Credit: K. Geary

Robotic astronomy, Data managment, the Virtual Observatory, and software for small astronomy research groups.

Ronan Cunniffe
(Blackrock Castle Observatory,
Cork Institute of Technology)

Malaga 19/05/2009

How similar are the data archive needs of many/most/all astronomy research groups?

A searchable dataset, management tools;
Tolerant in what it can accept;
Standards-compliant in what it returns;
Easily built on, to connect pipelines to;

Can I write a piece of archive software, installed and configured like a webserver that, for most people, Just Works™?

Cork, BOOTES, ... others?

	BCO (CIT)	**BOOTES (IAA)**
Images	750k(->5M)	>1M
Volume	500GB(->10-20TB)	>1TB
FITS types	8	?
Pipeline	Yes	>1?
Search via FS?	Yes (→ No)	...complicated!

New Instrument for BCO: ToΦcam (“ToffeeCam”):

- 2 channel photometer (FT cameras, differential photometry)
- 1-5 frames/sec....
- =100,000+ frames/night (0.5-1TB/night)
- Raw images → Reduced images → lightcurves
- Need for spot-checking raw & reduced images... clumsy!

How many pieces of the puzzle?

Scale:

- It must be easy to add extra capacity

Multi-dialect:

- It must handle any kind of FITS file *(and be able to search everything from a single query)*

Hold data products too:

- know (or at least remember if told) which files were reduced from which others, and how.

Easily worked with:

- wrappers need to be easy to write

No, everything won't be the same!

Holding the data

Database contains:

- **headers only... nothing gained storing images**
- **references to where images are stored**

Means all requests are 2 phase:

1: Query the DB, get references (URLs!)
2: Retrieve from data servers (web servers!)

🙂

😐 **Scalability - just add web servers**

😐 **Performance = network speed (*)**

☹ **Performance = network latency (*) (always cache!)**

Compatibility - breaks every script you have....

(but wrapping IRAF with wget/curl isn't *that* hard!)

(* If number-crunching machine is also a repository, store popular data there, and get higher performance)

Holding the data

Database contains:

- **headers only... nothing gained storing images**
- **references to where images are stored**

Means all requests are 2 phase:

1: Query the DB, get references (URLs!)
2: Retrieve from data servers (web servers!)

🙂

😐 **Scalability - just add web servers**

🙁 **Performance = speed of network**

Compatibility - breaks everything!

(but is writing wget/curl wrappers around IRAF *that* hard?)

FITS of recognition!

Dialect “A”:

```
...
EXPSECS  =     5.0 / Exposure time in seconds
FILTER   =       3 / Filter slot number
BAND     = '    R' / Sloan filter
```

...

Dialect “B”:

```
...
EXPOSURE =      10 / Exposure time
FILTER   = '    R' / Filter
FILTERNO =       3 / Filter slot number
...
```

A standard exists (for 53 basic headers), but many, many FITS files do not follow it.

FITS of recognition!

Dialect “A”:

```
...
EXPSECS  = <float> / Exposure time in seconds
FILTER   =   <int> / Filter slot number
BAND     =   <str> / Sloan filter
```

...

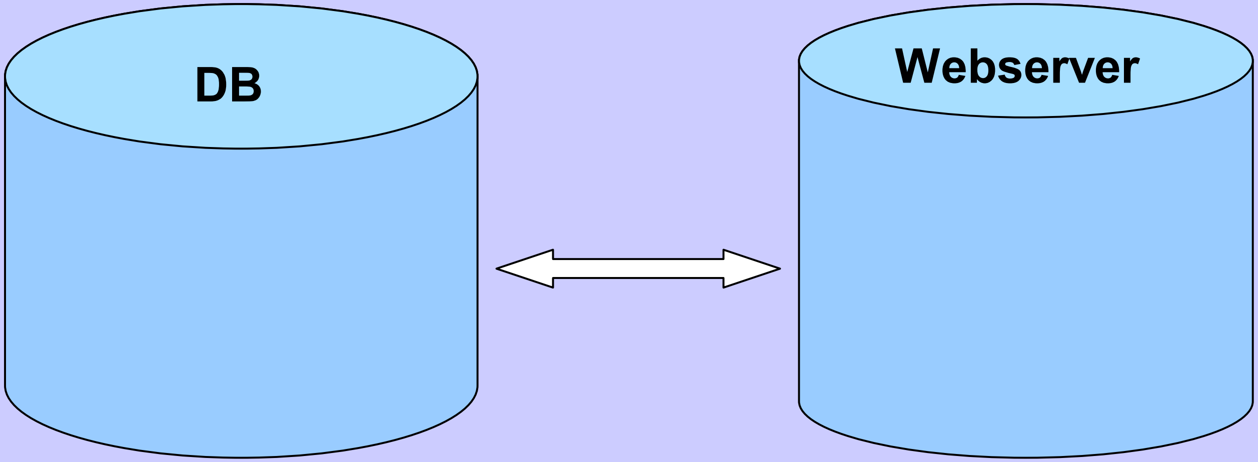
Dialect “B”:

```
...
EXPOSURE =   <int> / Exposure time
FILTER   =   <str> / Filter
FILTERNO =   <int> / Filter slot number
...
```

We can reasonably expect that *the full set of keywords in a FITS header are unique fingerprints of the software that wrote it.*

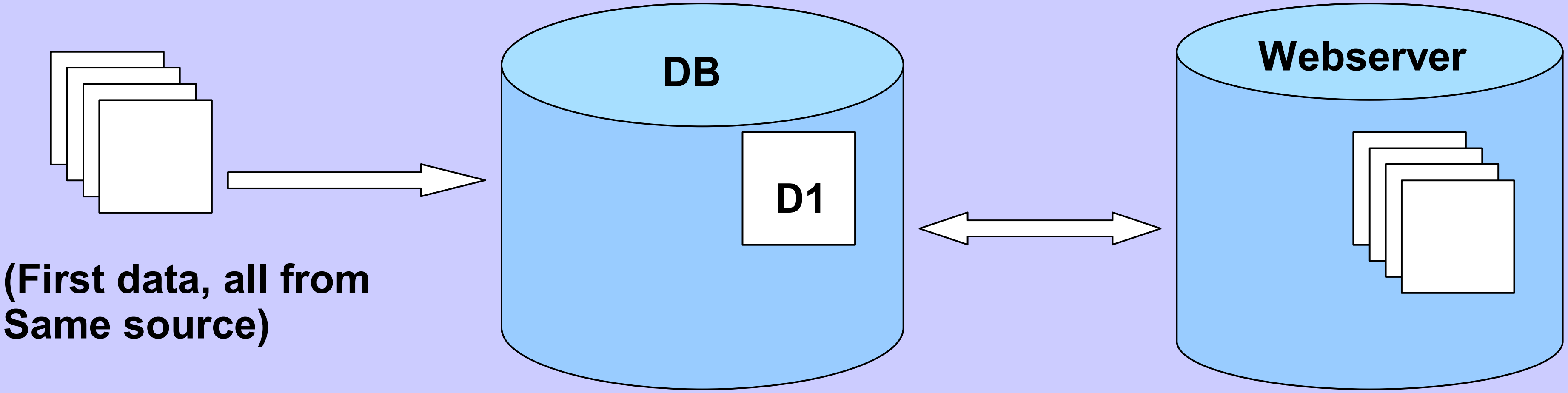
(okay, only *probably* unique..,)

From the beginning...



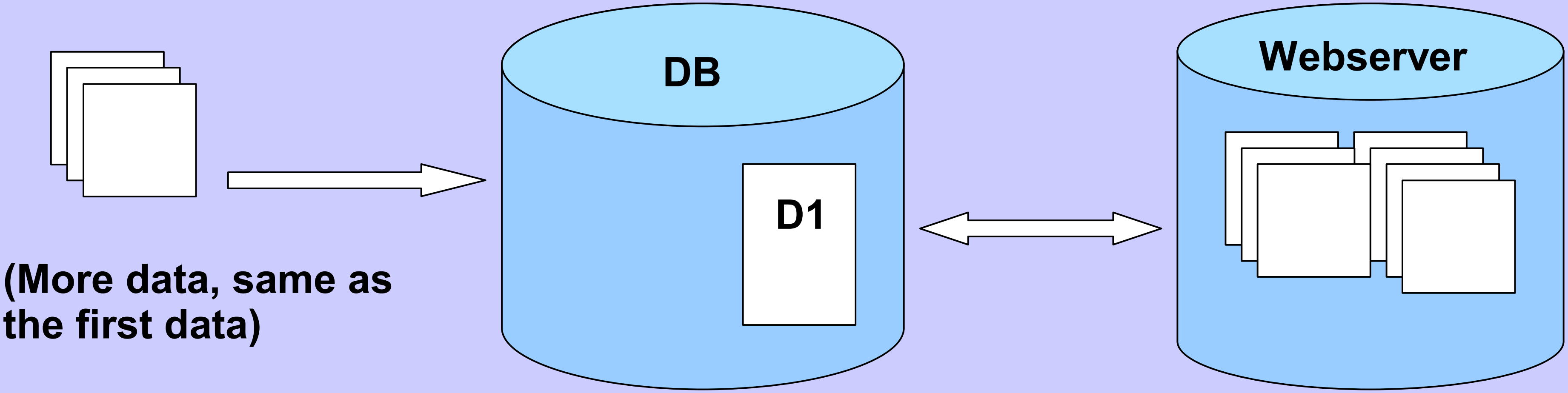
(Both empty)

First data....



Webserver: has copies of the files
DB: has pattern for Dialect 1

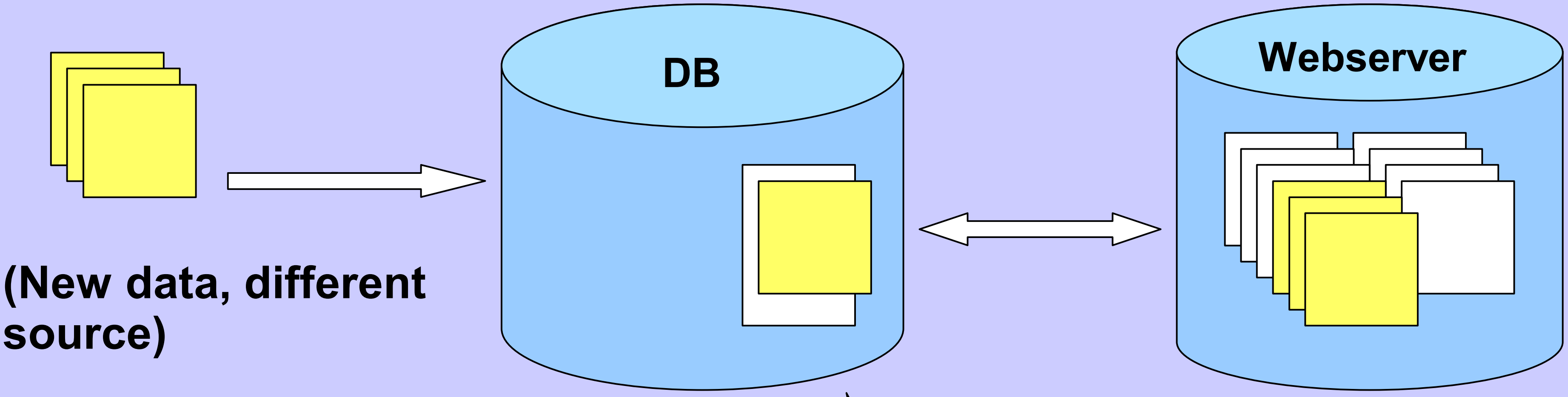
More of the same...



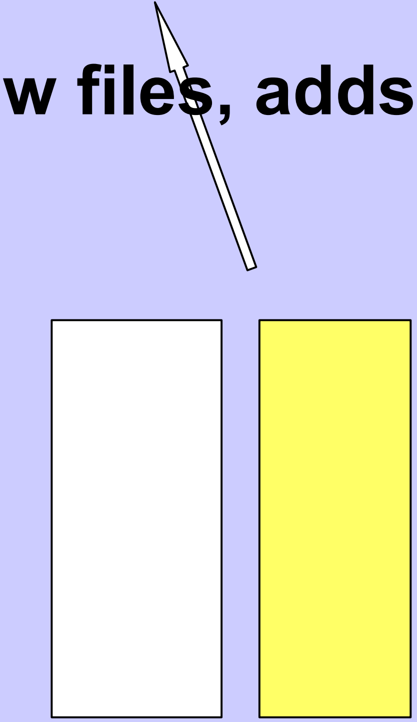
(More data, same as the first data)

DB recognises new files as Dialect 1
(extra rows in table, no new structure)

New Dialect

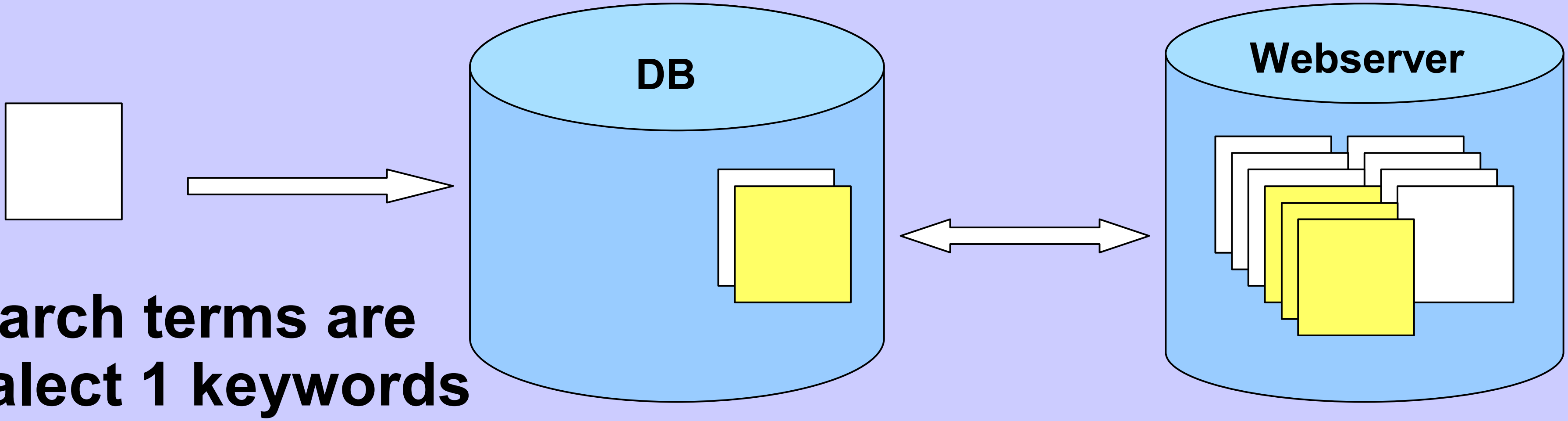


DB does *not* recognise new files, adds new structure....



A *human* needs to compare them, builds translation table

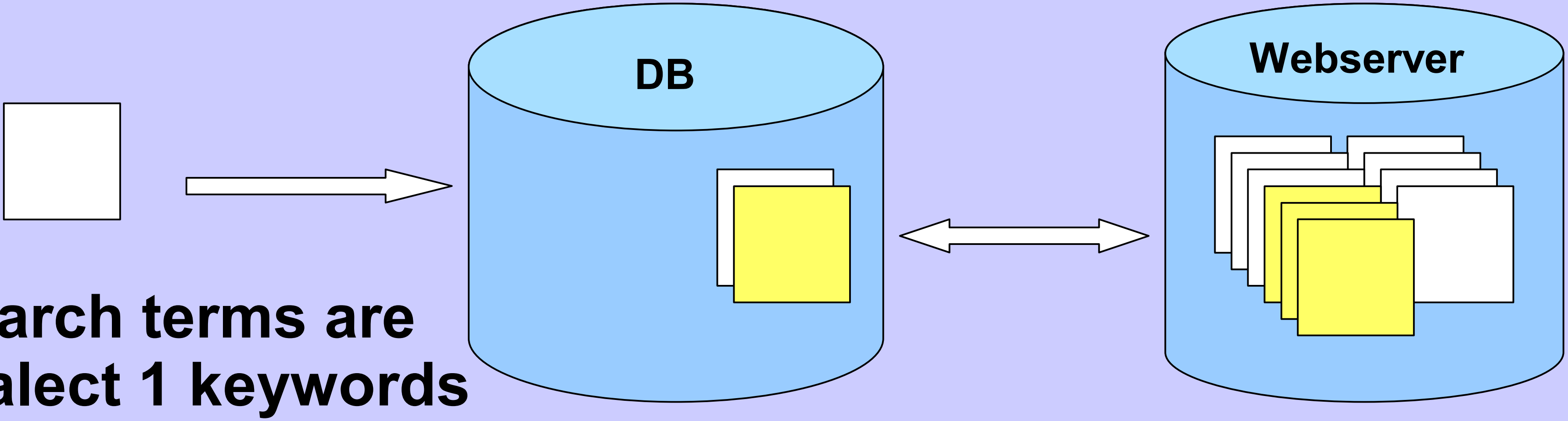
Query.....



Search terms are Dialect 1 keywords

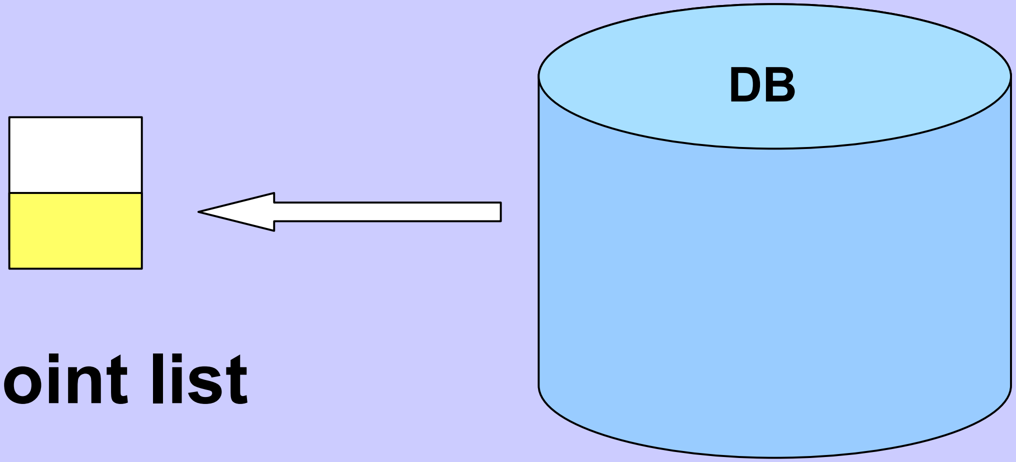
DB searches *both* dialects (because it now knows how to translate)

Query and reply.....



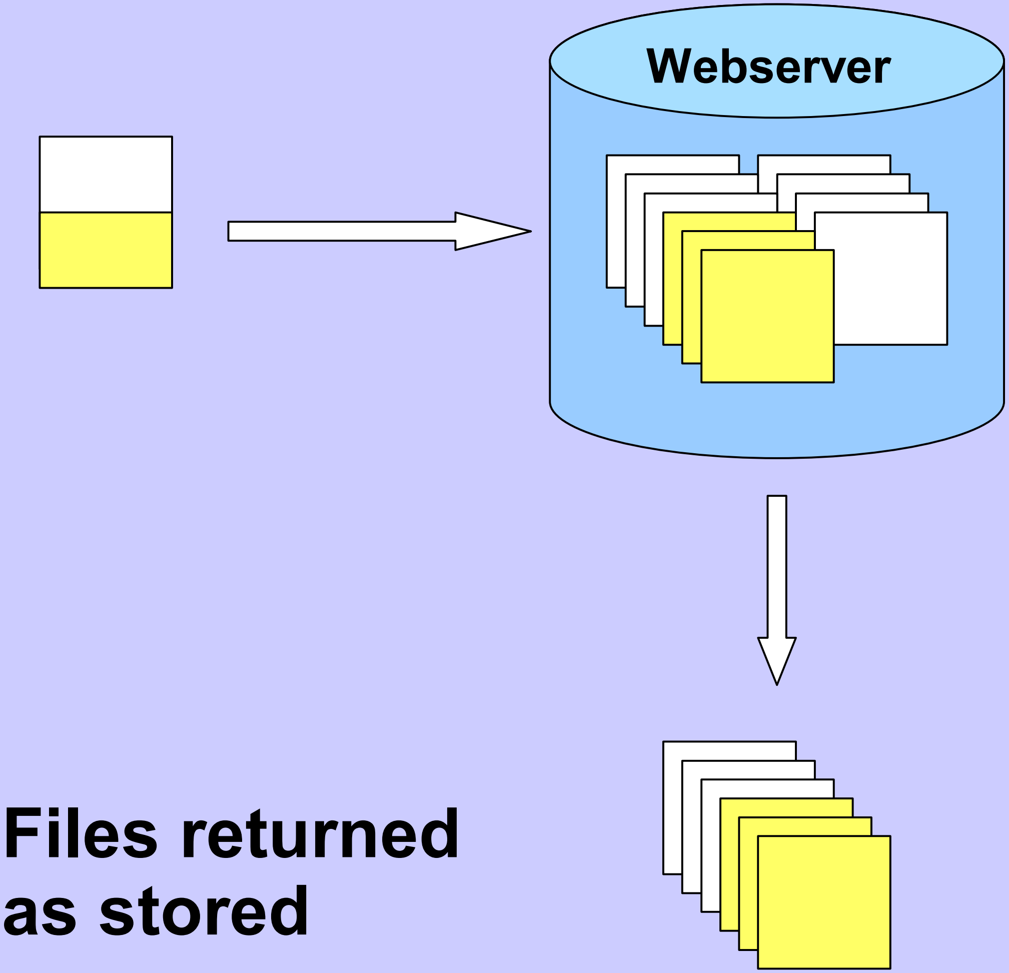
Search terms are Dialect 1 keywords

DB searches *both* dialects (because it can translate)



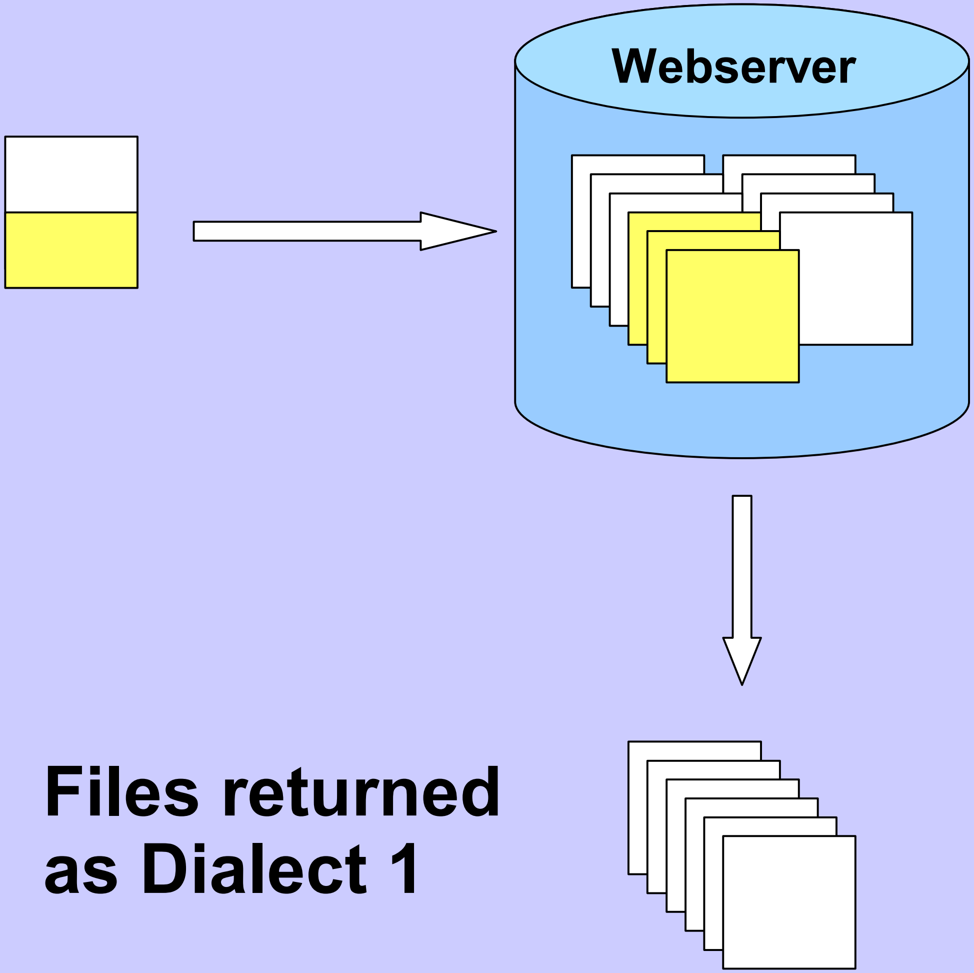
Joint list

Results....



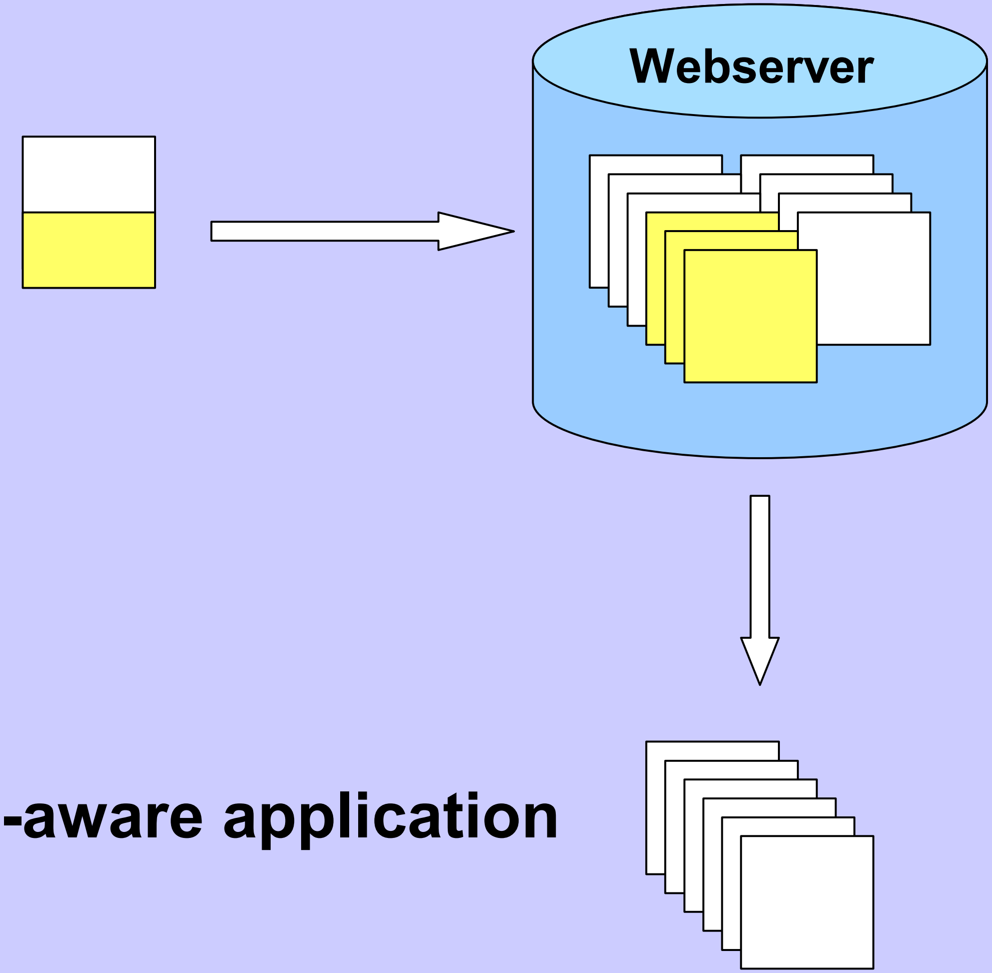
Files returned as stored

...or Results (Translated!)



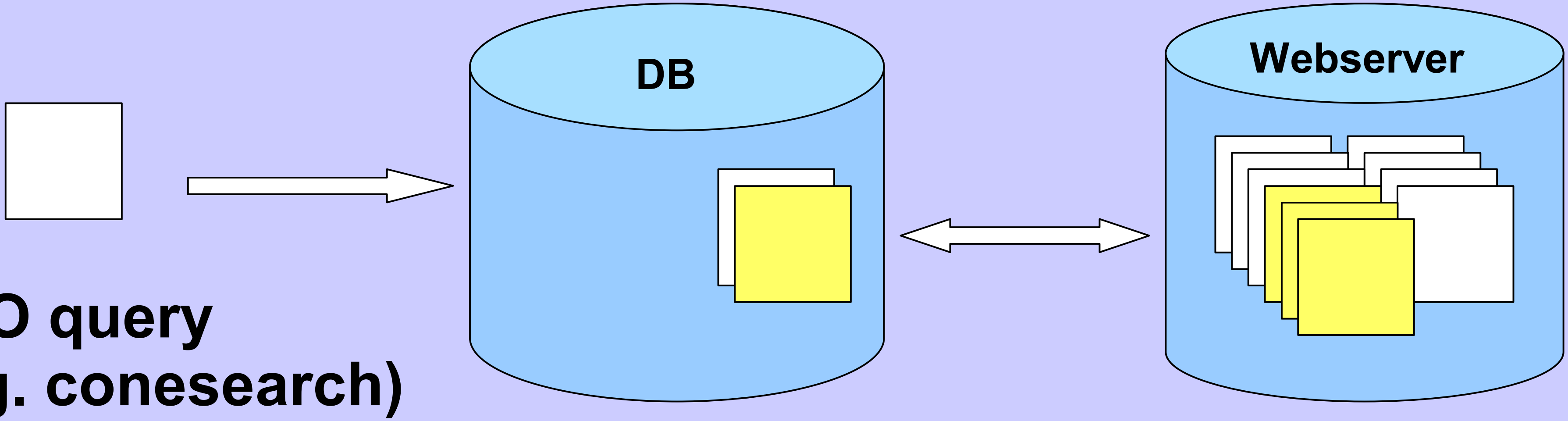
Files returned as Dialect 1

...or Results (IVOA-SIAP!)



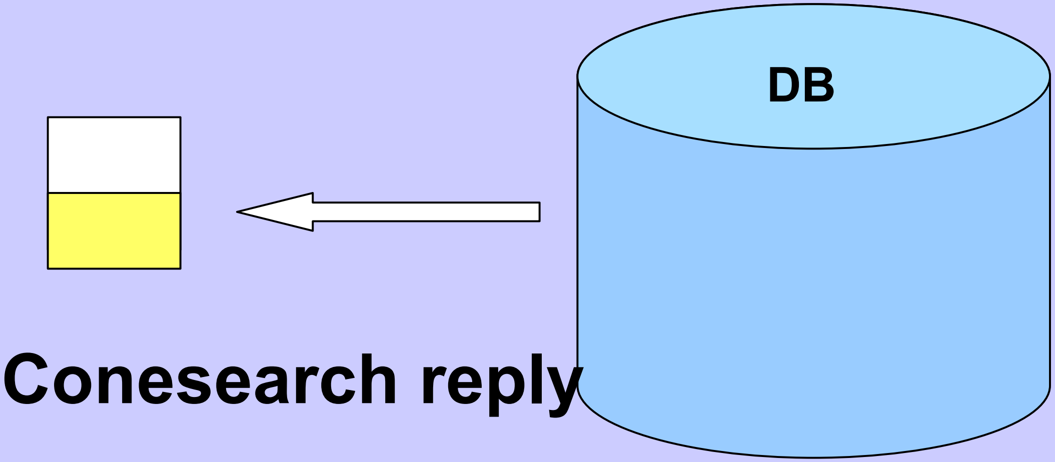
To VO-aware application

Query and reply.....



**(VO query
e.g. conesearch)**

DB searches *both* dialects (because it can translate)



Conesearch reply

Implemented so far

- Database:
    - import, dialect recognition work correctly
    - query translations works
    - some bad problems with evil data!
    - ugly Python interface only

- Webserver:
    - Can translate between different FITS dialects
    - also can convert FITS files to JPG on-the-fly
    - can store .DAT (headerless files) and build FITS on-the-fly.

Current dataset is 67K files in 6 dialects.

In development

- Database:
    - management tools for moving data from one webserver to another

- Webserver:
    - Much better GUI....

- Toolkits:
    - efficient network wrappers for IRAF.

- Future
    - VO interface... (?)

Questions?

Suggestions for a name?

Current candidates:
BCOAT: Blackrock Castle Observatory Archive Toolkit
BBVO: Black Box Virtual Observatory

The archive as processing platform

The obvious: a program can be launched from outside, using the DB search interface (via web, maybe RPC-XML) to find work to do.

The automatic: a script can be attached to a dialect profile in the DB (i.e. a particular data source). Incoming data matching that dialect is imported and stored as normal, then the URL is passed to the script.

The possibly unworkable..... using GUIDs as a form of citation...

Status

Storage:

-

Managing reduced data

Bad answer:

"The person who reduced it has it on their laptop hard disk somewhere. We think. They're at a conference on Robotic Astronomy right now, is it urgent?"

(Not so) bad answer:

"All reduced data goes into the same directory as the raw data, using an (informally?) agreed suffix/change to the name. Sorry, we don't actually record which flats/darks/version-of-pipeline was used."

Good answer (?): "Reduced data is *automatically* uploaded to the archive, referencing all the source frames, plus the version of the pipeline software used (of which the archive also has a copy). Searches on the data show the "

Citing frames

Every frame gets a unique ID

“The person who reduced it has it on their laptop hard disk somewhere. We think. They're at a conference on Robotic Astronomy right now, is it urgent?”

<u>(Not so) bad answer:</u>

“All reduced data goes into the same directory as the raw data, using an (informally?) agreed suffix/change to the

<u>Good answer (?):</u> “Reduced data is *automatically* uploaded to the archive, referencing all the source frames, plus the version of the pipeline software used (of which the archive also has a copy).”

VO:

- a great idea that nobody is implementing!
- public access is expensive to develop & maintain
- cui bono?

Programming skill required is high
Assume you have assembled a catalogue of 100 million stars, all near the equator (so RA,DEC are nearly cylindrical coordinates) angles)

```
"Select * from `stars_table` where (`stars_table`.`ra`-
$query_ra)^2+(`stars_table`.`dec`-
$query_dec)^2<$query_search_radius_squared"
```

(200 million subtracts + 200 million multiplies)
Correct version:

```
list_of_postcodes=SDSS_tessera($query_ra,$query_dec,
$query_search_radius, $precision_limit)
```

Header from hell!

```
SIMPLE  =                    T
BITPIX  =                   16 /8 unsigned int, 16 & 32 int, -32 & -64
real
NAXIS   =                    2 /number of axes
NAXIS1  =                 1024 /fastest changing axis
NAXIS2  =                 1024 /next to fastest changing axis
OBJECT  = '1803+784'
TELESCOP= 'AZT-11 (125cm, 1/13),'
INSTRUME= 'Ap6E    '
OBSERVER= 'Kurtanidze, Nikolashvili and Ivanidze, Petashvili'
NOTES   = 'R filter'
DATE-OBS= '2005-08-24'         /YYYY-MM-DD observation start date, UT
TIME-OBS= '16:50:34'           /HH:MM:SS observation start time, UT
EXPTIME =   300.00000000000000 /Exposure time in seconds
SET-TEMP=  -20.000000000000000 /CCD temperature setpoint in C
CCD-TEMP=  -21.532738095238095 /CCD temperature at start of exposure in C
XPIXSZ  =   24.000000000000000 /Image Pixel Width in microns
YPIXSZ  =   24.000000000000000 /Image Pixel Height in microns
XBINNING=                    1
YBINNING=                    1
XORGSUBF=                    0
YORGSUBF=                    0
IMAGETYP= 'LIGHT   '
BSCALE  =   1.0000000000000000 /physical = BZERO + BSCALE*array_value
BZERO   =   32768.000000000000 /physical = BZERO + BSCALE*array_value
```