Think parallel!

Why?

Reducing the elapsed time by distributing the tasks!

$$\sum_{i=1}^{N} A_i * B_i$$

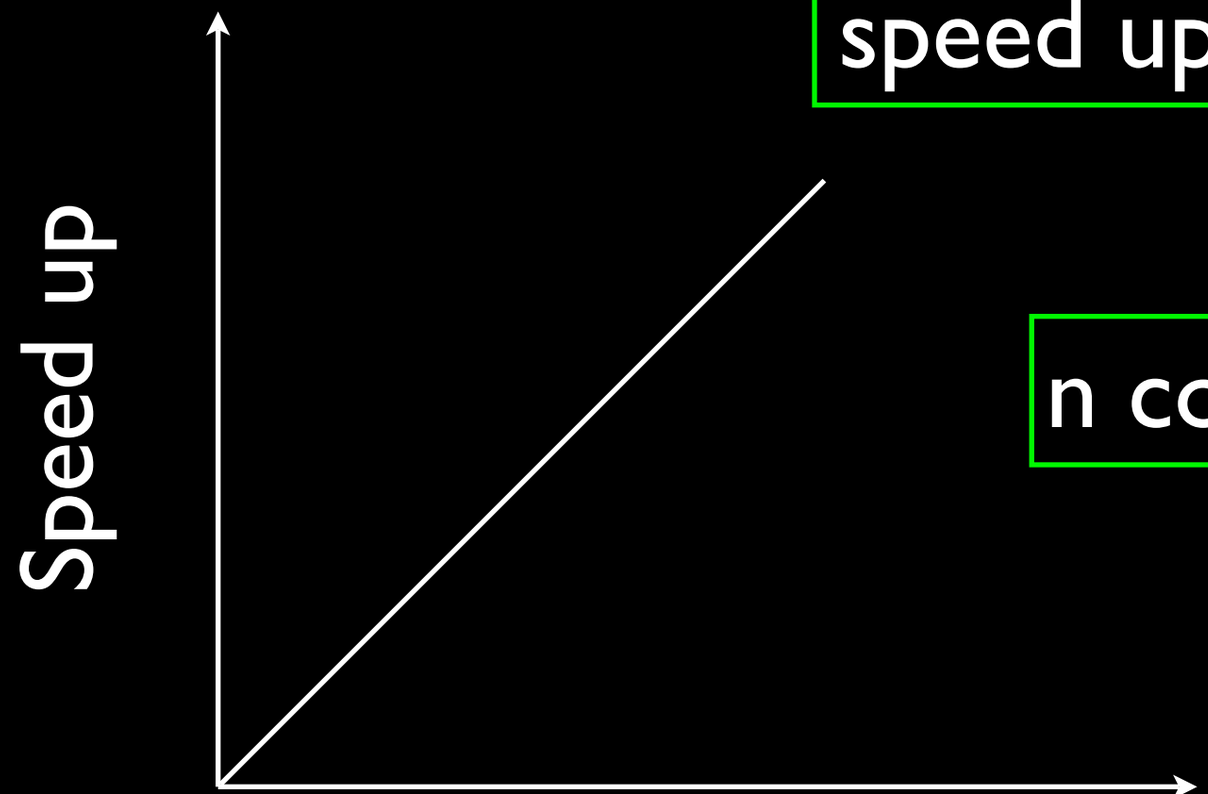
- Distribute the work: todos los cores tiene todos los valores de A y B. El indice de la suma viene distribuido.

- Distribute the work and data: cada core tiene una parte de A y B. El indice de la suma viene distribuido.

Scalability

Measure of how "good" the tasks have been distributed

speed up = $eTime_{n\ cores}/eTime_{n\ cores*m}$

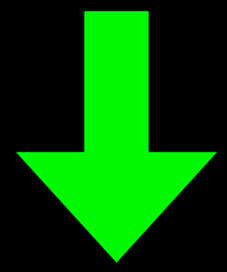


n cores *m -> speed up *m

Perfect scalability

Scalability

Measure of how "good" the tasks have been distributed



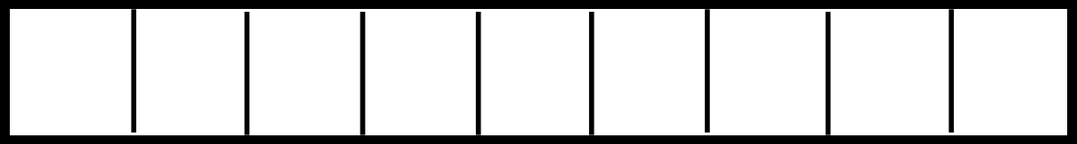
depends on the equations

- Equations naturally parallel
- Equations that are not naturally parallel

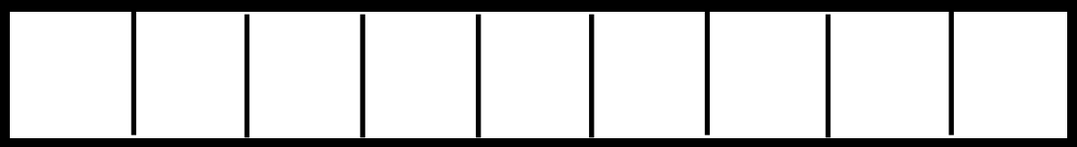
$$\sum_{i=1}^{N} A_i * B_i$$

```
do i=1, N
sum =sum + A(i)*B(i)
enddo
```

A



B

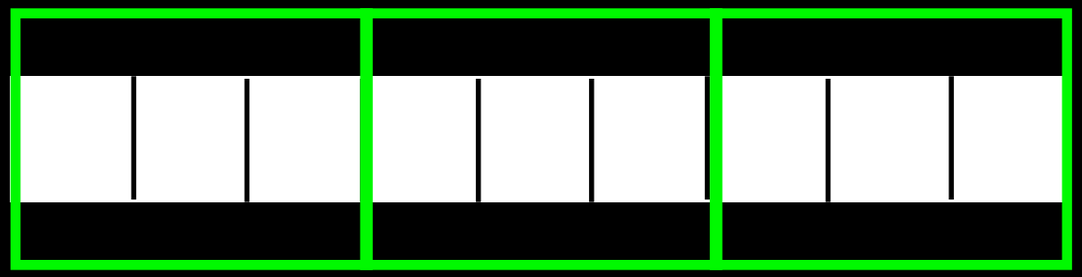


```
allocate(A(1,N), B(1,N))
fill A and B
do i=1, N/nproc
sum =sum + A(i)*B(i)
enddo
mpi_reduce(sum)
```

$$\sum_{i=1}^{N} A_i * B_i$$

```
do i=1, N
sum =sum + A(i)*B(i)
enddo
```

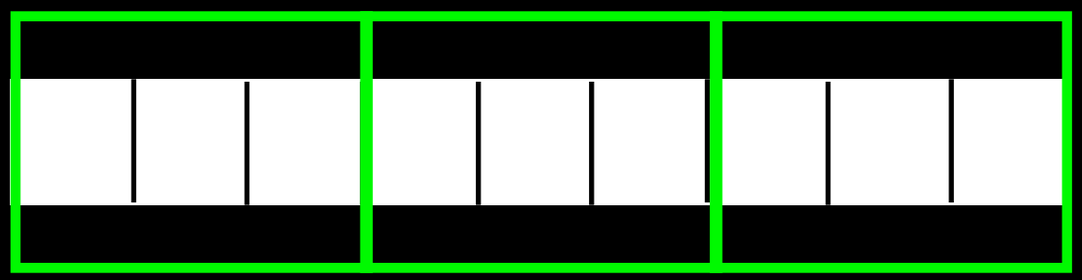
A



1 2 3

```
allocate(A(1,N/core),
B(1,N/core))
fill A and B
do i=1, N/core
sum =sum + A(i)*B(i)
enddo
mpi_reduce(sum)
```

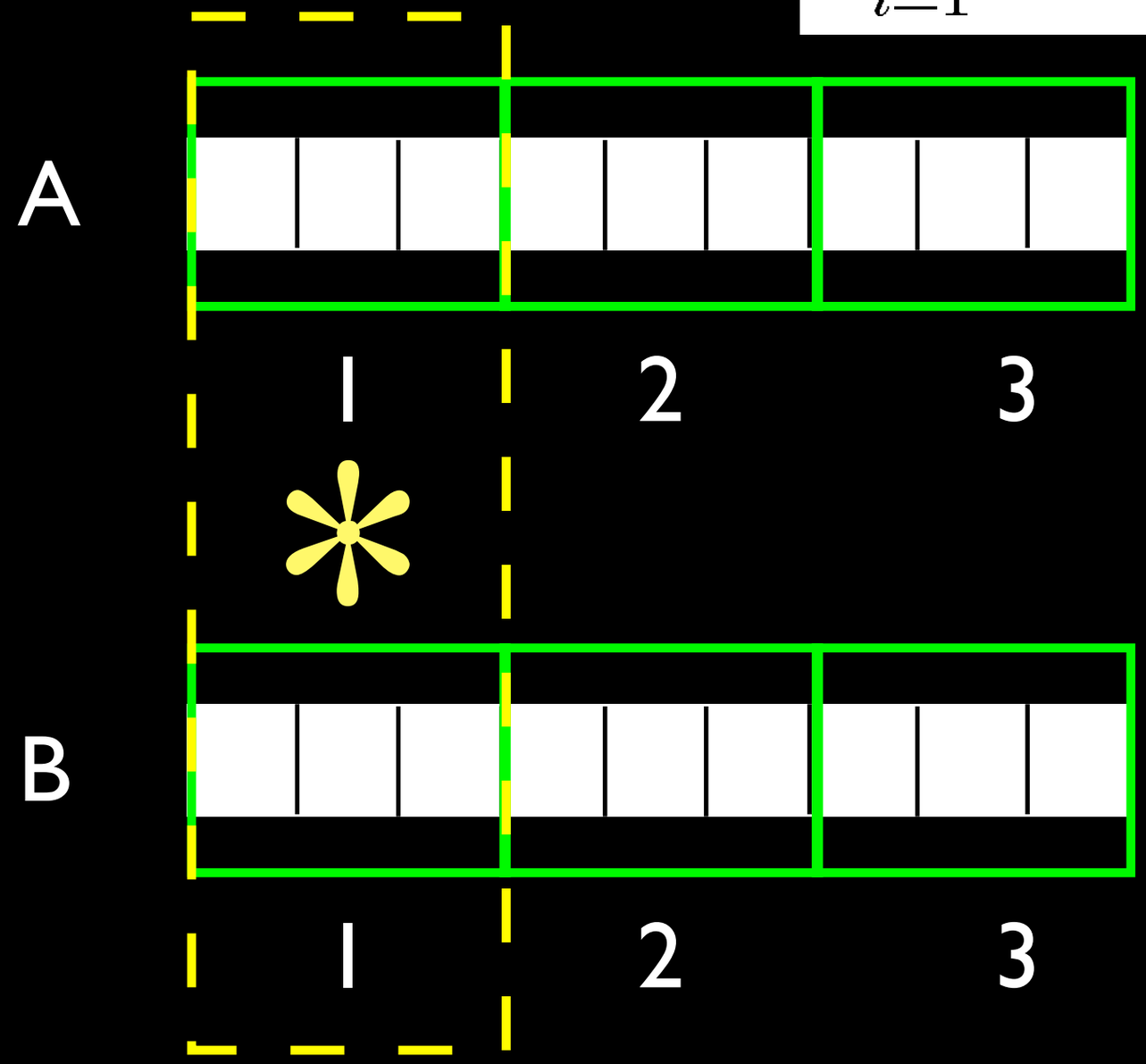
B



1 2 3

$$\sum_{i=1}^{N} A_i * B_i$$

```
do i=1, N
sum =sum + A(i)*B(i)
enddo
```




```
allocate(A(1,N/core),
B(1,N/core))
fill A and B
do i=1, N/core
sum =sum + A(i)*B(i)
enddo
mpi_reduce(sum)
```

No communication!

$$\sum_{i=1}^{N} A_i * B_i$$

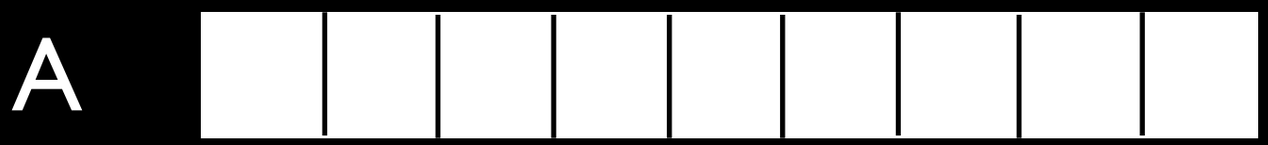
The problem is naturally parallel
naturally scalable

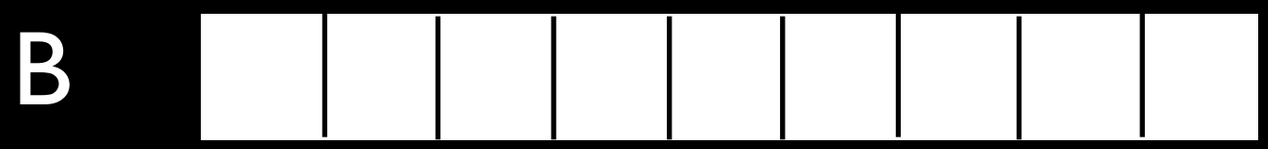
$$\sum_{i=1}^{N}\sum_{j=1}^{N} A_i * B_j$$

?

$$\sum_{i=1}^{N}\sum_{j=1}^{N} A_i * B_j$$

```
do i=1, N
  do j=1,N
     sum =sum + A(i)*B(j)
  enddo
enddo
```

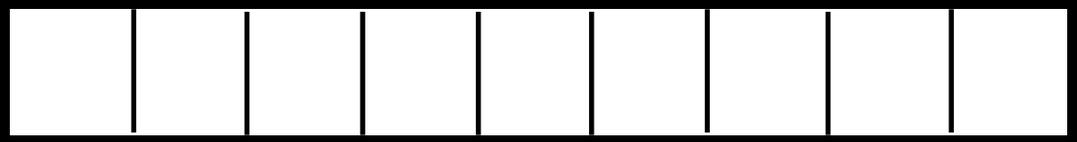




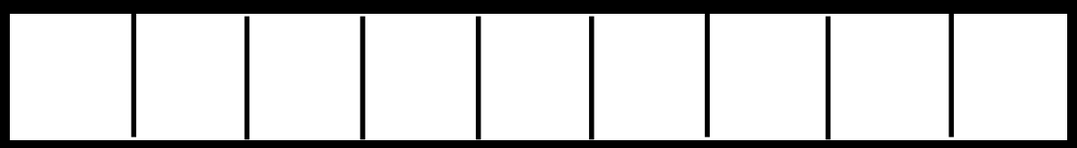

```
allocate(A(1,N), B(1,N))
fill A and B
do i=1, N/nproc
  do j=1,N
     sum =sum + A(i)*B(i)
  enddo
enddo
mpi_reduce(sum)
```

$$\sum_{i=1}^{N}\sum_{j=1}^{N} A_i * B_j$$

A



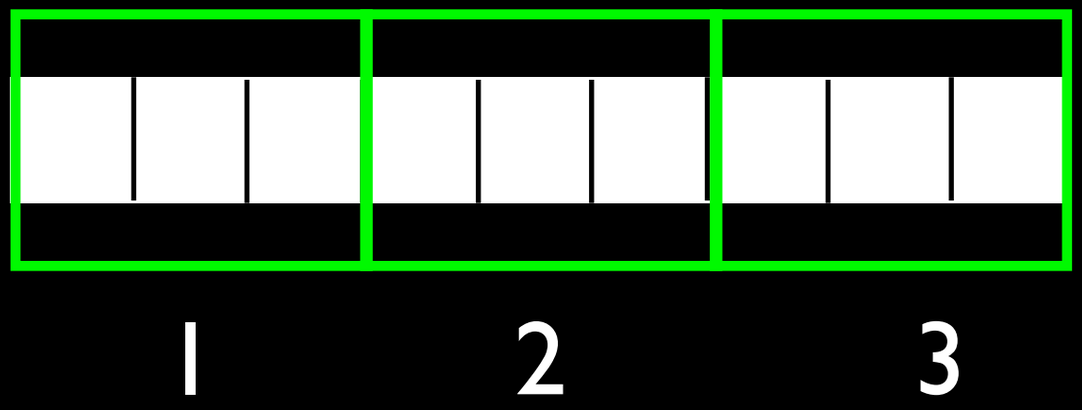
B

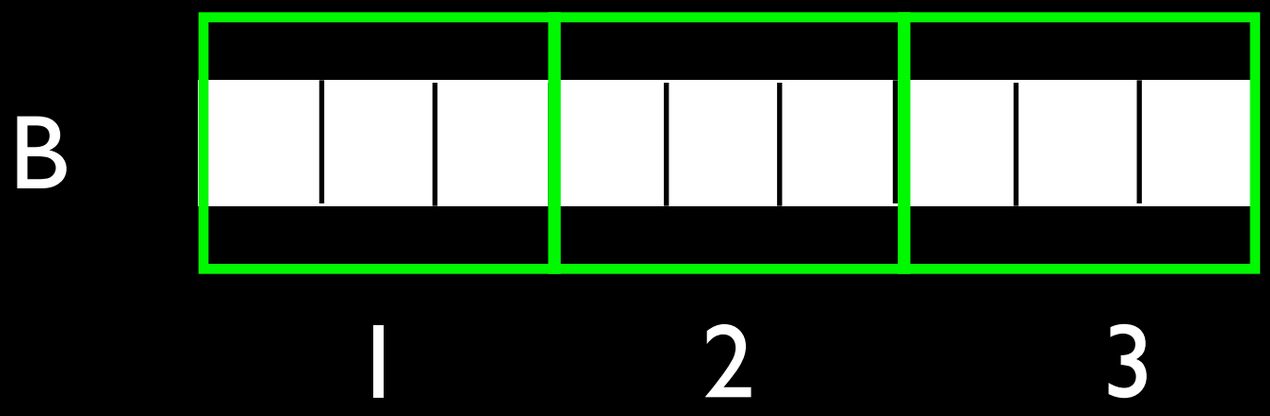


```
do i=1, N
  do j=1,N
     sum =sum + A(i)*B(j)
  enddo
enddo
```

```
allocate(A(1,N), B(1,N))
fill A and B
do i=1, N/nproc
  do j=1,N
     sum =sum + A(i)*B(i)
  enddo
enddo
mpi_reduce(sum)
```

$$\sum_{i=1}^{N}\sum_{j=1}^{N} A_i * B_j$$

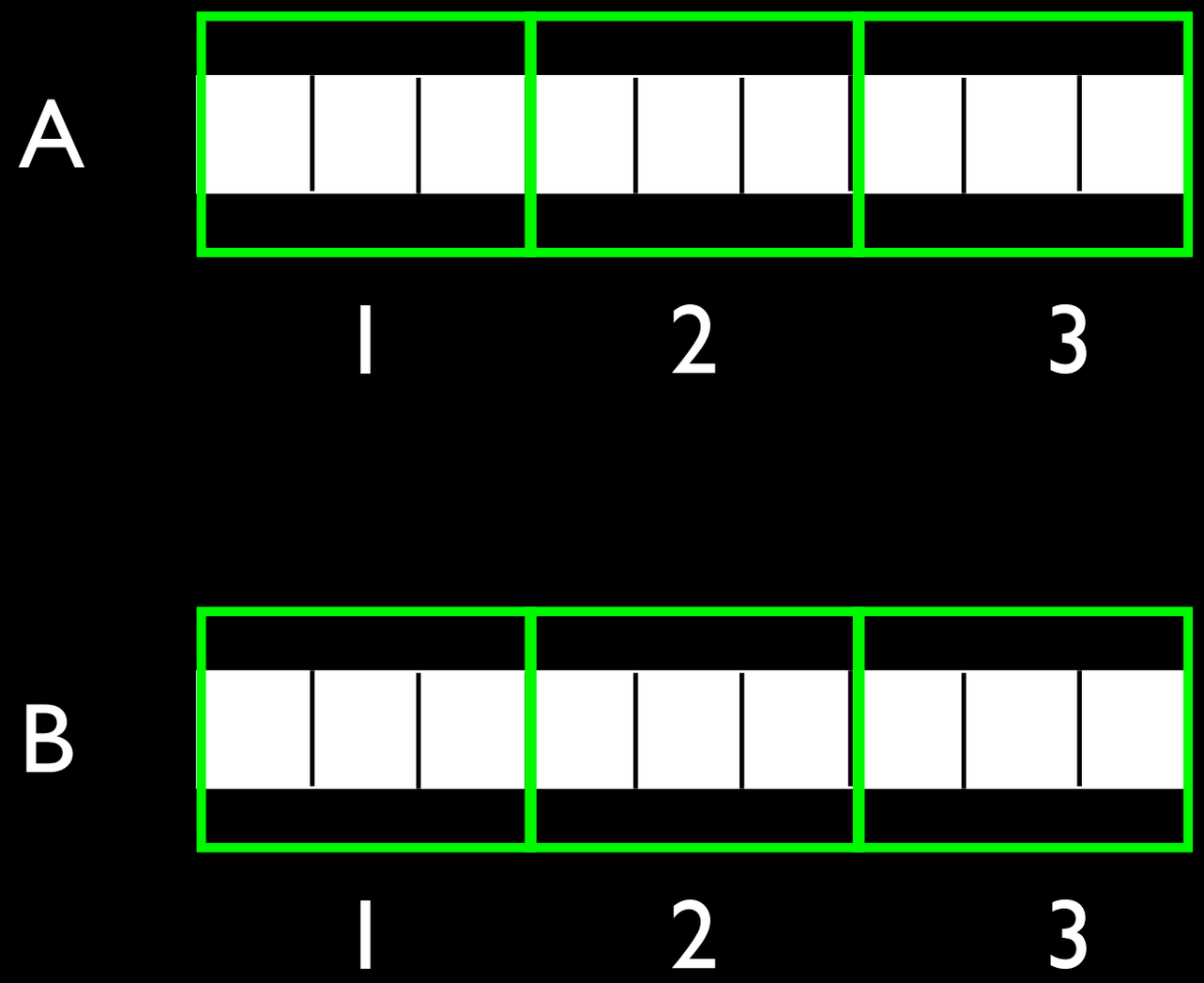





```
do i=1, N
  do j=1,N
     sum =sum + A(i)*B(j)
  enddo
enddo
```

```
allocate(A(1,N), B(1,N))
fill A and B
do i=1, N/nproc
  do j=1,N
     sum =sum + A(i)*B(i)
  enddo
enddo
mpi_reduce(sum)
```

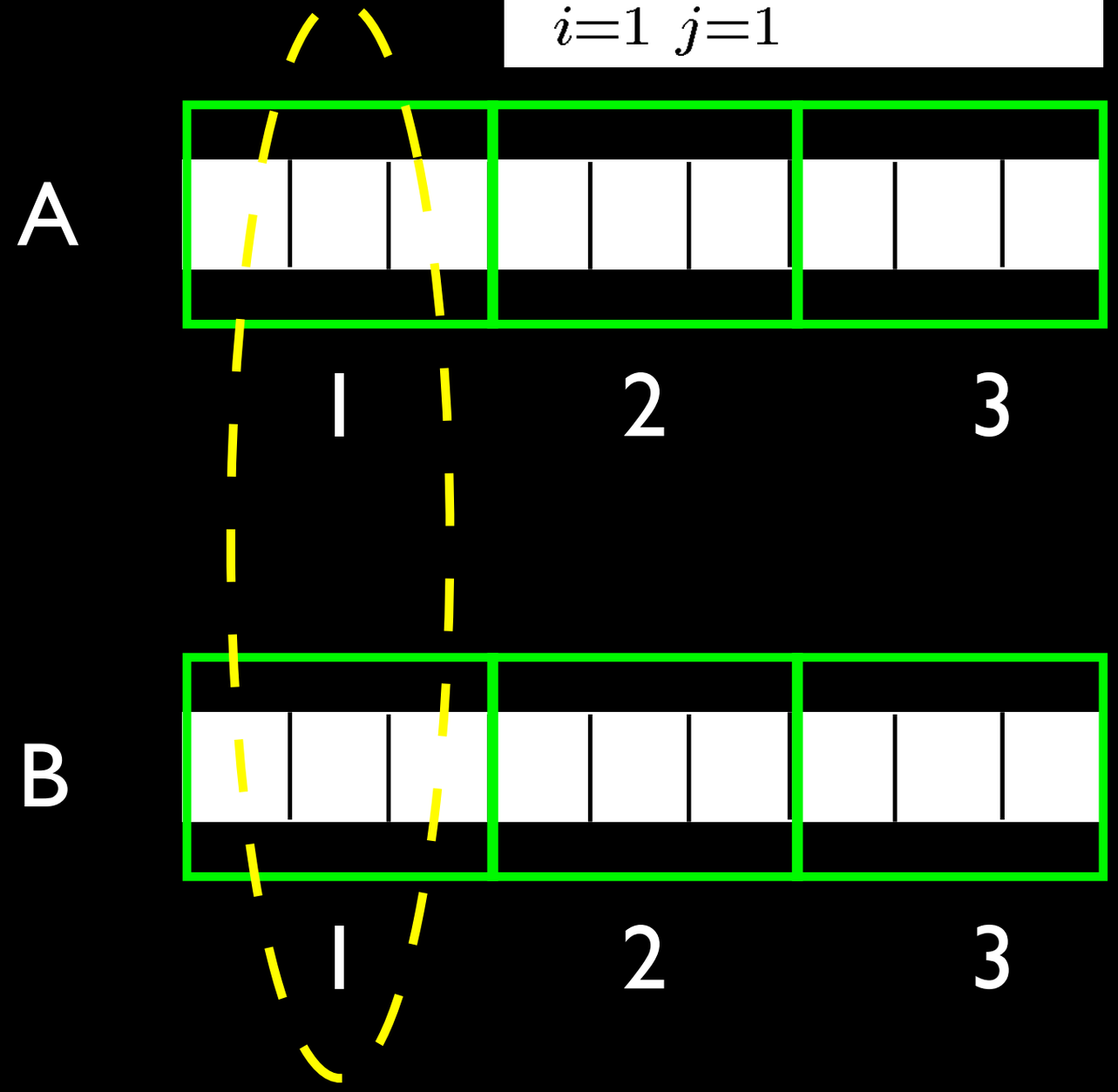
$$\sum_{i=1}^{N}\sum_{j=1}^{N} A_i * B_j$$




```
do i=1, N
  do j=1,N
     sum =sum + A(i)*B(j)
  enddo
enddo
```

```
allocate(A(1,N), B(1,N))
fill A and B
do i=1, N/nproc
  do j=1,N
    send/receive(B)
    sum =sum + A(i)*B(i)
  enddo
enddo
mpi_reduce(sum)
```

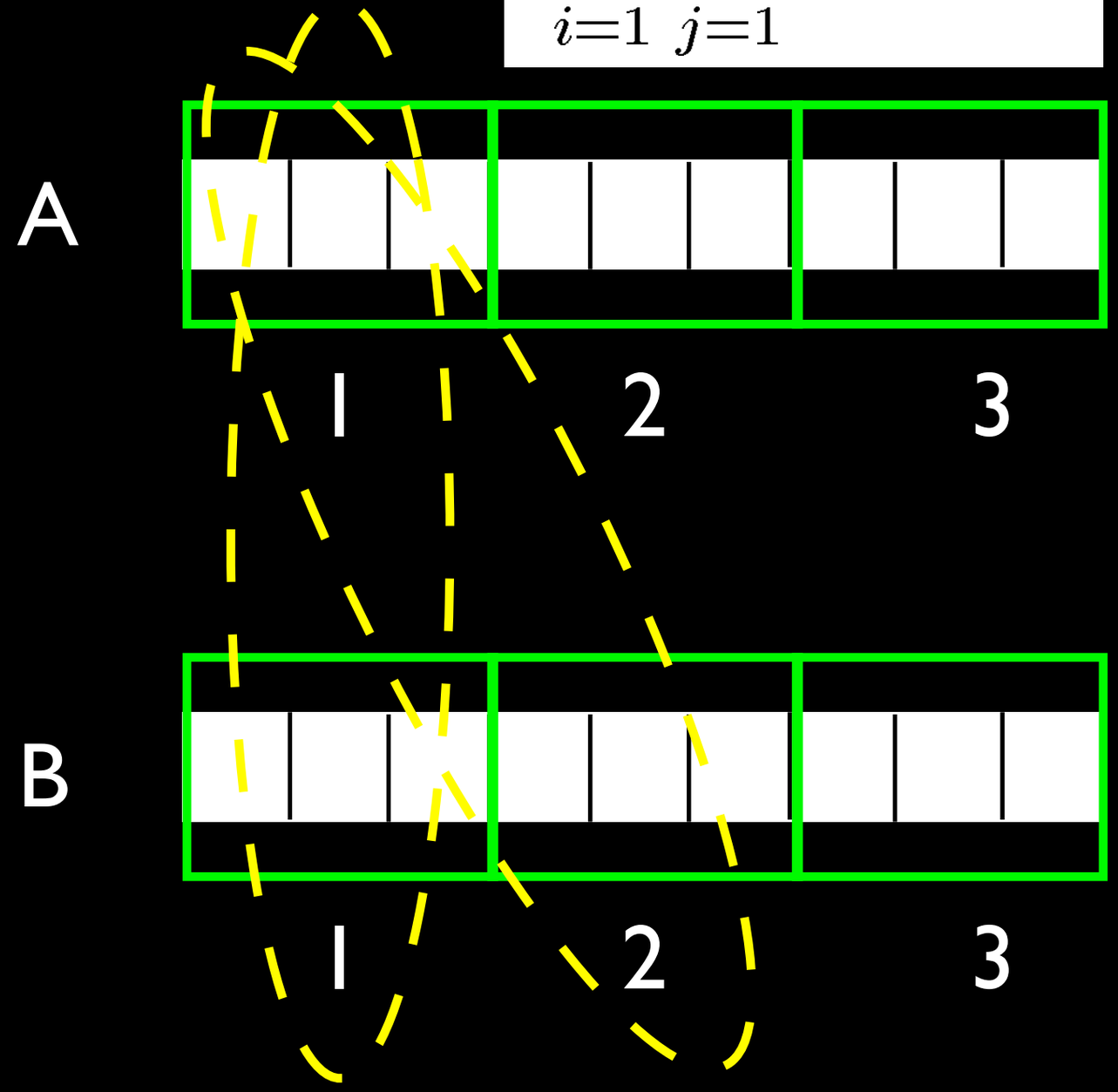
$$\sum_{i=1}^{N}\sum_{j=1}^{N} A_i * B_j$$




```
do i=1, N
  do j=1,N
     sum =sum + A(i)*B(j)
  enddo
enddo
```

```
allocate(A(1,N), B(1,N))
fill A and B
do i=1, N/nproc
  do j=1,N
    send/receive(B)
     sum =sum + A(i)*B(i)
  enddo
enddo
mpi_reduce(sum)
```

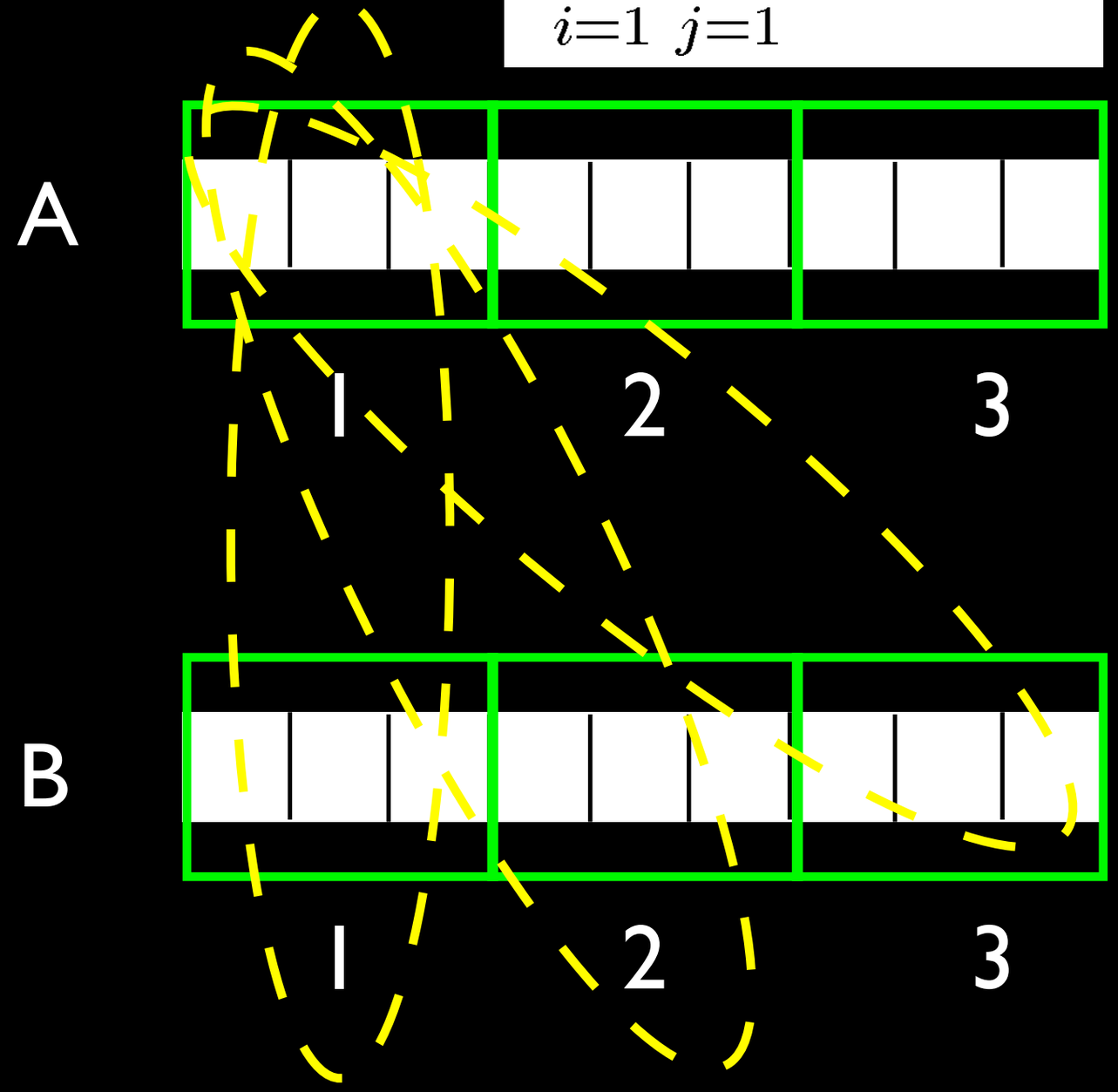
$$\sum_{i=1}^{N}\sum_{j=1}^{N} A_i * B_j$$




```
do i=1, N
  do j=1,N
     sum =sum + A(i)*B(j)
  enddo
enddo
```

```
allocate(A(1,N), B(1,N))
fill A and B
do i=1, N/nproc
  do j=1,N
    send/receive(B)
     sum =sum + A(i)*B(i)
  enddo
enddo
mpi_reduce(sum)
```

$$\sum_{i=1}^{N}\sum_{j=1}^{N} A_i * B_j$$




```
do i=1, N
  do j=1,N
     sum =sum + A(i)*B(j)
  enddo
enddo
```

```
allocate(A(1,N), B(1,N))
fill A and B
do i=1, N/nproc
  do j=1,N
    send/receive(B)
     sum =sum + A(i)*B(i)
  enddo
enddo
mpi_reduce(sum)
```

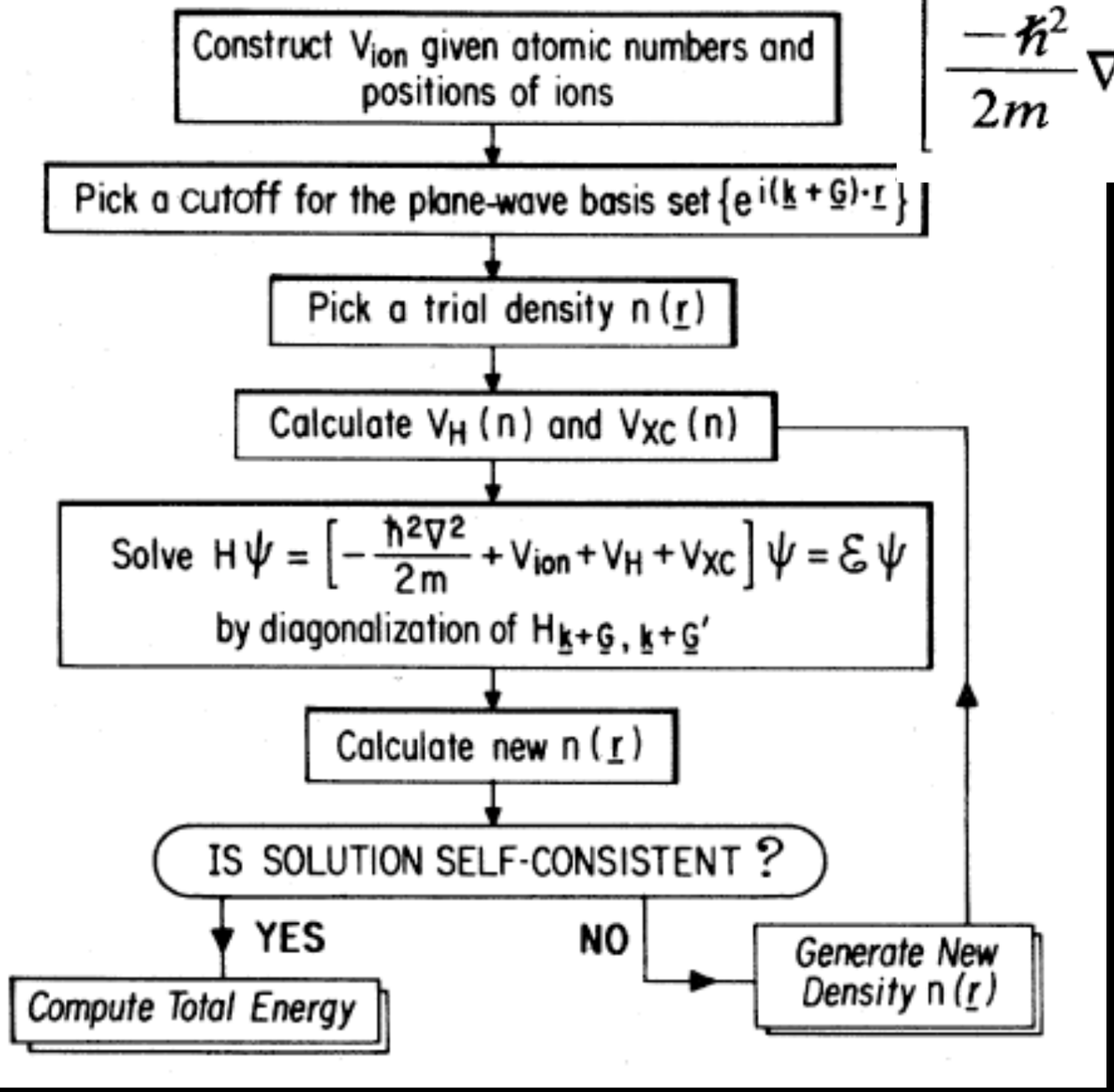
$$\sum_{i=1}^{N} \sum_{j=1}^{N} A_i * B_j$$

The problem is not naturally parallel
not naturally scalable

Balance between memory distribution and speed up

memory less distributed -> high speed up

DFT is naturally parallel?

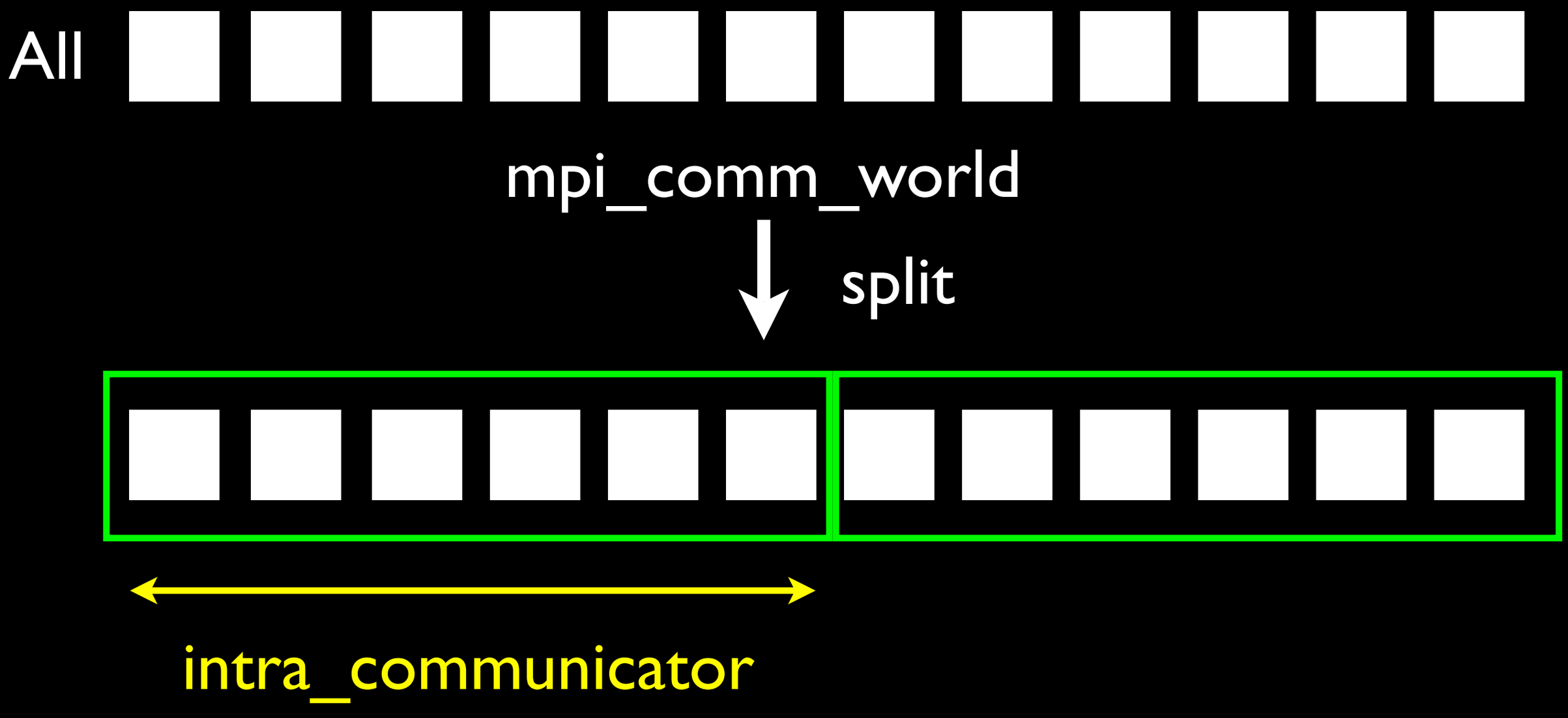


$$\left[\frac{-\hbar^2}{2m}\nabla^2 + V_{\text{ion}}(\mathbf{r}) + V_H(\mathbf{r}) + V_{XC}(\mathbf{r})\right]\psi_i(\mathbf{r}) = \varepsilon_i\psi_i(\mathbf{r})$$

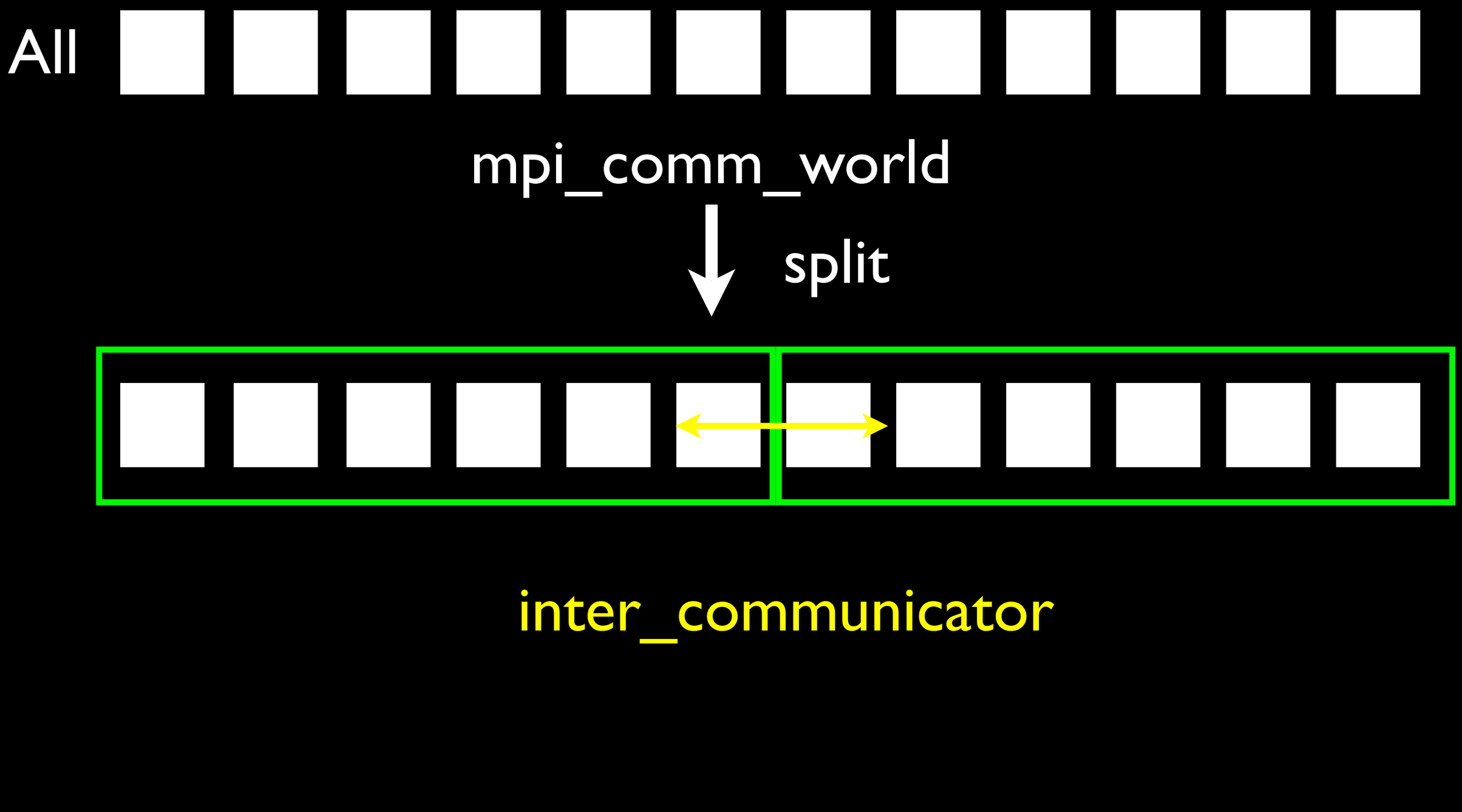
$$n(r) = \sum_i f_i \,|\Psi_i(r)|^2$$

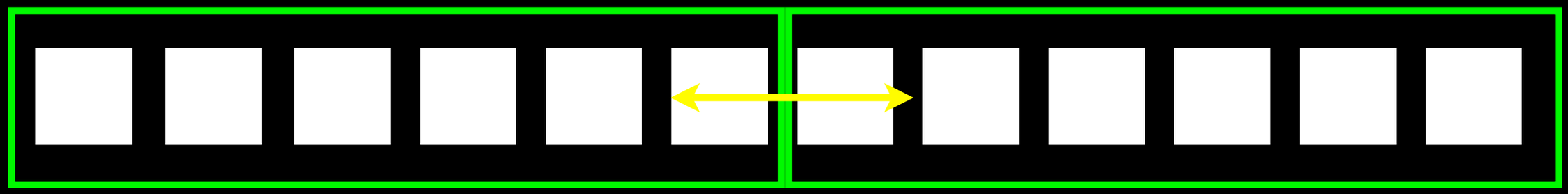
$$\sum_{\mathbf{G}'}\left[\frac{\hbar^2}{2m}|\mathbf{k}+\mathbf{G}|^2\delta_{\mathbf{G}\mathbf{G}'} + V_{\text{ion}}(\mathbf{G}-\mathbf{G}') + V_H(\mathbf{G}-\mathbf{G}') + V_{XC}(\mathbf{G}-\mathbf{G}')\right]c_{i,\mathbf{k}+\mathbf{G}'} = \varepsilon_i c_{i,\mathbf{k}+\mathbf{G}}$$

Parallelization levels: communicators

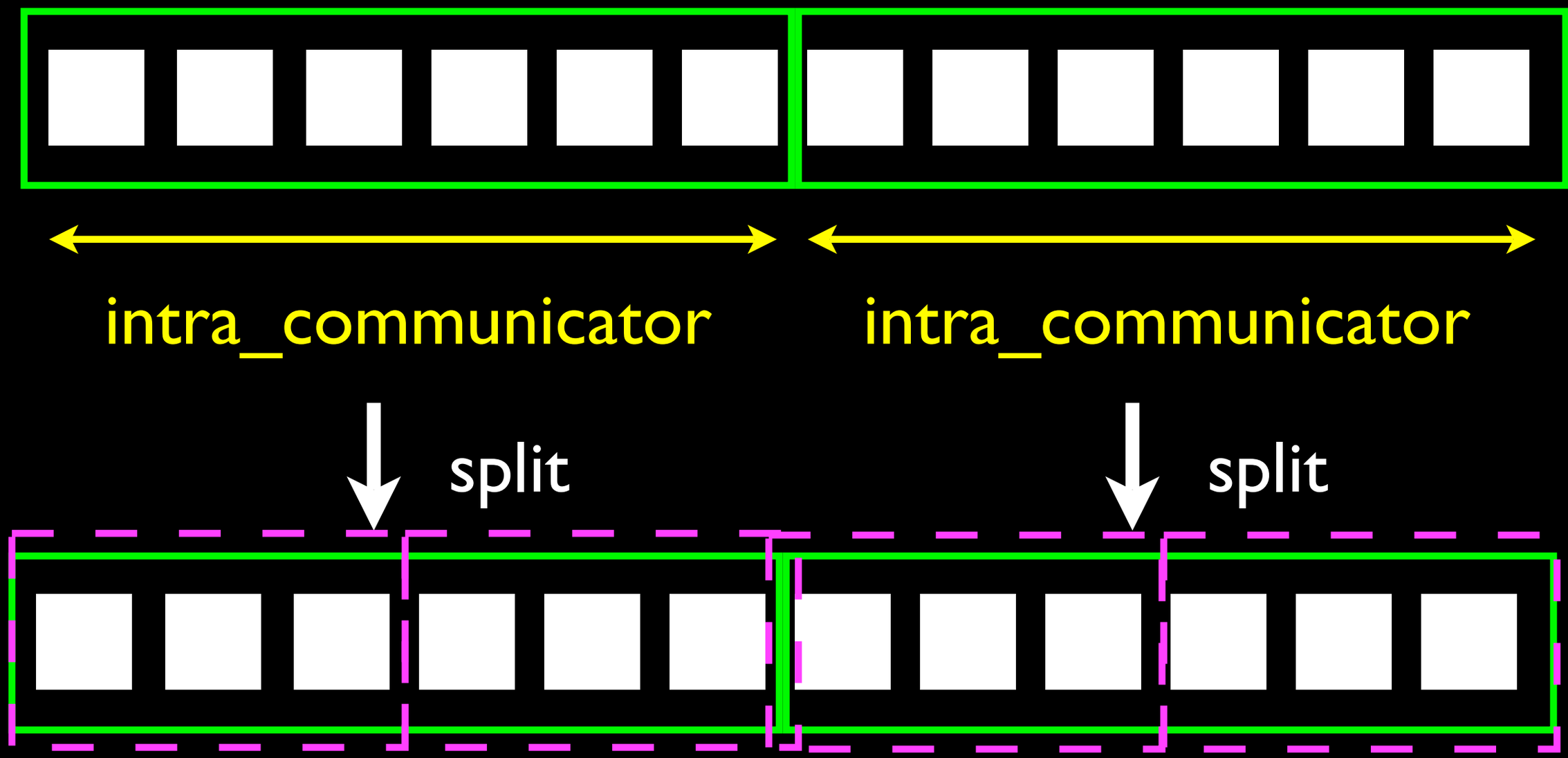


Parallelization levels: communicators



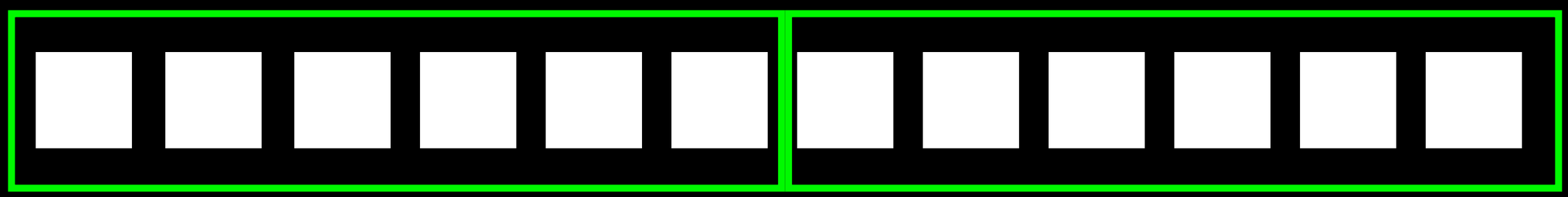


	0	1	2	3	4	5
0	0	1	2	3	4	5
1	6	7	8	9	10	11


intra_communicator
intra_communicator
split
split

$$f(k) = [\sum_{i=1}^{N} A_i * B_i] * k$$

```
do ik=1, nk
  sum = 0
  do i=1, N
       sum =sum + A(i)*B(i)
  enddo
  f(ik)=sum*k(ik)
enddo
```

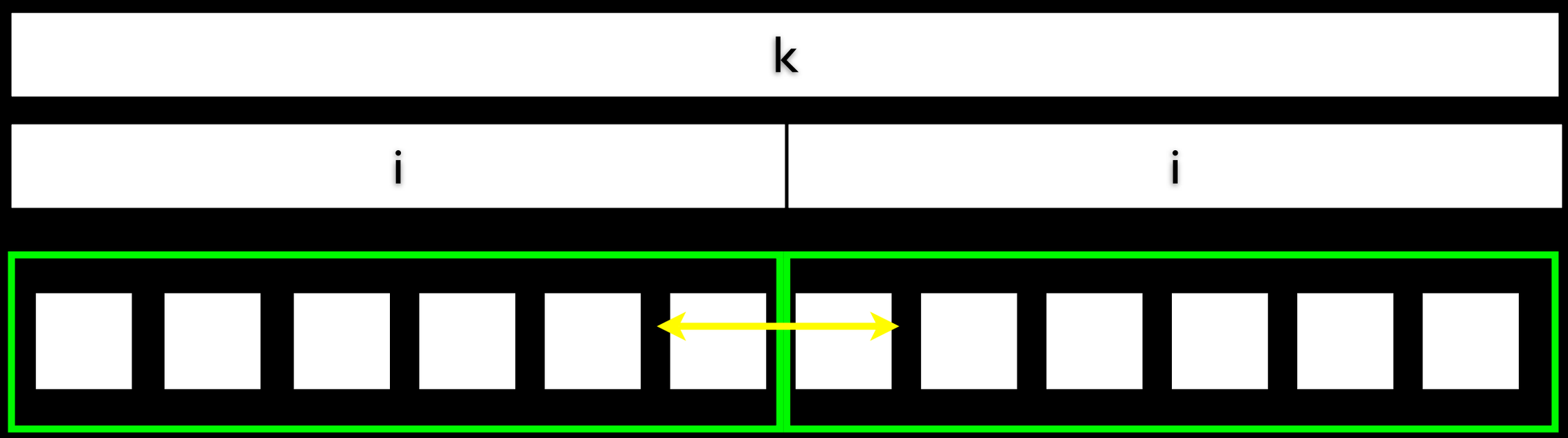


```
do ik=1, nk
  sum = 0
  do i=1, N
       sum =sum + A(i)*B(i)
  enddo
  f(ik)=sum*k(ik)
enddo
```

```
do ik=1, nk/size_inter_comm_1
  sum = 0
  do i=1, N/size_intra_comm_1
       sum =sum + A(i)*B(i)
  enddo
  mpi_reduce(intra_comm_1)
  f(ik)=sum*k(ik)
  mpi_reduce(inter_comm_1)
enddo
```

```
do ik=1, nk
  sum = 0
  do i=1, N
       sum =sum + A(i)*B(i)
  enddo
  f(ik)=sum*k(ik)
enddo
```

```
do ik=1, nk/size_inter_comm_1
  sum = 0
  do i=1, N/size_intra_comm_1
       sum =sum + A(i)*B(i)
  enddo
  mpi_reduce(intra_comm_1)
  f(ik)=sum*k(ik)
  mpi_reduce(inter_comm_1)
enddo
```



?

$$f(k) = [\sum_{i=1}^{N} A_i * B_i] * k$$

scalability?

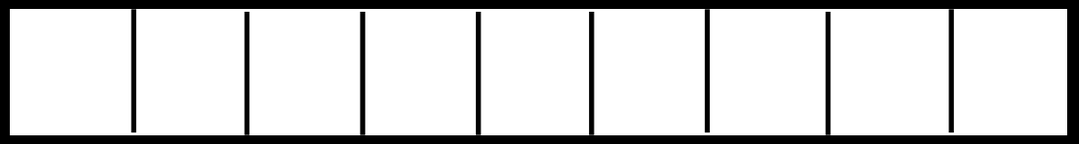
openmp

openmp: programming interface for platforms that allow shared memory multiprocessing

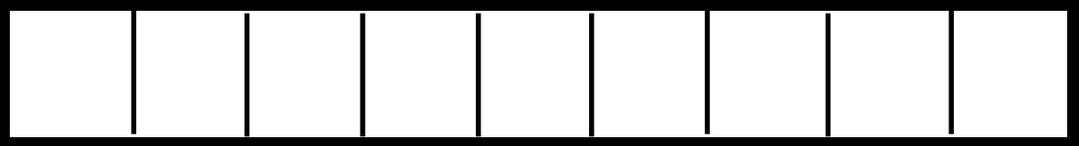
$$\sum_{i=1}^{N} A_i * B_i$$

```
do i=1, N
sum =sum + A(i)*B(i)
enddo
```

A

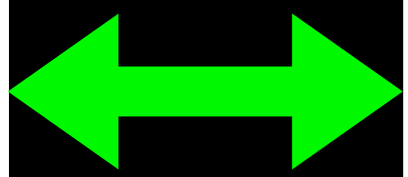


B



```
omp shared
allocate(A(1,N), B(1,N))
fill A and B
omp parallel shared(A,B,sum)
omp parallel private(i)
omp parallel do
do i=1, N
sum =sum + A(i)*B(i)
enddo
```

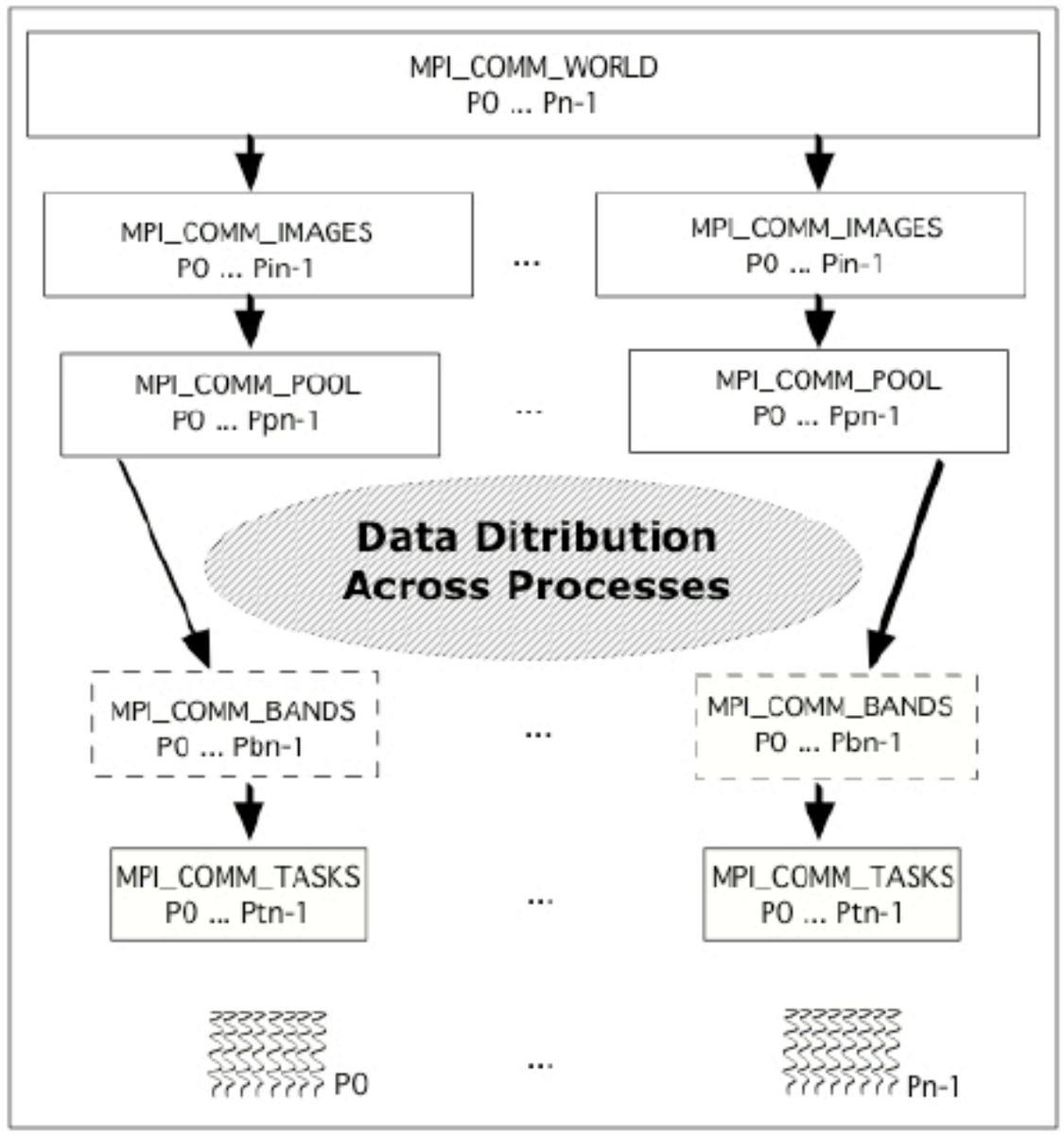
```
omp shared
allocate(A(1,N), B(1,N))
fill A and B
omp parallel shared(A,B,sum)
omp parallel private(i)
omp parallel do
do i=1, N
sum =sum + A(i)*B(i)
enddo
```



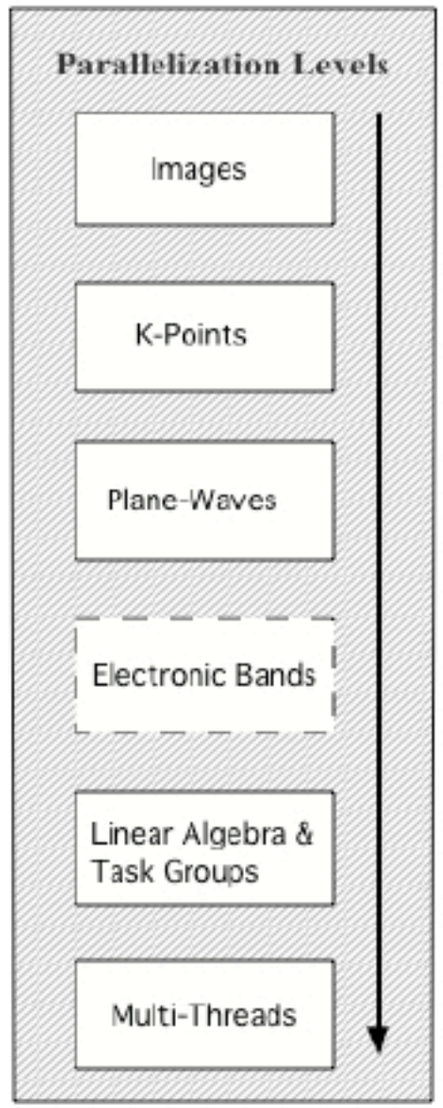
```
allocate(A(1,N), B(1,N))
fill A and B
do i=1, N/nproc
sum =sum + A(i)*B(i)
enddo
mpi_reduce(sum)
```

nproc*memory!!!!!!!

QE parallelization levels



MPI Communicators Hierarchy



you need to know your system and the computer architecture!

bulk silicon

- Converge with respect to the wfc cutoff, and record the time

PWSCF : 2m30.82s CPU 2m36.57s WALL

- Plot wfc_cutoff with respect to time
- Converge with respect to k points, and record the time
- Plot number of k points with respect to time

- Could we expect such results from the equations?
- Why?