Software Architecture and System Design of Rubin Observatory

William O'Mullane, Tim Jenness, KT Lim, Frossie Economou
Vera C. Rubin Observatory

31st October 2022

Software Architecture and System Design of Rubin Observatory

In this presentation we will cover some astronomy design patterns and perhaps some anti-patterns in astronomy. We will use our experience on several long projects such as Rubin Observatory, Gaia, SDSS, UKIRT and JCMT to highlight some of the the things which worked and a few things that did not work so well. For example separation of data access from data format is common across all of our projects and something we find critical. The Rubin Science Platform (and deployment system) underpinned by infrastructure as code is also a culmination of many previous efforts.

https://pretalx.com/adass2022/me/submissions/8CYMWV/

as imagined and in June 2022 - Ops 2024

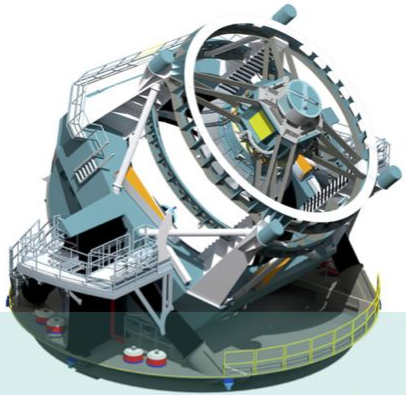
Photo: William O'Mullane

El Peñon, Cerro Pachón, Chile Altitude: 2647m Nice 1 min Video

System Vision

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

Raw & Processed Visit Images, Difference Images, Templates

Transient and variable sources from Difference Image Analysis

Solar System Objects: ~ 6 million

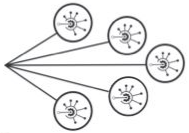
24h via Prompt Products DB

Data Release Data Products

Final 10yr Data Release:

- Images: 5.5 million x 3.2 Gpixels
- Catalog: 15PB, 37 billion objects

via Data Releases



Community Brokers

Rubin Data Access Centres (DACs)

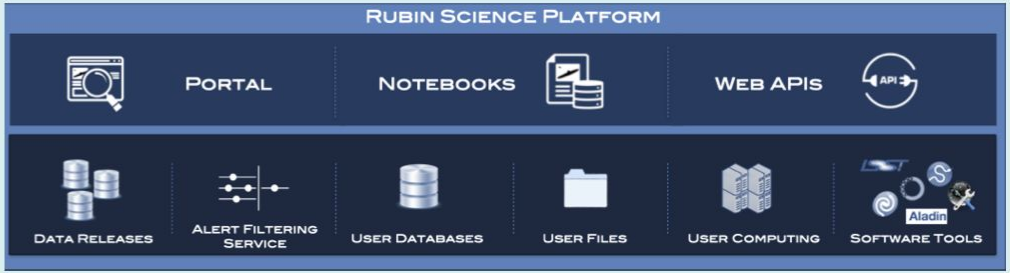
USA (USDF)
Chile (CLDF)
France (FRDF)
United Kingdom (UKDF)

Independent Data Access Centers (IDACs)

Access to proprietary data and the Science Platform require Rubin data rights

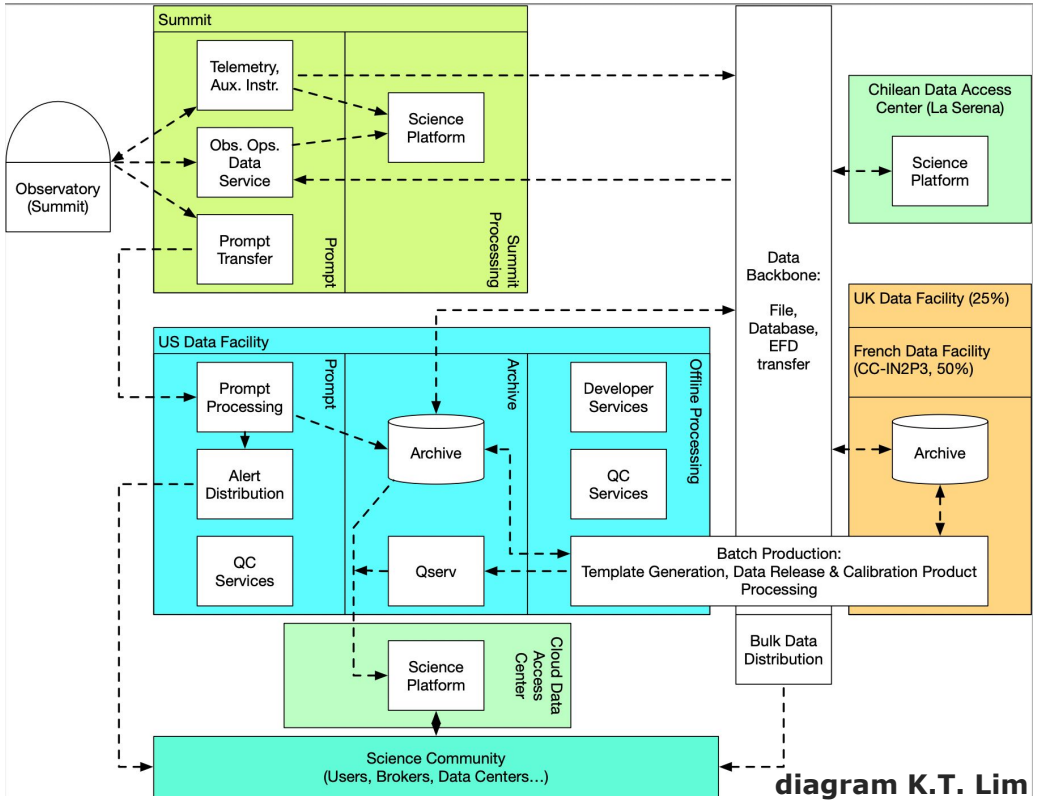
Rubin Science Platform

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



Credit: Leanne Guy

Rubin DM build and deploy

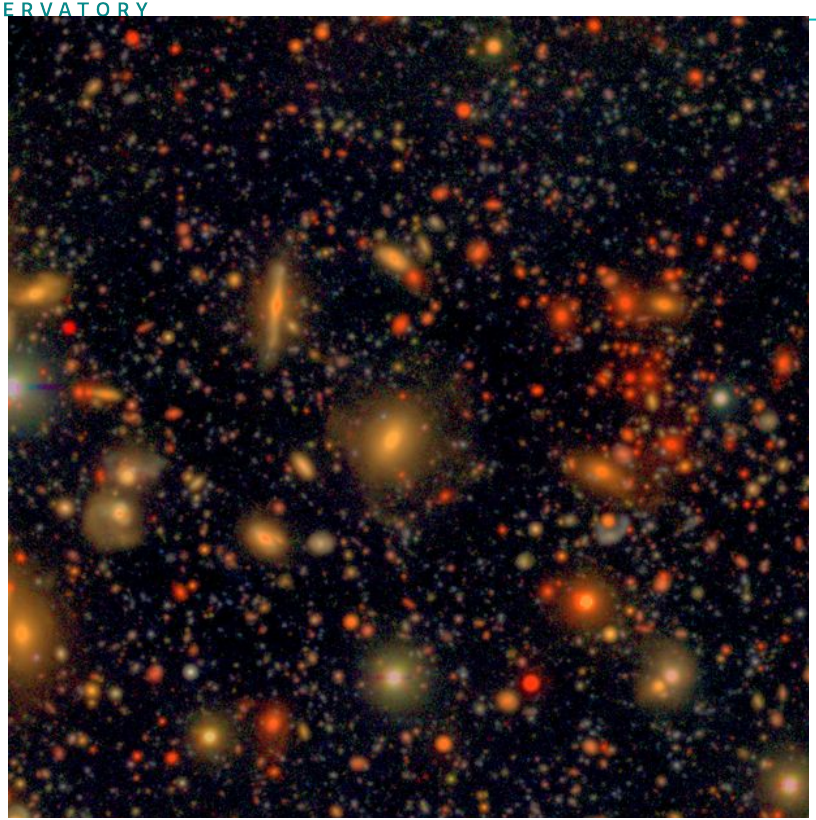


DM must build everything to get Rubin products to users.

- 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.

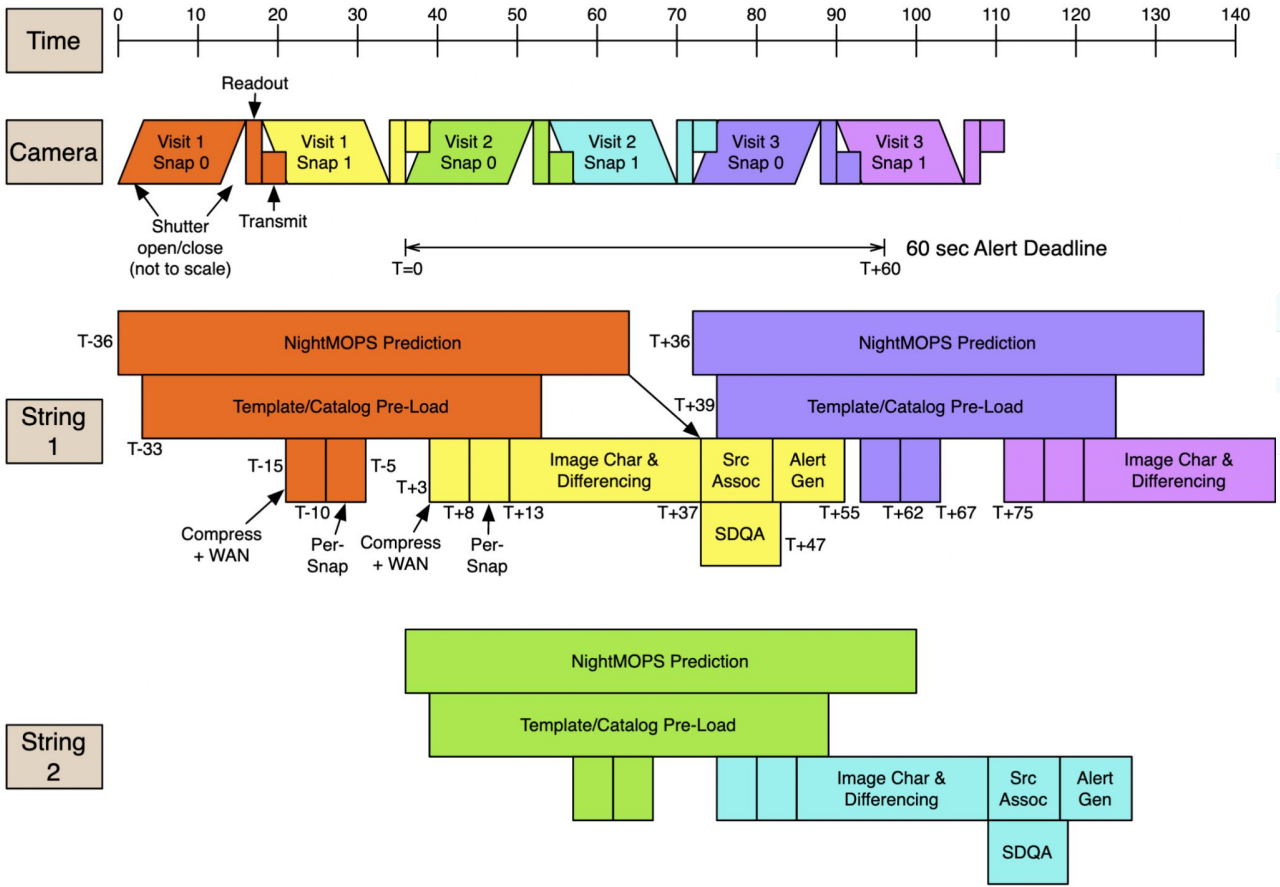
- Rubin's LSST is not the first wide-field imaging survey…
- …but the combination of depth, area, and throughput make it uniquely challenging e.g.
  - Everything is blended.
  - Many measurements are systematics limited.
- Testing on precursors like HSC!

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)

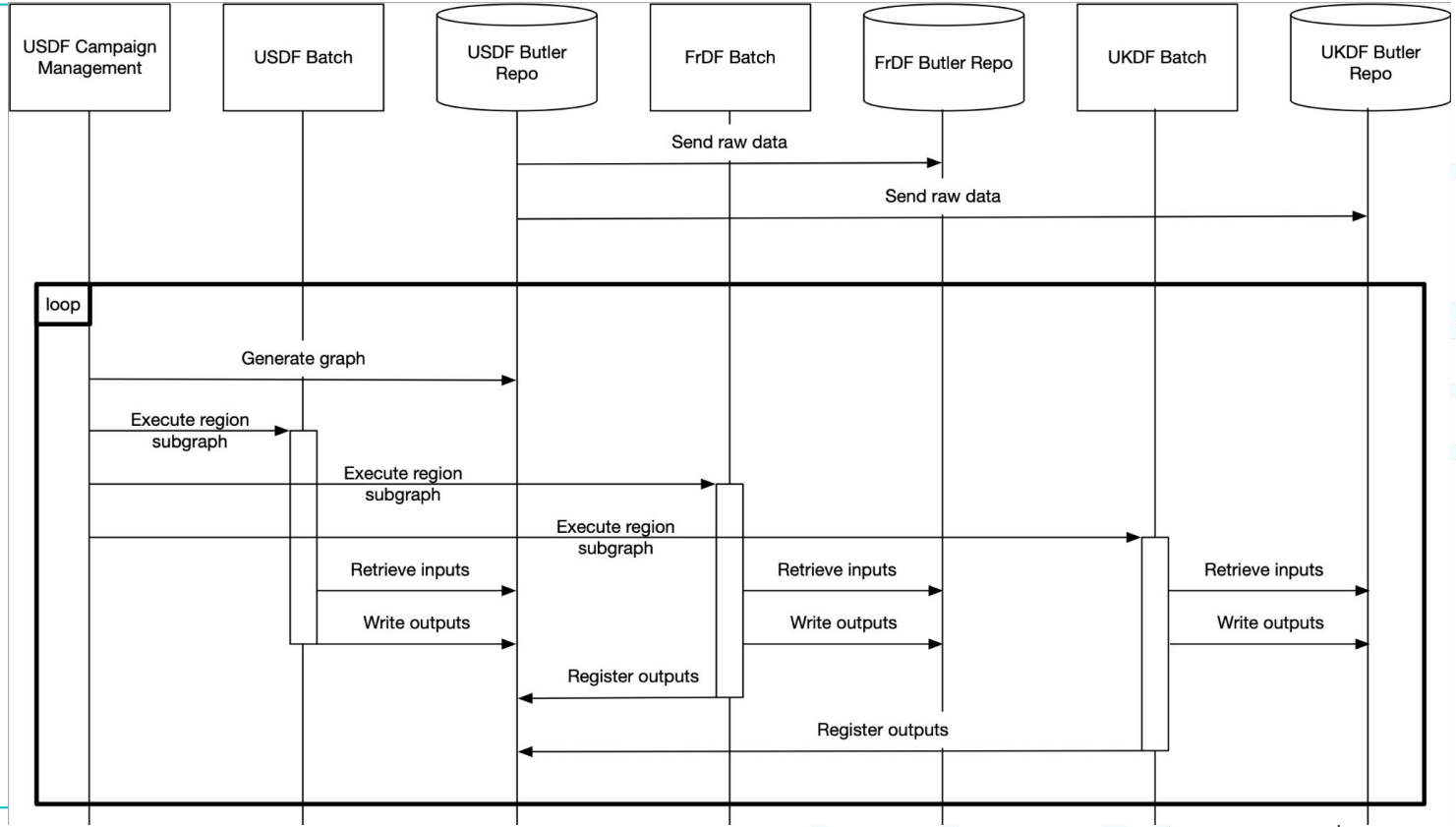
3 sites

US FR UK

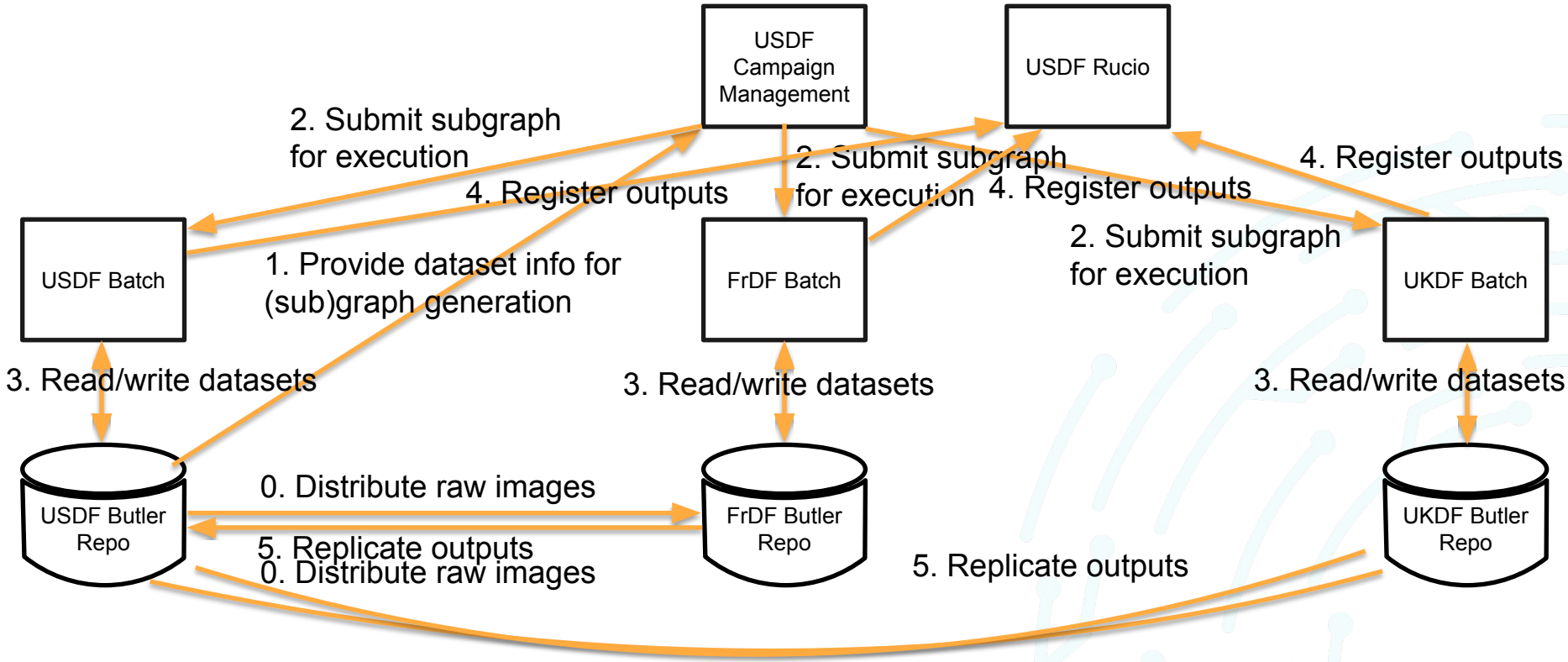
Batch Production System (BPS) in front.

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/Kotlin (Jenness et al., Proc SPIE, 2018).

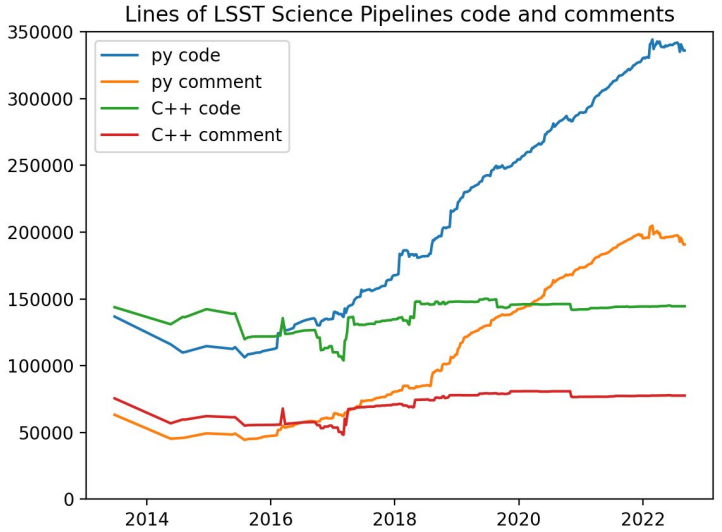


Figure courtesy Tim Jenness.
Based on the Science Pipelines codebase only.

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

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

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,

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.
  - 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 ..

Systems Engineering discontinuity and assumptions

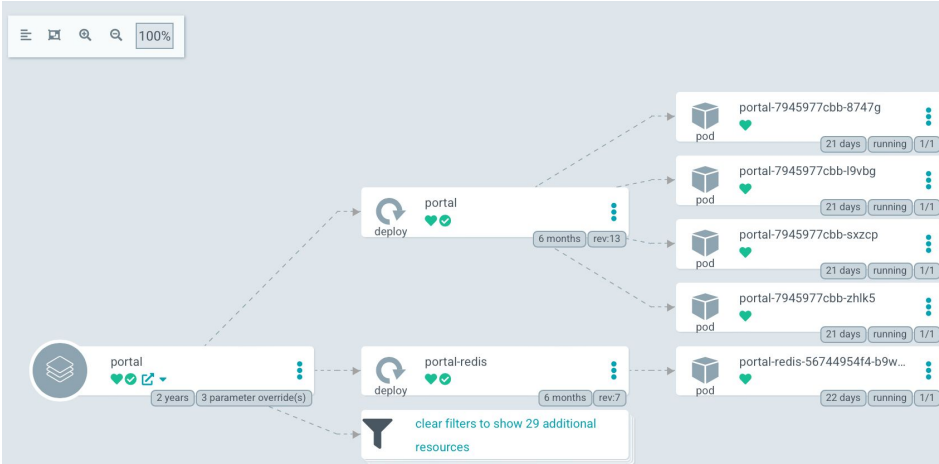
- System Engineering does not scale linearly necessarily (network effect for ICDs etc ..)
- Assumptions made early come back and cost you later
  - Write it all down
  - That is system engineering ..

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,
  -

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

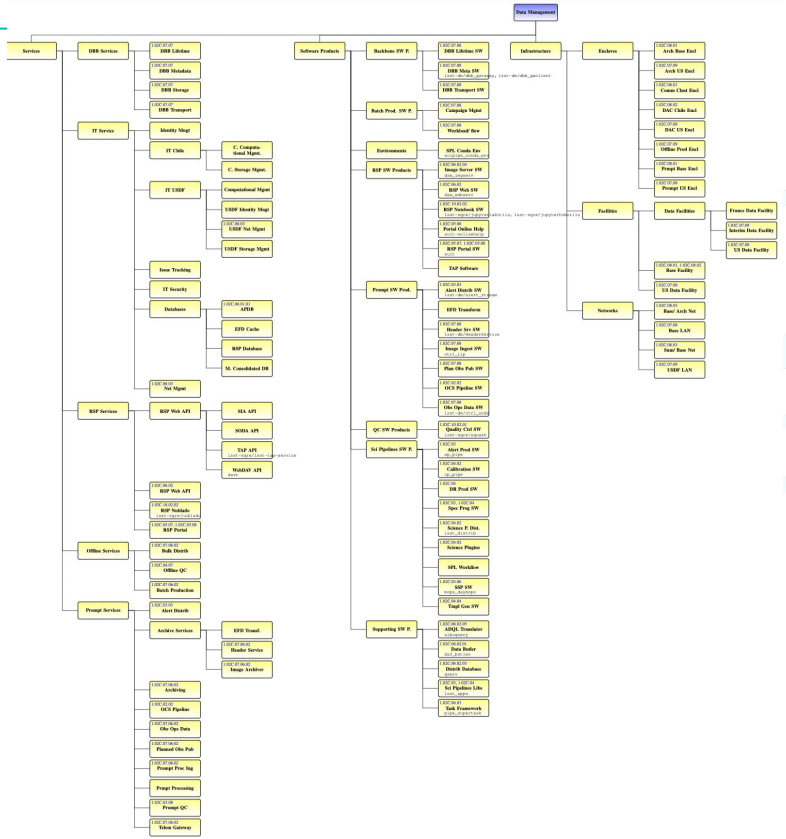
```yaml
environment: summit
fqdn: summit-lsp.lsst.codes
vault_path_prefix: secret/k8s_operator/summit-lsp.lsst.codes

alert_stream_broker:
  enabled: false
cachemachine:
  enabled: true
cert_manager:
  enabled: true
datalinker:
  enabled: false
exposurelog:
  enabled: true
gafaelfawr:
  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)

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
- Gaia
  - Huge SVN repo of everything based roughly on product tree - builds on parts of the SVN tree - dependencies strictly at jar level via
- Middleware was extracted and put on PyPi.
  -



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)
  - We developed a documentation infrastructure that lowers the barrier to documentation by providing templated creation via Slack and uses the same IDE/toolchain developers use for coding, supports Restructured Text 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
- 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

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) -
  - Felis holds the model butler passes Python Objects to clients
- 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)

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
- Influx for summit Engineering, Postgress for observing logs and ancillary info, AlertsDB. Cassandra for Prompt Products
- Pf course in house developed QSERV for catalog access

Build System

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.

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

KT: * Cloud-native architectures: inherent scalability, queues/messaging buses, object store > filesystem, Kubernetes deployment

* Cloud vs hybrid vs on-prem: cloud compute is cheap, storage is expensive, services evolve faster, egress can be managed

* Databases are great; databases are terrible: persistence, ability to query in different ways, ACID (for RDBMS) are good; centralization/replication, schema evolution, performance cliffs, difficulties with multi-user/multi-tenant are not so good; REST APIs in front help a lot; per-application databases, sometimes specialized (Redis, InfluxDB) add resilience and are more manageable nowadays

* Security: even if all your data is open and unauthenticated, you need to control misuse of compute, corruption of data, and service/admin credentials.

Xiuqin - this is a theme combining several suggestions in software system, architecture, design, and leveraging on APIs for Astronomy. I heard from Gregory that SPHEREx adopted some of the data processing pipeline architecture from Rubin. With Rubin in operation soon, its data management system could a very interesting topic.

- Nate Lust and Michelle Gower have both submitted talk abstracts. One for the pipeline design and the other for the way we run batch.

Do not asssume things are obvious .. do more system engineering wrote it down