4D-Var, Scalability and Code Design

Yannick Trémolet

Mike Fisher, Deborah Salmond, Anne Fouilloux, Tomas Wilhelmsson, ...

ECMWF

1 November 2010

Dedicated to Joe Sela

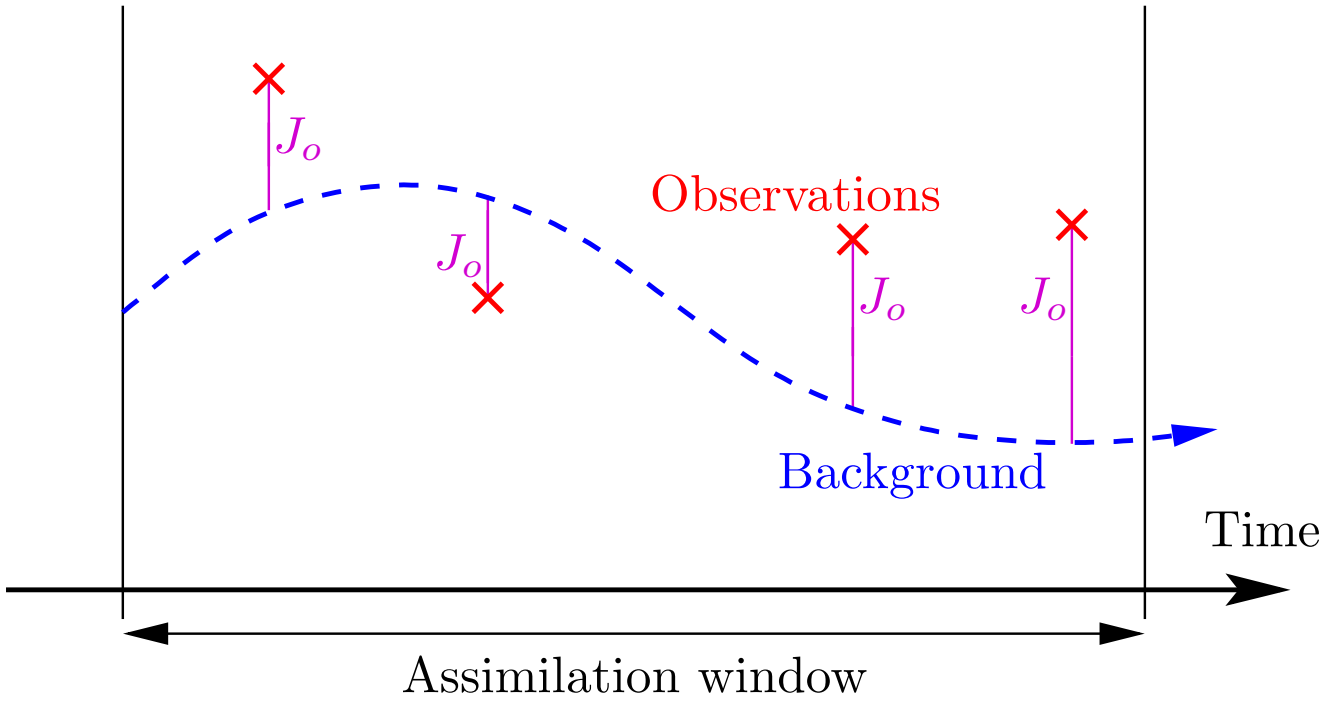
A Concern: Scalability

- The supercomputers of the next generation are expected to have 10,000s of processors or more.
- Scalability has become a major concern for our applications.
- In the current operational configuration, the 4D-Var data assimilation system does not scale very well:

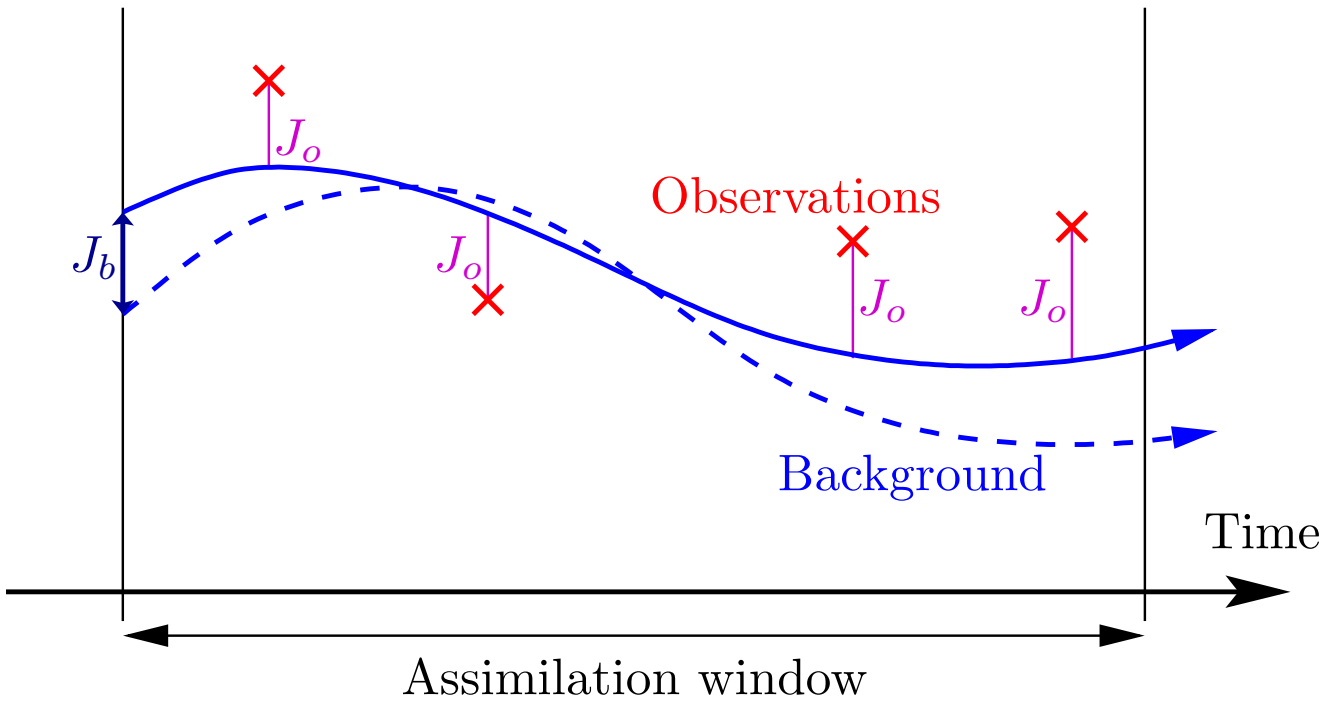
Resolution	Grid points	Threads	GP/thread	Halo width
T159	35718	3072	11	7
T255	88838	3072	29	11

- How can the scalability of 4D-Var be improved?

What is 4D-Var?



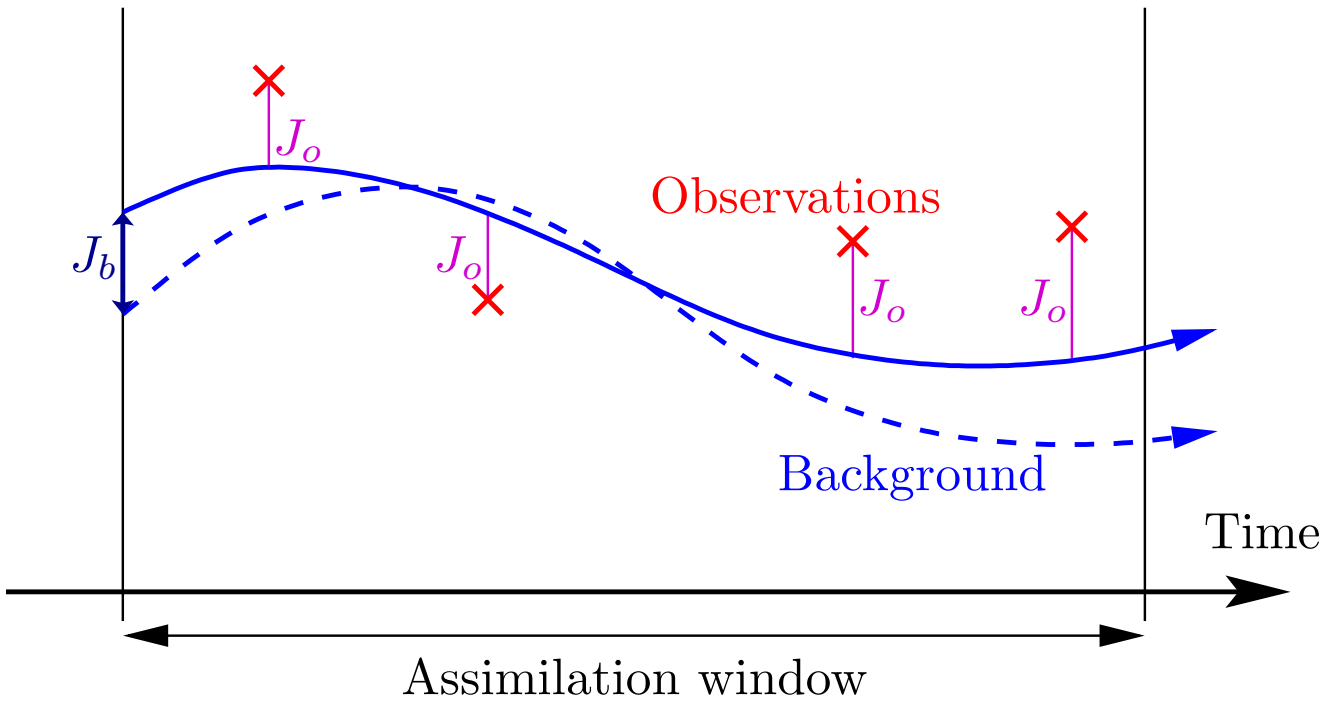
What is 4D-Var?



- The 4D-Var cost function is:

$$J(\mathbf{x}) = \frac{1}{2}\sum_{i=0}^{n}[\mathcal{H}(\mathbf{x}_i) - \mathbf{y}_i]^T \mathbf{R}_i^{-1}[\mathcal{H}(\mathbf{x}_i) - \mathbf{y}_i] + \frac{1}{2}(\mathbf{x}_0 - \mathbf{x}_b)^T \mathbf{B}^{-1}(\mathbf{x}_0 - \mathbf{x}_b)$$

What is 4D-Var?



- The 4D-Var cost function is:

$$J(\mathbf{x}) = \frac{1}{2}\sum_{i=0}^{n}[\mathcal{H}(\mathbf{x}_i) - \mathbf{y}_i]^T \mathbf{R}_i^{-1}[\mathcal{H}(\mathbf{x}_i) - \mathbf{y}_i] + \frac{1}{2}(\mathbf{x}_0 - \mathbf{x}_b)^T \mathbf{B}^{-1}(\mathbf{x}_0 - \mathbf{x}_b)$$

- It is minimized using an iterative algorithm.

The 5 Dimensions of 4D-Var

- The bulk of the 4D-Var algorithm comprises 5 nested loop directions:

  1. Minimisation algorithm iterations (inner and outer),
  2. Time stepping of the model (and TL/AD),
  3. Latitude,
  4. Longitude,
  5. Vertical.

The 5 Dimensions of 4D-Var

- The bulk of the 4D-Var algorithm comprises 5 nested loop directions:

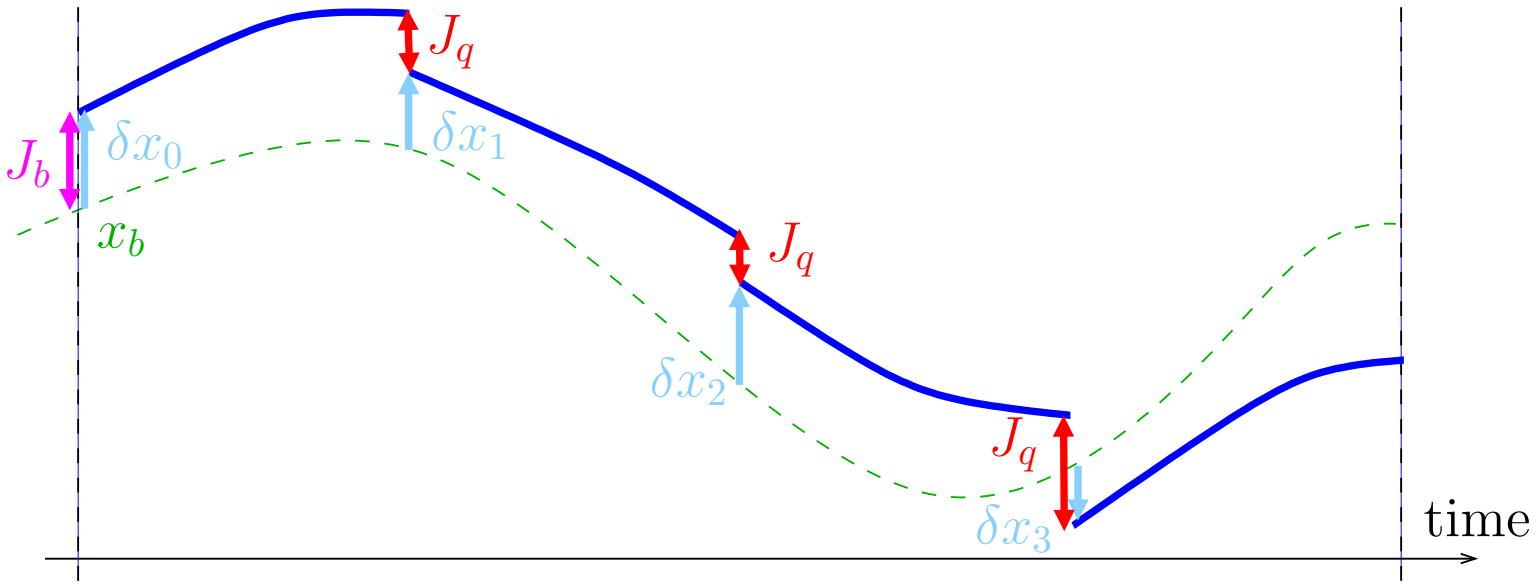
  1. Minimisation algorithm iterations (inner and outer),
  2. Time stepping of the model (and TL/AD),
  3. Latitude, `NPROMA`
  4. Longitude, `NPROMA`
  5. Vertical.

- Only two are parallel!

The 5 Dimensions of 4D-Var

- The bulk of the 4D-Var algorithm comprises 5 nested loop directions:
  1. Minimisation algorithm iterations (inner and outer),
  2. Time stepping of the model (and TL/AD),
  3. Latitude, `NPROMA`
  4. Longitude, `NPROMA`
  5. Vertical.
- Only two are parallel!
- We need to look at the other directions for more parallelism, for example:
  - Minimisation algorithm:
    - ⋆ Parallel search directions,
    - ⋆ Parallel preconditioner and less iterations,
    - ⋆ Observation space algorithms, saddle point algorithms.
  - Time stepping:
    - ⋆ Weak constraint 4D-Var.
- Scalability cannot be improved solely by technical or local optimizations!

An example: Weak Constraint 4D-Var



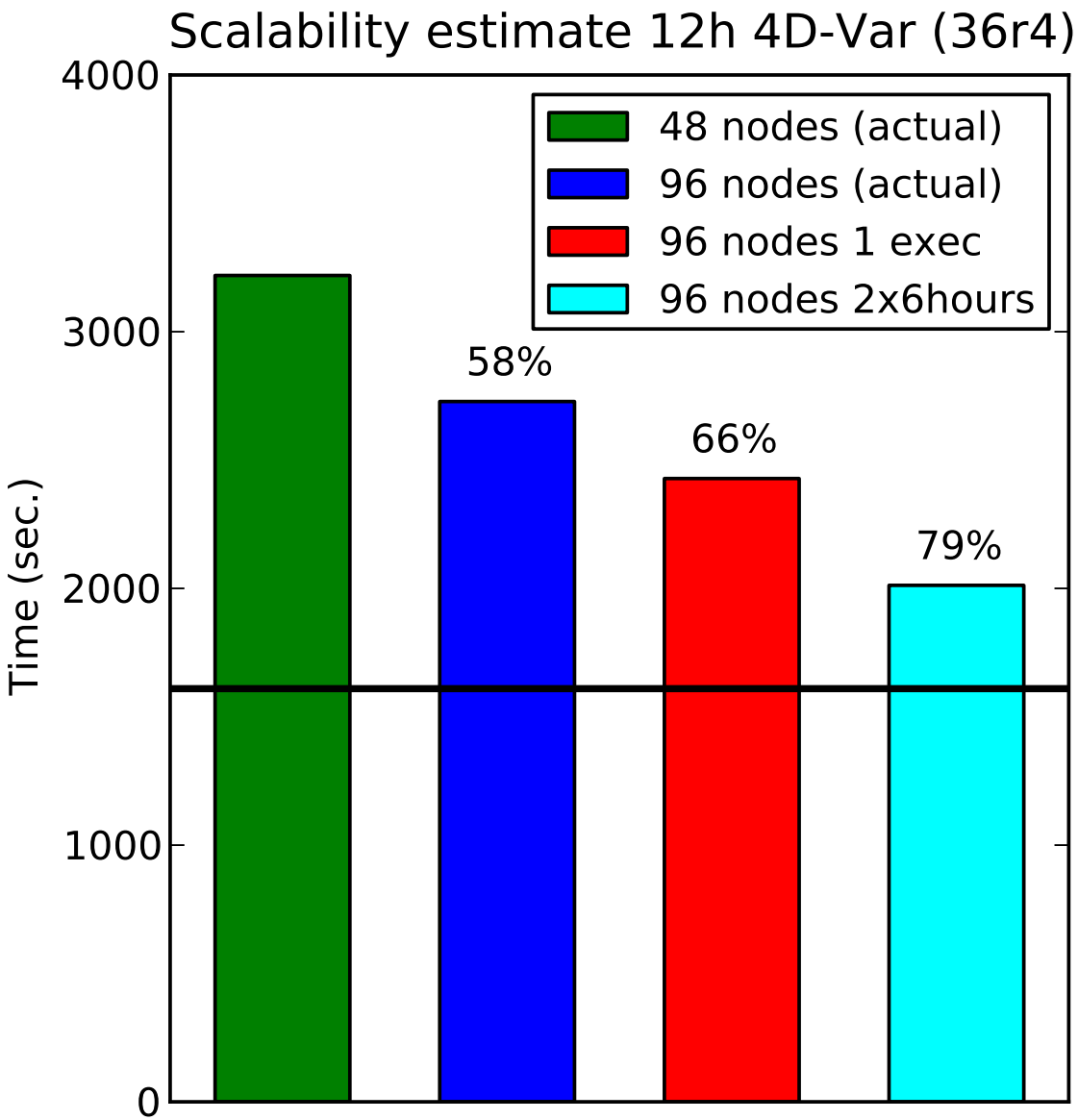
- Model integrations within each time-step (or sub-window) are independent:
  - Information is not propagated across sub-windows by TL/AD models,
  - $\mathcal{M}$ and $\mathcal{H}$ can be run in parallel over the sub-windows.
- Several shorter 4D-Var cycles are coupled and optimised together.
- 4D-Var becomes an elliptic problem and preconditioning becomes more complex.

Scalable Minimization Algorithms

- Parallel search directions: better for problems where many iterations are performed.
- Saddle point algorithms (Lagrange multiplier approach).
- Parallel preconditioner and less iterations.
- In the weak constraint formulation, the preconditioning is more complex but could be more parallel.

- In all cases, exploring parallelism in new directions, through minimization algorithms or weak constraint 4D-Var or both requires dramatic changes in the high level data assimilation algorithm.
- Evaluating these options requires a very flexible code.

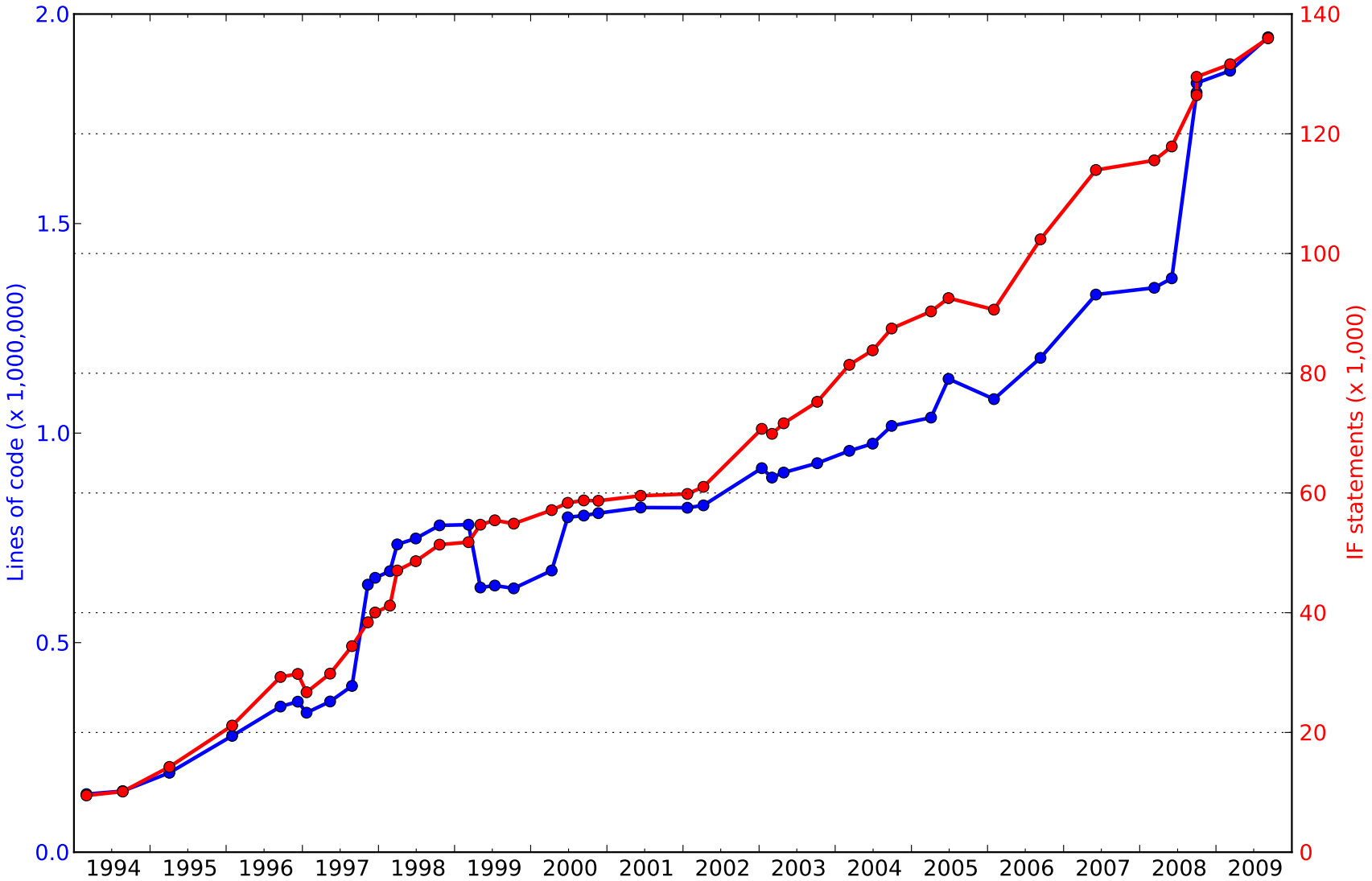
An Example of Better Scalability in 4D-Var



- One executable instead of 7 reduces I/O and start-up costs.
- Weak constraints 4D-Var with a split window gives access to more parallelism.

Figure from Deborah Salmond

Another concern: Code complexity



It means growth of maintenance and development costs.

The Needs for a Flexible Code

- The IFS is a very good global weather forecasting system. However, continuous improvements are necessary to stay at the forefront.
- Scalability has become a major concern in view of new computer architectures.
- The IFS code is more than twenty years old and, over this period, has reached a very high level of complexity. Such complexity is becoming a barrier to new scientific developments.
- The maintenance cost has become very high and new releases take longer and longer to create and debug.
- Code flexibility will reduce the learning curve for new scientists and visitors and enable a more efficient collaboration with external groups.
- There is more uncertainty in scientific methods that will be used in the future, in particular in the area of data assimilation.

Example: Data Assimilation

- Data assimilation aims at finding the best estimate (analysis) of the state of the atmosphere (or system of interest) given a previous estimate of the state (background) and recent observations of the system.

$$J(\mathbf{x}) = \frac{1}{2}(\mathbf{x}_0 - \mathbf{x}_b)^T \mathbf{B}^{-1}(\mathbf{x}_0 - \mathbf{x}_b) + \frac{1}{2}\sum_{i=0}^{n}[\mathcal{H}(\mathbf{x}_i) - \mathbf{y}_i]^T \mathbf{R}_i^{-1}[\mathcal{H}(\mathbf{x}_i) - \mathbf{y}_i]$$

- The 4D-Var problem, and the algorithm to solve it, can be described with a very limited number of entities:
  - Vectors: $\mathbf{x}$, $\mathbf{y}$, $\mathbf{g}$, $\delta\mathbf{x}$ and χ.
  - Covariances matrices: $\mathbf{B}$, $\mathbf{R}$ (and eventually $\mathbf{Q}$).
  - Two operators and their linearised counterparts: $\mathcal{M}$, $\mathbf{M}$, $\mathbf{M}^T$, $\mathcal{H}$, $\mathbf{H}$, $\mathbf{H}^T$.

- All data assimilation schemes manipulate the same limited number of entities (KF, EnKF, PSAS...).

- For future developments these entities should be easily available and reusable.

What is modularity?

- Decomposability: break the problem into small enough independent less complex subproblems.
- Composability: elements can be freely combined to produce a new system.
- Understandability: elements can be understood without knowing the others.
- Continuity: a small change in the problem specification triggers a change in just one (or few) module(s).
- Protection: an error does not propagate to other modules.
- Some rules for modularity:
  - Few interfaces,
  - Small interfaces,
  - Explicit (and well documented) interfaces,
  - Information hiding.

Modularity and Object Oriented Programming

- Many programmers had the same concerns before us.
- The general technique that has emerged in the software industry to address the needs for flexibility, reusability, reliability and efficiency is called object-oriented programming.
- Does it make sense to rethink the design of the IFS in that framework?
- To answer this question we have started to develop an Object Oriented Prediction System (OOPS) with simple models.

Programming Languages

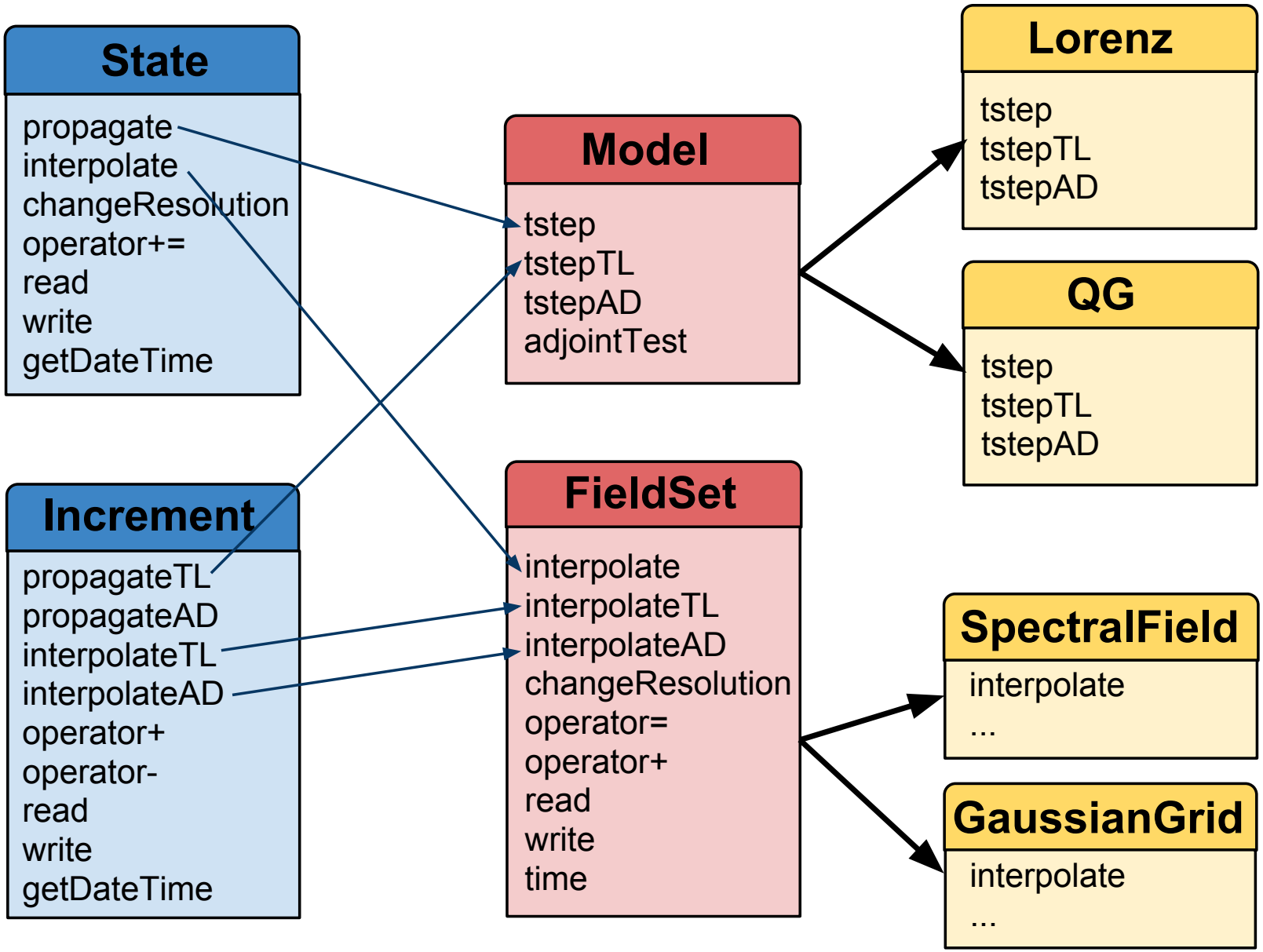
- Fortran 2003:
  - Compilers are becoming available but we have been debugging them.
  - The OO aspects are limited but enough for a scientific code.
  - `SELECT TYPE` construct is a cumbersome equivalent for (dynamic) cast.
  - User defined constructors are missing.
  - Fortran 2008 looks promising but when will we have it? (Co-Arrays exist at least since 1996.)
  - Easy-ish transition from existing code (at the risk of adopting existing solutions only because it is easy).
  - False impression of knowing the language.

- C++:
  - Widely used language even for supercomputing (outside meteorology).
  - Compilers are available, widely used and debugged.
  - Syntax takes getting used to (for Fortran programmers). Transition is sligthly more complex and might require training for some staff (but few would actually see the C++ layer).
  - More mature and flexible (OO aspects and memory management).

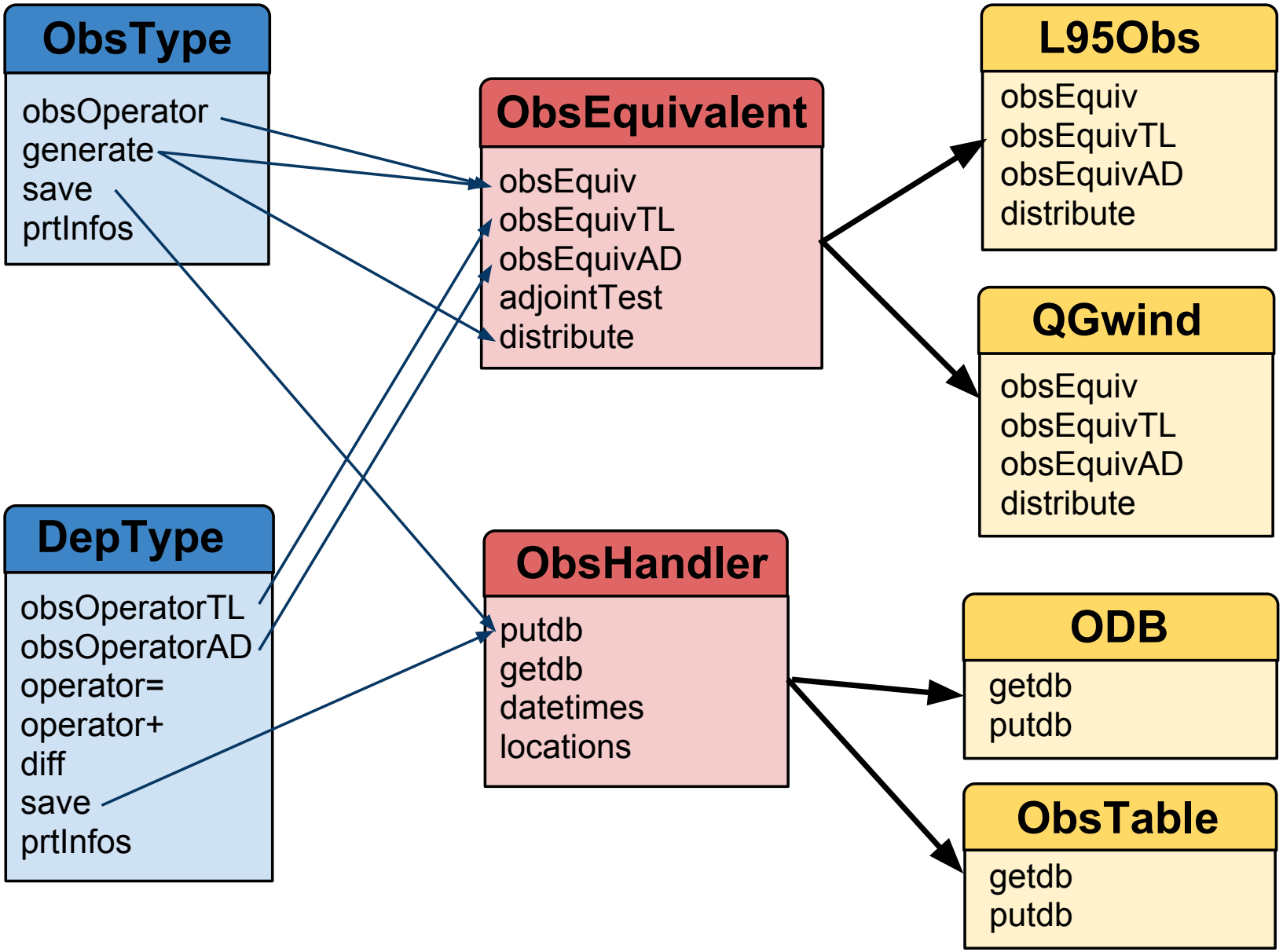
OOPS: Basic classes

- States:
  - Input, output (raw or post-processed), copy, assign.
  - Interpolate.
  - Move forward in time (propagate using the model).
- Increments:
  - Basic linear algebra operators,
  - Evolve forward in time linearly and adjoint (propagateTL, propagateAD).
  - Add to state.
- Observations:
  - Input, output, copy, assign.
  - Compute observation equivalent from a state (observation operator).
- Departures:
  - Compute as difference between observations,
  - Compute as linear variation from an increment (obsOperatorTL/AD).
- Covariance matrices:
  - Multiply by matrix, its inverse and/or square root.
- We don't need to know how these operations are performed, how the states are represented or how the observations are stored (ODB or other), or if covariance matrices are stored or implemented as a set of operators.

State related classes



Observation related classes



OOPS: Building-up a data assimilation system

- These classes form the basic building blocks for OOPS and already make the system flexible from the application point of view.
- The basic classes described previously can be used to build any data assimilation system.
- For example, for an incremental 4D-Var algorithm:
  - Observer (J_o):
    - ⋆ Iterate over time-steps, compute observation equivalent and move the state forward.
  - Control variables, control vectors and change of variable:
    - ⋆ To speed-up the minimisation, it is preconditionned by the square root of $\mathbf{B}$, the unknown variable is a non-physical control vector.
  - Cost function:
    - ⋆ Takes a control vector in input, does a change of variable, computes the cost function in physical space and performs the adjoint to obtain the gradient (control vector).
  - Minimisation algorithm:
    - ⋆ Abstract algorithm manipulating abstract vectors and a function to be minimised that takes a vector as input and returns its gradient as a vector.

From IFS to OOPS

- The main idea is to keep the computational parts of the existing code and reuse them in a re-designed flexible structure.
- This can be achieved by a top-down and bottom-up approach.
- From the top: Develop a new, modern, flexible structure (C++).
- From the bottom: Progressively create self-contained units of code (Fortran).
- Put the two together: Extract self-contained parts of the IFS and plug them into OOPS.
- Note that:
  - This can be done with other models.
  - The OO layer developed for the simple models is re-used.

There is a limit to Scalability

- Currently in 4D-Var, the dimensions are:
  - Minimisation algorithm iterations: $\mathcal{O}(100)$
  - Time stepping of the model: $\mathcal{O}(50)$
  - Latitude and longitude: $\mathcal{O}(100 \times 100)$
  - Vertical: $\mathcal{O}(150)$.

There is a limit to Scalability

- Currently in 4D-Var, the dimensions are:
  - Minimisation algorithm iterations: $\mathcal{O}(100)$ 3-5?
  - Time stepping of the model: $\mathcal{O}(50)$ 4-10
  - Latitude and longitude: $\mathcal{O}(100 \times 100)$
  - Vertical: $\mathcal{O}(150)$. Unexplored!

10 to 50 times more processors than today.

There is a limit to Scalability

- Currently in 4D-Var, the dimensions are:
  - Minimisation algorithm iterations: $\mathcal{O}(100)$ 3-5?
  - Time stepping of the model: $\mathcal{O}(50)$ 4-10
  - Latitude and longitude: $\mathcal{O}(100 \times 100)$
  - Vertical: $\mathcal{O}(150)$. Unexplored!

  10 to 50 times more processors than today.

- Comment: Ensemble Kalman Filters:
  - Scales with the number of members (for free).
  - Overall cost might be prohibitive?

  $\mathcal{O}(100)$ times more processors than today.

There is a limit to Scalability

- Currently in 4D-Var, the dimensions are:
  - Minimisation algorithm iterations: $\mathcal{O}(100)$ 3-5?
  - Time stepping of the model: $\mathcal{O}(50)$ 4-10
  - Latitude and longitude: $\mathcal{O}(100 \times 100)$
  - Vertical: $\mathcal{O}(150)$. Unexplored!

  10 to 50 times more processors than today.

- Comment: Ensemble Kalman Filters:
  - Scales with the number of members (for free).
  - Overall cost might be prohibitive?

  $\mathcal{O}(100)$ times more processors than today.

- The limit for scalability is of the same order of magnitude.

- In any case, scalability beyond 10,000s processors will be difficult to achieve for any data assimilation system.

There is a limit to Scalability

- Currently in 4D-Var, the dimensions are:
  - Minimisation algorithm iterations: $\mathcal{O}(100)$ 3-5?
  - Time stepping of the model: $\mathcal{O}(50)$ 4-10
  - Latitude and longitude: $\mathcal{O}(100 \times 100)$
  - Vertical: $\mathcal{O}(150)$. Unexplored!

  10 to 50 times more processors than today.

- Comment: Ensemble Kalman Filters:
  - Scales with the number of members (for free).
  - Overall cost might be prohibitive?

  $\mathcal{O}(100)$ times more processors than today.

- The limit for scalability is of the same order of magnitude.

- In any case, scalability beyond 10,000s processors will be difficult to achieve for any data assimilation system.

- The quality of the analysis and forecast determine the choice of algorithm, within our computing and operational constraints.

OOPS Benefits

- Introduce object oriented where most of the flexibility can be expressed: at the high level of the code structure.
- Because components are independent, complexity descreases.
  - Components can easily be developed in parallel.
  - Less bugs and easier debuging.
- Improved flexibility:
  - Explore and improve scalability.
  - Develop new data assimilation science.
  - Adding new observations types is easier.
  - Changes in one application do not affect other applications.
  - Ability to handle different models opens the door for coupled DA.
- Simplified systems are very useful to understand concepts and validate ideas.
  - It is possible to move to the full system without re-writing code (for algorithms that can be written at the abstract level).
- The cost of developing the system is compensated by increased productivity.
  - Development time (total human ressource and elapsed),
  - Computer time (less experiments because there are less errors).