

# Parallel Models

---

Different ways to exploit parallelism

# Reusing this material



This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License.

[http://creativecommons.org/licenses/by-nc-sa/4.0/deed.en\\_US](http://creativecommons.org/licenses/by-nc-sa/4.0/deed.en_US)

This means you are free to copy and redistribute the material and adapt and build on the material under the following terms: You must give appropriate credit, provide a link to the license and indicate if changes were made. If you adapt or build on the material you must distribute your work under the same license as the original.

Acknowledge EPCC as follows: “© EPCC, The University of Edinburgh, [www.epcc.ed.ac.uk](http://www.epcc.ed.ac.uk)”

Note that this presentation contains images owned by others. Please seek their permission before reusing these images.

# Outline

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

# Message Passing

Process-based parallelism

# Analogy

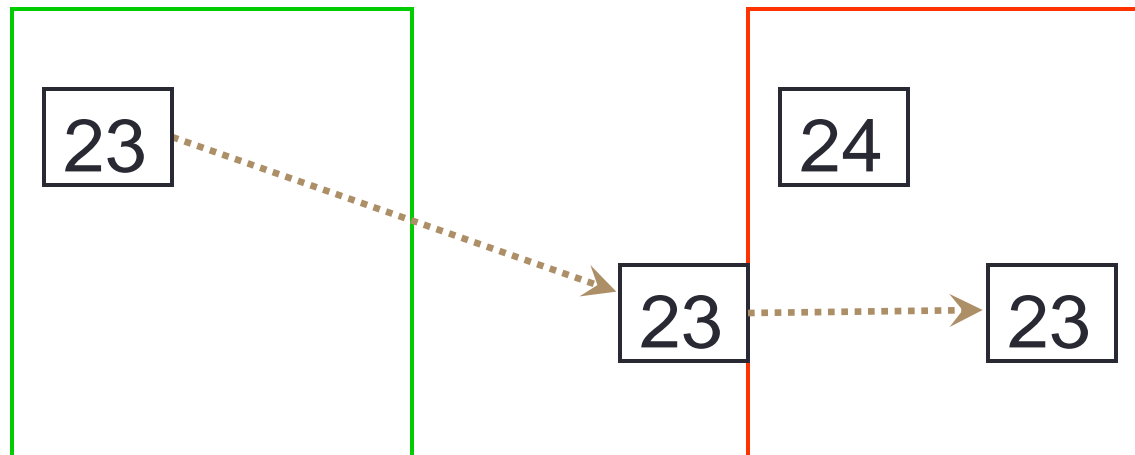
- Two whiteboards in different single-person offices
  - the distributed memory
- Two people working on the same problem
  - the processes on different nodes attached to the interconnect
- How do they collaborate?
  - to work on single problem
- Explicit communication
  - e.g. by telephone
  - no shared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



# Process communication

	<b>Process 1</b>	<b>Process 2</b>
Program	<code>a=23</code>	<code>Recv(1, b)</code>
	<code>Send(2, a)</code>	<code>a=b+1</code>

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

# Shared Variables

Threads-based parallelism

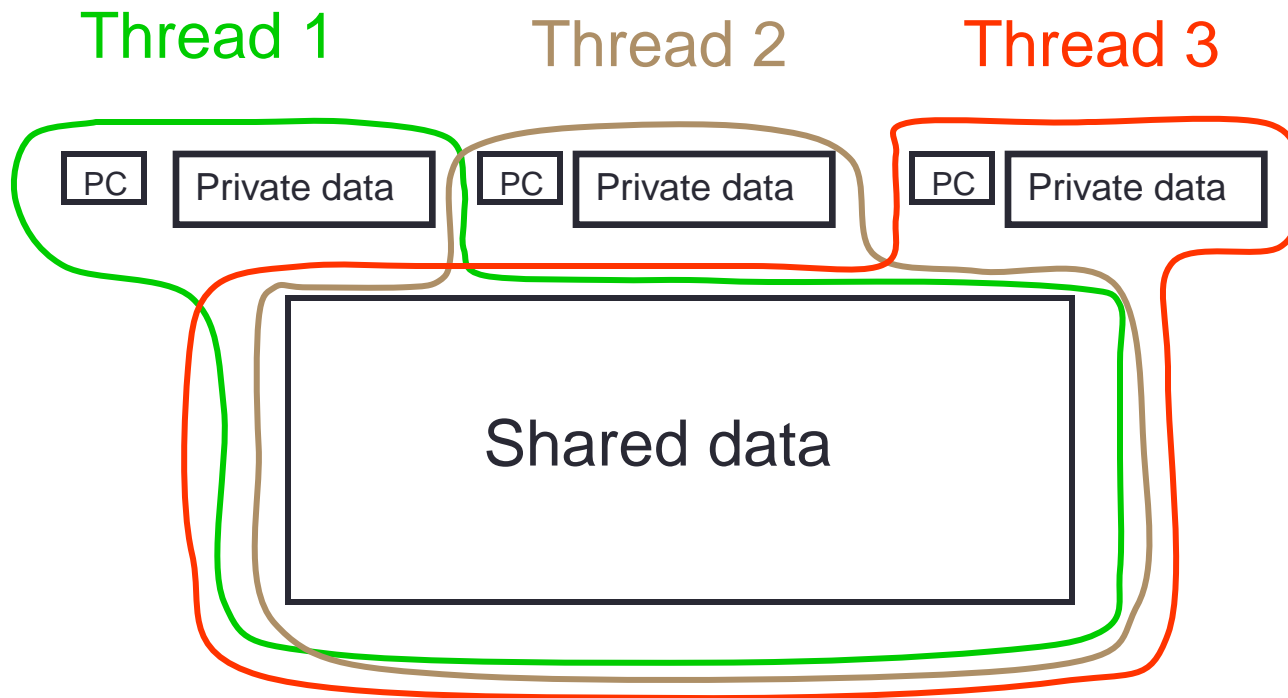


# Analogy

- 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



# Threads



# Thread Communication

Thread 1

Thread 2

Program

`mya=23`  
`a=mya`

`mya=a+1`

Private  
data

23

24

Shared  
data

23

# 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!

# Threads: Summary

- Shared blackboard a good analogy for thread 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 directives
  - supports common parallel patterns
  - e.g. loop limits computed by the compiler
  - e.g. summing values across threads done automatically

# OpenMP fork / join model

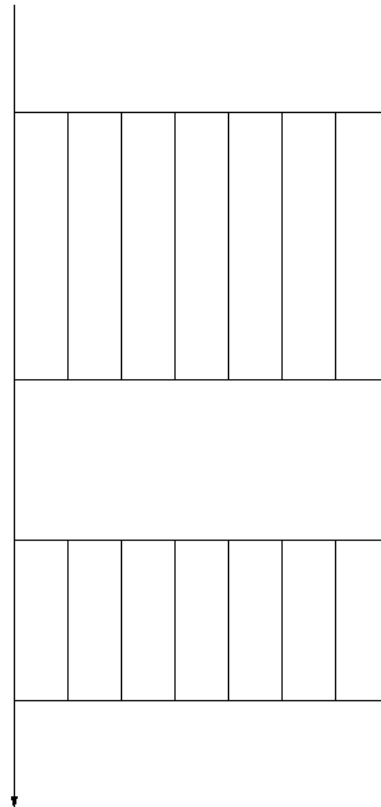
## Sequential part

## Parallel region

## Sequential part

## Parallel region

## Sequential part



PROGRAM FRED

! \$OMP PARALLEL

```
!$OMP END PARALLEL
```

! \$OMP PARALLEL

```
! $OMP END PARALLEL
```

```
int main() {
```

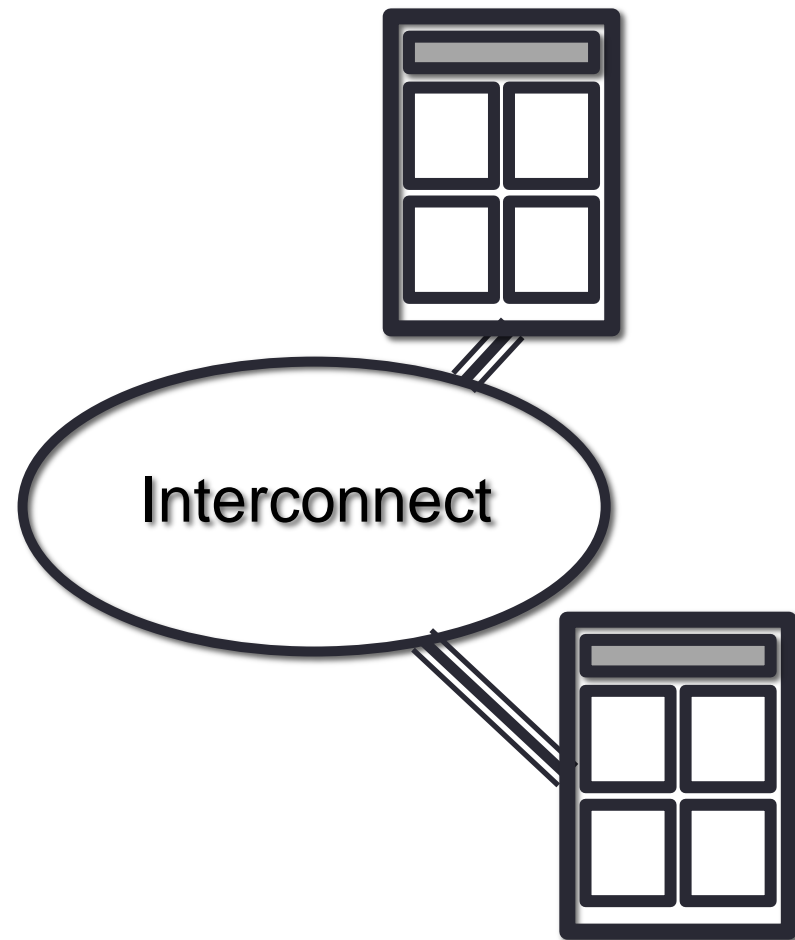
```
#pragma omp parallel
```

```
#pragma omp parallel
```

# Practicalities

How we use the parallel models

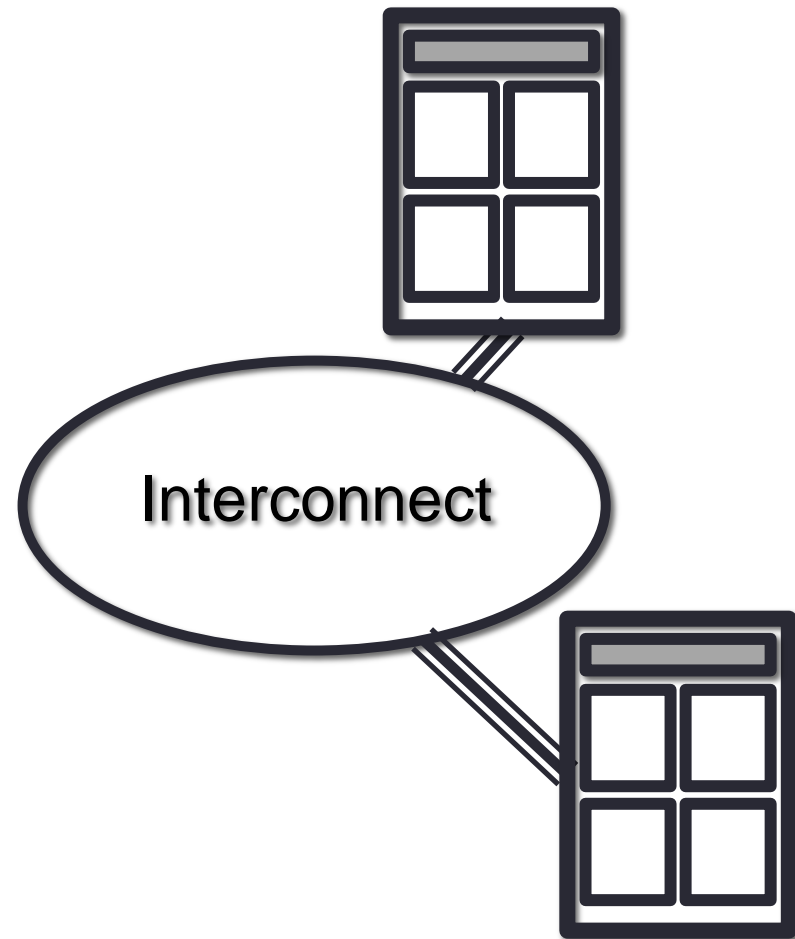
# Practicalities



- 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 processor-cores on the same node are fast
  - but remember they also share access to the network



# Hybrid MPI / OpenMP



- Take advantage of architecture
  - one MPI process per node
  - four OpenMP threads per process
    - one for each physical code