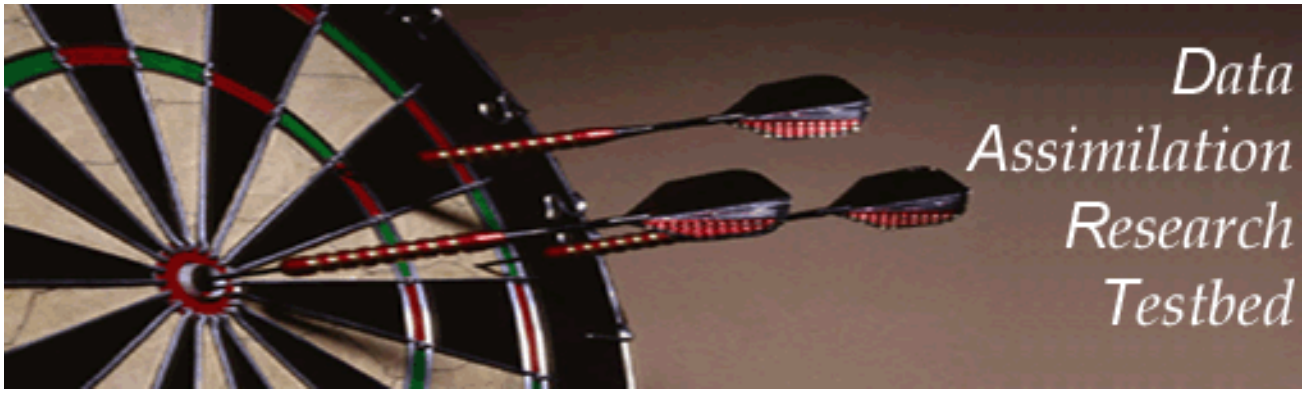
Data Assimilation Research Testbed Tutorial

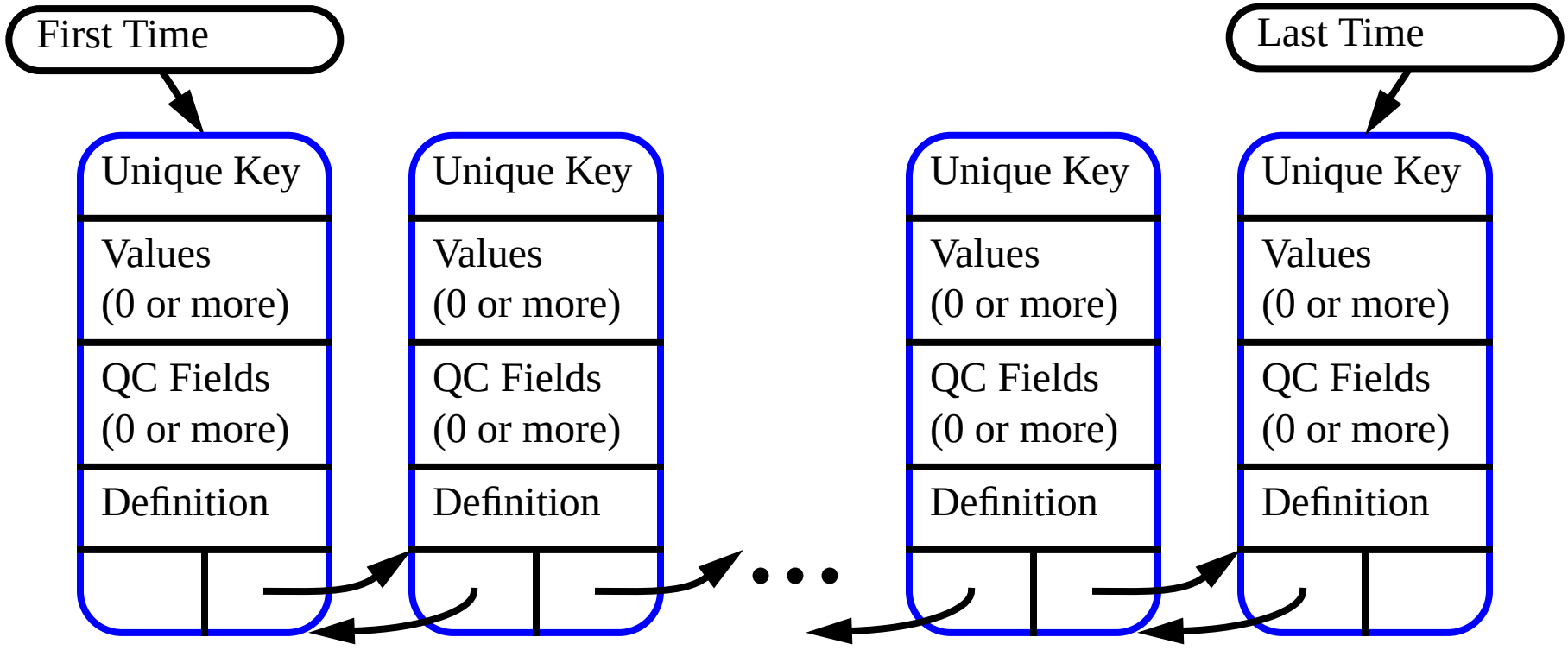


Section 21: Observation Types and Observing System Design

Version 1.0: October, 2005

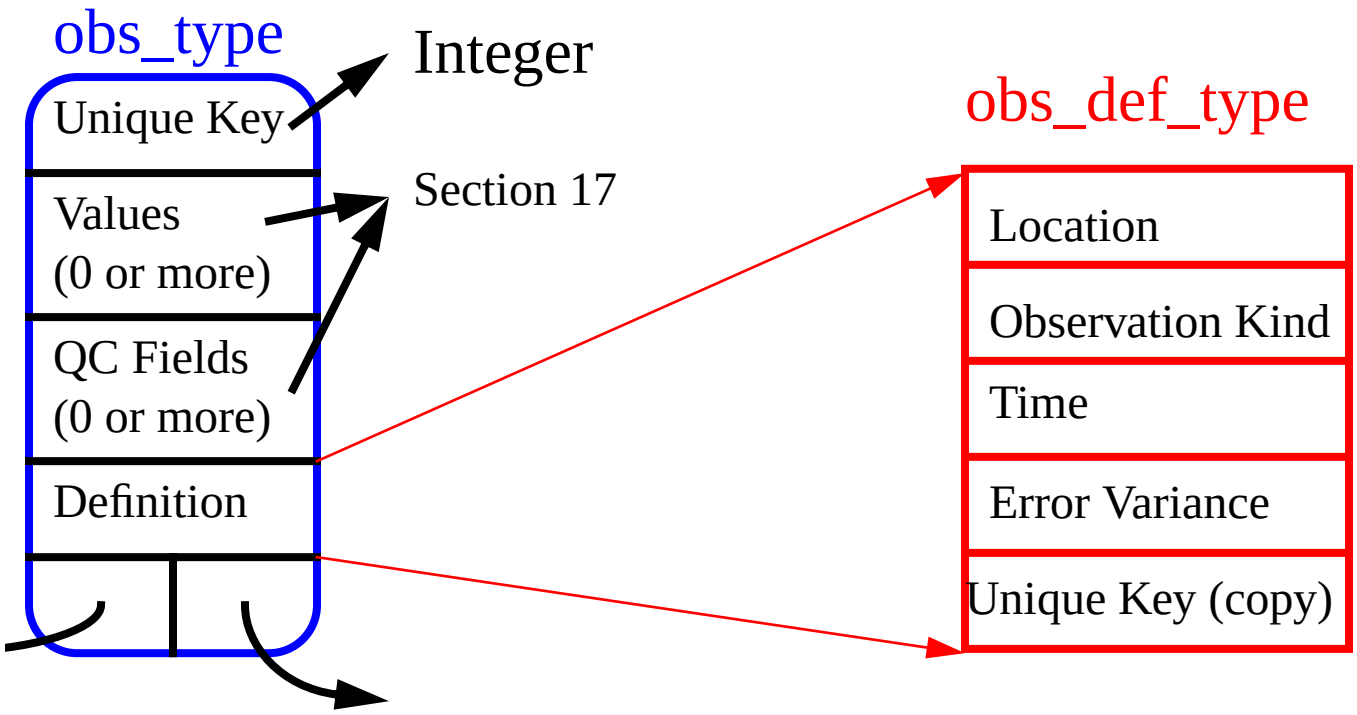
Dart assimilations are controlled by observation sequence files:

Observation sequence files contain a time-ordered list of observations.
Stored as a ‘linked list’ of observations.

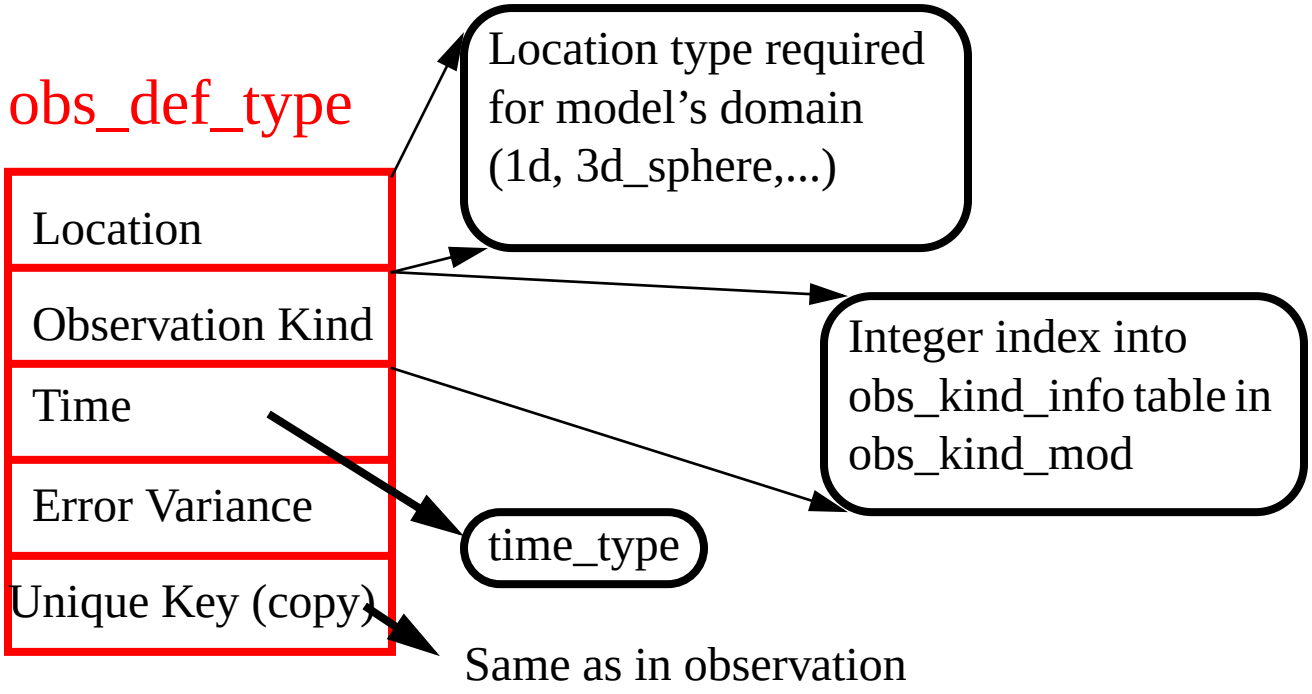


DART filter ‘assimilates’ until it runs out of observations.
Same for synthetic observation generation with perfect_model_obs.

Details of an observation type (type obs_type in obs_sequence_mod)

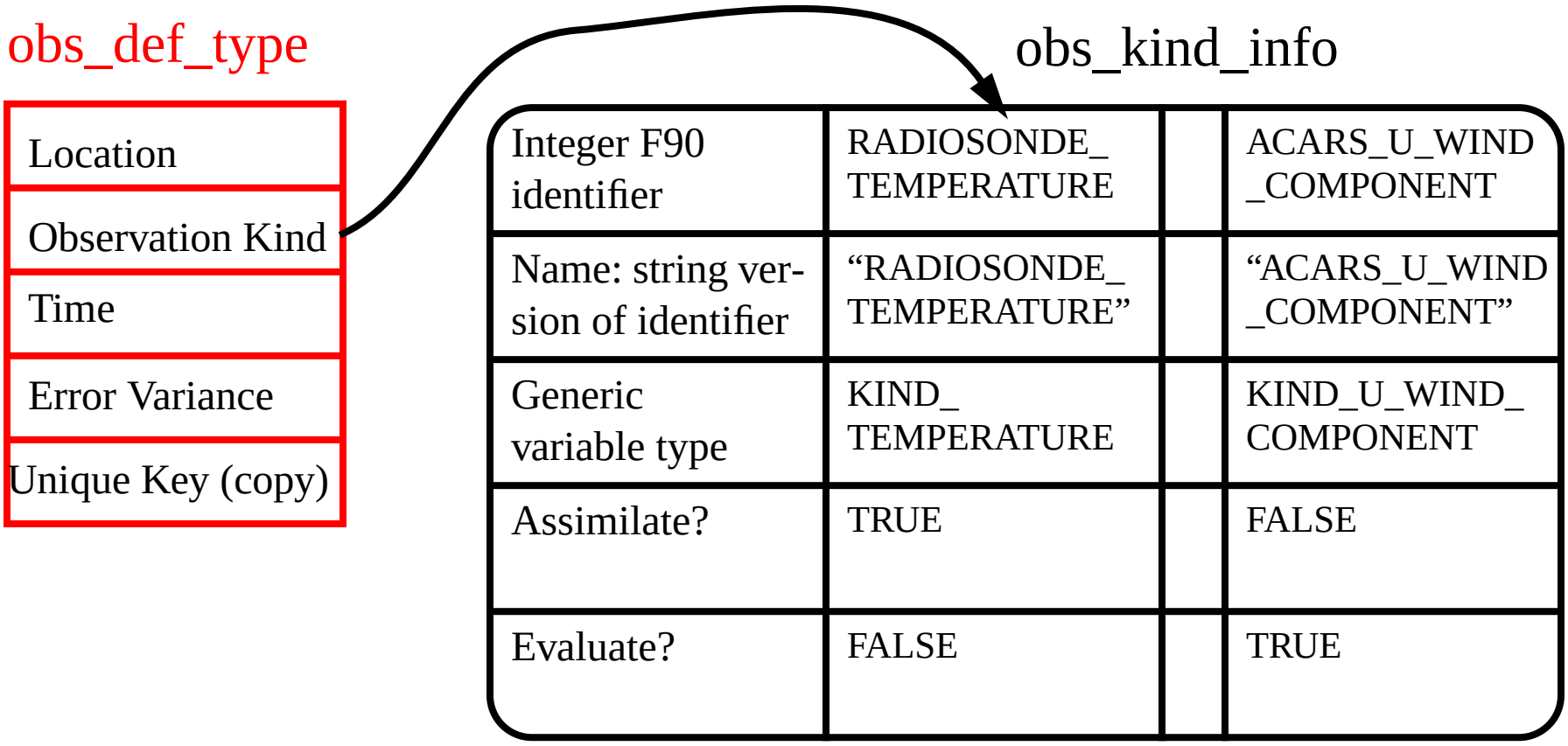


Details of observation definition (type obs_def_type in obs_def_mod)



Details of observation definition (type obs_def_type in obs_def_mod)

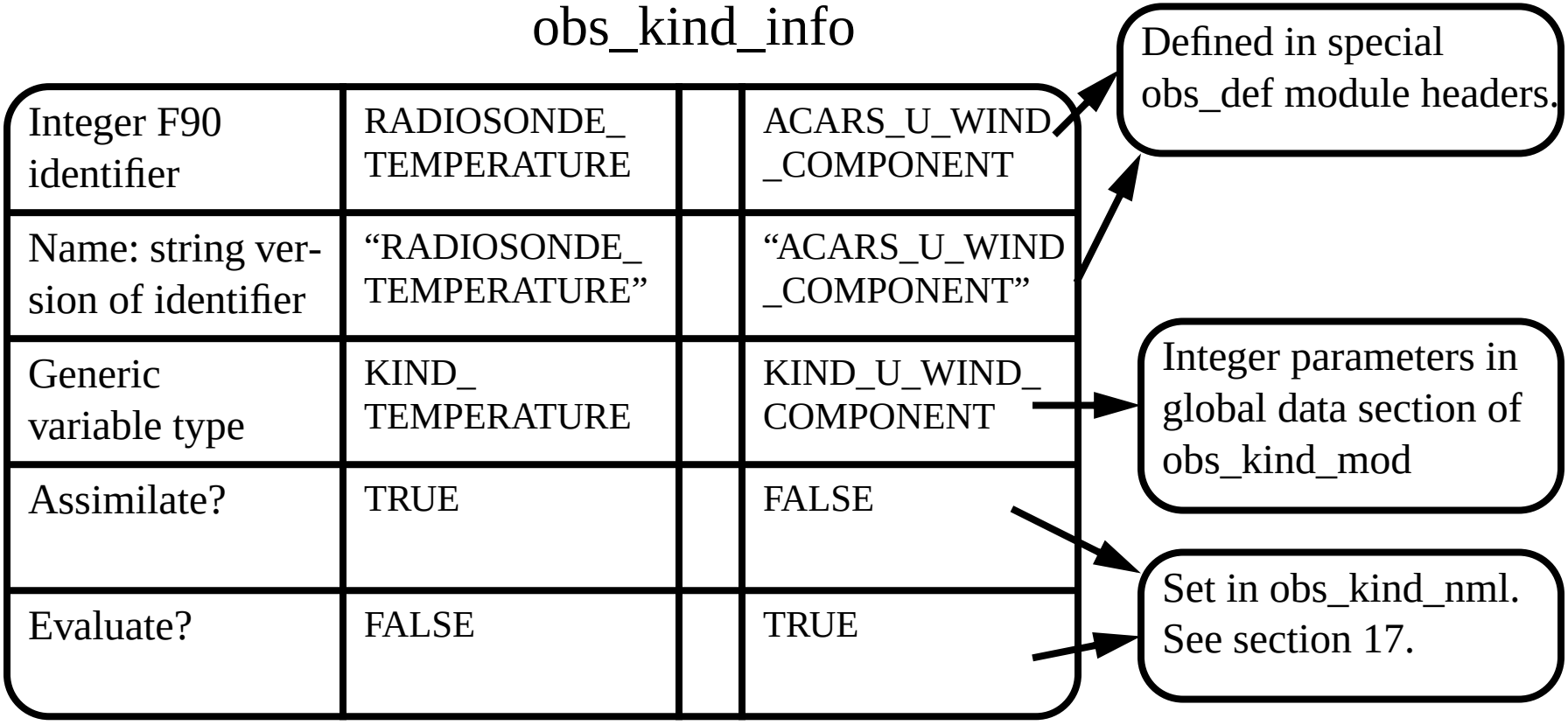
Description of obs_kind_info table.



Example: Observation is a radiosonde temperature.

The obs_kind_info table, generic types, and observation types:

obs_kind_info table built by DART preprocess program.



Radiosonde temps assimilated, forward operators only for ACARS u.

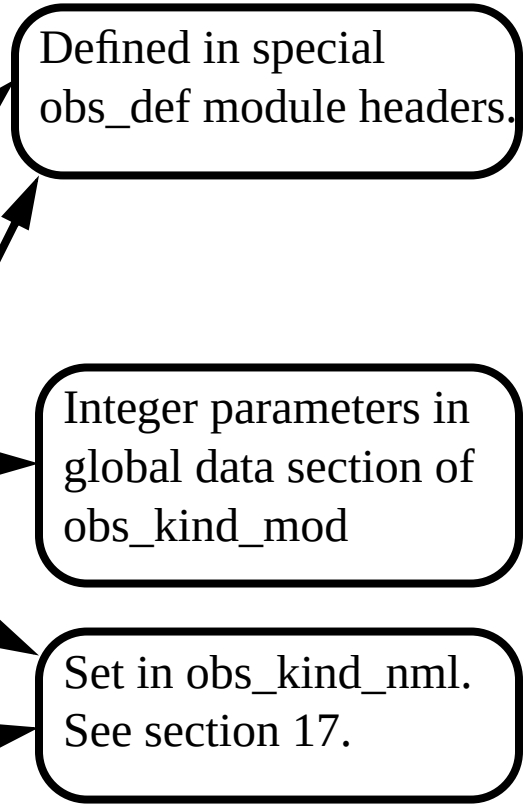
The obs_kind_info table, generic types, and observation types:

Many observation types may share a generic type.

Example: RADIOSONDE_TEMPERATURE, ACARS_TEMPERATURE...

obs_kind_info

Integer F90 identifier	RADIOSONDE_ TEMPERATURE		ACARS_U_WIND _COMPONENT
Name: string ver- sion of identifier	“RADIOSONDE_ TEMPERATURE”		“ACARS_U_WIND _COMPONENT”
Generic variable type	KIND_ TEMPERATURE		KIND_U_WIND_ COMPONENT
Assimilate?	TRUE		FALSE
Evaluate?	FALSE		TRUE



Both have generic KIND_TEMPERATURE.

Model state variables can also be associated with generic types.

The obs_kind_info table, generic types, and observation types:

Many observation types may share a generic type.

Example: RADIOSONDE_TEMPERATURE, ACARS_TEMPERATURE...

Both have generic KIND_TEMPERATURE.

Model state variables can also be associated with generic types.

Example: CAM/WRF interpolate in T field for all observation types with generic type KIND_TEMPERATURE.

Models can use the obs_kind_mod:

Have access to all generic types.

Also have access to all observation types if needed.

<u>CONFUSING generic types and observation types is common.</u>

Implementing observation definitions in DART

Give the observation type a name.
Associate the observation type with a generic type.

Four operations must be supported for each observation type:

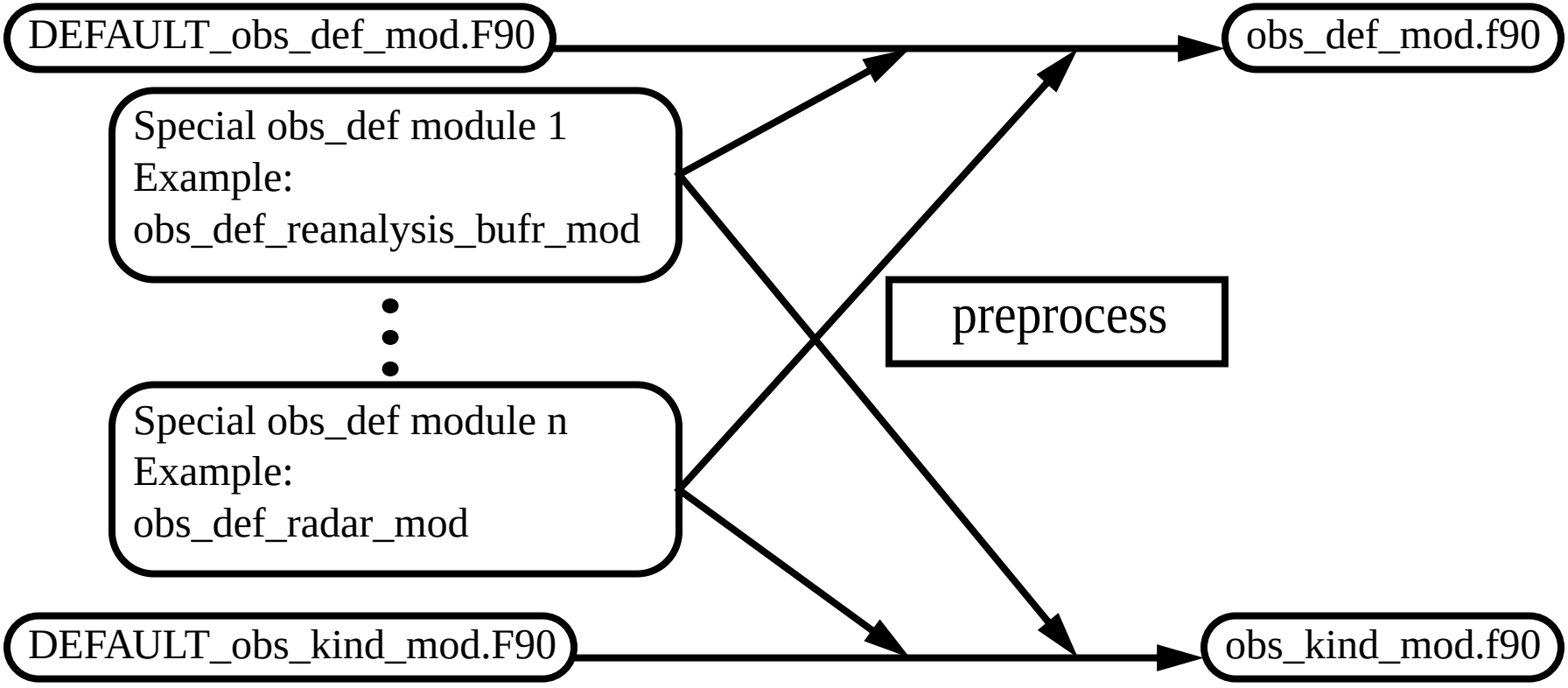
1. Compute forward operator given (extended) state vector.
2. Read any extra information not in obs_def_type from file.
   (For instance, location and beam angle for radar).
3. Write any extra information not in obs_def_type to file.
4. Get any extra information via interactive read of standard in.

This is done in a special obs_def_mod.

A special obs_def_mod is extended F90.
Contains special comments that guide the DART preprocess program.

Implementing observation definitions in DART

DART preprocess program creates obs_def_mod, obs_kind_mod

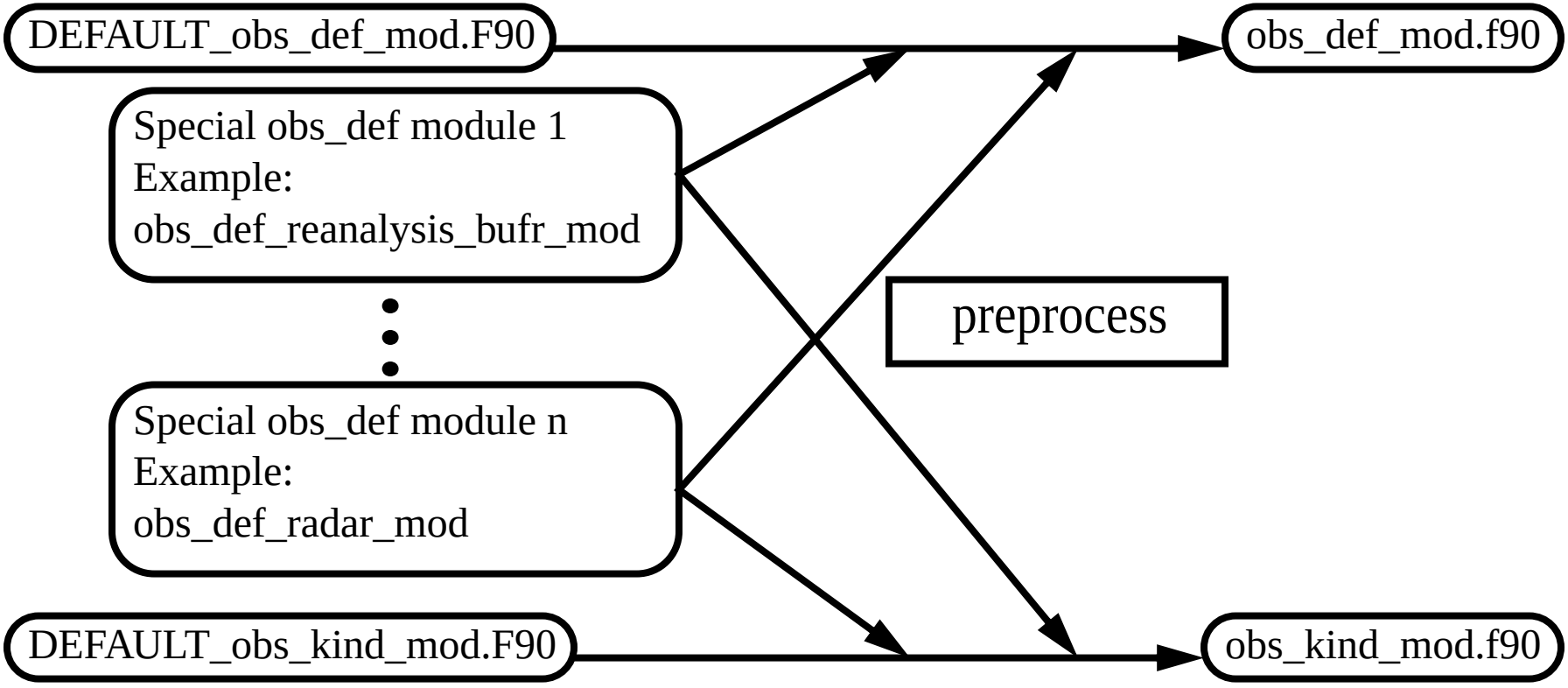


Namelist preprocess_nml lists all special obs_def modules to be used.

(Names of DEFAULT F90s and preprocessed f90s can be changed, too)

Implementing observation definitions in DART

DART preprocess program creates obs_def_mod, obs_kind_mod



If no special obs_def modules are selected, can do identity obs. only.
DEFAULT modules have special comment lines to help preprocess.

Implementing observation definitions in DART

Six special sections are required in a special obs_def_mod.
REQUIRED comment strings start and end each section.
All lines in special sections must start with F90 comment, !
Use obs_def_1d_state_vector_mod.f90 as an example here:

1. Define the observation types and associated generic types:

```
! BEGIN DART PREPROCESS KIND LIST
! RAW_STATE_VARIABLE,    KIND_RAW_STATE_VARIABLE
! RAW_STATE_1D_INTEGRAL, KIND_1D_INTEGRAL
! END DART PREPROCESS KIND LIST
```

Two observation types defined:

a. RAW_STATE_VARIABLE: generic type KIND_RAW_STATE_VARIABLE
b. RAW_STATE_1D_INTEGRAL: generic type KIND_1D_INTEGRAL

Generic kinds must be in parameter list in DEFAULT_obs_kind_mod

Implementing observation definitions in DART

Six special sections are required in a special obs_def_mod.

2. Use statements required for use of obs_def_1d_state_vector_mod

```
! BEGIN DART PREPROCESS USE OF SPECIAL OBS_DEF MODULE
!  ! Comments can be included by having a second ! at the start of the line
!  use obs_def_1d_state_mod, only : write_1d_integral, read_1d_integral, &
!                                   interactive_1d_integral, get_expected_1d_integral
! END DART PREPROCESS USE OF SPECIAL OBS_DEF MODULE
```

This special obs_def module has 4 subroutines which need to be used.

A special obs_def module can also have its own namelist if needed.

Implementing observation definitions in DART

Six special sections are required in a special obs_def_mod.

3. Case statements required to compute expected observation

```
! BEGIN DART PREPROCESS GET_EXPECTED_OBS_FROM_DEF
!       case(RAW_STATE_VARIABLE)
!       call interpolate(state, location, 1, obs_val, istatus)
!       case(RAW_STATE_1D_INTEGRAL)
!         call get_expected_1d_integral(state, location, obs_def%key, obs_val, istatus)
! END DART PREPROCESS GET_EXPECTED_OBS_FROM_DEF
```

Each observation type being defined must appear in a case.
Here, the RAW_STATE_VARIABLE observation type is a simple interpolation using the assim_model *interpolate* subroutine.

The RAW_STATE_1D_INTEGRAL is more complicated and calls the *get_expected_1d_integral* in the special obs_def module.

Implementing observation definitions in DART

Six special sections are required in a special obs_def_mod.

4. Case statements required to read extra info from obs_sequence file.

```
! BEGIN DART PREPROCESS READ_OBS_DEF
!     case(RAW_STATE_VARIABLE)
!       continue
!     case(RAW_STATE_1D_INTEGRAL)
!       call read_1d_integral(obs_def%key, ifile, fileformat)
! END DART PREPROCESS READ_OBS_DEF
```

RAW_STATE_VARIABLE requires no extra information, just continue.

(Case statement and continue required for all observation types)

RAW_STATE_1D_INTEGRAL requires extra information.

This is read with read_1d_integral subroutine.

Extra info stored in obs_def_1d_state_vector_mod, indexed by key.

Implementing observation definitions in DART

Six special sections are required in a special obs_def_mod.

5. Case statements required to write extra info from obs_sequence file.

```
! BEGIN DART PREPROCESS WRITE_OBS_DEF
!     case(RAW_STATE_VARIABLE)
!        continue
!     case(RAW_STATE_1D_INTEGRAL)
!        call write_1d_integral(obs_def%key, ifile, fileformat)
! END DART PREPROCESS WRITE_OBS_DEF
```

Same deal as for read

obs_def_1d_state_vector can read and write whatever it wants to describe the raw_state_1d_integral observation.

Only requirement is that it can read what it writes!

Implementing observation definitions in DART

Six special sections are required in a special obs_def_mod.

6. Case statements required to interactively create extra info.

```
! BEGIN DART PREPROCESS INTERACTIVE_OBS_DEF
!     case(RAW_STATE_VARIABLE)
!        continue
!     case(RAW_STATE_1D_INTEGRAL)
!        call interactive_1d_integral(obs_def%key)
! END DART PREPROCESS INTERACTIVE_OBS_DEF
```

DART uses interactive input from standard in to implement OO stuff.

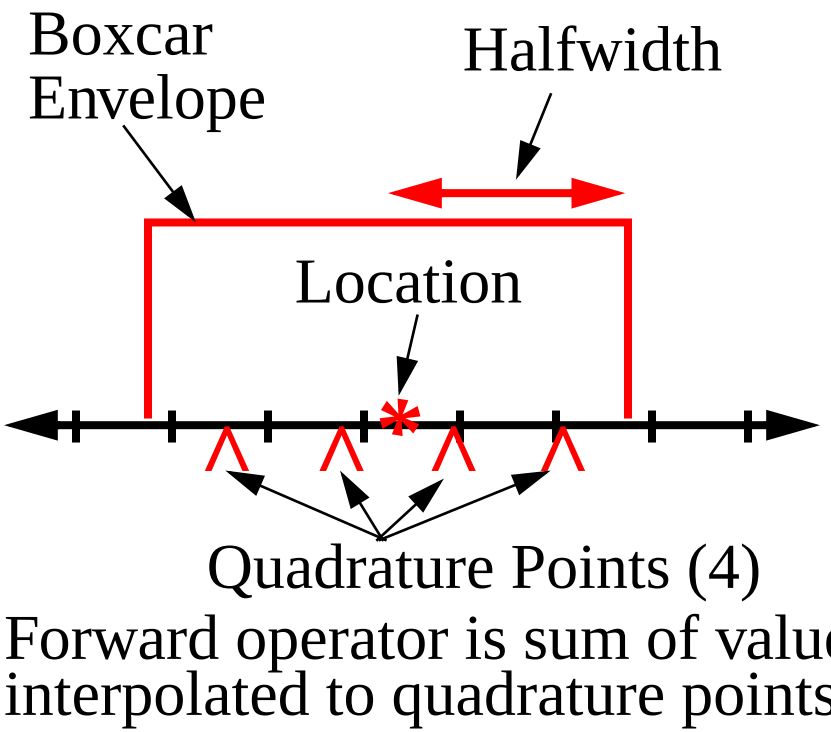
It’s nice to be able to do a keyboard create for testing.

Standard procedure: make file that drives creation (see section 17)

Implementing observation definitions in DART

What is the observation definition ‘extra information’?

obs_def_1d_state_vector_mod example.



Forward operator is sum of values interpolated to quadrature points.

raw_state_1d integral forward operator has 3 parameters:

1. Half-width of envelope,
2. Shape of envelope,
3. Number of quadrature points.

Interactive creation asks for these 3, then stores them with key.

First write outputs total number of these obs plus params for ALL.

First read reads number, params for ALL, and stores them with key.

(Could also write information for each obs separately).

Available special obs_def modules:

1. obs_def_1d_state_vector_mod: interpolations and integrals for models with one-dimensional domain, single state vector type.

2. obs_def_reanalysis_bufr_mod: All types of observations available in NCEP reanalysis bufr files. Most types from operational bufr files.

3. obs_def_metar_mod: 10 meter surface winds, 2 meter surface temperature and specific humidity, surface pressure.

4. obs_def_dew_point_mod: dew point temperature, free atmosphere or 2 meter surface.

5. obs_def_radar_mod: doppler radial velocity and reflectivity.

6. obs_def_gps_mod: GPS radio occultation refractivity.

General procedure for DART filtering with special obs_def modules

1. Compile and run preprocess: specify absolute or relative paths for all required special obs_def modules in *preprocess_nml*, *input_files*.

2. For all special obs_def modules that are ‘used’ by *obs_def_mod*
   add path to *path_names_file* for all but preprocess.

3. Compile all other required program units.

4. Select observation types to be assimilated or evaluated
   in *obs_kind_nml*.

How and where to compute forward observation operators...

Keeping models and observation definitions modular is hard.

DART recommendation: models should be able to spatially interpolate their state variables.

Forward observation operators in special obs_def modules should not expect more than this from models.

This may be too idealistic:

1. Models could do complicated forward operators for efficiency.
2. This makes it difficult to link models to DART in F90.

Different version of assim_model could help to buffer this.

Area for ongoing research.