

# Re: DAX Web Services

LSST-DM: Kenny Lo



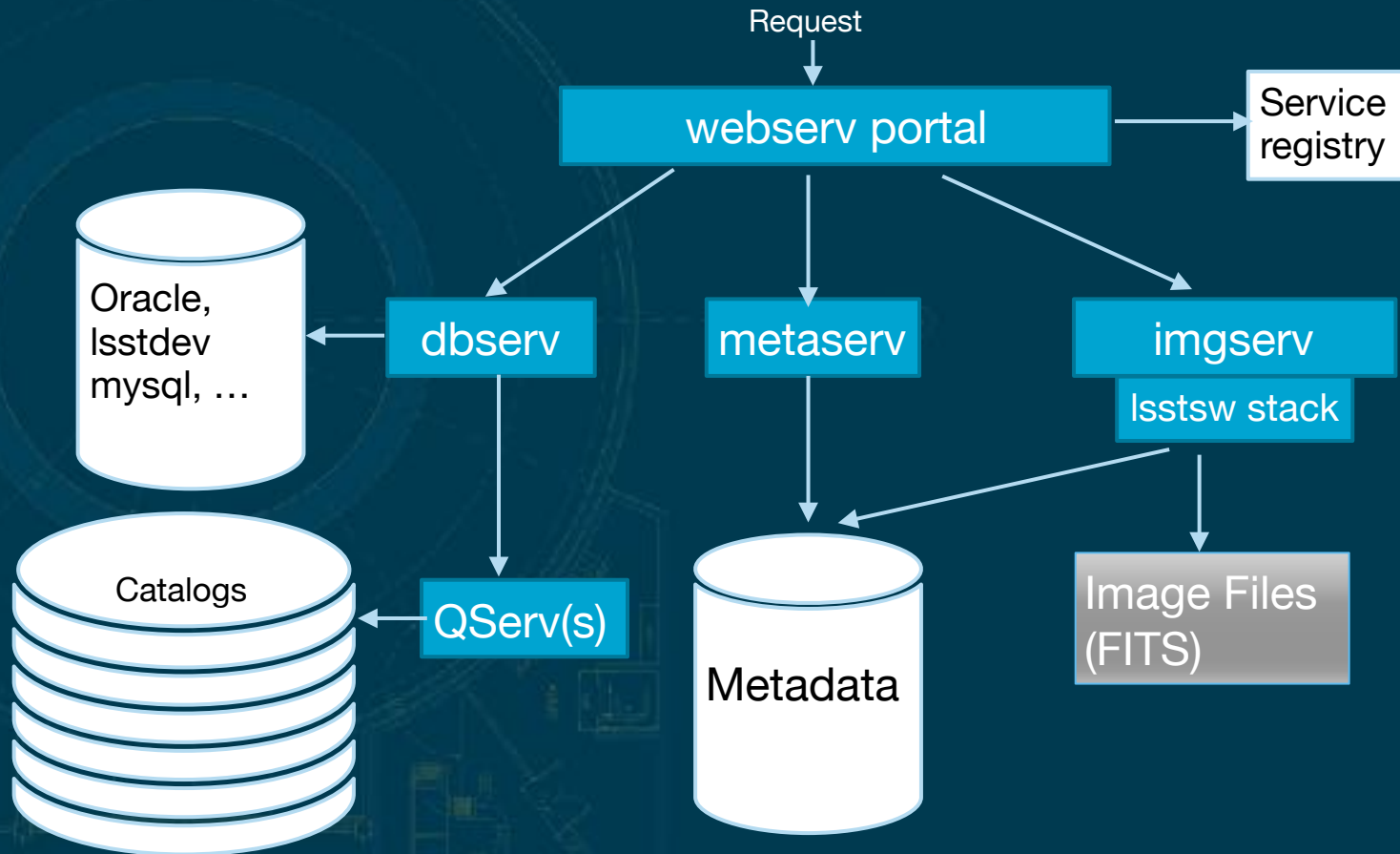
*Large Synoptic Survey Telescope*

# What is DAX Webserv

- DAX -> Data Access
- Collection of Data Access Interfaces for LSST, Implemented as 3 Core Sets of REST APIs
  - `dax_webserv` (webserv) -> Portal for DAX services, registry
  - `dax_dbserve` (dbserve) -> Database Services(TAP)
  - `dax_metaserv` (metaserv) -> Tools for metadata ingestion and APIs (VO Schemas, RegTAP, ObsCore, simple Metadata API serving JSON)
  - `dax_imgserv` (imgserv) -> Image Services (SODA-like), image cutouts/synthesis/generation



# Layout of DAX Web Services



# DAX Data Transfer Today



## Software Development Platforms:

- LSST Software Stack (python, C++)
- SQLAlchemy (python)
- Presto/JDBC (kotlin/java)

## RESTful Query Request:

- GET/POST
- JSON (imgserv)

## Query Response:

- JSON (metaserv)
- FITS (imgserv)



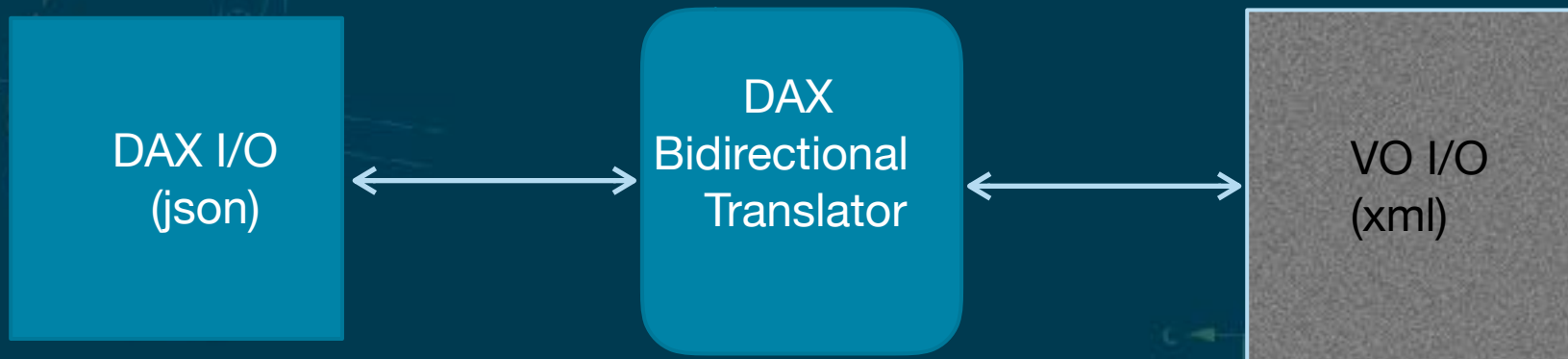
# DAX Webserv Interfaces



## Committed Support for VO Standards:

- metaserv: RegTAP, ObsTAP
- imgserv: SIA v2, DAL/SODA [ImageDM]
- dbserv: TAP/ADQL, SCS (backward-compatibility/public outreach, TBD)

# Approach to Support VO Standards



Data Transport: xml/json/fits over https

Interchange format: VOTable, JSON-Schema

Query Language: Interactive, declarative [ ADQL, SQL ]

Big Data: async [ batch/parallel processing ], cluster computing [ Docker containers orchestrated by Kubernetes ]

# JSON Input -> XML Output



```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "id": "https://github.com/lst/dax_imgserv/blob/master/python/lst/dax_imgserv/config/imageREST_v1.schema",
  "ucd": "https://lsst.org/schemas/vo_ucd.schema",
  "namespace": "lsst.dax.imgserv",
  "ucd": "obs.image",
  "description": "DAX_ImageServ API v1",
  "version": "1",
  "definitions": {
    {
      "orientation": {
        "type": "string",
        "ucd": "pos",
        "enum": [
          "Pixel_Coordinate", "Equatorial_J2000", "Equatorial_B1950", "Ecliptic_J2000",
          "Ecliptic_B1950", "Galactic", "Super_Galactic"
        ]
      },
      "ra": {
        "type": "number",
        "unit": "deg",
        "minimum": 0,
        "maximum": 360,
        "ucd": "pos.eq.ra"
      },
      "dec": {
        "type": "number",
        "unit": "deg",
        "minimum": -90,
        "maximum": 90,
        "ucd": "pos.eq.dec"
      }
    },
    ...
  }
}
```

DAX Schema



```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified"
  xmlns="http://www.ivoa.net/xml/VOTable/v1.3"
  targetNamespace="http://www.ivoa.net/xml/VOTable/v1.3"
>
<xs:annotation><xs:documentation>
  VOTable is meant to serialize tabular documents in the
  context of Virtual Observatory applications. This schema
  corresponds to the VOTable document available from
  http://www.ivoa.net/Documents/latest/VOT.html
</xs:documentation></xs:annotation>

<!-- Here we define some interesting new datatypes:
- anyTEXT may have embedded XHTML (conforming HTML)
- astroYear is an epoch in Besselian or Julian year, e.g. J2000
- arrayDEF specifies an array size e.g. 12x23x*
- dataType defines the acceptable datatypes
- ucdType defines the acceptable UCDs (UCD1+)
- precType defines the acceptable precisions
- yesno defines just the 2 alternatives
-->

<xs:complexType name="anyTEXT" mixed="true">
  <xs:sequence>
    <xs:any minOccurs="0" maxOccurs="unbounded" processContents="skip"/>
  </xs:sequence>
</xs:complexType>

...
</xs:schema>
```

VO Schema

# DAX Intermediate(Internal) Representation



```
{
  "$comment": [
    "Version: 1.0",
    "Source: https://lsst.org/schemas/imgserv_api_v1.schema",
    "HashOp: hashlib.sha256(str(sorted(api_id)).encode('utf-8')).hexdigest()"
  ],
  "2ca69ed8aa69c9f01469617000a6a30d5df02d3843677330c6b457eef0e5b0d9": {
    "api": "Image.full_nearest",
    "api_id": ["db", "ds", "filter", "center.x", "center.y", "center.unit"]
  },
  "1697b5040cb4fb249ab5152412cc95a3b7f5a9d74dd5ed5666cacb1e420fb539": {
    "api": "Image.full_from_data_id",
    "api_id": ["db", "ds", "run", "camcol", "field", "filter"]
  },
  "fd2dd2f42cbf6670533e20932165e4dbe7d9718c9ebcca8a48dc7afff21234d4": {
    "api": "Image.full_from_data_id",
    "api_id": ["db", "ds", "tract", "patch_x", "patch_y", "filter"]
  },
  "124ad4bde6e752bbc6e9d688b2c775adaaa59df6e1e53230b6302f165ccd7629": {
    "api": "Image.full_from_data_id",
    "api_id": ["db", "ds", "tract", "patch", "filter"]
  },
  ...
}
```



DAX Interfaces  
(json)



VO Interfaces  
(xml)



## Four-Tier Testing Approach:

1. Unit testing, at module level
2. Requests validated through JSON Schema
3. Suite of Integration tests with results (.fits) checked via SHA checksums
4. External service for VO validation, as in:
  - <http://wiki.ivoa.net/twiki/bin/view/IVOA/IvoaValidatorsSummary>

# DAX is Open Source



LSST on GitHub: <https://github.com/lsst>

## DAX GitHub Projects:

- `lsst/dax_webserv` [python]
- `lsst/dax_dbserve` [dbserve\_v0, python] -> deprecated
- `lsst/albuquery` [dbserve\_v1, kotlin/java]
- `lsst/dax_metaserv` [v1, python]
- `lsst/dax_imgserv` [v1, python]

# Closing Remarks

Q & A  
Thank You!

Contact: [kennylo@slac.stanford.edu](mailto:kennylo@slac.stanford.edu)

