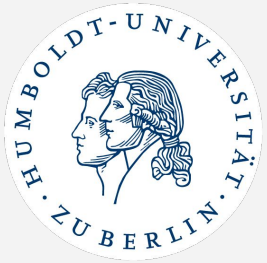
AMPEL



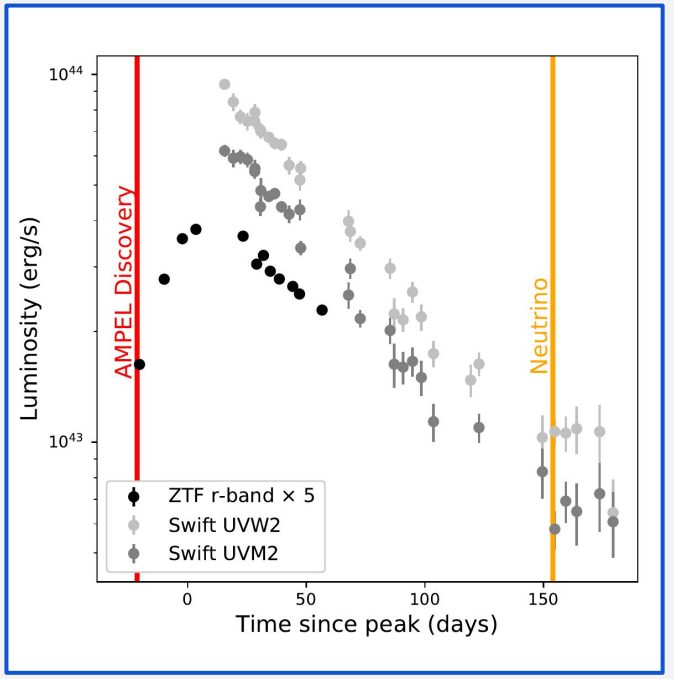
J. Nordin, V. Brinnel, J. van Santen, M. Kowalski, A. Franckowiak, M. Rigault, R. Stein, S. Reusch, M. Giomi

AMPEL is a software framework designed to enable complex real-time analysis of heterogeneous data streams.

Sample active programs:

1. Multi-messenger science

Real-time comparisons between optical and gravitational wave, neutrino and GRB alert streams.



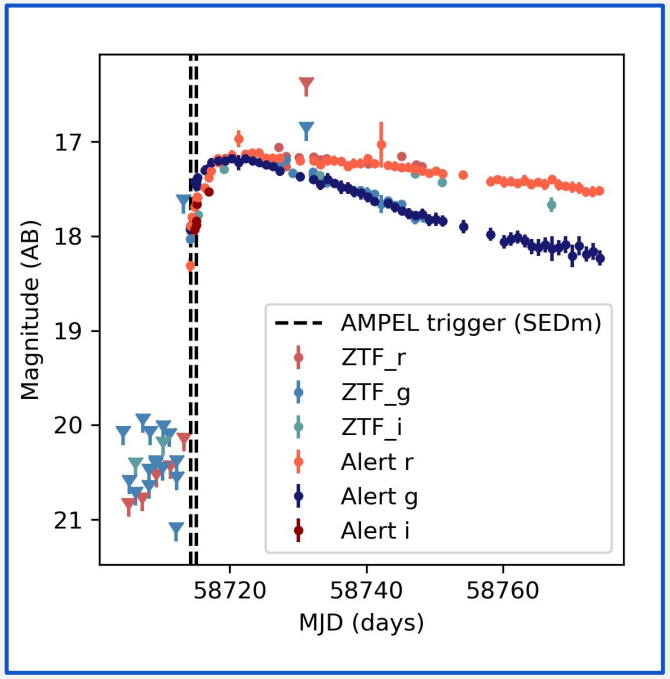
Right: The TDE 2019dsg was first detected by AMPEL, and then associated with a cosmic neutrino observed by IceCube, again via AMPEL. (Stein+ 20)

Sample active programs:

2. Autonomous transient selection

Immediate follow-up observation w. robotic telescopes of infant supernovae. Likely extragalactic transients posted to TNS in real-time (Nordin+ 19).

Right: Autonomous follow-up observation ~5h after SN explosion.

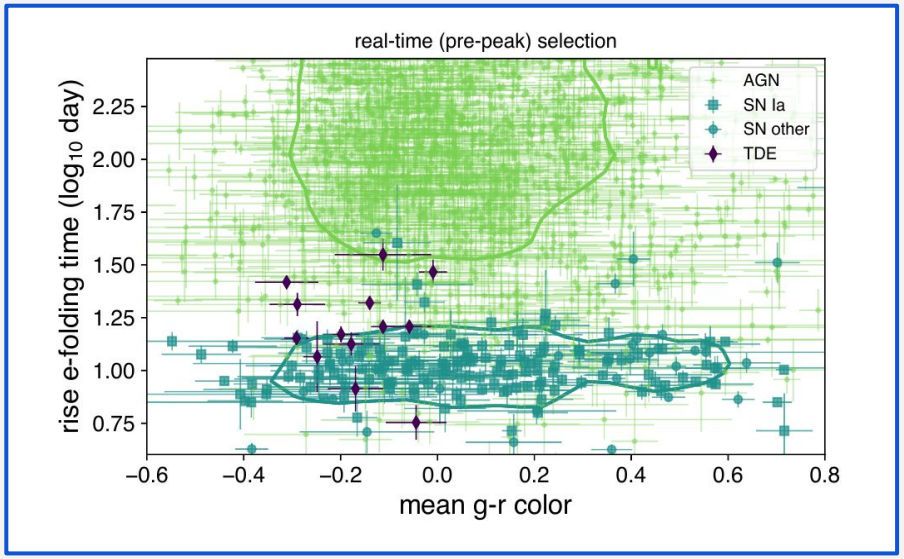


Sample active programs:

3. Complex lightcurve evaluation

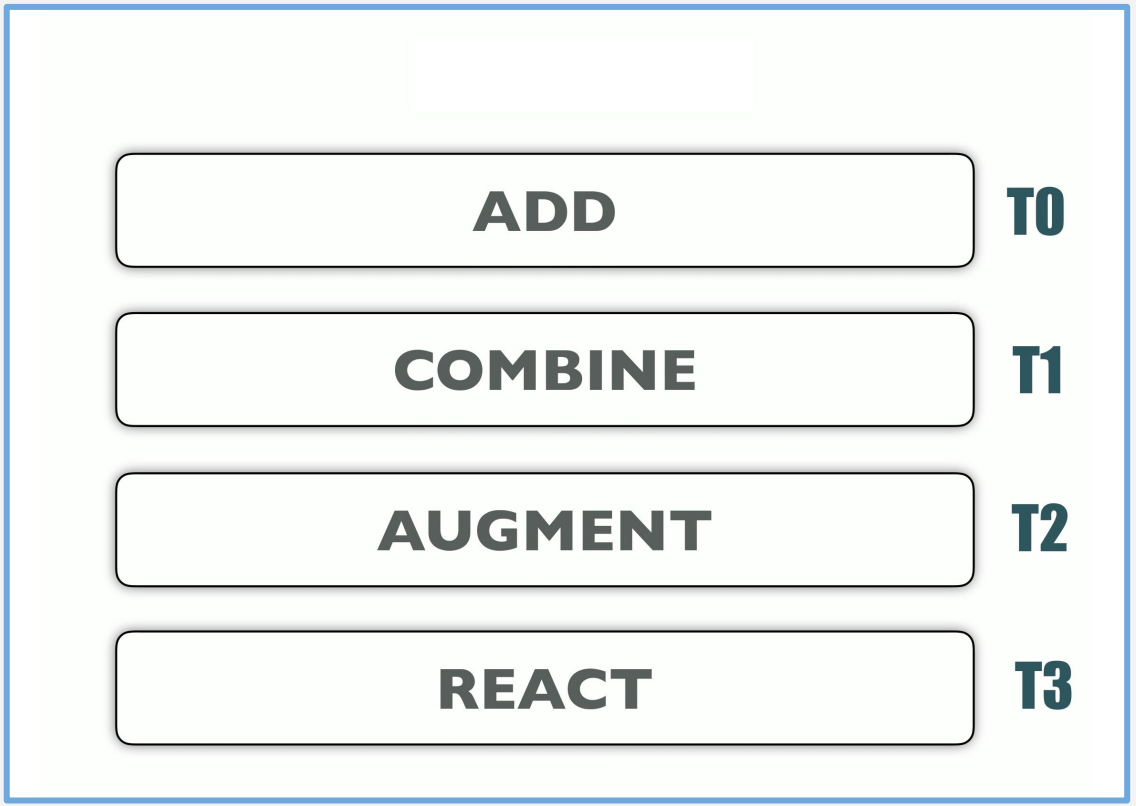
Domain specific algorithms allow e.g. the detection of TDEs (Velzen+ 20) and test for completeness in the ZTF Bright Transient Survey (Fremling+ 20).

Right: Sample real-time metrics for finding TDEs (Velzen+ 20).



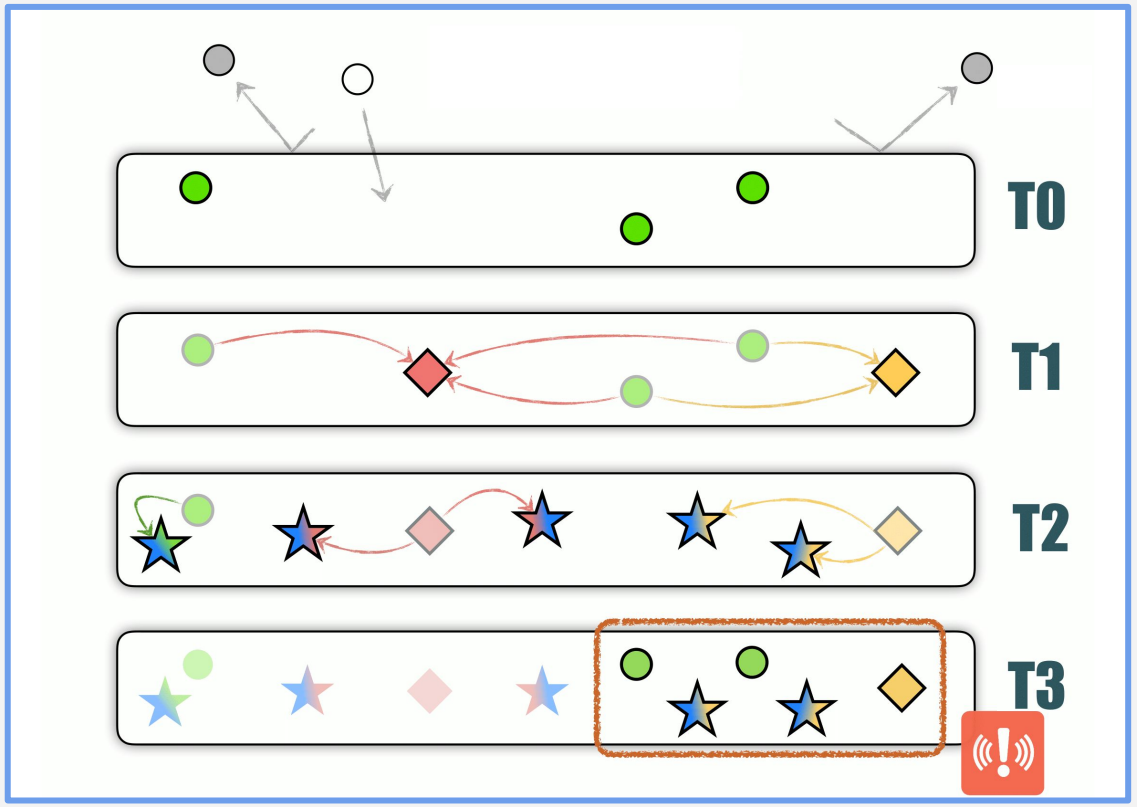
AMPEL internal structure

We divide transient processing into four *execution layers*:



AMPEL internal structure

We divide transient processing into four *execution layers*:



AMPEL internal structure

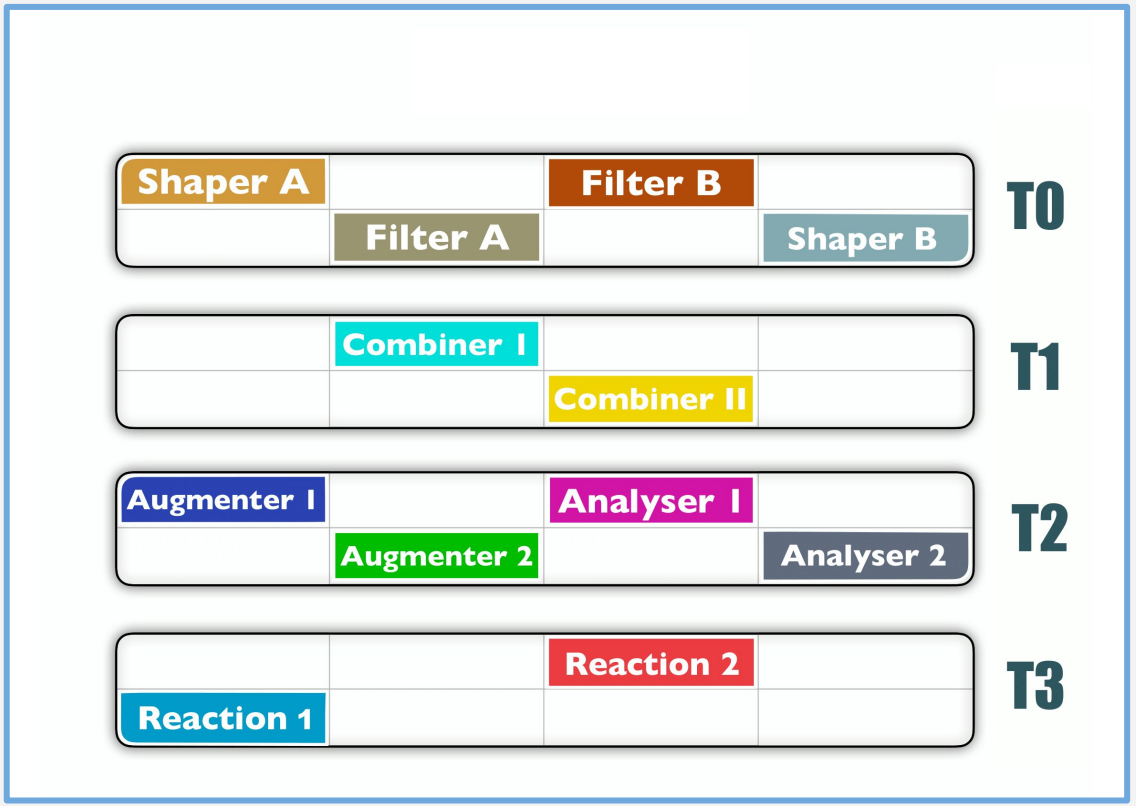
We divide transient processing into four *execution layers*:

1. *Add* data to system.
2. *Combine* data into transient states.
3. New properties of states are *calculated*.
4. *React* based calculated values.

A transient science program, a *channel*, is created by selecting *units* at each layer. A unit carries out a computation within a layer, while the AMPEL system communicates between layers.

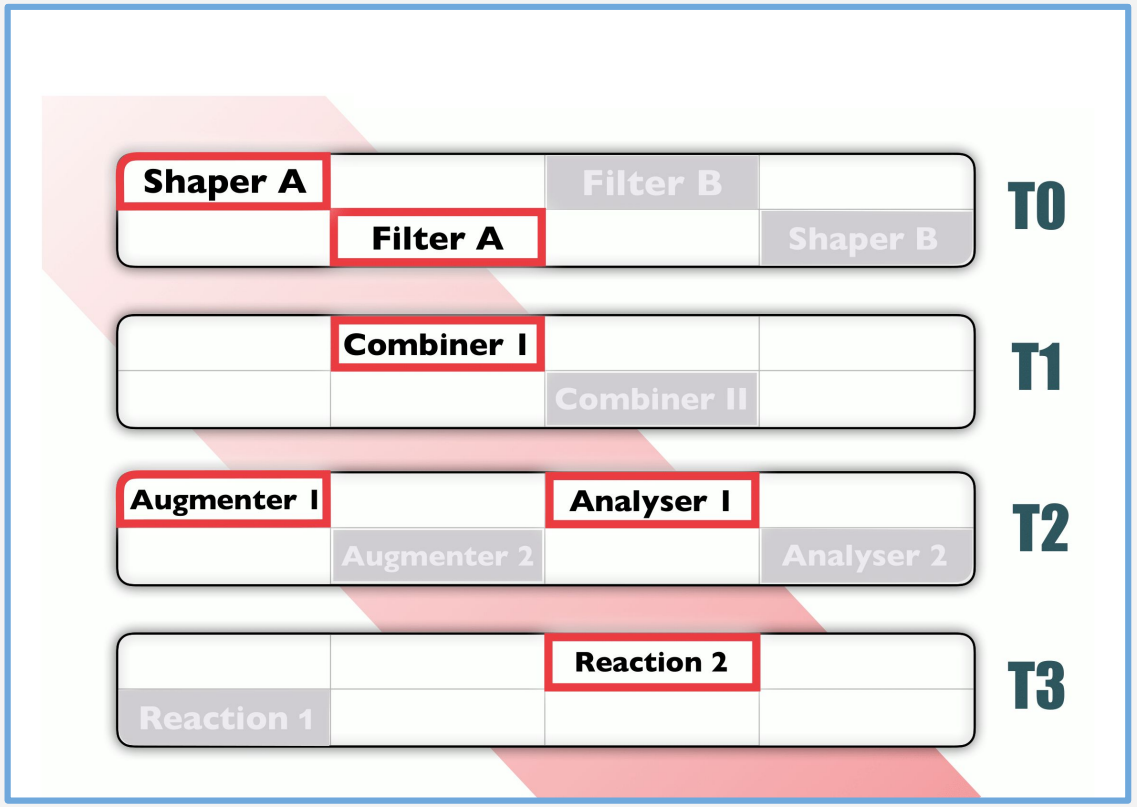
AMPEL internal structure

A transient science program, a *channel*, is created by selecting *units* at each layer.



AMPEL internal structure

A transient science program, a *channel*, is created by selecting *units* at each layer.



Is AMPEL a broker?

We call AMPEL a public platform for processing data-streams. Users supply analysis schema which are executed at the live AMPEL instance hosted at DESY Zeuthen.

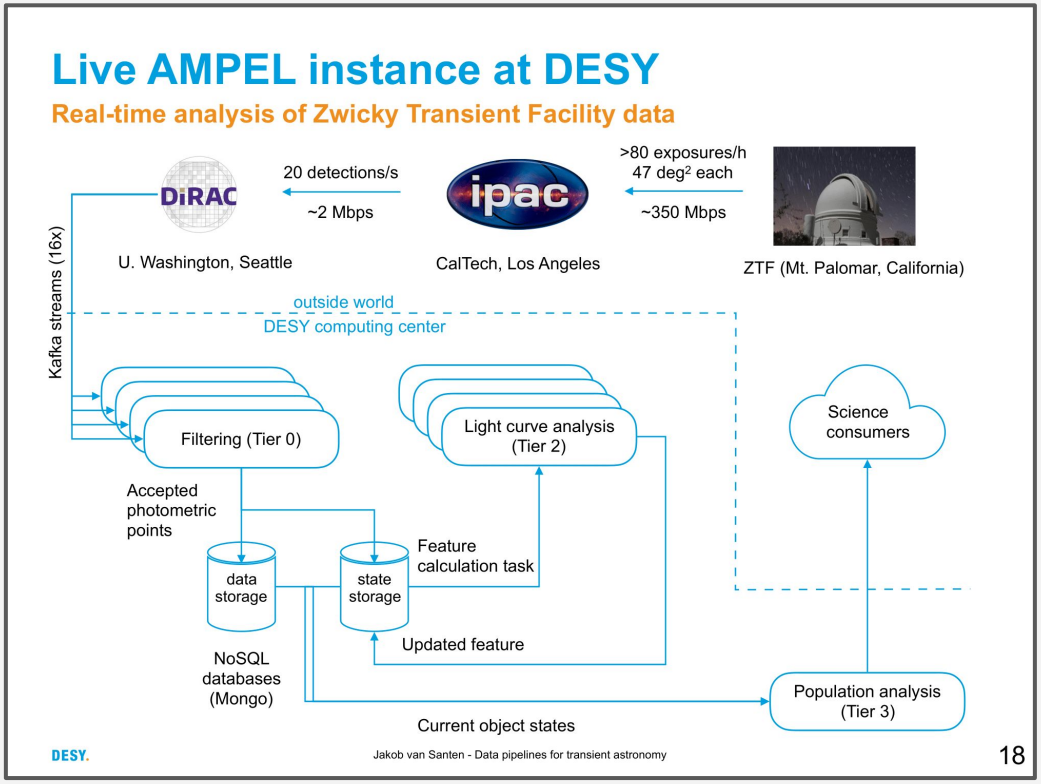
AMPEL can be configured to work as an alert broker for astronomical alerts. The modular execution layers also allow for more complex, user-defined analysis of heterogeneous real-time data streams.

Using AMPEL

AMPEL is a public tool - anyone can submit a channel for inclusion in the live ZTF data processing. Public and private ZTF alerts are parsed.

Development starts on your local computer, usually by adapting a base AMPEL class to carry out the desired computation. There is no single visual access point as there is no global AMPEL user or science program.

The AMPEL live instance



The AMPEL live instance is hosted at the DESY Zeuthen Computer Center. ZTF public and private alerts have been processed since since summer 2018.

Getting started

More information:

- Intro paper (Nordin et al; 2019A&A...631A.147N)
- Tools for creating channels:
  - Github: AmpelProject/Ampel-contrib-sample
  - Github: AmpelProject/Ampel-interact
- We assist in creating channels:
  - ampel-info@desy.de

Summary

User provided analysis schema

Complex calculations,
heterogeneous (multi-messenger) streams.

Real-time reactions

Trigger requests and linked events,
feedback processing.

Time machine

Repeatability and provenance,
search optimization,
“what if” questions.

Additional material:

AMPEL design origin

Sample real-time analysis questions:

How do I find a specific kind of transient?
Would I have detected this object with this schema?
Were any objects looking like this previously observed?
What test statistic should I use when combining optical with non-optical data?
How many similar, sub-threshold objects exist in a set of archives?
How is the use of a follow-up facility optimized for different science programs?
How would a sample change based on updated lightcurve calibration?
Was this transient observed by other facilities, and how can I add that data?
I developed this sophisticated model for one source - how does it apply to others?
How do I react in real-time using this other facility in case this happens?
Why did we not study this particular transient closer?
Do our combined observations agree with this particular rate model?
Can I select targets for this particular follow-up telescope?

Sample real-time analysis questions:

How do I find a specific kind of transient?
Wo
We
Wh
dat
Hov
Hov
pro
Hov
Wa
I de
oth
Hov
Wh
Do
Car

Can be combined into four requirements:

1. Data irreversibility
2. Stream heterogeneity
3. Analysis software
4. Connection simulation -> archive -> real-time

Four real-time challenges:

1. Data irreversibility

We cannot go back and examine an event once it is gone.

Choices are irreversible and still has to be connected to our model of the Universe.

Four real-time challenges:

2. Stream heterogeneity

Different astronomical messengers have fundamentally different properties (photons, neutrinos, gravitational waves).

Even the same messenger kind have different properties when observed at different facilities (sensitivity functions, image quality metrics).

Four real-time challenges:

3. Analysis software

Complex data evaluations require flexible analysis software that can make use of state-of-the-art algorithms and be continuously developed by generations of scientists.

Four real-time challenges:

4. Connecting simulation to archive to real-time process

A complete study would apply the same analysis schema to simulate, archived and real-time data.

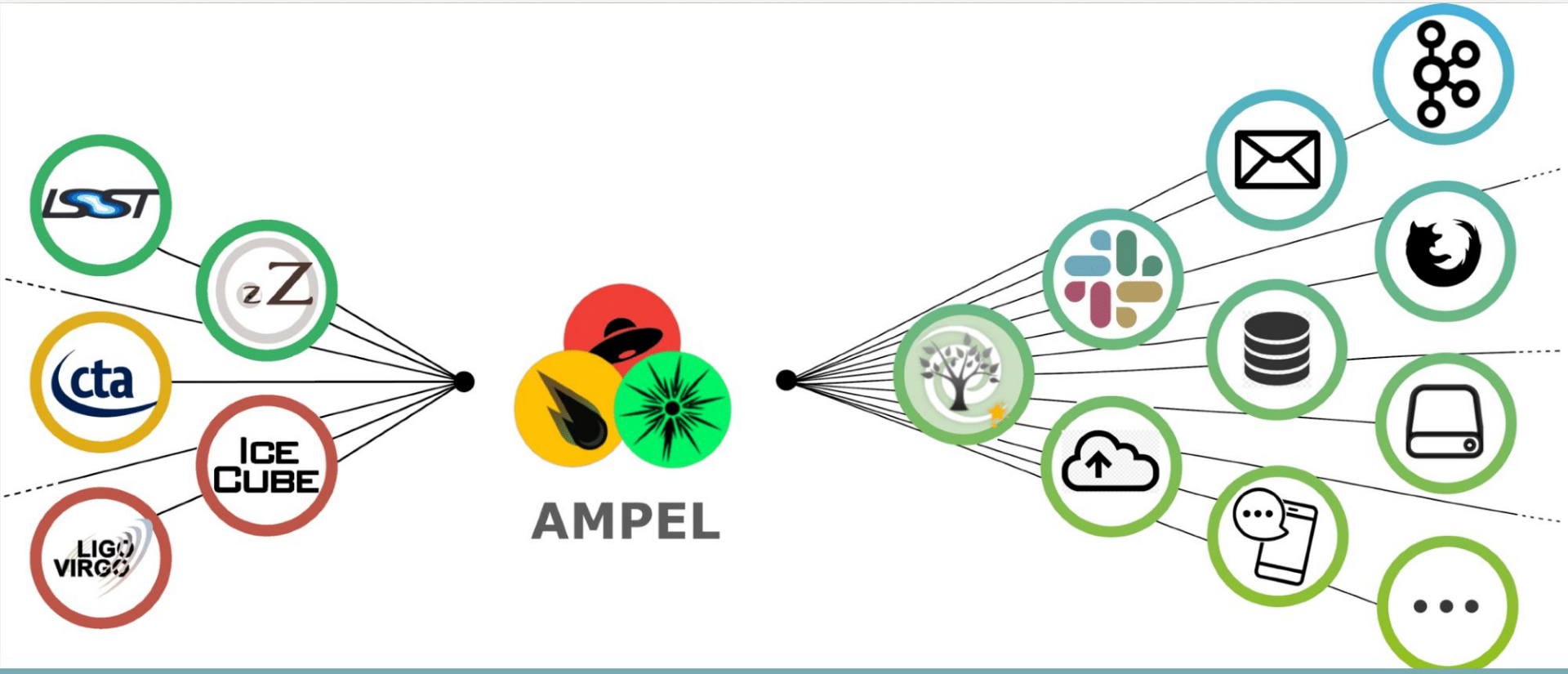
Allows optimization of real-time schema as well as tests as to whether a set of observations match a model of the Universe.

Four real-time challenges:

The AMPEL system, consisting of the modular units processing data in execution layers together with the backend database, is designed to meet these four challenges.

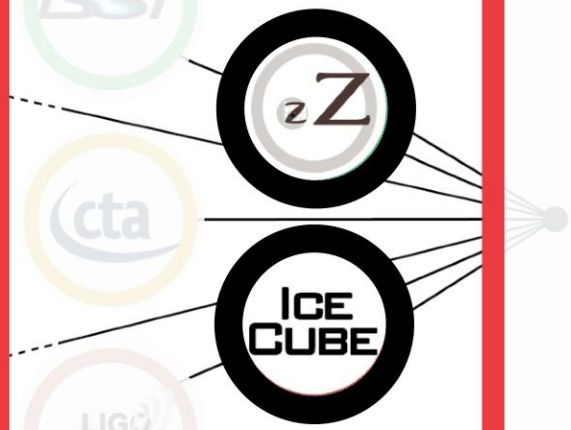
Execution layers and units

Execution layer summary



T0 T1/T2 T3

Execution layer summary



T0 config *regulates*:

- **Which instruments?**
- **Which filters?**

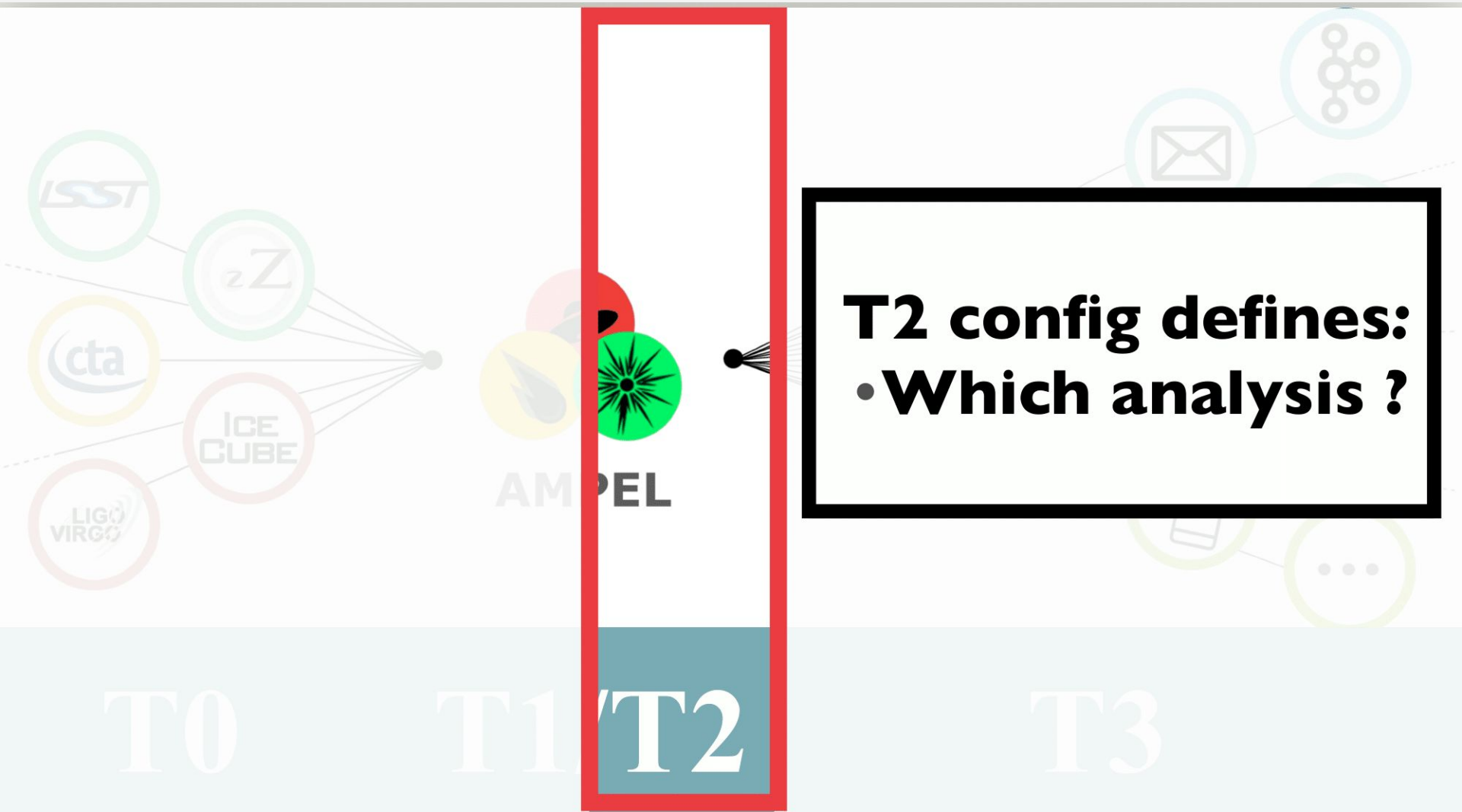
T0 T1/T2 T3

Execution layer summary

T1 config defines:

- **How to combine data?**

Execution layer summary

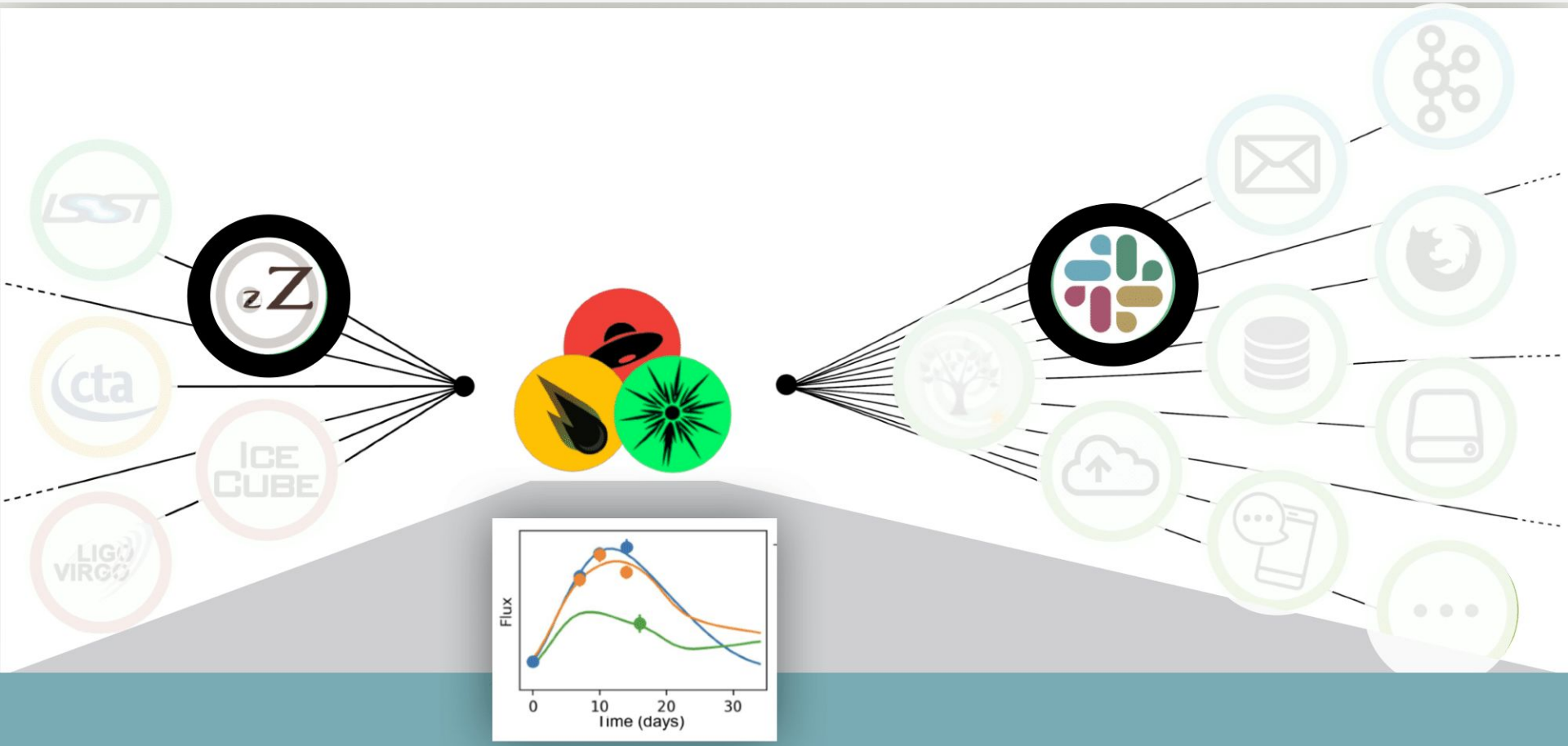


Execution layer summary

***T3* defines:**

- **Which output?**
- **Which action?**

Sample *channel*



Light curve fit + host info + redshift …

How does a unit look?

```
13
14  class T2ExamplePolyFit(AbsT2Unit):
15          """
16          Polynomial fitting
17          Fits data using numpy 'polyfit'
18          """
19
20          version = 1.0
21          author = "ztf-software@desy.de"
22
23          def __init__(self, logger, base_config=dict()):
24                  """
25                  'logger': instance of logging.Logger (std python module 'logging')
26                          -> example usage: logger.info("this is a log message")
27
28                  'base_config': optional dict with keys given by the `resources` property of the class
29                  """
30
31                  # Save the logger as instance variable
32                  self.logger = logger
33
34          def run(self, light_curve, run_config):
35                  """
36                  'light_curve': instance of ampel.base.LightCurve. See LightCurve docstring for more info.
37
```

From https://github.com/AmpelProject/Ampel-contrib-sample/blob/master/ampel/contrib/groupname/t2/T2ExamplePolyFit.py

How does a unit look?

```
33
34          def run(self, light_curve, run_config):
35                  """
36                  'light_curve': instance of ampel.base.LightCurve. See LightCurve docstring for more info.
37
38                  'run_config': dict instance containing run parameters defined in ampel config section:
39                          t2_run_config->POLYFIT_[run_config_id]->runConfig
40                          whereby the run_config_id value is defined in the associated t2 document.
41                          In the case of POLYFIT, run_config_id would be either 'default' or 'advanced'.
42                          A given channel (say HU_SN_IA) could use the runConfig 'default' whereas
43                          another channel (say OKC_SNIIP) could use the runConfig 'advanced'
44
45                  This method must return either:
46                          * A dict instance containing the values to be saved into the DB
47                                  -> IMPORTANT: the dict *must* be BSON serializable, that is:
48                                          import bson
49                                          bson.BSON.encode(<dict instance to be returned>)
50                                  must not throw a InvalidDocument Exception
51                          * One of these T2RunStates flag member:
52                                  MISSING_INFO:  reserved for a future ampel extension where
53                                                                  T2s results could depend on each other
54                                  BAD_CONFIG:         Typically when run_config is not set properly
55                                  ERROR:              Generic error
56                                  EXCEPTION:      An exception occured
57                  """
58
59                  x = light_curve.get_values("obs_date")
60                  y = light_curve.get_values("mag")
61                  p = numpy.polyfit(x, y, run_config['degree'])
62                  chi_squared = numpy.sum((numpy.polyval(p, x) - y) ** 2)
63
64                  self.logger.info("Please use 'self.logger' for logging")
65                  self.logger.debug("By doing so, log entries will be automatically recorded into the database")
66
67                  return {
68                          "polyfit": list(p),
69                          "chi2": numpy.sum((numpy.polyval(p, x) - y) ** 2)
70                  }
```

From https://github.com/AmpelProject/Ampel-contrib-sample/blob/master/ampel/contrib/groupname/t2/T2ExamplePolyFit.py

Sample units T0

At layer T0 *Filtering*:

- Select from a stream based on alert properties
- Select from stream based on astronomical catalog matching
- Select from a stream based on a specific region of interest

Sample units T1

At layer T1 *Aggregation*:

- Combine stream content with external data from other telescopes / messengers
- Obtain recalibrated / improved / flagged datapoints

Sample units T2

At layer T2 *Calculation*:

- Derive lightcurve properties
- Determine explosion date
- Calculate test statistics for data combinations

Sample units T3

At layer T3 *Reactions*:

- Send immediate alerts
- Autonomous scheduling of follow-up observation
- Distribute transient sample statistics

Including units

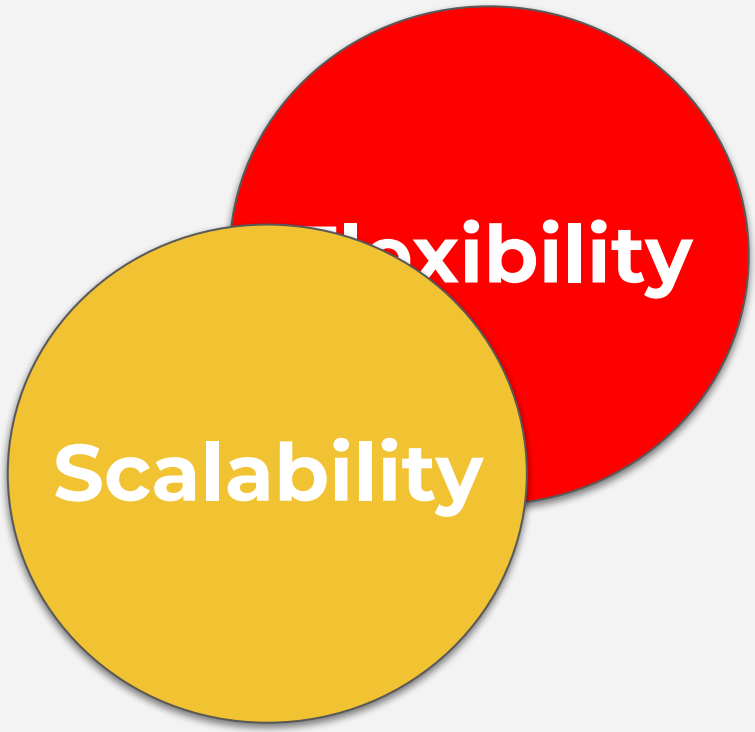
- Already existing units are added by name in the channel configuration
- Alternatively, specialized units are contributed as python modules through github linkage.
  - Use of existing astronomical libraries
  - Continuous, collaborative development

AMPEL benefits


Flexibility

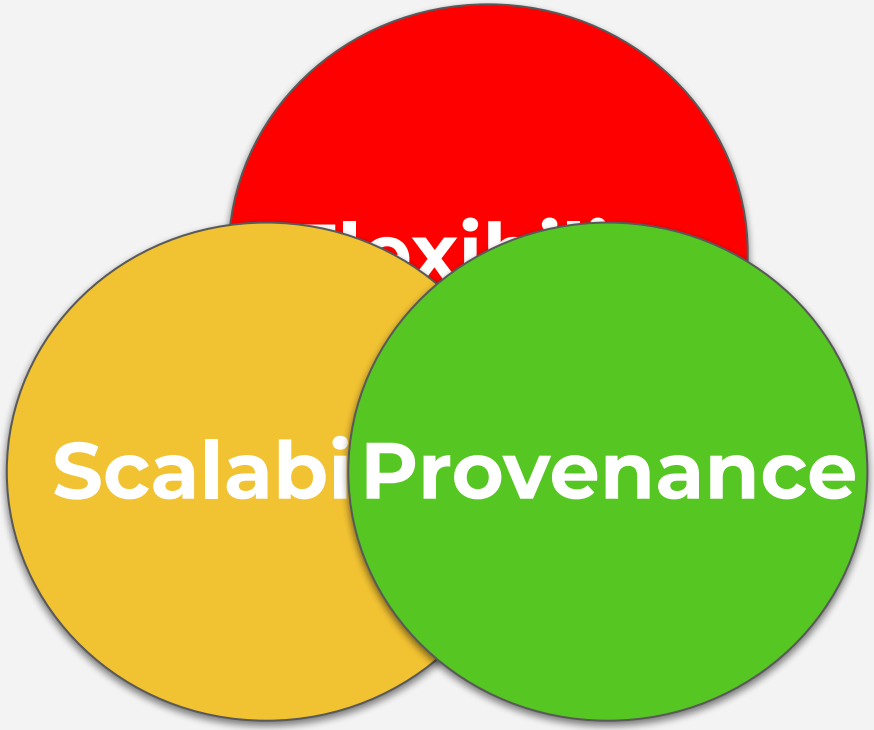
Flexibility

- States enable to combine information from different instruments & versions
- Easy to create new channels
- Modular analysis units
  - Re-use / citation of community work


xibility
Scalability

Scalability

- The execution layer layout enables:
  - Extensive and easy multi-processing
  - Near native distributed computing
- MongoDB scales well horizontally
- Identical computations requested by different channels shared internally


Scalabi
Provenance

Provenance

- States make computation with dynamic streams traceable and efficient
- Containerisation ensures repeatability
- The transient journal logs everything that happens
- Structured logs enable further efficient analysis
- Compatible with IVOA Provenance Data Model