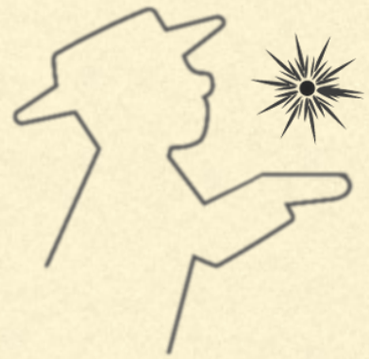


AMPEL

Alert **M**anagement, **P**hotometry and **E**valuation of **L**ight curves.

AMPEL DESIGN GOALS

SELECTION

- Flexible framework for selecting potentially interesting transients from stream

- The same transient can be selected via different channels, with different data rights and science goals

- Selection can be made based on cross-correlation with external data

AMPEL DESIGN GOALS

TRACKING

- Selected transients are followed
  - New observations are appended to tracked objects
  - Potentially include updated zero-points, updated subtraction or extraction
  - Incorporation of observations not yielding alerts

AMPEL DESIGN GOALS

SCIENCE

- Computation of additional information
  - photo-z
  - photometric type probabilities
  - Light curve fit
  - AGN proximity
  - …
- Custom science modules submittable

AMPEL DESIGN GOALS

TASKS

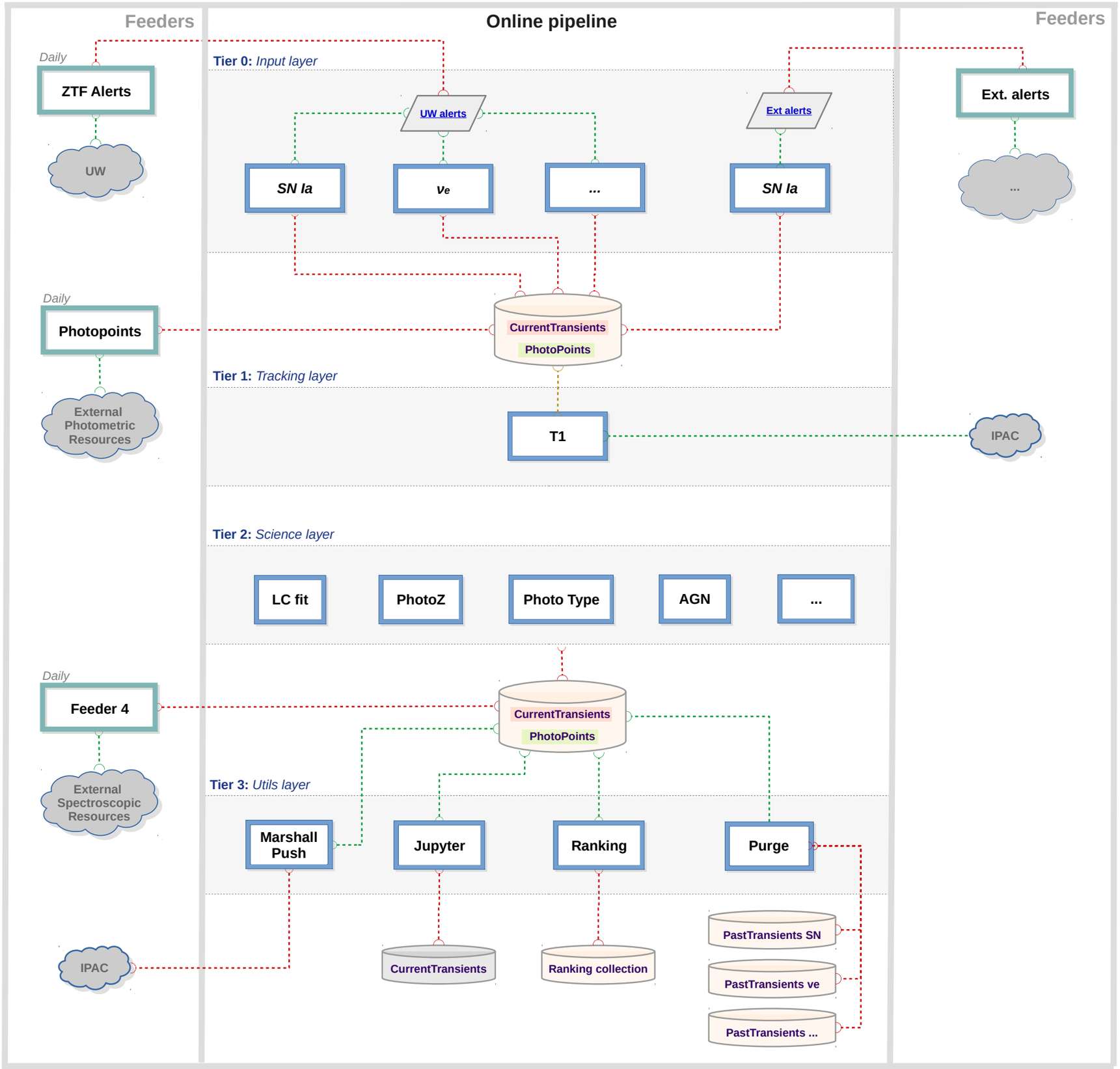
- User specified automatic candidate ranking and action
  - *"Send circular for any transient with z < 0.03 host galaxy"*
  - *"Export and push a list of all transients visible from Palomar tonight, brighter than 18.5 and ranked inversely by their predicted brightness in one week"*

AMPEL DESIGN GOALS

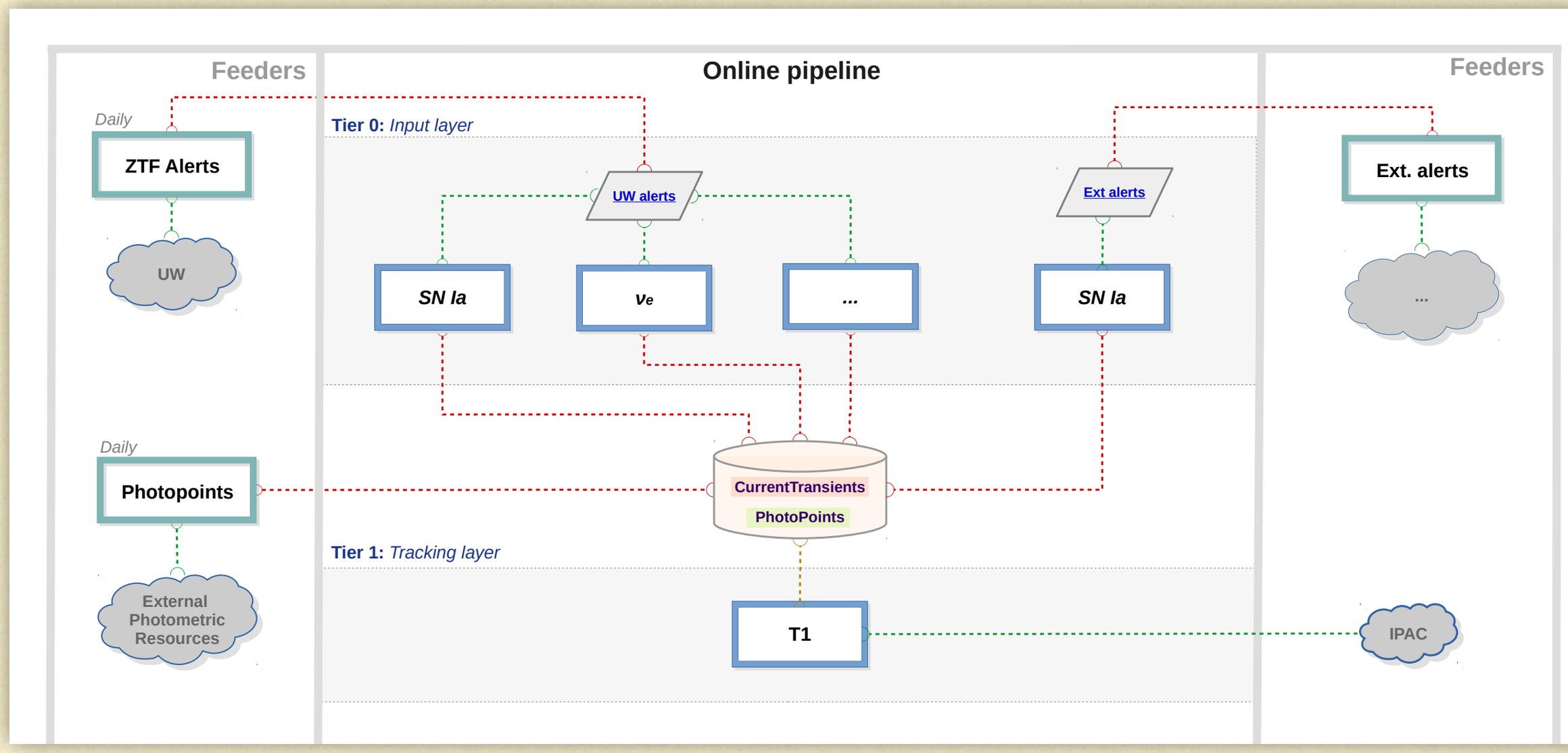
REPLAYABLE

- Available as re-executable container
  - Convenient detection *replay*: "*how would the transient sample have changed with different parameter settings*"
    - *"Rerun ZTF 2019 and provide photometric types of all transients in galaxy cores*"
    - "*Rerun ZTF 2020 based on the full set of low significance LIGO alerts – which transients would have been associated with these and what are their photometric types*"

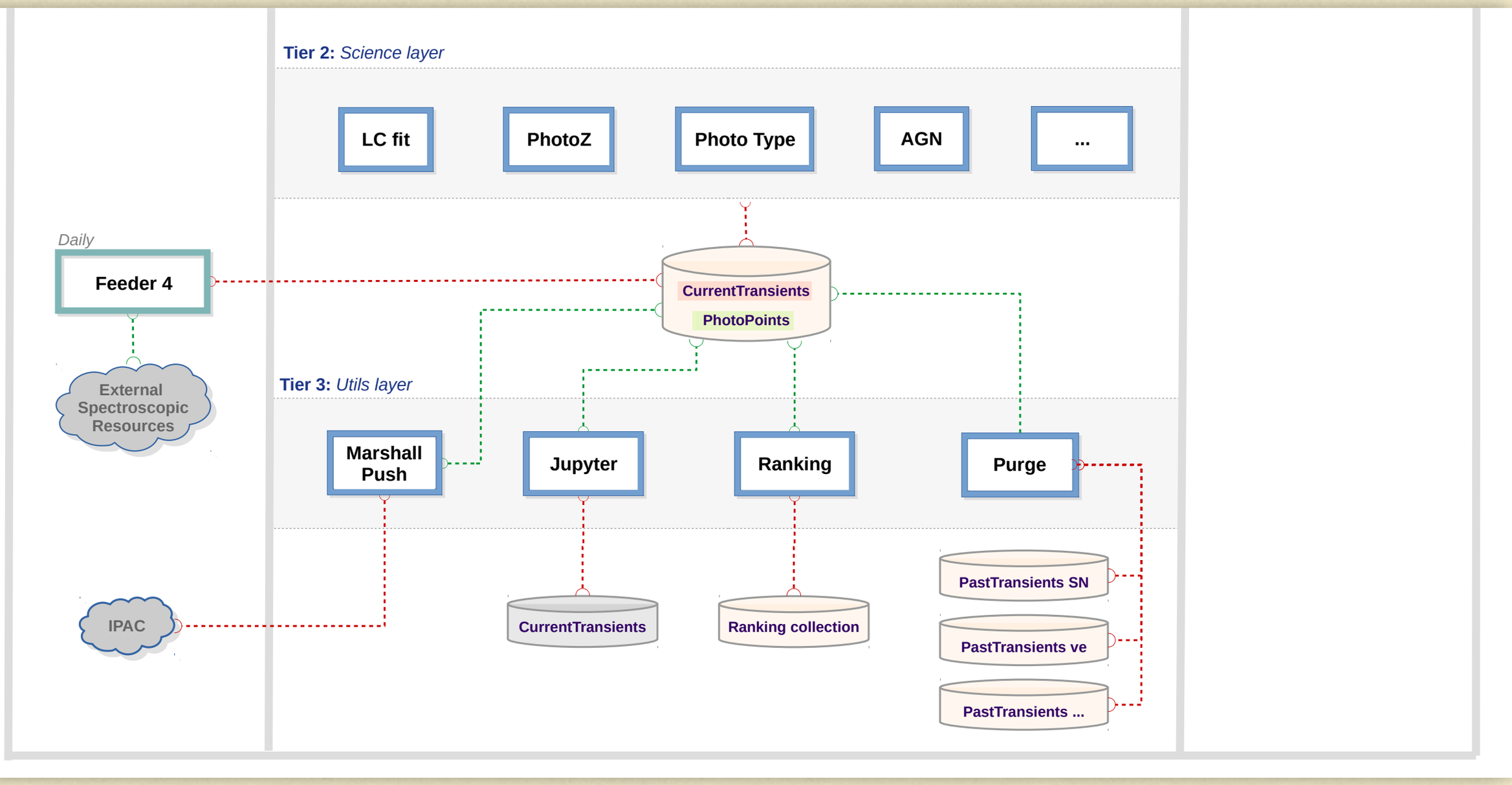
AMPEL
OVERVIEW



AMPEL
OVERVIEW



AMPEL
OVERVIEW



T0 FILTERING

- T0 is not the place for heavy computing (that's what T2 is for)
- Example: filter alerts based on:
  - rb value
  - magpsf value
  - number of photopoints in given bands
  - Coordinates of the transient
  - Any combination of the previous points

USER SCENARIO
(BASIC)

- Implement your T0 filter
- Provide a:
  - list of T2 modules to run
  - list of T3 modules to run

USER SCENARIO
(INTERMEDIATE)

- Implement your T0 filter
- Define a list of T2 modules for your channel
- Define custom parameters for your selected T2 modules
- Implement a T2 module tailored for your science goals
- Provide custom parameters for the T3 ranking module

USER SCENARIO
(ADVANCED)

- Implement your T0 filter
- Define a list of T2 modules for your channel
- Define custom parameters for your selected T2 modules
- Implement a T2 module tailored for your science goals
- Provide your own T3 modules (ranking, alternative subtraction, push to external systems)
- Provide custom purge strategy

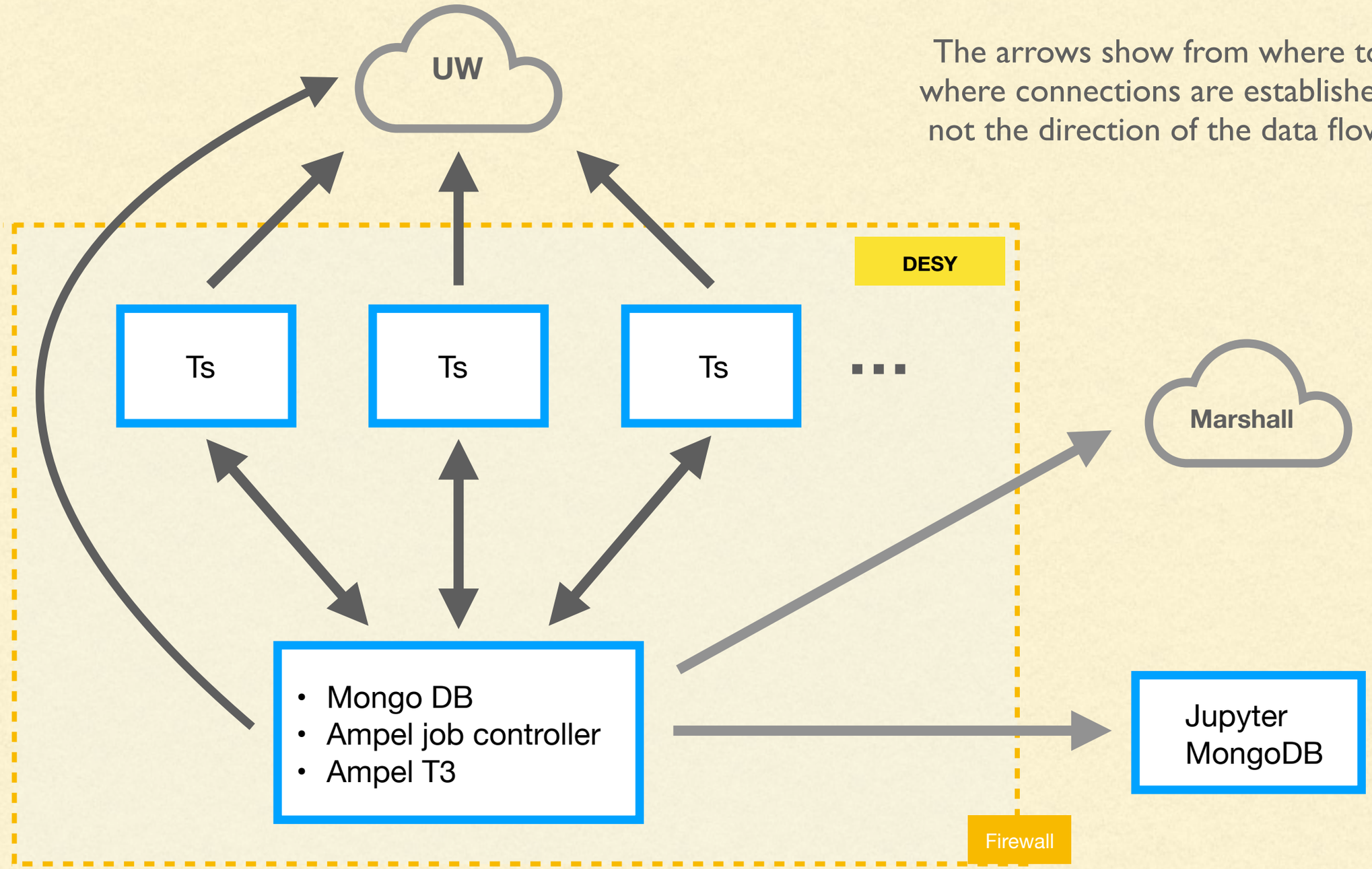
NUGENS SCENARIO
(CUSTOM)

- Implement T0 filter
- Define a list of T2 modules with custom parameters to run
- Provide following T3 modules:
  - M1: Push transients to NERSC if results from T2 science modules meet given criteria
  - M2: Pull results of heavy computations performed at NERSC back to Ampel DB

CHARACTERISTICS

- Python
- Uses MongoDB
- Jupyter access foreseen
- Will run on the DESY grid

DESY ARCHITECTURE



The arrows show from where to where connections are established, not the direction of the data flow

AMPEL DESIGN GOALS

Ampel goals
Pipeline
Multiprocessing capability
Ability to process non-ZTF alerts
Full history
Manually trigger operation
Modular structure
Accommodate different science cases (different filters and science modules)
Alerts
Alert order of no consequence
Robust against duplicated alerts
Detect reprocessed photopoints
Photopoints
Optional additional instruments
Manual exclusion
Optional alternative subtraction
Append zeropoint

Ampel goals
Science modules
Mild restrictions on the output of science modules
Allow variable module parameters
Share results between channels (put differently: compute science modules only once!)

WHICH DATABASE ?

- Requirement: *mild restriction on the output of science modules*

  → **Flexible DB Schema**

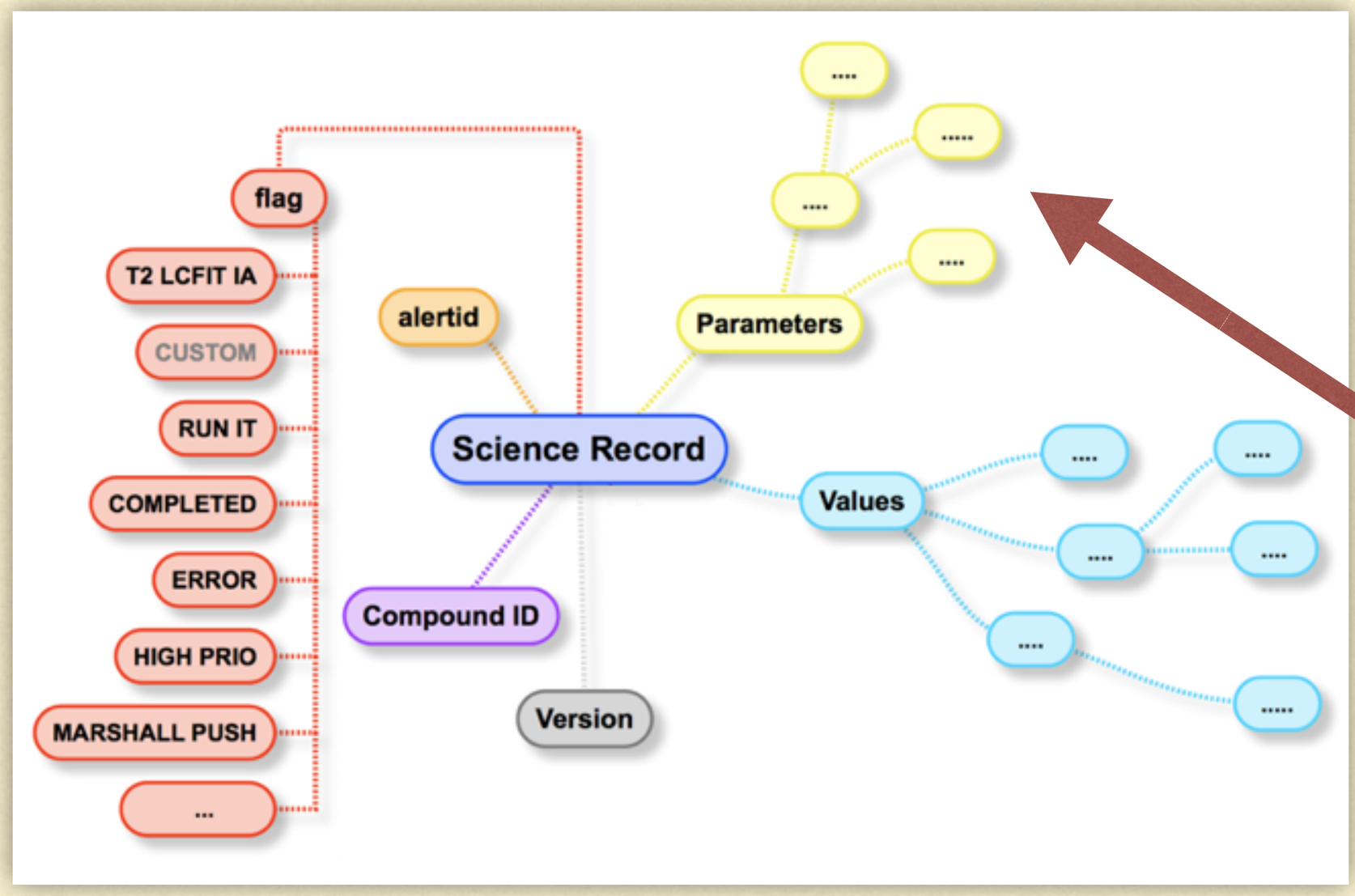
- *MongoDB* was chosen:
  - NoSQL database (Stands for *No*n SQL by means of “non relational”)
  - Document-oriented (a subclass of key-value databases)

- uses JSON documents binary-encoded in the BSON format

```
{
    "_id" : 1000020,
    "myDict": {"a": 1, "b": 2},
    "myArray": [1, 2, 3]
}
```

MySQL	NoSQL
Table	Collection
Row	Document
Column	Field

NOSQL DYNAMIC SCHEMA



Flexible data model
database schema
can evolve over time

MONGODB

- Robust Python interface (pymongo)
- Excellent scaling capabilities (that we do not plan on using yet but it is reassuring)
- Geospatial indexing allowing location based queries
- Strong community
- Rich query language

MySQL	NoSQL
SELECT * FROM users	db.users.find()
INSERT INTO users (user_id, age, status) VALUES ('bcd001', 45, ‘A')	db.users.insert({ user_id: 'bcd001', age: 45, status: 'A' })

STORAGE ENGINE

- *MongoDB* supports multiple storage engines
- Responsible for managing how data is stored (both in memory and on disk)
- Ampel uses *WiredTiger*:
  - Supports native compression (default: snappy)
  - Document-level lock
  - Scales on multi-CPU architectures

INDEXING

- Indexes enable efficient execution of queries

- They are special data structures that store the values of selected fields in an easy to traverse form

- Without indexes, *MongoDB* scans every document in a collection, to select those documents that match the query statement

  On a laptop with a fast SSD, querying one photopoint out of 10M takes:
  - more than 3 mins without indexing
  - less than 1 ms with indexing

- Performances are best if indexes fit in RAM.
  WiredTiger compresses indexes by default
  Practical example: 10M photopoints, 3 indexed fields -> 250 MB RAM usage

DESIGN STRATEGIES

- Modeling Ampel data as documents is challenging
- Meeting best practice recommendations is difficult since those can conflict with each other
- Goals while designing the Ampel collections:
    - Limit the number of queries required for tasks
    - Avoid I/O bottlenecks
    - Optimize RAM usage
    - Prevent race conditions
- The schema of the *PastTransients* collections will differ from the “online" collections as fewer constraints apply.

AMPEL SCHEMA

Core collections

Photopoints

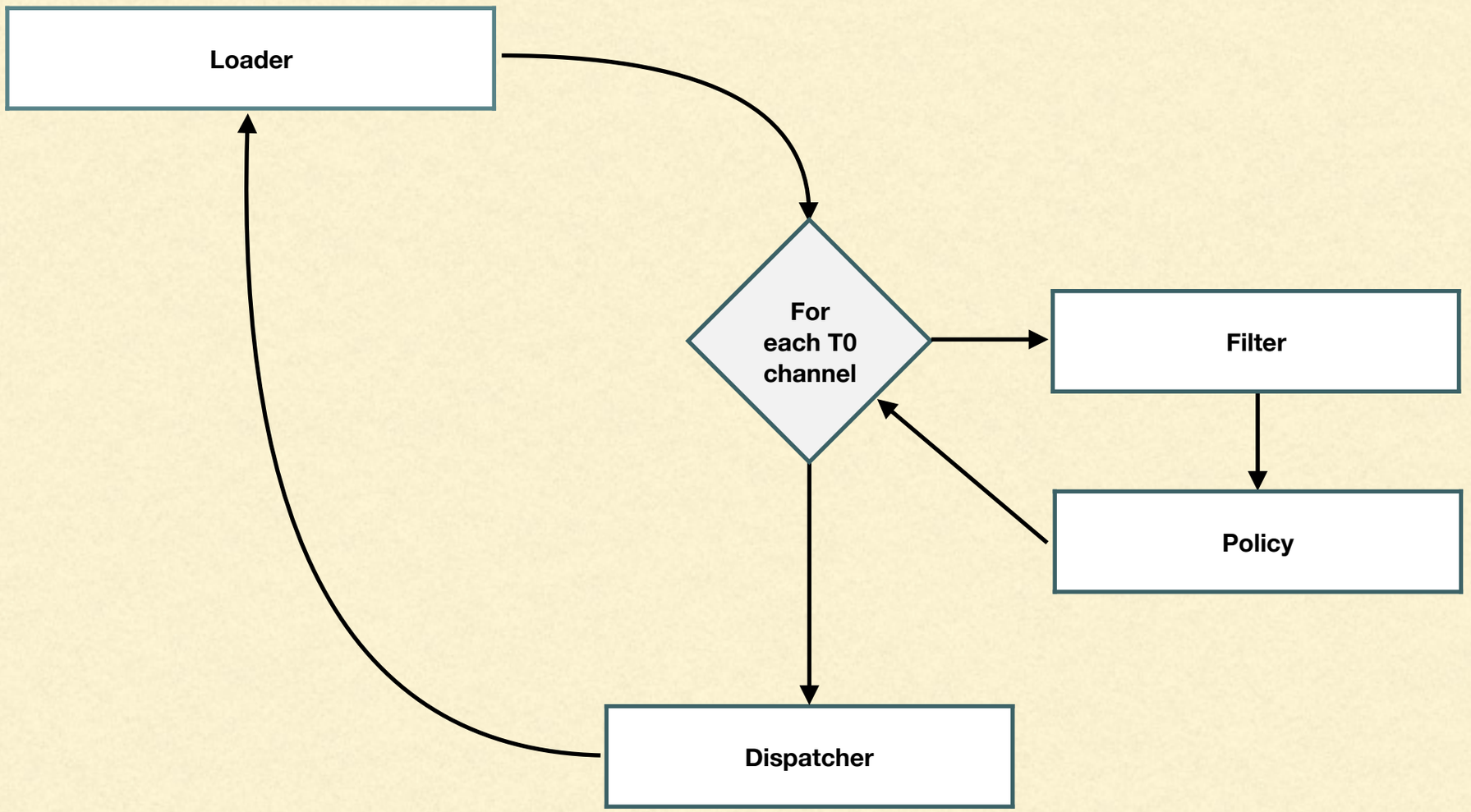
Transients

T2

Utility collection

Events (Jobs and Logs)

T0 PROCESSING



AMPEL STATUS

- Structural work is completed
- Alpha T0 functionality, lots of coding still required
- Code pushed to *AmpelProject* GitHub:

  https://github.com/AmpelProject/Ampel

WHAT'S NEXT ?

- T0 channels implementation (should be straightforward)
- T2 modules implementation or integration
- T3 modules (mix of individual and group efforts)
- Setup docker environment
- Setup main Ampel server
- Implement Ampel job scheduler
- Setup external Jupiter server
- Live testing when UW test alert stream is ready

BRAINSTORMING

- Define what fields require indexing
  Put differently: what efficient queries do we need for T2 and T3

- What to do with rejected alerts

THANK YOU