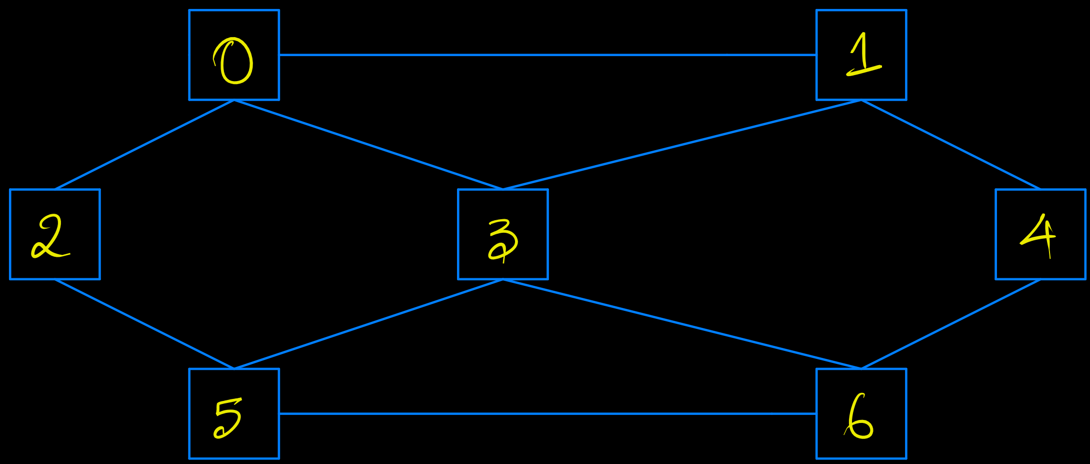
Mapping Techniques for Load Balancing

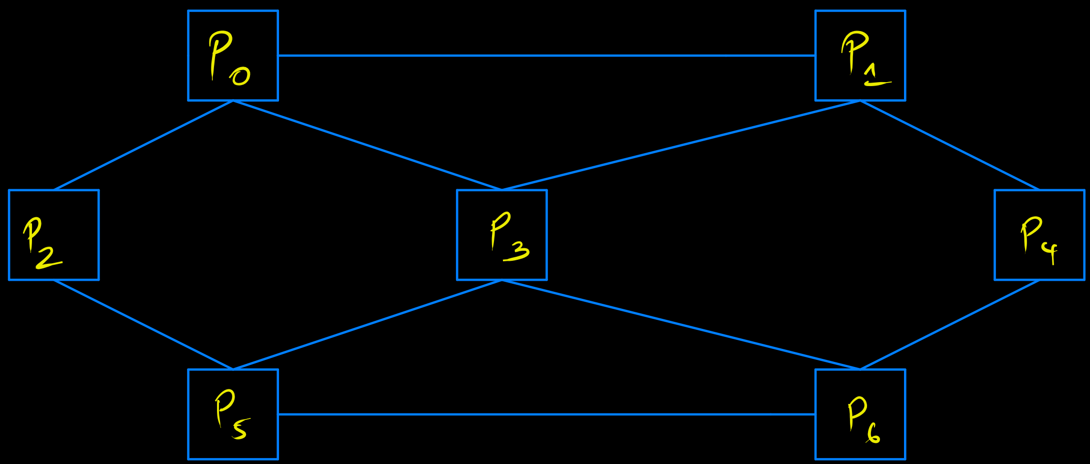
Two objectives:

→ Reduce amount of time in process Interaction

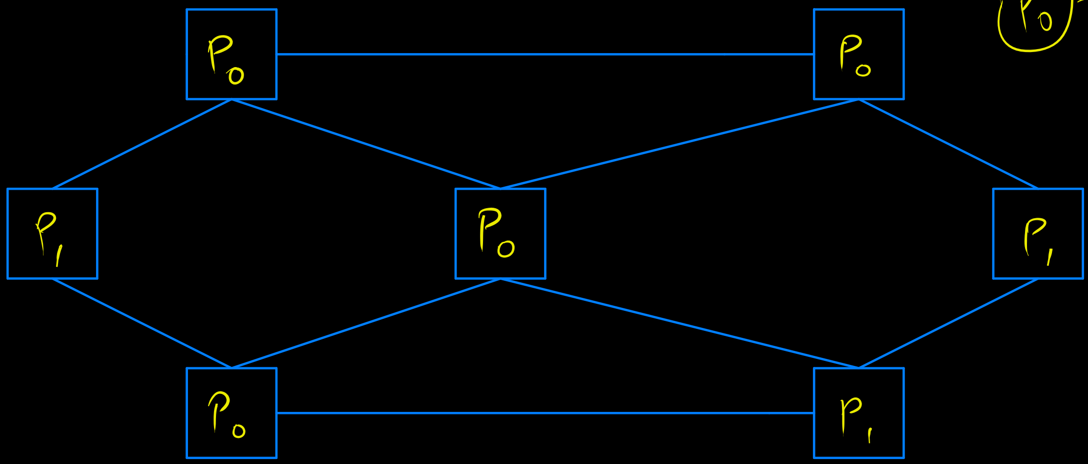
→ Reduce total idle time of Processes
(while others are engaged in performing tasks)

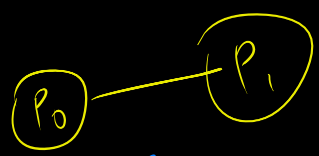


Task-Interaction Graph
(Assume No Dependency)



Zero Idle time
Process Interaction Same as Task Interaction?

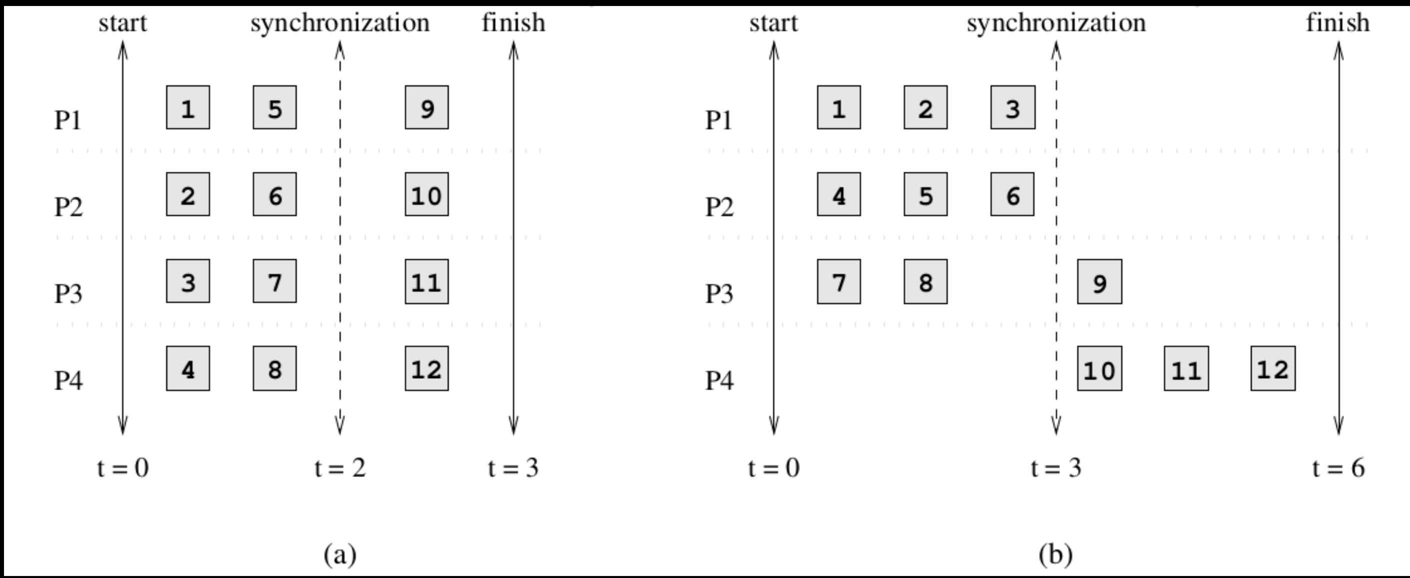




Reduced Process Interaction.
Idle time?

⇒ Optimal Mapping balances Computations and Interactions.
(Non-Trivial problem)

Task Dependency also plays a Critical role



(9 - 12 depend on Completion of 1 - 8)

Static	Dynamic
→ Maps Tasks to processes prior to execution of the algorithm	→ Map Tasks to processes during the execution of the algorithm
→ Can be applied only when Tasks are statically generated	→ Can be applied for both Statically or dynamically generated Tasks
→ Good Mapping depends on knowledge of Task Sizes, Data Sizes, inter-task interaction etc.	→ If Task Sizes are unknown dynamic mapping is more effective
→ Optimal Mapping for non-Uniform Tasks is not easy (NP-Complete)	→ May involve data movement at runtime for better data distribution

→ But inexpensive heuristics provide acceptable approximate solutions.

→ Generally, easier to design and program

→ Usually, more complicated, particularly in the Message-Passing paradigm

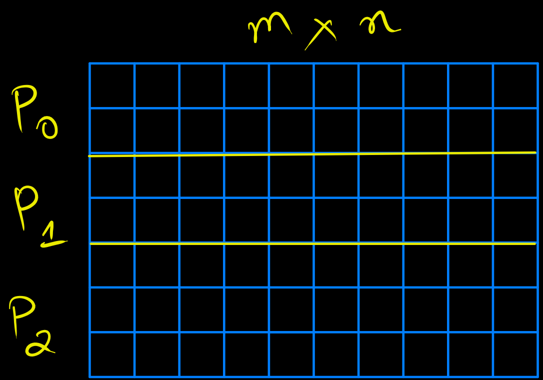
Static Mapping

Mapping based on Data partitioning

Mapping based on Task partitioning

[Data partitioning leads to Task Decomposition + Mapping]

I. Block Distributions: Assign uniform contiguous portions (Tasks) of an array to different Processes.



1×10

$p = 3$

$\frac{m}{p} = 2$

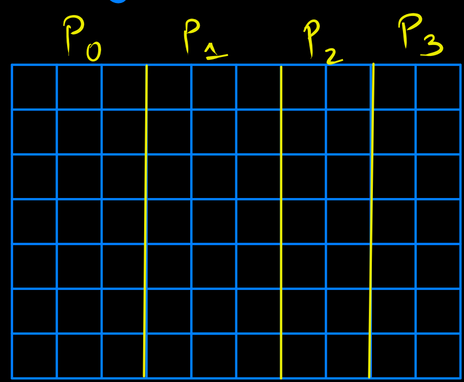
(Row-wise)

$k\,m/p$ to $(k+1)\,m/p$ (for $k = 0$ to $p-1$)

$0 \le x < 2$

$2 \le x < 4$

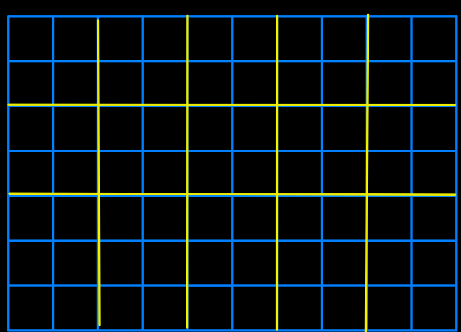
$4 \le x < 6$



$\frac{n}{p} = 2$

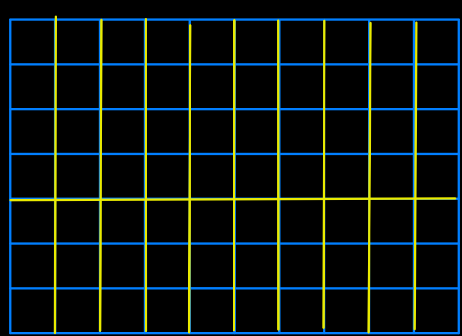
(Column-wise)

$k\,n/p$ to $(k+1)\,n/p$ (for $k = 0$ to $p-1$)



$3 \times 5 = 15$

$\frac{1}{3} \times \frac{10}{5}$



$p = 20$

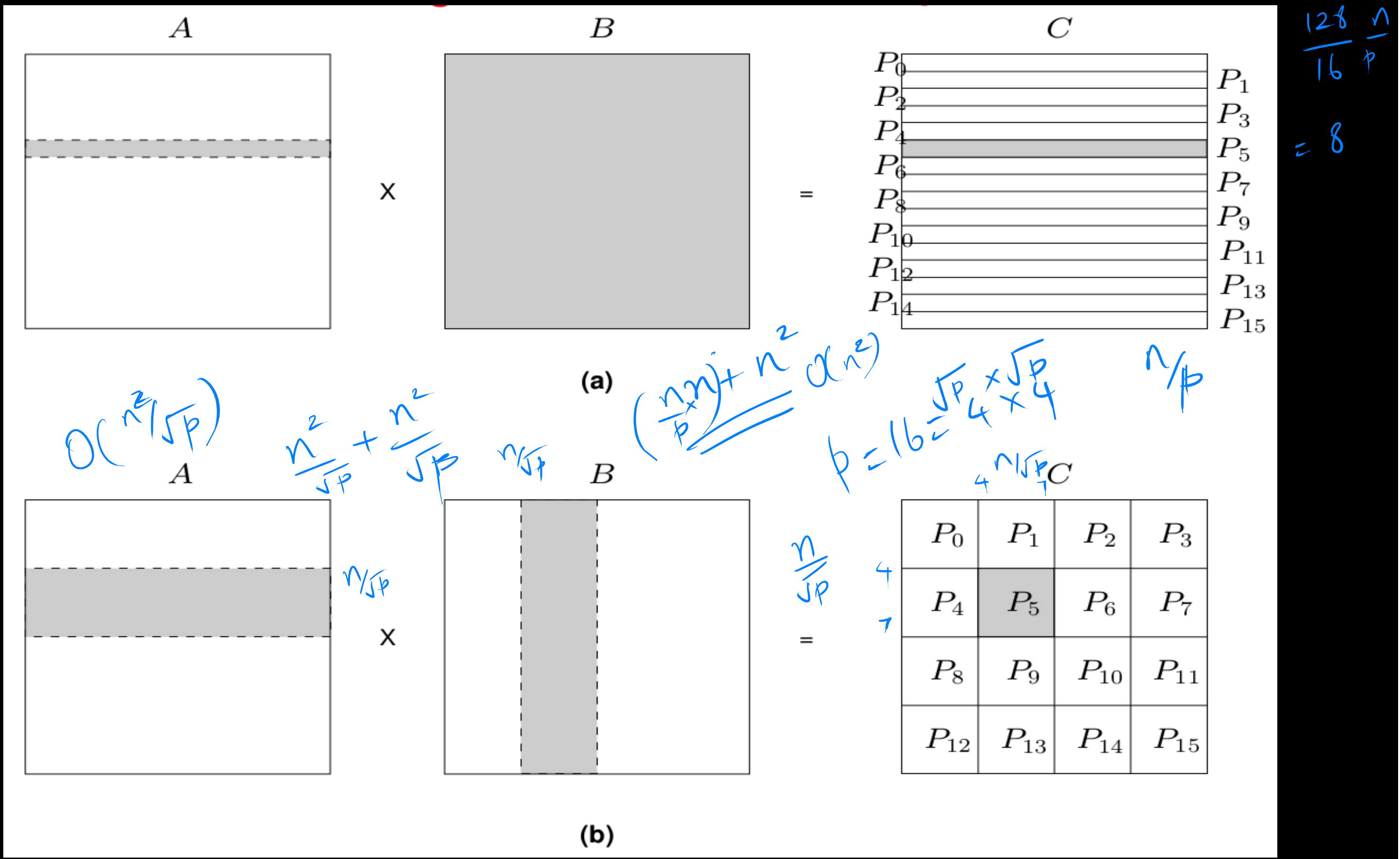
$p_1 \times p_2 = 2 \times 10$

$\frac{1}{2} \times \frac{10}{10}$

$\boxed{3 \times 1}$

$m/p_1 \times n/p_2$, $p_1 \times p_2 = p$

Let A, B & C be $n \times n$ matrices $\ni AB = C$



Two ways to decompose :

<u>One-dimensional</u>	<u>Two-dimensional</u>
Set of rows (block) of C to One Process	One block (2d) of c to One Process
Each Process gets n/p rows	Each Process gets $\frac{n}{\sqrt{p}} \times \frac{n}{\sqrt{p}}$ size block
Maximum $\underline{\underline{n}}$ Processes	Maximum $\underline{\underline{n^2}}$ processes
Lower degree of Concurrency	Higher degree of Concurrency
Higher Interaction between processes	Lower Interaction between Processes
(Data accessed : $\frac{n^2}{p} + n^2 : O(n^2)$)	(Data accessed : $\frac{n^2}{\sqrt{p}} + \frac{n^2}{\sqrt{p}} : O(\frac{n^2}{\sqrt{p}})$)

II. Cyclic & Block-cyclic Distributions:

Partition the array into many more blocks than the number of processes

Assign the partitions (Tasks) to processes in a round-robin manner so that each process gets several non-adjacent blocks.

LU Factorization: $A = LU$

Lower Triangular, 1's on the diagonal

Upper triangular

Algorithm: A is an $n \times n$ matrix

```
for (k = 0 ; k < n ; k++)
{
    for (j = k+1 ; j < n ; j++)
        A[j][k] = A[j][k]/A[k][k];

    for (j = k+1 ; j < n ; j++)
        for (i = k+1 ; i < n ; i++)
            A[i][j] = A[i][j] - A[i][k] * A[k][j];

    //After this iteration A[k+1 : n-1][k] → kth Column of L
                           A[k][k : n-1] → kth row of U
}
```

$$A = \begin{bmatrix} 2 & 1 & 1 \\ 4 & -6 & 0 \\ -2 & 7 & 2 \end{bmatrix} \xrightarrow[R_3 \leftarrow R_3 + R_1]{R_2 \leftarrow R_2 - 2R_1} \begin{bmatrix} 2 & 1 & 1 \\ 0 & -8 & -2 \\ 0 & 8 & 3 \end{bmatrix} \xrightarrow{R_3 \leftarrow R_3 + R_2} \begin{bmatrix} 2 & 1 & 1 \\ 0 & -8 & -2 \\ 0 & 0 & 1 \end{bmatrix}$$

$$U = \begin{bmatrix} 2 & 1 & 1 \\ 0 & -8 & -2 \\ 0 & 0 & 1 \end{bmatrix} \qquad L = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ -1 & -1 & 1 \end{bmatrix}$$

$$k = 0 \Rightarrow \begin{bmatrix} 2 & 1 & 1 \\ 4 & -6 & 0 \\ -2 & 7 & 2 \end{bmatrix} \xrightarrow{\text{I. for}} \begin{bmatrix} 2 & 1 & 1 \\ 2 & -6 & 0 \\ -1 & 7 & 2 \end{bmatrix}$$

II. for $j=1,\ i=1$ $\quad A_{11} \leftarrow A_{11} - A_{10}A_{01}$

$$\begin{bmatrix} 2 & 1 & 1 \\ 2 & -8 & 0 \\ -1 & 8 & 2 \end{bmatrix} \xleftarrow[j=1,\ i=2]{\text{II. for}} \begin{bmatrix} 2 & 1 & 1 \\ 2 & -8 & 0 \\ -1 & 7 & 2 \end{bmatrix}$$

$A_{21} \leftarrow A_{21} - A_{20}A_{01}$

II. for
$j=2,\ i=1$
$A_{12} \leftarrow A_{12} - A_{10}A_{02}$

$$\begin{bmatrix} 2 & 1 & 1 \\ 2 & -8 & -2 \\ -1 & 8 & 2 \end{bmatrix} \xrightarrow[j=2,\ i=2]{\text{II. for}} \begin{bmatrix} 2 & 1 & 1 \\ 2 & -8 & -2 \\ -1 & 8 & 3 \end{bmatrix}$$

$A_{22} \leftarrow A_{22} - A_{20}A_{02}$

0^{th} col of L : $\begin{bmatrix} 1 \\ 2 \\ -1 \end{bmatrix}$ $\quad$ 0^{th} row of U : $\begin{bmatrix} 2 & 1 & 1 \end{bmatrix}$

$$k = 1 \Rightarrow \begin{bmatrix} 2 & 1 & 1 \\ 2 & -8 & -2 \\ -1 & 8 & 3 \end{bmatrix} \xrightarrow{\text{I. for}} \begin{bmatrix} 2 & 1 & 1 \\ 2 & -8 & -2 \\ -1 & -1 & 3 \end{bmatrix}$$

II. for
$j=2,\ i=2$
$A_{22} \leftarrow A_{22} - A_{21}A_{12}$

$$\begin{bmatrix} 2 & 1 & 1 \\ 2 & -8 & -2 \\ -1 & -1 & 1 \end{bmatrix}$$

1^{st} col of L : $\begin{bmatrix} 0 \\ 1 \\ -1 \end{bmatrix}$

1^{st} row of U : $\begin{bmatrix} 0 & -8 & -2 \end{bmatrix}$

$k = 2 \Rightarrow$ Nothing to be done

2^{nd} col of L : $\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$ $\quad 2^{nd}$ row of U : $\begin{bmatrix} 0 & 0 & 1 \end{bmatrix}$

$A = LU \rightarrow$ indices from $1 \rightarrow$

$$\begin{bmatrix} L_{11} & 0 & 0 \\ L_{21} & L_{22} & 0 \\ L_{31} & L_{32} & L_{33} \end{bmatrix} \begin{bmatrix} U_{11} & U_{12} & U_{13} \\ 0 & U_{22} & U_{23} \\ 0 & 0 & U_{33} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{bmatrix}$$

$A_{11} = L_{11} U_{11} \Rightarrow$ Fix $L_{11} = 1 \Rightarrow U_{11} = L_{11}^{-1} A_{11}$

$A_{21} = L_{21} U_{11} \Rightarrow L_{21} = A_{21} U_{11}^{-1}$

$A_{31} = L_{31} U_{11} \Rightarrow L_{31} = A_{31} U_{11}^{-1}$

$A_{12} = L_{11} U_{12} \Rightarrow U_{12} = L_{11}^{-1} A_{12}$ (Compute needed if $L_{11} \neq 1$)

$A_{13} = L_{11} U_{13} \Rightarrow U_{13} = L_{11}^{-1} A_{13}$ (Compute needed if $L_{11} \neq 1$)

$A_{22} = L_{21} U_{12} + L_{22} U_{22} \Rightarrow A_{22} = A_{22} - L_{21} U_{12}$

and $U_{22} = L_{22}^{-1} A_{22}$

$A_{32} = L_{31} U_{12} + L_{32} U_{22} \Rightarrow A_{32} = A_{32} - L_{31} U_{12}$

and $L_{32} = U_{22}^{-1} A_{32}$

$A_{23} = L_{21} U_{13} + L_{22} U_{23} \Rightarrow A_{23} = A_{23} - L_{21} U_{13}$

and $U_{23} = L_{22}^{-1} A_{23}$

$A_{33} = L_{31} U_{13} + L_{32} U_{23} + L_{33} U_{33}$

$\Rightarrow A_{33} = A_{33} - L_{31} U_{13}$

$A_{33} = A_{33} - L_{32} U_{23}$

$U_{33} = L_{33}^{-1} A_{33}$

$$\begin{pmatrix} A_{1,1} & A_{1,2} & A_{1,3} \\ A_{2,1} & A_{2,2} & A_{2,3} \\ A_{3,1} & A_{3,2} & A_{3,3} \end{pmatrix} \rightarrow \begin{pmatrix} L_{1,1} & 0 & 0 \\ L_{2,1} & L_{2,2} & 0 \\ L_{3,1} & L_{3,2} & L_{3,3} \end{pmatrix} \cdot \begin{pmatrix} U_{1,1} & U_{1,2} & U_{1,3} \\ 0 & U_{2,2} & U_{2,3} \\ 0 & 0 & U_{3,3} \end{pmatrix}$$

1: $A_{1,1} \rightarrow L_{1,1}U_{1,1}$
2: $L_{2,1} = A_{2,1}U_{1,1}^{-1}$
3: $L_{3,1} = A_{3,1}U_{1,1}^{-1}$
4: $U_{1,2} = L_{1,1}^{-1}A_{1,2}$
5: $U_{1,3} = L_{1,1}^{-1}A_{1,3}$
6: $A_{2,2} = A_{2,2} - L_{2,1}U_{1,2}$
7: $A_{3,2} = A_{3,2} - L_{3,1}U_{1,2}$
8: $A_{2,3} = A_{2,3} - L_{2,1}U_{1,3}$
9: $A_{3,3} = A_{3,3} - L_{3,1}U_{1,3}$
10: $A_{2,2} \rightarrow L_{2,2}U_{2,2}$
11: $L_{3,2} = A_{3,2}U_{2,2}^{-1}$
12: $U_{2,3} = L_{2,2}^{-1}A_{2,3}$
13: $A_{3,3} = A_{3,3} - L_{3,2}U_{2,3}$
14: $A_{3,3} \rightarrow L_{3,3}U_{3,3}$

If we consider a 3×3, 2d block distribution, to compute respective elements, then we map all Tasks associated with 1×1 in A to a process (total 9).

$\Rightarrow$ Leads to Load Imbalance

P_0 T_1	P_3 T_4	P_6 T_5
P_1 T_2	P_4 T_6 T_{10}	P_7 T_8 T_{12}
P_2 T_3	P_5 T_7 T_{11}	P_8 T_9 T_{13} T_{14}

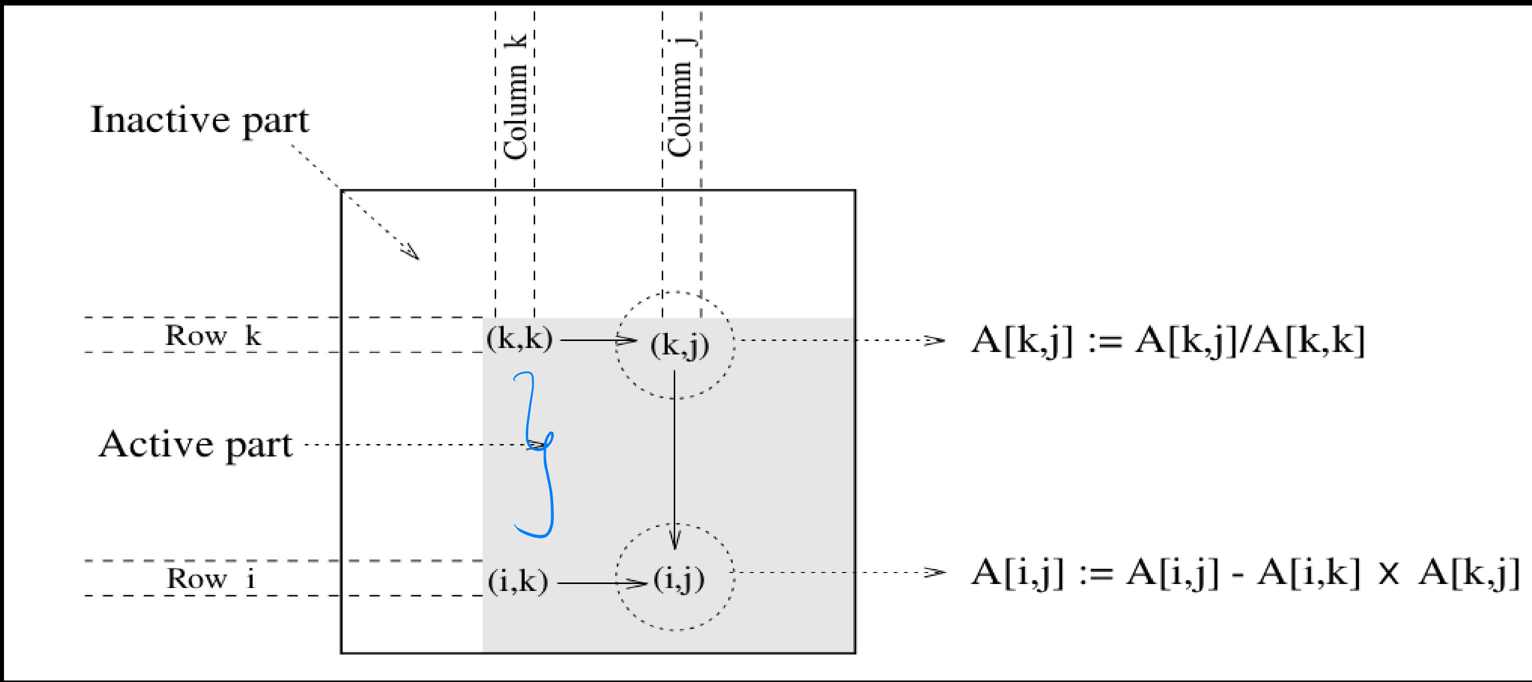
Computing final value of A_{11} needs only 1 Task. (Task 1)

But A_{33} needs 3 Tasks (Task 9, Task 13 & Task 14)

Also processes can be idle due to Task Dependency.

$T_7 \rightarrow T_{11}$ and $T_4 \rightarrow T_7$

⇒ Processes assigned to left columns & top rows perform much less work compared to processes that are assigned later rows and columns.



$\therefore$ Partition the array into many more blocks than the number of processes.

Assign the partitions (associated Tasks) to processes in a round-robin manner so that each process gets several non-adjacent blocks.

1d Block cyclic : $n \times n$ matrix, p processes

Rows of the matrix are divided into αp groups, where $1 \le \alpha \le n/p$, and each group (blocks) has $n/\alpha p$ consecutive rows.

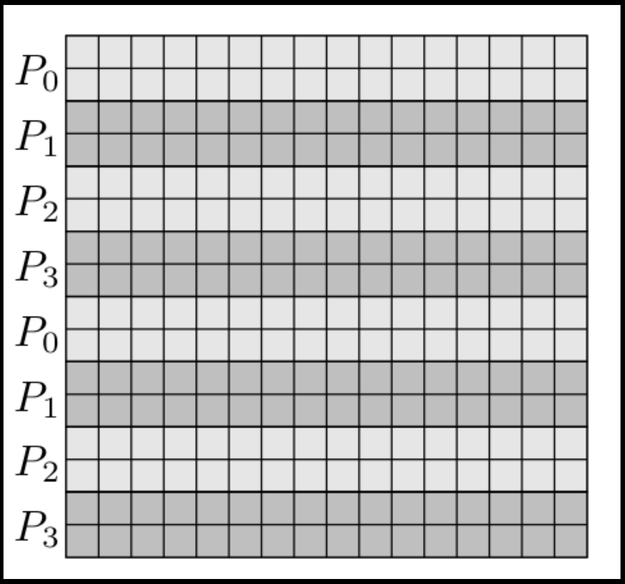
Assign each block to a process in a wrap-around fashion $\rightarrow$ b_i to $P_{i \% p}$

$\Rightarrow$ α blocks to each process and each subsequent block assigned to a process is p blocks away.

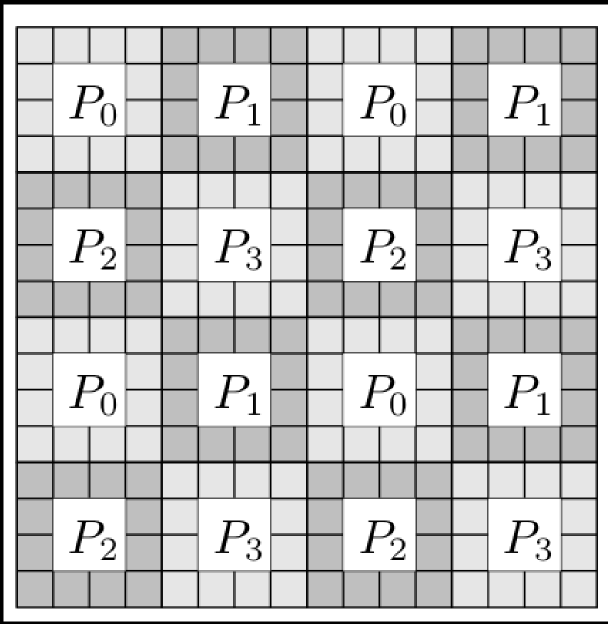
16×16

$2 \times 4 = 8$ blocks

αp



Sai



2-d Block cyclic distribution

$n \times n$, p processes $= \sqrt{p} \times \sqrt{p}$

$\downarrow$

$\alpha\sqrt{p} \times \alpha\sqrt{p}$

4×4

$\Rightarrow$ Reduces idling as processes get Tasks from all parts of the matrix

$\rightarrow$ Overall work gets balanced out

$\rightarrow$ Good chance that some Tasks are ready for execution at any given time

$\alpha = n/p$ $\Rightarrow$ 1d : Each row is a block

2d : Each element is a block ($\alpha = n/\sqrt{p}$)

$\rightarrow$ Cyclic distribution

Good load balance due to fine grained decomposition

But Performance penalties due to lack of locality

Can lead to high degree of Interaction relative to the amount of Computation

$\alpha = 1 \Rightarrow$ 1d : Each block is 4 rows

2d : Each block is 8×8 sub-matrix

→ Degenerates to Block Distribution

Maximum locality

Optimal Interaction

But load imbalance & process idling