CS 314 Principles of Programming Languages

Lecture 23: Parallelism and Dependence Analysis

Zheng (Eddy) Zhang

Rutgers University

April 30, 2018

Class Information

- Homework 8 is released.
- One extra-credit project on Scheme has been released.
- Extended TA office hours this week:
  Qiaoying: 5/1 Tuesday 7pm-9pm, HW 1, HW4, HW6, HW8.
  Qin : 5/3 Thursday 2pm - 4pm, for HW2 and HW7.
  Ari : 5/7 Monday 5:45pm-7:45pm, for HW 3 and HW5.
  (Ari's office is CoRE 334).
- My office hour before exam:
  5/5 Saturday 3pm — 5pm.
- All questions on Sakai before 5/8 Tuesday 10pm are guaranteed to be answered, within 3 days or before 5/8 midnight, whichever is sooner.

Review: Dependence Test

Given

```
do i1 = L1,U1
  ...
    do in = Ln,Un
        S1 :   A[ f1( i1, …, in), …, fm(i1,…, in) ] = …
        S2 :   …   =  A[ g1(i1, …, in), …, gm(i1, …, in) ]
```

Let α & β be a vector of n integers within the ranges of the lower and upper bounds of the n loops.

Does ∃ α, β in the loop iteration space, s.t.

$$f_k(\alpha) = g_k(\beta) \qquad \forall k, 1 \le k \le m?$$

Integer Linear Programming (ILP) for Dependence Test

Does ∃ α, β in the loop iteration space, s.t.

$$f_k(\alpha) = g_k(\beta) \qquad \forall k, 1 \le k \le m?$$

```
for (i=1; i<=100; i++)
    for (j=1; j<=100; j++){
        S1: X[i,j] = X[i,j] + Y[i-1, j];
        S2: Y[i,j] = Y[i,j] + X[i, j-1];
    }
```

Consider the two memory references:

S1(α): **X[i_1, j_1]**, S2(β): **X[i_2, j_2-1]**

α: (i_1, j_1)

β: (i_2, j_2)

Access the same memory location →

$i_1 = i_2$

$j_1 = j_2 - 1$

Loop bounds constraint →

$1 <= i_1 <= 100$

$1 <= j_1 <= 100$

$1 <= i_2 <= 100$

$1 <= j_2 <= 100$

Does there exist a solution to this integer linear programming (ILP) problem?

Integer Linear Programming (ILP) for Dependence Test

If we use the matrix vector notation <F_1, f_1, B_1, b_1> and <F_2, f_2, B_2, b_2> for two references at two iterations α: (i_1, j_1) and β: (i_2, j_2)

$i_1 = i_2$

$j_1 = j_2 - 1$

$1 <= i_1 <= 100$

$1 <= j_1 <= 100$

$1 <= i_2 <= 100$

$1 <= j_2 <= 100$

Memory reference **X[i_1, j_1]**

$$\mathbf{F_1} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad f_1 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}\begin{bmatrix} i_1 \\ j_1 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix} = \begin{bmatrix} i_1 \\ j_1 \end{bmatrix}$$

Memory reference **X[i_2, j_2-1]**

$$\mathbf{F_1}\begin{bmatrix} i_1 \\ j_1 \end{bmatrix} + f_1 = \mathbf{F_2}\begin{bmatrix} i_2 \\ j_2 \end{bmatrix} + f_2$$

$$\begin{bmatrix} i_1 \\ j_1 \end{bmatrix} = \begin{bmatrix} i_2 \\ j_2 - 1 \end{bmatrix}$$

$$\mathbf{F_2} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad f_2 = \begin{bmatrix} 0 \\ -1 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}\begin{bmatrix} i_2 \\ j_2 \end{bmatrix} + \begin{bmatrix} 0 \\ -1 \end{bmatrix} = \begin{bmatrix} i_1 \\ j_2 - 1 \end{bmatrix}$$

Integer Linear Programming (ILP) for Dependence Test

If we use the matrix vector notation <F_1, f_1, B_1, b_1> and <F_2, f_2, B_2, b_2> for two references at two iterations α: (i_1, j_1) and β: (i_2, j_2)

$i_1 = i_2$

$j_1 = j_2 - 1$

$1 <= i_1 <= 100$

$1 <= j_1 <= 100$

$1 <= i_2 <= 100$

$1 <= j_2 <= 100$

$$B_1 \begin{bmatrix} i_1 \\ j_1 \end{bmatrix} + b_1 >= 0$$

$$B_2 \begin{bmatrix} i_2 \\ j_2 \end{bmatrix} + b_2 >= 0$$

Loop bounds for $(\mathbf{i_1, j_1})$

$$B_1 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ -1 & 0 \\ 0 & -1 \end{bmatrix} \quad b_1 = \begin{bmatrix} -1 \\ -1 \\ 100 \\ 100 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \\ -1 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} i_1 \\ j_1 \end{bmatrix} + \begin{bmatrix} -1 \\ -1 \\ 100 \\ 100 \end{bmatrix} >= \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Loop bounds for $(\mathbf{i_2, j_2})$

$$B_2 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ -1 & 0 \\ 0 & -1 \end{bmatrix} \quad b_2 = \begin{bmatrix} -1 \\ -1 \\ 100 \\ 100 \end{bmatrix}$$

Putting Everything Together

If we use the matrix vector notation <F_1, f_1, B_1, b_1> and <F_2, f_2, B_2, b_2> for two references at two iterations α: (i_1, j_1) and β: (i_2, j_2)

$i_1 = i_2$
$j_1 = j_2 - 1$
$1 <= i_1 <= 100$
$1 <= j_1 <= 100$
$1 <= i_2 <= 100$
$1 <= j_2 <= 100$

$$\mathbf{F_1}\begin{bmatrix} i_1 \\ j_1 \end{bmatrix} + f_1 = \mathbf{F_2}\begin{bmatrix} i_2 \\ j_2 \end{bmatrix} + f_2$$

$$B_1\begin{bmatrix} i_1 \\ j_1 \end{bmatrix} + b_1 >= 0$$

$$B_2\begin{bmatrix} i_2 \\ j_2 \end{bmatrix} + b_2 >= 0$$

$$\mathbf{F_1} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad f_1 = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \qquad \mathbf{F_2} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad f_2 = \begin{bmatrix} 0 \\ -1 \end{bmatrix}$$

B_1, b_1, B_2, b_2 **see previous slides**

Review: Parallelizing Affine Loops

Three spaces

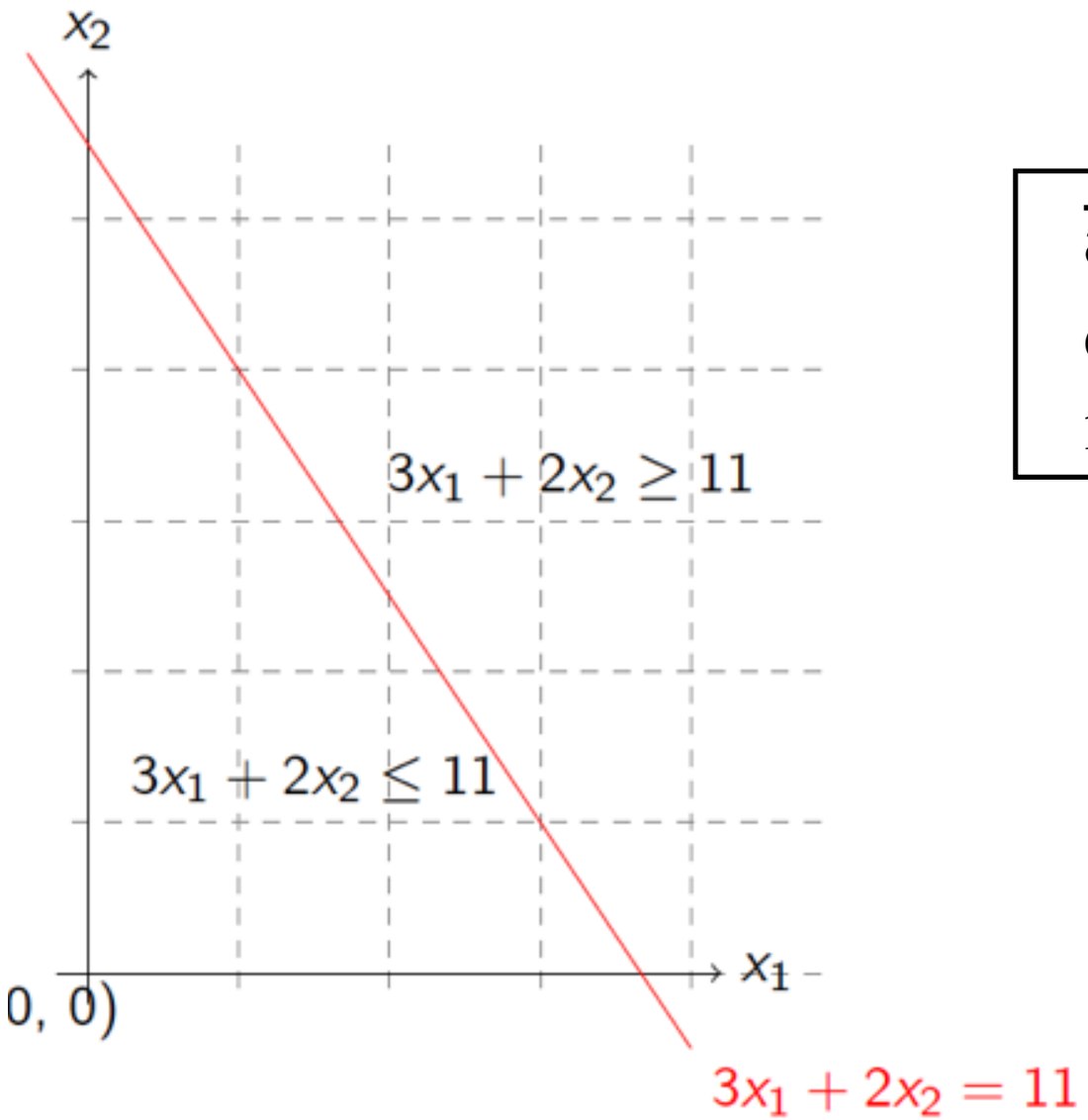
- Iteration space
  - The set of dynamic execution instances
    For instance, the set of value vectors taken by loop indices
  - A *k*-dimensional space for a *k*-level loop nest
- Data space
  - The set of array elements accessed
  - An *n*-dimensional space for an *n*-dimensional array
- Processor space
  - The set of processors in the system
  - In analysis, we may pretend there are unbounded # of virtual processors

Affine Half Space

Definition

An affine half-space of Z^d is defined as the set of points

$$\{ \vec{x} \in Z^d \mid \vec{a} * \vec{x} <= b \}$$



$\vec{a} * \vec{x} = b$ is the hyperplane that divides the d-dimensional space into two half-spaces.

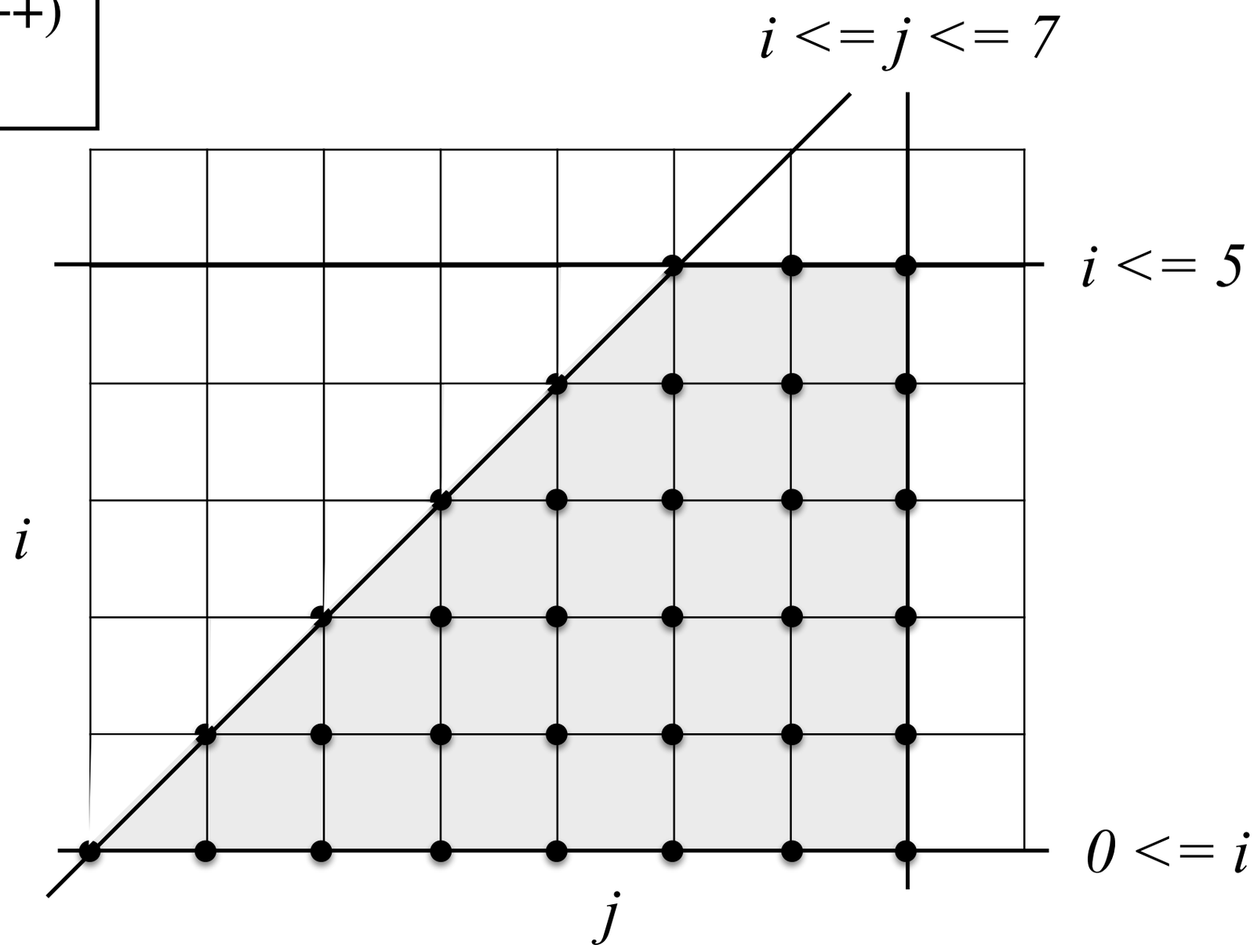
Iteration Space

- Bounded by multiple hyperplanes

```
for (i=0; i<=5; i++)
    for (j=i; j<=7; j++)
        Z[j, i] = 0;
```

$0 <= i <= 5$

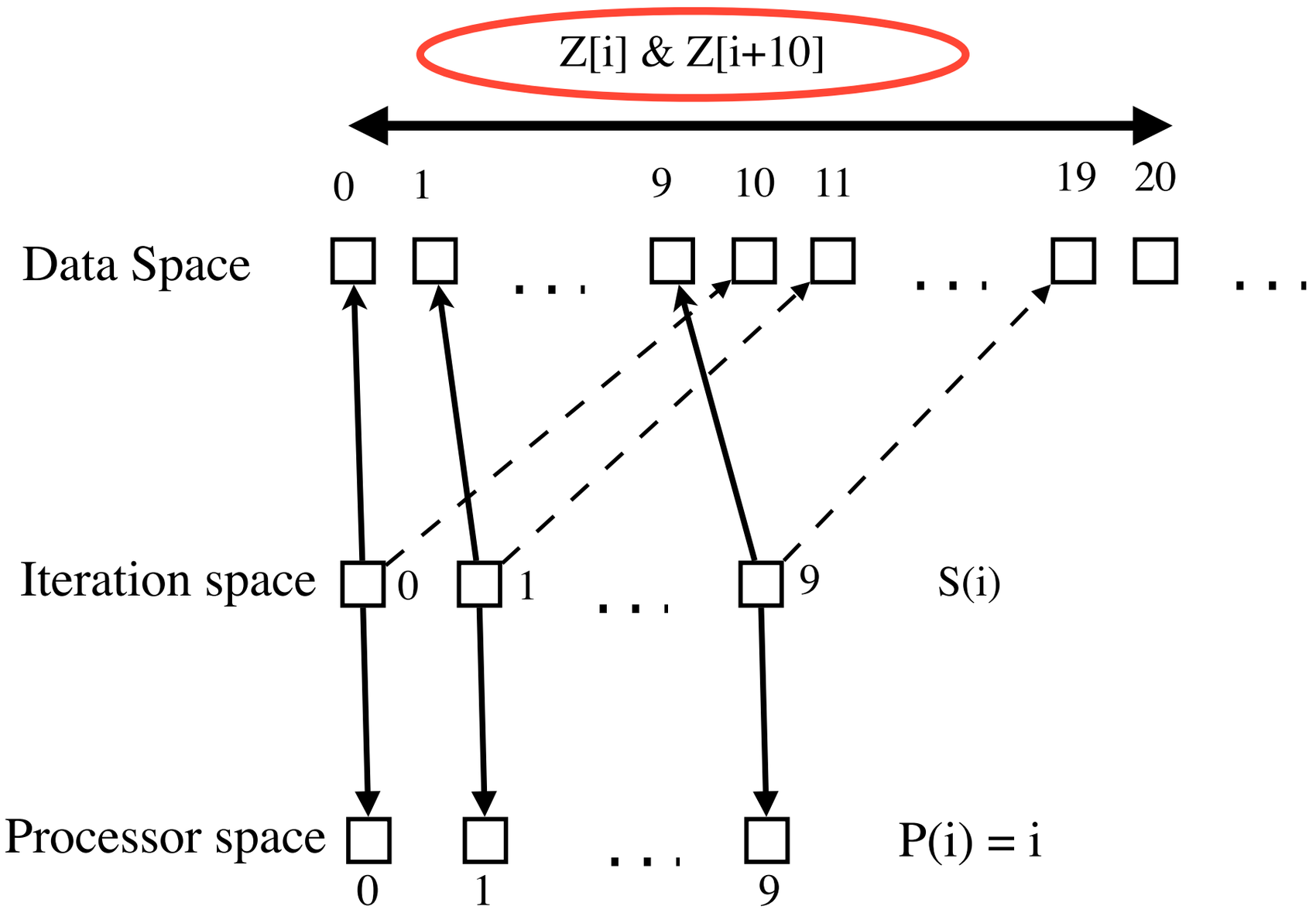
$i <= j <= 7$



Three Spaces

- **Iteration space, data space, and processor space**

```
float Z[100];
for (i=0; i<10; i++)
  S: Z[i+10] = Z[i];
```



Synchronization-free Parallelism

Parallelize an application **without** allowing any *communication* or *synchronization* among (logical) processors.

Example:

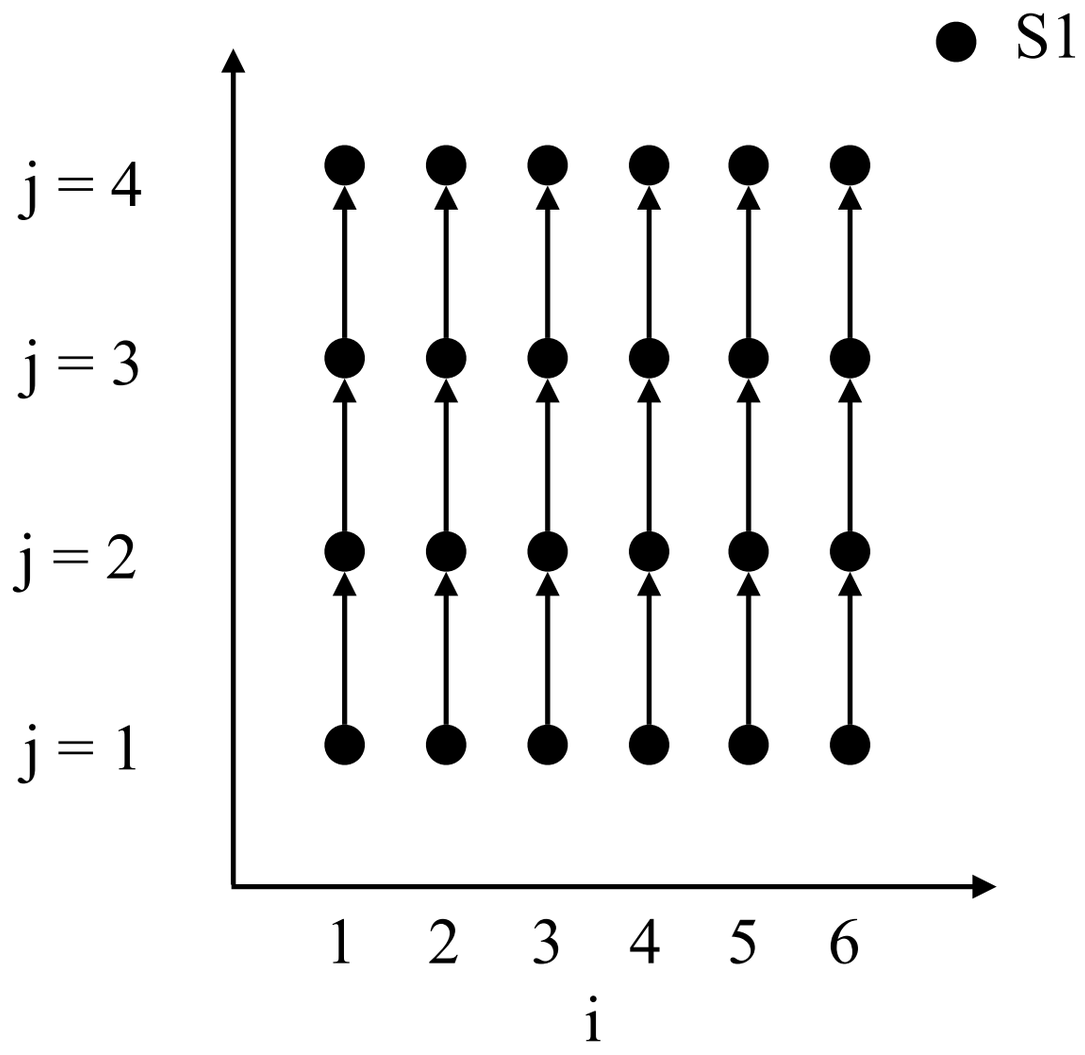
```
do i = 1, N
    do j = 1, N
        S1: A[i, j] = A[i, j - 1]
```

Write *in* $S_1(1,1)$ *to* ***Read*** *in* $S_1(1,2)$

↓

Write *in* $S_1(i, j)$ *to* ***Read*** *in* $S_1(i, j+1)$

Dependence from **S1** to **S1**



Synchronization-free Parallelism

Parallelize an application **without** allowing any *communication* or *synchronization* among (logical) processors.

Example:

```
do i = 1, N
    do j = 1, N
       S1: A[i, j] = A[i, j - 1]
```

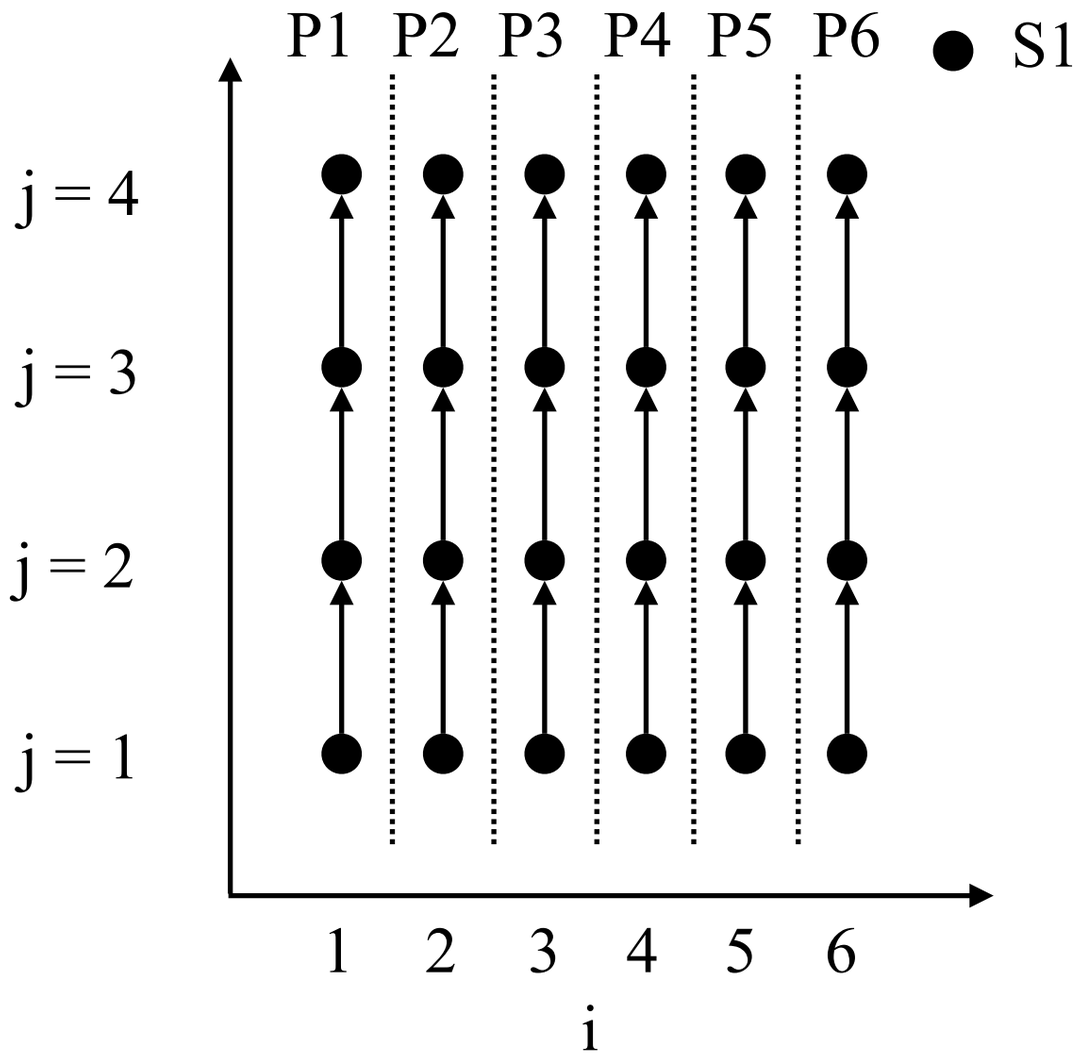
Write *in* $S_1(1,1)$ *to* ***Read*** *in* $S_1(1,2)$

↓

Write *in* $S_1(i, j)$ *to* ***Read*** *in* $S_1(i, j+1)$

Dependence from **S1** to **S1**

Communication is limited to the iterations on one processor.



Synchronization-free Parallelism

Parallelize an application **without** allowing any *communication* or *synchronization* among (logical) processors.

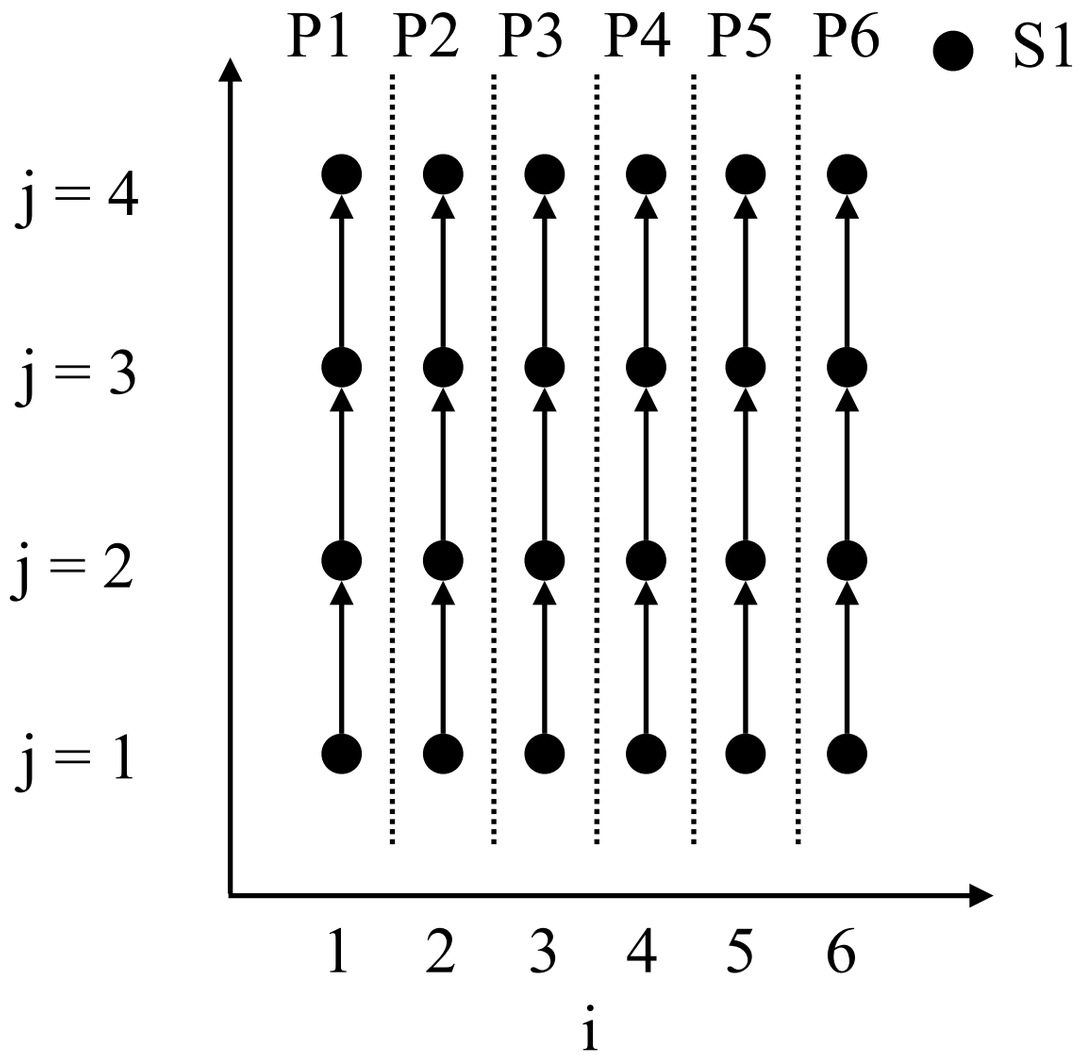
Example:

```
do i = 1, N
    do j = 1, N
        S1: A[i, j] = A[i, j - 1]
```

Write *in* $S_1(1,1)$ *to* ***Read*** *in* $S_1(1,2)$

↓

Write *in* $S_1(i, j)$ *to* ***Read*** *in* $S_1(i, j+1)$



Which loop can be parallelized?
The "i" loop or the "j" loop?

Answer: the "i" loop

Synchronization-free Parallelism

Parallelize an application **without** allowing any *communication* or *synchronization* among (logical) processors.

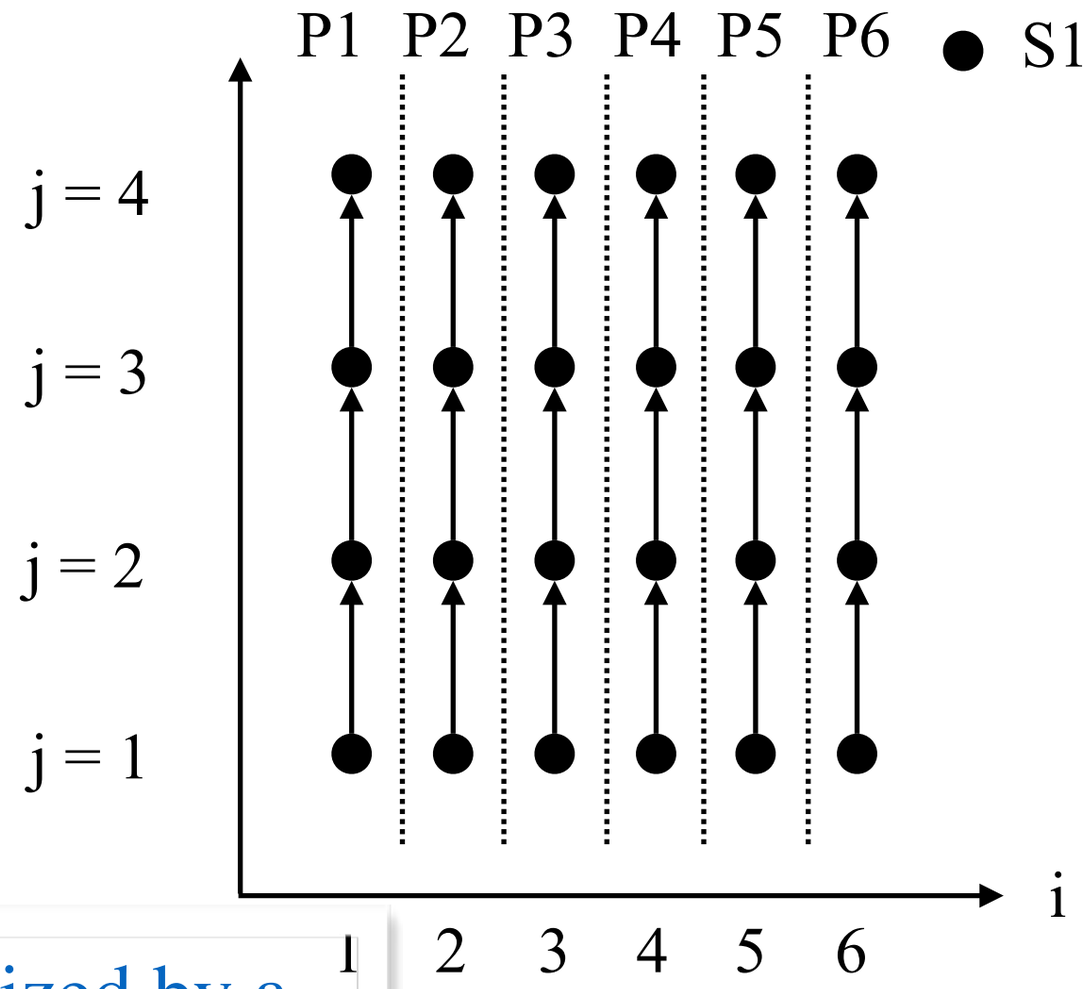
Example 1:

```
doall i = 1, N
    do j = 1, N
        S1: A[i, j] = A[i, j - 1]
```

***Write** in $S_1(1,1)$ to **Read** in $S_1(1,2)$*

↓

***Write** in $S_1(i, j)$ to **Read** in $S_1(i, j+1)$*



Dependence from **S1**(1,1) to **S2**(1,2)

The dependence chain is characterized by a **hyperplane**. In this case it is "i = constant".

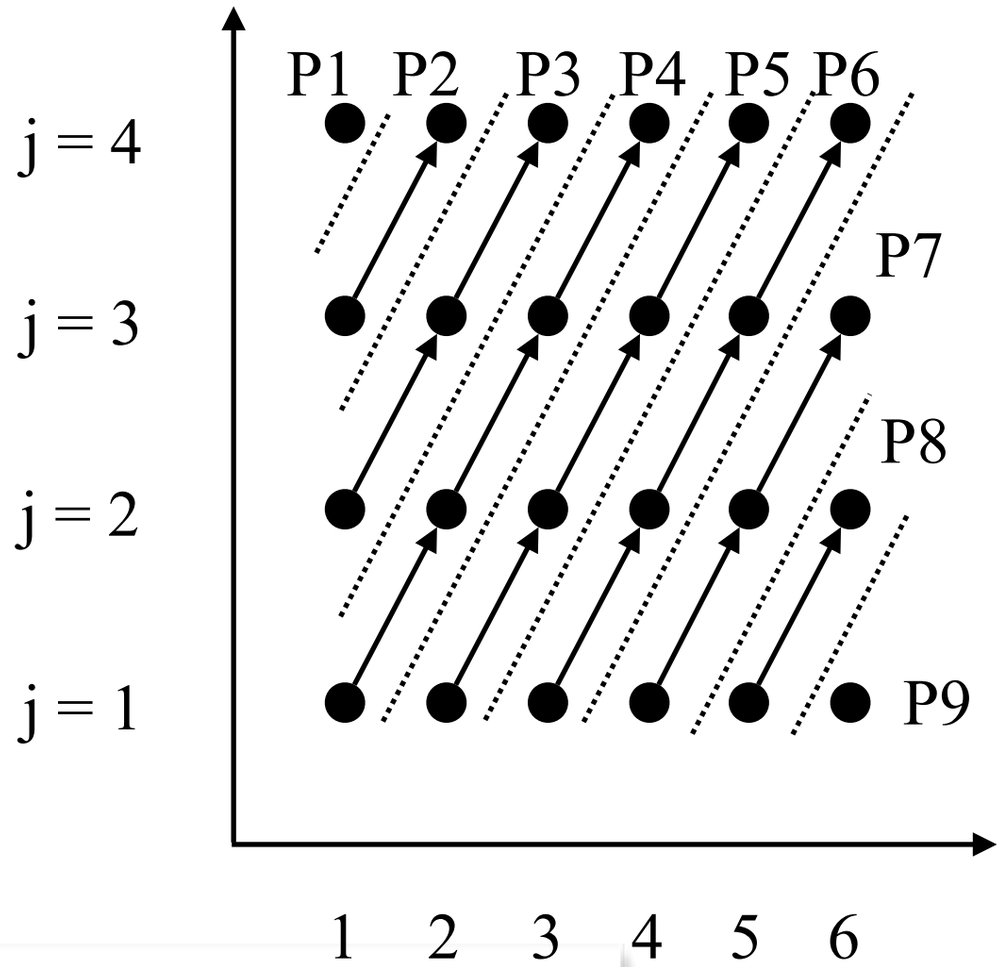
Synchronization-free Parallelism

Parallelize an application **without** allowing any *communication* or *synchronization* among (logical) processors.

Example 2:

```
do i = 1, N
    do j = 1, N
    S1: A[i, j] = A[i - 1, j - 1]
```

Dependence from **S1**(i,j) to **S1**(i+1,j+1)



The dependence chain is characterized by a **hyperplane**. In this case it is "j - i + constant = 0".

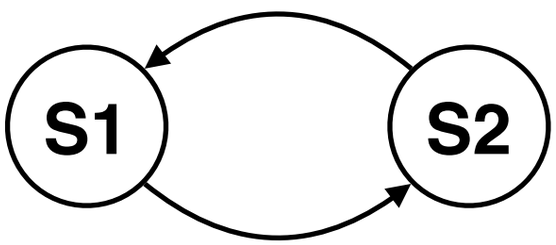
Synchronization-free Parallelism

Parallelize an application **without** allowing any *communication* or *synchronization* among (logical) processors.

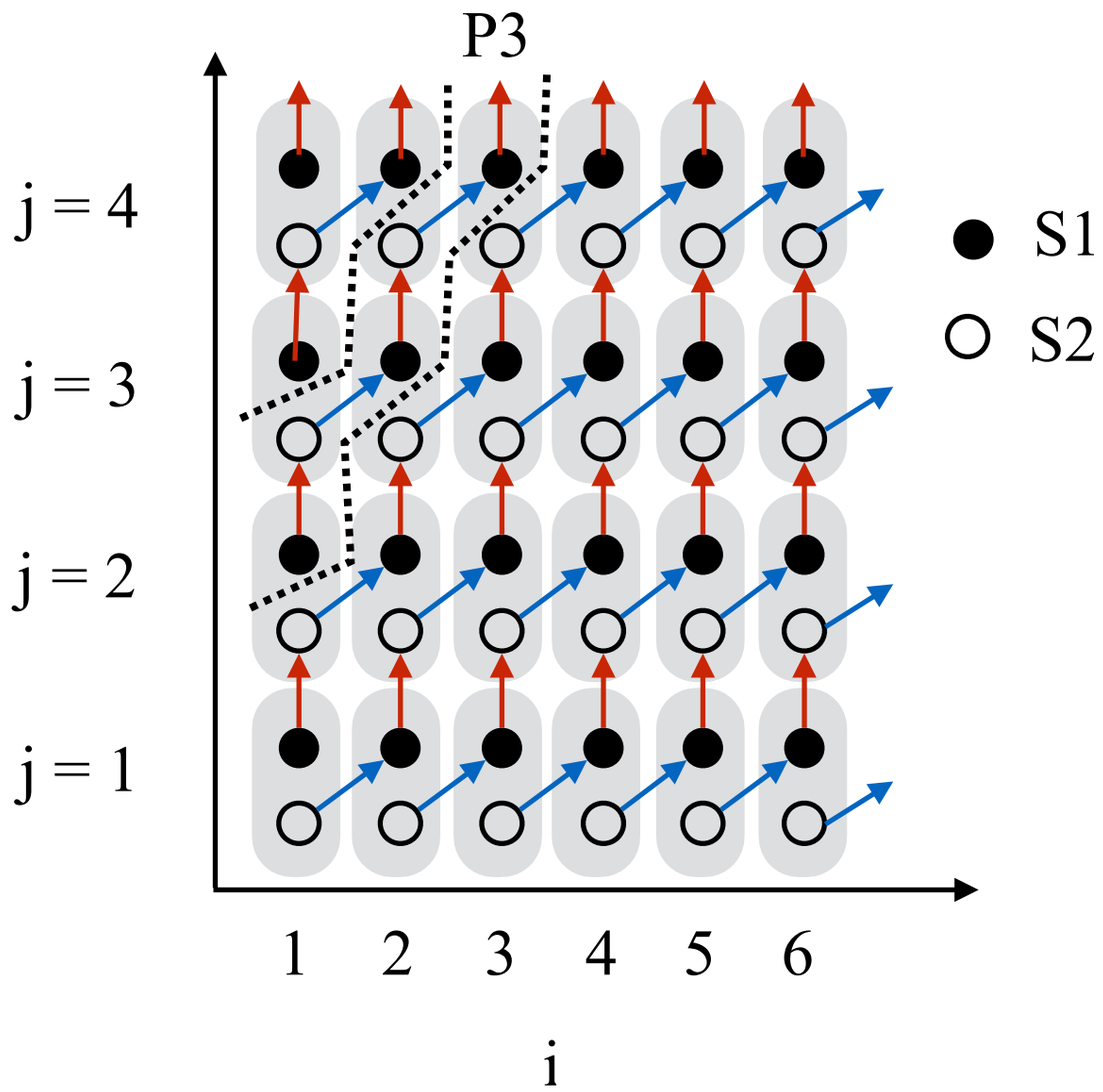
Example 3:

```
for (i=1; i<=100; i++)
    for (j=1; j<=100; j++){
        S1: X[i,j] = X[i,j] + Y[i-1, j];
        S2: Y[i,j] = Y[i,j] + X[i, j-1];
    }
```

True, i loop, for Y



True, j loop, for X



Dependence from **S1**(1,1) to **S2**(1,2)

Dependence from **S2**(1,1) to **S1**(2,1)

Dependence and Parallelization

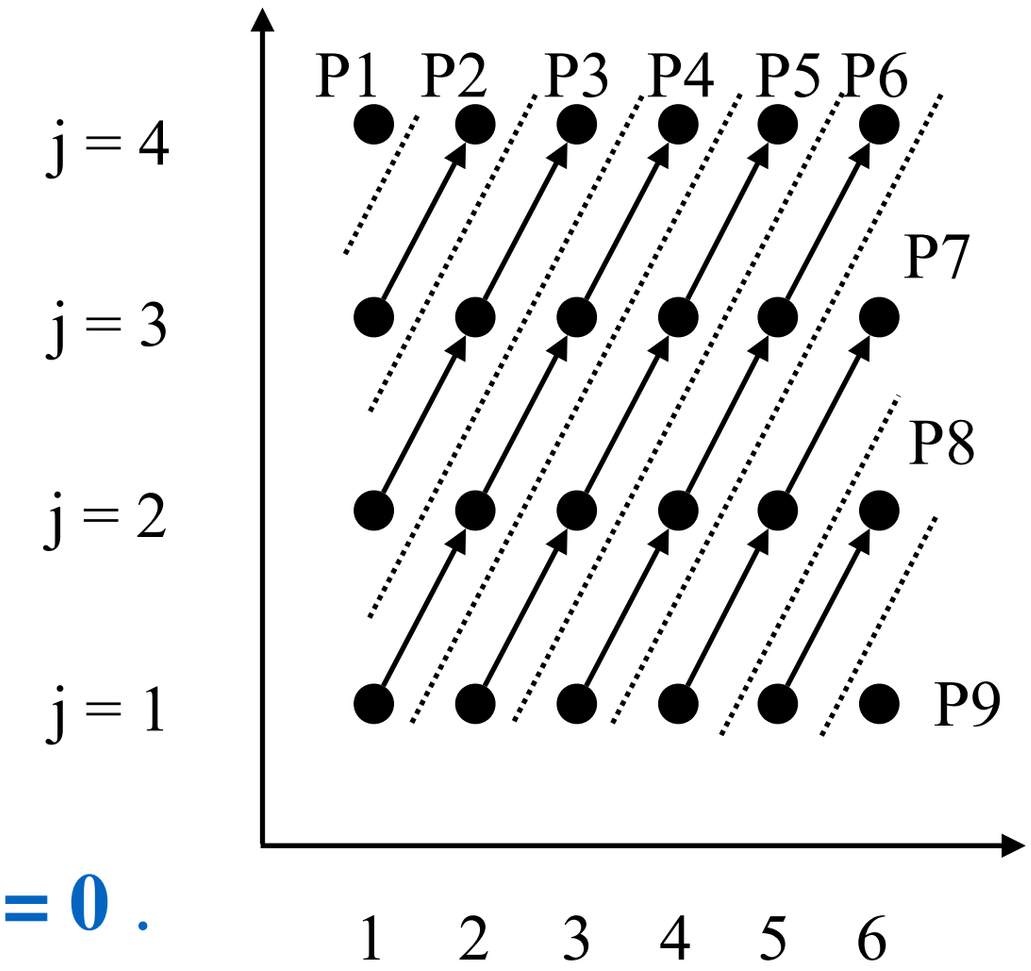
- Dependence chain in affine loops modeled as a hyperplane.
- Iterations along the same hyperplane must execute sequentially.
- **Iterations on different hyperplanes can execute in parallel.**

Example:

```
do i = 1, N
    do j = 1, N
    S1: A[i, j] = A[i - 1, j - 1]
```

Dependence from **S1**(i,j) to **S1**(i+1,j+1)

The hyperplane is **j - i + constant = 0** .



Processing Space: Affine Partition Schedule

- Map an iteration to a processor using < **C**, d >

  C is a *n by m* matrix

  - m = d (the loop level)
  - n is the dimension of the processor grid

  $\vec{d}$ is a n-element constant vector

  $\vec{p} = C\,\vec{x} + \vec{d}$**, where** $\vec{x}$ **is an iteration vector**

Processing Space: Affine Partition Schedule

- Map an iteration to a processor using < **C**, d >

$$\vec{p} = \mathbf{C}\,\vec{x} + \vec{d}\text{, where } \vec{x} \text{ is an iteration vector}$$

- Example

```
for (i=1; i<=N; i++)
  S: Y[i] = Z[i];
```

$\mathbf{C} = [1]$, $\mathbf{d} = [0]$

$\vec{p}(\ S(i)\ ) = 1*i + 0$

$= i$

Map iteration **i** to Processor **i**

Synchronization-free Parallelism

- Two memory references as <F_1, f_1, B_1, b_1> and <F_2, f_2, B_2, b_2> such that <F_2, f_2, B_2, b_2> at iteration α: (i_1, j_1) depends on <F_1, f_1, B_1, b_1> at iteration β: (i_2, j_2)
  - $\mathbf{F_1}$ is a matrix and f_1 is a vector.
    The affine memory access index is $\mathbf{F_1} * \alpha + f_1$.
  - $\mathbf{B_1}$ is a matrix and b_1 is a vector.
    The affine loop bounds can be expressed as $\mathbf{B_1} * \alpha + b_1 >= 0$
- Let <$\mathbf{C_1, d_1}$> and <$\mathbf{C_2, d_2}$> represent the respective processor schedule, to have synchronization-free parallelism,

$$\mathbf{C_1 * \alpha + d_1 = C_2 * \beta + d_2}$$

These two memory references must execute on the same processor (sequentially).

Synchronization-free Parallelism

- Two memory references as $<F_1, f_1, B_1, b_1>$ and $<F_2, f_2, B_2, b_2>$ such that $<F_2, f_2, B_2, b_2>$ at iteration (i_1, j_1) depends on $<F_1, f_1, B_1, b_1>$ at iteration (i_2, j_2)

```
for (i=1; i<=100; i++)
    for (j=1; j<=100; j++){
        S1: X[i,j] = X[i,j] + Y[i-1, j];
        S2: Y[i,j] = Y[i,j] + X[i, j-1];
    }
```

$$\mathbf{F_1}\begin{bmatrix} i_1 \\ j_1 \end{bmatrix} + f_1 = \mathbf{F_2}\begin{bmatrix} i_2 \\ j_2 \end{bmatrix} + f_2$$

$$B_1\begin{bmatrix} i_1 \\ j_1 \end{bmatrix} + b_1 >= 0$$

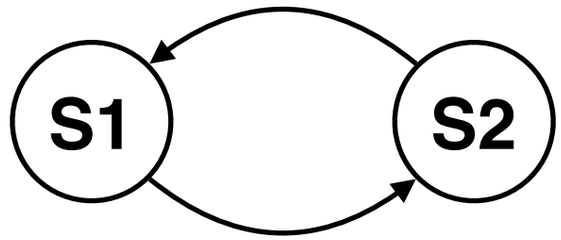
$$B_2\begin{bmatrix} i_2 \\ j_2 \end{bmatrix} + b_2 >= 0$$

- We want to find processor schedule $<\mathbf{C_1, d_1}>$ and $<\mathbf{C_2, d_2}>$ such that

$$[\ C_{11}\ C_{12}\ ]\begin{bmatrix} i_1 \\ j_1 \end{bmatrix} + [\ d_1\ ] = [\ C_{21}\ C_{22}\ ]\begin{bmatrix} i_2 \\ j_2 \end{bmatrix} + [\ d_2\ ]$$

Synchronization-free Parallelism

```
for (i=1; i<=100; i++)
    for (j=1; j<=100; j++){
        S1: X[i,j] = X[i,j] + Y[i-1, j];
        S2: Y[i,j] = Y[i,j] + X[i, j-1];
    }
```



$1 <= i_1 <= 100, \quad 1 <= j_1 <= 100,$

$1 <= i_2 <= 100, \quad 1 <= j_2 <= 100,$

$i_1 = i_2, \qquad j_1 = j_2 - 1,$

$$[C_{11}\ C_{12}]\begin{bmatrix} i_1 \\ j_1 \end{bmatrix} + [d_1] = [C_{11}\ C_{12}]\begin{bmatrix} i_2 \\ j_2 \end{bmatrix} + [d_2]$$

$$C_{11} - C_{21} = 0$$

$$C_{12} - C_{22} = 0$$

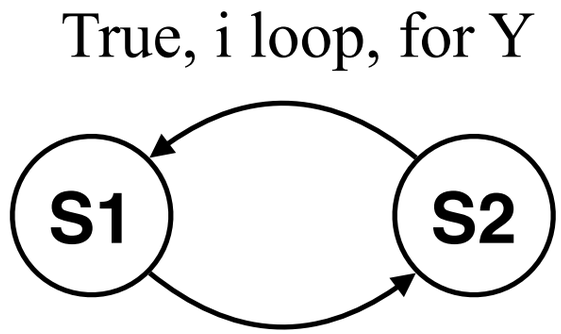
$$d_1 - d_2 - C_{22} = 0$$

$$[C_{11} - C_{21},\ C_{12} - C_{22}]\begin{bmatrix} i_1 \\ j_1 \end{bmatrix} + [d_1 - d_2 - C_{22}] = 0$$

S1 to S2 dependence

Synchronization-free Parallelism

```
for (i=1; i<=100; i++)
    for (j=1; j<=100; j++){
        S1: X[i,j] = X[i,j] + Y[i-1, j];
        S2: Y[i,j] = Y[i,j] + X[i, j-1];
    }
```



$1 <= i_3 <= 100, \quad 1 <= j_3 <= 100,$

$1 <= i_4 <= 100, \quad 1 <= j_3 <= 100,$

$i_3 - 1 = i_4, \quad j_3 = j_4,$

$$[C_{11}\ C_{12}]\begin{bmatrix} i_3 \\ j_3 \end{bmatrix} + [d_1] = [C_{11}\ C_{12}]\begin{bmatrix} i_4 \\ j_4 \end{bmatrix} + [d_2]$$

$$C_{11} - C_{21} = 0$$

$$C_{12} - C_{22} = 0$$

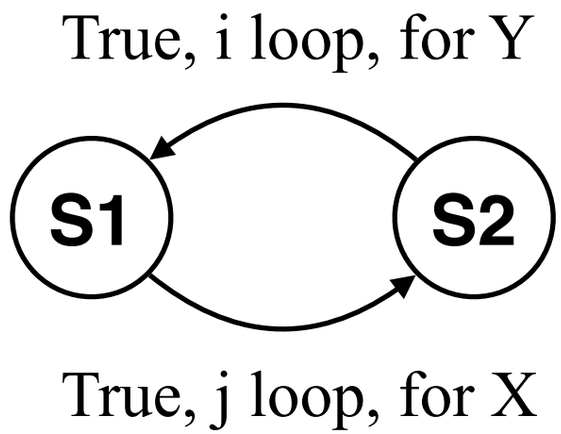
$$d_1 - d_2 + C_{21} = 0$$

$$[C_{11} - C_{21},\ C_{12} - C_{22}]\begin{bmatrix} i_3 \\ j_3 \end{bmatrix} + [d_1 - d_2 + C_{21}] = 0$$

S2 to S1 dependence

Synchronization-free Parallelism

```
for (i=1; i<=100; i++)
    for (j=1; j<=100; j++){
        S1: X[i,j] = X[i,j] + Y[i-1, j];
        S2: Y[i,j] = Y[i,j] + X[i, j-1];
    }
```



S1 to S2 dependence

$C_{11} - C_{21} = 0$

$C_{12} - C_{22} = 0$

$d_1 - d_2 - C_{22} = 0$

S2 to S1 dependence

$C_{11} - C_{21} = 0$

$C_{12} - C_{22} = 0$

$d_1 - d_2 + C_{21} = 0$

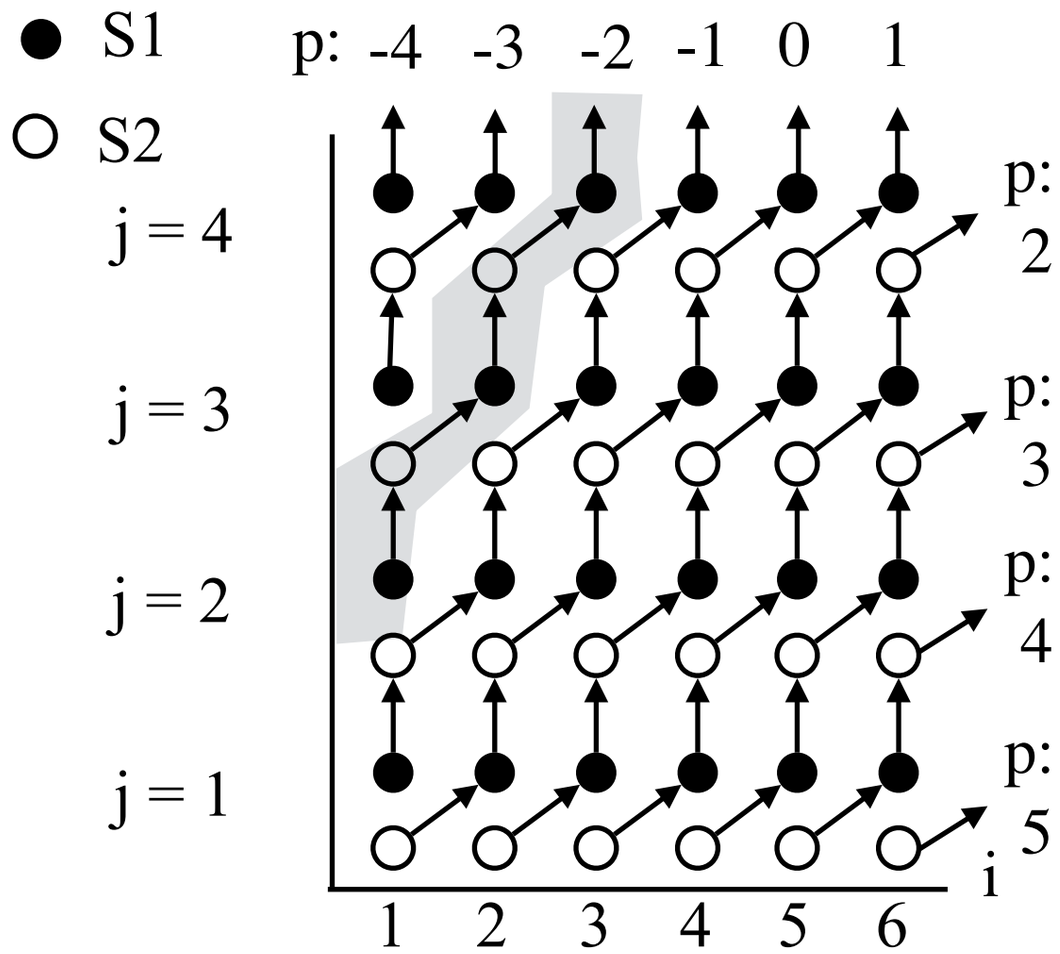
$$C_{11} = C_{21} = -C_{22} = -C_{12} = d_2 - d_1$$

Synchronization-free Parallelism

Example:

```
for (i=1; i<=100; i++)
    for (j=1; j<=100; j++){
        S1: X[i,j] = X[i,j] + Y[i-1, j];
        S2: Y[i,j] = Y[i,j] + X[i, j-1];
    }
```

$C_{11} = C_{21} = -C_{22} = -C_{12} = d_2 - d_1$



One Potential Solution:

Affine schedule for S1, p(S1): $[C_{11}\ C_{12}] = [1\ -1]$, $d_1 = -1$

i.e. (i, j) iteration of S1 to processor p = i - j - 1;

Affine schedule for S2, p(S2): $[C_{21}\ C_{22}] = [1\ -1]$, $d_2 = 0$

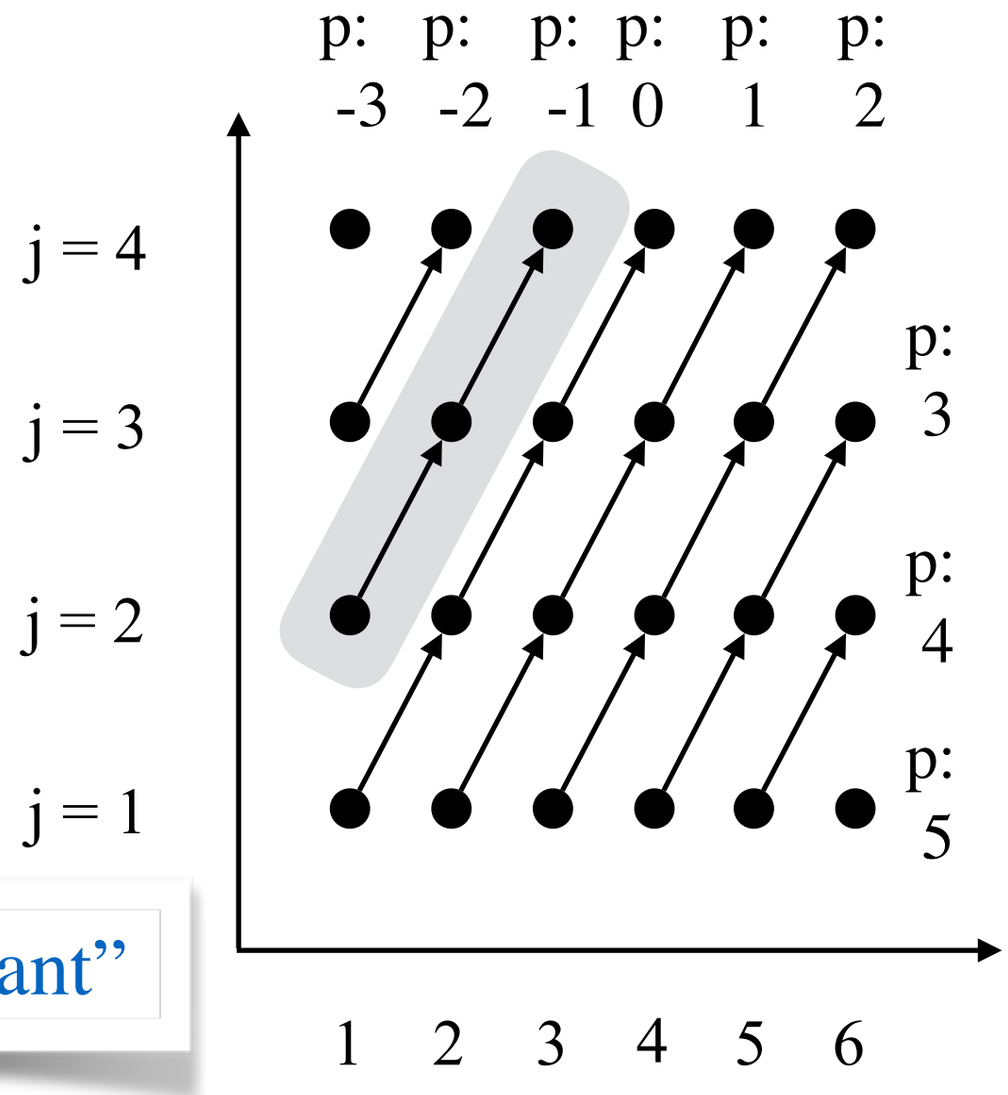
i.e. (i, j) iteration of S2 to processor p = i - j.

More Examples

Affine partition schedule

```
do I = 1, N
    do J = 1, N
      S1: A[I, J] = A[I-1, J - 1]
```

Read After Write



The hyperplane is j - i = “a constant”

Affine schedule for S_1, $p(S_1)$: $C = [C_{11}\ C_{12}] = [1\ -1]$, $d = 0$

i.e. (i, j) iteration of S_1 to processor $p = i - j$;

More Examples

Affine partition schedule

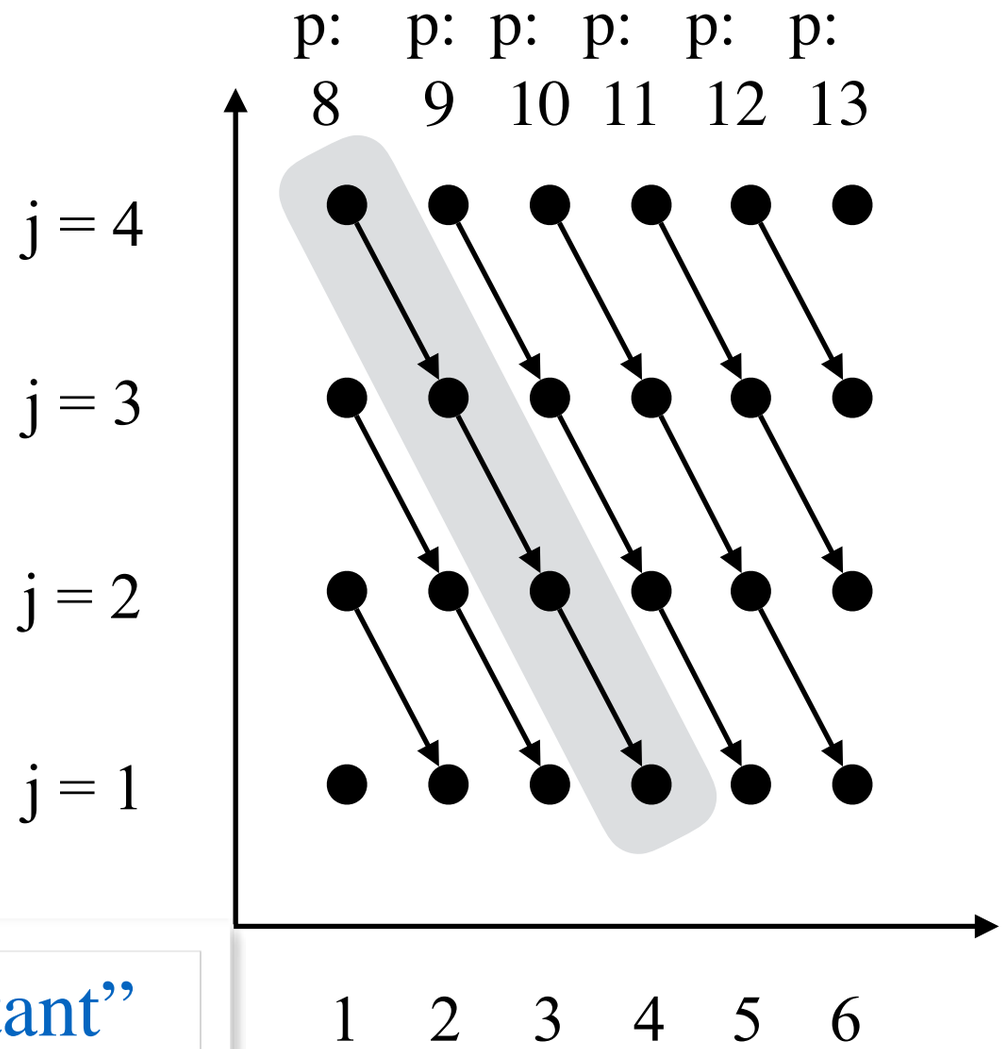
```
do I = 1, N
    do J = 1, N
    S1: A[I, J] = A[I+1, J-1]
```

Write After Read

Read in $S_1(1,2)$ to **Write** in $S_1(2,1)$

$S_1(i, i)$ to $S_1(i+1, i-1)$

The hyperplane is j + i = "a constant"



Affine schedule for S1, p(S1): C= [C_{11} C_{12}] = [1 1], d = 0

i.e. (i, j) iteration of S1 to processor p = i + j ;