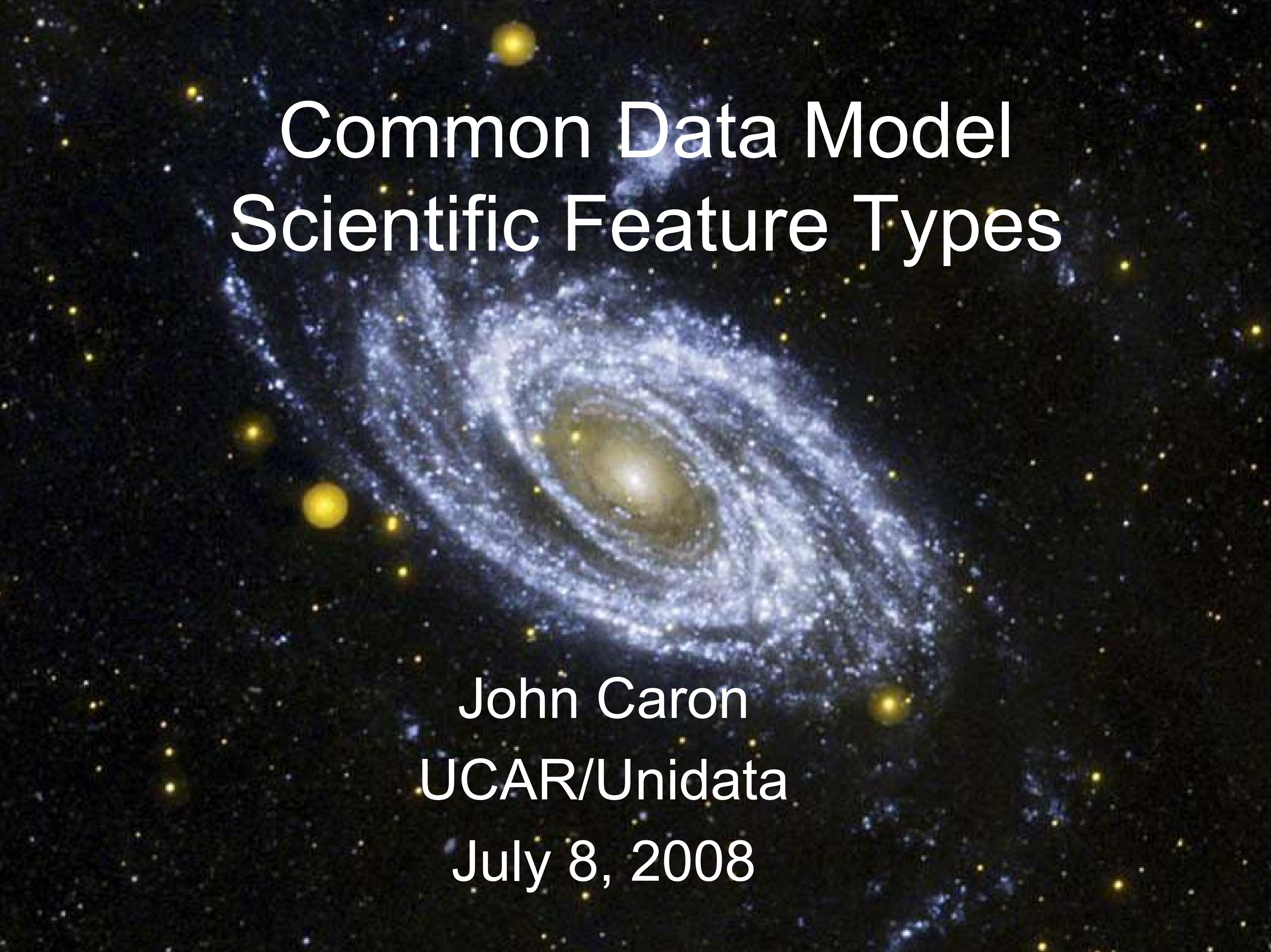


Common Data Model Scientific Feature Types

John Caron
UCAR/Unidata
July 8, 2008

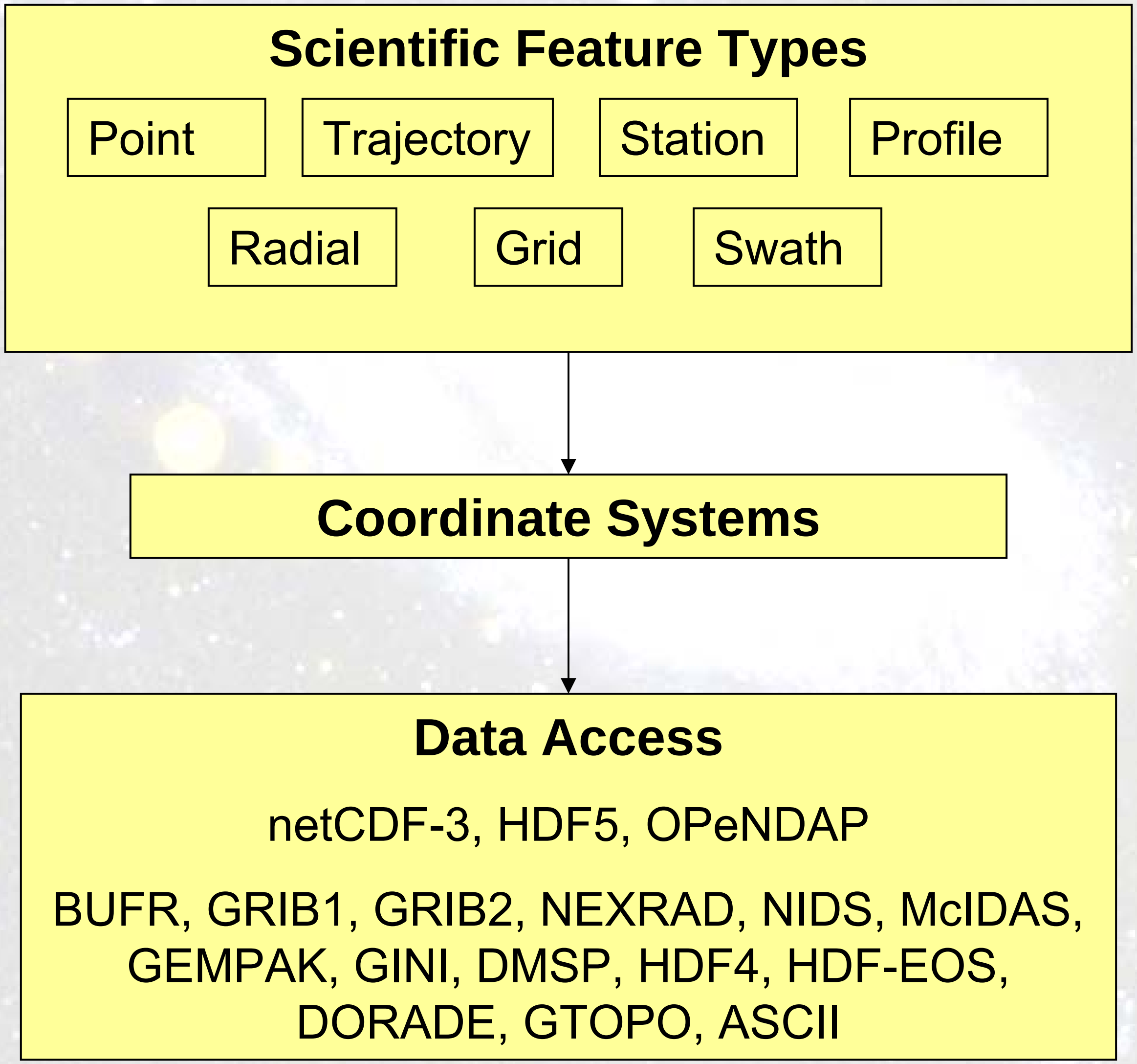
Contents

- Overview / Related Work
- CDM Feature types (focus on point data)
- Nested Table notation for Point Features
- Representing Point Data in Netcdf-3/CF
  - Preliminary experiments with BUFR

Unidata’s Common Data Model

- Abstract data model for scientific data
- NetCDF-Java library implementation/ prototype
- Features are being pushed into the netCDF-4 C library

Common Data Model



Related Standards/Models

- National and International committees are mandating compliance with ISO/OGC data standards
- Where does the CDM fit in ?


You are here

ISO/TC 211 Reference model

OGC Abstract Specification

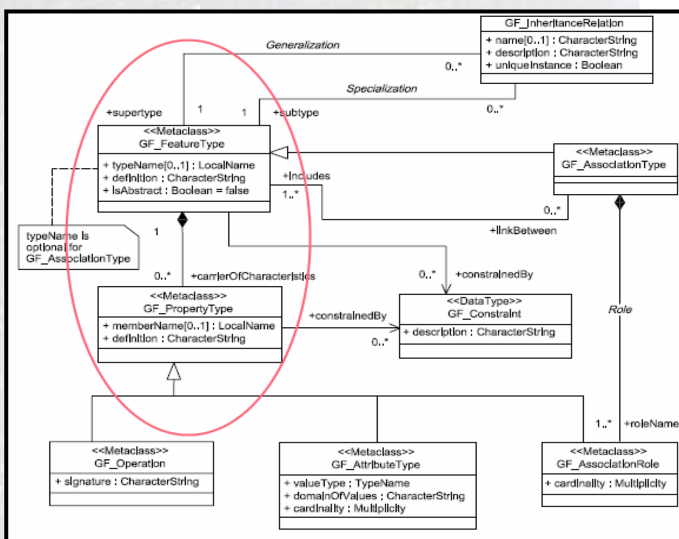
Abstract

OGC WXS = Web (MFC) Service client/server protocols

GML encoding

CSML

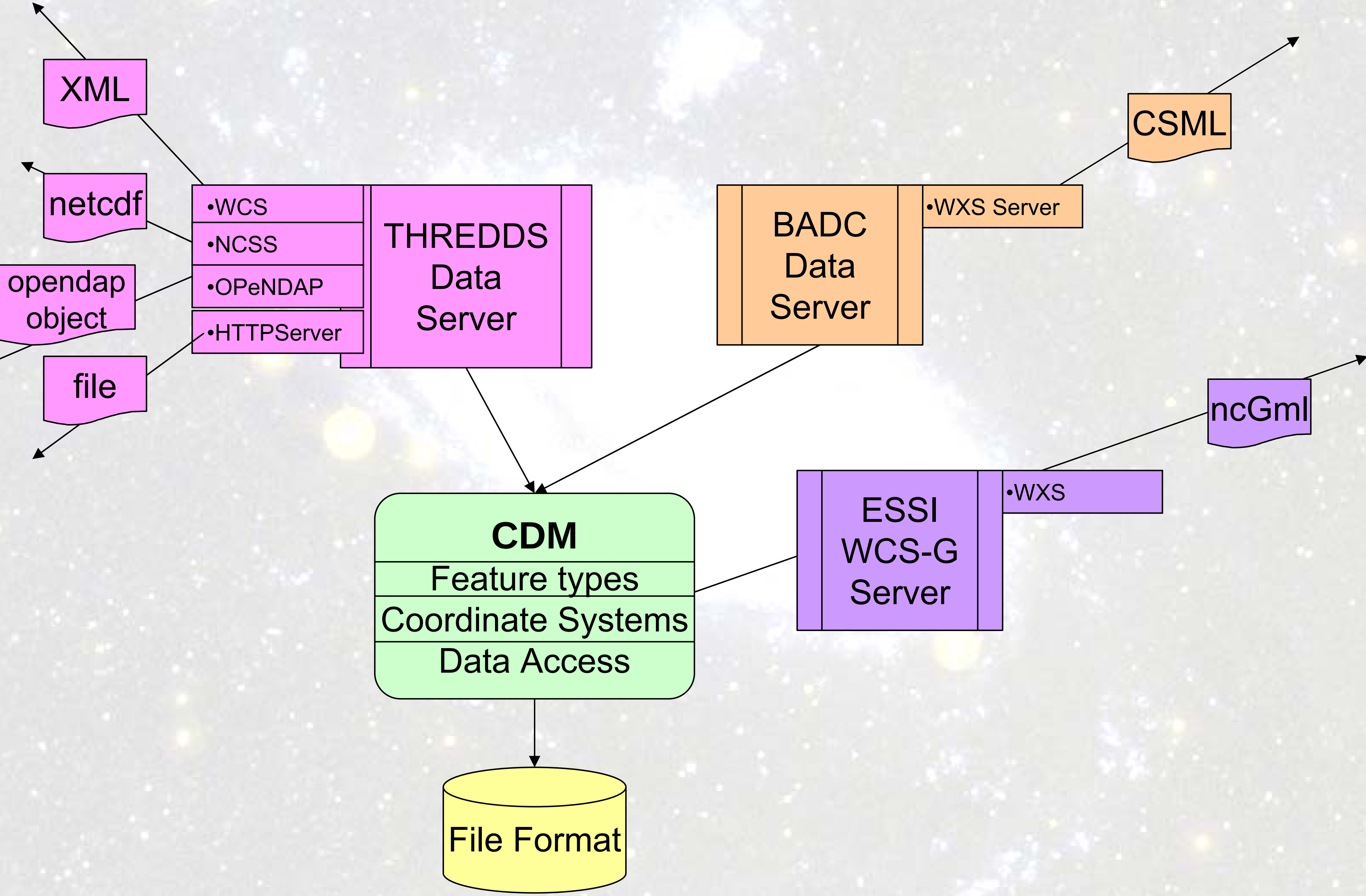
ncML-Gml



netCDF-3, HDF5, OPeNDAP, BUFR, GRIB1, GRIB2, NEXRAD, NIDS, McIDAS, GEMPAK, GINI, DMSP, HDF4, HDF-EOS, DORADE, GTOPO, ASCII

Where does CDM fit ?

- Bridge between actual datasets and abstract data model(s)
- Translate file’s “native data model” into higher-level semantic model
- “bottom-up” vs “top-down” approach

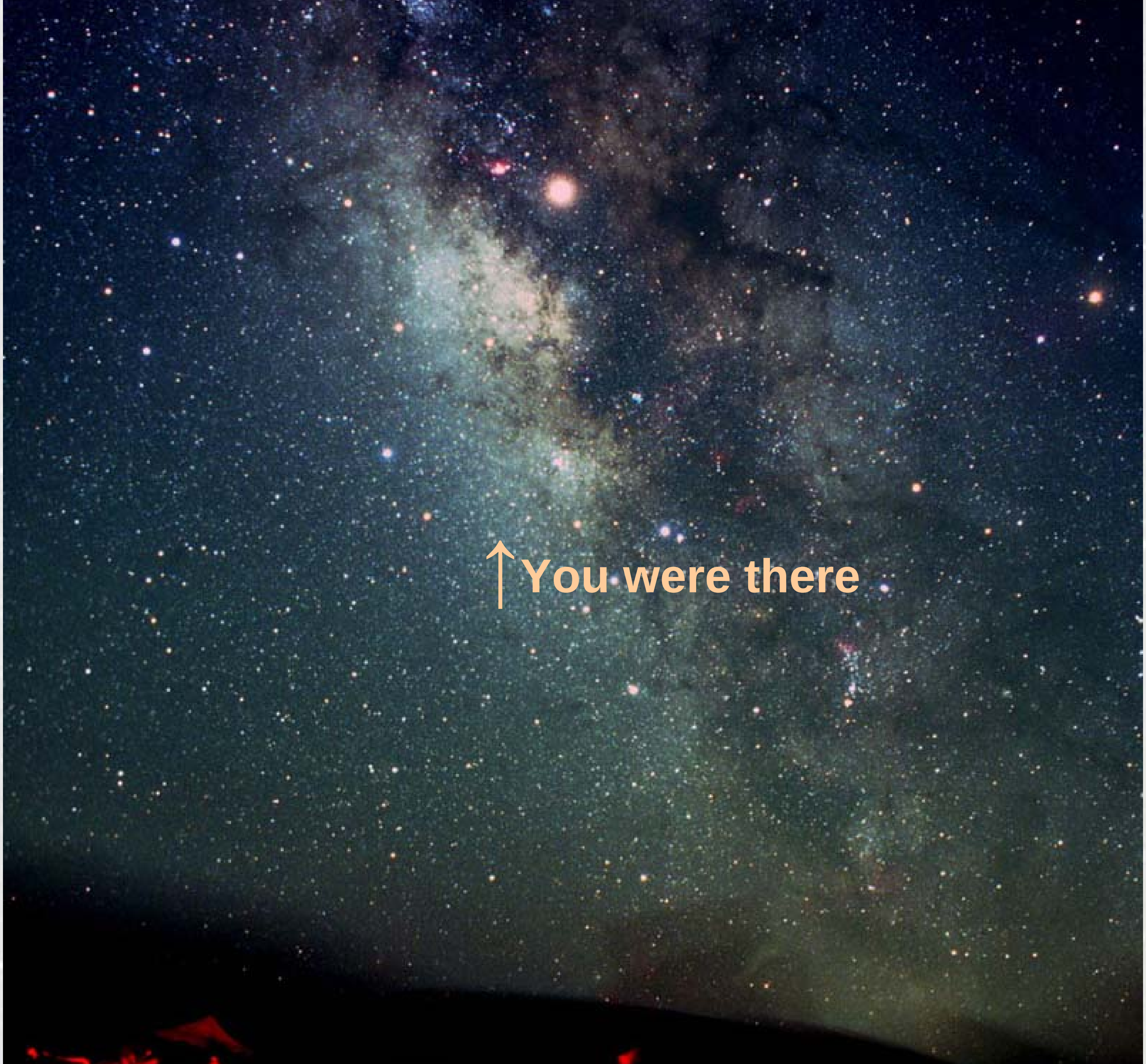

XML
netcdf
opendap object
file
•WCS
•NCSS
•OPeNDAP
•HTTPServer
THREDDS Data Server
BADC Data Server
•WXS Server
CSML
ncGml
ESSI WCS-G Server
•WXS
CDM
Feature types
Coordinate Systems
Data Access
File Format

Climate Science Modelling Language (CSML)

- British Atmospheric Data Center (BADC)
- Uses ISO/OGC semantic model
- GML application schema for atmospheric and oceanographic data

CSML - CDM Feature types

CSML Feature Type	CDM Feature Type
PointFeature	PointFeature
PointSeriesFeature	StationFeature
TrajectoryFeature	TrajectoryFeature
PointCollectionFeature	StationFeature at fixed time
ProfileFeature	ProfileFeature
ProfileSeriesFeature	StationProfileFeature at one location and fixed vertical levels
RaggedProfileSeriesFeature	StationProfileFeature at one location
SectionFeature	SectionFeature with fixed number of vertical levels
RaggedSectionFeature	SectionFeature
ScanningRadarFeature	RadialFeature
GridFeature	GridFeature at a single time
GridSeriesFeature	GridFeature
SwathFeature	SwathFeature


↑You were there

CDM Feature Types

- Formerly known as Scientific Data Types
- Based on examining real datasets “in the wild”
  - Attempt to categorize, so that datasets can be handled in a more general way
- Implementation for OGC feature services
  - Intended to scale to large, multifile collections
  - Intended to support “specialized queries”
    - Space, Time
- Data abstraction
  - Netcdf-Java has prototype implementation

Gridded Data

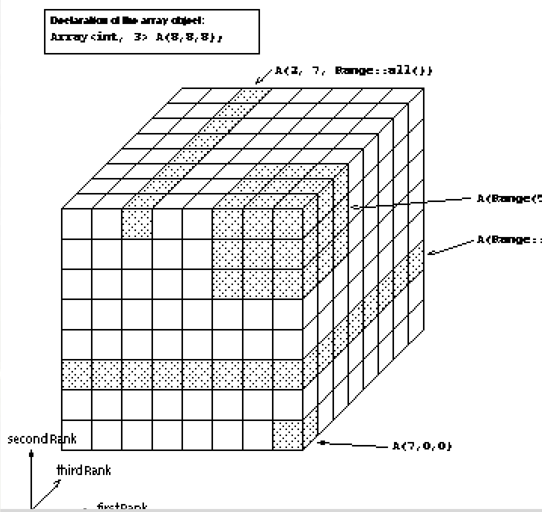
- **Grid**: multidimensional grid, separable coordinates
- **Radial**: a connected set of *radials* using polar coordinates collected into *sweeps*
- **Swath:** a two dimensional grid, *track* and *cross-track* coordinates

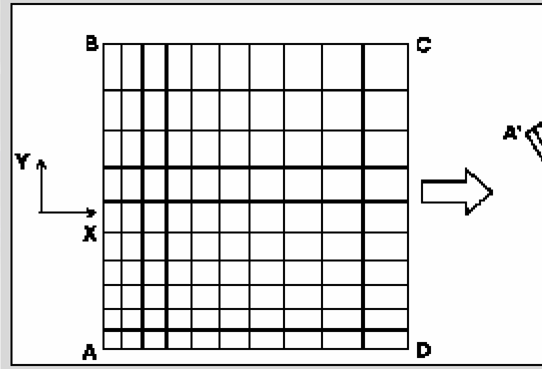
Gridded Data

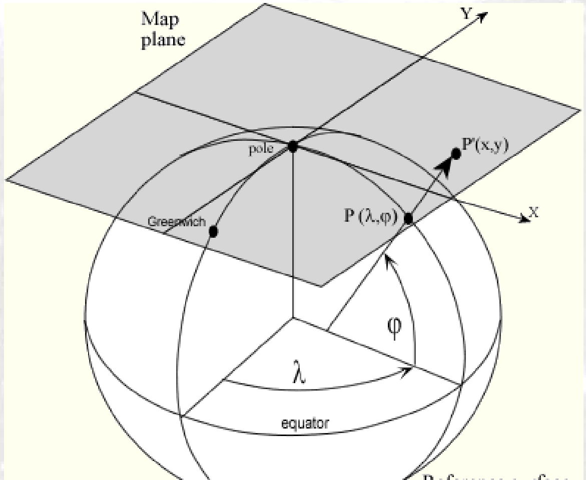
- Cartesian coordinates
- Data is 2,3,4D
- All dimensions have 1D coordinate variables (separable)

```
float gridData(t,z,y,x);
  float t(t);
  float y(y);
  float x(x);
  float z(z);

  float lat(y,x);
  float lon(y,x);
  float height(t,z,y,x);
```

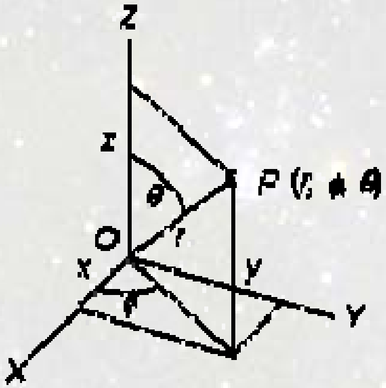






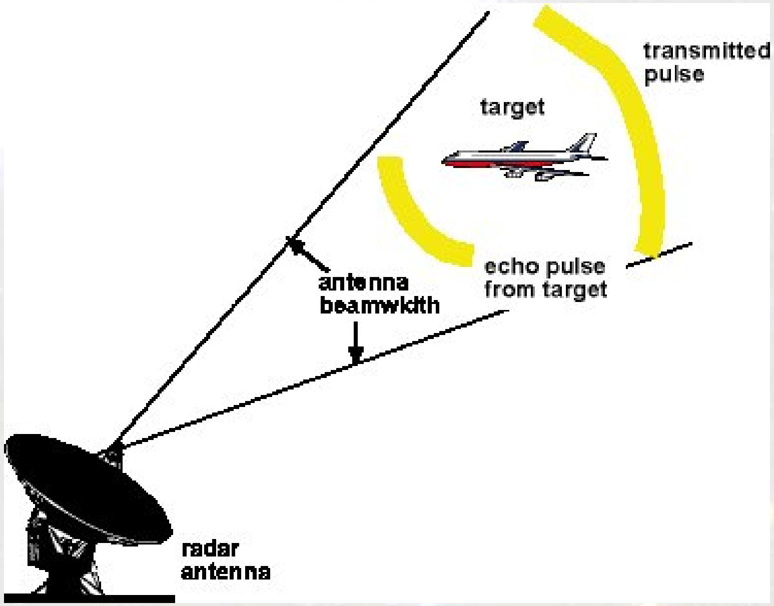
Radial Data

- Polar coordinates
- two dimensional
- Not separate time dimension

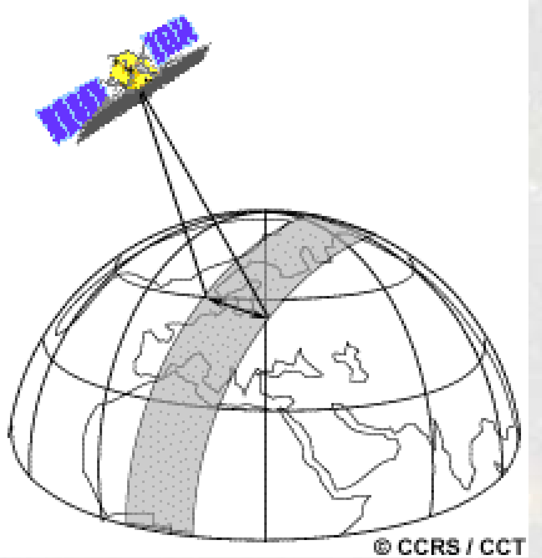


float **radialData**(radial, gate) :
  float **distance**(gate)
  float **azimuth**(radial)
  float **elevation**(radial)
  float **time**(radial)

float **origin_lat**;
float **origin_lon**;
float **origin_alt**;

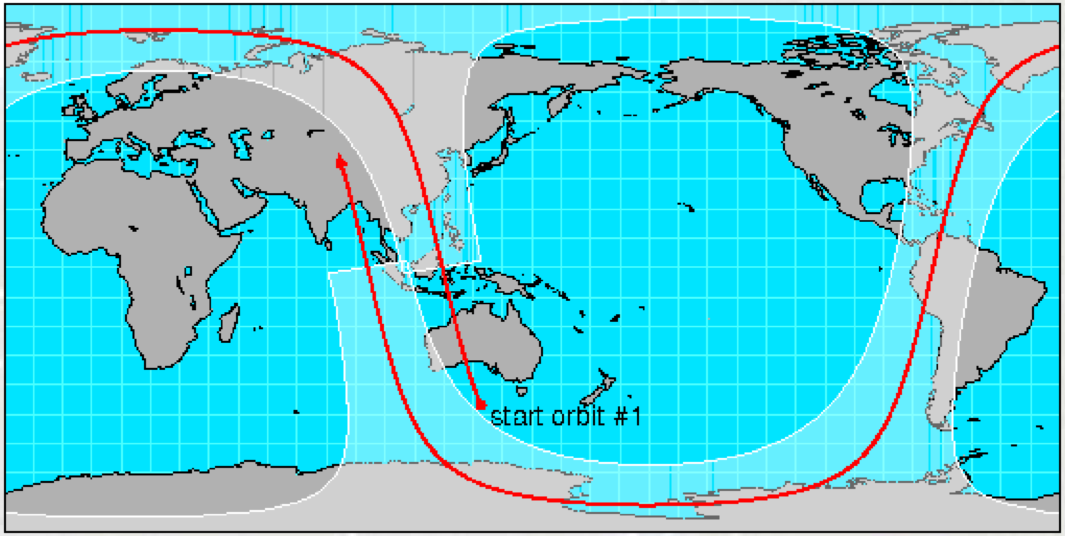


Swath



- two dimensional
- track and cross-track
- not separate time dimension
- orbit tracking allows fast search

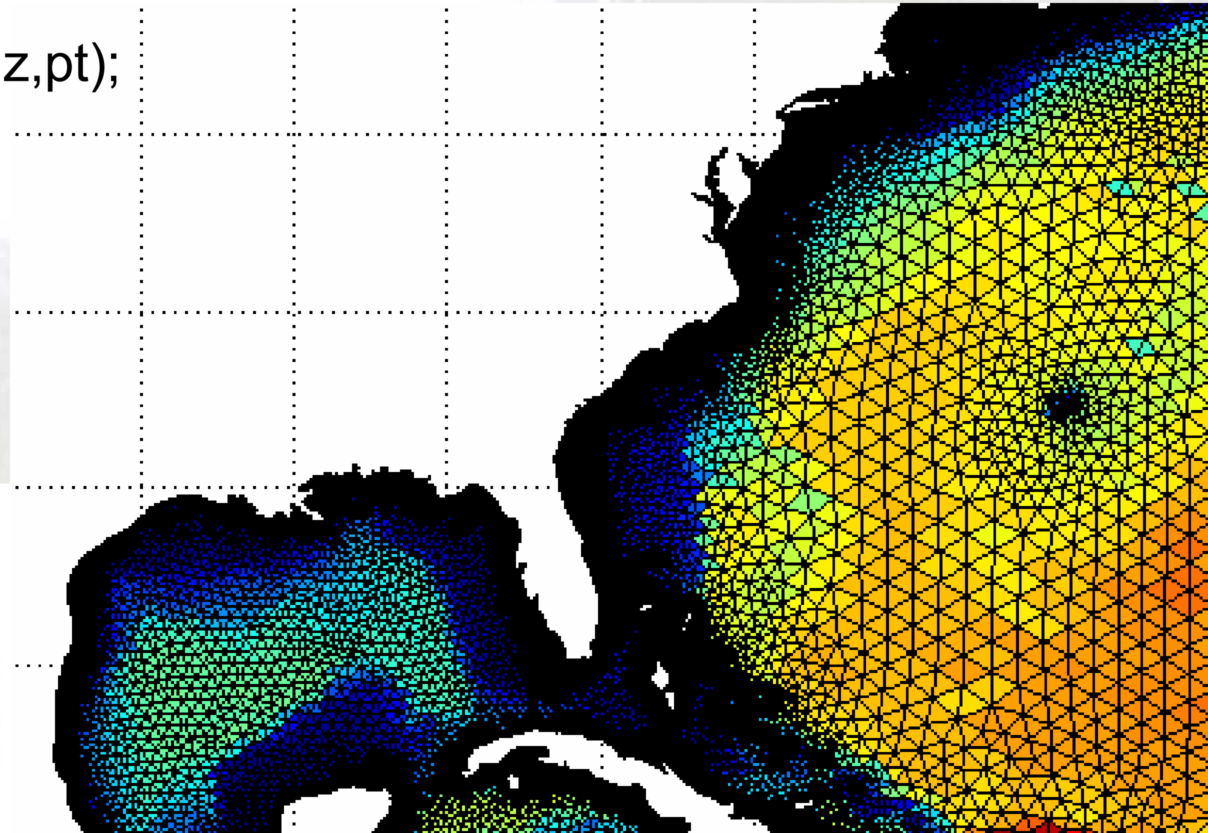
float **swathData**(track, xtrack)
  float **lat**(track, xtrack)
  float **lon**(track, xtrack)
  float **alt**(track, xtrack)
  float **time**(track)

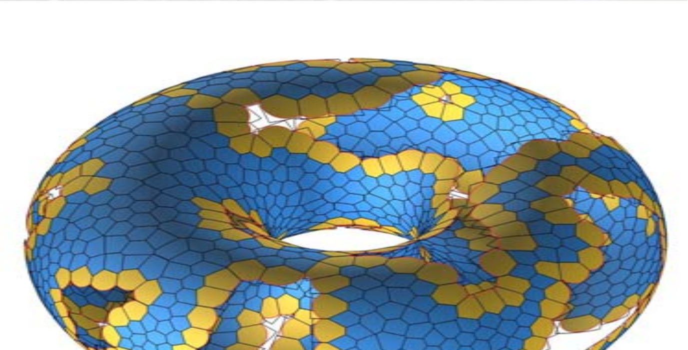


Unstructured Grid

- Pt dimension not connected
- Need to specify the connectivity explicitly
- No implementation in the CDM yet

```
float unstructGrid(t,z,pt);
  float lat(pt);
  float lon(pt);
  float time(t);
  float height(z);
```






Be here now

1D Feature Types (“point data”)

float data(sample);

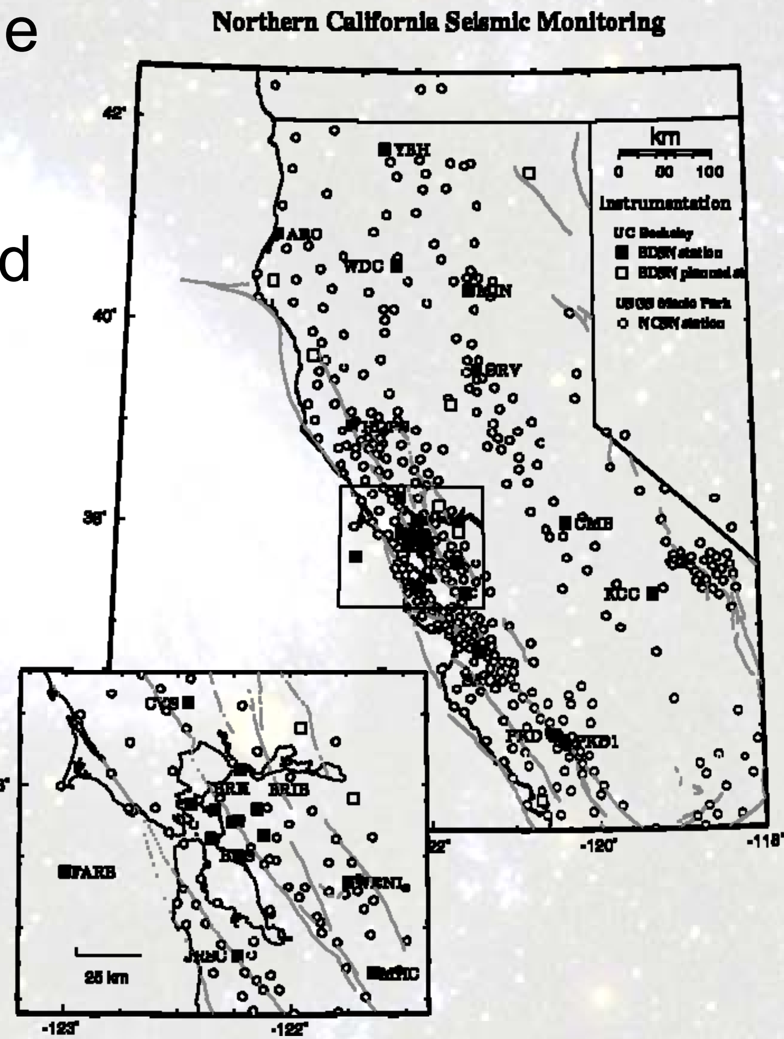
- **Point**: measured at one point in time and space
- **Station**: time-series of points at the same location
- **Profile**: points along a vertical line
- **Station Profile**: a time-series of profiles at same location.
- **Trajectory**: points along a 1D curve in time/space
- **Section**: a collection of profile features which originate along a trajectory.

Point Observation Data

- Set of measurements at the same point in space and time = obs
- Collection of obs = dataset
- Sample dimension not connected

```
float obs1(sample);
float obs2(sample);
  float lat(sample);
  float lon(sample);
  float z(sample);
  float time(sample);
```

```
Table {
  lat, lon, z, time;
  obs1, obs2, ...
} obs(sample);
```



Time-series Station Data

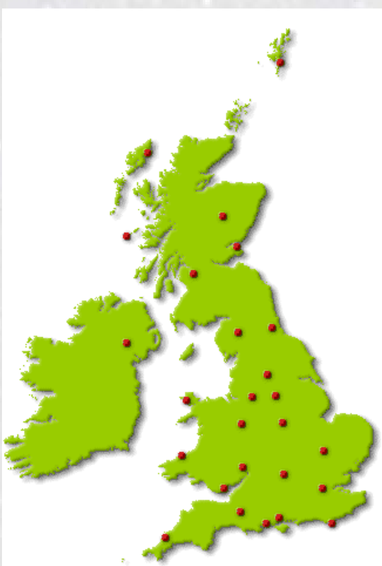
float obs1(sample);
float obs2(sample);
float lat(sample);
float lon(sample);
float z(sample);
float time(sample);

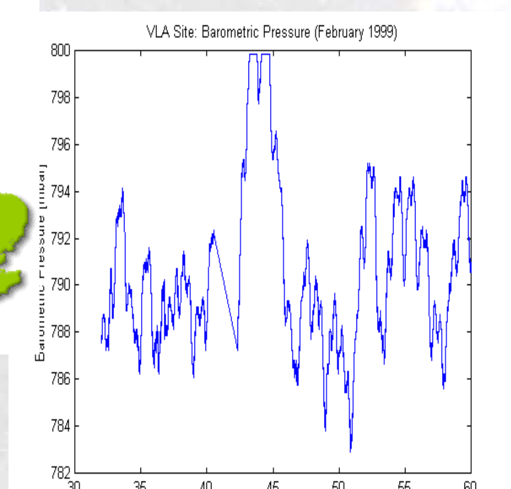
float obs1(**stn, time**);
float obs2(stn, time);
float time(stn, time);

int stationId(**stn**);
float lat(stn);
float lon(stn);
float z(stn);

float obs1(sample);
float obs2(sample);
int **stn_id**(sample);
float time(sample);

int stationId(**stn**);
float lat(stn);
float lon(stn);
float z(stn);






```
Table {
  stationId;
  lat, lon, z;
  Table {
    time;
    obs1, obs2, ...
  } obs(*); // connected
} stn(stn); // not connected
```

Profile Data

```
float obs1(sample);
float obs2(sample);
  float lat(sample);
  float lon(sample);
  float z(sample);
  float time(sample);
```

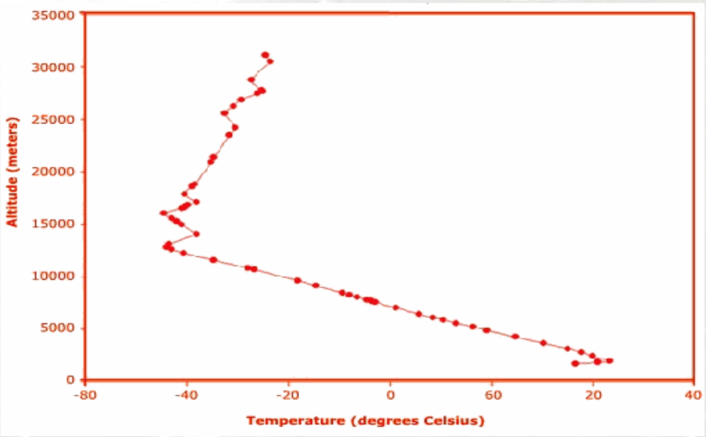
float obs1(**profile, level**);
float obs2(profile, level);
float z(profile, level);

float time(**profile**);
float lat(profile);
float lon(profile);

float obs1(sample);
float obs2(sample);
int **profile_id**(sample);
float z(sample);

int profileId(**profile**);
float lat(profile);
float lon(profile);
float time(profile);

```
Table {
  profileId;
  lat, lon, time;
  Table {
     z;
     obs1, obs2, ...
   } obs(*); // connected
} profile(profile); // not connected
```



Time-series Profile Station Data

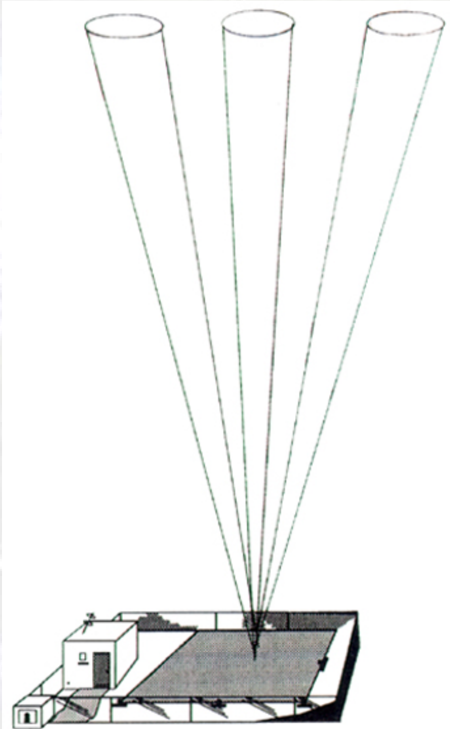
```
float obs1(profile, level);
float obs2(profile, level);
float z(profile, level);

float time(profile);
float lat(profile);
float lon(profile);
```

```
float obs1(stn, time, level);
float obs2(stn, time, level);
float z(stn, time, level);

float time(stn, time);
float lat(stn);
float lon(stn);
```

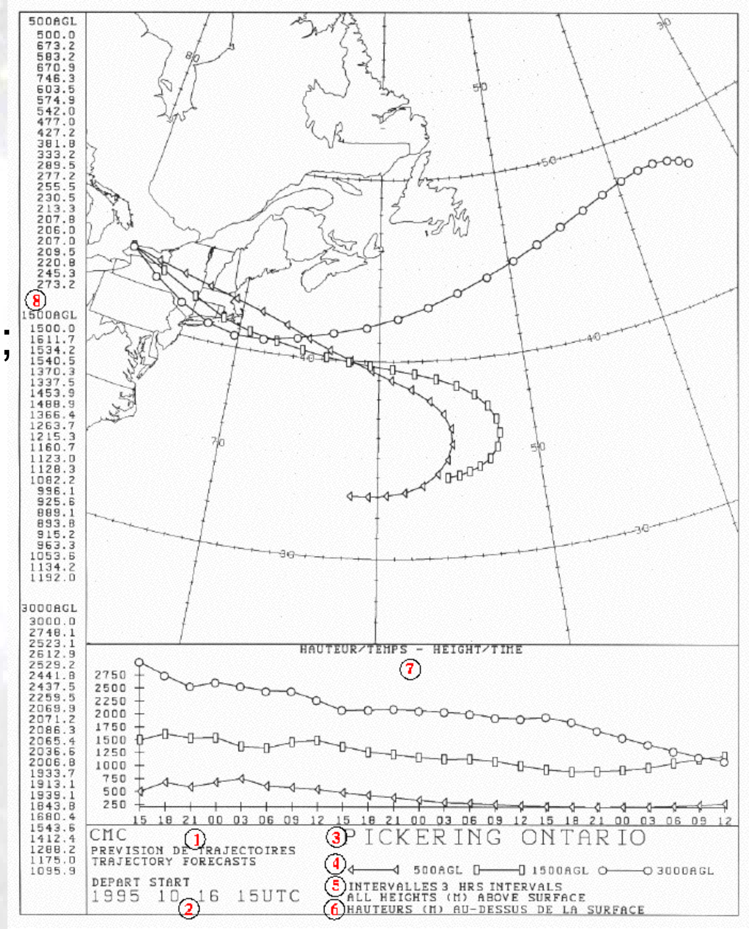
```
Table {
  stationId;
  lat, lon;
  Table {
    time;
    Table {
      z;
      obs1, obs2, ...
    } obs(*); // connected
  } profile(*); // connected
} stn(stn); // not connected
```



Trajectory Data

float obs1(sample);
float obs2(sample);
  float lat(sample);
  float lon(sample);
  float z(sample);
  float time(sample);
  int **trajectory_id**(sample);

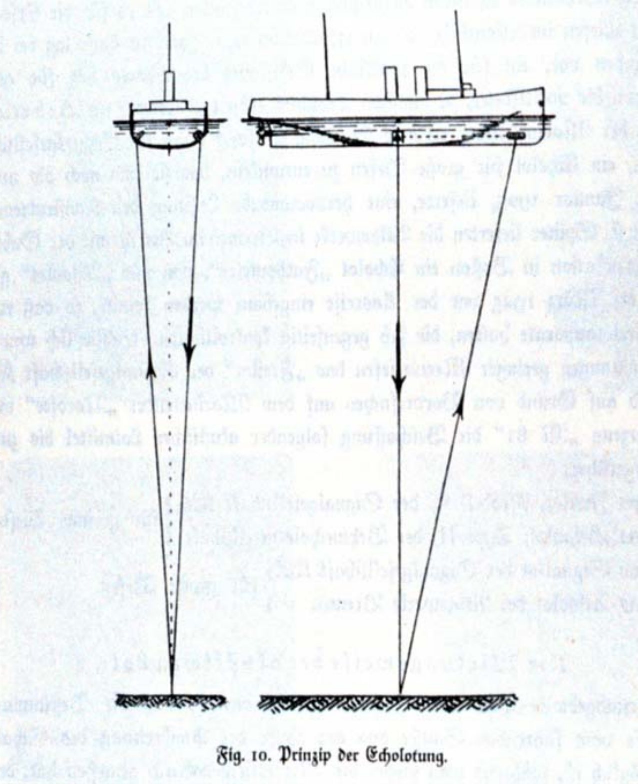
float obs1(**traj,obs**);
float obs2(traj,obs);
  float lat(traj,obs);
  float lon(traj,obs);
  float z(traj,obs);
  float time(traj,obs);
  int **trajectory_id**(traj);



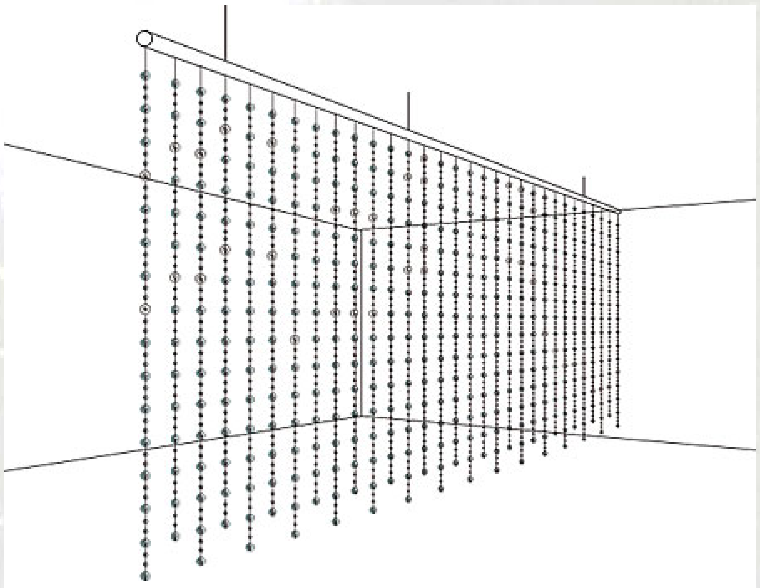

```
Table {
  trajectory_id;
  Table {
      lat, lon, z, time;
      obs1, obs2, ...
    } obs(*); // connected
  } traj(traj) // not connected
```

Section Data

float obs1(**traj,profile,level**);
float obs2(traj,profile,level);
float z(traj,profile,level);
float lat(traj,profile);
float lon(traj,profile);
float time(traj, profile);




```
Table {
  section_id;
  Table {
    surface_obs // data anywhere
    lat, lon, time
    Table {
       depth;
       obs1, obs2, ...
      } obs(*); // connected
    } profile(*); // connected
  } section(*) // not connected
```



Nested Table Notation (1)

1. A *feature instance* is a row in a table.
2. A *table* is a collection of features of the same type. The table may be fixed or variable length.
3. A nested (*child*) table is owned by a row in the *parent* table.
4. Both *coordinates* and *data variables* can be at any level of the nesting.
5. A *feature type* is represented as nested tables of specific form.
6. A *feature collection* is an unconnected collection of a specific feature type.

```
Table {
  data1, data2
  lat, lon, time;

  Table {
    z;
    obs1, obs2, ...
  } obs(17);

} profile(*);
```

Nested Table Notation (2)

- A constant coordinate can be factored out to the top level. This is logically joined to any nested table with the same dimension.

```
dim level = 17;
float z(level);

Table {
  data1, data2
  lat, lon, time;

  Table {
     obs1, obs2, ...
   } obs(level);

} profile(*);
```

Nested Table Notation (3)

- A coordinate in an inner table is connected; a coordinate in the outermost table is unconnected.

```
Table {
  stationId;
  lat, lon;
  Table {
    time;
    Table {
      z;
      obs1, obs2, ...
    } obs(*); // connected
  } profile(*); // connected
} stn(stn); // not connected
```

```
Table {
  trajectory_id;
  Table {
    lat, lon, z, time;
    obs1, obs2, ...
  } obs(*); // connected
} traj(traj) // not connected
```

```
Table {
  lat, lon, z, time;
  obs1, obs2, ...
} point(sample);
```

Relational model

- Nested Tables are a hierarchical data model (tree structure)
- Simple transformation to relational model – explicitly add join variables to tables

```
Table {
  stationId;
  lat, lon, z;

  Table {
    time;
    obs1, obs2, ...
   } obs(42);

} stn(stn);
```

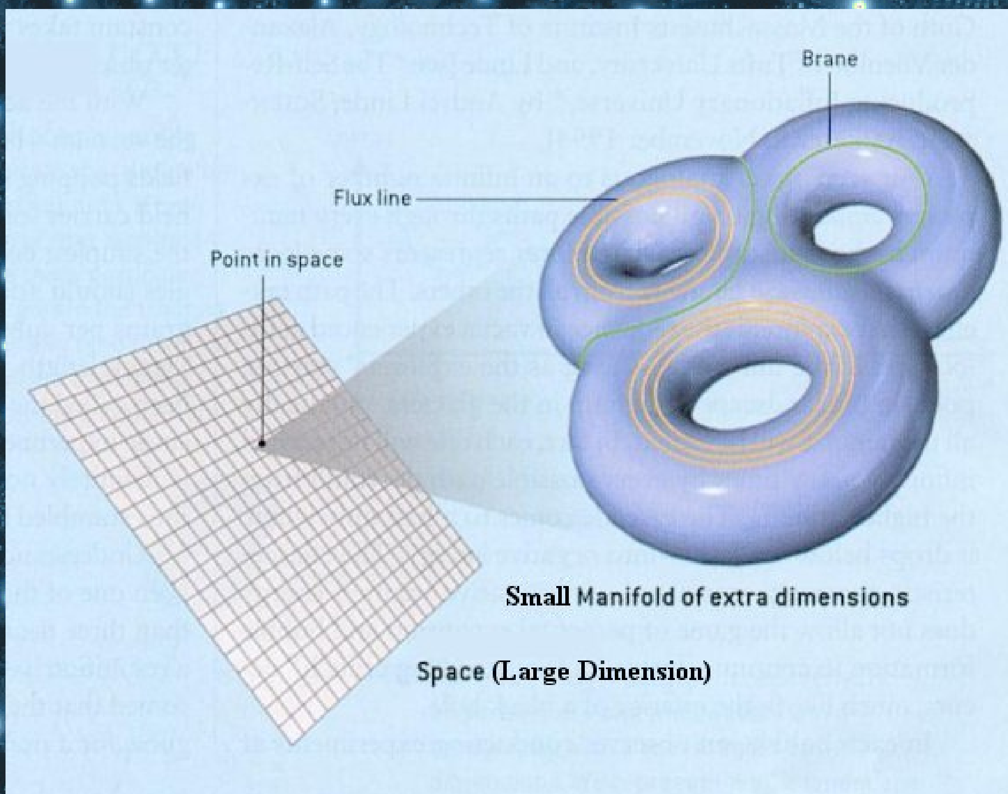
```
RTable {
  stationId // primary key
  lat, lon, z;
} stn

RTable {
  stationId // secondary key
  time;
  obs1, obs2, ...
} obs;
```

Nested Model Summary

- Compact notation to describe 1D point feature types
  - Connectivity of points is key property
  - Variable/fixed length table dimensions can be notated easily
  - Constant/varying coordinates can be easily seen
- Can be translated to relational model to get different performance tradeoffs

Representing point data in netCDF3/CF (or) Fitting data into unnatural shapes



Representing point data in NetCDF-3 / CF

- Many existing files already store point data in netCDF-3, but not standardized.
- CF Convention has 2 simple examples, no guidance for more complex situations
- Can use Nested Tables as comprehensive abstract model of data
- Look for general solutions

CF Example 1: Trajectory data

```
float O3(time) ;
  O3:coordinates = “time lon lat z" ;
double time(time) ;
float lon(time) ;
float lat(time) ;
float z(time) ;
```

Problem: what if multiple trajectories in same file?

CF Example 2: Station data

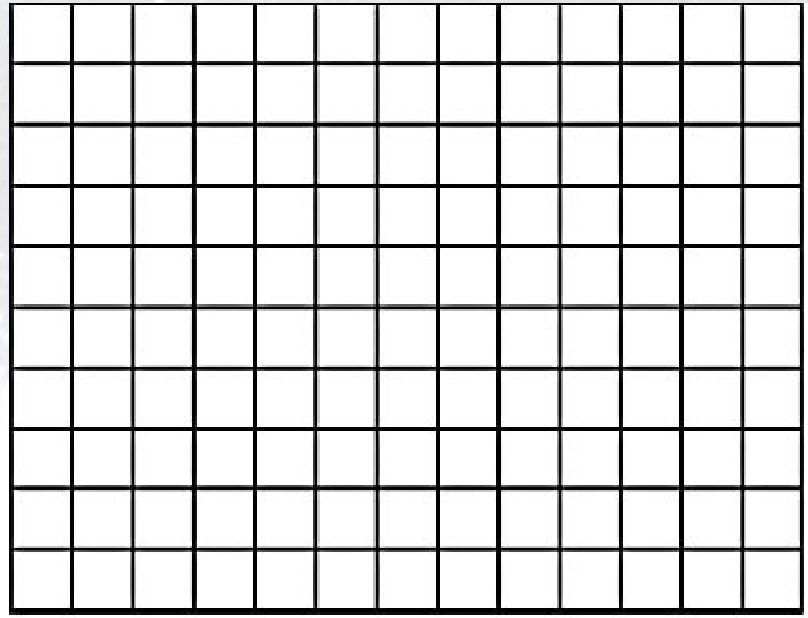
```
float data(time, station);
   data:coordinates = "lat lon alt time" ;
double time(time) ;
float lon(station) ;
float lat(station) ;
float alt(station) ;
```

If stations have different times, use

```
double time(time, station) ;
```

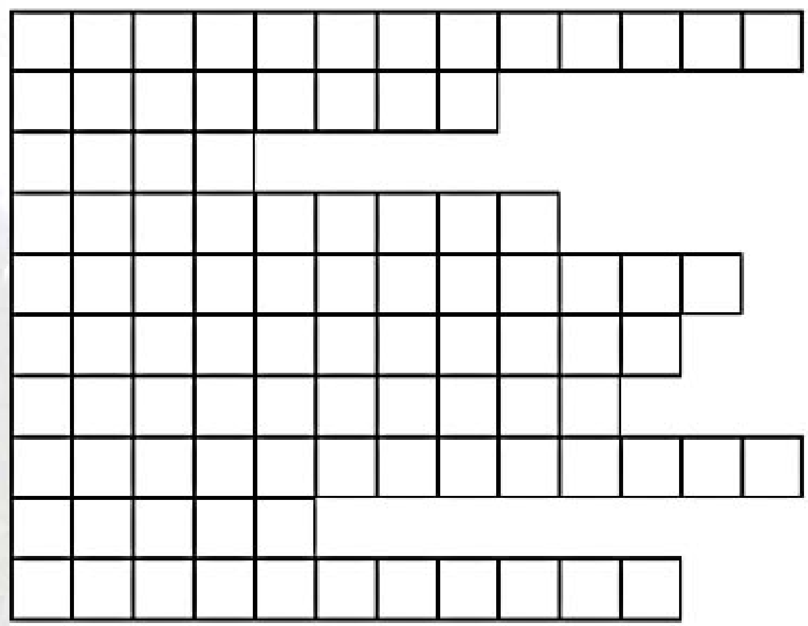
Problem: what if stations have different number of times?

Rectangular Array (netCDF-3)



A standard two-dimensional array is a rectangle.

Ragged Array



With Variant arrays, you need not waste space.

Storing Ragged Arrays

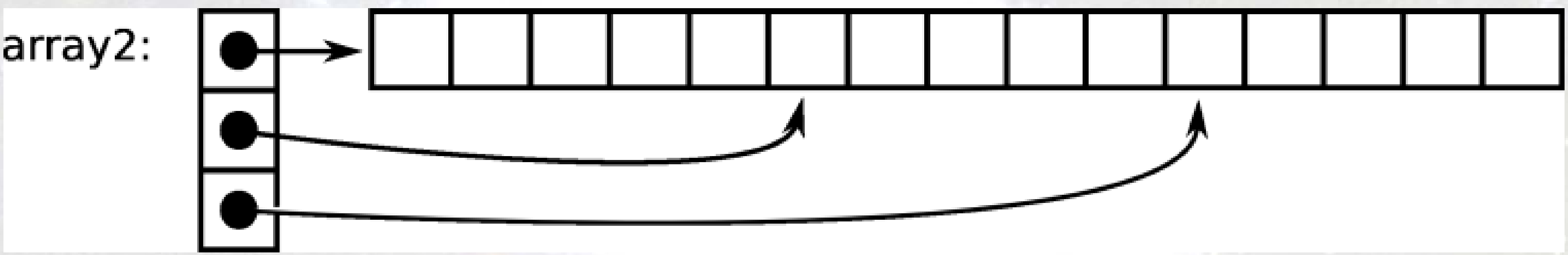
Rectangularize the Array: use maximum size of the ragged array, use missing values

- Works well if avg ~ max
- Or if you will store/transmit compressed

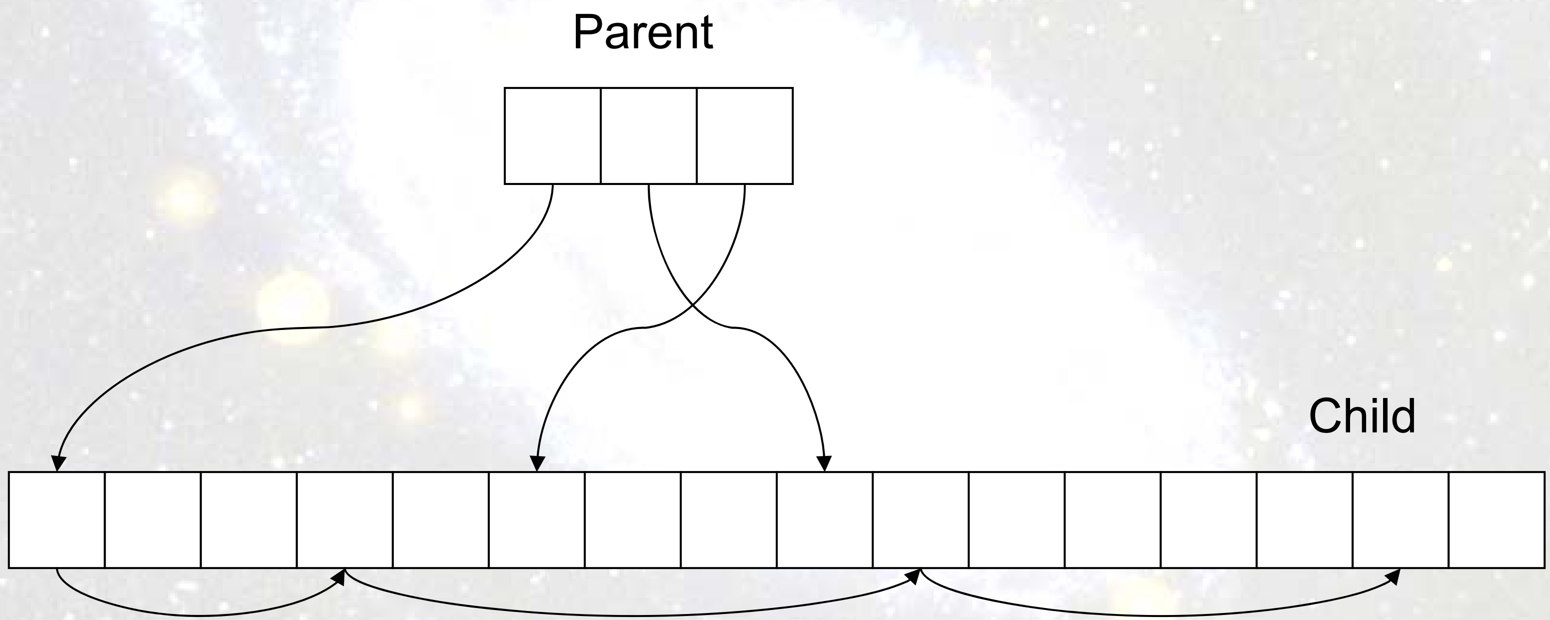
Linearize the Array: put all elements of the ragged array into a 1D array

1. Connect using index ranges
2. Connect using linked lists
3. Connect by matching field values (relational)
4. Index join

Linearize Ragged Arrays
Index Ranges



Linearize Ragged Arrays
Linked List of Indices



Linearize Ragged Arrays
Match field values (relational)

Lat	Lon	Alt	Stn
12.4	40.2	1033	KBO
77.2	-123	343	KFRC

Stn	Time	Data
KBO	12:05	32.8
KFRC	12:08	33.2
KFRC	12:13	28.9
KBO	12:13	33.8
KFRC	12:16	27.9
KFRC	12:19	19.9
KFRC	12:24	20.8
KBO	12:30	34.5

Linearize Ragged Arrays
Index Join

Lat	Lon	Alt	Stn
12.4	40.2	1033	KBO
77.2	-123	343	KFRC

Parent	Time	Data
1	12:05	32.8
2	12:08	33.2
2	12:13	28.9
1	12:13	33.8
2	12:16	27.9
2	12:19	19.9
2	12:24	20.8
1	12:30	34.5

Nested Model ➔ netCDF

1. Nested Table ➔ Pseudo-Structures

```
Table {
  profileId;
  lat, lon, time;

  Table {
     z;
     obs1, obs2, ...
   } obs(*);

} profile(profile);
```

➔

```
dimensions:
  profile = 42;
  obs = 714;

variables:
  int profileId(profile);
  float lat(profile);
  float lon(profile);
  float time(profile);

  float z(obs);
  float obs1(obs);
  float obs2(obs);
```

➔

Storing Ragged Arrays

Multidimensional / Rectangular

```
dimensions:
  profile = 42;
  levels = 17;

variables:
  float lat(profile);
  float lon(profile);
  float time(profile);

  float z(profile,level);
  float obs1(profile,level);
  float obs2(profile,level);
```

Relational

```
dimensions:
  profile = 42;
  obs = 2781;

variables:
  int profileId(profile);
  float lat(profile);
  float lon(profile);
  float time(profile);

  float z(obs);
  float obs1(obs);
  float obs2(obs);
  int profile(obs);
```

Index Join

```
dimensions:
  profile = 42;
  obs = 2781;

variables:
  float lat(profile);
  float lon(profile);
  float time(profile);

  float z(obs);
  float obs1(obs);
  float obs2(obs);
  int profileIndex(obs)
```

Storing Ragged Arrays

Index Range:

```
dimensions:
  profile = 42;
  obs = 2781;

variables:
  float lat(profile);
  float lon(profile);
  float time(profile);
  int firstObs(profile)
  int numObs(profile)

  float z(obs);
  float obs1(obs);
  float obs2(obs);
```

Linked List:

```
dimensions:
  profile = 42;
  obs = 2781;

variables:
  float lat(profile);
  float lon(profile);
  float time(profile);
  int firstObs(profile)

  float z(obs);
  float obs1(obs);
  float obs2(obs);
  int nextChild(obs)
```

Link + Parent:

```
dimensions:
  profile = 42;
  obs = 2781;

variables:
  float lat(profile);
  float lon(profile);
  float time(profile);
  int firstObs(profile)

  float z(obs);
  float obs1(obs);
  float obs2(obs);
  int nextChild(obs)
  int profileIndex(obs)
```

Case Study: BUFR

- WMO standard for binary point data
- Table driven
- Variable length
- Motherlode/IDD feed
  - 150K messages, 5.5M obs, 1 Gbyte per day
  - 350 categories of WMO headers
  - 70 distinct BUFR types

BUFR ➔ netCDF-3

- BUFR data is stored as unsigned ints
  - scale/offset/bit widths stored in external tables
  - bit packed
  - Variable-length arrays of data
- Translate to netCDF:
  - Align data on byte boundaries
  - Use standard scale/offset attributes
  - rectangularize or linearize ragged arrays

Profiler BUFR data

uncompressed, variable # of levels

	Size(Kb)	Zipped	ratio raw	ratio zip
BUFR	79.7	22.0		
NetCDF multidim	104.6	17.9	1.3	.81
netCDF linear	95.0	17.7	1.2	.80

Compressed BUFR data

fixed length nested tables

EUMETSAT, single level upper air
15 messages, 430 obs/message

	Size Kb	Zip Kb
BUFR	173	152
NetCDF	1914	145
ratio	11	.95

NCEP, satellite sounding
73 messages, 60 obs/message

	Size Kb	Zip Kb
BUFR	1291	1227
NetCDF	3550	1749
ratio	2.75	1.42

Point data in netCDF-3 Summary

- Main problem is ragged arrays
- Tradeoffs
  - ease-of-writing vs. ease-of-reading
  - storage size
  - More studies with BUFR data
- NetCDF-4 is likely straightforward, since it has variable length Structures
- CF proposal Real Soon Now

NetCDF-Java library 4.0 Point Feature API

- NetCDF-Java library 4.0 will have a new API based on Nested Table model
- New *Sequence* data type = variable length array of Structures
  - Iterators over StructureData objects
- Experimenting with:
  - Automatic analysis of datasets to guess feature type
  - Annotate/configure Feature Dataset to identify nested tables and coordinates (push into NcML?)
  - NcML aggregation over feature collections (?)

Conclusions

- CDM Feature Type model and implementation are evolving
- Nested Table notation provides a flexible way to characterize 1D point datasets
- Netcdf-Java 4.0 library has refactored point data implementation
- TDS will eventually provide new point subsetting services

Recent new documents

CDM Feature Types

CDM Point Feature Types

http://www.unidata.ucar.edu/software/netcdf-java/CDM/

Feedback:

- ***caron@ucar.edu***