Overlapping Calculation and Communication

Group B

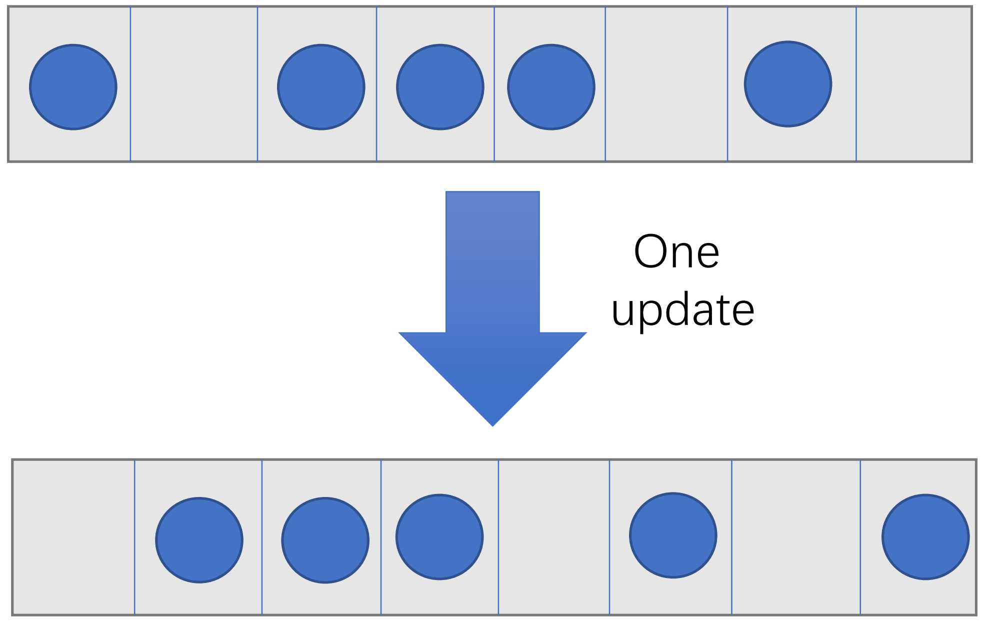
- Fu Yu
- Liu Kevin
- Nita Spyro
- Mackay Fraser

The traffic modelling example

- Propose a way of overlapping the halo swaps with some of the calculations for the new road in terms of the old road.
- Design a communications library that makes possible to switch between overlapped and non-overlapped communications without having to change the main code.
- On what types of parallel machines do you think this overlapping approach might be most beneficial?
- How can you generalise your solution to the Case Study example?

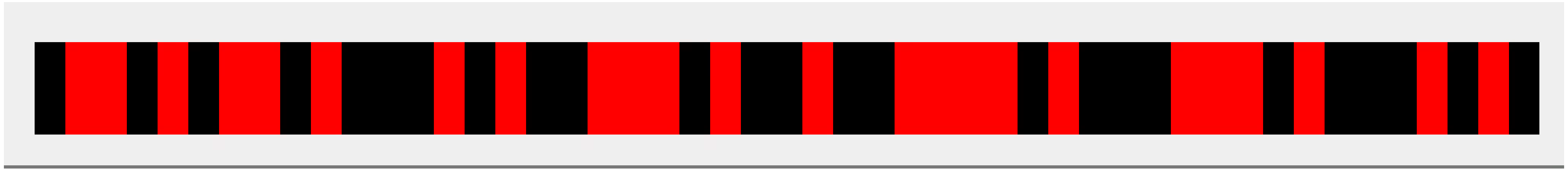
The simple traffic model

- In this simple traffic model, a ‘car’ can move into the next site if it is unoccupied



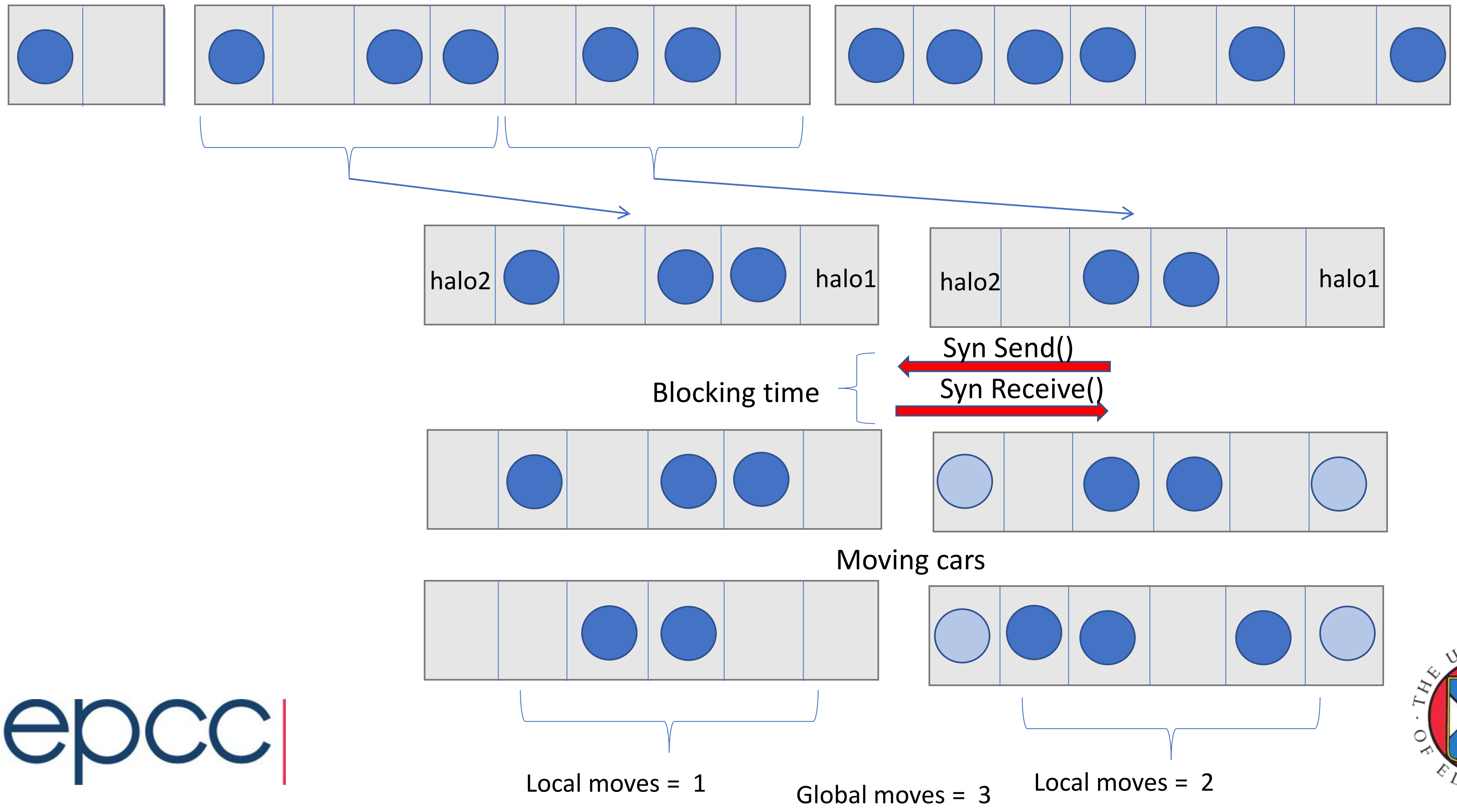
The simple traffic model

- The red occupied sites (cars) move from left to right filling the unoccupied black sites with open boundary conditions

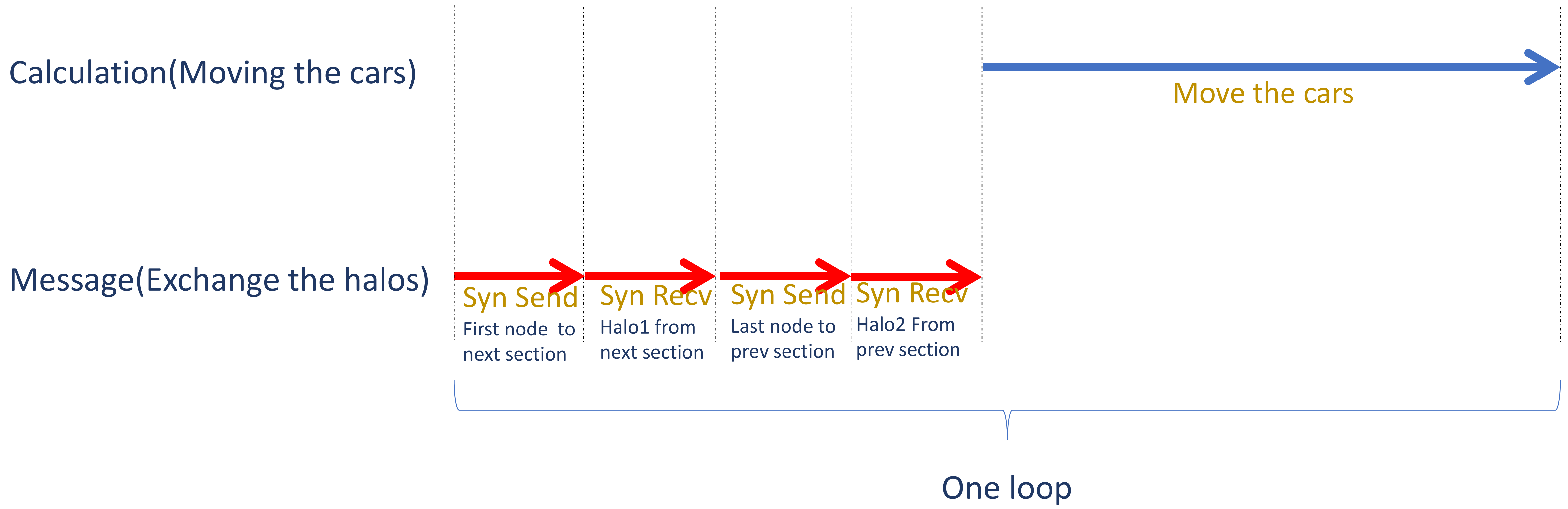


- If the probability of a car moving into an unoccupied site to its right is 1, the system will evolve into a steady state with cars at every second site

Synchronous communication

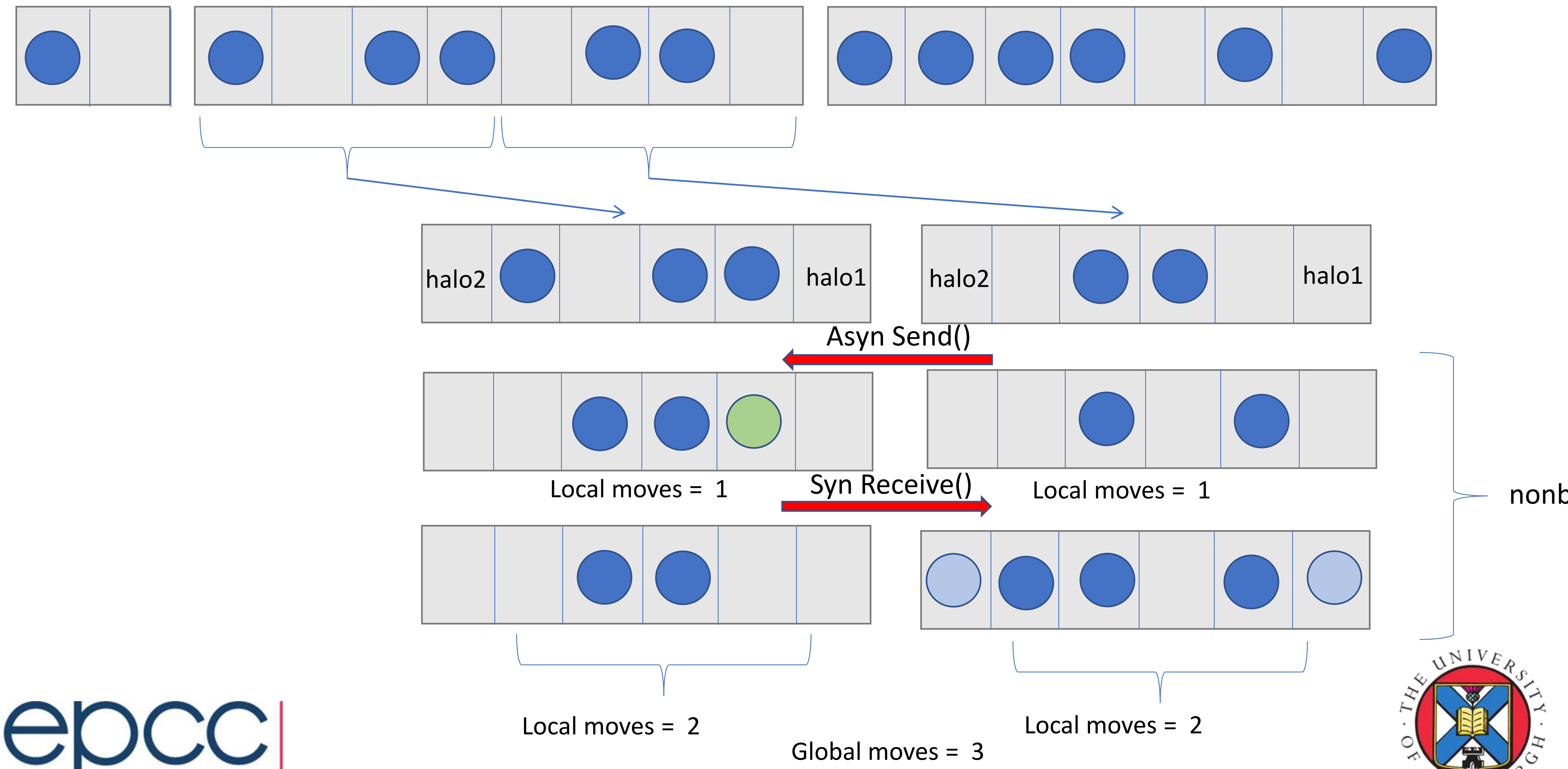


Synchronous communication (non-overlapping)

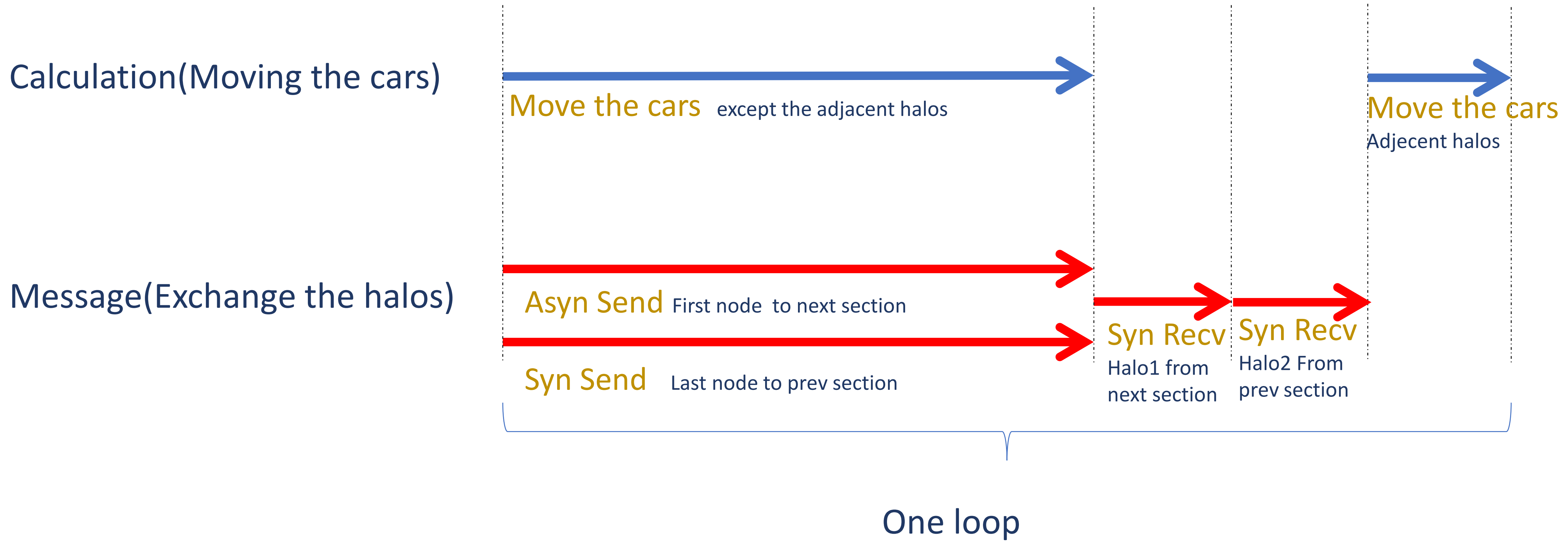

Calculation(Moving the cars)
Move the cars
Message(Exchange the halos)
Syn Send
First node to next section
Syn Recv
Halo1 from next section
Syn Send
Last node to prev section
Syn Recv
Halo2 From prev section
One loop

Global moves = 3

Asynchronous communication

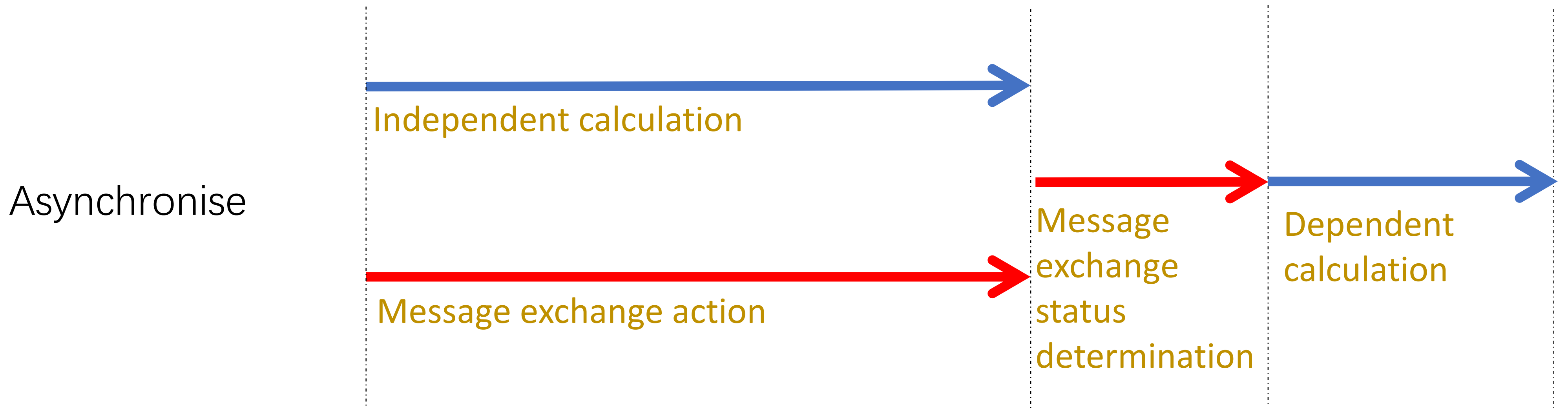


Asynchronous communication (halo overlapping)


Calculation(Moving the cars)
Message(Exchange the halos)
Move the cars
except the adjacent halos
Move the cars
Adjecent halos
Asyn Send
First node to next section
Syn Send
Last node to prev section
Syn Recv
Halo1 from next section
Syn Recv
Halo2 From prev section
One loop

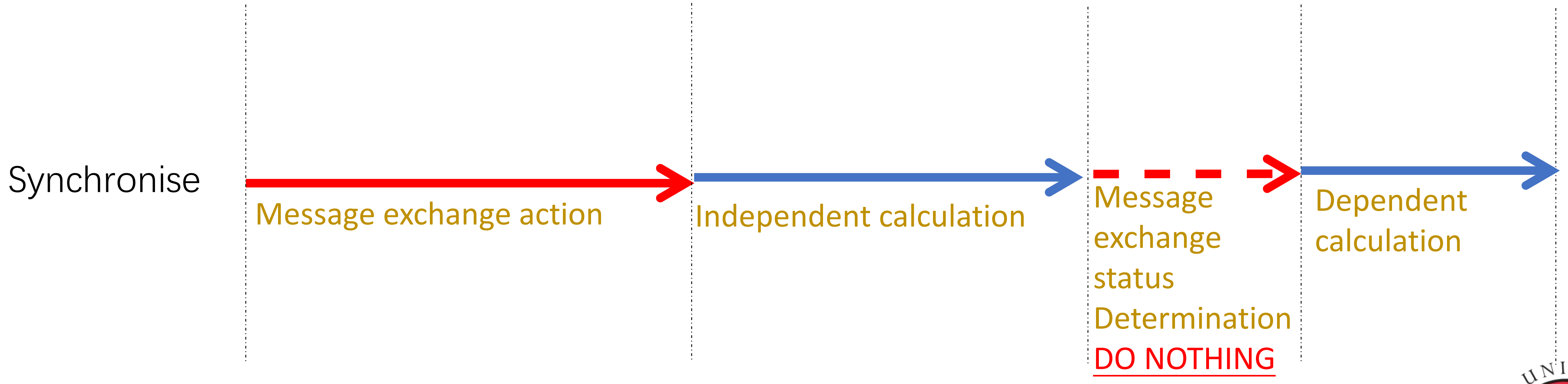
Parallel computing communication general model

- Work breakdown
  - -Independent calculation
  - -Dependent calculation
  - -Message exchange action
  - -Message exchange status determination



Parallel computing communication general model

- Work breakdown
  - -Independent calculation
  - -Dependent calculation
  - -Message exchange action
  - -Message exchange status determination



Library design – Message exchange API

- Library manipulation
  - -MsgHandle* CreateMsghandle(MSG_EXC_TYPE type)
    - -enum MSG_EXC_TYPE{TYPE_SYN, TYPE_ASYN}
  - -int DestroyMsghandle(MsgHandle *handle)

- Message exchange action
  - -int SendMessage(MsgHandle *handle, void* buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm)
  - -int ReveiveMessage(MsgHandle *handle, void* buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm)
    - -Return value: The index of action
- Message exchange status determination
  - -int CheckStatus(MsgHandle *handle, int index)
    - -Return value: the status of message exchange

Library design – Sodu code

- SendMessage

```
-If handle.MSG_EXC_TYPE equals TYPE_SYN
    Use MPI_Ssend to send message
    Change the status saved in handle to SUCCESSFUL
-If handle.MSG_EXC_TYPE equals TYPE_ASYN
    Use MPI_Issend to send message
    Change the status saved in handle to PROCESSING
    Save the MPI_Request structure and other info to handle
```

- CheckStatus

```
-If handle.MSG_EXC_TYPE equals TYPE_SYN
    Return SUCCESSFUL
-If handle.MSG_EXC_TYPE equals TYPE_ASYN
    Use the MPI_Request structure saved in handle to MPI_Wait
    Change the status saved in handle to SUCCESSFUL
    Return status
```

Parallel Machines

- Share memory

  -Multiple processors operate independently but access global memory space.

- Distributed memory

  - Processors have local memory.
  - Data from one processor must be communicated to another if required.

Parallel Machines

- Share memory

  -Multiple processors operate independently but access global memory space.

- Distributed memory

  - Processors have local memory.
  - Data from one processor must be communicated to another if required.

Case study example / Parallel Image Processing

- For the initial parallelisation used trivial parallelism

  - Each process worked on different sections of the image with no communication between them.

  -Had exactly the same data decomposition and parallel IO approaches as a fully working parallel code.

Case study example / Parallel Image Processing

- For the initial parallelisation used trivial parallelism

  - Each process worked on different sections of the image with no communication between them.

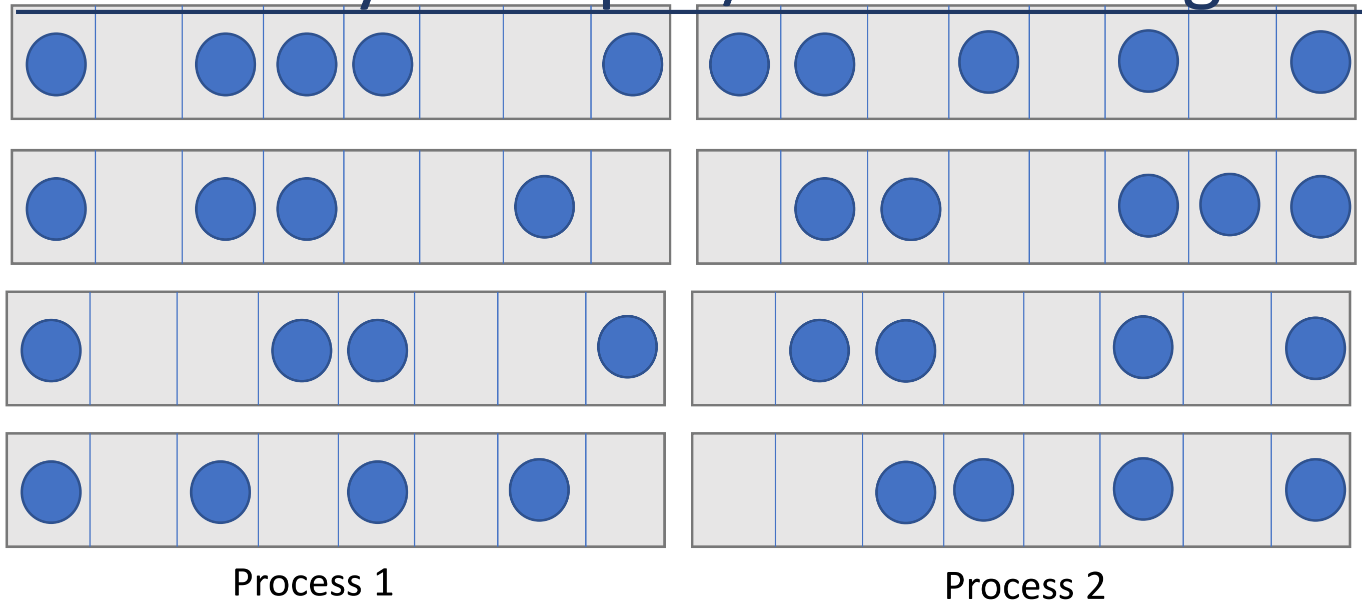
  -Had exactly the same data decomposition and parallel IO approaches as a fully working parallel code.

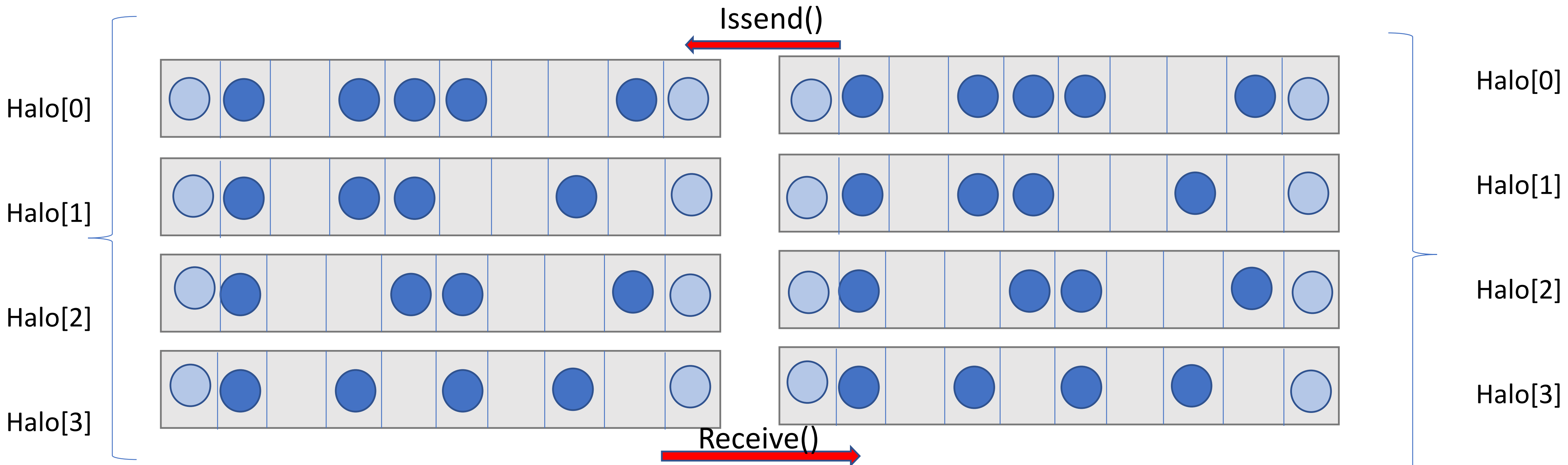
- Overlapping communication and calculation

  -Doing calculation that does not require the communicated halo data at the same time as the halo is being sent using non-blocking routines.

  -Calculations involving the halos are done separately after the halos have arrived.

Case study example / Parallel Image Processing


Process 1
Process 2


Issend()
Receive()
Halo[0]
Halo[1]
Halo[2]
Halo[3]
Halo[0]
Halo[1]
Halo[2]
Halo[3]