Using HDF5 in WRF

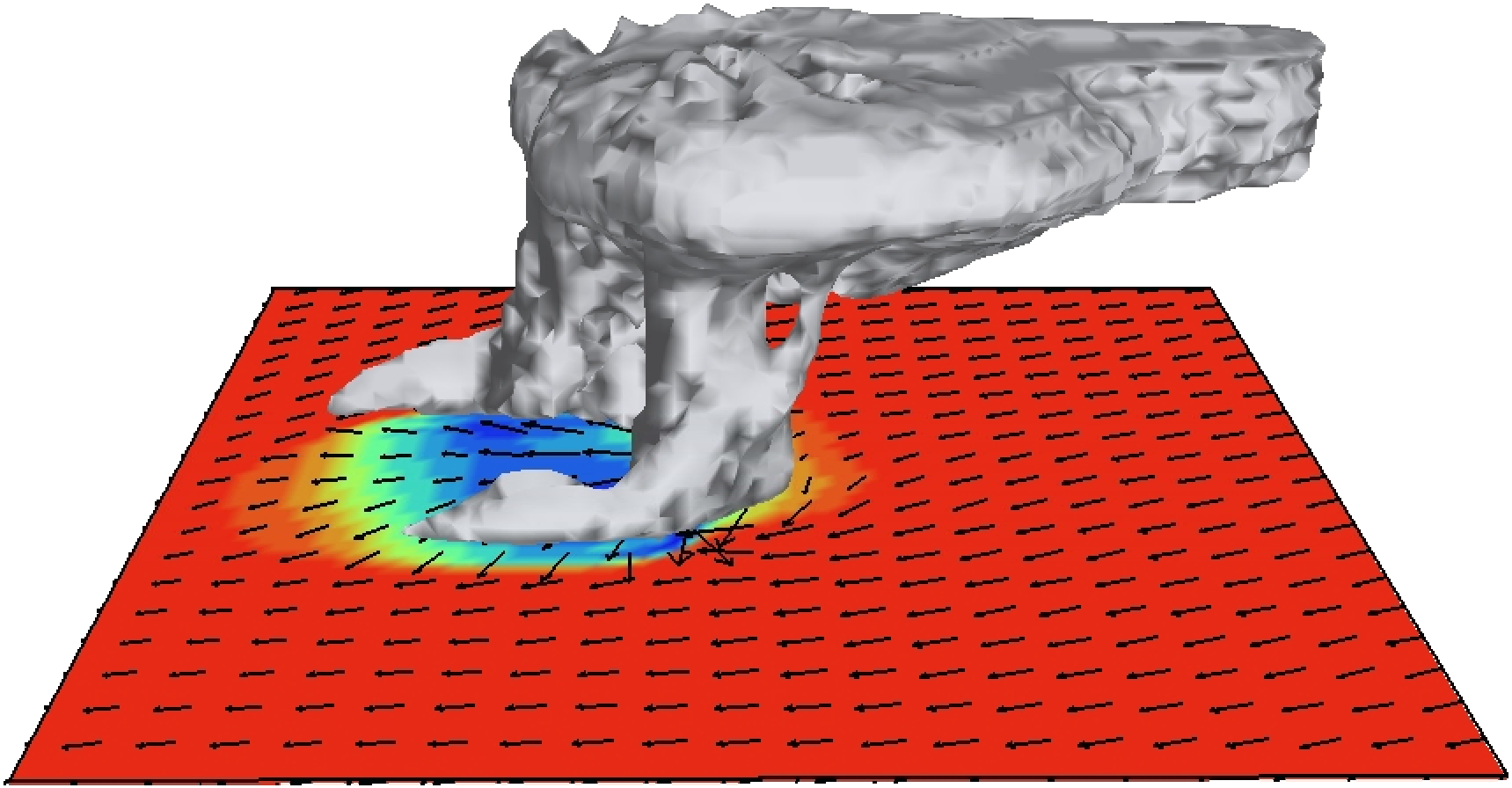
Part of MEAD - an alliance expedition

Contents

- Introduction to WRF
- Introduction to parallelism of WRF
- Introduction to WRF IO APIs
- WRF-HDF5 file structure
- WRF-HDF5 sequential mode
- WRF-HDF5 parallel design and potential problems

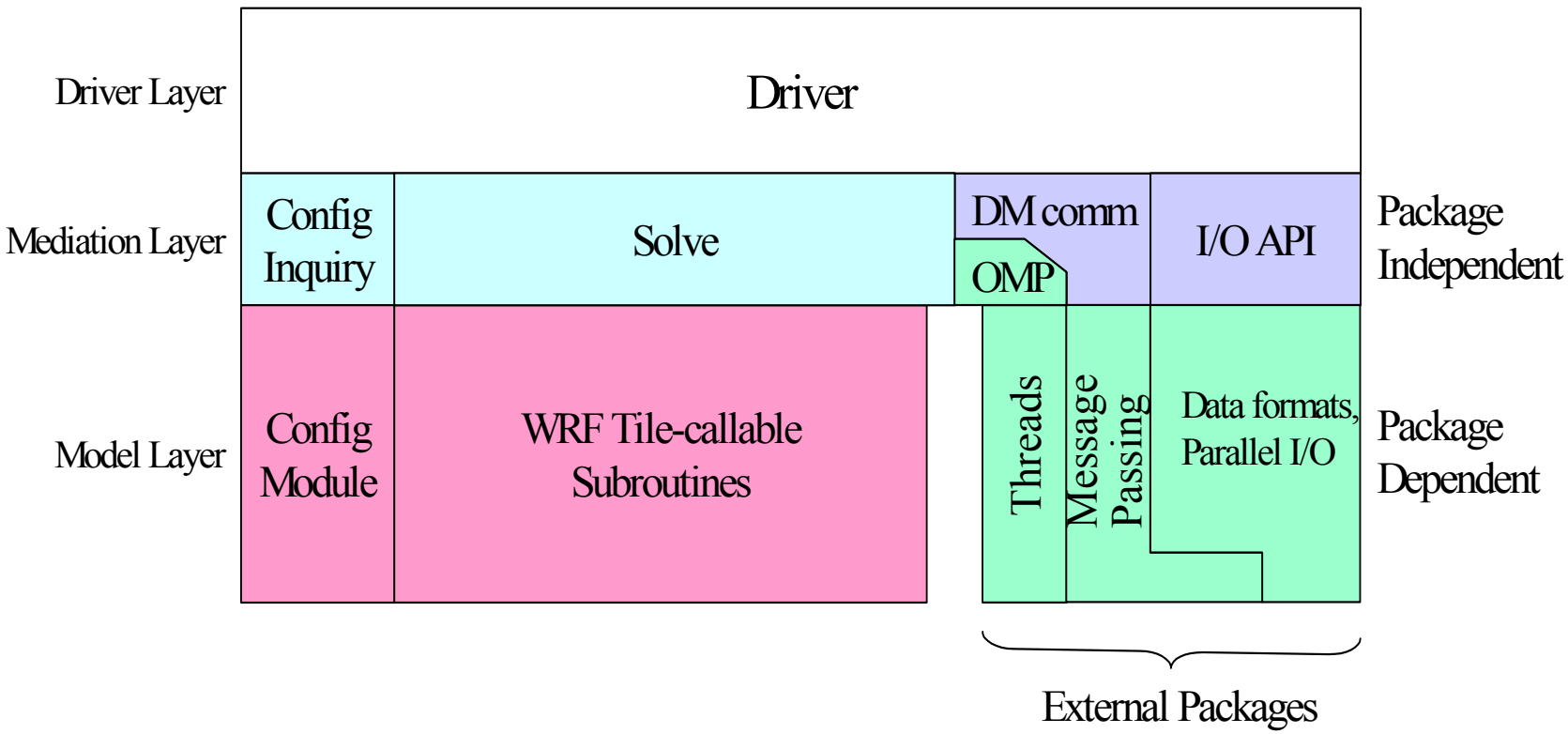
WRF(Weather Research Forecasting Model)

- A limited-area weather model for research and prediction
- Combine NCAR/PSU MM5(research model, dynamics) with NCEP ETA(operational model, physics)
- Have the tendency to become the most popular limited-area weather model in this country and in the world
- Many Potential users
- Most codes in Fortran 77 and Fortran 90



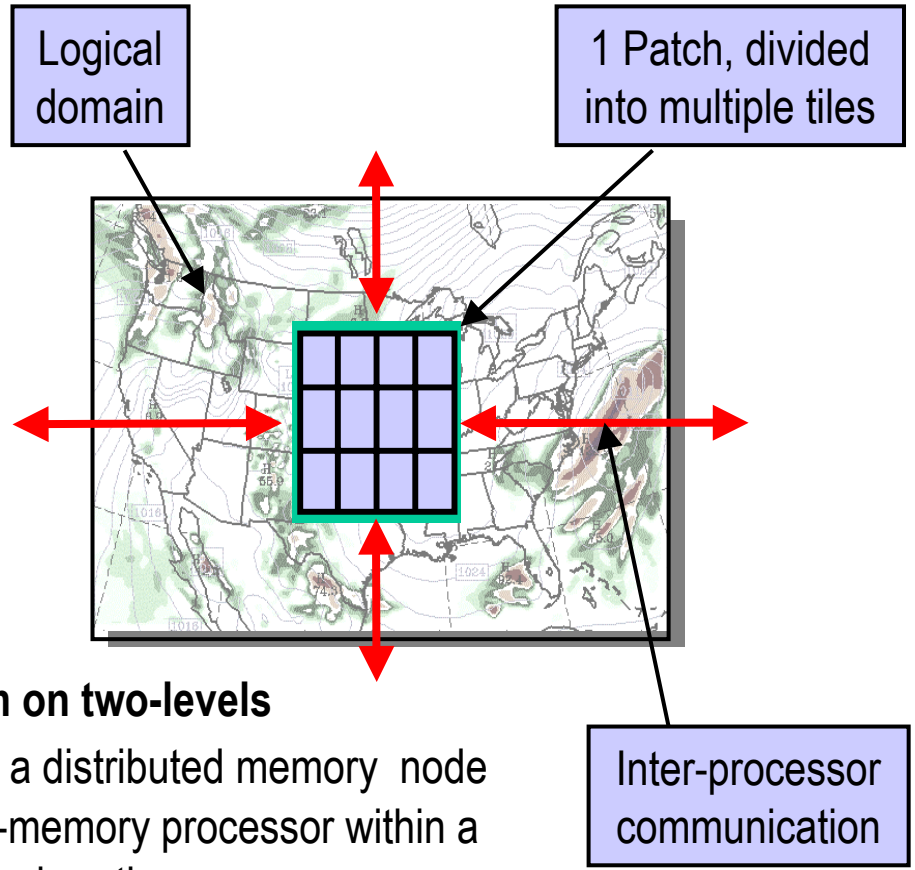
From WRF tutorial

WRF software design schematic



(Adapted from WRF software design document at
http://www.mmm.ucar.edu/wrf/users/WRF_arch_03.doc)

WRF Multi-Layer Domain Decomposition



- **Single version of code for efficient execution on:**
  - Distributed-memory
  - Shared-memory
  - Clusters of SMPs
  - Vector and microprocessors

Model domains are decomposed for parallelism on two-levels

Patch: section of model domain allocated to a distributed memory node

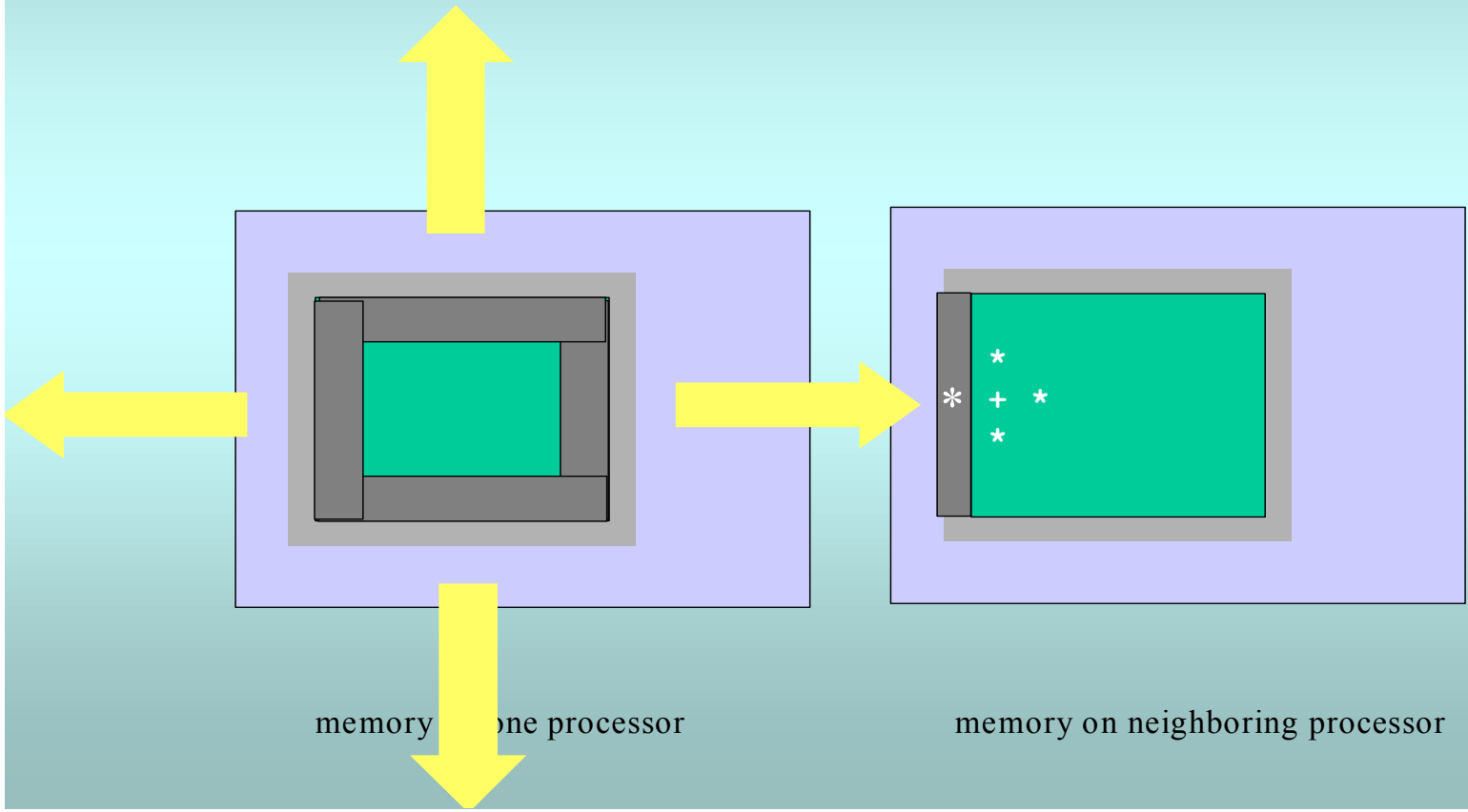
Tile: section of a patch allocated to a shared-memory processor within a node; this is also the scope of a model layer subroutine.

Distributed memory parallelism is over patches; shared memory parallelism is over tiles within patches

(From WRF tutorial:http://wrf-model.org/PRESENTATIONS/2002_06_NCAR_Tutorial/software.ppt

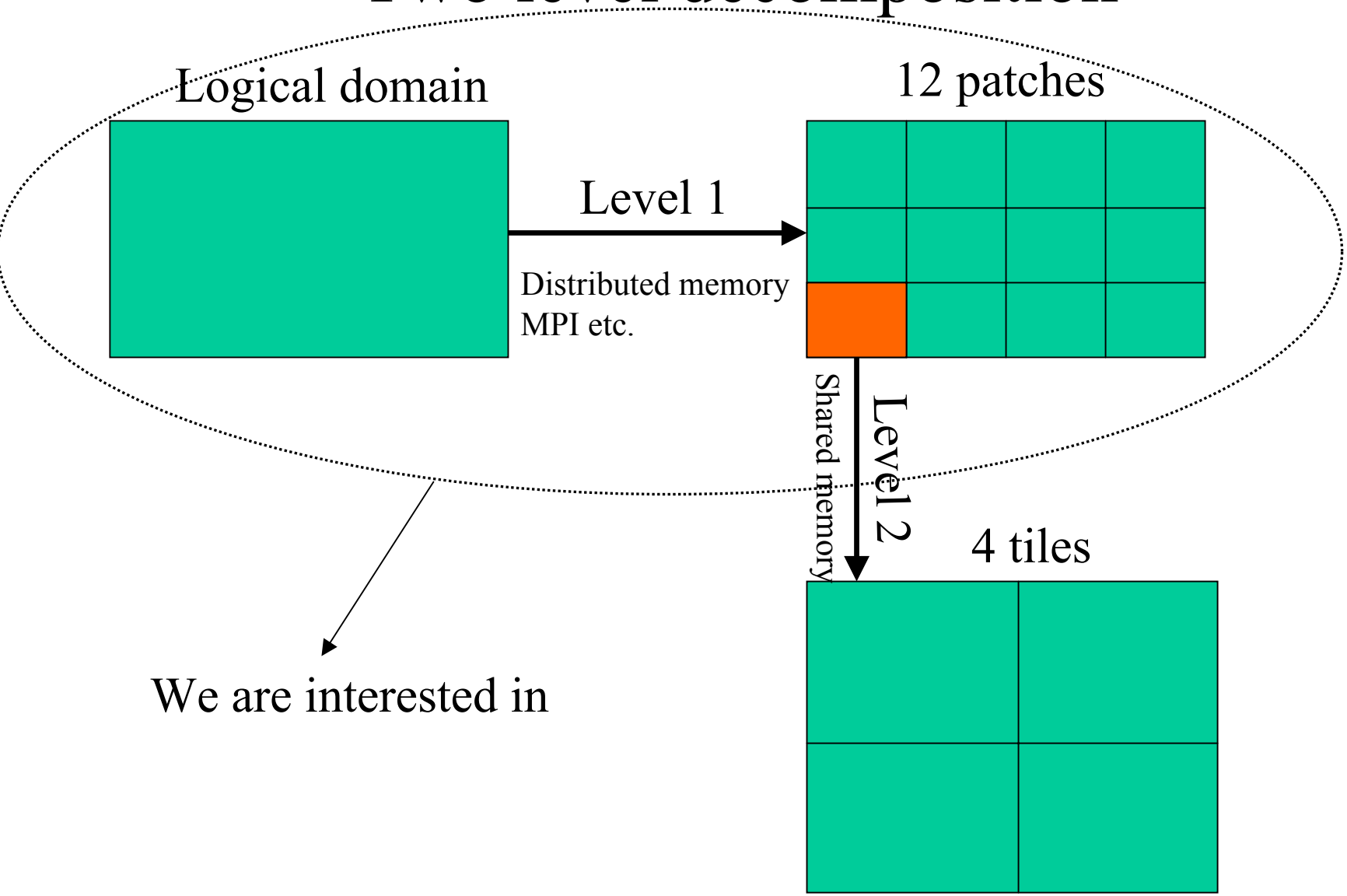
Distributed Memory Communications

- Halo updates



From WRF tutorial:http://wrf-model.org/PRESENTATIONS/2002_06_NCAR_Tutorial/software.ppt

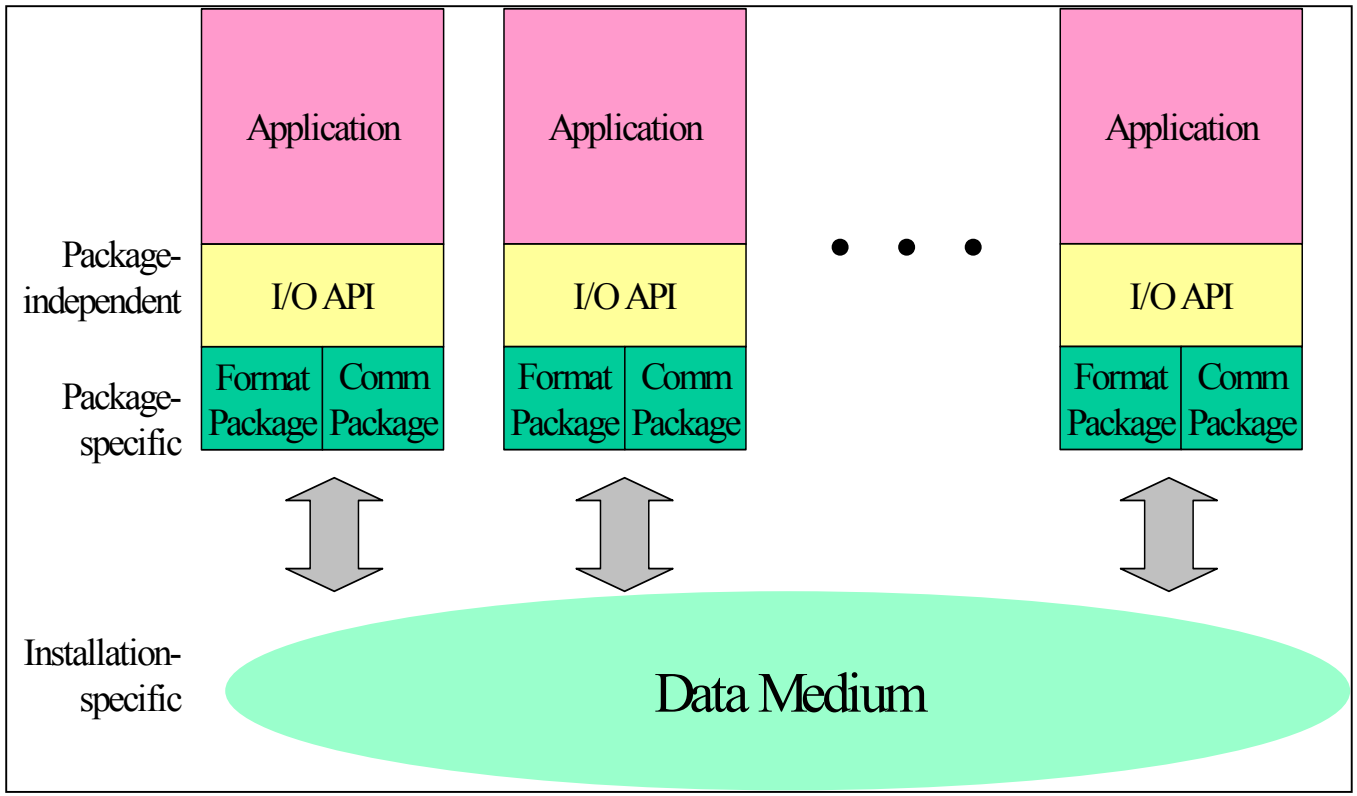
Two level decomposition



Goal of I/O API

- Make I/O infrastructures for both popular operational and research format
  - Operations: GRIB, BUFR
  - Research: NetCDF, HDF
- "Portable I/O" – treat each I/O package as an external package; easy to turn on or off

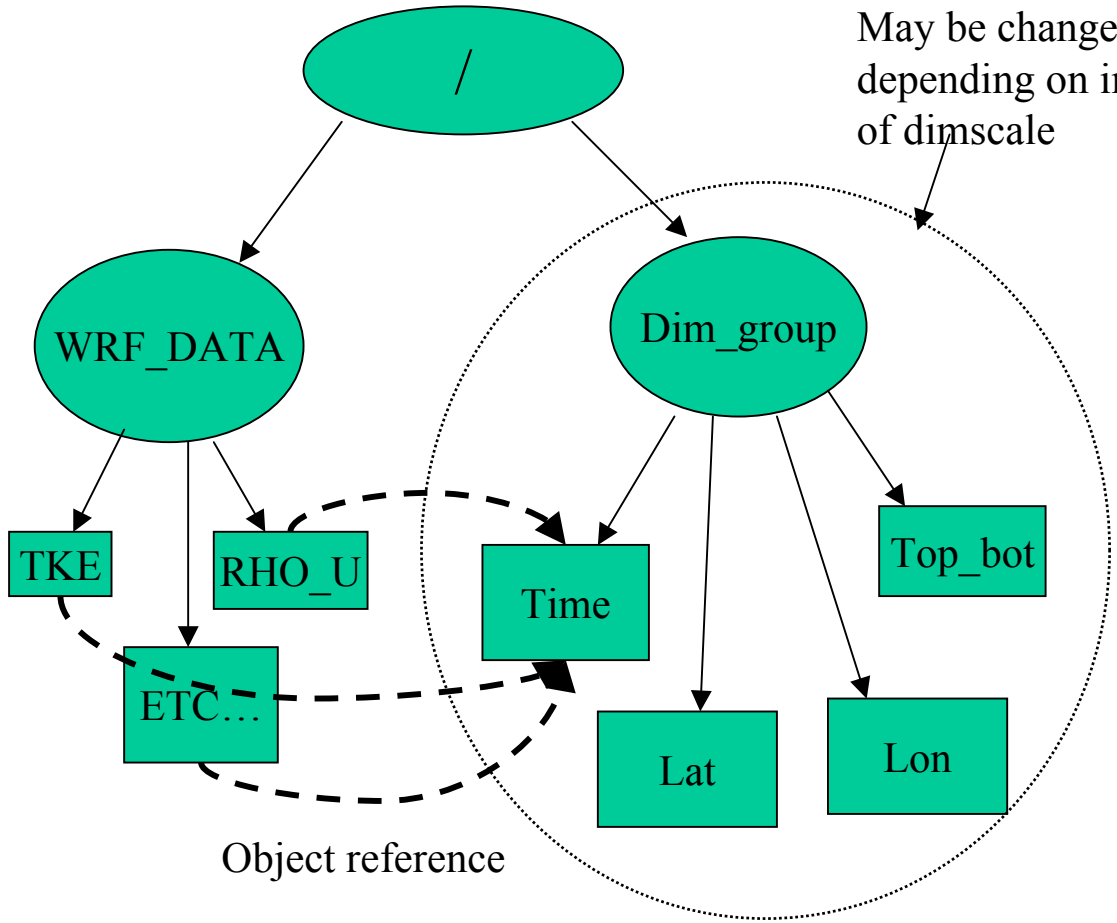
WRF I/O Schematic



WRF data characteristics

- Many HDF5-equivalent datasets
  - currently output >100 datasets, each one < 1MB
  - typical dataset: 4-D real array with the dimension of time extensible
- Many output files but each file not big
- Weakly dimensional scale requirement
  - no real dimensional scale data
- May need datasets to be extensible
- WRF has history, restart, initial and boundary dataset type. Each WRF dataset is equivalent to one HDF5 group.

First design of Schematic HDF5 File Structure of WRF Output



Solid line: HDF5 datasets or sub-groups (the arrow points to) that are members of the HDF5 parent group.
Dash line: the association of one HDF5 object to another HDF5 object;in terms of HDF5, object reference.

HDF5 dataset(WRF field)

HDF5 group(WRF dataset)

An example of WRF-HDF5 output

```
HDF5 "wrfrst_01_000002" {
GROUP "/" {
  GROUP "DATASET=RESTART" {
    DATASET "ACSNOM" {
      DATATYPE  H5T_IEEE_F32BE
      DATASPACE  SIMPLE { ( 1, 80, 40 ) / ( H5S_UNLIMITED,
80, 40 ) }
    }
                .
                .
                .
    }
}}
```

Must the dataset be extensible?

- Seems to best fit the description of the model (No timestamp limited for WRF)
- From J.M, the developer of WRF: impose no requirements at all on how the data are stored in the datasets.

WRF IO APIs

- Common APIs
  - Example: wrf_open_for_read `( DatasetName , Comm , IOComm_io,&SysDepInfo, DataHandle, Status )`
- Treat IO packages as external packages
  - Example: ext_open_hdf5_for_read `( DatasetName , Comm , IOComm_io,&SysDepInfo, DataHandle, Status )`
- May become common IO APIs for other atmospheric/ocean models like ROMS

A piece of example code

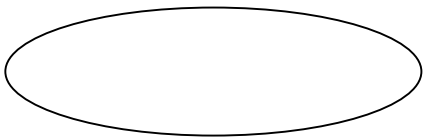
```
SUBROUTINE wrf_open_for_read ( FileName , Comm_compute, Comm_io, SysDepInfo, &DataHandle
      , Status )                                          USE module_state_description
      ……
Select CASE ( use_package( io_form ) ) {
#ifdef NETCDF
 CASE(IO_NETCDF)
 CALL ext_ncd_open_for_read
#endif
#ifdef HDF5
   CASE(IO_HDF5)
 CALL ext_hdf5_open_for_read
#endif
……
}
```

IO quilt server

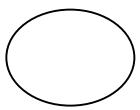
- Not using MPI-IO
- Use separate I/O nodes to generate output asynchronously while computing continues in computing nodes

IO Quilt server schematic

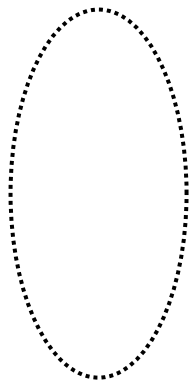
In the illustrated figure:
Totally there are 17 processors
12 processors : computing tasks
4 computing groups
5 processors: IO tasks
1 IO server group



Computing group

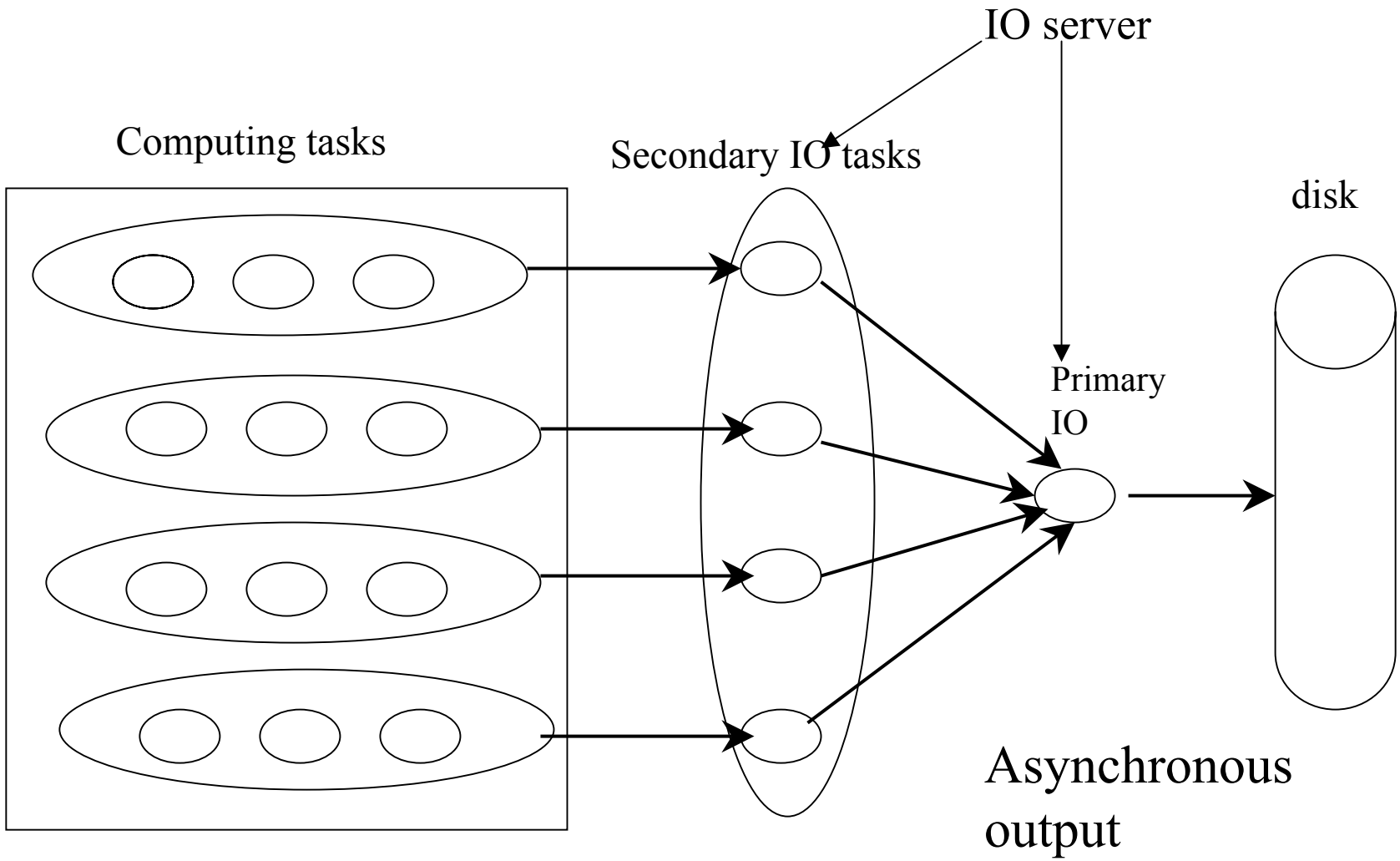


One task



IO group

IO Quilt server schematic continued



Evaluation of IO Quilt server

- Improve performance
- Somehow waste some computing processors
- If IO is too slow, computing nodes have to wait until the finishing of IO in the previous timestamp

Why using HDF5?

- HDF5 is the only data format that supports MPI-IO.
- Potential big file size in the future
- Potential user at NCSA- BW

Milestone highlights

- By March 31st, 2003:

  -Complete the initial prototype implementation of WRF-HDF5 including parallel I/O support.

- By June 30th, 2003:

  -Possibly re-design HDF5 structures of WRF output to take advantage of parallel HDF5 I/O.

  -Begin the initial implementation of serial and parallel versions of HDF5 module in WRF model.

Strategy of this project

1. Starting from the implementation of the sequential HDF5 parallel writer module
2. Learning parallel HDF5 on IBM NCAR SP machine
3. Designing parallel HDF5-WRF module
4. Implementing HDF5 parallel writer module
5. Testing with real case and do performance comparison study with sequential HDF5 and NetCDF.

Existing WRF IO modules

- NetCDF: sequential I/O that can be used in parallel run

  Internally a monitor will collect all the data from the computing node and write data out to the disk.
- IO quilt server
- Internal MM5 data format

Challenges for sequential mode

- To understand how and why WRF wants to transpose data from different memory orders to disk
- The current NetCDF implementation can only handle with fixed number of maximum timestep

Solutions to the challenge

- Transposing the data

  -In memory, to gain high performance of computing; fields may need to be in different orders (XYZ, XZY, YZX etc.)

  -In disk, we could store the data in a file with the same order as it is stored in memory. However, that may cause tool developers to do more work.

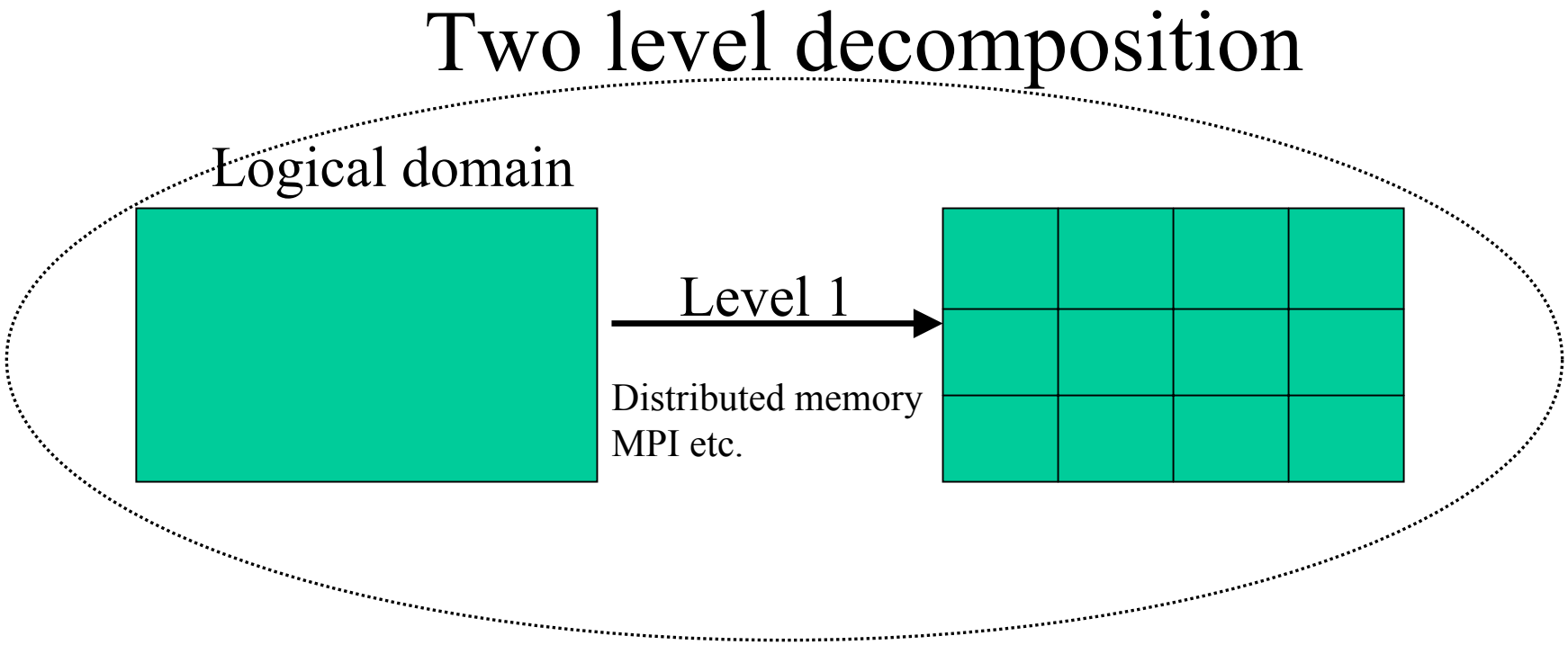
  For this reason, when data fields are written to the disk, they will always be transposed to the same order. When data fields are read from disk, the model will transpose to the correct memory order for each field.

An incomplete WRF-HDF5 writer

- Can write raw data in HDF5 format
- Doesn’t implement attributes yet
- Doesn’t implement dimensional scale yet

  - wait for implementation of HDF5 dimensional scale
- Has been tested on O2K, PC linux and NCAR IBM SP

Parallel WRF-HDF5 design



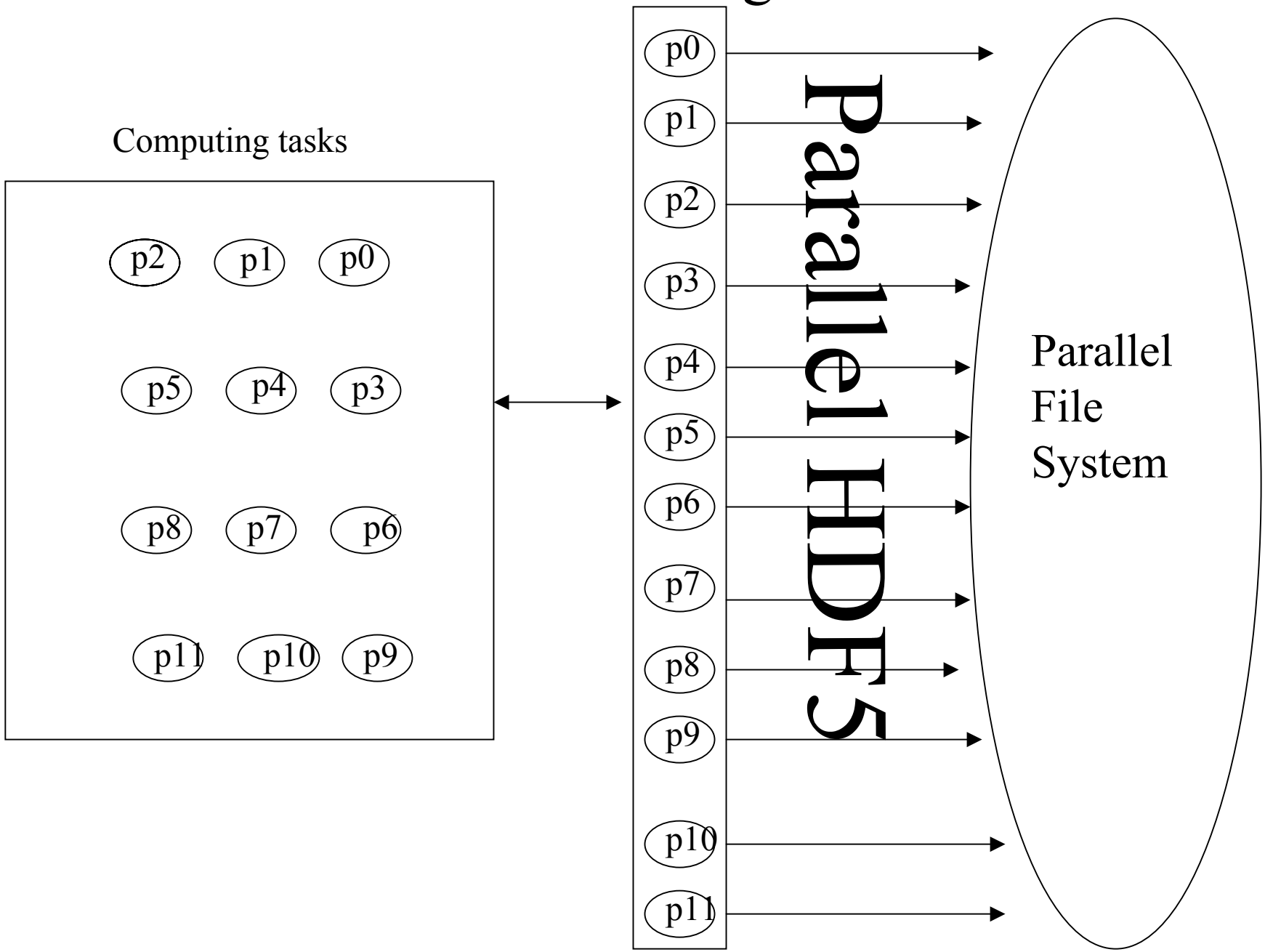
Parallel WRF-HDF5 design

Solution 1:

- Each computing node is also an I/O node;

  All nodes will participate in writing data to the output file in a parallel file system.

Parallel WRF-HDF5 design - Solution 1



Parallel WRF-HDF5 design 1

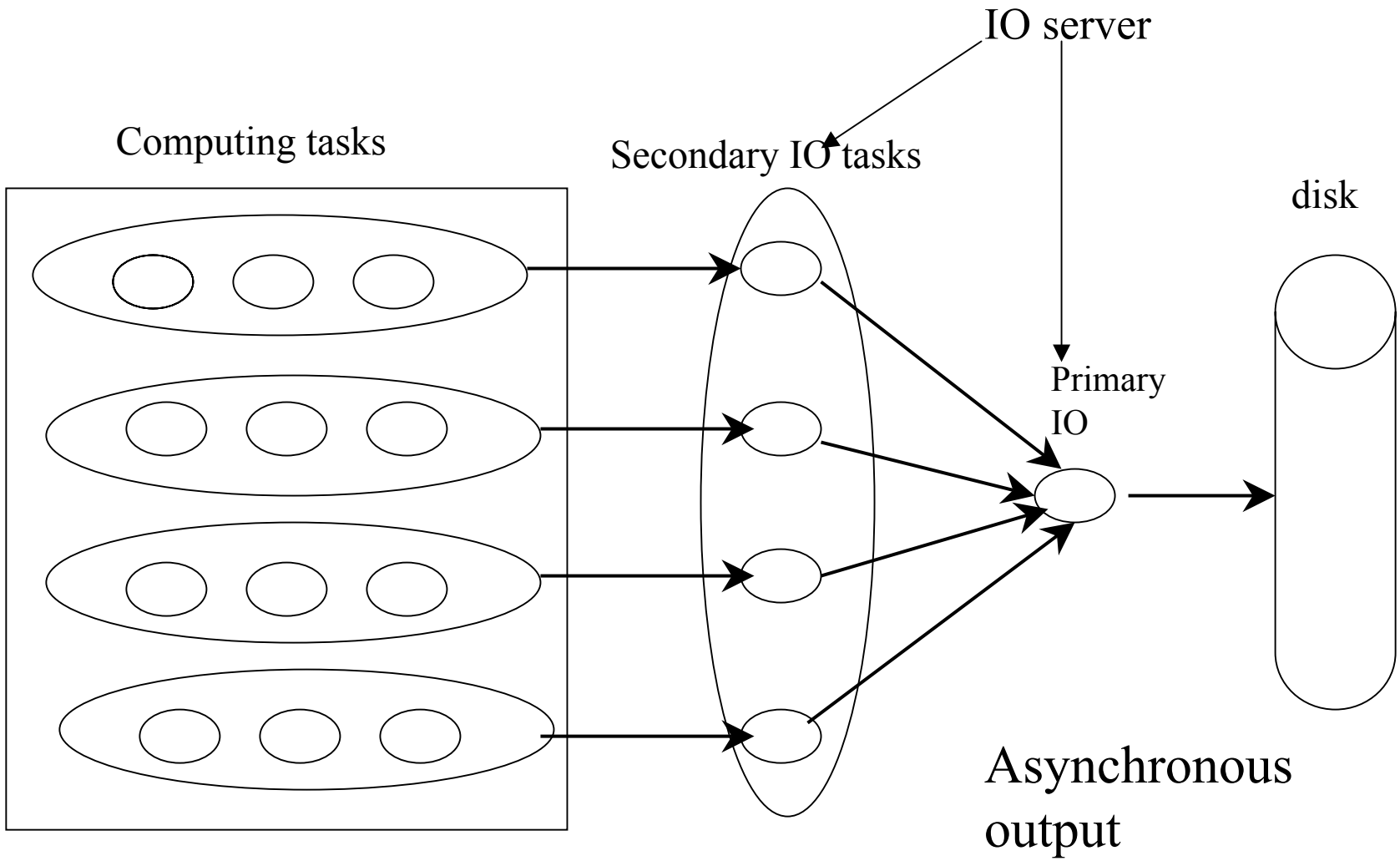
Pros:

1) Kind of easy to implement

Cons:

1) With many processors, overhead(many small writes) will be expensive.

IO Quilt server schematic continued

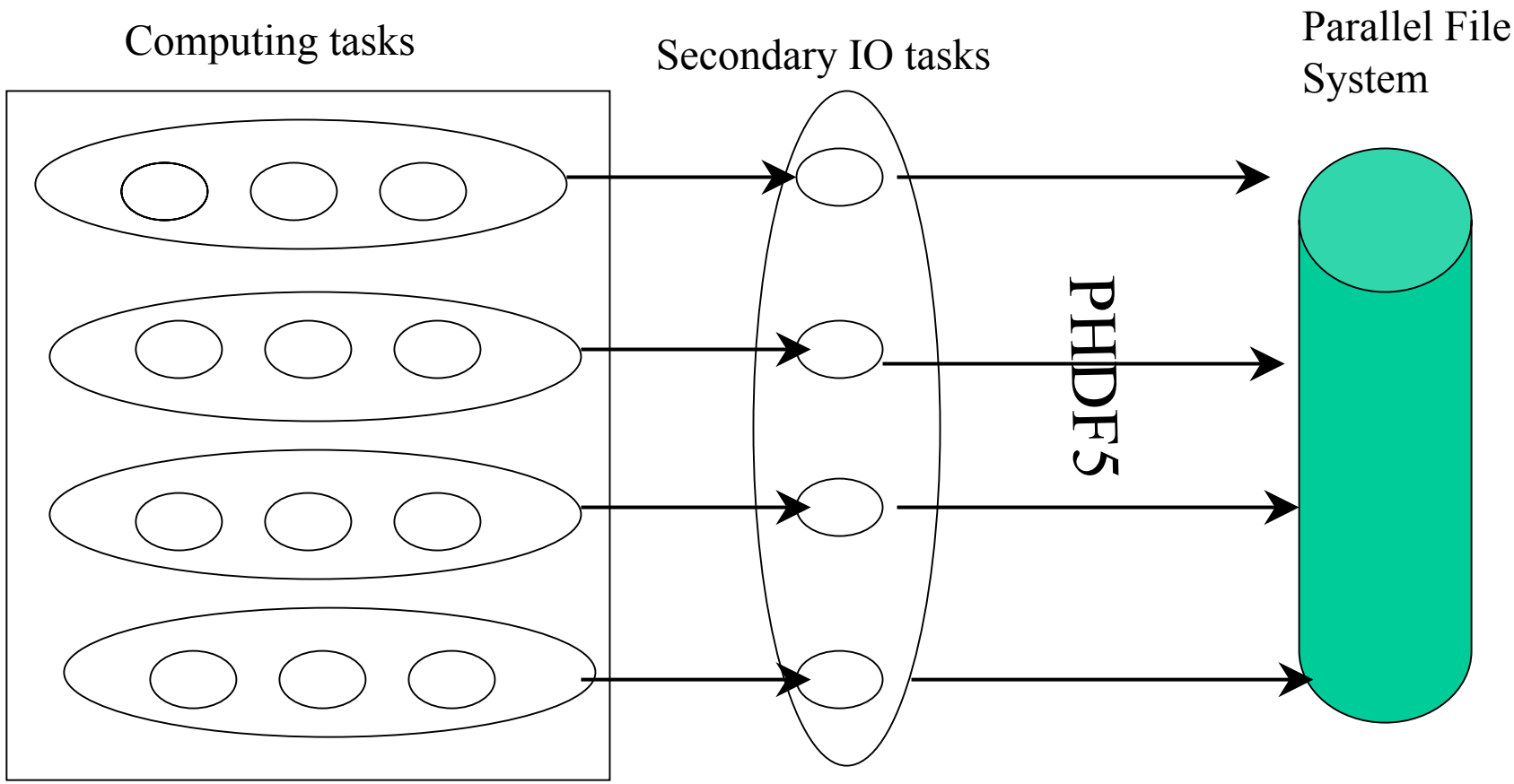


Parallel WRF-HDF5 design

Solution 2:

- Like I/O quilt server, Divide nodes into computing nodes and I/O nodes.
- WRF data fields are written to the disk through I/O nodes via parallel HDF5.

Parallel WRF-HDF5 design-solution 2



Parallel WRF-HDF5 design 2

Pros:

1) Overhead is small, scaleable?

Cons:

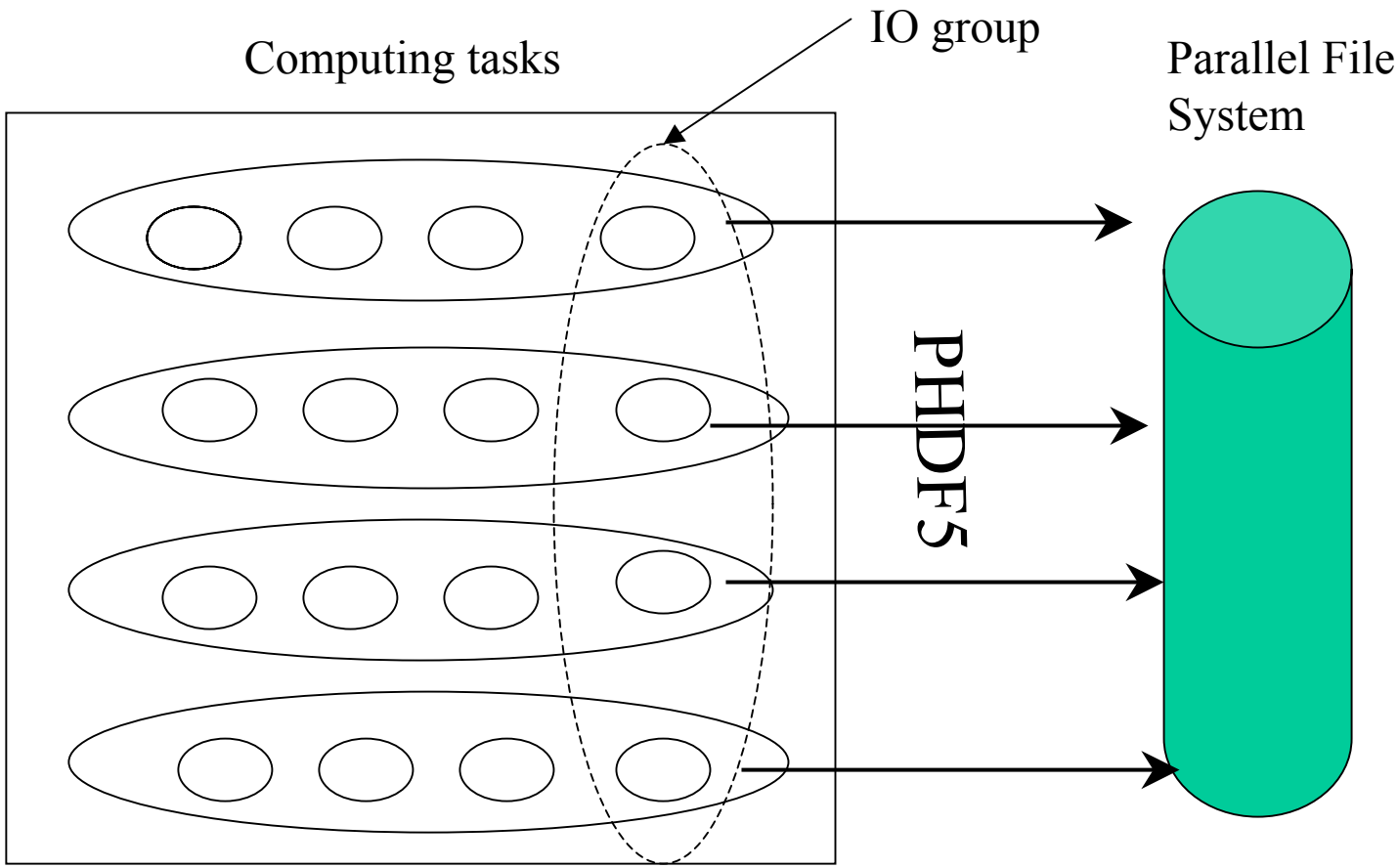
1) Need extra work to transfer data from computing nodes to IO nodes
2) For computing bound, IO nodes kind of Waste.

Parallel WRF-HDF5 design

Solution 3:

- Similar to solution 2, but all nodes will be used to compute; some nodes will also be used for IO.
- WRF data fields are written to the disk through I/O nodes via parallel HDF5.

Parallel WRF-HDF5 design-solution 3



Parallel WRF-HDF5 design 3

Pros:

1) Overhead is small, scaleable?
2) For computing bound, gain performance.

Cons:

1) Need extra work to transfer data from computing nodes to IO.
2) For IO bound, may affect performance.

An extra problem for all solutions

- The HDF5 dataset has to be extensible in time. This requires chunk storage.
- Challenge for chunk size:

  The dimensional size in each patch is not necessarily the same, the chunk size can not be assigned easily.
- Two potential solutions for chunk size problem:
  - find common factor of the chunk size: may cause very bad performance
  - Partial data transferring from computing nodes to I/O nodes and using synchronization schemes(semaphore etc.) to avoid inconsistencies when writing data from different I/O nodes to the same place in a dataset. Hard to implement and cannot assure good performance.

Another thought to handle this problem

- Does WRF really need WRF fields to be extensible?

  Answer: Not required according to WRF developers.

  Good news, bad news? Then how we store the raw data?

Some facts related to WRF-HDF5 module

			WRF-HDF5 module
Sequential Computing		Sequential IO	Sequential HDF5
Parallel Computing		Sequential IO	Sequential HDF5
Parallel Computing		Parallel IO	Parallel HDF5