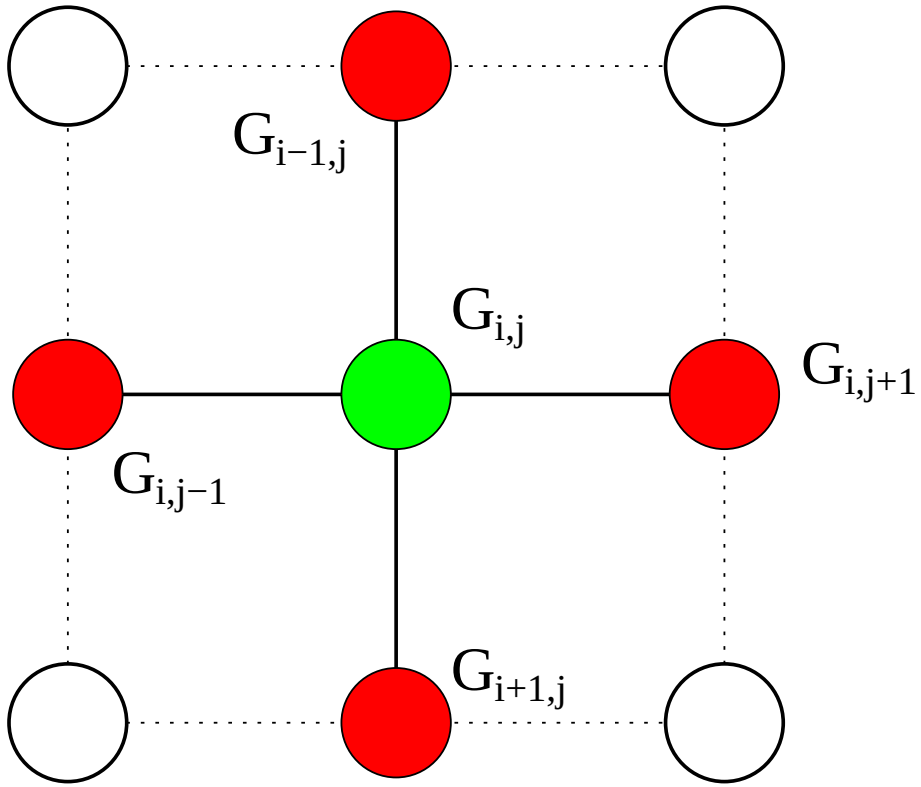
Problem Decompostion

- When writing parallel programmes, data decomposition is probably the most common approach
- Each MPI task is given part of the data set to work on
- After each timestep, updates are sent between processes and often reduction operations are carried out to test for convergeance
- Temporal decomposition, where each MPI task is given a subsection of the algorithm to execute is much rarer
- Used in languages like Scala

Example Problem

- We are going to work on some code that solves the heat equation in two dimensions using a finite difference scheme and the Jacobi method
- This isn't as complex as it sounds!
- We define a grid of $N \times M$ points whose temperature we want to calculate
- The temperature at the boundary of the grid is fixed to some value
- We evolve the temperature in the rest of the grid by taking an average of each of its nearest neighbours

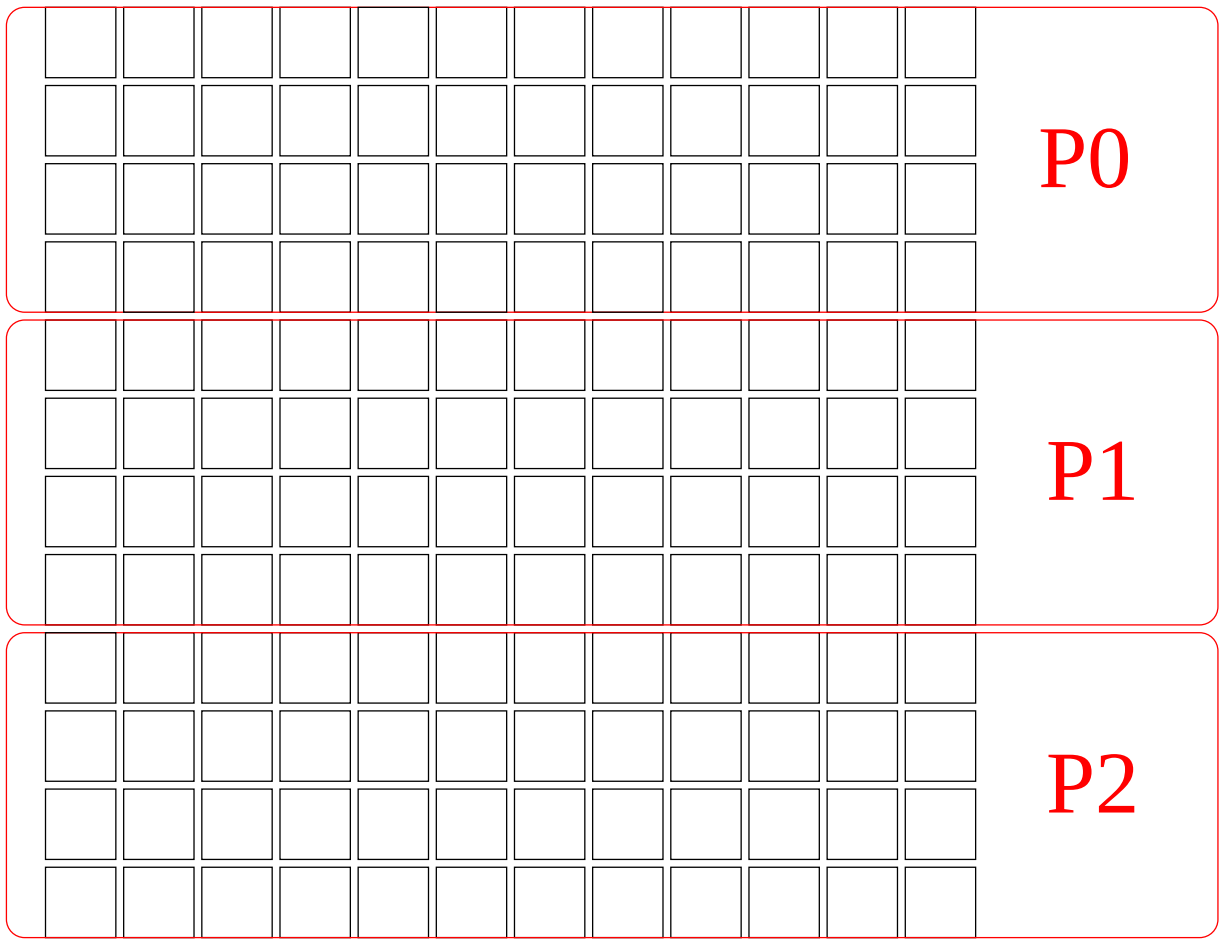
Example Problem



- $G_{i,j}^{t+1} = \frac{1}{4}(G_{i-1,j}^{t} + G_{i+1,j}^{t} + G_{i,j-1}^{t} + G_{i,j+1}^{t})$

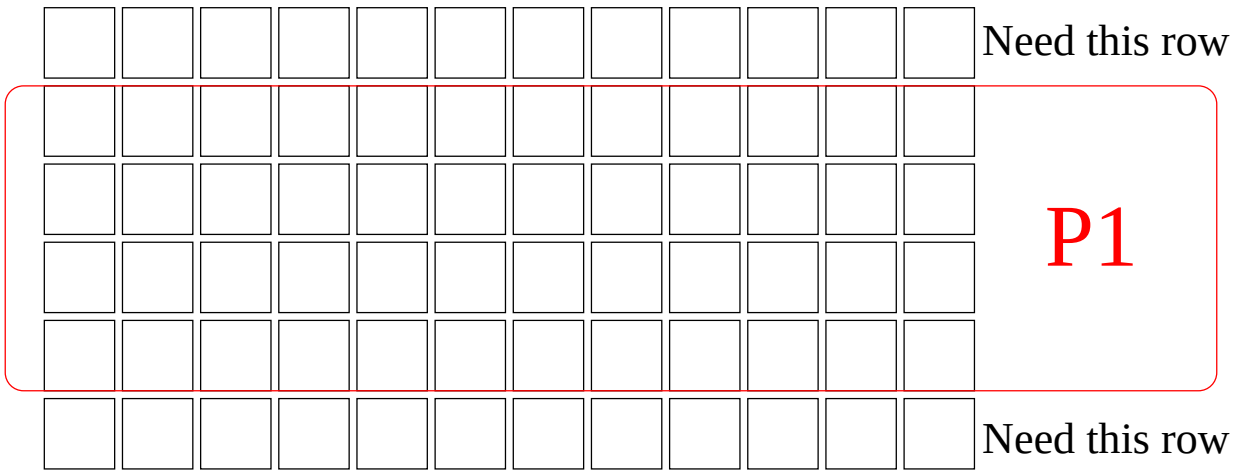
Data decomposition

- We can decompose the data in the heat equation by giving each MPI task a set of rows in the grid to manage
- At each time-step, each task will update the set of grid points that it is in charge of

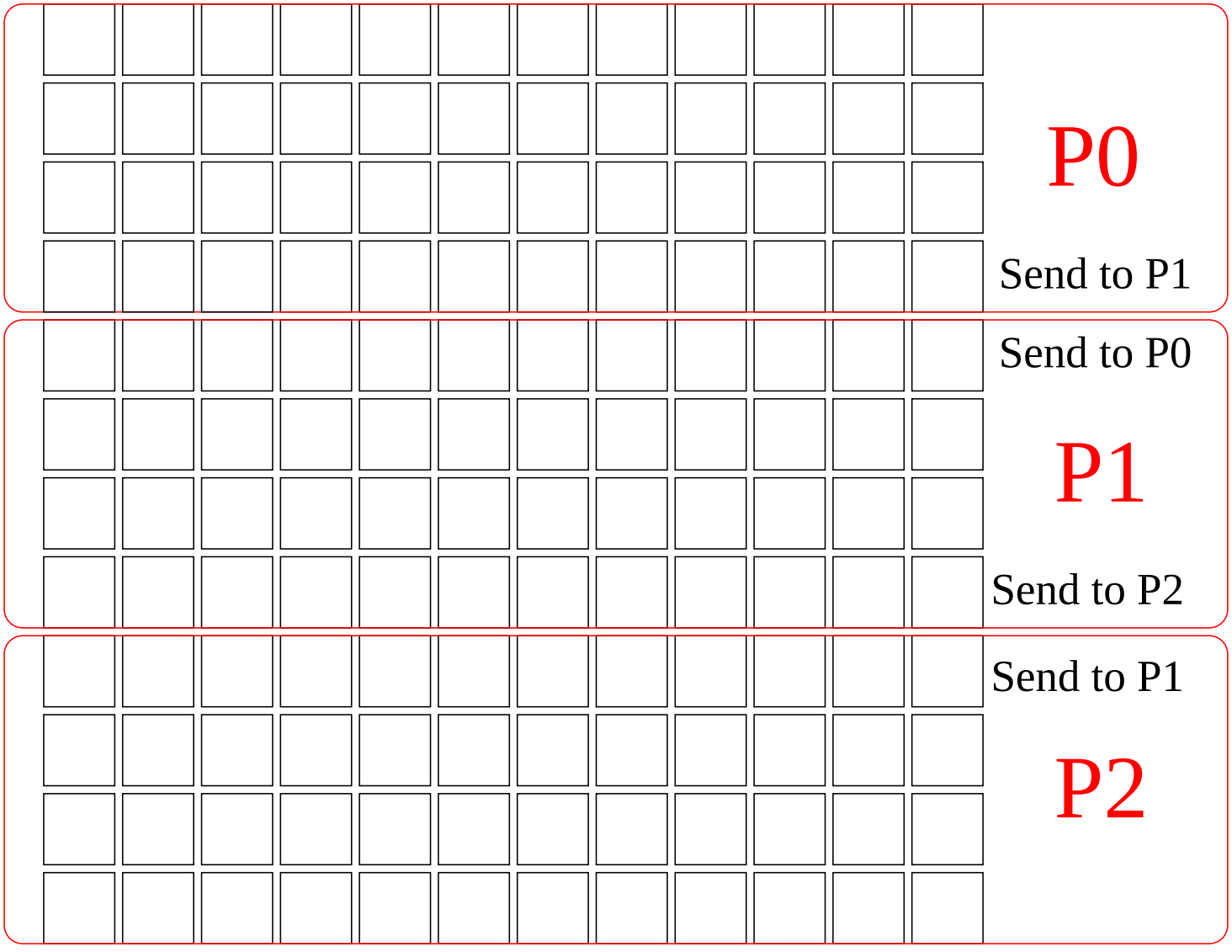


Halo points

- To update the edge rows of each process we need to have access to data under the control of another process
- We take a copy of the relevant row(s) at each timestep
- These points are often called ghost rows or halo points



Halo exchange



Deadlocks in MPI

- We can end up with deadlocks in MPI by not pairing up sends and receives

```
MPI_Send( ... , (rank+1) % size , ... );
MPI_Recv( ... , (size+rank-1) % size , ... );
```

- Everyone is trying to send to the next process (with the last one sending to rank zero
- Would be the case if we used periodic boundary conditions in jacobi problem

Buffering in MPI

- If the data being sent is small enough, and the version of MPI supports it the previous code might work
- MPI provides buffers for holding data while it is being sent
- If the data fits inside a single buffer, the `MPI_Send()` will return before the data is actually delivered
- This then allows each process to post their `MPI_Recv()` to get the new data
- If the data doesn't fit in the buffer then things will deadlock

Deadlocks in MPI

- Without PBC we were able to overcome the deadlocking by using a cacade of Send/Recv pairs

```
if(rank == 0)
    MPI_Send( ..., (rank+1) % size, ... );
else if(rank == size-1)
    MPI_Recv( ..., (size+rank-1) % size, ... );
else {
    MPI_Recv( ..., (size+rank-1) % size, ... );
    MPI_Send( ..., (rank+1) % size, ... );
}
```

- This does not work if we have PBC

Deadlocks in MPI

- If we are using PBC or want to avoid the delays introduced by a cascade we need another communication scheme
- We will use the checkerboard scheme (also called odd/even or black-white)
- We split MPI tasks into two groups based on their rank
- Each task in one group communicates with a task in the other group
- So in one group everyone posts an `MPI_Send()` while in the other group every posts `MPI_Recv()`

Deadlocks in MPI

```
if(rank%2 == 0) {
    MPI_Recv( ..., (size+rank-1) % size, ... );
    MPI_Send( ..., (rank+1) % size, ... );
} else {
    MPI_Send( ..., (rank+1) % size, ... );
    MPI_Recv( ..., (size+rank-1) % size, ... );
}
```

MPI_Sendrecv

- MPI provides the MPI_Sendrecv() function to get around some of the problems of deadlocking calls
- MPI_Sendrecv(send, sendcount, sendtype, dest, sendtag, recv, recvcount, recvtype, source, recvtag, communicator, stat);
- Basically mash the send and receive functions into one call
- You can send data to a different process than you recieve data from

2D Decomposition

- It is possible to carry out a 2D decomposition
- Adds more boundary points that need to be exchanged
- Sending items from different rows is a bit messy
  - Copy values into another array and then send
  - Lots of 1 element sends
  - Create new datatypes and use virtual topologies (soon!)
- Using the single malloc method of allocating 2D arrays makes 2D decomposition easier
- 1D is far simpler as we just send entire rows