Parallel Models

Different ways to exploit parallelism

Partners

Funding

Horizon 2020
European Union Funding
for Research & Innovation

Reusing this material

Outline

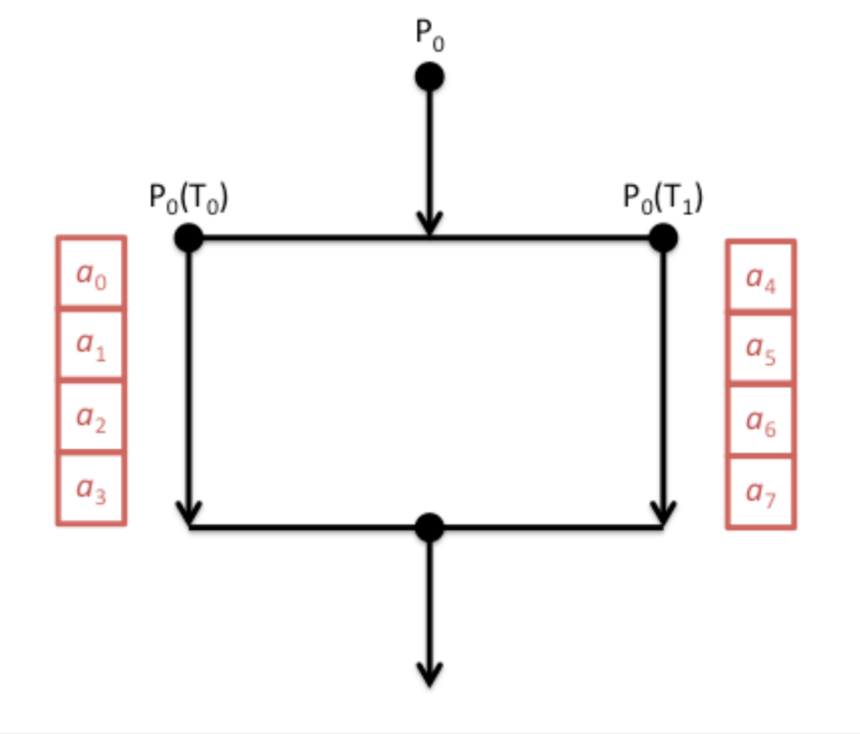
- Shared-Variables Parallelism
  - threads
  - shared-memory architectures
- Message-Passing Parallelism
  - processes
  - distributed-memory architectures
- Practicalities
  - compilers
  - libraries
  - usage on real HPC architectures

Shared Variables

Threads-based parallelism

Shared-memory concepts

- Have already covered basic concepts
  - threads can all see data of parent process
  - can run on different cores
  - potential for parallel speedup

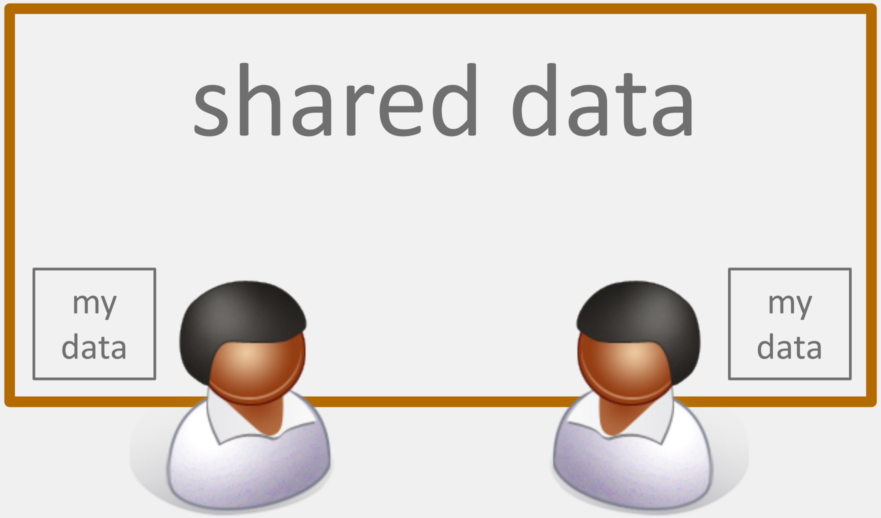


Analogy

- One very large whiteboard in a two-person office
  - the shared memory
- Two people working on the same problem
  - the threads running on different cores attached to the memory

- How do they collaborate?
  - working together
  - but not interfering

- Also need *private* data



Thread Communication

Thread 1

Thread 2

Program

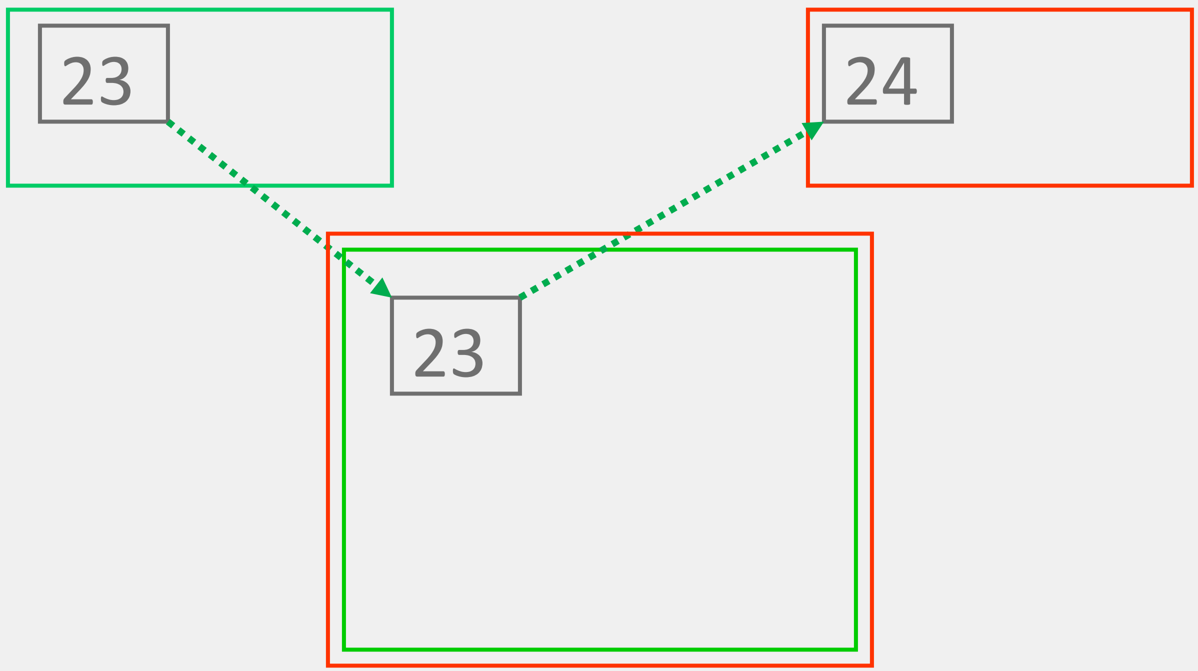
`mya=23`

`a=mya`

`mya=a+1`

Private data

Shared data

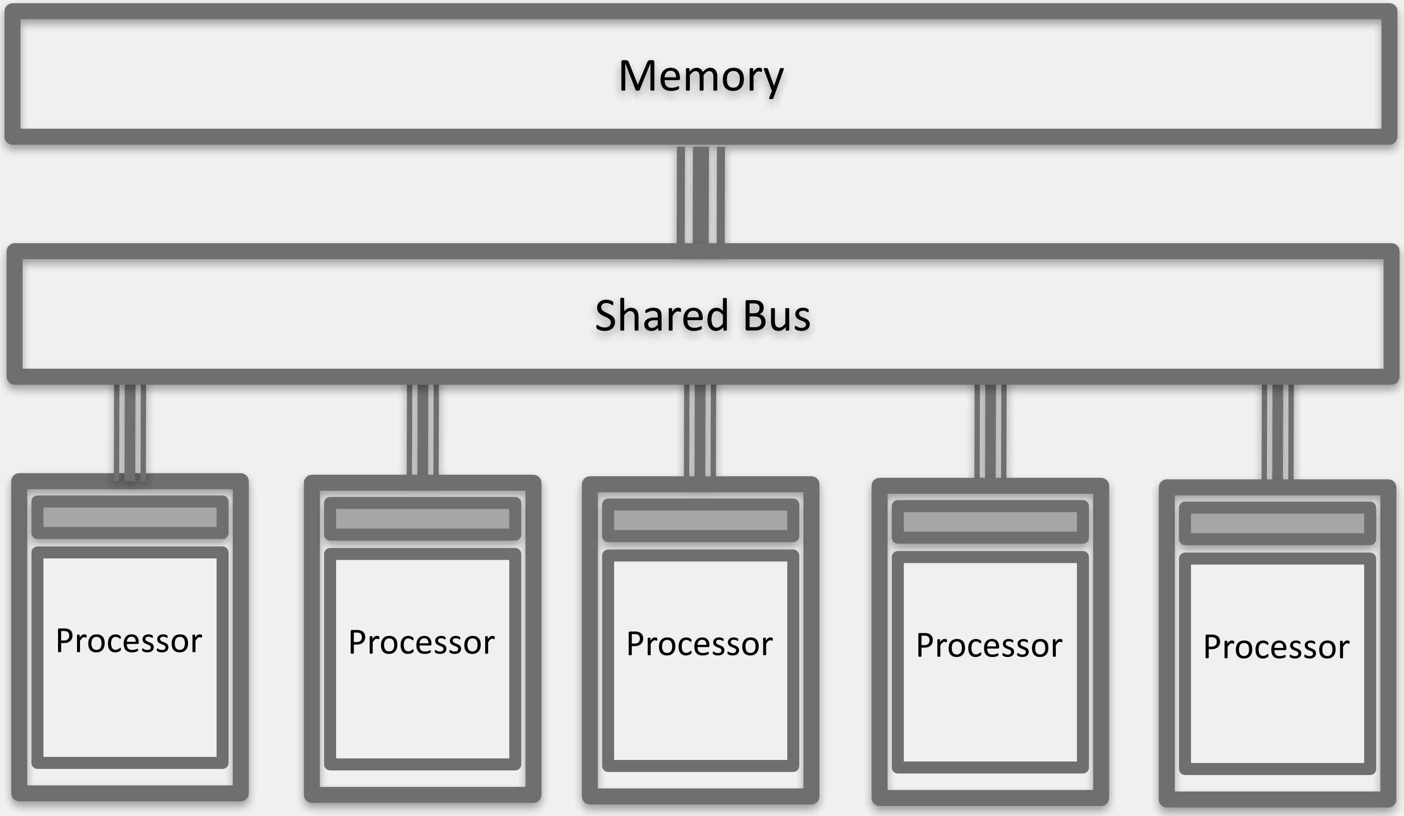


Synchronisation

- Synchronisation crucial for shared variables approach
  - thread 2's code must execute *after* thread 1
- Most commonly use global barrier synchronisation
  - other mechanisms such as locks also available
- Writing parallel codes relatively straightforward
  - access shared data as and when its needed
- Getting correct code can be difficult!

Hardware

Need a shared-memory architecture to use threads-based parallelism:



Threads: Summary

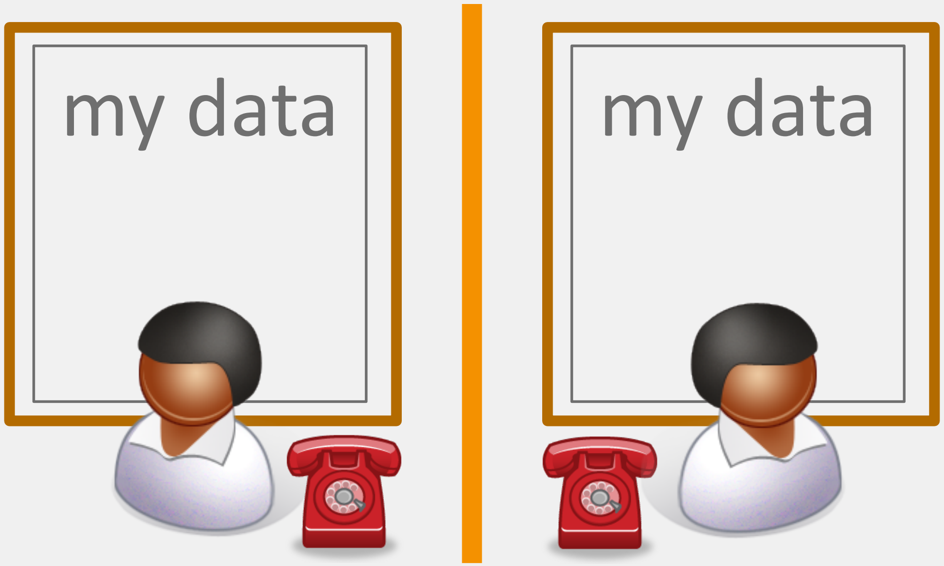
- Shared blackboard is a good analogy for thread parallelism
- Thread-base parallelism requires a shared-memory architecture
  - in HPC terms, cannot scale beyond a single node
- Threads operate independently on the shared data
  - need to ensure they don't interfere; synchronisation is crucial
- Threading in HPC usually uses **OpenMP** threads
  - OpenMP standard allows simple statements to be added to code
    - these control creation of threads, allocation of work
  - Supports common parallel decomposition patterns, e.g. loop parallelism
  - Provides flexible robust ways of managing threads' behaviour at runtime
    - this can make a big difference to performance

Message Passing

Process-based parallelism

Analogy

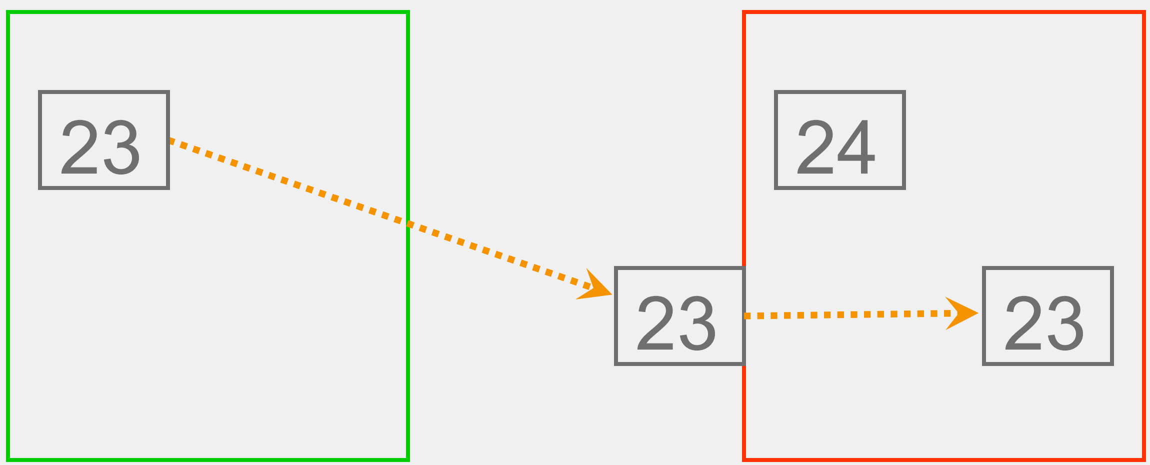
- Two whiteboards in different single-person offices
  - the distributed memory
- Two people working on the same problem
  - the processes on different nodes attached to the interconnect
- How do they collaborate?
  - to work on single problem
- Explicit communication
  - e.g. by telephone
  - no shared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

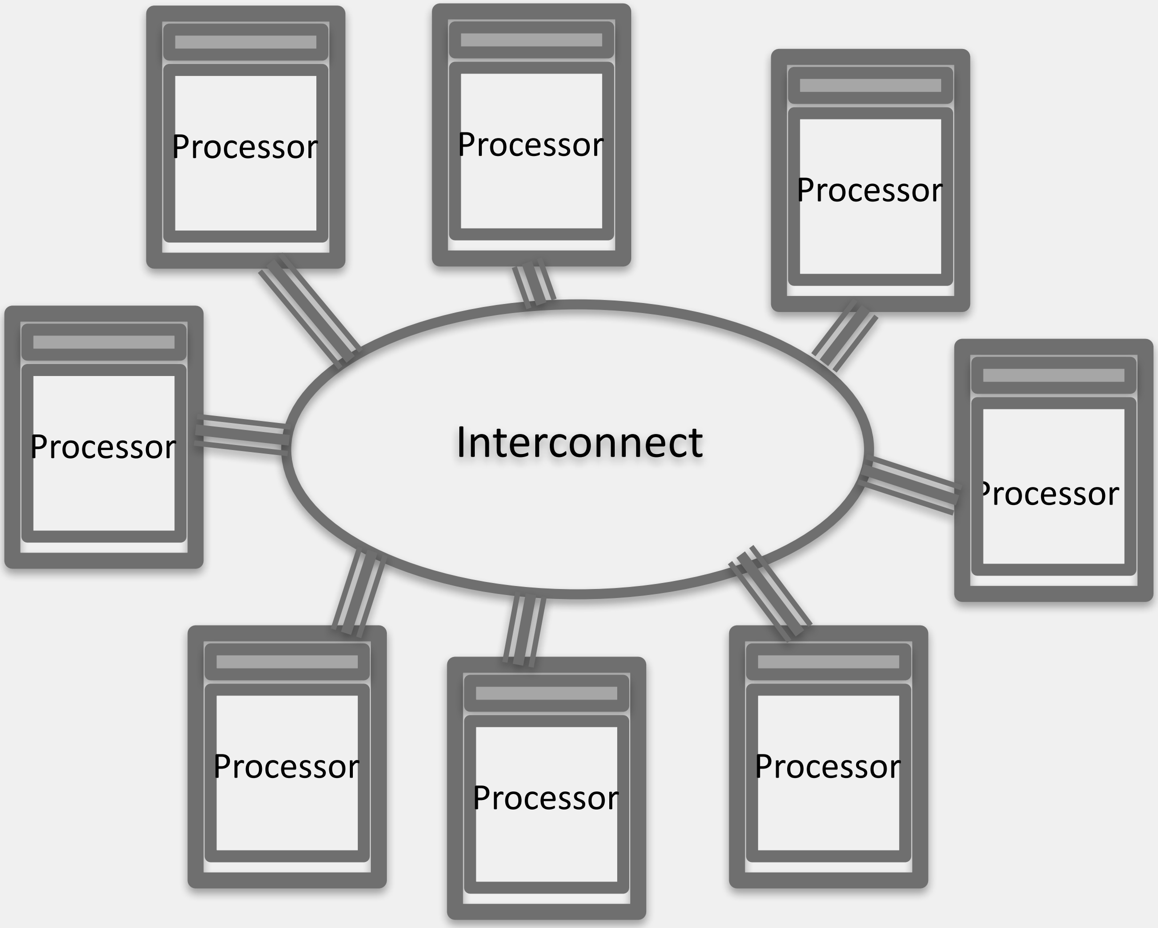
- Make a phone call …
  - … wait until the receiver picks up
- Receive a phone call
  - … wait until the phone rings

- No danger of corrupting someone else's data
  - no shared blackboard

Hardware

Natural map to distributed-memory:

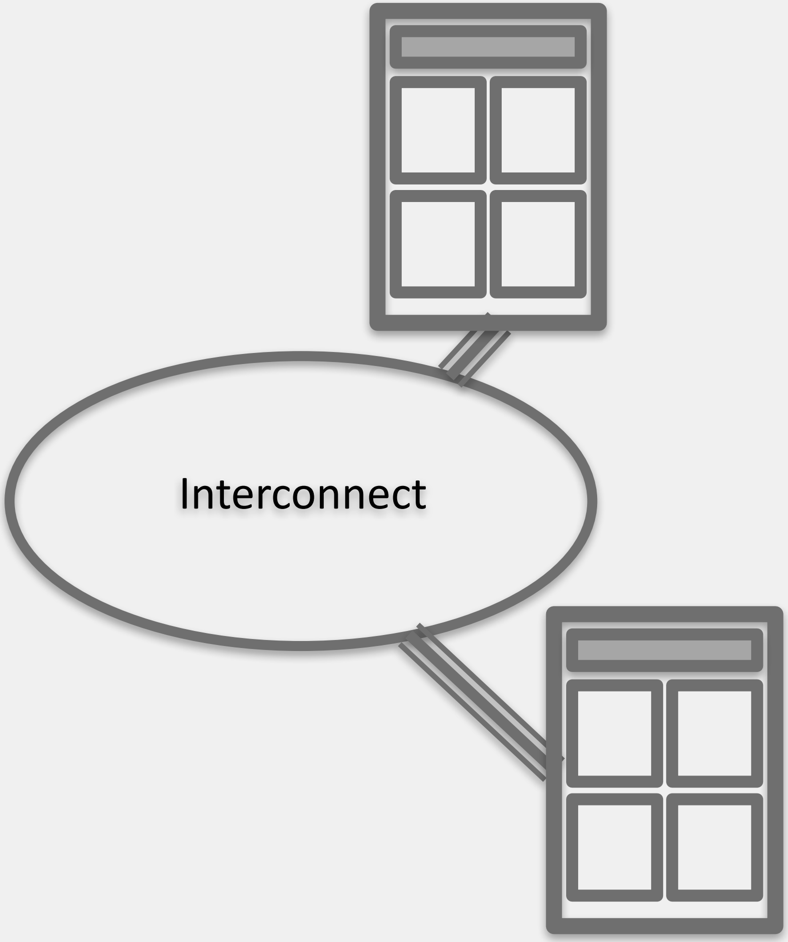
- one process per processor-core
- messages go over the interconnect, between nodes/OS's



Processes: Summary

- Processes cannot share memory
  - ring-fenced from each other
  - analogous to white boards in separate offices
- Communication requires explicit *messages*
  - analogous to making a phone call, sending an email, …
  - synchronisation is done by the messages
- Almost exclusively use Message-Passing Interface (**MPI**)
  - MPI is a library of function calls / subroutines
  - Allows control over how information is shared between processes and independent distributed memory spaces through sending of messages
  - Supported by and heavily optimised for HPC networks

Practicalities

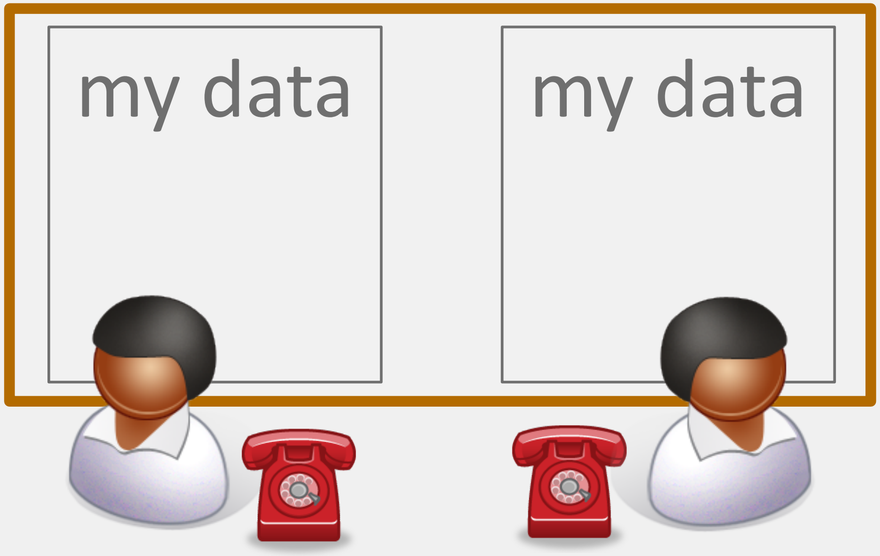


- 8-core machine might only have 2 nodes
  - how do we run MPI on a real HPC machine?
- Mostly ignore architecture
  - pretend we have single-core nodes
  - one MPI process per processor-core
  - e.g. run 8 processes on the 2 nodes
- Messages between processes on the same node are fast
  - but remember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



Summary

- Shared-variables parallelism
  - uses threads
  - requires shared-memory machine
  - easy to implement but limited scalability
  - in HPC, done using OpenMP

- Distributed memory
  - uses processes
  - can run on any machine: messages can go over the interconnect
  - harder to implement but better scalability
  - on HPC, done using MPI