CSE 160
Lecture 18

Parallel Matrix Multiplication
Working with communicators

Announcements

- Assign 4 deadline is extended by 6 hours
  Weds, Dec 3 @ 11.59PM

Today's lecture

- Quiz #4 return
- Cannon's Parallel Matrix Multiplication Algorithm
- Working with communicators

Quiz #4

- What is the significance of the values α and β_∞ in the formula
  $T(n) = \alpha + \beta_\infty * N$
- What are the values of s & t at the end?

Process P0	Process P1	Process P2
`Send(x,P2)`	`Send(z,P2)`	`Recv(s,*)` `Recv(t,*)`

Matrix Multiplication

- An important core operation in many numerical algorithms
- Given two *conforming* matrices A and B, form the matrix product A × B

  A is m × n

  B is n × p
- Operation count: $O(n^3)$ multiply-adds for an n × n square matrix

Simplest Serial Algorithm

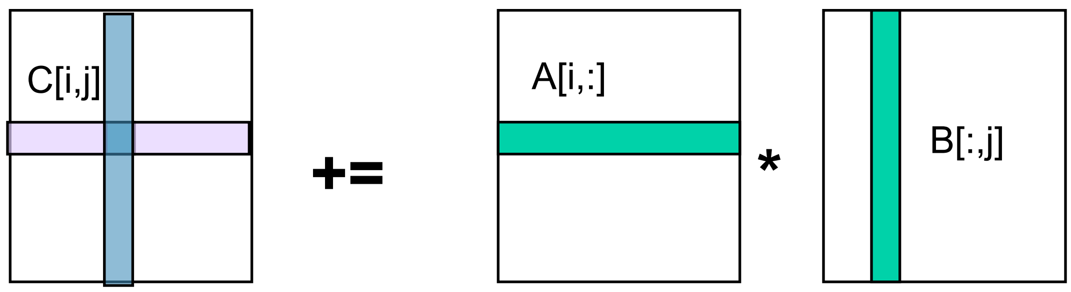
"ijk"

for i := 0 **to** n-1

  for j := 0 **to** n-1

    for k := 0 **to** n-1

      C[i,j] += A[i,k] * B[k,j]



Parallel matrix multiplication

- Assume p is a perfect square
- Each processor gets an $n/\sqrt{p} \times n/\sqrt{p}$ chunk of data
- Organize processors into rows and columns
- Process rank is an ordered pair of integers
- Assume that we have an efficient serial matrix multiply (dgemm, sgemm)

p(0,0)	p(0,1)	p(0,2)
p(1,0)	p(1,1)	p(1,2)
p(2,0)	p(2,1)	p(2,2)

Canon's algorithm

- Move data incrementally in $\sqrt{p}$ phases
- Circulate each chunk of data among processors within a row or column
- In effect we are using a ring broadcast algorithm
- Consider iteration i=1, j=2:

C[1,2] = A[1,0]*B[0,2] + A[1,1]*B[1,2] + A[1,2]*B[2,2]

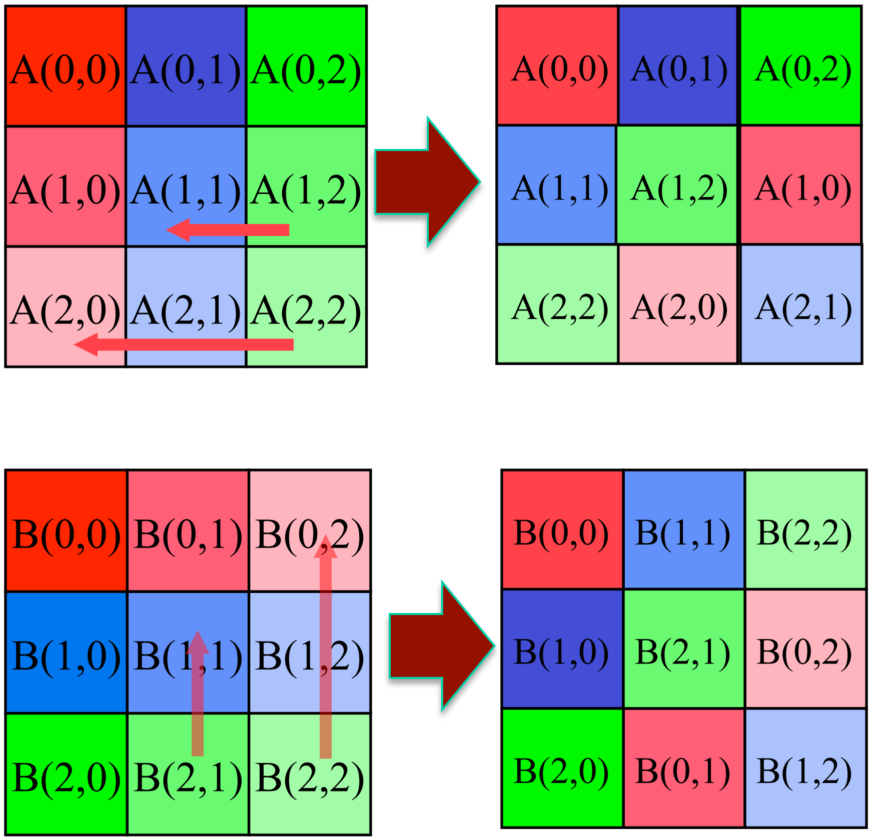
A(0,0)	A(0,1)	A(0,2)
A(1,0)	A(1,1)	A(1,2)
A(2,0)	A(2,1)	A(2,2)

B(0,0)	B(0,1)	B(0,2)
B(1,0)	B(1,1)	B(1,2)
B(2,0)	B(2,1)	B(2,2)

Image: Jim Demmel

Canon's algorithm

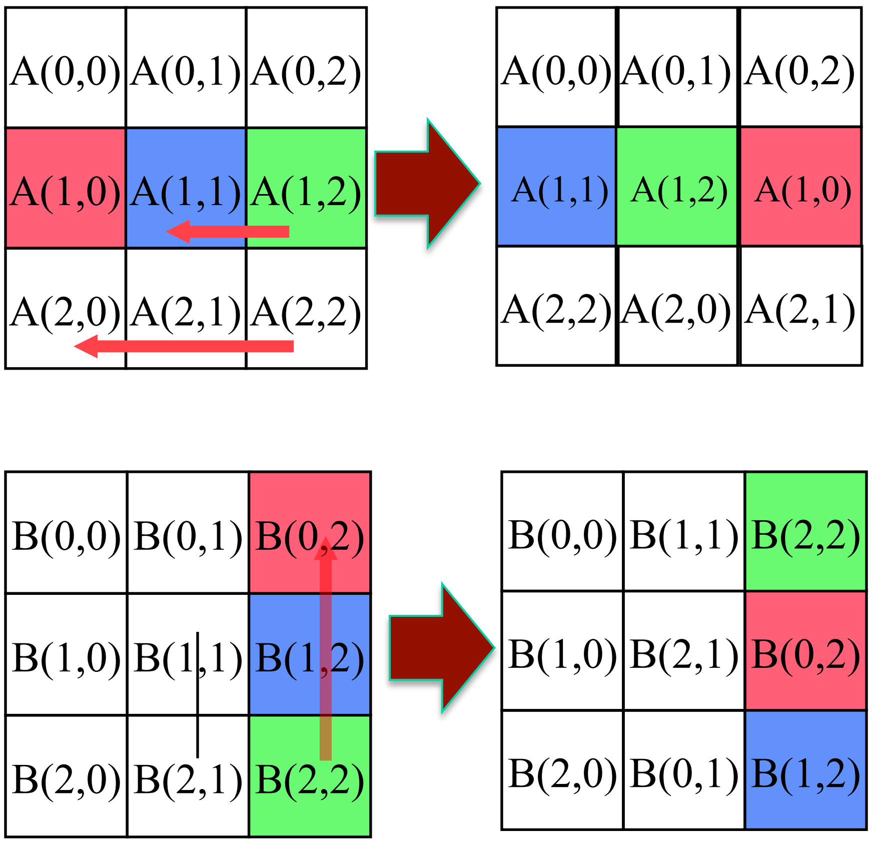
C[1,2] = A[1,0]*B[0,2] + A[1,1]*B[1,2] + A[1,2]*B[2,2]



- We want A[1,0] and B[0,2] to reside on the same processor initially
- Shift rows and columns so the next pair of values A[1,1] and B[1,2] line up
- And so on with A[1,2] and B[2,2]

Canon's algorithm – 1 element of C

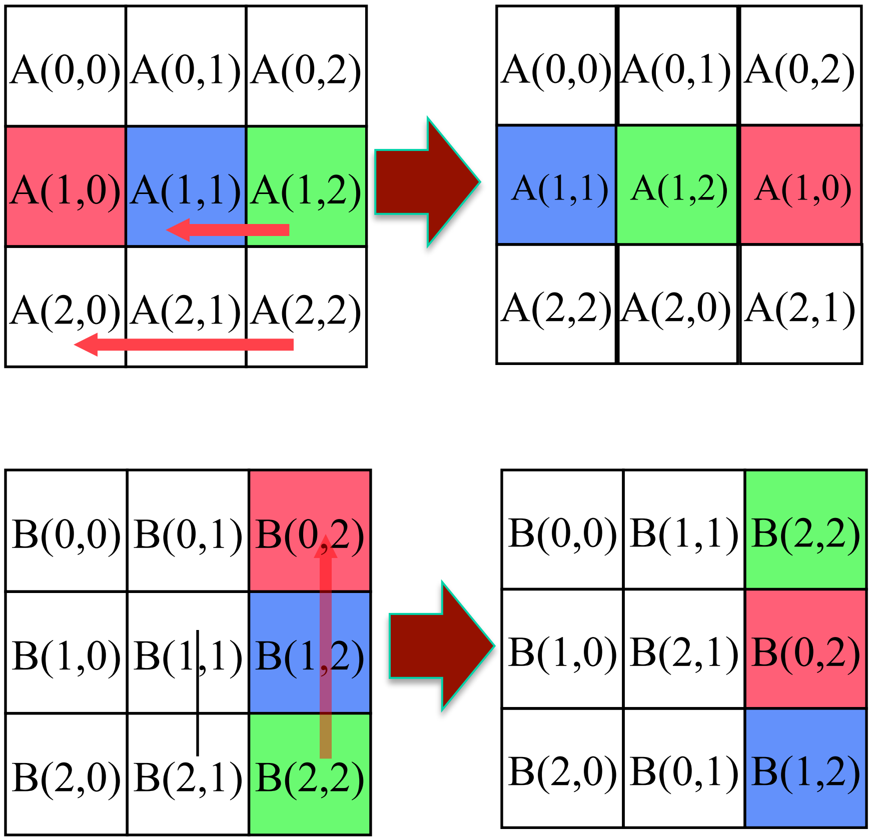
C[1,2] = A[1,0]*B[0,2] + A[1,1]*B[1,2] + A[1,2]*B[2,2]



- We want A[1,0] and B[0,2] to reside on the same processor initially
- Shift rows and columns so the next pair of values A[1,1] and B[1,2] line up
- And so on with A[1,2] and B[2,2]

Skewing the matrices

C[1,2] = A[1,0]*B[0,2] + A[1,1]*B[1,2] + A[1,2]*B[2,2]

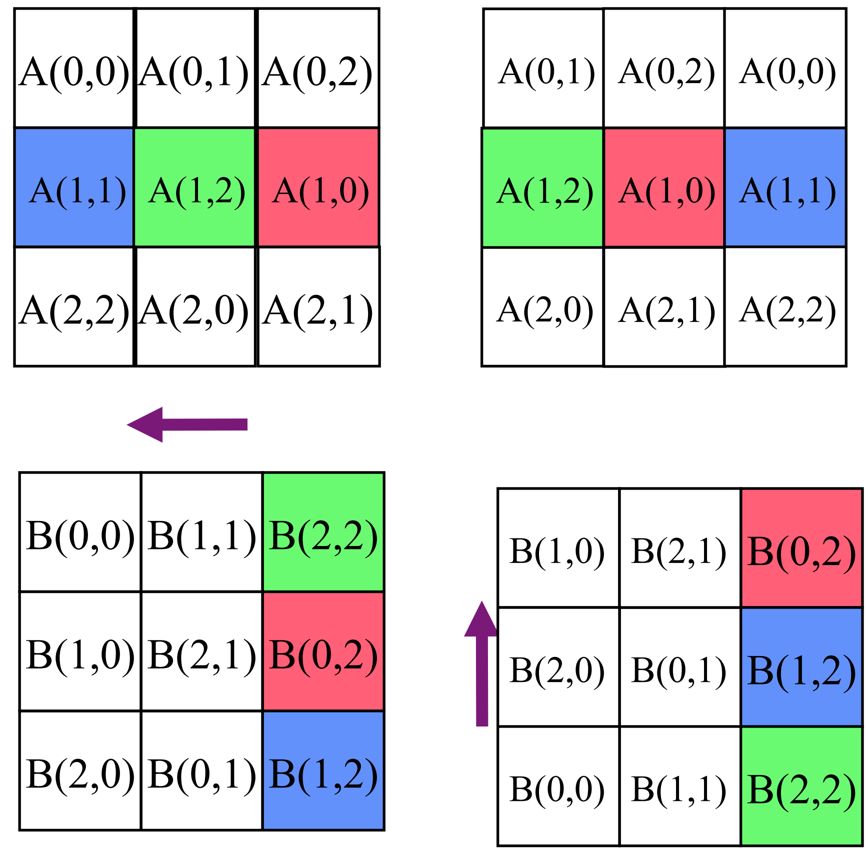


- We want A[1,0] and B[0,2] to reside on the same processor initially
- Shift rows and columns so the next pair of values A[1,1] and B[1,2] line up
- And so on with A[1,2] and B[2,2]

Shift and multiply

C[1,2] = A[1,0]*B[0,2] + A[1,1]*B[1,2] + A[1,2]*B[2,2]

- Takes √p steps
- Circularly shift
  - each row by 1 column to the left
  - each column by 1 row to the left
- Each processor forms the product of the two local matrices adding into the accumulated sum



Cost of Cannon's Algorithm

forall i=0 **to** $\sqrt{p}$ -1

CShift-left A[i; :] by i // $T= \alpha+\beta n^2/p$

forall j=0 **to** $\sqrt{p}$ -1

Cshift-up B[: , j] by j // $T= \alpha+\beta n^2/p$

for k=0 **to** $\sqrt{p}$ -1

forall i=0 **to** $\sqrt{p}$ -1 **and** j=0 to $\sqrt{p}$ -1

C[i,j] += A[i,j]*B[i,j] // $T = 2*n^3/p^{3/2}$

CShift-leftA[i; :] by 1 // $T= \alpha+\beta n^2/p$

Cshift-up B[: , j] by 1 // $T= \alpha+\beta n^2/p$

end forall

end for

$$T_P = 2n^3/p + 2(\alpha(1+\sqrt{p}) + \beta n^2/(1+\sqrt{p})/p)$$

$$E_P = T_1/(pT_P) = (1 + \alpha p^{3/2}/n^3 + \beta\sqrt{p}/n))^{-1}$$

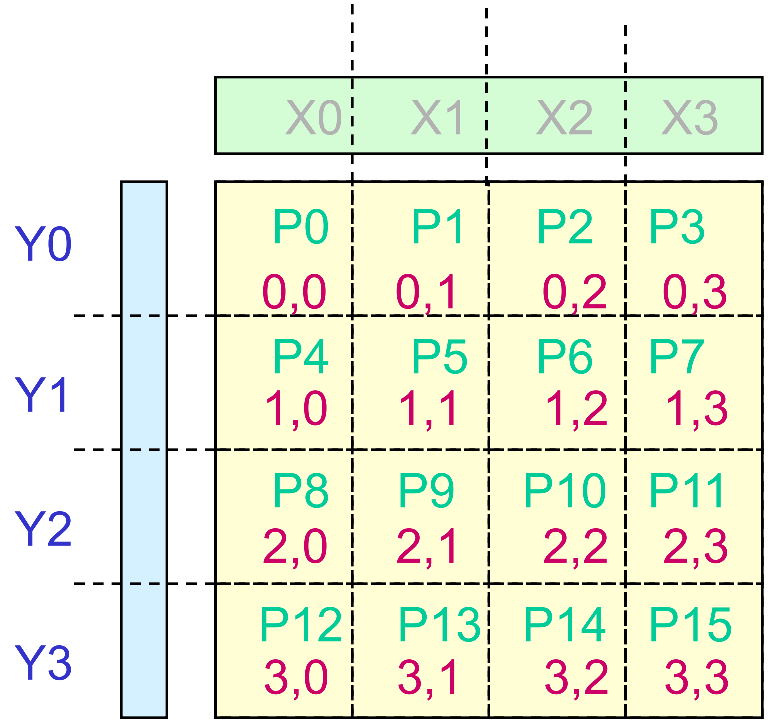
$$\approx (1 + O(\sqrt{p}/n))^{-1}$$

$E_P \rightarrow 1$ as $(n/\sqrt{p})$ grows [sqrt of data / processor]

Implementation

Communication domains

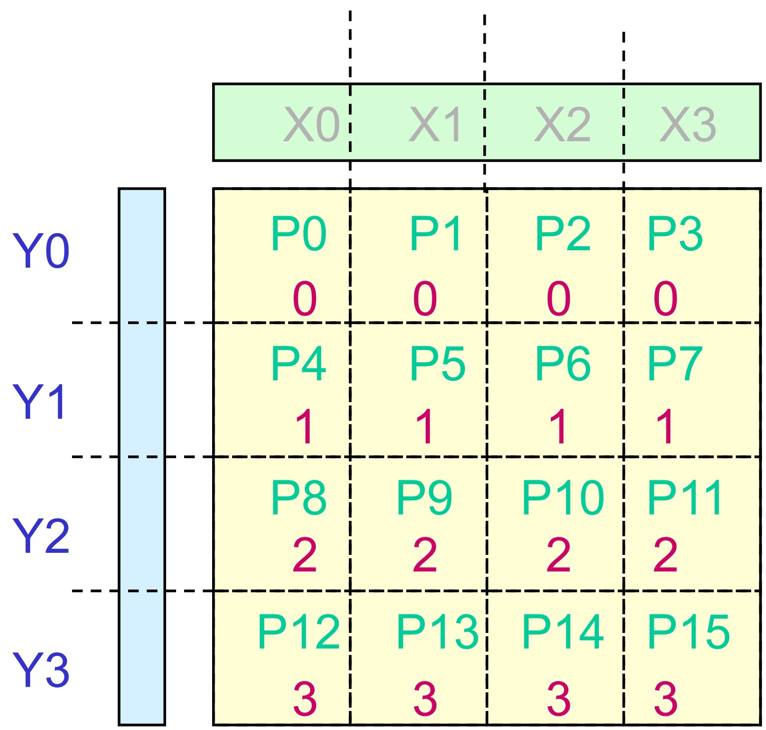
- Cannon's algorithm shifts data along rows and columns of processors
- MPI provides communicators for grouping processors, reflecting the communication structure of the algorithm
- An MPI communicator is a name space, a subset of processes that communicate
- Messages remain within their communicator
- A process may be a member of more than one communicator



Establishing row communicators

- Create a communicator for each row and column
- By Row

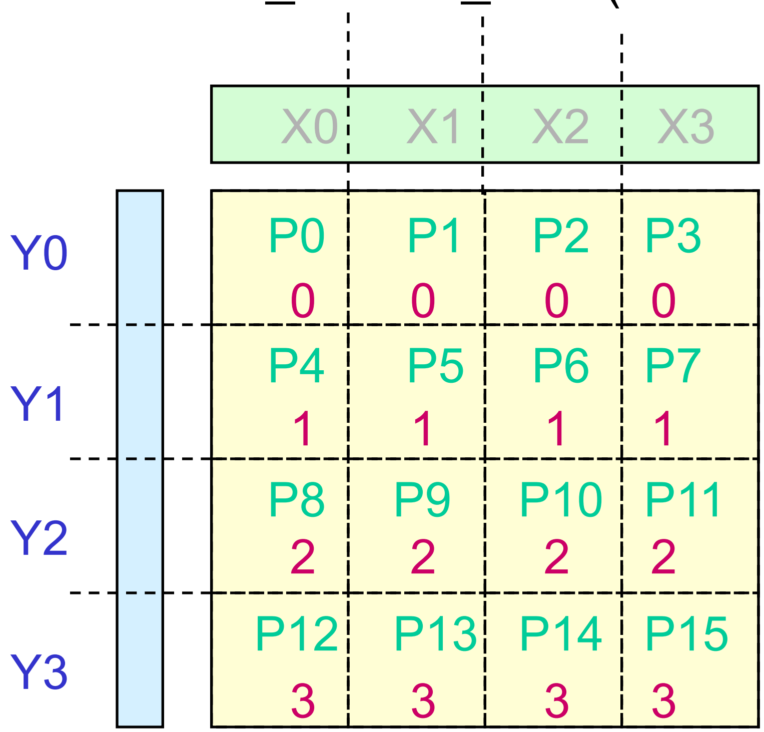
key = myRank div $\sqrt{P}$



Creating the communicators

- Create a row communicator   key = myRank div √P

```
MPI_Comm rowComm;
MPI_Comm_split( MPI_COMM_WORLD,
                myRank / √P, myRank, &rowComm);
MPI_Comm_rank(rowComm,&myRow);
```



- Each process obtains a new communicator
- Each process' rank relative to the new communicator
- Rank applies to the respective communicator only
- Ordered according to myRank

More on Comm_split

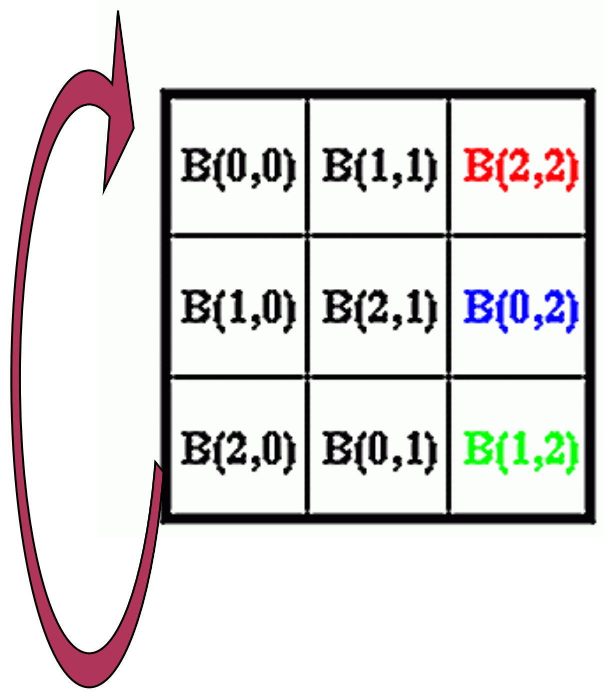
MPI_Comm_split(MPI_Comm comm, int splitKey,
int rankKey, MPI_Comm* newComm)

- Ranks assigned arbitrarily among processes sharing the same **rankKey** value
- May exclude a process by passing the constant **MPI_UNDEFINED** as the **splitKey**
- Return a special **MPI_COMM_NULL** communicator
- If a process is a member of several communicators, it will have a rank within each one

Circular shift

- Communication with columns (and rows

p(0,0)	p(0,1)	p(0,2)
p(1,0)	p(1,1)	p(1,2)
p(2,0)	p(2,1)	p(2,2)



More on Comm_split

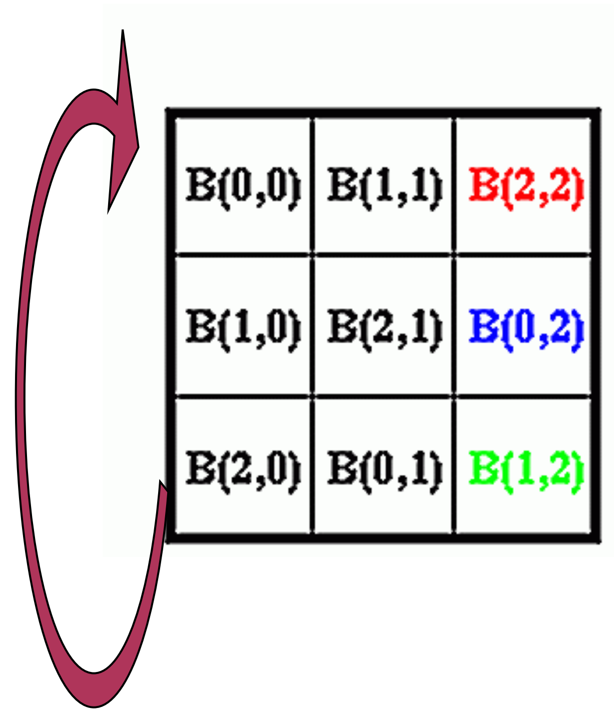
**MPI_Comm_split(MPI_Comm comm, int splitKey,
int rankKey, MPI_Comm* newComm)**

- Ranks assigned arbitrarily among processes sharing the same **rankKey** value
- May exclude a process by passing the constant **MPI_UNDEFINED** as the **splitKey**
- Return a special **MPI_COMM_NULL** communicator
- If a process is a member of several communicators, it will have a rank within each one

Circular shift

- Communication with columns (and rows

p(0,0)	p(0,1)	p(0,2)
p(1,0)	p(1,1)	p(1,2)
p(2,0)	p(2,1)	p(2,2)



Circular shift

- Communication with columns (and rows)

```
MPI_Comm_rank(rowComm,&myidRing);
MPI_Comm_size(rowComm,&nodesRing);
int next = (myidRng + 1 ) % nodesRing;
MPI_Send(&X,1,MPI_INT,next,0, rowComm);
MPI_Recv(&XR,1,MPI_INT,
         MPI_ANY_SOURCE,
         0, rowComm, &status);
```

- Processes 0, 1, 2 in one communicator because they share the same key value (0)
- Processes 3, 4, 5 are in another (key=1), and so on