Unidata's Common Data Model and NetCDF Java Library API Overview

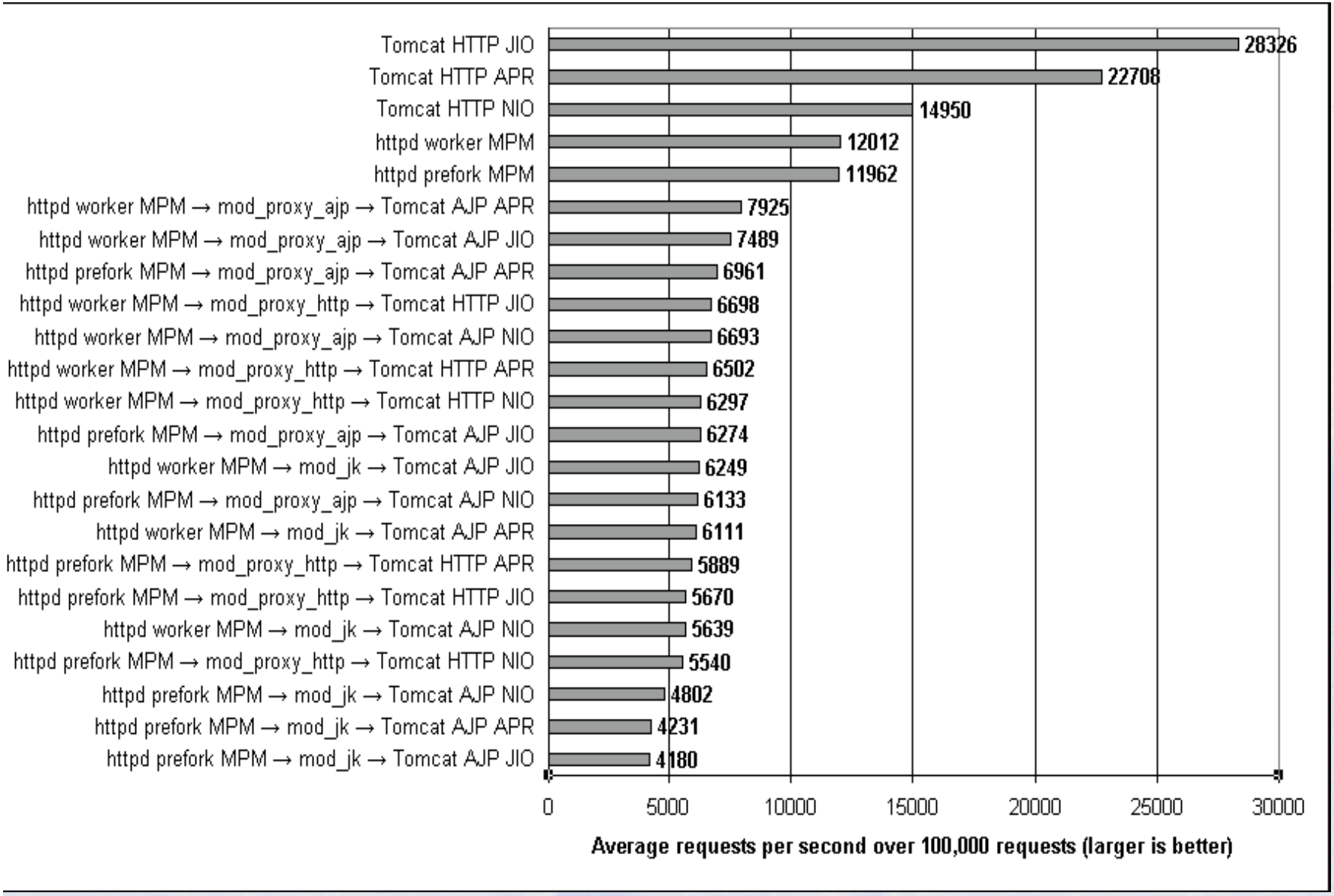
John Caron

Unidata/UCAR

Nov 2008

Java = Programmer Productivity

- Portability
- Object Oriented
- Libraries everywhere
- Thriving open source development
- Strong typing (aka type safety)
  - needed for large development projects
- Good tools: IDEs, debuggers, profilers
- Very productive
- Java is faster than C for some applications
  - eg multithreaded server



Tomcat: The Definitive Guide, Jason Brittain (O'Reilley 2007)

Java Virtual Machine / Operating Systems

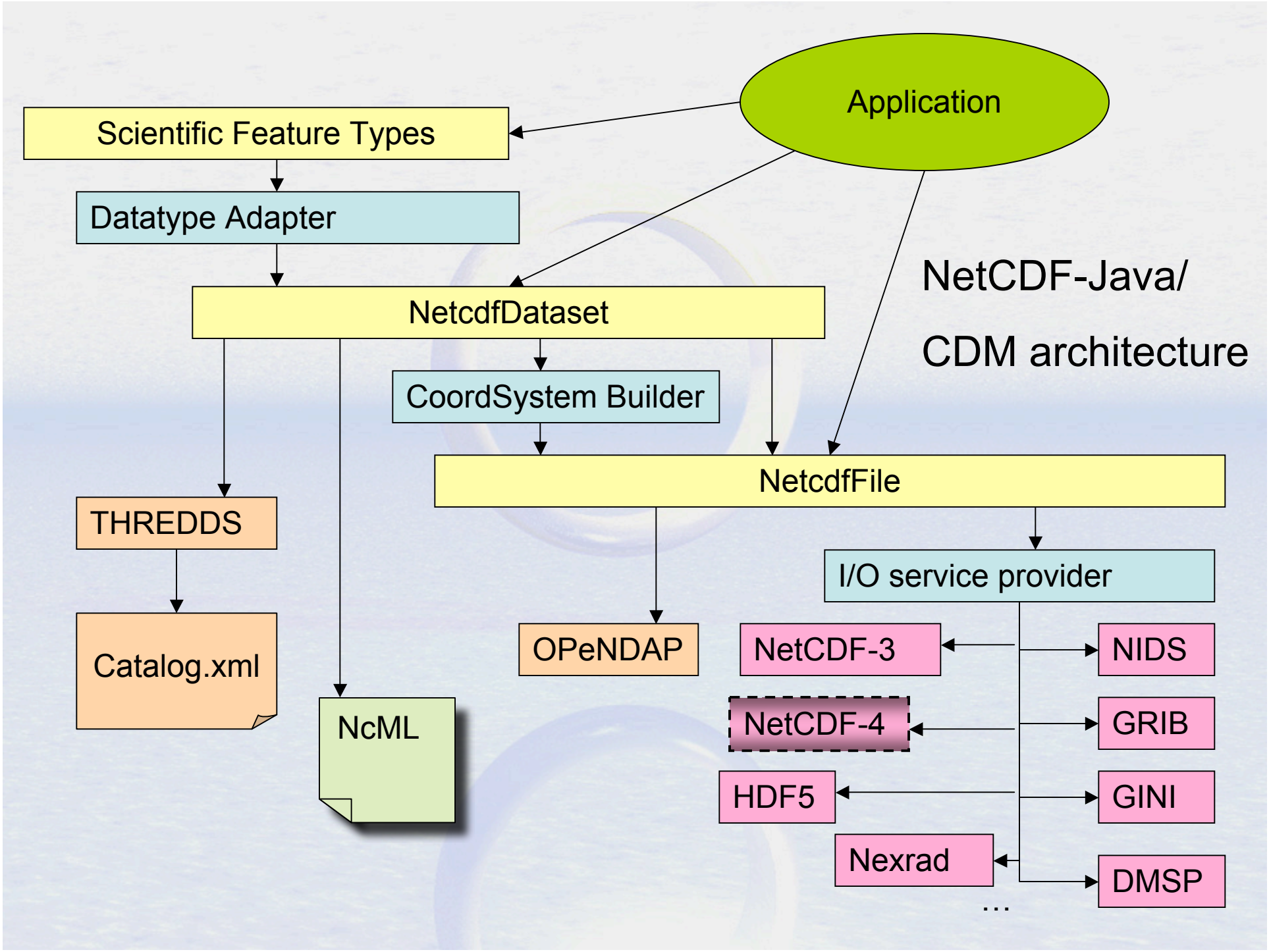
- JVM options
  - Linux, Solaris, Windows (Sun)
  - Mac OS X (Apple)
  - AIX, Linux, Windows, z/OS (IBM)
  - HP-UX (Hewlett-Packard)

Java Negatives

- Linking Java with C/Fortran apps is difficult
- Arguably not suitable for large scale numerical computation
  - Type safety, array safety, strict reproducibility
  - Multicore CPU challenge could shift
- Specialized languages can be more productive

NetCDF-Java library

- 100% Java
- Open Source (LGPL, MIT)
- Independent implementation
- Used as a component in other software (partial)
  - Integrated Data Viewer, THREDDS Data Server (Unidata)
  - Panoply (NASA)
  - ncBrowse (EPIC/NOAA)
  - Java NEXRAD Viewer (NCDC/NOAA)
  - MyWorld GIS (Northwestern)
  - EDC for ArcGIS, ERRDAP (SFSC/NOAA)
  - Live Access Server (PMEL/NOAA)
  - ncWMS (Reading)
  - Matlab plug-in (USGS)


Application
Scientific Feature Types
Datatype Adapter
NetcdfDataset
CoordSystem Builder
NetcdfFile
NetCDF-Java/
CDM architecture
THREDDS
Catalog.xml
NcML
OPeNDAP
I/O service provider
NetCDF-3
NetCDF-4
HDF5
Nexrad
…
NIDS
GRIB
GINI
DMSP

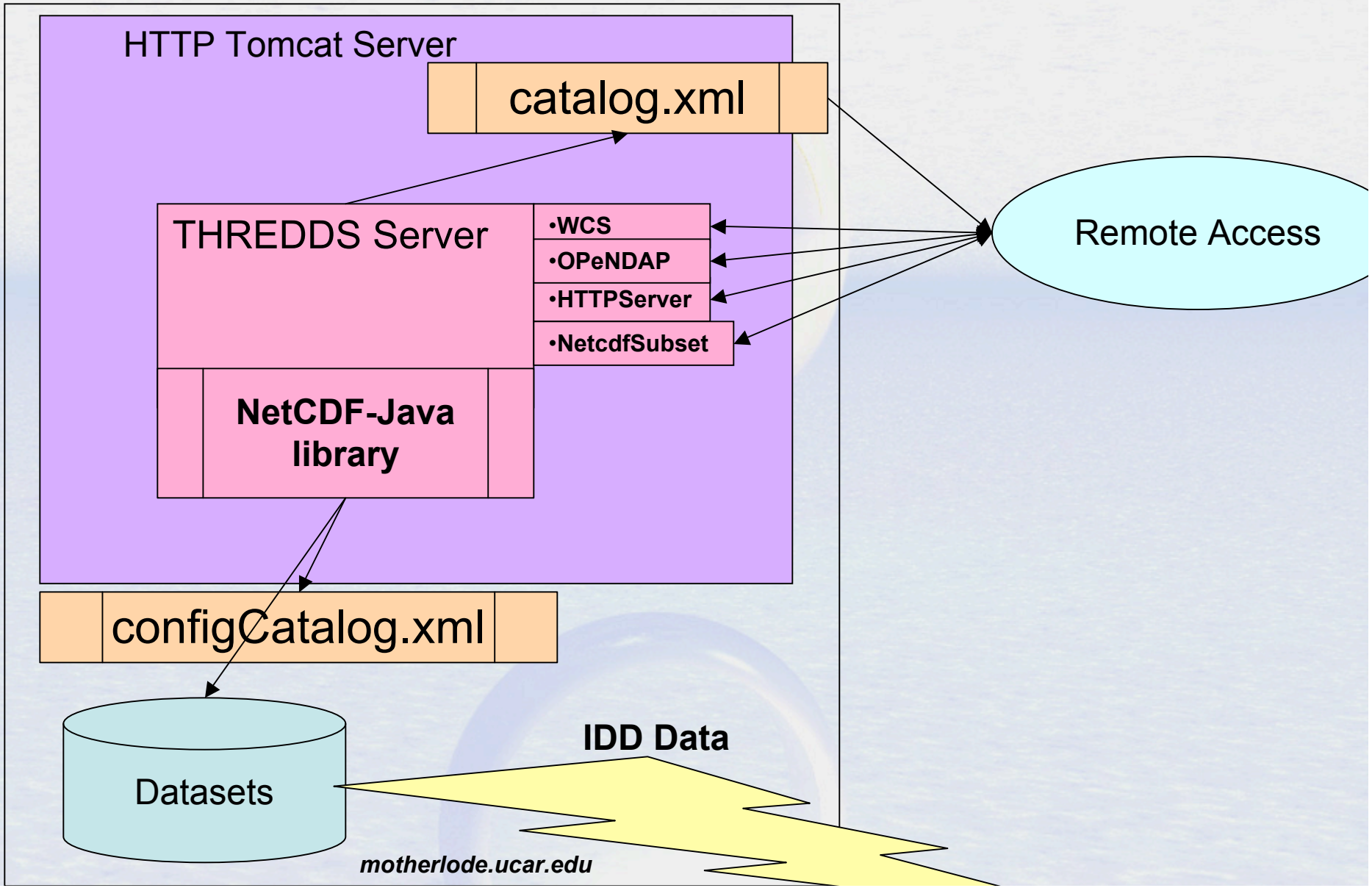
NetCDF Java Release Plans

- Current Stable Release NetCDF-Java 2.2
  - Maintenance, bug fixes only
- Development version 4.0
  - Extensive refactor, enhance performance
  - Extended data types for NetCDF4
  - Sequences : variable length Structures
  - *Scientific Feature Types* refactor
  - *Nested Tables* abstract model for point features (point, station, trajectory, profile)
  - By the end of the year

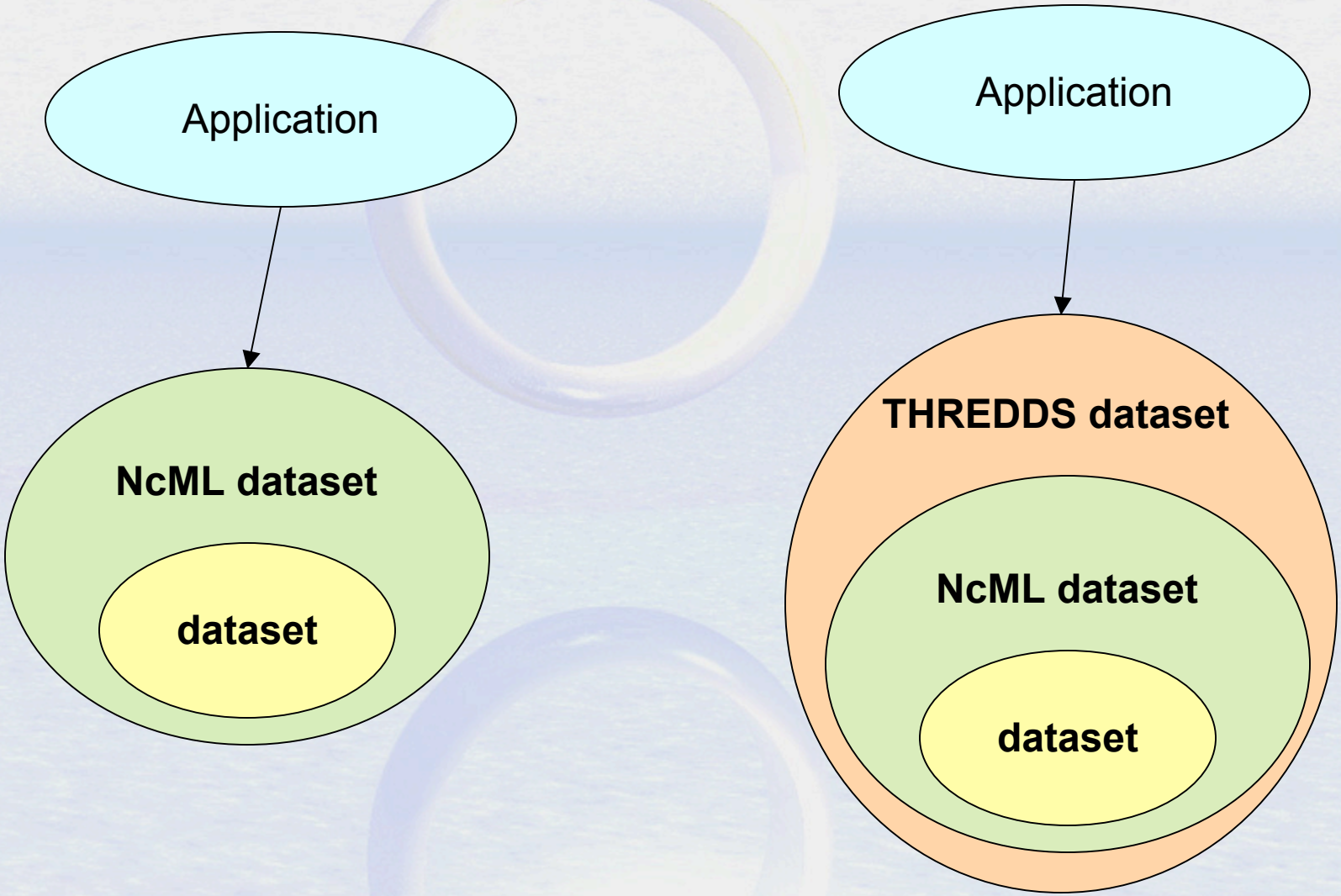
Format Readers (*CDM files*)

- *General*: NetCDF, OPeNDAP, **HDF5**, **NetCDF4, HDF4, HDF-EOS**
- *Gridded*: GRIB-1, GRIB-2, **GEMPAK**
- *Radar*: NEXRAD 2&3, DORADE, **CINRAD, Universal Format, TDWR**
- Point: **BUFR**, ASCII
- Satellite: DMSP, GINI, **McIDAS AREA**
- Misc: GTOPO, Lightning, etc
- Others in development (partial):
  - ***AVHRR, GPCP, GACP, SRB, SSMI, HIRS (NCDC)***

THREDDS Data Server



NcML Datasets



NcML example

```
<?xml version="1.0" encoding="UTF-8"?>

<netcdf xmlns="http://www.unidata.ucar.edu/schemas/netcdf/ncml-2.2"
   location="/data/nids/N0R_20041119_2147">

  <attribute name="DataType" value="Radar" />
  <remove type="attribute" name="password" />

  <variable name="Reflectivity" orgName="R34768">
    <attribute name="units" value="dBZ" />
  </variable>

</netcdf>
```

TDS / NcML example

```
<datasetScan name="Ocean Satellite Data" path="/data/ocean/sat/"
    dirLocation="R:/tds/netcdf/">

  <netcdf>
     <attribute name="Conventions" value="CF-1.0"/>
  </netcdf>

</datasetScan>
```

TDS / NcML aggregation

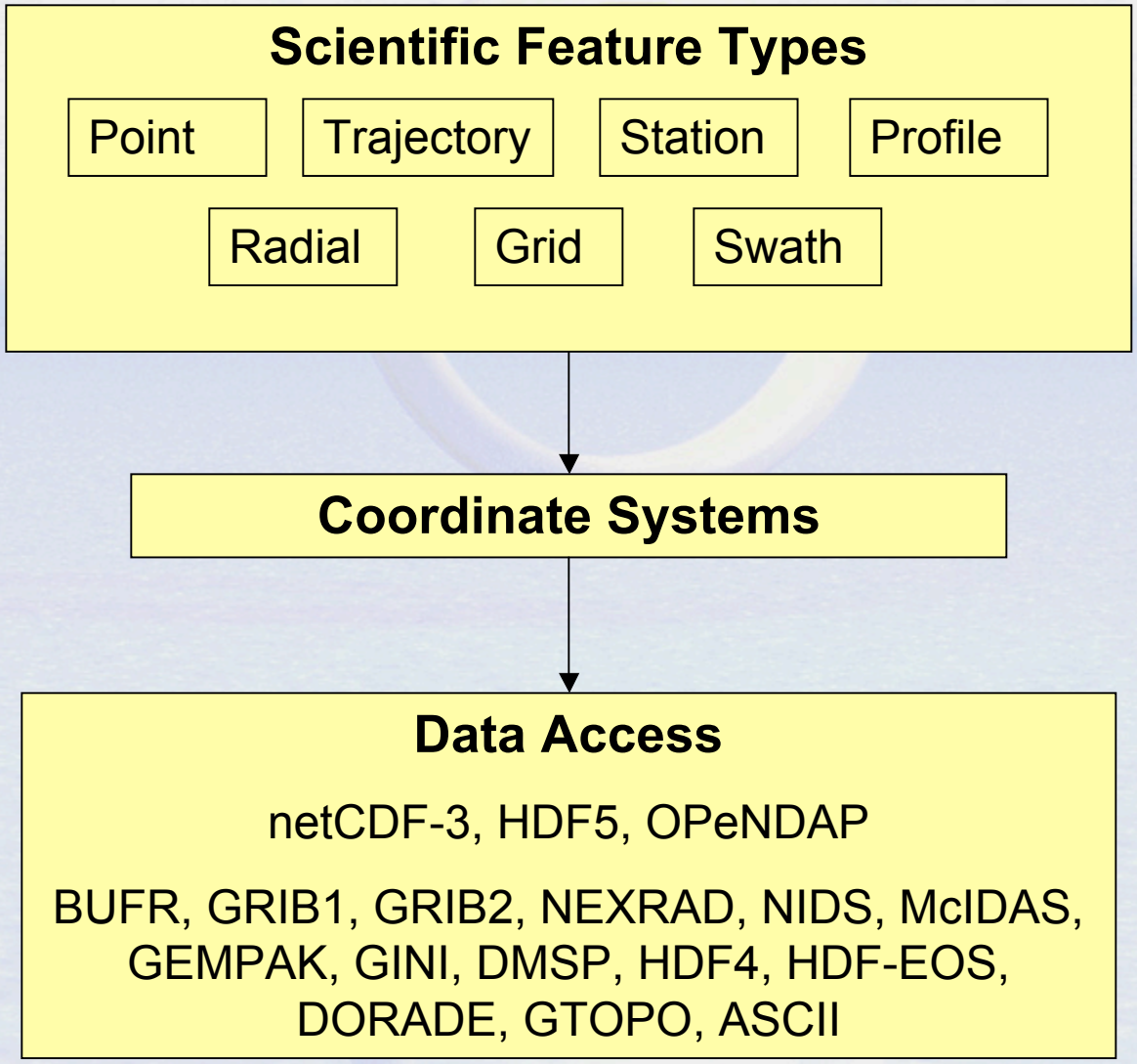
```
<dataset name="WEST-CONUS_4km Aggregation"
    urlPath="satellite/3.9/WEST-CONUS_4km">

  <netcdf >
   <aggregation dimName="time" type="joinNew">
     <scan  location="/data/satellite/WEST-CONUS_4km/" suffix=".gini" />
  </aggregation>
 </netcdf>

</dataset>
```

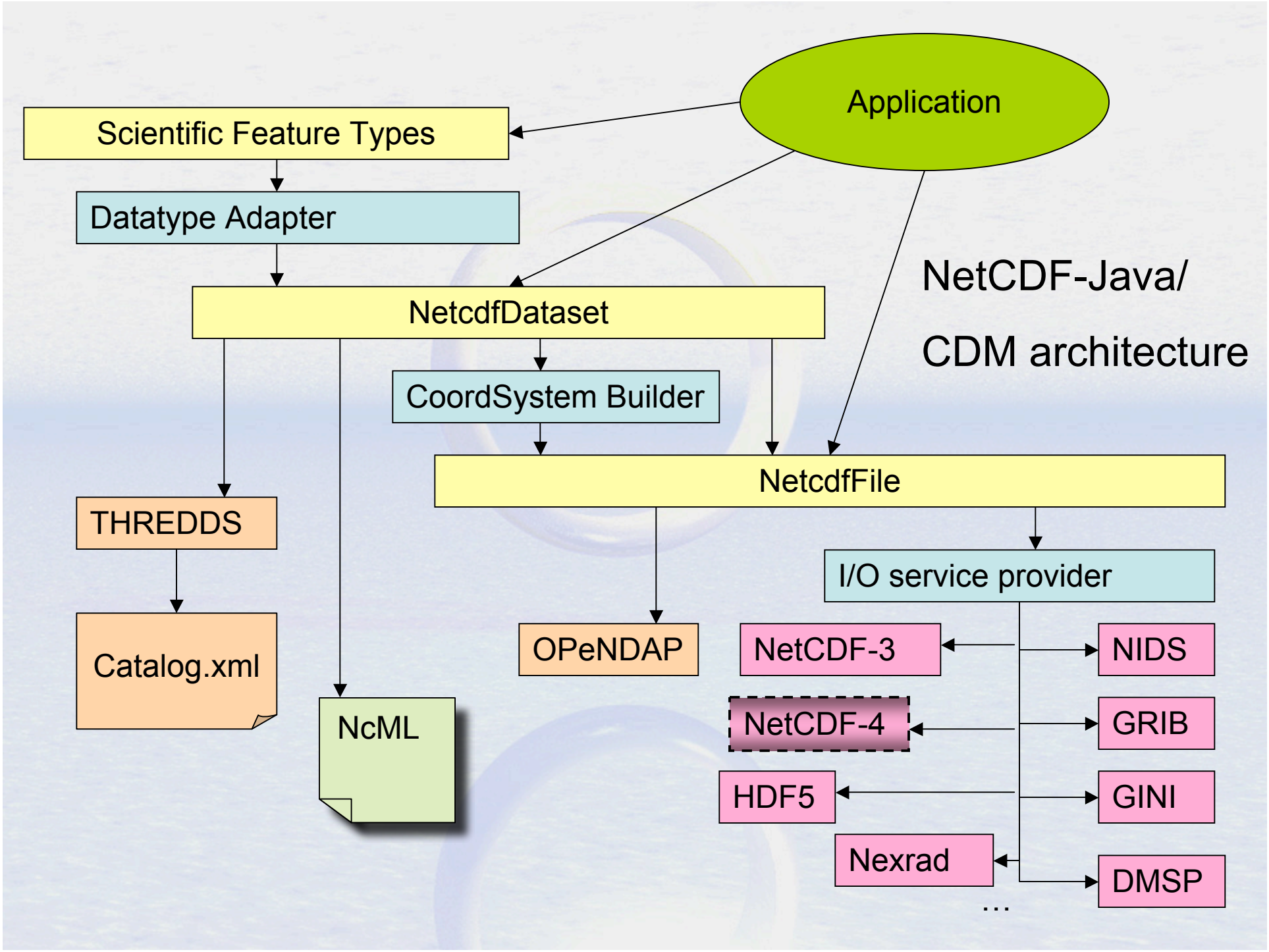
Common Data Model

What's a Data Model?

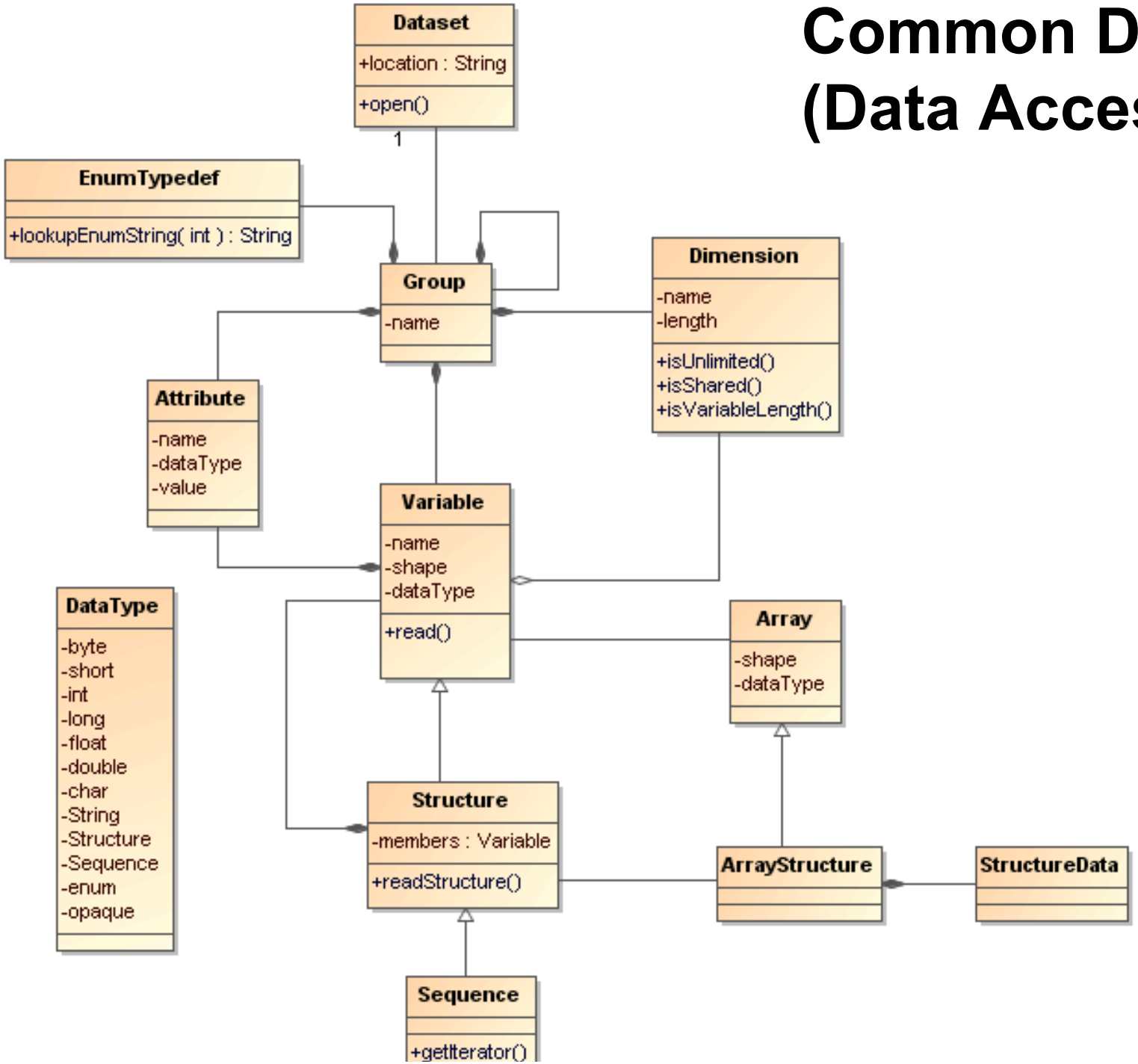
- An **Abstract Data Model** describes data objects and what methods you can use on them.
- An ***API*** is the interface to the Data Model for a specific programming language
- A **file format** is a way to persist the objects in the Data Model.
- A **data access protocol** like OPeNDAP plays the role of a file format (sort of).
- An **Abstract Data Model** removes the details of any particular API and the persistence format.

Common Data Model

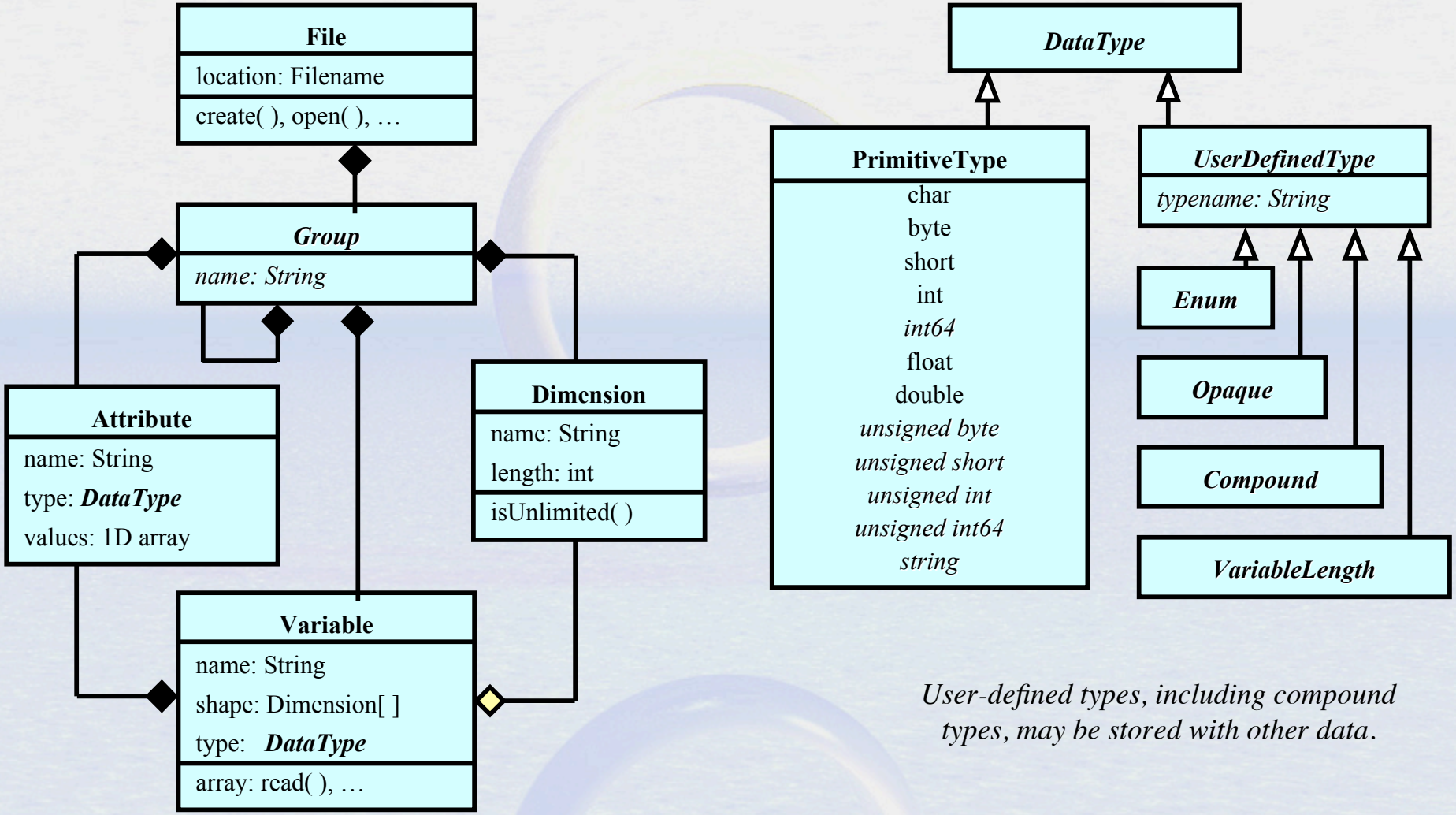



Application
Scientific Feature Types
Datatype Adapter
NetcdfDataset
CoordSystem Builder
NetcdfFile
NetCDF-Java/
CDM architecture
THREDDS
Catalog.xml
NcML
OPeNDAP
I/O service provider
NetCDF-3
NetCDF-4
HDF5
Nexrad
…
NIDS
GRIB
GINI
DMSP

Common Data Model (Data Access Layer)



NetCDF-4 Data Model



User-defined types, including compound types, may be stored with other data.

A file has a top-level unnamed group. Each group may contain one or more named subgroups, variables, dimensions, and attributes. A variable may also have attributes. Variables may share dimensions, indicating a common grid. One or more dimensions may be of unlimited length.

Coordinate Systems

Coordinate Systems as Functions

Data variable V, with n dimensions

$Vdim = \{dim_k , k=0,n-1\}$

is a function from domain Vdim to R

$V:Vdim \rightarrow R$

A coordinate variable for V is also a function

$CV:Vdim \rightarrow R$

A coordinate system for V, CSV, is a set of m coordinate variables for V

$CSV = \{CV_j , j=0,m-1\}$

$CSV:Vdim \rightarrow R^m$

The coordinates of the ***(i,j,k)*** data point are the m values

$\boldsymbol{\{CV_1(i,j,k), CV_2(i,j,k), CV_3(i,j,k),\ldots\}}$

A coordinate system must be ***invertible***.

Coordinate Systems

- NetCDF, OPeNDAP, HDF data models do not have integrated coordinate systems
  - so georeferencing not part of API
  - Need *conventions* to specify (eg CF-1, COARDS, etc)
- Contrast GRIB, HDF-EOS, other specialized formats

Coordinate Variables

```
dimensions:
   lat = 64;
   lon = 128;
variables:
   float lat(lat);
   float lon(lon);
   float time;
   double temperature(lat,lon);
     coordinates="lat lon time";
```

Limitations of 1D Coordinate Variables

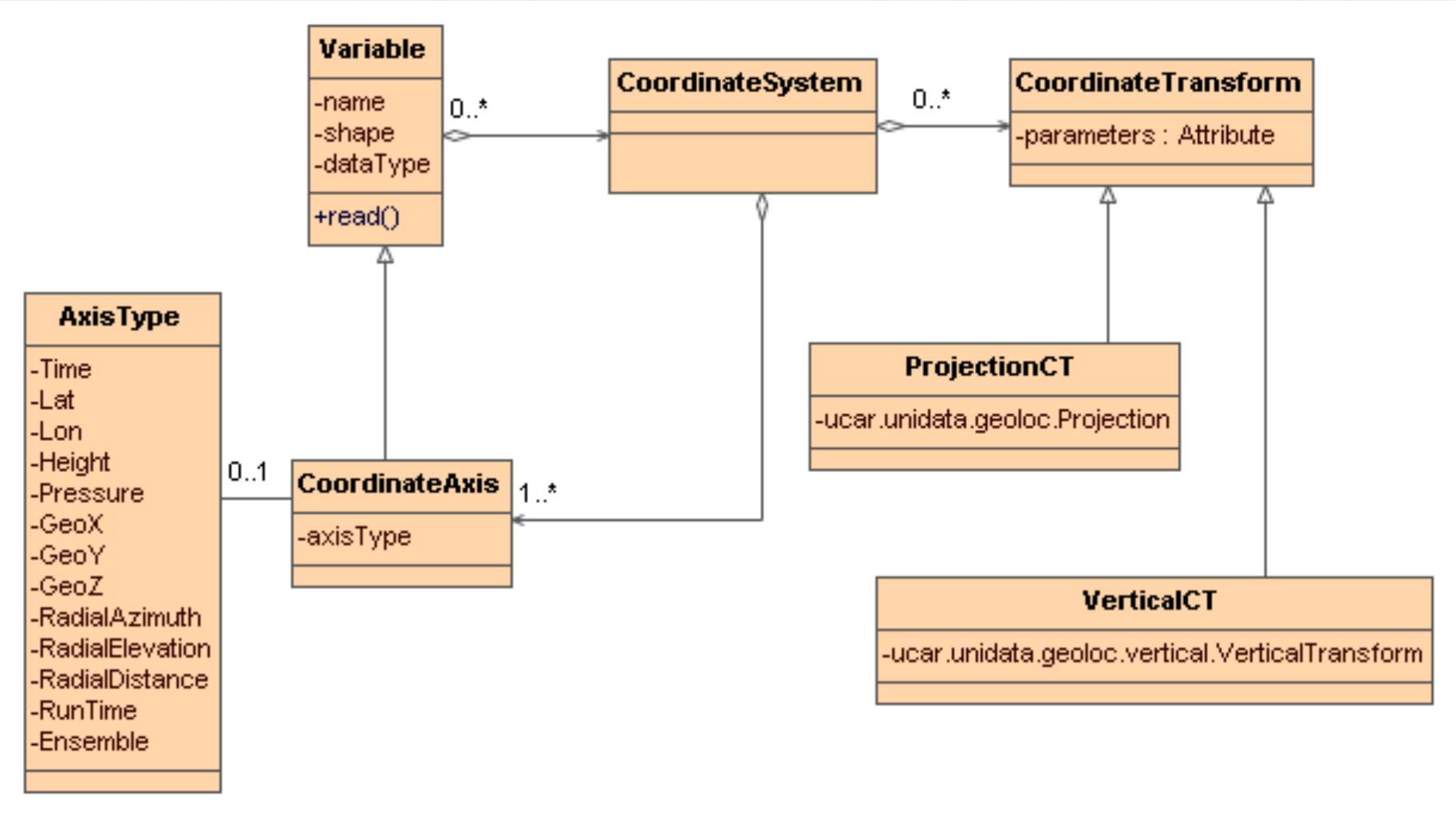
- Non lat/lon horizontal grids:

```
float temperature(y,x)
  float lat(y, x);
  float lon(y, x);
```

- Trajectory data:

```
float NKoreaRadioactivity(pt);
  float lat(pt);
  float lon(pt);
  float altitude(pt);
  float time(pt)
```

Coordinate Systems UML



Projections (CF)

- albers_conical_equal_area
- lambert_azimuthal_equal_area
- lambert_conformal_conic
- mcidas_area
- mercator
- orthographic
- rotated_pole
- stereographic (including polar)
- transverse_mercator
- UTM (ellipsoidal)
- vertical_perspective

Vertical Transforms (CF)

- atmosphere_sigma
- atmosphere_hybrid_sigma_pressure
- atmosphere_hybrid_height
- ocean_s
- ocean_sigma
- existing3DField

Add your own Transform

- Pluggable framework
  - Add at runtime
  - CoordTransBuilder.registerTransform()
- Implement *CoordTransBuilderIF*

Scientific Feature Types

Scientific Feature Types

- Based on datasets Unidata is familiar with
  - APIs are evolving
- Intended to scale to large, multifile collections
- Intended to support “specialized queries”
  - Space, Time
- These form the basis for NetCDF-Java implementations
- Two categories : Grids and Points

Gridded Data

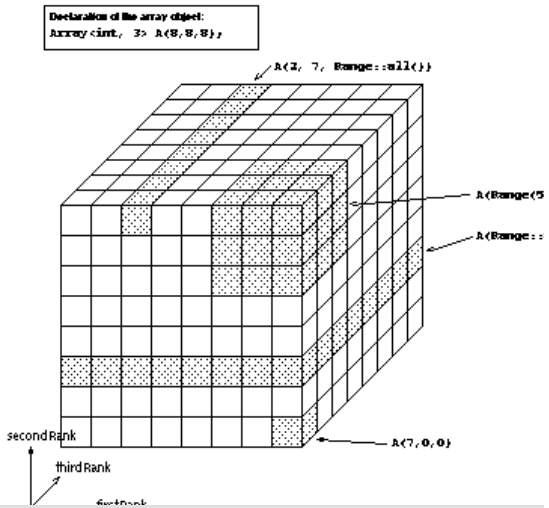
- **Grid**: multidimensional grid, separable coordinates
- **Radial**: a connected set of *radials* using polar coordinates collected into *sweeps*
- **Swath:** a two dimensional grid, *track* and *cross-track* coordinates
- **Unstructured Grids**: finite element models, coastal modeling

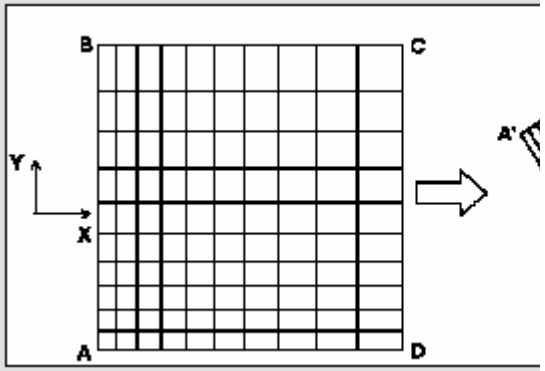
Gridded Data

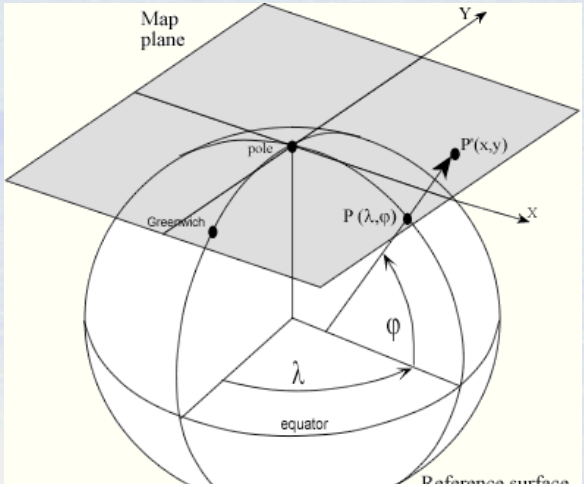
- Cartesian coordinates
- Data is 2,3,4D
- All dimensions have 1D coordinate variables (separable)

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

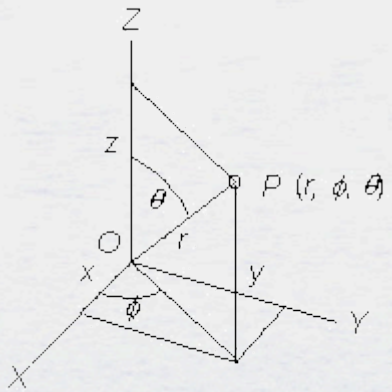
  float **lat**(y,x);
  float **lon**(y,x);
  float **height**(t,z,y,x);







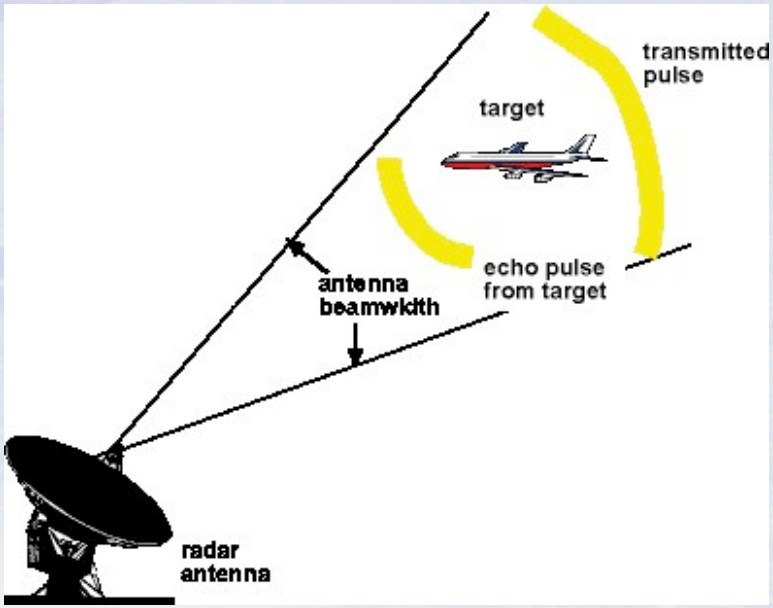
Radial Data



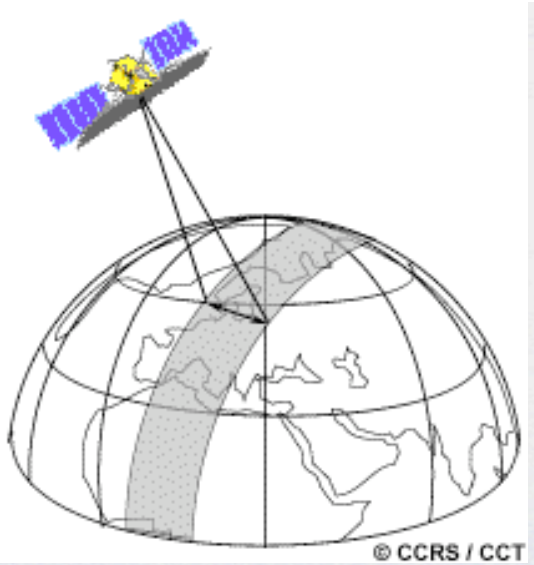
- Polar coordinates
- 2D: *radials* collected into *sweeps*
- Not separate time dimension

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**;

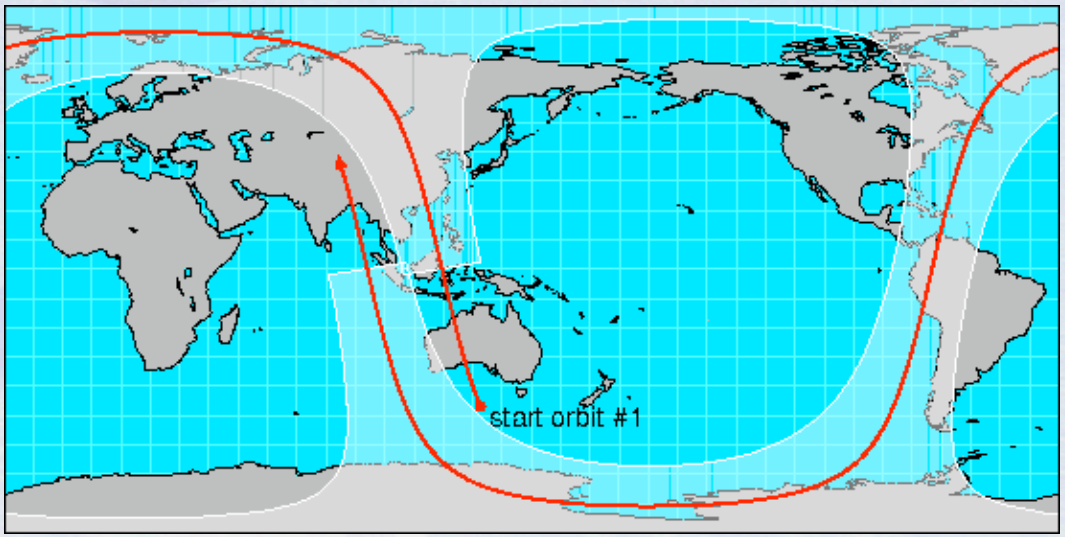


Swath



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

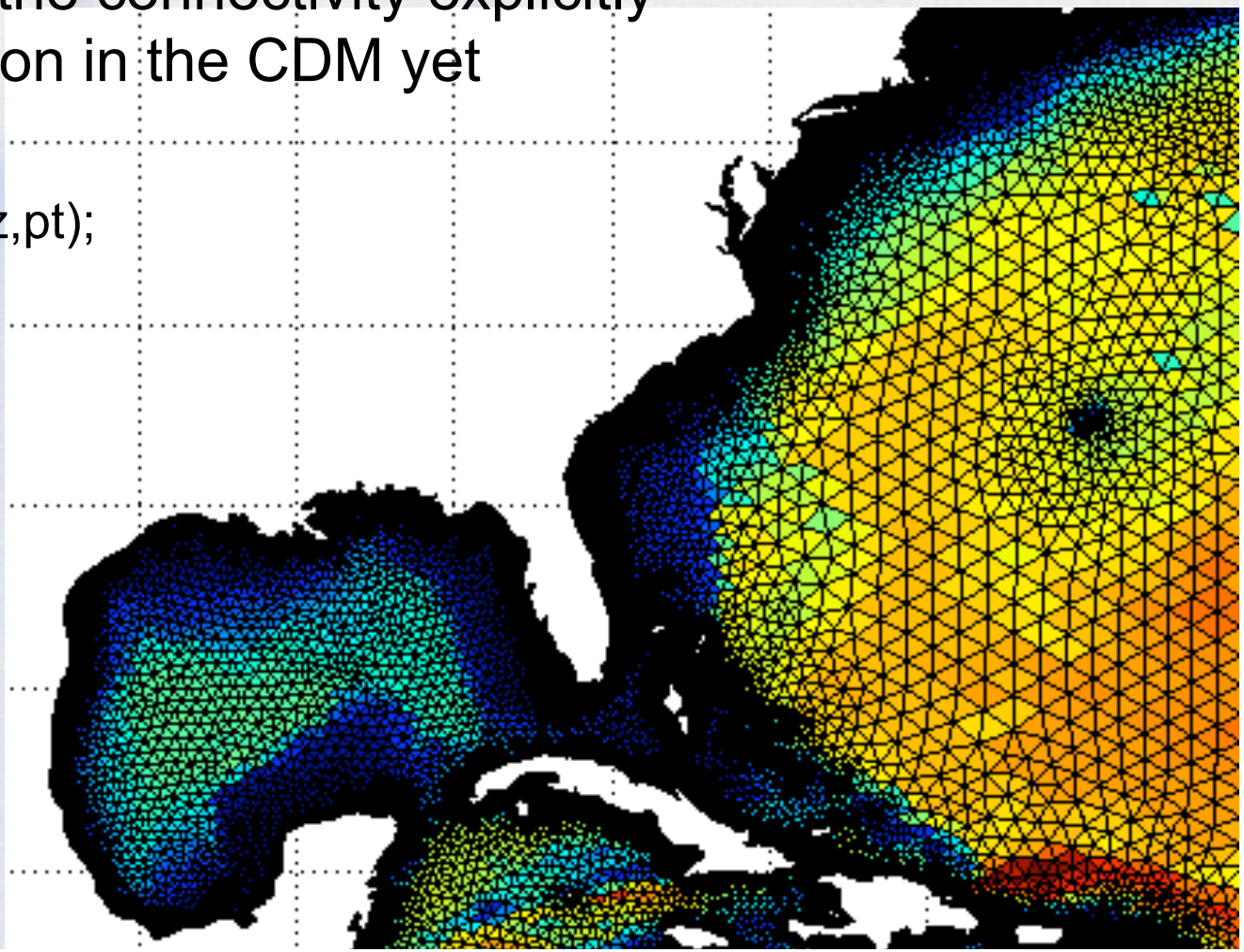
float **swathData**(track, xtrack)
  float **lat**(track, xtrack)
  float **lon**(track, xtrack)
  float **alt**(track, xtrack)
  float **time**(track)



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);



1D Feature Types (“point data”)

float data(sample);

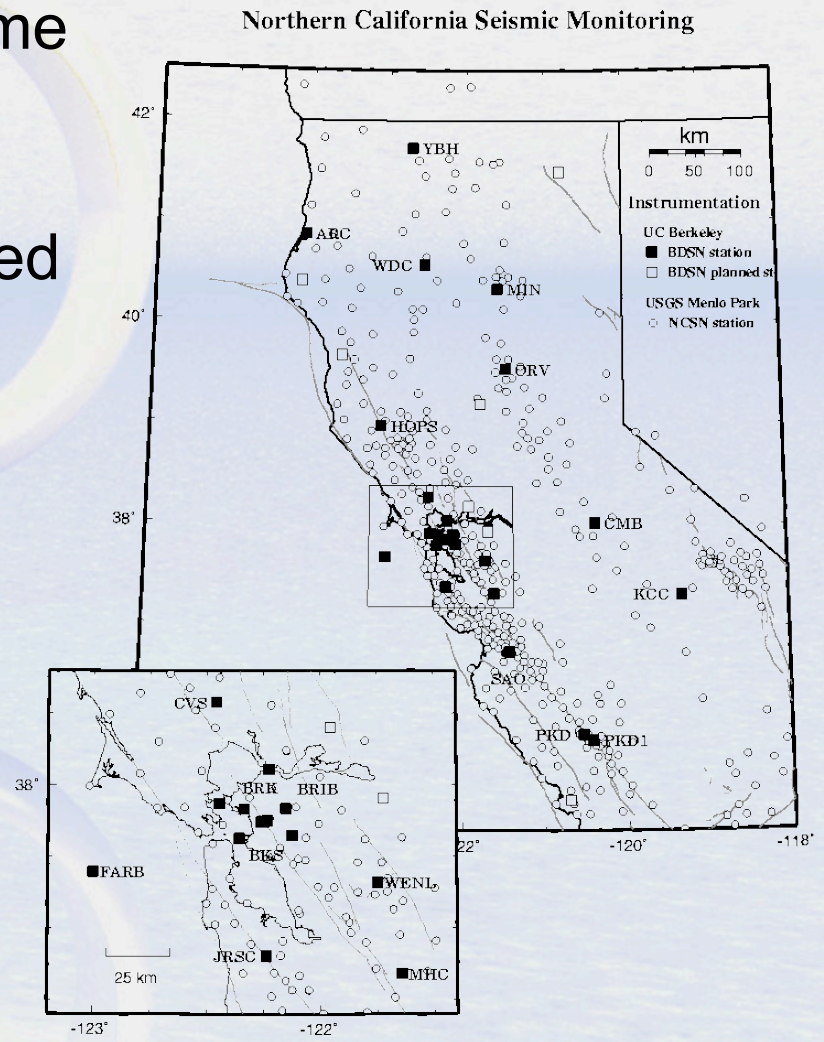
- **Point**: measured at one point in time and space
- **Station**: time-series of points at the same location
- **Profile**: points along a vertical line
- **Station Profile**: a time-series of profiles at same location.
- **Trajectory**: points along a 1D curve in time/space
- **Section**: a collection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

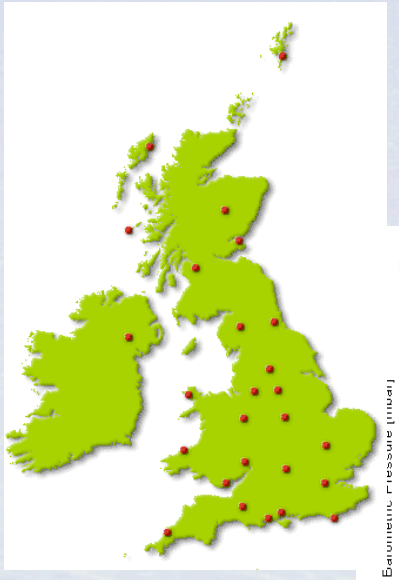


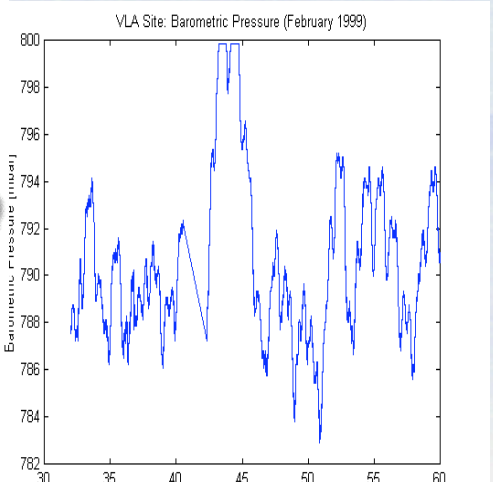
Time-series Station Data

```
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
```

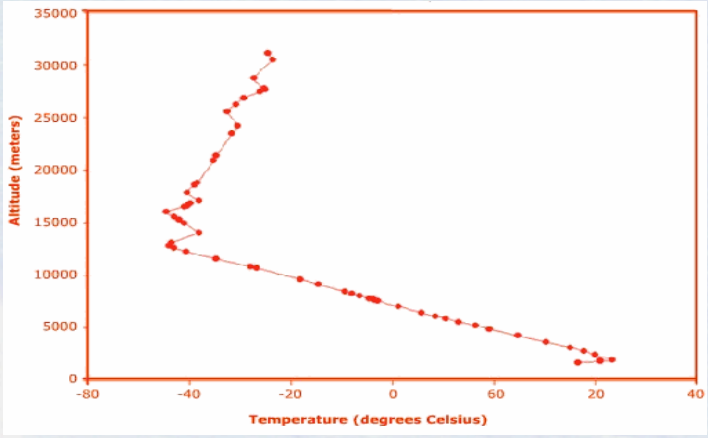
Profile Data

```
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

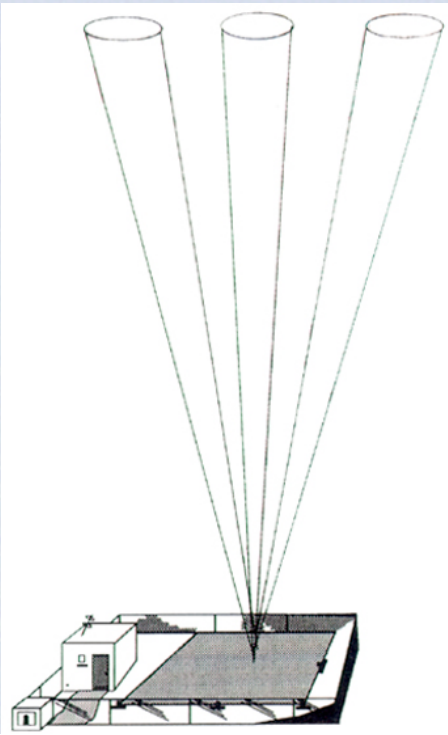
float obs1(**profile, level**);
float obs2(profile, level);
float z(profile, level);

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);

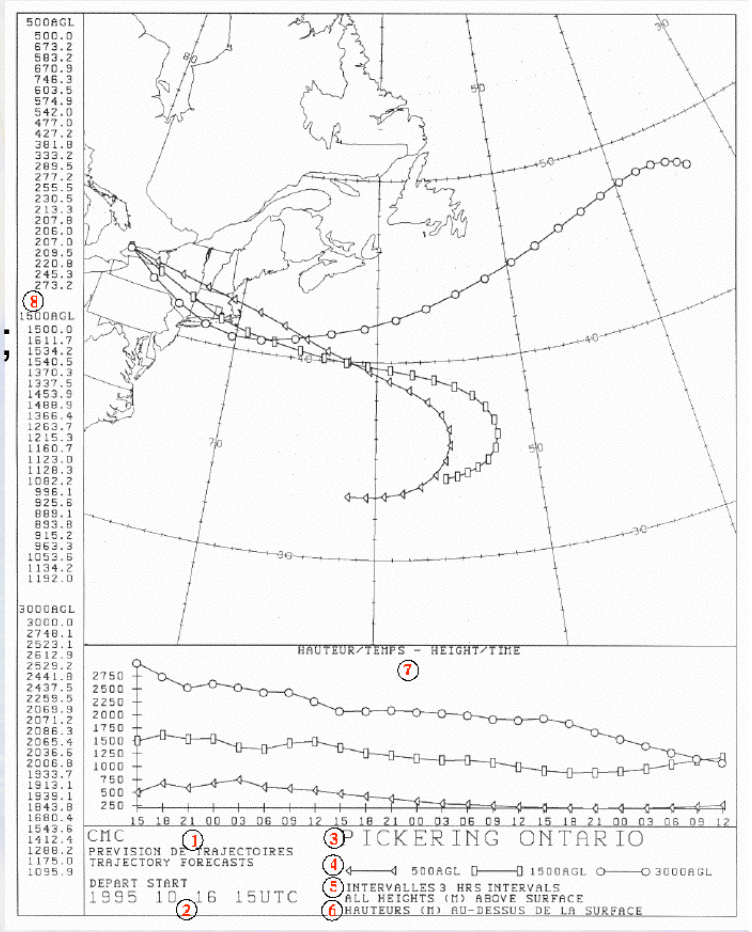
```
Table {
  stationId;
  lat, lon;
  Table {
    time;
    Table {
      z;
      obs1, obs2, ...
    } obs(*); // connected
  } profile(*); // connected
} stn(stn); // not connected
```



Trajectory Data

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

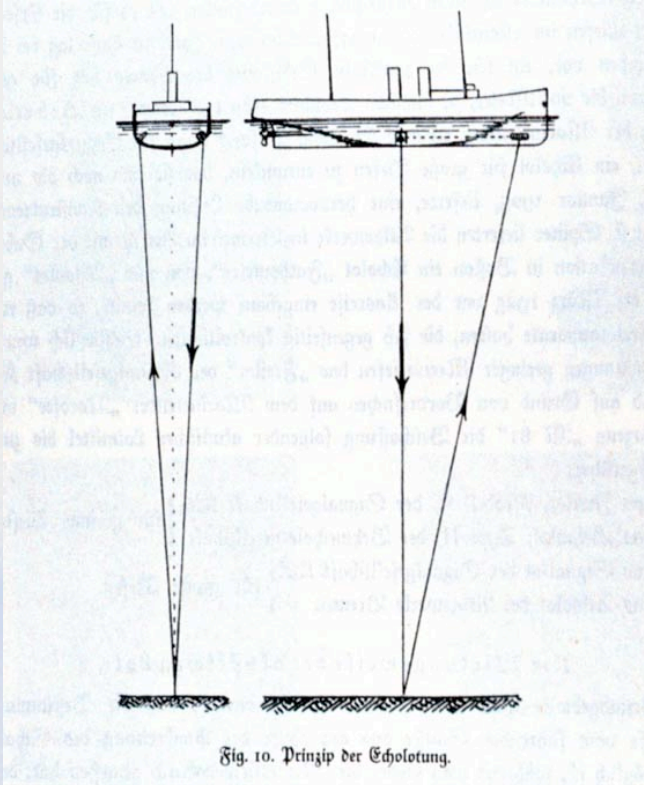
```
float obs1(traj,obs);
float obs2(traj,obs);
  float lat(traj,obs);
  float lon(traj,obs);
  float z(traj,obs);
  float time(traj,obs);
  int trajectory_id(traj);
```



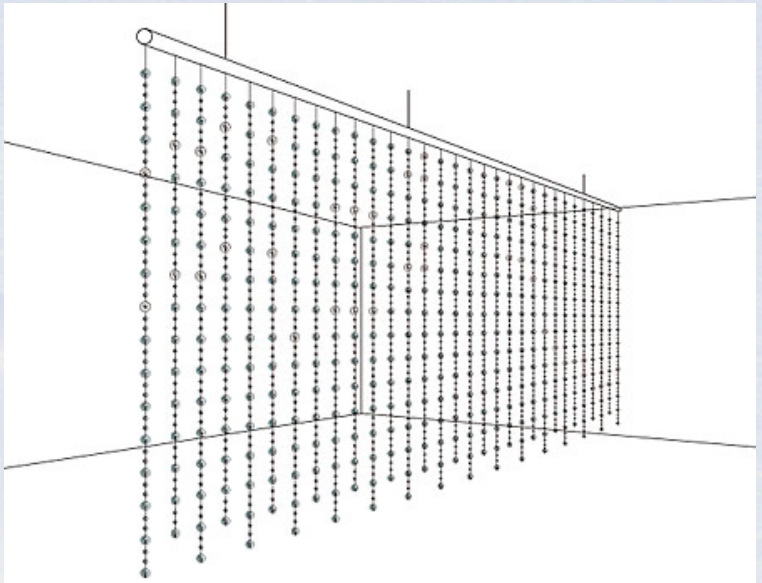

```
Table {
  trajectory_id;
  Table {
      lat, lon, z, time;
      obs1, obs2, ...
    } obs(*); // connected
  } traj(traj) // not connected
```

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
- More details

Feature Type implementations Netcdf-Java library

Grid	GridDatatype
Radial	RadialSweepFeature
Swath	-
Unstructured Grids	-
Point	PointFeature
Station	StationTimeSeriesFeature
Profile	ProfileFeature
Trajectory	TrajectoryFeature
StationProfile	StationProfileFeature
Section	SectionFeature

Encoding Feature Types in NetCDF Using CF Conventions

- CF-1.0 focused on Grids
- Other types are being studied / proposed
- Unidata proposal for point obs
- NCAR/EOL working on Radial data (netCDF4)
- NPOESS/GOES-R using netCDF4 for satellite (swath)
  - Unidata has proposal to NOAA/NASA
- Working group for unstructured grids
- Happening now!