Software Architecture and System Design of Rubin Observatory

William O'Mullane, Tim Jenness, KT Lim, Frossie Economou

Vera C. Rubin Observatory

31st October 2022 (I08)

as imagined and in June 2022 - Ops 2024

Photo: William O'Mullane
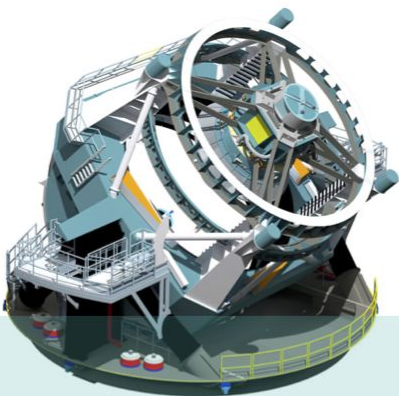
El Peñon, Cerro Pachón, Chile   Altitude: 2647m   Nice 1 min Video

System Vision

VERA C. RUBIN OBSERVATORY

Raw Data: 20TB/night

Sequential 30s images covering the entire visible sky every few days

LSST SCIENCE PIPELINES

Prompt Data Products

Alerts: up to 10 million per night

60s via nightly alert streams

Community Brokers

Raw & Processed Visit Images, Difference Images, Templates

Transient and variable sources from Difference Image Analysis

Solar System Objects: ~ 6 million

24h via Prompt Products DB

Rubin Data Access Centres (DACs)

USA (USDF)
Chile (CLDF)
France (FRDF)
United Kingdom (UKDF)

Data Release Data Products

Final 10yr Data Release:
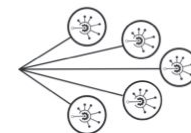- Images: 5.5 million x 3.2 Gpixels
- Catalog: 15PB, 37 billion objects

via Data Releases

Independent Data Access Centers (IDACs)
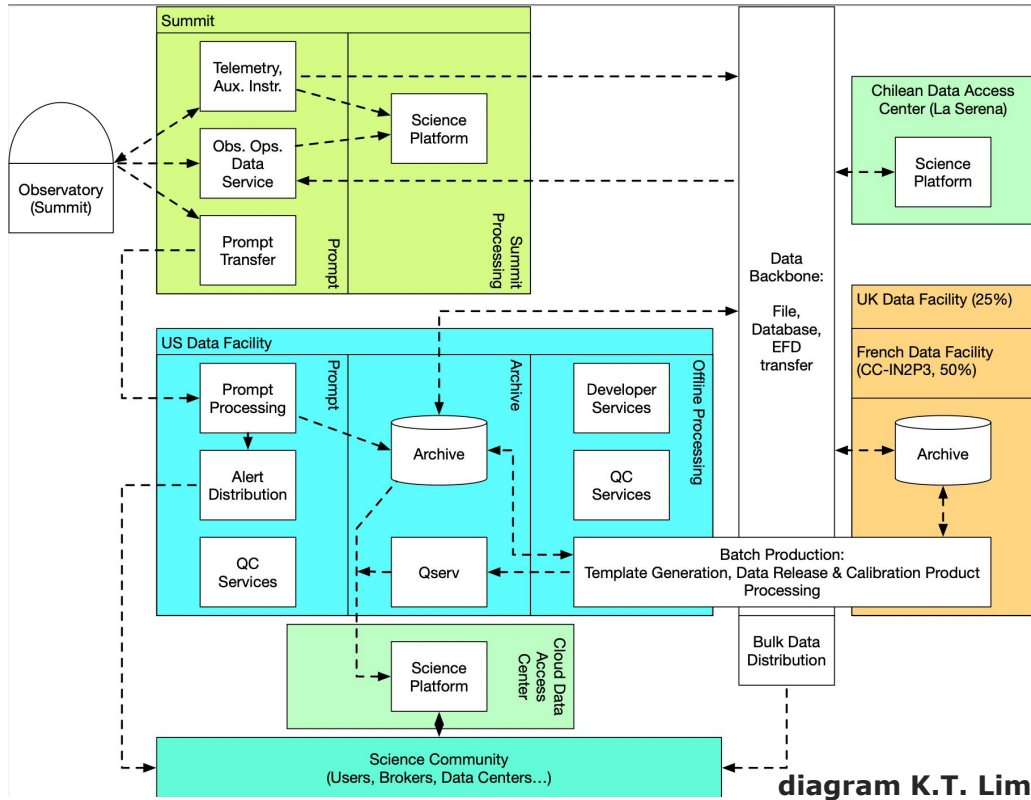
Access to proprietary data and the Science Platform require Rubin data rights

Rubin Science Platform

Provides access to LSST Data Products and services for all science users and project staff.

RUBIN SCIENCE PLATFORM

PORTAL | NOTEBOOKS | WEB APIS

DATA RELEASES | ALERT FILTERING SERVICE | USER DATABASES | USER FILES | USER COMPUTING | SOFTWARE TOOLS

Credit: Leanne Guy
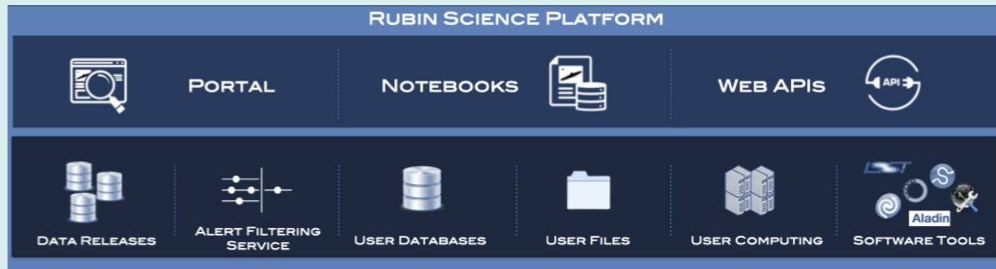
Rubin DM build and deploy

diagram K.T. Lim

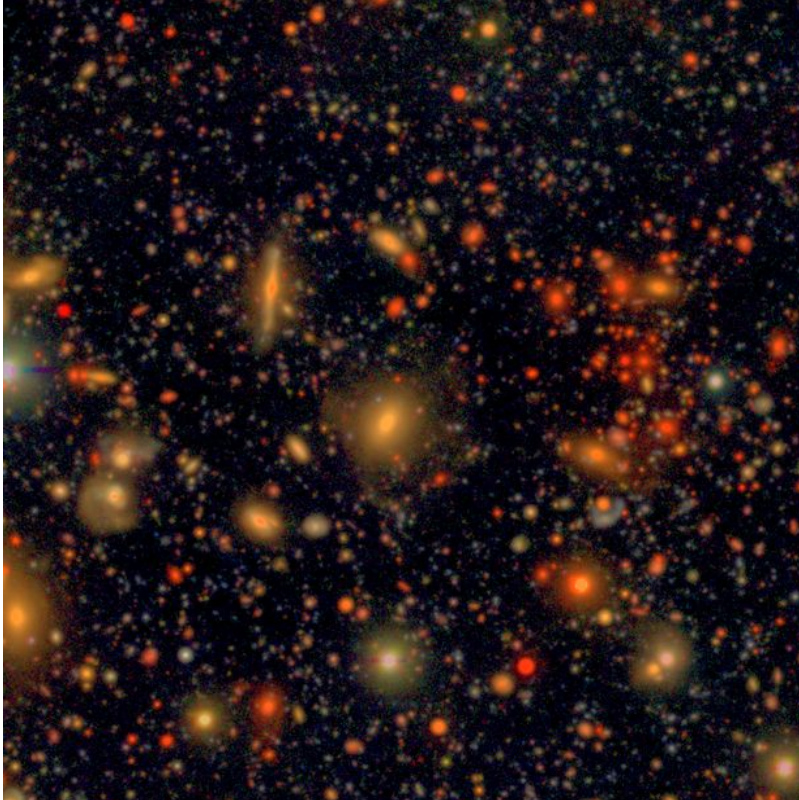DM must build everything to get Rubin products to users.

- large data sets (20TB/night)
- complex analysis
- aiming for small systematics
- Science Alerts in under 2 minutes .. (aiming for 1 minute)

All code on github:
https://github.com/lsst

Wide, Fast, Deep… Difficult?

The COSMOS field seen by Hyper Suprime-Cam, courtesy of the HSC Collaboration, R. Lupton, and N. Lust.

- Rubin's LSST is not the first wide-field imaging survey…

- …but the combination of depth, area, and throughput make it uniquely challenging e.g.

  - Everything is blended.

  - Many measurements are systematics limited.

- Testing on precursors like HSC!

Alerts, requirement 120s goal ~60s

Custom job dispatch to nodes with preloaded data (may be PanDA)

2 "Strings" of 189 cores - 1 per ccd.

MOPS = Moving Object Pipeline System
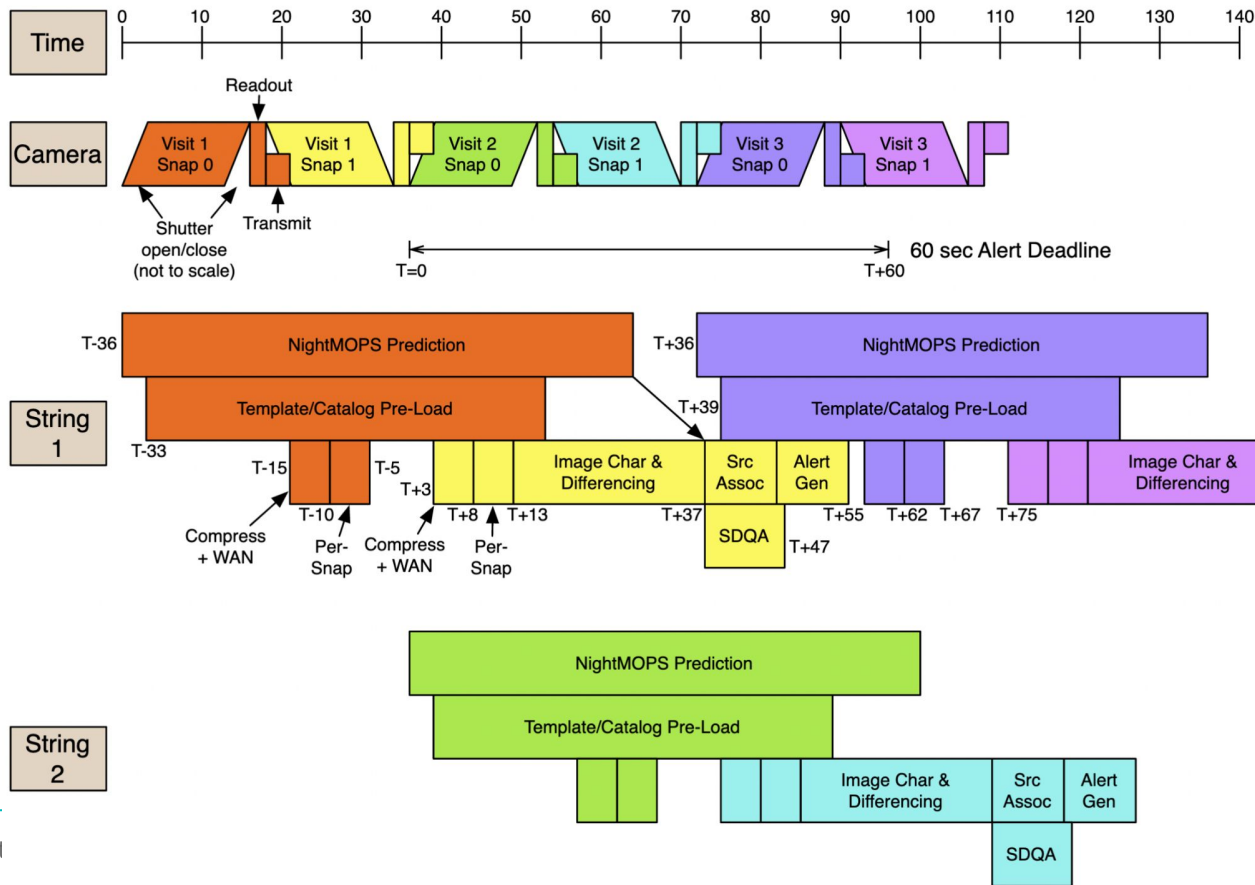
Data Release Processing (6-9 months)
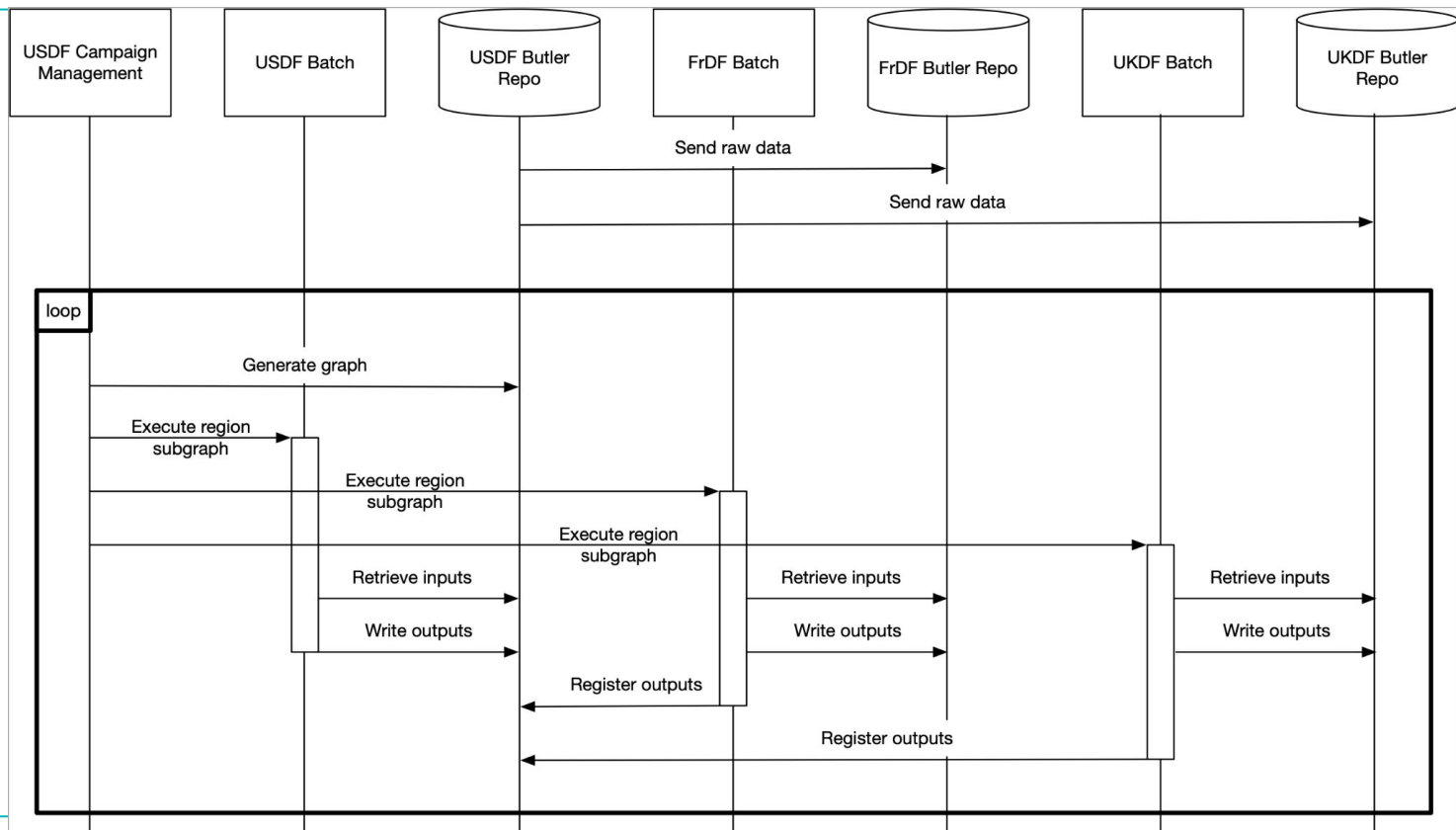
3 sites

US FR UK

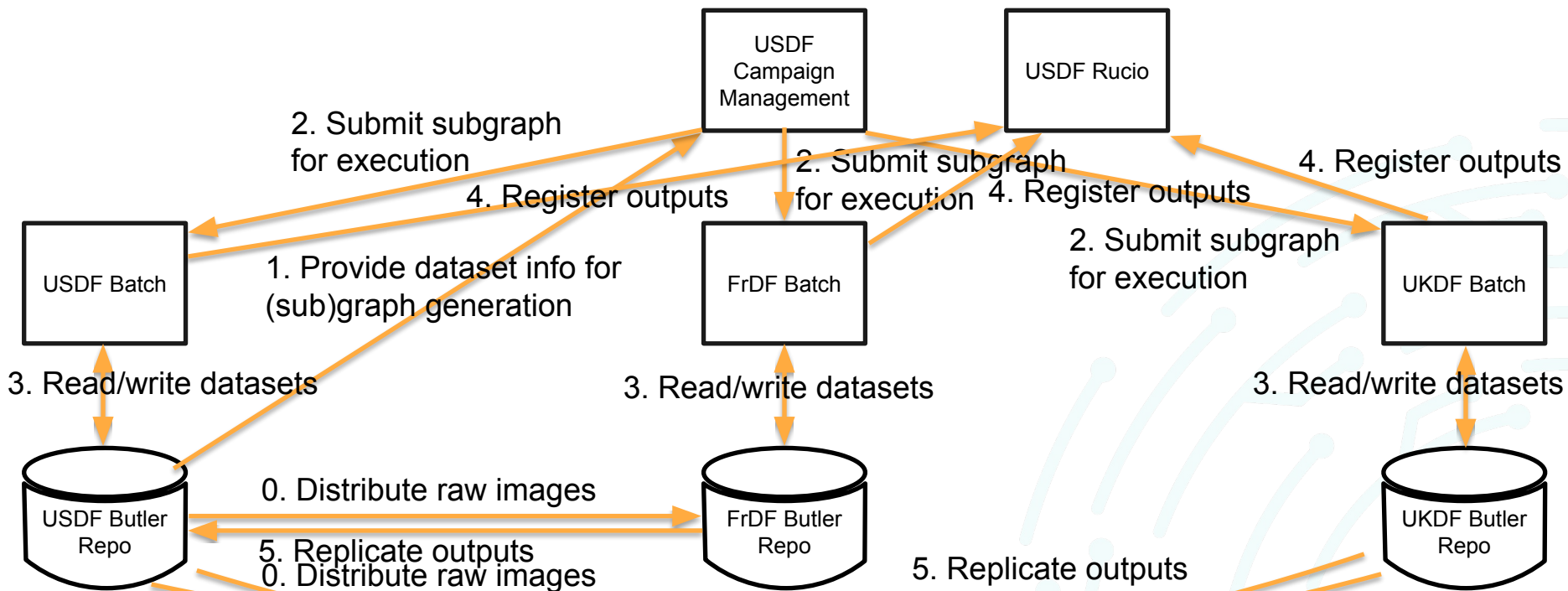Batch Production System (BPS) in front. **(see Poster Michelle Gower**)

PanDA managing full processing Graph, Slurm or other locally.

Some data location smarts to be built in.

Three-Facility DRP Dataflow

Rubin Science Pipelines

- Massive open source software project
  - ~200 (70FTE) on project + contributors
  - github.com/lsst/ Docs pipelines.lsst.io
- (lsst_distrib) for all processing algos
  - Third-party dependencies conda-forge or pypi.org if not on conda;
    - internal forks only when necessary.
    - Bug fixes etc. upstreamed .
  - Much conversation/workshops/notes shared with community on "best" algorithm e.g.:
    - Crowded Field Photometry (dmtn-129)
    - Deblending (dmtn-038)
  - Intention to allow experimentation
    - Alternative algorithms and configs

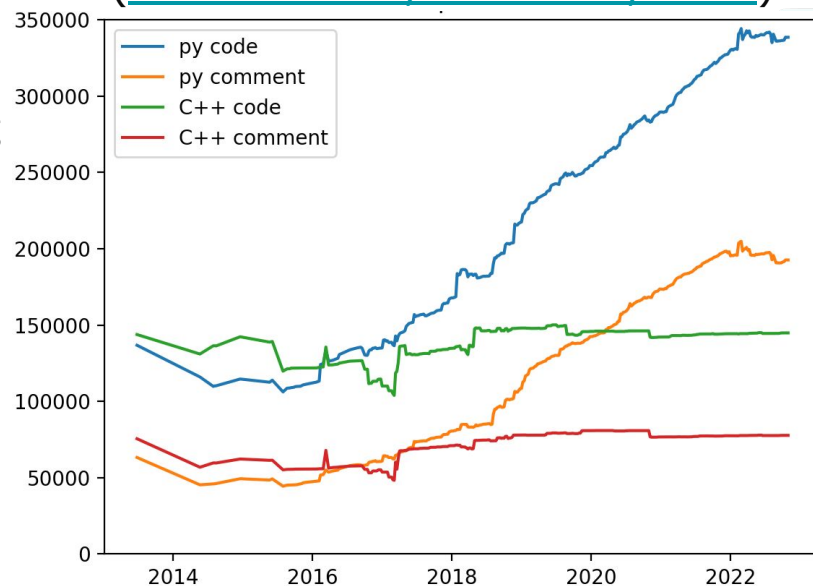Over 10^6 lines of code & comments C++/Python/Java/JavaScript (Jenness et al., Proc SPIE, 2018).

Figure courtesy Tim Jenness.
Based on the Science Pipelines codebase only.

Democratizing research in astronomy

- Open source software is key to open science: not just for traditional reproducibility arguments, but because it's not enough to be OPEN in order to be INCLUSIVE
- Supporting the democratization of science includes finding ways to support researchers who are:
  - Resource-poor (do not have the compute resources associated with major research universities)
  - Time-poor (have a high teaching load, few/no grad students or postdocs)
  - Work in liberal arts colleges, historically black colleges, or other places that lack a large peer network for technical and research support
- Lowering the barrier to entry goes beyond data rights and even beyond software; it requires minimizing the investment (time, money, experience) necessary to meaningfully engage with the scientific questions that can be resolved with the data

Data Access

Data rights holders will have access to the catalogs and images via the Science platform.

- We are currently considering having parquet files along side Qserv
  - We think we need "unpredictable" and "complex" access which Qserv may or may not handle
    - user-defined functions, pattern matching, unusual iteration schemes.
  - Many cloud tools work best with cloud formats like parquet
- A subset of data will be public via Education and Public outreach
  - Including for citizen science projects
- Alert stream will be public

We will hold most data at SLAC but run the science platform on Google (Hybrid model)

- Cloud vs hybrid vs on-prem: cloud compute is cheap, storage is expensive, services evolve faster, egress can be managed
- Rubin Science Platform
  - Vision: LSE-319, Design: LDM-542, Test: DMTR-51 - on Google (O'Mullane 2022)

Cloud Native

- Early adoption of Kubernetes drove service architectures that are well isolated from the underlying infrastructure
- This approach has already paid off massive dividends:
  - When funding lines suddenly shifted we were able to painlessly transition from an on-premises facility to an Interim Data Facility on Google Cloud
  - The Rubin Science Platform (RSP) became a generic data services platform that is currently deployed on 8 distinct (and distinctly managed) infrastructures (on-prem and cloud)
  - Cloud can now be freely leveraged for services like the RSP who benefit from its advantages such as elasticity, scalability, isolation
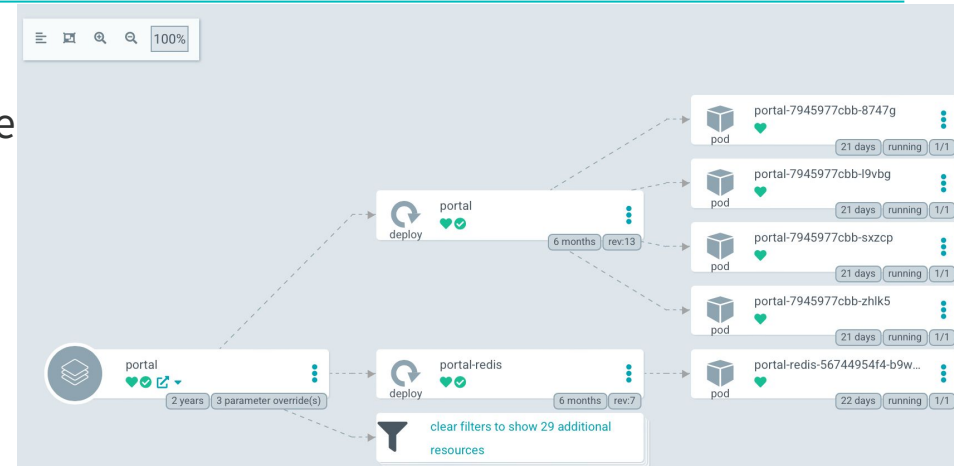
A live view of the RSP portal's deployment on the Google Cloud-based production RSP instance (data.lsst.cloud). 4 replicas of the firefly servers backing the service have been deployed in order to handle user load. This instance is currently open to ~600 early access users using simulated data provided by the DESC project in advance of the start of our own survey

GitOps k8s clusters for services with [Phalanx](#)

- Our architectural approach is geared towards lowering the cost for developing and deploying a new data service
- Services utilize a common infrastructure providing services such as authentication and authorization, secrets management, TLS certificates, and templates to speed up creation of new services in the FastAPI framework
- GitOps infrastructure for k8s deployment using ArgoCD takes care of easy per-infrastructure configuration and deployment

```
1   environment: summit
2   fqdn: summit-lsp.lsst.codes
3   vault_path_prefix: secret/k8s_operator/summit-lsp.lsst.codes
4
5   alert_stream_broker:
6     enabled: false
7   cachemachine:
8     enabled: true
9   cert_manager:
10     enabled: true
11  datalinker:
12     enabled: false
13  exposurelog:
14     enabled: true
15  gafaelfawr:
16     enabled: true
```

Extract from the configuration file defining the phalanx deployment on the Cerro Pachon summit k8s cluster. Individual services are enabled if they are needed on that deployment (such as the observation exposure log) or form part of the overall shared service infrastructure (such as the the TLS certificate manager)

Standards - use them

- Standards are good - they minimize learning curve (which is a major problem)
  - ECSS (O'Mullane 2008) for Gaia
  - MBSE (Selvy 2018)for Rubin
- For Astronomy we also have IVOA standards
  - Gaia Archive is all IVOA based (Salgado 2019, Gonzalez-Núñez 2015)
  - SDSS also did a lot with VO and IVOA (Thakar 2004)
  - Rubin IVOA first:
    - TAP, HiPS,
    - Image services (implements SODA)
    - DataLink (abstracts image access from ObsTAP results)
    - Microservices bouncing TAP results into additional queries via DataLink descriptors
- Even better use others implementations of them
  - Rubin uses CADC TAP with Qserv plugin
  - That allows us to use FireFly for visualization fairly easily as it is VO based.

So you defined your architecture

- You did a lot of analysis - you set out a design
- Perhaps using Unified Modeling Language, Model Based System engineering
  - Rubin used MBSE with MagicDraw to hold requirements and design.
    - Requirements it seems could always be clearer ..
  - Verification is with Jira Test Manager (Zephyr)
  - Mostly cloud native - see next slide
- Perhaps you did a functional breakdown or an OO breakdown
- Not sure it really matters eventually you end up with some way to attack the work
  - You will have a set of systems and subsystems and a vague idea how they will be deployed
    - 7 +/- 2   (Rubin DM - & subsystems, Gaia 9 Coordination units)
  - You may have some client server, some tiered parts, some pipes and filters
  - You used Model View controller and other patterns
  - If the project is big enough you have a bit of almost everything ..
- System Engineering does not necessarily scale linearly (network effect on ICDs etc.)
  - Assumptions made early come back and cost you a lot later
  - Write it all down as precisely as possible
  - Put effort in early on automated testing and test-> requirement tracking

Documentation, writing and finding

- Have a good document publishing/FINDING system
  - Provide templates for standard documents early on
    - Most Gaia docs used Latex templates provided and were in SVN - Livelink held PDFs (~seacharble)
  - Rubin developed a documentation infrastructure that further lowers the barrier to documentation by providing templated creation via Slack and uses the same IDE/toolchain developers use for coding, supports Restructured Text/Latex and is published via Github through the well-known GitOps workflow
    - Single page documents (technotes) and site-based documentation (eg pipelines.lsst.io) share the same infrastructure (see sqr-000.lsst.io) and search indexing hub (lsst.io)
    - This has also minimized useful information being consigned in hard to find (eg Google docs) and easy to forget or edit in one's preferred IDE (eg wikis)
    - Docushare holds change controlled docs (PDFs Word) has a search function..
  - Both have bibfile generation for all recorded docs : Gaia livelink - complete. Rubin (lsst.io) incomplete
- Glossaries are good, both Rubin and Gaia also have tools to generate acronym lists from documents (text, or tex - ~~Word~~)
  - http://gaia.esac.esa.int/gpdb/glossary.txt
  - https://www.lsst.org/scientists/glossary-acronyms

Not all parts need to start on day 1

- Frequently large project start all subsystems together
- A slower ramp up is better - think when you need things .. (yes hard to keep politics out of that)
- For Gaia :
  - Coordination Unit 1 (CU1 Architecture) and CU3 (Astrometry) were concentrated on initially with others trailing by some months - people from other CUs were in CU1.
  - CU9 (Gaia Archive) was purposefully delayed to many years after the other parts of the project were almost built and launch was close
- On Rubin all DM WBS elements started together
  - some teams ready but others did not need their components yet.
  - Good involvement in developer guide etc. developer.lsst.io,

Interfaces - especially to data

A core tenet : Separate the data model from the persistence mechanism

- Algorithms should not access data directly
  - Butler is great in Rubin, now used on SPHEREx also (Jenness 2022, Jenness 2018, see also talk by Nate Lust later)
  - Felis holds the model, Butler passes Python Objects to clients with algorithm code not knowing about data location or file formats.
- Gaia had data trains and the MDB dictionary
  - Insulation of algorithms from data access since the outset (O'Mullane 1999)
  - Dictionary holds all data models (Hernandez 2015, O'Mullane 2011)

Then there are the interfaces between systems and to others

- Interface Requirements Documents and Interface Control Documents
  - Needs system engineering and testing ..
  - Rubin ICDs are really only IRDs .. but we may survive …
  - Gaia : we placed too much faith in data ICDs, it was insufficient

Products and repos and stacks - Anti pattern

- Product tree identifies software and who is responsible - Good !(Rubin product tree→)
  - Should group packages and help dependencies
- Rubin
  - We have 100s of repos on GitHub
  - The dependencies are not straight forward - Pretty much need to build complete from source
  - So we have a Mono Build - suited to a mono svn repo
  - But we have a package based set of GitHub repos
  - Clearly the latter is the correct pattern getting there is hard
  - **Middleware was extracted and put on PyPi.**
- Gaia
  - Huge SVN repo of everything based roughly on product tree - builds on parts of the SVN tree - dependencies strictly at jar level via nexus
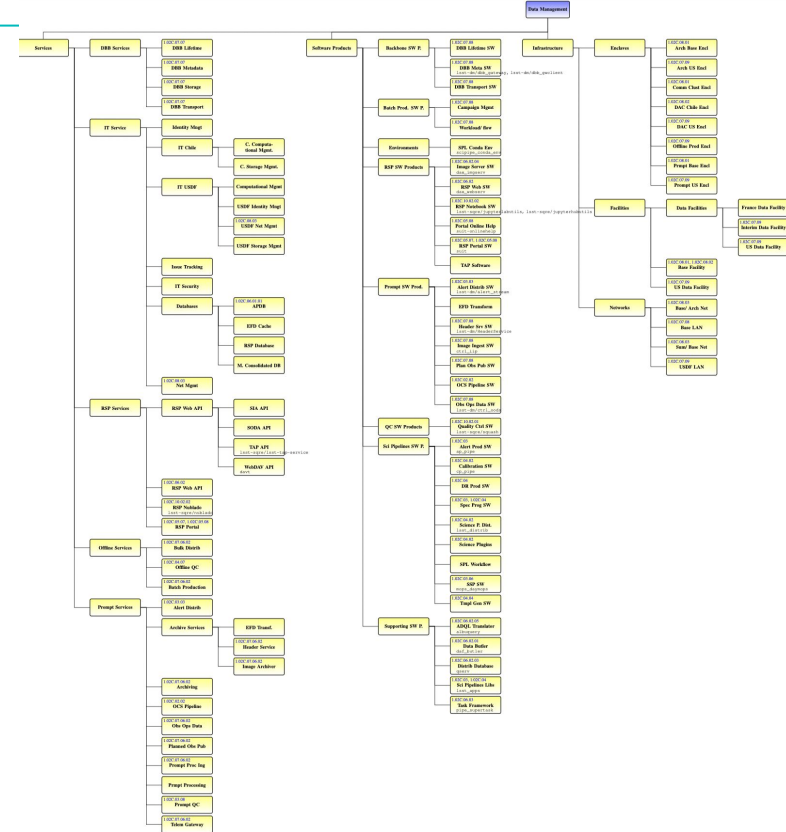
Deployment- Abstract it, Automate it !

Abstracting infrastructure effectively (Kubernetes / container orchestration, middleware etc) to facilitate wider adoption of software and services by others, reducing context switching penalty and supporting continuing expertise

Rubin

- Foreman/Puppet for bare metal setup security injection etc. up to kubernetes
    - Some machines on the summit are not under kubernetes .
    - SLAC use Chef
- Docker Hub for containers
- Kubernetes for orchestration also on the summit, ArgoCD to deploy

Gaia

- Java for portability - no containers but always deployed Jars (from Nexus)
- All configurations in SVN - deployment script pulled correct versions to a specific machine

Databases, you will have, love em, hate em

Databases are great

- persistence, ability to query in different ways, ACID (for RDBMS)

Databases are terrible:

- centralization/replication, schema evolution, performance cliffs, difficulties with multi-user/multi-tenant are not so good; REST APIs in front help a lot;

More are better:

- per-application databases, sometimes specialized (Redis, InfluxDB) add resilience and are more manageable nowadays
- Rubin: Influx for summit Engineering, Postgres for observing logs and ancillary info, AlertsDB. Cassandra for Prompt Products
    - Of course in house developed Qserv for catalog access - **Talk by Fritz Mueller**
- Gaia: Intersystems Cache for processing, Postgres for archive and dictionary ..

(Open) Software Project management lessons

- Leadership is required not just management
  - That requires domain expertise and management training
  - Must acknowledge this route is not for everyone - experimenting is good
- You can do agile but you probably want earned value (eXtreme ECSS, tracking, EV)
  - Have good managers who understand both technical and managerial needs (super hard)
  - Built a techno/scientific culture in leadership -
- Offering opportunities for getting career credit for supporting the mission and its community, not just first-to-publish
- Put open source in the contracts - most organisations have no clue
  - Licensing is important - we picked GPL but would now prefer APL - hard to change
  - Goal is to create community activity and collaboration around a codebase

Bunch of management takeaways in "Large Scientific Data Systems: analysis of some existing projects and their applicability to Gaia"

The End

Questions?

In the end it is all about teams !