Towards High Performance Data Analytics for Climate Change

S. Fiore[1], D. Elia[1,2], C. Palazzo[1], F. Antonio[1], A. D'Anca[1], I. Foster[3], G. Aloisio[1,2]

[1] *Euro-Mediterranean Center on Climate Change (CMCC) Foundation, Lecce, Italy*
[2] *University of Salento, Lecce, Italy*
[3] *University of Chicago & Argonne National Laboratory, Chicago, USA*

HPC-IODC 2019 Workshop
Frankfurt, 20 June 2019

esiwace
CENTRE OF EXCELLENCE IN SIMULATION OF WEATHER AND CLIMATE IN EUROPE

cmcc
Centro Euro-Mediterraneo sui Cambiamenti Climatici

*This work was supported in part by the EU H2020 ESiWACE project (Grant Agreement 675191)

The Ophidia project

Ophidia (http://ophidia.cmcc.it) is a CMCC Foundation research project addressing data challenges for eScience

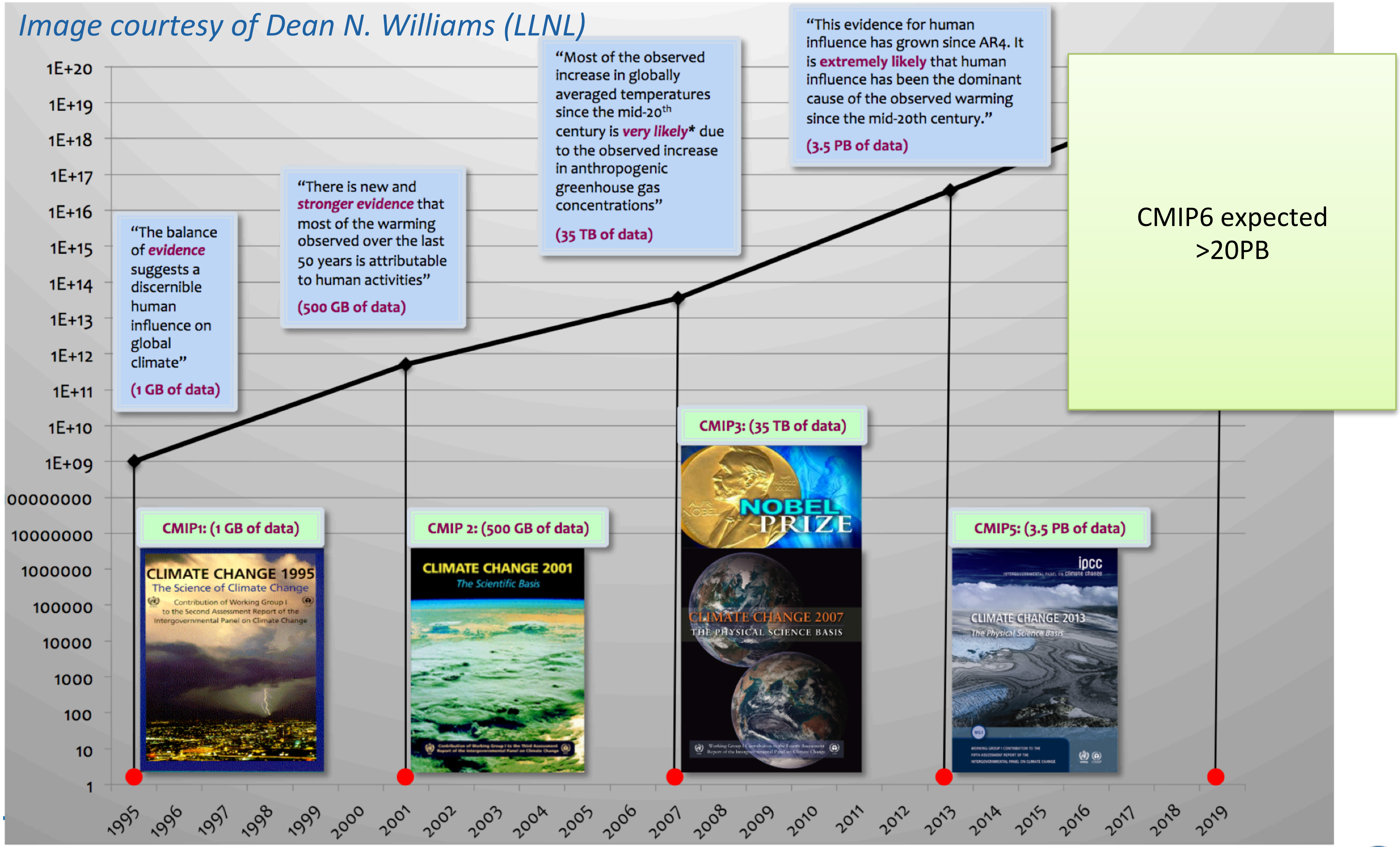
It provides:

- ✓ a *High Performance Data Analytics* (HPDA) framework joining HPC paradigms with scientific data analytics approaches
- ✓ support for declarative, in-memory, parallel, server-side data analysis exploiting parallel computing techniques and database approaches
- ✓ end-to-end mechanisms to support complex experiments and large workflows on scientific datacubes, primarily in climate domain

Ophidia

cmcc Centro Euro-Mediterraneo sui Cambiamenti Climatici

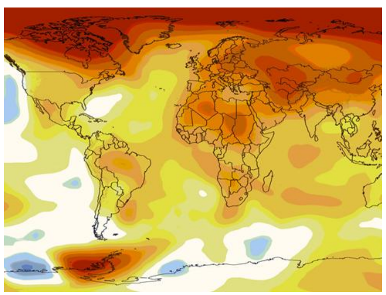
A data perspective of the CMIP experiments

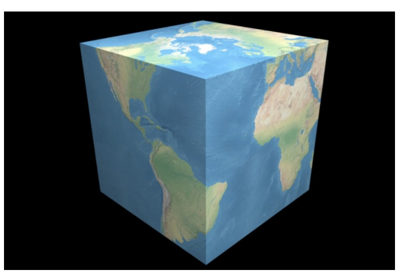


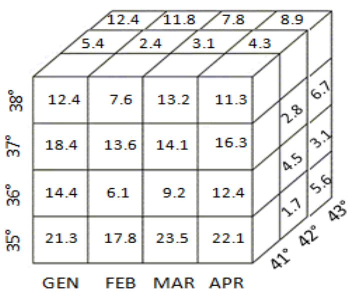
Scientific data analysis workflow & paradigm shift

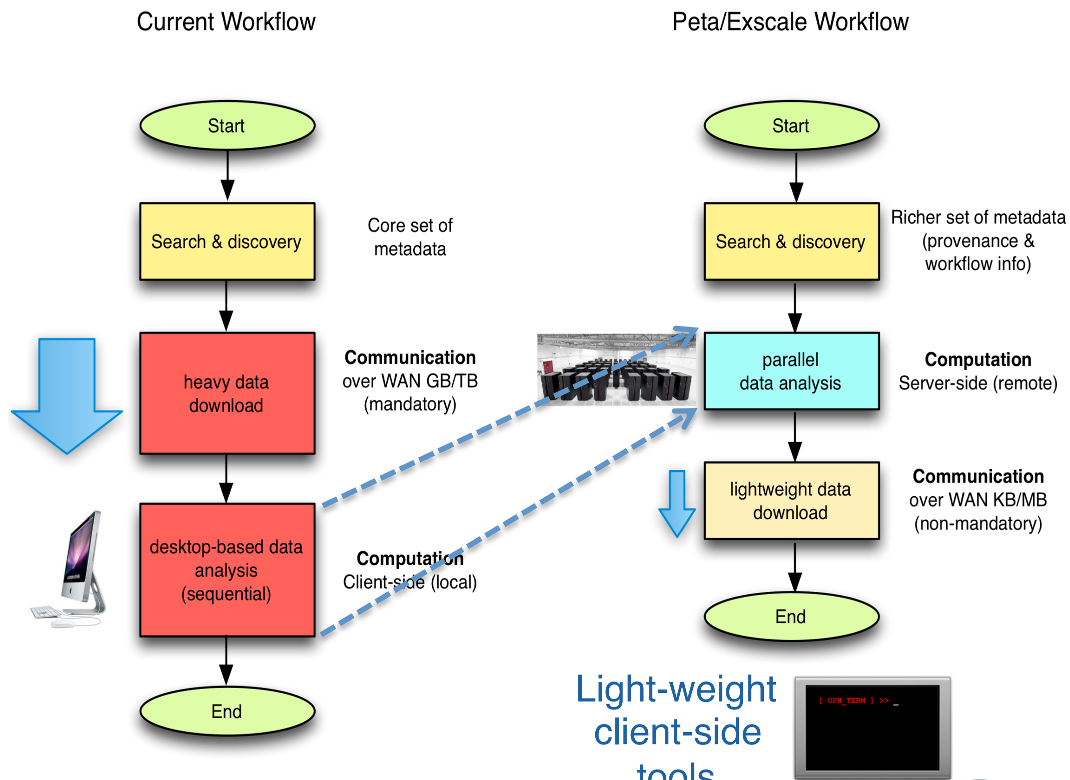
The deluge of data, poses challenges that must be tackled accordingly to cope with bigger data volumes, heterogeneous formats and different frequency in data generation.

Time-consuming downloads, client-side & sequential processing are three limiting factors for the traditional scientific data analysis workflow.









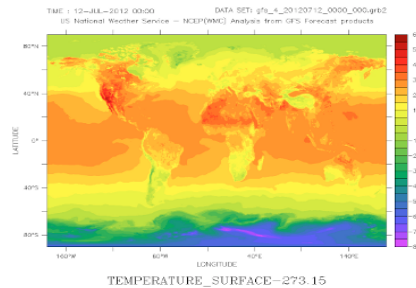
Data analytics challenges and requirements

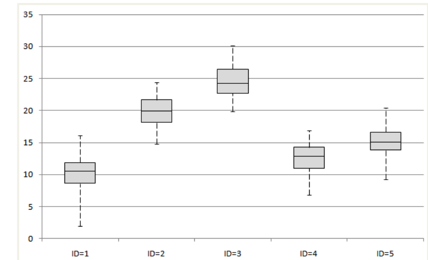
Requirements and needs focus on:

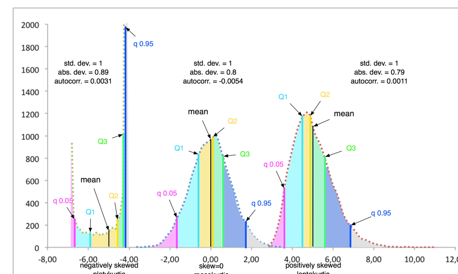
- *Time series analysis*
- *Data subsetting*
- *Multimodel means*
- *Massive data reduction*
- *Ensemble analysis*
- *Data analytics workflow support*
- *Metadata management*
- *Data/experiment provenance*
- *Information linking through cross-related digital object*

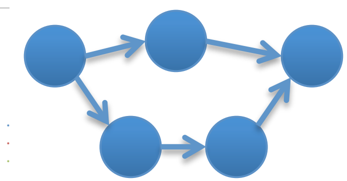
But also…

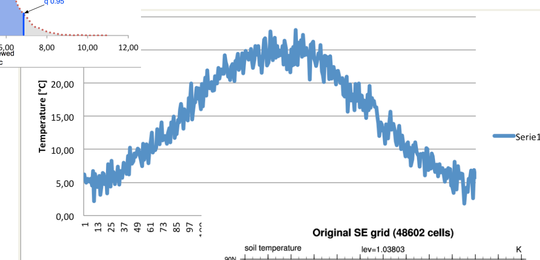
- *New storage models for multi-dimensional data(cubes)*
- *Data partitioning and distribution (parallel I/O)*
- *Performance (parallel analytics)*
- *re-usability and extensibility*

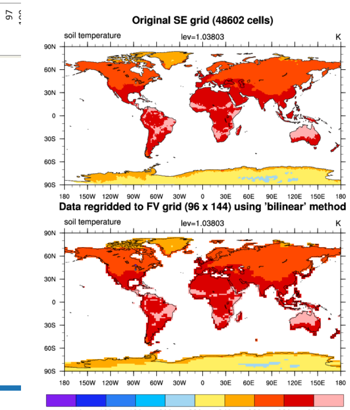










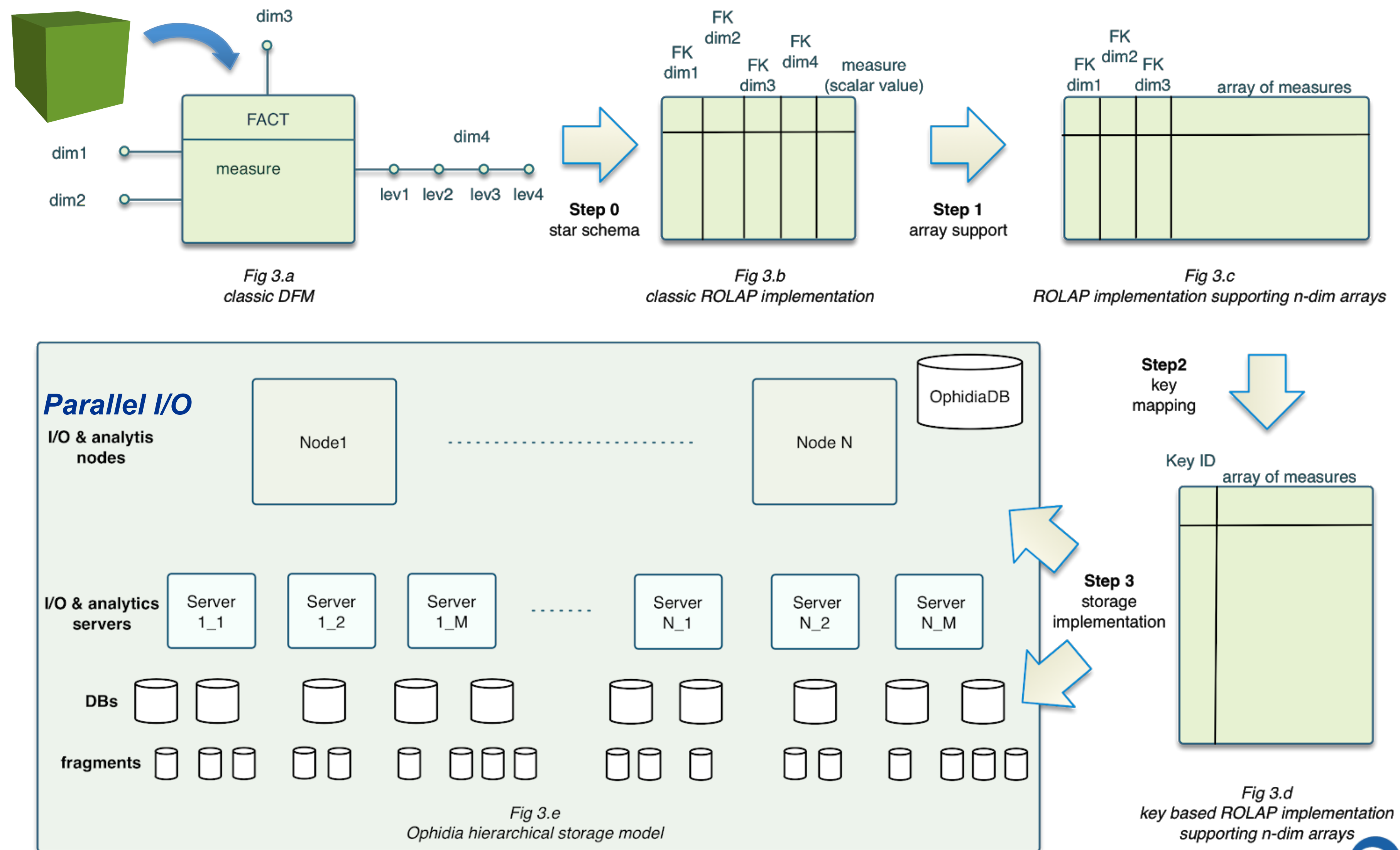


Ophidia in a nutshell

- ✓ *HPDA* software stack for multi-dimensional scientific data management
- ✓ Server-side, parallel, in-memory I/O & analytics
- ✓ Proposes a multi-dimensional storage model and partitioning schema for scientific data leveraging the datacube abstraction
- ✓ *eScience oriented features* (i.e. climate change): e.g. time series analysis, data subsetting, data aggregation, model intercomparison, OLAP
- ✓ Reusability of intermediate results and *provenance* management, targeting Open Science principles
- ✓ *Extensible and simple API* to support framework extensions in terms of operators and array-based primitives
- ✓ Programmatic access via *Python APIs* (*batch* & *interactive* data analysis)
- ✓ Support for complex *workflows* / operational chains

Storage model implementation

Datacube



Storage model implementation

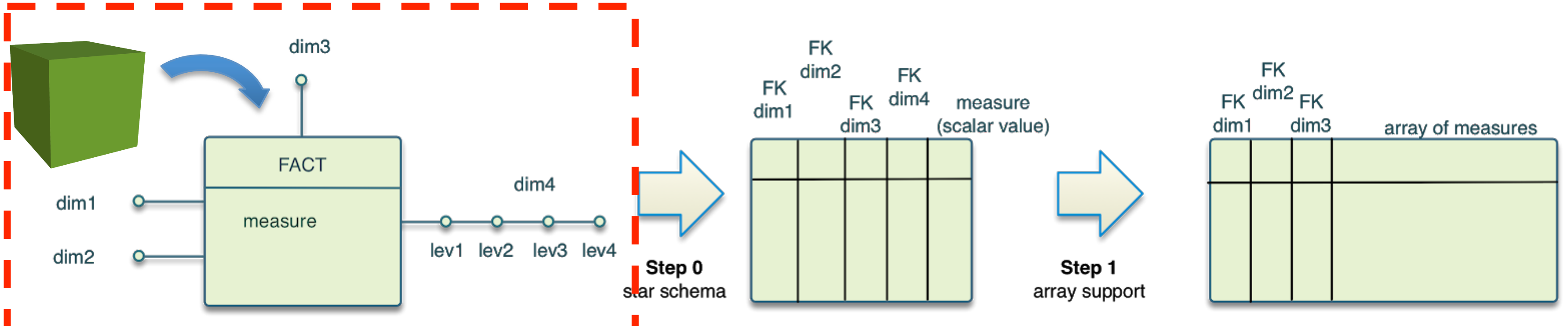


Fig 3.a
classic DFM

Fig 3.b
classic ROLAP implementation

Fig 3.c
ROLAP implementation supporting n-dim arrays

- ✓ A *datacube* consists of several *measures* representing numerical values that can be analyzed over *dimensions*.
- ✓ The Ophidia storage model builds on top of the classic *OLAP star schema*.
- ✓ The fact table is represented with the Dimensional Fact Model (DFM), a conceptual model for data warehouse
- ✓ This schema can be easily used to map a NetCDF file produced, for example, by a global climate simulation

Fig 3.e
Ophidia hierarchical storage model

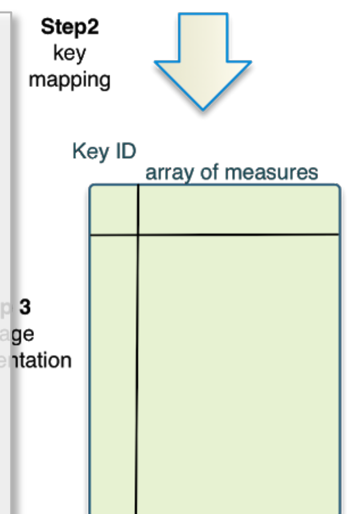
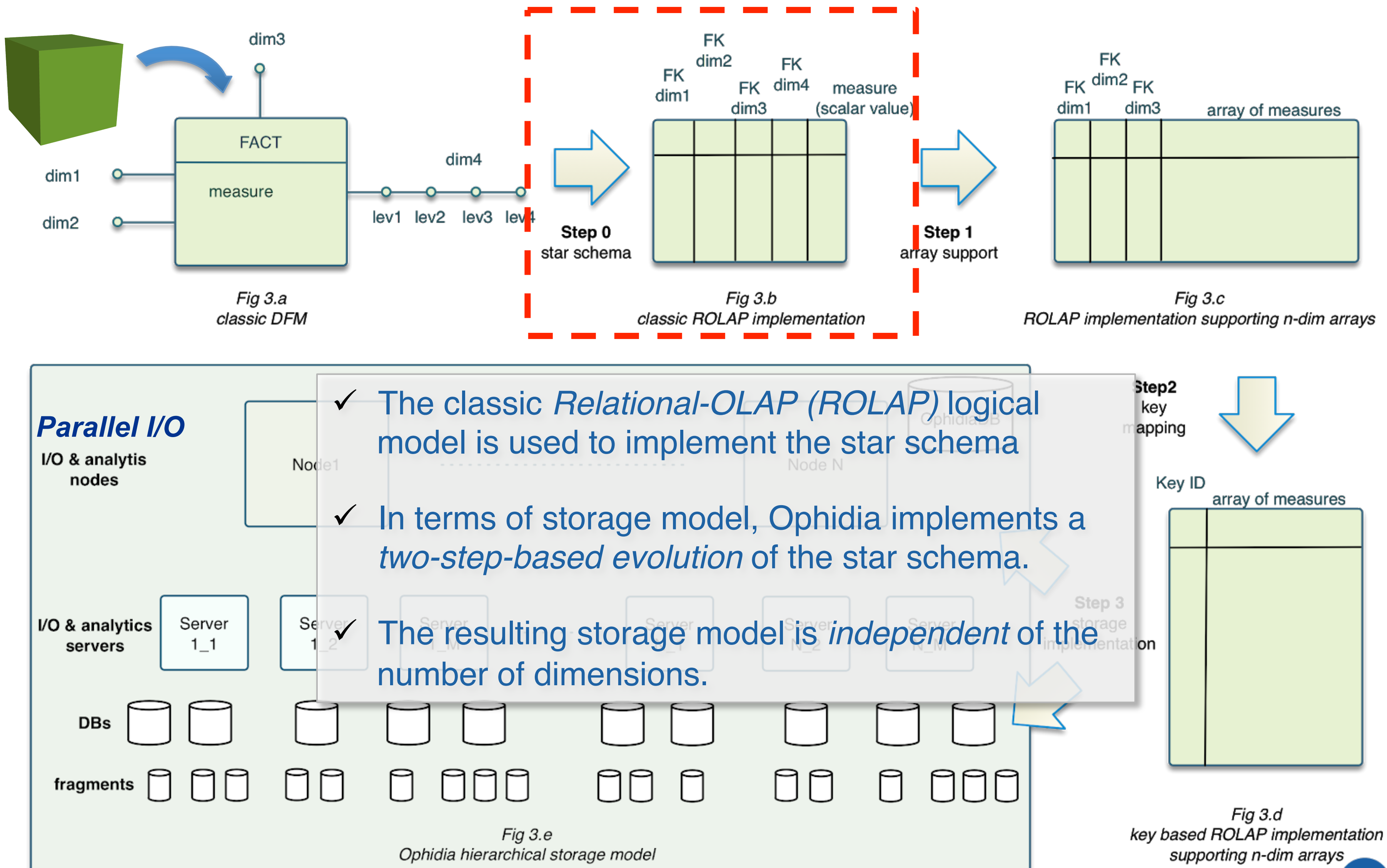


Fig 3.d
key based ROLAP implementation supporting n-dim arrays

Storage model implementation



- ✓ The classic *Relational-OLAP (ROLAP)* logical model is used to implement the star schema
- ✓ In terms of storage model, Ophidia implements a *two-step-based evolution* of the star schema.
- ✓ The resulting storage model is *independent* of the number of dimensions.

Storage model implementation

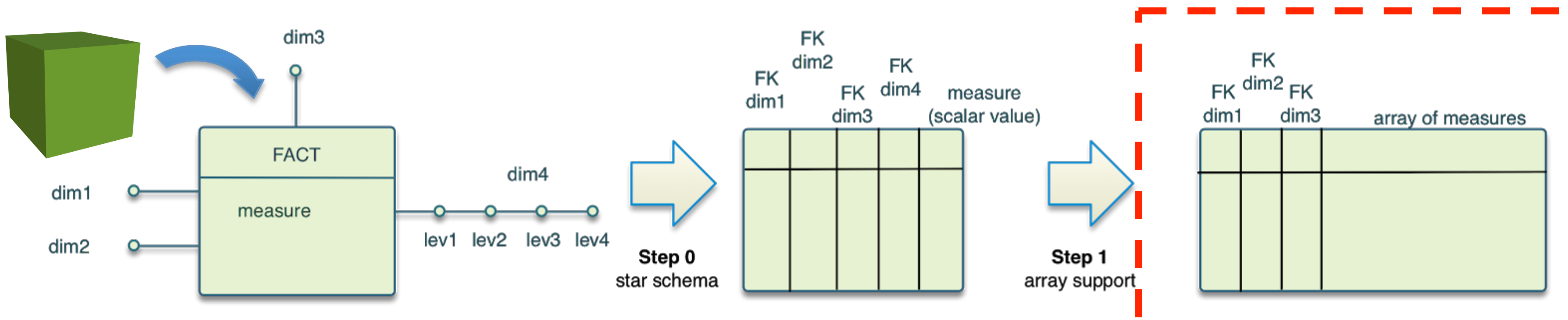


Fig 3.a
classic DFM

Fig 3.b
classic ROLAP implementation

Fig 3.c
ROLAP implementation supporting n-dim arrays

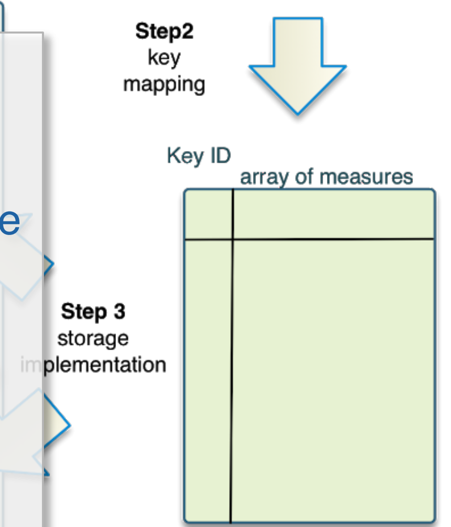


Fig 3.d
key based ROLAP implementation supporting n-dim arrays

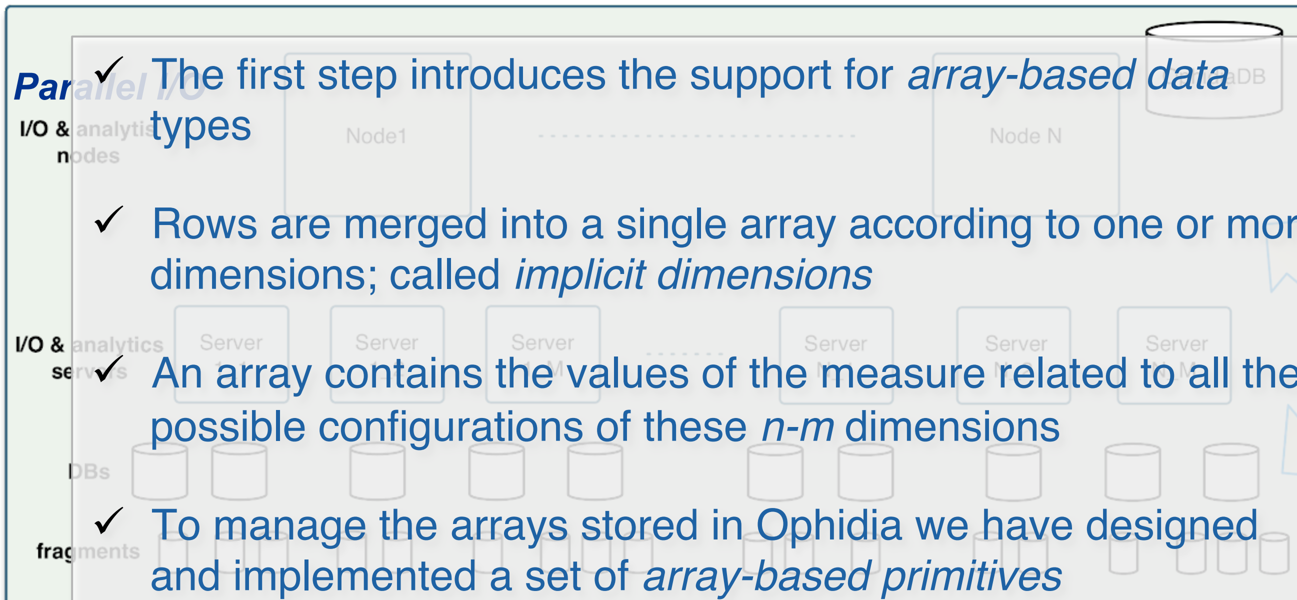


Fig 3.e
Ophidia hierarchical storage model

- ✓ The first step introduces the support for *array-based data* types
- ✓ Rows are merged into a single array according to one or more dimensions; called *implicit dimensions*
- ✓ An array contains the values of the measure related to all the possible configurations of these *n-m* dimensions
- ✓ To manage the arrays stored in Ophidia we have designed and implemented a set of *array-based primitives*

Storage model implementation

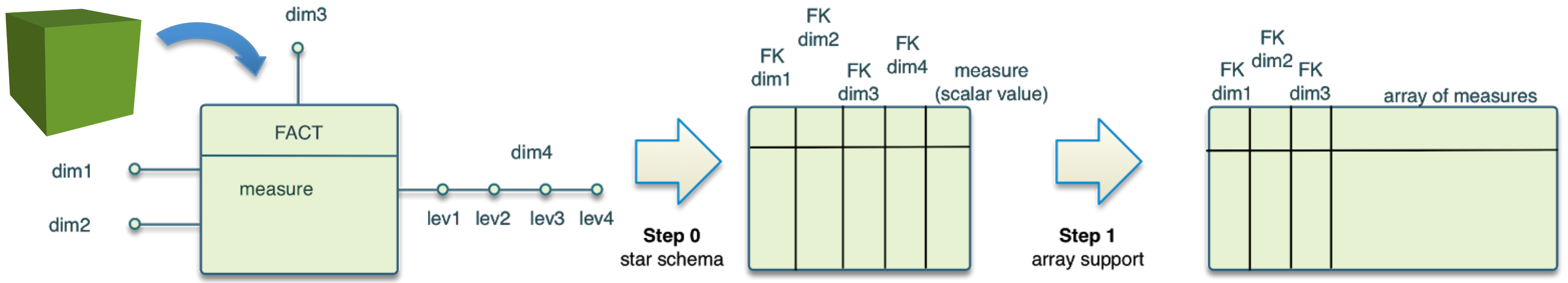


Fig 3.a
classic DFM

Fig 3.b
classic ROLAP implementation

Fig 3.c
ROLAP implementation supporting n-dim arrays

- ✓ The second step performs the mapping of the set of foreign keys (FKs) related to the remaining m dimensions to a single new key
- ✓ m dimensions, defined as *explicit dimensions*, are mapped through a numerical function onto the key attribute
- ✓ The mapping onto the Ophidia key-array storage model results in a single table with two attributes: an *ID* and a *binary array*
- ✓ A multidimensional array can be managed using a single tuple (e.g., an entire time series) identified by one key (a numerical ID)

Fig 3.e
Ophidia hierarchical storage model

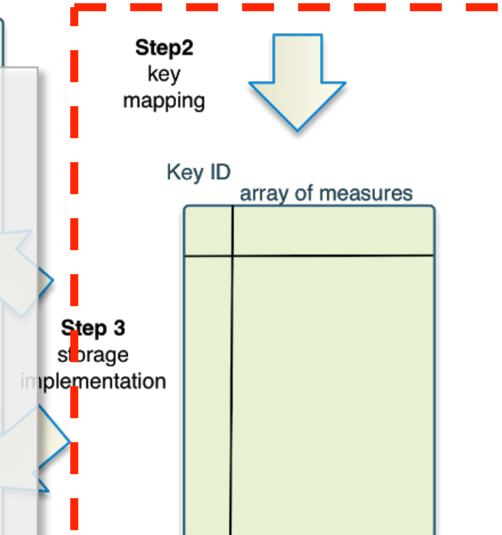


Fig 3.d
key based ROLAP implementation supporting n-dim arrays

Storage model implementation

Ophidia horizontally partitions this table into several *fragments* following a hierarchical approach composed of:

1) Level 0: multiple Ophidia I/O & analytics nodes (multi-host);
2) Level 1: multiple Ophidia I/O & analytics servers on the same node (multi-server);
3) Level 2: multiple instances of databases on the same IO & analytics server (multi-DB);
4) Level 3: multiple fragments on the same database (multi-table).

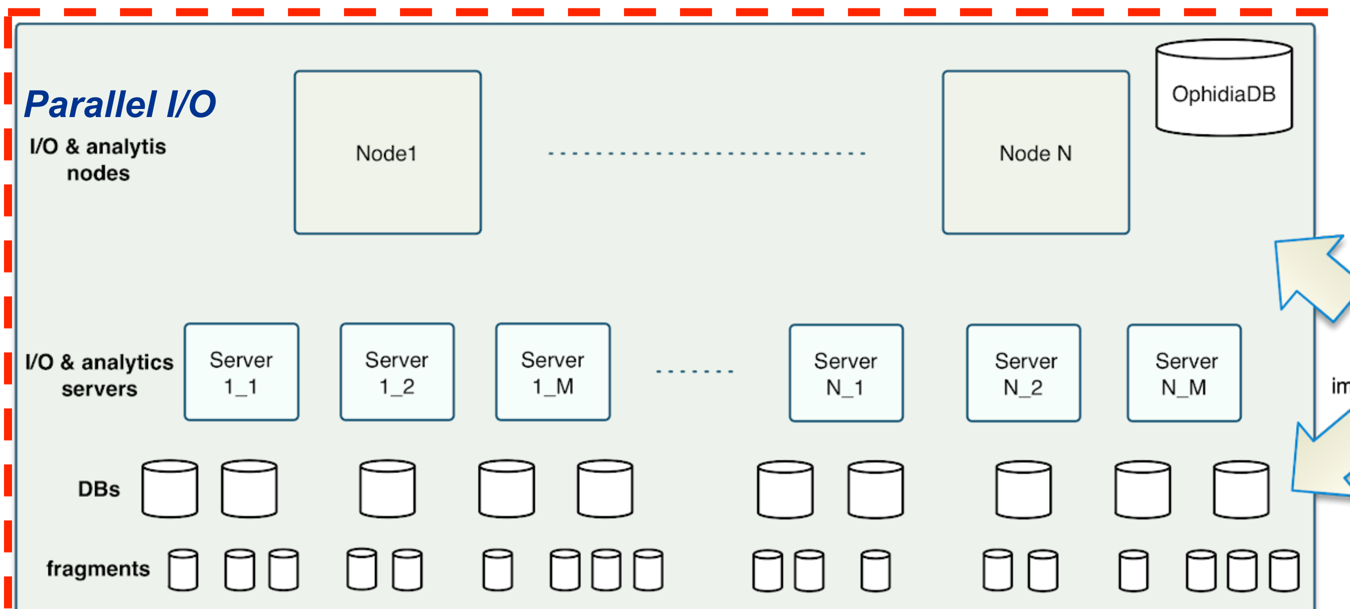


Fig 3.e
Ophidia hierarchical storage model

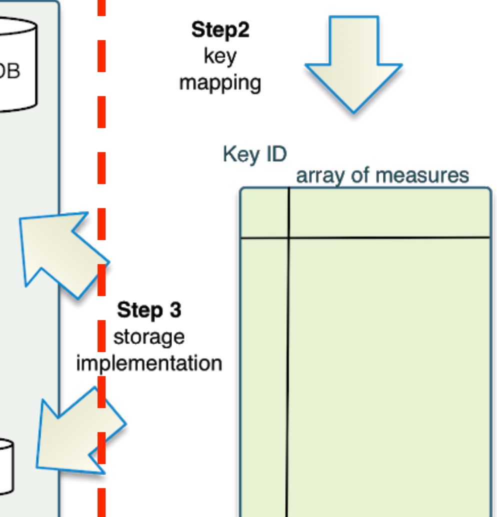
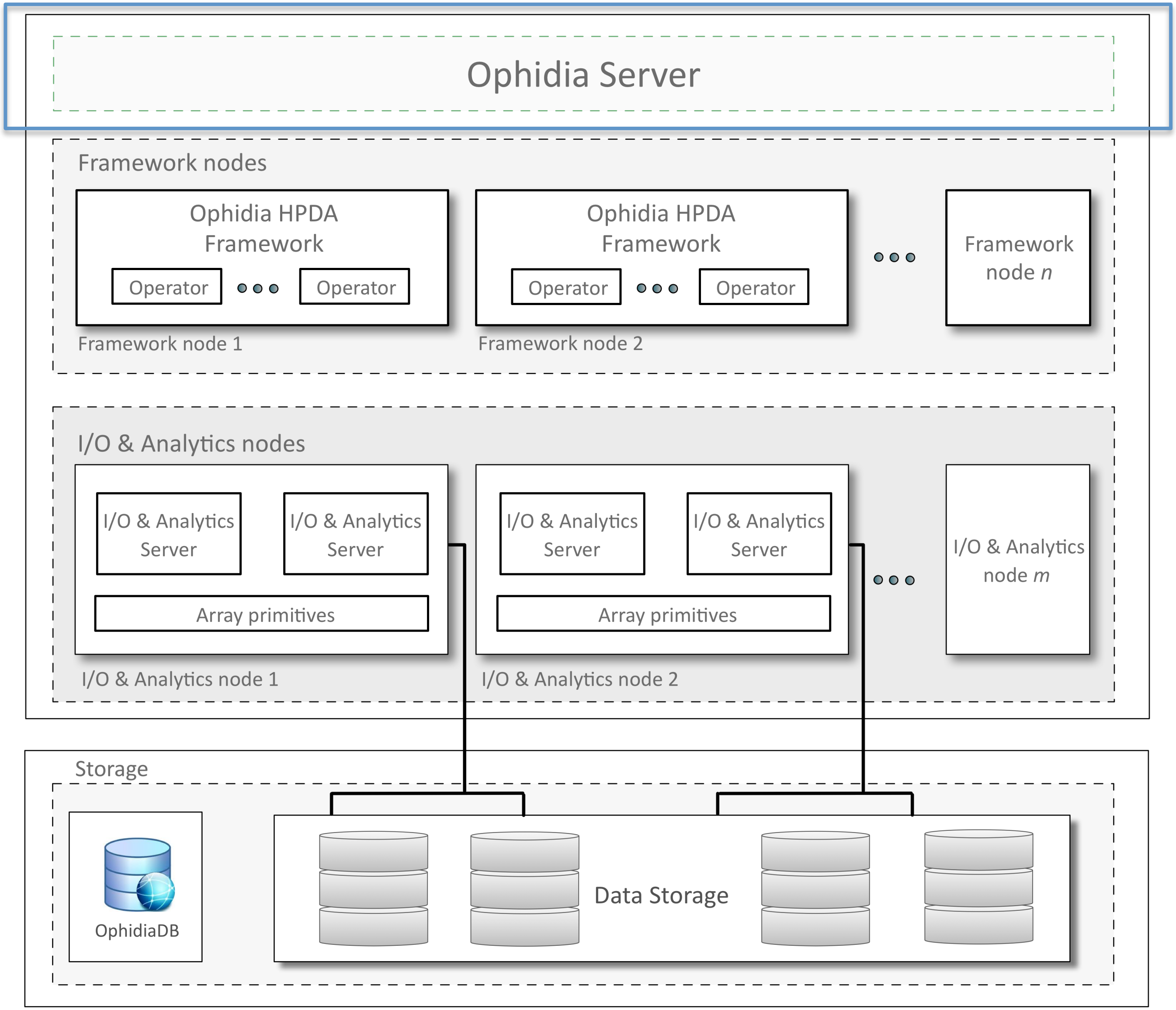


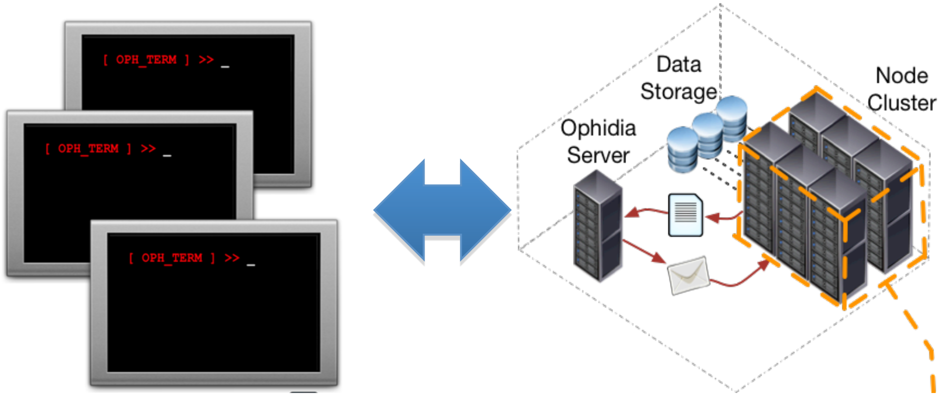
Fig 3.d
key based ROLAP implementation supporting n-dim arrays

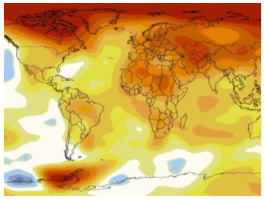
Ophidia architecture: front-end layer

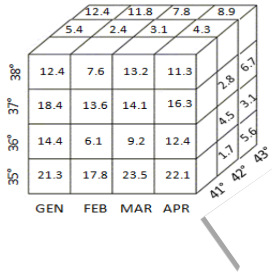
- ✓ Handles client-server interaction
- ✓ *Manages user **authN/authZ,** sessions and requests*
- ✓ *Manages **task/ workflow** execution*
- ✓ *Remote interactions with a CLI, WPS clients and Python modules*



Server-side paradigm and the datacube abstraction





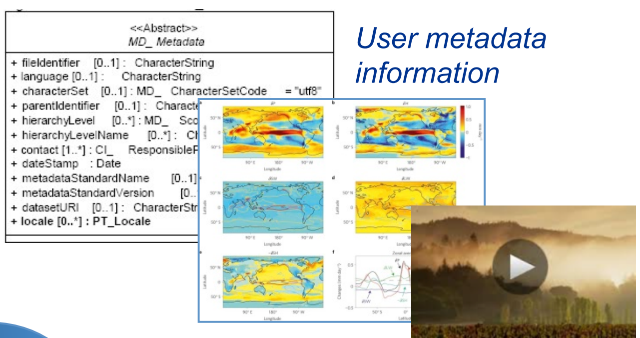


Oph_Term: a terminal-like commands interpreter serving as a client for the Ophidia framework

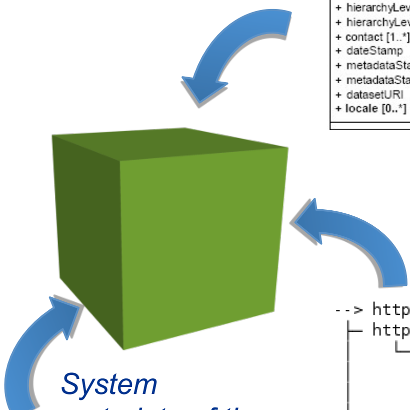
PyOphidia: a Python interface for datacube management & analytics with Ophidia

Ophidia framework: *declarative*, parallel server-side processing

Through **oph_term/PyOphidia** the user run ("send") commands ("operators") to the Ophidia framework to manipulate datasets ("datacubes")



User metadata information



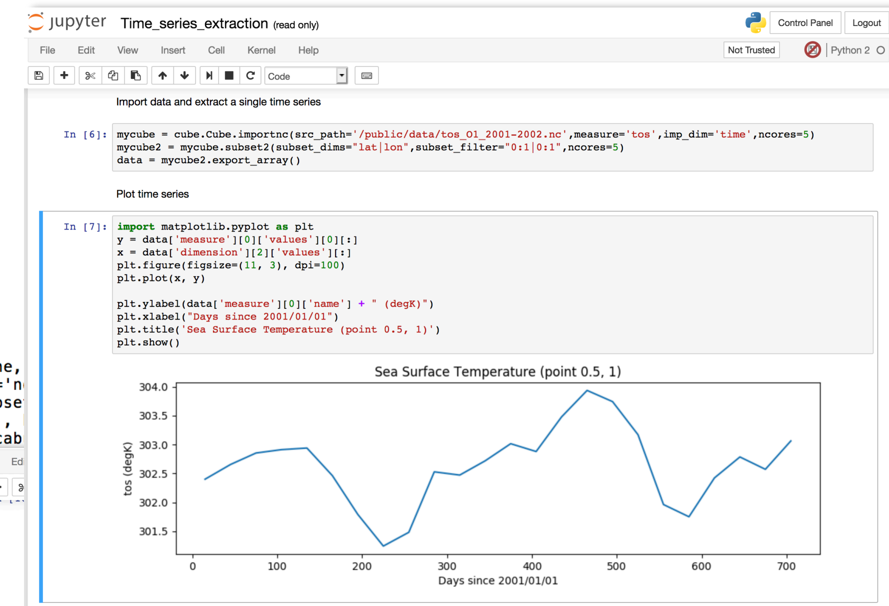
System metadata of the datacube (size, distribution, etc.)

Metadata provenance

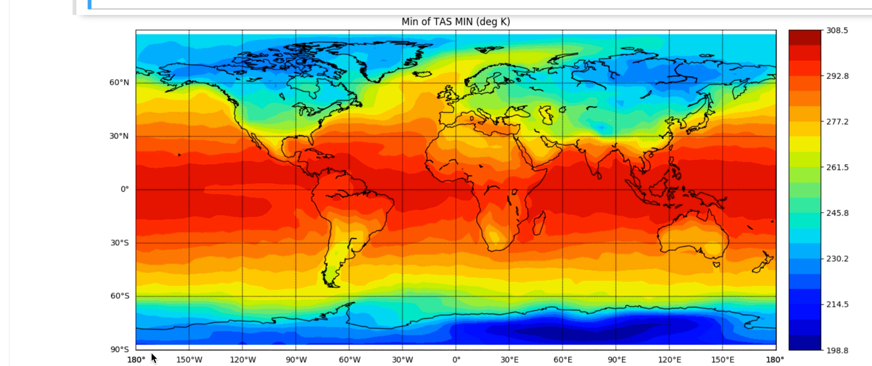
```
--> https://ophidia.cmcc.it:8443/162/169 (ROOT)
├ https://ophidia.cmcc.it:8443/162/170 (oph_reduce)
   └ https://ophidia.cmcc.it:8443/162/171 (oph_merge)
      ├ https://ophidia.cmcc.it:8443/162/172 (oph_aggregate2)
      └ https://ophidia.cmcc.it:8443/162/173 (oph_rollup)
         ├ https://ophidia.cmcc.it:8443/162/174 (oph_reduce)
         └ https://ophidia.cmcc.it:8443/162/175 (oph_reduce)
├ https://ophidia.cmcc.it:8443/162/176 (oph_aggregate)
└ https://ophidia.cmcc.it:8443/162/177 (oph_aggregate)
```

Programmatic access through the PyOphidia class

- ✓ **PyOphidia** provides a Python interface to submit commands to the Ophidia Server and to retrieve/deserialize the results (e.g. in Jupyter Notebooks)
- ✓ Two modules implemented:
  - ✓ **Client**: connect to the server, navigate the ophidia file system, submit workflows, manage sessions, etc.
  - ✓ **Cube class**: manipulate cubes objects through a Python abstraction



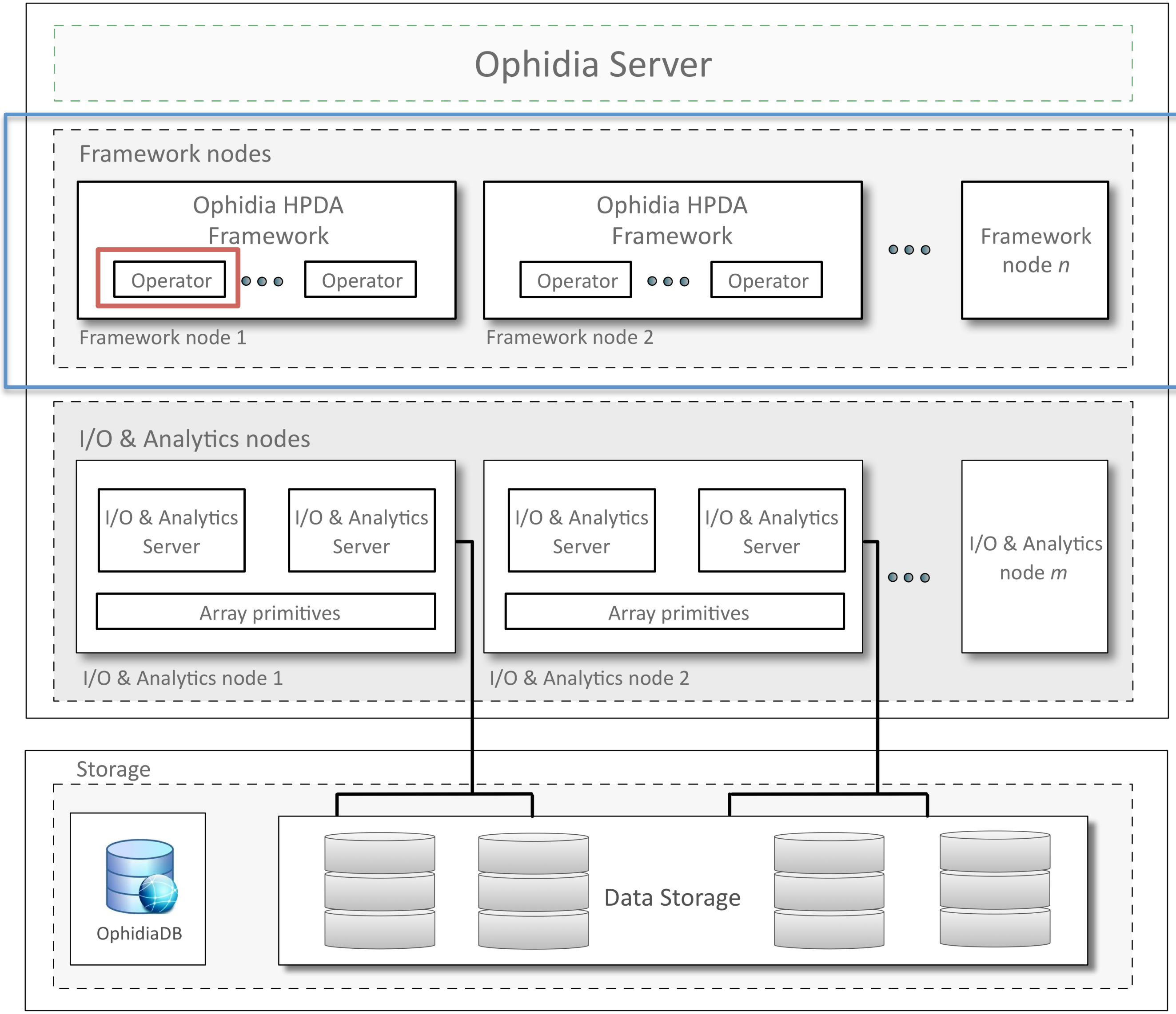

```
class Cube():
    """Cube(container='-', cwd=None, exp_dim='auto', host_partition='auto', imp_dim='auto', measure=None,
            exp_concept_level='c', filesystem='auto', grid='-', imp_concept_level='c', import_metadata='n
            ioserver='mysql_table', ncores=1, ndb=1, ndbms=1, nfrag=0, nhost=0, subset_dims='none', subse
            subset_type='index', exec_mode='sync', base_time='1900-01-01 00:00:00', calendar='standard',
            leap_year=0, month_lengths='31,28,31,30,31,30,31,31,30,31,30,31', run='yes', units='d', vocab
            pid=None, check_grid='no', display=False) -> obj
        or Cube(pid=None) -> obj

    Attributes:
        pid: cube PID
        creation_date: creation date of the cube
        measure: name of the variable imported into the cube
        measure_type: measure data type
        level: number of operations between the original imported cube and the actual cube
        nfragments: total number of fragments
        source_file: parent of the actual cube
        hostxcube: number of hosts associated with the cube
        dbmsxhost: number of DBMS instances on each host
        dbxdbms: number of databases for each DBMS
        fragxdb: number of fragments for each database
        rowsxfrag: number of rows for each fragment
        elementsxrow: number of elements for each row
        compressed: 'yes' for a compressed cube, 'no' otherwise
        size: size of the cube
        nelements: total number of elements
        dim_info: list of dict with information on each cube dimension

    Class Attributes:
        client: instance of class Client through which it is possible to submit all requests
```



https://pypi.org/project/PyOphidia/
https://anaconda.org/conda-forge/pyophidia

Ophidia architecture: framework layer

- ✓ *The Ophidia analytics framework can be executed with multiple processes/ threads*
- ✓ *Provides the environment for the execution of parallel MPI/OpenMP-based **operators***
- ✓ *Operators manipulate the entire set of fragments associated to a datacube*

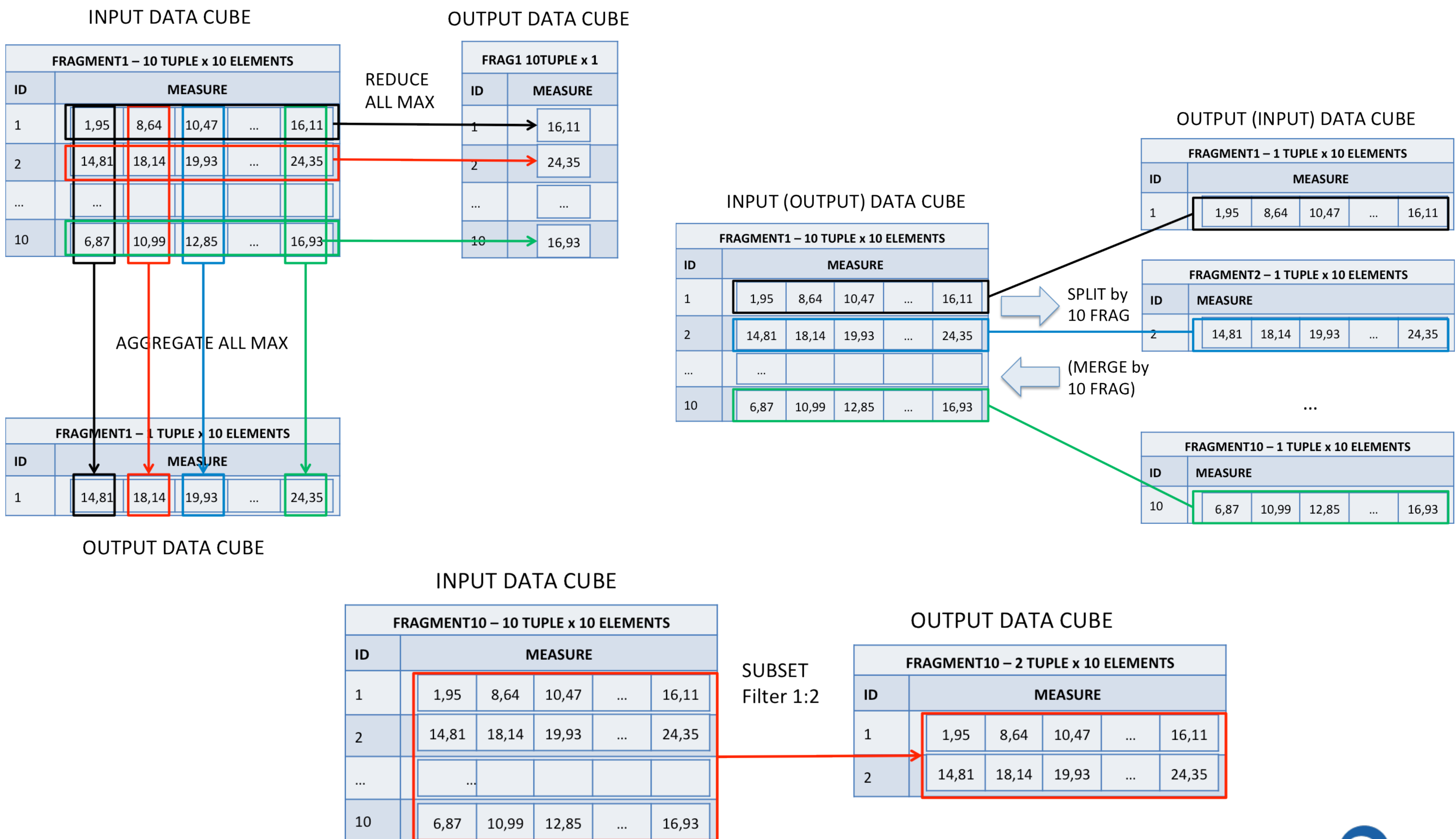


The Ophidia operators

CLASS	PROCESSING TYPE	OPERATOR(S)
I/O	Parallel	OPH_IMPORTNC, OPH_IMPORTFITS, OPH_EXPORTNC, OPH_CONCATNC, OPH_RANDUCUBE
Time series processing	Parallel	OPH_APPLY
Datacube reduction	Parallel	OPH_REDUCE, OPH_REDUCE2, OPH_AGGREGATE
Datacube subsetting	Parallel	OPH_SUBSET
Datacube combination	Parallel	OPH_INTERCUBE, OPH_MERGECUBES
Datacube structure manipulation	Parallel	OPH_SPLIT, OPH_MERGE, OPH_ROLLUP, OPH_DRILLDOWN, OPH_PERMUTE
Datacube/file system management	Sequential	OPH_DELETE, OPH_FOLDER, OPH_FS
Metadata management	Sequential	OPH_METADATA, OPH_CUBEIO, OPH_CUBESCHEMA
Datacube exploration	Sequential	OPH_EXPLORECUBE, OPH_EXPLORENC

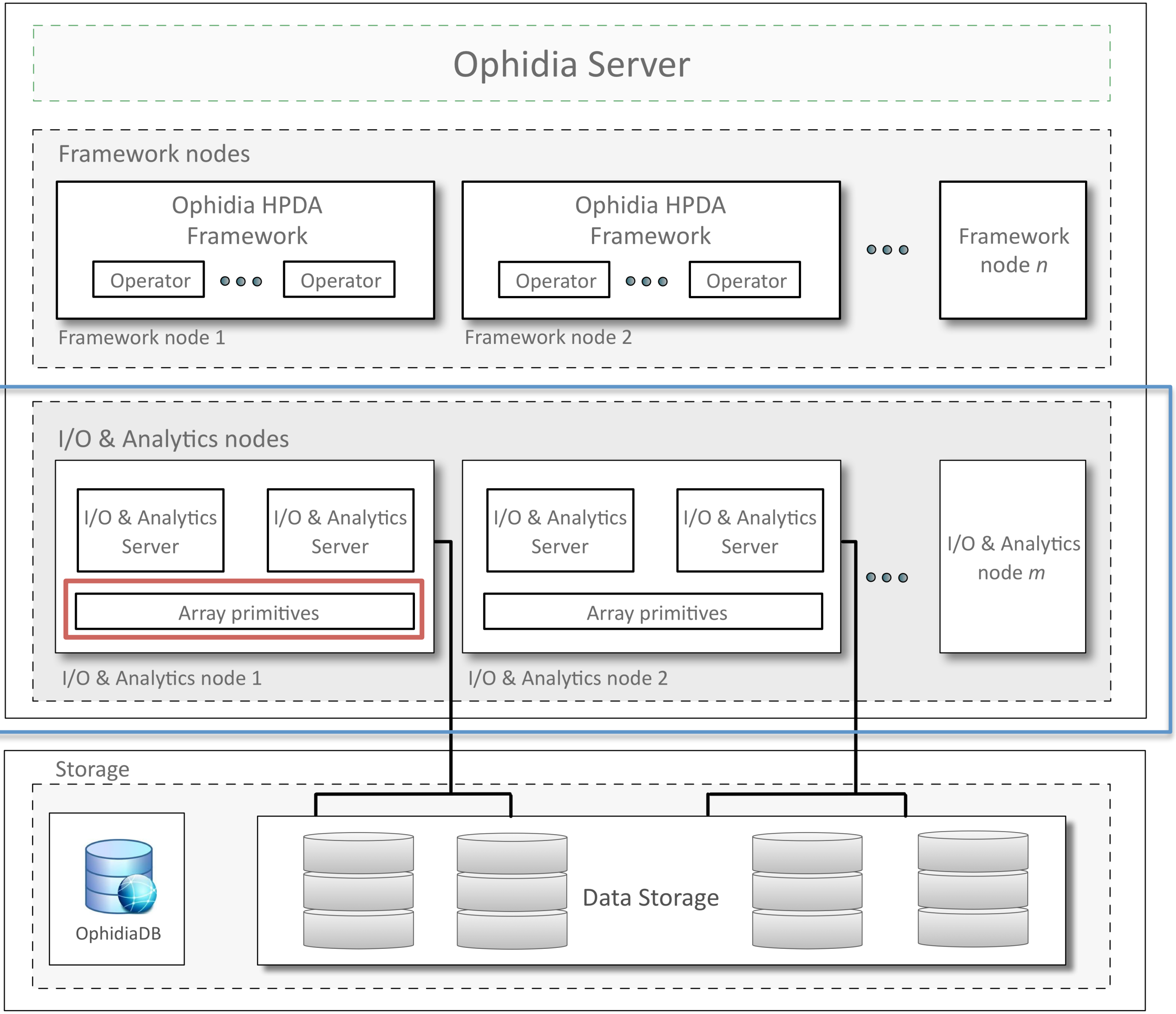
About 50 operators for data and metadata processing

The “data” operators



Ophidia architecture: I/O & analytics layer

- ✓ *Multiple* ***I/O & analytics nodes*** *execute one or more servers*
- ✓ *Servers run the array-based* ***primitives*** *(UDF)*
- ✓ *Server engine can transparently interface to different storage back-ends*
- ✓ *Support for a native in-memory array-based analytics & I/O engine*



Array-based primitives

- ✓ Ophidia provides a **wide set of array-based primitives** (around 100) to perform:
  - ✓ data summarization, sub-setting, predicates evaluation, statistical analysis, array concatenation, algebraic expression, regression, etc.
- ✓ Bit-oriented plugins have also been implemented to manage binary datacubes
- ✓ Primitives come as plugins and are applied on a single datacube chunk (fragment)
- ✓ **Primitives can be nested** to get more complex functionalities

oph_boxplot(oph_subarray(oph_uncompress(measure), 1,18), "OPH_DOUBLE")

Single chunk or fragment (input)

INPUTTABLE 5 tuples x 50 elements

ID	MEASURE									
1	10,73	8,66	7,83	11,20	6,02	1,95	...	16,11	...	8,70
2	22,85	17,84	21,82	18,57	14,81	18,71	...	19,83	...	21,13
3	19,89	30,17	24,95	30,07	25,40	26,31	...	23,18	...	24,82
4	11,60	12,49	13,91	13,53	9,48	15,27	...	14,17	...	11,66
5	13,94	12,43	17,95	14,70	20,41	14,46	...	18,00	...	18,30

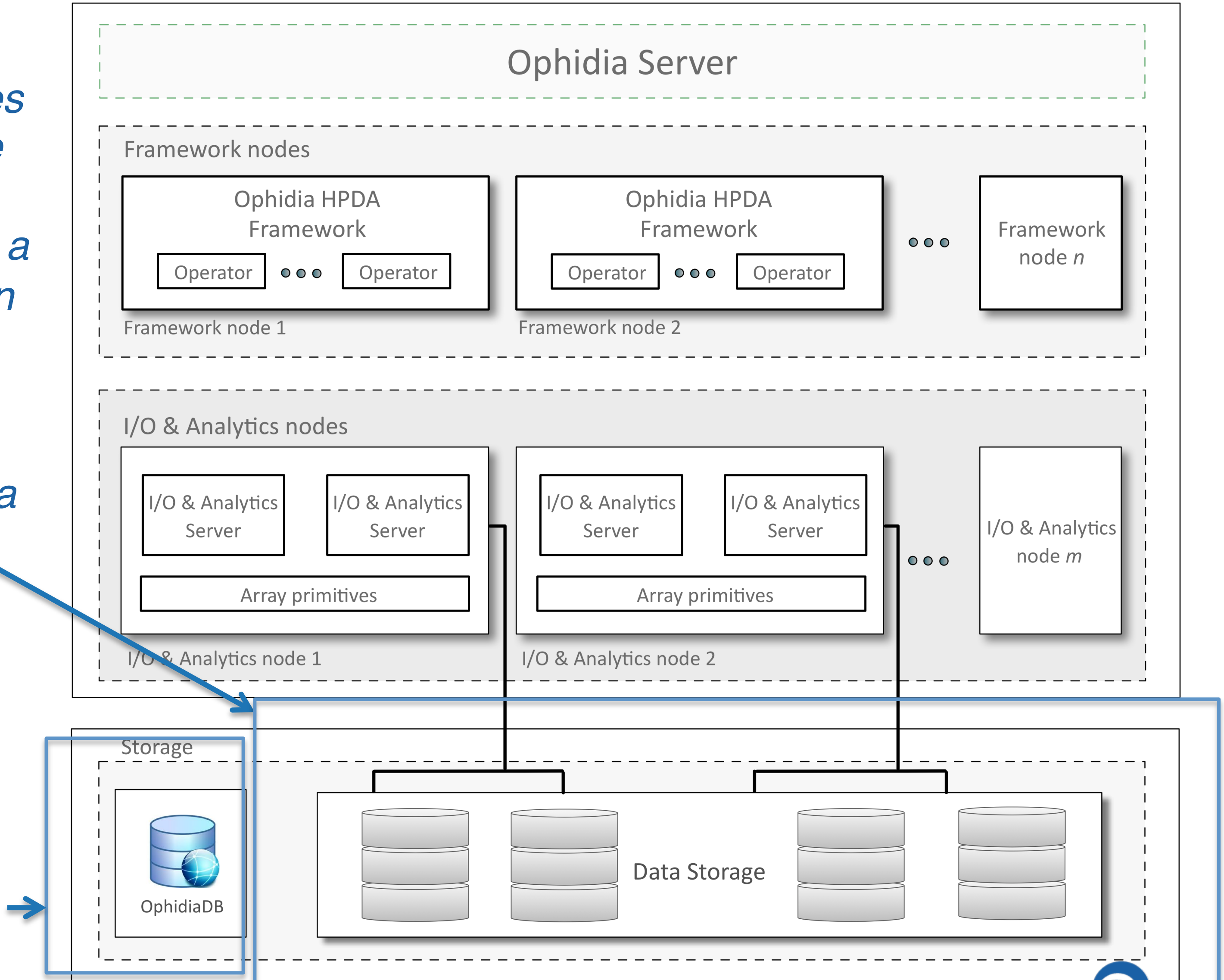
Single chunk or fragment (output)

OUTPUTTABLE 5 tuples x 5 elements (summary)

ID	MEASURE				
1	1,95	8,64	10,47	11,87	16,11
2	14,81	18,14	19,93	21,66	24,35
3	19,89	22,74	24,24	26,45	30,17
4	6,87	10,99	12,85	14,28	16,93
5	9,23	13,87	15,05	16,61	20,41

Ophidia architecture: storage layer

- ✓ *Distributed hardware resources to manage storage*
- ✓ *Data partitioned in a hierarchical fashion over the storage according to the storage model & partitioning schema*
- ✓ ***OphidiaDB** is the system catalog: maps data fragmentation and tracks metadata*



Performance evaluation

Evaluation of scalability of a core and one of the most used Ophidia operators with the *in-memory server*:

- ✓ compute parallel *data reduction* over a datacube (up to 1TB):
  - ✓ average value of the time series, for each point in a 3D spatial domain (lat, lon, height)
- ✓ all values are averaged across multiple run (with a 95% confidence interval whose maximum relative error is at most 7%)

In Ophidia most data operators are executed in a similar fashion

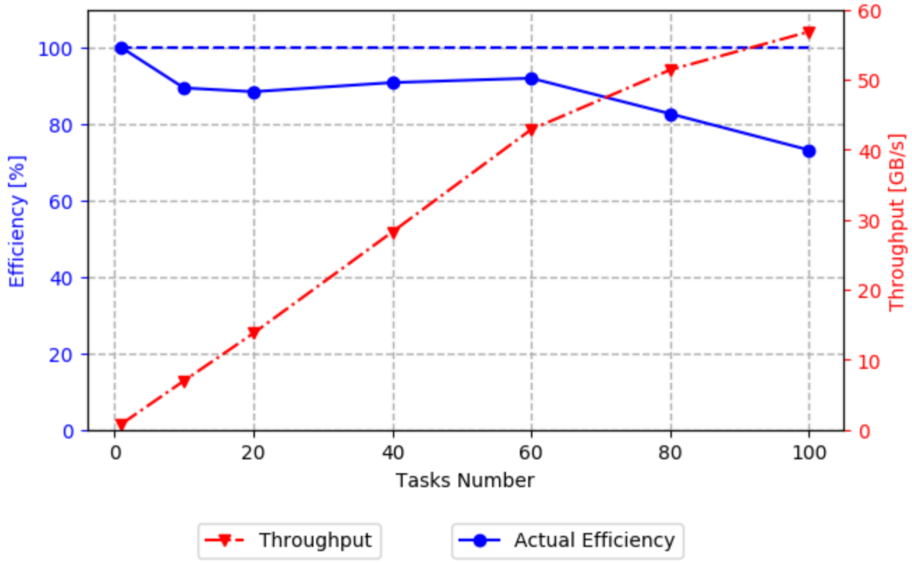
Benchmark executed on a cluster dedicated for in-memory analytics setup @ CMCC SuperComputing Centre

Test environment specs	
Number of nodes	5
RAM	1.3TB (256GB/node)
Number of cores	100 (2x10 cores/node - Intel Xeon CPU)
Storage size	60TB shared storage (GlusterFS)
Network	10Gb/s dedicate network
Ophidia deployment	an instance of a I/O & analytics server/node

Experimental results: strong scalability

Evaluate scalability by measuring the OPH_REDUCE2 execution time on a fixed problem size while increasing the number of executed parallel tasks

- ✓ datacube size about 1TB (270×10^9 floating point, organized into 23×10^6 time series of 11.7×10^3 elements each)
- ✓ data partitioned into 1200 fragments evenly distributed over the 5 I/O & analytics servers (200GB of data/node)



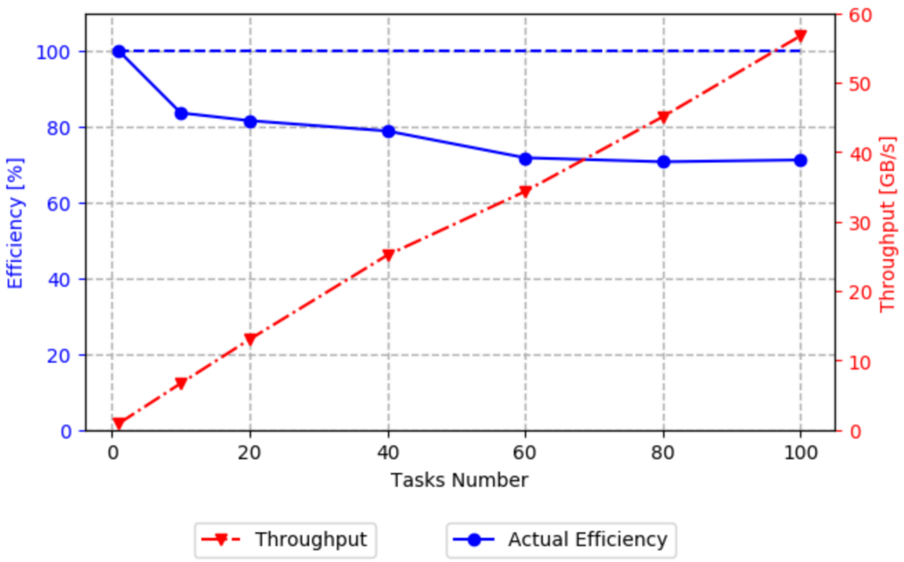
Tasks number	Eexecution time [s]	Efficiency [%]	Throughput [GB/s]
1	1290.8	100	0.8
10	144.3	89.4	6.9
20	73	88.5	13.7
40	35.5	90.8	28.2
60	23.4	91.8	42.9
80	19.5	82.7	51.4
100	17.6	73.2	56.8

Partitioning schema allows to effectively scale up with the data size over multiple nodes

Experimental results: weak scalability

Evaluate scalability by measuring OPH_REDUCE2 execution time while scaling up the data size along with the number of parallel tasks

- ✓ the number of fragments/task is fixed to 1 (20 frags/I/O & analytics servers)
- ✓ Each fragment contains about 2.8×10^9 floating point values organized into 240×10^3 time series of 11.7×10^3 elements each for a total of 10.4GB of data



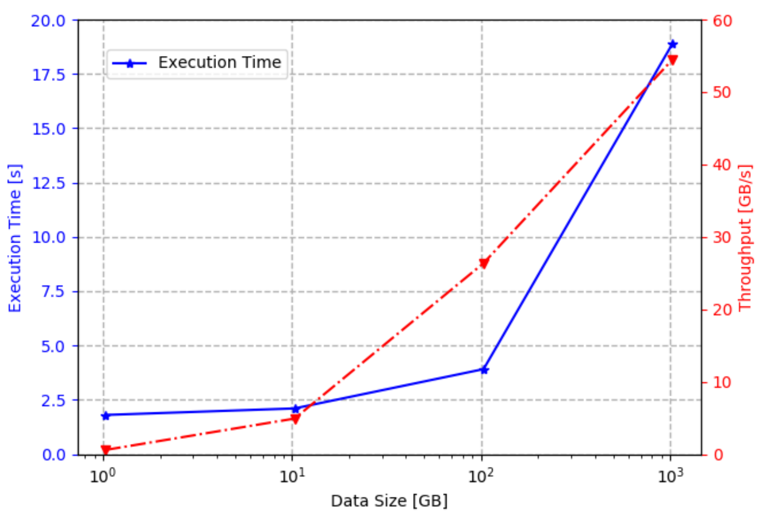
Tasks number	Nodes number	Execution time [s]	Efficiency [%]	Throughput [GB/s]	Data Size [GB]
1	1	13.1	100	0.8	10.4
10	1	15.7	83.6	6.7	104.4
20	1	16.1	81.6	13.0	208.9
40	2	16.6	78.8	25.1	417.7
60	3	18.3	71.8	34.4	626.6
80	4	18.5	70.7	45.1	835.4
100	5	18.4	71.2	56.7	1044.3

Storage model implementation allows good level of scalability over multiple nodes (efficiency does not degrade as more resources are added)

Experimental results: array-oriented tests

Evaluate scalability by measuring OPH_REDUCE2 execution time while increasing the array length, with fixed data partitioning and number of tasks

- ✓ the data is split into 100 fragments evenly distributed over 5 I/O & Analytics servers and 100 parallel tasks are always used (i.e. 1 frag/task)
- ✓ Each fragment consists of 230 x 10^3 time series each, with increasing length (one order of magnitude each time)



Array length	Execution time [s]	Throughput [GB/s]	Data Size [GB]
12	1.8	0.6	1
120	2.1	4.9	10.3
1200	3.9	26.4	103
12000	18.9	54.5	1030

The array-oriented physical data organization proves to be extremely efficient in the management of (very) long time series

Summary & future activities

Recap

- ✓ Ophidia provides a *HPDA framework* joining *HPC* paradigms with *scientific data analysis* approaches for parallel data analytics
- ✓ Implements a *multi-dimensional storage model* where data is *partitioned* and *hierarchically distributed*
- ✓ Experimental results show how the Ophidia data distribution and partitioning enable the operator to scale up to the full capacity of our cluster

Future activities

- ✓ Large-scale benchmark on Marenostrum (PRACE Tier0 machine at Barcelona Supercomputing Center) in the context of the ESiWACE projects
- ✓ Further extension of Ophidia to support the Earth System Data Middleware interface, developed in the ESiWACE projects

Thanks

http://ophidia.cmcc.it

@OphidiaBigData

www.youtube.com/user/OphidiaBigData

https://github.com/OphidiaBigData

ophidia-info at cmcc.it