基于PostGIS与Protobuf的空间数据渲染

黄泰瑚


我是谁？我为什么在这里？

Where do I come from

- 北京佳格天地科技

搭建中国现代农业全产业链、
全要素的数据应用平台。

地块管理

温度 实时
-1℃

湿度 实时
96%

风级 偏南

积温(℃) 时间
2460

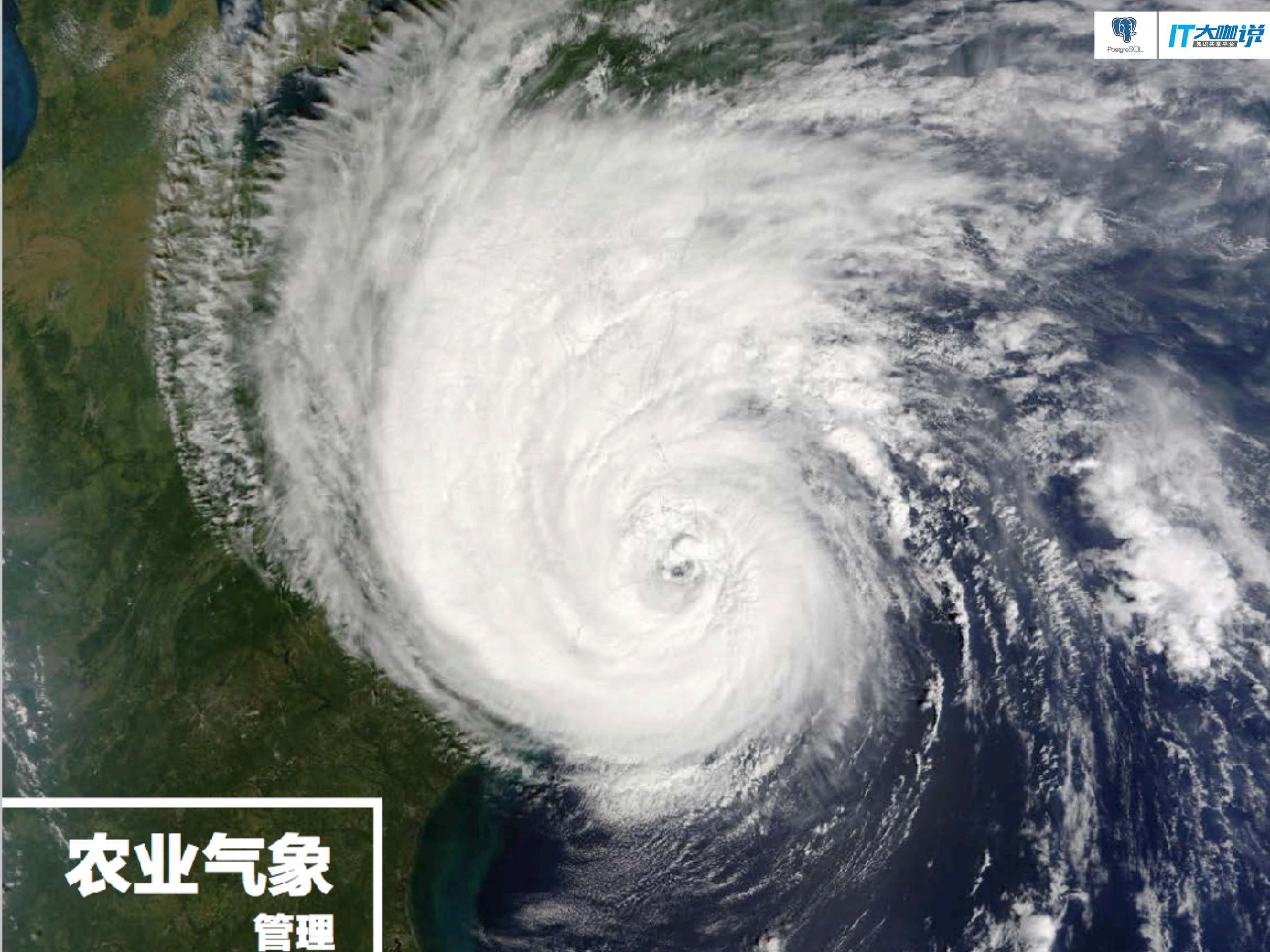
5级

地块

作物长势

100%
80%
60%
40%
20%

作物管理



农业气象管理

病虫害管理

PostgreSQL

IT大咖说 知识共享平台

病虫害预测

严重 重度 中度 轻度 轻微

低洼地势，病害多发

环保

PostgreSQL

IT大咖说 知识共享平台

农机调配

无人机实时监测

从“看天吃饭”到“知天而作”

基于PostGIS与Protobuf的空间数据渲染

Outline

- From Mysql to Postgres
- From a single process to queue based
- From hard code to library
- From now to future

From Mysql to Postgres

In actually, we used Mysql in our projects, due to our customers

And then, we suffered from it

It does not support GIS well, and it does not support lots of commands which works greatly in Postgres

Redering Vector Tiles

- In Mysql, we use Mapnik to generate tiles

    1. Generate the table of spatial reference system in database, and add some spatial references

    2. Get geojson which may be simplified from geometry data

    3. Use Mapnik to convert geojson to proto-buf

Redering Vector Tiles

- In Postgres, we just need to use a single query, no matter what you want
  - ST_AsMVT will help us generate MVT, and it is faster than Mapnik

So why to use MVT rather than Geojson ?

MVT

- Mapbox vector tiles
- It is a kind of protocol buffers which won't have any key when it is transmitted, so it is faster and smaller than Geojson
- The key is defined by Mapbox in client libraries, so you won't need to worry about that

Now, the system works…

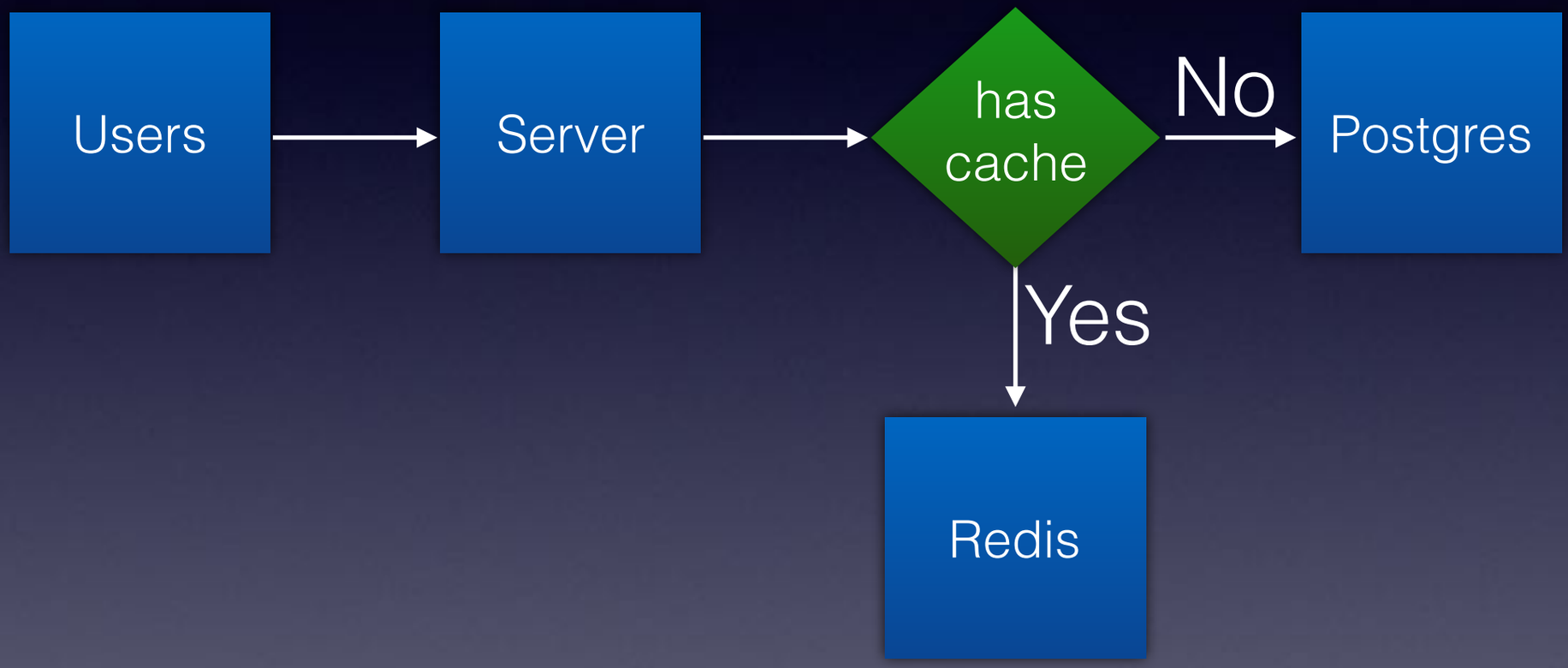
And crash, due to the poor performance

From a single process to queue based

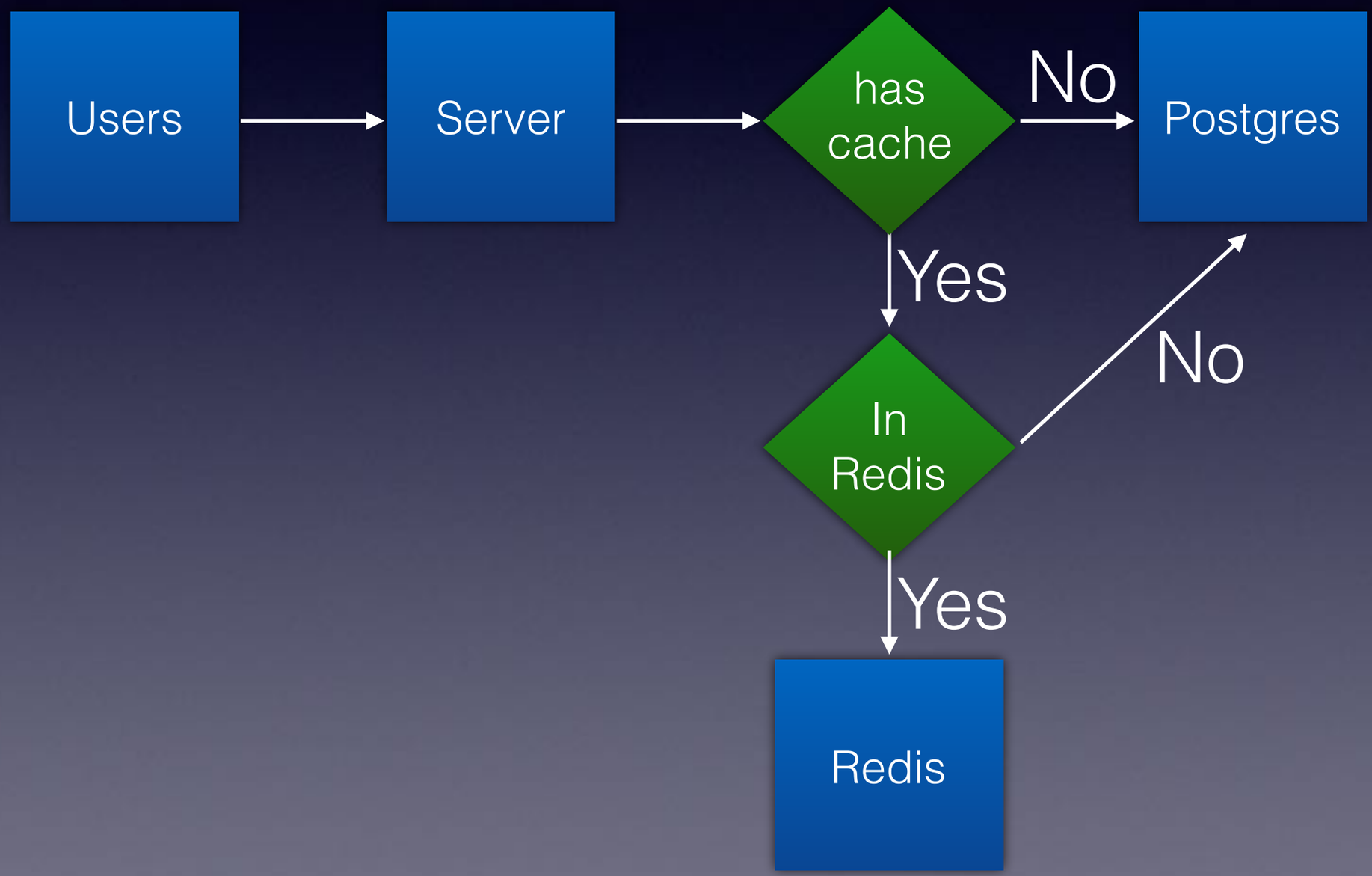
We earned money by data, but our enemy is data as well

We cannot render tiles correctly, because the amount of data, even if we did the sharding, added gist/sp-gist, etc

All of us know that we need cache which will be converted from binary to base64 string and stored into Redis


Users
Server
has cache
No
Postgres
Yes
Redis

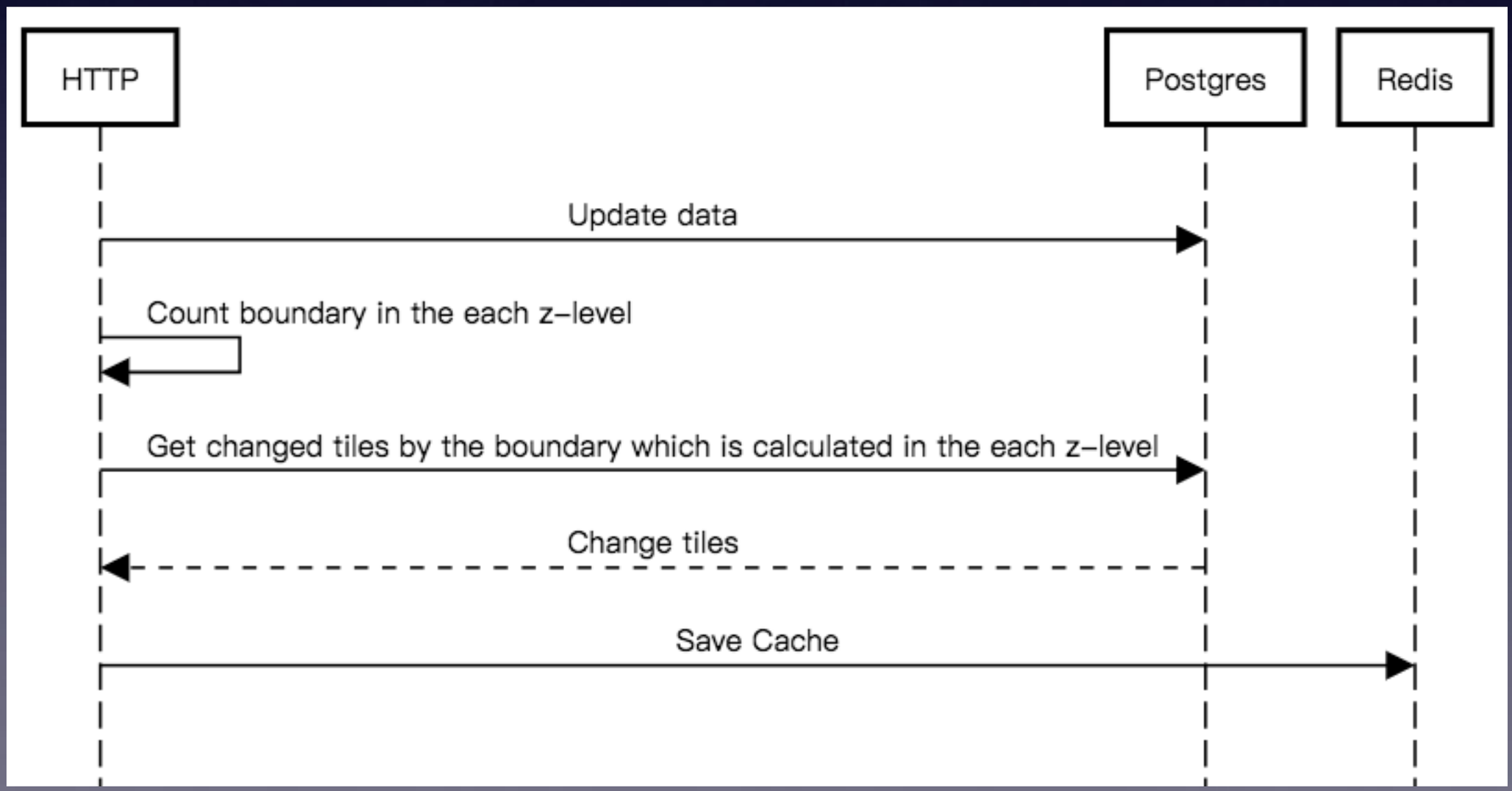
Unfortunately, our memory is not enough to store all caches, so we try to put some caches into our database


Users
Server
has cache
No
Postgres
Yes
In Redis
No
Yes
Redis

Our data will be changed, so the question is how to change our cache, now

Modify Cache

- Change it in HTTP server directly: it will affect the performance of HTTP Server



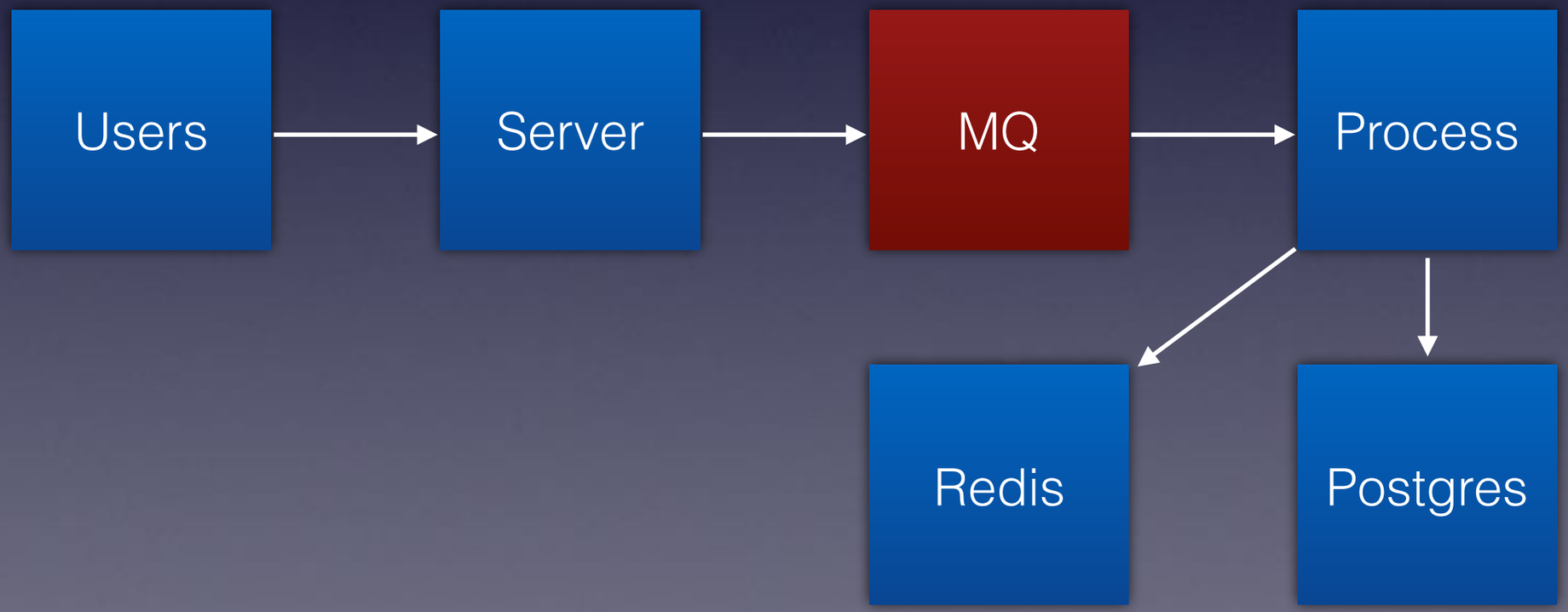
Modify Cache

- Change it by NOTIFY & LISTEN in Postgres
- The NOTIFY command sends a notification event together with an optional "payload" string to each client application that has previously executed LISTEN channel for the specified channel name in the current database.

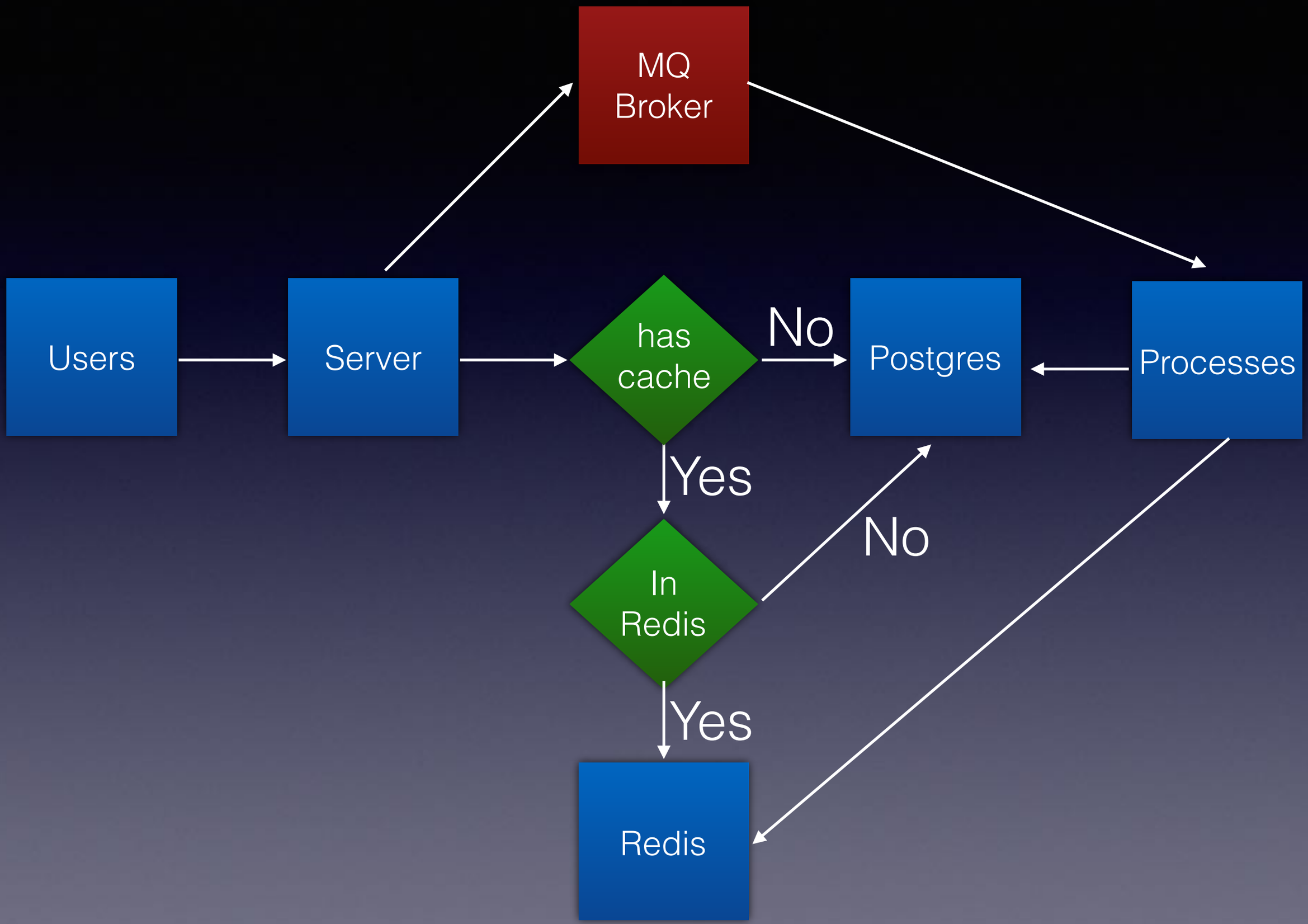
Unfortunately, it will make our processes be busy, because it will notify all listeners

Modify Cache

- To prevent from the issue of performance, we added new processes and message queues, they will be responsible for updating data and cache



Now, our architecture will become…


MQ
Broker
Users
Server
has
cache
No
Postgres
Processes
Yes
In
Redis
No
Yes
Redis

Now the system will be fine, but it hard to develop by our freshmen, so the next question is how to make the development fast and easily

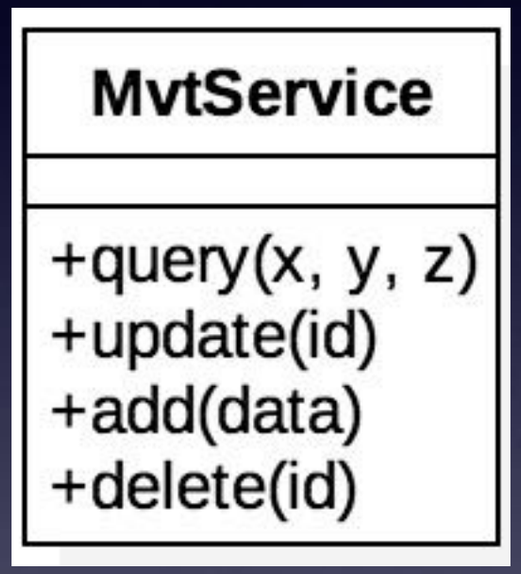
From hard code to library

Like GoF's Design

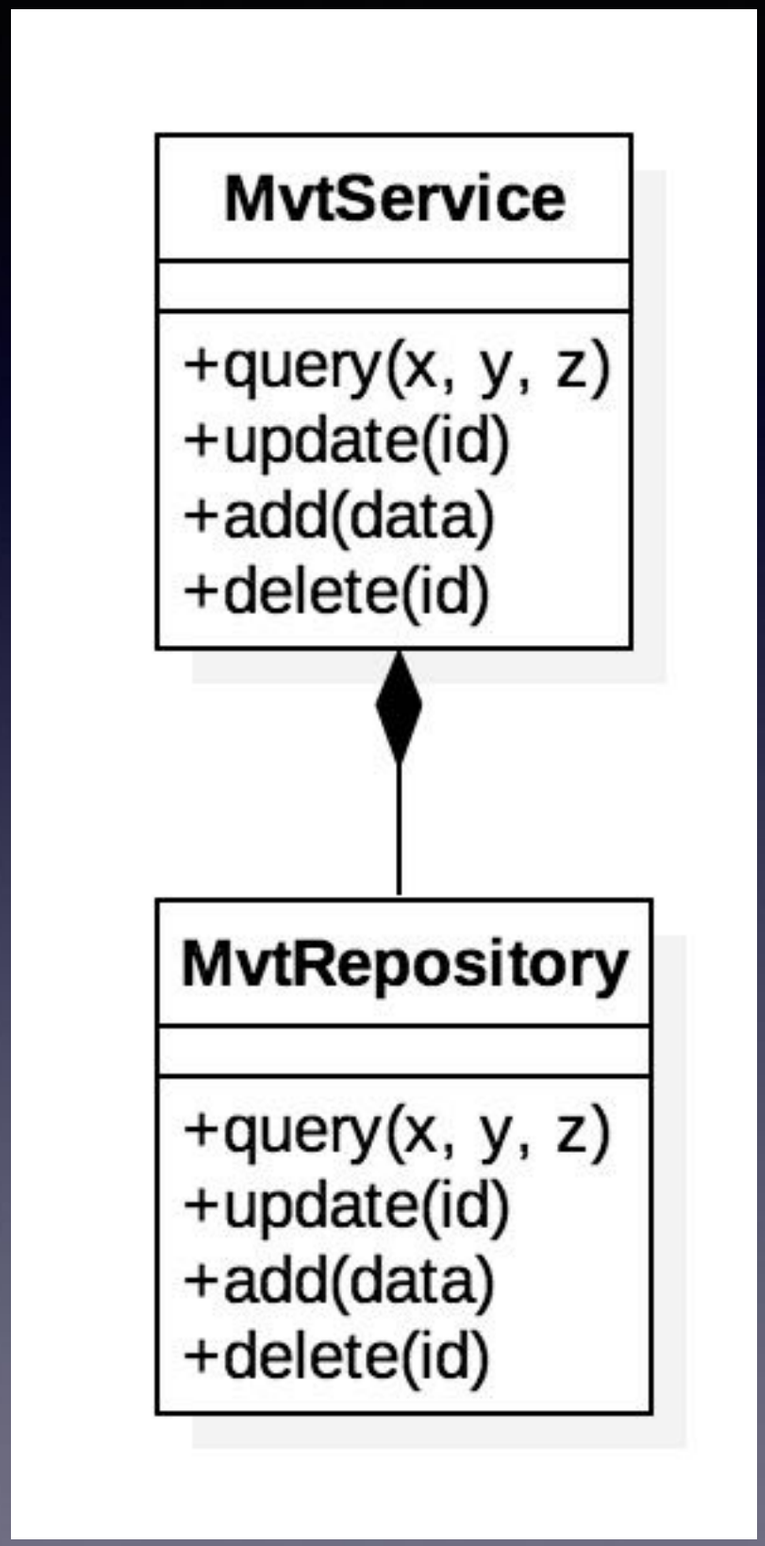
pattern, we define our context at first.

The context is that we want to have some functions to process some logic about CRUD geometry data which may be in Mysql or Postgres

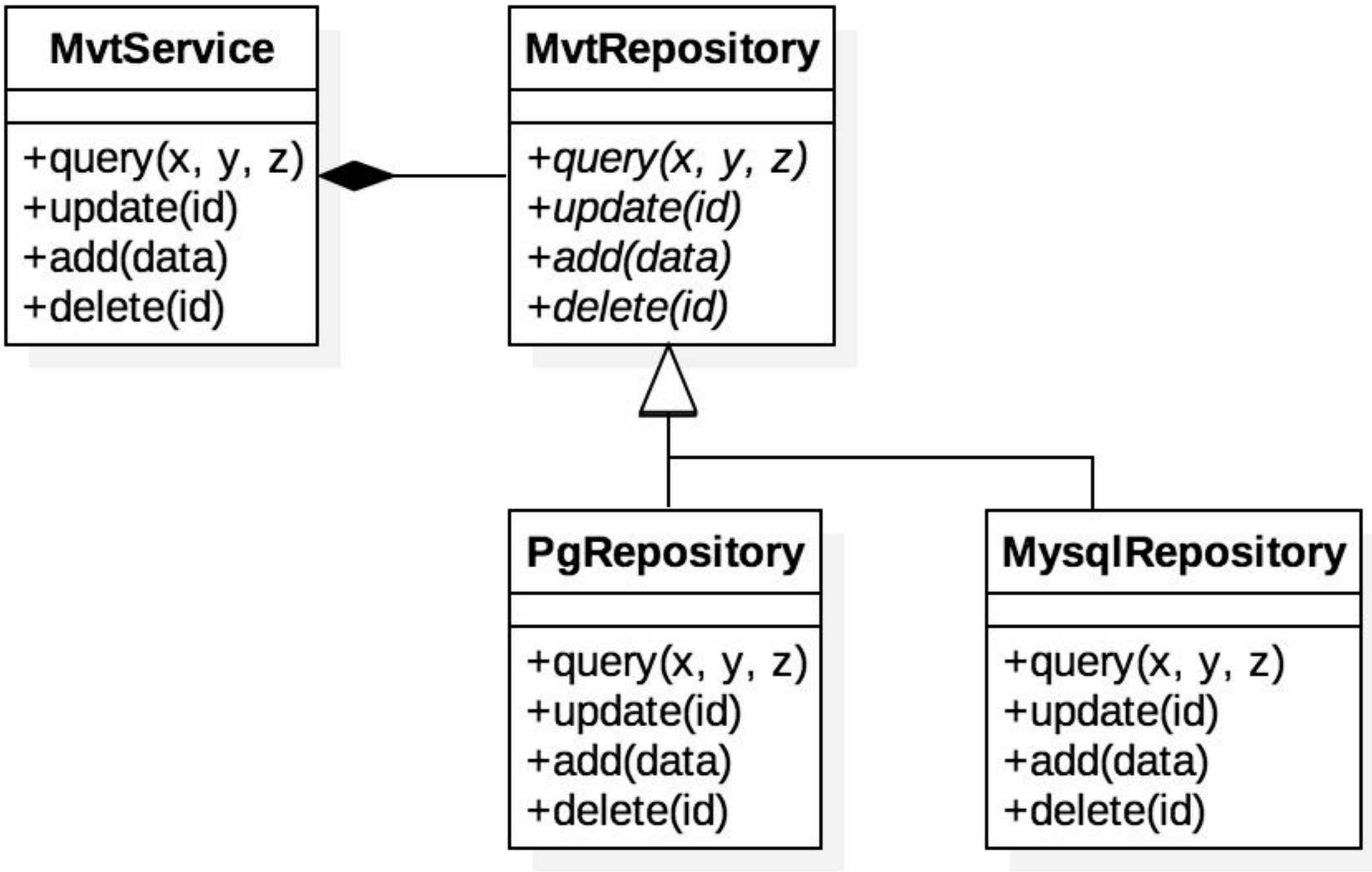
We will use Facade pattern to offer the required interface first


MvtService
+query(x, y, z)
+update(id)
+add(data)
+delete(id)

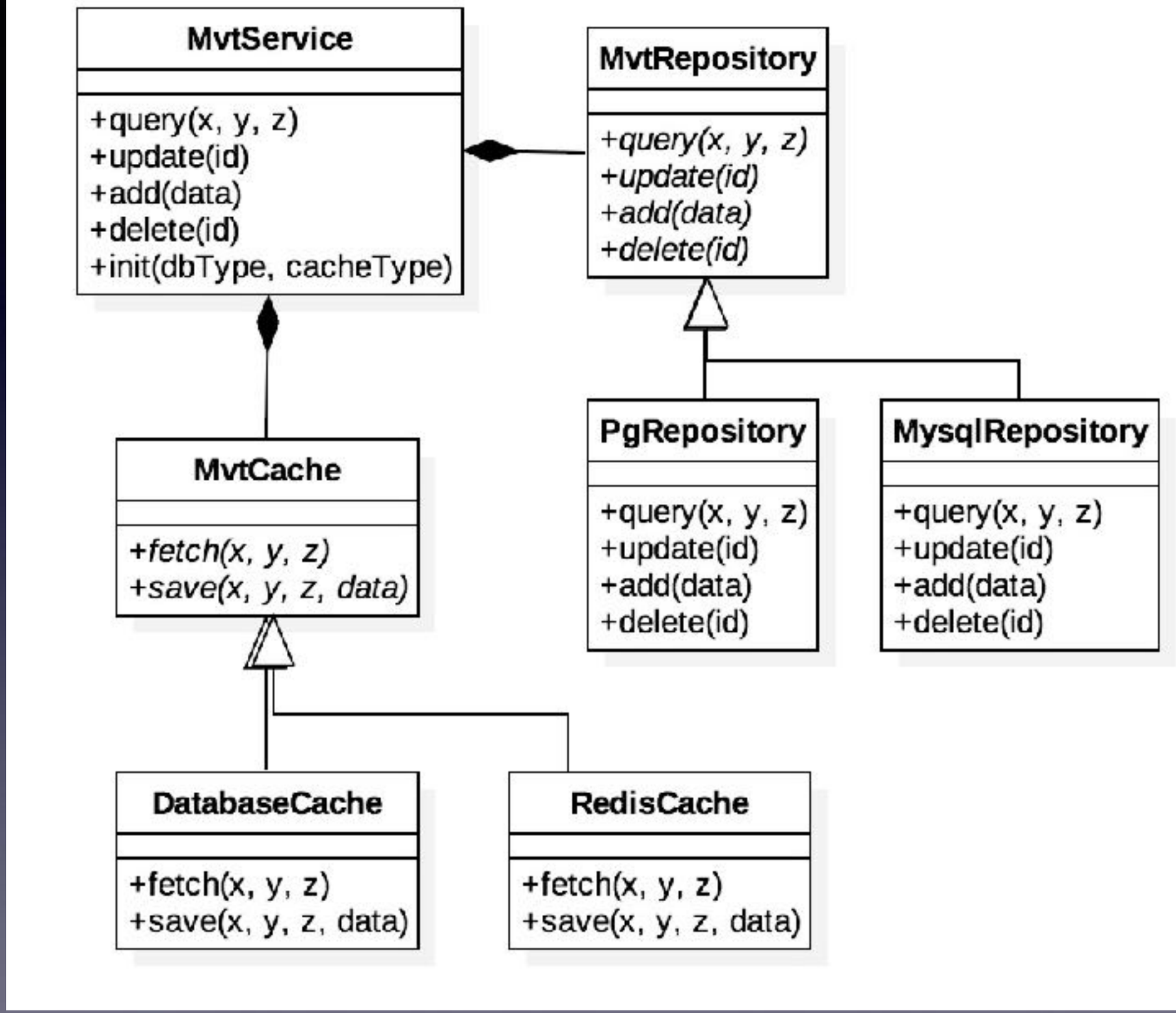
In order to make database logic decouple with main logic, we use Repository pattern


MvtService
+query(x, y, z)
+update(id)
+add(data)
+delete(id)
MvtRepository
+query(x, y, z)
+update(id)
+add(data)
+delete(id)

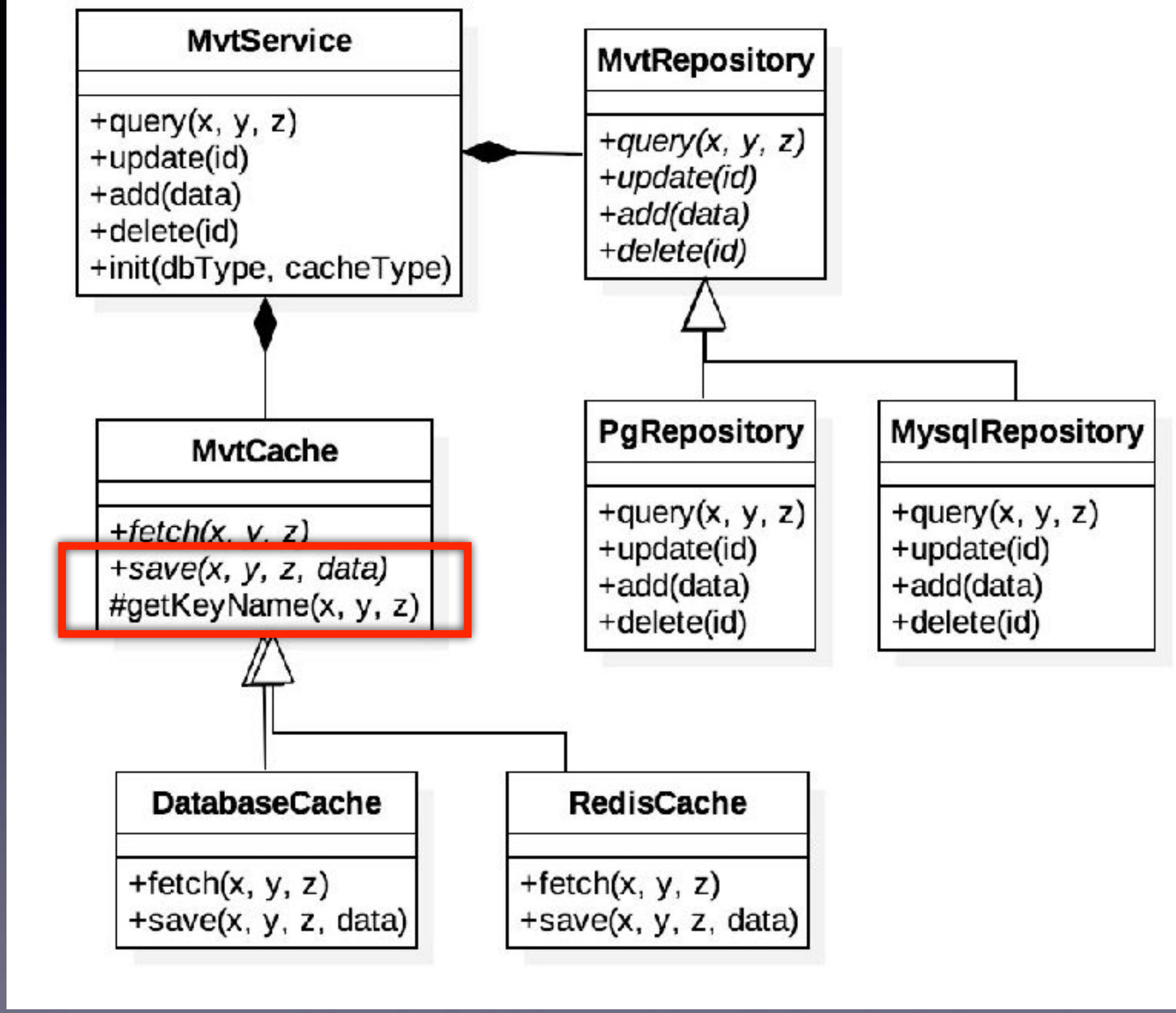
To fulfill our context, we use strategy pattern to support Postgres & Mysql


MvtService
+query(x, y, z)
+update(id)
+add(data)
+delete(id)
MvtRepository
+query(x, y, z)
+update(id)
+add(data)
+delete(id)
PgRepository
+query(x, y, z)
+update(id)
+add(data)
+delete(id)
MysqlRepository
+query(x, y, z)
+update(id)
+add(data)
+delete(id)

Don’t forget our cache architecture, we want to make it support the multiple types of cache, so we use strategy again


MvtService
+query(x, y, z)
+update(id)
+add(data)
+delete(id)
+init(dbType, cacheType)
MvtRepository
+query(x, y, z)
+update(id)
+add(data)
+delete(id)
MvtCache
+fetch(x, y, z)
+save(x, y, z, data)
PgRepository
+query(x, y, z)
+update(id)
+add(data)
+delete(id)
MysqlRepository
+query(x, y, z)
+update(id)
+add(data)
+delete(id)
DatabaseCache
+fetch(x, y, z)
+save(x, y, z, data)
RedisCache
+fetch(x, y, z)
+save(x, y, z, data)

To make the cache key name be the same, we use template method to generate key name


MvtService
+query(x, y, z)
+update(id)
+add(data)
+delete(id)
+init(dbType, cacheType)
MvtRepository
+query(x, y, z)
+update(id)
+add(data)
+delete(id)
MvtCache
+fetch(x, y, z)
+save(x, y, z, data)
#getKeyName(x, y, z)
PgRepository
+query(x, y, z)
+update(id)
+add(data)
+delete(id)
MysqlRepository
+query(x, y, z)
+update(id)
+add(data)
+delete(id)
DatabaseCache
+fetch(x, y, z)
+save(x, y, z, data)
RedisCache
+fetch(x, y, z)
+save(x, y, z, data)

It seems good, but it still needs to know that which fields are required, so we need a ORM. we use Decorator pattern to describe the shema

ORM

- Object Relational Mapping
- Converting data between incompatible type systems using object-oriented programming languages

```
@TableName("plot")
export class Land extends Model {

  @Column("id", SqlType.BIGINT, SqlFlag.PRIMARY_KEY)
  id: string;

  @Column("geom", SqlType.GEOMETRY, SqlFlag.NULLABLE)
  geometry: GeometryObject;

  @Column("identify_time", SqlType.TIMESTAMP, SqlFlag.NULLABLE)
  identifyTime: number;
}
```

Now, it could be working greatly in production, but we can do more in the future

From now to future

- Generate document automatically by our ORM
- Import data without programming, and support the multiple type
- Make the multiple map server deploy easily, even if we have used docker, it still waste lots of time to write some configurations

Thanks for listening