Observational DataBase (ODB) and its usage at ECMWF

anne.fouilloux@ecmwf.int

Satellite Data Section, ECMWF

Outline

- **Part-I: ODB Overview**
  - Introduction
  - Data partitioning
  - ODB I/O method
  - ODB/SQL
  - Fortran 90 interface to ODB
  - ODB-tools
  - Visualisation of ODB with Metview
- **Part-II: ODB and its usage in IFS at ECMWF**
  - ODB interface for IFS
  - ECMA/CCMA data layout
  - Observational arrays in IFS
  - Parallelisation with MPI/OpenMP
  - Observational data flow
  - ODB-tools for IFS: bufr2odb, odbshuffle, matchup, revmatchup
- **Conclusion and future developments**

PART-I :
ODB Overview

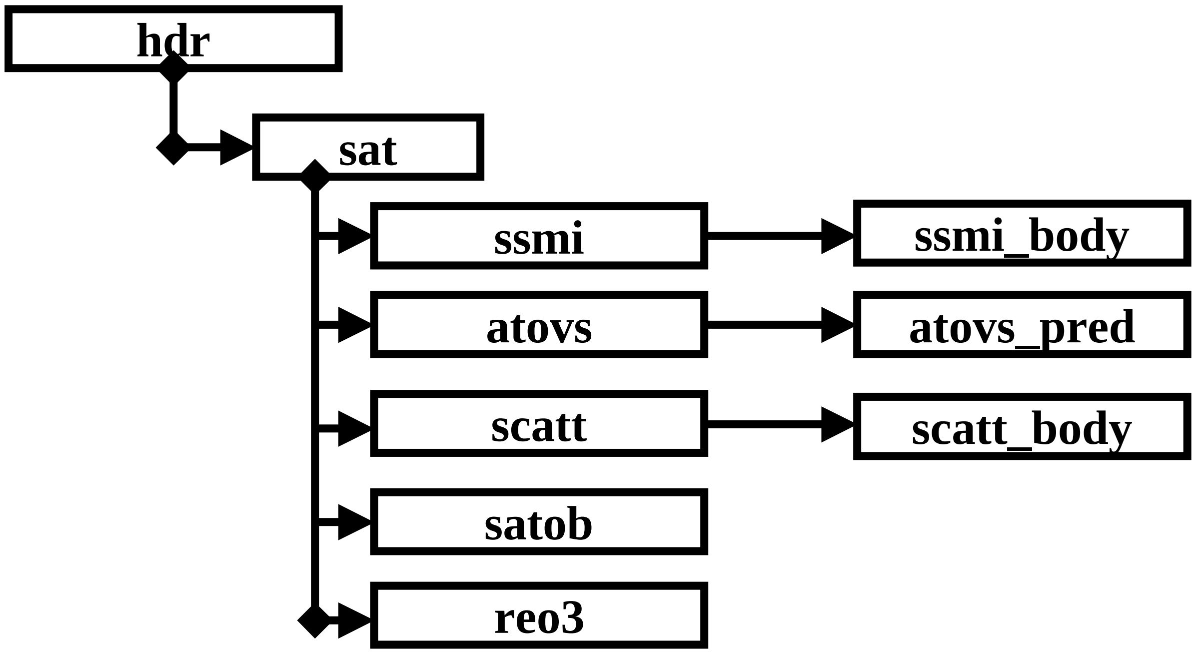
Introduction to ODB

- **ODB stands for Observational DataBase and is a tailor made software developed at ECMWF by Sami Saarinen to manage very large observational data volumes through the 4DVAR-system on highly parallel supercomputer systems. ODB has been developed with the following requirements:**
  - Fortran interface (IFS/ARPEGE is written in Fortran)
  - Suitable for MPI/OpenMP parallelisation
  - Perform efficient data extraction in our 4D-var (achieved via ODB/SQL)
- **ODB has been operational at ECMWF since 27th of June 2000**
- **ODB is also used at MeteoFrance through IFS/ARPEGE collaboration and has spread through their Aladin-collaboration…**
- **ODB is used in Australian Bureau of Meteorology, Melbourne**

ECMWF

ODB hierarchical data model

- **In ODB, data is organized into a *tree-like* structure. The structure allows “repeating” information using parent/child relationships: each parent can have many children but each child only has one parent.**



- **A table can be seen as a matrice (2D-array or so called flat file) with a number of rows and columns containing numerical data.**

Data Definition Layout (DDL)

- **This hierarchy is described in the Data Definition Layout (or schema) file.**
  - **Text file consisting of a number of named TABLEs**
  - **Each TABLE has got a number of named *columns* (or *attributes*)**
  - **Each column in turn has got a *specific type***
    - integer/ real/ string
    - packed,
    - bitfield type (can vary between 1 an 32 bits, access `column_name.bitfield_name`)
    - @LINK to define connections between TABLEs

```
CREATE TABLE table_name AS (
  column_name1 data_type1,
  column_name2 data_type2,
  column_name3 data_type3,
  ....
);
```

```
CREATE TYPE type_name AS (
    bitfield_name1 data_type1,
    bitfield_name2 data_type2,
    bitfield_name3 data_type3,
    ....
);
```


ECMWF

Example of ODB DDL file

```
CREATE TABLE hdr AS (
lat real,
lon real,
statid string,
obstype int,
date YYYYMMDD,
time HHMMSS,
status flags_ t,
body @LINK,
);
CREATE TABLE body AS (
varno pk5int,
press pk9real,
obsvalue pk9real,
);
```

lat	lon	statid	obstype	date	time	status
-14.78	143.5	' 94187'	1	20081021	230000	1

@LINK

varno	press	obsvalue
1	100350	804.14
30	100100	120
39	99900	277.6
40	100350	292.4
58	100350	0.57
111	100840	260
112	100100	2
41	97670	12.9
42	95310	-4.84e-15
80	100880	0

A *LINK* tells how many times a row needs to be repeated (10 times in our example) and which table is involved (body)

standard data type

column name or attribute

built-in date & time types

packed data type

composite data type (bit-field)

LINK data type

Data partitioning

- **The main purpose is to allow parallelism (requirement for usage in IFS model):**
  - → divide TABLEs “horizontally” into pools between processors; pools are assigned to the MPI-tasks in a round-robin fashion (max. PEs <= max. no. of pools). By default, an MPI-task cannot modify data on a pool that it does not own.
  - → each table can be assigned to an openMP threads
- **no. of pools "decided" in the Fortran90 layer**
- **SELECT data from *all* or a *particular* pool only**
- **How to distribute data?**
  - → latitude- bands, or time slots, or obs. types or due to load balancing etc.
  - → Distribution is done in bufr2odb in IFS for ECMA (pools done per obs. group). It is done again when creating CCMA from ECMA i.e. when creating a new database with active data only.

Table partitioning – example with 3 pools

Table **hdr**

Table **body**

Pool#1

lat	lon	statid	obstype	date	time	status
-14.78	143.5	' 94187'	1	20081021	230000	1

@LINK

varno	press	obsvalue
1	100350	804.14
30	100100	120
39	99900	277.6

Pool#2

lat	lon	statid	obstype	date	time	status
-14.78	143.5	' 94187'	1	20081021	230000	1

@LINK

varno	press	obsvalue
40	100350	292.4
58	100350	0.57
111	100840	260

Pool#3

lat	lon	statid	obstype	date	time	status
-14.78	143.5	' 94187'	1	20081021	230000	1

@LINK

varno	press	obsvalue
112	100100	2
41	97670	12.9
42	95310	-4.84e-15
80	100880	0

- **The first row in *hdr* is repeated in each pool. A single pool forms a 'sub-database'.**

ODB I/O method – ODB_IO_METHOD

- **ODB currently support 5 I/O methods which controls how the data is read/write from/to disk:**
  - **1** - Creates one file per every TABLE on a pool basis. Uses the CMA I/O-routines with the standard C I/O-library (i.e. fopen, fread, fwrite and fclose). Default value at Météo-France.
  - **2** - The same as method#1, but using system I/O-routines (read and fwrite) directly. *Not very well tested.*
  - **3** - qtar method, where an external ODB-specific utility (similar to tar) is invoked to store and extract data. One QTAR-file per pool is created i.e. all TABLEs will be saved into a single file on a pool basis. *Not very well tested.*
  - **4** - In this method each similar TABLE-file for a number of consecutive pools (ODB_IO_GRPSIZE) are concatenated together to achieve the maximum configured filesize given via ODB_IO_FILESIZE. Default value in ECMWF scripts from IFS cycle CY26R1 onwards. Information from the adjacent data pools are message passed to the nearest I/O-task for performing the I/O
  - **5** – Read/only method. It uses dca (Direct Column Access) files (dcagen –F –n –q –z). This will give a boost for data accesses and reduces memory consumption.

ODB/SQL Statements

```
[CREATE VIEW view_name AS]
SELECT [DISTINCT] column_ name( s)
FROM table( s)
[WHERE some_ condition( s)_ to_ be_ met ]
[ORDERBY sort_ column_ name( s) [ASC/ DESC] ]
```

- **ODB/SQL(*) is a small subset of international standard SQL used to manipulate relational databases.**
- **It allows to define data queries in order retrieve (normally) a subset of data items. This is the "main" motivation of using ODB ?!**
- **Except for the creation of a database or within IFS/ARPEGE where a Fortran program is necessary, ODB/SQL can be used in an interactive way via ODB-tools (odbviewer, odbsql, etc.).**

(*)SQL stands for Structured Query Language

ODB/SQL examples

- Find distinct values of obstype and sort them in DESCending order:

```
SELECT DISTINCT obstype
FROM hdr
ORDERBY obstype DESC ;
```

- Provide the following radio-sonde temperatures :

```
SELECT lat,lon,press,obsvalue
FROM hdr, body
WHERE obstype=$temp AND varno=$t
AND lldegrees(lon) BETWEEN 100W AND 80W
AND press < 500hPa ;
```

ODB/SQL – SET variables

- **Parameters are variables that start with $ and store numbers (integers or floating point values)**
- **For example:**

```
SET $temp = 5;
SET $t = 2;
```

- **This can be used to generalize certain kinds of queries (so-called *parameterized SQL-queries*)**
- **There are also useful when creating multiple columns or tables with (nearly) the same meaning**

```
SET $nmxupd = 3;
CREATE TABLE update[1:$nmxupd] AS (…);
```

- **These variables can also be some state variables, whose value can be changed on a permanent or temporary basis from Fortran.**

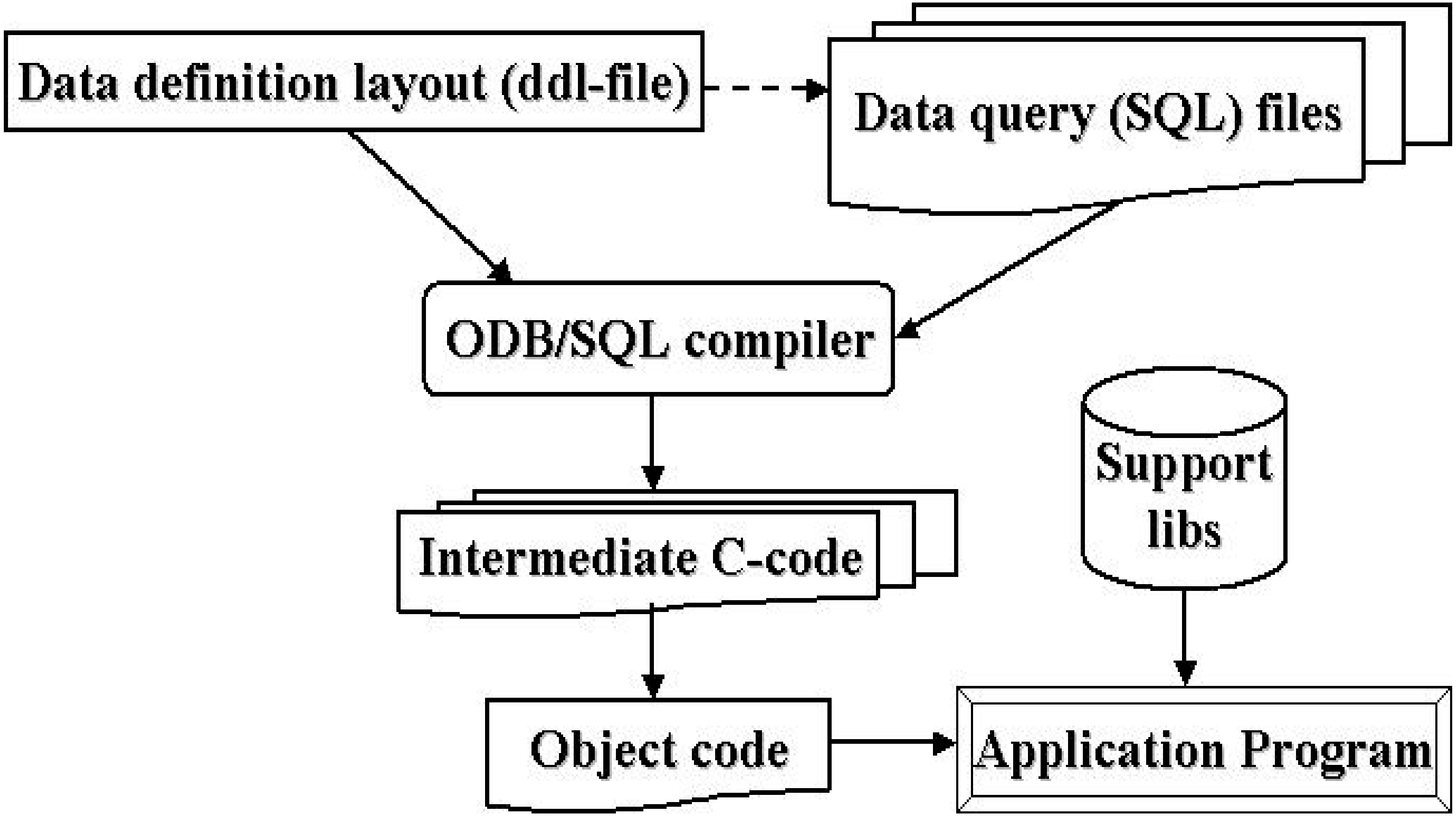
Fortran 90 interface to ODB/SQL

- ODB Fortran90 interface layer offers a comprehensive set of functions to
  - Open & close database
  - Attach to & execute precompiled ODB/SQL queries
  - Load, ***update*** & store queried data
  - Inquire information about database metadata
- Fortran90 interface of ODB can use Message Passing Interface (MPI) for **parallel** data queries.
- SELECT‘ ed data can be asked to be **shuffled** (“ part- exchanged”) or **replicated** across processors (**`ODB_select`**); by default data selection applies to the **local pools** only.
- Each query needs to be *pre-compiled/linked* with the main user program.
- Parameterized queries can be used.

An example of Fortran program with ODB

```
program main
use odb_module
implicit none
integer(4) :: h, rc, nra, nrows, ncols, npools, j, jp
real(8), allocatable:: x(:,:)
npools= 0
h = ODB_open("MYDB", "OLD", npools=npools)
DO jp=1,npools
   rc= ODB_select(h, "sqlview",nrows,ncols,poolno=jp)
   allocate(x(nrows,0:ncols))
   rc= ODB_get(h, "sqlview",x,nrows,ncols,poolno=jp)
   call update(x,nrows,ncols) ! Not an ODB-routine
   rc= ODB_put(h, "sqlview",x,nrows,ncols,poolno=jp)
   deallocate(x)
   rc= ODB_cancel(h, "sqlview",poolno=jp)
ENDDO
rc= ODB_close(h, save=.TRUE.)
end program main
```

ODB/SQL compilation system



Compile, link and run a Fortran program

```
[1] use odb              # once per session

[2] odbcomp MYDB.ddl  # once only;often from file MYDB.sch

[3] odbcomp –lMYDB sqlview.sql # recompile when changed

[4] odbf90 main.F90 update_data.F90 –lMYDB –o main.x

[5] ./main.x

[6] Go back to [3]
```

Note: [1] – [2] is not required for precompiled ODB databases (such as ECMA,CCMA)

ODB Tools

- Various ODB-tools are meant to simplify browsing and management of ODB databases.
- Some are generic and can be used with any ODB databases (no compiled queries or databases):
  - `odbsql`: a tool to access ODB data in read/only mode
  - `odbdiff`: a tool to compare two ODB databases
  - `odbdup`/`odbmerge`: to combine several databases
  - `odbcompress`: to create a sub-ODBs from an existing database
  - `simulobs2odb`: to create a new ODB from an ascii file
  - `odbviewer`: ODB visualization and text result browsing. Only available when ODB is built with Magics/Magics++.
  - `odb1to4` and `odb4to1`: convert from one I/O method to another
- Some are specific to IFS/ARPEGE usage (`bufr2odb`, `odb2bufr`, `odbshuffle`, `matchup`, `revmatchup`, etc.); See part-II.

odbsql

- **A tool to access ODB data in read/only –mode (`ODB_IO_METHOD=5`)**
  - **Does not generate C-code, but dives directly into data**
  - **It uses dca files (direct column access) which can be created with dcagen**
- **Usage:**

```
odbsql –v query.sql| -q “SELECT…” –s starting_row   \
       –n number_of_rows_to_display \
       -f output_format  -I dir_db  \
      [–X] [other_options]
```

- **For example:**

```
odbsql –q ‘SELECT lat,lon,fg_depar from hdr,body’ \
       -i /dir1/CCMA
```

odbdiff

- **Enables comparison of two ODB databases for differences**
- **A very useful tool when trying to identify errors/differences between operational and experimental 4DVAR runs**

```
odbdiff –v query.sql|-q ‘query_string’ \
       -p poolmask [other_options] ref_base comp_base
```

- **For example:**

```
odbdiff –q ‘SELECT lat,lon,fg_depar from hdr,body’ \
           /dir1/CCMA /dir2/CCMA
```

- **By default the command brings up an *xdiff*-window with respect to differences**
- **If *latitude* and *longitude* were also given in the data query, then it also produces a difference plot using *odbviewer*-tool**

odbcompress

- **Enables to create very compact databases from the existing ones**

```
odbcompress –i indput_db –o output_db \
            –l ddl_file [-1|-4]
```

- **Makes post-processing considerably faster**
- **The user can choose to**
  - **Truncate the data precision, and/or**
  - **Leave out columns that are less of an importance**

odbdup/odbmerge

- **Allows f.ex. database sharing between multiple users**
  - **Over shared (e.g. NFS, Lustre, GPFS, GFS) disks**
- **Duplicates [merges] database(s) by copying metadata (low in volume), but shares the actual (high volume) binary data**
- **Also enables creation of *time-series* database**

```
odbmerge –i indput_db –o output_db –l dbname
```

- **for example:** `odbmerge –i "200701*/ECMA.conv" –o USERDB`
- **The previous example creates a new database labelled as USERDB, which presumably spans over the all conventional observations during the January 2007**
  - **The *main point* : user has now access to whole month of data as if it was a single database !!**

simulobs2odb

- **simulobs2odb allows to load an ODB database directly from a text file. This can be a useful option when developing software or loading own databases and BUFR-definitions (for example) are not yet fixed.**

```
simulobs2odb [-l dbname] [-i file] [-n npools] \
             [-c] [-r rptfile] [-1|-4]
```

- **For instance:**

```
simulobs2odb –i hdr.txt –i body.txt –l USERDB
```

where `USERDB.ddl` is a user defined schema file.

- **It can also be used to create a new “mini” ODB**

```
simulobs2odb –r file.rpt –l USERDB
```

Here, there is no need to describe the schema file (done automatically from the report file)

odbviewer

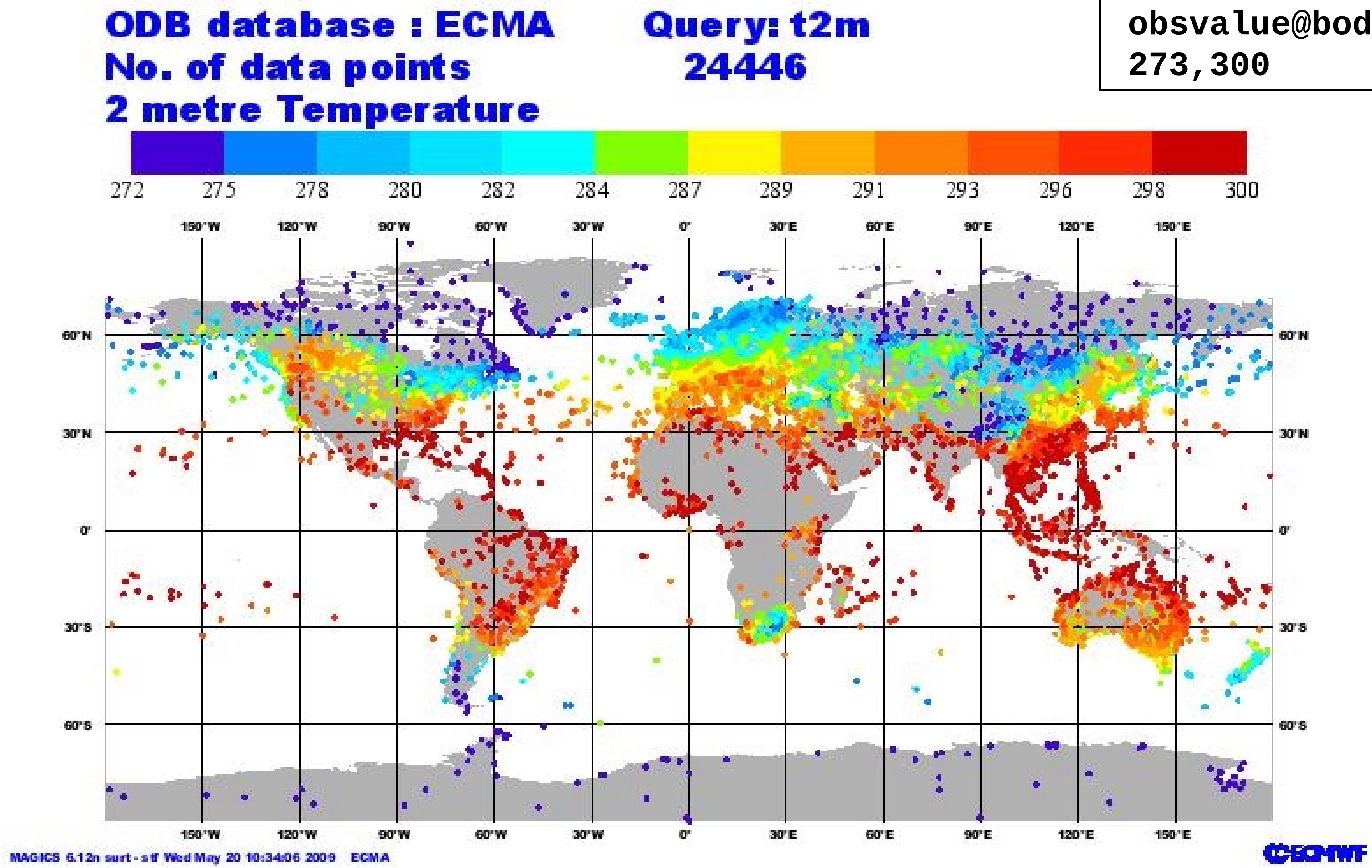
- **A very basic ODB data examination tool linked with ECMWF graphics package MAGICS/MAGICS++**
- **Executes given ODB/SQL-queries and tries to produce both *coverage plot* if (lat,lon) is available and *textual report* (ASCII-format)**
- **Example:**

```
// 2m Temperature – t2m.sql
SET $t2m = 39;
SET $synop = 1;
CREATE VIEW t2m AS
SELECT an_depar, fg_depar, lat, lon, obsvalue
FROM hdr, body
WHERE obstype = $synop    // Give me synops
AND varno = $t2m     // Give me 2 meter temperatures
AND obsvalue is not NULL ; // Don’t want missing data
```

2m temperature

color.cmap

```
2 m Temperature
obsvalue@body
273,300
```



odbviewer -v t2m.sql -i ECMA –C color.cmap

ECMWF

Visualization of ODB with Metview

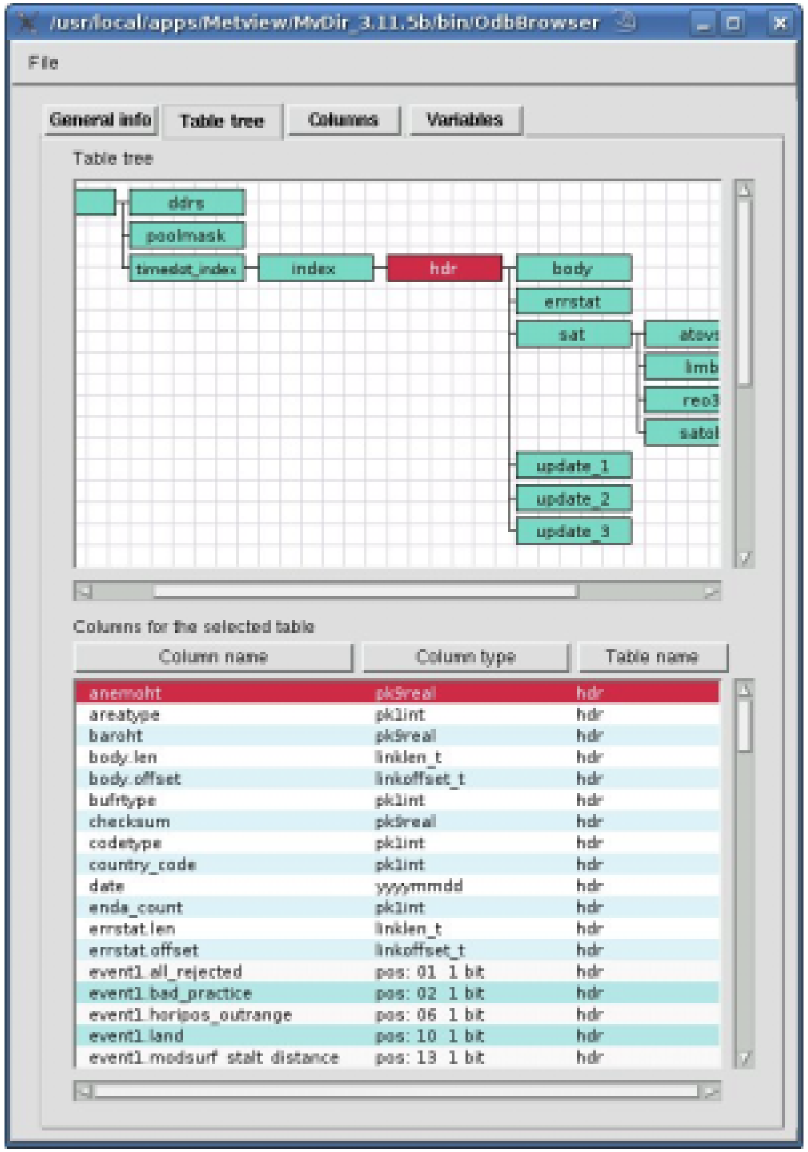
- Uses ODB API (part of ODB package)
  - C interface to access ODB databases in read-only mode
  - Direct or Client/server Access
- ODB Database icon 

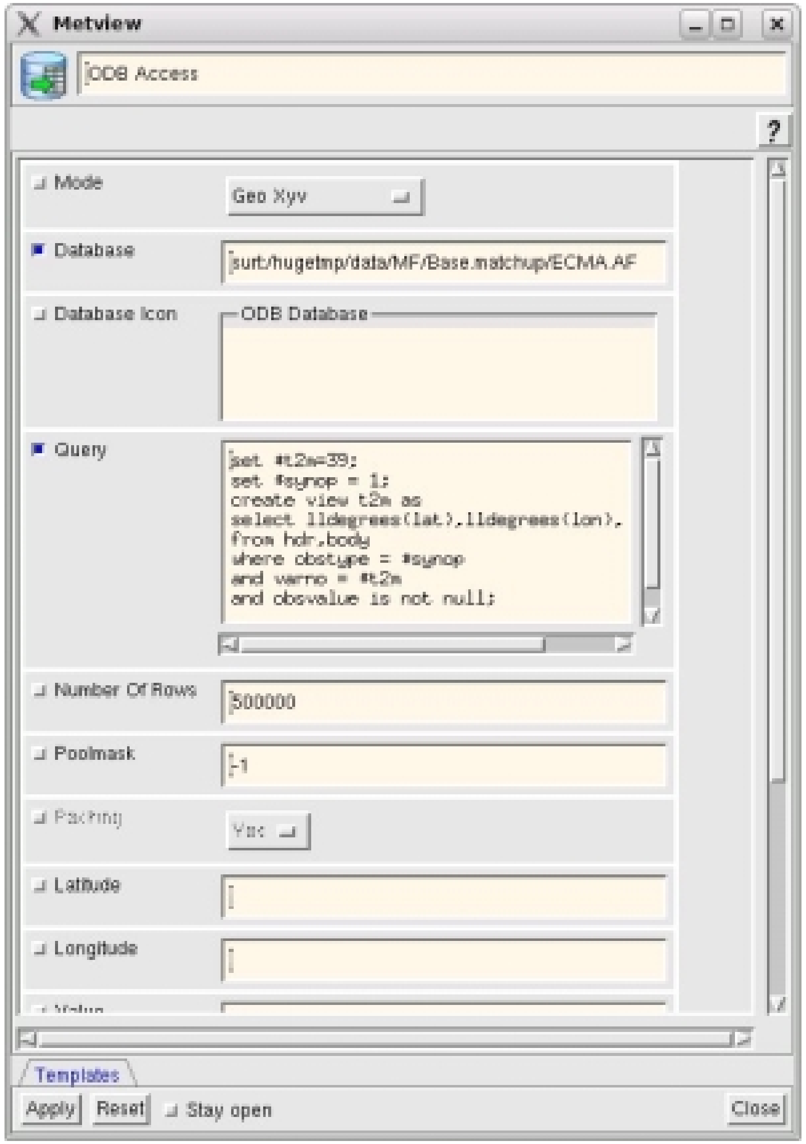
  - to specify the ODB database path and name
  - to browse the metadata contents
- ODB Access icon 

  - Defines the ODB/SQL query
  - Output in Geopoints format (geopoints visualisation)
- GeoTools icon 

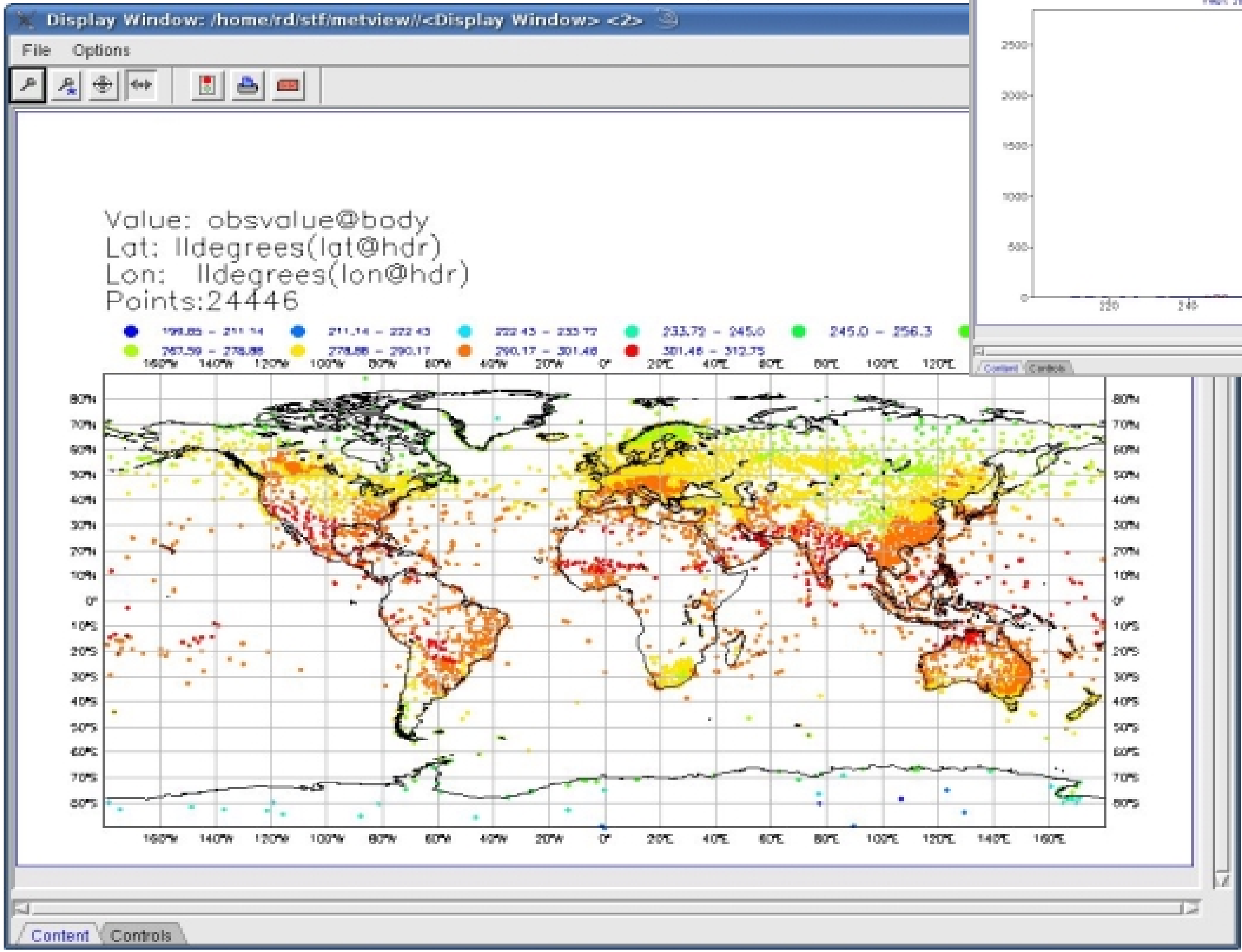
  - Preview and Histogram
  - Temporary tool until Metview 4 is available
- *This version of Metview is not available to member states yet*

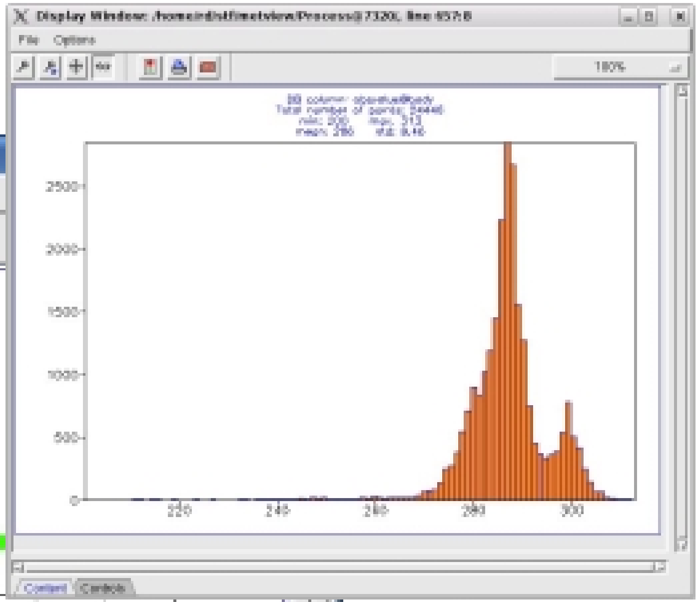
ODB Browser and ODB Access Examples





GeoTool example





ECMWF

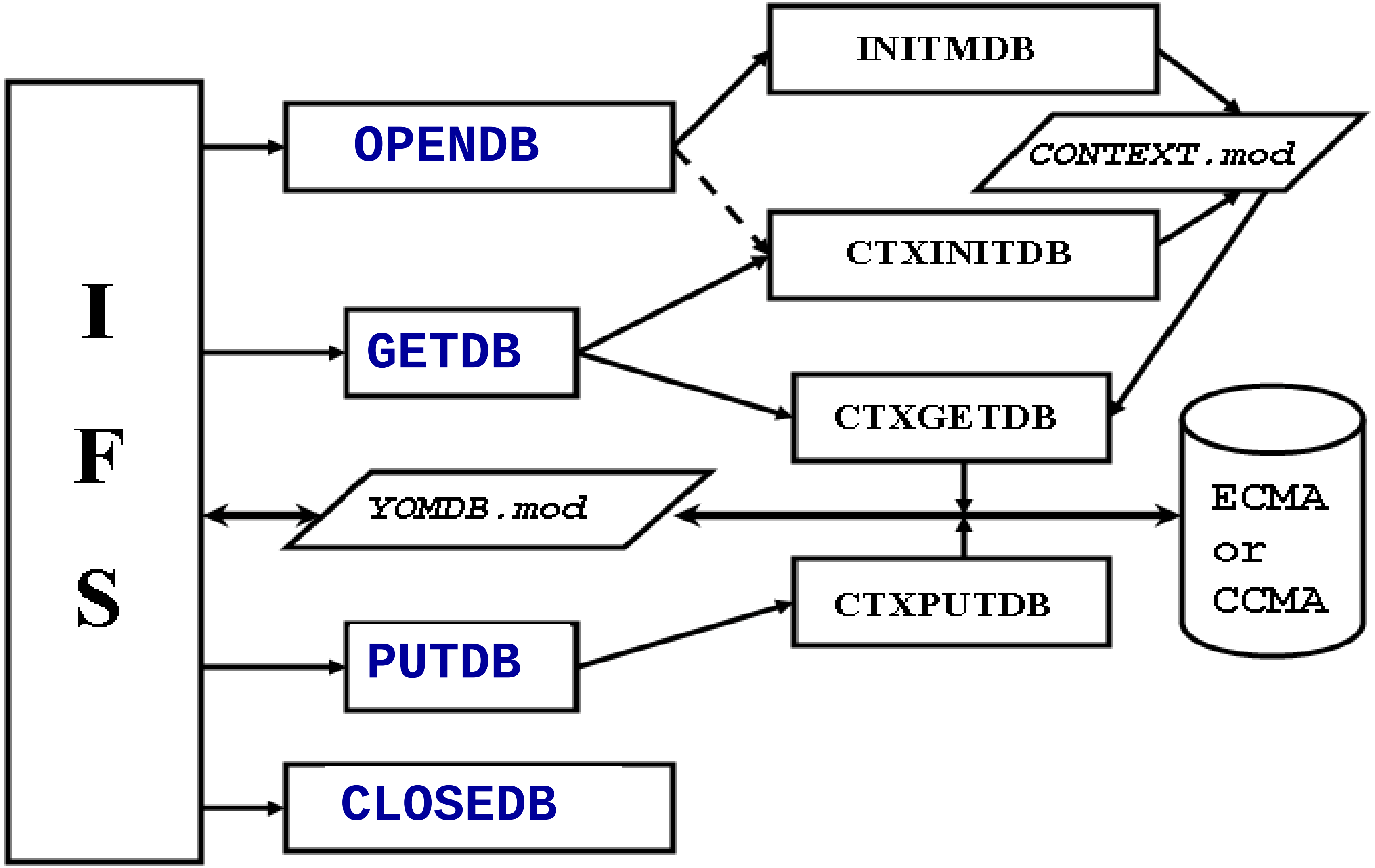
PART-II : ODB and its usage at ECMWF in IFS

ODB interface for IFS

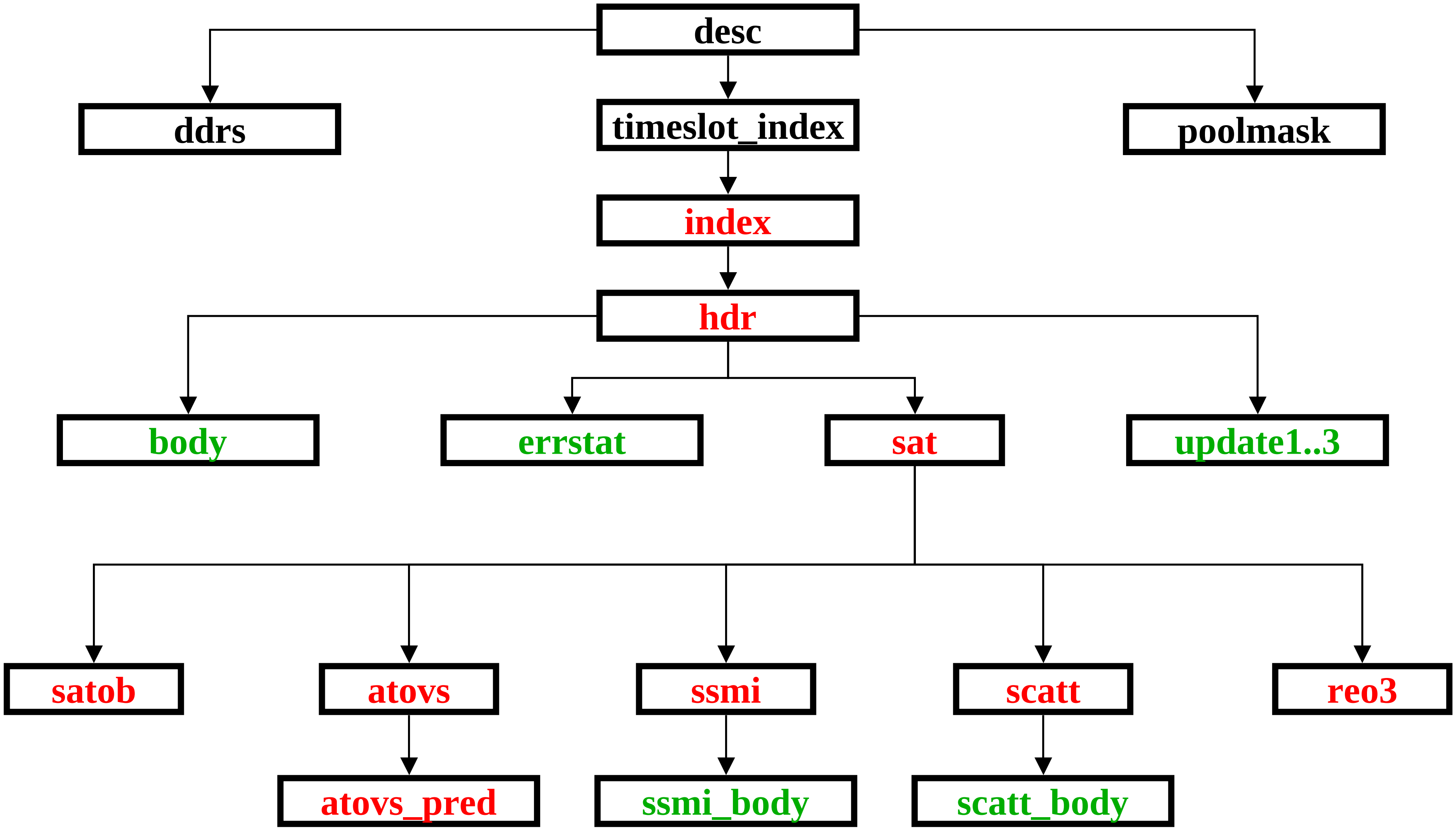
- The ODB/IFS interface is a high-level interface to ODB which mainly applies to ECMA and CCMA databases
  - ECMA contains all observations before the screening
  - CCMA contains only active observations
- OPENDB
  - Opens ECMA/CCMA databases
- GETDB
  - Executes one or more SQL queries (as defined in CTXINITDB of odb/cma2odb/ctxinitdb.F90) via routine CTXGETDB
  - Calls ODB_select, allocates matrices ROBHDR, ROBODY etc. and then calls ODB_get to fill out the observational matrices
- PUTDB
  - Returns the contents of the updated matrices back to (in-memory) database data structures via routine CTXPUTDB :
  - Calls ODB_put, deallocates matrices, calls ODB_cancel
- CLOSEDB
  - Closes ECMA/CCMA databases

ECMWF

ODB/IFS interface routines' interaction



ECMA – IFS usage of ODB



Working with observational arrays

- Once **GETDB** has been called, you usually get one or more of the following arrays filled with observational data:
  - **ROBHDR**: index & hdr – tables related data
  - **ROBODY**: body, errstat, update_* – tables' data
  - **MLNKH2B**: Coupling between **ROBHDR** & **ROBODY**
- **ROBHDR**, **ROBODY**, *etc*. contain a snapshot of report data and are only available between **GETDB-PUTDB calls!**

```
HDR_LOOP: do jobs=1, NROWS_ROBHDR
  ROBHDR(jobs,MDBLAT) = <some_thing>
  BODY_LOOP: do jbody= MLNKH2B(jobs), MLNKH2B(jobs+1) - 1
      if ( ROBODY(jbody,MDBVNM) == <varno> ) then
          ROBODY(jbody, MDBOMF) = <some_thing>
      endif
    enddo BODY_LOOP
enddo HDR_LOOP
```

Resolving MLNKH2B

- The linking vector between ROBHDR & ROBODY is called MLNKH2B and is created while in GETDB (more specifically while in CTXGETDB)
- Its length is always NROWS_ROBHDR + 1
- Each entry of MLNKH2B(JOBS) defines the offset to the ROBODY-row from ROBHDR(JOBS), thus the difference MLNKH2B(JOBS+1) - MLNKH2B(JOBS) is the number of body rows "belonging" to the ROBHDR(JOBS)
- There are currently two ways of defining MLNKH2B dynamically (see both CTXINITDB and CTXGETDB) :
  - Method#1 : `ctx(idctx,it)%view(1)%mlnkh2b = +2`

    view(1) must contain `body.len@hdr` (= MLNK_HDR2BODY(2)) as one of the entries and view(2) that retrieves the ROBODY should not contain any *restrictions* in WHERE-condition on how many body-entries to fetch
  - Method#2 : `ctx(idctx,it)%view(1)%mlnkh2b = -2`

    where MLNKH2B is computed automatically

    view(1) and view(2) should both contain `seqno@hdr` (= MDBONM) as the 1st entry

ECMWF

Other observational arrays

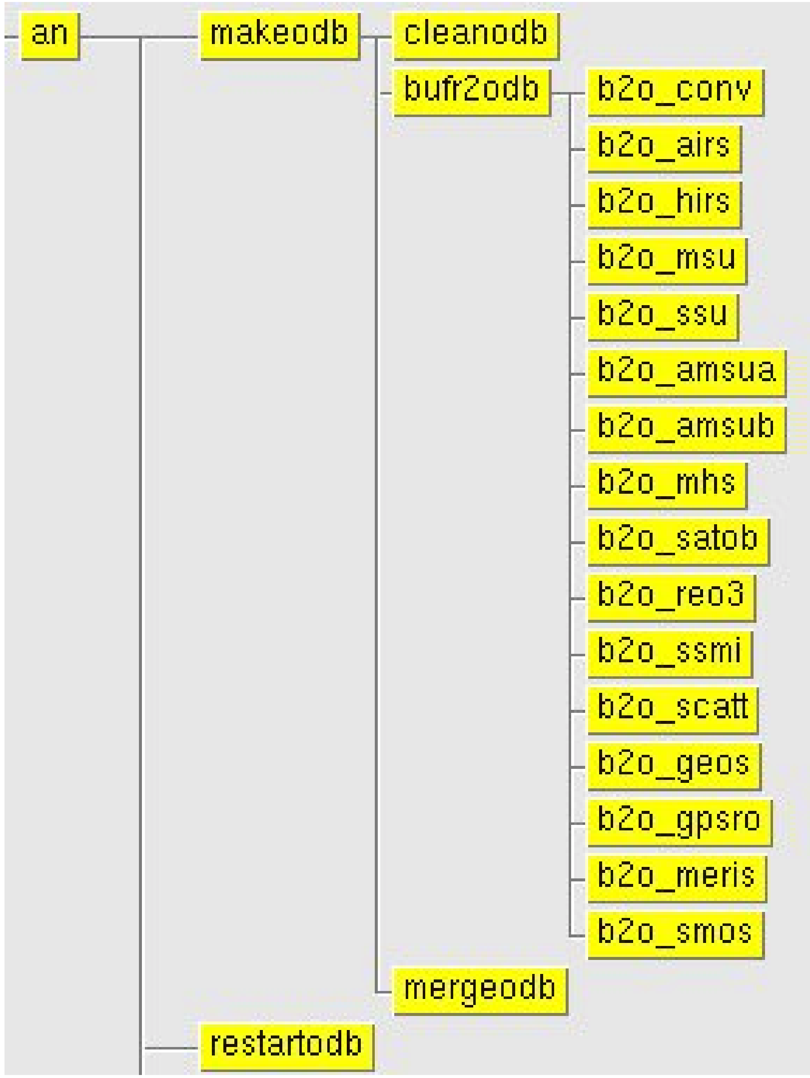
- Satellite specific data can be placed into **SATHDR** and **SATBODY** arrays. Also **SATPRED** for satellite data predictors is available separately from **SATHDR**
- These can correspond view#3 and view#4, respectively
- It also possible to have **SATHDR** only
- We usually require that **NROWS_SATHDR** equals to **NROWS_ROBHDR**. This consistency check is done in routine **GETDB**
- In some rare cases (like when creating **CCMA**) we may need **ROBHDR** "twice": once to **ECMA** and once for **CCMA**
  - For that purpose these is the array **ROBSU**
- There is also **ROBDDR** for Data Description Records

Parallelization with MPI and OpenMP

- **The data is normally extracted from the local pool(s) belonging to the particular MPI-task and arranged so that the different OpenMP threads `it` (`1..maxthreads`) get mutually exclusive datasets**
- **Each variable `ROBHDR`, `ROBODY`, `MDBVNM`, `MDBLAT`, *etc.* are in fact macros (*must be given* in CAPITAL letters) which are pre-processed with the Fortran90 data structure (see "`openmp_obs.h`")**
  - **For example, the `ROBHDR` becomes `o_(it)%robhdr`**
  - **And the `MDBVNM` becomes `o_(it)%mdbvnm`**
- **It is also possible to inquire *global data* with `GETDB`, but the following rules apply :**
  - **The *same* `GETDB` call must be issued by every MPI-task**
  - **Only local data can be modified and passed back to dbase**
  - **In `CTXINITDB`, you must remember to set :**

```
ctx(idctx,it)%replicate_PE = -1
```

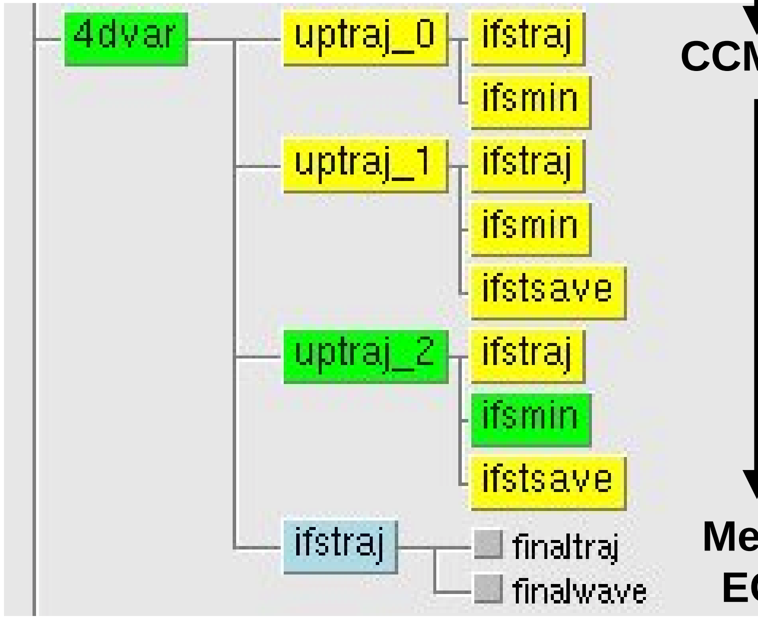
Observational data flow at ECMWF



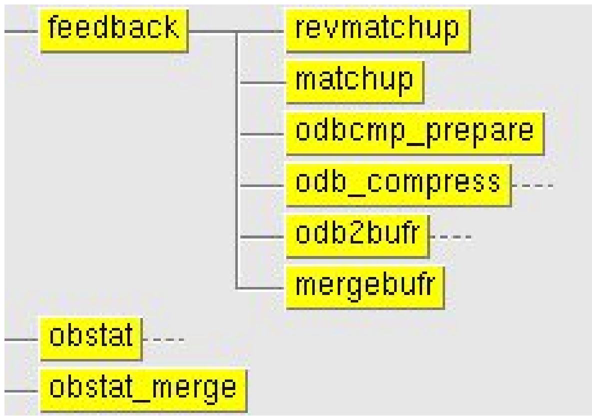
Creation of individual ECMAs

Merged ECMA

CCMA



Merged ECMA



individual ECMAs

CCMA

ECMWF bufr to ODB conversion

- **ODBs at ECMWF are normally created by using `bufr2odb`**
  - Enables MPI-parallel database creation → efficient
  - Allows retrospective inspection of Feedback BUFR data by converting it into ODB (slow & not all data in BUFR)

```
bufr2odb -i input_bufr_file -t task_id
         -n split_into_this_many_data_pools
         -I include_these_bufr_subtypes_in_database
         -E exclude_these_bufr_subtypes
         -b optional_bufr_table_directory
         -M Mergeodb → make DB ready for IFS/4DVAR
```

- **`bufr2odb` can also be used interactively, for example to create an ECMA database with 4 pools from the given BUFR input file, but includes only BUFR subtypes from 1 to 20 (inclusive): `bufr2odb –i bufr_input_file –I 1-20 –n 4`**
- **`odb2bufr`: used to archive feedback bufr in MARS**

odbshuffle – Creation of CCMA from ECMA

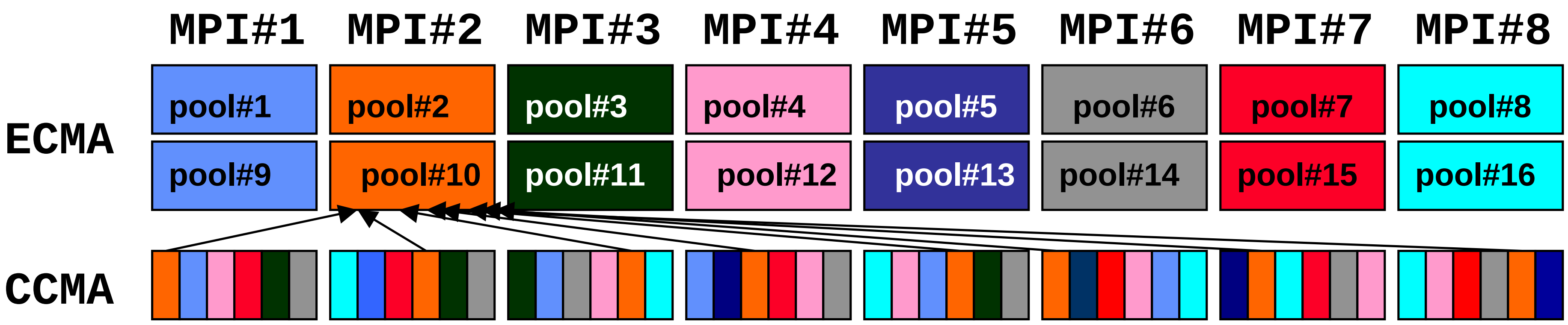
- `odbshuffle` **allows to create a new ODB database containing active observations only (assessed during screening task). To ensure a good load balancing data are re-distributed among the MPI-tasks**
  - `procid@index` **(pool number in the merged ECMA)**
  - `target@index` **(pool number in CCMA)**
- **It runs on an ECMA database containing all observations: all individual ECMAs are merged into one big ECMA (symbolic links);** `seqno@hdr` **is updated in order to be unique in the merged ECMA ;**
- **MPI over pools and OpenMP loop over observation types.**
- **The default observation weighting method is now 407 (instead of 107) to allow a better load balancing**

revmatchup at ECMWF - ECMA → CCMA

- **Used to feed information stored in `ECMAs` in the last trajectory back to `CCMA`**
- **Done for each individual `ECMAs`**
- **`ODB_IO_METHOD=` 5 for ECMA**
- **`ODB_IO_METHOD=` 4 for CCMA**
- **MPI to send data from `ECMA` to the right `CCMA` pool via the usage of the ODB `paral` function – `paral($pe,target@index)` in the `WHERE` statements of the corresponding SQL queries.**
- **`paral` is always true for the database opened in `WRITE` mode (`ECMA`) and is only used to select `CCMA` data from the right pool.**

matchup at ECMWF – CCMA → ECMA

- Used to feed information gathered during 4D-Var minimisation in `CCMA` back to individual ECMAs.
- `ODB_IO_METHOD = 5` for CCMA
- `ODB_IO_METHOD = 4` for ECMA
- OpenMP – done over sensor list but in the latest cycle, the number of openMP thread is forced to 1
- MPI to send data from `CCMA` to the right `ECMA` pool (usage of the ODB `paral` function – `paral($pe,procid@index -$hdr_min+1)`



Conclusion and future developments

Conclusions

- **Strengths of ODB**
  - It allows to process unprecedented amounts of satellite data through the IFS/4DVAR system
  - It is MPI and OpenMP parallel
  - It is portable (written in ANSI-C and Fortran 90, support for big/little endian)
- **Weaknesses of ODB**
  - ODB has got many components and few users have a good understanding of all capabilities of ODB
  - Cycle dependence of ODB (even if the dependence only exists because of precompiled ODB databases and queries)
  - Usage of ODB within IFS is complex and focused on database handling instead of observations
  - At ECMWF, resulting ODB databases (`ECMA/CCMA`) are archived in ECFS for a short period of time (feedback bufr are archived in `MARS`); users need to retrieve full `ECMA/CCMA` for post-processing (requires large local disk for each user)

Short-term outcomes

- **Distribution of stand-alone ODB package under investigation (now only available to member states).**
  - **At the last ACDP, it was proposed to distribute ODB at a handling fee charge; License to be investigated (Apache or ECMWF license)**
- **Documentation**
  - **ODB FAQ**
  - **ODB user guide (ODB core, generic Fortran 90 interface, ODB-tools)**
  - **ODB usage in IFS**
- **Archiving of resulting ECMA (feedback bufr) in MARS.**
  - **A new format ODA (Observational Data Archiving) has been defined (ODB has been considered as unsuitable)**
  - **A new C++ library is under development at ECMWF (Peter Kuchta) as well as ODA-tools (odb2oda, oda2odb, oda SQL engine to query ODA files)**
  - **This ODA format will become an internal format for Metview/Magics++.**

Future developments – Split ODB

- This new ODA library is an opportunity to split ODB
  - Can we use this new underlying format in ODB?
    - We would only change how we read and write data on disk
    - For now we can read ODA (Fortran 2003 to interface with C++ ODA library) and create an ODB to be used in IFS
  - Can we replace the current ODB/SQL engine by ODA/SQL engine?
    - We would avoid to pre-compile ODB databases and SQL queries
    - We would use the same set of tools
  - Having this ODA library outside IFS would allow to develop tools to post-process ODB data independently of IFS cycles.
  - Maintenance of this library will be done by ECMWF data and Services

Future developments – IFS interface

- **The current ODB interface to IFS was built on an existing software layer (pre-ODB) and the main objective was**
  - **to change from the static offsets (pre-calculated offsets, using so called NCMxxx pointers) into dynamic ones without changing the IFS data flow**
  - **to have a subset of observations available in dynamically allocated matrices (introduction of dynamic column pointers MDBxxx)**
  - **To minimize code changes necessary to use ODB: changes to the IFS code were nearly automatic (with Perl scripts)**
- **Can we ease the usage of ODB in IFS?**
  - **OOPS (Object Oriented Prediction System) is a good opportunity to replace the current ODB interface to IFS.**
  - **The objective would be to hide these observational arrays (ROBHDR, ROBODY, etc.) and to hide the usage of ODB databases (ECMA/CCMA). Users would handle observations.**