Parallélisme sur architectures à mémoires distribuées

Bastien DI PIERRO

Université Claude Bernard Lyon 1

3 Novembre 2016

Problématique : exemple de la météo

- Lorentz (1963)
- Premier modèle mathématique pour la convection atmosphérique

Problématique : exemple de la météo

- Lorentz (1963)
- Premier modèle mathématique pour la convection atmosphérique
- Petite erreur de décimale

Écart considérable sur les résultats

Problématique : exemple de la météo

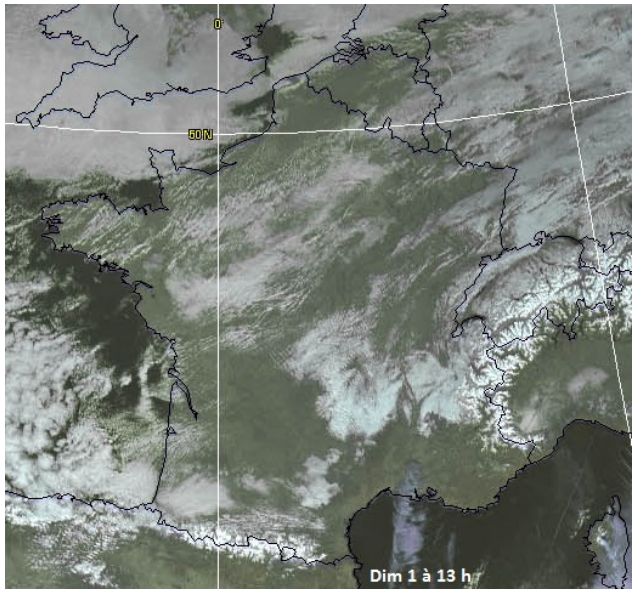
- Lorentz (1963)
- Premier modèle mathématique pour la convection atmosphérique
- Petite erreur de décimale

Écart considérable sur les résultats

- Naissance de la théorie du chaos

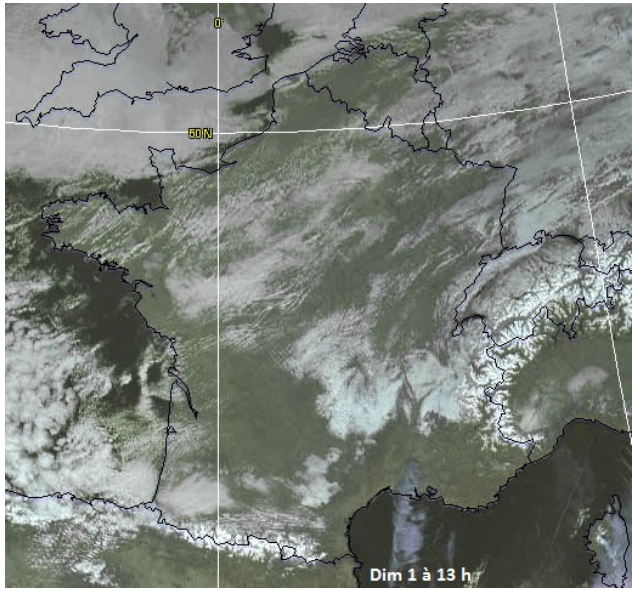
Besoin d'une précision extrême !

Problématique : exemple de la météo



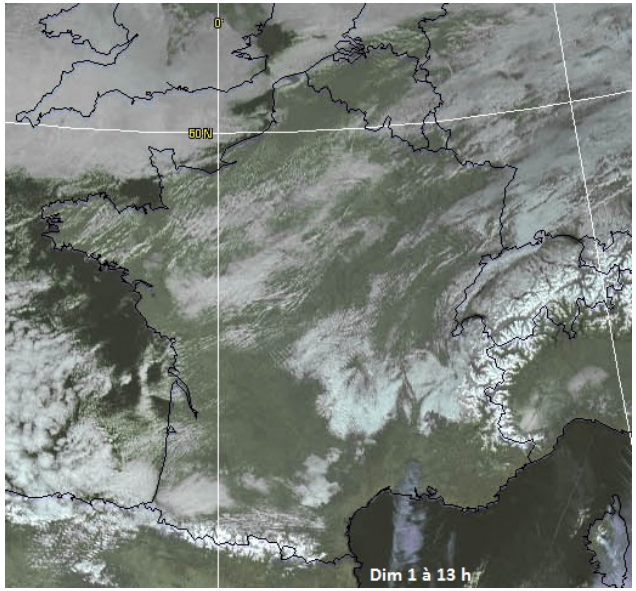
Problématique : exemple de la météo

- Résolution de toutes les échelles
  $\Rightarrow \approx 10^{26} - 10^{27}$ ddl



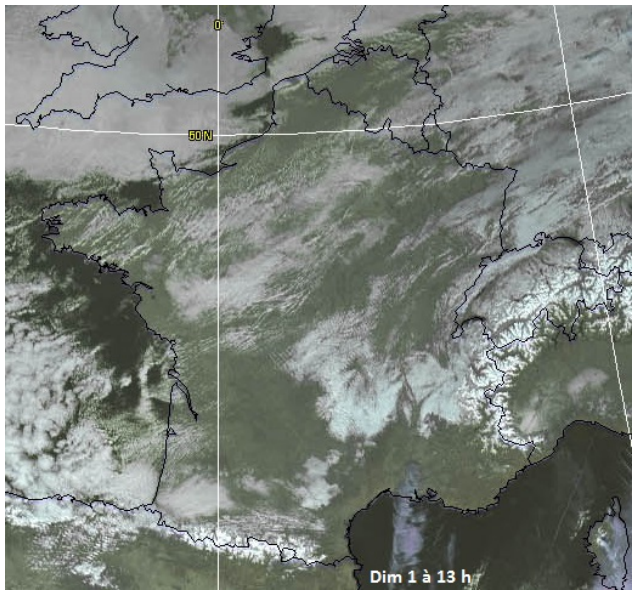
Problématique : exemple de la météo

- Résolution de toutes les échelles
  $\Rightarrow \approx 10^{26} - 10^{27}$ ddl
- Schéma le moins coûteux
  $\Rightarrow \approx 10^{17} - 10^{18}$ Go RAM



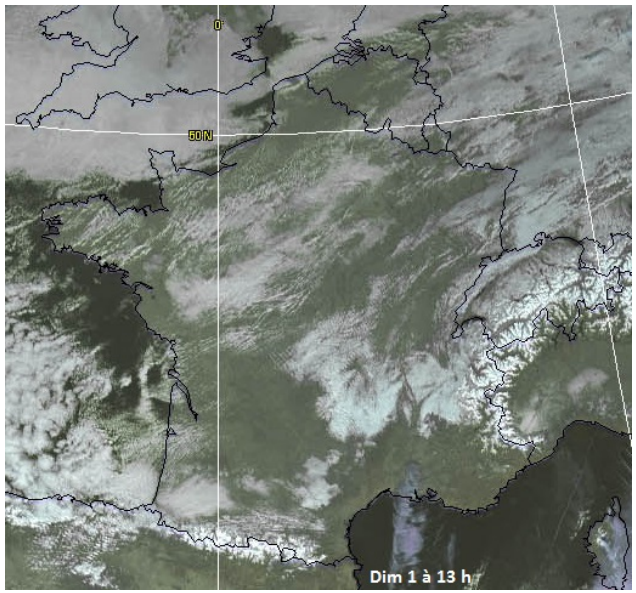
Problématique : exemple de la météo

- Résolution de toutes les échelles
  $\Rightarrow \approx 10^{26} - 10^{27}$ ddl
- Schéma le moins coûteux
  $\Rightarrow \approx 10^{17} - 10^{18}$ Go RAM
- Processeur classique (≈ 3 GHz)
- Intensité arithmétique idéale



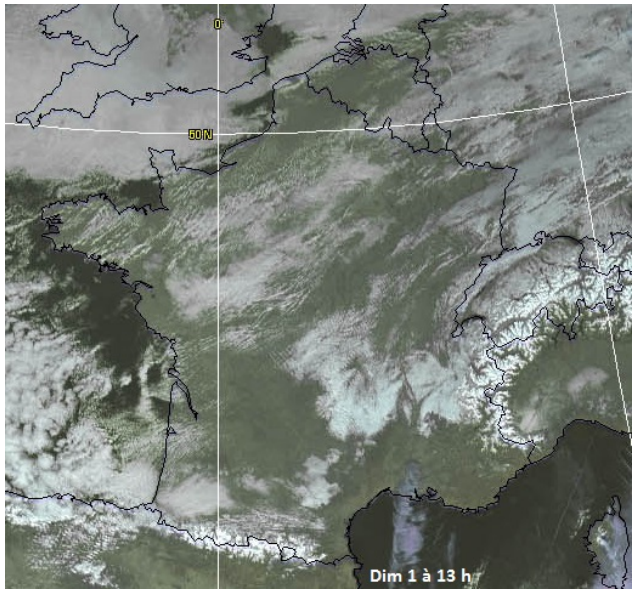
Problématique : exemple de la météo

- Résolution de toutes les échelles
  $\Rightarrow \approx 10^{26} - 10^{27}$ ddl
- Schéma le moins coûteux
  $\Rightarrow \approx 10^{17} - 10^{18}$ Go RAM
- Processeur classique (≈ 3 GHz)
- Intensité arithmétique idéale
- Simulation d'une heure physique
  $\Rightarrow \approx 10^{26}$ secondes
  $\approx 3.10^{8}$ âge de l'univers



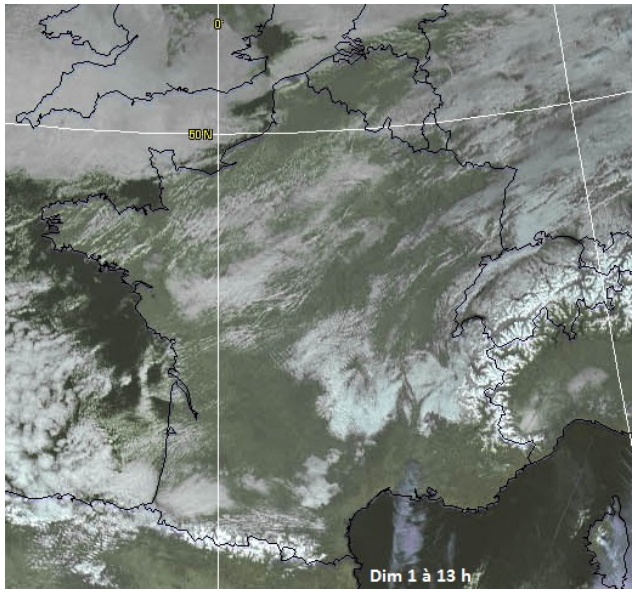
Problématique : exemple de la météo

- 1 “mesure” /100 m
  $\Rightarrow \approx 10^{11} - 10^{12}$ ddl



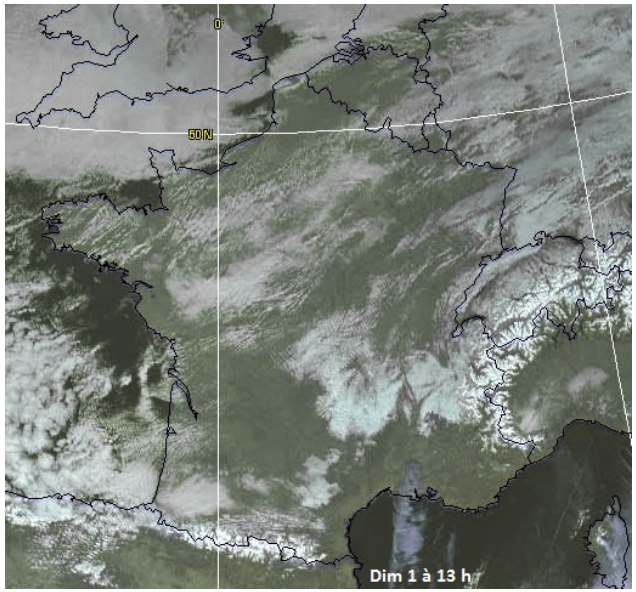
Problématique : exemple de la météo

- 1 “mesure” /100 m
  $\Rightarrow \approx 10^{11} - 10^{12}$ ddl
- Schéma le moins coûteux
  $\Rightarrow \approx$ 200 Go RAM



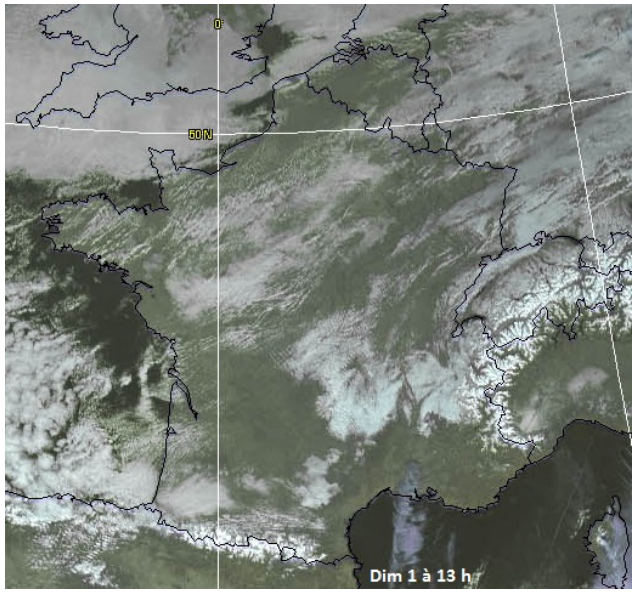
Problématique : exemple de la météo

- 1 “mesure” /100 m
  $\Rightarrow \approx 10^{11} - 10^{12}$ ddl
- Schéma le moins coûteux
  $\Rightarrow \approx$ 200 Go RAM
- Processeur classique ($\approx$ 3 GHz)
- Intensité arithmétique idéale



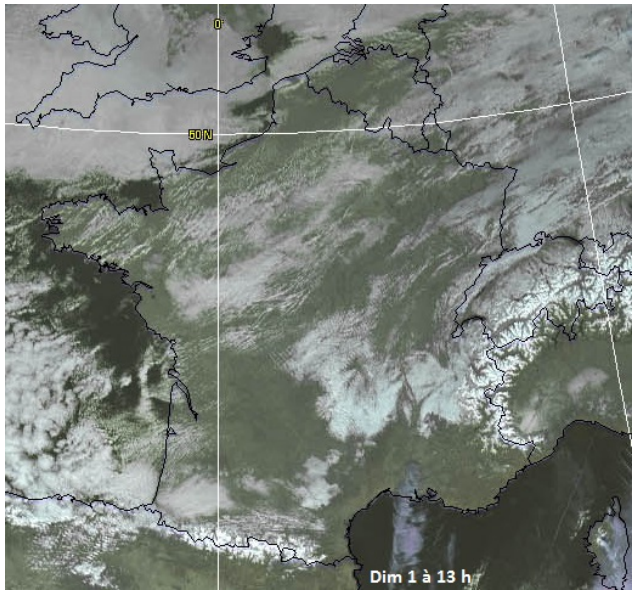
Problématique : exemple de la météo

- 1 “mesure” /100 m
  $\Rightarrow \approx 10^{11} - 10^{12}$ ddl
- Schéma le moins coûteux
  $\Rightarrow \approx$ 200 Go RAM
- Processeur classique ($\approx$ 3 GHz)
- Intensité arithmétique idéale
- Simulation d’une journée réelle
  $\Rightarrow \approx 3.10^7$ secondes
  $\approx$ 1 an



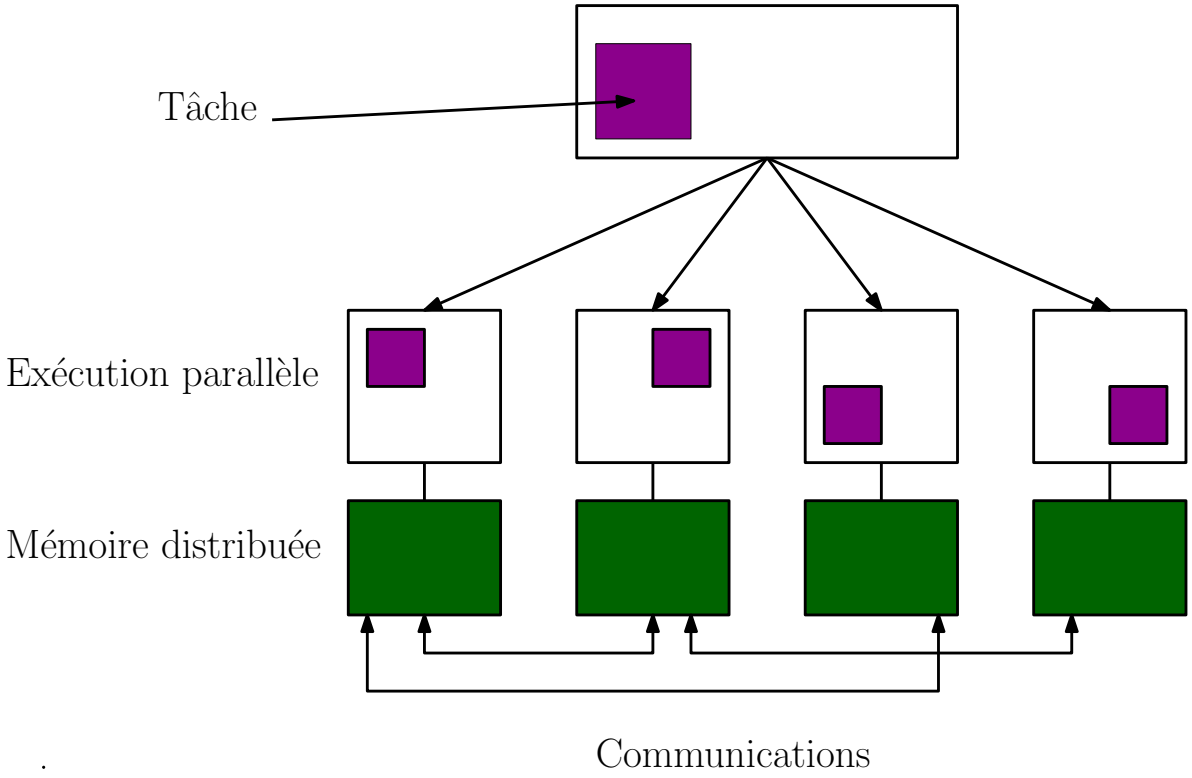
Problématique : exemple de la météo

- 1 “mesure” /100 m
  $\Rightarrow \approx 10^{11} - 10^{12}$ ddl
- Schéma le moins coûteux
  $\Rightarrow \approx$ 200 Go RAM
- Processeur classique ($\approx$ 3 GHz)
- Intensité arithmétique idéale
- Simulation d’une journée réelle
  $\Rightarrow \approx 3.10^7$ secondes
  $\approx$ 1 an



En simulant 365 fois plus vite, on pourrait donner des prévisions sur le lendemain ...
Comment ?

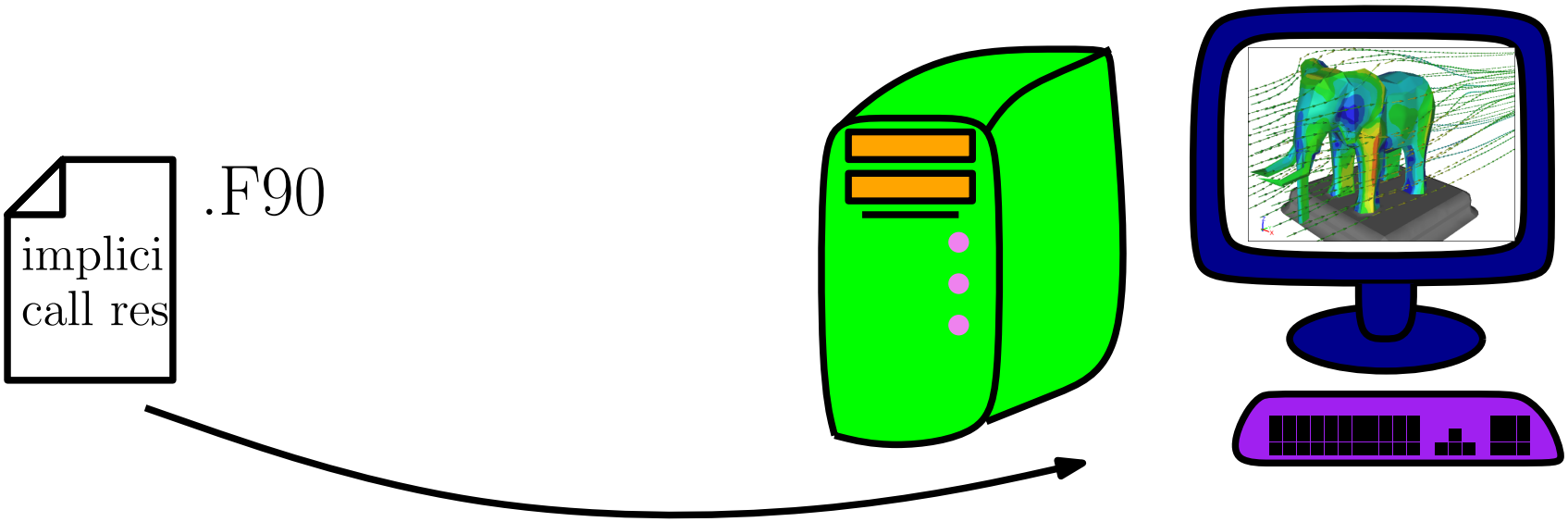
Architecture à mémoires distribuées



paradigme

- Chaque unité de calcul possède sa propre mémoire
  $\Rightarrow$ Pas d'accès concurent !
- Gestion explicite des communications

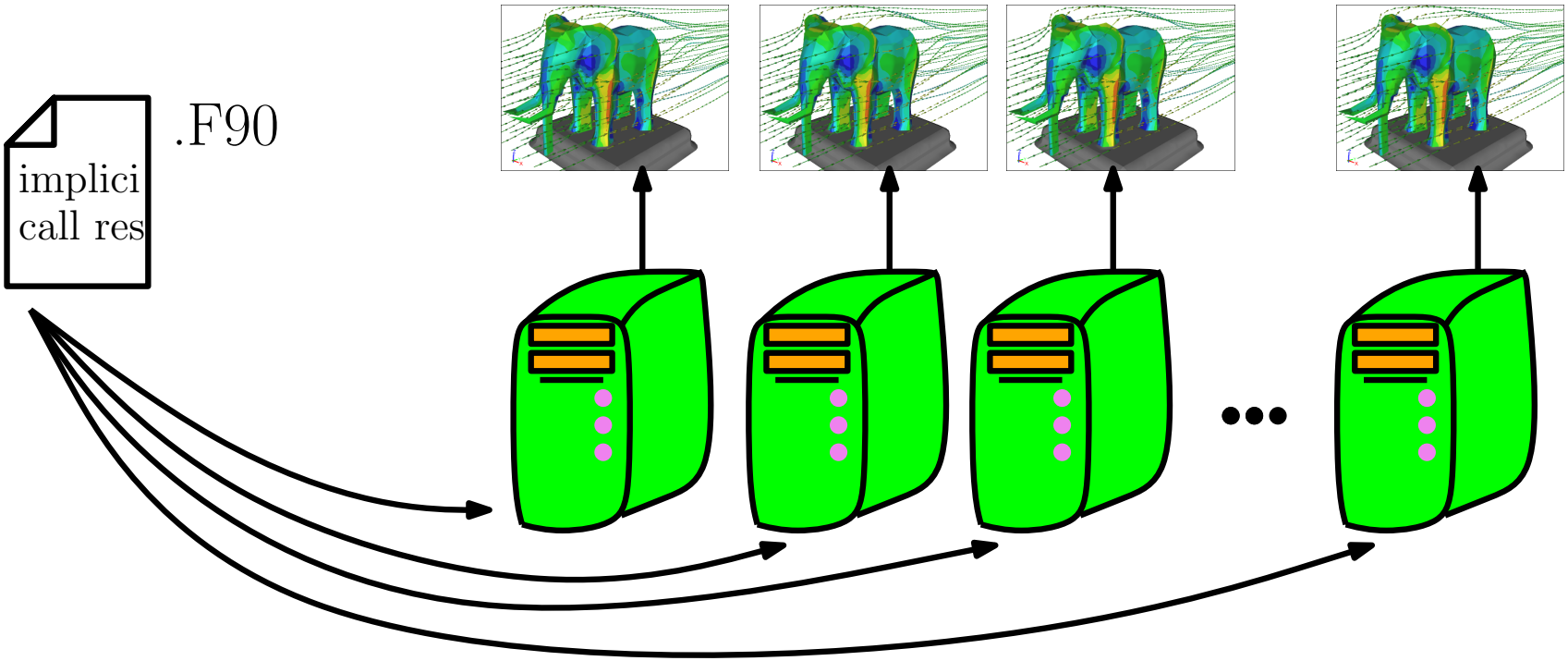
Parallélisation d'un calcul



Démarche

Cas classique : Le programme tourne séquentiellement sur un CPU

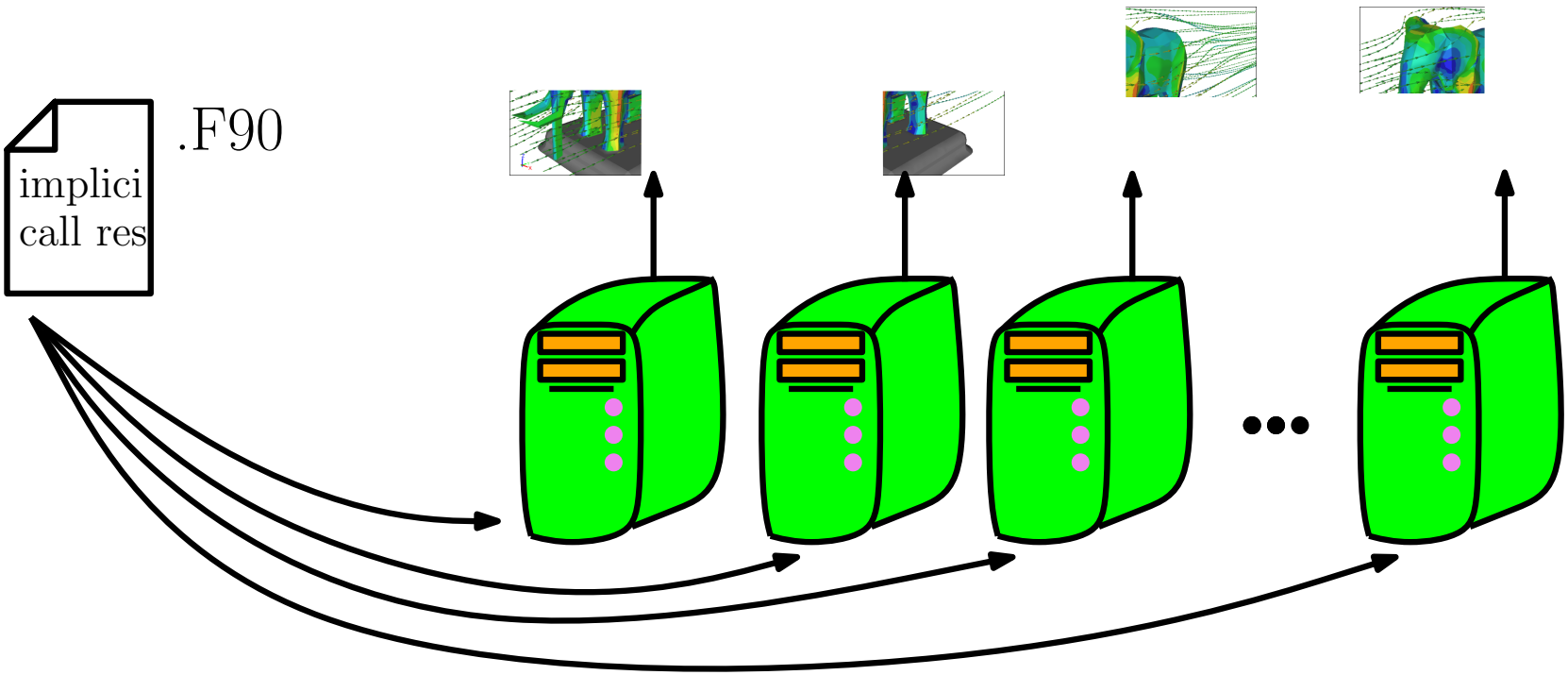
Parallélisation d'un calcul



Démarche

- Le programme est envoyé sur chacun des noeuds de calculs
  $\Rightarrow$ le même résultat pour chacun des noeuds

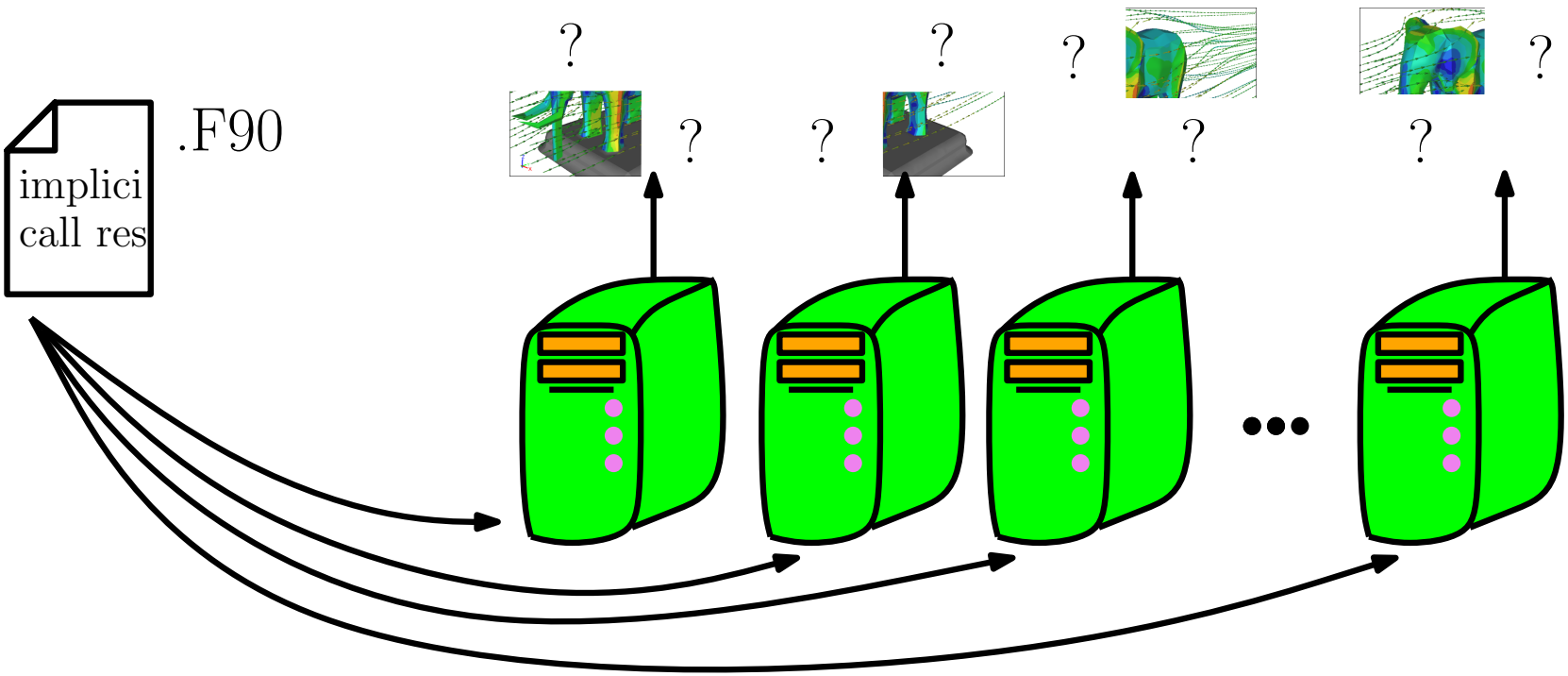
Parallélisation d'un calcul



Démarche

- Le programme est envoyé sur chacun des noeuds de calculs
- Le programme découpe sa tâche en **sous-tâches**

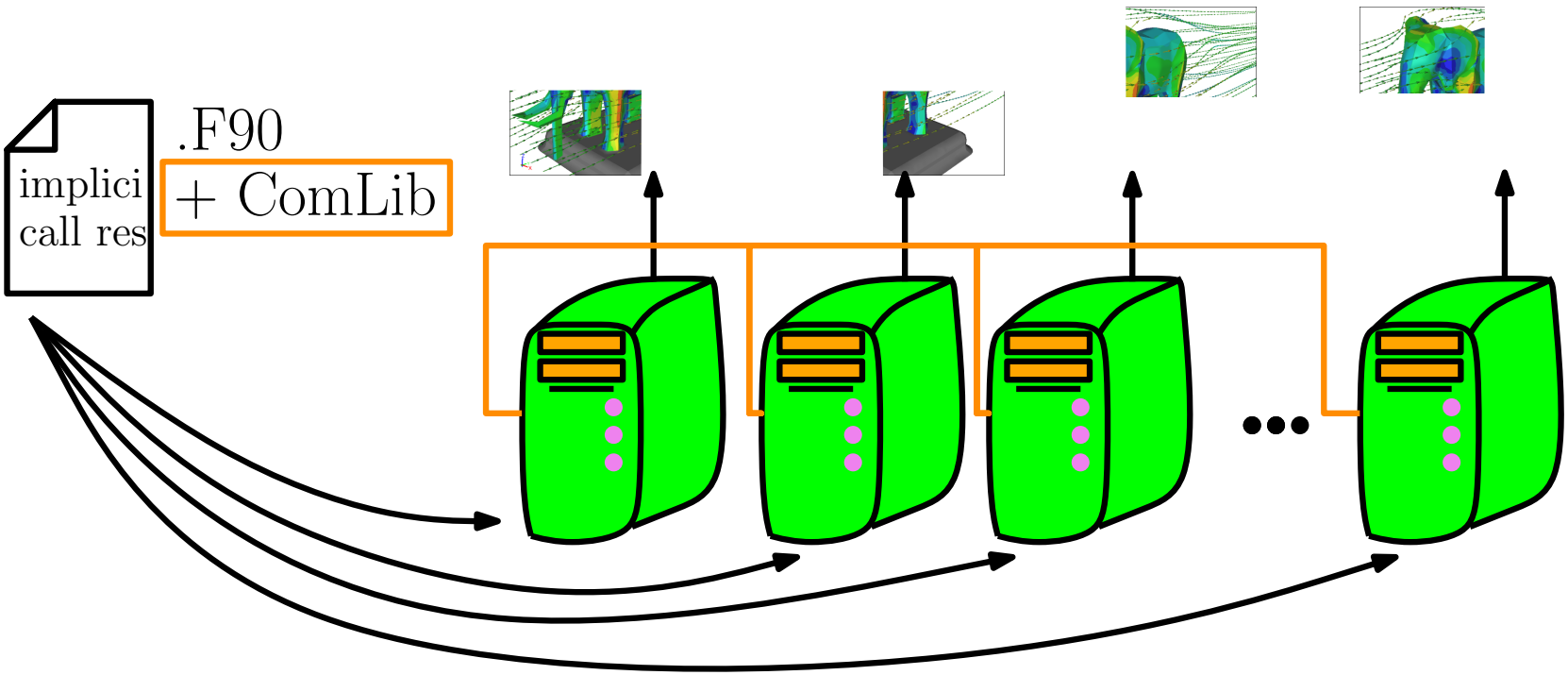
Parallélisation d'un calcul



Démarche

- Le programme est envoyé sur chacun des noeuds de calculs
- Le programme découpe sa tâche en **sous-tâches**
  Les sous-tâches ont besoin de communiquer !

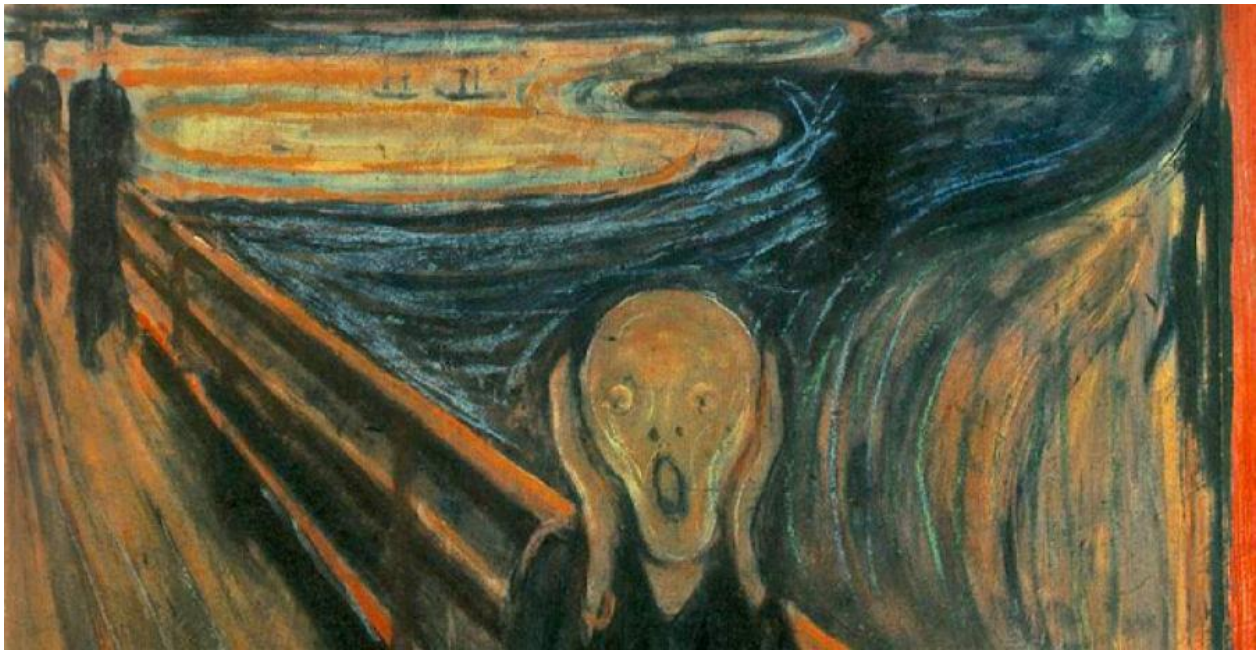
Parallélisation d'un calcul



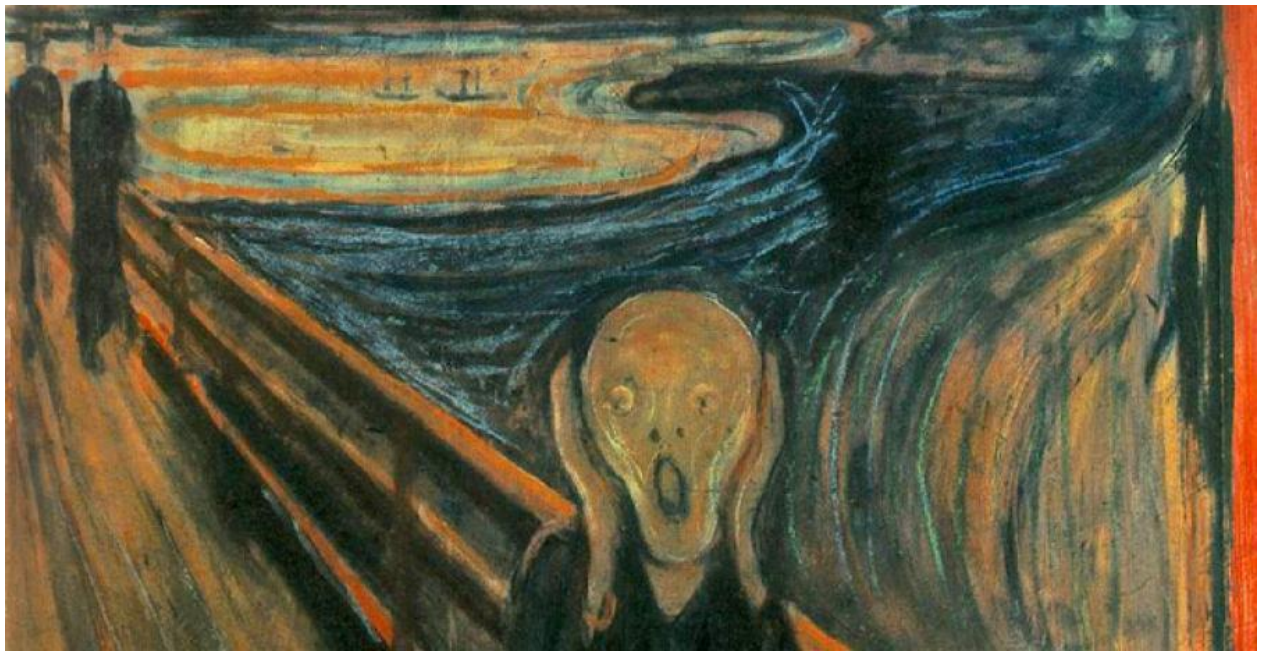
Démarche

- Le programme est envoyé sur chacun des noeuds de calculs
- Le programme découpe sa tâche en **sous-tâches**
- Les noeuds sont branchés sur un réseau de **communication**

Exemple : reproduction d'une peinture

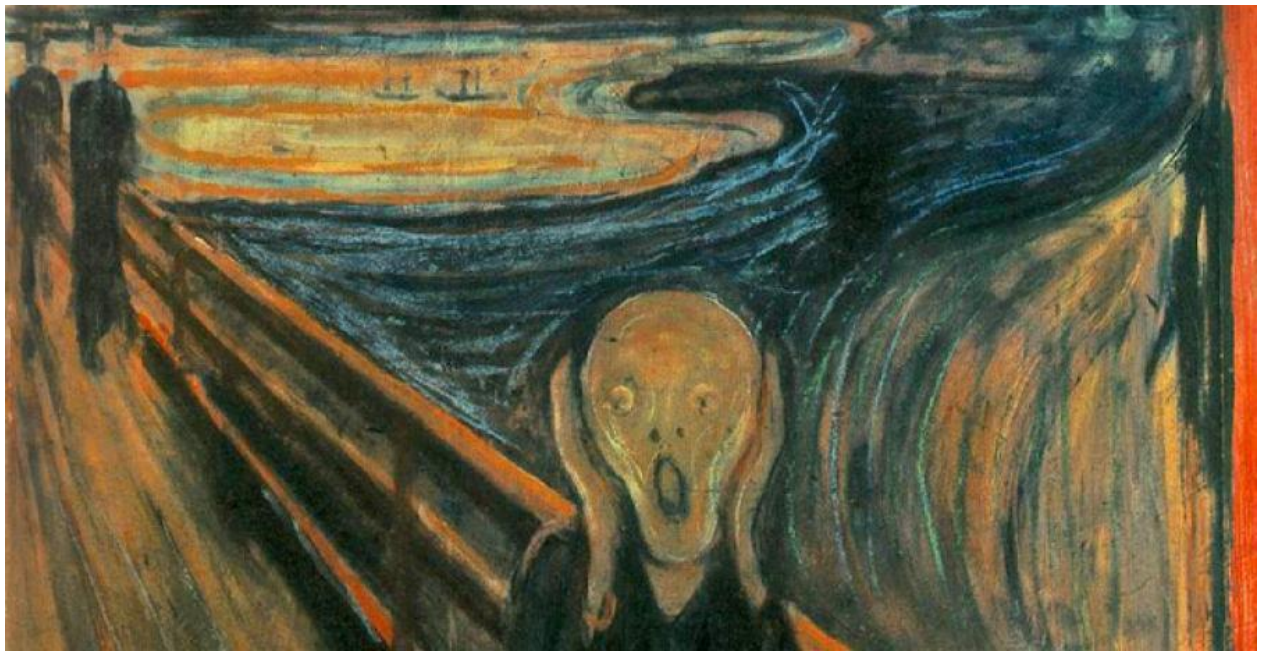


Exemple : reproduction d'une peinture



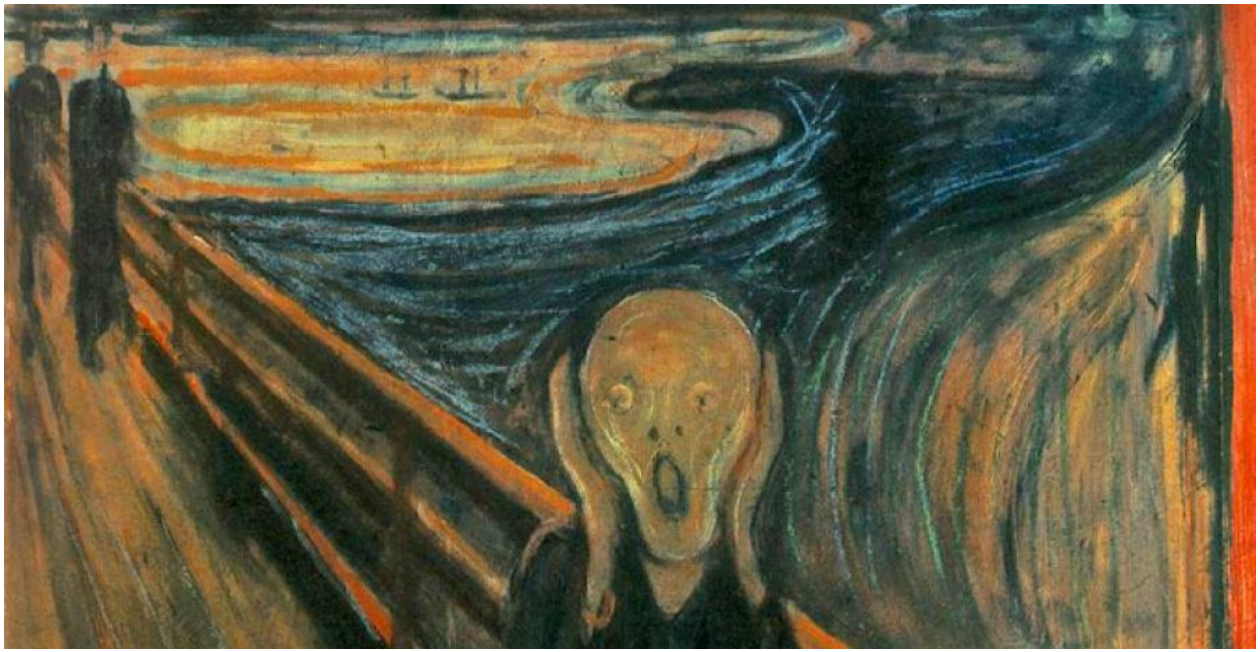
temps total : 4 jours

Exemple : reproduction d'une peinture



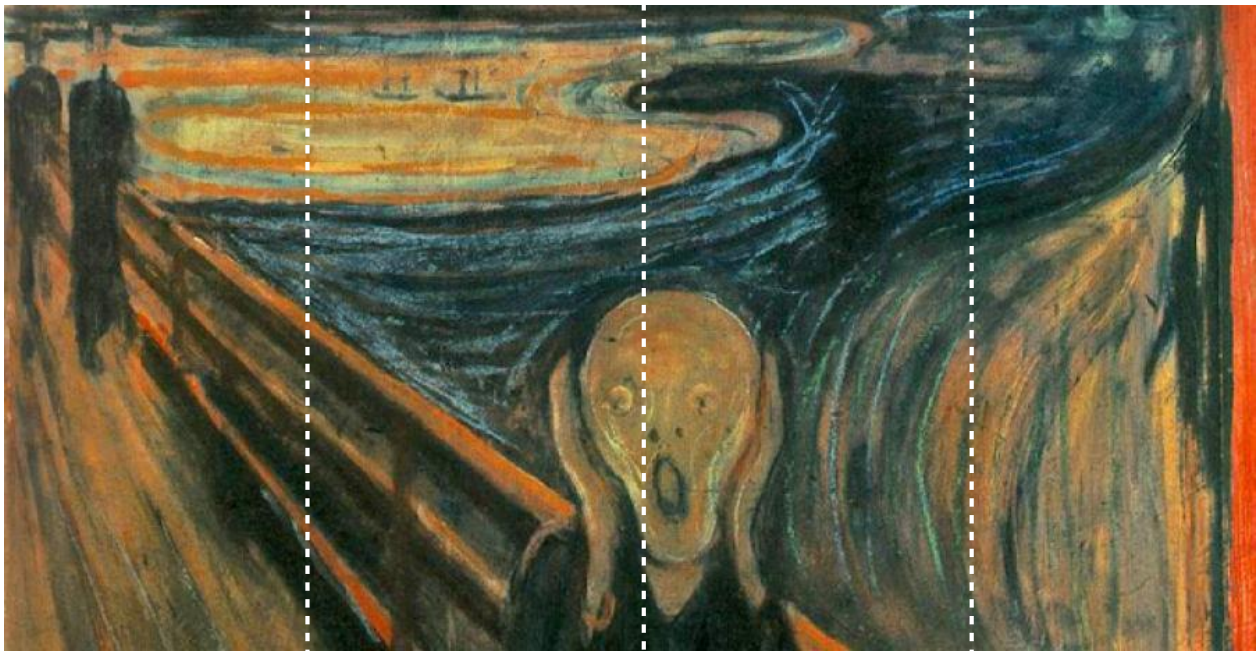
temps total : 1 jour ???

Exemple : reproduction d'une peinture



Comment paralléliser ?

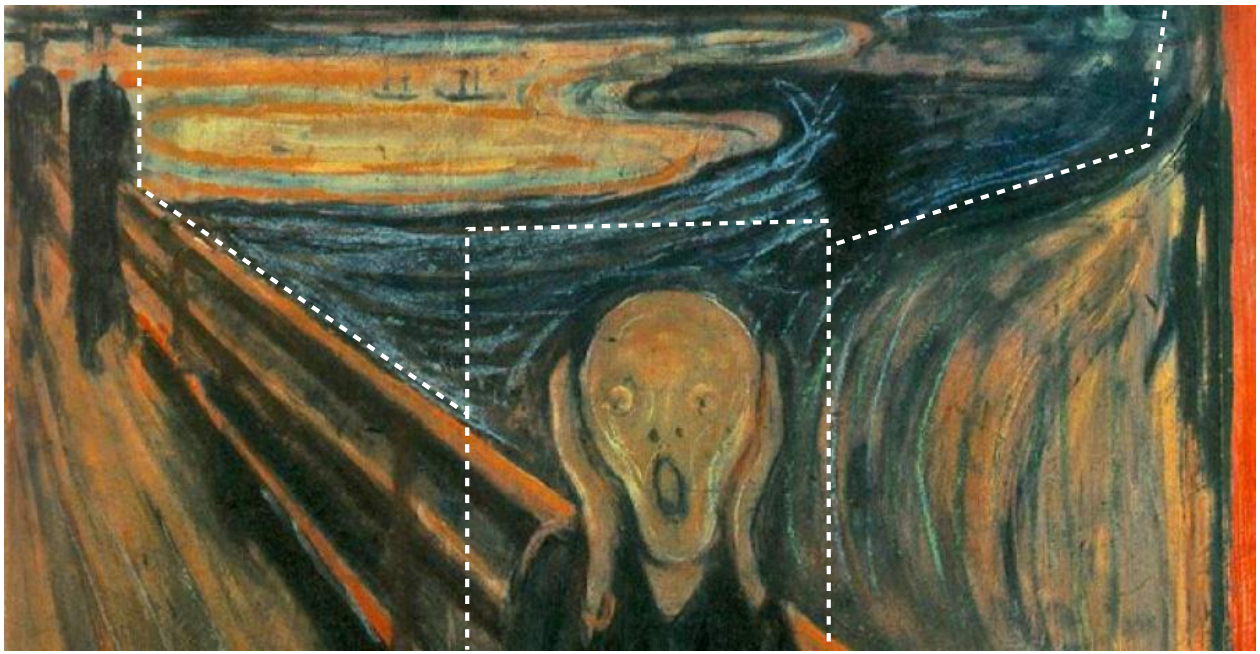
Exemple : reproduction d'une peinture



Comment paralléliser ?

- Découpage spatial ?

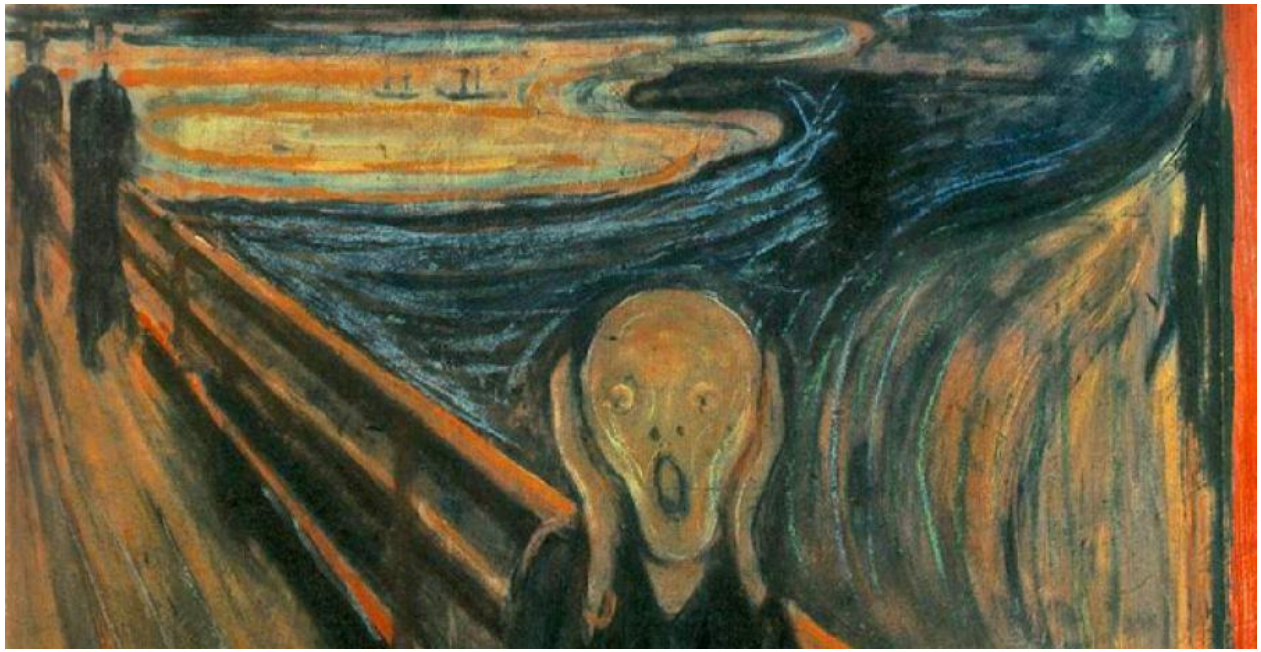
Exemple : reproduction d'une peinture



Comment paralléliser ?

- Découpage spatial ?
- Découpage irrégulier ?

Exemple : reproduction d'une peinture



Comment paralléliser ?

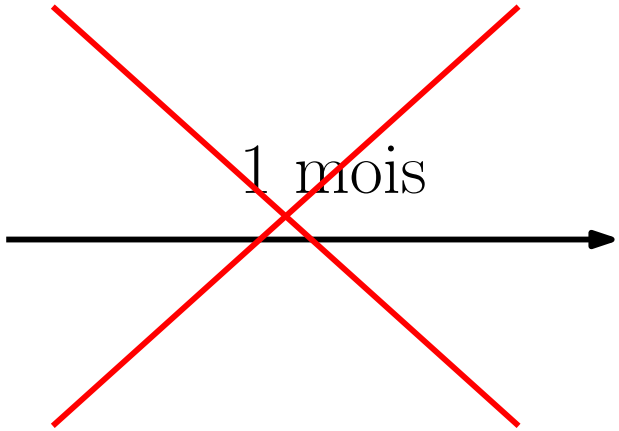
- Découpage spatial ?
- Découpage irrégulier ?
- Chacun sa couleur ?

Toutes les tâches sont-elles parallélisables ?

Toutes les tâches sont-elles parallélisables ?

9 mois

Toutes les tâches sont-elles parallélisables ?



Toutes les tâches sont-elles parallélisables ?

9 mois

Efficacité et accélération

- Soit $T(N)$ le temps de calcul d'une tâche avec N U.C.
- Parallélisation **parfaite** de cette tâche :

$$T(N) = \frac{T(1)}{N} \quad \Rightarrow \frac{T(1)}{T(N)} = N$$

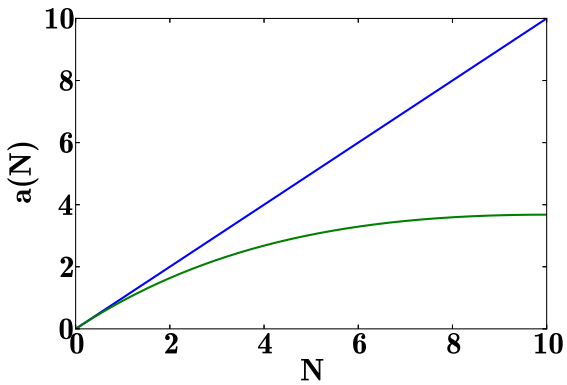
Efficacité et accélération

- Soit $T(N)$ le temps de calcul d'une tâche avec N U.C.
- Parallélisation **parfaite** de cette tâche :

$$T(N) = \frac{T(1)}{N} \quad \Rightarrow \frac{T(1)}{T(N)} = N$$

- On définit l'**accélération** (speed-up)

$$a(N) = \frac{T(1)}{T(N)}$$



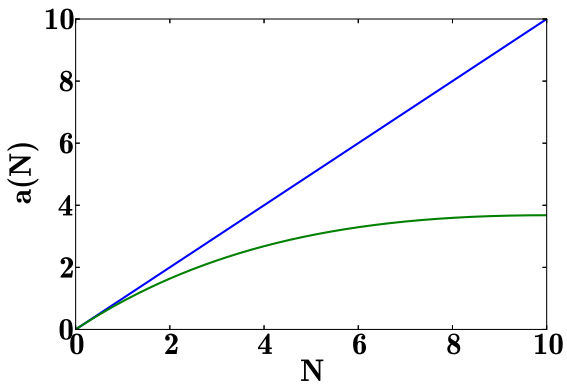
Efficacité et accélération

- Soit $T(N)$ le temps de calcul d'une tâche avec N U.C.
- Parallélisation **parfaite** de cette tâche :

$$T(N) = \frac{T(1)}{N} \quad \Rightarrow \frac{T(1)}{T(N)} = N$$

- On définit l'**accélération** (speed-up)

$$a(N) = \frac{T(1)}{T(N)}$$



Efficacité maximale

- Minimiser la partie séquentielle du calcul
- Minimiser les communications

Bibliothèques d'échanges de messages

- Début des années 80 :
  - Argonne P4
  - PICL
  - PVL
  - LAM
  - PCGMSG (chimie quantique)
  - ...

Bibliothèques conçues pour une architecture particulière !

Bibliothèques d'échanges de messages

- Début des années 80 :
  - Argonne P4
  - PICL
  - PVL
  - LAM
  - PCGMSG (chimie quantique)
  - ...

Bibliothèques conçues pour une architecture particulière !

- Conférence Supercomputing 1992 : MPI

Message Passing Interface

- Multi-plateforme, multi-OS
- Bibliothèque "standard" aujourd'hui

Identification de chacun des travailleurs

Hippolyte
Perpignan
07 88 54 67 92

Ursule
Toulon
06 22 31 58 41

Melchior
Reims
06 94 55 67 28

Gédéon
Lille
07 41 89 58 62

Le monde de communication MPI

```
program bonjour
implicit none
! Contient des variables utiles a MPI
include 'mpif.h'

integer :: erreur, totalCPU, monCPU

! Initialisation du monde de communication
call MPI_INIT(erreur)
! Combien sommes-nous ?
call MPI_COMM_SIZE(MPI_COMM_WORLD, totalCPU, erreur)
! Qui suis-je ?
call MPI_COMM_RANK(MPI_COMM_WORLD, monCPU, erreur)

print*, 'Bonjour, je suis le CPU',monCPU,' sur ',totalCPU

! Finalisation des directives MPI
call MPI_FINALIZE(erreur)

end program bonjour
```

Un monde de communication MPI - C

```c
#include <stdio.h>
#include <MPI.h>

int main()
{
  int erreur, totalCPU, monCPU;

  // Initialisation de MPI
  erreur = MPI_Init();
  // Combien sommes-nous ?
  erreur = MPI_Comm_size(MPI_COMM_WORLD, &totalCPU);
  // Qui Suis-je ?
  erreur = MPI_Comm_rank(MPI_COMM_WORLD, &monCPU);

  printf("Bonjour, je suis le CPU %d sur %d\n",monCPU, totalCPU);

  // Finalisation des directives MPI
  erreur = MPI_Finalize();

  return 0;
}
```

Compilation/Exécution

- On utilise le compilateur livré avec MPI

```
bastien@uranus:~/Codes$ mpif90 bonjour.F90 -o bonjour
```

Le programme peut fonctionner sur un nombre **arbitraire** d'unités de calculs ! $\Rightarrow$ On ne spécifie pas le nombre, ni dans le code ni à la compilation !

- On exécute le programme avec mpirun

```
bastien@uranus:~/Codes$ mpirun -n 8 ./bonjour
Bonjour, je suis le CPU           0  sur           8
Bonjour, je suis le CPU           1  sur           8
Bonjour, je suis le CPU           3  sur           8
Bonjour, je suis le CPU           7  sur           8
Bonjour, je suis le CPU           4  sur           8
Bonjour, je suis le CPU           5  sur           8
Bonjour, je suis le CPU           6  sur           8
Bonjour, je suis le CPU           2  sur           8
```

option -n : nombre de CPU

Pourquoi communiquer ?

Découpage de la tâche entre 4 peintres

Pourquoi communiquer ?

Découpage de la tâche entre 4 peintres
Les couleurs et formes doivent correspondre !

Comment communiquer ?

Messages

La communication des informations entres les unités de calculs se font au travers de messages **explicites** :

Comment communiquer ?

Messages

La communication des informations entres les unités de calculs se font au travers de messages **explicites** :

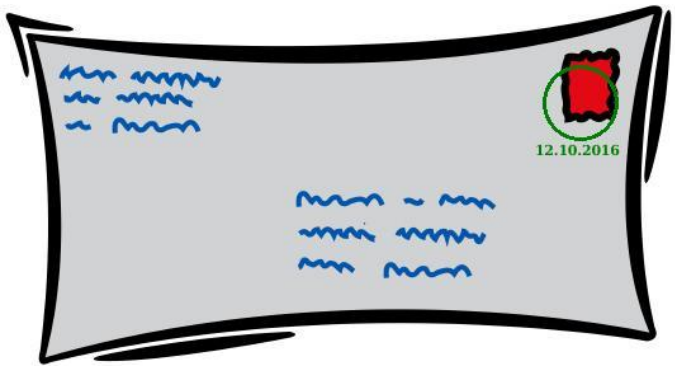
- Source/Destination
- Taille du message
- Identificateur
- Contenu

Comment communiquer ?

Messages

La communication des informations entres les unités de calculs se font au travers de messages **explicites** :

- Source/Destination
- Taille du message
- Identificateur
- Contenu



Comment communiquer ?

- Envoi d'un message :

```
call MPI_SEND(message, taille_mess, type_mess, destination,
              identificateur,MPI_COMM_WORLD, erreur)
```

- Réception d'un message

```
call MPI_RECV(message, taille_mess, type_mess, source,
              identificateur,MPI_COMM_WORLD, statut, erreur)
```

- Données :
  - message : pointeur sur le message lui-même
  - taille_mess : nombre de cases mémoire du message (pour un tableau ! Cas d'un scalaire : 1)
  - type_mess : type des données à envoyer (entier, réel, ...)
  - source/destination : numérotation MPI (entier)
  - identificateur : rend le message unique (entier)
  - statut/erreur : gérés par MPI (entier)

Exemple de communication - 4 unités de calculs

```
integer :: erreur, totalCPU, monCPU
integer :: statut(MPI_STATUS_SIZE)
real :: x

call MPI_INIT(erreur)
call MPI_COMM_SIZE(MPI_COMM_WORLD, totalCPU, erreur)
call MPI_COMM_RANK(MPI_COMM_WORLD, monCPU, erreur)

x = rand()

if(monCPU.eq.0) then ! envoi au CPU 3, identificateur 127
  call MPI_SEND(x, 1, MPI_REAL, 3, 127, MPI_COMM_WORLD, erreur)

else if(monCPU.eq.1) then ! envoi au CPU 2, identificateur 315
  call MPI_SEND(x, 1, MPI_REAL, 2, 315, MPI_COMM_WORLD, erreur)

else if(monCPU.eq.2) then ! recoit du CPU 1, identificateur 315
  call MPI_RECV(x, 1, MPI_REAL, 1, 315, MPI_COMM_WORLD, erreur)

else if(mpiCPU.eq.3) then ! recoit du CPU 0, identificateur 127
  call MPI_RECV(x, 1, MPI_REAL, 0, 217, MPI_COMM_WORLD, erreur)
endif
```

Communications

Vous pouvez tout faire avec MPI_SEND / MPI_RECV !

Communications

Vous pouvez tout faire avec MPI_SEND / MPI_RECV !

Fastidieux

- Envois multiples (section suivante)

Communications

Vous pouvez tout faire avec MPI_SEND / MPI_RECV !

Fastidieux

- Envois multiples (section suivante)
- Communications bloquantes : arrêt du code !

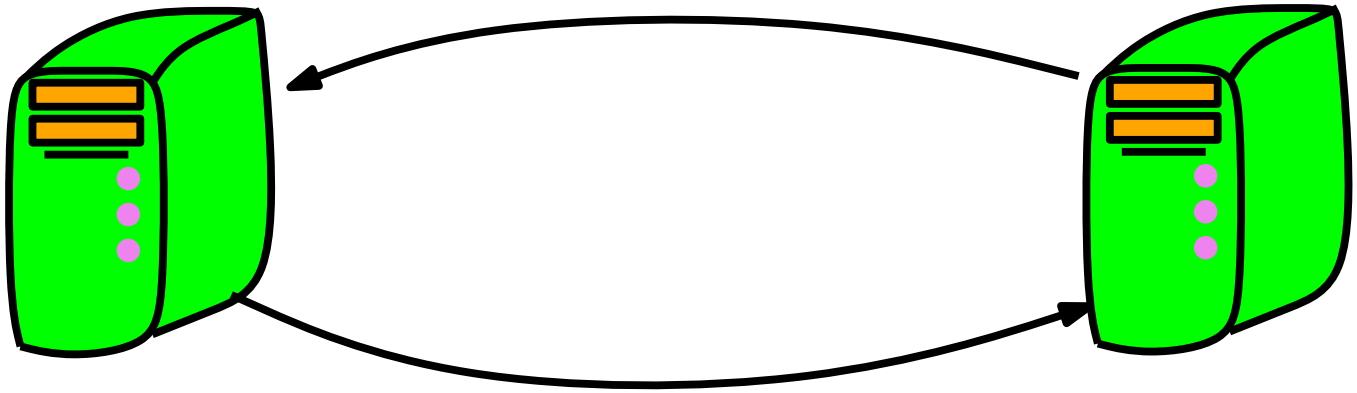
Communications

Vous pouvez tout faire avec MPI_SEND / MPI_RECV !

Fastidieux

- Envois multiples (section suivante)
- Communications bloquantes : arrêt du code !

Exemple : échange de données



Communications non-bloquantes (requêtes)

```
integer :: erreur, totalCPU, monCPU
integer :: autreCPU, send_req, recv_req
integer :: statut(MPI_STATUS_SIZE)
real :: x,y
call MPI_INIT(...
x = rand()

! selection de la source/destination
if(monCPU.eq.0) then
  autreCPU = 1
else
  autreCPU = 0
endif
! envoi de messages non bloquants
call MPI_ISEND(x, 1, MPI_REAL, autreCPU, 0, MPI_COMM_WORLD,
                                            send_req, erreur)
call MPI_IRECV(y, 1, MPI_REAL, autreCPU, 0, MPI_COMM_WORLD,
                                            recv_req, erreur)
! je peux faire autre chose pendant ce temps !
call MPI_WAIT(send_req, statut, erreur)
call MPI_WAIT(recv_req, statut, erreur)
```

Bouteille à la mer

- MPI_ANY_SOURCE :
  Réception d'un message provenant de n'importe quelle unité de calcul
- MPI_ANY_TAG :
  Réception d'un message avec n'importe quel identificateur

```
call MPI_RECV(message, taille, type_mess, statut,
                       MPI_ANY_SOURCE, MPI_ANY_TAG,
                       MPI_COMM_WORLD, erreur)
```

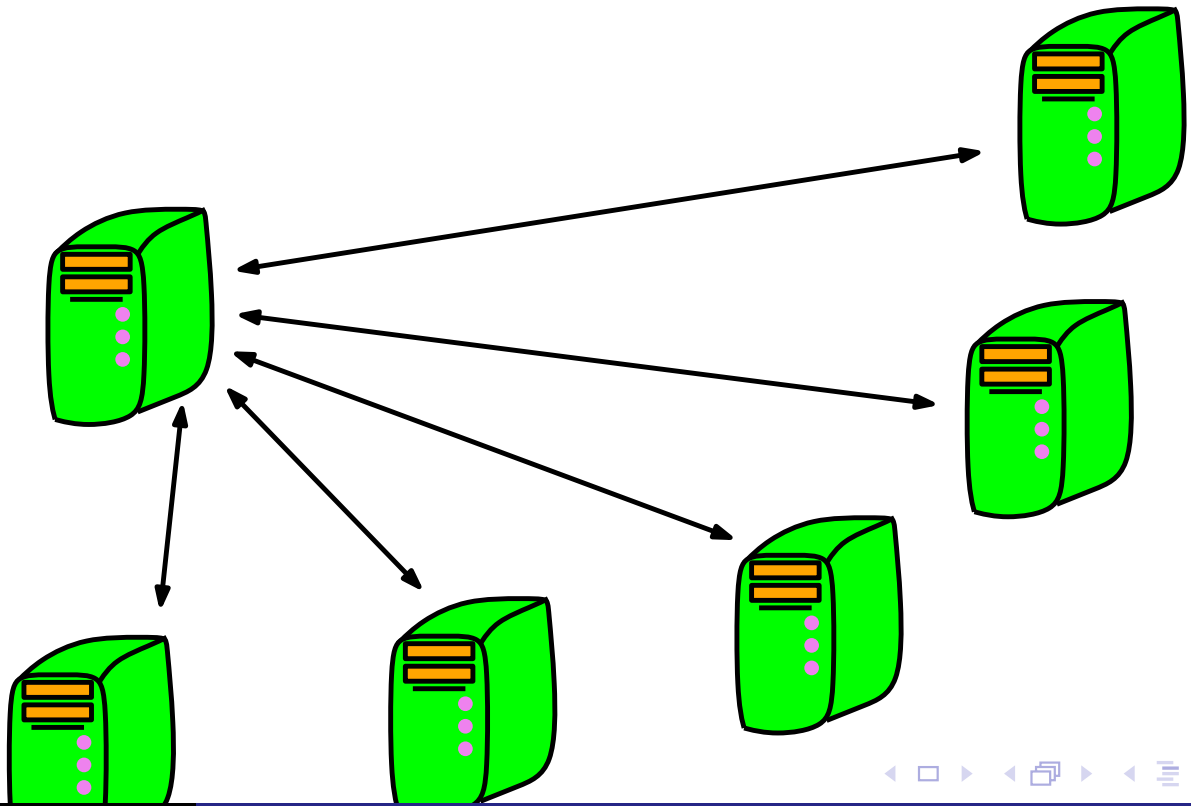
Attention

Les "messages à la mer" sont souvent sources d'erreurs !

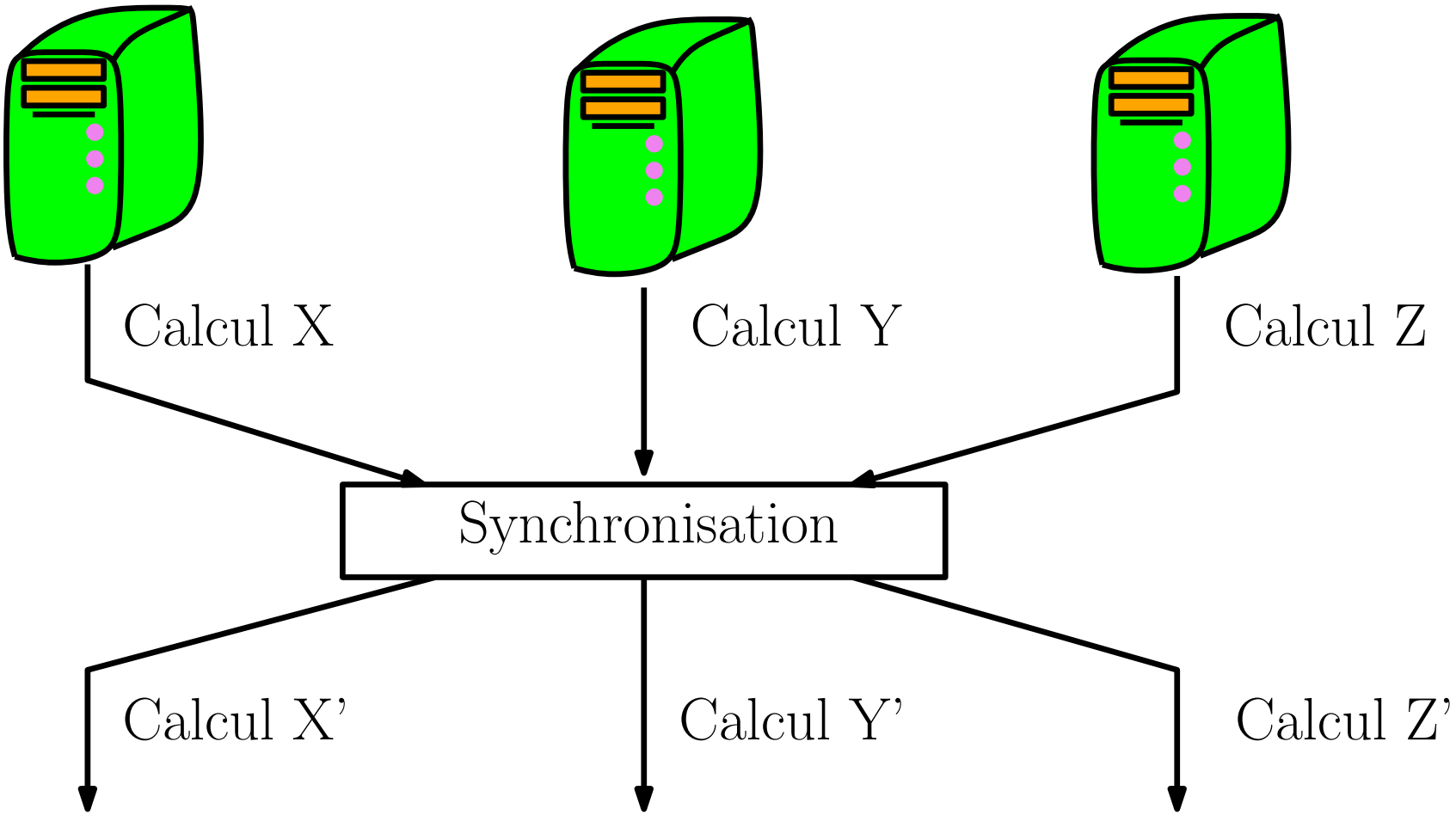
Communications globales

Communications globales (Communications collectives)

Une (ou plusieurs) unité(s) de calculs échange des données avec l'ensemble des autres unités de calculs.
$\Rightarrow$ Succession de SEND / RECV



Synchronisation



Synchronisation

- Syntaxe :

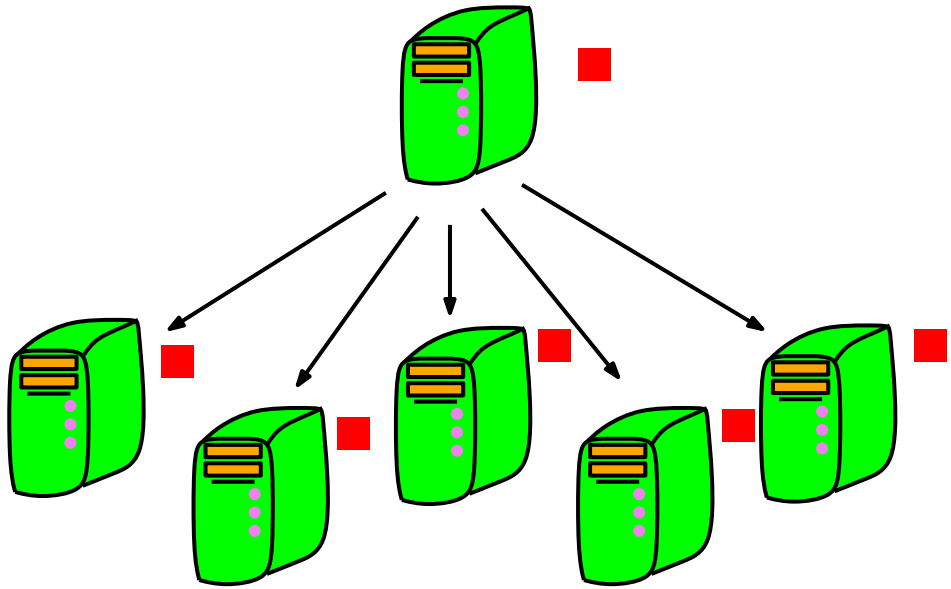
```
! On fait des calculs independants
x = ...

! Tout le monde doit s'attendre !
call MPI_BARRIER(MPI_COMM_WORLD, erreur)

! On reprend les calculs
y = ...
```

MPI_BARRIER

Les unités de calculs se mettent en attente **tant que** toutes les unités n'ont pas franchi cette barrière !

Broadcast

- Envoi d'une donnée d'une unité de calculs à toutes les autres



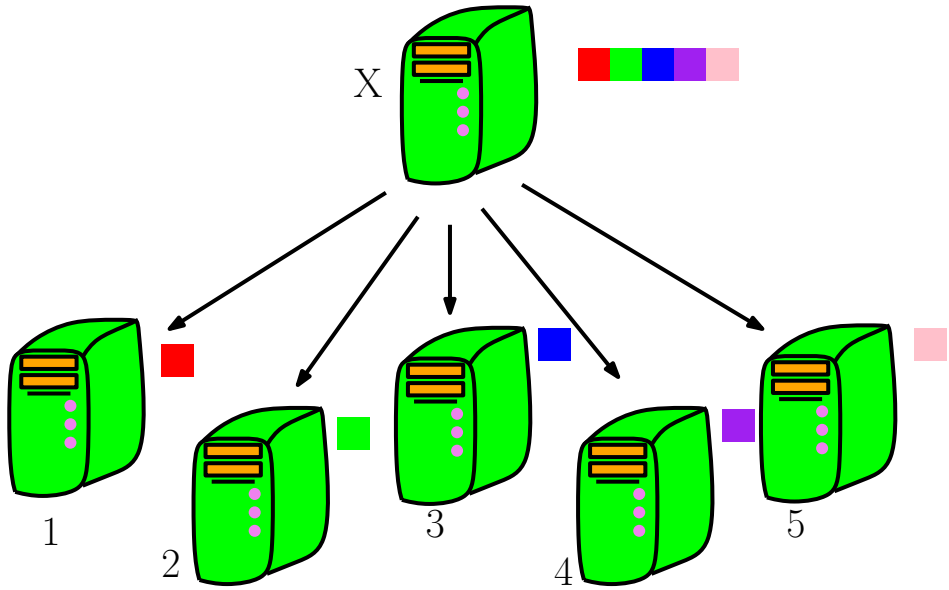
```
call MPI_BCAST(message, taille_mess, type_mess, source,
                                    MPI_COMM_WORLD, erreur)
```

Exemple : lecture fichier

```
call MPI_INIT(erreur)
call MPI_COMM_SIZE(MPI_COMM_WORLD, totalCPU, erreur)
call MPI_COMM_RANK(MPI_COMM_WORLD, monCPU, erreur)

if(monCPU.eq.0) then
  ! Ouverture du fichier
  open(unit = 21, file='data.txt')
  ! Lecture des donnees
  read(21, *) N
  close(21)
endif

call MPI_BCAST(N, 1, MPI_INT, 0, MPI_COMM_WORLD, erreur)
```

Scattering

- Envoi fractionné d'un tableau entre les différentes unités de calculs

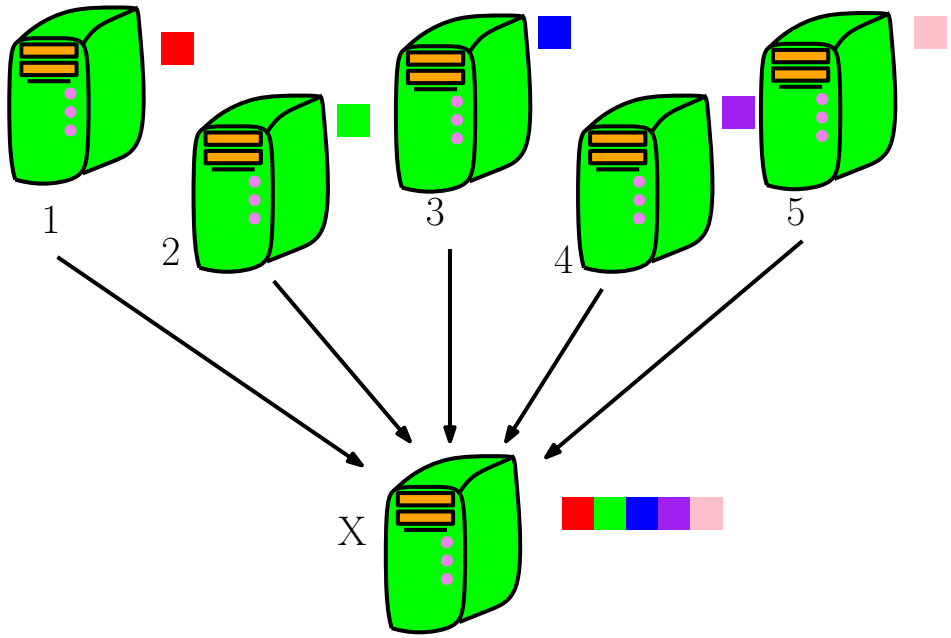



```
call MPI_SCATTER(tab_envoi, taille, type_mess,
                 tab_recep, nb_case, type_mess,
                 source, MPI_COMM_WORLD, erreur)
```

taille (in) : nb de cases à envoyer à chaque unité de calculs
nb_case (out) : nb de cases **effectivement** reçues (division non exacte)

Gathering

- Réception fractionnée d'un tableau depuis les différentes unités de calculs



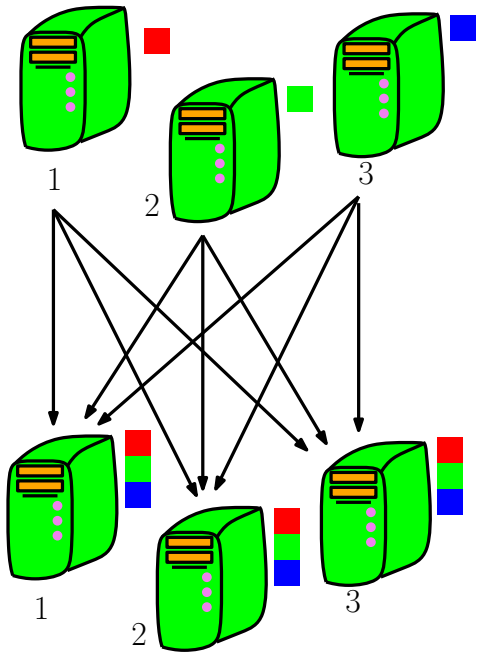

```
call MPI_GATHER(tab_envoi, taille, type_mess,
                tab_recep, nb_case, type_mess,
                source, MPI_COMM_WORLD, erreur)
```

taille (in) : nb de cases à recevoir depuis chaque unité de calculs

nb_case (out) : nb de cases **effectivement** reçues

All-Gathering

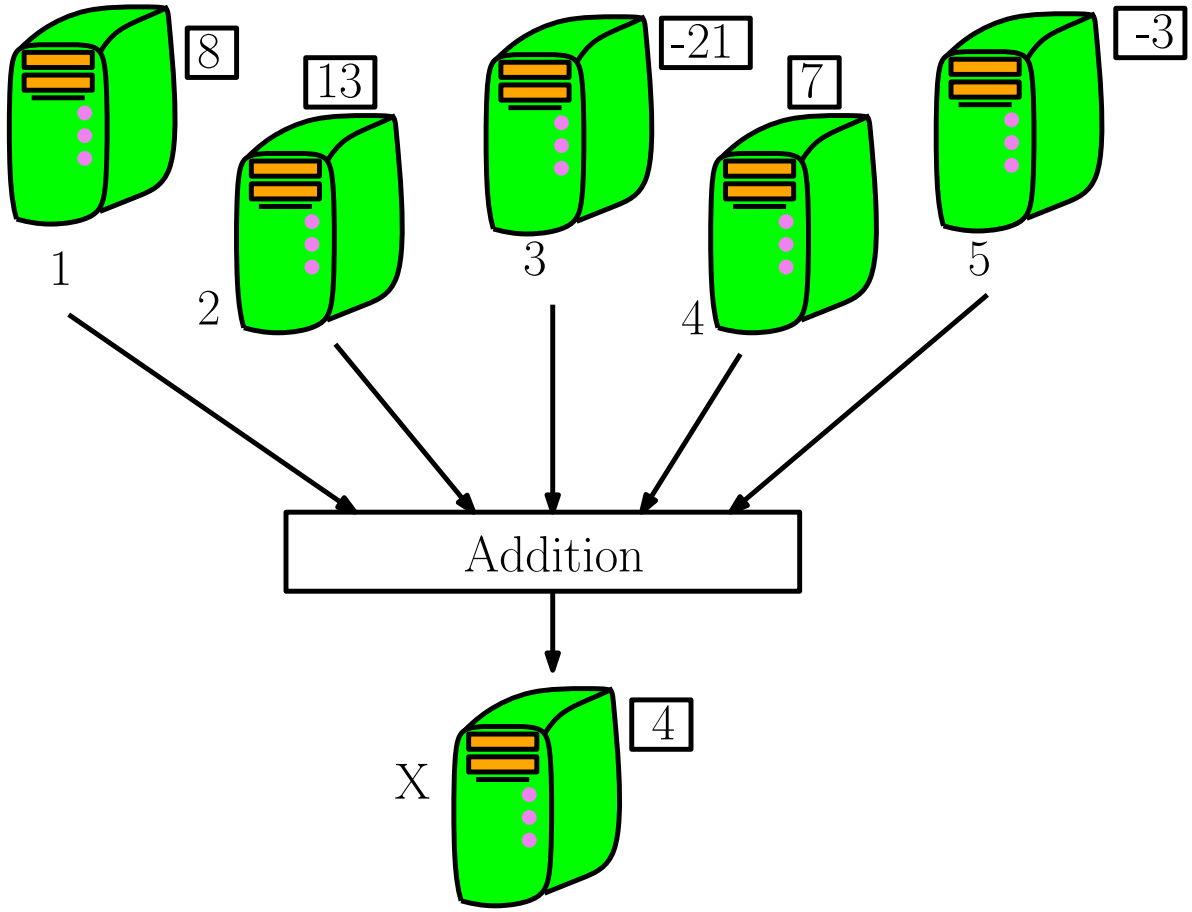
- Reception fractionné d'un tableau depuis les différentes unités de calculs




```
call MPI_ALLGATHER(tab_envoi, taille, type_mess,
                   tab_recep, nb_case, type_mess,
                   MPI_COMM_WORLD, erreur)
```

Reduce

- Réception des données depuis toutes les unités de calculs + Opération arithmétique !



Reduce

```
call MPI_REDUCE(donne_envoi, donne_recep, taille, type_donne,
                Operation, source, MPI_COMM_WORLD, erreur)
```

Opération :

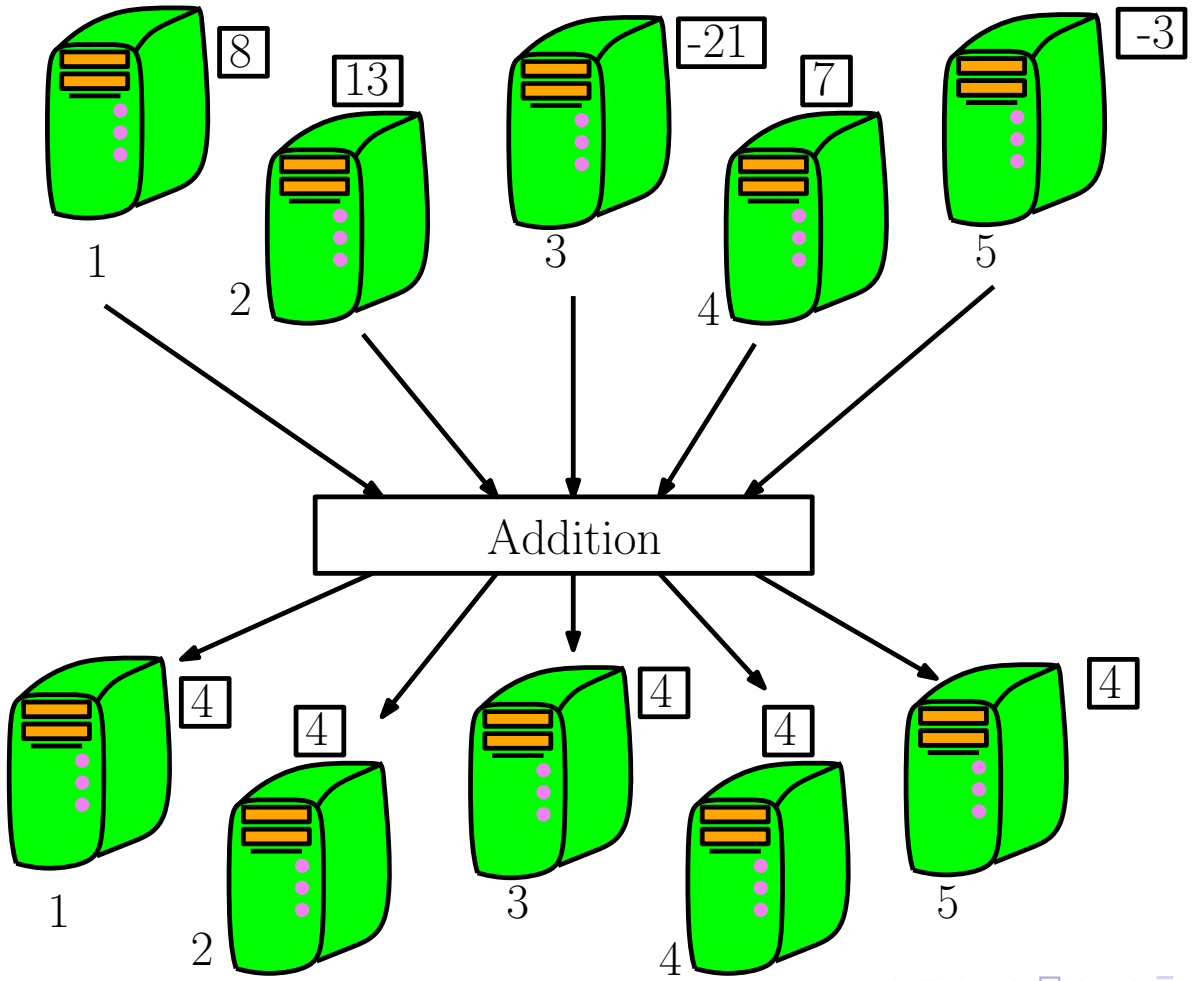
- Maximum, minimum : MPI_MAX, MPI_MIN
- Position (unité de calculs) : MPI_MAXLOC, MPI_MINLOC
- Somme, Produit : MPI_SUM, MPI_PROD
- Opération logiques ...

Cas des tableaux

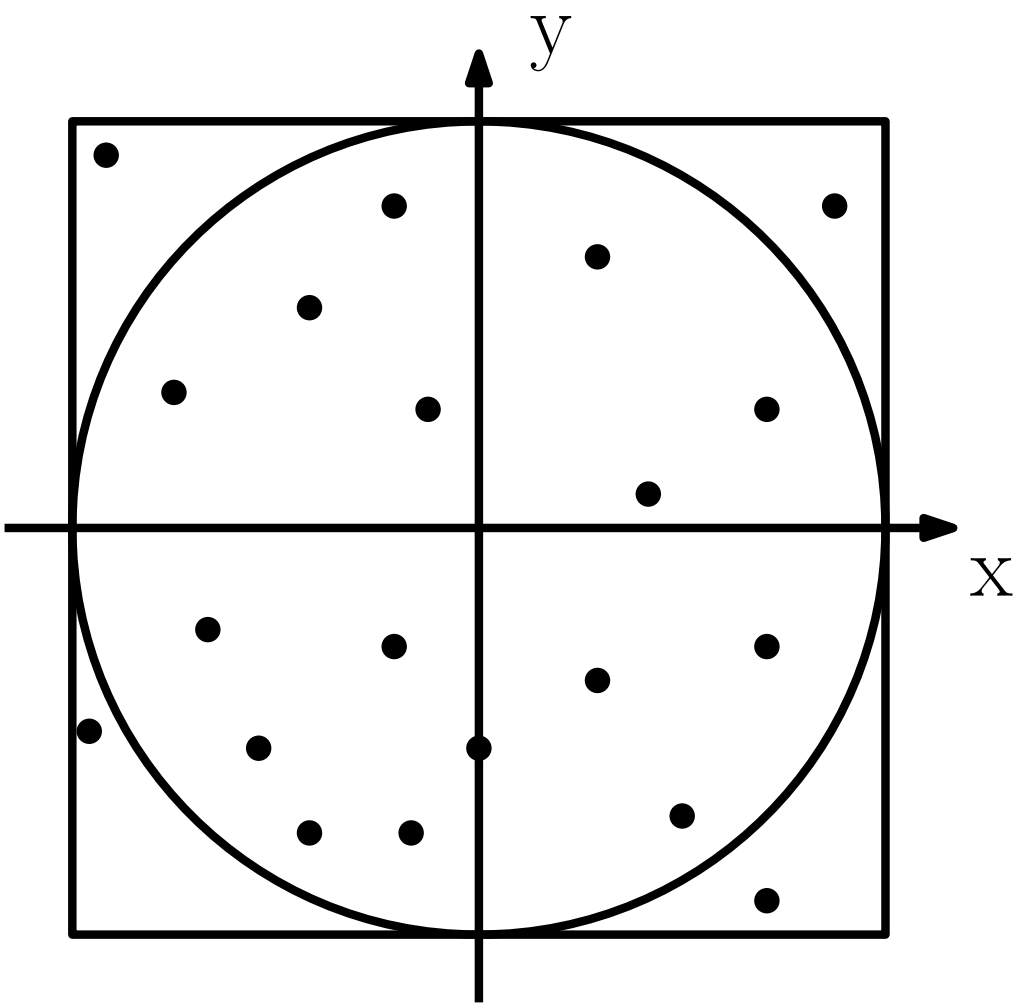
Les opérations s'effectuent cases par cases !

All-Reduce

```
call MPI_ALLREDUCE(donne_envoi, donne_recep, taille, type_donne,
                   Operation, MPI_COMM_WORLD, erreur)
```



Calcul de π - Méthode Monte-Carlo



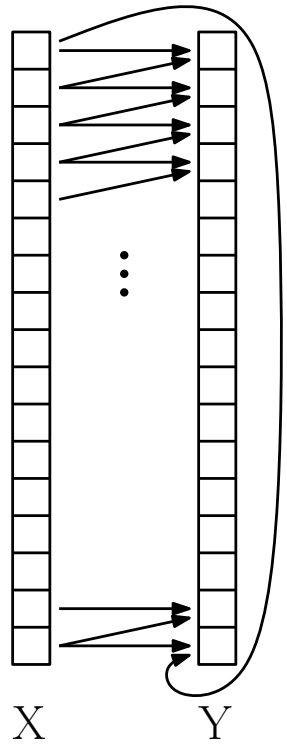
N lancés aléatoires

$$\pi = 4\frac{S_{cercle}}{S_{carre}}$$

$$\pi \approx 4\frac{N_{cercle}}{N_{total}}$$

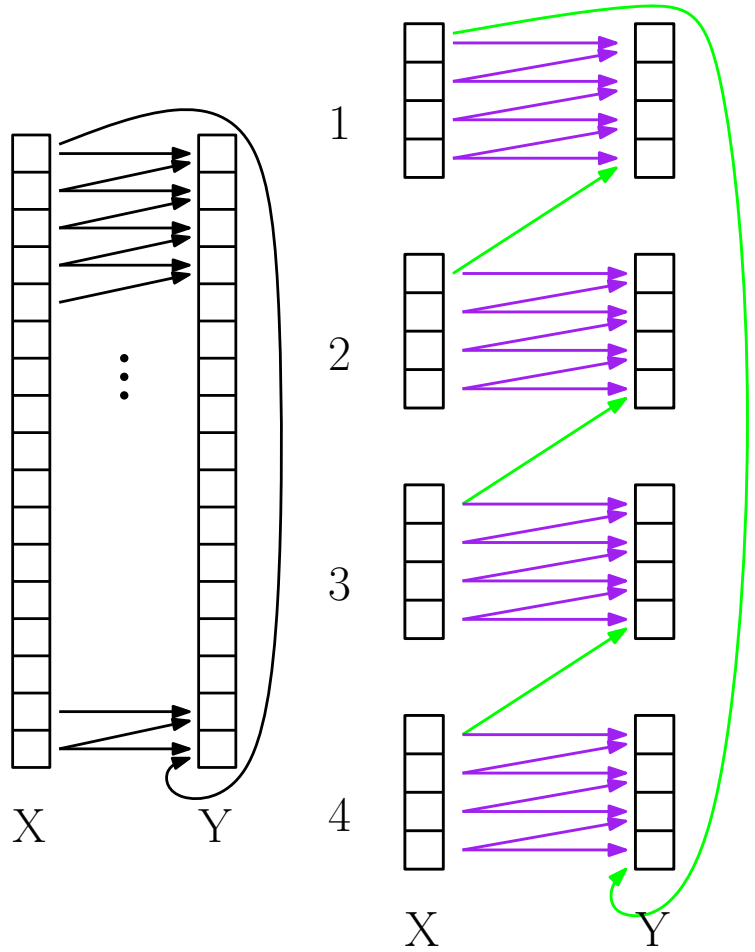
Algorithme sur des tableaux

- Algorithme :
  - Tirs aléatoires :
    Pour i de 0 a N-1
    X(i) = rand()
  - Calculs :
    Pour i de 0 a N-1
    Y(i) = X(i).X(i+1)/16

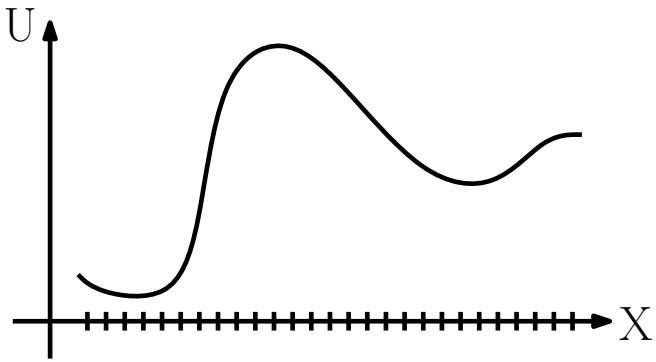


Algorithme sur des tableaux

- Algorithme :
  - Tirs aléatoires :
    Pour i de 0 a N-1
    X(i) = rand()
  - Calculs :
    Pour i de 0 a N-1
    Y(i) = X(i).X(i+1)/16
- Parallélisation
  - Découpage : $N_p = N/N_{CPU}$
  - Tirs aléatoires
  - Échange des données
    $X_{i+1}(0)$ envoyé au CPU i
  - Calcul de Y_i



Équation de diffusion



- Equation de diffusion :

$$\frac{\partial u}{\partial t^2} = D\frac{\partial^2 u}{\partial x^2}, \qquad x \in]-L; L[$$

$$u(-L) = u(L)$$

- Discrétisation :

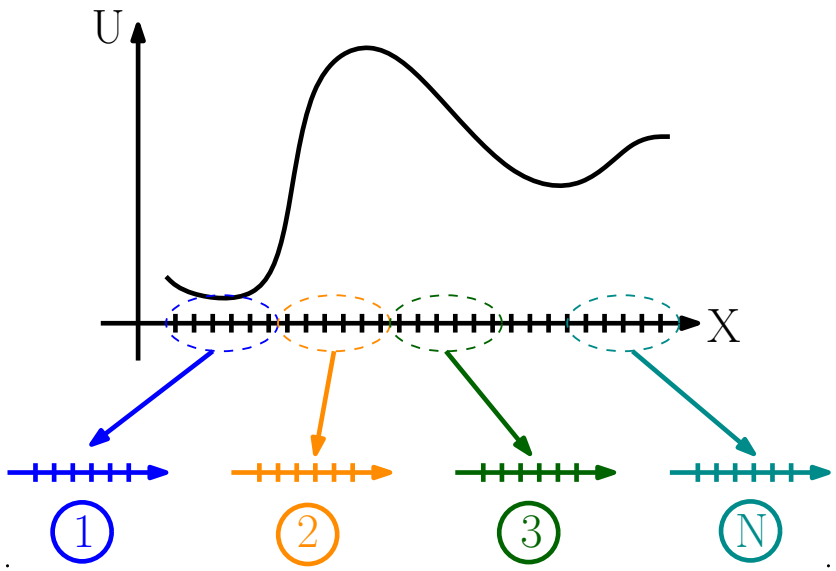
$$t^n = n\Delta t, \qquad x_i = i\Delta x$$

$$u(x_i, t^n) \approx u_i^n$$

- Différences finies centrées, Euler explicite :

$$\frac{\partial^2 u_i^n}{\partial x^2} = \frac{u_{i+1}^n - 2u_i^n + u_{i-1}^n}{\Delta x^2} \qquad u_i^{n+1} = u_i^n + \Delta t.D\frac{\partial^2 u_i^n}{\partial x^2}$$

Équation de diffusion



- Equation de diffusion :

$$\frac{\partial u}{\partial t^2} = D\frac{\partial^2 u}{\partial x^2}, \qquad x \in]-L; L[$$

$$u(-L) = u(L)$$

- Discrétisation :

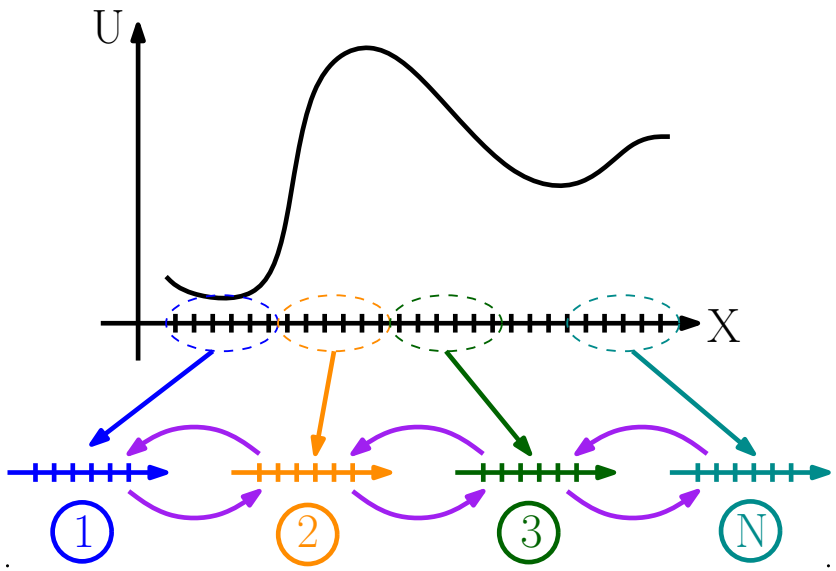
$$t^n = n\Delta t, \qquad x_i = i\Delta x$$

$$u(x_i, t^n) \approx u_i^n$$

- Différences finies centrées, Euler explicite :

$$\frac{\partial^2 u_i^n}{\partial x^2} = \frac{u_{i+1}^n - 2u_i^n + u_{i-1}^n}{\Delta x^2} \qquad u_i^{n+1} = u_i^n + \Delta t.D\frac{\partial^2 u_i^n}{\partial x^2}$$

Équation de diffusion



- Equation de diffusion :

$$\frac{\partial u}{\partial t^2} = D\frac{\partial^2 u}{\partial x^2}, \qquad x \in]-L; L[$$

$$u(-L) = u(L)$$

- Discrétisation :

$$t^n = n\Delta t, \qquad x_i = i\Delta x$$

$$u(x_i, t^n) \approx u_i^n$$

- Différences finies centrées, Euler explicite :

$$\frac{\partial^2 u_i^n}{\partial x^2} = \frac{u_{i+1}^n - 2u_i^n + u_{i-1}^n}{\Delta x^2} \qquad u_i^{n+1} = u_i^n + \Delta t.D\frac{\partial^2 u_i^n}{\partial x^2}$$