AMPEL

1. Motivation
2. Concepts
3. Selected tools
4. Lessons Learned

I would like...

High-throughput

Real-time

Large sample

Fast response

More pixels,
more dishes

Repeatability

FAIR

Have only one
Universe

Why did?

Test statistic

Green
computing

Flexibility

Software
development

Statistics

Big Data

Machine Learning

I would like...

High-throughput

Real-time

Large sample

Fast response

More pixels,
more dishes

Repeatability

FAIR

Have only one
Universe

Why did?

Test statistic

Green
computing

Flexibility

Software
development

Statistics

Big Data

Machine Learning

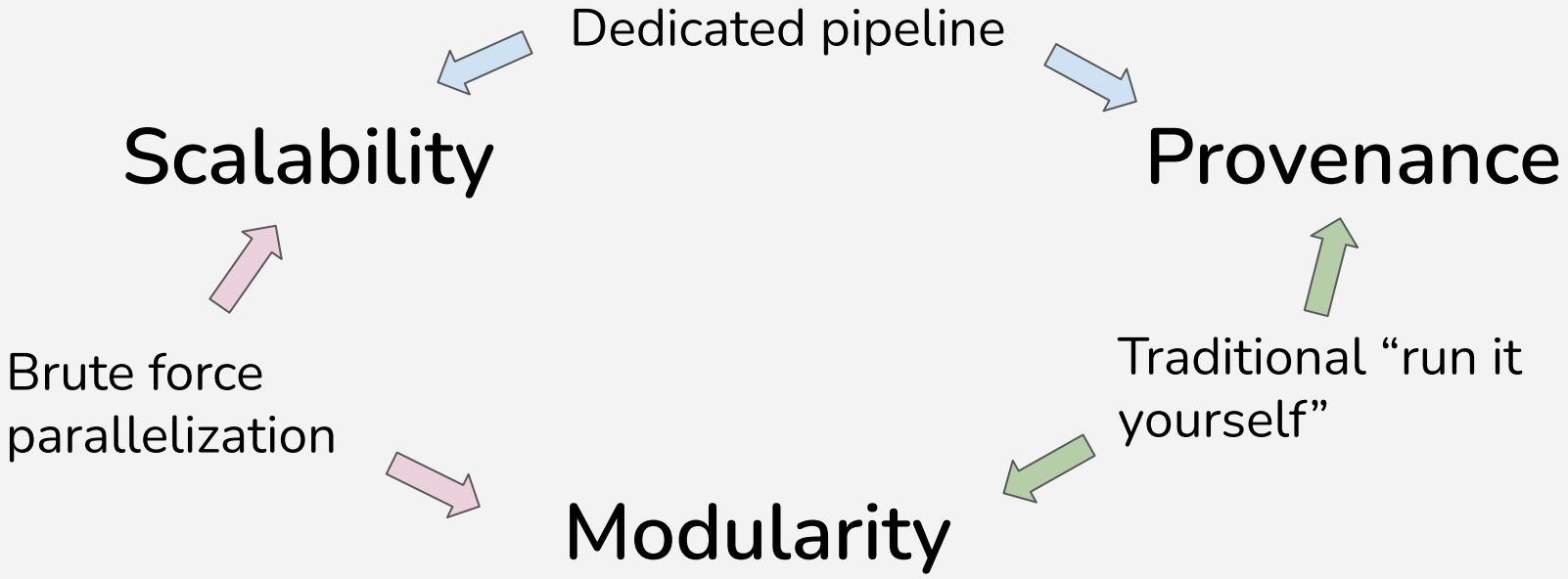
I would like...

Scalability

Provenance

Modularity

Any two are “straightforward”…



… all three harder.

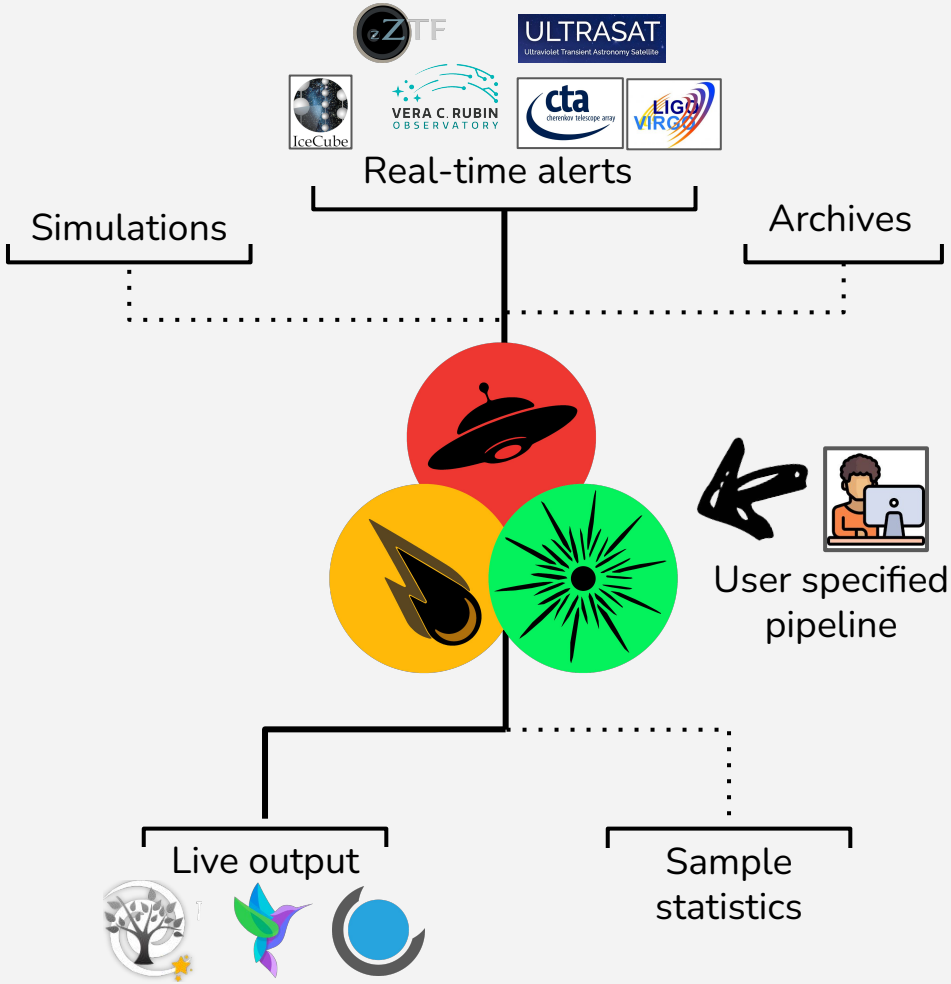
AMPEL

AMPEL is a *modular* and *scalable* platform with explicit *provenance* tracking, suited for systematically processing large (possibly complex and heterogeneous) data streams.

This includes selecting, analyzing, updating, combining, enriching and reacting to data.

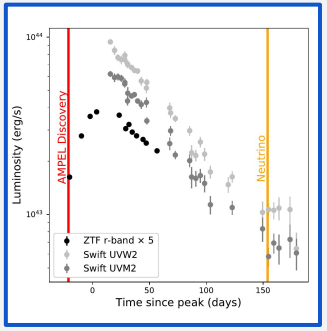
Code-To-Data: Teams develop analysis algorithms which are hosted in AMPEL and exposed to data streams in a high performance computer center.

AMPEL

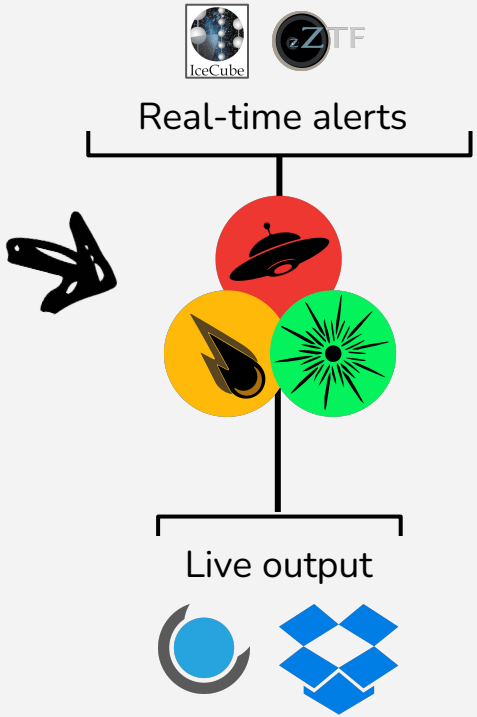

ZTF
ULTRASAT
Ultraviolet Transient Astronomy Satellite
IceCube
VERA C. RUBIN OBSERVATORY
cta
LIGO VIRGO
Real-time alerts
Simulations
Archives
User specified pipeline
Live output
Sample statistics

Sample use case:

Neutrino-TDE coincidence searches



TDE lightcurve model



Match TDE model to real-time stream.
Cross-match to IceCube alerts.

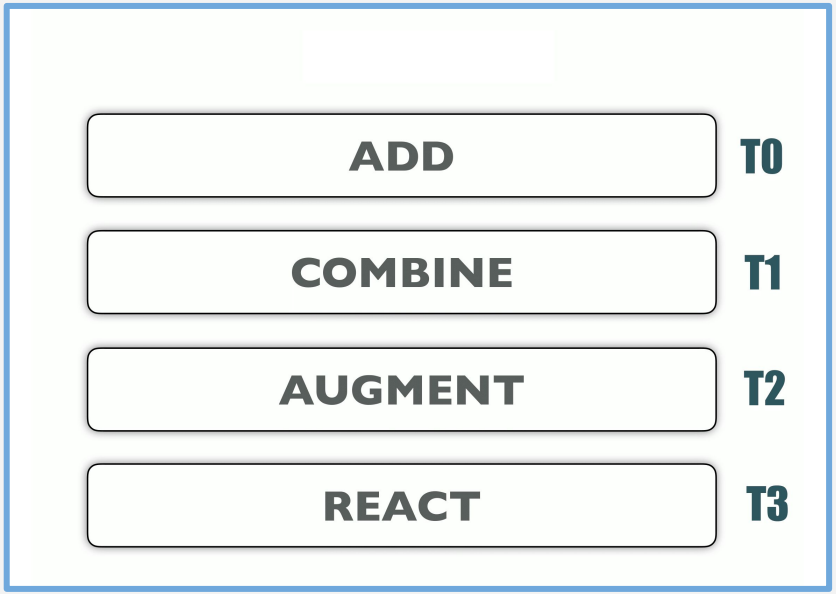
Selected conceptual components

1 Deconstructing scientific workflows.

Modularity requires rules for how modules interact.

Theory: workflows can be constructed from operations of four different kinds.

Four separate execution layers, each distinguished by input/output types.

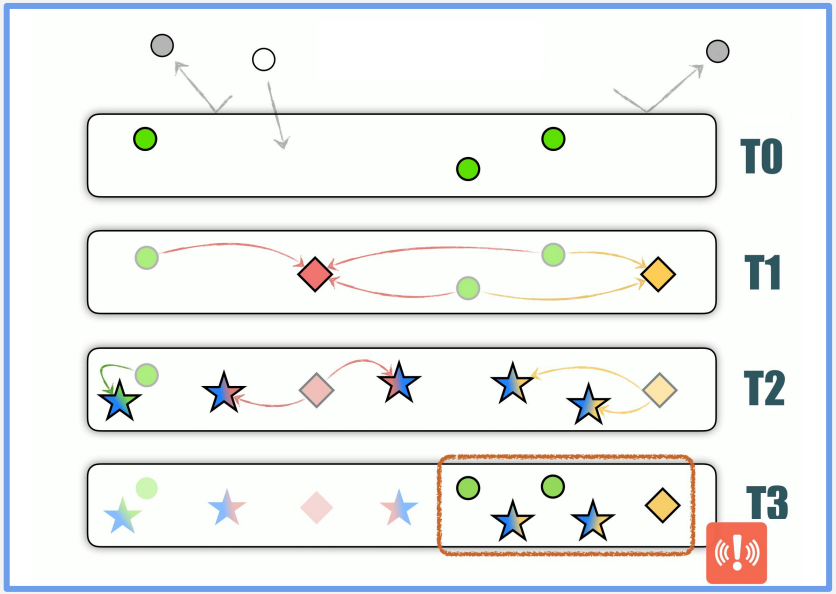


1 Deconstructing scientific workflows.

Modularity requires rules for how modules interact.

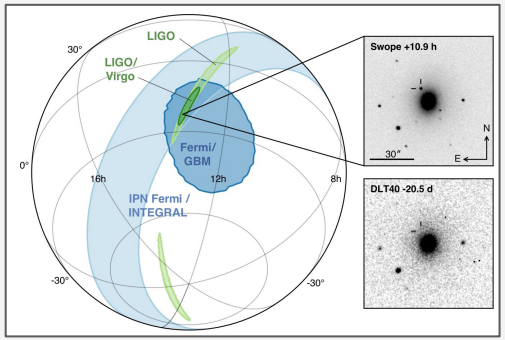
Theory: workflows can be constructed from operations of four different kinds.

Four separate execution layers, each distinguished by input/output types.

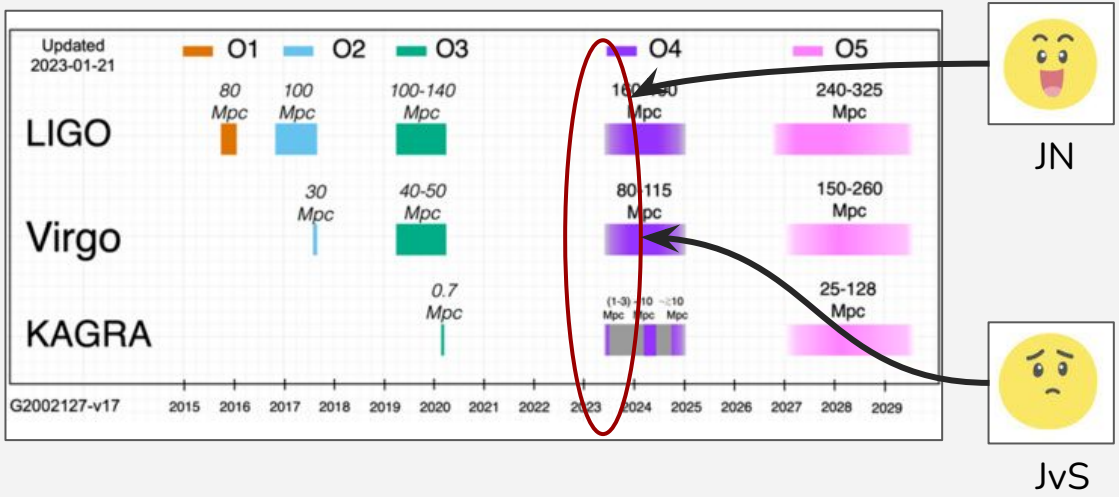


Sample use case:

Kilonova counterparts to gravitational wave events



The AMPEL alert archive, integrated astronomical catalogs and parallelizable T2 computations allow large data volume to be scanned.

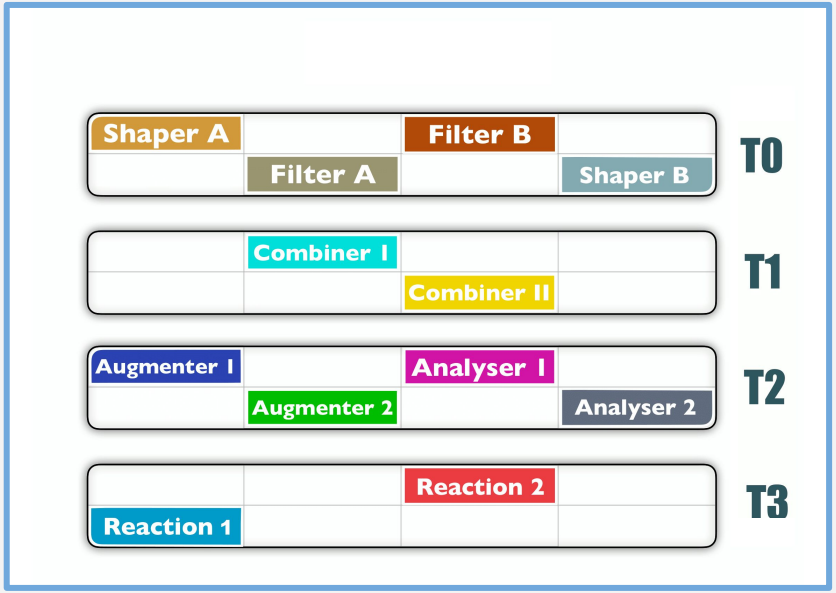


2 Merging user contributions

User:

Implements operations as units bound to a Tier.

Collects a series of units into a channel.

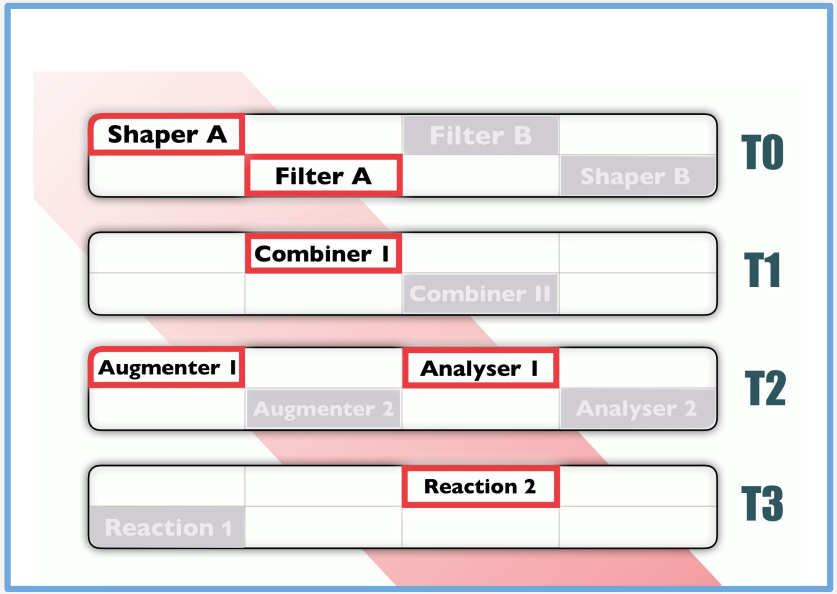


2 Merging user contributions

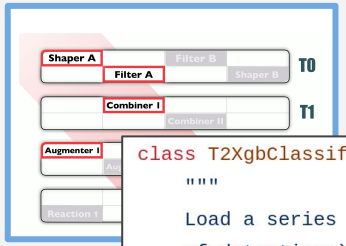
User:

Implements operations as units bound to a Tier.

Collects a series of units into a channel.



2 Merging user contributions




```python
class T2XgbClassifier(AbsTiedStateT2Unit):
    """
    Load a series of xgboost classifier models (distinguished by number
    of detections) and return a classification.

    Will test whether the first (null) model of the loaded classifier is
    more likely, P(model 0)>0.5. What this means is determined by the training.

    E.g. For the elasticc1v2 model, the null model corresponds to Elasticc class 1
    (and model 1 to class 2.)

    """
```

```python
class AbsTiedStateT2Unit(Generic[T], AbsTiedT2Unit, abstract=True):
        """
        A T2 unit bound to a :class:`~ampel.content.T1Document.T1Document` (state of a stock),
        as well as the results of other T2 units
        """

        t2_dependency: Sequence[StateT2Dependency[T]]

        @abstractmethod
        def process(self,
                compound: T1Document,
                datapoints: Sequence[DataPoint],
                t2_views: Sequence[T2DocView]
        ) -> UBson | UnitResult:
                """
                Returned object should contain computed science results to be saved into the DB.

                .. note:: the returned dict must have only string keys and be BSON-encodable
                """
```

Class methods determine I/O.

Unit: python module inheriting from Ampel-interface class.

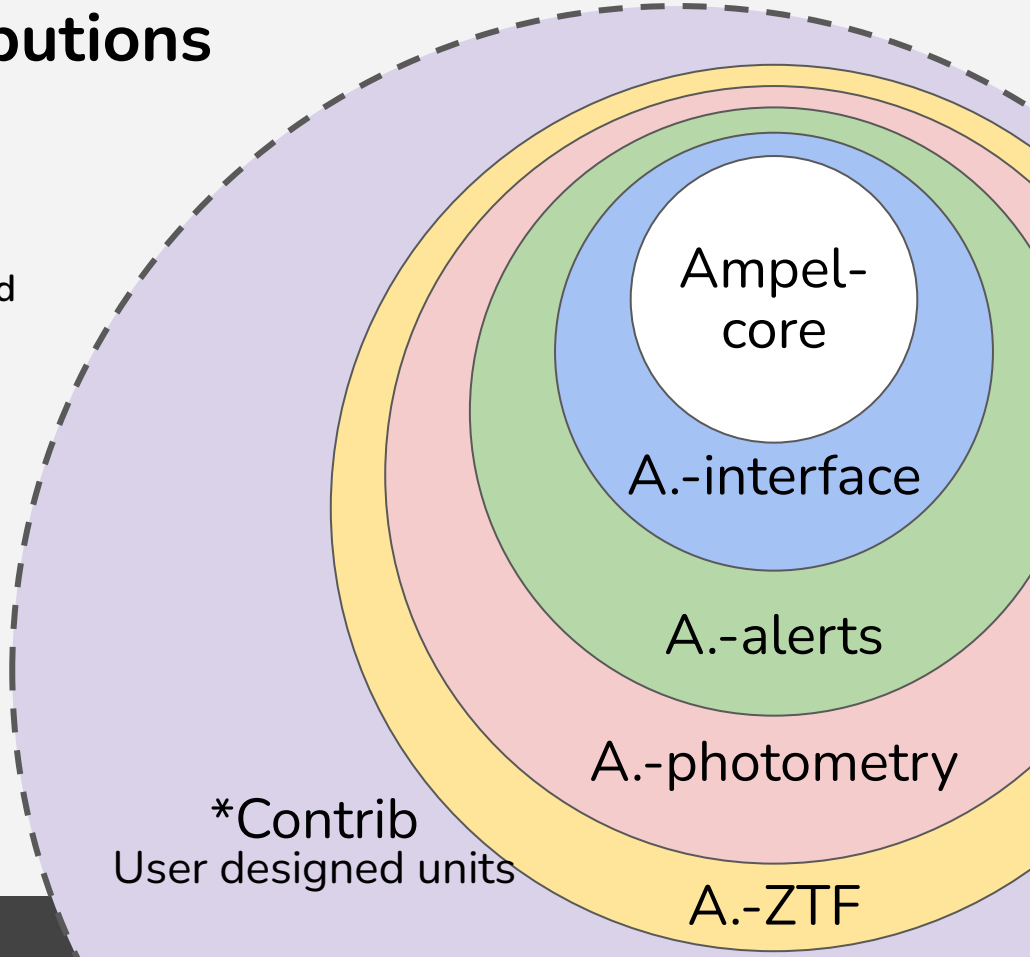
2 Merging user contributions

AMPEL Core:

- Workers at each tier executes units with requested input, allowing system control and parallelisation.
- Results stored in NoSQL (Mongo) DB.
- Built-in provenance tracking (event journal, logs and jobs)

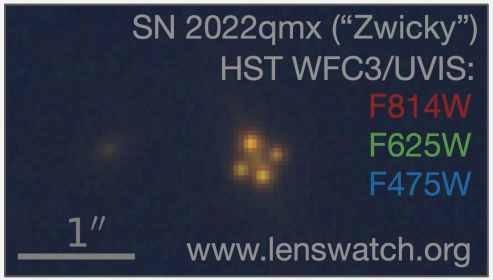
Execute a job:

- locally to develop,
- at a cluster for large-volume archive runs
- in a live instance to analyze real-time data

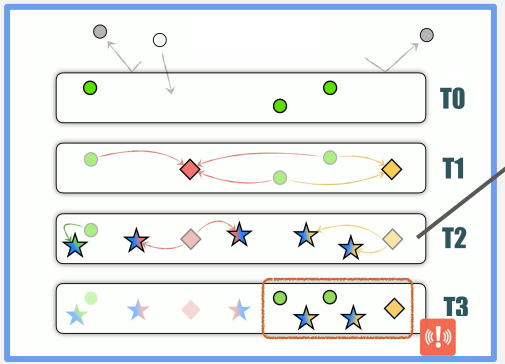


Sample use case:

Searching for gravitationally lensed supernovae



Exceedingly rare cosmological probes.




```
class T2GetLensSNParameters(T2RunSncosmo):
    unc: float

    def _get_fit_metrics(self, sncosmo_result, sncosm
        #------------------------------- Overriden m
        lc_metrics = super()._get_fit_metrics(sncosmc
        # Record the closest detection to peak in eac
        t0 = sncosmo_model.get('t0') # get time at pe
        sncosmo_table.add_column('temporary', name='e
        sncosmo_table.sort(['time']) # sort data chrc

        def take_closest(myList, myNumber):
```

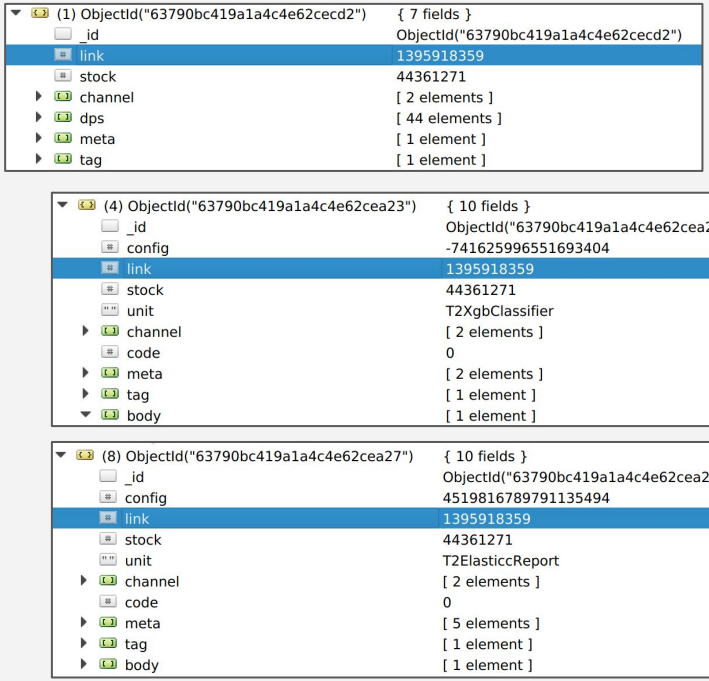
Extension of an existing unit allowed current pipelines to be reused for a lens searches.

3 States: key to provenance and deduplication

A *state* is the collection of data points associated with an (assumed) object at a specific time and visible to a specific user.

AMPEL systematically records every transient state as an immutable object.

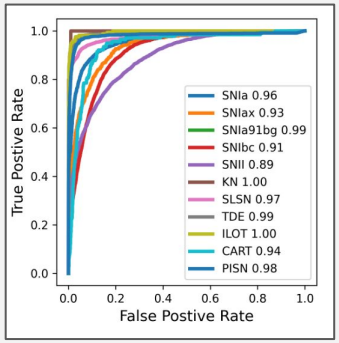
Every computation is directly tied to a state, a key component of the enforced data provenance model. Also enables deduplication - identical computations on the same state are elided.



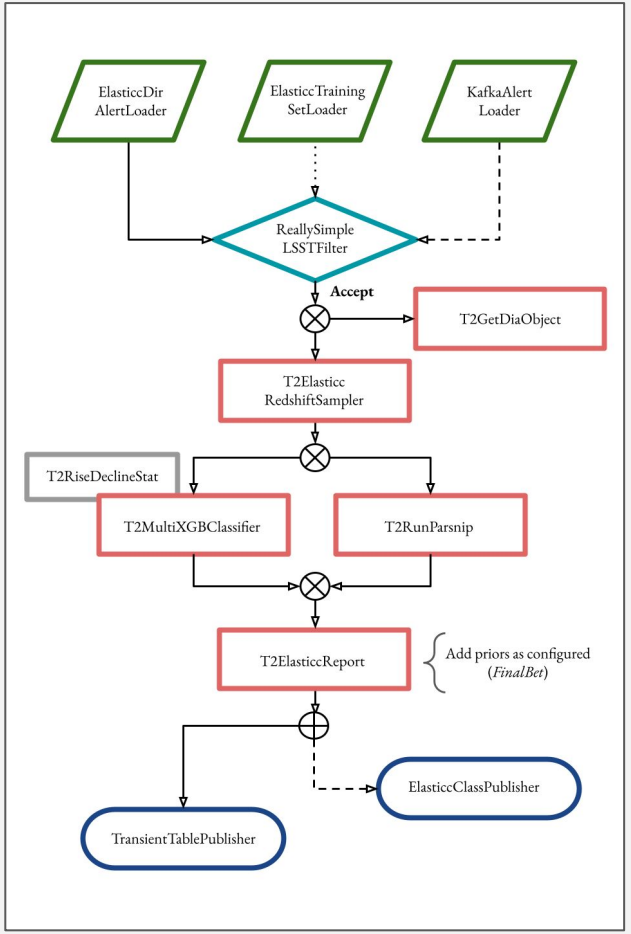
Sample use case:

Photometric classification for LSST

LSST has immense science potential - *if* transients can be photometrically classified.



Modularity allowed AMPEL to be quickly adopted to the ELASTiCC data challenge and emulate a realistic transient program.
Each classification tied to a state.



Useful tools

Contains product endorsements

Type annotations + mypy

- (Optionally!) declare types for variables, arguments, return values in Python >= 3.5
- Check for type safety (and stupid typos) with mypy, e.g. in refactoring large codebases with incomplete test coverage
- Bonus: more powerful tab completion in e.g. VSCode (Pylance) or vim (Jedi)

```python
def get_item(target: dict[str, int], key: str) -> None | int:
    return target.get(key)

def do_a_thing_with_a_list(target: list) -> None:
    return target.keys()

if __name__ == "__main__":
    get_item({"foo": 1}, "foo")
    get_item([("foo", 1)], "foo")
    get_item({"foo": 1.}, "foo")
```

Code with type annotations (and errors!)

```
> mypy mypy_demo.py
mypy_demo.py:5: error: "List[Any]" has no attribute "keys"
[attr-defined]
mypy_demo.py:9: error: Argument 1 to "get_item" has
incompatible type "List[Tuple[str, int]]"; expected "Dict[str,
int]"  [arg-type]
mypy_demo.py:10: error: Dict entry 0 has incompatible type
"str": "float"; expected "str": "int"  [dict-item]
Found 3 errors in 1 file (checked 1 source file)
```

Errors found statically by mypy

pydantic

- Sometimes you just want a struct that
  a. Generates __init__ and __repr__ for you => dataclasses
  b. Can also load itself from an untrusted source, and give useful error messages when it fails => pydantic
- Annotations define expected types
- Basic validation with constrained types, e.g. “list with at least one item”, “float < 0”
- Arbitrary validation with @validator
- Validation logic stays with the data model

A basic pydantic model

```
from typing import Sequence
from pydantic import BaseModel

class ChannelModel(BaseModel):
   channel: int | str
   version: None | int | float | str
   active: bool = True
   members: None | Sequence[str]

if __name__ == "__main__":
   ChannelModel()
```

A validation error

```
pydantic.error_wrappers.ValidationErr
or: 1 validation error for
ChannelModel
channel
  field required
(type=value_error.missing)
```

FastAPI

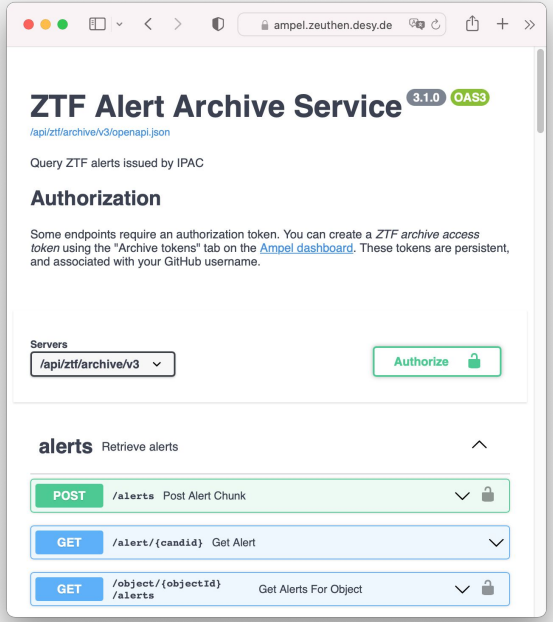
Does what it says: write REST APIs, fast

- Most of a REST API is input validation
- FastAPI delegates validation to pydantic, and returns an informative 422 Unprocessable Entity on failure
- Declare handlers with decorators

```python
@app.get("/process/{process}")
async def get_process(process: str) -> ProcessModel:
    for tier in range(4):
        try:
            doc = context.config.get(f"process.t{tier}.{process}", dict,
raise exc=True)
        except Exception:
            continue
        return ProcessModel(**doc)
    else:
        raise HTTPException(status_code=404, detail=f"{process} not found")
```

A GET handler with a single parameter

Autogenerated, interactive API docs

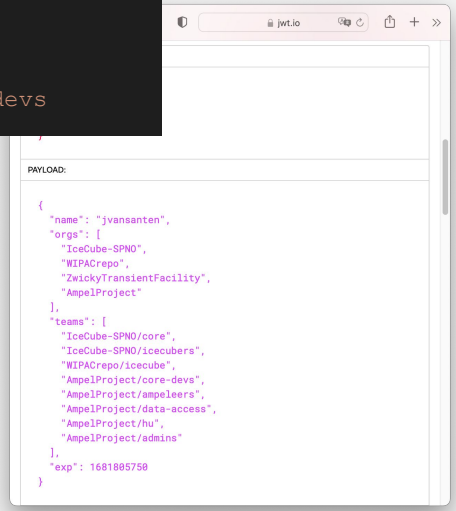


Authentication and Authorization

- Ampel is a multi-user system with a (read-only) API => restrict views to authorized users
- Most people have a GitHub account => use GitHub as identity provider
- Authorization backend issues a signed token containing username and org/group memberships
- Channel definitions include identities that should have access to data and logs

Example channel config (excerpt)

```
channel: TDE_RANKING
contact: simeon.reusch@desy.de
active: true
members:
 - sjoertvv
 - mitchellkarmen
 - simeonreusch
 - robertdstein
 - AmpelProject/core-devs
```

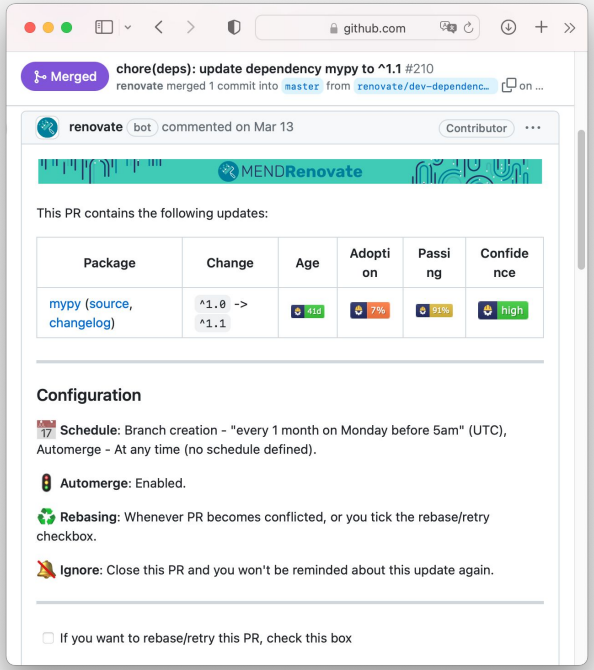


Decoded JWT payload in jwt.io debugger

Renovate

- Renovate is a dependency updater with support for nearly every manager you can think of (think Dependabot, but more configurable)
- Example from our config:
  a. Ampel sub-projects: update ASAP
  b. Dev dependencies: bundle and update once a month, automerge if CI passes
  c. Major updates: open a PR for each
- Run hosted or on-prem (e.g. for GitLab)

A Renovate PR



Miscellaneous morsels

- Stop arguing about style and just use black.
- Write tests without the boilerplate using pytest.
- Dependency lock files are a great way to keep a known-working set of dependencies for testing. Use a manager like poetry, pdm, or conda-lock to create and maintain them.
- Solving and installing large environments with conda is surprisingly slow. micromamba is much faster, and comes as a statically-linked binary.

Lessons learned

NoSQL databases are great!

- ...until you have your first many-to-many relationship
- SQL: store single copy, reference by id
- NoSQL: either
  a. Keep multiple copies, forget about consistency
  b. Build your own relational logic at the application level, using only idempotent and commutative operations
- Still a decent choice when you never update or delete records, or just have piles of unrelated documents.
- If you just want to avoid defining a schema, consider jsonb columns in PostgreSQL.

Tightly-coupled Git repositories

- Sometimes you have lots of tightly-coupled code that you want to update atomically, but want different permissions for each component (some astronomers care about their code being secret)
- Git makes this so much harder than SVN. Can enforce write permissions with PRs and CI, but there is no way to make fine-grained read permissions
- Least bad workaround so far:
  a. split into multiple repos
  b. publish packages from CI for basically any change
  c. update dependency spec on downstream packages
- But: can't detect when a change breaks a downstream package

Enabling scientific software development is not enough

- AMPEL was designed to encourage and support continuous scientific software development.
- … this does not mean it happens.

Multiple cases of well working units constructed by knowledgeable scientists. Appreciated by science community, but not supported once v1 is running.

Labels matter

- It is hard to introduce new workflow concepts.
- Broker is a new thing - everything is a broker - what is a broker?
- Code-To-Data actually seems to work in communication:

> AMPEL allows scientific time-domain software to be locally developed, shared with other users and uploaded to high performance computer centers - Code-To-Data.