

Complexité CM7

Antonio E. Porreca
aeporreca.org

**Précédemment dans
Complexité...**

Proposition 2-AO (p. 56)

Caractérisation existentielle de NP

Les deux conditions suivantes sont **équivalentes** :

- Le langage L est reconnu en temps polynomiale par une machine de Turing **non déterministe** N
- Il existe une machine de Turing **déterministe** M qui fonctionne en temps polynomial et un polynôme $p(n)$ tels que

$$x \in L \text{ ssi } \exists y \in \{0,1\}^{p(n)} M(x, y) \text{ accepte}$$

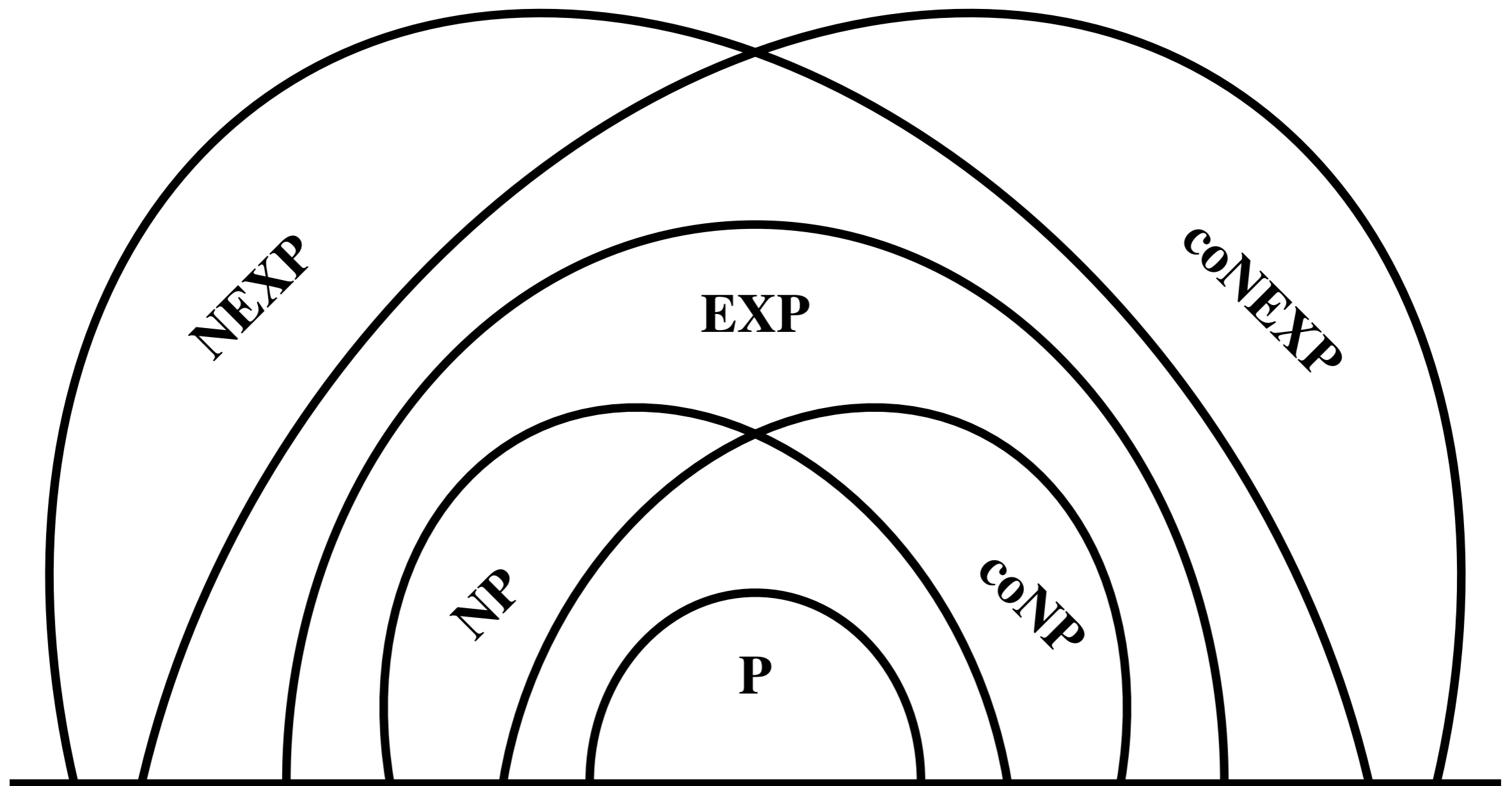
Le mot y qui justifie l'appartenance de x à L est appelé **preuve** ou **certificat**

P vs NP

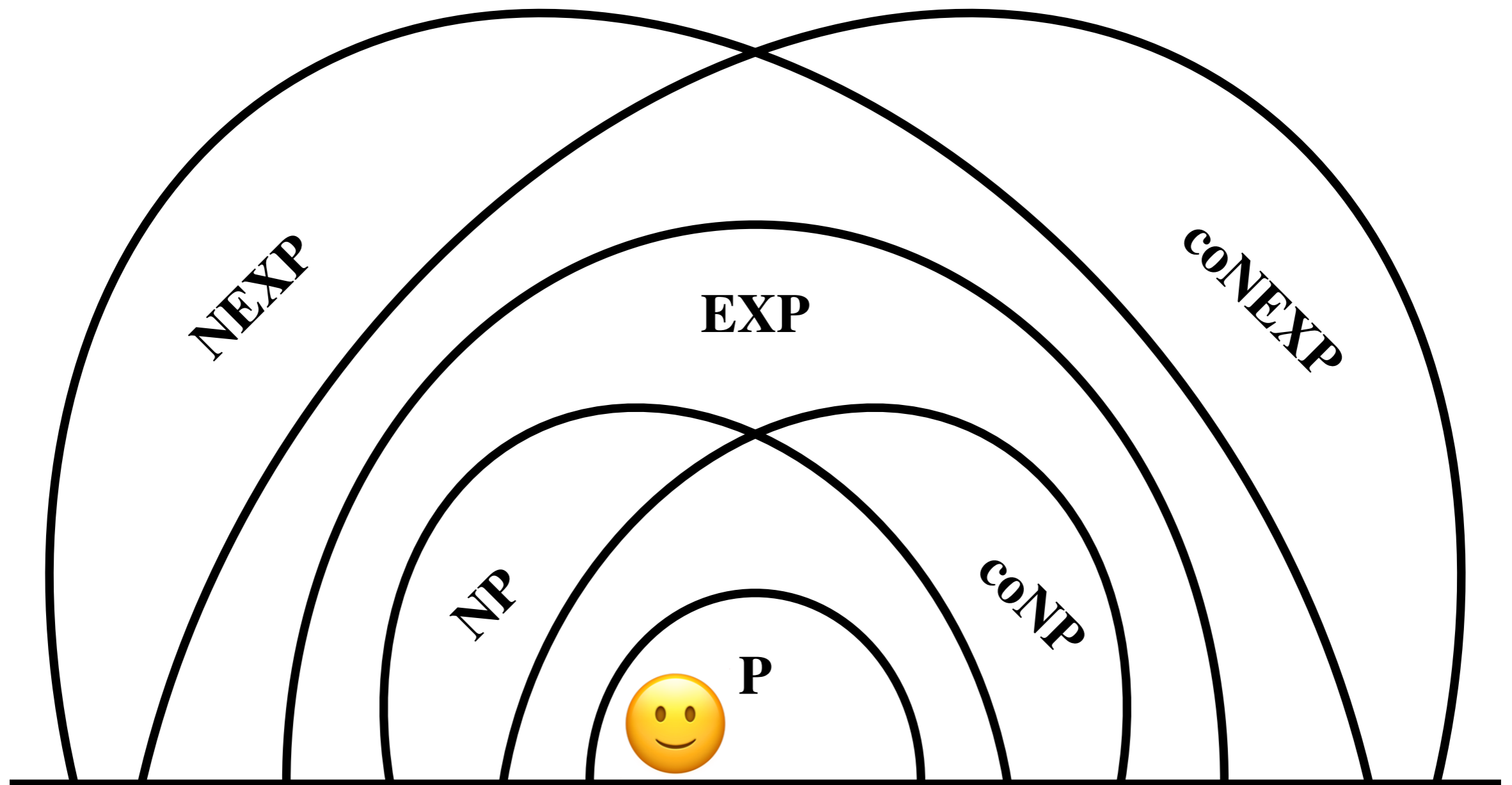
- **P** est la classe des problèmes **faciles à résoudre** (de façon déterministe)
- **NP** est la classe des problèmes avec des solutions **faciles à vérifier** (de façon déterministe)
- Facile à résoudre implique facile à vérifier, donc **$P \subseteq NP$**
- On pense que facile à vérifier n'implique nécessairement pas facile à résoudre, donc **$P \neq NP$**
- Mais ça reste un **problème ouvert**, d'où le 1000000 \$

Réductions, ou comment classer les problèmes par difficulté

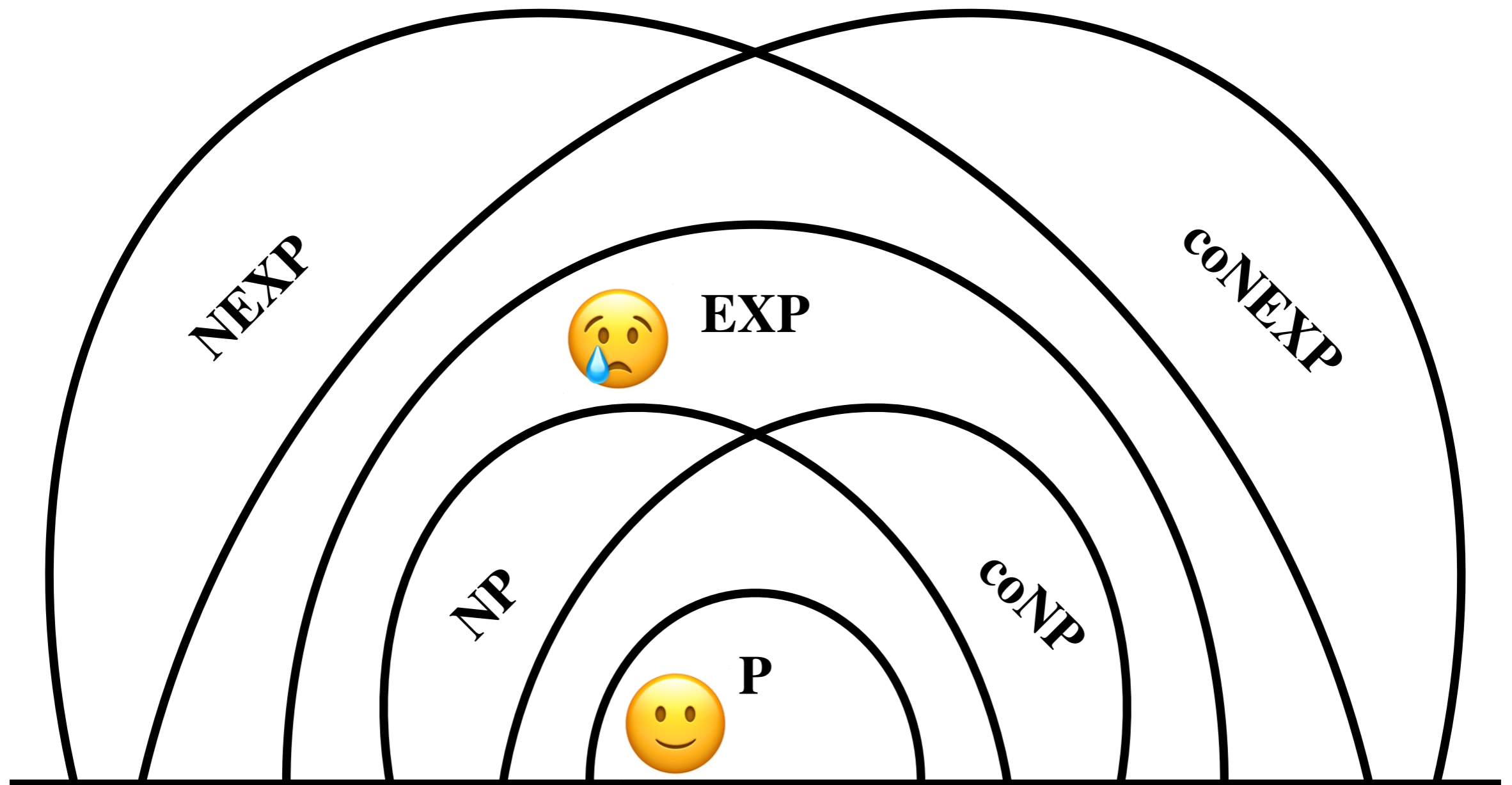
Hiérarchie des classes de complexité



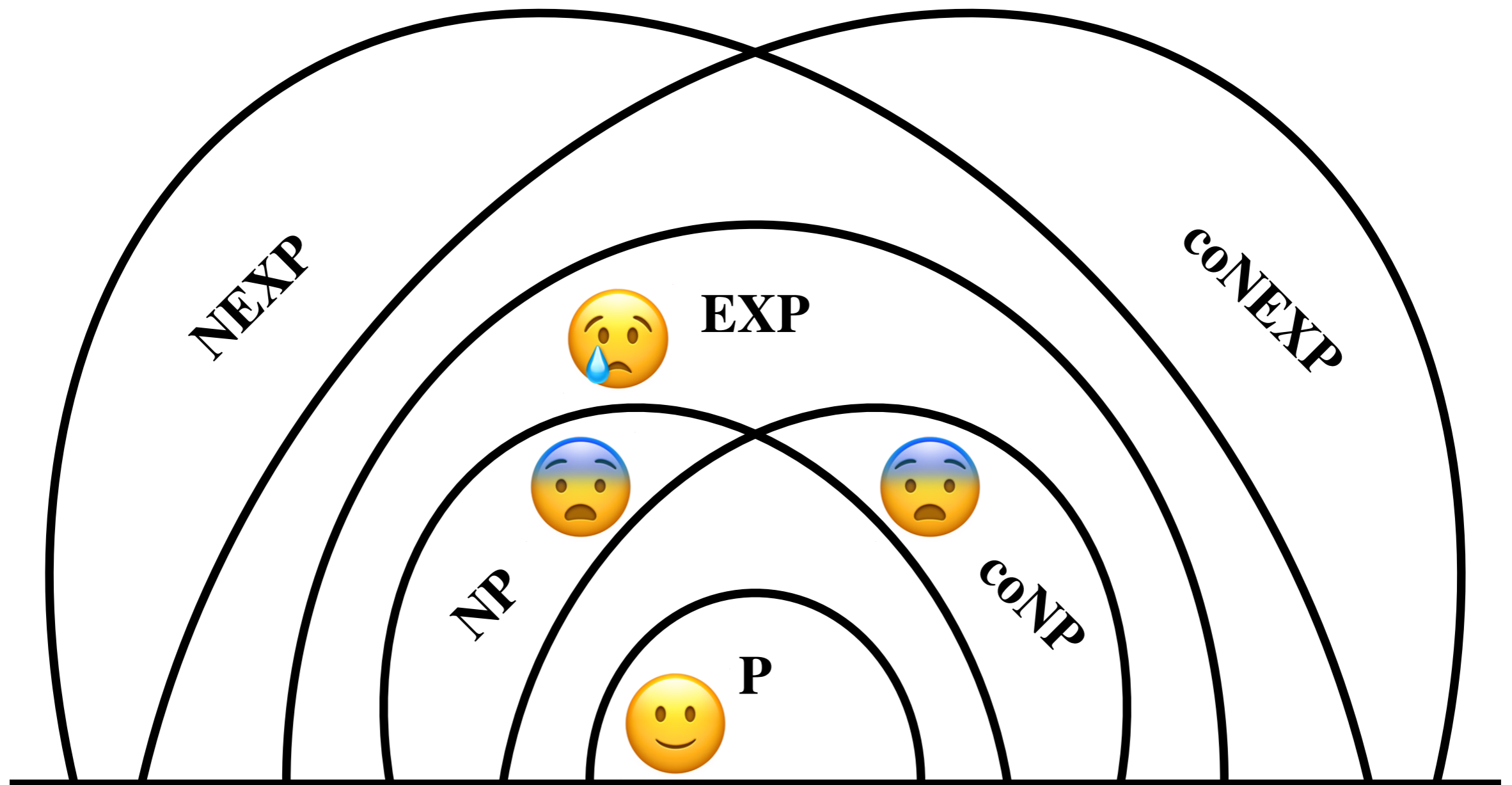
Hiérarchie des classes de complexité



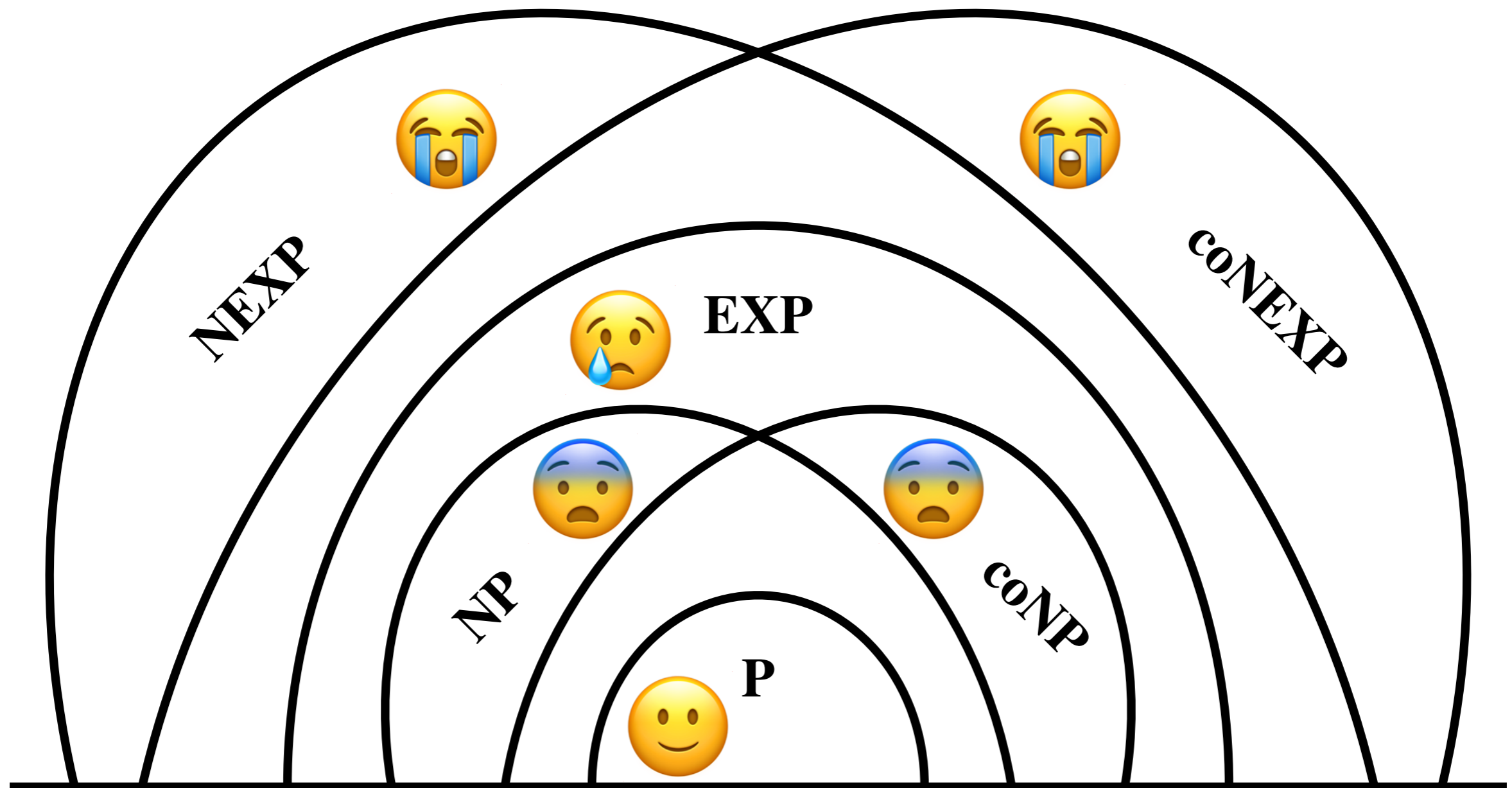
Hiérarchie des classes de complexité



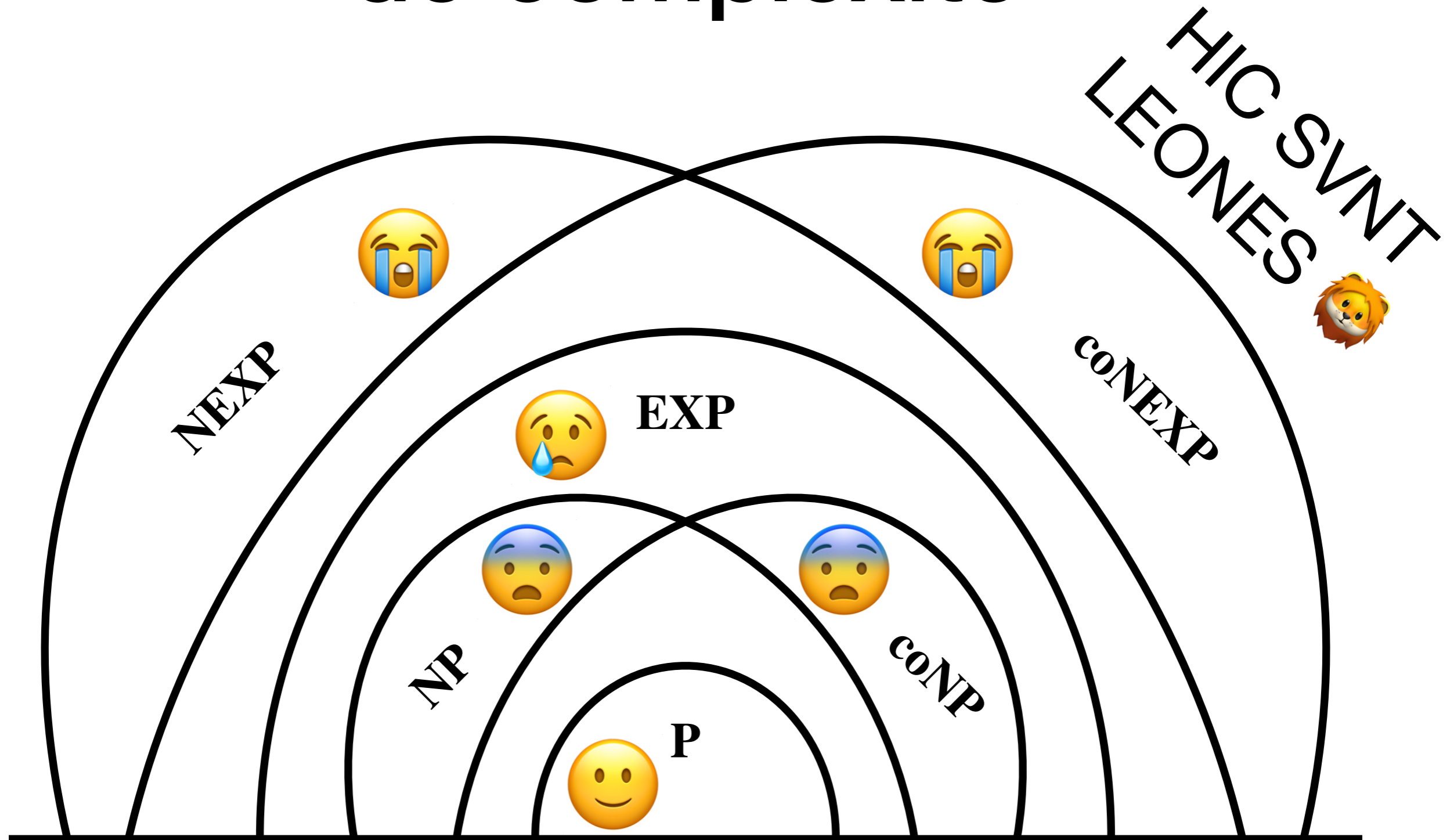
Hiérarchie des classes de complexité



Hiérarchie des classes de complexité



Hiérarchie des classes de complexité



Ordre de difficulté des problèmes

- On veut **classer les problèmes** (et pas les algorithmes !)
en ordre de difficulté
- C'est quoi la **difficulté** ?
- On voudrait que ce soit liée **de quelque façon**
au temps de calcul

Ordre de difficulté des problèmes

- Par exemple, $L_1 \leq L_2$ si le meilleur algorithme (ou machine de Turing) pour L_1 est plus rapide du meilleur algorithme pour L_2
- Problème : souvent on ne sait pas quel est le meilleur algorithme pour un problème !

Alternative : réductions

- On dit que L_1 est plus facile (ou de la même difficulté) que L_2 si on peut reconnaître facilement L_1 quand on a un moyen de reconnaître L_2
- Si c'est le cas, résoudre L_2 nous permet de résoudre L_1 , donc L_1 n'est pas plus difficile que L_2
- Dit autrement, pour résoudre L_1 on se ramène à L_2 ...
- ...ou, plus formellement, L_1 se réduit à L_2

Définition 3-A (p. 64)

Réductions (many-one) polynomiales

- Une **réduction (many-one) en temps polynomial** d'un problème L_1 (sur l'alphabet Σ_1) à un problème L_2 (sur l'alphabet Σ_2) est une fonction $f: \Sigma_1^* \rightarrow \Sigma_2^*$ calculable en temps polynomial déterministe telle que

$$\forall x \in \Sigma_1^* \quad x \in L_1 \iff f(x) \in L_2$$

- Si une telle f existe, on dit que **L_1 se réduit à L_2** (via f) et on notera **$L_1 \leq_m^P L_2$** (ou parfois, en bref, **$L_1 \leq L_2$**)

Exemple 3-1 (p. 66)

Une réduction

Problème CLIQUE

- **Entrée** : un graphe non orienté G et un entier k
- **Question** : G a-t-il une clique de taille k ?

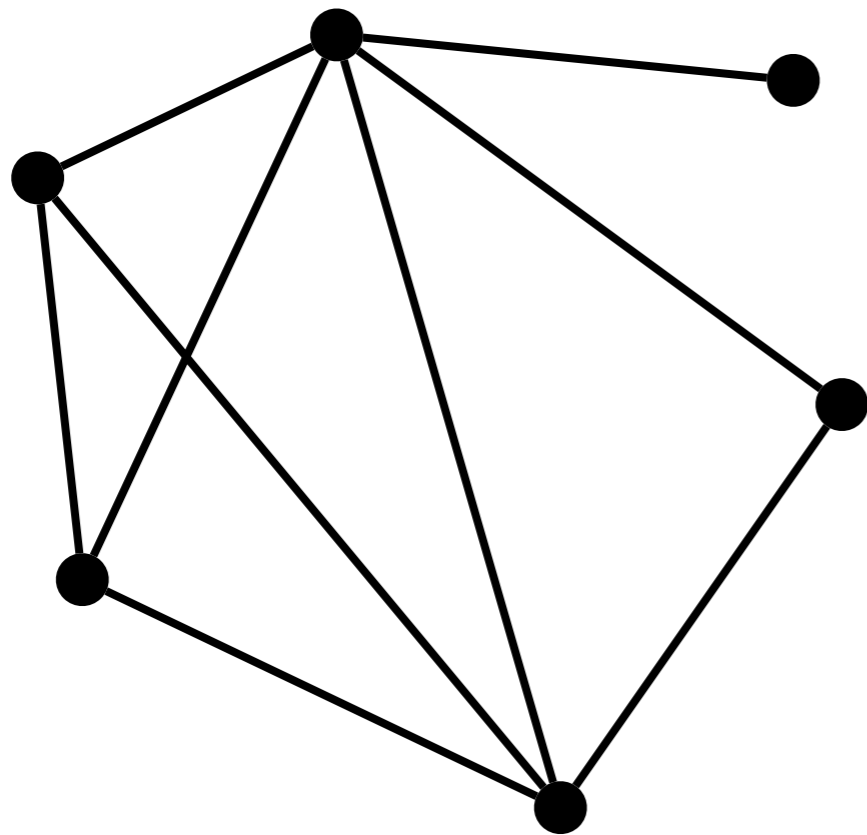
Problème

ENSEMBLE-INDÉPENDANT

- **Entrée** : un graphe non orienté G et un entier k
- **Question** : existe-t-il un ensemble de k sommets indépendants dans G , c'est-à-dire tous non reliés deux à deux ?

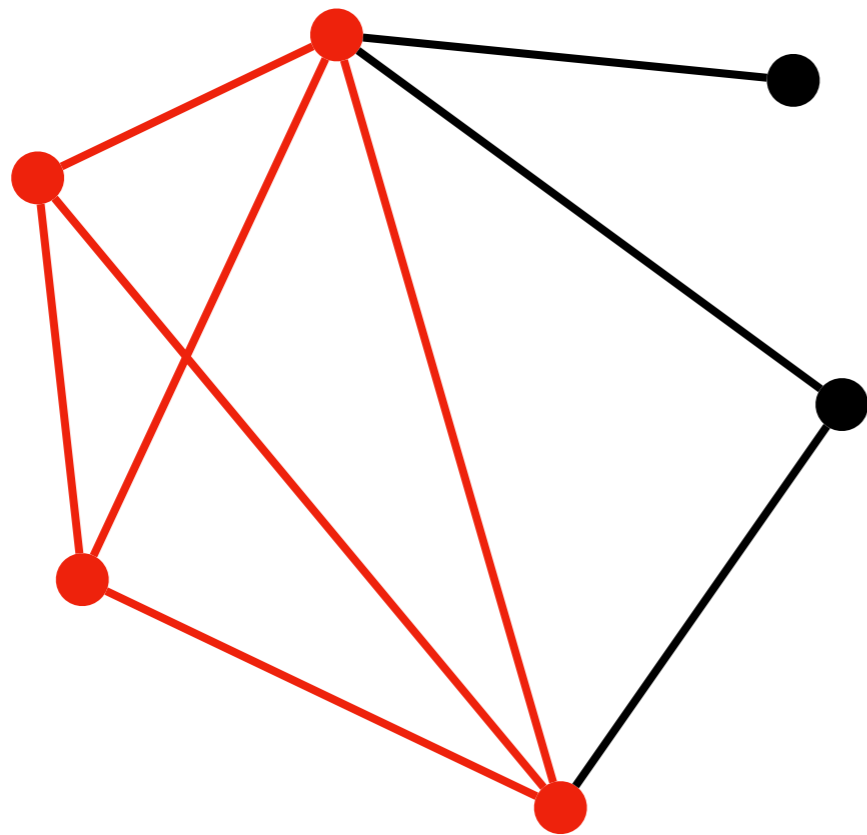
CLIQUE vs ENS-INDÉP

$$G = (V, E)$$



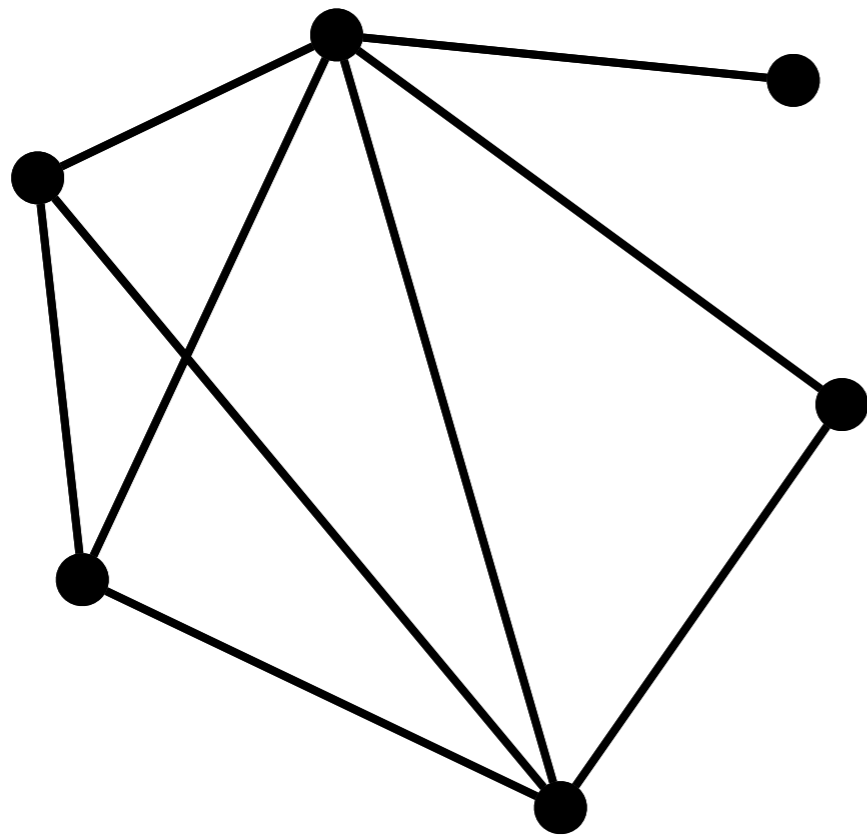
CLIQUE vs ENS-INDÉP

$$G = (V, E)$$



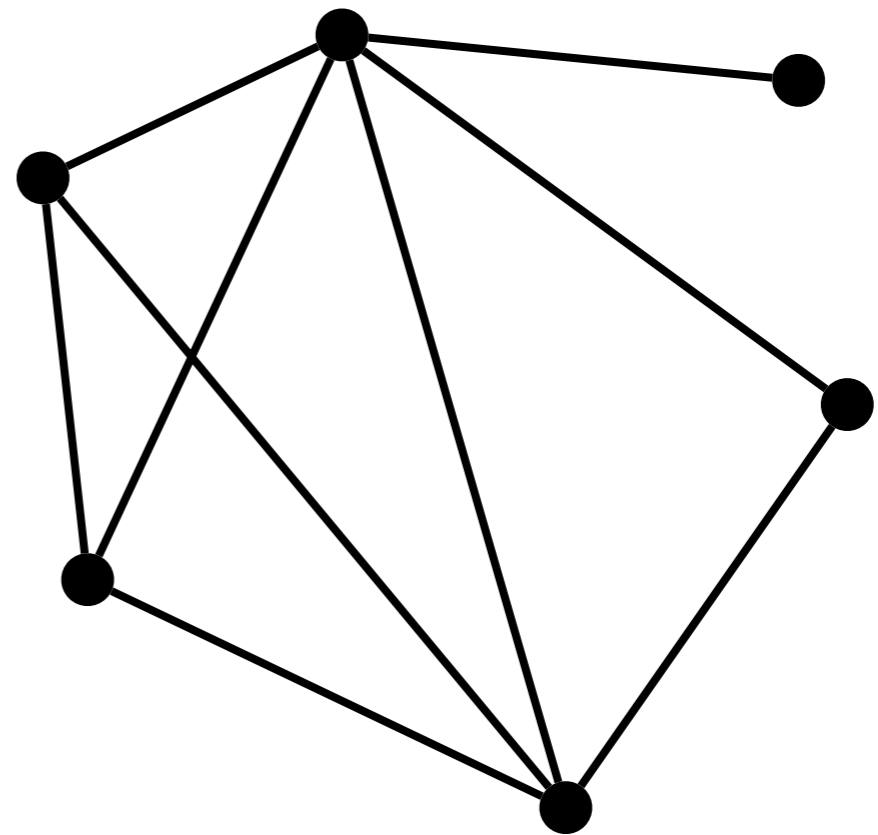
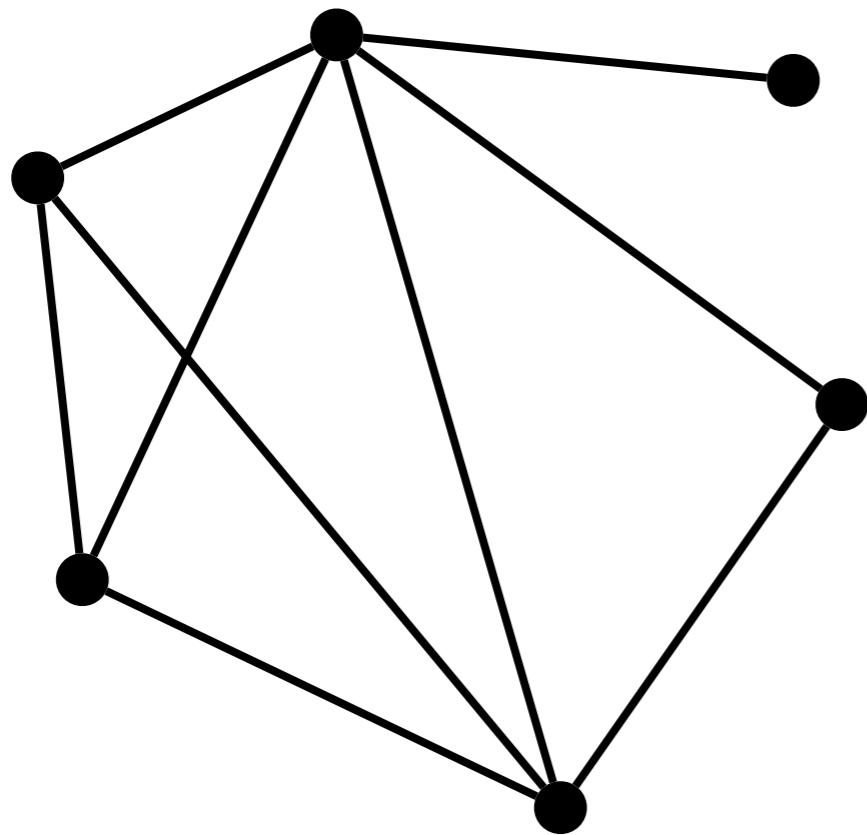
CLIQUE vs ENS-INDÉP

$$G = (V, E)$$



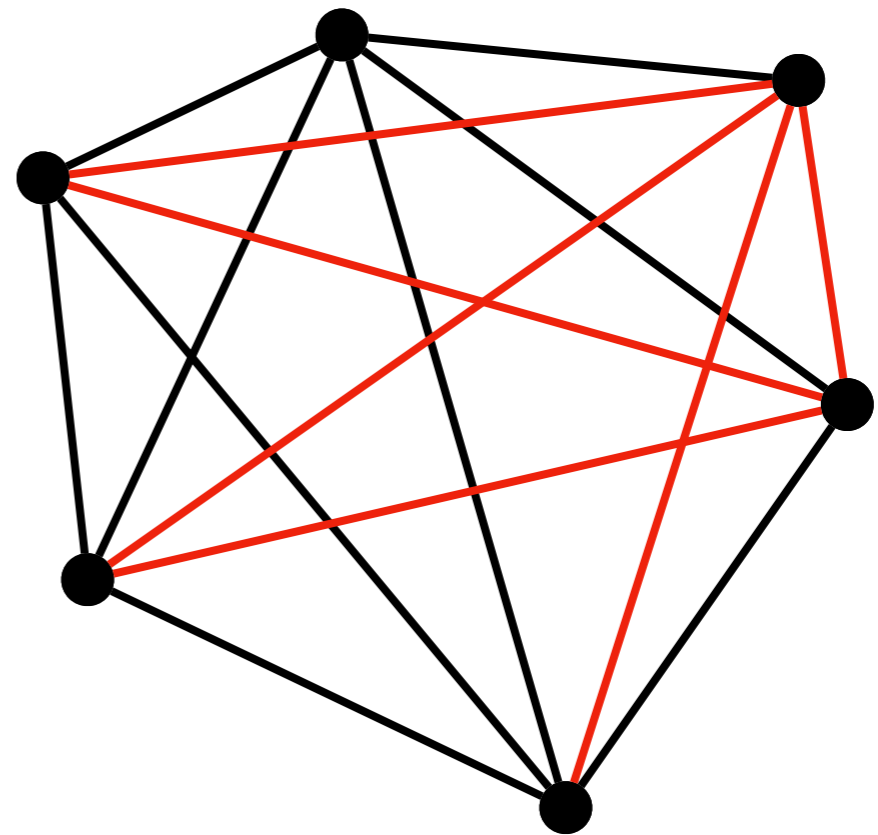
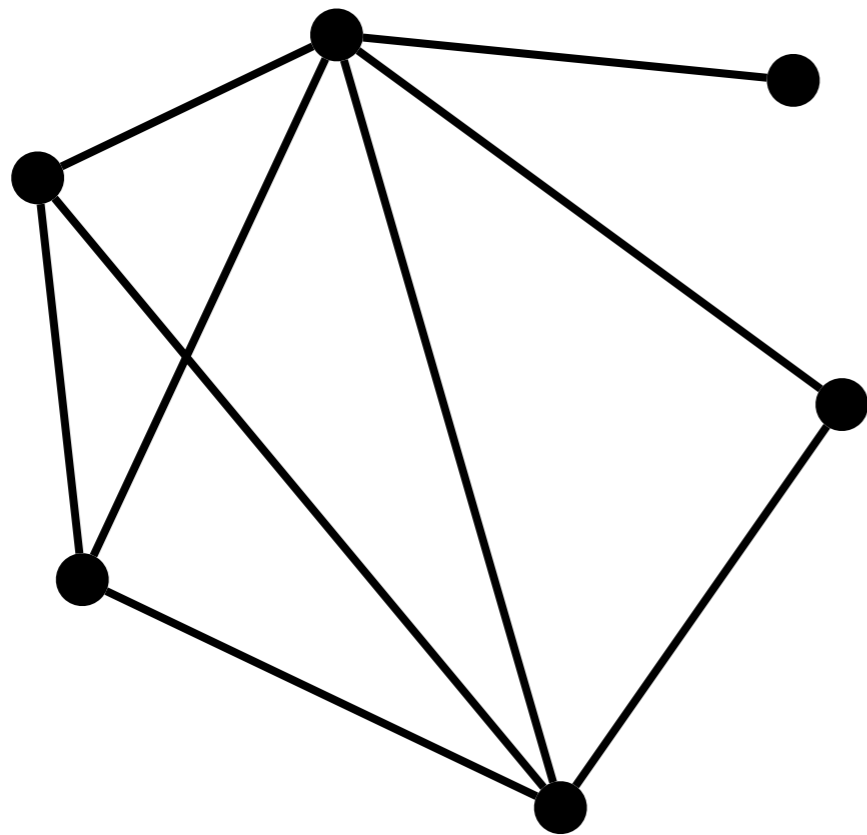
CLIQUE vs ENS-INDÉP

$$G = (V, E)$$



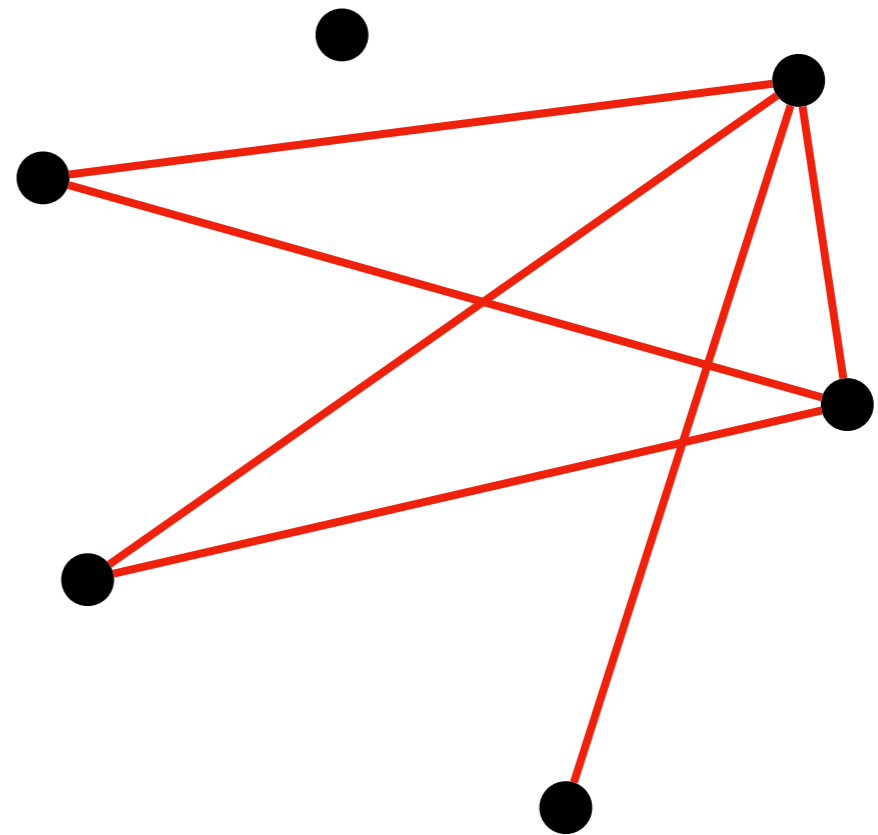
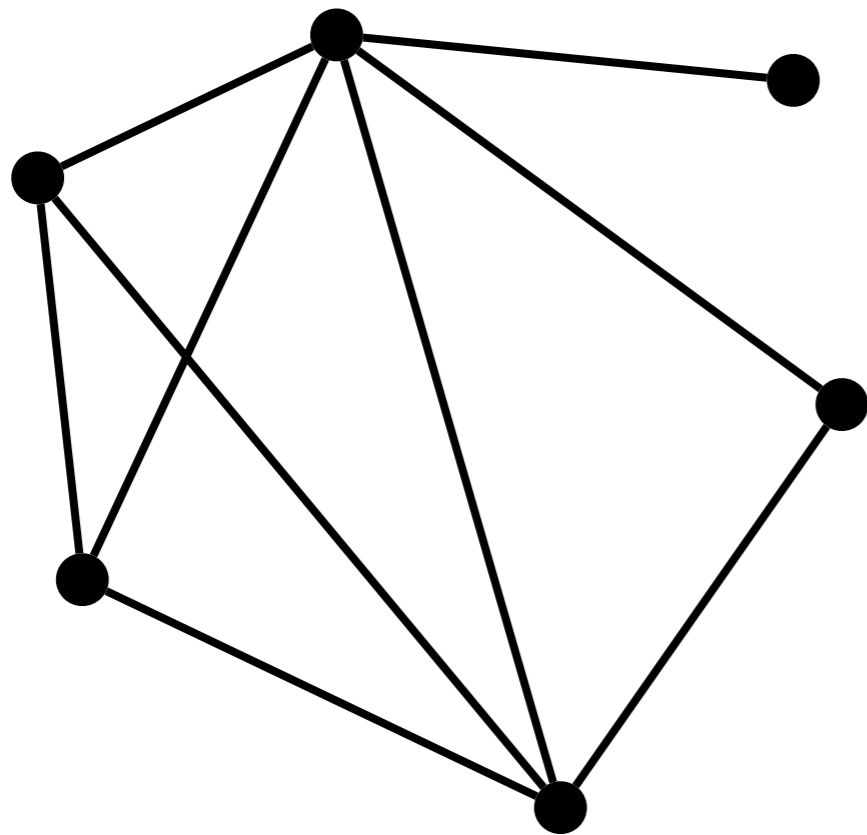
CLIQUE vs ENS-INDÉP

$$G = (V, E)$$



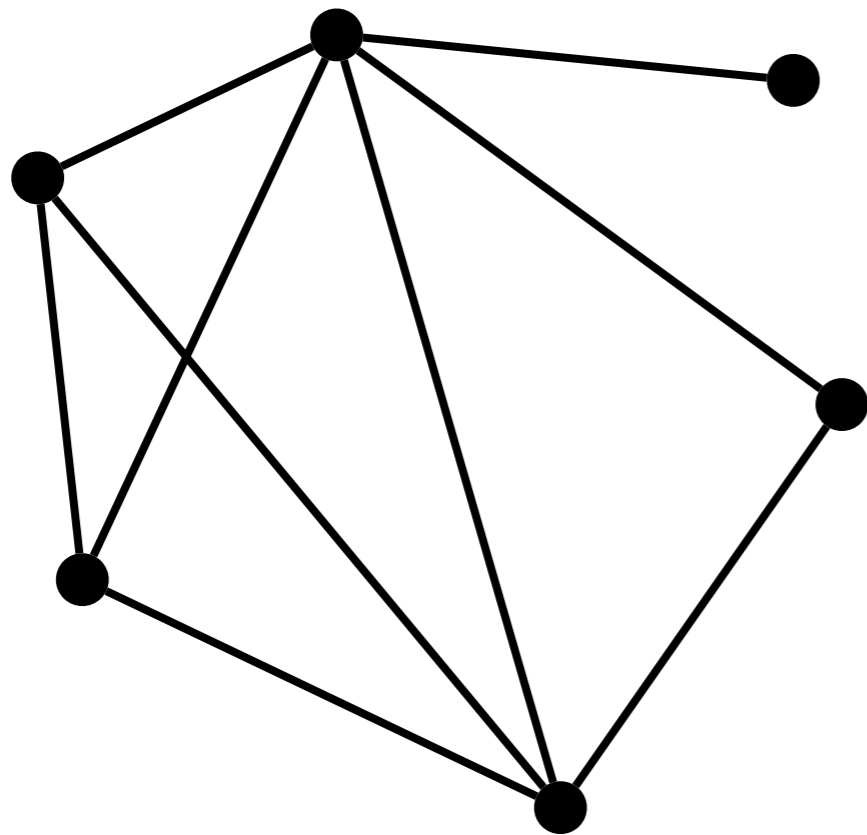
CLIQUE vs ENS-INDÉP

$$G = (V, E)$$

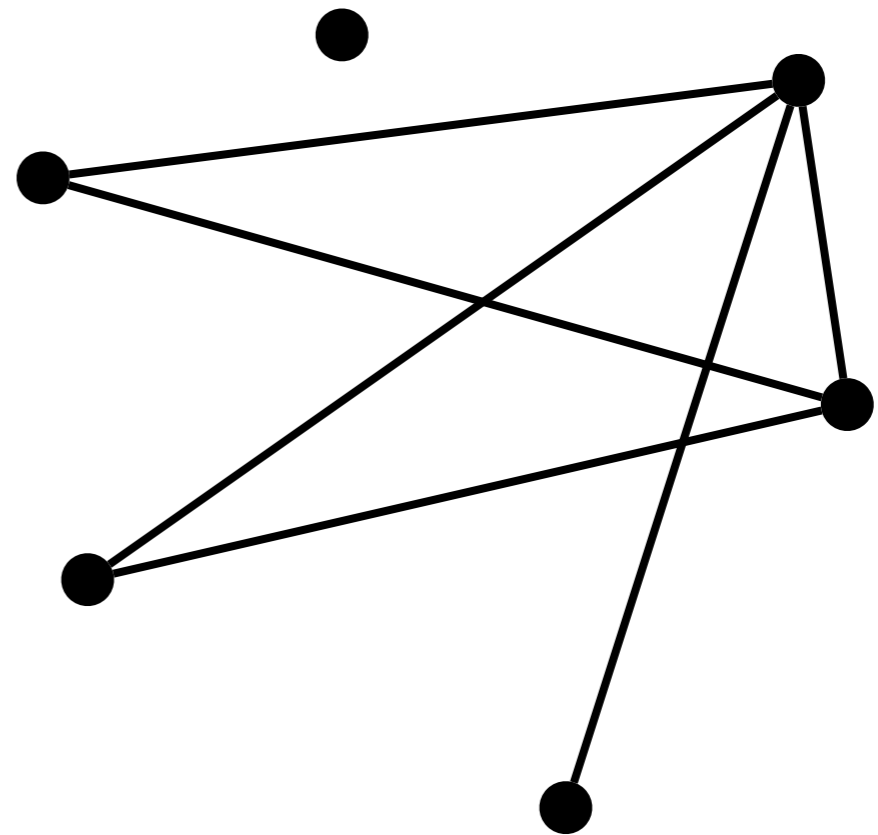


CLIQUE vs ENS-INDÉP

$$G = (V, E)$$

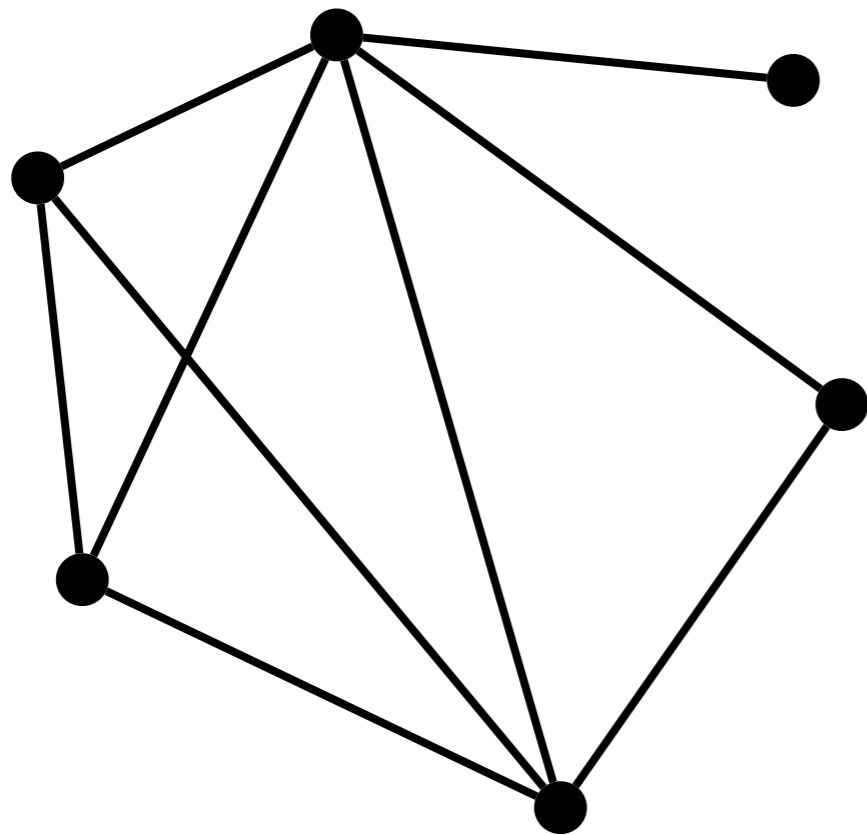


$$\bar{G} = (V, V^2 - E)$$



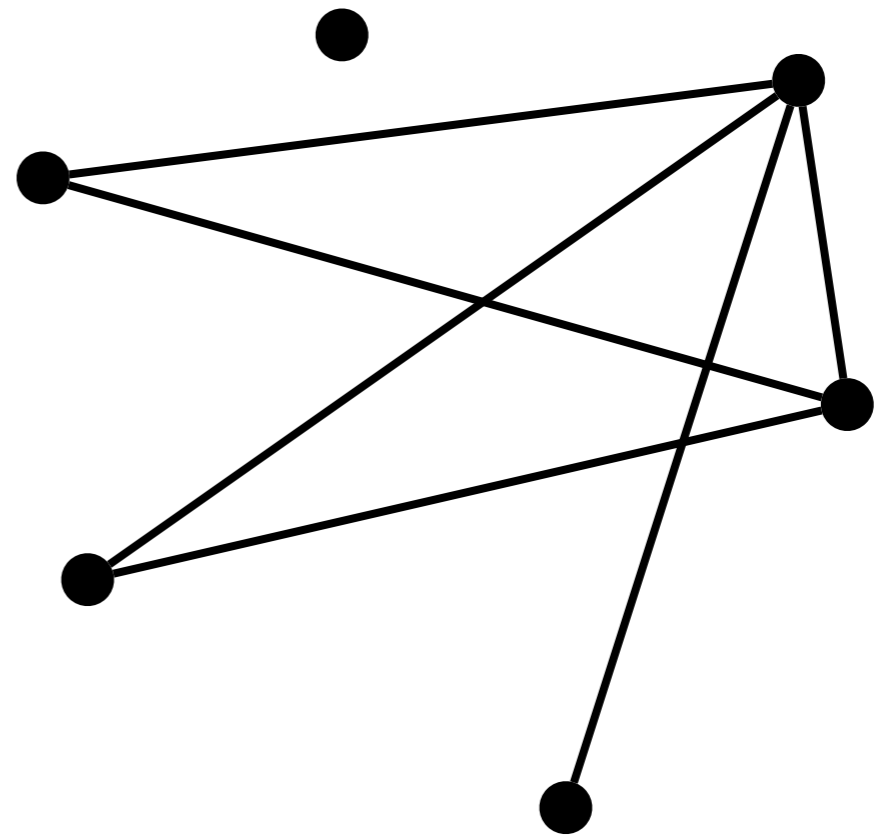
CLIQUE vs ENS-INDÉP

$$G = (V, E)$$



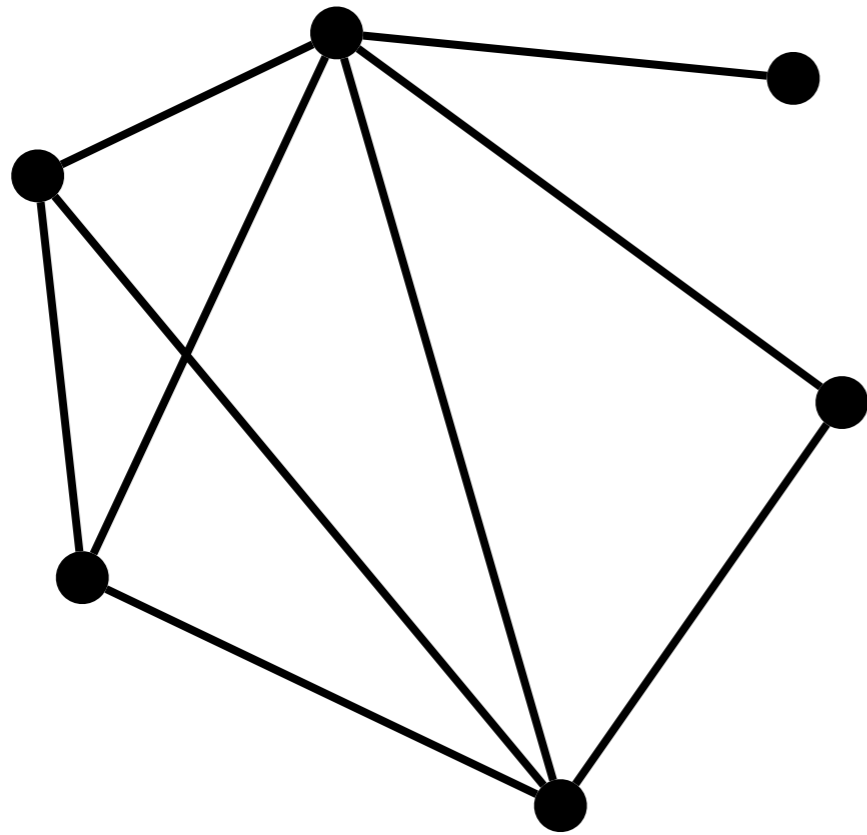
a une clique de taille k

$$\bar{G} = (V, V^2 - E)$$

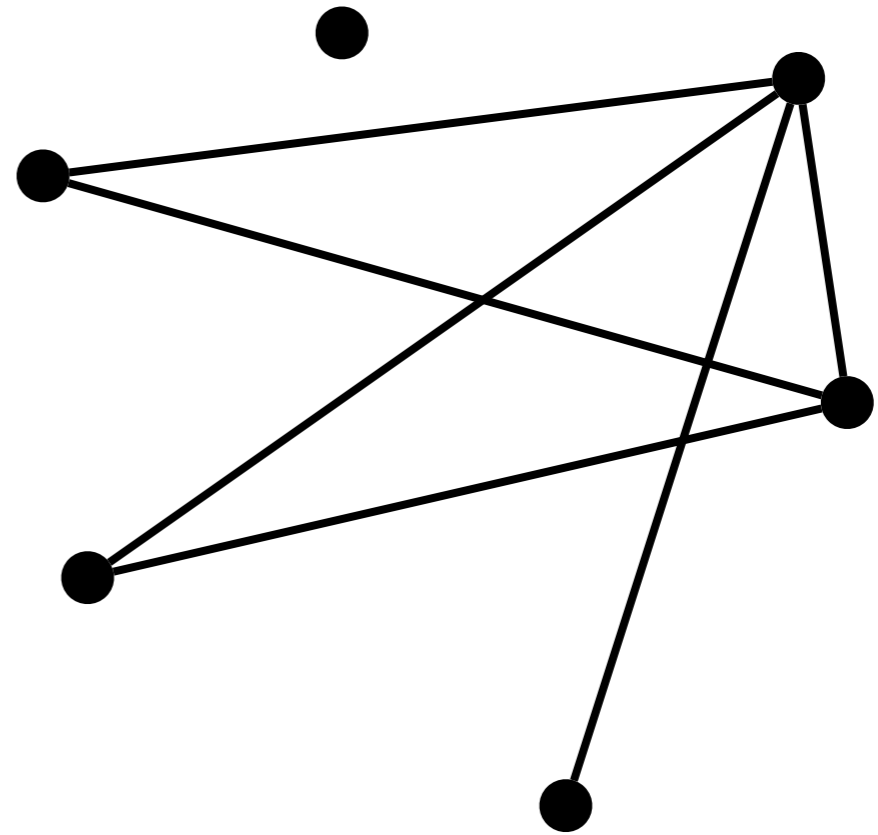


CLIQUE vs ENS-INDÉP

$$G = (V, E)$$



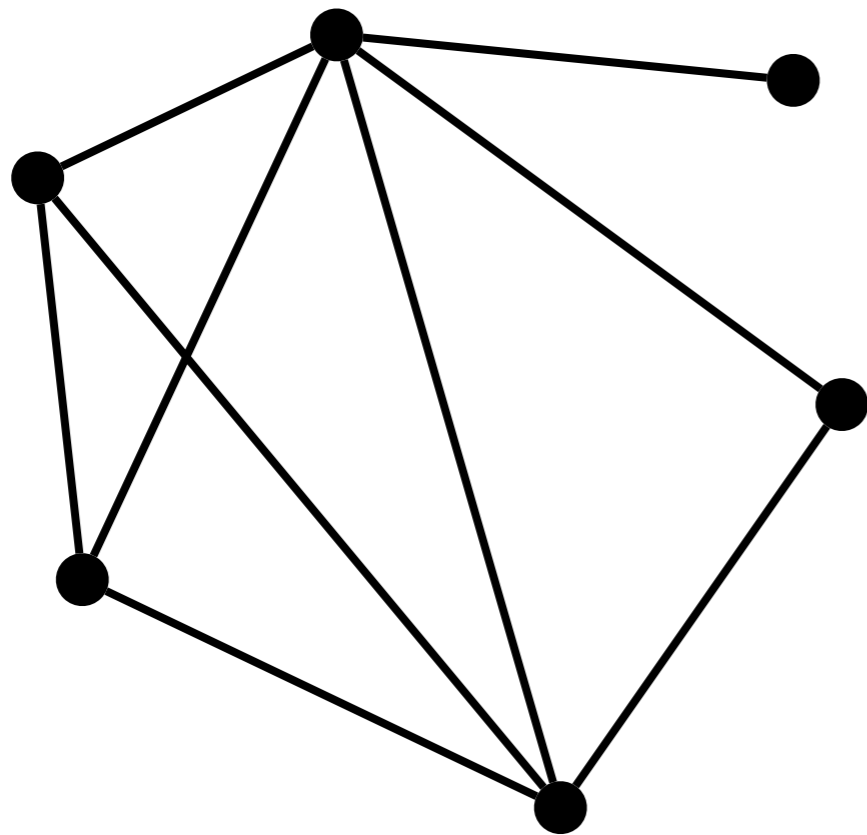
$$\bar{G} = (V, V^2 - E)$$



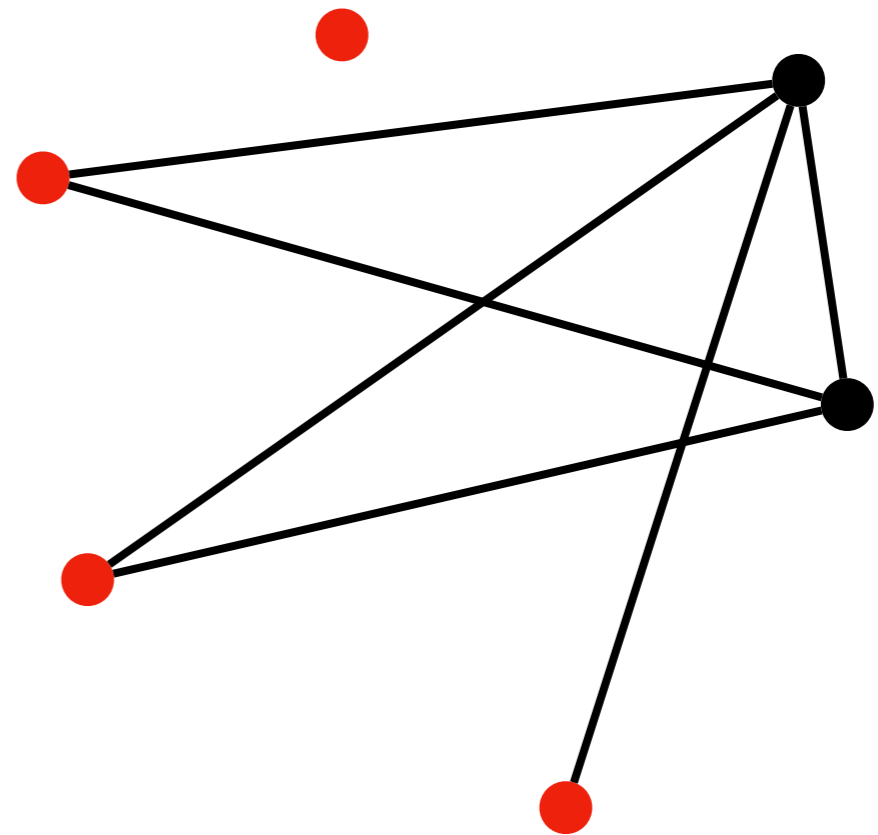
a une clique de taille $k \iff$ a un ens. indép. de taille k

CLIQUE vs ENS-INDÉP

$$G = (V, E)$$



$$\bar{G} = (V, V^2 - E)$$



a une clique de taille $k \iff$ a un ens. indép. de taille k

CLIQUE \leq ENS-INDÉP

- Supposons que CLIQUE et ENS-INDÉP sont définis sur le même alphabet Σ (raisonnable, c'est toujours des graphes !)
- La fonction $f: \Sigma^* \rightarrow \Sigma^*$ définie par $f(V, E, k) = (V, V^2 - E, k)$ est calculable en temps polynomial
- (V, E) a une clique de taille k ssi $f(V, E, k)$ a un ensemble indépendant de taille k
- Donc $(V, E, k) \in \text{CLIQUE}$ iff $f(V, E, k) \in \text{ENS-INDÉP}$

Mais aussi

ENS-INDÉP \leq CLIQUE !

- On utilise la même fonction $f: \Sigma^* \rightarrow \Sigma^*$ définie par $f(V, E, k) = (V, V^2 - E, k)$, toujours calculable en temps polynomial*
- (V, E) a un ensemble indépendant de taille k ssi $f(V, E, k)$ a une clique de taille k
- Donc $(V, E, k) \in \text{ENS-INDÉP}$ iff $f(V, E, k) \in \text{CLIQUE}$

* C'est un cas exceptionnel ! Normalement il faut changer de fonction

Lemme 3-G (p. 66)

\leq_m^P (pré-)ordonne les langages

- \leq est réflexive : $L_1 \leq L_1$ via l'identité $f: \Sigma_1^* \rightarrow \Sigma_1^*$, qui évidemment est calculable en temps polynomial
- \leq est transitive : soit $L_1 \leq L_2$ via $f: \Sigma_1^* \rightarrow \Sigma_2^*$ et $L_2 \leq L_3$ via $g: \Sigma_2^* \rightarrow \Sigma_3^*$
- alors $x \in L_1 \iff f(x) \in L_2$ et $f(x) \in L_2 \iff g(f(x)) \in L_3$, donc $x \in L_1 \iff g(f(x)) \in L_3$
- si f est calculable en temps polynomial $p(n)$
alors $|f(x)| \leq p(|x|) = p(n)$
- si g est calculable en temps polynomial $q(n)$, alors la fonction composée $g \circ f$ est calculable en temps polynomial $O(p(n) + q(p(n)))$

Définition 3-H (p. 66)

Problèmes équivalents

- Si $L_1 \leq_m^P L_2$ et $L_2 \leq_m^P L_1$ alors on écrit $L_1 \equiv_m^P L_2$
(parfois, en bref, $L_1 \equiv L_2$)
- On dit que les problèmes L_1 et L_2 sont **équivalents** pour les réductions (many-one) polynomiales
- Par exemple, on a toujours $L_1 \equiv L_1$

Résoudre L_1 avec une réduction

x



Résoudre L_1 avec une réduction

x



Résoudre L_1 avec une réduction



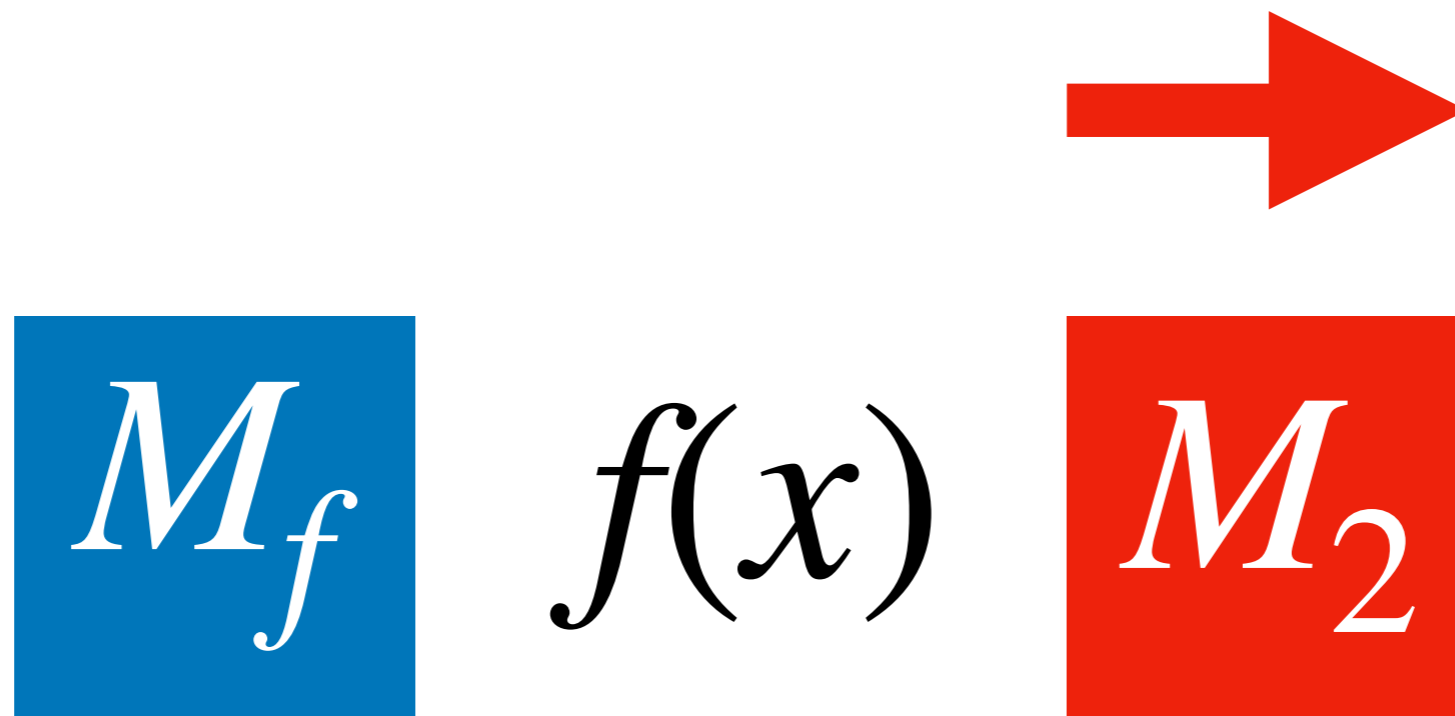
Résoudre L_1 avec une réduction



$f(x)$



Résoudre L_1 avec une réduction



Résoudre L_1 avec une réduction



oui

Résoudre L_1 avec une réduction



oui

non

Résoudre L_1 avec une réduction



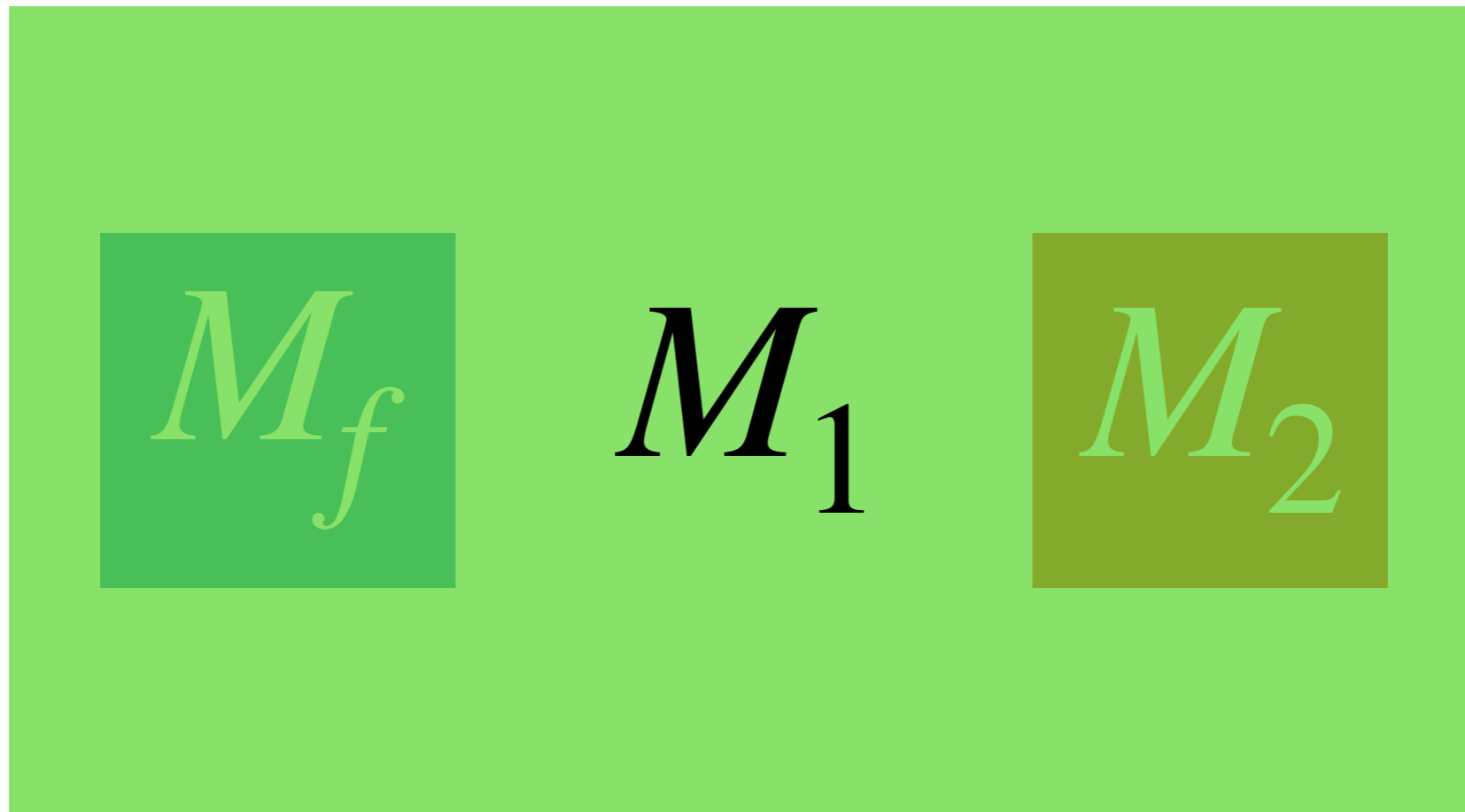
oui
non

$$\forall x \in \Sigma_1^* \quad x \in L_1 \iff f(x) \in L_2$$

Résoudre L_1 avec une réduction

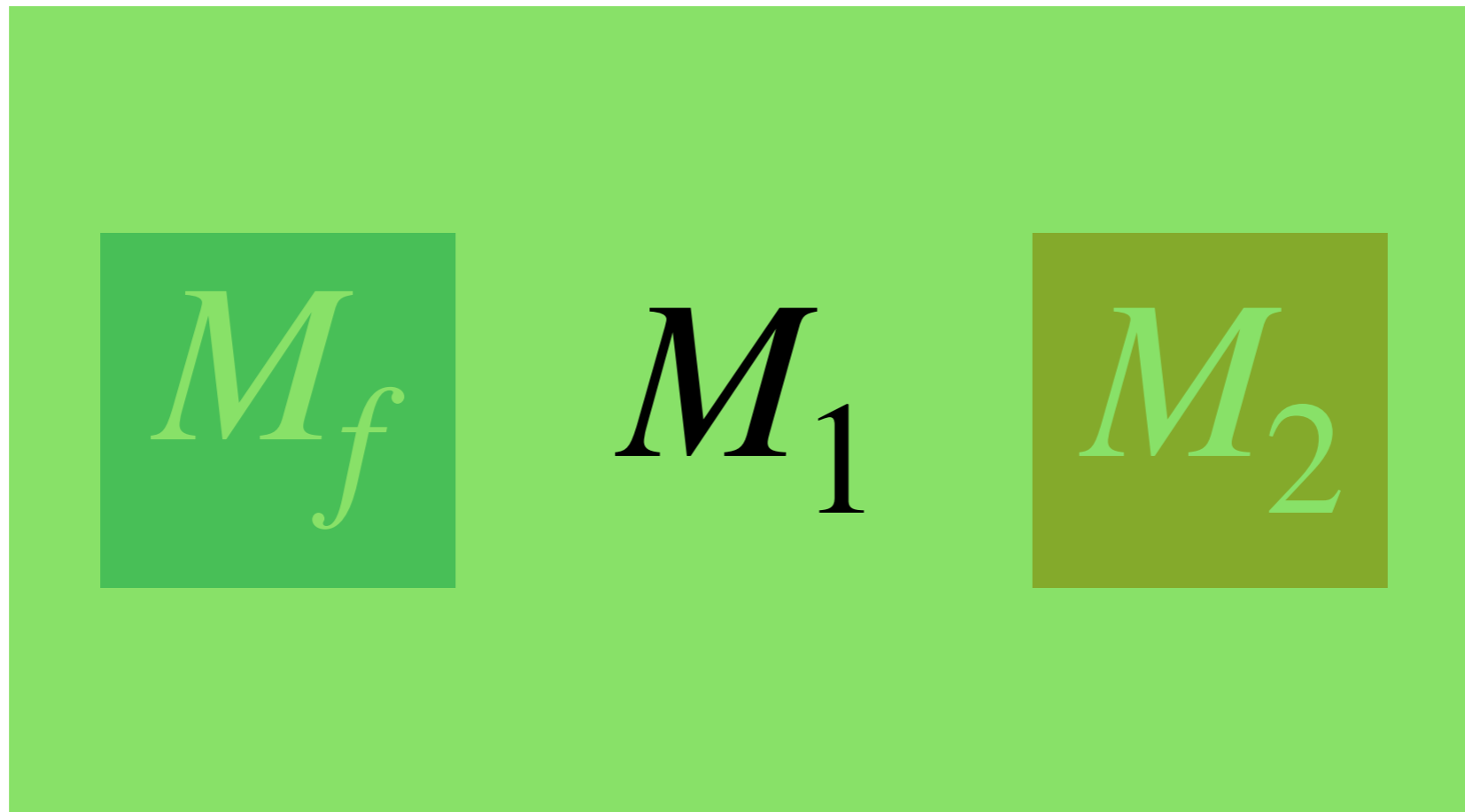


Résoudre L_1 avec une réduction

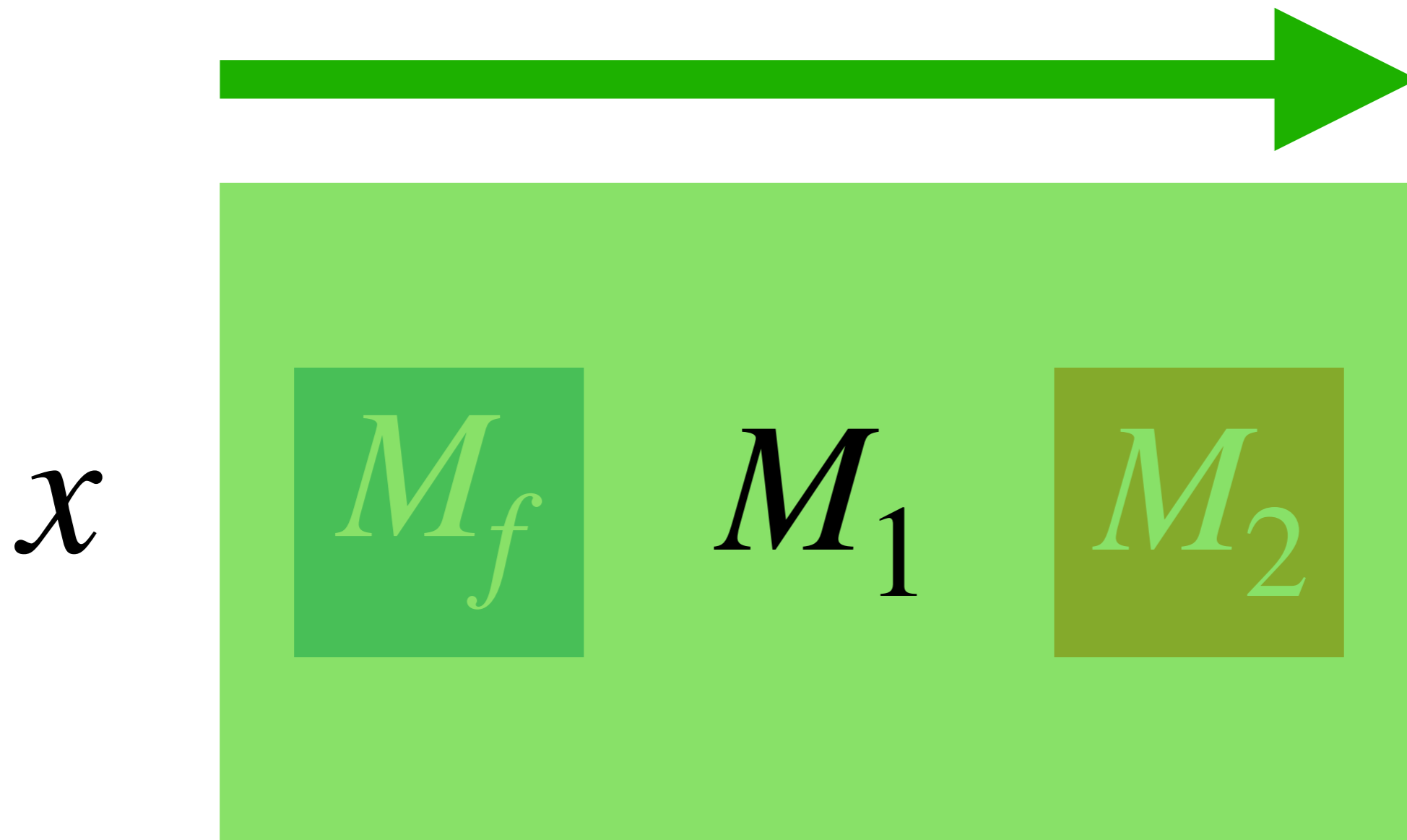


Résoudre L_1 avec une réduction

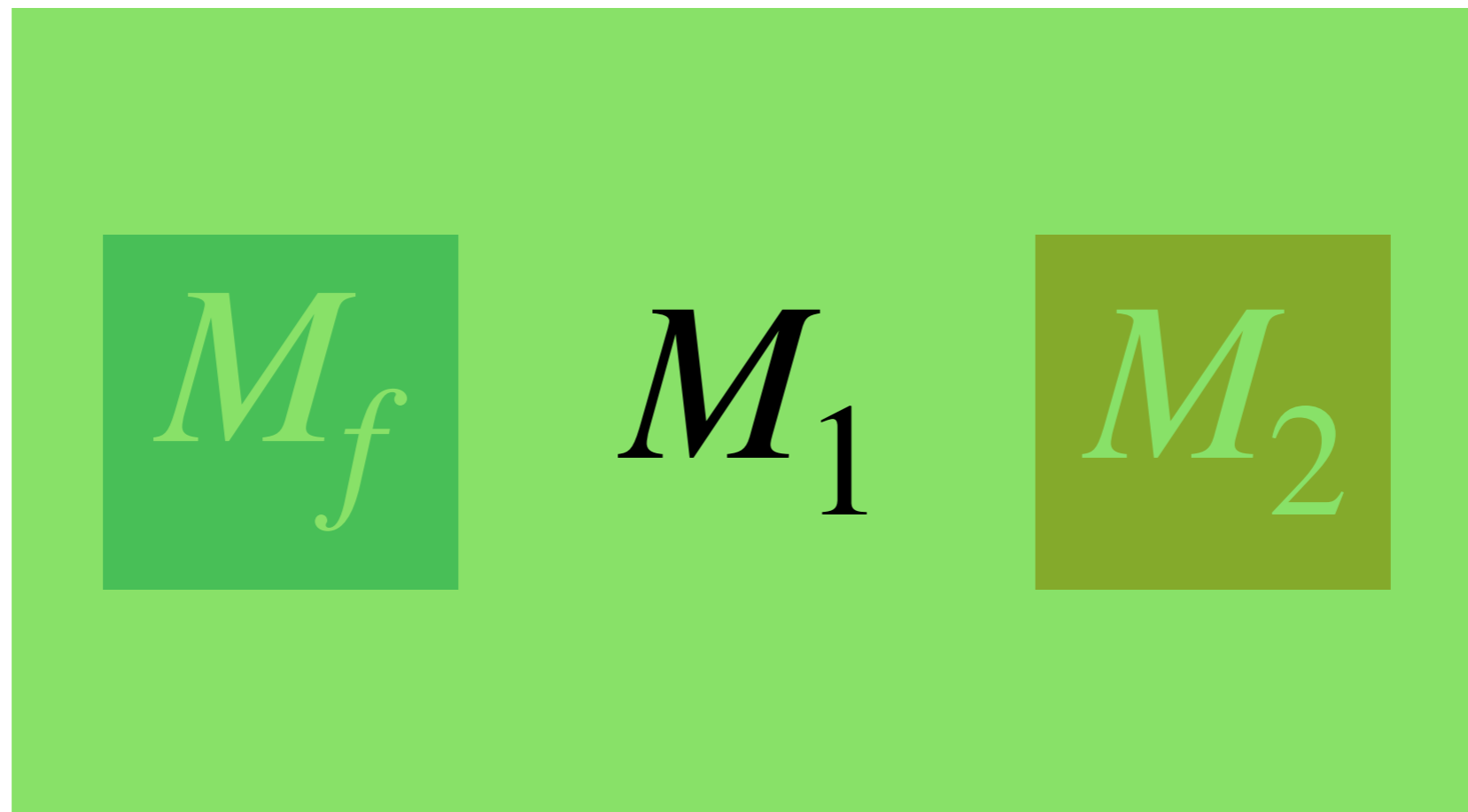
x



Résoudre L_1 avec une réduction

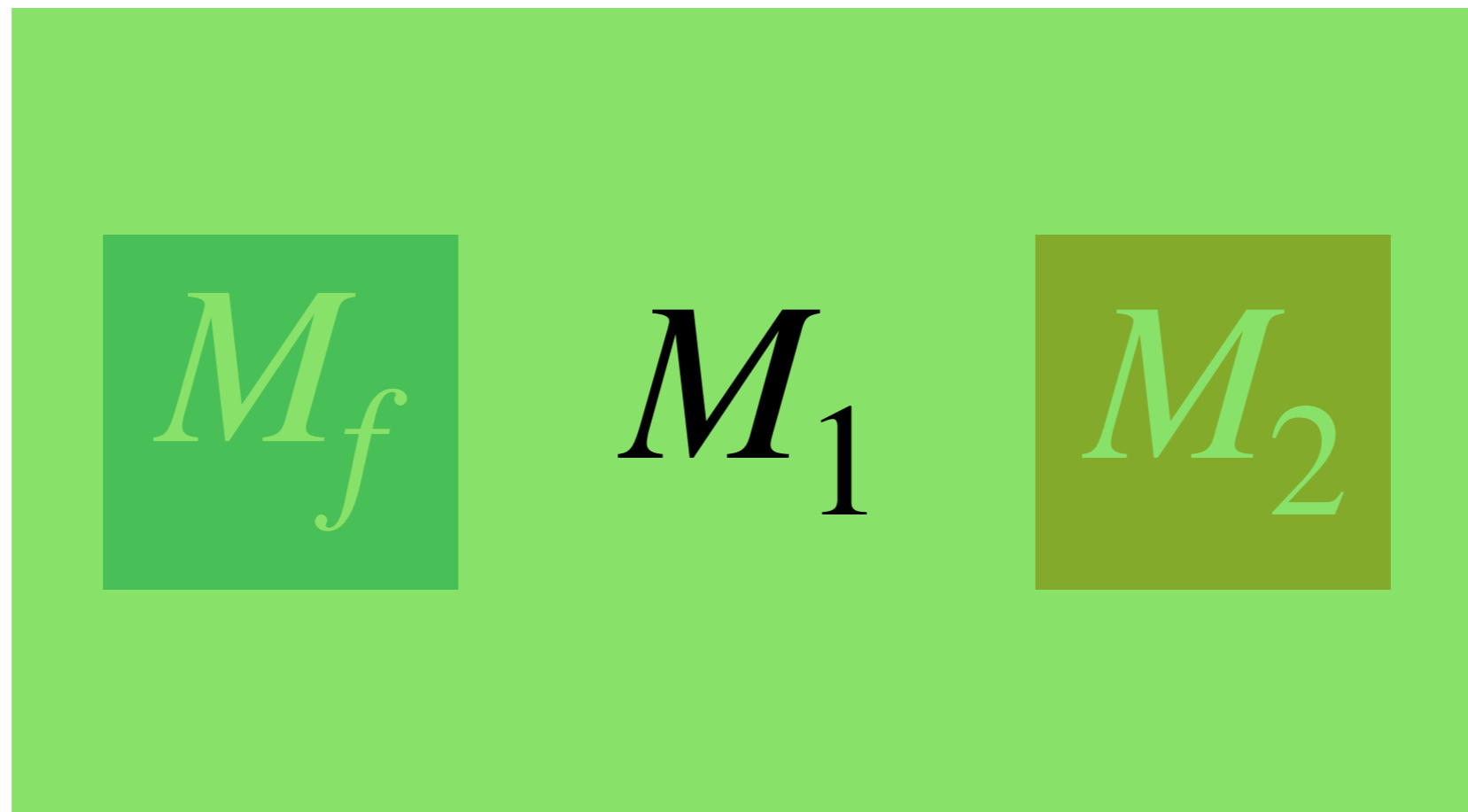


Résoudre L_1 avec une réduction



oui

Résoudre L_1 avec une réduction



oui
non

Proposition 3-C (p. 64)

P est close pour \leq

Si $L_2 \in \mathbf{P}$ et $L_1 \leq L_2$, alors $L_1 \in \mathbf{P}$

- Soit f la fonction de réduction de L_1 à L_2 en temps polynomial $p(n)$, et soit M_2 une machine déterministe qui reconnaît L_2 en temps polynomial $q(n)$
- Alors le suivant est un algorithme déterministe pour L_1 :
calculer $y = f(x)$ et retourner le résultat de $M_2(y)$
- Ça prend temps $p(n)$ pour calculer f et $q(p(n))$ pour simuler M_2 sur $y = f(x)$, donc temps polynomial, donc $L_1 \in \mathbf{P}$

Proposition 3-C (p. 64)

P est close pour \leq

Ça veut dire que si L_2 est efficacement résoluble et $L_1 \leq L_2$, c'est-à-dire que L_1 est plus simple que L_2 , alors L_1 est aussi efficacement résoluble, conformément à l'intuition

Proposition 3-C (p. 64)

NP est close pour \leq

Si $L_2 \in \mathbf{NP}$ et $L_1 \leq L_2$, alors $L_1 \in \mathbf{NP}$

- Soit f la fonction de réduction de L_1 à L_2 en temps polynomial $p(n)$, et soit M_2 une machine **non déterministe** qui reconnaît L_2 en temps polynomial $q(n)$
- Alors le suivant est un algorithme **non déterministe** pour L_1 :
calculer $y = f(x)$ et retourner le résultat de $M_2(y)$
- Ça prend temps $p(n)$ pour calculer f et $q(p(n))$ pour simuler M_2 sur $y = f(x)$, donc temps polynomial, donc $L_1 \in \mathbf{NP}$

Proposition 3-C (p. 64)

NP est close pour \leq

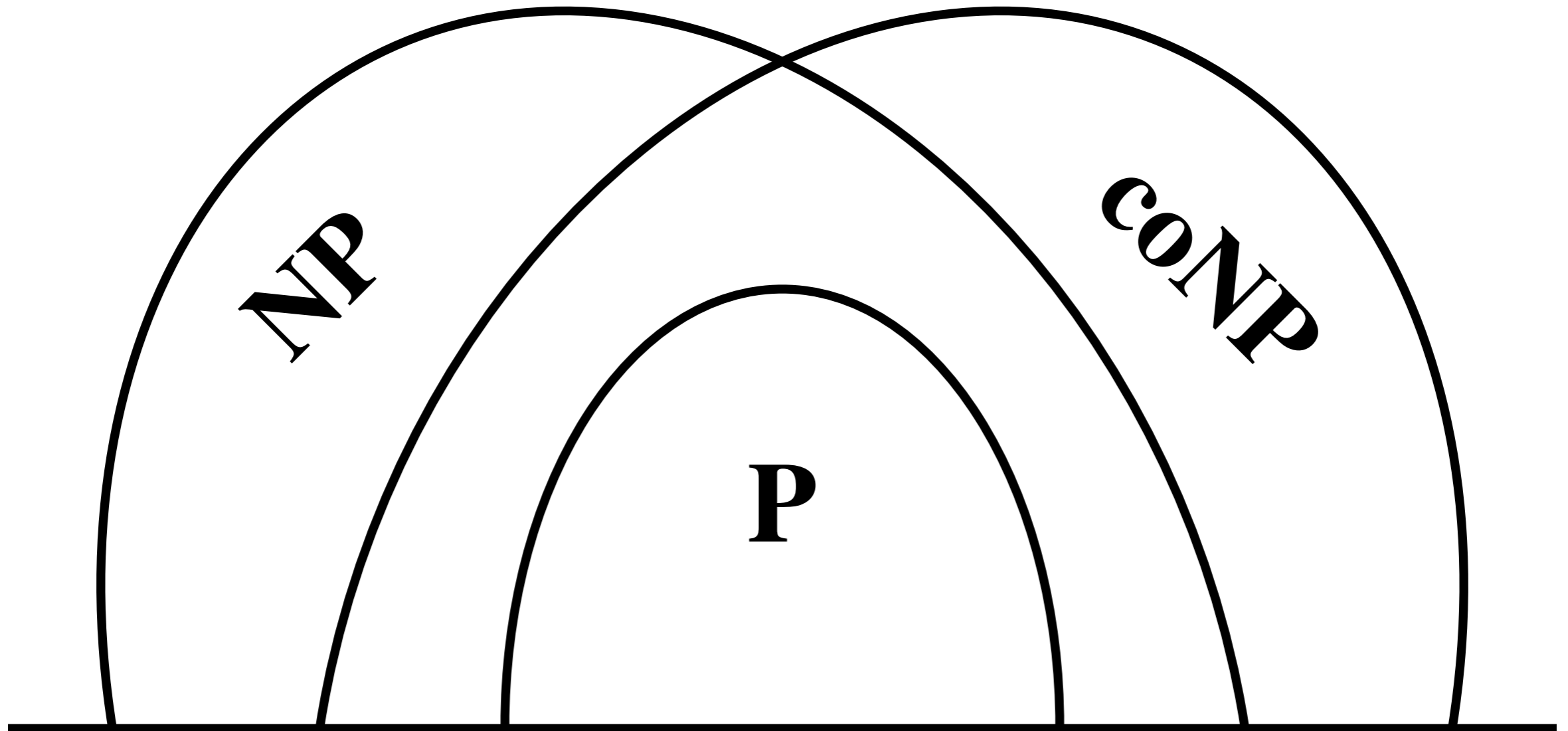
Ça veut dire que si L_2 est efficacement **verifiable** et $L_1 \leq L_2$, c'est-à-dire que L_1 est plus simple que L_2 , alors L_1 est aussi efficacement **verifiable**, conformément à l'intuition

ENS-INDÉP \equiv CLIQUE

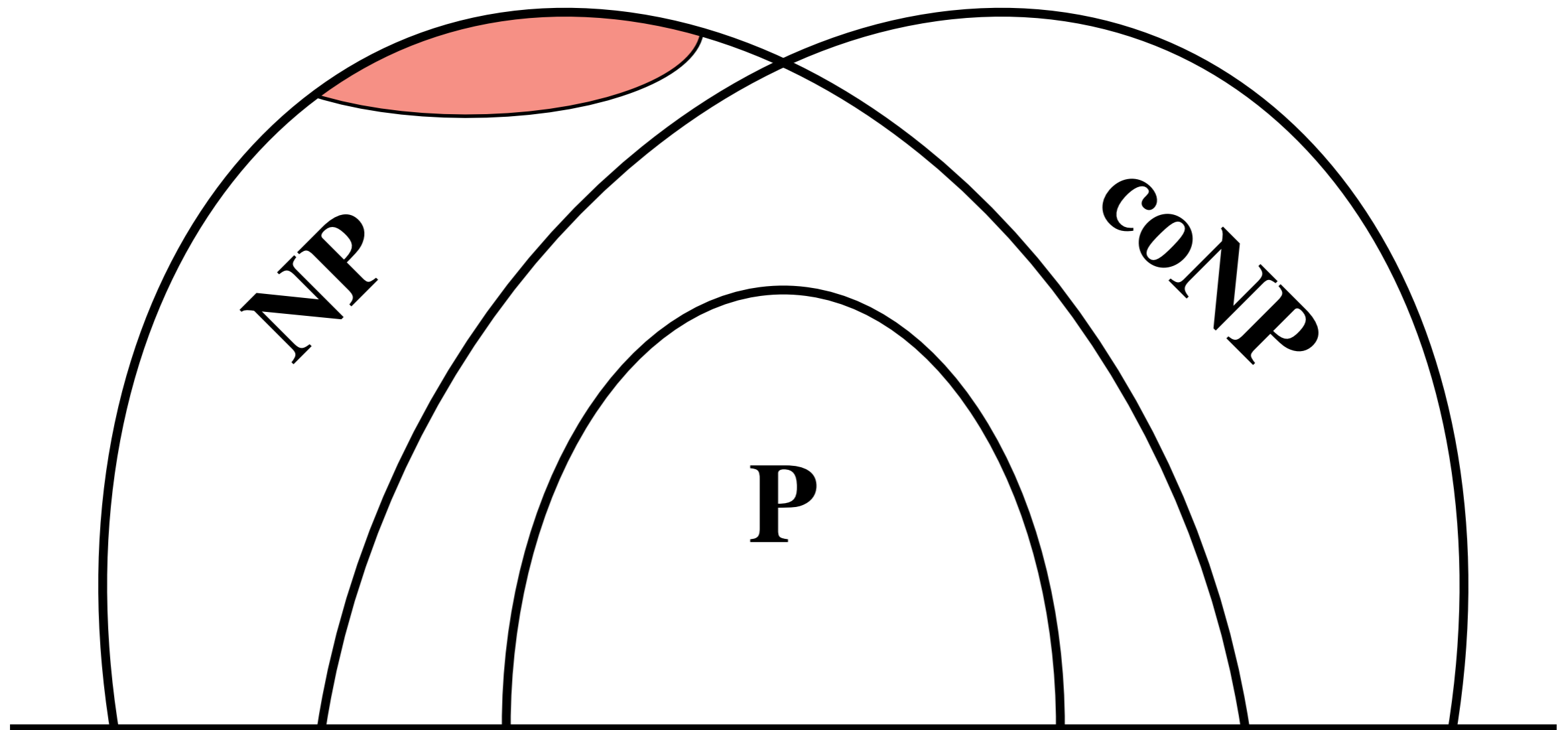
- On sait que CLIQUE \in NP (exercice du TD)
- Puisque ENS-INDÉP \leq CLIQUE, on a ENS-INDÉP \in NP, parce que NP est clos pour \leq
- Si on découvrait que CLIQUE \in P, on aurait aussi ENS-INDÉP \in P, parce que P est aussi clos par \leq
- Même chose si on découvrait que ENS-INDÉP \in P : ça impliquerait CLIQUE \in P

**Complétude,
ou les problèmes les plus
difficiles du monde**

Hiérarchie du « possible »

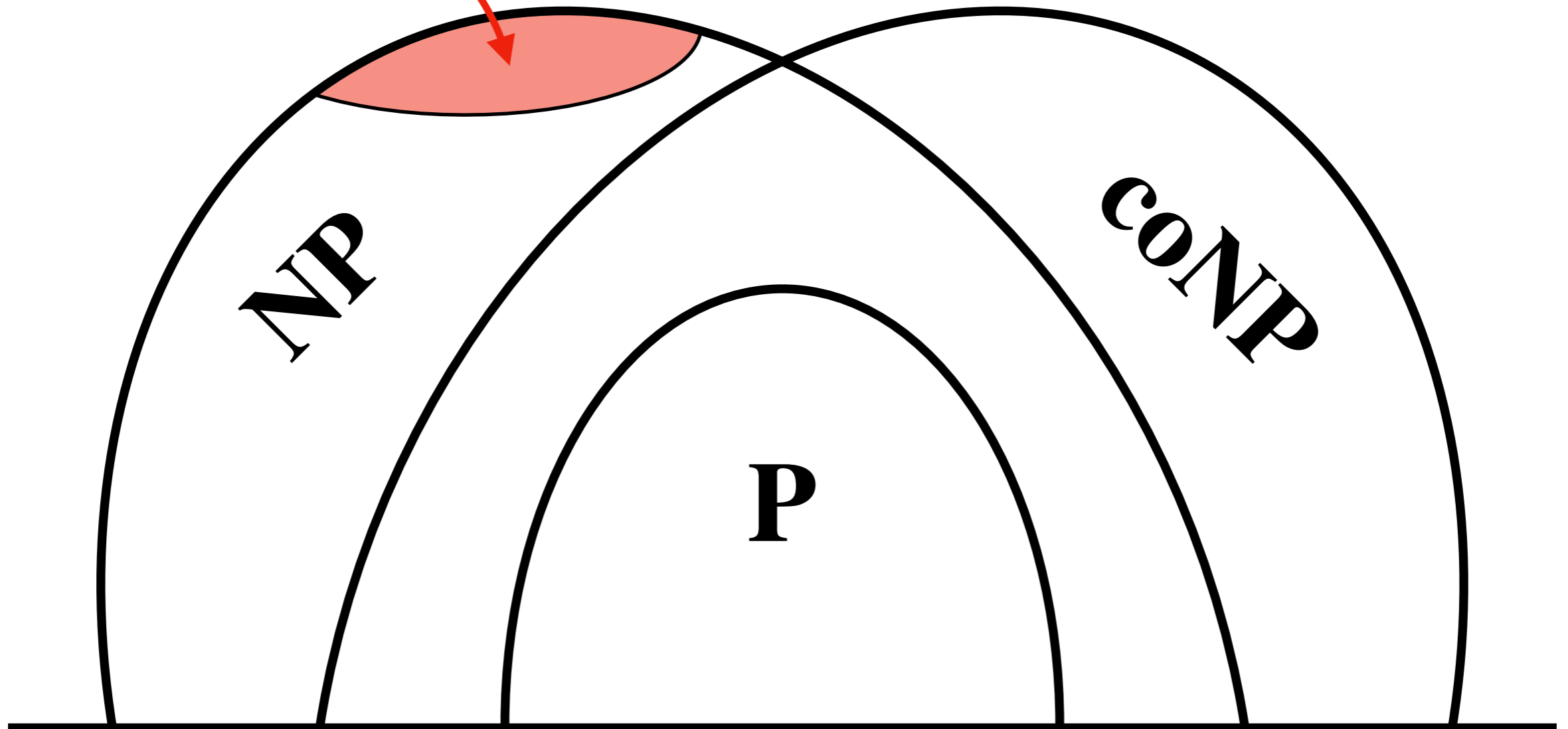


Hiérarchie du « possible »

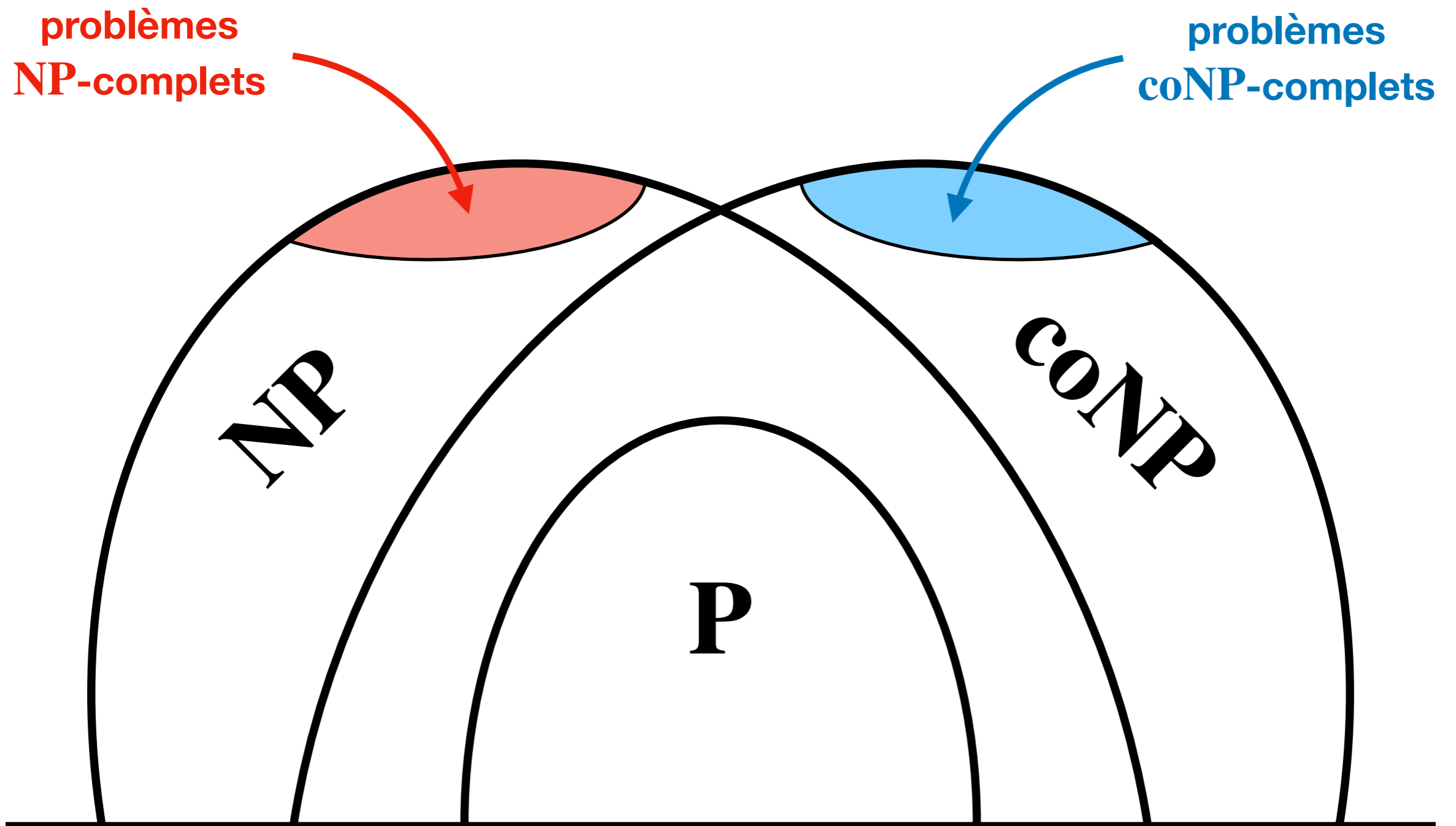


Hiérarchie du « possible »

problèmes
NP-complets



Hiérarchie du « possible »



Definition 3-J (p. 67)

Difficulté et complétude

Soit L un problème et \mathcal{C} une classe de complexité

- On dit que L est \mathcal{C} -difficile (ou \mathcal{C} -dur) si pour tout problème $L' \in \mathcal{C}$ on a $L' \leq L$
- On dit que L est \mathcal{C} -complet s'il est \mathcal{C} -difficile et en plus on a $L \in \mathcal{C}$

Difficulté et complétude

- Les définitions sont très fortes : il faut que **tous** les problèmes de \mathcal{C} se réduisent à L !
- À priori c'est n'est même pas évident qu'il existent des problèmes durs ou complets pour une classe...

P a (beaucoup de) problèmes complets

- Tout problème $L \in \mathbf{P}$ non trivial (c'est-à-dire, $L \neq \emptyset$ et $L \neq \Sigma^*$) est **P-complet** pour les reductions en temps polynomial
- Ça veut dire que cette notion de complétude n'est pas très intéressant pour **P**...

Demonstration

- Soit $L \in \mathbf{P}$ avec $L \neq \emptyset$ et $L \neq \Sigma^*$
- Alors il existe $a \in L$ et aussi $b \notin L$
- Soit $L' \in \mathbf{P}$ et soit $f(x) = \begin{cases} a & \text{si } x \in L' \\ b & \text{si } x \notin L' \end{cases}$
- Comme $L' \in \mathbf{P}$, la fonction f est calculable en temps polynomial déterministe
- Mais aussi $f(x) = a \in L$ ssi $x \in L'$, donc $L' \leq L$

**Il existe bien des problèmes
NP-complets,
ou : on est pas là pour
perdre notre temps**

Proposition 3-M (p. 68)

La prédiction est NP-complète

Le problème suivant est NP-complet :

$$A = \{(\langle N \rangle, x, 1^t) : N(x) \text{ accepte en temps } \leq t\}$$

code d'une MT
non déterministe

mot d'entrée
pour la MT N

entier $t \in \mathbb{N}$
en unaire



Idée de la démonstration

- $A \in \mathbf{NP}$ parce qu'on peut **simuler $N(x)$** avec une machine de Turing universelle non déterministe
- Pour tout $B \in \mathbf{NP}$, on a $B \leq A$ parce qu'**on peut prévoir le résultat** de la machine N_B qui reconnaît B
 - Le problème A est, justement, la prédiction du résultat d'une TM non déterministe

Démonstration : $A \in \text{NP}$

- Soit U une machine universelle non déterministe efficace
- Voilà un algorithme pour décider si $(\langle N \rangle, x, 1^t) \in A$:
 - **Simuler** $N(x)$ pendant **t étapes** en exécutant $U(\langle N \rangle, x)$
 - Si $U(\langle N \rangle, x)$ **accepte** en $\leq t$ étapes, **accepter**
 - Si elle **rejette** ou elle **n'a pas terminé** en t étapes, **rejeter**
- Comme U simule N en temps polynomial, cet algorithme fonctionne aussi en **temps polynomial**

Démonstration :

$$\forall B \in \mathbf{NP}, B \leq A$$

- Soit $B \in \mathbf{NP}$ et soit N_B une machine non déterministe qui reconnaît B en temps polynomial $p(n)$
- Donc on connaît N_B et son temps de calcul $p(n)$, puisque c'est nécessaire pour prouver que $B \in \mathbf{NP}$
- Voilà la réduction : $f(x) = (\langle N_B \rangle, x, 1^{p(|x|)})$
- $x \in B$ ssi $N_B(x)$ accepte en temps $\leq p(|x|)$ ssi $f(x) \in A$
- f est calculable en temps polynomial

f est calculable en temps polynomial, par exemple
 si N_B fonctionne en temps $p(n) = n^3 - 3n + 7$

fonction $f(x)$

$n := |x|$

écrire "(" sur le ruban de sortie

écrire le code $\langle N_B \rangle$ sur le ruban de sortie

écrire "," sur le ruban de sortie

écrire x sur le ruban de sortie

écrire "," sur le ruban de sortie

pour $i := 1$ à n faire

 pour $j := 1$ à n faire

 pour $k := 1$ à n faire

 écrire 1 sur le ruban de travail

pour $i := 1$ à 3 faire

 pour $j := 1$ à n faire

 effacer 1 du ruban de travail

pour $i := 1$ à 7 faire

 écrire 1 sur le ruban de travail

copier le ruban de travail sur le ruban de sortie

écrire ")" sur le ruban de sortie

fin

} $O(1)$

$O(n)$

$O(1)$

} $O(n^3)$

} $O(n)$

} $O(1)$

$O(p(n))$

$O(1)$

} $O(p(n))$



! Remarque !

- Ouais, on est bien d'accord, ce n'est **pas** un problème **très intéressant**... 😬
- C'est un problème ad-hoc calqué sur la définition de **NP**
- La **NP**-complétude devient pertinente lorsqu'elle concerne des problèmes **naturels**
- On verra qu'il y en a **plein** de naturels !

Les problèmes NP-complets,
ou entrons enfin
dans le vif du sujet

Proposition 3-P (p. 69)

NP-complétude : tout ou rien

Les affirmations suivantes sont équivalentes :

1. **$P = NP$**
2. tout problème **NP-complet** est dans **P**
3. il existe un problème **NP-complet** dans **P**

Démonstration

- Si $\mathbf{P} = \mathbf{NP}$ (1), alors en particulier $\mathbf{NP} \subseteq \mathbf{P}$,
donc tout problème \mathbf{NP} -complet est dans \mathbf{P} (2)
- Si tout problème \mathbf{NP} -complet est dans \mathbf{P} (2),
vu qu'il existe au moins un problème \mathbf{NP} -complet,
alors il existe un problème \mathbf{NP} -complet dans \mathbf{P} (3)
- S'il existe un problème \mathbf{NP} -complet A dans \mathbf{P} (3),
alors on peut le résoudre en temps polynomial.
Mais comme $B \leq A$ pour tout $B \in \mathbf{NP}$,
alors $B \in \mathbf{P}$ aussi, donc $\mathbf{NP} \subseteq \mathbf{P}$, donc $\mathbf{P} = \mathbf{NP}$ (1)

⚠ Proposition 3-W (p. 76) ⚠

NP-complétude par réduction

- Soit C un problème NP-complet et $A \in \text{NP}$
- Si $C \leq A$ alors A est aussi NP-complet

Démonstration

- Soit $B \in \mathbf{NP}$ n'importe quel problème dans \mathbf{NP}
- Comme C est \mathbf{NP} -complet, on a $B \leq C$
- Si $C \leq A$, alors $B \leq A$ aussi par transitivité de \leq
- Ça vaut pour tout $B \in \mathbf{NP}$, donc A est \mathbf{NP} -difficile
- Et comme $A \in \mathbf{NP}$ aussi, il est \mathbf{NP} -complet



! Remarque !

- On vient d'utiliser la notion de réduction $C \leq A$, qui montre qu'un **peut résoudre C efficacement si on a une solution efficace pour A ...**
- ...d'une façon **perverse**, pour montrer que **A n'a pas de solution efficace*** !
- Autrement dit, si C est difficile est A est au moins aussi difficile que C , alors A est difficile aussi

* Dans l'hypothèse que $P \neq NP$