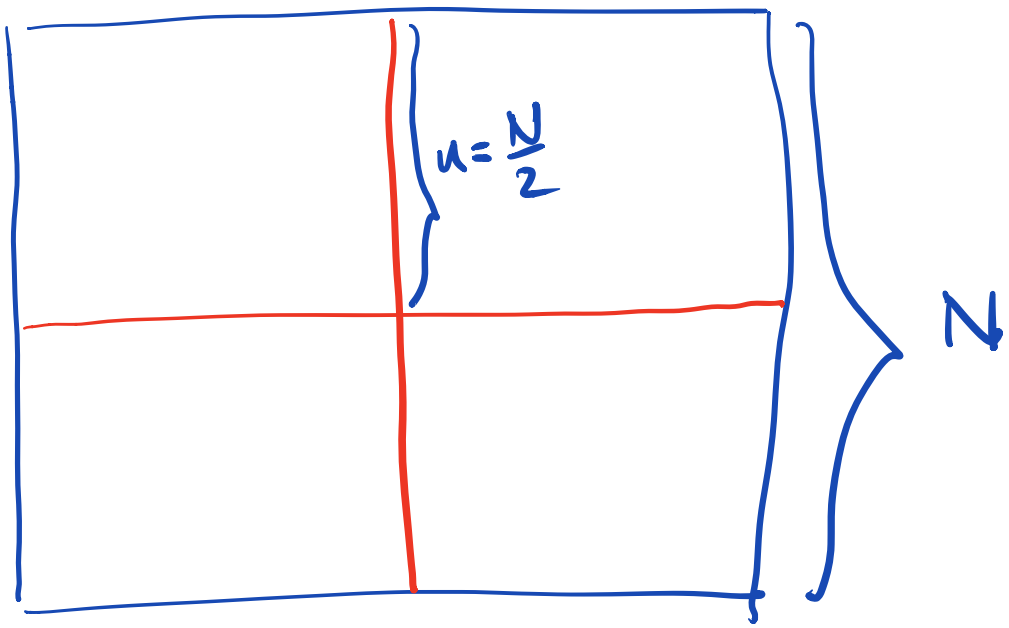
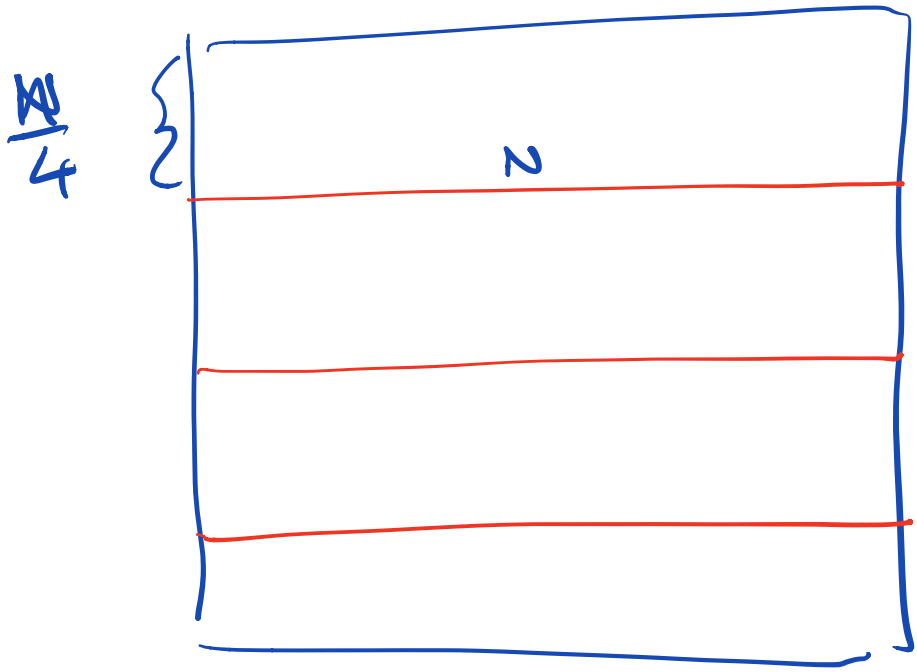
Last time:

Motivated why we want for finite differences, a decomposition that attempts to minimise surface to volume ratio:

$\Rightarrow$ ideally squares/cubes



$\Rightarrow$ each box has $\frac{N^2}{4}$ points and $2n$ side length.

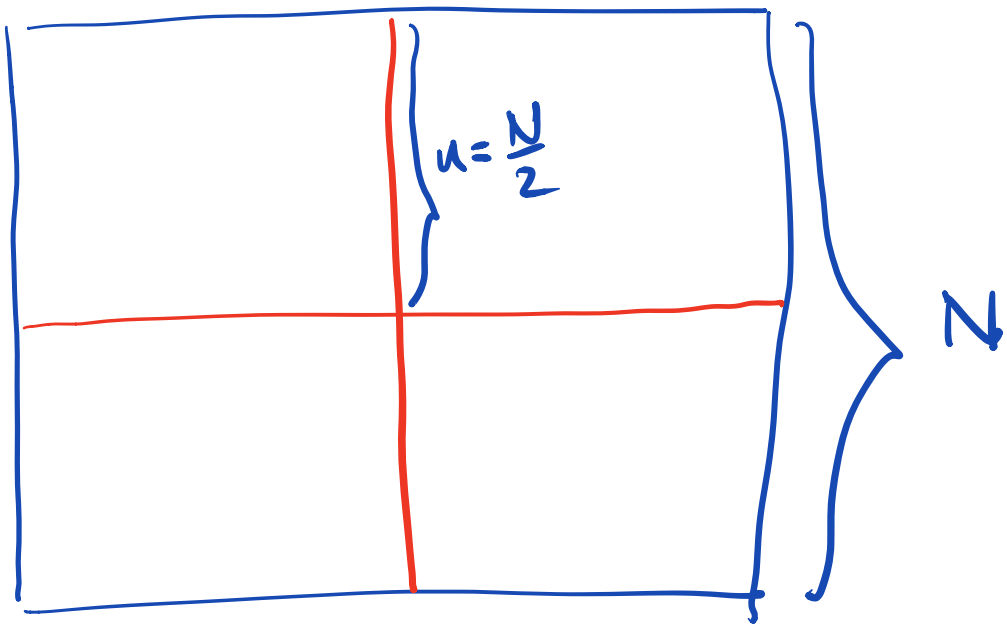


$\frac{N^2}{4}$ pts

$\frac{5}{2}N$ side length.

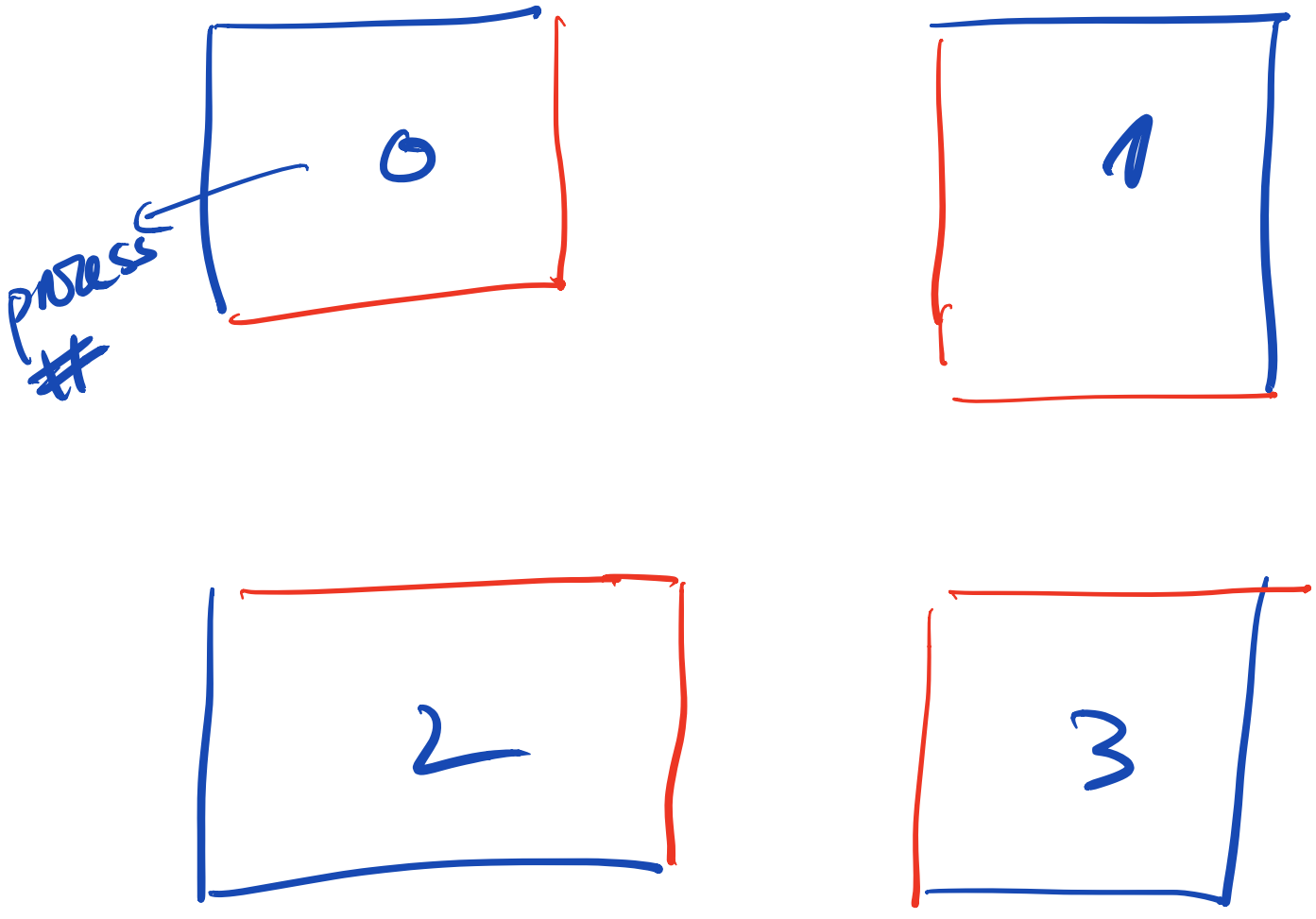
Computation time $\sim \frac{N^2}{4}$ # local pts

Communication volume $\sim N$ side length



MPI: Separate processes that don't share memory space

$\Rightarrow$ decompose grid.



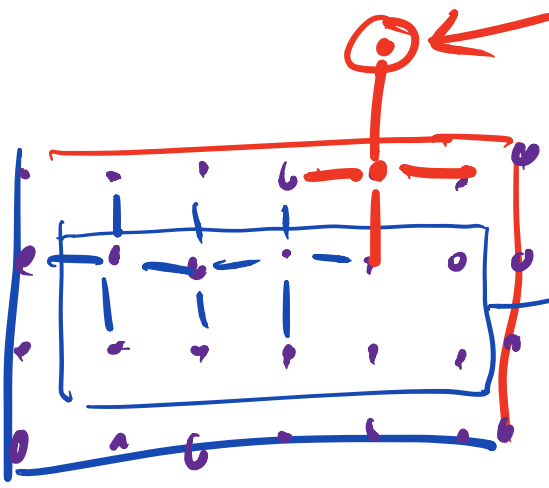
Ω is cannected subdomains.
responsible for computing updates to dofs.

$\Rightarrow \Omega^2 u = \ldots$

Accesses

u[i,j], u[i-1,j], u[i+1,j]

u[i,j+1], u[i,j-1].



don't have this data.

have all the information to compute here.

Context:

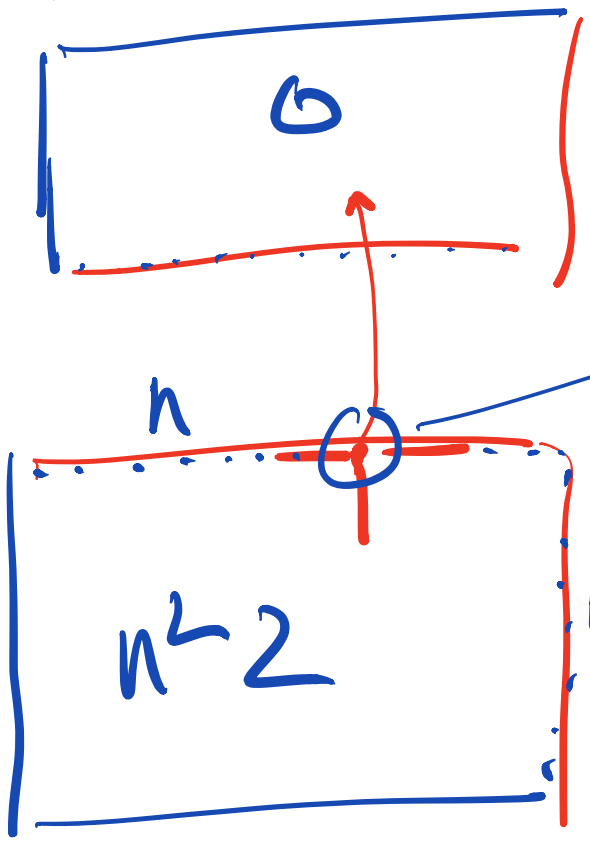
Shared memory model (OpenMP)

→ Not a problem.

→ just read this bit of the array.

→ still need to synchronize in case someone else is updating that part.

MPI:



Update needs to communicate to get val n from process 0.

message exchange.

Aside: these are same interfaces that after distributed one-sided comms ⇔ shared memory → synchronisation (we'll basically ignore these.

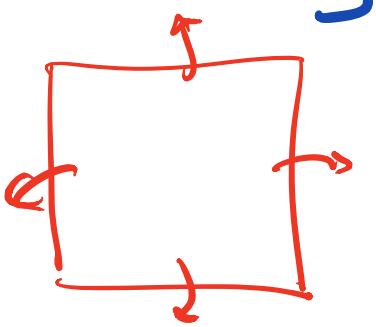
Message exchange synchronized "pairwise".

→ messaging is two sided.

Process 0 says "I am sending this value" MPI_Send

Process 2 says "I am receiving" MPI_Recv "Thanks, I've got it".

How much data? How many messages?



4n messages. each of size 1 double.

Individual messages:

$$4n(t_\alpha + \beta)$$

t_α: message latency; β: inverse bandwidth

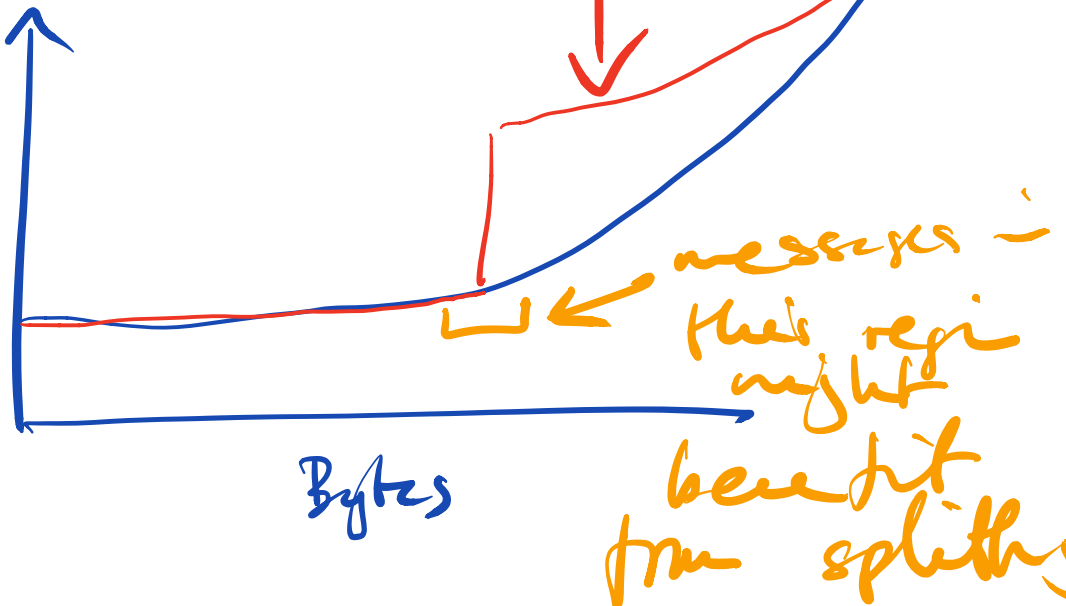
All at once.

$$t_\alpha + \beta 4n$$

$$4nt_\alpha > t_\alpha$$

Conclude ~ batch ~ messages up.

If model is



true

Conclude:

Communicate to exchange all messages.

Compute.

Communicate again.

Grid data structure.

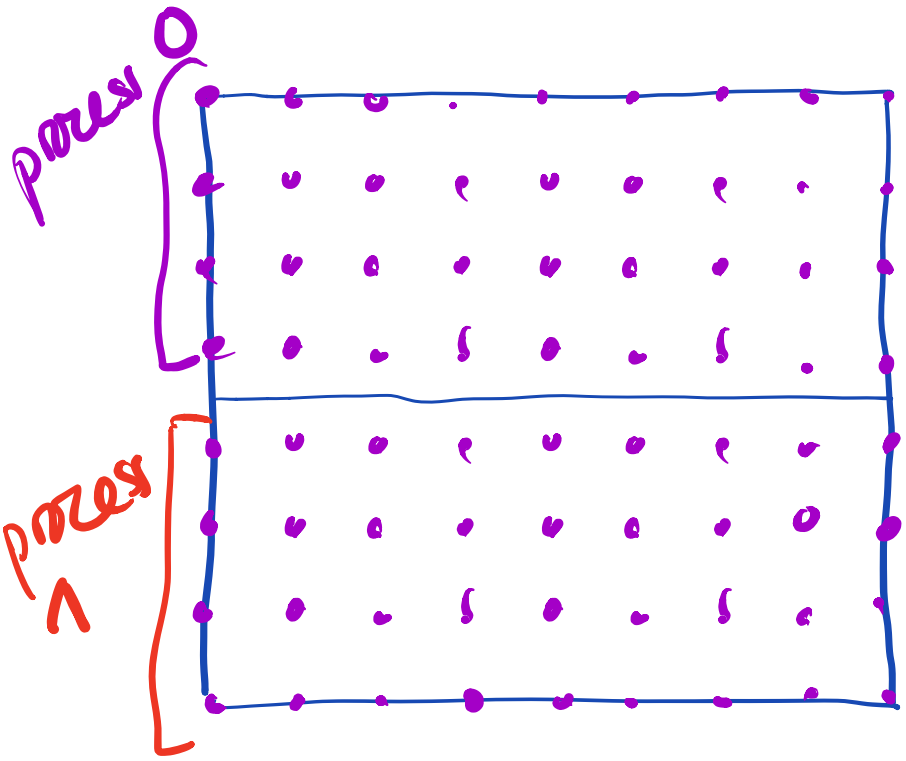
Solver stage: "bookkeeping".

"Assembly" stage: Ax — apply FD stencil.

Time step update.

$$u_{n+1} = u_n + \Delta t \underbrace{A u_n}_{\text{if we have } Au_n \text{ as a vector}}$$

This is pattern.



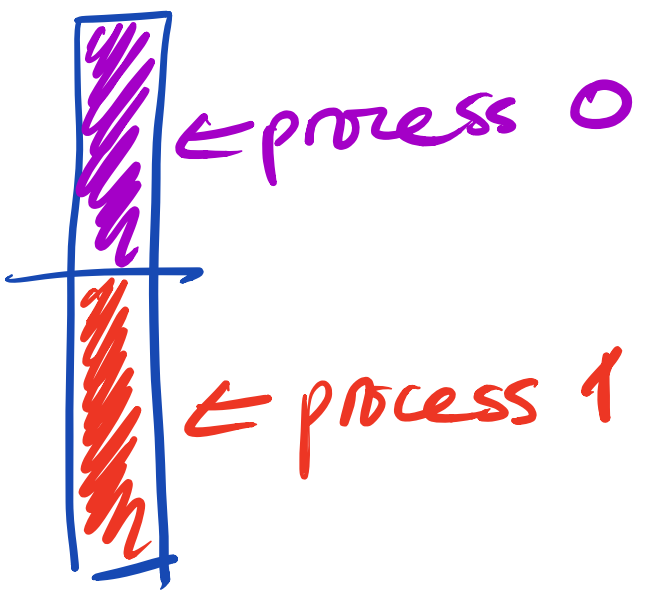
time step update doesn't communicate once Axu is calculated.

$u_{n+1} = \cdot u_n$

↑ pattern stencil.

Update can work on "global" vector.

⇒ all dots are unique.



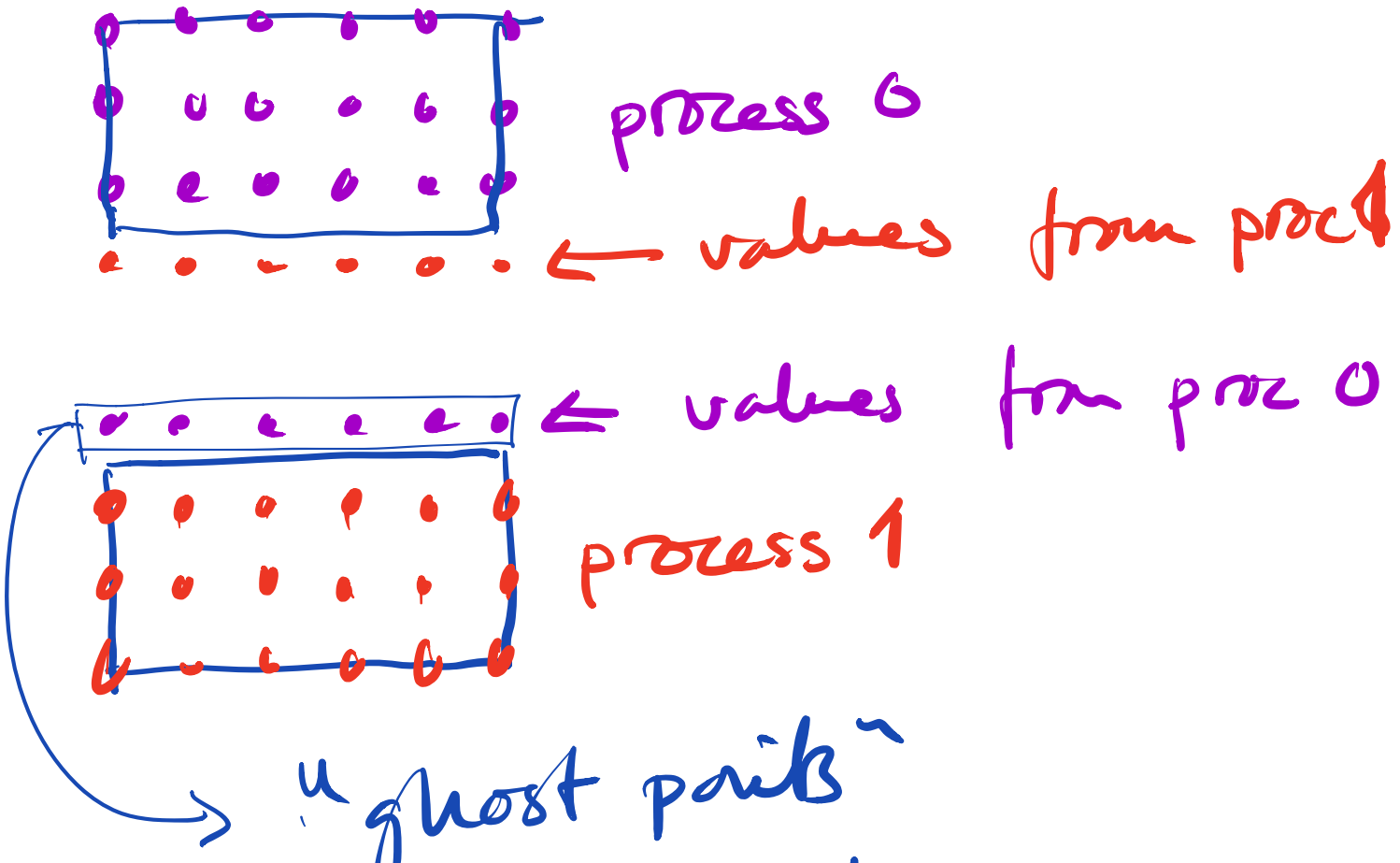
"non-overlapping"

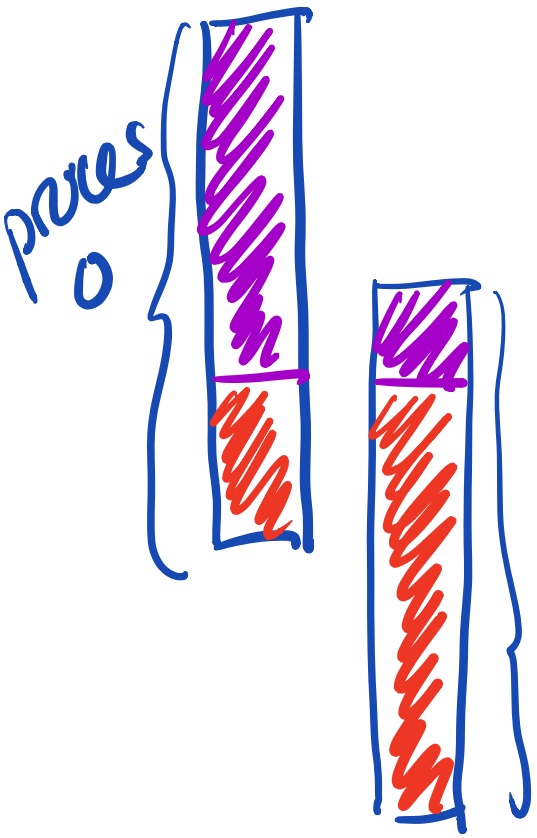
$r_n = \cdots - A u_n$

stencil domain.

Needs "local" vector:

⇒ includes shared dofs to compute stencil updates.





overlapping "local" vectors.

enough informati to compute Aux without further communicatei.

Grid: decompose dots into "owned" (global) partitii.

- determine ghost pts so that overlapped (local)

vectors can be produced.

```
def Au(u_g):                                  # insert ghost parts.
comm | u_l = grid.global_to_local(u_g)
compute | loop over local grid parts:
            r_l[i,j] = A(stencil, u_l)
```

↑ produces correct entries for all owned points.

→ that's all we need for the step.

synchronize → not for FD. used for FEM.

```
    r_g = grid.local_to_global(r_l)
```

For our purposes

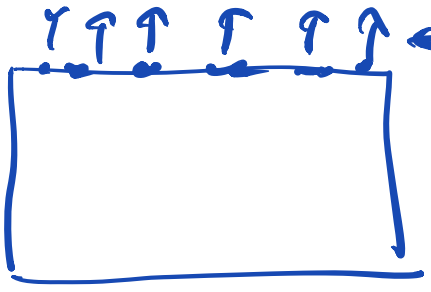
$r_g = r_L$

return r_g.

This gives an algorithm that takes T_C time to communicate & T_W time to compute

$\Rightarrow T_{total} = T_C + T_W$.

Why? work must wait for communication to be finished before it starts.

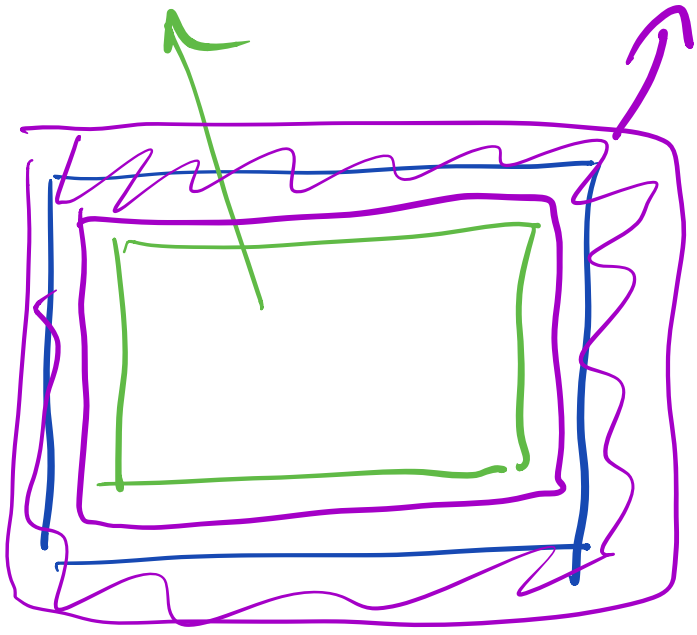


Can arrange for

$T_{total} = \max(T_C, T_W)$

How:

Split local domain into "interior" & "boundary"



interior doesn't need result of communicati.

```
| u_l = grid.g2l_begin(u_g)   ← initiates local ghost pts. (init)
    r_l = compute_interior(u_l)
| u_l = grid.g2l_end(u_g)     ← finishes comms
    r_l += compute_boundary(u_l)
```

→ Asynchronous communicati in background.

⇒ MPI offers this:
getting it to work in practice
is black magic
→ "asynchronous progress"
needs a background thread
inside the library
that is scheduled onto a core
by the OS.
Hamilton doesn't offer this.

Some supercomputing platforms
add a separate chip for
this purpose.

- Design:
  Separate comms & compute.
- Allows work on global
  vectors (portion)

$\Longleftrightarrow$ local vectors (overlapped).

Next time:

- See how to realize this with MPI.
- Scaling analysis after Fischer (2015) for Jacobi iteration:
  → scaling limits for PDE-based simulations.
- Communication latency
  → consequences for multigrid.

→ Read introduction up to end of section II.B.1 "Jacobi iteration". (3½ pages total).