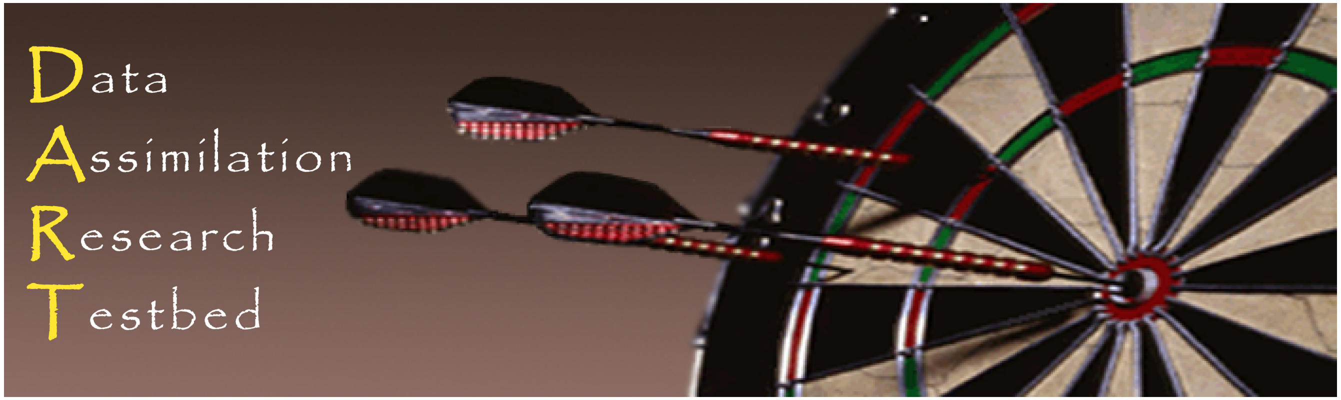
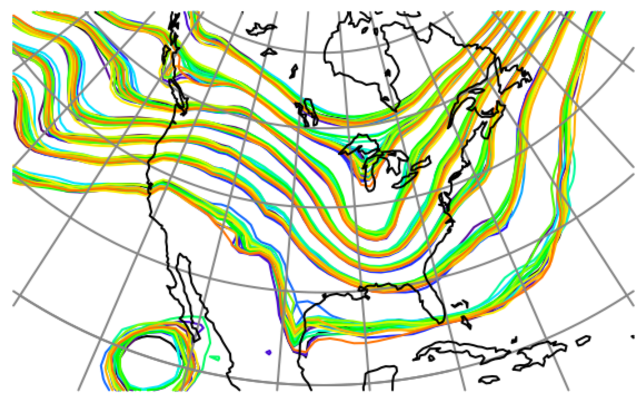


Parallel Implementations of Ensemble Kalman Filters for Huge Geophysical Models

Jeffrey Anderson, Helen Kershaw, Jonathan Hendricks, Nancy Collins, Ye Feng
NCAR Data Assimilation Research Section

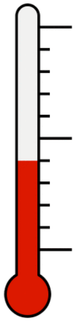


The National Center for Atmospheric Research is sponsored by the National Science Foundation. Any opinions, findings and conclusions or recommendations expressed in this publication are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

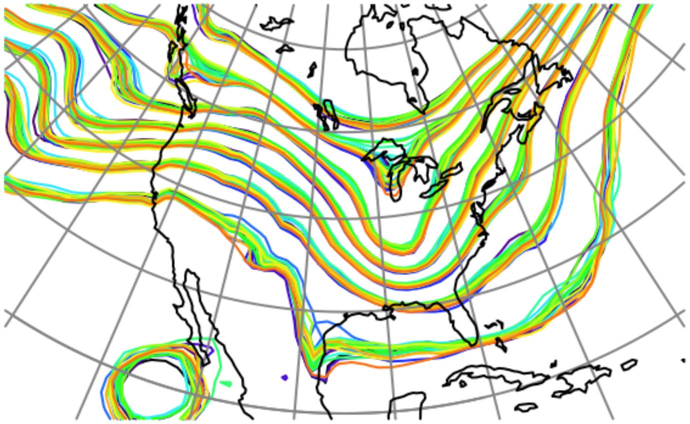
NCAR | UCAR | National Center for Atmospheric Research

What is Data Assimilation?

Observations combined with a Model forecast…



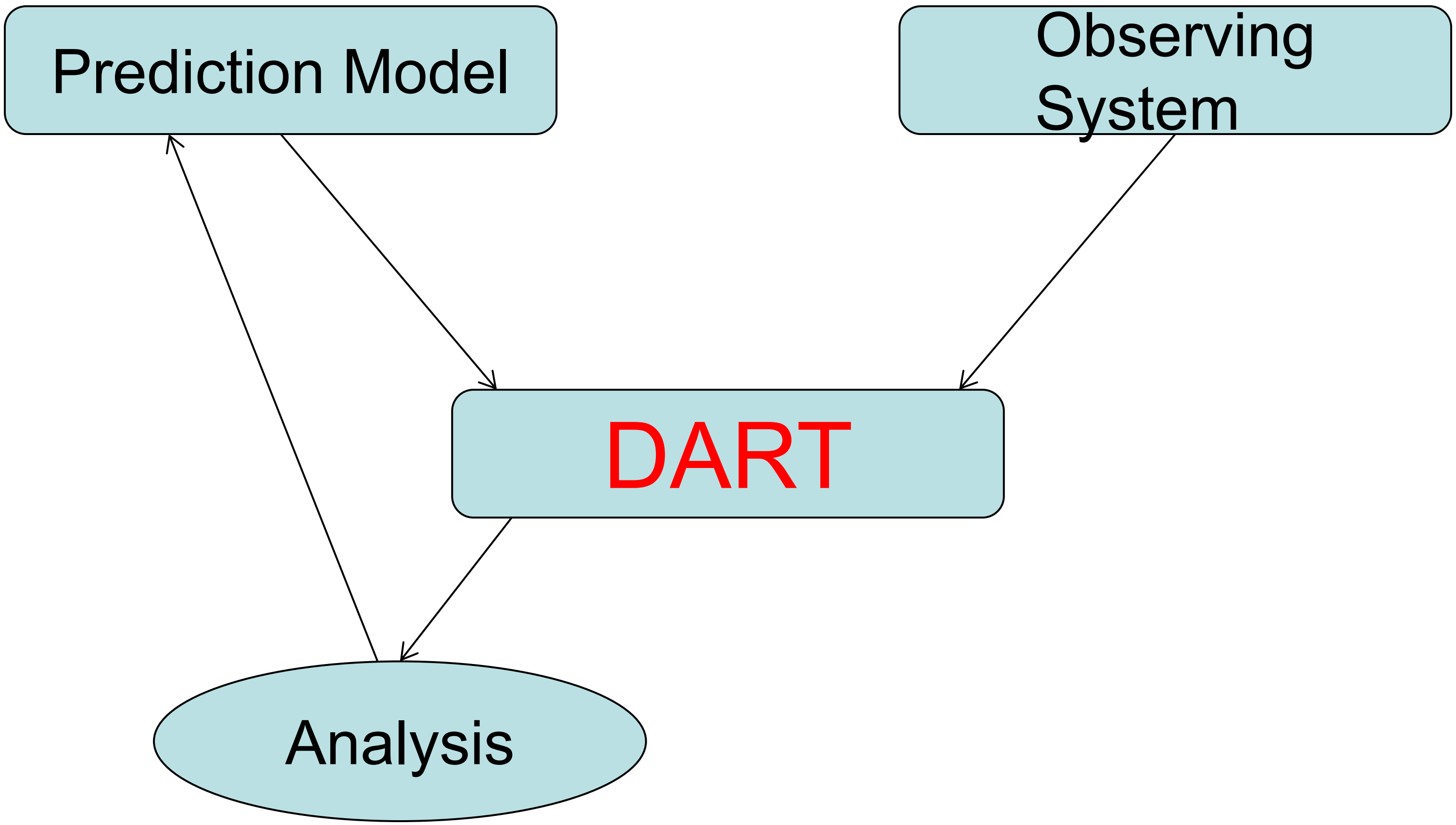
+



…to produce an analysis
(best possible estimate).

The Data Assimilation Research Testbed (DART)

DART provides data assimilation ‘glue’ to build state-of-the-art ensemble forecast systems for even the largest models.



DART Goals

Provide State-of-the-Art Data Assimilation capability to:

- Prediction research scientists,
- Model developers,
- Observation system developers,

Who may not have any assimilation expertise.

DART Design Constraints

- Models small to huge.
- Few or many observations.
- Tiny to huge computational resources.
- Entry cost must be low.
- Competitive with existing methods for weather prediction:

  Scientific quality of results,

  Total computational effort must be competitive.

How an Ensemble Filter Works for Geophysical Data Assimilation

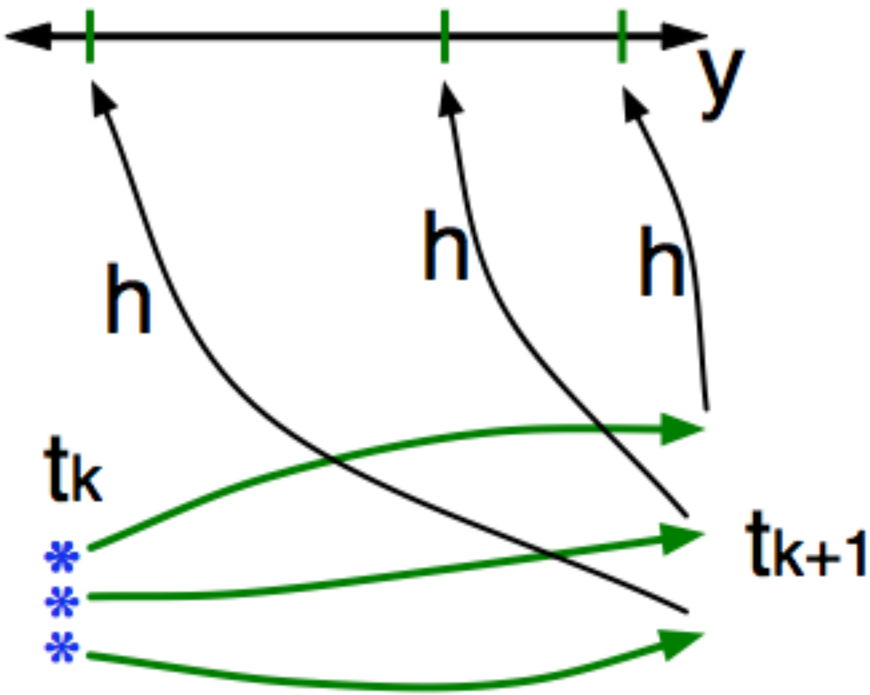
1. Use model to advance ensemble (3 members here) to time at which next observation becomes available.

Ensemble state estimate after using previous observation (analysis)

Ensemble state at time of next observation (prior)

How an Ensemble Filter Works for Geophysical Data Assimilation

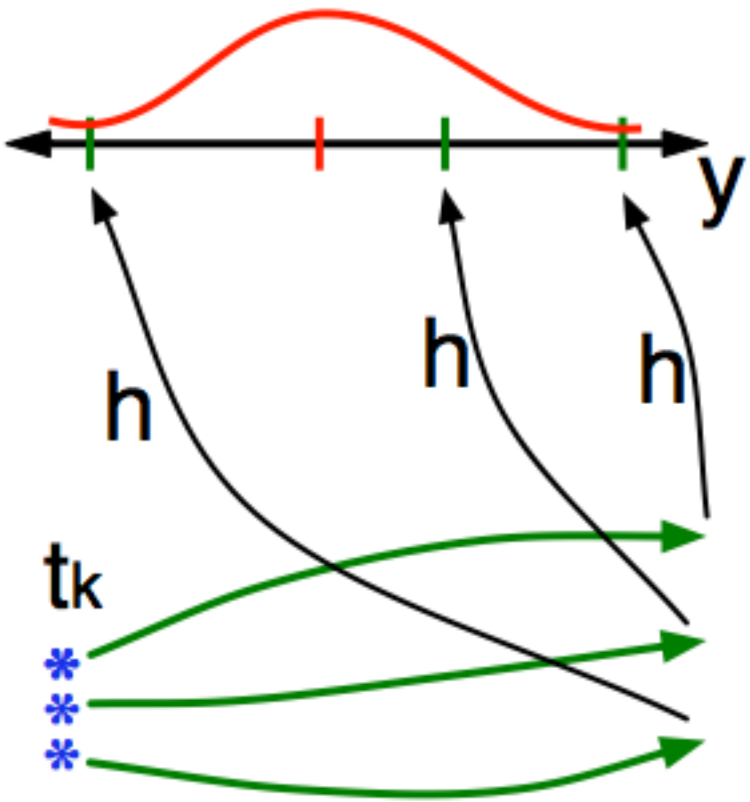
2. Get prior ensemble sample of observation, $y = h(x)$, by applying forward operator **h** to each ensemble member.



Theory: observations from instruments with uncorrelated errors can be done sequentially.

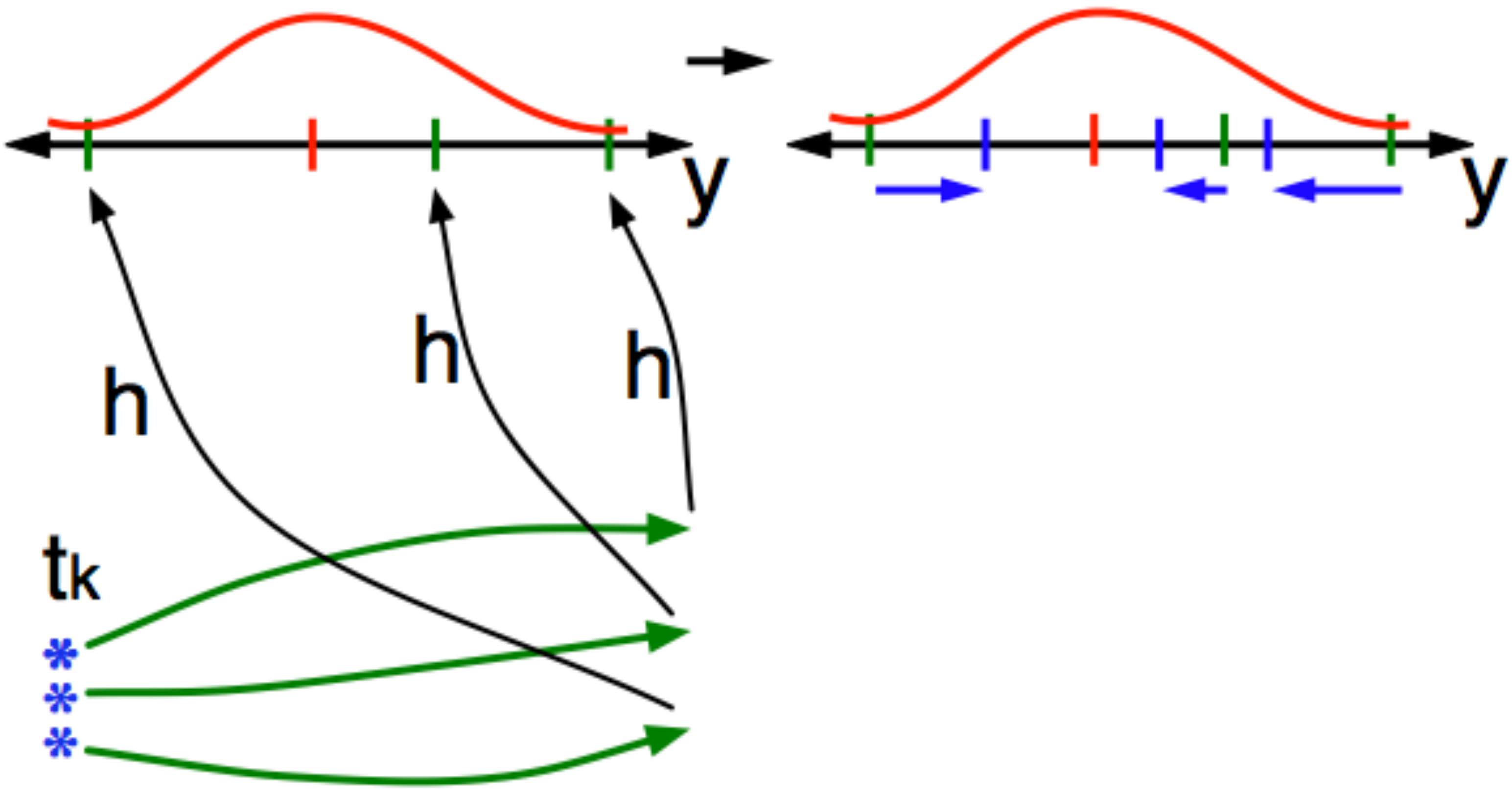
How an Ensemble Filter Works for Geophysical Data Assimilation

3. Get observed value and observational error distribution from observing system.



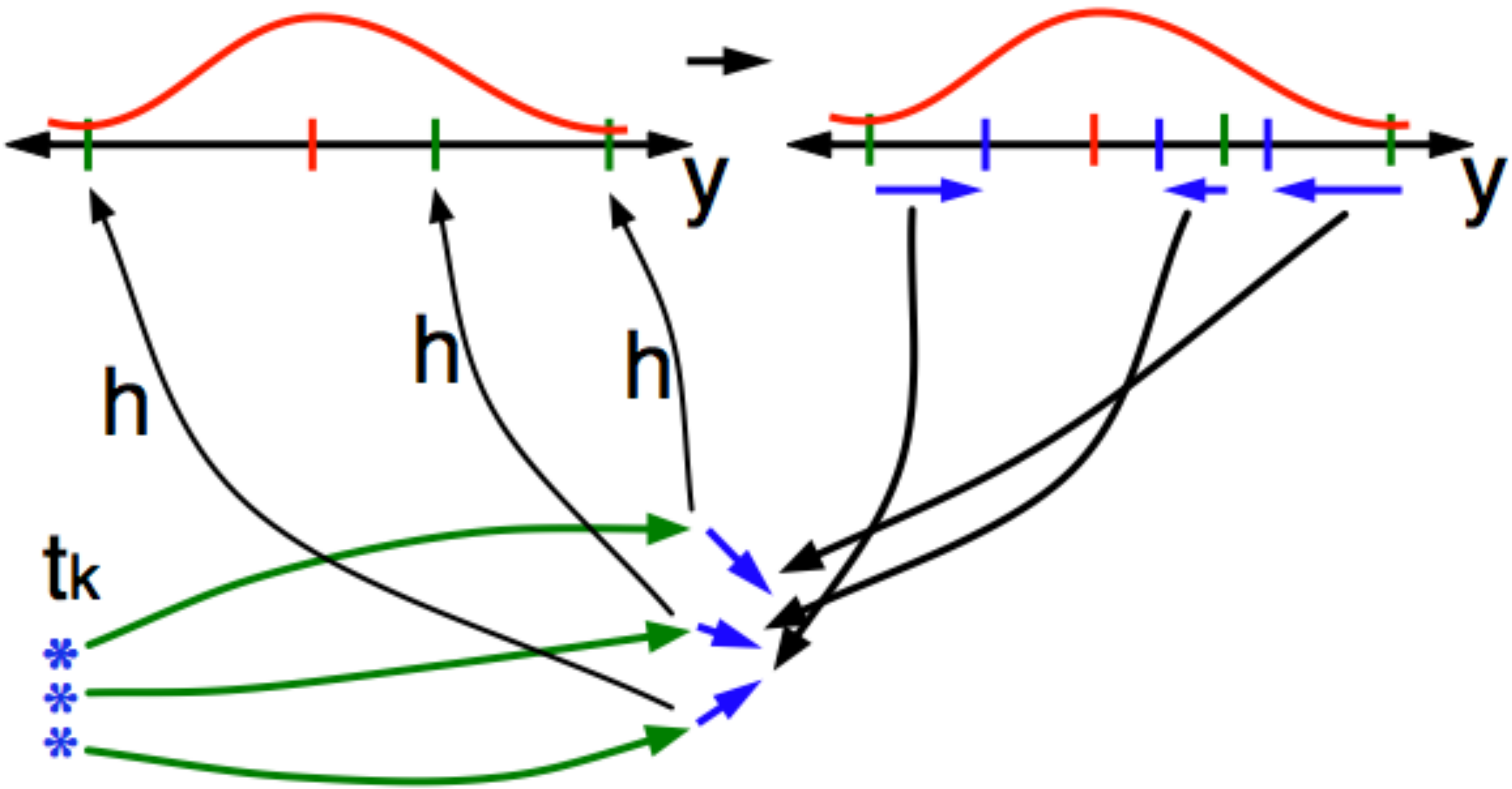
How an Ensemble Filter Works for Geophysical Data Assimilation

4. Find the increments for the prior observation ensemble (this is a scalar problem for uncorrelated observation errors).

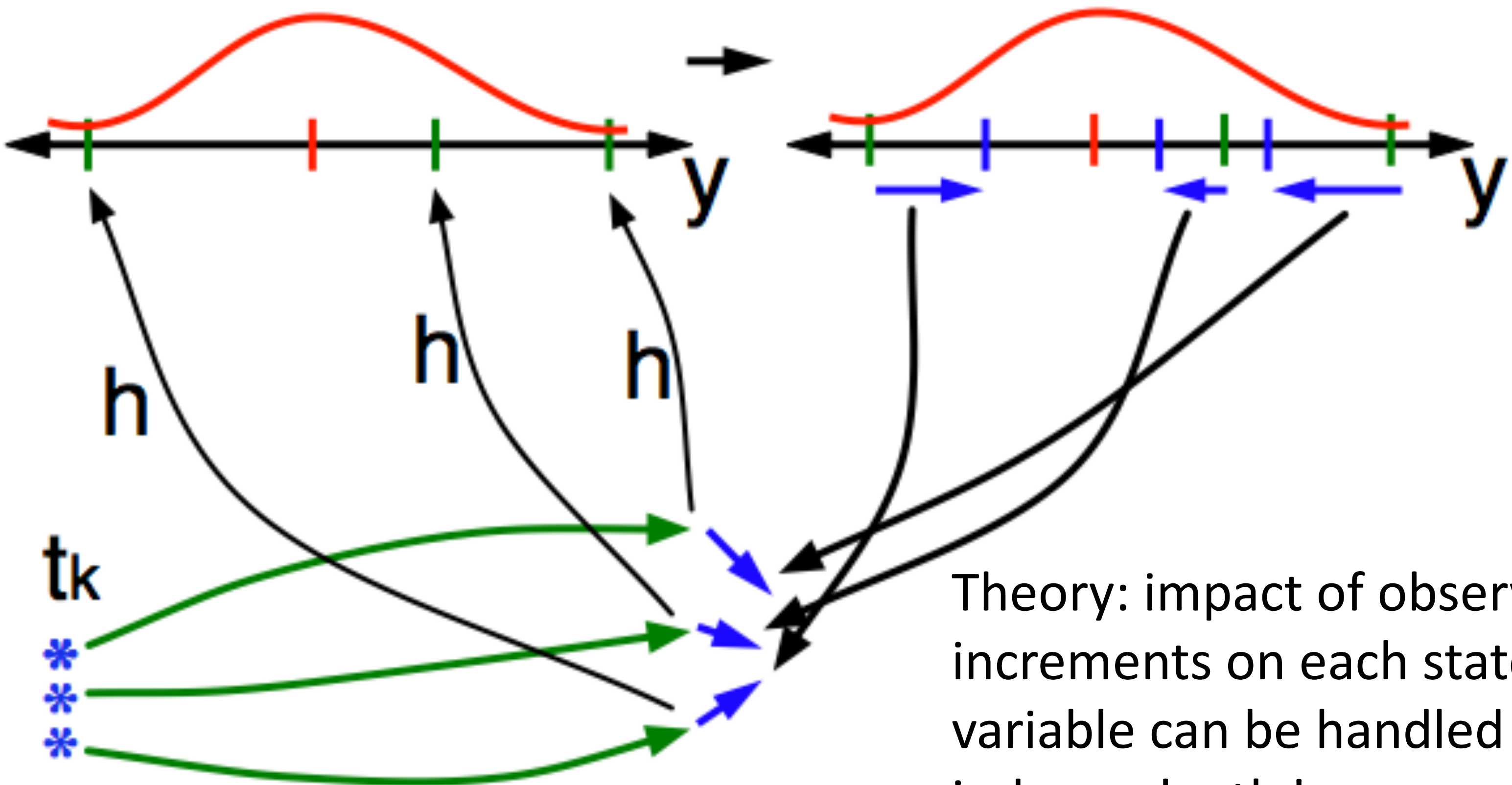


How an Ensemble Filter Works for Geophysical Data Assimilation

5. Use ensemble samples of y and each state variable to linearly regress observation increments onto state variable increments.

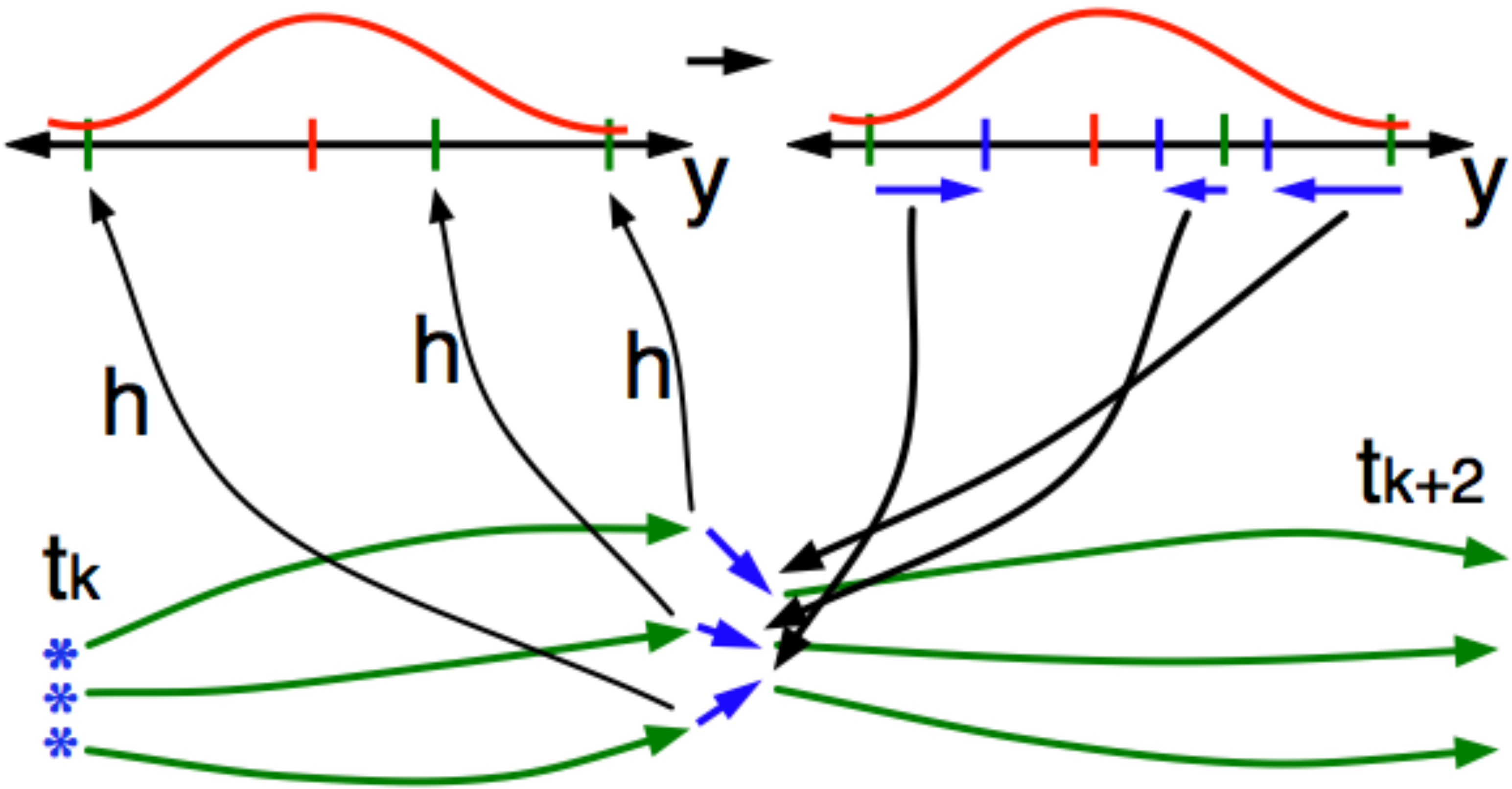


5. Use ensemble samples of y and each state variable to linearly regress observation increments onto state variable increments.



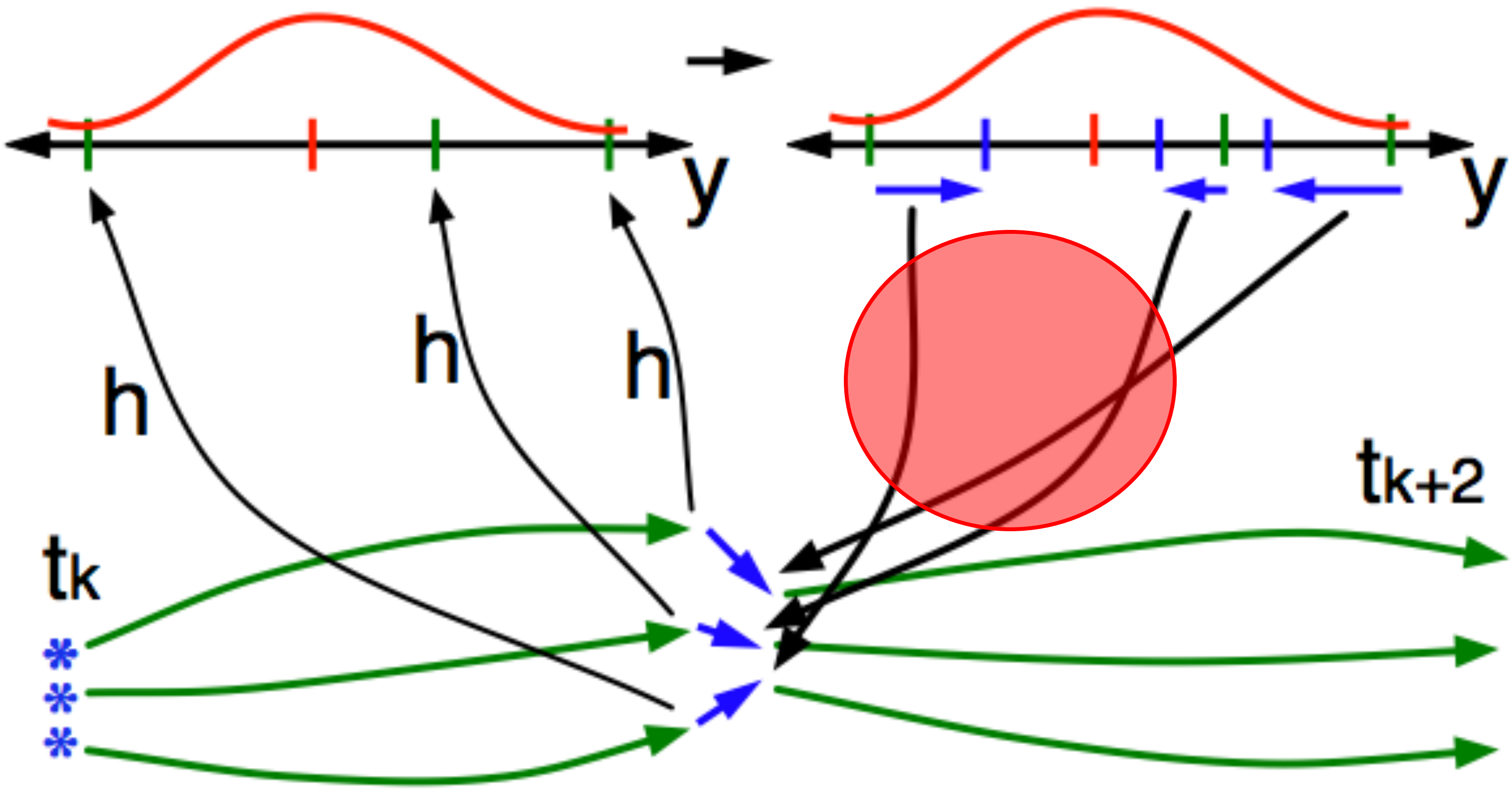
Theory: impact of observation increments on each state variable can be handled independently!

6. When all ensemble members for each state variable are updated, integrate to time of next observation …



How an Ensemble Filter Works for Geophysical Data Assimilation

For large models, regression of increments onto each state variable dominates time.

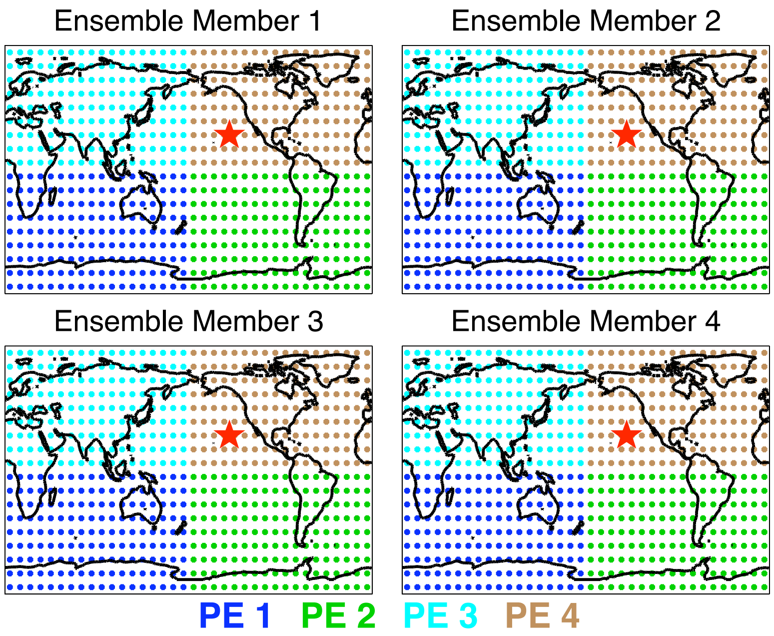


Parallelizing Implementation of the Regression

Data layout (option 1):
Each process stores all ensemble copies of subset of state.

Simple example:
    4 Ensemble members;
    4 PEs (colors).

Observation shown by red star.



Parallelizing Implementation of the Regression

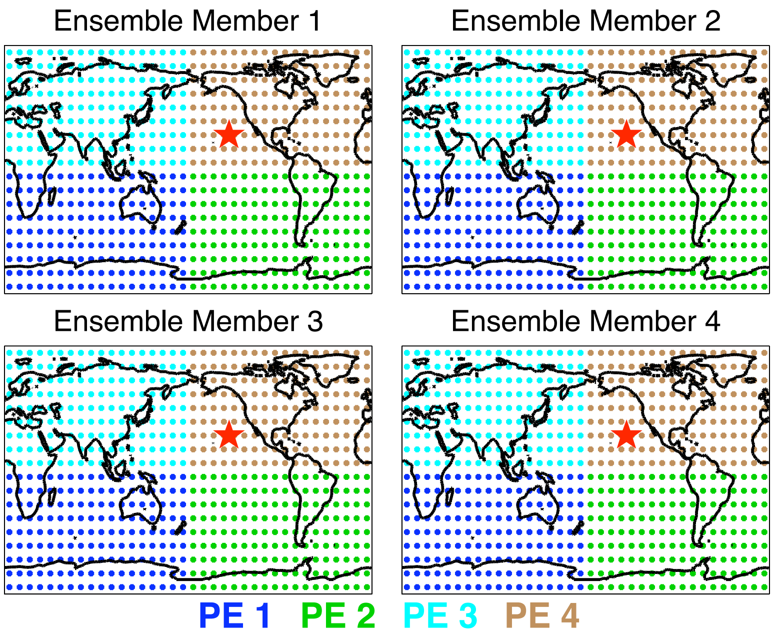
Data layout (option 1):
Each process stores all ensemble copies of subset of state.

One PE broadcasts obs. increments.

All ensemble members for each state variable are on one PE.

Can compute state mean, variance without communication.

All state increments computed in parallel.



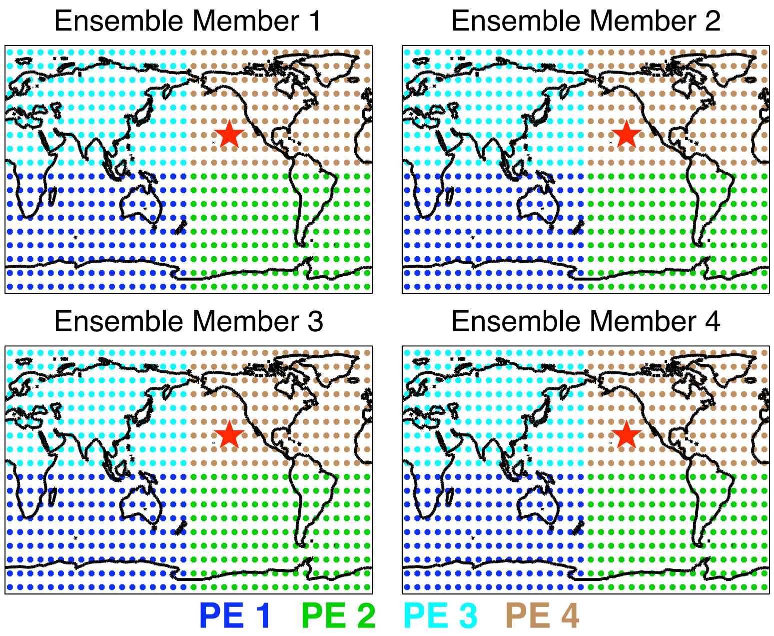
Computing Forward Operators

Data layout (option 1):
Each process stores all ensemble copies of subset of state.

Computing forward operator, h, is often local interpolation.

Most observations require no communication.

Those near boundaries or more complex operators require communication.



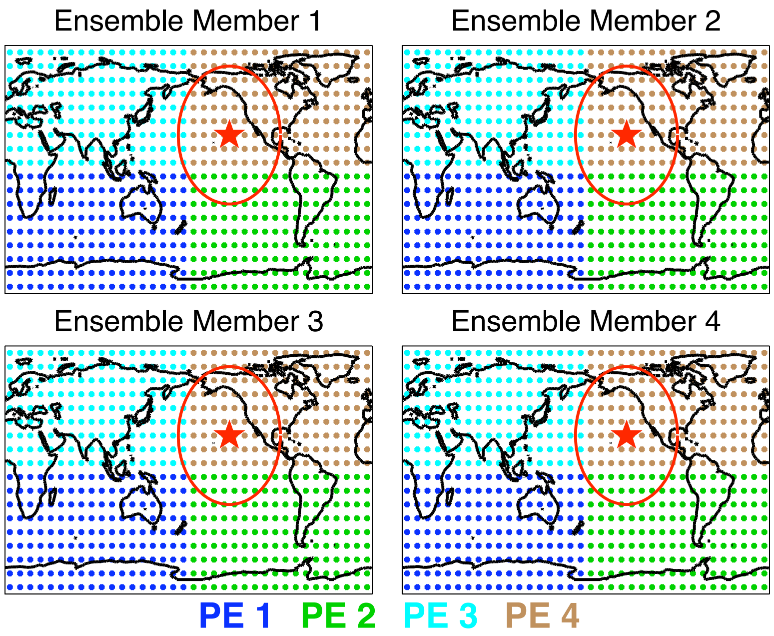
Load Balancing Issues for Regression with Localization

Data layout (option 1):
Each process stores all ensemble copies of subset of state.

Observation impact usually localized, reduces errors.

Observation in N. Pacific not expected to change Antarctic state.

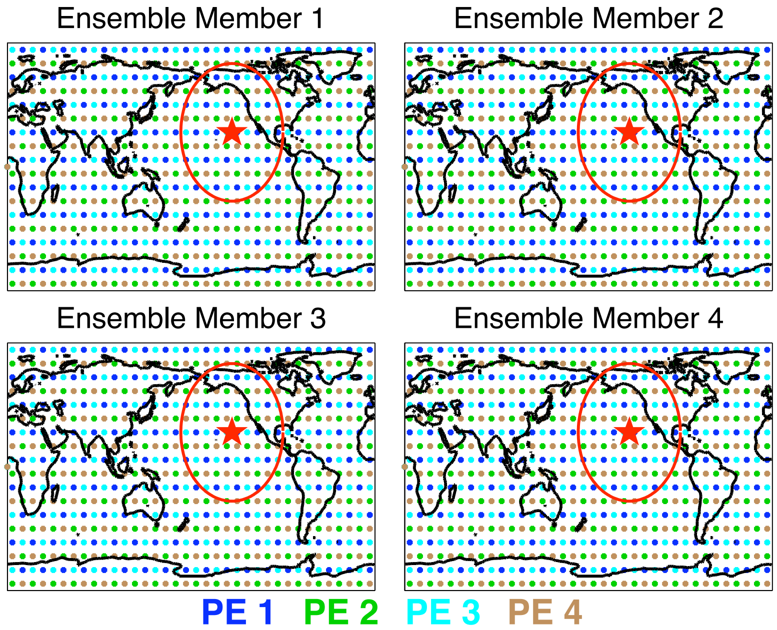
PE4 lots of work, PE1 has none.



Load Balancing Issues for Regression with Localization

Data layout (option 2):
Each process stores all ensemble copies of subset of state.

Can balance load by 'randomly' assigning state variables to PEs.

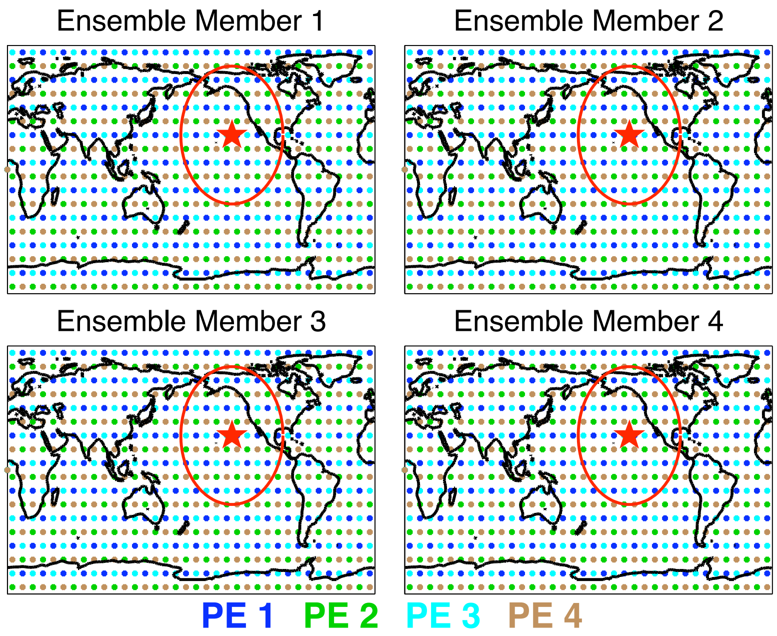


Load Balancing Issues for Regression with Localization

Data layout (option 2):
Each process stores all ensemble copies of subset of state.

Can balance load by 'randomly' assigning state variables to PEs.

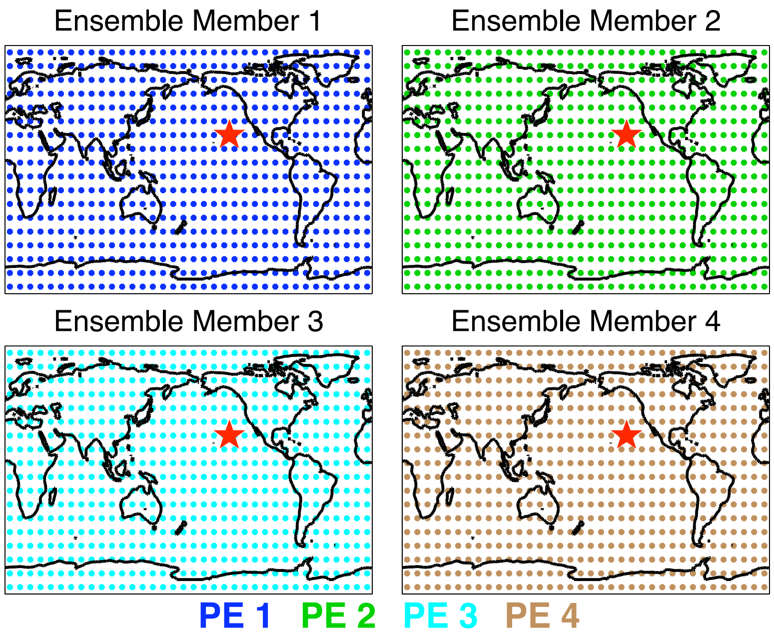
Now computing forward operators, h, requires communication.



Eliminating Communication for Forward Operators

Data layout (option 3):
Entire state for each ensemble on single PE.

If each PE has a complete ensemble, forward operators require no communication.

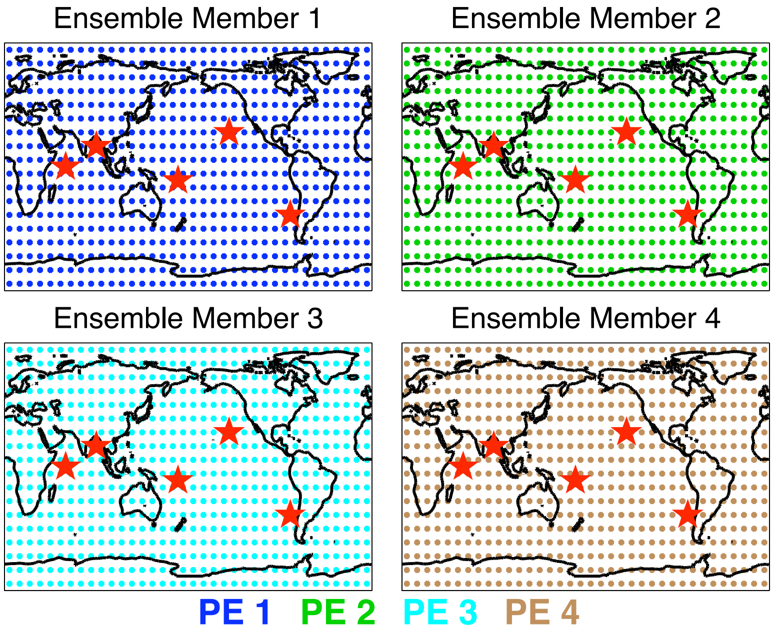


Eliminating Communication for Forward Operators

Data layout (option 3):
Entire state for each ensemble on single PE.

If each PE has a complete ensemble, forward operators require no communication.

Many forward operators could be done at once.



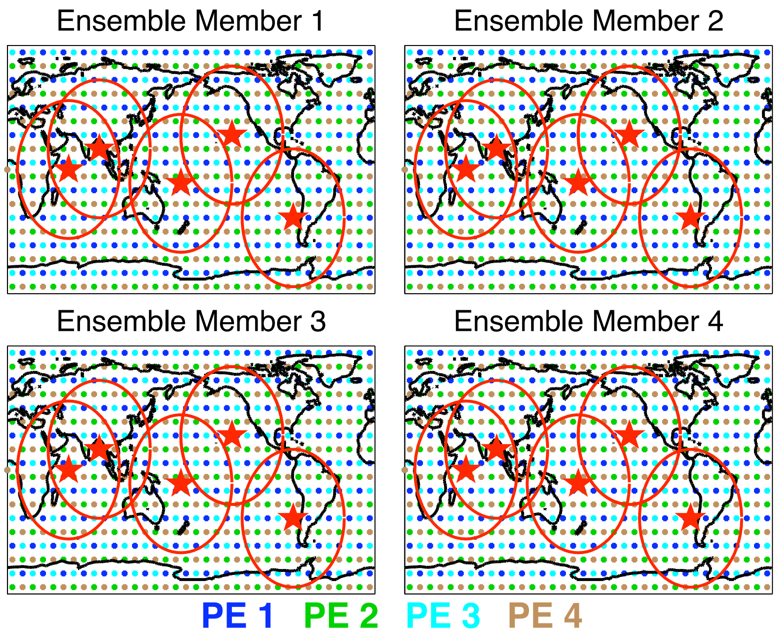
Best of Both Worlds? Using a Data Transpose.

Two Data layouts:

Option 2 for regression, Option 3 for forward operators

Do a data transpose between options 3 and 2, using all to all communication.

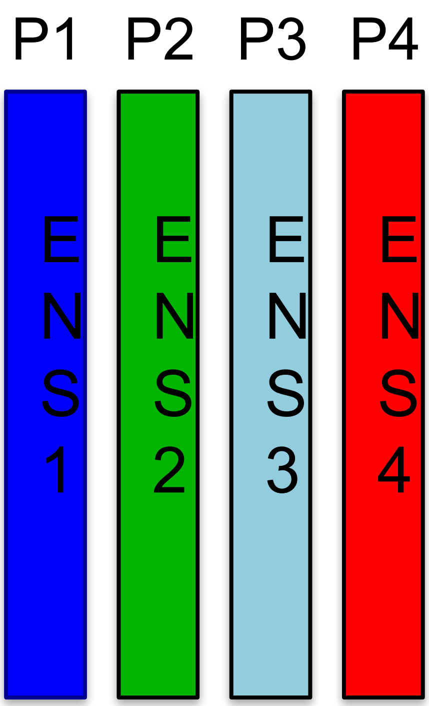
Then do state increments for each observation sequentially.



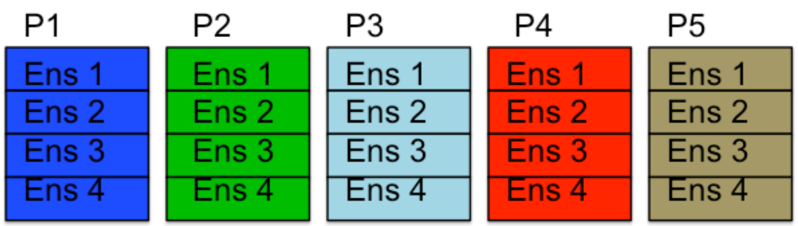
Best of Both Worlds? Using a Data Transpose.

Two Data layouts:

Option 2 for regression, Option 3 for forward operators



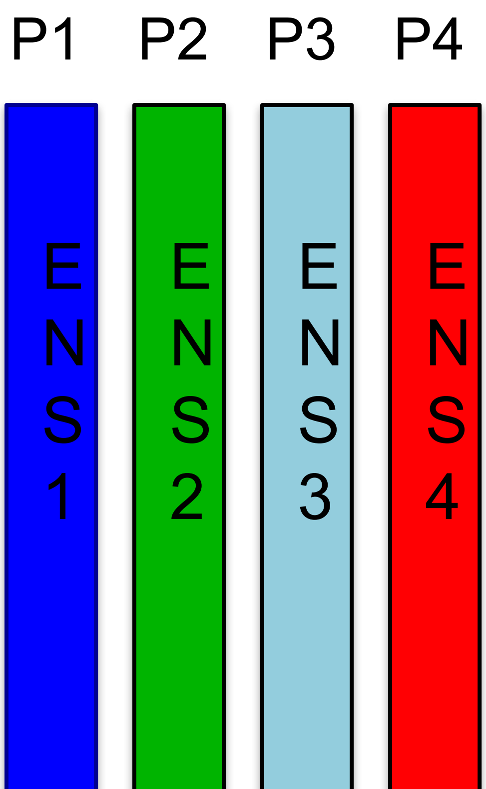
Whole model state available to a single processor.



All copies of some variables available to a single processor

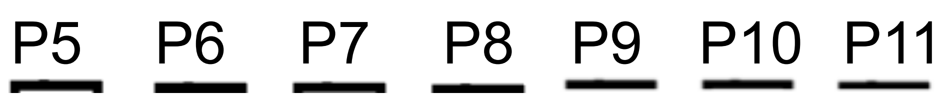
Problems with Using a Data Transpose.

1. Lots of communication, have to move all the data.
2. Not memory scalable, whole state must fit on a PE.
3. Load balancing for forward operators.



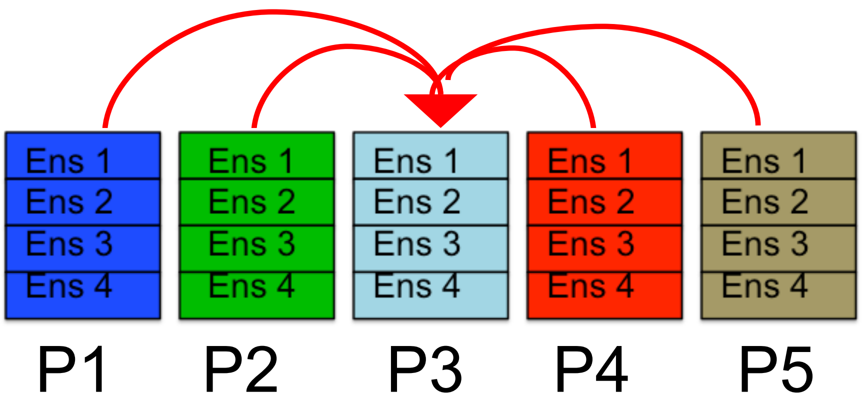
Ensemble size 4 example.
4 tasks have a whole copy of the model state.

Other tasks have no data and nothing to do during forward operators.



Avoiding the Transpose: Forward Operators with Distributed State

Use MPI2 **one sided communication** to grab state elements for forward operators.



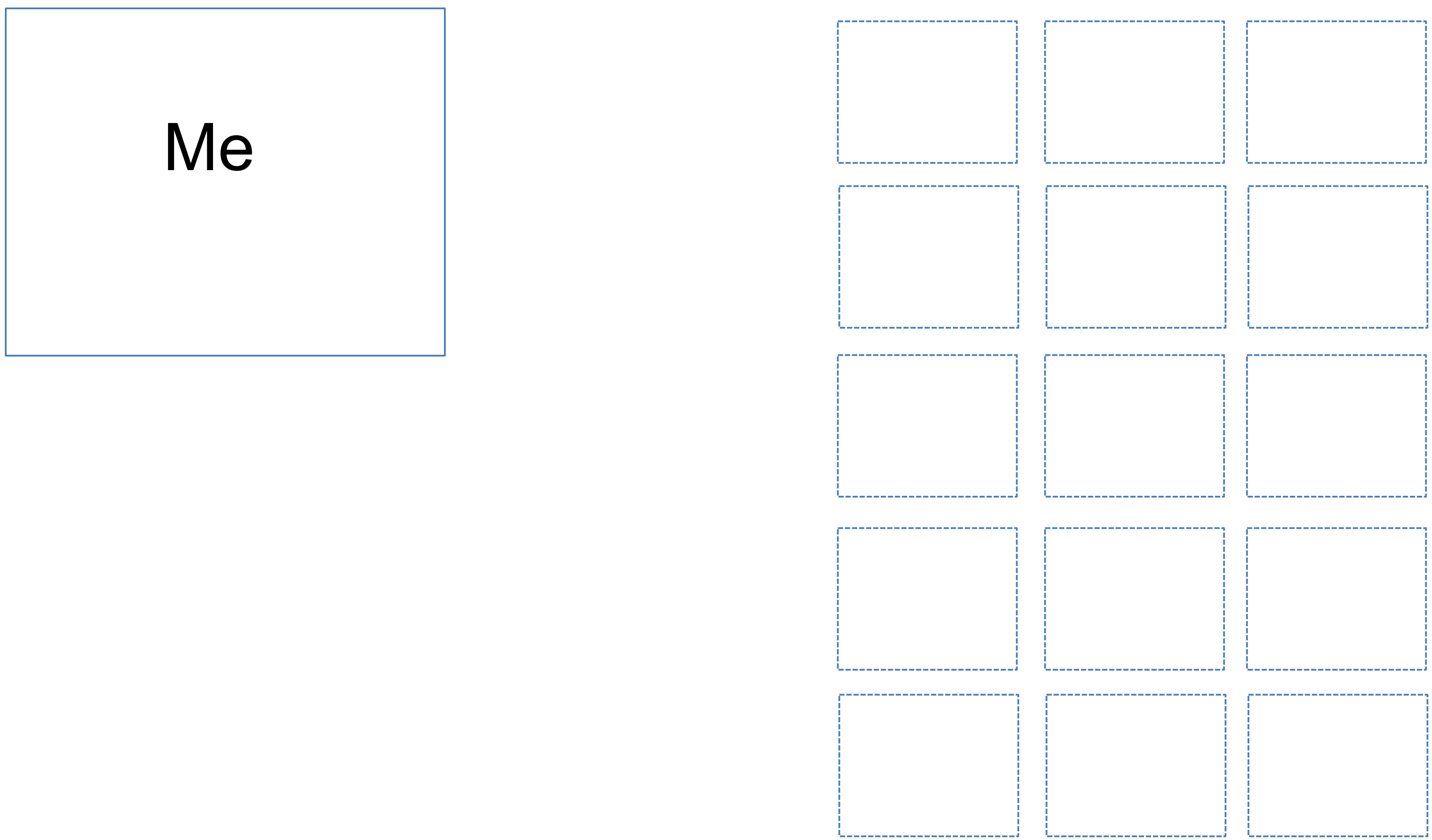
Reduces data movement.

Removes hard memory limit.

Allows Vectorization of forward operator calculations.

MPI2 One Sided Communication

Have my process and a set of other processes.



MPI2 One Sided Communication

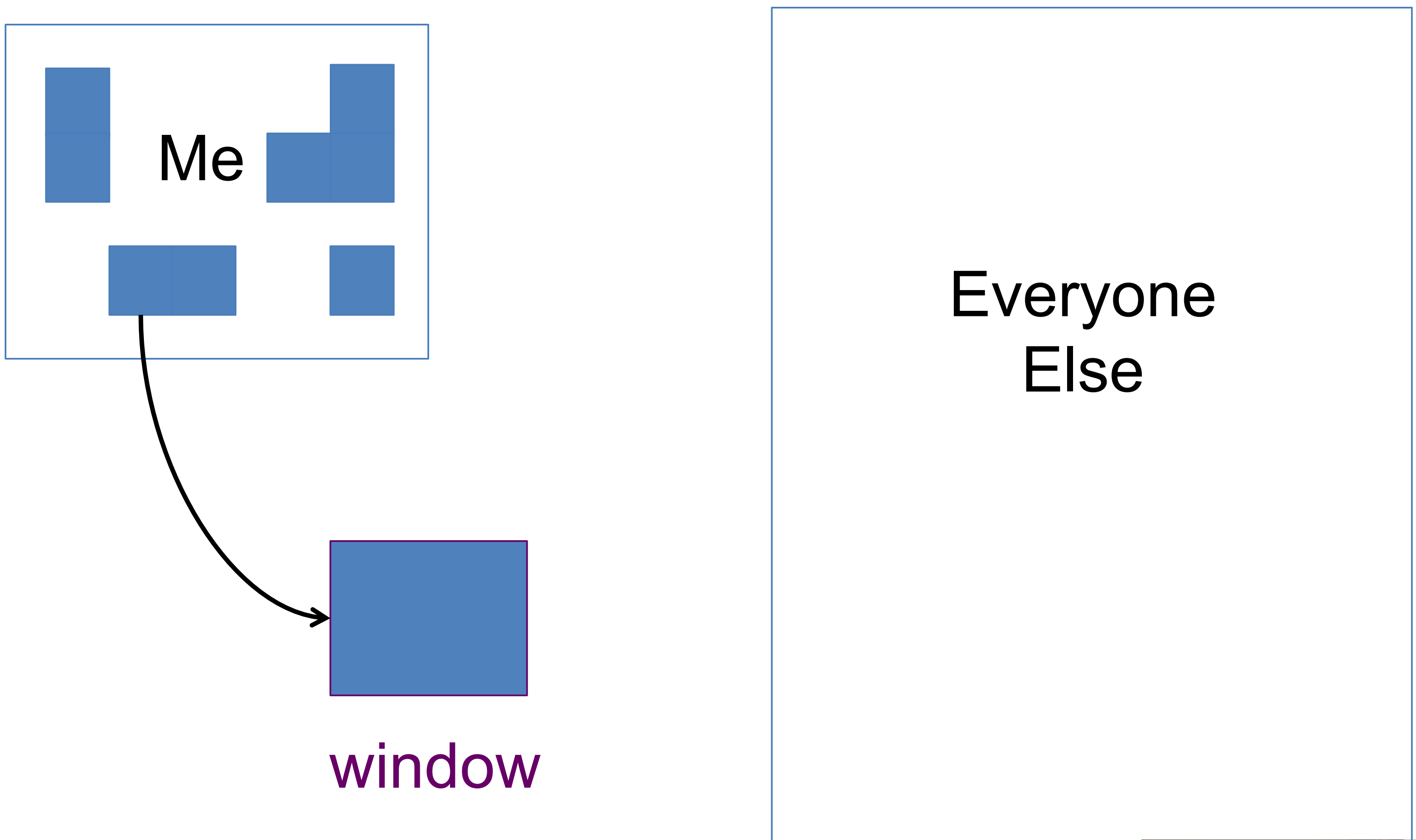
Have my process and a set of other processes.

Me

Everyone Else

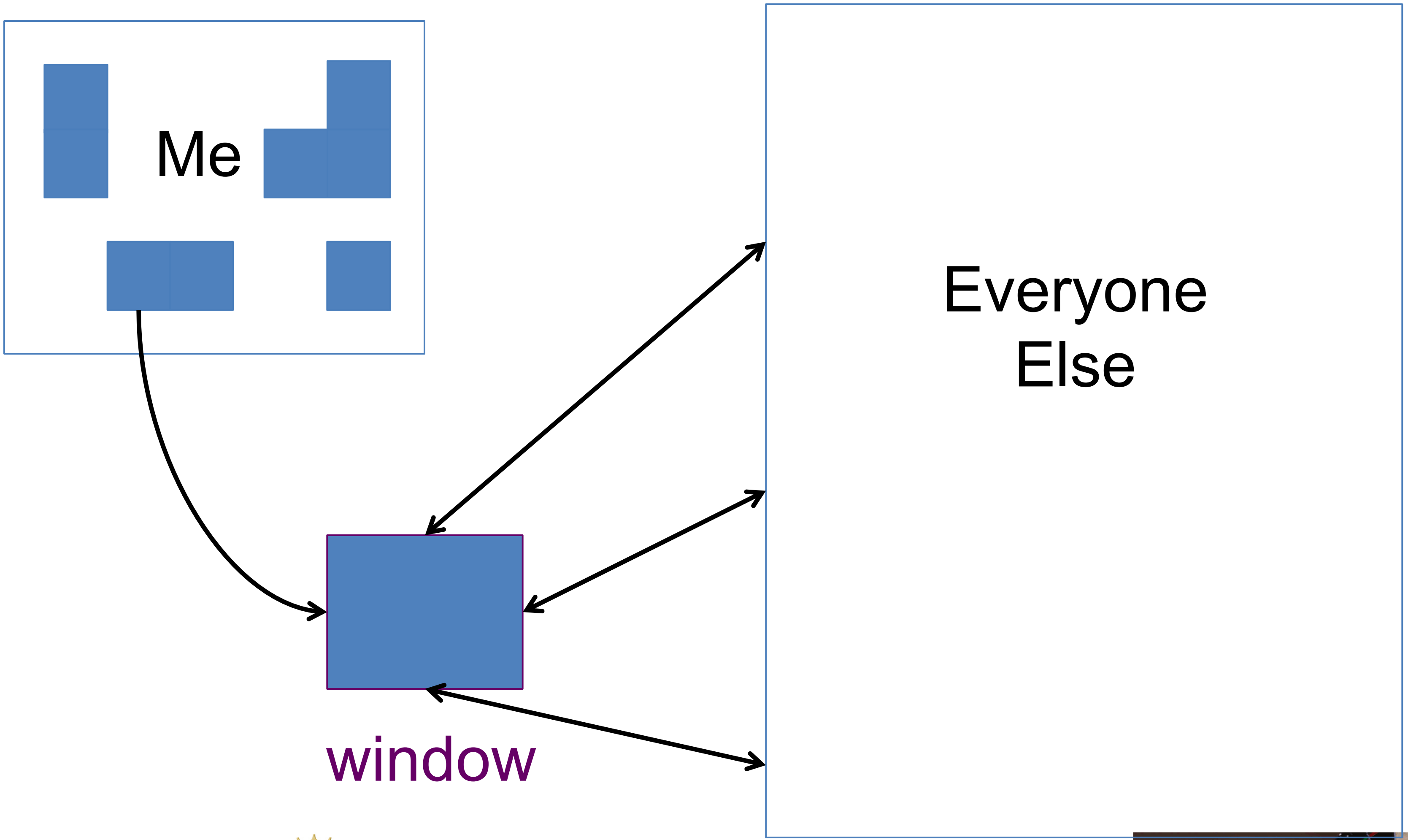
MPI2 One Sided Communication

Can place any of my data in a virtual window.



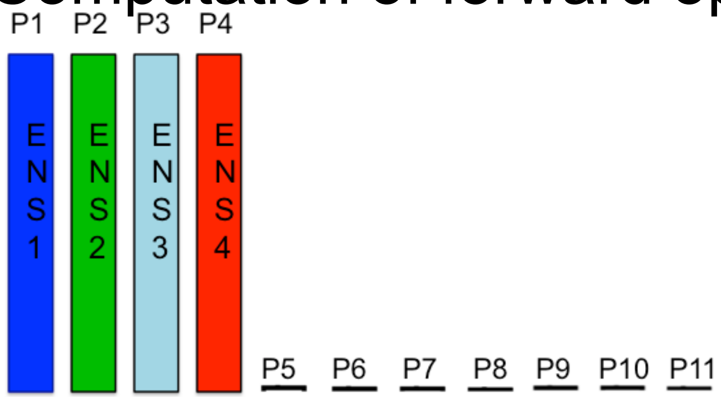
MPI2 One Sided Communication

Any other task can asynchronously grab data in 'window'.

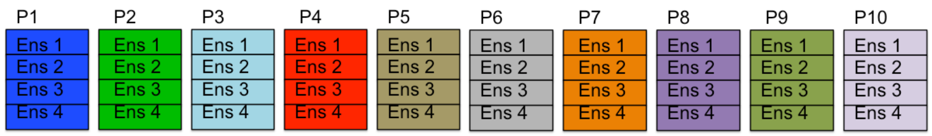


MPI2 One Sided Communication

Memory scales for forward operators; allows large models.
Computation of forward operators also scales and balances.



Old: 4 tasks doing all observations for 1 copy.



New: Lots of tasks doing some observations for all copies. Vectorizes, too.

WRF (regional weather forecast model) Results

Example problem specification:

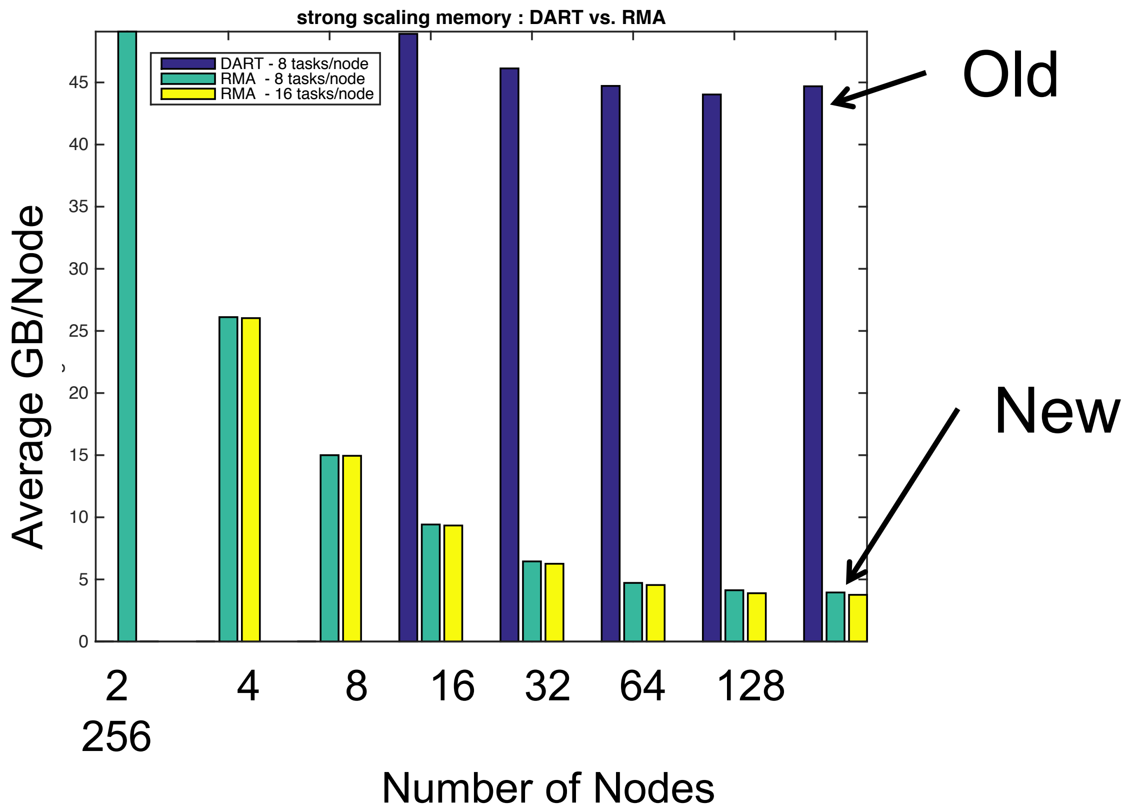
- 184 million model state variables
  - 1.5 GB per ensemble member
- 50 Ensemble members
- O(100,000) observations

WRF (regional weather forecast model) Results

Hardeware specification:

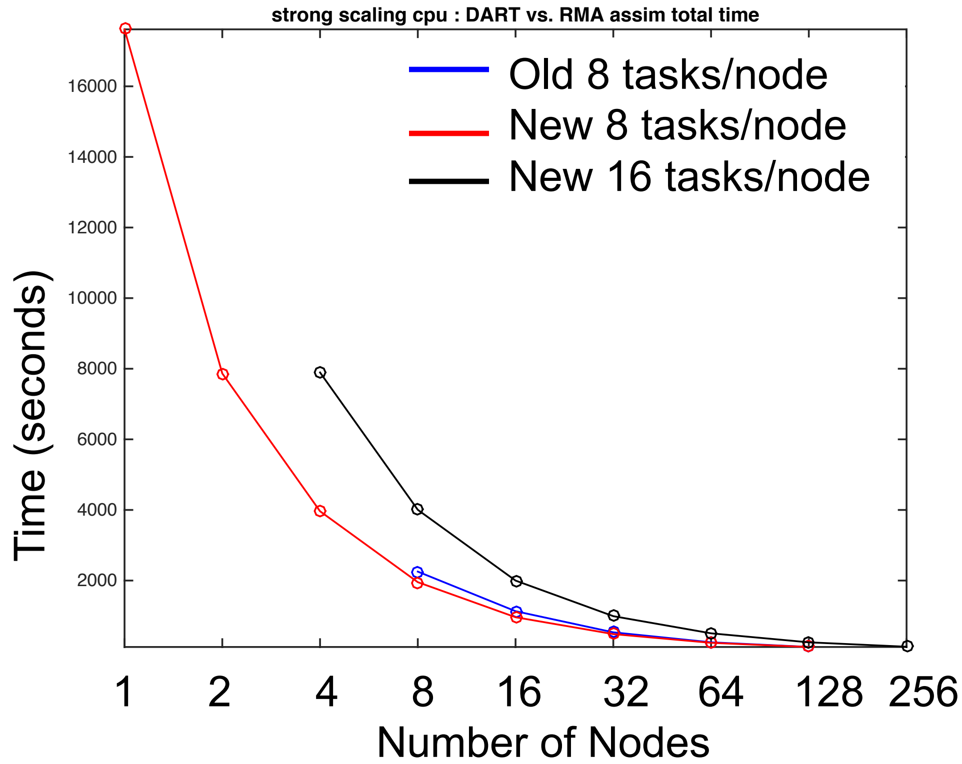
- NCAR's Yellowstone:
  - Intel Sandybridge
  - 16 cores per node
  - 25 GB usable memory per node

WRF Results: Memory Scaling



Original DART 8 tasks/node max.
New (RMA) version memory scales far better.

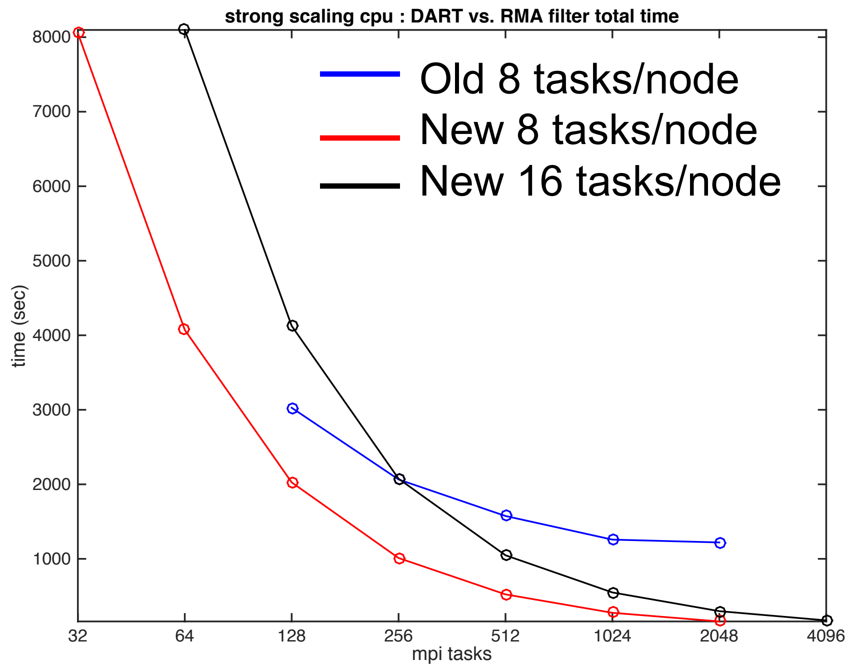
WRF Results: Computational Scaling for Assimilation



Very similar with 8 tasks per node.

New with 16 tasks/node slightly slower (memory overhead)

WRF Results: Bonus, I/O Scaling Improves



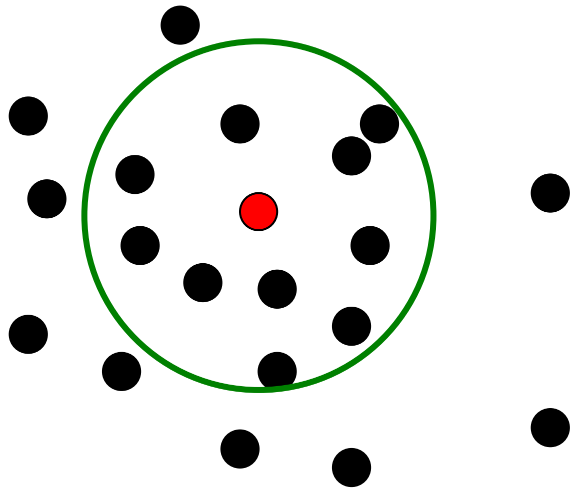
Total time scales much better for new (RMA).
Almost all due to writing separate output from each node.
Not gathering and doing single write.

Making Effective Use of Coprocessors

Focus on Specific Routines with Favorable Characteristics:

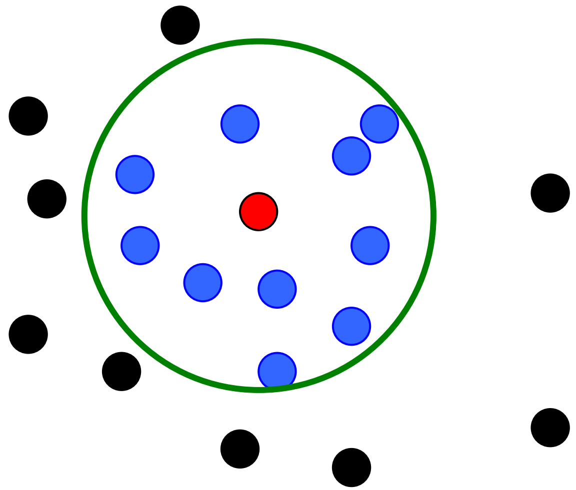
- High number of floating point instructions,
- Reasonably high floating point instructions per load/store,
- Isolated code – each process works on its own local data,
    - Can develop on one node, but apply to multinode runs.

Example: subroutine get_close



For a given observation computes:

Example: subroutine get_close



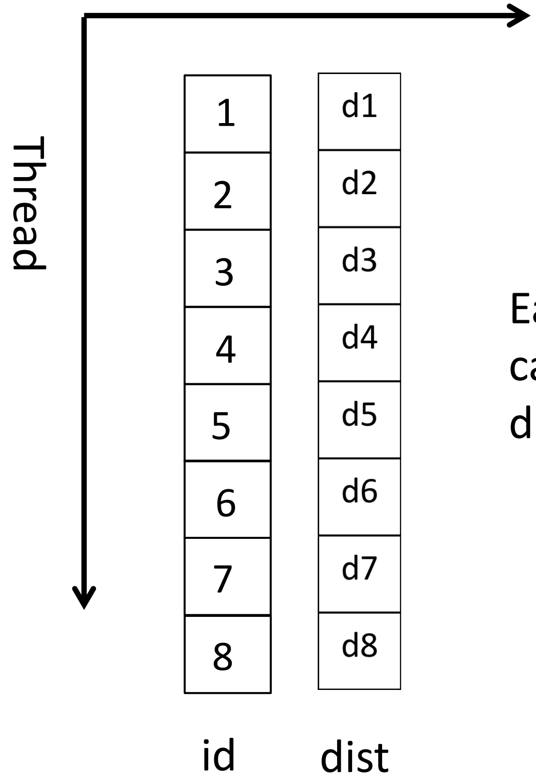
For a given observation computes:

- Number of state variables (or obs) within the localization radius,
- Distances to close state variables,
- Indices of the close states.

Making Effective Use of Coprocessors

GPU Algorithm for get_close:

Implemented in CUDA Fortran for NVIDIA GPUS by Ye Feng

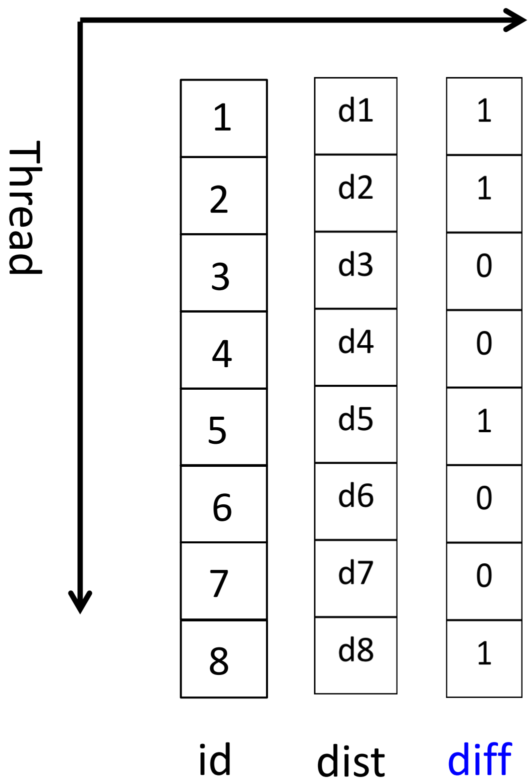


Each thread calculates a distance

Key idea for GPU implementation reduce branching.

Making Effective Use of Coprocessors

GPU Algorithm for get_close:



Most Significant Bit of (dist – cutoff) $\begin{cases} 1, \text{ dist<cutoff (close)} \\ 0, \text{ dist>cutoff (not close)} \end{cases}$

Key idea for GPU implementation reduce branching

Making Effective Use of Coprocessors

GPU Algorithm for get_close:

id	dist	diff	sum
1	d1	1	1
2	d2	1	2
3	d3	0	2
4	d4	0	2
5	d5	1	3
6	d6	0	3
7	d7	0	3
8	d8	1	4

Prefix Sum of diff

Last element gives number of close obs

Key idea for GPU implementation reduce branching

Making Effective Use of Coprocessors

GPU Algorithm for get_close:

Thread

id	dist	diff	sum	diff sum
1	d1	1	1	1
2	d2	1	2	2
3	d3	0	2	0
4	d4	0	2	0
5	d5	1	3	3
6	d6	0	3	0
7	d7	0	3	0
8	d8	1	4	4

diff x sum

Key idea for GPU implementation reduce branching

Making Effective Use of Coprocessors

GPU Algorithm for get_close:

Thread

id	dist	diff	sum	diff sum	close dist
1	d1	1	1	1	d1
2	d2	1	2	2	d2
3	d3	0	2	0	0
4	d4	0	2	0	0
5	d5	1	3	3	d5
6	d6	0	3	0	0
7	d7	0	3	0	0
8	d8	1	4	4	d8

diff x dist

Key idea for GPU implementation reduce branching

Making Effective Use of Coprocessors

GPU Algorithm for get_close:

Thread

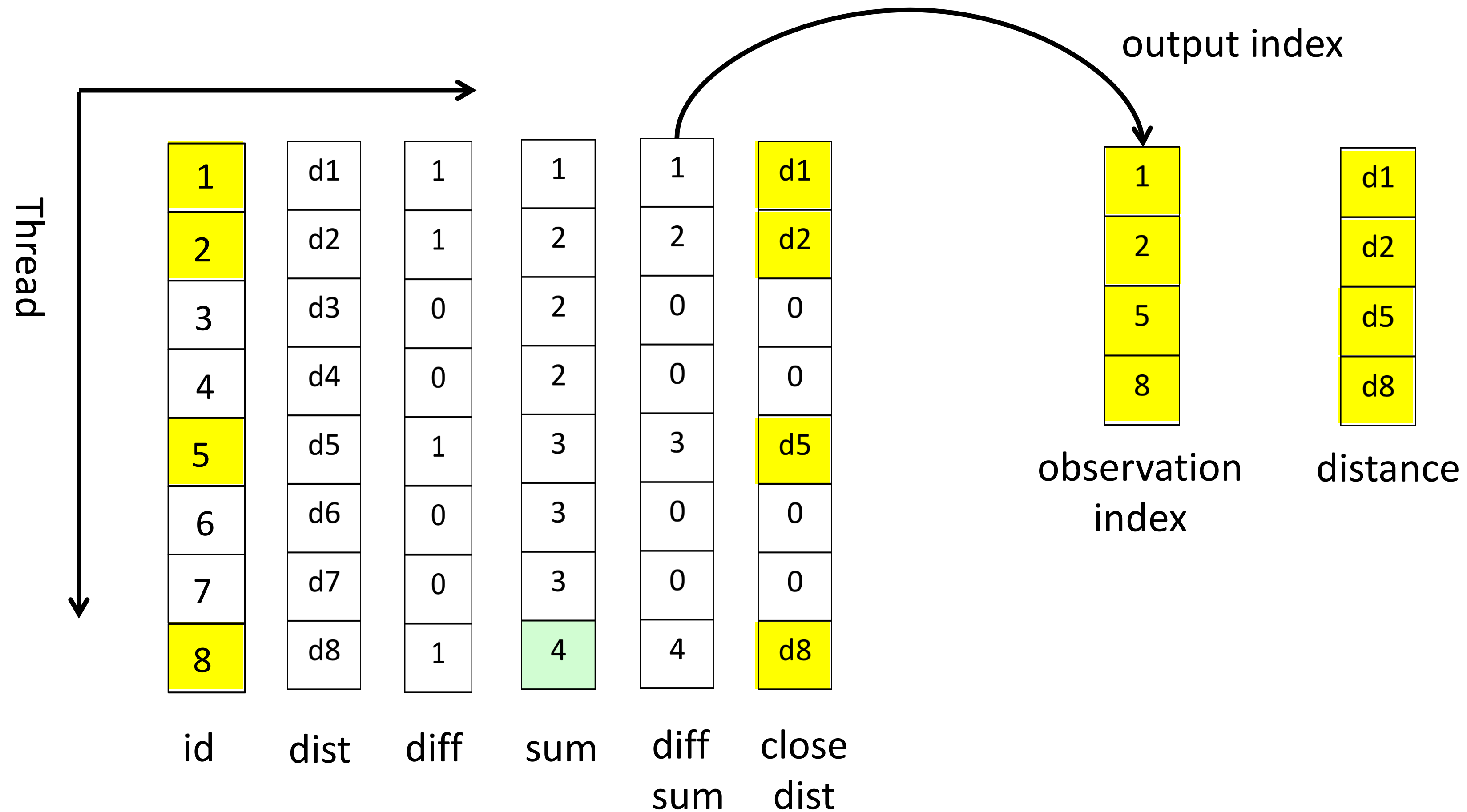
id	dist	diff	sum	diff sum	close dist
1	d1	1	1	1	d1
2	d2	1	2	2	d2
3	d3	0	2	0	0
4	d4	0	2	0	0
5	d5	1	3	3	d5
6	d6	0	3	0	0
7	d7	0	3	0	0
8	d8	1	4	4	d8

observation index	distance
1	d1
2	d2
5	d5
8	d8

Key idea for GPU implementation reduce branching

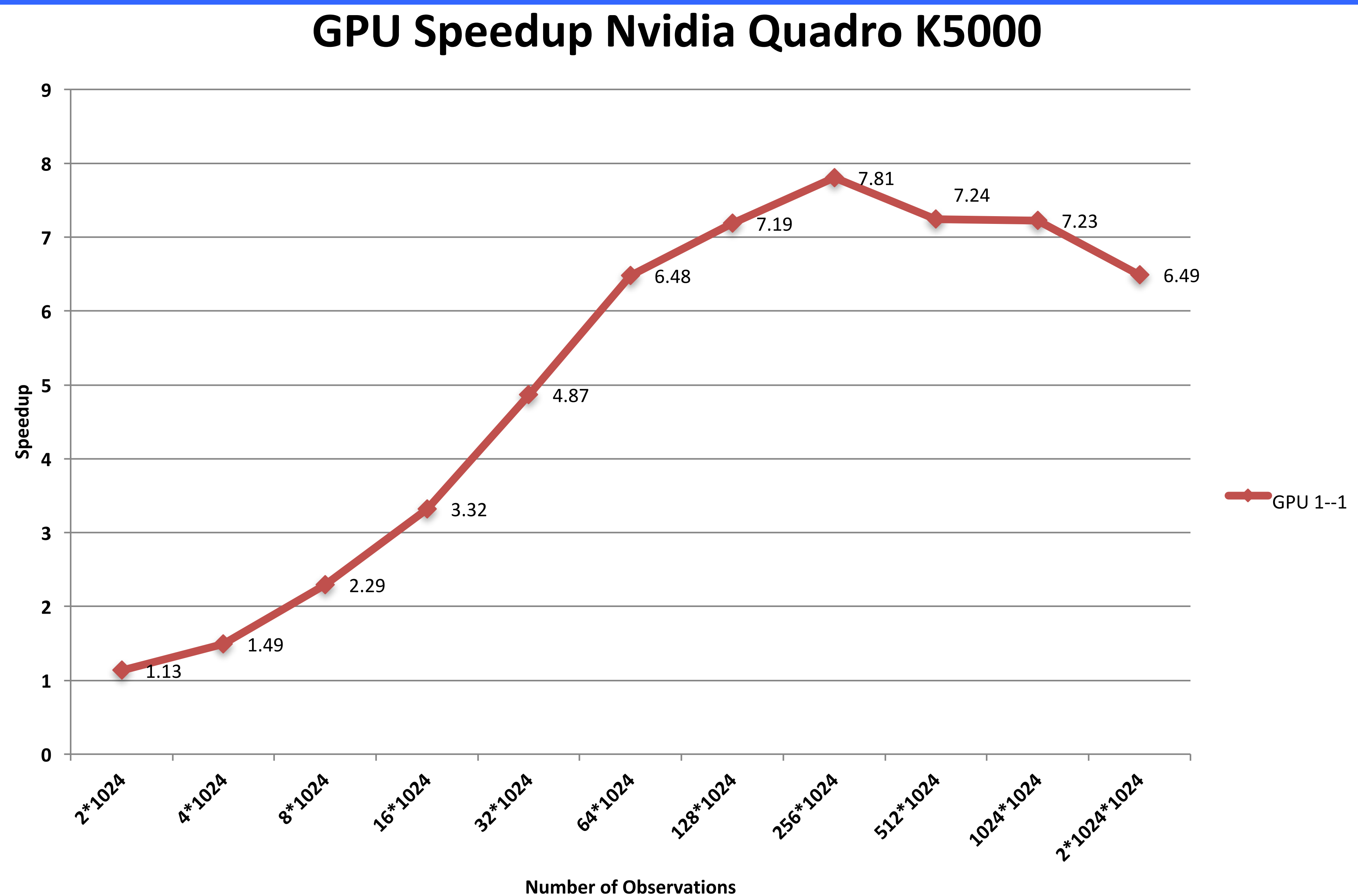
Making Effective Use of Coprocessors

GPU Algorithm for get_close:



Key idea for GPU implementation reduce branching

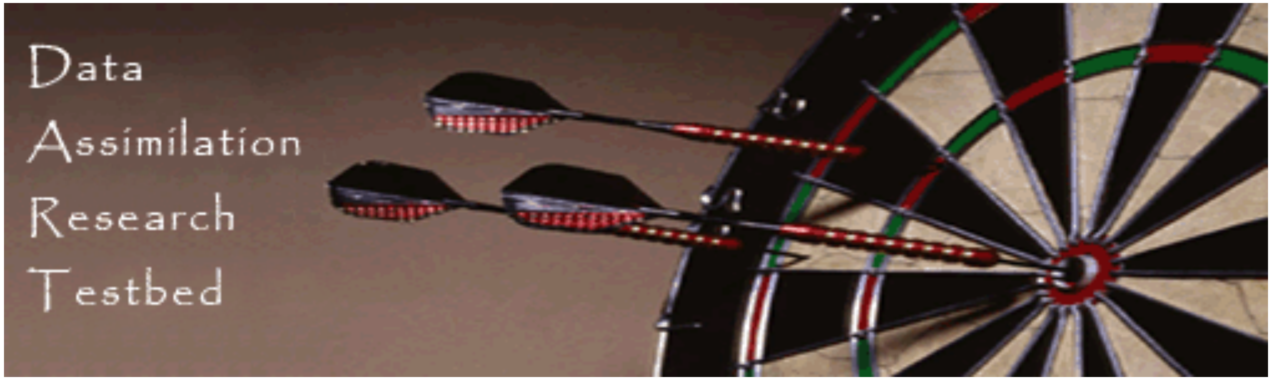
Making Effective Use of Coprocessors



Conclusions

- General purpose ensemble filters can scale well to many processes.
- Large geophysical problems can scale easily to O(10000) processes.
- General purpose facility must support flexible data distribution.
- IO is fast becoming the biggest bottleneck.
- Efficient use of coprocessors may be possible.
- A parallel implementation simulation facility is useful.

Learn more about DART at:



www.image.ucar.edu/DAReS/DART

dart@ucar.edu

Anderson, J., Hoar, T., Raeder, K., Liu, H., Collins, N., Torn, R., Arellano, A., 2009: *The Data Assimilation Research Testbed: A community facility.* BAMS, **90**, 1283—1296, doi: 10.1175/2009BAMS2618.1