From Hydras to TACOs: Evolving the Stanford Digital Repository

ELAG 2018

Christina Harlow, Erin Fahy

STANFORD UNIVERSITY LIBRARIES

http://bit.ly/HydrasToTacos

Goals of this Talk

1. Introduce the Stanford Digital Repository
2. Discuss our Approach(es) to re-architecting our system
3. Introduce SDR3, TACO, & our redesign so far
4. What’s next?

Goals of this Talk

1. Introduce the Stanford Digital Repository
2. Discuss our Approach(es) to re-architecting our system
3. Introduce SDR3, TACO, & our redesign so far
4. What's next?

We'd really love to hear your feedback on this work!

And special thanks to the Bootcamp group that went through a fast-paced deep dive of some of this work on Monday.

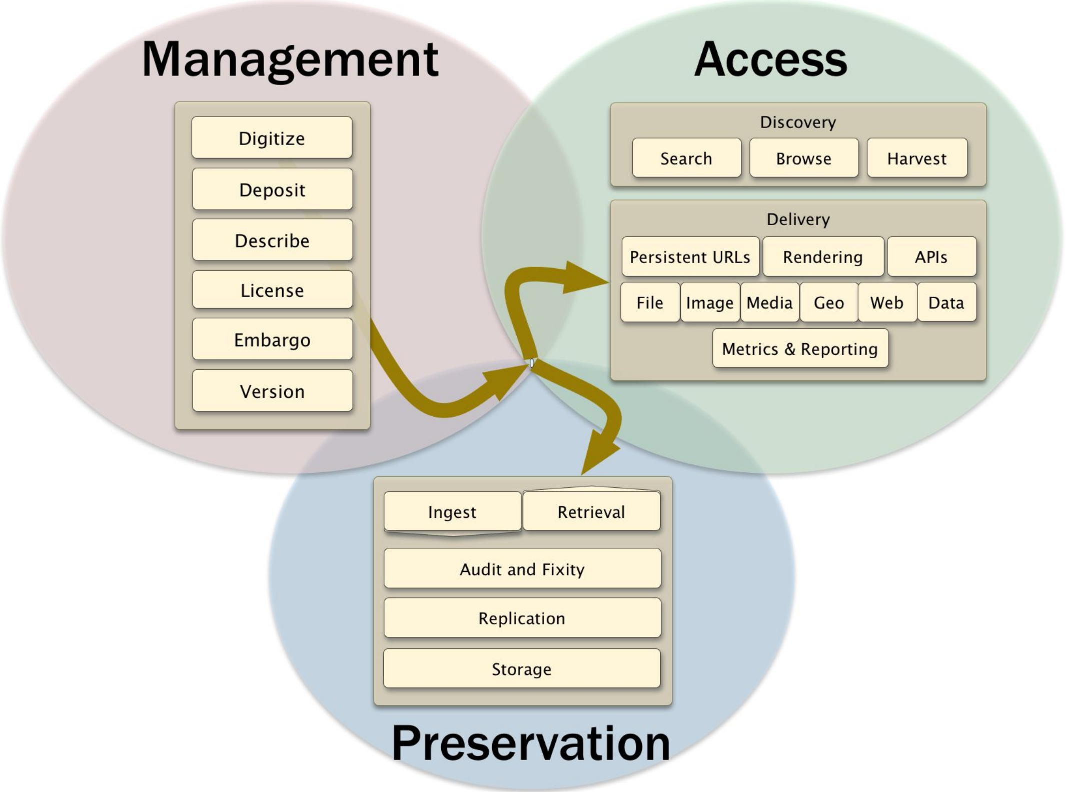
1. Some Context on SDR

Stanford Digital Repository (SDR)

Currently in it’s second iteration, architecturally (i.e. “SDR2”)

Been working for over ten years

Guided by a ‘3 Spheres Topology’



http://bit.ly/HydrasToTacos

Stanford Digital Repository

Variety of digital resources & assets:

- Bulk ingest of digitization labs work,
- Institutional repository self-deposit,
- Electronic dissertations & theses self-deposit,
- Geo-datasets,
- Web archiving,
- Electronic resources cataloged & preserved,
- ...

Stanford Digital Repository Metrics

Manages roughly **1.6 million** distinct resources currently

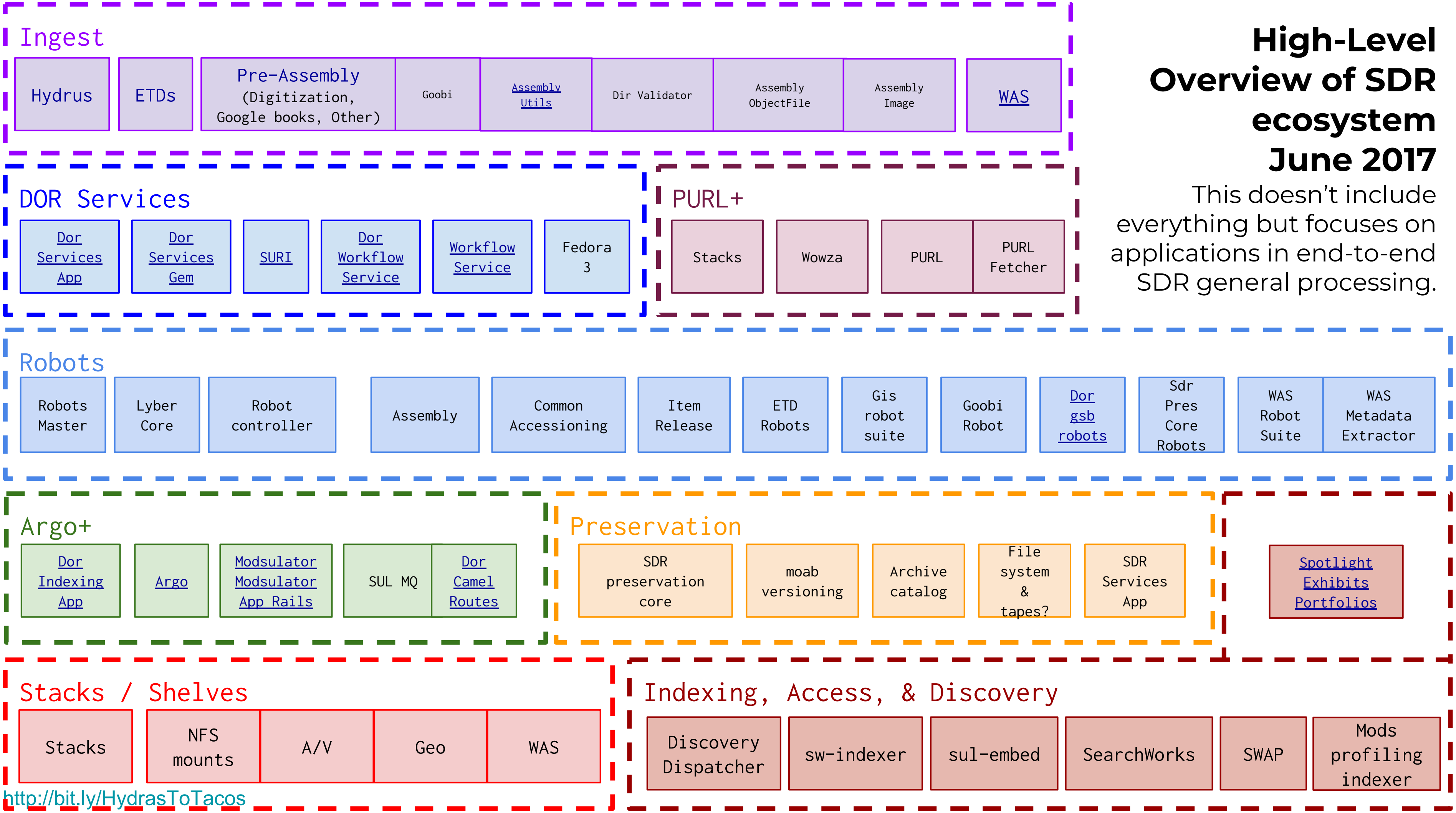
Has about **half a petabyte (455 TB)** of digital assets in our preservation layer

~426 TB of digital assets in our repository staging systems

455 TB of digital assets & **59.1 GB** of metadata in our access system(s)

High-Level Overview of SDR ecosystem June 2017

This doesn't include everything but focuses on applications in end-to-end SDR general processing.



Ingest

- Hydrus
- ETDs
- Pre-Assembly (Digitization, Google books, Other)
- Goobi
- Assembly Utils
- Dir Validator
- Assembly ObjectFile
- Assembly Image
- WAS

DOR Services

- Dor Services App
- Dor Services Gem
- SURI
- Dor Workflow Service
- Workflow Service
- Fedora 3

PURL+

- Stacks
- Wowza
- PURL
- PURL Fetcher

Robots

- Robots Master
- Lyber Core
- Robot controller
- Assembly
- Common Accessioning
- Item Release
- ETD Robots
- Gis robot suite
- Goobi Robot
- Dor gsb robots
- Sdr Pres Core Robots
- WAS Robot Suite
- WAS Metadata Extractor

Argo+

- Dor Indexing App
- Argo
- Modsulator Modsulator App Rails
- SUL MQ
- Dor Camel Routes

Preservation

- SDR preservation core
- moab versioning
- Archive catalog
- File system & tapes?
- SDR Services App

- Spotlight Exhibits Portfolios

Stacks / Shelves

- Stacks
- NFS mounts
- A/V
- Geo
- WAS

Indexing, Access, & Discovery

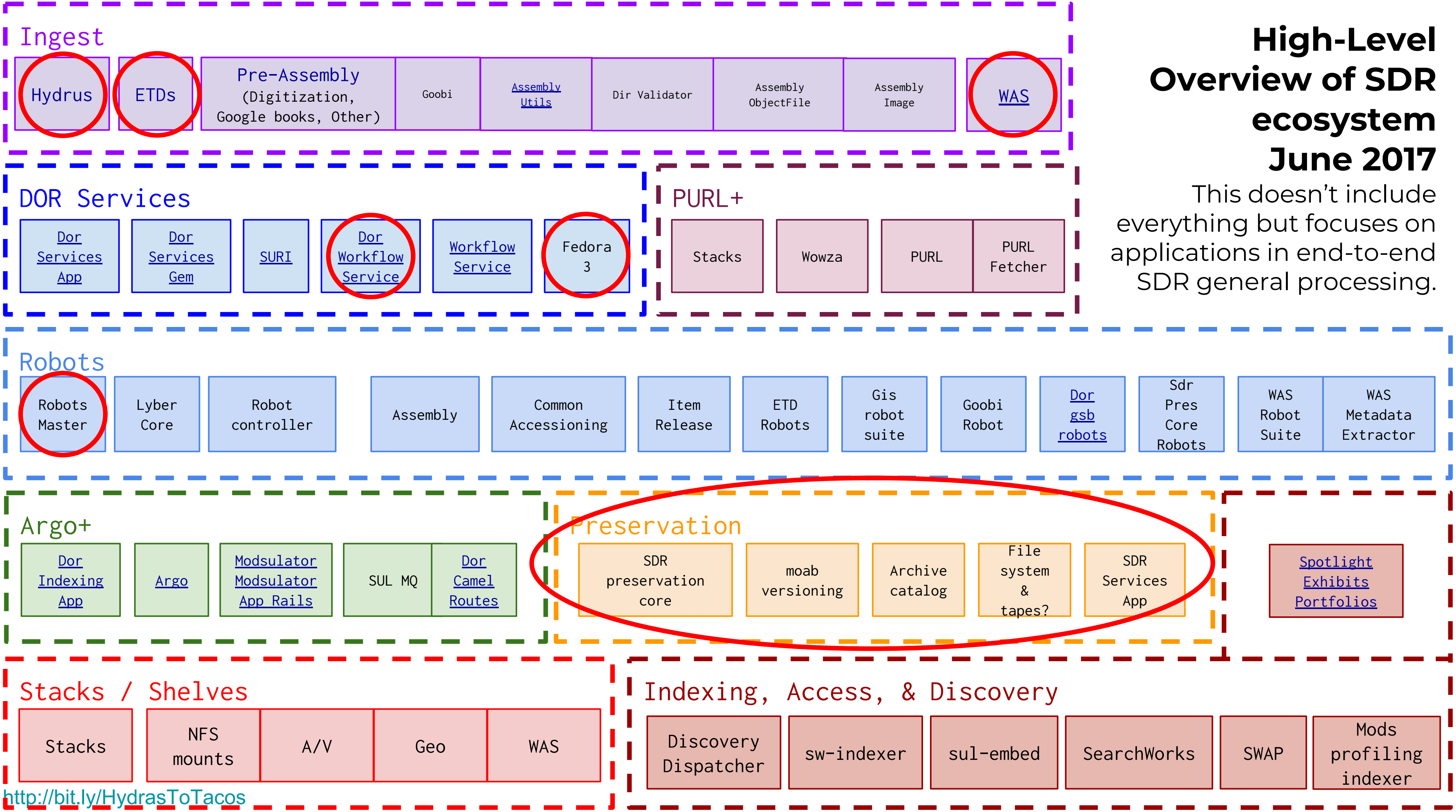
- Discovery Dispatcher
- sw-indexer
- sul-embed
- SearchWorks
- SWAP
- Mods profiling indexer

http://bit.ly/HydrasToTacos

2. Our Approach(es) to re-architecting our system

High-Level Overview of SDR ecosystem June 2017

This doesn't include everything but focuses on applications in end-to-end SDR general processing.



Ingest

- Hydrus
- ETDs
- Pre-Assembly (Digitization, Google books, Other)
- Goobi
- Assembly Utils
- Dir Validator
- Assembly ObjectFile
- Assembly Image
- WAS

DOR Services

- Dor Services App
- Dor Services Gem
- SURI
- Dor Workflow Service
- Workflow Service
- Fedora 3

PURL+

- Stacks
- Wowza
- PURL
- PURL Fetcher

Robots

- Robots Master
- Lyber Core
- Robot controller
- Assembly
- Common Accessioning
- Item Release
- ETD Robots
- Gis robot suite
- Goobi Robot
- Dor gsb robots
- Sdr Pres Core Robots
- WAS Robot Suite
- WAS Metadata Extractor

Argo+

- Dor Indexing App
- Argo
- Modsulator Modsulator App Rails
- SUL MQ
- Dor Camel Routes

Preservation

- SDR preservation core
- moab versioning
- Archive catalog
- File system & tapes?
- SDR Services App

- Spotlight Exhibits Portfolios

Stacks / Shelves

- Stacks
- NFS mounts
- A/V
- Geo
- WAS

Indexing, Access, & Discovery

- Discovery Dispatcher
- sw-indexer
- sul-embed
- SearchWorks
- SWAP
- Mods profiling indexer

http://bit.ly/HydrasToTacos

SDR2 ‘Retrospective’

- Lack of full system comprehension
- Lots of unmaintained codebases & workflows
- Over-engineered components
- Pain points on adding new features or processes
- Mismatch of design(s) & implementation(s)

> *“There are a lot of interaction points between layers of the technology stack and you often need to know a lot about all of these interactions even if you are only currently concerned with one part of the stack.”*

Looked to Samvera / Hydra & Hybox

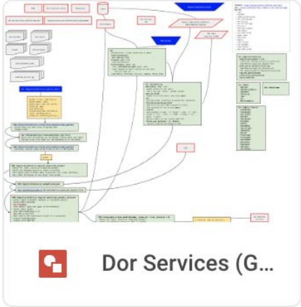
Hyrax

samvera

http://bit.ly/HydrasToTacos

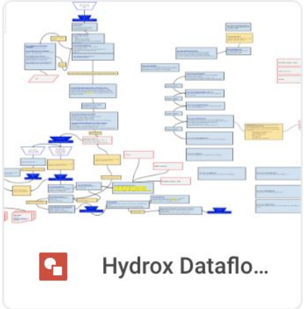
SDR3 Design Cycle

- 3 months of daily 1 hour meetings with architect, engineers, product owners, administrators, & others
- Produced requirements independent of system expectations
- Built shared understanding of our current needs & conceptual architecture
- In tandem: did a ‘current state’ deep dive on our existing code
- Generated a high-level conceptual design & plan

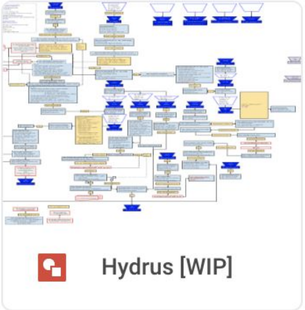


Dor Services (G...

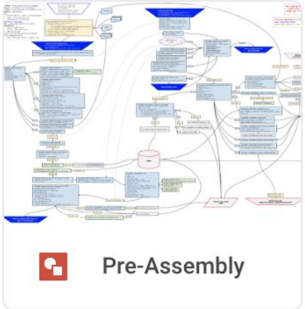
Dor Services Ap...



Hydrox Dataflo...



Hydrus [WIP]



Pre-Assembly

http://bit.ly/HydrasToTacos

Hyrax Analysis: SDR Options

1. Do not use Hyrax at all for SDR3. Non-starter.
2. Use Hyrax for SDR3 entirely. However...
    a. ~38% of our core, reviewed requirements are not covered by Hyrax.
    b. ~24% of those are ‘Maybe’, i.e. require config, model changes, or coding.
    c. Most alignment with UI / Self-Deposit, direction of analytics, web dev.
    d. Least alignment in overall architecture, bulk processing, back-end needs.
3. Integrate Hyrax & SDR3 via components & ‘seams’.

Hyrax Analysis: Possible ‘Seams’

- Valkyrie’s “internal air gap” approach for flexible data stores
- Actor Stack, Sipity, or Delayed Jobs:
  - Write Hyrax MiddlewareStack as seam to our Management API & asynchronous processing.
- Rely on both internal air gaps as well as crisp boundaries via ReST APIs.
  - Independent scalability.
  - Migration ‘hinge’ for components that don’t or shouldn’t fit into Hyrax.
  - Keeps separate areas of our work most aligned with the Samvera community:
    - self-deposit & access/discovery currently
    - analytics and administration dashboards in the future

Fedora 4 / Fedora API Analysis

- Incompleteness & uncertainty of specification work
- Graph store limitations
  - Keep Linked Data out of our back-end system
- Complexity & Comprehensibility
- Performance & Extensibility
- Data & Resource Handling
- System Expectations
- Re-approach Fedora overlap with our data publication (Access) systems

3. “SDR3” & TACO

SDR3 Evolutionary Plan

SDR3 Design Kick-Off (x3) & Hydrox Analysis Phase

(10/03/2017-01/12/2018)

TACO Skeleton Prototype Phase

- TACO Prototype Work Cycle *(01/12/2018-04/26/2018)*
- SDR3 Design Iteration *(4 month)*
- ETDs ⇔ TACO Prototype, Bulk smoke test *(3 months)*
- SDR3 Design Iteration *(1 month)*

TACO Prototype Integration Phase

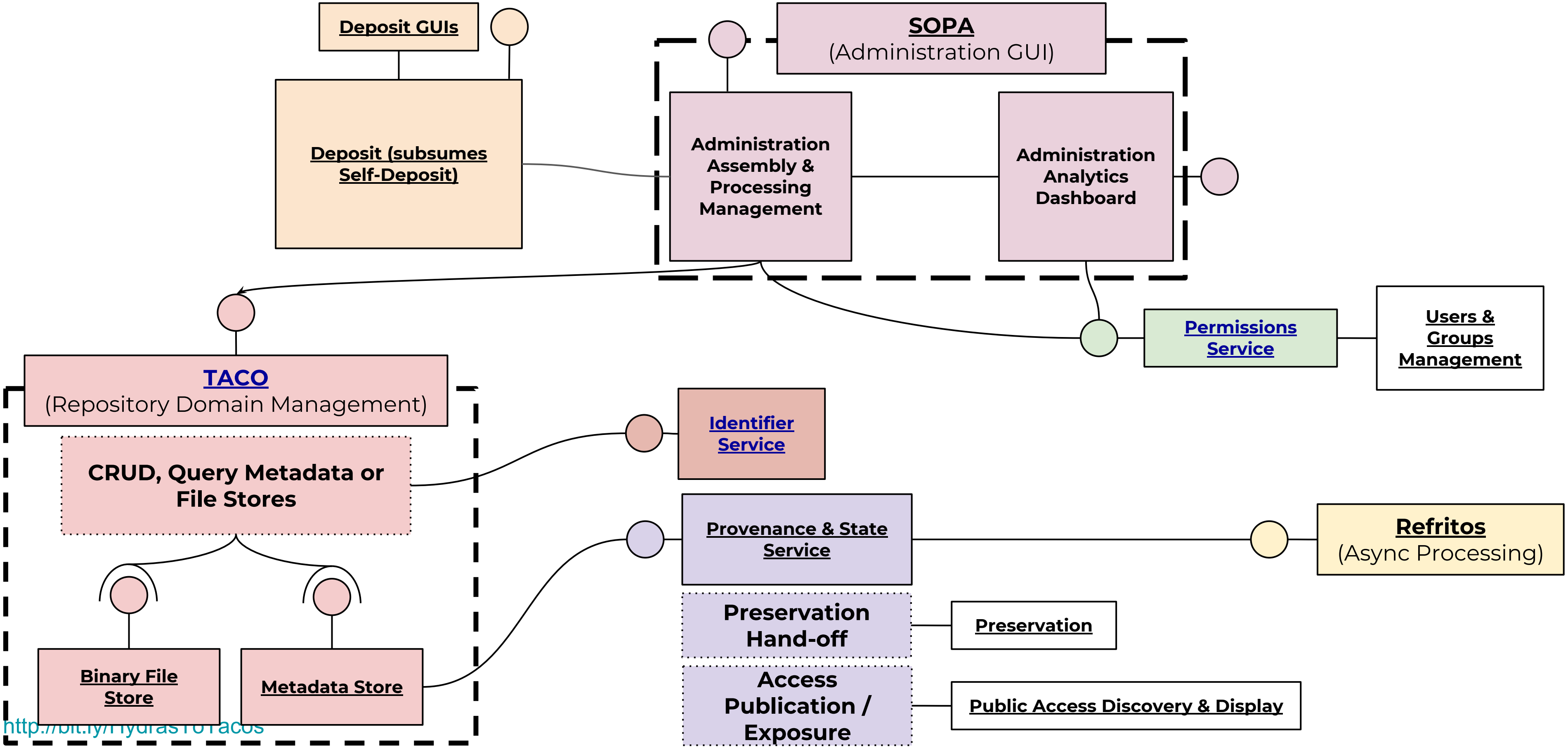
ETDs ⇔ TACO “go live” & data migration

ETDs ⇔ Hydrus “go live” & data migration

...

http://bit.ly/HydrasToTacos

SDR3 High Level Conceptual Design (so far)


Deposit GUIs
Deposit (subsumes Self-Deposit)
SOPA
(Administration GUI)
Administration Assembly & Processing Management
Administration Analytics Dashboard
Permissions Service
Users & Groups Management
TACO
(Repository Domain Management)
CRUD, Query Metadata or File Stores
Identifier Service
Provenance & State Service
Refritos
(Async Processing)
Preservation Hand-off
Preservation
Access Publication / Exposure
Public Access Discovery & Display
Binary File Store
Metadata Store

http://bit.ly/HydrasToTacos

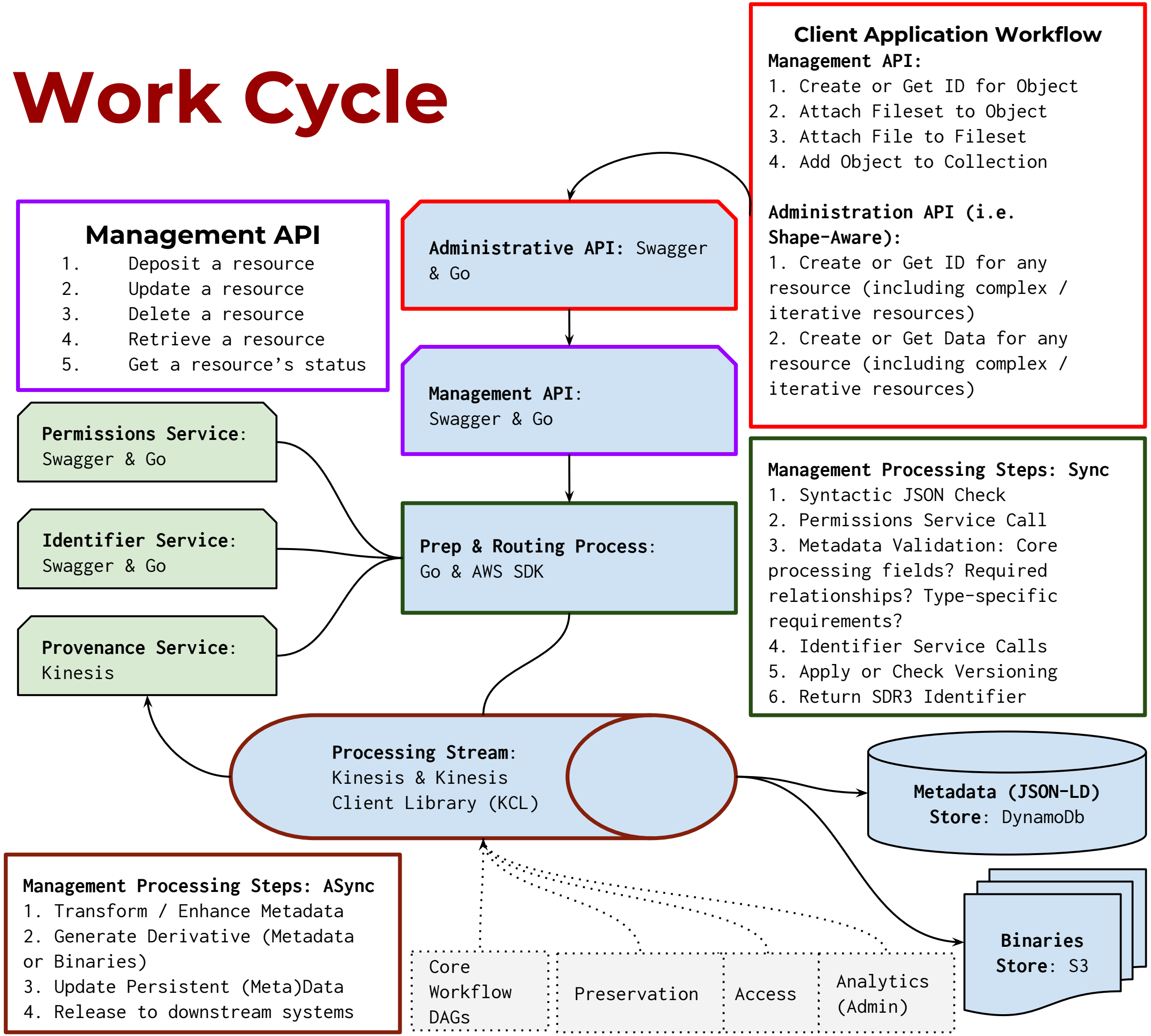
TACO Prototype’s Work Cycle 1 Goals

Functional Goals	**Technological Goals**	**Process Goals**
Deposit resources (binaries & metadata) into repository via API.	Drive forward Department API specifications, implementations, & practices.	Work towards new core with something visible to limited stakeholders to make it real-er.
Retrieve deposited resources from repository via API.	Test implementation options for our current SDR3 design.	Get feedback on SDR3 design, & check for roadblocks.
Persist resources.	Vet our data models & metadata profiles.	Keep to high-level, extensible functional blocks.
Perform skeletal resource processing (i.e. workflows).	Test feasibility of possible technologies: ● Hyrax integration points. ● Test throughput / scalability. ● SDR2 & SDR3 analytics. ● Inform cloud practices. ● Cloud first but Cloud neutral.	Showcase internal / lower stack rewrites needed for moving middle and end-user codebases forward.

TACO Prototype Work Cycle

TACO, or our *SDR3 Management API & Persistence Skeleton*:

- Foundational & extensible work for evolution of SDR2.
- Modular basis for new & existing components.
- **Addresses our core problematic technology, i.e. Fedora 3.**
- Serves **user requirements for flexible, consistent ingest & data models.**



Client Application Workflow

Management API:

1. Create or Get ID for Object
2. Attach Fileset to Object
3. Attach File to Fileset
4. Add Object to Collection

Administration API (i.e. Shape-Aware):

1. Create or Get ID for any resource (including complex / iterative resources)
2. Create or Get Data for any resource (including complex / iterative resources)

Management API

1. Deposit a resource
2. Update a resource
3. Delete a resource
4. Retrieve a resource
5. Get a resource's status

Administrative API: Swagger & Go

Management API: Swagger & Go

Permissions Service: Swagger & Go

Identifier Service: Swagger & Go

Provenance Service: Kinesis

Prep & Routing Process: Go & AWS SDK

Management Processing Steps: Sync

1. Syntactic JSON Check
2. Permissions Service Call
3. Metadata Validation: Core processing fields? Required relationships? Type-specific requirements?
4. Identifier Service Calls
5. Apply or Check Versioning
6. Return SDR3 Identifier

Processing Stream: Kinesis & Kinesis Client Library (KCL)

Metadata (JSON-LD) Store: DynamoDb

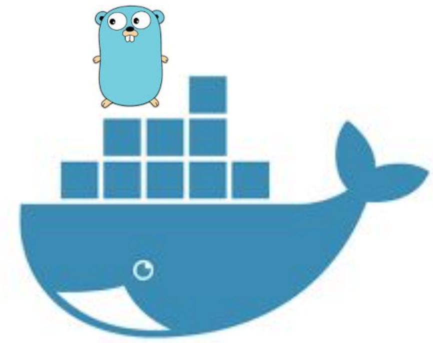
Binaries Store: S3

Management Processing Steps: ASync

1. Transform / Enhance Metadata
2. Generate Derivative (Metadata or Binaries)
3. Update Persistent (Meta)Data
4. Release to downstream systems

Go & Docker for TACO Codebase

- Ability to be modular, with APIs as clean boundaries & work in Cloud (AWS).
- Decision to use compiled language coupled Docker for deployment.
- Efficient Docker container deployment with small, executable binaries (as opposed to platforms that require an operating system and server).
- Focusing on compilable language for small, efficient services led us to Go language.



See the TACO Prototype GitHub Repository Wiki for all docs + more .

Go & Swagger Prototype Codebase

Additional TACO Prototype goals included:

- rapid development and delivery;
- SWAGGER specification support for consistent API to Code translations & share-ability of APIs across languages;
- support for continuous deployment & cloud solutions;
- parallelization fit for horizontal scalability.

See the TACO Prototype GitHub Repository Wiki for all docs + more .

AWS Selections for TACO

- Docker containers for sending off the codebase binary.
- AWS ECS (elastic container service) for running this image.
- CircleCI for Continuous Integration with AWS ECS & Docker due to its use by industry for similar set-ups.
- Terraform for building out AWS infrastructure
- AWS DynamoDb for metadata persistence **for the prototype**.
  - Very likely to use RDS in production.
- AWS S3 for binaries **for the prototype**.

See the TACO Prototype GitHub Repository Wiki for all docs + more .

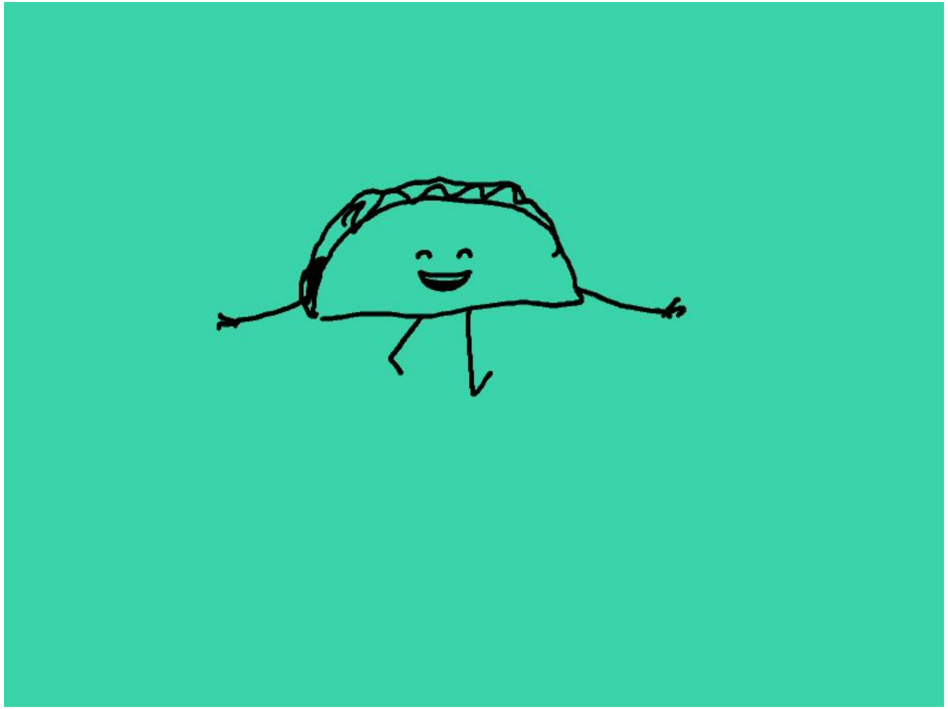
Cloud-first but Cloud-neutral

Our considered & kept-in-mind ***graceful degradation paths***:

- Docker => Docker is reusable.
- AWS ECS => Any system or VM that can run Docker. Docker swarm?
- Swagger 2.0 => Specification Built for Translateability
- Go + go-swagger => Just use Ruby.
- AWS dynamodb => CouchDB or Postgres.
- AWS s3 => File system.
- AWS kinesis => Kafka or Spark Streaming when ready.

Kafka / Kinesis?

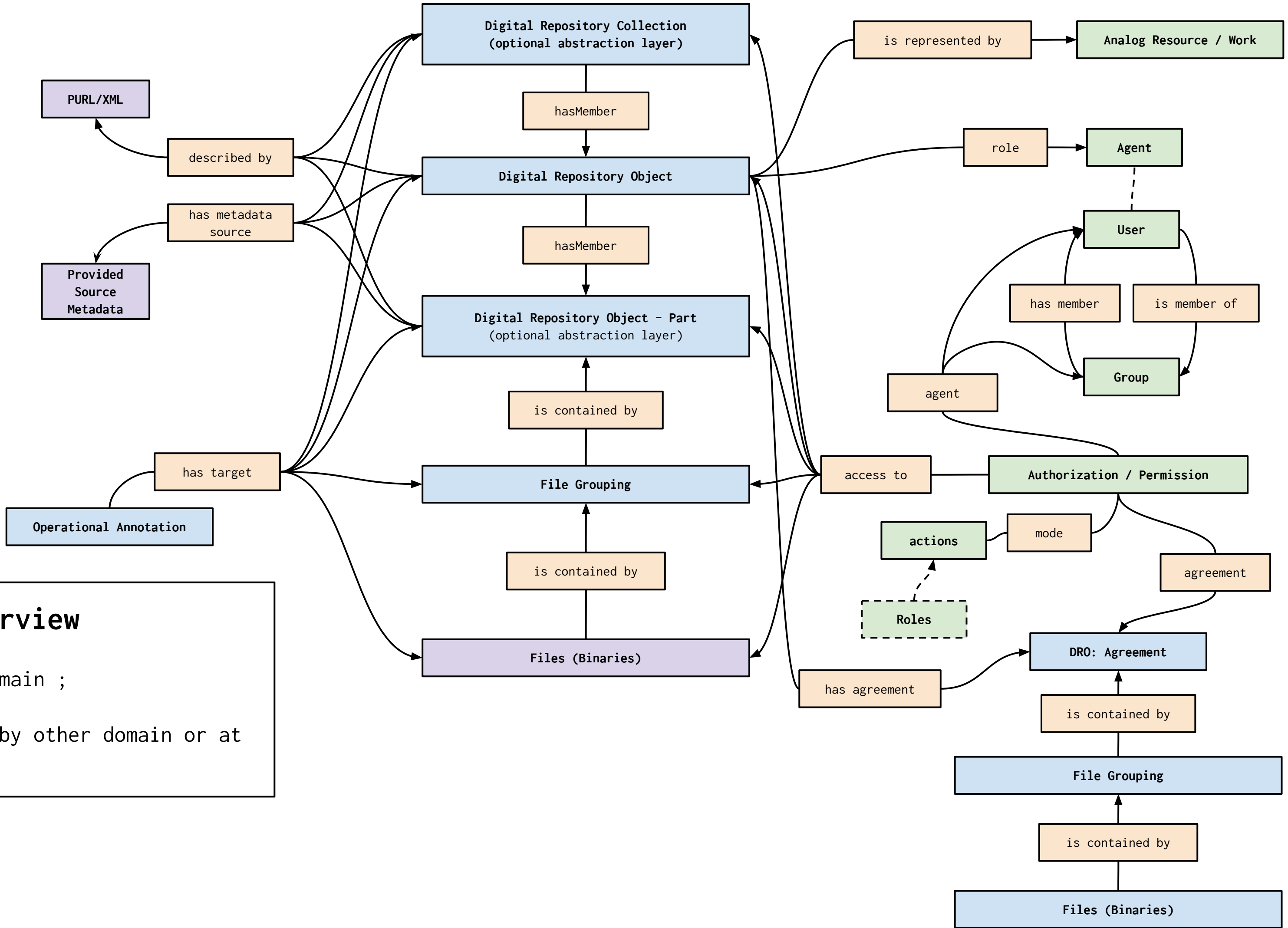
- Early design had event driven system for managing resource state & asych, DAG-based processing
- Put too much intelligence into TACO
- Kinesis deemed not suitable
- Re-designing to use Kafka-inspired event system within our Provenance & State Service
- Our asychronous, DAG-driven processing inspired by Airflow becomes parallel to SDR3



Special Note: Fedora 4 API vs TACO API

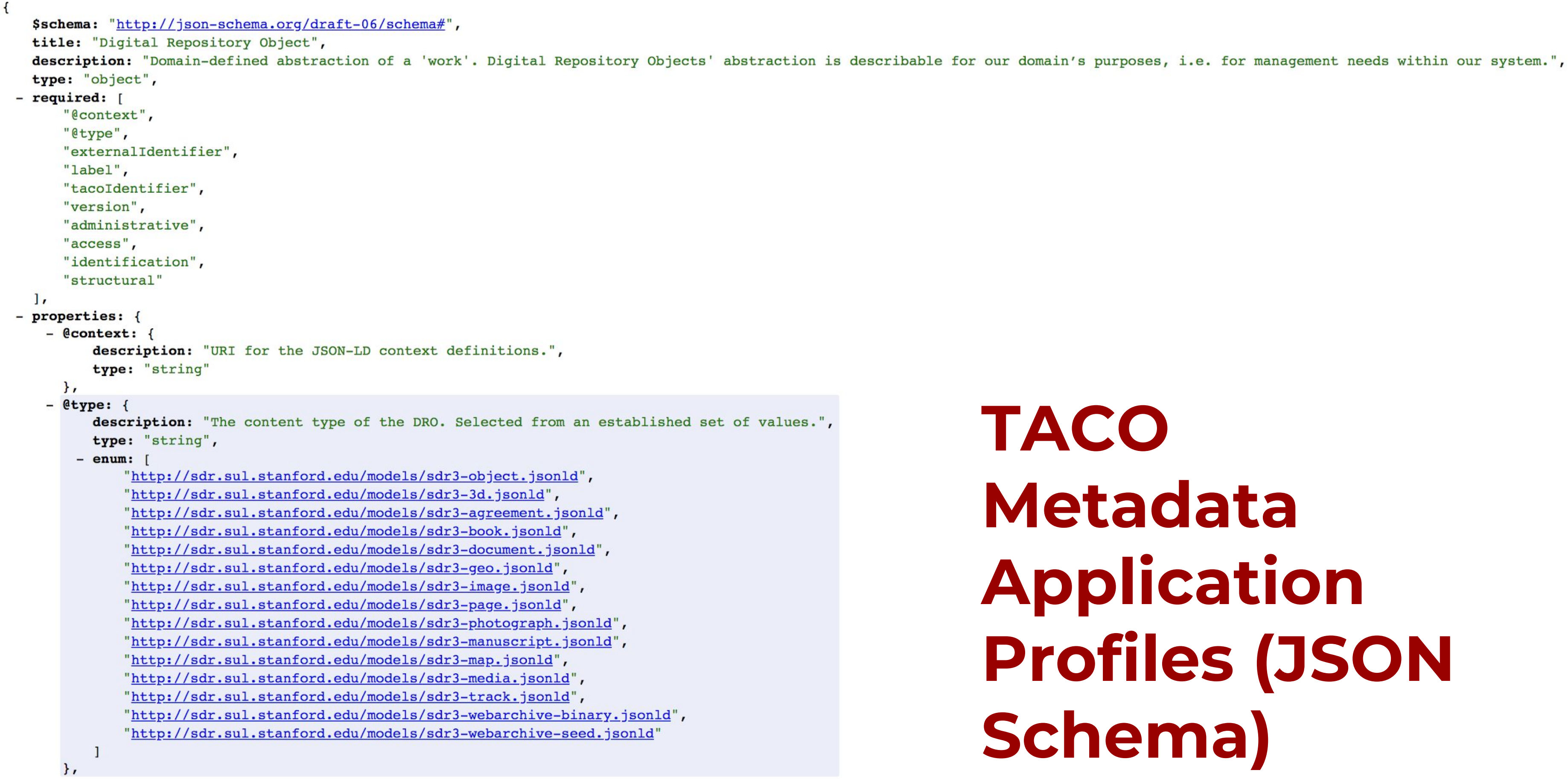
- TACO API aims to be much simpler than Fedora API.
- Decoupled from Linked Data Platform at this level of our stack.
  - We are supporting JSON / JSON-LD, which allows LD higher up.
- Reduced API calls, leading to increased performance.
  - Up to %50 less if we include ACLs, FileSets & ORE proxy ordering.

TACO Data Models



Model Structural Overview

Blue == Managed by Domain ;
Purple == File managed by Domain ;
Orange == relationship ;
Green == Externally managed by other domain or at application level



```
{
    $schema: "http://json-schema.org/draft-06/schema#",
    title: "Digital Repository Object",
    description: "Domain-defined abstraction of a 'work'. Digital Repository Objects' abstraction is describable for our domain’s purposes, i.e. for management needs within our system.",
    type: "object",
  - required: [
        "@context",
        "@type",
        "externalIdentifier",
        "label",
        "tacoIdentifier",
        "version",
        "administrative",
        "access",
        "identification",
        "structural"
    ],
  - properties: {
      - @context: {
            description: "URI for the JSON-LD context definitions.",
            type: "string"
        },
      - @type: {
            description: "The content type of the DRO. Selected from an established set of values.",
            type: "string",
          - enum: [
                "http://sdr.sul.stanford.edu/models/sdr3-object.jsonld",
                "http://sdr.sul.stanford.edu/models/sdr3-3d.jsonld",
                "http://sdr.sul.stanford.edu/models/sdr3-agreement.jsonld",
                "http://sdr.sul.stanford.edu/models/sdr3-book.jsonld",
                "http://sdr.sul.stanford.edu/models/sdr3-document.jsonld",
                "http://sdr.sul.stanford.edu/models/sdr3-geo.jsonld",
                "http://sdr.sul.stanford.edu/models/sdr3-image.jsonld",
                "http://sdr.sul.stanford.edu/models/sdr3-page.jsonld",
                "http://sdr.sul.stanford.edu/models/sdr3-photograph.jsonld",
                "http://sdr.sul.stanford.edu/models/sdr3-manuscript.jsonld",
                "http://sdr.sul.stanford.edu/models/sdr3-map.jsonld",
                "http://sdr.sul.stanford.edu/models/sdr3-media.jsonld",
                "http://sdr.sul.stanford.edu/models/sdr3-track.jsonld",
                "http://sdr.sul.stanford.edu/models/sdr3-webarchive-binary.jsonld",
                "http://sdr.sul.stanford.edu/models/sdr3-webarchive-seed.jsonld"
            ]
        },
```

TACO Metadata Application Profiles (JSON Schema)

See the SDR3 Metadata Models for MAPS, docs + more .

4. What’s Next?

Current Design Work

- File system analysis for options & costs
- Asynchronous & batch processing system design work going on
  - Heavily influenced by Apache Airflow
- Metadata efforts have free range approach
  - Starting with a metadata use cases analysis before jumping into schemas / ontologies
  - JSON[-LD] & JSON Schema used for flexibility, separation of external semantics & internal data shapes
- Preparing for next work cycle to revise & connect TACO ultimately to a self-deposit system & a bulk load job

Keeping Community Connections

- Samvera architecture & front-end work re-approach
- Interest in architectural overlaps with FOLIO
- Code4Lib Spark in the Dark overlaps
- Using PCDM, MOAB => OCFL, revisiting other places to share our data specifications
- Blacklight, IIIF, & related Access systems community work untouched
- Looking outside of cultural heritage for community partners & ideas
    - Airflow
    - AWS
- Asking our community friends & experts like ELAG participants for feedback

Questions or Feedback?

cmharlow@stanford.edu
@cm_harlow

https://github.com/sul-dlss-labs/taco/

STANFORD
UNIVERSITY
LIBRARIES