VOQL session 2

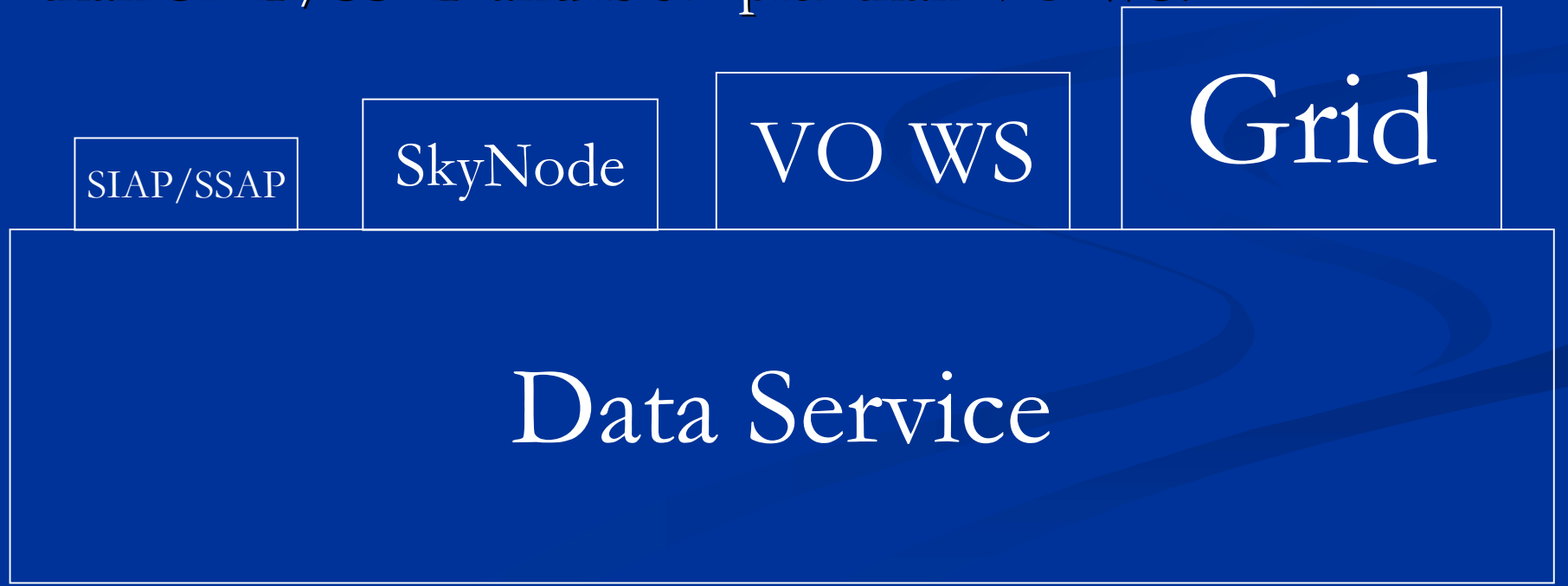
SkyNode

How to live with the VO WS standard

- GWS WG is preparing a common VO IF
  - UWS, CEA, VOStore …
- There will be a more global standard: Grid by GGF
- Will what we are defining in the SkyNode spec will be deprecated ?
  - No. (at least I think so)
  - We are defining SkyNode specific interface, and the interface is simpler and easier to use than the general standard interface.

Hierarchy of Protocol

- Capability of the data service is increased by adapting the higher level protocol
- But complexity is also increased
- Adapt appropriate interface which matches the scale or required visibility of the data service.
- VOQL WG defines the interface that has more capability than SIAP/SSAP and is simpler than VO WS.



Proposal of new interface

- Vodata = performQuery(adqlCore, format)
- Vodata = performQuery(adql, votable, format)
  - There was a xmatch() interface in earlier version but is was hidden by executePlan inerface. It is worthwhile to have this interface independent of executePlan.
- Jobid = performQueryAsync(adql, votable, format, listenerURL)
- Status = performPolling(jobid)
  - "Status" shows whether the query is running or finished. If finished it gives an URL to retrieve the data.
- destroy(jobid)
  - Remove all the resources generated by the job

What should "select into" returns ? empty votable ?

- This query should be used only for performQueryAsync() interface ?

Cross match proposal

- Which algorithm should the xmatch-able skynode support ?
  - Chi2 calculation vs angular distance
- “angular distance” based cross match as a primary algorithm → all the xmatch-able skynode must support this.
- “chi2 calculation” based cross match as an advanced functionality of the xmatch-able skynode.
- Any other algorithms may be supported.
- Supported algorithms (function names) should be exposed by metadata interface

Skynode classification.

- Only the two calssification is not enough : BASIC and FULL
- At least following types will exist:
  - BASIC Skynode
  - FILE UPLOADABLE Skynode
  - Cross match support Skynode
  - ExecutePaln support Skynode
  - Async Skynode

Content of a returned VOTable.

- The order of the FIELD should be the same as the order in the selection list, which enables to access to the data by index id.
- If "*" is specified in the selection list, the order should be decided on the server side.
- All the column metadata should be properly set to the FILED attributes
- "Name" attribute of the FIELD should have a qualified column name. Qualifier should be a table alias name
  - <tableAlias>.<columnName>
- Column metadata that cannot be set to the FIELD attribute may be set by using <VALUES> tag.
  - <VALUES><OPTION name="meta:name" value="value"/></VALUES>
  - This is not the correct usage of <VALUES> tag, but…

Table data model

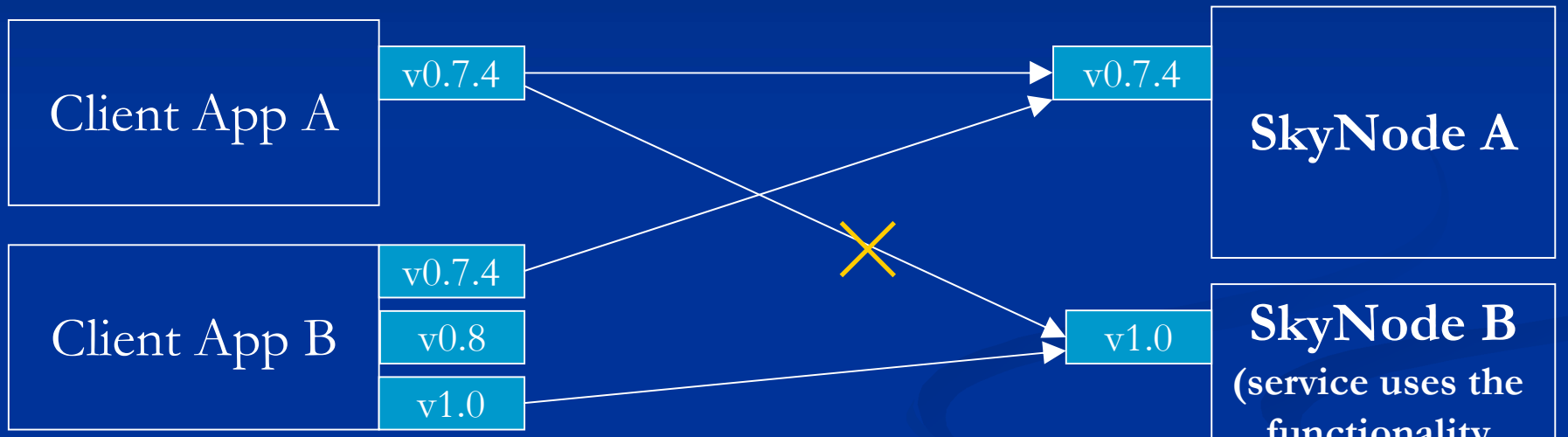
- Define table classes according to the contents of the table
  - General, ObjectCatalog, ObjectBrigthnessCatalog, ObservationCatalog, Image, Spectrum
- For each table class, define columns that must be included. Use utype.
  - General → no requirement
  - ObjectCatalog → utype = id, pos.ra, pos.dec
  - ObjectBrightnessCatalog → id, pos.ra, pos.dec, brightness[i], wavelength_range[i]
  - ObservationCatalog → TBD
  - Image → defined in SIAP
  - Spectrum → defined in SSAP

Metadata: metadata tables vs tables & columns interface.

- Do we need two ways to access to the metadata ?
- Use metadata tables to get more precise information about table and column metadata.
- Use tables and columns interface to get metadata defined as “must provide”.
- Metadata table “tables” and “columns” should have columns that defined as mandatory.
- Metadata table “tables” and “columns” may have columns that is specific to the service.

How to manage the ADQL versions

- Service may be implemented with any public version of ADQL
- Service should expose the supported versions as metadata



- I suggest that the multiple version functionality is implemented on client side, since the number of client app is fewer than the service.