COPE: update on latest plans

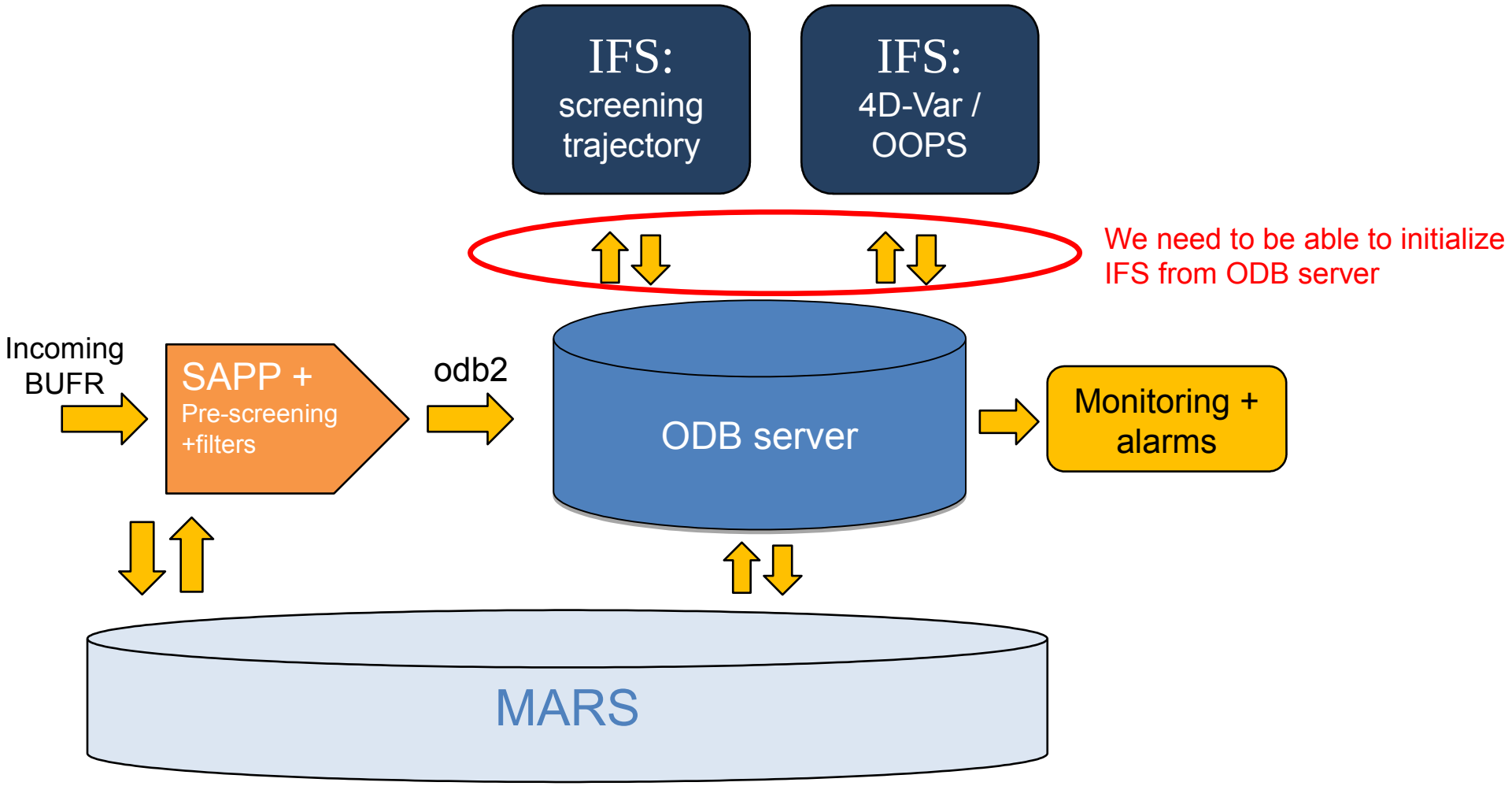
Peter Lean and Peter Kuchta

Thanks - Alan Geer, Tomas Kral, Deborah Salmond, Niels Bormann and John Hague

IFS / Arpège co-ordination meeting June 6th, 2016

ECMWF

COPE Architecture: from 10,000m

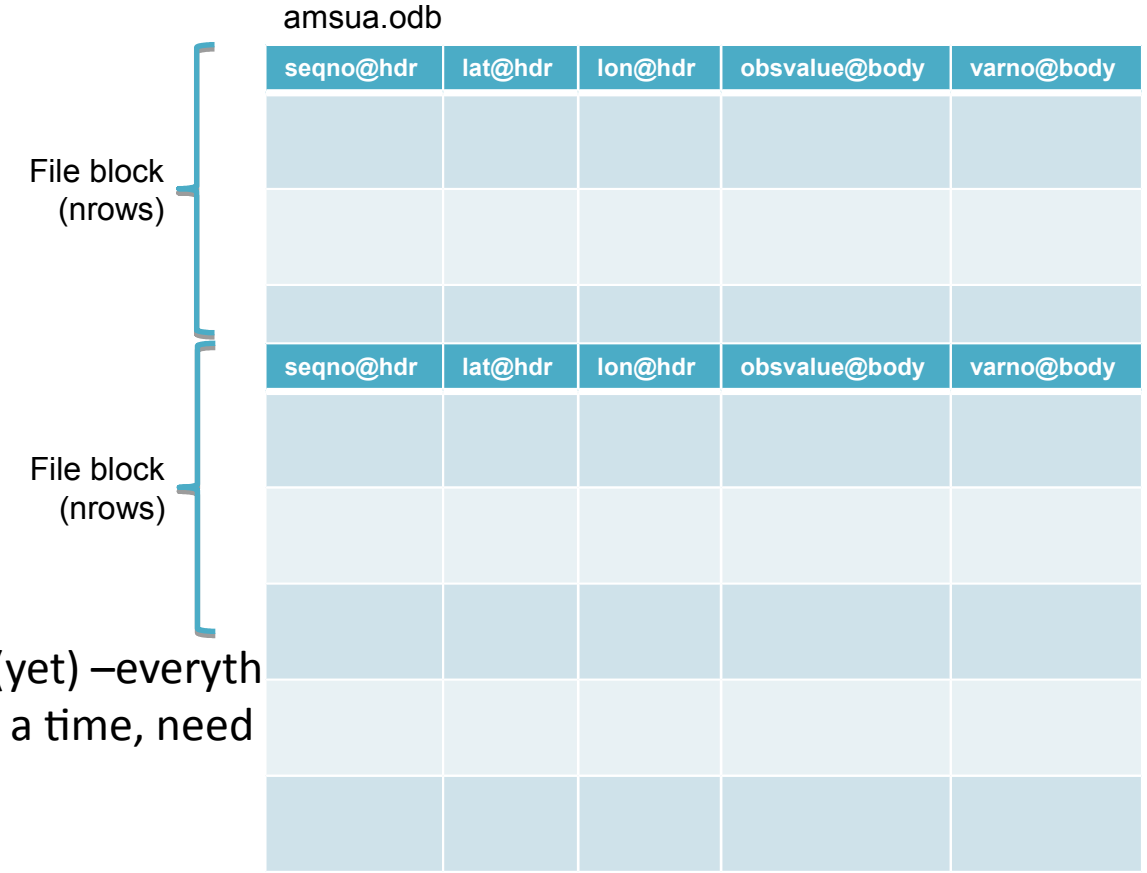


Benefits of COPE

- Increased resilience of forecasting system to corrupt incoming observations
  - data problems can be dealt with before the time critical path.
- Continuous monitoring and alarms.
- Time savings on the time critical path by running pre-screening earlier
  (up to 200s; not huge, but not negligible)
- An opportunity to make a cleaner architecture and modernize some old code (especially in bufr pre-screening used at ECMWF).

odb_api (“odb2”)

- It is not an API for ODB, it is a completely independent library.
- Written by the Development Section in the Forecast Department at ECMWF.
- Designed to meet the needs of archiving:
  - Files are streamable.
  - Flat table structure.
  - Single file.
- Similar odbsql functionality.
- Flexible schema.
- Not parallelized, or optimized for HPC:
  - It is not a parallel, in-memory database (yet) –everyth
  - Can be slow (can’t read just one table at a time, need
- We use odb1 for HPC applications (IFS)
  and odb2 for archiving and interactive (odbsql) use by scientists.



odb_api and MARS

- Allows you to reduce the volume of data returned from the archive.

- e.g. calculate standard deviation of first guess departures for active AMSU-A channel 12 for all of 2015. The calculation is performed on the server and only the result is returned.

```
retrieve,
   type = ofb,
   class = od,
   obsgroup = amsua,
   stream = lwda,
   expver = 001,
   date = 20150101/to/20151231,
   time = 00/12,
   filter = "select stdev(fg_depar) where vertco_reference_1=12 and datum_status@body=1;",
   target = "/scratch/rd/dipl/result.odb"
```

ODB server

- Has been used for several years at ECMWF as a staging area before odb(2) files are archived on MARS.

- Essentially, odb2 files stored in a defined directory structure

  `./{expid}/{yyyymmdd}/{hh}/{obsgroup}/{reportype}.odb`

- Server allows you to extract and use data using MARS requests:

```
retrieve,
   type = ofb,
   class = rd,
   database = ccb odb,
   obsgroup = amsua,
   stream = lwda,
   expver = gbdd,
   date = 20150819,
   filter = "select * where datum_status@body=1;",
   time = 00,
   target = "/scratch/rd/dipl/myfile.odb"
```

ifsobs:

a data access layer for observations in IFS

ECMWF

Interfacing IFS with an ODB server

- Currently, IFS uses ODB1 throughout:
  - reads and writes data to ODB1 files on disk.

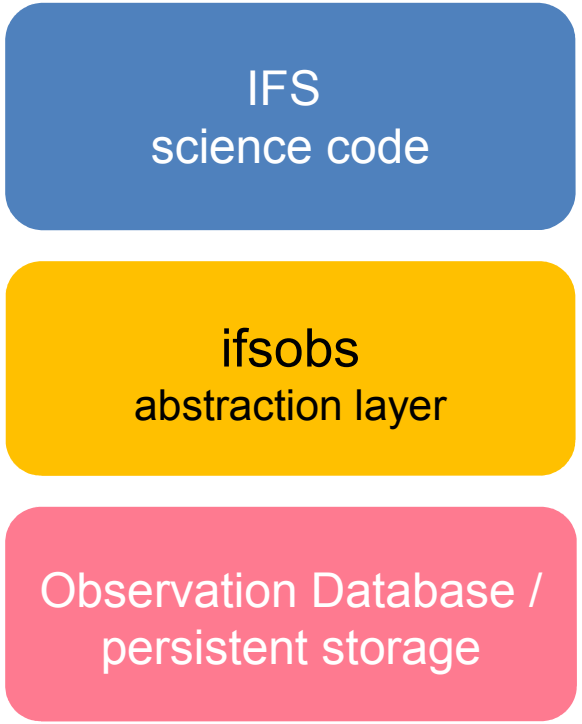
- For COPE, we will be requesting data from ODB server (odb2) over the network.

- To do this, we need to modify how we access and use observational data internally within IFS:

  - want to decouple IFS from the underlying file format and/or database engine.

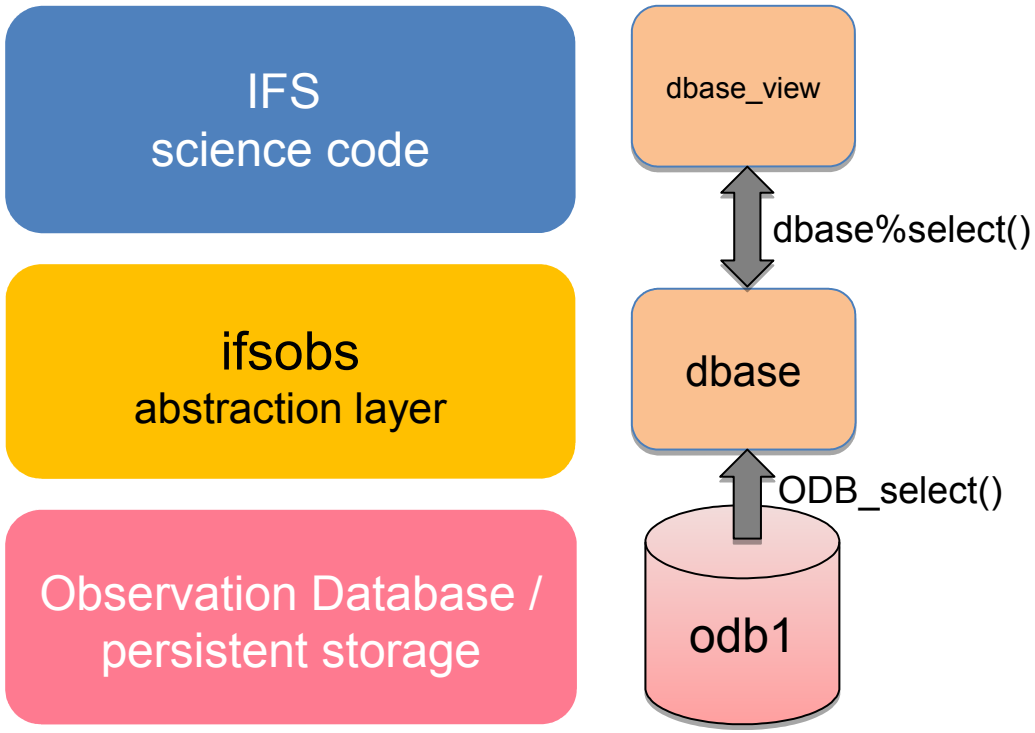
ifsobs

- a Data Access Layer for observational data in IFS:



ifsobs

- a Data Access Layer for observational data in IFS:

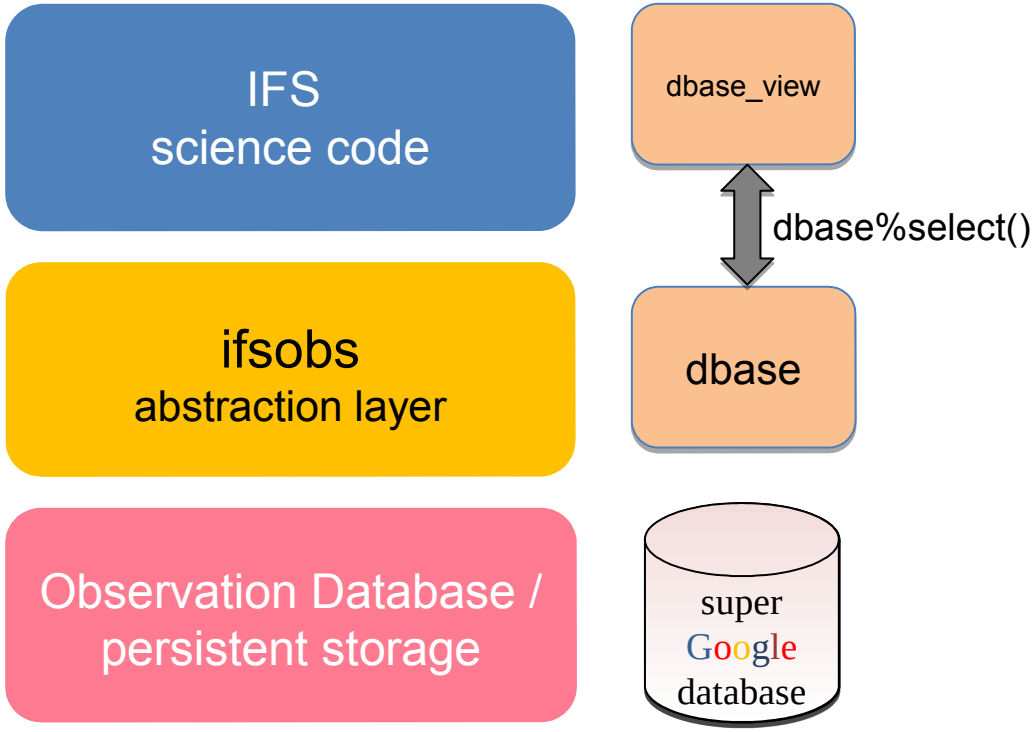


Two main objects:

- dbase : abstract database class provides common interface to different databases,
- dbase_view : encapsulates observation data and provides convenient access and manipulation methods.

ifsobs

- a Data Access Layer for observational data in IFS:



Decouples IFS from the underlying database / file format.

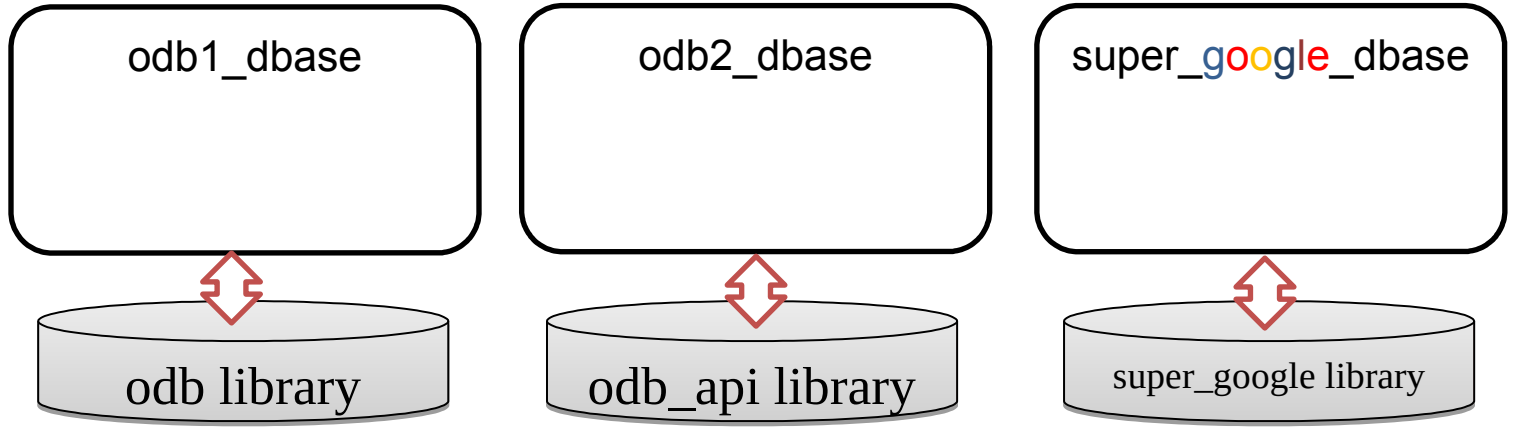
IFS only interacts with dbase and dbase_view objects, **not** with the underlying database directly.

Abstract dbase class F2003

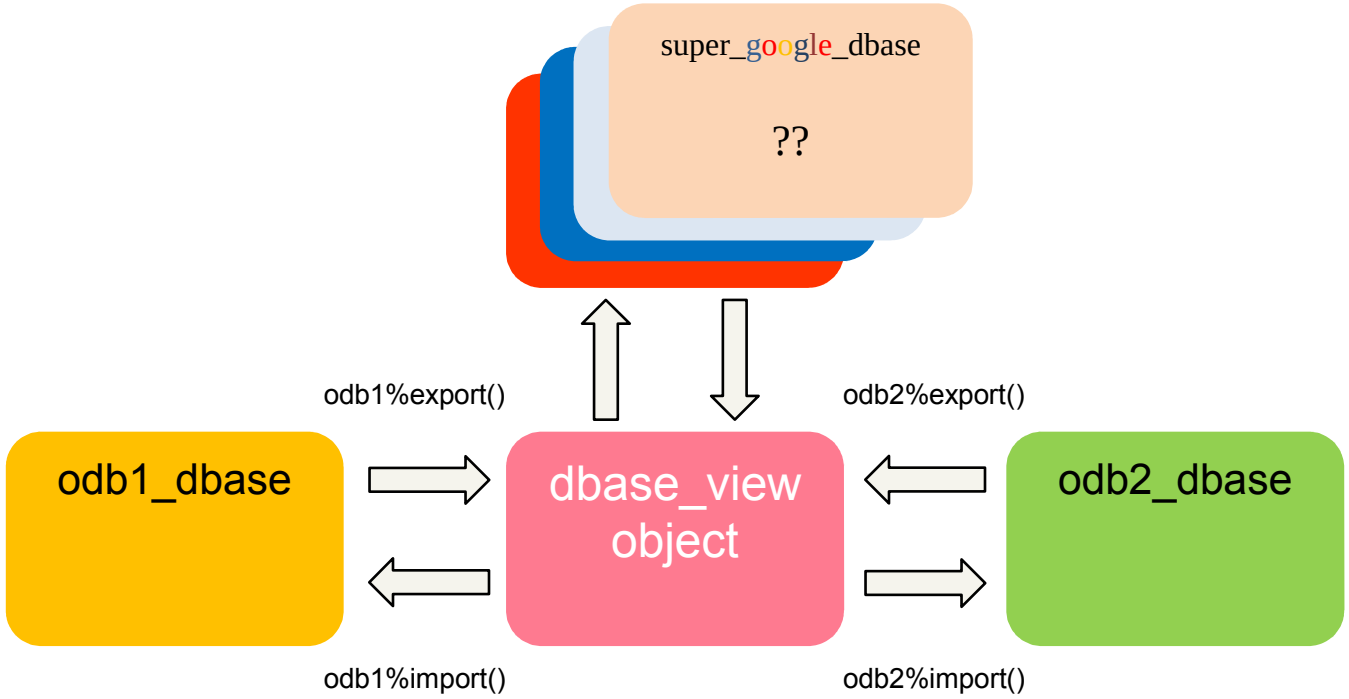
```
type, abstract  :: dbase
      character(len=strlen)          :: filename     = ""
      integer(kind=jpin)             :: handle       = ODB_INT_MDI
      logical                        :: readonly     = .true.
      logical                        :: inuse        = .false.
contains
      procedure(generic_open),       deferred, pass(this)  :: open
      procedure(generic_close),      deferred, pass(this)  :: close
      procedure(generic_import),     deferred, pass(this)  :: import
      procedure(generic_export),     deferred, pass(this)  :: export
      procedure(generic_select),     deferred, pass(this)  :: select
      procedure(generic_put),        deferred, pass(this)  :: put
      procedure(generic_destroy),    deferred, pass(this)  :: destroy
end type dbase
```

What can you do with databases?

~~How do they do it?~~



ifsobs

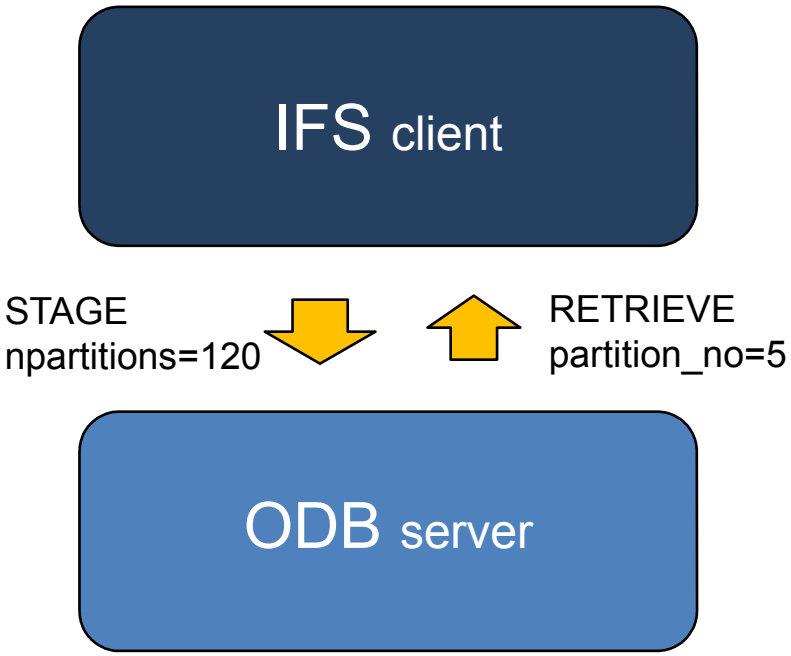


Provides convenient mechanism to transfer data between odb2 ↔ odb1 (as required by COPE).

Getting data from ODB server into IFS

Application doesn't know what type of database

```
type(dbase_factory)     :: db_factory
class(dbase), pointer   :: odb2 => null()
class(dbase), pointer   :: odb1 => null()
type(dbase_view)        :: view

! Create dbase objects
call db_factory%init("odb2")
odb2 => db_factory%create_dbase()

call db_factory%init("odb1")
odb1 => db_factory%create_dbase()

! Open an (empty) ECMA database
rc = odb1%open("ECMA","new",npools)

! STAGE your request on ODB server
rc = odb2%open(mars_request,"r",npools)

do i=1,npools

 ! Retrieve data from ODB server
 rc = odb2%export(view,poolno=i)

 ! Import data into odb1 in-memory
 rc = odb1%import(view,poolno=i)

 call view%destroy()

enddo
```



Essentially, we have replaced OPENDB with something more flexible.

New framework makes it easy to ingest data from the new ODB server.

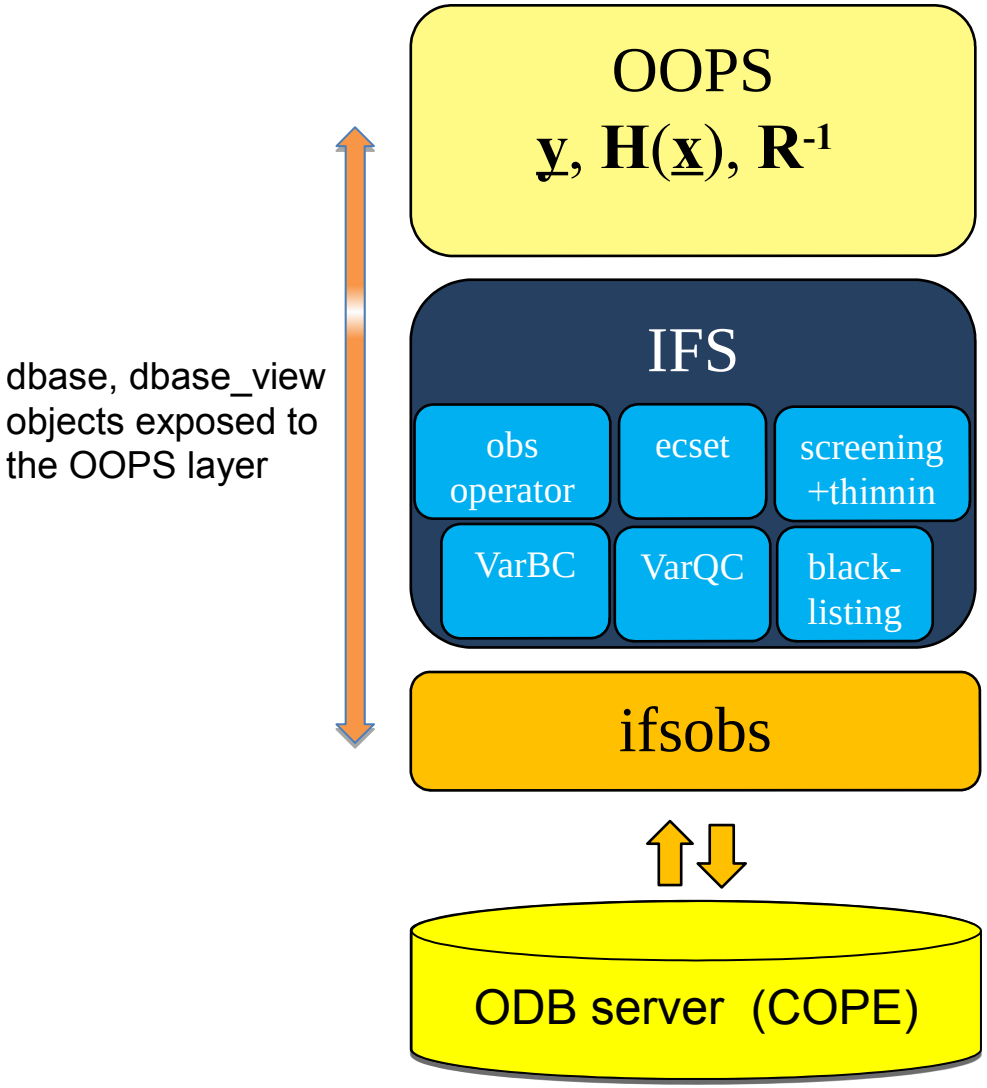
Recent progress

- "ifsobs" development maturing:
  - some elements of it are in 43r1 as part of the OOPS observation operator refactoring,
  - performance overhead of the object-oriented layer at an acceptable level.

- COPE ODB server / client demonstration:
  - partitioning functionality implemented in ODB server [Peter Kuchta]
  - full load of observational data from one cycle ingested into ODB server,
  - MPI-parallel Fortran client application written using ifsobs framework:
    - successfully requests partitioning and extraction of data from server,
    - converts into odb1 in-memory (takes ~3s for full ECMA).

- Next steps:
  - initialize an IFS experiment from ODB server : replace OPENDB,
  - demonstrate putting data back into ODB server from IFS.

ifsobs and OOPS



ifsobs rollout plans: one getdb at a time…

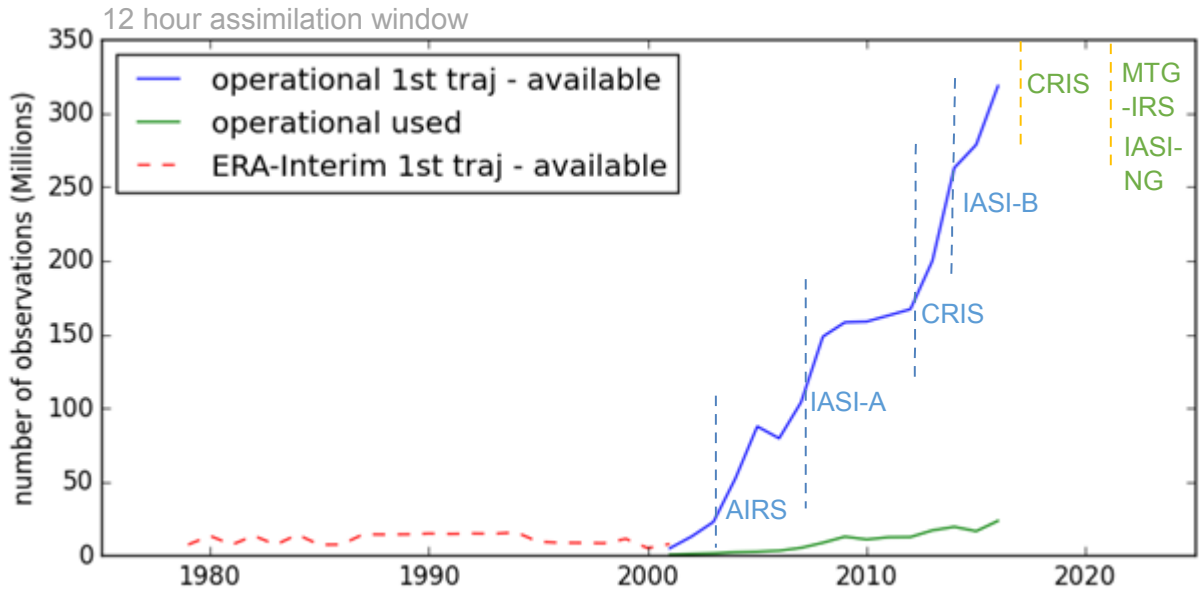
- **HOP** ✓
- **ECSET (43r3)**
- **VarBC (43r3)**
- **Screening:** including thinning, duplication checks etc
- **ObsDist:** re-pooling observations to match model domain decomposition
- **ObSort:** sub-selection and load balancing of observations in CCMA creation
- **Matchup:** passing feedback data from CCMA back into ECMA
- **Aim: retire cma2odb by 2018**

Summary

- ifsobs is a new abstraction layer for observations in IFS:
  - Helps COPE by decoupling the IFS from the underlying observation file format allowing IFS to be initialized from ODB server.
  - Helps OOPS by providing dbase and dbase_view objects that can be exposed to the OOPS layer.
  - Helps improve the usability, readability, debugability of the observation science code.

Thanks for listening

Context: Trends in number of observations

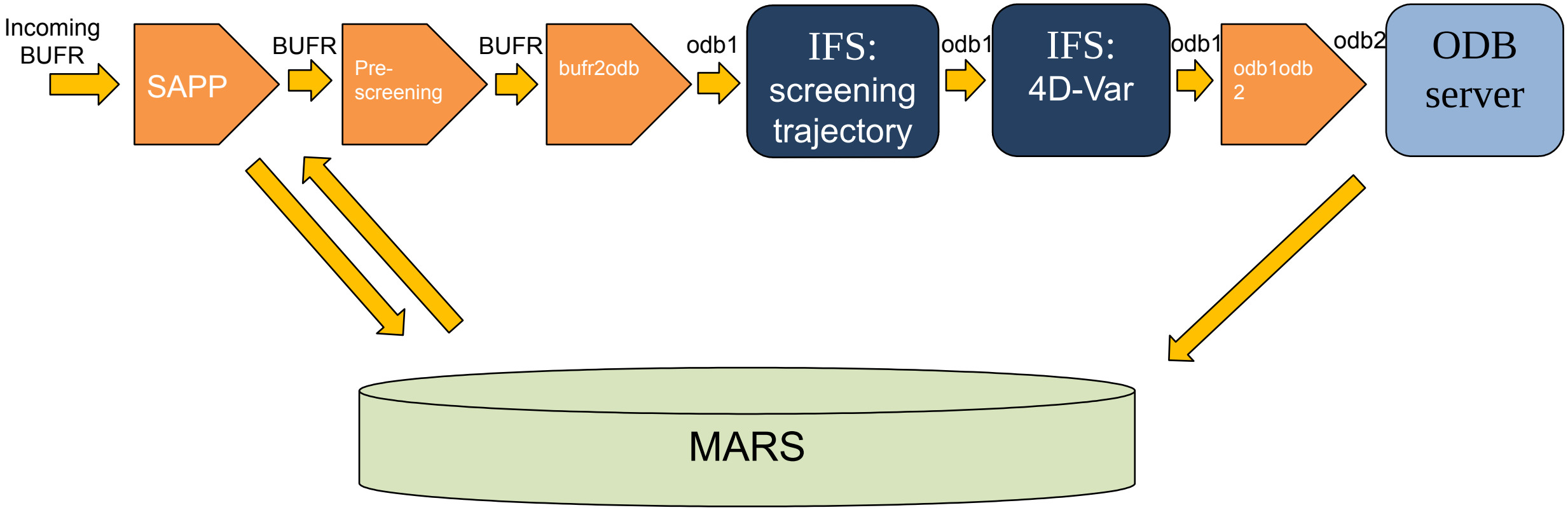


Other factors:

- Long window 4D-Var? 5 day = x10
- Spatial error correlations? Less thinning.
- Clear sky only → all-sky.

- Explosion in availability of satellite data since 2001.
- Mostly from infra-red sounders.
- Only around 5-10% are actively assimilated:
  - most are cloud-screened, thinned or blacklisted.
- Trend expected to continue with new CRIS, MTG-IRS, IASI-NG
- Projection for next 10 years:
  - x2 (current usage trends)
  - x10 (more aggressive usage)

Current Architecture: from 30,000ft



- **dbase_view**

Encapsulates everything needed to describe odb data

```
type dbase_view
   character(len=strlen)                :: view_name    = ""
   real(kind=jprl), allocatable         :: data(:,:)
   integer(kind=jpin)                   :: nrows        = 0
   integer(kind=jpin)                   :: ncols        = 0
   character(len=strlen), allocatable   :: colnames(:)
   integer(kind=jpin), allocatable      :: coltypes(:)
   character(len=strlen), allocatable   :: coltables(:)
   type(bitfield), allocatable          :: bitfields(:)
contains
   procedure                  :: get_column_index
   procedure                  :: get_column_ptr
   procedure                  :: copy_column
   procedure                  :: get_value
   procedure                  :: set_value
   procedure                  :: fill_mdi
   procedure                  :: trim_mdi
   procedure                  :: copy_common_columns_from
   procedure                  :: compatibility_mode
   procedure                  :: destroy
end type dbase_view
```