Cours 4 : (Un peu plus) d'algorithmique parallèle

Charles Bouillaguet
(charles.bouillaguet@univ-lille.fr)

2020-02-7

Résolution approchée d'EDP

Exemple : équation de la chaleur

$$\frac{\partial T}{\partial t} = \alpha \nabla^2 T = \alpha \left(\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} + \frac{\partial^2 T}{\partial z^2} \right)$$

- Diffusion de la chaleur dans un matériaux homogène.
- $T(x, y, z, t) =$ température au point (x, y, z) au temps t

Objectif :

- Calculer $T(x, y, z, t)$
- Sur un domaine fini
- $T(x, y, z, 0)$ connu (conditions initiales)
- Conditions aux limites éventuelles ($T(0, y, z, t) = cst$)

Résolution approchée d'EDP

Méthode d'Euler

$$\frac{\partial T}{\partial t} = \alpha \nabla^2 T = \alpha \left(\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} + \frac{\partial^2 T}{\partial z^2} \right)$$

Approximation

- Petit pas de temps

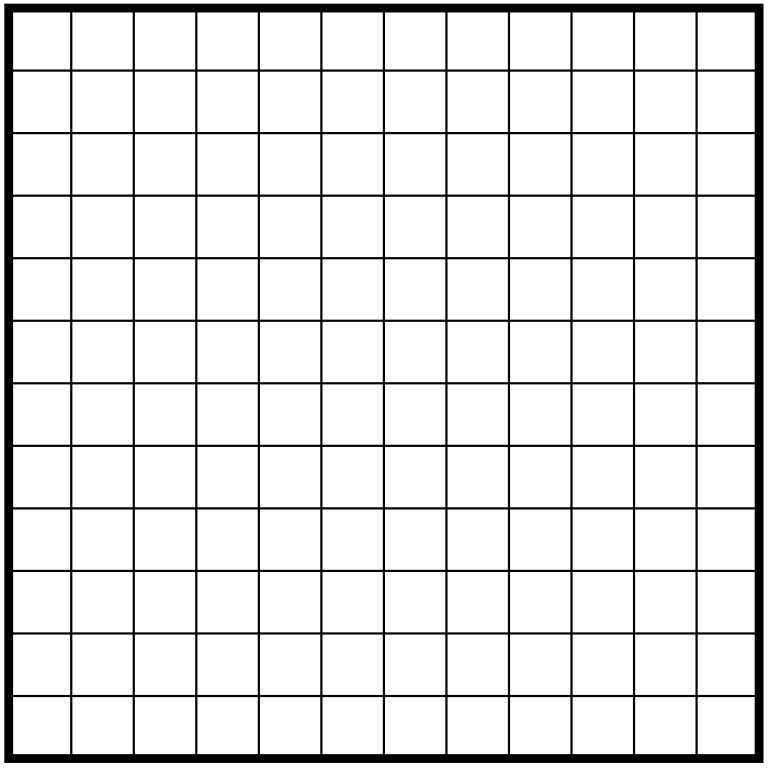
$$\frac{\partial T}{\partial t} \approx \frac{T(x, y, t + \Delta t) - T(x, y, t)}{\Delta t}$$

- Petites cellules dans l'espace

$$\frac{\partial^2 T}{\partial x^2} \approx \frac{T(x - \Delta x, y, t) + 2T(x, y, t) + T(x + \Delta x, y, t)}{\Delta x^2}$$

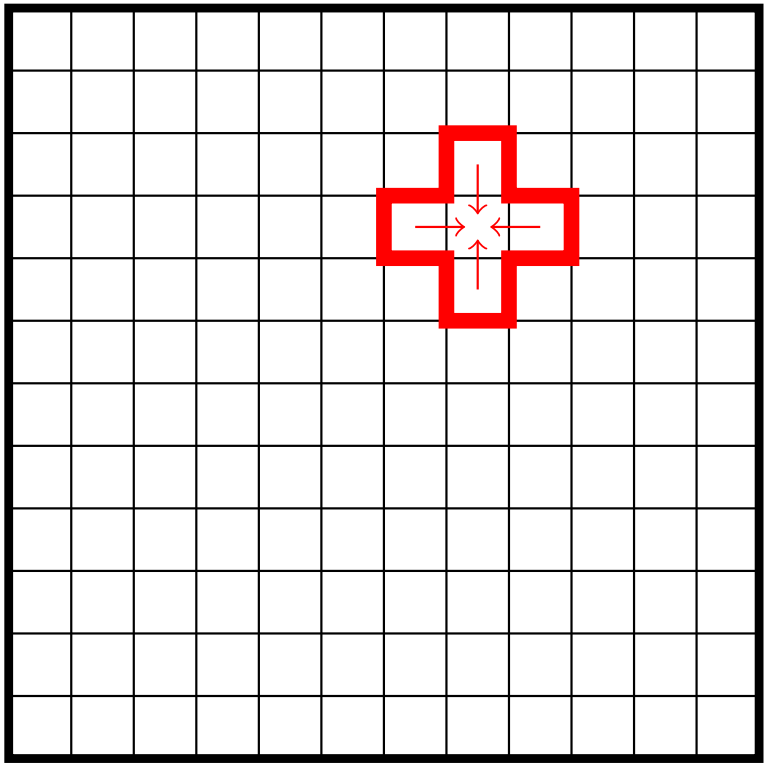
Résolution approchée d'EDP

Méthode d'Euler



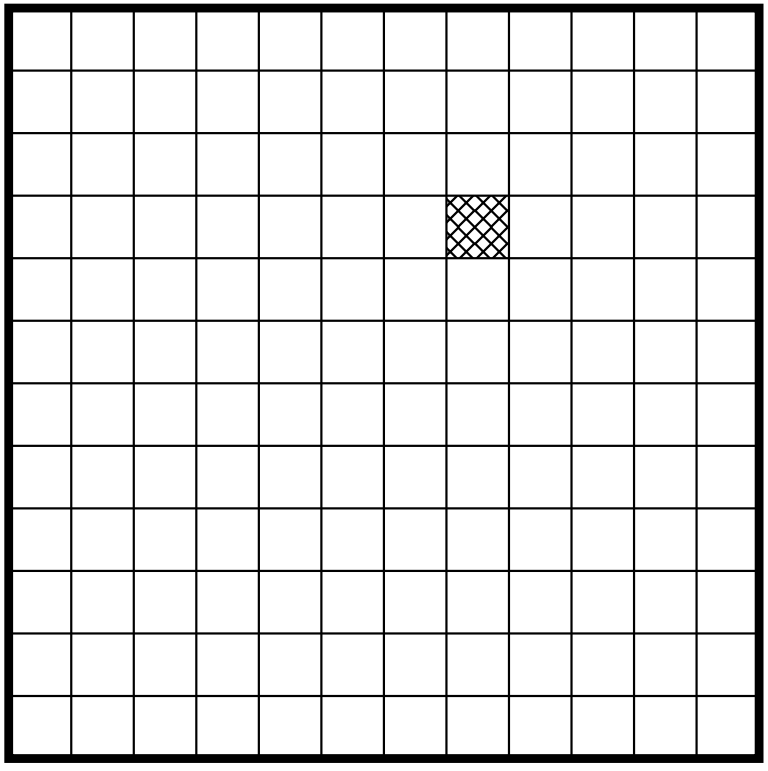
Résolution approchée d'EDP

Méthode d'Euler

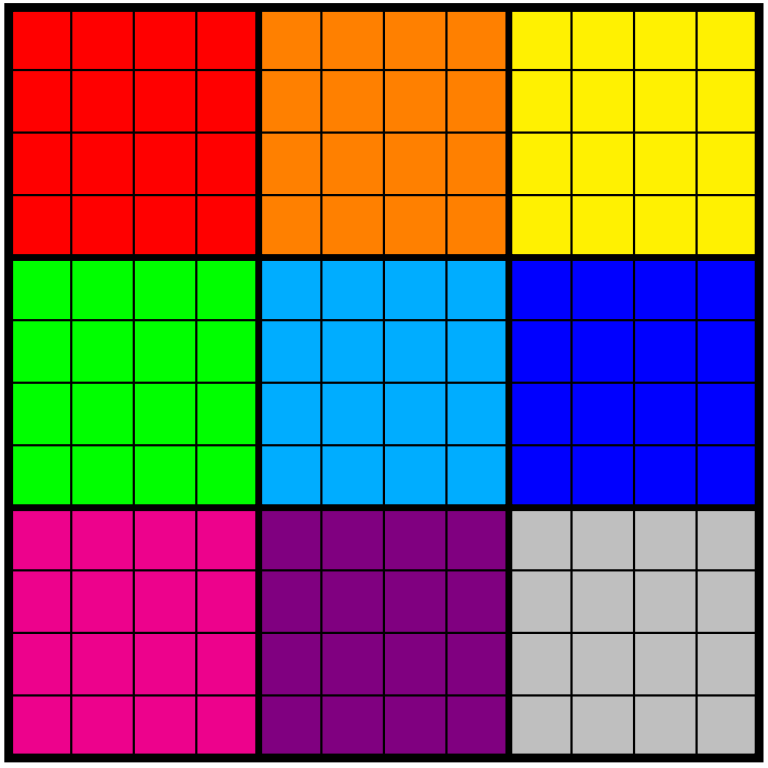


Résolution approchée d'EDP

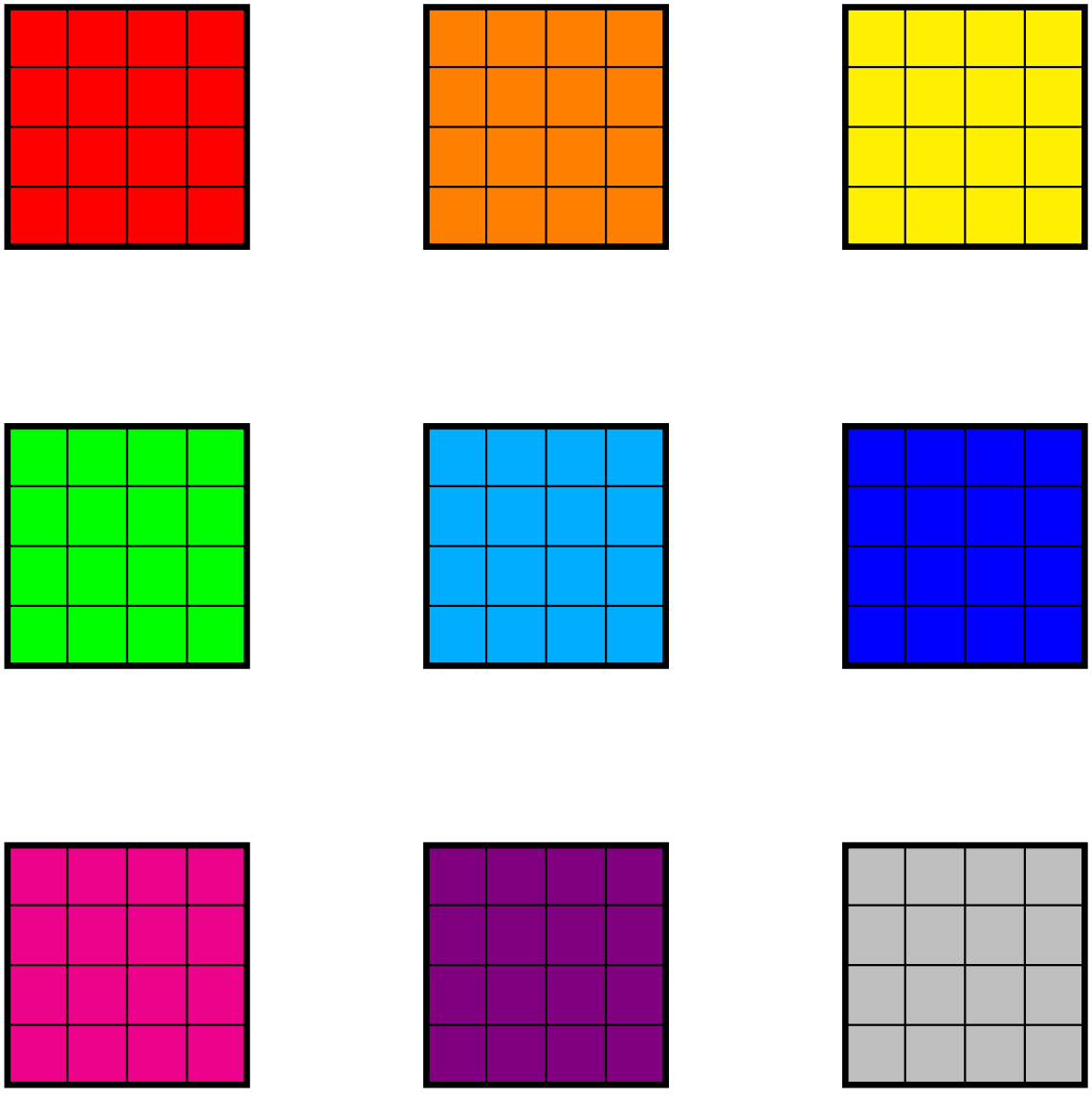
Méthode d'Euler



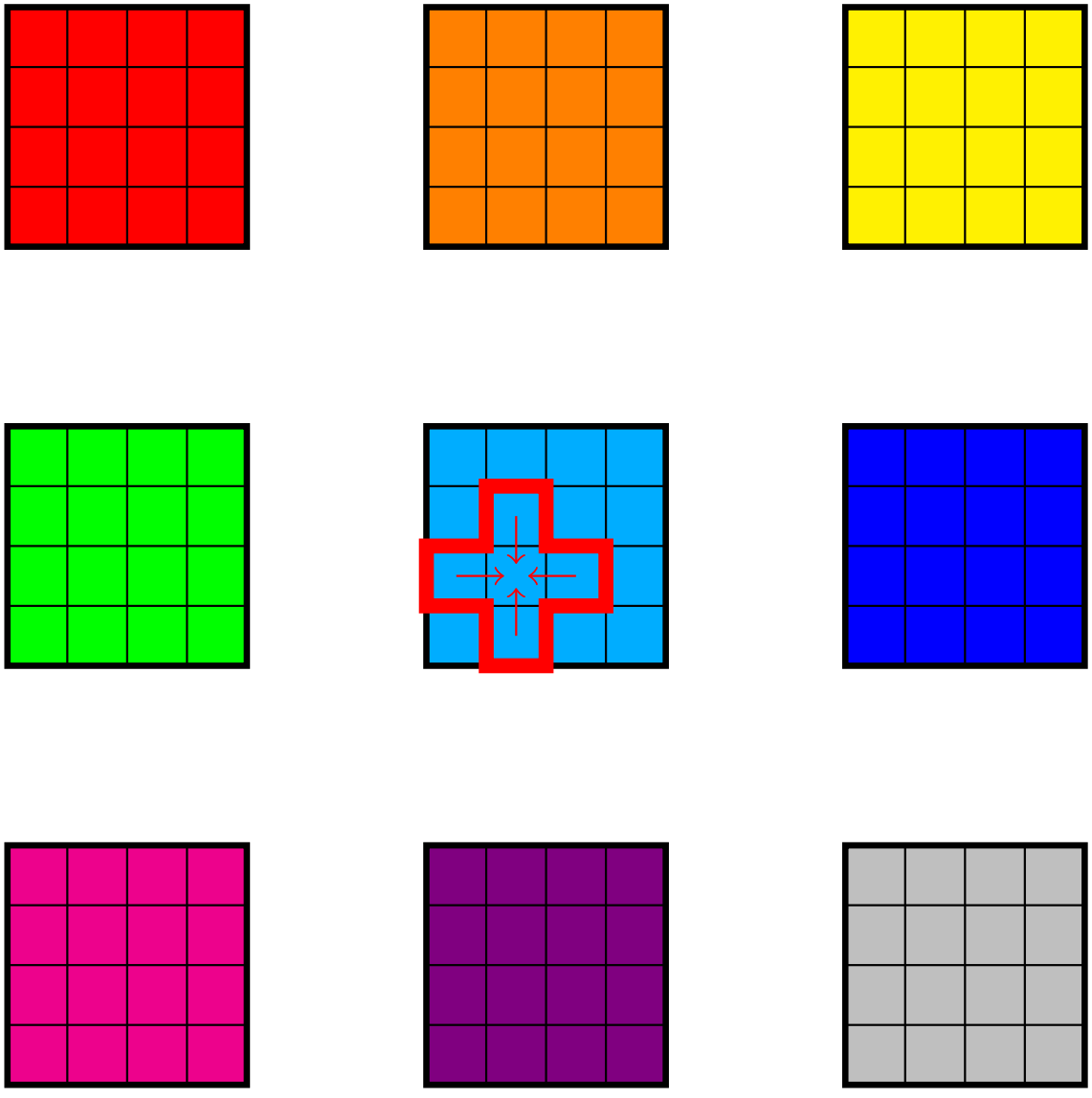
Décomposition de domaine



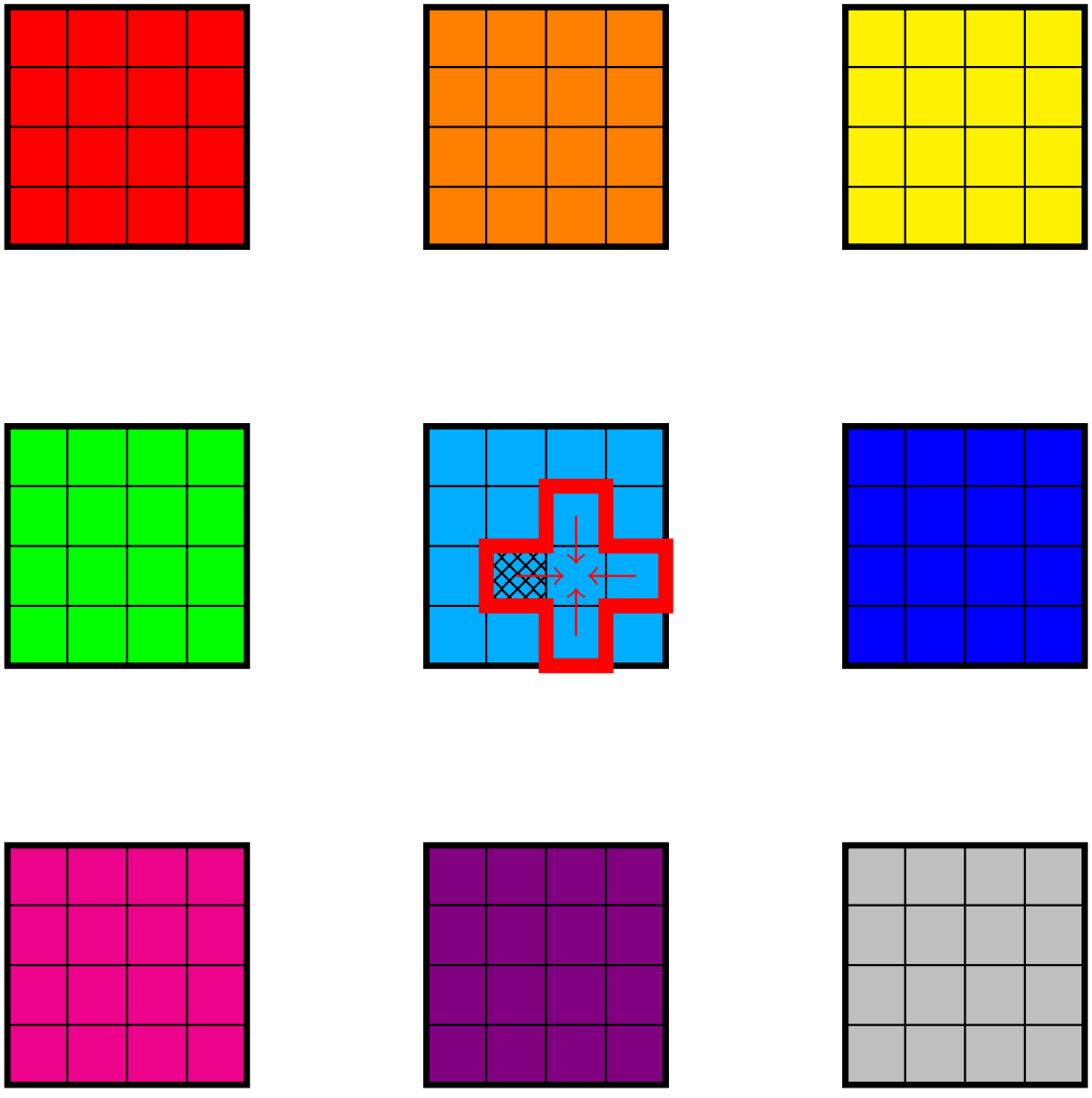
Décomposition de domaine



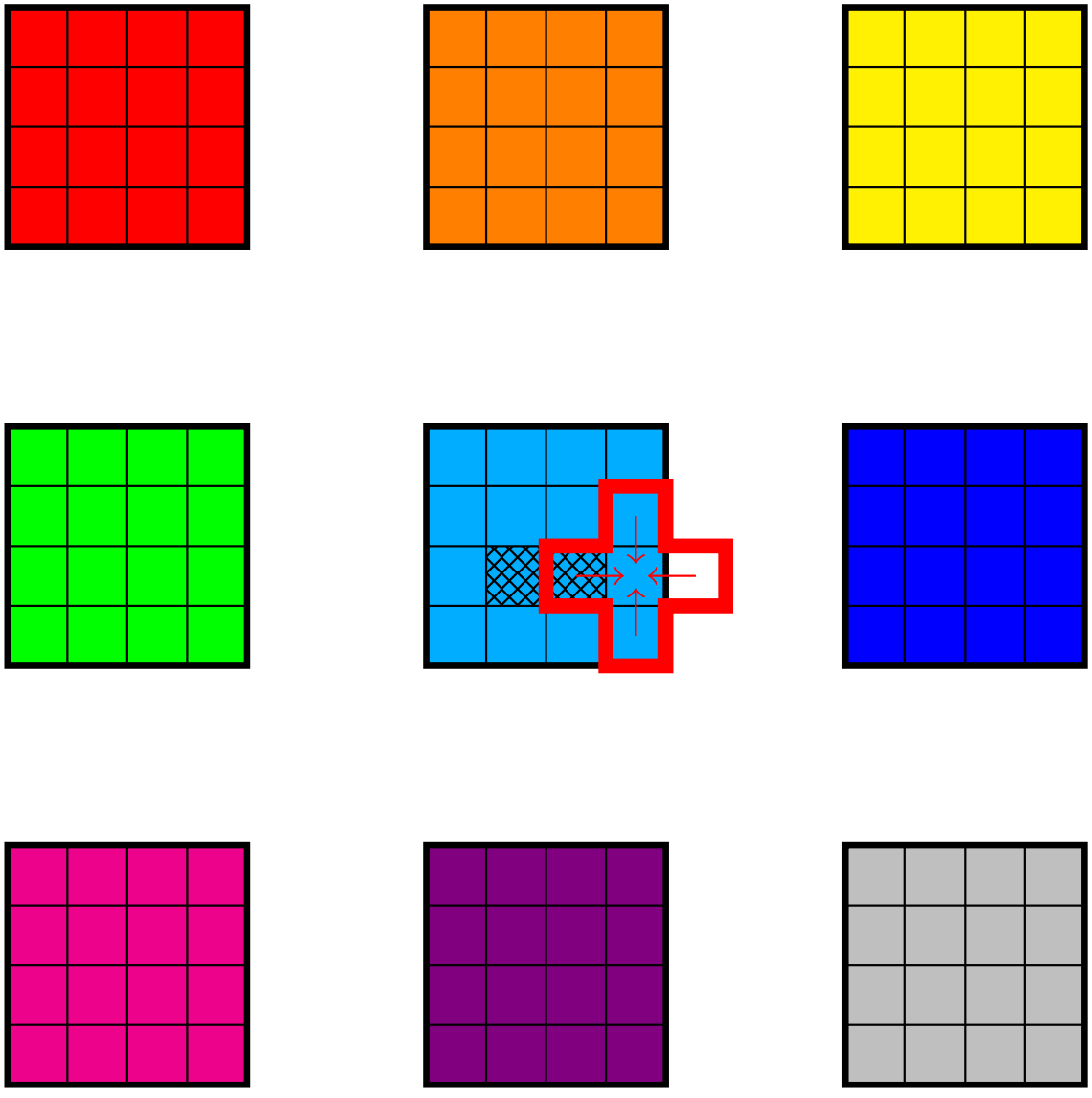
Décomposition de domaine



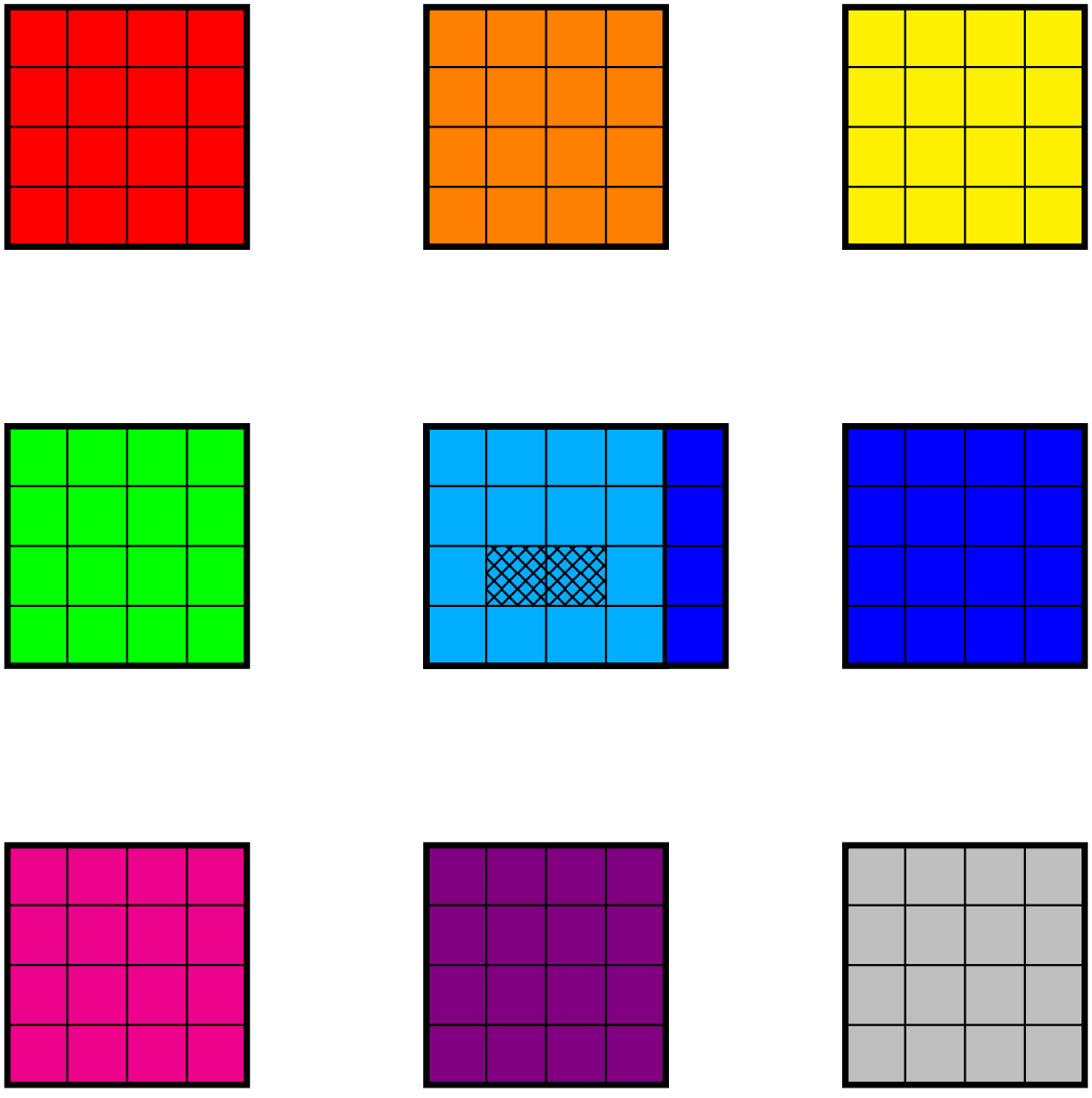
Décomposition de domaine



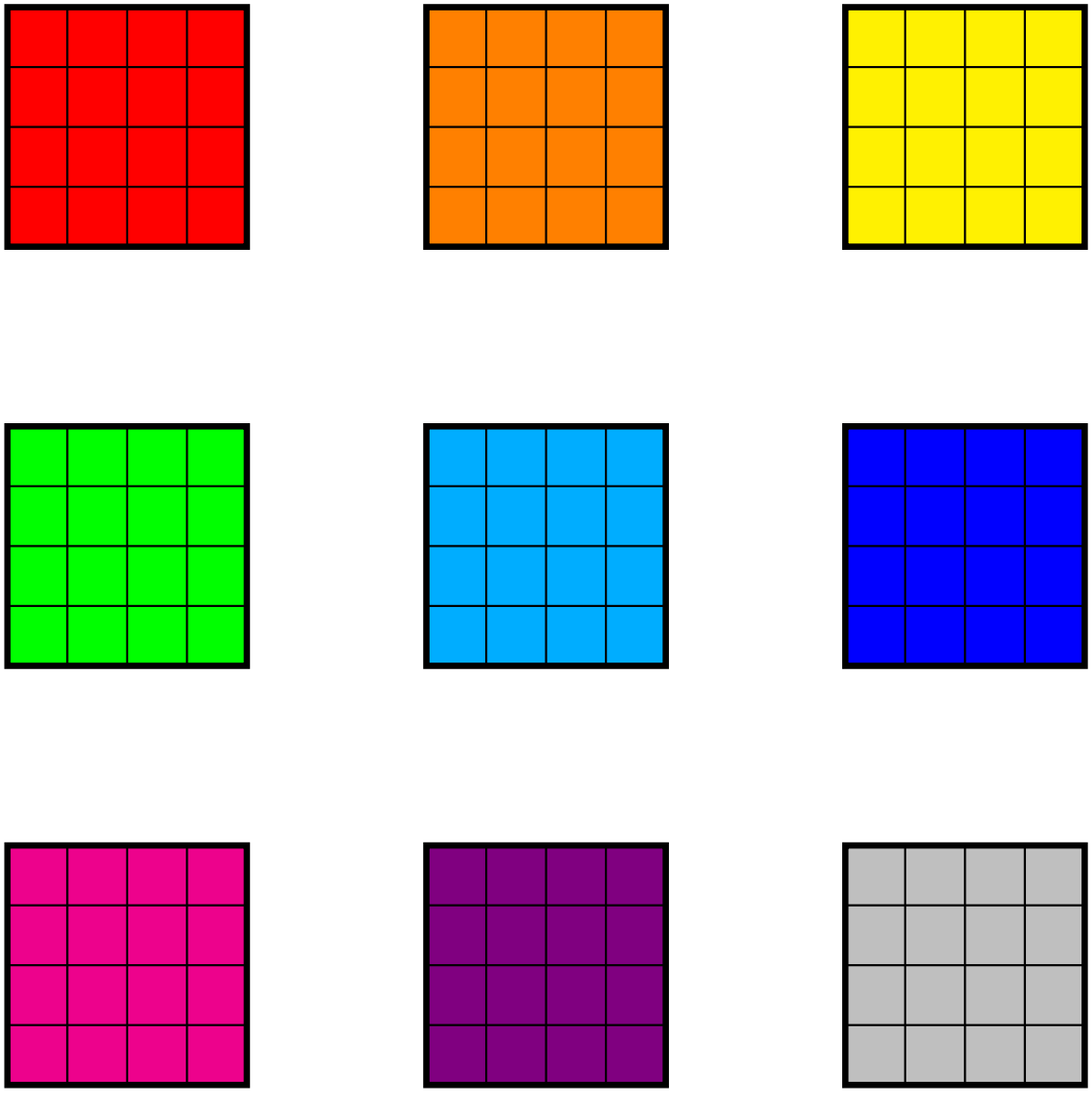
Décomposition de domaine



Décomposition de domaine



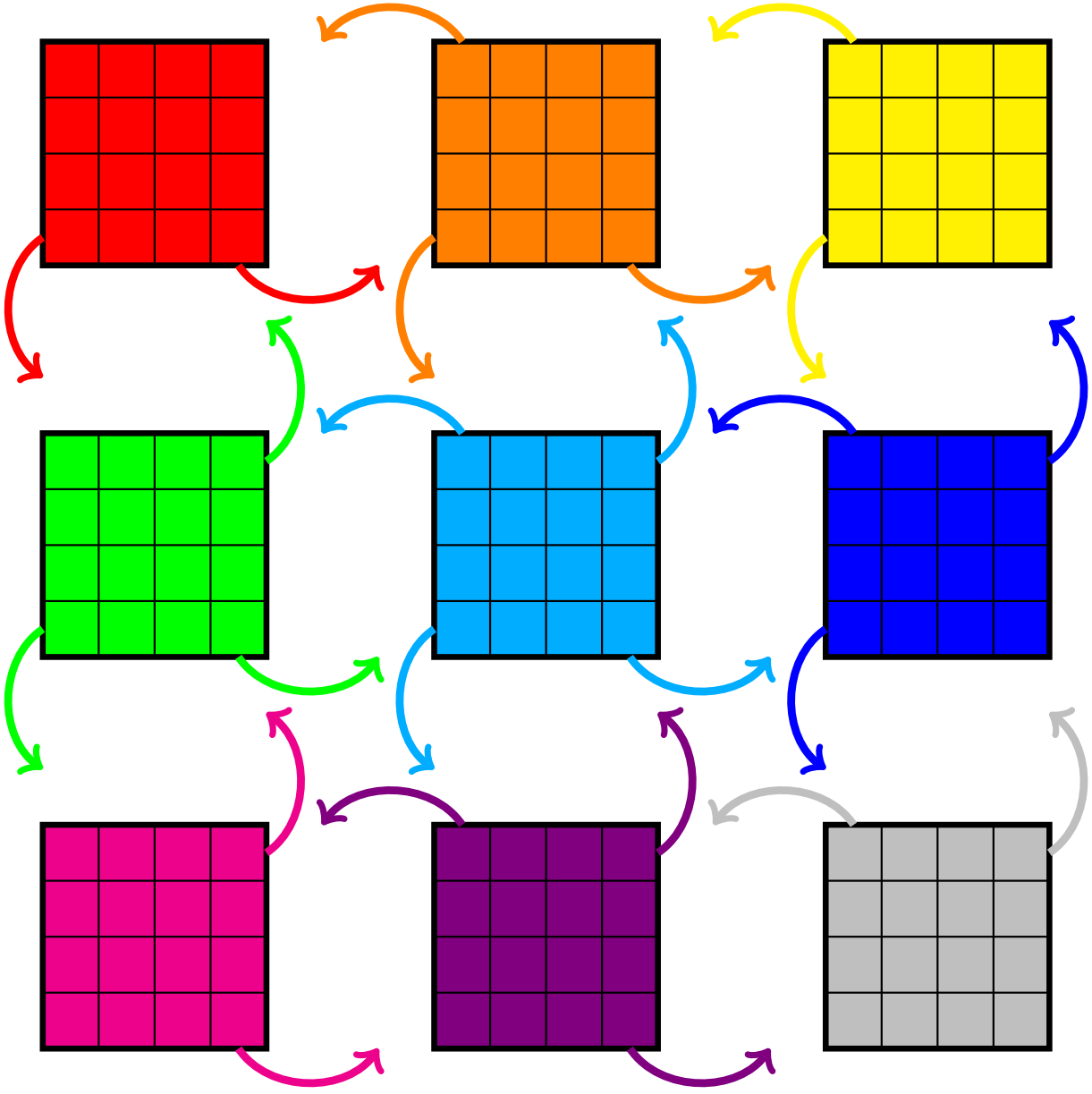
Décomposition de domaine



Chaque processeur :

- connaît T à l'instant t.

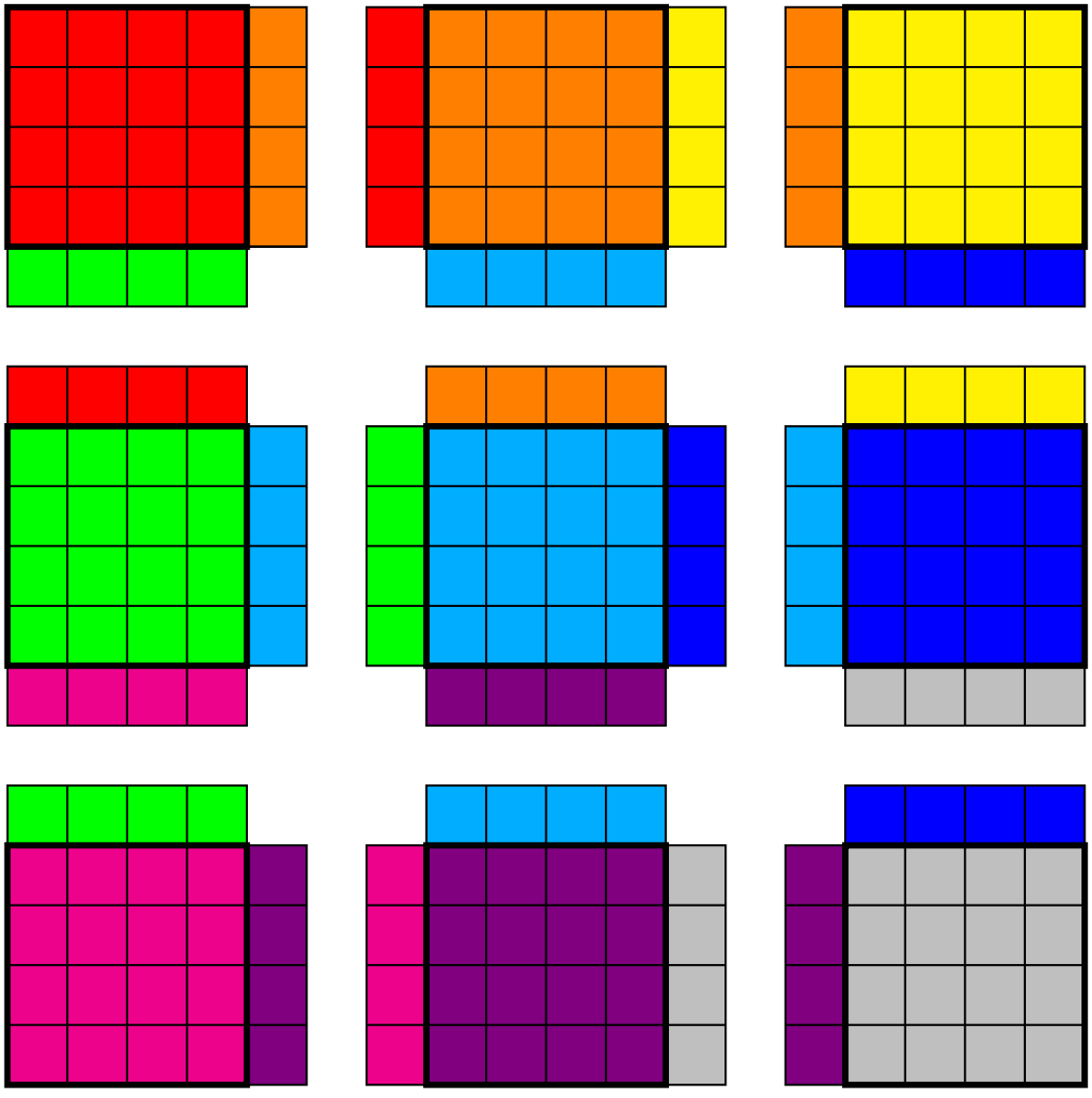
Décomposition de domaine



Chaque processeur :

- connaît T à l'instant t.
- envoie/reçoit le halo de ses voisins.

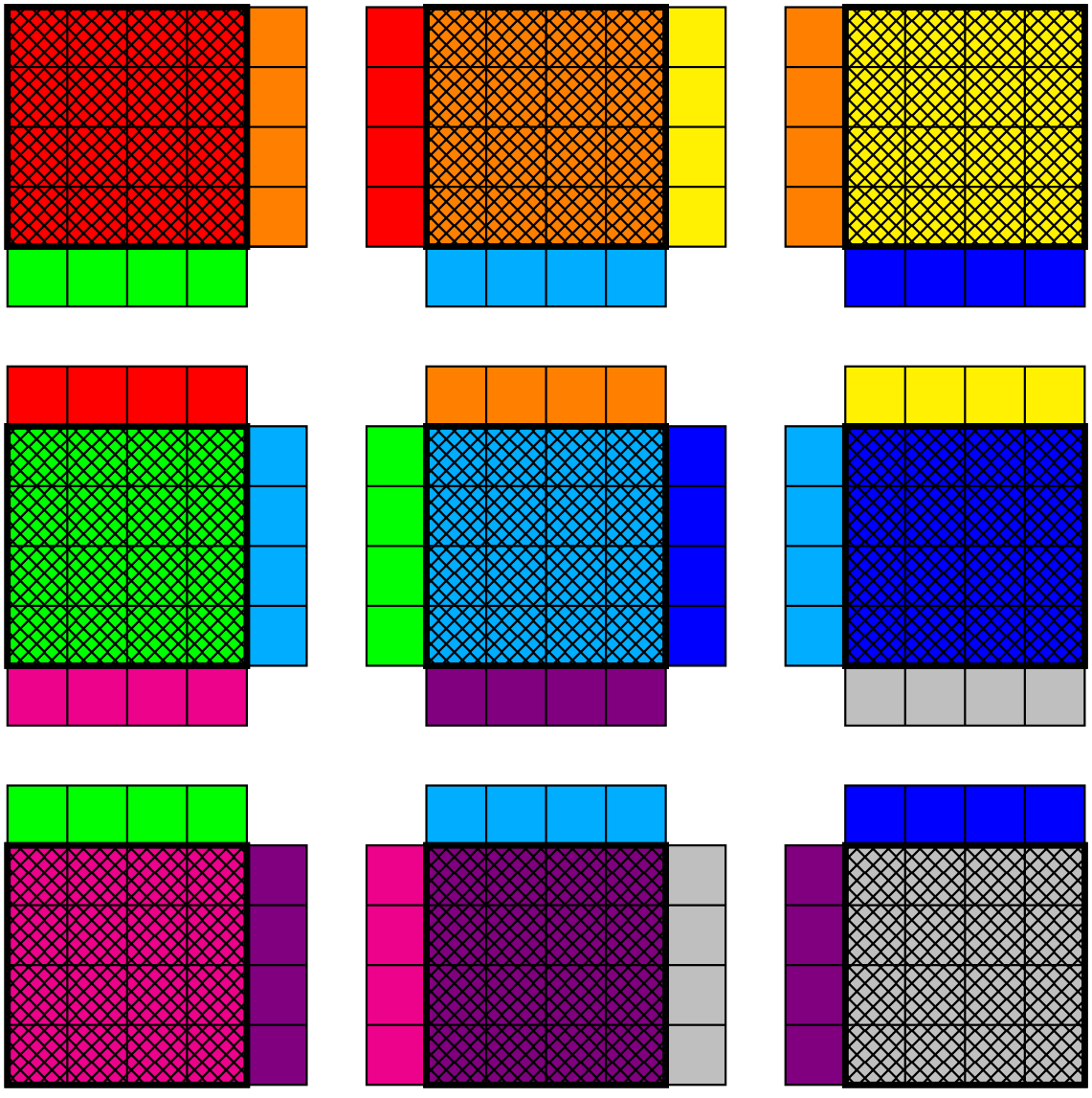
Décomposition de domaine



Chaque processeur :

- connaît T à l'instant t.
- envoie/reçoit le halo de ses voisins.

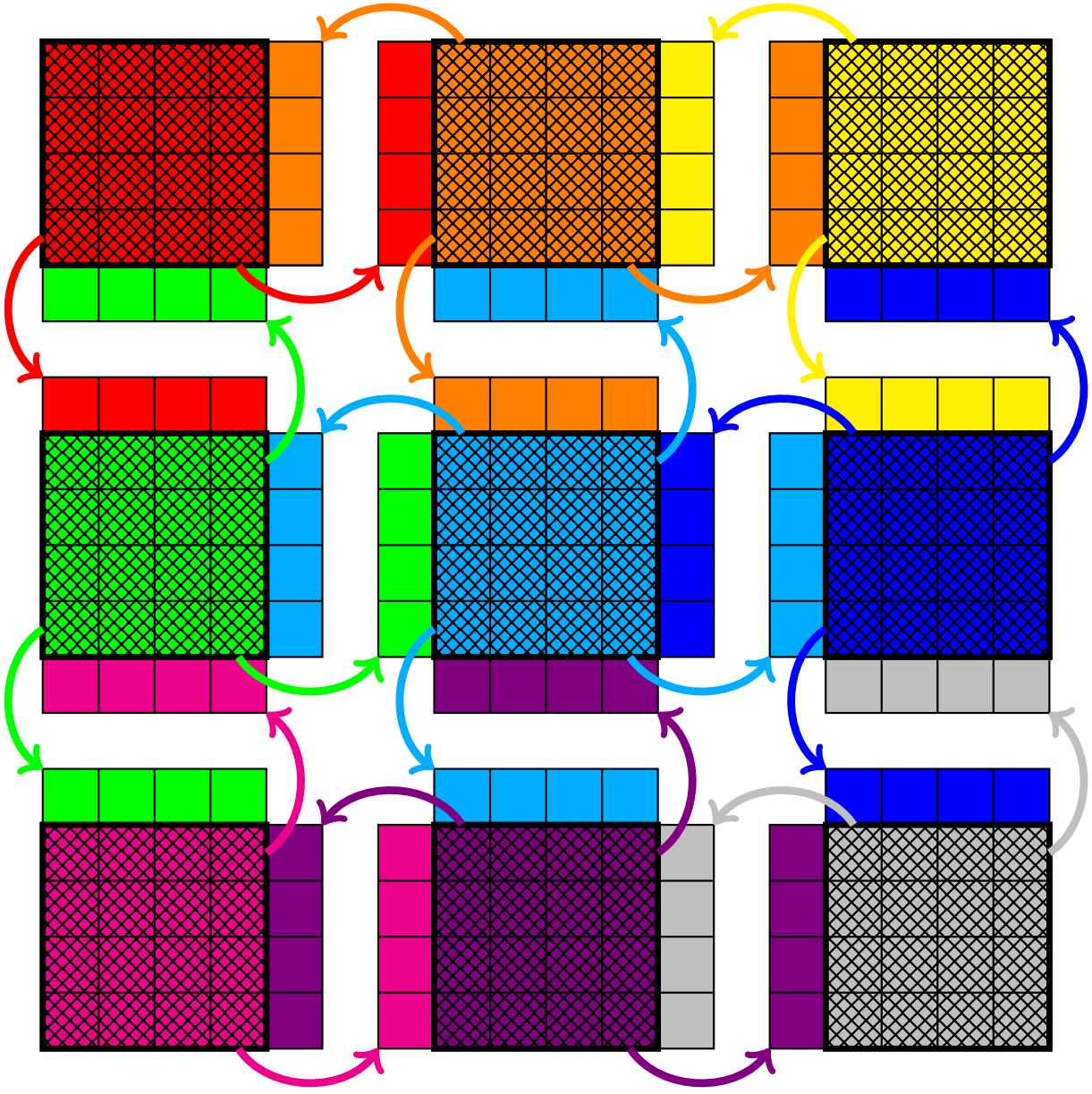
Décomposition de domaine



Chaque processeur :

- connaît T à l'instant t.
- envoie/reçoit le halo de ses voisins.
- calcule T à l'instant $t + \Delta t$.

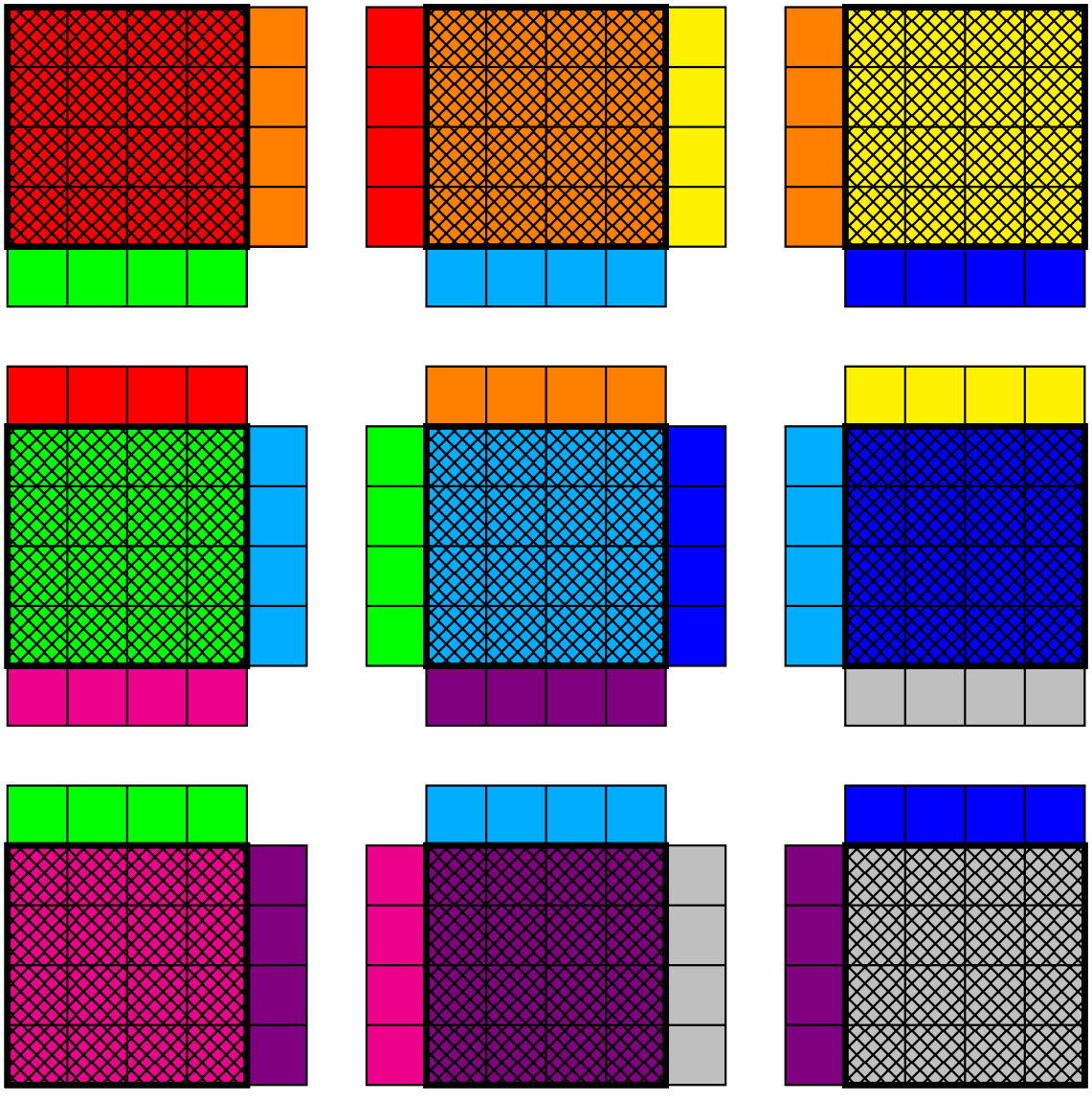
Décomposition de domaine



Chaque processeur :

- connaît T à l'instant t.
- envoie/reçoit le halo de ses voisins.
- calcule T à l'instant $t + \Delta t$.
- recommencer...

Décomposition de domaine



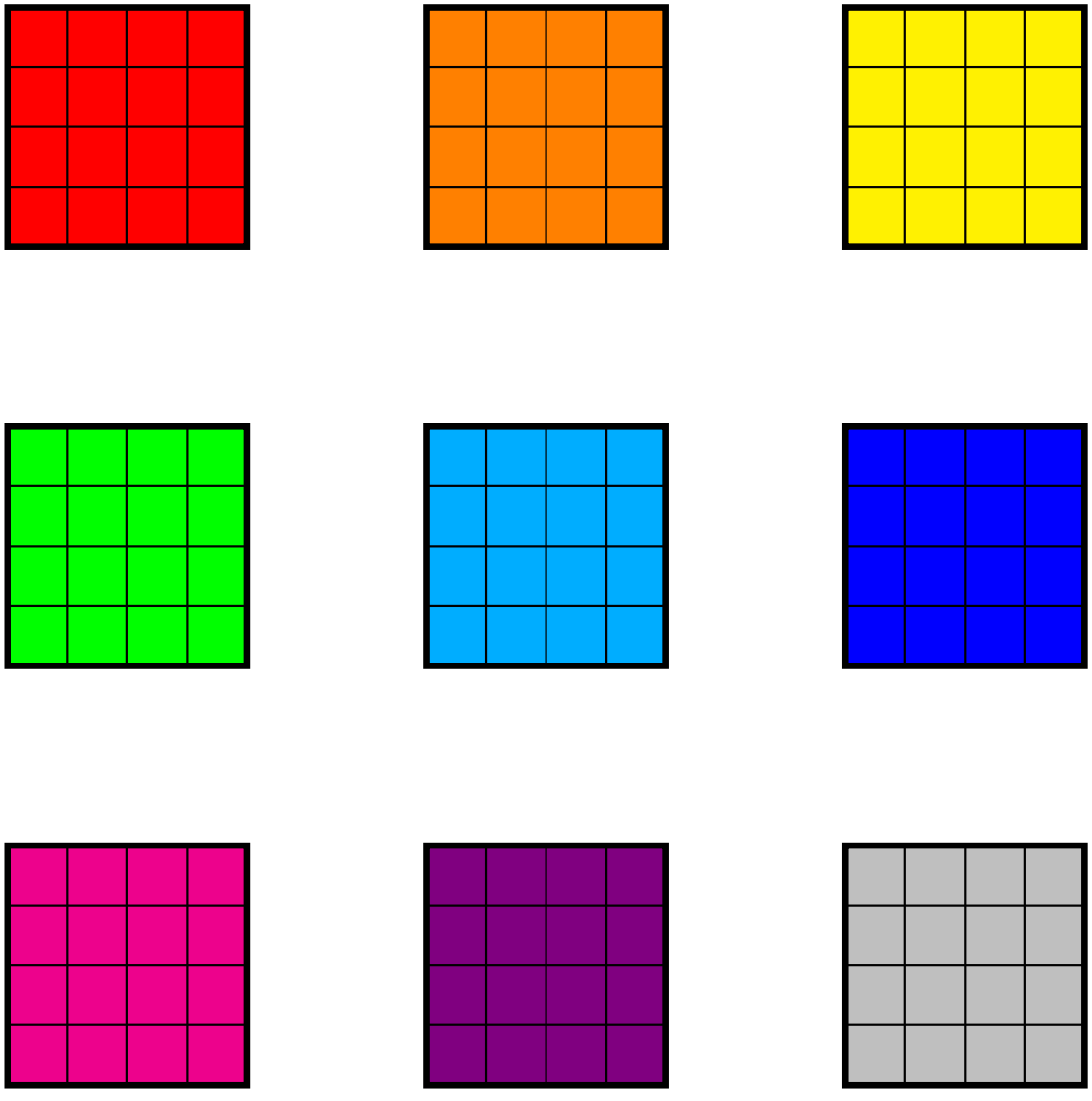
Chaque processeur :

- connaît T à l'instant t.
- envoie/reçoit le halo de ses voisins.
- calcule T à l'instant $t + \Delta t$.
- recommencer...

Problème

Calculs bloqués par les communications

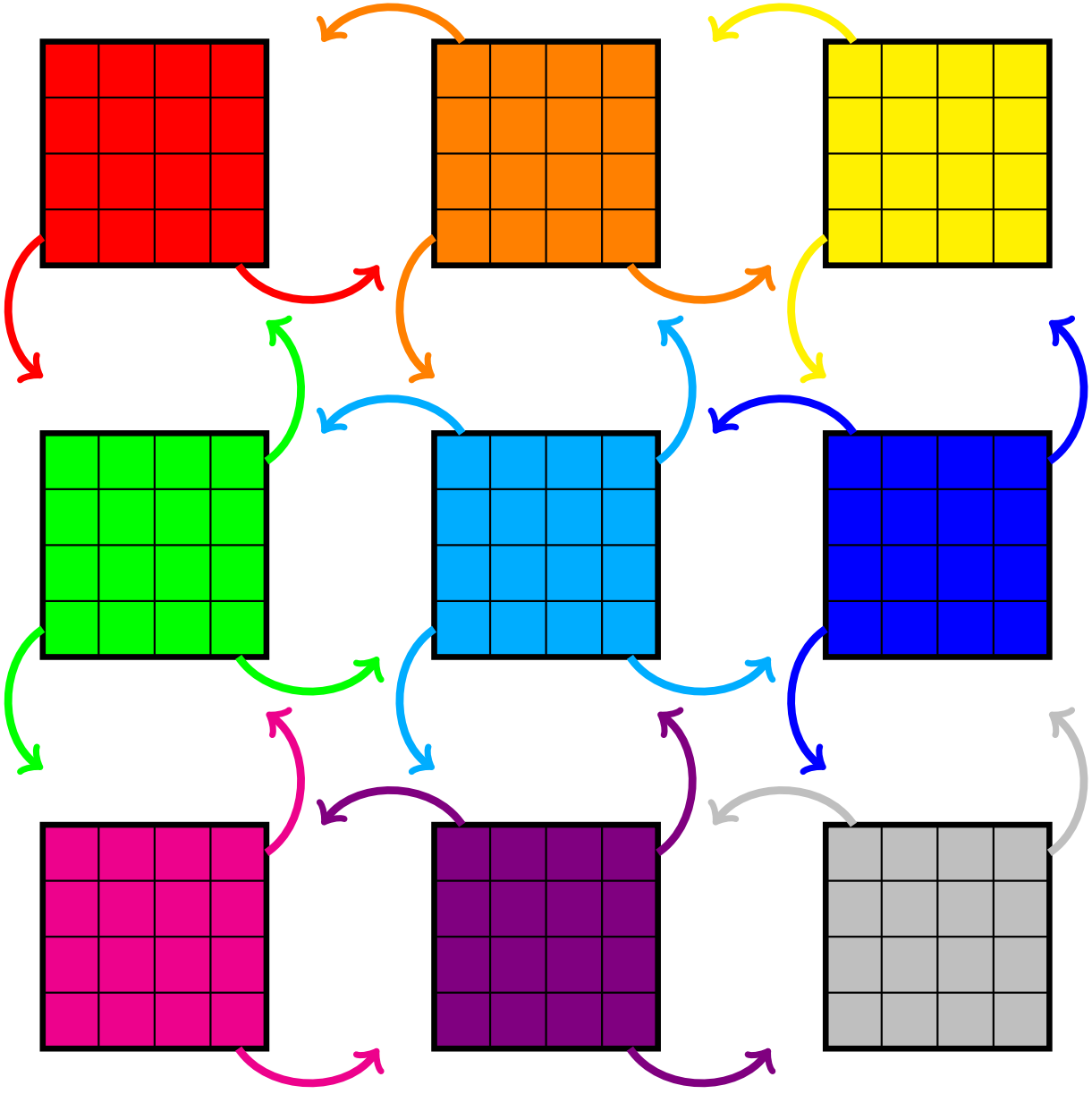
Décomposition de domaine



Chaque processeur :

- connaît T à l'instant t.

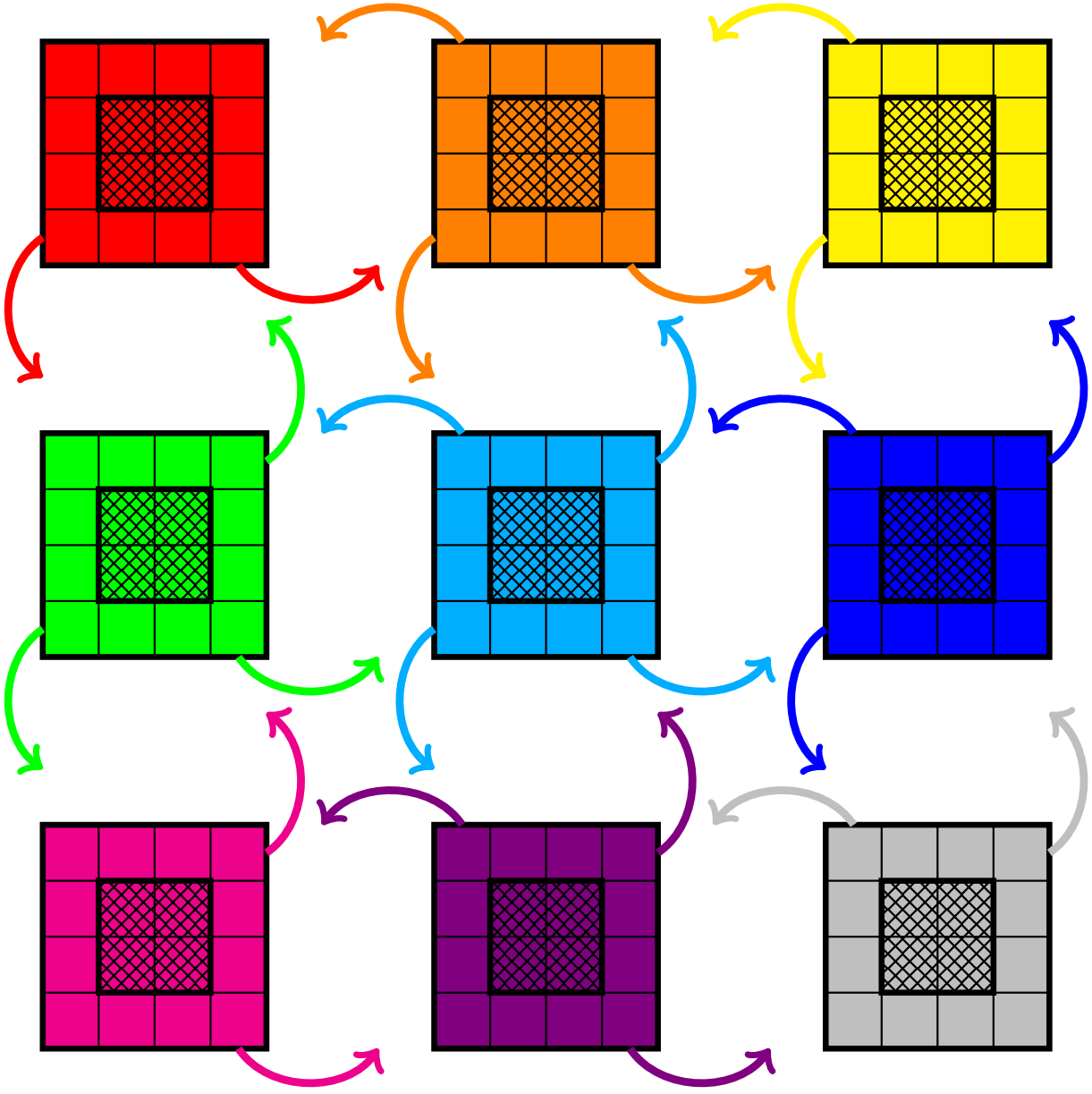
Décomposition de domaine



Chaque processeur :

- connaît T à l'instant t.
- envoie le halo à ses voisins (Isend) .

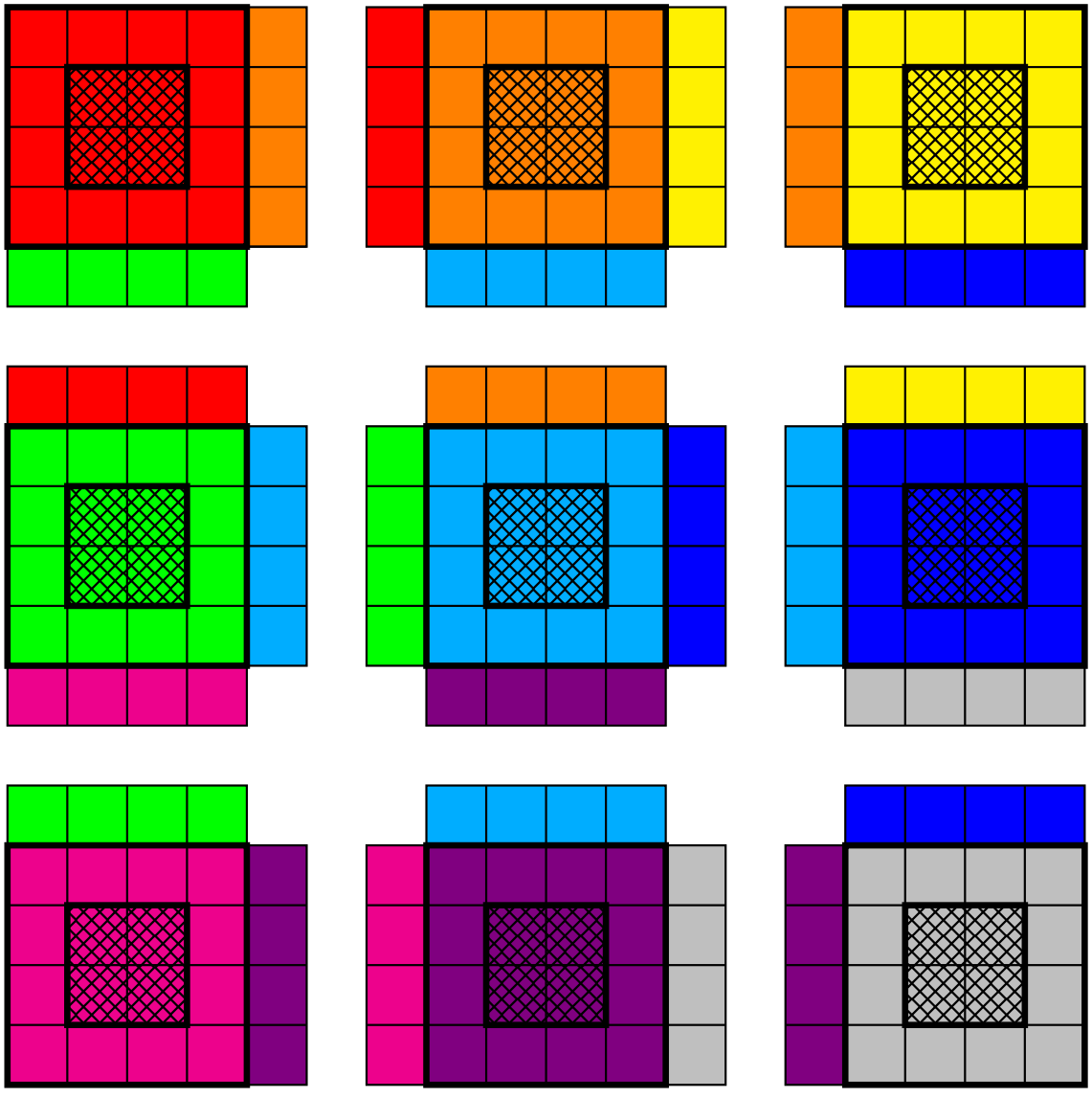
Décomposition de domaine



Chaque processeur :

- connaît T à l'instant t.
- envoie le halo à ses voisins (Isend) .
- calcule T à l'instant $t + \Delta t$ à l'intérieur.

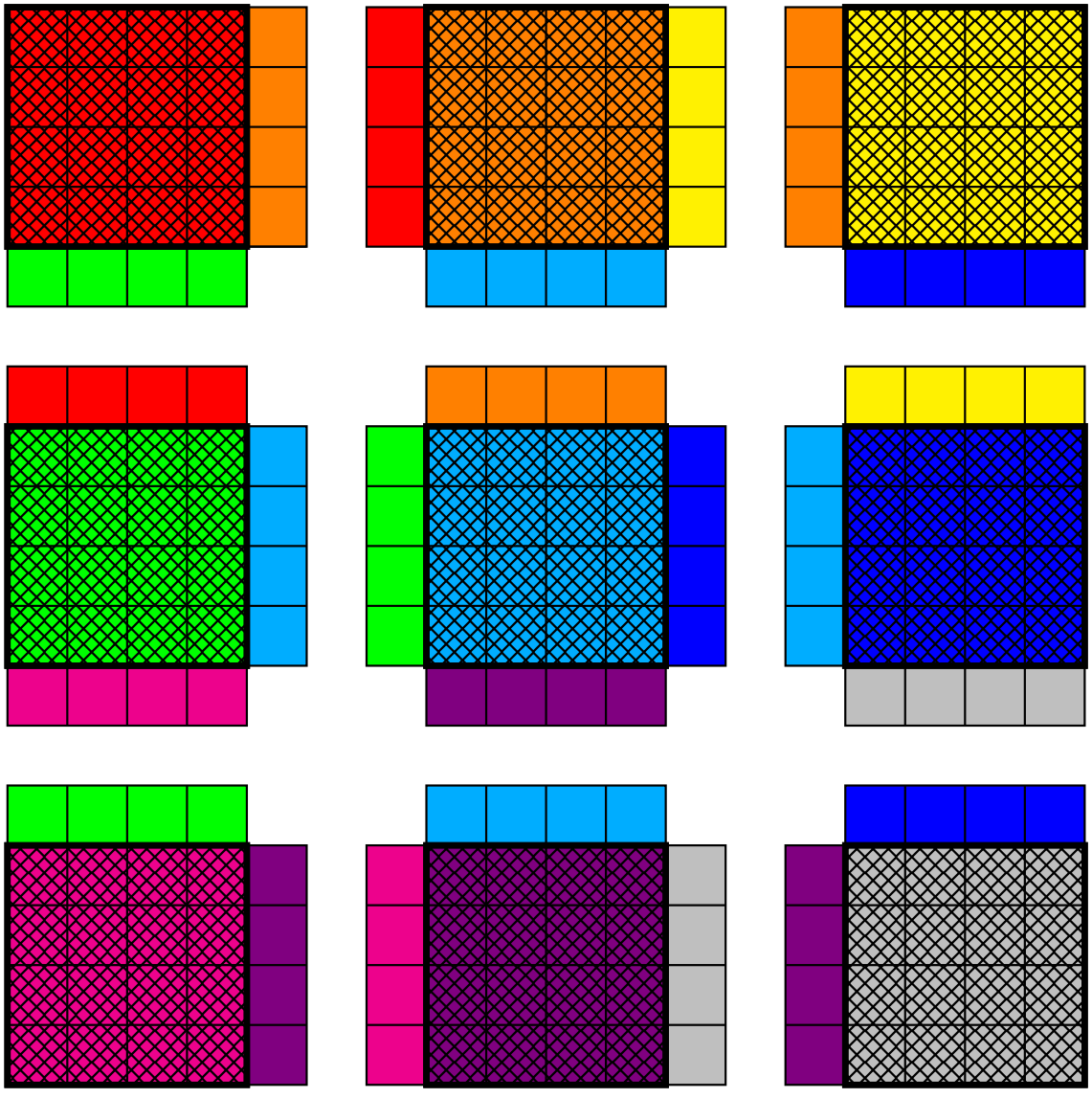
Décomposition de domaine



Chaque processeur :

- connaît T à l'instant t.
- envoie le halo à ses voisins (Isend) .
- calcule T à l'instant $t + \Delta t$ à l'intérieur.
- attend fin des comms

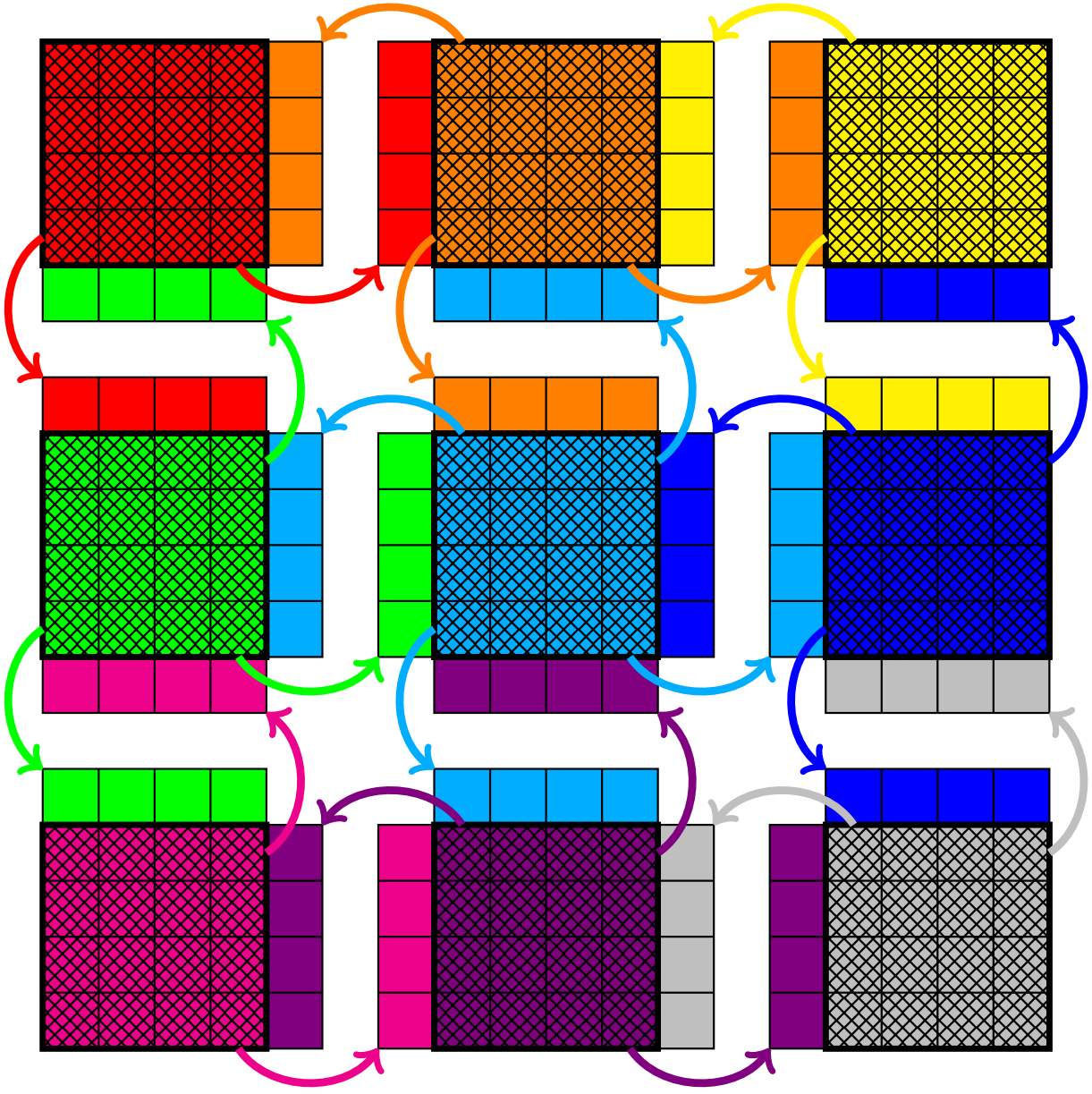
Décomposition de domaine



Chaque processeur :

- connaît T à l'instant t.
- envoie le halo à ses voisins (Isend) .
- calcule T à l'instant $t + \Delta t$ à l'intérieur.
- attend fin des comms
- calcule T à l'instant $t + \Delta t$ sur les bords.

Décomposition de domaine



Chaque processeur :

- connaît T à l'instant t.
- envoie le halo à ses voisins (Isend) .
- calcule T à l'instant $t + \Delta t$ à l'intérieur.
- attend fin des comms
- calcule T à l'instant $t + \Delta t$ sur les bords.
- recommencer...

Bloquant :

```
int MPI_Sendrecv(void *sendbuf, int sendcount, MPI_Datatype sendtype,
                 int dest, int sendtag,
                 void *recvbuf, int recvcount, MPI_Datatype recvtype,
                 int source, int recvtag,
                 MPI_Comm comm, MPI_Status *status);
```

Non-bloquant :

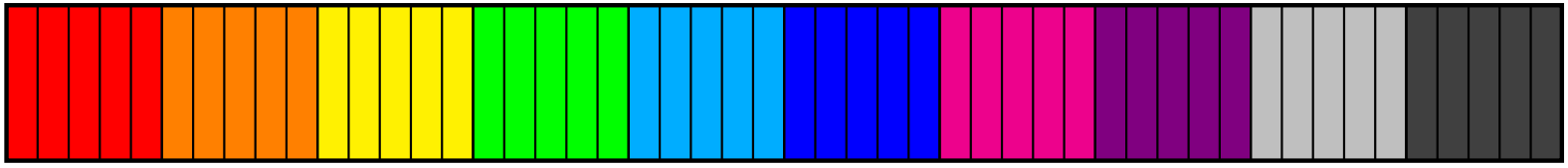
```
int MPI_Isend(void *buf, int count, MPI_Datatype datatype,
              int dest, int tag,
              MPI_Comm comm, MPI_Request *request);
int MPI_Irecv(void *buf, int count, MPI_Datatype datatype,
              int source, int tag,
              MPI_Comm comm, MPI_Request *request);
int MPI_Waitall(int count,
                MPI_Request requests[], MPI_Status statuses[]);
```

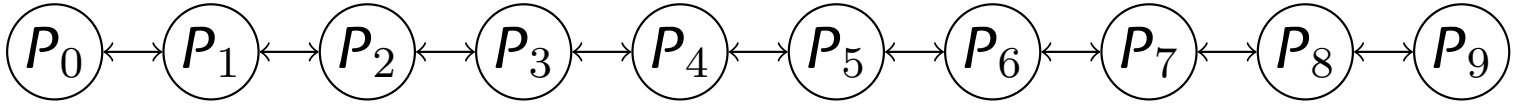
Attention !

- Bord est / ouest : données non-contigües en mémoire !

$\Rightarrow$ création de types MPI dérivés (`MPI_Type_vector(...)`)

Tri d'un tableau réparti



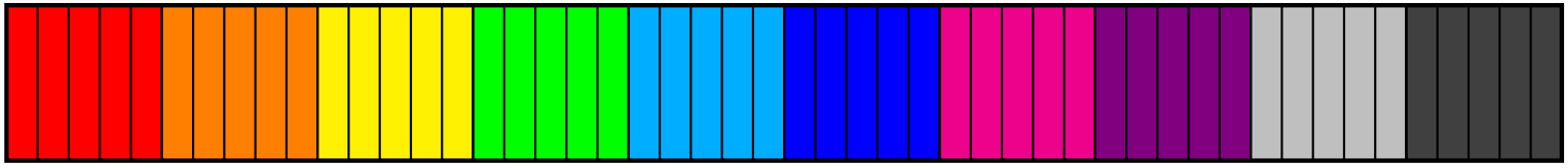


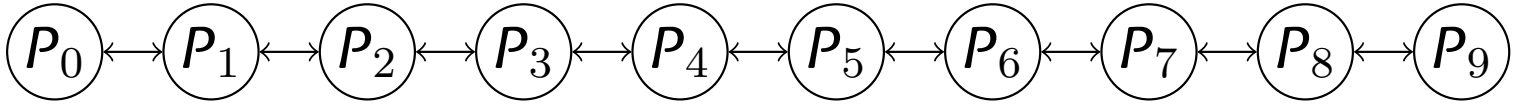
Algo TD n°1 : *Odd-Even Merge Sort*

1. P_i trie localement ses données.
2. $p-1$ phases :
   - 2.1 P_{2i} échange sa portion avec P_{2i+1} ; garde les n/p plus petits.
   - 2.2 P_{2i} échange sa portion avec P_{2i-1} ; garde les n/p plus grands.

$$T = \underbrace{\frac{n}{p}\log\frac{n}{p} + 2(p-1)\frac{n}{p}}_{\text{calcul}} + \underbrace{2(p-1)\left(\alpha + \frac{n}{p}\beta\right)}_{\text{comm}}$$

Tri d'un tableau réparti



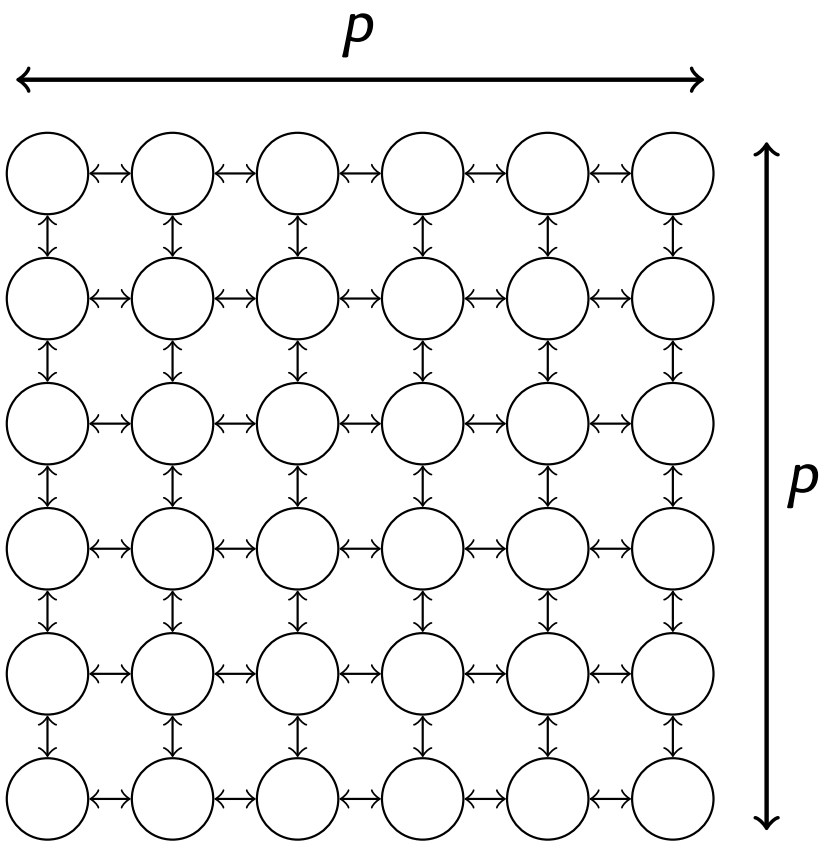


Algo TD n°1 : *Odd-Even Merge Sort*

1. P_i trie localement ses données.
2. $p-1$ phases :
   - 2.1 P_{2i} échange sa portion avec P_{2i+1} ; garde les n/p plus petits.
   - 2.2 P_{2i} échange sa portion avec P_{2i-1} ; garde les n/p plus grands.

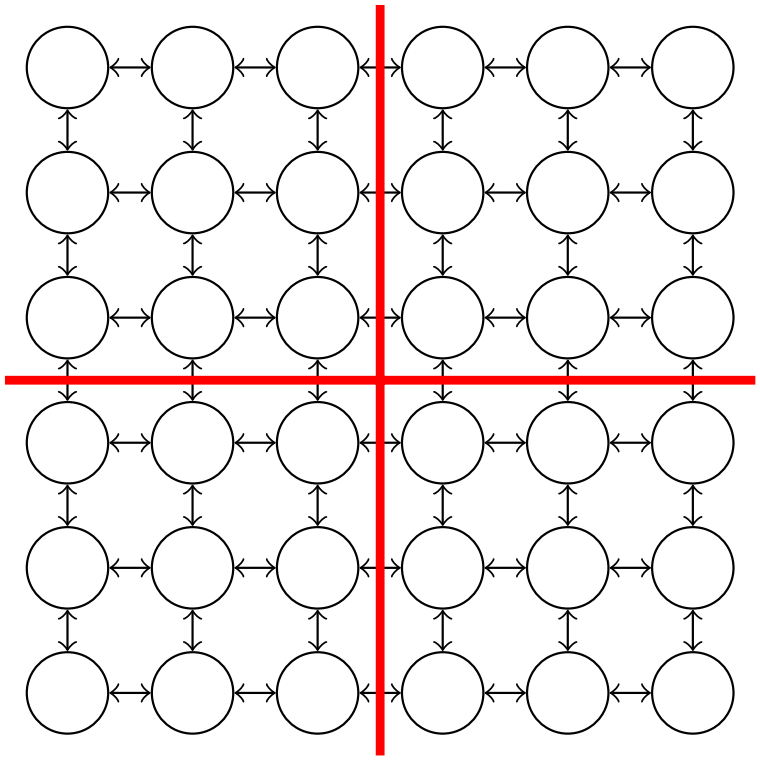
$$T = \underbrace{\frac{n}{p}\log\frac{n}{p} + 2(p-1)\frac{n}{p}}_{\text{calcul}} + \underbrace{2(p-1)\left(\alpha + \frac{n}{p}\beta\right)}_{\text{comm}} = \mathcal{O}\left(\frac{n\log n}{p} + p + n\right)$$

Tri d'un tableau réparti sur un *Mesh* 2D



Algo de Schimmler (1987)

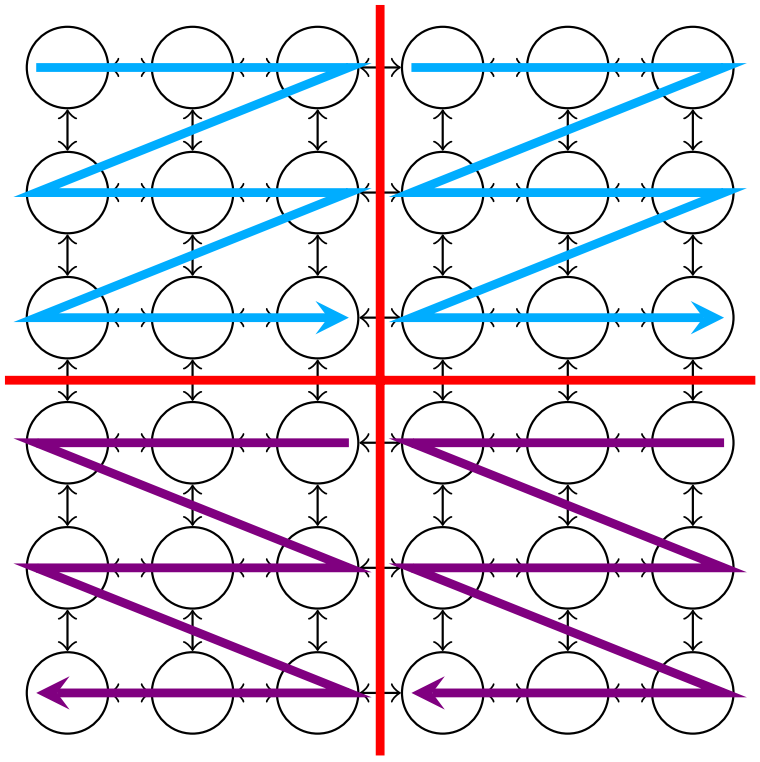
Tri d'un tableau réparti sur un *Mesh* 2D



Algo de Schimmler (1987)

1. Trier les quadrants (récursion)

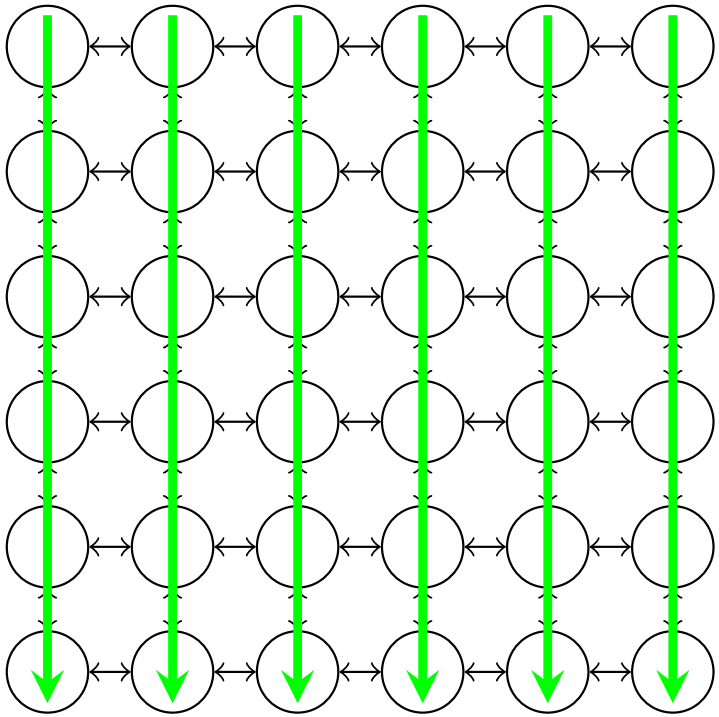
Tri d'un tableau réparti sur un *Mesh* 2D



Algo de Schimmler (1987)

1. Trier les quadrants (récursion)

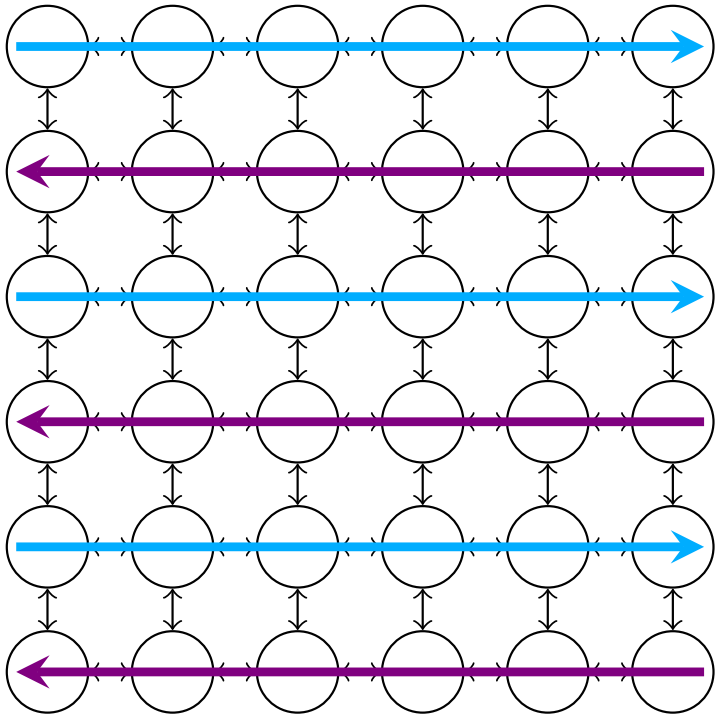
Tri d'un tableau réparti sur un *Mesh* 2D



Algo de Schimmler (1987)

1. Trier les quadrants (récursion)
2. Trier les colonnes (algo 1D)

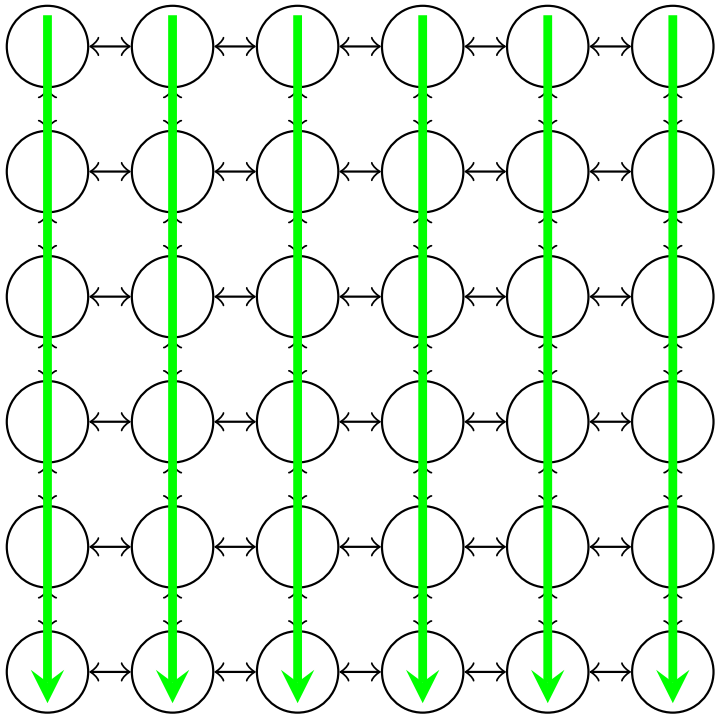
Tri d'un tableau réparti sur un *Mesh* 2D



Algo de Schimmler (1987)

1. Trier les quadrants (récursion)
2. Trier les colonnes (algo 1D)
3. Trier les lignes (algo 1D)

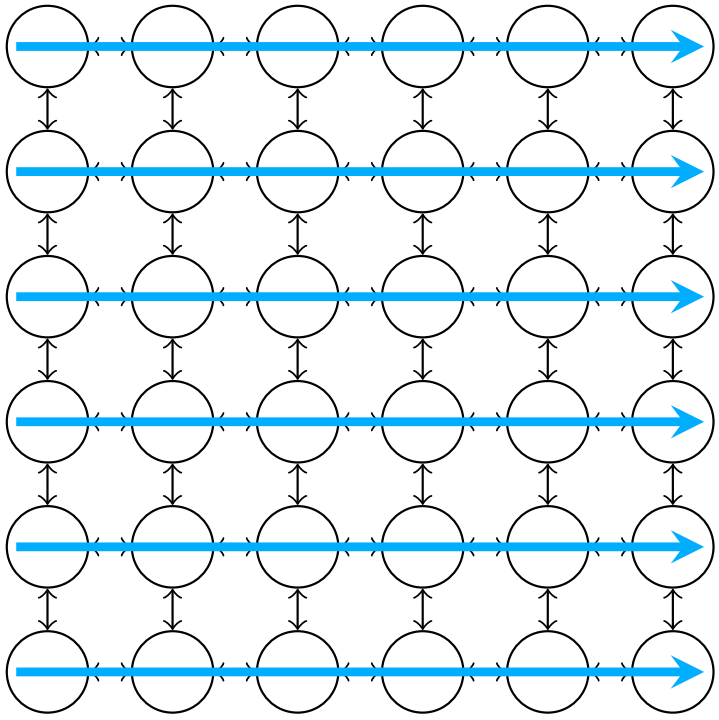
Tri d'un tableau réparti sur un *Mesh* 2D



Algo de Schimmler (1987)

1. Trier les quadrants (récursion)
2. Trier les colonnes (algo 1D)
3. Trier les lignes (algo 1D)
4. Trier les colonnes (algo 1D)

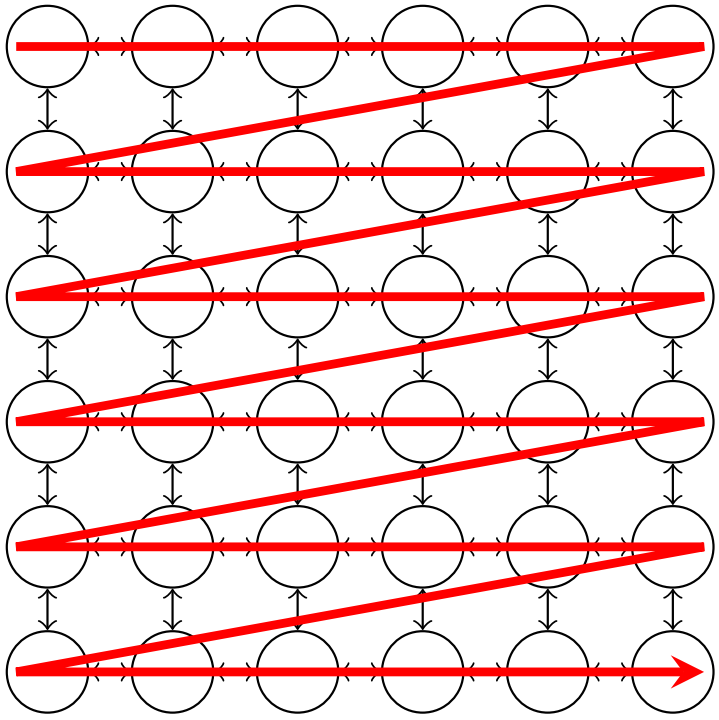
Tri d'un tableau réparti sur un *Mesh* 2D



Algo de Schimmler (1987)

1. Trier les quadrants (récursion)
2. Trier les colonnes (algo 1D)
3. Trier les lignes (algo 1D)
4. Trier les colonnes (algo 1D)
5. Trier les lignes (algo 1D)

Tri d'un tableau réparti sur un *Mesh* 2D

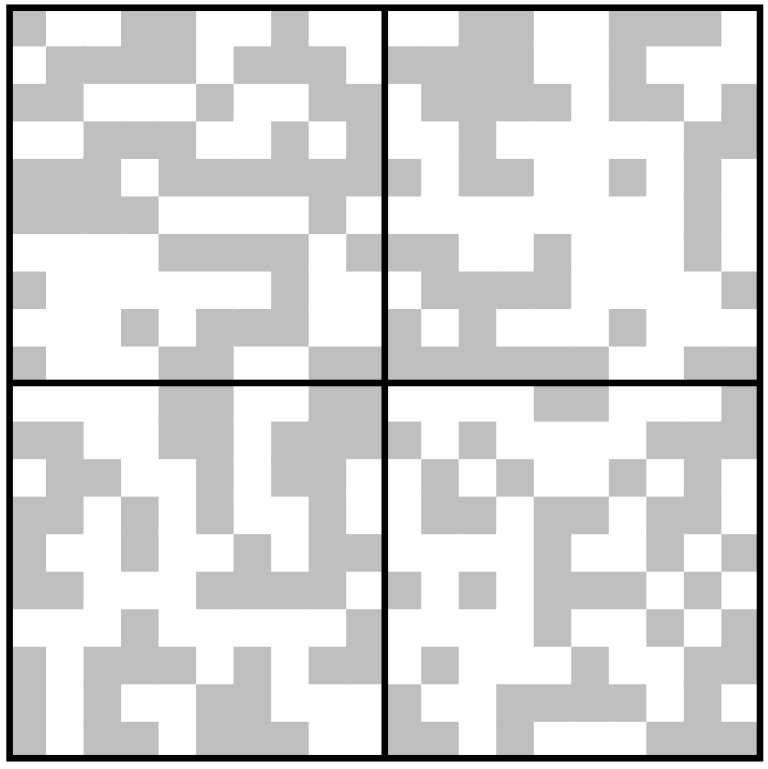


Algo de Schimmler (1987)

1. Trier les quadrants (récursion)
2. Trier les colonnes (algo 1D)
3. Trier les lignes (algo 1D)
4. Trier les colonnes (algo 1D)
5. Trier les lignes (algo 1D)
6. Et boum ! C'est trié.

Tri d'un tableau réparti sur un *Mesh* 2D

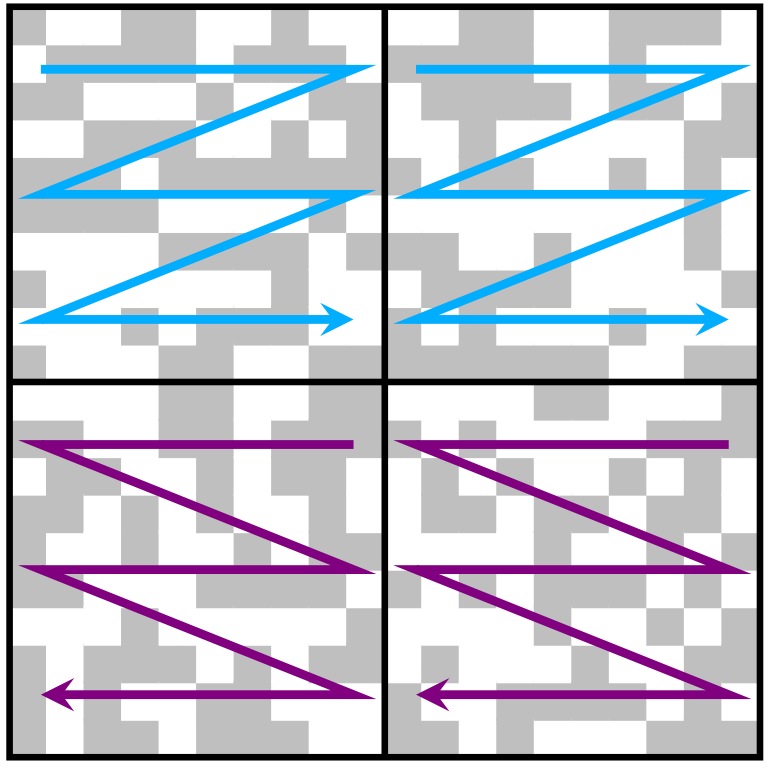
Justification



Tri par comparaison oblivious $\rightarrow$ principe 0/1 ;

Tri d'un tableau réparti sur un *Mesh* 2D

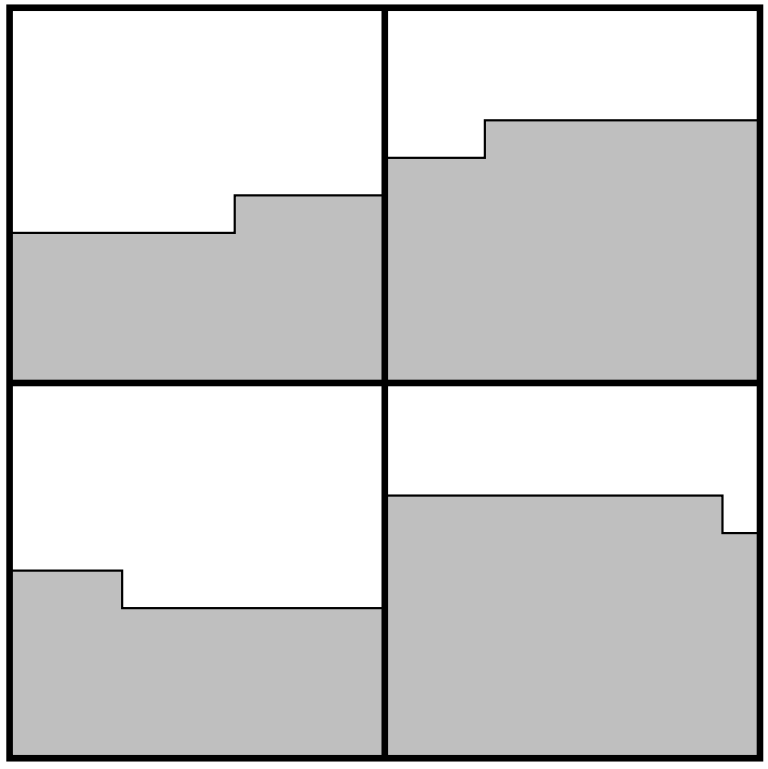
Justification



Tri par comparaison oblivious $\rightarrow$ principe 0/1 ;

Tri d'un tableau réparti sur un *Mesh* 2D

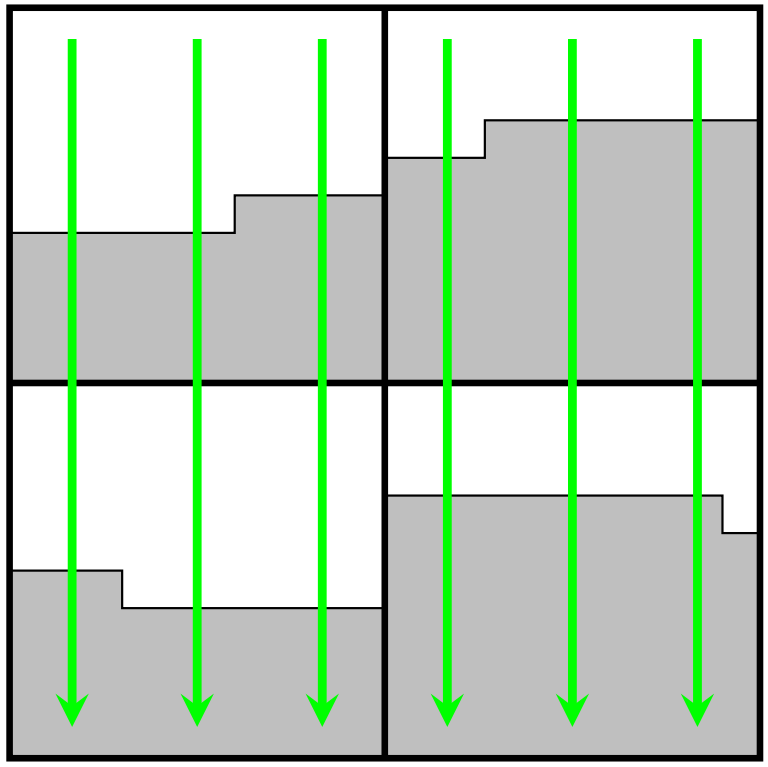
Justification



Tri par comparaison oblivious $\rightarrow$ principe 0/1 ;
≤ 1 ligne sale par quadrant ;

Tri d'un tableau réparti sur un *Mesh* 2D

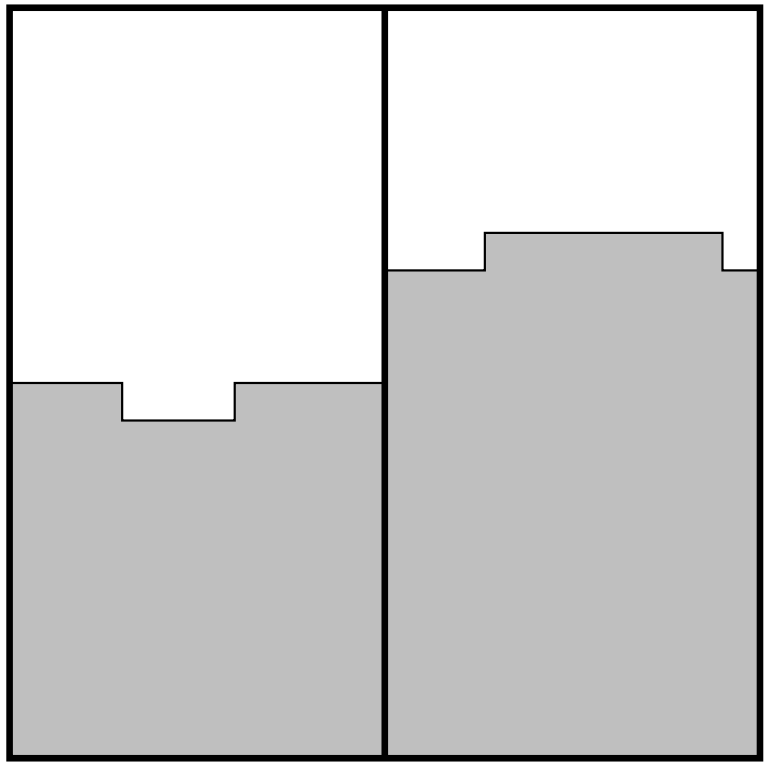
Justification



Tri par comparaison oblivious $\rightarrow$ principe 0/1 ;
≤ 1 ligne sale par quadrant ;

Tri d'un tableau réparti sur un *Mesh* 2D

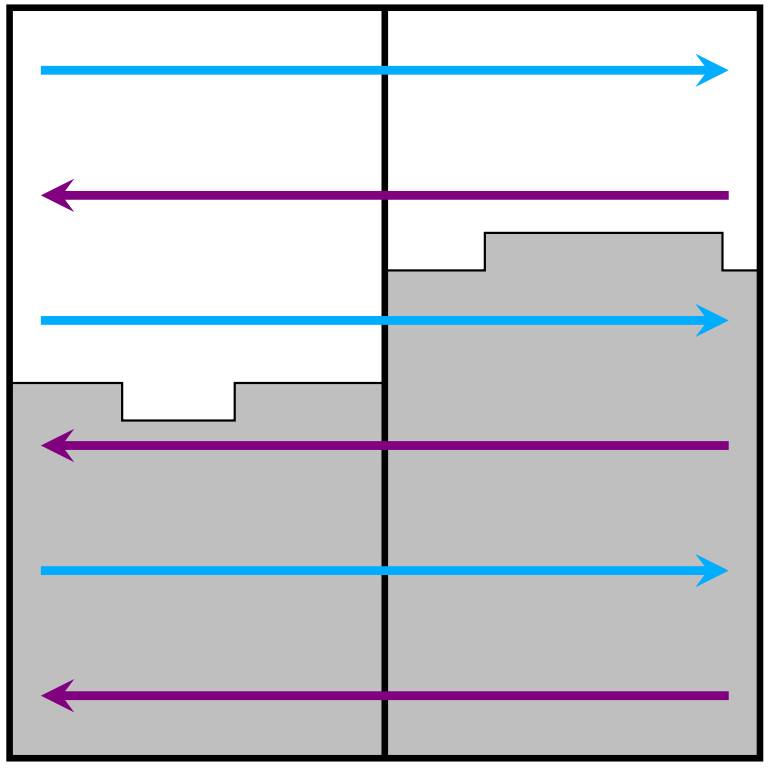
Justification



Tri par comparaison oblivious $\rightarrow$ principe 0/1 ;
≤ 1 ligne sale par quadrant ; ≤ 1 ligne sale par moitié ;

Tri d'un tableau réparti sur un *Mesh* 2D

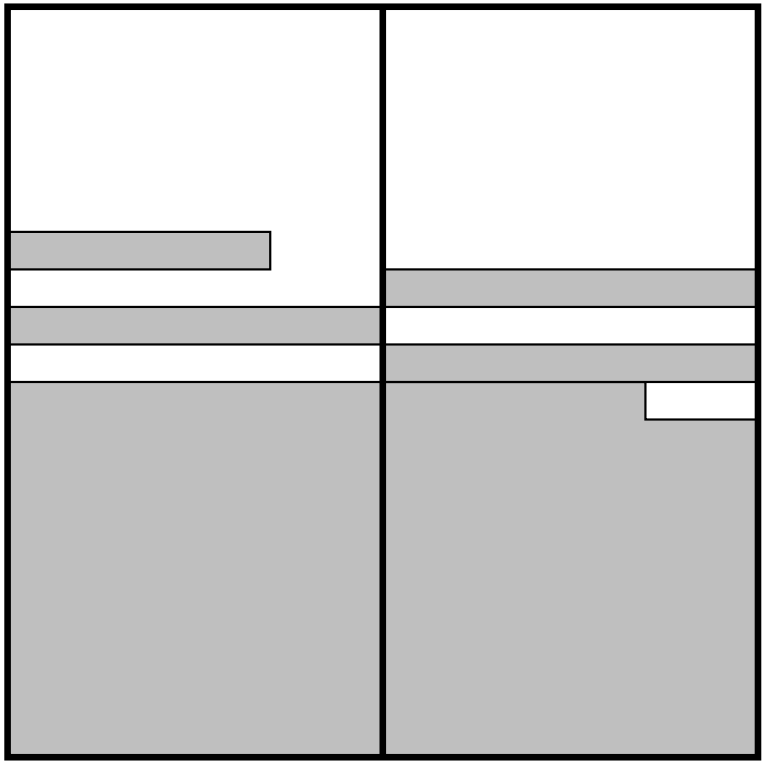
Justification

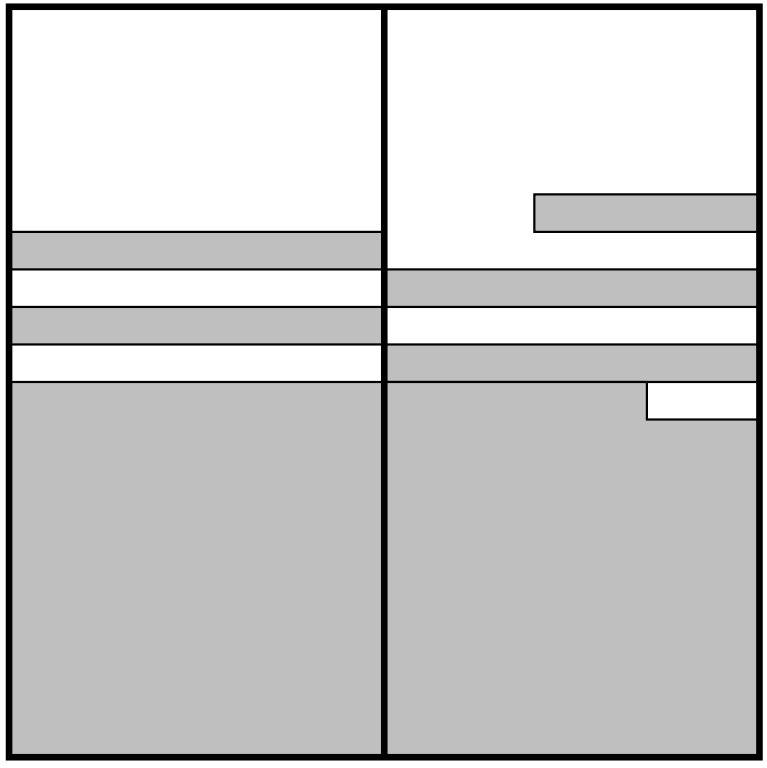


Tri par comparaison oblivious $\rightarrow$ principe 0/1 ;
≤ 1 ligne sale par quadrant ; ≤ 1 ligne sale par moitié ;

Tri d'un tableau réparti sur un *Mesh* 2D

Justification

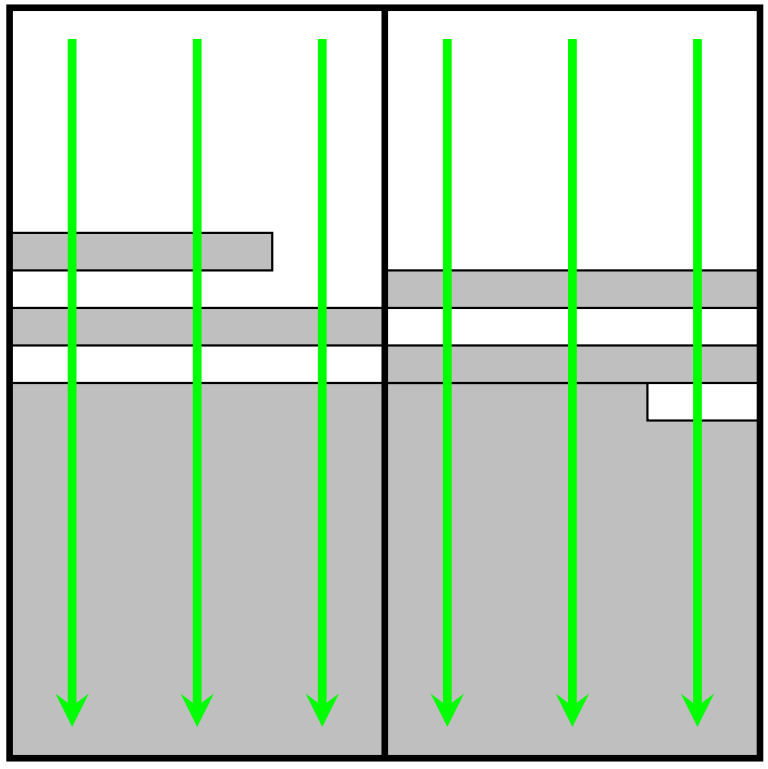


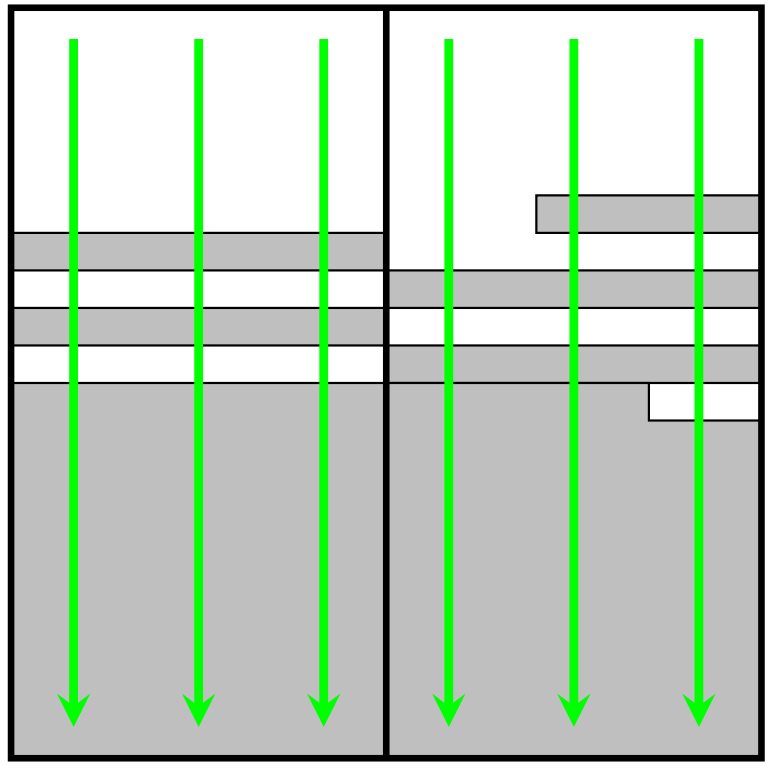


Tri par comparaison oblivious $\rightarrow$ principe 0/1 ;
≤ 1 ligne sale par quadrant ; ≤ 1 ligne sale par moitié ;
Du même côté ou pas ;

Tri d'un tableau réparti sur un *Mesh* 2D

Justification

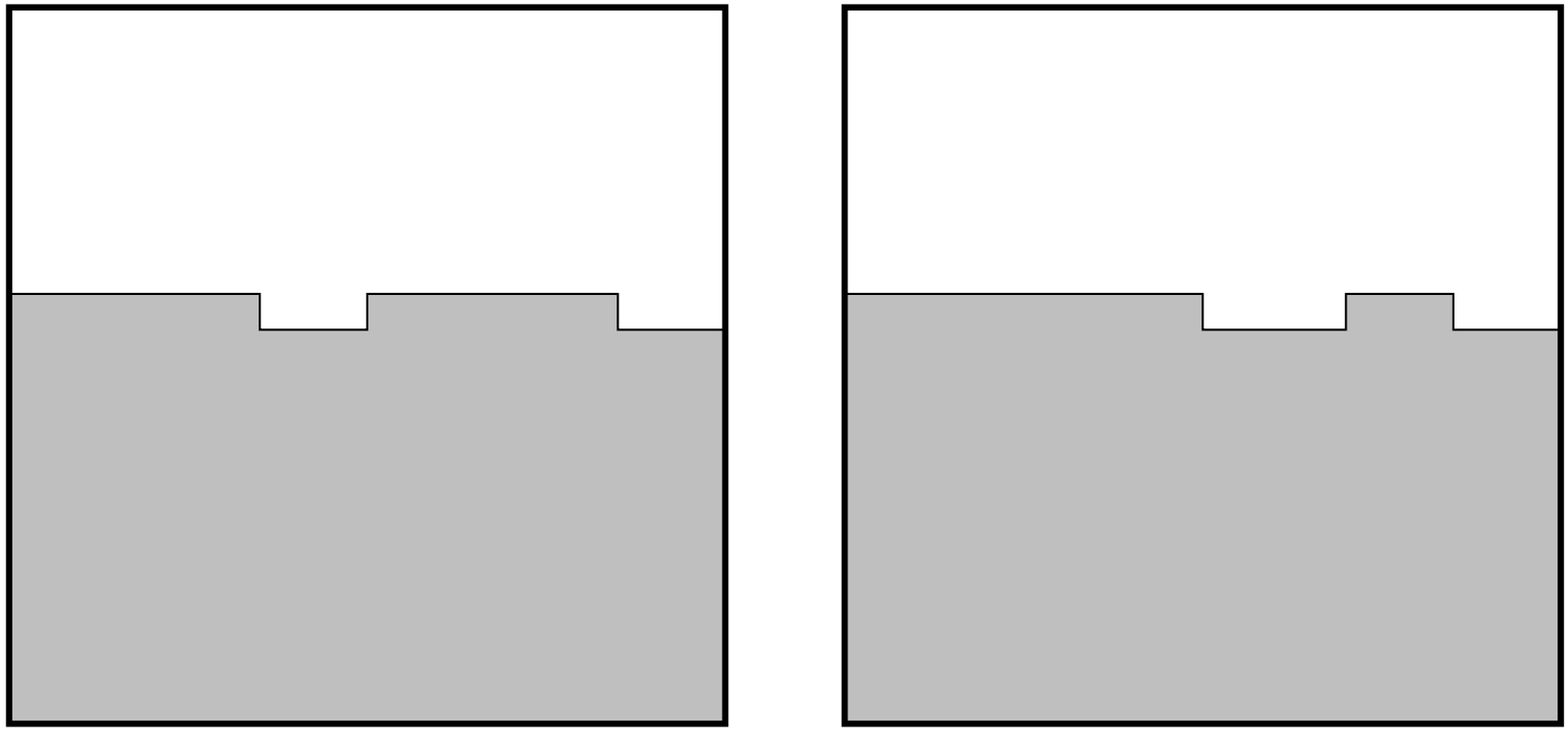




Tri par comparaison oblivious $\rightarrow$ principe 0/1 ;
≤ 1 ligne sale par quadrant ; ≤ 1 ligne sale par moitié ;
Du même côté ou pas ;

Tri d'un tableau réparti sur un *Mesh* 2D

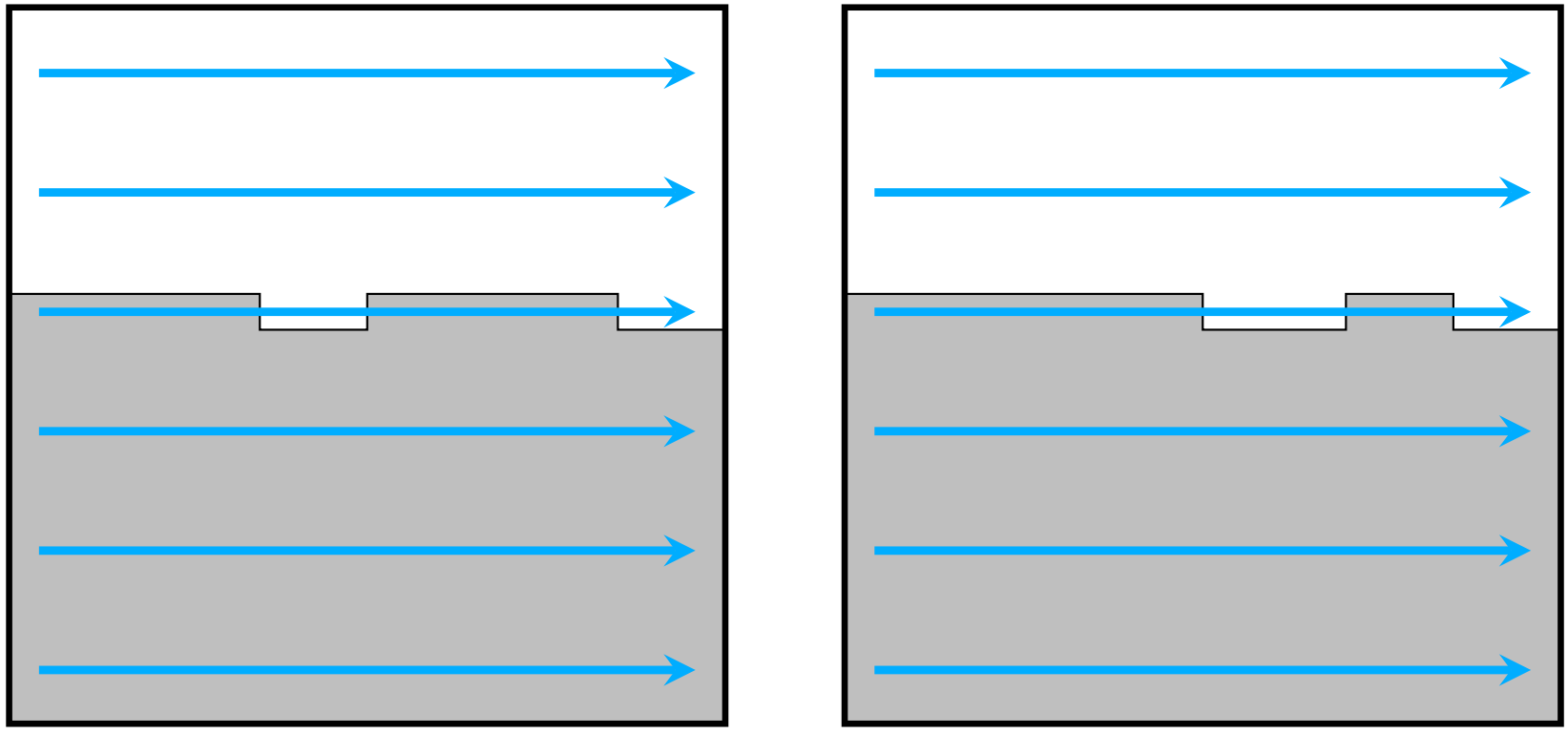
Justification



Tri par comparaison oblivious $\rightarrow$ principe 0/1 ;
≤ 1 ligne sale par quadrant ; ≤ 1 ligne sale par moitié ;
Du même côté ou pas ; ≤ 1 ligne sale en tout ;

Tri d'un tableau réparti sur un *Mesh* 2D

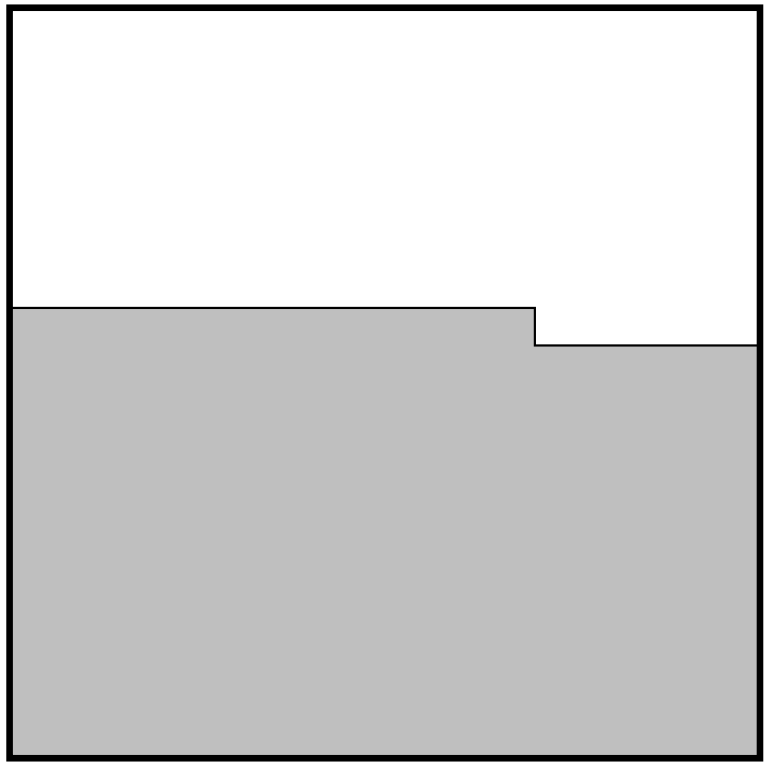
Justification

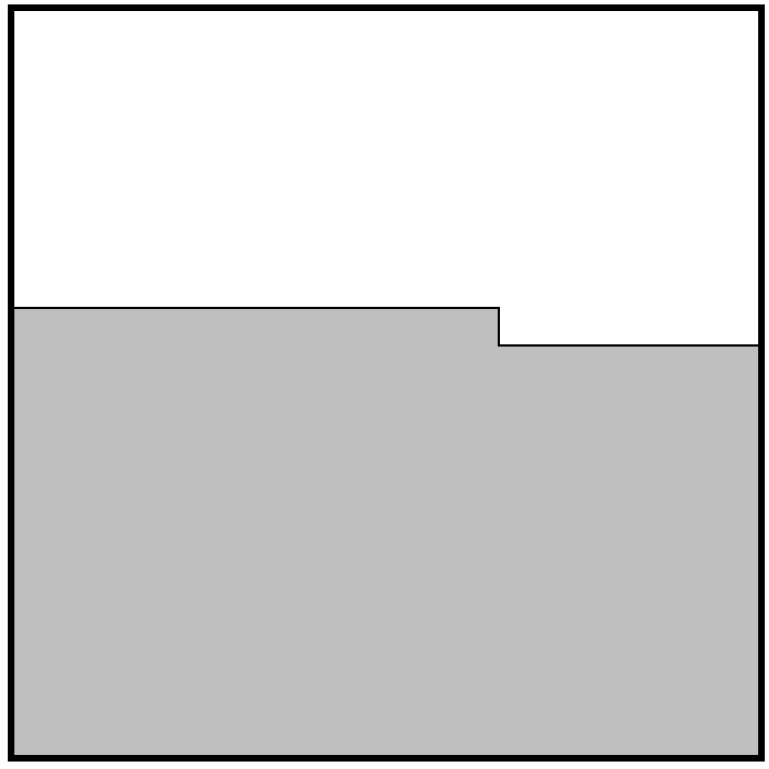


Tri par comparaison oblivious $\rightarrow$ principe 0/1 ;
≤ 1 ligne sale par quadrant ; ≤ 1 ligne sale par moitié ;
Du même côté ou pas ; ≤ 1 ligne sale en tout ;

Tri d'un tableau réparti sur un *Mesh* 2D

Justification





Tri par comparaison oblivious $\rightarrow$ principe 0/1 ;
≤ 1 ligne sale par quadrant ; ≤ 1 ligne sale par moitié ;
Du même côté ou pas ; ≤ 1 ligne sale en tout ; trié !

Tri d'un tableau réparti sur un *Mesh* 2D

Analyse

Tri 2D

1. Chaque processeur trie localement.
2. Fusion 2D (communications).

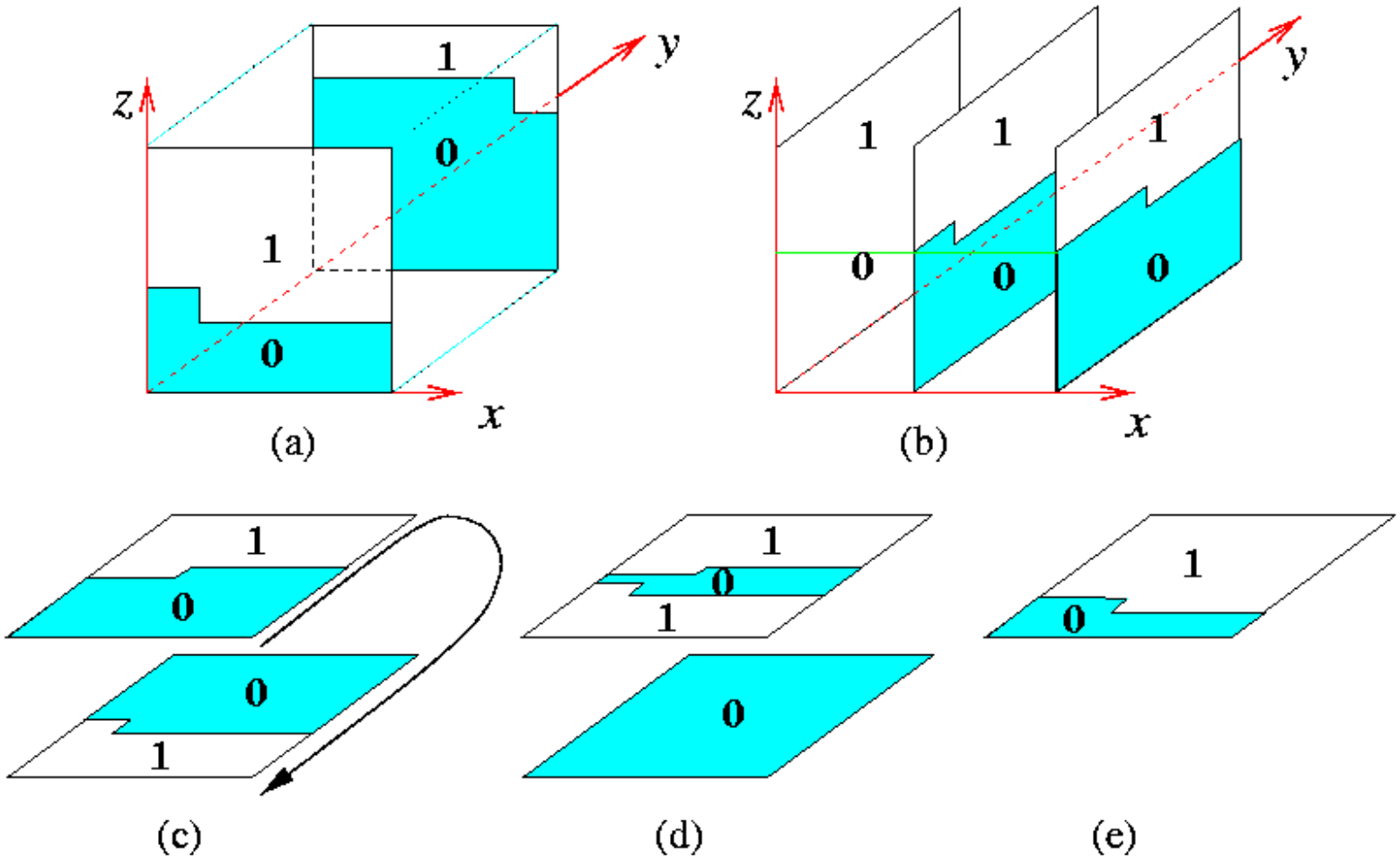
$$\begin{aligned} M(p) &= M\left(\frac{p}{2}\right) + 4 \times \text{Odd-EvenMerge}\,(p) \\ &\leq 16p\alpha + 16\frac{n}{p}\beta \end{aligned}$$

Et donc :

$$T(n,p) = \mathcal{O}\left(\frac{n\log n}{p} + \sqrt{p} + \frac{n}{\sqrt{p}}\right)$$

Tri d'un tableau réparti sur un *Mesh* 3D

« Facile » : tri 2D (xz), 2D (yz), 2D (xy zigzag), 1D (y), 2D (xy)



$$T \approx \mathcal{O}\left(\frac{n \log n}{p} + \sqrt[3]{p} + \frac{n}{\sqrt[3]{p}}\right)$$