JUEGOS

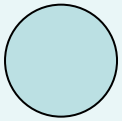
Algoritmo MiniMax

```
Minimax (nodo,nivel) devuelve (valor, nodo_resultado)
    si final(node) entonces  devuelve (+∞ / - ∞ , nodo_vacio)
    sino si  nivel = nivel_maximo_minimax                                  [constante]
            entonces    devuelve    (h(nodo) , nodo_vacio)
            sino    nodo_a_devolver := nodo_vacio
                    si nivel = MAX      entonces    valor_a_devolver := - ∞
                                        sino        valor_a_devolver := +∞
                    fsi
                    mientras quedan_hijos(nodo) hacer
                            F := siguiente_hijo(nodo)
                            (val , nuevo_nodo) := MINIMAX ( F , nivel +1)
                            si nivel = MAX
                            entonces    si val > val_a_devolver
                                        entonces    val_a_devolver := val
                                                    nodo_a_devolver := F
                                        fsi
                            sino        si val < val_a_devolver
                                        entonces    val_a_devolver := val
                                                    nodo_a_devolver := F
                                        fsi
                            fsi
                    fmientras
                    devuelve (val_a_devolver, nodo_a_devolver)
    fsi
```

Algoritmo MiniMax

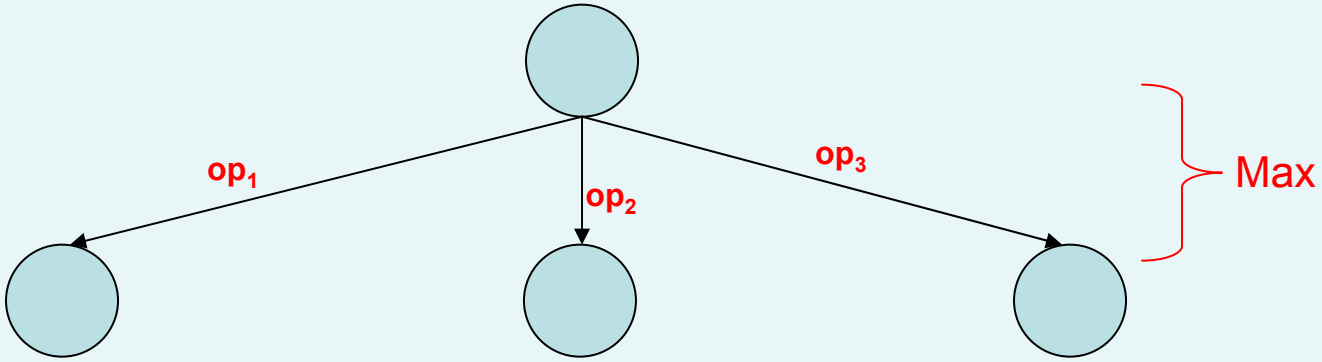
- Juegos de dos jugadores con un tablero común
- Se alternan los niveles MAX / MIN desde la raíz
- Se evaluan los tableros desde el último nivel con la función heurística
- Se pasan los valores hacia la raiz, con MAX y MIN
- El valor mas alto en la raiz indica la jugada a realizar (en caso de empate se escoge al azar)

Algoritmo MiniMax

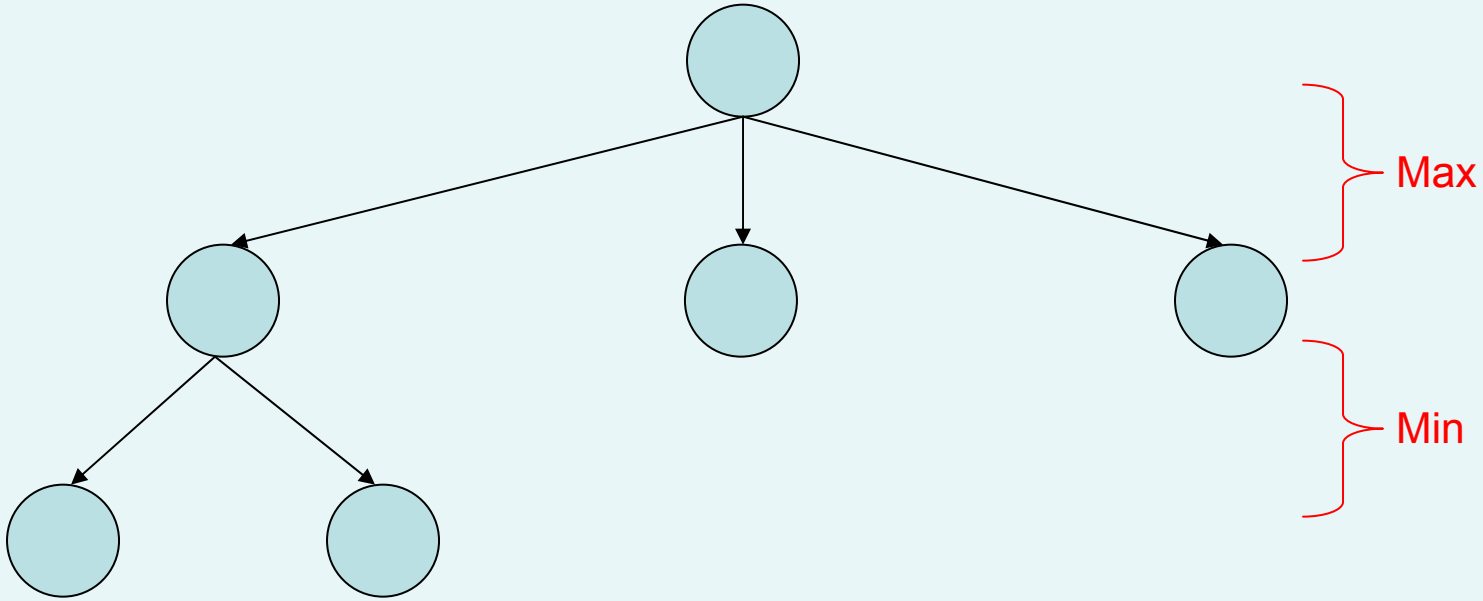


Nodo inicial

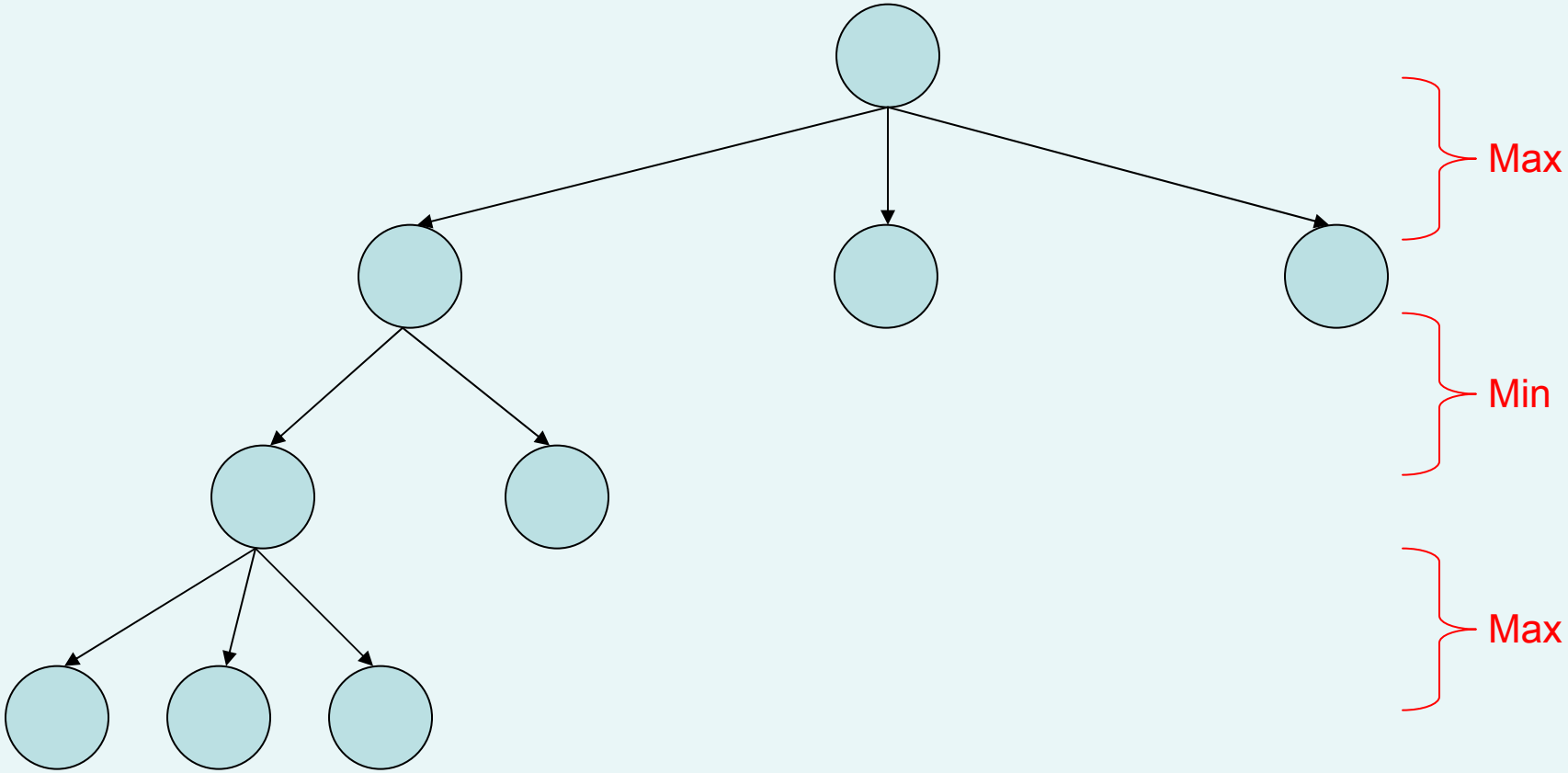
Algoritmo MiniMax



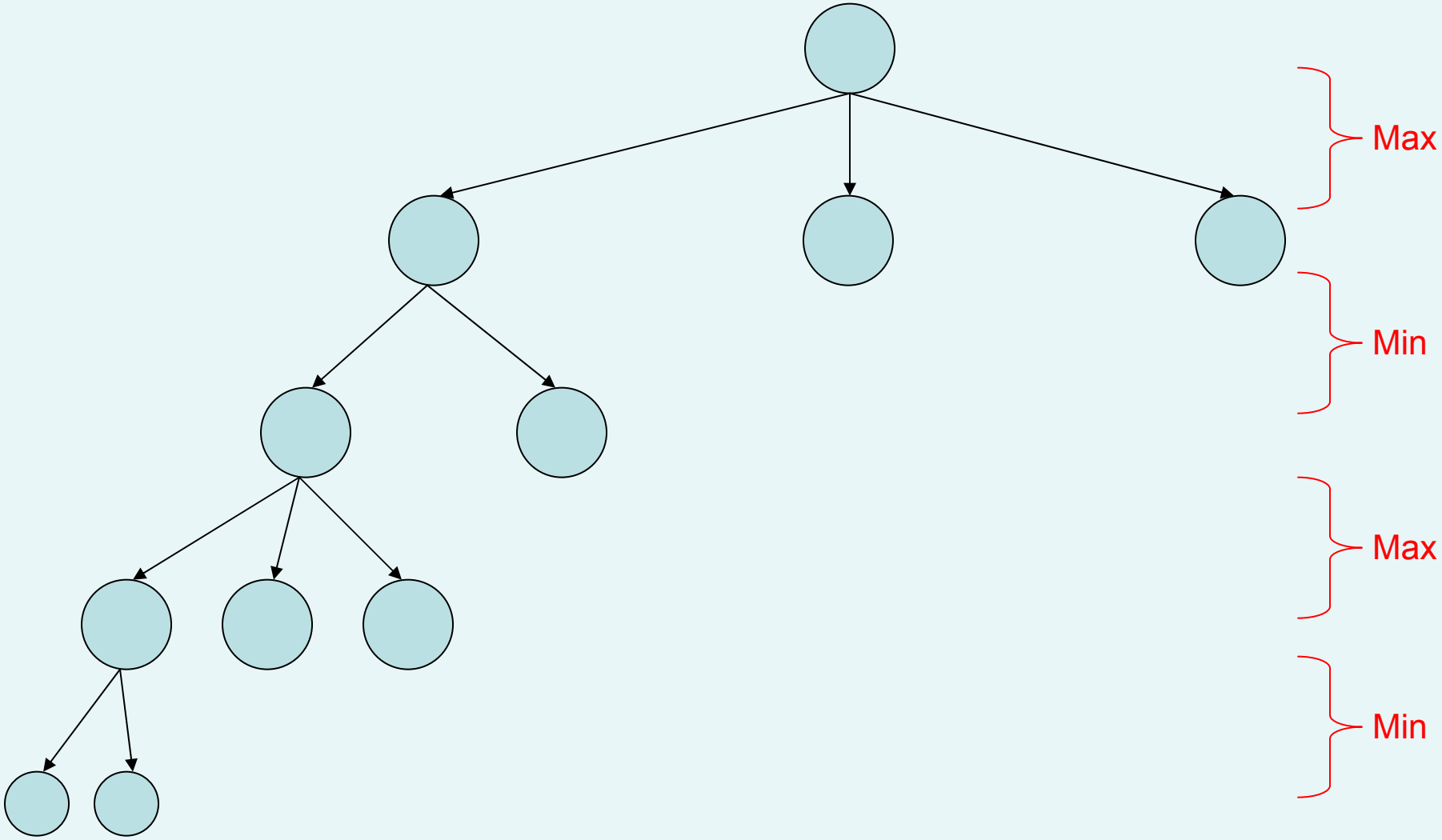
Algoritmo MiniMax



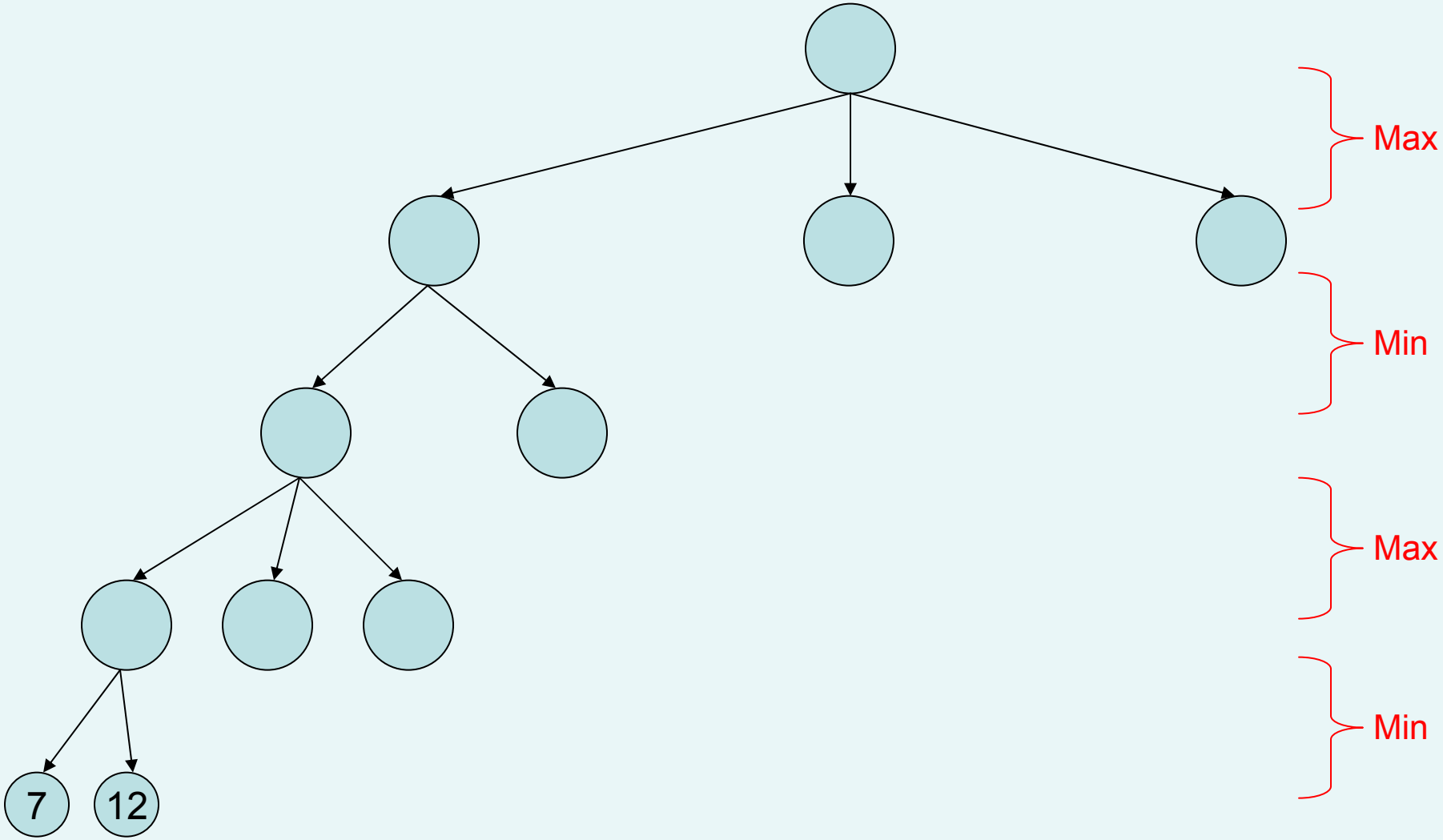
Algoritmo MiniMax



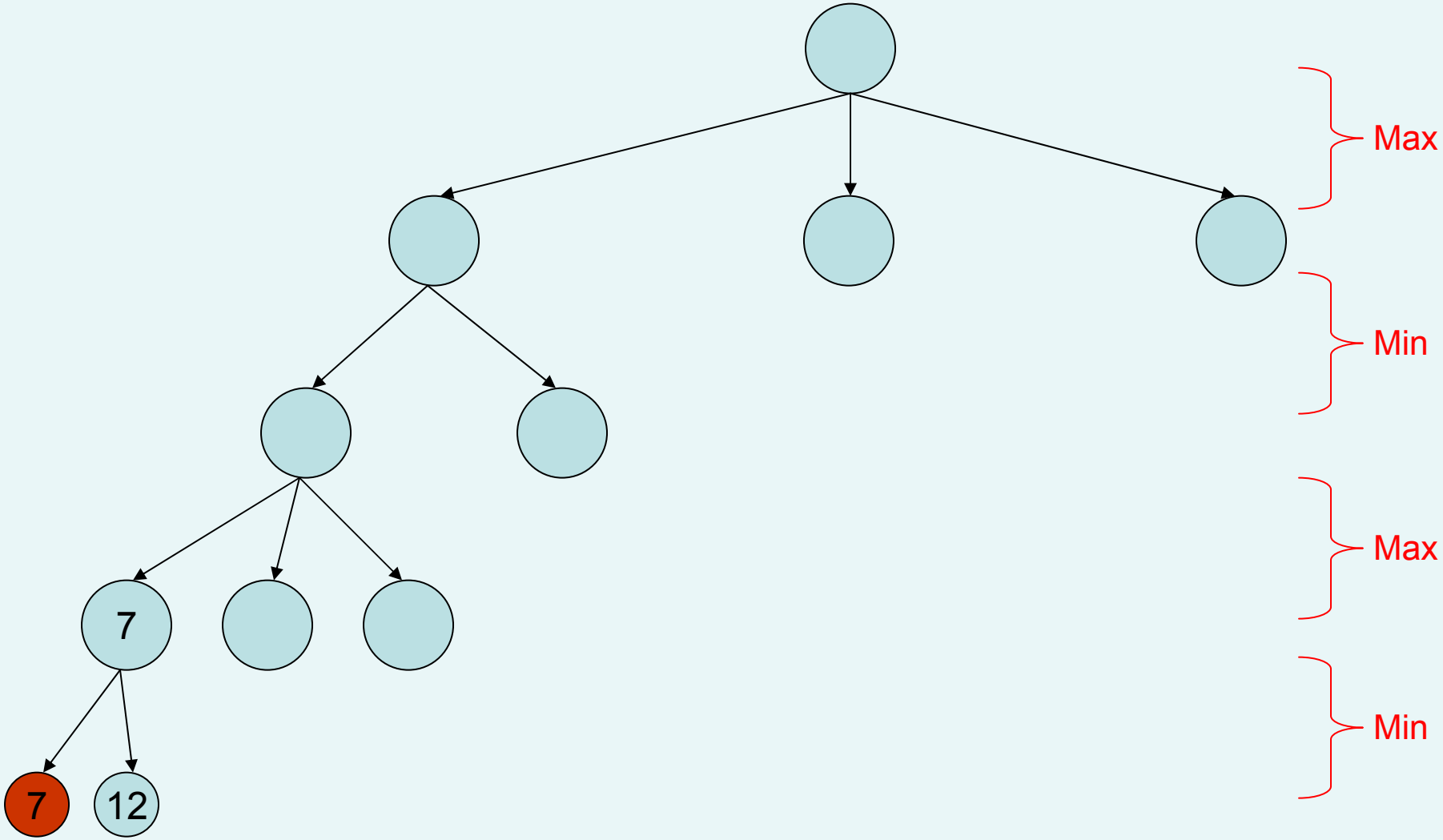
Algoritmo MiniMax



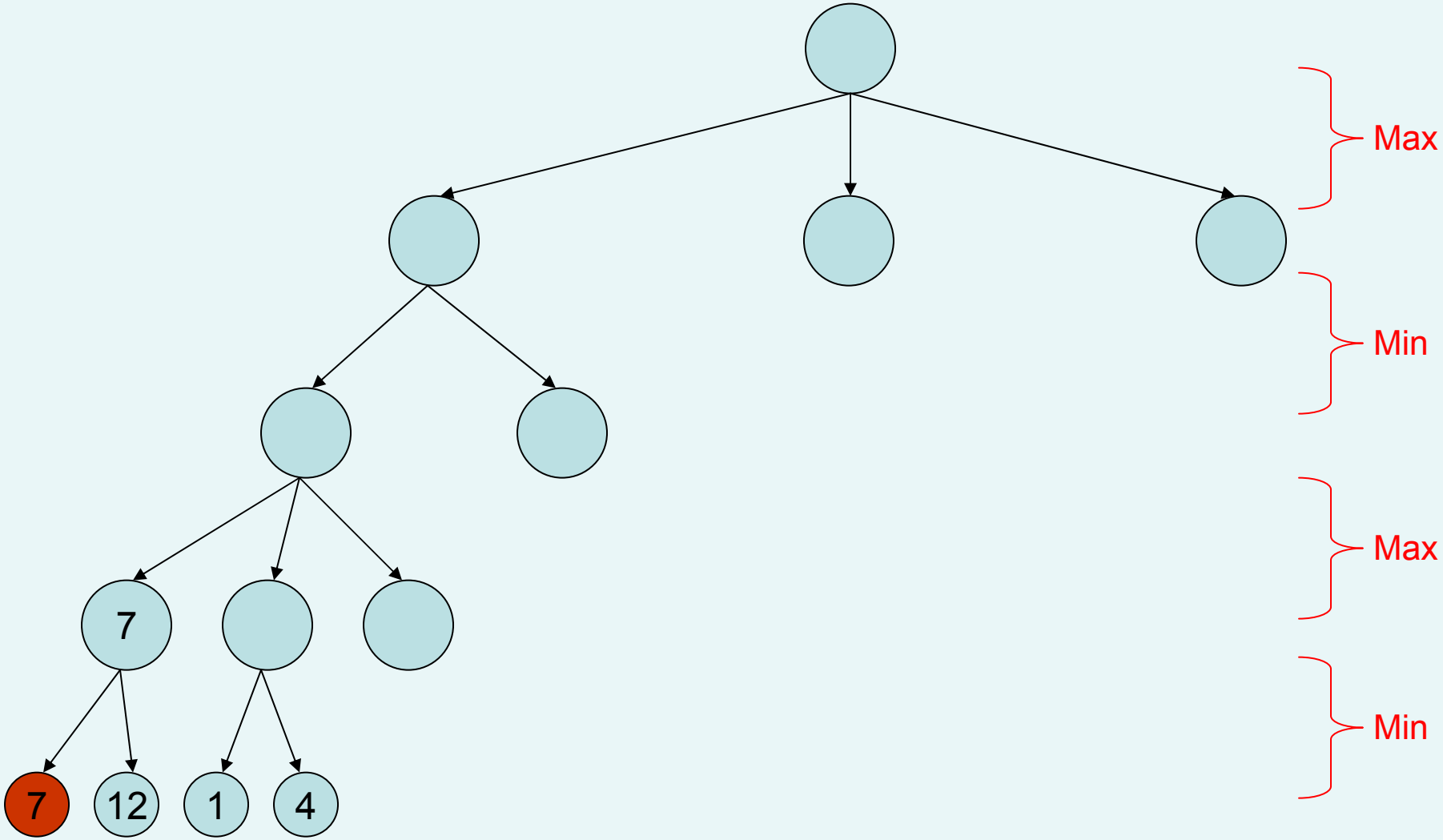
Algoritmo MiniMax



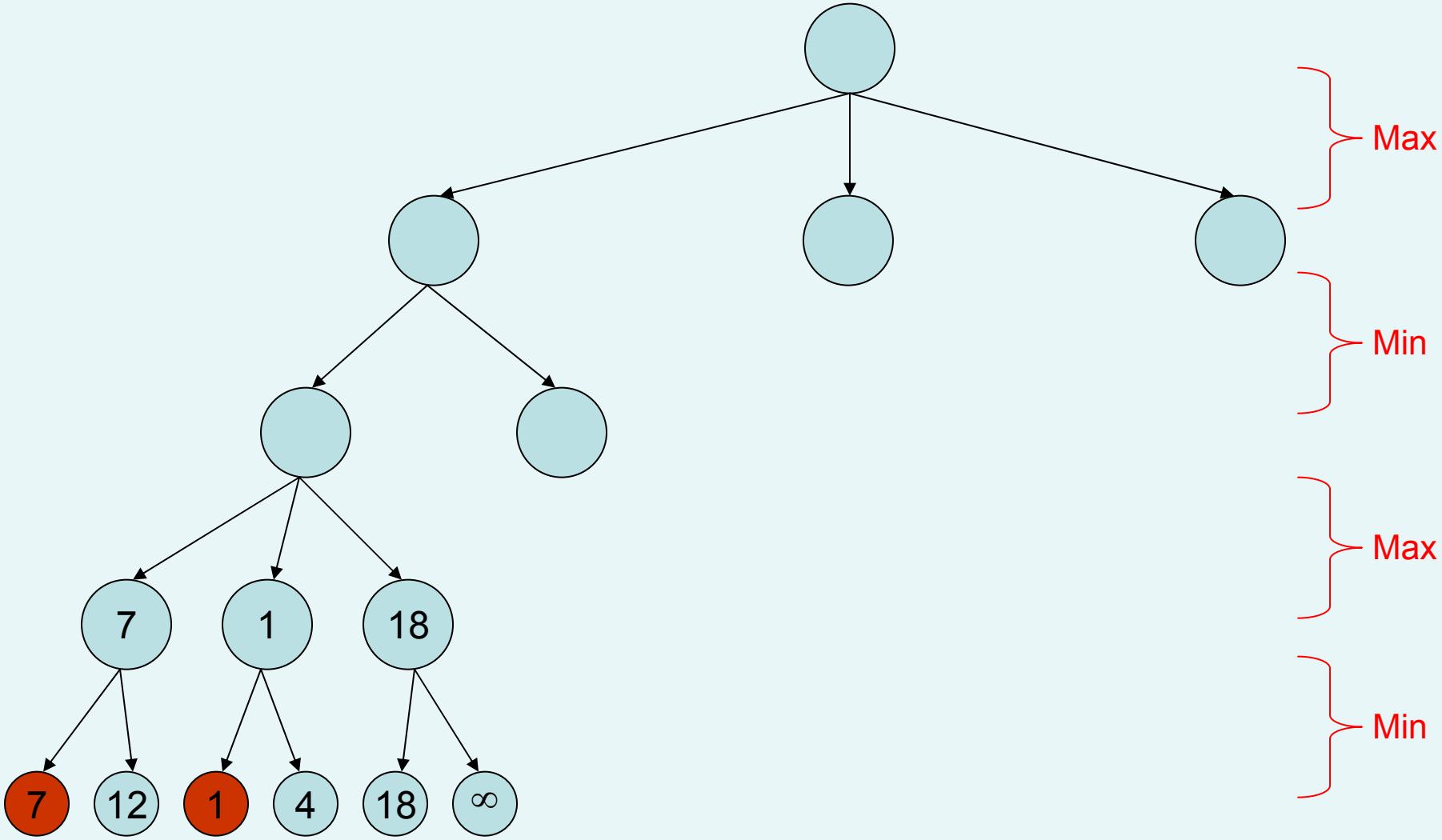
Algoritmo MiniMax



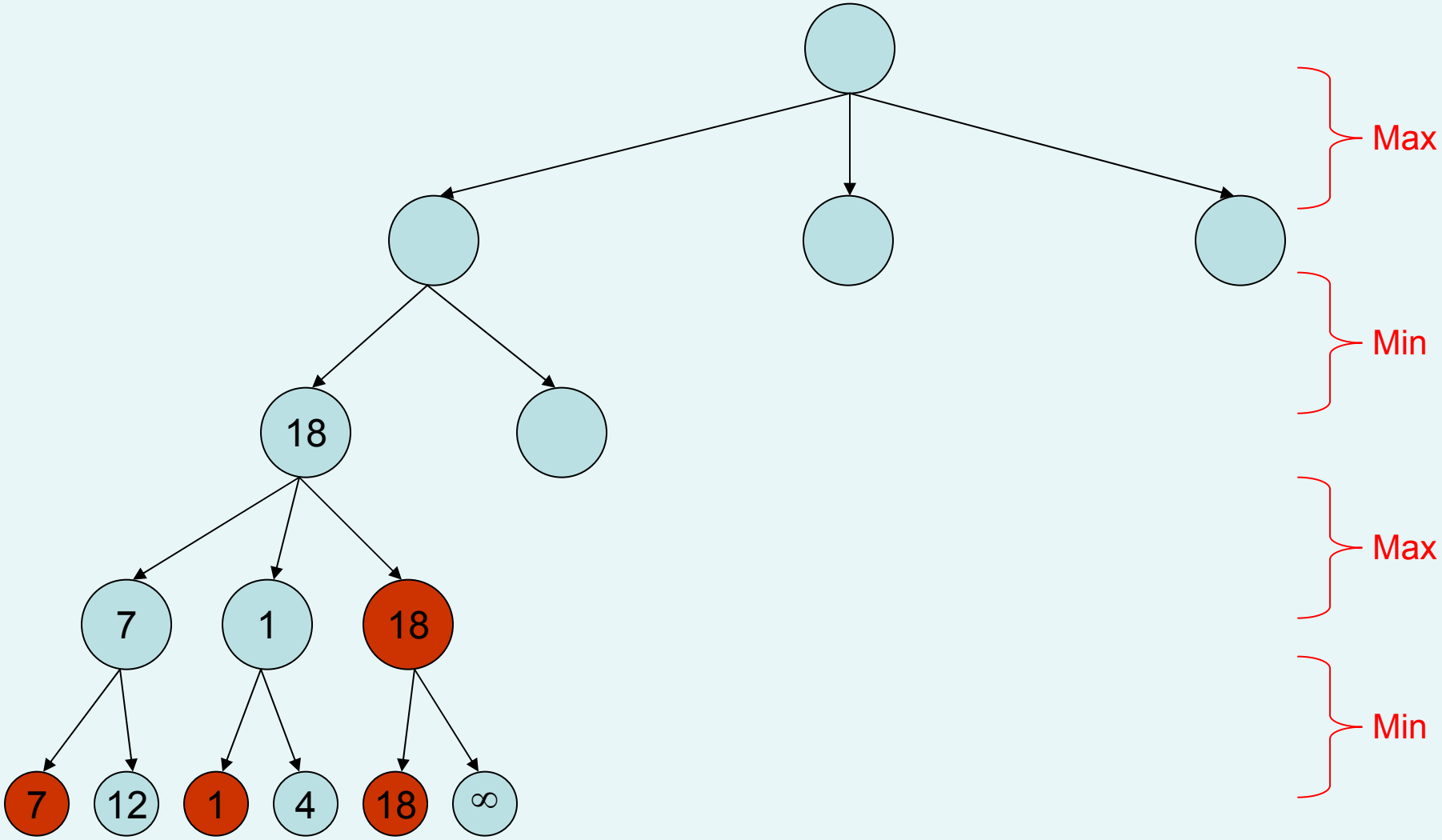
Algoritmo MiniMax



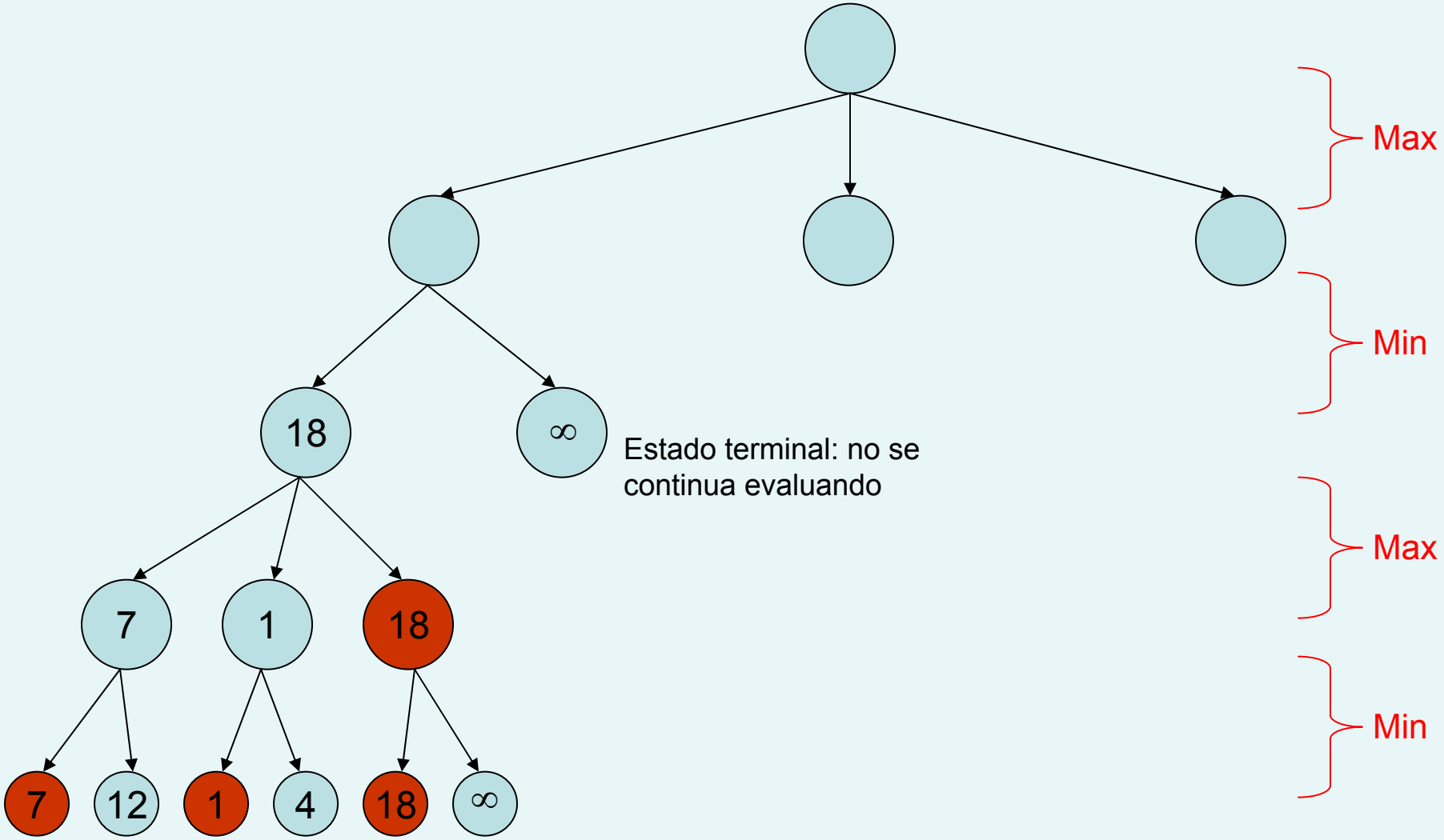
Algoritmo MiniMax



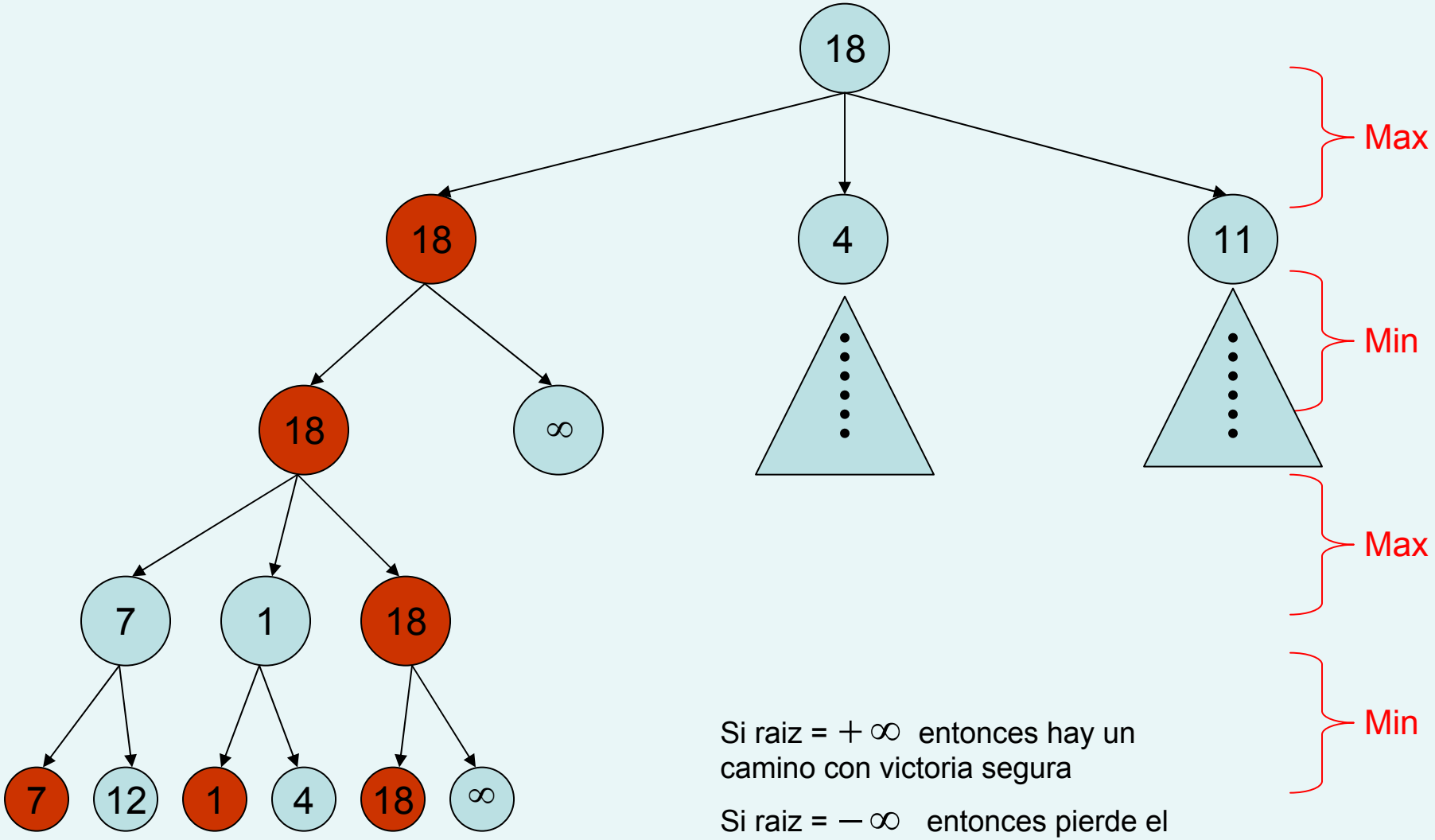
Algoritmo MiniMax



Algoritmo MiniMax



Algoritmo MiniMax



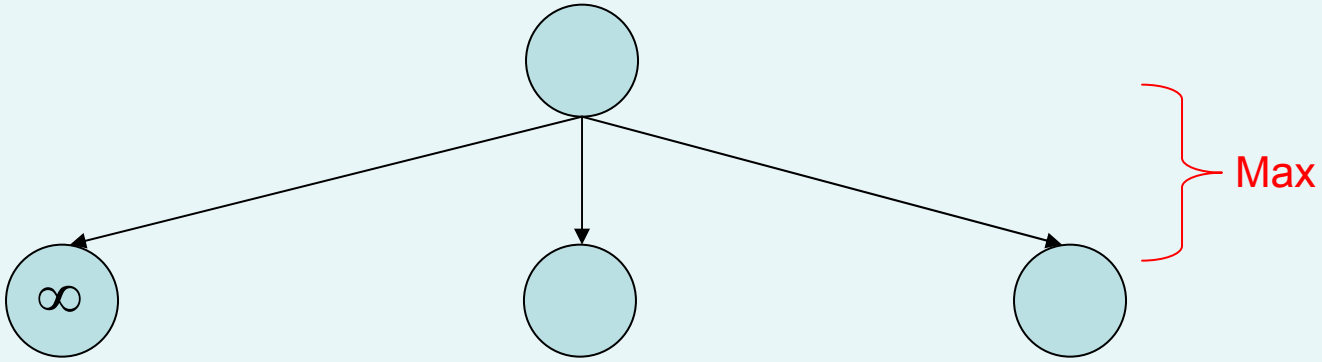
Si raiz = $+\infty$ entonces hay un camino con victoria segura

Si raiz = $-\infty$ entonces pierde el ordenador haga lo que haga

Factores que intervienen en el juego:

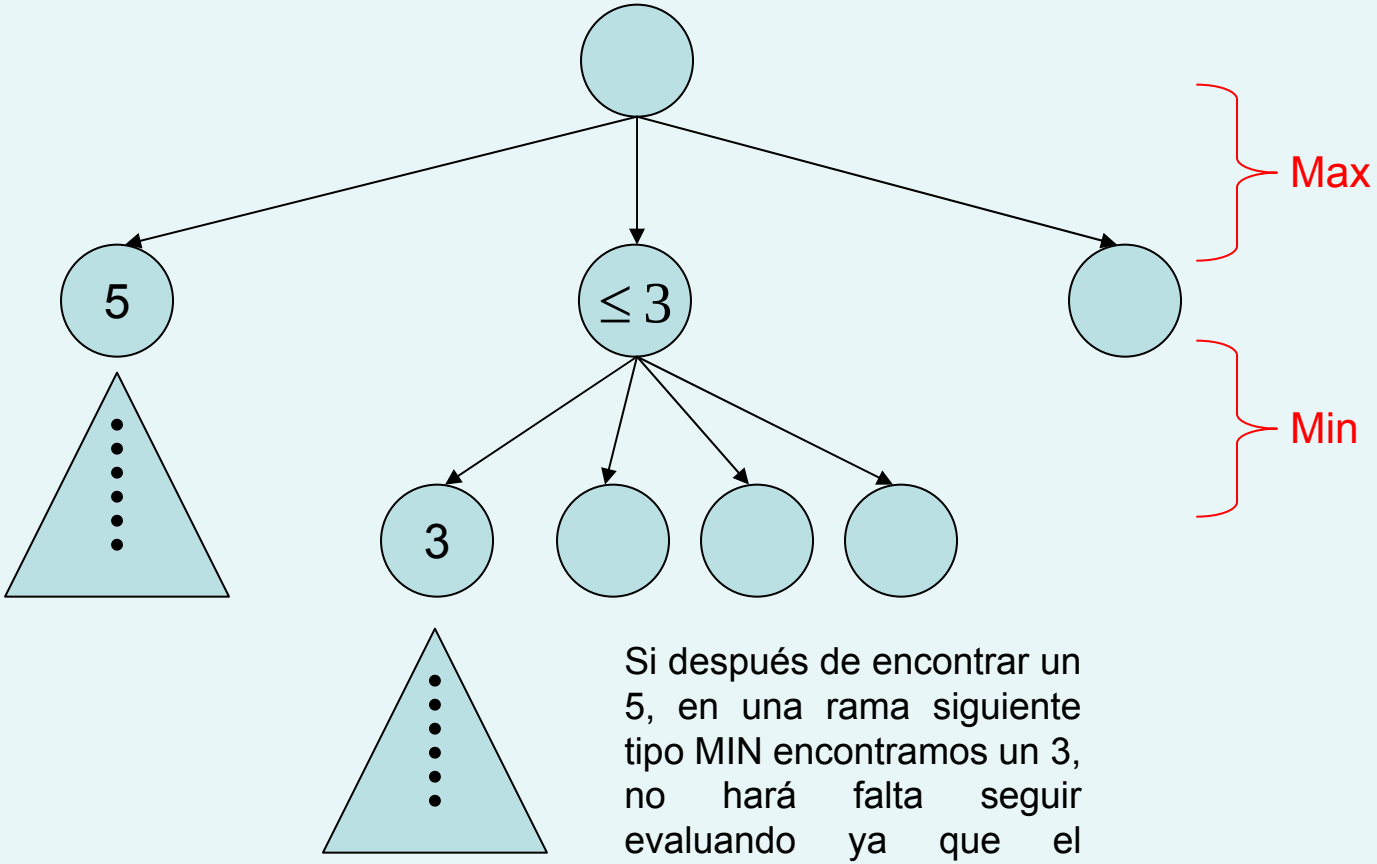
- Numero máximo de niveles
  - Mejor juego
  - Aumenta el espacio y MUCHO el tiempo necesario
- Profundidad del análisis del heurístico
  - Mejor juego
  - Aumenta MUCHO el tiempo necesario

Poda alfa-beta



Si el primer nodo que evaluamos es una victoria segura, no hace falta seguir evaluando mas nodos.

Poda alfa-beta



Si después de encontrar un 5, en una rama siguiente tipo MIN encontramos un 3, no hará falta seguir evaluando ya que el resultado no se mejorara. Subirá 3 o menos.

Si vamos evaluando y pasando los valores en profundidad, podemos ahorrarnos la exploración de ramas que no son útiles para mejorar los resultados.
Con el mismo tiempo, podemos mirar mas niveles y evaluar mejor.

Poda alfa-beta

α = Valor mínimo que tiene garantizado un nodo MAX (cota inferior)

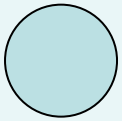
β = Valor máximo que tiene garantizado un nodo MIN (cota superior)

- Cuando a un nodo **MAX** le llega un valor de sus hijos, intentar subir **α**
- Cuando a un nodo **MIN** le llega un valor de sus hijos, intentar bajar **β**
- Un nodo **MAX** envía el máximo **α** a su padre
- Un nodo **MIN** envía el mínimo **β** a su padre
- Cuando en un nodo pasa que $\alpha \geq \beta$, quiere decir que tendrá un valor más pequeño que el que ya tiene asegurado un MAX
  → no hace falta mirar más ramas.

Poda alfa-beta

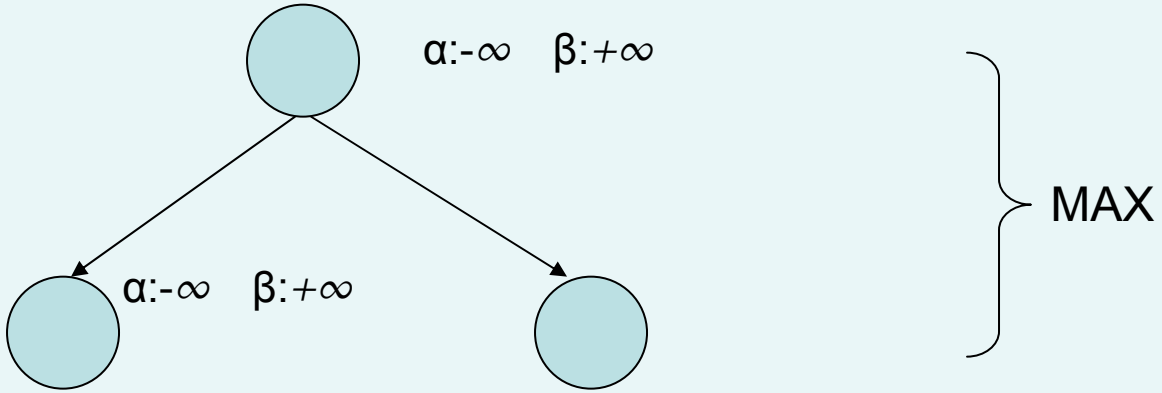
```
ALFA_BETA (nodo,nivel,alfa,beta) devuelve (valor, nodo_resultado)
    si final(node) entonces  devuelve (+∞ / - ∞ , nodo_vacio)
    sino si  nivel = nivel_maximo_minimax                                  [constante]
            entonces    devuelve    (h(nodo) , nodo_vacio)
            sino    nodo_a_devolver := nodo_vacio
                    _                   _______                        :
                                        ____                           :
                    __
                    mientras quedan_hijos(nodo) y (alfa<beta) hacer
                            F := siguiente_hijo(nodo)
                            (val,nuevo_nodo) := ALFA_BETA ( F , nivel +1, alfa, beta)
                            si nivel = MAX
                            entonces    si val > alfa   entonces    alfa := val
                                                                    nodo_a_devolver := F
                                        fsi
                            sino        si val < beta   entonces    beta := val
                                                                    nodo_a_devolver := F
                                        fsi
                            fsi
                    fmientras
                    si  nivel = MAX     entonces    devuelve (alfa, nodo_a_devolver)
                                        sino        devuelve (beta, nodo_a_devolver)
                    fsi
    fsi
```

Poda alfa-beta

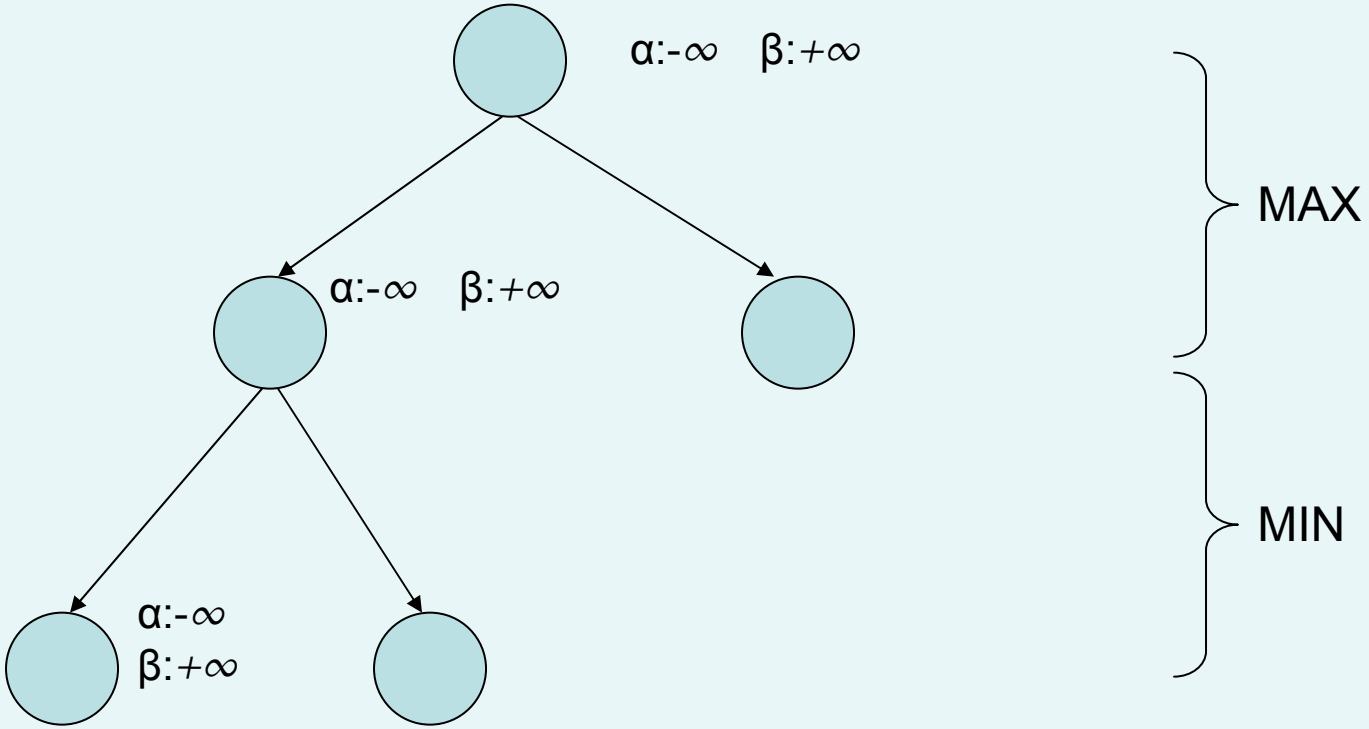


α:-∞ β:+∞

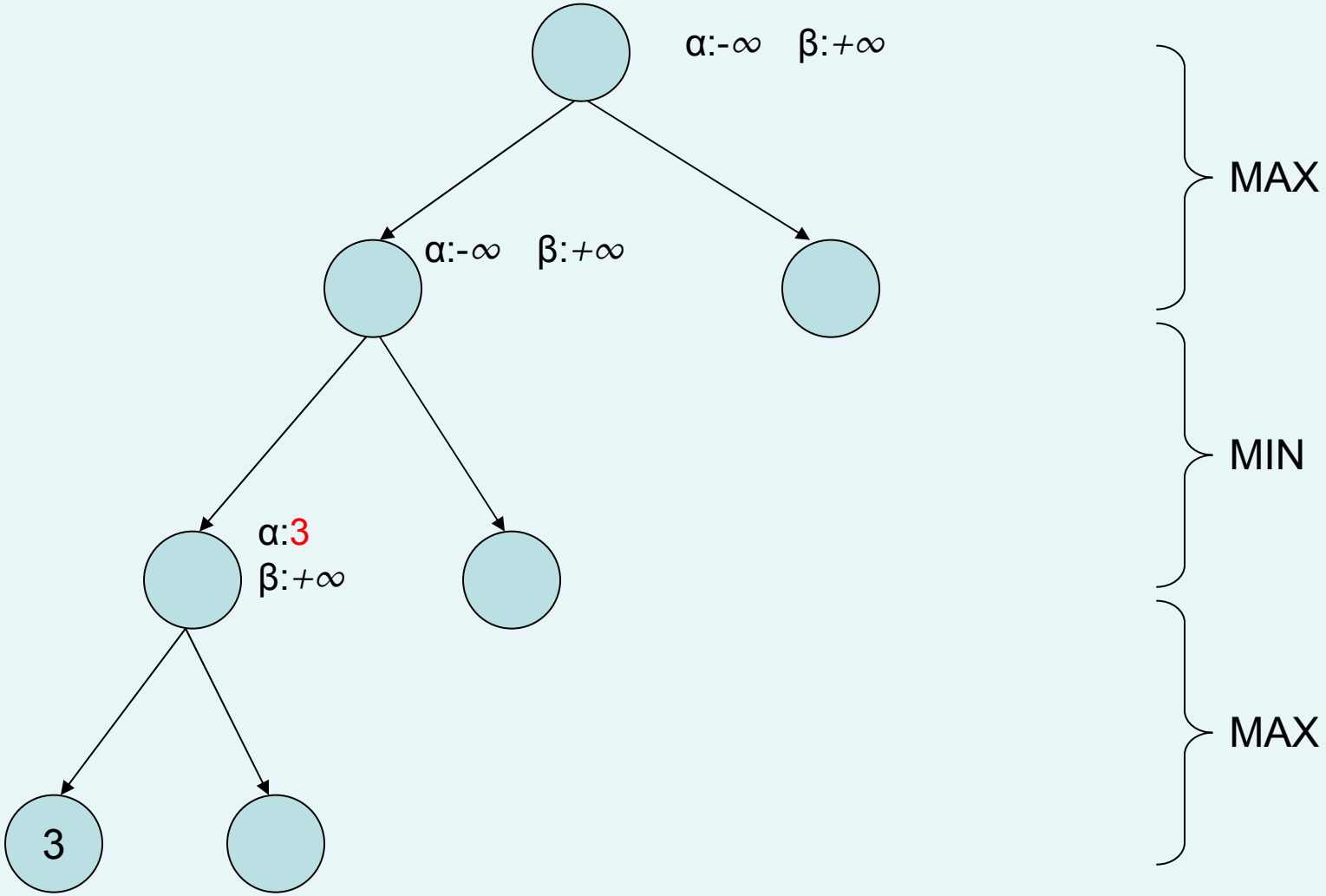
Poda alfa-beta



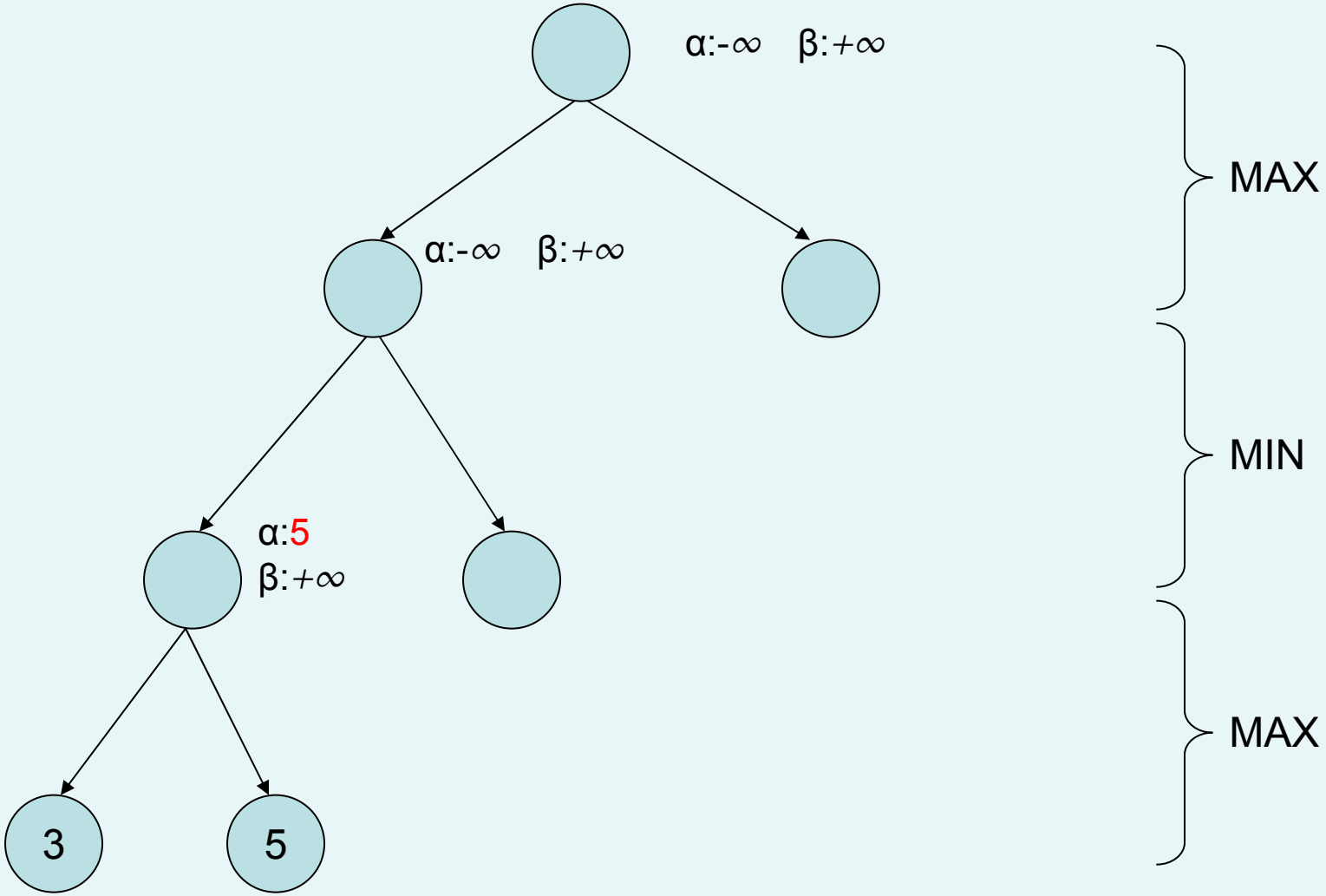
Poda alfa-beta



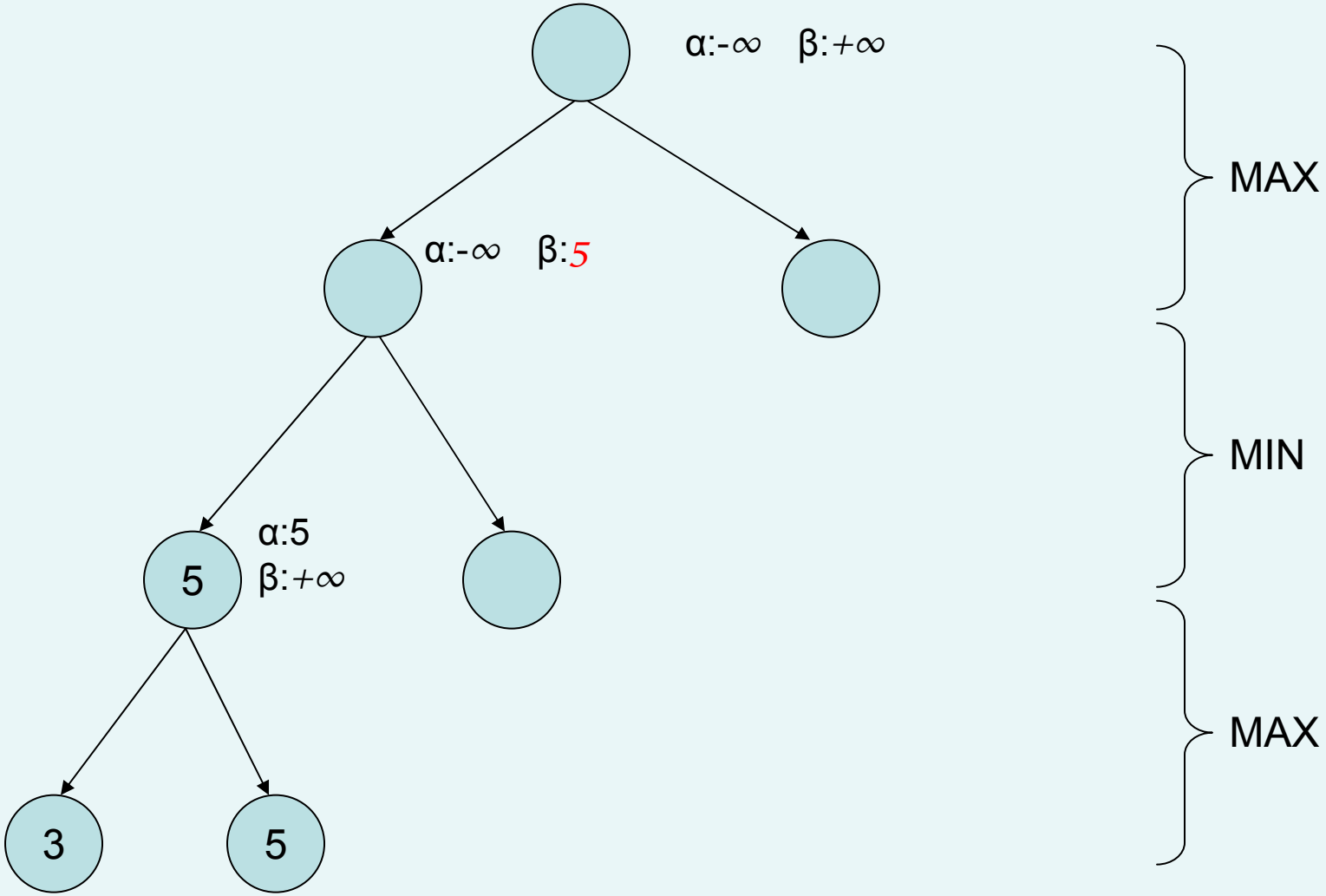
Poda alfa-beta



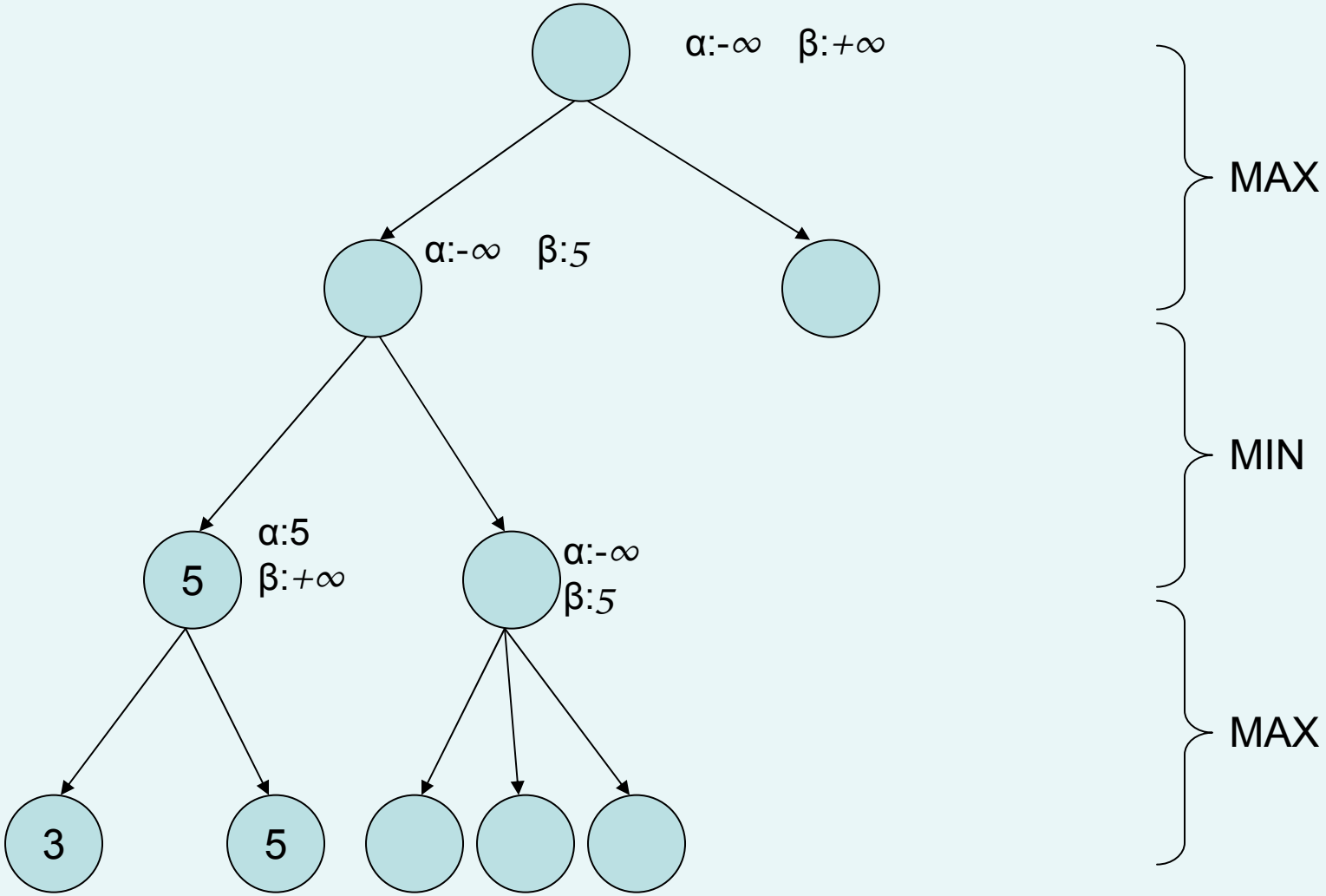
Poda alfa-beta



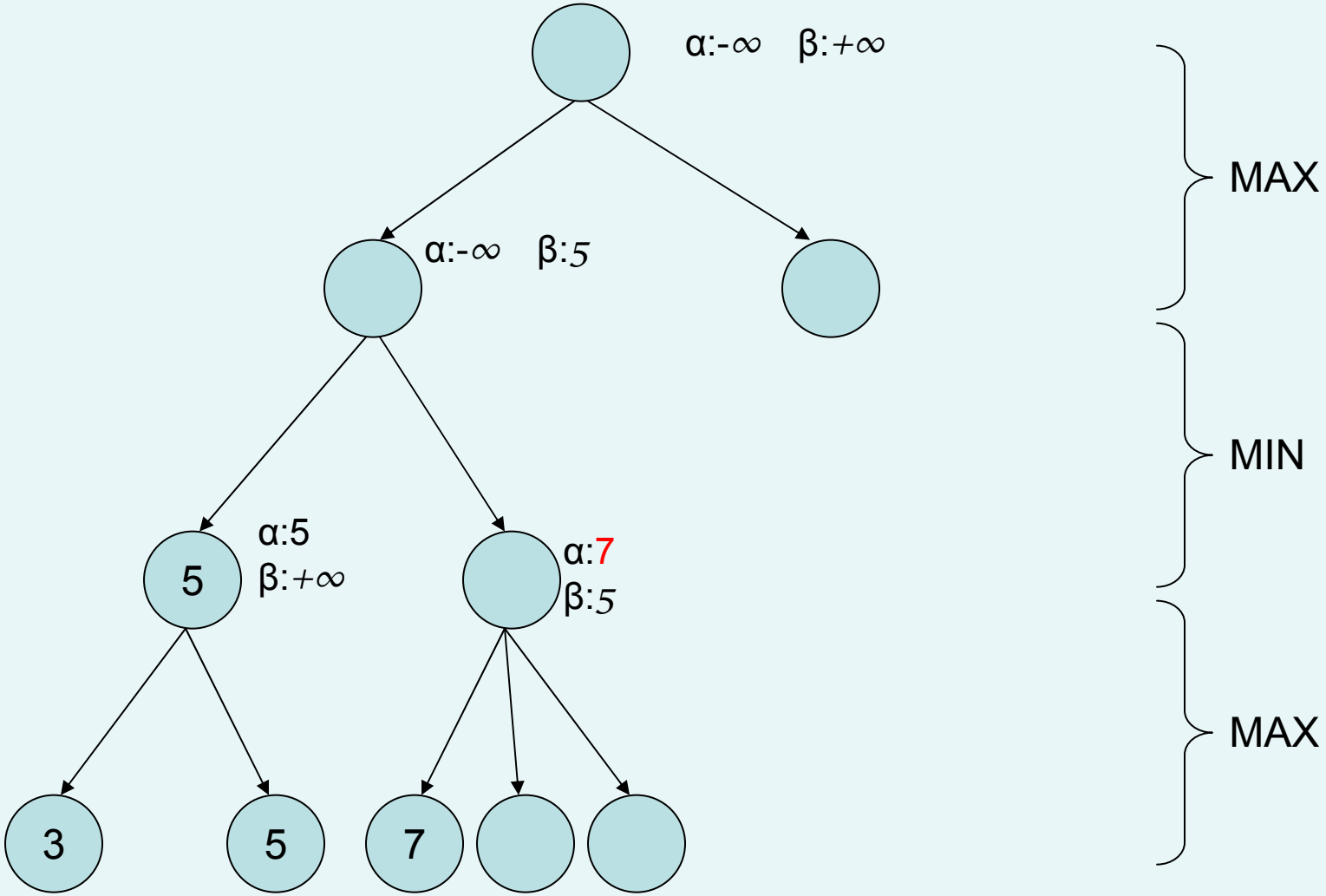
Poda alfa-beta



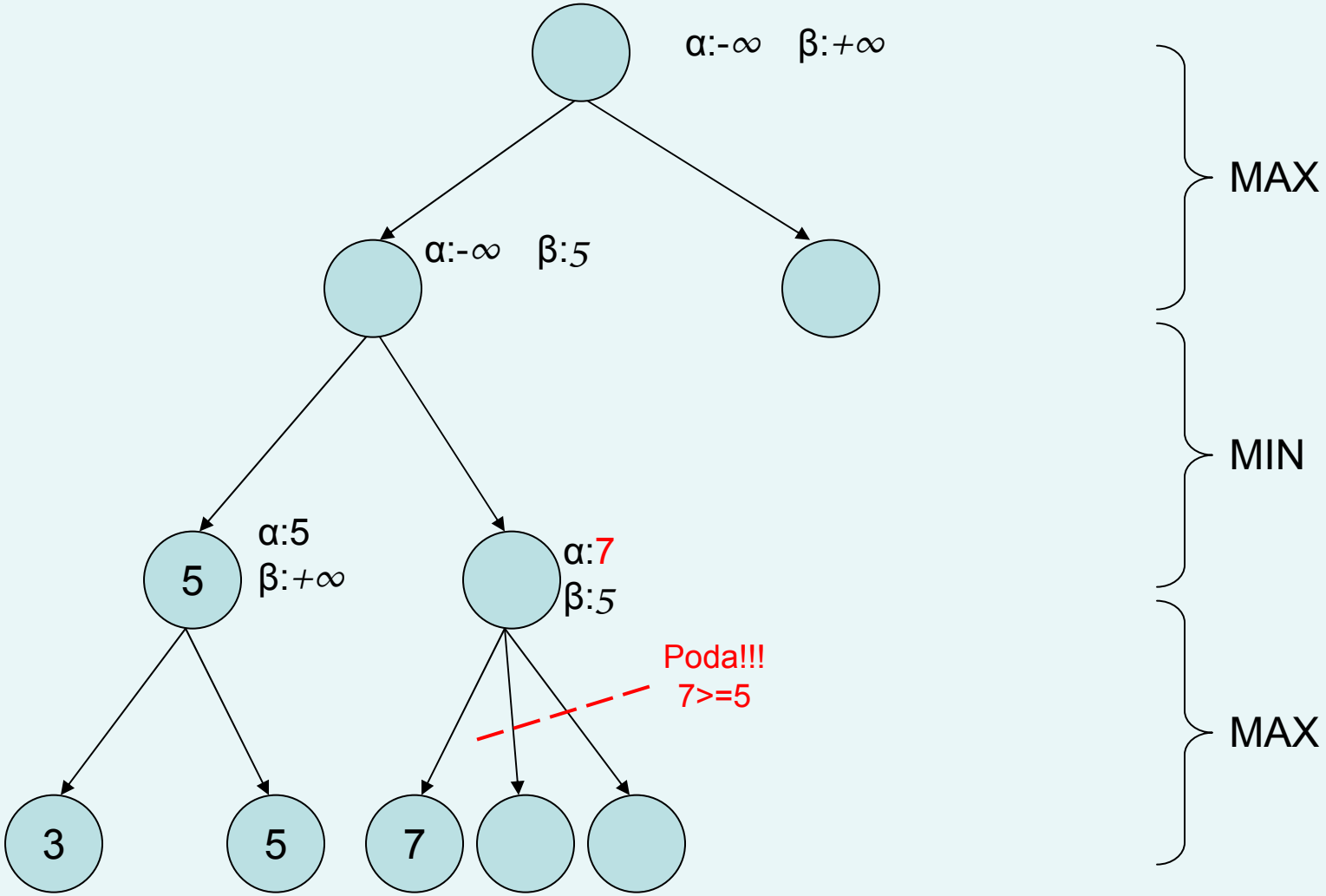
Poda alfa-beta



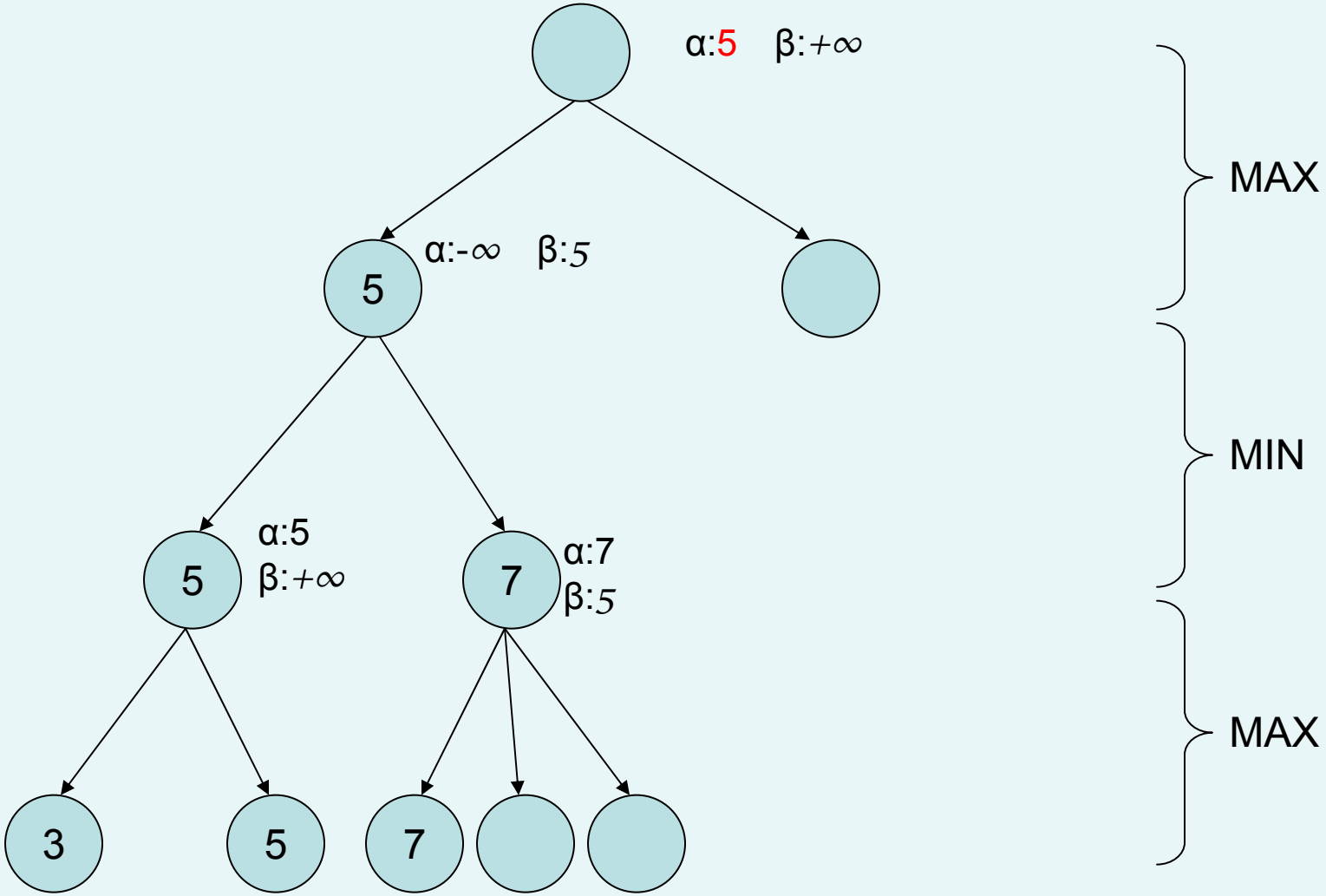
Poda alfa-beta



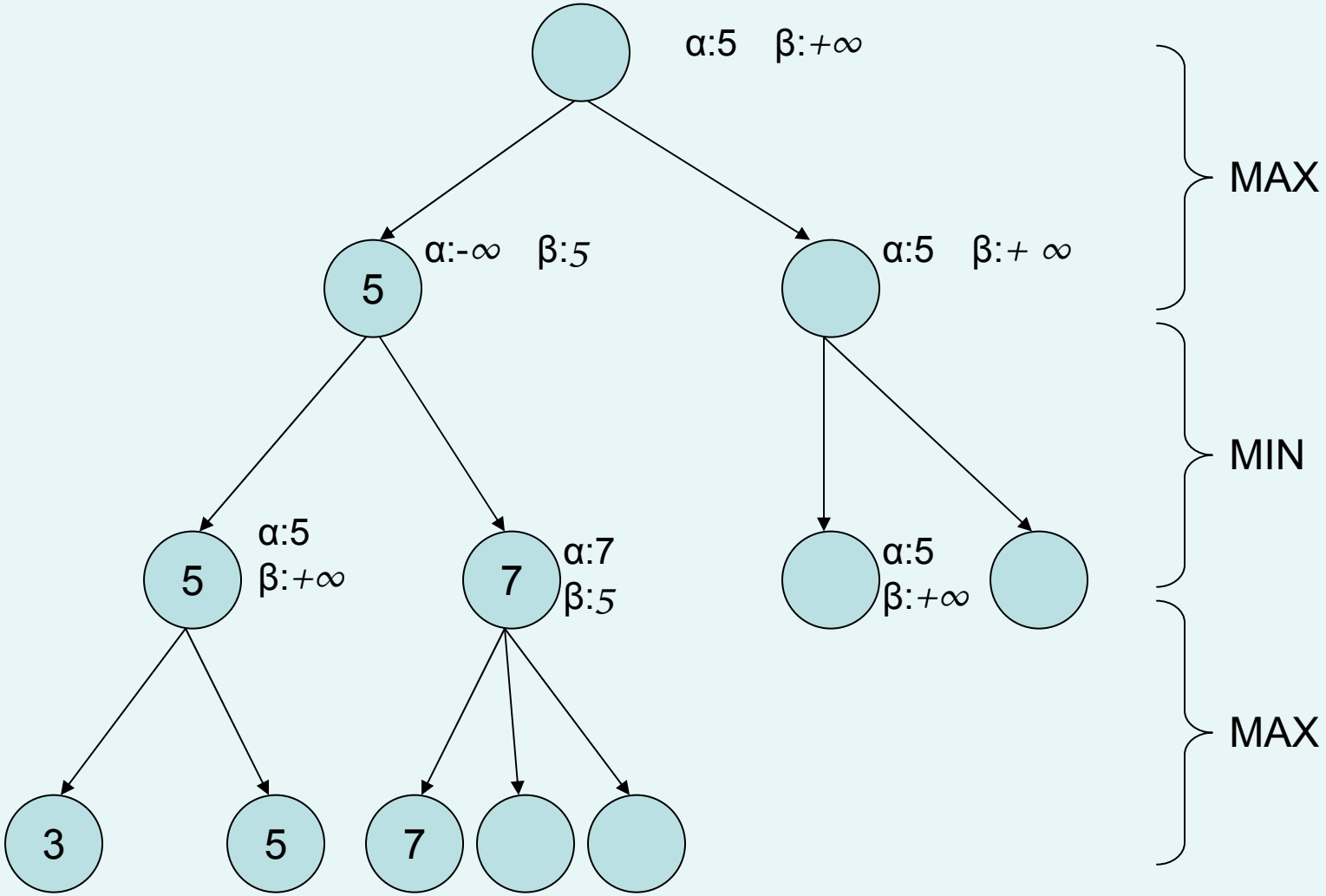
Poda alfa-beta



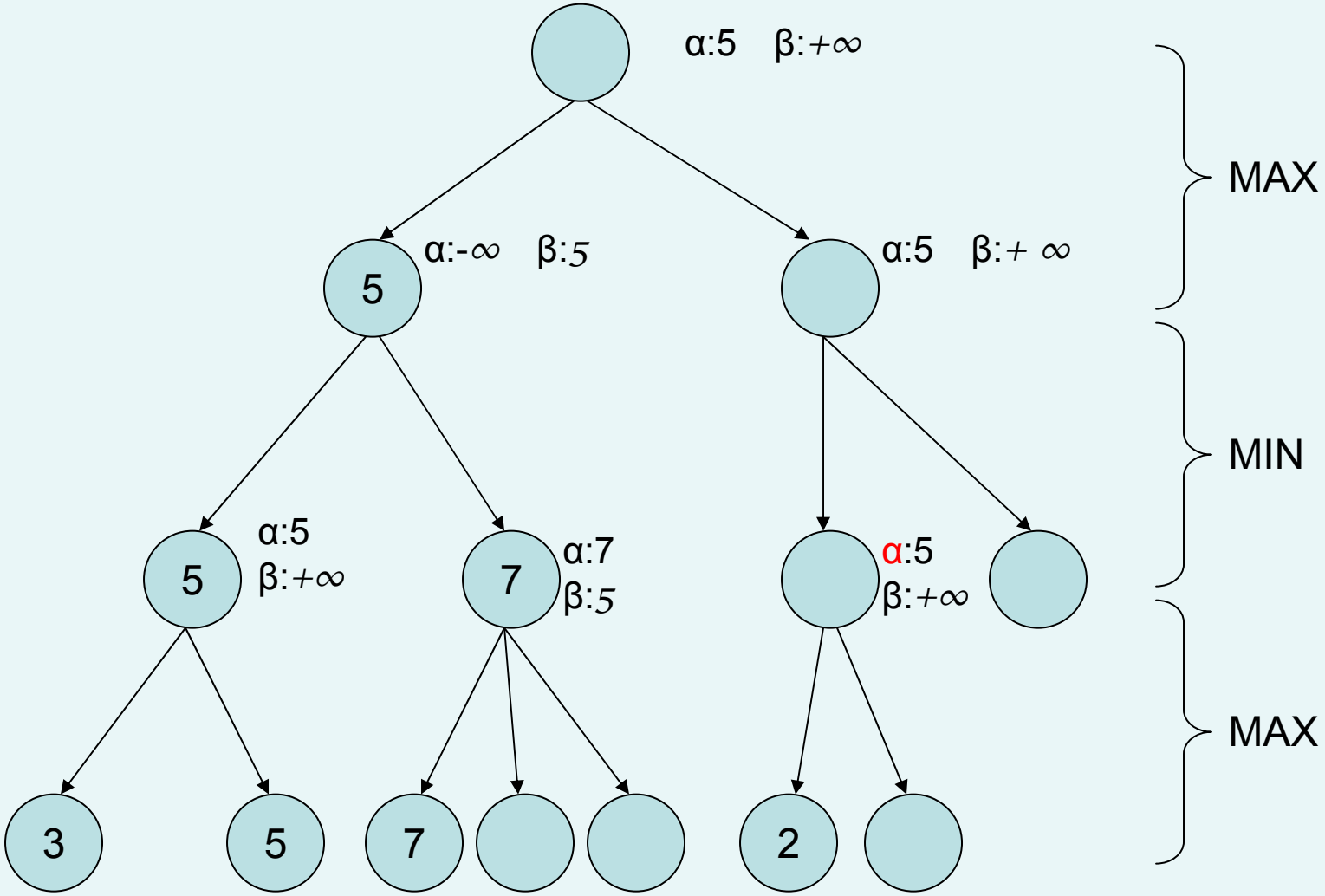
Poda alfa-beta



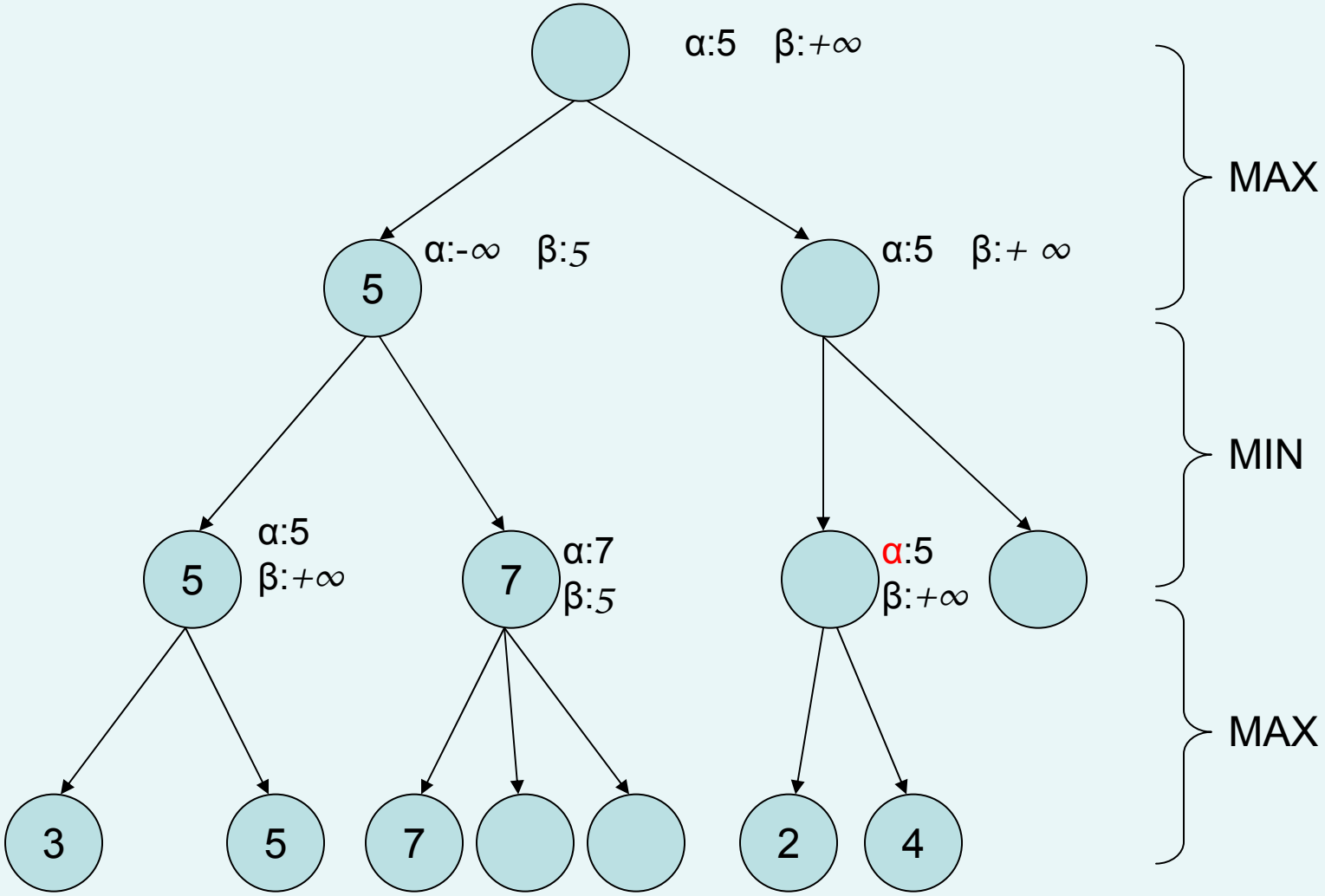
Poda alfa-beta



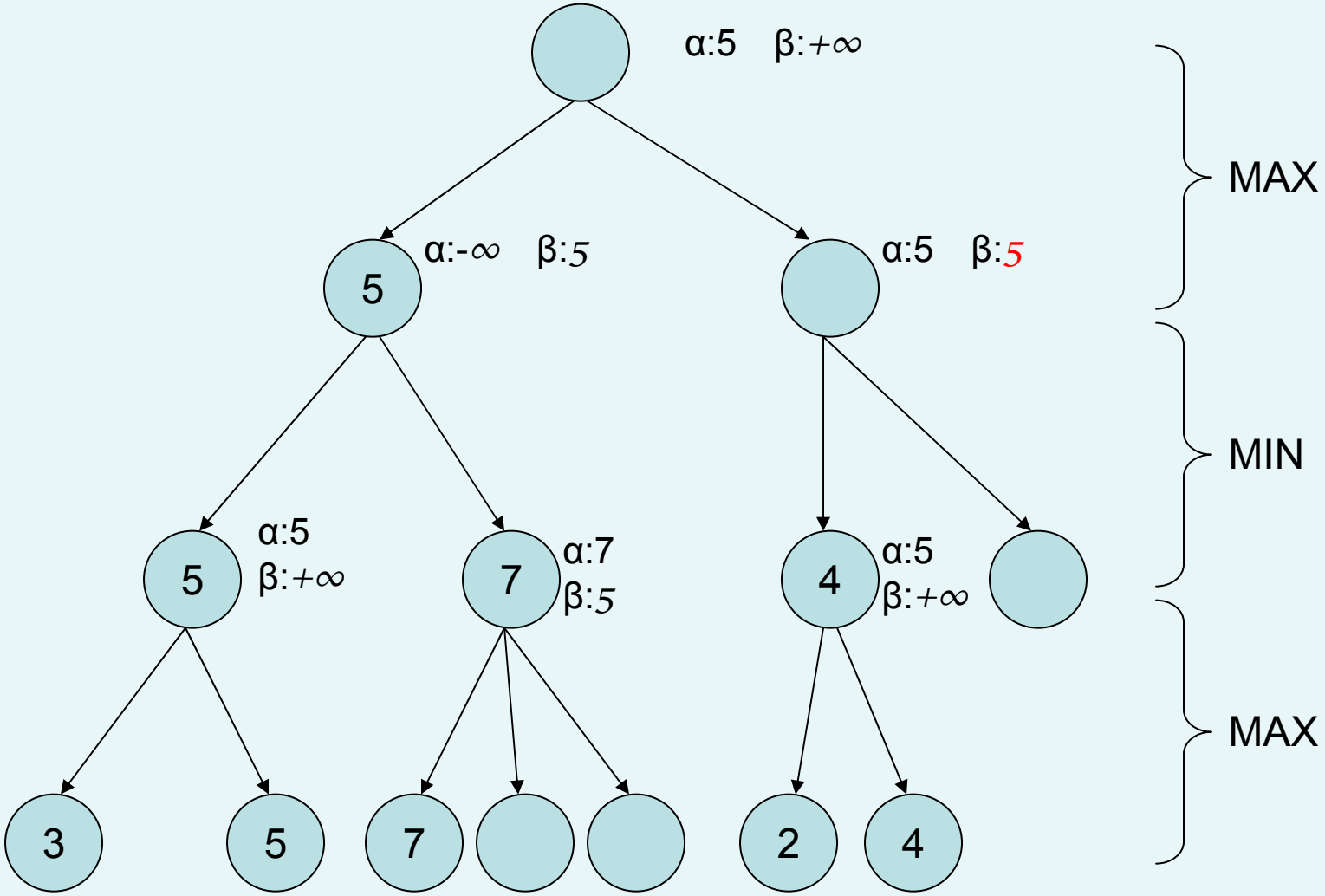
Poda alfa-beta



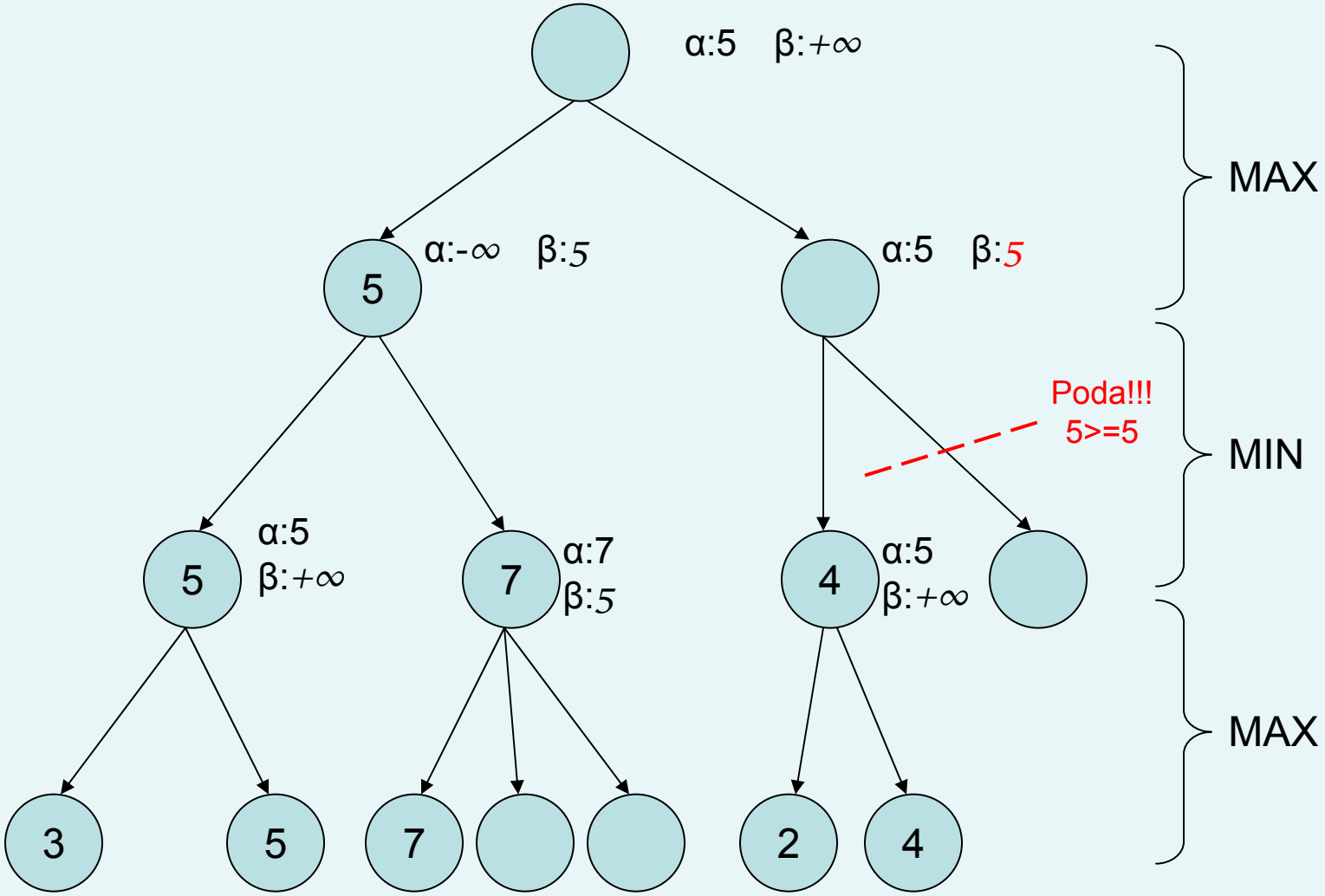
Poda alfa-beta



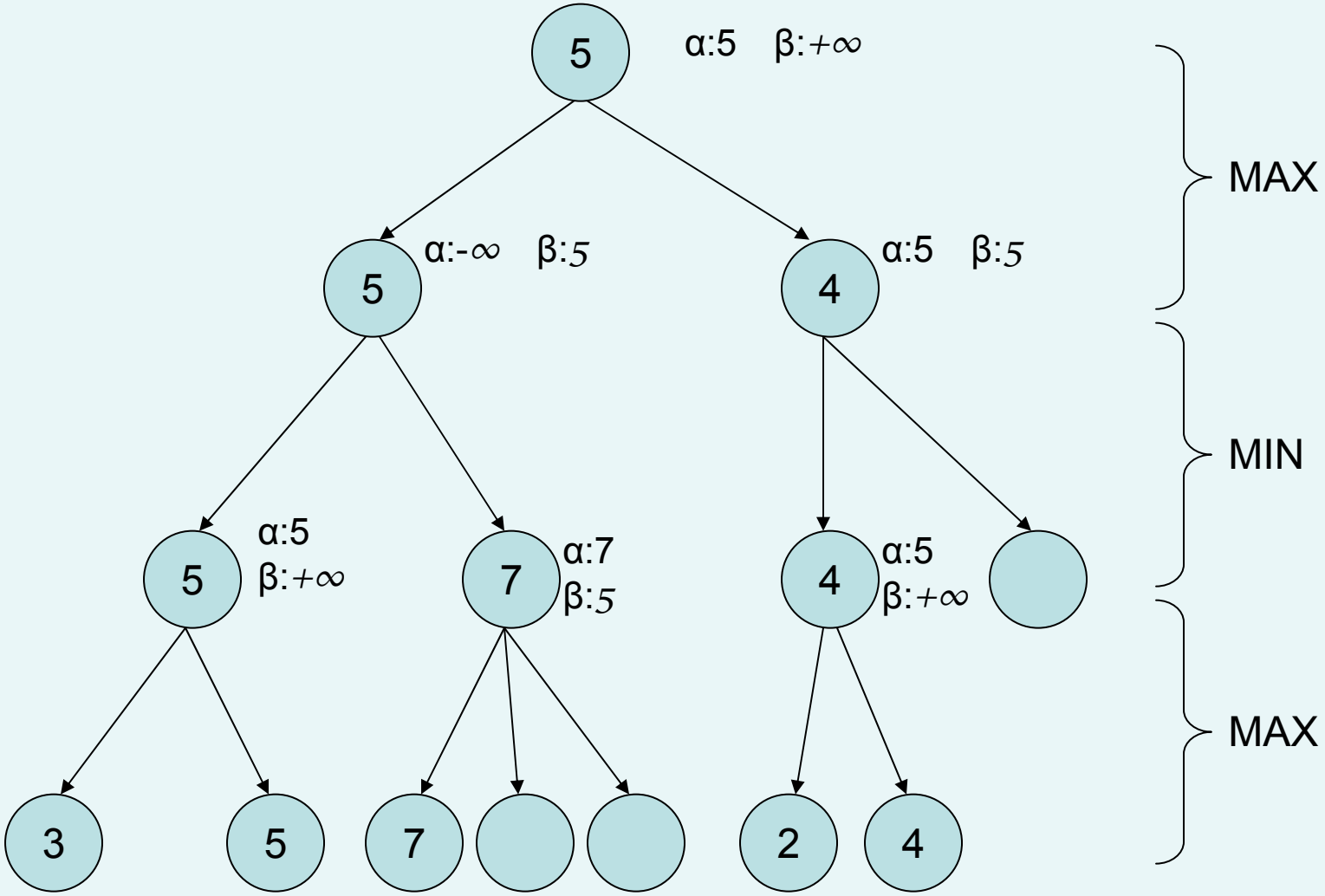
Poda alfa-beta



Poda alfa-beta



Poda alfa-beta



Negamax

Forma mas compacta del codigo de la poda alfa-beta

Idea:
Minimizar una cantidad de valores es equivalente a maximizar sus negativos

MIN (3, 5, 7, 8) = 3 - [MAX (-3, -5, -7, -8)] = - [-3] = 3

Usar el algoritmo de la poda alfa-beta, cambiando a cada nivel:

- el signo
- el rol de alfa y beta

Negamax

```
NEGAMAX (nodo,nivel, alfa, beta) devuelve (valor, nodo_resultado)
    si final(node) entonces   devuelve (+∞ / - ∞ , nodo_vacio)
    sino si  nivel = nivel_maximo_minimax                                  [constante]
            entonces      devuelve      (h(nodo) , nodo_vacio)
            sino    mientras quedan_hijos(nodo) y (alfa<beta) hacer
                        F := siguiente_hijo(nodo)
                        (val , nuevo_nodo) := - NEGAMAX ( F , nivel +1, - alfa, - beta)
                        si val > alfa
                                entonces      alfa := val
                        fsi
                    fmientras
                    devuelve (alfa, nodo_a_devolver)
        fsi
    fsi
```

Iterative Deeping

No tener un numero de niveles máximo, sino un tiempo máximo

Según el árbol a cada momento del juego, podemos mirar mas o menos niveles

Idea:
Ir haciendo el análisis por niveles hasta que el tiempo se acabe

JUEGOS