Bandit-based Monte-Carlo planning: the game of Go and beyond.

Vincent Berthier, Guillaume Chaslot, Christophe Fiter, Sylvain Gelly, Jean-Baptiste Hoock, Arpad Rimmel, Philippe Rolet, Fabien Teytaud, Olivier.Teytaud@inria.fr, Yizao Wang

TAO, Inria-Futurs, Cnrs 8623, Lri, Univ. Paris-Sud,
Digiteo Labs, Pascal Network of Excellence.

Inria / Nicta,
July 2009.

Outline

- Introduction: High-dimensional control in large dimension.
- Bandit-based Monte-Carlo Planning and UCT.
- Application to the game of Go and beyond

ubuntu

What is high-dimensional discrete time control ?

- There are time steps: 0, 1, 2, ..., H.
- There are states and transitions:

$$x_{i+1} = f(x_i, d_i)$$

- d_i is the decision at time step i:

$$d_i = u(x_i)$$

- There is a cost:

$$C = C(x_H)$$

==> We look for u(.) such that C is as small as possible.

ubuntu

High-dimensional discrete time control

$$x_{i+1} = f(x_i, d_i)$$

$$d_i = u(x_i)$$

$$C = C(x_H)$$

==> We look for u(.) such that C is as small as possible.

Discrete time + high dimension + uncertainty

$x_{i+1} = f(x_i, d_i, A_i)$ $\quad\quad$ $d_i = u(x_i)$

$C = C(x_H)$

A_i might be:

- a Markov model

- an opponent: A_i maximizes inf C

==> we look for u(.) such that C is as small as possible (e.g. on average).

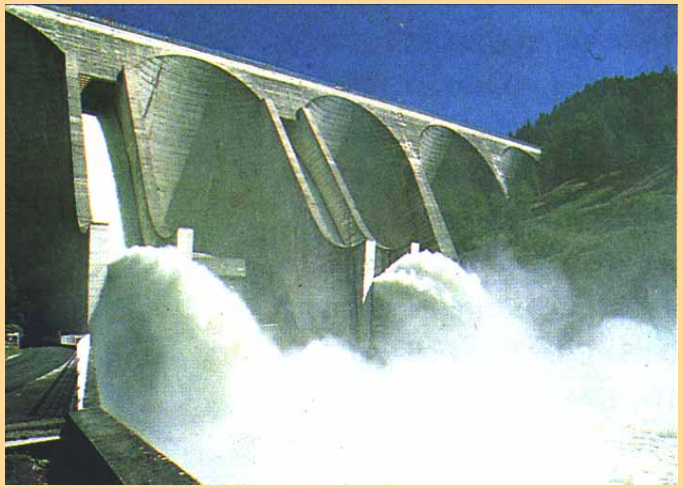
Summary

- High dimensional discrete time control is an important problem
- Many problems have no satisfactory solution.
- A new approach: Bandit-Based Monte-Carlo Planning

High-dimensional discrete time control

A main application: the management of many energy stocks in front of randomness

- At each time step we see random outcomes
- We have to take decisions
- After H time steps, we observe a cost



What are the approaches ?

- Dynamic programming (Massé – Bellman 50's) (still the main approach in industry)
- Reinforcement learning (some promising results, less used in industry)
- Some tree exploration tools (less usual in stochastic or continuous cases)
- Bandit-Based Monte-Carlo planning

Quelles sont les approches ?

- Dynamic programming (Massé – Bellman 50's) (still the main approach in industry)

Where we are:

Done: Presentation of the problem.

Now: We briefly present dynamic programming

Thereafter: We present MCTS / UCT.

Dynamic programming

- V(x) = expectation of $C(x_H)$ if optimal strategy. *(well defined)*
- u(x) such that the expectation of V(f(x,u(x),A)) is minimal
- Computation by dynamic programming

We compute V for all the X with horizon H.

Dynamic programming

- V(x) = expectation of $C(x_H)$ if optimal strategy. *(well defined)*
- u(x) such that the expectation of V(f(x,u(x),A)) is minimal
- Computation by dynamic programming

We compute V for all the X with horizon H.

We compute V for all the X with horizon H-1.

Programmation dynamique

- V(x) = expectation of $C(x_H)$ if optimal strategy. *(well defined)*
- u(x) such that the expectation of V(f(x,u(x),A)) is minimal
- Computation by dynamic programming

We compute V for all the X with horizon H.

We compute V for all the X with horizon H-1.

... ...

Extensions

- Approximate dynamic programming (e.g. for continuous domains)
- Reinforcement learning
  - Case where f(...) or A is black-box
  - Huge state spaces

  ==> but lack of stability
- ...

==> there is room for improvements

Monte-Carlo Tree Search

- Monte-Carlo Tree Search (MCTS) appeared in games.
- Its most well-known variant is termed Upper Confidence Tree (UCT).
- I here present UCT.
  - Bandits;
  - Monte-Carlo approach for tree-search;
  - UCT.

A ``bandit'' problem

- *p1,...,pN* unknown probabilities $\in$ *[0,1]*
- At each time step $i \in [1,n]$
  - choose $u_i \in \{1,...,N\}$ (as a function of u_j and r_j, $j<i$)
  - With probability p_{ui}
    - win ($r_i=1$)
    - loose ($r_i=0$)

A ``bandit'' problem: the target

- *p1,...,pN unknown probabilities* $\in [0,1]$
- At each time step $i \in [1,n]$
  - choose $u_i \in \{1,...,N\}$ (as a function of u_j and r_j, $j<i$)
  - With probability p_{ui}
    - win ($r_i=1$)
    - loose ($r_i=0$)

Regret: $Rn=n \max\{pi\} - \sum rj \quad (j<n)$

How to minimize the regret (worst case on p) ?

ubuntu

Bandits – a classical solution

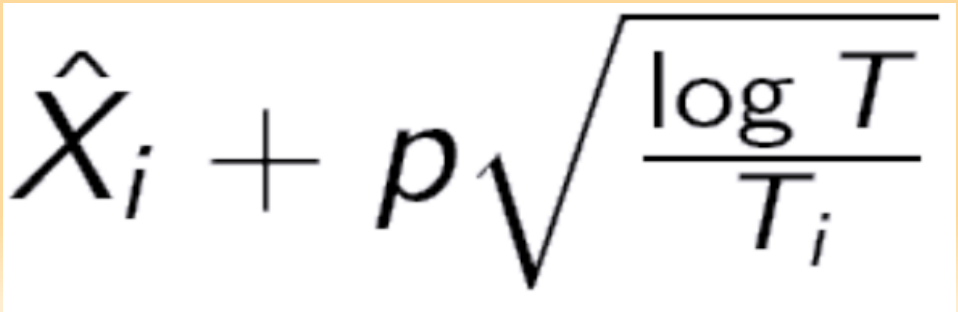
Regret: Rn=n max{pi} - $\sum$ rj (j<i)

UCB1: Choose u maximizing the compromise:

Empirical average for decision u

+ √(log(i)/ number of trials with decision u)

==> optimal regret O(log(n))



(Lai et al; Auer et al)

ubuntu

Bandits: much more

What is a bandit:

- a criterion (here a bandit)

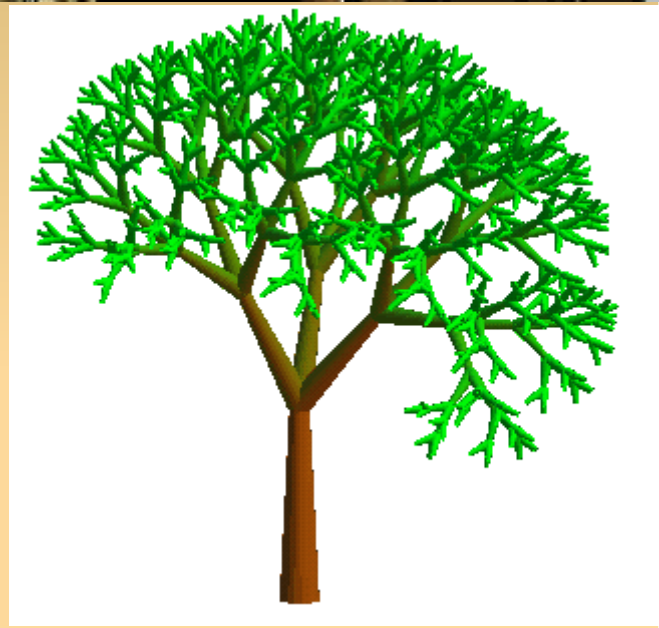
  defines the problem

- usually a score (typically exploration+exploitation)

  defines a criterion

==> an optimal score for a criterion is not optimal for another ==> a wide literature

Bandits and trees



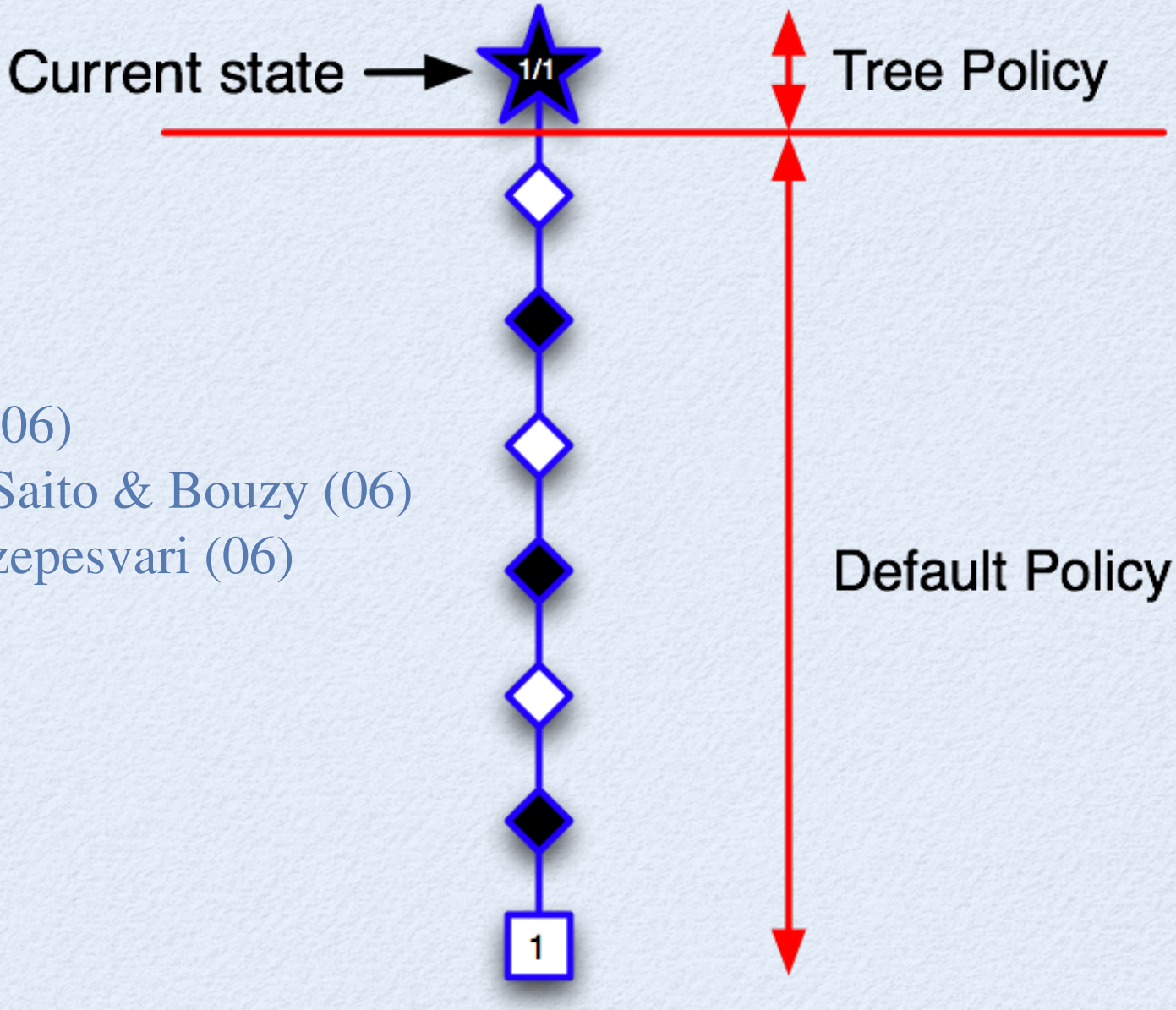
- we have seen the definition of discrete time control problems;

- we have seen what are bandits

- we now introduce trees and UCT

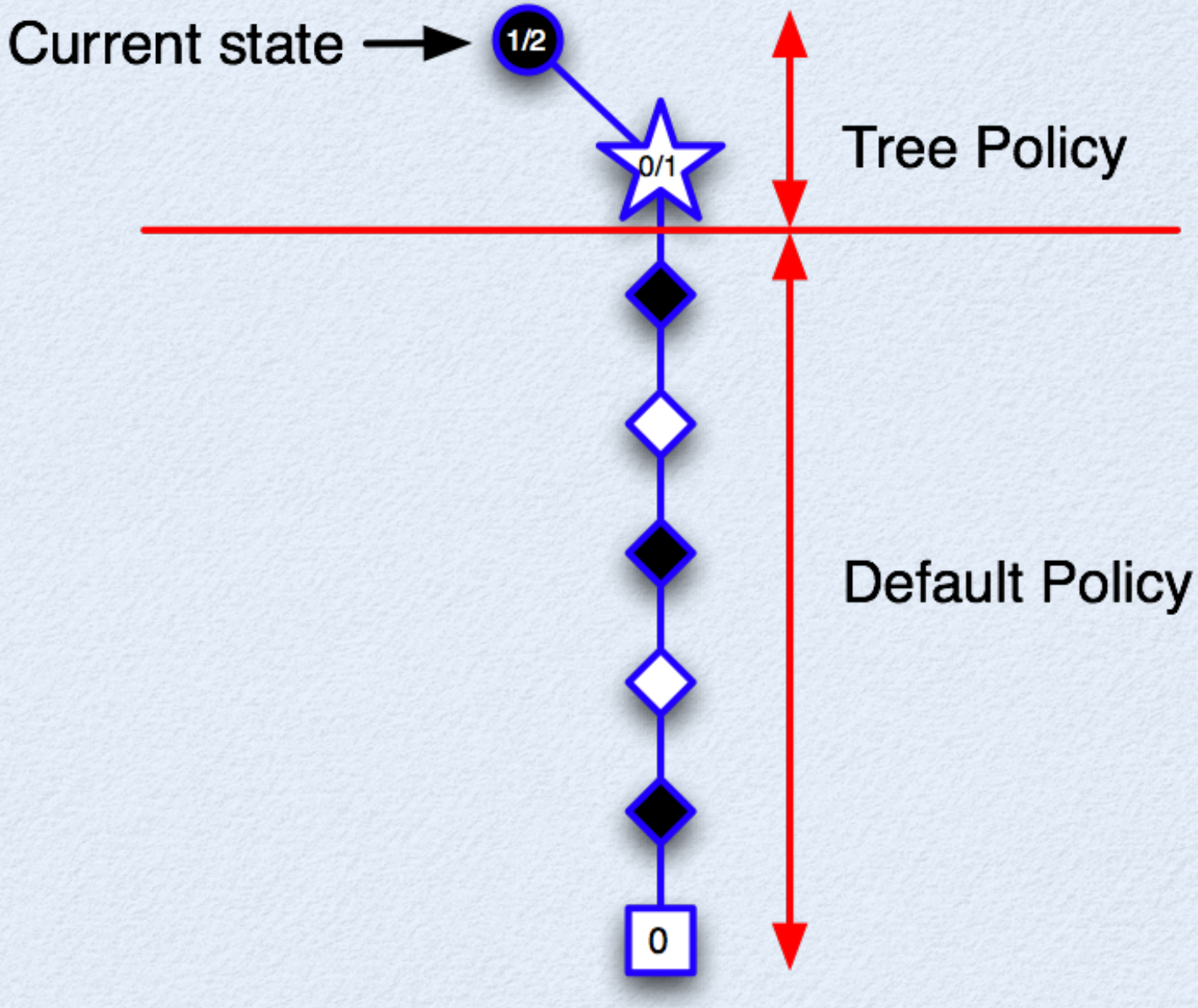
ubuntu

UCT (Upper Confidence Trees)

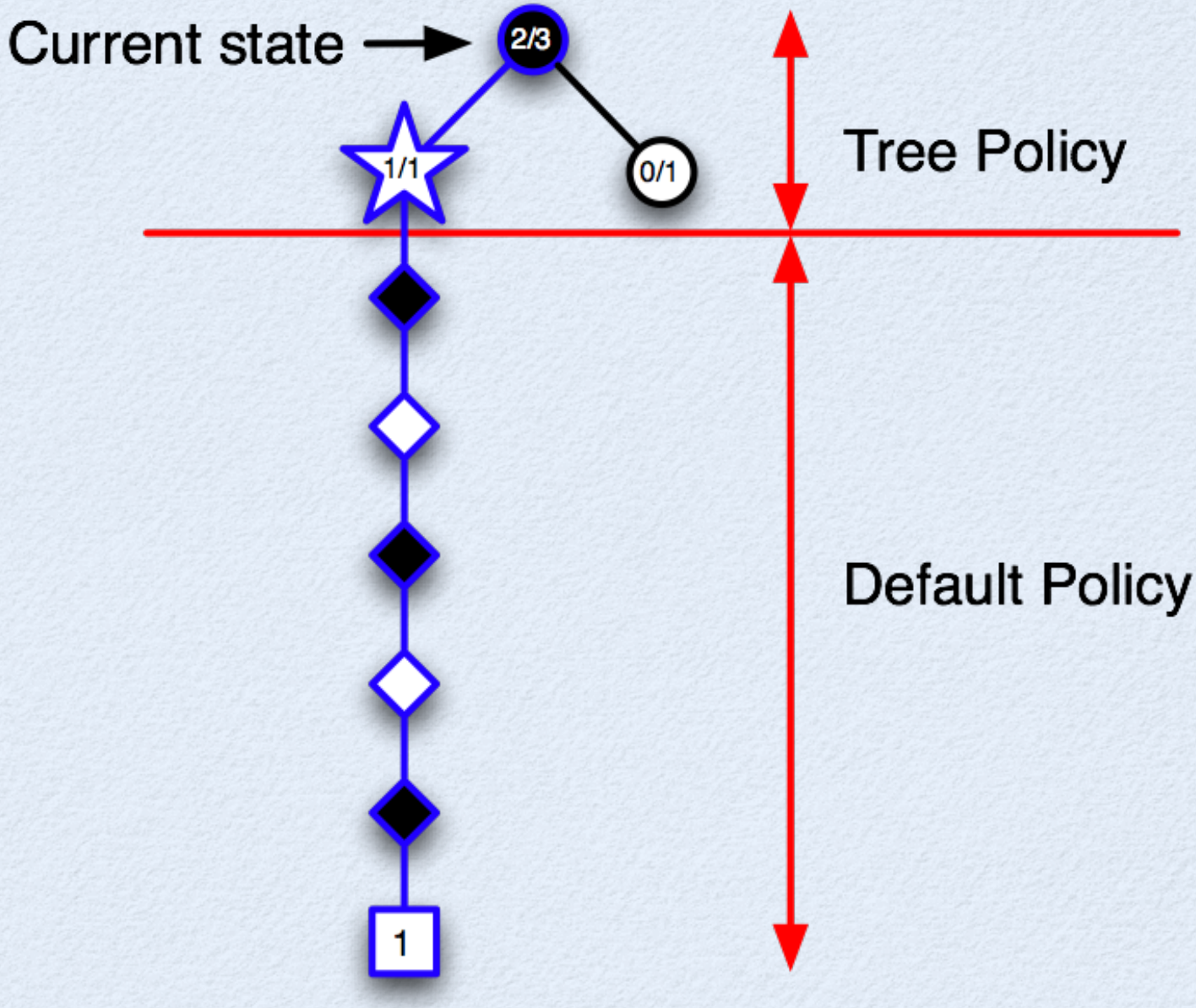


Coulom (06)
Chaslot, Saito & Bouzy (06)
Kocsis Szepesvari (06)

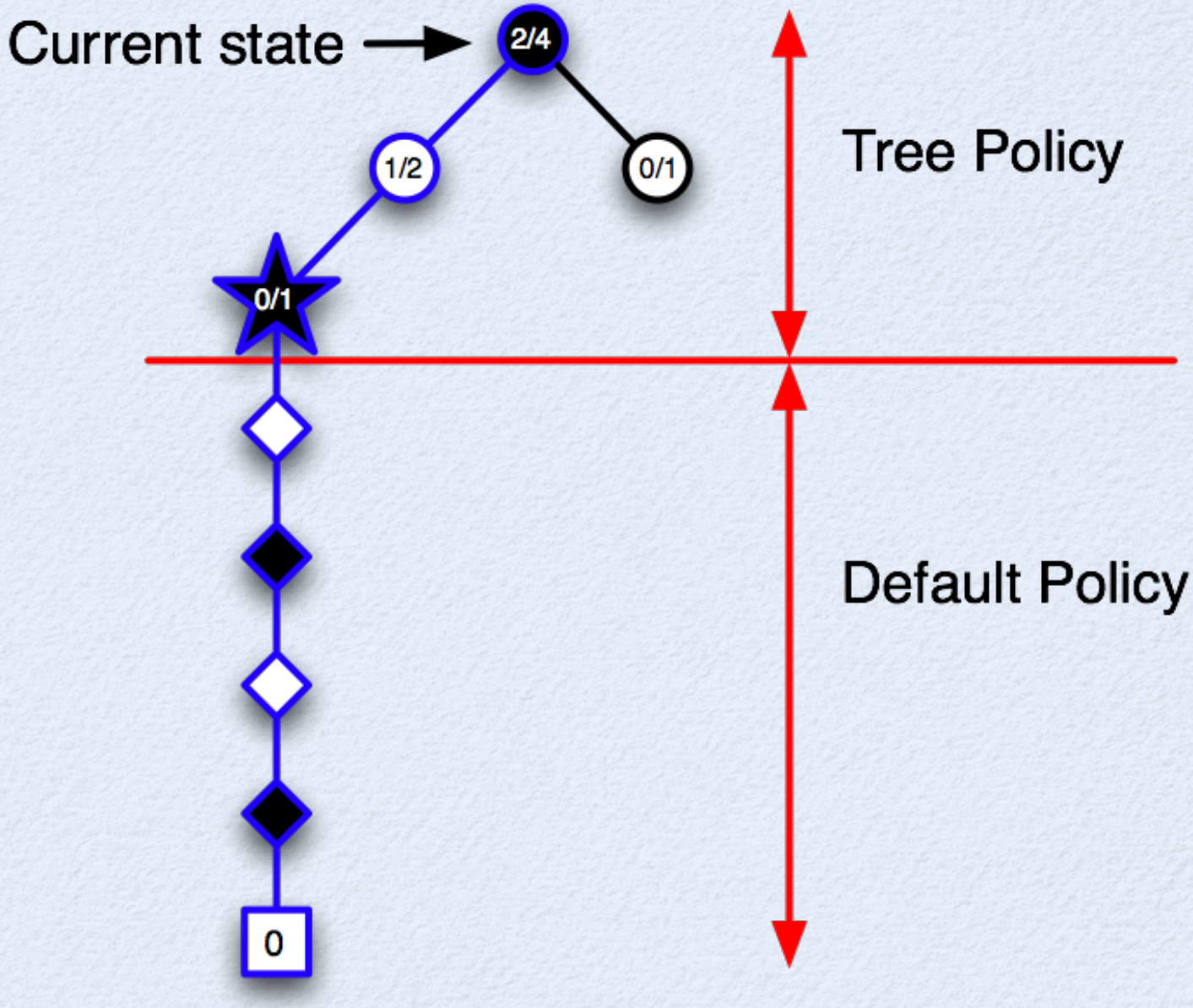
UCT



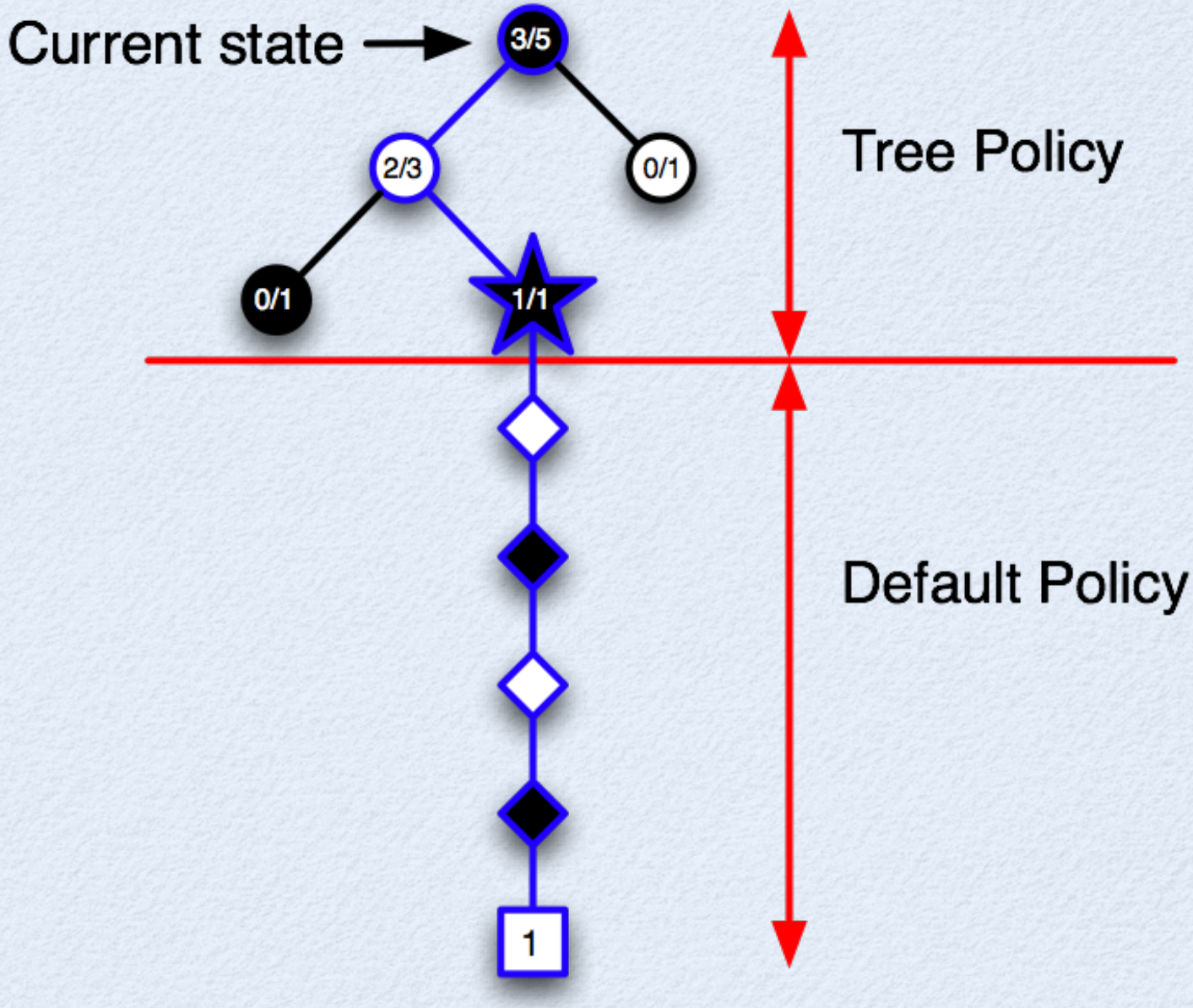
UCT



UCT

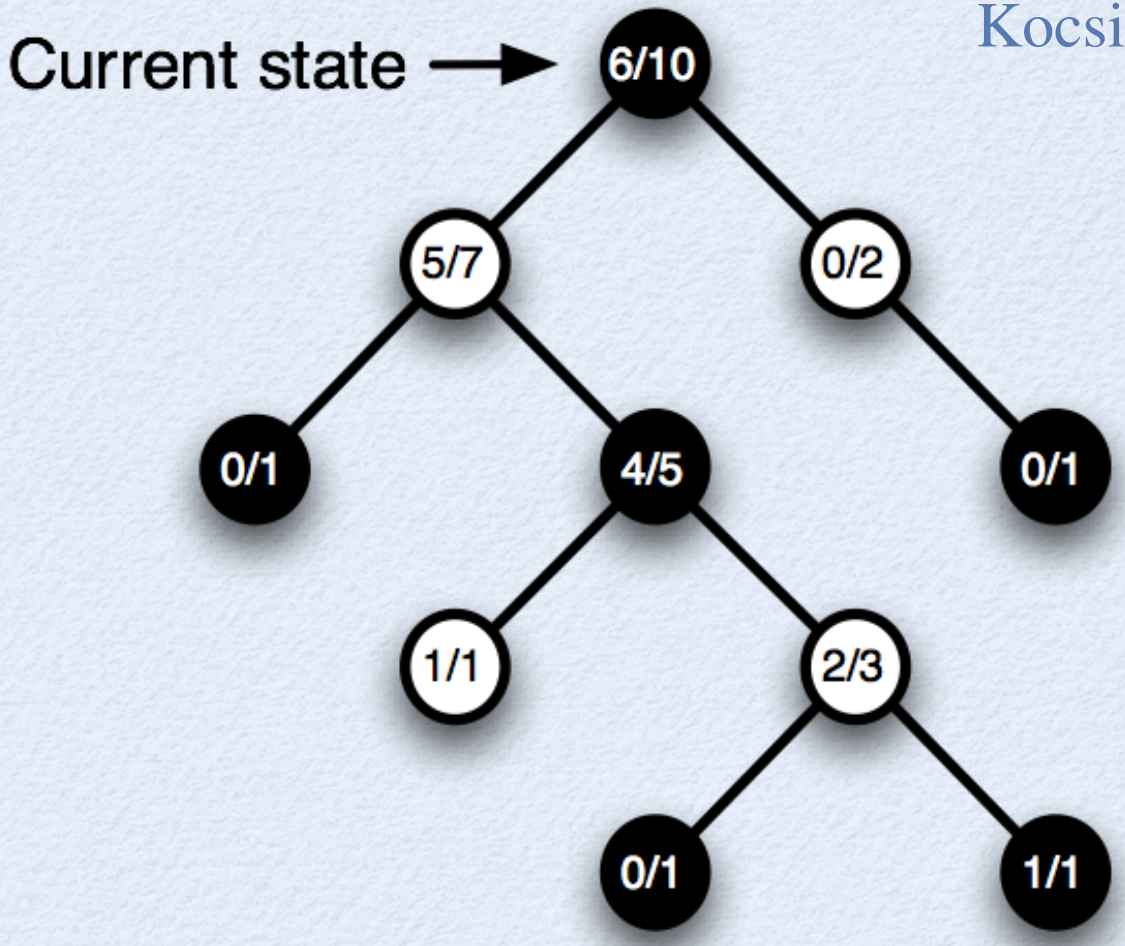


UCT

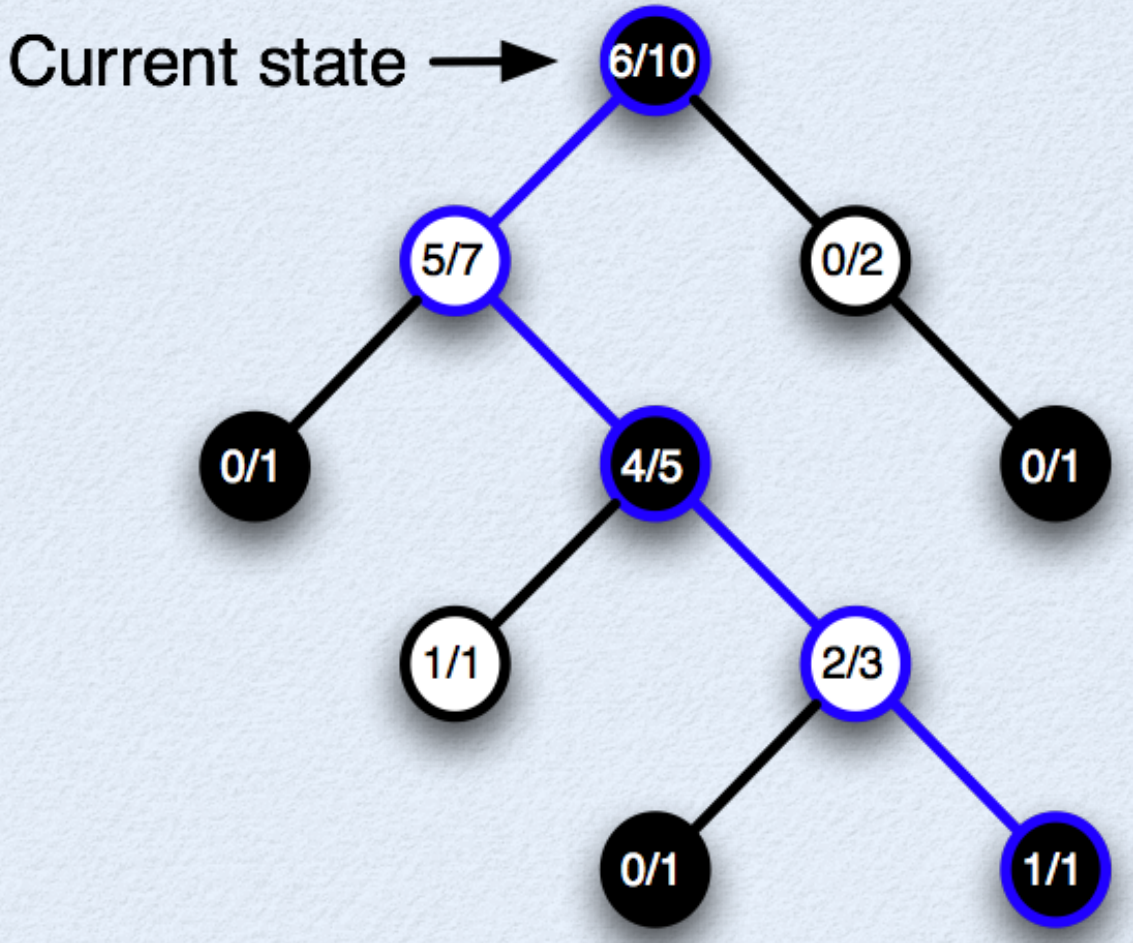


UCT

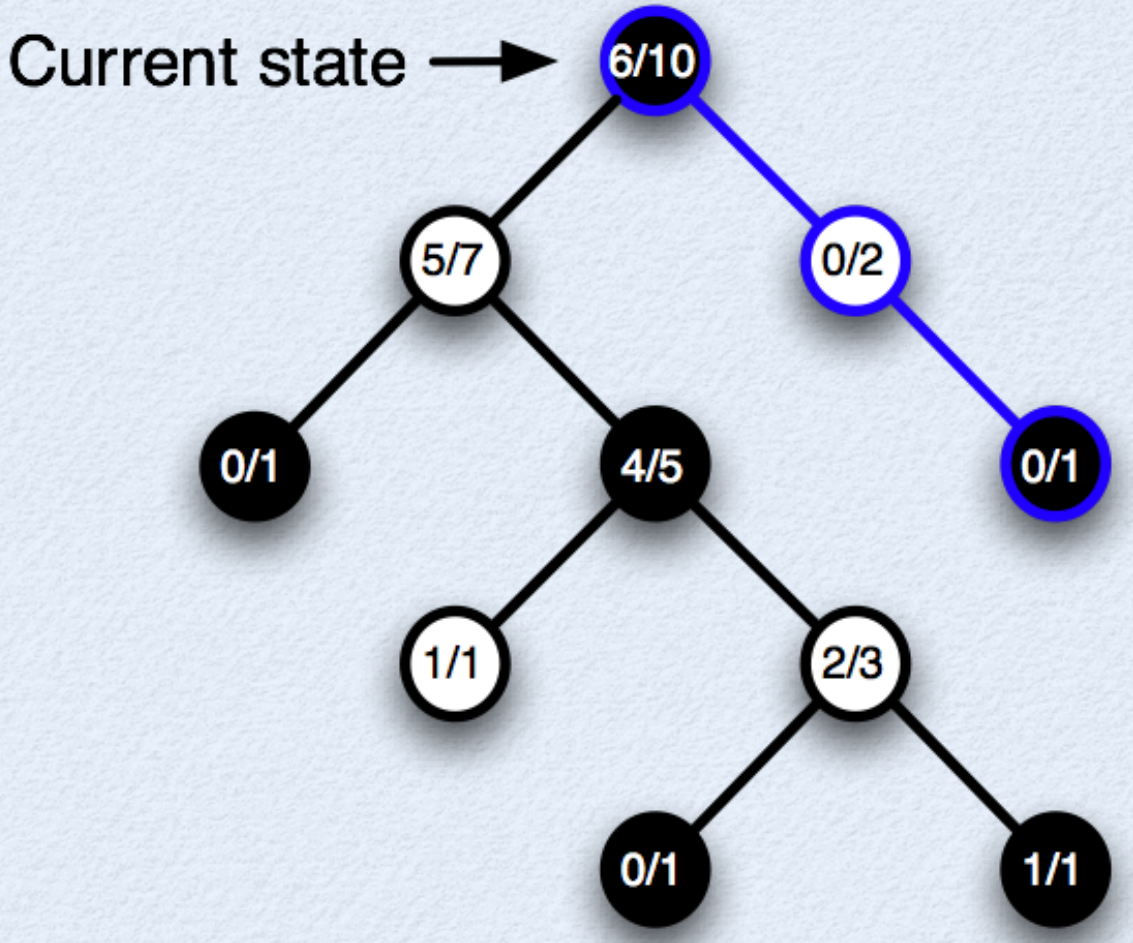
Kocsis & Szepesvari (06)



Exploitation ...



... or exploration ?



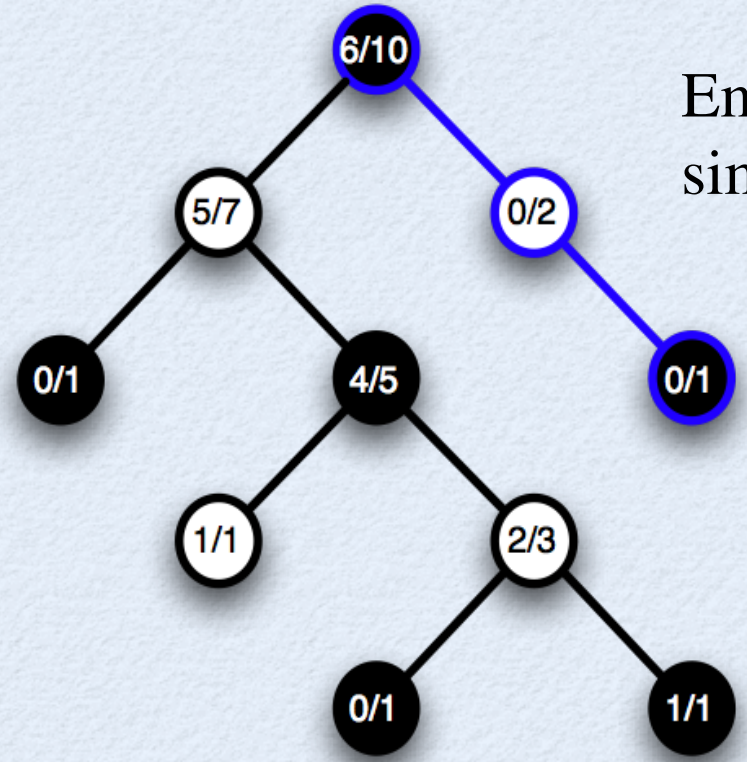
Go: from 29 to 6 stones

Formula for simulation

$$Select\ i_{next} = \arg \max_{i \in \text{children nodes}} \hat{\mu}_i + \sqrt{\frac{\log N}{n_i}}$$

Converges to the most likely sequence (asymptotically).

Empirically, the most simulated move is not so bad.



Go: from 29 to 6 stones

2007: win against a pro (5p) 9x9 en blitz	MoGo
2008: win against a pro (5p) 9x9	MoGo
2009: win against a pro (5p) 9x9 noir	MoGo
2008: win against a pro (8p) 19x19, H9	MoGo
2008: win against a pro (4p) 19x19, H8	CrazyStone*
2008: win against a pro (4p) 19x19, H7	CrazyStone*
2009: win against a pro (9p) 19x19, H7	MoGo
2009: win against a pro (1p) 19x19, H6	MoGo

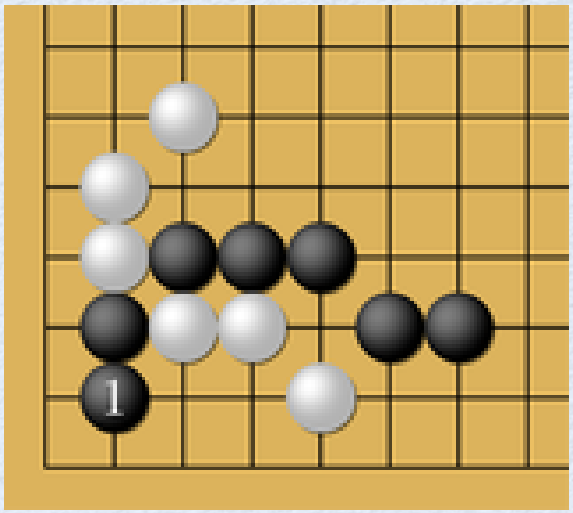
==> still 6 stones at least!

* French also!

As a stupid French politician claim that French research is bad I fell free of being chauvinistic.

Conclusion

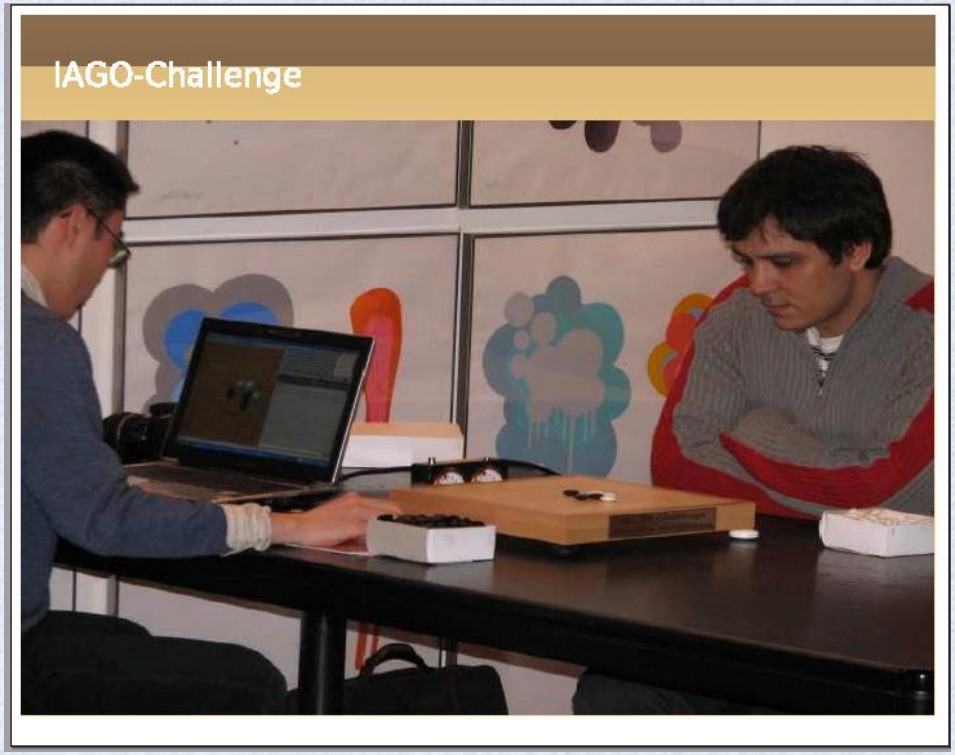
- Essentially asymptotically proved only
- Empirically good for
  - The game of Go
  - Some other games
  - Non-linear expensive optimization
  - Active learning
- Not (yet) tested industrially
- Understood weaknesses <--- No time for this, sorry



- Next challenge:
  - Solve these weaknesses
  - Industrial applications
  - **Partially observable cases** **<== collaboration ?**

Biblio

- Bandits: Lai, Robbins, Auer, Cesa-Bianchi...
- UCT: Kocsis, Szepesvari, Coquelin, Munos...
- MCTS (Go): Coulom, Chaslot, Fiter, Gelly, Hoock, Silver, Muller, Pérez, Rimmel, Wang...
- Tree + DP for industrial applicationl: Péret, Garcia...
- Bandits with infinitely many arms: Audibert, Coulom, Munos, Wang...
- Applications far from Go: Rolet, Teytaud (F), Rimmel, De Mesmay ...
- Links with “macro-actions”
- Parallelization, mixing with offline learning, bias...



Extensions

$$\hat{X}_i + p\sqrt{\frac{\log T}{T_i}}$$

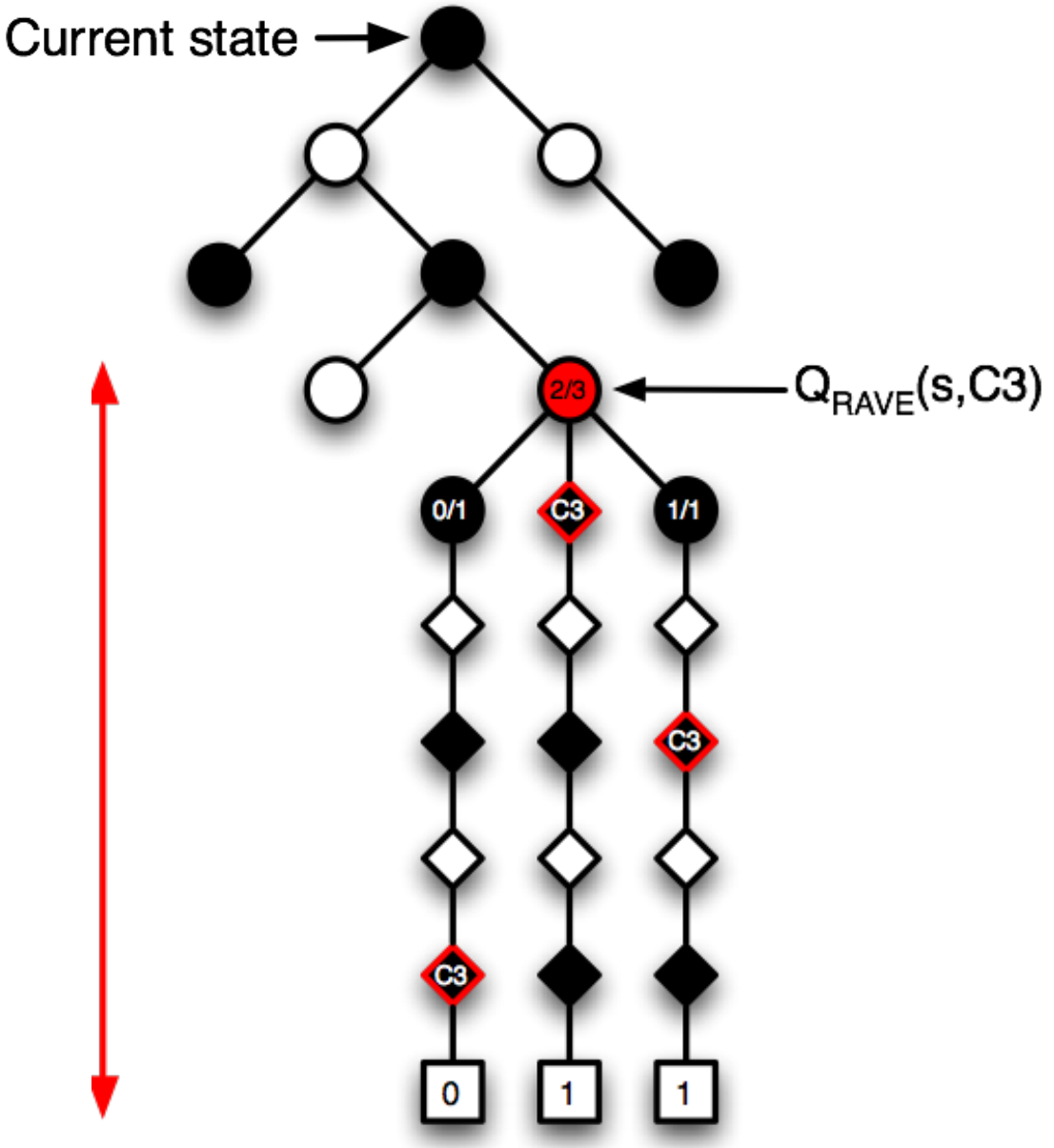
- Apprentissages:
  - En ligne (UCT) (tend vers la vraie valeur)
  - Transient (RAVE) (valeur approchée)
  - Hors ligne

- Parallélisation à mémoire partagée
- Parallélisation à passage de message

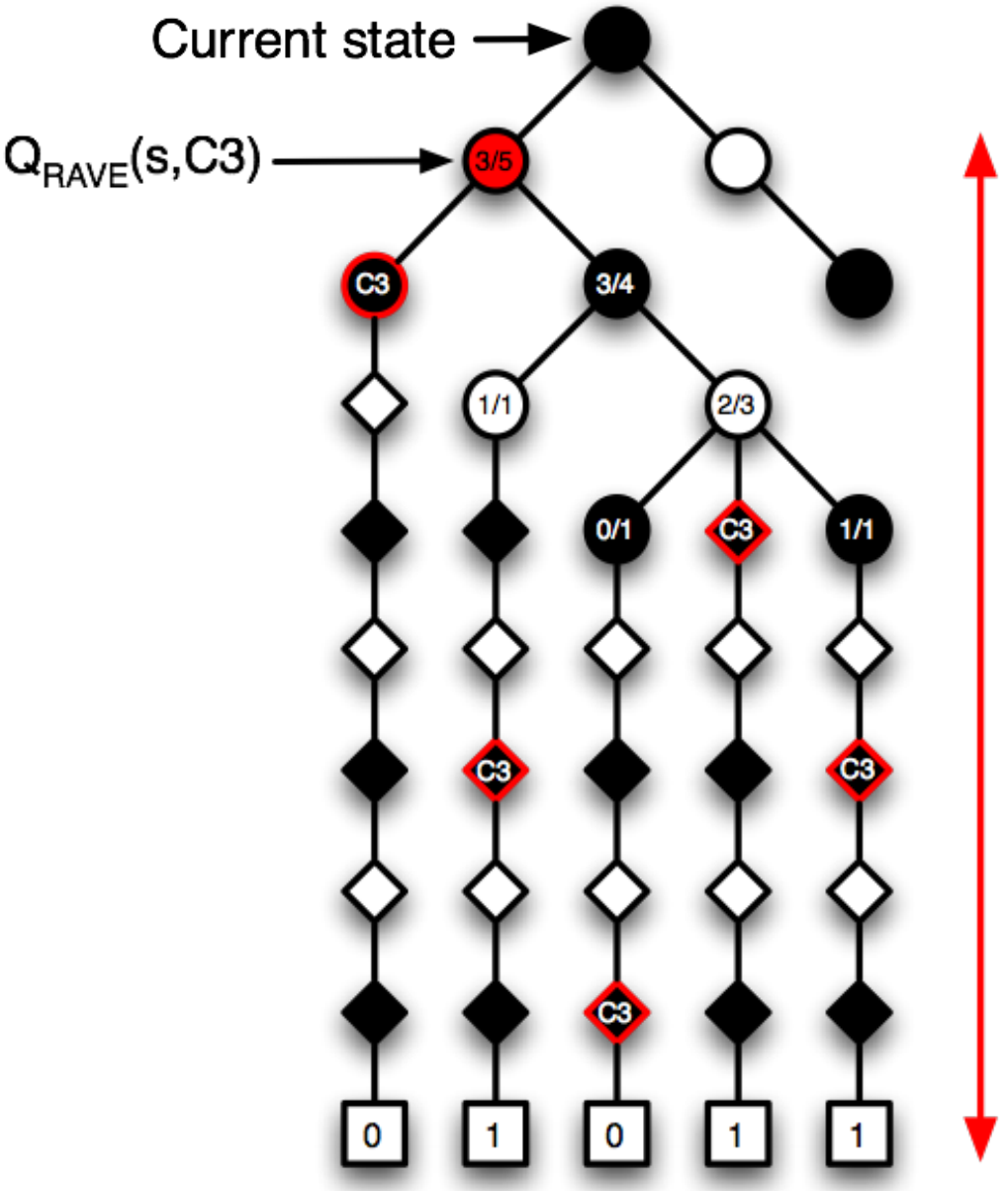
Apprentissage transient

- UCT ne “généralise” pas entre états
- RAVE apprend une fonction de valeur en ligne, qui fournit une première indication pour guider UCT

RAVE



RAVE



UCT-RAVE

- $Q_{UCT}(s,a)$ a moins de biais mais plus de variance
- $Q_{RAVE}(s,a)$ est plus biaisé mais a moins de variances
- On utilise un compromis entre les deux scores:

$$\alpha \; Q_{UCT}(s,a) + (1-\alpha) \; Q_{RAVE}(s,a)$$

$$\alpha = n/(n+K)$$

Apprentissage hors ligne

Q(s,a) = fréquence de choix du motif (appris sur bases de données).

$$Q = Q(s,a)/\log(n) + \alpha \; Q_{UCT}(s,a) + (1-\alpha) \; Q_{RAVE}(s,a)$$

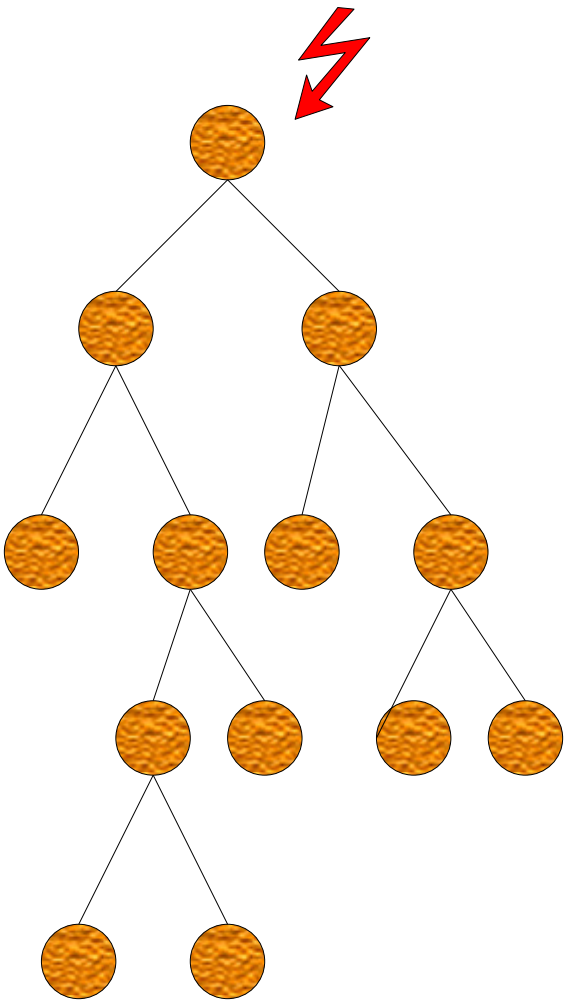
$$\alpha = n/(n+K)$$

Parallélisation à mémoire partagée

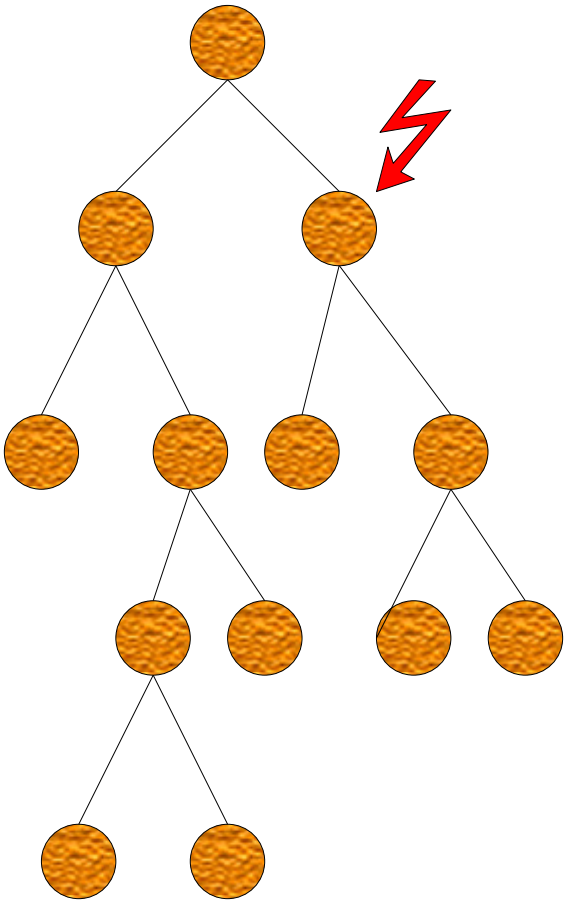
- Chaque coeur fait ses propres simulations.
- On met à jour le même arbre.

  ==> solution simple et pas mal efficace.

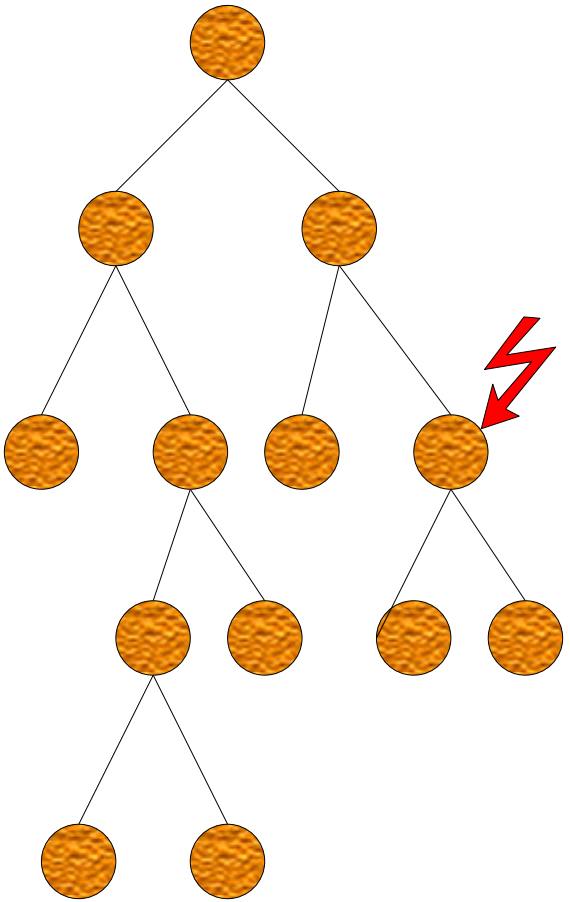
Multi-core machines



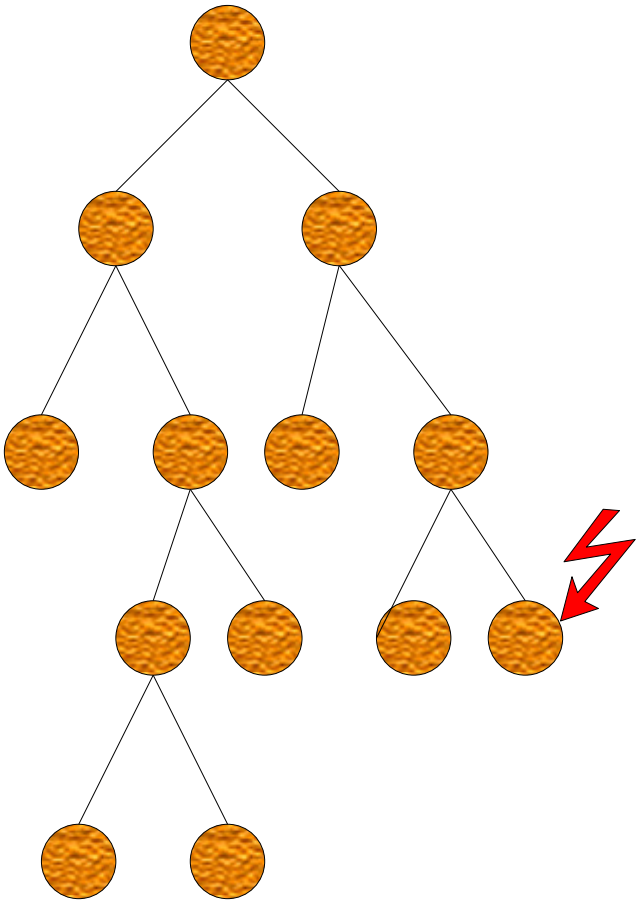
Multi-core machines



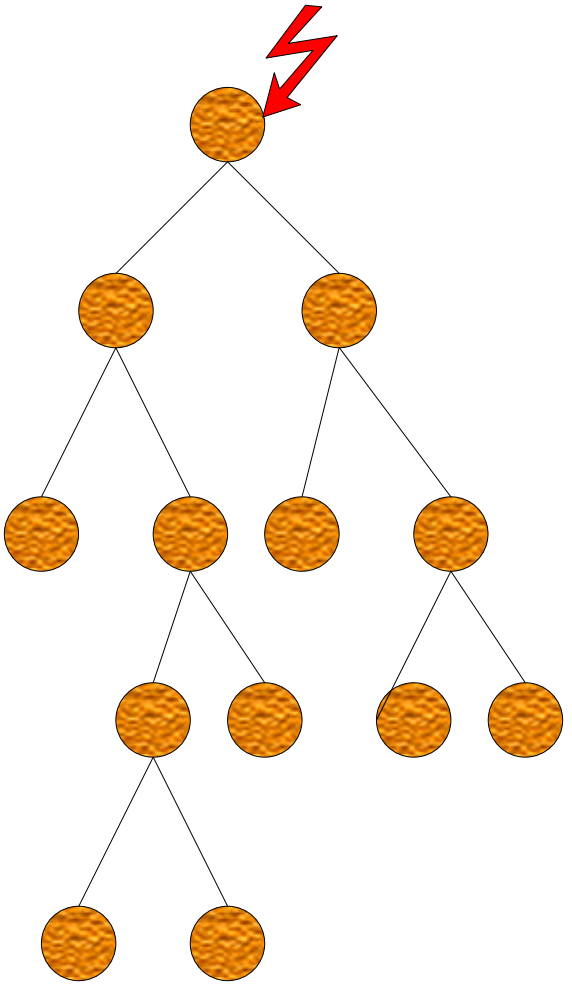
Multi-core machines



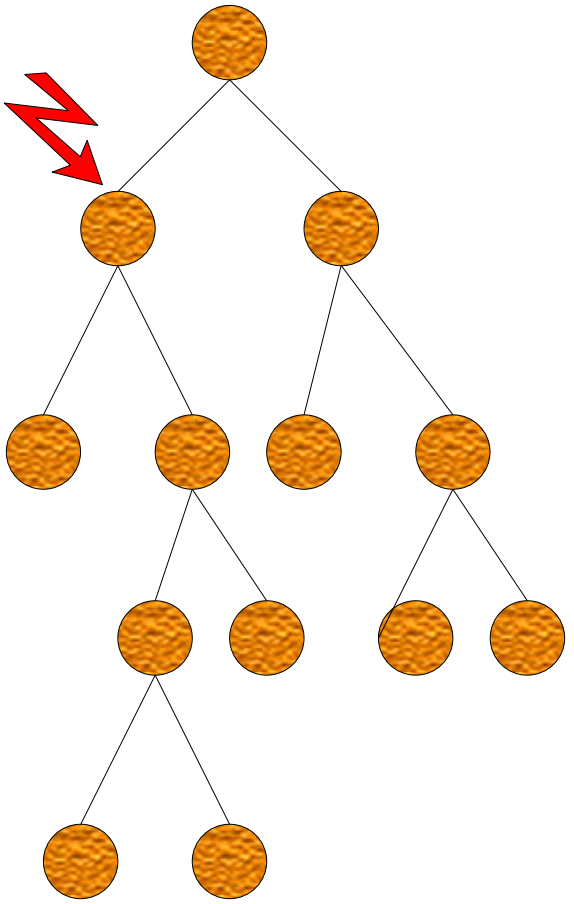
Multi-core machines



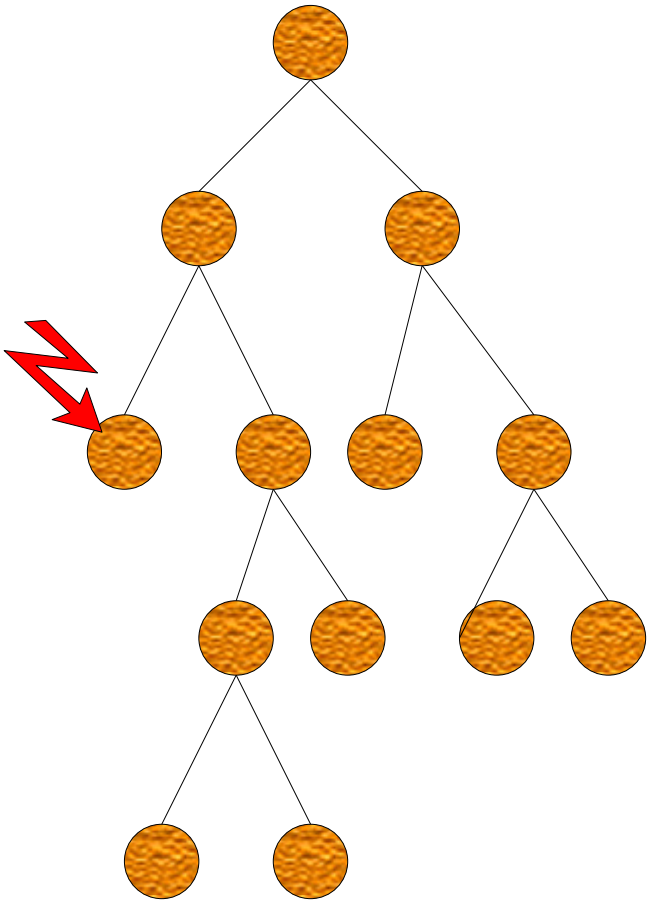
Multi-core machines



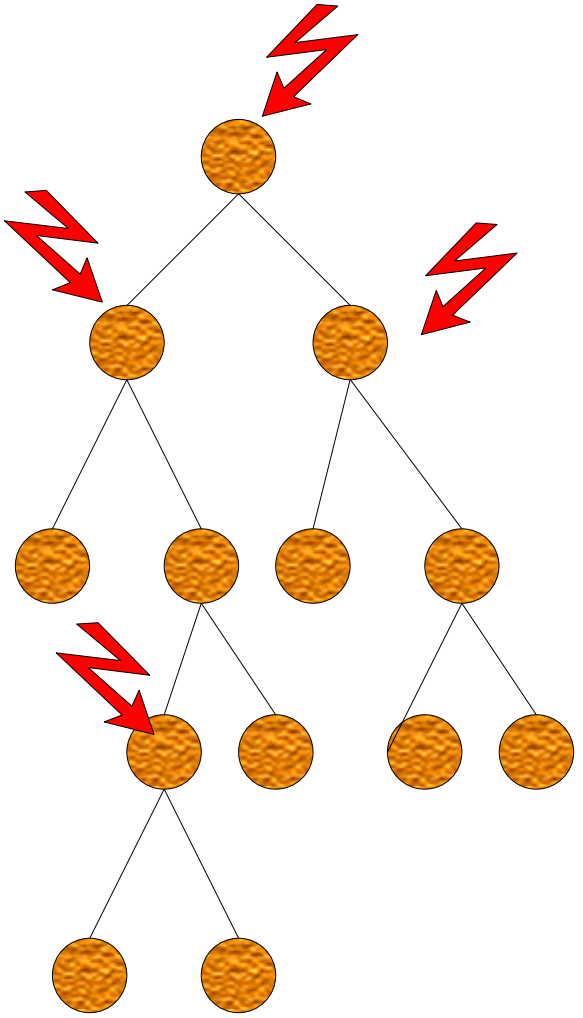
Multi-core machines



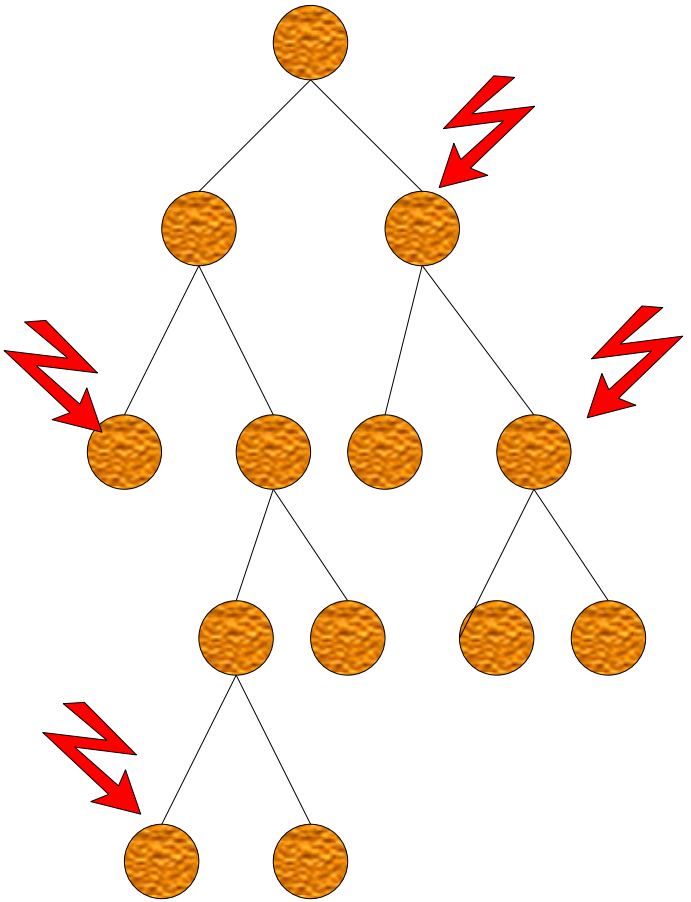
Multi-core machines



Quad-core machines

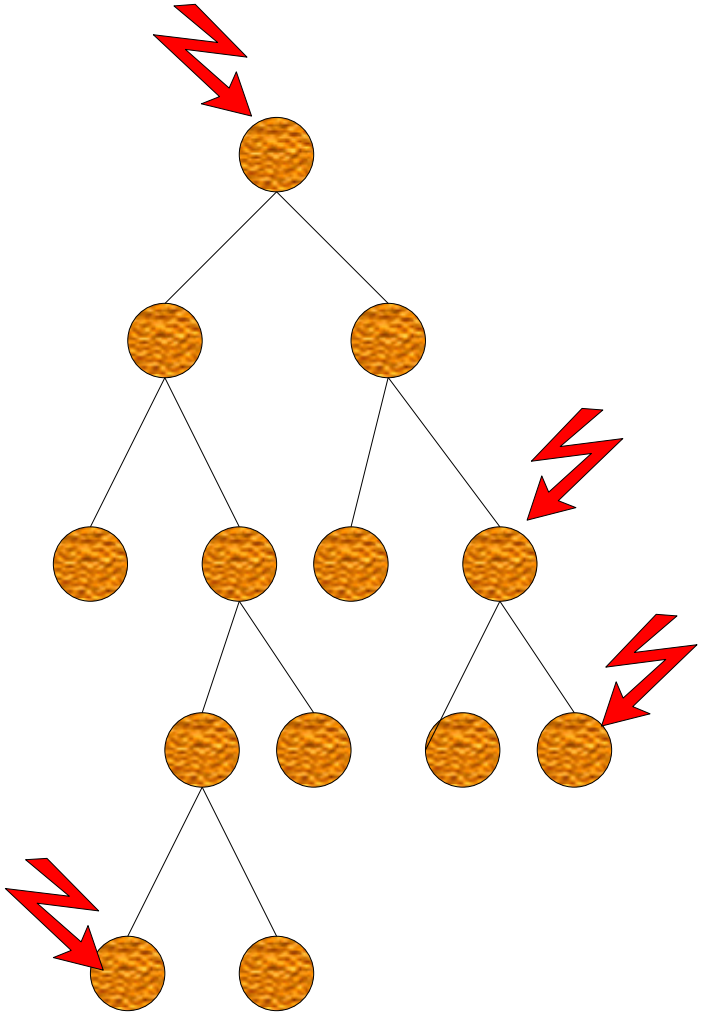


Quad-core machines



===> pas équivalent au séquentiel

Quad-core machines



===> pas équivalent au séquentiel

Parallélisation à passage de messages

- Tous les 1/3 s. faire une moyenne entre tous les arbres:
  - Moyenne limitée aux noeuds ayant au moins 5% du total de simus
  - Moyenne limitée aux noeuds pas trop profonds
- Etonnament bonnes performances surtout en 19x19.

Biblio

- Bandits: Lai, Robbins, Auer, Cesa-Bianchi...
- UCT: Kocsis, Szepesvari, Coquelin, Munos...
- MCTS (Go): Coulom, Chaslot, Fiter, Gelly, Hoock, Silver, Muller, Pérez, Rimmel, Wang...
- Tree + DP for industrial applicationl: Pére
- Bandits with infinitely many arms: Audibert, Coulom, Munos, Wang...
- Applications far from Go: Rolet, Teytaud (F), Rimmel, De Mesmay ...
- Links with “macro-actions”