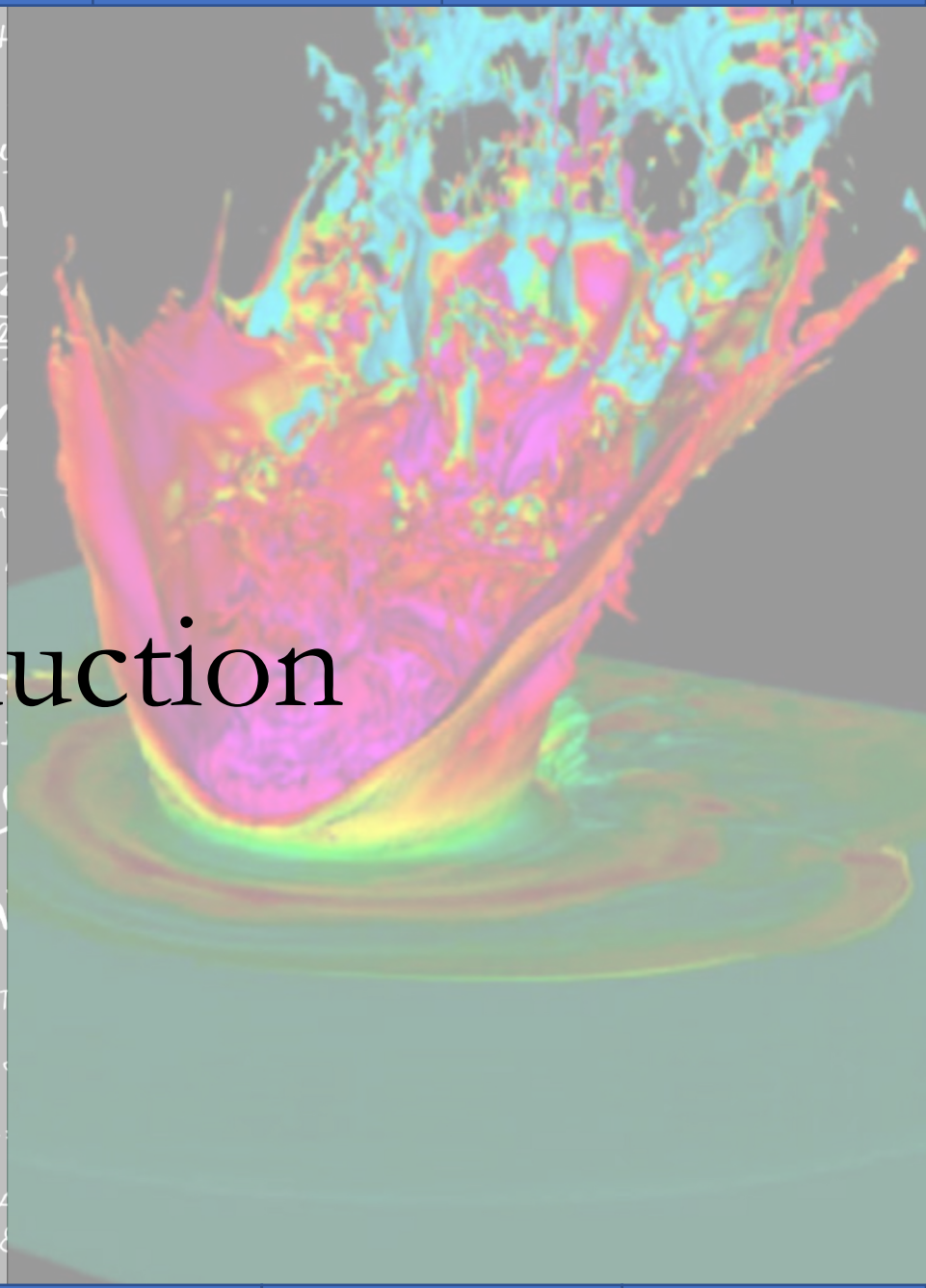
Méthodes Numériques

Cours 8 :

Optimisation des méthodes numériques

Dr. Barbara PERRI

barbara.perri@universite-paris-saclay.fr



Introduction

Plan de l'UE

Idée générale :

Au premier semestre, on va introduire les notions de base, et s'intéresser en détails à une méthode numérique précise

Tous les jeudi matin (8h45-12h45) au bâtiment 625

21 Novembre : Cours 1 + Cours 2
4 Décembre : Cours 3 + TP
5 Décembre : Cours 4 + TP
12 Décembre : Cours 5 + TP
19 Décembre : Cours 6 + TP
9 Janvier : Cours 7 + TP
16 Janvier : **Cours 8** + TP
23 Janvier : TP
30 Janvier : Examen

Modalités d'évaluation :

TPs + examen oral (question de cours + exercice)

Résumé des méthodes numériques

Les méthodes numériques permettent de passer d'un problème mathématique à un problème numérique, de telle sorte que la solution numérique soit physique

Il existe de nombreuses méthodes
(différences/volumes/éléments finis, méthodes spectrales, etc.)

Dans ce cours, nous avons vu :

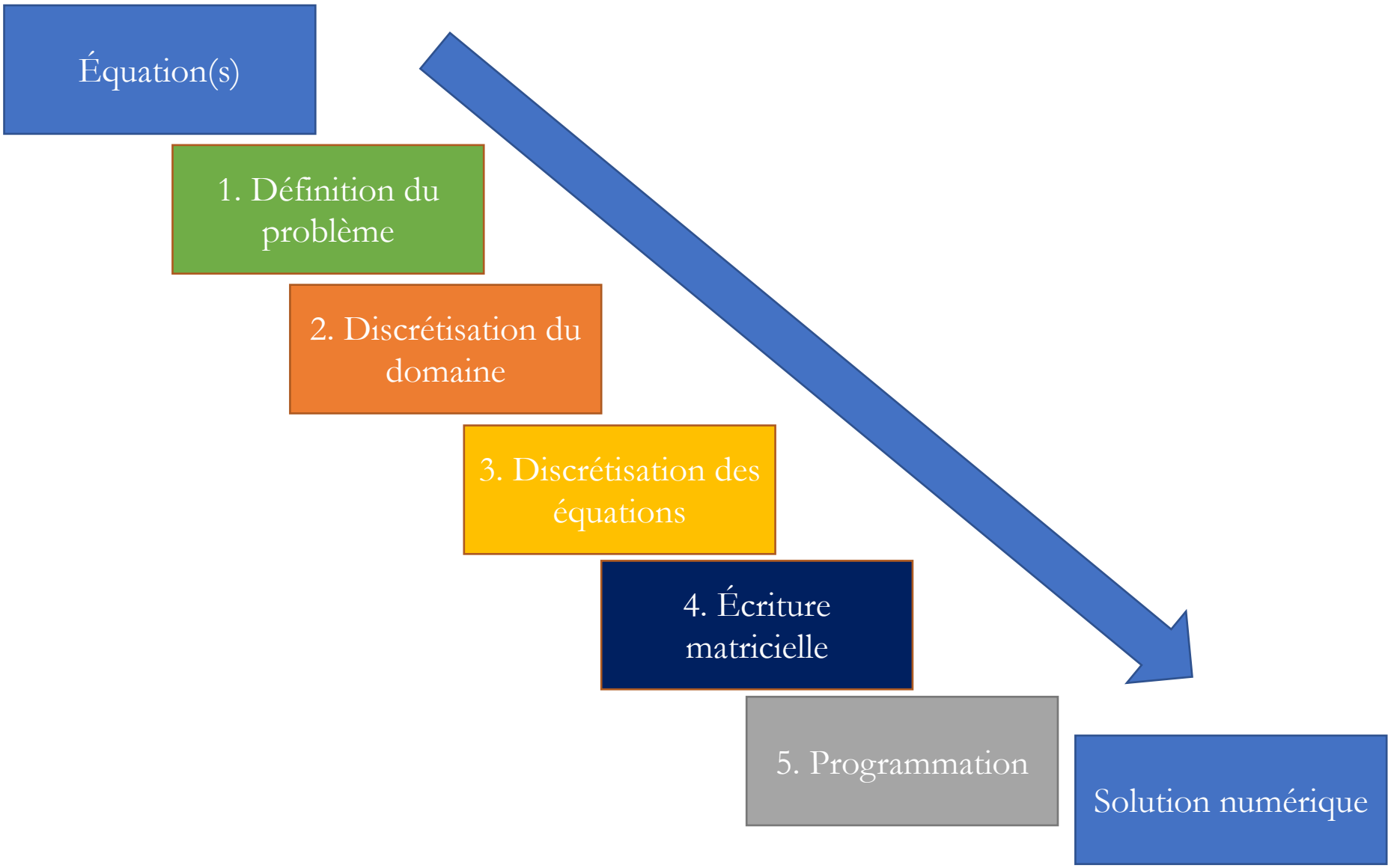
Équations différentielles ordinaires (EDO)

Méthode de Newton
Interpolation par Polynôme de Lagrange
Méthodes de Runge-Kutta

Équations aux dérivées partielles (EDP)

Méthode des différences finies
→ Par la tangente
→ Par les développements limités
→ Par le polynôme de Lagrange

Cycle des méthodes numériques



Cycle des méthodes numériques (I)

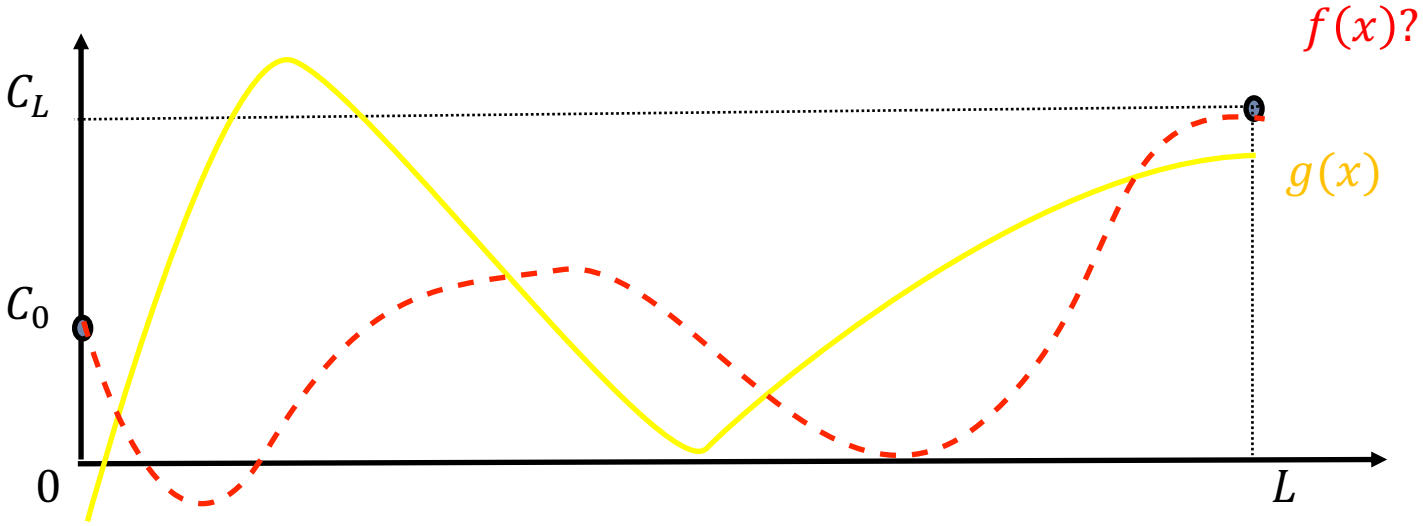
La première étape consiste à **définir le problème :**

→ Équation,

→ Inconnue,

→ Domaine,

→ Conditions aux limites.



Cycle des méthodes numériques (II)

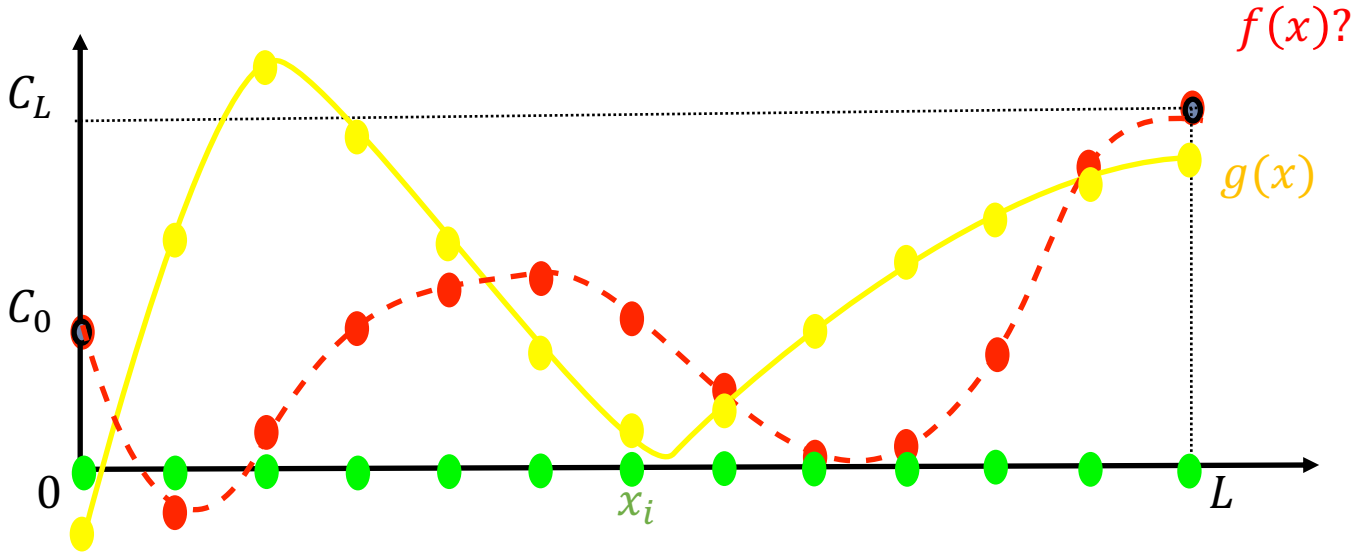
La deuxième étape consiste à **discrétiser le domaine :**

→ Nombre de points,

→ Pas d'espace (type de maillage),

→ Numérotation,

→ Équivalence discret/continu.



Cycle des méthodes numériques (III)

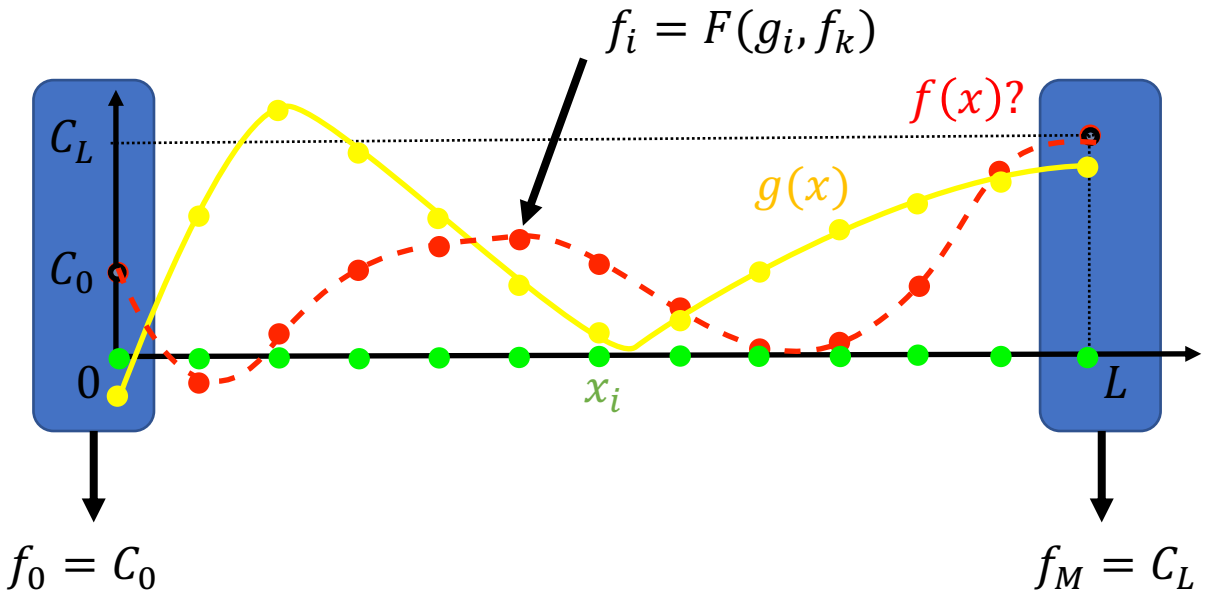
La troisième étape consiste à **discrétiser les équations :**

Bords

→ Condition aux limites,
→ Dirichlet : immédiat,
→ Neuman : discrétiser la dérivée.

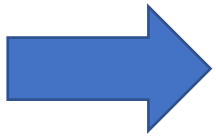
Nœuds du maillage

→ Choisir une discrétisation des dérivées du problème,
→ Mettre le schéma sous forme générale.

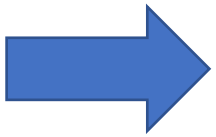


Cycle des méthodes numériques (IV+V+VI)

La quatrième étape consiste à **écrire le problème sous forme matricielle :**

Système linéaire  $AF = G$

La cinquième étape consiste à **programmer et résoudre le problème :**

$AF = G$  $F = A^{-1}G$

La sixième étape consiste à **valider la solution :**

→ Trouver une solution physique au problème,
→ La calculer et la tracer en plus de la solution numérique,
→ Calculer l'erreur entre les deux pour quantifier la solution numérique.

Caractérisation d'un schéma numérique

On a vu les 4 propriétés essentielles qui permettent de caractériser un schéma :

Consistance :

$(schéma)_i^n \xrightarrow[\delta x,\delta t\to 0]{} (équation\ exacte)_i^n$

$R^n \xrightarrow[\delta x,\delta t\to 0]{} 0, \forall n$

→ Développements limités (à un ordre suffisant)

Ordre :

$R_i^n = termes + O(\delta x^{p+1}) + O(\delta t^{q+1})$

p : ordre spatial, q : ordre temporel

$min(p,q)$: ordre global

→ Développements limités (à un ordre suffisant)

Stabilité :

$\exists C \in R, ||\varepsilon^n X|| \leq C||X||$

$\forall X, 0 \leq n \leq T/\delta t$

→ Calcul du facteur d'amplification dans l'espace de Fourier

Majoration pour stabilité L2 (critère de von Neumann)

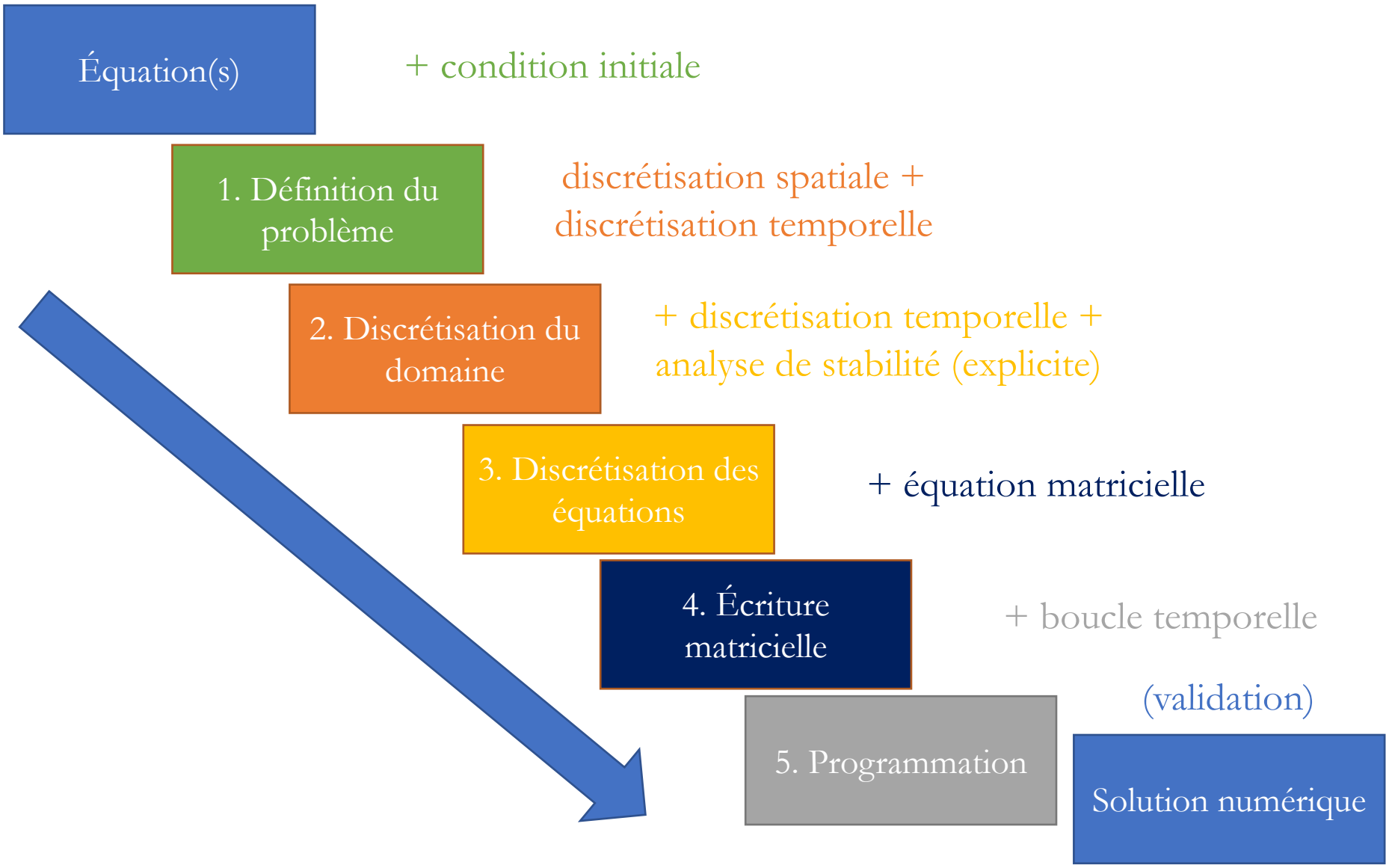
Convergence :

$\forall n, ||F_{ex}^n - F^n|| \xrightarrow[\delta x,\delta t\to 0]{} 0$

$||E^n|| \xrightarrow[\delta x,\delta t\to 0]{} 0, \forall n$

→ Consistance + Stabilité (Théorème de Lax)

Ajout de la dimension temporelle (I)



Ajout de la dimension temporelle (II)

Une fois le schéma mis sous forme générale,
on voit tout de suite s'il est explicite ou implicite

Explicite

= on exprime l'état futur du système en fonction des états passés

$f^{n+1} = F[f^n, f^{n-1}, \dots]$

$$F^0 = G^0$$

$$F^{n+1} = \varepsilon F^n + G$$

→ La résolution est itérative

→ Pas toujours stable

Implicite

= on exprime l'état futur du système en fonction de l'état futur également

$G[\dots, f^{n-1}, f^n, f^{n+1}] = 0$

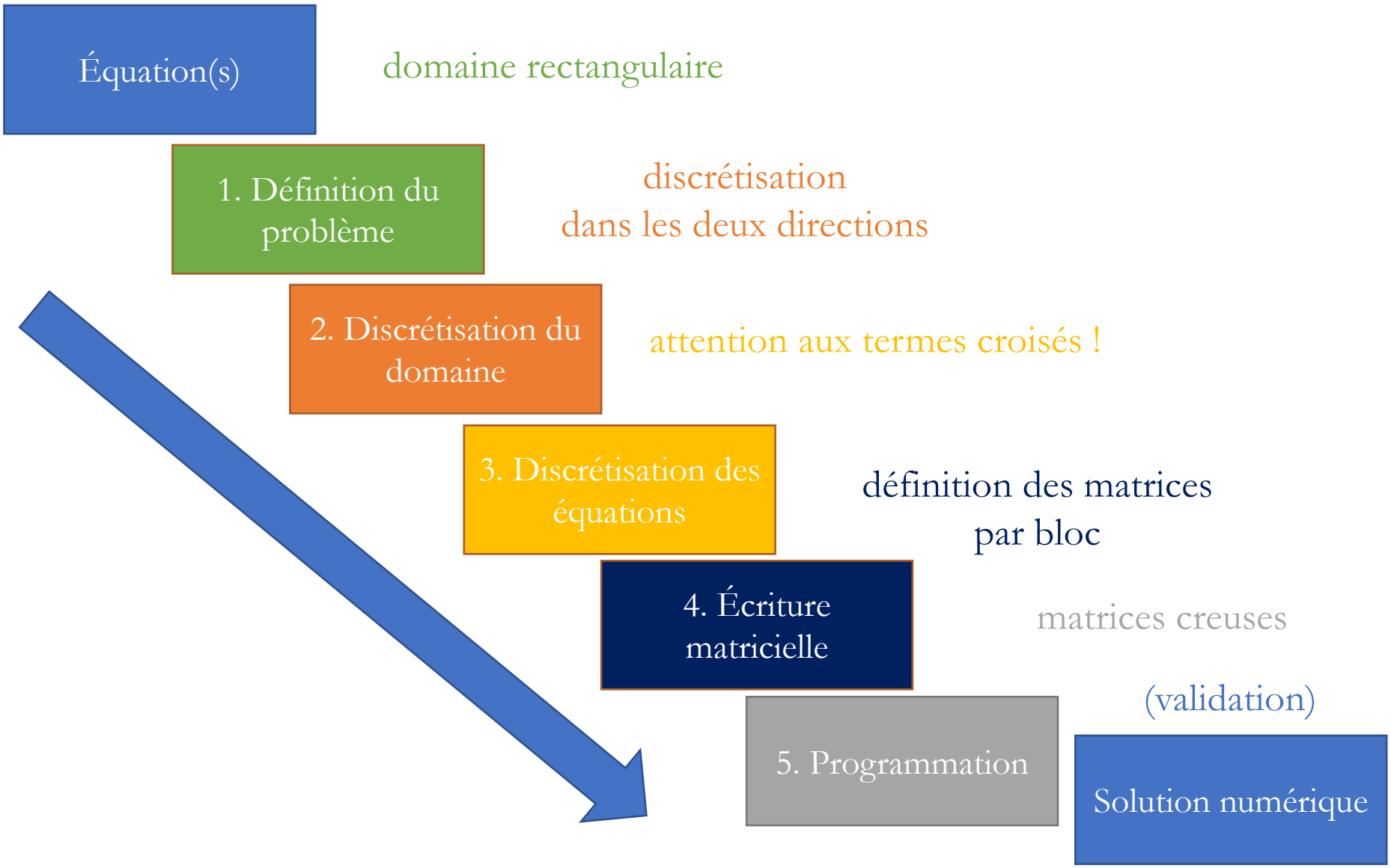
$$F^0 = G^0$$

$$O_g F^{n+1} = O_d F^n + G$$

→ La résolution est matricielle

→ Toujours stable

Passage en 2D



Optimisation

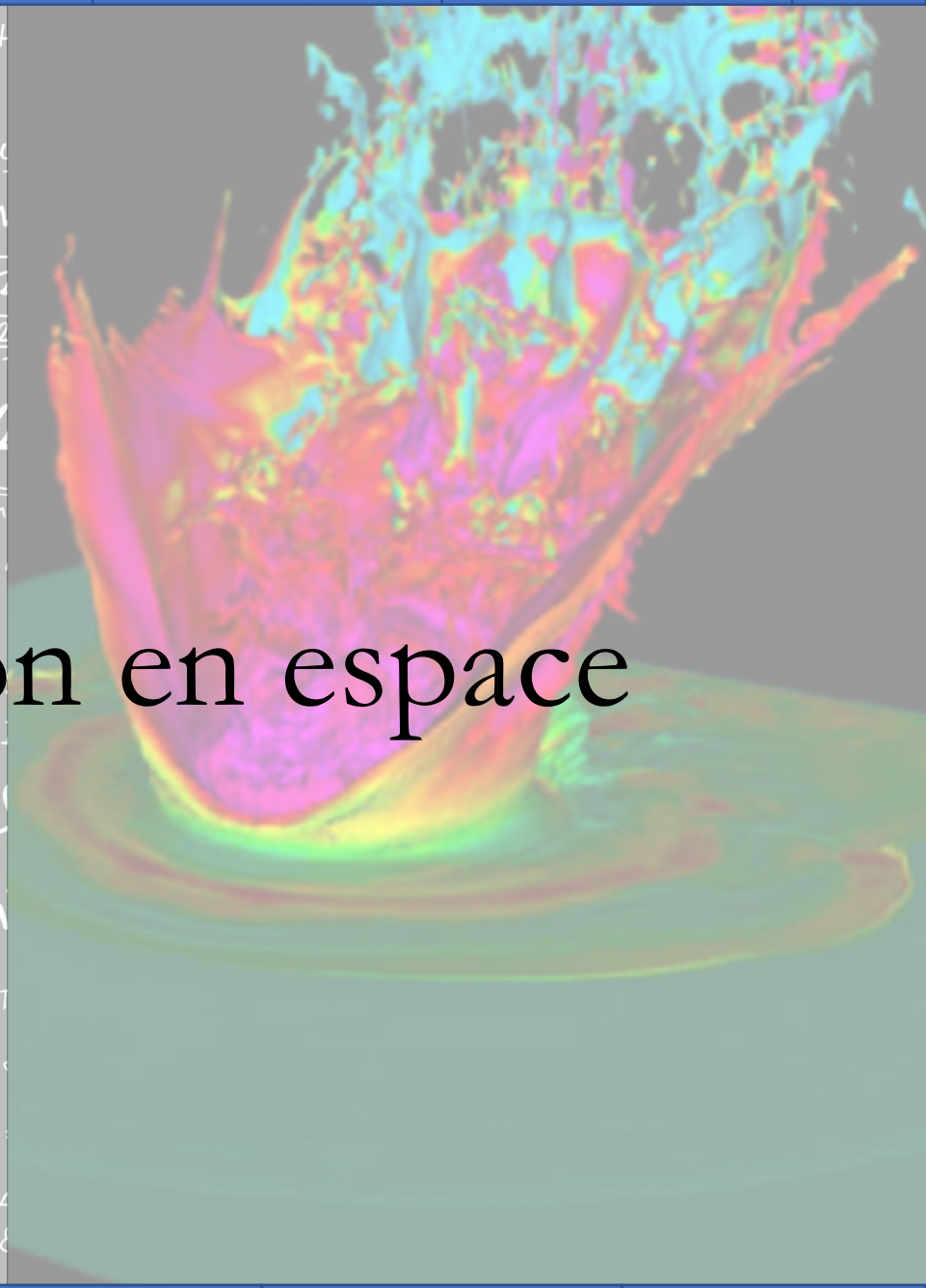
Dans ce cours, on a construit de zéro des briques élémentaires
de programmation de schémas numériques en différences finies
→ Mais lors de vos stages, vous travaillerez probablement sur des codes déjà existants !

Ça veut dire :

- Des codes de plusieurs milliers de ligne (rédigées sur plusieurs années),
- Un environnement qui n'est pas forcément votre ordinateur (supercalculateur),
- Des méthodes numériques plus avancées que vu en cours.

L'idée de ce cours d'ouverture, c'est donc de vous donner du **vocabulaire**
et de la **culture générale** pour mieux comprendre ces codes complexes
(et ne pas s'en servir juste comme d'une boîte noire)
→ En particulier, on va s'intéresser ici aux surcouches **d'optimisation numérique**
(recherche actuelle !)

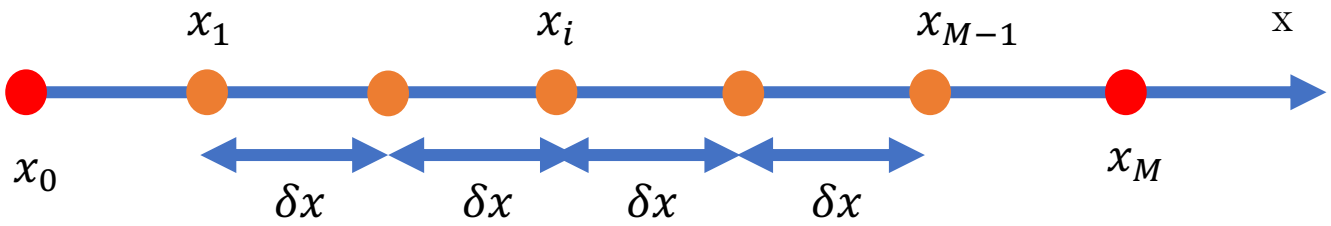
Optimisation en espace



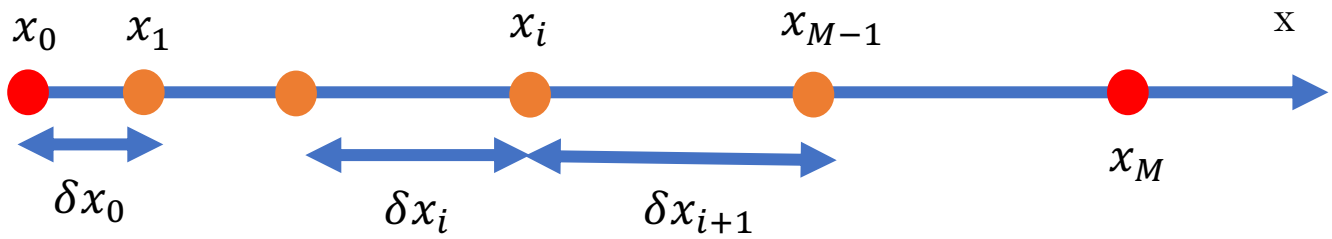
Rappel sur les maillages

On a vu que pour discrétiser le domaine spatial, on peut utiliser des maillages plus ou moins complexes en fonction du problème :

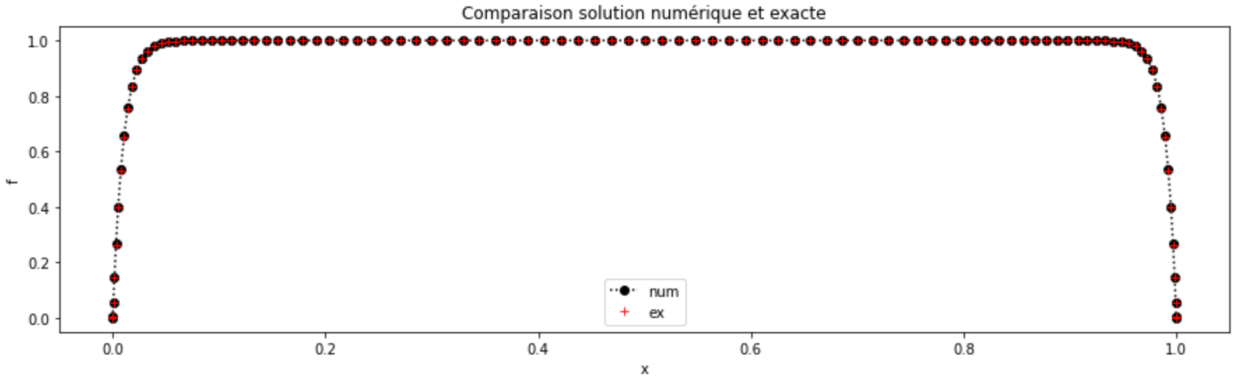
Maillage uniforme :



Maillage irrégulier :



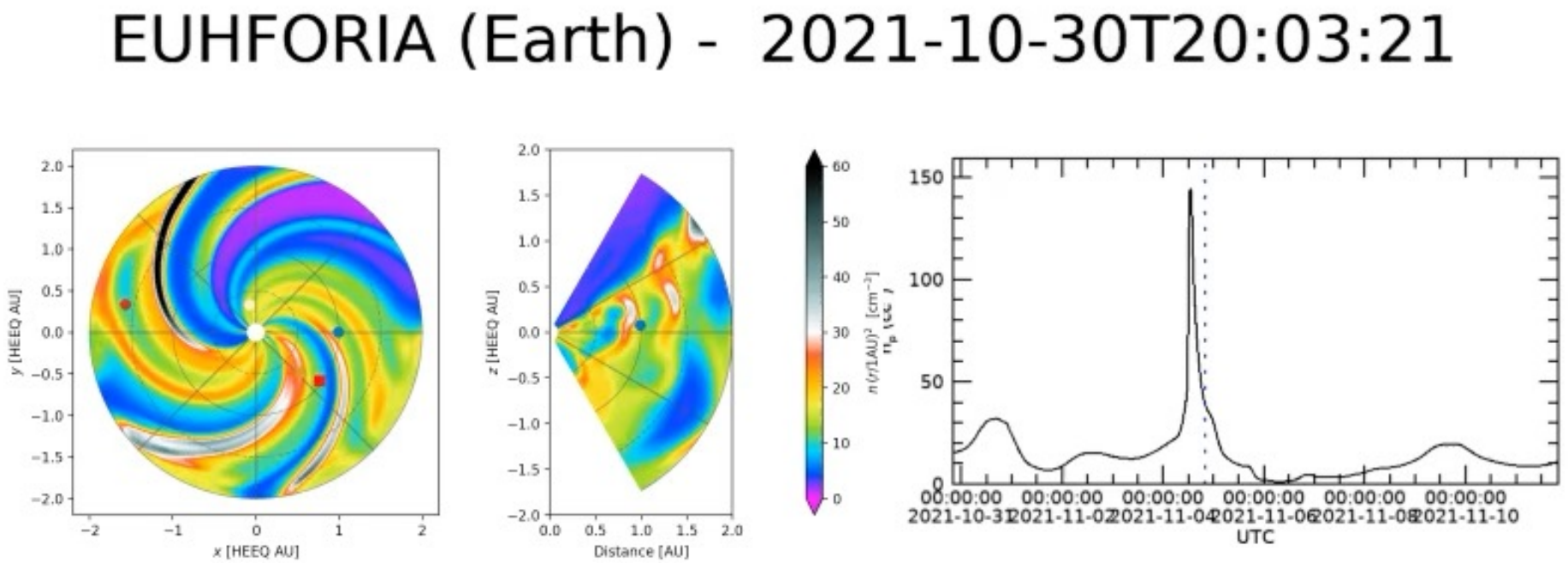
MAIS cela suppose qu'on sait déjà où sera l'information physique du problème !



Problème de maillage

Sauf que la plupart du temps, on ne peut pas savoir où sera l'élément physique intéressant
+ il peut se déplacer au cours du temps !

Exemple : simulation de plusieurs éjections de masse coronale du Soleil à la Terre

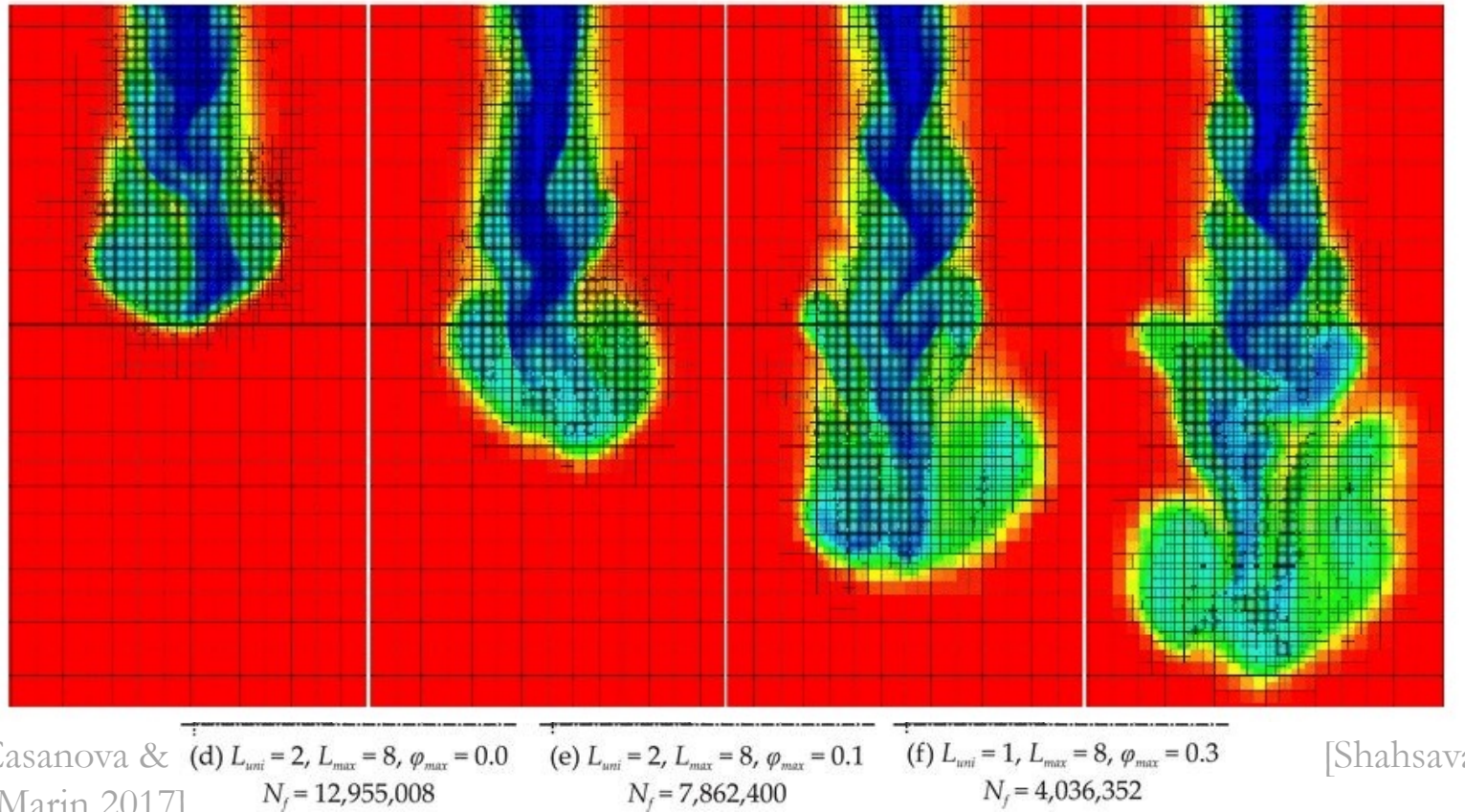


→ On voudrait donc un moyen de raffiner automatiquement
le maillage aux endroits intéressants !

Notion de maillage adaptatif

C'est justement le cœur de la technique d'AMR (*Adaptative Mesh Refinement*)

→ On définit un critère et on raffine le maillage en fonction de ce dernier



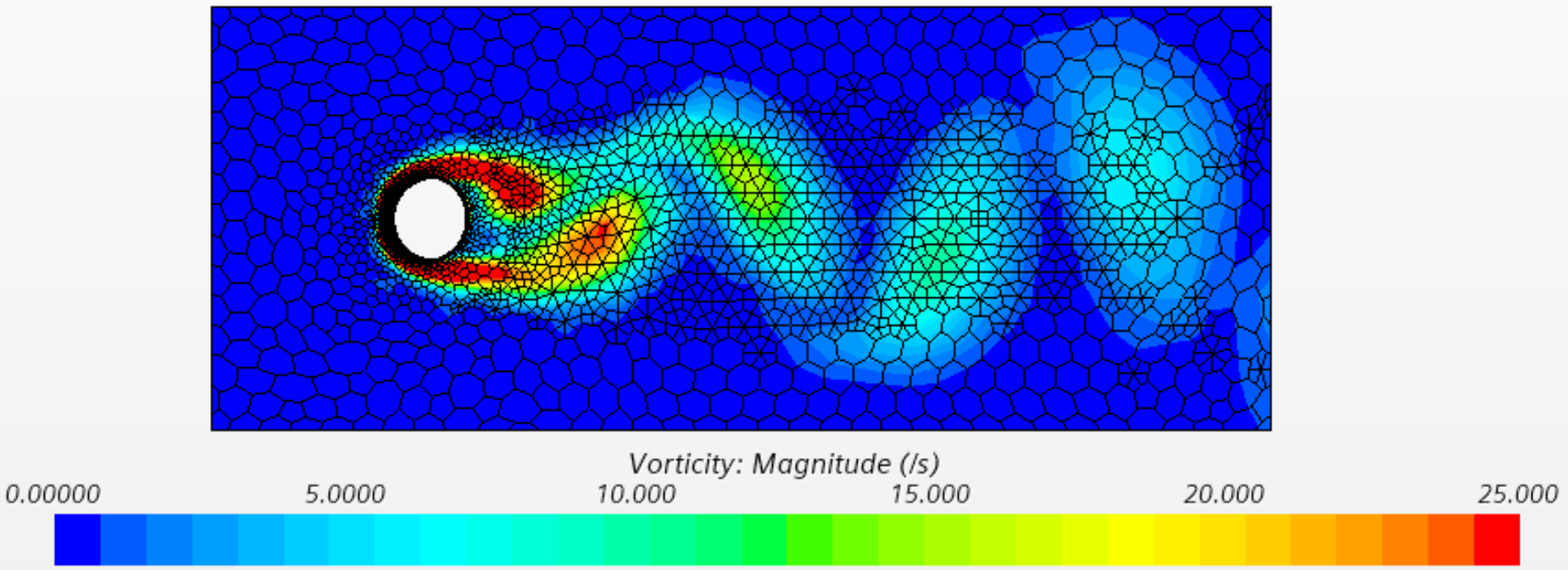
[Roda-Casanova & Sanchez-Marin 2017] [Shahsavan+2018]

→ Voyons le vocabulaire associé à cette technique

Critère de raffinement

Avant de pouvoir raffiner, il faut d'abord définir où c'est intéressant de pouvoir le faire

→ c'est le critère de raffinement, pour sélectionner les régions d'intérêt



En général, on utilise :

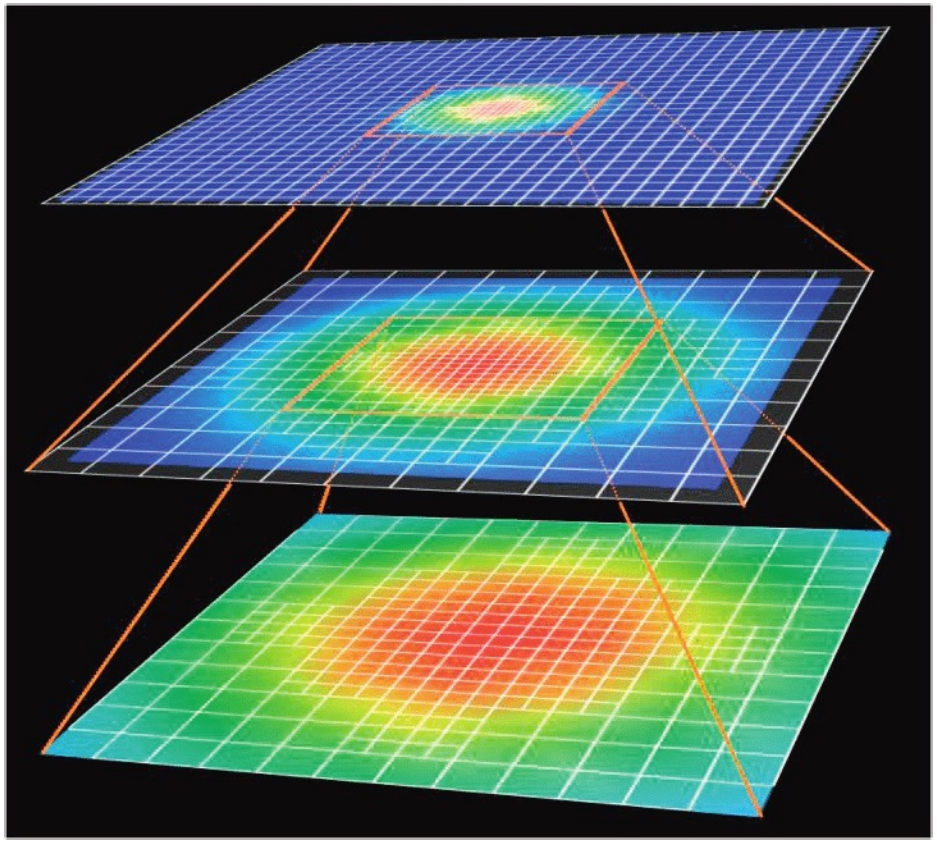
→ Les fortes valeurs d'une quantité physique (seuil),

→ Les fortes variations d'une valeur physique (gradient).

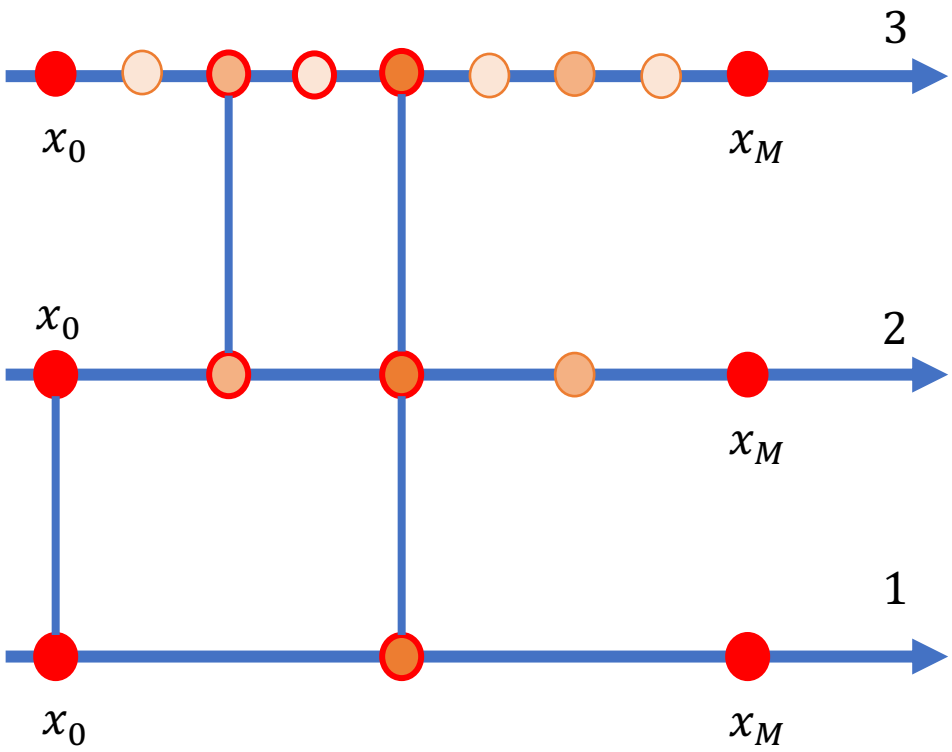
Niveau de grille

Cette méthode mène à la manipulation de grilles composites

→ On peut comprendre leur construction avec la notion de niveaux de grille :



[Matsumoto+2014]



→ Un inconvénient majeur est la manipulation (conditions aux limites ?) et visualisation des résultats calculés sur ces grilles

Exemple de simulations avec AMR (I)

On reprend l'exemple des éjections de masse coronale avec un maillage AMR :

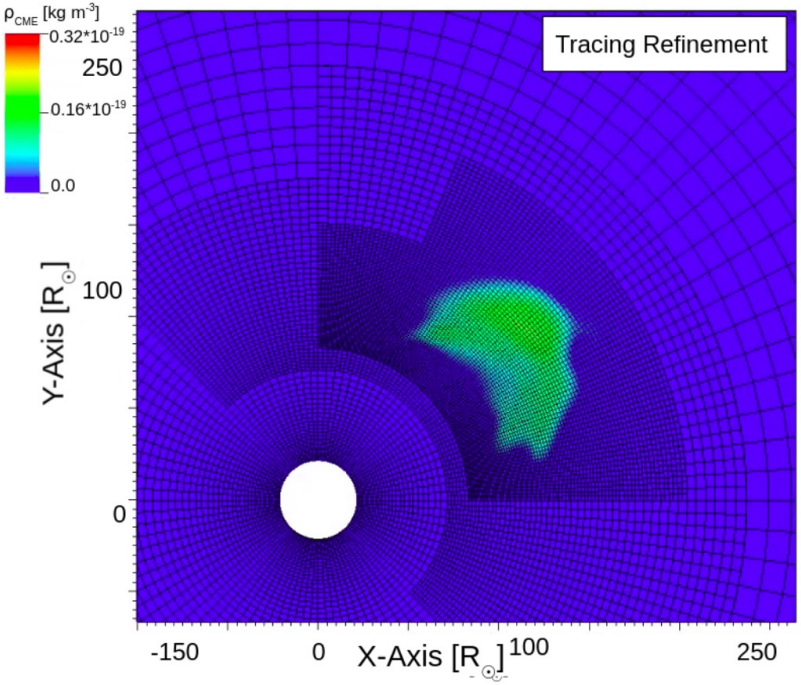
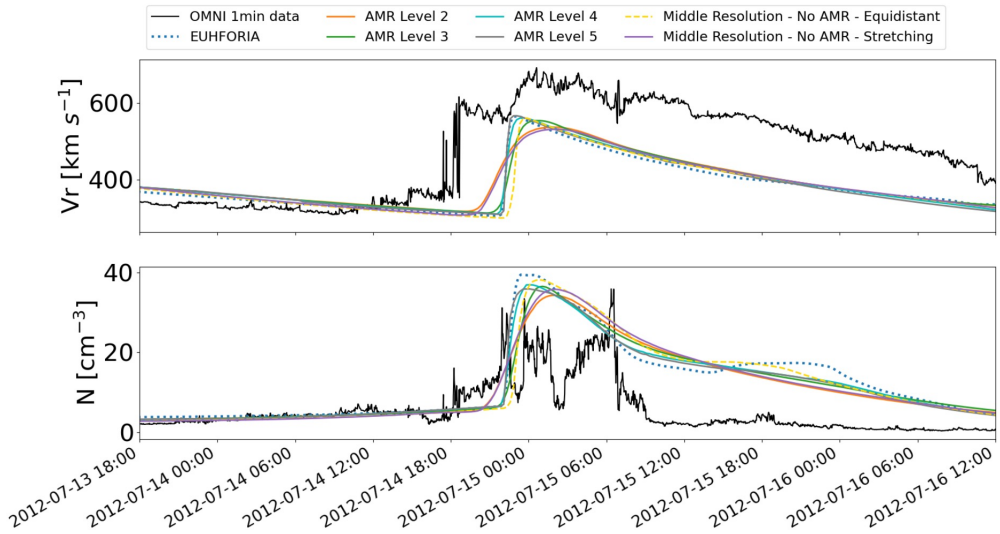


Fig. 2. Adaptive mesh refinement level 3 for the CME density tracing function. The *X*- and *Y*-axis show the distance from the centre in solar radii. The CME density tracing function is shown in the equatorial plane with the corresponding colour map.



[Baratashvili+2022]

→ Peu de perte en précision

Exemple de simulations avec AMR (II)

On reprend l'exemple des éjections de masse coronale avec un maillage AMR :

Table 4. Run times required for the EUHFORIA simulation, middle-resolution simulations on the equidistant ($Middle_{EQ}$) and stretched ($Middle_{STR}$) grids, and AMR level 2 and 3 runs (starting from Low_{STR}).

EUHFORIA	$Middle_{EQ}$	$Middle_{STR}$	AMR2	AMR3
14 h 13 m	6 h 16 m	1 h 30 m	0 h 9 m	0 h 17 m

Notes. All the simulations were performed on 1 node with 36 processors on the Genius cluster at the Vlaams Supercomputing Centre.

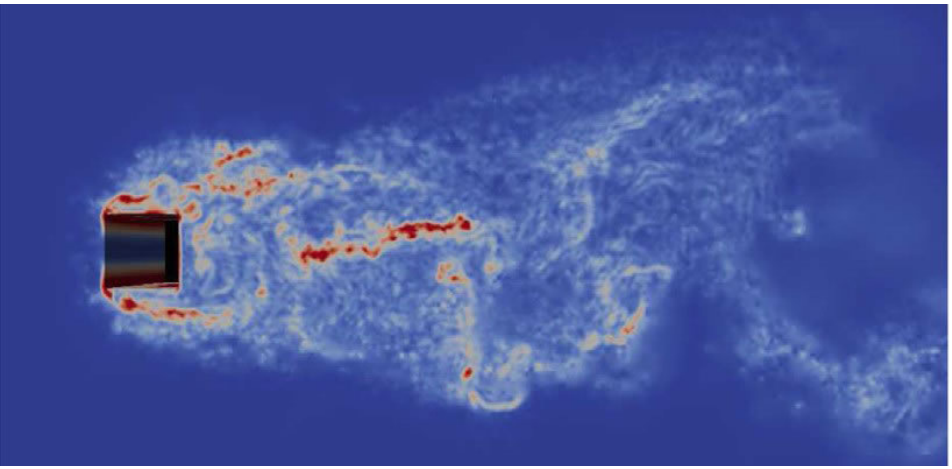
→ Gain de temps plus que significatif !

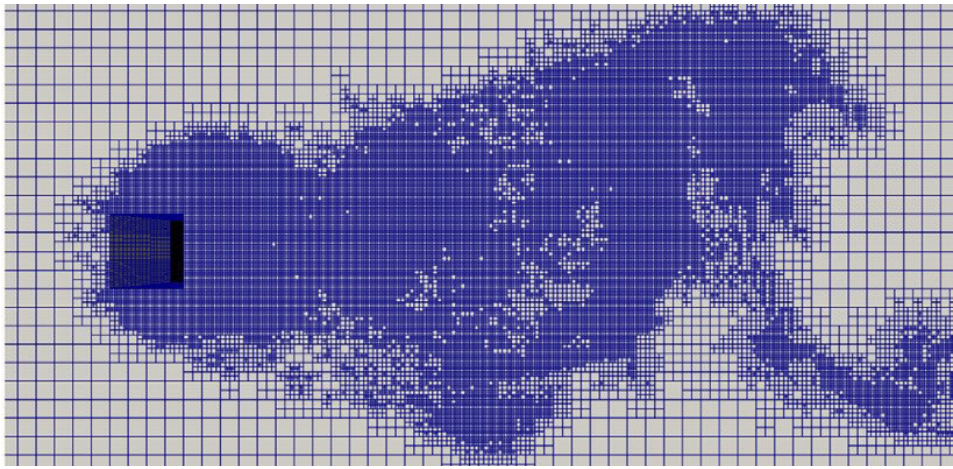
Résumé de l'AMR

L'*Adaptative Mesh Refinement* ou AMR permet d'obtenir un maillage plus précis uniquement dans les régions d'intérêt du domaine

On a besoin :

- d'un critère de raffinement,
- d'un niveau maximal de grille





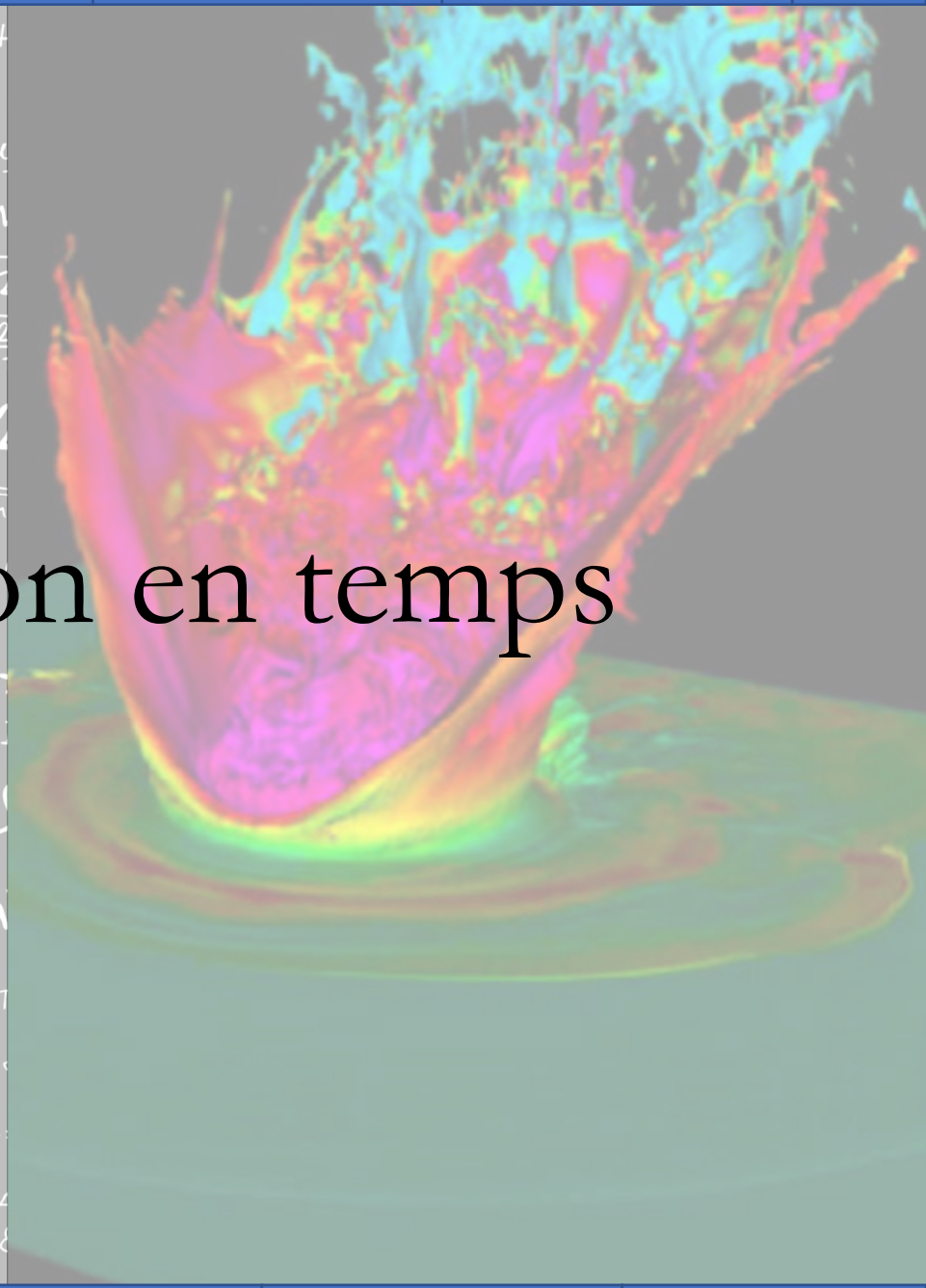
Avantages :

Beaucoup plus rapide car moins coûteux, peu de perte en résolution

Inconvénients :

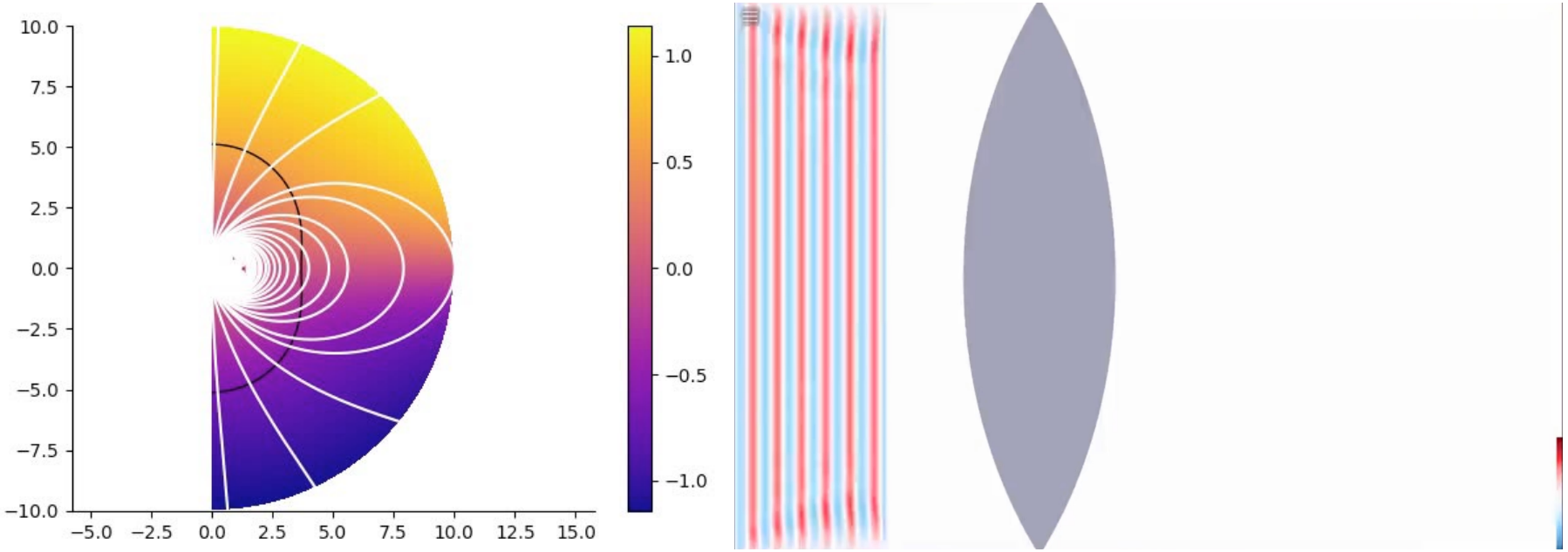
Données finales difficiles à manipuler et visualiser, Résultats dépendants du critère choisi, Beaucoup plus difficile à coder !

Optimisation en temps



Rappel de l'évolution temporelle

On a vu qu'une fois la solution calculée sur le domaine donné, on pouvait avoir besoin de calculer son évolution temporelle (convergence ou modulation) :



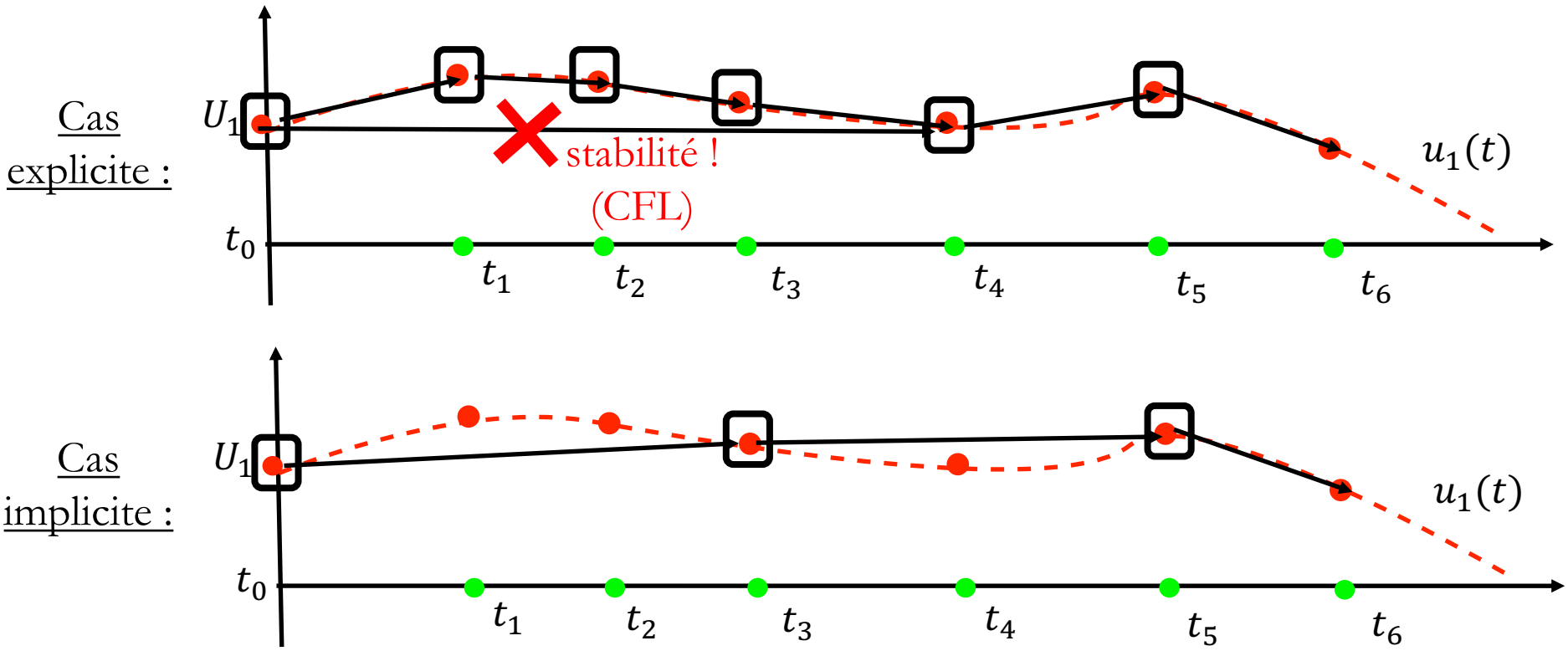
[Perri+2018]

Le temps de calcul dépend alors du temps nécessaire pour atteindre l'état / la séquence d'états voulus

→ Que pourrait-on faire pour optimiser cette étape ?

Rappel : Explicite vs. Implicite

On a vu dans le cours que les schémas implicites étaient inconditionnellement stables :



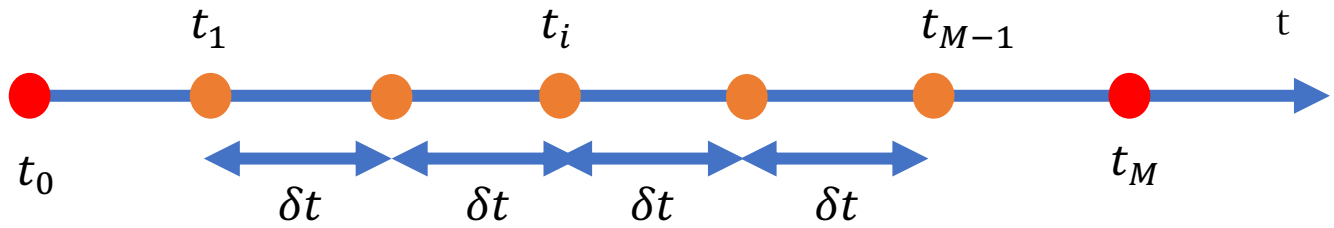
→ Ça permet d'atteindre beaucoup plus rapidement les états de convergence !

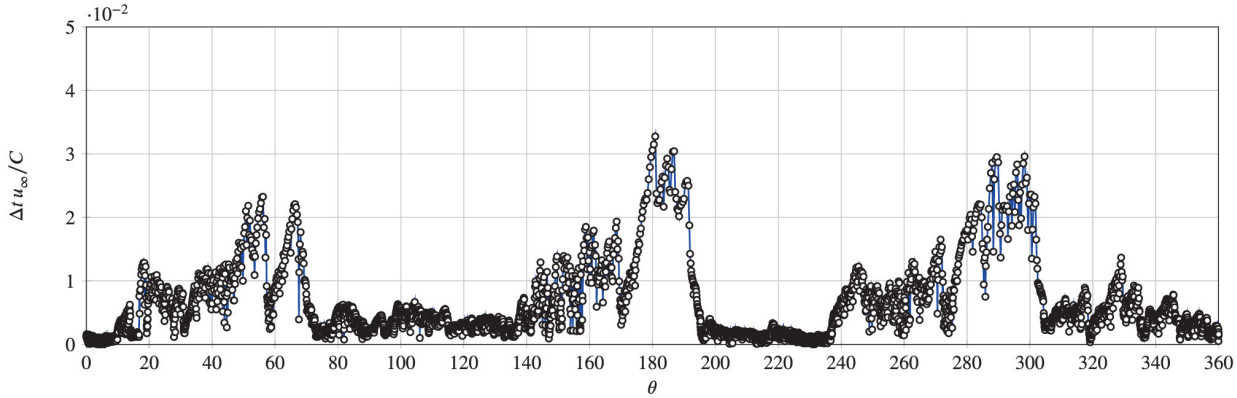
→ MAIS dépend de la matrice à inverser + pas applicable pour modulations

Notion de pas de temps adaptatif

Une option possible est de faire comme avec la discrétisation spatiale

→ Utiliser une discrétisation temporelle non régulière, optimisée pour le problème :



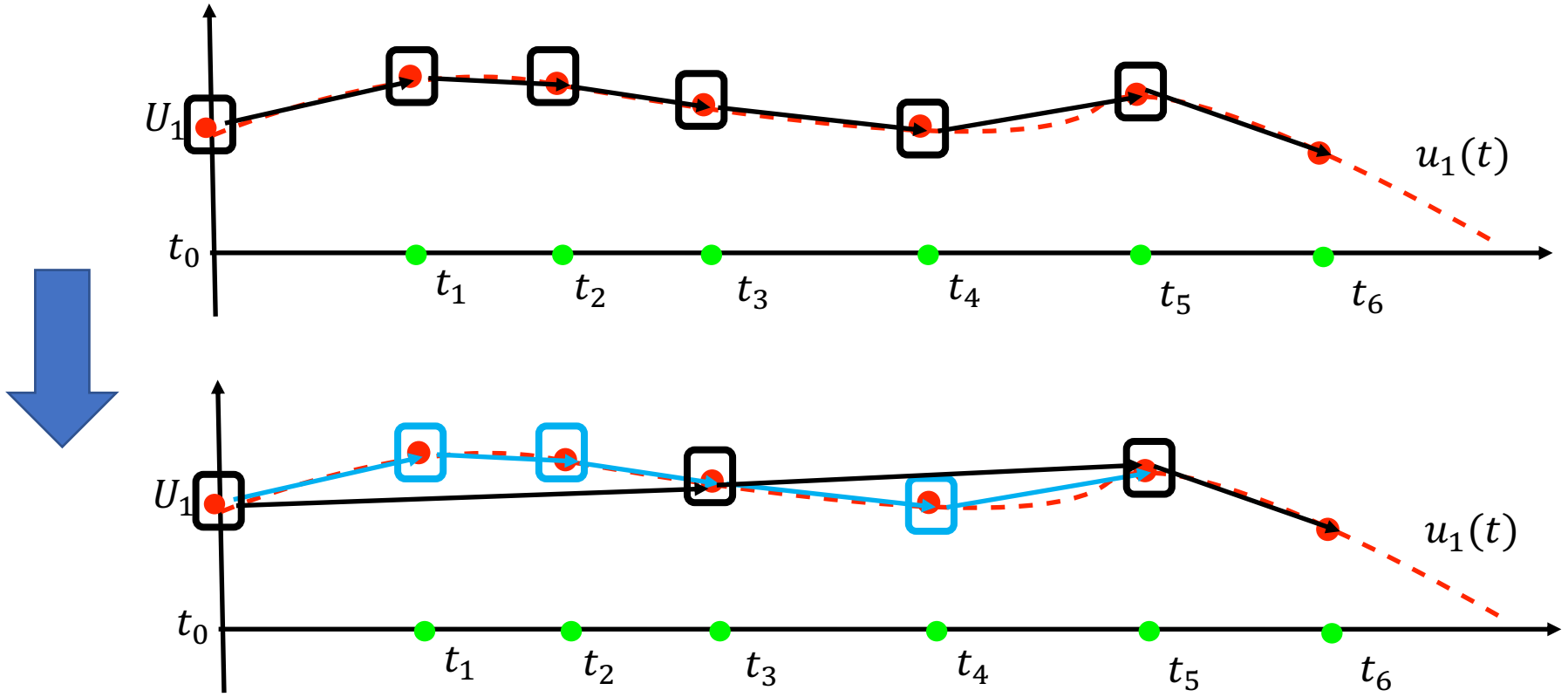


[Ghidoni+2022]

→ À chaque pas de temps, on prend le temps de calculer le pas de temps maximal possible pour la prochaine étape

Algorithmes pararéels

Une option plus osée conceptuellement est d'arrêter de considérer le temps comme séquentiel :



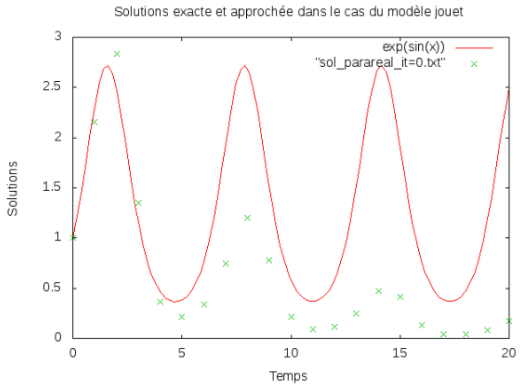
→ On peut également utiliser une approche similaire quand des phénomènes physiques à différentes échelles sont assez découplés pour utiliser des solveurs différents

Exemples d'optimisation en temps

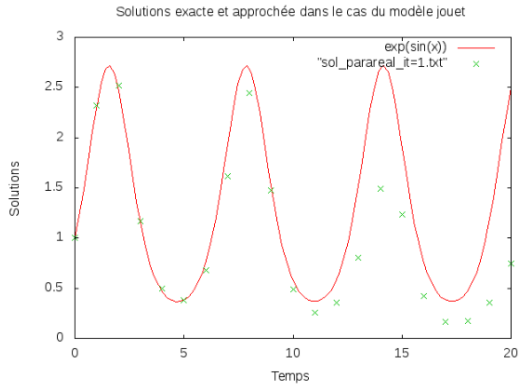
Exemple sur une équation d'onde

→ Il faut plusieurs synchronisations pour obtenir la bonne solution

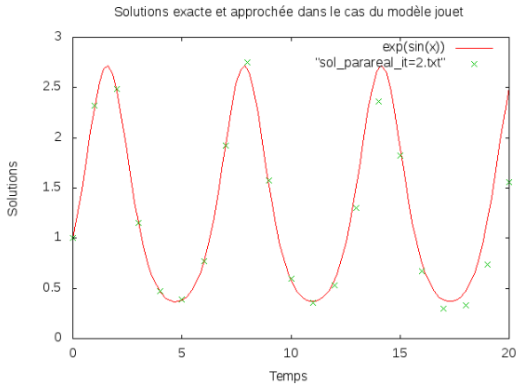
→ Mais globalement plus rapide que si on utilisait un seul solveur fin, et plus précis que si on utilisait un seul solveur grossier



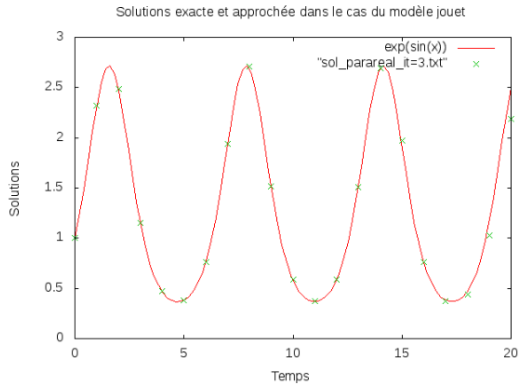
(a) Solutions exacte et approchée à l'itération 1.



(b) Solutions exacte et approchée à l'itération 2.



(c) Solutions exacte et approchée à l'itération 3.



(d) Solutions exacte et approchée à l'itération 4.

Pré-conditionnement pour implicite

On a vu que les problèmes implicites reviennent à inverser une matrice :

$$F^{n+1} = O_g^{-1}(O_d F^n + G)$$

→ Toute la difficulté revient à calculer la matrice O_g^{-1} !

→ Et on a vu qu'avec le nombre de dimensions qui augmente, c'est de plus en plus difficile !

On peut alors simplifier le problème en utilisant un pré-conditionnement :

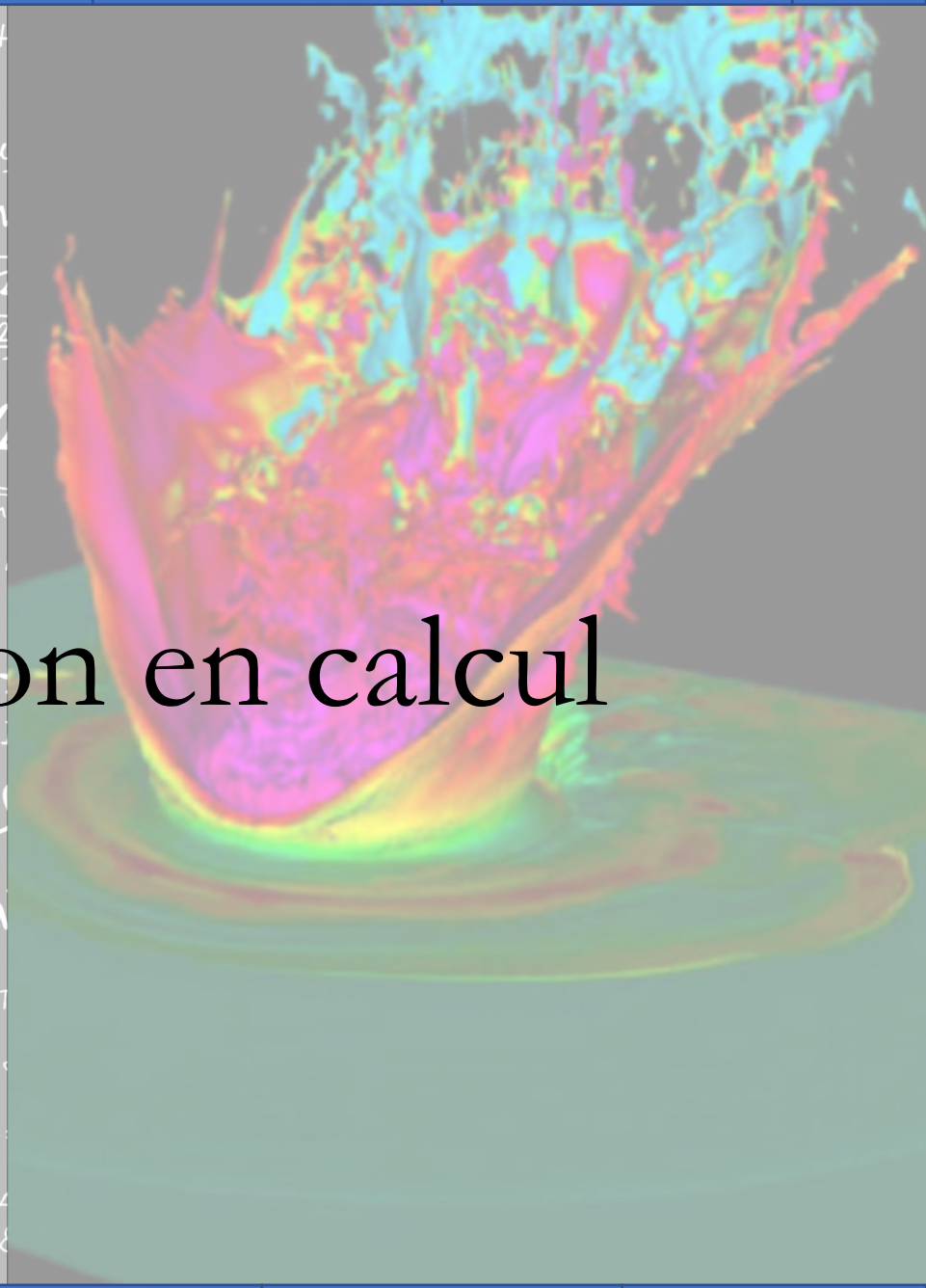
$$AX = B \quad \Longleftrightarrow \quad MX = B'$$

où M est plus facile à inverser que A !

→ Plein de manières de procéder, mais aucune générale (problème-dépendant) :

$$LAX = LB \qquad ARY = B \qquad LARY = LB \qquad S^T ASY = S^T B$$

Optimisation en calcul



Python comme langage de programmation

Dans ce cours, nous avons utilisé le langage Python comme langage de programmation

→ c'est un langage dit **interprété** :

Avantages :

Plus facile à lire et à écrire

Multi-plateformes

Inconvénients :

Nécessite la présence d'un interpréteur

Moins performant qu'un langage compilé

Pour exécuter les scripts, nous avons également utilisé des Jupyter Notebooks

= environnement de programmation interactif

→ Il s'agit de programmation en local sur l'ordinateur en question :

Avantages :

Toujours disponible

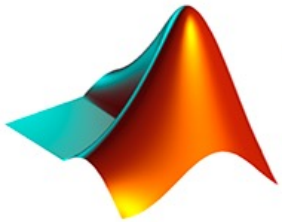
Inconvénients :

Limité par la puissance de l'ordinateur

→ Comment peut-on également optimiser les outils de programmation ?

Autres langages de programmation

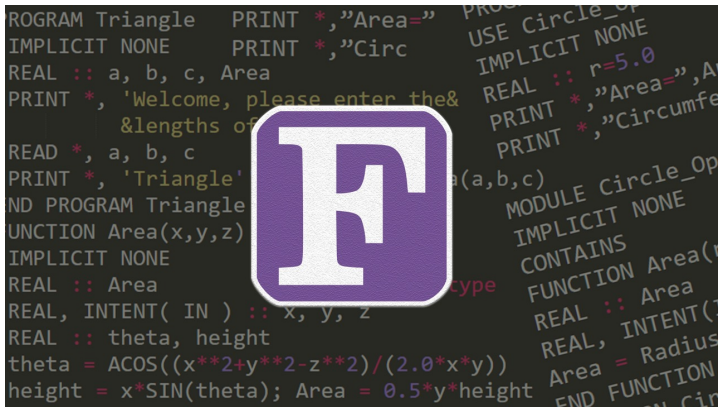
Il existe d'autres langages interprétés qui possèdent d'autres avantages :



MATLAB

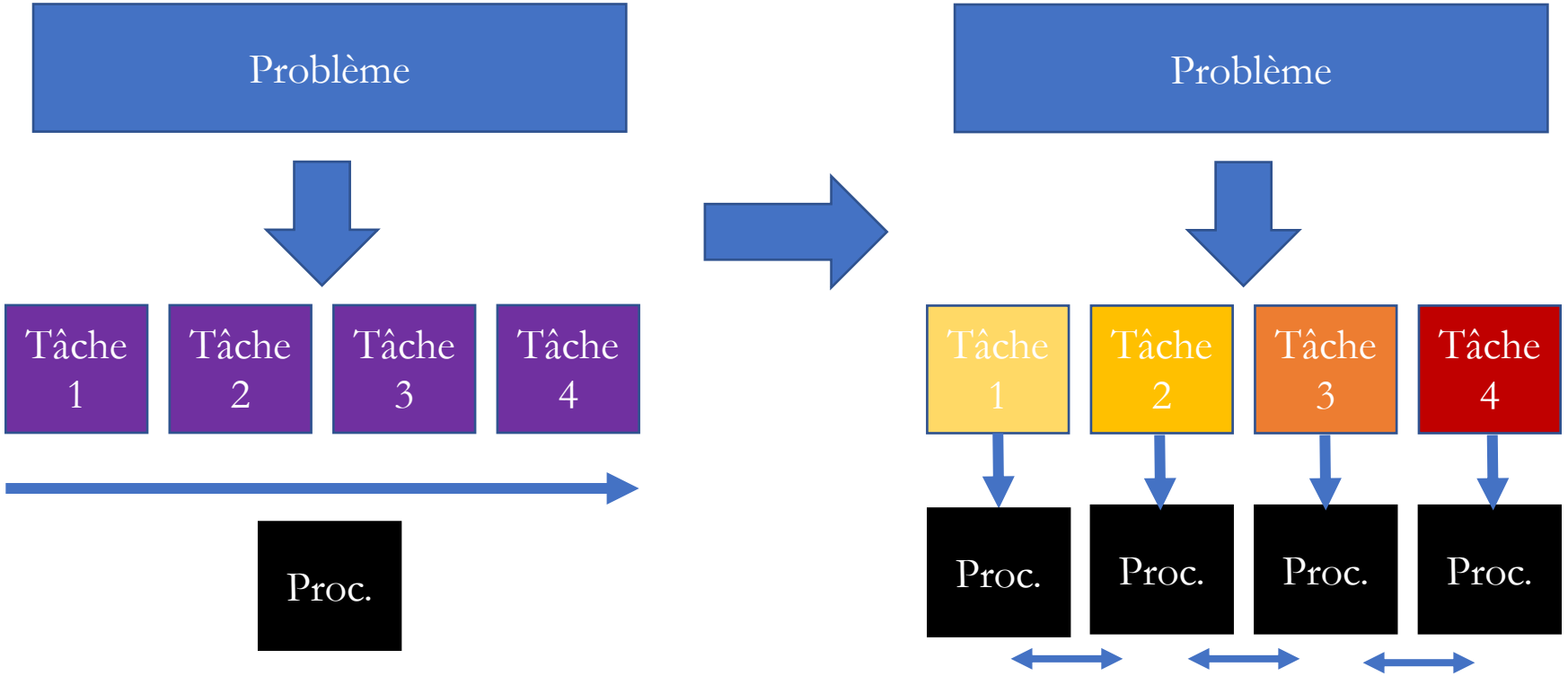
Un langage non interprété est dit **compilé**

→ On n'exécute pas directement le script, on doit d'abord le compiler pour obtenir un exécutable qui dépend de la plate-forme de compilation



Notions de calcul parallèle

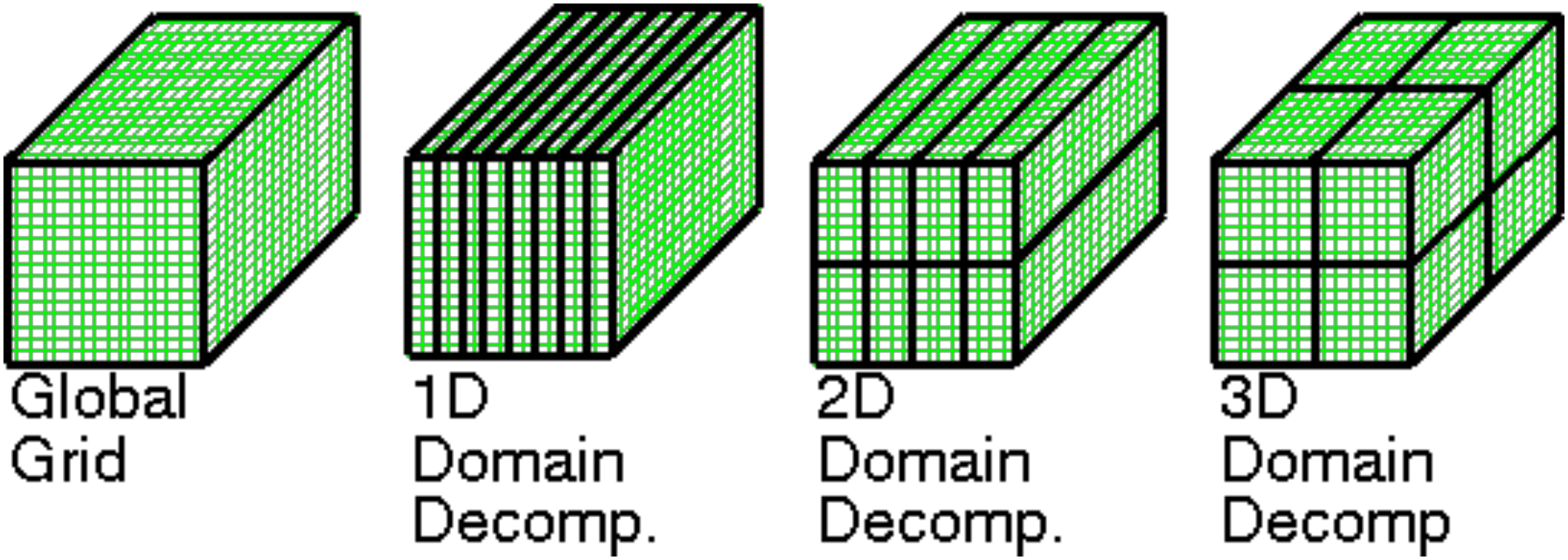
Dans les codes que nous avons utilisés, nous avons codé en **séquentiel**
= on n'utilise qu'un seul processeur pour effectuer
toutes les tâches requises les unes après les autres



→ Coder en **parallèle** permet de mieux exploiter les ressources de l'ordinateur
→ MAIS il faut identifier au préalable des sous-tâches indépendantes

Exemples de parallélisation

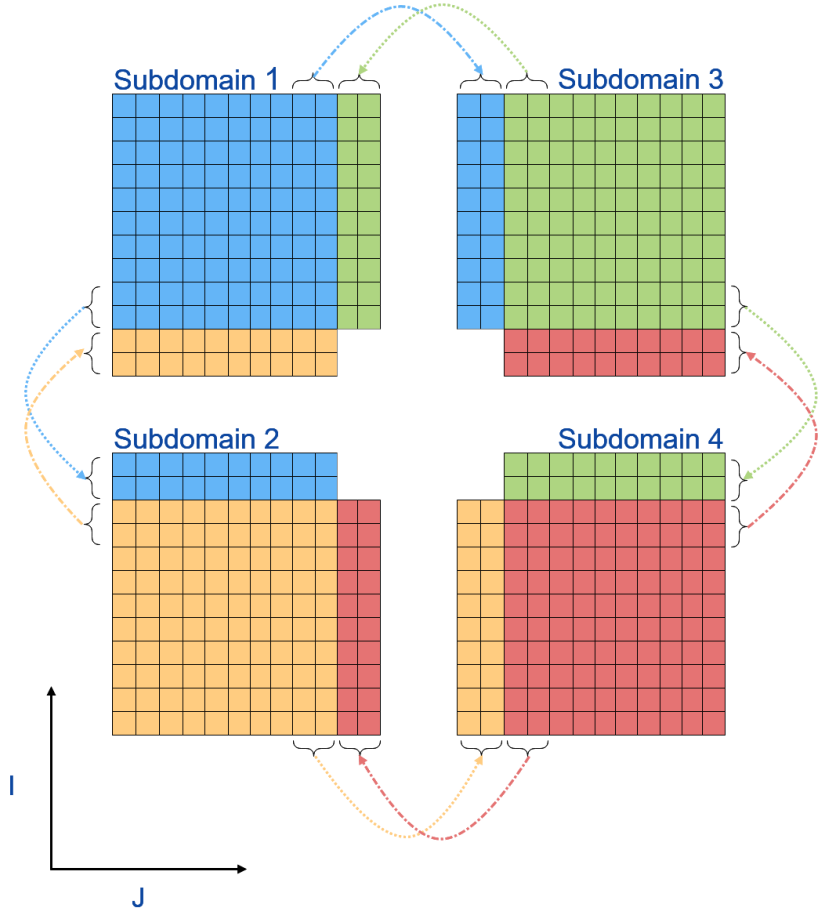
Le cas le plus utilisé pour la parallélisation d'algorithmes
est la **décomposition de domaines**



NB : Les algorithmes pararéels constituent eux une parallélisation du temps !

Difficultés du calcul parallèle

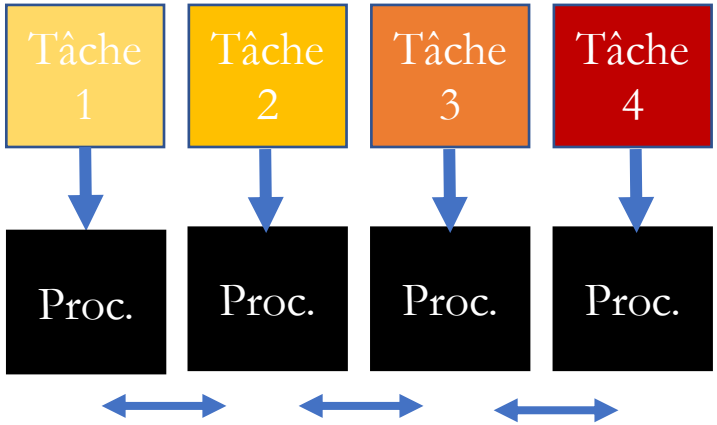
Le calcul parallèle implique cependant d'identifier les sous-tâches parallélisables
+ d'établir des **communications** entre processeurs pour échanger les informations

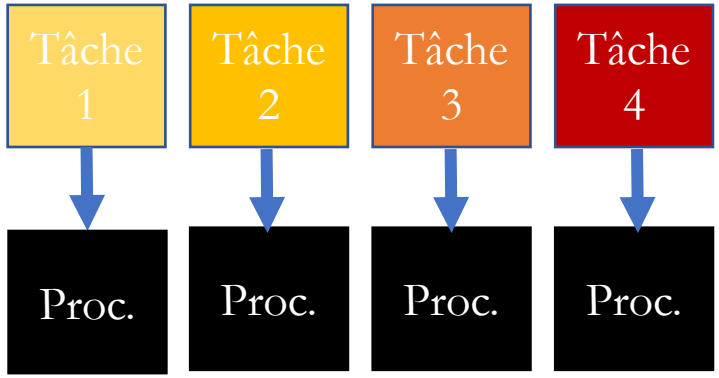


→ La gestion des conditions aux limites devient beaucoup plus difficile par exemple !

Différents types de parallélisme

On distingue ce qu'on appelle le **calcul distribué** (parallélisme avec communications) de ***l'embarrasingly parallel*** (parallélisme parfait)

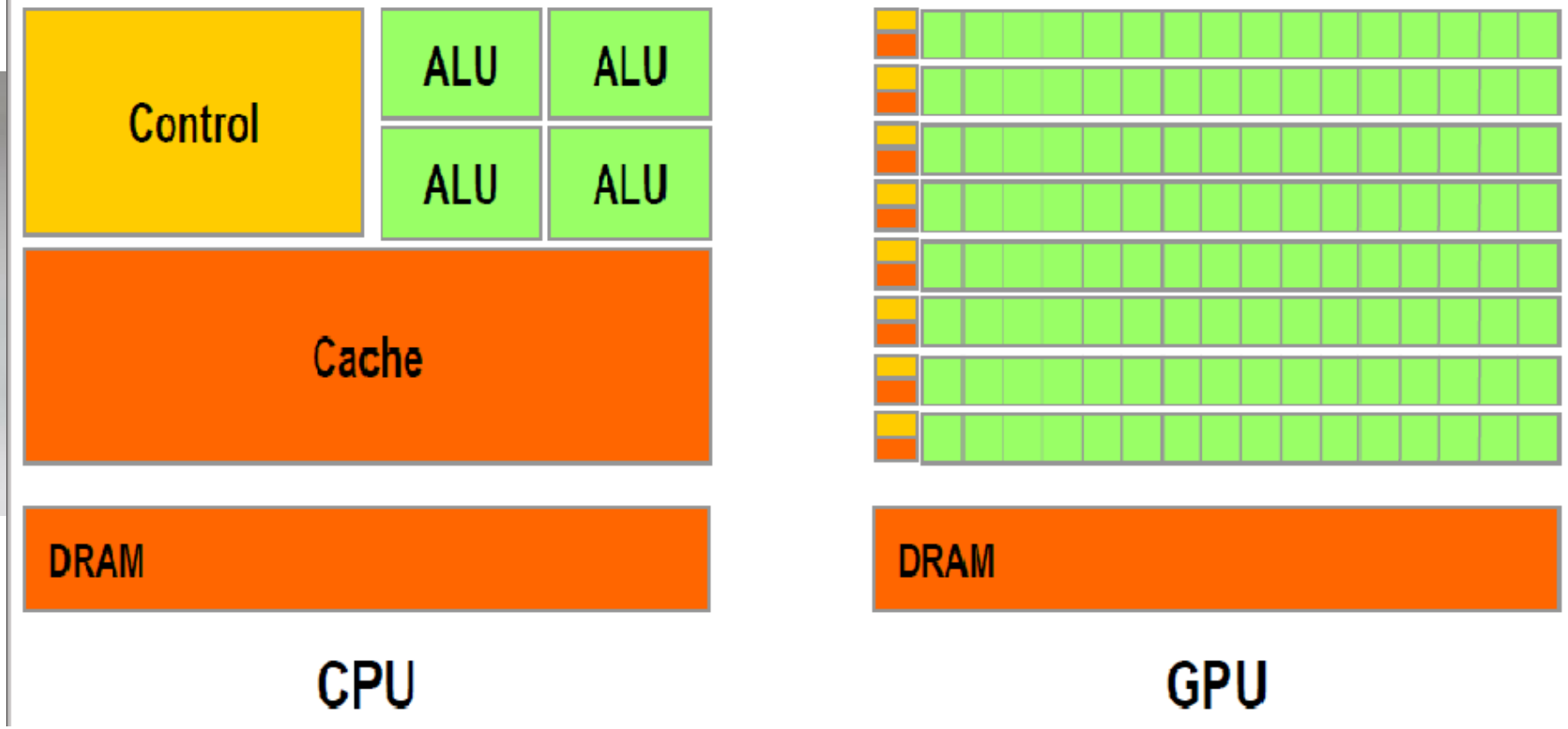




→ Pour le calcul distribué, on utilise plutôt OpenMPI
→ Pour *l'embarrasingly parallel*, on utilise plutôt OpenMP

Différentes architectures informatiques

On programme le plus souvent sur des processeurs (**CPU**)
MAIS on peut maintenant aussi programmer sur carte graphique (**GPU**)



→ Le CPU est plus optimisé pour le calcul distribué,
→ Le GPU est plus optimisé pour l'*embarrasingly parallel*

Supercalculateurs

Pour faire tourner des codes très coûteux, on a besoin d'une excellente architecture

→ On ne calcule plus sur son ordinateur, mais sur un supercalculateur

= machine très puissante de renommée nationale ou internationale !

TGCC en France
200 000 cœurs
83ème meilleure machine au monde

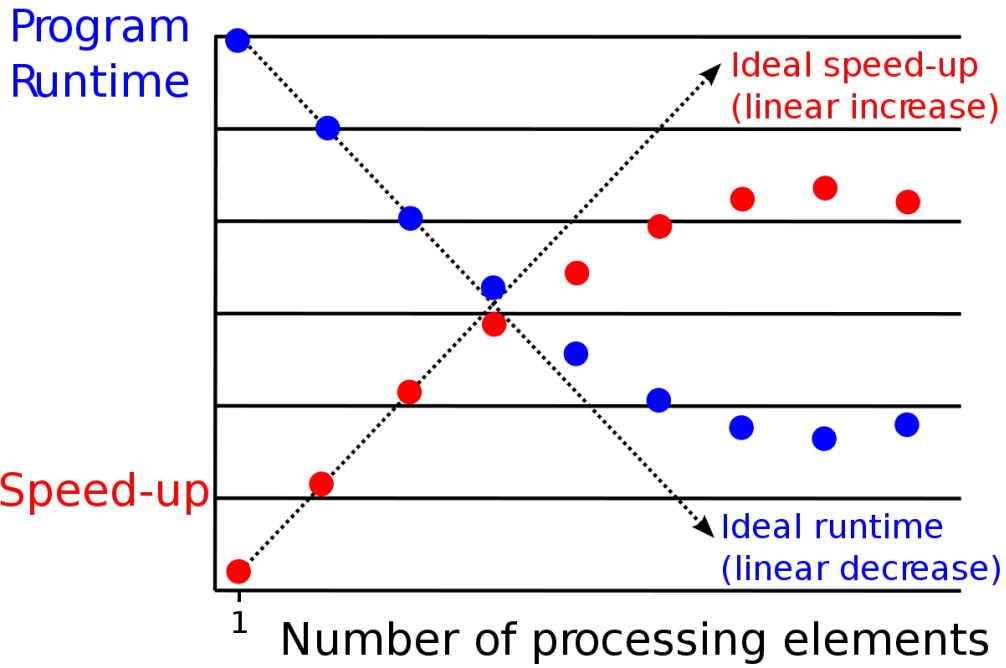
Adastra (GENCI) en France
300 000 cœurs
11ème meilleure machine au monde

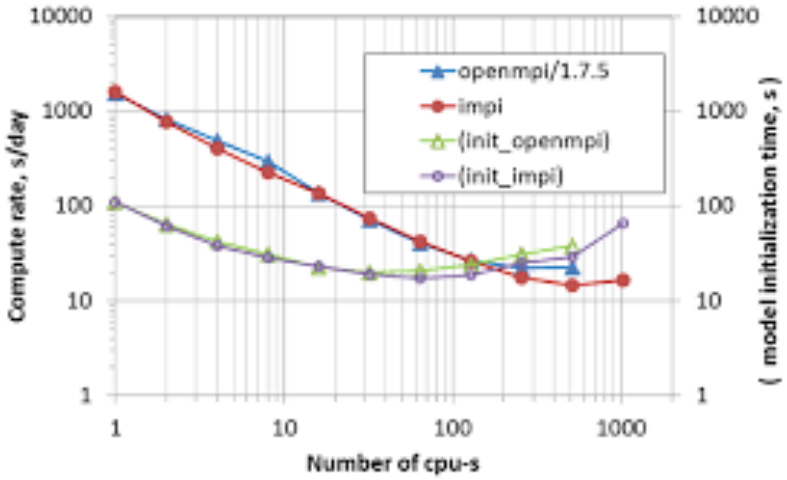
→ On doit alors se connecter à distance (SSH)

→ Il faut gérer différemment son espace de travail

Accélération de codes parallélisés

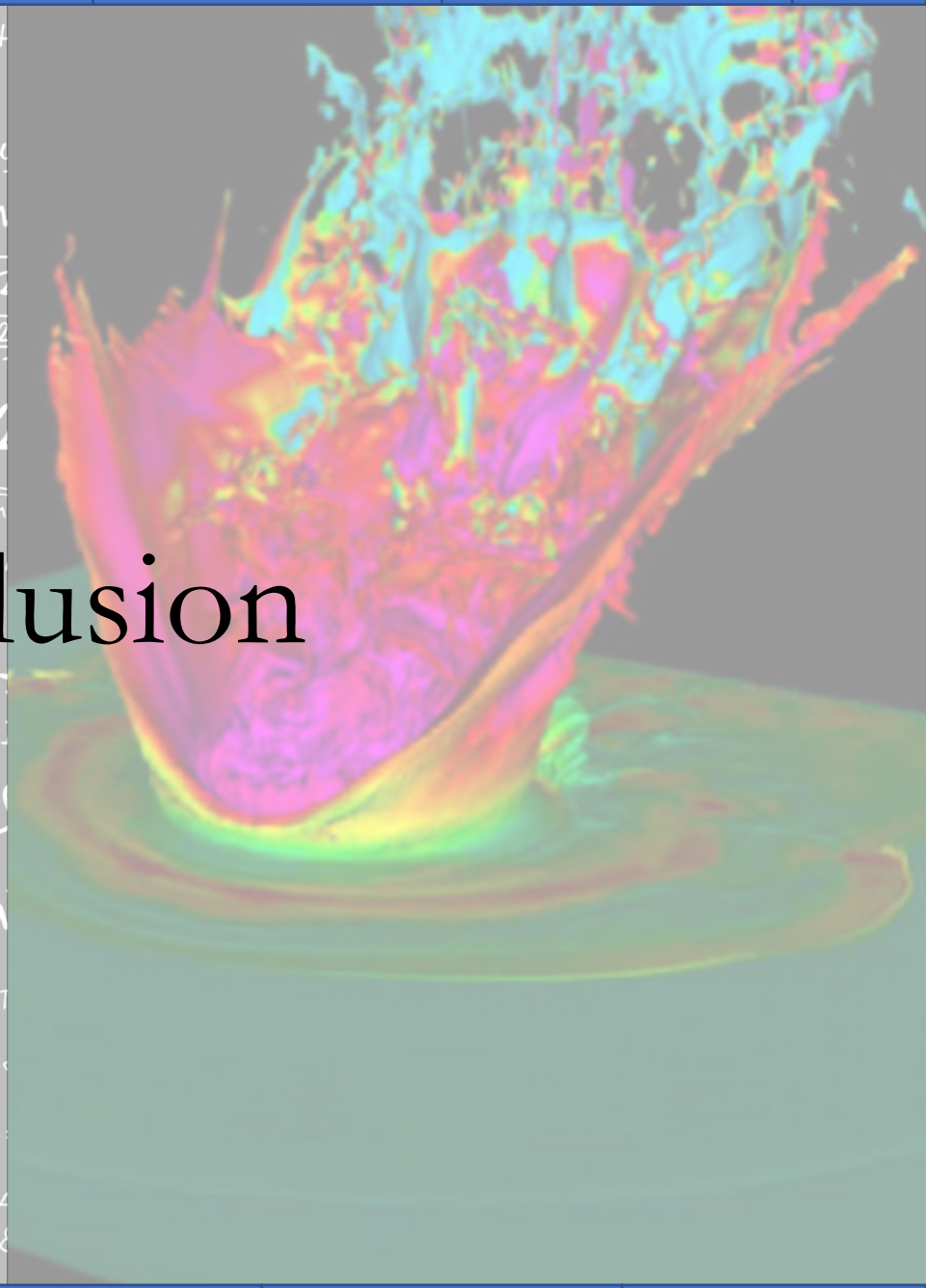
Une fois qu'un code est parallélisé, il peut calculer très rapidement malgré la taille du domaine de calcul, et d'autant plus rapidement qu'on utilise de CPUs/GPUs
= extensibilité (**scalability** en anglais)





→ Il n'existe pas de *scaling* parfait, mais il est important d'estimer le nombre de processeurs à partir duquel le *scaling* devient moins performant

Conclusion

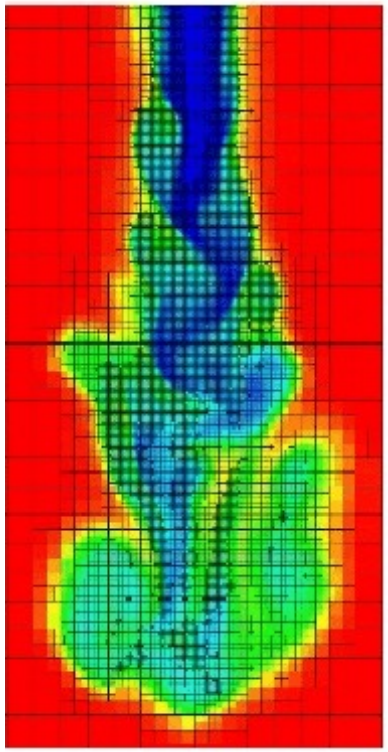


Optimisation des méthodes numériques

Il y a de nombreuses manières d'optimiser un code numérique :

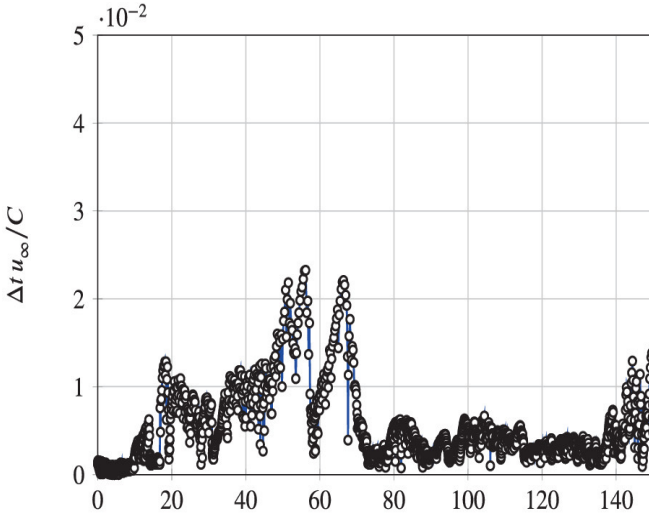
En espace

→ AMR (grille adaptative)



En temps

→ Pas de temps adaptatif
→ Parallélisation en temps
→ Pré-conditionnement



En calcul

→ Parallélisation du code
→ Architecture (CPU/GPU)
→ Supercalculateur

Résumé des compétences à maîtriser

Cours 1 :

Vocabulaire des méthodes numériques.

Cours 2 :

- Méthode de Newton,
- Polynôme de Lagrange,
- Quadrature d'interpolation.

Cours 3 :

Différences finies :

- Par tangente,
- Par les développements limités,
- Par le polynôme de Lagrange.

Cours 4 :

- Résolution d'un problème 1D stationnaire,
- Problème aux valeurs propres.

Cours 5 :

- Analyse de l'ordre d'un schéma,
- Analyse de stabilité d'un schéma.

Cours 6 :

- Résolution d'un problème 1D instationnaire,
- Résolution avec schéma explicite ou implicite.

Cours 7 :

- Passage au 2D,
- Numérotation par blocs,
- Programmation en matrices creuses.

Cours 8 :

Méthodes d'optimisation numérique.

Résumé des modalités d'évaluation

Comptes-rendus de TP

Questions de cours (oral)

Questions sur les TPs (oral)

Plan de l'UE

Idée générale :

Au premier semestre, on va introduire les notions de base, et s'intéresser en détails à une méthode numérique précise

Tous les jeudi matin (8h45-12h45) au bâtiment 625

21 Novembre : Cours 1 + Cours 2
4 Décembre : Cours 3 + TP
5 Décembre : Cours 4 + TP
12 Décembre : Cours 5 + TP
19 Décembre : Cours 6 + TP
9 Janvier : Cours 7 + TP
16 Janvier : Cours 8 + **TP**
23 Janvier : TP
30 Janvier : Examen

→ À la fin de l'UE, vous serez capable de coder vous-mêmes un solveur pour une équation de type Navier-Stokes !