The INTERTWinE project and in-situ data analytics

http://www.intertwine-project.eu

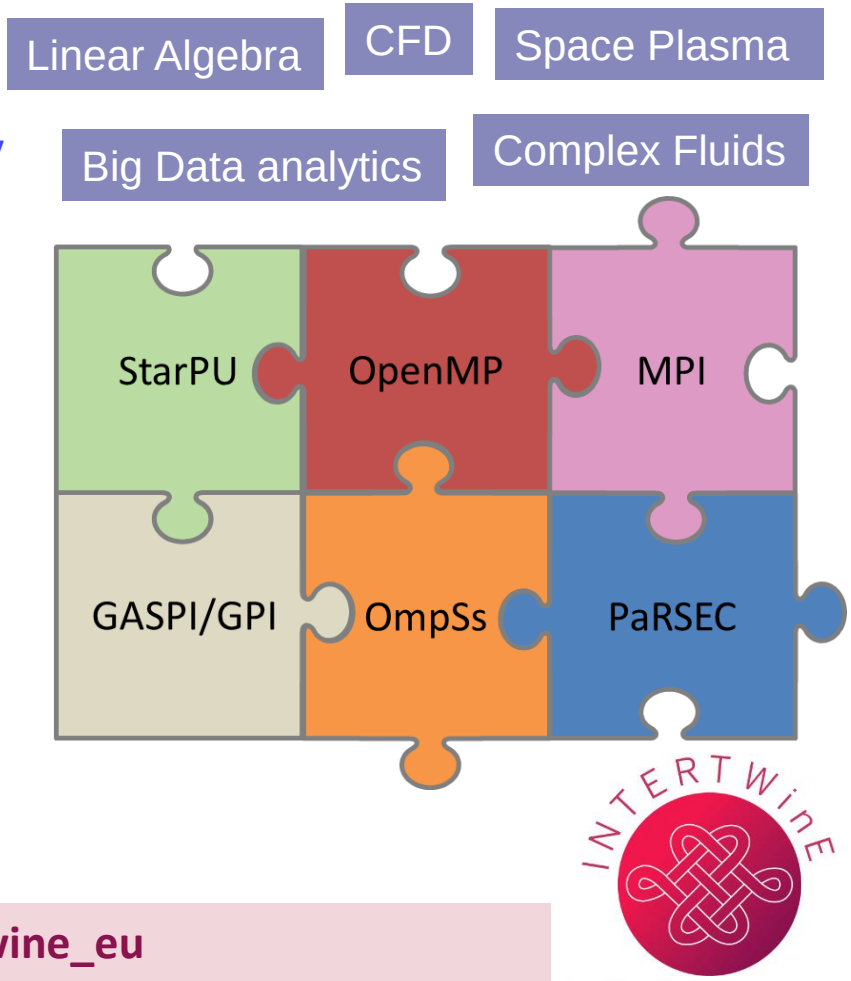
Nick Brown, EPCC – n.brown@epcc.ed.ac.uk

This project is funded from the European Union's Horizon 2020 Research and Innovation programme under Grant Agreement no. 671602.

INTERTWinE

Interoperability between programming models for scalable performance on extreme-scale supercomputers

- **Computational Resource Management**
  - Coordinated resource sharing for interoperability between runtime systems, libraries
- **Distributed Data Management**
  - Scalable, transparent data sharing on heterogeneous, distributed memory hierarchies
- **Engagement with HPC community**
  - Standards bodies: OpenMP, MPI, GASPI
  - Courses, workshops and Best Practice Guides

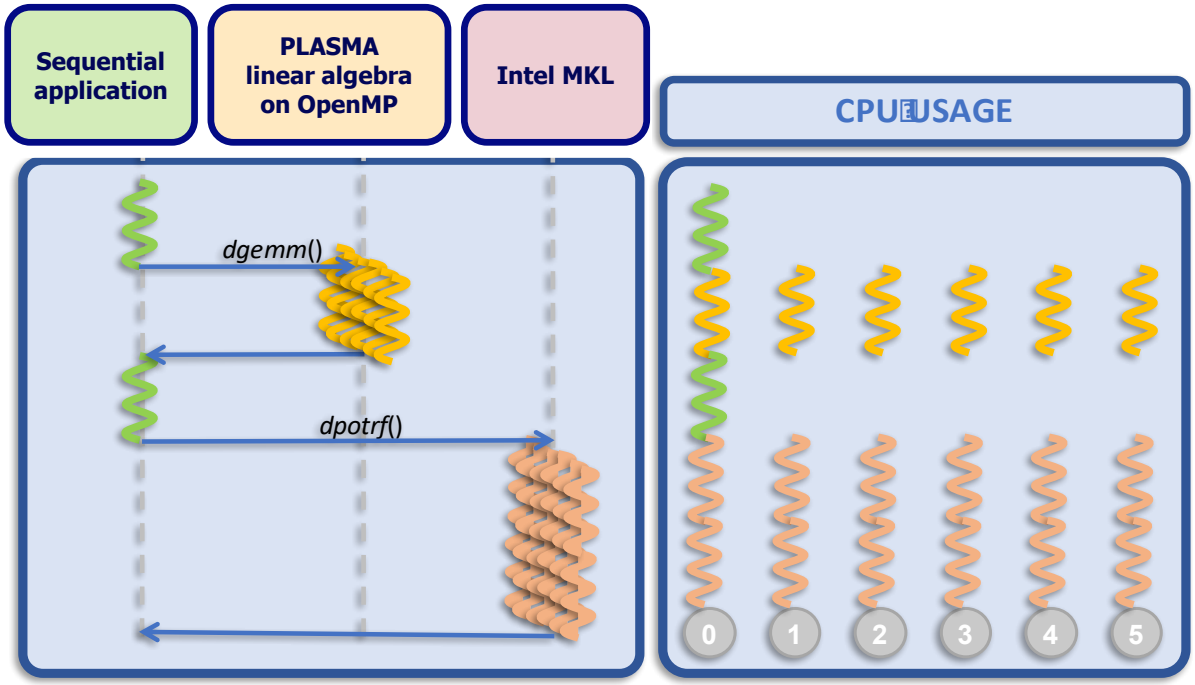


Follow INTERTWinE on Twitter: @intertwine_eu

Interoperable node-level resource sharing

Computational Resource Sharing

- **Multiple codes compete for CPU cores, accelerator devices on cluster nodes**
  - Application threads
  - Numerical libraries threads
  - Runtime systems threads
  - Communication library threads

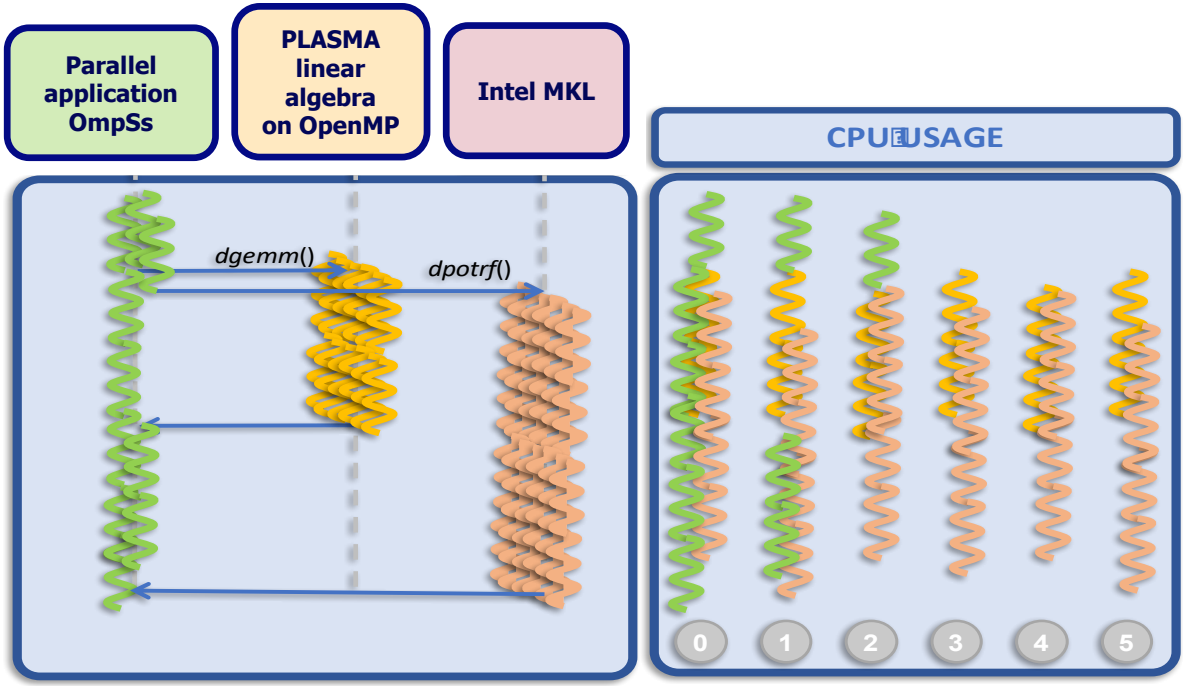


- **Interference leads to resource over-subscription or under-subscription on cluster nodes**
  - Interoperability?
- **Need coordinated resource sharing:**
  - Ability to express general resource needs
  - Ability to express dynamic resource requirements:
    - computational-heavy periods, idleness periods

Interoperable node-level resource sharing

Computational Resource Sharing

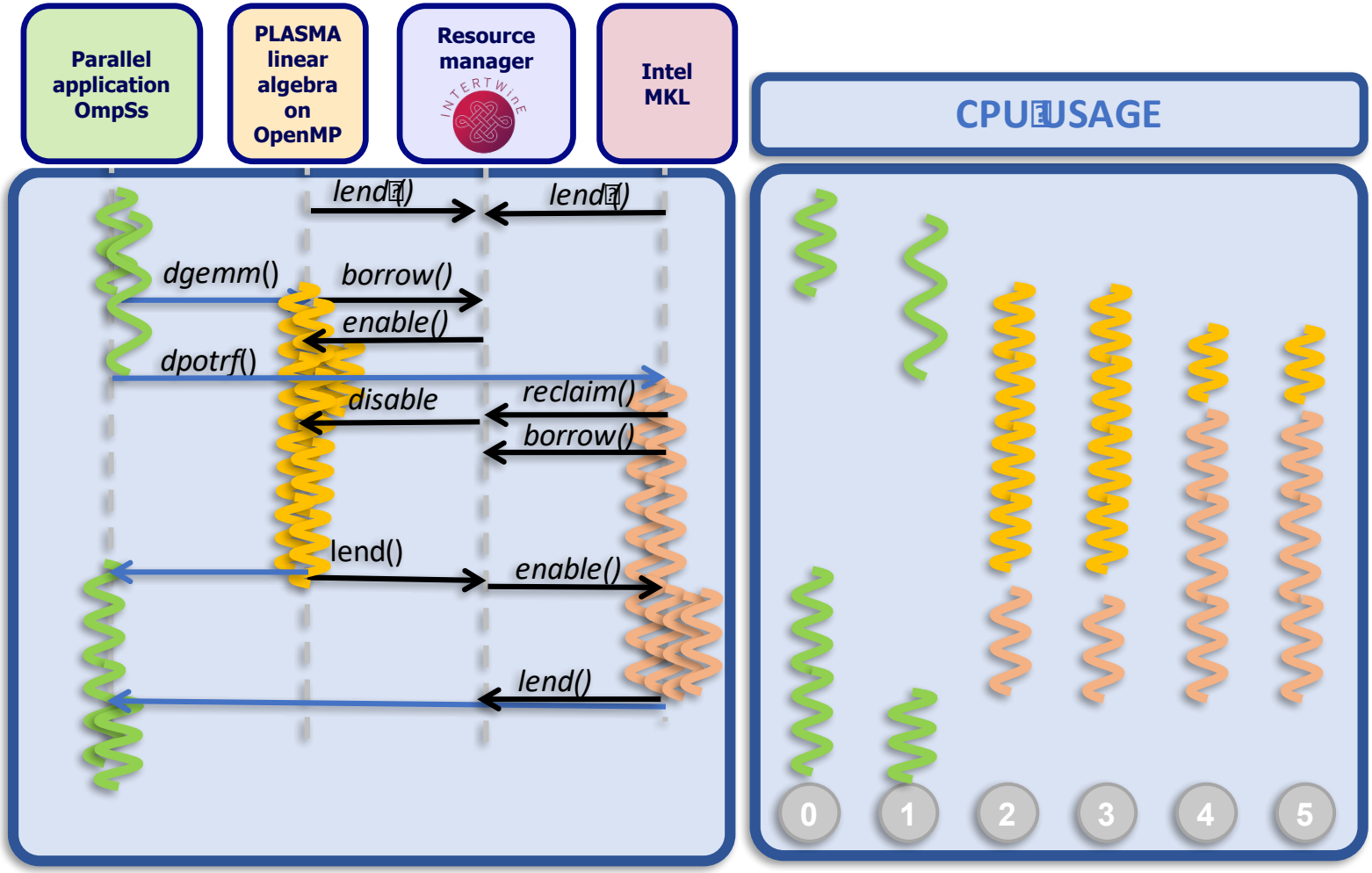
- **Multiple codes compete for CPU cores, accelerator devices on cluster nodes**
  - Application threads
  - Numerical libraries threads
  - Runtime systems threads
  - Communication library threads



- **Interference leads to resource over-subscription or under-subscription on cluster nodes**
  - Interoperability?
- **Need coordinated resource sharing:**
  - Ability to express general resource needs
  - Ability to express dynamic resource requirements:
    - computational-heavy periods, idleness periods

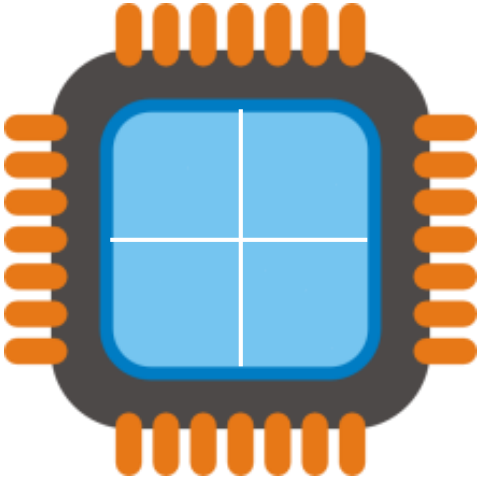
INTERTWinE — Resource Manager APIs
Dynamic Resource Sharing

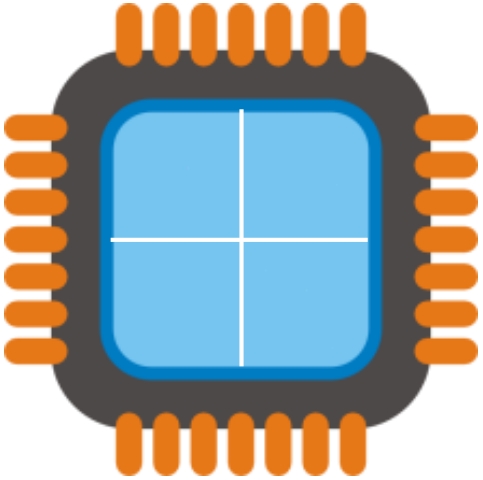
Enable runtime systems to dynamically negotiate and adjust resource usage



MPI and tasks deadlock issue

S1
S2
S3
S4
S5
S6
S7
S8
S9





R1
R2
R3
R4
R5
R6
R7
R8
R9

MPI and tasks deadlock issue

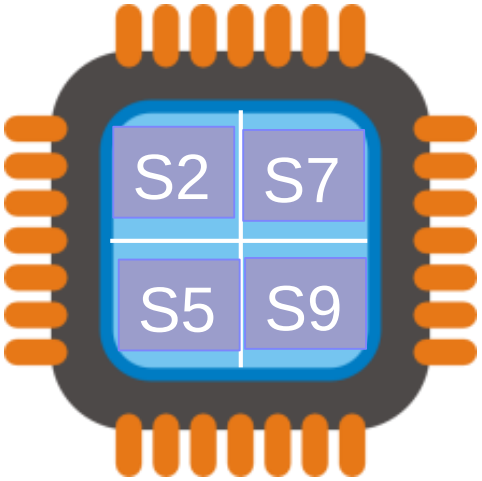
S1

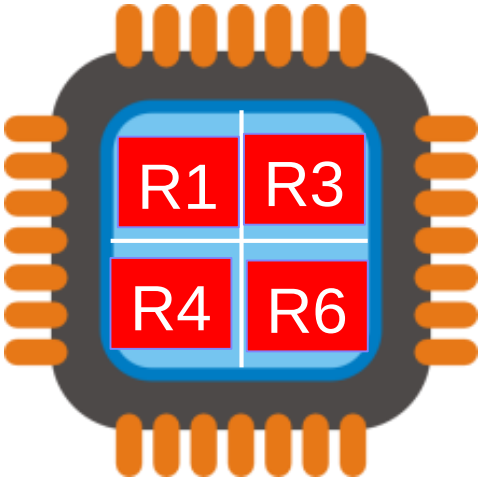
S3

S4

S6

S8





R2

R5

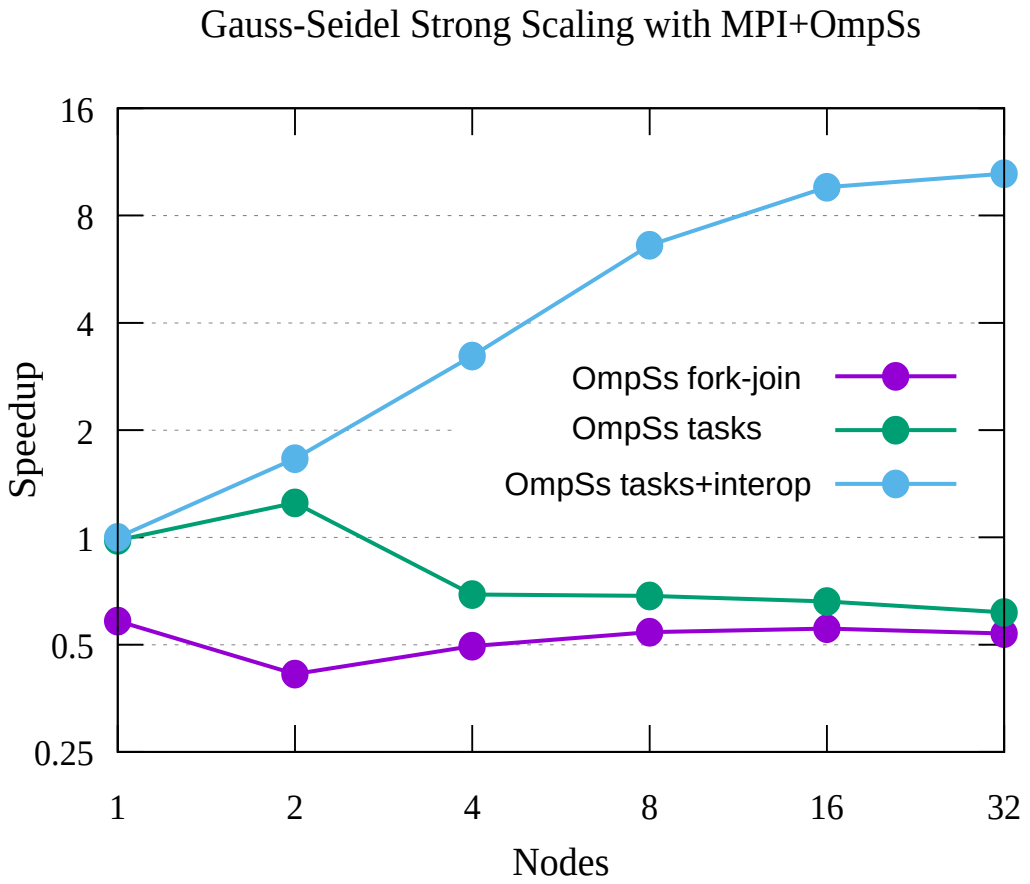
R7

R8

R9

Interoperability MPI for task-based communications

- **Using the resource manager we have extended the MPI library**
  - The resource manager will pause threads/tasks that issue blocking communication call (lend)
  - MPI library polls outstanding requests and uses the resource manager to rewake threads when communications complete (reclaim)

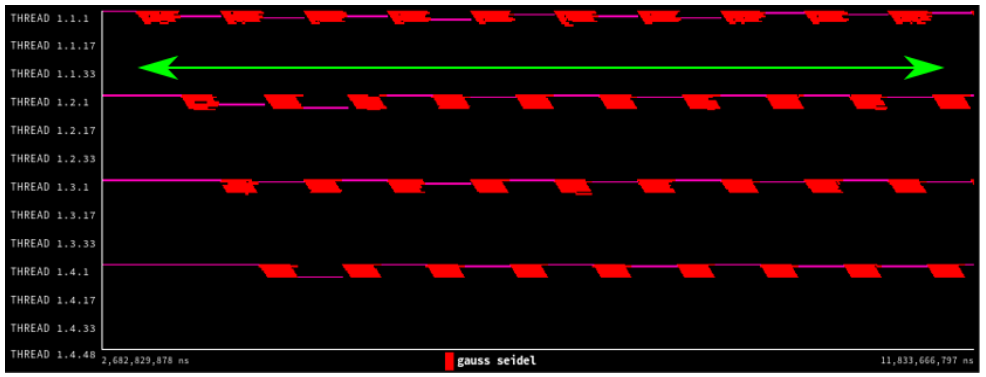


Strong scaling in 2 dimensions, global size is 64k by 64k.

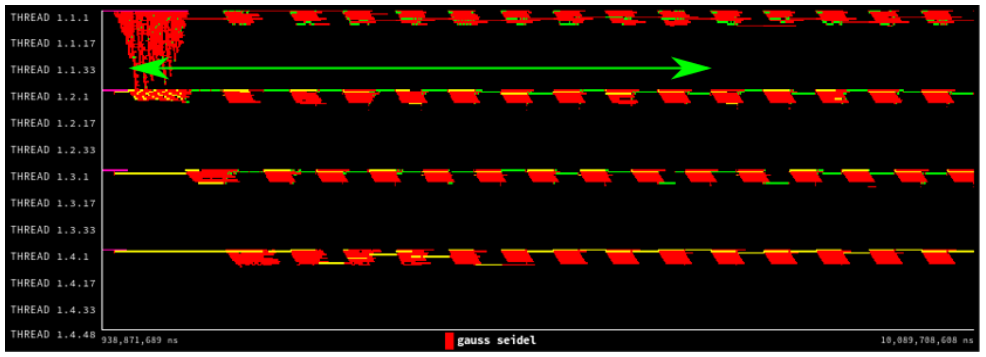
100 iterations and 48 cores per node (MareNostrum4.)

Task multiple parallelism: 4 nodes (48 cores per node)

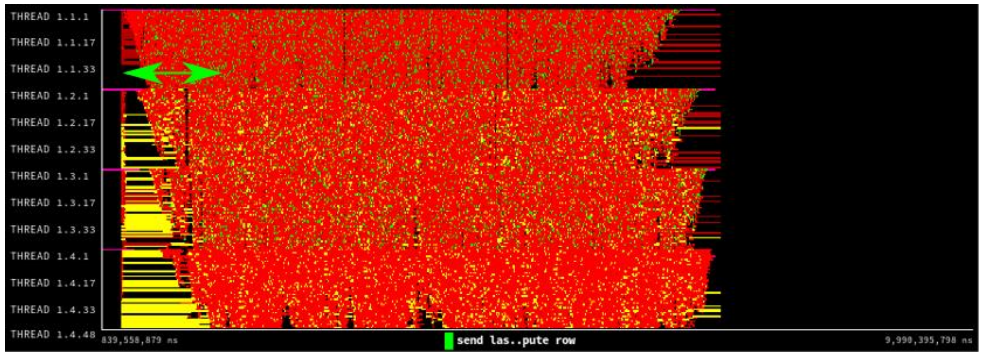
OmpSs Fork / join



OmpSs tasks



OmpSs tasks + interop



- **Interoperability MPI, built upon the resource manager significantly cuts down the amount of synchronisation required**
  - Lots more threads can be active

Green arrow represents 10 iterations

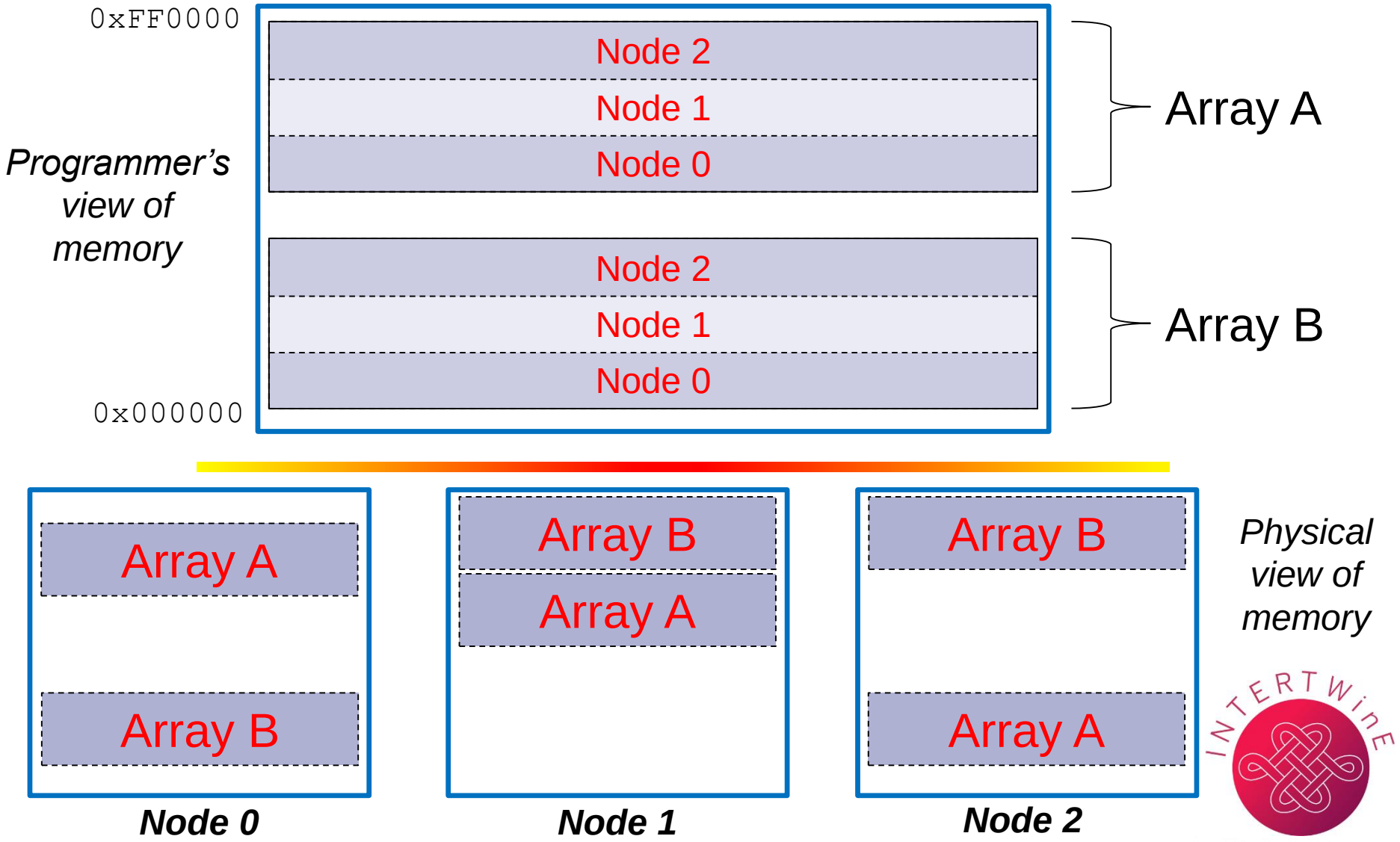
Implicit task and distributed memory interoperability

```
for (i=0; i<x; i++) {
    for (j=0; j<y; j++) {
#pragma omp task in(A[i,j], B[i,j]) out(A[j,i], B[j,i])
            my_task(A[i,j], B[i,j], A[j,i], B[j,i])
}}
```

- **Schedules lots of tasks, which will not run until their input dependencies are met.**
  - Each produces outputs that other tasks may depend upon

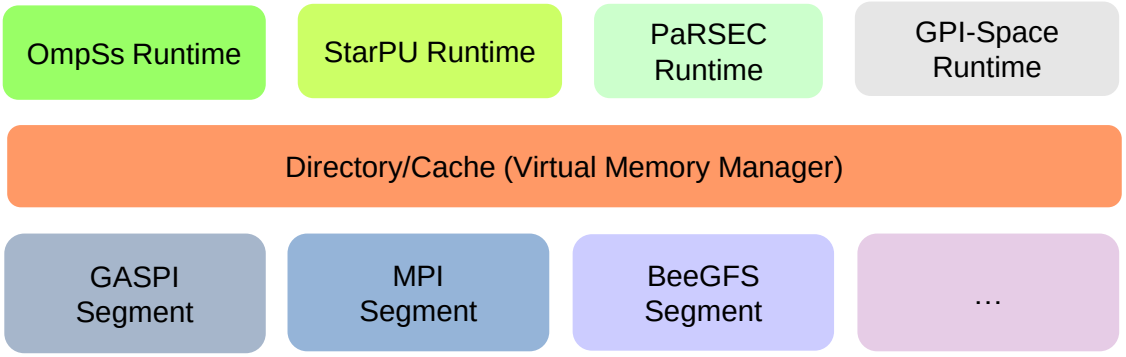
- **Task-based models tend to be limited to a single memory space**
  - A significant limitation and hence we want to extend this to distributed memory machines
  - For instance, the array A might be distributed amongst the nodes
  - When the task starts, the data it needs is available (irrespective of where that is located.) When the task completes the resulting data is available to any other task.

Directory/cache: For distributed shared memory



Directory/cache integration and transport layers

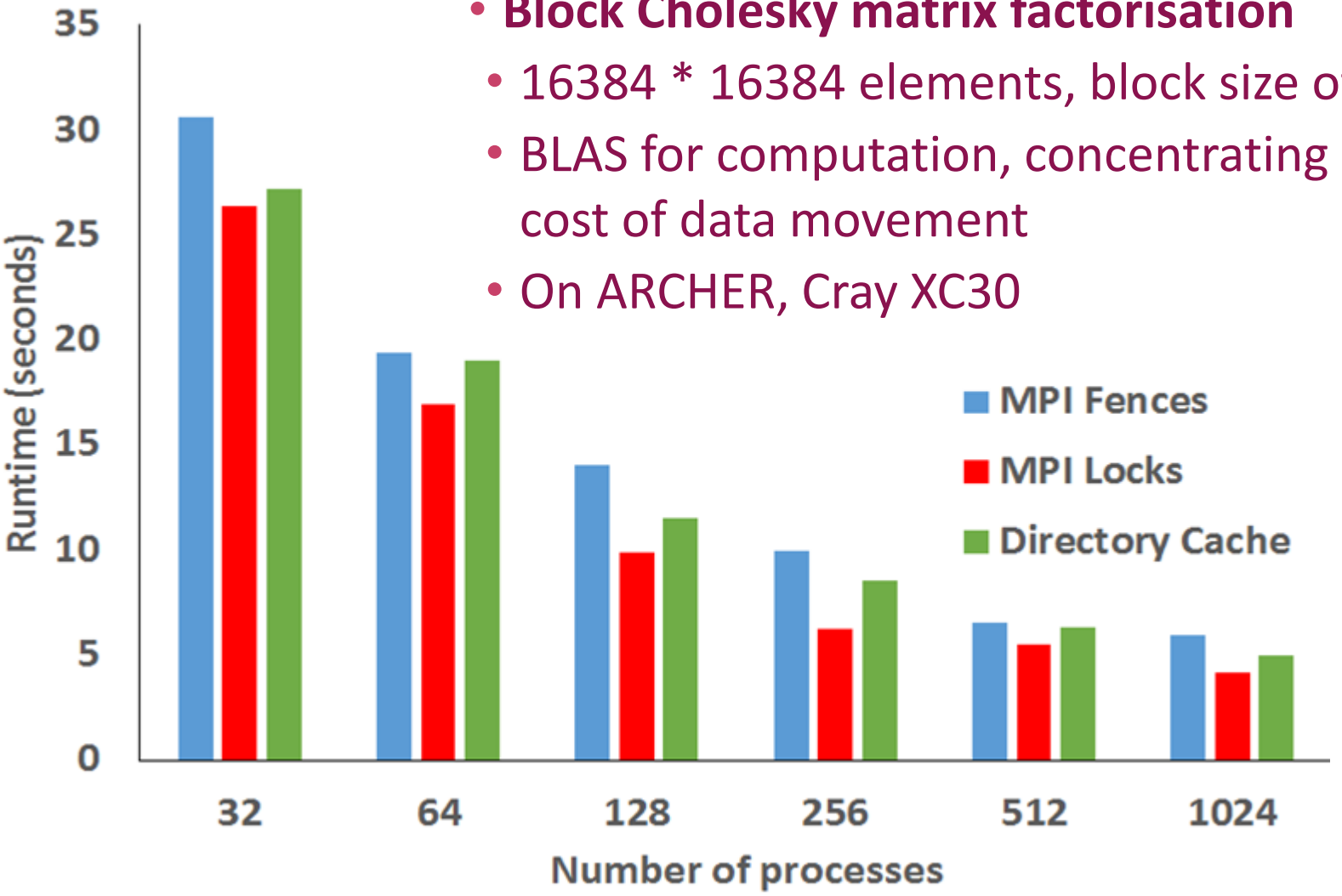
- **It is intended to be integrated into the runtimes of task based models (but can be used directly)**
  - Such as OmpSs, StarPU, PaRSEC, GPI-Space
  - So its use (and existence!) is transparent to the programmer



- **Transport layers are provided that implement the underlying data movement**
  - GASPI, MPI RMA, BeeGFS
  - Very easy to switch in and out

Early performance measures

- **Block Cholesky matrix factorisation**
  - 16384 * 16384 elements, block size of 16 * 16
  - BLAS for computation, concentrating here on the cost of data movement
  - On ARCHER, Cray XC30



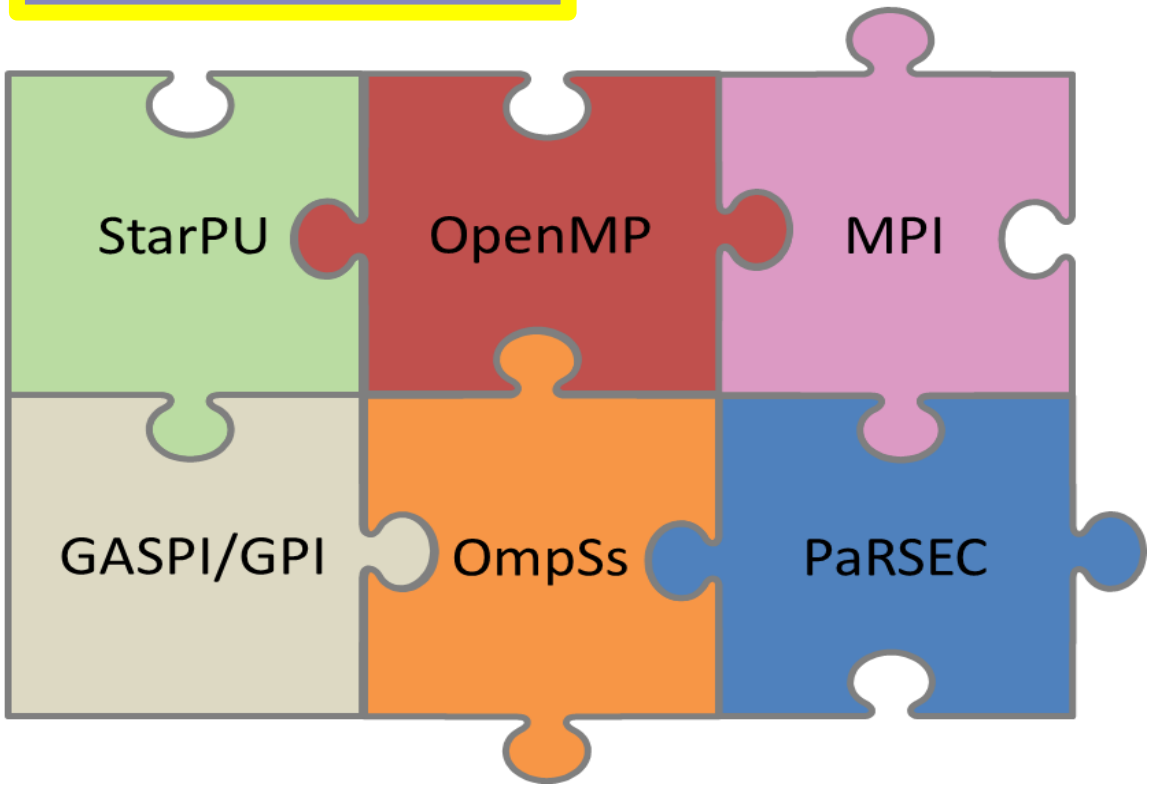
Linear Algebra

CFD

Space Plasma

Big Data analytics

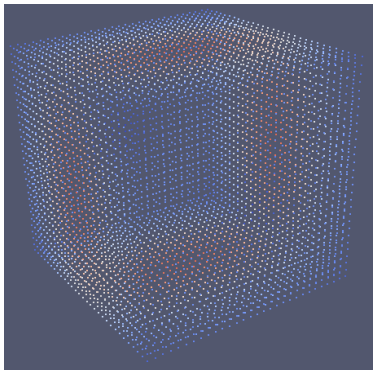
Complex Fluids



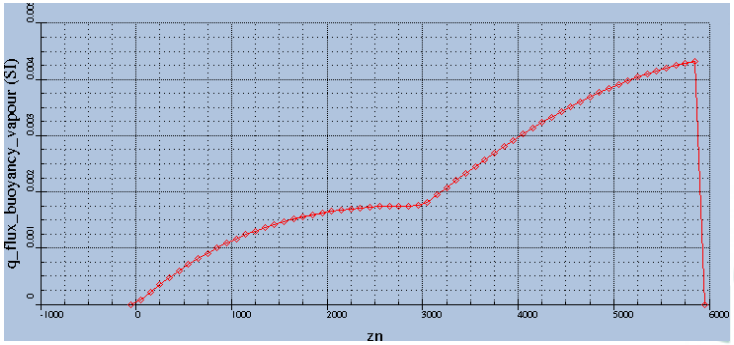
Big data analytics for atmospheric modelling

- Separately we have worked with the Met Office on a large scale atmospheric code

- The previous code was capable of maximum 20 million grid points over 256 cores
- We routinely run our model (MONC) on 2.1 billion grid points, 32768 cores



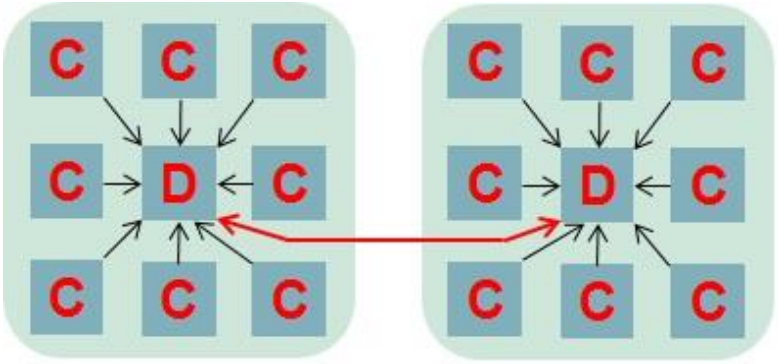
Prognostics



Diagnostics

Analysing the data in-situ

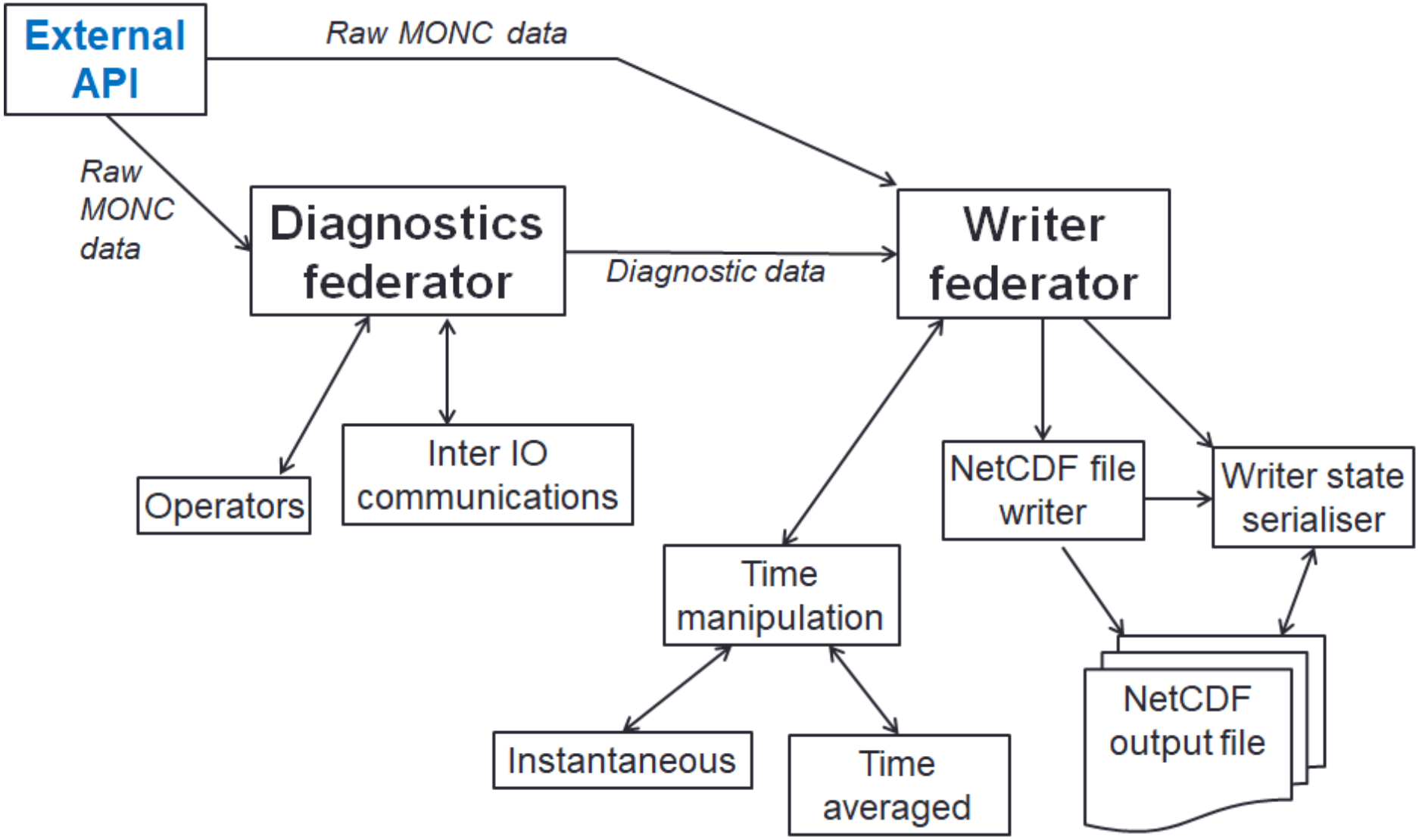
- **Have many computational processes and a number of data analytics cores**
  - Typically one core per processor is dedicated to IO, serving the other cores running the computational model
  - Computational cores “fire and forget” their data
- **In-situ as raw data is never written out**
  - Would be too time consuming



- **Avoids blocking the computational cores for analytics and IO**
- **Dynamic time-stepping, checkpoint-restart of analytics, bit reproducibility, easy configuration**

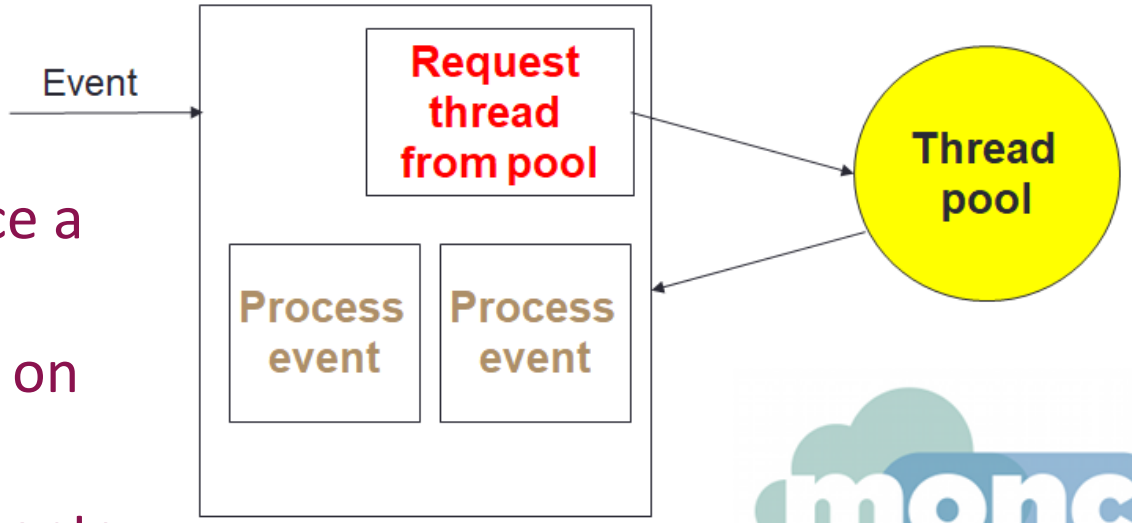
monc

Our analytics pipeline



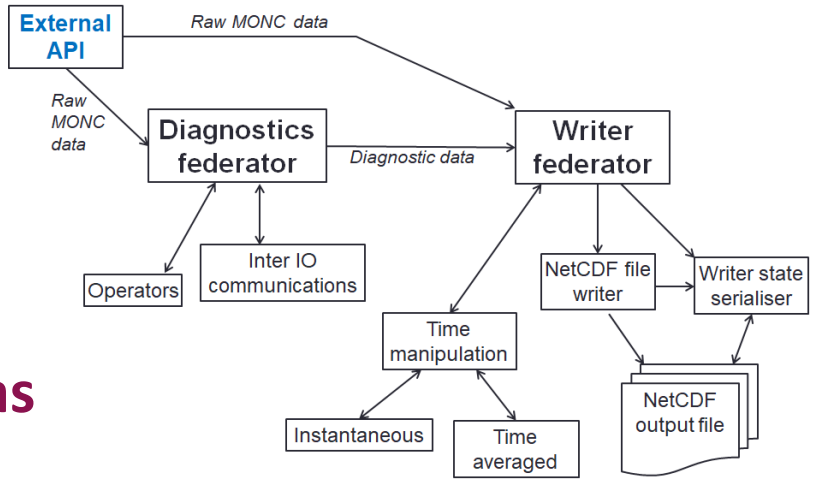
Under the hood: Event handling

- **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**

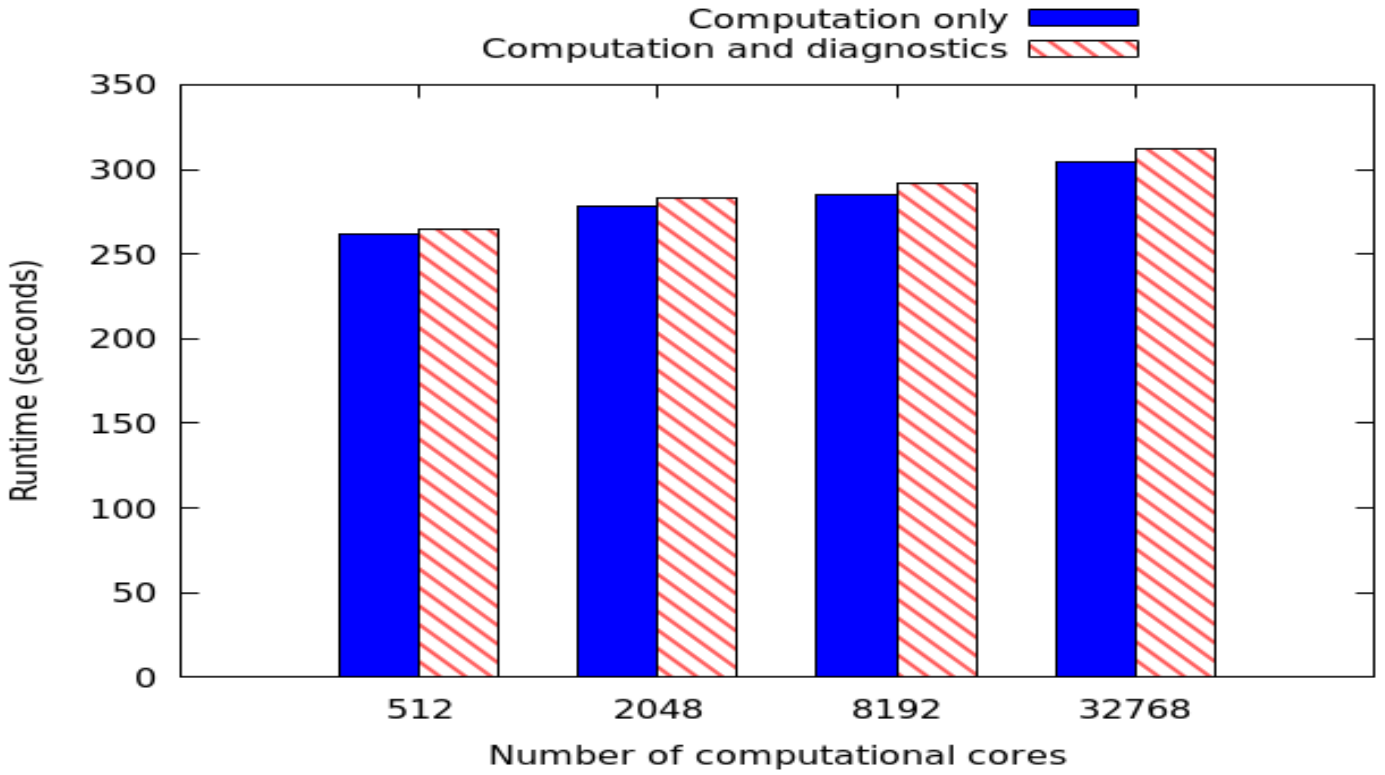
Active messaging for inter-IO communications

- **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, action, 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)

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


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

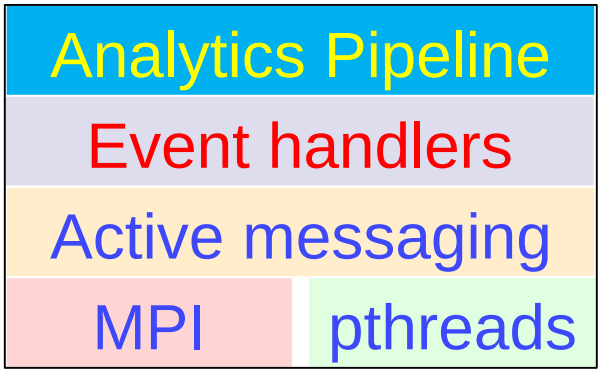
Performance and scalability

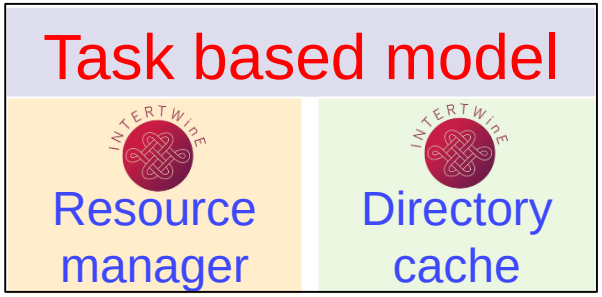


- **Standard MONC stratus cloud test case**
  - Weak scaling on Cray XC30, 65536 local grid points
  - 232 diagnostic values every timestep, time averaged over 10 model seconds. File written every 100 model seconds. Run terminates after 2000 model seconds.

What's the link to INTERTWinE?

- **This is (distributed) task-based parallelism**
  - Active messaging addresses the interoperability challenge between tasks (threads from the pool) and MPI communications
  - Our analytics pipeline is currently built on-top of our own lower layer which provides active messaging via MPI P2P





- **Instead, can we keep our abstraction of the analytics pipeline but build this upon existing task-based models (OmpSs, OpenMP etc)**
  - Build upon the resource manager and directory/cache of INTERTWinE
  - Ideally extend the notion of task dependencies
  - The work of INTERTWinE is key here

Summary and collaboration opportunities

- **Best Practice Guides:**
  - Writing GASPI-MPI Interoperable Programs
  - MPI + OpenMP Programming
  - MPI + OmpSs Interoperable Programs
  - Open MP/OmpSs/StarPU + Multi-threaded Libraries Interoperable Programs
- **“Developer Hub” of resources for developers & application users**
- **We are very happy to make available our directory/cache and resource manager implementations**
- **In-situ data analytics:**
  - Keen to explore generalising these ideas to wide variety of parallel codes generating lots of data