

2.5. Minimax



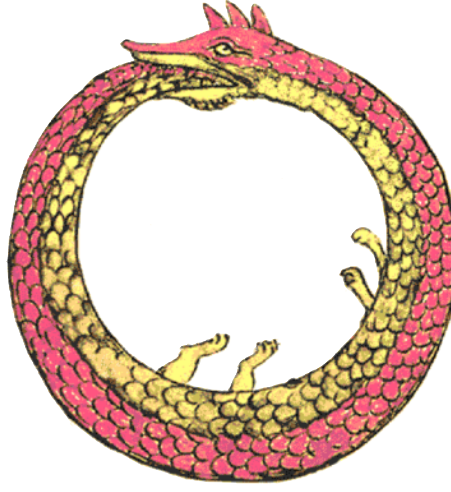
Tic Tac Toe



- Premier jeu sur ordinateur en 1952
- Electronic Delay Storage Automatic Calculator (EDSAC)



But de l'algorithme



Pour un coup donné :

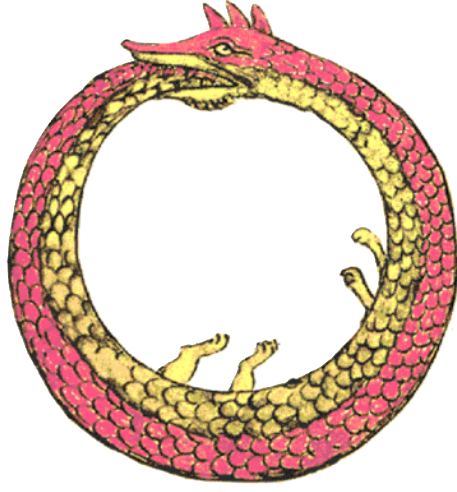
Prédire qui gagnera, si on joue parfaitement.

└ **je** joue parfaitement, puis
 l'autre joue parfaitement, puis
 je joue parfaitement, puis
 l'autre joue parfaitement, puis
 je joue parfaitement, puis

...

Jouer parfaitement ? Jouer le coup qui me fera gagner.

Idée de l'algorithme



Pour *calculer qui gagnera*, pour un coup donné :

Jouer ce coup en tant que **moi**.

Cas trivial

Si ce coup termine la partie (victoire ou match nul), c'est clair

Cas général

Si la partie n'est pas terminée

Pour chaque coup que **l'autre** peut faire

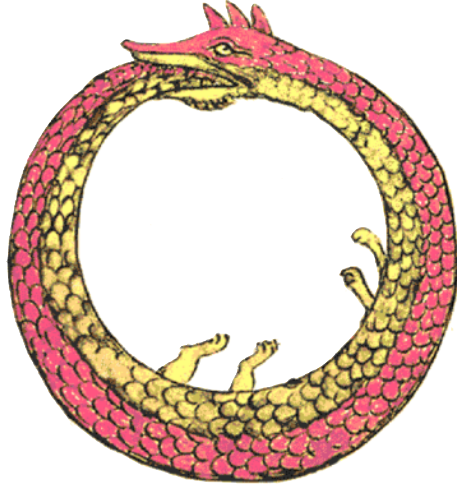
Récur­sif Calculer qui gagne si l'autre joue ce coup.

Noter le coup qui favorise **l'autre**.

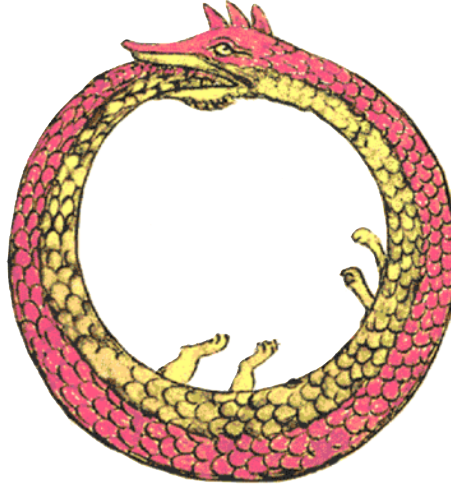
*Simulation du meilleur
coup **de l'autre**.*

Retourner qui gagne si l'autre joue ce coup.

Algorithme récursif



- 2 cas triviaux arrêtent la récursion
 - 3 X ou 3 O sont alignés : victoire
 - 9 cases sont remplies : match nul
- Le cas général :
 - L'adversaire joue le tour qui maximise ses chances, récursivement.
 - Nos objectifs sont opposés. Sa victoire est ma défaite et vice-versa.



Retourne le score final si joueur joue case, puis si tout le monde joue le meilleur coup.

fonction calculeScore(case, joueur)

marquer la case

si la grille est gagnante pour joueur,

alors score $\leftarrow$ +1

sinon si la grille est pleine,

alors score $\leftarrow$ 0

sinon

scoreAdverse $\leftarrow$ $-\infty$

pour toute case vide c

scoreAdverse $\leftarrow$ max(scoreAdverse,
calculeScore(c, adversaire))

fin pour

score $\leftarrow$ -1 * scoreAdverse

fin si

effacer la marque dans la case

retourner score

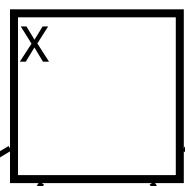
Cas triviaux

Cas général

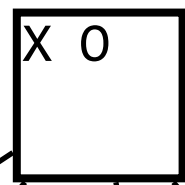
*(Pourquoi ?
Voir permutations)*

*Trouver le meilleur
coup de l'autre.*

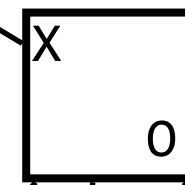
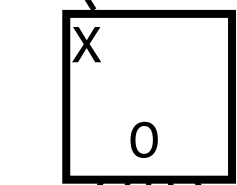
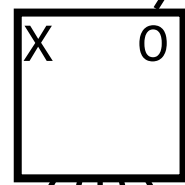
Inversion des objectifs



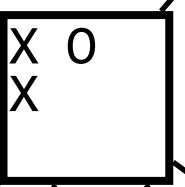
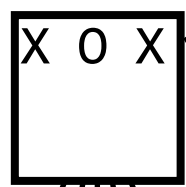
son
coup



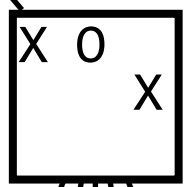
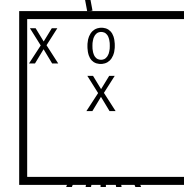
• • •



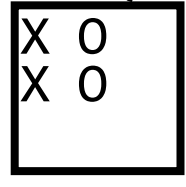
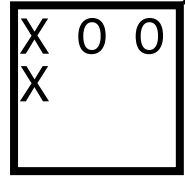
*mon
coup*



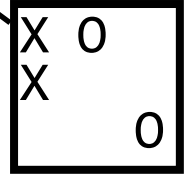
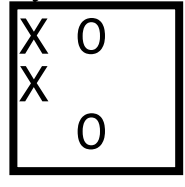
• • •



son
coup



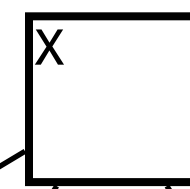
• • •



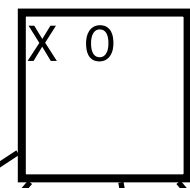
• • •

*Garder le coup qui maximise **mon** score*

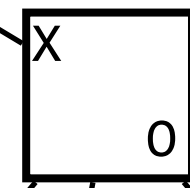
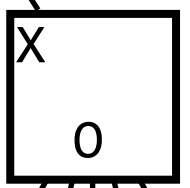
Garder le coup qui



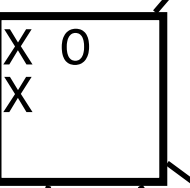
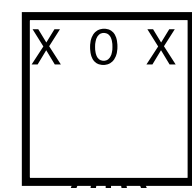
son
coup



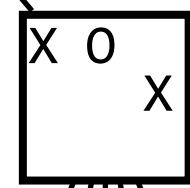
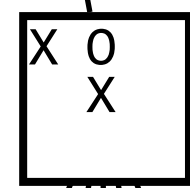
• • •



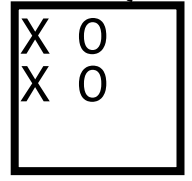
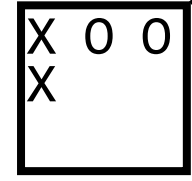
*mon
coup*



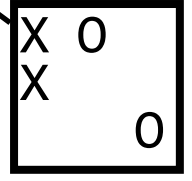
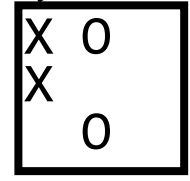
• • •



son
coup



• • •



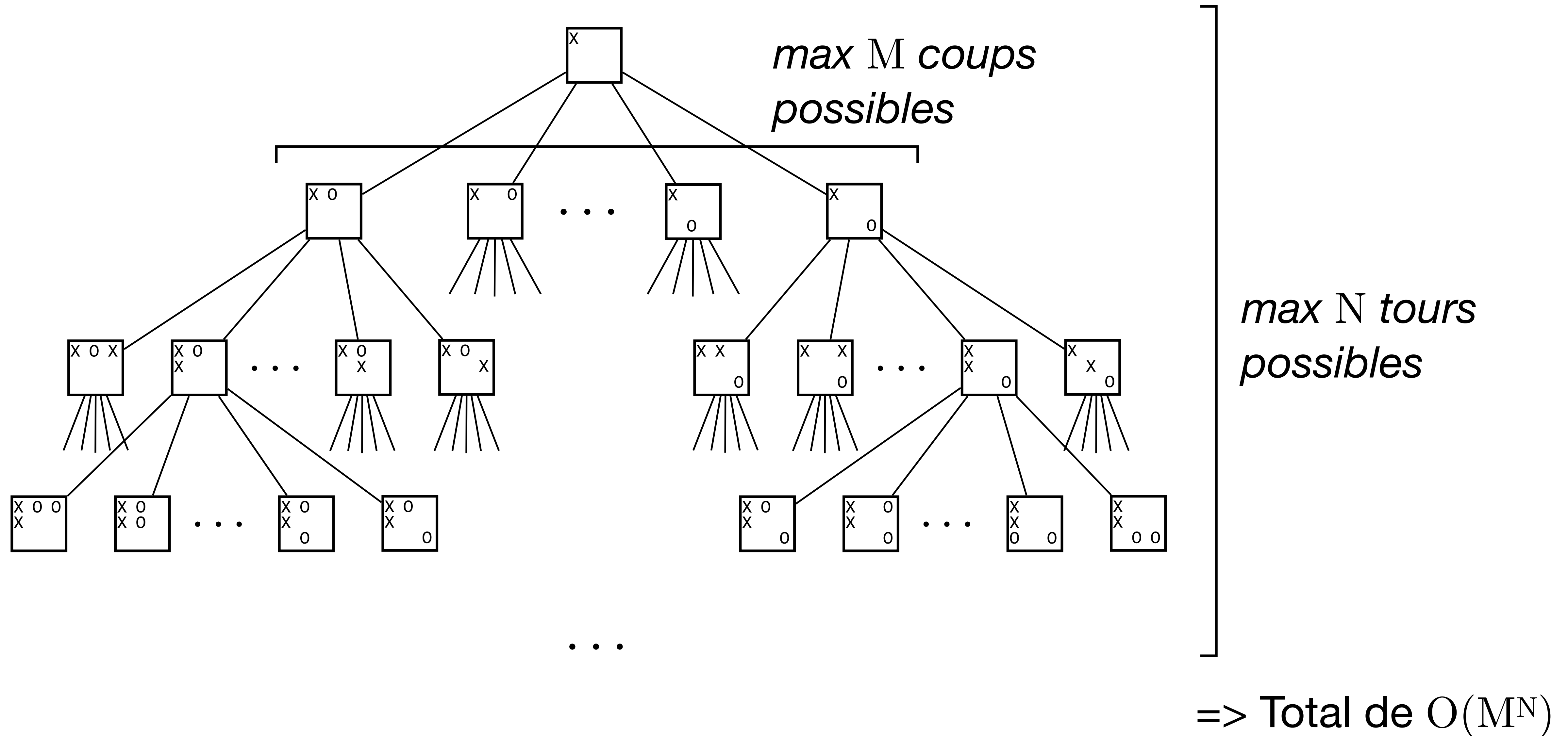
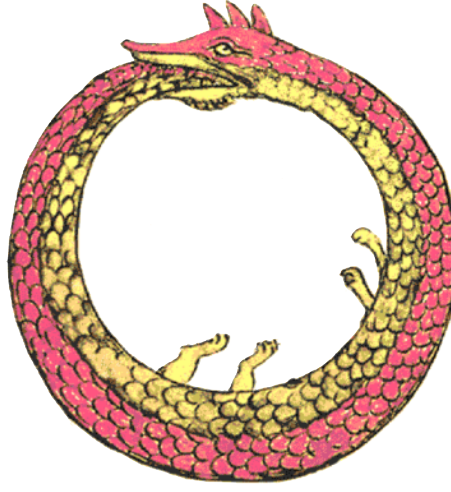
...

Garder le coup qui
maximise *mon score*

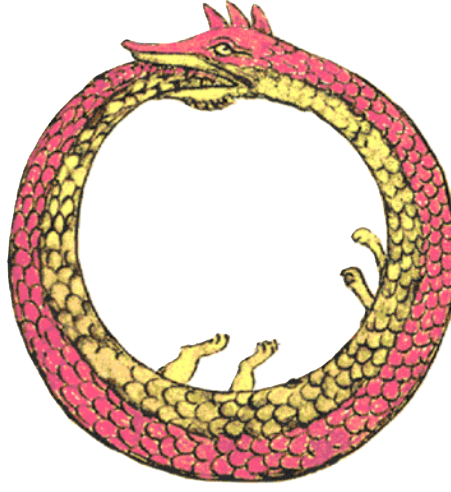
Garder le coup qui



Complexité ?

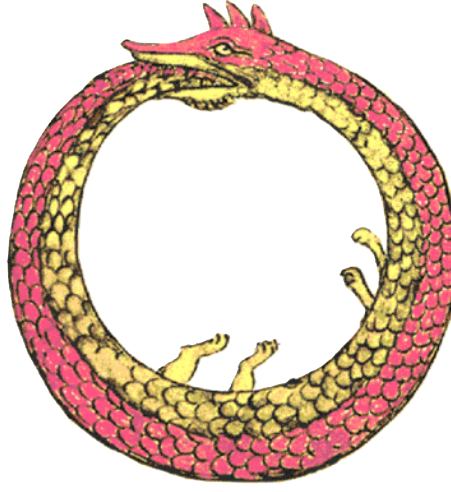


Complexité de ce type d'algorithme ?



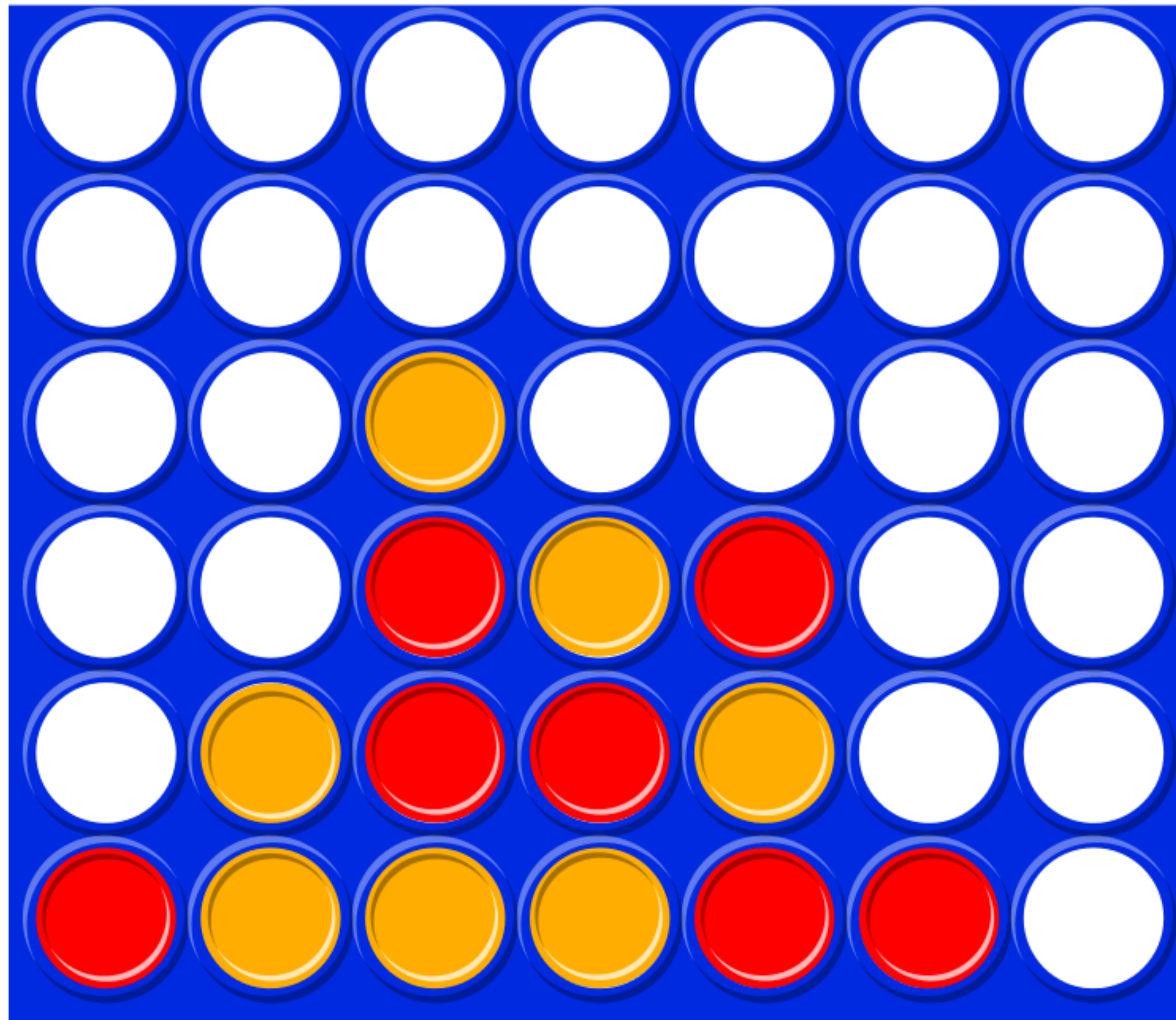
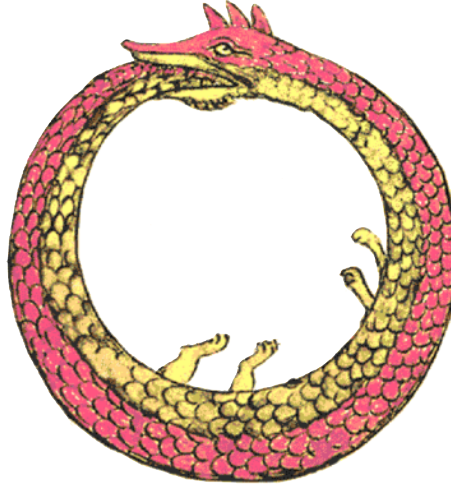
- Le joueur peut choisir parmi au plus M coups, chacun requérant un appel récursif.
- On explore jusqu'à N (demi-)coups successifs
- il y aura $O(M^N)$ appels récursifs.
- M et N dépendent
 - des règles du jeu
 - de la position dans le jeu.

Tic Tac Toe



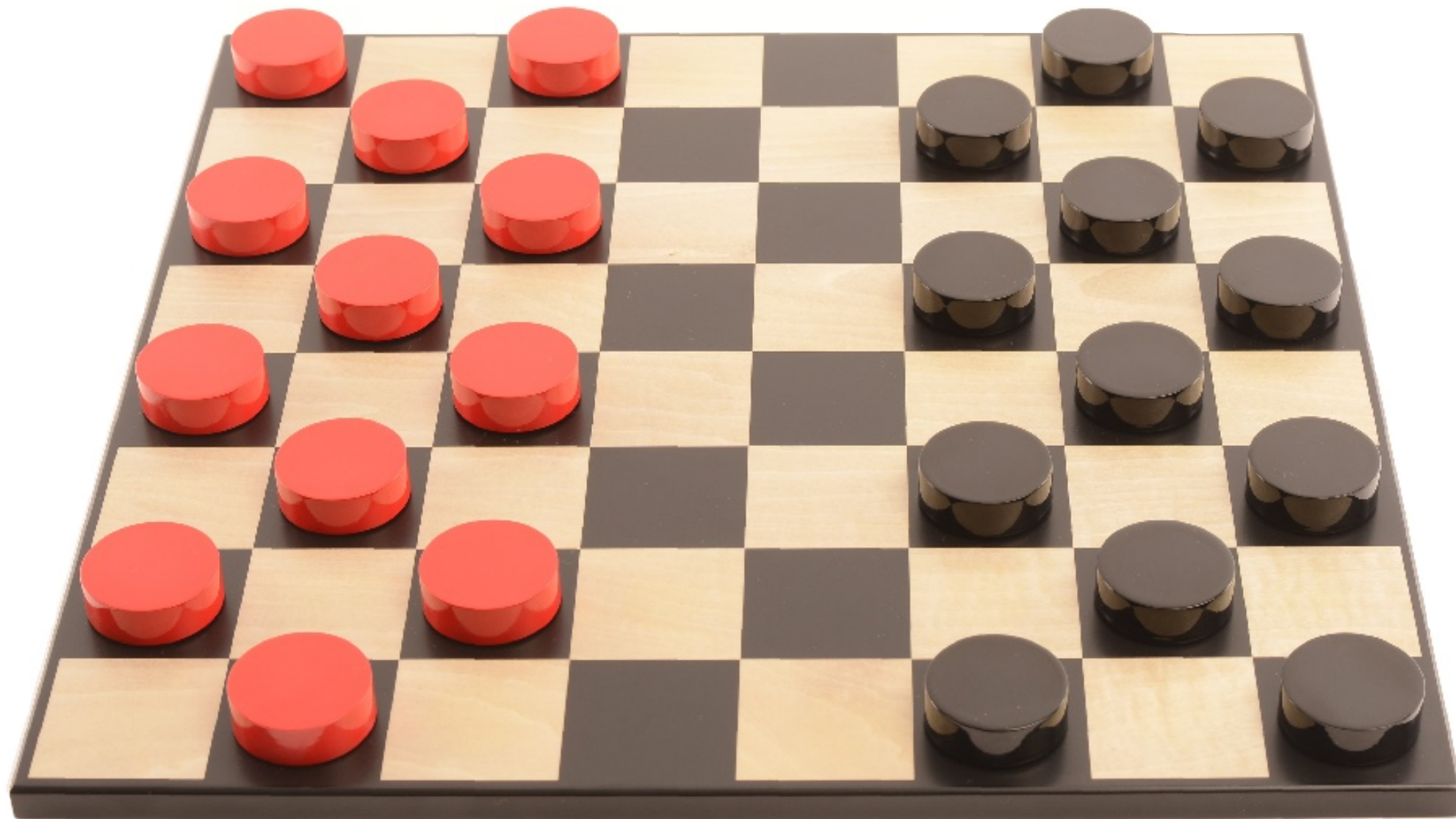
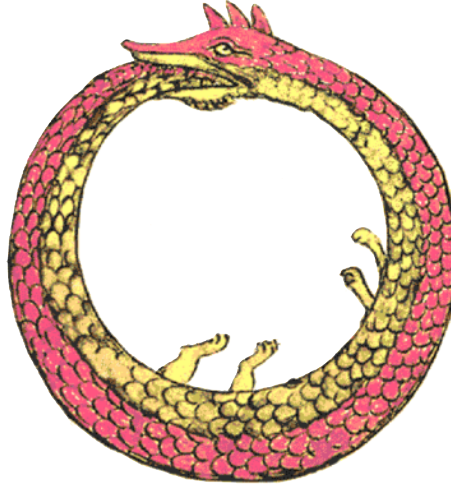
- $9! = 362880$ manières différentes de remplir la grille.
- Seulement 255168 parties possibles.
 - 131184 victoires de X
 - 77904 victoires de O
 - 46080 égalités
- Toujours égalité si les deux joueurs jouent parfaitement

Puissance 4



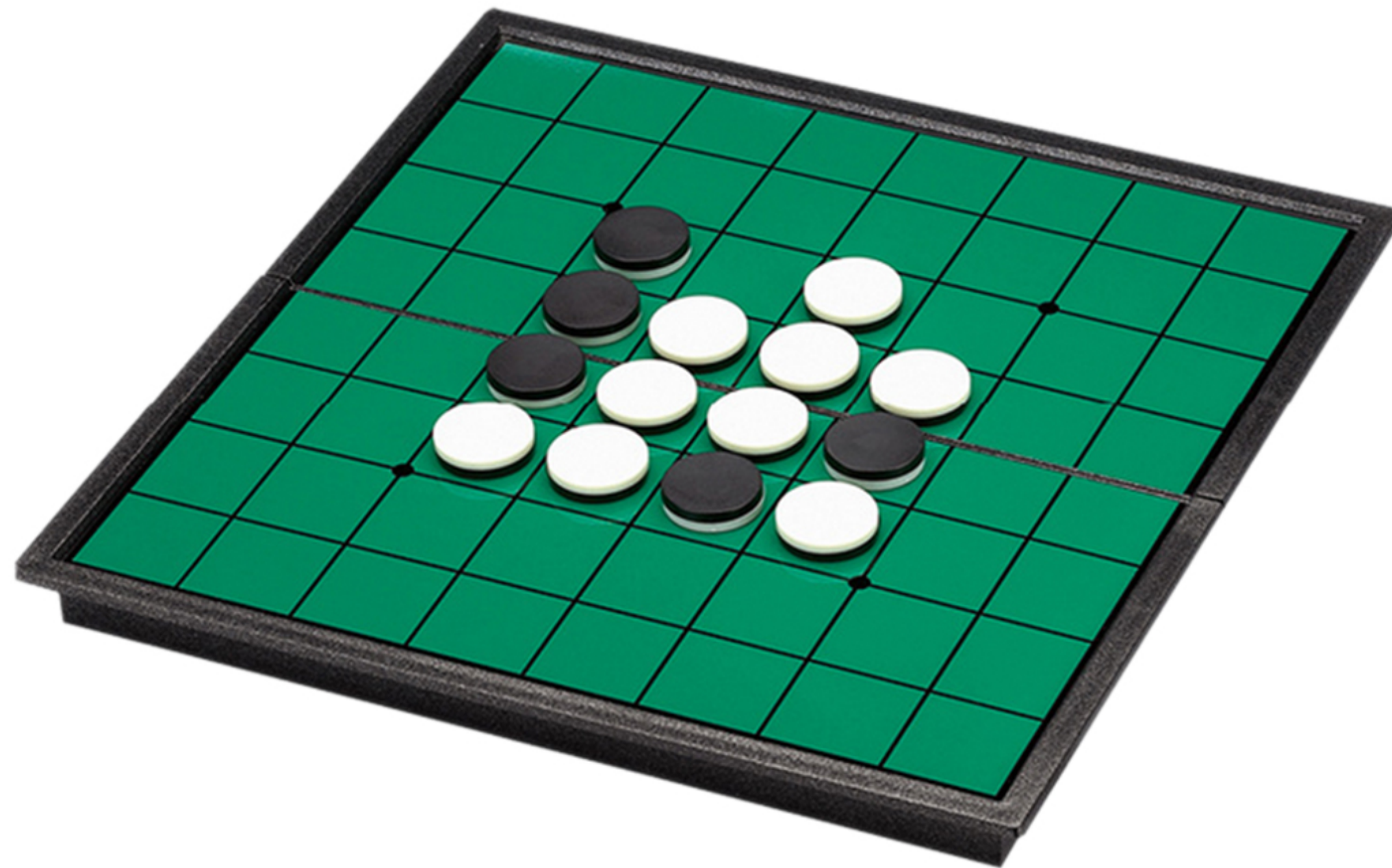
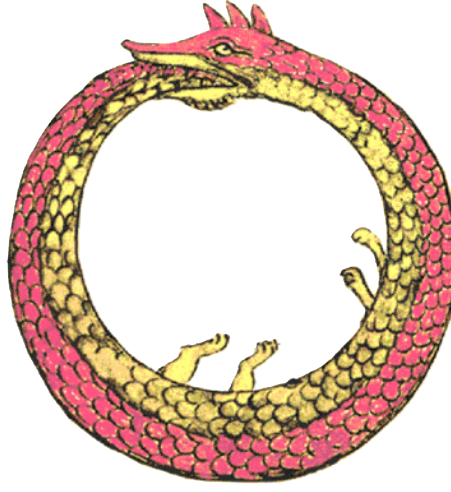
- Au début, 7 colonnes, donc $O(7^n)$ pour n coups.
- Résolu par James D. Allen en 1988.
- 4'531'985'219'092 parties possibles.
- Joueur 1 gagne au 41ème coup s'il joue parfaitement en entamant dans la colonne centrale.
- Egalité s'il entame dans une colonne entourant le centre.
- Joueur 2 gagne sinon

Jeu de dames



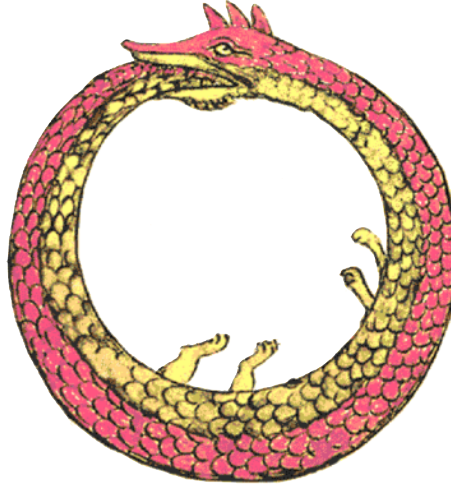
- 5×10^{20} positions possibles sur grille 8x8
- Le programme Chinook gagne le championnat du monde en 1994 face aux humains.
- Résolu par Schaeffer et al. en 2007. Environ 18 ans de calcul non-stop
- Égalité si les deux joueurs jouent parfaitement

Reversi



- Pas de solution complète ni de stratégie parfaite
- On dispose d'heuristiques suffisamment bonnes pour jouer aussi bien ou mieux que des humains
- En 1997, le logiciel Logistello gagne 6 victoires à 0 face au champion du monde humain Takeshi Murakami

Jeu d'échecs



- En 1997, Deep Blue - qui tourne sur un super ordinateur avec du hardware dédié aux échecs - bat Gary Kasparov, champion du monde humain
- En 2003, Deep Junior et X3D Fritz - qui tournent sur des PC - font tous deux match nul face à Kasparov

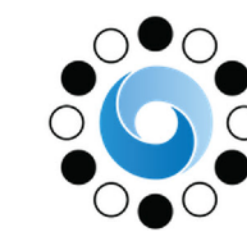
Jeu de Go



- Longtemps considéré comme hors d'atteinte des ordinateurs de par le nombre de combinaisons possibles: 10^{170} .
- En 2016 le logiciel AlphaGo bat le champion du monde Lee Sedol



Humankind
vs
Computer



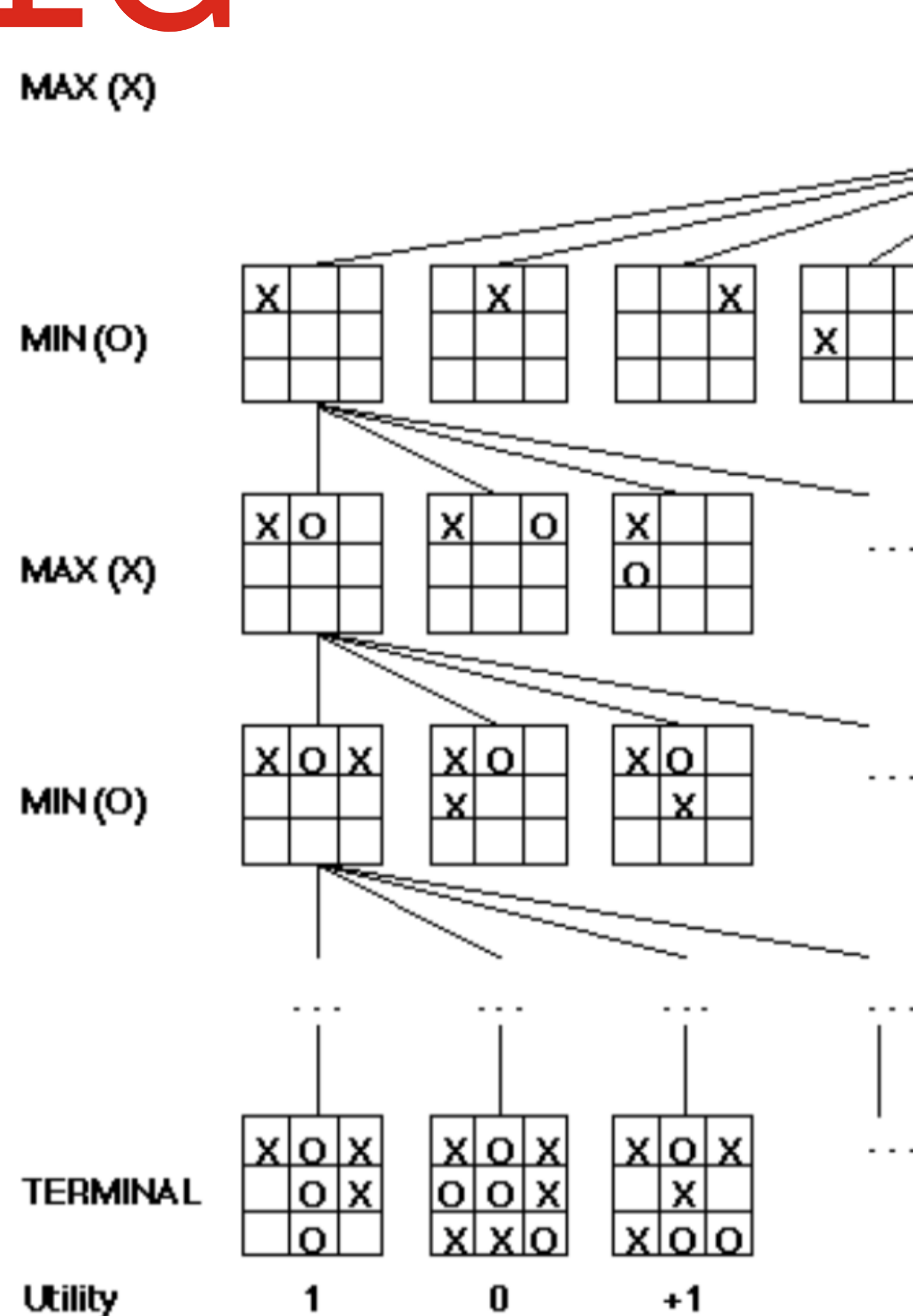
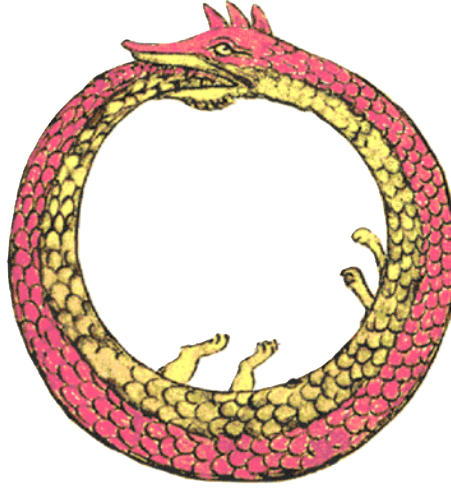
AlphaGo vs Lee Sedol

人類と人工知能の叡智をかけた
史上最強の五番勝負が始まる。
勝つのは、李世乭か？ Googleか？

賞金100万ドル

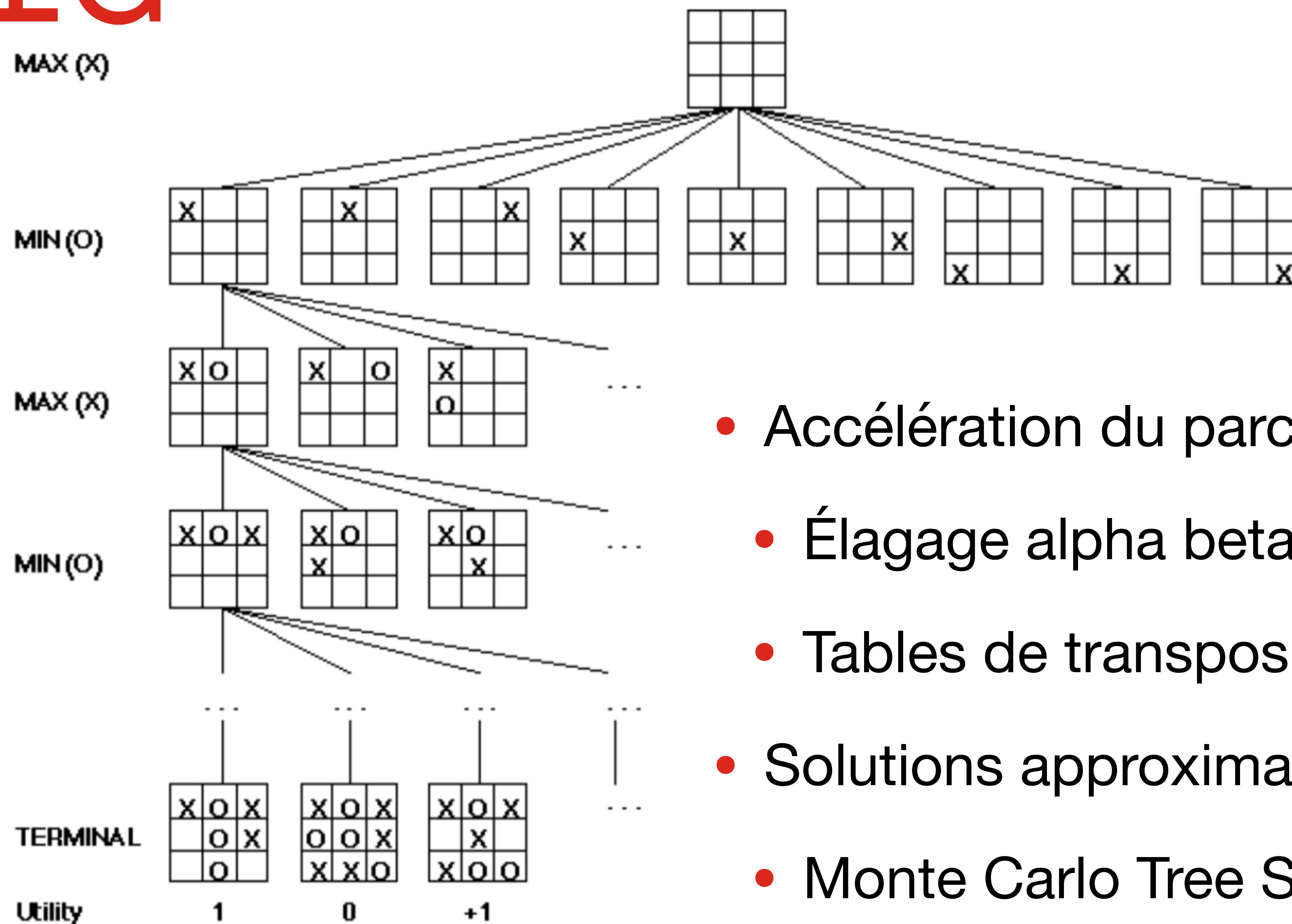
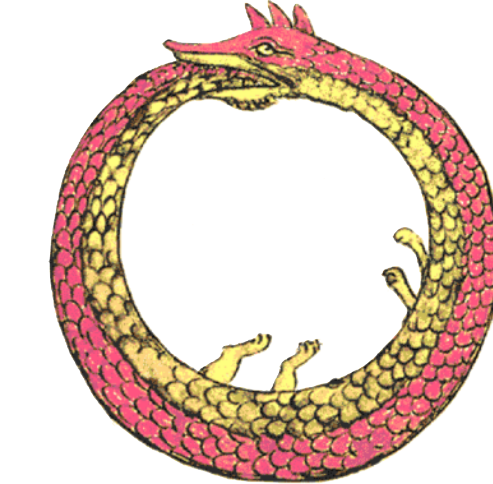
対局日程：3月9、10、12、13、15日／対局場：韓国ソウル

Comment jouent-ils si bien ?



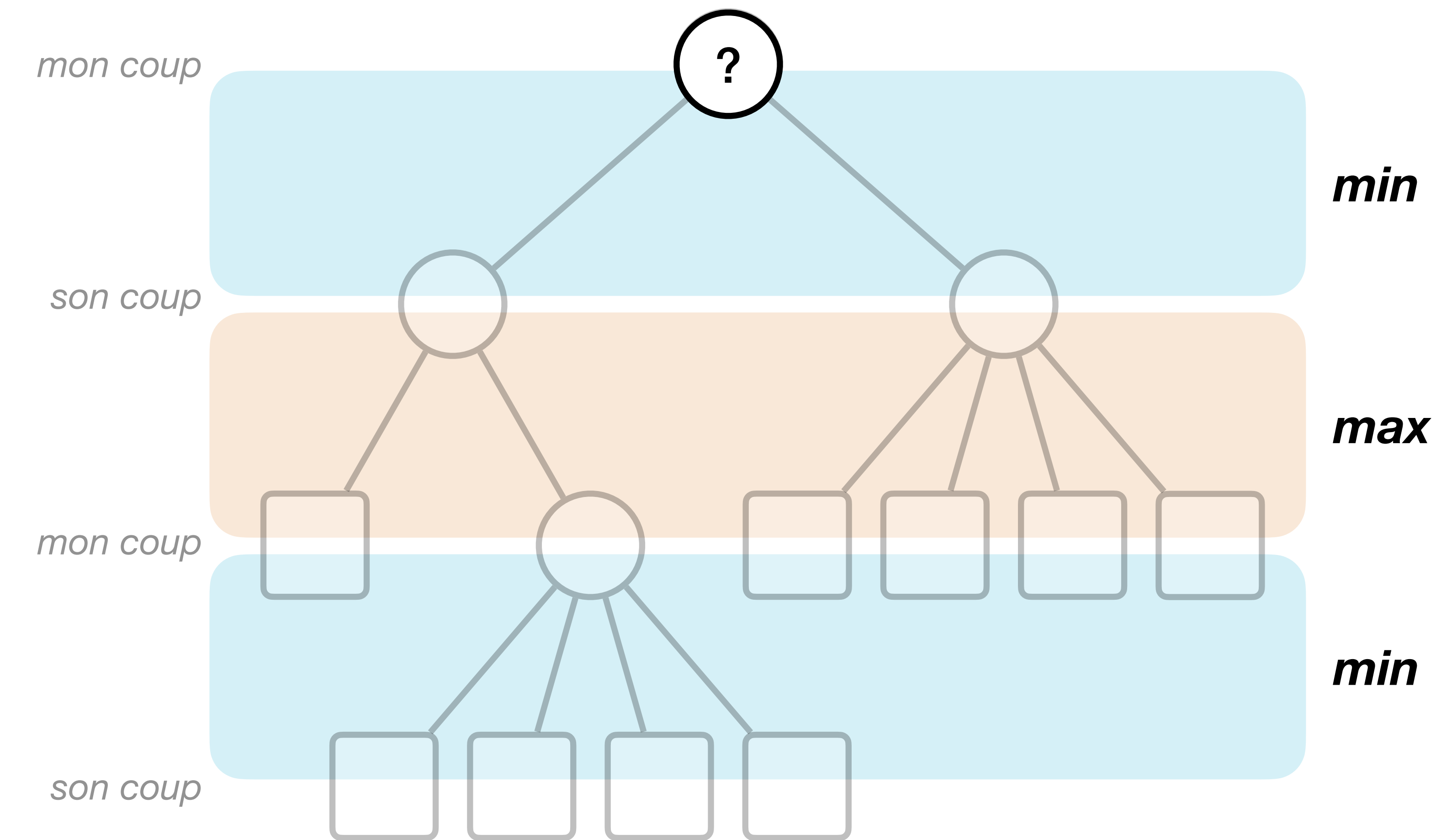
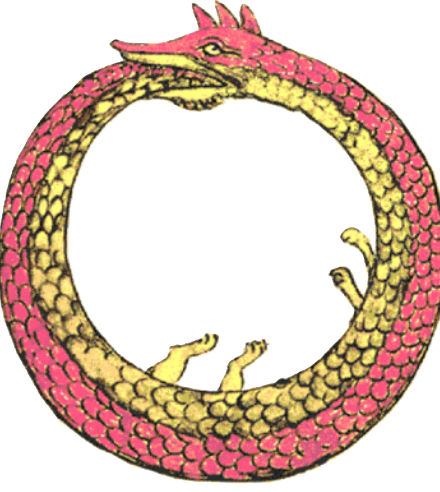
- Variantes de l'algorithme MiniMax
- Exploration limitée à une profondeur maximale
- Heuristique pour évaluer les positions non concluantes.
- Basée sur des stratégies connues
- Apprise (TD-Gammon)

Comment jouent-ils si bien ? (2)

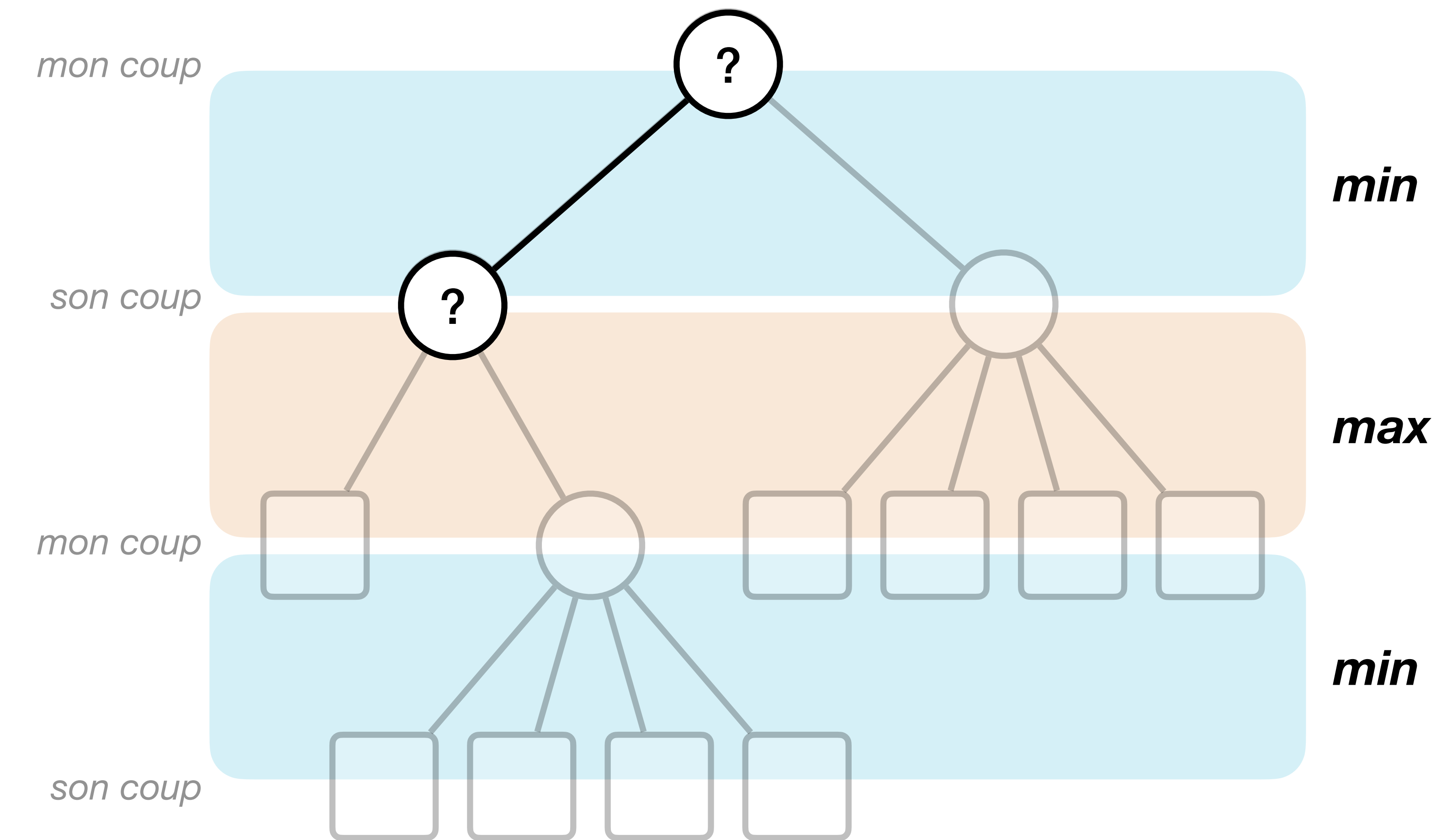
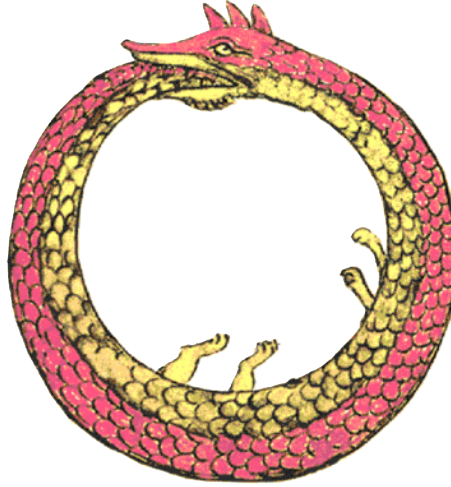


- Accélération du parcours exact de l'arbre
- Élagage alpha beta
- Tables de transposition
- Solutions approximatives
- Monte Carlo Tree Search (AlphaGo)

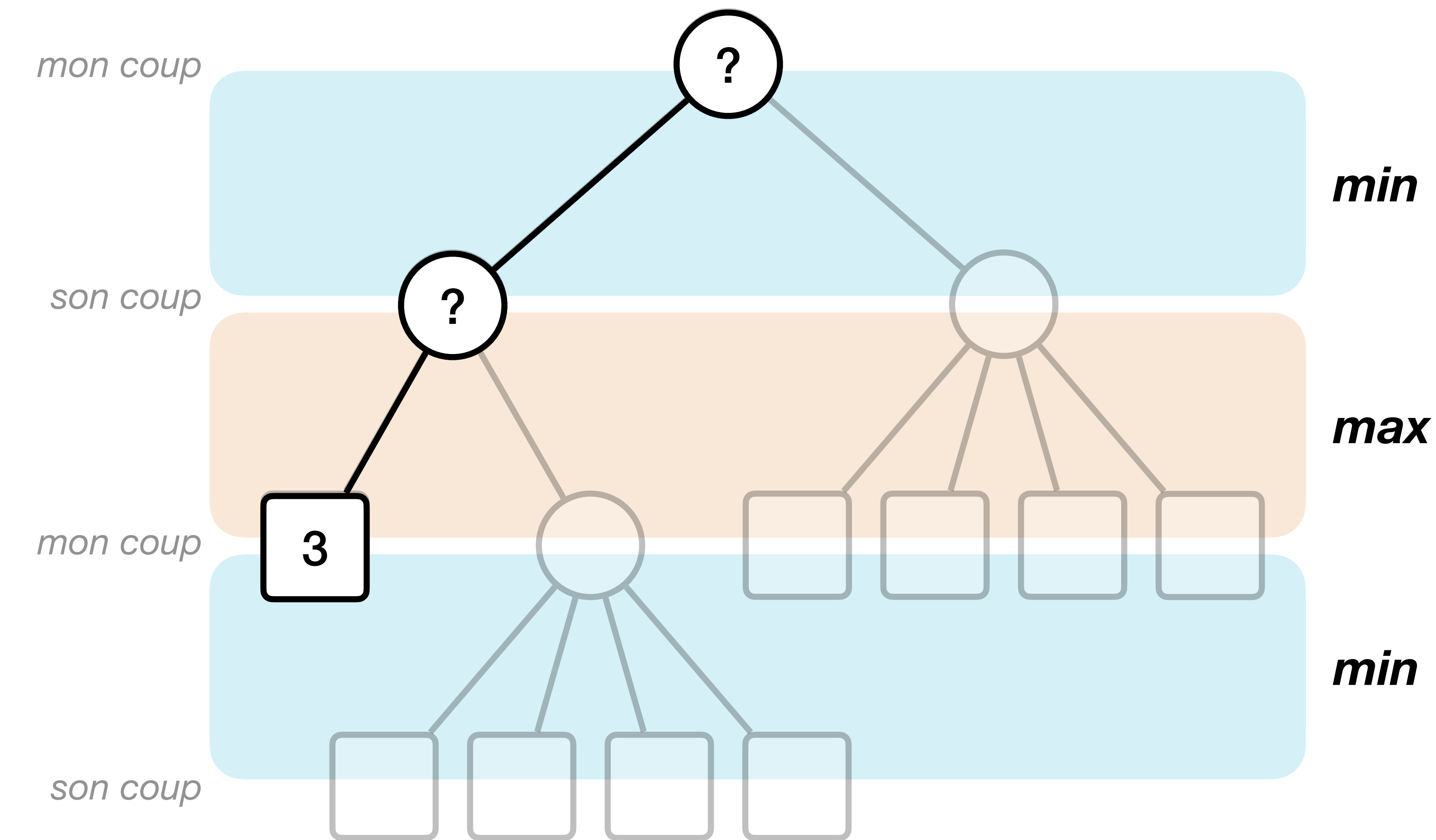
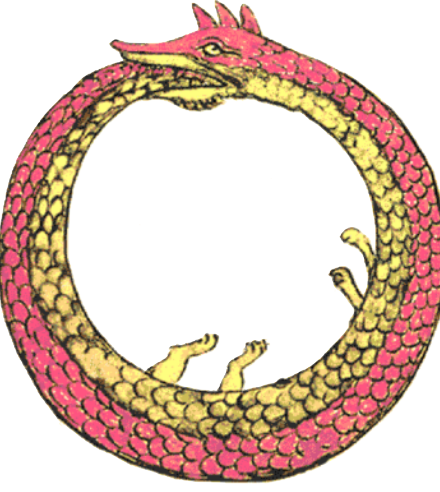
Elagage alpha-beta



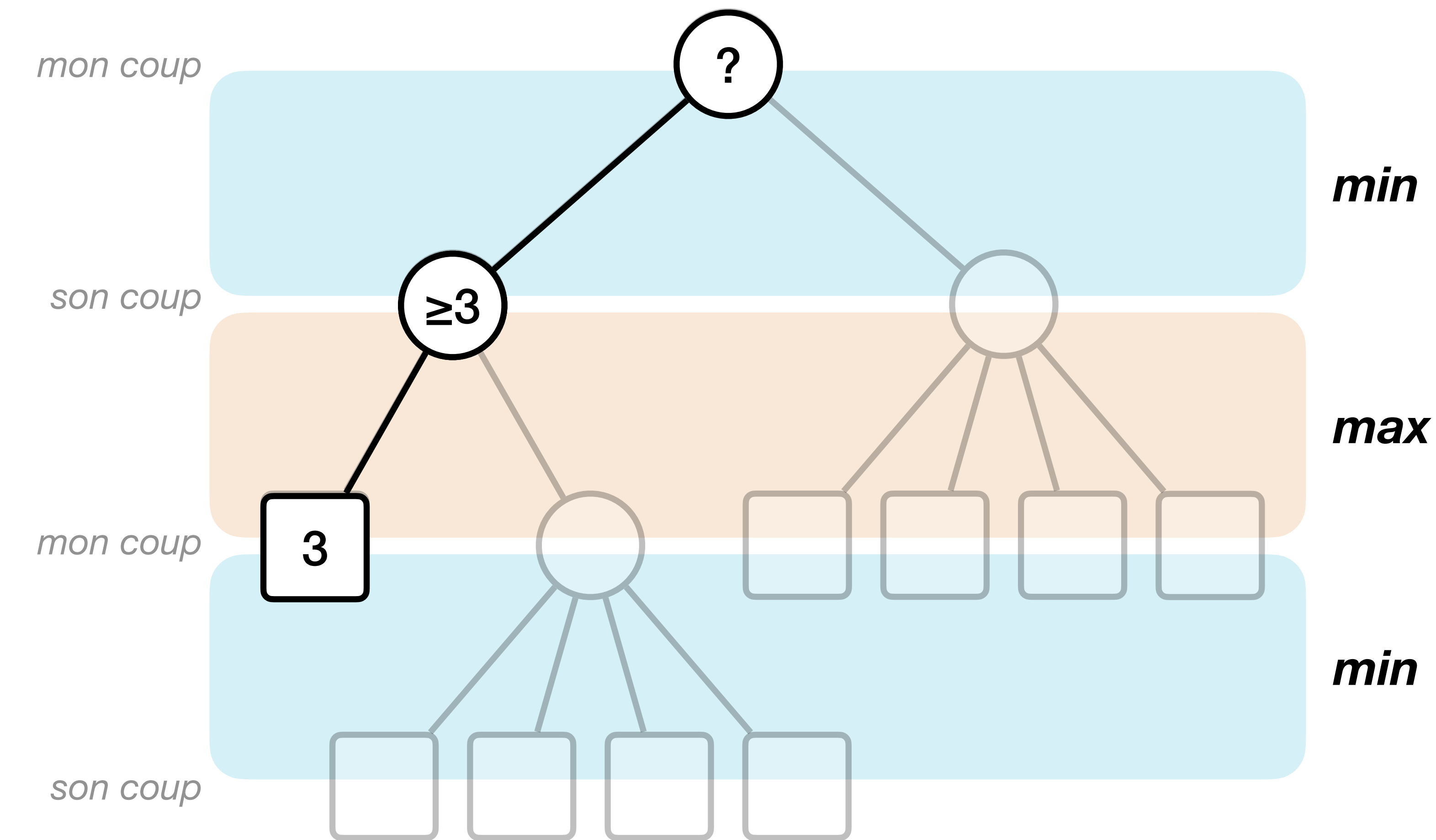
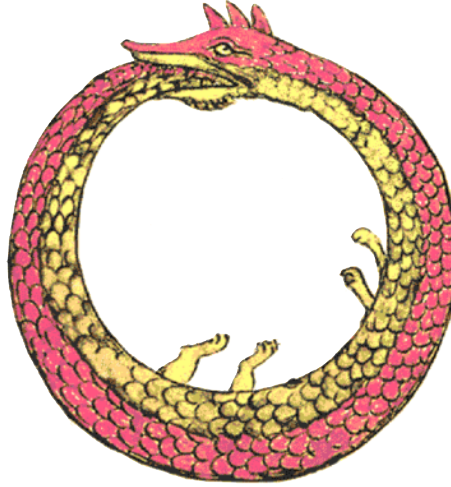
Elagage alpha-beta



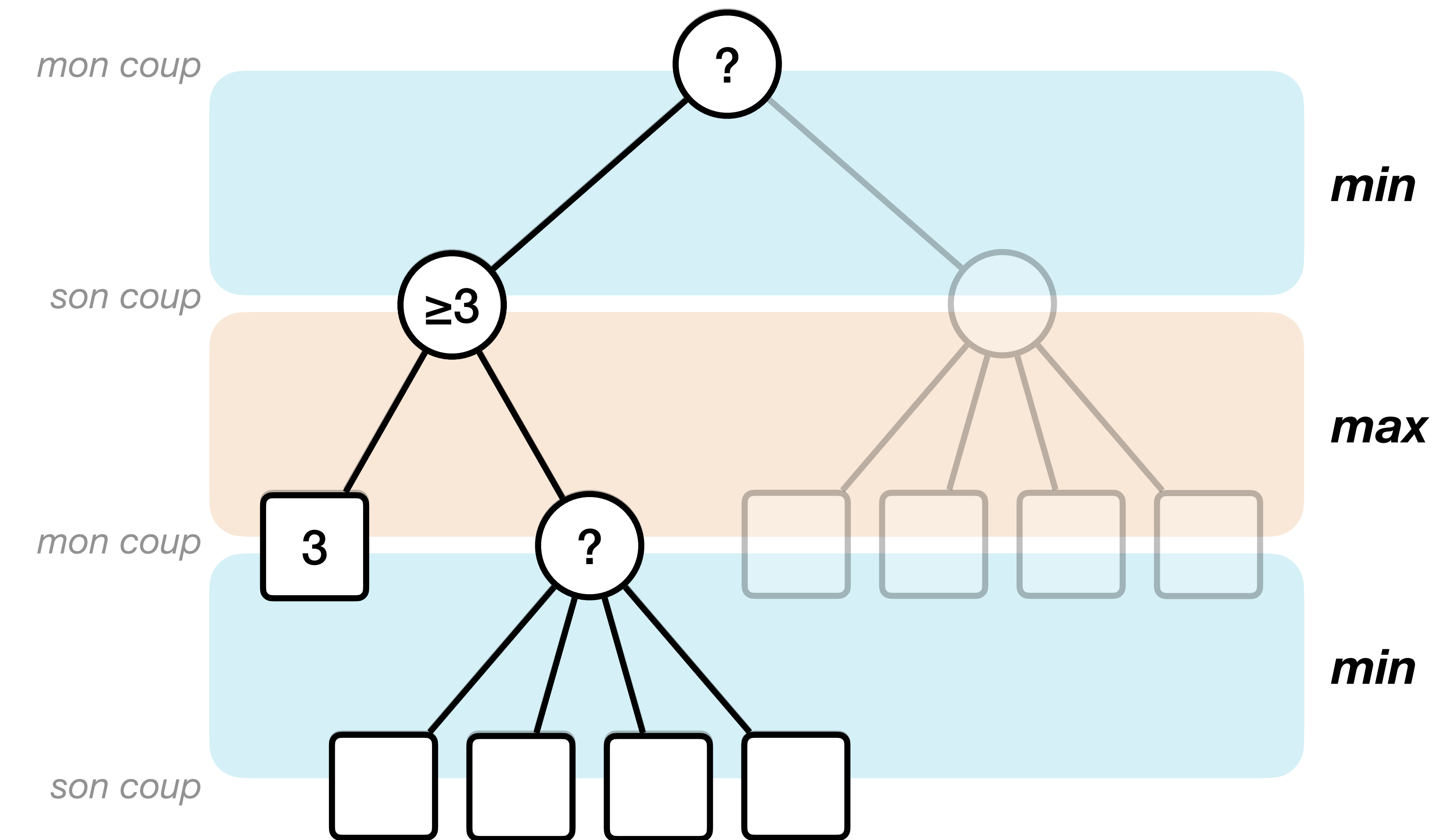
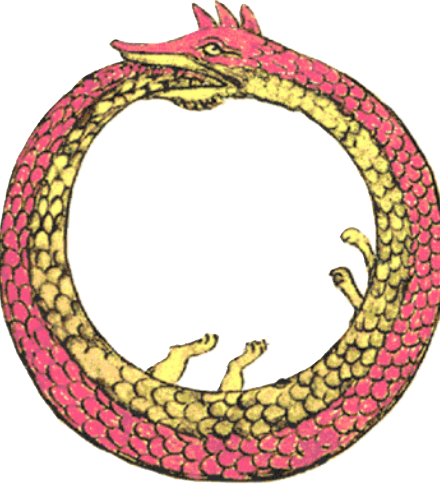
Elagage alpha-beta



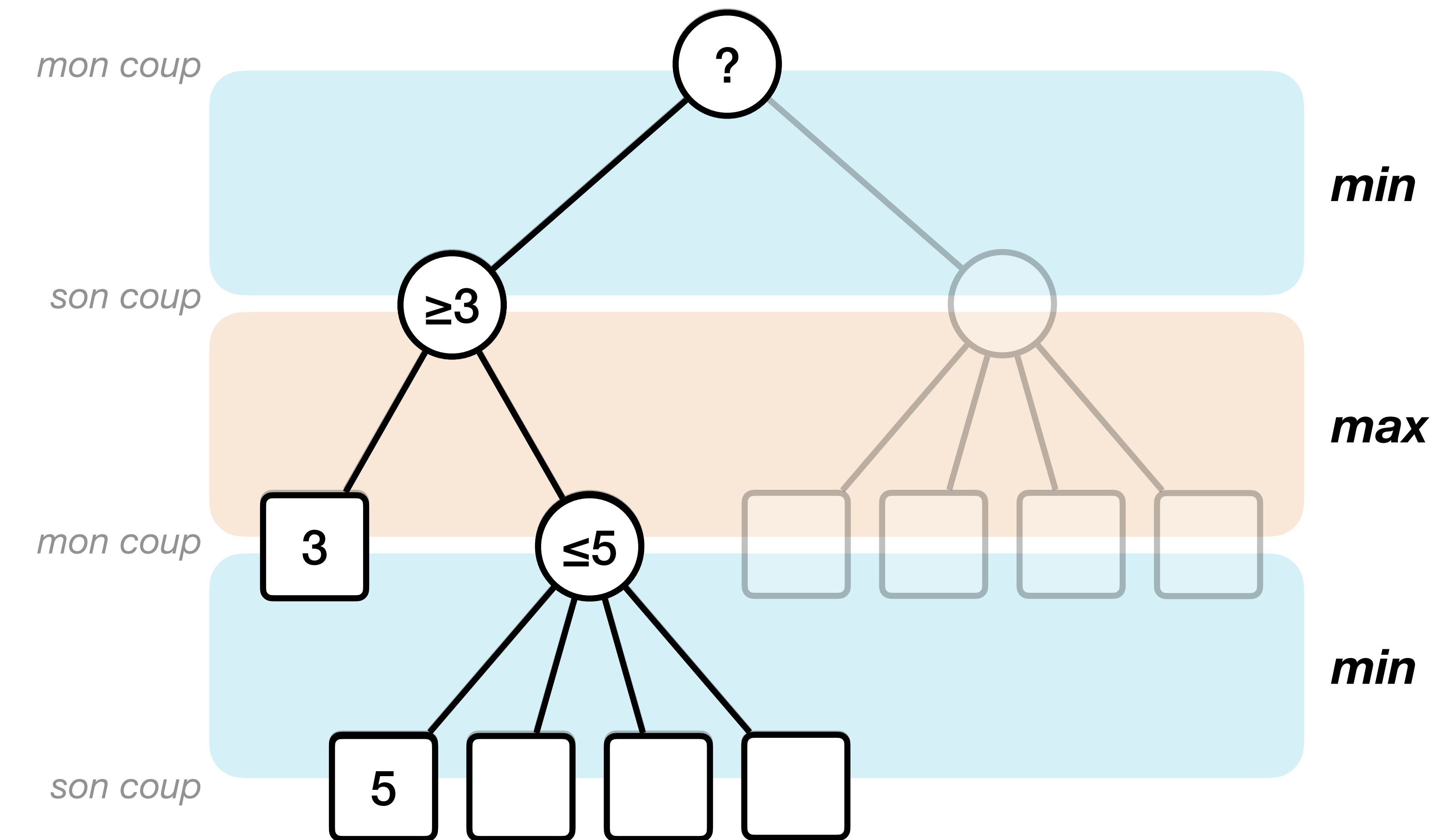
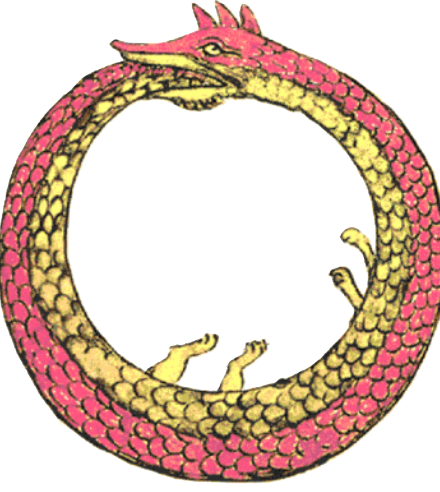
Elagage alpha-beta



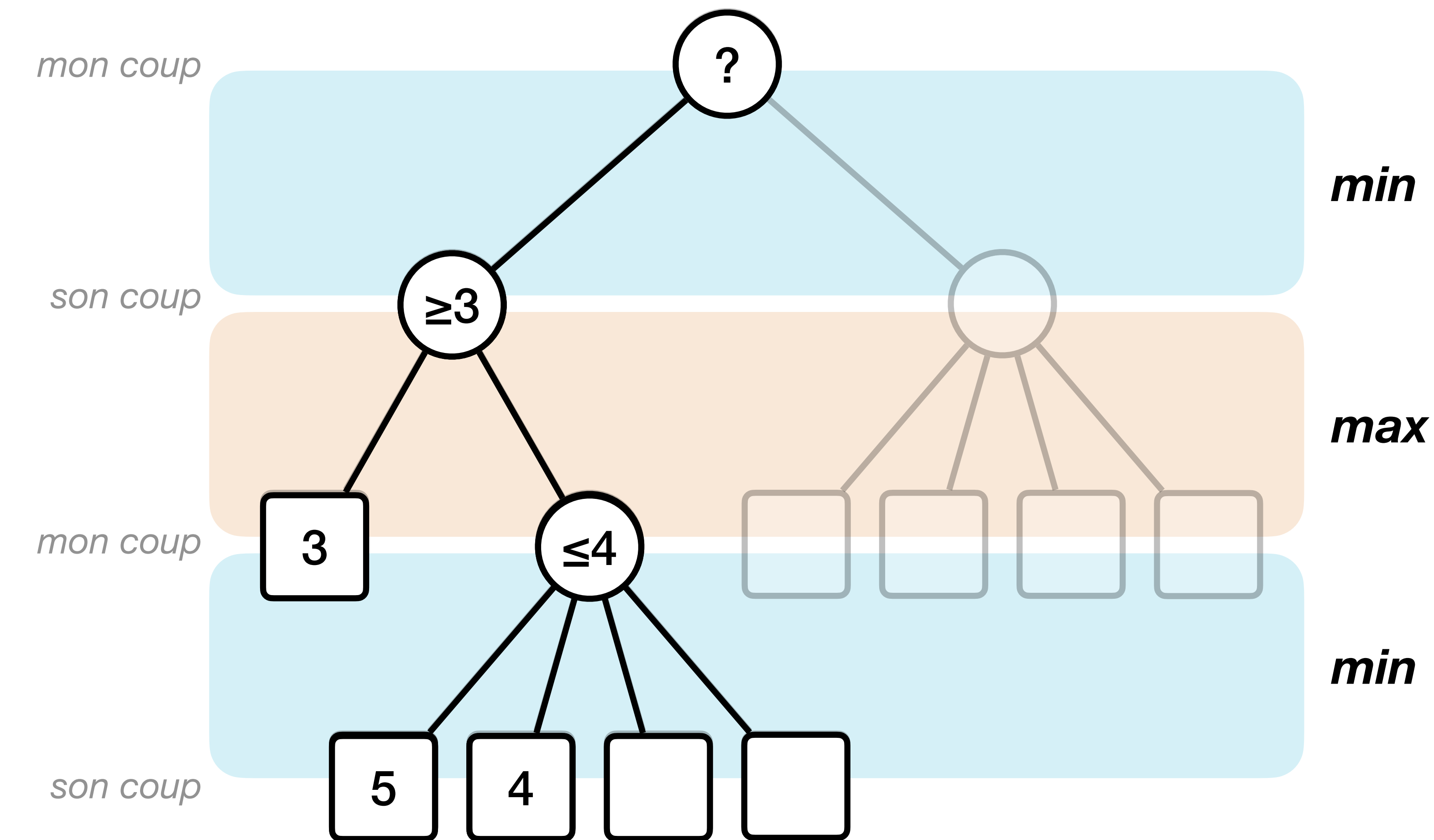
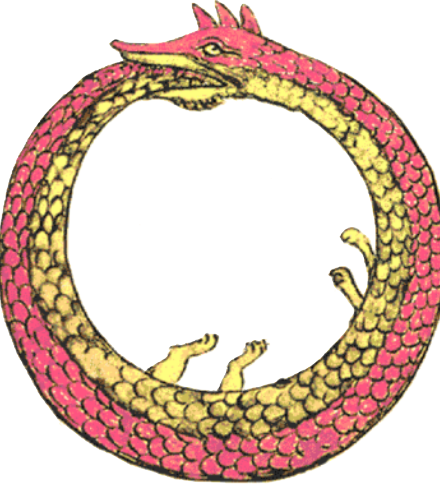
Elagage alpha-beta



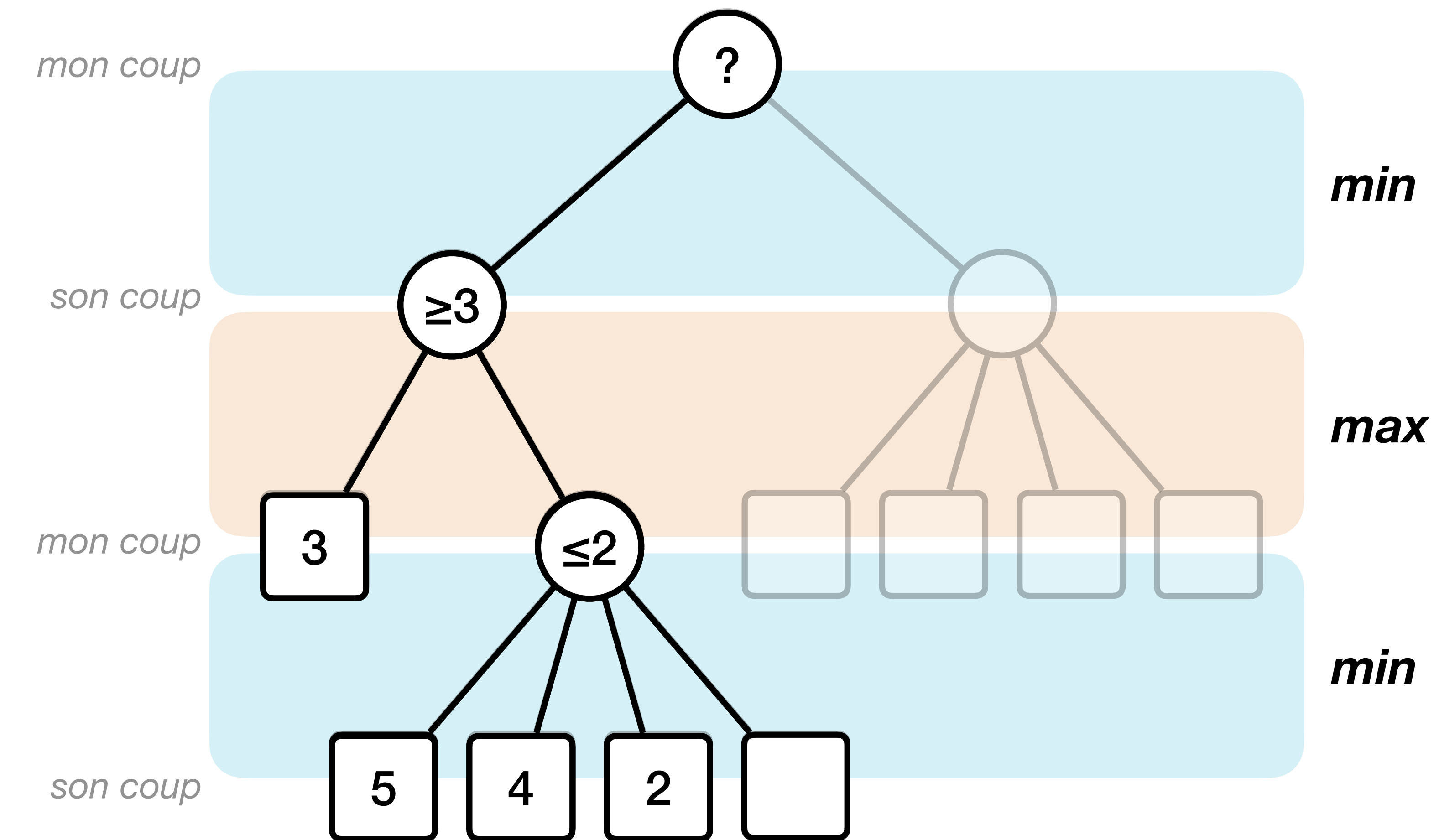
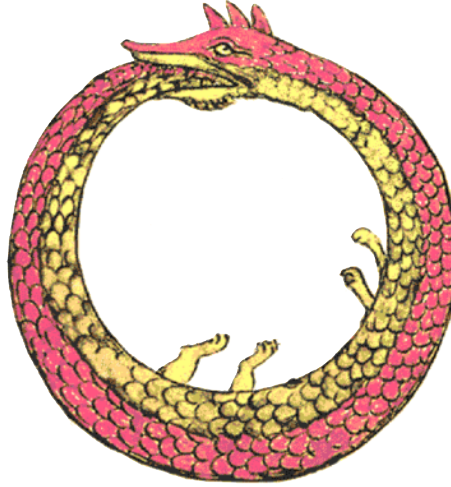
Elagage alpha-beta



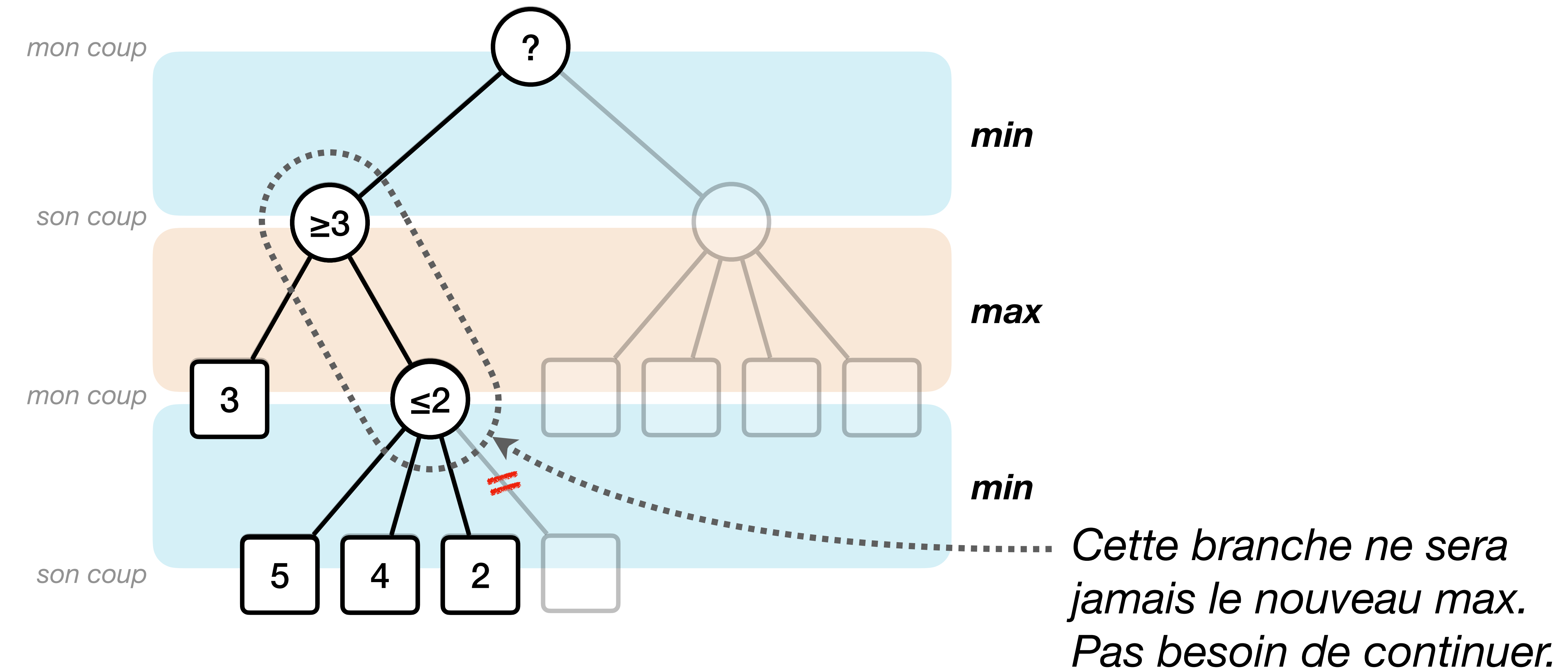
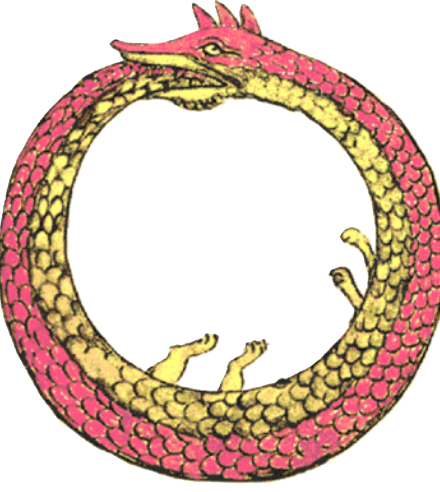
Elagage alpha-beta



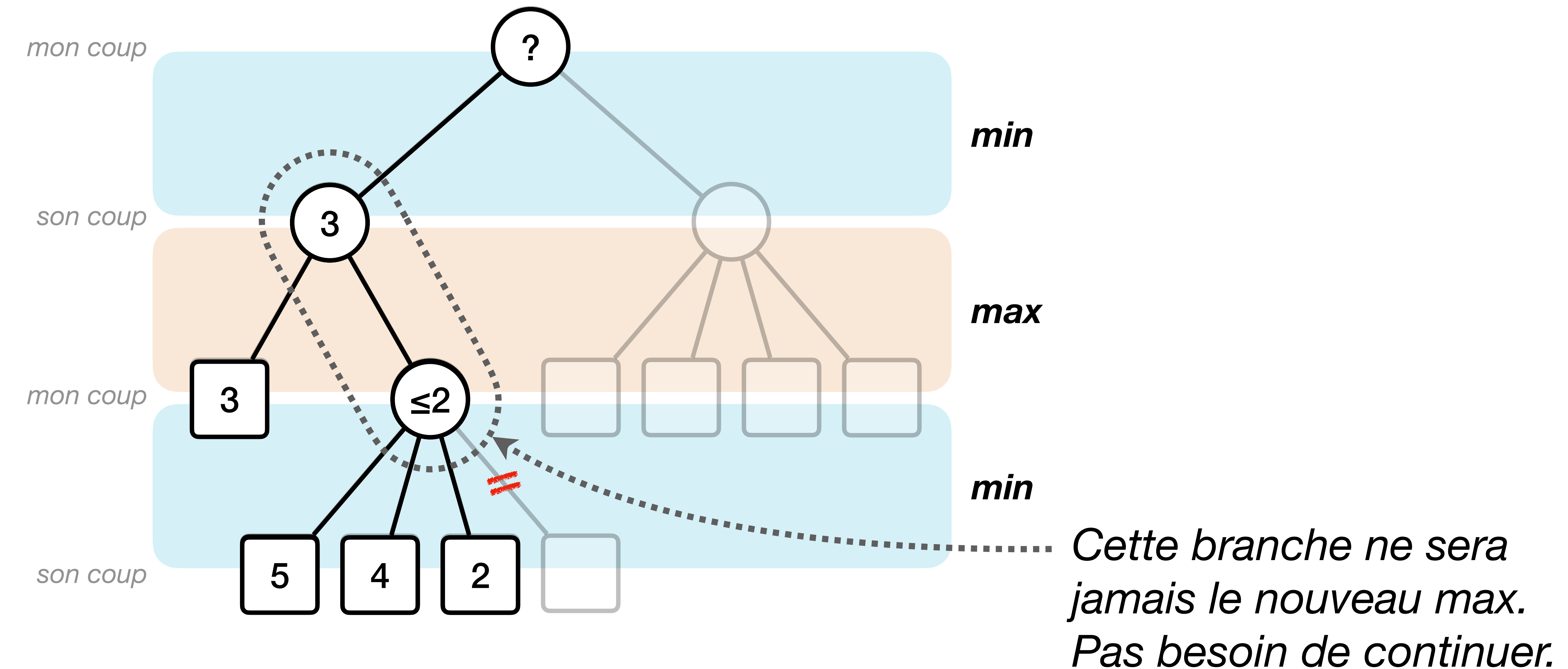
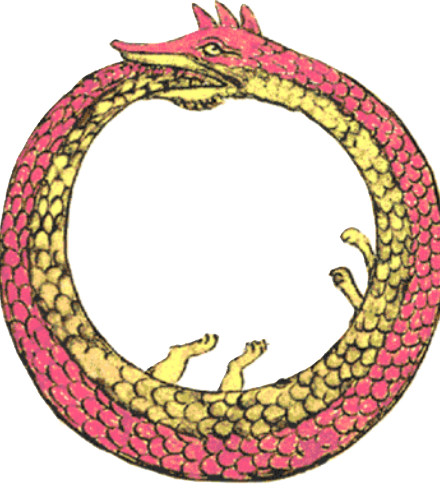
Elagage alpha-beta



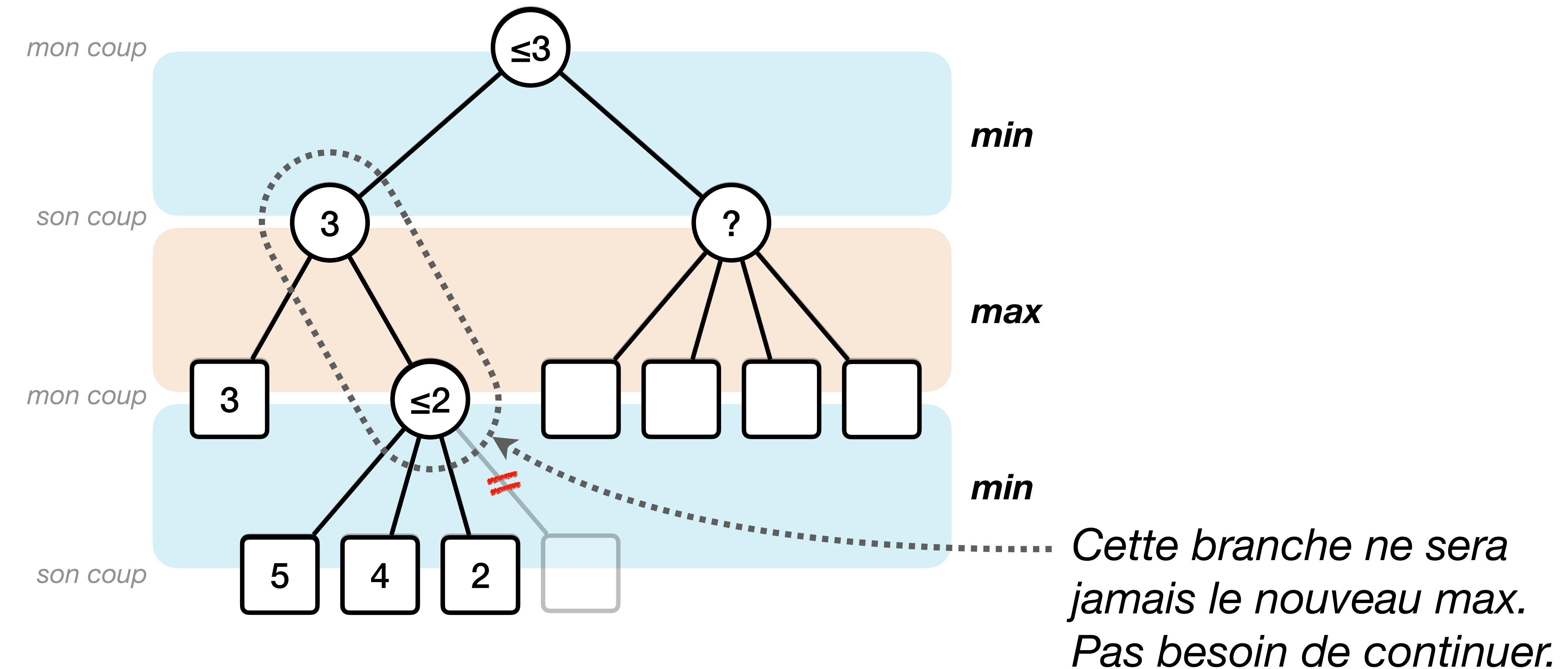
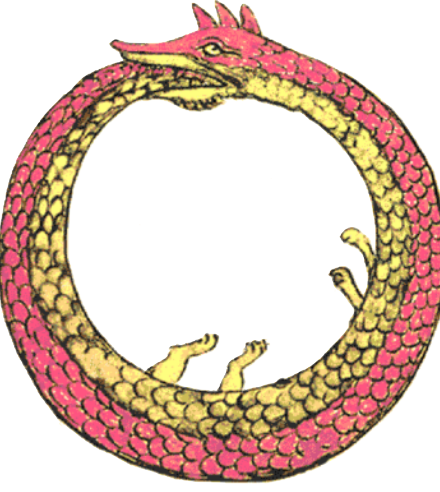
Elagage alpha-beta



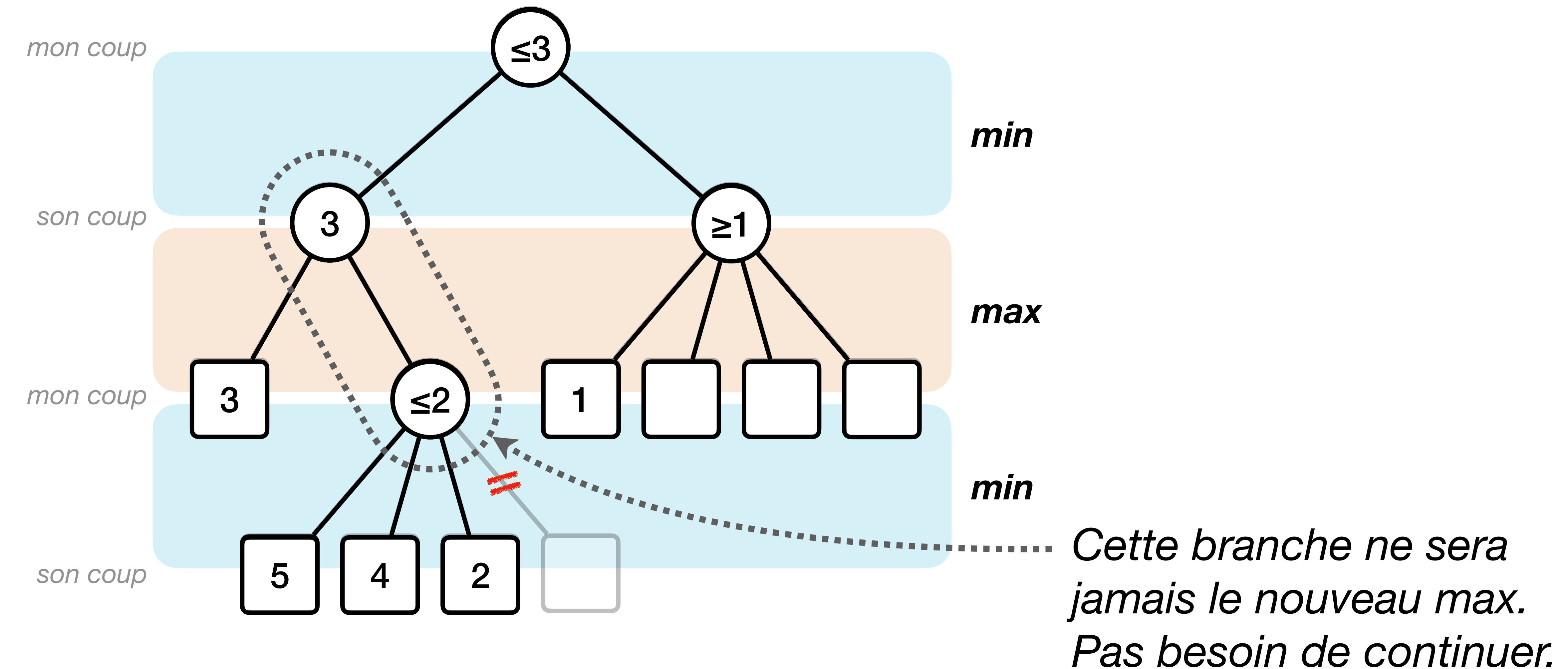
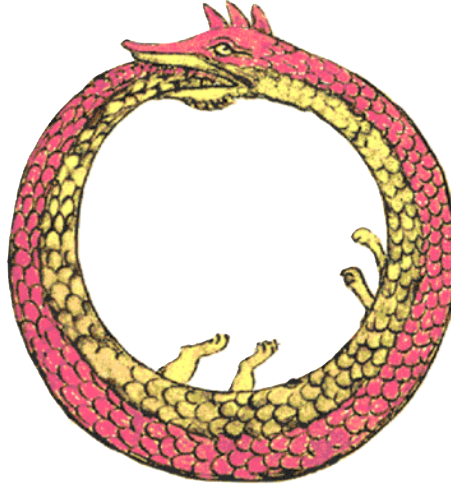
Elagage alpha-beta



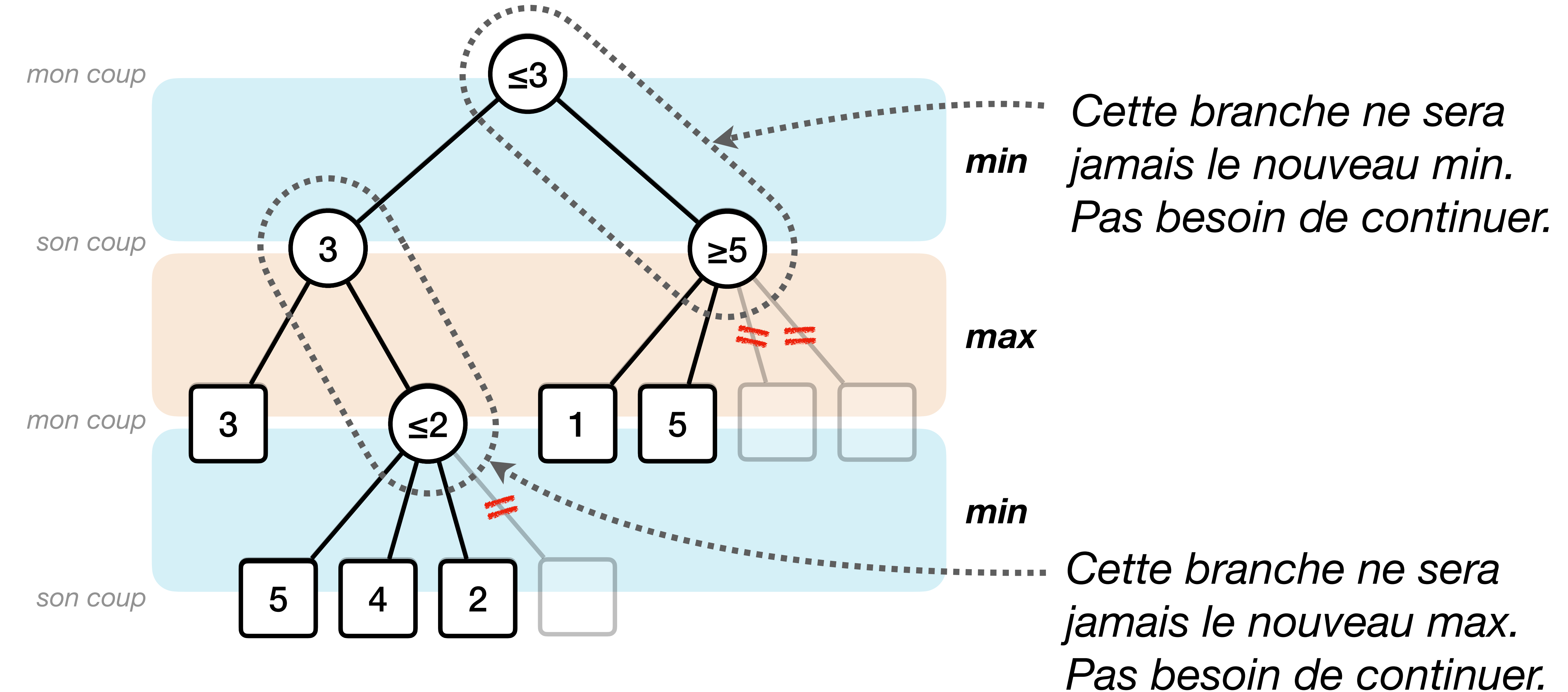
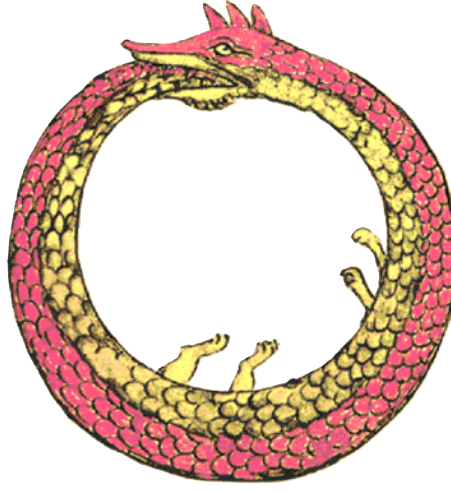
Elagage alpha-beta



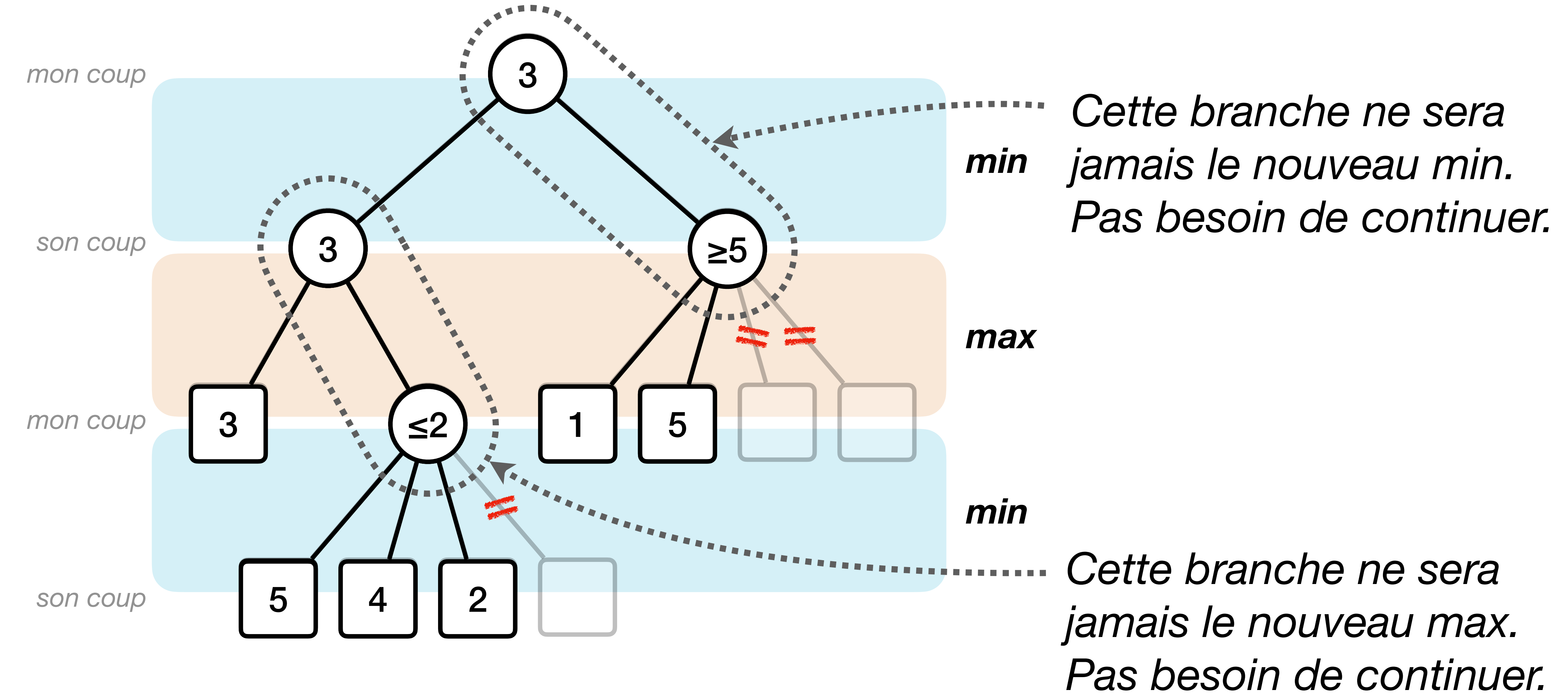
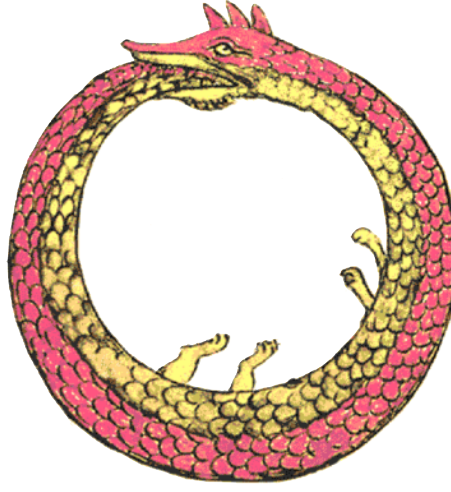
Elagage alpha-beta



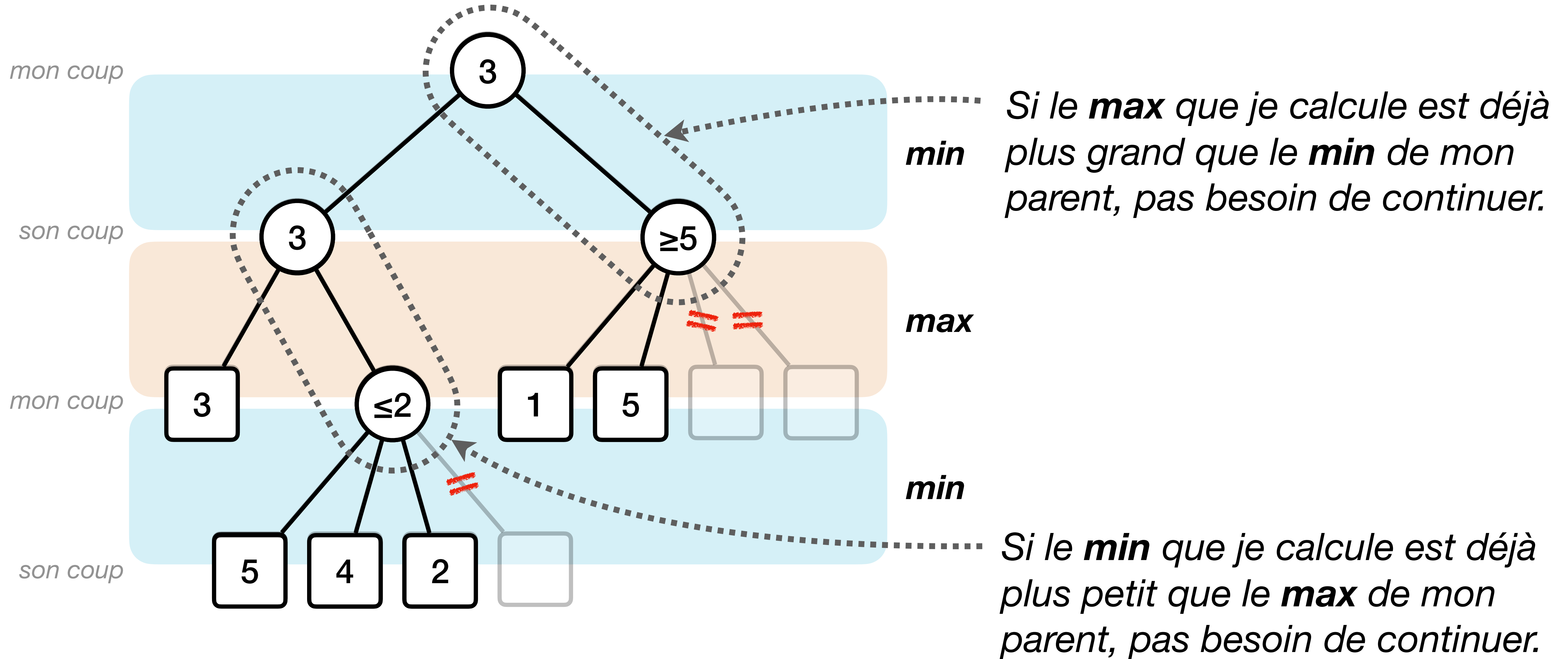
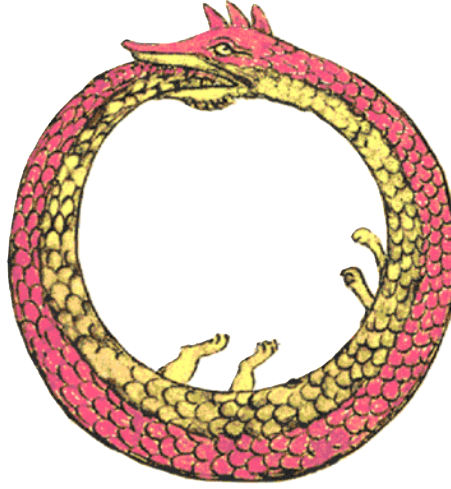
Elagage alpha-beta



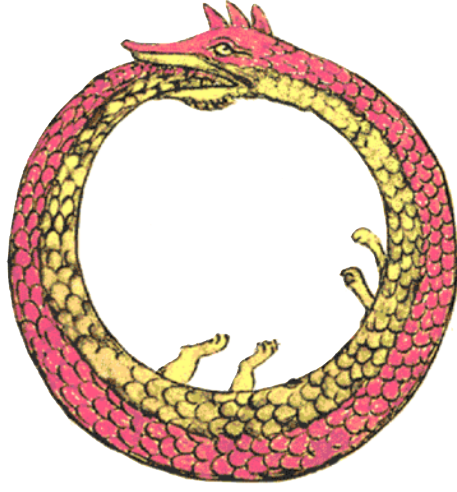
Elagage alpha-beta



Elagage alpha-beta

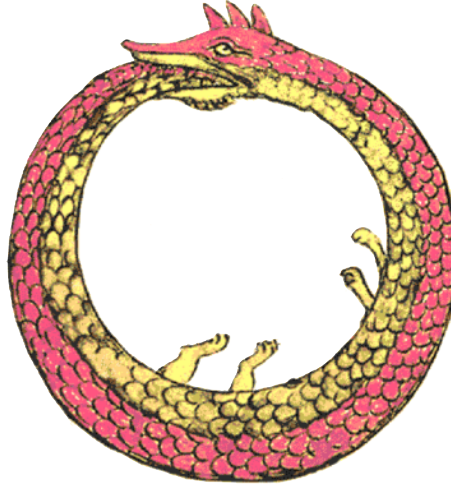


Gain lié à l'élagage ?



- Cela dépend de l'ordre dans lequel les coups sont considérés.
- Dans le meilleur cas, il permet de grandement augmenter la profondeur d'exploration
- Dans le pire cas, il n'apporte aucune amélioration
- Trouver une bonne heuristique pour l'ordre permet d'accélérer considérablement l'algorithme.

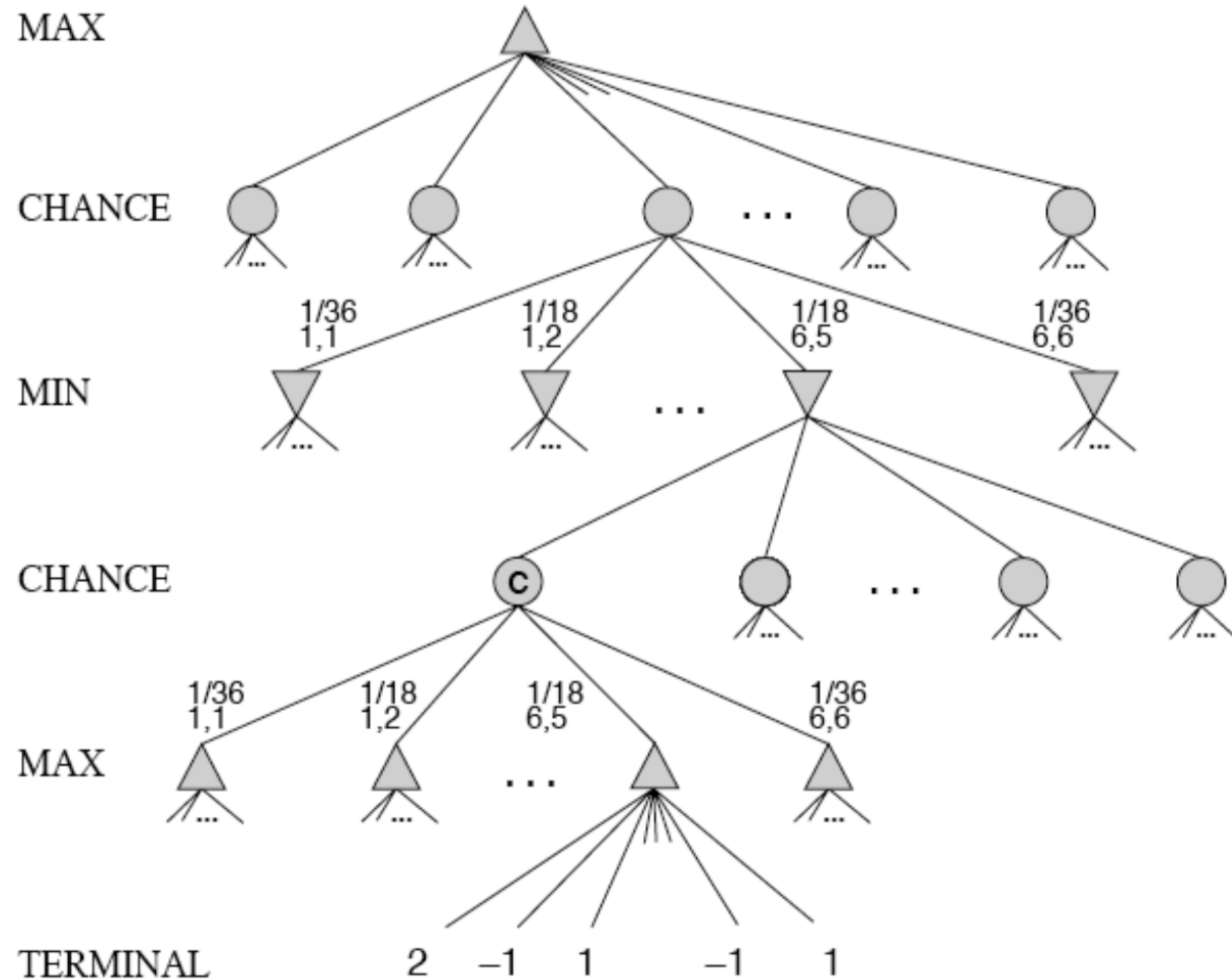
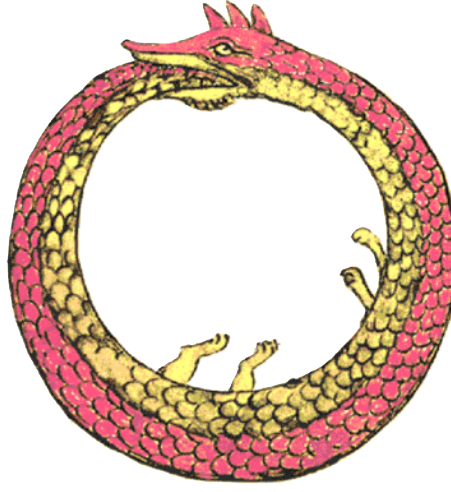
Backgammon



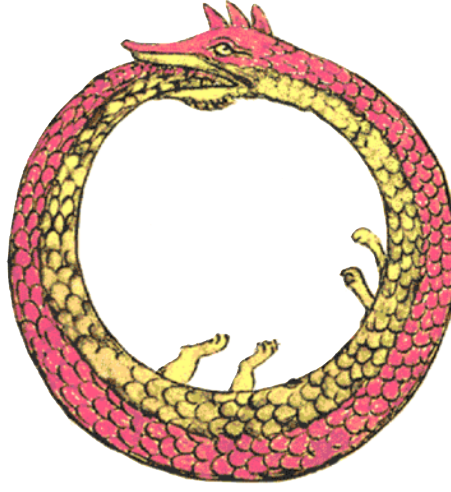
- Le lancer de dés introduit du hazard dans l'arbre des coups successifs possibles
- On intercale des moyennes pondérées par les probabilités $1/18$ ou $1/36$ entre les noeuds « Min » et « Max » de l'algorithme



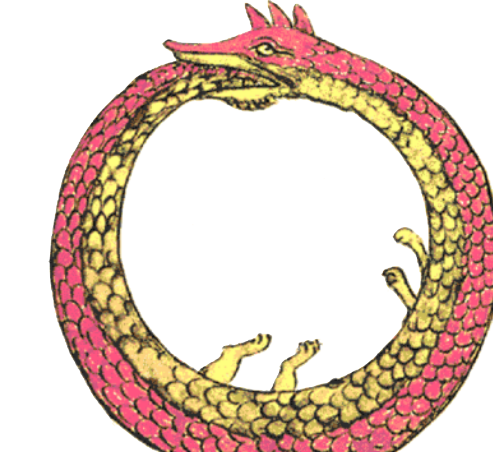
Minimax pour backgammon



Texas Hold'em à 2 joueurs



- La version la plus simple du jeu - Texas Holdem, 2 joueurs, mises limitées - est résolue en 2015
- Le programme Cepheus y joue parfaitement. Il utilise 12 terabytes de données précalculées



Brown *et al.*, *Science* **365**, 885–890 (2019)

30 August 2019

<https://doi.org/10.1126/science.aay2400>

RESEARCH

RESEARCH ARTICLE

COMPUTER SCIENCE

Superhuman AI for multiplayer poker

Noam Brown^{1,2*} and Tuomas Sandholm^{1,3,4,5*}

In recent years there have been great strides in artificial intelligence (AI), with games often serving as challenge problems, benchmarks, and milestones for progress. Poker has served for decades as such a challenge problem. Past successes in such benchmarks, including poker, have been limited to two-player games. However, poker in particular is traditionally played with more than two players. Multiplayer games present fundamental additional issues beyond those in two-player games, and multiplayer poker is a recognized AI milestone. In this paper we present Pluribus, an AI that we show is stronger than top human professionals in six-player no-limit Texas hold'em poker, the most popular form of poker played by humans.

Poker has served as a challenge problem for the fields of artificial intelligence (AI) and game theory for decades (1). In fact, the foundational papers on game theory used poker to illustrate their concepts (2, 3). The reason for this choice is simple: No other popular recreational game captures the challenges of hidden information as effectively and as elegantly as poker. Although poker has been useful as a benchmark for new AI and game-theoretic techniques, the challenge of hidden information in strategic settings is not limited to recreational games. The equilibrium

by, for example, trying to detect and exploit weaknesses in the opponent. A Nash equilibrium is a list of strategies, one for each player, in which no player can improve by deviating to a different strategy. Nash equilibria have been proven to exist in all finite games—and many infinite games—though finding an equilibrium may be difficult.

Two-player zero-sum games are a special class of games in which Nash equilibria also have an extremely useful additional property: Any player who chooses to use a Nash equilibrium is guaranteed to not lose in expectation no matter what

exact Nash equilibrium. For example, the Nash equilibrium strategy for Rock-Paper-Scissors is to randomly pick Rock, Paper, or Scissors with equal probability. Against such a strategy, the best that an opponent can do in expectation is tie (10). In this simple case, playing the Nash equilibrium also guarantees that the player will not win in expectation. However, in more complex games, even determining how to tie against a Nash equilibrium may be difficult; if the opponent ever chooses suboptimal actions, then playing the Nash equilibrium will indeed result in victory in expectation.

In principle, playing the Nash equilibrium can be combined with opponent exploitation by initially playing the equilibrium strategy and then over time shifting to a strategy that exploits the opponent's observed weaknesses (for example, by switching to always playing Paper against an opponent that always plays Rock) (11). However, except in certain restricted ways (12), shifting to an exploitative nonequilibrium strategy opens oneself up to exploitation because the opponent could also change strategies at any moment. Additionally, existing techniques for opponent exploitation require too many samples to be competitive with human ability outside of small games. Pluribus plays a fixed strategy that does not adapt to the observed tendencies of the opponents.

Although a Nash equilibrium strategy is guaranteed to exist in any finite game, efficient algorithms for finding one are only proven to exist for special classes of games, among which two-player zero-sum games are the most prom-

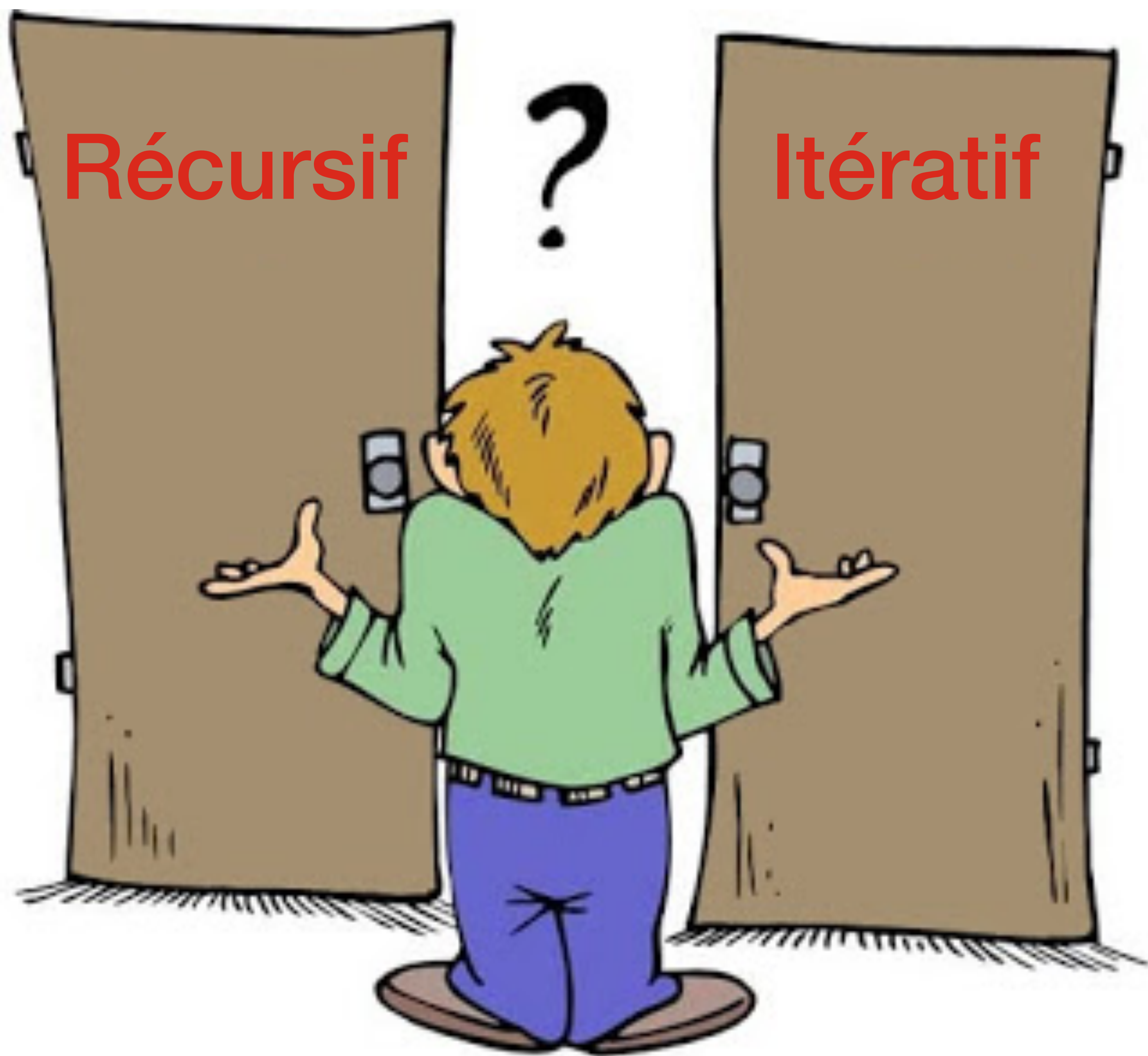


2.6.

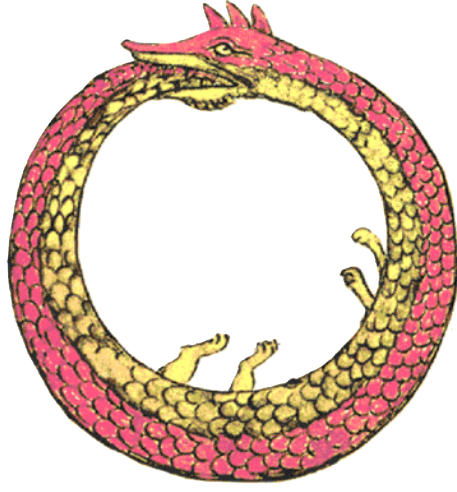
Récuratif

?

Itératif



Factorielle

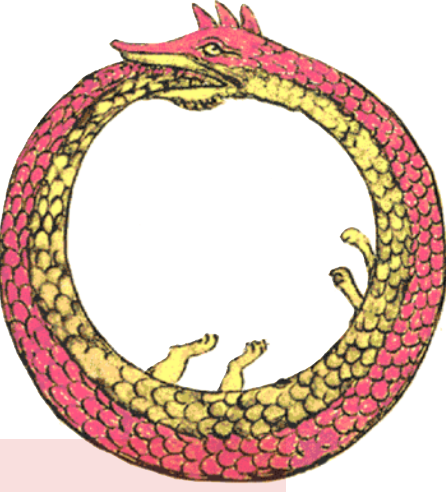


- Avec un seul appel récursif, la fonction récursive est équivalente à une simple boucle.
- Attention, les calculs sont effectués dans l'ordre inverse
- Récursif : $(((((1) \cdot 2) \cdot 3) \dots (n-1)) \cdot n)$
- Itératif : $1 \cdot n \cdot (n-1) \dots 3 \cdot 2$

```
fonction factorielle(n)
  si n vaut 0 alors
    retourner 1
  sinon
    retourner
      n x factorielle(n-1)
  fin si
```

```
fonction factorielle(n)
  r ← 1
  tant que n > 1
    r ← r x n
    décrémenter n
  fin tant que
  retourner r
```

Fibonacci

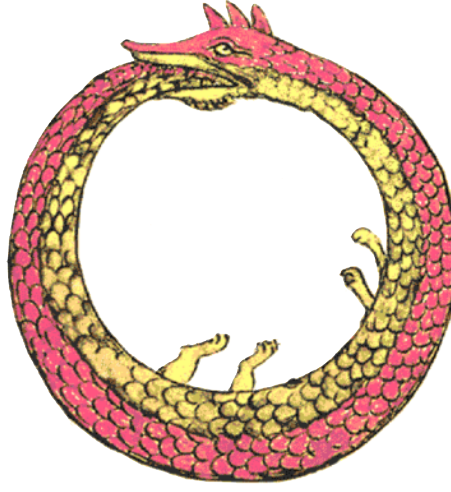


- Version itérative de Fibonacci plus complexe à écrire
- Complexité itérative bien meilleure
- Récursif en $O(\varphi^n)$
- Itératif en $O(n)$

```
fonction Fibonacci(n)
  si n vaut 0 ou 1 alors
    retourner n
  sinon
    retourner Fibonacci(n-1)
              + Fibonacci(n-2)
  fin si
```

```
fonction Fibonacci(n)
  si n vaut 0 ou 1 alors
    retourner n
   $F_{n-1} \leftarrow 0$ 
   $F_n \leftarrow 1$ 
  pour i de 2 à n
     $F_{n-2} \leftarrow F_{n-1}$ 
     $F_{n-1} \leftarrow F_n$ 
     $F_n \leftarrow F_{n-1} + F_{n-2}$ 
  fin pour
  retourner  $F_n$ 
```

Somme des n premiers entiers



- L'algorithme du chapitre 1 peut être écrit itérativement, récursivement ou en utilisant la formule de Gauss
- L'approche récursive est-elle une bonne idée ?

```
long sumRec(int n) {  
    if(n<=0) return 0;  
    return n + sumRec(n-1);  
}
```

```
long sumIter(int n) {  
    long sum = 0;  
    for (int i=0; i<=n; ++i)  
        sum += i;  
    return sum;  
}
```

```
long sumGauss(int n) {  
    return (long)n*(n+1)/2;  
}
```

playground > main.cpp

main.cpp

5using namespace std;

6

7long sumRec(int n) { n: 25267

8if(n<=0) return 0;

9return n + sumRec(n-1); n: 25267

10}

11

12int main() {

13cout << sumRec(200000);

14}

15

sumRec

Debug: playground

Debugger

Console

Frames

Variables

LLDB

Memory View

Threa...hread>

sumRec(int) main.cpp:9

sumRec(int) main.cpp:9

sumRec(int) main.cpp:9

sumRec(int) main.cpp:9

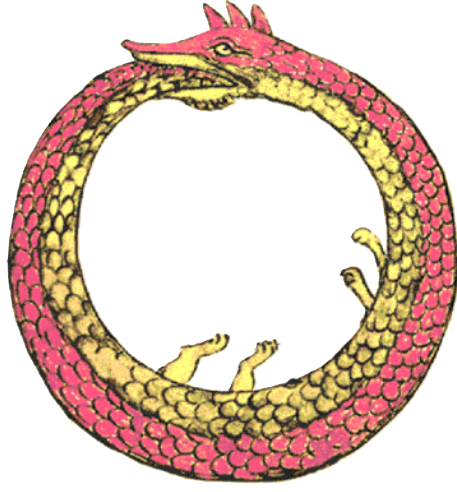
Exception = EXC_BAD_ACCESS (code=2, address=0x7ffee048dff8)

01 n = {int} 25267

Watches

No watches

Pile d'appels (call stack)



- À chaque appel de fonction, le programme empile dans une structure de pile
- les variables locales de fonction, et
- l'adresse de retour où sauter en sortant de la fonction.
- À chaque sortie de fonction, le programme dépile les informations liées à cette fonction et saute à l'adresse indiquée.
- Une récursion trop profonde peut dépasser la capacité de cette pile.
- Et pourtant ...

playground

main.cpp

main.cpp

1#include <iostream>

2#include <vector>

3#include <numeric>

4

5using namespace std;

6

7long sumRec(int n) {

8

9???

10}

11

12int main() {

13cout << sumRec(200000);

14}

15

f sumRec

Run:SommeRecursive

▶

⬆

⚙

■

⌂

>>

/Users/oce/OneDrive/GitHub/ocuisenaire/playground/cmake-build-release/SommeRecursive

20000100000

Process finished with exit code 0

Run

Debug

Problems

Terminal

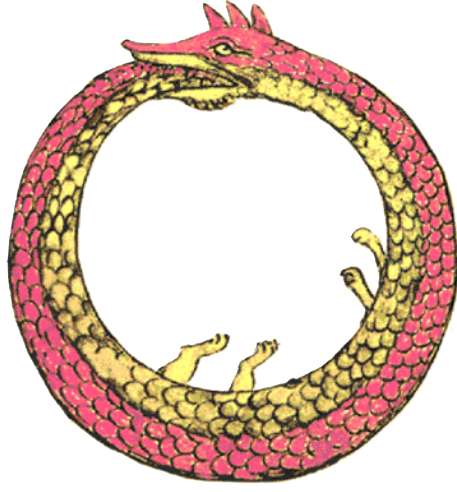
CMake

Messages

TODO

Event Log

Récursion finale (tail recursion)



Quand la toute dernière opération de la fonction réursive est l'appel récursif

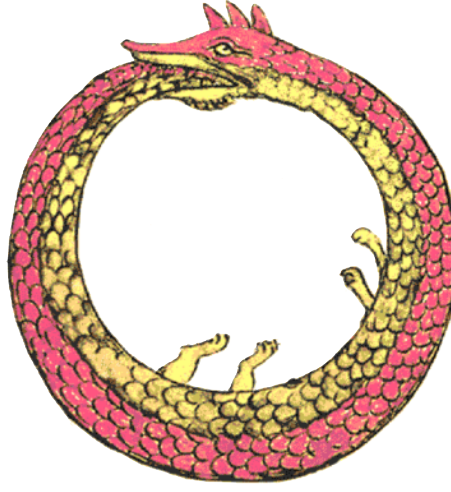
- Pas besoin d'empiler de nouvelles infos
- On peut réutiliser celles de la fonction courante

Mais... pas en mode debug

```
long sumRec(int n) {  
    if(n<=0) return 0;  
    return n + sumRec(n-1);  
}
```

```
long sumTailRec(int n, long r) {  
    if(n==0) return r;  
    return sumTailRec(n-1,n+r);  
}
```

Itération vs. récursion terminale



```
fonction recursion(n)
  faire A
  si B, alors
    faire C
    appeler recursion(..)
  fin si
```

```
fonction iteration(n)
  boucler
    faire A
    si non B, alors
      sortir boucle
    fin si
    faire C
  fin boucle
```

Tour de Hanoï itératif

fonction Hanoi(n)

bouger le plus petit disque

vers la droite (n pair)

ou la gauche (n impair)

boucler $2^{n-1}-1$ fois

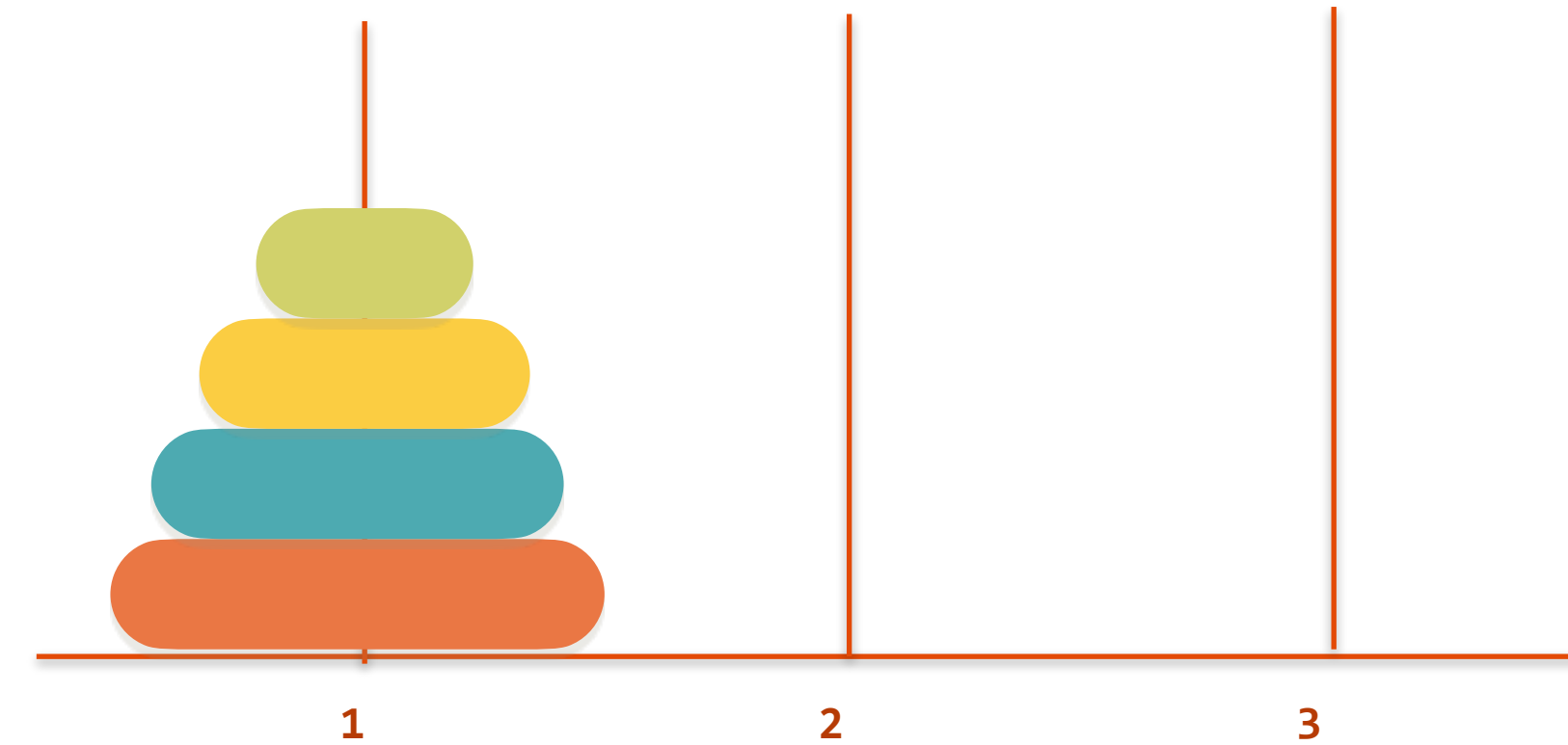
bouger le disque qui n'est pas le plus petit

bouger le plus petit disque

vers la droite (n pair)

ou la gauche (n impair)

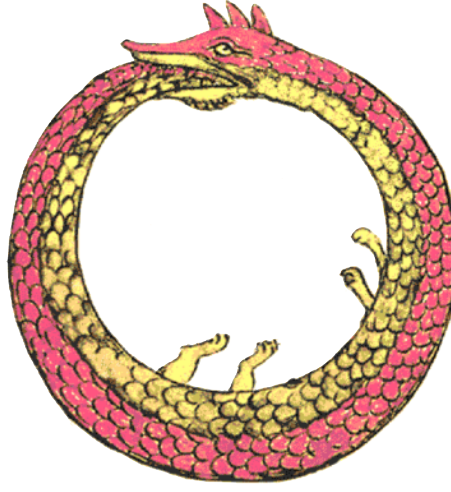
fin boucler



T0:	A	B	C
T1:			
T2:			
T0:	A	B	
T1:			
T2:	C		
T0:	A		
T1:	B		
T2:	C		
T0:	A		
T1:	B	C	
T2:			
T0:			
T1:	B	C	
T2:	A		
T0:	C		
T1:	B		
T2:	A		
T0:	C		
T1:			
T2:	A	B	
T0:			
T1:			
T2:	A	B	C

Algorithmes

Complexités



Factorielle récursif	$O(n)$
Factorielle itératif	$O(n)$
Fibonacci récursif	$O(1.618^n)$
Fibonacci itératif	$O(n)$
PGCD (Euclide)	$O(\log(n))$
Tours de Hanoï récursif	$O(2^n)$
Tours de Hanoï itératif	$O(2^n)$
Permutations	$O(n!)$
Tic Tac Toe	$9!$
Puissance 4, profondeur d'exploration de d tours	$O(7^d)$
Minimax (negamax), m mouvements possibles par tour, profondeur de d tours	$O(m^d)$