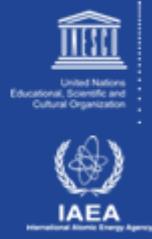




The Abdus Salam
International Centre
for Theoretical Physics



Overview of Common Strategies for Parallelization

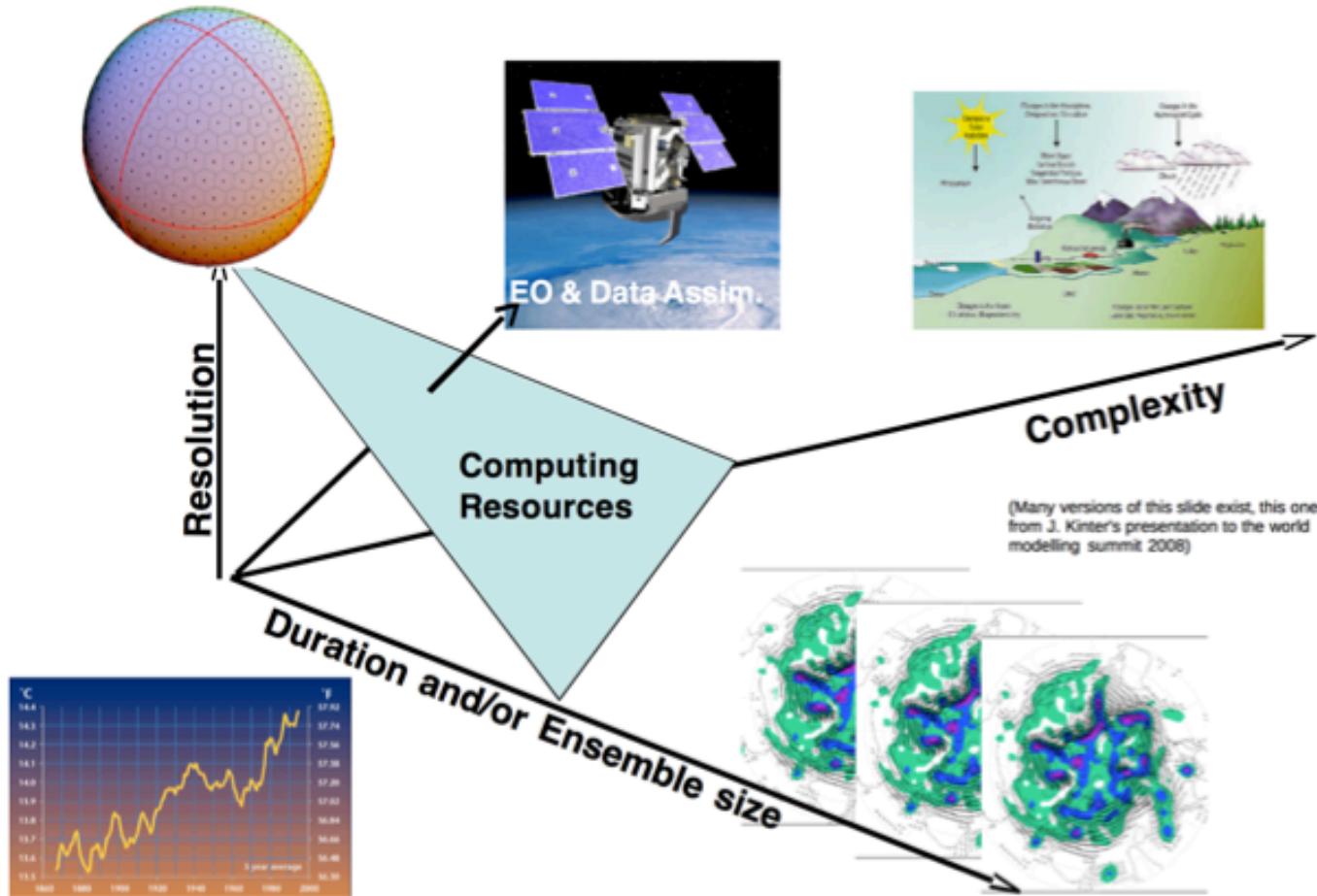
Ivan Girotto – igirotto@ictp.it

International Centre for Theoretical Physics (ICTP)

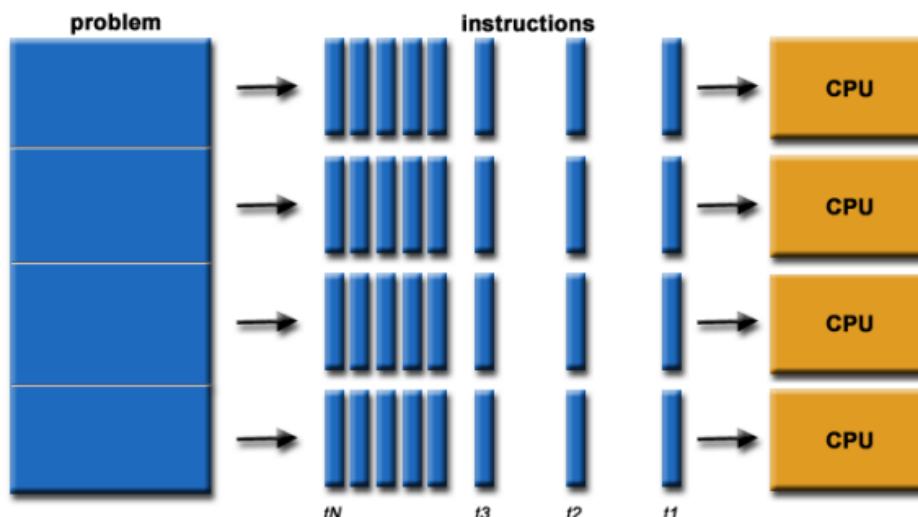


Parallelism - 101

- there are two main reasons to write a parallel program:
 - access to larger amount of memory (aggregated, going bigger)
 - reduce time to solution (going faster)



The first step in developing a parallel algorithm is to decompose the problem into tasks that can be executed concurrently



- A problem is broken into discrete parts that can be solved concurrently
- Each part is further broken down to a series of instructions
- Instructions from each part execute simultaneously on different processors
- An overall control / coordination mechanism is employed

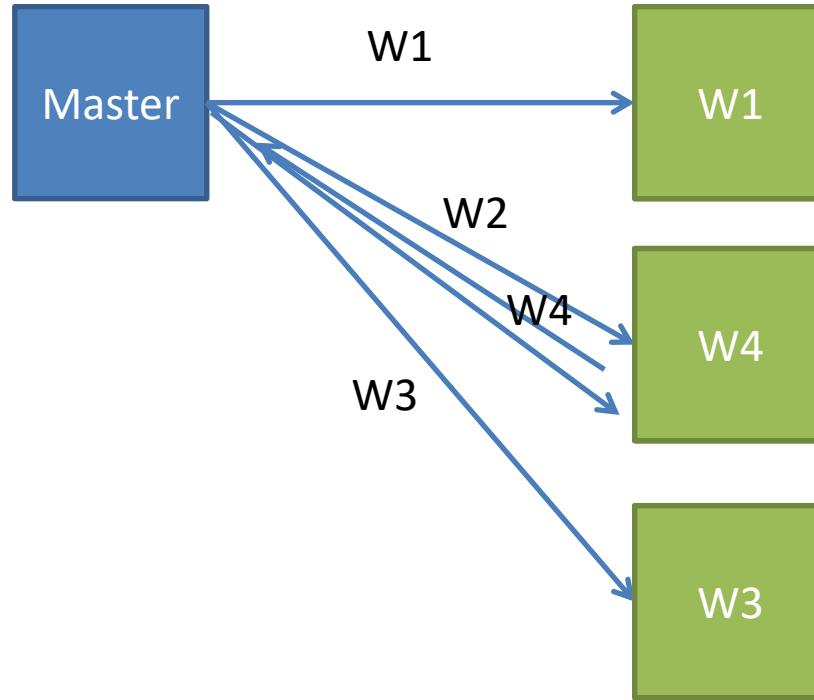
- Identify portions of the work that can be performed concurrently
- Mapping the concurrent pieces of work onto multiple processes running in parallel
- Distributing the input, output and intermediate data associated within the program
- Managing accesses to data shared by multiple processors
- Synchronizing the processors at various stages of the parallel program execution



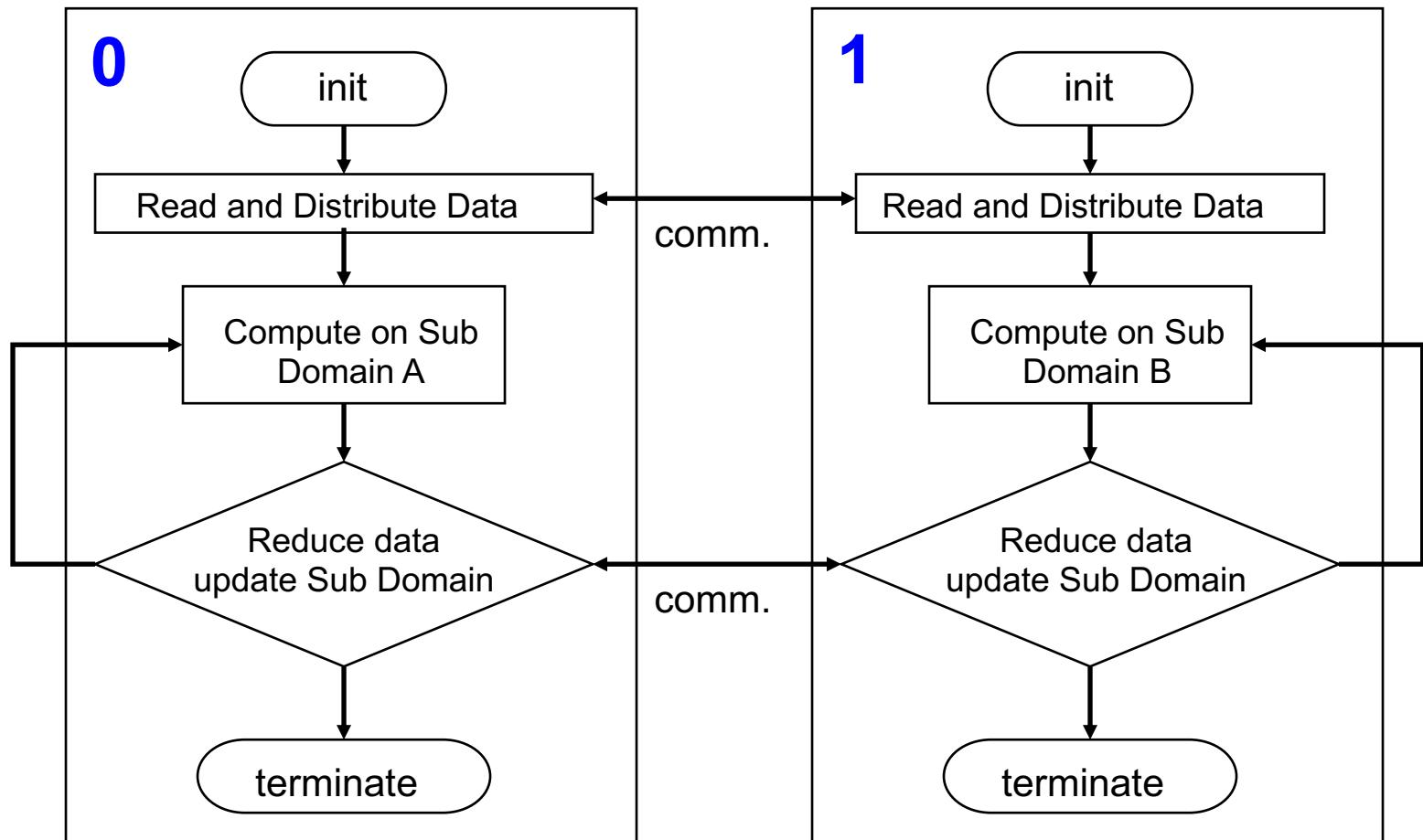
Easy Parallel Computing

- Farming, embarrassingly parallel
 - Executing multiple instances on the same program with different inputs/initial cond.
 - Reading large binary files by splitting the workload among processes
 - Searching elements on large data-sets
 - Other parallel execution of embarrassingly parallel problem (no communication among tasks)
- Ensemble simulations (weather forecast)
- Parameter space (find the best wing shape)

Master/Slave



A Parallel Program on Distributed Memory



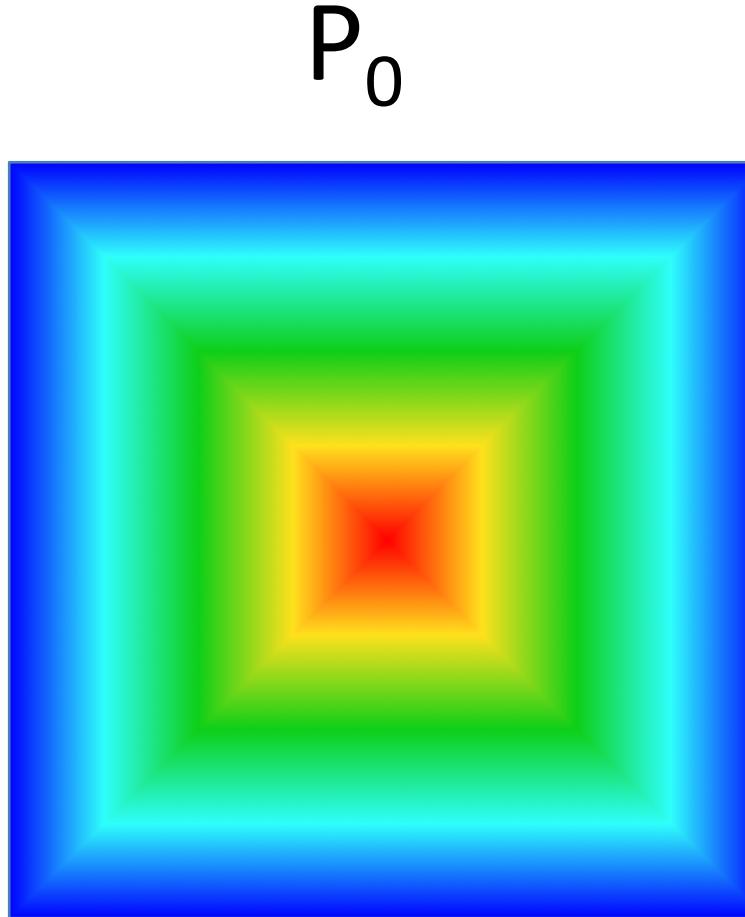


Distributed Data Vs Replicated Data

- Replicated data distribution is useful if it helps to reduce the communication among process at the cost of bounding scalability
- Distributed data is the ideal data distribution but not always applicable for all data-sets
- Usually complex application are a mix of those techniques => distribute large data sets; replicate small data



The Abdus Salam
**International Centre
for Theoretical Physics**



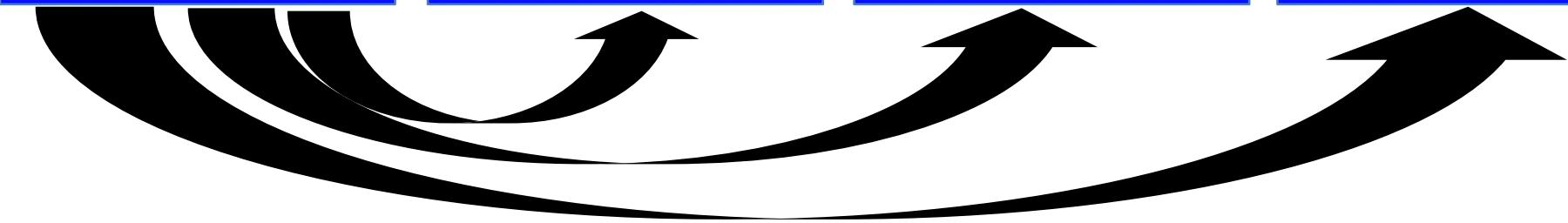
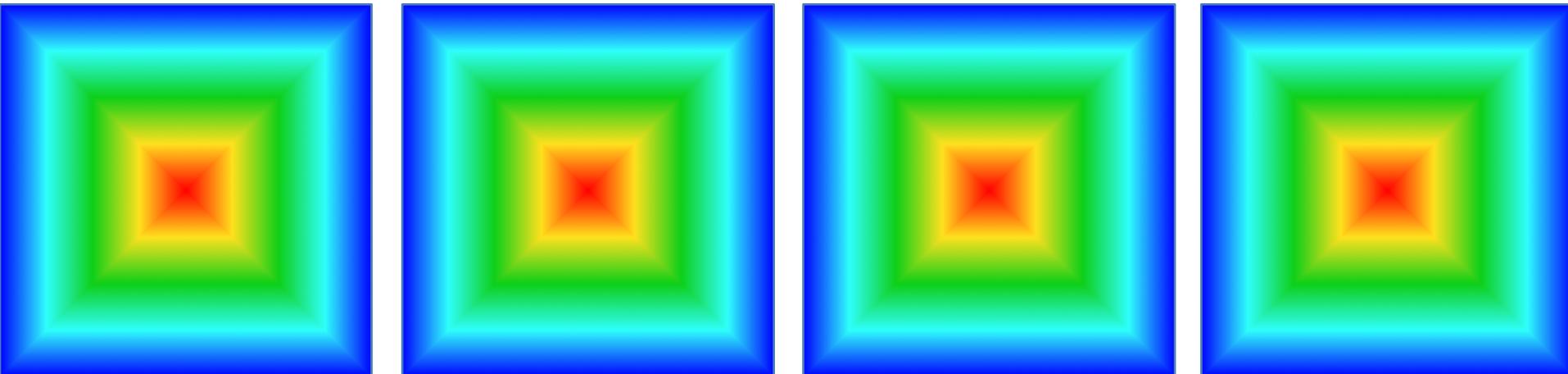
call MPI_BCAST(...)

P_0 (root)

P_1

P_2

P_3





P_0

P_1

P_2

P_3



call evolve(dtfact)

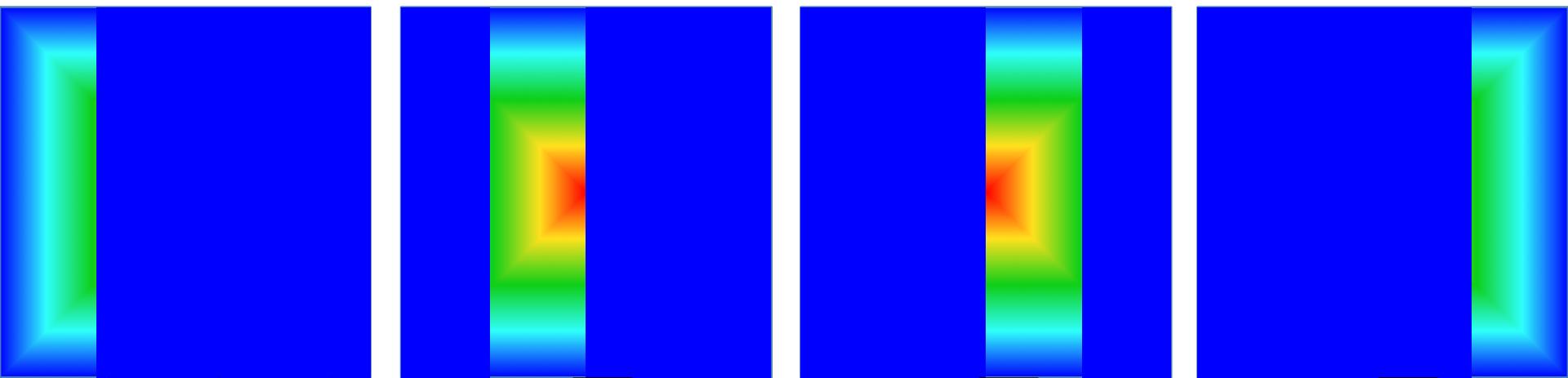
call MPI_Gather(..., ..., ...)

P_0 (root)

P_1

P_2

P_3



Replicated data

- Compute domain (and workload) distribution among processes
- Master-slaves: P_0 drives all processes
- Large amount of data communication
 - at each step P_0 distribute data to all processes and collect the contribution of each process
- Problem size scaling limited in memory capacity



Static Data Partitioning

The simplest data decomposition schemes for dense matrices are 1-D block distribution schemes.

row-wise distribution

P_0
P_1
P_2
P_3
P_4
P_5
P_6
P_7

column-wise distribution

P_0	P_1	P_2	P_3	P_4	P_5	P_6	P_7
-------	-------	-------	-------	-------	-------	-------	-------

Block Array Distribution Schemes

Block distribution schemes can be generalized to higher dimensions as well.

P_0	P_1	P_2	P_3
P_4	P_5	P_6	P_7
P_8	P_9	P_{10}	P_{11}
P_{12}	P_{13}	P_{14}	P_{15}

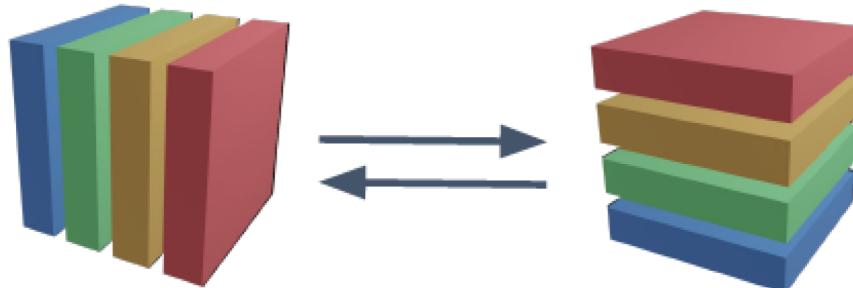
(a)

P_0	P_1	P_2	P_3	P_4	P_5	P_6	P_7
P_8	P_9	P_{10}	P_{11}	P_{12}	P_{13}	P_{14}	P_{15}

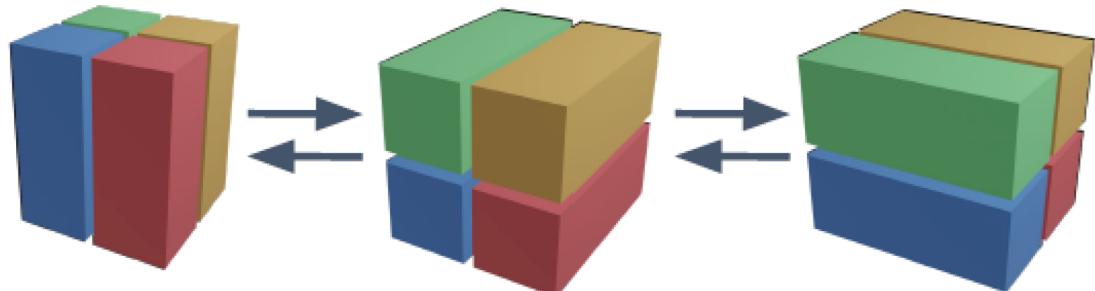
(b)

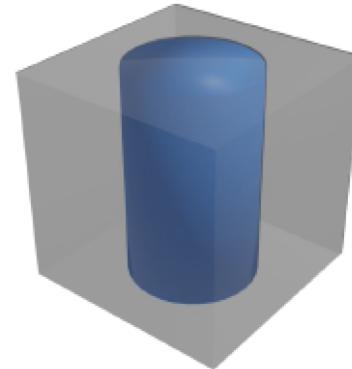
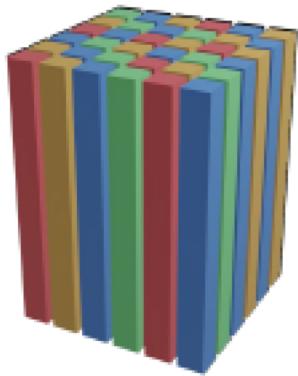
Degree to which tasks/data can be subdivided is limit to concurrency and parallel execution!!

Slab decomposition:

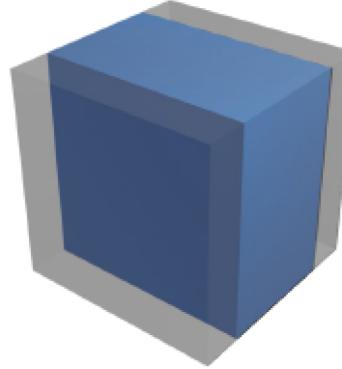
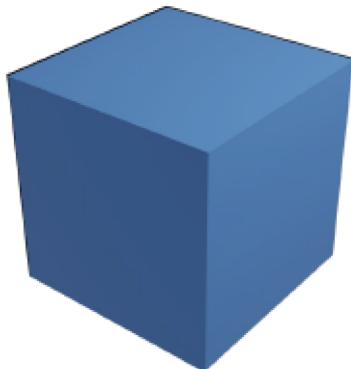


Pencil decomposition:





Flexible pencil decomposition



Global Vs Local Indexes

- In sequential code you always refer to global indexes
- With distributed data you must handle the distinction between global and local indexes (and possibly implementing utilities for transparent conversion)

Local Idx

1	2	3
---	---	---

1	2	3
---	---	---

1	2	3
---	---	---

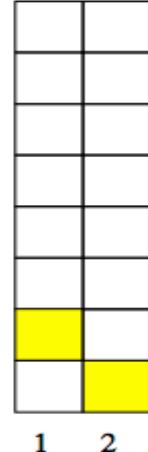
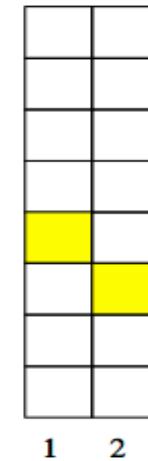
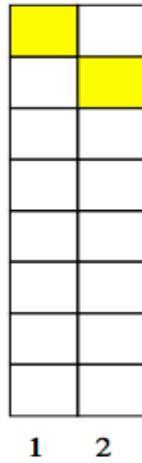
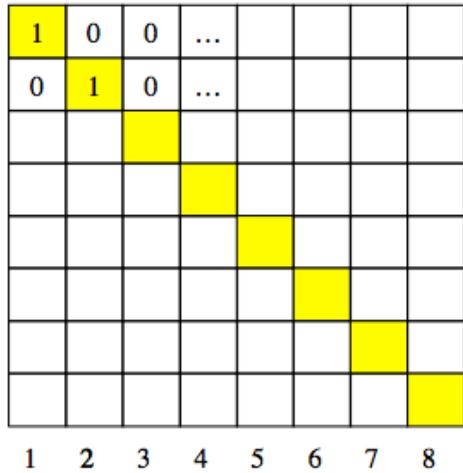
Global Idx

1	2	3
---	---	---

4	5	6
---	---	---

7	8	9
---	---	---

Domain Decomposition



```
// Initialization of a Identity Matrix
for( i = 0; i < SIZE; i++ ){
    for( j = 0; j < SIZE; j++ ){
        if( i == j ) mat[ i * SIZE + j ] = 1.0;
        else mat[ i * SIZE + j ] = 0.0;
    }
}
```

Identity Matrix /1



```
[...]
```

```
int size_loc = SIZE / npes;
```

```
// NO!!!! double * mat = malloc( SIZE * SIZE * sizeof(double) );
```

```
double * mat = malloc( SIZE * size_loc * sizeof(double) );
int i, j;
```

```
// Initialization of the distributed Indenty Matrix
```

```
for( i = 0; i < size_loc; i++ ){
    for( j = 0; j < SIZE; j++ ) {
```

```
        int i_global = rank * size_loc + i;
```

```
        if( i_global == j ) mat[ i * SIZE + j ] = 1.0;
```

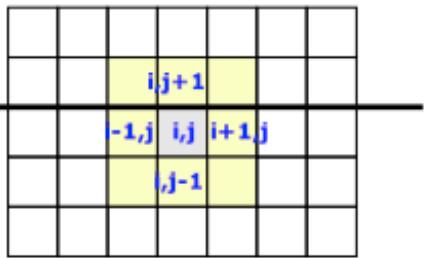
```
        else mat[ i * SIZE + j ] = 0.0;
```

```
}
```

```
}
```

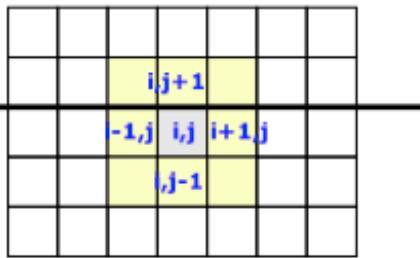
Again on Domain Decomposition

sub-domain boundaries



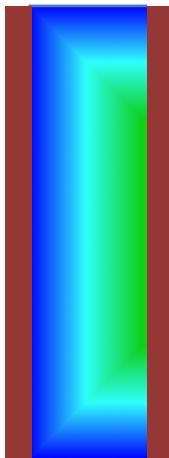
Collaterals to Domain Decomposition /2

sub-domain boundaries

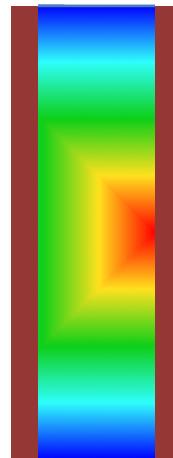


The Transport Code - Parallel Version

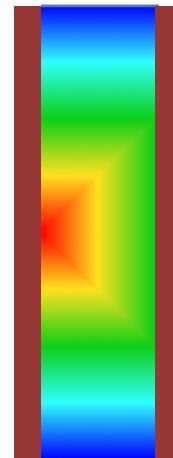
P_0



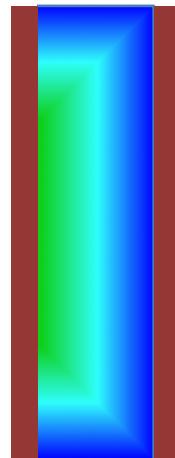
P_1



P_2



P_3



call evolve(dtfact)

Distributed Matrix /1

[...]

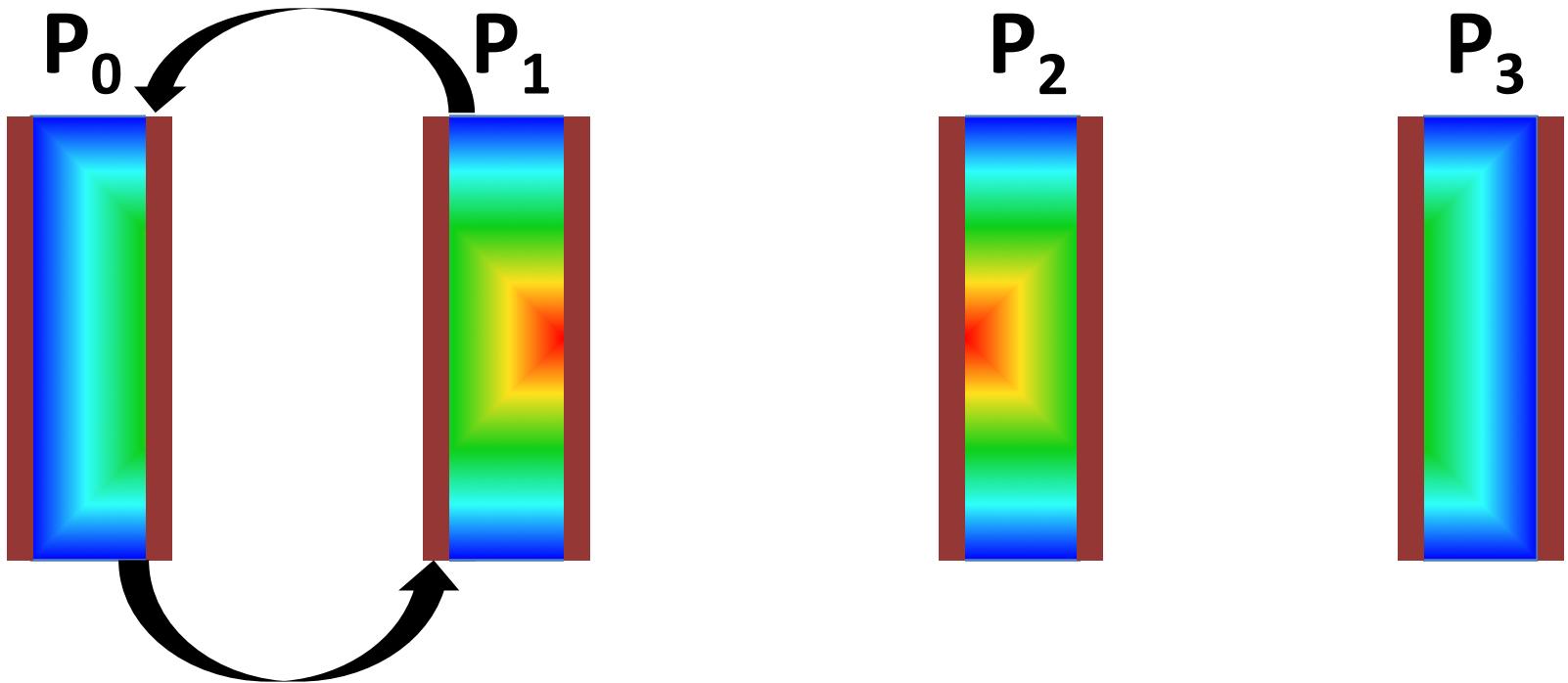
```
int size_loc = SIZE / npes;
```

```
// NO!!!! double * mat = malloc( SIZE * SIZE *  
sizeof(double) );
```

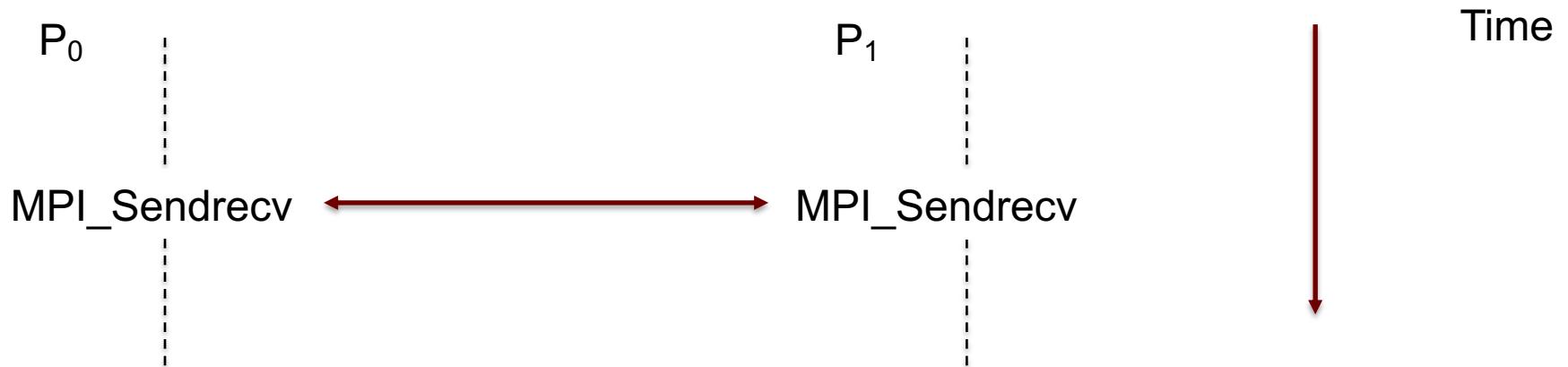
```
double * mat;
```

```
mat = malloc( SIZE * ( size_loc + 2 ) * sizeof(double) );
```

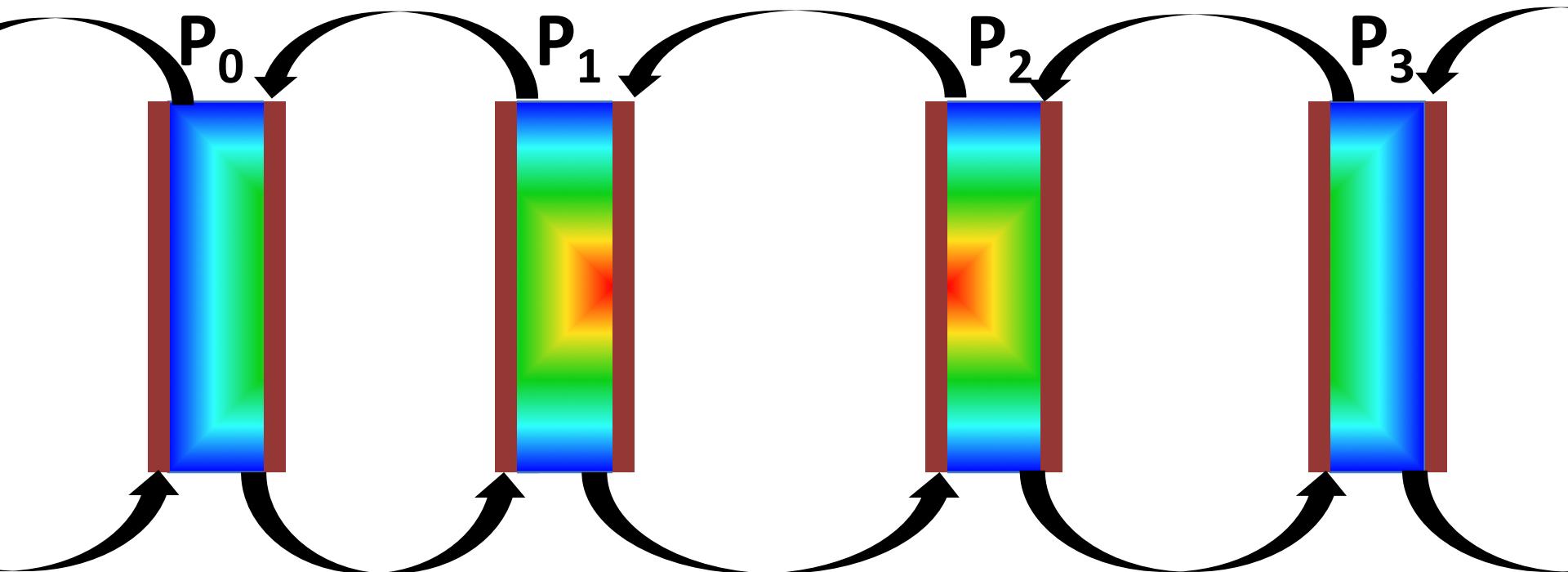
Data exchange among processes



Efficient Data Exchange Between 2 Processes



```
[...]  
  
int * buf_s = (int *) malloc( N * sizeof(int) );  
  
int * buf_r = (int *) malloc( N * sizeof(int) );  
  
init_buffer( buf_s, N );  
  
MPI_Sendrecv( buf_s, N, MPI_INT, !rank, 100, buf_r, N, MPI_INT, !rank, 100,  
             MPI_COMM_WORLD, MPI_STATUS_IGNORE );
```

$$\text{proc_down} = \text{mod}(\text{proc_me} - 1 + \text{nprocs}, \text{nprocs})$$


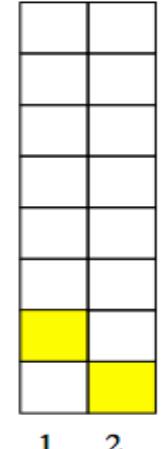
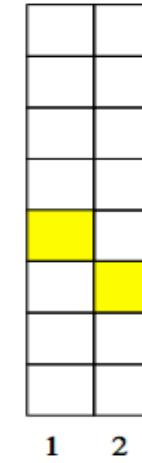
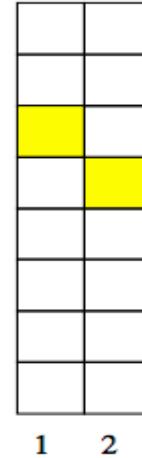
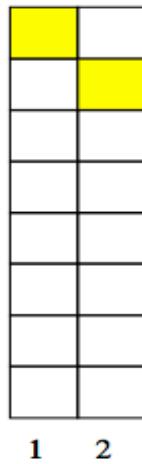
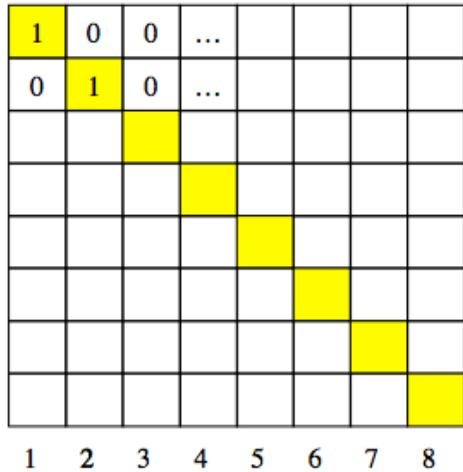
$$\text{proc_up} = \text{mod}(\text{proc_me} + 1, \text{nprocs})$$



Distributed Data

- Global and Local Indexes
- Ghost Cells Exchange Between Processes
 - Compute Neighbor Processes
- Parallel Output

Collaterals to Domain Decomposition /1



Are all the domain's dimensions always multiple of the number of tasks/processes we are willing to use?

Identity Matrix /1

```
[...]
int size_loc = SIZE / npes;

rest = offset = SIZE % npes;

if( rank < rest ){
    size_loc += 1; offset = 0;
}

double * mat = malloc( SIZE * size_loc * sizeof(double) );
int i, j;

// Initialization of the distributed Indenty Matrix
for( i = 0; i < size_loc; i++ ){
    for( j = 0; j < SIZE; j++ ){

        int i_global = rank * size_loc + i + offset;

        if( i_global == j ) mat[ i * SIZE + j ] = 1.0;
        else mat[ i * SIZE + j ] = 0.0;
    }
}
```

Fundamental Tools of Parallel Programming

