Real-time Spatiotemporal Reasoning for Safe and Efficient Space using **AstroLibrary**

Shawn SH Choi[123], Peter JH Ryu[13], C. Song[23], H. Kim[23], J. Jang[23], M. Ji[23],
John Kim[1], Lowell Kim[1], Douglas Deok-Soo Kim[123*]

SpaceMap INC. 1, Hanyang University[2],
Voronoi Diagram Research Center[3]

There are many RSOs.

There will be many more!

Efficient solution of spatiotemporal problem is …

Challenge!

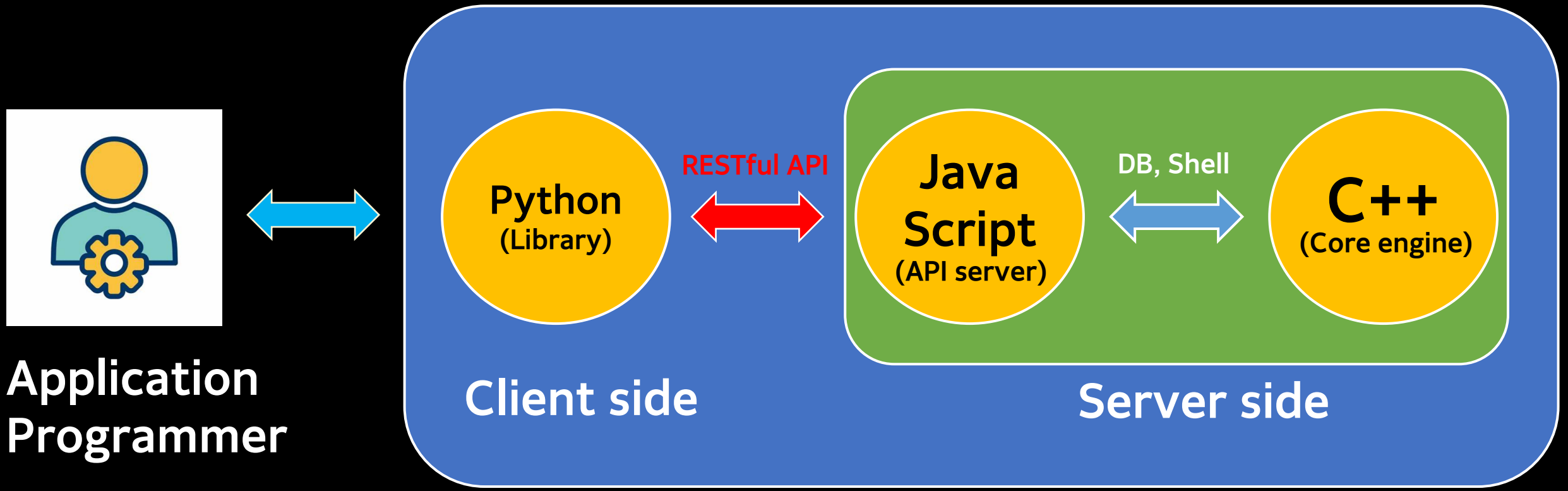
AstroLibrary ? *Why?*

To solve spatiotemporal problems in real-time

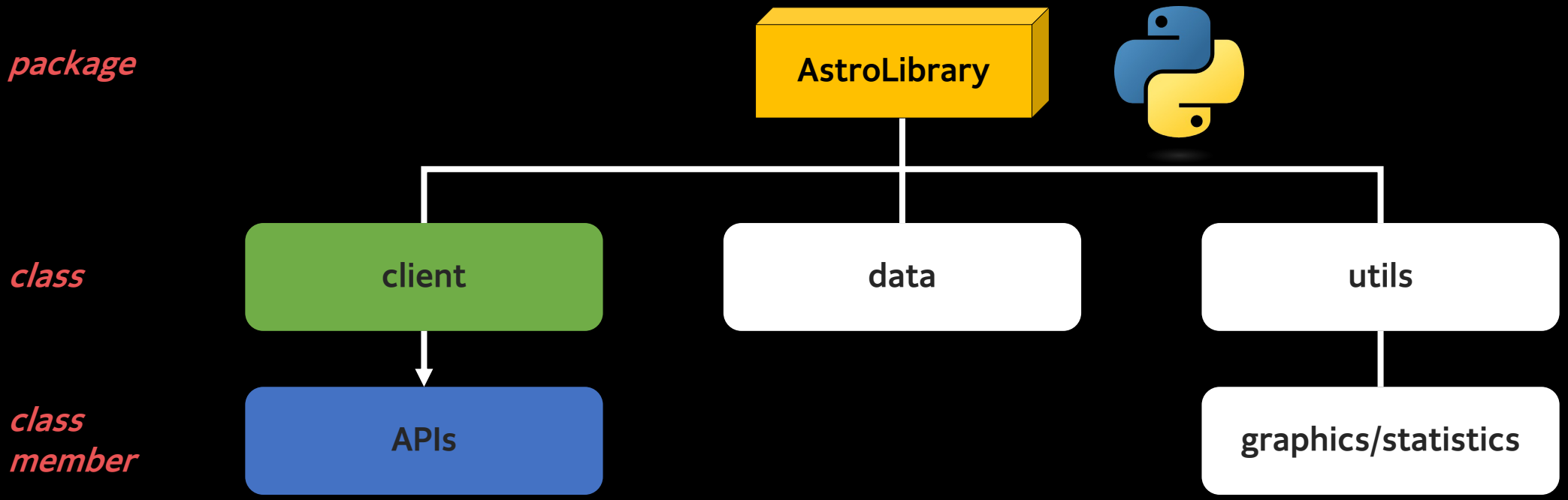
To reuse the best library

AstroLibrary : *Client – Server Model*

- **Python library** that wraps **RESTful APIs**,
- Makes application programmers' life easy



AstroLibrary : *Architecture*



- Client: A class that manages sessions and APIs
- APIs: An API class that performs calculations
- Data: A data class that stores results(e.g. Conjunctions or TLE)
- Utils: A class for post-processing the Data class (e.g. visualization, statistics)

Example1:
Search Conjunctions


```
{
    "total_count": 75,
    "current_count": 75,
    "conjunctions": [
        {
            "created_at": "2023-04-19T12:00:00.000Z",
            "primary_id": 39227,
            "primary_name": "KOMPSAT 5",
            "secondary_id": 47358,
            "secondary_name": "STARLINK-2055",
            "dca": 7.266,
            "tca": "2023-04-18T09:51:42.271Z",
            "probability": "N/A"
        },
        {
            "created_at": "2023-04-19T12:00:00.000Z",
            "primary_id": 39227,
            "primary_name": "KOMPSAT 5",
            "secondary_id": 47358,
            "secondary_name": "STARLINK-2055",
            "dca": 7.692,
            "tca": "2023-04-18T11:27:23.672Z",
            "probability": "1.876e-13"
        },
```

Example1: Search Conjunctions

```python
import astrolibrary
from astrolibrary.data.constellation import Constellation

if __name__ == "__main__":
    # create an astrolibrary client named ROK_airforce
    ROK_airforce = astrolibrary.Client('input your accees token!')

    # call api with default parameters
    result = ROK_airforce.conjunction_API.search_conjunctions()
    print(result)

    # <class 'astrolibrary.data.conjunction.Conjunction'>
    print(type(result.conjunctions[0]))

    # An error occurs because conjunction result is not a dictionary type
    # print(result['conjunctions'][0]['p_id'])    # error code
    print(result.conjunctions[0].primary_id)  # success code

    # How to call an API with differenet parameters
    result2 = ROK_airforce.conjunction_API.search_conjunctions(
        limit=5, page=10, sort="dca"
    )
    print(result2)

    # How to search for a specific space object you want
    result3 = ROK_airforce.conjunction_API.search_conjunctions(
        limit=10, sort="tca", norad_id_or_name="starlink"
    )
    print(result3)

    # API to find a conjunction between a specific satellite and satellite constellation
    result4 = ROK_airforce.conjunction_API.search_conjunctions_by_target_object(
        target_norad_id=39227, constellation=Constellation.STARLINK
    )
    print(result4)
```

① ②

1. Create client class instance

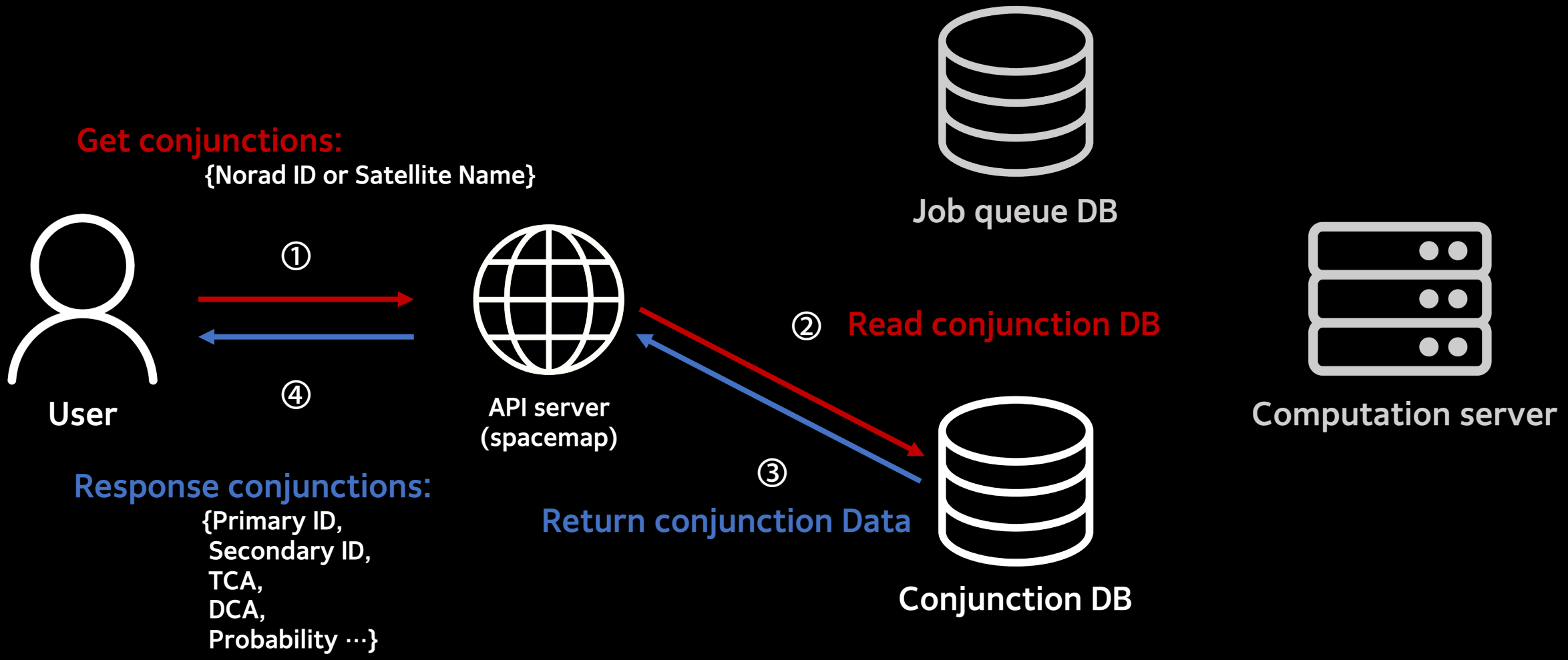
- Copy your access token from the platform website
- Use the copied token to create a client. In the example on the right, a client named ROK_airforce is created.

2. Call get conjunctions API

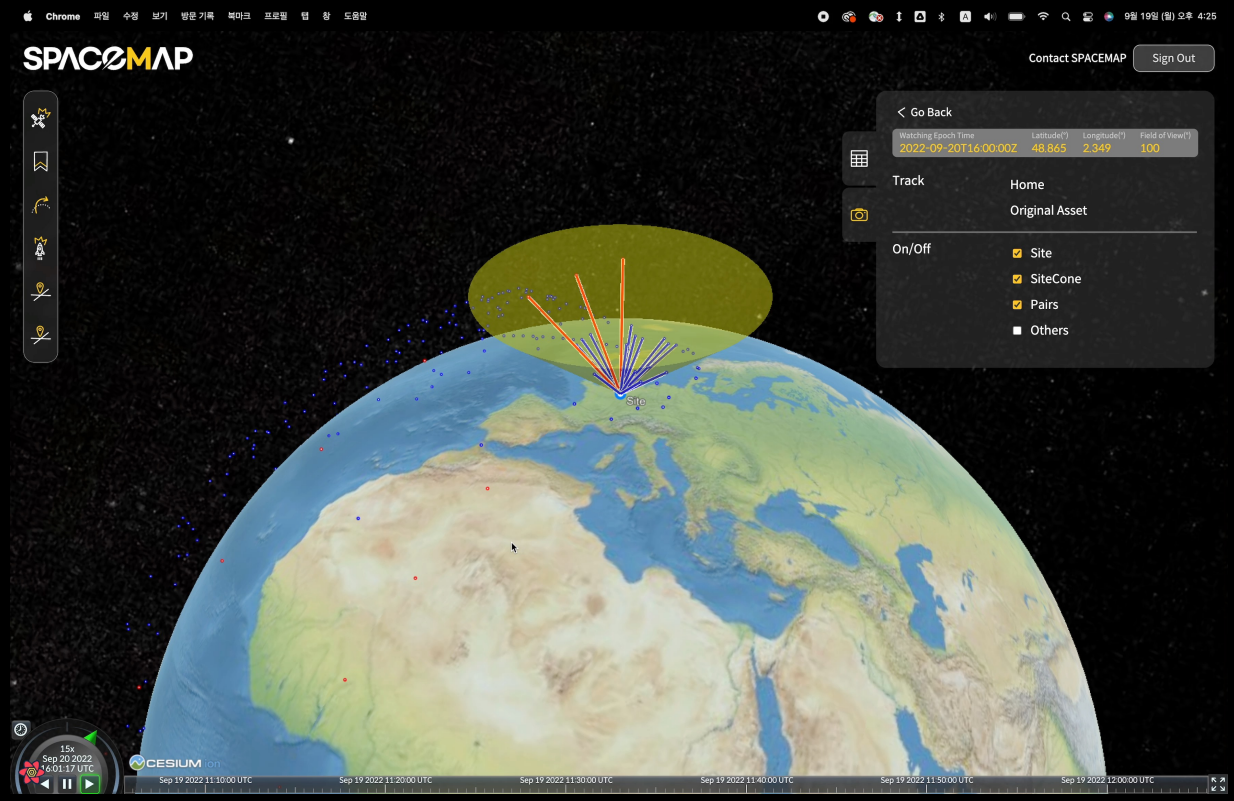
- Call search conjunctions API with default parameters.
- As shown in the previous slide, the output is in JSON format but the actual data type is a conjunction class.
- You can use the API in various ways, such as specifying the limit and page, selecting a sorting method, and designating the target space object.

Example 1: Search Conjunctions

REST API: GET /ppdb/conjunctions



Example2:
Predict watcher catchers



Input: {Apex, Cone, Timeline}

```
{
    "id": "643d0a524623d86432a04ce4",
    "apex_latitude": 37.5326,
    "apex_longitude": 127.024612,
    "cone_range": 2000,
    "cone_field_of_view": 40,
    "start_time_of_timeline": "2023-04-17T08:58:58.170Z",
    "end_time_of_timeline": "2023-04-17T09:58:58.170Z",
    "prediction_epoch_time": "2023-04-16T09:00:00.000Z",
    "watching_time_interval": [
        {
            "primary_id": 0,
            "primary_name": "Site",
            "secondary_id": 7061,
            "secondary_name": "DELTA 1 DEB",
            "start_time_of_time_interval": "2023-04-17T08:59:03.000Z",
            "end_time_of_time_interval": "2023-04-17T08:59:22.000Z"
        },
        {
            "primary_id": 0,
            "primary_name": "Site",
            "secondary_id": 8179,
            "secondary_name": "THORAD DELTA 1 DEB",
            "start_time_of_time_interval": "2023-04-17T08:59:03.000Z",
            "end_time_of_time_interval": "2023-04-17T08:59:57.000Z"
        }
    ]
}
```

Output: {Watching Time Interval}

Example2: Predict Watcher-Catchers

```python
import astrolibrary

if __name__ == "__main__":
    # create an astrolibrary client named ROK_airforce
    ROK_airforce = astrolibrary.Client("input access token")

    # 1. call api with default parameters
    # send request to watcher catcher server
    response = ROK_airforce.watcher_catcher_API.predict_watcher_catcher()

    # 2. list of statuses of requests sent to the watcher catcher server
    request_list = ROK_airforce.watcher_catcher_API.get_requests_status_list()["data"]
    print(request_list)

    # 3. Select a very last request from the list and read data
    # Save the database id, then read the value from that database
    # If you want to get the previous result rather than the last one, you can query the database.
    id = ROK_airforce.watcher_catcher_API.get_requests_status_list()["data"][-1]["_id"]
    print(ROK_airforce.watcher_catcher_API.get_predicted_result(id))

    # API to clear the id of a specific database from the list
    ROK_airforce.watcher_catcher_API.delete_predicted_result(id)

    # A function that performs steps 1-3 above at once
    print(ROK_airforce.watcher_catcher_API.predict_watcher_catcher_and_get_result())
```

①

②

③

★

Create client class instance

- Use your access token to create a client.

1. Call predict watcher catcher API

- Call predict watcher catcher API with default parameters. (Apex, Cone, Timeline)

2. Call get request status list API

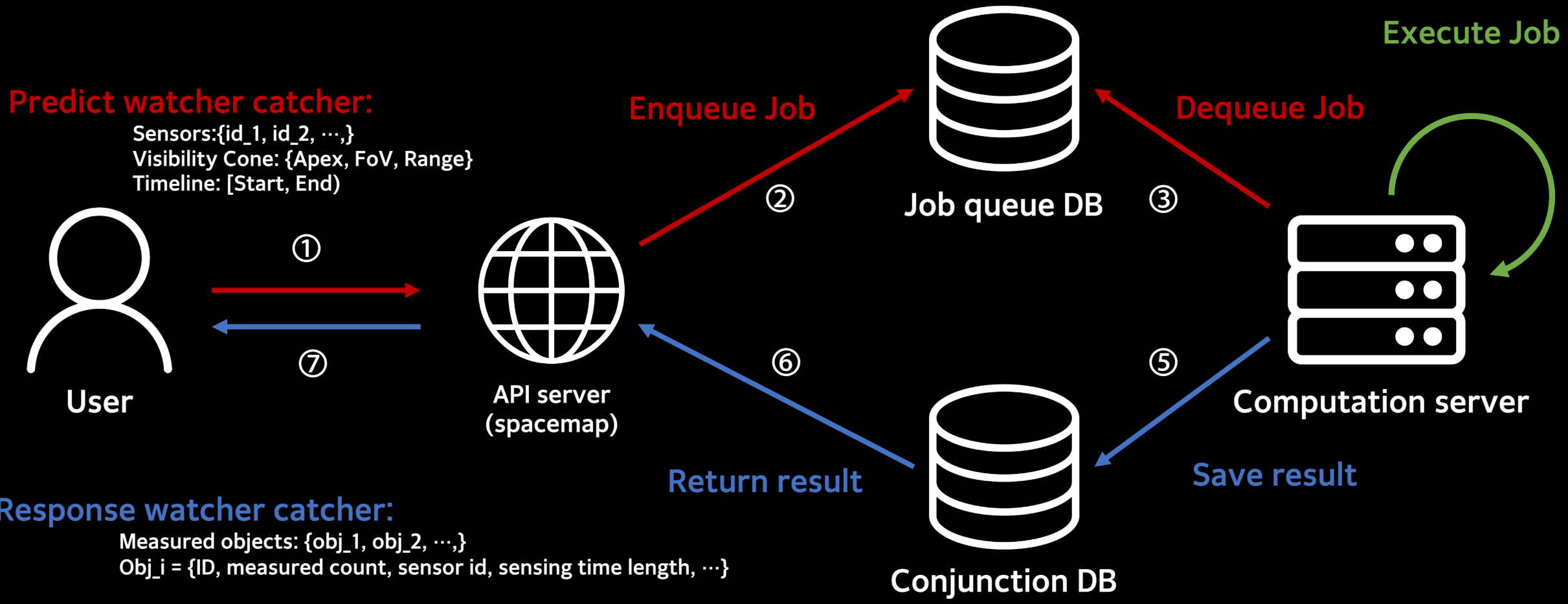
- Retrieve a status list for the watcher catcher requests previously. The status can be DONE, FAILED, PENDING.

3. Call get predicted result API

- You can select your desired request and retrieve the watcher catcher data from the database.

Example 2: Predict watcher catcher

RESTful API: POST /watcher-catcher



Summary

1. Client-Server model via RESTful APIs
2. Easy development via Python wrapping
3. Library installation via wheel file

Try us @ spacemap42.com

Thank You!

shawn.choi@spacemap42.com
douglas.kim@spacemap42.com