edinburgh data-intensive research

VERCE

DISPEL introduction

Malcolm Atkinson

Malcolm.Atkinson@ed.ac.uk

16th July 2012

NSF PIRE Open Science Data Cloud

workshop

Informatics Forum

Edinburgh

FP7-INFRASTRUCTURES-2011 project# 283543

Outline

- Data Intensive
  - What is it?
  - Why use it?
- DISPEL
  - What is it?
  - Why design it?
  - Is it different?
- A simple example
- Summary and Conclusions

picture from Erica Salmon
Cornish Coast Path
where I call home

Data-Intensive Thinking

Data-Intensive Thinking

epcc

OGSA-DAI

DATA MINING ADMIRE DATA INTEGRATION

Computing over data pedigree!

edinburgh
data-intensive
research

- **Tycho Brahe and Johannes Kepler**

  1546-1601 & 1571-1630

- **Dennis Noble uses Mercury**
  - *The* London University Computer in 1959
  - to demonstrate heart beats as emergent behaviour
  - by simulating two ion channels
  - 2 papers in Nature 1960
    - read "The Music of Life" by Dennis Noble

Gray's Laws of Data Engineering

Jim Gray:

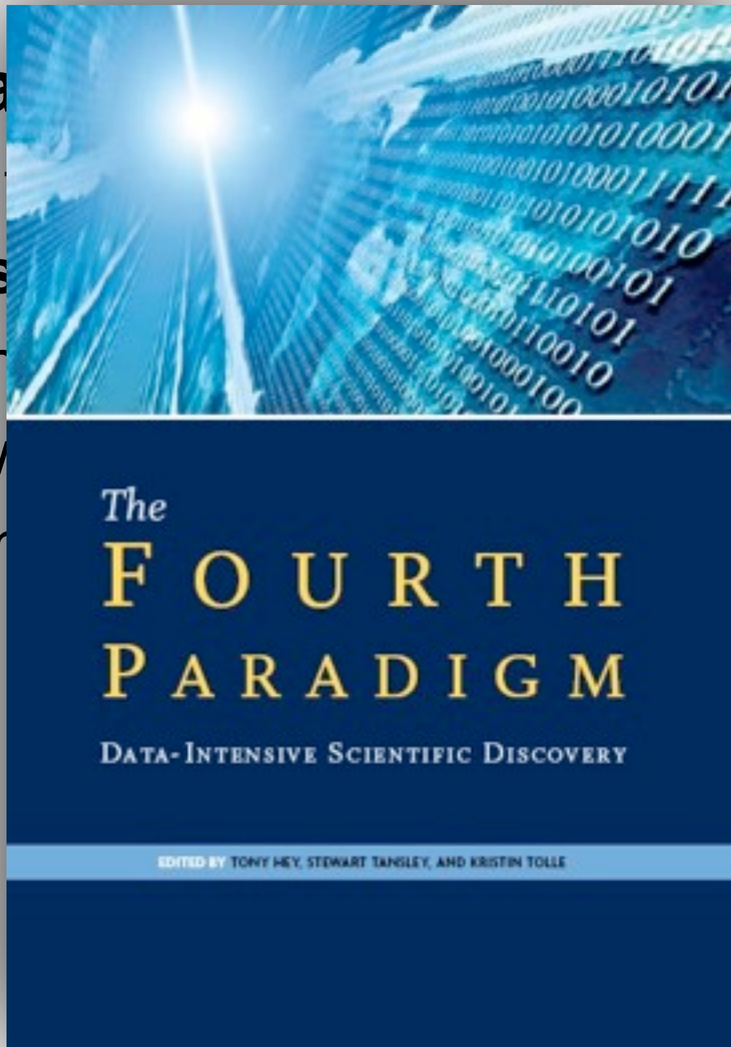
- Scientific computing is revolving around **data**
- Need **scale-out** solution for analysis
- Take the **analysis to the data**!
- Start with "**20 queries**"
- Go from "**working to working**"

From: Alex Szalay, JHU

Gray's Laws of Data Engineering

Jim Gra

- Scienti ... around **data**
- Need **s** ... ysis
- Take th
- Start w
- Go fror



From: Alex Szalay, JHU

Defining “Data-Intensive”

Defining “Data-Intensive”

- **Generally**
  - ***A computational task is data-intensive if you have to think hard about an aspect of data handling to make progress***
    - distribution, permissions and rules of use, complexity, heterogeneity, rate of arrival, unstructured or changing structure, long tail of small and scattered instances, size of data, number of users
    - often in combination

Defining "Data-Intensive"

- **Generally**
  - ***A computational task is data-intensive if you have to think hard about an aspect of data handling to make progress***
    - distribution, permissions and rules of use, complexity, heterogeneity, rate of arrival, unstructured or changing structure, long tail of small and scattered instances, size of data, number of users
    - often in combination
- **Quantitatively**
  - **The computation's Amdahl numbers are close to 1**
    - CPU operations : bits transferred in or out of memory
    - 1000 CPU operations : 1 I/O operation
  - **Total volumes expensive to store**
  - **Total requests/unit time hard to accommodate**

Data-Intensive Strategies 1

Data-Intensive Strategies 1

- Use commodity components and low power
  - So that you can afford a lot of them
  - Balanced for data-intensive work
  - Treat memory bandwidth as a scarce resource

Data-Intensive Strategies 1

- **Use commodity components and low power**
  - So that you can afford a lot of them
  - Balanced for data-intensive work
  - Treat memory bandwidth as a scarce resource
- **Data & computation as close together as possible**
  - in the processor cache in fewest steps & not disrupted

Data-Intensive Strategies 1

- Use commodity components and low power
  - So that you can afford a lot of them
  - Balanced for data-intensive work
  - Treat memory bandwidth as a scarce resource
- Data & computation as close together as possible
  - in the processor cache in fewest steps & not disrupted
- Work on small chunks of data
  - as small as logically possible
  - a column of a table
  - a row of a table
  - a file
  - data unbundled, in computational format & compressed

Data-Intensive Strategies 1

- Use commodity components and low power
  - So that you can afford a lot of them
  - Balanced for data-intensive work
  - Treat memory bandwidth as a scarce resource
- Data & computation as close together as possible
  - in the processor cache in fewest steps & not disrupted
- Work on small chunks of data
  - as small as logically possible
  - a column of a table
  - a row of a table
  - a file
  - data unbundled, in computational format & compressed
- Once data is close to a processor do all you can with it
  - multiple derivatives in one pass
  - pipelining
  - re-use of intermediate data, caching and forwarding

Data-Intensive Strategies 1

- Use commodity components and low power
  - So that you can afford a lot of them
  - Balanced for data-intensive work
  - Treat memory bandwidth as a scarce resource
- Data & computation as close together as possible
  - in the processor cache in fewest steps & not disrupted
- Work on small chunks of data
  - as small as logically possible
  - a column of a table
  - a row of a table
  - a file
  - data unbundled, in computational format & compressed
- Once data is close to a processor do all you can with it
  - multiple derivatives in one pass
  - pipelining
  - re-use of intermediate data, caching and forwarding
- Use directories and indexes to avoid revisiting data randomly

Data-Intensive Strategies 2

Data-Intensive Strategies 2

- **Exploit very large scale parallelism and distribution**
  - many subtasks at modest rate per task in large numbers
  - NOT tightly coupled parallelism!!!
  - distribution for availability, ownership & persistence
  - proximity to data sources or destinations for speed

Data-Intensive Strategies 2

- Exploit very large scale parallelism and distribution
  - many subtasks at modest rate per task in large numbers
  - NOT tightly coupled parallelism!!!
  - distribution for availability, ownership & persistence
  - proximity to data sources or destinations for speed
- Replicate
  - for more parallelism and for durable persistence

Data-Intensive Strategies 2

- **Exploit very large scale parallelism and distribution**
  - many subtasks at modest rate per task in large numbers
  - NOT tightly coupled parallelism!!!
  - distribution for availability, ownership & persistence
  - proximity to data sources or destinations for speed
- **Replicate**
  - for more parallelism and for durable persistence
- **Most data WORM (Write Once Read Many)**
  - or WORN (Write Once Read Never) - automatically eliminate or clean up

Data-Intensive Strategies 2

- Exploit very large scale parallelism and distribution
  - many subtasks at modest rate per task in large numbers
  - NOT tightly coupled parallelism!!!
  - distribution for availability, ownership & persistence
  - proximity to data sources or destinations for speed
- Replicate
  - for more parallelism and for durable persistence
- Most data WORM (Write Once Read Many)
  - or WORN (Write Once Read Never) - automatically eliminate or clean up
- Updates local and mostly append (mostly non-Transactional)

Data-Intensive Strategies 2

- Exploit very large scale parallelism and distribution
  - many subtasks at modest rate per task in large numbers
  - NOT tightly coupled parallelism!!!
  - distribution for availability, ownership & persistence
  - proximity to data sources or destinations for speed
- Replicate
  - for more parallelism and for durable persistence
- Most data WORM (Write Once Read Many)
  - or WORN (Write Once Read Never) - automatically eliminate or clean up
- Updates local and mostly append (mostly non-Transactional)
- Coordination & Catalogue DBs
  - distributed shared structures
  - just enough synchronisation

Data-Intensive Strategies 2

- Exploit very large scale parallelism and distribution
  - many subtasks at modest rate per task in large numbers
  - NOT tightly coupled parallelism!!!
  - distribution for availability, ownership & persistence
  - proximity to data sources or destinations for speed
- Replicate
  - for more parallelism and for durable persistence
- Most data WORM (Write Once Read Many)
  - or WORN (Write Once Read Never) - automatically eliminate or clean up
- Updates local and mostly append (mostly non-Transactional)
- Coordination & Catalogue DBs
  - distributed shared structures
  - just enough synchronisation
- Fine-grained local protection & authorisation

Data-Intensive Strategies 2

- Exploit very large scale parallelism and distribution
  - many subtasks at modest rate per task in large numbers
  - NOT tightly coupled parallelism!!!
  - distribution for availability, ownership & persistence
  - proximity to data sources or destinations for speed
- Replicate
  - for more parallelism and for durable persistence
- Most data WORM (Write Once Read Many)
  - or WORN (Write Once Read Never) - automatically eliminate or clean up
- Updates local and mostly append (mostly non-Transactional)
- Coordination & Catalogue DBs
  - distributed shared structures
  - just enough synchronisation
- Fine-grained local protection & authorisation
- Statistical and quantised accounting

Data-Intensive Strategies 3

Data-Intensive Strategies 3

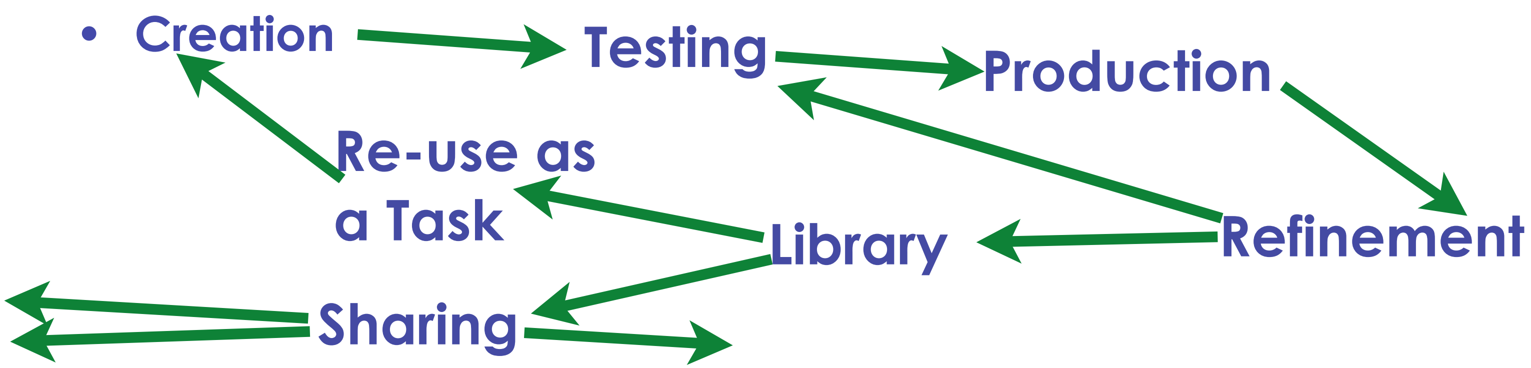
- **High-level notations for describing methods /composing tasks**
  - *with well-developed optimised transformations before execution*
  - query languages: SQL/AQL, (Xquery &SPARQL), ...
  - workflow languages: Kepler, Pegasus, DISPEL, ...
  - MapReduce: PigLatin, ZigZag, ...

Data-Intensive Strategies 3

- High-level notations for describing methods /composing tasks
  - *with well-developed optimised transformations before execution*
  - query languages: SQL/AQL, (Xquery &SPARQL), ...
  - workflow languages: Kepler, Pegasus, DISPEL, ...
  - MapReduce: PigLatin, ZigZag, ...
- Providers + Community + User definition of (libraries of) tasks
  - your signal processing, geophysics & data-presentation steps
  - your existing code & preferred languages

Data-Intensive Strategies 3

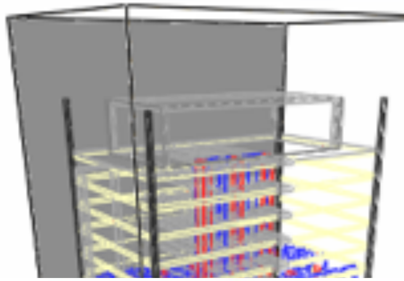
- **High-level notations for describing methods /composing tasks**
  - *with well-developed optimised transformations before execution*
  - query languages: SQL/AQL, (Xquery &SPARQL), ...
  - workflow languages: Kepler, Pegasus, DISPEL, ...
  - MapReduce: PigLatin, ZigZag, ...
- **Providers + Community + User definition of (libraries of) tasks**
  - your signal processing, geophysics & data-presentation steps
  - your existing code & preferred languages
- **Support for the query & workflow lifetime: new research objects**

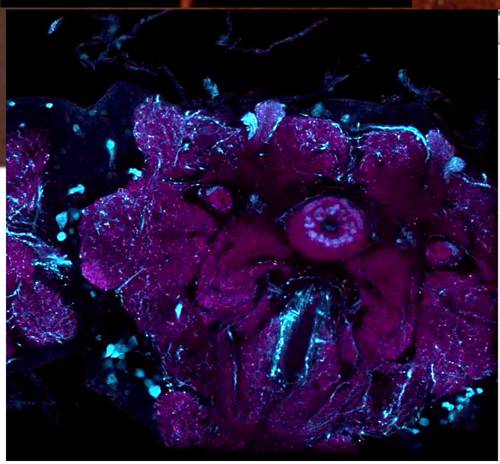


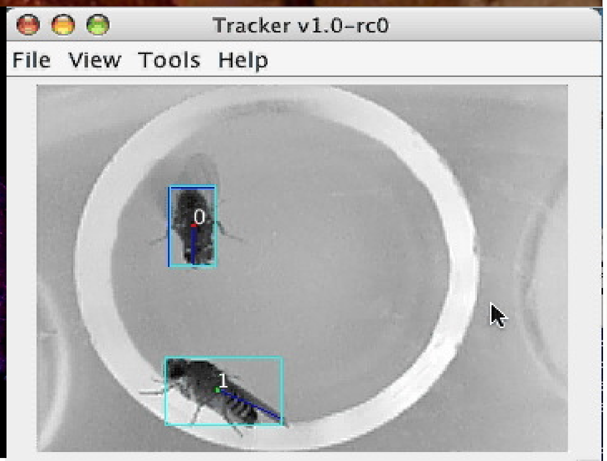
Diversity, Urgency, Complexity, …

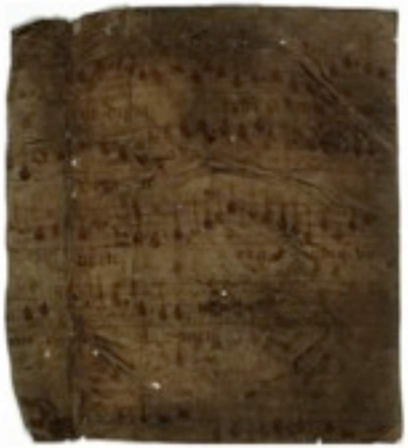
Diversity, Urgency, Complexity,

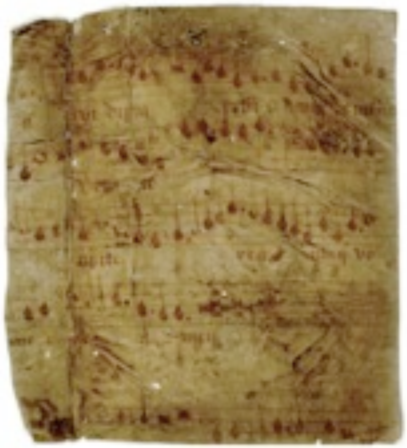
RESEARCH COUNCILS UK



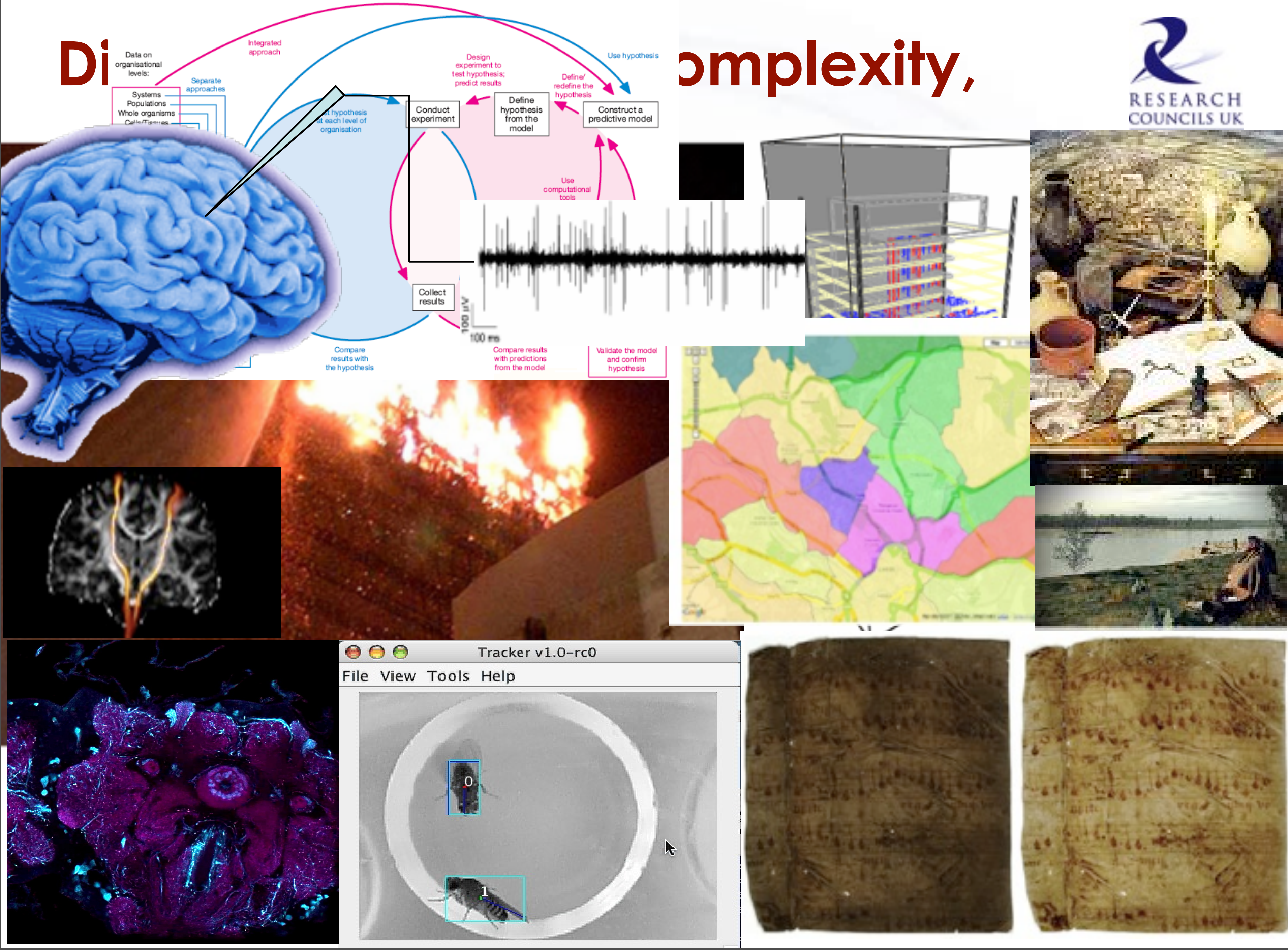








Di omplexity,

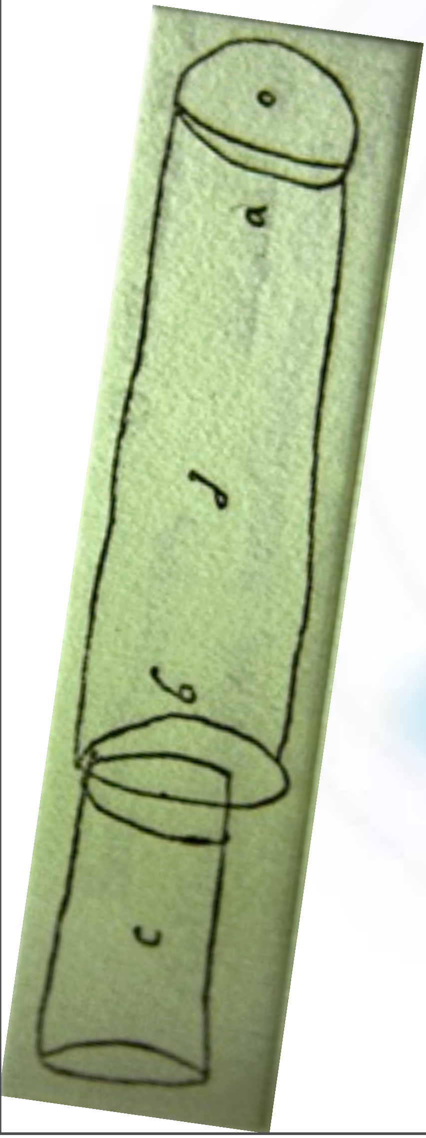


Tuesday, 17 July 12

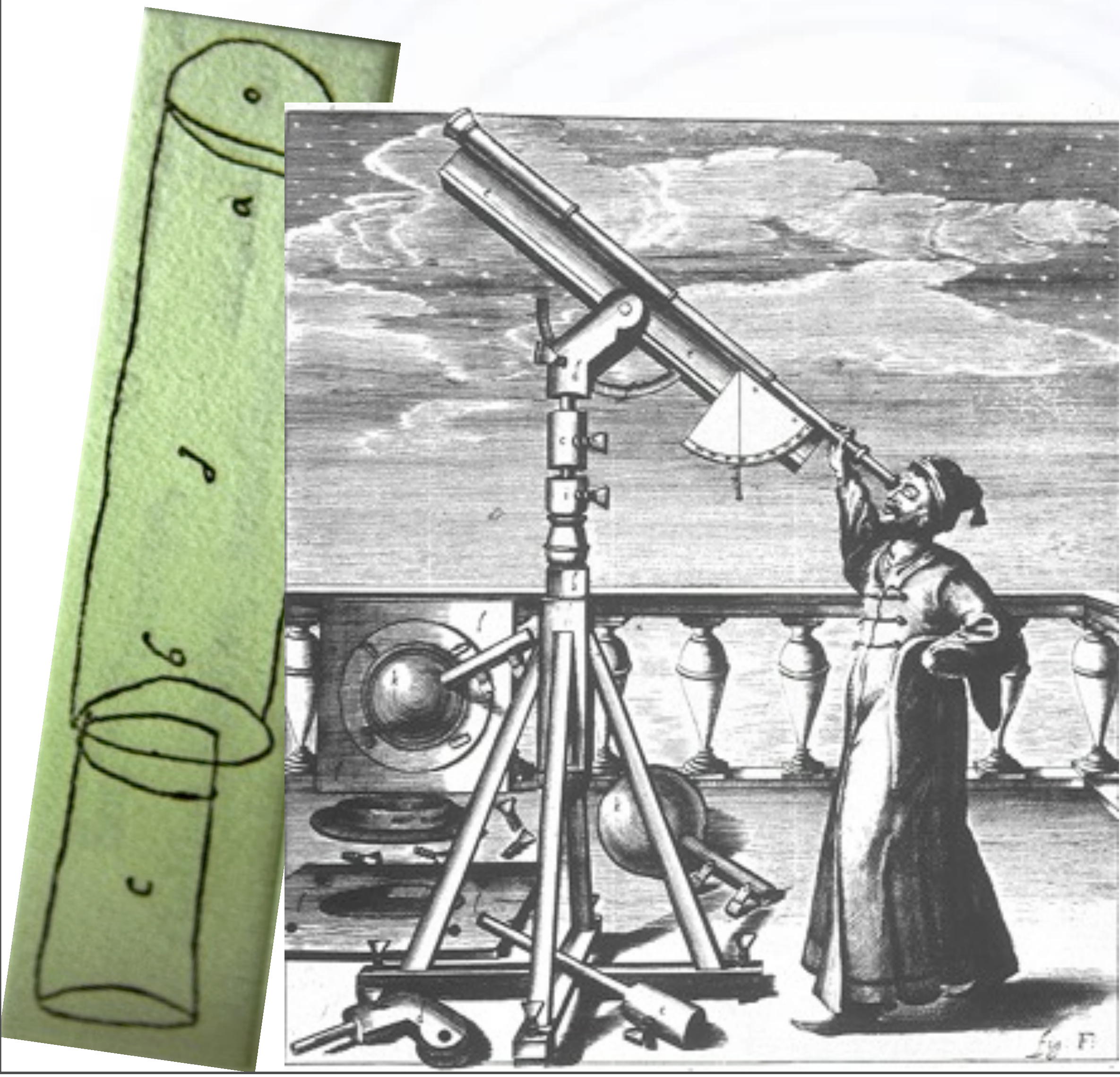
Datascopes for the naked mind

Datascopes for the naked mind

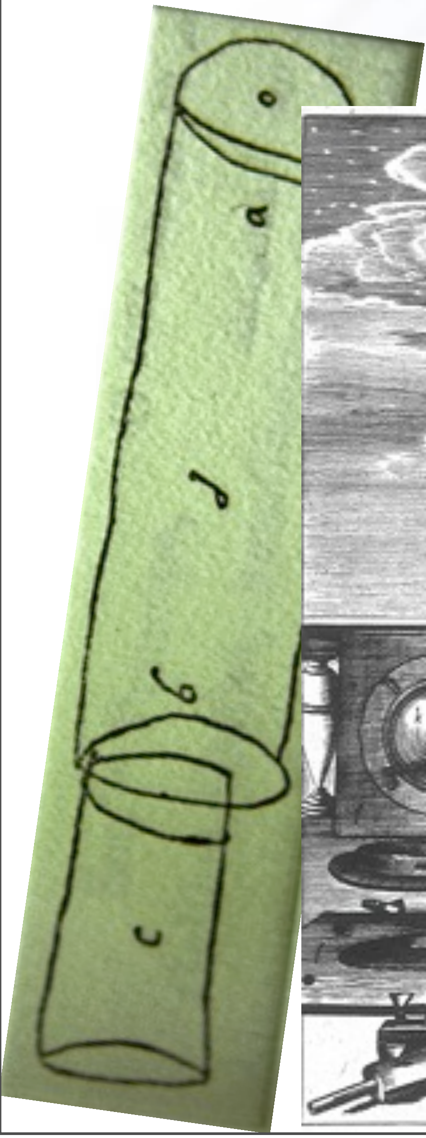
RESEARCH COUNCILS UK

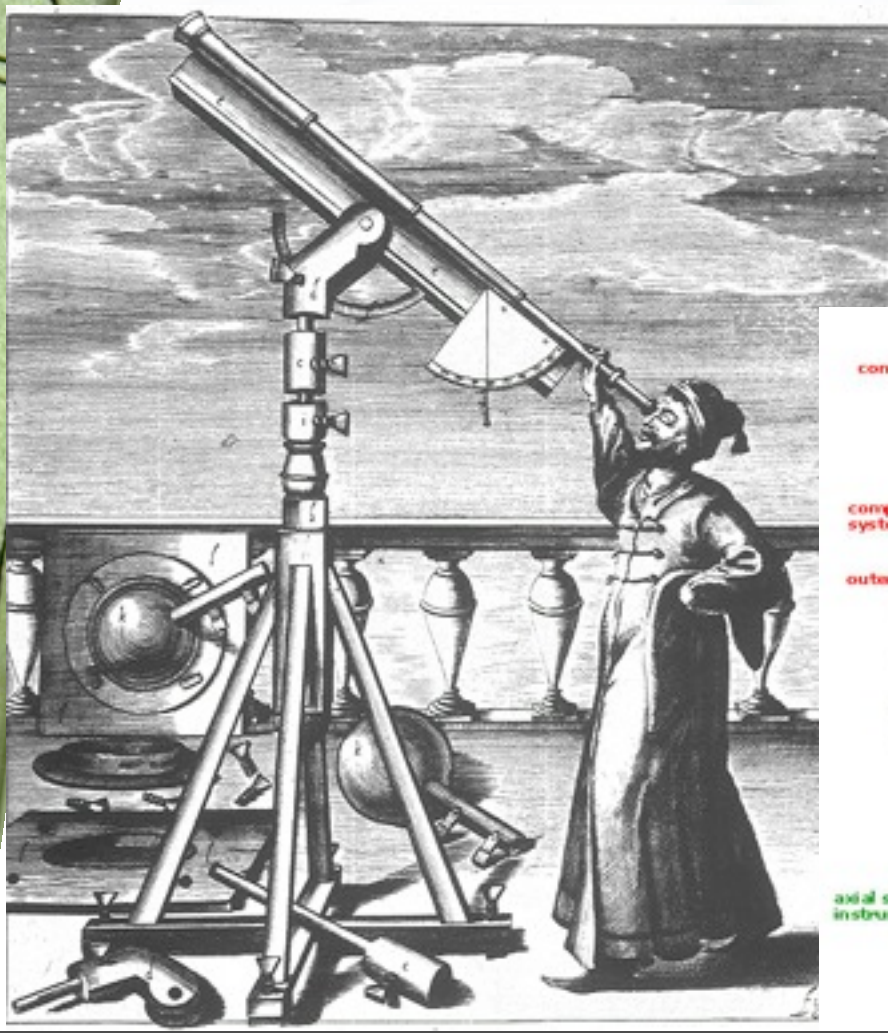


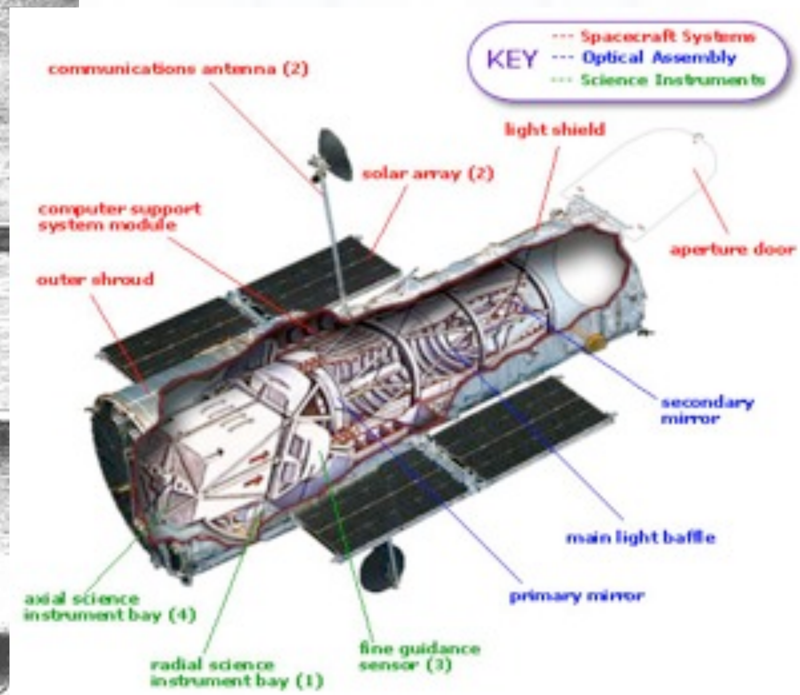
Datascopes for the naked mind



Datascopes for the naked mind

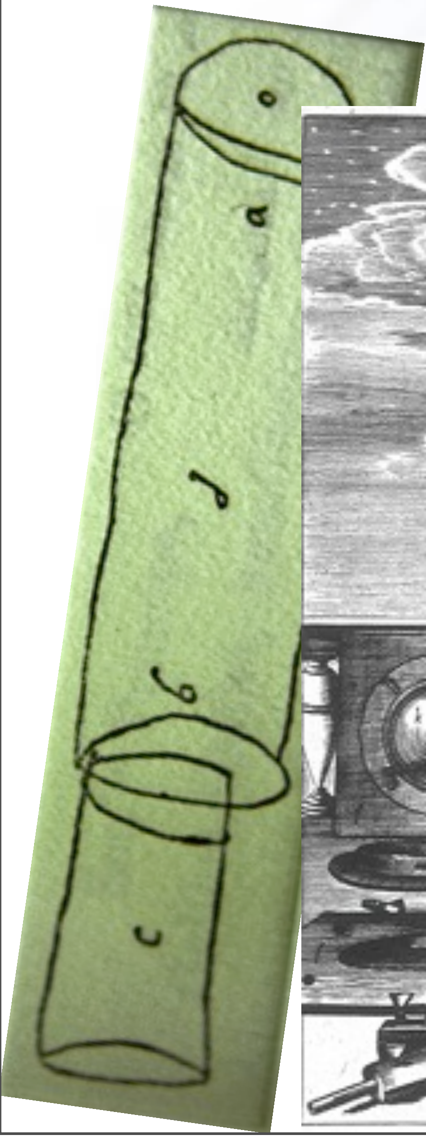


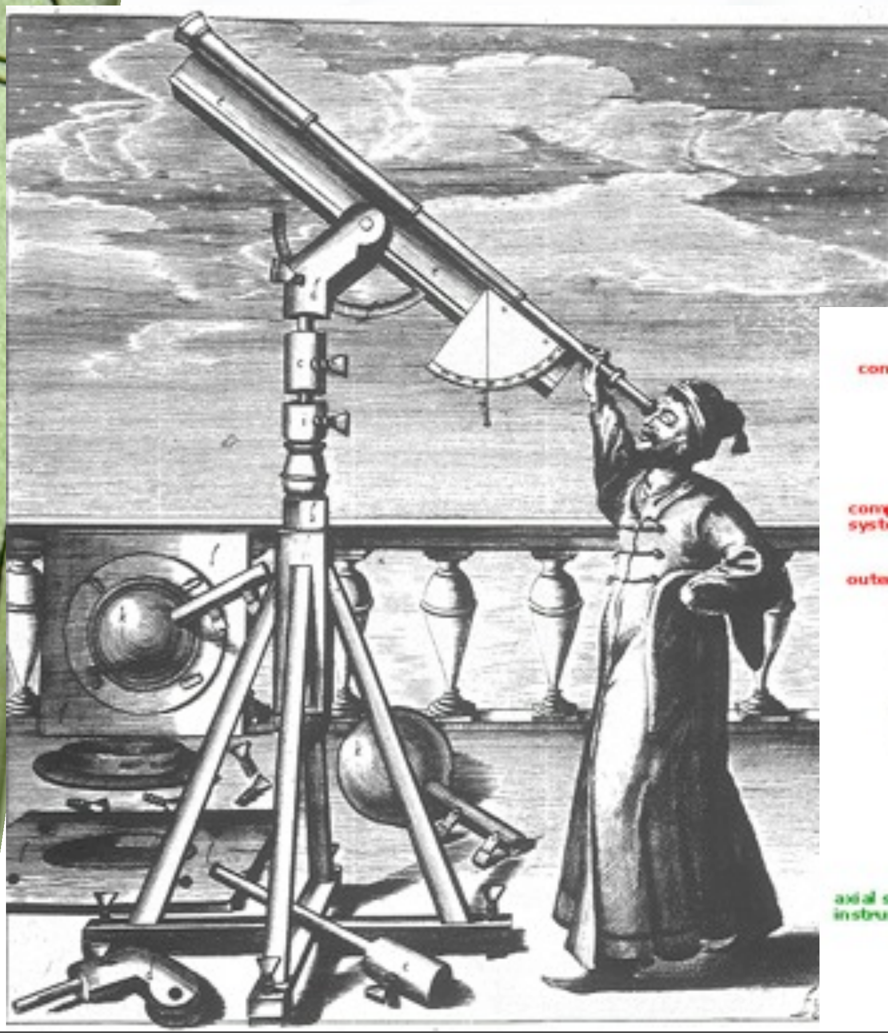


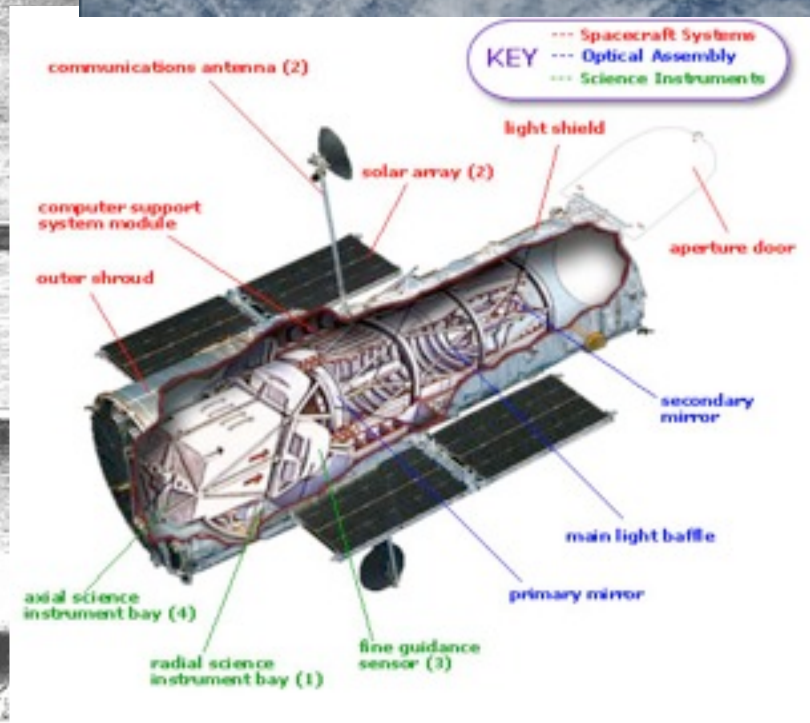


Datascopes for the naked mind

RESEARCH COUNCILS UK

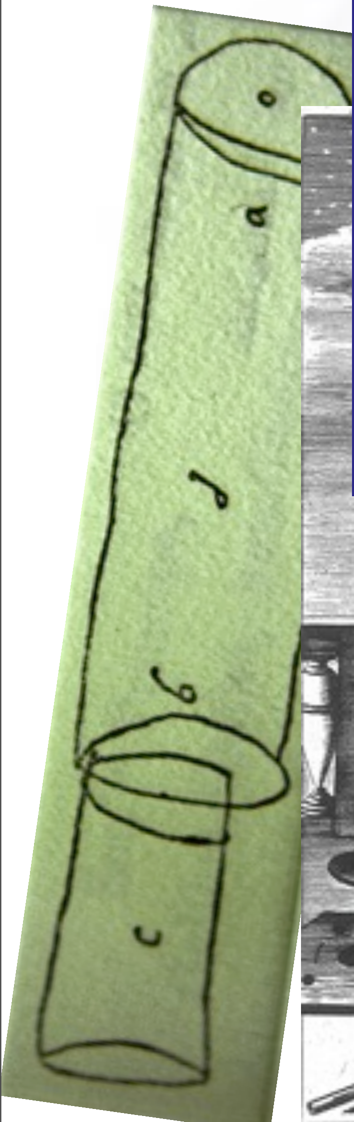


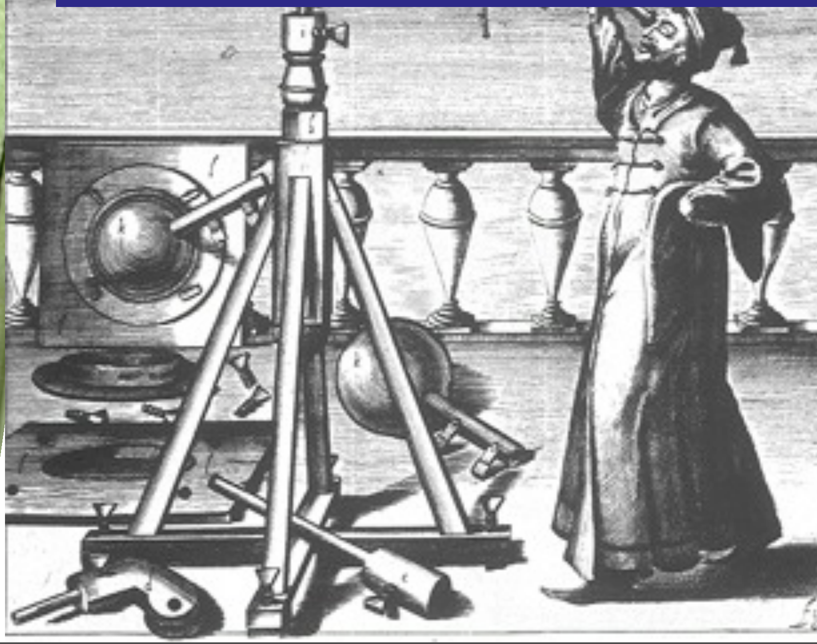


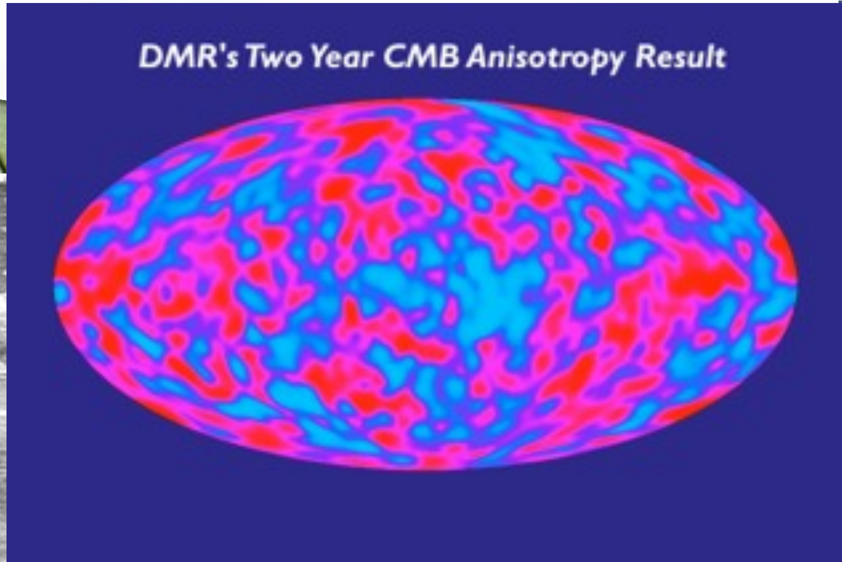


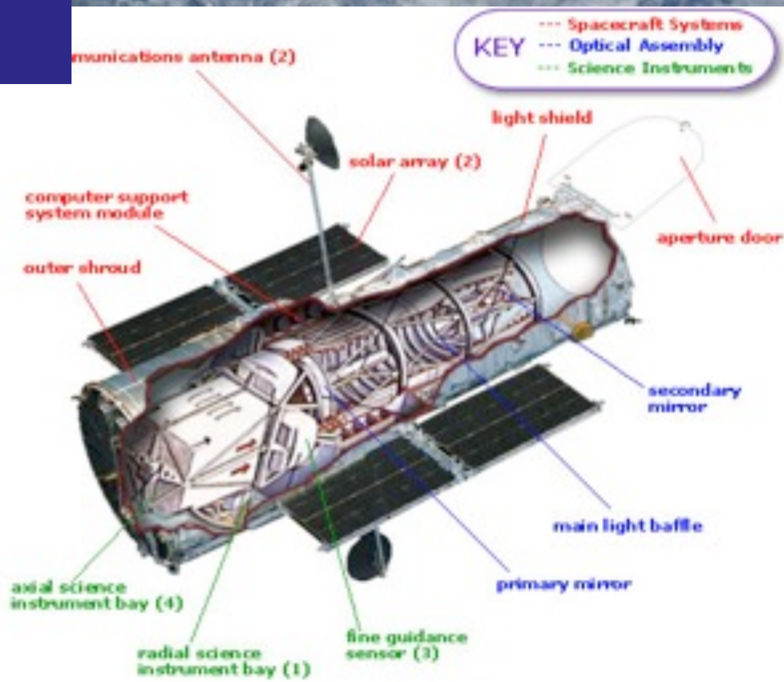
Datascopes for the naked mind

RESEARCH COUNCILS UK



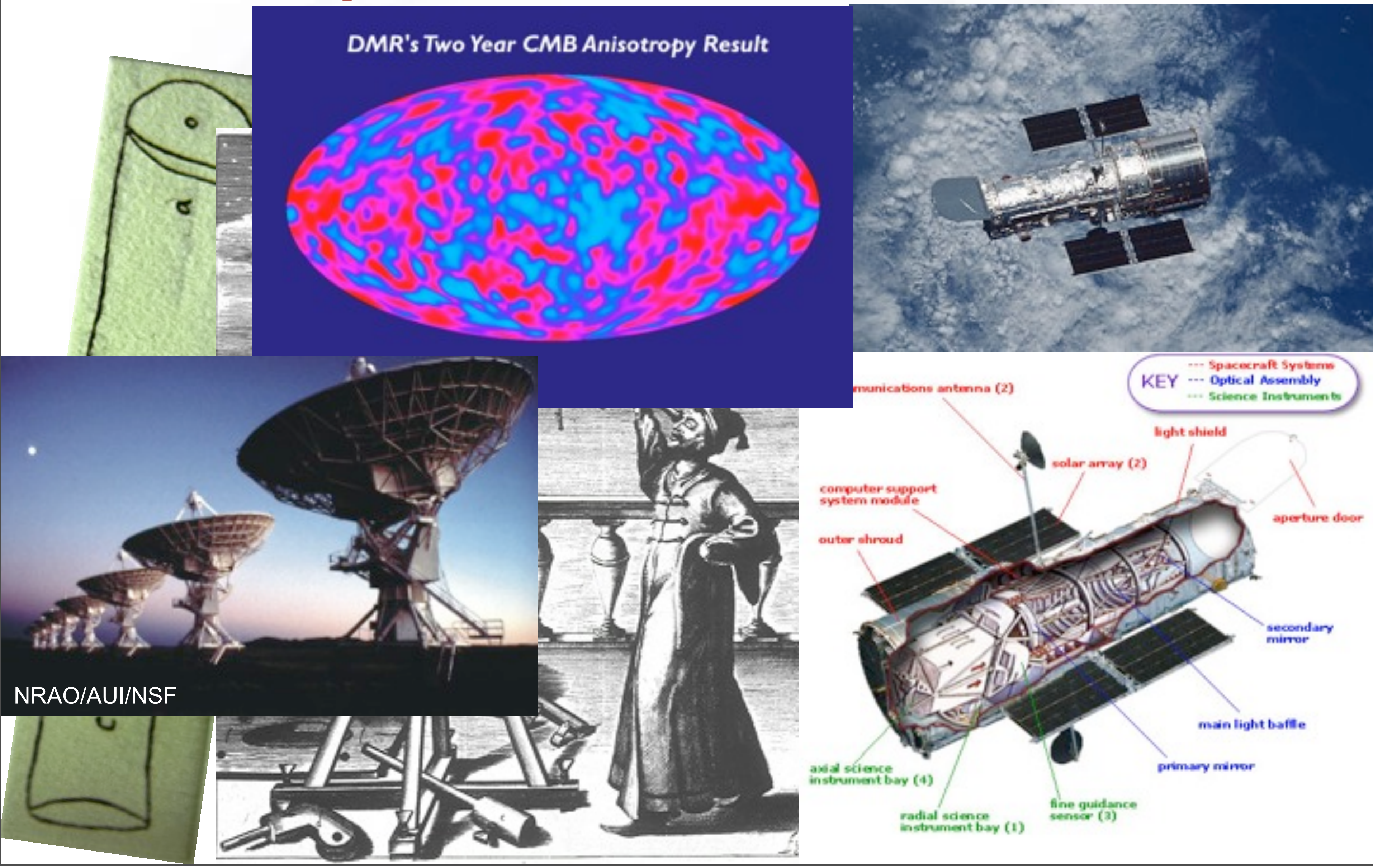






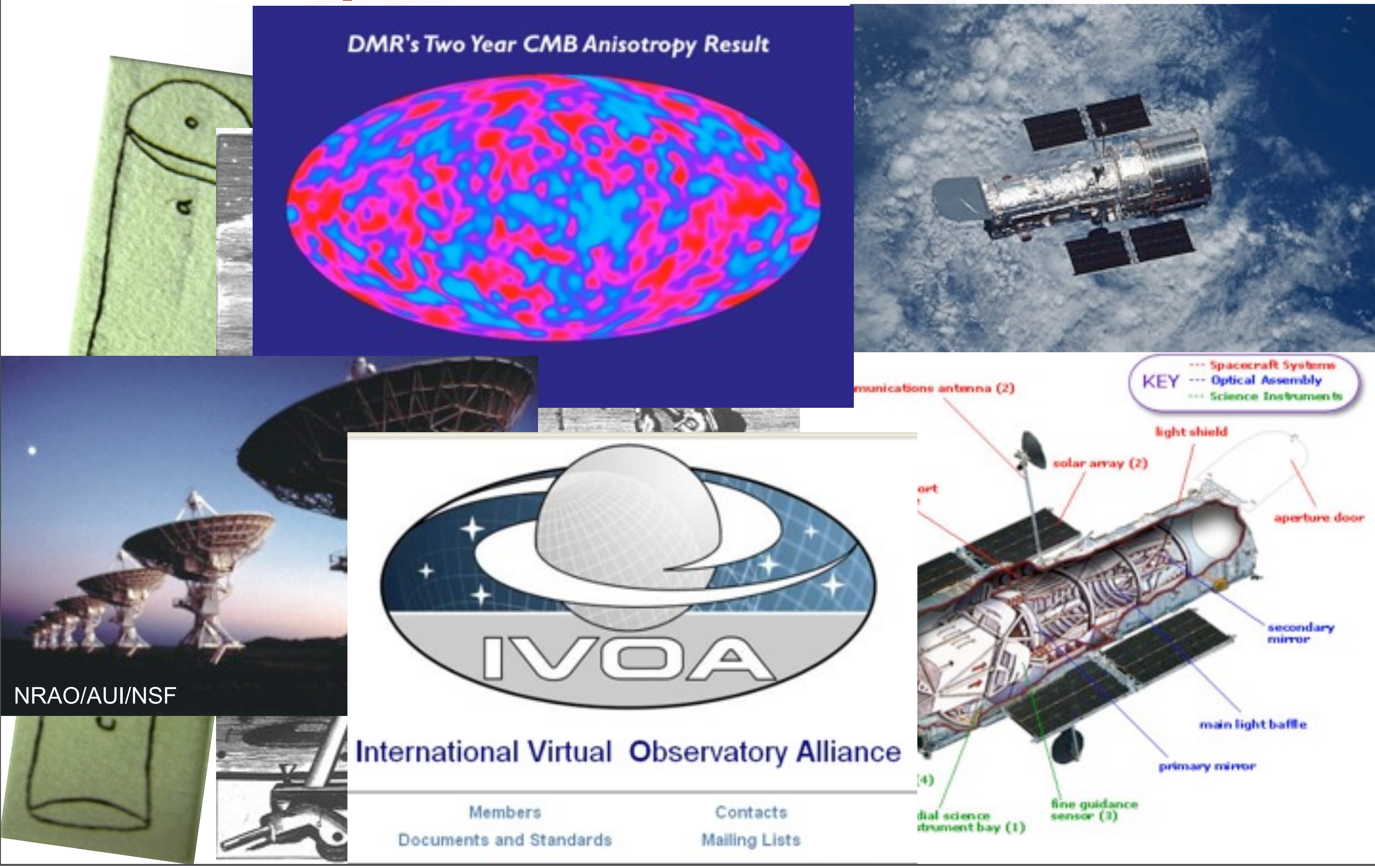
Datascopes for the naked mind

RESEARCH COUNCILS UK



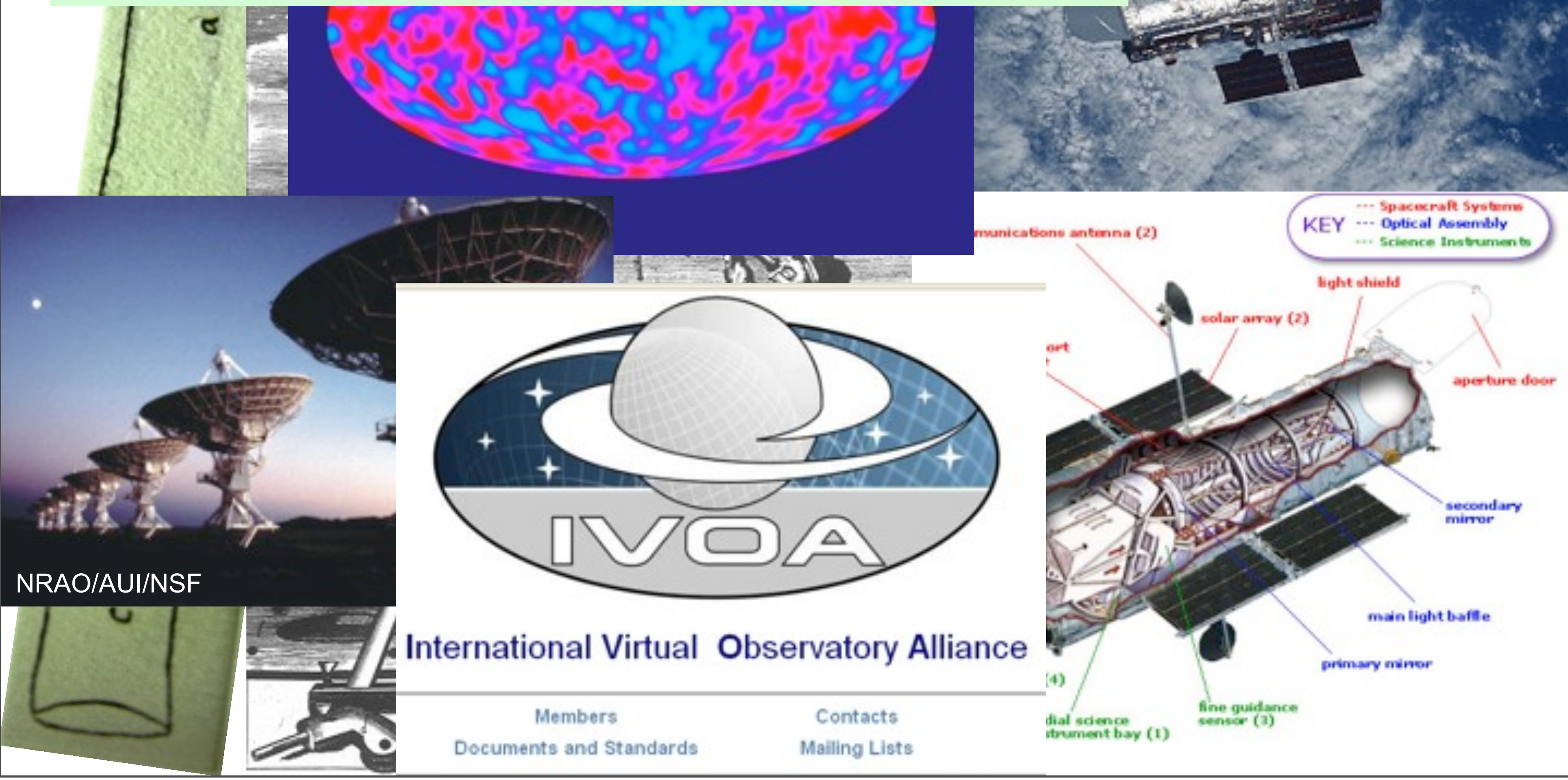
Datascopes for the naked mind

RESEARCH COUNCILS UK



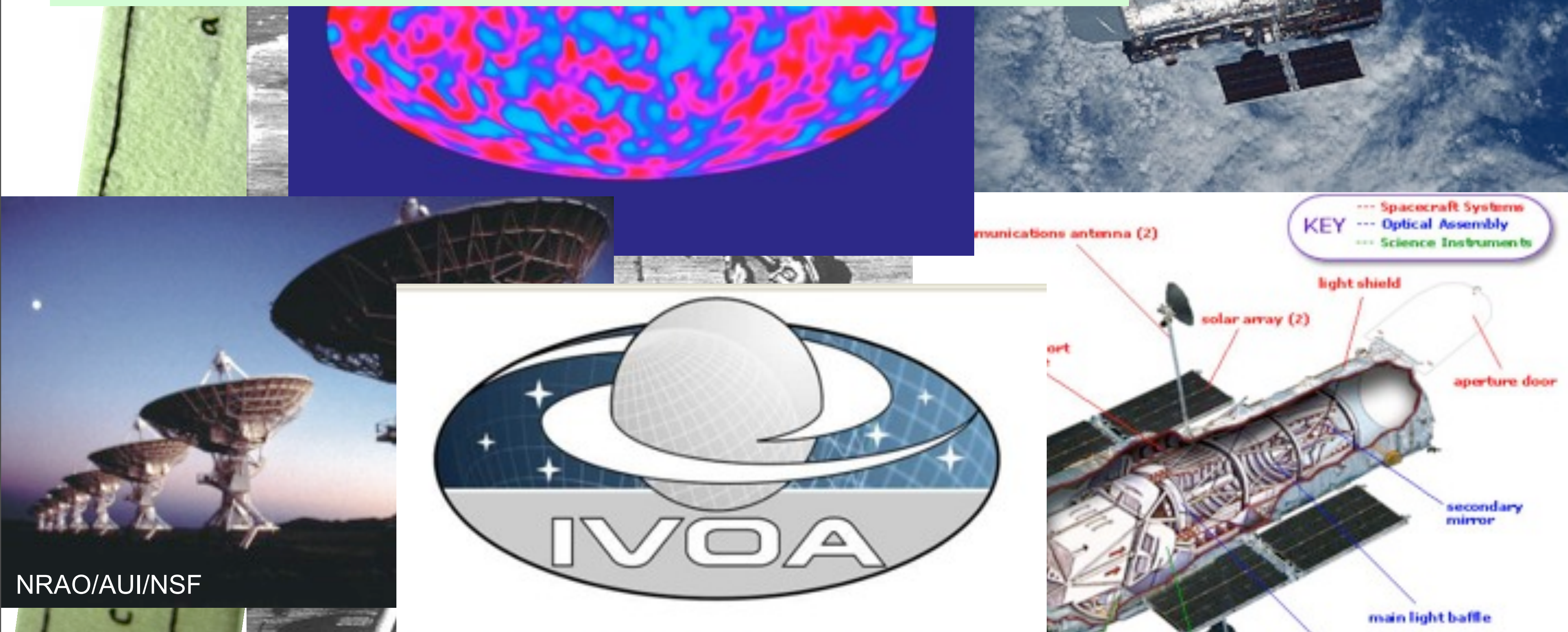
Datascopes for the naked mind

To reveal evidence in data you could never see before



Datascopes for the naked mind

To reveal evidence in data you could never see before

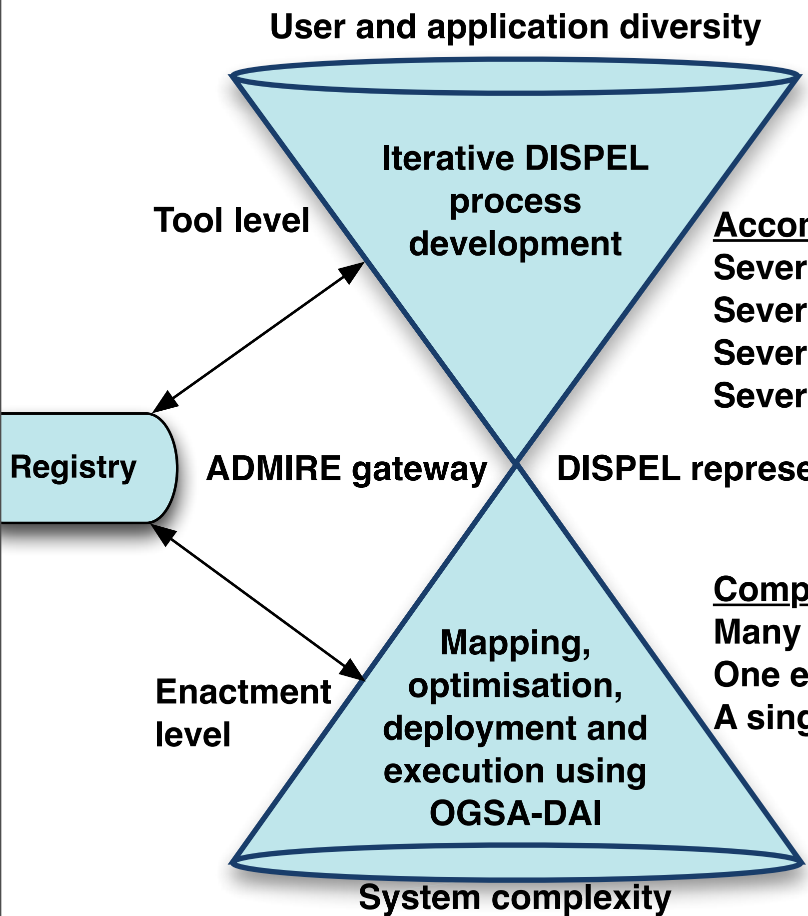


Changed our place in the universe

DISPEL

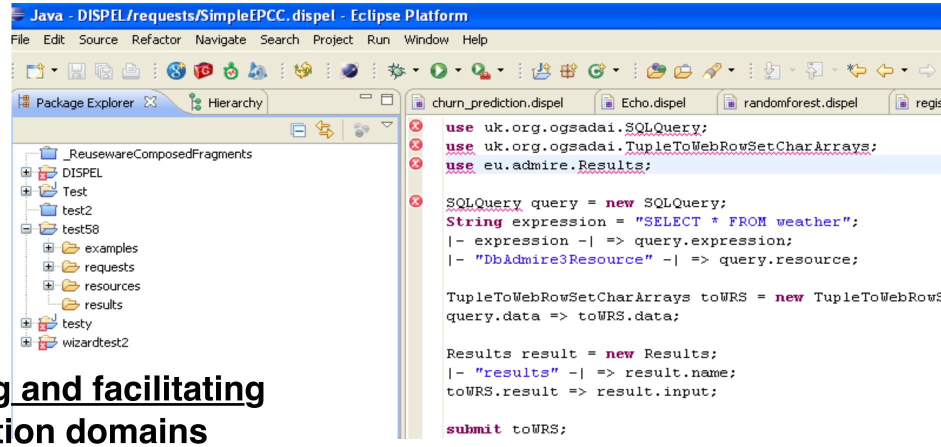
Data-Intensive Process Engineering Language

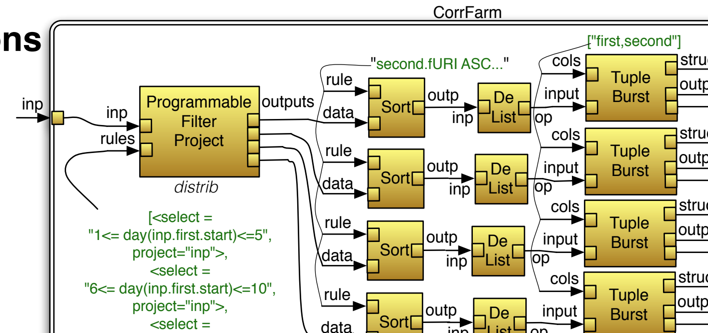
- A language for constructing data-flow graphs
  - Nodes are processing elements
  - Arcs are data-flow paths
- A language for generating data-flow patterns
  - Functions hide detail of graphs
  - Functions generate graphs
- A language for discussing data-flow engineering
  - Designed to be read and written by humans
  - As well as by programs
  - Supports validation and optimisation



Accommodating and facilitating
Several application domains
Several tool sets
Several process representations
Several working practices

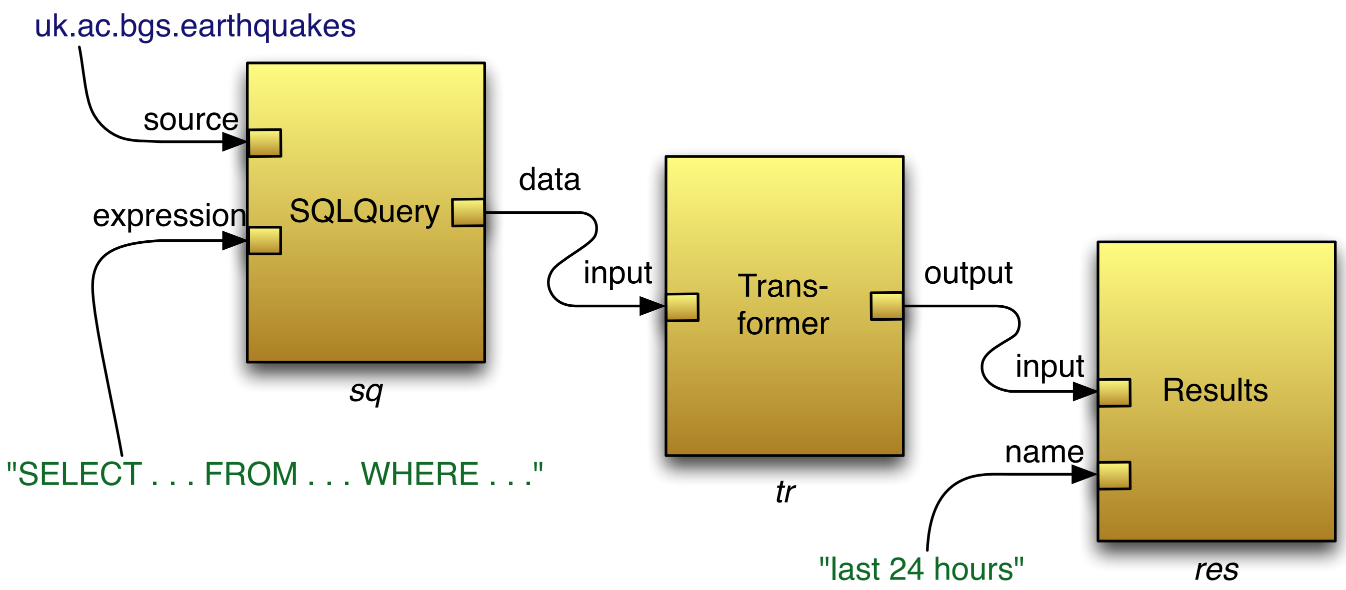
Composing and providing
Many autonomous resources
One enactment mechanism
A single platform





EDIM1

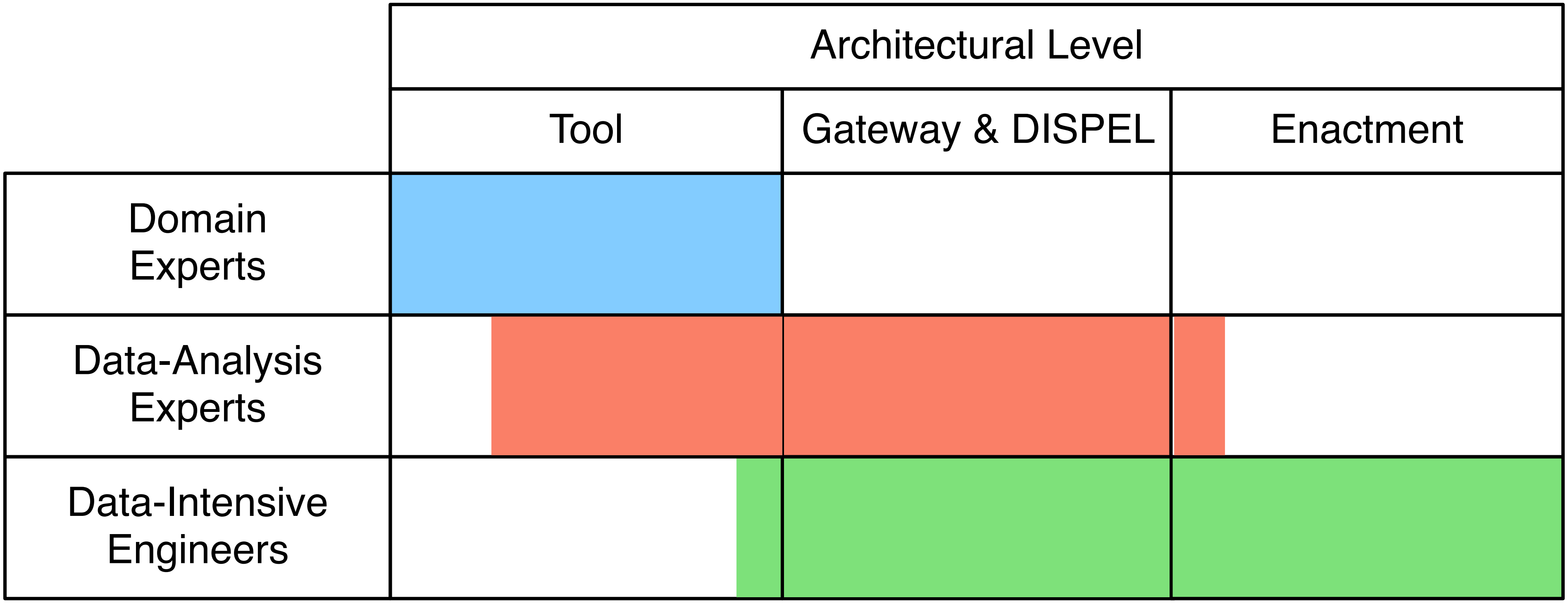
A simple DISPEL graph



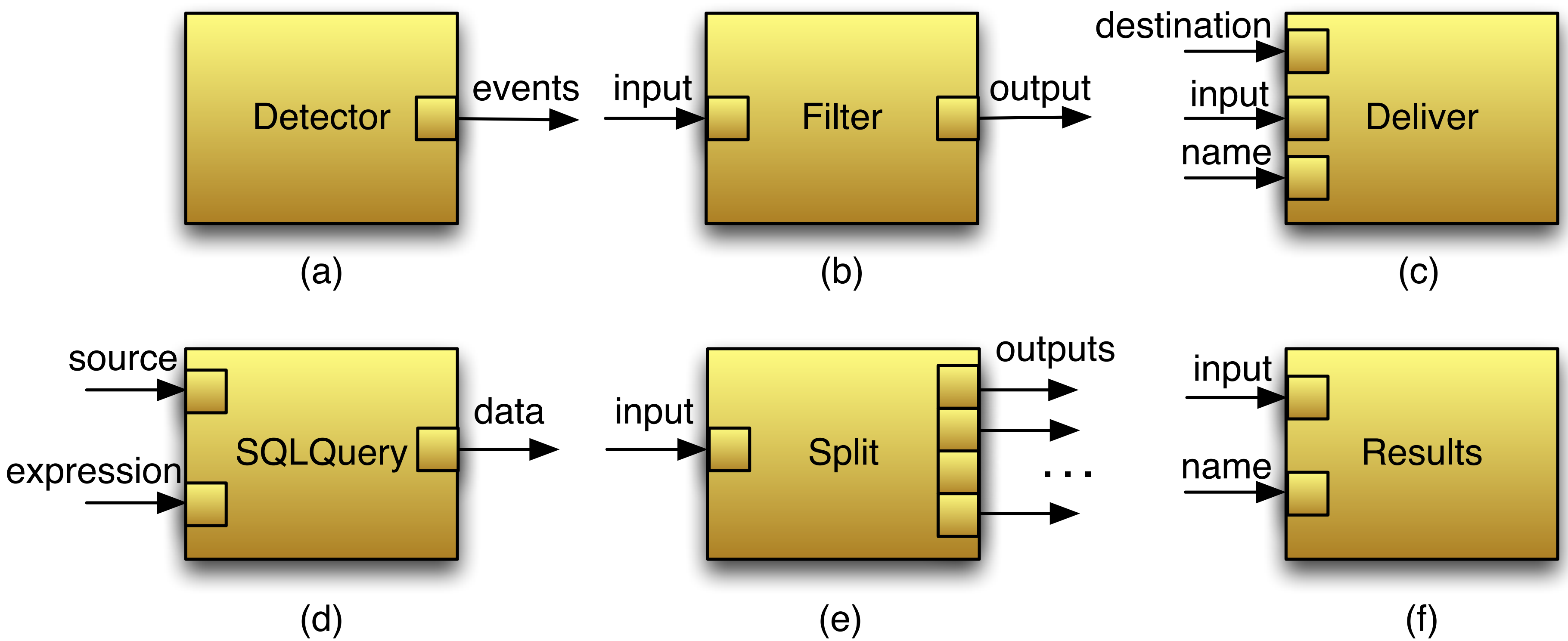
The DISPEL to Generate it

```
package book.examples.seismology {                    //set working context
  use dispel.db.SQLQuery;                             //import PE SQLQuery
  use book.examples.seismo.Transform;                 //import PE Transform
  use dispel.lang.Results;                            //import PE Results

  SQLQuery sq = new SQLQuery;                         // new instance of SQLQuery
  Transform tr = new Transform;                       // new instance of Transform
  Results res = new Results;                          // new instance of Results

  sq.data => tr.input;                                // set up data flow from sq to tr
  tr.output => res.input;                             // set up data flow from tr to res
  |- "uk.ac.bgs.earthquakes" -| => sq.source;         // URI of source of data
  |- "SELECT ... FROM ... WHERE ..." -| => sq.expression;    //query gets traces
  |- "last 24 hours" -| => res.name;                  //name of results for user

  submit res;                                         // submit for enactment
}
```

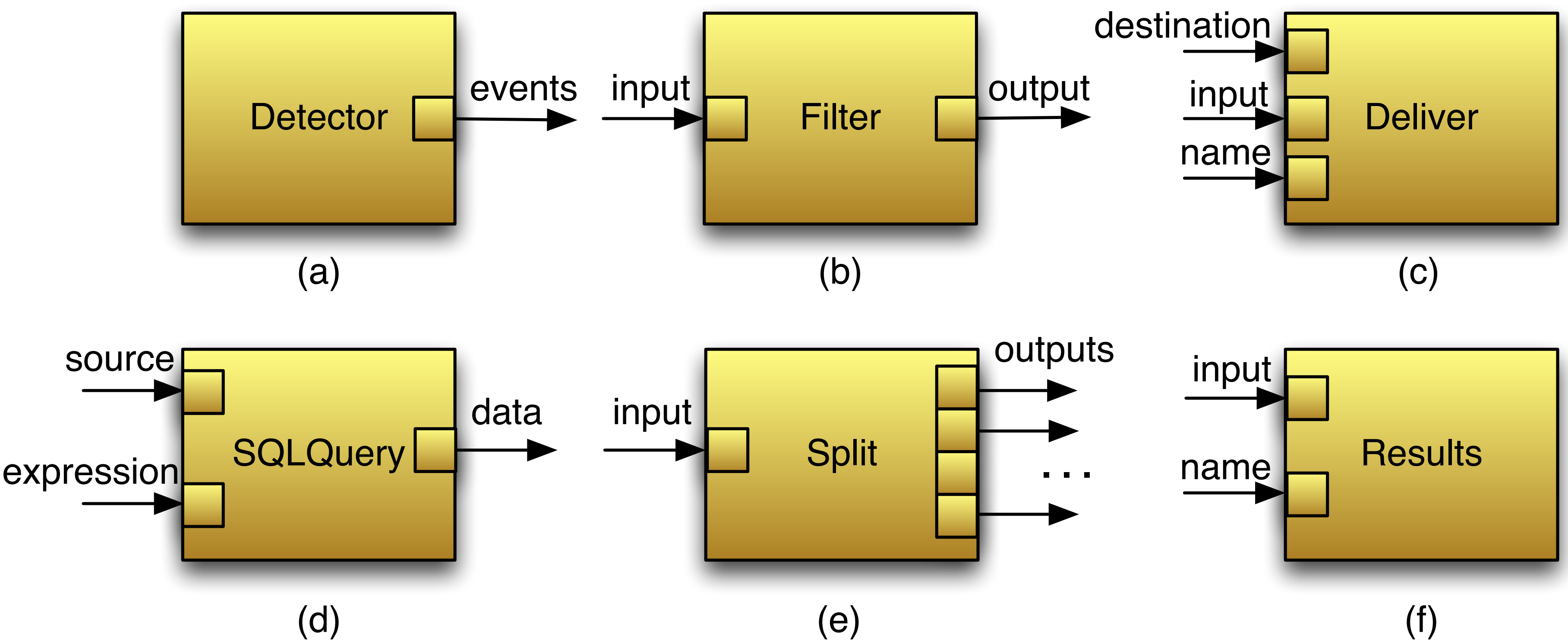
Who ‘speaks’ DISPEL



Processing Elements



Processing Elements



Expect well-organised libraries of well-described PEs

Description:

- names, inputs, outputs
- formats & meaning of each input and output
- auto-iteration behaviour, termination & errors
- optimisation properties
- use, relationships and logical properties

Functions

- Algorithms to generate graphs
  - parametric variation
  - patterns
  - parameters
  - subgraphs
- Abstraction and Optimisation
  - smart methods for common patterns
  - hiding pattern implementation for stability
  - late evaluation permits contextual optimisation

Functions

- Algorithms to generate graphs
    - parametric variation
    - patterns
    - parameters
    - subgraphs
- Abstraction and Optimisation
    - smart methods for common patterns
    - hiding pattern implementation for stability
    - late evaluation permits contextual optimisation

Expect well-organised libraries of well-described Functions
Description:
- names, type signature

Enactment Model

1. DISPEL language processing
   1.1. Validation & Import from Registry
   1.2. Format & Meaning mis-match handling
   1.3. Interpretation to generate graph
2. Graph optimisation & mapping
   2.1. Re-ordering & Parallelisation
   2.2. Identification of target locations
   2.3. Selection of PE implementations / instances
   2.4. Partitioning into co-located subgraphs
3. Deployment
4. Execution, Monitoring & Clean up

DAGMan

Condor
High Throughput Computing

VisTrails

meandre

swift

Kepler

MOTEUR

pegasus

Taverna

Trident
Microsoft Research

TRIANA

ASKALON

KNIME

Ian J. Taylor, Ewa Deelman, Dennis B. Gannon, and Matthew Shields.
Workflows for e-Science: Scientific Workflows for Grids. Springer London, 2007.

DISPEL is Different 1

- Spanning Distributed Independent Hosts
  - Fragments of one workflow can run in different regimes
  - Different security models
  - Different file systems
  - Different DBMS
  - Different Operating Systems
  - Different DISPEL implementations
- Agnostic about Size & Scale
  - Processing Elements of any size
  - Data values in streams of any size
  - Streams of any length
  - Graphs of any size

DISPEL is Different 2

- Patterns & Pattern Composition
  - Functions define & generate patterns
  - Higher-order functions compose patterns
  - Functions can be refined to optimise
- Component-Description Driven
  - Rich description of components
  - Capturing logical properties
  - Collecting component-builders' hints
- Restricted language for workflow longevity
  - Only hints and *no* WF-definition time concrete mappings
  - Late mapping permits optimisation and enactment, *for the system is at execution time* - much different from definition time!

Summary and Conclusions

Summary

Summary

- DISPEL is an experimental data-intensive language
  - draws on workflows & database query internals
  - auto-iteration over values flowing through connections
  - agnostic about value sizes - implementation challenge
  - controlled access to system information
  - optimisation based on description & operation
  - distributed termination protocol

Summary

- DISPEL is an experimental data-intensive language
  - draws on workflows & database query internals
  - auto-iteration over values flowing through connections
  - agnostic about value sizes - implementation challenge
  - controlled access to system information
  - optimisation based on description & operation
  - distributed termination protocol
- Several years of experience
  - seven different application domains

Summary

- DISPEL is an experimental data-intensive language
  - draws on workflows & database query internals
  - auto-iteration over values flowing through connections
  - agnostic about value sizes - implementation challenge
  - controlled access to system information
  - optimisation based on description & operation
  - distributed termination protocol
- Several years of experience
  - seven different application domains
- Differences
  - functional pattern handling
  - multi-scale streams
  - restricted information to permit platform evolution

Summary

- DISPEL is an experimental data-intensive language
    - draws on workflows & database query internals
    - auto-iteration over values flowing through connections
    - agnostic about value sizes - implementation challenge
    - controlled access to system information
    - optimisation based on description & operation
    - distributed termination protocol
- Several years of experience
    - seven different application domains
- Differences
    - functional pattern handling
    - multi-scale streams
    - restricted information to permit platform evolution
- Status
    - two implementations: to OGSA-DAI & to Java
    - much still to do to fully explore the ideas

?

www.verce.eu

research.nesc.ac.uk/node/828

www.ogsadai.org.uk

omii-uk

www.omii.ac.uk

epcc

Picture composition by Luke Humphry based on prior art by Frans Hals