Practical lessons learned from implementing an Object-Oriented toy system

- History of Toy OOPS
- What is Toy OOPS?
- Code examples from Toy OOPS
- Mixed Fortran / C++ / C ?
- OO_ODB
- Summary

Deborah Salmond and
Anne Fouilloux

Toy OOPS

- ‘Toy’ data assimilation system to try out Object-Oriented programming for IFS

- Abstract Part
  - Code the algorithm in terms of base classes which serve to define interfaces to the data structures & functions
    - can be compiled separately

- Implementations
  - Code Lorenz and QG models in terms of derived classes from the base classes which define data structures and functions
    - without change of abstract part

History of Toy OOPS

- F2003 version with ‘abstract part’ + Lorenz model
- QG added to F2003 without change in ‘abstract part’
- C++ version ‘translated’ from F2003 + Lorenz model
- QG with compute parts in F90 added to C++ version
- Improved F2003 and C++/F90 versions
  - available in Dec 2009

Toy OOPS abstract part

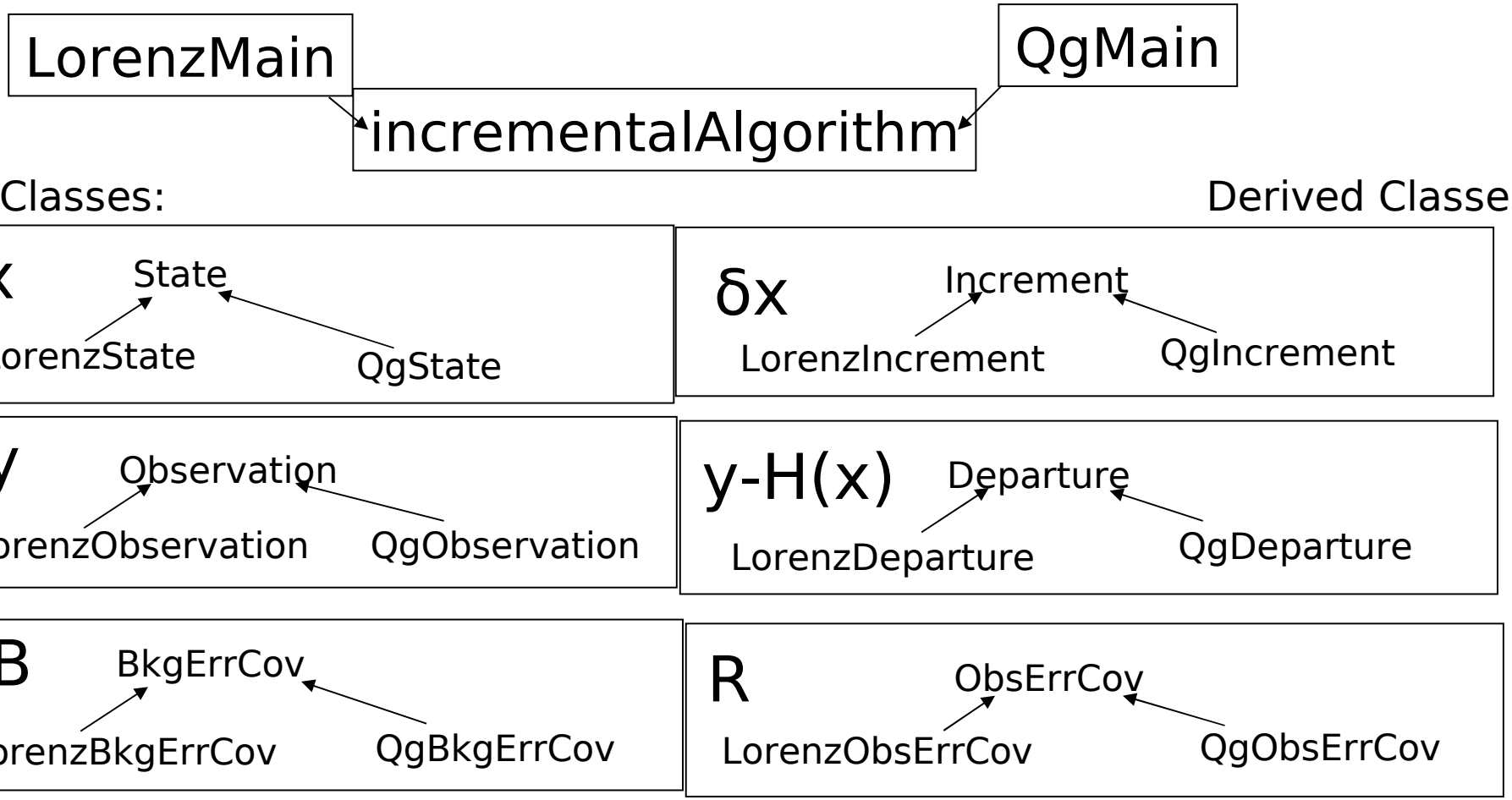
$$J(x)=\frac{1}{2}(x_0-x_b)^T B^{-1}(x_0-x_b)+\frac{1}{2}\sum_i [H(x)-y_i]^T R^{-1}[H(x)-y_i]$$

incrementalAlgorithm

Base Classes:

x	State	δx	Increment
y	Observation	y-H(x)	Departure
B	BkgErrCov	R	ObsErrCov

Toy OOPS implementations



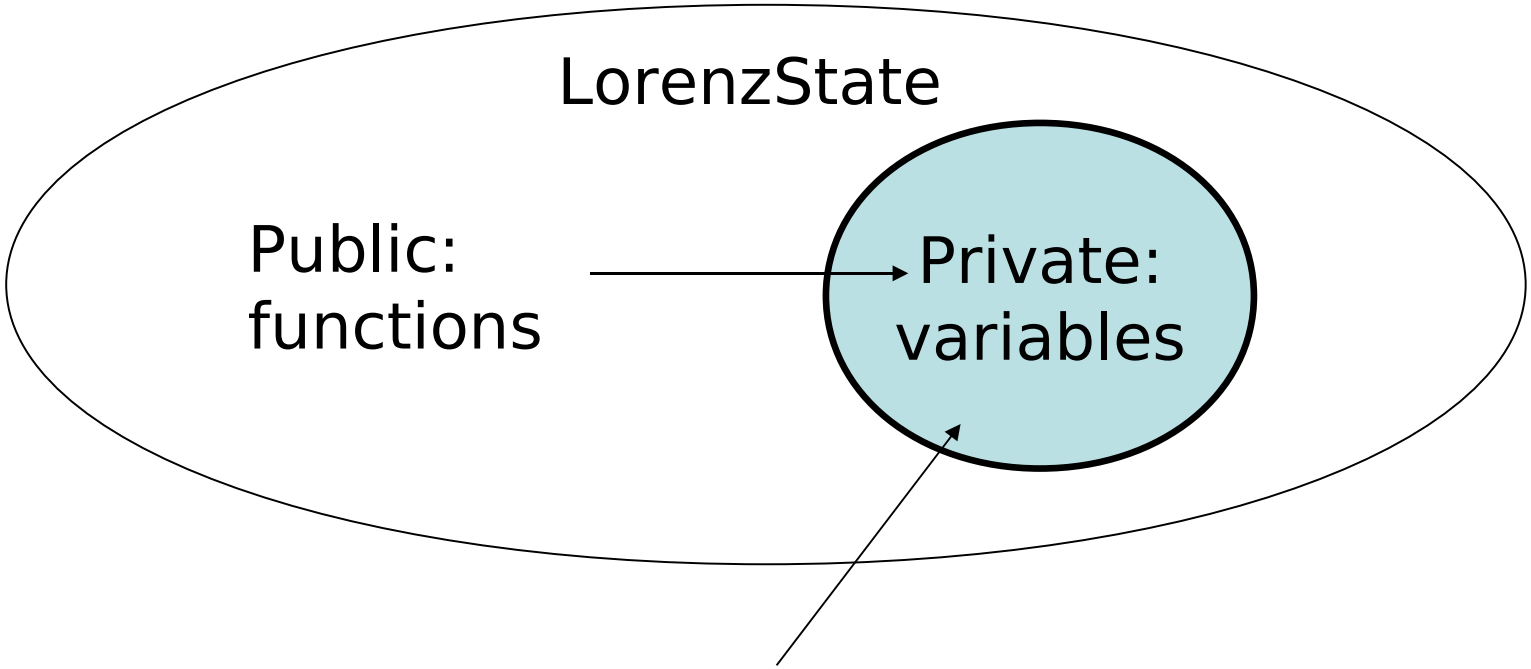
OO C++ and F2003 words

- Class
- Object
- Constructor & Destructor
- {}
- Overloading of operators & functions
- Pointer
- ;

Class

- A Class allows encapsulation of a ‘user-defined’ type together with its functions
- Instances of these Classes are known as Objects
- Visibility of member variables and functions can be controlled
- A base class can serve to define interfaces
- A derived class can add or modify features of an existing class – ‘inheritance’
- Objects of these classes can be manipulated identically by other parts of the program – ‘polymorphism’

Class access control



Friend functions from other classes

→ Compare with IFS where everyone can access everything

Example of C++ Class, constructor, destructor & function

```
class LorenzState {

private:
  double* x;

public:
   LorenzState(){
    x = new double [3];
    for (int i=0; i< 3; i++) x[i] = 0.0;
   }
   ~LorenzState() {
    delete [] x;
   }
   void read() {
     this->x[0]=1.0;
     this->x[1]=3.0;
     this->x[2]=5.0;
  }
};
```

```
{
LorenzState x1;
x1.read();
}
```

Example of C++ Class & overloaded operator

```
class LorenzState {

private:
  double* x;

public:
   LorenzState(){
    x = new double [3];
    for (int i=0; i< 3; i++) x[i] = 0.0;
   }
  ~LorenzState() {
    delete [] x;
   }
   void operator=(LorenzState & rhs) {
    for (int i=0; i< 3; i++) {
      this->x[i]  = rhs.x[i];
  }
};
```

```
{
LorenzState x1;
LorenzState x2;
x1=x2;
}
```

Example of C++ base & derived Class

Derived Class

```
class LorenzState : public State {

private:
  double* x;

public:

LorenzState():State() {
  x = new double [3];
  for (int i=0; i< 3; i++) x[i] = 0.0;
}
~LorenzState(){
  delete [] x;
}
void read() {
   this->x[0]=1.0;
   this->x[1]=3.0;
   this->x[2]=5.0;
 }
};
```

```
{
LorenzState xb;
xb.read();
}
```

Base Class

```
class State {
public:
   State(){};
   virtual void read()=0;
};
```

Polymorphic Incremental Algorithm

Lorenz_main.cpp

```
LorenzState xb,xc;
IncrementalAlgorithm(xb,xc);
```

QG_main.cpp

```
QgState xb,xc;
IncrementalAlgorithm(xb,xc);
```

```
void IncrementalAlgorithm (State &xb, State &xc){
    xb=xc
}
```

```
class State {
public:
   State(){};
  virtual State & operator=(State &)=0;
};
```

```
class LorenzState
   operator =
```

```
class QGState
   operator =
```

- Classes LorenzState and QgState must overload the "=" operator
- = function used depends on class of xb and xc
- IncrementalAlgorithm is written without knowing which it will use

For comparison some F2003

```
module lorenz_states
……
public :: assignment(=) =>ls_assign_s
……
subroutine ls_assign_s(self,rhs)
implicit none
class(lorenz_state), intent(inout) :: self
class(state), intent(in)           :: rhs
SELECT TYPE (rhs)
CLASS IS (lorenz_state)
  self%x(:)=rhs%x(:)
CLASS DEFAULT
  call abor1('error')
END SELECT
end subroutine ls_assign_s
end module lorenz_states
```

```
type(lorenz_state) :: x1,x2

x1=x2
```

Toy OOPS Classes

Base classes

State

Increment

Observation

Departure

BkgErrCovariance

ObsErrCovariance

Derived classes

Lorenz QG

DummyOdb

Abstract Part

ControlVariable

IncrementalControlVariable

ChangeVariable

ControlVector

Costfunction

Observer

Trajectory

IncrementalAlgorithm

ConjugateGradient

C++ pointers

- double* ptr_name;
- double name;
- ptr_name = &name;
- double name1 = *ptr_name;

- <base class>* pointer can point to <derived class> object

- Can pass by reference (like in Fortran)
  - function (double &name);

→IFS : a ‘F90 C++ sandwich’

Main program: master.F90
calls mpl_init etc.

Control layer in C++ : IFS_main
Abstract part: IncrementalAlgorithm.cpp,
Stepo.cpp, Hop.cpp,
State.cpp, Increment.cpp, etc.
IFS specific: IFS_State.cpp, IFS_Increment.cpp, etc.

Computational parts in F90:
cpg.F90, callpar.F90, rttov.F90 etc.

Testing of Toy OOPS

C++ & Fortran90

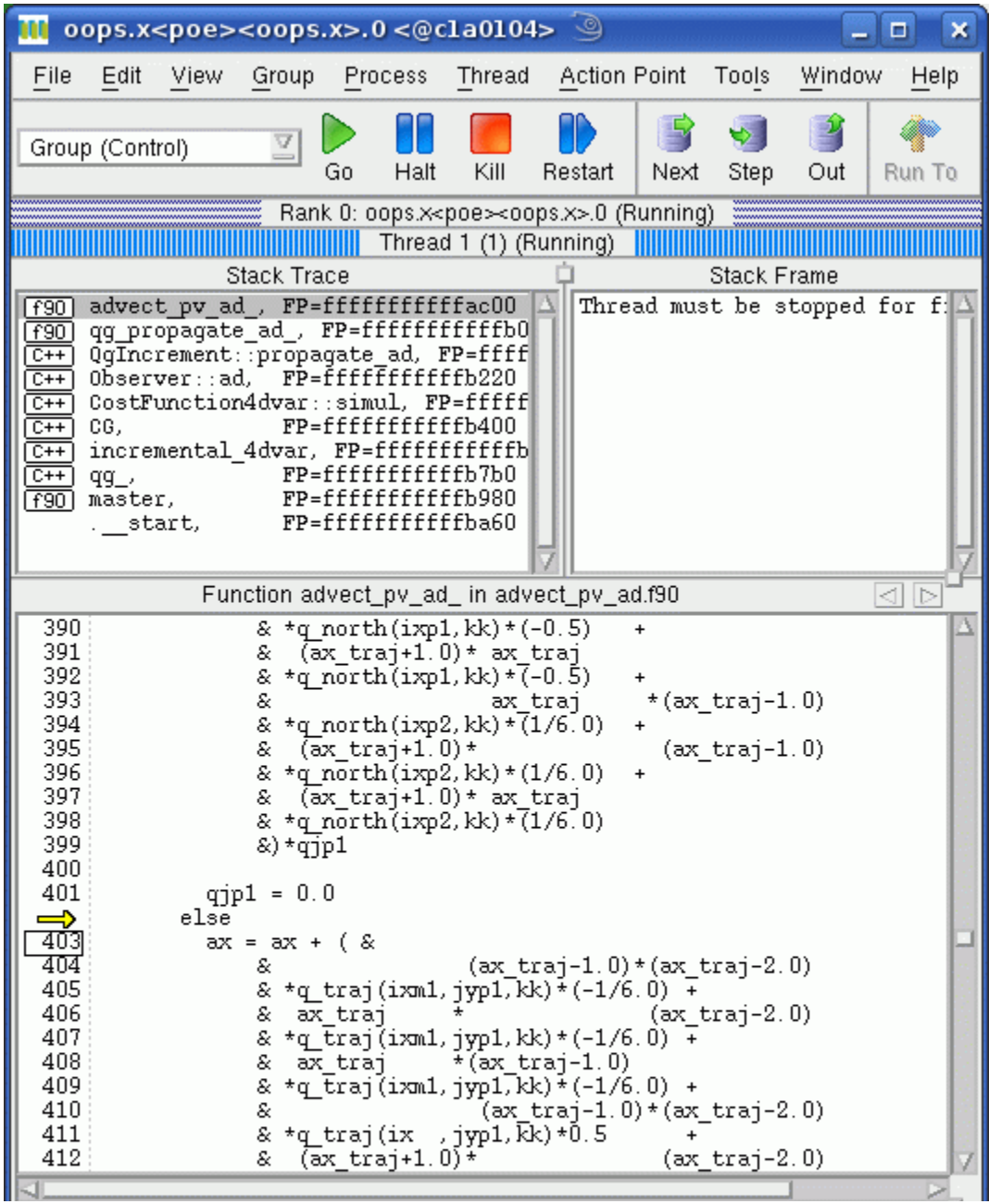
- IBM
  - xlf90 and xlC
- NEC
  - sxf90 and sxc++
- Linux
  - pgf90 and pgcc
  - gfortran and gcc

Fortran2003

- IBM
  - xlf
  - fortran/xlf/12.1.0.4
- NEC
  - not available
- Linux
  - nagfor
  - gfortran

Totalview

- Works OK ☺



Xprofiler for QG

File Code Display Utility

%time	cumulative seconds	self seconds	calls	self ms/call	total ms/call	name
22.9	19.79	19.79				.advect_pv_ad_ [1]
17.7	35.14	15.35				.advect_pv_tl_ [2]
16.4	49.38	14.24				.__xlidflr [3]
16.0	63.25	13.87				.__xlimdlo [4]
8.2	70.34	7.09				.solve_helmholz_ [5]
3.9	73.75	3.41				.vpassm_ [6]
2.5	75.94	2.19				.__xlimdlo.GL [7]
2.5	78.10	2.16				.laplacian_2d_ [8]
2.0	79.82	1.72				.advect_pv_ [9]
1.2	80.90	1.08				.pv_operator_ [10]
0.8	81.62	0.72				.meridional_wind_ [11]
0.8	82.32	0.70				.pv_operator_ad_ [12]
0.7	82.92	0.60				.solve_elliptic_system_ [13]
0.6	83.42	0.50				.zonal_wind_ [14]
0.6	83.90	0.48				.fft99b_ [15]
0.4	84.21	0.31				.invert_pv_ad_ [16]
0.4	84.52	0.31				.meridional_wind_ad_ [17]
0.3	84.80	0.28				.zonal_wind_ad_ [18]
0.3	85.07	0.27				.pv_ [19]
0.3	85.30	0.23				.fft99a_ [20]
0.3	85.53	0.23				.zonal_wind_tl_ [21]
0.2	85.74	0.21				.fft991_ [22]
0.2	85.94	0.20				.__sqrt [23]
0.1	86.00	0.06				.ControlVector::operator+(ControlVector&) [24]
0.1	86.05	0.05				.__mcount [25]
0.1	86.10	0.05				.qg_propagate_ad_ [26]
0.0	86.14	0.04				.ControlVector::dot_product(ControlVector&) [27]
0.0	86.18	0.04				.QgBkgErrCovariance::sqrt_mult(Increment&,ControlVect

Traceback

```
0:  Signal received: SIGFPE - Floating-point exception
0:      FP division by zero
0:
0:  Instruction that generated the exception:
0:    fdiv fr00,fr00,fr01
0:    Source Operand values:
0:      fr00 =   5.00000500000000e+11
0:      fr01 =   0.00000000000000e+00
0:
0:  Traceback::
0:    Offset 0x000001bc in procedure qg_propagate_@OL@1
0:    Location 0x0900000000848d58
0:    Offset 0x000001a4 in procedure qg_propagate_, near line 33 in file QG/qg_propagate.f90
0:    Offset 0x00000080 in procedure propagate__7QgStateFv, near line 71 in file QG/qgState.cpp
0:    Offset 0x00000370 in procedure generate_odb__Fv, near line 56 in file QG/qg_utility.cpp
0:    Offset 0x000000ac in procedure qg_, near line 35 in file QG/qg.cpp
0:    Offset 0x00000170 in procedure master, near line 23 in file master.f90
0:    --- End of call chain ---
```

Toy OOPS Summary

- Demonstrate writing a data assimilation algorithm in abstract terms such that each part is easily identifiable and switching one part does not mean complete code re-write
- Mixed C++/Fortran90 technically OK
- Compute done in F90 so Gflops same as now
- By design OO layer at top level – for data structure and algorithm definition
- Improve IFS interface to ODB - very suitable for OO

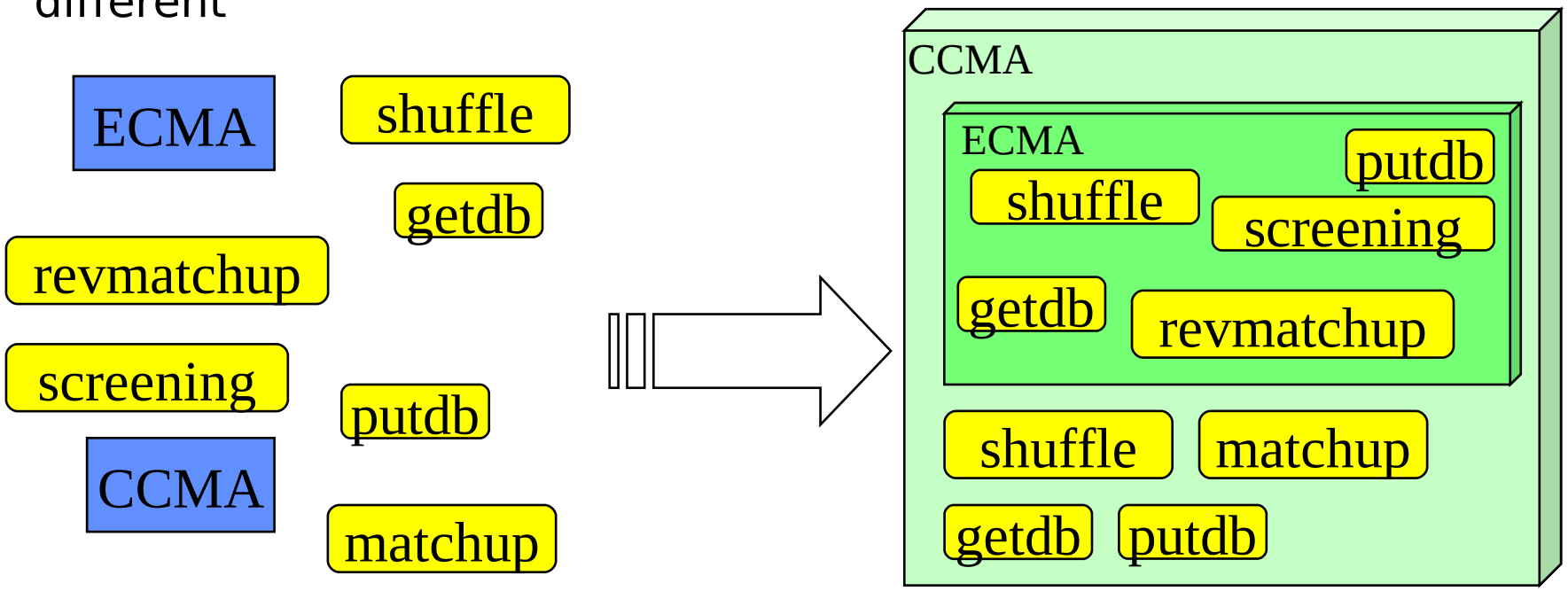
From the toy system to ODB/IFS...

- How helpful is the toy system to design the real application?
  - it highlights user needs and gives confidence to go further
  - objects and their associated methods are more easily identified
- A simple case study: the ODB interface for IFS
  - It specializes the usage of ODB for ECMA/CCMA databases
    - How?
    - if LLECMA, LLCCMA, LLSCREEN, etc.; yomdb Fortran 90 module, ctxinitdb.F90 and associated SQL retrievals
    - Why?
    - Some tasks (shuffle, (rev)matchup, observation operator, etc.) require some background knowledge on the database organisation (cma.h) or content (some ODB attributes only apply to specific observation types)

Would an object-oriented approach help to simplify

HIERARCHY BETWEEN ECMA/CCMA

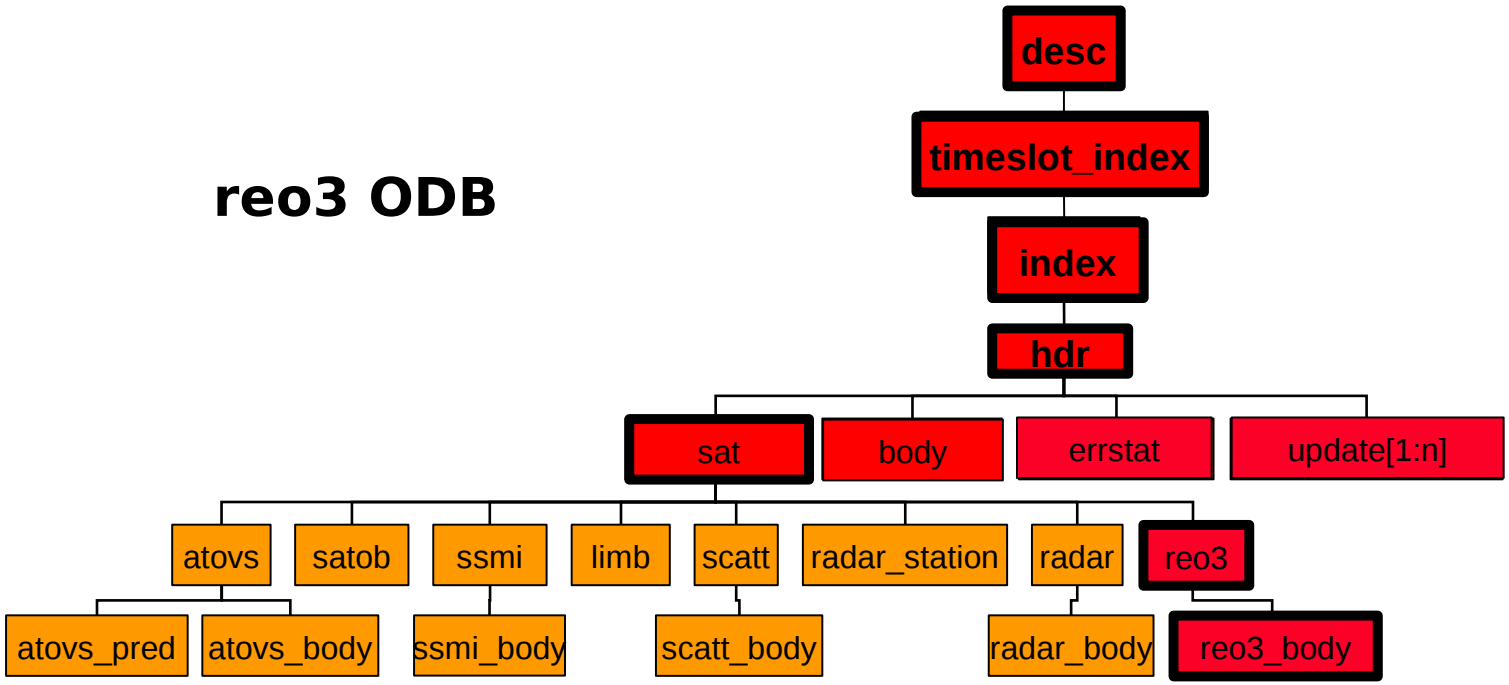
- ECMA defines a small subset of attributes valid for active/passive/blacklisted observations
- CCMA inherits from ECMA attributes and also has specific attributes valid for active observations only (where status.active@hdr=1 and status.active@body=1)
- Some functions apply both to ECMA and CCMA but their behaviour is different



HIERARCHY BETWEEN TABLES

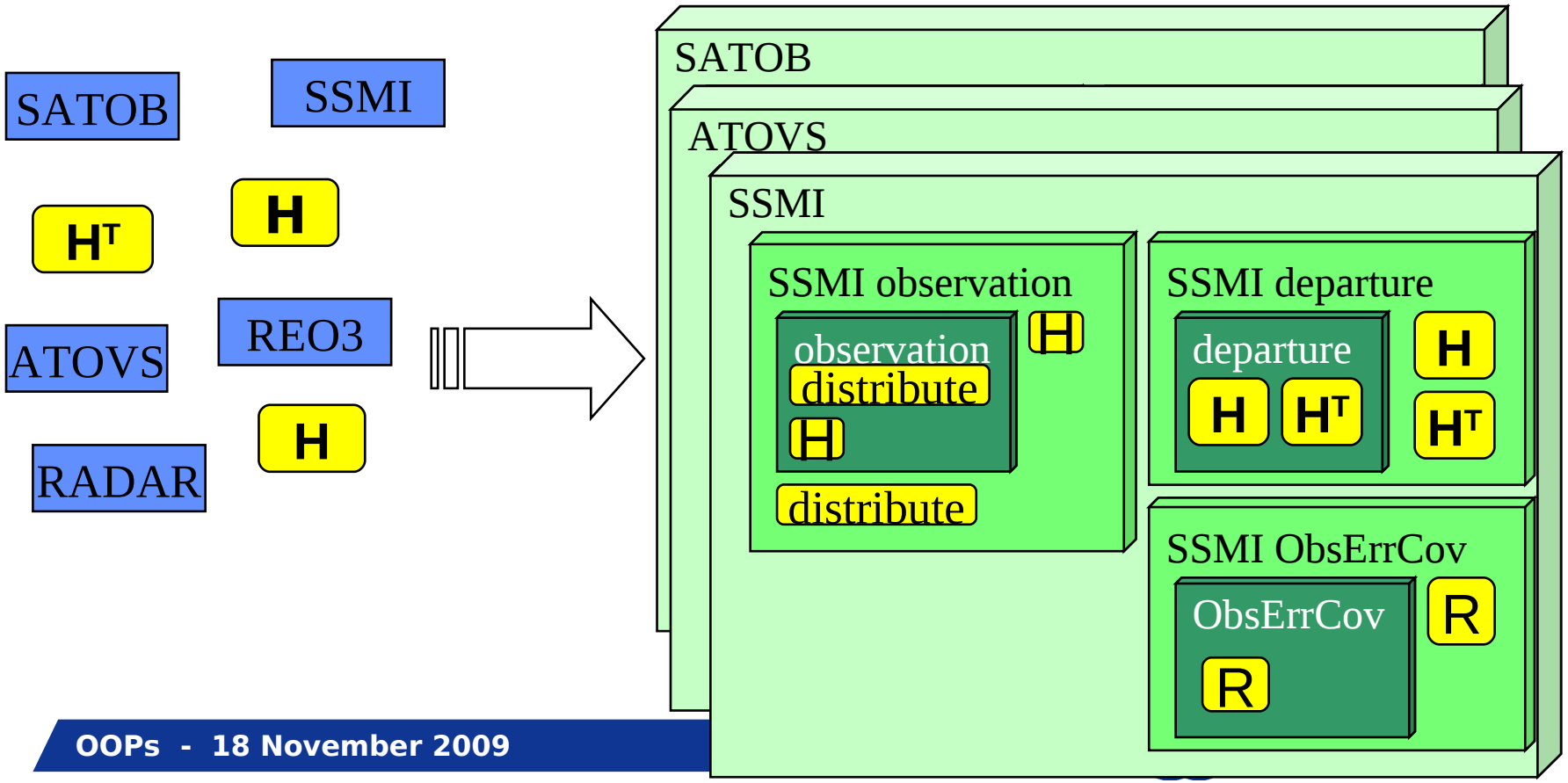
- Some tables are common to all observation types while others are specific (sat, satob, scatt, reo3, etc.)

  → It defines families of attributes: as a child of reo3, reo3_body "inherits" all attributes of reo3, sat, hdr, index, timeslot_index and desc)



Polymorphism

- ODB retrievals in H (hop.F90), H (hoptl.F90), H^T (hopad.F90) depend on the observation type (see ctxinitdb.F90)
- OpenMP loop over observation type and each operator has a behaviour depending on the observation type



Conclusion and further work

- The current ODB/IFS interface tries to “mimic” an object-oriented approach
  - A “real” object-oriented implementation would simplify the code (no IF statements); adding new observation types would be much easier
- Which object-oriented language for the ODB/IFS interface?

→ No preferences but it has to be the same language as IFS.

→ C++ was chosen for archiving ODBs (MARS server is in C++).

- Toy system to be extended to test various possibilities for the ODB/IFS interface:
  - With different kind of observations and their associated observation operator, observation error covariance, etc.
  - With parallel queries
- Then we can start with IFS…