Jamboree 🎉

(*Parallel* Alpha-Beta)

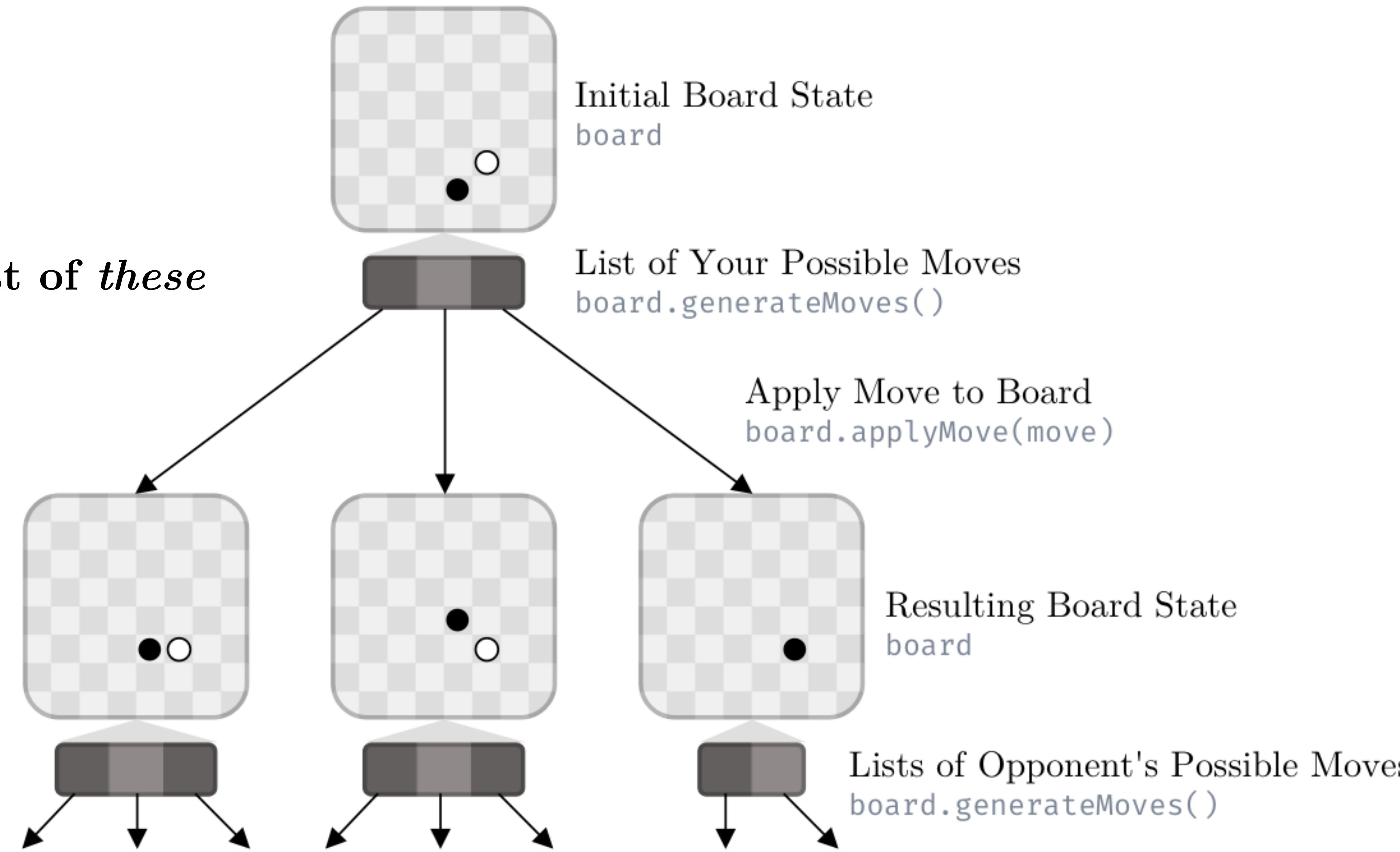
All Searchers, Basically

Choose between moves by
imagining what would happen

All Searchers, Basically

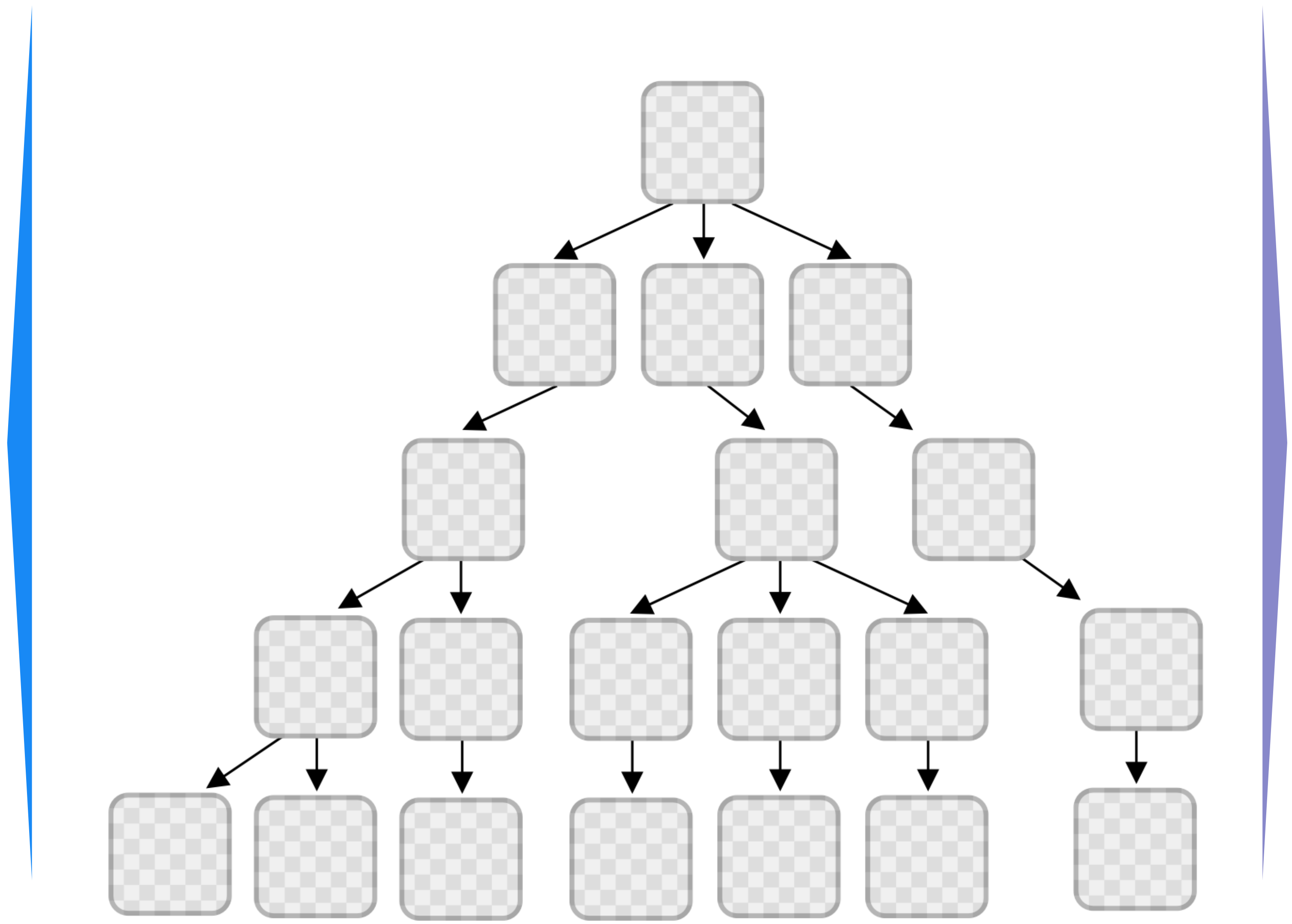
Find the best of *these*

Assuming your opponent responds with the best of *these*...



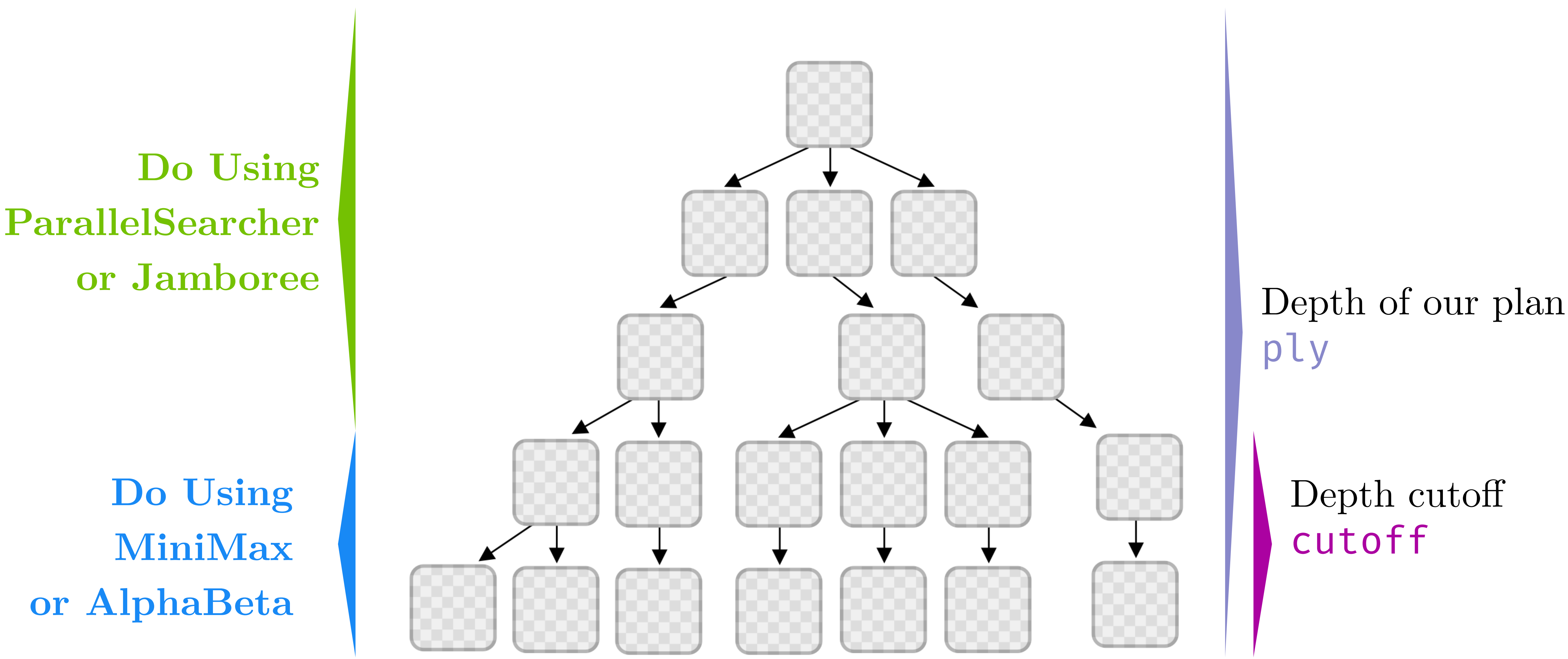
Sequential Searchers

Do Using MiniMax or AlphaBeta



Depth of our plan
`ply`

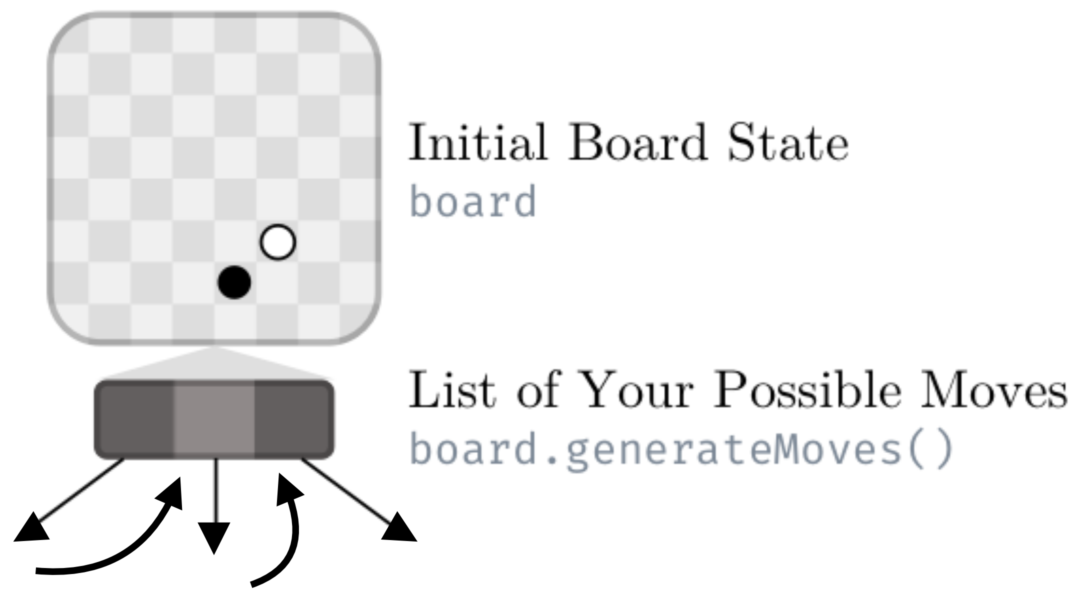
Parallel Searchers



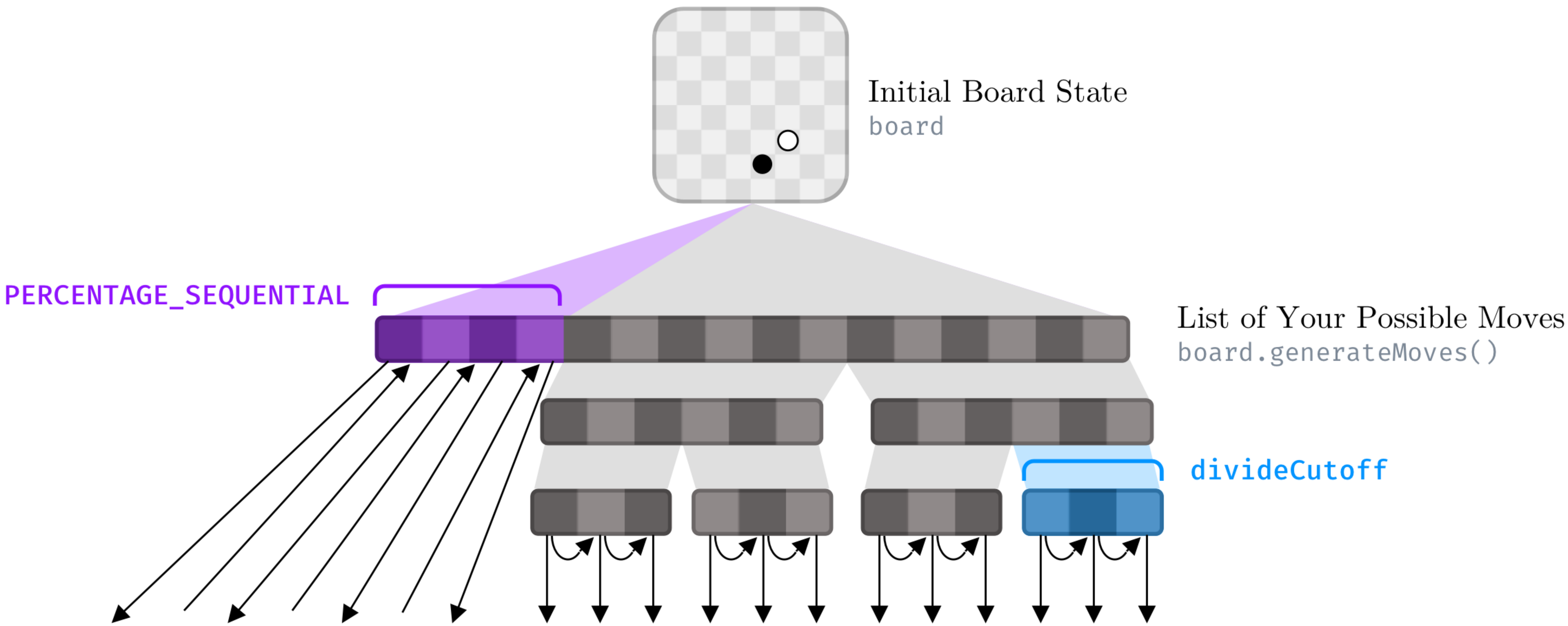
AlphaBeta

- Pruning in AlphaBeta relies on having a good value for α; if we see a good move early on, we can prune a lot later.
- If we try all moves in parallel, we can't prune at all!
- So we do *some* moves sequentially to get a reasonable value of α, and then use that "good enough" α to do the rest in parallel.

AlphaBeta


Initial Board State
board
List of Your Possible Moves
board.generateMoves()

Jamboree



Initial Board State
board
PERCENTAGE_SEQUENTIAL
List of Your Possible Moves
board.generateMoves()
divideCutoff

Jamboree

```
PERCENTAGE_SEQUENTIAL = 0.5;
int jamboree(Position p, int alpha, int beta) {
   if (p is a leaf) {
      return p.evaluate();
   }

   moves = p.getMoves();

   for (i = 0; i < PERCENTAGE_SEQUENTIAL * moves.length; i++) {
      p.applyMove(moves[i]);
      int value = -jamboree(p, -beta, -alpha);
      p.undoMove();

      if (value > alpha) {
         alpha = value;
      }
      if (alpha >= beta) {
         return alpha;
      }
   }

   parallel (i = PERCENTAGE_SEQUENTIAL * moves.length; i < moves.length; i++) {
      p = p.copy();
      int value = -jamboree(p, -beta, -alpha);

      if (value > alpha) {
         alpha = value;
      }
      if (alpha >= beta) {
         return alpha;
      }
   }

   return alpha;
}
```

Jamboree

Full Tree

Single Board State