Korea Astronomy & Space Science Institute

Applications of the in-memory database Redis in processing transient event alerts

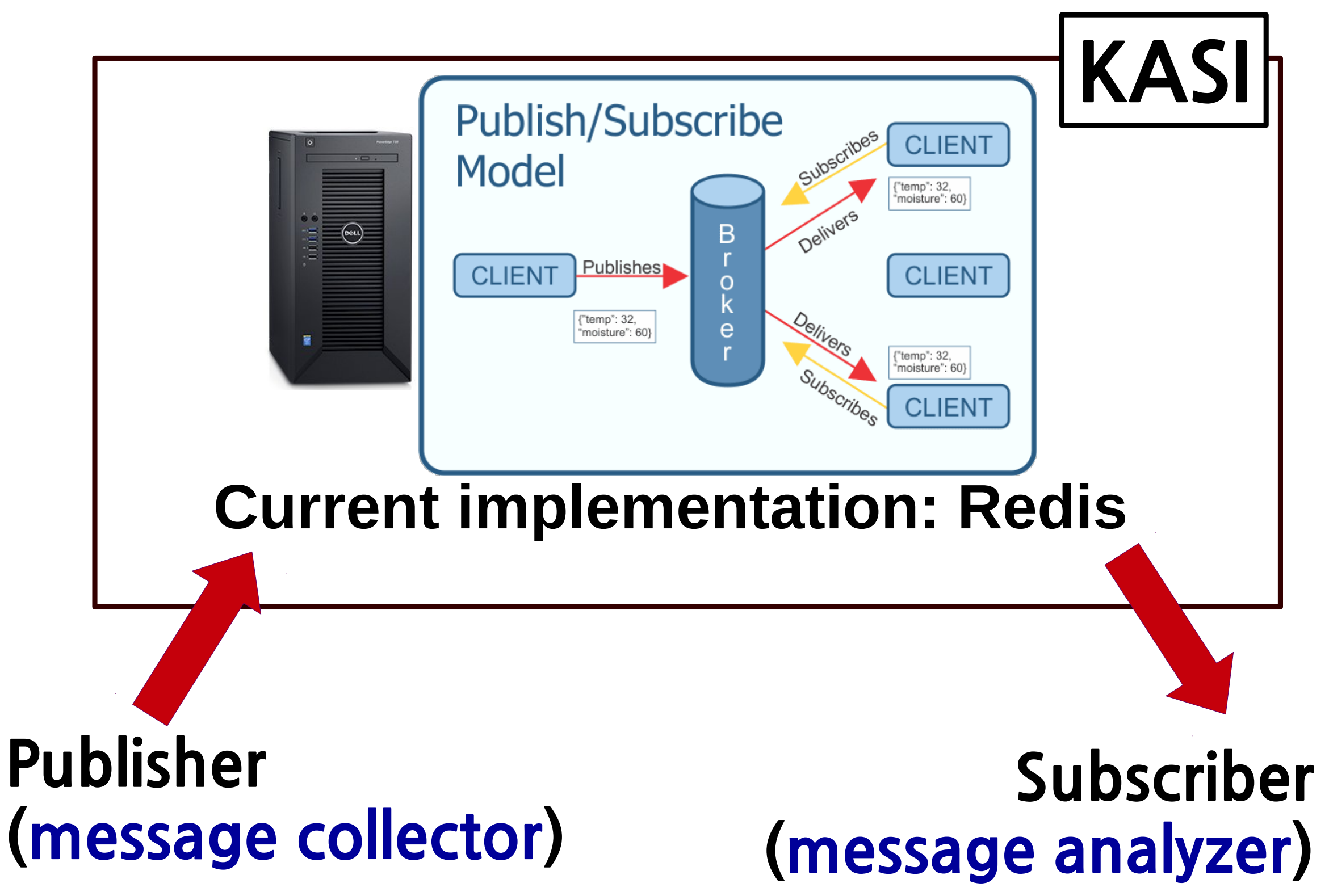
Min-Su Shin (Korea Astronomy and Space Science Institute) msshin@kasi.re.kr

1. Introduction

We use the Redis in two different ways for processing alerts:

1) the publication-subscription model as a message delivery system for multiple local alert clients,
2) storing spatial information to enable low-latency matching of transient locations with (custom) catalogs.

The current system collects event alerts by using VOEvent streams and detecting changes in web pages/feeds.



Current implementation: Redis

Publisher (message collector)

Subscriber (message analyzer)

2. Redis: pub-sub model

The system is made of:

1) **message collectors** of VOEvent alerts, web pages, RSS feeds, etc., publishing the messages to the Redis for local message subscribers,

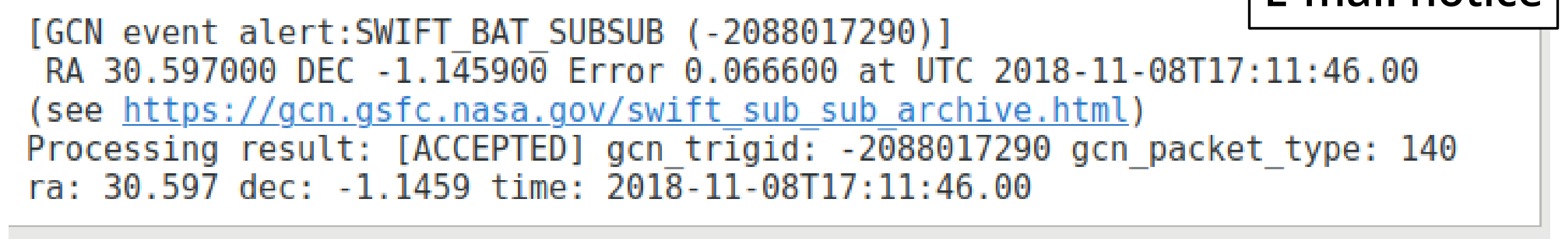
2) **message analyzers** conducting a specific task such as storing the message in a time-series database, filtering out the messages, cross-matching the message to preexisting catalogs, etc.

The **modular structure** allows the KASI members to add their own publishers and subscribers to the current system for given local APIs.

Example subscribers

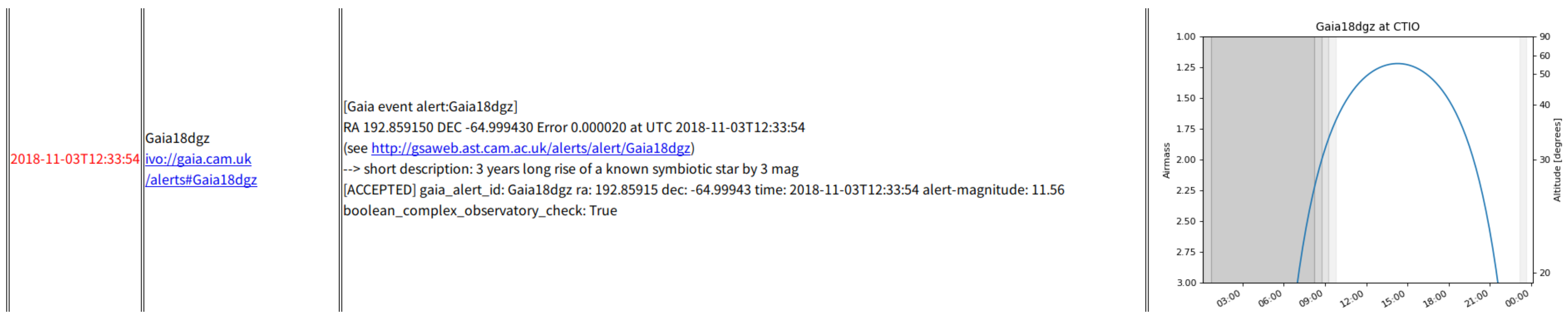
-**Subscribers for specific observing facilities** such as the Korean VLBI network and the Korean Microlensing Telescope Network: filtering and ranking alerts and producing customized notices for the observatories.
-**Subscribers for specific science projects**: the specialized analysis of alerts and ingesting/searching with local alert and science object databases.
-**Subscribers for monitoring and development**: testing machine learning algorithms of low-latency classification for given limited amount data in alerts.

E-mail notice



```
[GCN event alert:SWIFT_BAT_SUBSUB (-2088017290)]
 RA 30.597000 DEC -1.145900 Error 0.066600 at UTC 2018-11-08T17:11:46.00
(see https://gcn.gsfc.nasa.gov/swift_sub_sub_archive.html)
Processing result: [ACCEPTED] gcn_trigid: -2088017290 gcn_packet_type: 140
ra: 30.597 dec: -1.1459 time: 2018-11-08T17:11:46.00
```

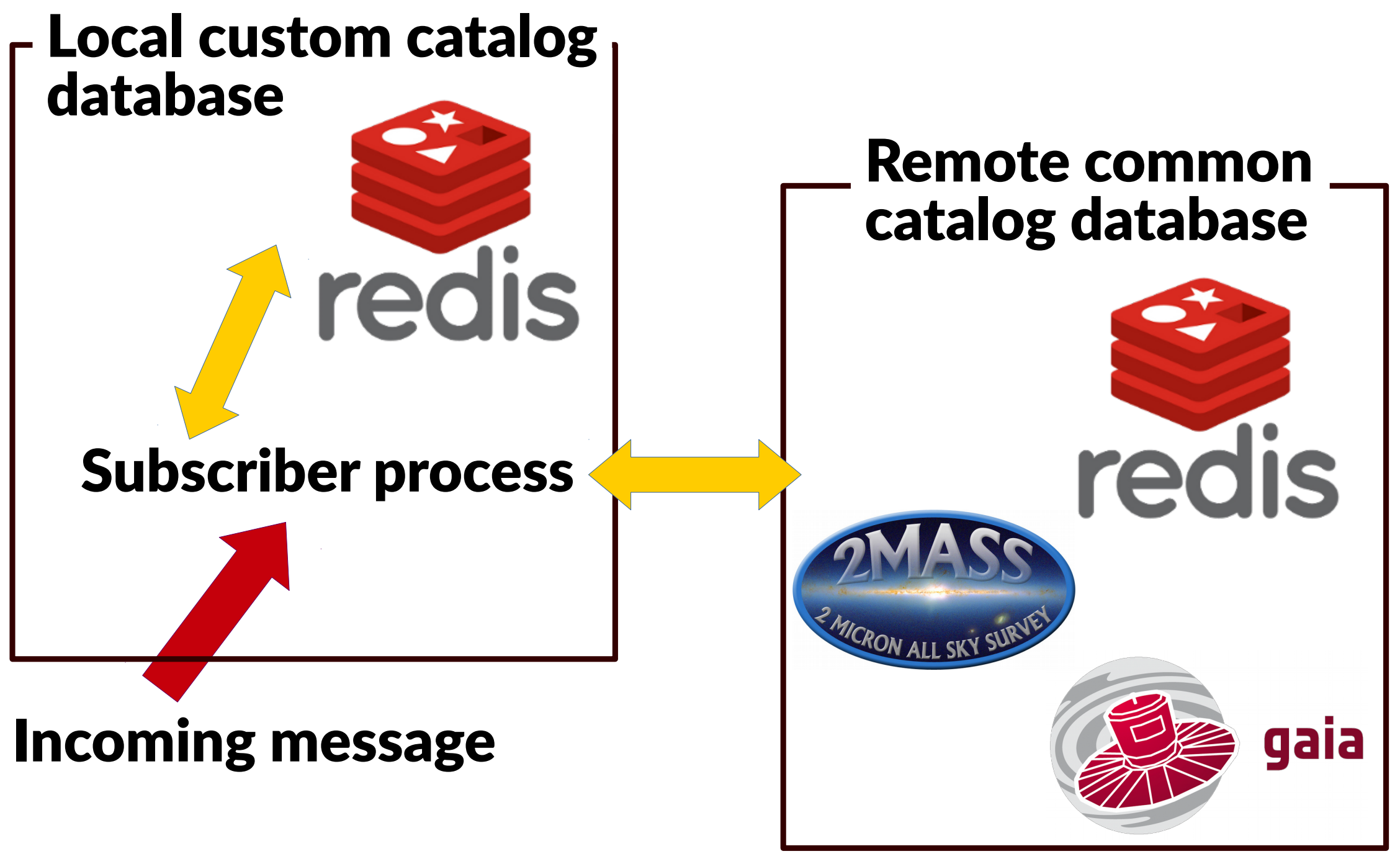
Web notice



2018-11-03T12:33:54

Gaia18dgz
ivo://gaia.cam.uk/alerts#Gaia18dgz

[Gaia event alert:Gaia18dgz]
RA 192.859150 DEC -64.999430 Error 0.000020 at UTC 2018-11-03T12:33:54
(see http://gsaweb.ast.cam.ac.uk/alerts/alert/Gaia18dgz)
--> short description: 3 years long rise of a known symbiotic star by 3 mag
[ACCEPTED] gaia_alert_id: Gaia18dgz ra: 192.85915 dec: -64.99943 time: 2018-11-03T12:33:54 alert-magnitude: 11.56
boolean_complex_observatory_check: True



Local custom catalog database

Remote common catalog database

Subscriber process

Incoming message

3. Redis: GeoSet (a sorted set with latitude and longitude)

Low-latency in-memory spatial data store for astronomical coordinates.

-We modify the Redis source code to deal with astronomical coordinates as presented in the ADASS XXVII.
-Custom catalogs with coordinates are stored and managed by each subscriber.
-A Redis server serves frequently used catalogs such as Gaia DR2 sources.
-Typical search response time is ~ microseconds to milliseconds.

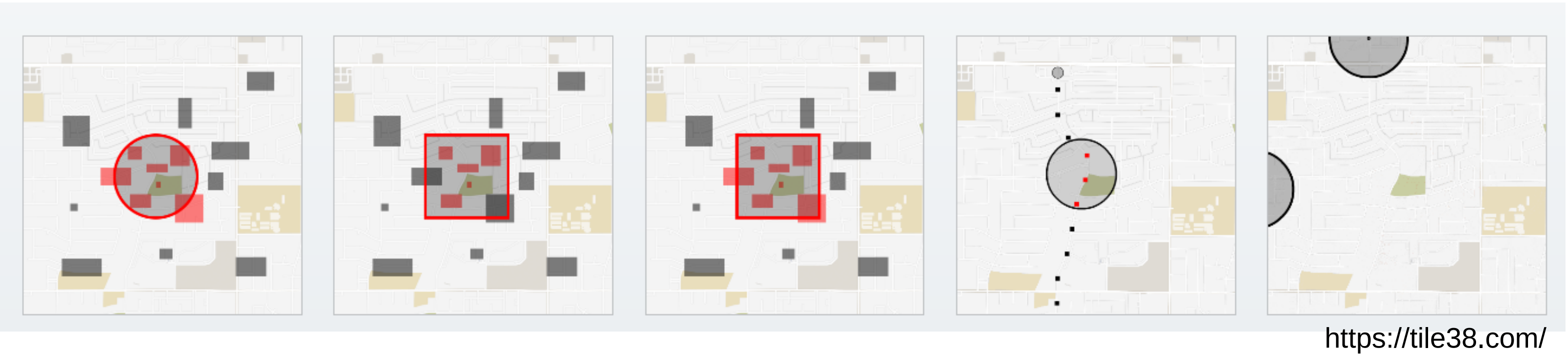
4. Exploring migration to NATS and Tile38

We are exploring the possibility of using **NATS** for the large-scale pub-sub model implementation and using **Tile38** for the low-latency spatial search with various query types. The goal is a horizontally easily scalable system in the framework of cloud computing to process 2 million messages per hour.

-NATS is an open-source, cloud-native messaging system. We are testing the NATS Streaming's pub-sub implementation.
-Tile38 supports more types of spatial query than Redis. We modified Tile38!

```
def ingest_file_into_redis(in_fn, redis_hostname):
    redis_pos_key = 'CC'
    in_key_prefix = 'CC_'
    with open(in_fn, 'r') as fd:
        fd.readline() # header
        for oneline in fd.readlines():
            temp = oneline.strip().split()
            redis_value_key = in_key_prefix + temp[0]
            use_name2 = temp[1]
            use_ra = temp[2] # deg
            use_dec = temp[3] # deg
            use_z = temp[4]
            use_r500 = temp[5] # deg
            use_list = [use_name2, use_ra, use_dec, use_z,
use_r500]
            reference.ingest_redis_coord_value(redis_pos_key,
use_ra, use_dec, redis_value_key, use_list, redis_hostname)
        fd.close()
```

Ingestion of a custom catalog



https://tile38.com/

Tile38 can store about 10M coordinates with about 5GB memory.

The both tools can be easily deployed by Kubernetes in the cloud computing environment.

5. Plan

-Conducting load tests with the current system and comparing it to the new one with the NATS and Tile38.
-Including the low-latency machine learning classification/ranking algorithm in the current system APIs.