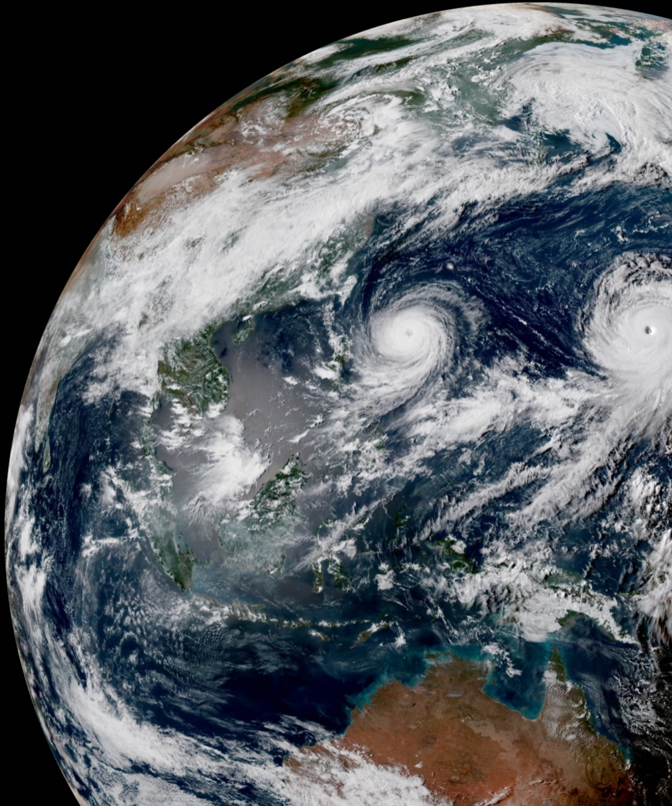
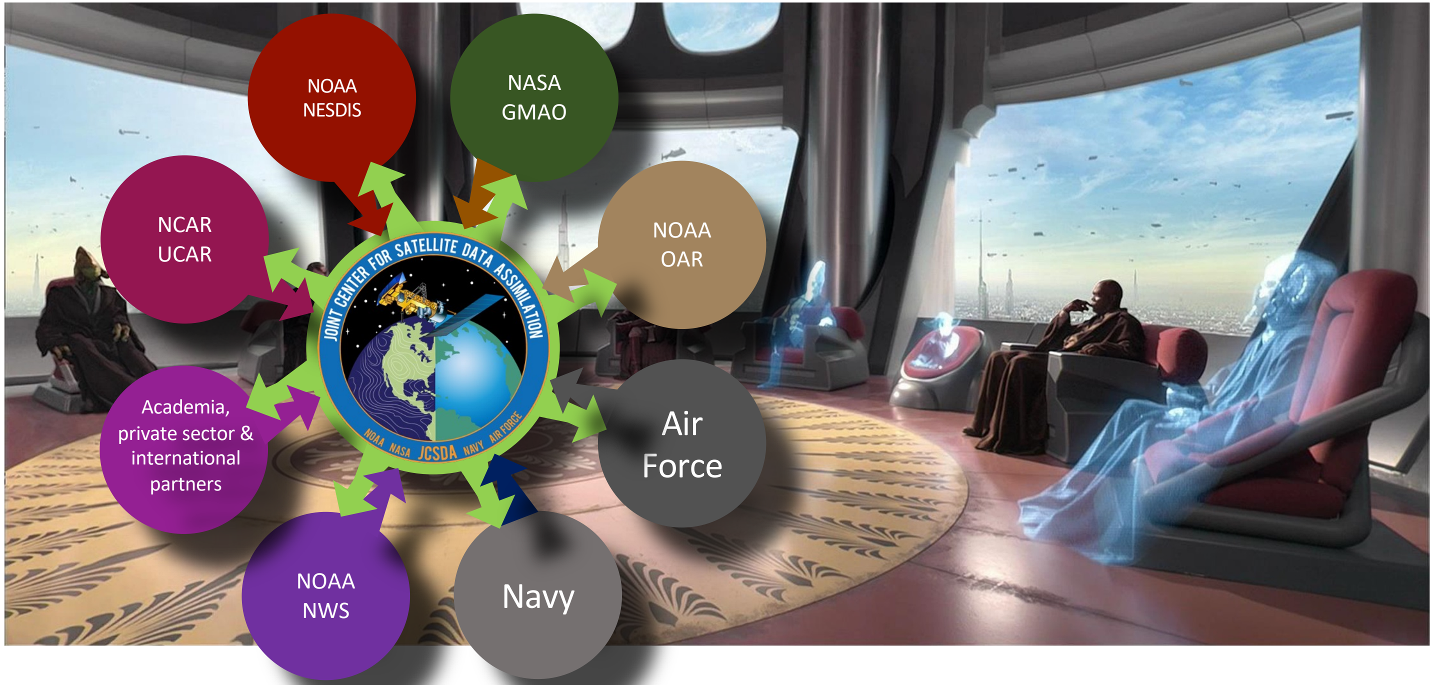
Introduction to JEDI



MPAS-JEDI Tutorial St. Andrews, UK June 2025
Christian Sampson

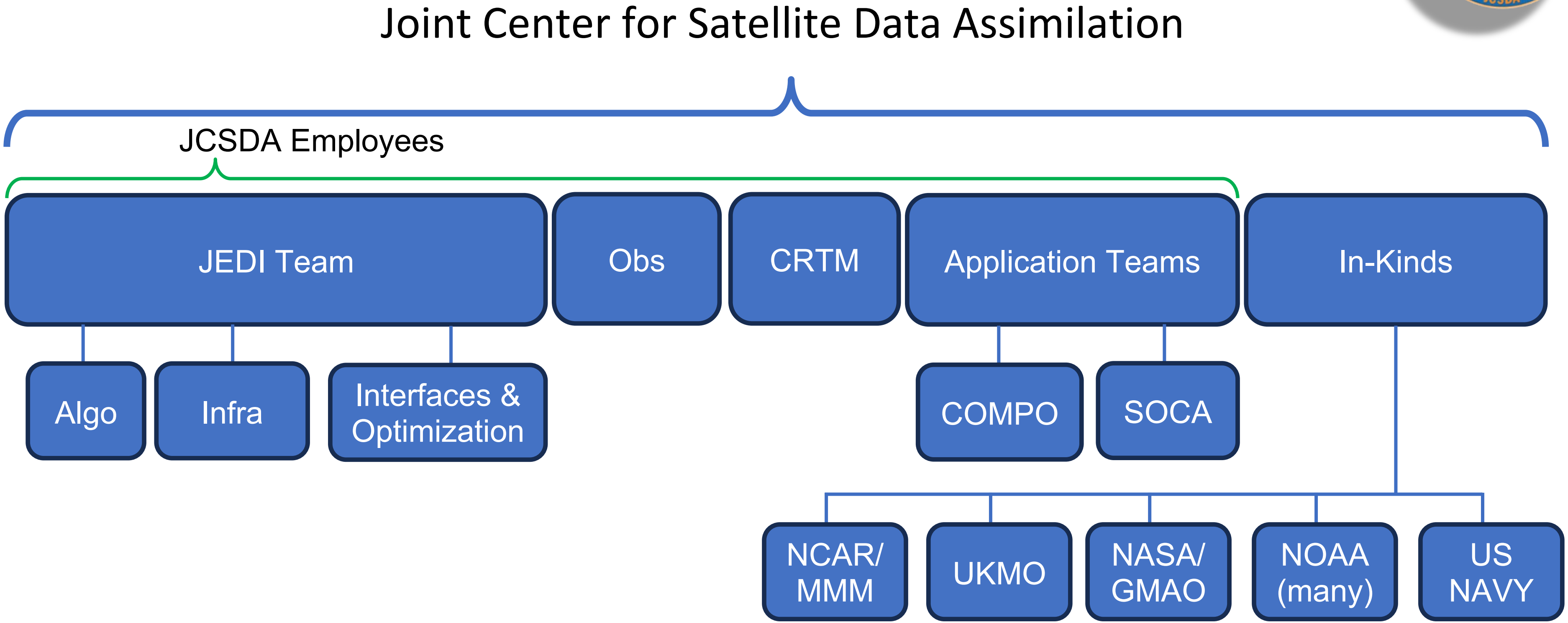
Contributors: Nate Crossette, Fabio Diniz

The JCSDA



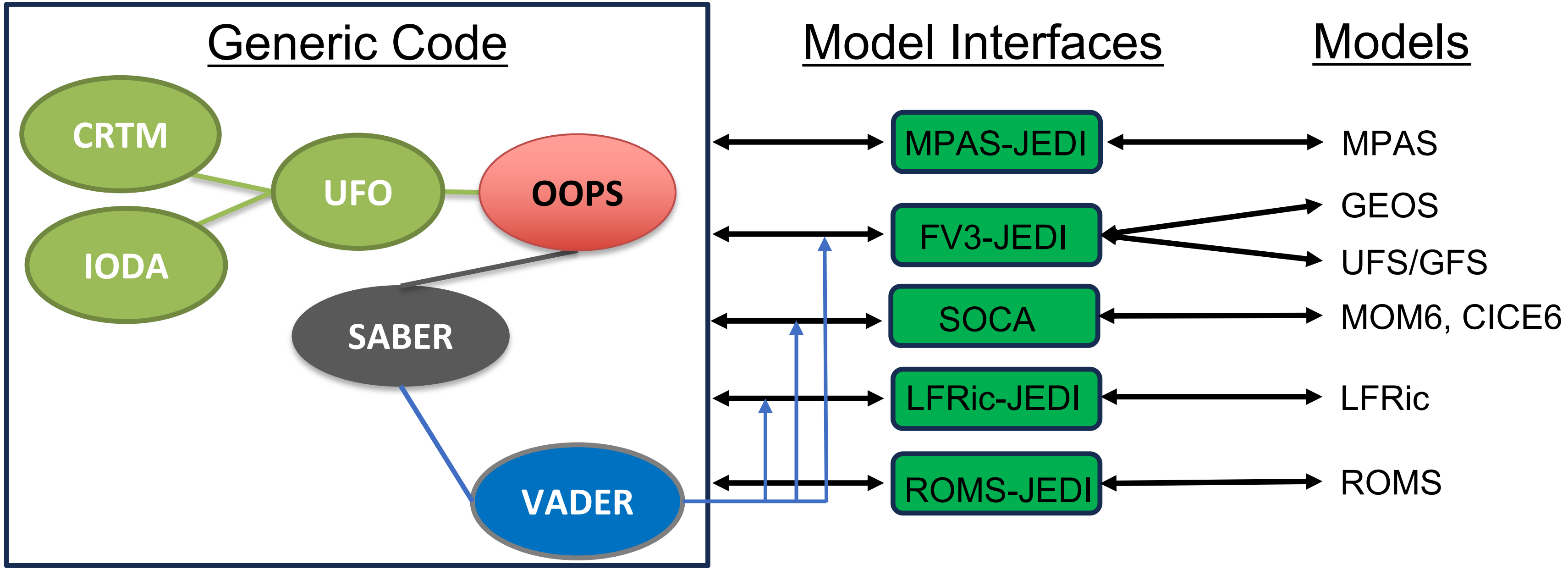
NOAA NESDIS
NASA GMAO
NCAR UCAR
NOAA OAR
JOINT CENTER FOR SATELLITE DATA ASSIMILATION
NOAA NASA JCSDA NAVY AIR FORCE
Academia, private sector & international partners
Air Force
NOAA NWS
Navy

The People of the JCSDA



Joint Effort for Data assimilation Integration

- Generic (model-agnostic), unified data assimilation framework for **Research AND Operations**
  - JEDI is run with toy models (Lorenz95/QG) up to Earth system coupled models



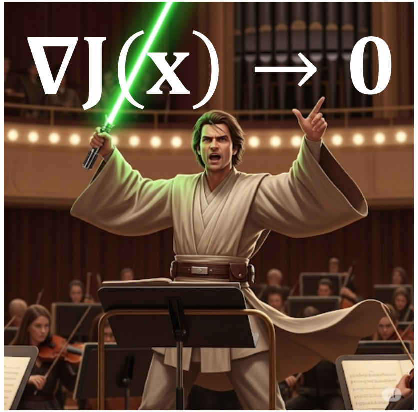
JEDI Components

- OOPS: Object Oriented Prediction System
  - Generic data assimilation algorithms
- VADER: VAriable DErivation Repository
  - Generic variable changes
- SABER: System Agnostic Background Error Representation
  - Generic background error covariance models (B-matrix)
- UFO: Unified Forward Operator
  - Collection of model-agnostic observation operators
- CRTM: Community Radiative Transfer Model
  - Simulation of satellite radiances
- IODA: Interface for Observation Data Access
  - Performs all the I/O for the observations

JEDI is 80-90% C++17 except:
- CRTM ~90% FORTRAN
- UFO ~30% FORTRAN

JEDI Components

OOPS



SABER

UFO, CRTM

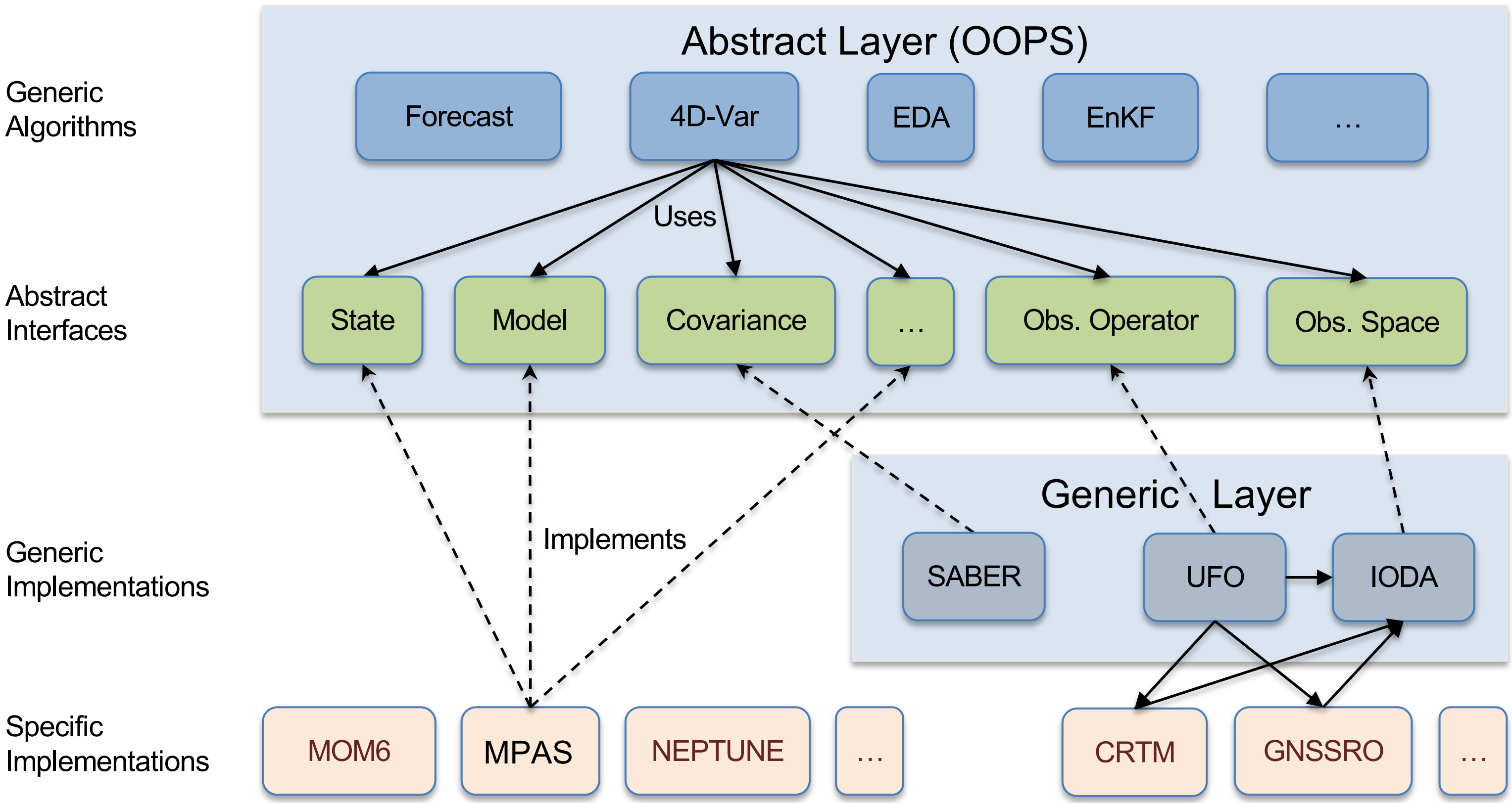
$$J(x) = \frac{1}{2}\left(x - x^b\right)^T B^{-1}\left(x - x^b\right) + \frac{1}{2}\left(y^0 - Hx\right)^T R^{-1}\left(y^o - Hx\right)$$

VADER

(If model variables differ from analysis variables)

IODA

JEDI - Abstraction and Genericity



Model-agnostic Abstract Interface

Model-agnostic components implementing part of the OOPS interface.

How is JEDI code 'Generic'?

DA in one sentence:

Update a model **state** using new observations accounting for errors: obs, background, (model).

JEDI uses the `atlas::FieldSet` as a generic data structure to hold state information (and `atlas::FunctionSpace` for model geometry).

'Workhorse' object in generic JEDI algorithms

ndarray is to numpy; as atlas::FieldSet is to JEDI

https://github.com/ecmwf/atlas

https://sites.ecmwf.int/docs/atlas/

atlas

From ECMWF

A library for Numerical Weather Prediction and Climate Modelling

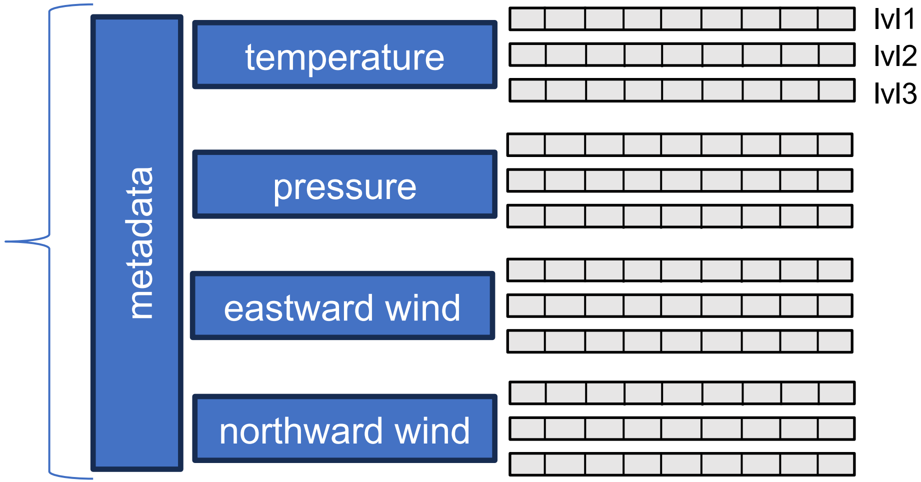
Atlas is an open source library providing grids, mesh generation, and parallel data structures targetting Numerical Weather Prediction or Climate Model developments.

How is JEDI Code ‘Generic’?

```
class FieldSet3D : public util::Serializable,
                   public util::Printable {
public:
  /// Constructors
  FieldSet3D(const util::DateTime &, const eckit::mpi::Comm &); //empty fieldSet
  FieldSet3D(const FieldSet3D &); //deep copy of other fieldSet

  /// Accessors
  const Variables & variables() const;
  const util::DateTime validTime() const {return validTime_;}
  const atlas::FieldSet & fieldSet() const {return fset_;}
  atlas::FieldSet & fieldSet() {return fset_;}

  /// Get individual fields from inside fieldSet
  atlas::Field & operator[](const int & fieldIndex) {return fset_[fieldIndex];}
  atlas::Field & operator[](const std::string & fieldName) {return fset_[fieldName];}
  atlas::Field & operator[](const Variable & var) {return fset_[var.name()];}

  /// Arithmetic operations
  void zero();
  FieldSet3D & operator+=(const FieldSet3D &);
  FieldSet3D & operator*=(const FieldSet3D &);
  FieldSet3D & operator*=(const double &);
  double dot_product_with(const FieldSet3D &, const Variables &) const;
  double norm(const Variables &) const;
  void sqrt();

  /// Utilities
  // iterators, add/remove fields, etc...

private:
  oops::Variables currentVariables() const;
  void print(std::ostream &) const override;

  atlas::FieldSet fset_;
  const util::DateTime validTime_;
  const eckit::mpi::Comm & comm_;
  std::string name_;
  mutable oops::Variables vars_;
};
```

JEDI wrapper for `atlas::FieldSet`

- Access variables/fieldnames
- Access individual field
- Perform operations



How is JEDI code 'Generic'?

```
template <typename MODEL>
class State : public util::Printable,
              public util::Serializable,
              private util::ObjectCounter<State<MODEL> > {
  typedef typename MODEL::State            State_;
  typedef oops::Geometry<MODEL>            Geometry_;

 public:
  /// Constructors/Destructor
  State(const Geometry_ & resol, const Variables & vars, const util::DateTime & time);
  ...
  /// Assignment operator
  State & operator =(const State &);
  /// Accessors
  State_ & state() {if (fset_) {fset_->clear();} return *state_;}
  const State_ & state() const {return *state_;}
  ...
  /// Accessor to variables associated with this State
  const Variables & variables() const;
  /// Class Methods
    ...

  ///  IMPORTANT METHODS ::
  void toFieldSet(atlas::FieldSet &) const;
  void fromFieldSet(const atlas::FieldSet &);

 private:
  std::unique_ptr<State_> state_;
  size_t ID_;
  void print(std::ostream &) const override;

 protected:
  mutable std::unique_ptr<FieldSet3D> fset_;
};
```

Model-facing classes in OOPS are class templates

Methods are implemented on generic concepts of `MODEL::state` and `MODEL::geometry`

Model-specific definitions are written in model interfaces and ***fill*** the class template

```
oops/scr/
 |- test/
 |- oops/
     |- assimilation/
-->  |- interface/
          State.h
          ...
     |- runs/
     ...
```

Example of Templated Function

```cpp
// Template function to create a vector from two numbers of the same type
template <typename T>
std::vector<T> makeVec(T num1, T num2) {
    std::vector<T> resultVector;
    resultVector.push_back(num1);
    resultVector.push_back(num2);
    return resultVector;
}
```

T is the type, could be int, double, float…

If in the main program these are called

```cpp
std::vector<int> v1=makeVec(int num1, int num2);
```

```cpp
std::vector<double> v2=makeVec(double num1, double num2);
```

The compiler knows to make these two functions for you

```cpp
std::vector<int> makeVec(int num1, int num2) {
    std::vector<int> resultVector;
    resultVector.push_back(num1);
    resultVector.push_back(num2);
    return resultVector;
}
```

```cpp
std::vector<double> makeVec(double num1, double num2) {
    std::vector<double> resultVector;
    resultVector.push_back(num1);
    resultVector.push_back(num2);
    return resultVector;
}
```

Example of a Templated Class

```cpp
// T will be the type of the vector's components (e.g., int, double, float)
template <typename T>
class vectorClass {
private:
    T m_x; // The x-component of the 2D vector
    T m_y; // The y-component of the 2D vector
public:
    // Constructor: This function takes two numbers and initializes
    // the x and y components of our 2D vector.
    vectorClass(T x_val, T y_val) : m_x(x_val), m_y(y_val) {
    }
    // Member function: Calculates the dot product of this vector
    T dotProduct(const vectorClass<T>& other) const {
        // The dot product of two 2D vectors (x1, y1) and (x2, y2)
        return (m_x * other.m_x) + (m_y * other.m_y);
    }
    //A public getter for the x-component
    T getX() const {
        return m_x;
    }
    //A public getter for the y-component
    T getY() const {
        return m_y;
    } };
```

Main Program

```cpp
int main() {

    // --- Example 1: Using vectorClass with integers ---
    // Create the first integer vector (3, 4)
    vectorClass<int> vec1_int(3, 4);
    // Create the second integer vector (5, 6)
    vectorClass<int> vec2_int(5, 6);
    // Calculate the dot product
    int dot_product_int = vec1_int.dotProduct(vec2_int);


    // --- Example 2: Using vectorClass with doubles ---
    // Create the first double vector (1.5, 2.0)
    vectorClass<double> vec1_double(1.5, 2.0);
    // Create the second double vector (2.5, 3.0)
    vectorClass<double> vec2_double(2.5, 3.0);
    // Calculate the dot product
    double dot_product_double = vec1_db.dotProduct(vec2_db);

  }
```

What is a Model Interface?

- Defines the `MODEL` type (using traits)
- Defines model-specific implementations of:
  - State
    - Defines `toFieldSet()` & `fromFieldSet()` methods
  - Increment
  - Geometry
    - Ports model geometry to `atlas::FunctionSpace`
  - Variable Changes (can use VADER)
- Provides access to TLM (if one exists; needed for 4DVar)
- Setup & run model forecast
- Provides model-filled templates of `OOPS::Applications`

OOPS Orchestration

How does JEDI do things with a model?

MPAS-JEDI Variational Application

```
/*
 * (C) Copyright 2017 UCAR
 *
 * This software is licensed under the terms of the Apache Licence Version 2.0
 * which can be obtained at http://www.apache.org/licenses/LICENSE-2.0.
 */

#include "oops/generic/instantiateModelFactory.h"
#include "oops/runs/Run.h"
#include "oops/runs/Variational.h"

#include "saber/oops/instantiateCovarFactory.h"
#include "saber/oops/instantiateLocalizationFactory.h"

#include "ufo/instantiateObsErrorFactory.h"
#include "ufo/instantiateObsFilterFactory.h"
#include "ufo/ObsTraits.h"

#include "mpasjedi/Traits.h"

int main(int argc,  char ** argv) {
  oops::Run run(argc, argv);
  oops::instantiateModelFactory<mpas::Traits>();
  saber::instantiateCovarFactory<mpas::Traits>();
  saber::instantiateLocalizationFactory<mpas::Traits>();
  ufo::instantiateObsErrorFactory();
  ufo::instantiateObsFilterFactory();
  oops::Variational<mpas::Traits, ufo::ObsTraits> var;
  return run.execute(var);
}
```

FV3-JEDI Variational Application

```
/*
 * (C) Copyright 2017-2020 UCAR
 *
 * This software is licensed under the terms of the Apache Licence Version 2.0
 * which can be obtained at http://www.apache.org/licenses/LICENSE-2.0.
 */

#include "fv3jedi/Utilities/Traits.h"
#include "oops/generic/instantiateModelFactory.h"
#include "oops/runs/Run.h"
#include "saber/oops/instantiateCovarFactory.h"
#include "ufo/instantiateObsErrorFactory.h"
#include "ufo/instantiateObsFilterFactory.h"
#include "ufo/ObsTraits.h"

#include "oops/runs/Variational.h"

int main(int argc,  char ** argv) {
  oops::Run run(argc, argv);
  saber::instantiateCovarFactory<fv3jedi::Traits>();
  ufo::instantiateObsErrorFactory();
  ufo::instantiateObsFilterFactory();
  oops::instantiateModelFactory<fv3jedi::Traits>();
  oops::Variational<fv3jedi::Traits, ufo::ObsTraits> var;
  return run.execute(var);
}
```

OOPS Orchestration

DA 'tasks' are defined as subclasses of `OOPS::Application`

```
class Application : public util::Printable {
 public:
  explicit Application(const eckit::mpi::Comm & comm) : comm_(comm) {}
  virtual ~Application() {}
  virtual int execute(const eckit::Configuration &, bool validate) const = 0;
```

What the application does is defined in the `execute()` method.

```
template <typename MODEL, typename OBS> class Variational : public Application {
  typedef Increment<MODEL>            Increment_;
  typedef State<MODEL>                State_;
  typedef Model<MODEL>                 Model_;
  typedef ModelAuxControl<MODEL>       ModelAux_;

 public:
// -----------------------------------------------------------------------------
  explicit Variational(const eckit::mpi::Comm & comm = oops::mpi::world()) : Application(comm) {
    instantiateCostFactory<MODEL, OBS>();
    instantiateCovarFactory<MODEL>();
    ...
  }
  int execute(const eckit::Configuration & fullConfig, bool validate) const override {
    Log::trace() << "Variational: execute start" << std::endl;
```

```
oops/src/
  |- test/
  |- oops/
       |- assimilation/
       |- base/
       |- generic/
       |- interface/
 -->   |- runs/
           Application.h
           Variational.h
           ...
       ...
```

YAML Ain't Markup Language

Experiments are setup in yaml files

YAML files are read by eckit and written into `eckit::Configuration` object that is passed to OOPS.

```cpp
template <typename MODEL, typename OBS> class Variational : public Application {
  typedef Increment<MODEL>             Increment_;
  typedef State<MODEL>                 State_;
  typedef Model<MODEL>                  Model_;
  typedef ModelAuxControl<MODEL>        ModelAux_;

 public:
// -----------------------------------------------------------------------------
  explicit Variational(const eckit::mpi::Comm & comm = oops::mpi::world()) : Application(comm) {
    instantiateCostFactory<MODEL, OBS>();
    instantiateCovarFactory<MODEL>();
    ...
  }
  int execute(const eckit::Configuration & fullConfig, bool validate) const override {
    Log::trace() << "Variational: execute start" << std::endl;
```

```yaml
cost function:
  cost type: 3D-Var
  time window:
    begin: '2018-04-14T21:00:00Z'
    length: PT6H
  geometry:
    nml_file: "./Data/480km/namelist.atmosphere_2018041500"
    streams_file: "./Data/480km/streams.atmosphere"
  analysis variables: [list of an_vars]
  background:
    state variables: [List of vars to read from state file]
    filename: "./Data/480km/bg/restart.2018-04-15_00.00.00.nc"
    date: &analysisdate '2018-04-15T00:00:00Z'
  background error:
    covariance model: SABER
    saber central block:
      saber block name: ID
      active variables: [sub-set of analysis vars]
    saber outer blocks:
    - saber block name: StdDev
      ...
  observations:
    observers:
    - obs space:
        name: MY_OBS
        ...
          obsfile: <path to MY_OBS file>
        ...
variational:
  minimizer:
    algorithm: DRPCG
  iterations:
  - geometry:
      nml_file: "./Data/480km/namelist.atmosphere_2018041500"
      streams_file: "./Data/480km/streams.atmosphere"
    ninner: '10'
```

Configuring a Run

```yaml
cost function:
  # 4dfgat utilizes the 4D-Var cost function with an identity linear model
  cost type: 4D-Var
  time window:
    begin: '2018-04-14T21:00:00Z'
    length: PT3H
  geometry:
    nml_file: "./Data/480km/namelist.atmosphere_2018041421"
    streams_file: "./Data/480km/streams.atmosphere"
  model:
    name: MPAS
    tstep: PT30M
    model variables: &modvars
                    [temperature, spechum, uReconstructZonal, uReconstructMeridional, surface_pressure,
                    theta, rho, u, qv, pressure, landmask, xice, snowc, skintemp, ivgtyp, isltyp,
                    qc, qi, qr, qs, qg, pressure_p, snowh, vegfra, u10, v10, lai, smois, tslb, w]
  analysis variables: &incvars
  - temperature
  - spechum
  - uReconstructZonal
  - uReconstructMeridional
  - surface_pressure
  background:
    state variables: *modvars
```

Type of cost function

DA window

Resolution/Geometry

The model we want

Variables

Cost Function Name and window

Model

Analysis variables

Background

Cost Function Whole

```yaml
  - spechum
  - uReconstructZonal
  - uReconstructMeridional
  - surface_pressure
background:
  state variables: *modvars
  filename: "./Data/480km/bg/restart.2018-04-14_21.00.00.nc"
  date: '2018-04-14T21:00:00Z'
background error:
  covariance model: MPASstatic
  date: '2018-04-14T21:00:00Z'
observations:
  observers:
  - obs space:
      name: Radiosonde
      obsdatain:
        engine:
          type: H5File
          obsfile: Data/obs/mpasobsappend1/sondes_obs_2018041500_m.nc4
      obsdataout:
        engine:
          type: H5File
          obsfile: Data/os/obsout_4dfgat_append_obs_sondes.nc4
      simulated variables: [airTemperature, windEastward, windNorthward, specificHumidity]
    obs operator:
      name: VertInterp
      observation alias file: testinput/obsop_name_map.yaml
    obs error:
      covariance model: diagonal
```

Variables

Analysis variables

Background

File location

B matrix we choose

B Matrix

Start Obs Section

First Obs Type

Output / OMA,OMB

Observations

Obs Operator we want to the obs type

R matrix for the obs type

Cost Functior
Whole

```
    obs operator:
      name: VertInterp
      observation alias file: testinput/obsop_name_map.yaml
    obs error:
      covariance model: diagonal
    obs filters:
    - filter: PreQC
      maxvalue: 3
    - filter: Background Check
      threshold: 3
  - obs space:
      name: GnssroRefNCEP
      obsdatain:
        engine:
          type: H5File
          obsfile: Data/ufo/testinput_tier_1/gnssro_obs_2018041500_s.nc4
      obsdataout:
        engine:
          type: H5File
          obsfile: Data/os/obsout_4dfgat_append_obs_gnssroref.nc4
      simulated variables: [atmosphericRefractivity]
    obs operator:
      name: GnssroRefNCEP
      obs options:
        use_compress: 0
    obs error:
      covariance model: diagonal
    obs filters:
    - filter: Domain Check
```

Obs Operator we want to the obs type

R matrix for the obs type

QC for the obs type

Similar section for

Observations

Cost
Func
Who

```yaml
        threshold: 3
    - filter: ROobserror
      apply at iterations: 0,1
      variable: refractivity
      errmodel: NCEP
  - obs space:
      name: SfcPCorrected
      obsdatain:
        engine:
          type: H5File
          obsfile: Data/ufo/testinput_tier_1/sfc_obs_2018041500_m.nc4
      obsdataout:
        engine:
          type: H5File
          obsfile: Data/os/obsout_4dfgat_append_obs_sfc.nc4
      simulated variables: [stationPressure]
    obs operator:
      name: SfcPCorrected
      da_psfc_scheme: UKMO   # or WRFDA
    linear obs operator:
      name: Identity
      observation alias file: testinput/obsop_name_map.yaml
    obs error:
      covariance model: diagonal
    obs filters:
    - filter: PreQC
      maxvalue: 3
    - filter: Difference Check
      apply at iterations: 0,1
```

Similar section for another obs type

Observations

Cost Function Whole

```yaml
      value: GeoVaLs/height_above_mean_sea_level_at_surface
      threshold: 500
    - filter: Background Check
      apply at iterations: 0,1
      threshold: 3
variational:
  minimizer:
    algorithm: DRPCG
  iterations:
  - geometry:
      nml_file: "./Data/480km/namelist.atmosphere_2018041421"
      streams_file: "./Data/480km/streams.atmosphere"
    linear model:
      # 4dfgat utilizes the generic identity linear model, implemented in oops::IdentityLinearModel
      name: Identity
      increment variables: *incvars
      tstep: PT30M
    ninner: '10'
    gradient norm reduction: 1e-10
  - geometry:
      nml_file: "./Data/480km/namelist.atmosphere_2018041421"
      streams_file: "./Data/480km/streams.atmosphere"
    linear model:
      # 4dfgat utilizes the generic identity linear model, implemented in oops::IdentityLinearModel
      name: Identity
      increment variables: *incvars
      tstep: PT30M
    ninner: '10'
    gradient norm reduction: 1e-10
```

Define how the variational run will go

Choose minimizer scheme

Resolution of inner loop

Linear model, here it's a 4d-Var cost function so we need one

Number of inner loops or, norm reduction to reach

First
outer loop

next
outer loop

Varia
Run
Confi

```yaml
minimizer:
  algorithm: DRPCG
iterations:
- geometry:
    nml_file: "./Data/480km/namelist.atmosphere_2018041421"
    streams_file: "./Data/480km/streams.atmosphere"
  linear model:
    # 4dfgat utilizes the generic identity linear model, implemented in oops::IdentityLinearModel
    name: Identity
    increment variables: *incvars
    tstep: PT30M
  ninner: '10'
  gradient norm reduction: 1e-10
- geometry:
    nml_file: "./Data/480km/namelist.atmosphere_2018041421"
    streams_file: "./Data/480km/streams.atmosphere"
  linear model:
    # 4dfgat utilizes the generic identity linear model, implemented in oops::IdentityLinearModel
    name: Identity
    increment variables: *incvars
    tstep: PT30M
  ninner: '10'
  gradient norm reduction: 1e-10
  test: 'on'
  continuous DA:
    obs append directory: Data/obs/mpasobsappend2
output:
  filename: "Data/states/mpas.4dfgat_append_obs.$Y-$M-$D_$h.$m.$s.nc"
  stream name: analysis
```

Choose minimizer scheme

Resolution of inner loop

Linear model, here it's a 4d-Var cost function so we need one

Number of inner loops or, norm reduction to reach

New feature in JEDI, add more obs in before this outer loop starts

Output path

First outer loop

next outer loop

Variational Run Config

Output

Instantiation

```yaml
cost function:
  cost type: 4D-Var
  time window:
    begin: '2018-04-14T21:00:00Z'
    length: PT3H
    …
```

```
jedi-bundle/oops/src/oops/assimilation
── BMatrix.h
── CMatrix.h
── ControlIncrement.h
── ControlVariable.h
── CostFct3DVar.h
── CostFct4DEnsVar.h
── CostFct4DVar.h
── CostFctFGAT.h
── CostFctWeak.h
── CostFunction.h
── CostJb3D.h
── CostJb4D.h
── CostJbJq.h
── CostJbModelAux.h
── CostJbObsAux.h
── CostJbState.h
── CostJbTotal.h
── CostJcDFI.h
── CostJo.h
── CostPert.h
── CostTermBase.h
── DRGMRESRMinimizer.h
── DRIPCGMinimizer.h
── DRMinimizer.h
```

oops/src/oops/Variational.h

```cpp
// ---------------------------------------------------------------------------
  virtual ~Variational() {}
// ---------------------------------------------------------------------------
  int execute(const eckit::Configuration & fullConfig) const override {
    Log::trace() << "Variational: execute start" << std::endl;
    util::printRunStats("Variational start");
/// The background is constructed inside the cost function because its valid
/// time within the assimilation window can be different (3D-Var vs. 4D-Var),
/// it can be 3D or 4D (strong vs weak constraint), etc...

//  Setup cost function
    eckit::LocalConfiguration cfConf(fullConfig, "cost function");
    std::unique_ptr<CostFunction<MODEL, OBS>>
      J(CostFactory<MODEL, OBS>::create(cfConf, this->getComm()));

//  Initialize first guess from background
    ControlVariable<MODEL, OBS> xx(J->jb().getBackground());
```

oops/src/oopsassimilation/CostFunction.h

```cpp
// ---------------------------------------------------------------------------
template <typename MODEL, typename OBS>
CostFunction<MODEL, OBS>* CostFactory<MODEL, OBS>::create(const eckit::Configuration & config,
                                                          const eckit::mpi::Comm & comm) {
  std::string id = config.getString("cost type");
  Log::trace() << "Variational Assimilation Type=" << id << std::endl;
  typename std::map<std::string, CostFactory<MODEL, OBS>*>::iterator j = getMakers().find(id);
  if (j == getMakers().end()) {
    throw std::runtime_error(id + " does not exist in cost function factory.");
  }
  Log::trace() << "CostFactory::create found cost function type" << std::endl;
  return (*j).second->make(config, comm);
}
```

JEDI configurability and choices

From : jedi-bundle/fv3-jedi/test/testinput/4dvar_hybrid_linear_model.yaml

```yaml
linear model:
  name: HTLM
  simple linear model:
    linear model:
      name: FV3JEDITLM
      namelist filename: Data/fv3files/input_gfs_c12.nml
      linear model namelist filename: Data/fv3files/inputpert_4dvar.nml
      tstep: PT1H
      lm_do_dyn: 1
      lm_do_trb: 0
      lm_do_mst: 0
      tlm variables:
      - u
      - v
      - T
      - delp
      - sphum
      - ice_wat
      - liq_wat
      - o3mr
      trajectory:
        model variables: *modelvars
  update tstep: PT3H
  variables: *anavars
  coefficients:
    input:
      base filepath: Data/hybrid_linear_model_coeffs
      one file per task: true
    update variables:
    - eastward_wind
    - northward_wind
    influence region size: 3
    time window:
      begin: 2020-12-14T21:00:00Z
      length: PT6H
```

From : jedi-bundle/fv3-jedi/test/testinput/4dvar_geos_cf.yaml

```yaml
linear model:
  name: FV3JEDITLM
  namelist filename: Data/fv3files/input_geos_c12_p12.nml
  linear model namelist filename: Data/fv3files/inputpert_4dvar.nml
  tstep: PT3H
  tlm variables: &modelvars
  - u
  - v
  - ua
  - va
  - T
  - Q
  - delp
  - NO2
  lm_do_dyn: 1
  lm_do_trb: 0
  lm_do_mst: 0
  trajectory:
    model variables: *modelvars
```

Running the Executable

Assuming you have set your JEDI_ROOT and JEDI_BUILD directories you can execute by passing your yaml config to the relevant executable like so providing the path to your yaml file of course.

```
JCSDA:~$ $JEDI_ROOT/$JEDI_BUILD/bin/mpasjedi_variational.x PATH/TO/YAML_CONFIG/config.yaml
```

- Applications with one initial state
  - `mpasjedi_convertstate.x` (`oops::ConvertState`)
  - `mpasjedi_dirac.x` (`oops::Dirac`)
  - `mpasjedi_forecast.x` (`oops::Forecast`): essentially does the same as the `mpas_atmosphere` executable, but through the JEDI generic framework via the MPAS-JEDI interface. There is more overhead than when running the non-JEDI exectuable, and this requires a YAML file in addition to the standard `namelist.atmosphere` used to configure `mpas_atmosphere`.
  - `mpasjedi_gen_ens_pert_B.x` (oops::GenEnsPertB)
  - `mpasjedi_hofx.x` (`oops::HofX4D`)
  - `mpasjedi_hofx3d.x` (`oops::HofX3D`)
  - `mpasjedi_parameters.x` (`saber::EstimateParams`): used to estimate static background error covariance and localization matrices
  - `mpasjedi_staticbinit.x` (`oops::StaticBInit`)
  - `mpasjedi_variational.x` (`oops::Variational`): carries out many different flavors of variational data assimilation (3DVar, 3DEnVar, 3DFGAT, 4DEnVar) with a variety of incremental minimization algorithms
- Applications with multiple initial states
  - `mpasjedi_eda.x` (`oops::EnsembleApplication<oops::Variational>`)
  - `mpasjedi_enshofx.x` (`oops::EnsembleApplication<oops::HofX4D>`)
  - `mpasjedi_rtpp.x` (`oops::RTPP`): standalone application that carries out Relaxation to Prior Perturbation, as introduced by Zhang et al. (2004). The intended purpose is to inflate the analysis ensemble spread after running the EDA application.

JEDI Capabilities & Features

Models:

- Lorenz95, QG (native toy models)
- FV3-JEDI
- MPAS-JEDI
- SOCA

DA Flavors:

- 3DVar/3D-FGAT
- 4DVar (-Weak Constraint)
- 4DEnVar
- EnKF (LETKF, GETKF)

Background Error Models (SABER):

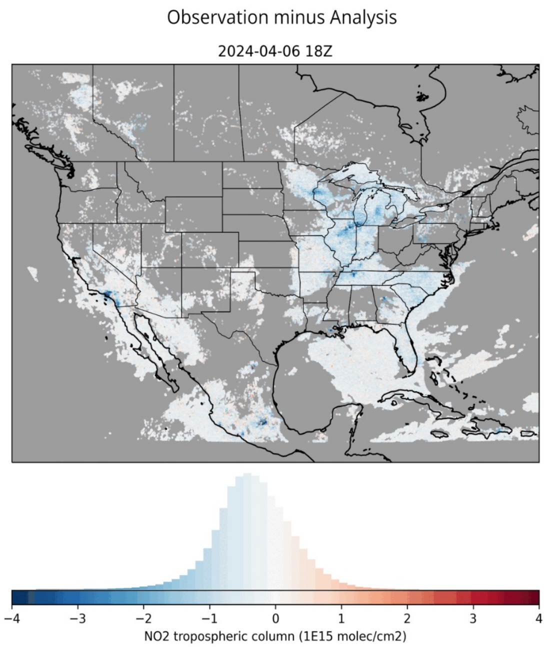
- BUMP
- Explicit Diffusion
- Spectral Filtering
- GSI Interface (limited)

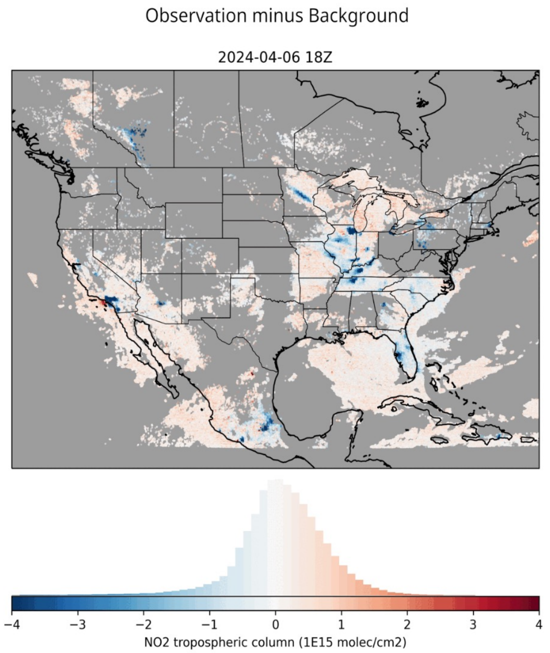
UFO:

- Wide collection of Obs Operators (remote & in-situ)
- Bias correction
- Quality Control

JEDI Recent Accomplishments: TEMPO

Assimilation of recently released TEMPO observations, first to do it and thus available to the partners!

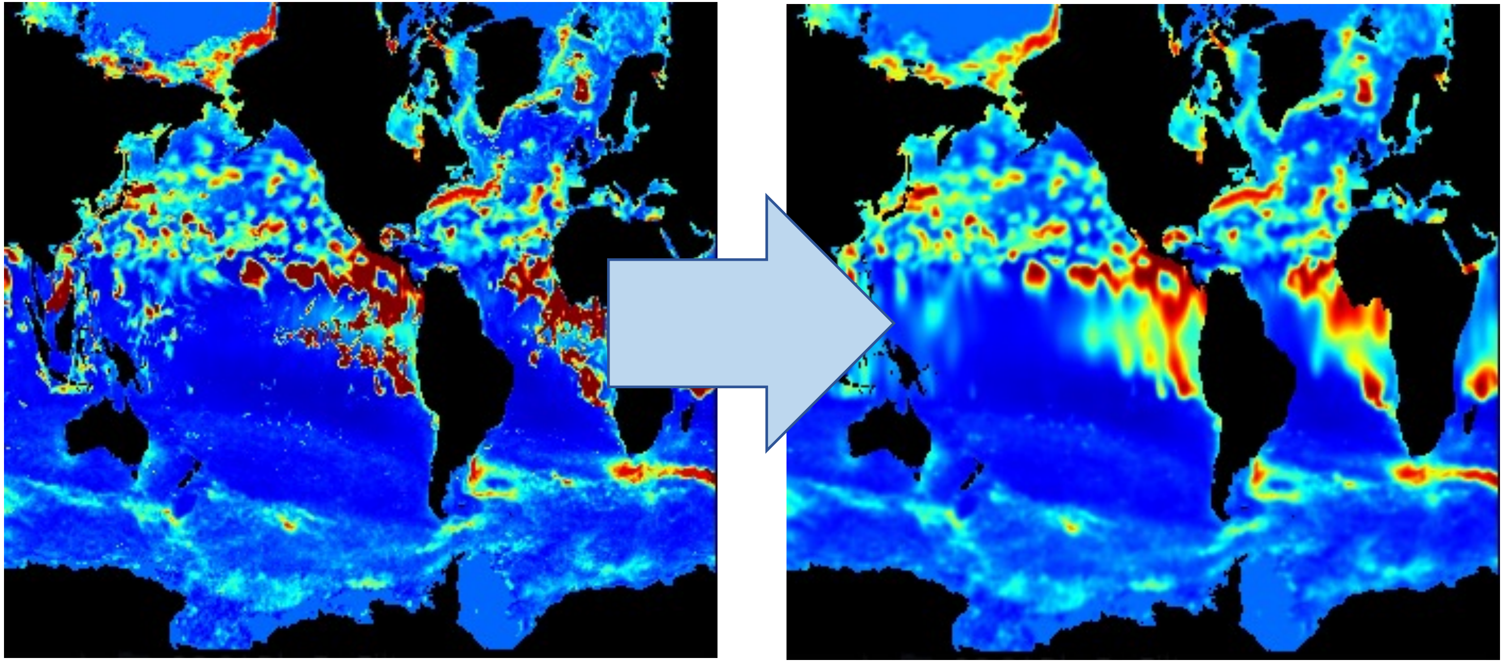




TEMPO NO2 assimilation using JEDI and GEOS-CF at ~25km (c360) resolution.

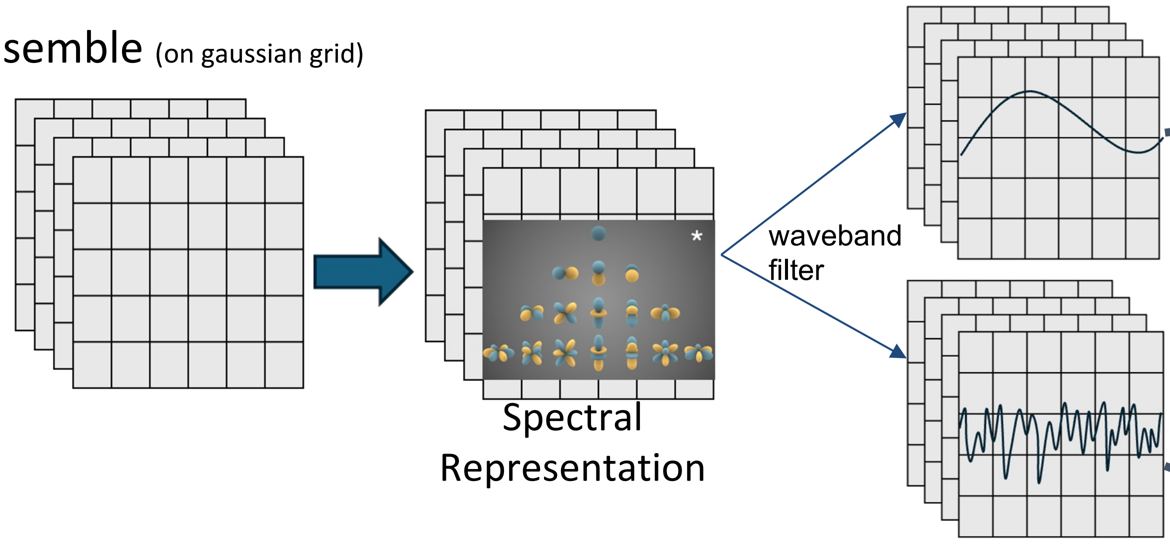
JEDI Recent Accomplishments: B Diffusion

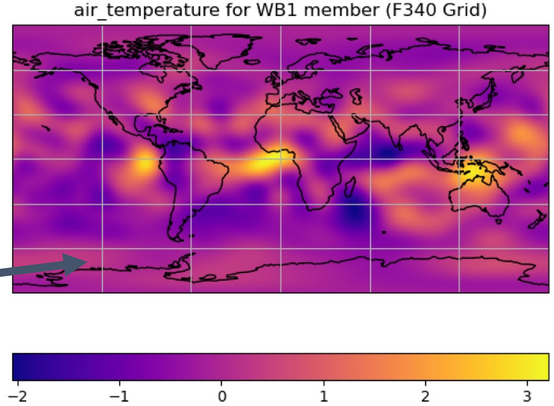
Addition of explicit diffusion model for background errors



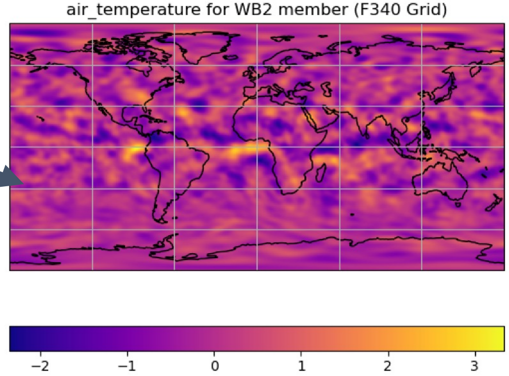
JEDI Recent Acomplishments: Multi-Scale Localization

Ensemble (on gaussian grid)



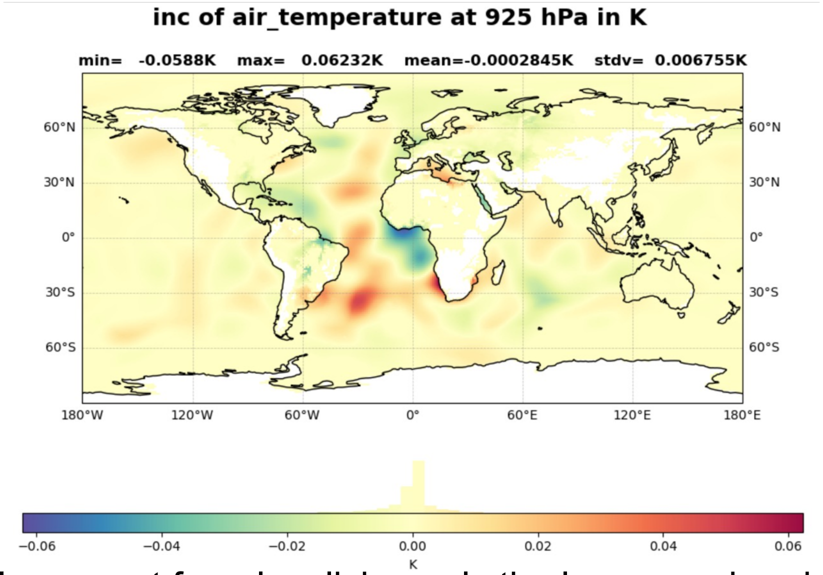


Filtered ensemble members: long waveband (above); short wave-band (below)

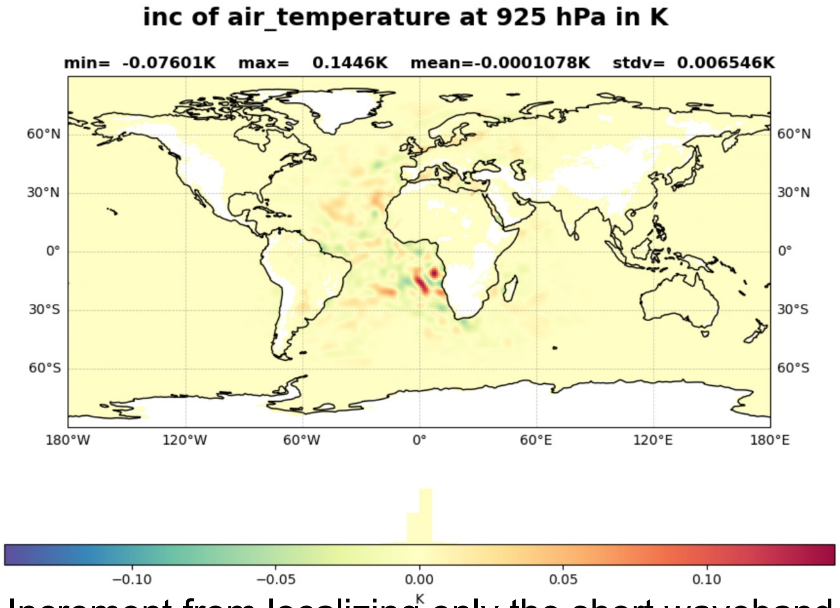


JEDI Recent Acomplishments: Multi-Scale Localization

Results from Single Obs Test:

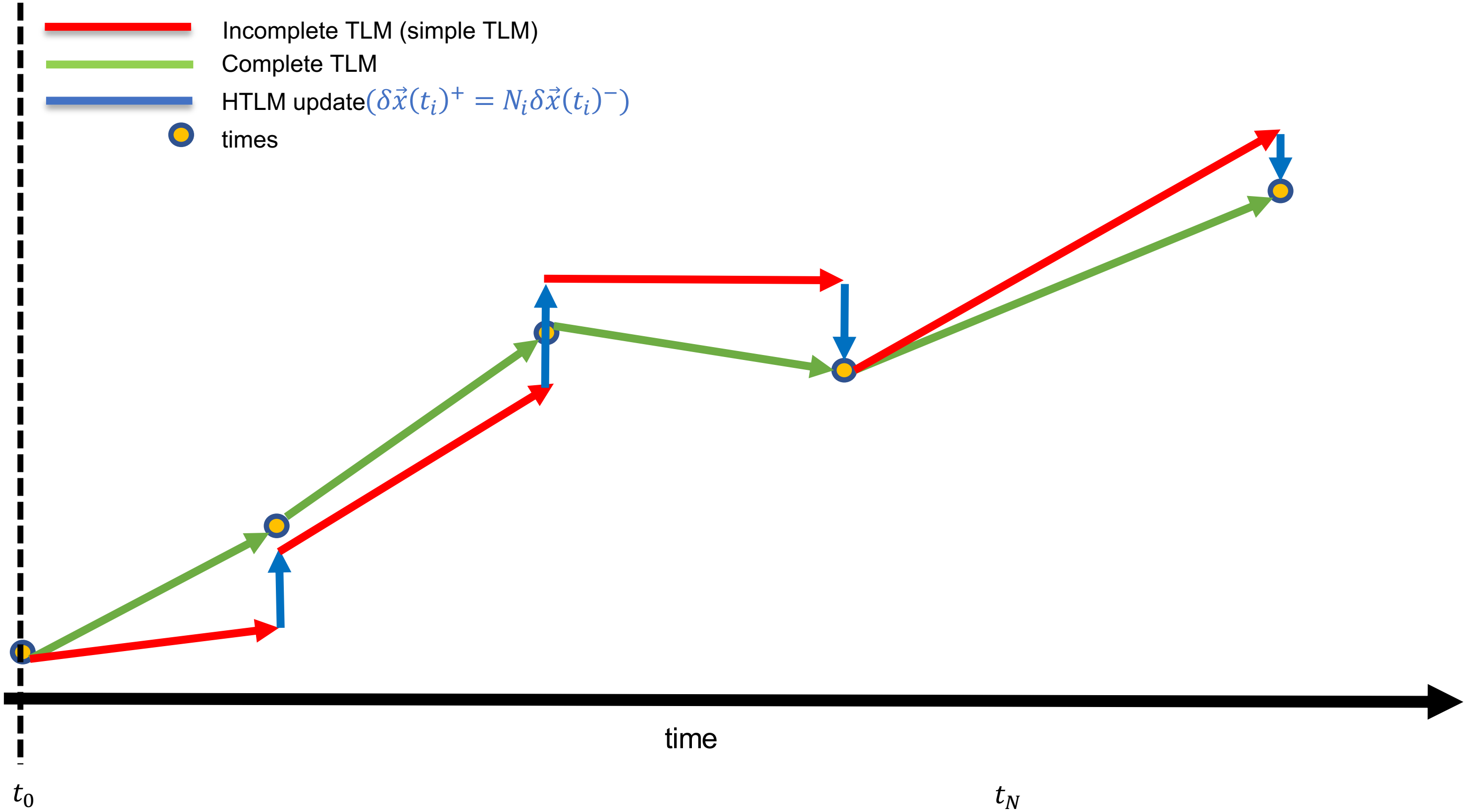


Increment from localizing only the long waveband



Increment from localizing only the short waveband

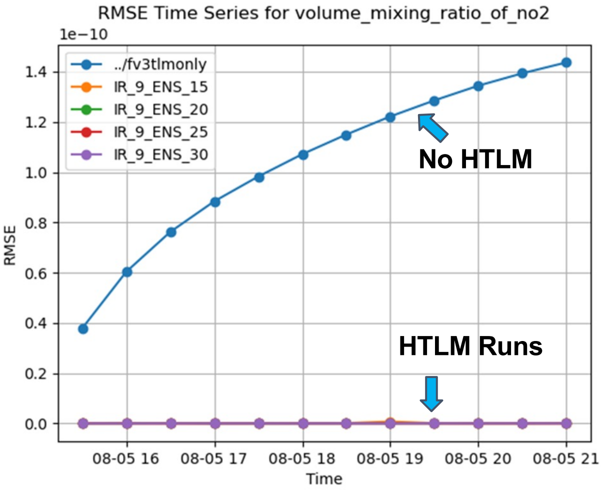
JEDI Recent Accomplishments: HTLM



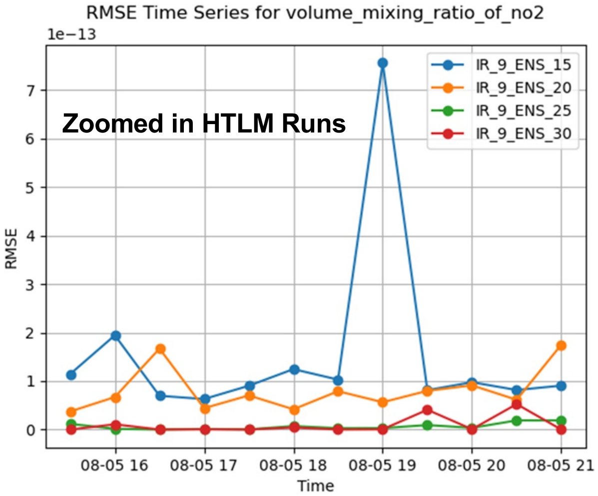
Hybrid Tangent Linear Model (HTLM)

JEDI Recent Accomplishments: HTLM

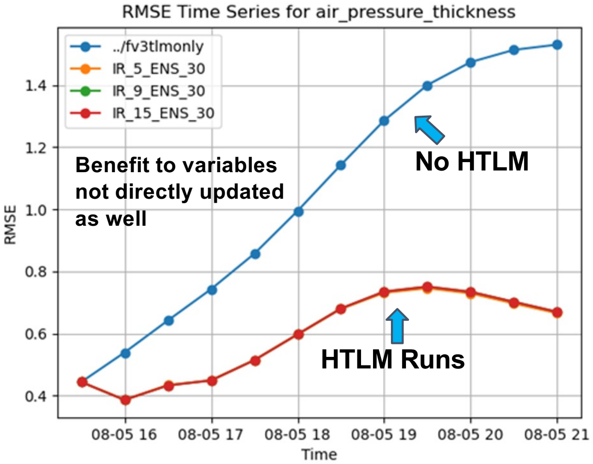
- The HTLM is showing good reduction in linearization errors
- The linearization error is still reduced well even with relatively small ensemble sizes
- We also see reduction in linearization error for variables we don't directly update with the HTLM
- The linearization error reduction is also robust in update frequency.



Linearization error with GEOS and fv3-tlm comparing no HTLM and HTLM runs with different numbers of ensemble members for the updated variable of NO2



Linearization error with GEOS and fv3-tlm comparing HTLM runs with different numbers of ensemble members (ZOOMED IN FROM LEFT) for the updated variable NO2

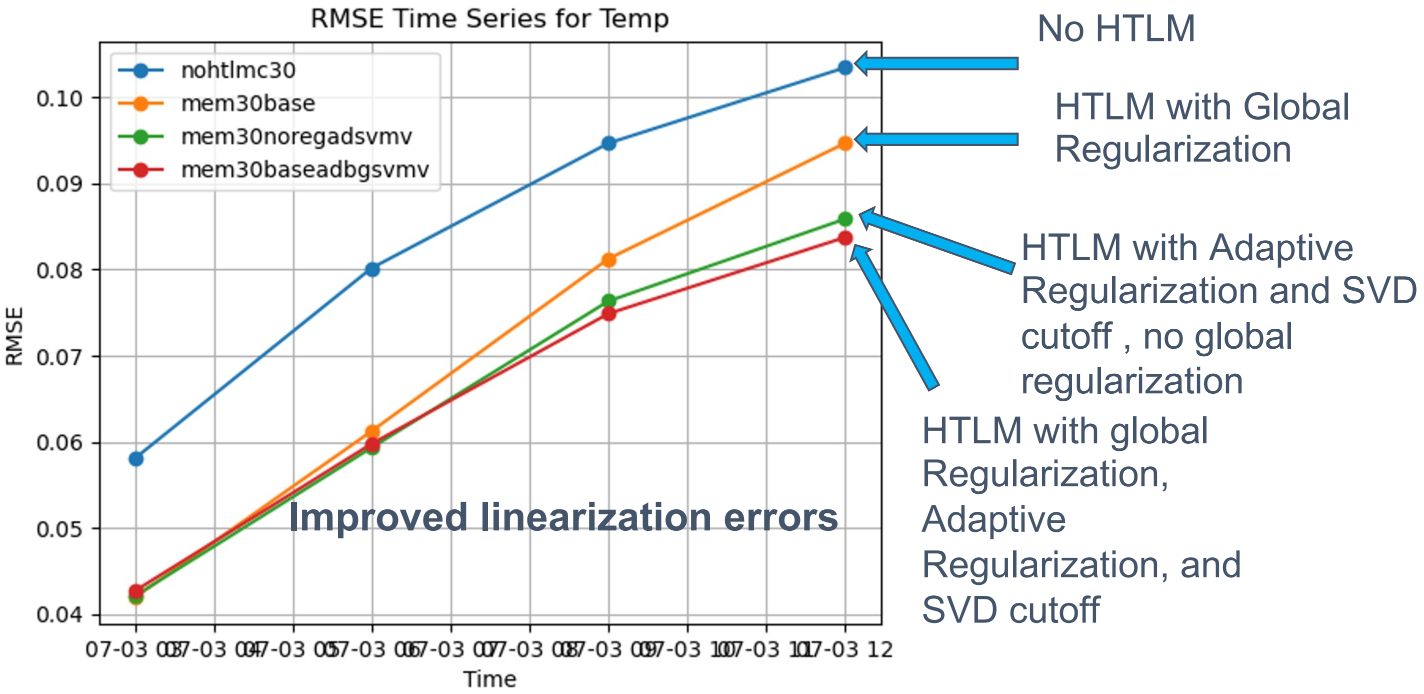


Linearization error with GEOS and fv3-tlm comparing no HTLM and HTLM runs with different influence region sizes for the air pressure thickness which was **not** directly updated by the HTLM

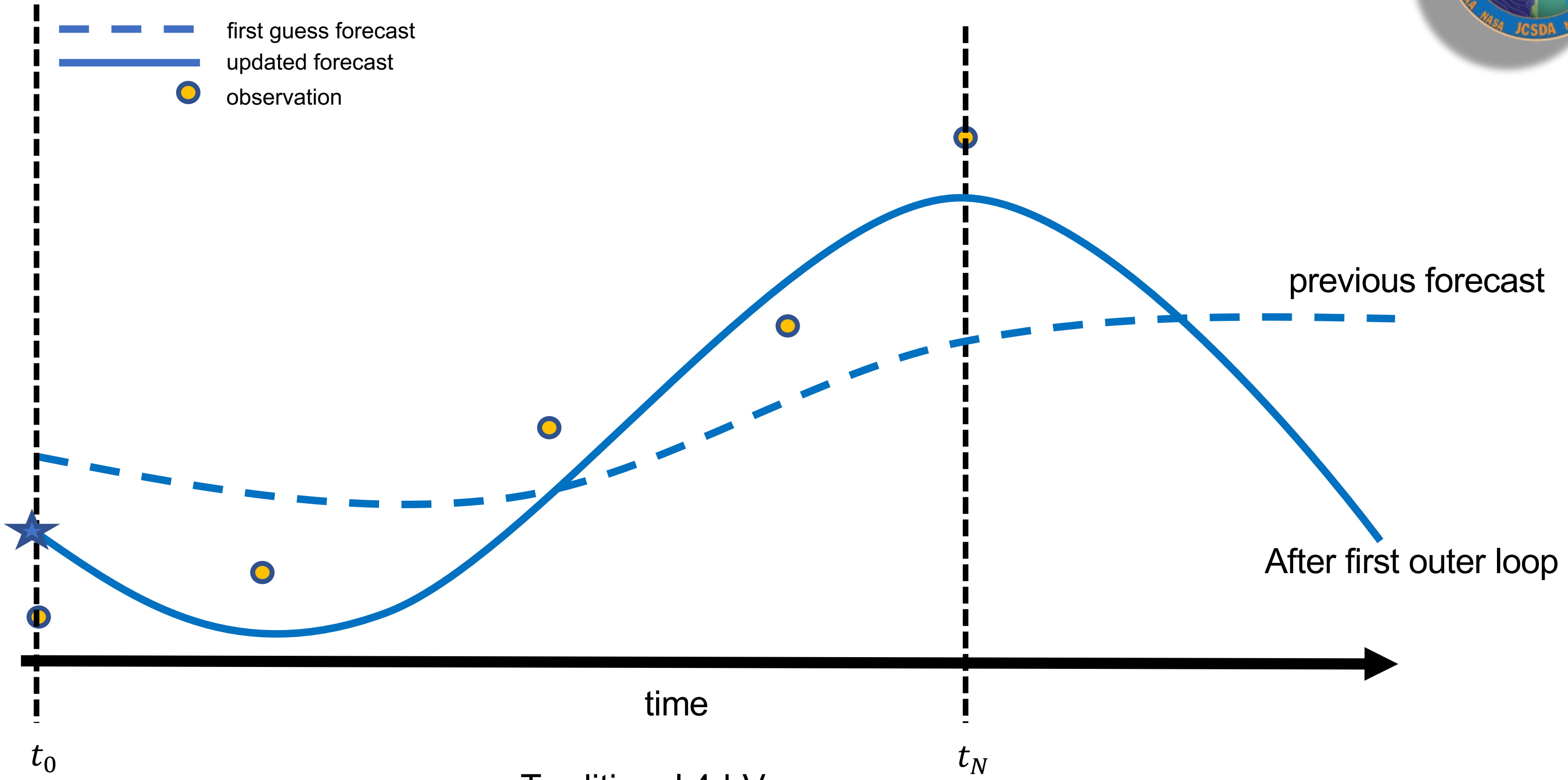
JEDI Recent Accomplishments: HTLM

Added an (maximum condition number based) adaptive regularization scheme and svd truncation to the HTLM.

When enabled shows improved linearization error, especially over several timesteps.

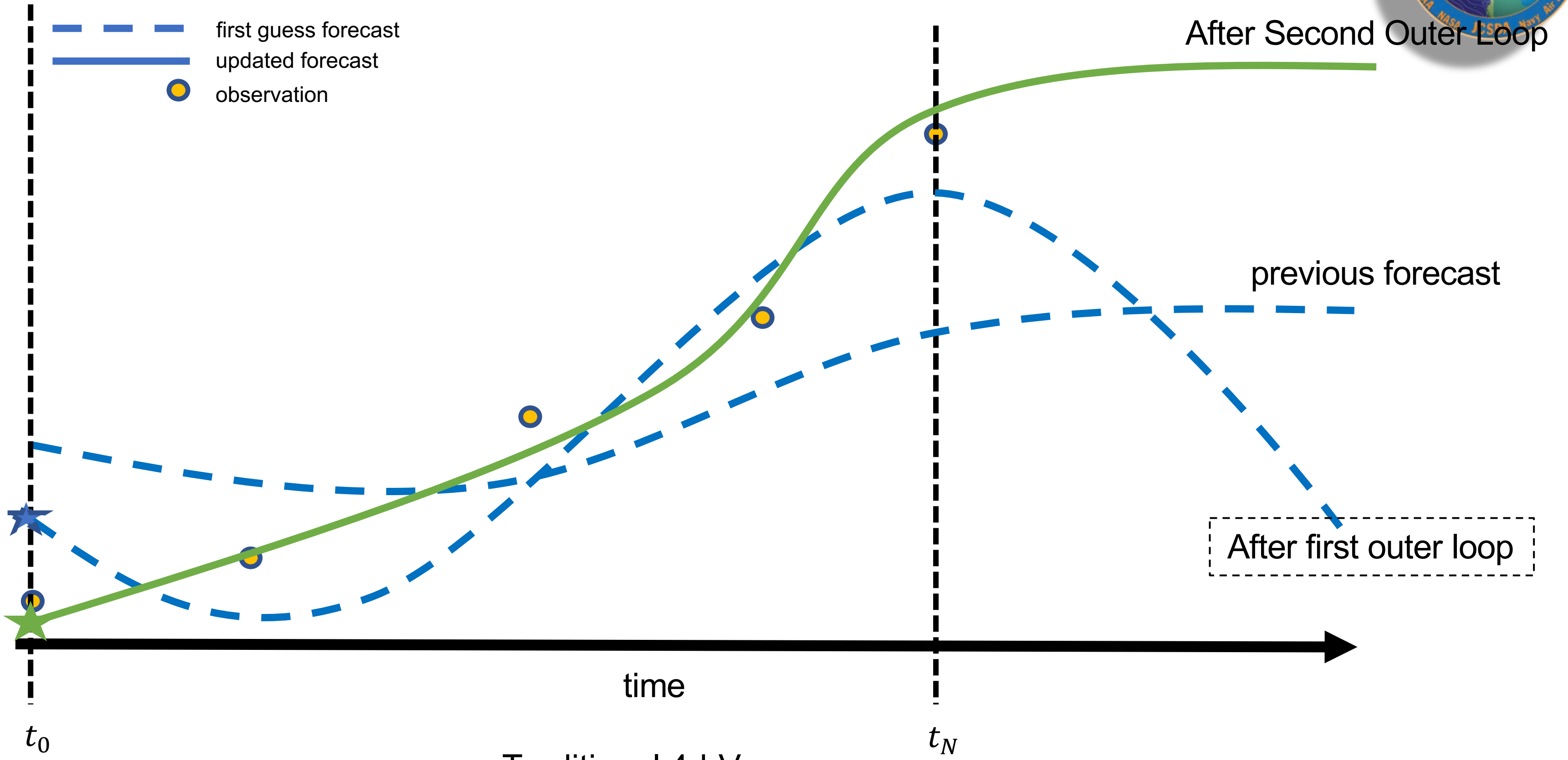


JEDI Recent Accomplishments: CDA



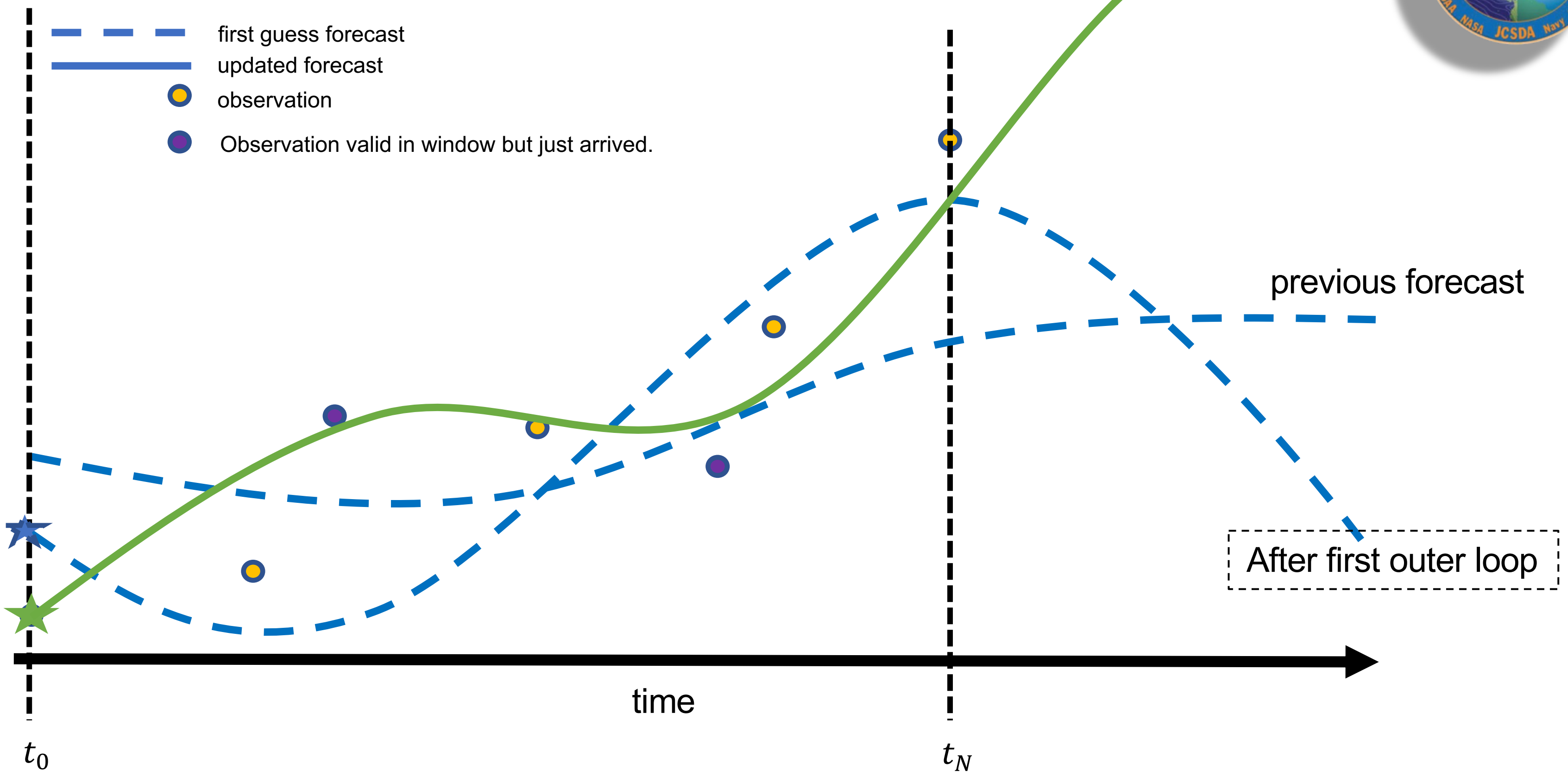
Traditional 4d-Var

JEDI Recent Accomplishments: CDA



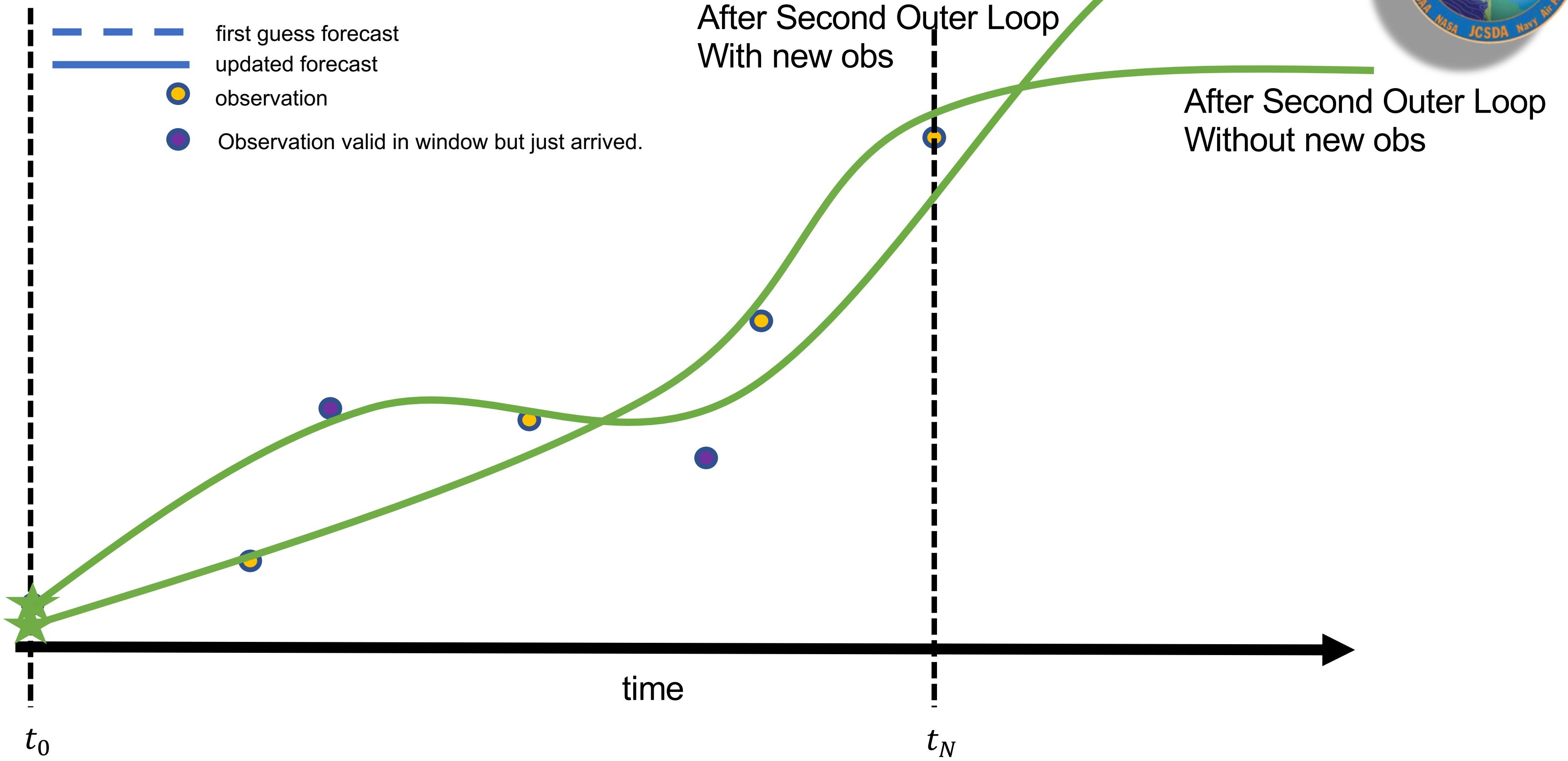
Traditional 4d-Var

JEDI Recent Accomplishments: CDA



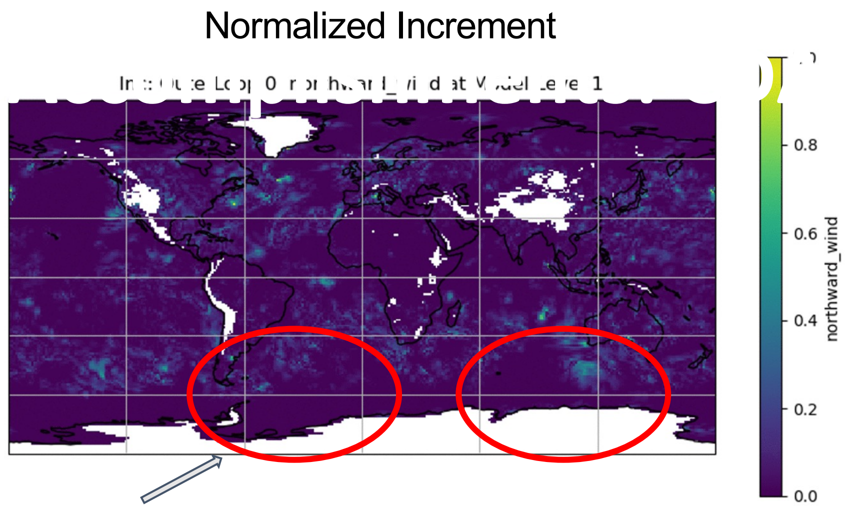
Continuous DA (CDA)

JEDI Recent Accomplishments: CDA

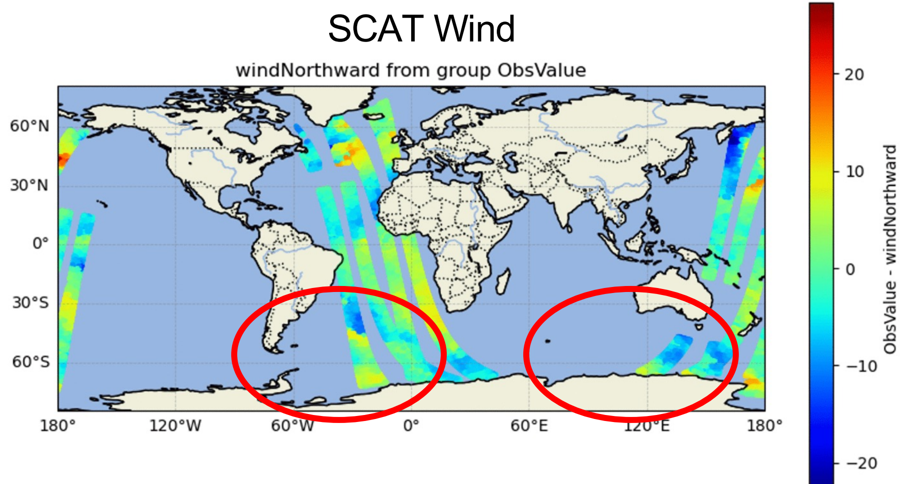


Continuous DA (CDA)

JEDI Rec

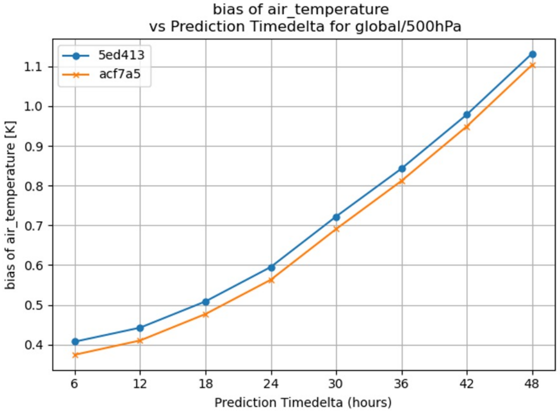


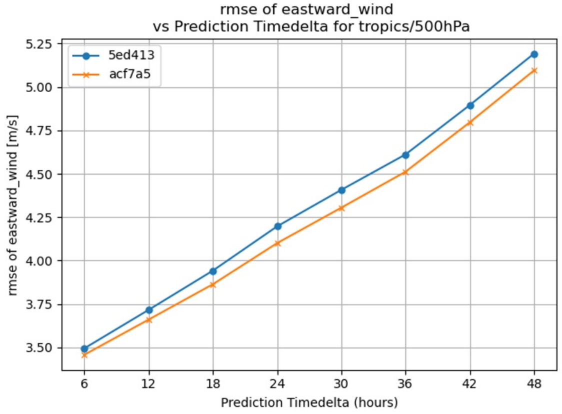
Additional increments in locations where obs are added between outer loops.

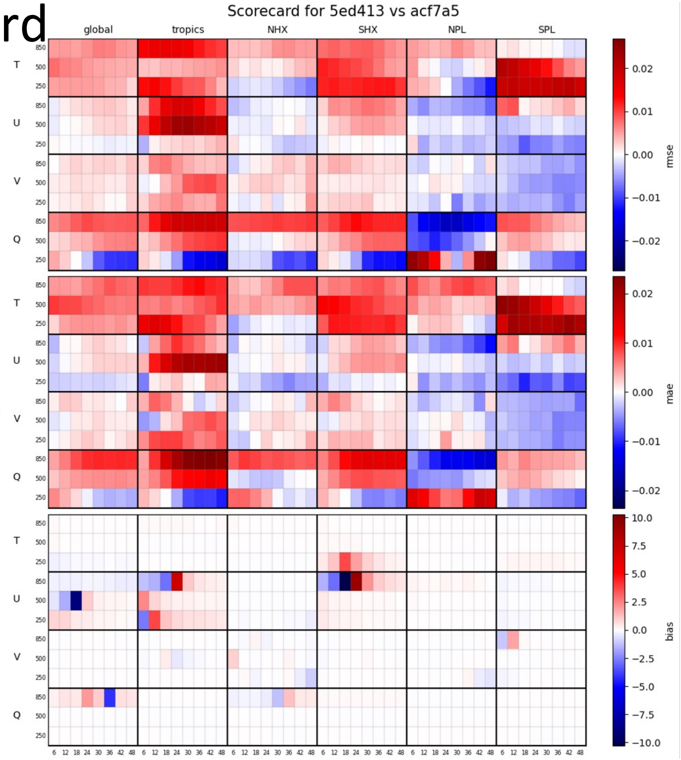


JEDI Recent Acomplishments: Benchmarking

- Weatherbench scores against ERA5 and scorecard capability to compare experiments available in EWOK (work with OBS and INFRA teams)
- **Metrics** RMSE, MAE, Forecast bias
- **Variables** from ERA5 reanalysis



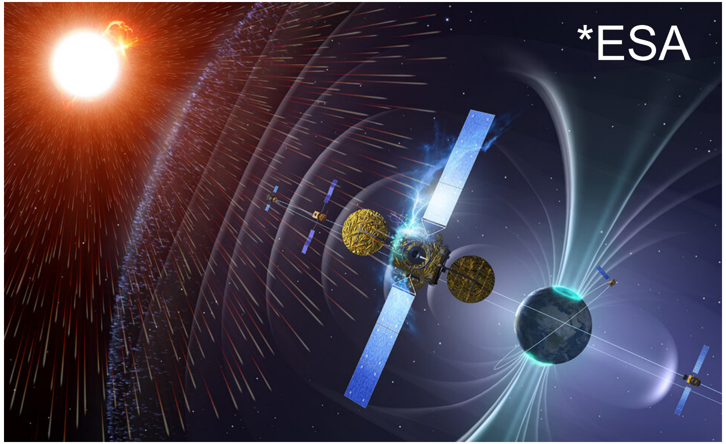




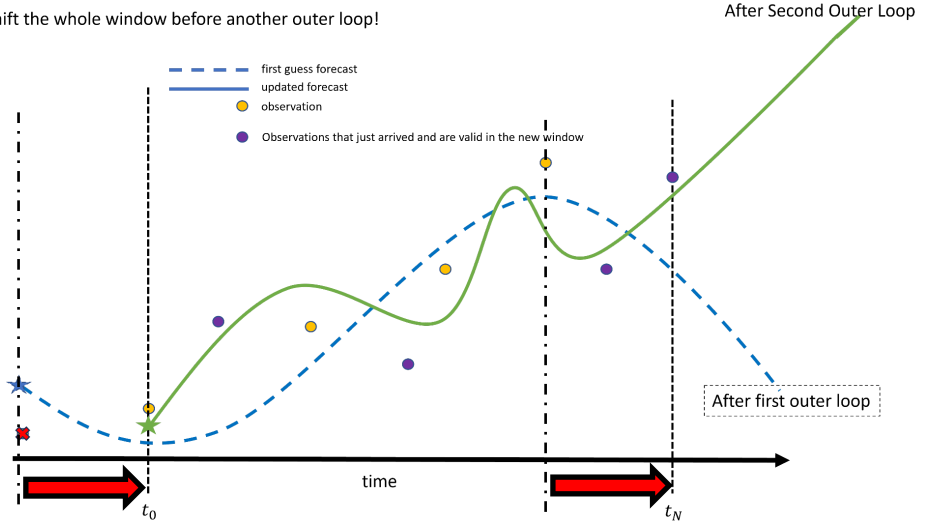
Current JEDI Efforts

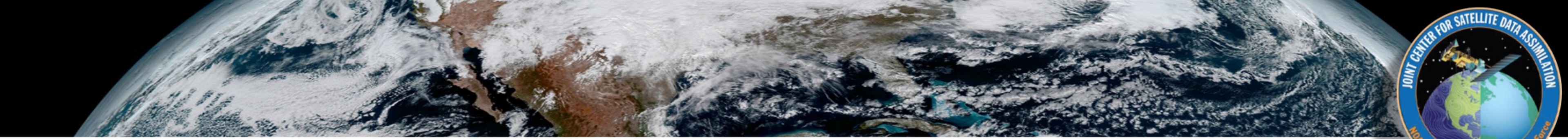
- Transition to Operations
- JEDI optimizations for CPUs & GPUs
- Continuous DA, with window shifts.

JEDI is meant to focus efforts in such a way that innovation is shared, accelerated, and does not need to be reproduced. In addition, once observations are available in JEDI they should be available to any model with a JEDI interface.



C^{∞} DA shift the whole window before another outer loop!





Questions?

JEDI Documentation: https://jointcenterforsatellitedataassimilation-jedi-docs.readthedocs-hosted.com/en/latest/index.html
JEDI Forum: https://forums.jcsda.org/ (requires account to post/comment)
Github: https://github.com/JCSDA (public)