# Récursivité et Paradigme Diviser pour Régner

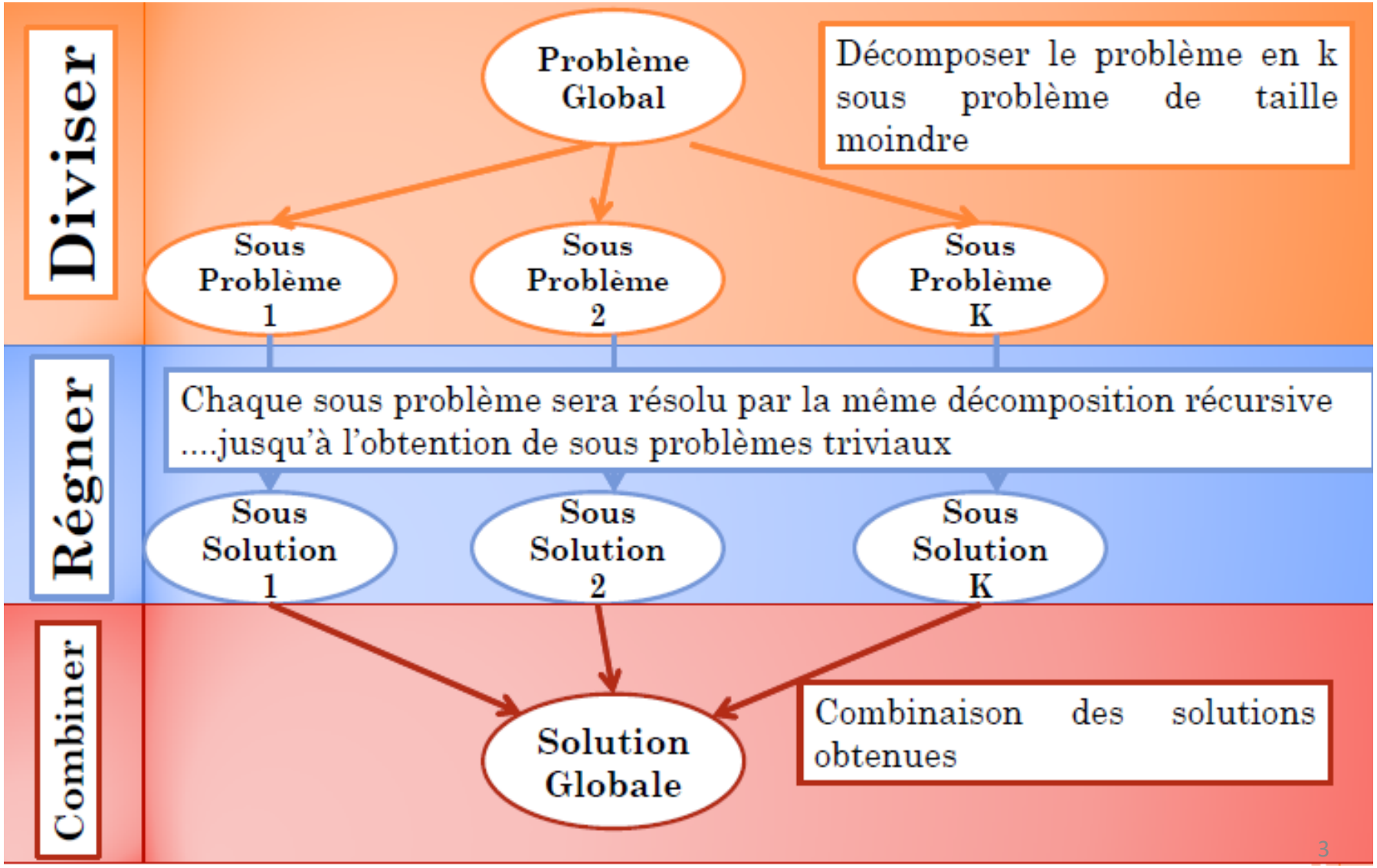
**Master : Cyber Sécurité et Intelligence Artificielle**
**CSIA**

**Année universitaire : 2025 / 2026**

# PARTIE 2: Diviser pour régner

- La récursivité est un outils puissant permet de concevoir des solutions (simples) sans se préoccuper des détails algorithmiques internes
- *Paradigme Diviser pour Régner: spécifier la solution du problème en fonction de la (ou des) solution(s) d'un (ou de plusieurs) sous-problème plus simple(s).*
- résolvent les sous problèmes de manière récursive puis combinent les résultats pour trouver une solution au problème initial

# Principe

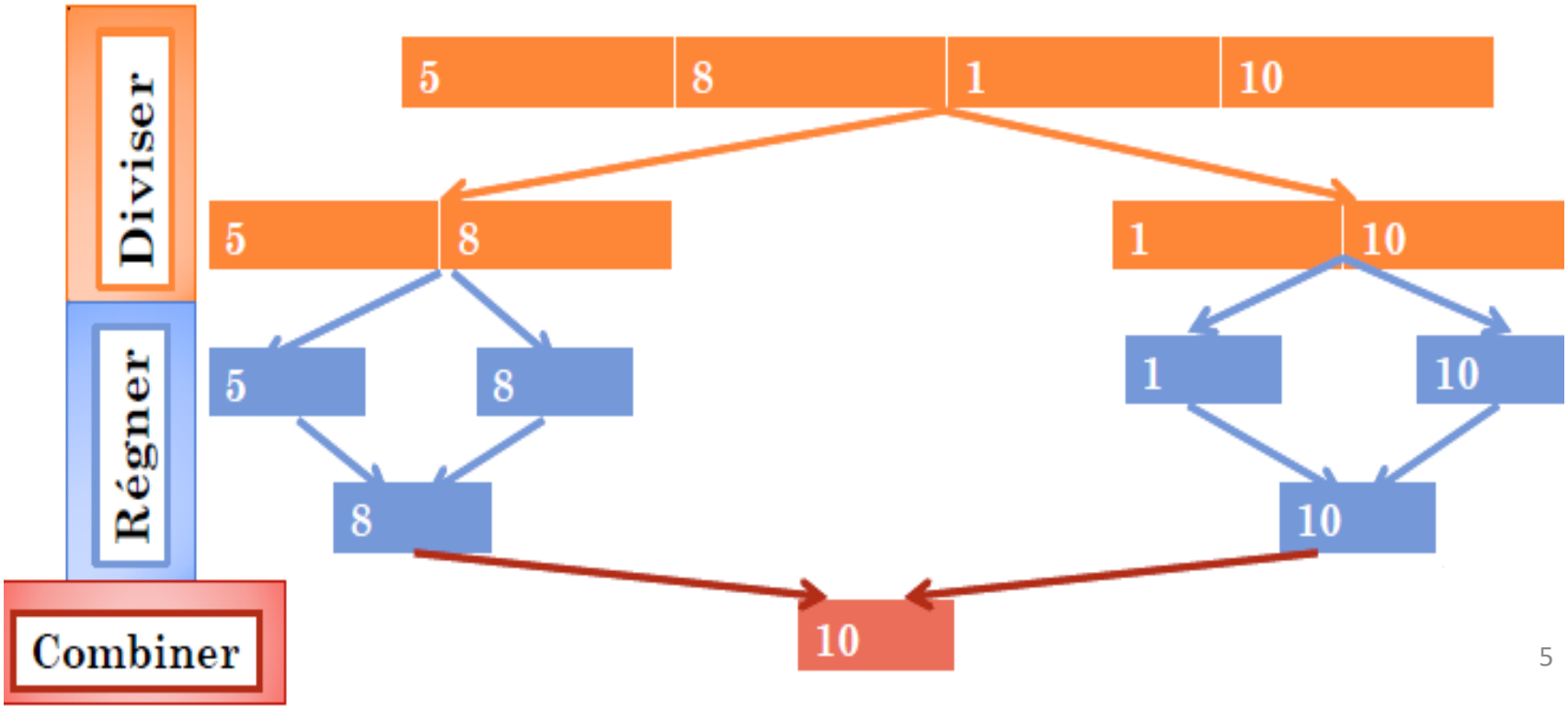

Diviser
Problème Global
Décomposer le problème en k sous problème de taille moindre
Sous Problème 1
Sous Problème 2
Sous Problème K
Régner
Chaque sous problème sera résolu par la même décomposition récursive ....jusqu'à l'obtention de sous problèmes triviaux
Sous Solution 1
Sous Solution 2
Sous Solution K
Combiner
Solution Globale
Combinaison des solutions obtenues

# Principe

- Le paradigme « diviser pour régner » donne lieu à trois étapes à chaque niveau de récursivité :
- **Diviser :** le problème en un certain nombre de sous-problèmes ;
- **Régner :** sur les sous-problèmes en les résolvant récursivement ou, si la taille d'un sous-problème est assez réduite, le résoudre directement ;
- **Combiner :** les solutions des sous-problèmes en une solution complète du problème initial.

# EXEMPLE 1 : RECHERCHE MAXIMUM

- Soit Tab un tableau à n éléments, écrire une fonction récursive permettant de rechercher de l'indice du maximum dans Tab en utilisant le paradigme diviser pour régner.



# EXEMPLE 1 : RECHERCHE MAXIMUM

Fonction maximum ( Tab , indDeb, indFin) **T(n)**

Si ( indDeb = indFin) alors retourner (indDeb)

Sinon

M←(indDeb+indFin)/2 **// division du problème en 2 sous-problèmes**

k1 ←maximum (Tab, indDeb, m )**// régner sur le 1er sous-problème** **T(n/2)**

K2←maximum (Tab, m+1, indFin)**// régner sur le 2ème sous-problème** **T(n/2)**

**// Combiner les solutions**

Si (Tab[k1] > Tab[k2]) alors retourner (k1)

Sinon retourner (k2)

Fsi

Fsi

Fin

**T(n) = 2 T(n/2) + c**

# EXEMPLE 2 : RECHERCHE DICHOTOMIQUE

- Soit Tab un tableau trié (ordre croissant) à n éléments.

```
Fonction RechDicho(Tab :Tableau, borneinf :entier, bornesup :entier, x :entier)
    : bool
Si (borneinf<=bornesup) alors
    mil ← (borneinf+bornesup) DIV 2 ;
    Si (Tab[mil]=x ) Alors retourner (vrai)
    Sinon
        Si (Tab[mil]>x) Alors Retourner (RechDicho(Tab, borneinf, mil-1, x))
        Sinon
         Retourner(RechDicho(Tab, mil+1, bornesup, x))
     Fsi
Sinon
  Retourner (Faux)
Fsi
```

**T(n) = T(n/2) + c**

# EXEMPLE 3 : multiplication de matrices

- Nous nous intéressons ici à la multiplication de matrices carrés de taille *n.*

MULTIPLIER-MATRICES(*A, B)*

Soit *n la taille des matrices carrés A et B*

Soit *C une matrice carré de taille n*

**Pour *i* ←*1 à n faire***

**Pour *j* ←*1 à n faire***

$c_{i;j} \leftarrow 0$

**Pour *k* ←*1 à n faire***

$c_{i;j} \leftarrow c_{i;j} + a_{i;k} * b_{k;j}$

**renvoyer *C***

Cet algorithme effectue O($n^3$) *multiplications et autant d'additions.*

# EXEMPLE 3 : multiplication de matrices

- Dans la suite nous supposerons que *n est une puissance exacte de 2. Décomposons les matrices A, B et C en* sous-matrices de taille *n/2 * n/2. L'équation C = AB peut alors se récrire :*

$$\begin{pmatrix} r & s \\ t & u \end{pmatrix} = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} e & g \\ f & h \end{pmatrix}$$

$$r = ae + bf, \quad s = ag + bh, \quad t = ce + df \quad \text{et} \quad u = cg + dh.$$

# EXEMPLE 3 : multiplication de matrices

- Chacune de ces quatre opérations correspond à deux multiplications de matrices carrés de taille *n/2 et une addition* de telles matrices. À partir de ces équations on peut aisément dériver un algorithme « diviser pour régner » dont la complexité est donnée par la récurrence :

$$T(n) = 8\,T(n/2) + O(n^2)$$

l'addition des matrices carrés de taille *n/2 étant en* $O(n^2)$.

# Analyse des algorithmes « diviser pour régner »

- Le temps d'exécution d'un algorithme « diviser pour régner » se décompose suivant les trois étapes du paradigme de base :
- **Diviser:** le problème en a sous-problèmes chacun de taille 1/b de la taille du problème initial. Soit D(n) le temps nécessaire à la division du problème en sous-problèmes.
- **Régner:** Soit aT (n/b) le temps de résolution des a sous-problèmes.
- **Combiner:** Soit C(n) le temps nécessaire pour construire la solution finale à partir des solutions aux sous-problèmes.

$$\mathbf{T(n) = a\ T\ (n/b) + D(n) + C(n)}$$

- Soit la fonction f (n) la fonction qui regroupe D(n) et C(n). La fonction T(n) est alors définie :

$$\mathbf{T(n) = a\ T(n/b) + f(n)}$$

# THÉORÈME DE RÉSOLUTION DE LA RÉCURRENCE

Pour $\mathbf{f(n) = c\ n^k}$, on a : $\mathbf{T(n) = a.T(n / b) + c\ n^k}$

si $a > b^k \rightarrow T(n) = O(n^{\log_b a})$

si $a = b^k \rightarrow T(n) = O(n^k \log_b n)$

si $a < b^k \rightarrow T(n) = O(f(n)) = O(n^k)$

# Complexité de l'algorithme de recherche du maximum

$$T(n) = 2\,T(n/2) + c$$

$$\mathbf{a = 2\ ,\ b = 2,\ k = 0 \rightarrow a > b^k}$$

$$\log_a a = 1$$

$$\mathbf{T(n) = O(n)}$$

# Complexité de l'algorithme de recherche dichotomique:

$$T(n) = T(n/2) + c$$

$$\mathbf{a = 1\ ,\ b = 2,\ k = 0 \rightarrow a = b^k}$$

$$\mathbf{T(n) = O(log(n))}$$

# AUTRES RÉSOLUTIONS DE RÉCURRENCE

$$T(n) = aT(n-1) + f(n) \quad \Rightarrow \quad T(n) = a^n \left( T(0) + \sum_{i=1}^{n} \frac{f(i)}{a^i} \right)$$

**Les Tours de Hanoi**

$$T(n) = 2*T(n-1) + c$$

$$T(n) = 2^n \left( 0 + \sum_{i=1}^{n} \frac{c}{2^i} \right) = c\,2^n \left( \sum_{i=1}^{n} \frac{1}{2^i} \right) = c\,2^n (1 - 2^{-n}) = c\,(2^n - 1)$$

**$T(n) = O(2^n)$**

# AUTRES RÉSOLUTIONS DE RÉCURRENCE

- Équations de récurrence linéaires sans second membre (f(n) = cte)

$$T(n) = a_1\,T(n-1) + a_2\,T(n-2) + a_3\,T(n-3) + \cdots + a_k\,T(n-k) + cst$$

- A une telle équation, on peut associer un polynôme:

$$P(x) = x^k - a_1 x^{k-1} - a_2 x^{k-2} - \ldots - a_k$$

- La résolution de ce polynôme nous donne m racines ri ( avec m<=k)
- La solution de l'équation de récurrence est ainsi donnée par :

$$T(n) = c_1 r_1^n + c_2 r_2^n + c_3 r_3^n + \ldots + c_m r_m^n$$

- Cette solution est en général exponentielle