Board Representation

57	58	59	60	61	62	63	64
49	50	51	52	53	54	55	56
41	42	43	44	45	46	47	48
33	34	35	36	37	38	39	40
25	26	27	28	29	30	31	32
17	18	19	20	21	22	23	24
9	10	11	12	13	14	15	16
1	2	3	4	5	6	7	8

8 x 8 Matrix

- Problem: doubles subscript computation

Array (64)

- Subscript range checking: every move generated must be checked
- Illegal moves easily generated: example King on square 8, adding 1 generates move to square 9

Board Representation: Array 120

111	112	113	114	115	116	117	118	119	120
7	7	7	7	7	7	7	7	7	7
7	7	7	7	7	7	7	7	7	7
7	-4	-2	-3	-6	-5	-3	-2	-4	7
7	-1	-1	-1	-1	-1	-1	-1	-1	7
7	0	0	0	0	0	0	0	0	7
7	0	0	0	0	0	0	0	0	7
7	0	0	0	0	0	0	0	0	7
7	0	0	0	0	0	0	0	0	7
7	1	1	1	1	1	1	1	1	7
7	4	2	3	6	5	3	2	4	7
7	7	7	7	7	7	7	7	7	7
7	7	7	7	7	7	7	7	7	7
1	2	3	4	5	6	7	8	9	10

Illegal Move Detection

Assume Machine plays White:
CASE destination square value

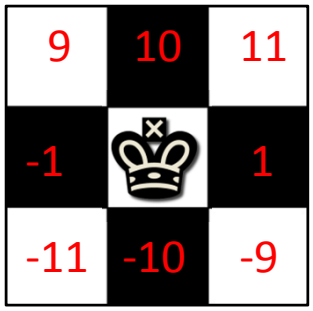
7	: Off Board Move
1 . . . 6	: Square occupied by Machine‘s piece
0	: open square
< 0	: capturing move

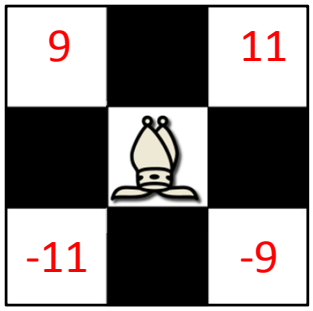
Move Generation

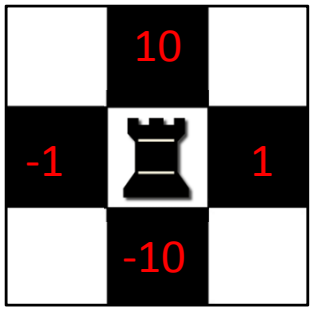
- Routine required for each type of piece
- Piece location passed as parameter

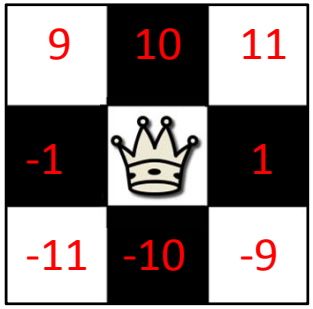
Board Representation: Array 120

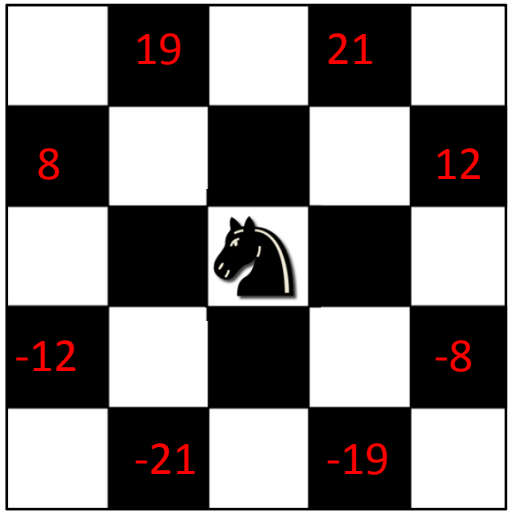
Move Generation

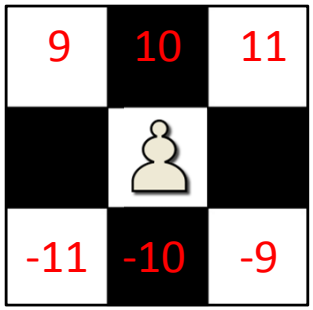












Kings, Knights, pawns move number applied only once.

Queens, Bishops, Rooks move number applied repeatedly until off board square or occupied square generated.

Bit-Board Representation

- 64 Bits, 1 per square, used to store pieces' locations
- Bit value:
  - 1 = represents a piece
  - 0 = empty square
- 12 x 64 bits to store a position
  - 6 white pieces
  - 6 black pieces
  - 64 bits for each different type of piece:
    King, Queen, Bishop, Knight, Rook, Pawn
- 2 extra words are used to store all piece locations for both sides for efficient move generation

Bit-Board Move Generation

- For each type of piece all possible moves for every square must be stored: 6 x 64 x 64 = 3K bytes
- Move bit-maps are searched for the piece to be moved & the square piece is on to locate piece move bit-map
- Logical operations are performed to generate legal bit-map moves:

& (piece move bit-map,~(bit-map ∀ pieces of moving side))

- Generates bit-map for all legal piece moves in a given position

Bit-Map Move Generation

Bit-Map ∀ possible Knight moves from E3 (21)

0001010000100010000000000010001000010100 ≡
12345678901234567890123456789012345678901 ≡
1 2 3 4

Bit-Map for White Pieces

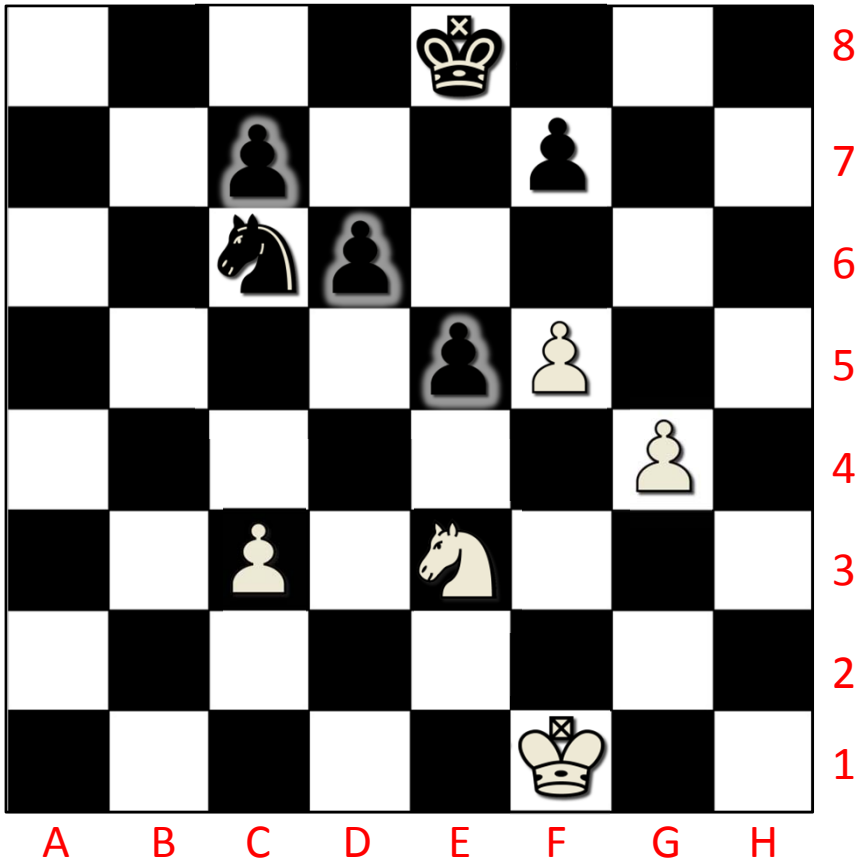
0000010000000000001010000000001000000100 ≡
12345678901234567890123456789012345678901 ≡
1 2 3 4

NOT (Bit-Map for White Pieces)

1111101111111111111010111111111110111111011 ≡
12345678901234567890123456789012345678901 ≡
1 2 3 4

AND(Knight moves , Not (White Pieces))

0001000000100010000000000010000000010000 ≡
12345678901234567890123456789012345678901 ≡
1 2 3 4



Evaluation Function

Numerical value assigned to a given position
Programs next move decision is based *entirely* on Fn
Tradeoff:

Complex Fn	more computing time less search time
Simple Fn	less computing time more search time

Material Evaluation

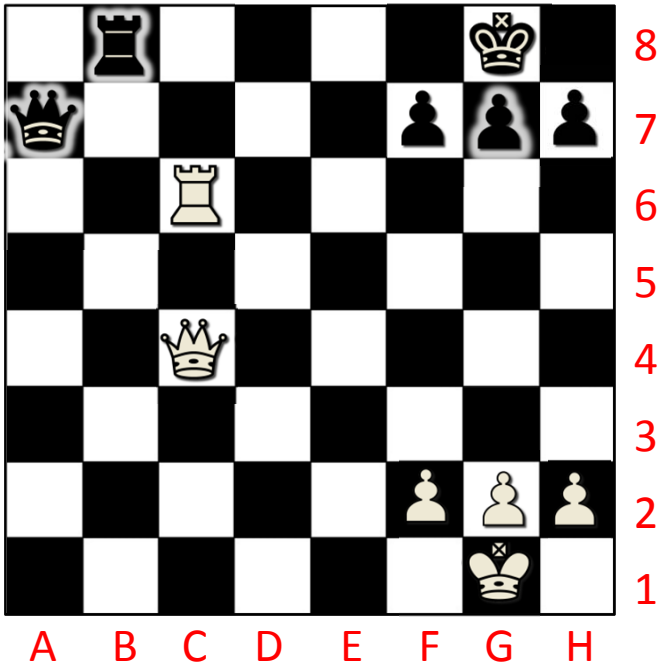
Values:	Pawn	Knight	Bishop	Rook	Queen	King
	100	300	300	500	900	∞

//all computation performed with integer operations

```
FUNCTION Chess ( P , N , B , R , Q ) : INTEGER ;
    Chess =  P * 100 + N * 300 + B * 300 + R * 500 + Q * 900
                    where:
                            P = # of m/c's pawns - # of opponent's pawns
                            N = # of m/c's knights - # of opponent's knights
```

Evaluation Function

Positional Factors (game heuristics)



Mobility
- bonus for each square attacked by a piece

Pawn Structure
- bonus for passed pawns, central pawns
- penalty for doubled pawns, isolated pawns

Attacks
- bonus for attacks against opponent‘s pieces

Castling
- bonus for completion

Development
- penalty for unmoved pieces

Doubled pieces
- bonus for pieces attacking/defending same square

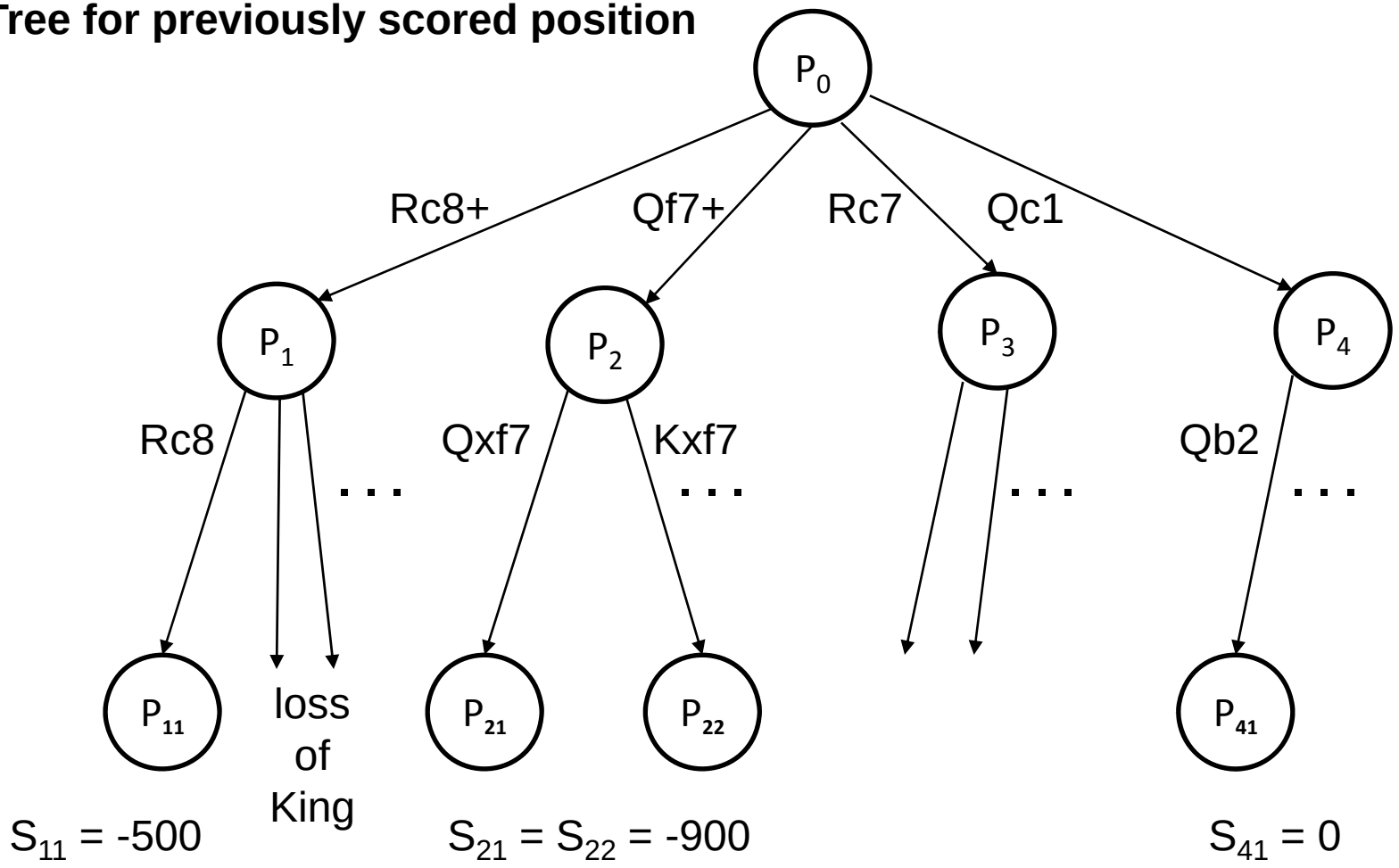
King Safety
- penalty for enemy attacks
- bonus for pieces near King (piece tropism)

Position Score = 0

Symmetrical Positions = 0

Game Tree

Tree for previously scored position



Game Tree

Can a computer play the perfect game of chess?

Can the entire game tree be formed & searched?

Chess 50 move Rule:

If 50 moves occur without a capture or pawn move the game is a draw.

[*IJ Good 1969*] (Va Tech Emeritus University Distinguished Professor)

- Maximum length of a game is 3150 moves.
- Maximum number of possible moves in a game = 320
- Theoretical maximum number of possible chess games =

$$320^{6300} \cong 10^{15790}$$

Game Tree Size

Average middle game position:

- Approximately 30 legal moves
- Tree of depth 4 (2 moves)
  - 10^6 leaves (positions)
  - 10^4 positions for move

Assume

- Scoring = 10 μsecs & move generation = 1μsec
- A move for tree of depth 4 would take 40 secs
- A tree of 6 ply (3 moves) would take 11 hours

Game Tree Size

Tree Size Control Methods: Width

- *Width Control*: number of moves considered per position
- Fanout Parameters
  - Limits the number of moves considered at each level.
  - Decreased with depth 0 [25,30] . . . depth 10 [5,10]
- Plausibility Ordering (iterative deepening)

  plausibility score: heuristic value that determines the worth of a move being searched.
- assigns a *plausibility* score to each generated move
- moves ordered by the *plausibility* score for application of the depth fanout parameters

Game Tree Size

Tree Size Control Methods: Depth

- *Depth Control*: number of moves a line (tree path) is searched
- Fixed Depth → Horizon Effect Problem
  - Key move is missed at the next immediate depth.
  - Example: previous position & 2-ply tree program would choose the highest score = S_{41} (Qc1) missing the 2 move mate.
- Quiescence
  - Position must be static (quiescent: no captures, checks, ≡) before analysis can be stopped & score function applied.
- Depth Cutoff Lists
  - Lists containing positional features to determine quiescence.
  - Number of features decreases with depth.
  - When applied to a position if no feature is found position is considered static, otherwise it is explored further.

Game Theory: Min Max Method

Technique of selecting the best move in a tree

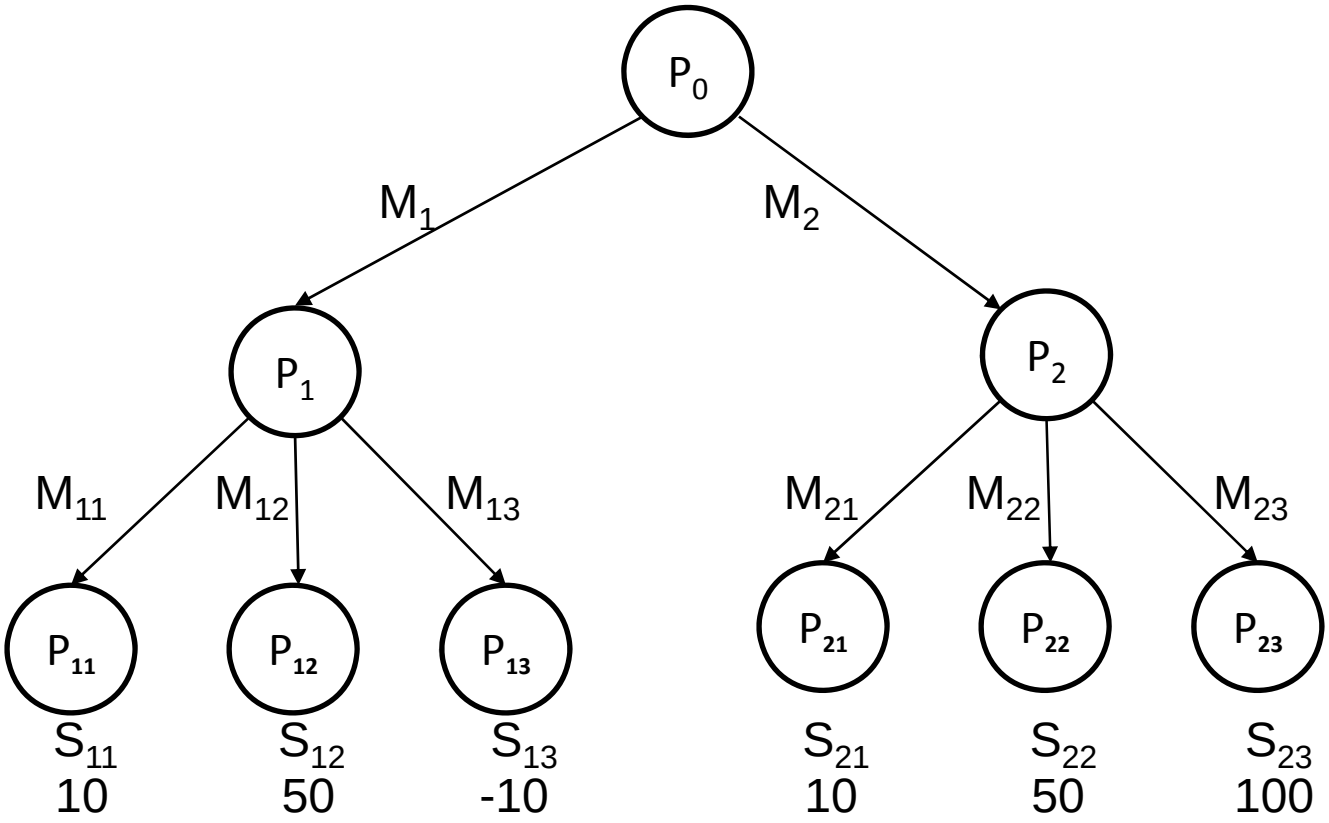
Assumes best play by opponent

- High score is best possible move (position) for program.
- Low score is best possible move for opponent.

Evaluation Fn is applied to static (quiescent) positions

- Minimum or Maximum (depending upon player's/opponent's move) of scores for lower level nodes are selected assigned to the higher level nodes.
- Process repeats until root (current position) is reached.
- The move that is selected is the one that leads to the backed-up value for the root.

Game Theory: Min Max Method



$S_1 = \text{Min}(S_{11} , S_{12} , S_{13}) = S_{13} = -10 = M_{13}$

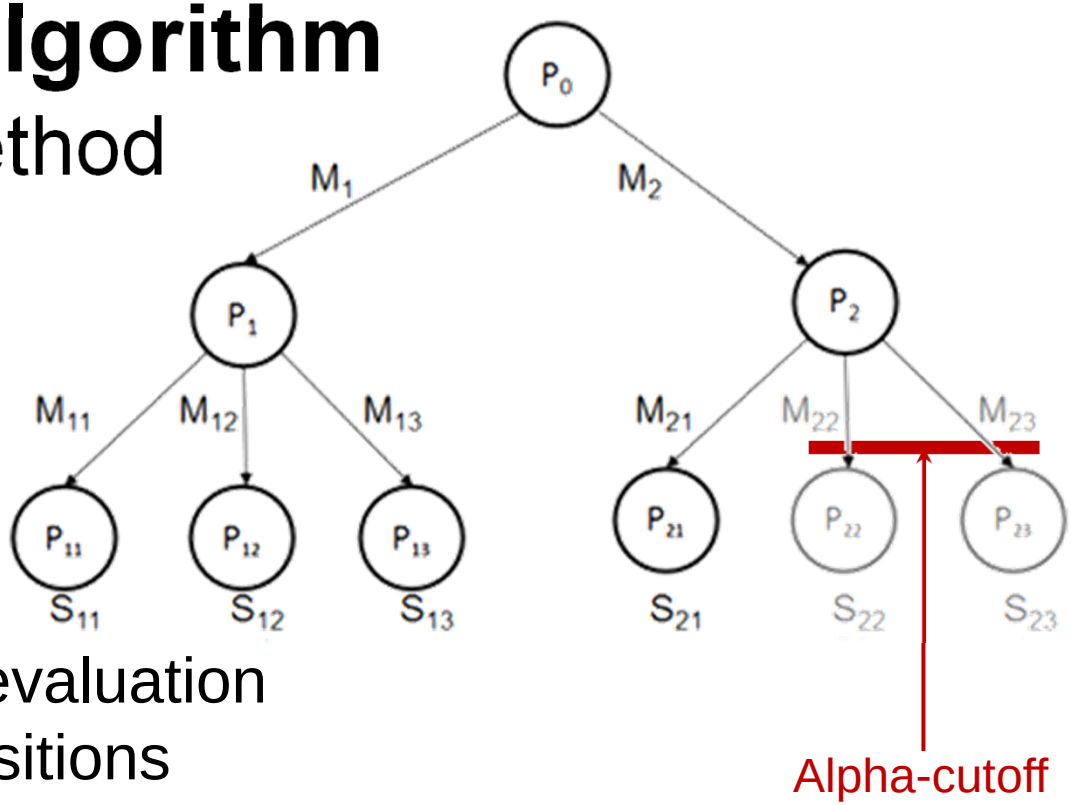
$S_2 = \text{Min}(S_{21} , S_{22} , S_{23}) = S_{21} = 10 = M_{21}$

$S_0 = \text{Max}(S_1 , S_1) = S_2 = 10 = M_2$

Thus M_2 is the selected move.

Alpha-Beta Algorithm

Tree pruning method

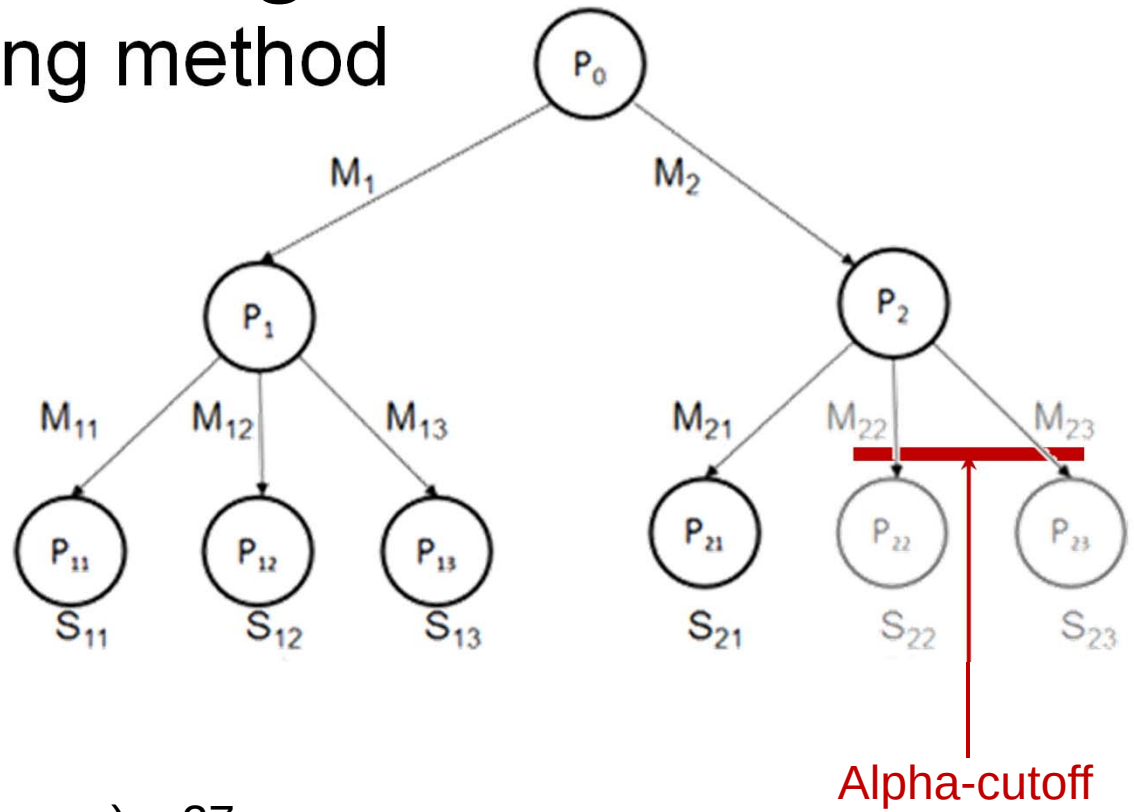


- Tree growth control
- Eliminates useless evaluation and searching of positions
- Based upon Move Refutation strategy

After a move has been searched & scored (tree path), when other moves are searched if they lead to positions that allow the opponent to force a lower score then the program need not explore the moves.

Alpha-Beta Algorithm

Tree pruning method



S_1 = Min (S_{11} , S_{12} , S_{13} , ≡) = 27

S_{21} = 22

Since $S_{21} < S_1$ positions P_{22} , P_{23} ≡ need not be evaluated.

Reasoning:

IF S_{22} , S_{23} , ≡ > S_{21} THEN opponent would not choose them (*min*)

IF S_{22} , S_{23} , ≡ < S_{21} THEN $S_2 < S_1$ & m/c will not choose M_2 (*max*)

Alpha-Beta Algorithm

Tree pruning method

Alpha highest score currently achieved for the machine
Beta highest score (negative) currently achieved for the opponent

Alpha Cutoffs

- Occur at odd depths when opponent's move could reduce program's score

Beta Cutoffs

- Occur at even depths when program could reduce the "best" score achieved for the opponent

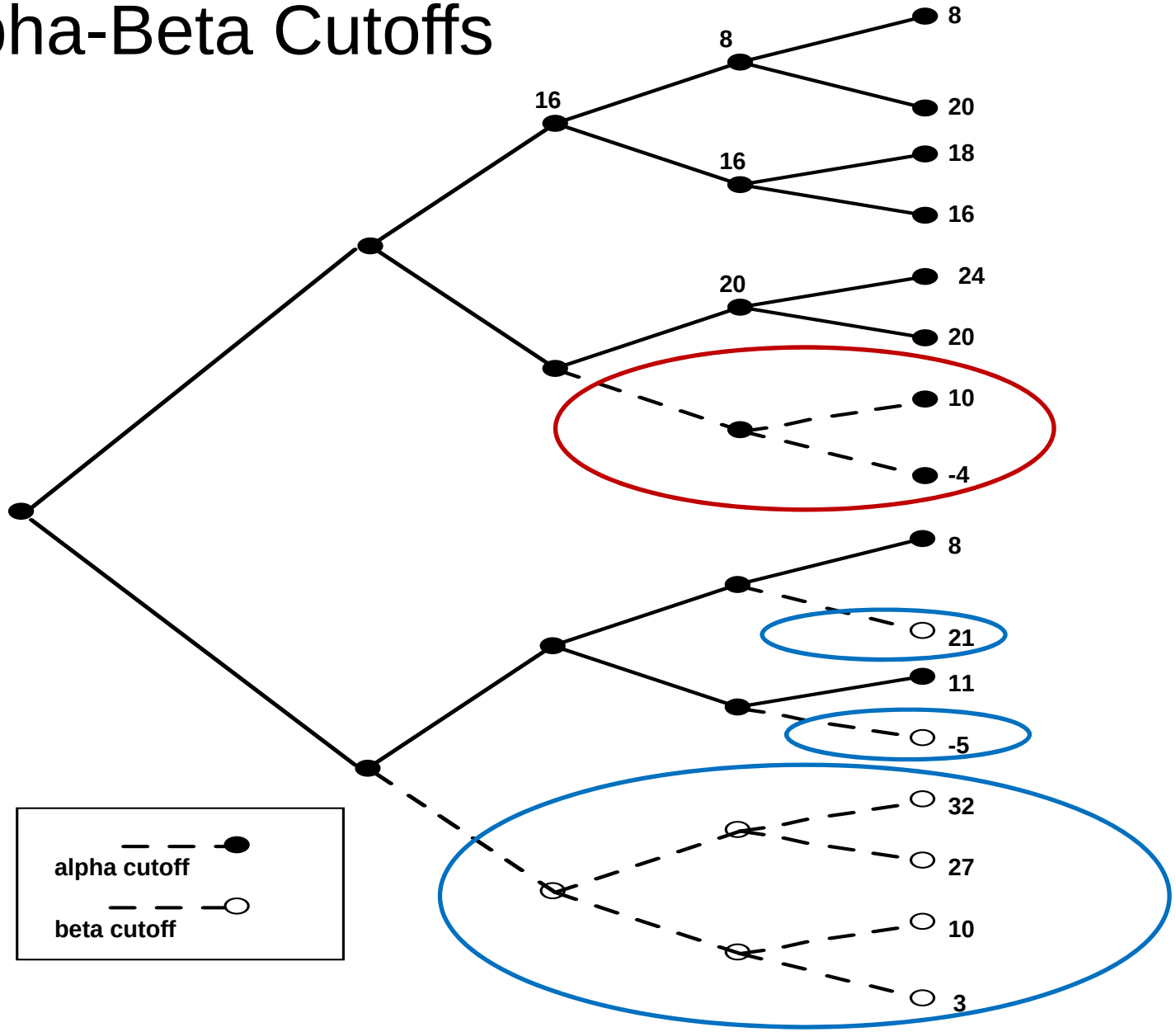
$\alpha-\beta$ pruning reduces the search time by 80%

→ 15 minute move is reduced to a 3 minute move.

$\alpha-\beta$ pruning combined with iterative deepening (plausibility scoring) achieves a total reduction of 90%

→ 30 minute move is reduced to a 3 minute move.

Alpha-Beta Cutoffs



YouTube: Computer Chess

- The History of Chess Computers
- Alan Turing - The first ever Chess program

- Chess Computer Science - Artifical Intelligence Paper: How do Chess Engines work?

- Nova: Kasparov versus Deep Thought
  - Part 1
  - Part 2
  - Part 3
  - Part 4

- Deep Blue beat G. Kasparov in 1997

- **Chess Programming on the Web**
  - chessprogramming.wikispaces.com


At the critical move, the Super Chess Computer went blank, once again proving humankind's superiority over machines.