ZADS

The ZTF Alert Distribution System

Eric Bellm, Maria Patterson, Frank Masci, Steve Groom, et al.

August 8, 2017

Why are we building ZADS?

ZTF promised (as an MSIP deliverable) an “LSST-like” near-real-time alert stream of the public survey to build up community infrastructure

⇒ ZTF needs a production alert stream

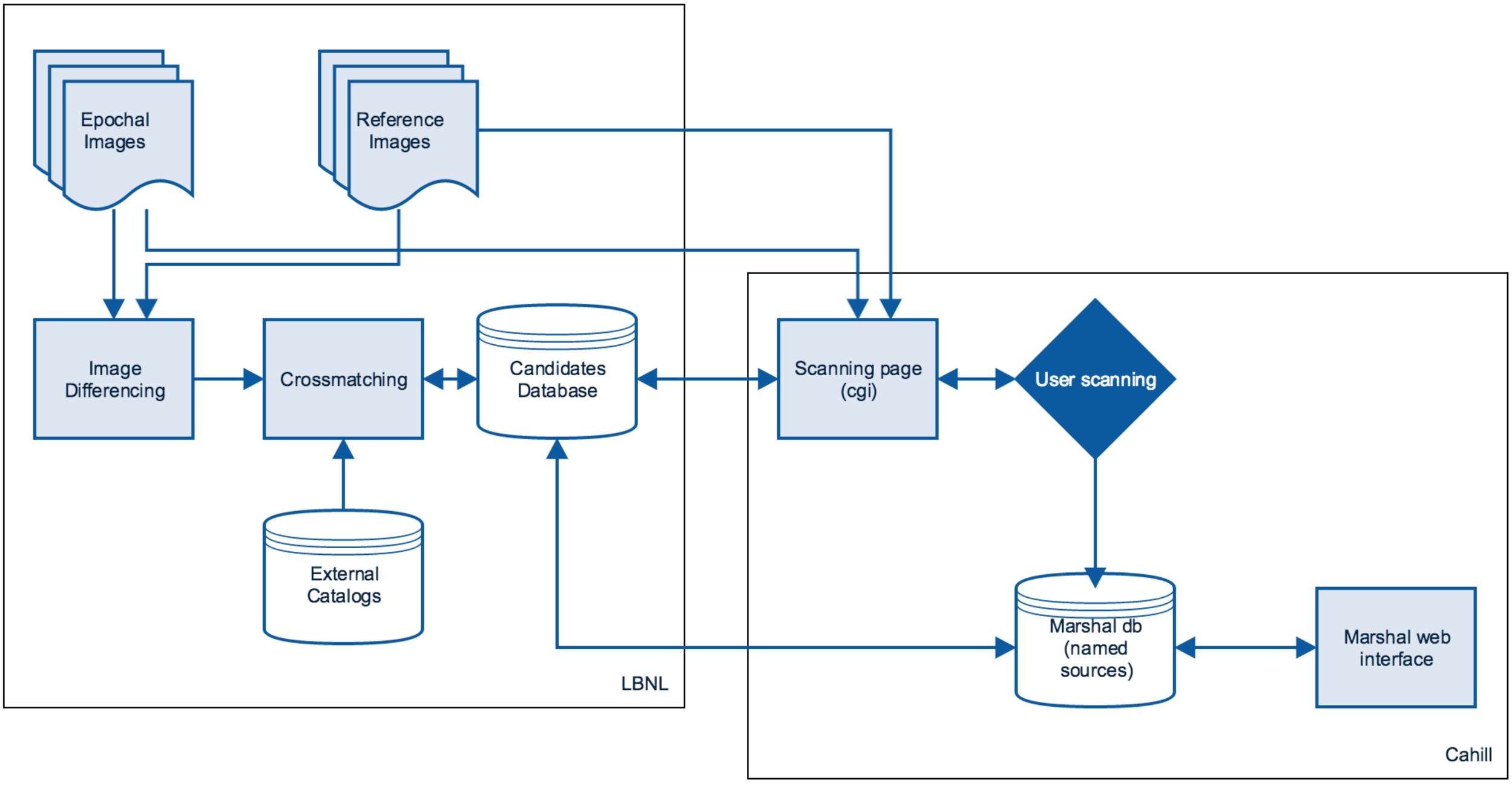
UW LSST group is responsible for producing the actual LSST transient stream (“Level 1”, “Alert Production”).

- Image processing
- Image differencing
- Alert packaging and distribution

⇒ UW interested in prototyping distribution system on real data

ZTF collaboration can benefit from early adoption of technology expected to be adopted by LSST

PTF used a database-centric model to access candidates.



find events by *querying* central database (batch)

IPAC has moved away from database-centered access for ZTF.

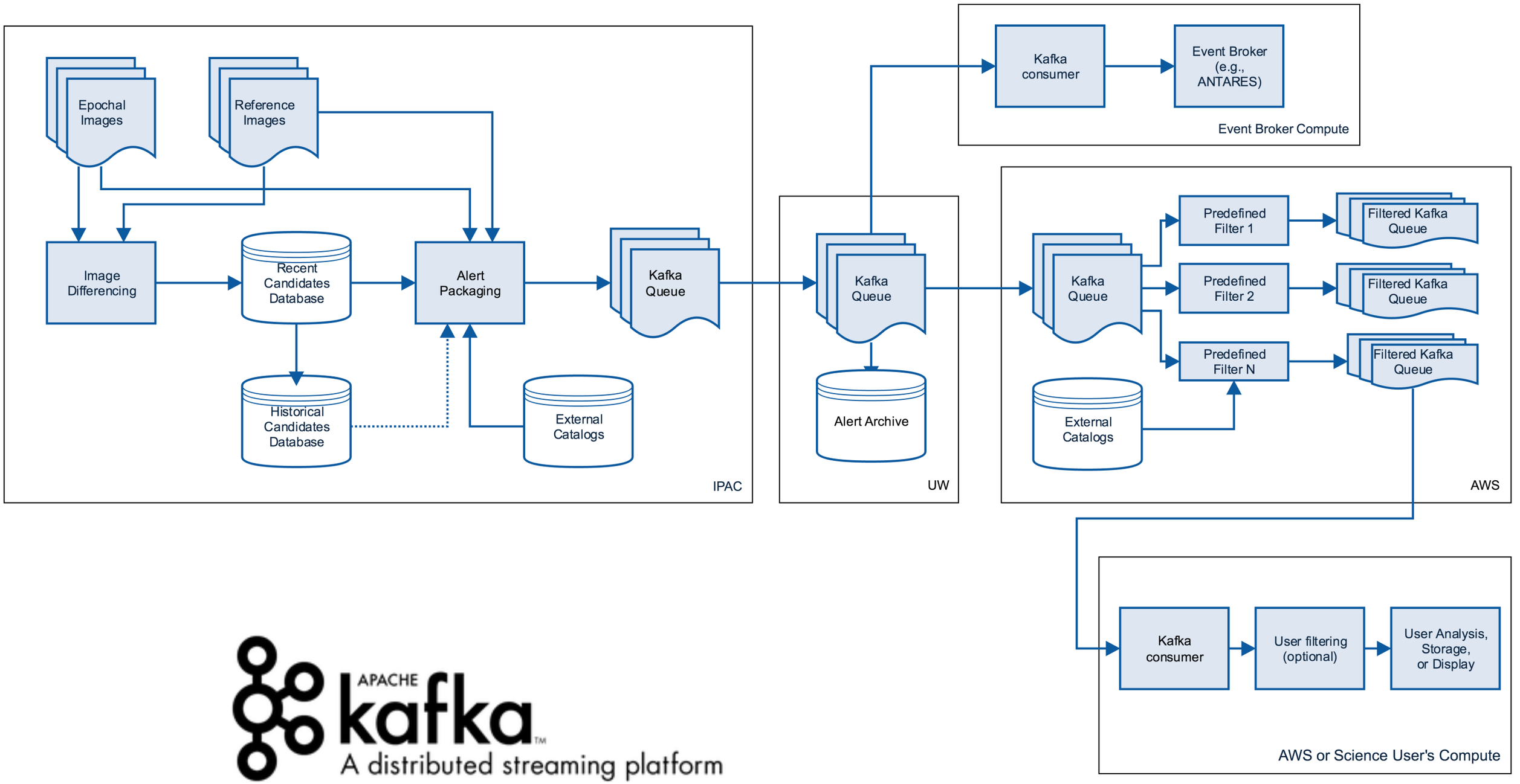
Large and *unpredictable* user load on operations database
    Wide range of query types and frequencies

15x larger data volume than PTF
  (modulo possible improvements in image subtraction)
Larger potential user base (MSIP)

⇒ Move filtering load from centralized database to users

+ Technically less risky, lower cost
+ Aligns with LSST “alert” model
+ Great flexibility for science users
- Larger outbound bandwidth for file transfer
- Users need to write own retrieve/parse/filter/process code

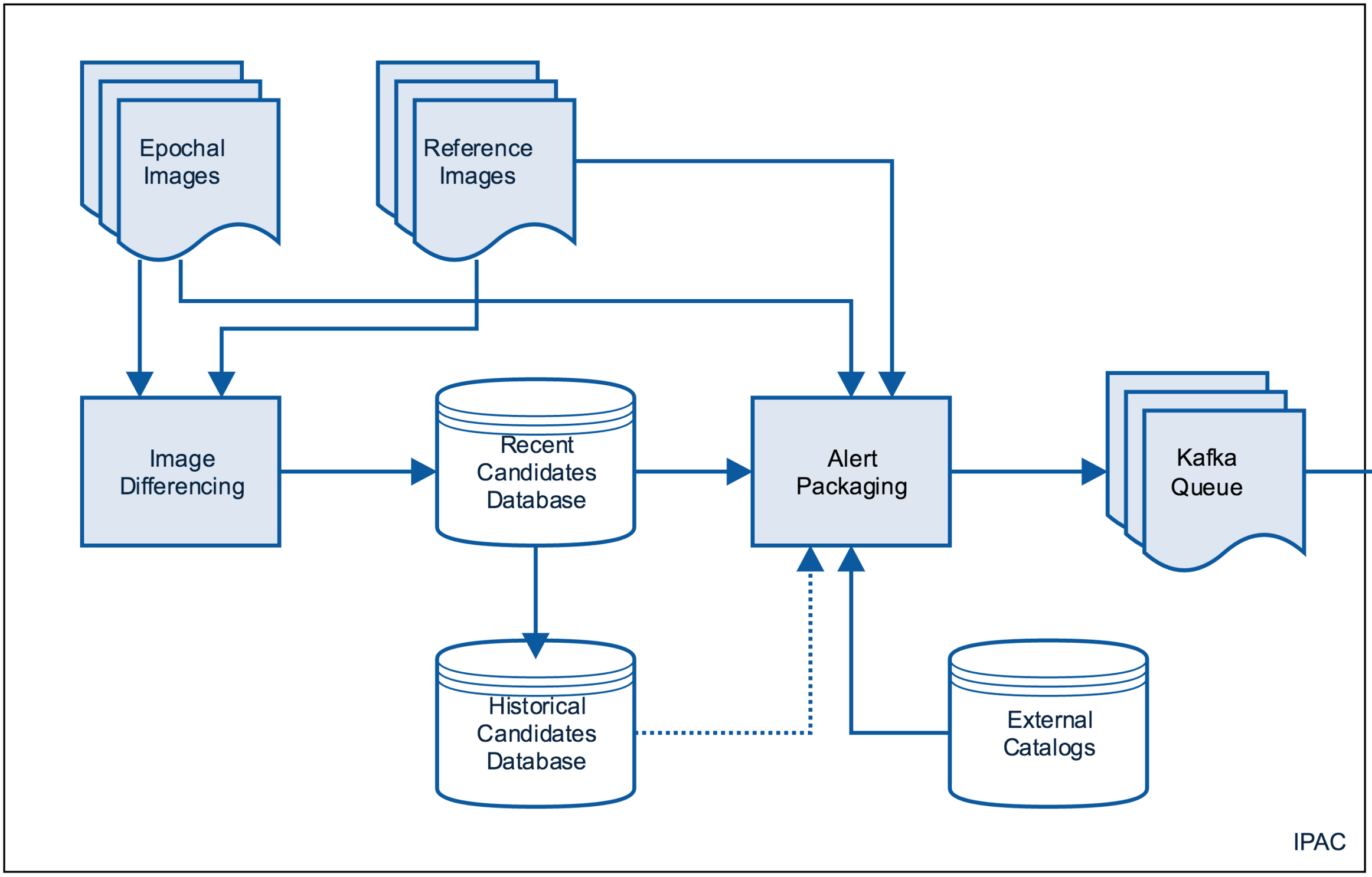
ZADS will use Kafka, an industrial queue system.



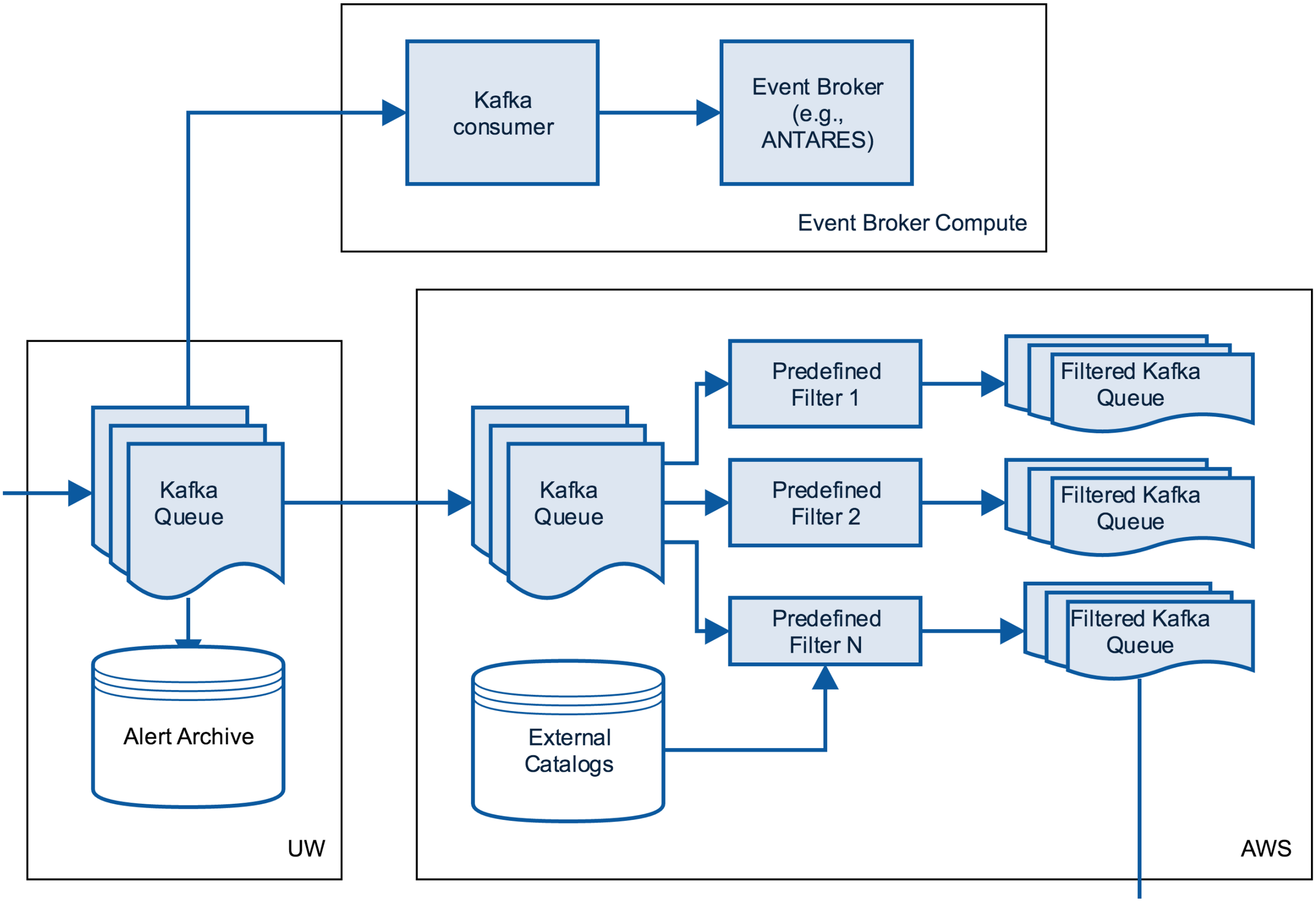
APACHE kafka™
A distributed streaming platform

find events by *filtering* alert stream (stream)

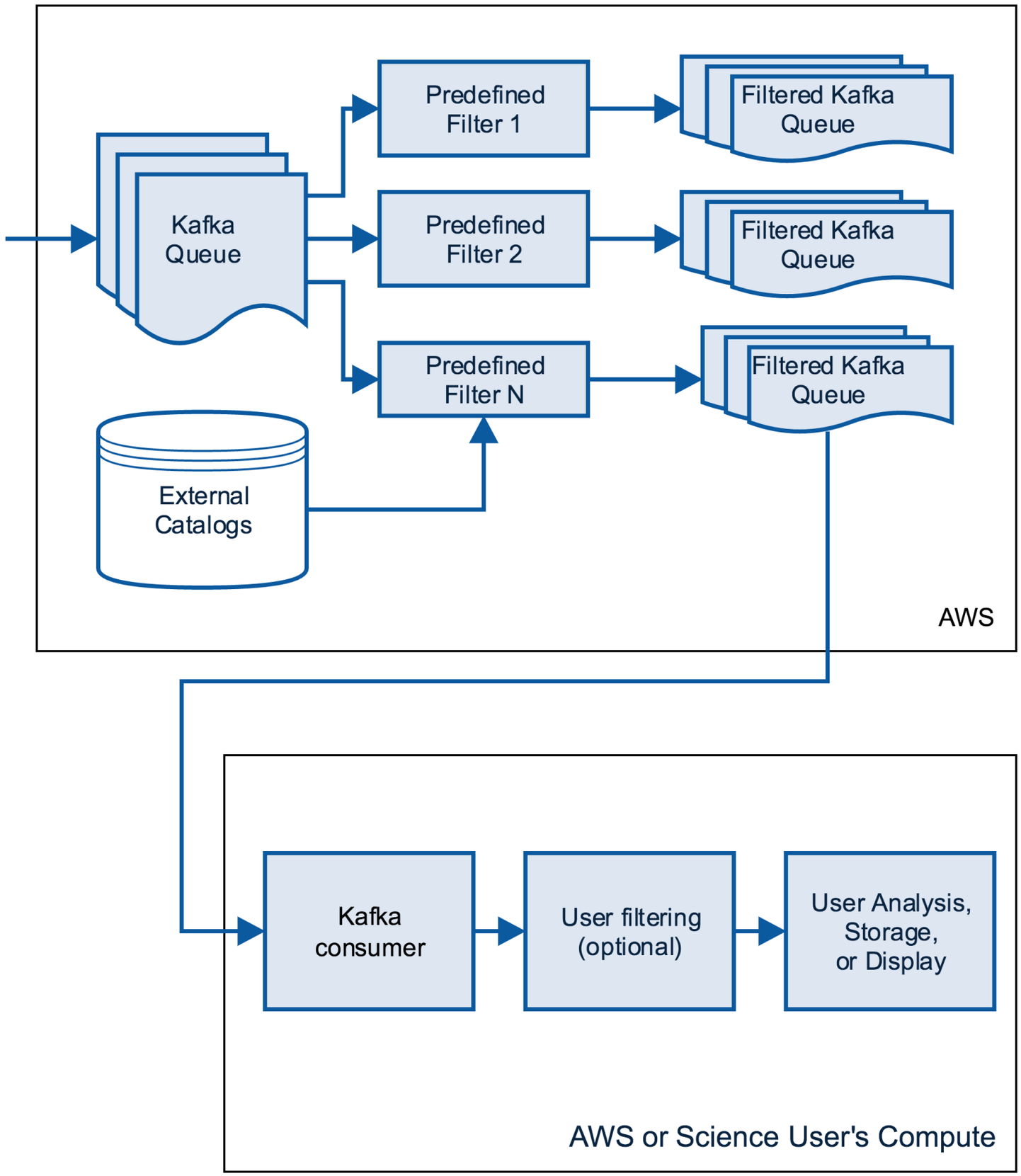
Image processing and alert packaging happens at IPAC.



ZADS feeds event brokers and a filter system.



We expect marshals will consume filtered streams.



Kafka offers several advantages over file-based retrieval.

file-based candidate retrieval	ZADS Kafka-based system
No way to control outbound bandwidth (denial of service)	Per-user throttling
latency between image processing and user retrieval	Queued events available immediately after ingest
Users must check manually for new candidates	ZADS consumers receive alerts automatically
Downstream consumers must parse candidate files, infer types	Existing libraries for parsing; formats enforced by schema
Users must pull full stream to apply filters	Flexible filtering service can be built naturally “in line”

ZADS will use rich alert packets.

ztf.alert https://zwickytransientfacility.github.io/ztf-avro-alert/

The top-level alert contains the following fields:

Field	Type	Contents
`alertId`	long	unique identifier for the alert
`candid`	long	unique identifier for the subtraction candidate
`candidate`	`ztf.alert.candidate`	candidate record
`prv_candidates`	array of `ztf.alert.prv_candidate` or null	candidate records for 30 days' past history
`cutoutScience`	`ztf.alert.cutout` or null	cutout of the science image
`cutoutTemplate`	`ztf.alert.cutout` or null	cutout of the coadded reference image
`cutoutDifference`	`ztf.alert.cutout` or null	cutout of the resulting difference image

candidates record contains:
position, time, filter, magnitudes, Real/Bogus score, distance to nearest reference source, PSF metrics, solar system counterpart (if applicable), star/galaxy score, number of past detections in the survey, number of past observations

Who receives ZADS streams? (preliminary)

	MSIP observations	Collaboration observations
full stream	finite number (TBD) of external “brokers” or “TOMs”	?
filtered sub-streams	external science users; public marshals; ZTF collaboration users & marshals	ZTF collaboration users & marshals

ZADS Filtering Service

ZADS will provide a filtering service.

In an alert-based architecture, filtering is critical for science productivity: return only the subset of events of interest

ZADS provides a natural *stream->filter->stream* interface: input stream and output stream have same UI

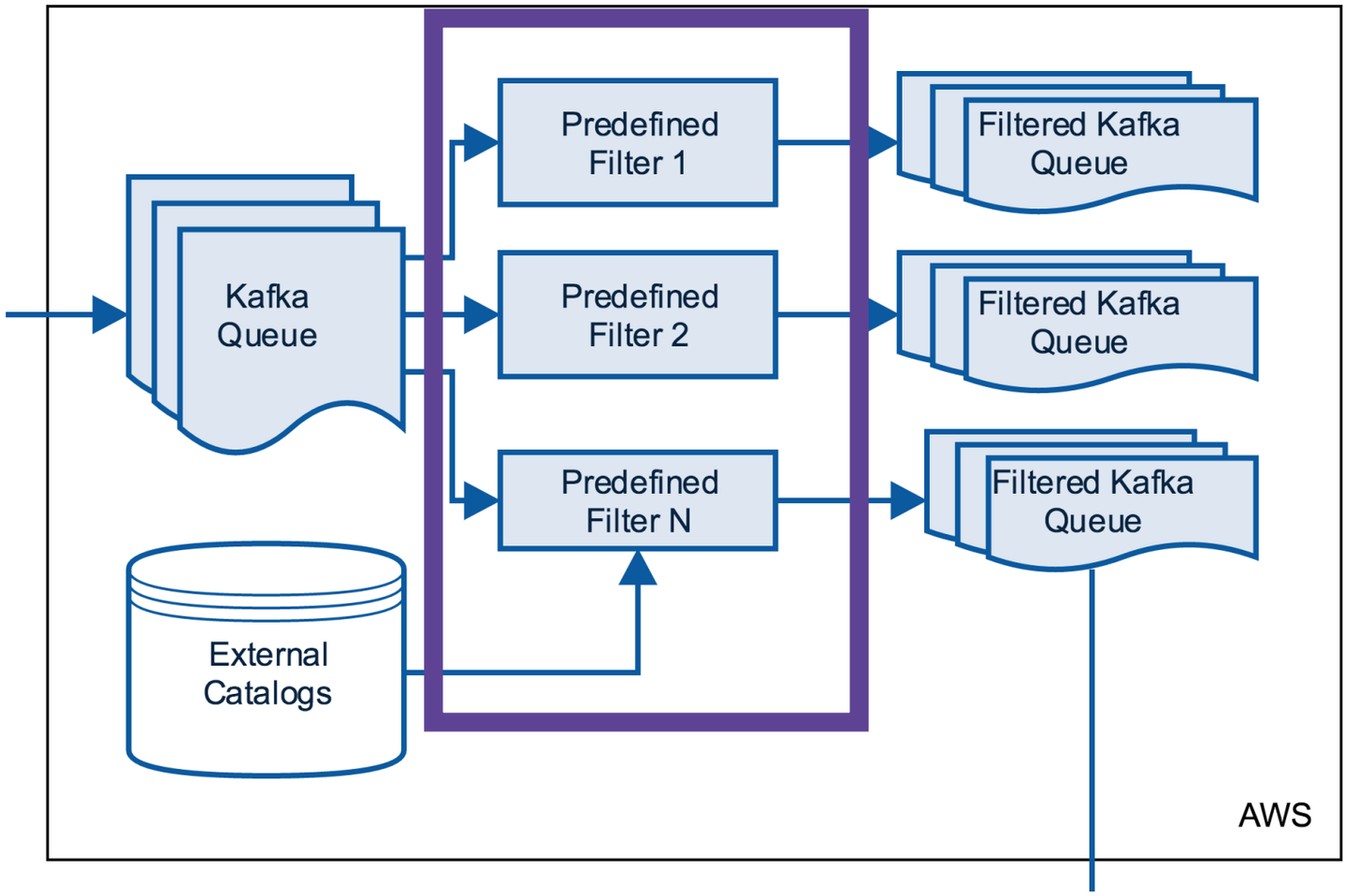
Containerized filter service: downstream users can append further filters (re)using the same code.

We propose to *begin* by building a set of ~10 hard-coded filters to get up and running.

E.g., listen to the Young Supernova channel, or potential asteroids channel

Expand to more sophisticated approaches as time and resources allow—no inherent technical limitation.

We envision building the ZADS filtering service on Spark Streaming.



APACHE Spark™ Streaming

Distributed, scalable, open source
Filters can be written in pure Python
Supports joins to other data/streams
(Batch processing of potential alert archive)

Filters can be organized into a mechanistic taxonomy.

Filters operating only on data within a single alert packet

Filters operating on an aggregated stream of alert packets

Filters incorporating external information

Single-packet alert filtering covers many use cases.

Description	Examples
Drop columns/alert fields	Remove image cutouts from the alert packet
Filter on a scalar value in the packet	RB cut, star-galaxy cut
Filter on logical combinations of several fields	PS1 color cut AND outburst amplitude cut
Filter on past detection history	Two detections separated by > 20 minutes with no previous
Filter on image cutouts	User-computed RB computation
Filter on a classifying model	Goodness of fit to a SN Ia lightcurve model

Aggregated stream use cases center on QA.

Description	Examples
Filter on alert rate	Reject (or sample/throttle) alerts which occur at a higher-than expected incidence per time
Filter on spatial correlation with other ZTF candidates	Reject events occurring in spatial proximity, such as around a saturated star bleed trail

Possible in Spark Streaming, but not envisioned for initial implementation

Filters can add value with external information.

Description	Examples
Filter on cross-match with external static catalogs	Return events coincident with GALEX sources
Filter on temporal & spatial correlation with other alert streams	Identify events in Advanced LIGO error boxes

Possible in Spark Streaming.
Crossmatch coming once the basic service is deployed

What are intermediate-term goals for the filtering service?

Crossmatching to additional catalogs within ZADS

Supported by Spark Streaming; just need time to build it

Filtering within ZADS based on parameters that change

(e.g., field id; changing RB cut values)

This is hard in any alert-based architecture!

Clients can filter broad queries further, downstream, but must transfer the full stream

Goal:

DIY service for users to upload/clone filters and direct them to a new Kafka endpoint

Replay from archive

We need the science teams to develop filters.

Pseudocode only; UW team will deal with implementation and deployment for initial set.

"Two detections tonight separated by > 30 minutes and no previous detections"

"Has stellar counterpart & this detection > 5x RMS of past history"

First filters implemented will be single packet only;
consult documentation for contents

Priorities for crossmatch catalogs and other features also helpful.

Current development status

Initial alert packet format implemented at IPAC.

Sample ZTF packets (from random simulated input) produced

Kafka alert distribution and Spark filtering systems prototyped

UW/AWS hosting being arranged; initial deployment this month

Science collaborations solicited for initial filters

Documentation work beginning, focus on interfaces & sample code

Open question: how do we best architect the marshals?