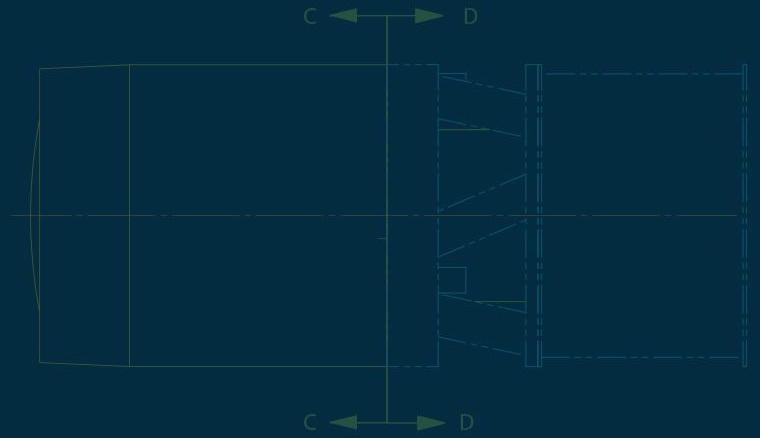
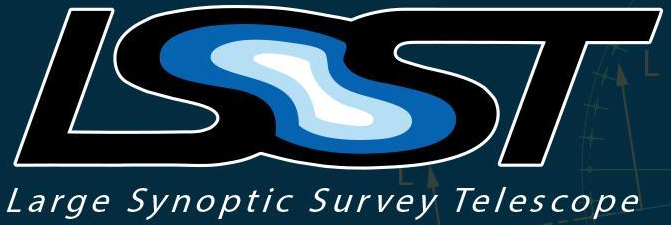
DM Middleware Updates, 2019-08

Agenda

- High-level progress, roadmap, and milestones
- Technical deep-dive into ongoing refactoring work
- PCW Hack Session retrospective



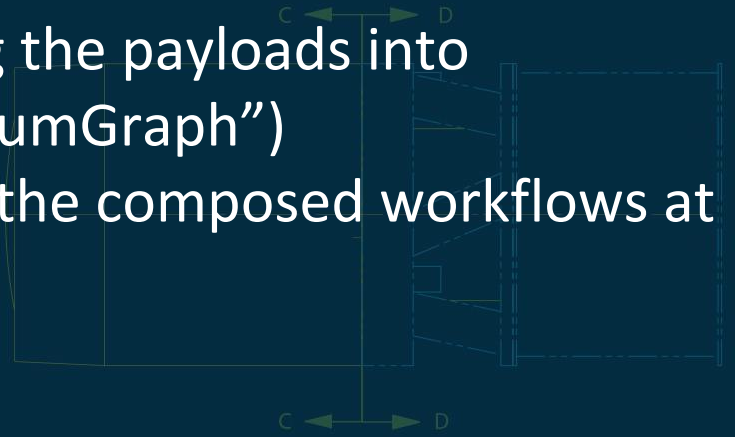
Progress, Roadmap, and Milestones



What We're About

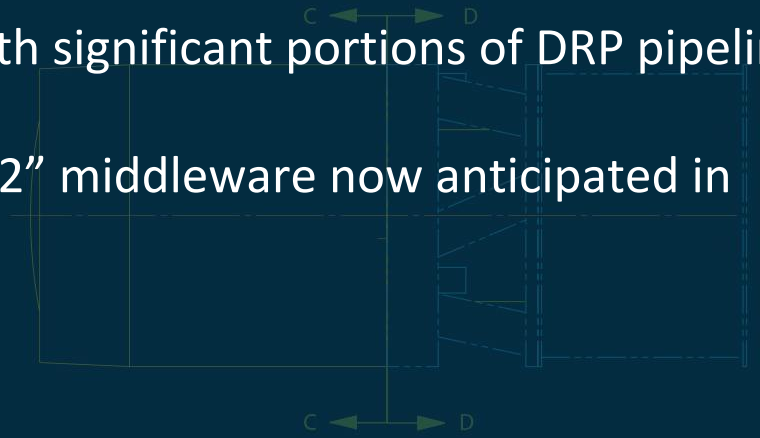
LSST Middleware comprises:

- A framework/API for composable scientific payloads ("PipelineTask")
- An archive abstraction for data consumed and produced by those payloads ("Data Butler")
- A graph abstraction for composing the payloads into computational workflows ("QuantumGraph")
- A system to mediate execution of the composed workflows at scale ("BPS")



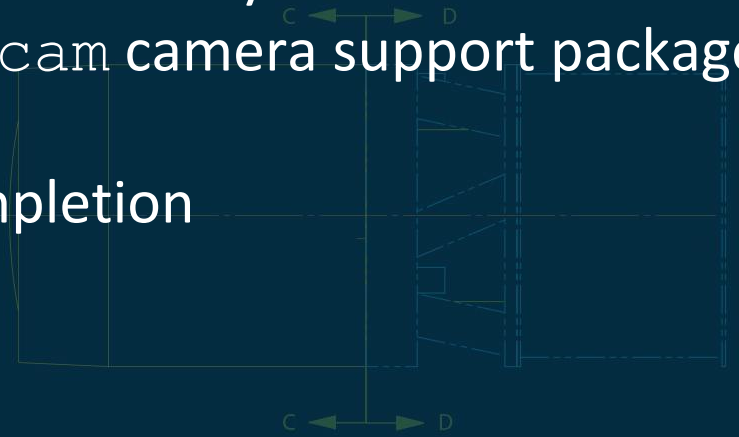
Why “Gen 3”?

- Early middleware systems were underspecified and grew organically as we learned; difficult to maintain and behind schedule
- Decision to replace in 2017; working group convened to compile use-cases and formalize requirements
- Development team assembled to design and implement a ground-up system to meet these requirements
- Implementation now well underway, with significant portions of DRP pipeline already ported to new systems
- Feature parity and deprecation of “Gen 2” middleware now anticipated in early 2020



Recent Progress

- HSC RC2 single-tract reprocessing carried out at NCSA w/ Gen3 Butler, Oracle-backed registry, and prototype BPS workflow
- Gen3 middleware now in use in AWS POC
- PipelineTask API review and improvements
- Gen3 registry refactoring/cleanup underway
- Ports of `obs_lsst` and `obs_decam` camera support packages nearing completion
- RDS/Postgres registry nearing completion

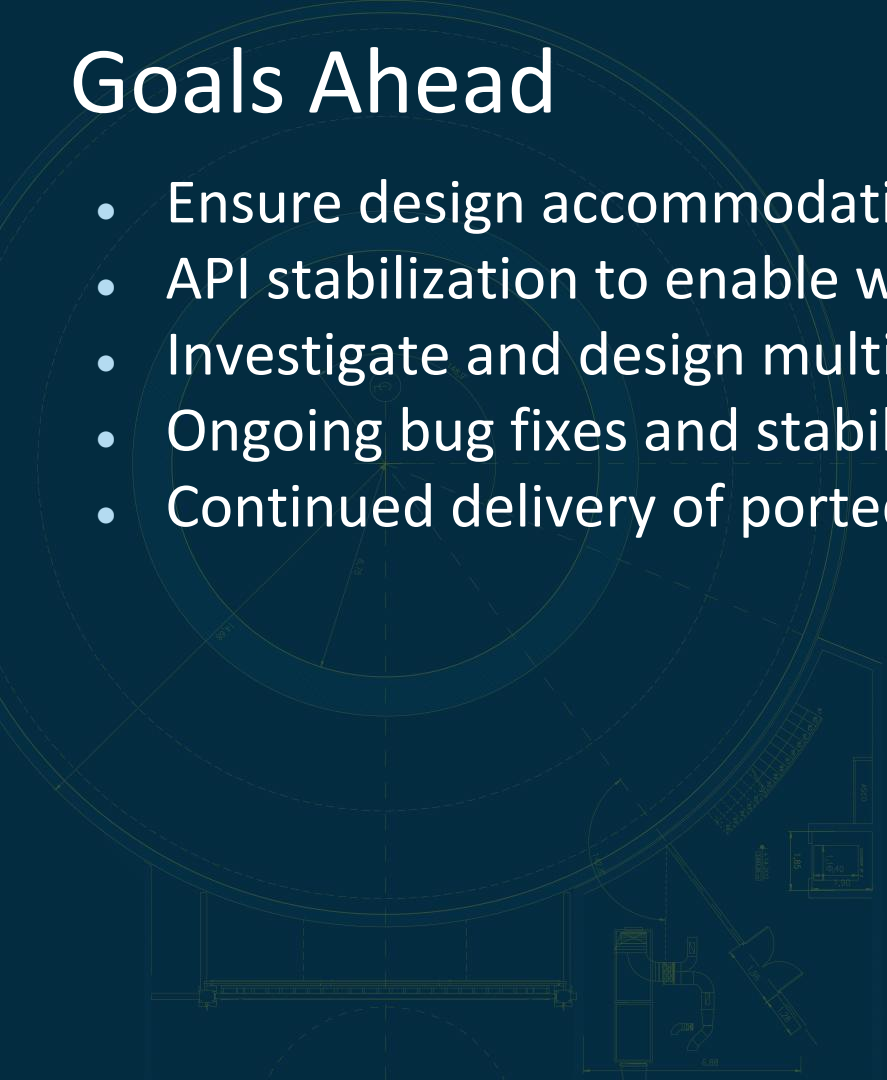


Really Recent Progress (Hack Session)

- Gathered requirements and drew up basic design for Calibration Products Pipeline QuantumGraph generation
- Made significant progress on adding DECam and LSST support for Gen 3
- Had discussions on database access and now have a plan for handling authentication credentials
- Released S3 datastore
- Educated lots of science collaboration members and DM stack users about upcoming middleware changes

Goals Ahead

- Ensure design accommodation for CPP and AP
- API stabilization to enable wider adoption
- Investigate and design multi-user registry for a prototype in fall
- Ongoing bug fixes and stabilization
- Continued delivery of ported tasks

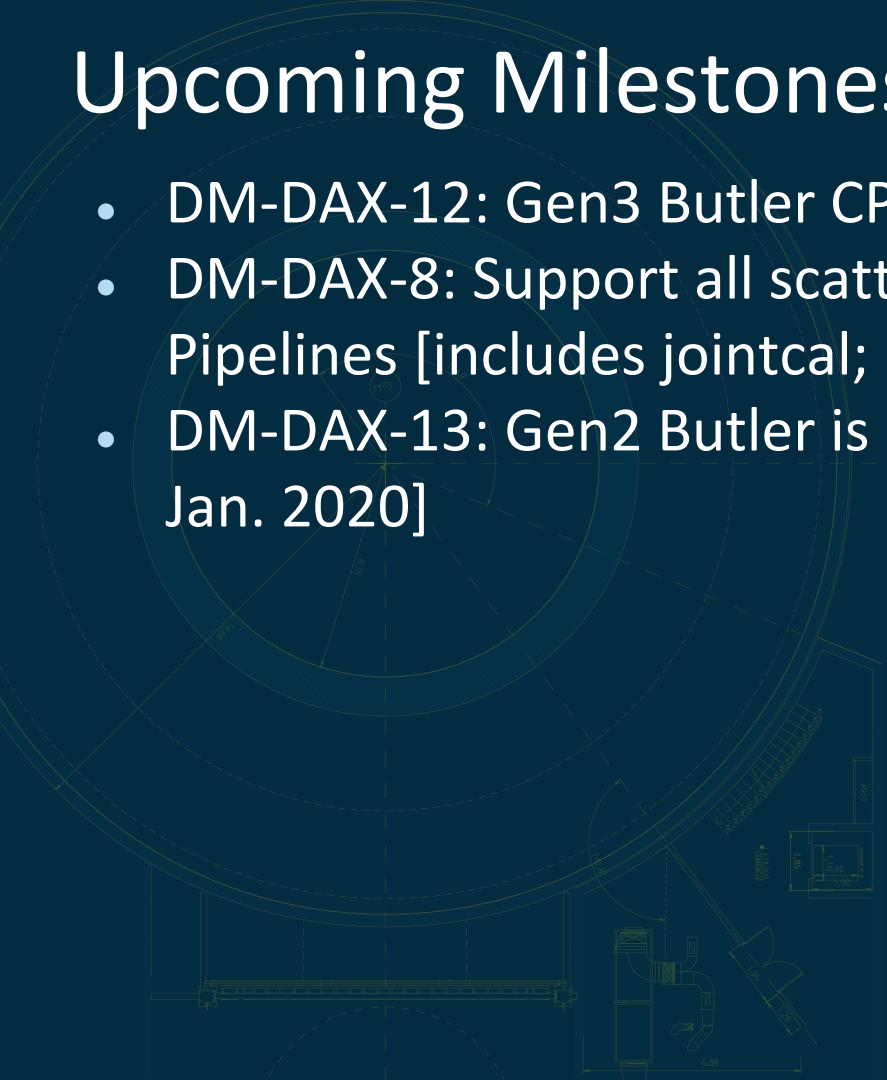


Goals Ahead

- Integration activities at LDF with converted PipelineTasks
- HTCondor pool cluster integration
- QuantumGraph enhancements in support of continued BPS development
- Batch activator

Upcoming Milestones

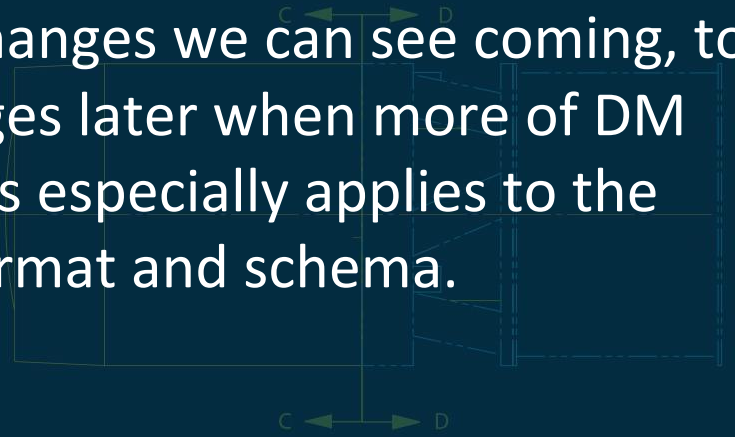
- DM-DAX-12: Gen3 Butler CPP-ready [Sep. 2019]
- DM-DAX-8: Support all scatter-gather needed for current DRP Pipelines [includes jointcal; Nov. 2019]
- DM-DAX-13: Gen2 Butler is deprecated [Gen3 feature parity; Jan. 2020]



Ongoing Registry Refactoring

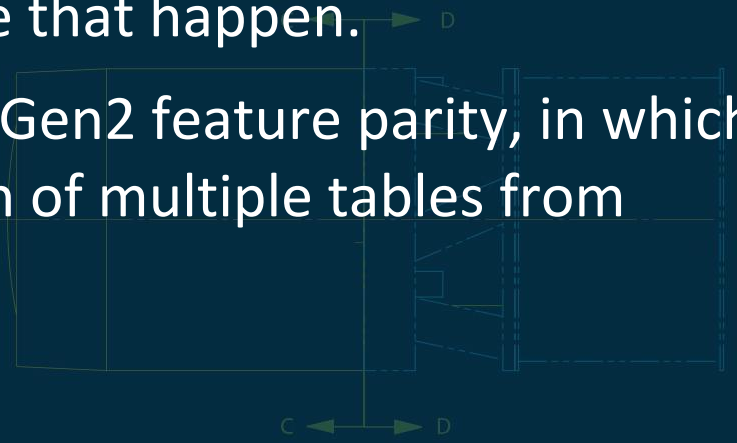
Why now?

- It's time to apply lots of lessons-learned from the last two demo-milestone sprints and recover from (minor) "just get it done" technical debt built up working in that mode.
- We want to make any breaking changes we can see coming, to avoid having to make those changes later when more of DM depends on us for daily work. This especially applies to the on-disk/in-database repository format and schema.



Why now?

- Bulk inserts (i.e. converting Gen2 repos, raw ingest) don't scale well due to structural transaction nesting problems. We know what SQL we should be emitting to speed this up, but we need to restructure the Python to make that happen.
- We need multi-user registries for Gen2 feature parity, in which each logical table is a combination of multiple tables from multiple per-user schemas.



Underway now: DM-17023

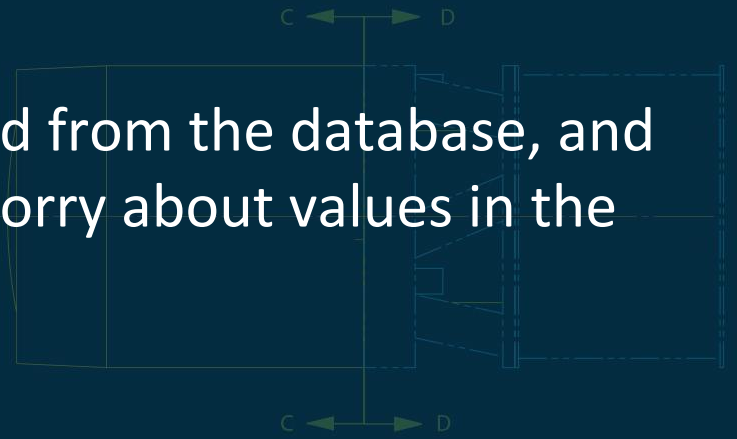
Dimension many-to-many relationships are now either temporal or spatial (*maybe* spectral in the future), rather than explicit pairwise relationships.

- Extensibility: easier to add new dimensions that relate to existing ones without modifying existing ones.
- Lays the groundwork for pulling spatial regions or timestamps for dimensions from dataset tables (e.g. visit-detector regions from raws, validity ranges from flats) rather than dimension tables.

Underway now: DM-17023

Classes for validated data IDs are immutable and structured as zipped keys and values sequences.

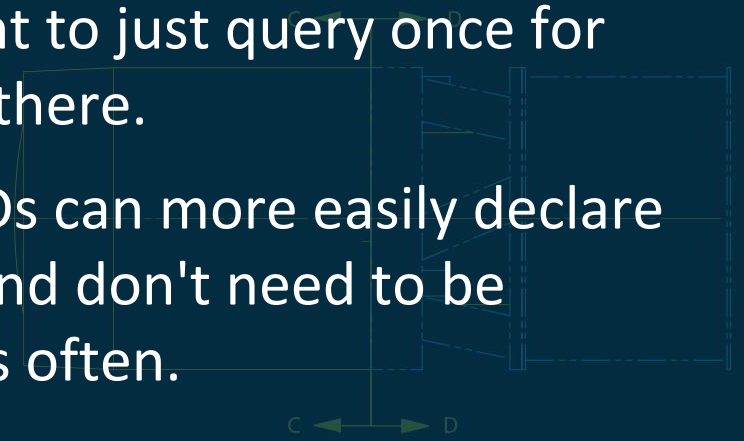
- Easier to share (common) key sequences between data IDs instead of duplicating them.
- Easier to cache metadata retrieved from the database, and safer because we don't need to worry about values in the cache changing.



Underway now: DM-17023

Classes for validated data IDs are either minimal with only the required keys, or fully expanded to include all dimension metadata.

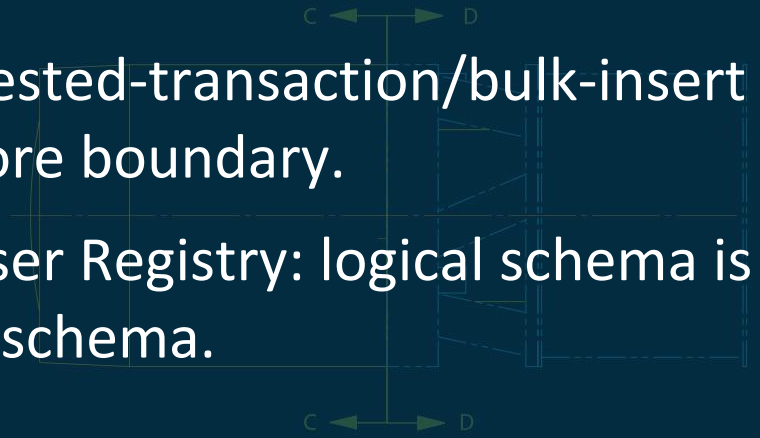
- Avoids complex, slow logic for conditionally querying metadata; it's much more efficient to just query once for everything and then know it's all there.
- Python APIs that consume data IDs can more easily declare what kind of data ID they need, and don't need to be responsible for validating them as often.



Underway now: DM-17023

Schema for dimension tables is now partially machine-generated from dimension relationships.

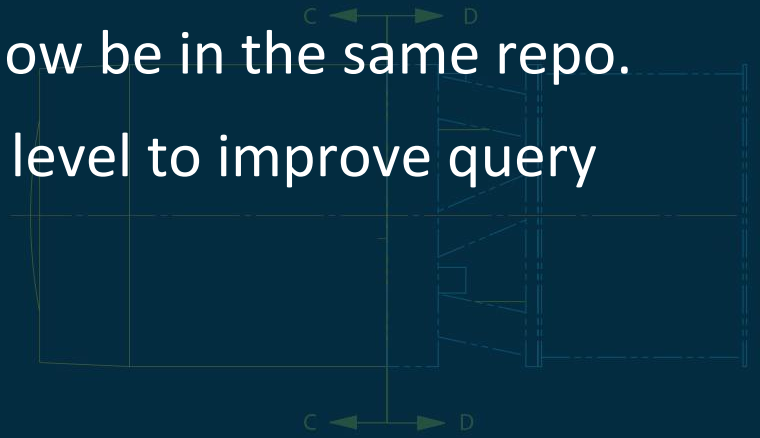
- Reduces duplication in configuration.
- Lays the groundwork for fixing nested-transaction/bulk-insert problems at the Registry/Datastore boundary.
- Lays the groundwork for multi-user Registry: logical schema is decoupled from actual database schema.



Underway now: DM-17023

The "common" skypix system used to tied together spatial dimensions no longer has to be the same one used by all reference catalogs.

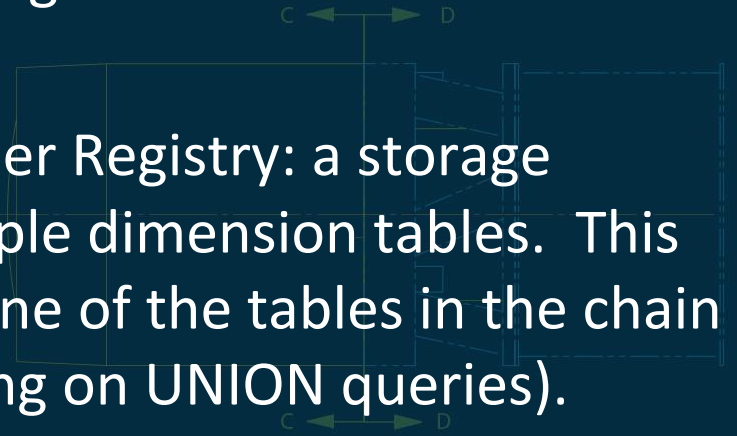
- Different reference catalog can now be in the same repo.
- We can tune the common skypix level to improve query performance.



Underway now: DM-17023

Database backing for dimension tables has been pulled out of Registry into a separate class hierarchy (`DimensionRecordStorage`):

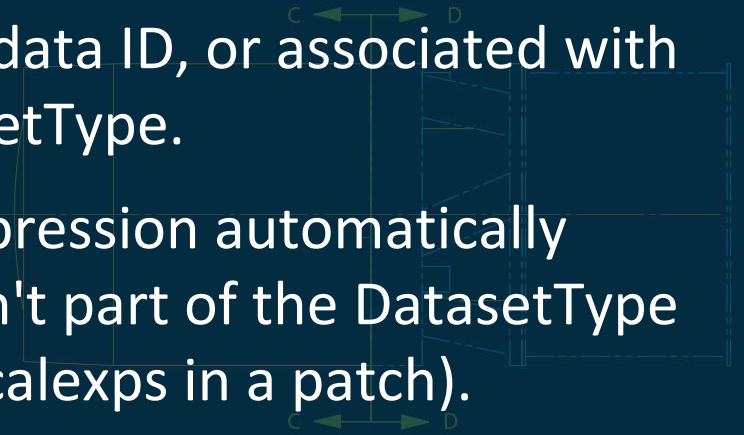
- Makes dimension metadata caching easier and more configurable.
- Prototypes one aspect of multi-user Registry: a storage implementation that chains multiple dimension tables. This includes limiting queries to only one of the tables in the chain in some contexts (instead of relying on UNION queries).



Underway now: DM-17023

Dimension-based query/expression system has been largely rewritten:

- New public APIs for retrieving data IDs or DatasetRefs matching an expression, related to a given data ID, or associated with existing dataset of another DatasetType.
- Mentioning a dimension in an expression automatically includes it in a query, even if it isn't part of the DatasetType being queried for (e.g. query for calexps in a patch).



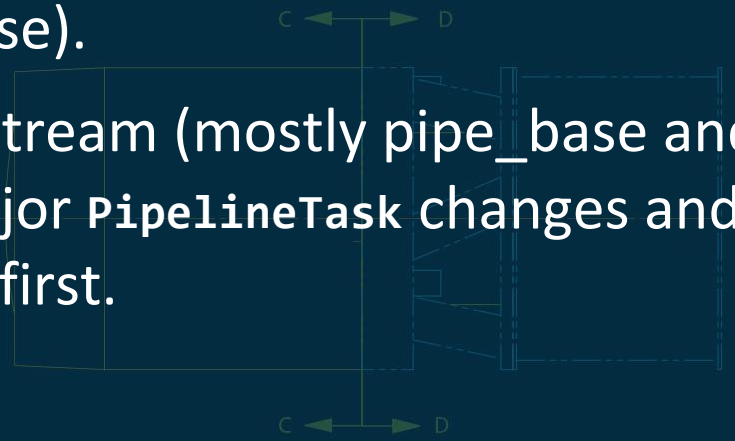
Underway now: DM-17023

Dimensions can now have multiple unique keys (e.g. both string names and integer IDs for visits and detectors).

- One of these still has to be the primary key (and that can't be different for different instruments), but it ensures that each instrument we support can have the kind of identifier its team expects.
- Alternate keys can already be used in query expressions, and could be usable directly in data ID dictionaries in the future.

DM-17023 status

- daf_butler branch is functionally complete and passing tests, 90% of new code is documented. Now is probably the time to write overview docs for the dimension classes as well (to help the poor reviewer(s), if nothing else).
- Still need to make changes downstream (mostly pipe_base and obs_base); will wait for Nate's major `PipelineTask` changes and John's `Instrument` changes to land first.



Next up: normalizing the dataset table[s]

We currently have a monolithic `dataset` table with records of all DatasetTypes.

- It has columns for every possible data ID key, even though each record only utilizes a small fraction of these;
- In our performance-critical queries, we *always* include the `dataset` table via subqueries like:

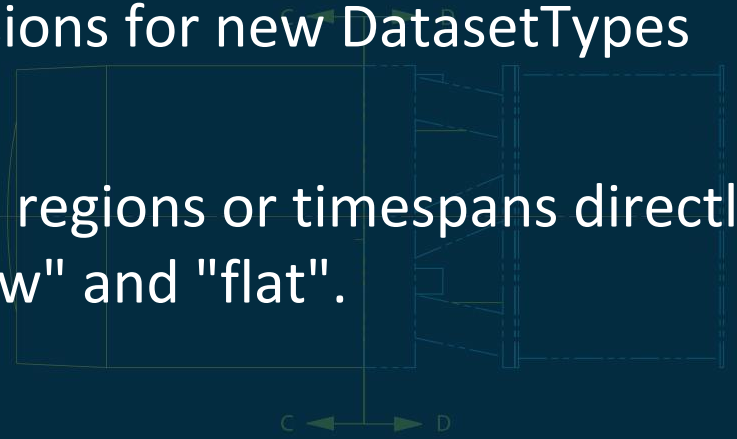
  ```
  SELECT ... FROM dataset WHERE dataset_type_name = <literal>
  ```

  (often multiple times in a single larger query).

Next up: normalizing the dataset table[s]

What I want instead: one thin `dataset` table with no data ID columns for all DatasetTypes, as well as per DatasetType tables with only the data ID columns relevant for that DatasetType.

- Extensibility: can add new dimensions for new DatasetTypes without altering existing tables.
- With DM-17023 changes, can add regions or timespans directly to per-DatasetType tables like "raw" and "flat".



Next up: denormalizing collections

We currently have a **`dataset_collections`** table that has one entry for every dataset + collection combination. Those **`dataset`** subqueries actually always look like:

```
SELECT ...
FROM dataset INNER JOIN dataset_collection
    ON dataset.dataset_id = dataset_collection.dataset_id
WHERE dataset.dataset_type_name = <literal> AND
    dataset_collection.collection IN (<literals>)
```

(or worse, if we are trying to select from an *ordered* list of input collections).

Next up: denormalizing collections

The common case is that a dataset is in only one collection, not many.

I'd like to try just putting the collection column into the per-DatasetType dataset tables, and duplicate the rest of the record when a dataset appears in multiple collections.

This would also make it much more natural to define the unique constraint on DatasetType + data ID + collection.

We could also consider per-DatasetType, per-collection tables.

Next up: fixing unique conflict resolution

Many of our nested-transaction performance problems stem from using them with expected unique constraint violations to support two specific use cases:

- when ingesting raw data, silently ignore files that have already been ingested (when configured to);
- when associating a dataset with a collection, do nothing if that exact dataset is already in that collection, but raise an exception if a different dataset with the same data ID is.

Next up: fixing unique conflict resolution

I think we can avoid using nested transactions in either case:

- in ingest, just retrieve a list of ingested exposures from the database in advance, and ignore duplicates in Python;
- when associating a dataset with a collection, disassociate any dataset with the same data ID that is already in that collection.

This should remove a lot of SAVEPOINT calls in both Oracle and SQLite, and possibly allow us to stop using extremely aggressive locking in SQLite as well.

Next up: abstracting dataset storage

I'd like to pull dataset insertion and retrieval out of `SqlRegistry` itself and into a chainable interface like `DimensionRecordStorage`.

- Lays most of the rest of the groundwork for multi-user Registry.
- As we rewrite these APIS, we'll do bulk insertion and transaction nesting right this time, and resolve at least most of the scaling problems we currently have in ingest/gen2convert.
- Abstracts away the design choices from the last few slides that I'm less certain about, so we're not locked into any of them.

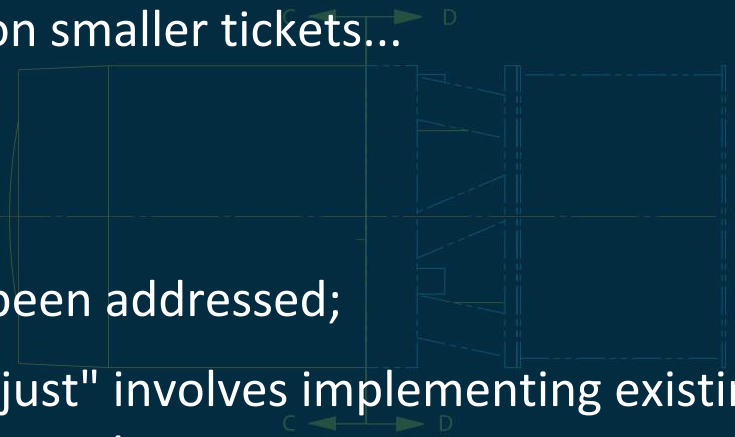
Where does that leave us?

That's two big tickets:

- DM-17023 is probably 50%;
- the dataset/collection/transaction refactoring is another 30%;
- the last 20% is easier to split up and do on smaller tickets...

where 100% means:

- Registry schema is *structurally* stable;
- major scaling/performance issues have been addressed;
- multi-user Registry isn't done, but now "just" involves implementing existing abstract interfaces and working out construction patterns.



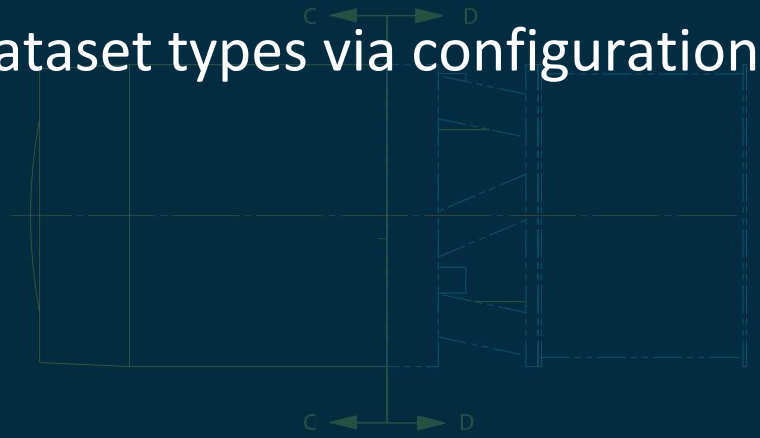
Most of the last 20%:

- Split autoincrement IDs into autoincrement + site IDs.
- Generalize visit/exposure and filter relationships.
- Immutability, sane comparisons, and more validation guarantees in the DatasetRef class.
- [Consider/evaluate] multi-collection Butlers and collection chaining.
- [Consider/evaluate] using ranges for skypix IDs in the database.
- Subpackage restructuring: **`daf.butler.core`** is getting unwieldy and disorganized.
- Merge SqlRegistry and Registry: it's time to admit that all Registry classes will use SQLAlchemy.

Hack Session Retrospective

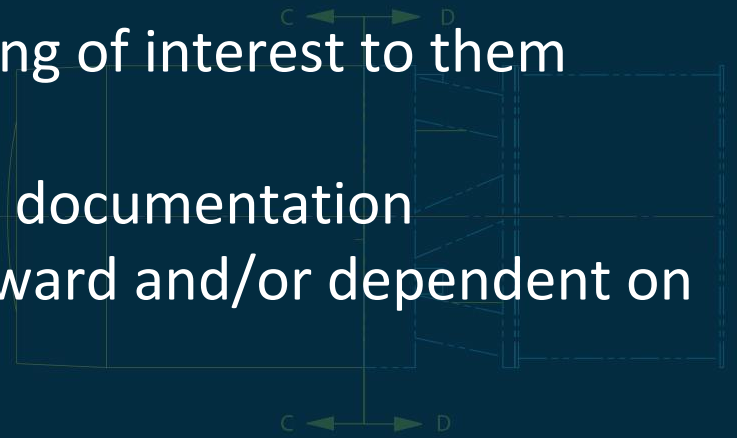
Went Well / Likes

- Plenty of people around to help/answer questions (+)
- Hack session is good alternative to fill schedule “holes”
- Nate L. is a great communicator/evangelist!
- Gen3 MW looks good from obs package perspective: a lot less to write ad greater clarity
- Like the feature to declare new dataset types via configuration



Could Be Better / Dislikes

- Specific goals relied on assumption just the right people would be there; with schedule conflicts not so
- Messaging on agenda was unclear:
  - Some missed an ad-hoc tutorial that they really wanted to see.
  - Others stopped by when nothing of interest to them specifically was occurring
- Hard to get started without better documentation
- Example topics heavily focused toward and/or dependent on DRP pipelines codes/concepts



Suggestions

- Some simple canned recipes/tutorials for basic tasks
- Some persistent record of material presented
- Have a Gen3 rewrite of the HSC tutorial
- Some notification of new things coming out (community?)
- Explicitly schedule sub-meetings for specific goals.
- Clarify the agenda!
- Share a session w/ stack club?
- Do this kind of activity more often, possibly at smaller scale?