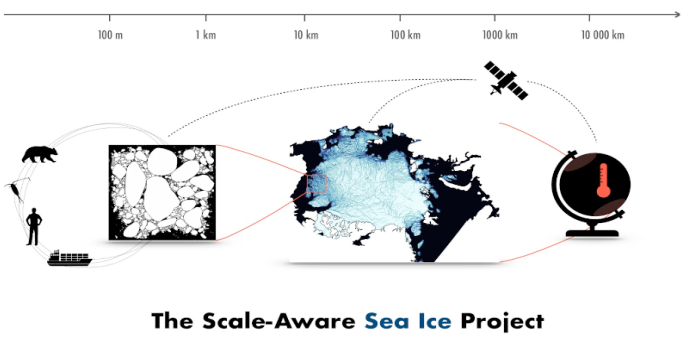
Introducing NEDAS:
A Light-weight and Scalable Python Solution for Ensemble Data Assimilation

Yue (Michael) Ying

NERSC

Supported by: *SASIP, ACCIBERG*



Ensemble Data Assimilation

Model state $\boldsymbol{\psi}$ updated by observation $\boldsymbol{\varphi}$ to find best estimate

$$p(\boldsymbol{\psi}|\boldsymbol{\varphi}) = \frac{p(\boldsymbol{\varphi}|\boldsymbol{\psi})p(\boldsymbol{\psi})}{p(\boldsymbol{\varphi})}$$

Use an ensemble of state $\boldsymbol{\Psi} = \left(\boldsymbol{\psi}_1, \ldots, \boldsymbol{\psi}_{N_e}\right) \in \mathbb{R}^{N_{state} \times N_e}$
as samples of $p(\boldsymbol{\psi})$

and "observation priors" $\boldsymbol{\Phi} = \left(\boldsymbol{\varphi}_1, \ldots, \boldsymbol{\varphi}_{N_e}\right) \in \mathbb{R}^{N_{obs} \times N_e}$
comparing with actual observation $\boldsymbol{\varphi}^o$ to give the likelihood $p(\boldsymbol{\varphi}|\boldsymbol{\psi})$

How to update $\boldsymbol{\Psi}$ so that it characterizes $p(\boldsymbol{\psi}|\boldsymbol{\varphi})$?

Algorithm: $\boldsymbol{\Psi} \leftarrow \mathcal{A}(\boldsymbol{\Psi}, \boldsymbol{\Phi}, \boldsymbol{\varphi}^o)$

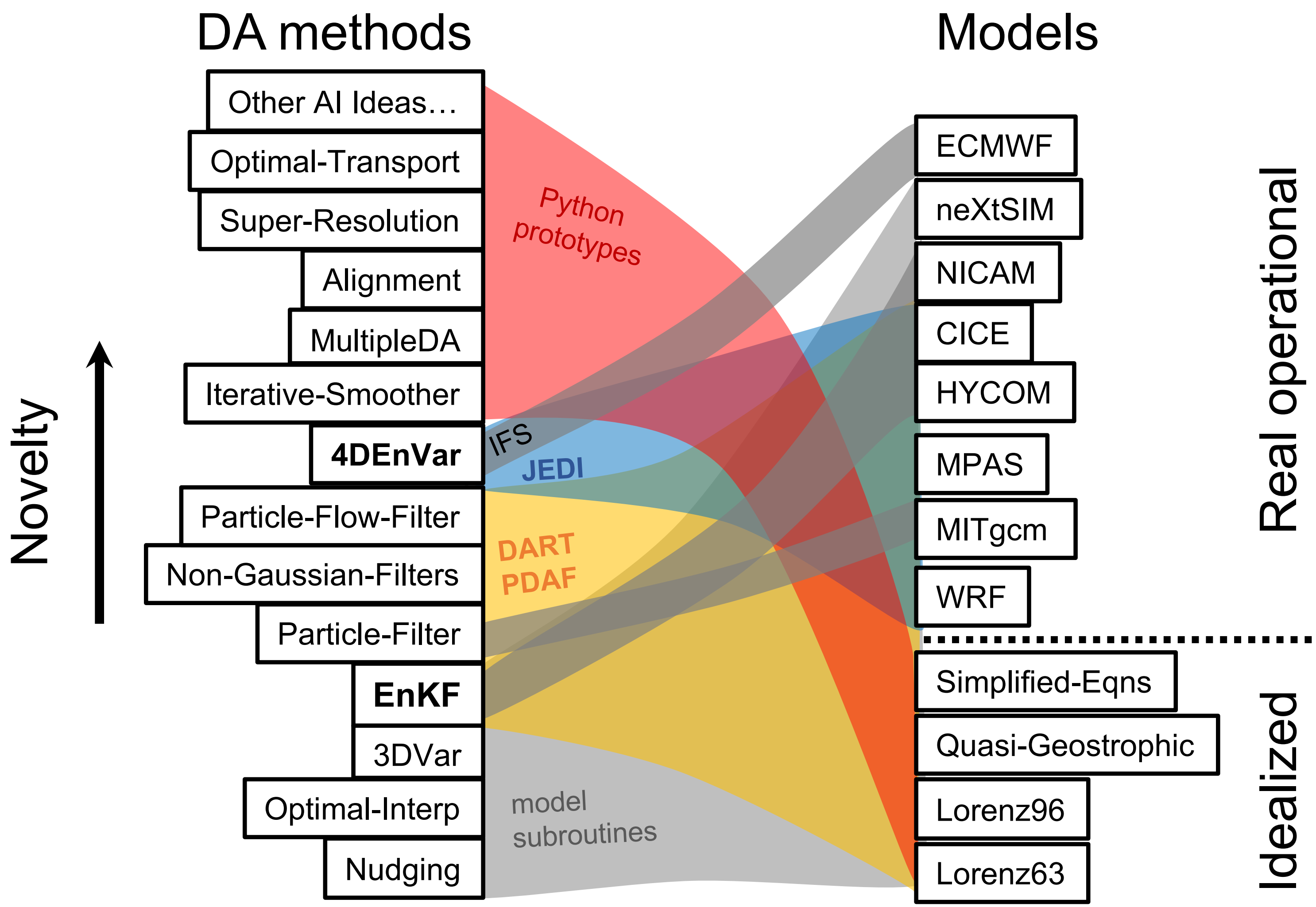
How much effort is needed for testing novel algorithms in real models?

Simple method: just implement in the model code (WRF nudging/fdda)

Complex method: dedicated DA software:

Data Assimilation Research Testbed (DART; Anderson et al. 2009)
Parallel Data Assimilation Framework (PDAF; Nerger & Hiller 2013)
Joint Effort in DA Integration (JEDI; JCSDA)

Conception → Python prototype → implement in *DART / PDAF / JEDI* → test in real model → operational use



New ideas for nonlinear filtering for large dimensional systems, but a lot of them stuck at Python prototype phase…

Next-generation Ensemble Data Assimilation System
NEDAS enters the market

Conception → Python prototype → test in real models → implement in DART / PDAF / JEDI → operational use

Python code is light-weight and easier to maintain

`mpi4py`, `numpy`, `numba.jit` allow scalability and efficiency

operating system integration, error handling and testing

plenty of open-source libraries: machine learning, optimization…

NEDAS implementation

Sequential DA with pause-restart strategy

Require: $\mathbf{\Psi}(t_0), \boldsymbol{\varphi}^o(t_{1:N_t}), \mathbf{R}(t_{1:N_t})$

for $n = 1, \ldots, N_t$ **do**

  $\mathbf{\Psi}(t_{n-1}) \leftarrow \mathcal{P}[\mathbf{\Psi}(t_{n-1})]$

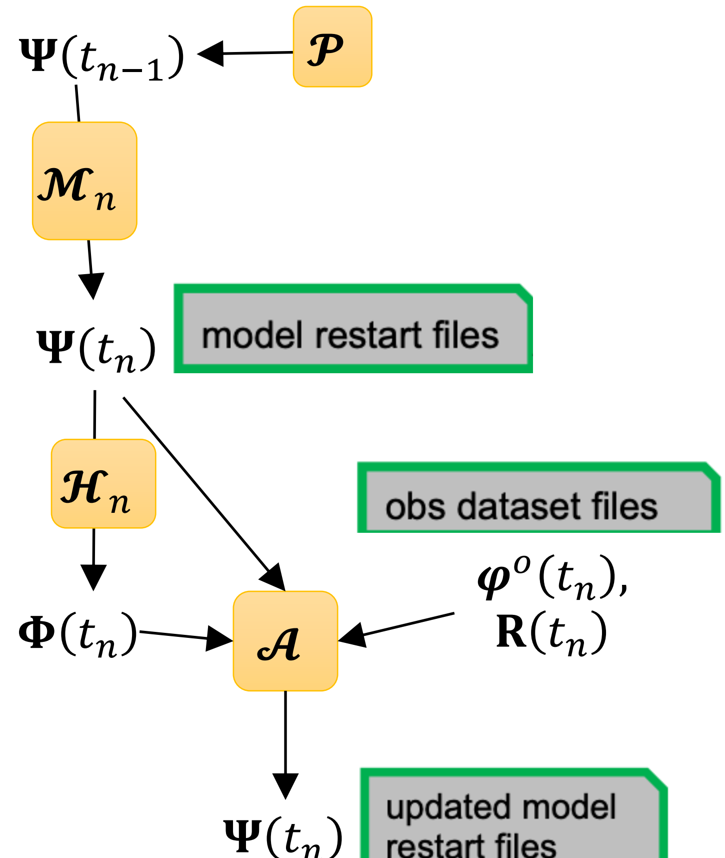
  $\mathbf{\Psi}(t_n) = \mathcal{M}_n\left[\mathbf{\Psi}(t_{n-1})\right]$

  $\mathbf{\Phi}(t_n) = \mathcal{H}_n\left[\mathbf{\Psi}(t_n)\right]$

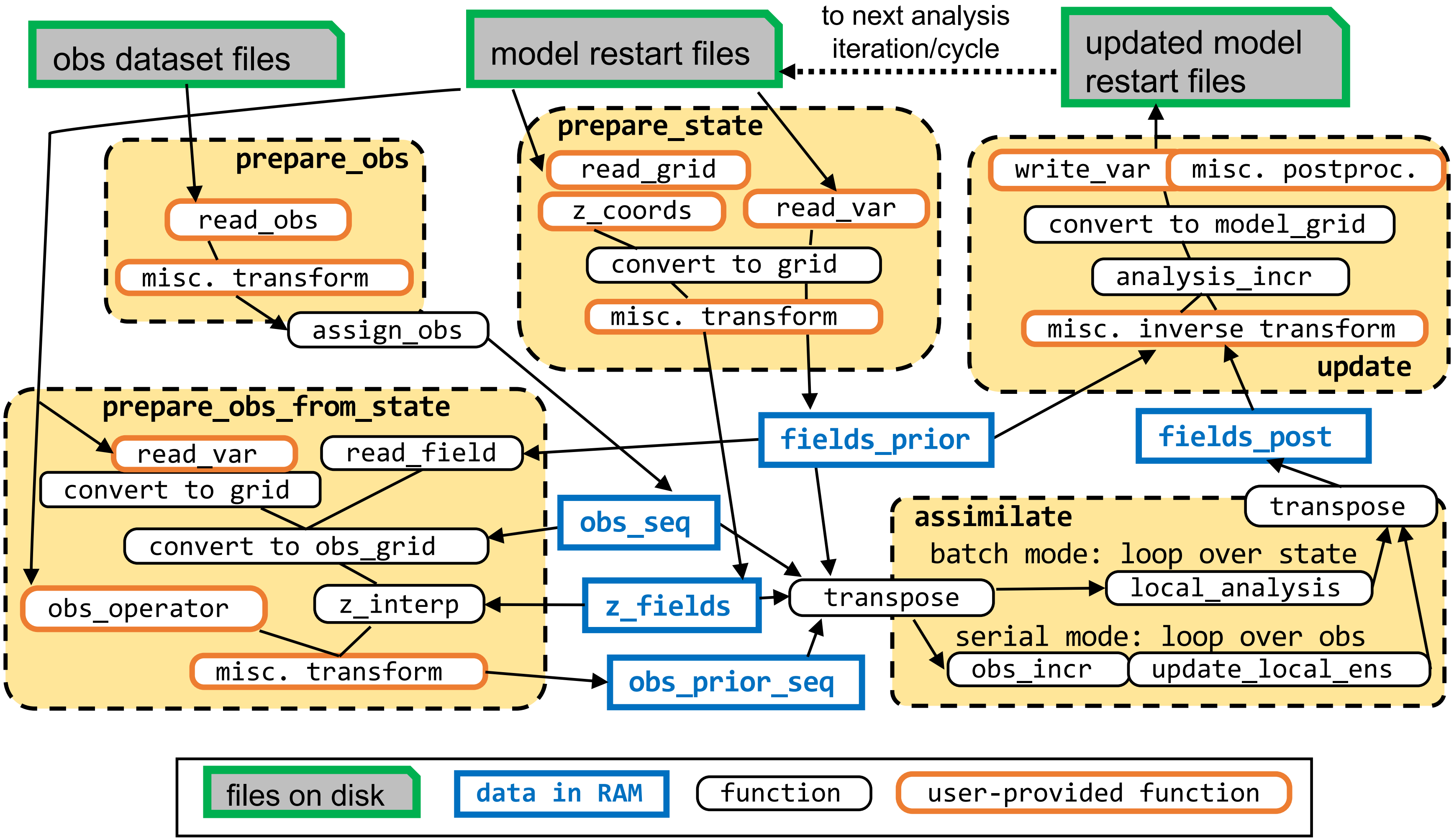
  $\mathbf{\Psi}(t_n) \leftarrow \mathcal{A}[\mathbf{\Psi}(t_n), \mathbf{\Phi}(t_n), \boldsymbol{\varphi}^o(t_n), \mathbf{R}(t_n)]$

end for

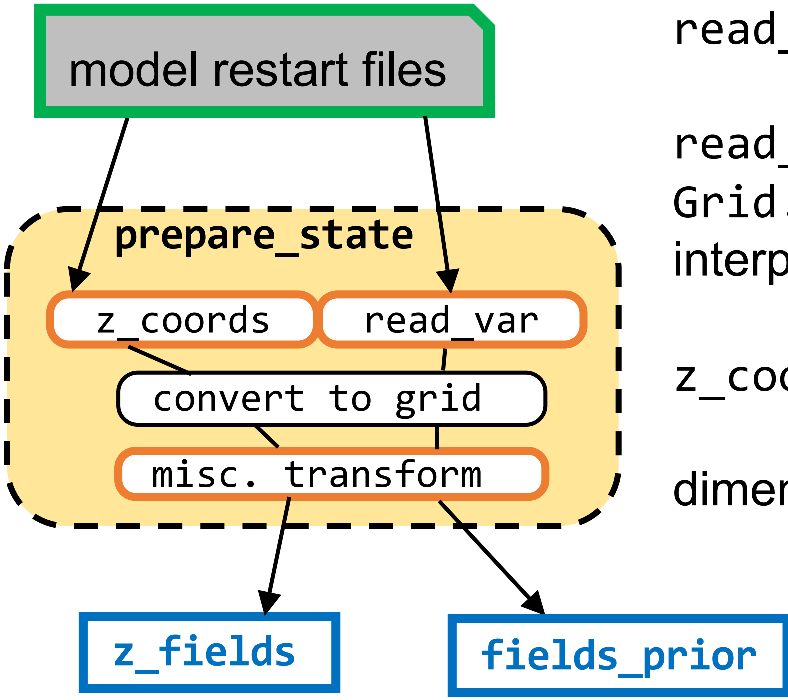
return $\mathbf{\Psi}(t_{1:N_t})$



NEDAS implementation: more details



model.<model_name> modules



`read_var`: reading a variable **field** from restart

`read_grid`: getting the 2D grid
`Grid.convert`: supports many grid types; chaches interpolation coeffs for efficiency

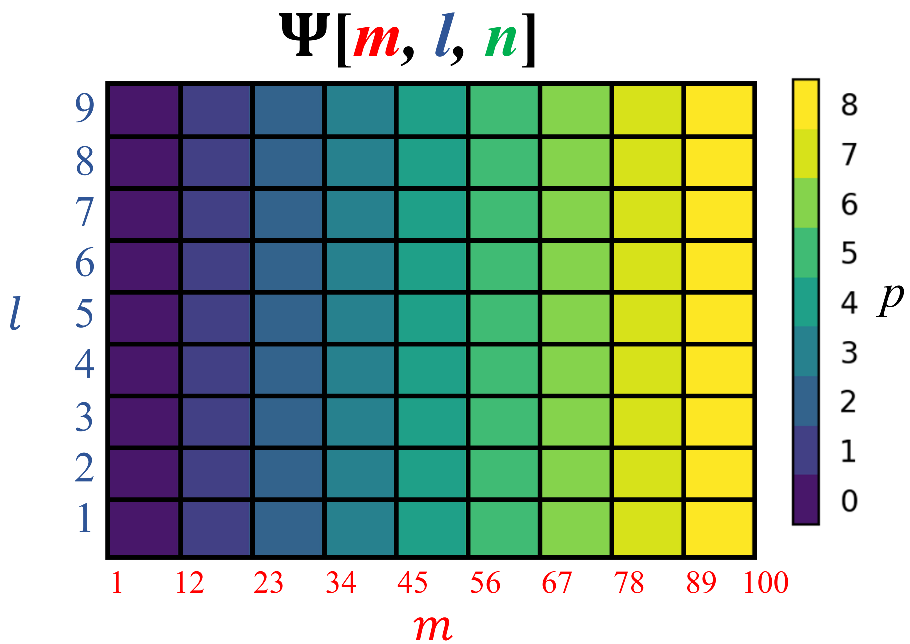
`z_coords`: getting the z coordinate fields

dimensions: member, location, `(x,y)`, field record `(z,time,varname)`

$$\Psi[\underbrace{m,}_{N_e}\ \underbrace{l,\ n}_{N_{\text{state}}}]$$

The smallest I/O task for a processor is to read a field with record index n and member index m

Parallel processing of state data



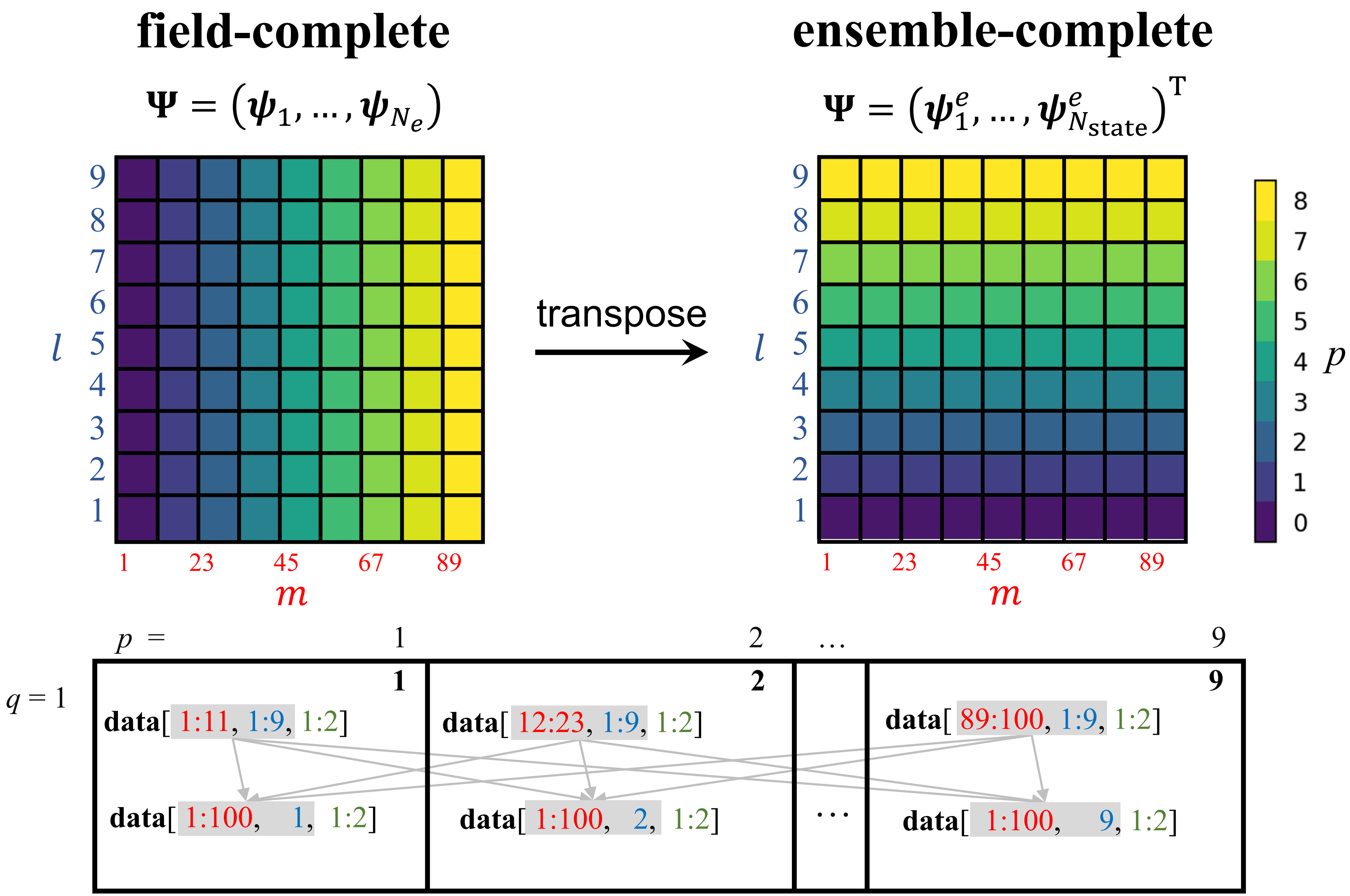
In this example:

m = 1…100 members
l = 1…9 spatial locations
n = 1…4 field records

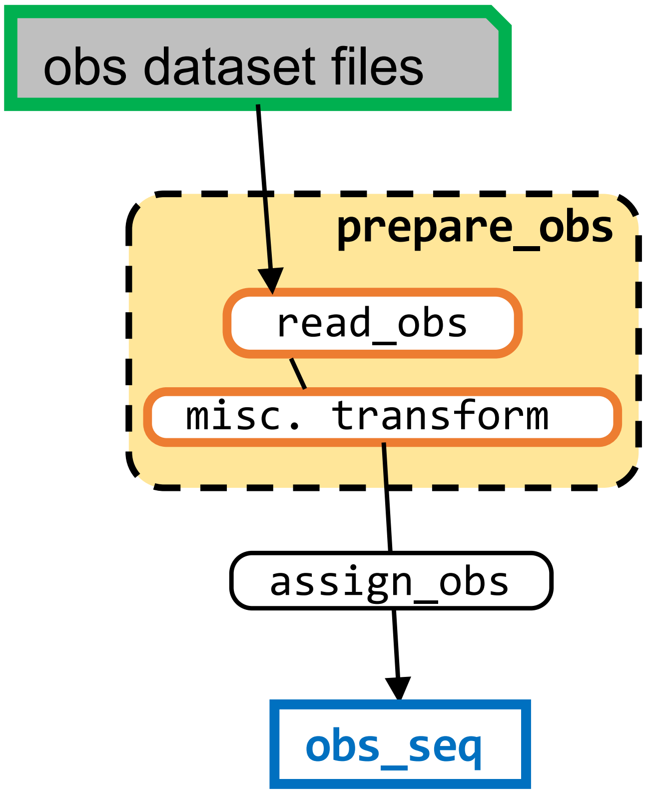
Nq = 2 groups of Np = 9 processors, total 18 processors

The data is "**field-complete**" (also called "state-complete" in Anderson & Collins 2007)

	p = 1	2	…	9
q = 1	**1** **data**[1:11, 1:9, 1:2]	**2** **data**[12:23, 1:9, 1:2]		**9** **data**[89:100, 1:9, 1:2]
2	**10** **data**[1:11, 1:9, 3:4]	**11** **data**[12:23, 1:9, 3:4]		**18** **data**[89:100, 1:9, 3:4]


field-complete
Ψ = (ψ1, ..., ψNe)
ensemble-complete
Ψ = (ψ1e, ..., ψeNstate)T
transpose
l
m
1 23 45 67 89
1 2 3 4 5 6 7 8 9
0 1 2 3 4 5 6 7 8
p
p = 1 2 … 9
q = 1
1
data[1:11, 1:9, 1:2]
data[1:100, 1, 1:2]
2
data[12:23, 1:9, 1:2]
data[1:100, 2, 1:2]
…
9
data[89:100, 1:9, 1:2]
data[1:100, 9, 1:2]

dataset.<dataset_name> modules



`read_obs` reads obs dataset, preprocessing

`random_network` generates synthetic obs network

`assign_obs` finds indices of “local obs”

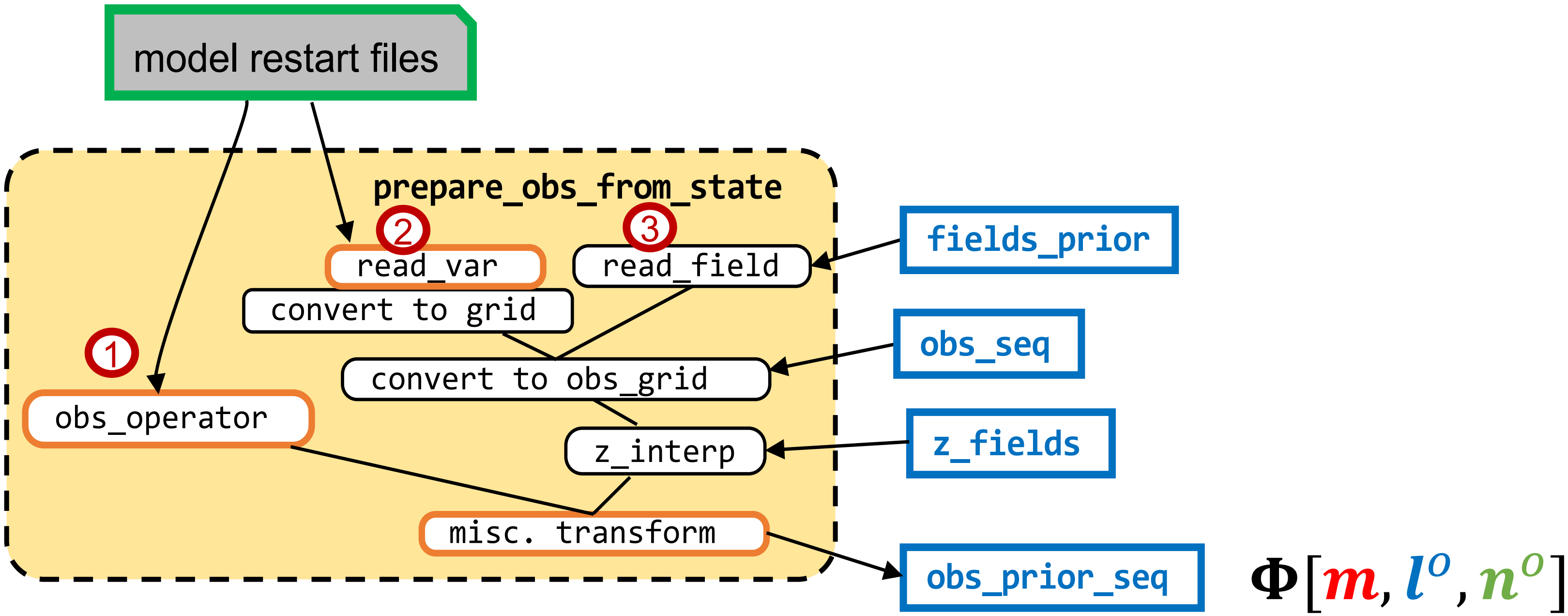
misc. transform: placeholder for algorithmic flexibility

dimensions	location	record
	`(x,y,z)`	`(time,varname)`

$$\boldsymbol{\varphi}^o[\ \boldsymbol{l}^o,\ \boldsymbol{n}^o\ ]$$

processor $p = 0$ is in charge of reading the obs

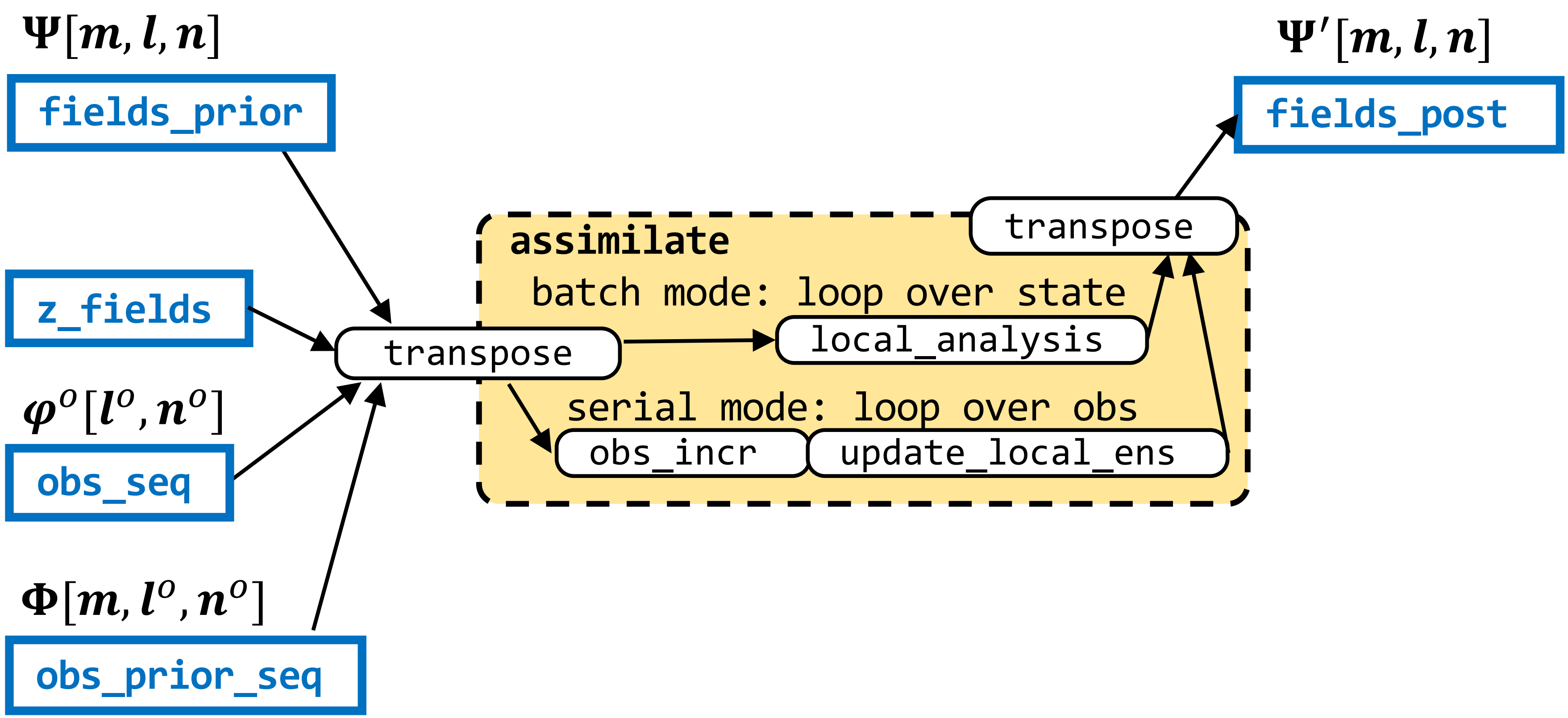
Observation operator $\mathbf{\Phi} = \mathcal{H}(\mathbf{\Psi})$



(1): obs_operator provided by the dataset module (space-time integral, nonlinear function, etc.)
(2): obs fields from read_var provided by the model module
(3): obs is one of the state variable: just read from fields_prior

Same parallel processing as the state:
each processor reads an obs network with record index n^o and member index m

Assimilation algorithm $\mathcal{A}(\boldsymbol{\Psi}, \boldsymbol{\Phi}, \boldsymbol{\varphi}^o)$



EnKF implementation

Make Gauss-linear assumption $p(\boldsymbol{\psi}) = \mathcal{N}\left(\overline{\boldsymbol{\psi}}, \mathbf{C}_{\psi\psi}\right)$
where $\overline{\psi} = \mathbf{\Psi}\mathbf{1}^{\mathrm{T}}/N_e$ and $\mathbf{C}_{\psi\psi} = \mathbf{\Psi}\left(\mathbf{\Psi} - \overline{\psi}\mathbf{1}^{\mathrm{T}}\right)^{\mathrm{T}}/(N_e - 1)$

and observation $\boldsymbol{\varphi}^o = \mathcal{H}(\boldsymbol{\psi}^{\mathrm{tr}}) + \boldsymbol{\varepsilon}^o$, $\boldsymbol{\varepsilon}^o \sim \mathcal{N}(\mathbf{0}, \mathbf{R})$

Algorithm:

$\mathbf{\Phi} = \mathcal{H}(\mathbf{\Psi})$

$\mathbf{\Psi} \leftarrow \mathbf{\Psi} + \mathbf{C}_{\psi,\varphi}\left(\mathbf{C}_{\varphi,\varphi} + \mathbf{R}\right)^{-1}(\mathbf{\Phi}^o - \mathbf{\Phi})$ (perturbed-obs EnKF)

There are two different strategies in parallelization:
- batch assimilation
- serial assimilation

Parallelization strategy

Batch assimilation (e.g. PDAF)

for $i = 1, \ldots, N_{\text{state}}$:

$$\mathbf{S} = \mathbf{R}^{-1/2}\left(\mathbf{\Phi} - \overline{\boldsymbol{\varphi}}\mathbf{1}^{\mathrm{T}}\right) \circ (\boldsymbol{\rho}\mathbf{1}^{\mathrm{T}}) / \sqrt{N_e - 1}$$

$$\boldsymbol{d} = \mathbf{R}^{-1/2}(\boldsymbol{\varphi}^o - \overline{\boldsymbol{\varphi}}) \circ \boldsymbol{\rho} / \sqrt{N_e - 1}$$

$$\mathbf{\Xi} = \left(\mathbf{I} + \mathbf{S}^{\mathrm{T}}\mathbf{S}\right)^{-1}$$

$$\mathbf{T} = \mathbf{\Xi}\mathbf{S}^{\mathrm{T}}\boldsymbol{d}\mathbf{1}^{\mathrm{T}} + \mathbf{\Xi}^{1/2}$$

$$\boldsymbol{\psi}_i^{e\,T} \leftarrow \boldsymbol{\psi}_i^{e\,T}\mathbf{T}$$

return $\mathbf{\Psi} = \left(\boldsymbol{\psi}_1^e, \ldots, \boldsymbol{\psi}_{N_{\text{state}}}^e\right)^{\mathrm{T}}$

cost: $\mathcal{O}(N_{\text{lobs}} N_e^2 + N_e^3)\ \times N_{\text{state}}$

“local analysis”

Serial assimilation (e.g. DART)

for $j = 1, \ldots, N_{\text{obs}}$:

$$\xi = \sigma_{o,j}^2 / (\sigma_j^2 + \sigma_{o,j}^2)$$

$$\boldsymbol{\delta}_j^e = \xi\overline{\varphi}_j + (1 - \xi)\varphi_j^o + \sqrt{\xi}\left(\boldsymbol{\varphi}_j^e - \overline{\varphi}_j\right) - \boldsymbol{\varphi}_j^e$$

broadcast $\boldsymbol{\delta}_j^e$

$$\mathbf{\Psi} \leftarrow \mathbf{\Psi} + \left(\boldsymbol{\rho}^{\psi} \circ \boldsymbol{c}_{\psi,\varphi_j} / \sigma_j^2\right) \boldsymbol{\delta}_j^{e\,\mathrm{T}}$$

$$\mathbf{\Phi} \leftarrow \mathbf{\Phi} + \left(\boldsymbol{\rho}^{\varphi} \circ \boldsymbol{c}_{\varphi,\varphi_j} / \sigma_j^2\right) \boldsymbol{\delta}_j^{e\,\mathrm{T}}$$

return $\mathbf{\Psi}$

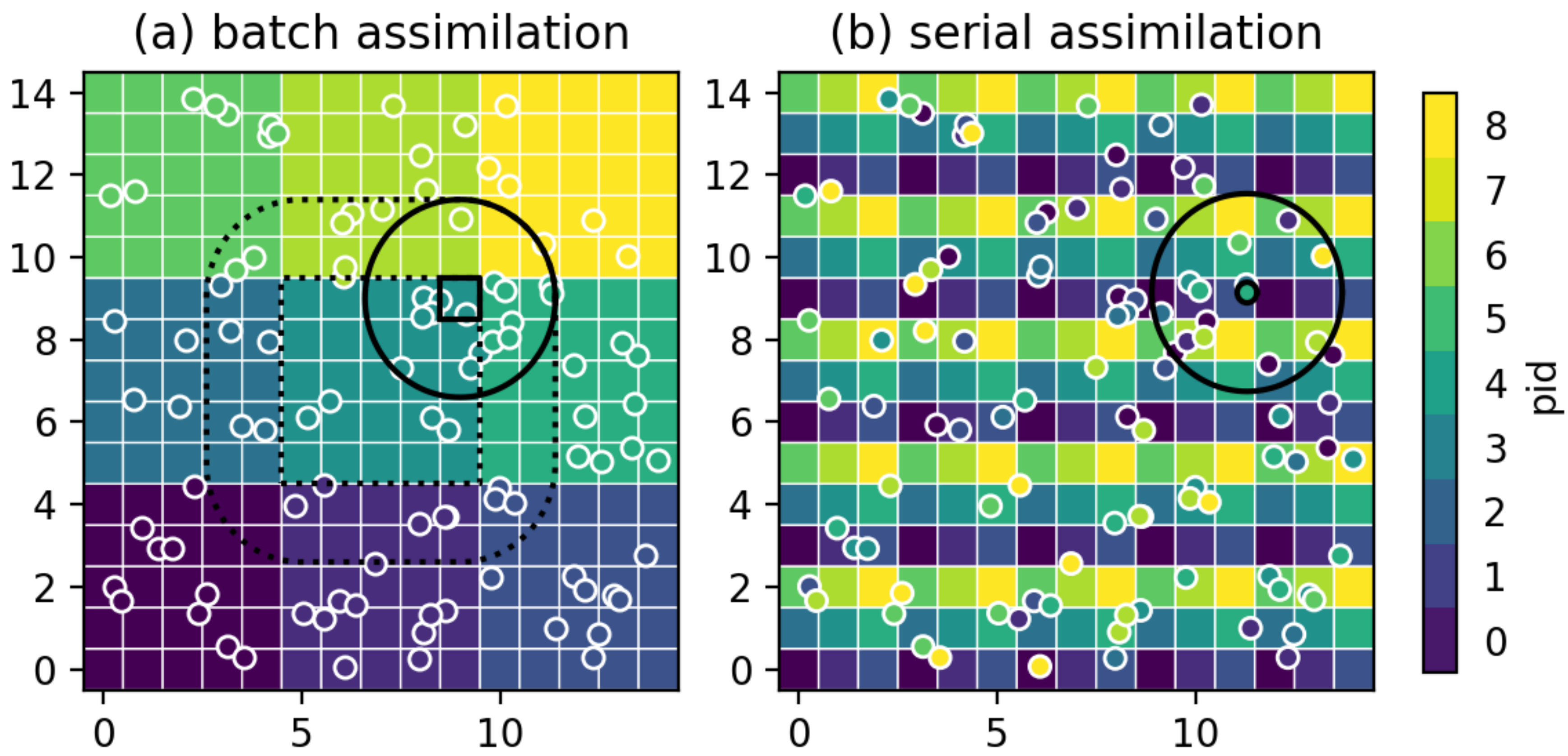
cost:

$\mathcal{O}\left(N_e \log N_p + N_e N_{\text{lstate}} + N_e N_{\text{lobs}}\right)\ \times N_{\text{obs}}$

“obs_incr”: nonlinear filters possible

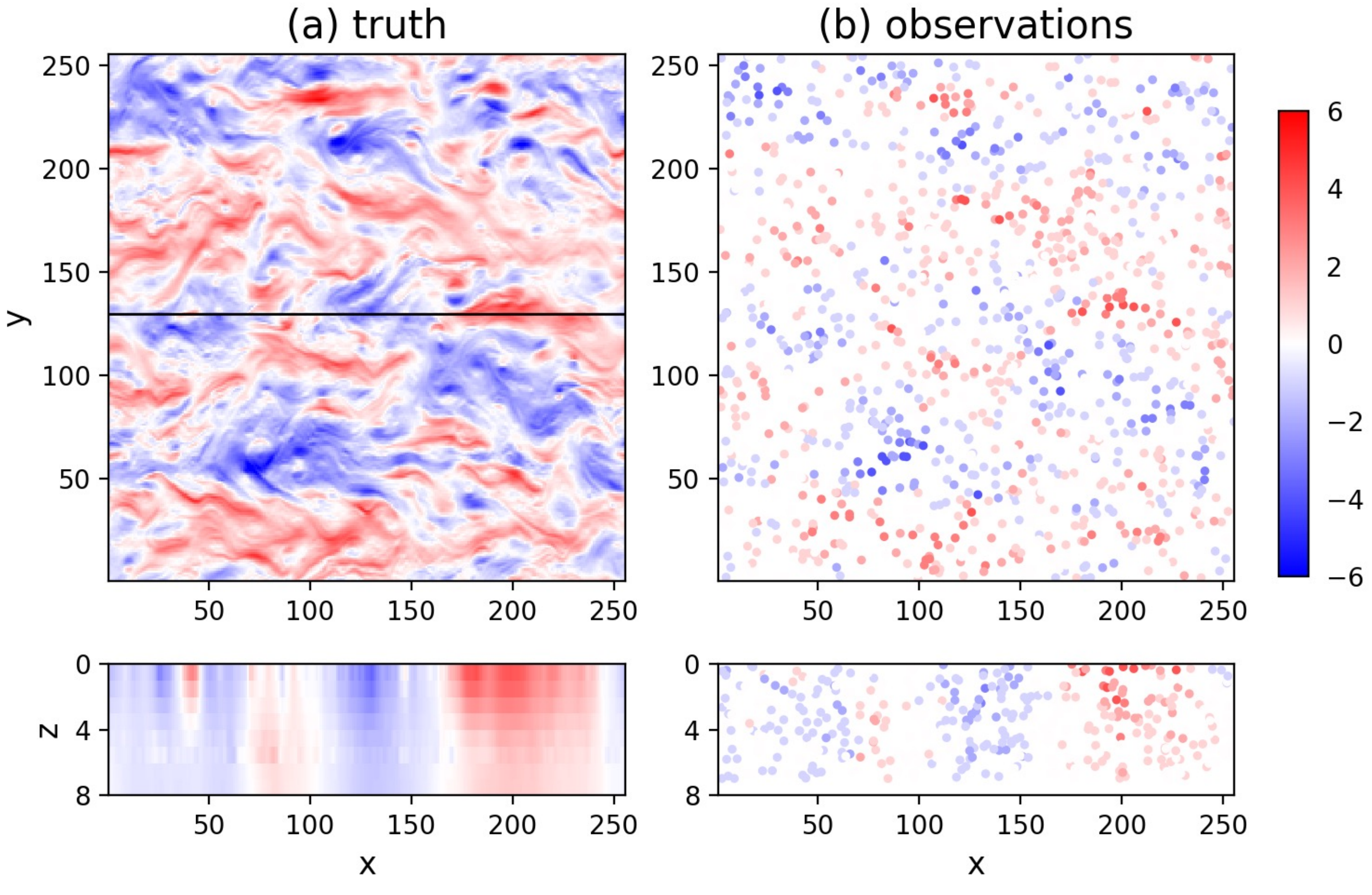
“obs_updates_ens”: linear, probit-space

Memory layout for state/obs

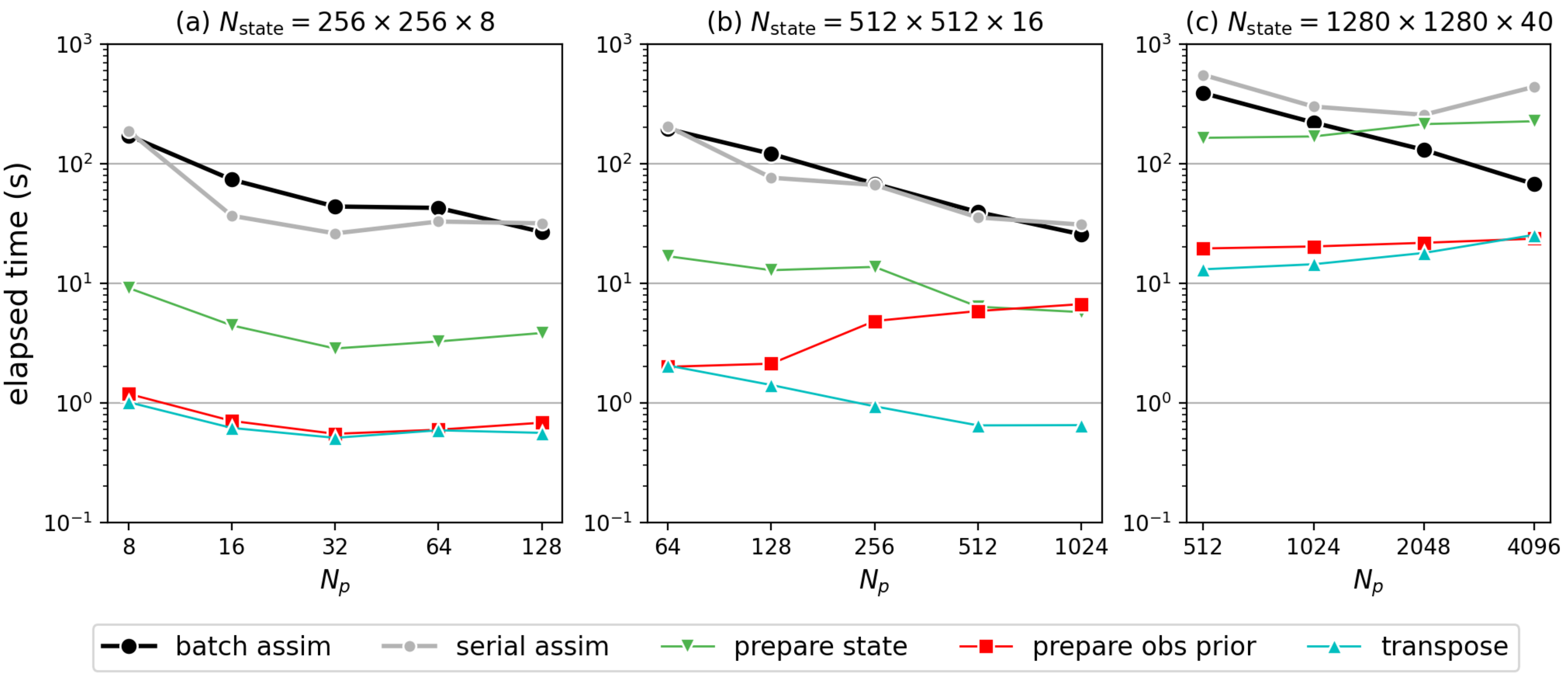


Benchmarking: QG model example

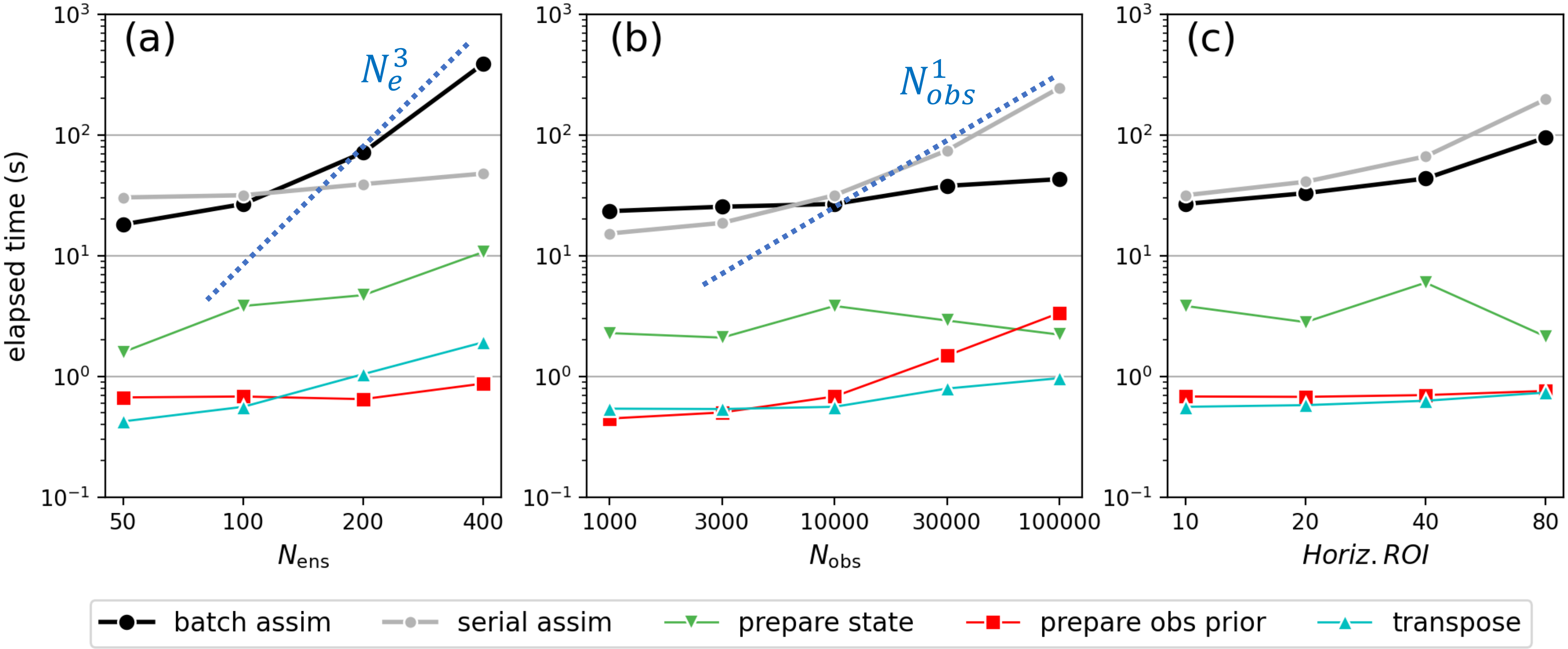
state and observations: velocity fields
Nstate = 256x256x8, Nobs = 10000, obs_err = 0.5, Ne = 100, hroi=10, vroi=5



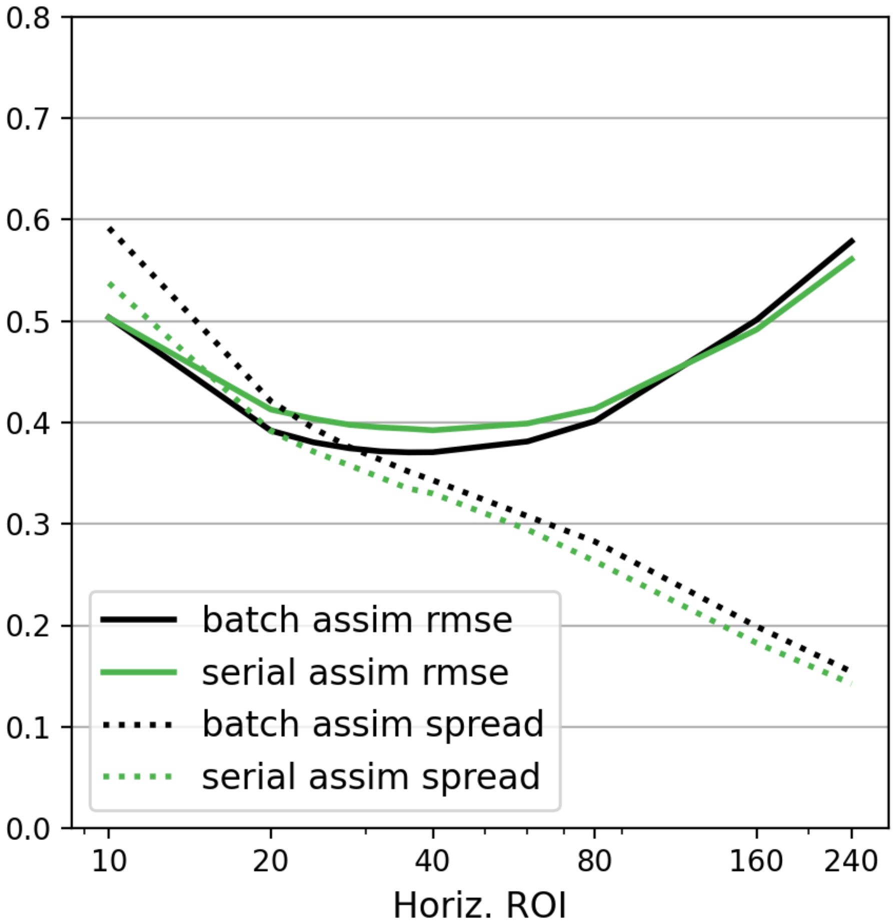
Scalability of $\mathcal{A}$ as N_p increases



How $\mathcal{A}$ scales as dimensionality increases



Analysis error/spread comparison



Both strategies produced comparable results

Serial assimilation fits more to observations (lower posterior spread); slightly less accurate (higher rmse)

Consistent with previous findings (Holland & Wang 2012; Nerger 2015).

Algorithmic flexibility:
Miscellaneous transform functions $\mathcal{T}$

for $s = 1, \ldots, N_s$ **do**

$\widetilde{\varphi}^o = \mathcal{T}_s^o(\varphi^o)$

for $m = 1, \ldots, N_e$ **do**

$\widetilde{\varphi}_m = \mathcal{T}_s^o(\varphi_m)$

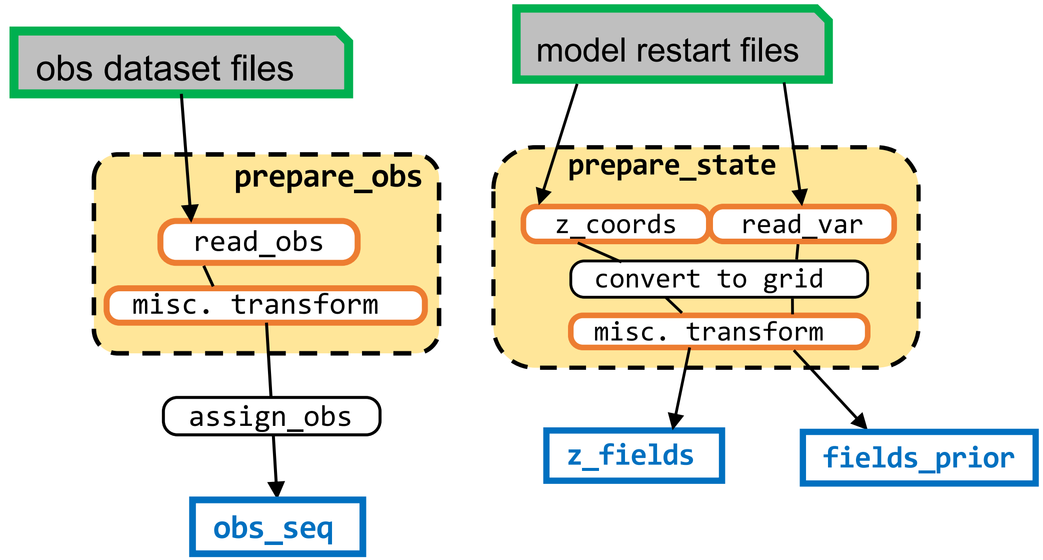
$\widetilde{\psi}_m = \mathcal{T}_s(\psi_m)$

end for

$\widetilde{\mathbf{\Psi}} = (\widetilde{\psi}_1, \ldots, \widetilde{\psi}_{N_e})$

$\widetilde{\mathbf{\Phi}} = (\widetilde{\varphi}_1, \ldots, \widetilde{\varphi}_{N_e})$

$(\widetilde{\psi}'_1, \ldots, \widetilde{\psi}'_{N_e}) = \widetilde{\mathbf{\Psi}}' = \widetilde{\mathcal{A}}(\widetilde{\mathbf{\Psi}}, \widetilde{\mathbf{\Phi}}, \widetilde{\varphi}^o, \widetilde{\mathbf{R}}_s, \boldsymbol{r}_s^{\psi}, \boldsymbol{r}_s^{\varphi}, \boldsymbol{L}_s)$

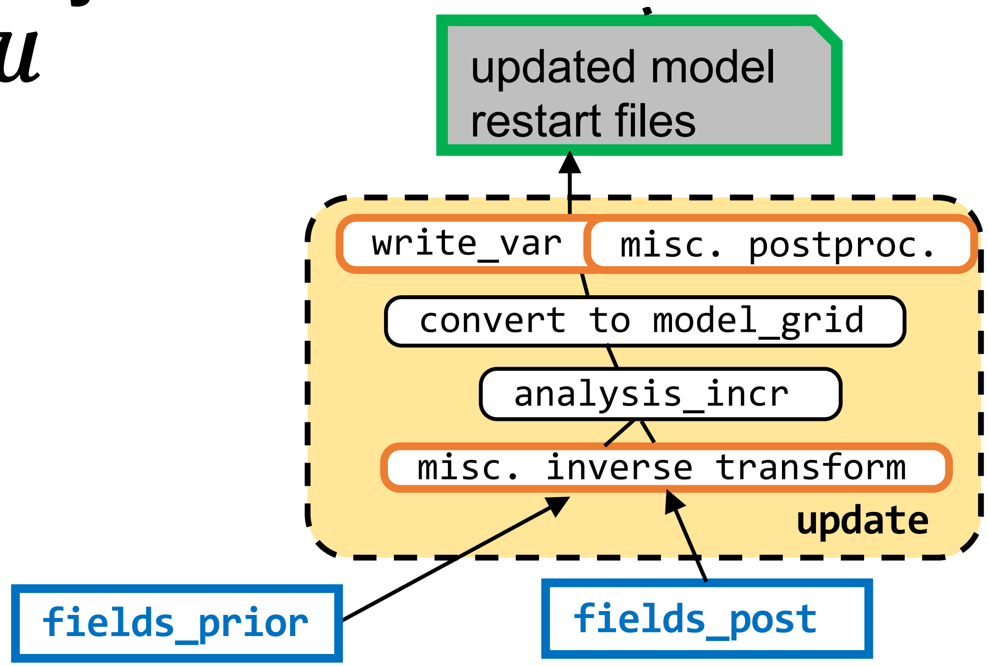


- Gaussian anamorphosis (Simon & Bertino 2009)
- Multiscale decomposition (Ying 2019, 2020), gcm-filters (Grooms et al. 2021)
- Super-resolution (Barthelemy et al 2022)
- Mapping to latent space (Chipilski 2023)

…

Algorithmic flexibility: Update functions $\mathcal{U}$

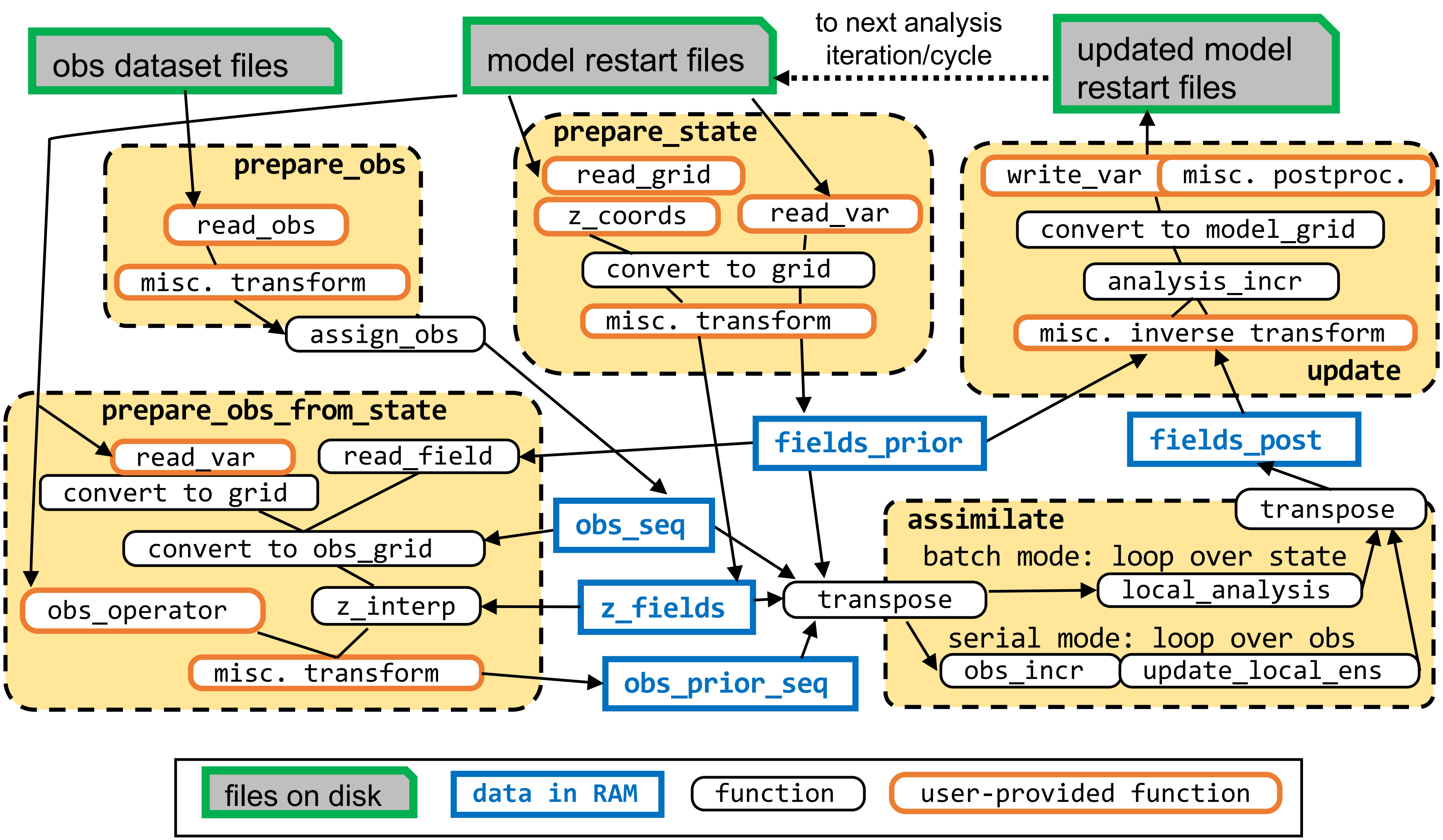
for $s = 1, \ldots, N_s$ **do**
  $\widetilde{\varphi}^o = \mathcal{T}_s^o(\varphi^o)$
  for $m = 1, \ldots, N_e$ **do**
    $\widetilde{\varphi}_m = \mathcal{T}_s^o(\varphi_m)$
    $\widetilde{\psi}_m = \mathcal{T}_s(\psi_m)$
  end for
  $\widetilde{\boldsymbol{\Psi}} = (\widetilde{\psi}_1, \ldots, \widetilde{\psi}_{N_e})$
  $\widetilde{\boldsymbol{\Phi}} = (\widetilde{\varphi}_1, \ldots, \widetilde{\varphi}_{N_e})$
  $(\widetilde{\psi}'_1, \ldots, \widetilde{\psi}'_{N_e}) = \widetilde{\boldsymbol{\Psi}}' = \widetilde{\mathcal{A}}(\widetilde{\boldsymbol{\Psi}}, \widetilde{\boldsymbol{\Phi}}, \widetilde{\varphi}^o, \widetilde{\mathbf{R}}_s, \boldsymbol{r}_s^\psi, \boldsymbol{r}_s^\varphi, \boldsymbol{L}_s)$
  for $m = 1, \ldots, N_e$ **do**
    $\psi_m \leftarrow \mathcal{U}_s(\psi_m, \widetilde{\psi}_m, \widetilde{\psi}'_m)$
  end for
end for



- Inverse transform, additive increments
- Alignment techniques (Ying 2019)

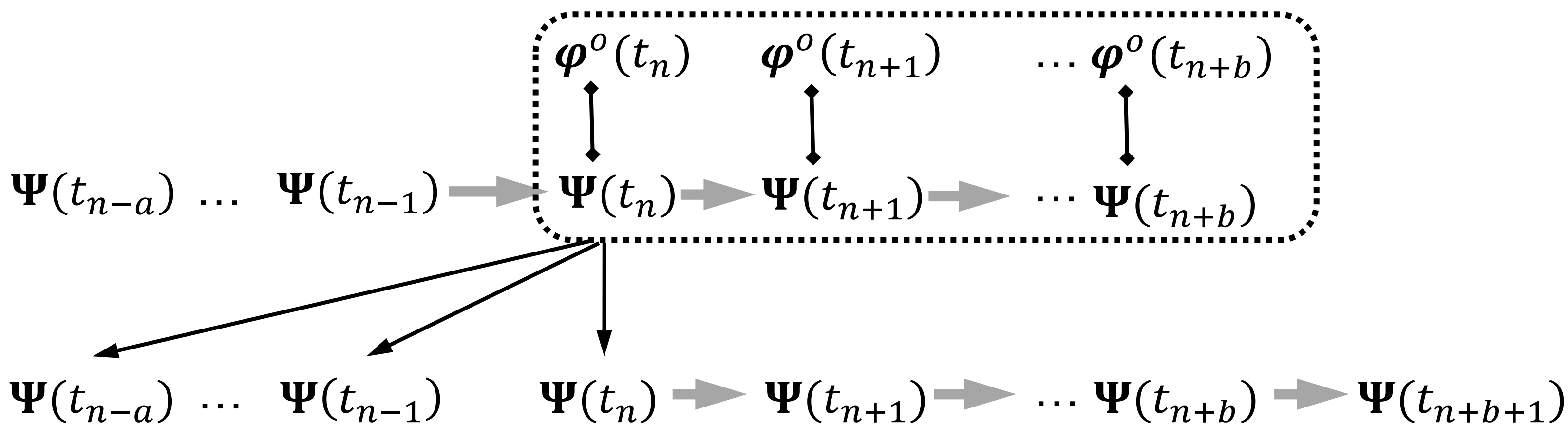
- Update not only model states, but also hyperparameters.

NEDAS analysis workflow



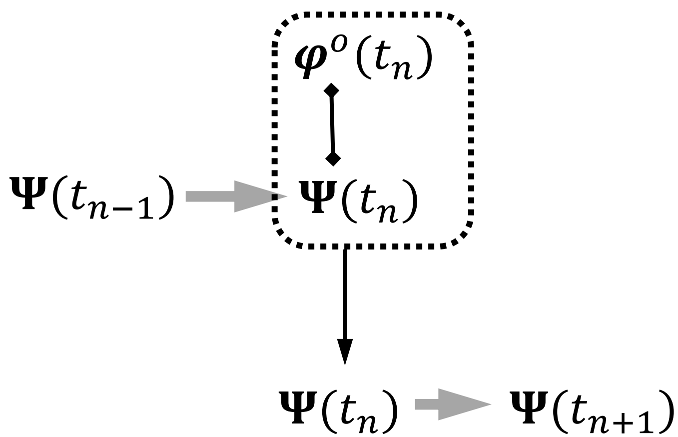
NEDAS sequential DA includes 4D model states

$$
\begin{array}{l}
\textbf{for } n = 1, \dots, N_t \ \ \textbf{do} \\
\quad \boldsymbol{\Psi}(t_{n-1}) \leftarrow \mathcal{P}[\boldsymbol{\Psi}(t_{n-1})] \\
\quad \textbf{for } k = 0, \dots, b \ \textbf{do} \\
\quad\quad \boldsymbol{\Psi}(t_{n+k}) = \mathcal{M}_{n+k}\left[\boldsymbol{\Psi}(t_{n+k-1})\right] \\
\quad\quad \boldsymbol{\Phi}(t_{n+k}) = \mathcal{H}_{n+k}\left[\boldsymbol{\Psi}(t_{n+k})\right] \\
\quad \textbf{end for} \\
\quad \boldsymbol{\Psi}(t_{n-a:n}) \leftarrow \mathcal{A}[\boldsymbol{\Psi}(t_{n-a:n}), \boldsymbol{\Phi}(t_{n:n+b}), \boldsymbol{\varphi}^o(t_{n:n+b}), (b+1)\mathbf{R}(t_{n:n+b})] \\
\textbf{end for} \\
\textbf{return } \boldsymbol{\Psi}(t_{1:N_t})
\end{array}
$$



$a = b = 0$: *filter*

for $n = 1, \dots, N_t$ **do**
  $\boldsymbol{\Psi}(t_{n-1}) \leftarrow \mathcal{P}[\boldsymbol{\Psi}(t_{n-1})]$
  for $k = 0, \dots, b$ **do**
    $\boldsymbol{\Psi}(t_{n+k}) = \mathcal{M}_{n+k}\left[\boldsymbol{\Psi}(t_{n+k-1})\right]$
    $\boldsymbol{\Phi}(t_{n+k}) = \mathcal{H}_{n+k}\left[\boldsymbol{\Psi}(t_{n+k})\right]$
  end for
  $\boldsymbol{\Psi}(t_{n-a:n}) \leftarrow \mathcal{A}[\boldsymbol{\Psi}(t_{n-a:n}), \boldsymbol{\Phi}(t_{n:n+b}), \boldsymbol{\varphi}^o(t_{n:n+b}), (b+1)\mathbf{R}(t_{n:n+b})]$
end for
return $\boldsymbol{\Psi}(t_{1:N_t})$



$b = 1, a > 0$: *recursive smoother*

Evensen & van Leeuwen 2000

for $n = 1, \ldots, N_t$ **do**

  $\boldsymbol{\Psi}(t_{n-1}) \leftarrow \mathcal{P}[\boldsymbol{\Psi}(t_{n-1})]$

  for $k = 0, \ldots, b$ **do**

    $\boldsymbol{\Psi}(t_{n+k}) = \mathcal{M}_{n+k}\left[\boldsymbol{\Psi}(t_{n+k-1})\right]$

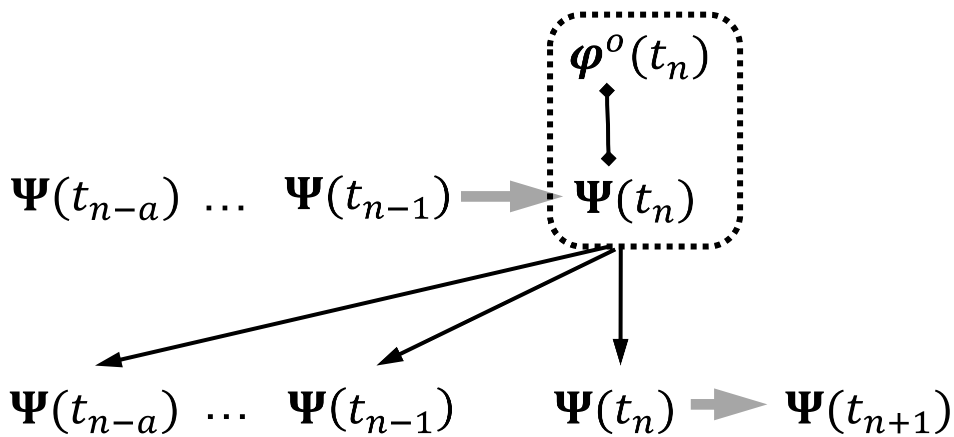
    $\boldsymbol{\Phi}(t_{n+k}) = \mathcal{H}_{n+k}\left[\boldsymbol{\Psi}(t_{n+k})\right]$

  end for

  $\boldsymbol{\Psi}(t_{n-a:n}) \leftarrow \mathcal{A}[\boldsymbol{\Psi}(t_{n-a:n}), \boldsymbol{\Phi}(t_{n:n+b}), \boldsymbol{\varphi}^o(t_{n:n+b}), (b+1)\mathbf{R}(t_{n:n+b})]$

end for

return $\boldsymbol{\Psi}(t_{1:N_t})$

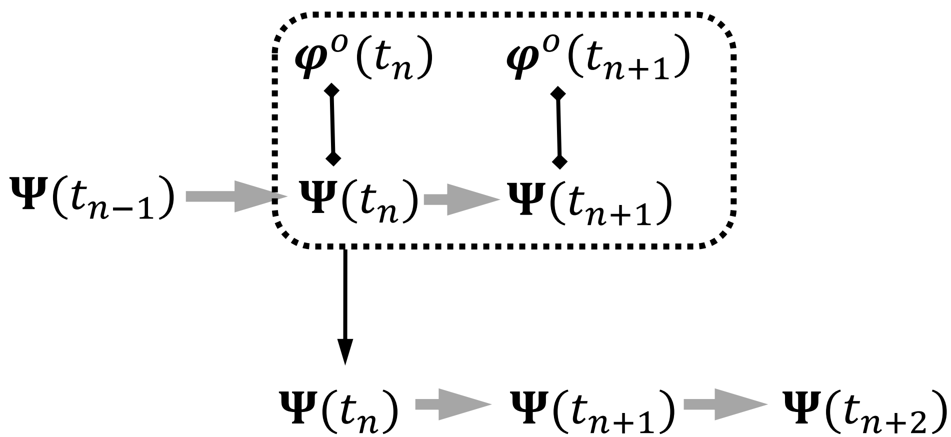


$b = 1, a = 0$: *one-step-ahead smoother*

for $n = 1, \dots, N_t$ **do**
    $\mathbf{\Psi}(t_{n-1}) \leftarrow \mathcal{P}[\mathbf{\Psi}(t_{n-1})]$
    for $k = 0, \dots, b$ **do**
        $\mathbf{\Psi}(t_{n+k}) = \mathcal{M}_{n+k}\left[\mathbf{\Psi}(t_{n+k-1})\right]$
        $\mathbf{\Phi}(t_{n+k}) = \mathcal{H}_{n+k}\left[\mathbf{\Psi}(t_{n+k})\right]$
    end for
    $\mathbf{\Psi}(t_{n-a:n}) \leftarrow \mathcal{A}[\mathbf{\Psi}(t_{n-a:n}), \mathbf{\Phi}(t_{n:n+b}), \boldsymbol{\varphi}^o(t_{n:n+b}), (b+1)\mathbf{R}(t_{n:n+b})]$
end for
return $\mathbf{\Psi}(t_{1:N_t})$

Desbouvries et al. 2011; Gharamti et al. 2015; Ait-El-Fquih & Hoteit 2022

More general smoother formulation:
Khare et al., 2008; Bocquet & Sakov, 2014; Grudzien & Bocquet, 2022



Summary

- NEDAS provides a light-weight Python solution for testing ensemble DA algorithms in real model settings.
- Both batch and serial assimilation strategies are efficient and scalable for large models, but they are favored in different scenarios.
- NEDAS has a modular design, which gives flexibility for integrating new algorithmic ideas in its workflow.

Code publicly available:
https://github.com/nansencenter/NEDAS

Manuscript submitted to JAMES:
Ying: “Introducing NEDAS: a light-weight and scalable Python solution for ensemble data assimilation”

Contact me: yue.ying@nersc.no