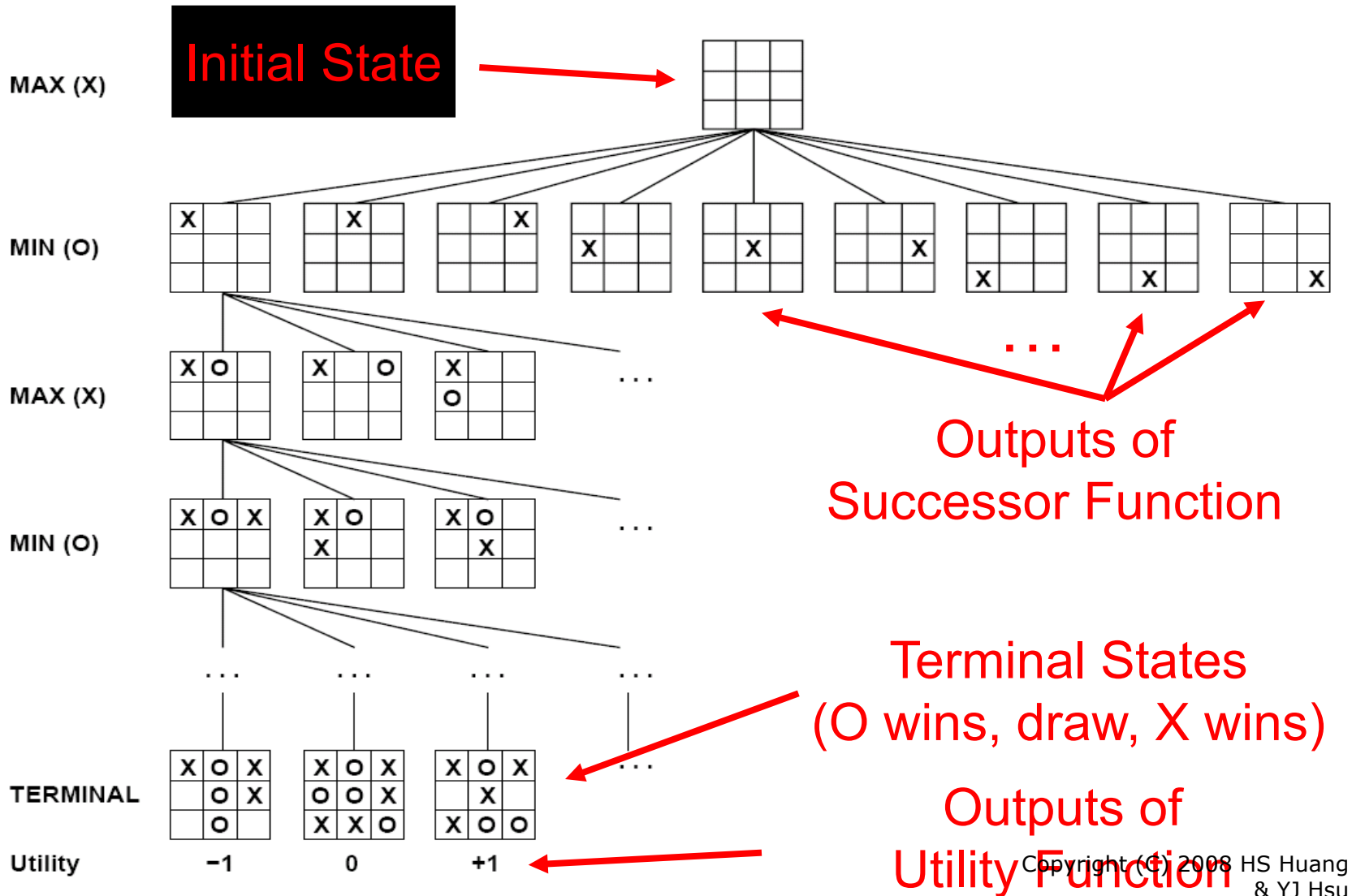
Game
Game Tree
Optimal Strategy

Improving Decision
Quality

Games as Search Problems

- Games are idealization of worlds in which
 - the world state is fully accessible
 - the (small number of) actions are well-defined
 - uncertainty exists due to moves of the opponent, and the complexity of games
- A game can be defined as a search problem:
 - initial state
 - successor function (next moves or board situations)
 - terminal states
 - utility function (chance of win)

Game Tree



When Game Tree Is Huge...

In a typical chess game, the game tree is huge such that exploration is limited within a given depth.

- Average branching factor: 35
- Average moves by a player: 50 (100 plies)
- Average size of a game tree: 35^{100}

If a leaf node is not in a terminal state, an efficiently computable **evaluation function** is used to approximate its utility.

Evaluation Function: Chinese Chess

- Given a board layout B of Chinese chess, the evaluation function consists of:
 - Piece score: king=5000, rook=500, horse=250, cannon=250 ...
 - Position score: based on the position of each piece. The position score of a piece is high if its location is good for attacking and/or defending in most cases.
 - Chinese chess: 中炮, 巡河車, 正馬, 邊馬

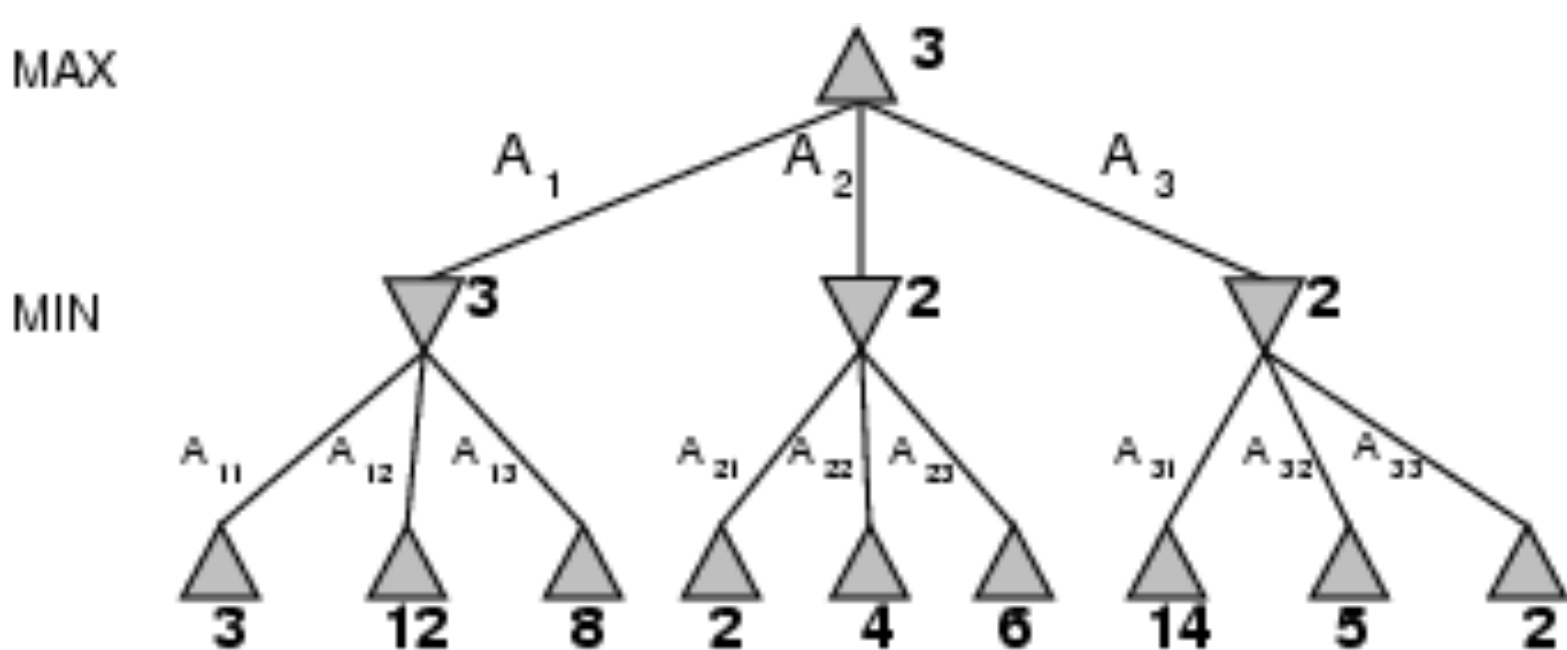
Optimal Strategy to Make a Decision

Given a game tree where every leaf node is evaluated, how can we determine the best move?

- Optimal strategy: assumes that both players play optimally. That is, we maximize our score and the opponent minimizes our score.
- In zero-sum games, the opponent maximizes his score at the same time our score is minimized

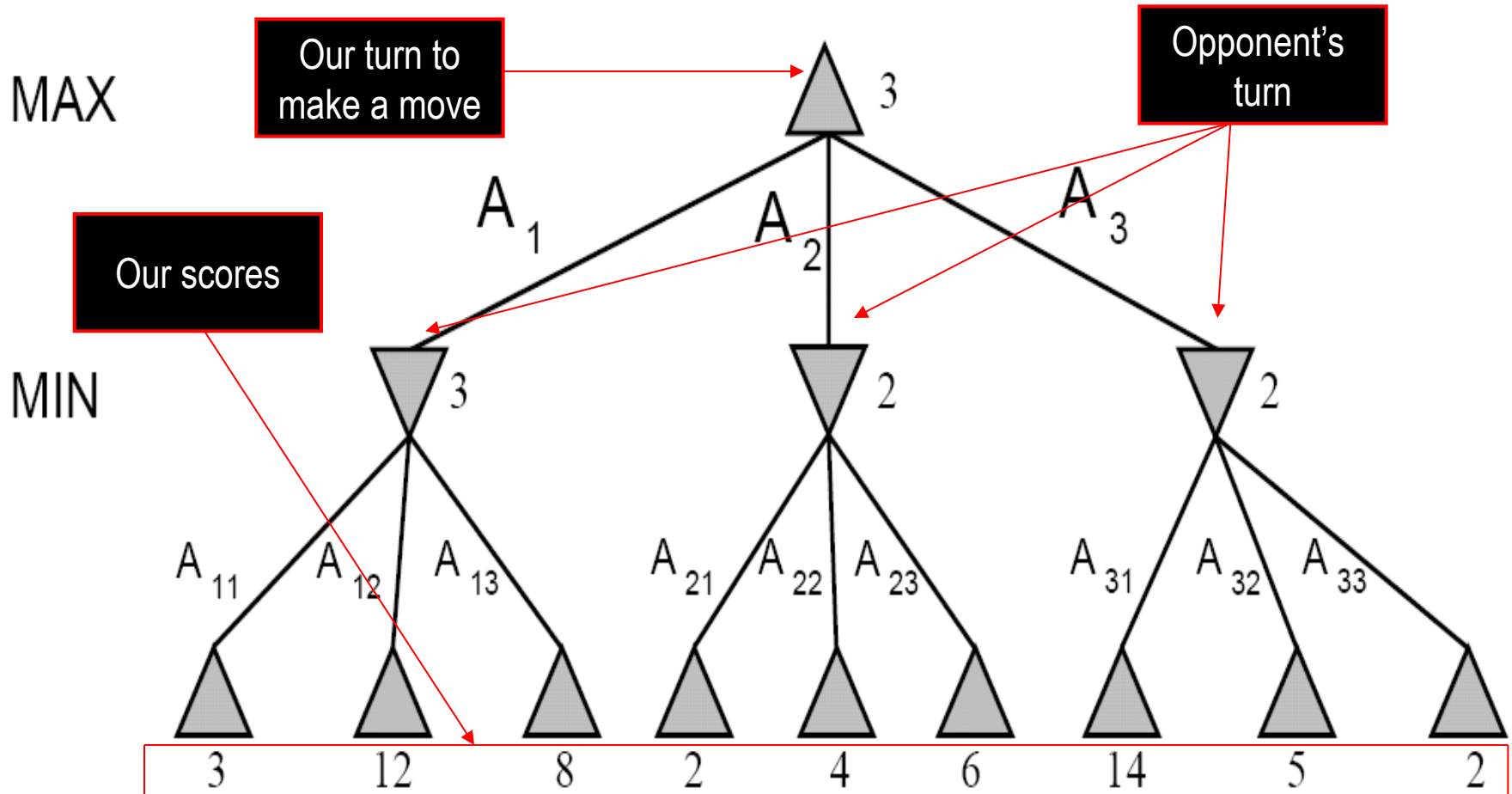
Minimax

- Perfect play for deterministic games
- Idea: choose move to position w/ highest
 - *minimax value* = best achievable payoff against best play
- E.g., 2-ply game tree:



MiniMax Algorithm

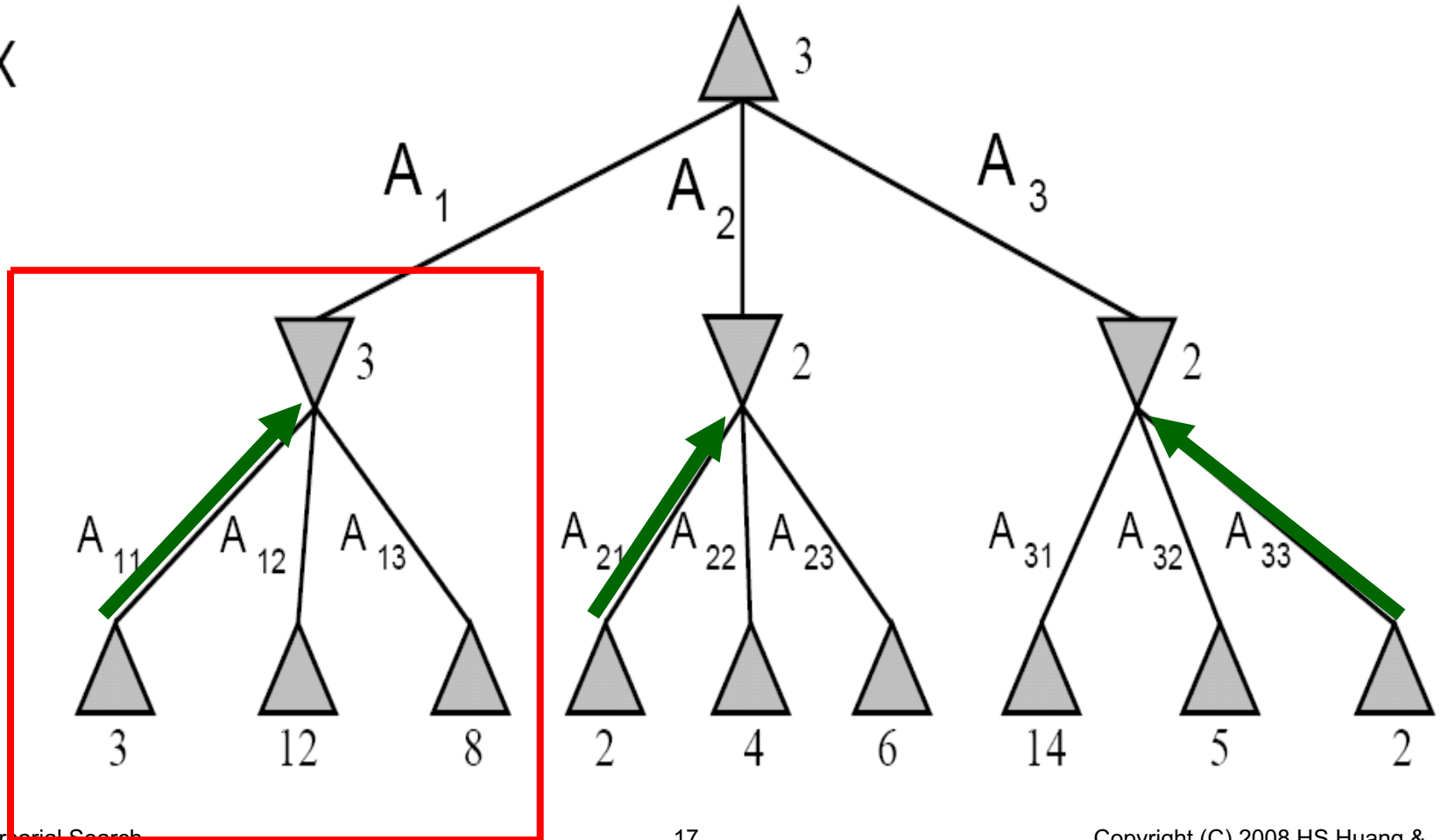
A two-ply game-tree for deterministic games



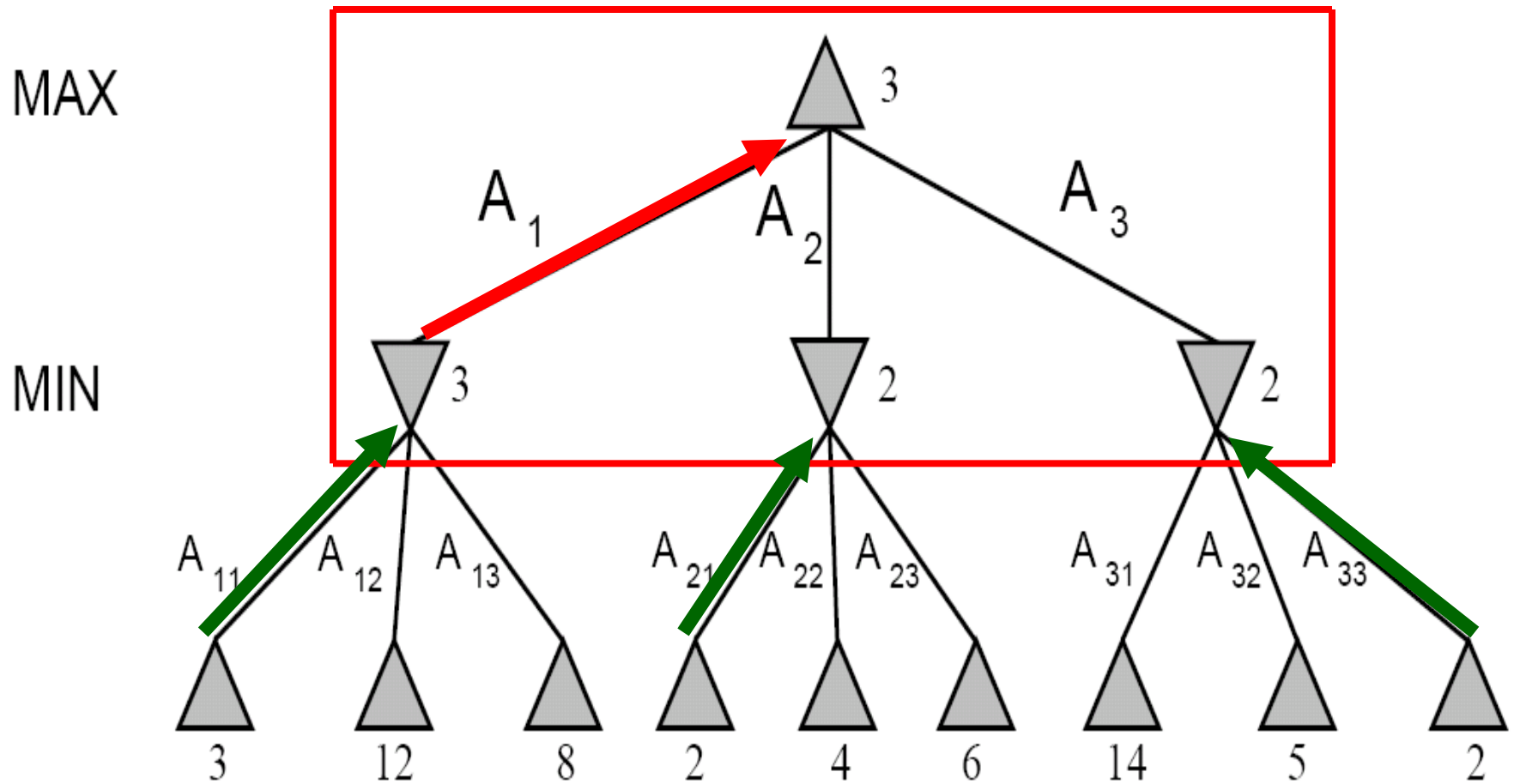
Opponent Minimizes Our Score

MAX

MIN



We Maximize Our Score



Minimax Algorithm

```
function MINIMAX-DECISION(state) returns an action  
  inputs: state, current state in game  
  return the a in ACTIONS(state) maximizing MIN-VALUE(RESULT(a, state))
```

```
function MAX-VALUE(state) returns a utility value  
  if TERMINAL-TEST(state) then return UTILITY(state)  
   $v \leftarrow -\infty$   
  for a, s in SUCCESSORS(state) do  $v \leftarrow \text{MAX}(v, \text{MIN-VALUE}(s))$   
  return v
```

```
function MIN-VALUE(state) returns a utility value  
  if TERMINAL-TEST(state) then return UTILITY(state)  
   $v \leftarrow \infty$   
  for a, s in SUCCESSORS(state) do  $v \leftarrow \text{MIN}(v, \text{MAX-VALUE}(s))$   
  return v
```

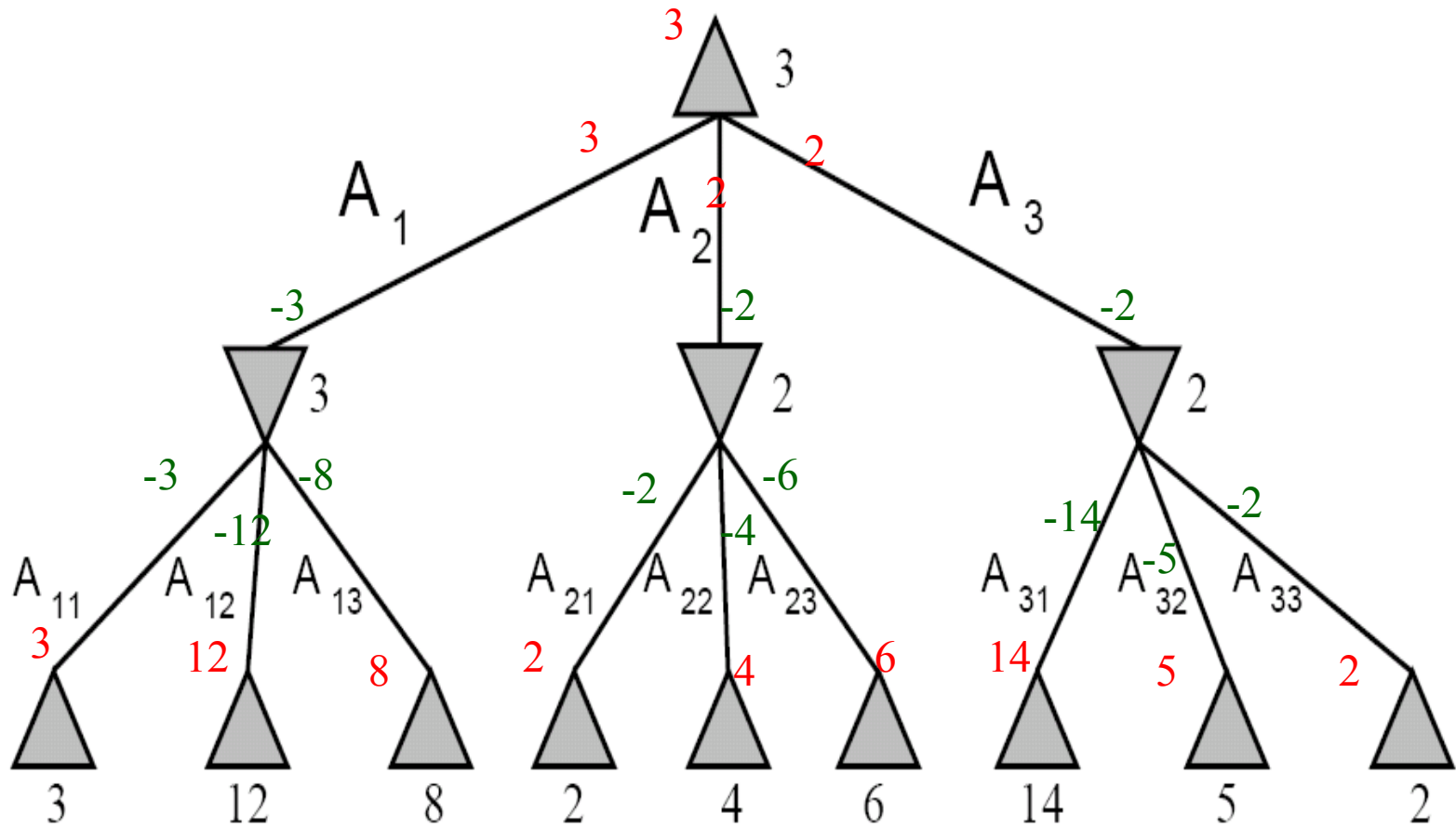
MaxMax Algorithm

- The output is the same as MiniMax
- Players maximize their scores
- Zero-sum: if the score of one player is 10, then the score of the other is -10.
- Advantage of MaxMax over MiniMax:
 - Consistent view: maximize scores
 - Subroutine Min is not required
 - If there are bugs, we only need to modify one subroutine

MaxMax Algorithm

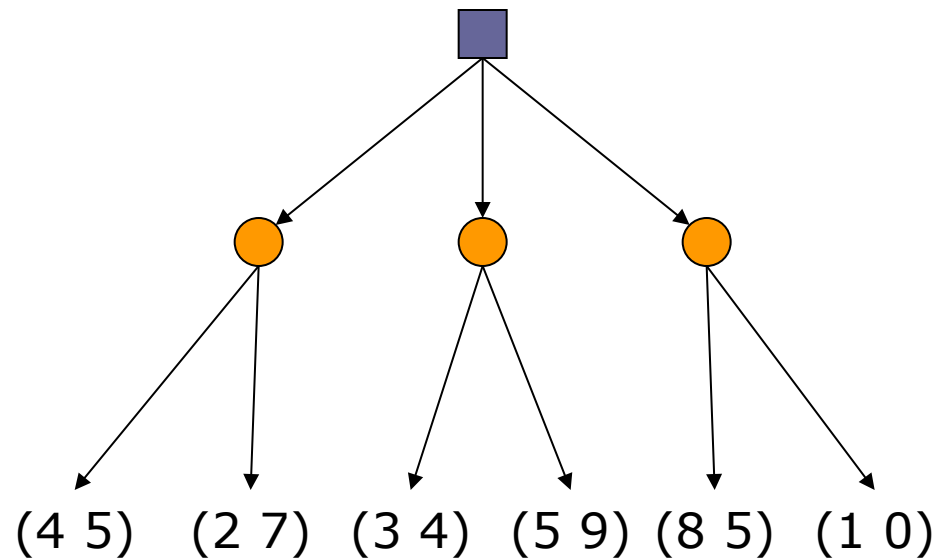
MAX

MIN



Non-Zero-Sum Game

- Each player has his/her own utility function
- Known to each other
- No constraints on utility values between the players
- Strategy: to maximize individual utility
- Does MiniMax work?



Outline

Alpha-Beta Pruning

MiniMax Search

Done!

Faster and Faster

Game
Game Tree
Optimal Strategy

Done!

Improving Decision
Quality

Alpha-Beta Pruning

- Goal: to speed up MiniMax (MaxMax) algorithm, such that it
 - traverses fewer nodes in a game tree, and
 - returns a solution with the same score as MiniMax.

What kind of node (move) can be omitted?

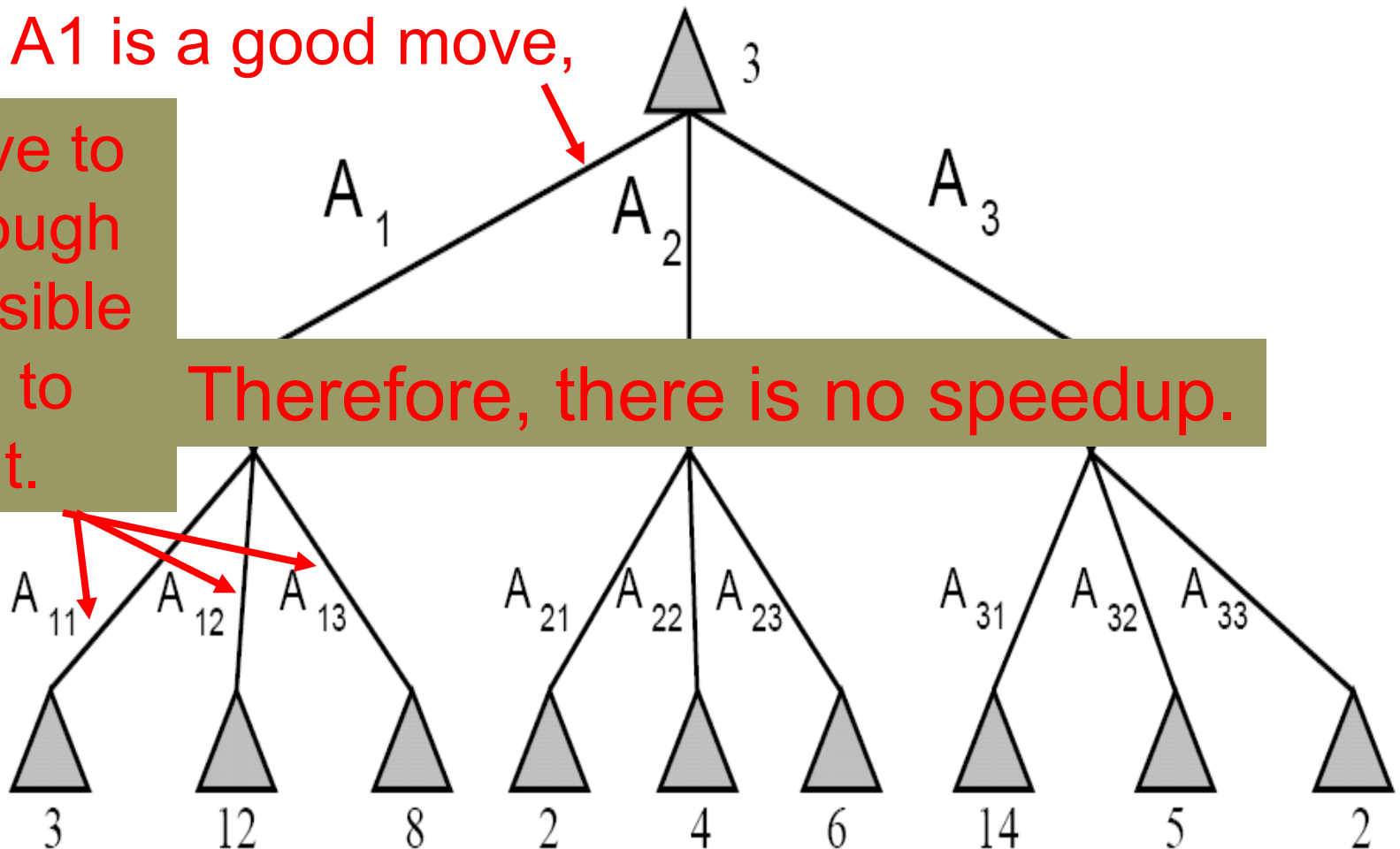
Intuition: Good and Bad Moves

MAX

If A_1 is a good move,

we have to go through all possible moves to prove it.

Therefore, there is no speedup.



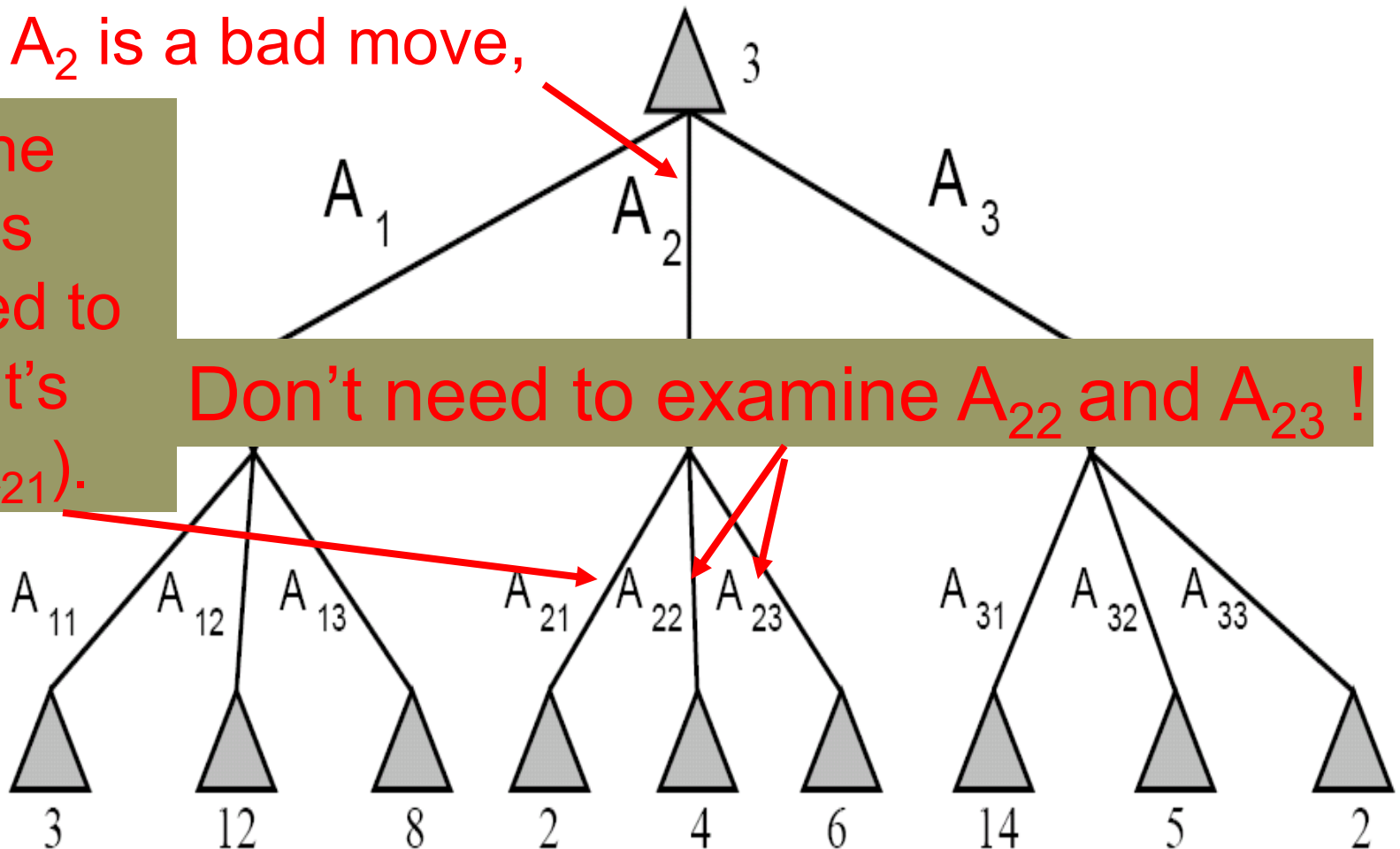
Intuition: Good and Bad Moves

MAX

If A_2 is a bad move,

only one
move is
required to
prove it's
bad (A_{21}).

Don't need to examine A_{22} and A_{23} !



Pruning Example

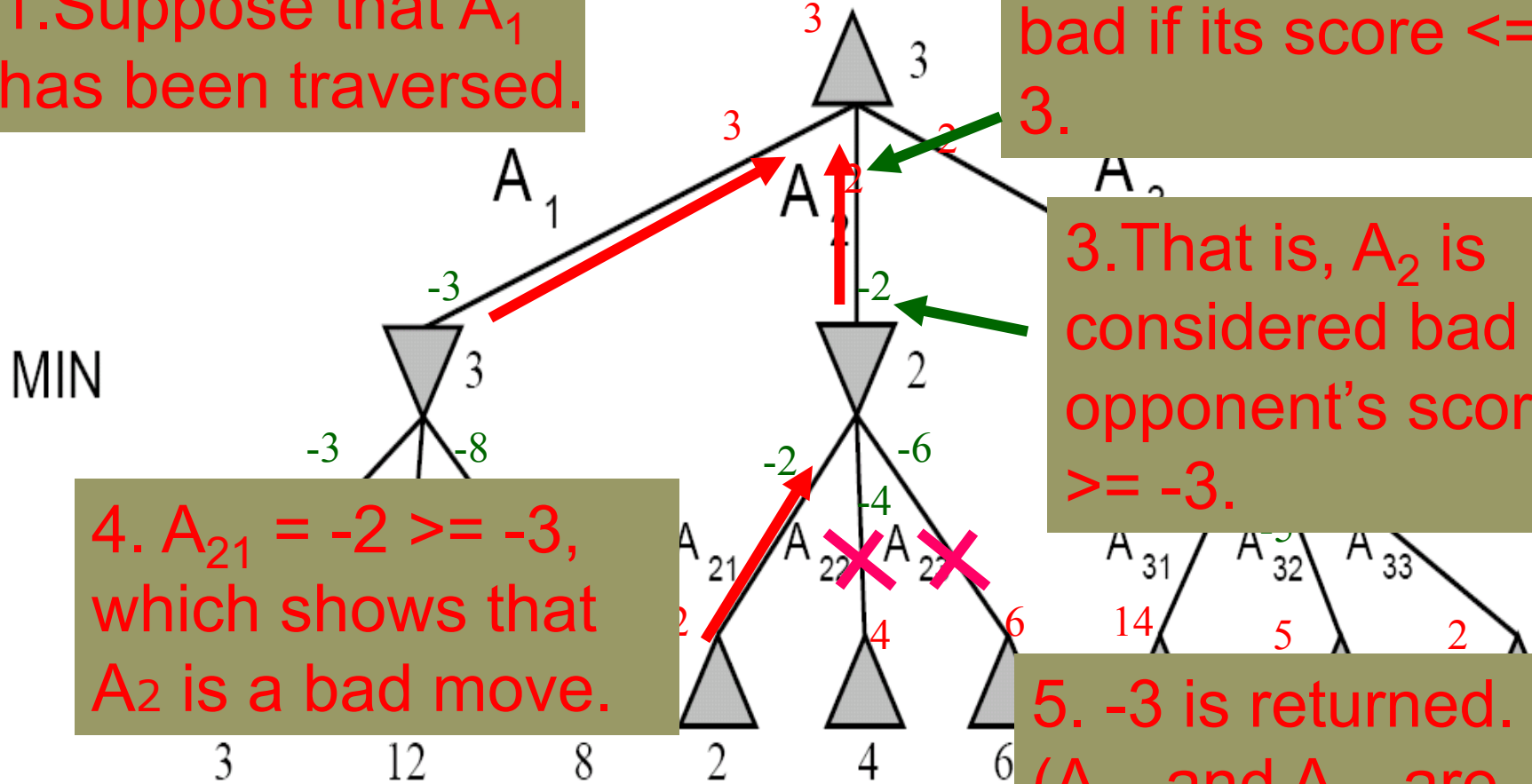
1. Suppose that A_1 has been traversed.

2. A_2 is considered bad if its score ≤ 3 .

3. That is, A_2 is considered bad if opponent's score ≥ -3 .

4. $A_{21} = -2 \geq -3$,
which shows that
 A_2 is a bad move.

5. -3 is returned.
(A_{22} and A_{23} are not traversed.)



Implementation of Pruning

1. Pass the current best score to child nodes
2. Stop searching and return when a branch exceeds the score from the parent

Alpha-Beta Pruning Algorithm

```
function ALPHA-BETA-DECISION(state) returns an action
  return the a in ACTIONS(state) maximizing MIN-VALUE(RESULT(a, state))
```

```
function MAX-VALUE(state,  $\alpha$ ,  $\beta$ ) returns a utility value
  inputs: state, current state in game
          $\alpha$ , the value of the best alternative for MAX along the path to state
          $\beta$ , the value of the best alternative for MIN along the path to state

  if TERMINAL-TEST(state) then return UTILITY(state)
   $v \leftarrow -\infty$ 
  for a, s in SUCCESSORS(state) do
     $v \leftarrow \text{MAX}(v, \text{MIN-VALUE}(s, \alpha, \beta))$ 
    if  $v \geq \beta$  then return v
     $\alpha \leftarrow \text{MAX}(\alpha, v)$ 
  return v
```

```
function MIN-VALUE(state,  $\alpha$ ,  $\beta$ ) returns a utility value
  same as MAX-VALUE but with roles of  $\alpha$ ,  $\beta$  reversed
```

Can We Prune More?

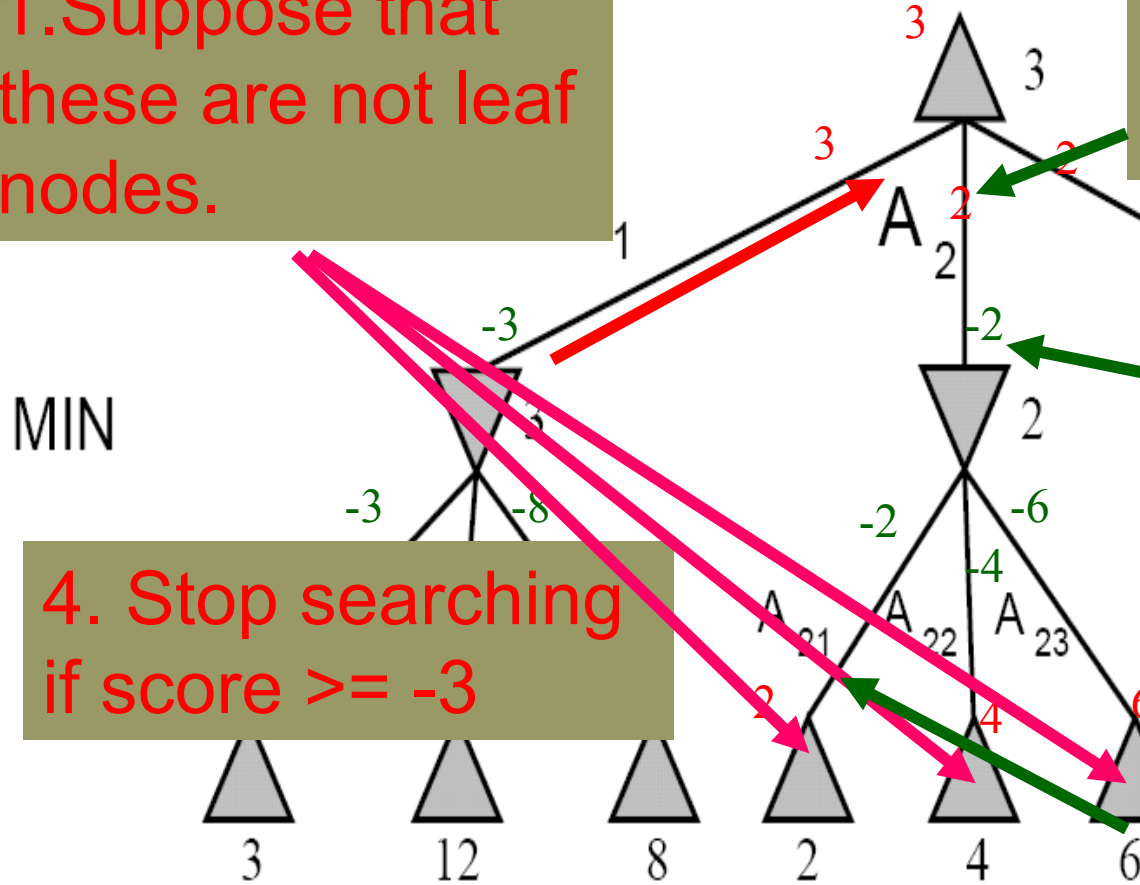
1. Suppose that these are not leaf nodes.

2. A_2 is considered bad if its score ≤ 3 .

3. That is, A_2 is considered bad if opponent's score ≥ -3 .

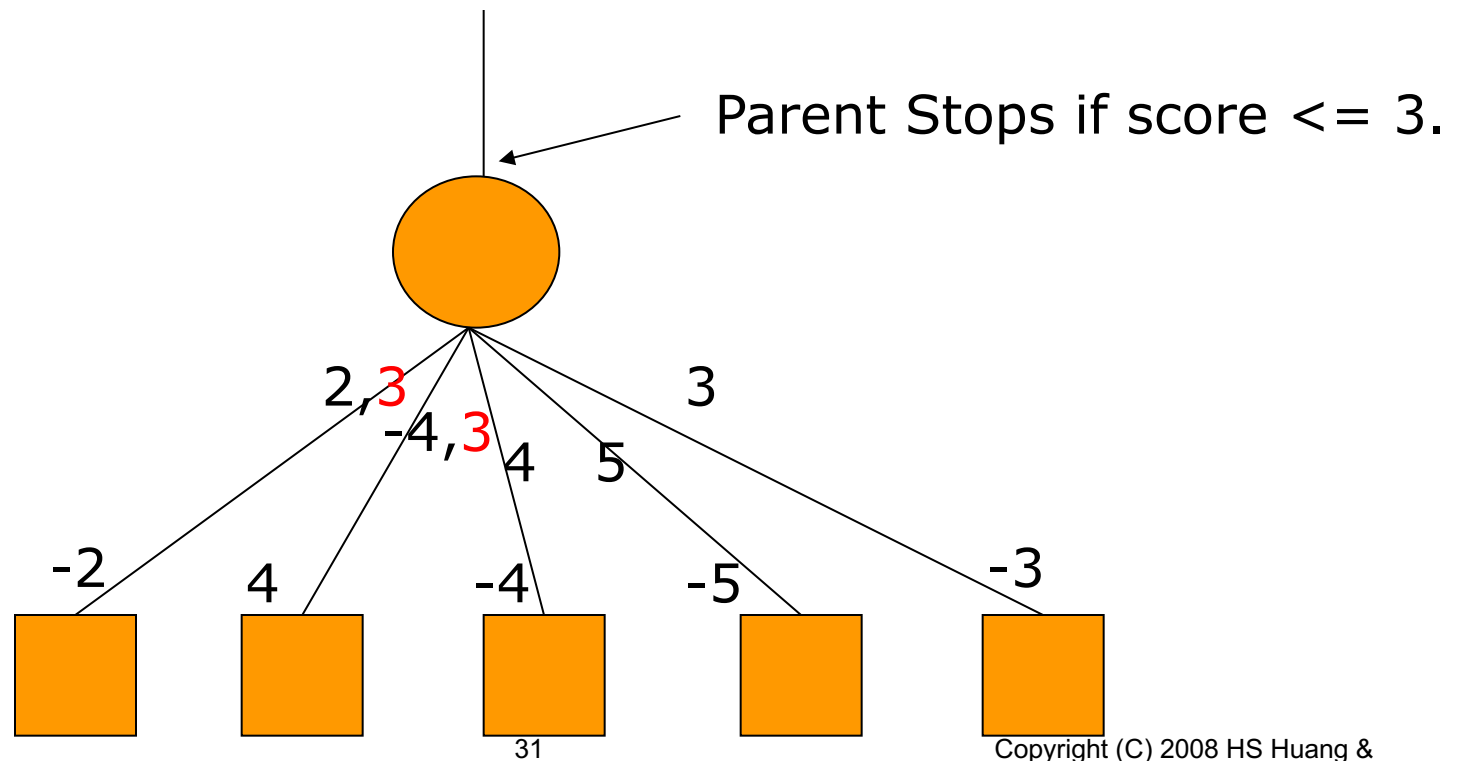
4. Stop searching if score ≥ -3

5. Parent stops searching if score ≤ 3 .



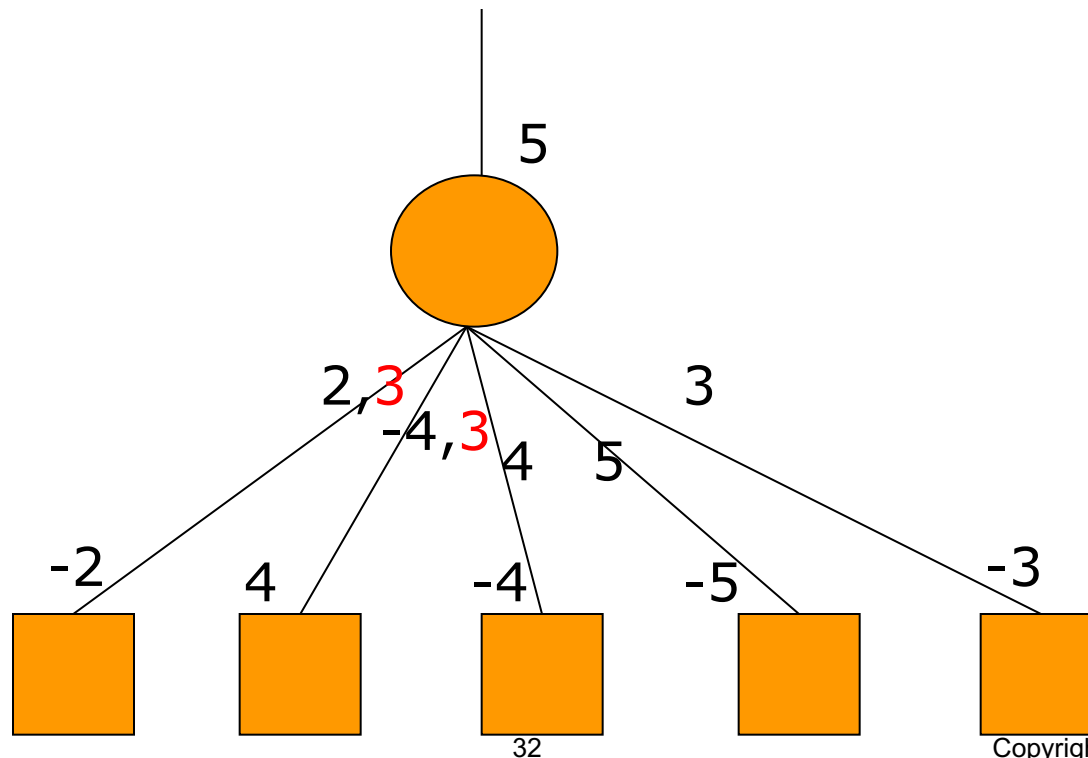
Can We Prune More?

Parent stops searching if my score ≤ 3 . What if we assign 3 to the moves whose scores ≤ 3 ?



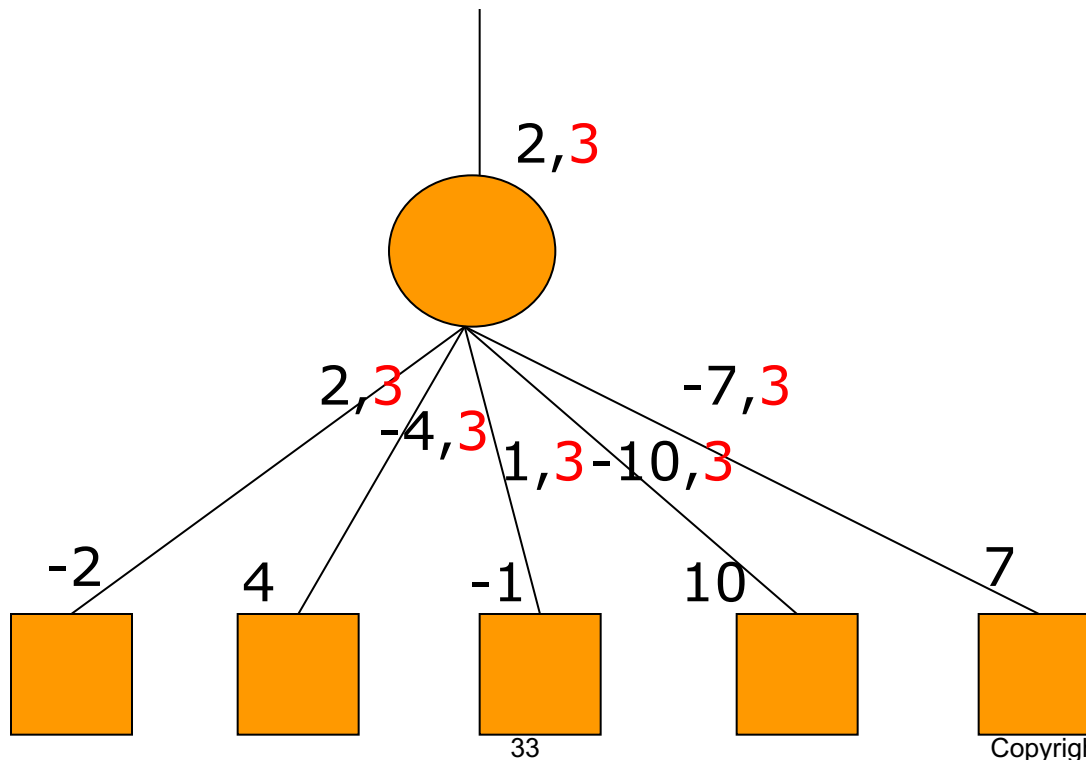
Can We Prune More?

If there is at least one move whose score > 3 , the returned score is the same:



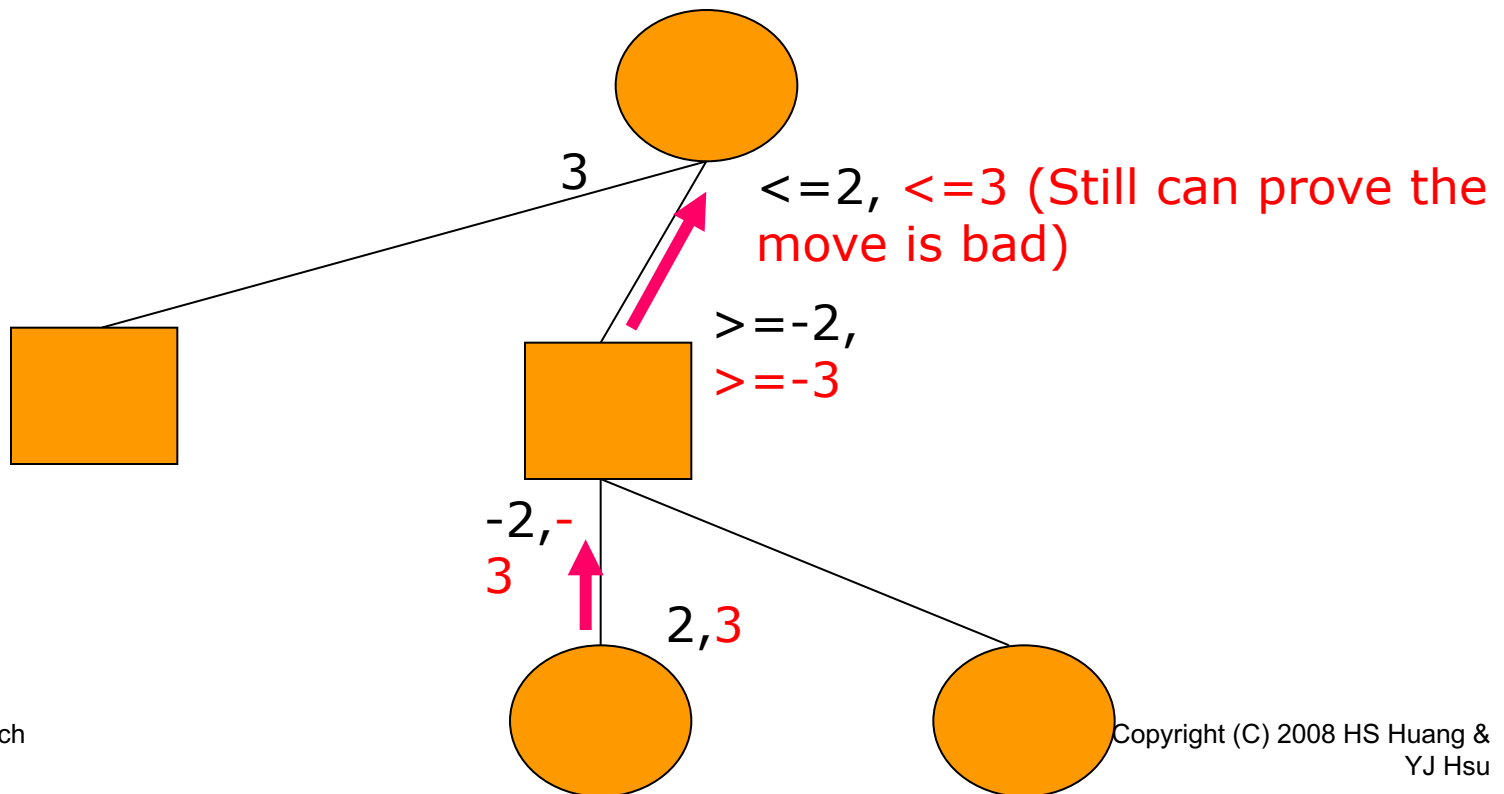
Can We Prune More?

If no move whose score > 3 , the returned score is 3 instead of 2. The parent gets -3, less than the true score (-2).



Can We Prune More?

- However, -3 is enough for the parent to prove that the grandparent's move is bad.



Can We Prune More?

Yes, we can assign 3 to the moves whose scores ≤ 3 .

1. In other words, 3 is the lowest score from the child nodes.
2. “3” is $-\beta$ of the parent node.

Pseudo Code of Alpha-Beta Pruning

```
int AlphaBetaMax(int depth, int player, int alpha, int beta) {  
    int best = alpha; // int best = -INFINITY;  
    if (depth <= 0)  
        return EvalFunc() * player;  
    for each move  $m$  {  
        MakeMove( $m$ );  
        score = -AlphaBetaMax(depth - 1, -player, -beta, -best);  
        BackMove( $m$ );  
        if (score >= beta) return beta;  
        if (score > best) {  
            best = score;  
        }  
    }  
    return best;  
}
```

Pseudo Code of Alpha-Beta Pruning

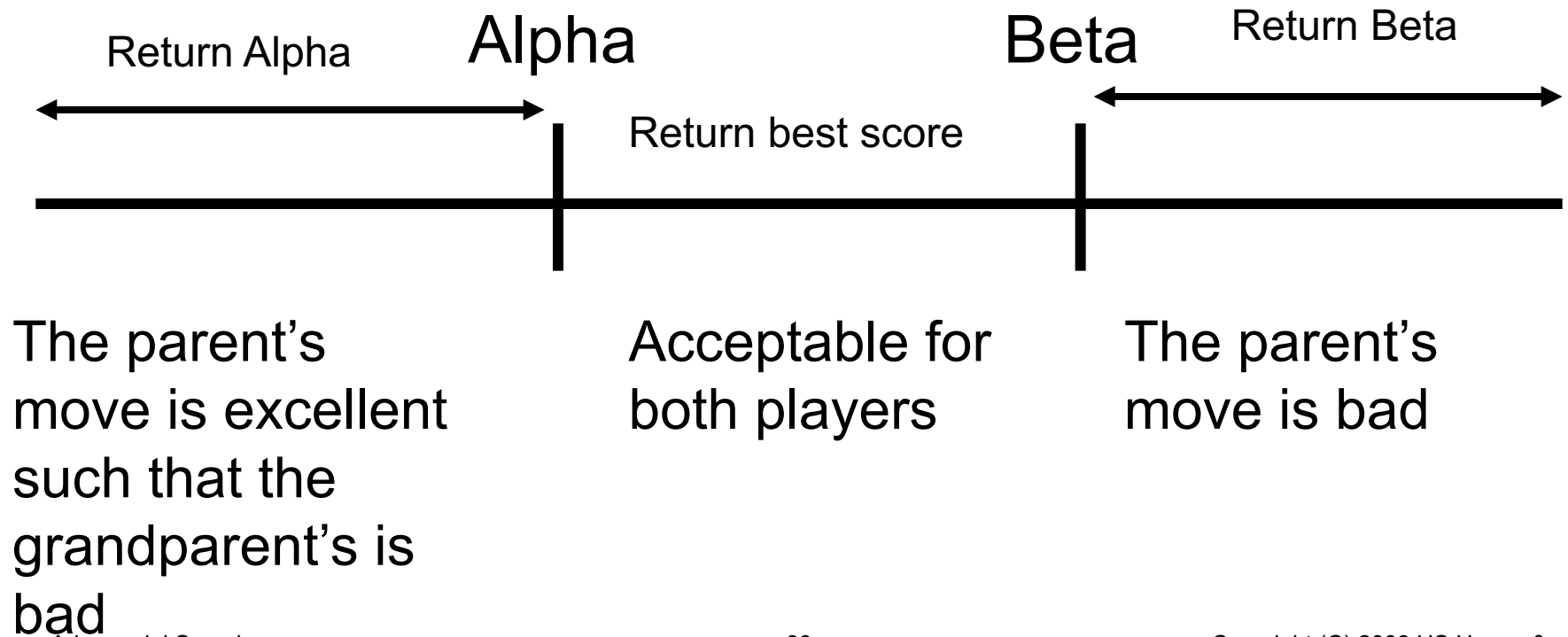
```
move AB_Decision(int depth, int player) {  
    int best = -INFINITY;  
    for each move  $m$  {  
        MakeMove( $m$ );  
        score = -AlphaBetaMax(depth - 1, -player,  
                                -INFINITY, -best);  
  
        BackMove( $m$ );  
        if (score > best) {  
            best = score;  
            best_move =  $m$ ;  
        }  
    }  
    return best_move;  
}
```

Summary

- AlphaBetaMax(depth, player, alpha, beta) asks for the score of a board layout:
 - “player” makes the next move.
 - If the utility of a leaf node is less than alpha, set it to alpha.
 - If the utility of a leaf node is greater than alpha, set it to beta.
 - Perform MaxMax algorithm with beta pruning

Summary

- The returned score:



Outline

Alpha-Beta Pruning

Done!

MiniMax Search

Done!

Faster and Faster

Game
Game Tree
Optimal Strategy

Done!

Improving Decision
Quality

Make Alpha-Beta Faster

- Search good move first
 - Heuristic move ordering
 - Iterative deepening
 - Principle variation search
 - Zero-window search
- Reuse scores for duplicate nodes
 - Transposition table
- Risky estimation of best score
 - Null-move search

Heuristic Move Ordering

```
int AlphaBetaMax(int depth, int player, int alpha, int beta) {  
    int best = alpha; // int best = -INFINITY;  
    if (depth <= 0)  
        return EvalFunc() * player;  
    for each move m {  
        MakeMove(m);  
        score = -AlphaBetaMax(depth - 1, -player, -beta, -best);  
        BackMove(m);  
        if (score >= beta) return beta;  
        if (score > best) {  
            best = score;  
        }  
    }  
    return best;  
}
```

Heuristic Move Ordering

- Checkmate
- Recapture
- Killer
 - The move that results in beta pruning earlier
- Capture opponent's rook, cannon, horse...
- Move my rook, cannon, horse...

Iterative Deepening

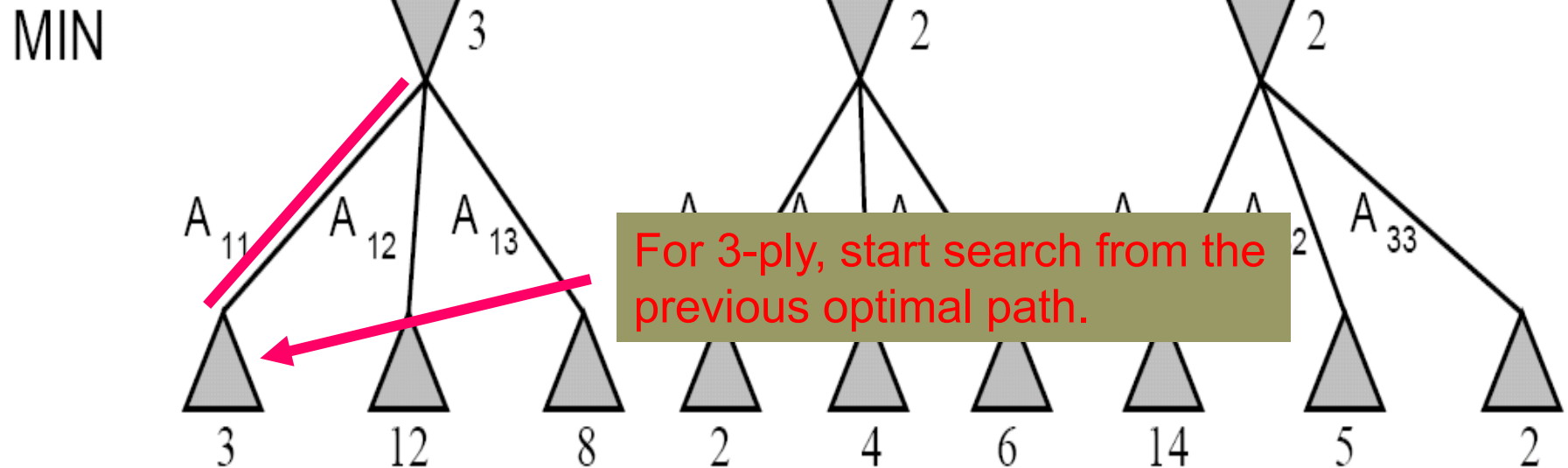
- Intuition: good moves in the search of N-ply game tree are usually good in the search of N+1-ply one:
 - Search 2-ply game tree (depth=2)
 - Sort the candidate moves
 - Search 3-ply game tree
 - Sort the candidate moves
 - ...

Iterative Deepening

```
move AB_Decision(int depth, int player) {  
    int best = -INFINITY;  
    for d=2 to depth {  
        for each move  $m$  {  
            MakeMove( $m$ );  
            score = -AlphaBetaMax( $d - 1$ , -player, -INFINITY, -best);  
            BackMove( $m$ );  
            if (score > best) {  
                best = score;  
                best_move =  $m$ ;  
            }  
            save_score( $m$ );  
        }  
        sort_moves();  
    }  
    return best_move;  
}
```

Principle Variation Search

MAX
Optimal path for both for
2-ply: A_1, A_{11}

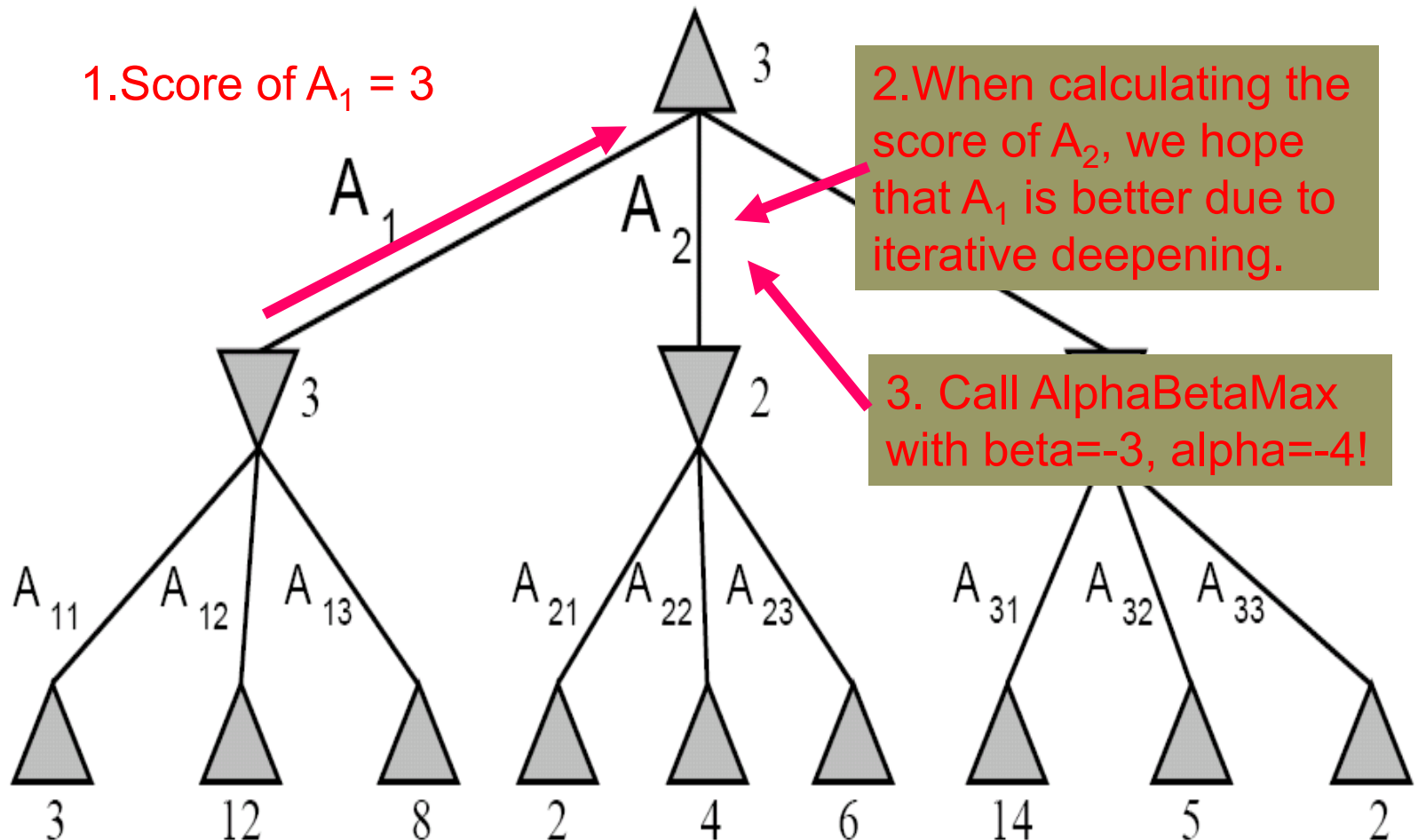


Zero-Window Search

MAX

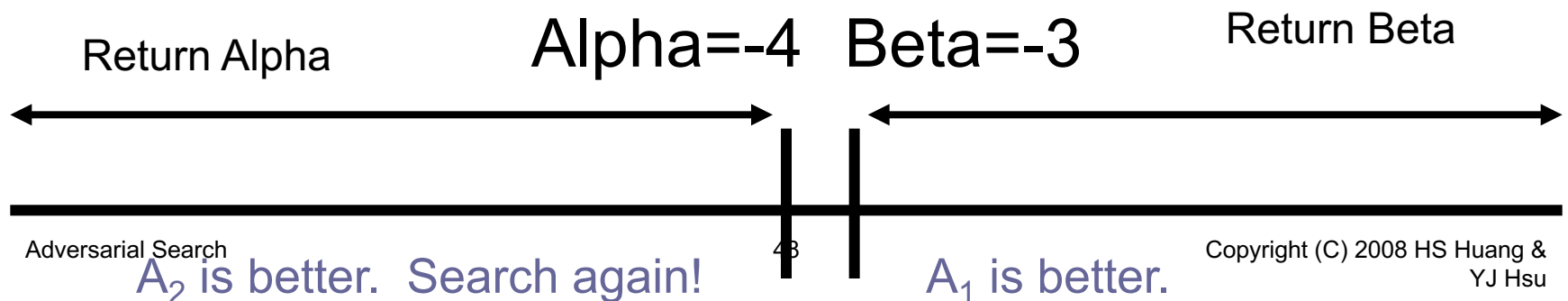
1. Score of $A_1 = 3$

MIN



Zero-Window Search

- We expect the returned value is -3 (beta pruning)
- If the returned value is -4 ($A_2 = 4$), the actual score falls between 4 and INFINITY.
- Run again with $\alpha = -\text{INFINITY}$ and $\beta = -4$ to get the correct score.



Transposition Table

- Different move sequences may reach the same board layout. Then, the same board layout may be evaluated more than once in a search.
- Transposition table is a hash table of evaluated board layouts.
- Look up the given layout in the transposition table before searching.

Null-Move Search

- In a search of N -ply game tree, we give up the first chance to move (null move).
- The depth becomes $N-1$, and the elapsed time should be fewer.
- Usually, our score is less than the score of N -ply game tree because we give up one move.
- The returned score serves as an estimate of the best score in alpha-beta pruning.

Outline

Alpha-Beta Pruning

Done!

MiniMax Search

Done!

Faster and Faster

Done!

Game
Game Tree
Optimal Strategy

Done!

Improving Decision
Quality

Quiescent Search

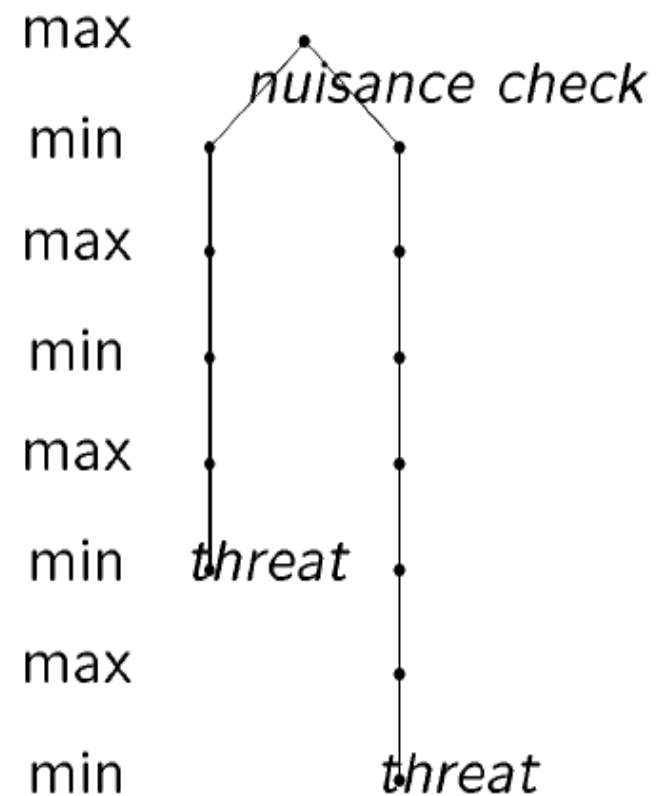
- Leaf nodes are evaluated by an evaluation function which is based on piece and position scores. Their scores may not be accurate if the board situation is not quiescent. For example, there is a favorable capture move in the board.
- Quiescent search, which usually considers capture moves, can be performed to see if the leaf node is quiescent.

Singular Extension

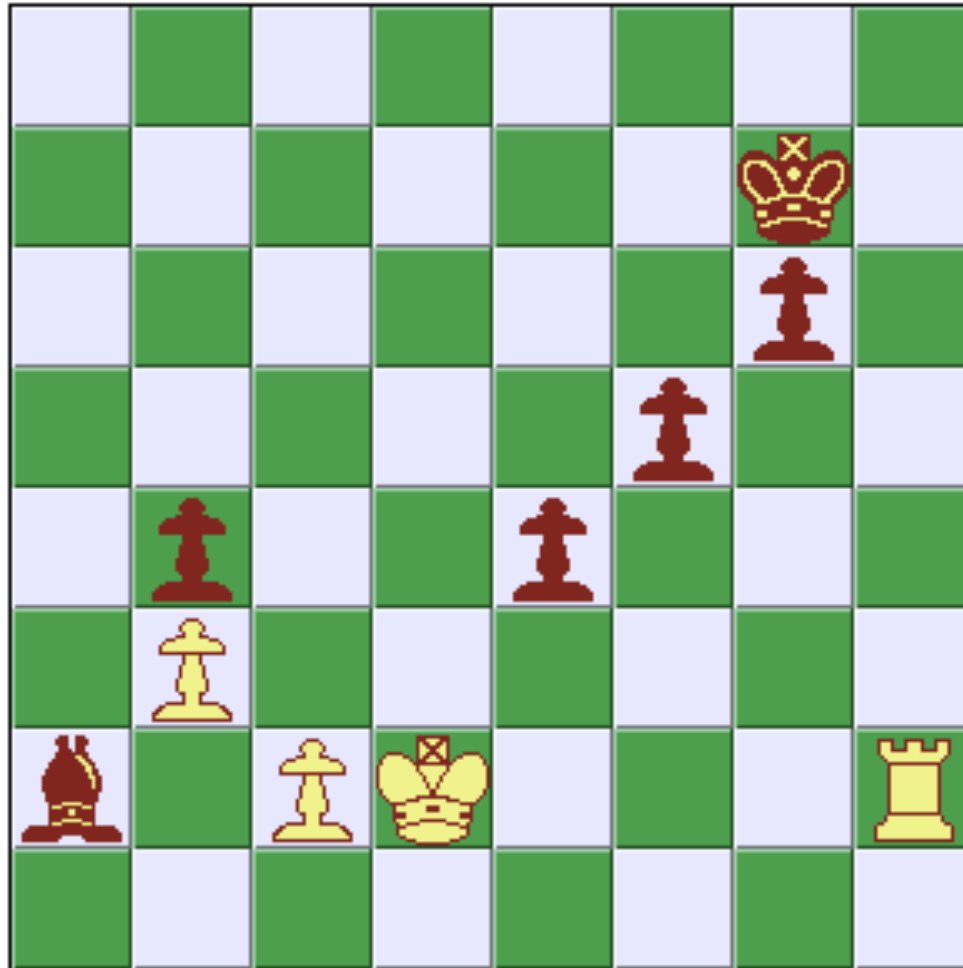
- In some situations we should extend the depth for some leaf nodes:
 - A series of recapture moves
 - Checkmates
 - Single response
- The branches of the above situations are usually small. Therefore, extension will not increase the computational cost too much.

Horizon Effect

- If the search depth is 5, a 5-ply search will fail to detect the threat.
- Nuisance move pushes threat past the search horizon, but actually leads to a worse position.
- Solution: faster search algorithm so that the search depth can be increased!



Example: The Horizon Effect



Outline

Alpha-Beta Pruning

Done!

MiniMax Search

Done!

Faster and Faster

Done!

Game
Game Tree
Optimal Strategy

Done!

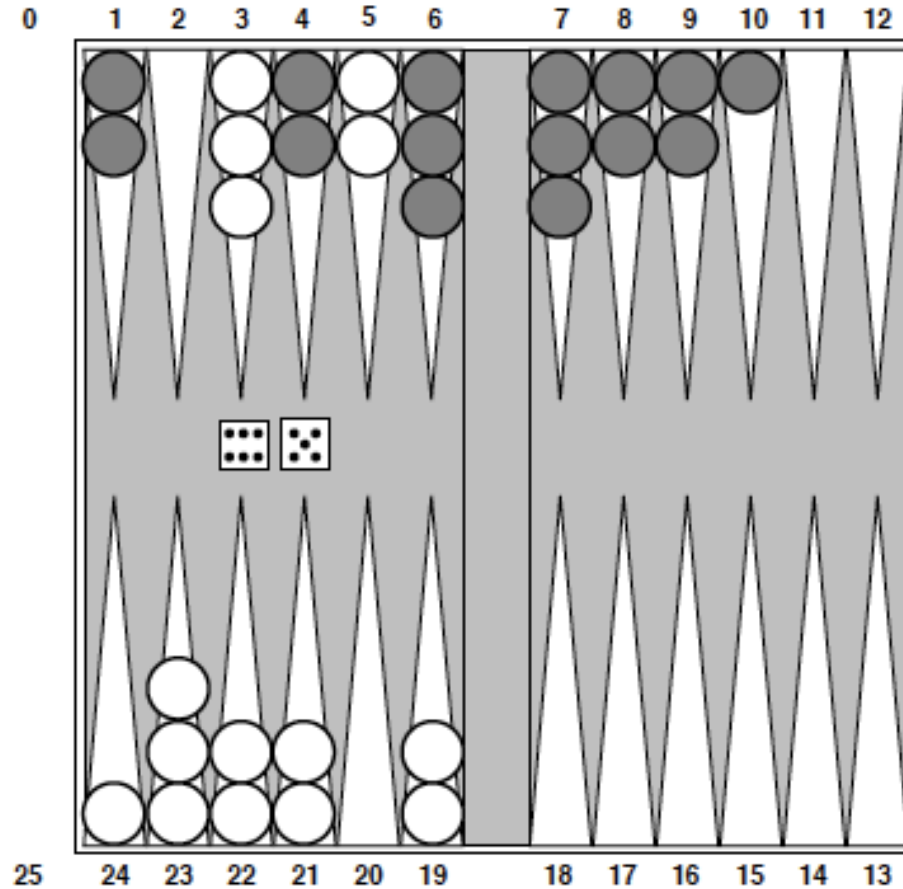
Improving Decision
Quality

Done!

Copyright (C) 2008

10/11/08

BackGammon

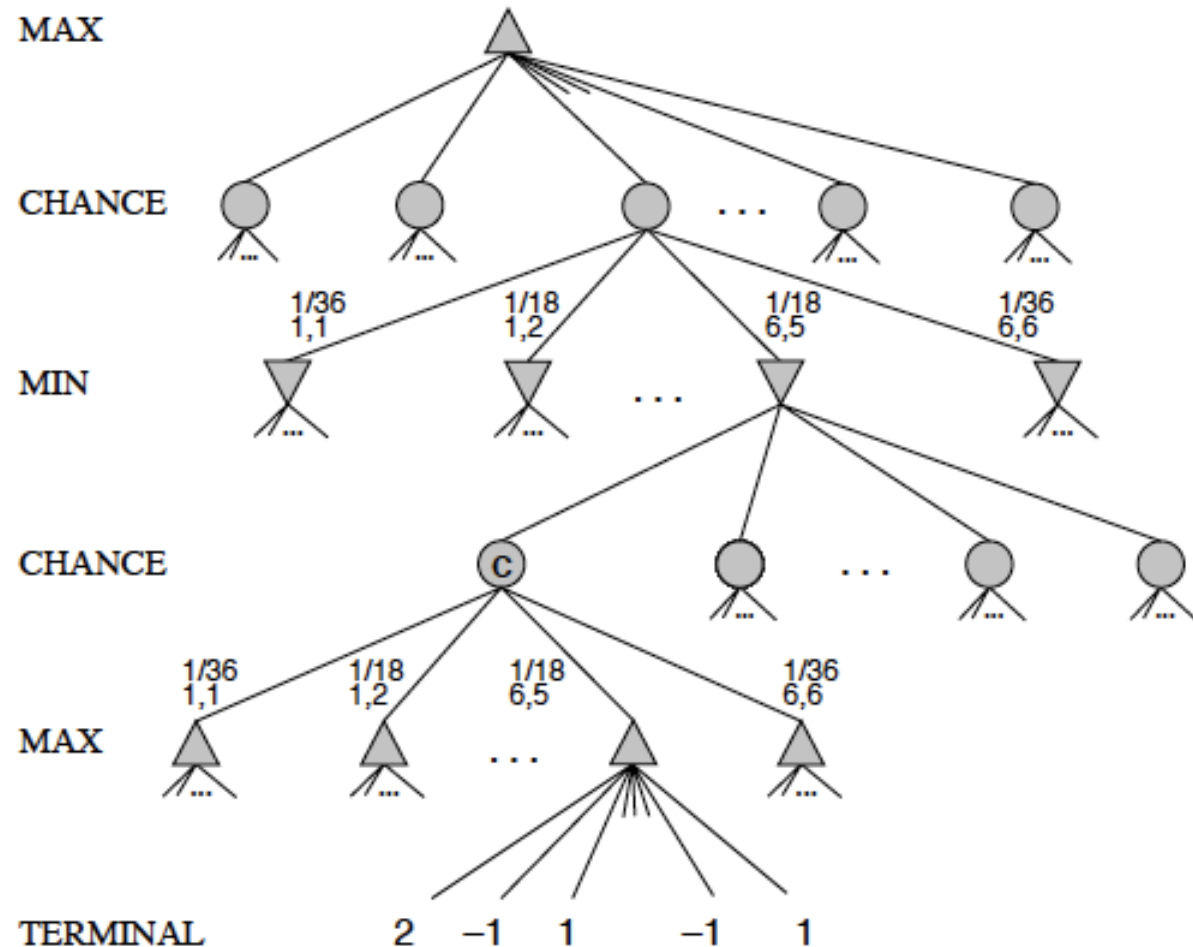


- Goal: to move all one's pieces off the board.

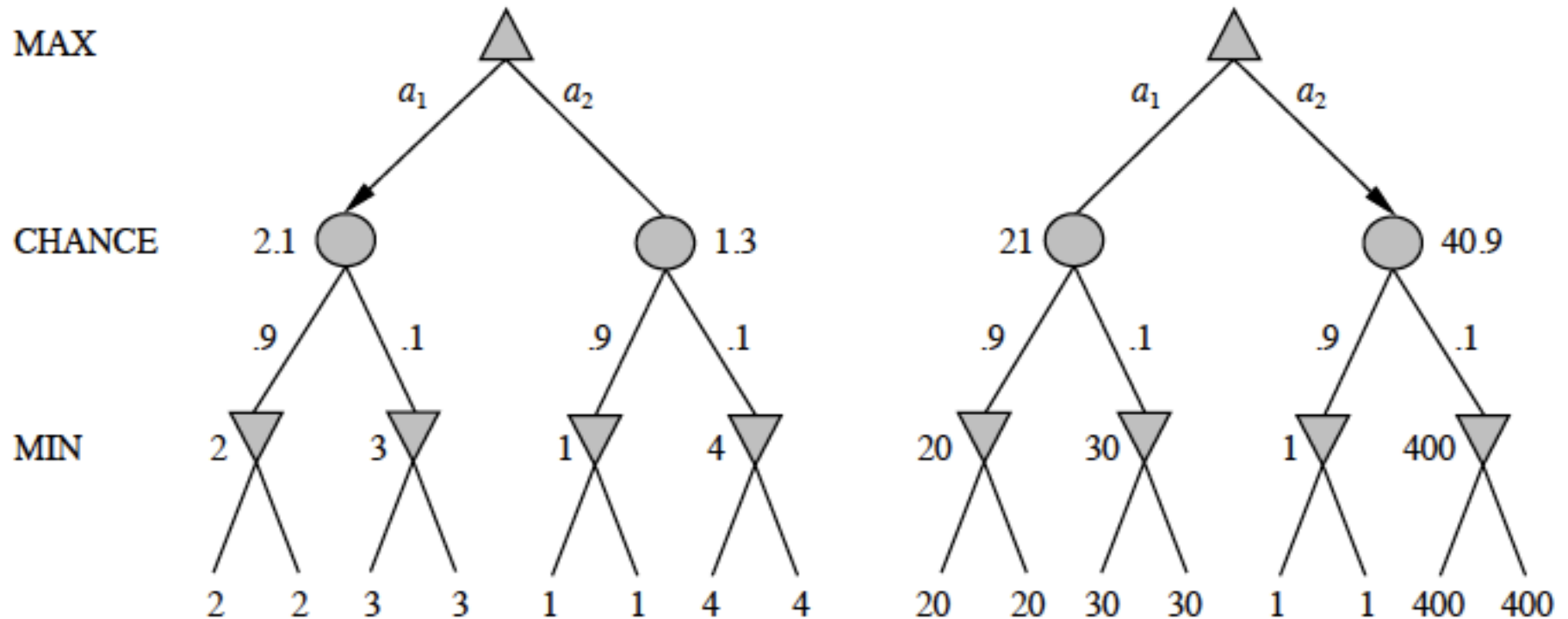
How do you represent it
as a game tree?



Game Tree



Order-preserving Transformation




Densei-sen: The Electronic Holy War



- May 1997
 - Deep Blue vs. Kasparov
- July 1997
 - NY Times proposed the challenge of GO (圍棋)
- March 2013
 - Crazy Stone (given a small handicap) defeated Yoshio Ishida, a professional Go player and a five-time Japanese champion.
 - programmed by Rémi Coulom, a CS professor at [Université Lille 3](#) in France
 - [Monte Carlo tree search](#)

"Brute-force searching is completely
and utterly worthless for Go,"
"You have to make a program play
smart like a person."



David Fotland

Author of *The Many Faces of Go*

Why is Go hard for computers?

- It takes ten moves for the number of possible game states to equal the number of stars in the universe.
 - cf. 15 moves for Chess
- The average Go game has a 150 moves, adding a new universe of possibilities beyond 10 moves.
- It is often hard to determine at any given time whether a group of pieces is being surrounded or doing the surrounding, and thus who is ahead.



Early Approaches to Playing Go

□ Proverbs/Heuristics

- “Never cut a bamboo joint,”
- “Don’t go fishing when your house is on fire,”
- “prioritize local responses versus global search,”
- “Never chase a dragon”

□ Human brain as inspiration

- neuroscientists in Japan, China, and Korea put Go professionals into brain scanners
- Human brain excels in pattern recognition, learning, intuition—are some of the hardest unsolved problems in A.I.

□ Early efforts amounted to programs that were effective on a small scale, for tactical fighting but little else.

Poker-Playing Computer

