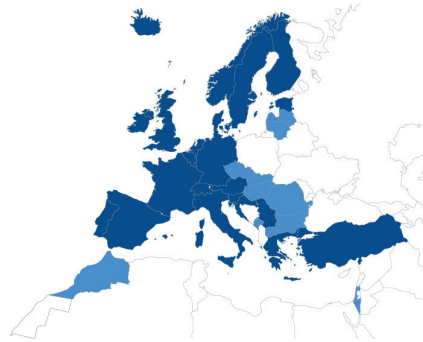
Data assimilation as viewed by a programmer

Olivier Marsden
olivier.marsden@ecmwf.int
European Centre for Medium-Range Weather Forecasts

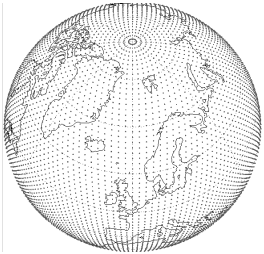
ECMWF

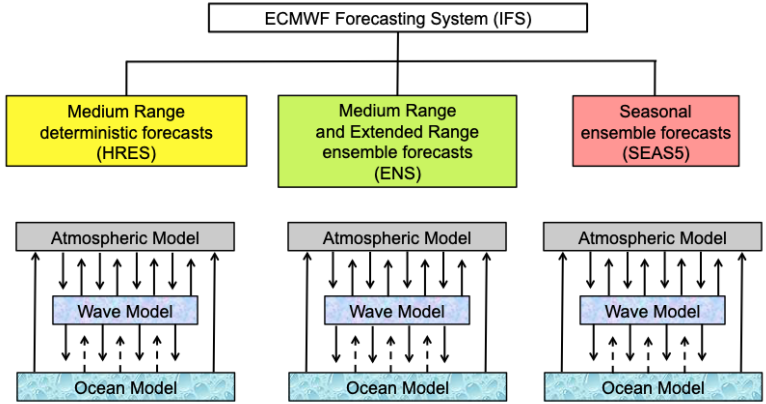
The European Centre for Medium-Range Weather Forecasts

- Independent intergovernmental organisation
- Established in 1975, today supported by **23 member** and **11 cooperating states**
- Headquarters in **Reading (UK)**
- Data center in **Bologna (IT)** (almost finished); Offices in **Bonn (DE)** (later this year)
- Research institute and 24/7 operational service:
  - produce and disseminate NWP
  - operate meteorological data archive
  - implement Copernicus services CAMS and C3S
  - provide computing resources to member states



Configurations





Forecast period	10 days	15 days / 46 days	7 months / 13 months
Ensemble members	1	51	51 / 15
Frequency	2x daily	2x daily / 2x weekly	monthly / quarterly
Resolution	TCo1279L137 (9km)	TCo639L137 (18km) / TCo319L137 (36km)	TCo319L91 (36km)

What is data assimilation?

Given a sytem to be analyzed,

- data assimilation aims to produce best possible estimate of the system's state
- making use of different sources of knowledge about a system
- typically two sources will be observations and models
- used throughout earth-system disciplines (atmospheric, oceanographic, land surface, vegetation index, ...)
- In real life, "best estimate" is inexact due to uncertainties:
  - models are imperfect (simplifications, errors in parameters, in inputs)
  - measurements are insufficient and imperfect (random errors, biases, correlated errors, observation operator)

Aim in meteorology

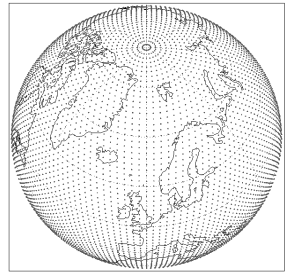
Establish estimate of the discretized atmospheric state at a given time ($\mathbf{x}$, $\mathbf{x} \in \Re^{N_x}$)

with quantification of the uncertainty of the estimate

using observations in a time window ($\mathbf{y}$, $\mathbf{y} \in \Re^{N_y}$),

a background state $\mathbf{x}^b$,

and a model of state evolution M, $\mathbf{x}(t + dt) = M(\mathbf{x})$



Aim in meteorology

Establish estimate of the discretized atmospheric state at a given time ($\mathbf{x}$, $\mathbf{x} \in \Re^{N_x}$)

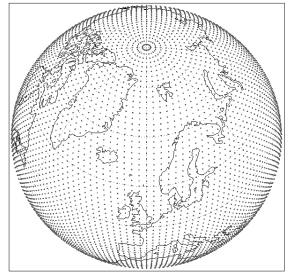
with quantification of the uncertainty of the estimate

using observations in a time window ($\mathbf{y}$, $\mathbf{y} \in \Re^{N_y}$),

a background state $\mathbf{x}^b$,

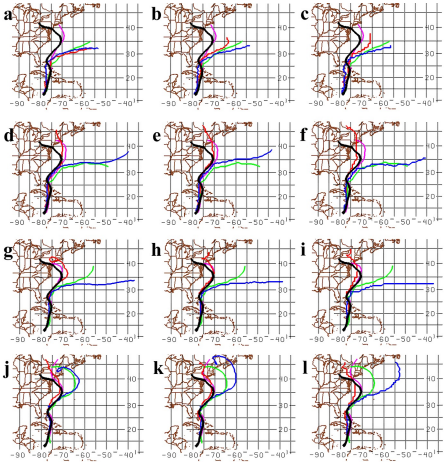
and a model of state evolution M, $\mathbf{x}(t + dt) = M(\mathbf{x})$

Do it quickly!



Is it a big deal ?

Hurricane Sandy (2012)



Initialization times of (a–c) 0000 UTC 23 October, (d–f) 1200 UTC 23 October, (g–i) 0000 UTC 24 October, and (j–l) 1200 UTC 24 October. ECMWF (pink), GFS (green), TWRF (red), and SWRF (blue) tracks are shown in addition to Sandy's best track (black).

Bassill, *GRL* 41(9) 2014

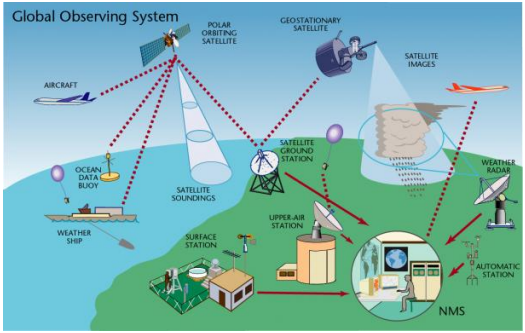
Problem size

- current ECMWF operational grid resolution (TCO1279) : $\simeq 9$ km, 137 vertical levels
- $\simeq 7 \times 10^8$ atmospheric grid points
- initial value to be specified for all prognostic variables at every grid point
- for ensemble prediction suite, this is to be done for each ensemble member

Observations

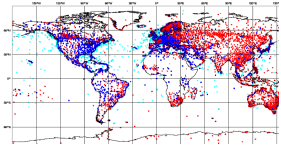
The atmosphere is continually being measured all over the world

- strong global collaboration on sharing atmospheric observations
- World Meteorological Organisation coordinates data handling and sharing

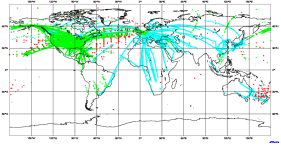


Observations

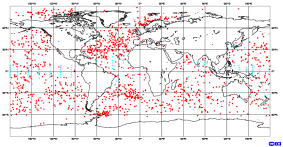
In situ observations



Synop

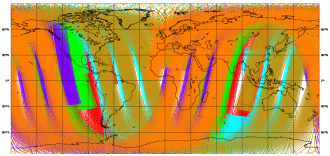


Aircraft

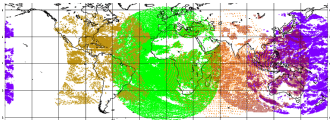


Buoys

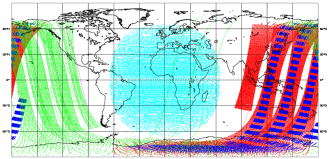
Satellite



Sounder



Geostationary



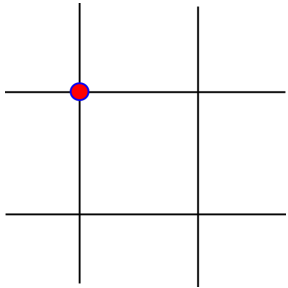
Ozone

Observations

- most observations are not located at a grid point
- most (satellite) observations are not of model variables (for ex. radiances)
- observation operators are used to generate equivalent observation values (same quantity, same location) from model values $y = \mathcal{H}(\mathbf{x})$
- observation operators range from very simple to very complicated

Observations

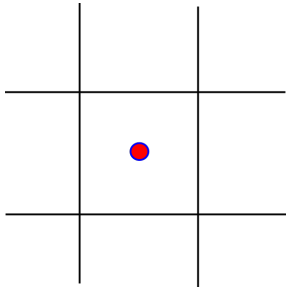
Temperature observation example



$$\mathcal{H} = \begin{bmatrix} 0 \\ \dots \\ \dots \\ \dots \\ 0 \\ 1 \\ 0 \\ \dots \\ \dots \\ \dots \\ 0 \end{bmatrix}$$

Observations

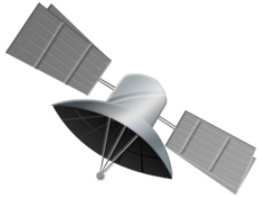
Temperature observation example



$$\mathcal{H} = \begin{bmatrix} 0 \\ \dots \\ 0 \\ 0.25 \\ 0.25 \\ 0.25 \\ 0.25 \\ 0 \\ \dots \\ 0 \end{bmatrix}$$

Observations

Temperature observation example



$$L(\nu) = \int_0^{\infty} B(\nu, T(z)) \left[\frac{d\tau(\nu)}{dz} \right] dz + \text{Surface emission} + \text{Surface reflection/scattering} + \text{Cloud/rain contribution} + \ldots$$

Planck source term* depending on temperature of the atmosphere

Absorption in the atmosphere

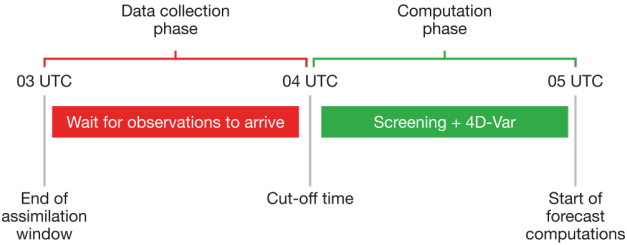
Other contributions to the measured radiances

Observations

- at ECMWF $\mathcal{O}(10^8)$ observations processed daily
- offline quality control of observations :
  - elimination of duplicates
  - thinning
  - blacklisting

Observations

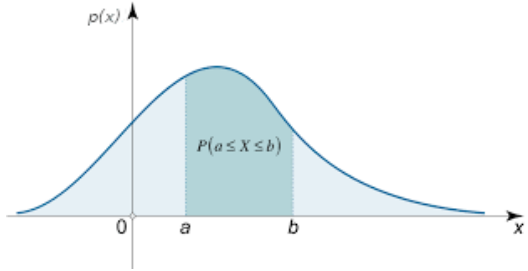
Traditional operational schedule



Bayes' theorem

Formalism for melding different information sources (observations and models) : Bayes Theorem.
With both models and observations being imperfect, they can be viewed as random variables, and described by their Probability Distribution Functions.

For random variable X with density $p(x)$, $P[a \leq X \leq b] = \int_a^b p(x)$



Bayes' theorem

Expresses the link between joint probability $p(A, B)$, conditional probability $p(A|B)$ and marginal probability $p(A)$

$$p(A|B) = \frac{p(B|A)\,p(A)}{p(B)}$$

Bayes' theorem

Expresses the link between joint probability $p(A, B)$, conditional probability $p(A|B)$ and marginal probability $p(A)$

$$p(A|B) = \frac{p(B|A)\,p(A)}{p(B)}$$

In a meteorological context,

$$p(\mathbf{x}|\mathbf{y}) = \frac{p(\mathbf{y}|\mathbf{x})\,p(\mathbf{x})}{p(\mathbf{y})}$$

- $p(\mathbf{x}|\mathbf{y})$ (posterior) pdf of the atmospheric state for given observations
- $p(\mathbf{y}|\mathbf{x})$ pdf of observed values for given state
- $p(\mathbf{x})$ (prior) pdf of state values being predicted by model
- $p(\mathbf{y})$ (marginal) pdf of observed values

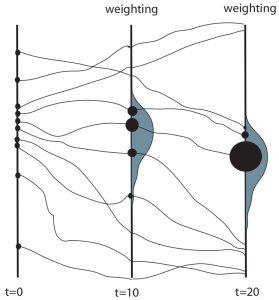
In practice ...

- Estimation of full probabilty functions is intractable, assumptions are needed
- different assimilation methods stem from this :
  - sequential filter methods (particle filter, Kalman)
  - variational methods

Particle filter approach:

try to sample the posterior probability density

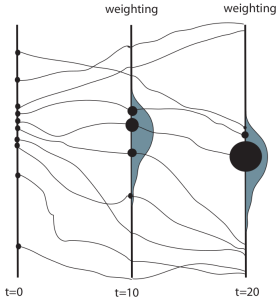
- sample initial pdf of starting state $p(\mathbf{x}) \simeq \sum_i w_i \delta(\mathbf{x} - \mathbf{x}_i), i = 1..N$
- propagate set of “particles” in time with the model
- analysis step : weights are updated from obs $w_i^a \propto w_i p(\mathbf{x}|y_i)$
- resampling : duplication of high-weight particles, elimination of low-weight particles

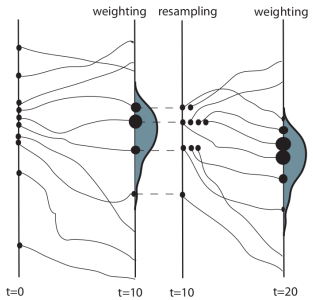


Particle filter approach:

try to sample the posterior probability density

- sample initial pdf of starting state $p(\mathbf{x}) \simeq \sum_i w_i \delta(\mathbf{x} - \mathbf{x}_i), i = 1..N$
- propagate set of “particles” in time with the model
- analysis step : weights are updated from obs $w_i^a \propto w_i p(\mathbf{x}|y_i)$
- resampling : duplication of high-weight particles, elimination of low-weight particles





Sequential filtering

Solve for mean and covariance of posterior probability density
Fun fact : was first implemented to estimate trajectories for the Apollo missions

Sequence of pairs of prediction/forecast steps and analysis/update steps.

- prediction step :

$$\mathbf{x}_i^b = M(\mathbf{x}_{i-1}^a)$$

- analysis step :

$$\mathbf{x}_i^a = \mathbf{x}_i^b + \mathbf{K}_i(H_i(\mathbf{x}_i^b) - \mathbf{y}_i)$$

with

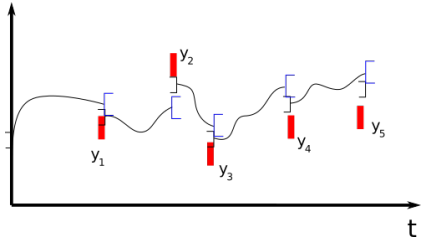
- $\mathbf{K}_i = \mathbf{B}_i\mathbf{H}_i^T(\mathbf{H}_i\mathbf{B}_i\mathbf{H}_i^T + \mathbf{R}_i)^{-1}$
- $\mathbf{B}_i = \{(\mathbf{x}_i - \mathbf{x}_i^b)(\mathbf{x}_i - \mathbf{x}_i^b)^T\}$ is explicitly propagated

Variational assimilation

Goal is to find analysis state $\mathbf{x}^a$ that minimizes distance between its trajectory and observations, while not differing strongly from background.

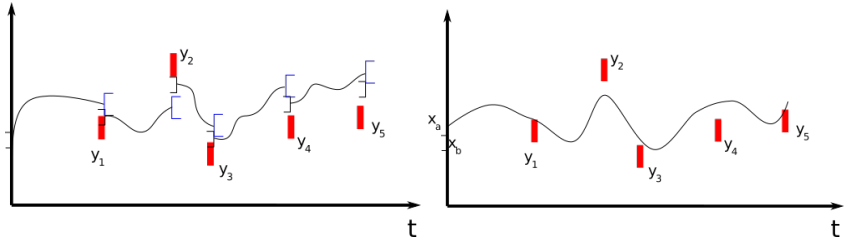
Variational assimilation

Goal is to find analysis state $\mathbf{x}^a$ that minimizes distance between its trajectory and observations, while not differing strongly from background.



Variational assimilation

Goal is to find analysis state $\mathbf{x}^a$ that minimizes distance between its trajectory and observations, while not differing strongly from background.



Variational assimilation

Minimization of a cost function with background term and observation term

$$J(\mathbf{x}(t_0)) = J_B(\mathbf{x}(t_0)) + J_O(\mathbf{x}(t_0))$$

with

$J_B(\mathbf{x}(t_0) = \frac{1}{2}(\mathbf{x}(t_0) - \mathbf{x}_b)^T \mathbf{B}^{-1}(\mathbf{x}(t_0) - \mathbf{x}_b)$

and

$J_O(\mathbf{x}(t_0)) = \frac{1}{2}\sum_i (\mathcal{H}_i \mathcal{M}_{t_0..t_i}(\mathbf{x}(t_0)) - \mathbf{y}_i)^T R_i^{-1} (\mathcal{H}_i \mathcal{M}_{t_0..t_i}(\mathbf{x}(t_0)) - \mathbf{y}_i)$

Variational assimilation

Gradient of the cost function :

$$\nabla_{\mathbf{x}_0} J = \mathbf{B}^{-1}(\mathbf{x}(t_0) - \mathbf{x}_b) + \sum_i \mathbf{M}^T_{t_0..t_i} \mathbf{H}_i^T \mathbf{R}_i^{-1} (\mathcal{H}_i \mathcal{M}_{t_0..t_i}(\mathbf{x}(t_0)) - \mathbf{y}_i)$$

or, by defining innovation vector $\mathbf{d}_i = (\mathbf{y}_i - \mathcal{H}_i \mathcal{M}_{t_0..t_i}(\mathbf{x}(t_0)))$

$$\nabla_{\mathbf{x}_0} J = B^{-1}(\mathbf{x}(t_0) - \mathbf{x}_b) - \sum_i \mathbf{M}^T_{t_0..t_i} \mathbf{H}_i^T \mathbf{R}_i^{-1} \mathbf{d}_i$$

Gradient can be used to inform minimization technique (steepest descent, conjugate gradient, quasi-Newton etc)

Variational assimilation

Schematic 4dvar algorithm :

- compute innovation vector components (obs - obs equivalent at observation time) during **forward model integration**
- multiply each term by $\mathbf{H}_i^T \mathbf{R}_i^{-1}$
- integrate terms back in time with **adjoint model**
- update $\mathbf{x}(t_0)$ accordingly
- repeat until convergence

Variational assimilation

Incremental 4dvar :

Solve iteratively a series of quadratic problems progressively approaching full non-linear one.

- Outer loop (non-linear) stores linearization state (trajectory) and innovations
- Inner loop minimizes linear approximation of non-linear cost-function, with respect to $\delta\mathbf{x} = \mathbf{x} - \mathbf{x}_g$

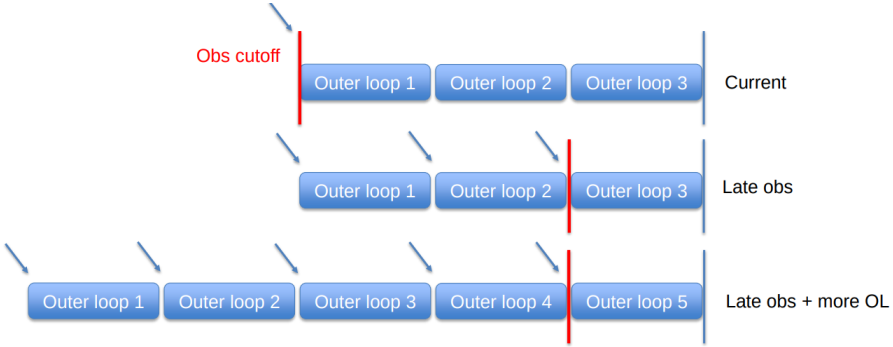
$$J(\delta\mathbf{x})) = \frac{1}{2}[\delta\mathbf{x} - (\mathbf{x}_b - \mathbf{x}_g)]^T\mathbf{B}^{-1}[\delta\mathbf{x} - (\mathbf{x}_b - \mathbf{x}_g)] + \frac{1}{2}\sum_i(\mathbf{H}_i\mathbf{M}_{t_0..t_i}\delta\mathbf{x} - \delta\mathbf{d}_i)^T\mathbf{R}_i^{-1}(\mathbf{H}_i\mathbf{M}_{t_0..t_i}\delta\mathbf{x} - \delta\mathbf{d}_i)$$

Variational assimilation

Incremental 4dvar :

- each inner loop runs to convergence criterion, $\mathbf{x}_g$ is then updated by $\delta\mathbf{x}$
- next outer loop starts by updating trajectory and innovations
- inner loops and outer loops are not necessarily at the same resolution
- initial conditions for forecast are the final analysis x progpagated forward to forecast start time
- at ECMWF 3 outer loops, and typically around 30 inner iterations per outer loop

Variational assimilation



Peter Lean

Variational assimilation

Other things I would have liked to mention

- variational bias control
- ensemble of data assimilations
  - allows the initialisation of the forecast ensemble
  - allows $\mathbf{B}$ matrix to be more than just climatological

Variational assimilation and OOPS

OOPS was designed to allow rapid exploration of variational assimilation algorithms.

It exposes high-level building blocks of assimilation algorithms (*e.g.* State4D, Increment4D, ControlVariable, ControlVector, CostFunction, Minimizer etc) and their associated methods (*e.g.* CostFunction.linearize(), Minimizer.minimize() etc).

Three categories of OOPS classes :

- *assimilation*, e.g. CostFunction, CostFnc3D/4D/4DEnVar/Weak Minimizer, SQRTPLanczos/PCG/PFOM, ...
- *interface*, e.g. Incr/State, (Linear)Model, (Linear)ObsOperator, ...
- *base*, e.g. Accumulator, TrajectorySaver, Observer(TL/AD) ...

Although initially designed with variational assimilation in mind, the classes are appropriate building blocks for other algorithms (forecast (done), singular vector computation (to be done), fsobs (to be done), ...).

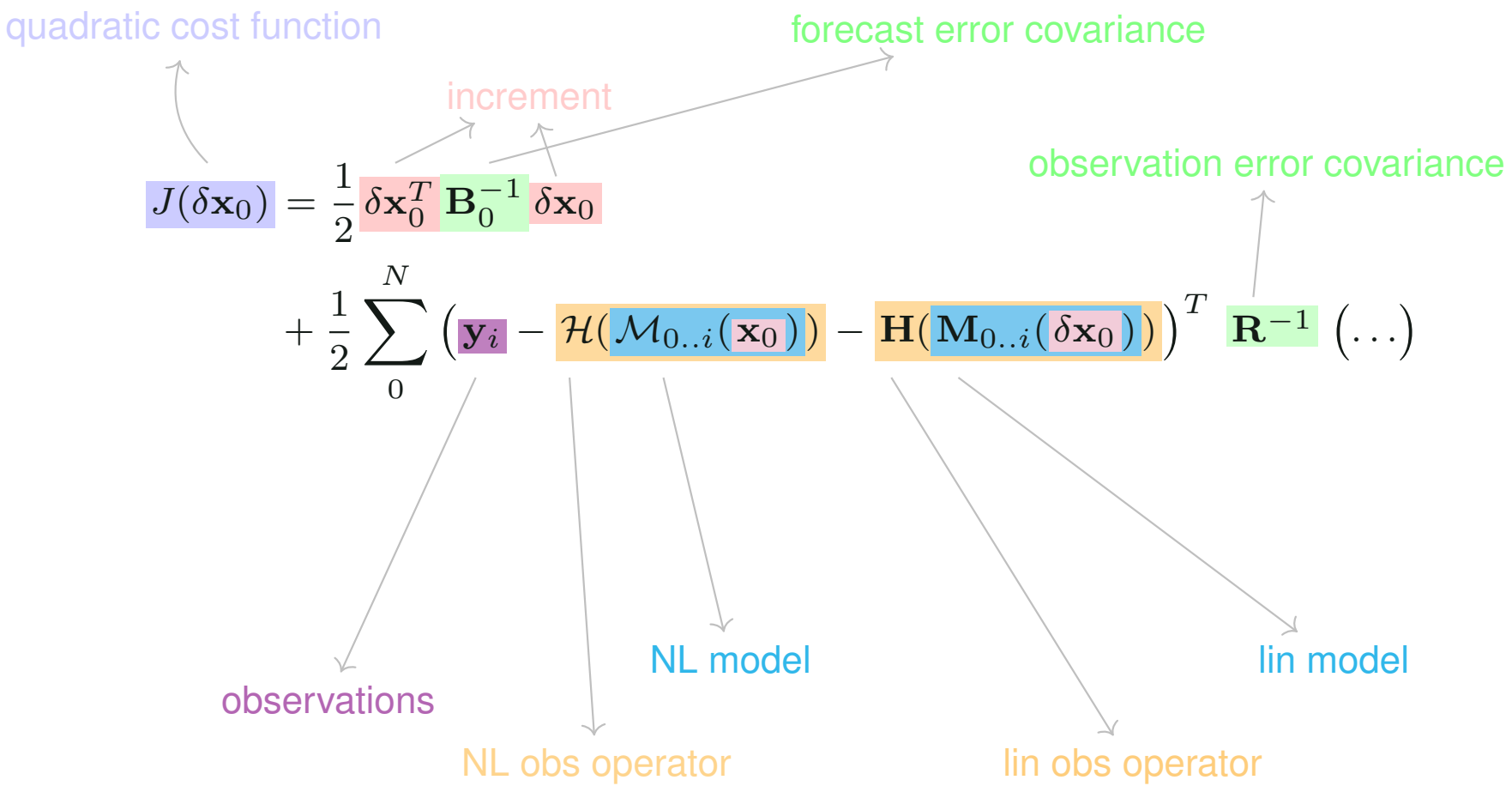
Variational assimilation and OOPS

Illustrative example

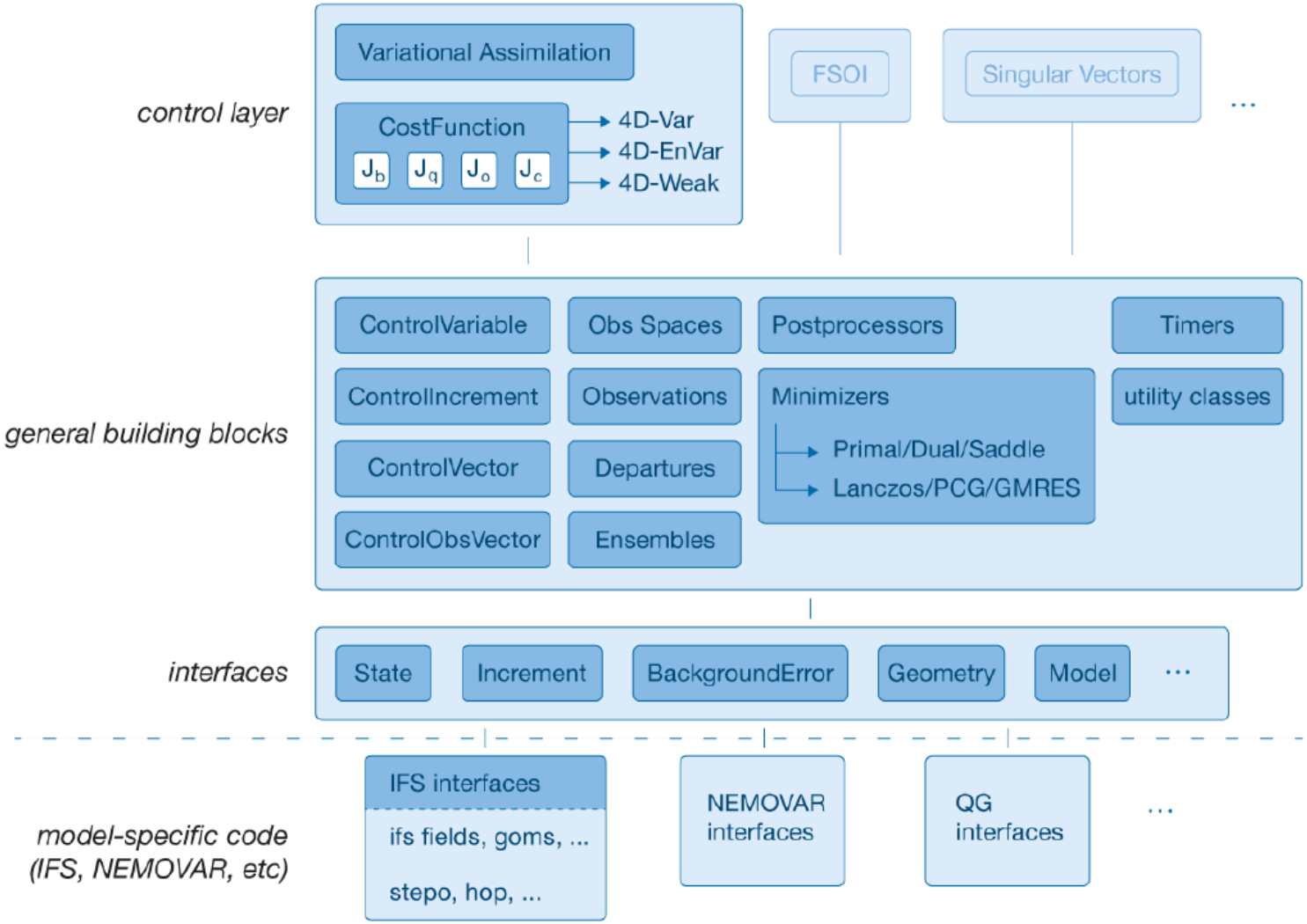
$$J(\delta\mathbf{x}_0) = \frac{1}{2}\,\delta\mathbf{x}_0^T\,\mathbf{B}_0^{-1}\,\delta\mathbf{x}_0$$
$$+\frac{1}{2}\sum_{0}^{N}\Big(\mathbf{y}_i - \mathcal{H}(\mathcal{M}_{0..i}(\mathbf{x}_0)) - \mathbf{H}(\mathbf{M}_{0..i}(\delta\mathbf{x}_0))\Big)^T\,\mathbf{R}^{-1}\,\Big(\ldots\Big)$$

Variational assimilation and OOPS

Illustrative example



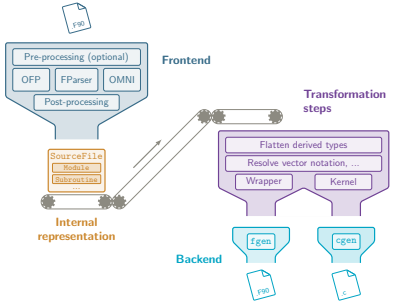
OOPS design

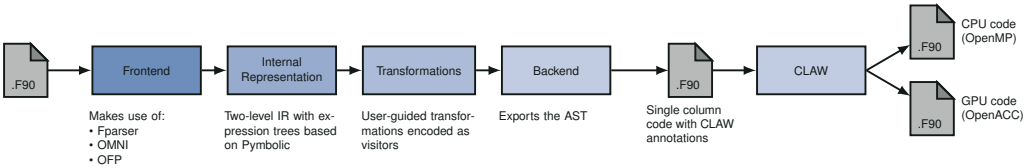

control layer
Variational Assimilation
CostFunction
J_b J_q J_o J_c
4D-Var
4D-EnVar
4D-Weak
FSOI
Singular Vectors
...
general building blocks
ControlVariable
Obs Spaces
Postprocessors
Timers
ControlIncrement
Observations
Minimizers
utility classes
Primal/Dual/Saddle
Lanczos/PCG/GMRES
ControlVector
Departures
ControlObsVector
Ensembles
interfaces
State
Increment
BackgroundError
Geometry
Model
...
model-specific code
(IFS, NEMOVAR, etc)
IFS interfaces
ifs fields, goms, ...
stepo, hop, ...
NEMOVAR
interfaces
QG
interfaces
...

Loki - Freely programmable source-to-source translation

Transforming large sub-trees of a complex code base

- Two-step GPU transformation for IFS physics
  - Auto-extract single-column code via Loki
  - Use **bespoke knowledge of IFS code** (eg. variable names, data layouts, etc.)
  - Generate CPU/GPU parallelisation via CLAW
- Encode transformations as preprocessing step
  - Integration alongside scientific development
  - Restrict changes to defined sub-branches
  - Composable with downstream tools





GPU code generated by source-to-source tools

SCA: Pure single-column Fortran kernel format

- Restrict science code from accessing horizontal dimension
- Insert parallelism in CPU/GPU-friendly ways via CLAW
- Assume the following *original(!)* kernel signature

```
subroutine kernelA(nlev, nproma, array, ...)
  integer, intent(in) :: nlev, nproma
  real, intent(inout) :: arr1d(nproma)
  real, intent(inout) :: arr2d(nlev, nproma)
```

```
!--- Original CPU format  --

do jl=1, nproma
  arr2d(jl,1) = arr1d(jl)
end do

do jk=1, klev
  do jl=1, nproma
    arr2d(jl,jk) = <something>
  end do
end do

! Nested kernel call
call kernelB(..., arr2d(:,:))

!----------------------
```

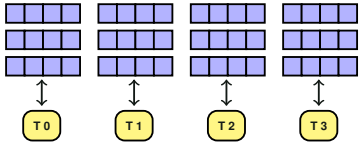
```
!--- SCA format from Loki  --

! Simple arithmetic
arr2d(1) = arr1d

! Vertical loop
do jk=1, klev
  array(jk) = <something>
end do

! Nested kernel call
call kernelB(..., arr2d(:))

!----------------------
```

```
!--- GPU format from CLAW  --
!$acc loop gang
do jl=1, nproma
  arr2d(jl,1) = arr1d(jl)

  do jk=1, klev
    arr2d(jl,jk) = <something>
  end do

  ! Nested kernel call
  call kernelB(..., arr2d(jl,:))
end do
!----------------------
```

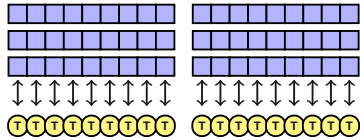
Memory accesses on CPU vs. GPU

Example structure for 2D column physics (eg. CLOUDSC)

- Horizontal dimension is data parallel; vertical is sequential!
- "NPROMA" memory blocking exposes block-stride memory alignment
- **CPU:** Thread-parallel outer loop, vectorised inner loops
- **GPU:** Same data layout, but loop-nest inverted => **coalesced memory accesses!**




```
real :: array(NPROMA,NLEVELS,NBLOCK)

do bl = 1, NBLOCKS  ! <= thread-parallel
  ! <block data association via subroutine call>
  do lev = 1, NLEVELS  ! <= sequential
    do jl = 1, NPROMA  ! <= vectorised
      array(jl, lev) = <expression>
      ...
    end do
  end do
end do
```



```
real :: array(NPROMA,NLEVELS,NBLOCK)

do bl = 1, NBLOCKS  ! <= device-offload
  ! <block data association via subroutine call>
  do jl = 1, NPROMA  ! <= thread-parallel
    do lev = 1, NLEVELS  ! <= sequential
      array(jl, lev) = <expression>
    ...
    end do
  end do
end do
```

Another approach: Single-column coalesced (SCC)

Direct CPU-to-GPU transformation in Loki

```
!--- Loki-CLAW driver format  --
!$acc update(...)

do ibl=1, nblocks
  call kernel(point(:,ibl), column(:,:,ibl))
end do

!$acc end update


!--- Loki-CLAW kernel format  --
subroutine kernel(nproma,nlev,point,column)
  real :: column_tmp(nproma,nlev)


  !$acc parallel loop gang
  do jl = 1, nproma
    do lev = 2, nlev
      column_tmp(jl,lev) = <f(point(jl),
                            column(jl,lev-1))>
    end do

    do lev = 1, NLEVELS
      column(jl,jl) = <g(column_tmp(lev), ...)>
    end do
  end do
  !$acc end parallel
end subroutine kernel
```

```
!--- Loki-SCC driver format  --
!$acc update(...)
!$acc parallel loop gang
do ibl=1, nblocks
  call kernel(point(:,ibl), column(:,:,ibl))
end do
!$acc end parallel
!$acc end update


!--- Loki-SCC kernel format  --
subroutine kernel(nproma,nlev,point,column)
  real :: column_tmp(nproma,nlev)
  !$acc routine vector

  !$acc loop vector
  do jl = 1, nproma
    do lev = 2, nlev
      column_tmp(jl,lev) = <f(point(jl),
                            column(jl,lev-1))>
    end do

    do lev = 1, NLEVELS
      column(jl,lev) = <g(column_tmp(lev), ...)>
    end do
  end do

end subroutine kernel
```

Another approach: Single-column coalesced (SCC)

Direct CPU-to-GPU transformation in Loki

```
!--- Loki-SCC driver format  --


!$acc update(...)
!$acc parallel loop gang
do ibl=1, nblocks
  call kernel(point(:,ibl), column(:,:,ibl))
end do
!$acc end parallel
!$acc end update



!--- Loki-SCC kernel format  --
subroutine kernel(nproma,nlev,point,column)
real :: column_tmp(nproma,nlev)
  !$acc routine vector

  !$acc loop vector
  do jl = 1, nproma
    do lev = 2, nlev
      column_tmp(jl,lev) = <f(point(jl),
                            column(jl,lev-1))>
    end do

    do lev = 1, NLEVELS
      column(jl,lev) = <g(column_tmp(lev), ...)>
    end do
  end do
end subroutine kernel
```

```
!--- Loki-SCC-hoisted driver format  --
real, intent(inout) :: column_tmp(nproma,nlev,nblocks)

!$acc update(...,create(column_tmp))
!$acc parallel loop gang
do ibl=1, nblocks
  !$acc loop vector
  do jl = 1, nproma
    call kernel(point(:,ibl), column(:,:,ibl),
                column_tmp(:,:,ibl), jl=jl)
  end do
end do
!$acc end parallel
!$acc end update


!--- Loki-SCC-hoisted kernel format  --
subroutine kernel(jl,nproma,nlev,point,column)
  real, intent(inout) :: column_tmp(nlev)
  !$acc routine seq

  do lev = 2, nlev
    column_tmp(jl,lev) = <f(point(jl),
                          column(jl,lev-1))>
  end do

  do lev = 1, NLEVELS
    column(jl,lev) = <g(column_tmp(lev), ...)>
  end do
end subroutine kernel
```