YAO
toward an efficient data assimilation environment

Luigi NARDI Fouad BADRAN Sylvie THIRIA

LOCEAN : Laboratoire d'Océanographie et du Climat Expérimentation et Approches Numériques.
CNRS / IRD / UPMC / MNHN / IPSL

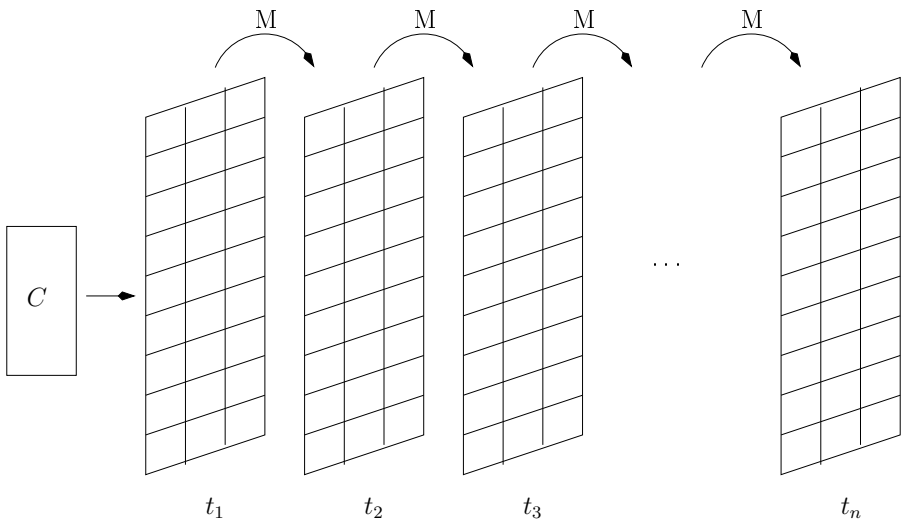
CEDRIC : Centre d'Etude et De Recherche en Informatique du Cnam

Paris France

Outline

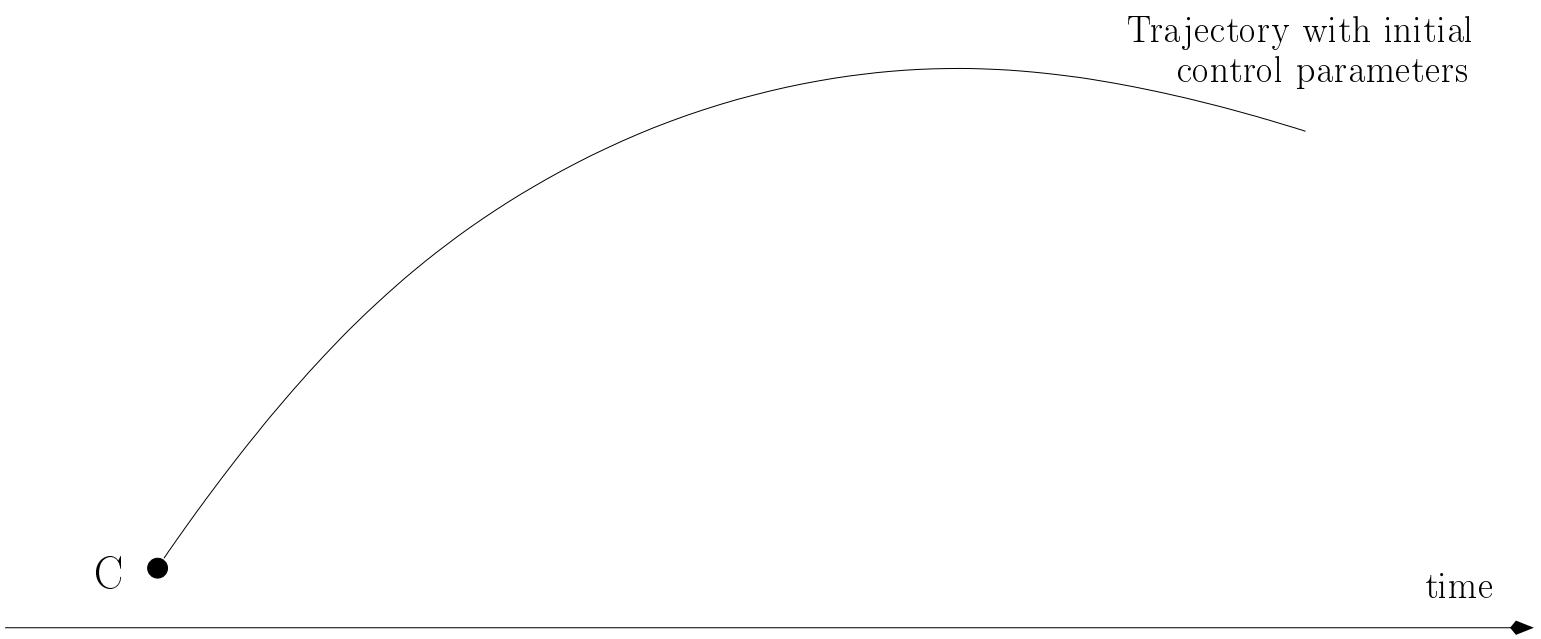
Numerical direct model

- M : direct model between two time steps t_i, t_{i+1}
- C : control parameters



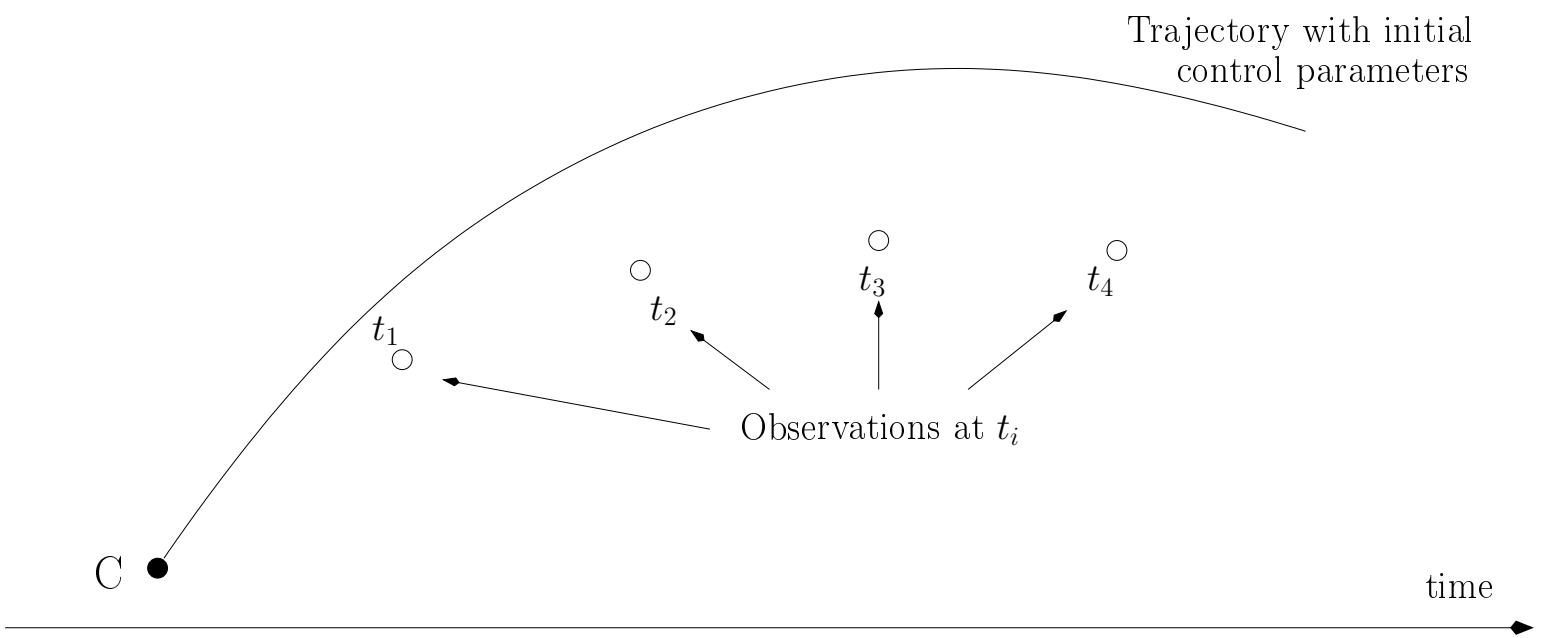
Models are used to forecast or analyse the evolution of phenomena but. . . numerical models are not perfects !

Variational data assimilation idea



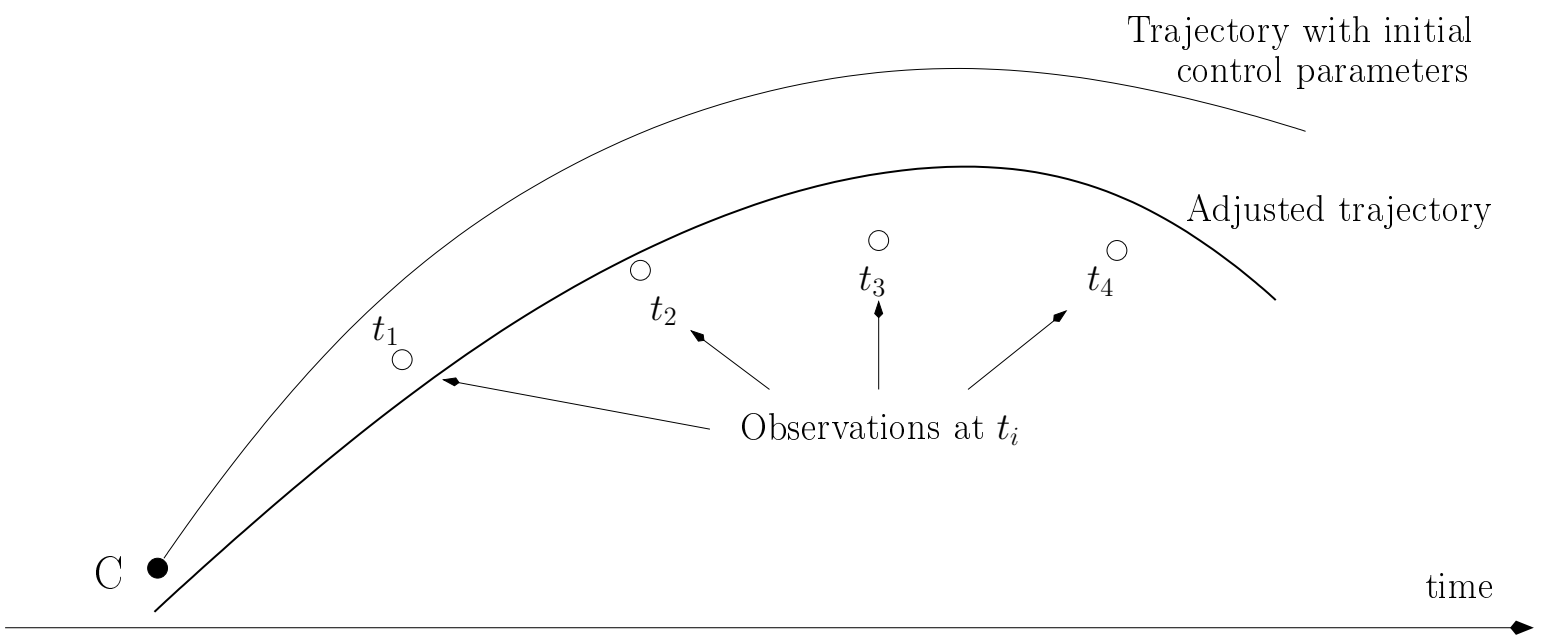
Cost function : $J(C)$ Gradient : $\nabla_C J$ Adjoint model : $\nabla_C M^T$

Variational data assimilation idea



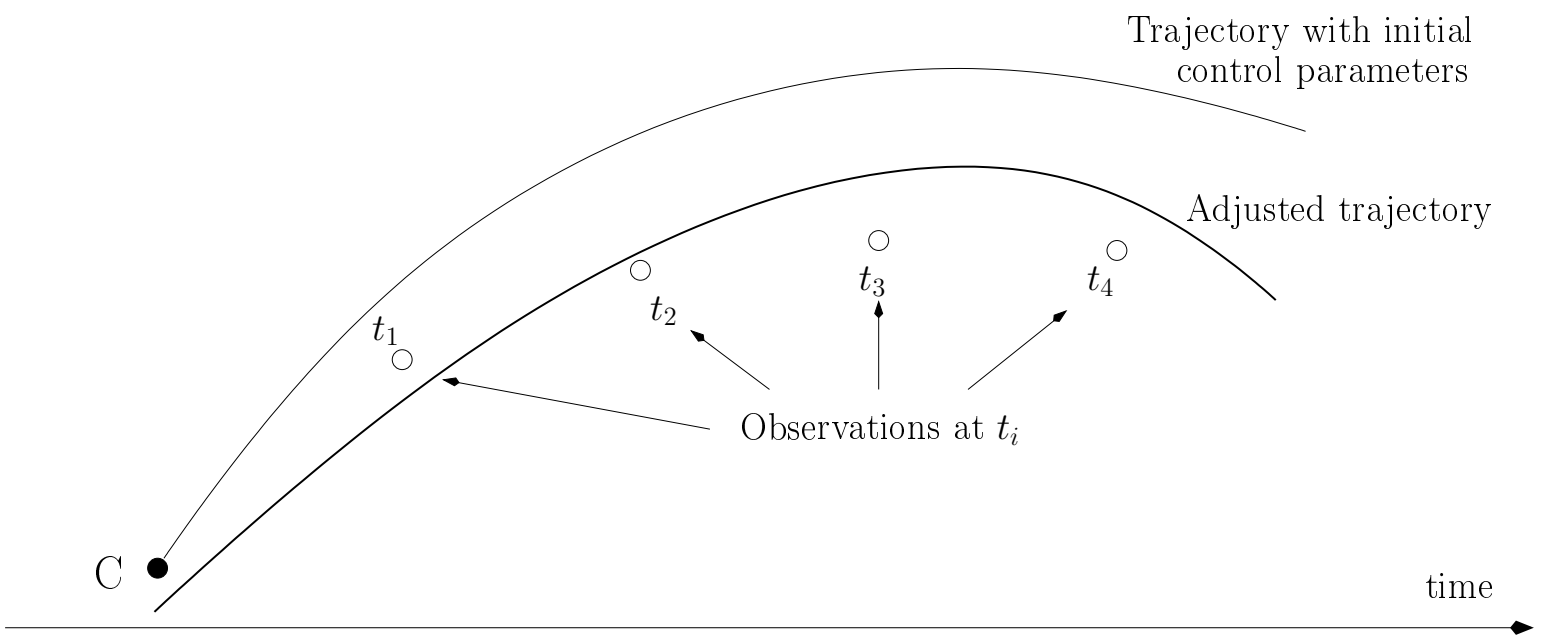
Cost function : $J(C)$ Gradient : $\nabla_C J$ Adjoint model : $\nabla_C M^T$

Variational data assimilation idea



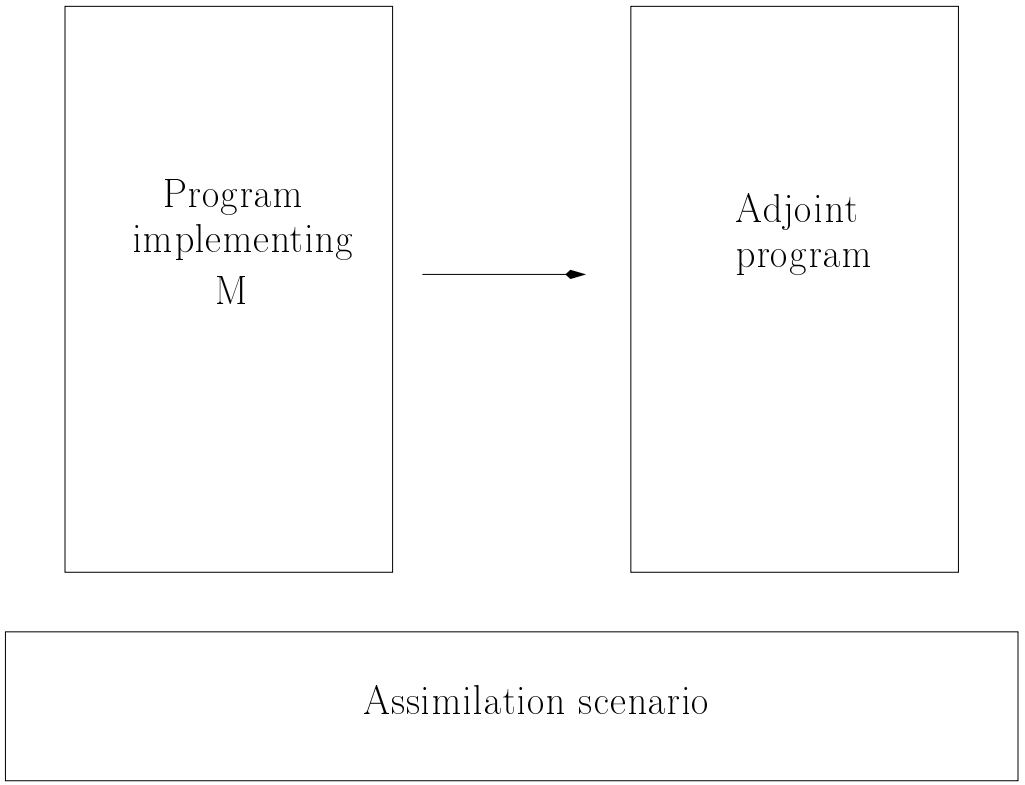
Cost function : $J(C)$ Gradient : $\nabla_C J$ Adjoint model : $\nabla_C M^T$

Variational data assimilation idea



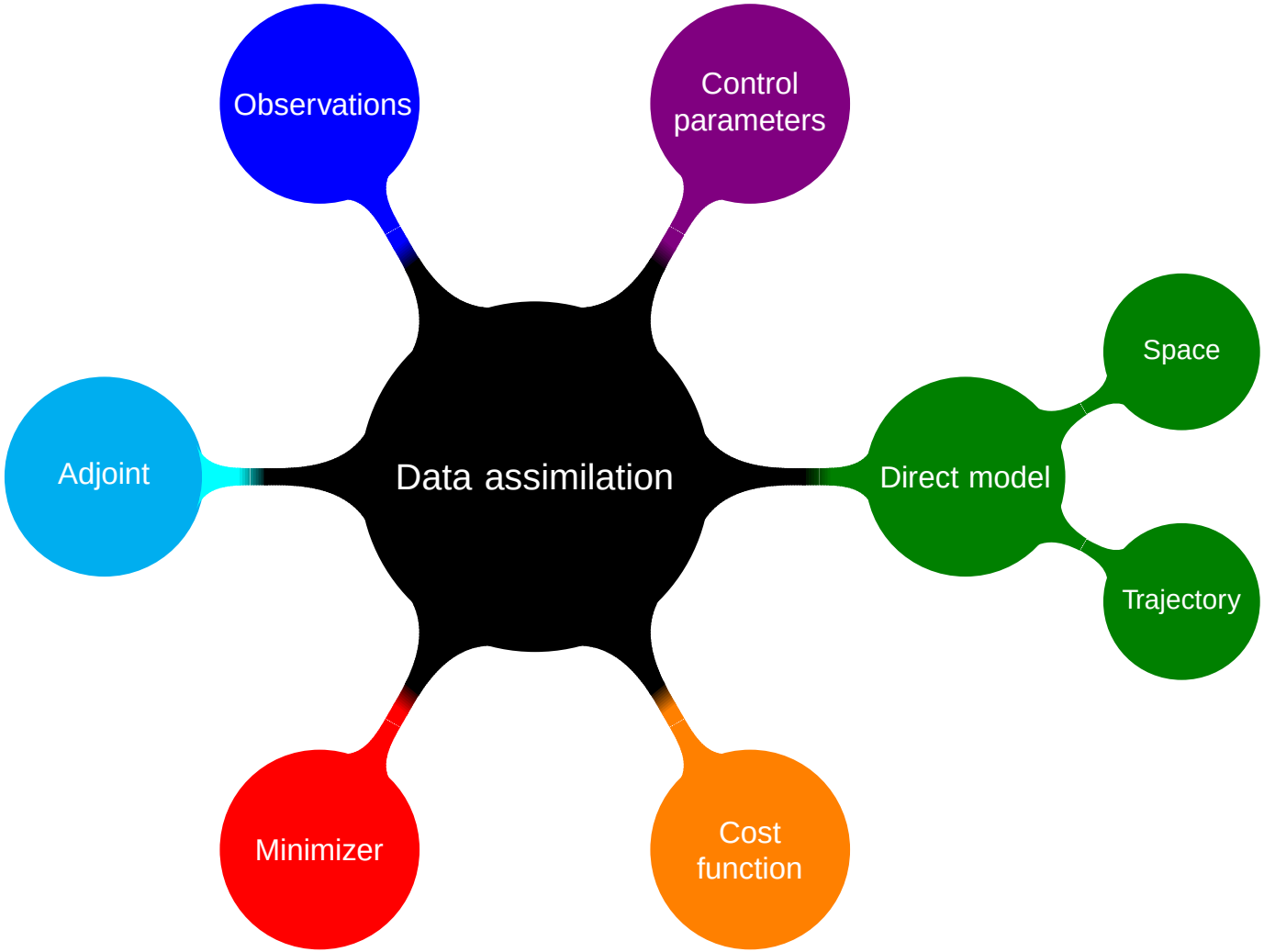
Cost function : $J(C)$ Gradient : $\nabla_C J$ Adjoint model : $\nabla_C M^T$

Programming codes



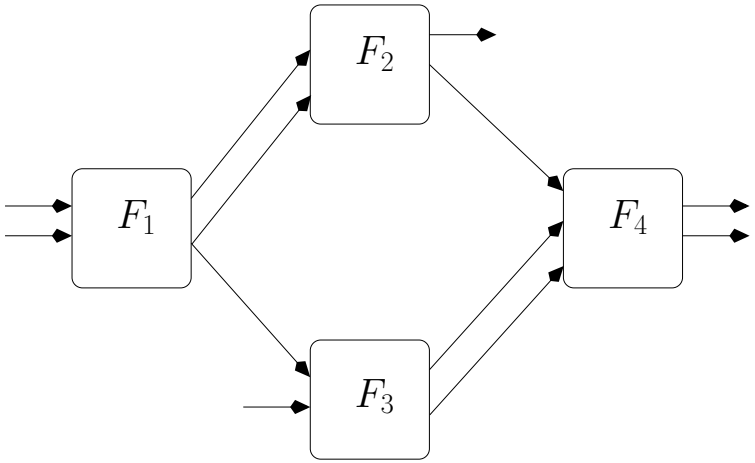
Write the adjoint program is very time expensive

Data assimilation components



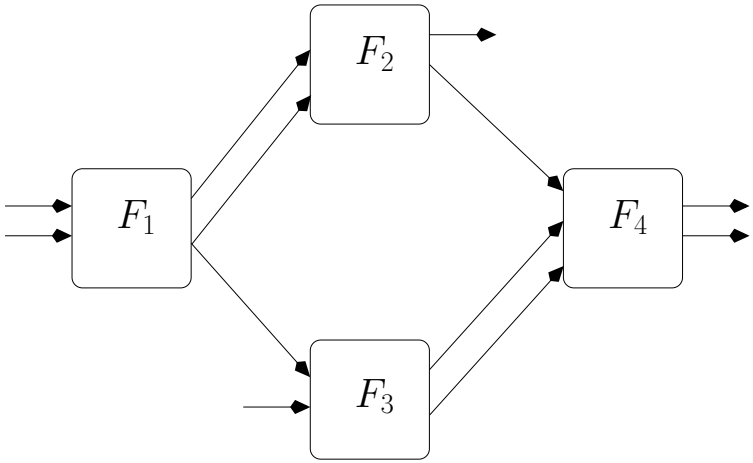
Modular graph

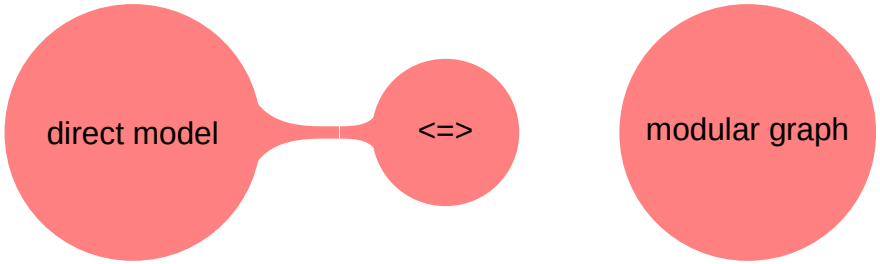
- *module* : function representing a computation entity
- *basic connection* : data transmission between modules



Modular graph

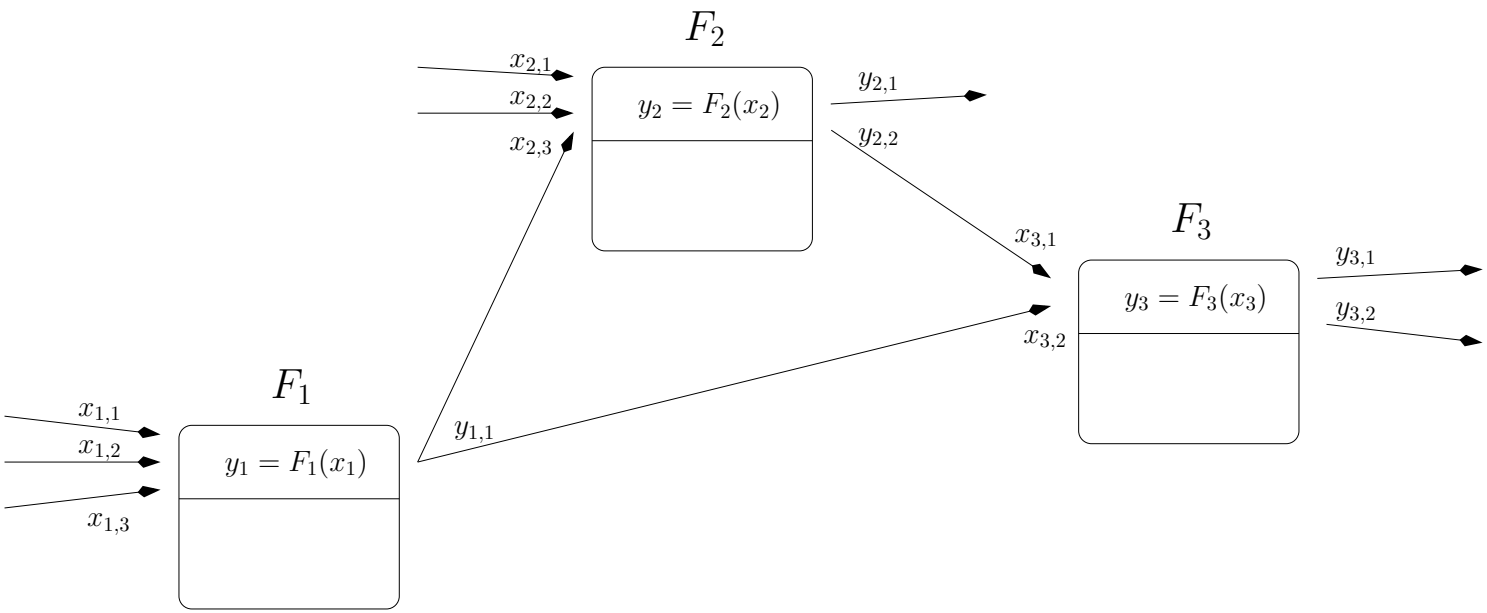
- *module* : function representing a computation entity
- *basic connection* : data transmission between modules





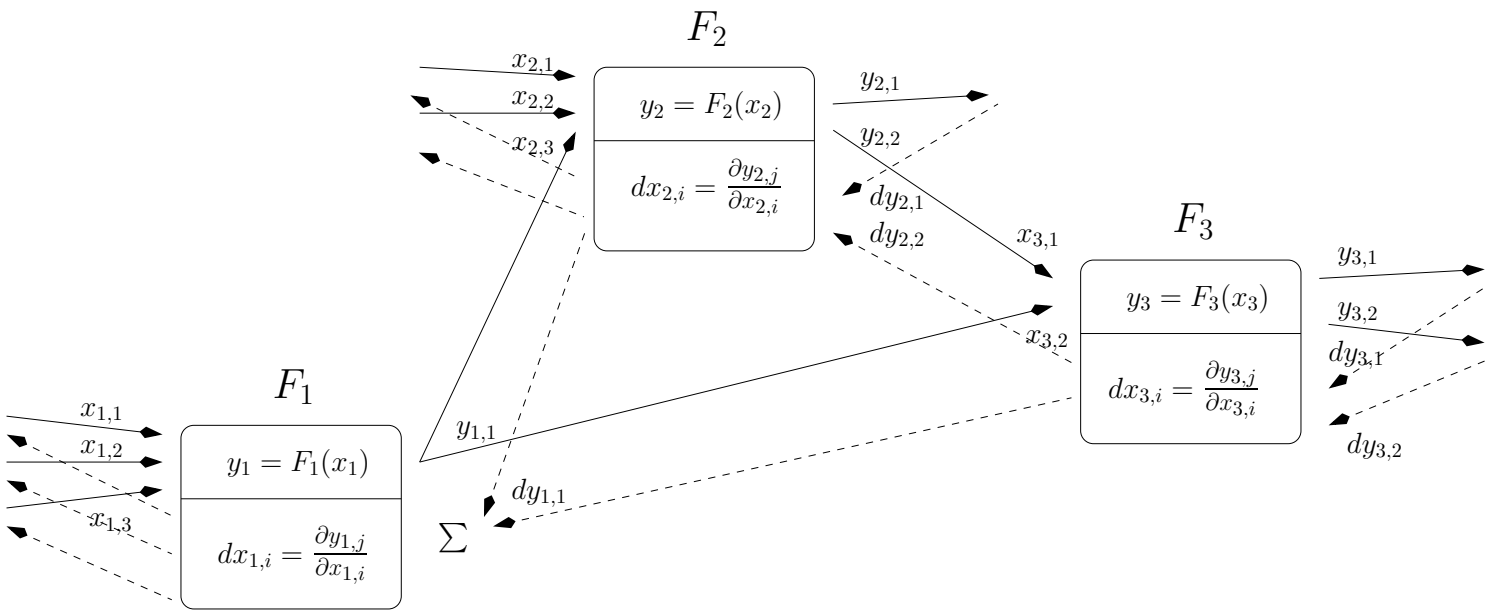
Direct model

Passing through the graph in a topological order we calculate the direct model : forward algorithm



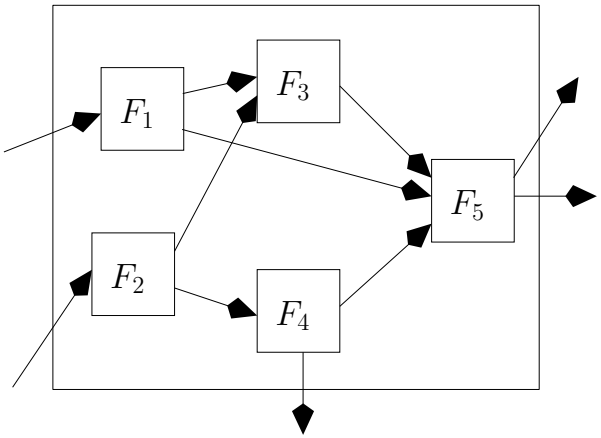
Adjoint model

Passing through the graph in a reverse topological order we calculate the adjoint model : backward algorithm



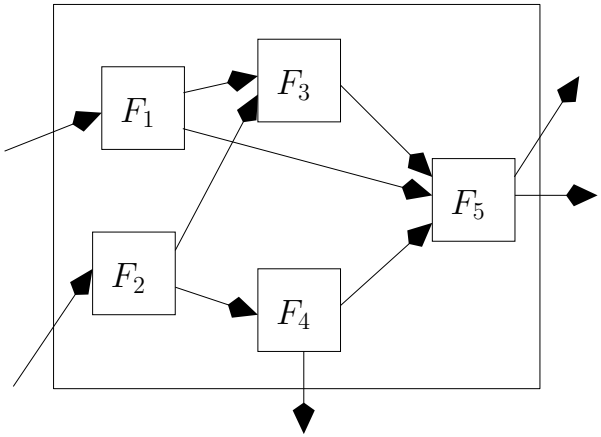
Graph replication

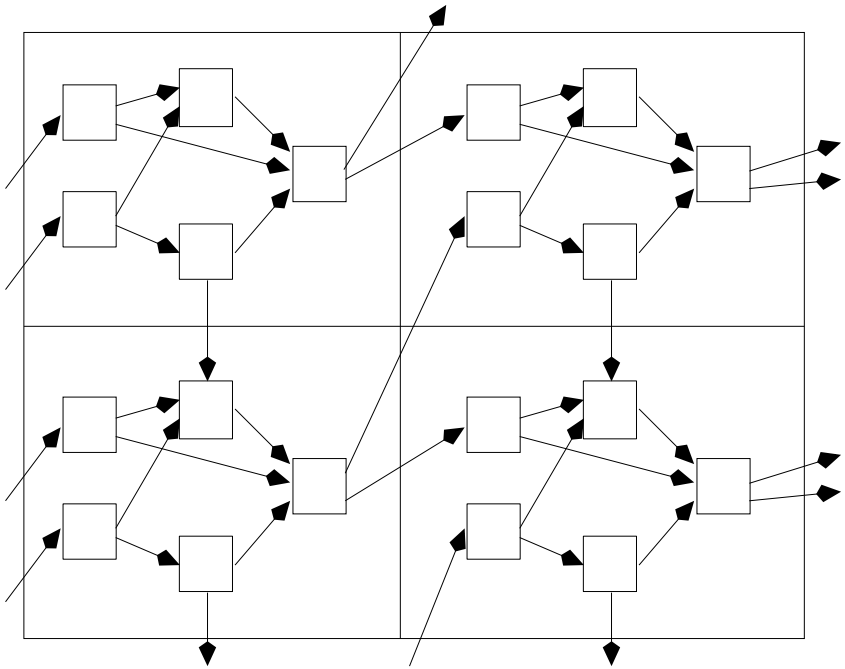
The same graph is repeated for each grid point and time step



Graph replication

The same graph is repeated for each grid point and time step





The shallow water numerical model

Dynamic variables :

$$u_{ijt} = \hat{u}_{ijt-2} + 2\Delta t \left(\frac{-g^*}{\Delta x} \left[h_{i+1jt-1} - h_{ijt-1} \right] + \frac{f}{4} \left[v_{ijt-1} + v_{ij+1t-1} + v_{i+1jt-1} + v_{i+1j+1t-1} \right] - \gamma \cdot \hat{u}_{ijt-2} \right)$$

$$v_{ijt} = \hat{v}_{ijt-2} + 2\Delta t \left(\frac{-g^*}{\Delta y} \left[h_{ijt-1} - h_{ij-1t-1} \right] - \frac{f}{4} \left[u_{i-1j-1t-1} + u_{i-1jt-1} + u_{ij-1t-1} + u_{ijt-1} \right] - \gamma \cdot \hat{v}_{ijt-2} \right)$$

$$h_{ijt} = \hat{h}_{ijt-2} - 2\Delta t \cdot H \left(\frac{u_{ijt-1} - u_{i-1jt-1}}{\Delta x} + \frac{v_{ij+1t-1} - v_{ijt-1}}{\Delta y} \right)$$

Asselin filter :

$$\begin{aligned} \hat{u}_{ijt} &= u_{ijt-1} + \alpha(\hat{u}_{ijt-1} - 2u_{ijt-1} + u_{ijt}) \\ \hat{v}_{ijt} &= v_{ijt-1} + \alpha(\hat{v}_{ijt-1} - 2v_{ijt-1} + v_{ijt}) \\ \hat{h}_{ijt} &= h_{ijt-1} + \alpha(\hat{h}_{ijt-1} - 2h_{ijt-1} + h_{ijt}) \end{aligned}$$

u,v : horizontal velocities

h : height of the water

YAO description language

```
trajectory 100
space 50 50

module Ĥ input 3 output 1
module Û input 3 output 1
module V̂ input 3 output 1
module H input 5 output 1
module U input 7 output 1
module V input 7 output 1

connection Ĥ from Ĥ i j t-1
connection Ĥ from H i j t-1
connection Ĥ from H i j t
connection Û from Û i j t-1
connection Û from U i j t-1
connection Û from U i j t
connection V̂ from V̂ i j t-1
connection V̂ from V i j t-1
connection V̂ from V i j t
connection H from Ĥ i j t-1
connection H from U i j t-1
...
...

order i j
    H U V Ĥ Û V̂
endorder
```

YAO description language

```
trajectory 100
space 50 50

module Ĥ input 3 output 1
module Û input 3 output 1
module V̂ input 3 output 1
module H input 5 output 1
module U input 7 output 1
module V input 7 output 1

connection Ĥ from Ĥ i j t-1
connection Ĥ from H i j t-1
connection Ĥ from H i j t
connection Û from Û i j t-1
connection Û from U i j t-1
connection Û from U i j t
connection V̂ from V̂ i j t-1
connection V̂ from V i j t-1
connection V̂ from V i j t
connection H from Ĥ i j t-1
connection H from U i j t-1
...
...

order i j
    H U V Ĥ Û V̂
endorder
```

YAO description language

```
trajectory 100
space 50 50

module Ĥ input 3 output 1
module Û input 3 output 1
module V̂ input 3 output 1
module H input 5 output 1
module U input 7 output 1
module V input 7 output 1

connection Ĥ from Ĥ i j t-1
connection Ĥ from H i j t-1
connection Ĥ from H i j t
connection Û from Û i j t-1
connection Û from U i j t-1
connection Û from U i j t
connection V̂ from V̂ i j t-1
connection V̂ from V i j t-1
connection V̂ from V i j t
connection H from Ĥ i j t-1
connection H from U i j t-1
...
...

order i j
   H U V Ĥ Û V̂
endorder
```

YAO description language

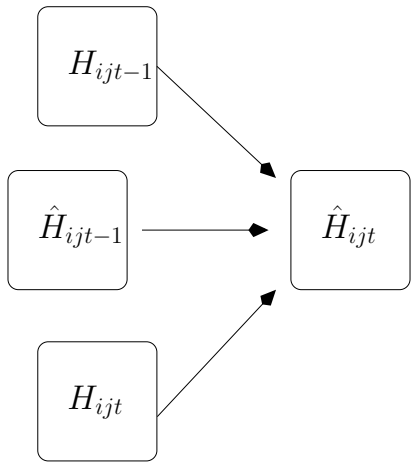
```
trajectory 100
space 50 50

module Ĥ input 3 output 1
module Û input 3 output 1
module V̂ input 3 output 1
module H input 5 output 1
module U input 7 output 1
module V input 7 output 1

connection Ĥ from Ĥ i j t-1
connection Ĥ from H i j t-1
connection Ĥ from H i j t
connection Û from Û i j t-1
connection Û from U i j t-1
connection Û from U i j t
connection V̂ from V̂ i j t-1
connection V̂ from V i j t-1
connection V̂ from V i j t
connection H from Ĥ i j t-1
connection H from U i j t-1
...
...

order i j
    H U V Ĥ Û V̂
endorder
```

$$\hat{h}_{ijt} = h_{ijt-1} + \alpha(\hat{h}_{ijt-1} - 2h_{ijt-1} + h_{ijt})$$



YAO description language

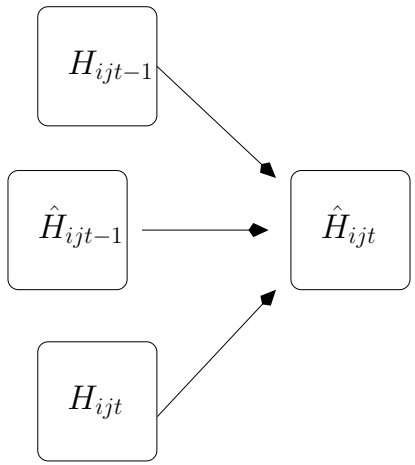
```
trajectory 100
space 50 50

module Ĥ input 3 output 1
module Û input 3 output 1
module V̂ input 3 output 1
module H input 5 output 1
module U input 7 output 1
module V input 7 output 1

connection Ĥ from Ĥ i j t-1
connection Ĥ from H i j t-1
connection Ĥ from H i j t
connection Û from Û i j t-1
connection Û from U i j t-1
connection Û from U i j t
connection V̂ from V̂ i j t-1
connection V̂ from V i j t-1
connection V̂ from V i j t
connection H from Ĥ i j t-1
connection H from U i j t-1
...
...

order i j
    H U V Ĥ Û V̂
endorder
```

$$\hat{h}_{ijt} = h_{ijt-1} + \alpha(\hat{h}_{ijt-1} - 2h_{ijt-1} + h_{ijt})$$



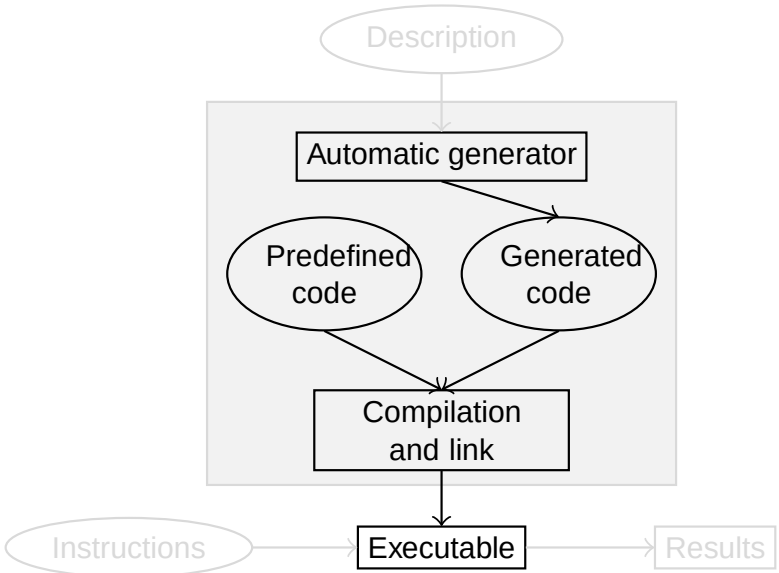
Real applications

Several variational data assimilation applications have been developed with YAO :

- **NEMO** : adjoint model of the GYRE configuration of NEMO (Nucleus for European Modelling of the Ocean)
- **NEUROVARIA** : variational inversion of multi-spectral satellite ocean color measurements for the restitution of the chlorophyll-a
- **Marine acoustics** : variational inversion of sound speed profile and retrieval of geoacoustic parameters (celerity, density, attenuation, . . .)
- **PISCES** : ocean color variational data assimilation in a biogeochemical model
- **ISBA** : variational data assimilation in the hydrology model of interface ground - vegetation - atmosphere

YAO architecture

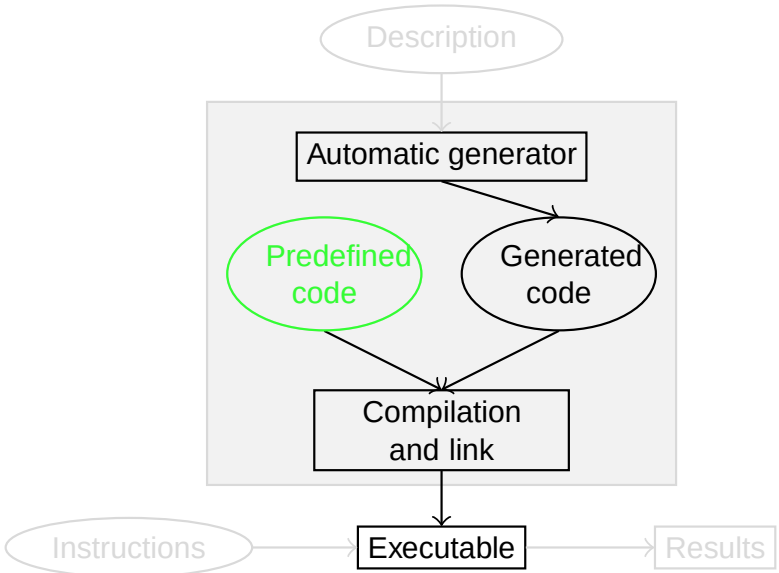
YAO is composed by an assimilation predefined code and a model dependent code



YAO architecture

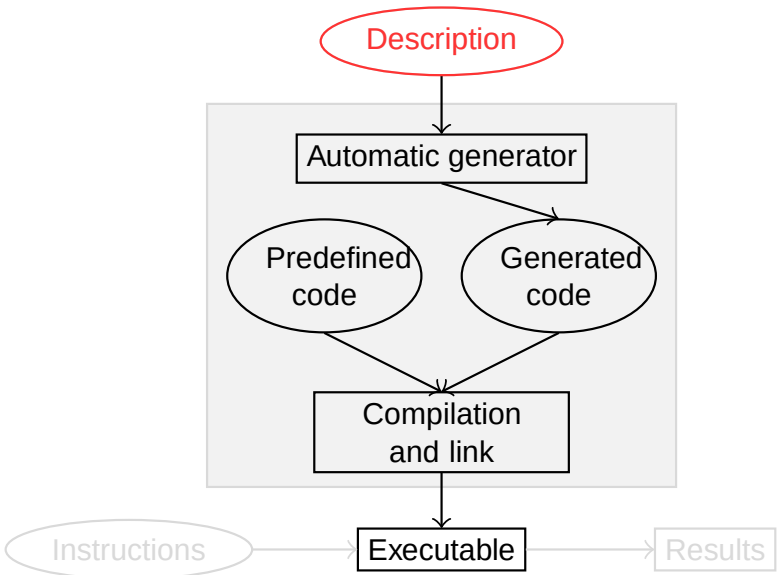
YAO is composed by an assimilation predefined code and a model dependent code

Common data assimilation code provided by YAO



YAO architecture

YAO is composed by an assimilation predefined code and a model dependent code

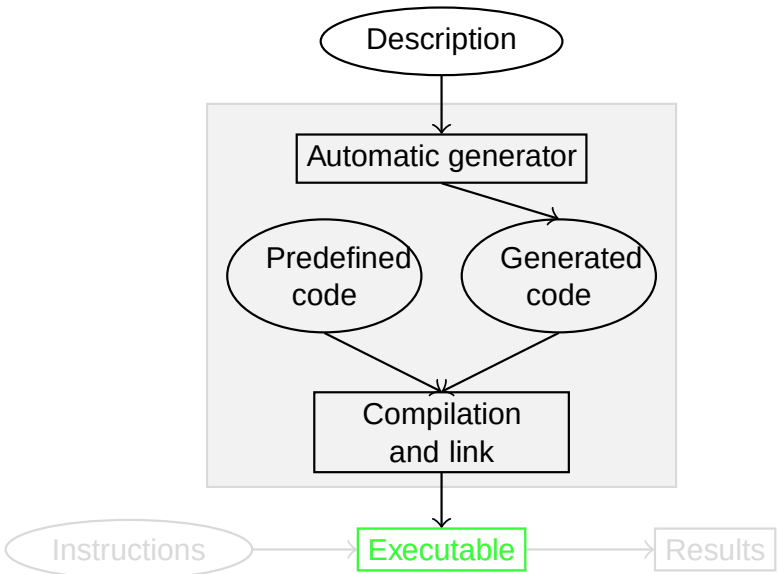


Specific application code provided by the user

YAO architecture

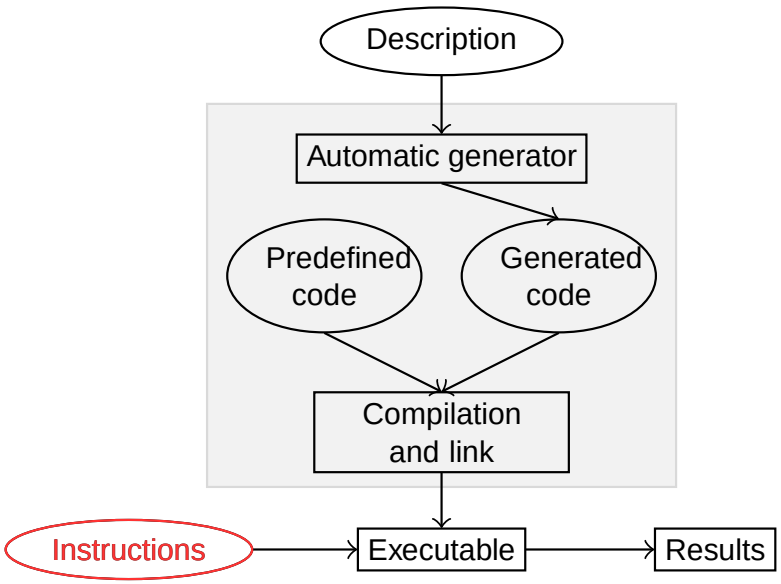
YAO is composed by an assimilation predefined code and a model dependent code

Generation and compilation made by YAO



YAO architecture

YAO is composed by an assimilation predefined code and a model dependent code



Assimilation scenario defined by the user

Target :
generate a parallel code automatically

Problem :
the code generated by YAO depends on the numerical model

- all the generated codes have mainly the same structure
- the modular graph gives the dependencies between the modules

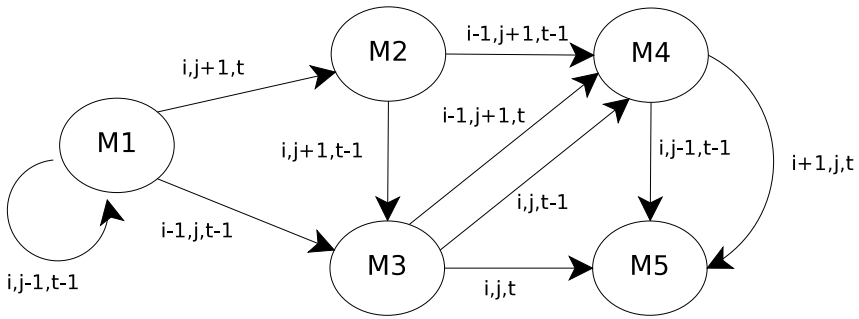
Example

```
connection M1 from M1 i   j-1 t-1
connection M2 from M1 i   j+1 t
connection M3 from M1 i+1 j   t-1
connection M3 from M2 i   j+1 t-1
connection M4 from M2 i-1 j+1 t-1
connection M4 from M3 i-1 j+1 t
connection M4 from M3 i   j   t-1
connection M5 from M3 i   j   t
connection M5 from M4 i   j-1 t-1
connection M5 from M4 i+1 j   t
```

```
order i j
    M1 M2 M3
endorder

order i j
    M4 M5
endorder
```

Representation of the modular graph dependencies in a point of the space :



Forward translation for a time step t

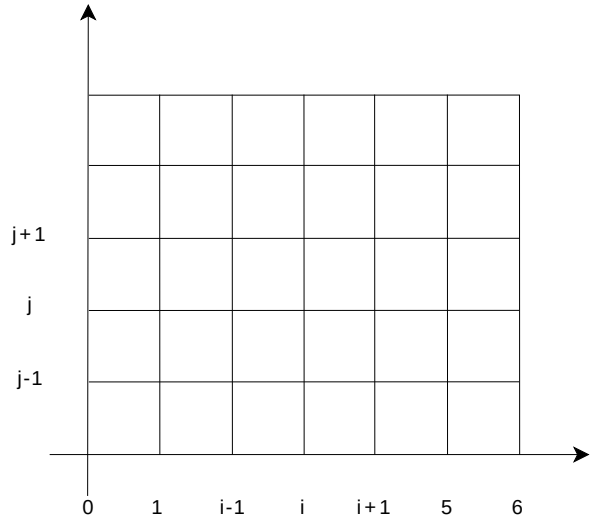
```
loop i ascendant
   loop j ascendant
      input[0] = M1(i,j-1,t-1)
      M1->forward(input)

      input[0] = M1(i,j+1,t)
      M2->forward(input)

      input[0] = M1(i+1,j,t-1)
      input[1] = M2(i,j+1,t-1)
      M3->forward(input)
```

```
loop i ascendant
   loop j ascendant
      input[0] = M2(i-1,j+1,t-1)
      input[1] = M3(i-1,j+1,t)
      input[2] = M3(i,j,t-1)
      M4->forward(input)

      input[0] = M3(i,j,t)
      input[1] = M4(i,j-1,t-1)
      input[2] = M4(i+1,j,t)
      M5->forward(input)
```



The data set can be logically partitioned : domain decomposition.

Each thread Ti carries out the same work, but on its own portion of data set.

Problem : synchronization and race conditions.

Forward translation for a time step t

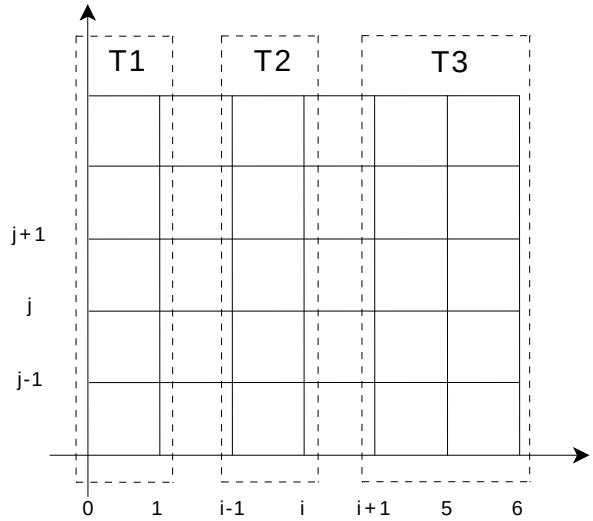
```
loop i ascendant
   loop j ascendant
      input[0] = M1(i,j-1,t-1)
      M1->forward(input)

      input[0] = M1(i,j+1,t)
      M2->forward(input)

      input[0] = M1(i+1,j,t-1)
      input[1] = M2(i,j+1,t-1)
      M3->forward(input)
```

```
loop i ascendant
   loop j ascendant
      input[0] = M2(i-1,j+1,t-1)
      input[1] = M3(i-1,j+1,t)
      input[2] = M3(i,j,t-1)
      M4->forward(input)

      input[0] = M3(i,j,t)
      input[1] = M4(i,j-1,t-1)
      input[2] = M4(i+1,j,t)
      M5->forward(input)
```



The data set can be logically partitioned :

domain decomposition.

Each thread Ti carries out the same work, but on its own portion of data set.

Problem : synchronization and race conditions.

Forward translation for a time step t

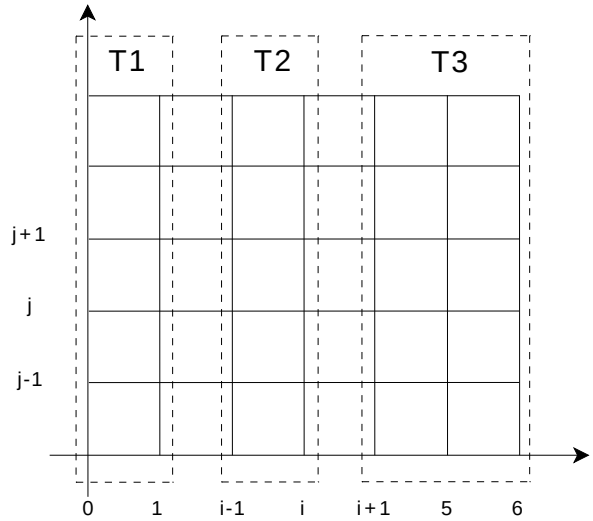
```
loop i ascendant
   loop j ascendant
      input[0] = M1(i,j-1,t-1)
      M1->forward(input)

      input[0] = M1(i,j+1,t)
      M2->forward(input)

      input[0] = M1(i+1,j,t-1)
      input[1] = M2(i,j+1,t-1)
      M3->forward(input)
```

```
loop i ascendant
   loop j ascendant
      input[0] = M2(i-1,j+1,t-1)
      input[1] = M3(i-1,j+1,t)
      input[2] = M3(i,j,t-1)
      M4->forward(input)

      input[0] = M3(i,j,t)
      input[1] = M4(i,j-1,t-1)
      input[2] = M4(i+1,j,t)
      M5->forward(input)
```

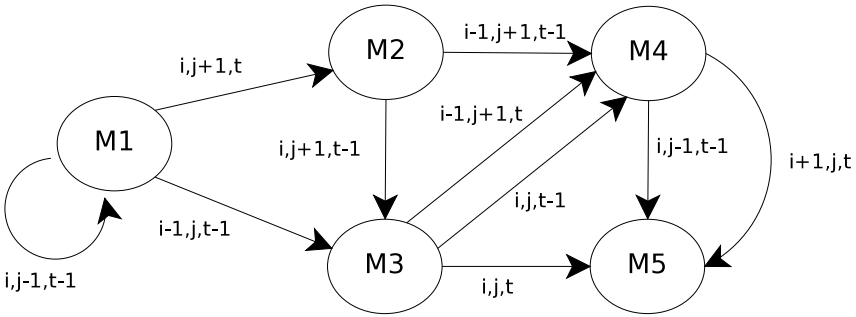


The data set can be logically partitioned :
domain decomposition.

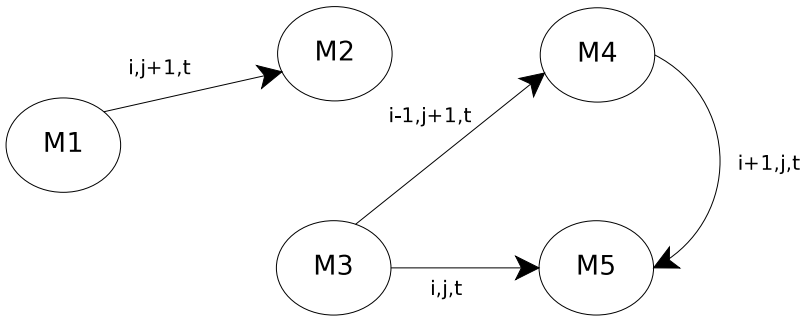
Each thread Ti carries out the same work, but on its own portion of data set.

Problem : synchronization and race conditions.

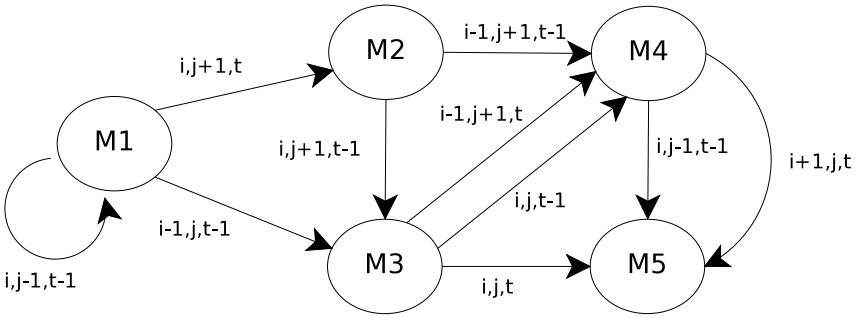
Synchronization problem



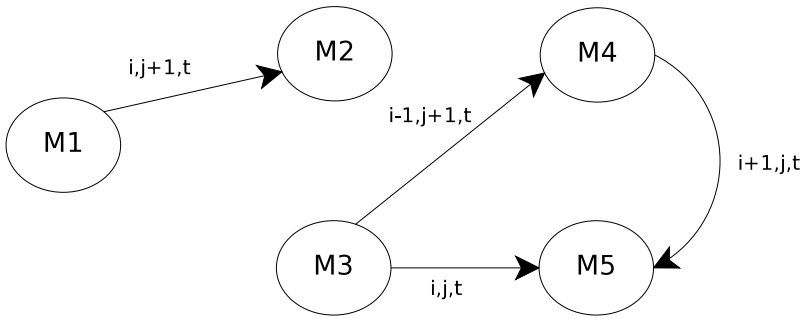
at time t-1 modules
are already computed

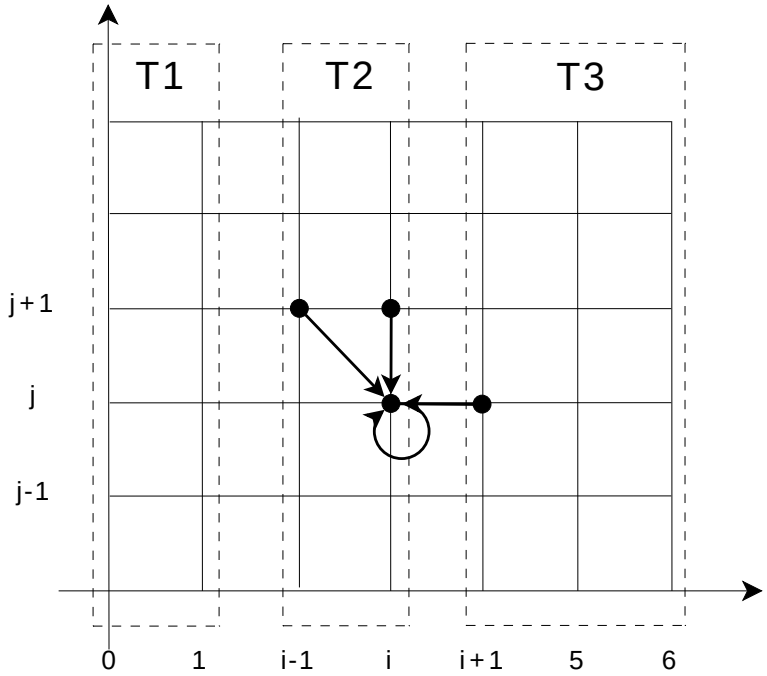


Synchronization problem



at time t-1 modules
are already computed





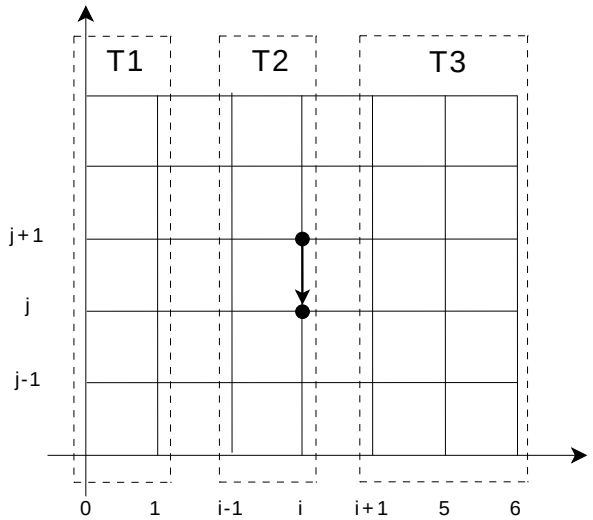
First nested loops

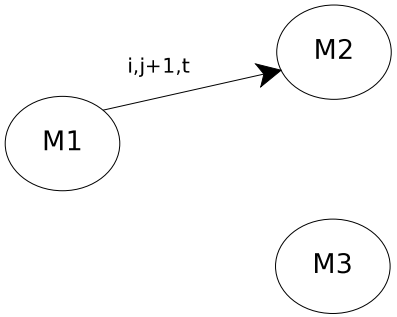
```
loop i ascendant
   loop j ascendant
      input[0] = M1(i,j-1,t-1)
      M1->forward(input)

      input[0] = M1(i,j+1,t)
      M2->forward(input)

      input[0] = M1(i+1,j,t-1)
      input[1] = M2(i,j+1,t-1)
      M3->forward(input)
```

```
order i j
    M1 M2 M3
endorder
```



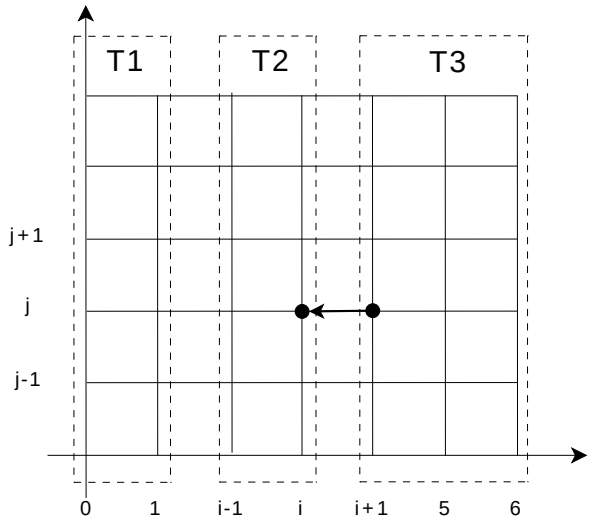


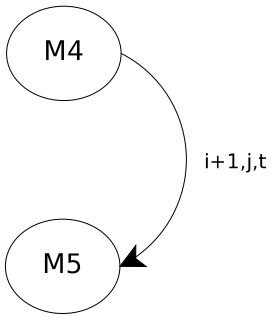
Second nested loops

```
loop i ascendant
   loop j ascendant
      input[0] = M2(i-1,j+1,t-1)
      input[1] = M3(i-1,j+1,t)
      input[2] = M3(i,j,t-1)
      M4->forward(input)

      input[0] = M3(i,j,t)
      input[1] = M4(i,j-1,t-1)
      input[2] = M4(i+1,j,t)
      M5->forward(input)
```

```
order i j
    M4 M5
endorder
```

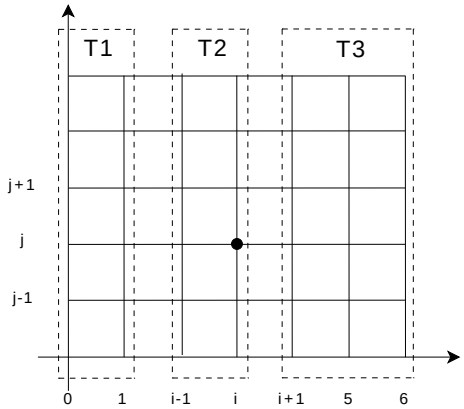




Second nested loops decomposition

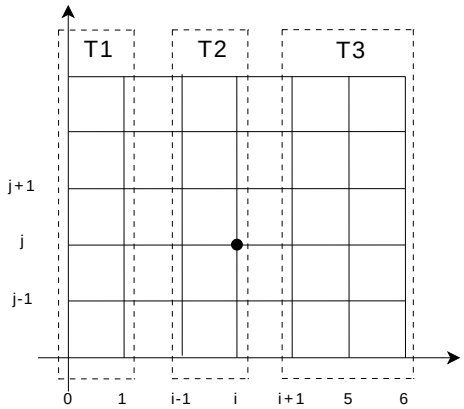
```
loop i ascendant
   loop j ascendant
      input[0] = M2(i-1,j+1,t-1)
      input[1] = M3(i-1,j+1,t)
      input[2] = M3(i,j,t-1)
      M4->forward(input)
```

```
order i j
    M4
endorder
```




```
loop i ascendant
   loop j ascendant
      input[0] = M3(i,j,t)
      input[1] = M4(i,j-1,t-1)
      input[2] = M4(i+1,j,t)
      M5->forward(input)
```

```
order i j
    M5
endorder
```



Performance considerations

- is very often possible to split an order directive to obtain a major percentage of parallel code
- the scalability is limited by the external loop magnitude but a double dimension domain decomposition is possible
- OpenMP vs MPI :
  - the computation in an order directive is not huge. The danger in using MPI for each nested loop is that the time spent in passing messages may be more than the time spent for computation
  - the downside of OpenMP is that in shared memory architectures the quantity of CPUs is often limited (but evolutions . . .)

Future works

- generate the order directive automatically analyzing the modular graph (in progress)
- the adjoint computation needs a huge memory allocation. Checkpointing : the idea is to store less data and to recompute from a snapshot
- graphic interface to help the user to conceive the modular graph (in progress)

Conclusions

YAO is a software for variational data assimilation

Minimal programming effort required

The numerical direct model and its adjoint model are provided

Flexible : if a model is upgraded little modifications are required

Conclusions

YAO is a software for variational data assimilation

Minimal programming effort required

The numerical direct model and its adjoint model are provided

Flexible : if a model is upgraded little modifications are required

Conclusions

YAO is a software for variational data assimilation

Minimal programming effort required

The numerical direct model and its adjoint model are provided

Flexible : if a model is upgraded little modifications are required

Conclusions

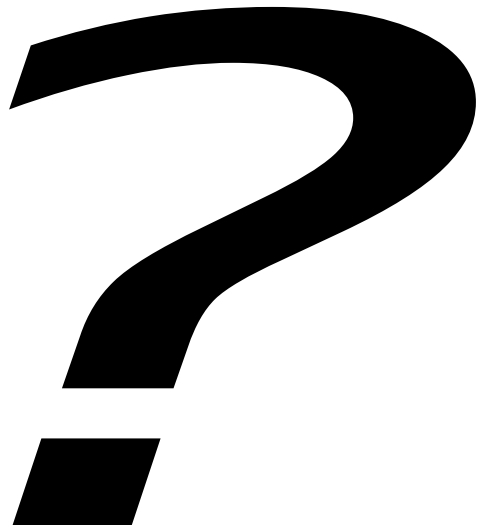
YAO is a software for variational data assimilation

Minimal programming effort required

The numerical direct model and its adjoint model are provided

Flexible : if a model is upgraded little modifications are required

Questions



Luigi Nardi, Charles Sorror, Fouad Badran and Sylvie Thiria,
YAO : A Software for Variational Data Assimilation Using Numerical Models.
LNCS 5593, Computational Science and Its Applications - ICCSA 2009, 621-636, Springer-Verlag.

YAO home page : *http://www.locean-ipsl.upmc.fr/∼yao*
YAO group email : *lnalod@locean-ipsl.upmc.fr*