

Re: DAX Web Services

LSST-DM: Kenny Lo



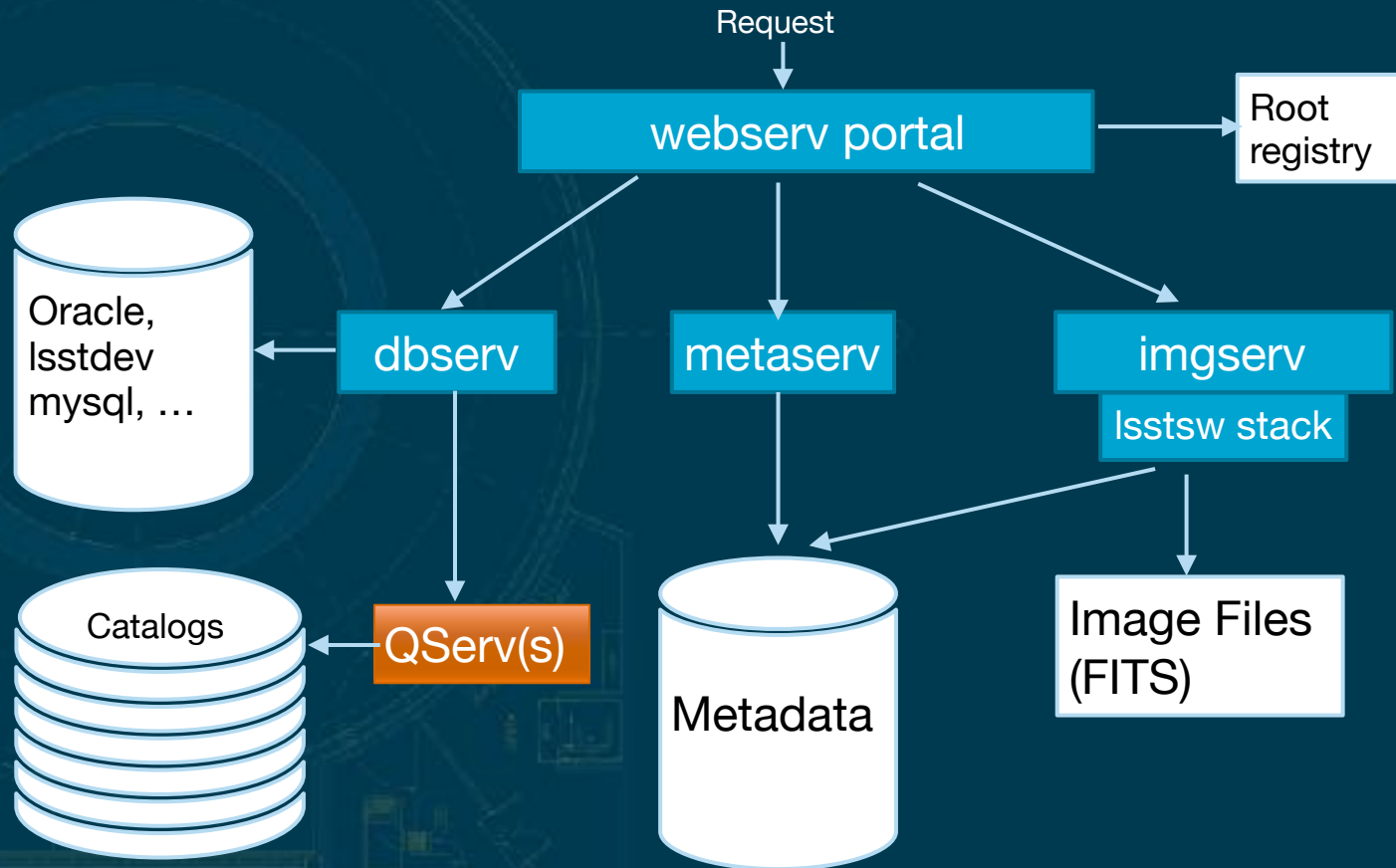
Large Synoptic Survey Telescope

What is DAX Webserv

- DAX -> Data Access for LSST, dax_webserv -> Portal for all DAX services, w/ root service registry
- Test Datasets: SDSS_Strip82, WISE_All_Sky, HSC (Subaru Telescope)
- Collection of Data Access Interfaces, implemented as 3 Core Sets of REST APIs
 1. dax_dbserve (dbserve) -> Database Services(TAP)
 2. dax_metaserv (metaserv) -> Tools for metadata ingestion and APIs (VO Schemas, RegTAP, ObsCore, simple Metadata API serving JSON)
 3. dax_imgserv (imgserv) -> Image Services (SODA-like), image cutouts/synthesis/generation



Layout of DAX Web Services



DAX Data Transfer Today



Software Development Stacks:

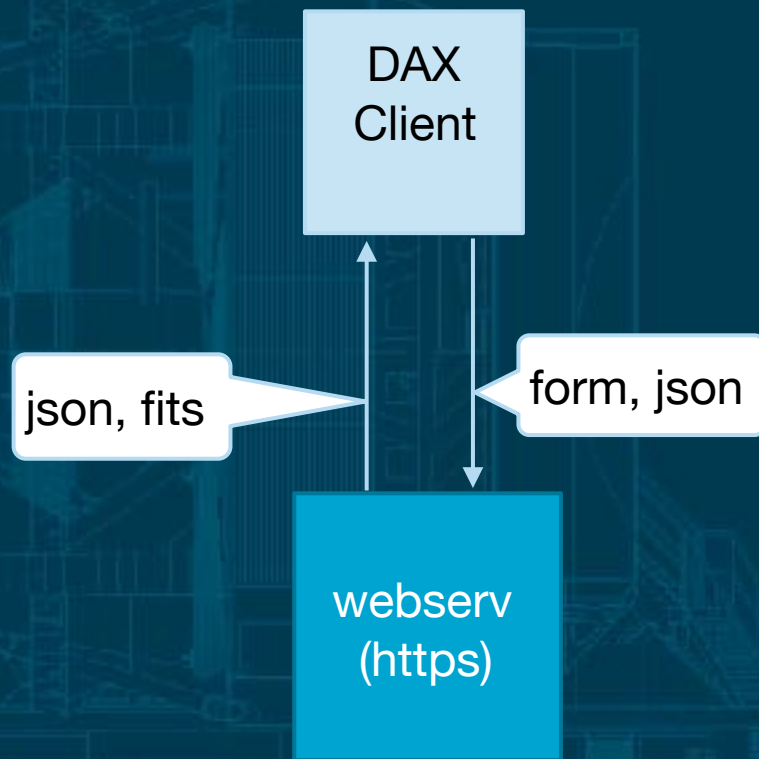
- LSST Software Stack (Python, C++)
- SQLAlchemy (Python)
- Presto/JDBC (Java)
- Flask (Java)
- DropWizard (Kotlin)

RESTful Query Request:

- GET/POST sync/async
- JSON (imgserv)

Query Response:

- JSON (metaserv)
- FITS (imgserv)



DAX Roadmap for VO Interfaces



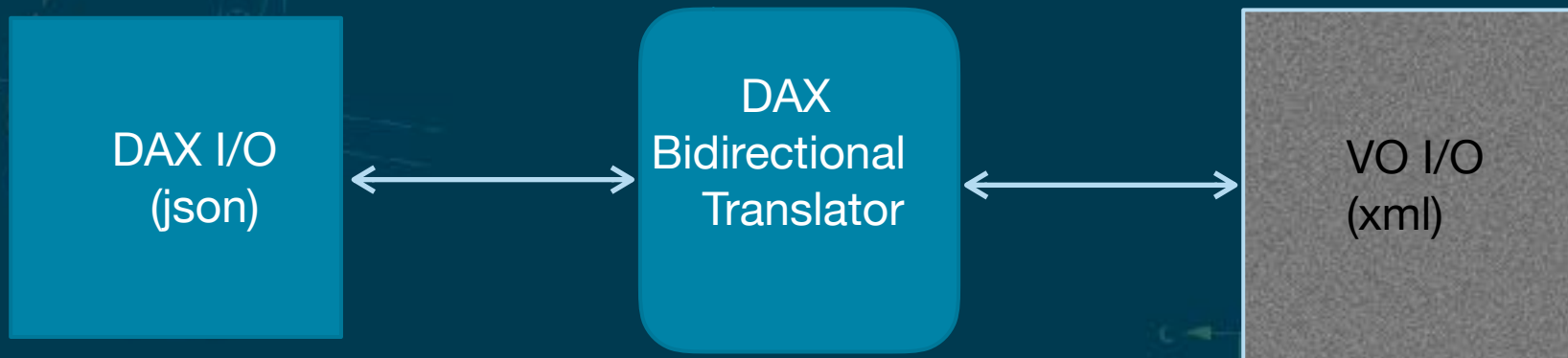
Pragmatic Approach: Embrace, Adapt, Extend

Status: coming out of v0 or pilot implementations, now in v1

In v1, committed support for essential VO standards:

- metaserv: RegTAP, ObsTAP
- imgserv: SIAv2, 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]

Use Case: 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 Internal API 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 Automated Testing Plan:

1. Unit testing, at module level
2. Requests validated through JSON Schema
3. Suite of Integration tests with results (.fits) checked via checksums (SHA256)
4. External service for VO compliance validation:
 - <http://wiki.ivoa.net/twiki/bin/view/IVOA/IvoaValidatorsSummary>

DAX is Open Source



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

GitHub DAX 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]

Thank You

Acknowledging DAX Contributors at
LSST-DM:

- Brian Van Klaveren
- John Gates
- Kian-Kat Lim, aka KT
- Jacek Becla (Emeritus)

