CUG Talk

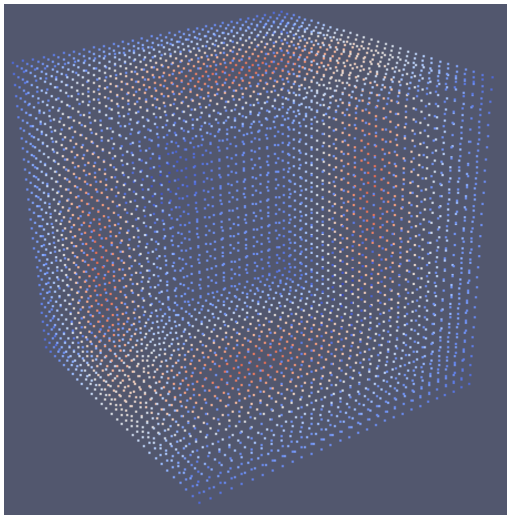
In-situ data analytics for highly scalable cloud modelling on Cray machines

Nick Brown, EPCC
nick.brown@ed.ac.uk

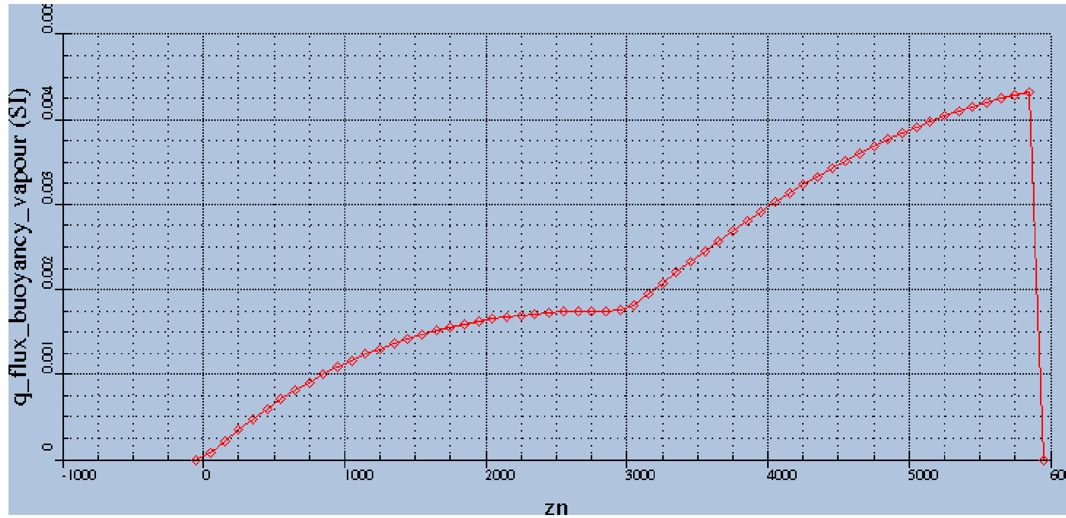
Met Office NERC Cloud model (MONC)

- Uses Large Eddy Simulation for modelling clouds & atmospheric flows
  - Written in Fortran 2003 due to scientist familiarity, uses MPI for parallelisation
  - Designed to be a community model which will be accessible to be changed by non expert HPC programmers and scale/perform well.
  - For use not just by Met Office scientists, but also those in the wider weather/climate community
- Replaces an older model, the LEM from the 1980s
  - From 22 million to billions of grid points
  - From 256 cores to many thousands

A challenge for analysis



Prognostics



Diagnostics

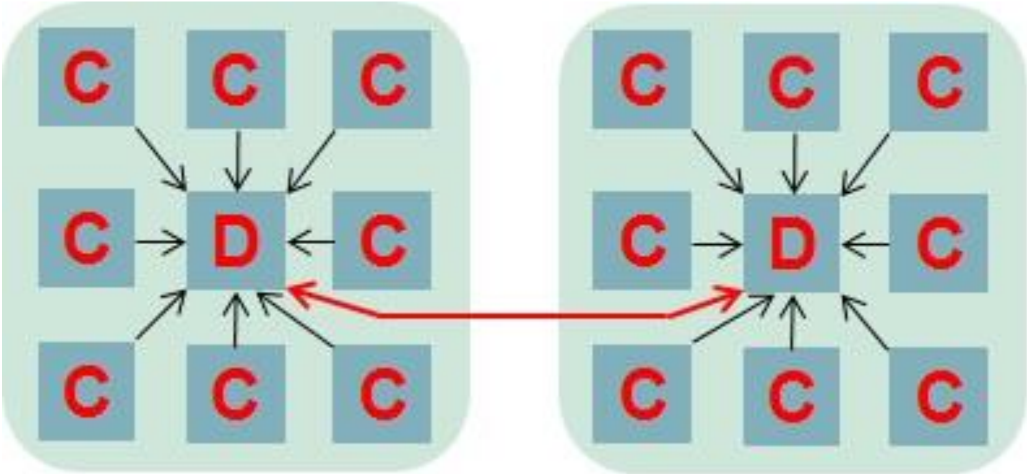
- With much larger domains (billions of grid points) how can we best analyse the data in a scalable fashion?
  - Previous LEM model did this in line with computation, where the model would stop and calculate diagnostics before continuing with computation
  - Could write to disk and analyse offline

In-situ approach

- Have many computational processes and a number of data analytics cores
  - Typically one core per processor is dedicated to IO, serving the other cores running the computational model
  - Computational cores "fire and forget" their data

- In-situ as raw data is never written out
  - Would be too time consuming

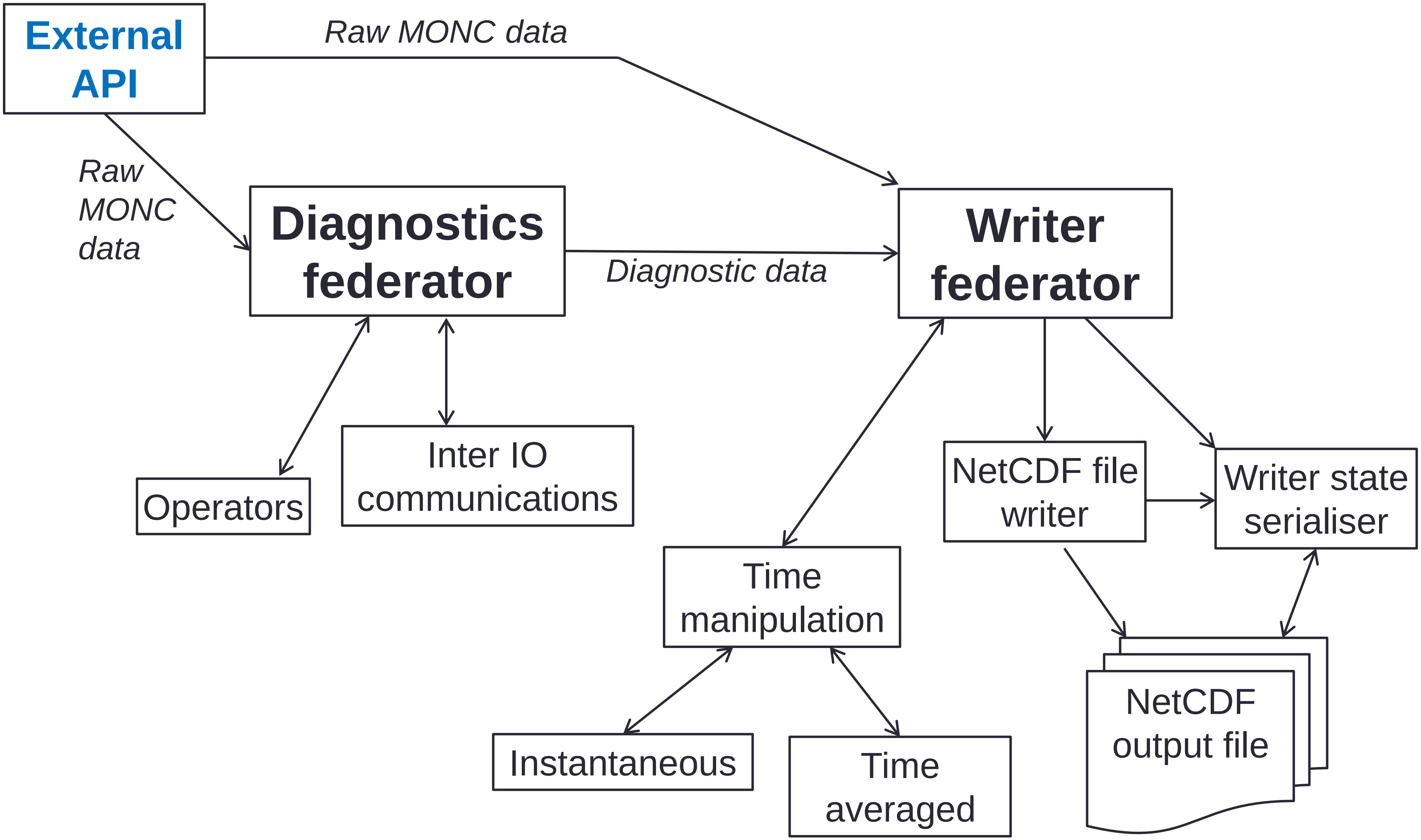
- Avoids blocking the computational cores for analytics and IO



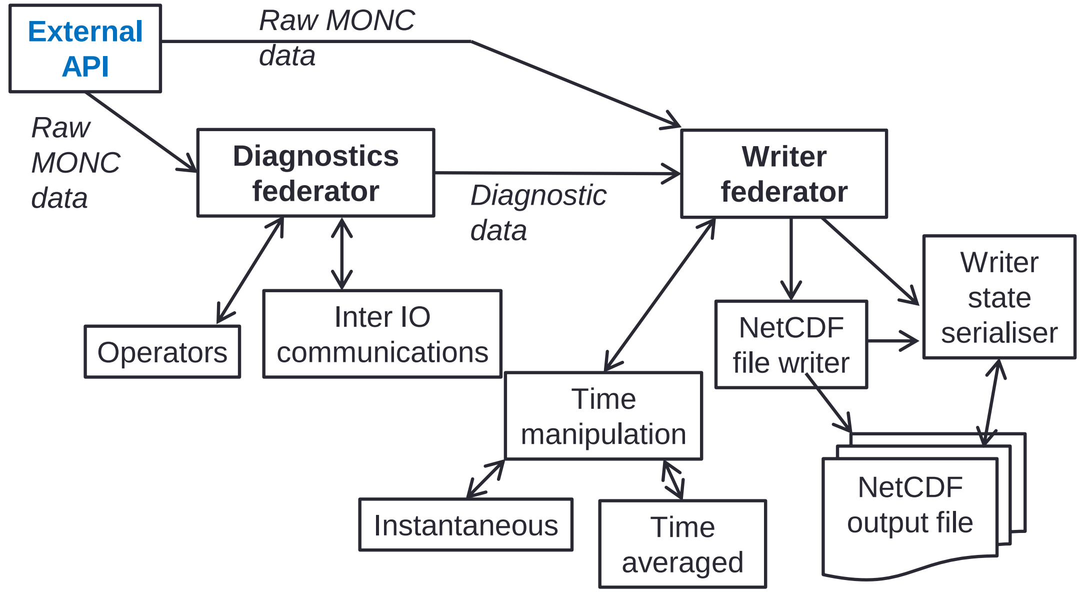
Existing approaches……

- Some existing approaches:
  - XIOS
  - Damaris
  - ADIOS
  - Unified Model IO server

- We need:
  - To support dynamic time stepping
  - Checkpoint-restart of the IO server itself
  - Bit reproducibility
  - Scalability and performance
  - Easy configuration & extendibility


External API
Raw MONC data
Raw MONC data
Diagnostics federator
Diagnostic data
Writer federator
Operators
Inter IO communications
Time manipulation
NetCDF file writer
Writer state serialiser
Instantaneous
Time averaged
NetCDF output file

- Define the data from MONC
  - Arrays, scalars or maps
  - Mandatory (default) or optional
  - A unique subset of a field (collective) or not. If collective need to provide sizes per dimension; z, zn, y ,x and qfields
  - Integer, double, float, boolean, string

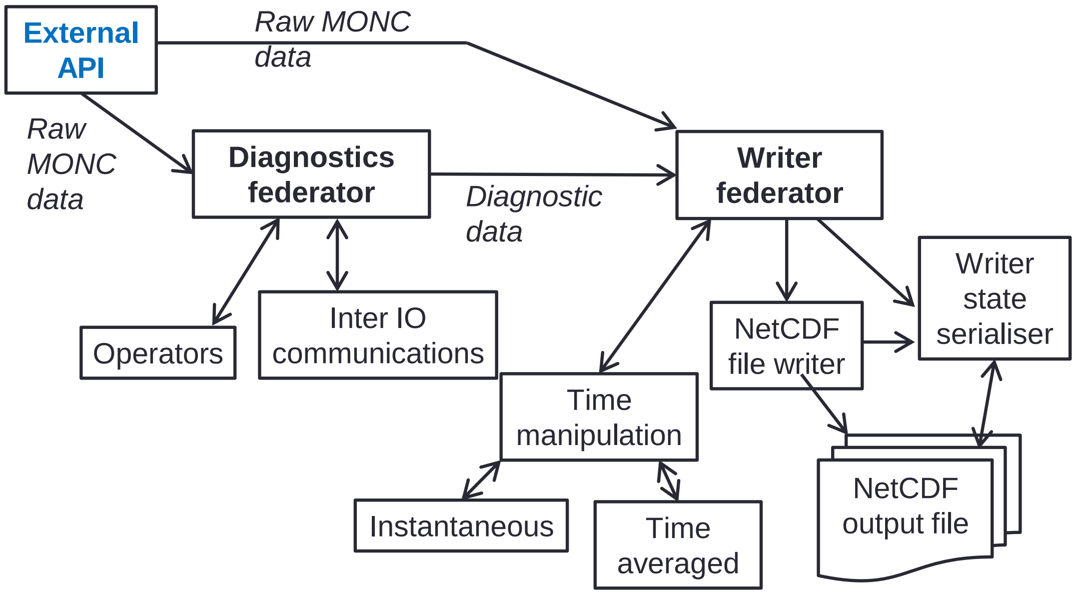


- The IO server expects this data every “frequency” timesteps
- On registration the MONC process is told what data it should send & when. MONC process tells the IO server the sizes.

```
<data-definition name="data_parcel" frequency=2>
  <field name="u" type="array" data_type="double" size="z,y,x" collective=true/>
  <field name="vwp_local" type="array" data_type="double" optional=true/>
</data-definition>
```

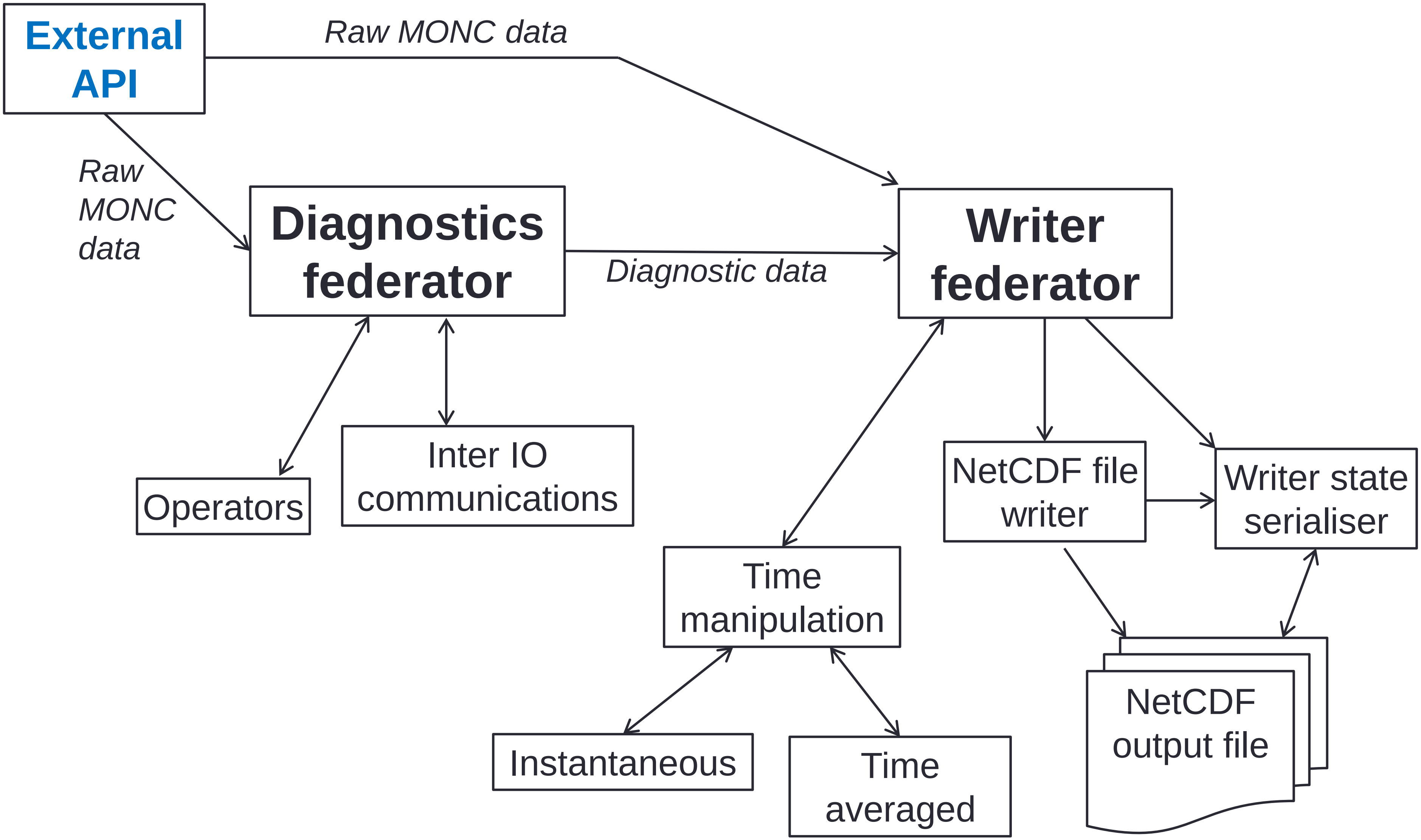
epcc

- Define the diagnostics & its attributes
  - Along with how to generate this diagnostic
- Organised as communication and operators



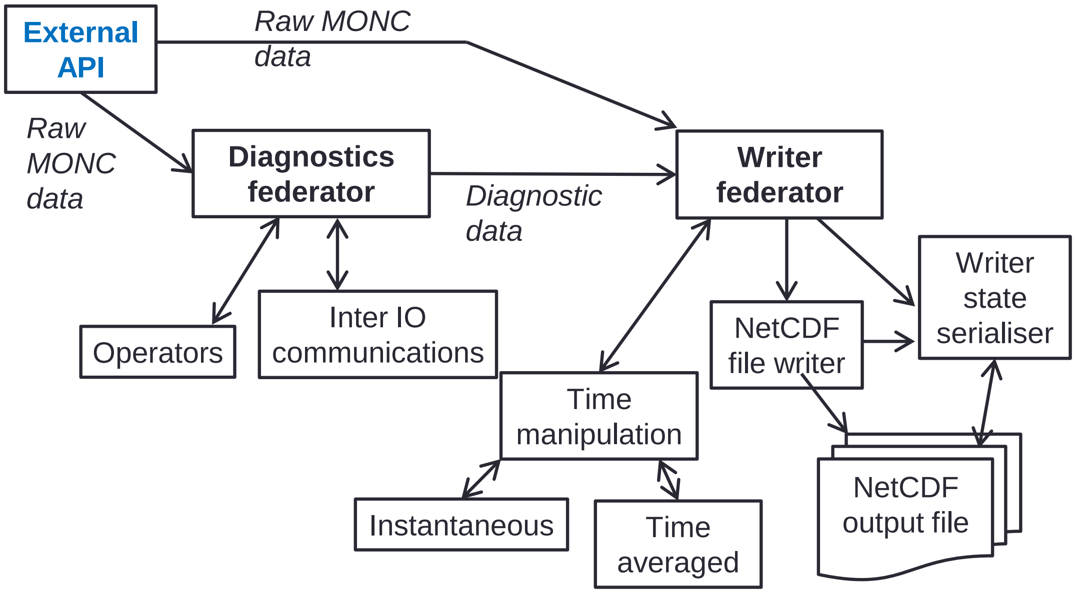

```
<data-handling>
  <diagnostic field="VWP_mean"  type="scalar" data_type="double" units="kg/m^2">
    <operator name="localreduce" operator="sum" result="VWP_mean_loc_reduced"
         field="vwp_local"/>
    <communication name="reduction" operator="sum" result="VWP_mean_g"
         field="VWP_mean_loc_reduced" root="auto"/>
    <operator name="arithmetic" result="VWP_mean"
         equation="VWP_mean_g/({x_size}*{y_size})"/>
  </diagnostic>
<data-handling>
```

epcc


External API
Raw MONC data
Raw MONC data
Diagnostics federator
Diagnostic data
Writer federator
Operators
Inter IO communications
NetCDF file writer
Writer state serialiser
Time manipulation
Instantaneous
Time averaged
NetCDF output file

User configuration - writing

```
<group name="3d_fields">
  <member name="w"/>
  <member name="u"/>
</group>
```

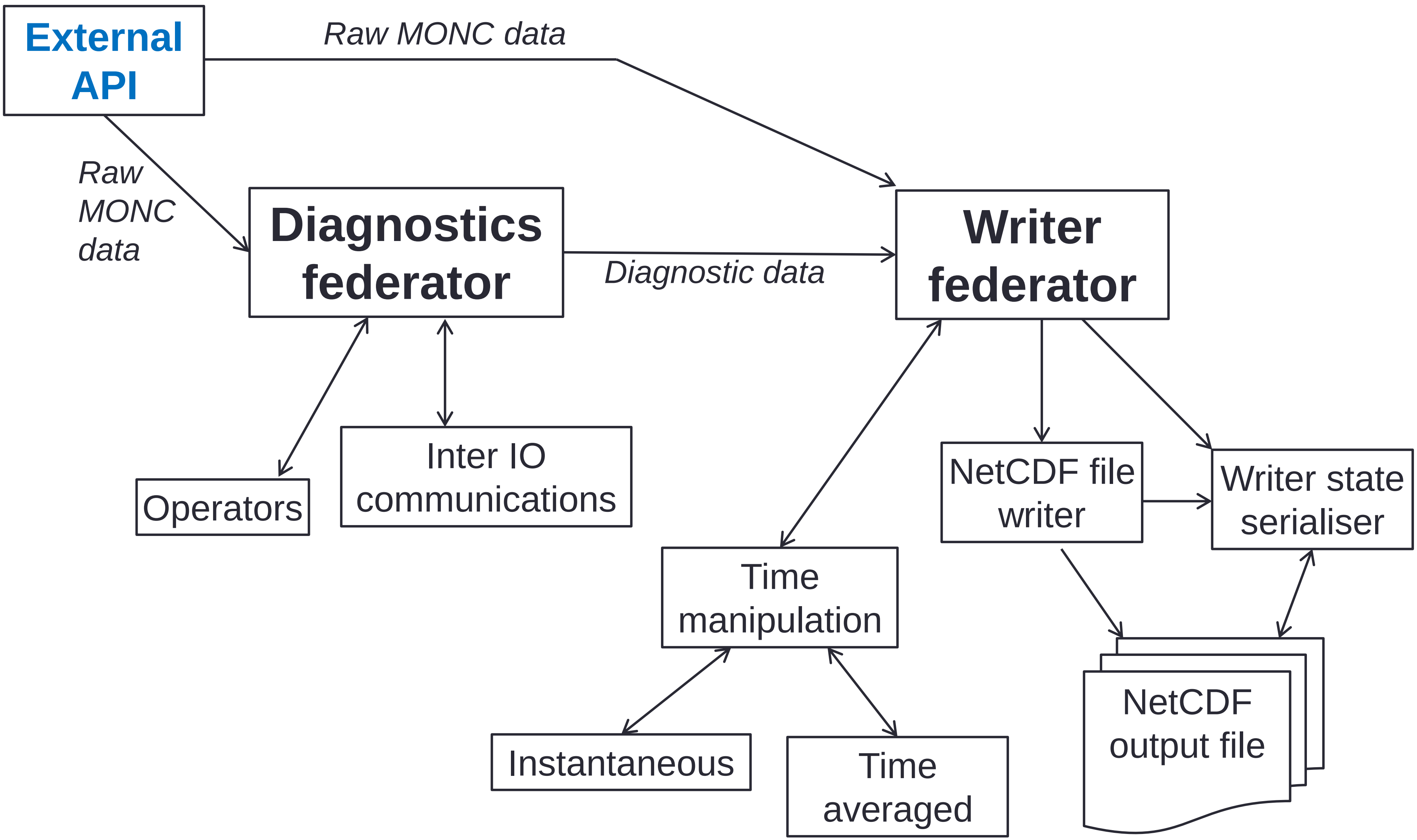



```
<data-writing>
  <file name="profile_ts.nc" write_time_frequency="100.0" title="Profile">
      <include field="VWP_mean" time_manipulation="averaged"
           output_frequency="2.0"/>
      <include group="3d_fields" time_manipulation="instantaneous"
           output_frequency="5.0"/>
  </file>
</data-writing>
```

Reuse of configuration

- Numerous existing XML configurations provided which can be included by the user
- Raises the issue of name conflicts
  - Handled by the concept of namespaces

```
#include “checkpoint.xml”

#include “profile_fields.xml”

#include “scalar_fields.xml”
```

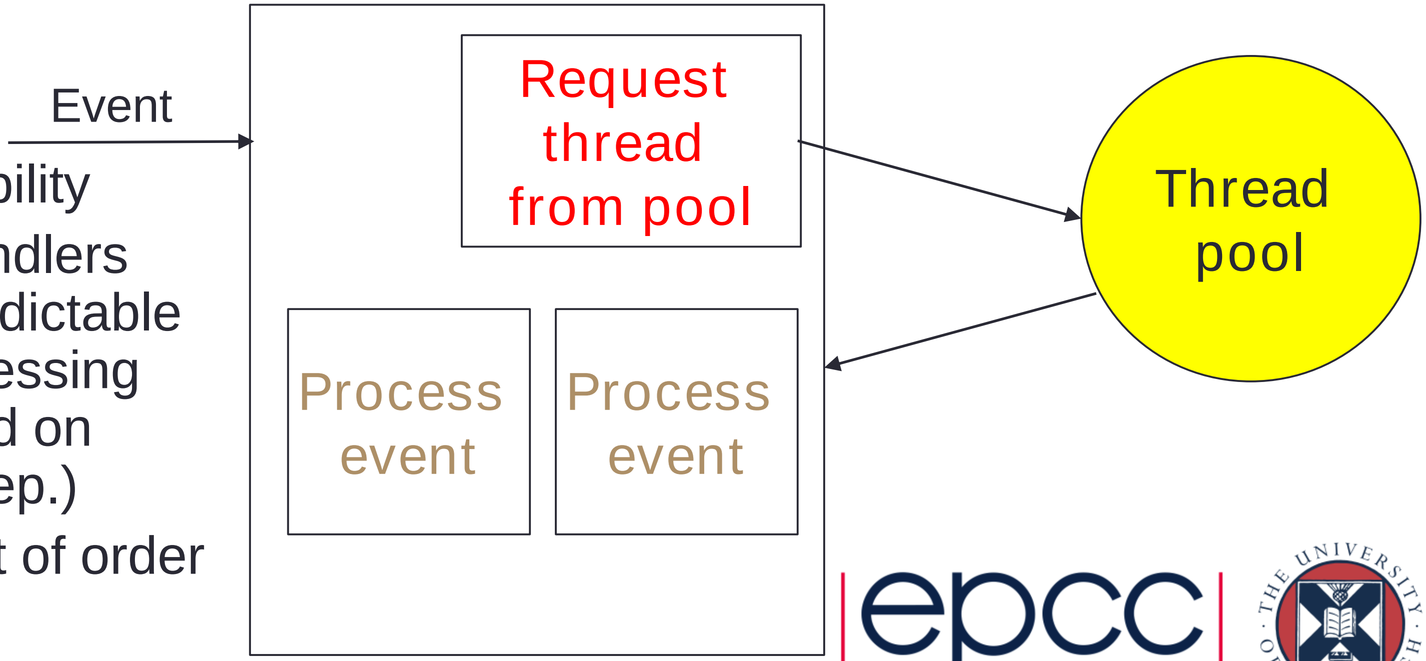


External API
Raw MONC data
Raw MONC data
Diagnostics federator
Diagnostic data
Writer federator
Operators
Inter IO communications
Time manipulation
NetCDF file writer
Writer state serialiser
Instantaneous
Time averaged
NetCDF output file

Event handling

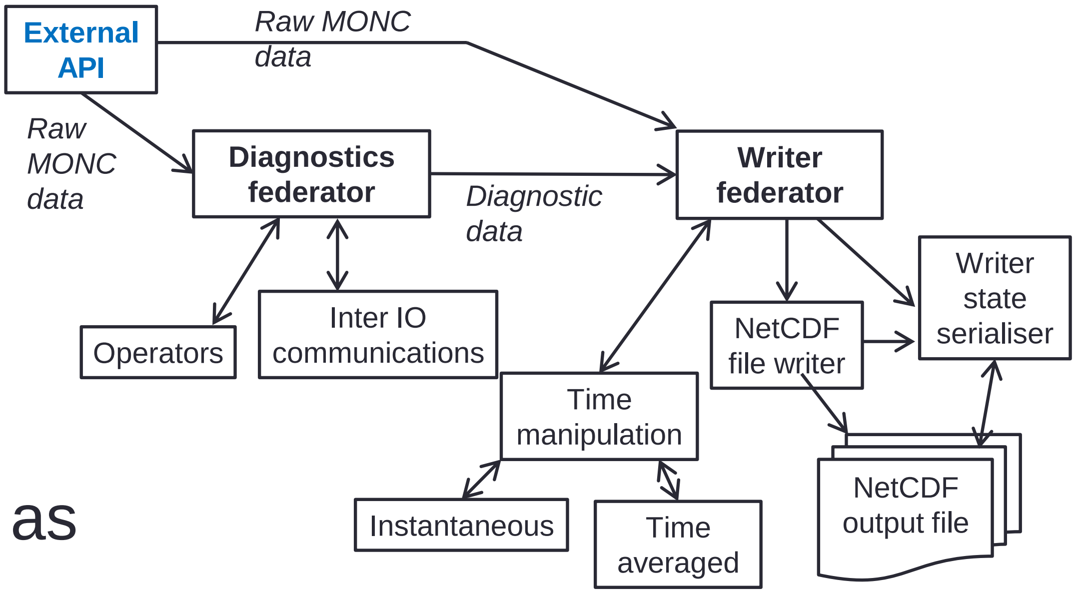
- The federators and their sub activities are event handlers
  - Process events concurrently by assigning these to these from a pool
  - Aids asynchronous data handling
    - As soon as data arrives process it
    - Internal state of event handlers needs protection (mutexes/rw locks)

- Challenge:
  - Bit reproducibility
  - For some handlers enforce a predictable order of processing events (based on model timestep.)
  - Queue up out of order events



Inter-IO communications challenge

- We promote asynchronicity and processing of events out of order where possible
- Many inter IO communications involve collective operations (such as a reduction)
  - We would like to use MPI, but issue order of collectives matters (i.e. if IO server 1 issues a reduce on field A and then B, then all other IO servers must issue reductions in that same order)
  - But ensuring this would require additional overhead and/or coordination
- Solution: Abstract through active messaging



Active messaging for inter IO comms

- These communication calls additionally provide
  - UniqueID: matching collectives even if they are issued out of order
  - Callback : Handling procedure called on the root when the data arrives
- inter_io_reduce(data, data_type, root, uniqueId, callback)
  - When this reduction is completed on the root, a thread is activated from the pool and calls the handling function (typically in the event handler)
  - The Unique ID here is the concatenation of field name and timestep
  - Built upon MPI P2P communication calls

```
call inter_io_reduce(data, type, 0, fieldname//"_"//timestep, handler)


subroutine handler(data, data_type, uniqueId)
   .........
end subroutine handler
```

Active messaging for synchronisation

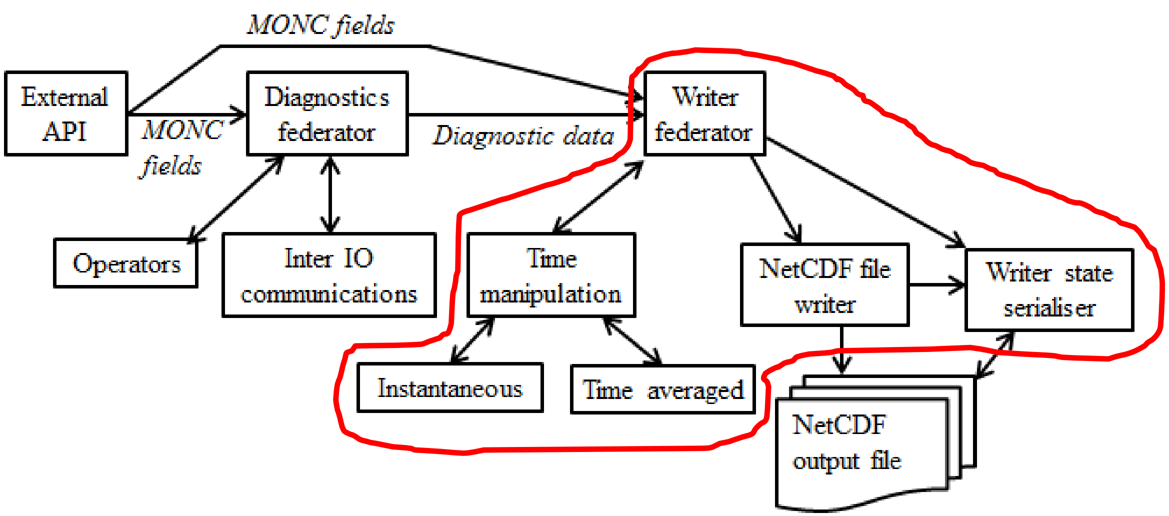
- File writing is done by NetCDF
  - But this is not thread safe, so crucial that only one thread per IO server process calls NetCDF functions concurrently
  - Many NetCDF calls are collective (i.e. will block until called on all processes in the communicator.)
  - NetCDF *close* is an example of this, where each process will block here until same call issued on all other processes
    - Which we don't want, as it will block access to NetCDF (and MPI)
- Active messaging barrier calls the handling function on every process once a barrier has been issued by all processes

```
call inter_io_barrier(filename_uniqueID, closeHandler)

subroutine closeHandler(uniqueId)

   ………

   call close_netcdf_file(………)

end subroutine closeHandler
```

epcc

Checkpointing

- We need to support checkpoint-restart of the IO server and analytics
  - This is challenging due to the amount of asynchronicity, especially in the analytics
- Wait for all analytics to that point to complete and just store the state of the writer federator

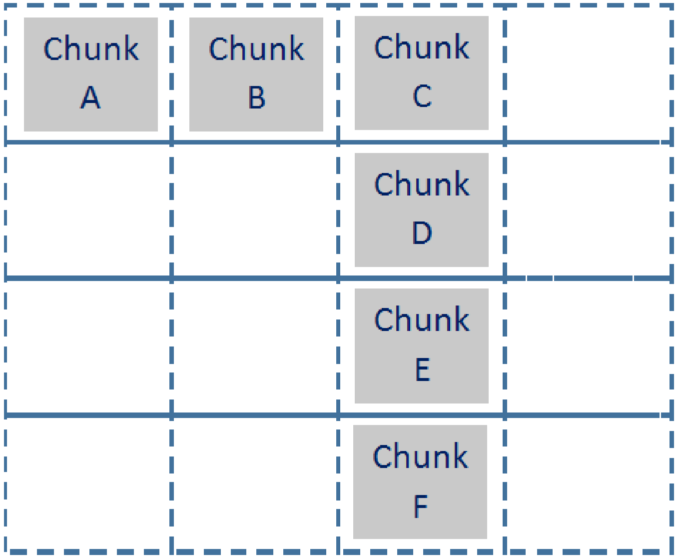


- Two step process
  - Walk the state, lock it and determine the amount of memory needed
  - Serialise state, write this and unlock

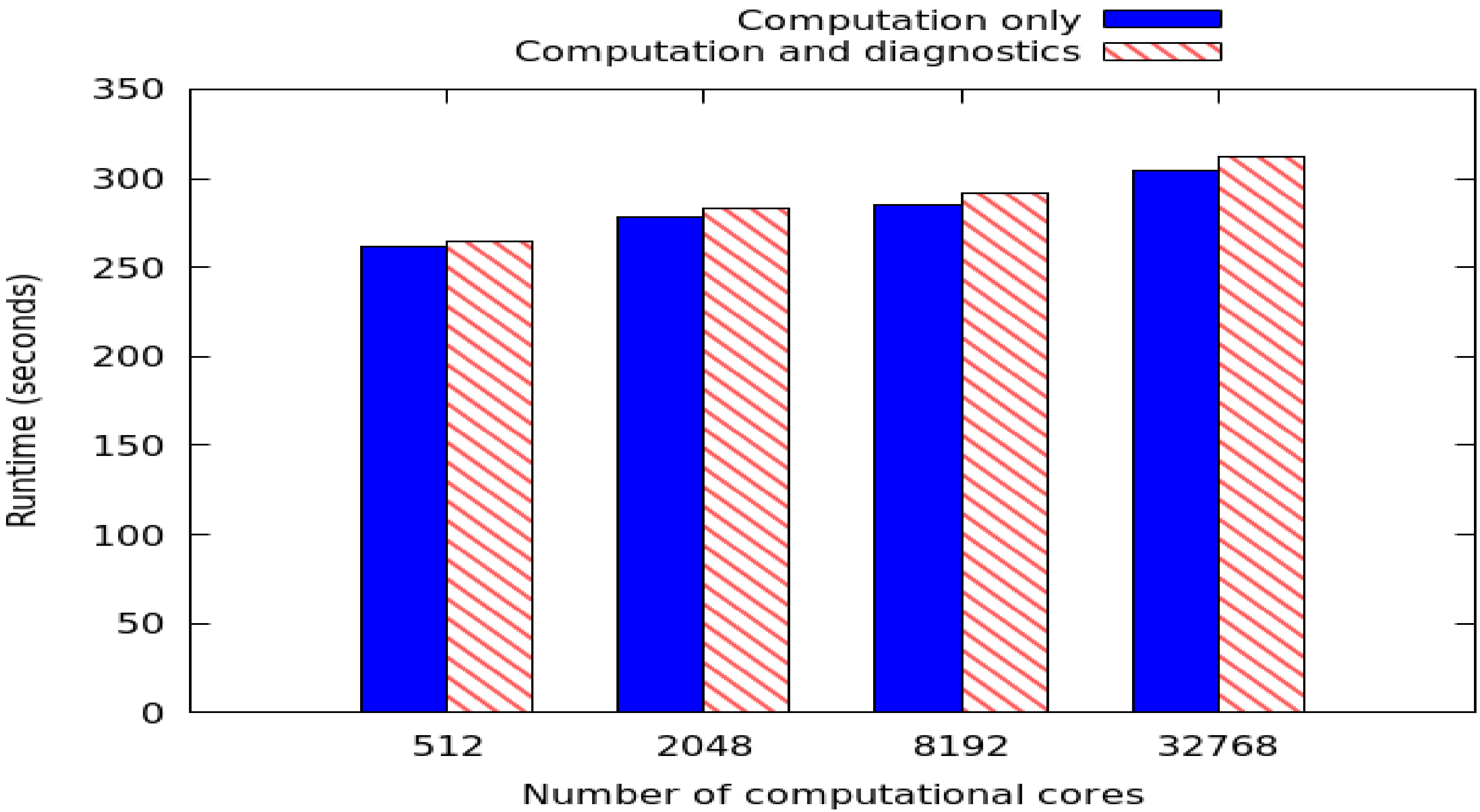
Prognostic writing optimisation

- IO servers servicing many computational cores means that the data is naturally split up
  - For prognostic field writes this can be a problem as it is very inefficient to do lots of independent writes to the file

- Want to perform minimal collective writes instead
  - Search through the domain of local computational cores and merge data chunks together where possible to produce smaller number of large contiguous chunks
  - Must do the same number of writes on every core, some perform empty writes if not enough chunks



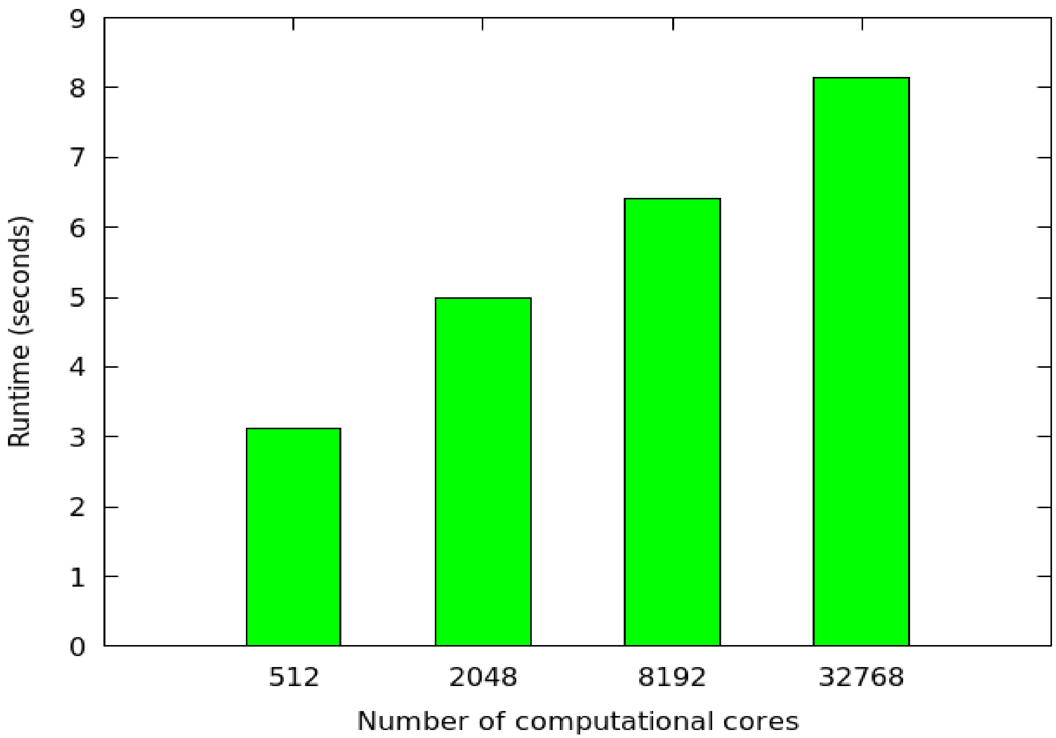
Performance and scalability



- Standard MONC stratus cloud test case
  - Weak scaling on Cray XC30, 65536 local grid points
  - 232 diagnostic values every timestep, time averaged over 10 model seconds. File written every 100 model seconds. Run terminates after 2000 model seconds.

IO overhead as a metric

- To measure the performance of the IO server and different configurations we adopt an overhead metric
  - This is the time difference from the MONC communication that should trigger a write, to that write having being physically performed

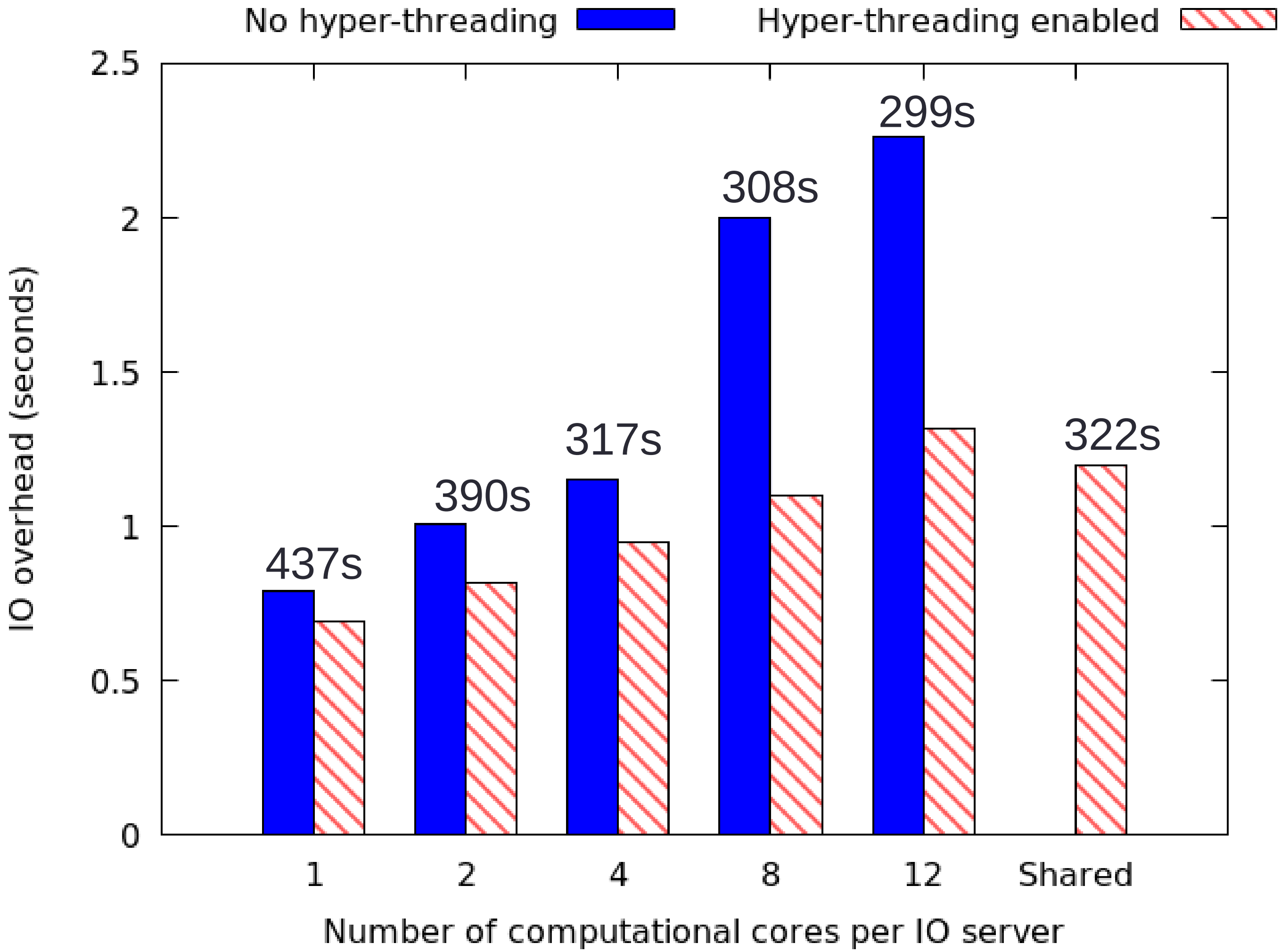


Configuration	Overhead
MPI serialised	8.92
MPI multiple	12.02
MPI serialised + hyperthreading	8.14
MPI multiple + hyperthreading	9.71

epcc

THE UNIVERSITY OF EDINBURGH

Performance on the KNL



- Cray XC40, 64 core KNL 7210
  - Same stratus test case. 3.3 million grid points
  - As MONC is not multi-threaded can we run one IO server per MONC on the hyper-thread?

Conclusions and further work

- We have discussed our approach for in-situ data analytics and IO
  - Which performs and scales well up to 32768 computational cores
  - As well as the architecture, challenges and lessons learnt from implementing this

- Extend the active messaging layer to build upon something other than MPI
- Plug in other writing mechanisms such as visualisation tools
- Extract this from MONC to enable others to integrate with their models