Artificial Intelligence:
Search Methods for Problem Solving

Game Playing

A First Course in Artificial Intelligence: Chapter 8

Deepak Khemani

Department of Computer Science & Engineering

IIT Madras

Game Theory

noun: **game theory**;
the branch of mathematics
concerned with the analysis of strategies
for dealing with competitive situations
where the outcome of a participant's choice of action
depends critically on the actions of other participants.

Game theory has been applied to contexts in war, business, and biology.

What Is Game Theory?

Game theory is a theoretical framework for conceiving social situations among competing players.

In some respects, game theory is the science of strategy, or at least the optimal decision-making of independent and competing actors in a strategic setting.

The key pioneers of game theory were mathematicians John von Neumann and John Nash, as well as economist Oskar Morgenstern

The Basics of Game Theory

The focus of game theory is the game, which serves as a model of an interactive situation among rational players.

The key to game theory is that one player's payoff is contingent on the strategy implemented by the other player.

The game identifies the players' identities, preferences, and available strategies and how these strategies affect the outcome.

Depending on the model, various other requirements or assumptions may be necessary.

It is assumed players within the game are rational and will strive to maximize their payoffs.

The Nash Equilibrium

Nash Equilibrium is an outcome reached that,

once achieved,

means no player can increase payoff by changing decisions unilaterally.

It can also be thought of as "no regrets,"

in the sense that once a decision is made, the player will have no regrets concerning decisions considering the consequences.

Game Theory

Stanford Encyclopedia of Philosophy

Game theory is the study of the ways in which *interacting choices* of *economic agents* produce *outcomes* with respect to the *preferences* (or *utilities*) of those agents, where the outcomes in question might have been intended *by none* of the agents.

A group whose members pursue rational self-interest may all end up worse off than a group whose members act contrary to rational self-interest.

The Prisoner’s Dilemma

The prisoner's dilemma is a standard example of a game analyzed in game theory that shows why two completely rational individuals might not cooperate, even if it appears that it is in their best interests to do so.

It was originally framed by Merrill Flood and Melvin Dresher while working at RAND in 1950.

Albert W. Tucker formalized the game with prison sentence rewards and named it "prisoner's dilemma”.

https://en.wikipedia.org/wiki/Prisoner_dilemma

Prison Sentences

Two members of a criminal gang are arrested and imprisoned. Each prisoner is given the opportunity either to betray the other by testifying that the other committed the crime, or to cooperate with the other by remaining silent. The offer is:

- If A and B each betray the other, each of them serves two years in prison
- If A betrays B but B remains silent, A will be set free and B will serve three years in prison (and vice versa)
- If A and B both remain silent, both of them will serve only one year in prison (on a lesser charge).

https://en.wikipedia.org/wiki/Prisoner_dilemma

PD: The Payoff Matrix

B

A

A, B payoff	Cooperate	Defect
Cooperate	-1, -1	-3, 0
Defect	0, -3	-2, -2

A, B payoff	Cooperate	Defect
Cooperate	R, R	S, T
Defect	T, S	P, P

R: Reward if both cooperate
P: Punishment if both do not
T: Temptation to betray
S: Sucker punch for betrayed

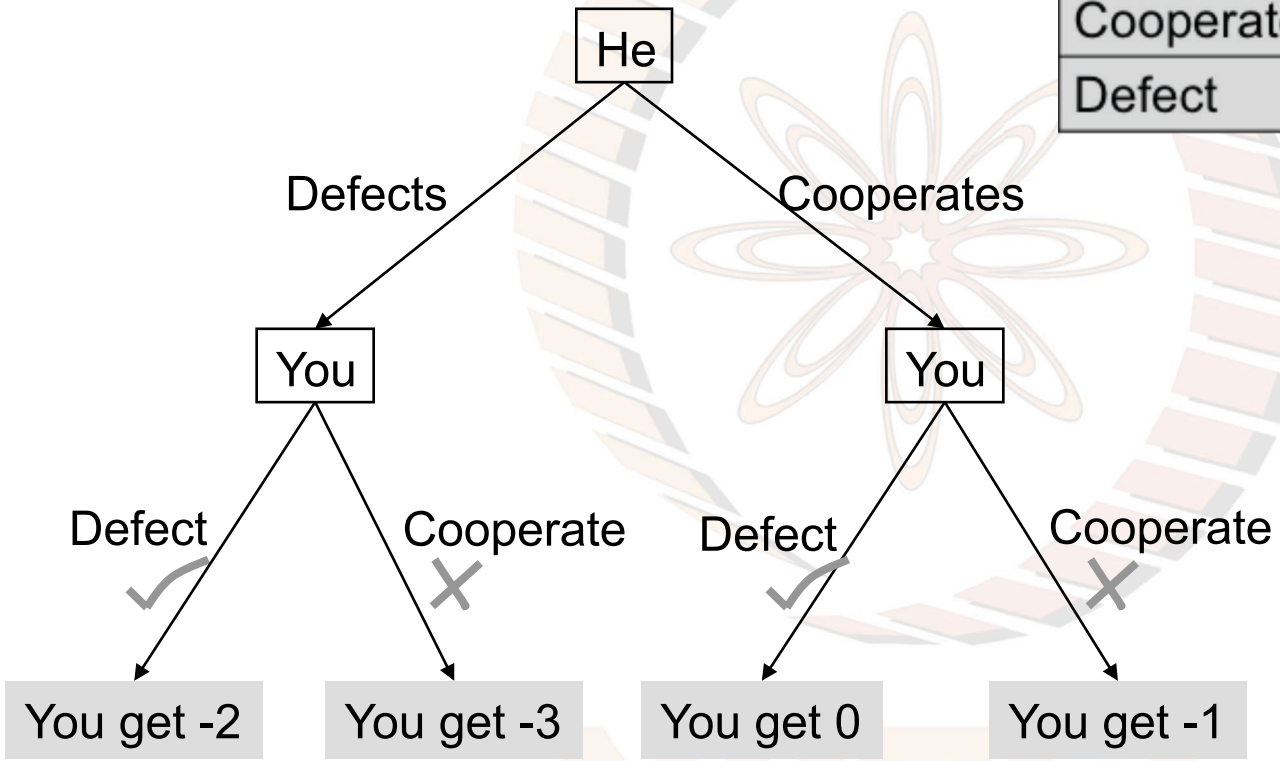
As long as
$T > R > P > S$
both players defect!

Even if the game is repeated many times

https://plato.stanford.edu/entries/prisoner-dilemma/

The rationale…

A, B payoff	Cooperate	Defect
Cooperate	-1, -1	-3, 0
Defect	0, -3	-2, -2



Types of Games

Games: payoff

Games can be classified as

- zero sum games
  - here the total payoff is zero
  - some players may gain while others lose
  - competing for the payoff
- positive sum games
  - the total payoff is positive
  - most or all players gain
  - cooperation
- negative sum games
  - total sum is negative
  - most or all players lose
  - for example price wars

Games: two or more players

- Many games are treated as two player games
  - for example price war between two companies – negative sum

- Multi player games have more than two players
  - price wars become a zero sum game when the consumer is included

- Multi player games are often two-teams-two-person
  - competition between two teams – often zero sum
  - each team may have team members collaborating
  - for example, contract bridge, football,
    armies on a battlefield, teams of lawyers in a courtroom,
    members of species in an ecosystem

Games: uncertainty

- Incomplete information games lead to uncertainty
  - in card games one cannot see other player's cards
  - in the corporate world we are not aware of what others are planning
    - and hence corporate espionage, lobbying with governments
  - in war, what the enemy is up to is not known
    - espionage – spies reporting from the other side
    - misinformation – double agents
    - deception – for example Operation Fortitude in WW2 – Normandy

- Stochasticity in domain leads to uncertainty
  - throw of the dice in Backgammon
  - draw of cards in Poker
  - shooting a basket in basketball

Popular Recreational Games

Chess: History

https://simple.wikipedia.org/wiki/History_of_chess

- The precursors of chess originated in northern India during the Gupta empire, where its early form in the 6th century was known as *Chaturanga*.
- This translates as '*the four divisions*', meaning infantry, cavalry, elephantry, and chariotry, which evolved into the modern pawn, knight, bishop, and rook, respectively.
- In Sassanid Persia around 600 the name became *Chatrang* (in Hindi it is still called *Shatranj)*
- Players started calling *Shāh!* (Persian for 'King') when threatening the opponent's king, and *Shāh māt!* (Persian for 'the king is finished')
  – modern day *check* and *checkmate*.

Chess: History

https://simple.wikipedia.org/wiki/History_of_chess

- The game reached Western Europe and Russia by at least three routes, the earliest being in the 9th century.
- Introduced into the Iberian Peninsula by the Moors in the 10th century, it was described in a famous 13th century manuscript covering *shatranj*, *backgammon* and *dice* named the *Libro de los juegos*.
- Buddhist pilgrims, Silk Road traders and others carried it to the Far East, where it was transformed into a game often played on the *intersection of the lines of the board rather than within the squares*.
- *Chinese chess* and *Shogi* are the most important of the oriental chess variants.

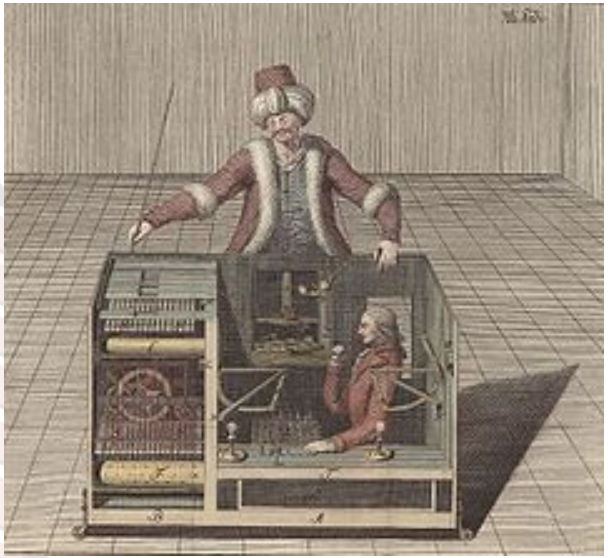
Chess terminology across regions

Sanskrit	Persian	Arabic	English
Raja	Shah	Shah	King
Mantri	Vazir	Wazir/Firzān	Queen
Gajah*	Pil	Al-Fil	Bishop
Ashva	Asp	Fars/Hisan	Knight
Ratha*	Rukh	Rukh	Rook
Padati	Piadeh	Baidaq	Pawn

* Hindi users often use *Oont* (camel) for Bishop
and *Haathi* (elephant) for Rook

https://simple.wikipedia.org/wiki/History_of_chess

Von Kempelen's Chess Playing Turk



Wolfgang von Kempelen (1734 – 1804) is most well known for a chess playing machine also known as the *Mechanical Turk* in 1770 to impress the Empress Maria Theresa of Austria.

The Turk was in fact a mechanical illusion that allowed a human chess master hiding inside to operate the machine. who defeated many challengers including statesmen such as Napoleon Bonaparte and Benjamin Franklin.

Source: http://en.wikipedia.org/wiki/The_Turk

Samuel's Checkers program

- Pamela McCorduck in *Machines Who Think*

Arthur Samuel (1901-1990) was one of the attendees in the Dartmouth Conference where the term AI was coined.

He wrote the first Checkers playing program in 1952 on IBM's 701 computer.

Samuel's goal was to explore how to get computers to learn – he felt that if computers could learn from experience then there would be no need for detailed and painstaking programming.

His Checker's program improved as it played more and more games, eventually "beating its own creator" – evoking fears of Frankenstein (Mary Shelley) like creatures overwhelming humankind.

The Chess saga: Genesis

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

1912: Leonardo Torres y Quevedo builds a machine that could play King&Rook vs. King.

1950: Claude Shannon publishes "*Programming a Computer for Playing Chess*".

1951: Alan Turing develops on paper the first program capable of playing a full game of chess.

1956: John McCarthy invents the alpha-beta search algorithm (also credited to others…).

1957: Alex Bernstein develops first program to play full chess at IBM.

1967: *Mac Hack Six*, by Richard Greenblatt et al. introduces transposition tables and becomes the first program to defeat a person in tournament play.

1968: David Levy bet:
No computer program will beat him within 10 years.

1970: The ACM North American Computer Chess Championships

1974: *Kaissa* wins the first World Computer Chess Championship

1977: The first microcomputer chess playing machine, *CHESS CHALLENGER*, was created.

The Chess saga: Progress

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

1977: *Chess 4.6* is the first chess computer to be successful at a major chess tournament.

1978: David Levy wins the bet defeating the Chess 4.7 in a six-game match. Score: 4.5–1.5.

1980: The Fredkin Prize is established ($100,000 to beat a reigning world champion).

1981: *Cray Blitz* wins the Mississippi State Championship with a perfect 5–0 score and a performance rating of 2258. The first computer to beat a master in tournament play.

1982: Ken Thompson's hardware chess player *Belle* earns a US master title.

1988: *HiTech*, by Hans Berliner and Carl Ebeling, wins a match against grandmaster Arnold Denker 3.5 – 0.5.

1988: *Deep Thought* shares first place with Tony Miles, ahead of former world champion Mikhail Tal

1989: *Deep Thought* loses two exhibition games to Garry Kasparov, the reigning champion.

Named after the famous novel by Douglas Adams
– The Hitchiker's Guide to the Galaxy

The Chess saga: Triumph

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

1992: A microcomputer, the *ChessMachine Gideon 3.1* by Ed Schröder, wins the 7th World Computer Chess Championship in front of mainframes, supercomputers and special hardware.

1994: *ChessGenius*, defeated a World Champion (Garry Kasparov) at a non blitz time limit.

1996: *Deep Blue* loses a six-game match against Garry Kasparov.

1997: *Deep Blue* wins a six-game match against Garry Kasparov. The *Deep Blue* inventors Fang Hsu, Murray Campbell, and Joseph Hone awarded the Fredkin Prize.

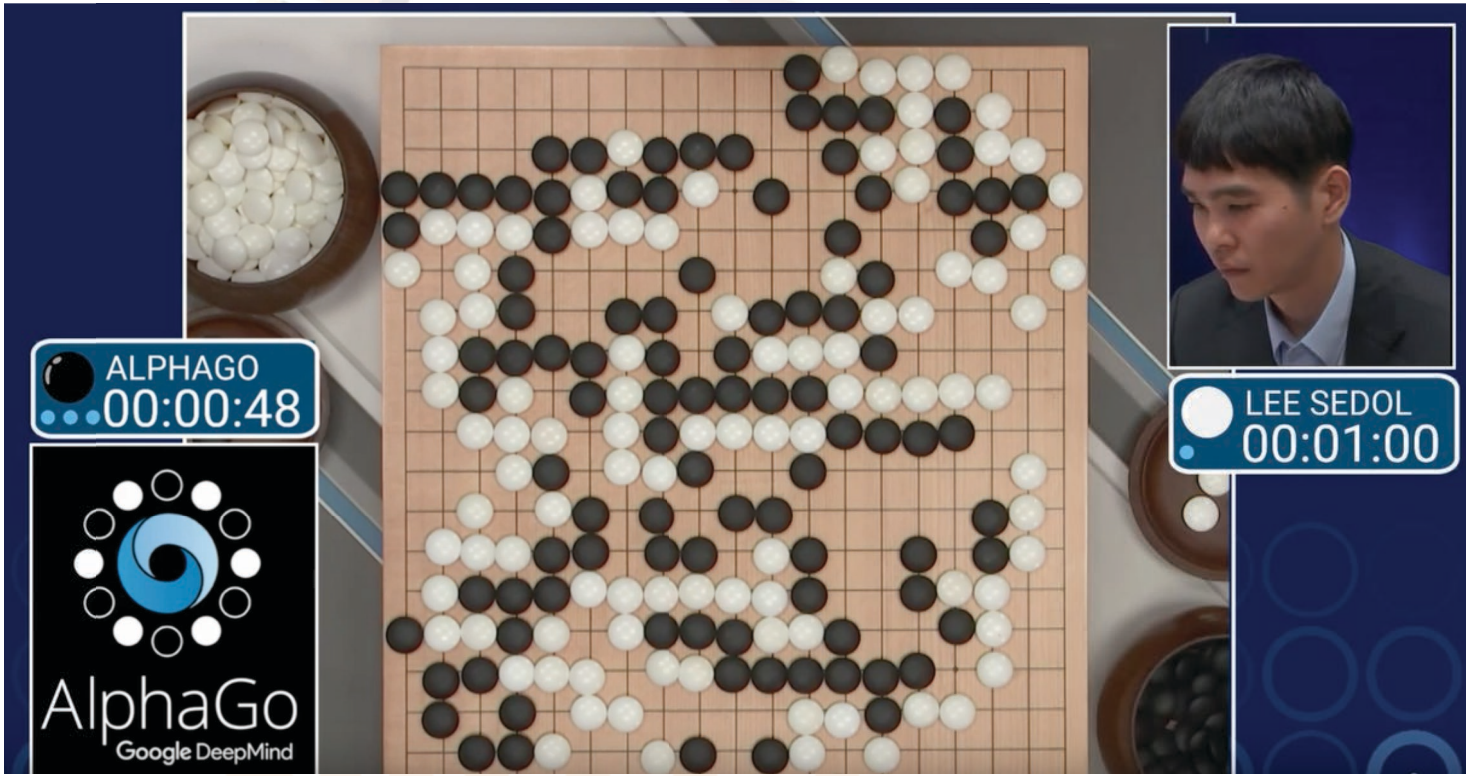
2002: Vladimir Kramnik draws an eight-game match against *Deep Fritz*.

2005: *Hydra* defeats Michael Adams 5.5–0.5.

2006: The undisputed world champion, Vladimir Kramnik, is defeated 4–2 by *Deep Fritz*.

2010: Before the World chess championship, Topalov prepares by sparring against the supercomputer *Blue Gene*

AlphaGo beats World Champion at Go in 2016



from this Popular Mechanics article

Other popular games: Scrabble

- Scrabble is a word game in which two to four players score points by placing tiles bearing a single letter onto a board divided into a 15×15 grid of squares.
- The tiles must form words that, in crossword fashion, read left to right in rows or downward in columns, and be included in a standard dictionary or lexicon.
- *Maven* was a computer program created by Brian Sheppard (2002). The official Scrabble computer game in North America uses a version of Maven.
- Computers can beat humans easily.

Other popular games: Backgammon

- Backgammon is amongst the oldest known games, reputedly being played a thousand years before Chess (Tesauro, 1995).
- It has stochastic moves based on the throw of dice.
- A similar game called *Chausar* has been described in ancient Indian literature.
- In the Indian epic *Mahabharata* an episode is ingrained the Indian psyche because it involved the gambling away of *everything* eventually leading to the epic battle.
- Also called *Pachisi*, it was popular in the Mughal courts of India.
- The champion program *TD-Gammon* by Gerald Tesauro, learnt the principles of *Backgammon* strategy, playing 300,000 games against itself

Other popular games: Poker

- Poker is a card game played in which each player is dealt five cards, and essentially bets on the whether her holding is “better” than that of other players.
- The strategy goes beyond calculating odds
- Deception is a key component
  - players need to bluff on poor hands to gain on good hands
  - a straight player would give his hand away
- An AI program, called *Pluribus*, developed in CMU in 2019 defeated leading professionals in six-player no-limit Texas hold'em poker, the world's most popular form of poker.
  - It does so by doing a limited lookahead, and
  - learning by playing against copies of itself

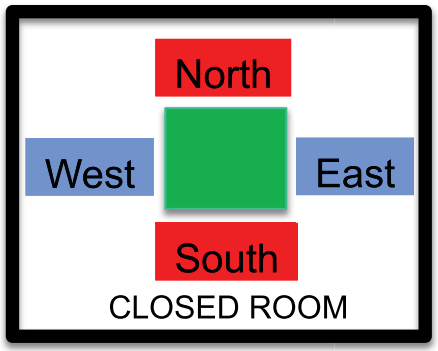
Other popular games: Contract Bridge

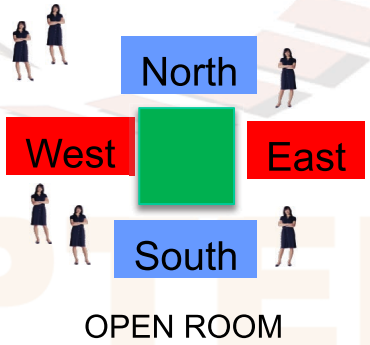
- Bridge is a card game in which on a table two partners play against two opponents who are also partners
- In a tournament format each team will have two members in another room holding their opponent hands
- There are two phases – bidding and play
- Both phases involve formal communication, making inferences, planning, deception and probabilistic reasoning
- Several groups have worked on bridge playing programs since the 1980s
- In 1996 the American Contract Bridge League established an official World Computer-Bridge Championship,
  to be run annually at an international bridge event.
- We still await a program better than humans!

Bridge vs. Chess

- The number of different deals or starting positions is 5 x 10^{27}
  - Practically every deal is a new deal
  - The game tree for *each deal* has about 10^{21} leaf nodes
- Chess always has the same starting position, and the same goal
  - Chess is played between two players, there is no communication, no deception, and no hidden information

A bridge match: Blue team vs. Red team





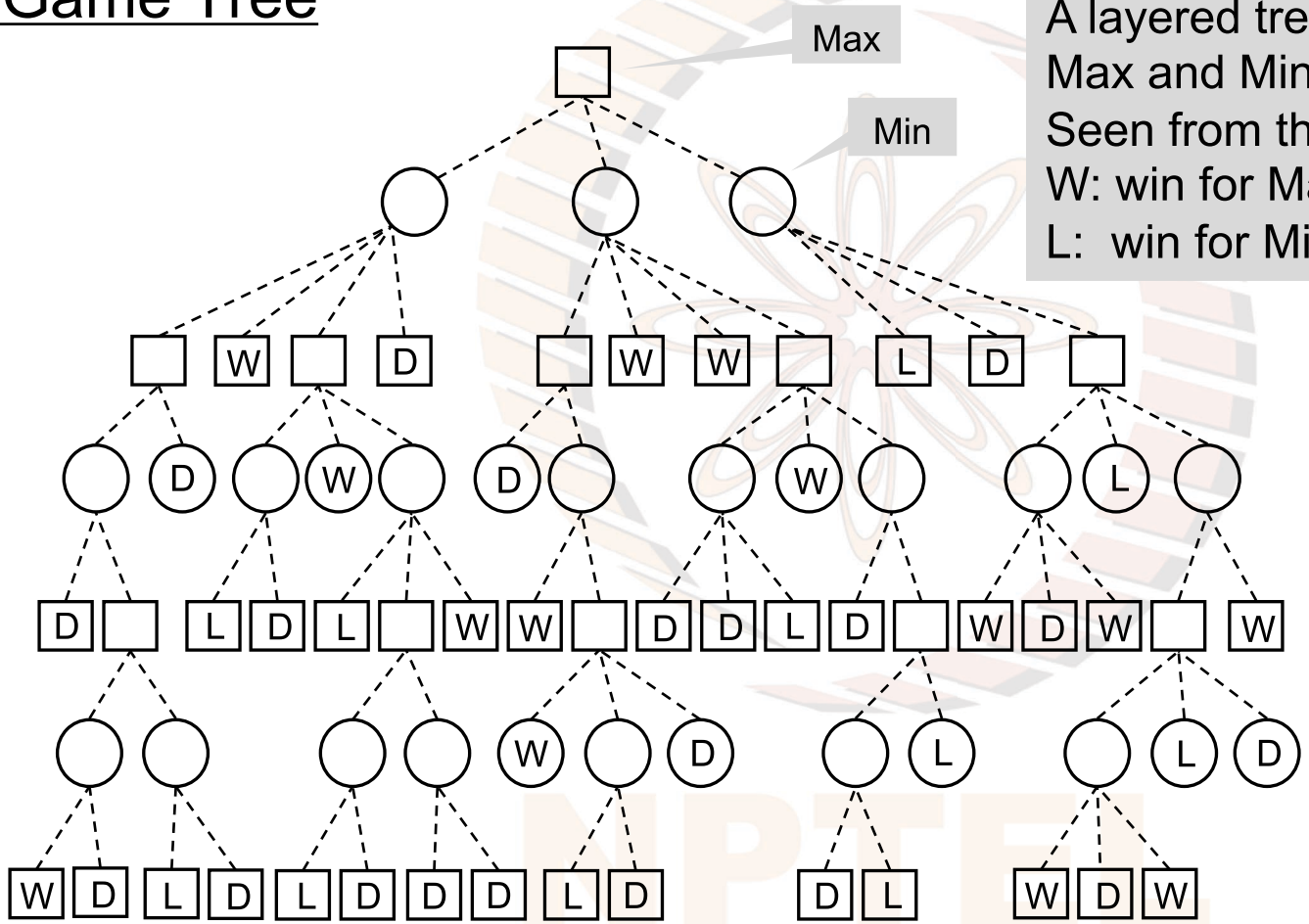
Board Games and Game Trees

NPTEL

Board Games

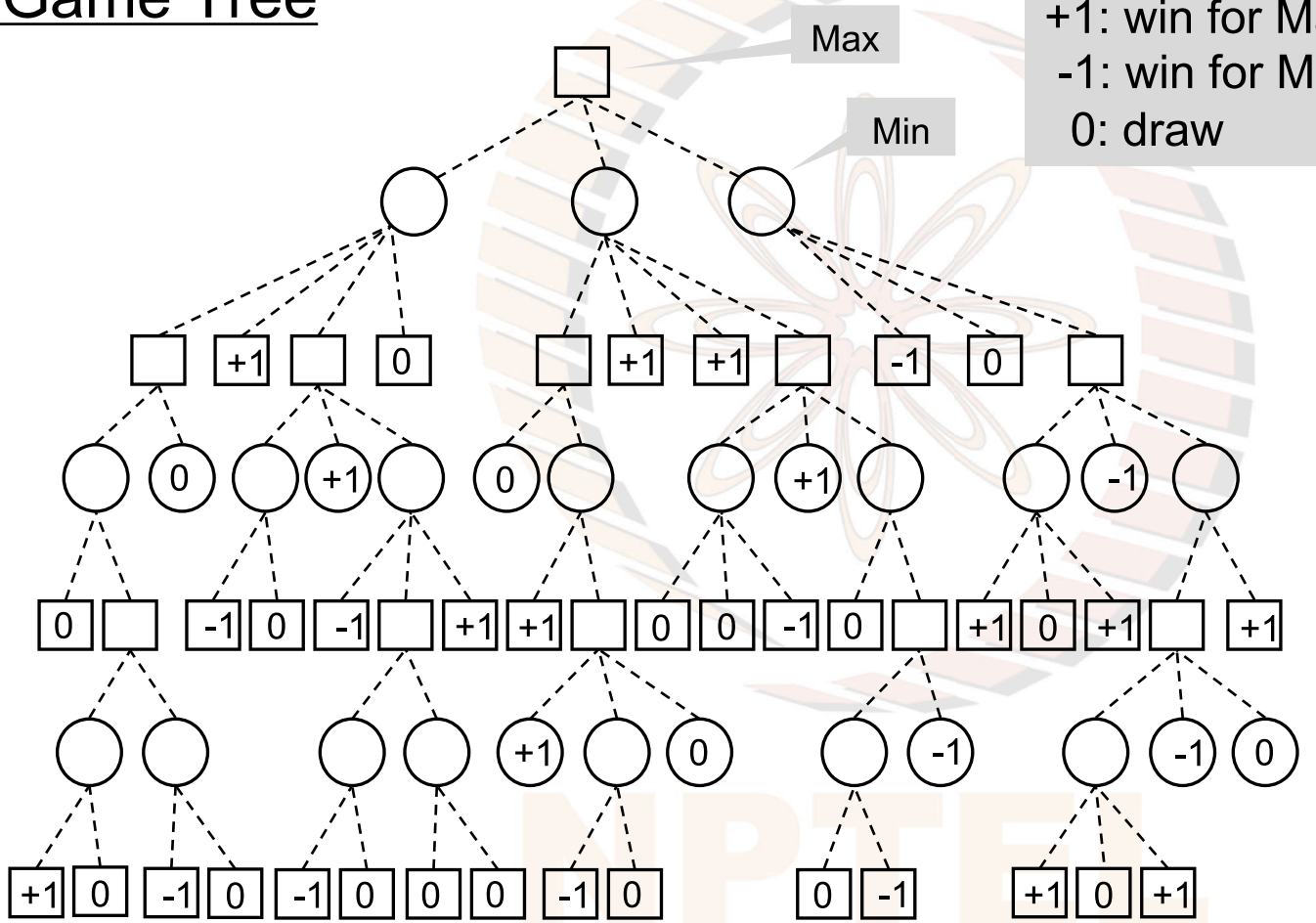
- Two person
- Zero sum
  - total payoff is zero
  - one player wins and the other loses
  - or the game ends in a draw
- Complete information
  - both players can see the board
  - and the moves available to the opponent
- Alternate moves
  - under some conditions some games allow more than one move, or a compound move to a player
- Deterministic
  - there is no element of chance

A Game Tree



A layered tree
Max and Min choose alternately
Seen from the perspective of Max
W: win for Max, loss for Min
L: win for Min, loss for Max

A Game Tree


Max
Min
+1: win for Max, loss for Min
-1: win for Min, loss for Max
0: draw

Minimax backup rule

The *leaves* are labeled with the *outcome* of the game.

When both players are *rational* the *value of the game* is fixed.

The *minimax value* is the *outcome* when *both play perfectly*.

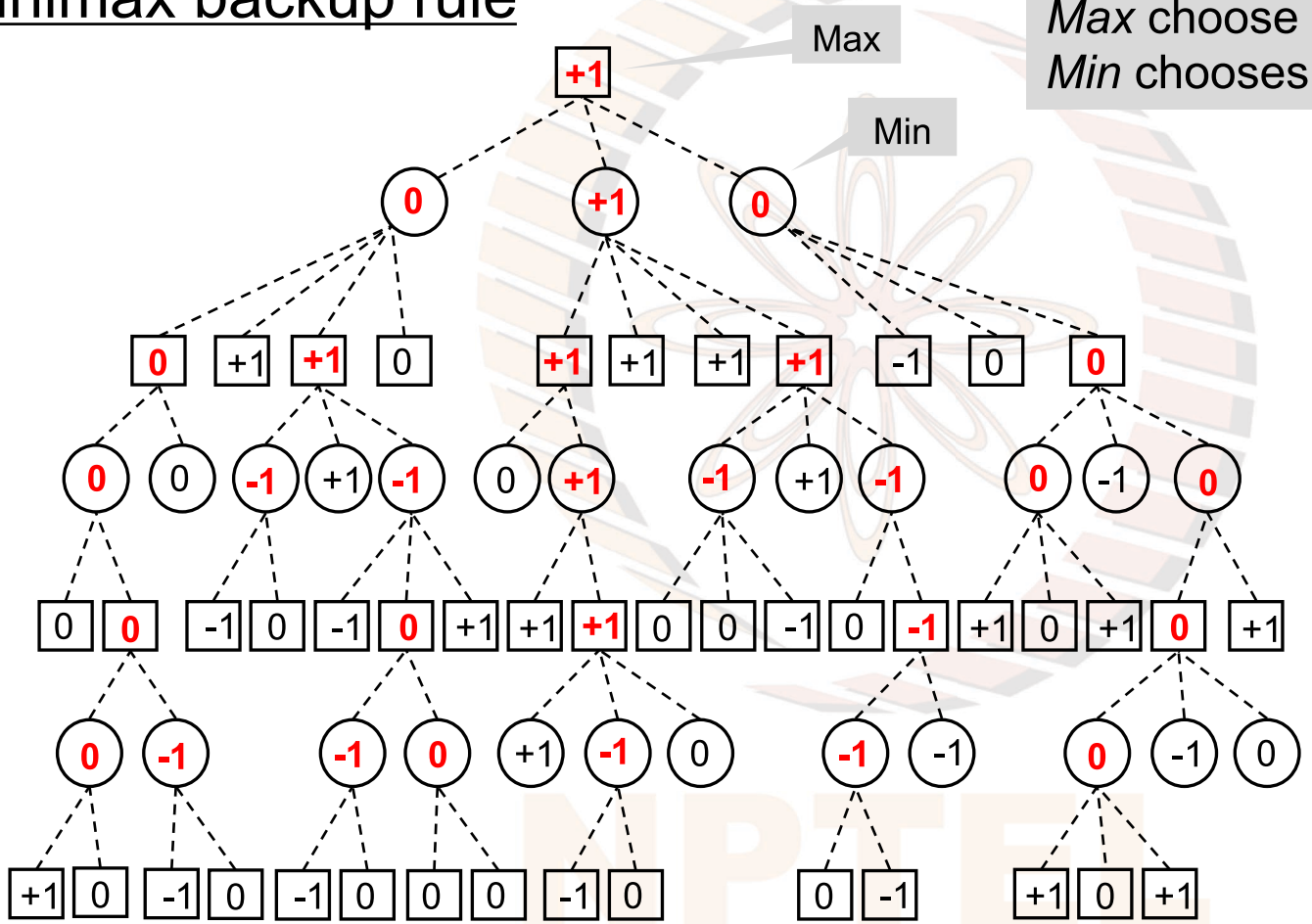
Minimax value can be computed by applying the *minimax rule* bottom up.

If J is a Max node backup values from its children as follows
Backup a W (or +1) if available
else backup D (or 0) if available
else backup L (or -1)

If J is a Min node backup values from its children as follows
Backup a L (or -1) if available
else backup D (or 0) if available
else backup W (or +1)

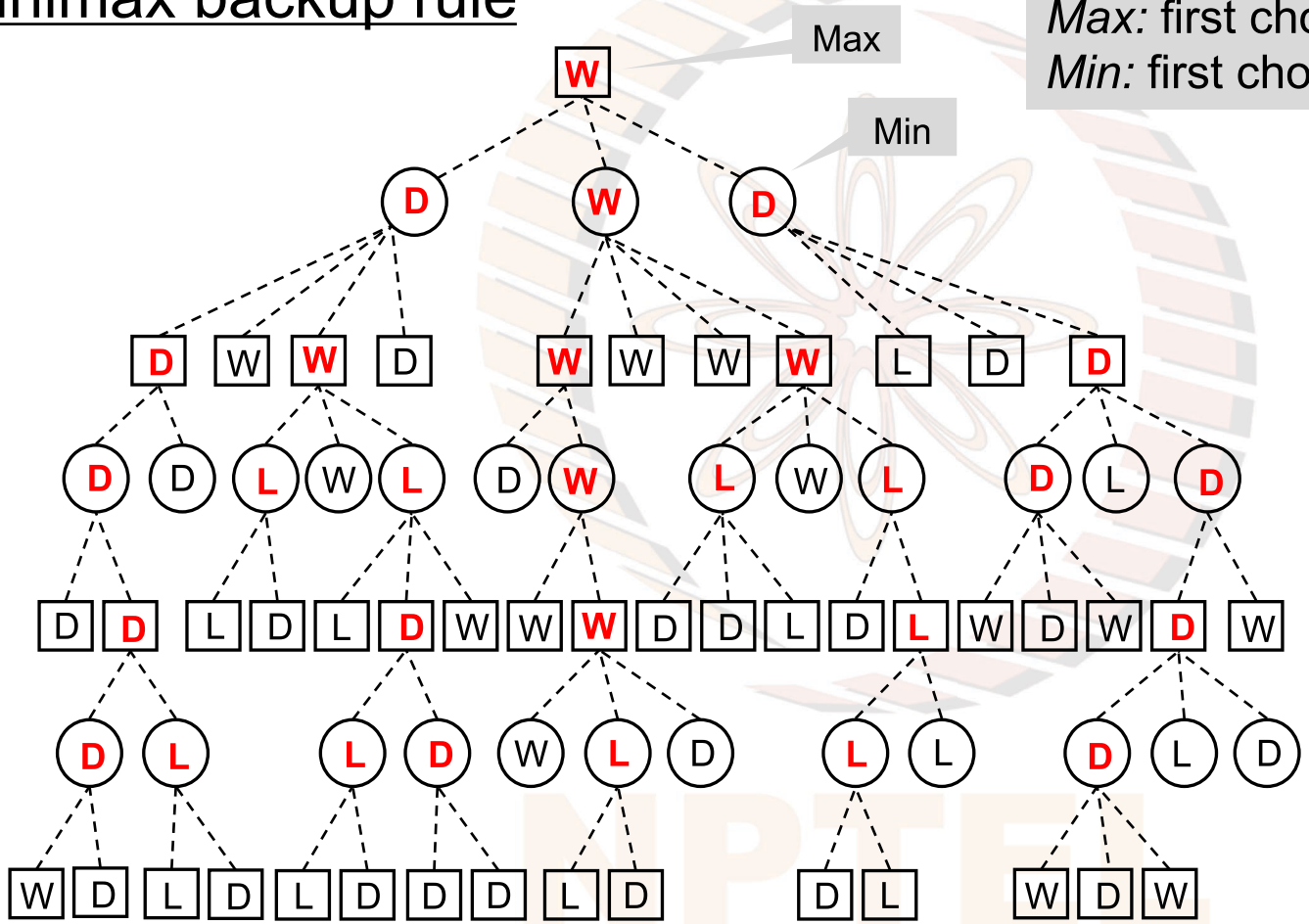
Minimax backup rule

Max choose *maximum* child value
Min chooses *minimum* child value



Minimax backup rule

Max: first choice W, next D, else L
Min: first choice L, next D, else W



Strategies

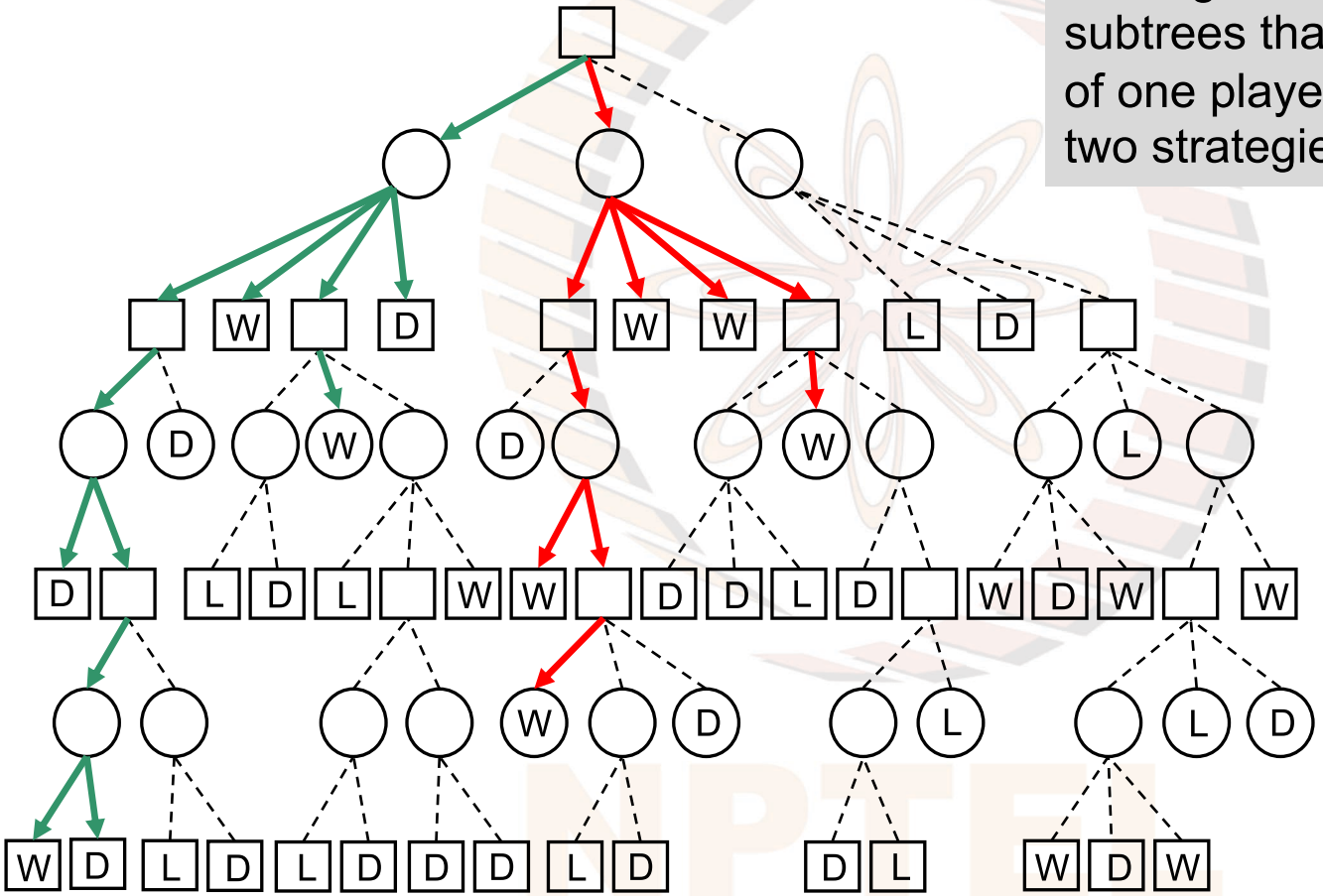
A strategy for a player is a subtree of the game tree that completely specifies the choices for that player. The algorithm below constructs a strategy for Max

CONSTRUCT-STRATEGY(MAX)

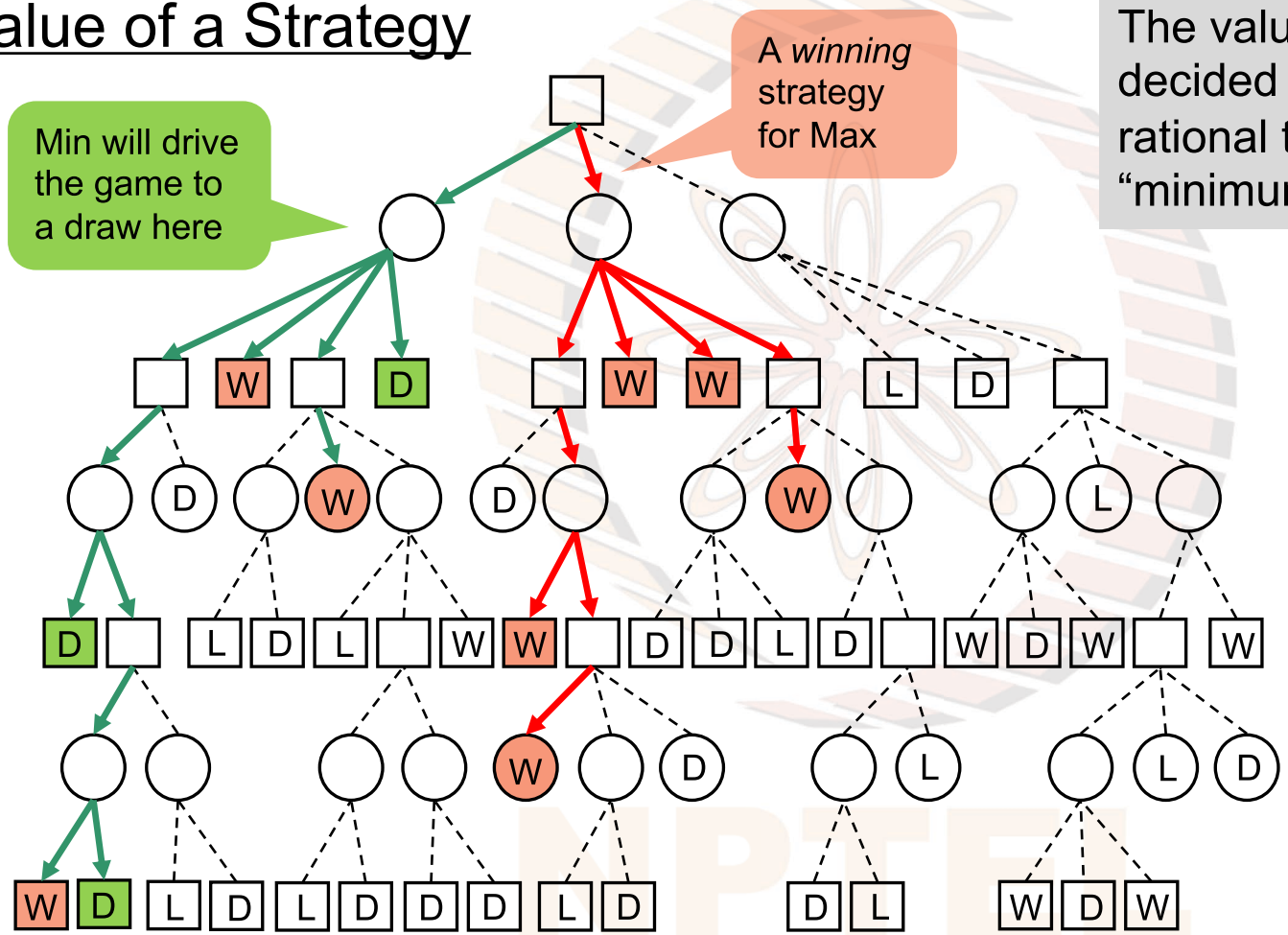
```
1  traverse the tree starting at the root
2  if level is MAX
3        choose one branch below it
4  if level is MIN
5        choose all branches below it
6  return the subtree constructed
```

Two Strategies for Max

Strategies in a game tree are subtrees that represent choices of one player. The figure shows two strategies for Max.



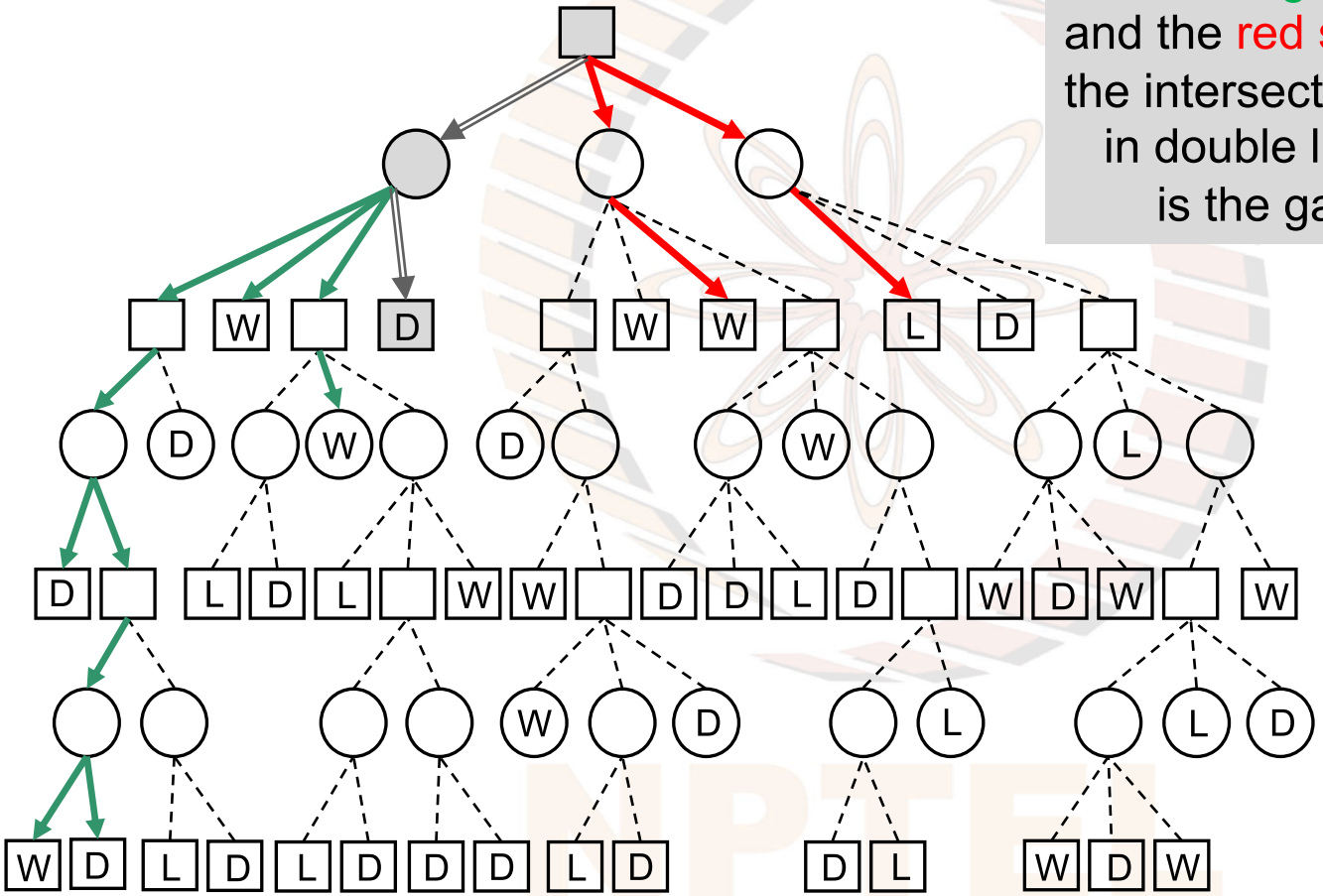
Value of a Strategy



The value of a strategy is as decided by Min, who being rational too, will choose the "minimum" value from a leaf.

The Game Played

Given the green strategy for Max
and the red strategy for Min
the intersection
in double lined arrows
is the game that is played



Complexity of Games = Size of Game Tree

Tic-tac-toe: First player has choice of 9 moves, the second 8 and so on.

Total possible games = 9!

Chess: First player 20 moves, and then the board opens up with more moves becoming possible. On the average there are 35 choices and the game is typically 50 moves long.

Total possible games is $35^{50} = 10^{120}$!!!

10^{120} is a number we find difficult even to comprehend.

IBM's Deep Blue program beat World Champion Garry Kasparov in 1997

Go is played on a 19x19 board!

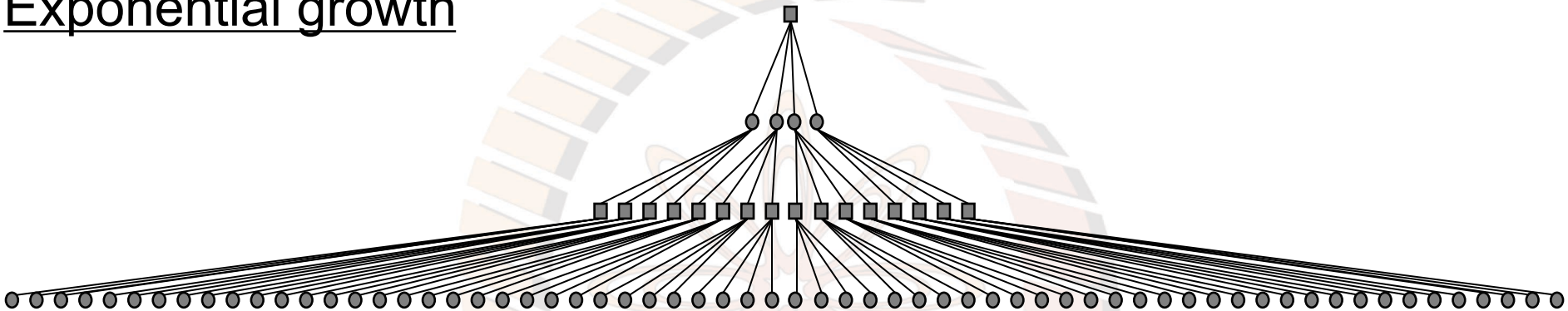
The size of the tree is humungous!

Deep Mind's AlphaGo program

beat World Champion Lee Sedol in 2016

Most game trees
are too big
to be analyzed completely

Exponential growth



A game tree with branching factor 4.

A three ply search needs to inspect 64 nodes.

The next level will have 256,
the one after that 1024, and the next one 4096.

Not trees we can draw on these pages.

Most games have a higher branching factor.

Size of Chess Tree

Total possible games is $35^{50} = 10^{120}$!!!

10^{120} is a number we find difficult even to comprehend.

If we had a a machine that could inspect a trillion or 10^{12} leaves a second

it would require 10^{108} seconds

Given 100 seconds in a minute

it would require 10^{106} minutes

Given 100 minutes in an hour

it would require 10^{104} hours

Given 100 hours in a day

it would require 10^{102} days

Given a thousand days in a year

it would take 10^{99} years

which is 10^{97} centuries

If *every particle in the Universe* was one such computer...

Total possible games is $35^{50} = 10^{120}$!!!

10^{120} is a number we find difficult even to comprehend.

If we had 10^{75} machines and each could inspect 10^{12} leaves a second

it would require 10^{33} seconds

Given 100 seconds in a minute

it would require 10^{31} minutes

Given 100 minutes in an hour

it would require 10^{29} hours

Given 100 hours in a day

it would require 10^{27} days

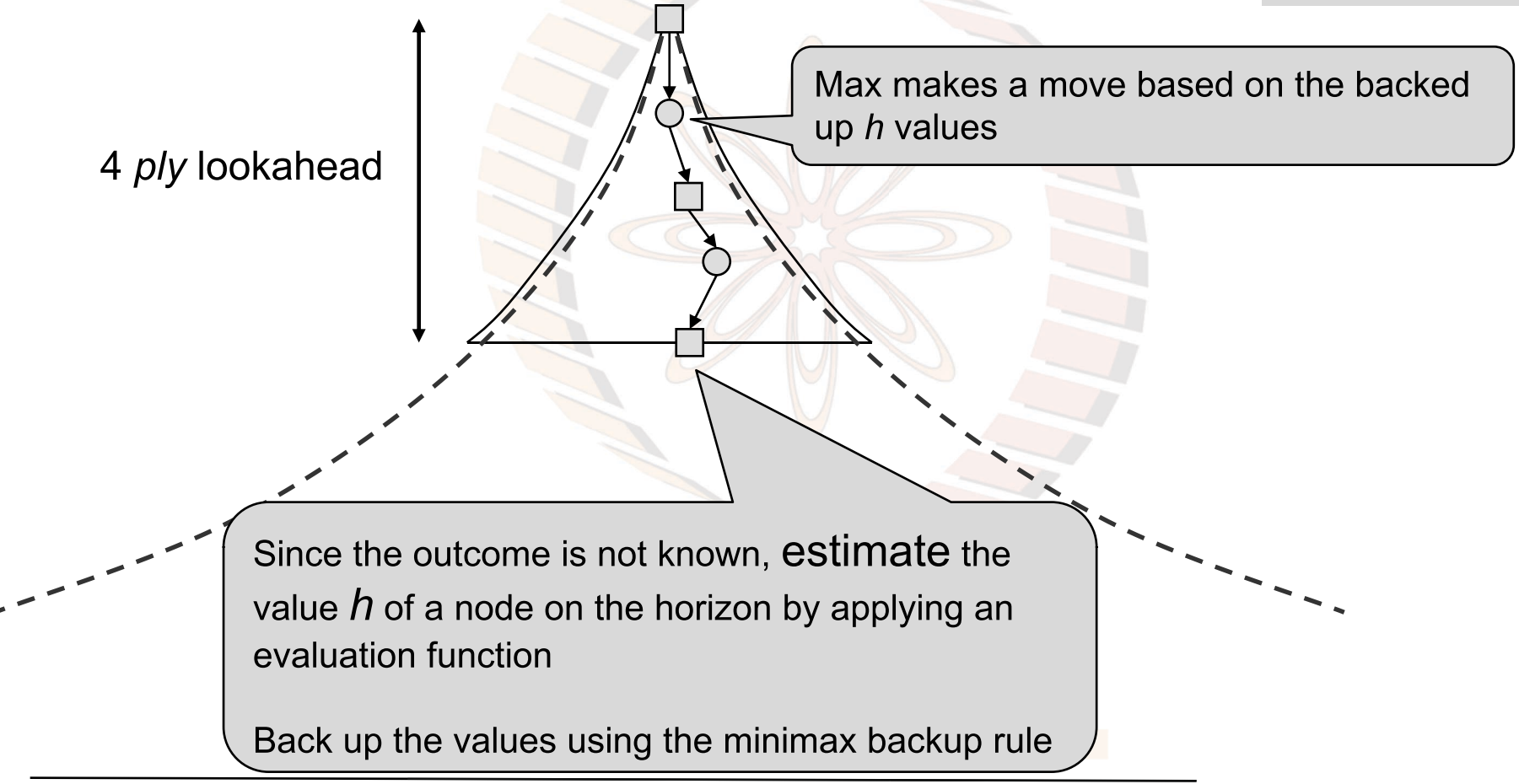
Given a thousand days in a year

it would take 10^{24} years

which is 10^{22} centuries

Finite Lookahead

K *ply* lookahead



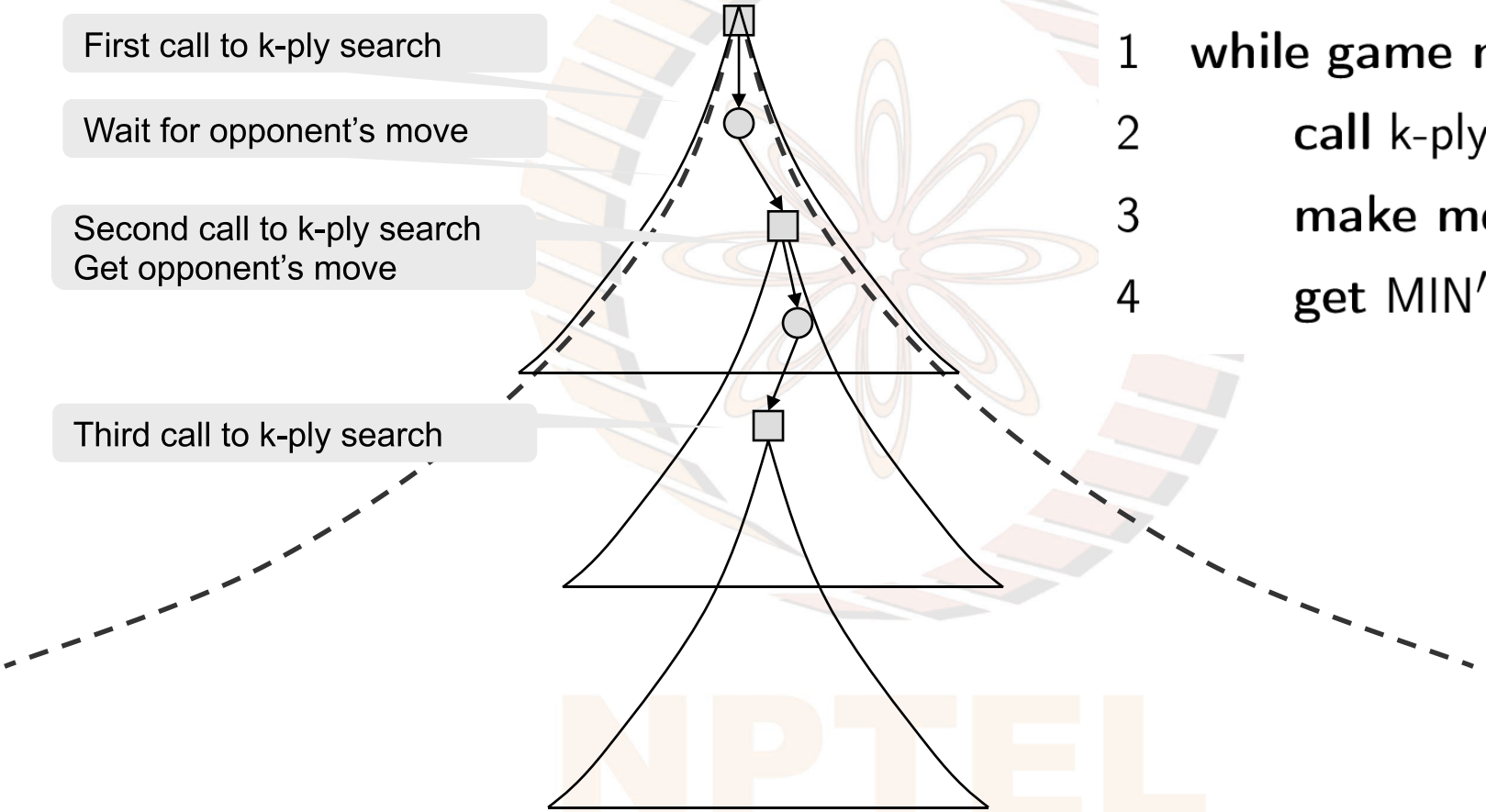
Game Playing

First call to k-ply search

Wait for opponent's move

Second call to k-ply search
Get opponent's move

Third call to k-ply search



GAME-PLAY(MAX)

```
1  while game not over
2      call k-ply search
3      make move
4      get MIN's move
```

The Evaluation Function *h*

The evaluation function is a *static* function,
like the *heuristic function*, that *looks at a board position* and returns a value in some range [-Large, +Large]

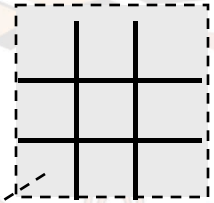
This interval is convenient scaling up of [-1, +1] which converts the set {-1,0,+1} of outcomes into a continuous interval

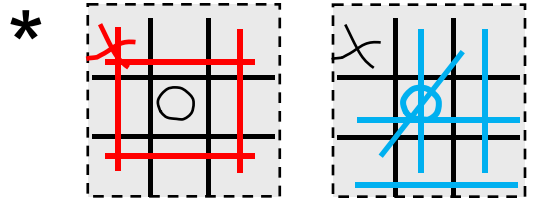
+1 is still a *win* for *Max*, and -1 for *Min*

Positive values are good for *Max*,
and negative ones good for *Min*

Typically the evaluation function is computed by
adding up features capturing *material* strength,
along with features representing *positional* strength

Evaluation fn.: Tic-tac-toe



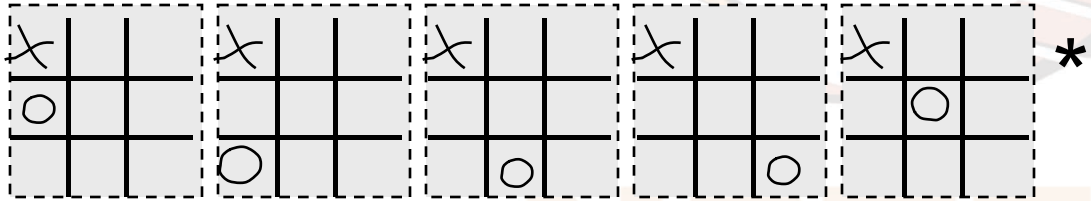


backed up value of node A = -1

A

2 ply lookahead

eval(N) =
number of free rows, columns, diagonals for Max
– number of free rows, columns, diagonals for Min



6-5=1 5-5=0 6-5=1 5-5=0 4-5=–1

A simple evaluation function: Chess

attributed to Claude Shannon

KQRBNP = number of kings, queens, rooks, bishops, knights and pawns
D,S,I = doubled, blocked and isolated pawns
M = Mobility (the number of legal moves)

$$
\begin{aligned}
f(p) = {} & 200(K-K') \\
& + 9(Q-Q') \\
& + 5(R-R') \\
& + 3(B-B' + N-N') \\
& + 1(P-P') \\
& - 0.5(D-D' + S-S' + I-I') \\
& + 0.1(M-M') + \ldots
\end{aligned}
$$

weights

The evaluation function of the program *Deep Blue* has about 8000 components (Campbell et al., 2002).

other positional features ... king safety, center control, pawn structure, king tropism, doubled rooks + ...

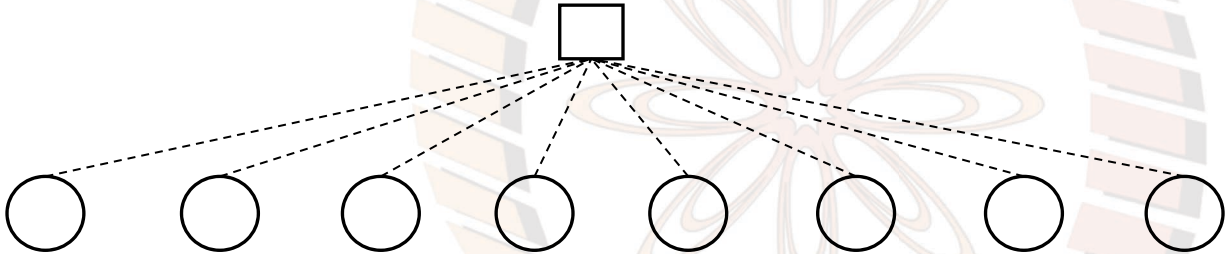
Chess: positional features

Concerned with mobility, board control, development, pawn formations, piece combinations, king protection etcetera.

- two rooks in the same column have added value;
- a pair of bishops is better than a bishop and a knight;
- knights are valuable in certain kinds of end positions.
- pieces must become mobile and either occupy the center or control it.
- chained pawns that support each other are better than isolated pawns;
- pawns in the same column or opposing pawns head to head are not good;
- pawns heading for promotion have added value.
- The king needs protection in the opening and middle games, and structures like those obtained by castling are valued high;
- In the end game the king may be an offensive piece
- Pins, forks and discovered attacks also need to be considered while computing positional value.

Evaluation vs. Lookhead

If we *have* an evaluation function then why do we need to do a lookhead at all? Why not just do 1-ply search?

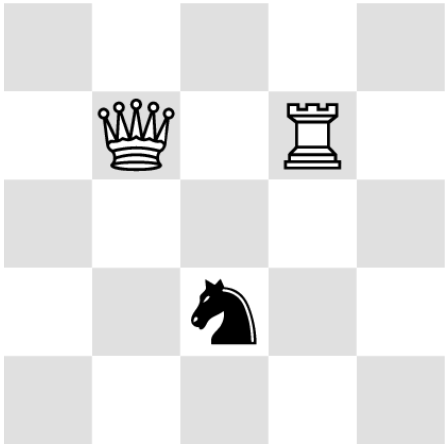


Adriaan de Groot showed that grandmasters do store tens of thousands of *Chess schemata* and evaluate them directly.

But it is difficult to devise an evaluation function that is good enough to play with one ply lookahead.

We rely on a combination of evaluation and lookahead.

Evaluation vs. Lookhead



Sometimes an evaluation function tells us something that further lookahead would have revealed.

The fork on the right is a pattern in which the black knight simultaneously attacks two important white pieces – the queen and the rook.

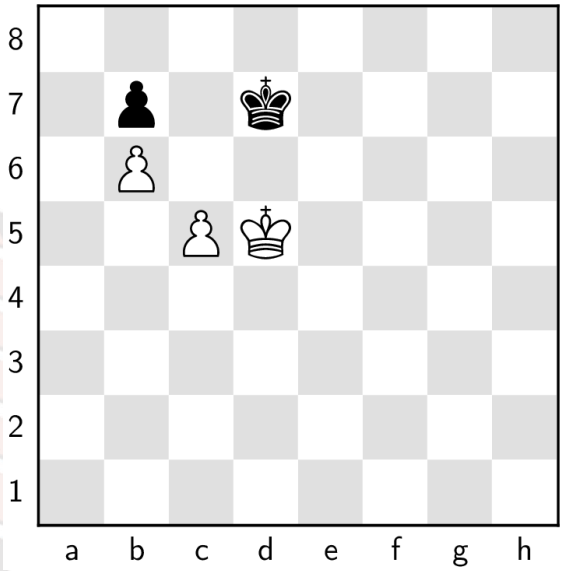
Losing a knight in exchange with one of them would result in material advantage for black.

White can move only one of the two away.

This could have been discovered by further lookhead.

But remember that the evaluation function is applied on the horizon, where lookahead ends.

The Zugzwang



Zugzwang (German for "compulsion to move”), is a situation found in chess and other games wherein one player is put at a disadvantage because they must make a move when they would prefer to pass and not move.

A player is said to be "in zugzwang" when any possible move will worsen their position

In the position shown here if white is to move black is safe.

But if black is to move, then the black pawn, and subsequently the game, will be lost.

Some evaluation functions includes “turn to move”.

The Zugzwang

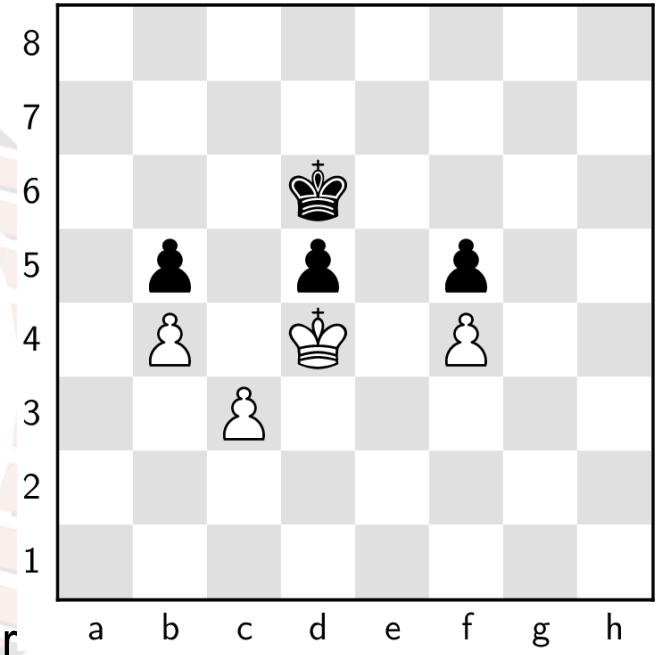
Another zugwang position for black

If white is to move black is safe, because white cannot attack any black pawn.

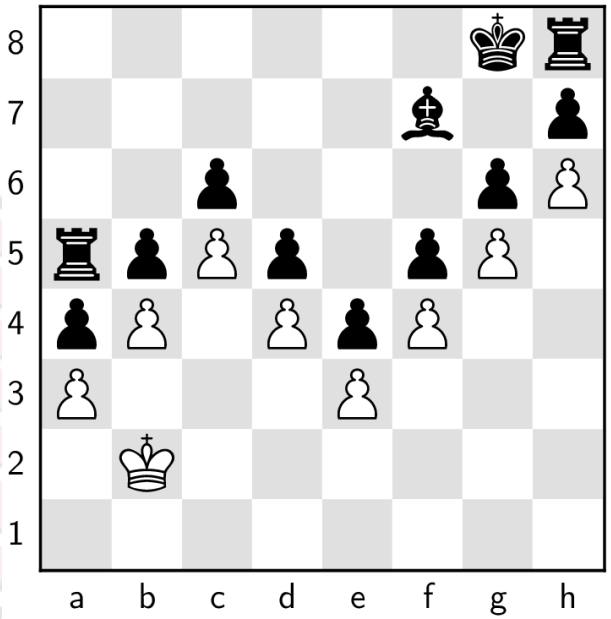
If white pushes the pawn then black will capture it with the middle black pawn.

But if black is to move then black is forced to withdraw protection from one of its black pawns.

White will then be able to capture it clearing the way for its own pawn advancing and becoming a queen.



Limitations of finite lookahead



The board shows a contrived chess position known as *The Poisoned Rook.*

Deep Thought 2, which could selectively look ahead 10 or 11 ply captured the black rook with the white pawn.

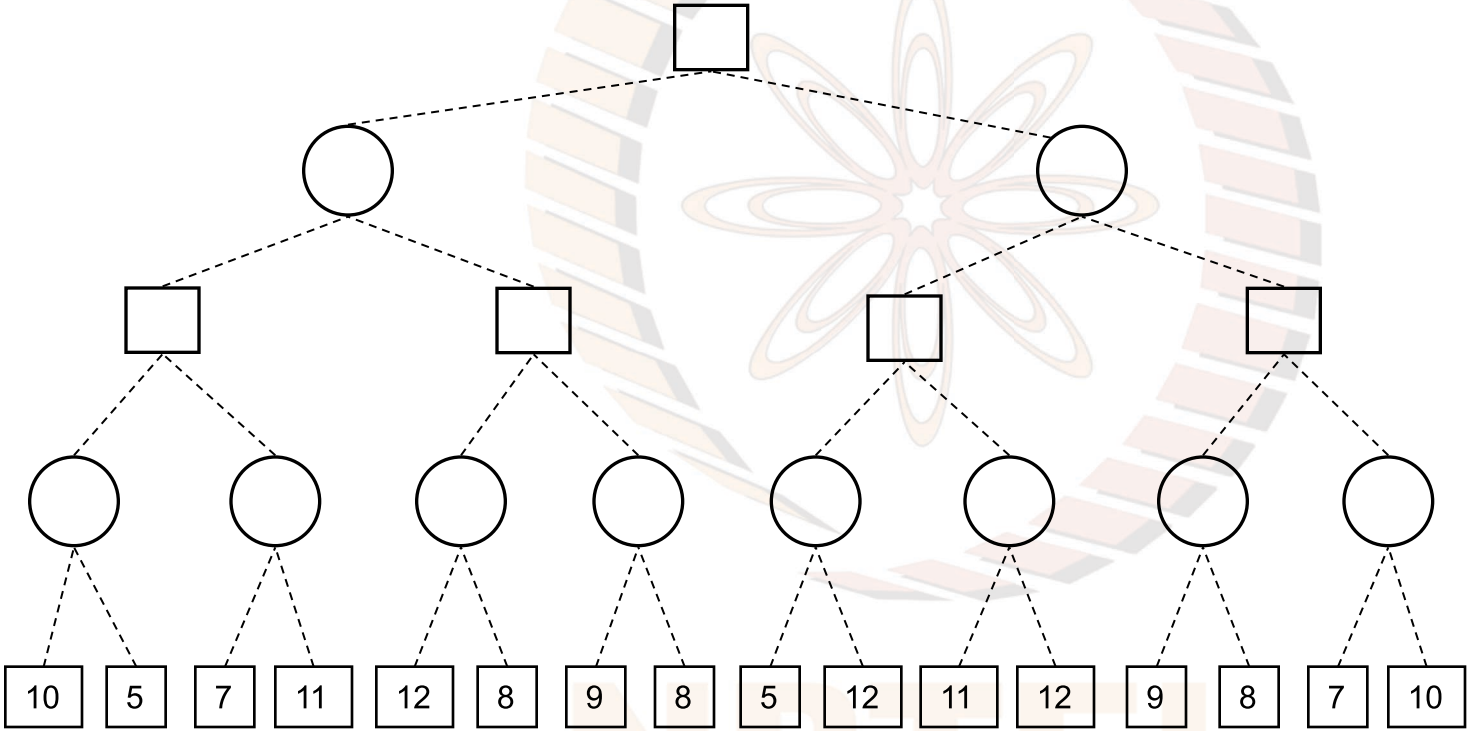
It eventually went on the lose the game.

Human players quickly realize that the contrived pawn structure makes white's region impregnable.

This is because black does not have a black bishop that could have attacked a white pawn.

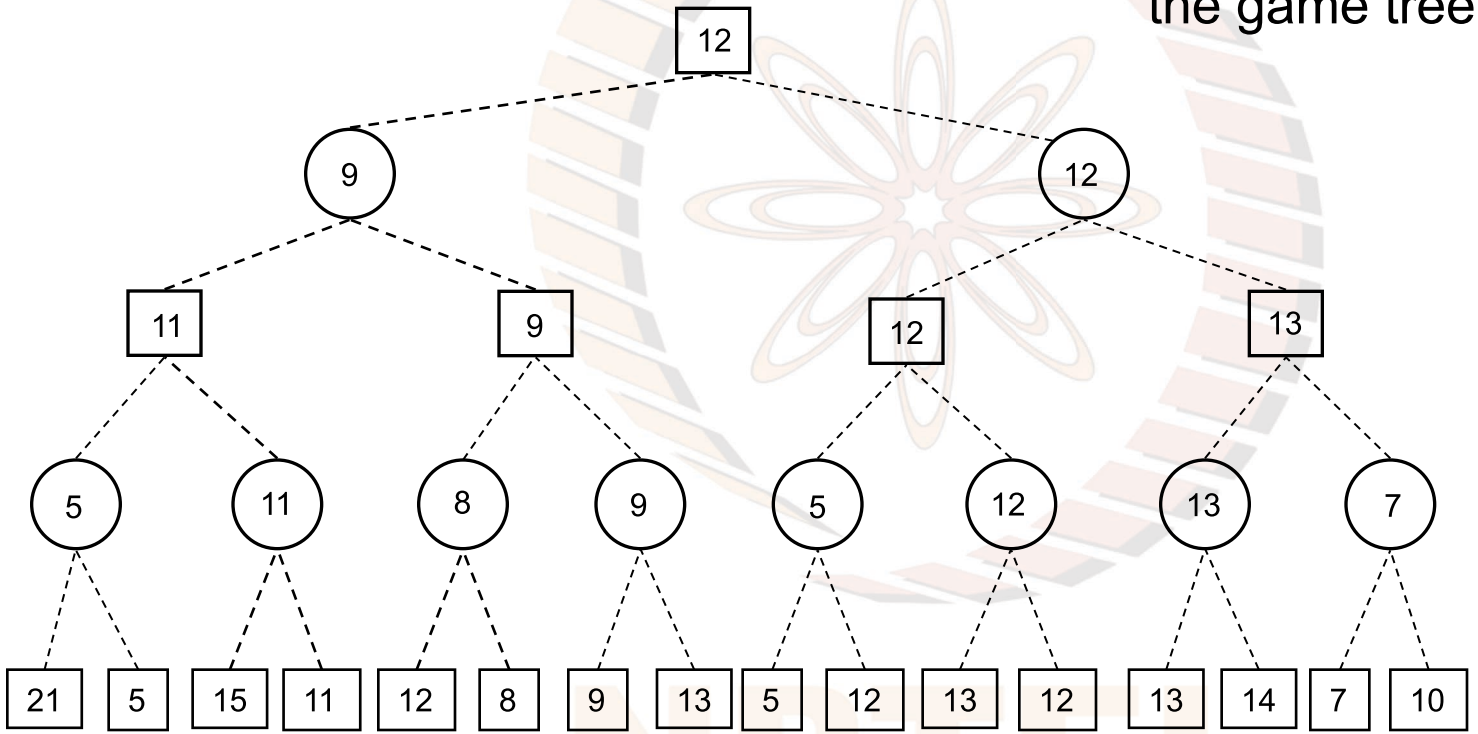
Black's powerful pieces are unable to threaten the white king.

A Binary game tree



The minimax value

Applying the minimax backup rule determines the value of the game tree



The MiniMax Algorithm

```
MINIMAX(N)
1   if N is a terminal node
2         return eval(N)
3   If N is a MAX node
4         value ← −∞
5         for each child C of N
6               value ← max(value, MINIMAX(C))
7   else ▷ N is a MIN node
8         value ← ∞
9         for each child C of N
10              value ← min(value, MINIMAX(C))
11  return value
```

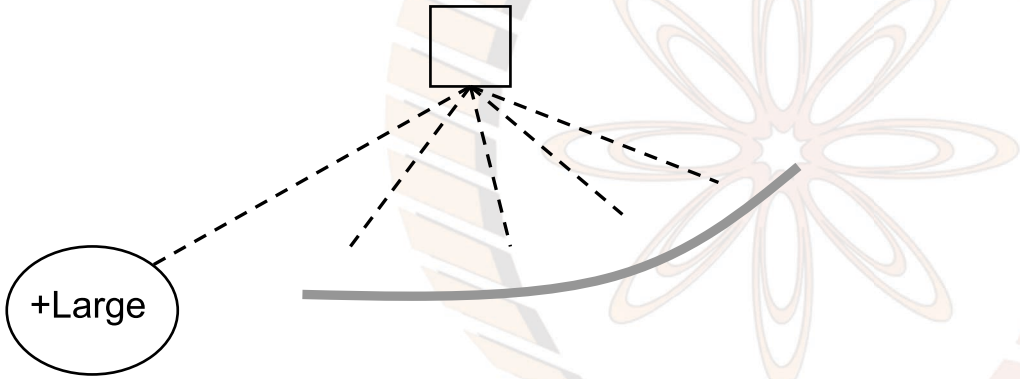
Initialize *Max* to *-Large*
Look for higher value

Does a Depth First Traversal of the tree applying the minimax backup rule

Recursive Version

Pruning the search tree

Consider the following situation for Max



When Max has found a winning move it does not need to explore any further.

This notion can even be extended to moves that are not winning moves

Alpha nodes and Beta nodes

We adopt the following terminology and the corresponding observations.

We call *Max* nodes as *Alpha* nodes which store the *alpha* values.

We call *Min* nodes as *Beta* nodes which store the *beta* values.

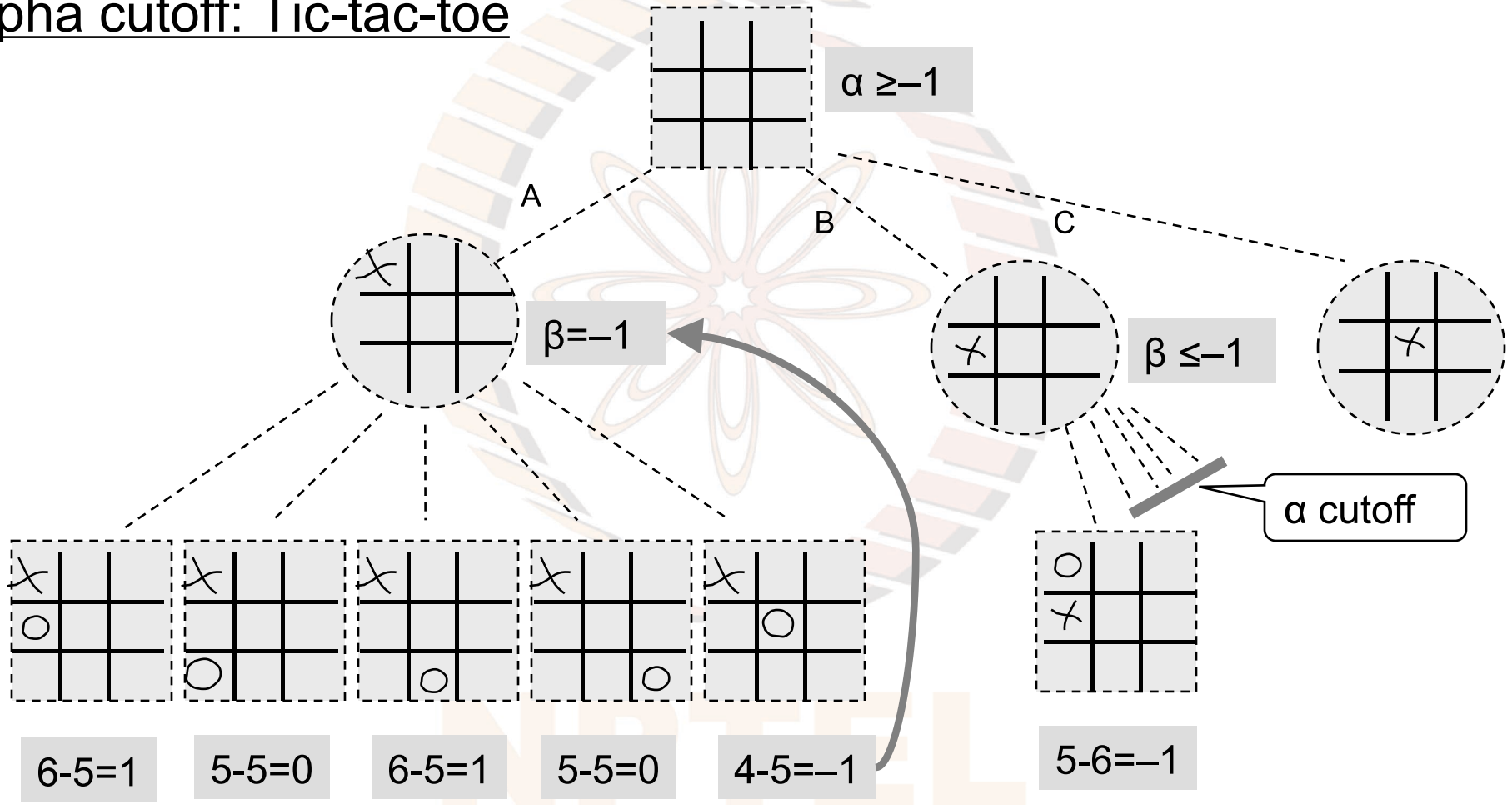
Alpha value is the *value found so far* for the alpha node and it is *a lower bound* on the value of the node.

It can *only be revised upwards*.
It *will reject any lower values subsequently*.

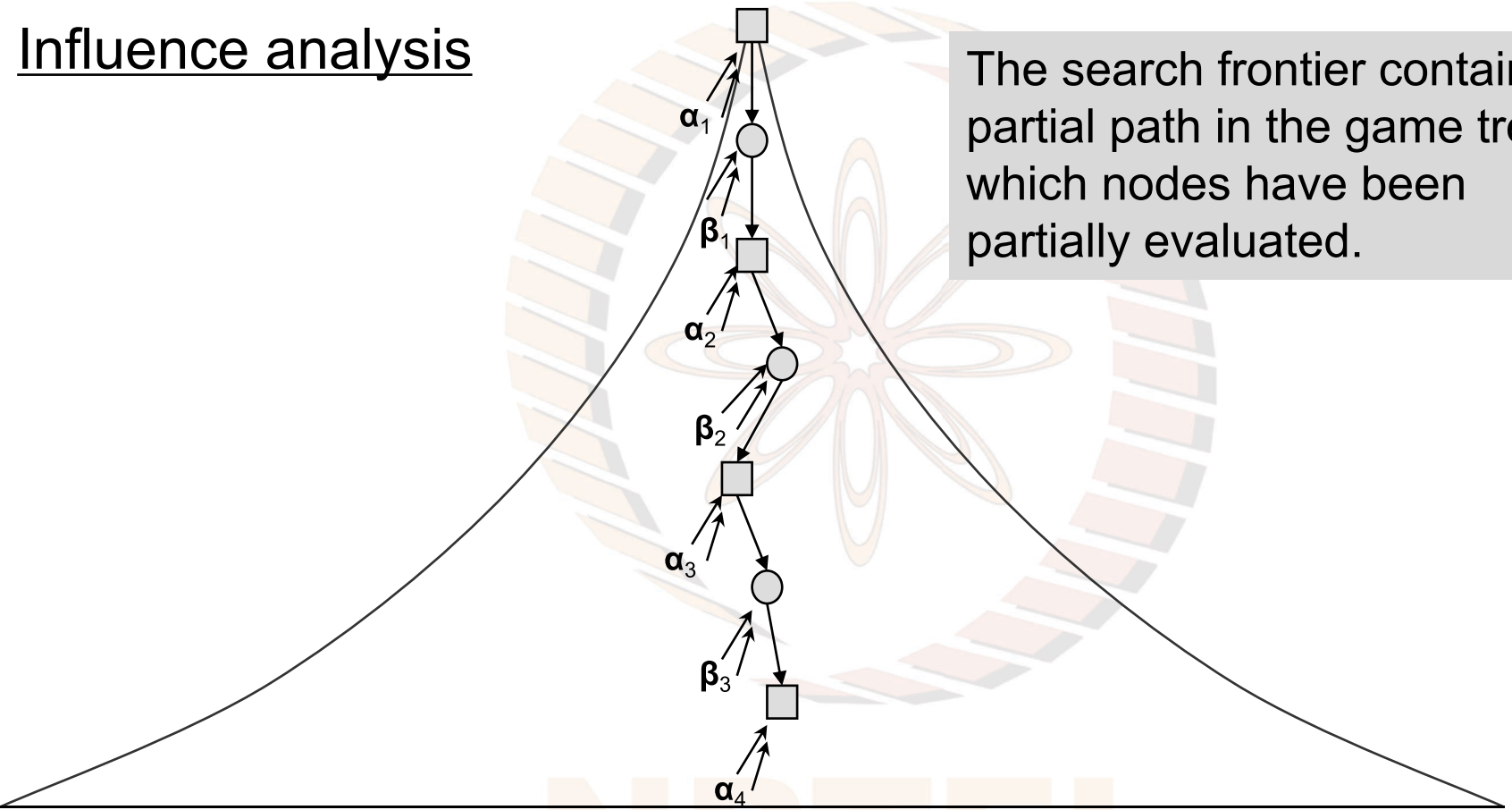
Beta value is the *value found so far* for the beta node and it is *an upper bound* on the value of the node.

It can *only be revised downwards*.
It *will reject any higher values subsequently*.

Alpha cutoff: Tic-tac-toe



Influence analysis



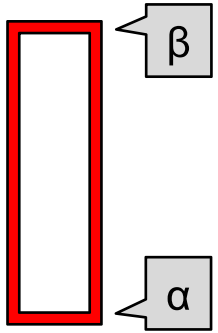
The search frontier contains a partial path in the game tree in which nodes have been partially evaluated.

courtesy Judea Pearl

Influence analysis

$\beta = \min(\beta_3, \beta_2, \beta_1, \ldots)$
$\alpha = \max(\alpha_3, \alpha_2, \alpha_1, \ldots)$
continue as long as $\alpha < \beta$

Node J will influence the root if
$\alpha_4 < \beta_3$, $\alpha_4 < \beta_2$, $\alpha_4 < \beta_1$
and
$\alpha_4 > \alpha_3$, $\alpha_4 > \alpha_2$, $\alpha_4 > \alpha_1$,

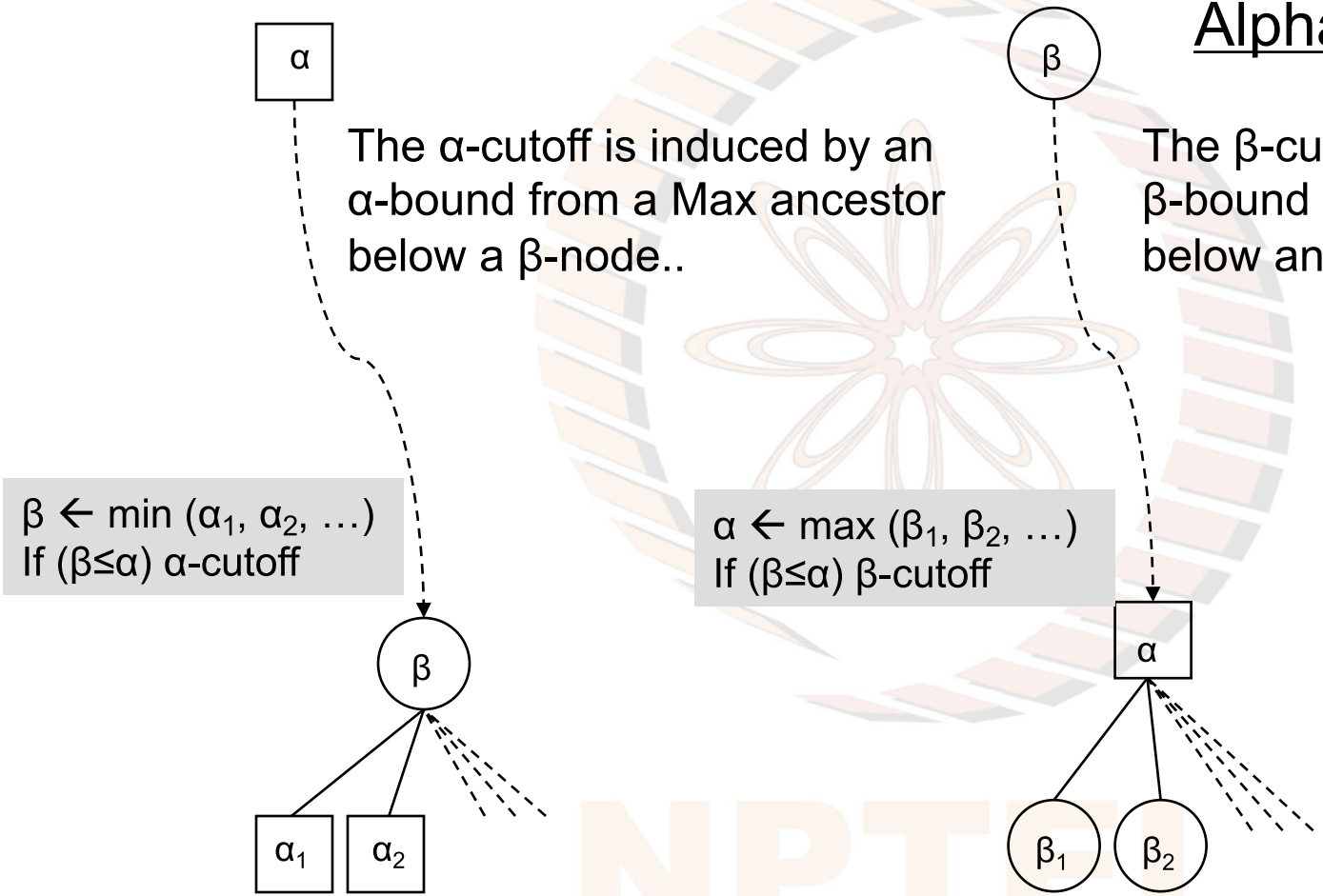


courtesy Judea Pearl

Alpha and Beta cutoffs

The α-cutoff is induced by an α-bound from a Max ancestor below a β-node..

The β-cutoff is induced by a β-bound from a Min ancestor below an α node.



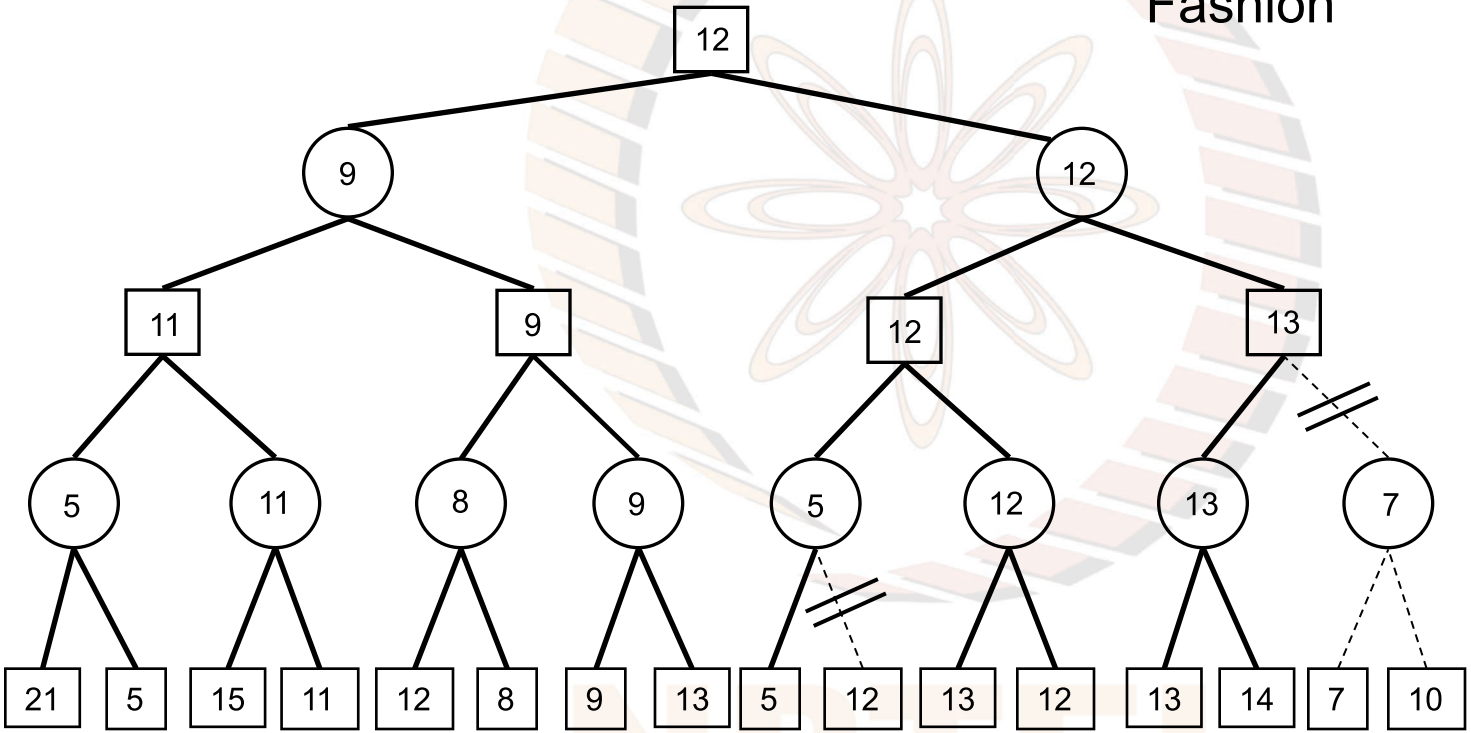
The AlphaBeta Algorithm

Initially **α** = -Large
β = +Large

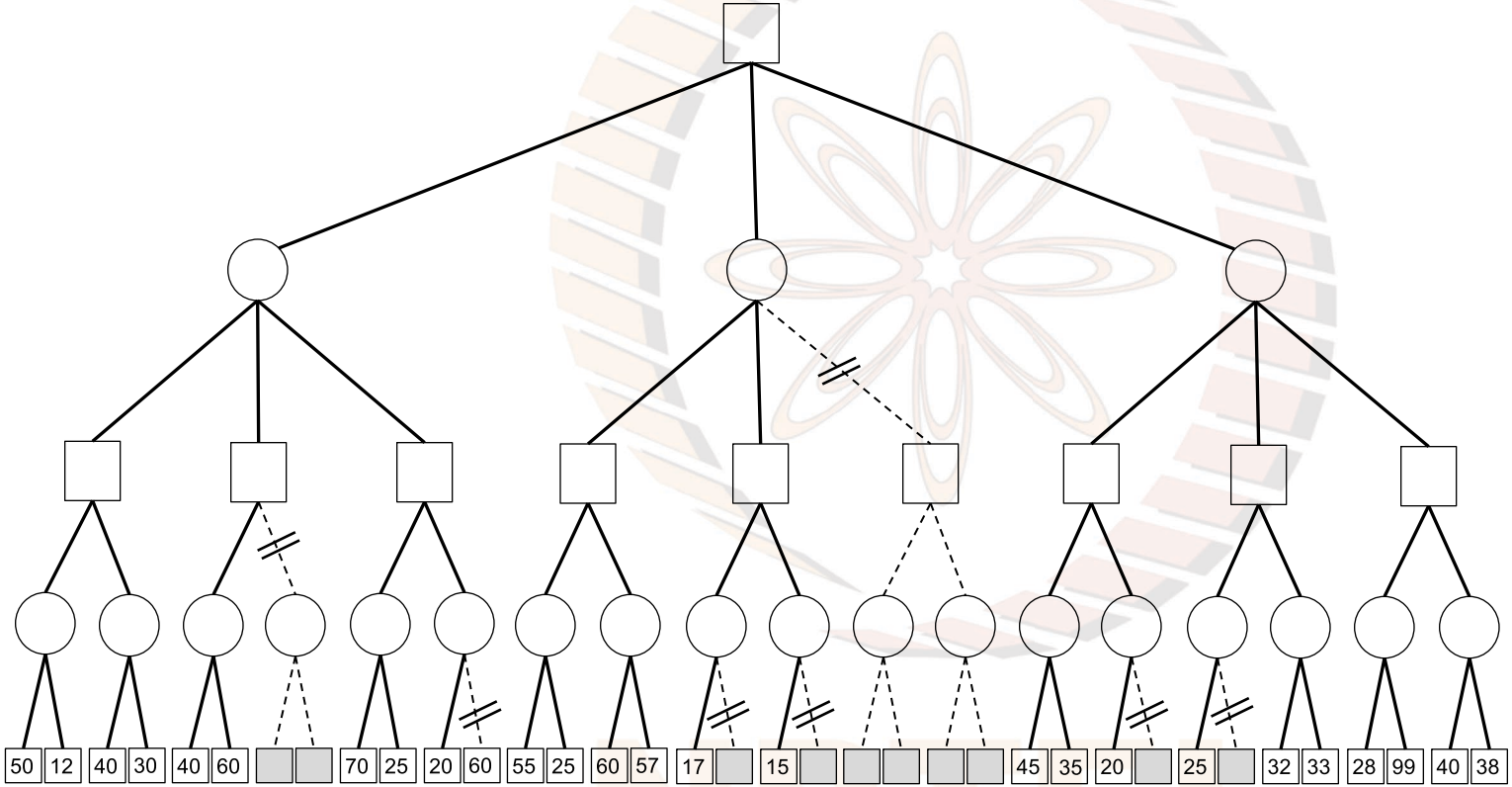
```
ALPHA-BETA(N, α, β)
 1  if N is a terminal node
 2        return eval(N)
 3  If N is a MAX node
 4        for each child C of N
 5              α ← max( α,  ALPHA-BETA(C, α, β) )
 6               if α ≥ β then return β
 7        return α
 8  else ▷ N is a MIN node
 9        for each child C of N
10              β ← min( β,  ALPHA-BETA(C, α, β) )
11               if α ≥ β then return α
12        return β
```

A small example

Both Minimax and AlphaBeta search in a Depth First Fashion

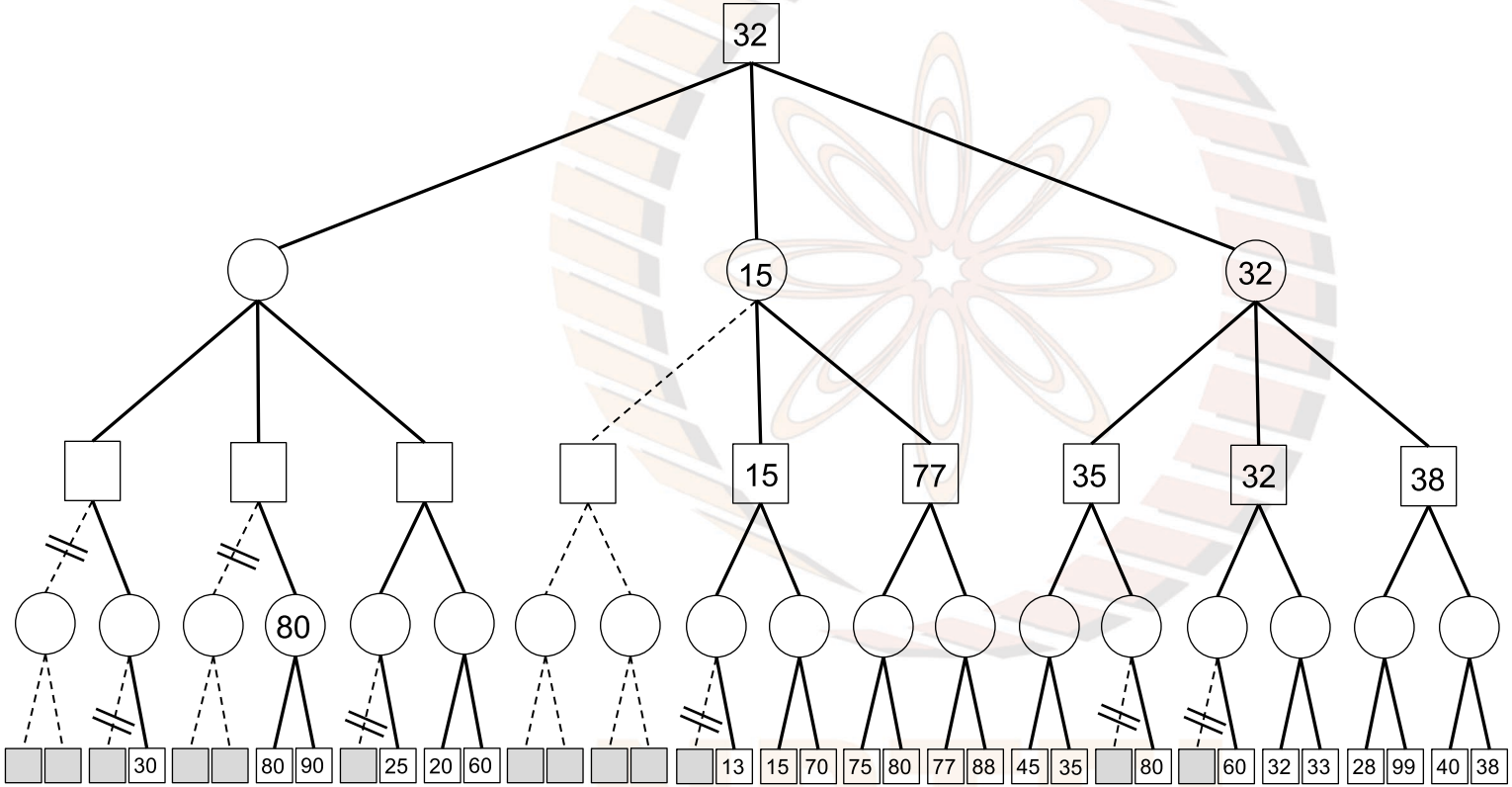


A larger game tree: AlphaBeta Search



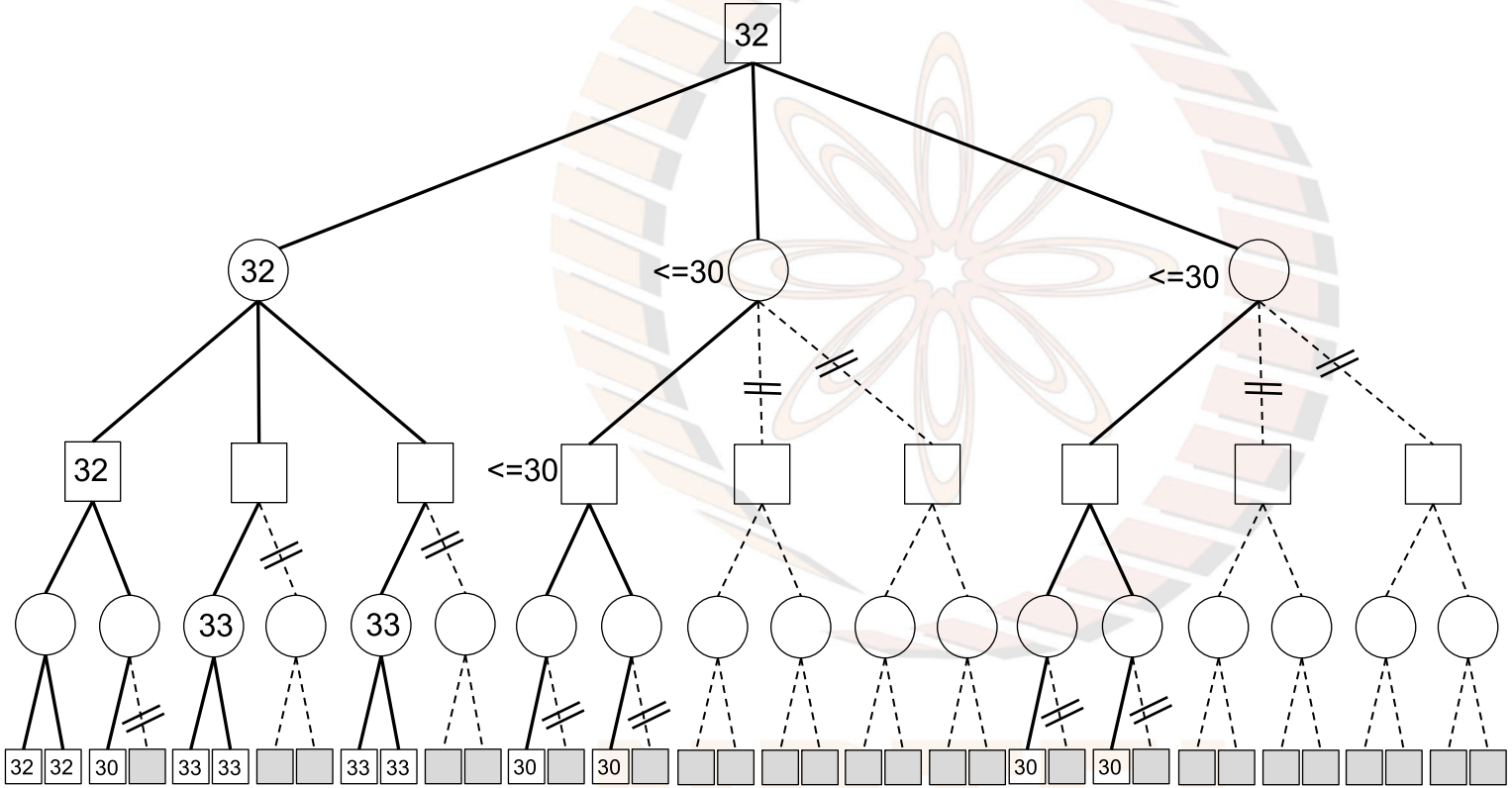
AlphaBeta inspects 26 of the 36 nodes

AlphaBeta: Searching from right to left



AlphaBeta inspects 23 of the 36 nodes

An extreme example of pruning



AlphaBeta inspects 11 of the 36 nodes

AlphaBeta: Reordering move generation

AlphaBeta pruning *depends on the order of move generation*.

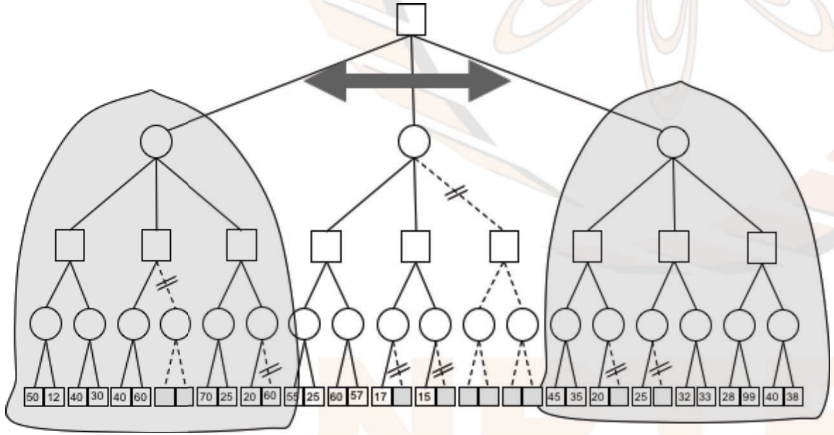
Having called k-ply search, *while waiting for opponent's move*, one can reorder the moves for the *next* call to k-ply.

The MiniMax value *does not change*.

The same value is found, *but faster*.

GAME-PLAY(MAX)

```
1  while game not over
2      call k-ply search
3      make move
4      get MIN's move
```

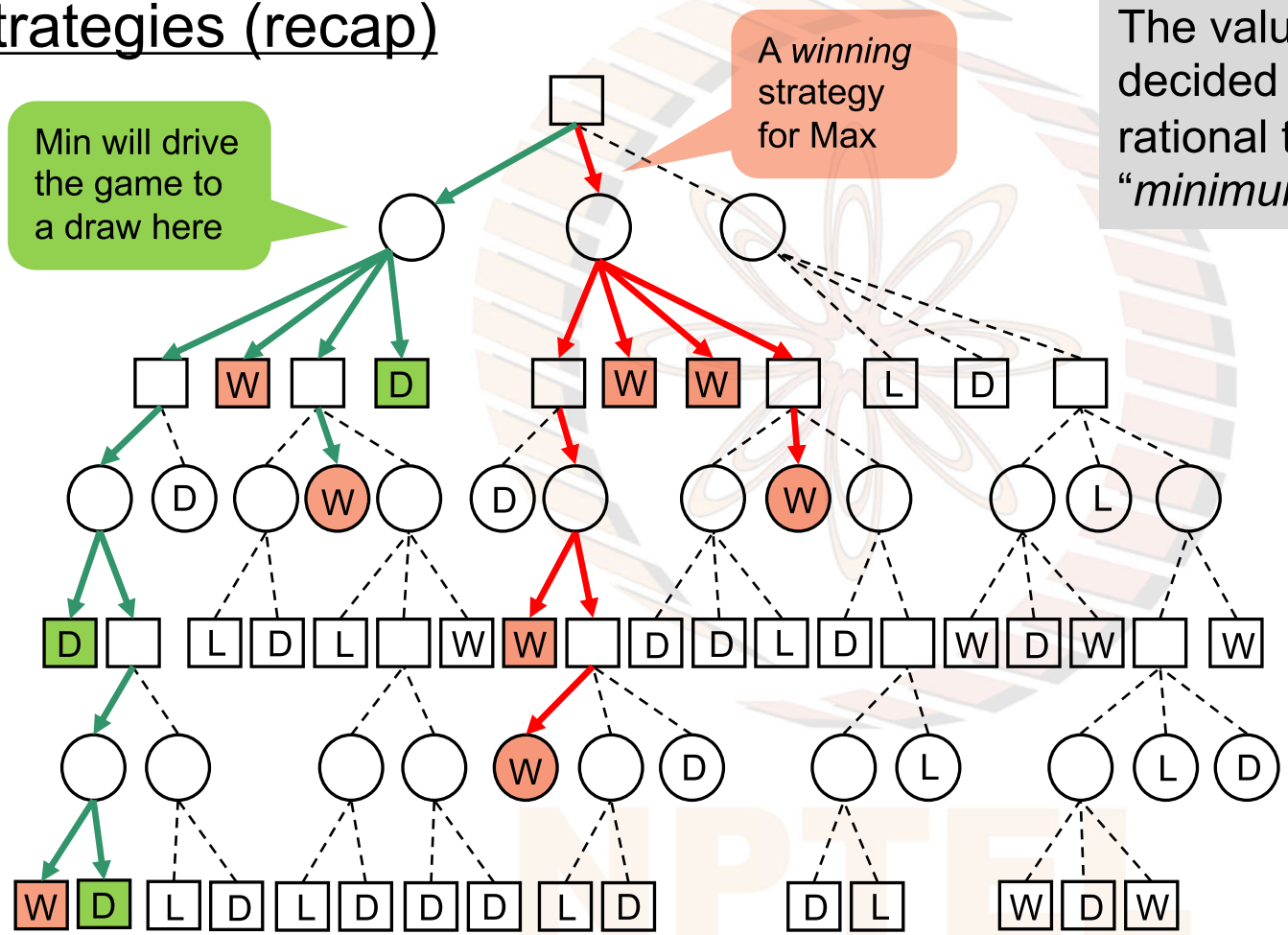


Next

A *Best First* Version of a Game Playing direction

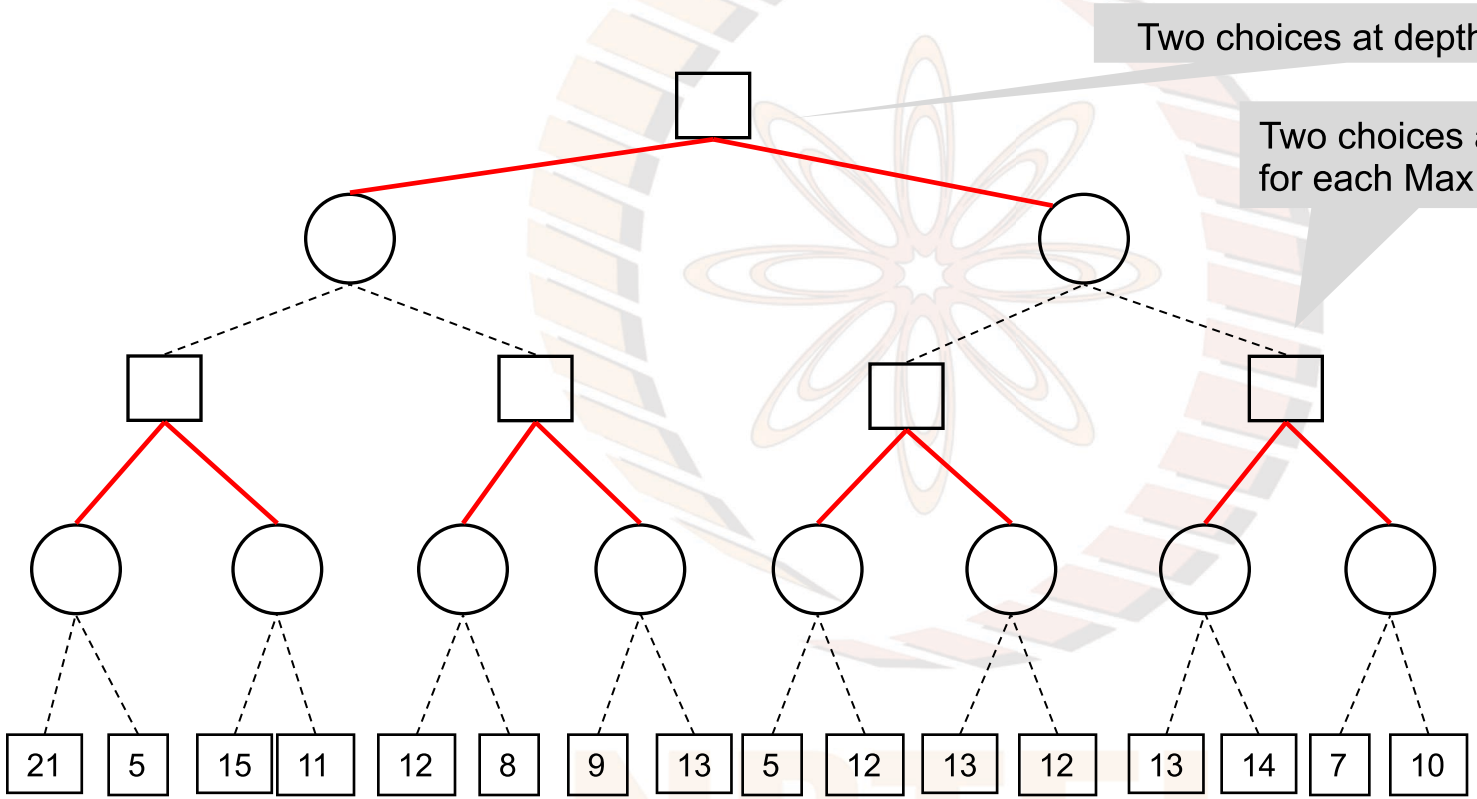
An algorithm with *a sense of direction*

Strategies (recap)



The value of a strategy is as decided by Min, who being rational too, will choose the "*minimum*" value from a leaf.

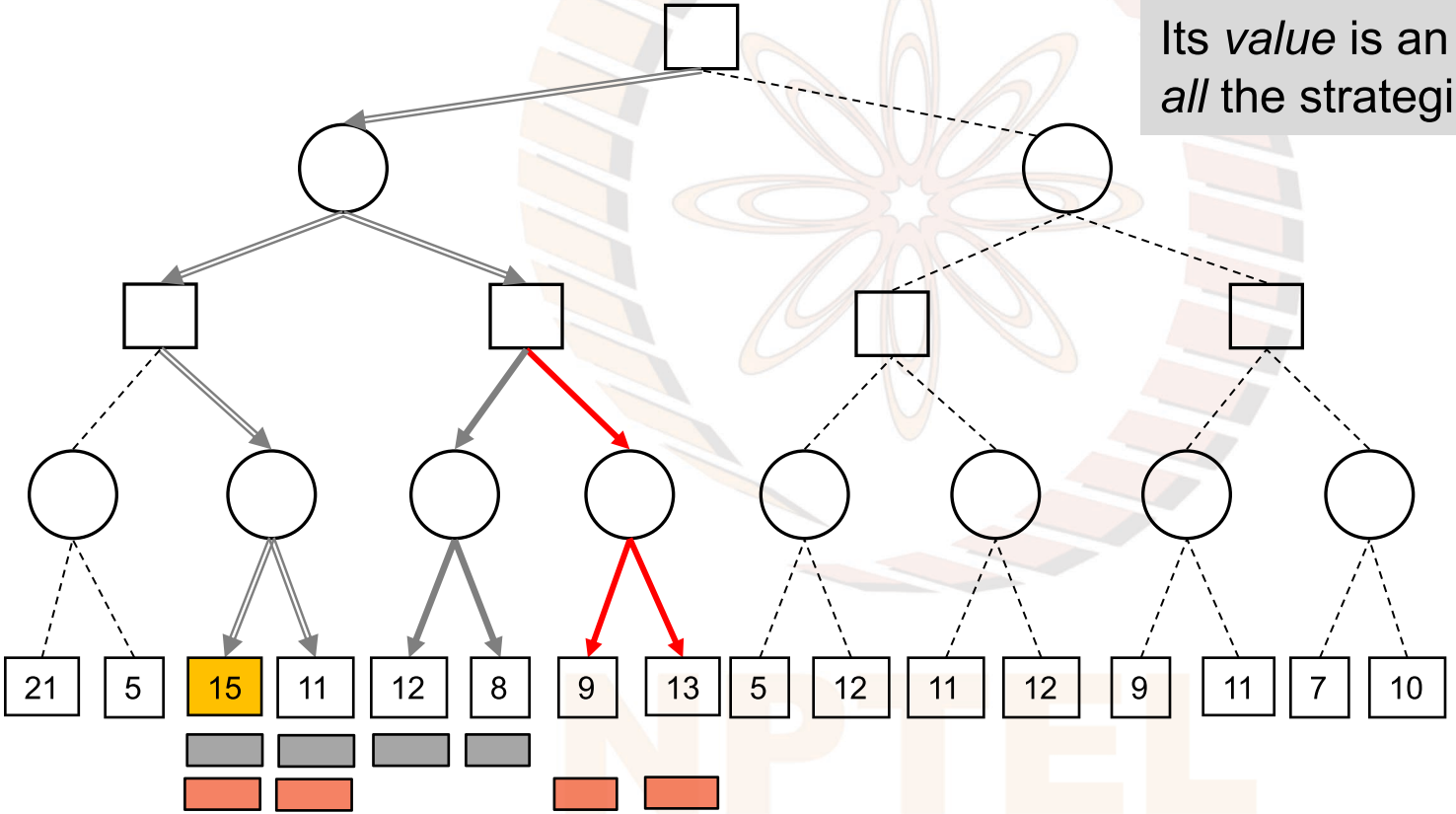
This small binary game tree has 8 strategies



A Cluster of Strategies

A node represents a cluster of partial strategies

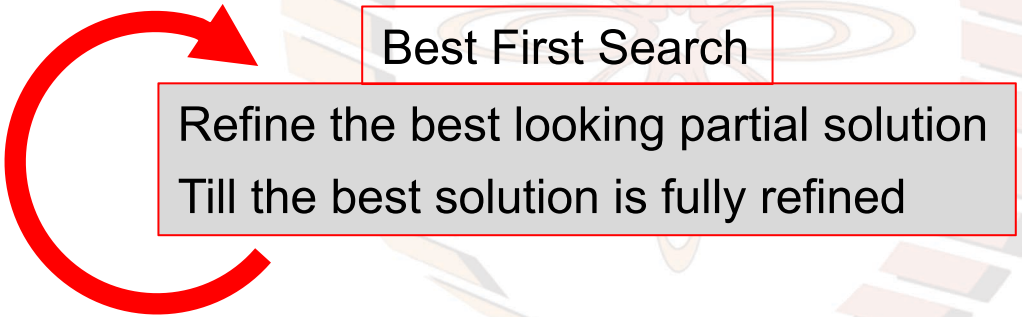
Its *value* is an *upper bound* on *all* the strategies in the cluster



Algorithm SSS*

Devised by George Stockman in 1979

A Best First approach to searching the game tree



A solution in a game tree is a *strategy*

A *partial solution* is a *partial strategy*
and *stands for a cluster of strategies*

Algorithm SSS* searches in the Solution Space

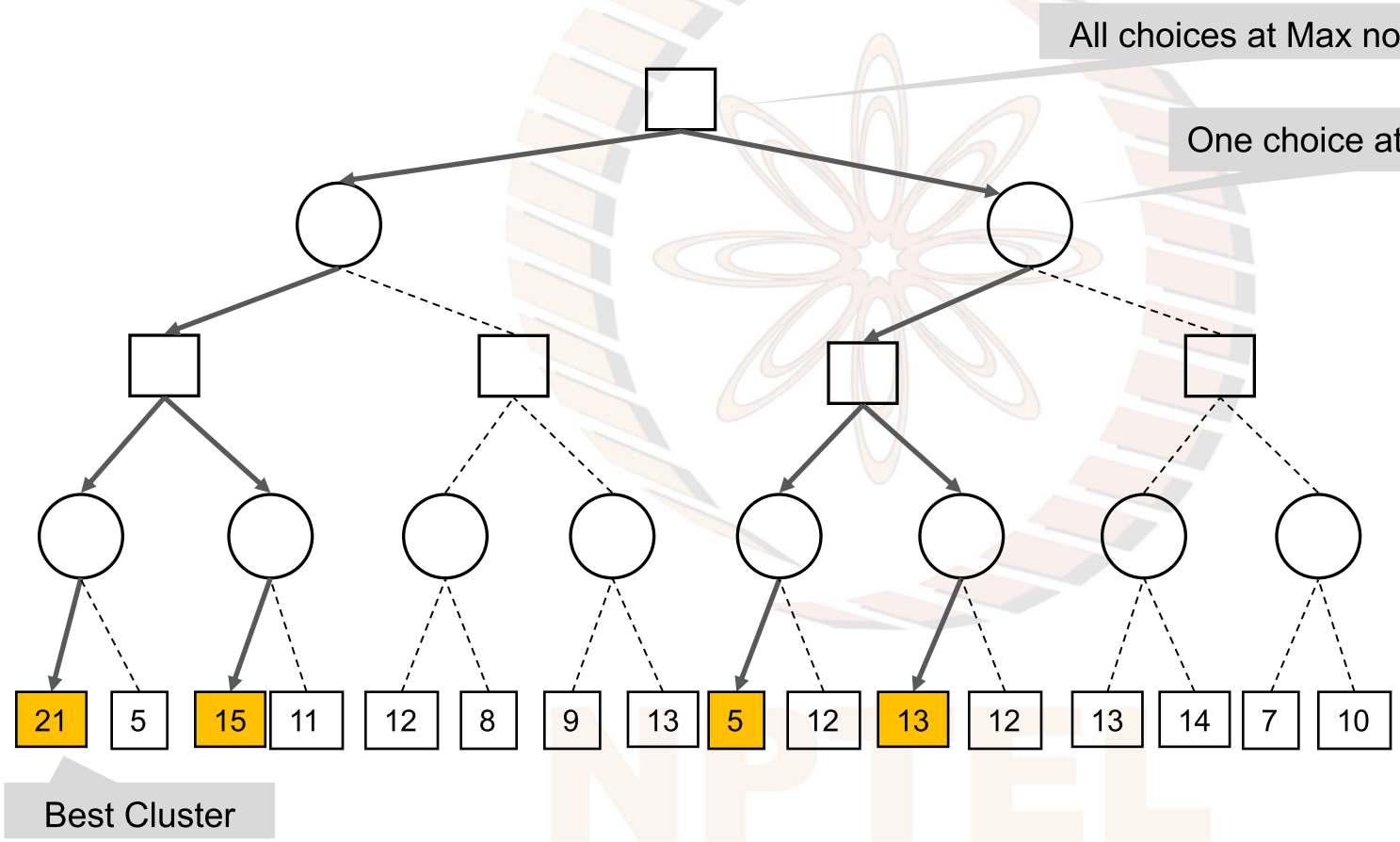
Algorithm SSS* first defines clusters to cover *all* strategies
- necessary for completeness

The procedure for constructing the initial clusters is as follows

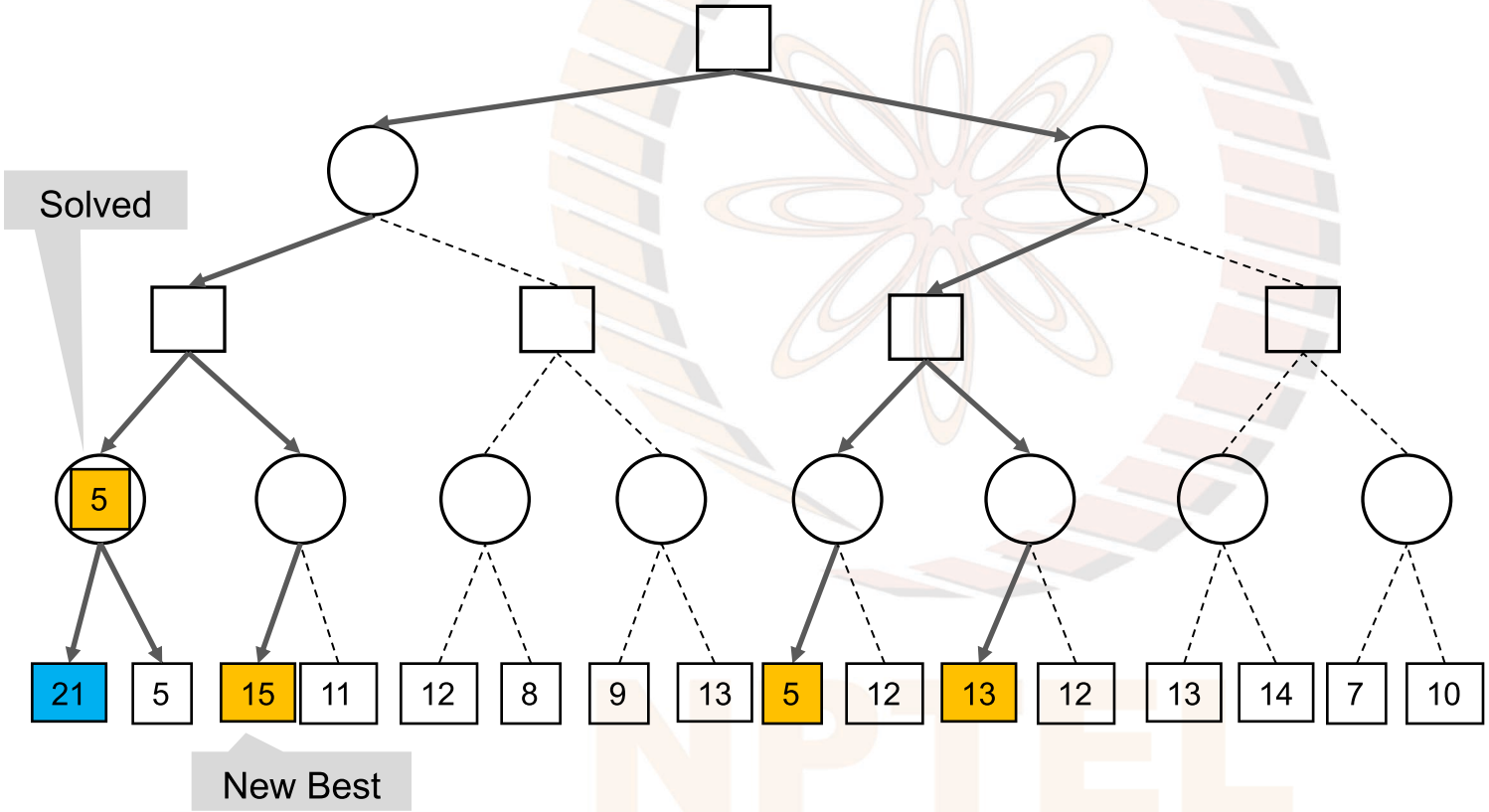
Start at the root
Repeat
    At the Max level choose *all* children
    At the Min level choose *one** child
Until the horizon is reached

* Without loss of generality we always choose the leftmost child

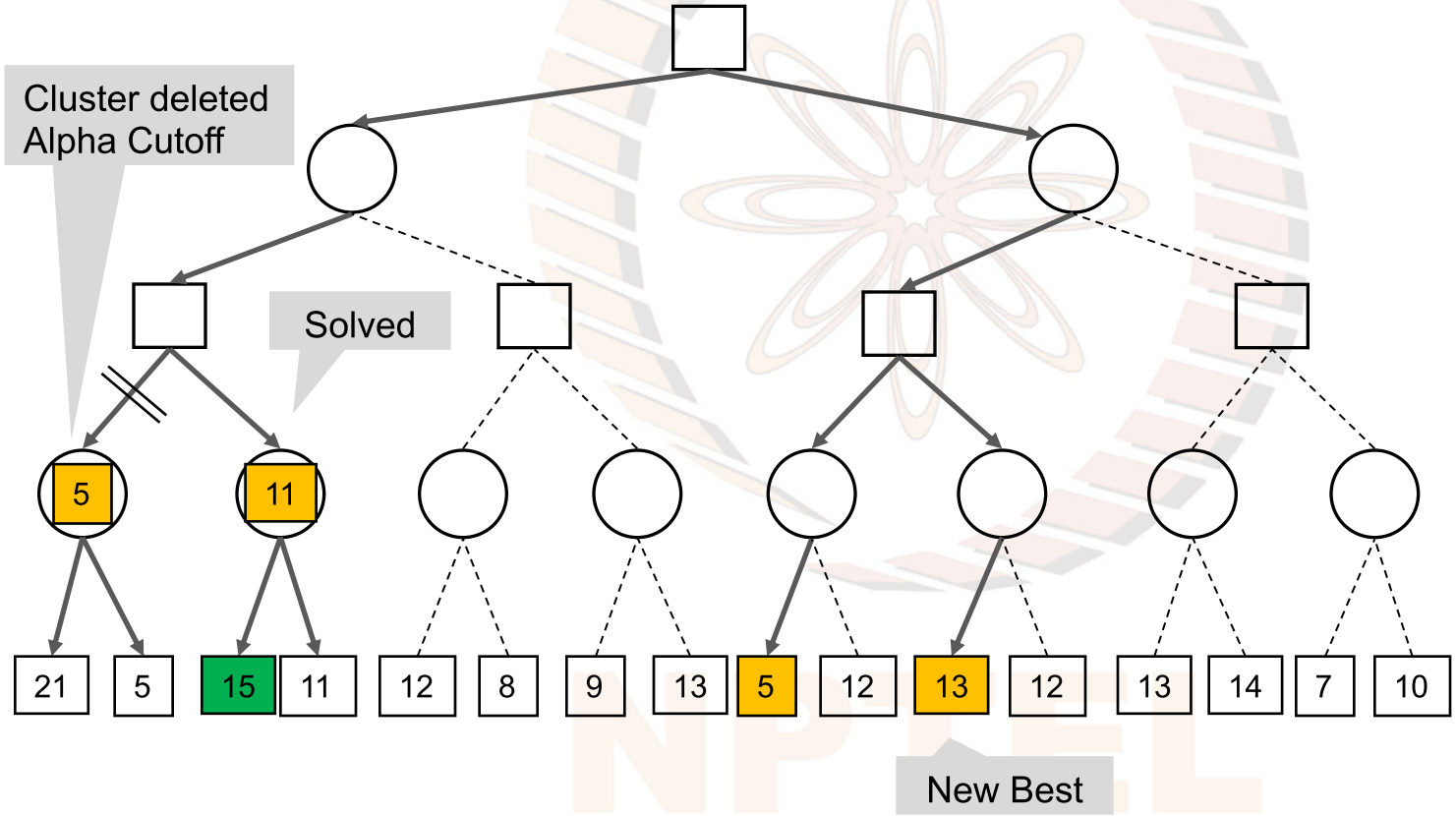
Eight strategies in four initial clusters



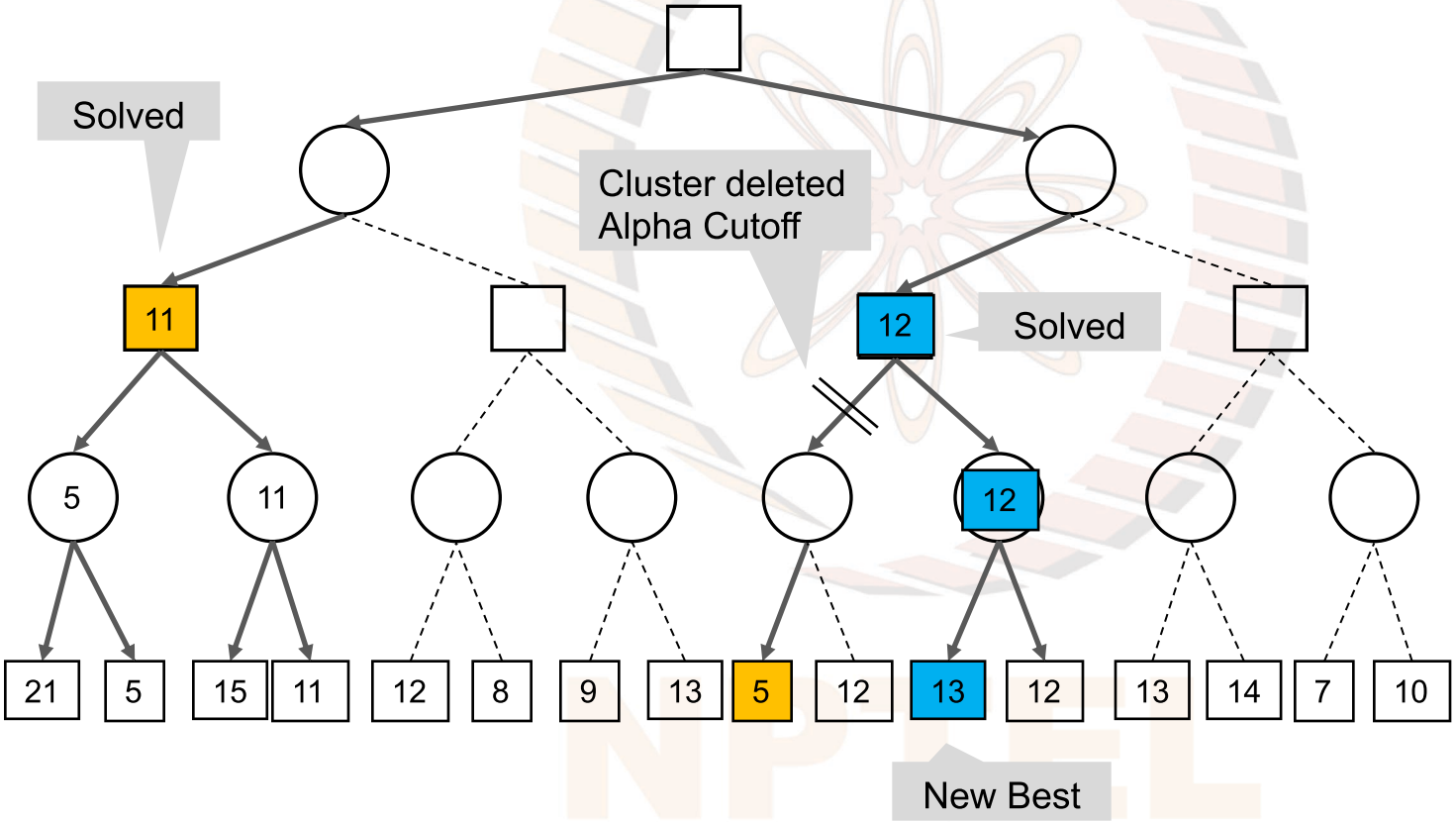
Refine the best partial strategy



Refine the best partial strategy



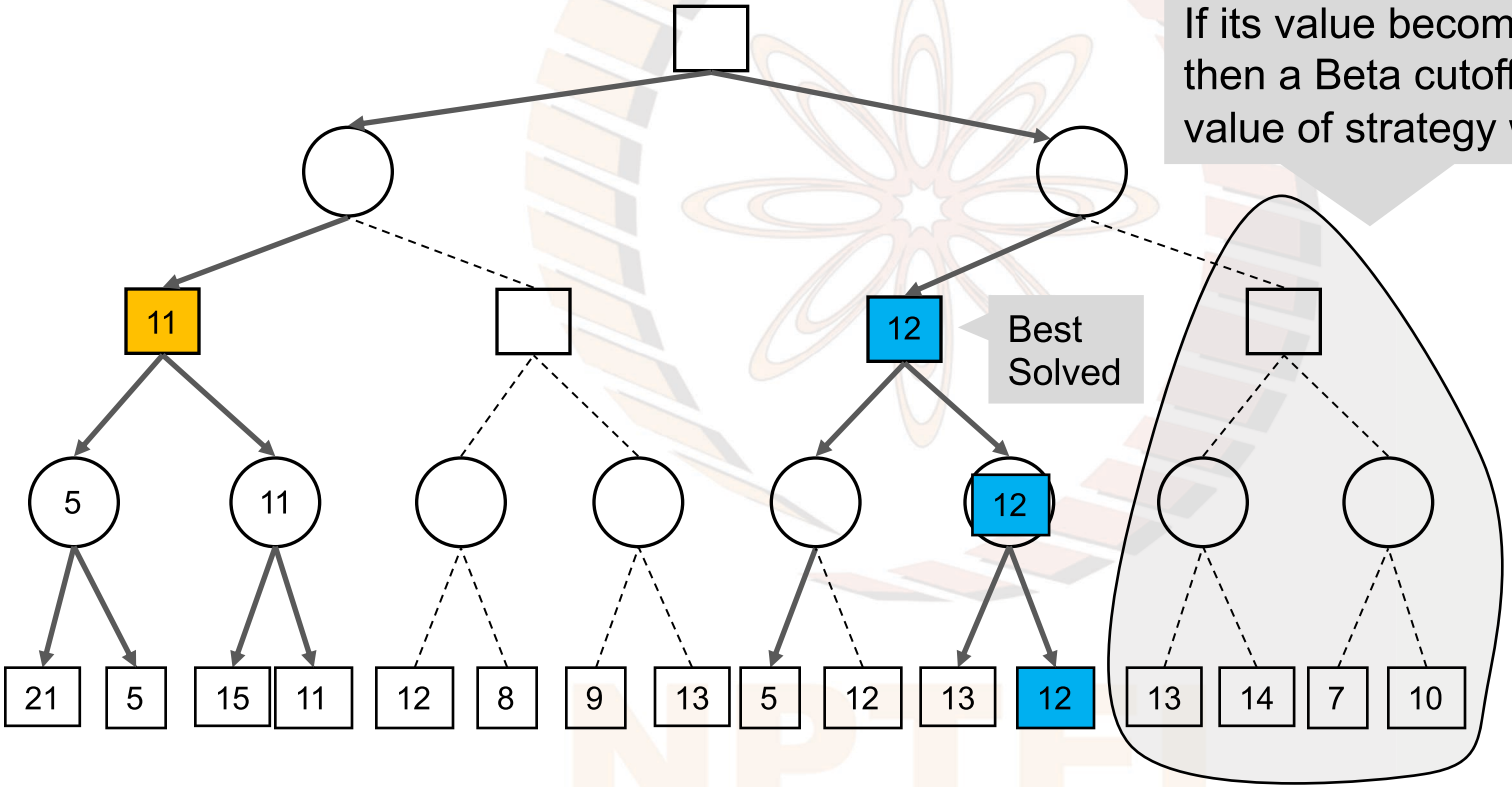
Refine the best partial strategy



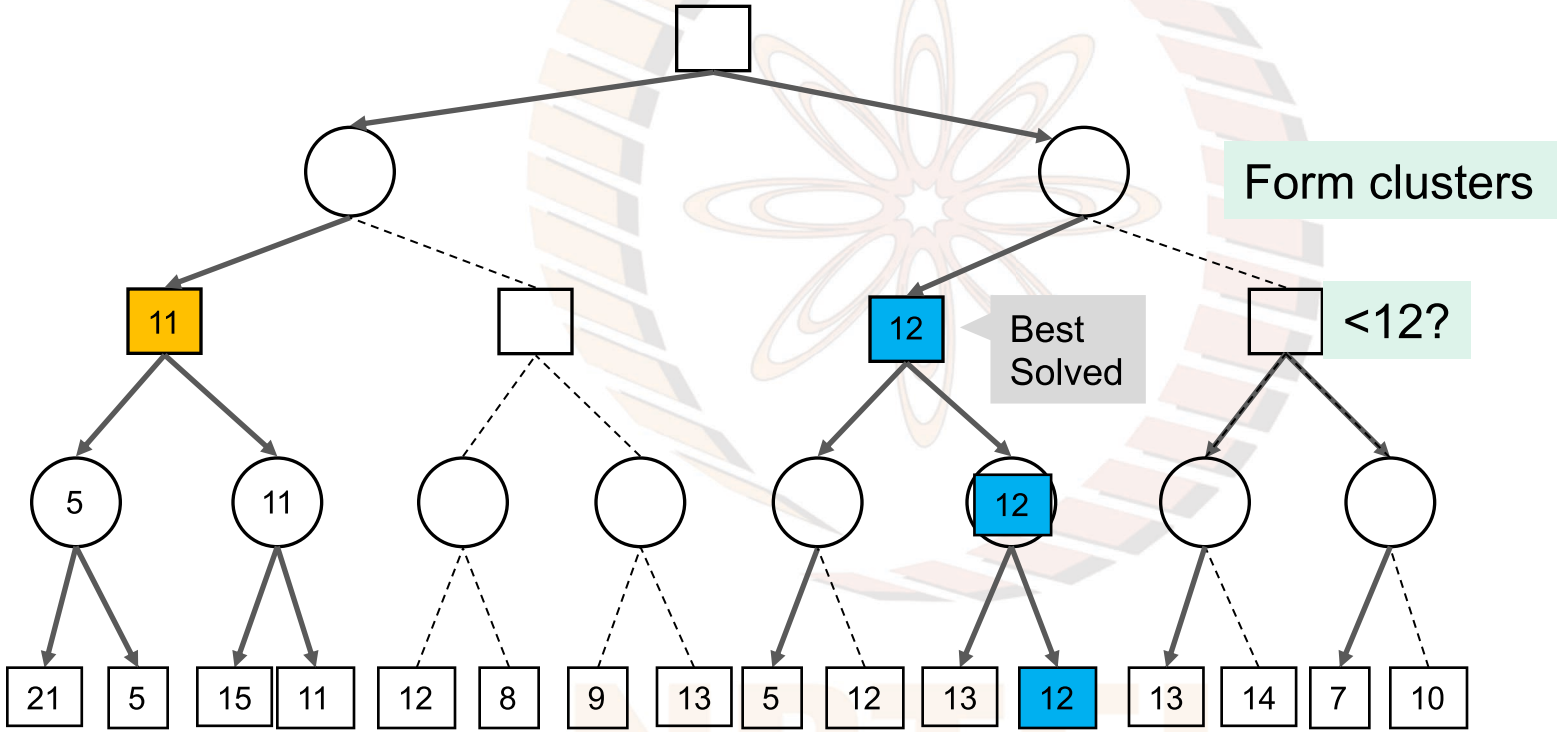
Max is SOLVED → add LIVE sibling

Recursively solve for sibling Max with an upper bound of 12

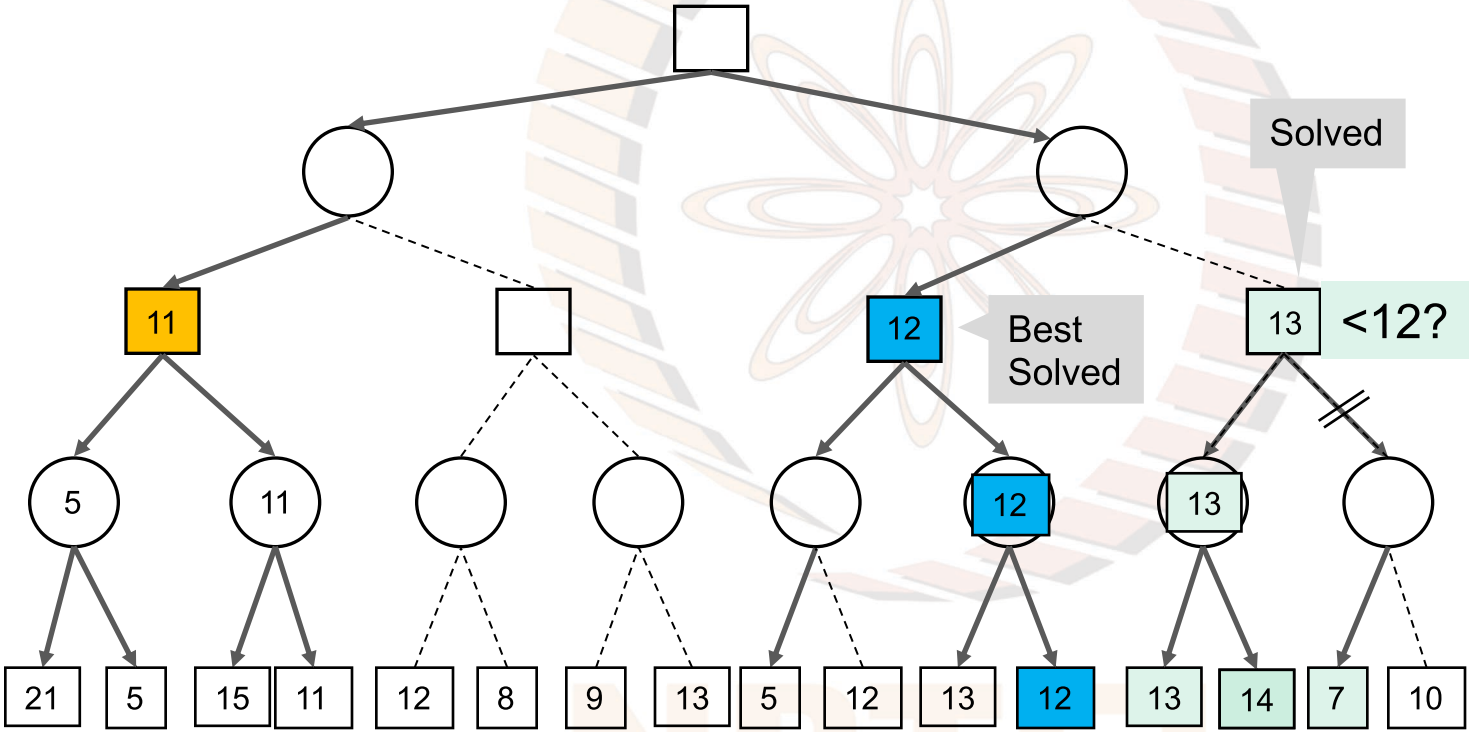
If its value becomes larger that 12 then a Beta cutoff occurs else value of strategy will go down.



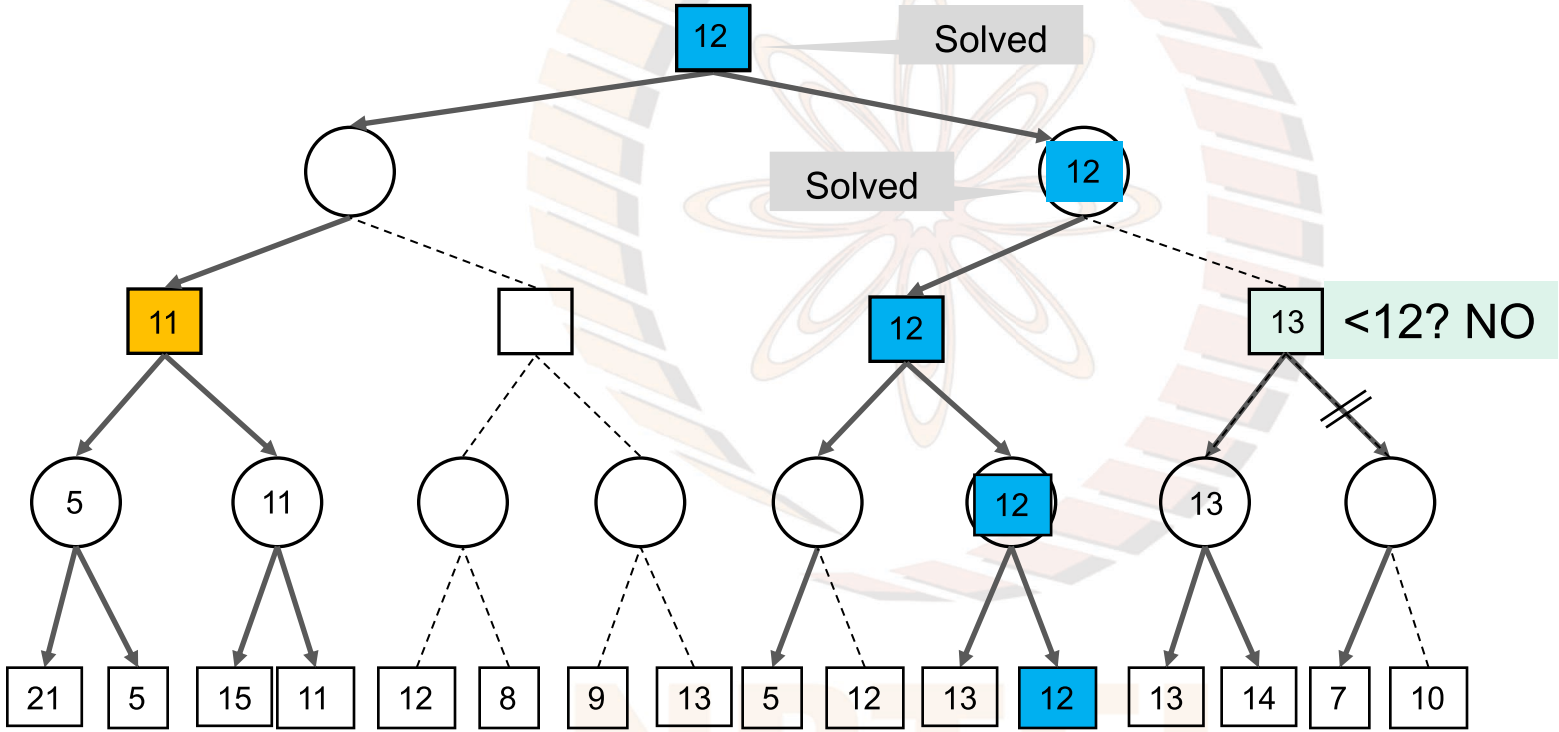
Recursively solve sibling Max node



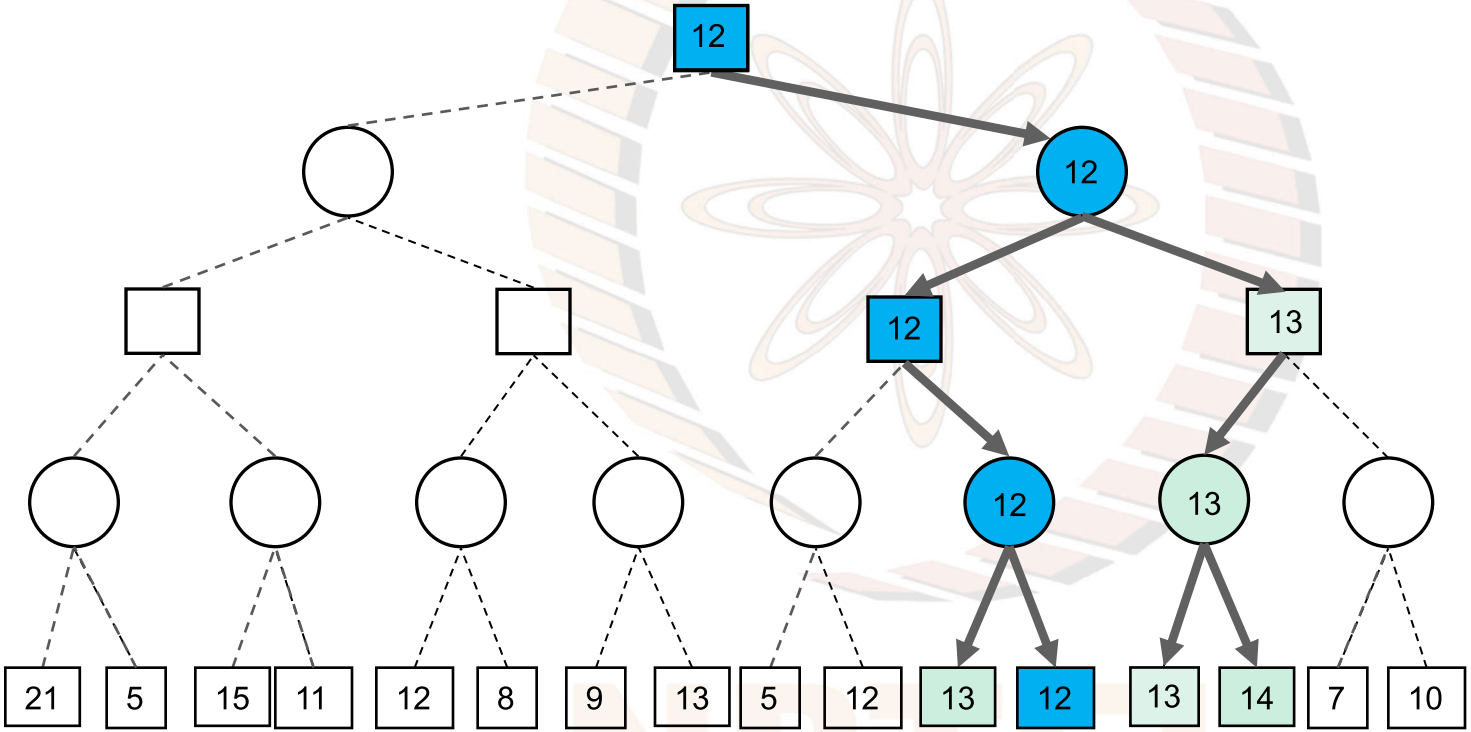
Refine the best partial strategy



When SOLVED *Max* is last child of parent



Strategy returned by SSS*



SSS*: Iterative version

```
SSS*(root)
1   OPEN ← empty priority queue
2   add ( root, LIVE, ∞ ) to OPEN
3   loop
4       ( N, status, h ) ← pop top element from OPEN
5       if N = root and status is SOLVED
6           return h

7       if status is LIVE
8           if N is a terminal node
9               add ( N, SOLVED, min(h, eval(N)) ) to OPEN
10          else if N is a MAX node
11              for each child C of N
12                  add ( C, LIVE, h ) to OPEN
13          else if N is a MIN node
14              add ( first child of N, LIVE, h ) to OPEN

15      if status is SOLVED
16          P ← parent(N)
17          if N is a MAX node and N is the last child
18              add ( P, SOLVED, h ) to OPEN
19          else if N is a MAX node
20              add ( next child of P, LIVE, h ) to OPEN
21          else if N is a MIN node
22              add ( P, SOLVED, h ) to OPEN
23              remove all successors of P from OPEN
```

Start by adding root to the priority queue and continue till root appears SOLVED at the head

Forward phase: add LIVE nodes till the horizon.

Evaluate on the horizon

Backward phase: when SOLVED apply the backup rule

SSS*: Initialize priority queue

```
SSS*(root)
1   OPEN ← empty priority queue
2   add ( root, LIVE, ∞ ) to OPEN
3   loop
4       ( N, status, h ) ← pop top element from OPEN

5       if N = root and status is SOLVED
6           return h
        ...
```

Terminate when the root pops out of the priority queue with *status* = SOLVED.

SSS: popping a LIVE node

```
SSS*(root)
 1  OPEN ← empty priority queue
 2  add ( root, LIVE, ∞ ) to OPEN
 3  loop
 4      ( N, status, h ) ← pop top element from OPEN
        ...
 7      if status is LIVE
 8          if N is a terminal node
 9              add ( N, SOLVED, min(h, eval(N)) ) to OPEN
10          else if N is a MAX node
11              for each child C of N
12                  add ( C, LIVE, h ) to OPEN
13          else if N is a MIN node
14              add ( first child of N, LIVE, h ) to OPEN
        ...
```

For terminal nodes invoke *eval(N)* and choose *minimum* of bound and *eval(N)*

else insert all children of *Max* and one child for *Min*

SSS*: popping a SOLVED node

```
SSS*(root)
 1  OPEN ← empty priority queue
 2  add ( root, LIVE, ∞ ) to OPEN
 3  loop
 4      ( N, status, h ) ← pop top element from OPEN
        ...
15      if status is SOLVED
16          P ← parent(N)
17          if N is a MAX node and N is the last child
18              add ( P, SOLVED, h ) to OPEN
19          else if N is a MAX node
20              add ( next child of P, LIVE, h ) to OPEN
21          else if N is a MIN node
22              add ( P, SOLVED, h ) to OPEN
23              remove all successors of P from OPEN
```

If *Max* node is not the last child then add sibling (looking for a lower value)

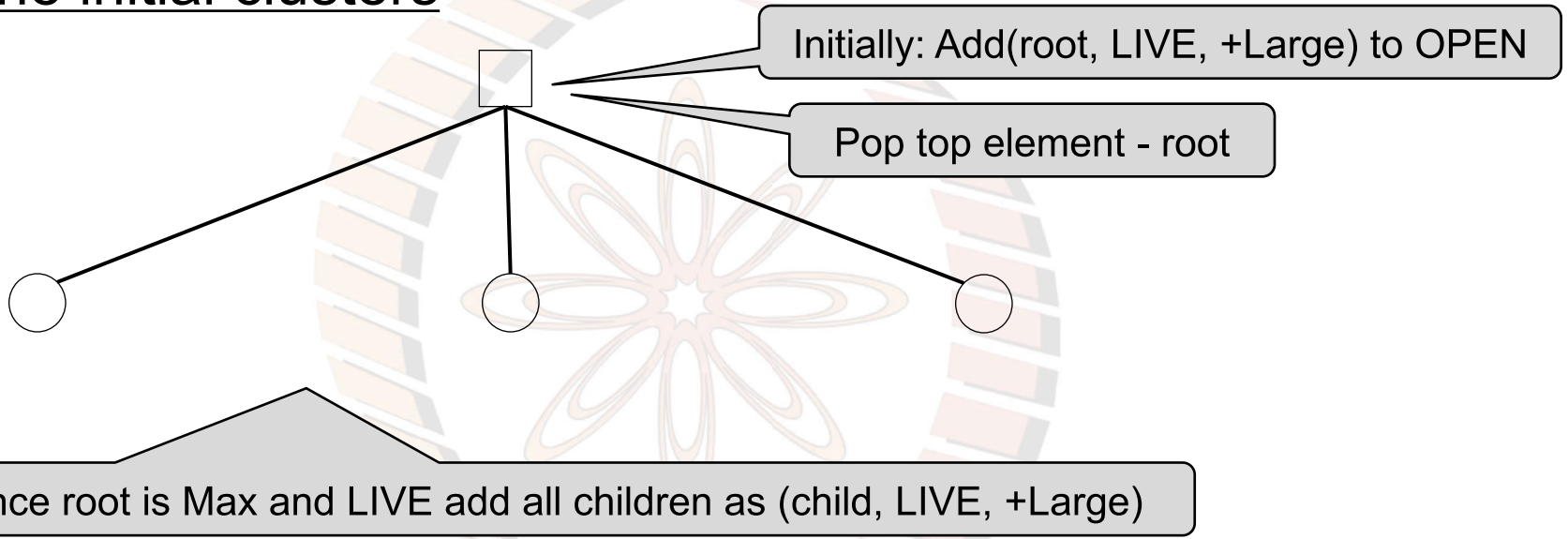
Else for both *Max* and *Min* add the parent as SOLVED

SSS*: A programming note

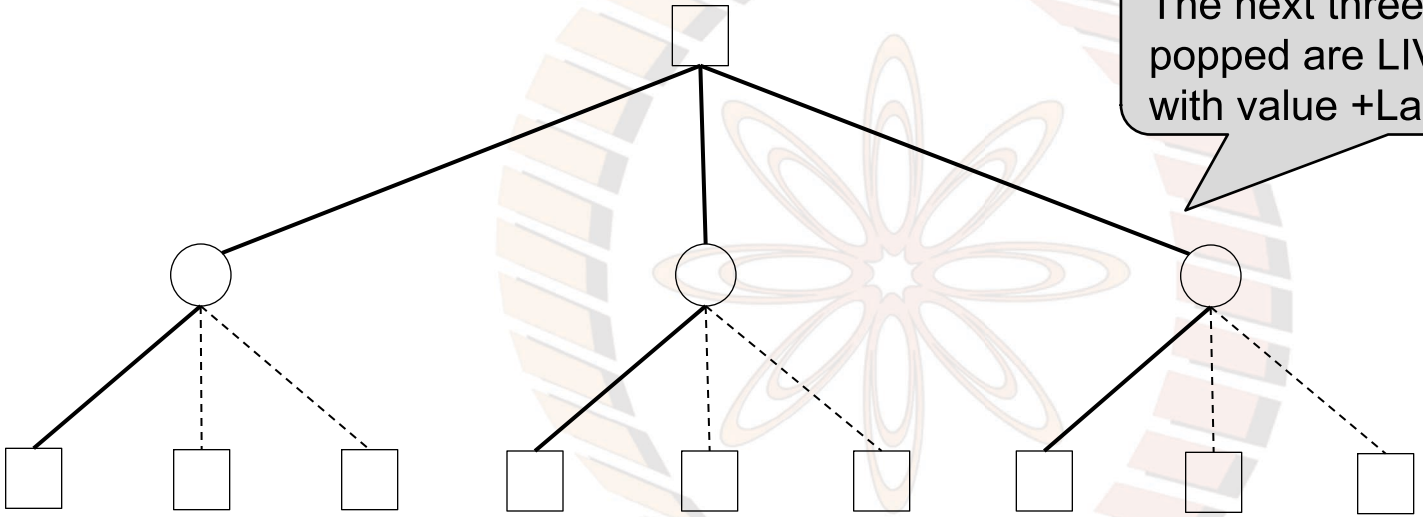
```
SSS*(root)
1  OPEN ← empty priority queue
2  add ( root, MAX, LIVE, ∞ ) to OPEN
3  loop
4      ( N, player, status, h ) ← pop top element from OPEN
5      if N = root and status is SOLVED
6          return h

7      if status is LIVE
8          if N is a terminal node
9              add ( N, player, SOLVED, min(h, eval(N)) ) to OPEN
10         else if player is MAX
11             for each child C of N
12                 add ( C, MIN, LIVE, h ) to OPEN
13         else if player is MIN
14             add ( first child of N, MAX, LIVE, h ) to OPEN

15     if status is SOLVED
16         P ← parent(N)
17         if player is MAX and N is the last child
18             add ( P, MIN, SOLVED, h ) to OPEN
19         else if player is MAX
20             add ( next child of P, MAX, LIVE, h ) to OPEN
21         else if player is MIN
22             add ( P, MAX, SOLVED, h ) to OPEN
23             remove all successors of P from OPEN
```

Add a parameter “player” that can be *Max* or *Min* – easier to check the case

SSS*: the initial clusters



SSS*: the initial clusters



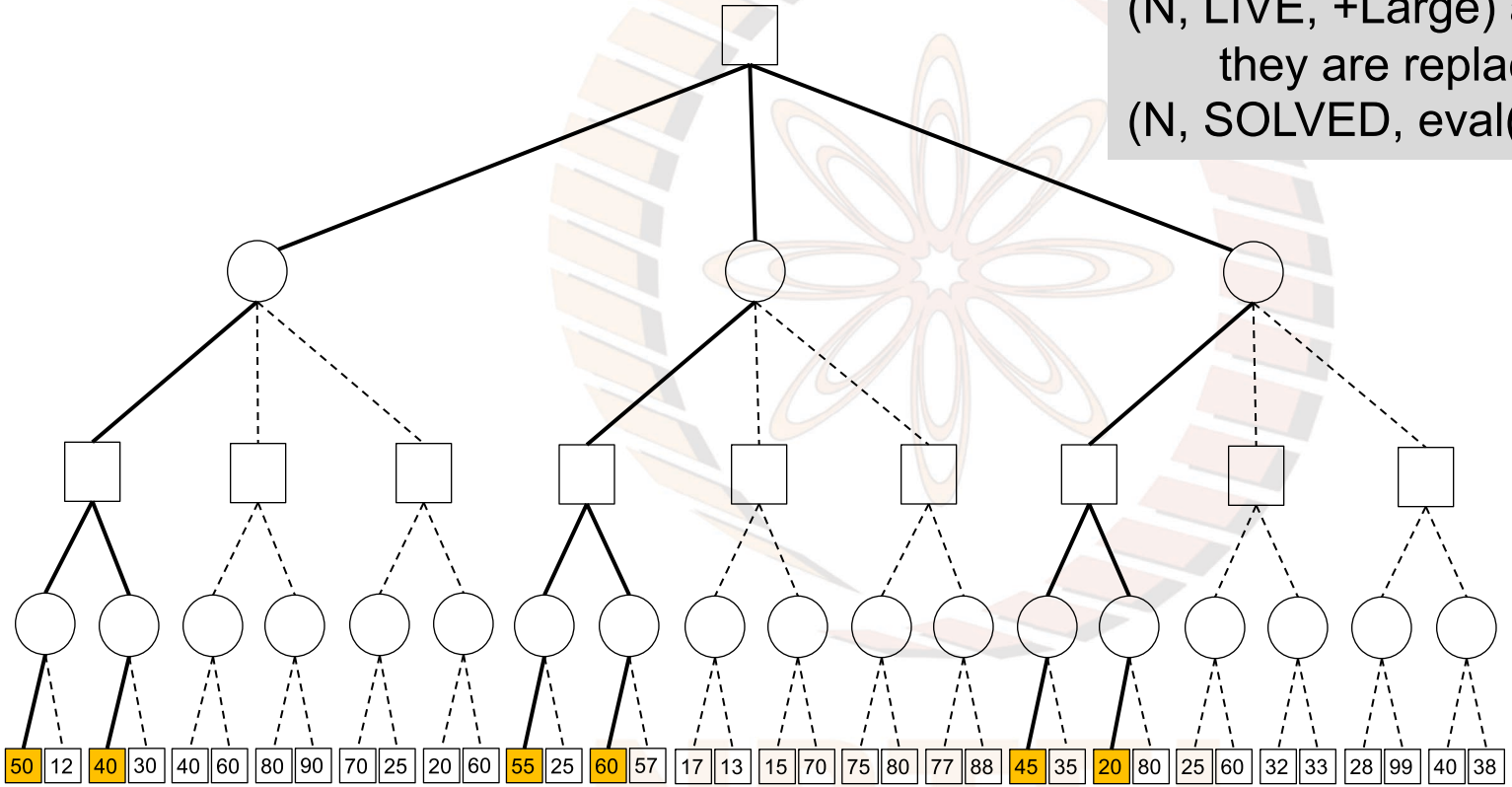
The next three elements to be popped are LIVE Min nodes with value +Large

For each of the Min nodes add *one* child (child, LIVE, +Large)

Continue till nodes on horizon are added

SSS*: the initial clusters

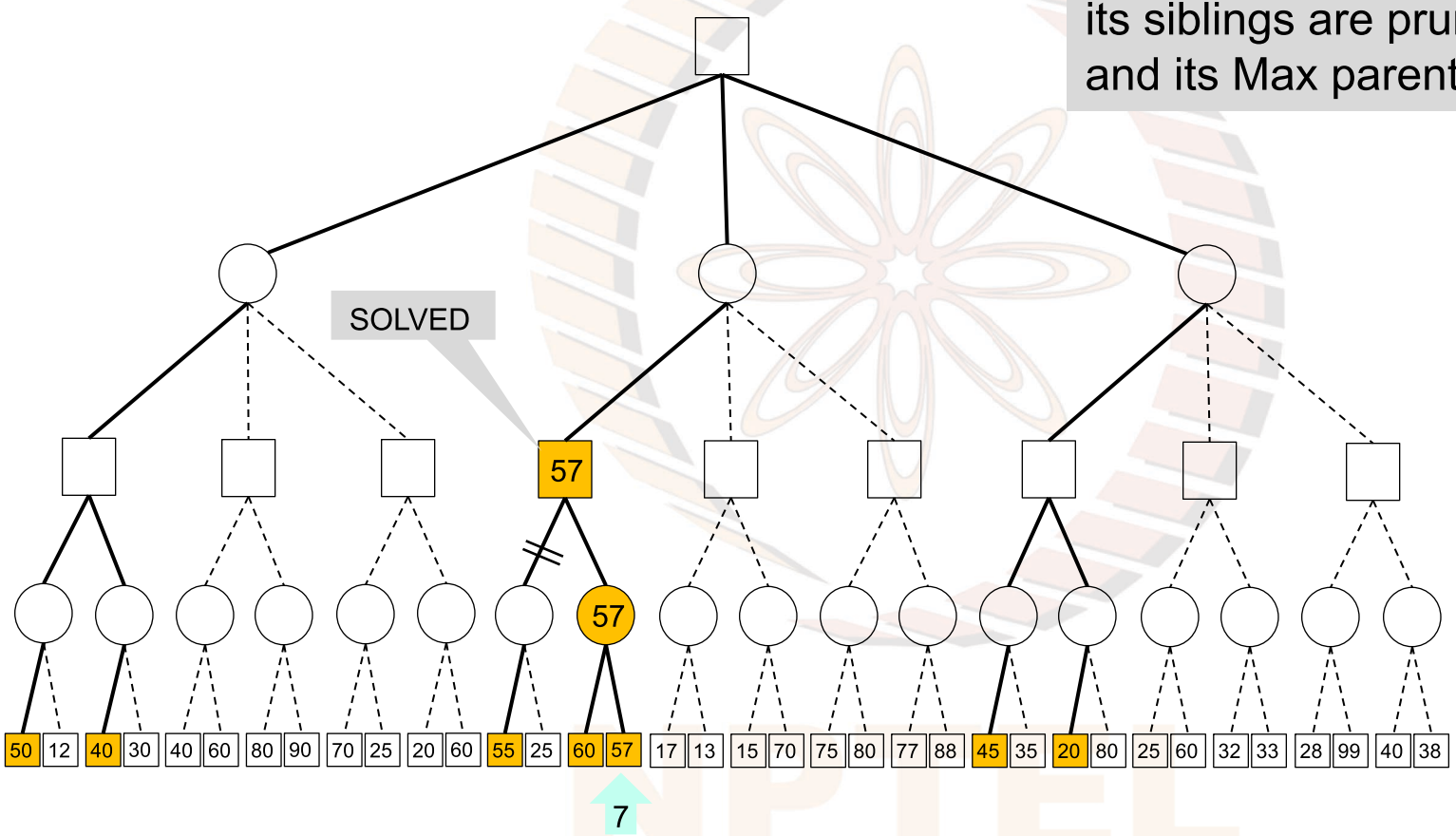
When terminal nodes of the form
(N, LIVE, +Large) are popped
they are replaced by
(N, SOLVED, eval(N))



Refine best cluster

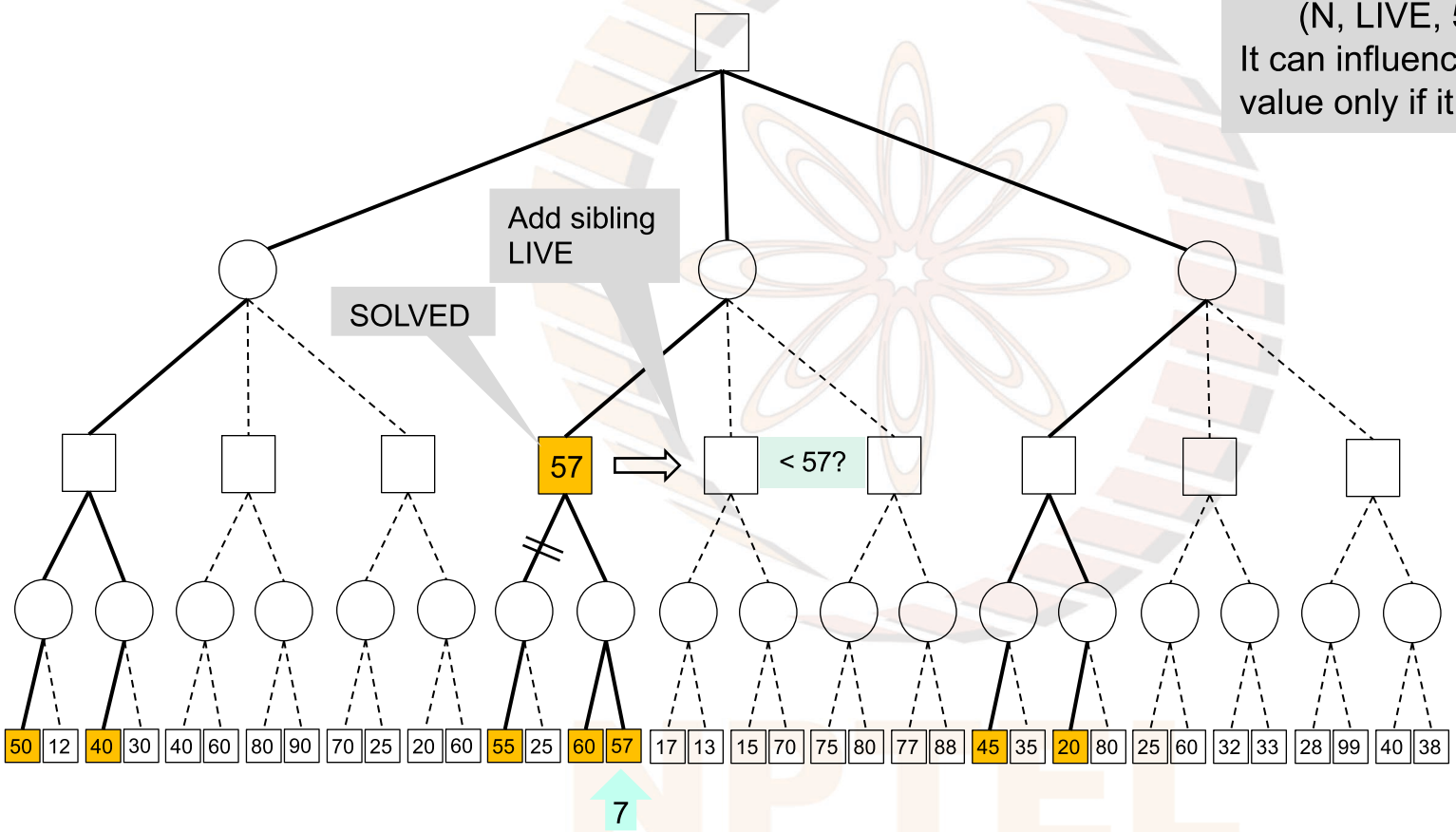
SSS*: refine best cluster

When the Min parent is SOLVED
its siblings are pruned
and its Max parent is SOLVED too



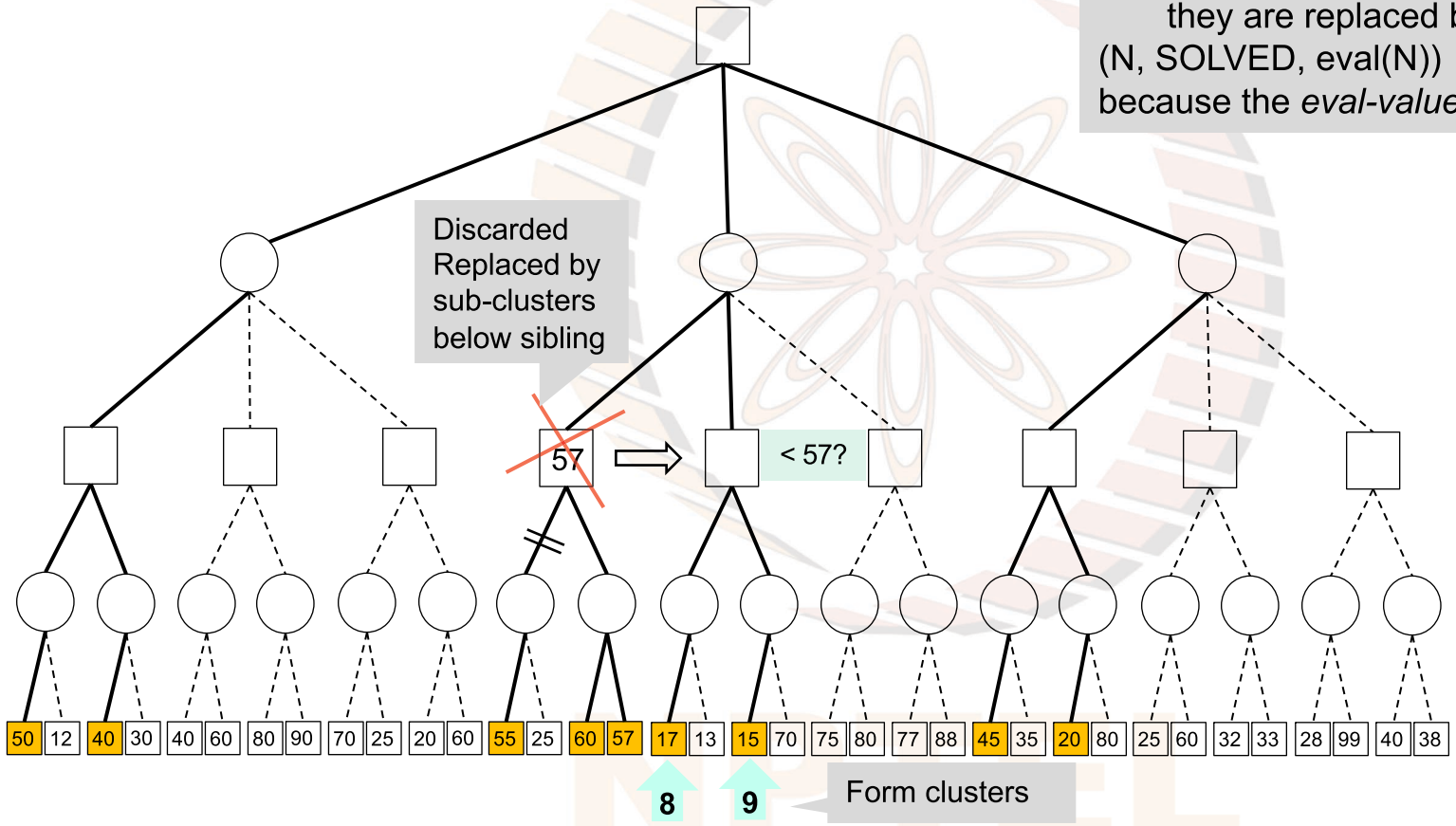
SSS*: Max SOLVED → Add sibling

The sibling is added as
(N, LIVE, 57)
It can influence the minimax value only if it is less than 57

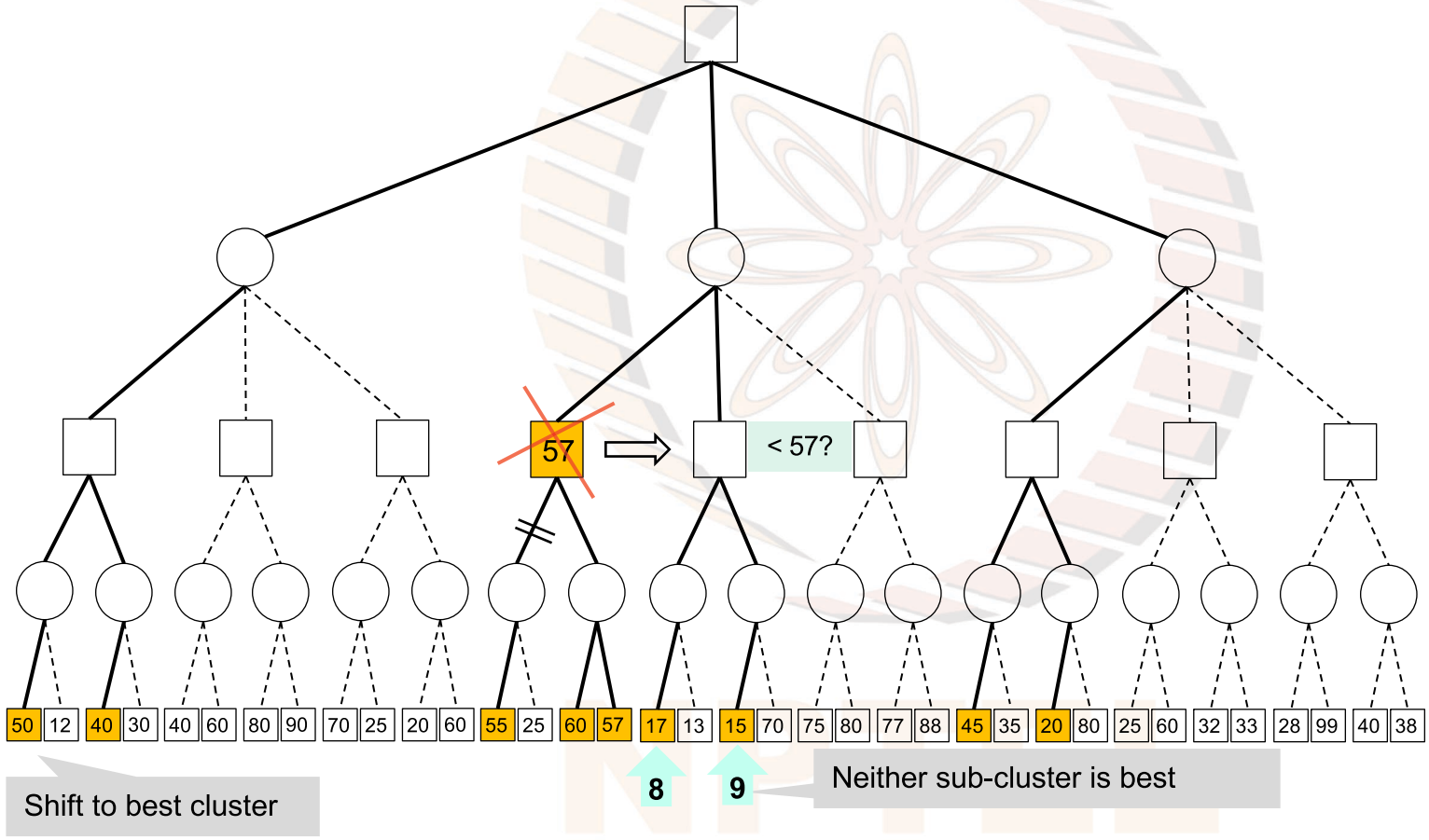


SSS*: recursively form clusters

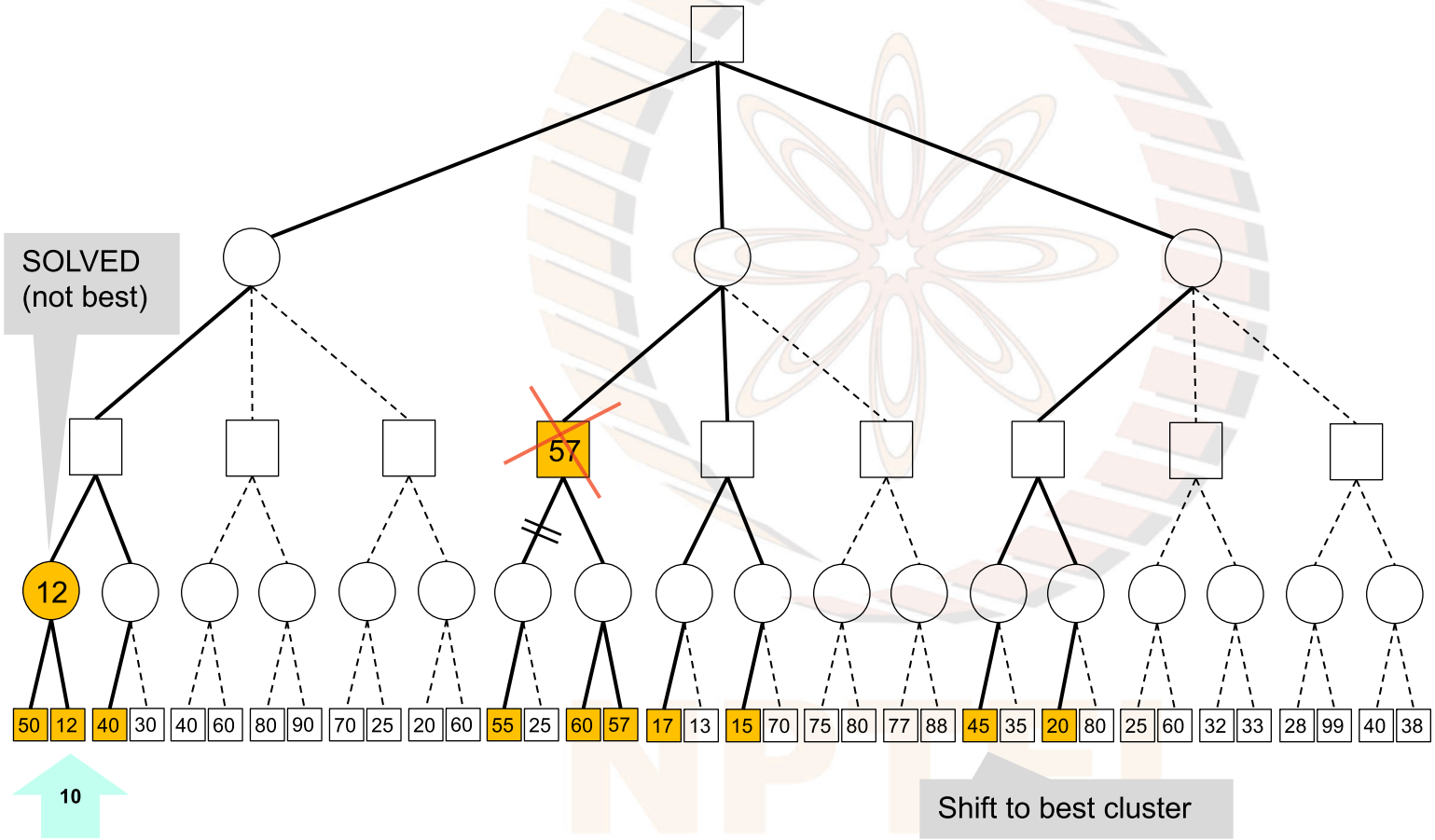
When terminal nodes of the form
(N, LIVE, +57) are popped
they are replaced by
(N, SOLVED, eval(N))
because the *eval-values are smaller*



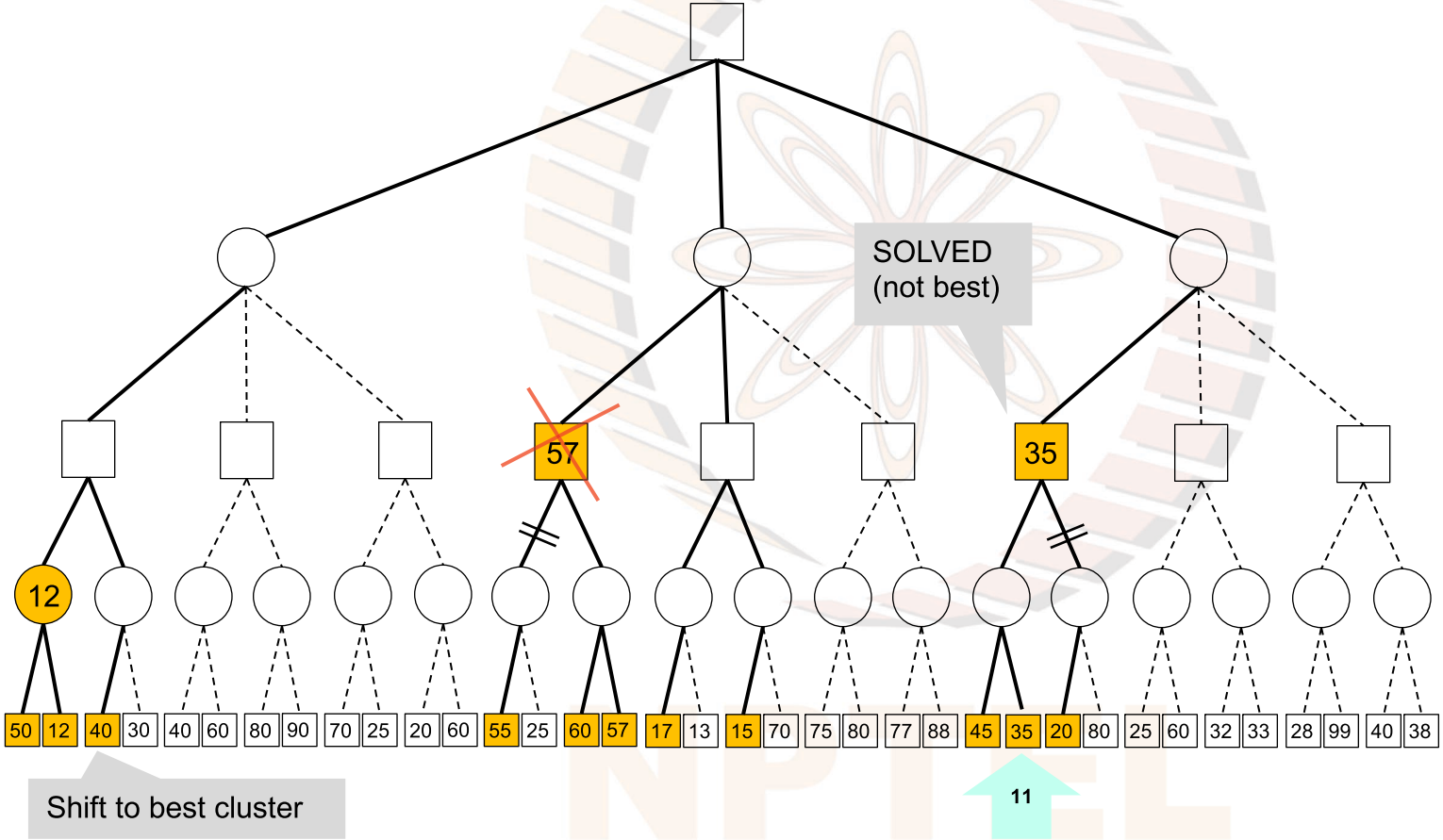
SSS*: shift if clusters not promising



SSS*: refine best cluster

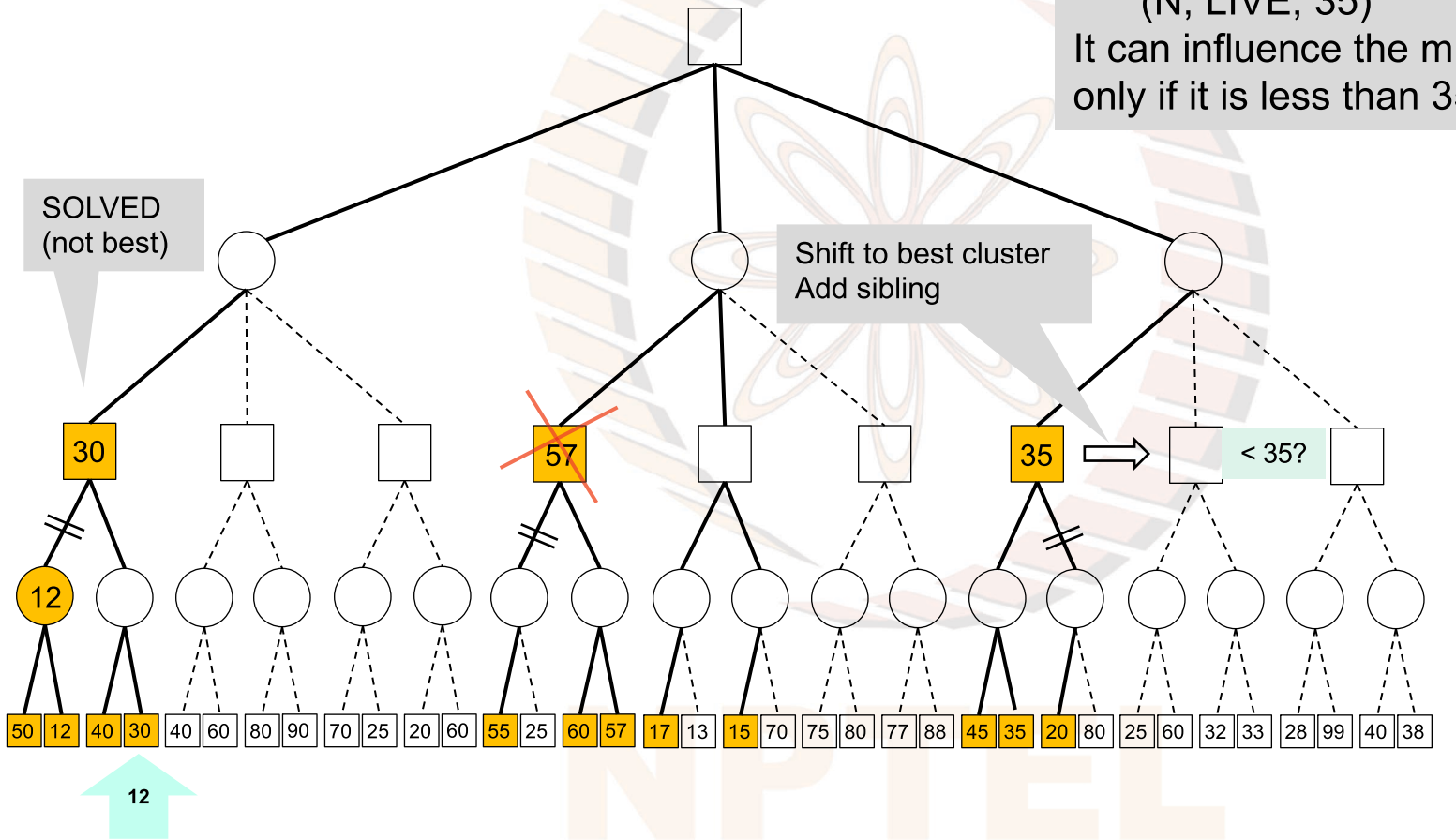


SSS*: refine best cluster



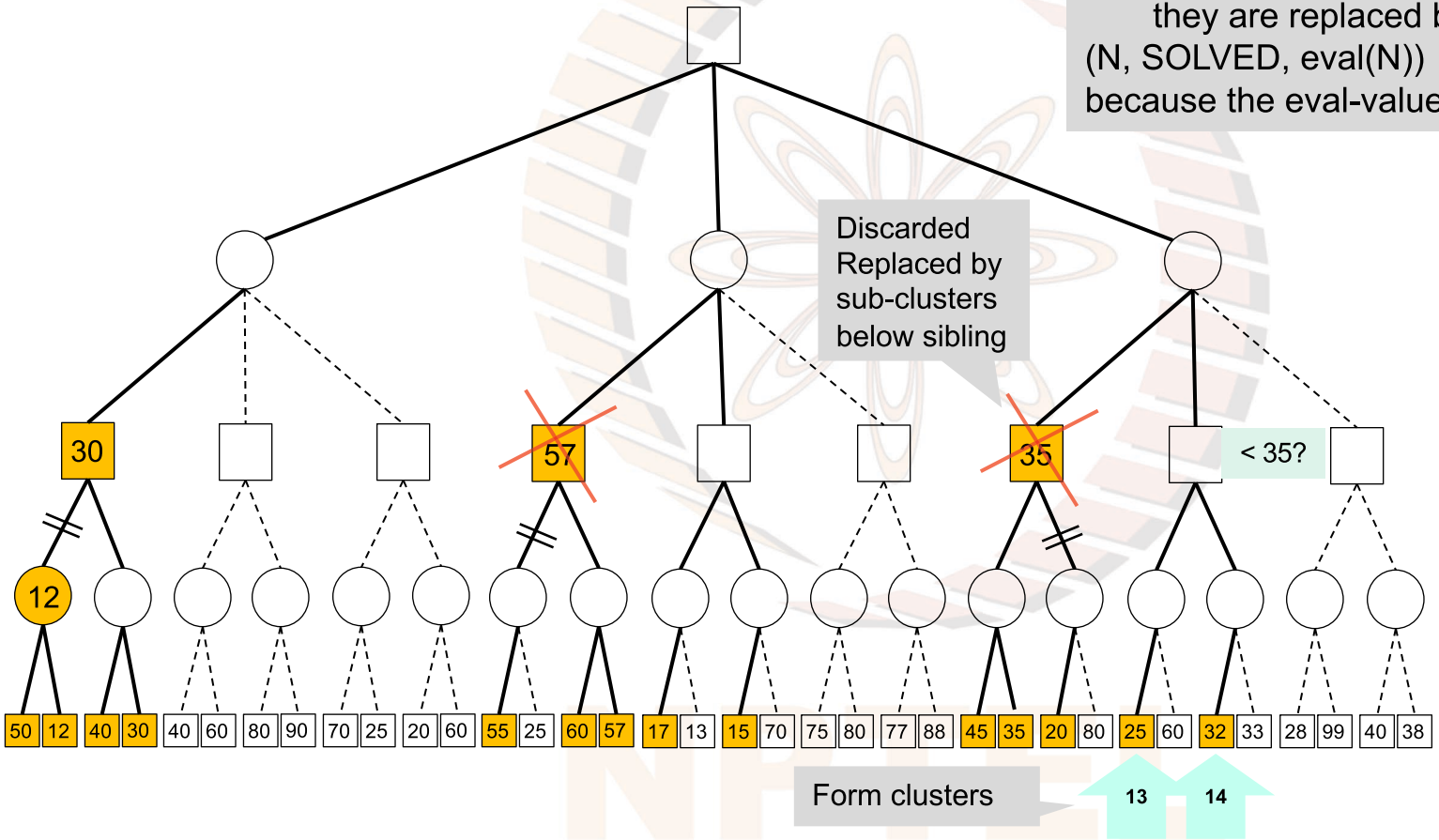
SSS*: Max SOLVED → add sibling

The sibling is added as
(N, LIVE, 35)
It can influence the minimax value
only if it is less than 35



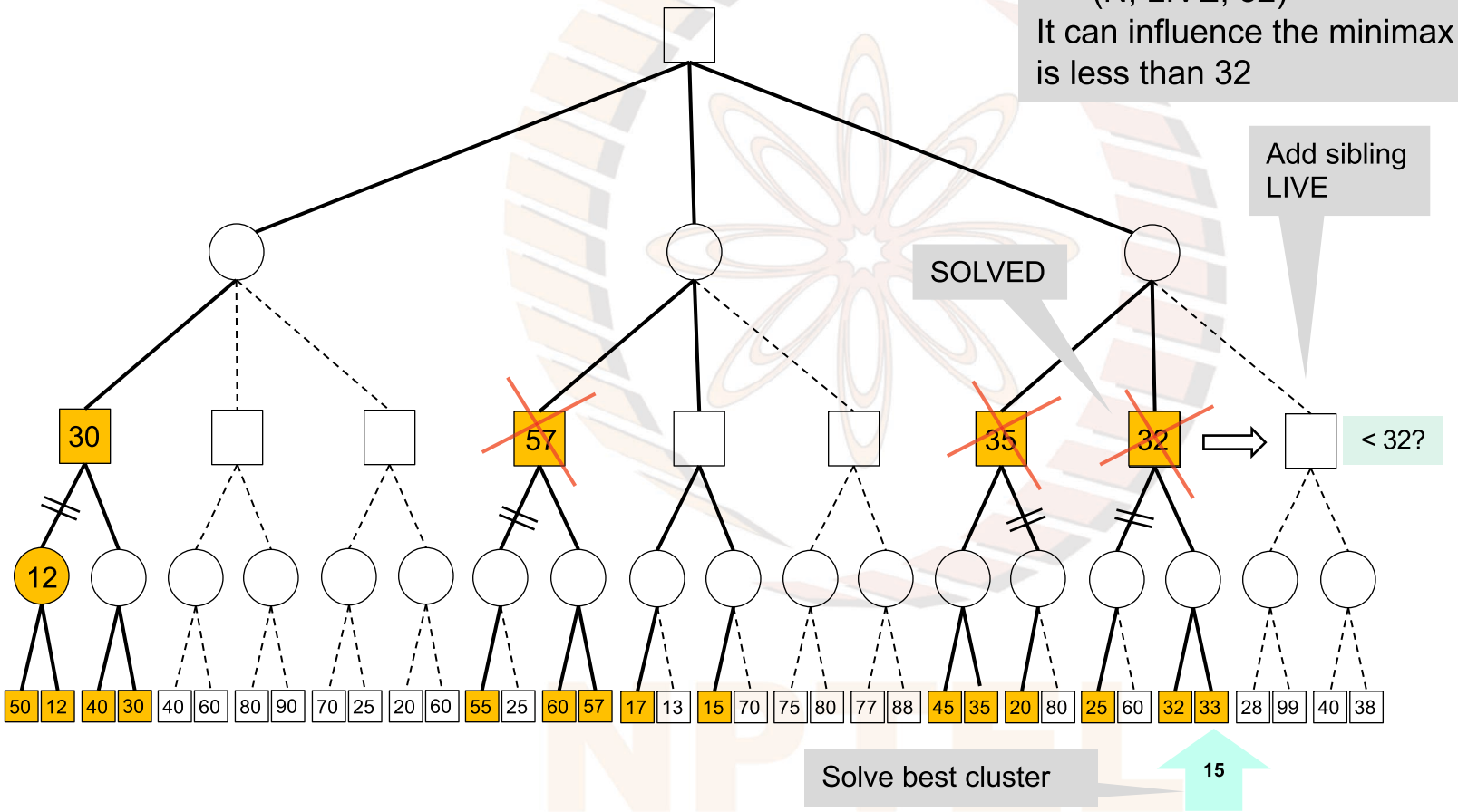
SSS*: recursively form clusters

When terminal nodes of the form
(N, LIVE, +35) are popped
they are replaced by
(N, SOLVED, eval(N))
because the eval-values are smaller



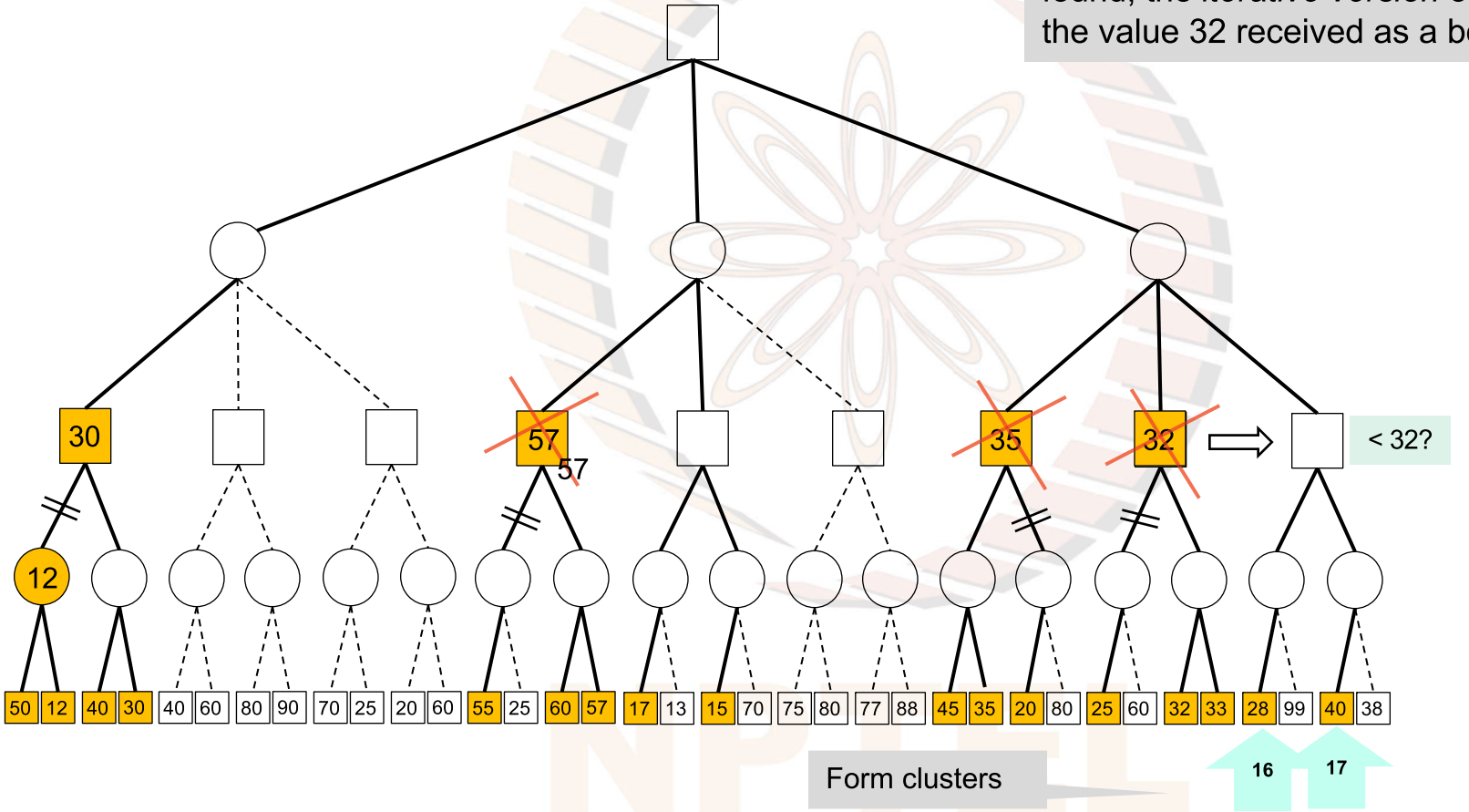
SSS*: add another sibling

The sibling is added as
(N, LIVE, 32)
It can influence the minimax value only if it is less than 32



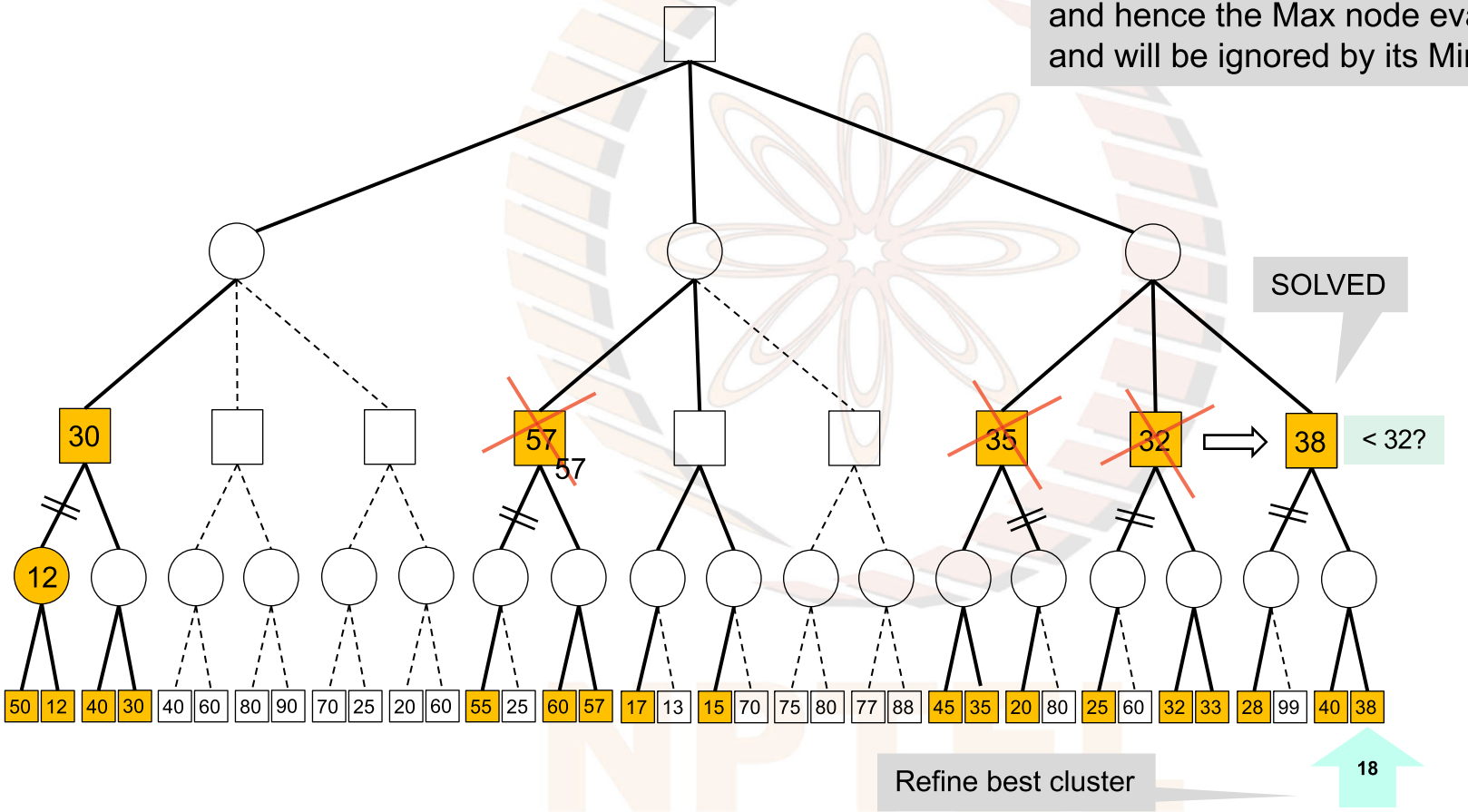
SSS*: form clusters

When the terminal node with value 40 is found, the *iterative version* of SSS* keeps the value 32 received as a bound.

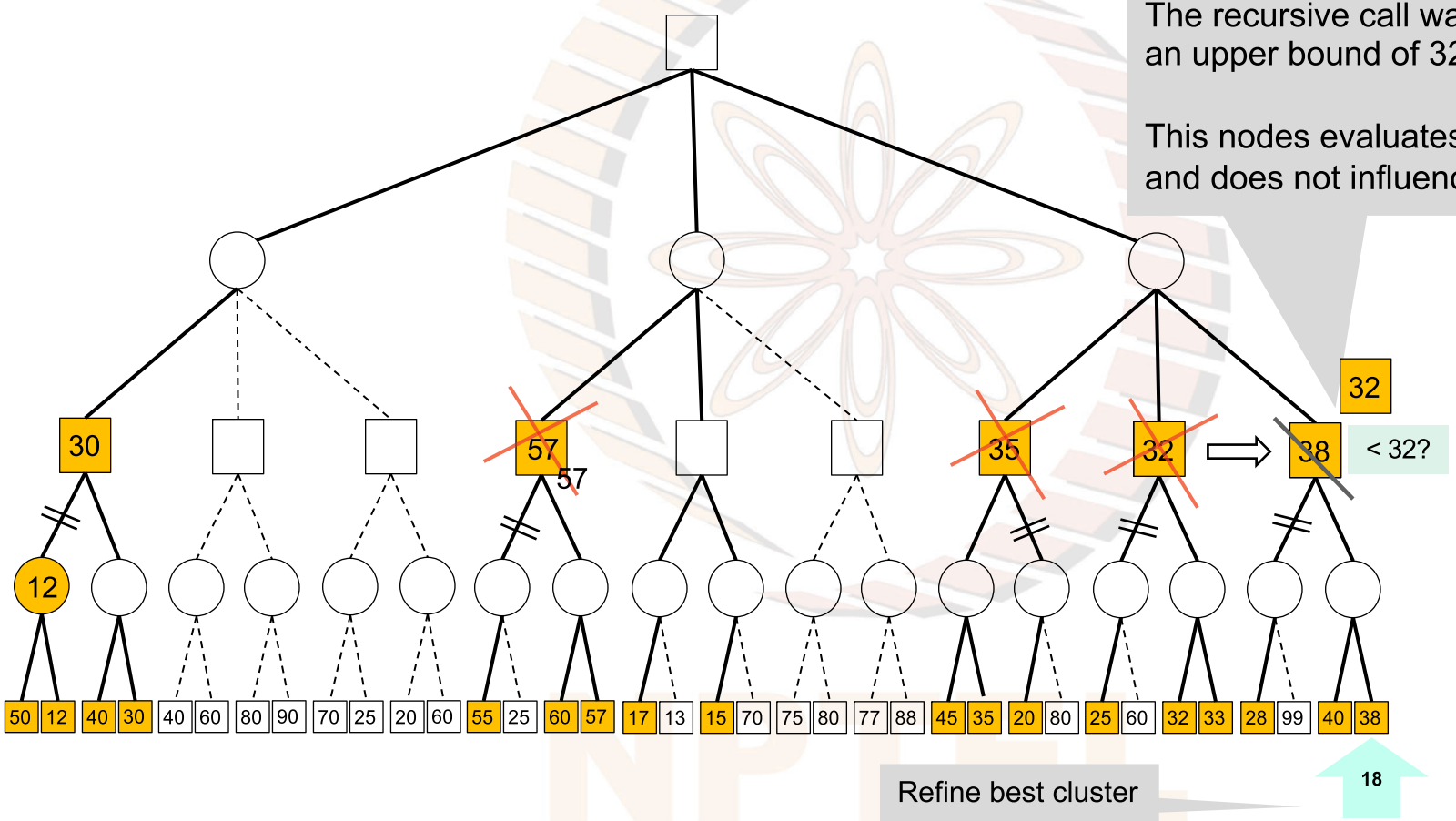


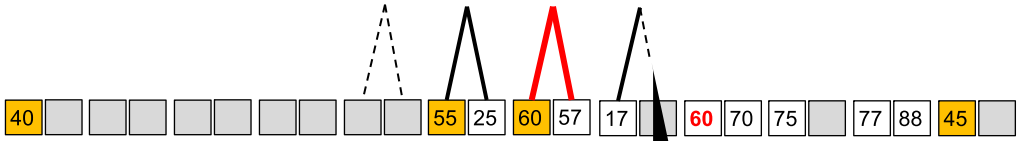
SSS*: the initial clusters

In this “recursive” illustration we retain 40 and then 38,
and hence the Max node evaluates to 38, and will be ignored by its Min parent

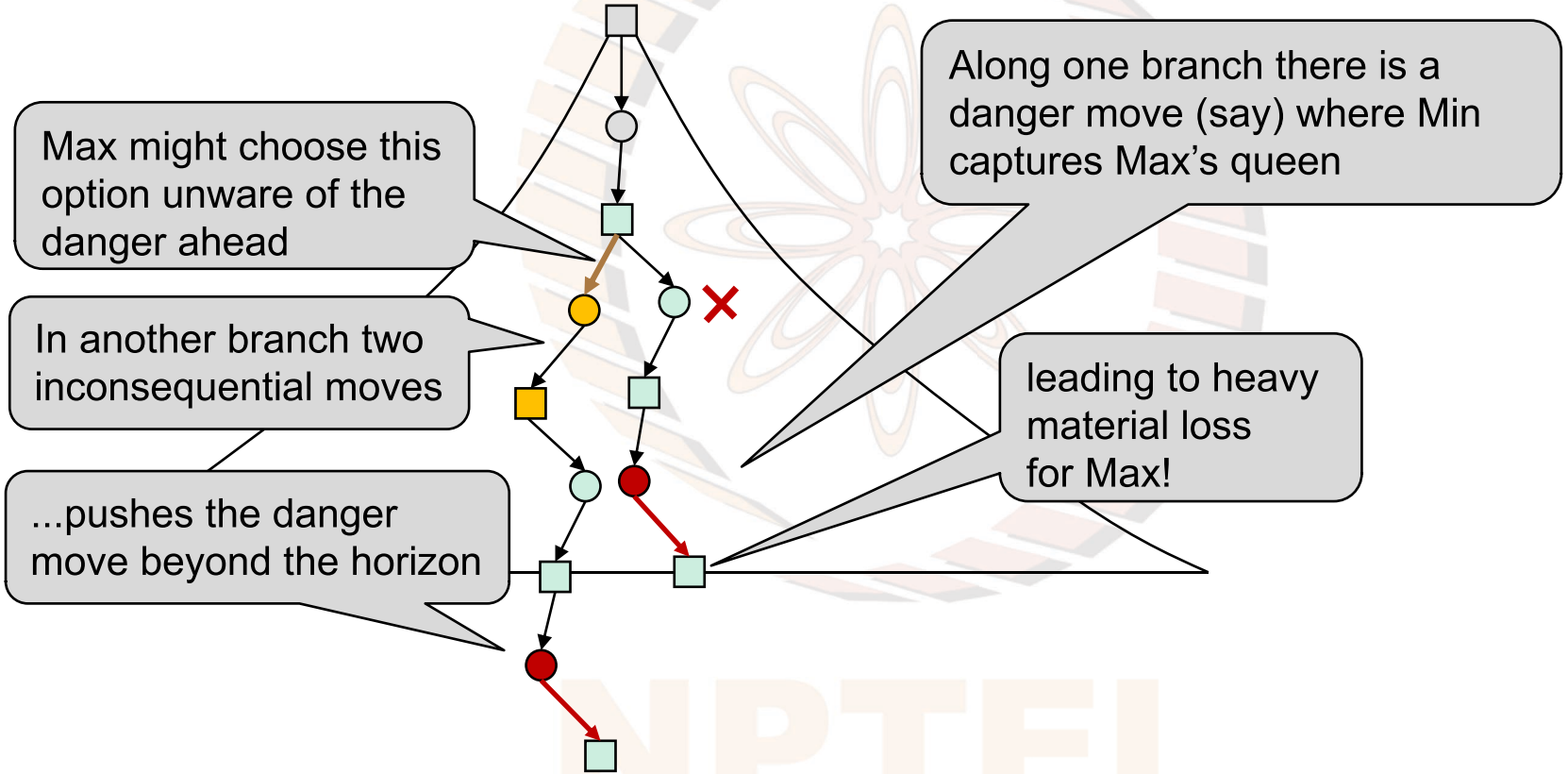


SSS*: ignore if higher than beta bound

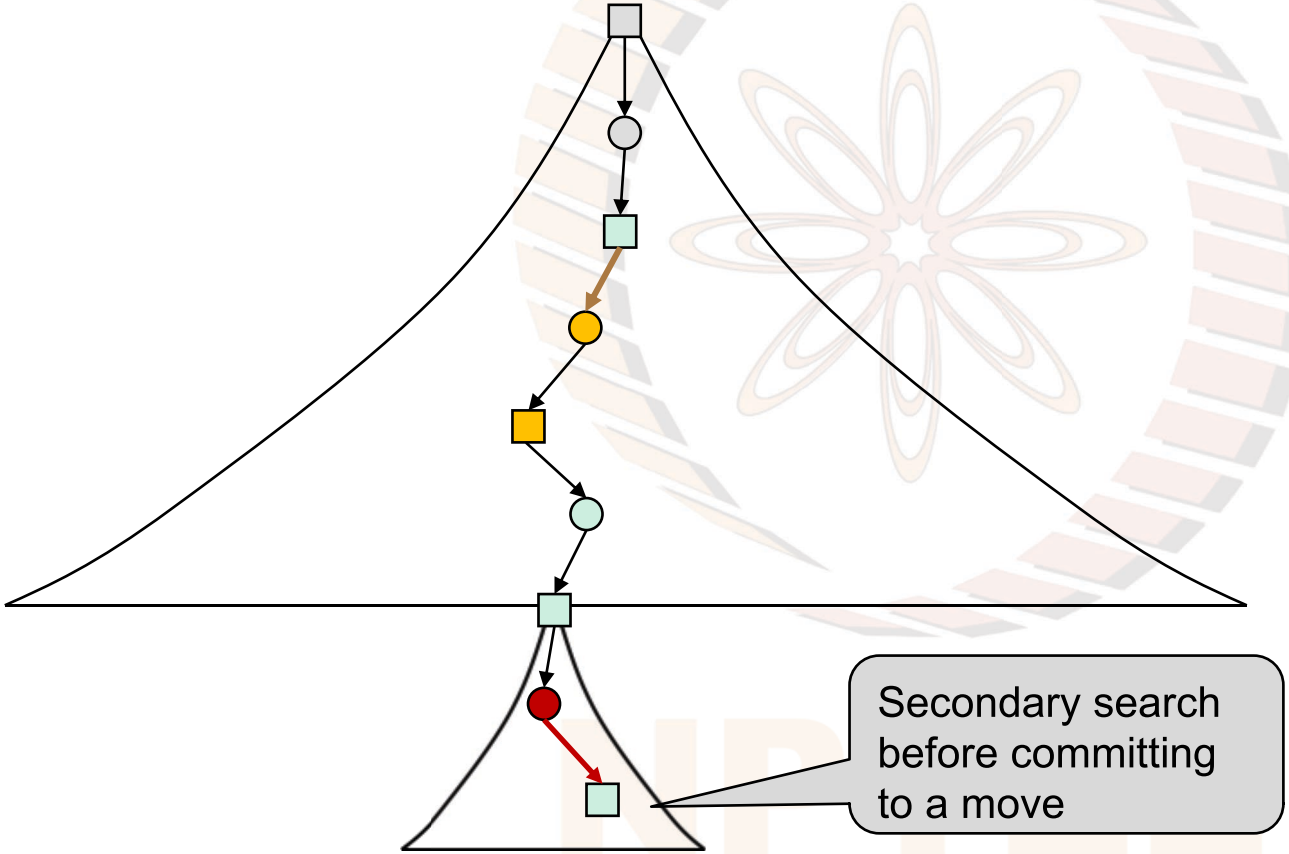



40
55
25
60
57
17
60
70
75
77
88
45

Finite lookahead: The Horizon Effect



Finite lookahead: Secondary Search



End: Game Playing