Code Changes

Warwick RSE

Not writing data

What to do?

- The best way of avoiding difficulty with writing files is not to write them at all
- Often you can do your analysis as the code runs and only output the result of the analysis and this will be smaller
  - Or at least reduce the amount of data to be written - if you want to sum over a direction, do this before writing the data!
- This is generally called "in-code data reduction"
- If you are using stock/community software, check the manual to see if there are options to do it
- For your own code, maybe add this if possible

Reading and Writing Data

What to do?

- Basically the best solution to all of these problems is to avoid using many small files
- Instead store all of your data into far fewer files
- Classic worst case scenario from HPC
  - File per processor per variable per timestep
  - Coalesce the files into at least one file per timestep
- Now one file contains a lot of different data, how do you cope with that?

File Formats

Structured Files

- If your output is always of the same form then you can simply put in data chunk after data chunk and read them back out knowing the form of the output

- This has very severe limitations, you have to know exactly what data is in your file, in what order and the lengths of each data chunk.

- Changes to output involve changes to readers and you have to be careful to know what version of the code wrote the output or you will read it wrong

- More generally, you need **metadata**

“meta-

1 Transcending, encompassing

2 Pertaining to a level above or beyond; reflexive; about itself or about other things of the same type. For example, metadata is data that describes data, metalanguage is language that describes language, etc. [From 17th century]

3 Having analogies with metaphysics.”

Wiktionary, used under creative commons

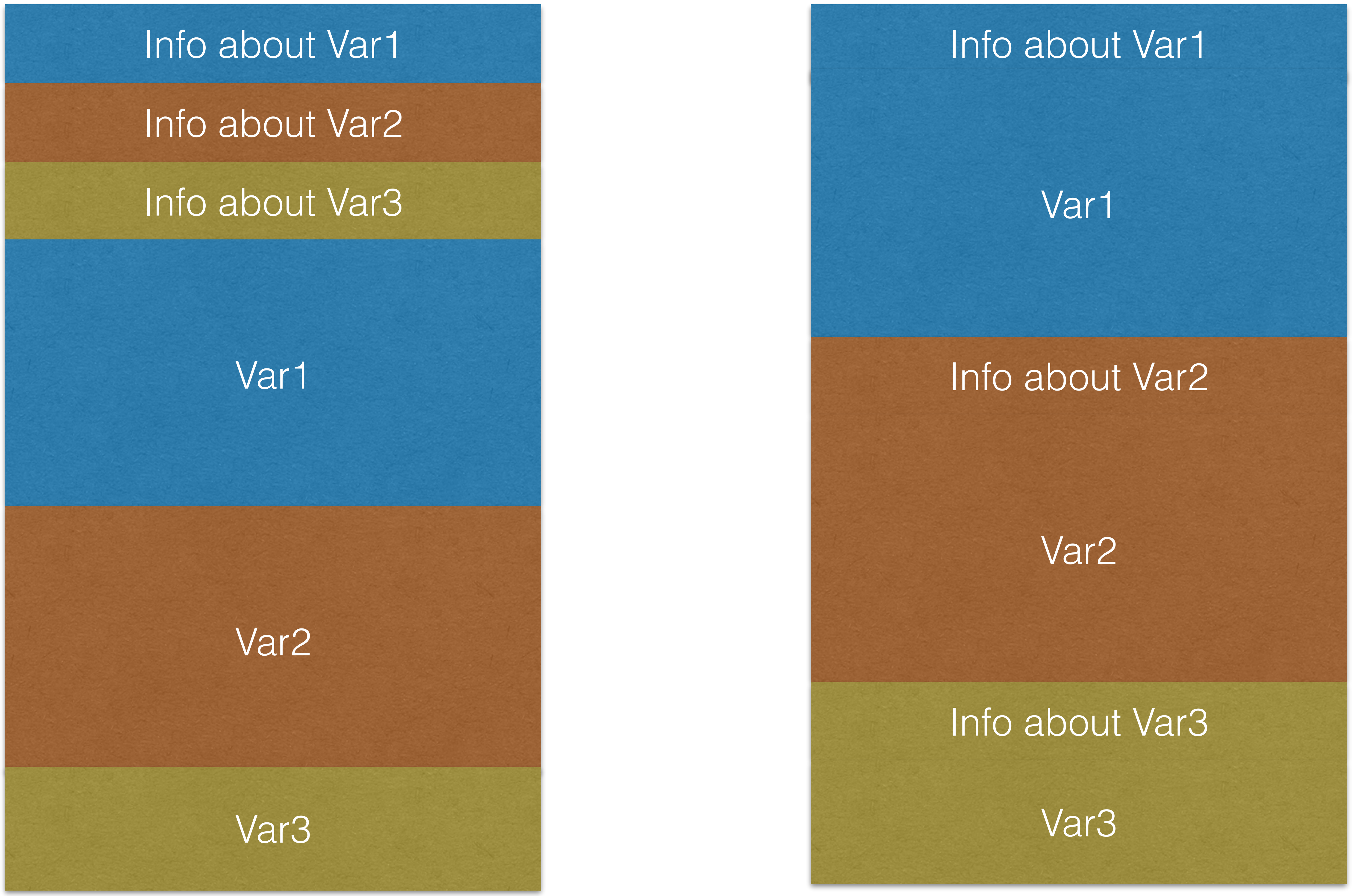
Block Metadata

- Generally work by splitting your data into self contained blocks that fully describe themselves
- Typically this description involves
    - Name of variable
    - Type of data (Integer/Real, 32 bit or 64 bit, bytestream)
    - Rank of data
    - Size of array in each dimension
    - Layout structure data

Layout Data

- Generally you want to be able to read only parts of a file
- Part of the metadata is usually information on where the actual data is stored
  - Allows the reader to jump straight to the actual data for a specific variable
- Different strategies

Metadata


Info about Var1
Info about Var2
Info about Var3
Var1
Var2
Var3
Info about Var1
Var1
Info about Var2
Var2
Info about Var3
Var3

Versioning

- It is unlikely that the first version of your file format is going to work forever
  - None of the popular standard file formats managed this (or even close)
- One of the most important things is to have some way of specifying the version of your output so that you can tell exactly how to read the format

File header

- Lots of things go in the file header, but common examples include
    - “Magic sequence” - something at the start of the file that lets you know it is one of your files
    - Version information
    - Endian-ness of your file
    - Code information - version number, parameters etc.
    - Date file was written at, description of the file, name of person who generated etc.

Writing your own format

- In an ideal world people would be able to write their own structured data files that are well suited to the data their codes are handling
- This is **a lot** of work so it isn't really terribly practical
- Usually use some kind of standard file format
    - NetCDF
    - ADIOS
    - **HDF5**

Standard formats

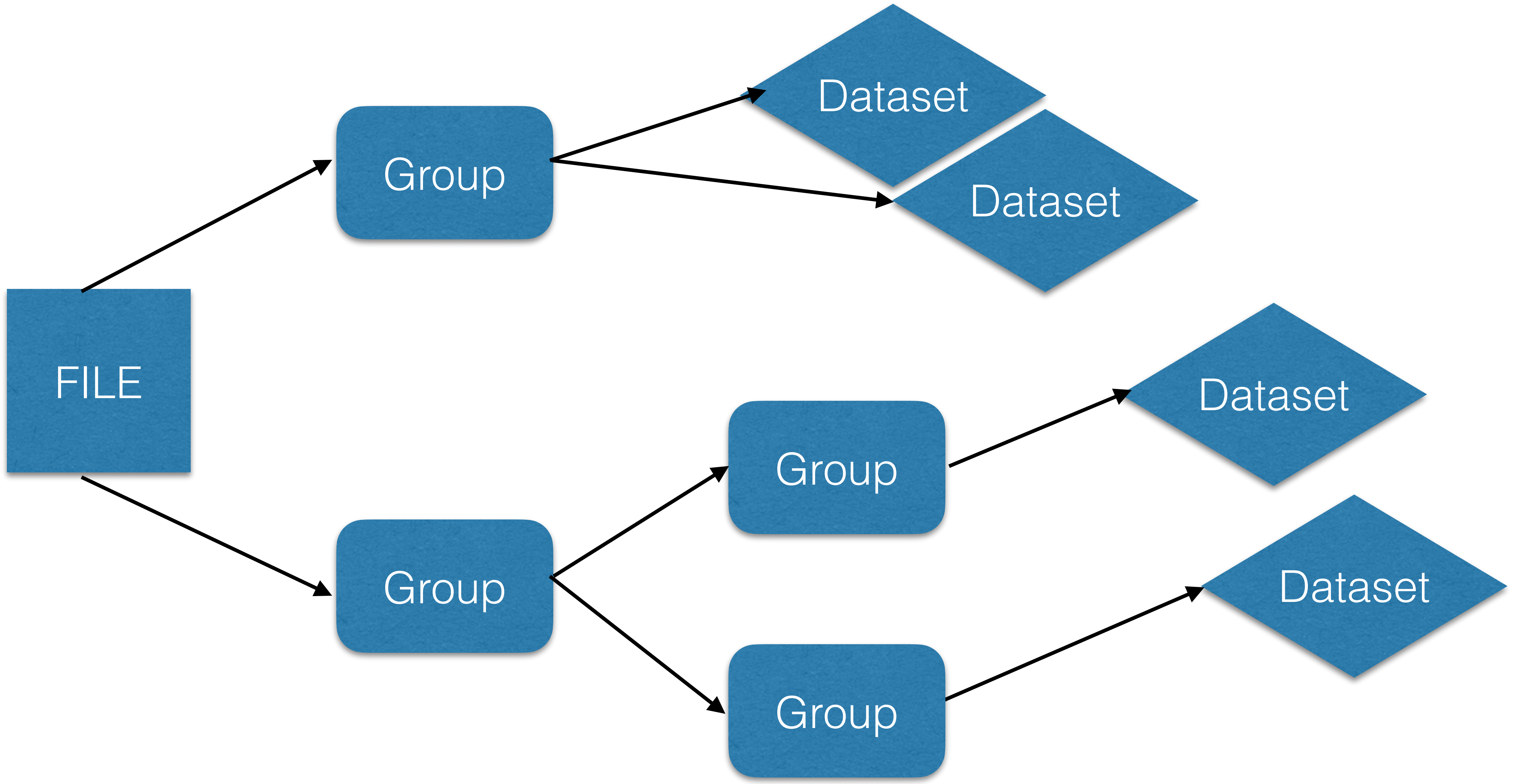
- Standard formats always make compromises on suitability for any particular purpose
  - Also on performance, memory usage etc, but mostly this isn't terrible
- None of them are really perfect for all purposes, but HDF5 is probably the most flexible
- N.B. Other than NETCDF, these standard formats simplify writing and retrieving data but they don't tell you what to do with the data when you have it
  - i.e. You have to know that *this* named variable is the axis to plot *that* named variable against

HDF5

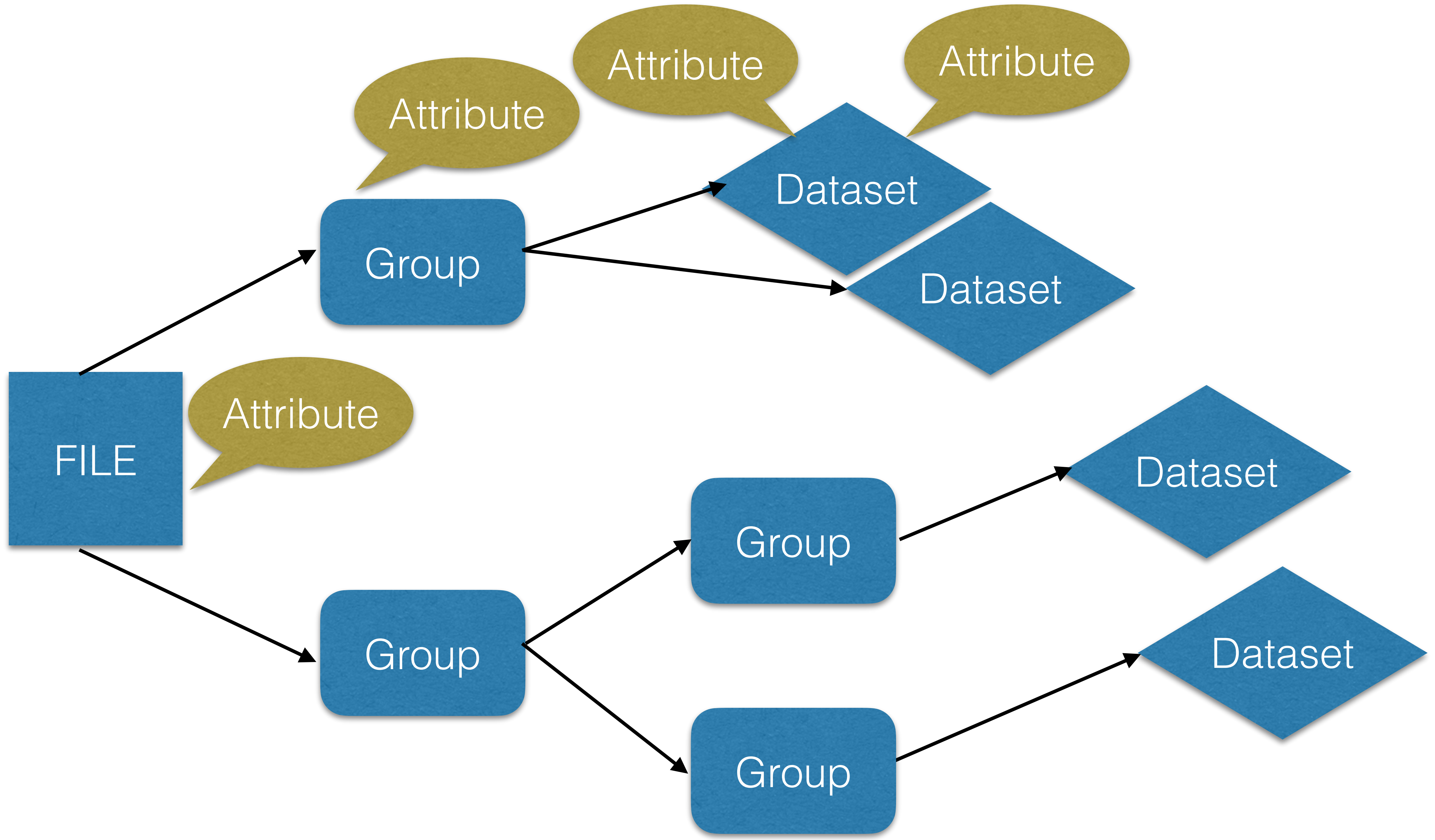
HDF5

- HDF5 defines an hierarchy of groups and datasets that allows you to store arbitrary data with any structure that you like
- There is metadata at every level, files, groups and datasets, but this metadata is very general
  - Can attach “attributes” to almost anything else in HDF
- In many senses HDF5 isn’t really a file format - you can’t just grab an HDF5 file from a code that you don’t know and start plotting the data
- But, it is a good way of storing data and retrieving it later

HDF5


FILE
Group
Group
Dataset
Dataset
Group
Group
Dataset
Dataset

HDF5


Attribute
Attribute
Attribute
Dataset
Group
Dataset
FILE
Attribute
Dataset
Group
Group
Dataset
Group

Simple HDF5 Creation

```python
import h5py
import numpy as np

# Create the HDF5 file
file = h5py.File('my_file.h5', 'w')

# Create a group called "test group"
group = file.create_group('test group')

# Create a dataset "ds1" with a numpy array of 100x100
ds1_data = np.random.rand(100, 100)
ds1 = group.create_dataset('ds1', data=ds1_data)

# Create a dataset "ds2" with a numpy array of 25x75
ds2_data = np.random.rand(25, 75)
ds2 = group.create_dataset('ds2', data=ds2_data)

# Close the HDF5 file
file.close()
```

HDF5

- There is a very similar interface in C++ and Java
- There are also C, Fortran90 and Fortran2003 interfaces that use a slightly different model
  - They use unique ID handles for things like groups rather than an object oriented approach but the general idea is the same
- There are unofficial bindings for almost everything else - Matlab, Rust, IDL, Perl etc. etc.

MPI-IO

MPI-IO

- When writing files from a single program (even one that uses multiple threads to use multiple processors on a machine) it is “easy” to write a single file with all the data

- If you are writing a distributed parallel code using MPI then you have an extra problem - the data isn’t all together in any one place to write it into a single file

- MPI-IO is part of the MPI library standard that works much like writing a file normally but allows you to specify a “file view” that says which bit of the data each rank holds

- HDF5 can use MPI-IO behind the scenes. Performance is acceptable rather than impressive, but it still works pretty well. ADIOS is really designed for parallel IO and works well at very large scale

Databases

Databases

- If your problem is as much about accessing the data of interest as it is about storing the actual data then you might want to look at databases

- Databases fall into three categories

    - RDBMS - Classical database with data organised into tables, with each table consisting of columns of what data to be stored and rows or records of what data is actually being stored. Often accessed through a language called SQL. e.g. **MySQL/MariaDB**, **PostGRES**, **InnoDB**, **SQLite**

Databases

- Document databases - They store data as individual documents that are not all alike. You can create indices that allow you to search on data that **is** shared. e.g. **MongoDB**

- Graph databases - Databases that organise relationships between data as graphs (in a graph theory sense). Quite specialised, but very useful for tasks where knowing one piece of data tells you what other data you might want e.g. **Neo4j**

- The general idea of these is that the data does not map well onto the idea of rows and columns of data, or that you don't need the same guarantees of consistency that RDBMS systems give you

- These two are sometimes called **NOSQL** databases, even though you can sometimes use SQL to interact with them!

SQLite and SQL

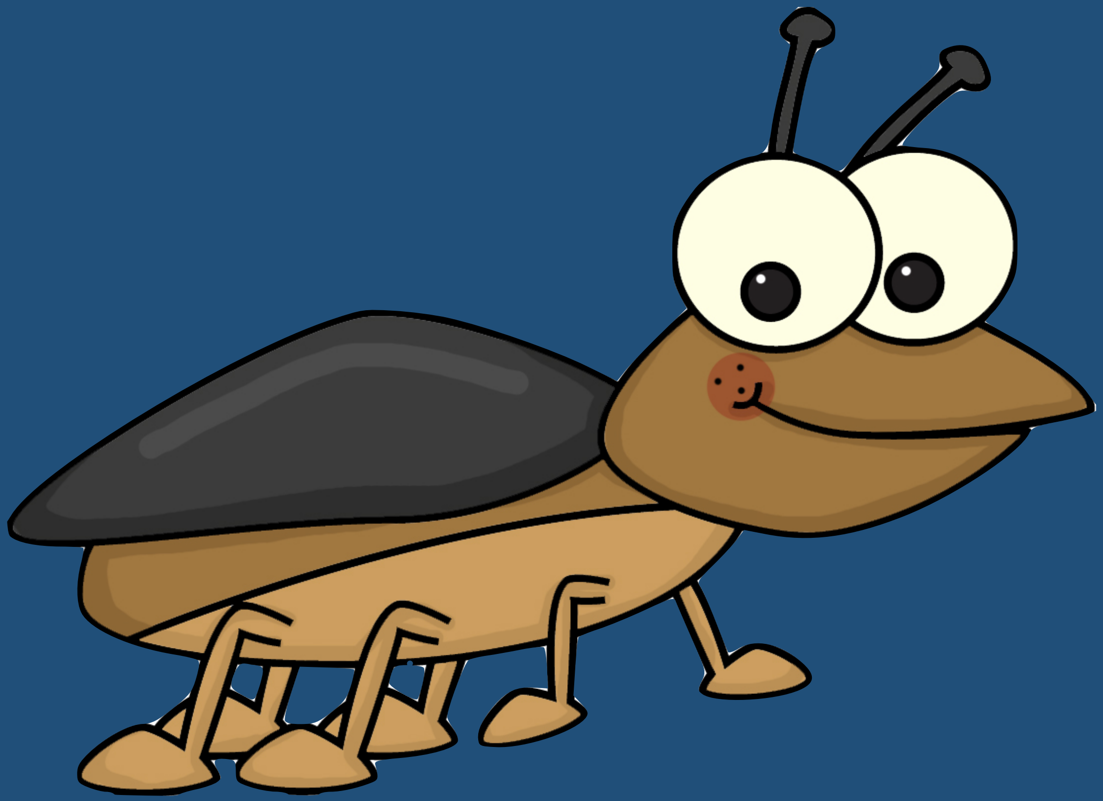
```
import sqlite3

# Connect to the database (creates a new database if it doesn't exist)
conn = sqlite3.connect('mydatabase.db')

# Create a cursor object to execute SQL commands
cursor = conn.cursor()

# Create the table
cursor.execute('''CREATE TABLE IF NOT EXISTS mytable
                  (ID INTEGER PRIMARY KEY AUTOINCREMENT,
                   Name TEXT,
                   Count INTEGER)''')

# Commit the changes
conn.commit()
#Add three records to the table
cursor.execute("INSERT INTO mytable (Name, Count) VALUES ('A', 1)")
cursor.execute("INSERT INTO mytable (Name, Count) VALUES ('B', 2)")
cursor.execute("INSERT INTO mytable (Name, Count) VALUES ('C', 3)")
# Commit the changes and close the connection
conn.commit()
conn.close()
```

Databases

- Databases can scale from tiny (with something like SQLite) to massive and distributed (MySQL powers companies at Google like scales) and are some of the most optimised pieces of software in the world

- While they can store chunks of binary data (see BLOB (Binary Large OBject) fields), they are mainly intended for data where you want to search or otherwise query it in more complex ways

- There are database systems built on top of HDF5 (see Pytables and HDFql), that are intended to give "database-like" behaviour while remaining able to store large datasets



The End