Message-Passing

Parallel Programming using Processes

Outline

- Message-Passing Parallelism
  - processes
  - messages
  - communications patterns

- Practicalities
  - usage on real HPC architectures

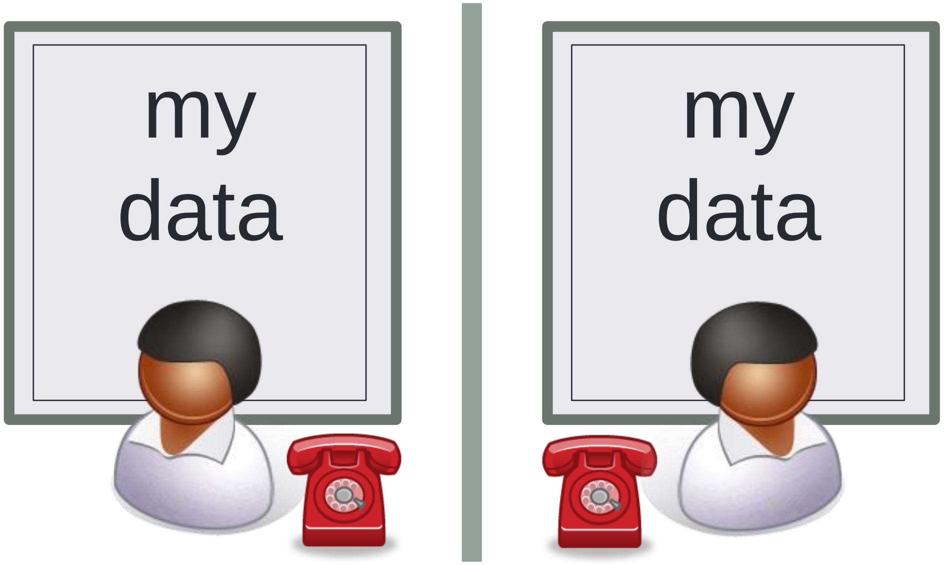
Generic Parallel Machine

- Good conceptual model is collection of laptops
- Connected together by a network

- Each laptop is called a *compute node*
  - each has its own operating system and network connection
- Suppose each node is a quadcore laptop
  - total system has 20 processor-cores

Analogy

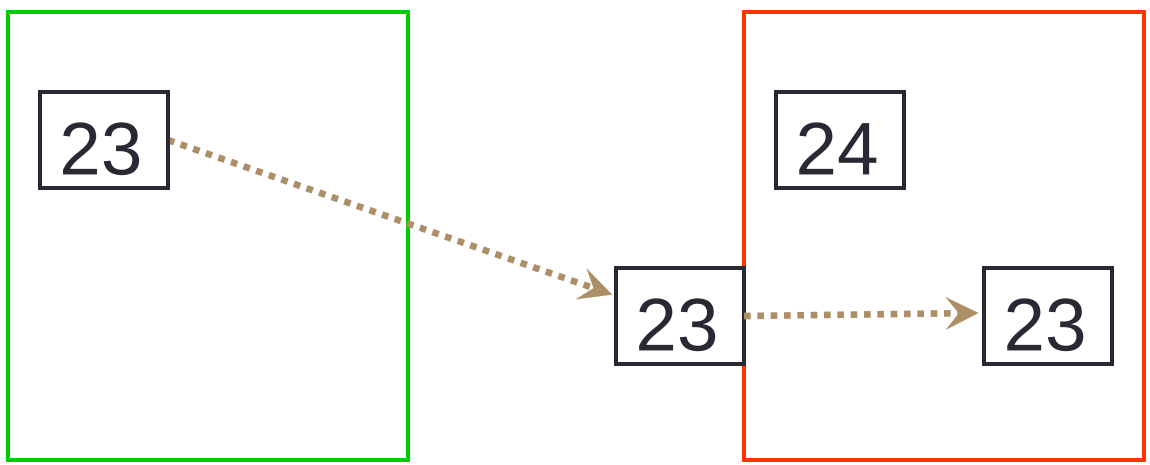
- Two whiteboards in different single-person offices
  - the distributed memory
- Two people working on the same problem
  - the processes on different nodes attached to the interconnect
- How do they collaborate?
  - to work on single problem
- Explicit communication
  - e.g. by telephone
  - no shared data



Process communication

	Process 1	Process 2
Program	`a=23` `Send(2,a)`	`Recv(1,b)` `a=b+1`

Data



Synchronisation

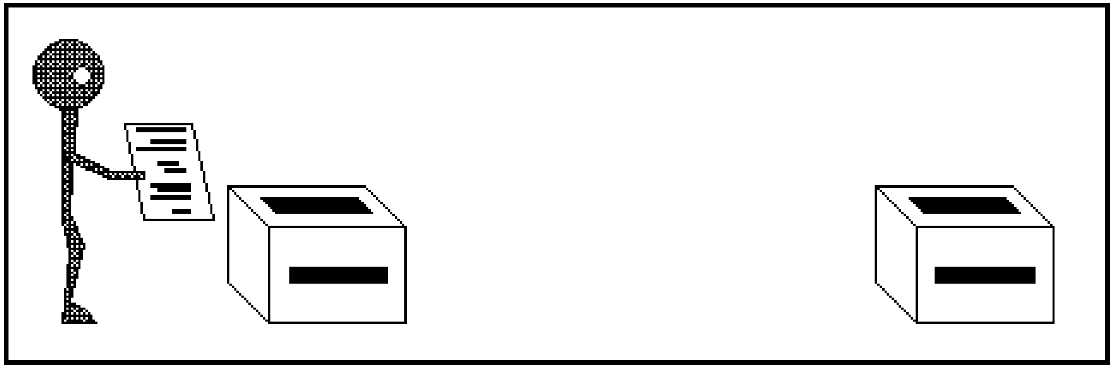
- Synchronisation is automatic in message-passing
  - the messages do it for you
- Make a phone call …
  - … wait until the receiver picks up
- Receive a phone call
  - … wait until the phone rings
- No danger of corrupting someone else's data
  - no shared blackboard

Communication modes

- Sending a message can either be synchronous or asynchronous
- A synchronous send is not completed until the message has started to be received
- An asynchronous send completes as soon as the message has gone
- Receives are usually synchronous - the receiving process must wait until the message arrives

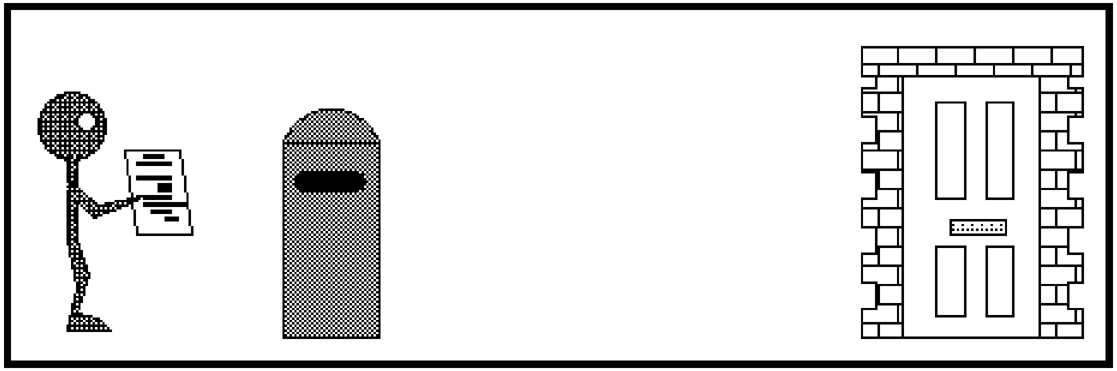
Synchronous send

- Analogy with faxing a letter.
- Know when letter has started to be received.



Asynchronous send

- Analogy with posting a letter.
- Only know when letter has been posted, not when it has been received.



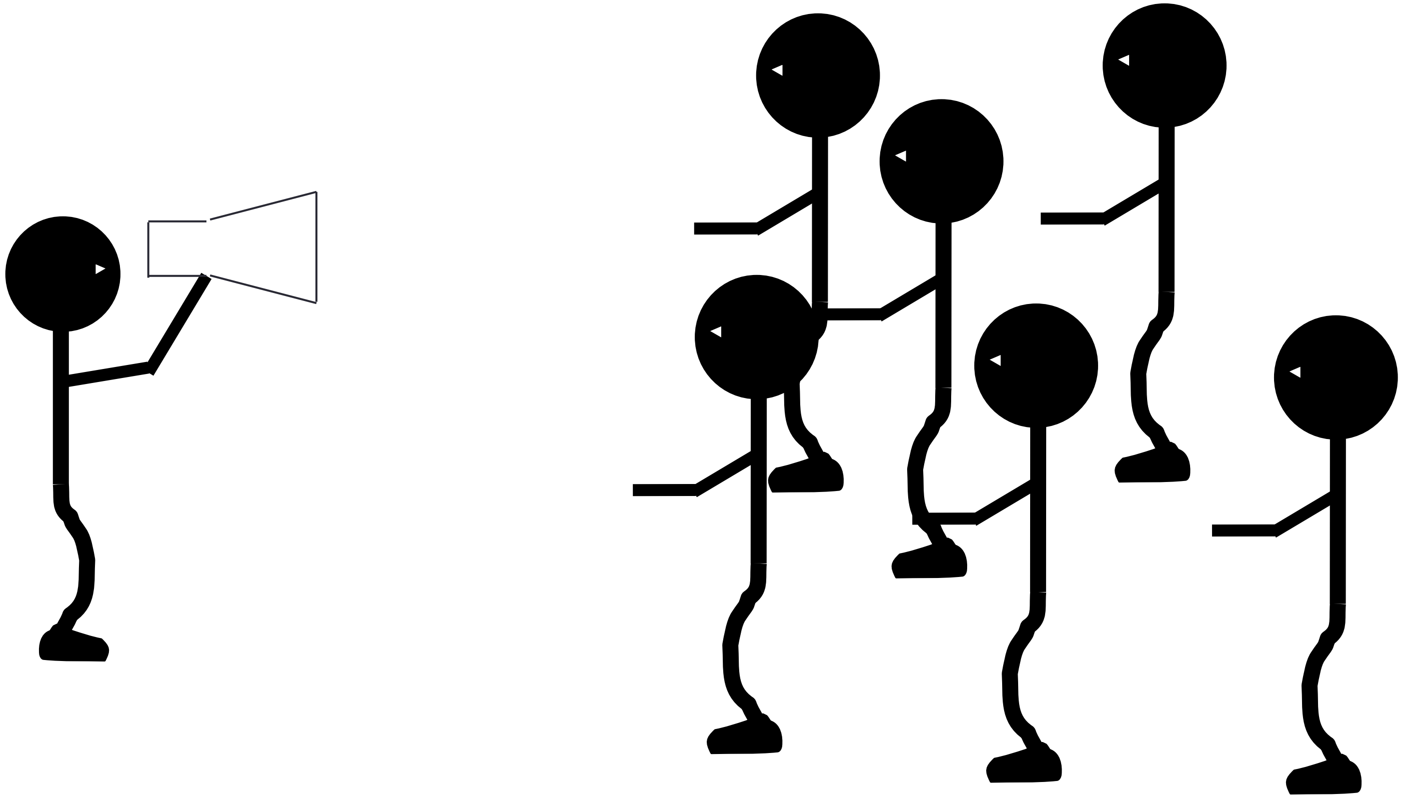
Point-to-Point Communications

- We have considered two processes
  - one sender
  - one receiver
- This is called point-to-point communication
  - simplest form of message passing
  - relies on matching send and receive
- Close analogy to sending personal emails

Collective Communications

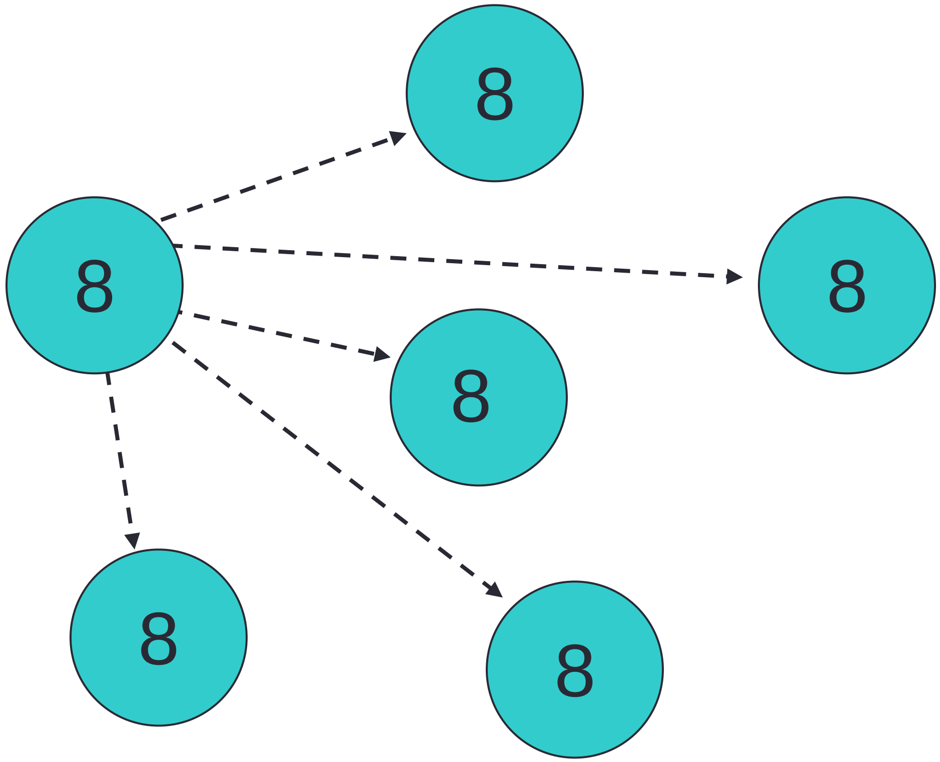
- A simple message communicates between two processes
- There are many instances where communication between groups of processes is required
- Can be built from simple messages, but often implemented separately, for efficiency

Broadcast: one to all communication



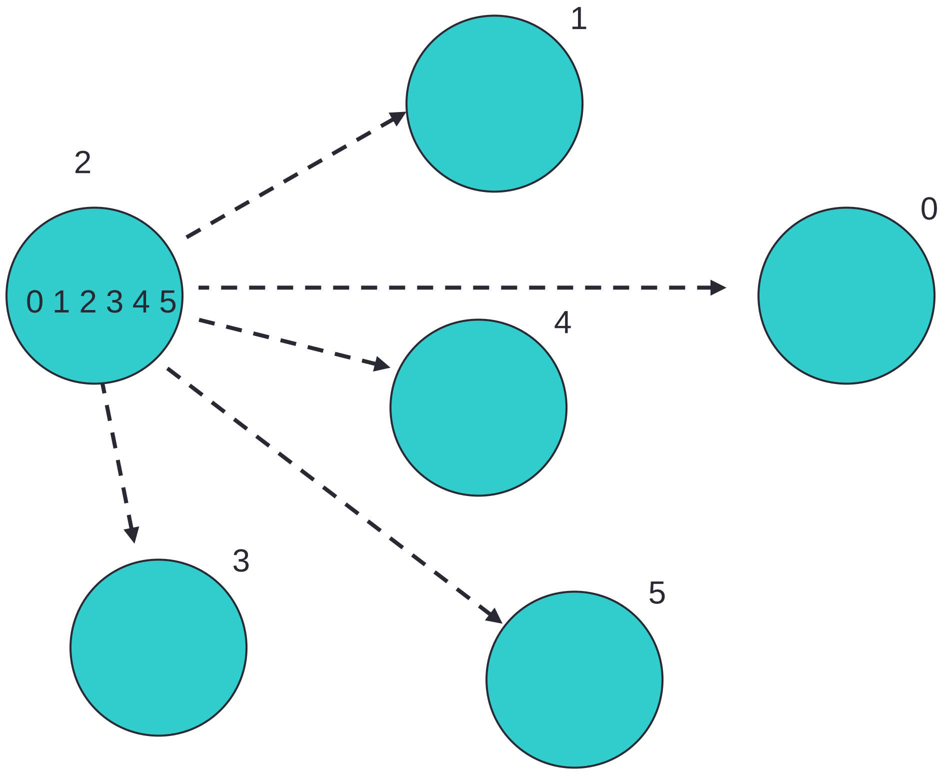
Broadcast

- From one process to all others



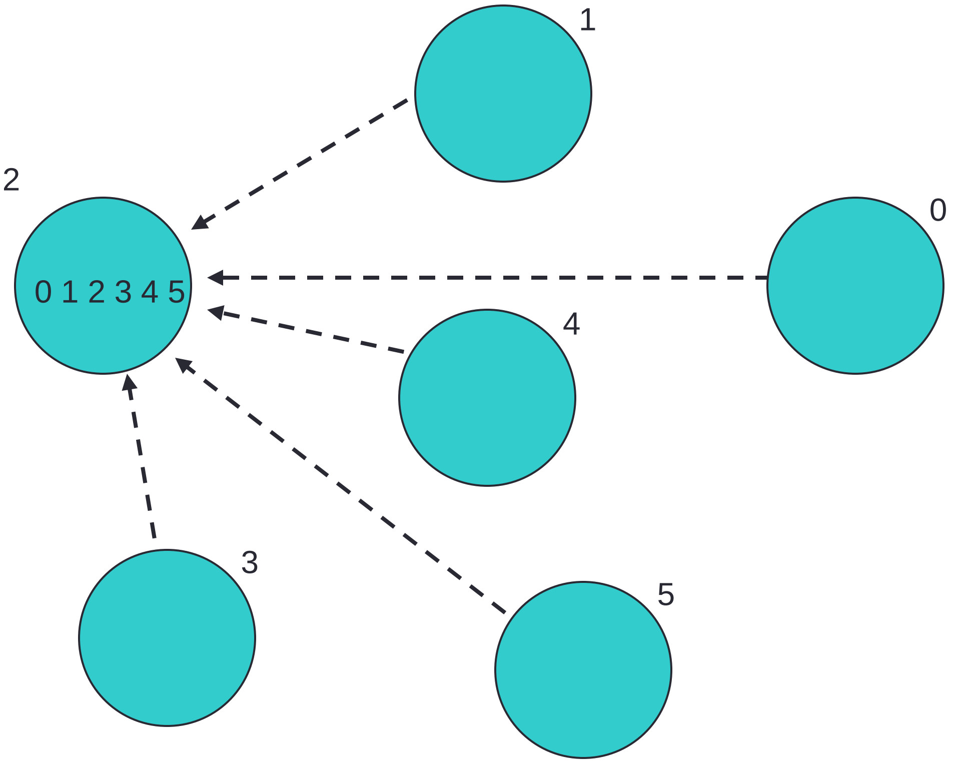
Scatter

- Information scattered to many processes



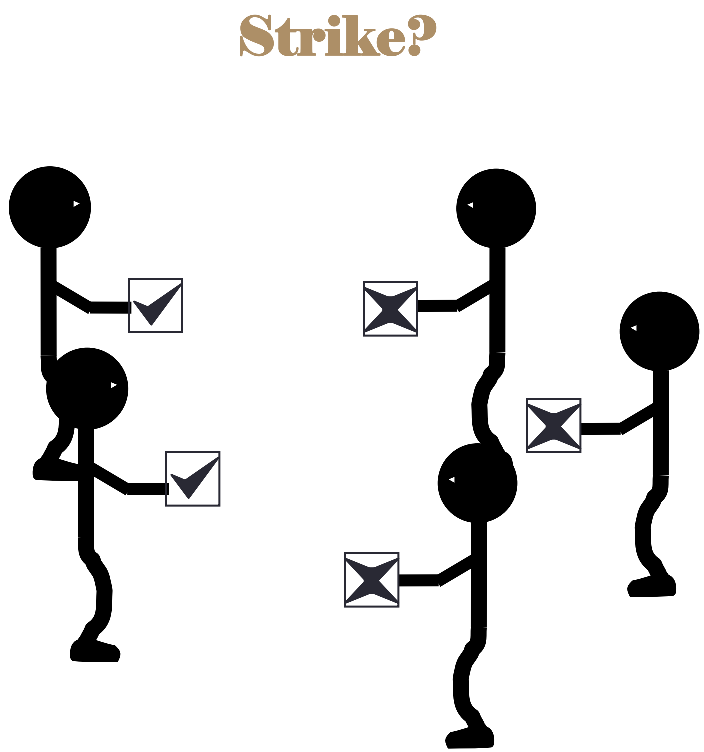
Gather

- Information gathered onto one process



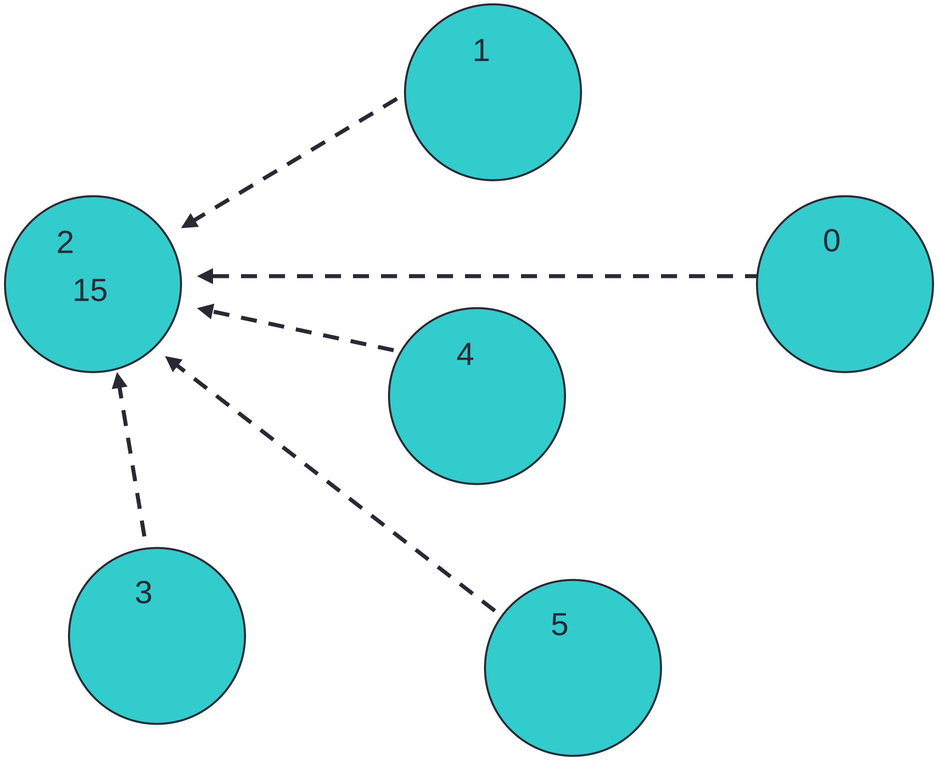
Reduction Operations

- Combine data from several processes to form a single result

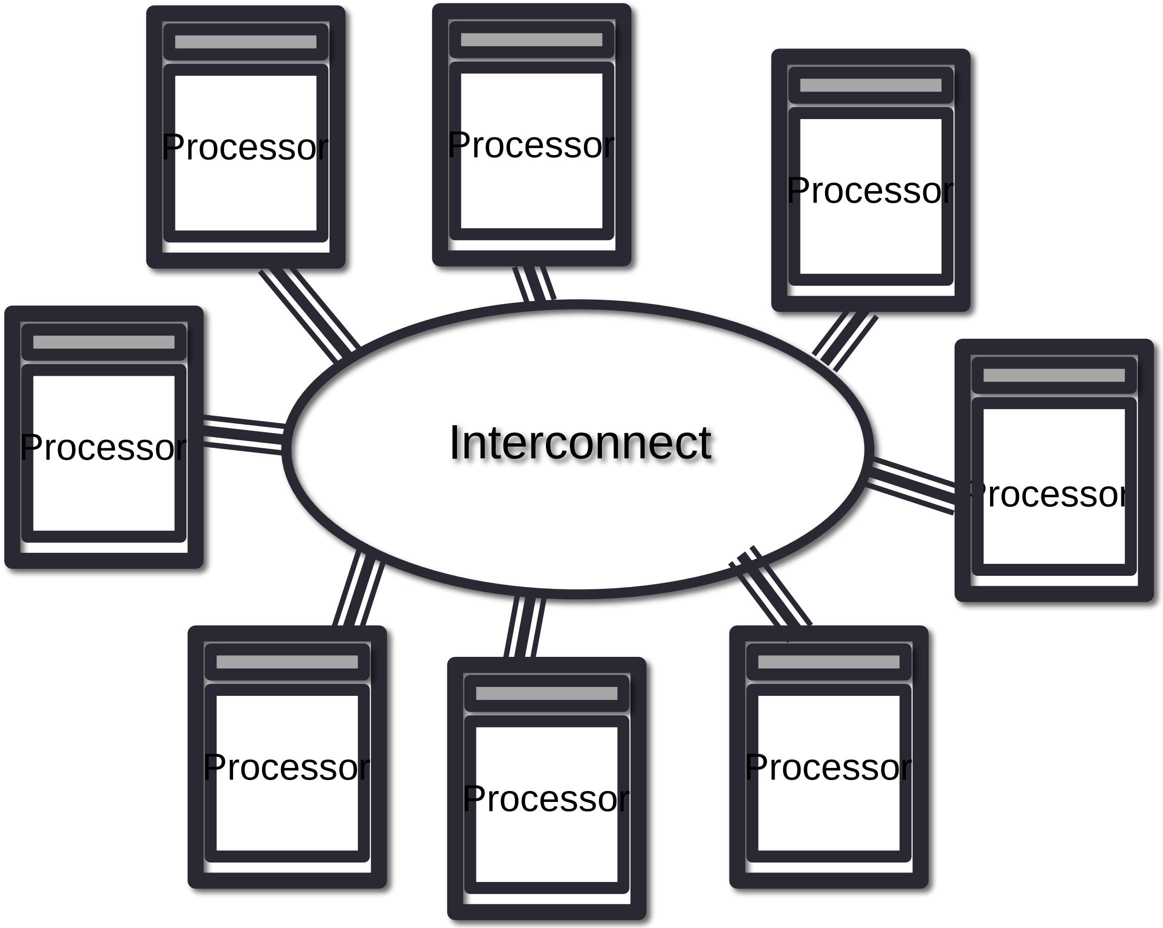


Reduction

- Form a global sum, product, max, min, etc.

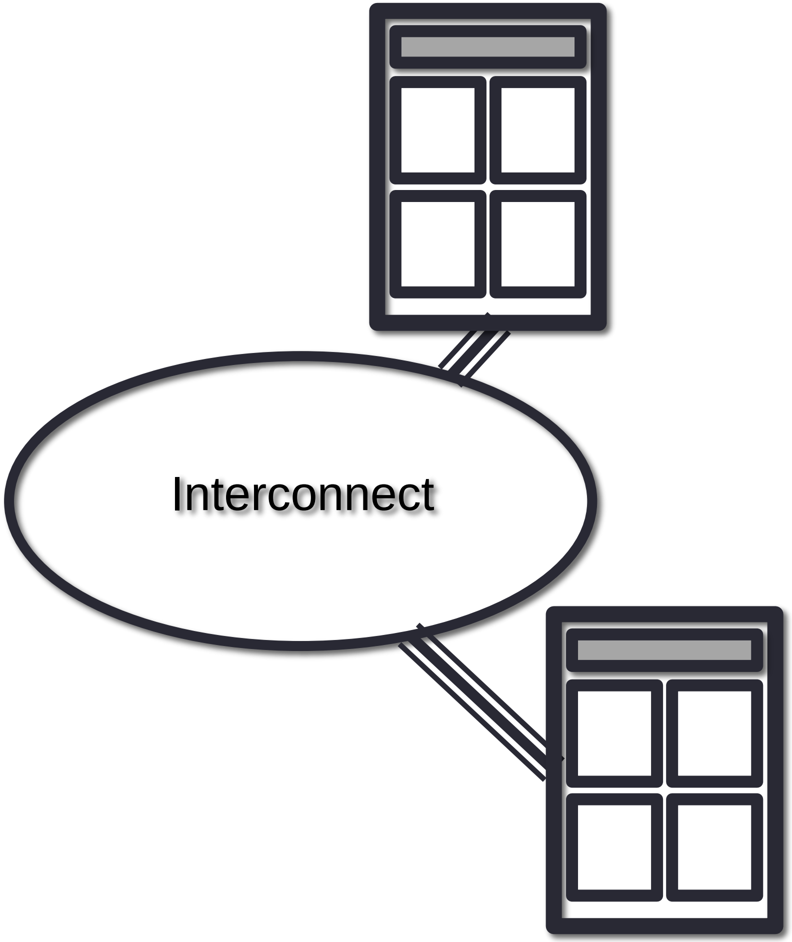


Hardware



- Natural map to distributed-memory
  - one process per processor-core
  - messages go over the interconnect, between nodes/OS's

Practicalities

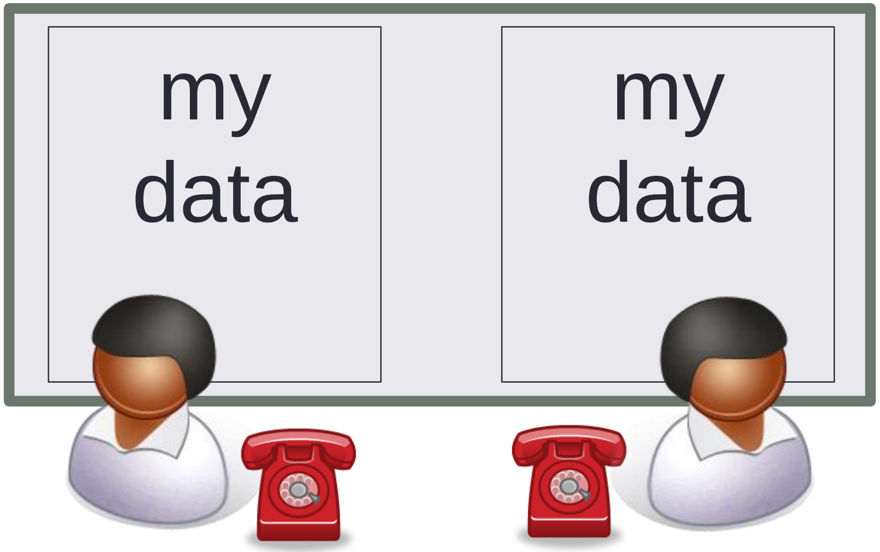


- 8-core machine might only have 2 nodes
  - how do we run MPI on a real HPC machine?
- Mostly ignore architecture
  - pretend we have single-core nodes
  - one MPI process per processor-core
  - e.g. run 8 processes on the 2 nodes
- Messages between processes on the same node are fast
  - but remember they also share access to the network

Message Passing on Shared Memory

- Run one process per core
  - don't directly exploit shared memory
  - analogy is phoning your office mate
  - actually works well in practice!

- Message-passing programs run by a special job launcher
  - user specifies #copies
  - some control over allocation to nodes



Issues

- Sends and receives must match
  - danger of deadlock
  - program will stall (forever!)
- Possible to write very complicated programs, but …
  - most scientific codes have a simple structure
  - often results in simple communications patterns
- Use collective communications where possible
  - may be implemented in efficient ways

Summary (i)

- Messages are the *only* form of communication
  - all communication is therefore explicit
- Most systems use the SPMD model
  - Single Program Multiple Data
  - all processes run exactly the same code
  - each has a unique ID
  - processes can take different branches in the same codes
- Basic communications form is point-to-point
  - collective communications implement more complicated patterns that often occur in many codes

Summary (ii)

- Message-Passing is a programming model
  - that is implemented by MPI
  - the Message-Passing Interface is a library of function/subroutine calls
- Essential to understand the basic concepts
  - private variables
  - explicit communications
  - SPMD
- Major difficulty is understanding the Message-Passing model
  - a very different model to sequential programming

```
if (x < 0)
   print("Error");
   exit;
```

Exercise: computing pi

An approximation to the value π can be obtained from the following expression

$$\frac{\pi}{4} = \int_0^1 \frac{dx}{1+x^2} \approx \frac{1}{N}\sum_{i=1}^{N} \frac{1}{1+\left(\frac{i-\frac{1}{2}}{N}\right)^2}$$

where the answer becomes more accurate with increasing N. Iterations over i are independent so the calculation can be parallelised.

- Will use this as a simple example for MPI and OpenMP
- Traffic Model (see later) is a much better analogue of a real simulation code
  - but pi calculation illustrates basic concepts