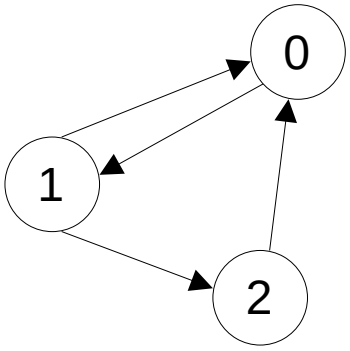
# PageRank




Matrice d'adjacence A

| | | |
|---|---|---|
| 0 | 1 | 0 |
| 1 | 0 | 1 |
| 1 | 0 | 0 |

Matrice d'adjacence transposée $A^T$

| | | |
|---|---|---|
| 0 | 1 | 1 |
| 1 | 0 | 0 |
| 0 | 1 | 0 |

Matrice P = $(A^T)$ normalisée sur les colonnes
= (A normalisée sur les lignes)$^T$

| | | |
|---|---|---|
| 0 | 0,5 | 1 |
| 1 | 0 | 0 |
| 0 | 0,5 | 0 |

q vecteur résultat de longueur n
« nombre de nœuds dans le graphe »
Initialement :

| | (exemple ici) |
|---|---|
| 1/n | 1/3 |
| 1/n | 1/3 |
| . | 1/3 |
| . | |
| 1/n | |

A chaque itération du PageRank :

$$q = (P \cdot q) \times \beta + \frac{norme(q) \times (1-\beta)}{n}$$

L'opération délicate (en parallèle) est le produit matrice-vecteur $P \cdot q$

Il y a cependant plusieurs manières d'appliquer le PageRank, c'est ce qui est présenté par la suite.

Explication détaillée de la Version 5, dans un cas où la grille de processus n’est pas carrée

## PageRank Version 5

PageRank appliqué à la matrice d’adjacence transposée (non normalisée), avec vecteur résultat réparti sur les processus.

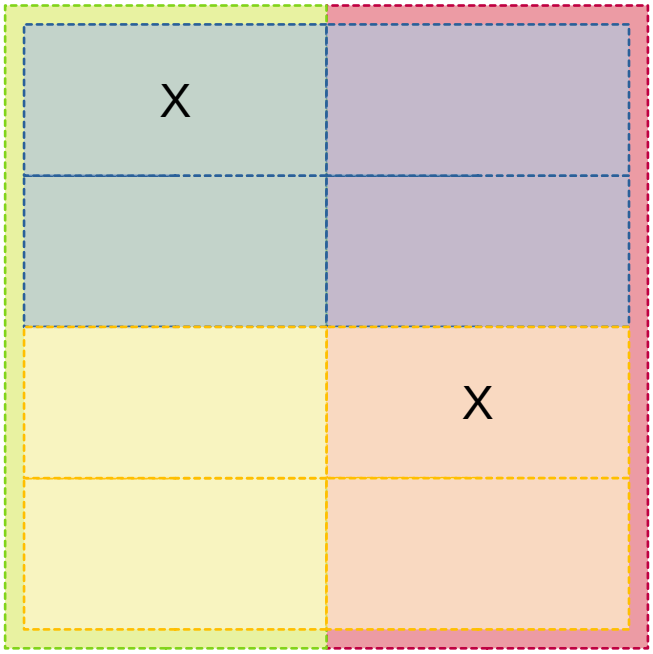
# PageRank Version 5

Groupe de calcul et groupe de besoin - exemple

Vecteur q

Résultat

Matrice binaire $A^T$




Groupe de calcul bleu
(calcule la partie bleue du vecteur résultat)

Groupe de calcul jaune
(calcule la partie jaune du vecteur résultat)

**Groupe de besoin mauve**
**(a besoin de la partie mauve du vecteur q pour effectuer les calculs)**

**Groupe de besoin vert**
**(a besoin de la partie verte du vecteur q pour effectuer les calculs)**

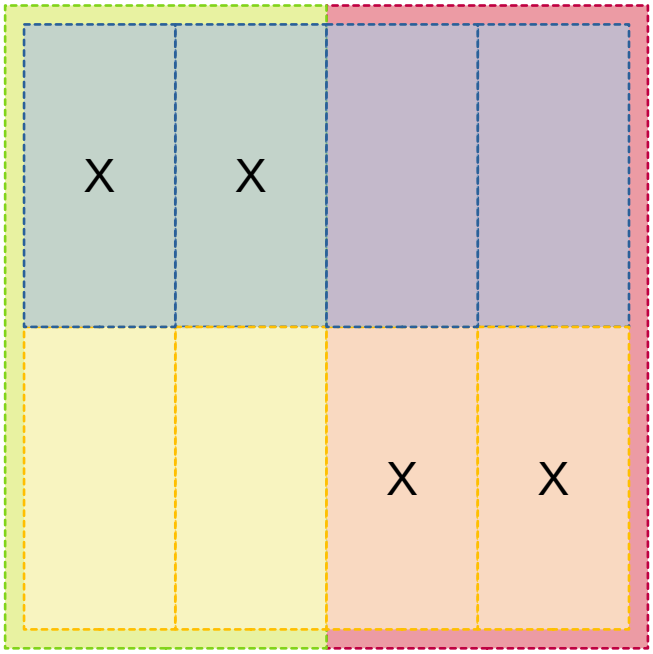
# PageRank Version 5

Groupe de calcul et groupe de besoin - exemple

Vecteur q

Résultat

Matrice binaire $A^T$




Groupe de calcul bleu
(calcule la partie bleue du vecteur résultat)

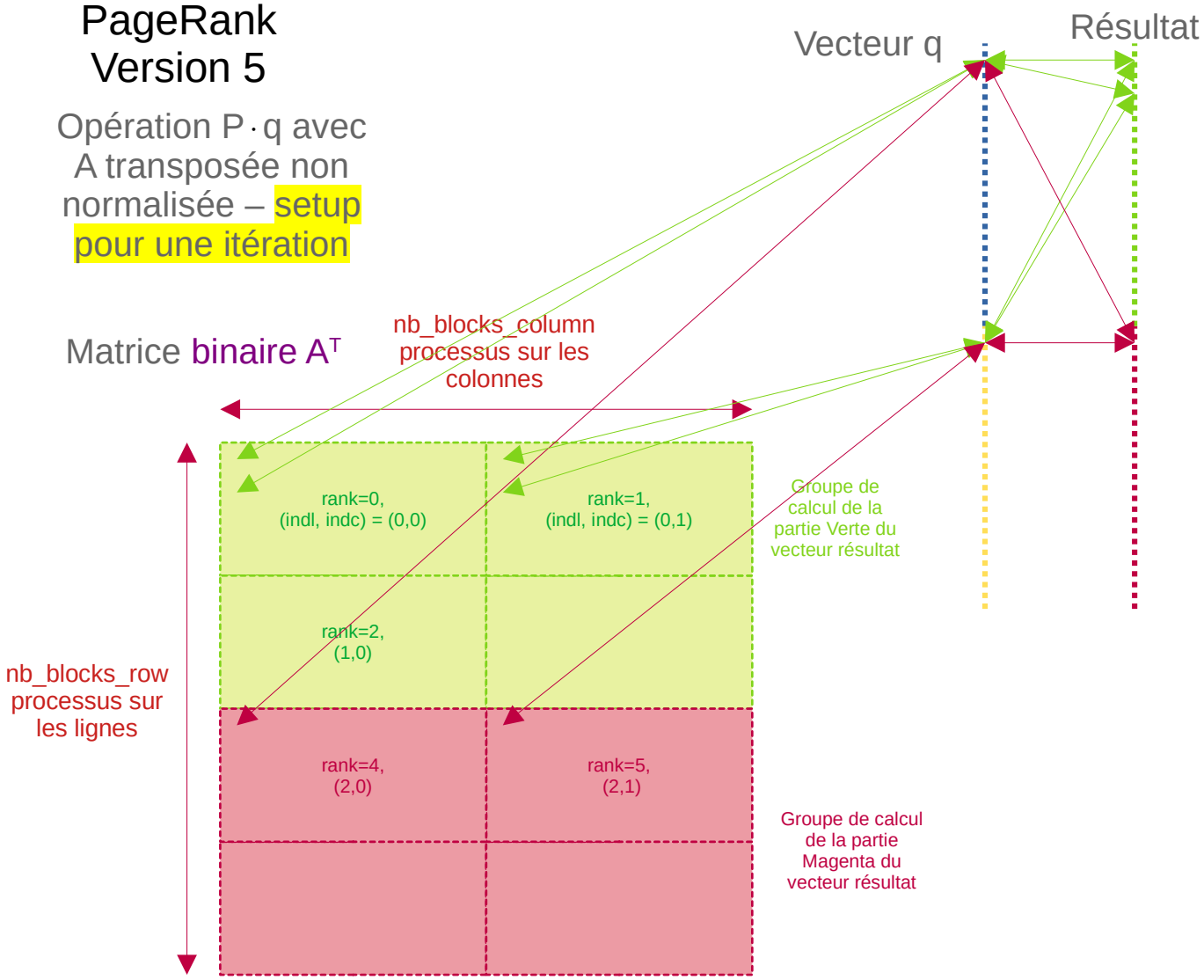
Groupe de calcul jaune
(calcule la partie jaune du vecteur résultat)

**Groupe de besoin mauve**
**(a besoin de la partie mauve du vecteur q pour effectuer les calculs)**

**Groupe de besoin vert**
**(a besoin de la partie verte du vecteur q pour effectuer les calculs)**

# PageRank Version 5

Opération $P \cdot q$ avec A transposée non normalisée – setup pour une itération




**Variables utiles pour l'algorithme :**

Facteur de dimensions de la grille :

grid_dim_factor = nb_blocks_column / nb_blocks_row

Taille du vecteur q / résultat local (en nombre de blocks sur les lignes/colonnes) :

local_result_vector_size_**row**_blocks = nb_blocks_row / pgcd(nb_blocks_row, nb_blocks_column)

local_result_vector_size_**column**_blocks = nb_blocks_column / pgcd(nb_blocks_row, nb_blocks_column)

Indice de groupe de calcul du vecteur résultat (choix du groupe) :

result_vector_calculation_group = *partie entière de* indl / local_result_vector_size_row_blocks

Indice de ligne du block dans le groupe de calcul du vecteur résultat :

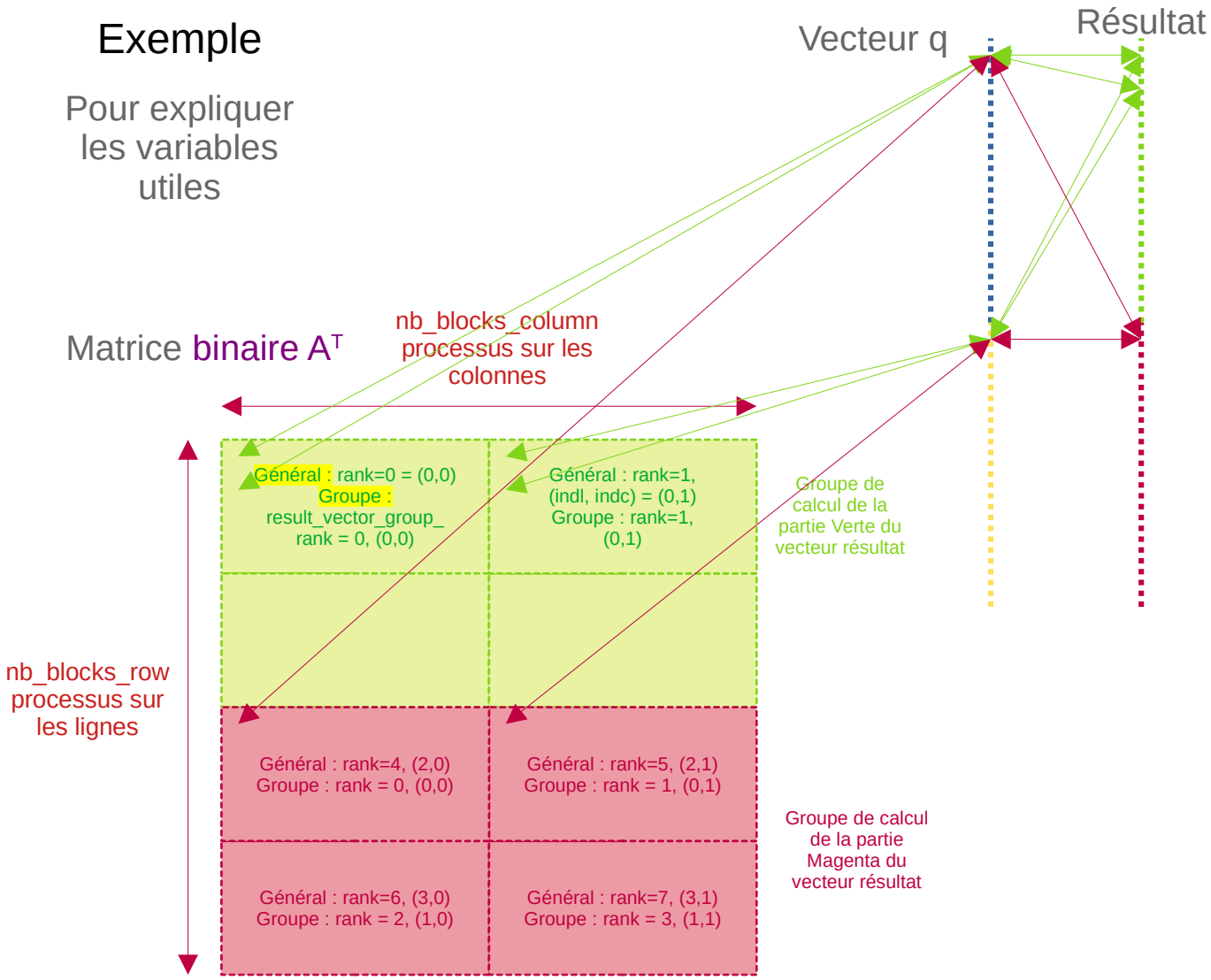
indl_in_result_vector_calculation_group = indl % (modulo) local_result_vector_size_row_blocks

Rank dans le groupe de calcul du vecteur résultat :

my_result_vector_calculation_group_rank = indl_in_result_vector_calculation_group + indc * local_result_vector_size_row_blocks

# Exemple

Pour expliquer les variables utiles




## Variables utiles pour l'algorithme :

Facteur de dimensions de la grille :

grid_dim_factor = nb_blocks_column / nb_blocks_row = 2/4 = 0,5

Taille du vecteur q / résultat local (en nombre de blocks sur les lignes/colonnes) :

local_result_vector_size_**row**_blocks = nb_blocks_row / pgcd(nb_blocks_row, nb_blocks_column)
= 4 / (pgcd(2,4) = 4/2 = 2 blocks de ligne

local_result_vector_size_**column**_blocks = nb_blocks_column / pgcd(nb_blocks_row, nb_blocks_column)
= 2 / (pgcd(2,4) = 2/2 = 1 block de colonne

Indice de groupe de calcul du vecteur résultat (choix du groupe) :

result_vector_calculation_group = *partie entière de* indl / local_result_vector_size_row_blocks

indl = 0 ou 1, *partie entière de* (0/2) = E(0/2) = 0 ; E(1/2) = 0

indl = 2 ou 3, *partie entière de* (2/2) = E(2/2) = 1 ; E(3/2) = 1

Indice de ligne du block dans le groupe de calcul du vecteur résultat :

indl_in_result_vector_calculation_group = indl % (modulo) local_result_vector_size_row_blocks

indl = 0, 0 modulo 2 = 0
indl = 1, 1 modulo 2 = 1
indl = 2, 2 modulo 2 = 0
indl = 3, 3 modulo 2 = 1

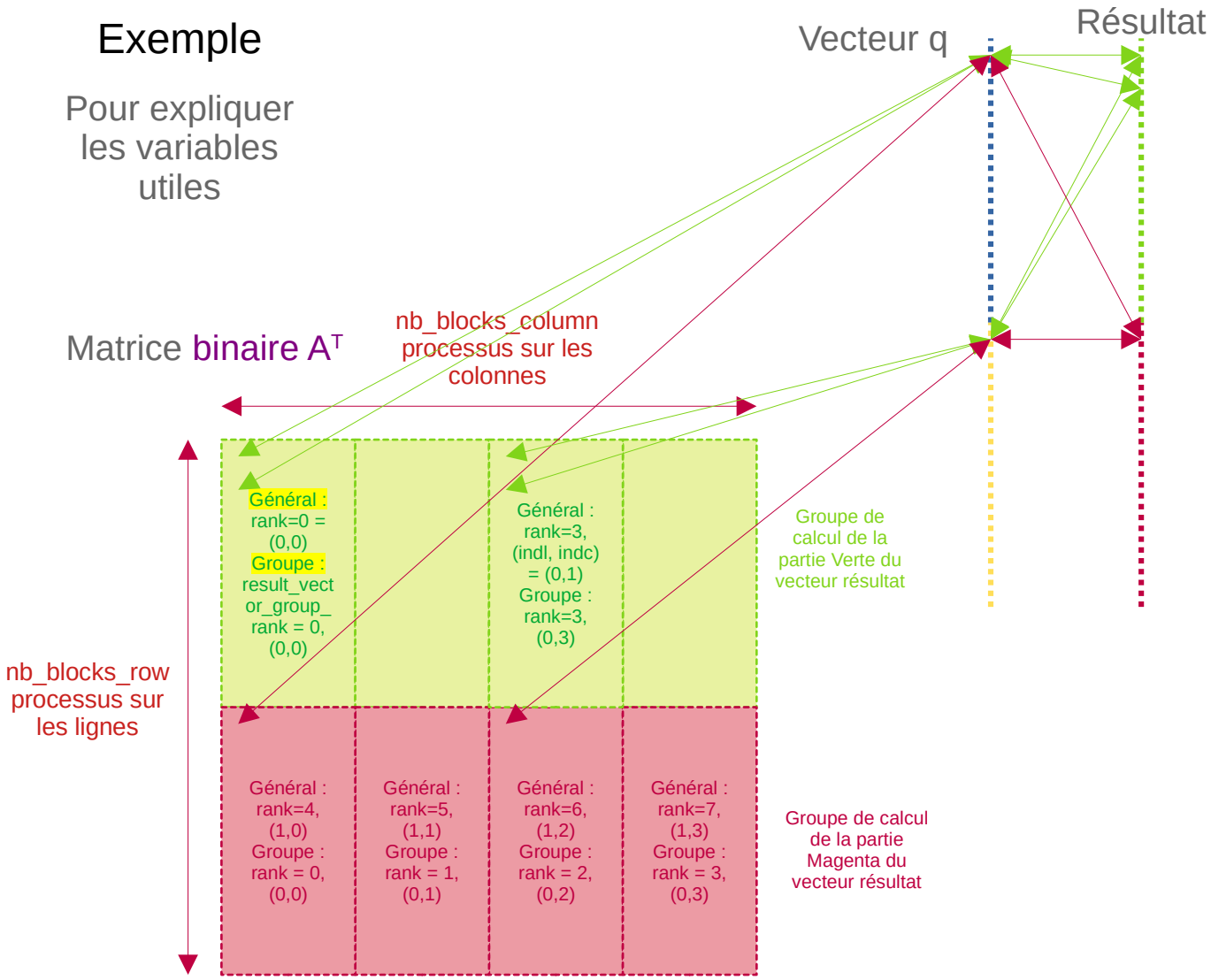
Rank dans le groupe de calcul du vecteur résultat :

my_result_vector_calculation_group_rank = indl_in_result_vector_calculation_group * local_result_vector_size_row_blocks + indc

exemple pour rank général = 7
**my_result_vector_group_calculation_rank** = 1*2 + 1 = 3

# Exemple

Pour expliquer les variables utiles




## Variables utiles pour l'algorithme :

Facteur de dimensions de la grille :

grid_dim_factor = nb_blocks_column / nb_blocks_row = 4/2 = 2

Taille du vecteur q / résultat local (en nombre de blocks sur les lignes/colonnes) :

local_result_vector_size_**row**_blocks = nb_blocks_row / pgcd(nb_blocks_row, nb_blocks_column)
= 2 / (pgcd(2,4) = 2/2 = 1 block de ligne

local_result_vector_size_**column**_blocks = nb_blocks_column / pgcd(nb_blocks_row, nb_blocks_column)
= 4 / (pgcd(2,4) = 4/2 = 2 blocks de colonne

Indice de groupe de calcul du vecteur résultat (choix du groupe) :

result_vector_calculation_group = *partie entière de* indl / local_result_vector_size_row_blocks

indl = 0 , *partie entière de* (0/1) = 0
indl = 1, *partie entière de* (1/1) = 1

Indice de ligne du block dans le groupe de calcul du vecteur résultat :

indl_in_result_vector_calculation_group = indl % (modulo) local_result_vector_size_row_blocks

indl = 0, 0 modulo 1 = 0
indl = 1, 1 modulo 1 = 0

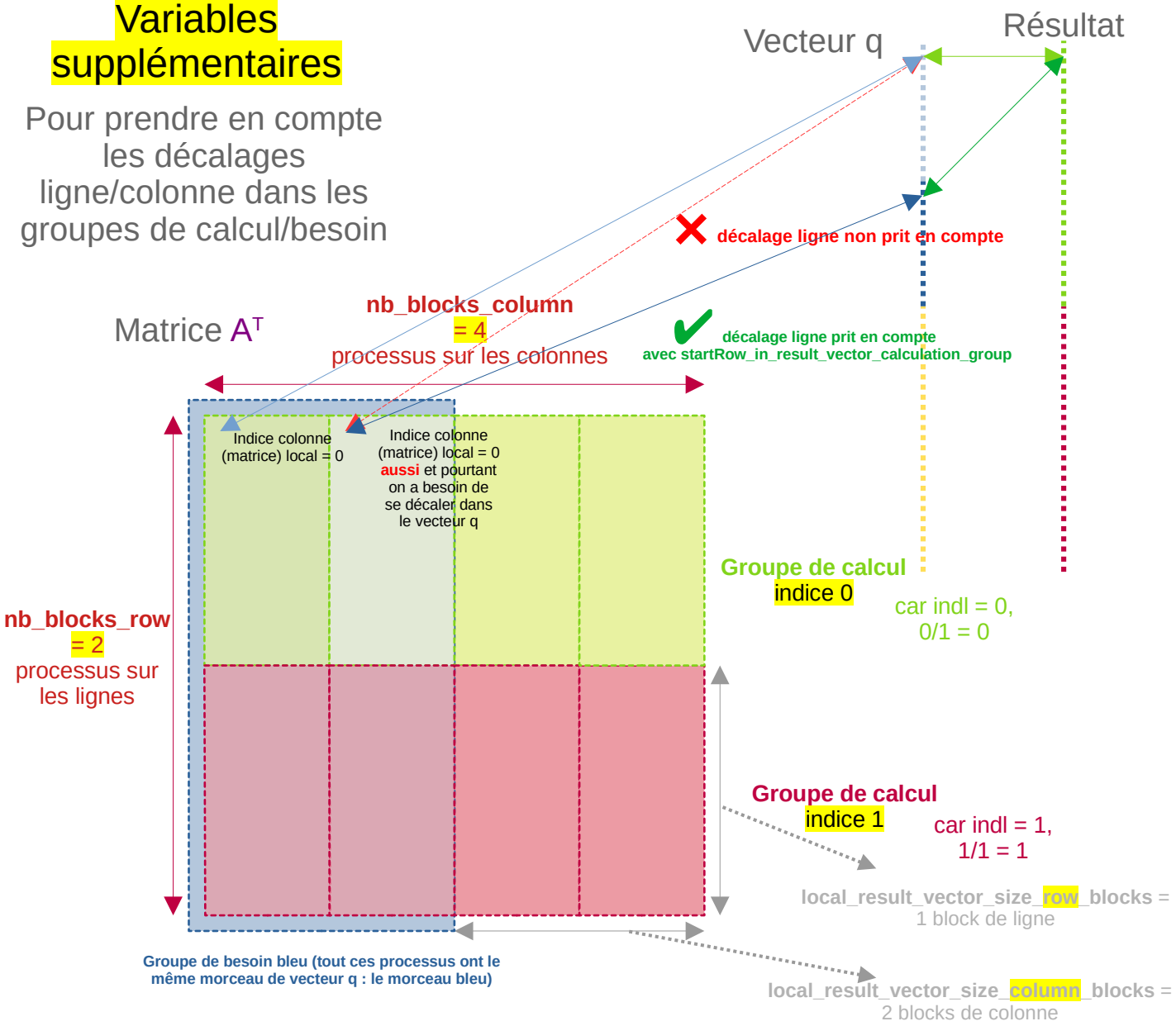
Rank dans le groupe de calcul du vecteur résultat :

my_result_vector_calculation_group_rank = indl_in_result_vector_calculation_group * local_result_vector_size_row_blocks + indc

exemple pour rank général = 6
**my_result_vector_group_calculation_rank** = 0*1 + 2 = 2

# Variables supplémentaires

Pour prendre en compte les décalages ligne/colonne dans les groupes de calcul/besoin




**grid_dim_factor** = nb_blocks_row / nb_blocks_column = 2/4 = 0,5

**local_result_vector_size_row_blocks** = nb_blocks_row / pgcd(nb_blocks_row, nb_blocks_column)
= 2 / (pgcd(2,4) = 2/2 = 1 blocks de ligne

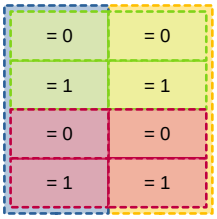
Dans le cas où on a moins de processus sur les colonnes que sur les lignes, on a le même morceau de vecteur q sur plusieurs lignes de processus. On a besoin (pour chaque ligne de processus) de prendre en compte ceci pour savoir quelle partie du vecteur résultat on calcul.
Dans le cas où on a plus de processus sur les colonnes que sur les lignes, on a le même morceau de vecteur q dans plusieurs colonnes. On a besoin (pour chaque colonne de processus) de prendre en compte ceci pour savoir où aller prendre les valeurs dans le vecteur q..
C'est pourquoi on introduit quelques variables supplémentaires.

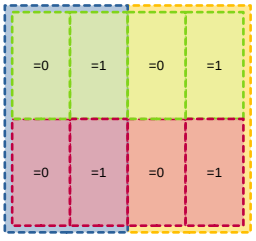
## Variables utiles supplémentaires pour l'algorithme :

Indice de départ en colonne dans le groupe de calcul :
**startRow_in_result_vector_calculation_group** =
nb_ligne * indl_in_result_vector_calculation_group

Indice de départ en colonne dans le groupe de calcul :
**startColumn_in_result_vector_calculation_group** =
nb_colonne * (indc *modulo* local_result_vector_size_column_blocks)

Les 0 calculent la première partie du morceau de vecteur résultat, et les 1 la deuxième partie du morceau de vecteur résultat
Pareil côté mauve

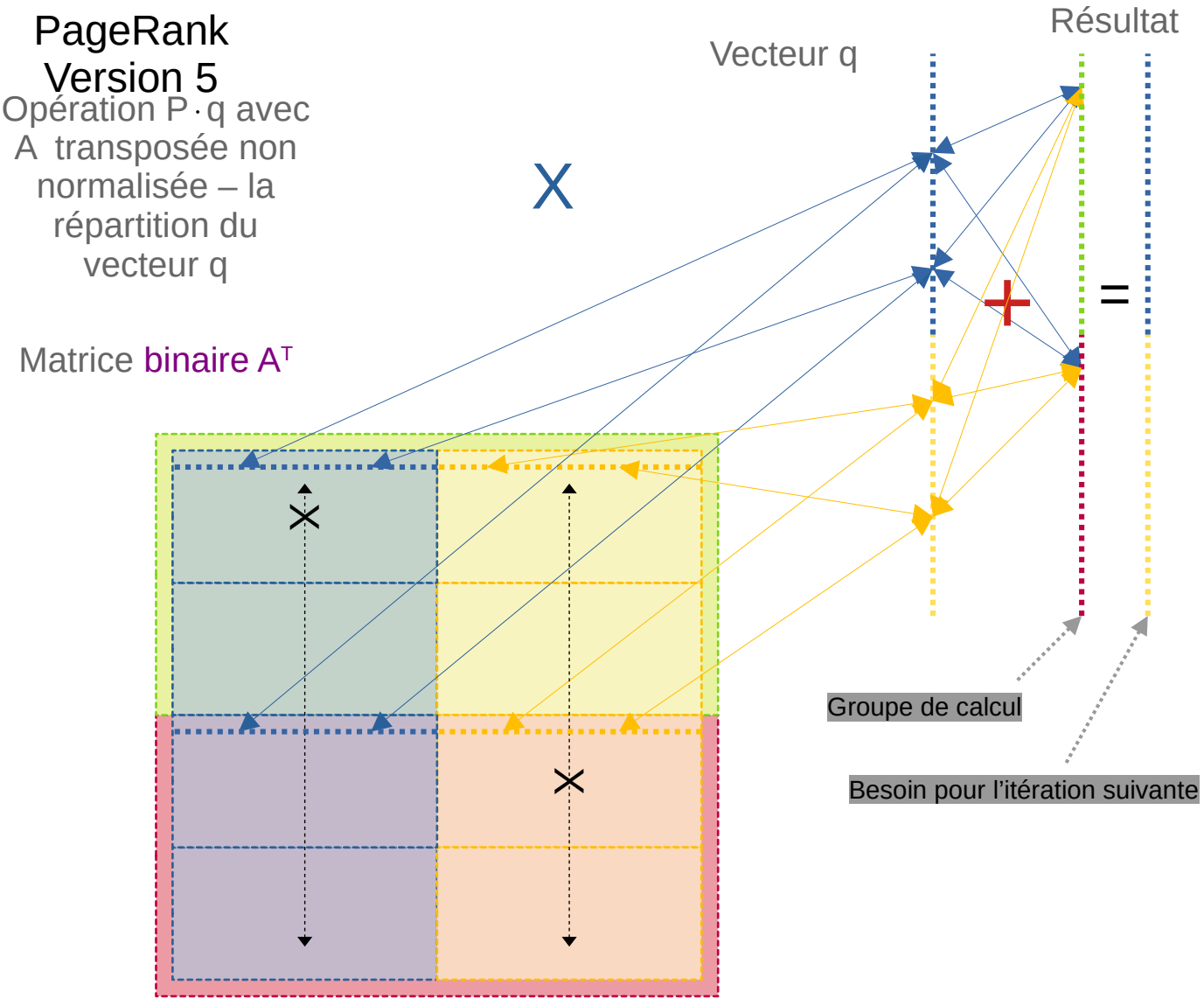



Les 0 vont chercher la valeur du vecteur q dans la première partie du morceau de q, les 1 dans la deuxième partie (décalage en colonne prit en compte)
Pareil côté jaune

# PageRank Version 5

Opération $P \cdot q$ avec A transposée non normalisée – la répartition du vecteur q




Les blocs bleus ont besoin de la partie bleue du **vecteur q** pour calculer, et les blocs jaunes ont besoin de la partie jaune

**Problème :**

Les groupes de calcul vert et magenta ont chacun une partie du vecteur (à la fin d'une itération). Une communication correct sur les lignes est nécessaire pour se préparer pour l'itération suivante.

Il faut choisir, pour chaque ligne, les processus qui serviront de root pour la communication (broadcast) du morceau de vecteur résultat

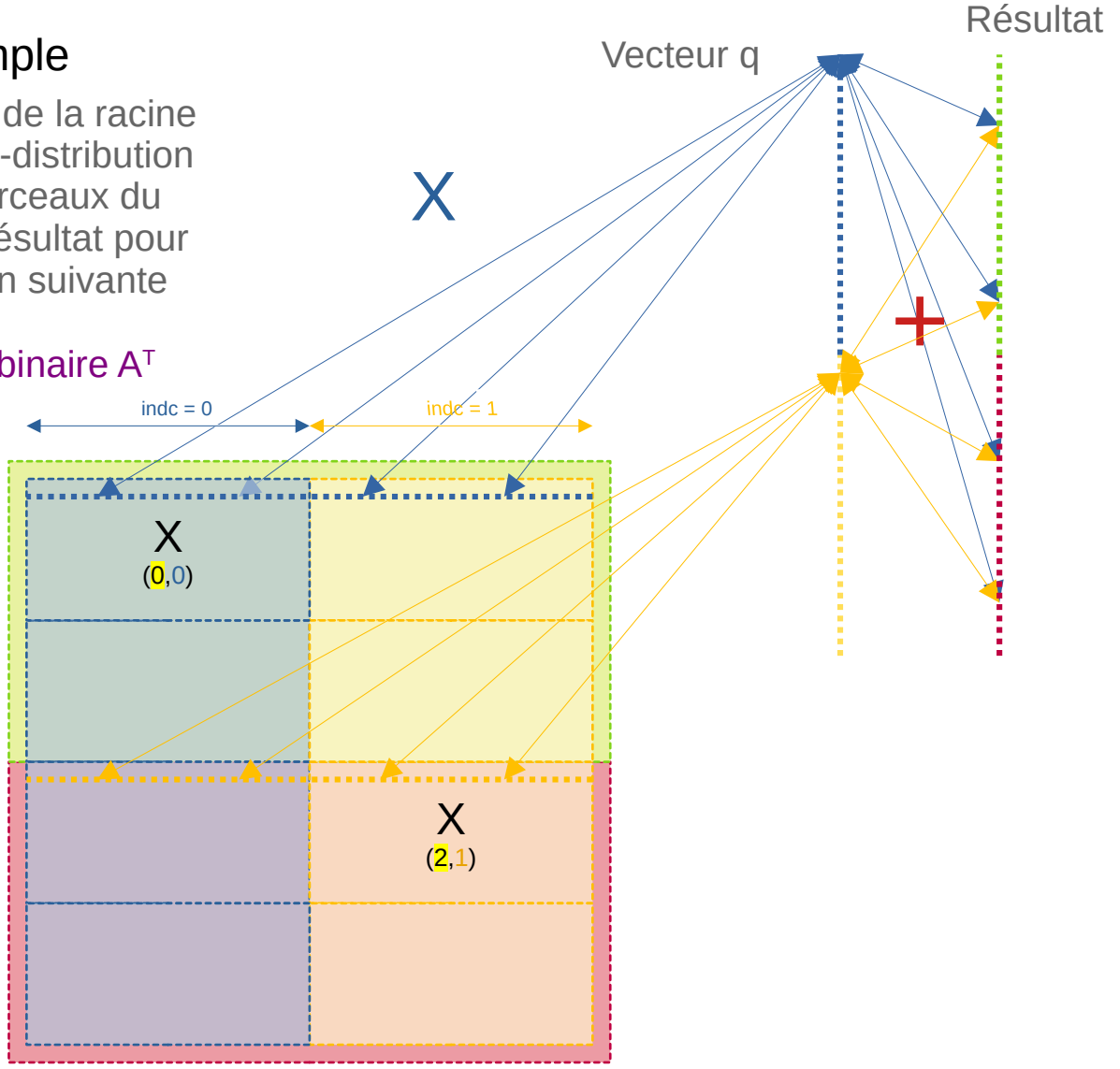
**pr_result_redistribution_root** =
*partie entière de* indc / grid_dim_factor

**pr_result_redistribution_root** est l'indl (indice de ligne) du processus qui servira de racine pour le broadcast

# Exemple

De calcul de la racine pour la re-distribution des morceaux du vecteur résultat pour l'itération suivante

Matrice binaire $A^T$




Les blocs bleus ont besoin de la partie bleue du **vecteur q** pour calculer, et les blocs jaunes ont besoin de la partie jaune

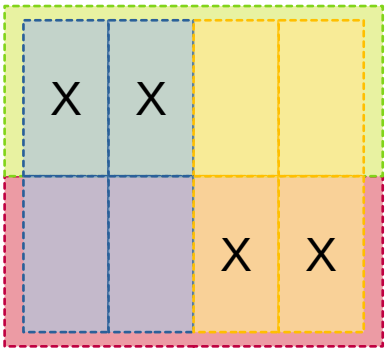
**grid_dim_factor** =
nb_blocks_column / nb_blocks_row
= 2/4 = 0,5

Choix des processus qui serviront de root pour la communication par ligne (broadcast) du morceau de vecteur résultat :

**pr_result_redistribution_root** =
*partie entière de* indc / grid_dim_factor

Première colonne (indc = 0) de la grille : root = E(0/0,5) = 0
Deuxième colonne (indc = 1) de la grille : root = E(1/0,5) = 2

(autre cas)



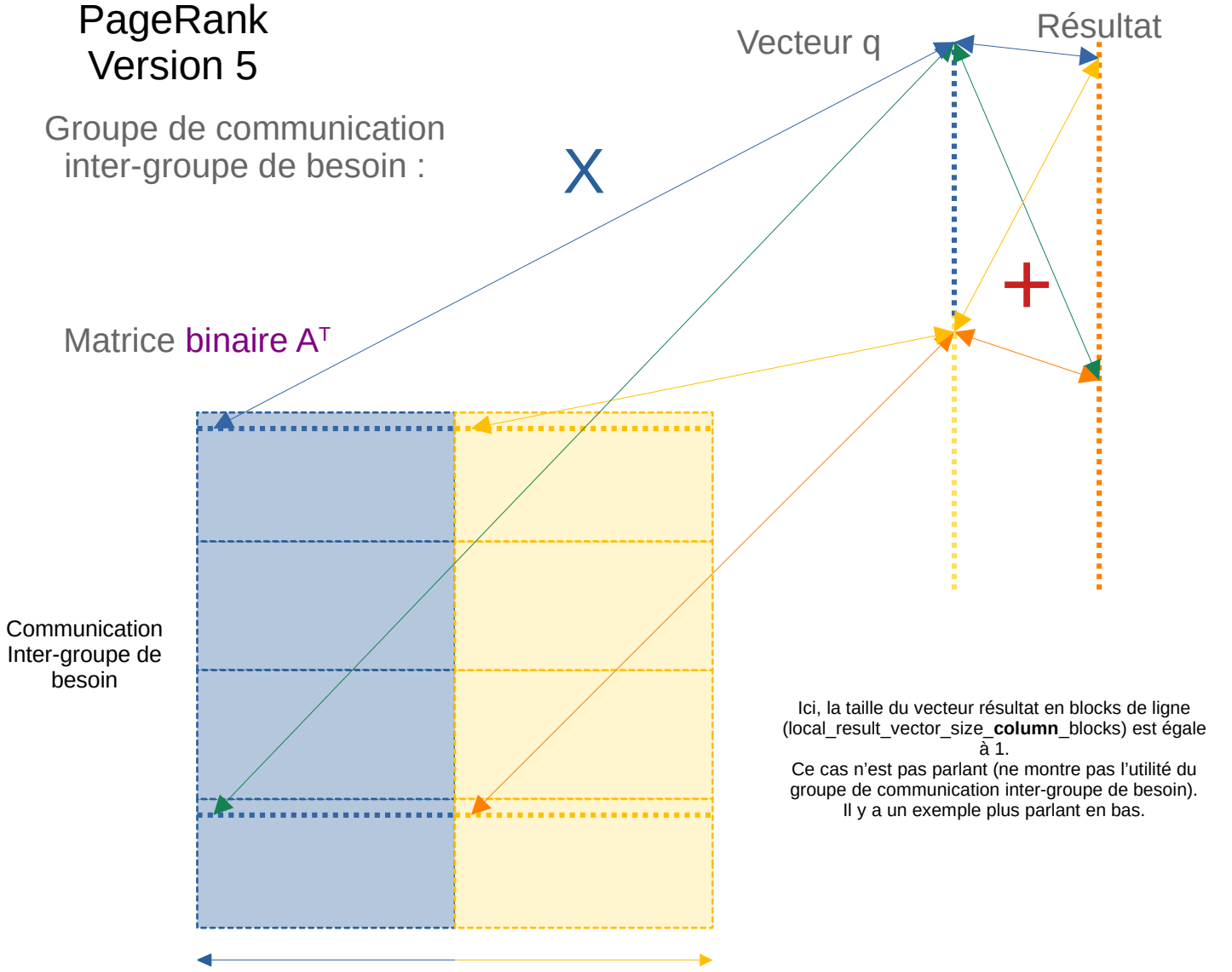

**grid_dim_factor** =
nb_blocks_column / nb_blocks_row
= 4/2 = 2

(indc = 0) : root = E(0/2) = 0
(indc = 1) : root = E(1/2) = 0
(indc = 2) : root = E(2/2) = 1
(indc = 3) : root = E(3/2) = 1

**indl** du processus qui servira de racine pour le broadcast

# PageRank Version 5

Groupe de communication inter-groupe de besoin :




Ici, la taille du vecteur résultat en blocks de ligne (local_result_vector_size_**column**_blocks) est égale à 1.
Ce cas n'est pas parlant (ne montre pas l'utilité du groupe de communication inter-groupe de besoin).
Il y a un exemple plus parlant en bas.

Pour pouvoir normaliser le vecteur résultat ou récupérer le vecteur résultat complet, il faut un communicateur spécifique sur les colonnes : le « communicateur inter-groupe de besoin »

**inter_result_vector_need_group_communicaton_group** =
( indc modulo local_result_vector_size_**column**_blocks ) * nb_blocks_row + indl

## Exemple

Choix de l'indice de groupe de communication inter-groupe de besoin de vecteur résultat :

local_result_vector_size_**row**_blocks = 2
local_result_vector_size_**column**_blocks = 1
nb_blocks_row = 4

| | |
|---|---|
| 0 | 0 |
| 1 | 1 |
| 2 | 2 |
| 3 | 3 |

(0 modulo 1) * 4 + 0
(0 modulo 1) * 4 + 1
(0 modulo 1) * 4 + 2
(0 modulo 1) * 4 + 3

(1 modulo 1) * 4 + 0
(1 modulo 1) * 4 + 1
(1 modulo 1) * 4 + 2
(1 modulo 1) * 4 + 3

(autre cas plus parlant)

local_result_vector_size_**row**_blocks = 1
local_result_vector_size_**column**_blocks = 2
nb_blocks_row = 2

| | | | |
|---|---|---|---|
| 0 | 2 | 0 | 2 |
| 1 | 3 | 1 | 3 |

(0 modulo 2) * 2 + 0
(1 modulo 2) * 2 + 0

(2 modulo 2) * 2 + 0
(3 modulo 2) * 2 + 0

(0 modulo 2) * 2 + 1
(1 modulo 2) * 2 + 1

(2 modulo 2) * 2 + 1
(3 modulo 2) * 2 + 1

# Déroulé du PageRank Version 5

Étape par étape

La partie intéressante du PageRank est la boucle interne (où on fait les itérations).

Choix du **beta**, **epsilon** (erreur max)
Initialisation du **vecteur q**, et de la **somme totale de q**, et de l'**erreur** (pour entrer dans la boucle interne)

Tant que (erreur > epsilon) :
    …
Fin Tant que

Vecteur résultat q = Classement PageRank

Nous allons nous intéresser à ce qui se passe dans la boucle interne « Tant que »

## Déroulé du PageRank Version 5

Étape par étape

Tant que (erreur > epsilon) :

- Calcul du vecteur q, de manière répartie (en réalité : calcul de morceaux du vecteur q dans chaque processus)

- Communication des nouveaux morceaux du vecteur q aux processus qui en ont besoin (groupes de besoin), pour l'itération suivante

- Normalisation du nouveau vecteur q (besoin de la somme totale)

- Calcul de l'erreur (nouveau vecteur q - ancien vecteur q)

Fin Tant que

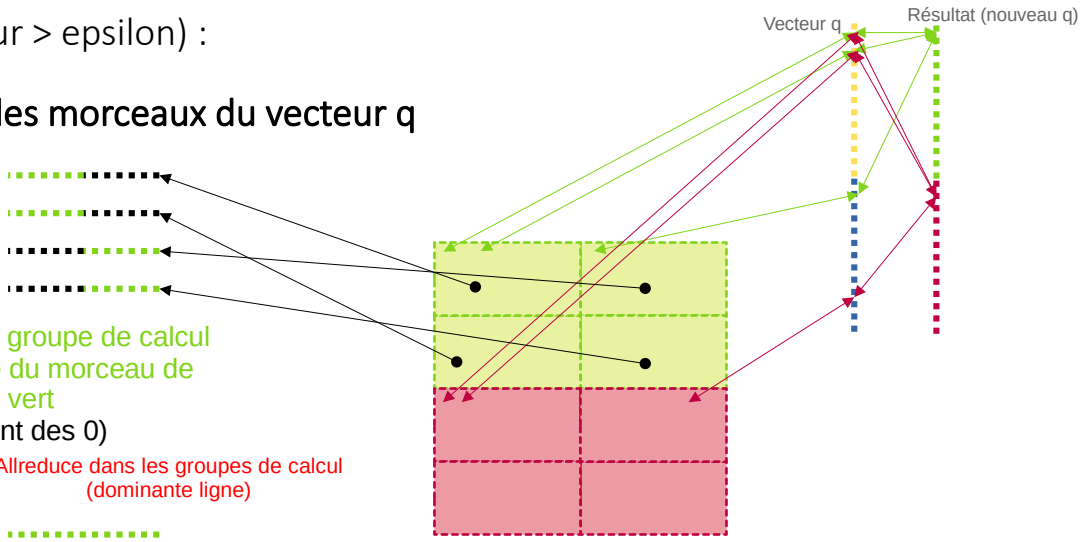
Nous allons expliquer, avec l'aide de tout ce que nous avons défini dans les pages précédentes, le dérouler de ces étapes

**Déroulé du PageRank Version 5** (effectué avec $A^T$ non normalisée)

Étape par étape

**Tant que** (erreur > epsilon) :

- Calcul des morceaux du vecteur q

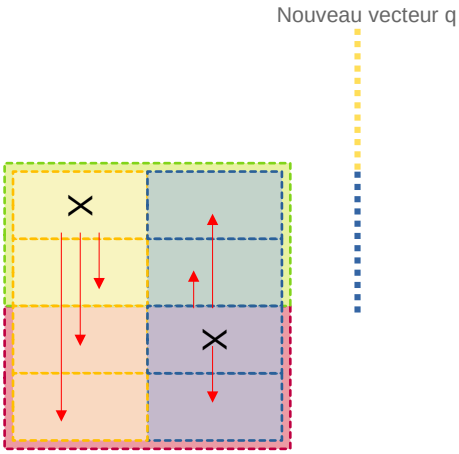



Chaque processus du groupe de calcul vert calcul une partie du morceau de vecteur q vert
(en noir ce sont des 0)

Allreduce dans les groupes de calcul (dominante ligne)

Morceau de q vert complet dans tout les processus vert

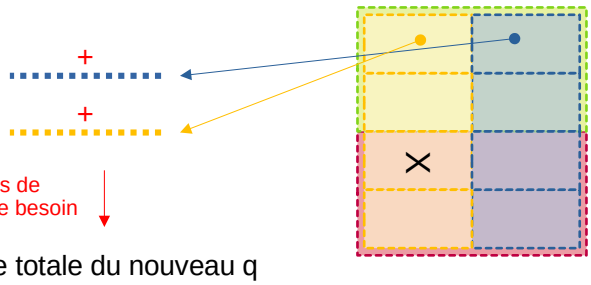
- Communication des nouveaux morceaux du vecteur q aux processus qui en ont besoin (groupes de besoin), pour l'itération suivante




Les processus racine (''X'') communiquent leurs morceau du nouveau vecteur q aux autres de la même colonne

Broadcast sur les colonnes

- Normalisation du nouveau vecteur q (besoin de la somme totale)




Somme des éléments du morceau de q local

Allreduce dans les groupes de communication inter-groupe de besoin (dominante ligne)

Somme totale du nouveau q

Division de chaque élément du nouveau vecteur q par la somme totale

- Calcul de l'erreur (nouveau vecteur q - ancien vecteur q)

Somme locale des éléments du vecteur q – ancien q

Allreduce dans les groupes de communication inter-groupe de besoin (dominante ligne)

Somme totale = erreur

**Fin Tant que**