XIOS-3
Toward a new infrastructure of HPC services and model coupling

Y. Meurdesoif (IPSL - CEA/DRF/LSCE)

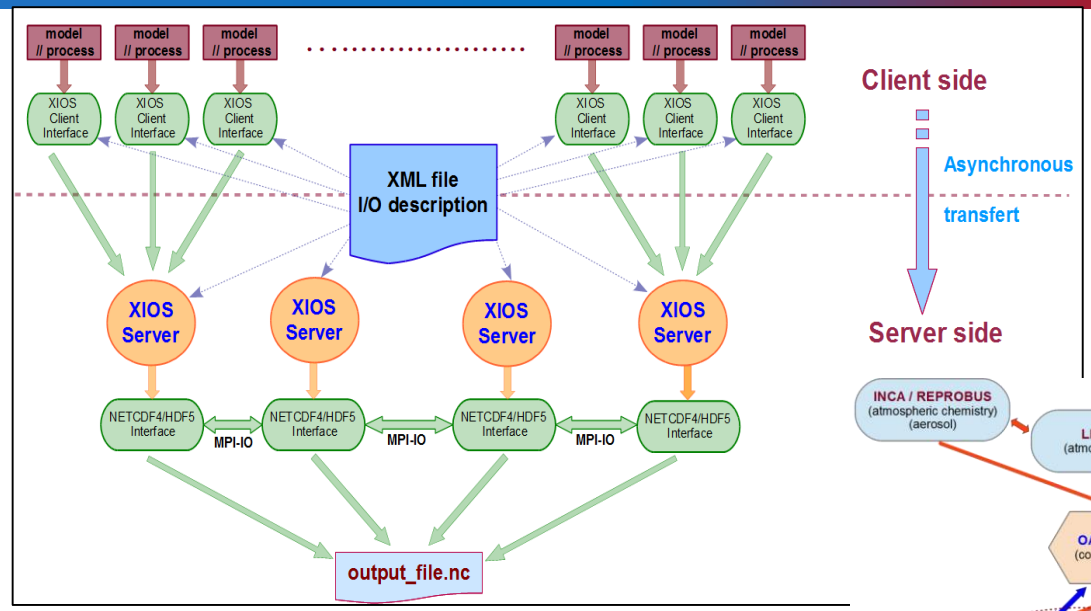
J. Dérouillat (IPSL - CEA/DRF/LSCE)

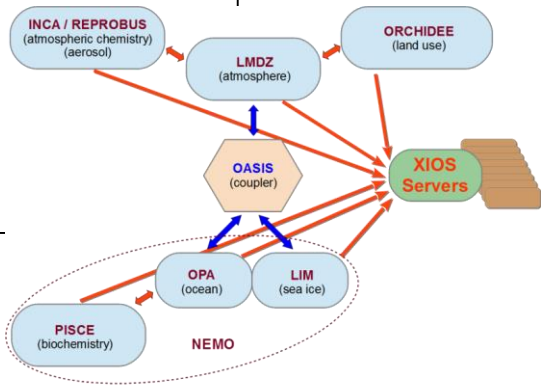
A. Caubel (IPSL - CEA/DRF/LSCE)

XIOS : some history about major evolution

XIOS-1 (2014)

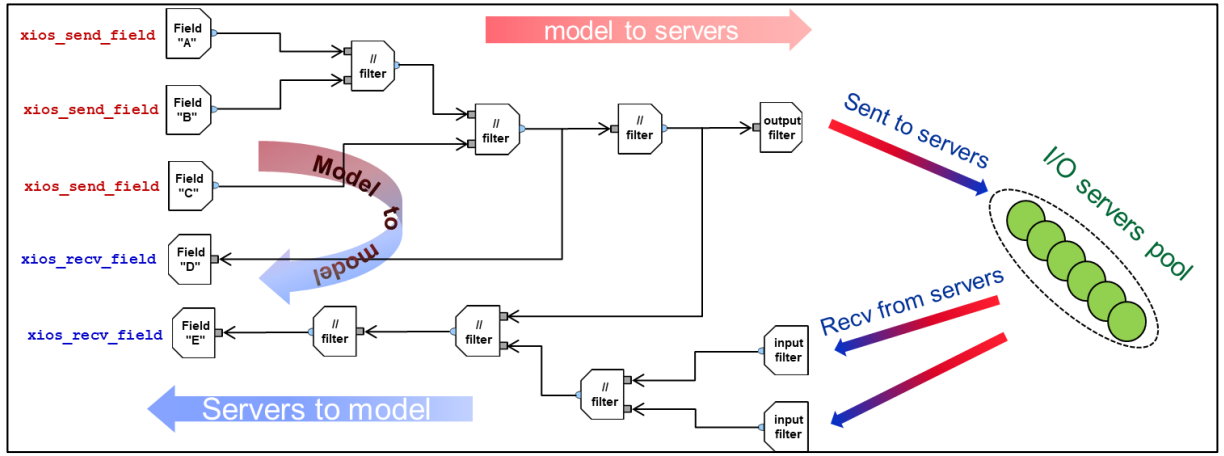
- I/O description outsourced of models in external XML files
  - Simple fortran interface : xios_send_field("id", field)
  - Compact and flexible XML description using hierarchical concept
- Asynchronous transfer to dedicated parallel I/O Servers
  - Overlap transfer and writing time by computation
- Parallel write using parallel file system capability
- Targeted for coupled models
  - Interfaced with OASIS





XIOS-2 (2017)

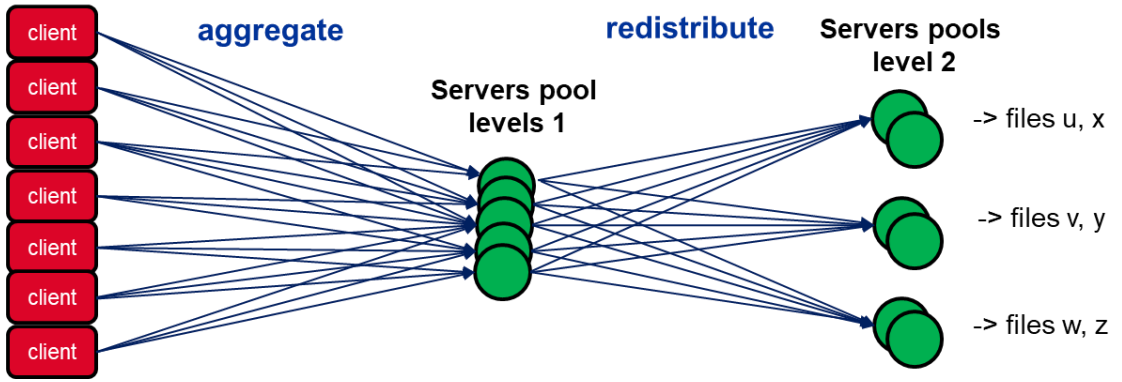
- Add asynchronous reading capability from servers
  - xios_recv_field("id", field)
- Add "in-situ" parallel workflow computing, developing filters for :
  - Time integration (instant, averaging, min, max…)
  - Arithmetic combination of fields
  - Spatial transformation
    - horizontal and vertical interpolation, sub-domain extraction, reductions, etc.
    - Interpolation : weight computation "on the fly"
- Complex workflow can be achieved by chaining filters before data flux are sent to servers or returned to model (reading)



XIOS : some history about major evolution

XIOS-2.5 (2018)

- Add second levels of servers in order to increase file writing concurrency between servers
  - Activating netcdf writing compression in parallel runs
  - Time series management



⇒ **Reference version for CMIP6 experiments**

- DR2XML (CNRM) : translate automatically CMIP6 data Request into xml xios files
- ~1000 of different variables generated for one CMIP6 deck
- All post-treatments done “in the fly”, automatically CMORized (IPSL and Météo-France/CNRM ESM)

XIOS-3 (end-2022) : total rewrite of the internal XIOS core engine

- 3 years work of intense developments, touching more than the half part of code lines
  - 514 file modified, 244 SVN commits, 60 000 code lines modified/added/deleted (over 110 000 of total code lines)

```
irene171 work~/XIOS3/src>svn diff  -r 1749 | diffstat -m -s
 514 files changed, 36824 insertions(+), 19441 deletions(-), 5209 modifications(!)
```

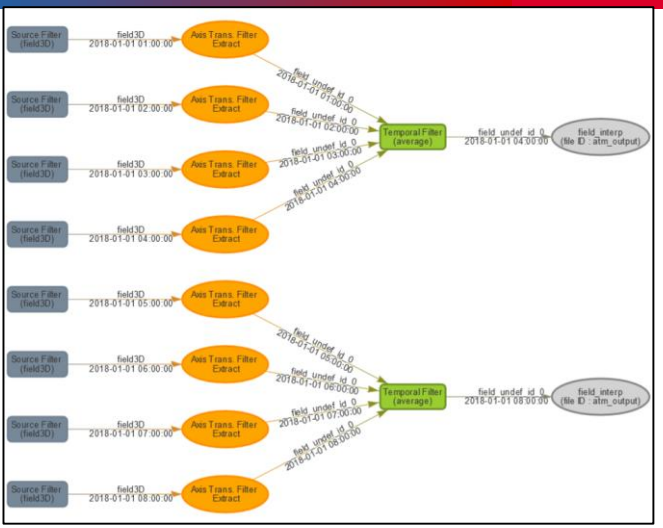
- Goals
  - Cleaning code and rationalizing internal concept due to years of eclectic development
  - Improvement of workflow performance and memory footprint reduction
  - Improvement of robustness and reliability
  - New infrastructure introducing XIOS HPC services concept and model coupling

XIOS-3 : improving robustness and reliability

- **Implementation of a non regression suite testcase for continuous integration**

Table of XIOS unit tests results

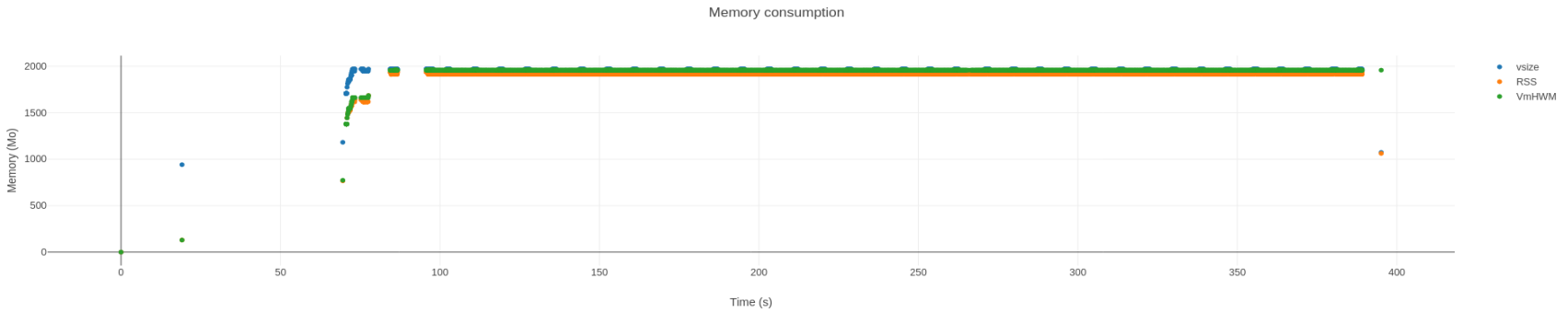
Revision	Jean-Zay	Irene
2384	X64_JEANZAY_PGI_prod X64_JEANZAY_prod	X64_IRENE_prod test_scalar_algo test_grid_algo CONFIG_AxisMask_false_DomMask_false_Dom_lmdz_Duration_1d_NDfTr_false_NbClnt_4_NbSrv_2_OneSided_false_PctSrv2_50_ScaMask_sparse_Srv2_false CONFIG_AxisMask_false_DomMask_false_Dom_lmdz_Duration_1d_NDfTr_false_NbClnt_4_NbSrv_2_OneSided_false_PctSrv2_50_ScaMask_none_Srv2_false test_dynamico_algo test_nemo_algo CONFIG_AxisMask_false_DomMask_true_Dom_nemo_Duration_1d_NDfTr_false_NbClnt_4_NbSrv_2_OneSided_false_PctSrv2_50_ScaMask_none_Srv2_false atm_output_reduce_axis.nc CONFIG_AxisMask_false_DomMask_false_Dom_nemo_Duration_1d_NDfTr_false_NbClnt_4_NbSrv_2_OneSided_false_PctSrv2_50_ScaMask_none_Srv2_false test_domain_algo test_function test_axis_algo test_memory X64_IRENE_GNU_prod



EXAMPLE OF THE XIOS WORKFLOW VISUALISATION.

- **Representation of XIOS workflow execution in the form of graphs, viewable through a web browser**

- **Tools to track internal memory usage and memory leak, time line visualization through web browser**



- **Help for debugging : output of the XIOS software stack in case of a crash, with relevant information**

- **Additional internal output timers at the end of the simulation for better performance profiling**

XIOS 3 : improving performance and memory footprint

- Development of new client/server transfer protocols based on passive one-sided MPI3 communication
- New concepts of 'views' and 'connectors' for distributed management of workflow grids
- Reduction of the memory footprint by applying tensor product properties onto elements (domains, axis, scalars) composing a grid
- Full rewrite of transformation engine
- Full rewrite of the chaining filters engine

⇒ **Increase the transfer protocol fluidity, performance improvement**

- Under evaluation : testcase : NEMO 4 configuration 1440 x 1680 x 75, 20000 timesteps, 2688 process, 80 XIOS servers, 2 levels of server, write every 50 ts
  - Whole time NEMO no IO (without initialization) Reference => 3051 s
  - Whole time NEMO (without initialization) XIOS 2 => 3462 s : XIOS overhead 411 s => 13% overhead
  - Whole time NEMO (without initialization) XIOS 3 => 3186 s : XIOS overhead 135 s => 4.4% overhead

⇒ Reduction of the XIOS overhead by a factor 3

⇒ 8% speed-up on this configuration

- Preliminary results, can be configuration dependant

3.5 Tb generated over 3000s => 1.2 Gb/s

⇒ **Memory footprint reduction**

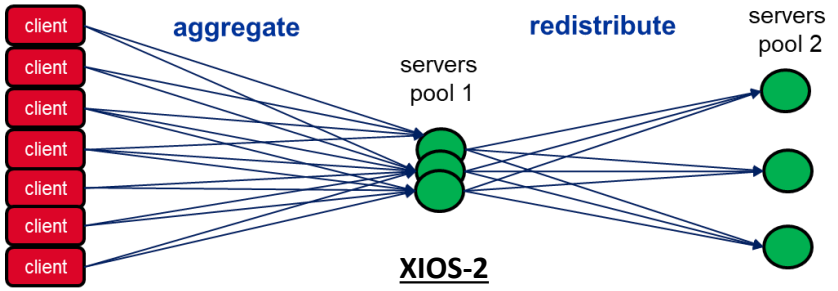
- Same NEMO configuration
- XIOS2 Vs XIOS3 Client+Model : reduction of 20% of whole virtual memory
- XIOS2 Vs XIOS3 Client part : reduction by a factor 3 of virtual memory consumption
- XIOS2 Vs XIOS3 Server side : reduction of virtual memory consumption up to a factor 3

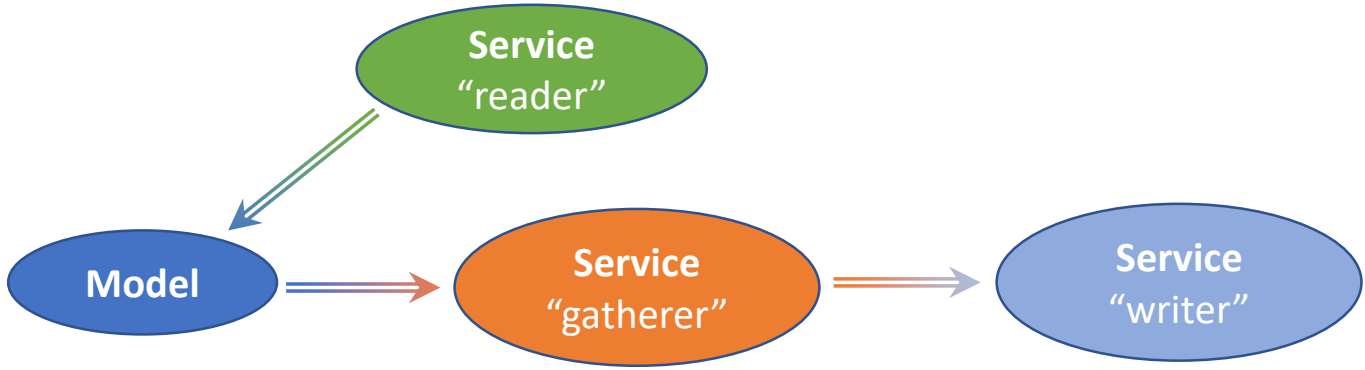
	XIOS2	XIOS3
Client + Model	150 Mo	120 Mo
Server N1	3.75 Go	2 Go
Server N2	30 Go	10 Go

What is an XIOS service ?

- **A parallel and asynchronous task running over a fraction of the dedicated pool of server processes**
  - XIOS schedules dynamically the launching of the required services in free resources
  - Interconnection between models and services are done through the XIOS middleware which provide mechanism for grid and data flux exchange
  - A model is saw by the XIOS middleware like a specific service which generate data periodically

- **XIOS-2 was providing IO services, but that was hardly coded**

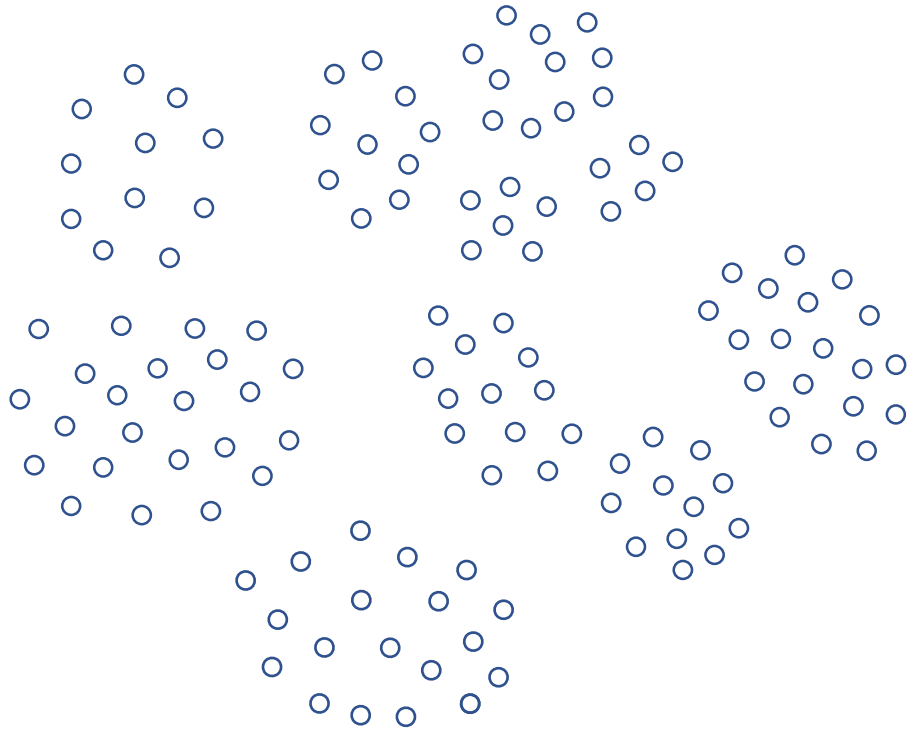




- **XIOS-3 rewrote XIOS-2 functionalities in term of interconnected services**
  - Rationalized way to exchange data flux through MPI partition
    - **model<->service, service<->service, model<->model**
    - **Enabling model coupling**
  - Description of services launching and models coupling remains described in a flexible way through external XLM files
  - Flexible management : services can run in separate resources or totally overlaping an other service resources

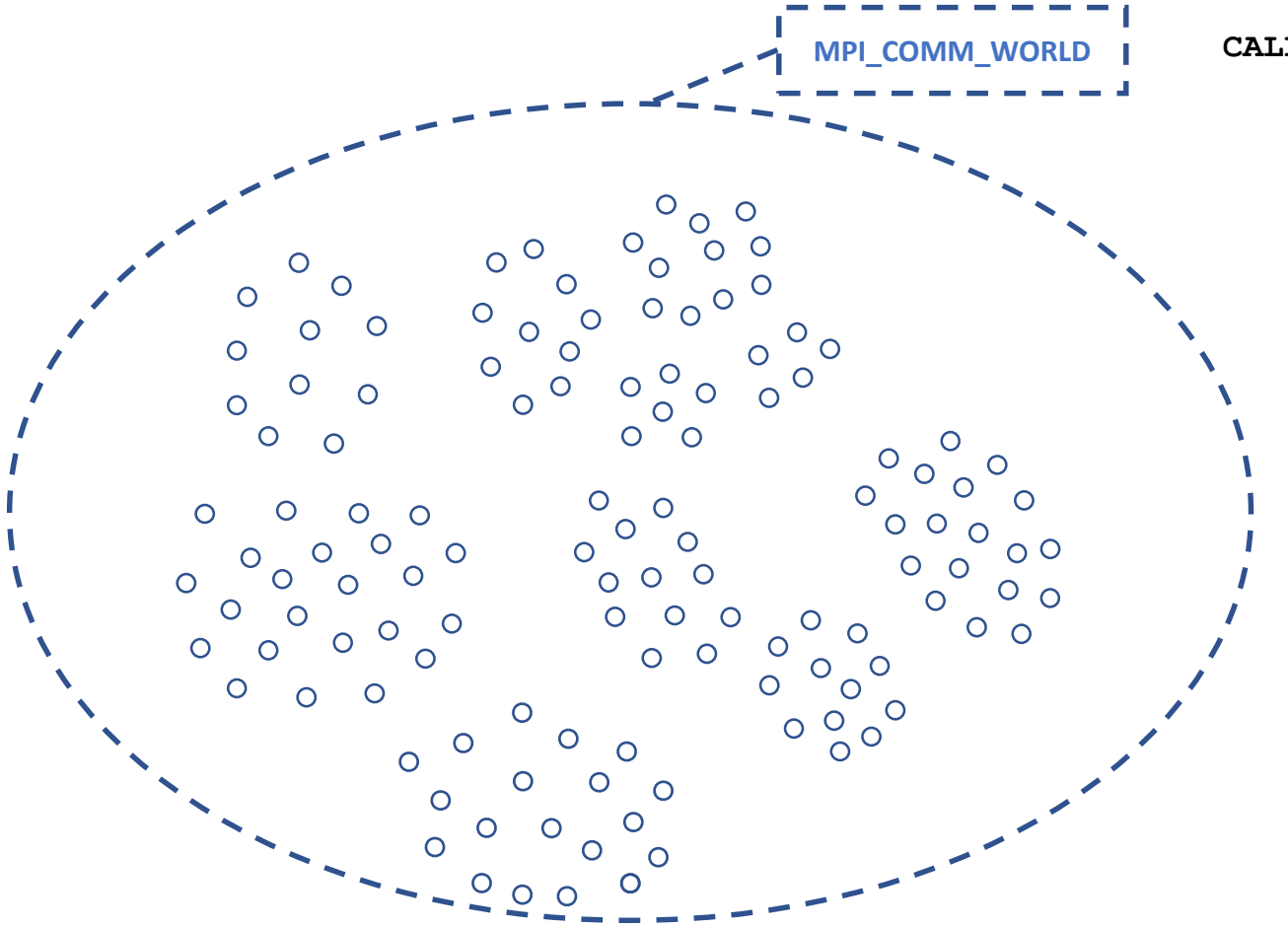
XIOS-3 services : how is it working ?

mpirun –np 11 atm : -np 23 ocean : -np 20 land : -np 76 xios_server.exe



XIOS-3 services : how is it working ?

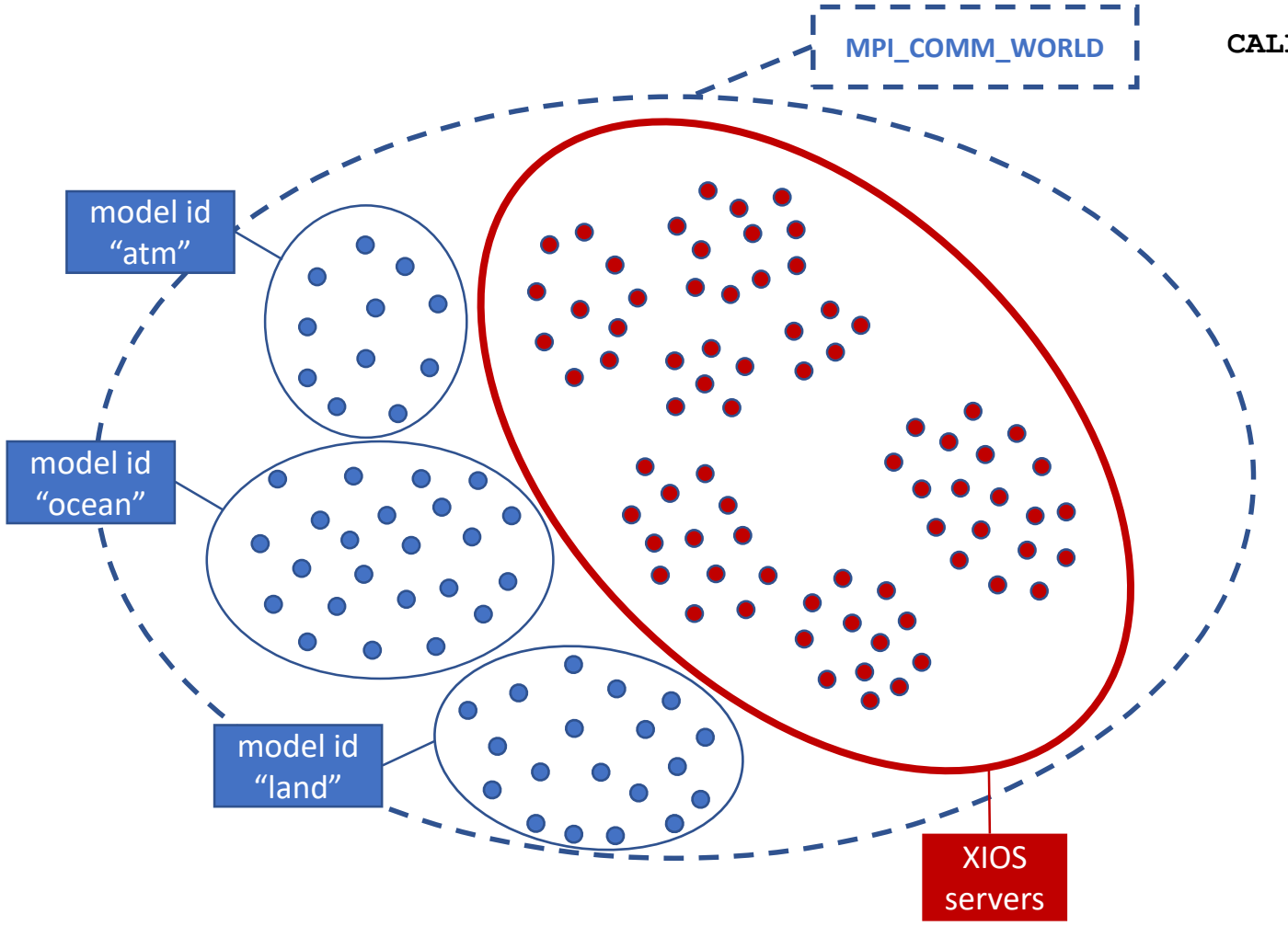
mpirun –np 11 atm : -np 23 ocean : -np 20 land : -np 76 xios_server.exe




```
CALL xios_initialize("model_id") => Communicator splitting
```

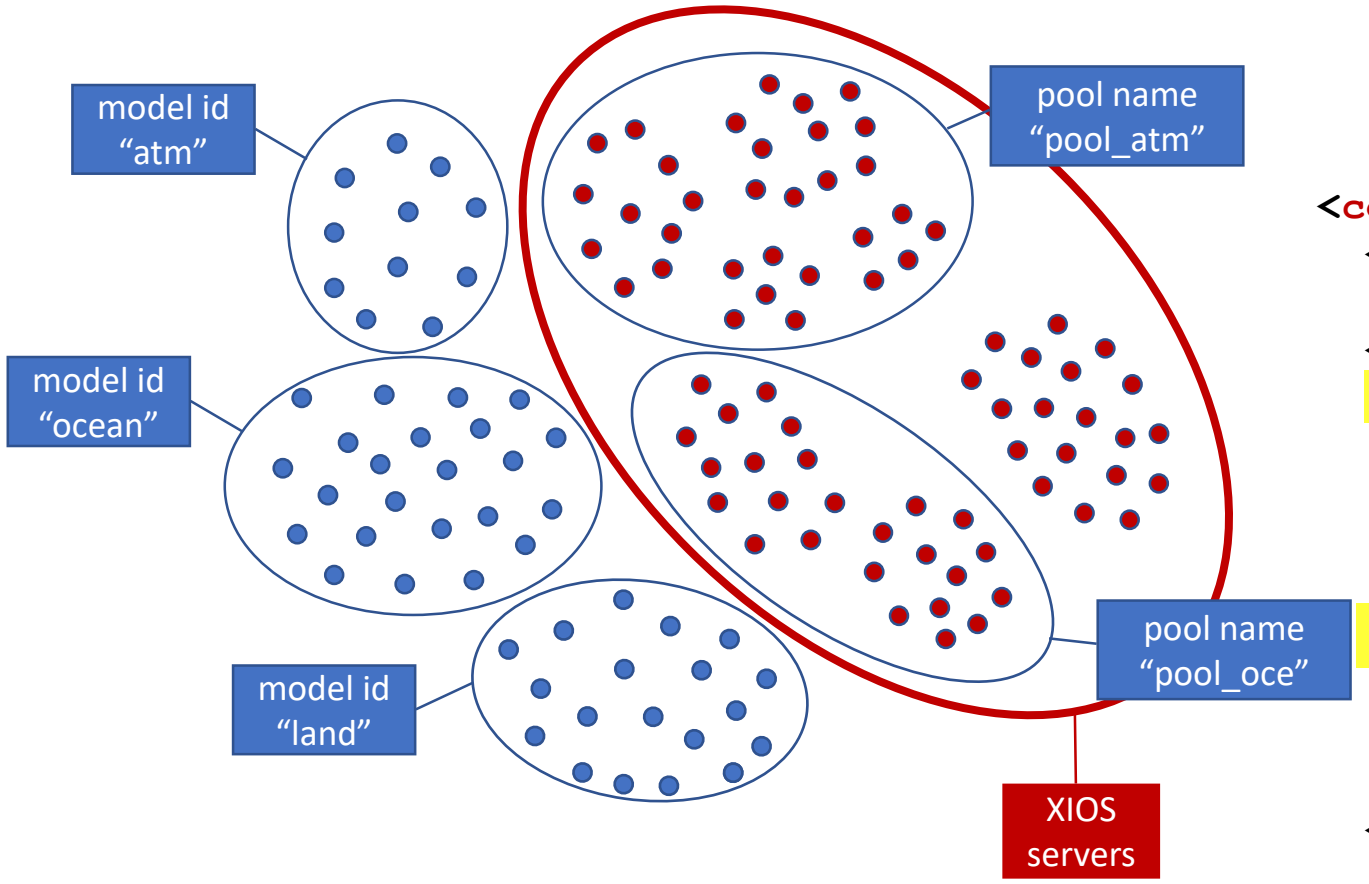
XIOS-3 services : how is it working ?

mpirun –np 11 atm : -np 23 ocean : -np 20 land : -np 76 xios_server.exe




```
CALL xios_initialize("model_id") => Communicator splitting
```

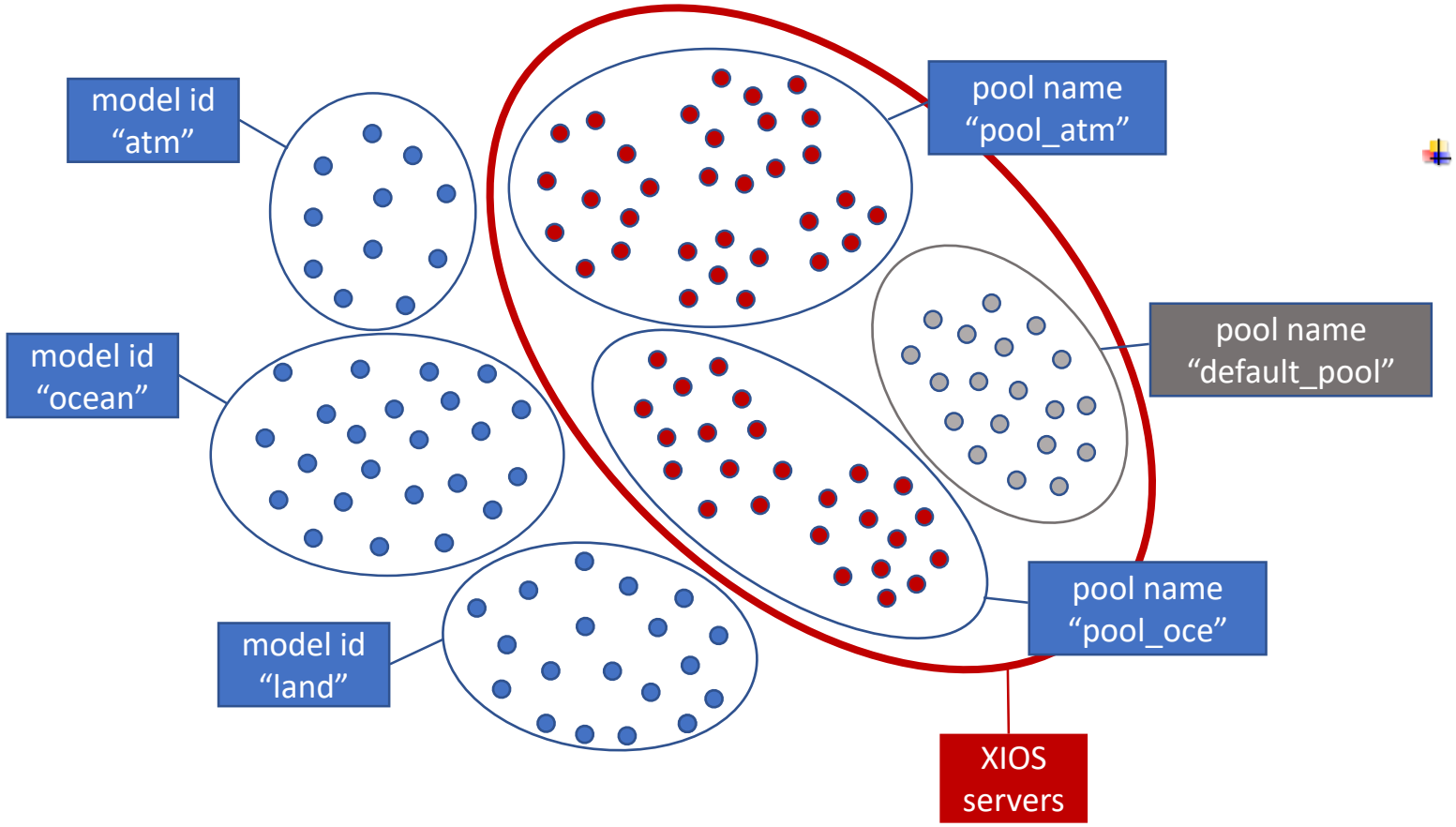
XIOS-3 services : how is it working ?



- **split servers into pools through XML definition**
  - **Allocated ressources defined by “pool” attributes**
    - **global_fraction (double) : fraction of whole server resources**
    - **remain_fraction (double) : fraction of remaining free servers**
    - **nprocs (int) : number of servers**
    - **remain (bool) : remaining free servers**

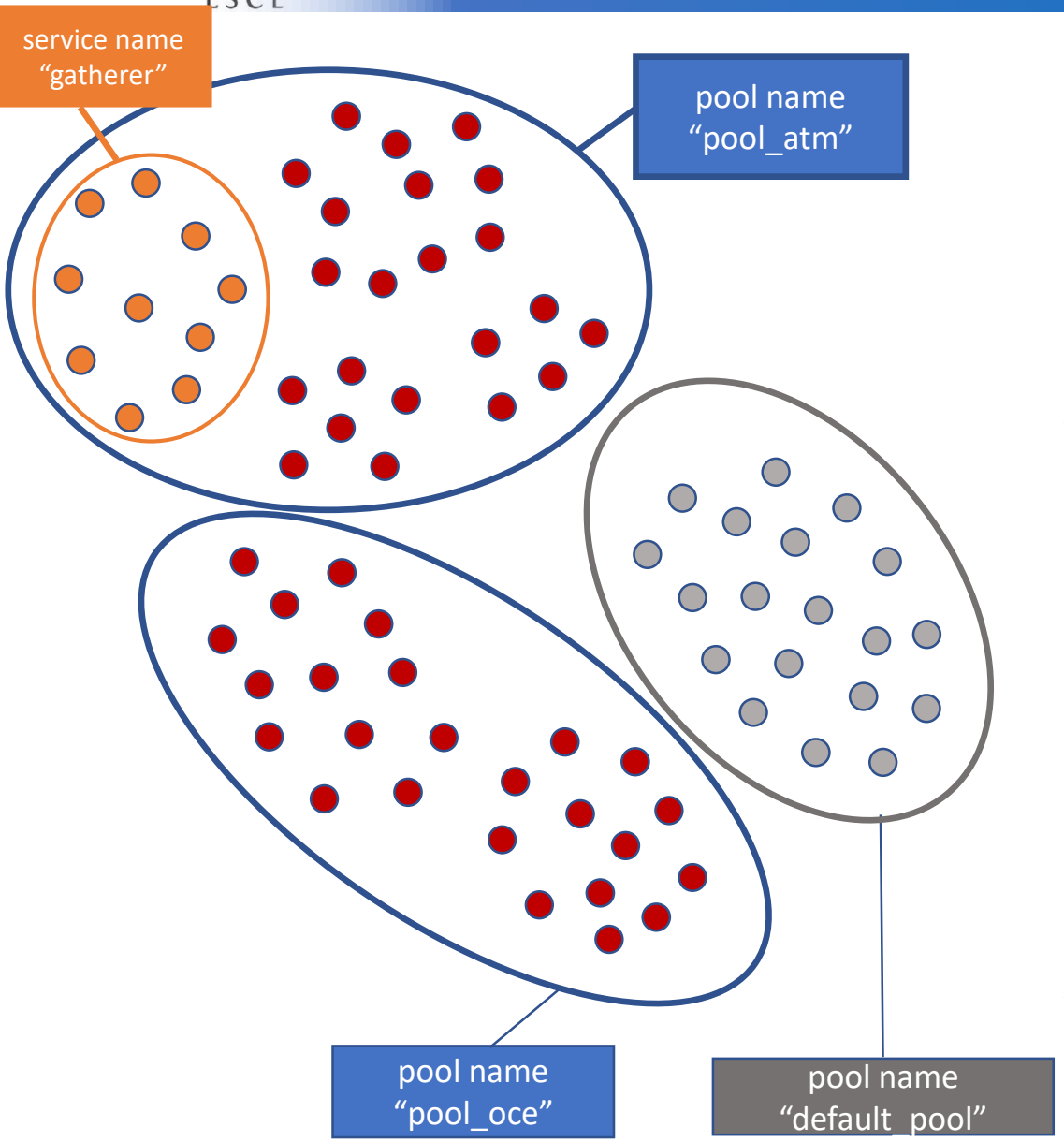
```
<context id="xios">
  <variable id="using_server2"> false </variable>

  <pool_definition>          => 76 free
    <pool name="pool_atm" global_fraction="0.25">

                             => 32 allocated,  remaining 44

    </pool>

    <pool name="pool_ocean" nprocs="25">

                             => 25 allocated,  remaining 19

    </pool>

  </pool_definition>

</context>
```

XIOS-3 services : how is it working ?



- **“default_pool” is created on remaining free resources**
  - **=> 19 servers**

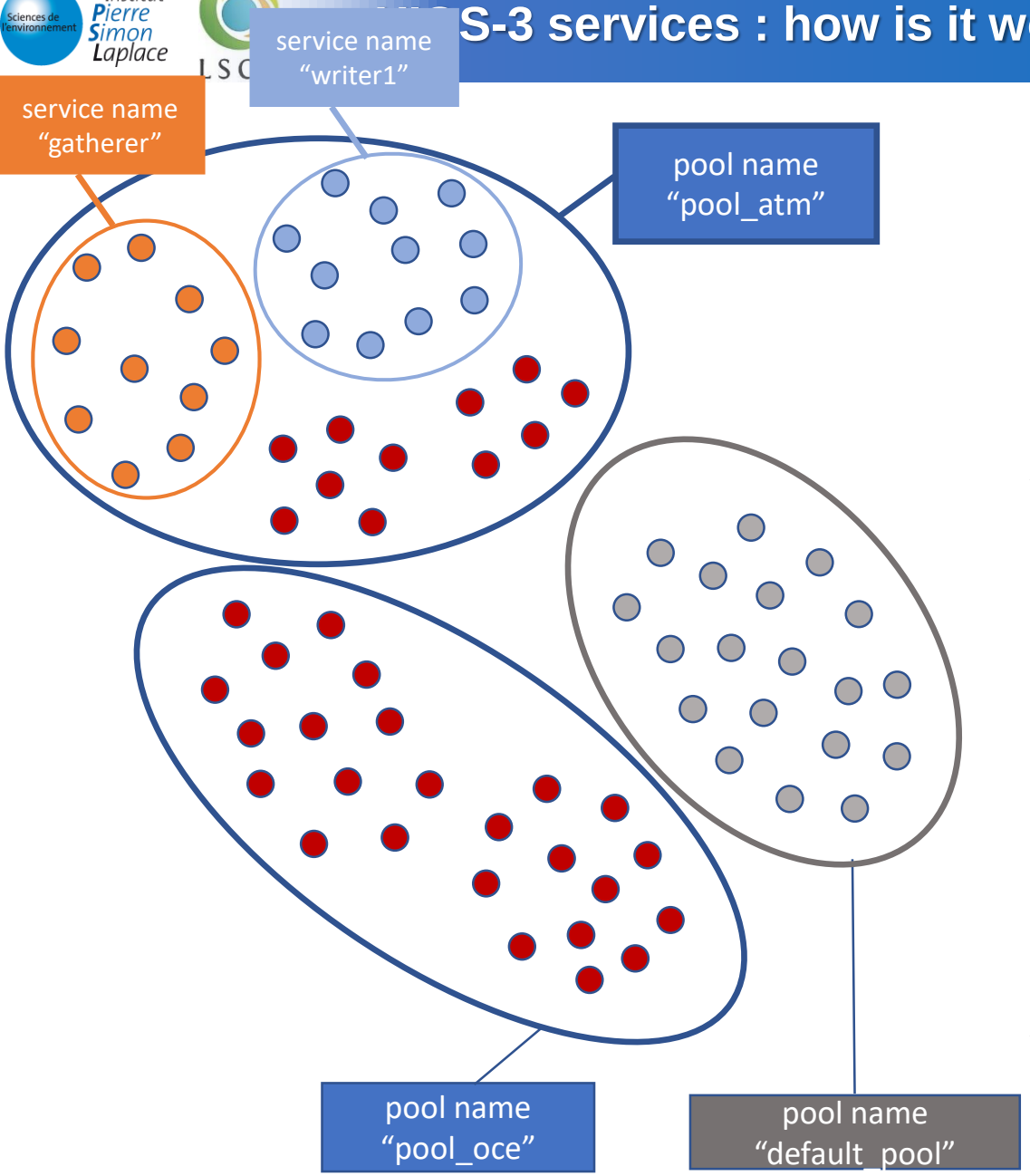
XIOS-3 services : how is it working ?



- **Launching services into allocated pools**
  - Same attributes that for allocating pools

```
<context id="xios">
  <variable id="using_server2"> false </variable>

  <pool_definition>
     <pool name="pool_atm" global_fraction="0.25">
       <service name="gatherer"  global_fraction="0.31" type="gatherer"/>


     </pool>
     <pool name="pool_ocean" nprocs="25">


     </pool>
    </pool_definition>
</context>
```

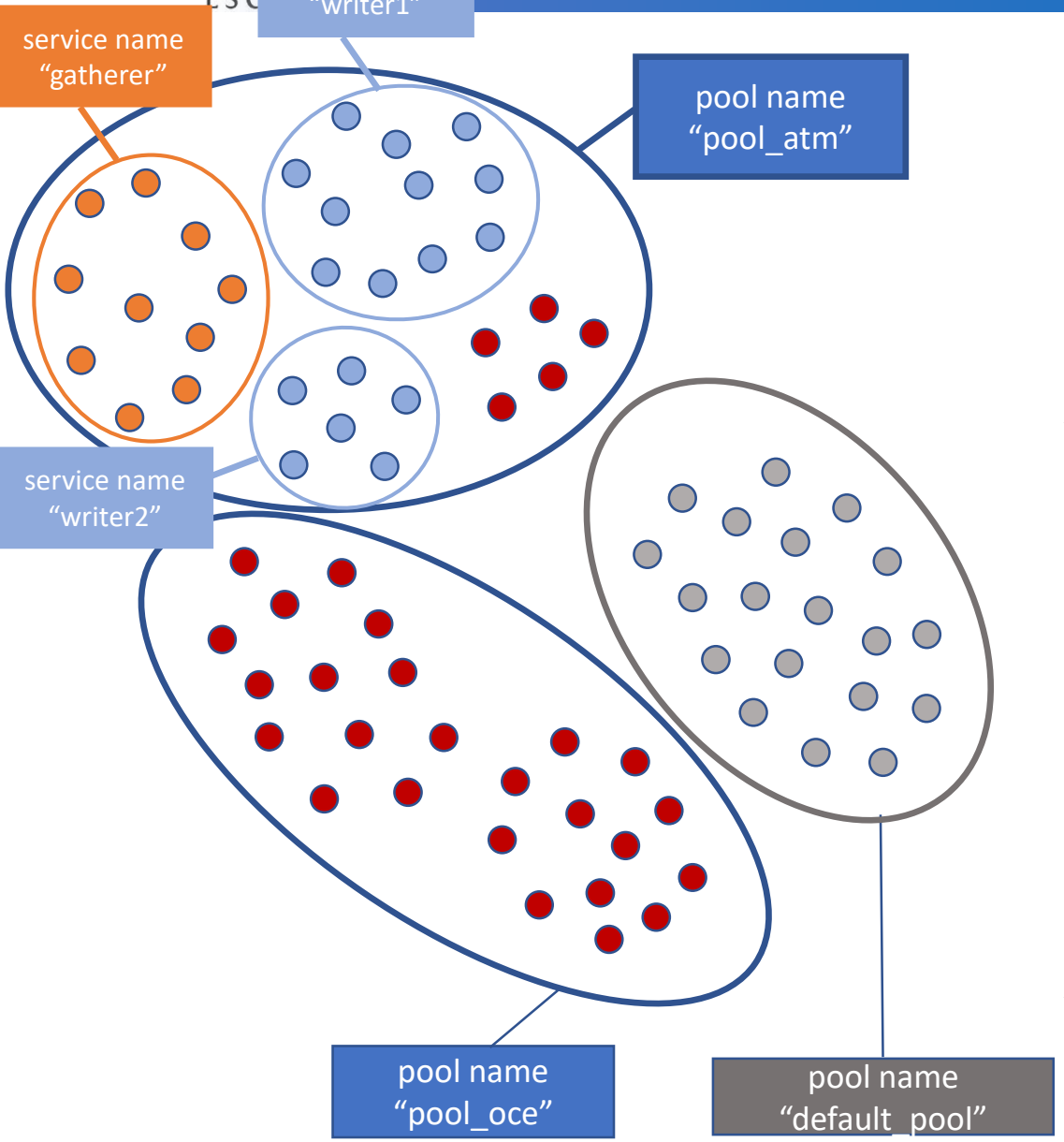
S-3 services : how is it working ?



- **Launching services into allocated pools**
  - Same attributes that for allocating pools

```
<context id="xios">
  <variable id="using_server2"> false </variable>

  <pool_definition>
     <pool name="pool_atm" global_fraction="0.25">
       <service name="gatherer"  global_fraction="0.31" type="gatherer"/>
       <service name="writer1"   remain_fraction="0.5" type="writer"/>


     </pool>
     <pool name="pool_ocean" nprocs="25">



     </pool>
    </pool_definition>
</context>
```

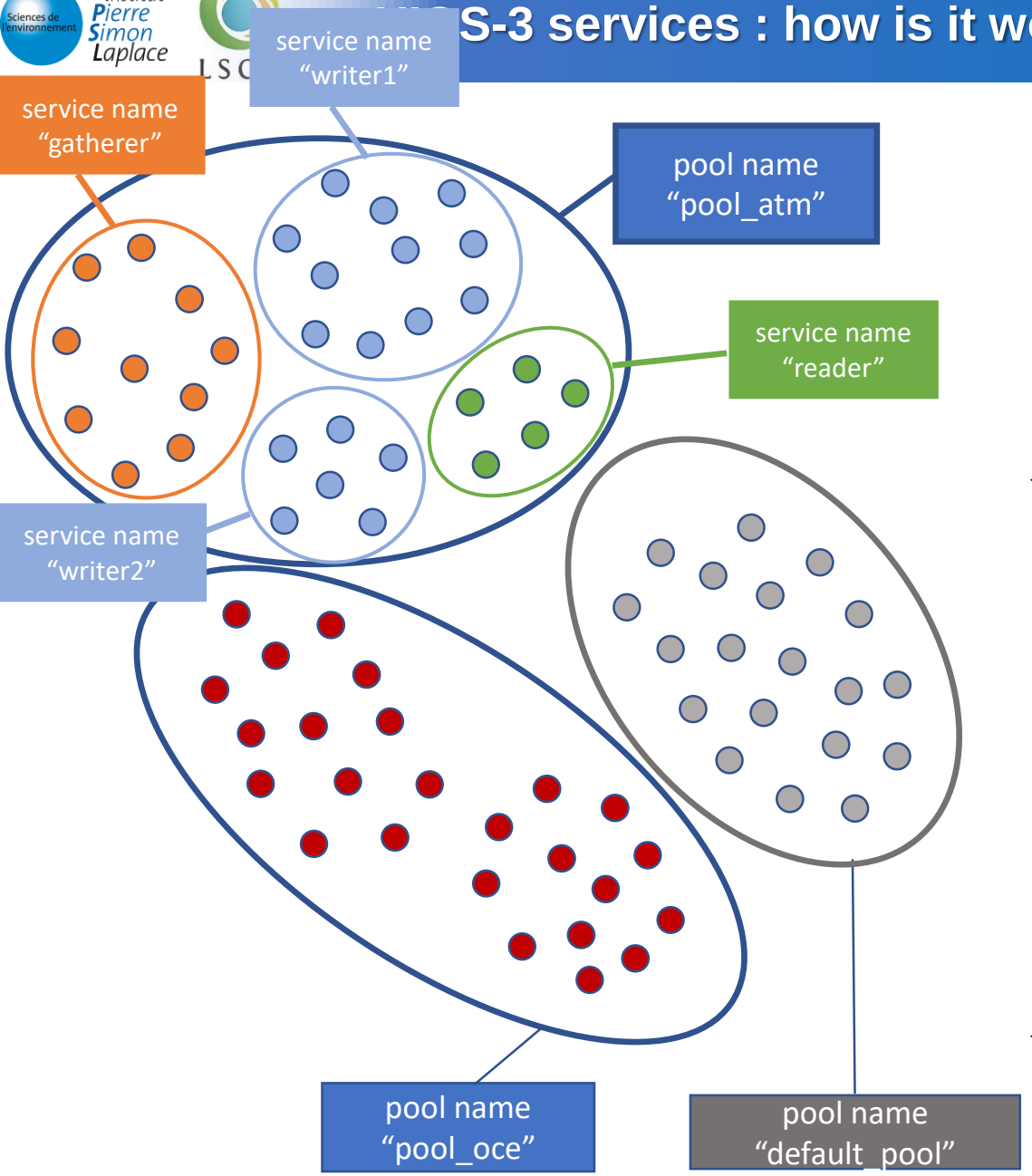
S-3 services : how is it working ?



- **Launching services into allocated pools**
  - Same attributing that for allocating pools

```
<context id="xios">
  <variable id="using_server2"> false </variable>

  <pool_definition>
     <pool name="pool_atm" global_fraction="0.25">
       <service name="gatherer"  global_fraction="0.31" type="gatherer"/>
       <service name="writer1"   remain_fraction="0.5" type="writer"/>
       <service name="writer2"   nprocs="6" type="writer"/>

     </pool>
     <pool name="pool_ocean" nprocs="25">



     </pool>
    </pool_definition>
</context>
```

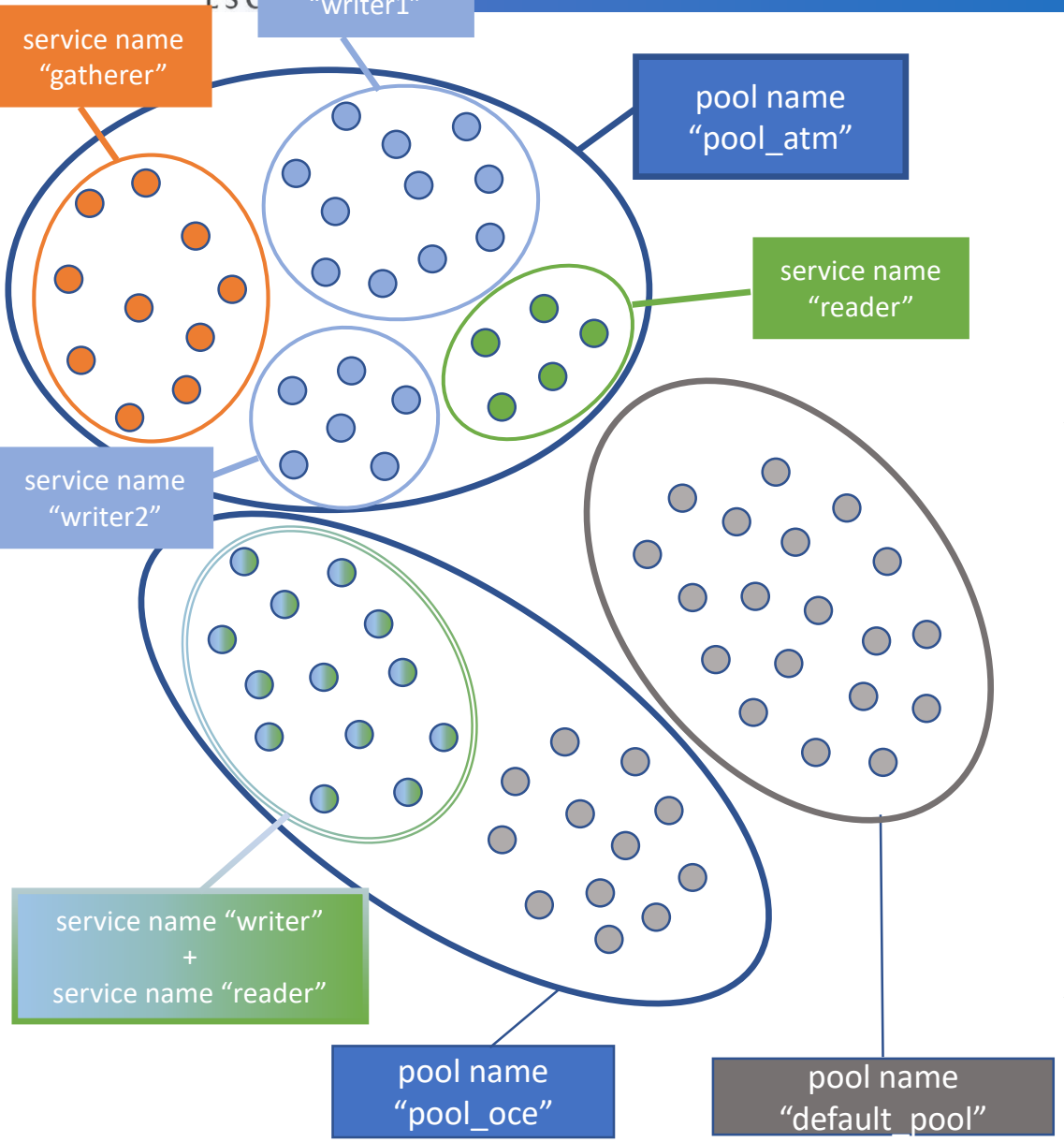
S-3 services : how is it working ?



- **Lauching services into allocated pools**
  - **Same attributes that for allocating pools**

```
<context id="xios">
  <variable id="using_server2"> false </variable>

  <pool_definition>
     <pool name="pool_atm" global_fraction="0.25">
       <service name="gatherer"  global_fraction="0.31" type="gatherer"/>
       <service name="writer1"   remain_fraction="0.5" type="writer"/>
       <service name="writer2"   nprocs="6" type="writer"/>
       <service name="reader"    remain="true" type="reader"/>
     </pool>
     <pool name="pool_ocean" nprocs="25">



     </pool>
    </pool_definition>
</context>
```

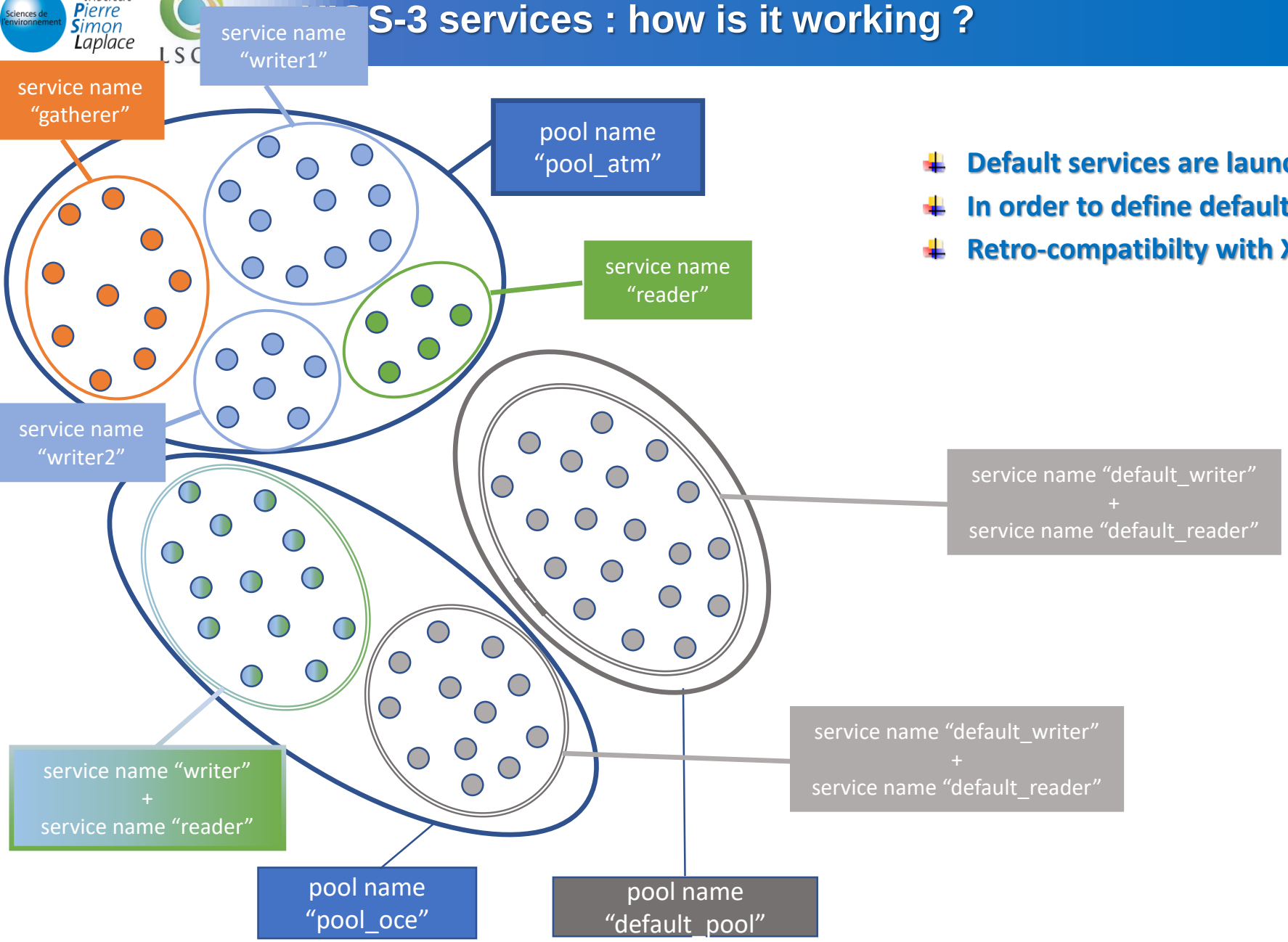
S-3 services : how is it working ?



- **Lauching services into allocated pools**
  - Same attributes that for allocating pools

```
<context id="xios">
  <variable id="using_server2"> false </variable>

  <pool_definition>
     <pool name="pool_atm" global_fraction="0.25">
       <service name="gatherer"  global_fraction="0.31" type="gatherer"/>
       <service name="writer1"   remain_fraction="0.5" type="writer"/>
       <service name="writer2"   nprocs="6" type="writer"/>
       <service name="reader"    remain="true" type="reader"/>
     </pool>
     <pool name="pool_ocean" nprocs="25">
       <service name="writer" nprocs="13" type="writer">
         <service name="reader"           type="reader"/>
       </service>
     </pool>
    </pool_definition>
</context>
```

S-3 services : how is it working ?



- **Default services are launched on unallocated servers**
- **In order to define default behaviour**
- **Retro-compatibilty with XIOS2**

Service association

- Targeted service is identified by pool name and service name => id = pool_name::service_name
- Can be assigned at context level => default behaviour
- Or Can be assigned at file level

```
<context id="atm" default_pool_writer="pool_atm" default_pool_writer="pool_atm" default_pool_reader="pool_atm" >
  <file_definition output_freq="1d">

    <file name="out1" mode="write" using_server2="true"  gatherer="gatherer" writer="writer1">
       <field field_ref="field_out1"/>
    </file>

    <file name="out2" mode="write" writer="writer2">
       <field field_ref="field_out2"/>
    </file>

    <file name="in" mode="read" reader="reader">
       <field field_ref="field_in"/>
    </file>

  </file_definition>
</context>
```

Sent to service **pool_atm::gatherer** chained to service **pool_atm::writer1**

Sent to service **pool_atm::writer2**

Received from service **pool_atm::reader**

- With no service specification, we find the XIOS2 behaviour

```
<context id="land" >
  <file_definition output_freq="1d">
     <file name="land_out" mode="write" / >
     <file name="land_in" mode="read" />
  </file_definition>
</context>
```

Sent to service **default_pool::default_writer**

Received service **default_pool::default_reader**

XIOS-3 services : future plans

New middleware infrastructure to manage I/O services in a flexible way

- Only gatherer, writer and reader services are currently implemented
- Main interest is for performance tuning
  - using dedicated services for models, aggregating more parallelism

Potentiality will be fully exploited with future development of new services, which can be interconnected with I/O services

Future plans are developing :

- Offload service : a piece of costly XIOS workflow diagnostic can be offloaded on dedicated resources
  - Short term
- Ensemble service : dedicated to efficient management of large ensemble runs
- AI services : deep learning training and inference could be done “In situ” and asynchronously
  - Making the bridge between the Fortran world of models and the Python world of deep learning technology
- User defined services
  - Users can write their own service for specific diagnostic
  - Could be written in Python to fully benefit of the software stack of python library

XIOS-3 : new coupling functionalities

New service infrastructure enable exchange of grid and data flux between different XIOS contexts

- Context can be attached to a service
- Context can be attached to a model
  - A model is saw like a service that produce specific data periodically

=> Exchange is now possible between 2 contexts running onto 2 different models

XIOS coupling time scheme

- Fields and associated grids are described as usually in XML context file
- Field to be sent from source to a destination context are imbedded into "coupler_out" elements

```
<coupler_out context="ocean::ocean_context">
  <field id="field3D_oce" field_ref="field3D"/>
</coupler_out>
```

Targeted context = dest_model_id::dest_context_id

- Field to be received are embedded into "coupler_in" elements

```
<coupler_in context="atm::atm_context">
  <field id="field3D_atm" grid_ref="grid_atm"/>
</coupler_in>
```

Source context = src_model_id::src_context_id

- At context initialisation (close_context_definition) grid are sent and redistributed from source to destination context
- Remapping can be achieved by chaining existing transformation filters (horizontal and vertical remapping)
- In time loop, coupling fields can be sent and received from/into models using the standard Fortran interface
  - `CALL xios_send_field("field_out_id", field_out)`
  - `CALL xios_recv_field("field_in_id",field_in)`

XIOS-3 Hello world coupling : one way coupling, data exchange at each timestep

Source model id =“atm”

```
<context id="atm_context">

  <coupler_out_definition>
    <coupler_out context="ocean::ocean_context" >
      <field id="field_atm" field_ref="field3d" />
    </coupler_out>
  </coupler_out_definition>

  <field_definition>
    <field id="field3d" grid_ref="grid3d" />
  <field_definition/>

  <grid_definition>
    <grid id="grid3d">
      <domain id="domain"/>
      <axis   id="axis"/>
    </grid>
  </grid_definition>

</context>
```

Grid transfer

Target model id =“ocean”

```
<context id="ocean_context">

  <coupler_in_definition>
    <coupler_in context="atm::atm_context" >
      <field id="field_atm" grid_ref="grid3d_atm"/>
    </coupler_in>
  </coupler_in_definition>

  <grid_definition>
    <grid id="grid3d_atm">
      <domain id="domain_atm"/>
      <axis   id="axis_atm"/>
    </grid>

    <grid id="grid3d_oce">
      <domain id="domain_oce">
        <interpolate_domain order="2"/>
      </domain>
      <axis id="axis_oce">
        <interpolate_axis>
      </axis>
    </grid>
  </grid_definition>

  <field_definition>
    <field id="field3d_oce" field_ref="field_atm"
                            grid_ref="grid3d_oce" />
  <field_definition/>

</context>
```

Remapping

CALL xios_send_field(“field3d”,field)

Field_atm → (Asynchronous transfer) → Field_atm → (horizontal interpolation) → (vertical interpolation) → Field_oce → **CALL xios_recv_field(“field3d_oce”,field)**

Coupling : conclusions

More complex configurations can easily be achieved by combining more of the XIOS workflow functionalities

- 2 way coupling
- Coupling at different time step
- Exchanging averaged or cumulated fields…

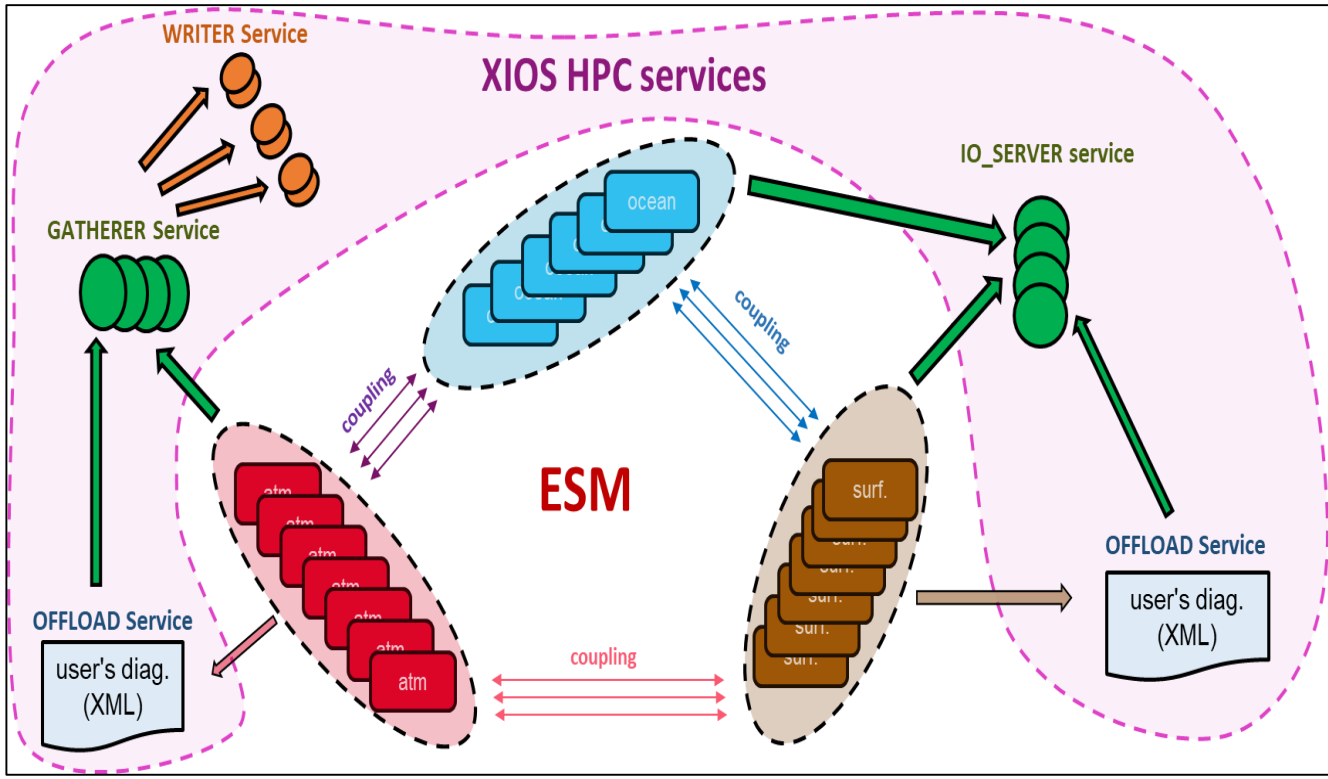
Some works is remaining to have a stable and efficient coupler

- Couple from previous times step
- Restartability
- More interpolation methods
  - Currently first and second order conservative
- Dead-lock hunting

Vision of future : a multitude of model components and services fully interconnected through an single middleware

- Simple minimalist Fortran interface
- Flexible management
- Asynchronous data exchange through the MPI partition to exhibit more parallelism and concurrency
- Light weight coupling written in Python or Fortran

  ⇒ User defined service



Future plans

- **Stabilization and consolidation of the services and coupling functionalities**
  - Must be implemented and tested on a full ESM model (IPSL-ESM)
- **Development of new kinds of service**
  - Offload, ensemble, AI…
- **Development of a Python interface**
  - User defined services
- **Revisit the XIOS timeline management**
  - Time interpolations
  - Adaptative time step
  - Make XIOS restartable
- **GPU porting, accelerators**
  - Will be the main priority for the next years
  - Be easier by new recoding
  - CPU consumption in time loop is now localized in small fractions of code : connectors and filters
  - Incremental approach, filters after filters…
  - Not decided which technology to use : language based directive (OpenAcc, OpenMP), kokkos or others..