NBP – Customer Data

External Data Distribution Platform

Distribution, Refresh & Reconciliation

Pieter Dubois

ING Belgium

November 2025 – V1.3

Reviewed

do your thing

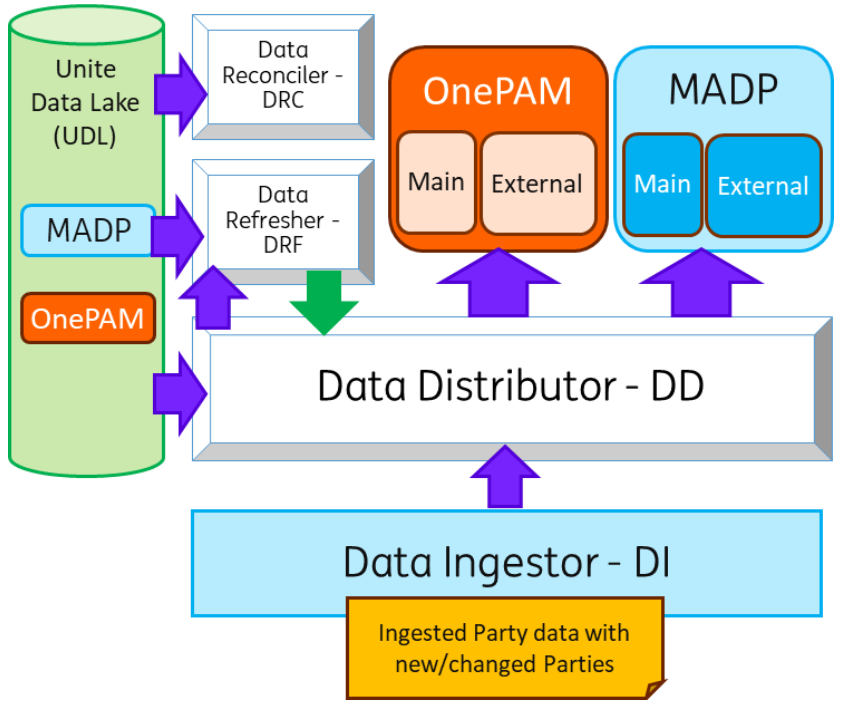
Versioning

Version	date	Remark
pre-V0.5	April/May 2025	Different iterations in developing the architecture + feedback Michel Applaincourt
V0.5	June 2025	Version presented at ARM (25/06/2025)
V0.6	June 2025	ARM Questions & responses added in backup
V0.7	July 2025	Complete rework after new requirements Stijn Carmen and invalidation some original assumptions + introducing Data Reconciler
V0.8	August 2025	Rework by adding APIs as an integration mechanism and API vs OIL evaluation (feedback Livia Jercan PO OIL) & further details for architecture (backup)
V0.9	September 2025	Published version of V0.8 for feedback Michel Applaincourt
V1.0	October 2025	New version after agreement parallelization Stijn Carmen + additional input OIL & APIs +introduction Refresher (feedback Michel Applaincourt)
V1.1	October 2025	New version proposing a phased approach after input Christiaan Douma on APIs & René Vermaat & Alain Heremans feedback + Thomas Durieux feedback no MADP attributes currently.
V1.2	November 2025	Published version of V1.1 for review Michel Applaincourt.
V1.3	November 2025	Reviewed version of V1.2 with remarks Michel Applaincourt incorporated.

Summary of this presentation

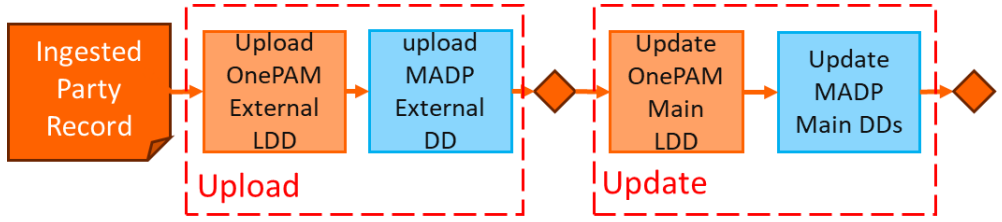
The **Ingest & Distribution Platform (IDP)** has 2 major components & 2 supporting components:

- **Data Ingestor (DI)** takes (external) ingested Party data, reformatting & enriching it into an ING format. Its Implementation is done by the **Data Management tribe**.
- **Data Distributor (DD)** makes changed & new "Core" **Party** data available to ING Groups' Party Data platform, **OnePAM**, & associated *"Local"* **Party** *attribute data* to **MADP**. Its implementation is done by **Customer Data tribe.** It is supported by:
  - The **Data Refresher (DRF)*** transforms a full data set of external ingested Party data into a smaller data set that only contains changes with the **OnePAM & MADP External (Landing) zones** for upload in those zones.
  - The **Data Reconciler (DRC)*** compares data for which the External (Landing) Zones of **OnePAM & MADP** are master with data in the **Main Zones of OnePAM & MADP** and creates extracts that only contains differences between them for updating the Main zones.



We focus on the **Data Distributor** which manages 2 distinct phases:

1. **Upload:** the ING normalized ingested **Party** data into the **OnePAM & MADP External zones**, consumable by **Customer Onboarding journeys**.
2. **Update:** extract **Involved Party** data updates from ingested **Party** data that is uploaded in the **OnePAM & MADP External zones** and update different attributes into the **OnePAM &MADP Main zones.**



* DRF and DRC use extracts from copies of OnePAM and MADP in the Unite Data Lake to support their activities, rather than the operational OnePAM and MADP systems themselves

Table of Content

Part 1: High-Level Architecture

Table of Content

Context - Key business requirements*

- Import Party-related data files in different formats from different (external) providers (**Coface & Graydon**) into the Party Data Platform (**OnePAM & MADP**).
- Types of data files:
  - **Daily Delta files**: vendor-indicated changed Party data
  - **(Monthly) refreshes**: vendor provided full Party data sets
  - **Initial loads**: initial full data set to populate the Party Platform Data
  - **Reconciliations**: differences between Party Data platform External and Main zones.
- In the target format:
  - **Converted**: mappings, transformations & quality mgd.
  - **Model transformed**: Split composite Party records into individual Parties and Relationships
- **Upload changes in Party records** in **"External" data stores** of the Party Data Platform (Landing zones):
  - **OnePAM**: Parties, relationships & core attributes
  - **MADP**: "Local" attributes
- **Update Involved Party records** in "**Main" data stores** with fine grained attribute comparisons (policy driven)
  - "*Self-healing*": update *different* attributes between external ingested Party data & Involved Party data in the Main zone if the external data source attribute is master.
  - **Block** propagation of some changed data to the Main Zone.
- All uploads to be finished before the start of the next business day after receiving them.
- Ability to provide fine-grained configuration by Data Stewards

**to be finalized with documented work of Ellen Delagrange and Thomas Durieux (for example: Coface Legal Entities).*

Context - What data are we talking about?

ING BE ingests *curated* Party data from **external suppliers** (*Coface*, *Greydon*) on:

- Belgian **Legal Entities (LE)** (*KBO,* Operating Sites, Legal Representatives)
- **Private Individuals** (court appointed guardians, under administrations, private bankruptcies, …).

This external ingested Party data contains both:

- ING BE ***Involved Parties:*** customers, associated parties like legal representatives of customer LE's, administrators of incapables that are customers…
- ***Non-Involved Parties:*** *have* no direct relation with ING, like LE's and legal representatives that are not customers

The target **Party Data Platform, OnePAM**, manages in principle only **Involved Parties,** aka **"Core" data** in a **Main zone**.

Non-Involved Party data* is important for ING BE during **onboarding** of a new customer (eg. validate the provided data for LEs), or to block them (people under administration, upon bankruptcy) from becoming customer.

OnePAM provided ING BE with a segregated **"External" (Landing) zone** to store external ingested data that can include **non-Involved Party Data** that conforms to the **"Core" OnePAM model.**

OnePAM doesn't manage Party-associated **"Local" attribute data*** relevant to ING BE only, the latter of which will be managed by the **Managed Auxiliary Data Platform (MADP)**.

Our ingested Party data can contain all of these. We must separate all these data types out of the ingested Party records and store them correctly in the right **OnePAM** and **MADP** data store

** Formerly known as « Party Orphans » & « Data Orphans » respectively*

Context - Input files: types of data, formats

There are **4 main ingested sources**, each in different formats:

- **Coface Legal Entities** (incl Operating Sites) -XML –**5-13k daily**
- **Coface Incapables** –XML –**150 daily**
- **Graydon Legal Representatives** –Excel - **??? daily***
- **Graydon Bankruptcies** –CSV - **< 100 daily**.

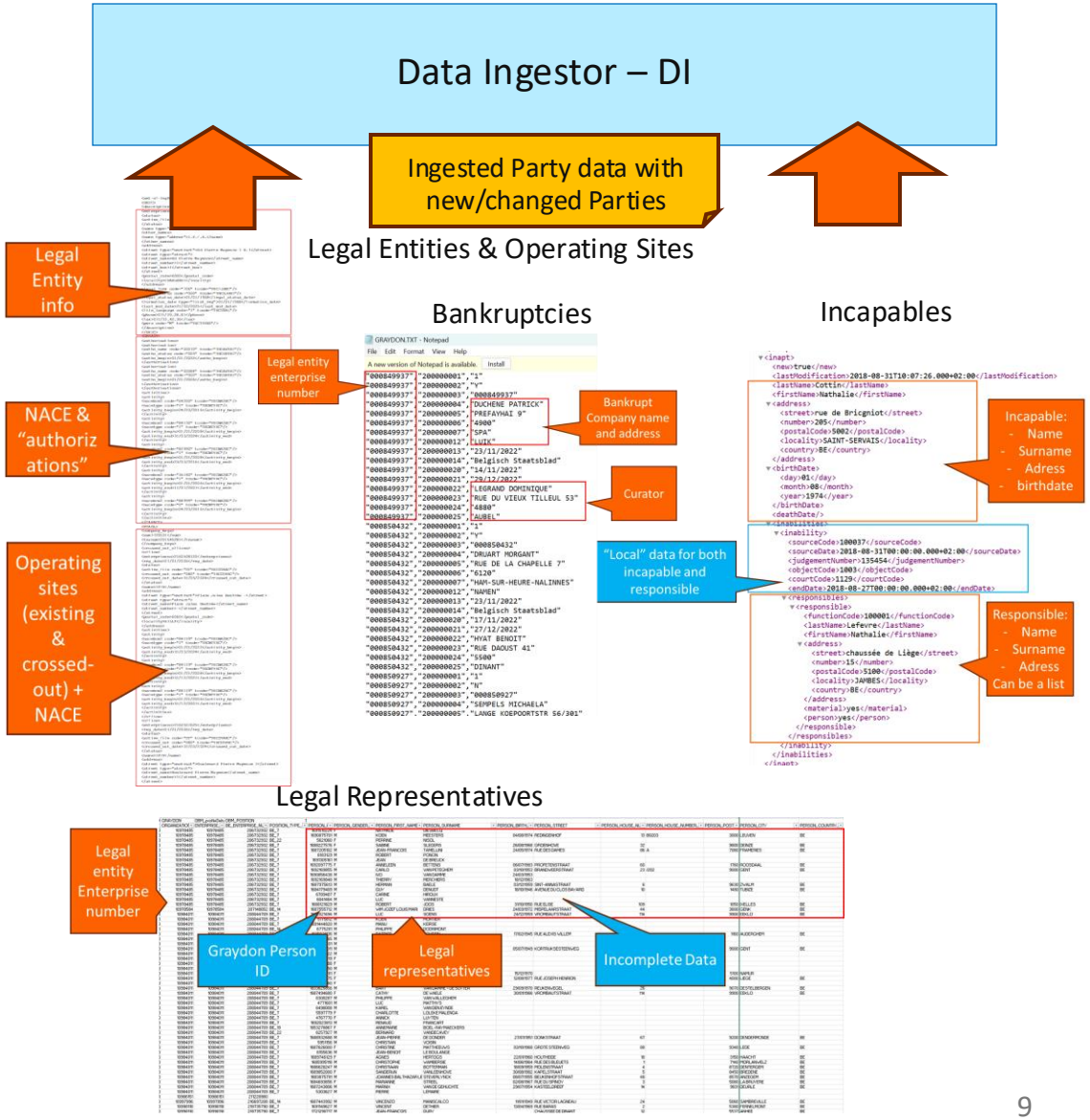
A full refresh is ~3M records

Each of these files have different **data quality**:

- **High**: Coface Legal Entities (authoritative source: Moniteur Belge/Staatsblad)
- **Low**: Graydon Legal Representatives (Graydon own curation without a single legal source, and mixed data quality like missing addresses of legal representatives)

Some of the attributes in these files fits in the **OnePAM** "Core" model, other data may be considered "Local" and must be stored in **MADP**.

For data for which the external source is authoritative (fi. Legal it is considered "**master data**" and we normally will overwrite data in the OnePAM Main zone for Involved Parties if it is different.

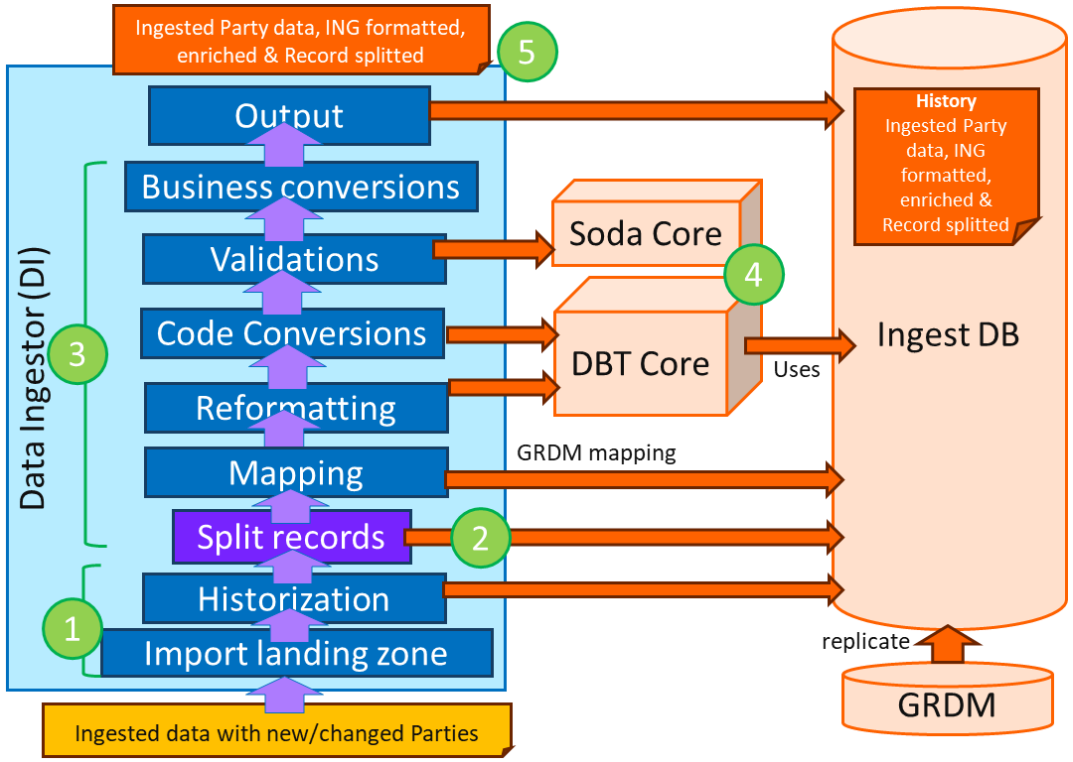


**Not automated today, comes in as excel file.*

Environment – Associated Component - The Data Ingestor (DI) as source

The Data Ingestor platform is architected by Els Van Opstal (here). Here we provide input to understand Data Distributor starting point

1. **The "Data Ingestor" (DI)** ingests data in the form of **changed and new** Party data records*. These records are in different formats which are defined by the information supplier.
   - The **DI** archives the original content for data lineage, support & to ensure only actually changed records are processed (efficiency).
2. Composite Party records with multiple parties & implicit relations will be **dissagregated** into individual Party records.
3. The **DI** *maps, reformats, converts, validates & adds business conversions* (not business rules) to the data entities.
4. The **DI** relies on **ING Group platform tooling** and repositories for its functionalities (glued with python, whereas the **Data Distributor** will be custom development (probably Java).
   - Difference in technologies (and required team skill sets) is a reason to segregate DD & DI both architecturally as organizationally.
   - The **Data Management tribe** will implement the **Data Ingestor**.
5. The "**ING formatted**" **DI** output is input for the **Data Distributor.** The full history of processed records is available in the ingest DB

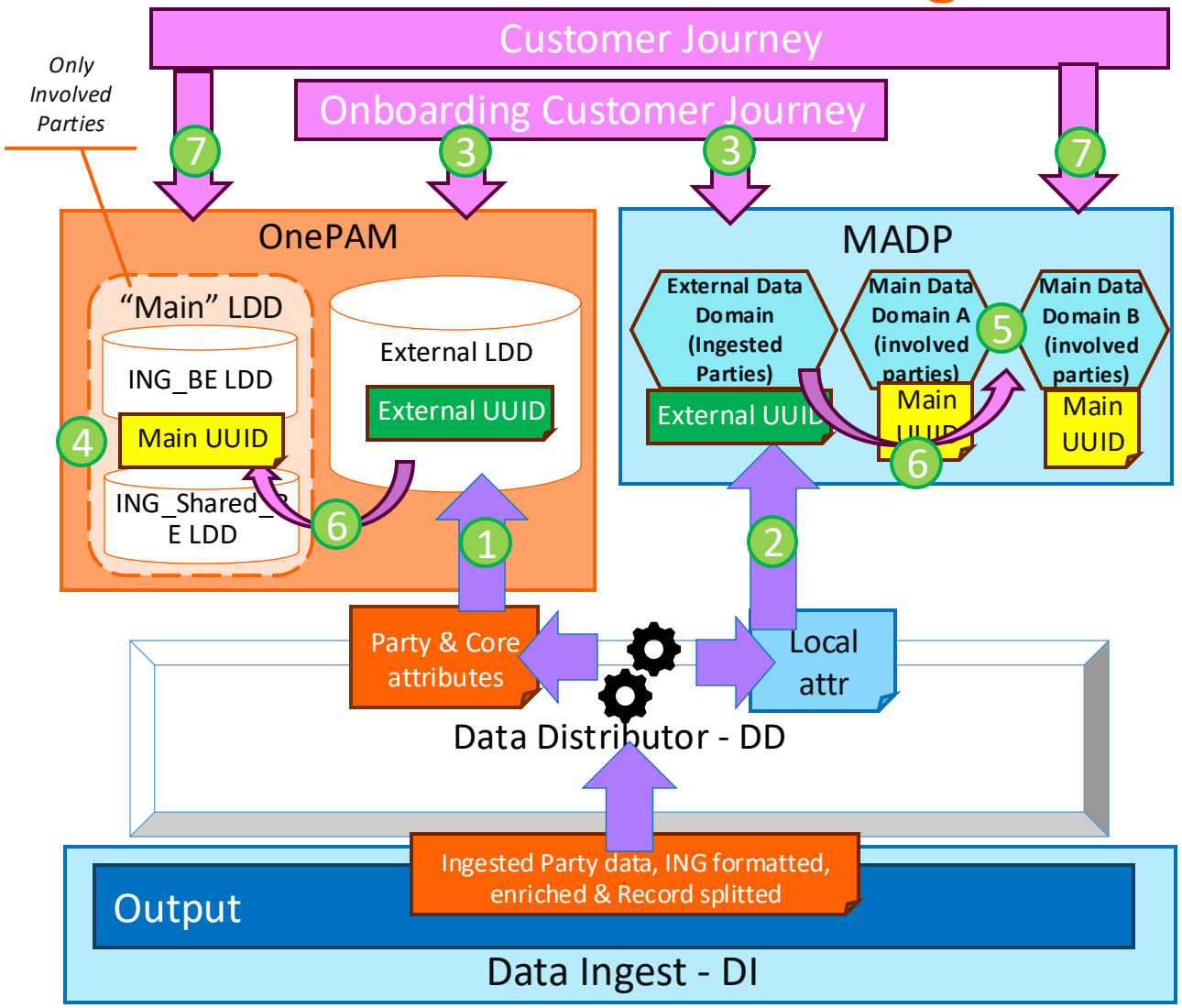


The DI makes no connections to Customer Data Platforms like OnePAM/MADP because of separation-of-concerns & genericity

** Even when only a single attribute of a Party record changed, the whole Party record is provided but we don't know which one.*

Environment – Associated component OnePAM & MADP as targets

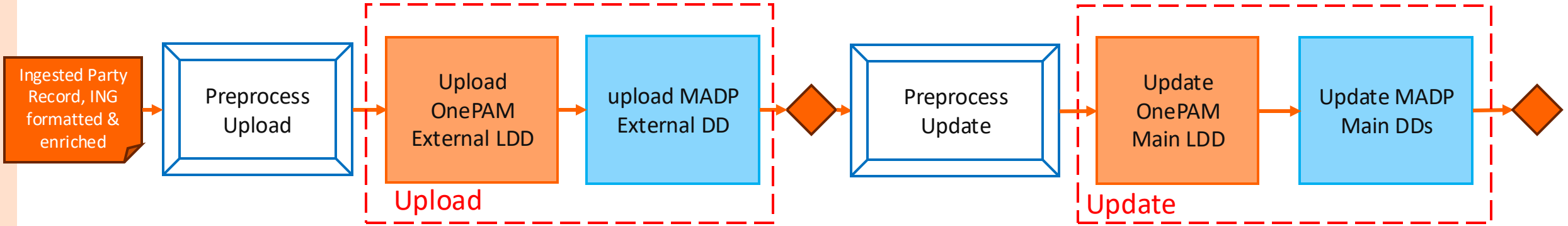
1. We store **ingested Party*** data that fits the Core **OnePAM** model, in a *read-only* **OnePAM External LDD**.**.
   - Each **Party** is identified by a **OnePAM** "External" UUID.
2. We store "Local" attributes of **Parties** (with an "External" UUID) in a dedicated **MADP External Data Domain.**
3. The **External LDD** & **External Data Domain** will be used by **Onboarding Customer Journeys** as pre-check data. It is considered a **Landing zone** to derive Main data from
4. **OnePAM** stores **Involved Parties (IP)** in 2 "**Logical Data Domains" (LDDs)** collectively called **"Main" LDD**; one for Private Individuals, one for Legal Entities.
   - An **Involved Party** is identified by a OnePAM "Main" UUID. (technically there is no distinction with "External" UUIDs).
5. **MADP** stores **functionally-grouped** "Local" attributes of **IPs** (having a "Main" UUID) in "**Data Domains**" **(DDs).**
6. The **Party** data in the OnePAM **External LDD & MADP External DD** is used to update **Involved Parties** in the **OnePAM "Main" LDD** and **MADP "Main" Data Domains**.
7. All non-onboarding Customer Journeys using **Involved Party** data will access the **"Main" LDDs** & **Data Domains**.



** Ingested Parties contain both Involved Parties (where ING has a relationship, like customers etc), as non-involved Parties (where ING has no relationship).*

*** Agreed between ING BE Customer Data and GCDM (who manages OnePAM "Core" data model). This is an architectural decision to be documented.*

Context - 2 distinct phases: Upload and Update



During the **Upload phase** we upload ingested **Party data** into **OnePAM External LDD** for the "Core" attributes and into **MADP External Data Domain** for "Local" attributes.

The **External zones** are considered "**Landing zones**", **master** for ingested **Party** data. Therefore, we must have certainty on what data was successfully uploaded. This must happen before we update other stores from the same data, like the **Main zones**.

In principle **happens the upload in MADP after the upload in OnePAM** because "Local" attributes must have a **OnePAM UUID** attached, and we only know that after we create Parties in OnePAM.

During the **Update phase** we update ingested **Involved Party data** into **OnePAM Main LDD** for the "Core" attributes and into **MADP Main Data Domains** for "Local" attributes.

We only update data in the **Main zones** that was successfully uploaded first into the **External zones**, and we will only update **Involved Party data**.

We not only update ingested attributes that changed but may also overwrite existing unchanged ingested attributes in the Main zones which were changed by other process if we are master for them.

Refreshes and updates are participating as preprocessing or input steps in the Upload & Update process respectively

Table of Content

Constraint 1: number of records to process

We start with an Initial load of each file in the production databases, which can have millions of records and can take multiple days to finish.

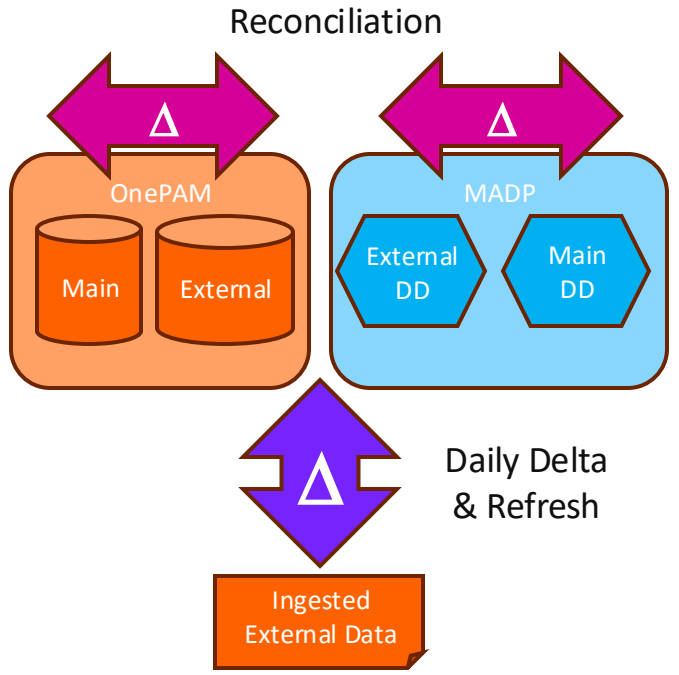
Then we must process **several files with changes per day**, every day.

Some are smaller (10's to 100's of records) some larger (1000's to low 10000's of records)

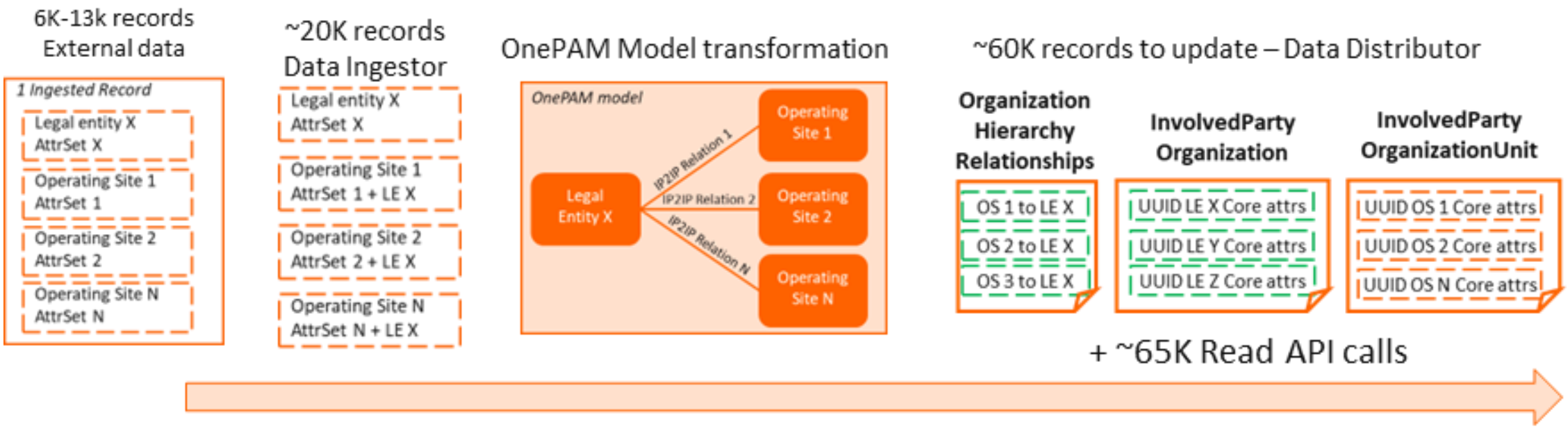
But on a periodic basis (monthly) we must do **full refreshes and reconciliations** (millions of records).

Refreshes are external ingested files coming from the external supplier, but contain the complete data set, rather than the daily delta's. They are like initial loads that you redo.

Reconciliations are, comparisons between the External and Main zones, where the information in the External zones are considered "master" for some attributes. It introduces no new data.



Constraint 2: ingested record multiplication

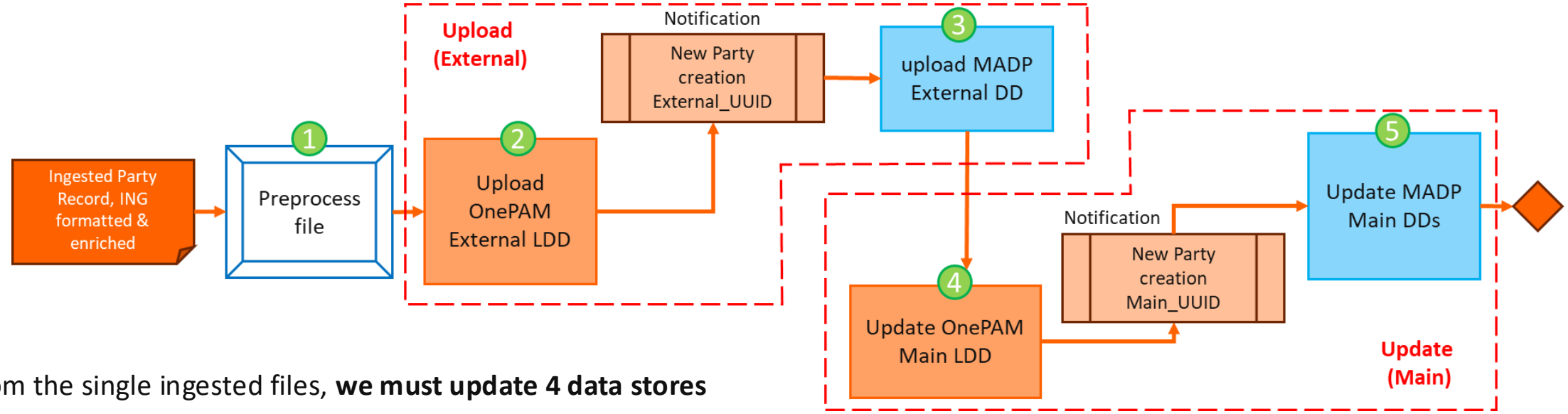


The **number of records processed multiplies** due to:

- Disaggregation by the **Data Ingestor** of *composite* Party records into *atomic* Party records.
- *Model transformation* into objects & relationships
- Upload formats (different OIL files for Parties and relationships, multiple API endpoints for different entities within a single **Involved Party** object (postal address, name, …)

Leading to **a large number of overall transactions** to execute in the end repository leading to longer throughput times.

Constraint 3: multiple repositories that are updated sequentially



From the single ingested files, **we must update 4 data stores** which are **sequentially dependent**.

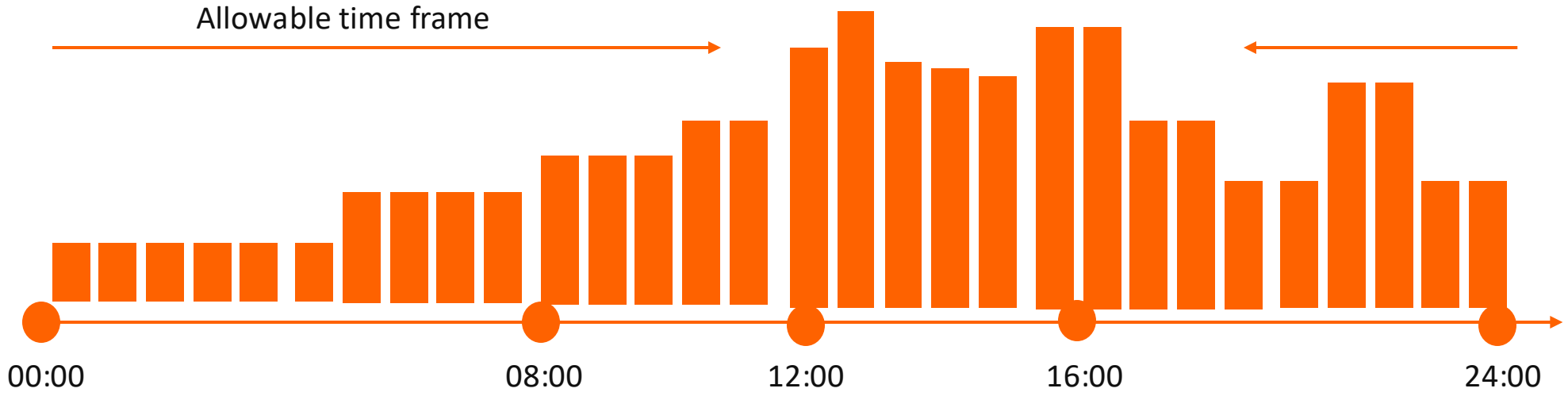
Updates of the **Main** data zones may only happen after updates in the **External** zones happened as the External zones are considered "*Master*" for external ingested data.

MADP objects are dependent on their creation in **OnePAM** first, capturing the **OnePAM** UUID and then creating MADP objects, referencing this **OnePAM** UUID.

The sequential nature also requires use to obtain the *confirmation of uploads* before moving on to the next zone, as well as to ensure only successful uploaded records/ attributes are propagated so no inconsistencies arise.

This increases the end-to-end throughput times of uploads

Constraint 4: Limited time windows



OnePAM has **time-of-day constraints** because, as an interactive production system, it has spikes during certain hours of the day. In the architecture of OnePAM there is currently no difference between batch and interactive traffic.

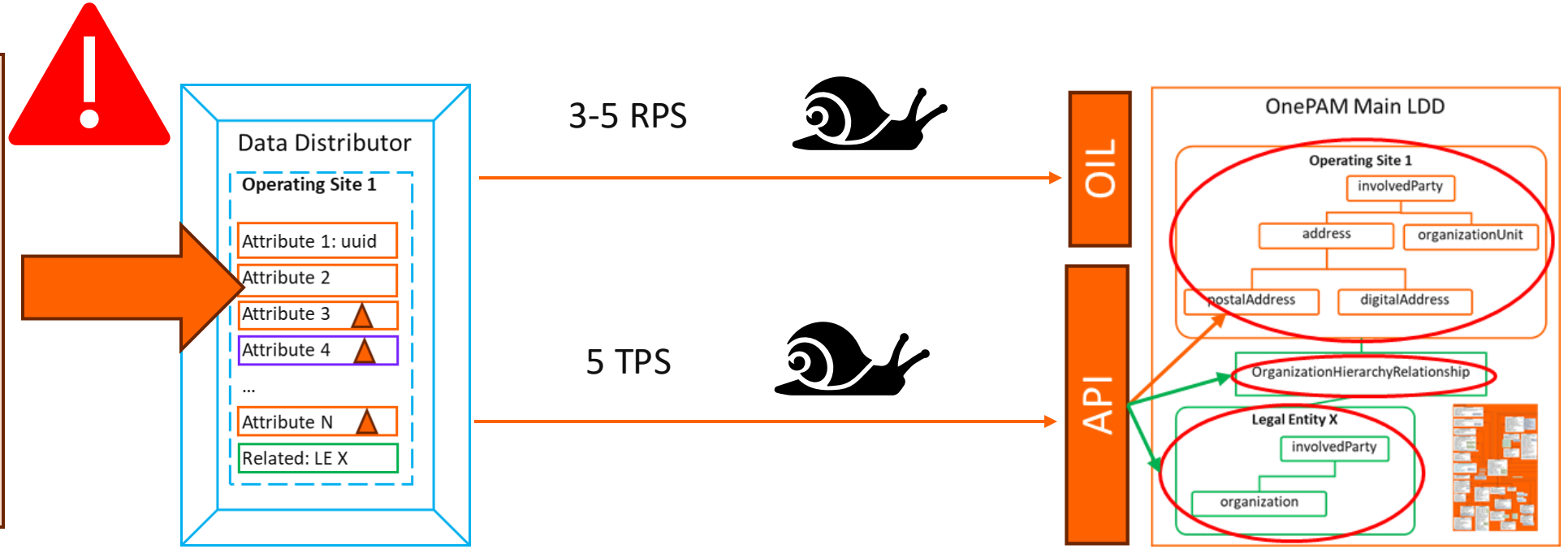
Our volume-based updates should **not interfere** with customers and employees using the system. Smaller uploads will not impact normal operations, but larger daily uploads should happen in the quiet hours.

Very large initial loads must be handled manually and planned by the OIL team of OnePAM and not be uploaded using automated means.

Should not be a constraint anymore in OnePAM NG

Constraint 5: Slow upload throughput

This is the single most determining throughput factor



OnePAM OIL & Fact Write APIs are **by default constraint** in # records per second (RPS) & transactions per second (TPS) respectively, putting **upper boundaries on what can be processed** within a given time period.

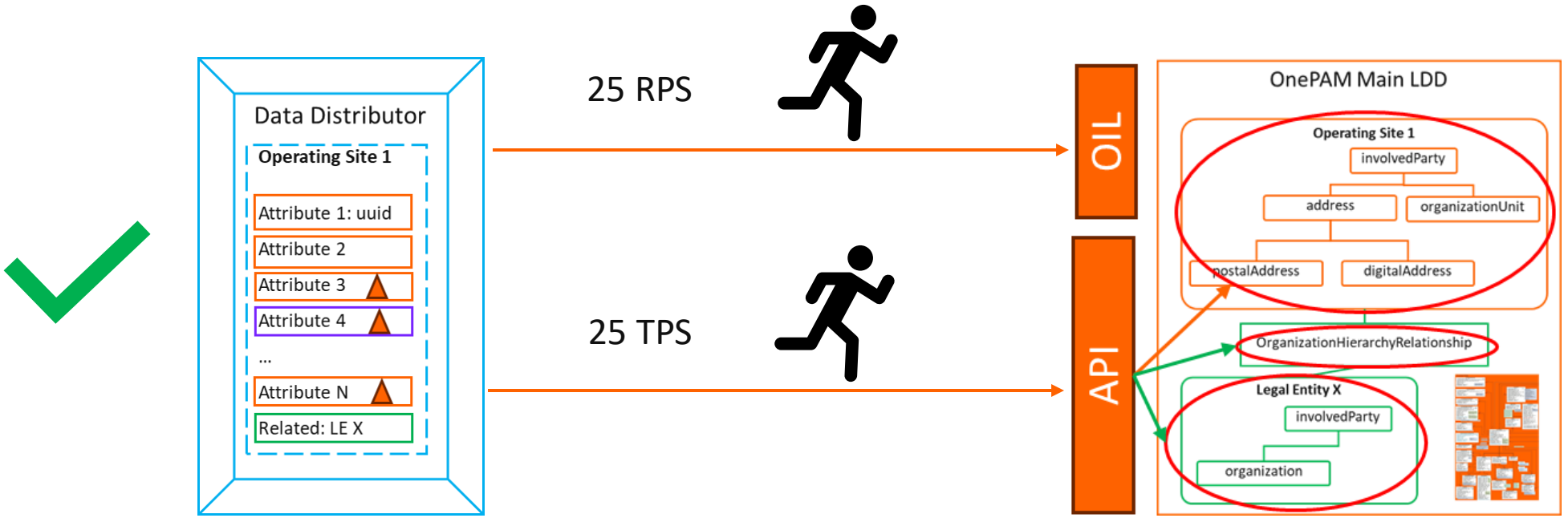
The cumulative effect of daily multiple file uploads, some large, record & call multiplication, serialized uploads to multiple data stores, limited time windows and slow upload throughputs create real issues to ensure updates can be made within the required time windows.

Should not be a constraint anymore in OnePAM NG

Table of Content

Strategy 1: Request higher throughput rates to OnePAM

5 Strategies to alleviate the constraints



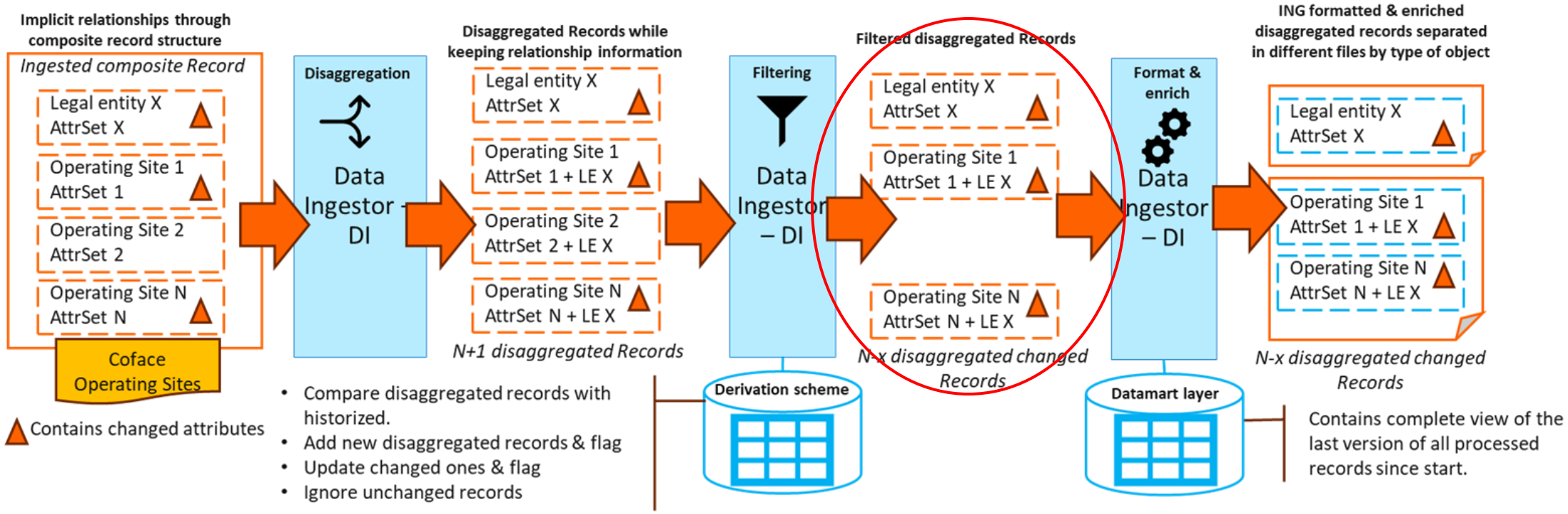
We request to use higher throughput rates (25 TPS & RPS) for APIs and OIL respectively.

This is in principle possible but **must be agreed with GCDM**.

The question has been asked but no answer yet, and this must be followed-up on.

Strategy 2: Filtering out unchanged Party records & attributes

5 Strategies to alleviate the constraints



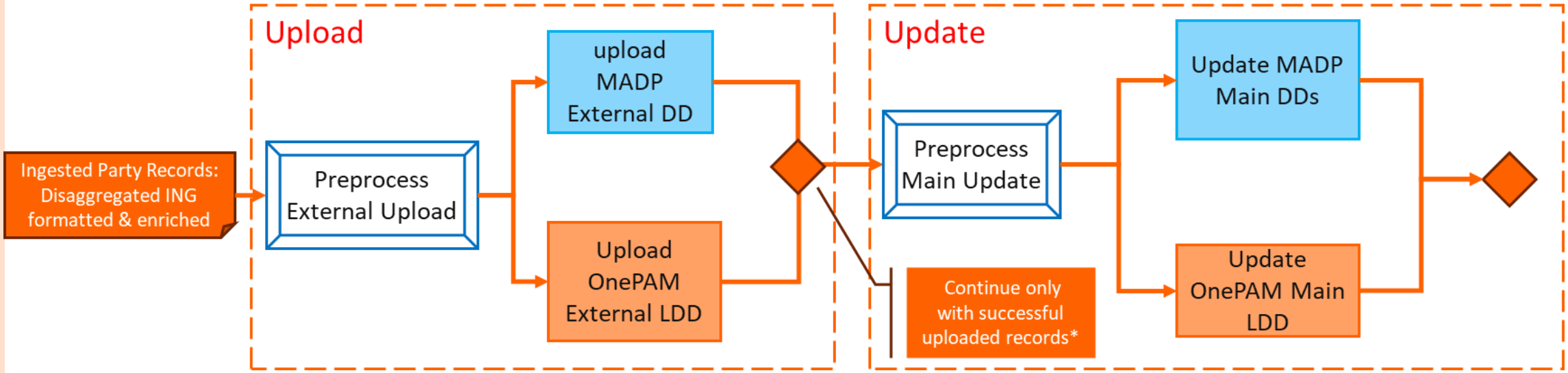
Data Ingestor prunes the set of ingested disaggregated, atomic Party records of all unchanged records, and we **process only disaggregated records with changes**.

This reduces the total number of records to process. However it will still be a multiple of the ingested number of composite records as all of them have at least one change, and most have relationships to other Party objects.

This is an architecture decision: we don't update or even 'touch' data that didn't change

Strategy 3: Flow parallelization

5 Strategies to alleviate the constraints



We **parallel update OnePAM and MADP** Landing & Main zones respectively to **reduce end-to-end throughput time**. But there are conditions to achieve this.

There is a constraint where **MADP requires a OnePAM UUID** for all "Local" attributes when uploading, also for new Parties. We do know what ingested records are new before uploading as we will read all current Party objects in OnePAM upfront through *entity matching** and a miss indicates a new Party that must be created.

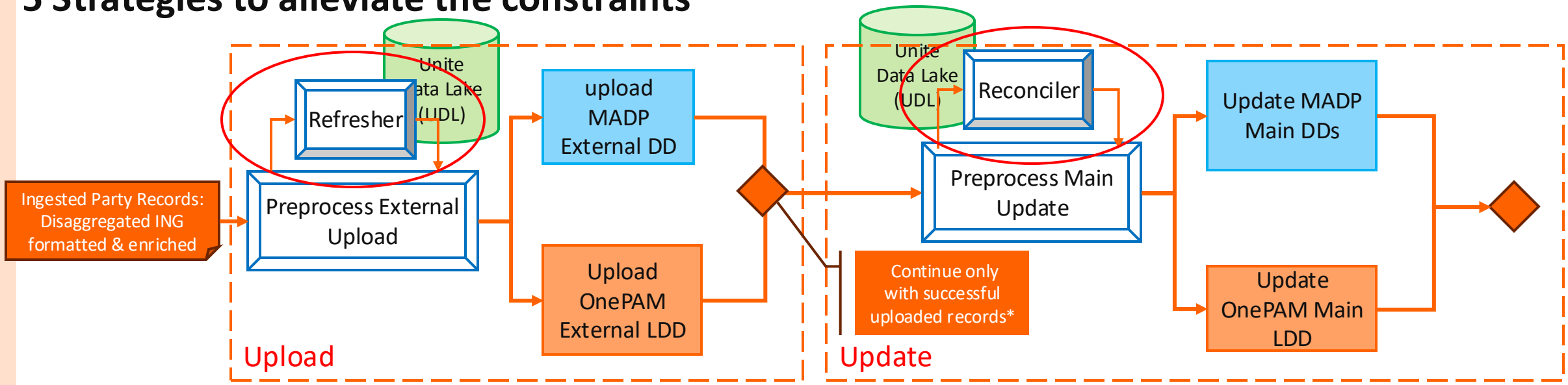
OnePAM has a feature** to **define new UUIDs outside OnePAM for new Parties before creating them inside OnePAM**. To enable this, **we must raise a feature request with GCDM**. *This will take some time & justification.*

If we can use this feature, we can upload data to OnePAM and MADP in parallel as we know all UUIDs upfront.

* Entity Matching is the using of information in the ingested atomic Party record (like Legal Entity number, or demographics) to unambiguously identify a Party object in OnePAM and obtain the OnePAM UUID.
** Only enabled for Austrialia and only for OIL and the Fact API endpoints Austrialia uses

Strategy 4: out-of-band processing for refreshes & reconciliations

5 Strategies to alleviate the constraints



Refreshes & **Reconciliations** require us to process ALL data in **OnePAM** and **MADP** which, given the constraints of OnePAM is impractical to execute directly on the operational system. Most activity, once the full dataset is extracted, is comparing either Ingested data with the External zones, or compare Main and External Zones, respectively. Only differences between them must be uploaded to the External zones, or Main zones respectively.

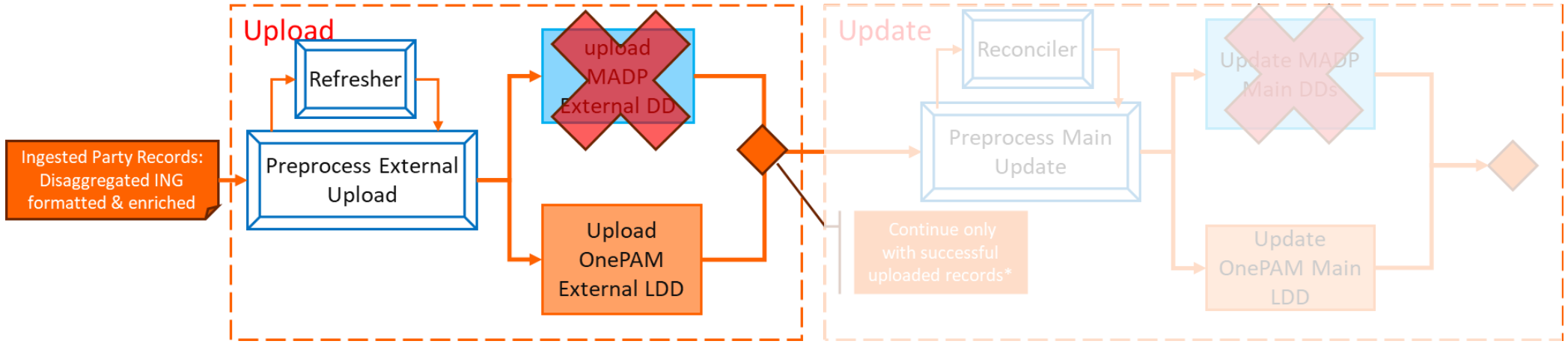
We will process the data out-of-band, by using the **Unite Data Lake (UDL)**, with only actual differences to be processed into **OnePAM/MADP**, turning it into a delta that is treated like a load done daily (albeit on steroids).

For this we introduce 2 new components (which, in implementation could be a single one):

- **The Refresher:** compares external ingested full datasets with what is in the UDL copied OnePAM/MADP External zones
- **The Reconciler:** compares the data between the in the UDL copied OnePAM/MADP External and Main zones.

Strategy 5: Phase-based implementation approach

5 Strategies to alleviate the constraints



Not all enablers to cope with the constraints have been guaranteed yet by GCDM for OnePAM, higher TPS/RPS & externally defined OnePAM UUIDs.

But **we don't need to have the full possible architecture available from the start** (Big Design Up Front).

- There have been **no "Local" attributes identified in the current ingested files** (Legal Entities, Operating Sites, Legal representatives, Bankruptcies & Incapables). That means **we don't need MADP** (for now), nor the functionality to create **OnePAM** UUIDs outside OnePAM.
- **We don't Update Involved Party data to the OnePAM Main LDD until later**, so for now, given current volumes, with only updating the OnePAM External LDD we don't necessarily need the 25 TPS/RPS for daily updates (for initial loads it would be nice but initial loads are not that time sensitive and can happen over multiple days).

This allows us to **create an evolutionary architecture** that provides the required functionality over time a.) when it is needed, and b.) after we obtained the buy-in of GCDM and subsequent enablement of the features.

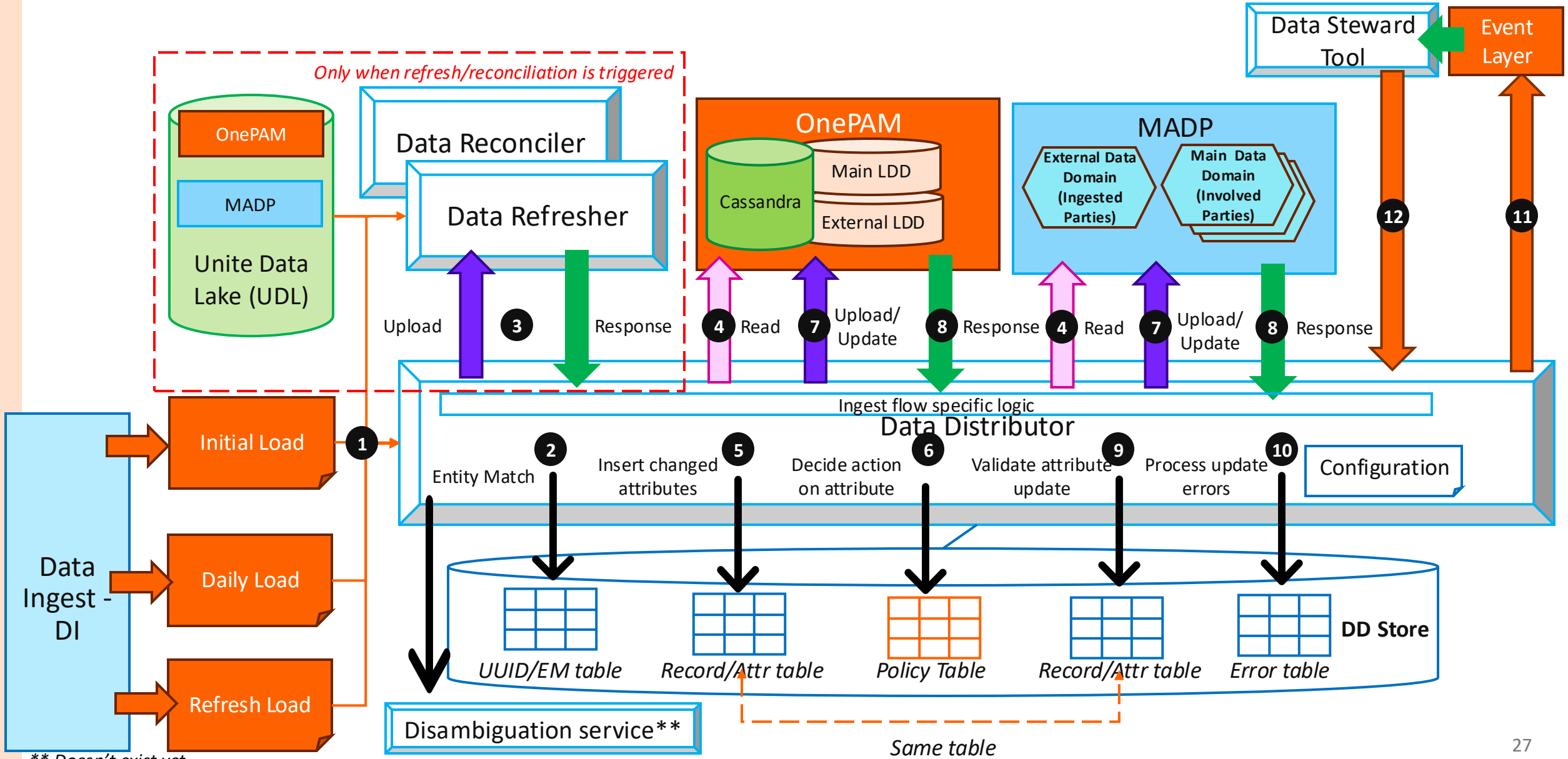
Table of Content

Data Distributor Architecture Solution Principles

1. The **Data Ingestor** converts external ingested Party data into a suitable format for the **Data Distributor** to consume.
   - It does not apply business logic to the ingested Party data.
2. The **Data Distributor** updates the **Party Data Platform** with external ingested data:
   - "Landing" zones (**OnePAM External LDD** & **MADP External Data Domain**) with **Party** data changes
   - "Main" zones (**OnePAM Main LDD** & **MADP Main Data Domains**) with **Involved Party** attributes.
3. Only data passing through **Data Ingestor** & **Data Distributor** can be stored in the landing zones.
   - The **Data Distributor** is the only component that can update the **Party Data Platform** landing zones*
   - The **Party Data Platform** landing zones are **Read-Only for everyone** but the **Data Distributor**.
   - The **Data Distributor** only updates the **Party Data Platform** landing zones with **Data Ingestor & Data Refresher** provided data.
   - The **Party Data Platform** landing zones are also **Read-Only** for the **Data Distributor** otherwise to ensure their integrity
4. The Data Distributor only acts on changed Ingested Party Data (deltas must exist).
   - All its input records must have **at least one changed attribute** for any data to be updated in the Landing or Main zones.
   - **Full refreshes** are first processed by the **Data Refresher** which provides the Data Distributor with the remaining deltas to process.
   - **Reconciliations** are first processed by the **Data Reconciler** which provides the Data Distributor with deltas for the Main zones only
5. The **Data Distributor** does not implement the business consequence of ingested Party data updates.
   - It updates ingested Involved Party data in the **Main OnePAM LDD** and **MADP Main Data Domains**.
   - It updates changed attributes or sets flags on matched Involved Party objects (e.g. *flag an Involved Party as an incapable*).
   - It creates & deletes **Involved Party** objects and associated relationships (e.g. *deleting removed Legal Reps that are not customers*).
   - It will not apply "business logic" on other attributes (e.g. *it will flag an incapable but will not take away mandates*).
   - Derived and downstream alterations must be done by "Business Event" services acting on changes by the **Data Distributor.**

*including data stewardship

High-level* End-to-end Architecture of the Data Distributor



** Doesn't exist yet.

*In the phased approach not all components are there and at a lower level of abstraction (File vs API, External vs Main) there are some variations

Generic flow of the Data Distributor

Preprocessing

1. Read input
2. Entity matching
   - Local (DD UUID)
   - Remote disambiguation
3. (if needed) Hand-off to Refresher or Reconciler for removing unchanged Parties

Preparing

4. Read existing values (API)
5. Identify changed attributes & store for post-processing validation
6. Evaluate update policy (for updating Main zones)

Uploading

7. Upload to target
   - API-calls
   - Files
   - Create/Change /Delete
   - Parties/ Relationships/ Attributes
8. Capture responses
   - API response
   - Events

Postprocessing

9. Validate responses
   - Check-off stored changed attributes
10. Process failures
    - Response errors (API)
    - Decide out-of-time (Events)

11. The Data Distributor may send out status events to an event layer to be picked up by management tools for human or automated action.
12. A human management interface (like the data Stewardship tool) should have access to the Data Distributor to view logged messages, change configuration, adapt policies and restart or retry to upload records that failed before.

Functional architecture of the Data Distributor

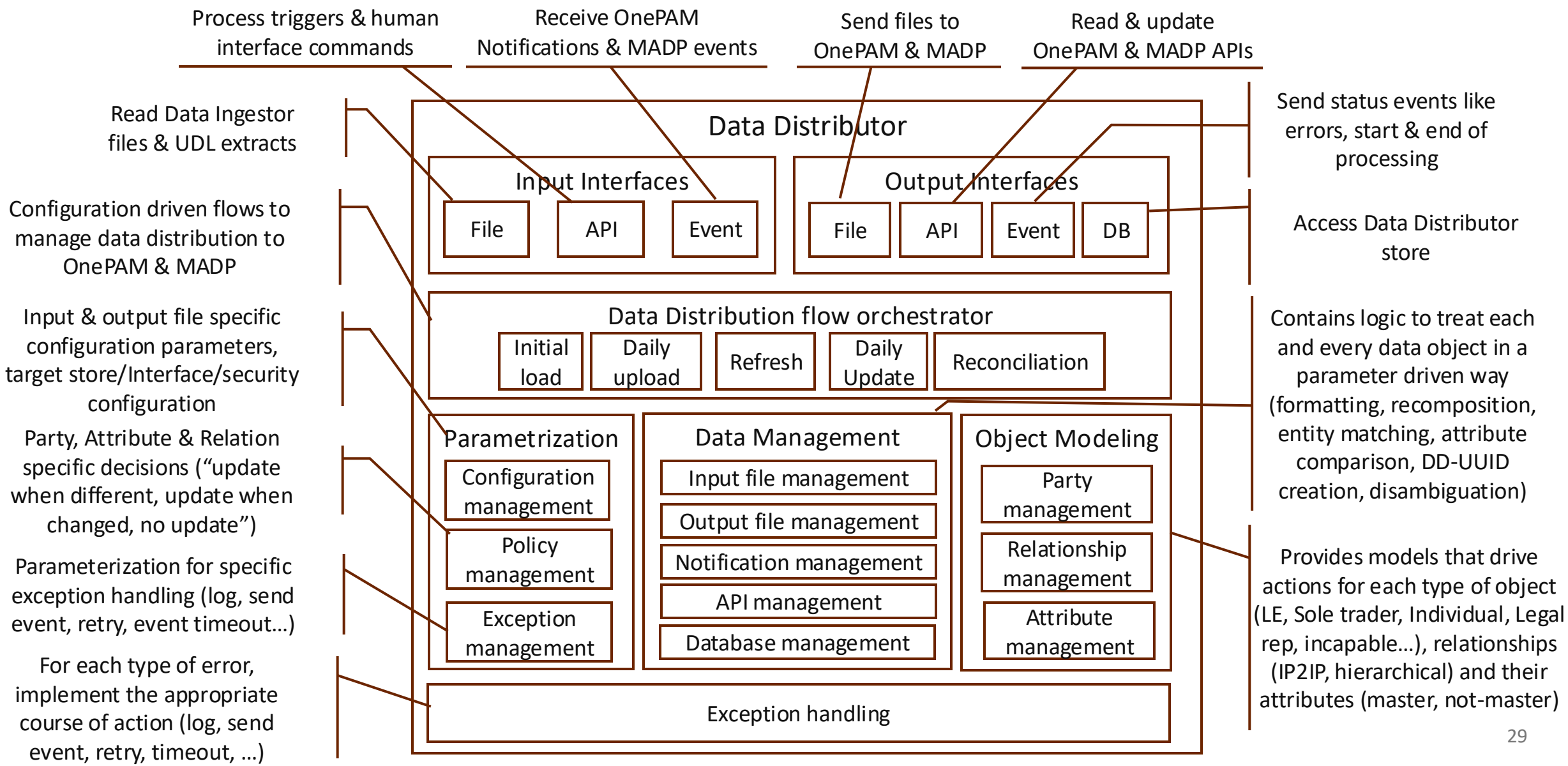


Table of Content

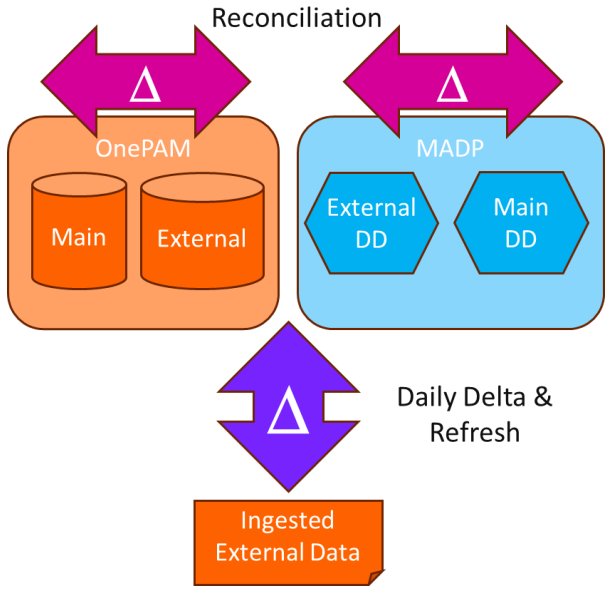
Introducing Data Refresher (DRF) & Data Reconciler (DRC)

Refreshes are external ingested files coming from the external supplier, but contain the complete data set, rather than the daily delta's. They are like initial loads that you regularly redo, but on an already existing body of data.

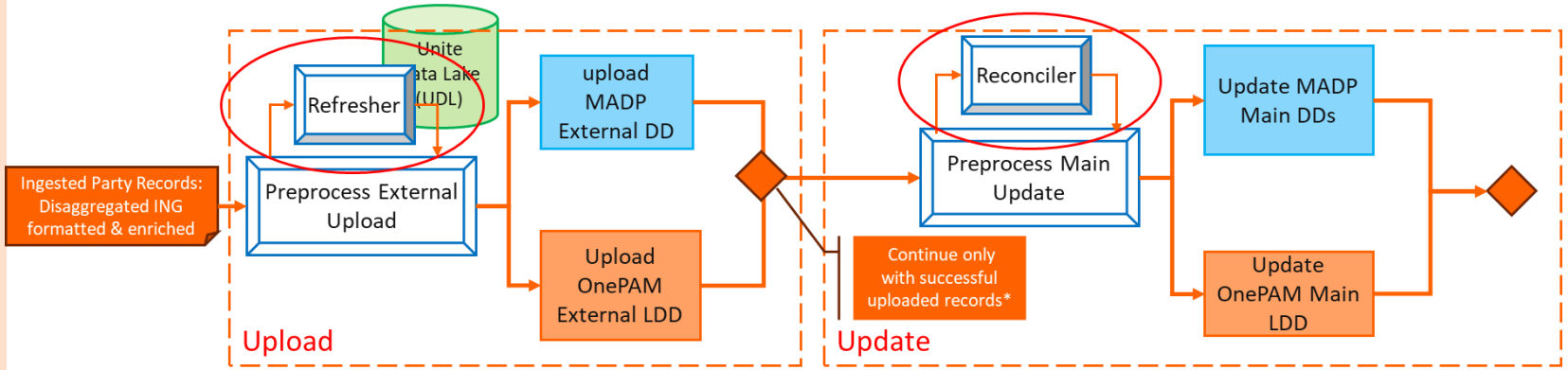
It is a "*vertical*" movement of data, from the supplier to the **Data Ingestor** to the **Data Distributor** to the **OnePAM & MADP External zones.**

Reconciliations are comparisons between the **OnePAM & MADP External and Main zones**, where the information in the External zones are considered "master" for some attributes. It introduces no new data.

It is a "*horizontal*" movement of data from the **External Zones** to the **Main zones**.



While complete data sets with millions of records must be compared, we are only interested in updating the differences between the refreshes & the External zones data, and, for reconciliation, the External zones & the Main zones data.

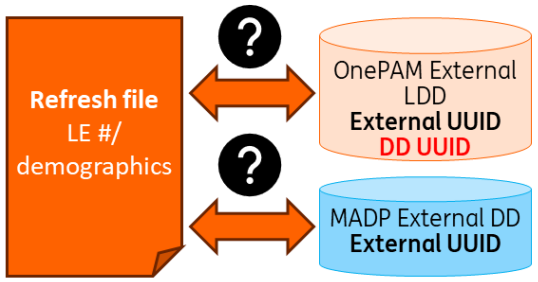


- **The Refresher:** compares external ingested full datasets with what is in the UDL copied OnePAM/MADP External zones and extracts deltas.
- **The Reconciler:** compares the data between the in the UDL copied OnePAM/MADP External and Main zones and extracts deltas.
- **Deltas** are processed by the Data Distributor as if daily deltas.

Table of Content

Refresh OnePAM & MADP : Exploit DD UUID as common UUID

Step 1: Start situation

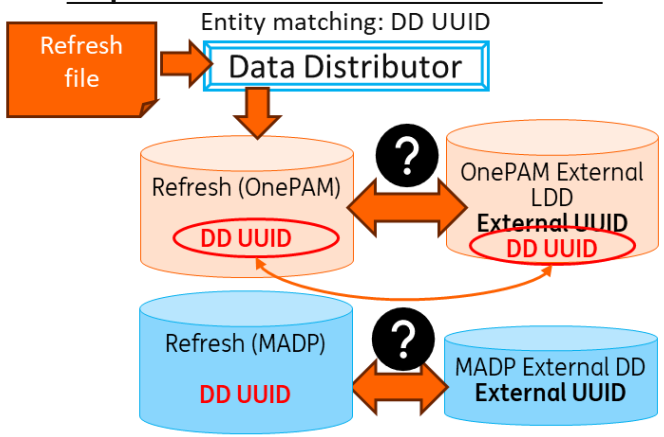


During refresh we must compare attributes between the:

- **OnePAM External LDD** & **MADP External DD** and
- the Parties & relationships in the refresh file

The problem is that there is **no common key** of comparison between External Zones & records in the refresh file.

Step 2: add DD UUID to OnePAM refresh

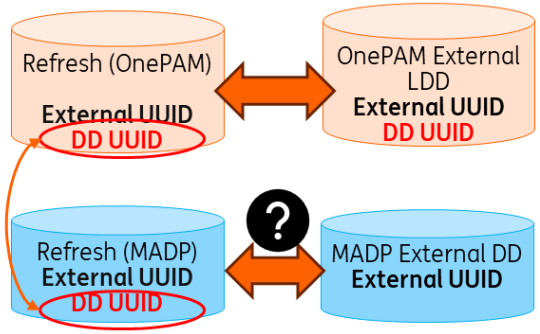


We use the **DD UUID** as a common key and the **Data Distributor** which has all the entity matching information for the **OnePAM External LDD.**

We load the refresh file to the **DD** doing entity matching, adding the **DD UUID** to matching records. New Parties from the refresh file create new **DD UUIDs**.

We split* the refresh file in **OnePAM** and **MADP** parts & can compare the **OnePAM External LDD** with the **OnePAM** refresh part, using the **DD UUID** to enrich it with the **External UUID**.

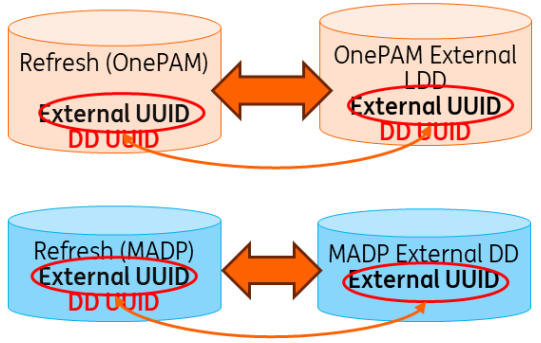
Step 3: enrich MADP Refresh with External UUID



To support comparing the **MADP** refresh & **External DD** parts, we have enriched the **OnePAM** refresh part with the **External UUID** from the **OnePAM External LDD**.

Then we compare the **DD UUIDs** of the **OnePAM** and **MADP** refresh parts and enrich the **MADP** refresh part with the **External UUID**.

Step 4: compare using matching UUIDs



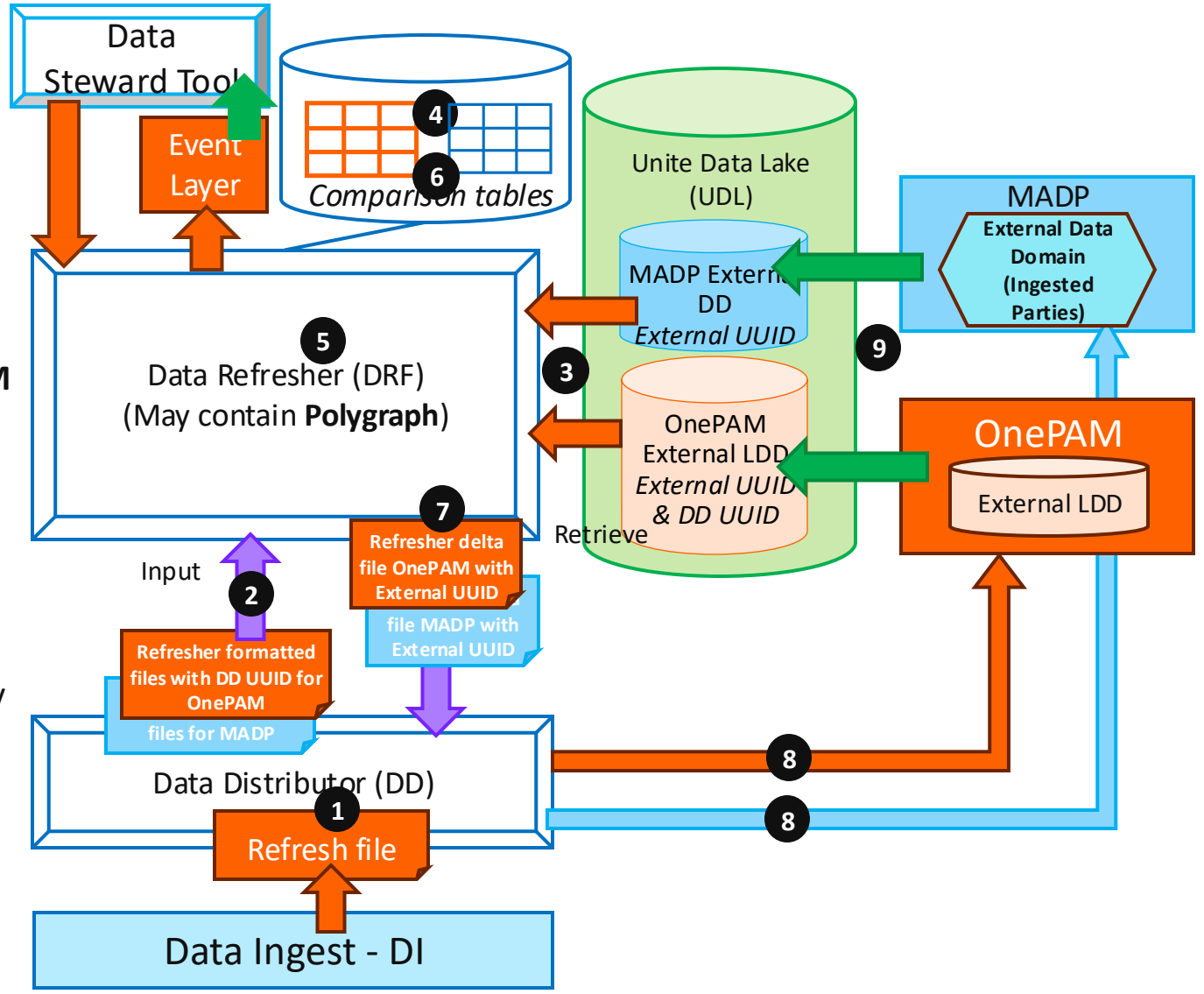
We can now compare the **Party** Core attributes and relationships of the **OnePAM External LDD** with the **OnePAM** refresh using either **DD UUID** or **External UUID** as key.

We can also now compare the "Local" attributes of the **MADP External DD** with the **MADP** refresh using the **External UUID** as key

* We split up the OnePAM & MADP refresh parts in the Data Distributor which has all the context information. This is before matching wither External UUIDs, which happens in the Refresher. We don't want the Refresher to "know" the difference between Core and Local attributes for code duplication avoidance. It just compares different sources.

Full (high-level) architecture of the Data Refresher

1. **Data Distributor** reads *refresh file*, entity matches, adding **DD UUID** (creates if new Party) & splits into **OnePAM** Core attributes and **MADP** Local attributes.
2. **DD** uploads files to **Data Refresher (DRF).**
3. **DRF** obtains **UDL** extracts of **OnePAM External LDD** & **MADP External Data Domain**.
4. **DRF** compares **OnePAM External LDD** extract with **OnePAM** part of the *refresh file* using **DD UUID** as key.
5. **DRF** enriches the **MADP** part of the *refresh file* with **External UUID** from **OnePAM External LDD UDL** extract using **DD UUID** as common key
6. It then compares (**Polygraph?**) **MADP External DD** extract with **MADP** *refresh file* using **External UUID** as common key
7. Delta **OnePAM** & **MADP** files with changed Parties, attributes & **External UUID** are send back to **DD**.
8. **DD** continues to upload the **OnePAM** and **MADP External Zones**
9. All updates generate events that update the **UDL**



The **Data Refresher** acts like a bypass for the Data Distributor to reduce the size of the refresh files to similar as for daily uploads but keeping the overall flow the same as for daily uploads

Functional architecture of the Data Refresher

The architecture of the **Data Refresher** is very similar to the one of the **Data Distributor (DD)**.

With a modular approach it could be embedded in the same codebase as the **Data Distributor** where its specificities are modeled in separate modules:

- It ingests different input files from the **Data Distributor**, as well as from the **UDL**.
- It has a different, complementary refresh flow than the DD.
- It doesn't do entity matching, but it does key-matching & addition
- It uses different tables
- It doesn't track response events
- It does similar delta verifications on attributres.
- Sends files back to **DD**, not **OnePAM/MADP**

In a multi-instance deployed setup, one of the deployed Data Distributors could take the role of Data Refresher.

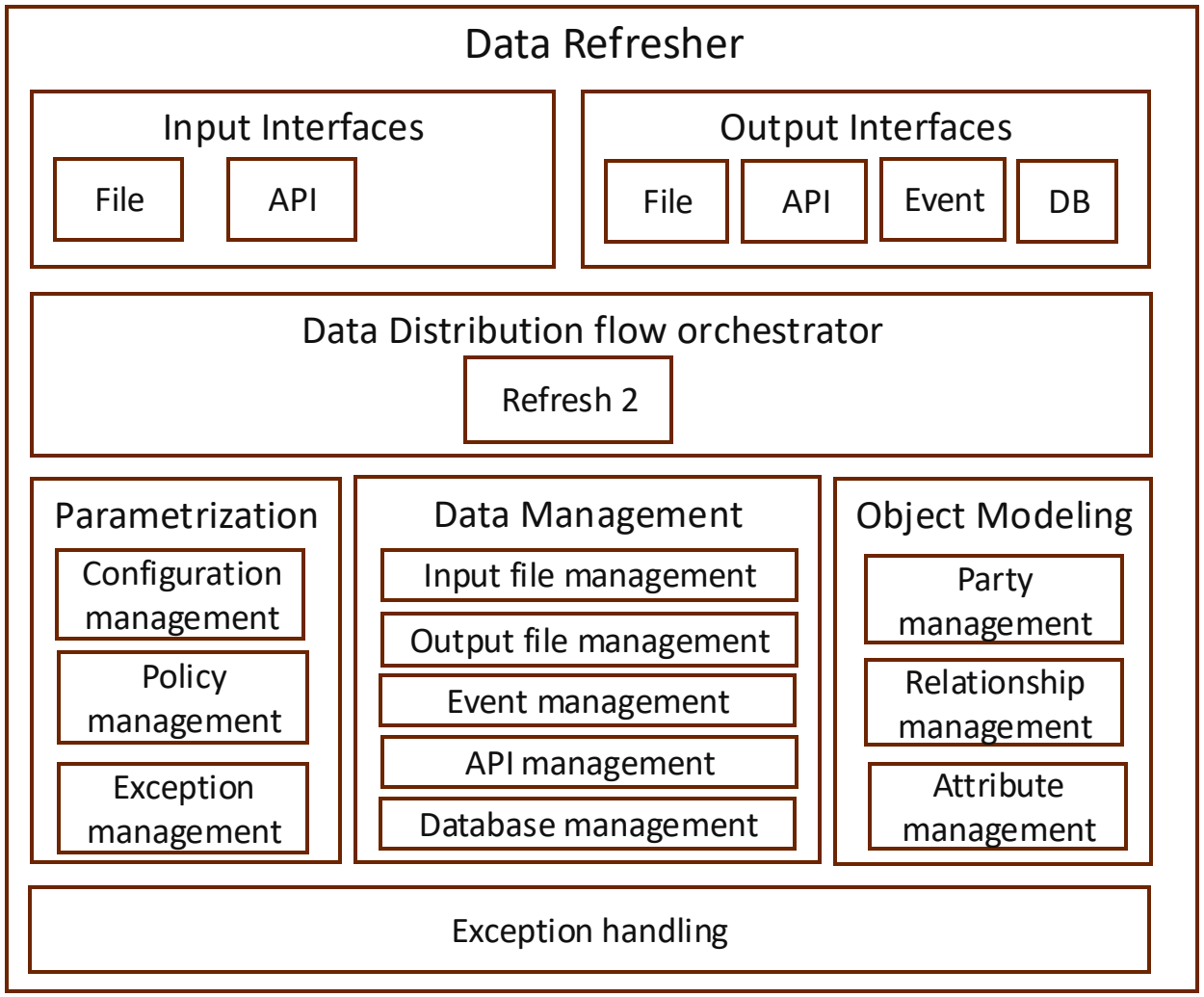
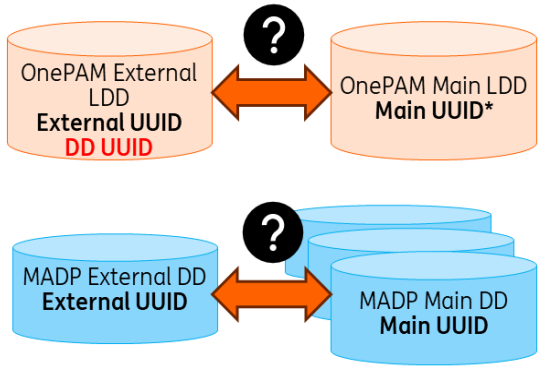


Table of Content

Reconciliation OnePAM & MADP: exploit DD UUID as common UUID

Step 1: Start situation



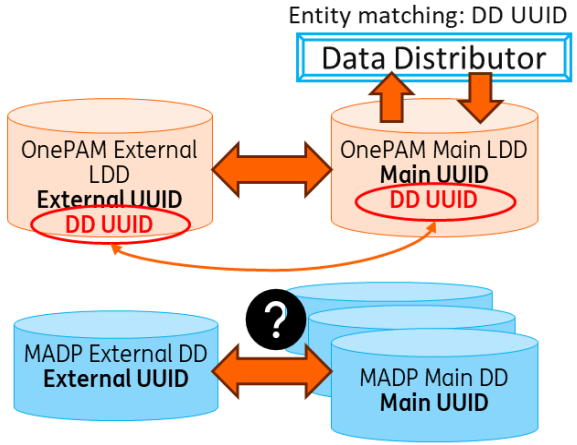
During reconciliation we must compare attributes between:

- The **OnePAM Main LDD** and **OnePAM External LDD**
- The **MADP External DD** and **MADP Main DDs**

The challenge is that there is ***no common key*** of comparison between External & Main data stores.

This is especially an issue with **MADP** where also the attributes are different between External & Main DDs.

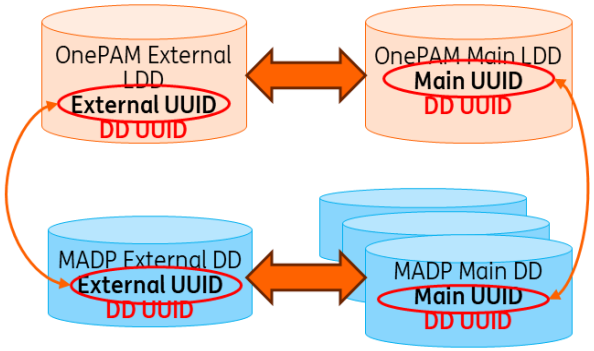
Step 2: add DD UUID to Main LDD extract



We will use the **DD UUID** as a ***common key*** and use the **Data Distributor** which has all the entity matching information for the **OnePAM External LDD**, which is what we want to compare with.

We extract the **OnePAM Main LDD** to the **DD** that does entity matching and adds the **DD UUID** to matching records. Unmatched records are dropped as they are not in the External LDD.

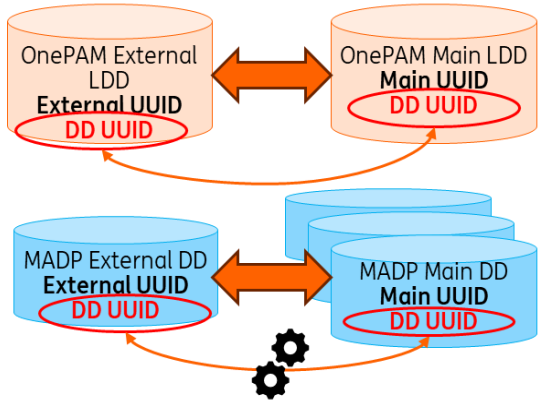
Step 3: enrich MADP DDs with DD UUID



We will then use the **External UUID** as a common key between **OnePAM External LDD** and **MADP External DD** to enrich **MADP External DD** with the **DD UUID.**

We will use the **External UUID** as a common key between **OnePAM Main LDD** and **MADP Main DDs** to enrich **MADP Main DDs** with the **DD UUID**

Step 4: reconcile using DD UUID to match



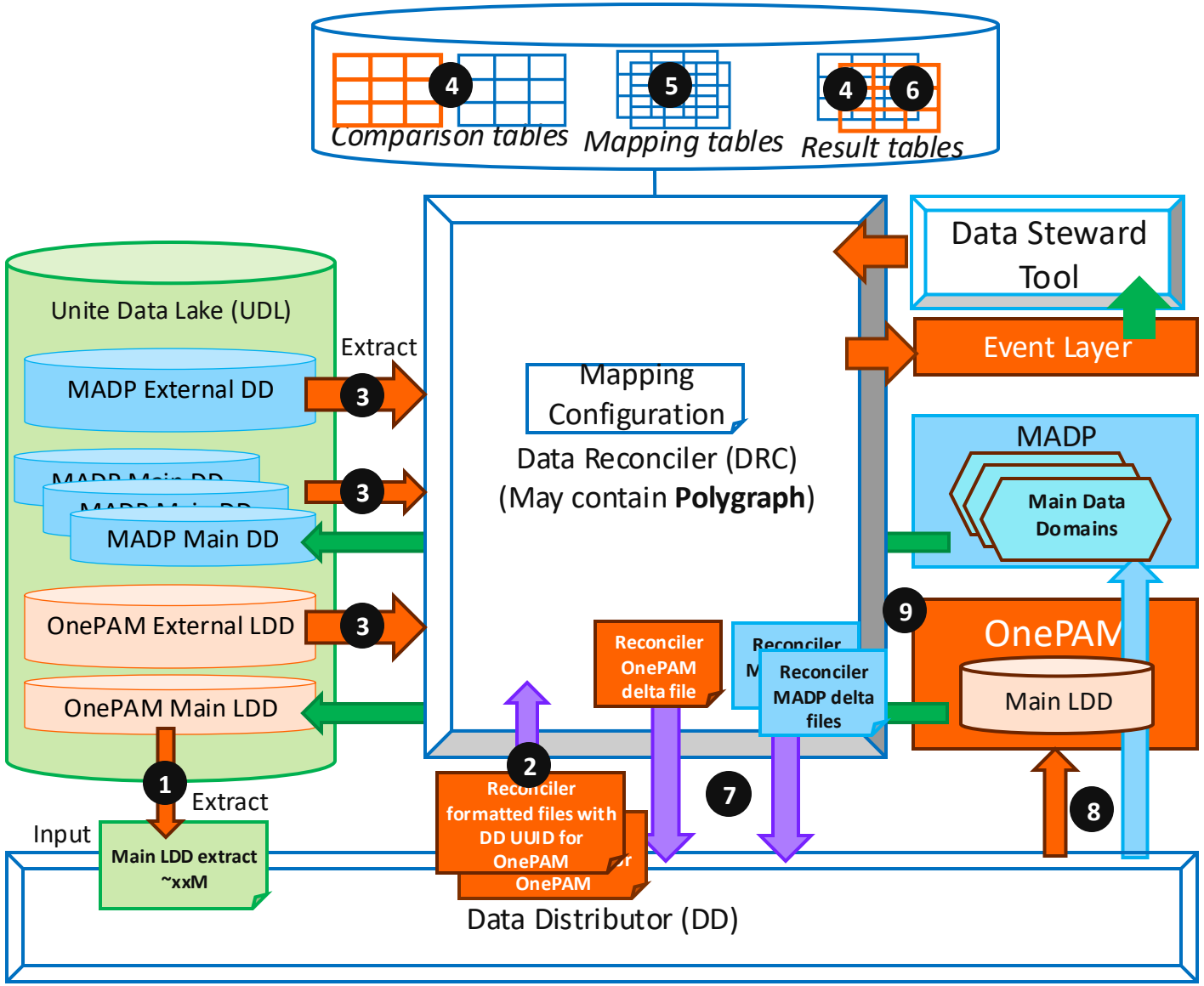
We now reconcile **OnePAM Main LDD** & **OnePAM External LDD** by matching **DD UUID** and *comparing* the attributes the External LDD is master for with matching Main LDD attributes.

We do the same for the **MADP External DD** & **MADP Main Data Domains**, matching the **DD UUIDs**. We must *transform* External DD attributes into Main DD attributes as their models are different.

* Over time the Main LDD will also have the External UUID as an Internal Identifier but that is through daily updates, not through reconciliation (unless we want a giant reconciliation).

Full architecture of the Data Reconciler

1. **Data Distributor** reads extract of **OnePAM Main LDD** from the **UDL**, entity matches & adds **DD UUID** on matched records, removing non-matching records (not in External LDD).
2. **DD** uploads DD UUID-tagged & Main LDD extract files to the **Data Reconciler** (DRC).
3. DRC gets **OnePAM External LDD**, **MADP External DD** & **MADP Main DDs** extracts from UDL.
4. **OnePAM Main & External LDD** are stored in *Comparison tables* to be directly compared using the **DD UUID** with deltas written in the *Result tables*. **(Polygraph?)**
5. **MADP External & Main DDs** are first stored in *Mapping tables.* We enrich them with **DD UUIDs,** by mapping their **External/Main UUIDs** on the **OnePAM External & Main LDD** extracts in the *Comparison tables*. *We* delete records with no matches.
6. DRC compares **MADP External & Main DDs** using the **DD UUID** with deltas written to the *Result tables*.
7. DRC creates delta files for **OnePAM & MADP Main zones** & sends them back to the **Data Distributor**.
8. **DD** updates the OnePAM & MADP Main zones.
9. All updates generate events that update the **UDL**



Functional architecture of the Data Reconciler

The architecture of the **Data Reconciler** is very similar to the one of the **Data Distributor (DD)**.

With a modular approach it could be embedded in the same codebase as the **Data Distributor** where its specificities are modeled in separate modules:

- It ingests different input files from the **Data Distributor**, as well as from the **UDL**.
- It has a different, complementary reconciliation flow than the **DD**.
- It doesn't do entity matching, but it does key-matching & addition
- It uses different tables
- It doesn't track response events
- It does similar delta verifications on attributes.
- Sends files back to **DD**, not **OnePAM/MADP**

In a multi-instance deployed setup, one of the deployed Data Distributors could take the role of Data Reconciler.

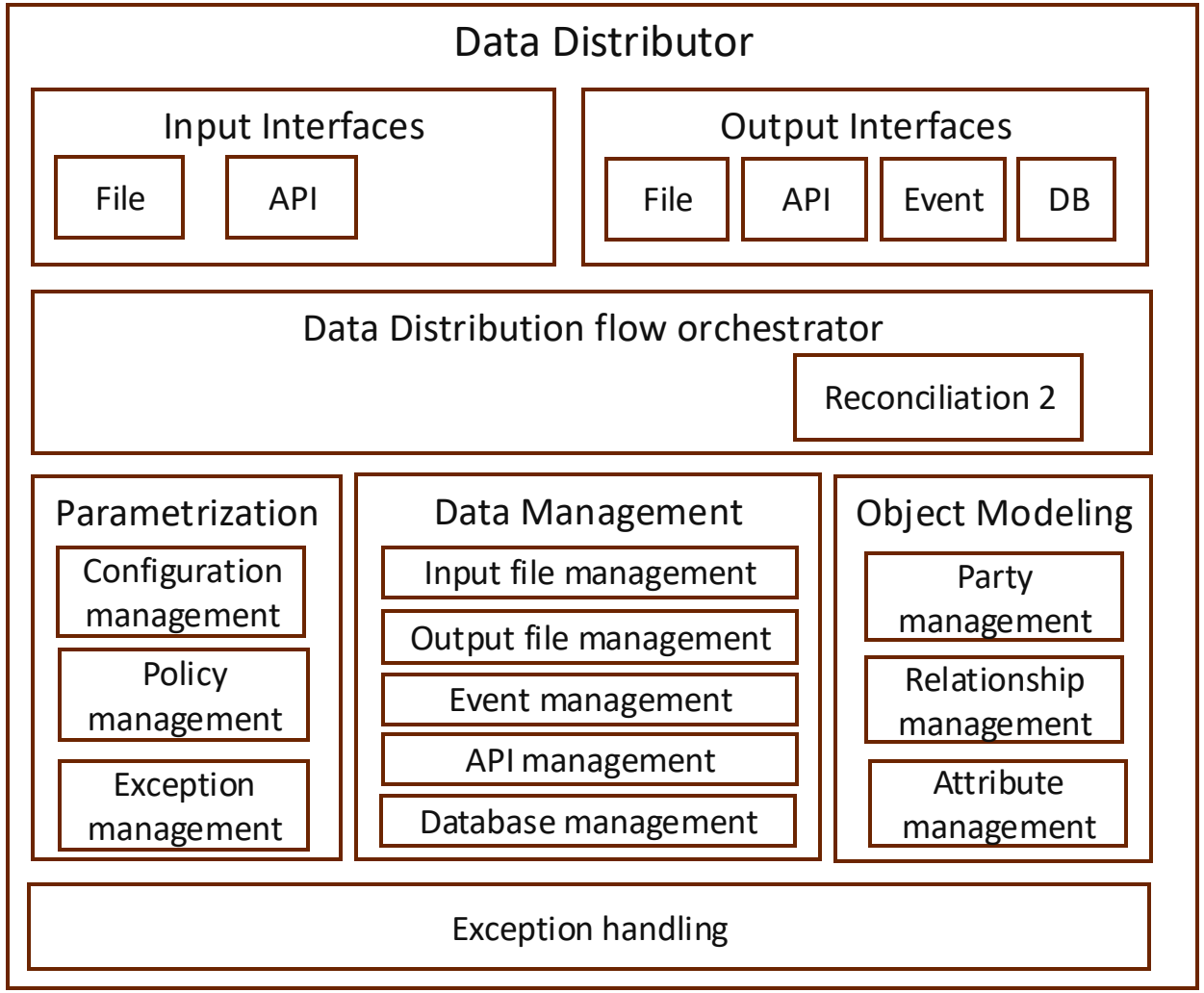


Table of Content

Architectural Decisions (1/2)

1. **We support both File-based as API-based upload and update scenarios**
   - **Why**: different drivers for using API and Files. For small numbers of ingested records that are urgent (bancruptcies, Incapables) APIs give us better control over the ingest process. For large numbers of Ingested records, file-based uploads are better (also required by GCDM).
   - **Consequence**: more flexibility (if configuration driven) but more complex application
2. **Data Ingestor filters out disaggregated ingested records that have not changed**
   - **Why**: We want to be efficient and remain within upload time windows. As upload throughputs are limited and OnePAM anyway verifies and doesn't overwrite attributes that didn't change, leading to wasted cycles.
   - **Consequence**: unchanged records don't 'touch' the associated OnePAM objects and don't change the meta-data
3. **Before uploading/updating data we read current OnePAM & MADP Parties & Relationships to obtain a before-view.**
   - **Why**: We don't know what exact attributes changed of an ingested disaggregated record. Comparing the before-view with the ingested change-view allows us to identify the exact attribute changes and provide only changed data to upload improving throughput. It is necessary when using asynchronous notifications (file uploads) to know when all updates have been captured.
   - **Consequence**: balance between time spent reading (1000 tps) vs uploading unchanged attributes (5 tps).
4. **We use a dedicated DD-UUID, managed by the Data Distributor.**
   - **Why**: To support deduplication of Parties by storing demographics & LE# in the DD-store (*disambiguation service* impact tbd). To create new Parties in the OnePAM External LDD using OIL, a 3rd party identifier is required. To Support reconciliation and refreshes as a primary key to provide mappings to External and Main UUIDs. (Quid Polygraph?)
   - **Consequence:** We need to request GCDM for a new "InternalIdentifier" attribute value for OnePAM (e.g. BE_CD_DD).
   - **Remark:** OnePAM has a feature where its UUIDs can be created outside OnePAM (feature request to be made). This may eliminate the need for a separate DD-UUID.

Architectural Decisions (2/2)

4. **Historization is in the UDL, we don’t keep historization of attribute changes in the Ingestor or Distributor.**
    - **Why:** Because. It’s what the UDL if for. The Data Distributor as an operational system should not be concerned with this.
    - **Consequence:** we use data from the UDL for refreshes and reconciliations rather than that in OnePAM and MADP.

Risks

- **The largest risk is that current OnePAM throughput is limited.**
  - Regardless of implementation (file/API) we are not sure if it can manage all Ingest loads, when we have a full end-to-end scenario with OnePAM External & Main LDD and MADP data.
  - We should open a negotiation with GCDM to get at least 25 TPM (API)/25 RPM.
  - For the moment we can live with the lower througputs as we only have to update the OnePAM External LDD.
- **Total throughput time** must remain within permitted time windows (6-8 hours)*.
  - The total throughput time from arrival of the external file to be ingested through Data Ingestor/Distributor up to OnePAM Main LDD & MADP Main DDs could be extensive due to amount of processing, handovers and asynchronicity.
  - We update in 6 stages: Read/update OnePAM External LDD, Read/Update MADP External DD, Read/Update OnePAM Main LDD, Read/Update MADP Main DDs.
  - With Files for OIL and MADP, given that the file-based uploads are asynchronous, we must capture Notifications of changes to be sure all records are successfully written before moving to the next phase. We don't know yet how fast Notifications will come through.
  - APIs are synchronous giving us more control but constraint by API throughput limits (5 tx/s). This works for smaller updates but breaks down for larger updates. GCDM requires file uploads for larger volumes.
- The assumption that Notifications will allow us to capture all changes to account whether all updates have been done remains to be tested.
- Updating Main LDD may be difficult because of the multiple sources of data that concur to define the "truth" (survivorship) – to be tackled in the policies.

**In principle daily upload files should be < 10K updated records per file and at supported OIL update rates this should be feasible*

Open Points

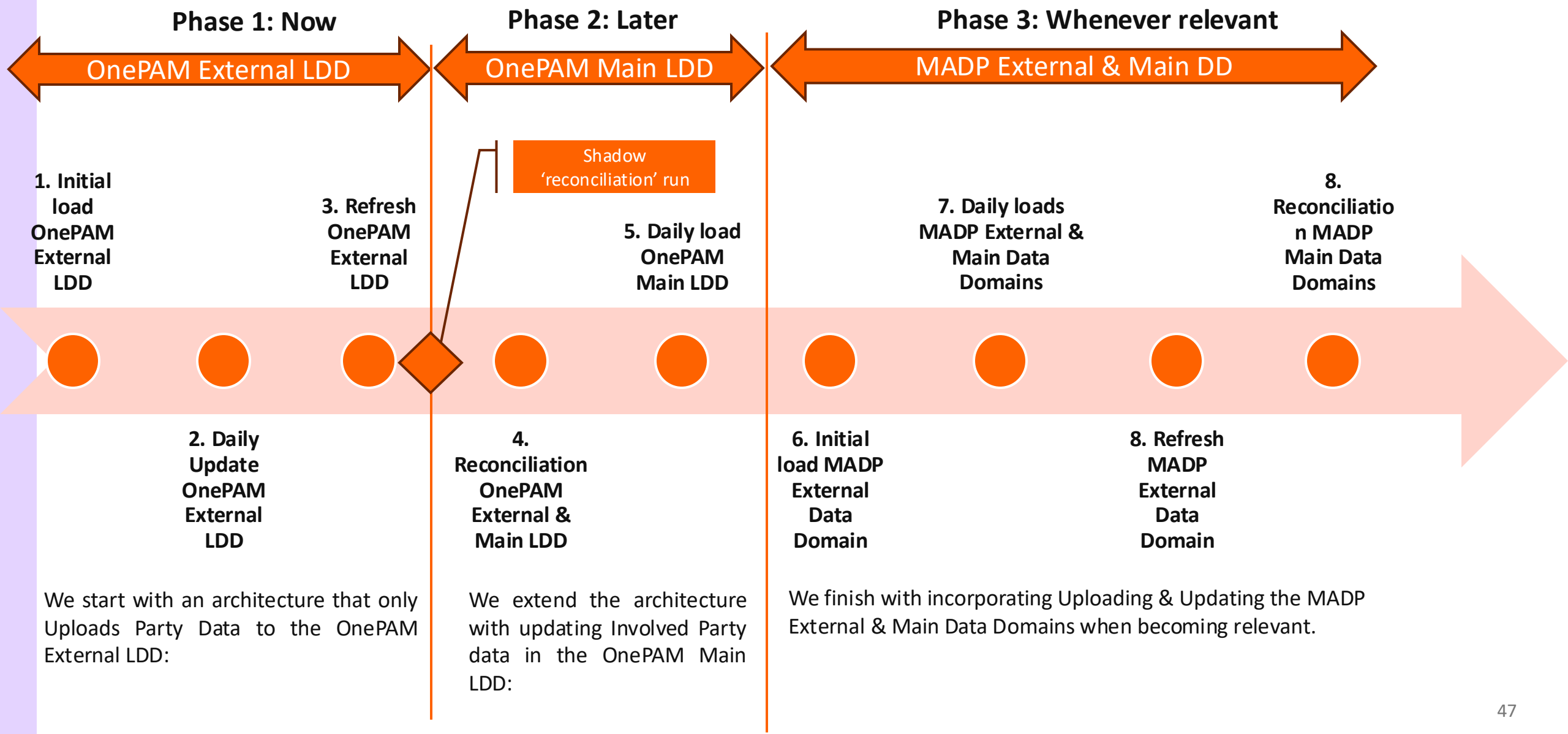
- A number of business modeling decisions have not yet been taken:
  - Sole Trader ingested Party object not yet modeled for external LDD (proposal is different than Main LDD) or know how to treat it.
  - Legal representative approach not yet defined.
  - Some ingested Party objects have bad data quality (Legal Representatives without address) or have limited information (administrators) risking to have several duplicates. Decision how to treat those is open.
- An important open question to move from External to Main zones is what the granularity must be for an upload in the External Zones to be considered successful before updating the Main zones:
  - The original ingested composite record? (How will DD even know what records it ingests were part of it?)
  - The by DD ingested disaggregated record?
  - Individual uploaded attributes?
- For the OnePAM Main LDD there may be competing updates for attributes (e.g. Cube). How to prioritize and know that the attribute you may update is not updated by another source having a higher priority (see 'survivorship' in the next slide)
- The use of a **central entity matching (disambiguation)** mechanism is on the table but not yet designed.
- GCDM open points:
  - Challenge on 'voluminous' data volumes: use of External LDD for LE/OpSites (10M disaggregated records)
  - Throughput performance (25 tps/rps instead of 3-5 tps/rps),
  - define OnePAM UUID outside OnePAM
  - Ability to not need an external source UUID (DD-UUID).

**In principle daily upload files should be < 10K updated records per file and at supported OIL update rates this should be feasible*
*** Airflow could be a possible solution according to Kjell Moens.*

Part 2: Phased Approach & Patterns

Table of Content

An evolutionary architectural approach



We start with an architecture that only Uploads Party Data to the OnePAM External LDD:

We extend the architecture with updating Involved Party data in the OnePAM Main LDD:

We finish with incorporating Uploading & Updating the MADP External & Main Data Domains when becoming relevant.

Different Data Distributor Upload and Update Patterns

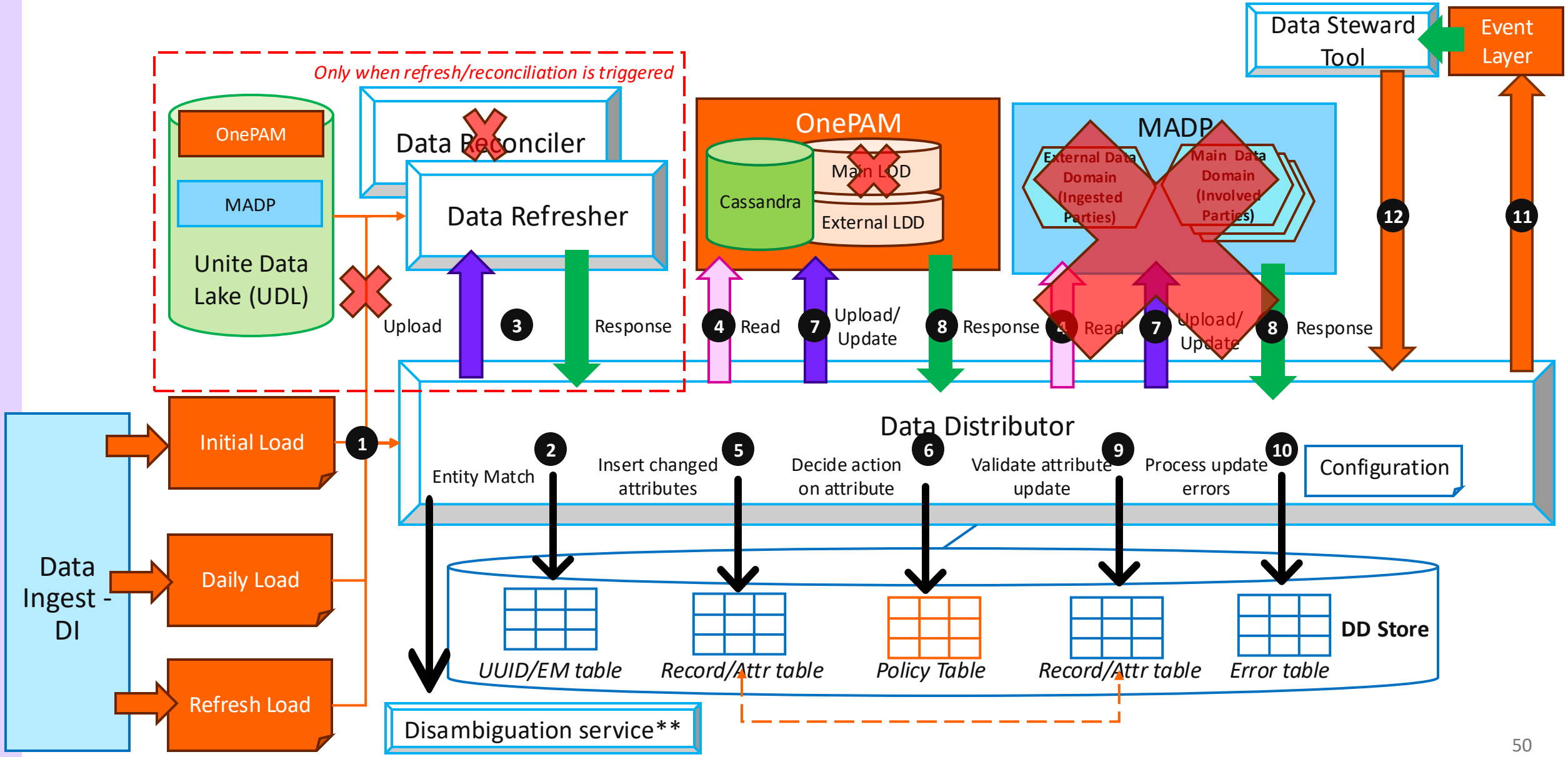
	Phase 1: OnePAM External LDD Upload	Phase 2: OnePam Main LDD Update	Phase 3a: MADP External DD Upload	Phase 3b: MADP Main DD Update
Initial Load	• Very first (empty db) • Independent (no overlap) • Relationship creating • Object sharing diff attr. • Object sharing same attr.	*There is **no initial load** from the perspective of the Data Distributor as it already contains data by other sources & DD is not master.*	• Object sharing diff attr.	*There is **no initial load** from the perspective of the Data Distributor as it already contains data by other sources & DD is not master.*

Daily Load	• File based
	  • larger files
	  • Scheduled OIL
	• API based
	  • Urgent, smaller files
		• API based	• File based (larger files)
			  • Automated
			• API based (urgent files)
			Follows OnePAM External LDD Upload
				• API based
				Follows OnePAM Main LDD Update

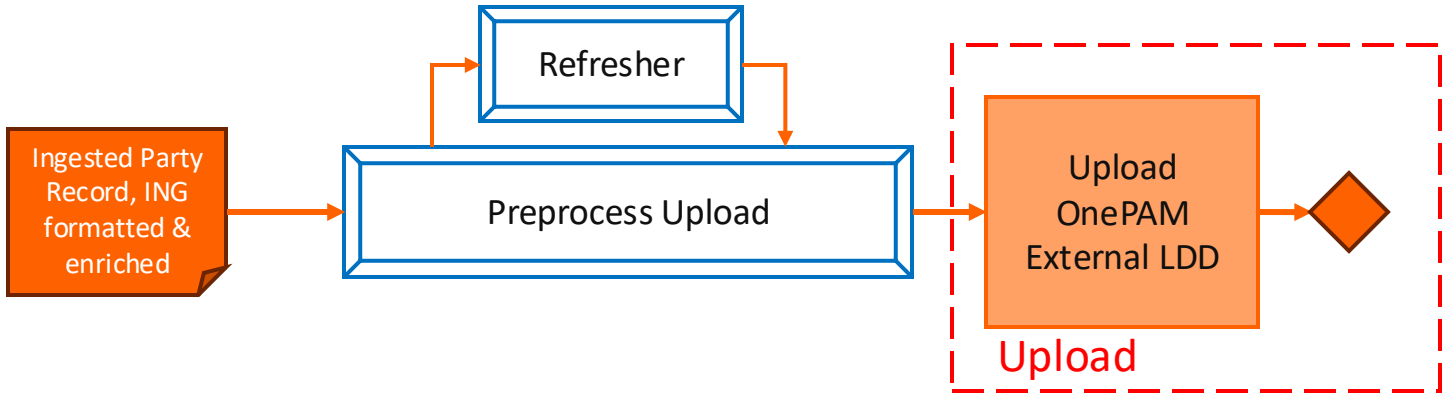
Corrections	• **Refresh**
	  • converted into “daily load”
	  • File based (larger)
	  • Manual OIL
		• **Reconciliation**
		  • First one acts as initial load equivalent
		  • Converted into “daily load”
		  • File based
		  • Manual OIL
			• **Refresh**
			  • converted into “daily load”
			  • File based
			  • Automated
				• **Reconciliation**
				  • First one acts as initial load equivalent
				  • Converted into “daily load”
				  • File based
				  • Automated

Table of Content

Phase 1: OnePAM External LDD upload



Phase 1: OnePAM External LDD Upload



3 activities:

1. **Initial Upload** of Parties (~10M disaggregated uploadable records) – File based
   - **The very first upload** of any ingested source – LE & Operating Sites
   - **Subsequent first uploads** of other ingested sources (bankruptcies, incapables, legal representatives)
2. **Daily Upload** of Parties – **File & API based**, depending on type of file
   - **Legal Entities & Operating Sites** (~50-60K disaggregated uploadable records): File or API-based
   - **Bankruptcies** (10s-100s of disaggregated records): API-based
   - **Incapables** (10s-100s of disaggregated records): API-based
   - **Legal representatives**: TBD (not in immediate scope)
3. **Refreshes** (~10M disaggregated uploadable records) – File based
   - We don't do an upload of all these records as most didn't change.
   - We will transform them into a Daily Upload using the UDL

2 process steps:

1. **Preprocess External Upload**:
   - Entity Matching of **Parties** (if needed)
   - Data Distributor UUID creation for new Parties
2. **Upload OnePAM External LDD**:
   - Get 'before' view to identify changed attributes (if needed)
   - **Upload** Core model *changed* attributes (OIL files or API)
   - Capture upload results (Notifications or API response)

The overlapping Party nature in initial loads must be considered

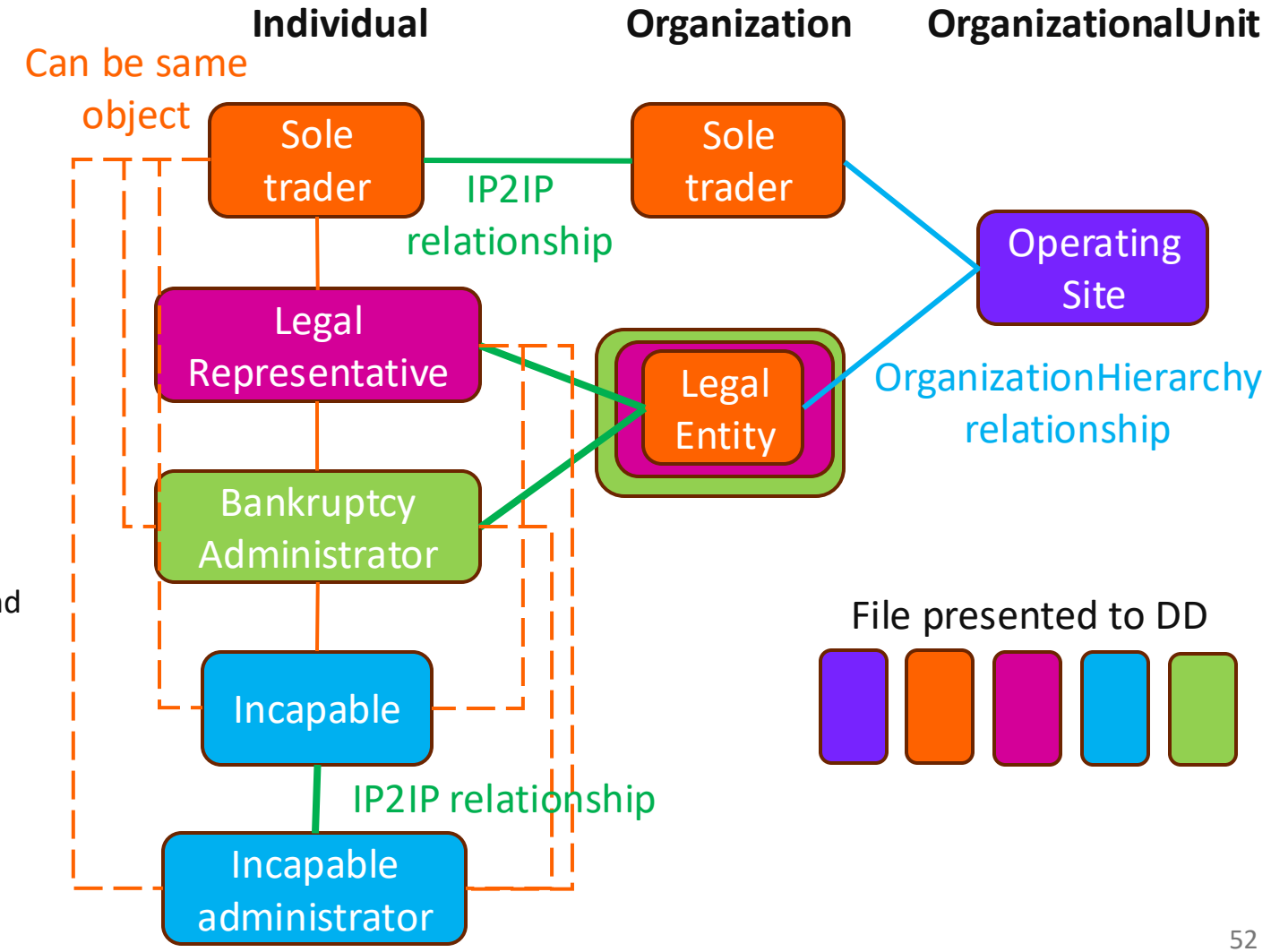
The **Data Distributor** gets presented with different files as initial loads in a serial way: *Legal Entities (& Sole traders*), Operating Sites, Bankruptcies, Incapables*.

The **first file** will encounter an empty **External LDD** & there is no assumption of overlaps of relationships between parties

From the **second file on** there may be **overlaps in Parties** (*Individuals/Organization*) requiring entity matching as well as **relationships** with *Organizations* that must be created.

That means that **from the second file onwards**, when there is an entity match, we must **update** the object, rather than create it, and whether we update overlapping attributes.

We also must evaluate if an upload would result in a Notification (there is no Notification when all info is the same) to know whether the upload succeeded or not.



* Sole traders exact modeling in OnePAM External LDD is still under consideration. The current shown proposal is not yet approved and may change as it is not aligned with the model in OnePAM Main LDD.

Impact assessment in order of initial uploads

Order	File	Object	Potential overlap	Relationship
1	Legal Entity	LE	No	No
2	Operating Site	Operating site	No	Legal Entity (Hierarchy)
3	Bankruptcies	Bancruptcy Administrator	No	Legal Entity (IP2IP)
4	Incapable	Incapable	Yes Individual*	No
5	Incapable	Incapable Administrator	Yes Individual	Incapable (IP2IP)
6	Sole trader	Sole trader Organization*	No	No
7	Sole trader	Sole trader Individual	Yes Individual	Sole trader organization (IP2IP)
8	Sole trader	Sole trader Operating Site	No	Sole trader organization
9	LE representatives	LE representative	Yes Individual	Legal Entity (IP2IP)

4 types of initial uploaded files based on order:

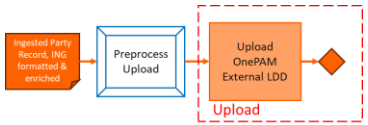
1. The very first one has by definition no impact on any other file. The database is empty. We assume this will be *Legal Entities*.
2. Some initial uploads (after the first one) **don't affect** earlier objects (*Incapable* doesn't affect *Legal Entity* or *Operating Site*)
3. Some initial uploads (after the first one) **only have a relationship** with earlier uploaded (*Operating site* with *Legal Entity*)
4. Some initial uploads (after the first one) **can update** the same object (*LE representative* and *Incapable*)

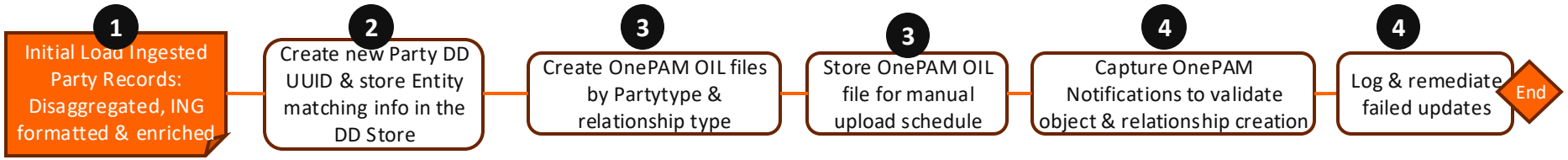
We assume that for each file we initially upload after the first one, when entity matching indicates an update & not a create, either:

- **there is differentiating information** (attributes) for those objects, and we know for reach Party updated there should be a **Notification** sent that allows us to gauge the success or failure (for example: setting an incapable flag on an existing individual).
- There is **no differentiating information** in the object (**no Notification**) and the differentiation lies, for example, in the creation of a relationship. (e.g. LE representative that is also an incapable) and we know we will get a **Notification** for that.

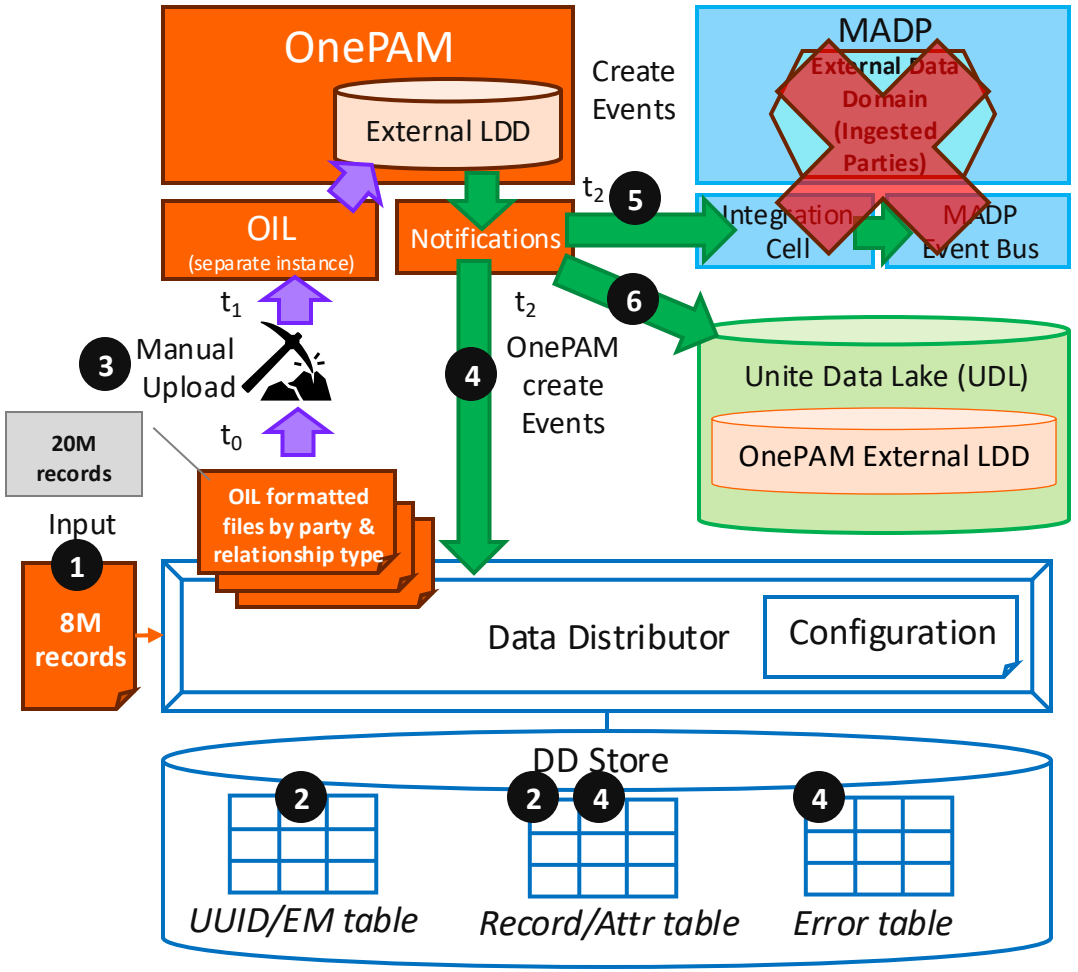
**OnePAM InvolvedParty type*

1a. OnePAM "External" LDD – 'first' initial Upload pattern





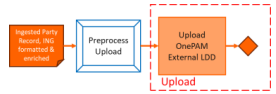
1. The **first** initial load (Legal entities) is an exceptional event due to:
   - The very large number of records (3M ingested ~ 8-10M disaggregated records after processing by the Data Ingestor).
   - The **OnePAM External LDD** is empty*.
2. The **Data Distributor creates a Data Distributor UUID (DD UUID**)** for each record and **stores it permanently** together with identifying info (LE #) in the DD Store. We also keep a **temporary record/attribute table** to tabulate create results.
3. The **DD creates the OIL files** & **transfers them** for **manual scheduling** of the upload into **OnePAM External LDD** by **OIL**. **OnePAM** uploads the files over a prolonged period in the required order.
4. We **capture all creation events** from **OnePAM External LDD** to account for all created records listed in the *Record/Attr table* and update the *Error table* for the ones we don't receive (after a timeout).
5. **MADP** will not capture **Party** creation events to process into an External Data Domain (which doesn't exist at this point).
6. **Notifications** with created & changed **Parties** & relationships are also picked up by the **UDL** & create a copy of the **OnePAM External LDD.**



** OIL requires always an external identifier to be present when uploading data

* We assume that if Incapables is the first upload with 'Individual' objects we can treat it as if it is a first initial upload as it has no relationships with Legal Entity and Operating Sites

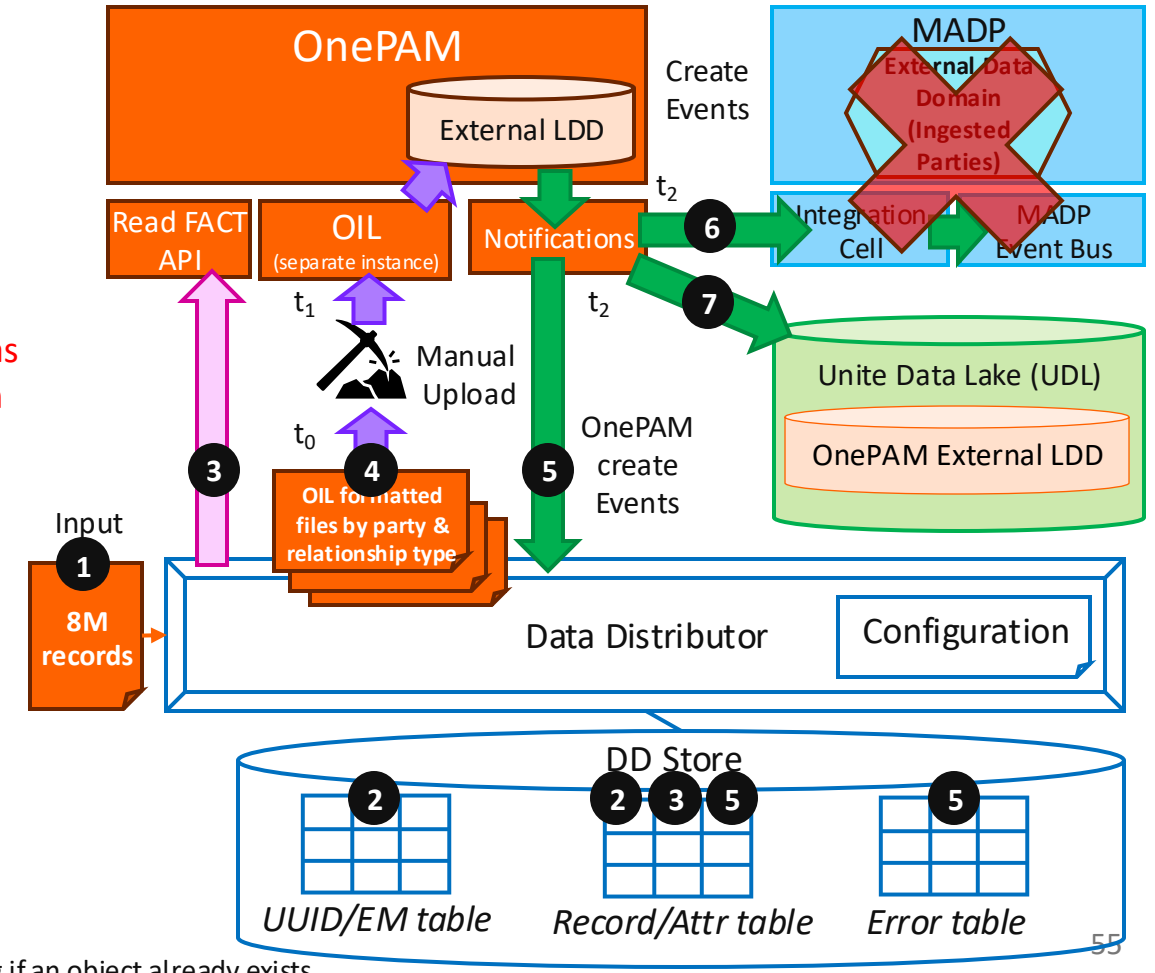
1b. OnePAM "External" LDD – subsequent Initial uploads (1/2)



1. Initial Load Ingested Party Records: Disaggregated, ING formatted & enriched
2. Read & do entity matching in DD Store~
2. Create new Party DD UUID & store Entity matching info in the DD Store
3. If relevant, read current Party values from OnePAM External LDD
4. Create OnePAM OIL files by Partytype & Relationship record
4. Store ONEPAM OIL file for manual upload schedule
5. Capture OnePAM Notifications to validate object/rel creation
5. Log & remediate failed updates

End

1. Initial loads of the **subsequent** sources (Operating sites, …) must be treated differently as there may be **overlaps** with existing Parties:
   - We assume we know where to expect **Notifications*** or not.
   - The **OnePAM External LDD** is **not empty**.
2. The **Data Distributor** matches each record on existing records in the **DD Store** using demographics/LE # & disambiguates**. It creates **DD UUIDs** for new Parties. We indicate for all records whether to create/update.
3. To know whether to expect a **Notification** for the update, if our initial upload has higher priority on object attributes than earlier overlapping initial uploads, then we will read the as-is view for those objects we update to see if there are really changes on those attributes.
4. The **DD creates the OIL files** & **transfers them** for **manual scheduling** of the upload into **OnePAM External LDD** by **OIL**. **OnePAM** uploads the files over a prolonged period in the required order.
5. We **capture all creation/update events** from **OnePAM External LDD** to account for all created/updated records listed in the *Record/Attr table* and update the *Error table* for the ones we don't receive.

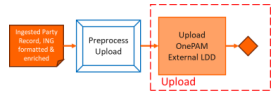


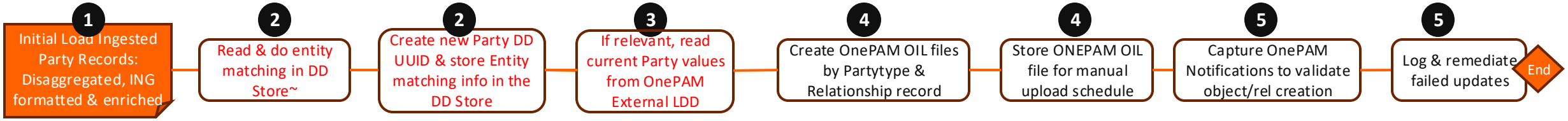
~difference with first initial upload in red

* Notifications are triggered when data is updated. Incapables set a flag. Legal reps may not update anything if an object already exists.

** Some ingested files have duplicates. for example, a legal representative can be related to different Legal Entities. Others not, like an Operating Site can only be attached to a single Legal Entity

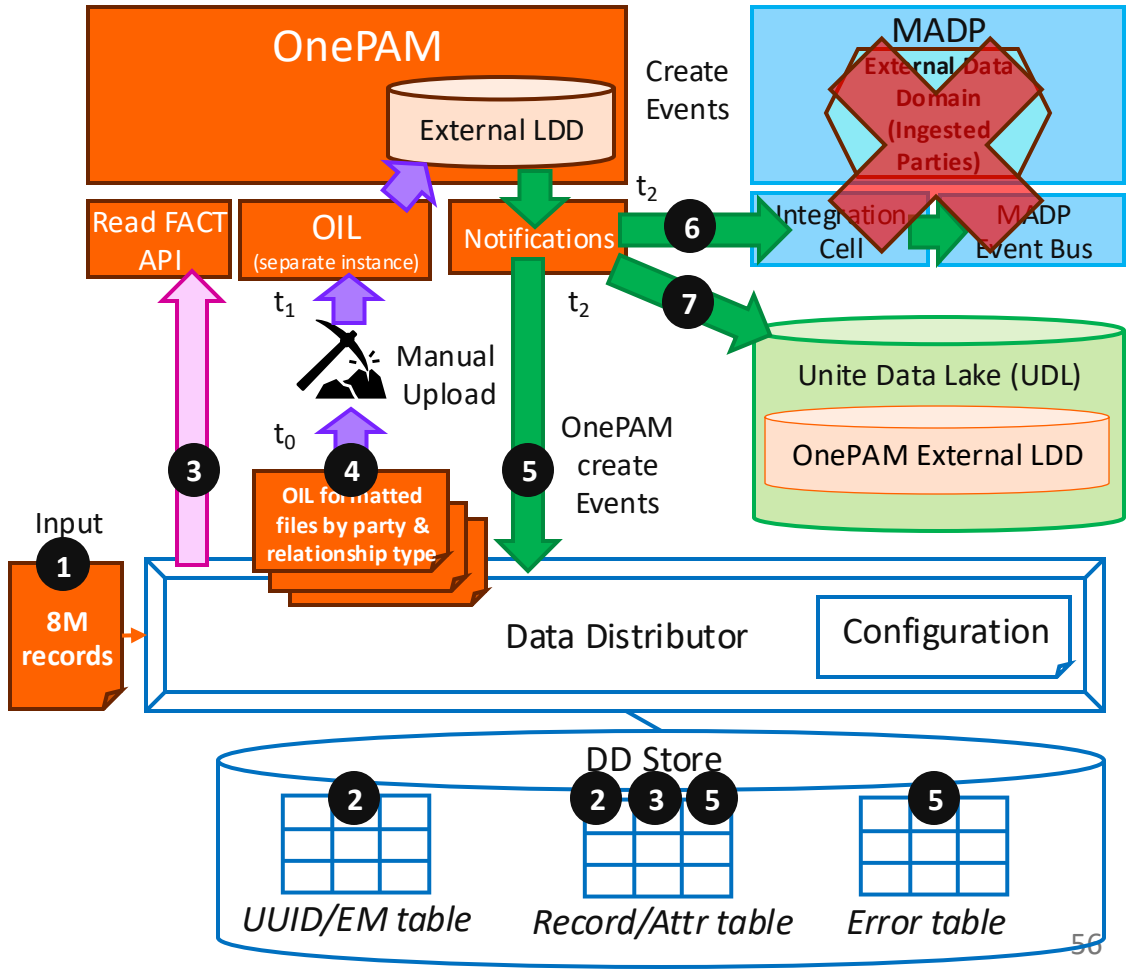
1b. OnePAM "External" LDD – subsequent Initial uploads (2/2)



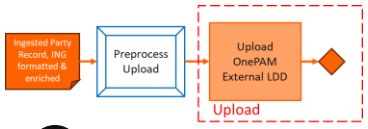


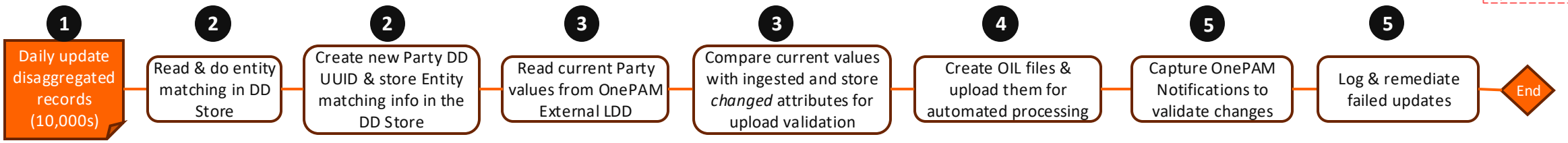
6. **MADP** will not capture **Party** creation events to process into an External Data Domain (which doesn't exist at this point).

7. **Notifications** with created & changed **Parties** & relationships are also picked up by the **UDL** & create a copy of the **OnePAM External LDD.**

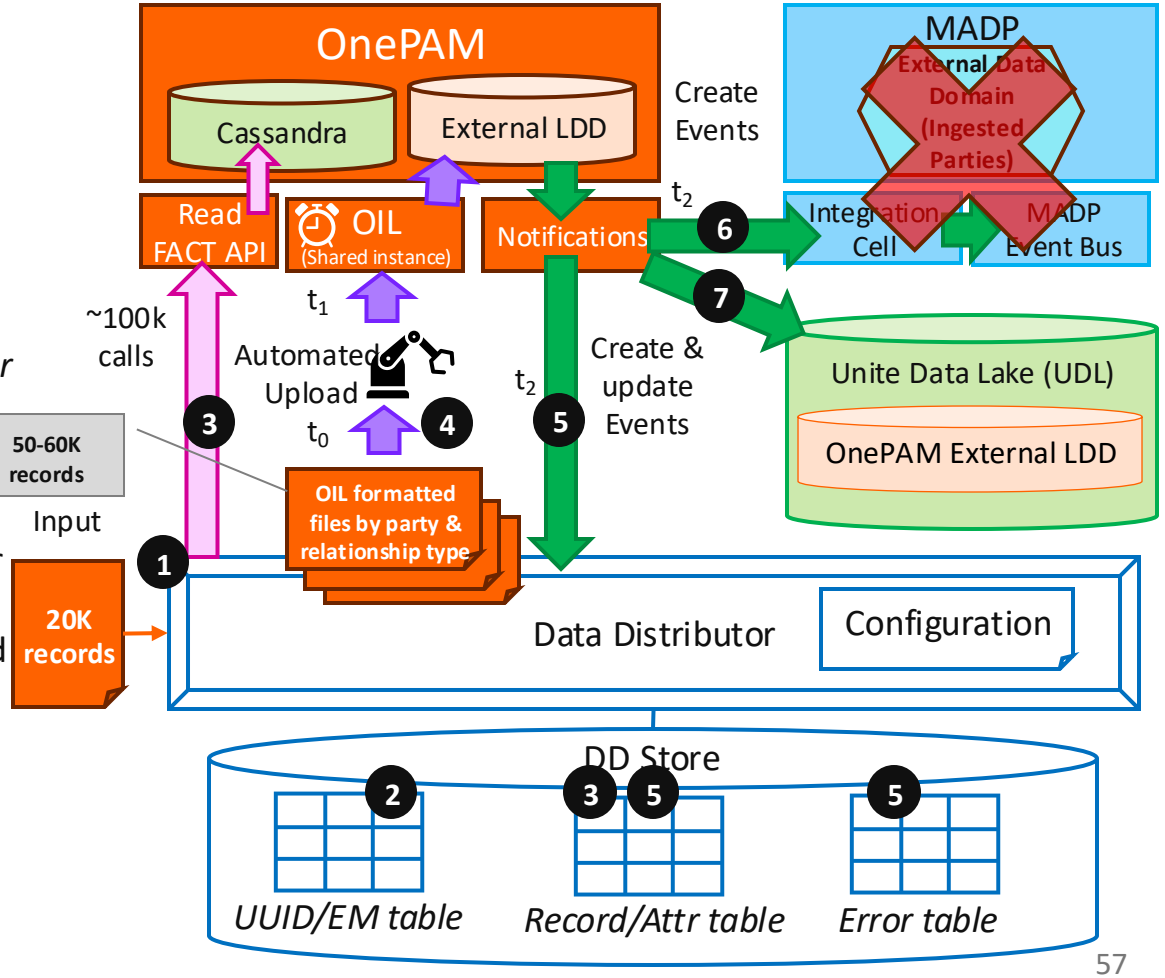


2a. OnePAM "External" LDD - Daily upload pattern – OIL



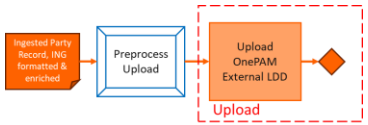


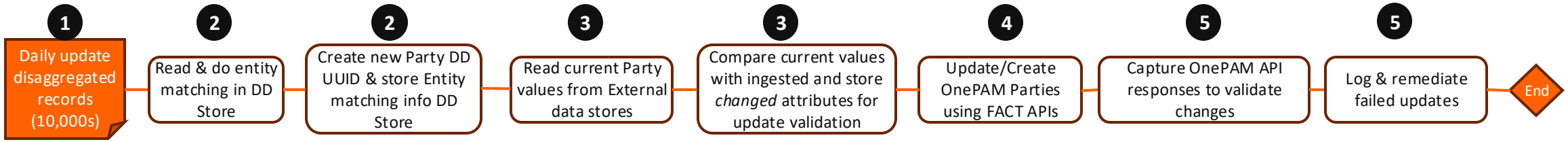
1. The daily upload contains around 20k disaggregated records which become 50-60k actually uploadable records (Parties & relationships)
2. The **Data Distributor** matches each record on already existing records in the **DD Store** using demographics/LE # & if necessary, disambiguates* them. It creates **DD UUIDs** for new Parties.
3. The **DD** reads all existing ingested Parties from **OnePAM** using the DD UUID to compare the attributes for changes & stores the changed ones in the *Record/Attr table* to account for updates.
4. The **DD creates the OIL files** & **transfers them to OIL** for **automated scheduling** of the upload into **OnePAM External LDD**.
5. We **capture all update, creation & deletion events** from **OnePAM** to account for all created Parties/relationships in the *Record/Attr table* & updated **Party** attributes and update the *Error table* for the ones we don't receive. Also deleted Parties & relationships are accounted for.
6. **MADP** will not capture Party creation events to process into an External Data Domain (which doesn't exist at this point).
7. **Notifications** with created & changed **Parties** & relationships are also picked up by the **UDL** & create a copy of the **OnePAM External LDD**



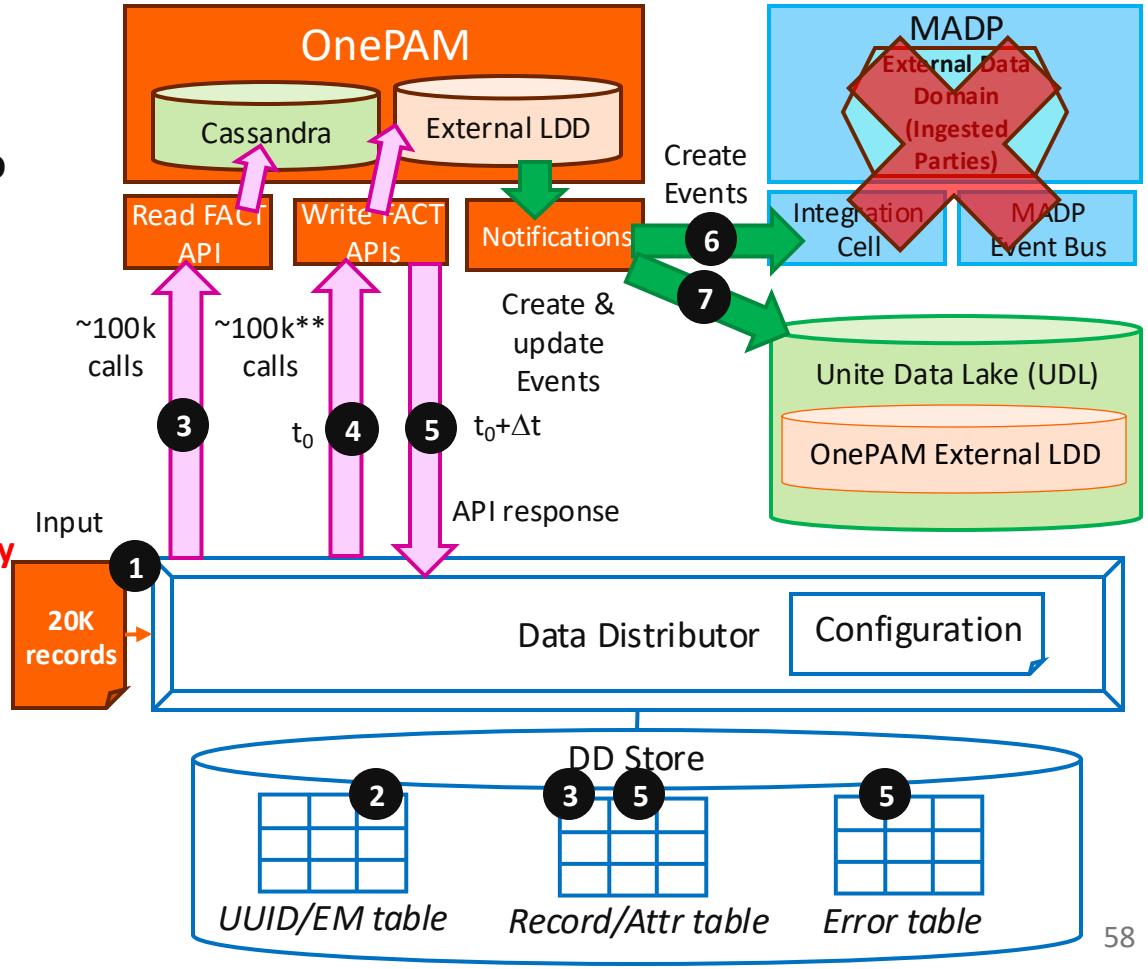
* Disambiguation is needed when there are multiple matches coming back. There might be an external service that will be created at some point to support this.

2b. OnePAM "External" LDD - Daily upload pattern – API





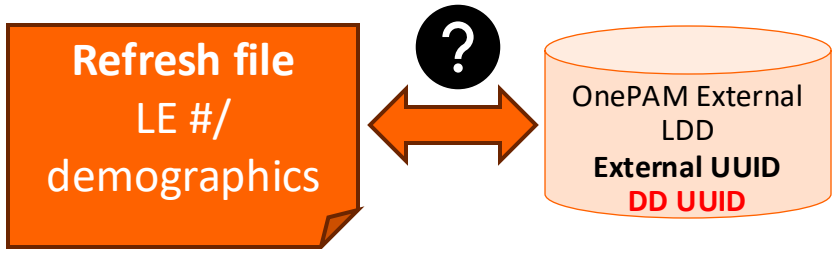
1. The daily upload contains around 20k disaggregated records which become 50-60k actually uploadable records (Parties & relationships)
2. The **Data Distributor** matches each record on already existing records in the **DD Store** using demographics/LE # &where necessary disambiguates them. It creates **DD UUIDs** for new Parties.
3. The **DD** reads all existing ingested Parties from **OnePAM** using the DD UUID to compare the attributes for changes & stores the changed ones in the *Record/Attr table* to account for updates.
4. The **DD updates** attributes of existing Parties and creates new parties & relationships as well as performs (soft) deletes using APIs*
5. The **DD** collects API responses to account for all created Parties & updated **Party** attributes and updates the *Error table* for the ones where errors are thrown. Also deleted Parties/relationships are accounted for.
6. **MADP** will not capture Party creation events to process into an External Data Domain (which doesn't exist at this point).
7. All **Notifications** with created & changed **Parties** & relationships are also picked up by the **UDL** & create a copy of the **OnePAM External LDD.**



*In red are the steps that are different with OIL, file-based uploads
** we assume on average >1 API call per updated Party/relationship.

Refresh OnePAM External LDD : exploit DD UUID as common UUID

Step 1: Start situation

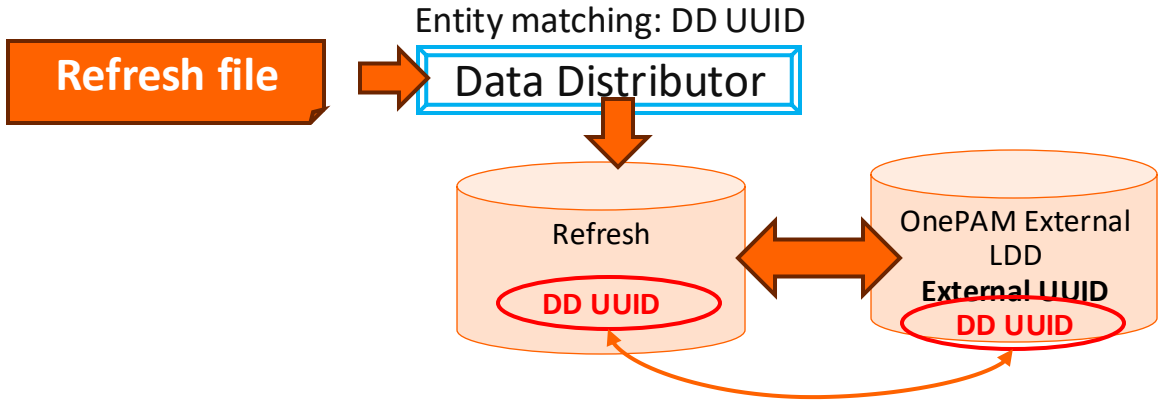


During refresh we must compare attributes between the **OnePAM External LDD** and the Parties & relationships in the refresh file

The challenge is that there is **no common key** of comparison between External LDD and records in the refresh file.

We will use the **DD UUID** as a common key and the **Data Distributor** which has all the entity matching information for the **OnePAM External LDD**, which is what we want the refresh file to compare with.

Step 2: add DD UUID to Refresh file



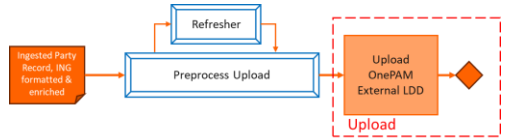
We load the refresh file to the **Data Distributor** who does entity matching and adds the **DD UUID** to matching records. Unmatched records from the refresh file create new **DD UUIDs** as they are not yet in the External LDD.

We now can match **External LDD Parties** with **Parties** in the refresh file through the **DD UUID** and compare them, as well as add the new **Parties & relationships** from the Refresh file in the delta output.

Given the volumes involved we will use the replicated OnePAM data from the **Unite Data Lake (UDL)** to do the refresh rather than production data in OnePAM.

* There is/was a discussion wherefor Sole Traders could be differently modeled between Main and External LDD due to different ways they could be created in the Main LDD. This discussion must be resolved.

3. OnePAM "External" LDD – Refresh pattern



1. Refresh Ingested Party Records: Disaggregated, ING formatted & enriched
2. Read & do entity matching in DD Store
2. Create new Party DD UUID & store Entity matching info in the DD Store
3. Create DRF files & upload them for processing.
4. Retrieve External LDD dump from UDL
5. Compare UDL data with refresh data
6. Send refresher deltas back to DD
7 8. Store *changed* attr in table & upload to OnePAM
9. Capture OnePAM Notifications to validate object/rel creation
9. Log & remediate failed updates
End

1. The refresh contains around 8M disaggregated records which become ~15-20M uploadable records (Parties & rels.).
2. The **Data Distributor** matches each record on already existing records in the **DD Store** using demographics/LE # &, if necessary, disambiguates them. It creates **DD UUIDs** for new Parties.
3. **DD** creates files with all refresh records and the **DD UUID** & uploads them to the **Data Refresher (DRF)** component.
4. The **DRF** retrieves all **Parties** & relationships of the **OnePAM External LDD** from the **UDL**.
5. Both are stored in a comparison table by **DD UUID** & compares differences (polygraph?) between the refresh files and **OnePAM External LDD.**
6. **DRF** creates new files with deltas or Parties & relationships and send it to the **DD**. These records are already **DD UUID** tagged.
7. We assume all attributes in the records have changed & store them in the *Record/Attr table* to account for updates in **OnePAM.**
8. **DD** creates OIL formatted files and transfers them to OIL for **manual scheduling** of the upload into **OnePAM External LDD**.
9. We **capture all update, creation & deletion events** from **OnePAM** and update the *Error table* for the ones we don't receive.
10. All **Notifications** with created/changed **Parties** & relationships are also

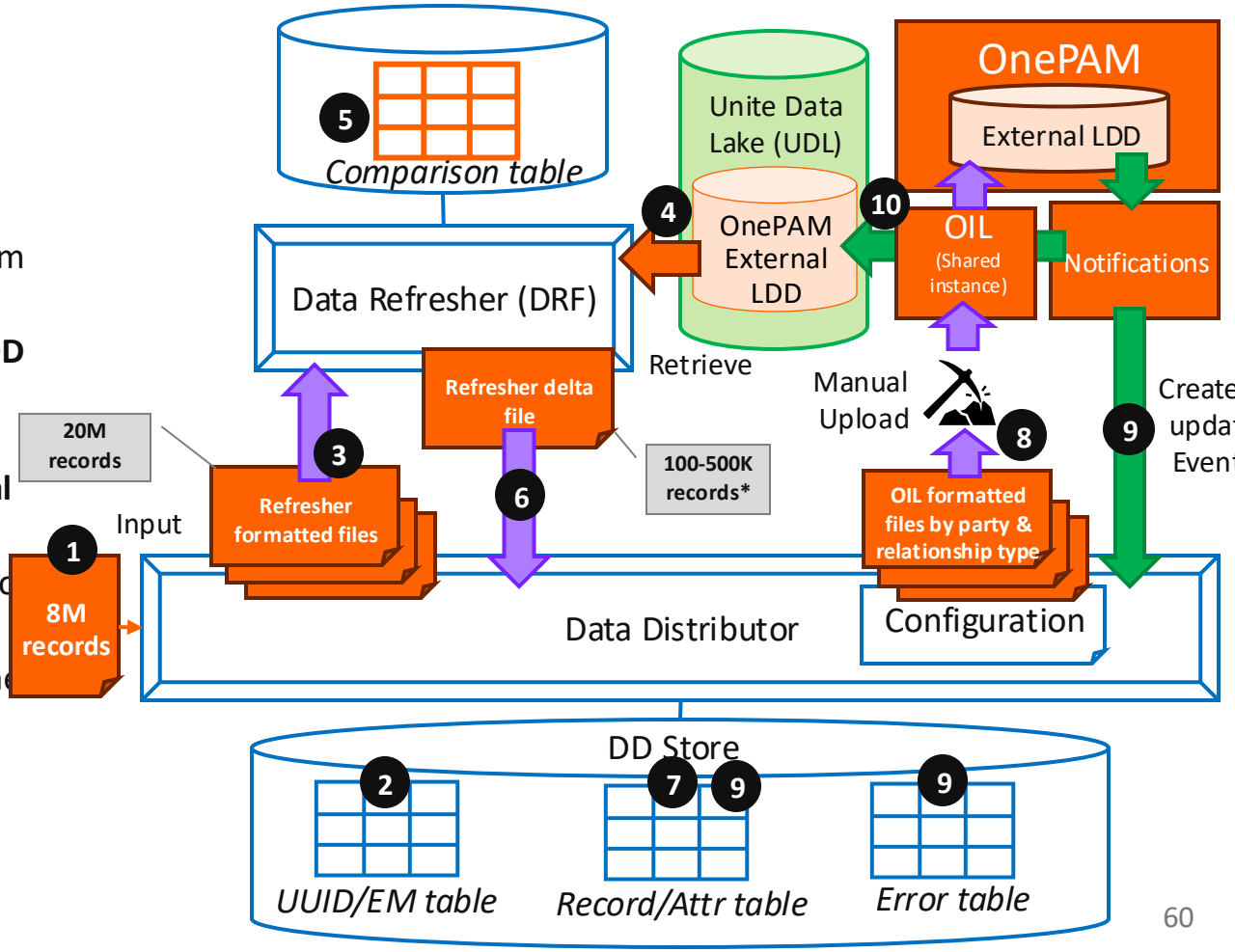
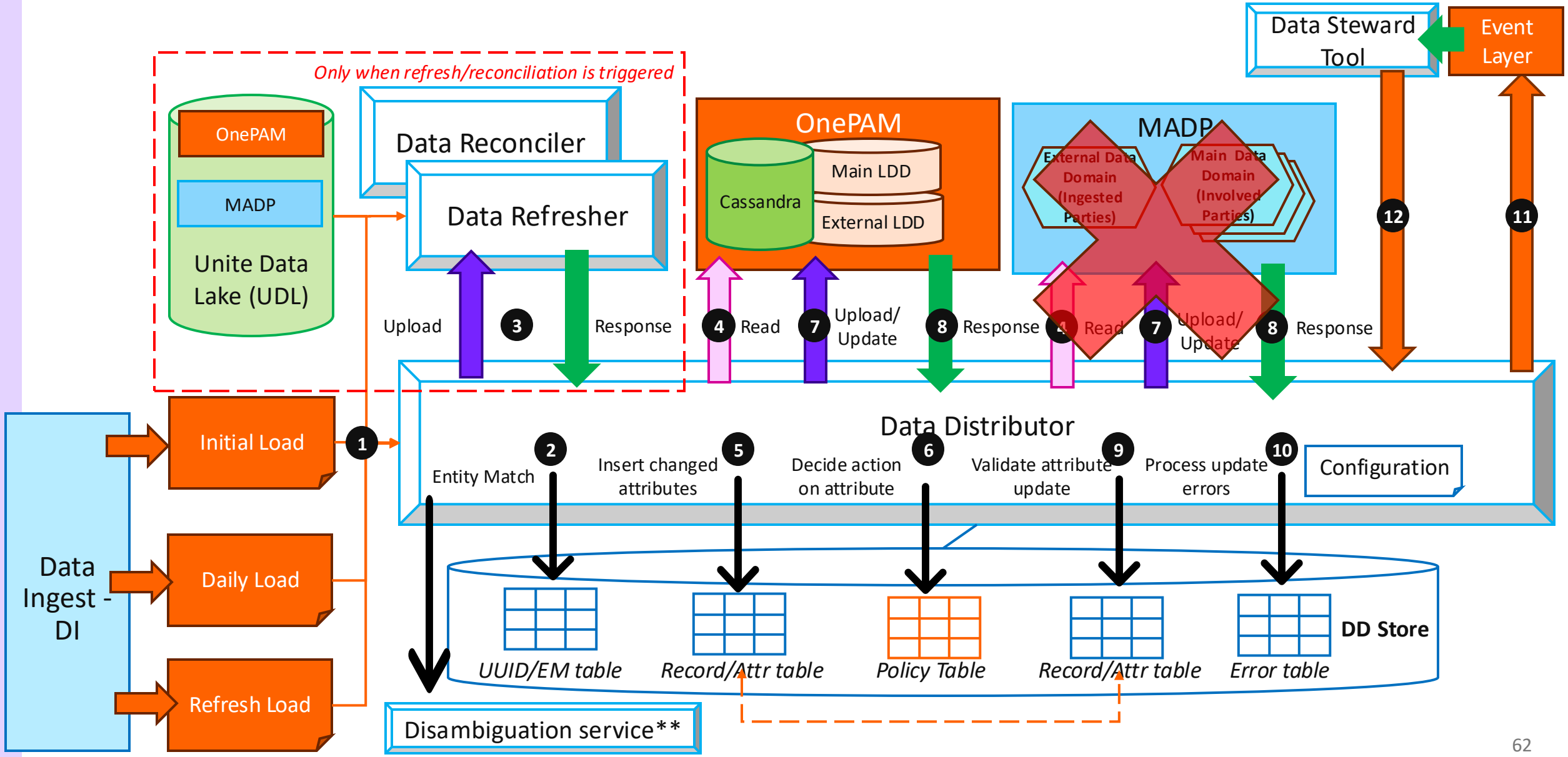
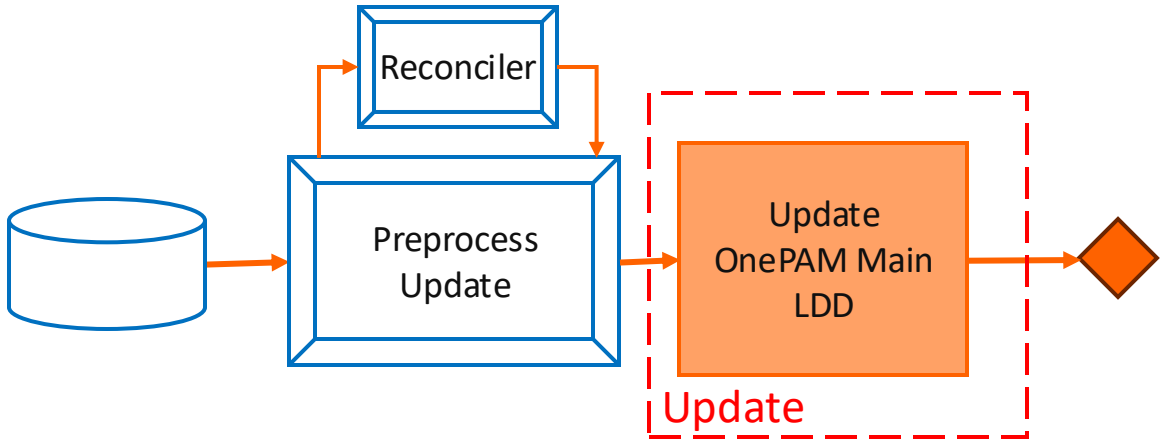


Table of Content

Phase 2: OnePAM Main LDD update



Phase 2: Update OnePAM Main LDD



2 activities:

1. **Reconciliations** – File based, assuming larger numbers
   - This is the comparison between OnePAM External and Main LDD using data in the UDL
   - There is no external ingested input file like for refreshes
2. **Daily Upload** of **Involved Parties** – API based, assuming smaller numbers
   - **Legal Entities & Operating Sites**
   - **Bankruptcies**
   - **Incapables**
   - **Legal representatives**

We start with reconciliations because that will be the equivalent mechanism of “initial load” for the Main LDD

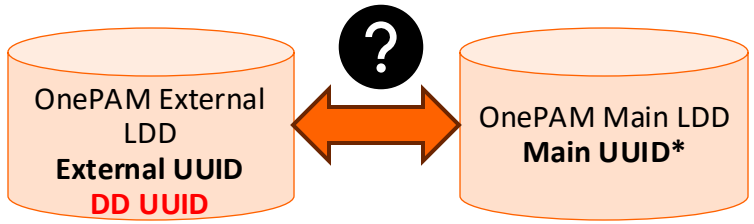
2 process steps:

1. **Preprocess External Updated**:
   - Check update permission configuration
   - Retrieve successful uploaded Parties*.
   - Get ‘before’ view to entity match with **Involved Parties** & identify *different* attributes.
   - Policy based decision making for attribute updates in Main LDD
2. **Update OnePAM Main LDD**:
   - **Update** *different* attributes in Core model (API)
   - Capture update results (API response)

* It still must be decided what the scope is we require to consider an upload succesful; a whole record incl relationships, or, individual attributes?

Reconciliation OnePAM: exploit DD UUID as common UUID

Step 1: Start situation

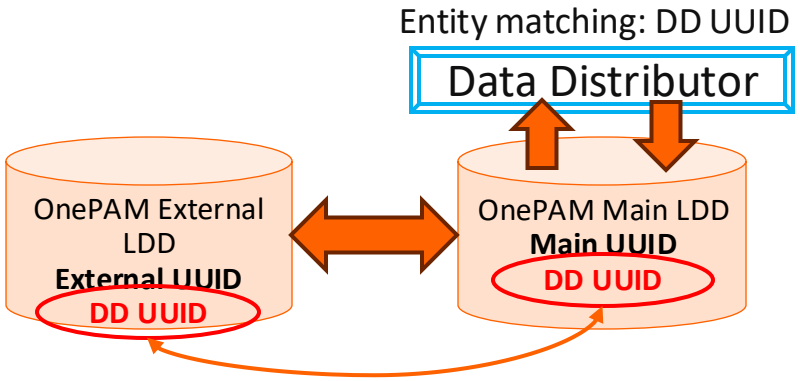


During reconciliation we must compare attributes between the **OnePAM Main LDD** and **OnePAM External LDD**

The challenge is that there is **no common key** of comparison between External & Main data stores.

We will use the **DD UUID** as a common key and the Data Distributor which has all the entity matching information for the **OnePAM External LDD**, which is what we want the Main LDD to compare with.

Step 2: add DD UUID to Main LDD extract



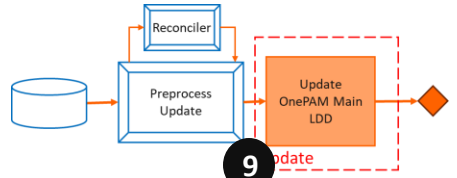
We extract the **OnePAM Main LDD** to the **Data Distributor** who does entity matching and adds the **DD UUID** to matching records. Unmatched records from the Main LDD are dropped as they are not in the External LDD.

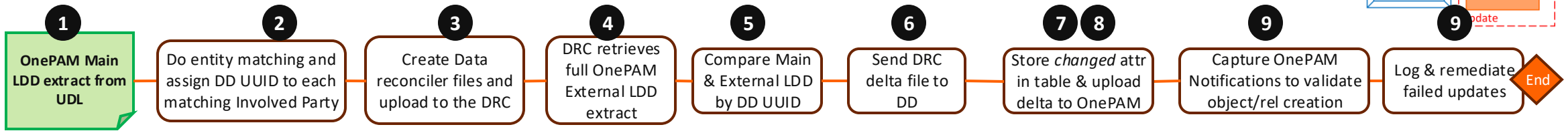
We now can match **External LDD Parties** with **Main LDD Involved Parties** through the **DD UUID** and compare the attributes the **OnePAM External LDD** is master for with those in **OnePAM Main LDD**. As the data model is the same, in principle no transformations should be necessary*.

Given the volumes involved we will use the data in the **Unite Data Lake (UDL)** to do the reconciliation rather than production data in OnePAM.

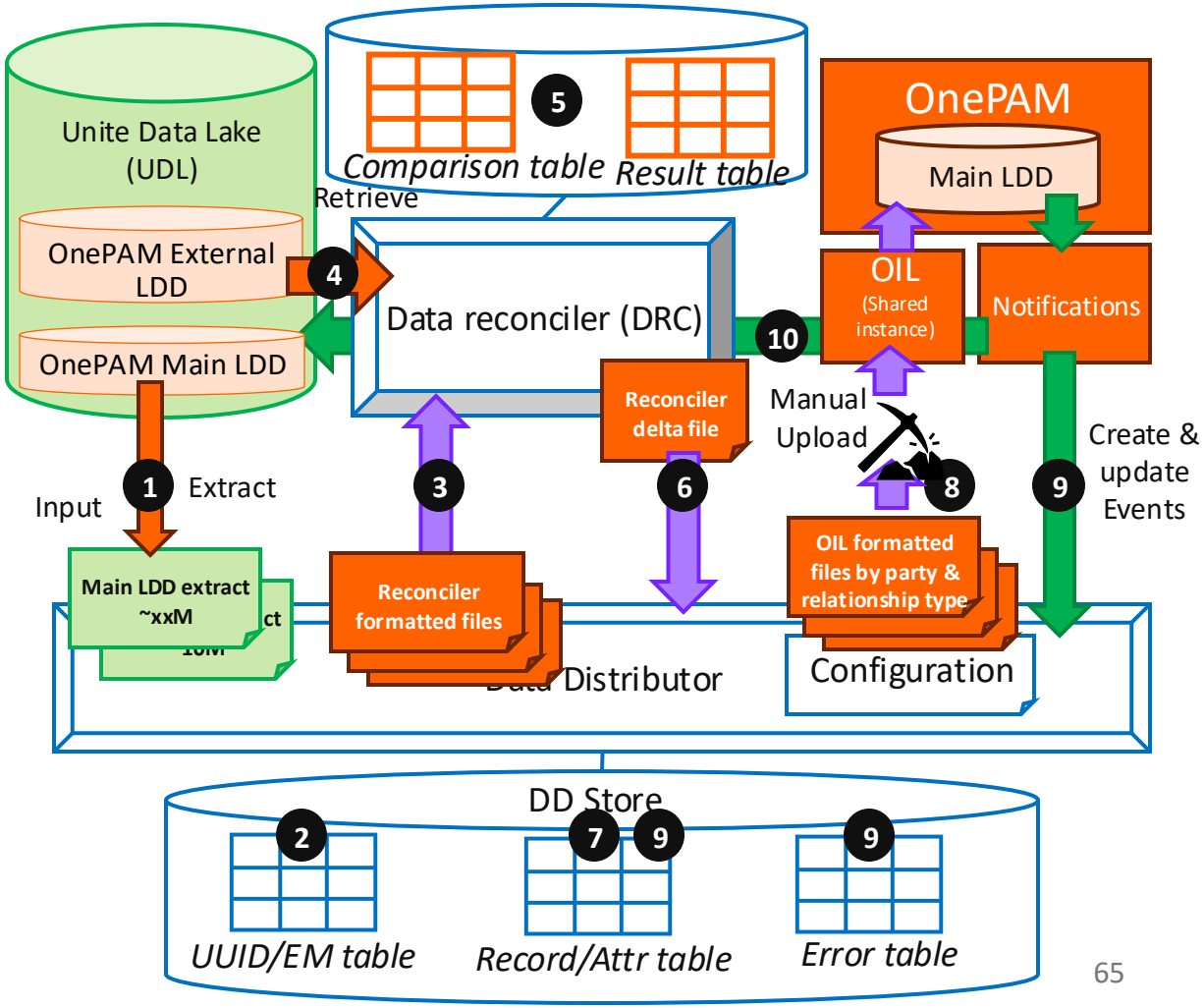
* There is/was a discussion wherefor Sole Traders could be differently modeled between Main and External LDD due to different ways they could be created in the Main LDD. This discussion must be resolved.

1. OnePAM "Main" LDD – Reconciliation pattern

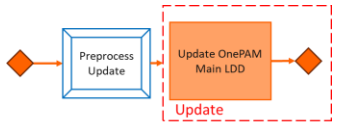


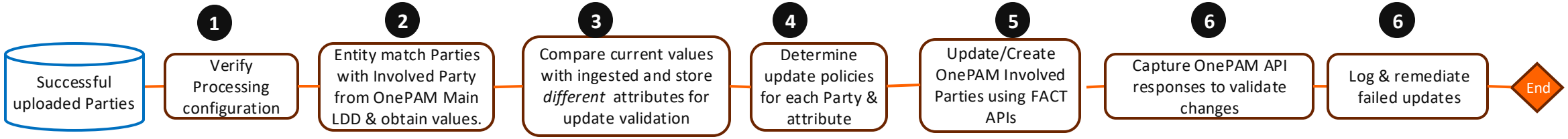


1. As input for the **Data Distributor,** we extract the **OnePAM Main LDD** data from **UDL** & send it to the **DD**.
2. The **Data Distributor** matches each record on already existing ingested records in the **DD Store** using demographics/LE # & to map them on a **DD UUID**, and, if necessary, disambiguates them.
3. **DD** creates new files with all OnePAM records and the **DD UUID** & uploads them to the **Data Reconciler (DRC)** component.
4. The **DRC** retrieves all Parties & relationships of the **OnePAM External LDD** from the **UDL**.
5. Both are stored in a *Comparison table* by **DD UUID** & compared with resulting delta records stored in a *Result table*.
6. **DRF** creates new files with deltas of **Involved Parties** & relationships and send it to the **DD**. These records are already **Main UUID** tagged.
7. We assume all attributes in the records received from the **DRC** have changed & store them in the *Record/Attr table* to account for updates in **OnePAM Main LDD**.
8. **DD** creates OIL formatted files and transfers them to OIL for **manual scheduling** of the upload into **OnePAM Main LDD**.
9. We **capture all update, creation & deletion events** from **OnePAM Main LDD** and update the *Error table* for the ones we don't receive.
10. All **Notifications** with created/changed **Involved Parties** & relationships are also picked up by the **UDL.**



2. OnePAM "Main" LDD - Daily upload pattern – API





1. We verify the **configuration** as to what sources & successful external ingested & uploaded Parties in the **External LDD** must be propagated.
2. The **Data Distributor** reads the **OnePAM Main LDD (API)** to match each Party with **Involved Parties** using **External UUID** or demographics/LE # & where necessary disambiguates them. It saves attributes in the *Record/Attr table.* For matching records (existing **Parties**), it obtains the **OnePAM Main UUID.**
3. **DD** compares matching ingested & uploaded Party attributes with Involved Party attributes and see where they are different. It also determines what new **Involved Parties** & relationships must be created.
4. The **DD** determines the update policy for each type of **Involved Party** attribute (overwrite if different, update if changed, don't overwrite…)
5. The **DD updates** changed attributes of existing **Involved Parties (API),** using the **Main UUID,** creating new Parties/relationships) & performs (soft) deletes.
6. The **DD** collects API responses to account for all created **Involved Parties** & updated attributes and updates the *Error table* for the ones where errors are thrown. Also deleted Parties/relationships are accounted for.
7. **MADP** will capture **Involved Party** creation **Notification** events to process into **Main Data Domains**. We don't have "Local" attrs., so no followup.
8. All **Notifications** with created & changed **Involved Parties** & relationships are also picked up by the **UDL** & create a copy of the **OnePAM Main LDD.**
9. The UDL also picks up **MADP** object creation events to create a copy.

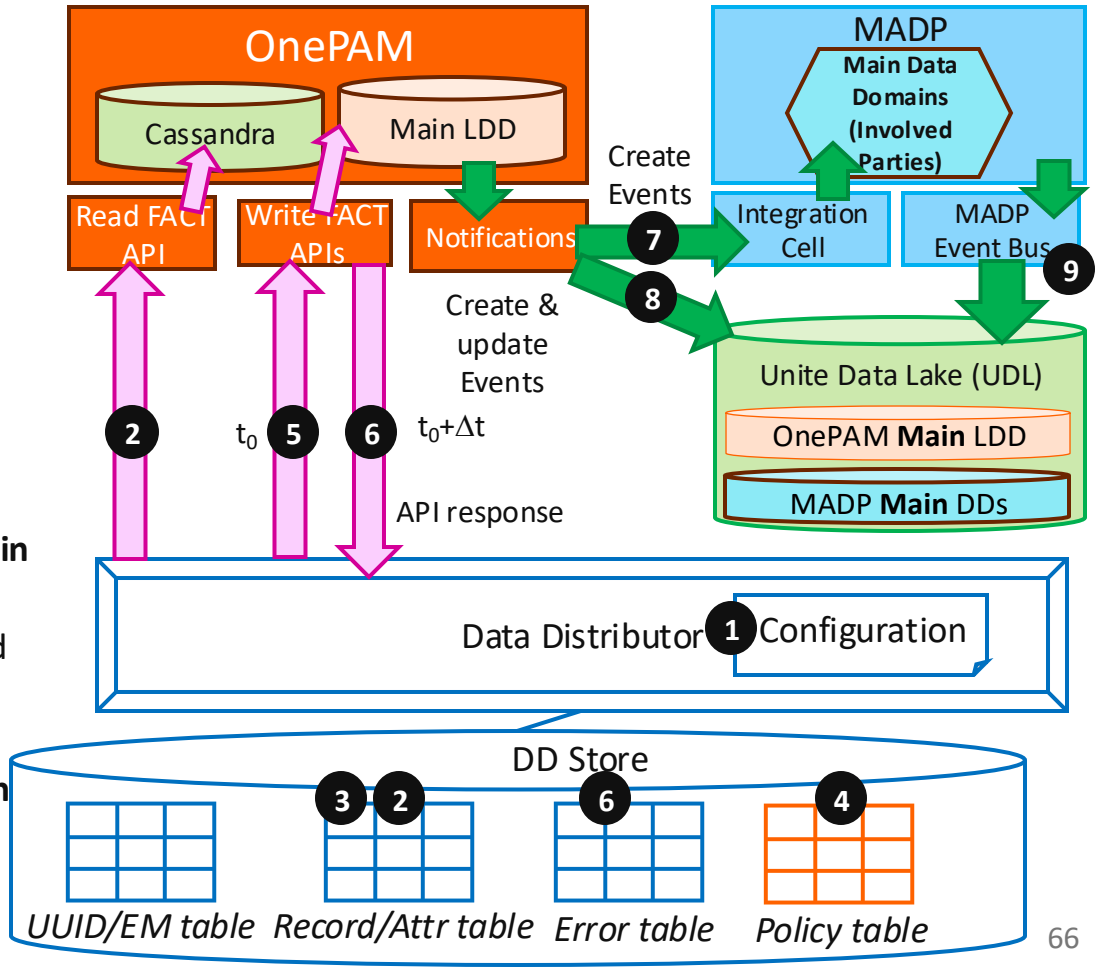
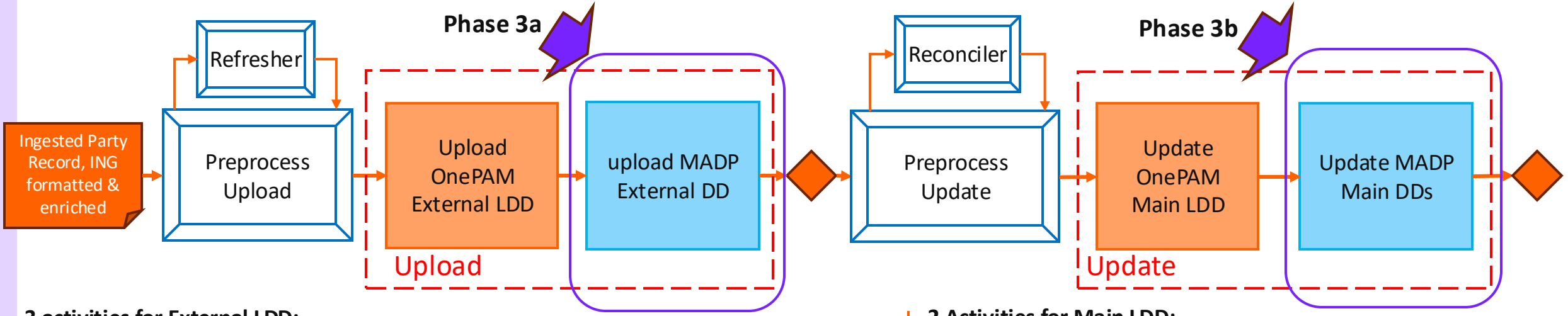


Table of Content

Phase 3: Add MADP Upload External DD & Update Main DDs



3 activities for External LDD:

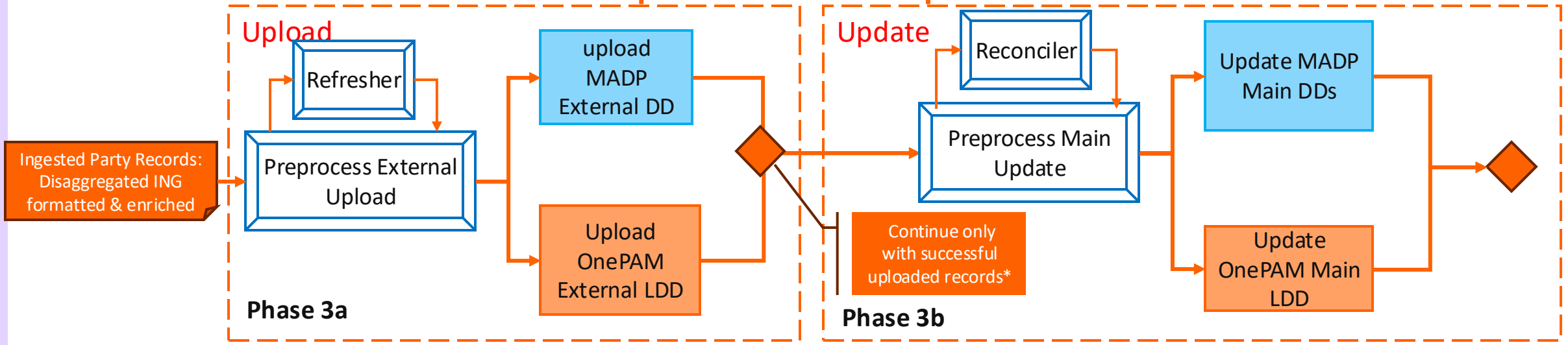
1. **Initial upload of "Local" attributes of Parties in MADP External Data Domain**
   - **As part of an initial upload of a new external ingested file** that happens to also have "Local" attributes. **We assume this scenario here.**
   - **As part of an evolution of an existing file now having "Local" attributes** interesting to ING but not fitting the OnePAM Core model. **We treat this as a refresh, not as an initial upload for MADP.**
   - We assume each additional file with "Local" attributes has distinctive "Local" attributes from every other file & hence generates MADP events.
2. **Daily Upload of "Local" attributes of Parties** – File or API based
   - Depending on number of records
3. **Refreshes** – UDL based
   - Similarly transformed to daily uploads by comparing the refresh ingested file with "Local" Attributes of the MADP External Data Domain

2 Activities for Main LDD:

1. **Daily Upload of "Local" attributes of Involved Parties in MADP Main Data Domains** – API based, assuming smaller numbers
2. **Reconciliations** – File based, assuming larger numbers
   - This is the comparison between MADP External and Main Data Domains using the UDL
   - There is no external ingested input file like for refreshes

The nested serial order of updates (External/Main & OnePAM/MADP) leads to prolongued throughput times we seek to reduce.

Phase 3: Parallelization of uploads and updates.



The Landing zones ("External") must be uploaded before the Main zones are updated as they are Master Data for the Main zones

We update the Main zones only with successful External uploaded data to ensure External data mastership

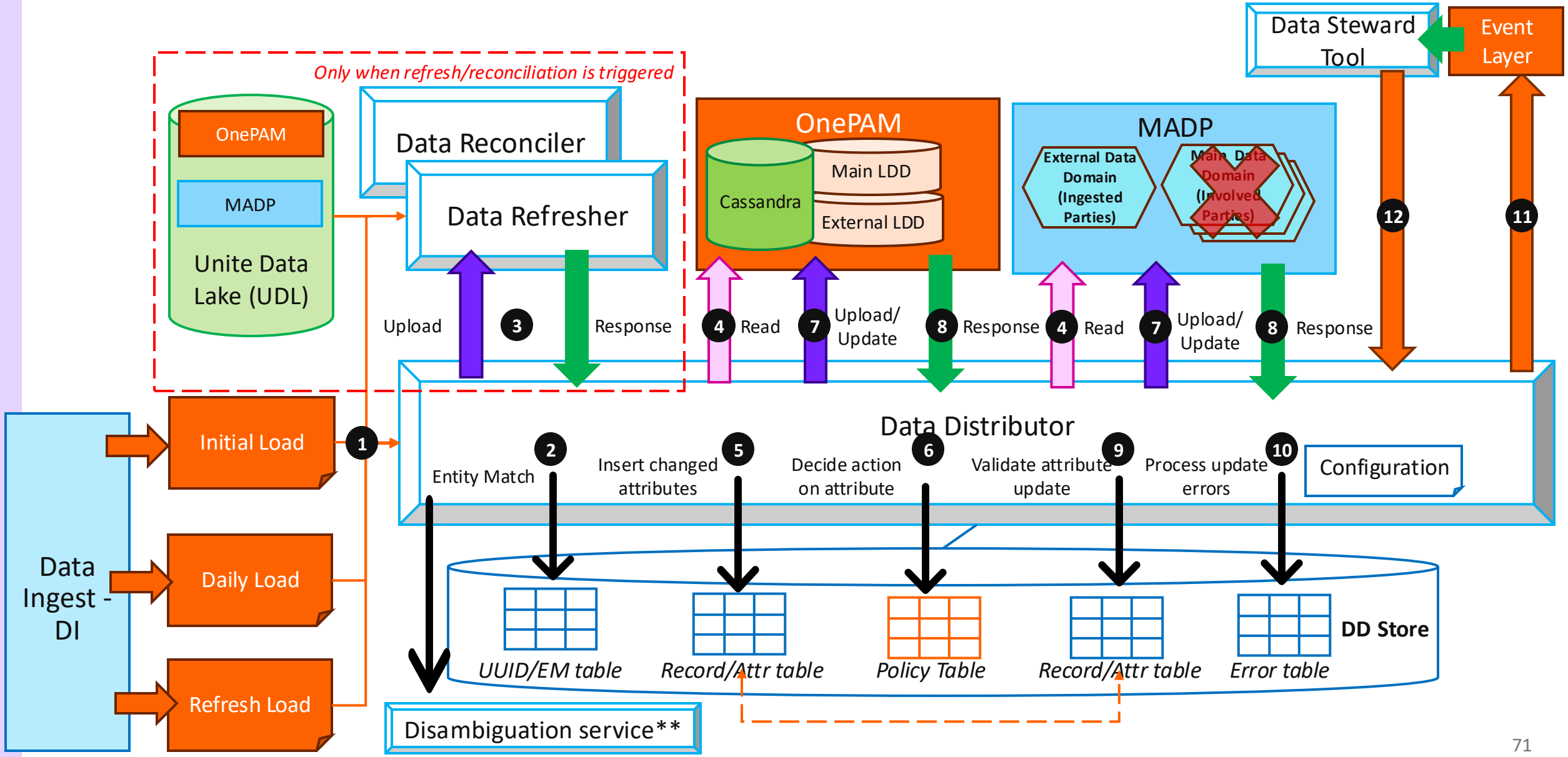
We update the OnePAM and MADP zones in parallel to reduce throughput times

- **MADP**, as a sidecar to OnePAM, requires a **OnePAM External or Main UUID** to be present when uploading "Local" attribute records.
  - **MADP** captures Notifications of new **Parties** created in **OnePAM** and creates an associated object.
  - For ingested new **Parties**, that means by default that they first must be created in **OnePAM**, their UUIDs captured by the **Data Distributor** and be added to the **MADP** uploads/updates.
- **OnePAM** has a feature that allows to create **OnePAM UUIDs** outside **OnePAM** (standard V4 UUIDs), but this must be configured. Today it's only used for the Australian deployment.
  - It's available for **OIL** and a couple of Fact API endpoints (used by Australia), not all of them.
  - **GCDM wants this to be a formal feature request and enabled for all endpoints**. This will take time
- Being able to create UUIDs outside **OnePAM** would allow us to create the UUIDs by the **Data Distributor** before the (Involved) **Parties** are created in **OnePAM** and allow to upload/update **MADP** "Local" attributes in parallel with **OnePAM***. This is mostly beneficial for **batch-oriented file-based uploads**.

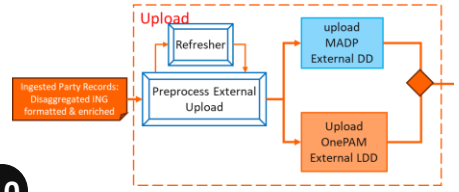
* Race conditions, where a new Party is not yet created in MADP while a record with "Local" attributes is uploaded by File toMADP would be temporary cached by MADP. API calls would give an error.

Table of Content

Phase 3a: MADP External DD upload



1. MADP "External" LDD - Initial upload pattern (1/2)



2 Initial Load Ingested Party Records (xM): Disaggregated ING formatted & enriched

3 Read & do entity matching in DD Store & obtain existing External UUIDS

4 Create new Party DD UUID AND OnePAM External UUID & store Entity matching info in DD Store

6 Create OnePAM OIL files & create MADP files

6 Upload OnePAM files for upload & processing

7 When the first Notification is captured from OnePAM we use the automated upload facility for the MADP file

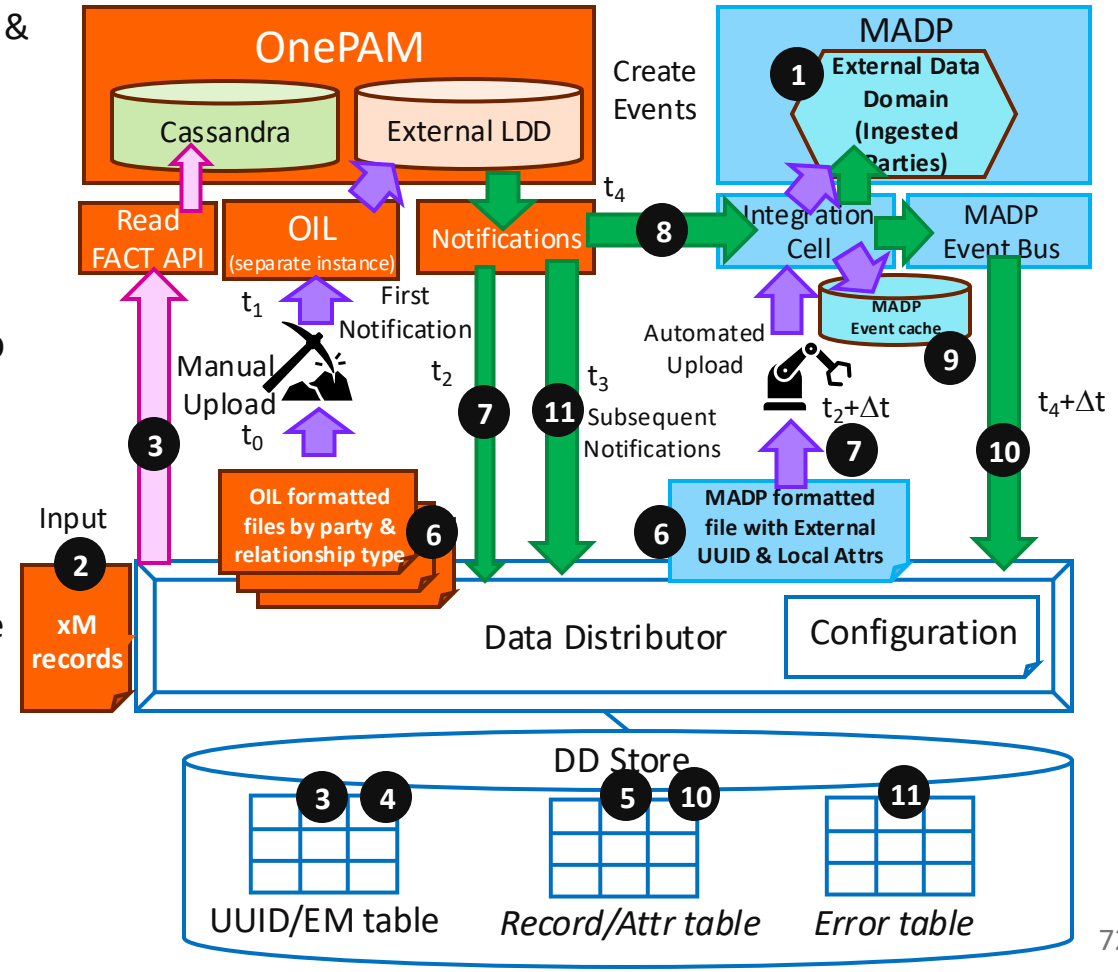
9 Attributes of not yet MADP processed new Party Notifications are cached.

10 We capture OnePAM Notifications and MADP events to validate changes

11 Log & remediate failed updates

End

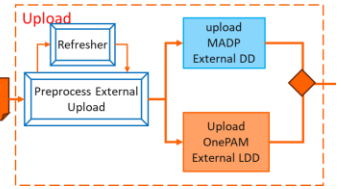
1. Before the first upload happens, the **MADP External Data Domain** gets created & populated with all **OnePAM External UUIDs** by the **MADP** team.
2. The initial load of a new source is an exceptional event due to
   - The large number of records to upload
   - The **OnePAM External LDD** is not empty anymore.
   - The **MADP Data Distributor Store** is treated as empty.
3. The **Data Distributor** matches each record on already existing records in the **DD Store** using demographics/LE # & if necessary, disambiguates* them. For matching records (existing **Parties**), it obtains the **External UUID** using the **DD UUID** as search criterium*.
4. It creates both **DD UUIDs & OnePAM External UUIDs**** for new Parties.
5. We keep a **temporary record/attribute table** to tabulate upload results, and we indicate for all records whether create/update & whether to expect a **OnePAM Notification** or **MADP event** for the upload.
6. The **DD creates the OIL files** & **transfers them** for **manual scheduling** of the upload into **OnePAM External LDD** by **OIL**. It also creates the **MADP** files with **OnePAM External UUID** embedded but doesn't upload them yet.



* Differences in flow by adding MADP is given in red.

**We need the ability to create OnePAM UUIDs outside OnePAM to enable parallel uploads to MADP which requires OnePAM UUIDs for all uploaded records.

1. MADP "External" LDD - Initial upload pattern (2/2)



2 Initial Load Ingested Party Records (xM): Disaggregated ING formatted & enriched

3 Read & do entity matching in DD Store & obtain existing External UUIDS

4 Create new Party DD UUID AND OnePAM External UUID & store Entity matching info in DD Store

6 Create OnePAM OIL files & create MADP files

6 Upload OnePAM files for upload & processing

7 When the first Notification is captured from OnePAM we use the automated upload facility for the MADP file

9 Attributes of not yet MADP processed new Party Notifications are cached.

10 We capture OnePAM Notifications and MADP events to validate changes

11 Log & remediate failed updates

End

7. The first **Notification** tells **DD** that **OnePAM** processing started & triggers it to upload the **MADP** file (with External UUID) for automated processing.

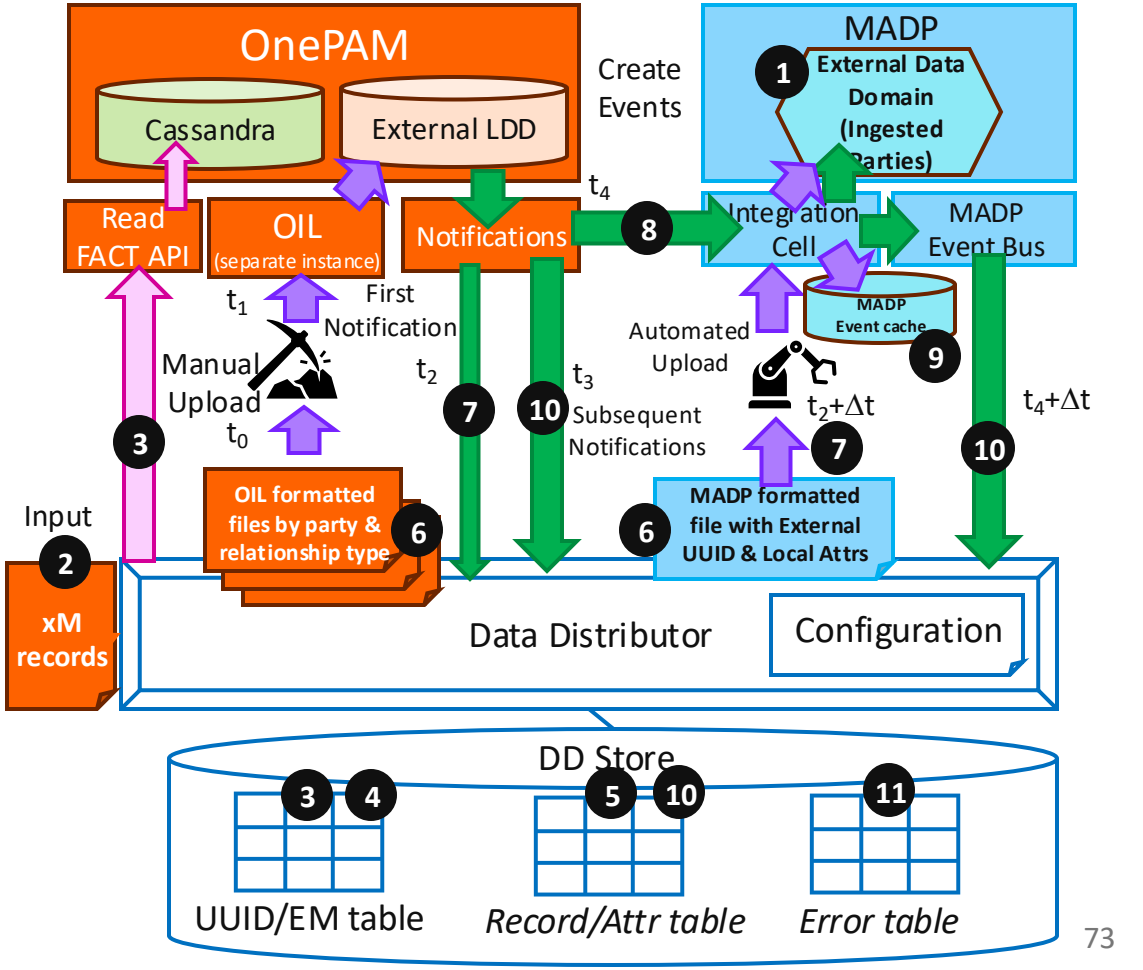
8. New **OnePAM Party** creation **Notifications** trigger **MADP** to create the object with **External UUID** in the **MADP External Data Domain**.

9. If records with new **OnePAM External UUIDs** arrive before the creation of the object in the **MADP External Data Domain**, it will be cached in the **MADP Event Cache** and retried by **MADP** after the object is created.

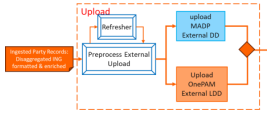
10. Data Distributor collects all **OnePAM Notifications** and **MADP creation events** and validates it with information in the *Record/Attr table*.

11. We update the *Error table* for those records for which we don't receive **OnePAM Notifications and MADP events**.

Not shown on the drawing, but all **OnePAM Notifications** and **MADP Events** are also captured and stored in the **UDL**.



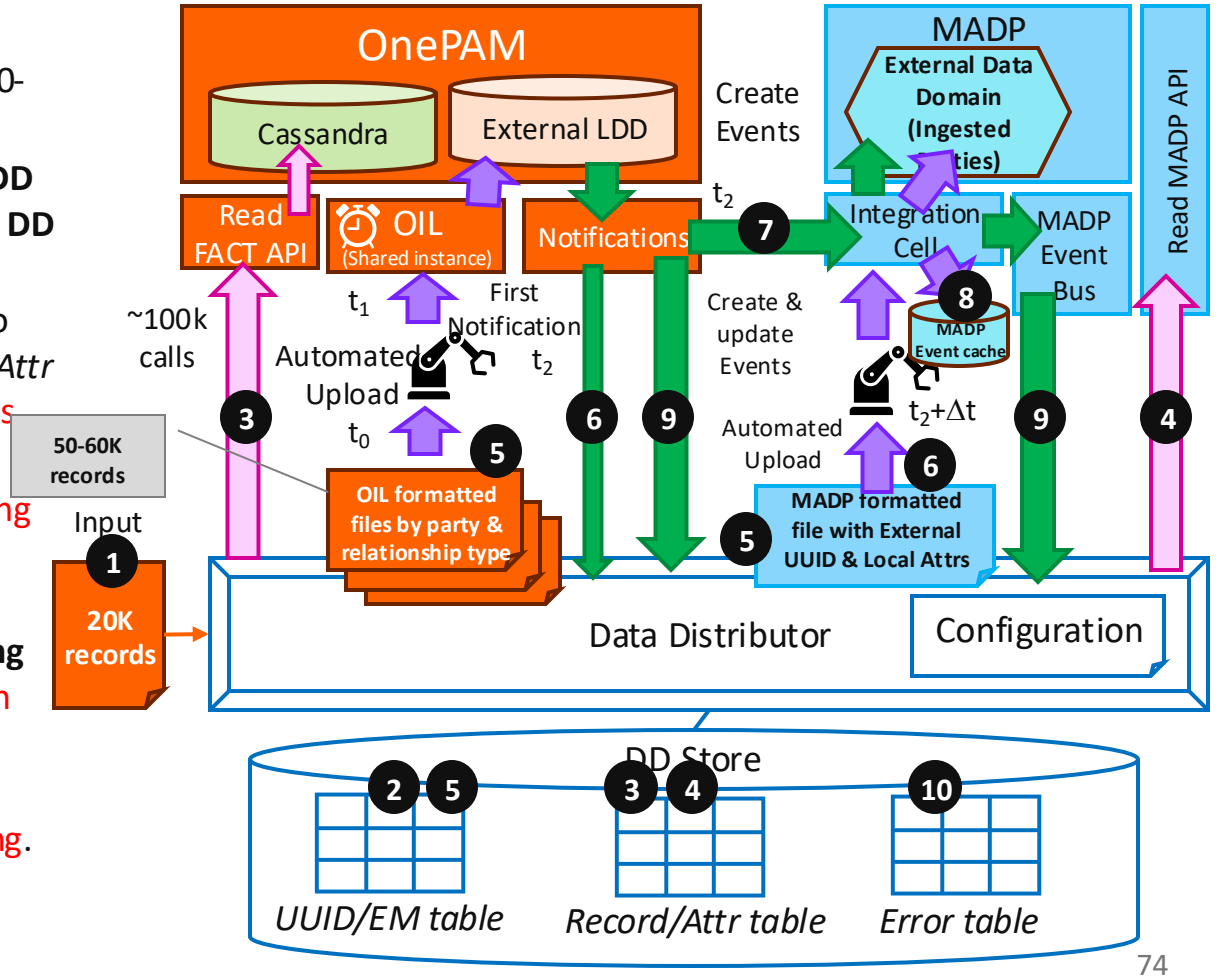
2a. MADP "External" LDD - Daily upload pattern – File (1/2)



1. Daily update disaggregated records (10,000s)
2. Read & do entity matching in DD Store
2. Create new Party DD UUID AND OnePAM External UUID & store Entity matching info in DD Store
3. 4. Read current Party values from OnePAM External LDD & MADP External DDD
3. 4. Compare current values with ingested and store *changed* attributes for upload validation
5. Create OIL files & upload them for automated processing
6. When the first Notification is captured from OnePAM we use the automated upload facility for the MADP file
8. Attributes of not yet MADP processed new Party Notifications are cached.
9. We capture OnePAM Notifications and MADP events to validate changes
10. Log & remediate failed updates

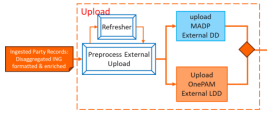
End

1. The daily upload contains around 20k disaggregated records which become 50-60k actually uploadable records (Parties & relationships)
2. The **Data Distributor** matches each record on already existing records in the **DD Store** using demographics/LE # & if necessary, disambiguates them. It creates **DD UUIDs & OnePAM External UUIDs*** for new Parties.
3. The **DD** reads all existing ingested Parties from **OnePAM** using the DD UUID to compare the attributes for changes & stores the changed ones in the *Record/Attr table* to account for updates. For matching records (existing **Parties**), it obtains the **OnePAM External UUID**.
4. The **DD** reads all existing ingested "Local" attributes from MADP read API, using the earlier read **OnePAM External UUID** & stores the changed ones in the *Record/Attr table* to account for updates.
5. The **DD creates the OIL files** & **transfers them to OIL** for **automated scheduling** of the upload into **OnePAM External LDD**. . It also creates the **MADP** files with **OnePAM External UUID** embedded but doesn't upload them yet.
6. The first **Notification** tells **DD** that **OnePAM** processing started & triggers it to upload the **MADP** file (with **OnePAM External UUID**) for automated processing.



* Differences in flow by adding MADP is given in red.

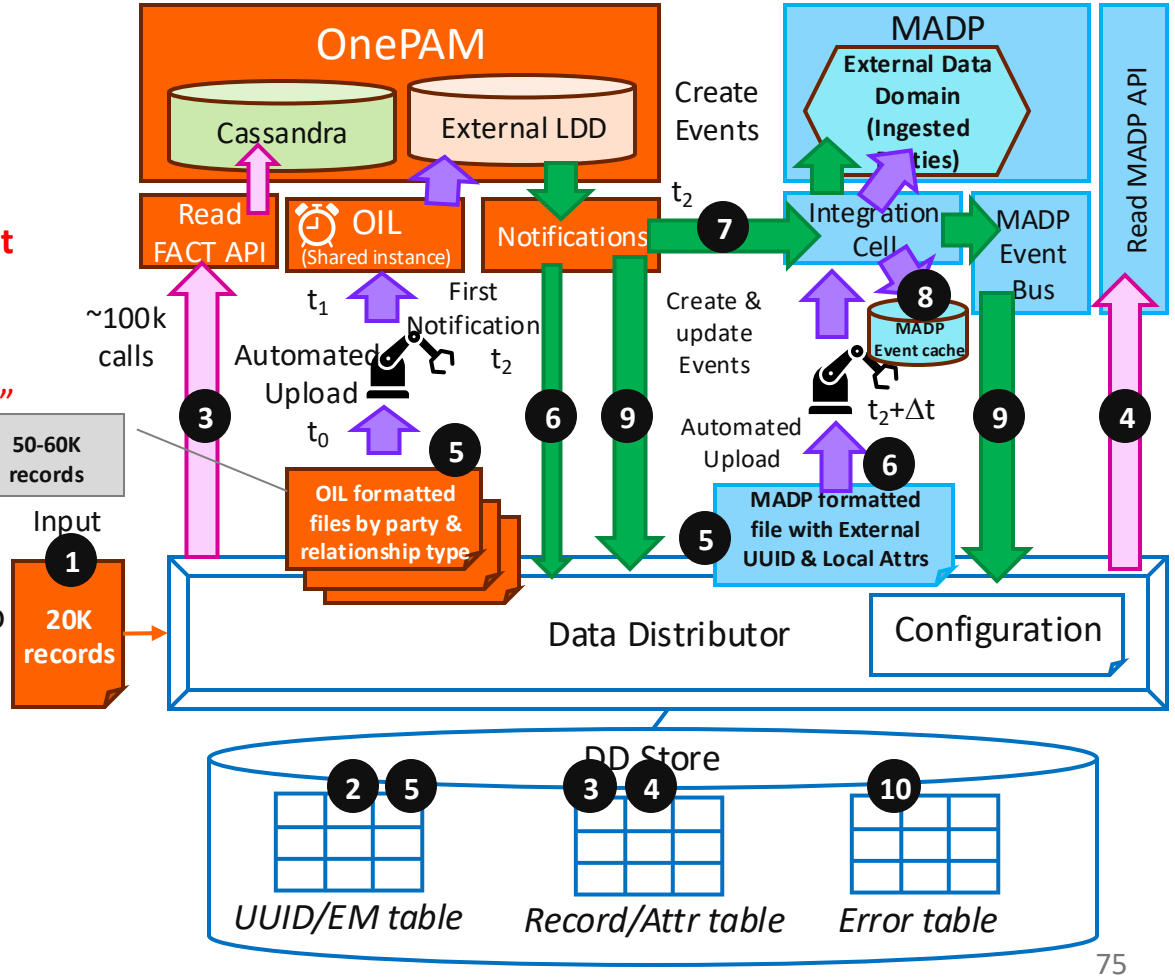
2a. MADP "External" LDD - Daily upload pattern – File (2/2)



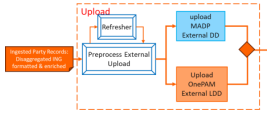
1. Daily update disaggregated records (10,000s)
2. Read & do entity matching in DD Store
2. Create new Party DD UUID AND OnePAM External UUID & store Entity matching info in DD Store
3 4. Read current Party values from OnePAM External LDD & MADP External DDD
3 4. Compare current values with ingested and store *changed* attributes for upload validation
5. Create OIL files & upload them for automated processing
6. When the first Notification is captured from OnePAM we use the automated upload facility for the MADP file
8. Attributes of not yet MADP processed new Party Notifications are cached.
9. We capture OnePAM Notifications and MADP events to validate changes
10. Log & remediate failed updates
End

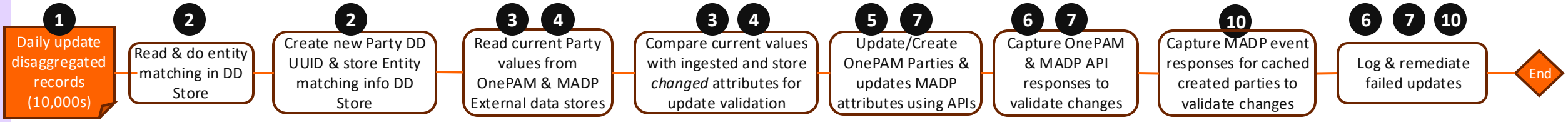
7. New **OnePAM Party** creation **Notifications** trigger **MADP** to create the object with **External UUID** in the **MADP External Data Domain**.
8. If records with new **OnePAM External UUIDs** arrive before the creation of the object in the **MADP External Data Domain**, it will be cached in the **MADP Event Cache** and retried by **MADP** after the object is created.
9. We **capture all update, creation & deletion events** from **OnePAM & MADP** to account for all created Parties/relationships & updated **Party** "Core" and "Local" attributes with the *Record/Attr table*.
10. We update the *Error table* for the ones we don't receive. Also deleted Parties & relationships are accounted for.

Not shown on the drawing, but all **OnePAM Notifications** and **MADP Events** are also captured and stored in the **UDL**.

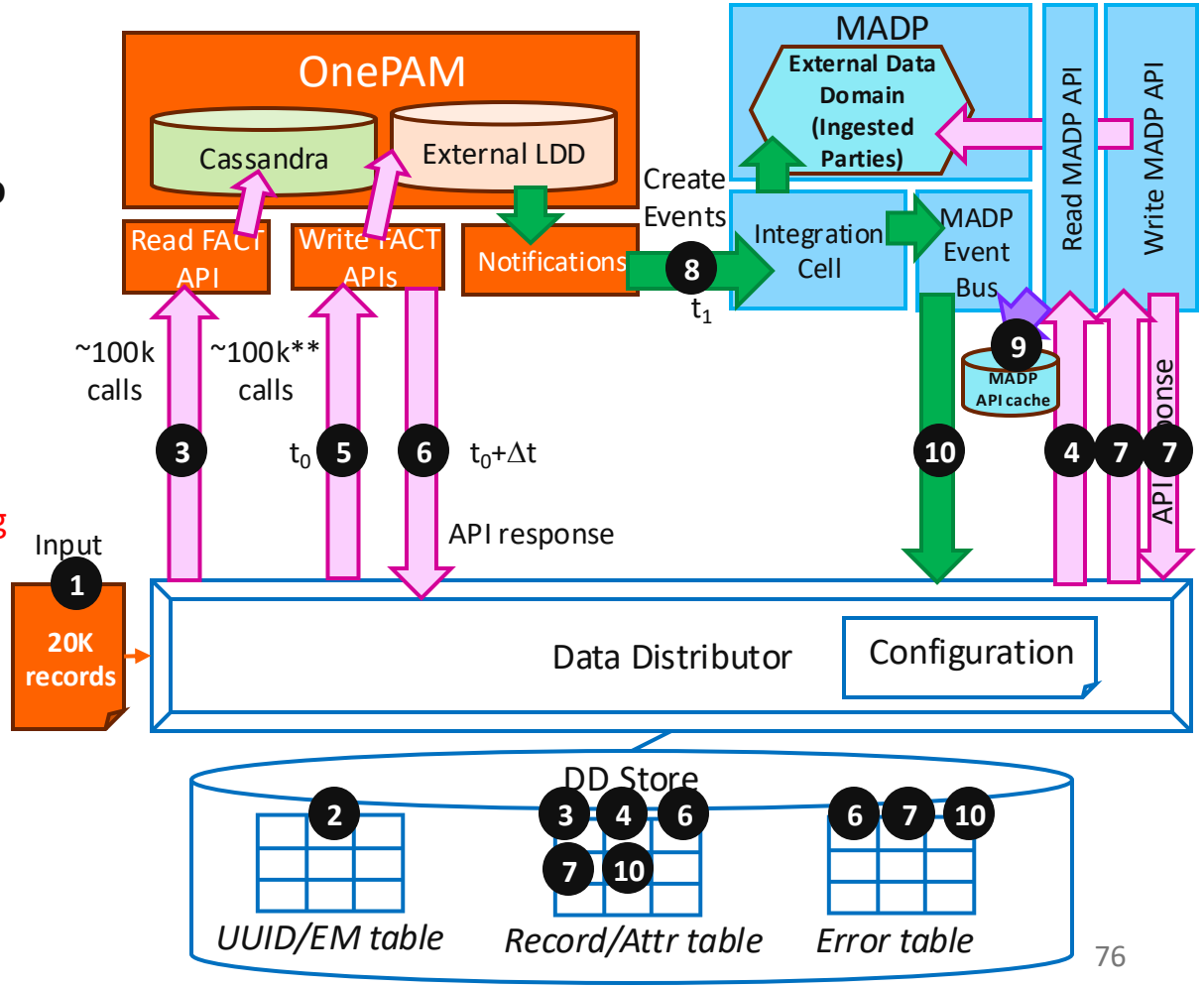


2b. MADP "External" LDD - Daily upload pattern – API (1/2)





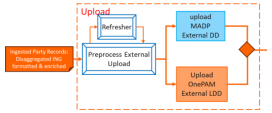
1. The daily upload contains around 20k disaggregated records which become 50-60k actually uploadable records (Parties & relationships)
2. The **Data Distributor** matches each record on already existing records in the **DD Store** using demographics/LE # &where necessary disambiguates them. It creates **DD UUIDs** for new Parties.
3. The **DD** reads all existing ingested Parties from **OnePAM** using the **DD UUID** to compare the attributes for changes & stores the changed ones in the *Record/Attr table* to account for updates. For matching records (existing **Parties**), it obtains the **OnePAM External UUID***.
4. The **DD** reads all existing ingested "Local" attributes from **MADP** read API, using the earlier read **OnePAM External UUID** & stores the changed ones in the *Record/Attr table* to account for updates.
5. The **DD updates** changed Core attributes of existing Parties in **OnePAM** & creates new parties & relationships as well as performs (soft) deletes using APIs*
6. The **DD** collects API responses to account for all created Parties (including the new **OnePAM External UUIDs**) & updated **Core Party** attributes & updates the *Error table* for the ones where errors are thrown. Also deleted Parties/relationships are accounted for.

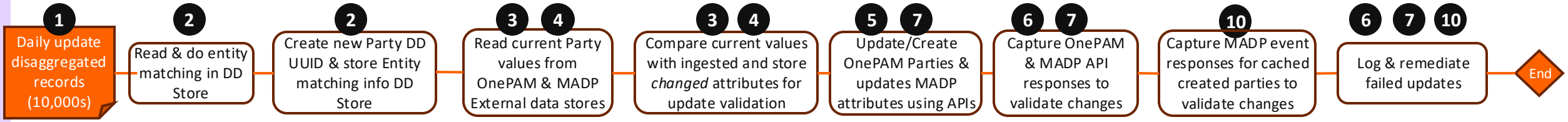


* Differences in flow by adding MADP is given in red.

** we assume on average >1 API call per updated Party/relationship.

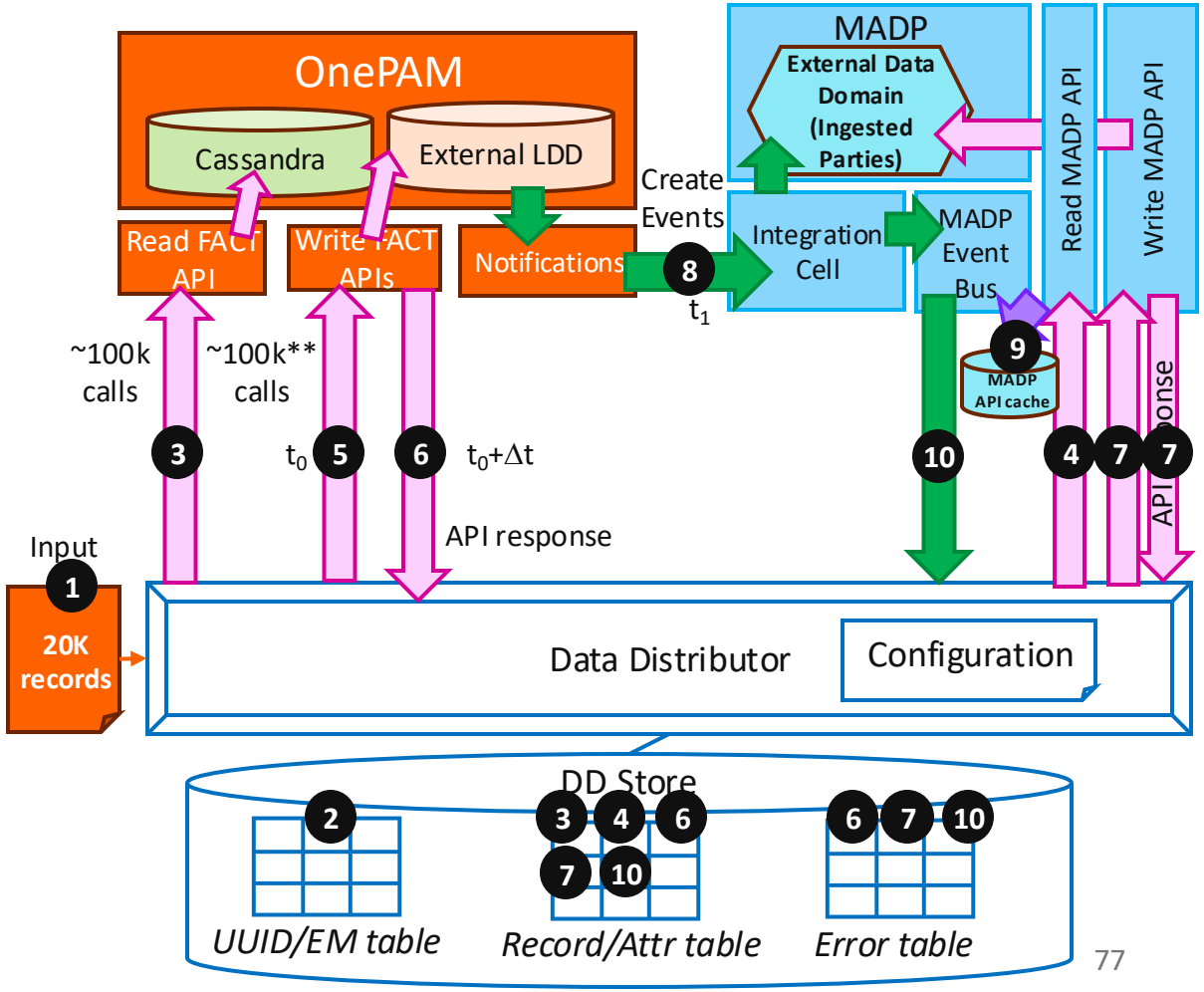
2b. MADP "External" LDD - Daily upload pattern – API (2/2)





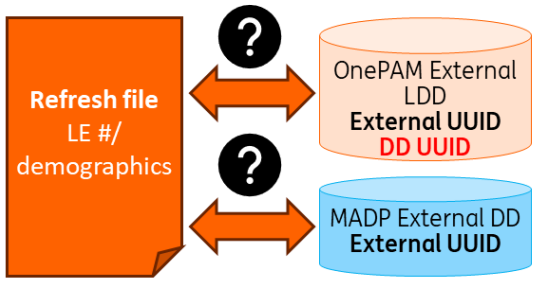
7. The **DD** updates changed "Local" attributes of Parties in **MADP** using write APIs, and collects responses, updating the *Record/Attr table* to account for updates & updates the *Error table* for the ones where errors are thrown
8. New **OnePAM Party** creation **Notifications** trigger **MADP** to create the object with the **OnePAM External UUID** in the **MADP External Data Domain.**
9. For new created Parties with **External UUIDs** which have not yet been processed by **MADP**, API updates will be cached in the **MADP API Cache.**
10. **DD** captures the **MADP** cached creation events containing the new **OnePAM External UUIDs** with the updated attributes & updates the *Record/Attr table* to account for updates & updates the *Error table* for the ones where errors are thrown or expected events never arrive.

Not shown on the drawing, but all **OnePAM Notifications** and **MADP Events** are also captured and stored in the **UDL**.



Refresh add MADP External DD : Exploit DD UUID as common UUID

Step 1: Start situation

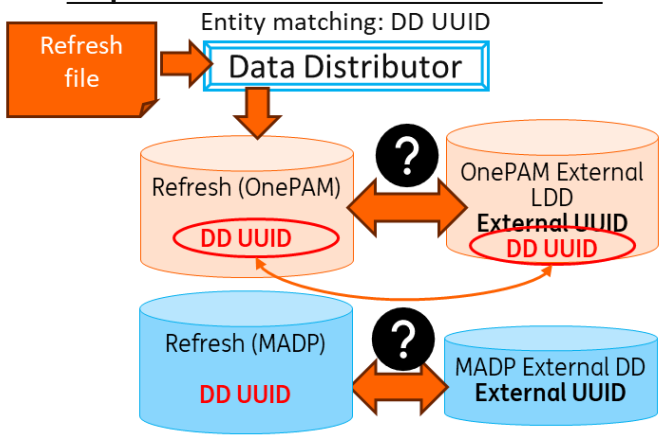


During refresh we must compare attributes between the:

- **OnePAM External LDD** & **MADP External DD** and
- the Parties & relationships in the refresh file

The problem is that there is **no common key** of comparison between External Zones & records in the refresh file.

Step 2: add DD UUID to OnePAM refresh

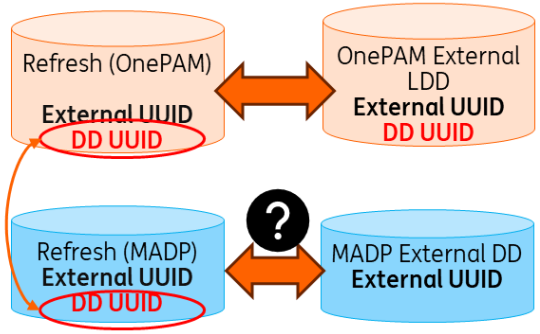


We use the **DD UUID** as a common key and the **Data Distributor** which has all the entity matching information for the **OnePAM External LDD.**

We load the refresh file to the **DD** doing entity matching, adding the **DD UUID** to matching records. New Parties from the refresh file create new **DD UUIDs**.

We split* the refresh file in **OnePAM** and **MADP** parts & can compare the **OnePAM External LDD** with the **OnePAM** refresh part, using the **DD UUID** to enrich it with the **External UUID**.

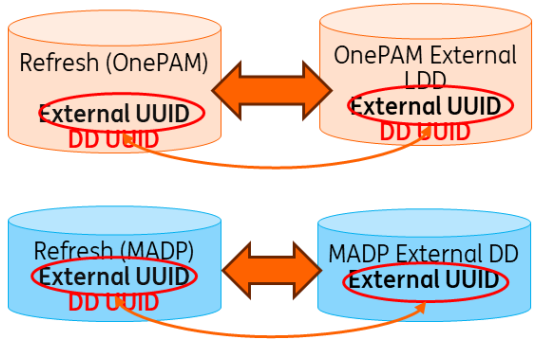
Step 3: enrich MADP Refresh with External UUID



To support comparing the **MADP** refresh & **External DD** parts, we have enriched the **OnePAM** refresh part with the **External UUID** from the **OnePAM External LDD**.

Then we compare the **DD UUIDs** of the **OnePAM** and **MADP** refresh parts and enrich the **MADP** refresh part with the **External UUID**.

Step 4: compare using matching UUIDs

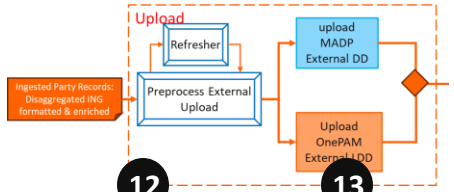


We can now compare the **Party** Core attributes and relationships of the **OnePAM External LDD** with the **OnePAM** refresh using either **DD UUID** or **External UUID** as key.

We can also now compare the "Local" attributes of the **MADP External DD** with the **MADP** refresh using the **External UUID** as key

* We split up the OnePAM & MADP refresh parts in the Data Distributor which has all the context information. This is before matching wither External UUIDs, which happens in the Refresher. We don't want the Refresher to "know" the difference between Core and Local attributes for code duplication avoidance. It just compares different sources.

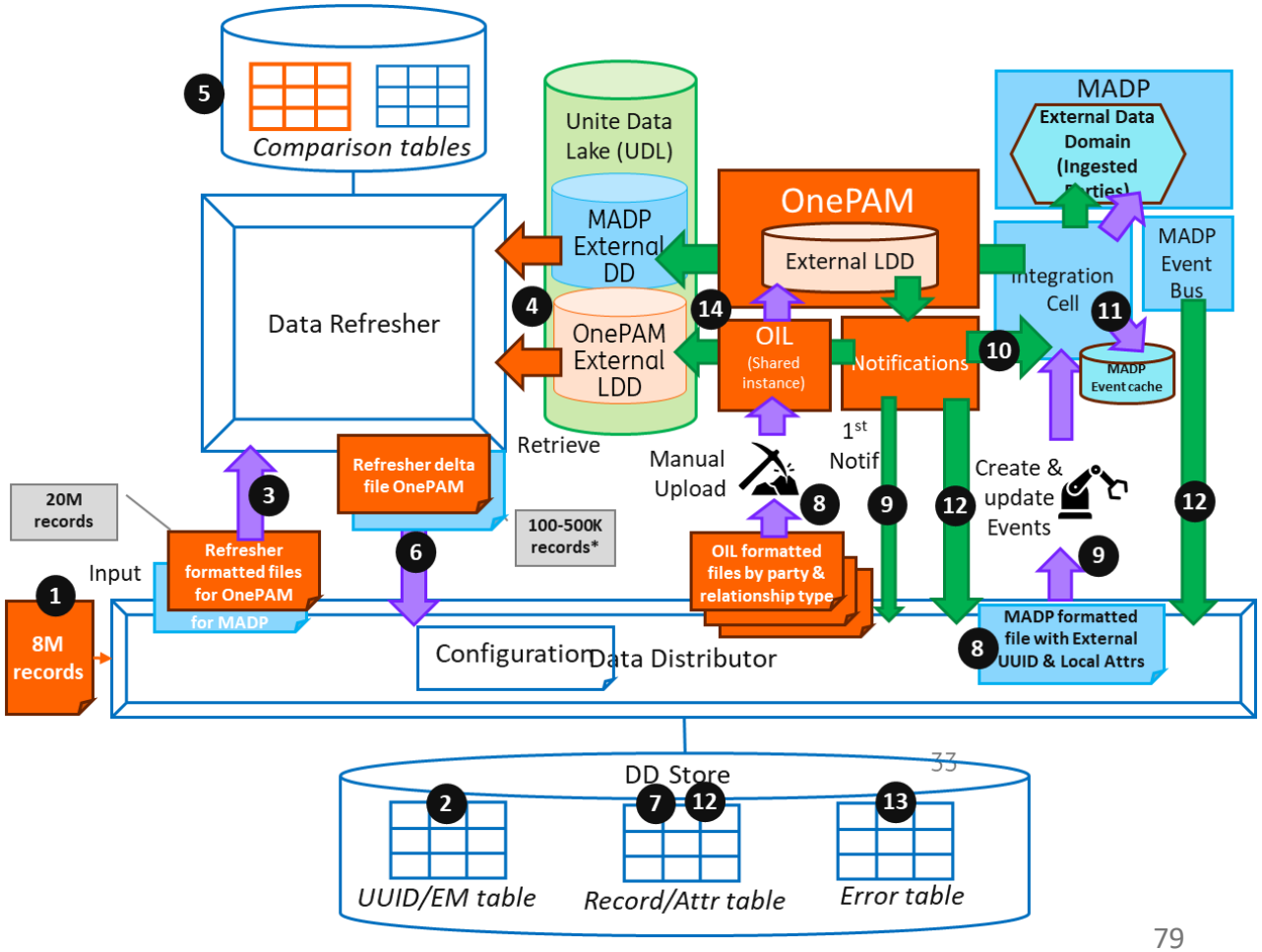
3. MADP "External" DD – Refresh pattern (1/2)



1. Refresh data
2. Read & do entity matching in DD Store & create new Party DD UUID
3. Create DRF files & upload them for processing.
4. Retrieve OnePAM External LDD & MADP External DD dump from UDL
5. Compare UDL data with refresh data
6. Send refresher deltas back to DD
7. 8. Store *changed* attr in table & upload delta to OnePAM
9. When the first Notification is captured from OnePAM we use the automated upload facility for the MADP file
11. Attributes of not yet MADP processed new Party Notifications are cached.
12. We capture OnePAM Notifications and MADP events to validate changes
13. Log & remediate failed updates

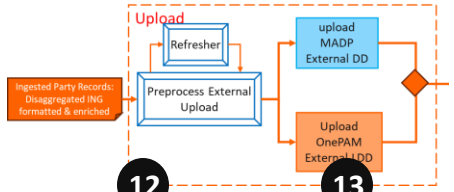
End

1. The refresh contains around 8M disaggregated records which become ~15-20M uploadable records (Parties & rels.).
2. The **Data Distributor** matches each record on already existing records in the **DD Store** using demographics/LE # &, if necessary, disambiguates them. It creates **DD UUIDs** for new Parties.
3. **DD** creates files with all refresh records and the **DD UUID, split by OnePAM & MADP attributes*** & uploads them to the **Data Refresher (DRF)** component.
4. The **DRF** retrieves all **Parties** & relationships of the **OnePAM External LDD** and the **MADP External DD** from the **UDL**.
5. All are stored in comparison tables. The **DRF** does **DD UUID** to **OnePAM External UUID** Mapping for linking with **MADP** records and compares differences between the refresh files and **OnePAM External LDD** & **MADP External DD UDL** extracts.
6. **DRF** creates new files with deltas or Parties & relationships and send it to the **DD**. These records are already **DD UUID** tagged and contain the **OnePAM External UUIDs** to support **MADP** uploads.
7. We assume all attributes in the records have changed & store them in the *Record/Attr table* to account for updates in **OnePAM & MADP.**



* Differences in flow by adding MADP is given in red

3. MADP "External" DD – Refresh pattern (2/2)



1. Refresh data
2. Read & do entity matching in DD Store & create new Party DD UUID
3. Create DRF files & upload them for processing.
4. Retrieve OnePAM External LDD & MADP External DD dump from UDL
5. Compare UDL data with refresh data
6. Send refresher deltas back to DD
7. 8. Store *changed* attr in table & upload delta to OnePAM
9. When the first Notification is captured from OnePAM we use the automated upload facility for the MADP file
11. Attributes of not yet MADP processed new Party Notifications are cached.
12. We capture OnePAM Notifications and MADP events to validate changes
13. Log & remediate failed updates

End

8. **DD** creates OIL formatted files and transfers them to OIL for **manual scheduling** of the upload into **OnePAM External LDD**. It also creates the **MADP** files with **OnePAM External UUID** embedded but doesn't upload them yet.
9. The first **Notification** tells **DD** that **OnePAM** processing started & triggers it to upload the **MADP** file (with **OnePAM External UUID**) for automated processing.
10. New **OnePAM Party** creation **Notifications** trigger **MADP** to create the object with **External UUID** in the **MADP External Data Domain**.
11. If records with new **OnePAM External UUIDs** arrive before the creation of the object in the **MADP External Data Domain**, it will be cached in the **MADP Event Cache** and retried by **MADP** after the object is created.
12. We **capture all update, creation & deletion events** from **OnePAM & MADP** to account for all created Parties/relationships & updated **Party** "Core" and "Local" attributes with the *Record/Attr table*.
13. We update the *Error table* for the ones we don't receive. Also deleted Parties & relationships are accounted for.
14. All **OnePAM Notifications** and **MADP Events** are also captured and stored in the **UDL**.

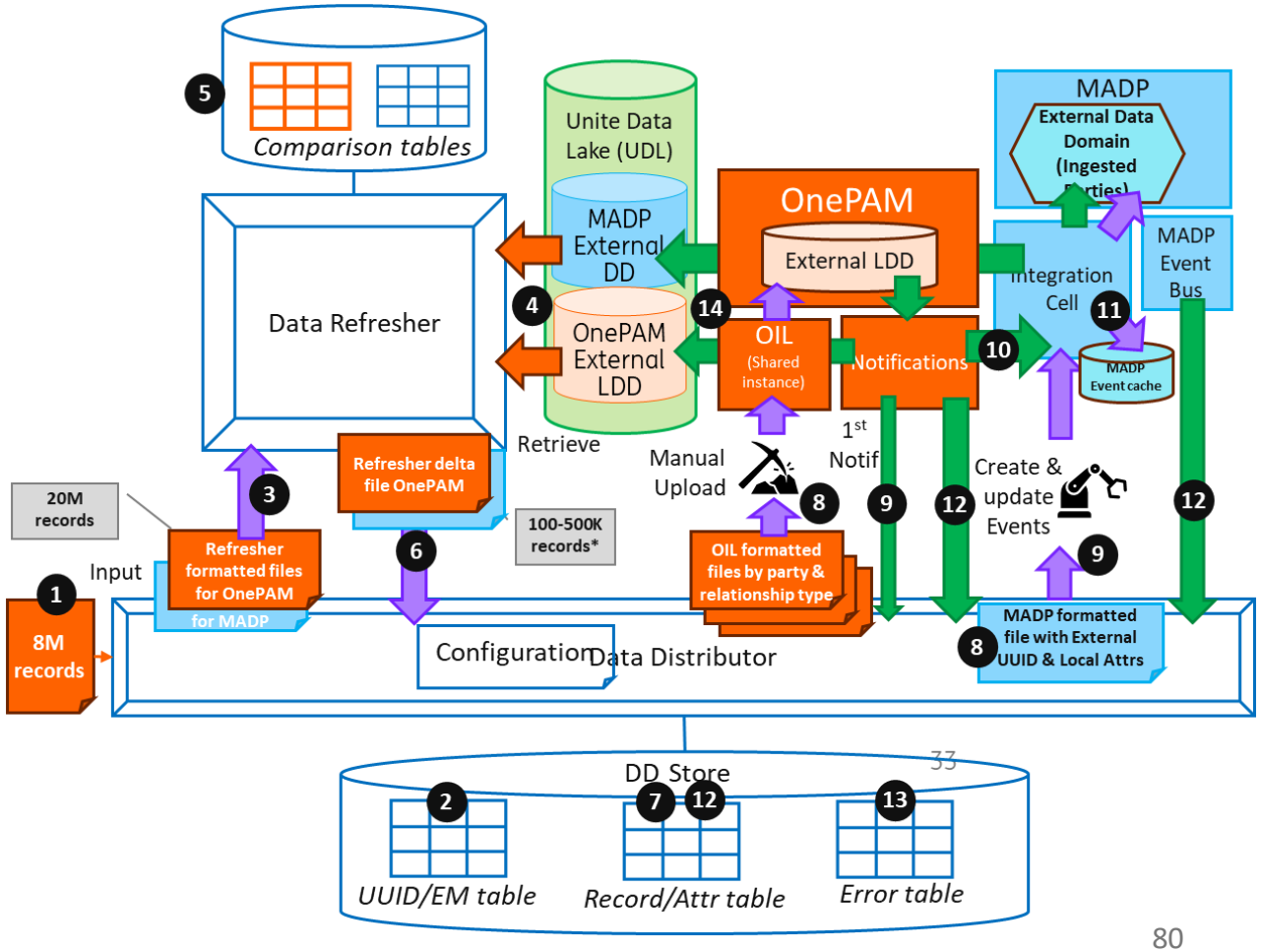
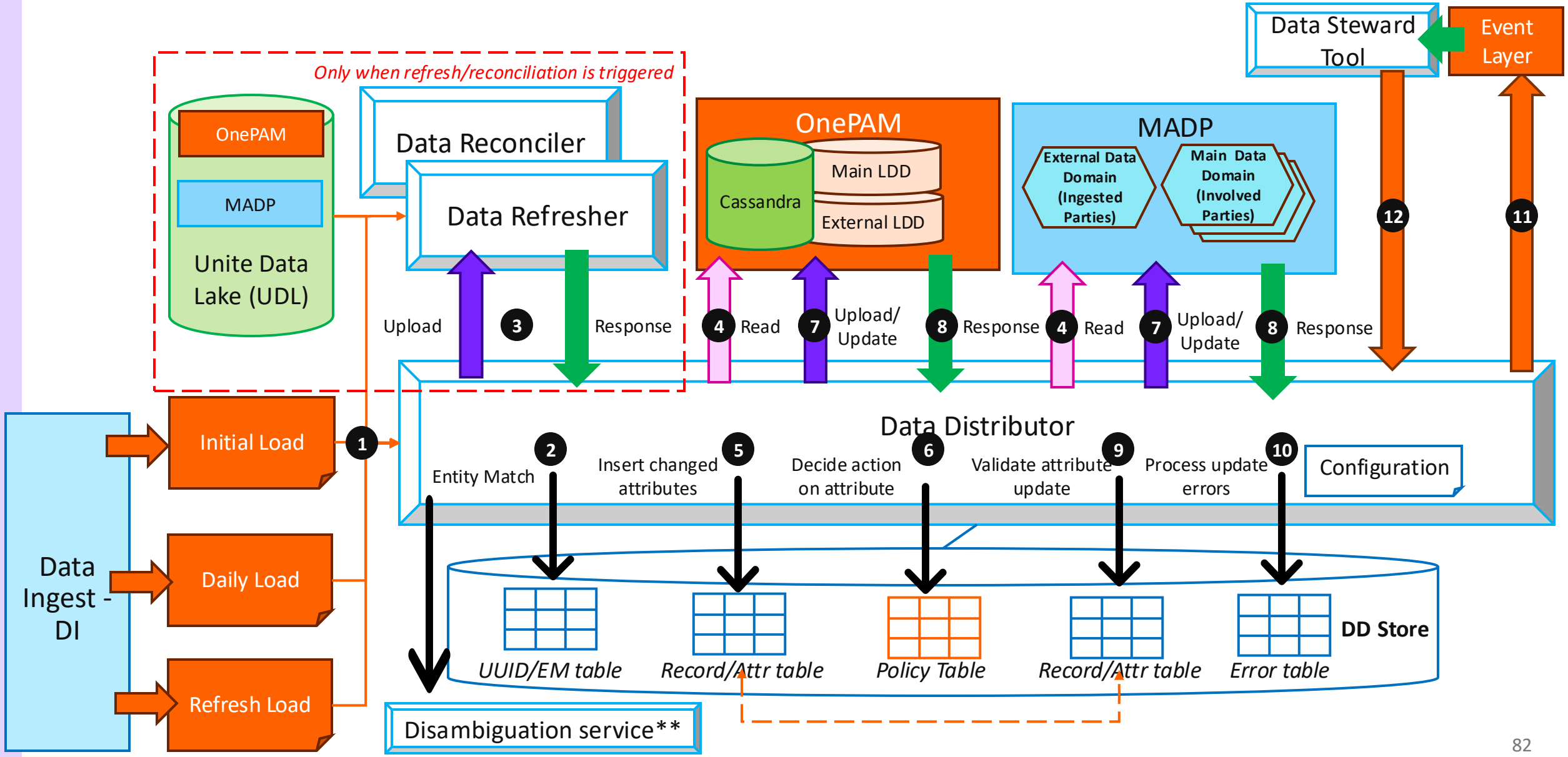
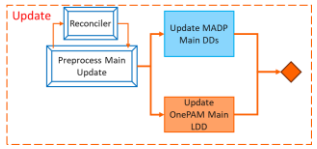


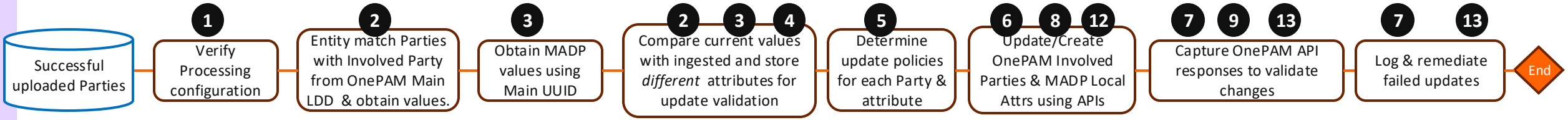
Table of Content

Phase 3b: MADP Main DD update

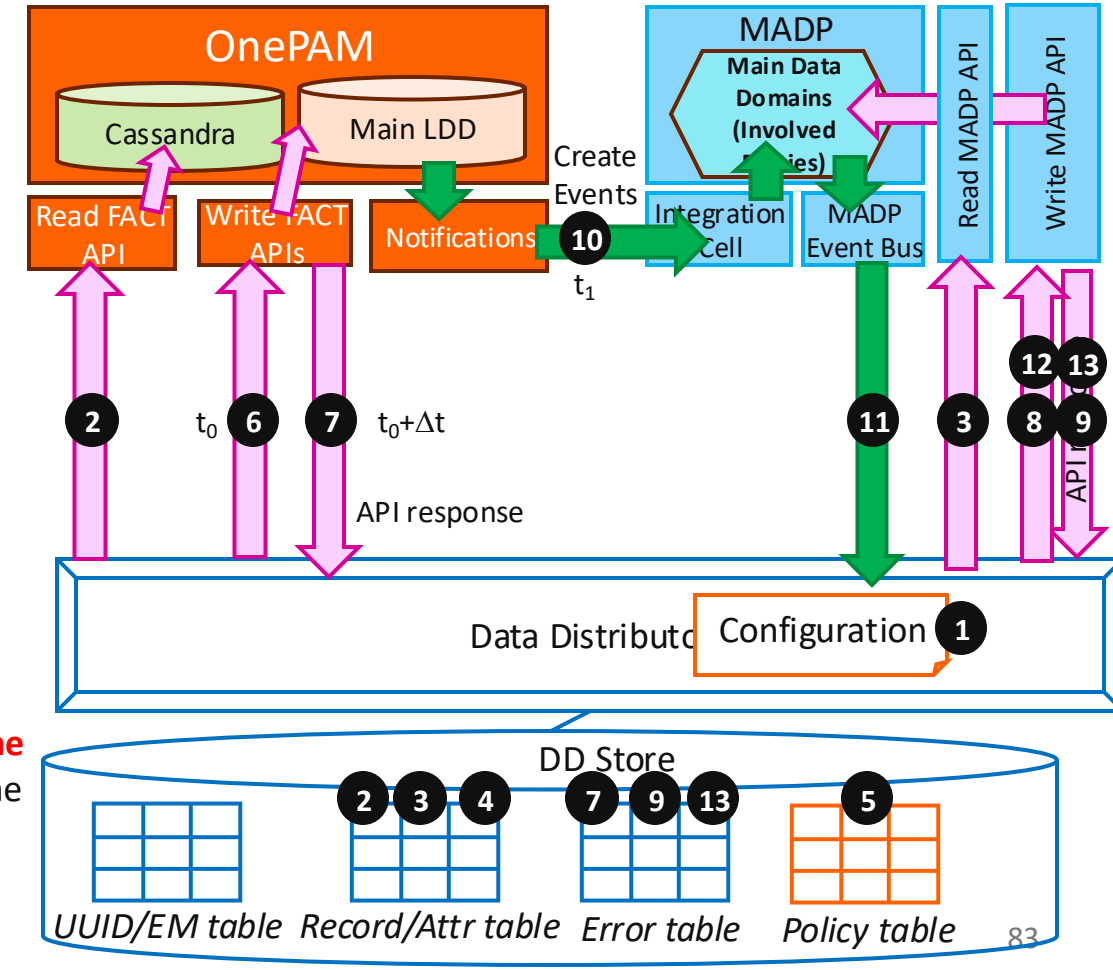


4. MADP "Main" DDs - Daily upload pattern – API (1/2)



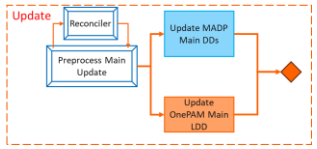


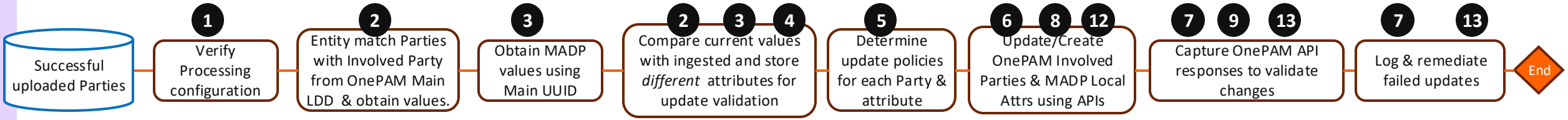
1. We verify the **configuration** as to what sources & successful external ingested & uploaded Parties in the **External LDD** must be propagated.
2. The **Data Distributor** reads the **OnePAM* Main LDD (API)** to match each Party with **Involved Parties (IP)** using **External UUID** or demographics/LE # & where necessary disambiguates them. It saves Core attributes in the *Record/Attr table.* For matching records (existing **Parties**), it obtains the **OnePAM Main UUID**
3. The **DD** reads the **MADP Main Data Domains** (API) using the **OnePAM Main UUIDs** to obtain the "Local" attributes & saves them in the *Record/Attr table.*
4. **DD** compares matching ingested & uploaded Party attributes with Involved Party Core & Local attributes and see where they are different. It also determines what new **Involved Parties** & relationships must be created.
5. The **DD** determines the update policy for each type of **Involved Party** attribute (overwrite if different, update if changed, don't overwrite…)
6. The **DD updates** changed Core attributes of existing **Involved Parties (API),** using the **Main UUID,** creating new Parties/relationships) & performs (soft) deletes.
7. The **DD** collects API responses to account for all created **Involved Parties, collecting the new OnePAM Main UUIDs,** updated Core attributes and updates the *Error table* for the ones where errors are thrown. Also deleted Parties/relationships are accounted for.



* Differences in flow by adding MADP is given in red.

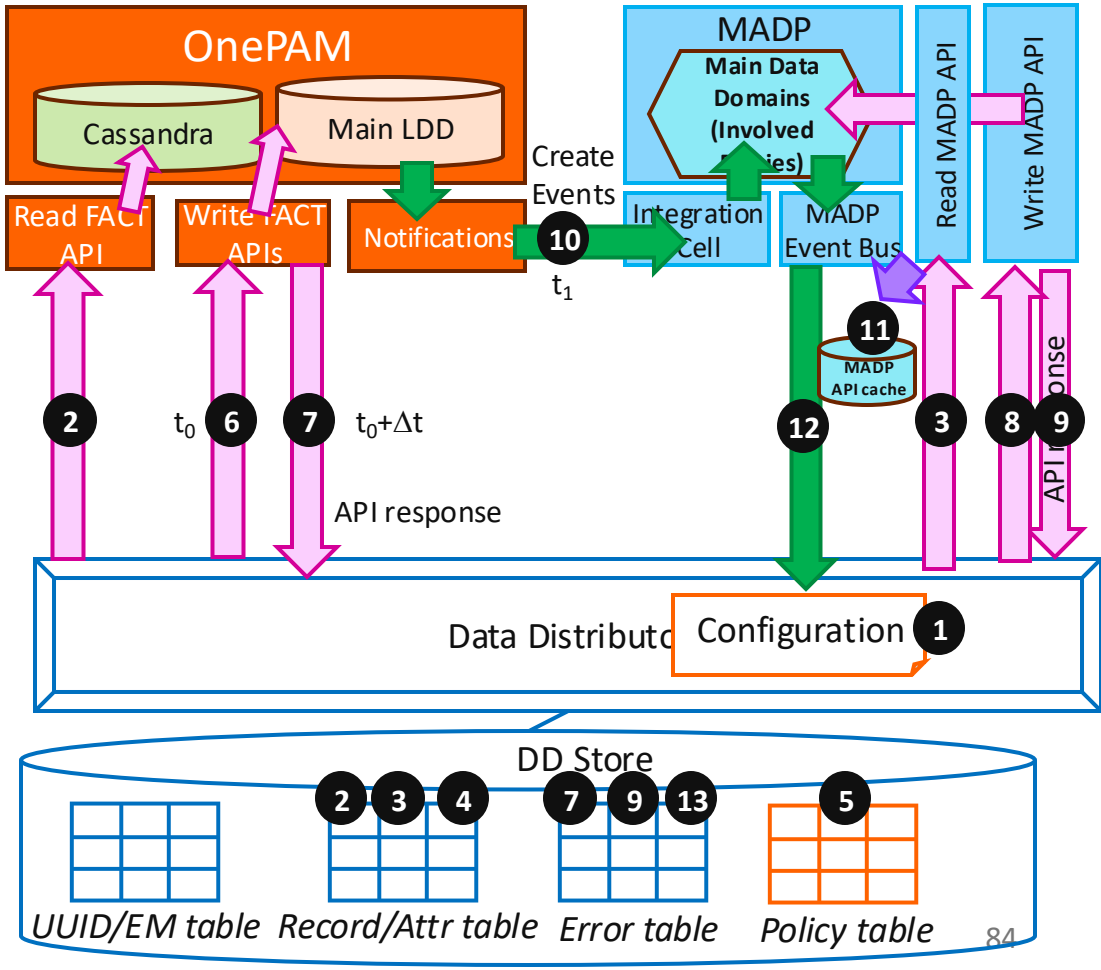
4. MADP “Main” DDs - Daily upload pattern – API (2/2)





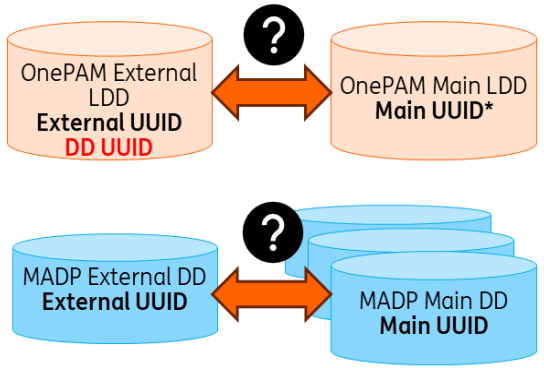
8. The **DD updates** changed Local attributes of **Involved Parties (API)** in **MADP,** using the earlier collected existing **OnePAM Main UUID.**
9. It captures responses and updating the *Record/Attr table* to account for updates & updates the *Error table* for the ones where errors are thrown.
10. New **OnePAM Party** creation **Notifications** trigger **MADP** to create the object with the **OnePAM External UUID** in the **MADP External Data Domain.**
11. For new created Parties with **External UUIDs** which have not yet been processed by **MADP**, API updates will be cached in the **MADP API Cache**
12. **DD** captures the **MADP** cached creation events containing the new **OnePAM Main UUIDs** with the updated attributes.
13. It updates the *Record/Attr table* to account for updates & updates the *Error table* for the ones where errors are thrown, or expected events never arrive.

Not shown on the drawing, but all **OnePAM Notifications** and **MADP Events** are also captured and stored in the **UDL**.



Reconciliation add MADP: exploit DD UUID as common UUID

Step 1: Start situation



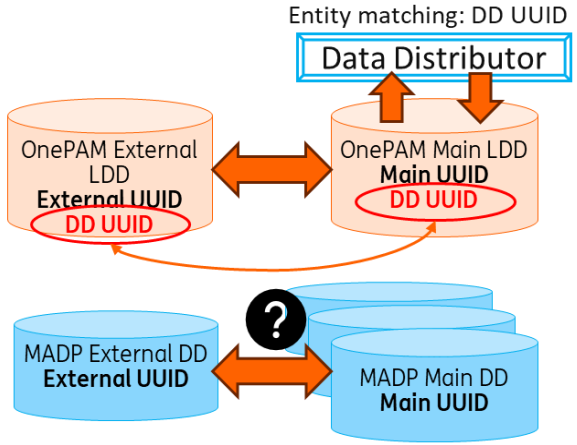
During reconciliation we must compare attributes between:

- The **OnePAM Main LDD** and **OnePAM External LDD**
- The **MADP External DD** and **MADP Main DDs**

The challenge is that there is ***no common key*** of comparison between External & Main data stores.

This is especially an issue with **MADP** where also the attributes are different between External & Main DDs.

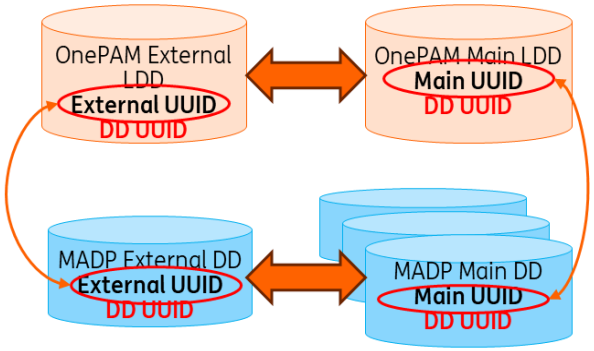
Step 2: add DD UUID to Main LDD extract



We will use the **DD UUID** as a ***commo,n key*** and use the **Data Distributor** which has all the entity matching information for the **OnePAM External LDD**, which is what we want to compare with.

We extract the **OnePAM Main LDD** to the **DD** that does entity matching and adds the **DD UUID** to matching records. Unmatched records are dropped as they are not in the External LDD.

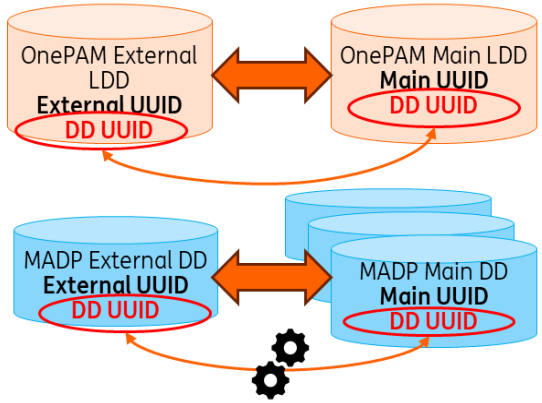
Step 3: enrich MADP DDs with DD UUID



We will then use the **External UUID** as a common key between **OnePAM External LDD** and **MADP External DD** to enrich **MADP External DD** with the **DD UUID.**

We will use the **External UUID** as a common key between **OnePAM Main LDD** and **MADP Main DDs** to enrich **MADP Main DDs** with the **DD UUID**

Step 4: reconcile using DD UUID to match

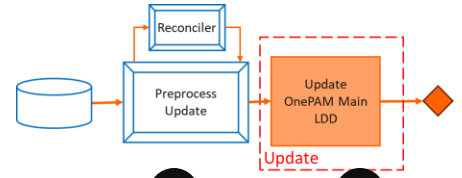


We now reconcile **OnePAM Main LDD** & **OnePAM External LDD** by matching **DD UUID** and *comparing* the attributes the External LDD is master for with matching Main LDD attributes.

We do the same for the **MADP External DD** & **MADP Main Data Domains**, matching the **DD UUIDs**. We must *transform* External DD attributes into Main DD attributes as their models are different.

* Over time the Main LDD will also have the External UUID as an Internal Identifier but that is through daily updates, not through reconciliation (unless we want a giant reconciliation).

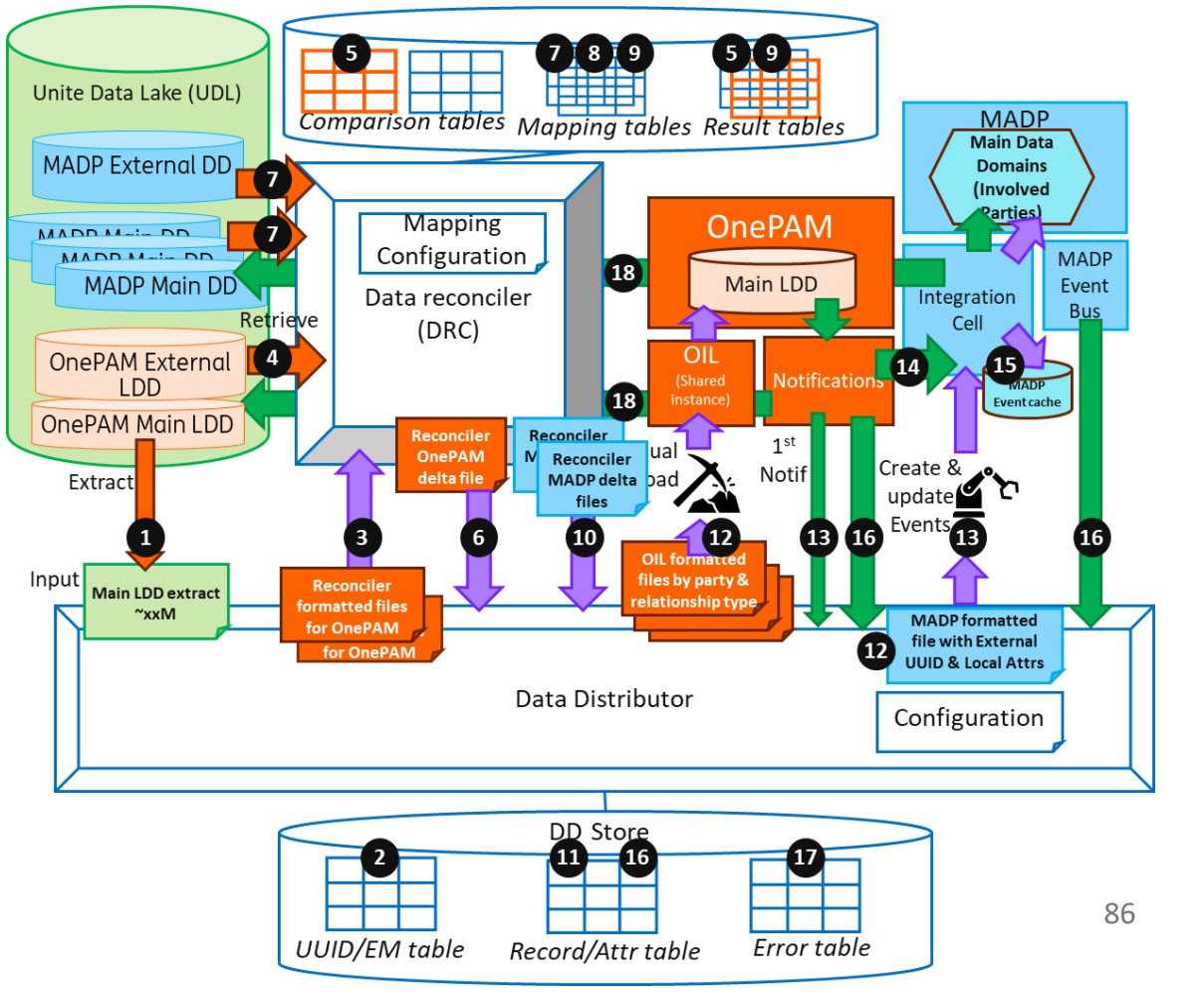
5. MADP "Main" LDD – Reconciliation pattern (1/3)



1. OnePAM Main LDD extract from UDL
2. Do entity matching and assign DD UUID to each matching Involved Party
3. Create Data reconciler files and upload to the DRC
4. DRC retrieves full OnePAM External LDD & MADP External & Main extracts
5. Compare Main & External LDD by DD UUID
6. Send DRC delta files Main OnePaM & MADP to DD
7. DRC retrieves full OnePAM External LDD & MADP External & Main extracts
8. Map DD UUID on MADP extracts & compare Main & External DD
9. Map DD UUID on MADP extracts & compare Main & External DD
10. Send DRC delta files Main OnePaM & MADP to DD
11. Store *changed* attr in table & upload delta to OnePAM
12. Store *changed* attr in table & upload delta to OnePAM
13. When the first Notification is captured from OnePAM we use the automated upload facility for the MADP file
15. Attributes of not yet MADP processed new Party Notifications are cached.
16. We capture OnePAM Notifications and MADP events to validate changes
17. Log & remediate failed updates

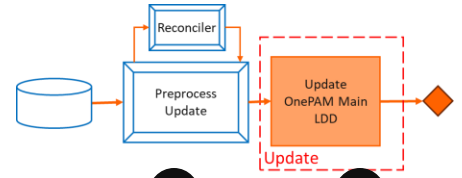
End

1. As input for the **Data Distributor,** we extract the **OnePAM Main LDD** data from **UDL** & send it to the **DD**.
2. The **Data Distributor** matches each record on already existing ingested records in the **DD Store** using demographics/LE # & to map them on a **DD UUID**, and, if necessary, disambiguates them.
3. **DD** creates new files with all OnePAM records and the **DD UUID** & uploads them to the **Data Reconciler (DRC)** component.
4. The **DRC** retrieves all Parties & relationships of the **OnePAM External LDD** from the **UDL**.
5. Both are stored in a *Comparison table* by **DD UUID** & compared with resulting delta records stored in a *Result table*.
6. **DRC** creates new files with deltas of **Involved Parties** & relationships and send it to the **DD**. These records are already **Main UUID** tagged.
7. **DRC** retrieves all records from the **MADP External DD** and **MADP Main DDs** from the **UDL** & stores all **MADP** extracts in *Mapping tables*.
8. **DRC** links them with the **OnePAM External/Main UUIDs** in the *Comparison tables* & it adds the **DD UUID** to the matching records in the *Mapping tables*. It deletes the records that don't have matches.



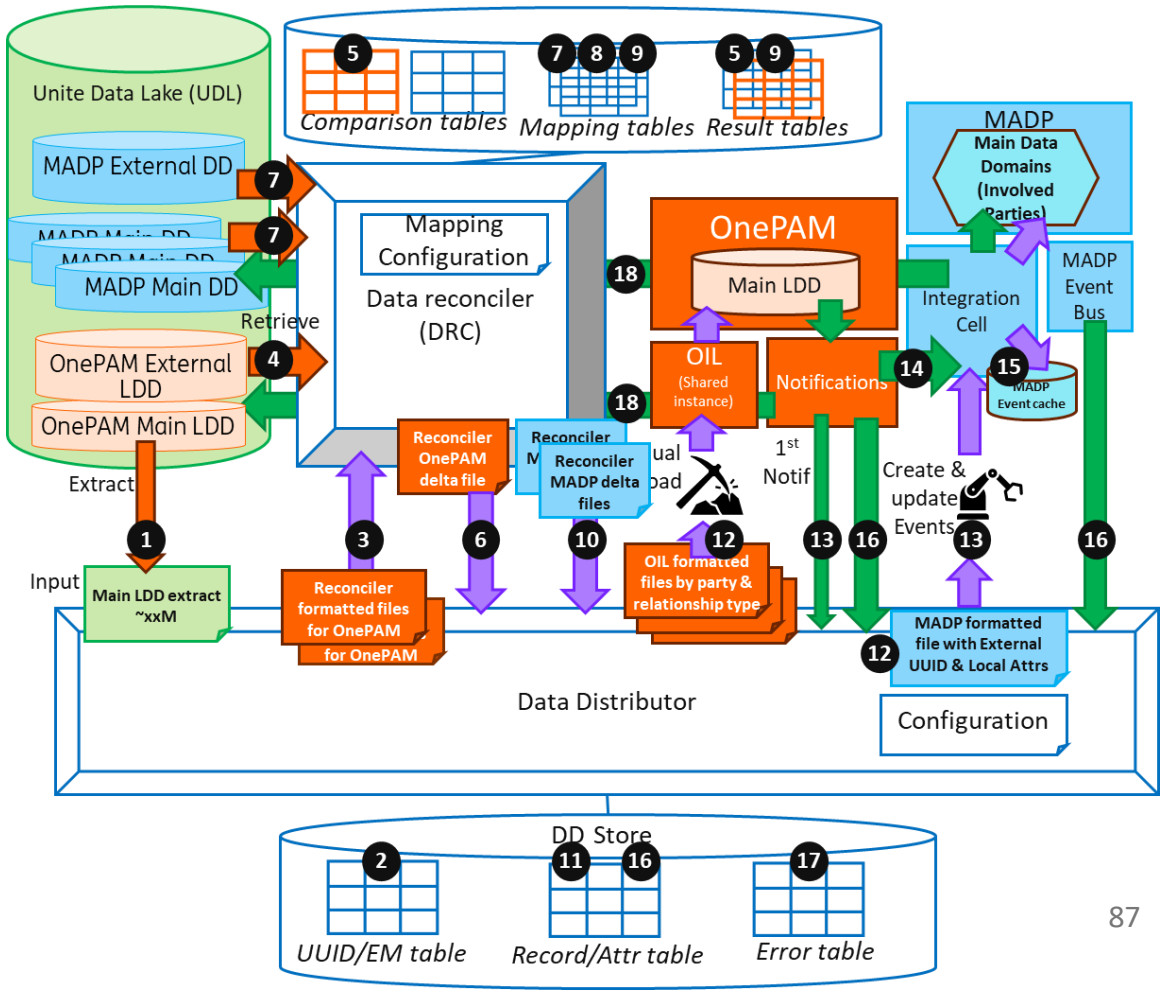
* Differences in flow by adding MADP is given in red.

5. MADP "Main" LDD – Reconciliation pattern (2/3)

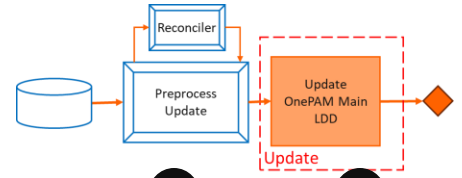


1 OnePAM Main LDD extract from UDL
2 Do entity matching and assign DD UUID to each matching Involved Party
3 Create Data reconciler files and upload to the DRC
4 7 DRC retrieves full OnePAM External LDD & MADP External & Main extracts
5 Compare Main & External LDD by DD UUID
8 9 Map DD UUID on MADP extracts & compare Main & External DD
6 10 Send DRC delta files Main OnePaM & MADP to DD
11 12 Store *changed* attr in table & upload delta to OnePAM
13 When the first Notification is captured from OnePAM we use the automated upload facility for the MADP file
15 Attributes of not yet MADP processed new Party Notifications are cached.
16 We capture OnePAM Notifications and MADP events to validate changes
17 Log & remediate failed updates
End

9. **DRC** hen maps and compares the **MADP External DD** attributes with the different **MADP Main DDs attributes** in the *Mapping tables*, considering that the **External DD** has a different model/attr names than the **Main DDs** and stores the results in the *Result tables*.
10. **DRC** creates new files with deltas of **MADP Main DDs attributes** and send it to the **DD**. These records are already **Main UUID** tagged.
11. We assume all "Core" and "Local" attributes in the received records from the **DRC** have changed & store them in the *Record/Attr table* to account for updates in **OnePAM Main LDD & MADP Main DDs**.
12. **DD** creates OIL formatted files and transfers them to OIL for **manual scheduling** of the upload into **OnePAM Main LDD**. It also creates the **MADP** files with **OnePAM Main UUID** embedded but doesn't upload them yet.
13. The first **Notification** tells **DD** that **OnePAM** processing started & triggers it to upload the **MADP** file (with **OnePAM Main UUID**) for automated processing.
14. New **OnePAM Party** creation **Notifications** trigger **MADP** to create the object with **Main UUID** in the **MADP Main Data Domains**.



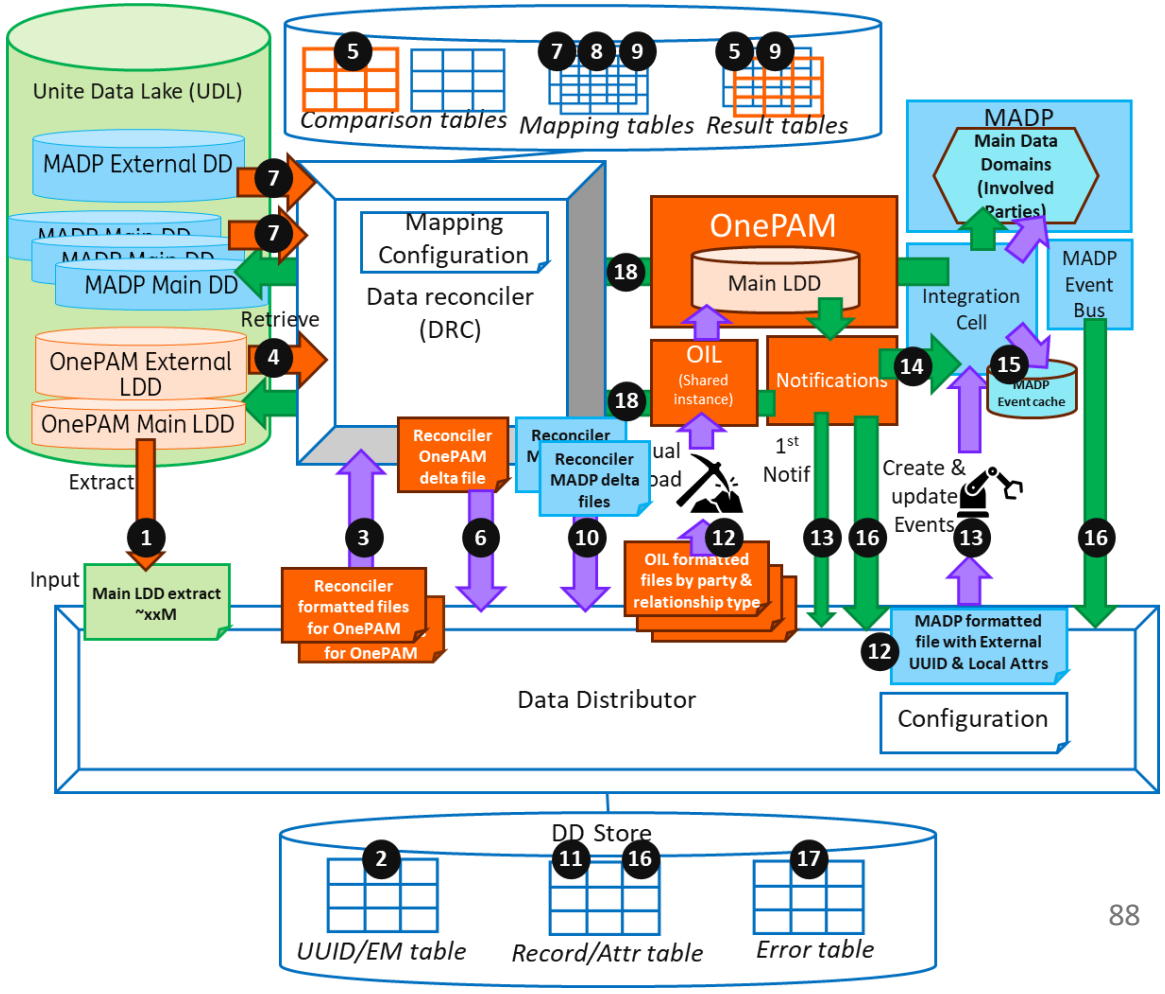
5. MADP "Main" LDD – Reconciliation pattern (3/3)



1. **OnePAM Main LDD extract from UDL**
2. Do entity matching and assign DD UUID to each matching Involved Party
3. Create Data reconciler files and upload to the DRC
4. 7 DRC retrieves full OnePAM External LDD & MADP External & Main extracts
5. Compare Main & External LDD by DD UUID
6. 9 Map DD UUID on MADP extracts & compare Main & External DD
7. 10 Send DRC delta files Main OnePaM & MADP to DD
8. 12 Store *changed* attr in table & upload delta to OnePAM
9. 13 When the first Notification is captured from OnePAM we use the automated upload facility for the MADP file
10. 15 Attributes of not yet MADP processed new Party Notifications are cached.
11. 16 We capture OnePAM Notifications and MADP events to validate changes
12. 17 Log & remediate failed updates

End

15. If records with new **OnePAM Main UUIDs** arrive before the creation of the object in the **MADP External Data Domain**, it will be cached in the **MADP Event Cache** and retried by **MADP** after the object is created.
16. We **capture all update, creation & deletion events** from **OnePAM & MADP** to account for all created Parties/relationships & updated **Party** "Core" and "Local" attributes with the *Record/Attr table*.
17. We update the *Error table* for the ones we don't receive. Also deleted Parties & relationships are accounted for.
18. All **OnePAM Notifications** and **MADP Events of changes of Involved Parties attributes and relationships in the OnePAM Main LDD and MADP Main DDs** are also captured and stored in the **UDL**.



Part 3: Detailed Architecture

Table of Content

OnePAM technology architecture & access patterns

OnePAM implements the **Command Query/Response Segregation (CQRS) pattern** where reads & writes are separated in different, usage optimized components:

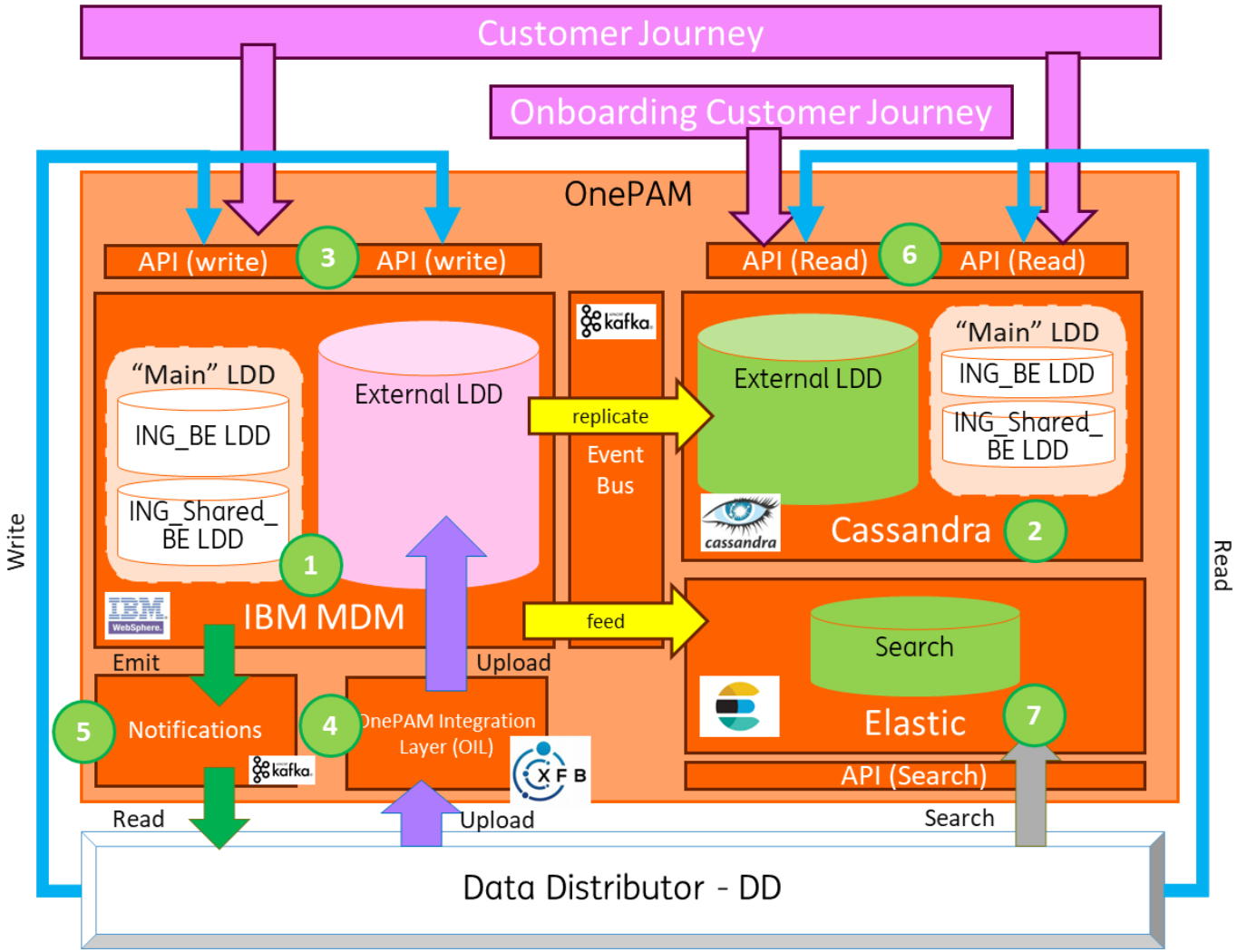
1. **Writes happen in IBM MDM** (5% of all calls). IBM MDM is the master of the data
2. **Reads happen in Cassandra** (95% of all calls) and all updates in MDM are replicated hhere (there is a delay).

There are several ways to **write data** to OnePAM:

3. **APIs:** currently this is through **"Fact" APIs.** They are **rate limited to 5 tx/sec/user**. They insert the data into **IBM MDM**.
4. **OIL**: the **OnePAM Integration Layer**, is for bulk uploads. It reads files and then loads the records in **IBM MDM.**
   - It provides scheduling and file order preservation
   - **Scheduled Daily uploads** are **rate limited to ~3RPS**
   - **Manual uploads** have a **guaranteed 5 RPS**.

There are several ways to **read data** from OnePAM:

5. **Notifications**: are 'business' events emitted when changes occur in **OnePAM LDDs**.
6. **APIs**: read-only Fact APIs on **Cassandra** (read-optimized)
   - **Read APIs** are **not rate limited** (1000 tx/sec/user)
7. **Search API**: There is also an **Elastic Search** capability.



**Surprise! There are some features as scheduling, order of processing etc that take away work you'd only have to do yourself.*

Context – MADP technology architecture & access patterns

MADP implements a "Cell"-based architecture segregating "Local" Party attributes into functional "**Data Domains**" (DD).

- The **External Data Domain** contains "Local" Party attributes linked to **External UUIDs** of the OnePAM External LDD.
- The **"Main" Data Domains** contain "Local" Involved Party attributes linked to **Main UUIDs** of the OnePAM Main LDD.

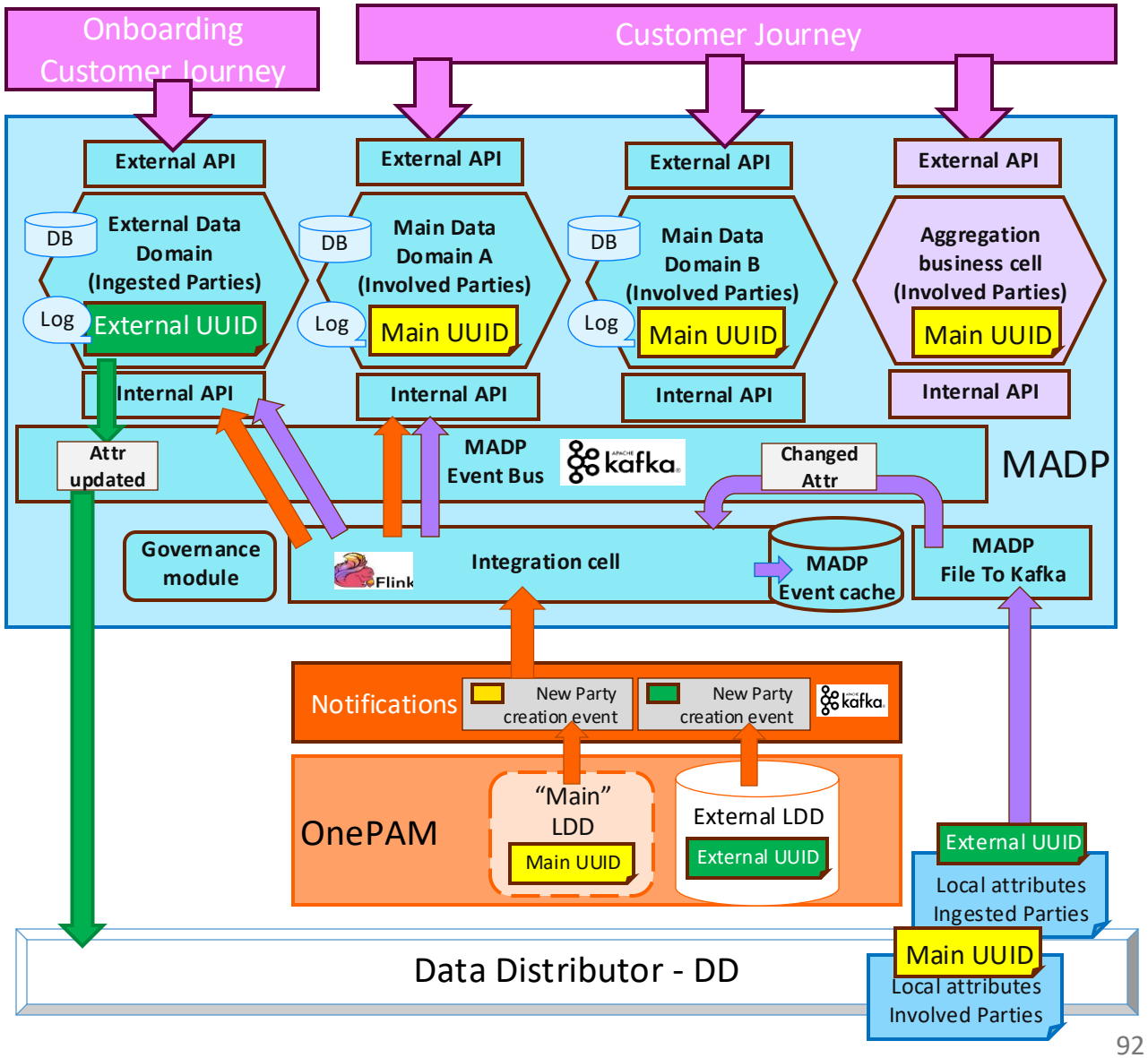
The **MADP Integration Cell** captures **Notifications** of new created Parties in OnePAM LDD's and creates Party objects in the right Data Domains* using info in the **Governance module**.

MADP data **requires to know the UUID** to be read and updated. It can be read & updated by:

- **External exposed APIs per Data Domain** using UUIDs. Not yet created UUIDs of new Parties give an error.
- **Files uploaded to MADP**, where records with UUIDs are turned into events, processed by the Integration Cell who puts it in the right DD. Changed attributes for Parties not yet created are cached & retried.

All objects with updated attributes in a Data Domain generate events with before/after information that are published on the **MADP Event Bus** and are captured by the **Data Distributor** to ensure all changed attributes are updated.

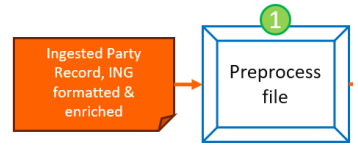
There is **no information on the throughput capacity** of MADP APIs and File based processing.



**This is based on the LDD name in the event to segreggate External Party objects from Main Involved Party objects.*

Table of Content

Ingested Party records become fragmented for uploads



1. All ingested input files have **their own format** and are provided as **Party-type records** (*Legal Entities, Private Individuals, Operating Sites…*) with attributes linked to them. . We know they **represent changed records**, but don't know upfront what attributes changed. All files come **in their own schedule, processed individually & separately** due to time sensitivity.

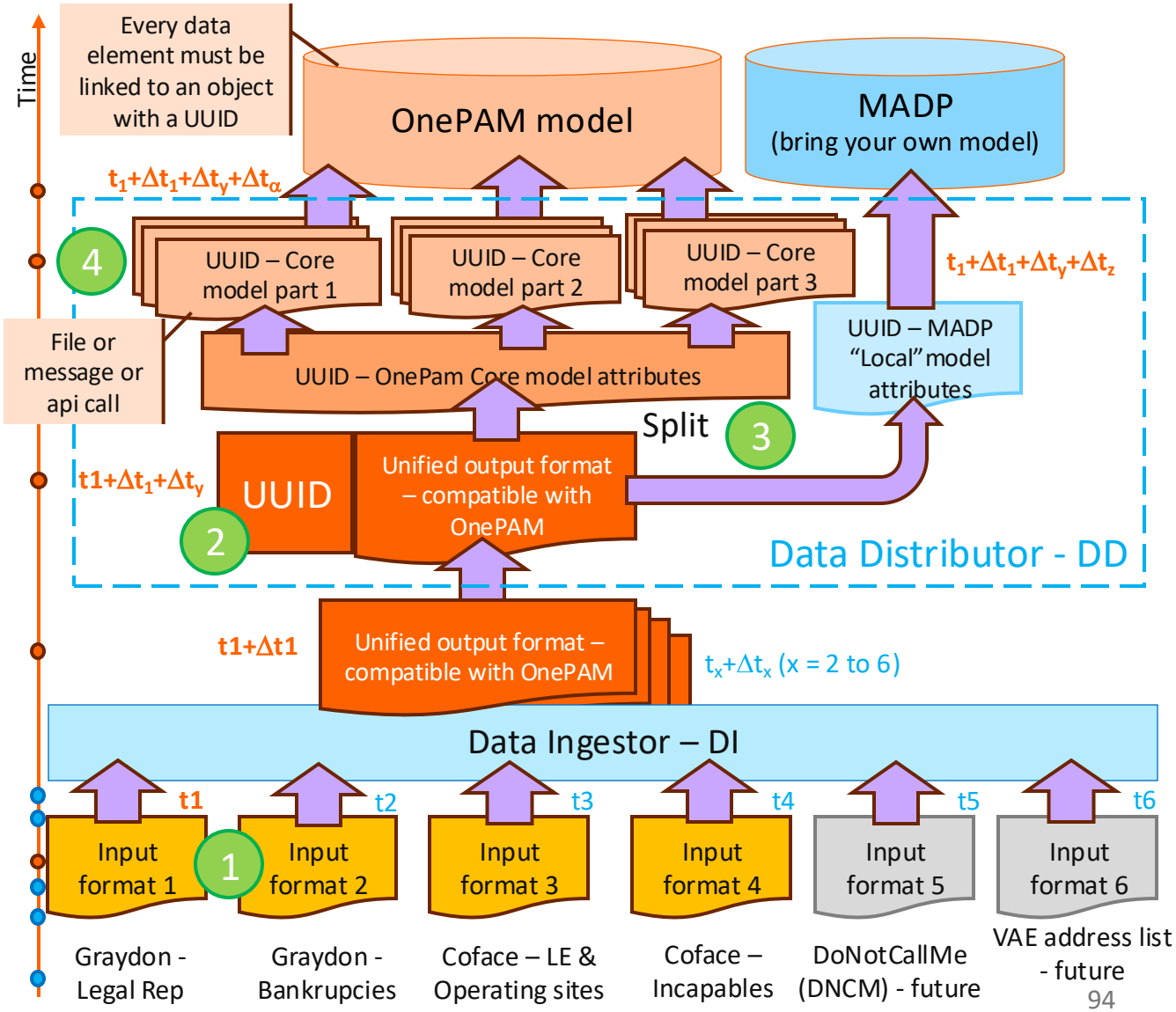
2. All ingested Party records **must be linked** to a Party in **OnePAM** (or be newly created) so they must be **enriched with a UUID** that OnePAM can process. **This is true for existing as well as new Parties that must still be created** (see later)**.**

3. Processed **input files are split** for **different target platforms** (OnePAM & MADP)

4. **They may be split further per target platform** to facilitate upload requirements:

    - There are a number of OnePAM OIL-file formats depending on actual data to upload (see later) but one OIL-file can contain all records to upload with the subset of attributes as defined by the OIL-file format.

    - When using fine-granular APIs records must be splitted over several API calls, and uploading each record may ultimately translate into ten-of-thousands API calls



Transform Ingested Party Record to OnePAM model

The Ingested Party record format is different from the **OnePAM** or **MADP** model:

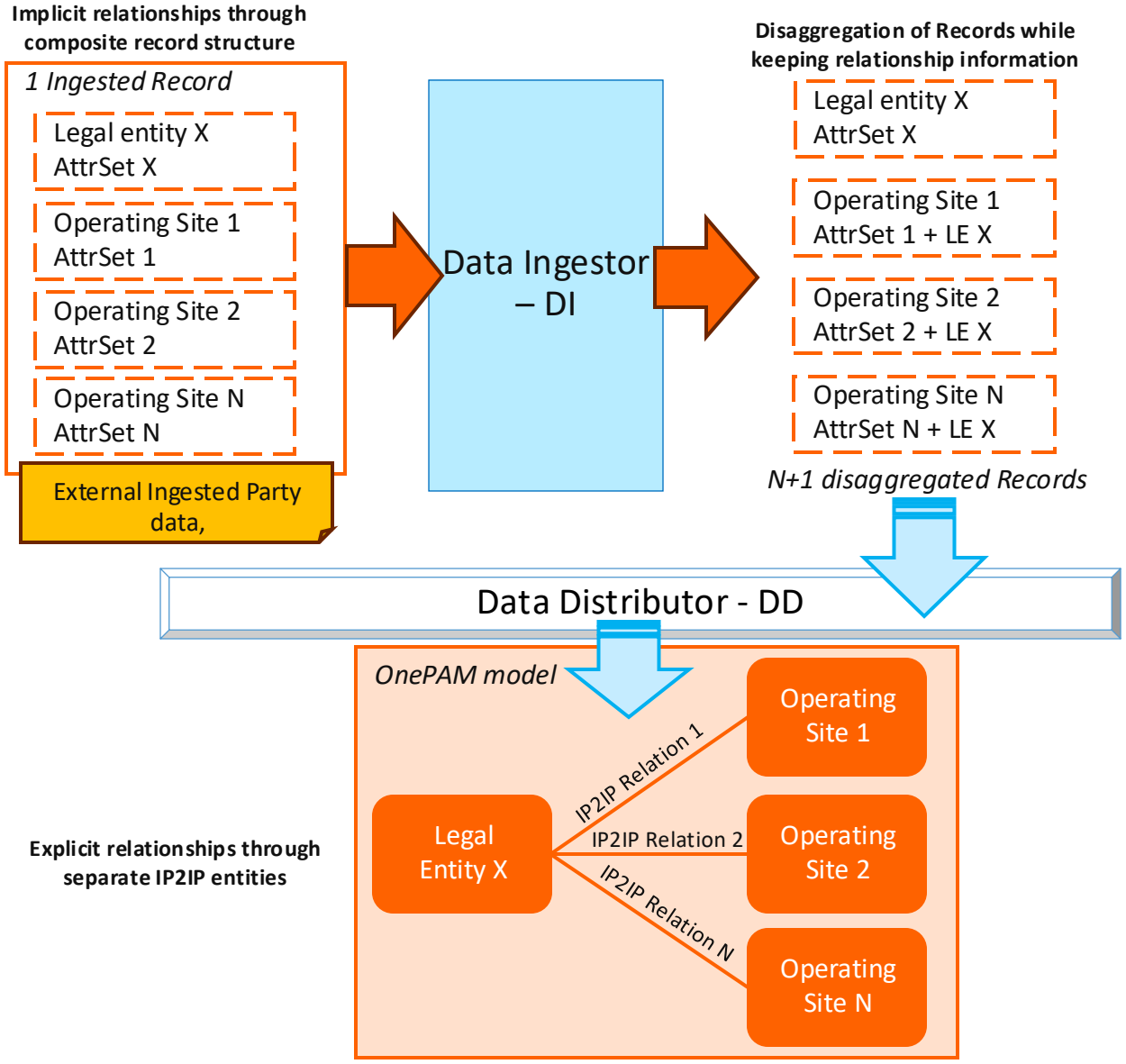
- Ingested Party Records contain **implicit Parent-Child relationships** inferred from the record formatting
- The OnePAM data model has different explicit object entities for each entity type (Legal Entity, Operating Site…) that are connected by **explicit modeled Party2Party relations**.

To upload data in OnePAM, **we must create new, separate records for each object entity** (Legal Entity X, Operating Site 1 …) **as well as for each relation** between each object entity.

The **Data Ingestor** will disaggregate to ingested Record into individual object pairs, making the relationship visible.

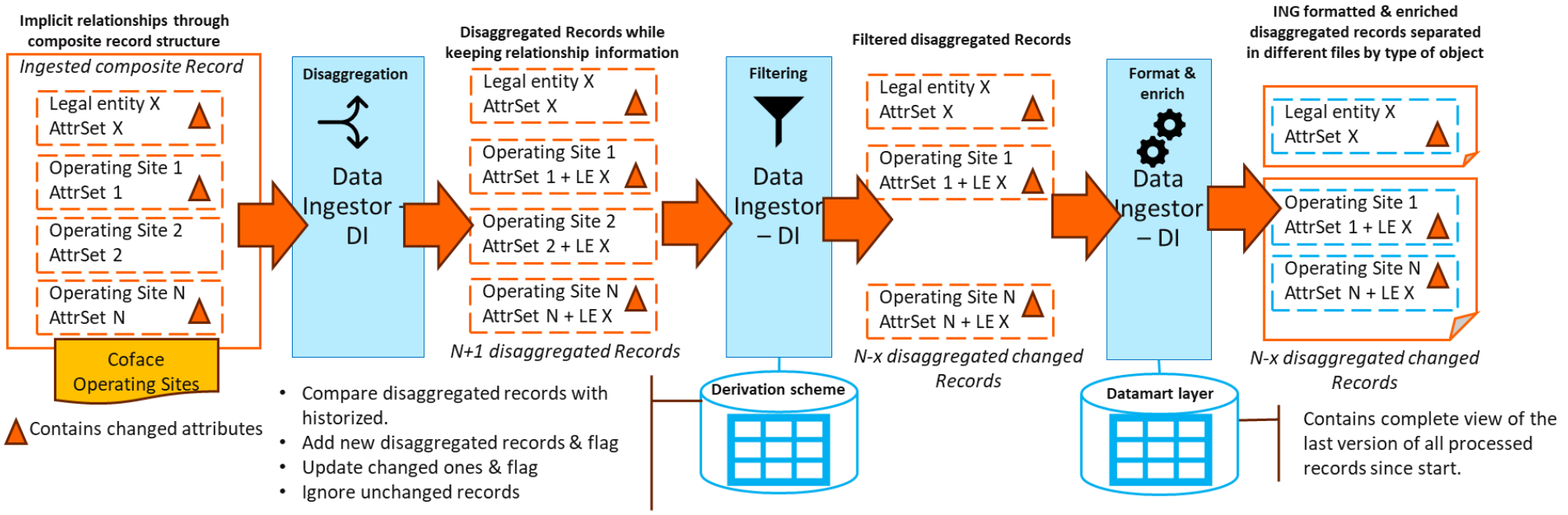
The **Data Distributor** must translate these object pairs into OnePAM Party objects and explicitize the relationship between them as IP2IP relations. The DD also must manage their lifecycle, when Parties & Relations must be created or deleted*.

Important remark: At minimum we keep information about the the relationship between objects at the subordinate level (Operating Site must know to which Legal Entity it belongs).



** We don't hard delete, only soft-delete. Technically we fill in an « end date » attribute on the object.*

Data Ingestor - Disaggregate, filter & format/enrich ingested records



- The **Data Ingestor** starts with **disaggregating each ingested composite record** in as many records as there are objects, adding the relationship info to at least the subordinate object records. It stores this in the "***Derivation Scheme***" of the **Data Ingestor Store**.
  - This multiplies each record to process by N+1 but keeps the total amount of information the same.
- While each composite record will have changes, but not all disaggregated records will & we **filter these out** so we don't process them.
- After processing, we send a separate file for each object type (Legal Entity, Operating Site) that will **contain only changed disaggregated records** (with relationships embedded) to the **Data Distributor** for further processing.

Deduplicating Parties using a unique DD UUID.

Within and across the ingested files **duplicate parties** exist:

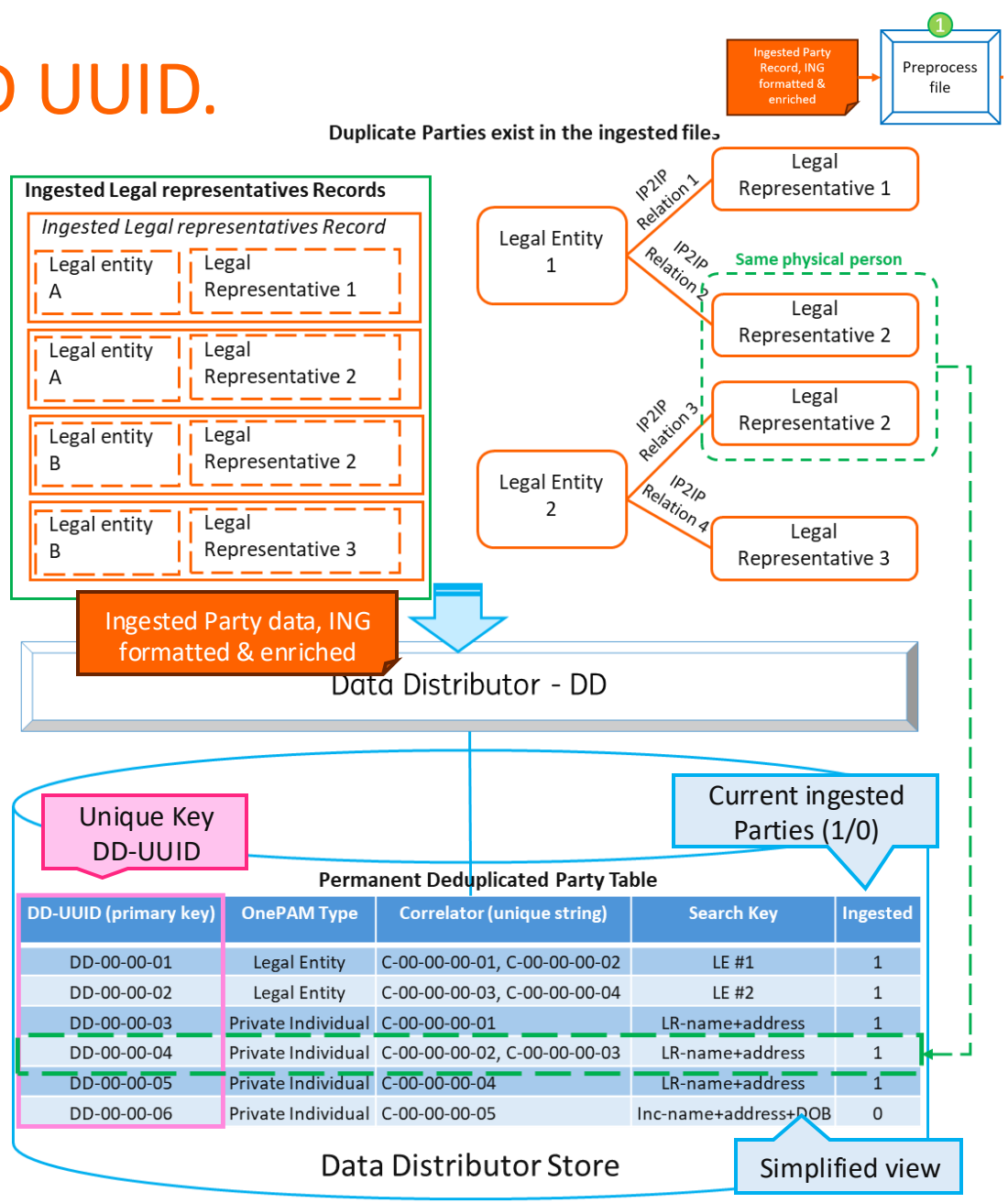
- The same Legal Entities can be found in Legal representative, bankruptcies and Operating Sites ingested records
- A Party can be a Legal Representative in different Legal Entities
- A Bankruptcy curator can be a Legal Representative
- A Legal Representative can be(come) an Incapable or an administrator

We need sufficient data as search key to merge 2 Parties & avoid false positives:

- **Legal Entities**: Its ***Enterprise Number*** is unique (easy case)
- **Operating Site**: is uniquely identified by its associated Legal Entity in the disaggregated record.
- **Private Individuals**: either ***name+surname+address*** and/or ***name+surname+birthdate(+birth location).*** This seems relatively safe but is vulnerable to errors in names and addresses, incomplete or even missing addresses, different addresses (after a move). Where we don't have sufficient certainty, we may ask a **central disambiguation service*** or, when important mandates are involved, leave the linked Parties as attributes of the main Party and leave it up to a downstream business process to disaggregate them (for example after receiving additional documentation).

For each newly ingested Party we search for duplicates, both the just ingested ones as the already existing in the database and merge their rows,while managing overlapping but possibly conflicting data (typically demographics).

For newly created Parties (in the initial load this will be all of them) we assign a unique **Data Distributor UUID (DD-UUID) uniquely identifies the Party**. This DD-UUID will be used in the uploading of data to OnePAM**.



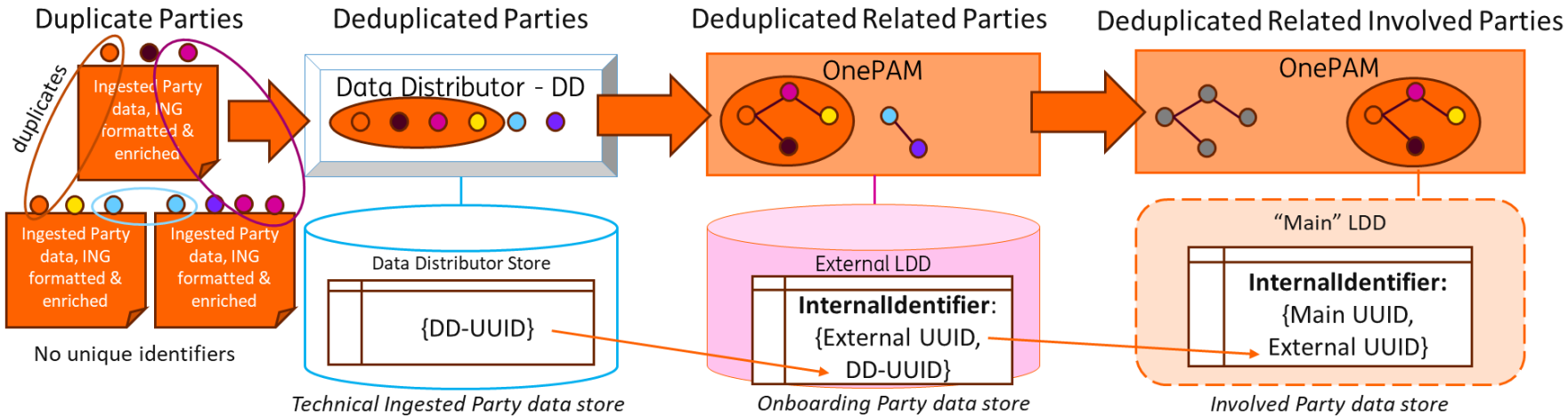
** Proposal Stijn Carmen on having a central service that indicates, given certain inputs, whether the object already exists and which it is.*

*** This is a similar approach as used by the CSI-Bridge that also uses the CSI-ID rather than a OnePAM UUID to upload data. See further for explanation.*

Chain-of-Custody UUID principle in Data Distributor

To come from **unmarked, possibly duplicated** Ingested **Party** data to deduplicated **Involved Parties** with relationships, we drive Ingested **Party** data progressively by way of the **Data Distributor,** first through the **External LDD** towards **Involved Parties in the Main LDD,** identified by UUIDs.

OnePAM requires external sources to have their own UUID, defined outside OnePAM, when uploading data using files (OIL), which we do in the initial load of data. Hence, we create a "**DD UUID**" in the **Data Distributor** and attach it to each Party we upload to OnePAM. The **DD UUID** will become a linking pin in the architecture to organize the initial & dail uploads, refreshes & reconciliations.



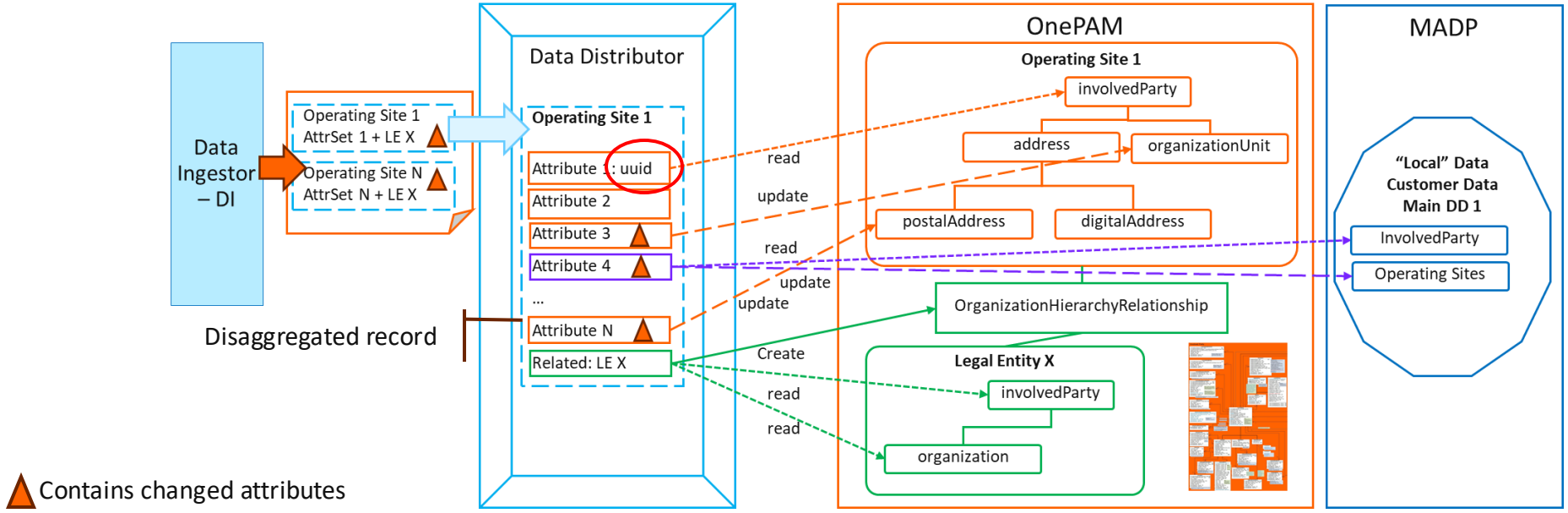
The **DD UUID** will be used to:

- **Deduplicate** ingested records (entity matching) by being attached to mapping attributes like Legal Entity number or demographics.
- **Search** for Parties to obtain the **OnePAM External UUID or update** them using the **DD UUID** given that ALL uploaded parties in the **OnePAM External UUID** will have both the External UUID and DD UUID attached (MADP only supports the OnePAM UUID as key).
- **Map** Parties from different **OnePAM LDDs** with different **Main/External UUIDs** on each other during **refresh** and **reconciliation**.

Equally we attach the OnePAM External UUID when creating or updating **Involved Parties** in the **OnePAM Main LDD** after upload in the **OnePAM External LDD.** This creates a "Chain of Custoday" between the repositories. An **important caveat** is that merging Involved Parties (**and External UUIDs**) in the **"Main" LDD** must be backpropagated through the chain up to the **Data Distributor**.

Table of Content

Mapping attributes on OnePAM & MADP objects

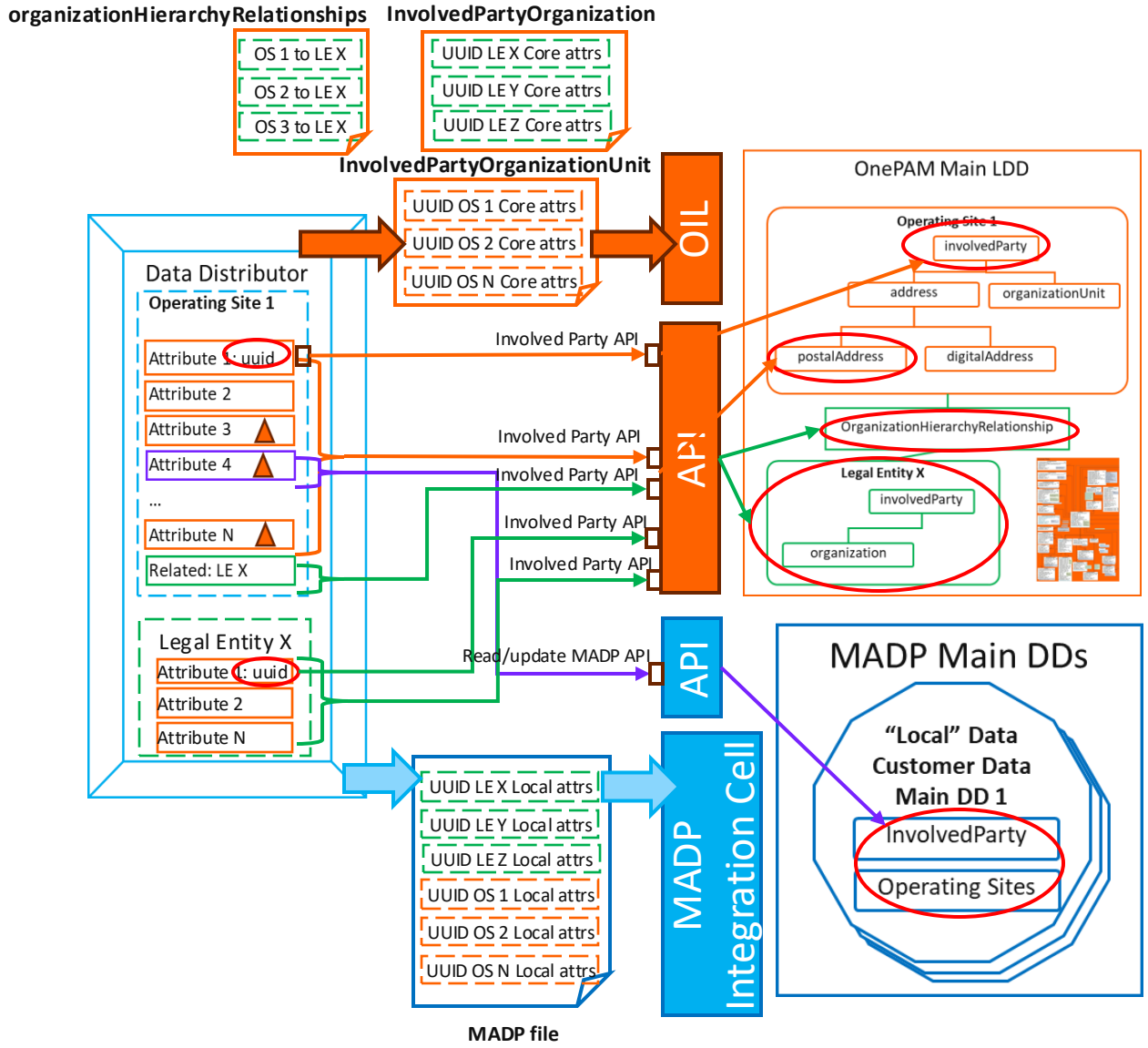


- The **Data Distributor** takes each disaggregated record & **maps it to Object attributes** in the OnePAM Exchange Model as well as for MADP.
  - A **single disaggregated record is thus potentially further disaggregated** into different Object updates in different repositories.
- As part of updating changed attributes, we must:
  - Identify which **OnePAM object** (UUID) each record refers to know what object to update.
  - Identify which records represent **new/deleted objects** and must be created/deleted*.
  - Identify which **attributes** actually changed and must be updated.
  - Identify which **relationships** must be created/deleted*.

** Soft delete by marking the 'end date' attribute of an object/relationship*

Reading & Updating Parties & Relationships

- The **Data Distributor must use** APIs to read information from **OnePAM** and **MADP** Objects.
- The **Data Distributor can use** either APIs or Files to create/ update/delete OnePAM and MADP Objects & attributes.
- **API: Involved Party API** is used for **both reading & writing** of Private Individuals/Legal Entities & Operating Sites
  - It requires at least 3 API calls to update an Involved Party attributes that are spread over different sub-objects (e.g. Involved Party postal address, external identifier),
  - **Separate API calls** are necessary for different **InvolvedParty Objects**, Local attributes in **MADP**, or when the ingested record would have attributes not part of the InvolvedParty aggregated object.
- **Files:** we can aggregate ingested records together in files to update **OnePAM (OIL)** and **MADP (Integration Cell).**
  - **OnePAM OIL** requires different files for different types of OnePAM objects: PI, LE & OS, as well as for relations between them. It can contain the whole **Involved Party** object and sub-objects.
  - **MADP** requires a single file with all Local attributes for all ingested records. It requires to have the associated OnePAM UUID as part of the input for each record.



** Soft delete by marking the 'end date' attribute of an object/relationship*

3 API calls to create/update a OnePAM InvolvedParty object

Involved Party API - Clarification

Yeluguri, S. (Suneel)
To Squad Sierra
Cc DUBOIS, P. (Pieter)

Reply | Reply All | Forward

Mon 27/10/2025 08:40

Dear,

I am a solution Architect for NBP – Data Ingest track, As part of the Data Ingest we have to load the Coface Operating sites in to OnePAM.
This involves creating a party (Organization Unit) in OnePAM External LDD, including its address and external reference ID. During our analysis of the available API endpoints, we identified the following three relevant endpoints:

1. Create an involved party https://touchpoint.ing.net/asm/resources/OpenApiDocument/Involved_Party_API?itProduct=P00007#post+apis.ing.com+/v5/involved-parties
2. Create a postal address https://touchpoint.ing.net/asm/resources/OpenApiDocument/Involved_Party_API?itProduct=P00007#post+apis.ing.com+/v5/involved-parties/{uuid}/postal-addresses
3. Add an external Identifier https://touchpoint.ing.net/asm/resources/OpenApiDocument/Involved_Party_API?itProduct=P00007#post+apis.ing.com+/v5/involved-parties/{uuid}/external-identifiers

However, we are wondering if there is a single API endpoint available that allows us to create all this information in one go. Please let us know If I have missed anything.

Thanks & regards
Y K Suneel

Data Distributor key integration patterns: File (OIL) & API-based

OnePAM & MADP can use both APIs as Files to Upload/Update data.

Files:

- easy to create
- hand off a lot of complexity of treatment to OIL
- batch should be faster processing than APIs … in principle

But

- asynchronous feedback through Notifications
- lack of control over timing
- OIL is not really faster than APIs.

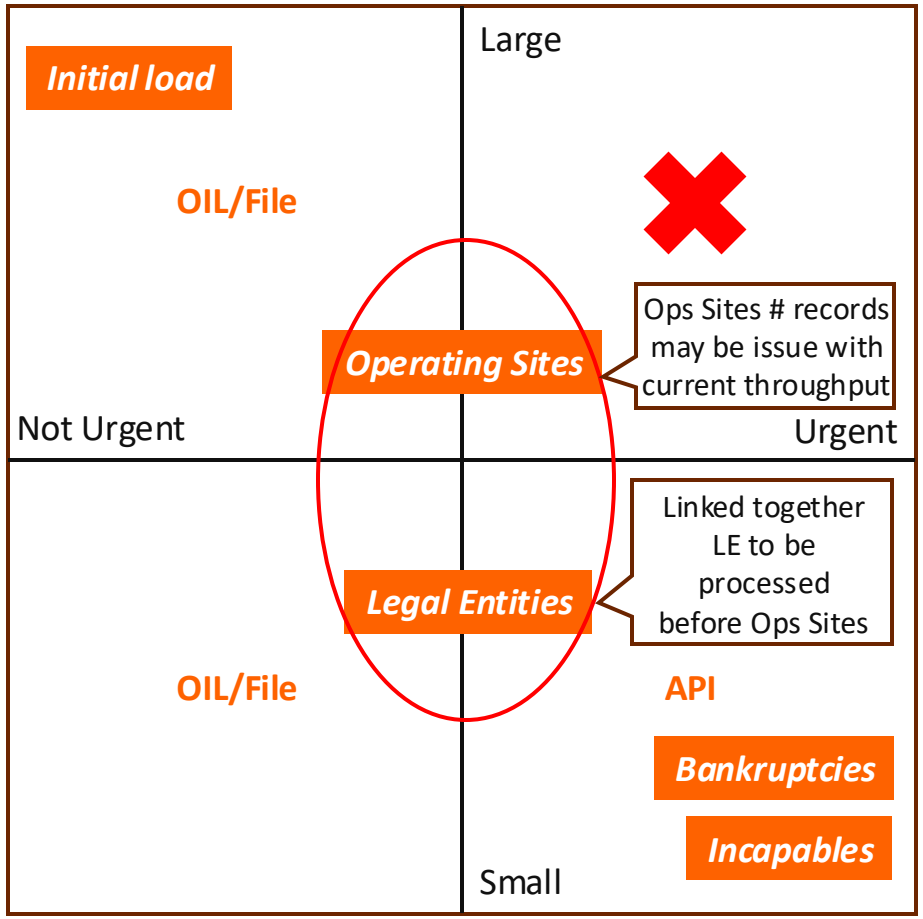
APIs:

- More control over processing with immediate feedback
- More flexible strategies in updating (order, error handling, mixing read & update, multiple repo updating.

But

- More overhead in processing, less suited for large updates
- More complexity to manage ourselves
- Higher resource consumption

We provide both mechanisms, using APIs for urgent small updates and file for large initial loads. For medium updates we may decide either way based on a variety of factors (GCDM policy, complexity, reliability…)



When to use Files, when to use APIs.

RE: BECD - question on feasibility performance increase of OIL for particular use case (loading plan) <- question for René Vermaat

Heremans, A. (Alain)
To DUBOIS, P. (Pieter); Applaincourt, M. (Michel)
Cc Jercan, L.M. (Livia - Maria); Vermaat, R. (Rene)

Reply | Reply All | Forward

Mon 13/10/2025 14:34

You replied to this message on 13/10/2025 14:42.

Hi Pieter,

Regarding the initial load, it should be done using OIL. That it takes 1 or 10 days is not important.

Regarding the synch, 10K/day is “nothing”. Still, you were mentioning much more below: “We calculated that this would be tens of thousands of records every day (between 50-100k)”, what could be 10x more.
If 10K, this should indeed be done using API calls in a throttle way (to setup at your side). Improved API tps is to consider.
If 100K, use OIL for this. Improved OIL tps is to consider.
Thanks for clarifying.

Alain Heremans

Enterprise Architect for OnePAM & Global Customer Data Mgmt
Tech Strategy & Enterprise Architecture / Banking Technology
Amsterdam, The Netherlands
Mobile +32 477 414940
mailto: alain.heremans@ing.com

Table of Content

Note on using read APIs before uploading data

Except for –some- initial uploads we read the current attributes of existing Parties in OnePAM matching ingested Parties, using read APIs before uploading the changes:

- We don't know upfront what attributes changed. This way we know upfront we can compare which ones changed.
- **It is necessary for file-based uploads** that send events as acknowledgements as we must be able to account for all updates (before moving to the next step) so we know when all attributes have been updated or decide that errors happened (after a timeout) when no events arrived for changed attributes.
- **It is beneficial for API-based External LDD uploads** as you need multiple API endpoint update calls for a Party, and throughput is limited for writes. Knowing upfront exactly what attributes and which API endpoints to call reduces the number of calls.
- **It is necessary to know what External LDD Party attribute/relationships updates succeeded** or failed before moving on to the Main LDD as we only update Parties attributes and relationships for successful uploads
- **It is necessary when updating the Main LDD** as we must obtain the Main UUID, using LE #/Demographics/External UUID anyway, and must know whether the attributes are different to apply the right policies.
- Most Parties have relationships and some ingested records provide full sets (ag. legal representatives) rather than updates. We must look first what the existing set of relationships is for a Party in OnePAM and compare with the ingested to see which Parties and Relationships must be created, updated & deleted.

Read APIs are **not throughput constraint** (1000+ tps possible), so reading is relatively 'cheap'.

Given the multitude of scenarios where it is needed to have an upfront view of what the before-status is of Parties & Relationships we decide to

Retrieve the as-is view of Parties in OnePAM

OnePAM Fact & Search APIs are the mechanisms to obtain Party attribute as-is information.

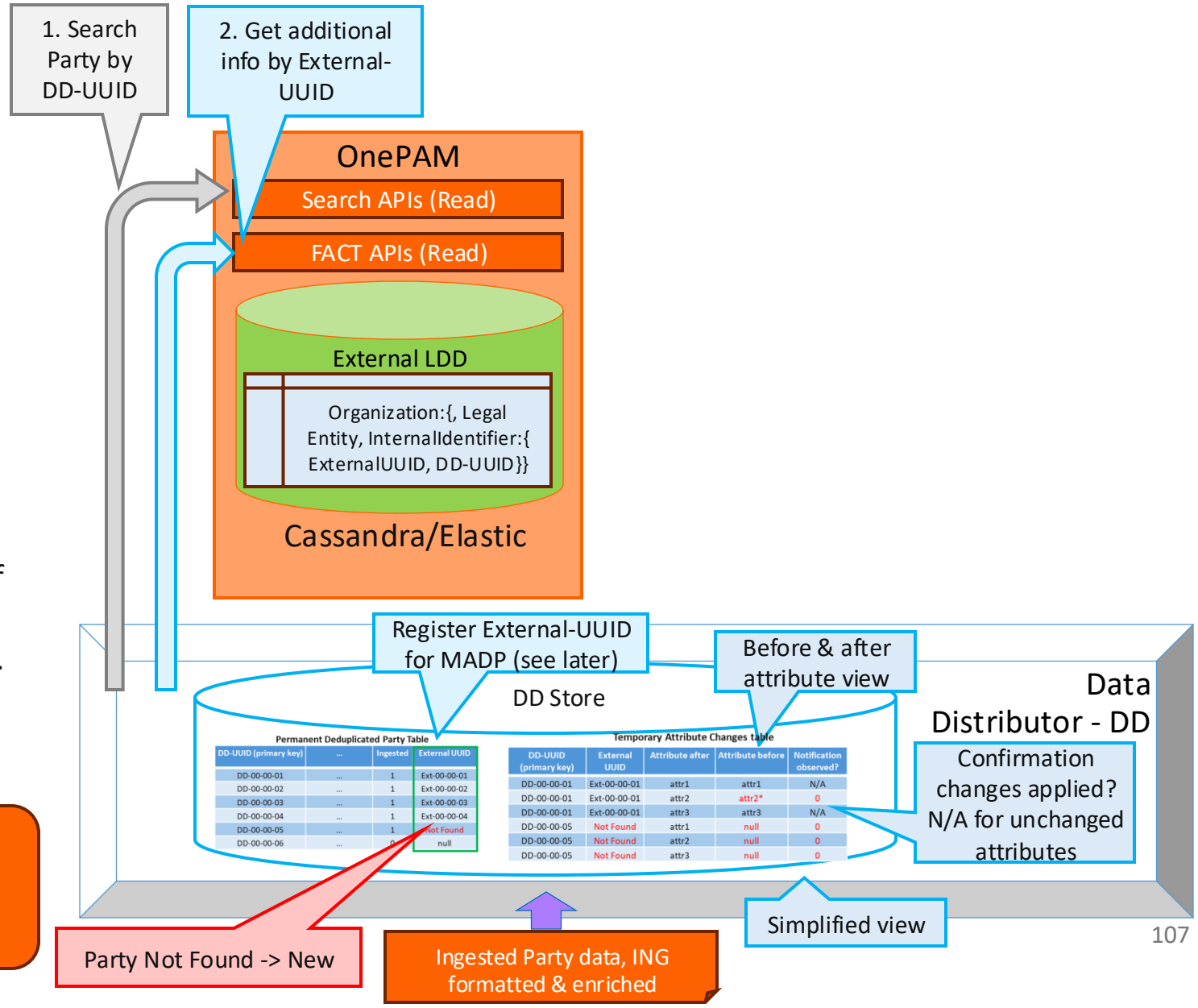
The 2 main APIs we will use are for OnePAM are:

- PartyAndAgreementSearch API to search
- InvolvedParty API to get more information

We **query the External LDD* using the DD-UUID** which is one of the *InternalIdentifiers* attributes that we register with each Party that we upload, starting with initial load onwards (see later).

This query does **not only return the External UUID** if the Party exists **but also a lot of additional attributes**. If we need to know more attributes, we make additional queries to the External LDD using the InvolvedParty API.

When querying MADP we'll use the APIs it exposes.



Remark:
Fact APIs will be replaced over time with Business Service APIs and this is one of the reasons (together with performance) we don't want to rely on them for updates

* We are inquiring about the read performance of fact APIs

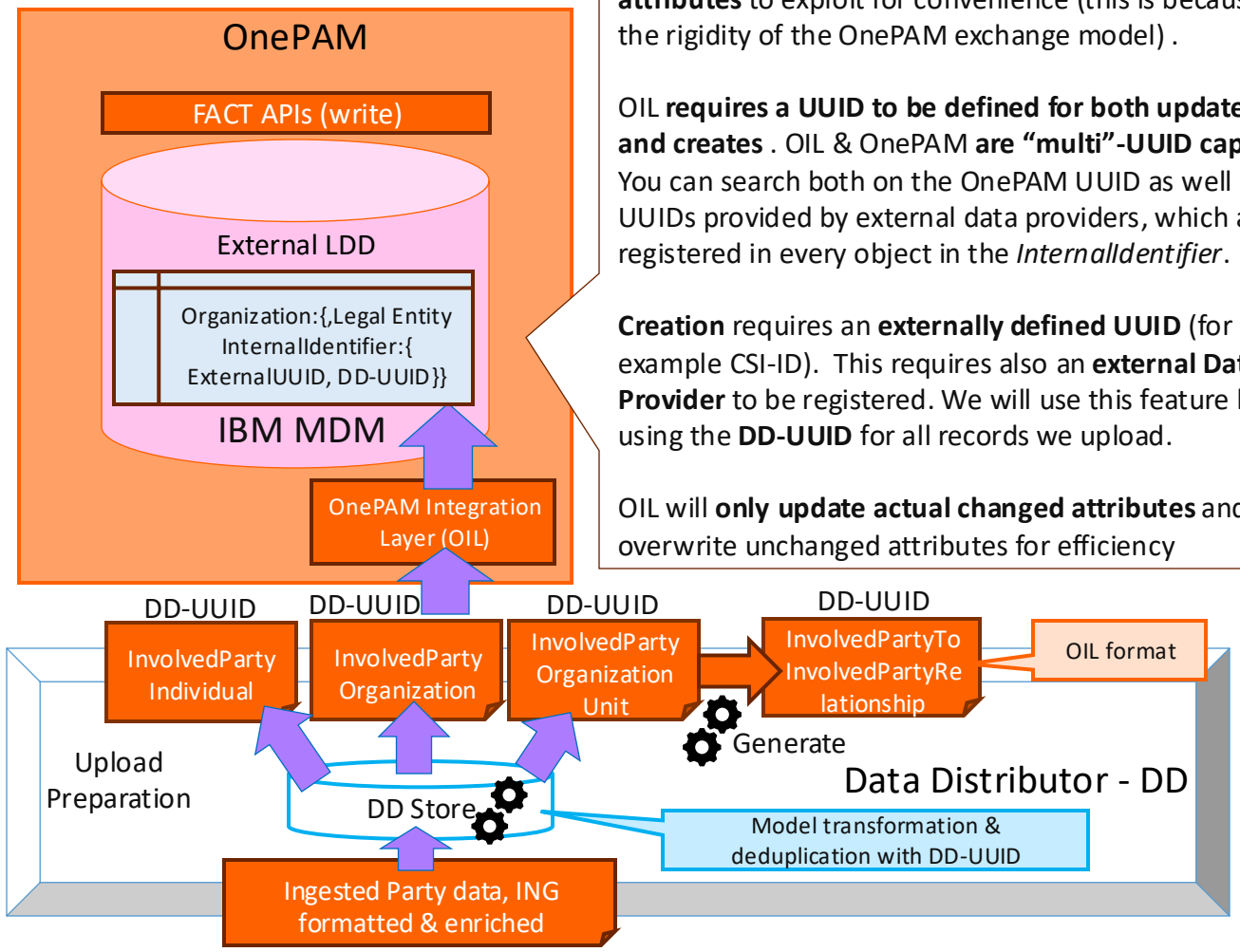
OIL as one upload/update mechanism for OnePAM

OIL is the **bulk data upload mechanism** of OnePAM. We will use this as one upload mechanism* the other being Fact APIs**.

OIL has different file formats for each type of entity according to the OnePAM exchange model Visio-OnePAM Data Model v5.4.0 (Final).vsdx:

- **InvolvedPartyIndividual**: for Private Individual parties like Incapables, Administrators, Legal Representatives and bankruptcy curators.
- **InvolvedPartyOrganization**: for Legal Entities
- **InvolvedPartyOrganizationUnit**: for Operating Sites
- **InvolvedPartyToInvolvedPartyRelationship**: for all relationships where Private Individuals and/or agreements are involved
- **OrganizationHierarchyRelationships:** between Legal entities and Operating Sites
- Others TBD – There are 10 OIL file formats in total.

We generate the different OIL formats from the **DD Store's Permanent Deduplicated Parties Table**, using the DD-UUID as unique identifier.



OIL requires files in a **particular format** for every type of Entity/relation. The format is prescribed and doesn't allow adding your own attributes and has **no flexible attributes** to exploit for convenience (this is because of the rigidity of the OnePAM exchange model) .

OIL **requires a UUID to be defined for both updates and creates** . OIL & OnePAM **are "multi"-UUID capable**. You can search both on the OnePAM UUID as well as on UUIDs provided by external data providers, which are registered in every object in the *InternalIdentifier*.

Creation requires an **externally defined UUID** (for example CSI-ID). This requires also an **external Data Provider** to be registered. We will use this feature by using the **DD-UUID** for all records we upload.

OIL will **only update actual changed attributes** and not overwrite unchanged attributes for efficiency

*The exception could be bankruptcies where we would use Fact APIs because of the time sensitivity of the data (2x per day) as OIL is asynchronous and non-deterministic

** OIL is file based and asynchronous and is made for bulk operations but does not provide feedback, Fact APIs are request/reply but have lower capacity (5tx/sec)

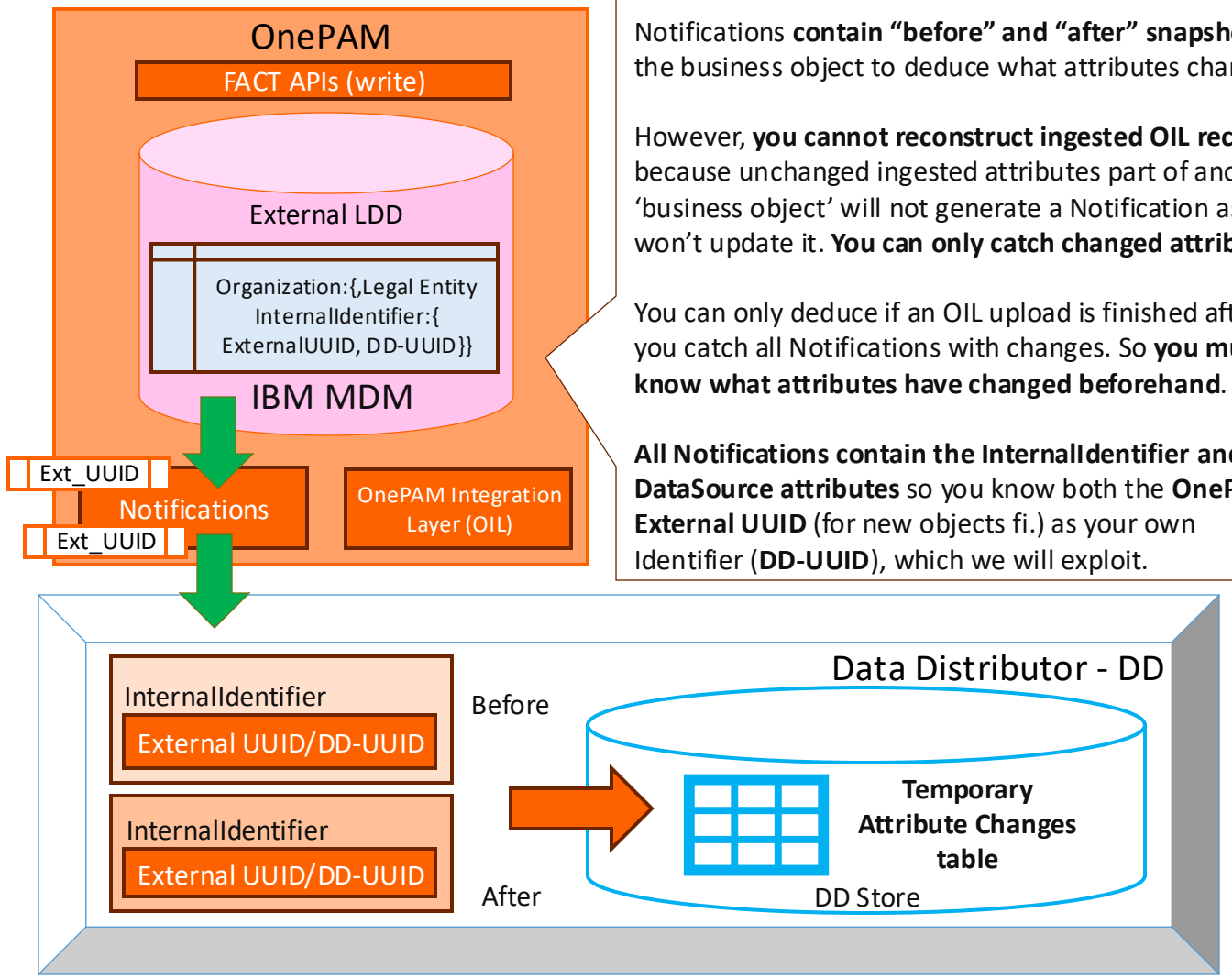
Notifications to inform OIL upload/update status

Notifications (events) are the main mechanism to inform external applications of changes in OnePAM data when using **OIL**. **Notifications** are generated whenever an attribute of an object changes.

Notifications group attributes in objects with a “Business” meaning:

- **NotifyIndividual5**: for Private Individual parties
- **NotifyOrganization5**: For Legal Entities
- **NotifyOrganizationUnit5**: For Operating Sites
- **NotifyAssociatedPartyRelationship5**: For relationships
- **Others**: NotifyPostalAddress5, NotifyDigitalAddress5, NotifyOrganizationUnitName5, NotifyIndividualName5, NotifyOrganizationUnitName5, NotifyCitizenship5, NotifyIndustryClassification5, … if individual attributes we capture change.

There are **30 “Business” Notifications in total**. They don’t follow therefore the OIL format* nor is there a direct link with the OnePAM table objects.



Notifications emit business objects, they provide all attributes of the business object, changed or not.

Notifications **contain “before” and “after” snapshots** of the business object to deduce what attributes changed.

However, **you cannot reconstruct ingested OIL records** because unchanged ingested attributes part of another ‘business object’ will not generate a Notification as OIL won’t update it. **You can only catch changed attributes**

You can only deduce if an OIL upload is finished after you catch all Notifications with changes. So **you must know what attributes have changed beforehand**.

All Notifications contain the InternalIdentifier and DataSource attributes so you know both the **OnePAM External UUID** (for new objects fi.) as your own Identifier (**DD-UUID**), which we will exploit.

* As there are only 10 OIL files, uploading an OIL file can generate multiple Notifications depending on attributes. Or zero if none of the attributes happen to have changed (only “Local” data in MADP changed)

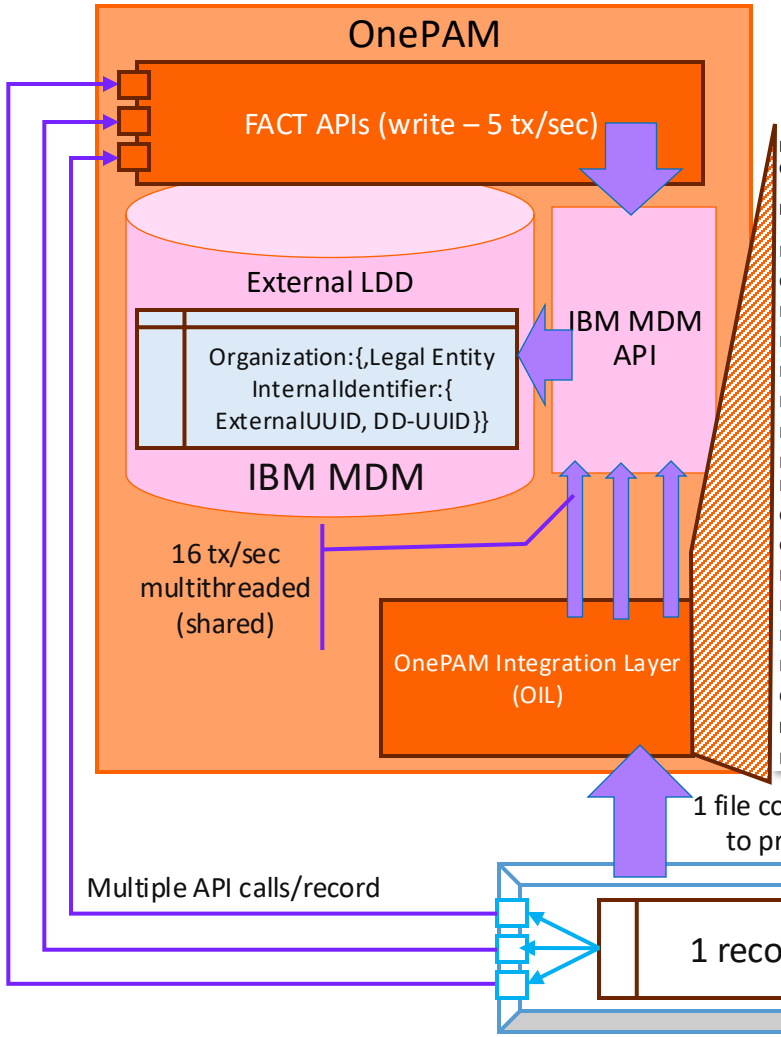
OnePAM API & OIL performance characteristics

API-based **updates** use FACT APIs.

- They are rate limited to 5 transactions per second per user as they are shared by everyone.
- Undercover they call IBM MDM interfaces (“call tx”).
- You might need to call multiple APIs to process a single record update depending on # changes.

OIL-based updates distinguish between daily updates and planned updates (like refreshes).

- Daily updates are:
  - processed on a single server shared by everyone
  - Rate limited by type of file as per the table
  - Executed on a single server
  - Update speed depends on type of file.
  - Schedules for & order of processing can be set (UAC)
  - Updates MDM using APIs at 16 tx/s (shared)
- Planned (on-demand) updates (like refreshes) are:
  - Executed manually (during week, business hours)
  - Use 4 servers
  - Fixed speed of 5 records per second



Performance (Records/secons – rps) for daily updates shared by all OIL users

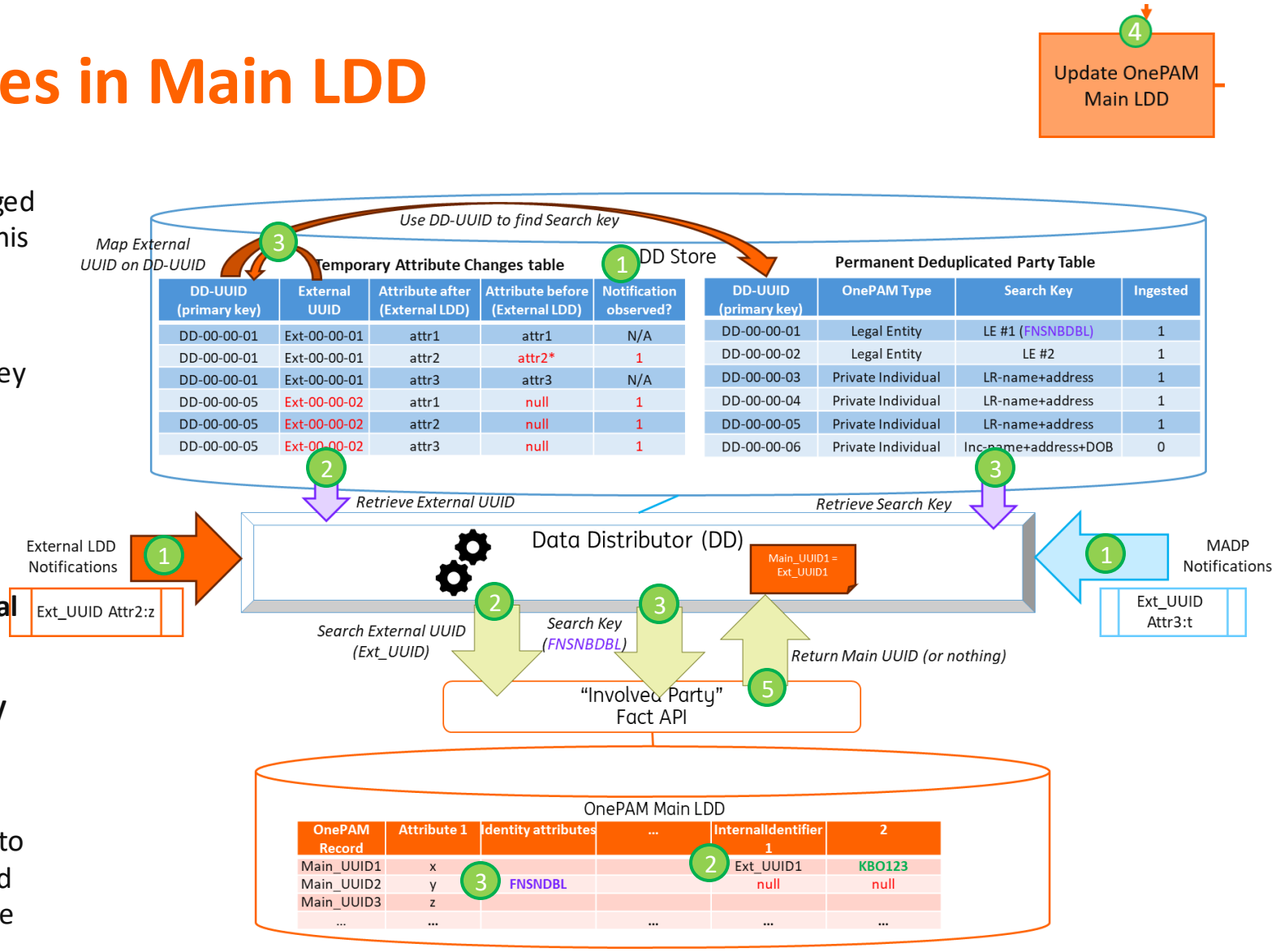
BASELINE Speed(Except NL, CSI)			
Pattern (in RPS)	**Day**	**Night**	**For Agr and Role Combined**
INDIVIDUALS (Parties)	30	30	NA
ORGANIZATIONS (Parties)	50	50	NA
IP2IP (Contracts	100	100	NA
MEANS_AGR_ROLES.	50	50	50
MANDATE AGR_ROLES	35	35	35
PRODUCT_AGR	50	50	20
DELETE INTERNAL IDENTIFIERS	30	30	NA
DELETE AGREEMENT IDENTIFIERS	25	25	NA
CLOSE ACCESS PROFILES	30	30	NA
CLOSE IP2IP	23	23	NA
DELETE AGREEMENT	25	25	NA
DELETE ACCESS PROFILE	30	30	NA
DELETE IP2IP	30	30	NA
DELETE PARTY	5	5	NA
ORGANISATION HIERARCHIES	70	70	NA
BASELINED Speed: RECON All Entities			
IND, ORG, IP2IP, AGR’s, ROLES	100	150	NA

The limitations in OIL & APIs have to do with MDM limitations as well as complex dependencies

Table of Content

Identifying Involved Parties in Main LDD

1. The **Data Distributor (DD)** receives **OnePAM Notifications** & **MADP events** to account for all changed attributes in the **OnePAM & MADP External zones.** This updates the "Notified?" column in the **Temporary Attribute Changes table.**
2. We search the **Main LDD** with the **External UUID** as key to see if there is a matching **Main UUID**.
3. If not, we use the associated **DD-UUID** to retrieve the **Search Key** (demographics or LE #) in the **Permanent Deduplicated Party Table** in the **DD Store**
4. We search for this key in the **Main LDD** or ask a **central disambiguation service** (not shown).
5. If we get a **Main UUID** returned it is an **Involved Party** that we must update.
   - **If not**, we must evaluate based on existence of an IP2IP Relationship in the **External LDD** if we have to create the ingested Party as an **Involved Party** and its IP2IP Relationship or if we don't have to update anything (not shown)



** It has to be refined what exact attribute (combination) can be used for Private Individuals/Legal Entities as unique identifier to search for UUID matches*

Bidirectional update or “Self-healing” in Main Zone

The Data Distributor propagates changes of ingested Parties to the External LDD. There are no other applications who update this data, so **the only changes are the changed attributes in the ingested files**.

But **to know what attributes changed we must read the current attributes** in the **External LDD** before we update them to compare the differences. We want to know this because some updates are conditional in the **Main LDD**.

In the Main LDD, other applications can also update attributes. For some (Operating Site for example) the ingested Party data is master, and **we must correct this when wrong, regardless if the attribute changed** in the ingested data or not (bidirectional or “**self-healing**” concept).

If we want fine-grained control **we must say for each attribute under what conditions we will update them** in the Main LDD:

- **NeverUpdate**: for attributes other applications are master
- **WhenChanged**: for attributes without a clear master
- **WhenDifferent**: when ingested Party data is the clear master
- **HumanProcess**: when changes in ingested Involved Party data must be updated manually rather than automatically (regulatory)

But **to know what attributes are different we must also read them** from the Main LDD **before updating** to compare them.

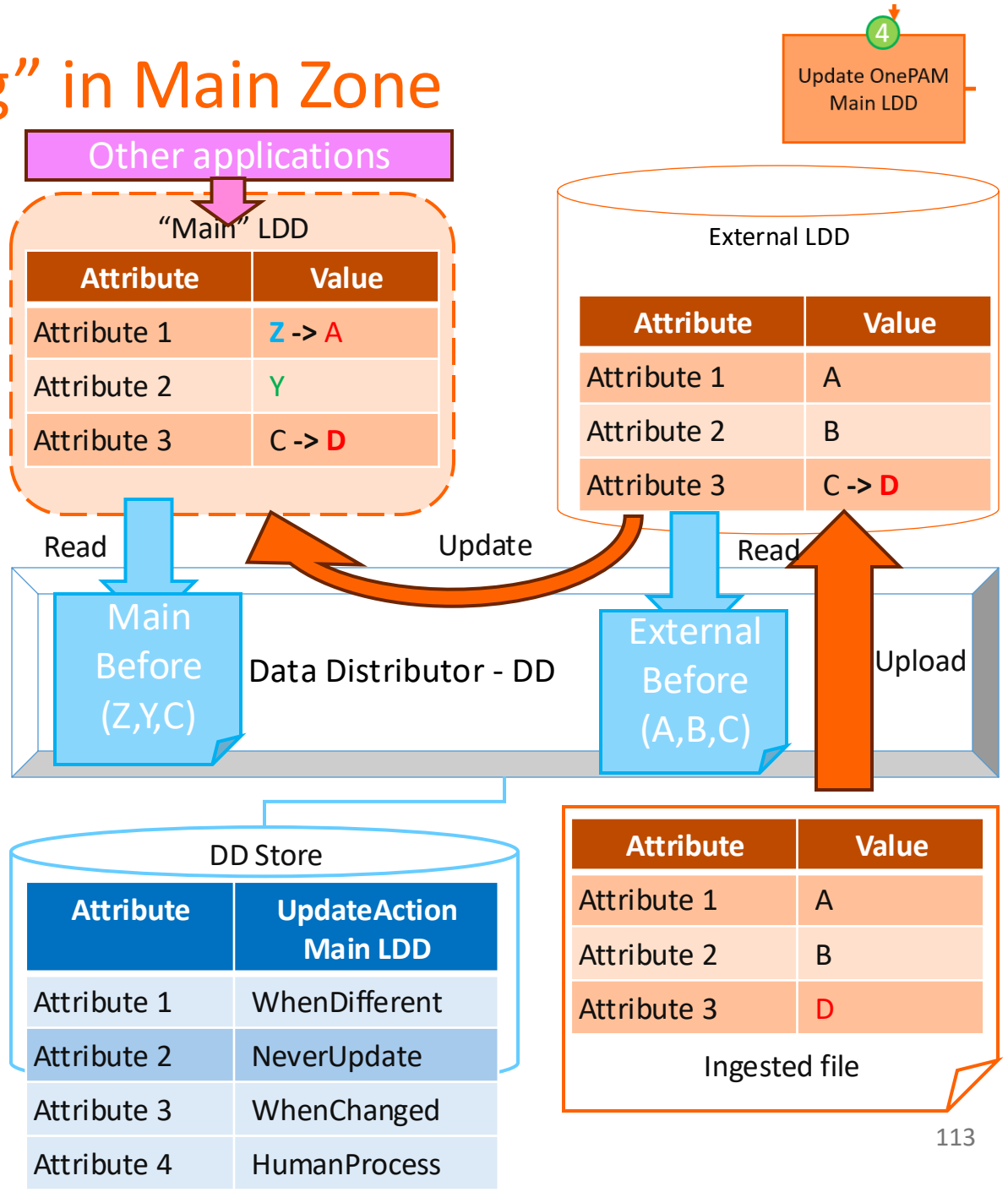
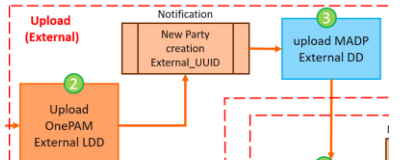


Table of Content

MADP requires a OnePAM UUID in its inputs



MADP is conceived as a **side-car** for **OnePAM** and cannot act as a standalone application. All objects in it are referencing a OnePAM object through its **OnePAM UUID.**

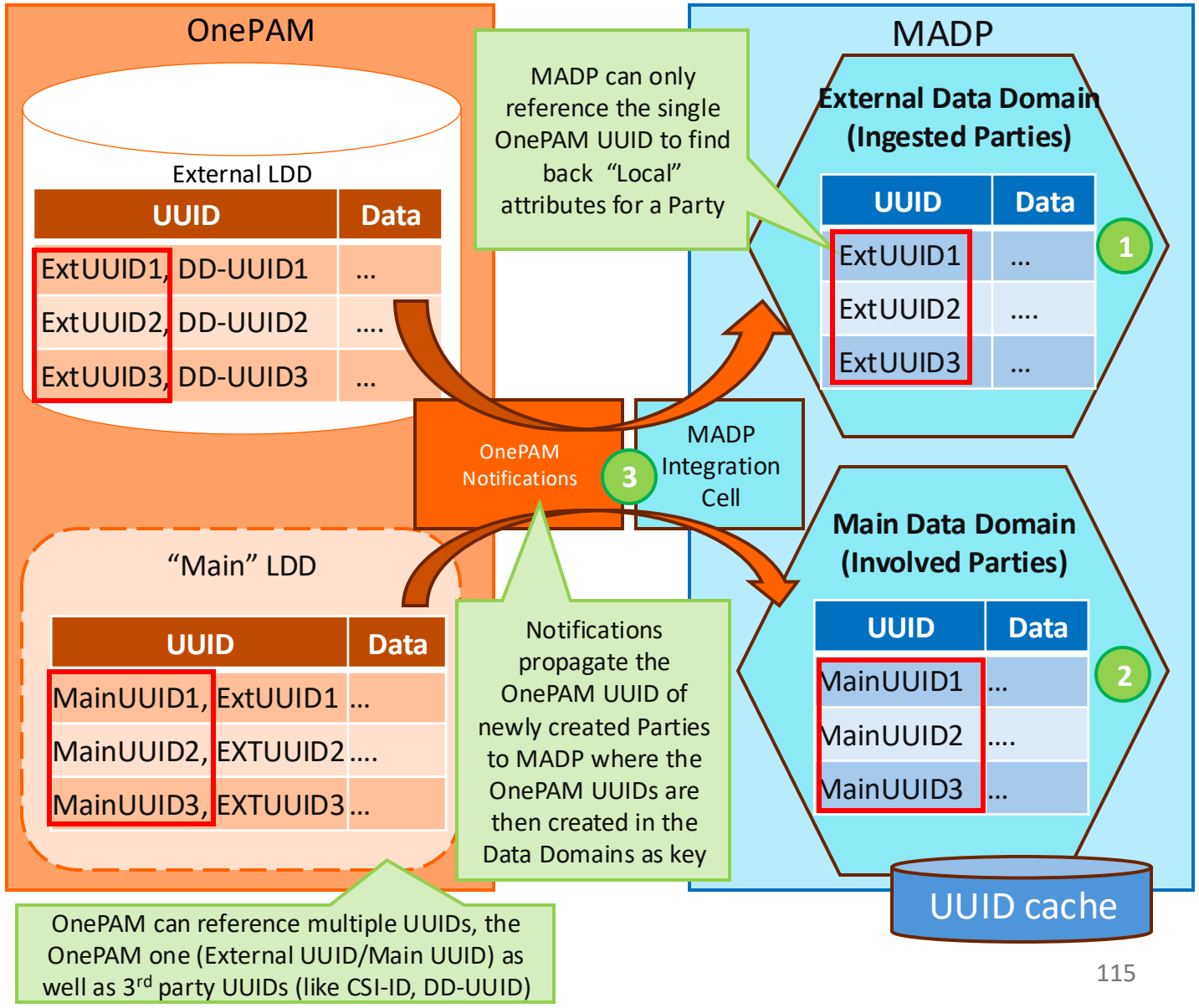
1. **External Data Doman**: extends (ingested) **Parties** in the External LDD with an External UUID
2. **Main Data Domains**: Extends **Involved Parties** in the Main LDD with a Main UUID
3. **Whenever a new object is created in OnePAM it generates a Notification** that **MADP** picks up and, depending on source (Main LDD, External LDD) it will create this new object UUID in the right Data Domain(s).

Therefore, we can **only update objects** in **MADP after we create them** (new Parties) in **OnePAM until after MADP has received the Notification and processed it**

- File uploads with not-yet existing UUIDs are cached
- API calls referencing them will fail.

This enforces an order of update in the overall flow.

MADP cannot reference a third Party UUID (like DD-UUID). That means that **before** uploading/updating a MADP object **you must know the right OnePAM UUID**. When creating new Parties in OnePAM **we first must collect & put the OnePAM UUIDs in the ingested records before uploading to MADP**.



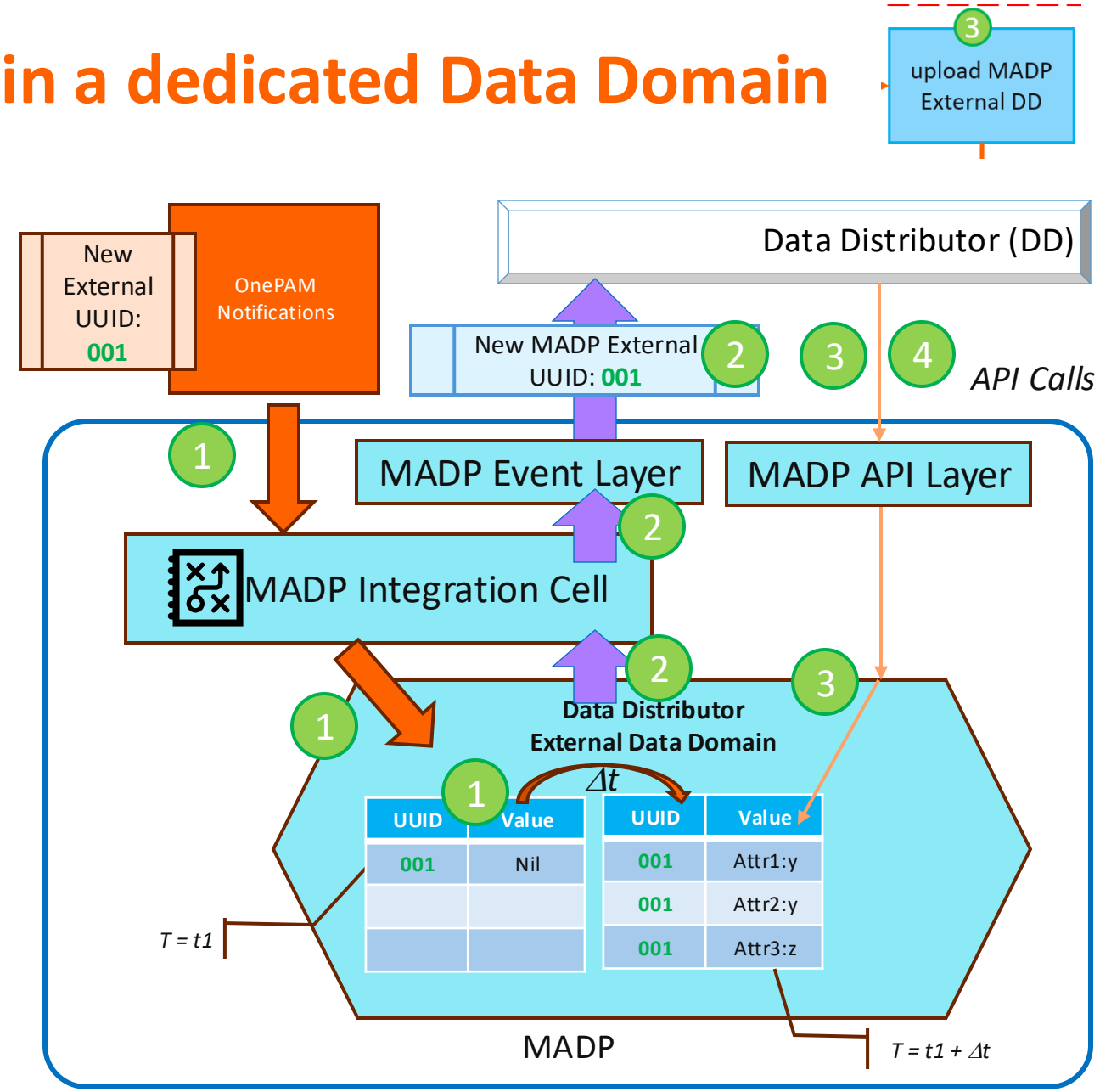
Loading Party “Local” attributes in a dedicated Data Domain (Example for APIs, new Parties)

1. When a new Party is created in the **OnePAM External LDD**, a ‘creation’ **Notification** is sent which is picked up in by the **MADP Integration Cell**. It will add an empty object with the new **External UUID** in the **MADP External Data Domain**.

2. The creation of this new External UUID in the MADP External Data Domain also generates a MADP ‘creation’ event on the MADP Event Layer, which is picked up by the Data Distributor.

3. The **Data Distributor** must wait until this creation event before it can update the **MADP External Data Domain** with the information of the new Party using APIs. If it doesn’t, the API call will fail and provide an error (polling is an anti-pattern).

4. For existing Parties (ie Parties the **Data Distributor** found in the **OnePAM External LDD** during the processing of the current ingested Parties) it knows that the **MADP** objects with the existing **External UUIDs** should have already been created, and it can just execute the API calls to update the attributes. APIs give the result immediately in the reply, so we know if all changes have been executed.



Loading Party "Local" attributes in a dedicated Data Domain (Example for Files)

1. The **MADP Integration Cell** picks up ingested Party "Local" data **changes** enriched with **External UUIDs** from a file provided by the **Data Distributor**, which is transformed into events in the **MADP Event Layer**.

2. We want to store all these "Local" attribute changes **exclusively** in the **MADP External Data Domain*** to make them available to Onboarding flows, like we do for ingested "Core" Party data changes in the **OnePAM External LDD.**

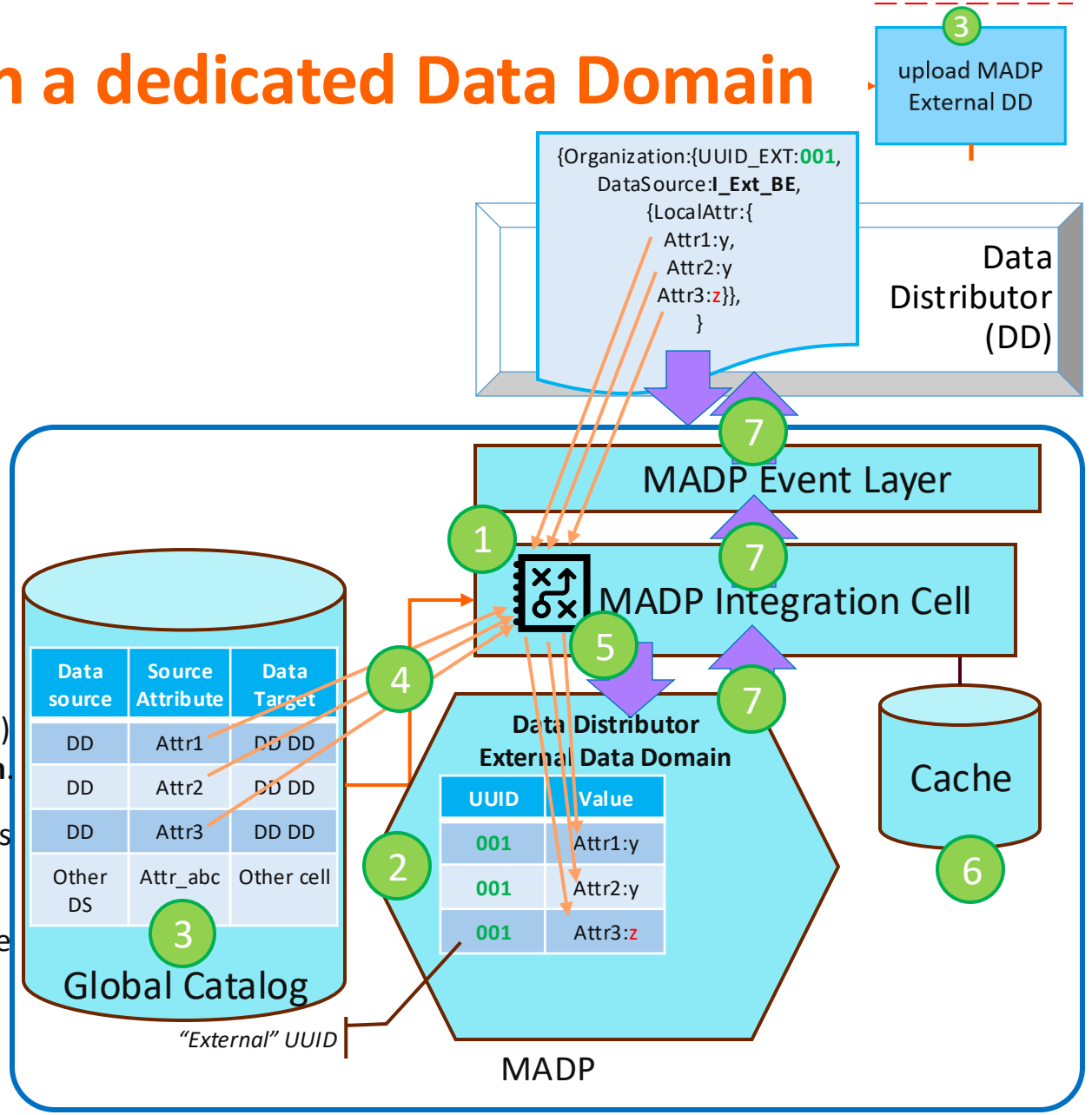
3. The **MADP Global Catalog** stores the schemas of all Data Domains so it knows the attributes in the file and the target Data Domains where they must be updated to.

4. The **MADP Integration Cell** uses this info** (Data Source & Attribute names) to route ingested "Local" Party attribute changes to the **External Data Domain**.

5. The **MADP Integration Cell** updates ingested "Local" Party attribute changes in the **External Data Domain** using the **"External" UUID**.

6. If the **External UUID** doesn't exist yet in the **External Data Domain**, then the **MADP Integration Cell** will cache the record and retry.

7. All updates generate **MADP** events captured by the **Data Distributor** to account for all attribute changes



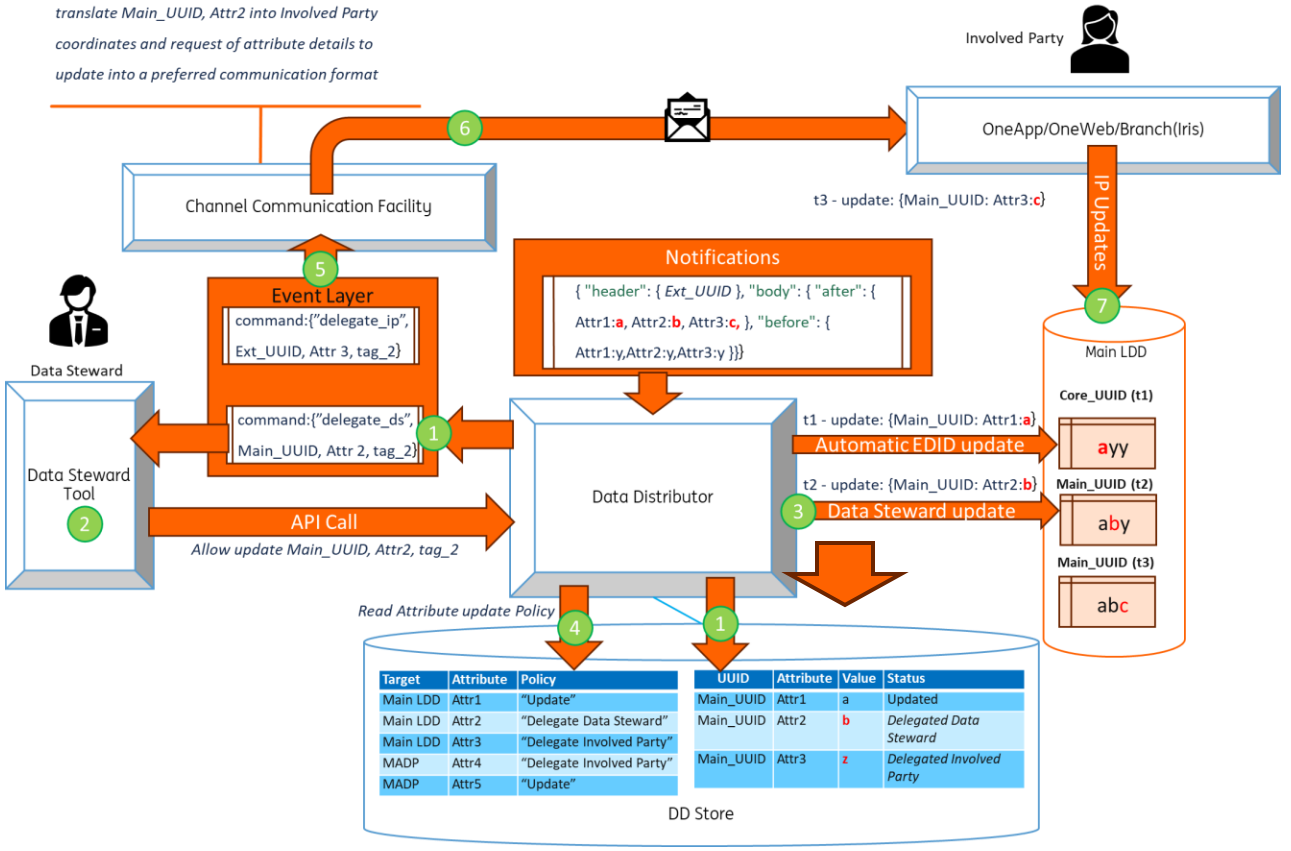
** External UUIDs are only created in the External Data Domain & Main UUIDs will only be created in other Data Domains.*
*** Details of the Global Catalog and the MADP Integration Cell routing requirements must be refined.*

Table of Content

Interaction with Human process

The **Data Distributor (DD)** will not always update the **OnePAM Main LDD** directly itself when it detects attribute changes*. For example:

- **It finds multiple Party matches** : when searching for an **Involved Party** in **OnePAM Main LDD** with demographics
    1. We store the different matches for the record in the **DD Store** and send an event to a Human Process for manual handling**.
    2. A *Data Steward* selects the correct **Involved Party** &
    3. The **DD** will update the **Main LDD**.
- **A Policy is set in the DD that an attribute must be updated by the Involved Party themselves**:
    4. because of regulatory requirements (fe. GDPR with Private Individual address change).
    5. The **DD** sends an event to the *Channel Communication Facility**** that
    6. informs the **Involved Party** to
    7. update their records in OnePAM through OneWeb etc.



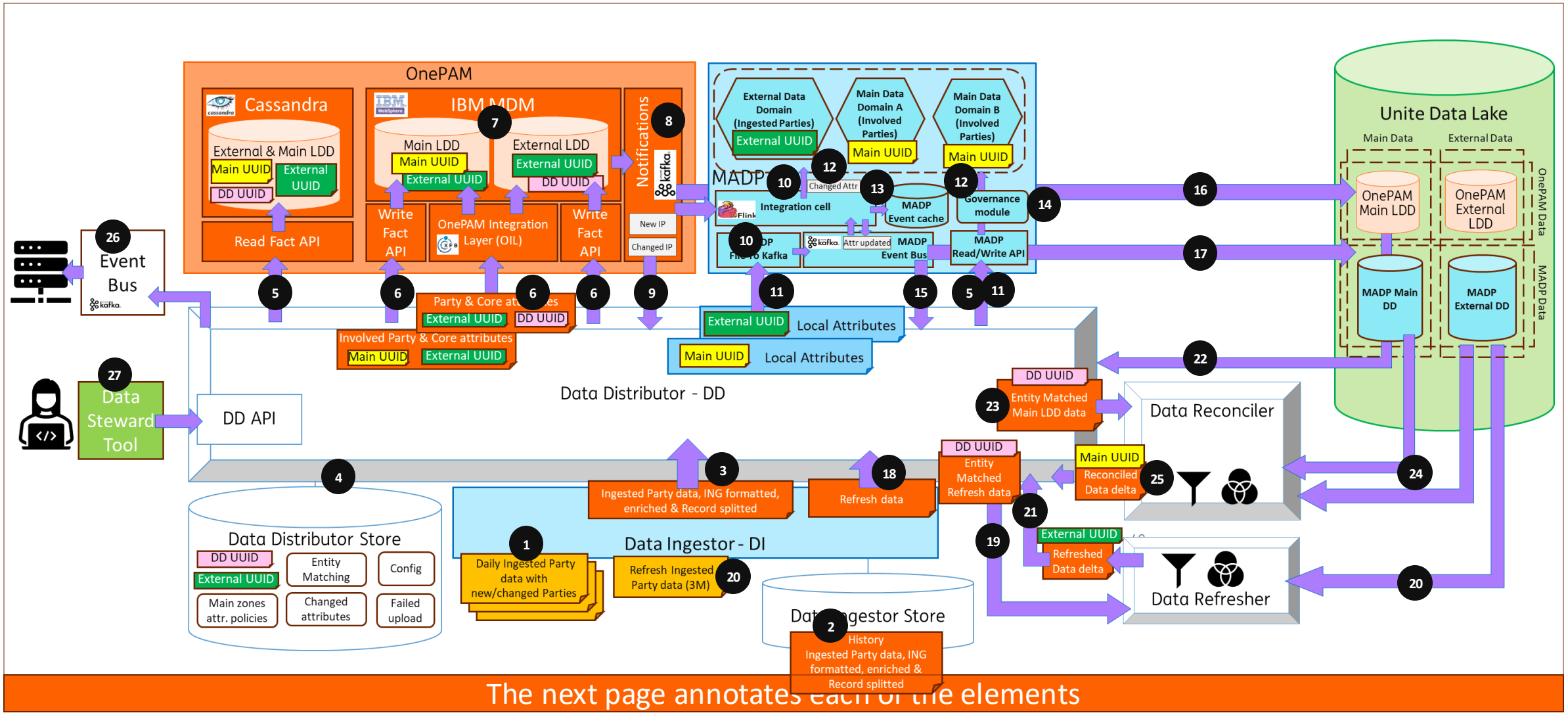
** That means that for each attribute there must be Policies on how to handle them by the DD. These policies are stored in the DD Store.*

*** There is a "disambiguation service" that is discussed as an automated way.*

****The Channel Communication Facility provides ways to inform Involved Parties, like customers, about actions they must take. This will be typically be a message in OneWeb/OneAPP. This will probably require the Involved Party to have an ING Profile, whether they are customer or not. In some cases, an ING employee will have to manage the request and explicitly contact the Involved Party directly (for example by letter).*

Table of Content

High-level End-to-End architecture landscape



The next page annotates each of the elements

High-level End-to-End architecture landscape - details

6. DD sends updated attributes, Parties/relations to be created/deleted to OnePAM either by file (OIL) or using Write Fact APIs.

5. DD reads from OnePAM and MADP the current value of Party attributes to compare with the equivalent ingested Party attributes.

26. DD can send events for other applications to react on

27. Data Steward tool interacts with DD/DRs through APIs.

4. The Data Distributor distributes the processed disaggregated records. It uses the DD Store to:
   - Keep configuration how to process each file
   - Manages local UUID creation
   - Does ingested Party entity matching with External ones
   - Identifies changed attributes & manages their updates
   - Maintains policies of which attributes to overwrite
   - Manages Failed uploads

7. Fact APIs or OIL updates changed attributes in the External or Main LDDs

8. All changes or new created Parties generate an event Notification with before/after information

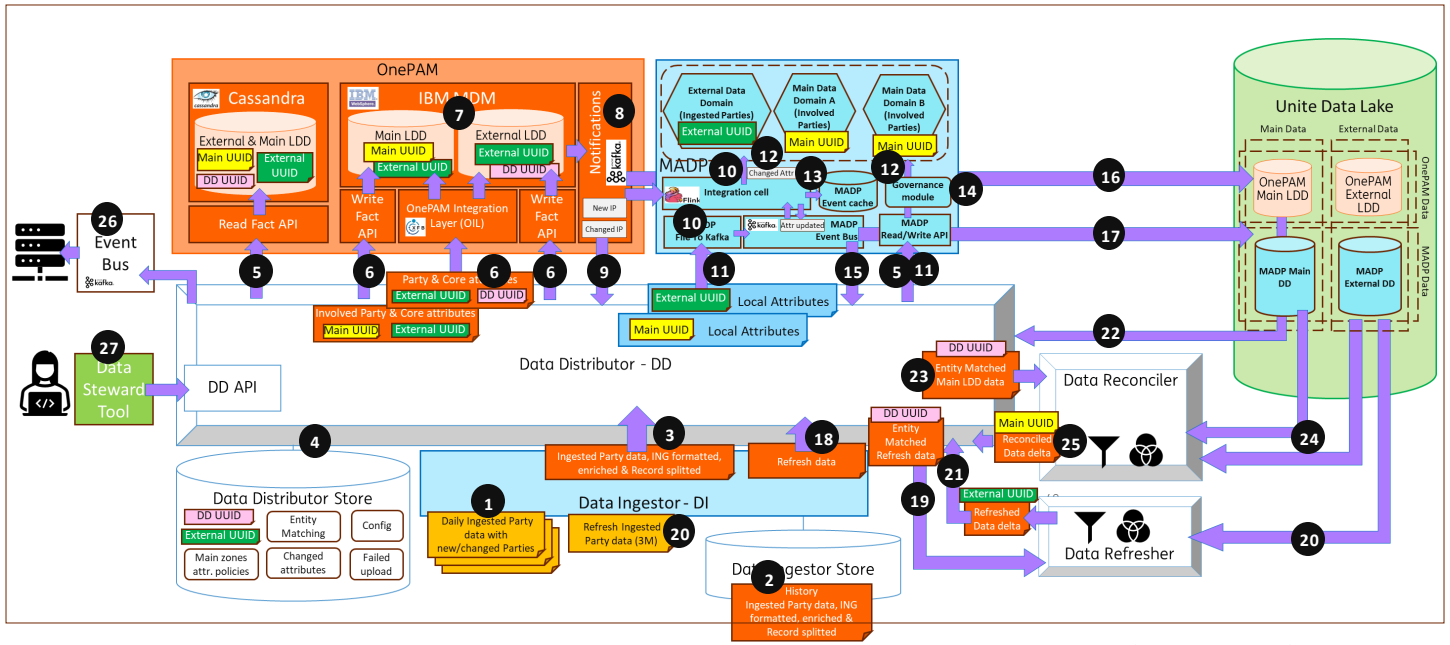
9. DD captures Notifications to confirm updated attrs.

10. MADP captures new Party Notifications & creates objects in the Data Domains using the UUID

11. DD sends updated Local attrs. to MADP by file or APIs

12. MADP Integration Cell updates the right Data Domain with the updates

13. MADP IC caches update events if objects have not yet been created



1. Multiple daily Ingest files (10's to 1000's of records)

20. Monthly full refresh data is ingested (~3M records)

2. After disaggregation & processing a full history is kept in the Data Ingestor Store

3. The daily disaggregated & processed records, with only changed attributes are sent as a file to the Data Distributor

18. DD ingests refresh data after processing by the DI.

19. DD entity matches refresh records and sends the results to the Data Refresher (DRF)

21. After processing the DRF sends the refresh delta back to the DD

14. MADP IC uses the governance module to route to the right Data Domain

15. DD captures MADP events to confirm updated attrs.

16. All OnePAM updates are sent to the Unite Data Lake

17. All MADP updates are sent to the Unite Data Lake

22. DD gets a full extract of the OnePAM Main LDD from the Unite Data Lake.

23. DD entity matches Main LDD extract and sends it to the Data Reconciler (DRC)

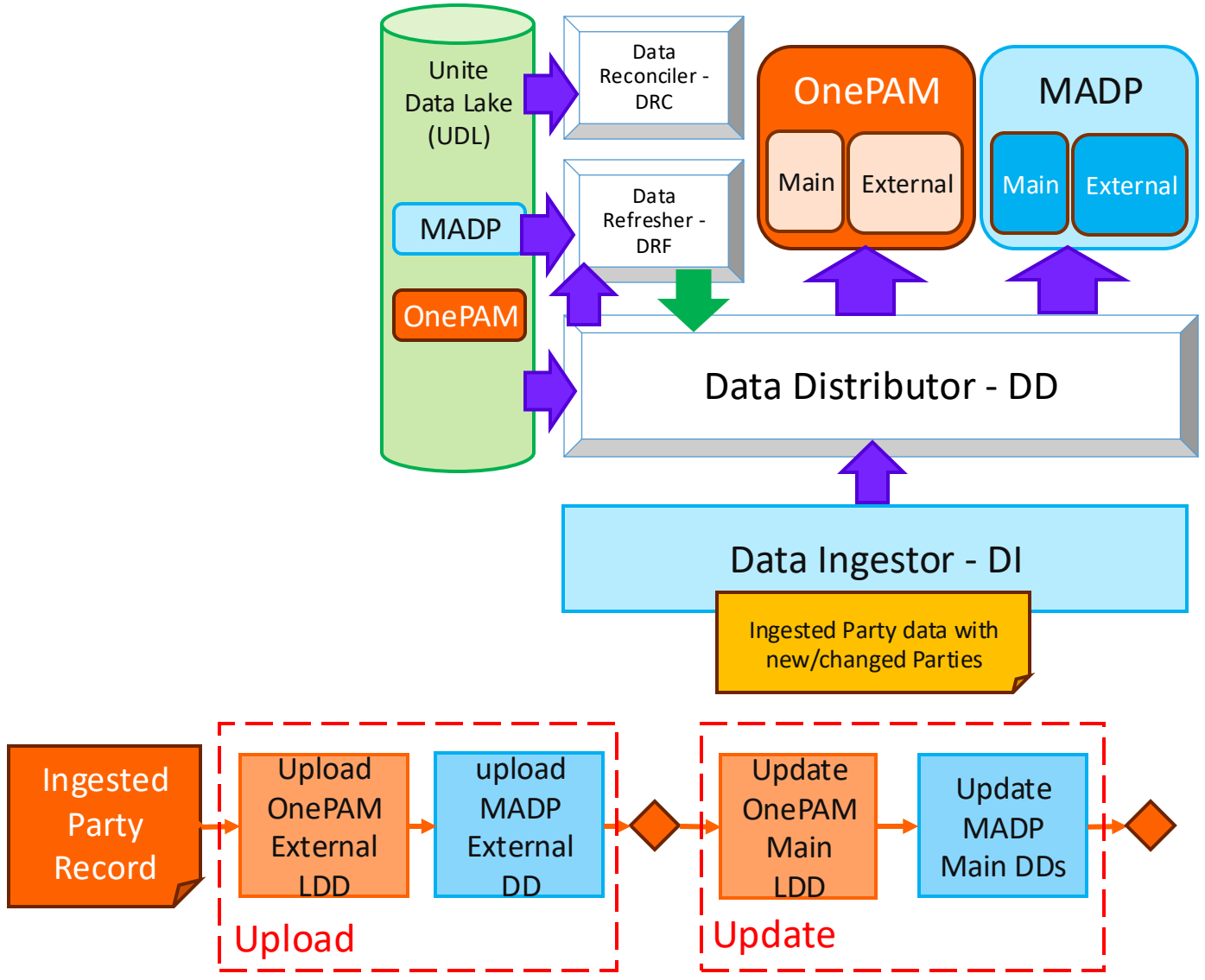
24. DRC gets full extracts from UDL of the OnePAM External LDD and MADP.

25. After processing the DRC sends the reconciliation delta back to the DD

20. Data Refresher gets a full extract of the External Data (OnePAM & MADP) from the Unite Data Lake.

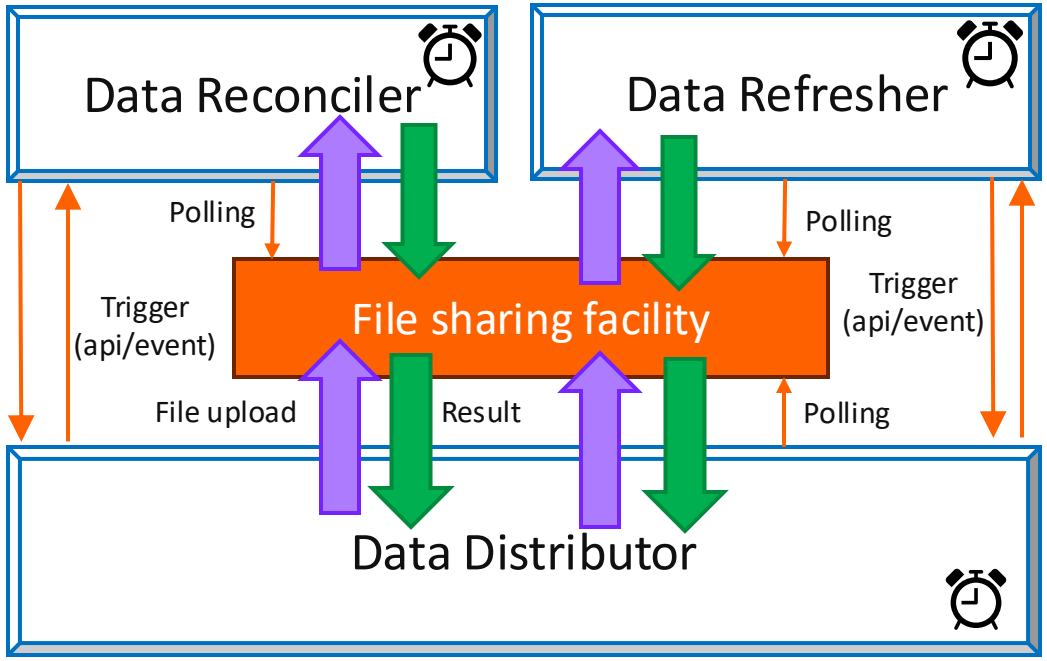
Thank you

Summary of this presentation



Data Distributor integration choices with reconciler/refresher

File-based Integration



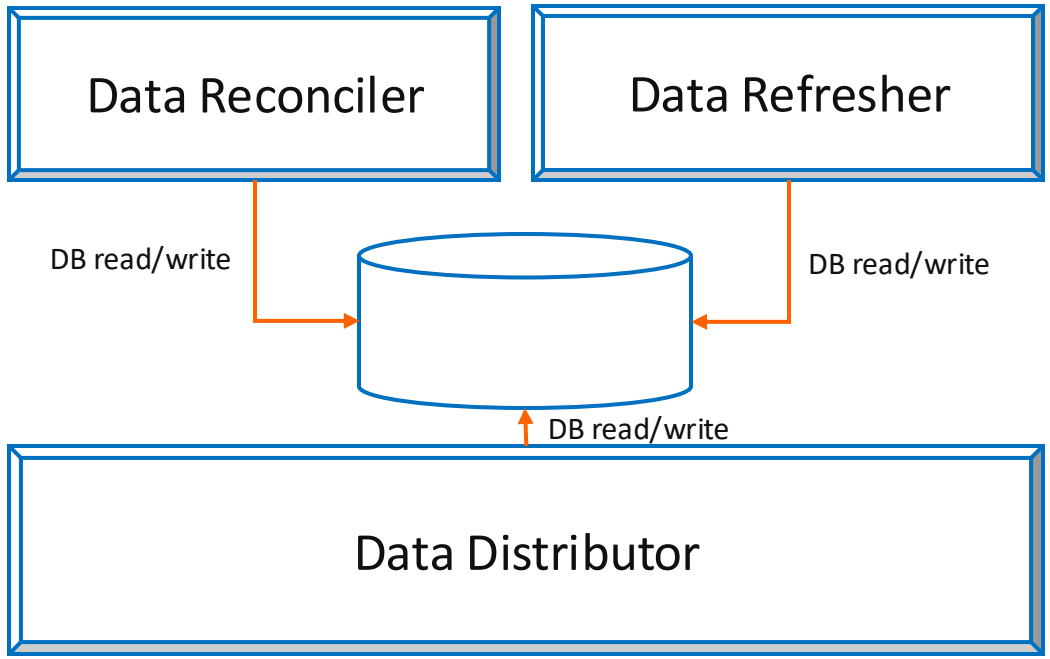
Pro:

- Easy to do.
- If different implementations, preserves loose coupling
- Easier process only once (? Merak library functionality ?)

Con:

- Requires polling (wasteful), triggering (complex) and/or scheduling (imprecise, increasing throughput time)
- Multiple database implementations to maintain
- If different implementations, requires more deployments

Database-based integration



Pro:

- When same implementation, less compute resources
- Heartbeat/trigger ability through database (cheap)
- Only one database implementation needed
- Lower throughput times

Con:

- Anti-pattern (hard coupling) if different implementation
- Possible database contention if parallel processing
- It's still polling (db pooling makes it cheaper)
- Mutex mechanism to ensure process-only-once