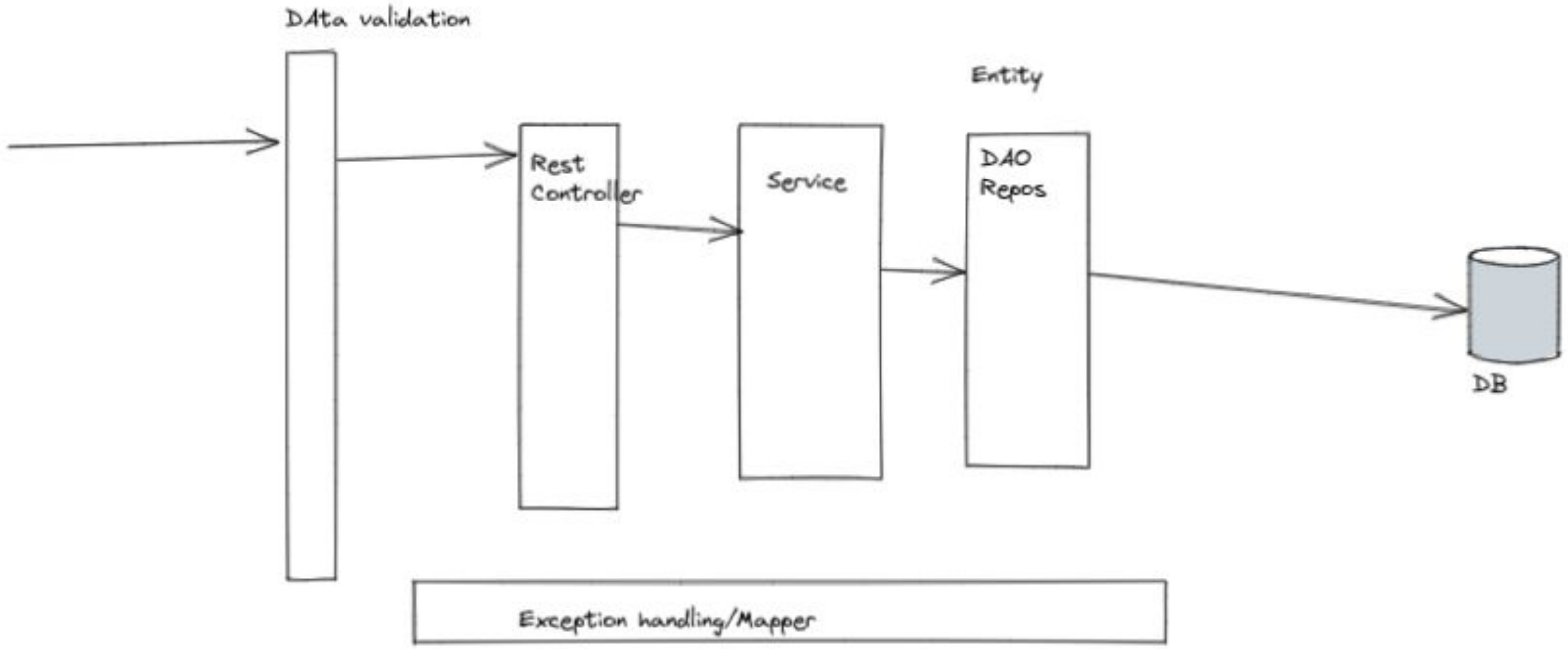
Java Backend Development Live-85

lecture-16

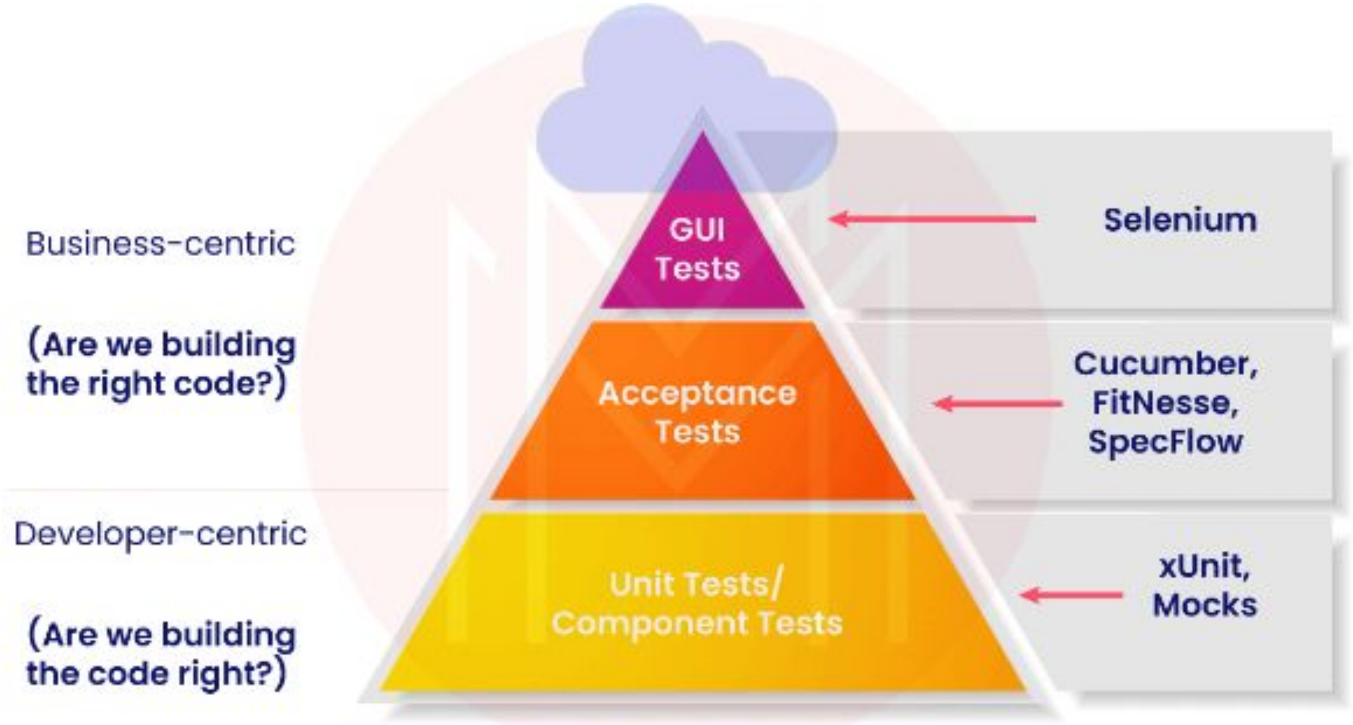
Agenda

- Data Validation Layer
- Unit Testing
  - JUnit
  - Mockito
  - H2 DB
- Integration Testing
  - @SpringBootTest
  - Code Coverage
- Cache Intro

API Request flow



Automation Testing


Business-centric
(Are we building the right code?)
Developer-centric
(Are we building the code right?)
GUI Tests
Acceptance Tests
Unit Tests/ Component Tests
Selenium
Cucumber, FitNesse, SpecFlow
xUnit, Mocks

Adaptation of Mike Cohn'stest automaton pyramid

Unit Testing

Unit testing is a type of software testing that focuses on individual units or components of a software system.

The purpose of unit testing is to test the correctness of isolated code. A unit component is an individual function or code of the application.

Unit testing helps tester and developers to understand the base of code that makes them able to change defect causing code quickly.

Unit testing fixes defects very early in the development phase that's why there is a possibility to occur a smaller number of defects in upcoming testing levels.

JUnit

Unit testing framework for java.

JUnit-5 is next generation of JUnit.

The goal is to create an up-to-date foundation for developer-side testing on the JVM. This includes focusing on java-8 and above, as well as enabling many different styles of testing.

Reference: https://junit.org/junit5/

AssertJ

AssetJ is a Java library that provides a rich set assertions and truly helpful error messages.

It improve test code readability, and is designed to be super easy to use within your favorite IDE.

```
import static org.assertj.core.api.AssertionsForClassTypes. assertThat;
```

Reference: https://assertj.github.io/doc/

Mockito

Mockito is a mocking framework for java.

It lets you write tests with clean and simple API.

The tests are very readable and they provide clean verification errors.

Mockito 3.x requires Java 8 or above.

H2 Database

H2 database is a java SQL database.

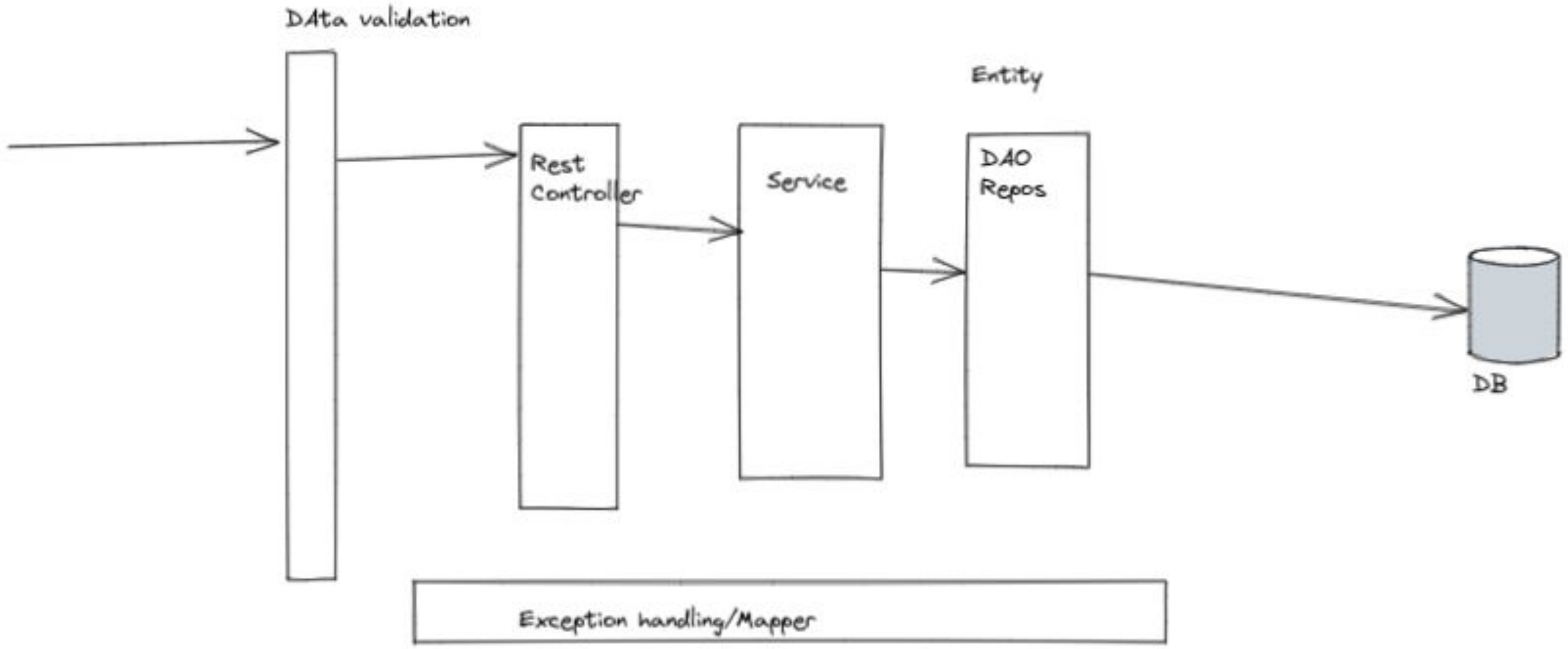
Very fast and open source.

Used as in-memory database.

Spring-Boot can auto-configure embedded H2 databases.

```
<dependency>
        <groupId>com.h2database</groupId>
        <artifactId>h2</artifactId>
        <scope>test</scope>
</dependency>
```

API Request flow (Components or Units)



Integration Testing

An integration test can be any of the following:

- a test that covers multiple “units”. It tests the interaction between two or more clusters of cohesive classes.
- a test that covers multiple layers.
- a test that covers the whole path through the application. In these tests, we send a request to the application and check that it responds correctly and has changed the database state according to our expectations.

@SpringBootTest

Spring Boot provides the `@SpringBootTest` annotation which we can use to create an application context containing all the objects we need for all of the above test types.

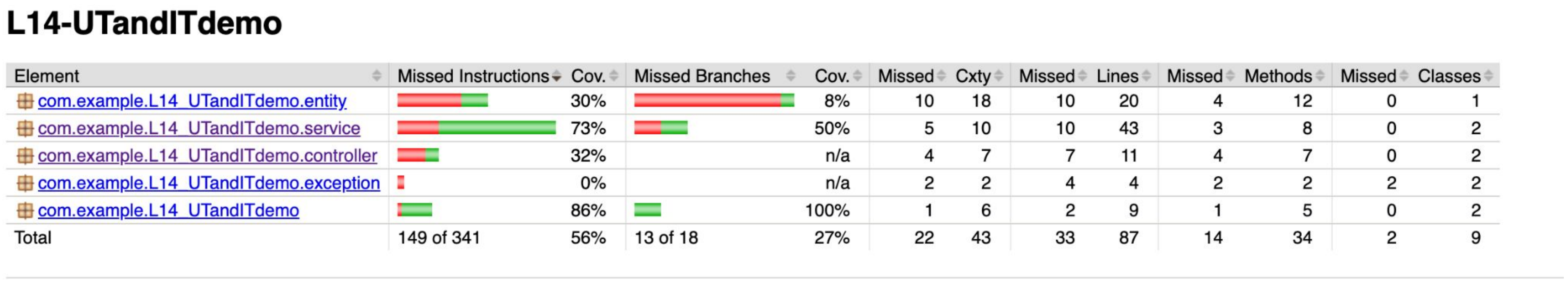
Because we have a full application context, including web controllers, Spring Data repositories, and data sources, `@SpringBootTest` is very convenient for integration tests that go through all layers of the application

Code Coverage

In software development, Code Coverage is a measure used to describe the degree to which the source code of an application is executed when a test suite is executed. A report is generated to view and analyze the code coverage of a software application. This code coverage report could then be used for ensuring code quality.

For optimal code testing, many companies use the **JaCoCo-Maven** plugin that helps generate detailed code coverage reports. JaCoCo-Maven plugin is a free code coverage library for Java projects.

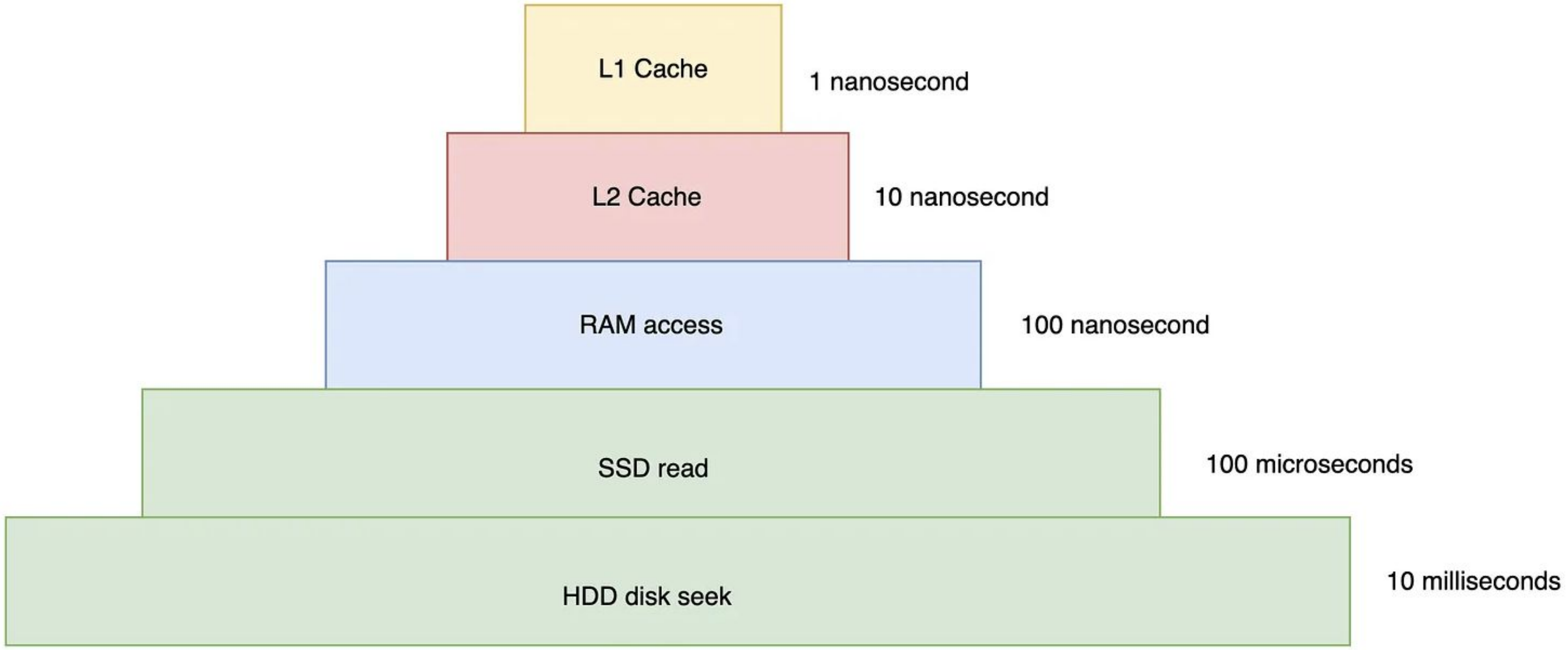
JaCoCo Code Coverage



L14-UTandITdemo

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
com.example.L14_UTandITdemo.entity		30%		8%	10	18	10	20	4	12	0	1
com.example.L14_UTandITdemo.service		73%		50%	5	10	10	43	3	8	0	2
com.example.L14_UTandITdemo.controller		32%		n/a	4	7	7	11	4	7	0	2
com.example.L14_UTandITdemo.exception		0%		n/a	2	2	4	4	2	2	2	2
com.example.L14_UTandITdemo		86%		100%	1	6	2	9	1	5	0	2
Total	149 of 341	56%	13 of 18	27%	22	43	33	87	14	34	2	9

Computer Memory Hierarchy



Latency to read the data

Why Cache ?

Reduce API Latency.

Save network calls.

Avoid computation.

Reduce DB load.

Cache Limitations

RAM is expensive. We can not store all data is cache.

Data Consistency.

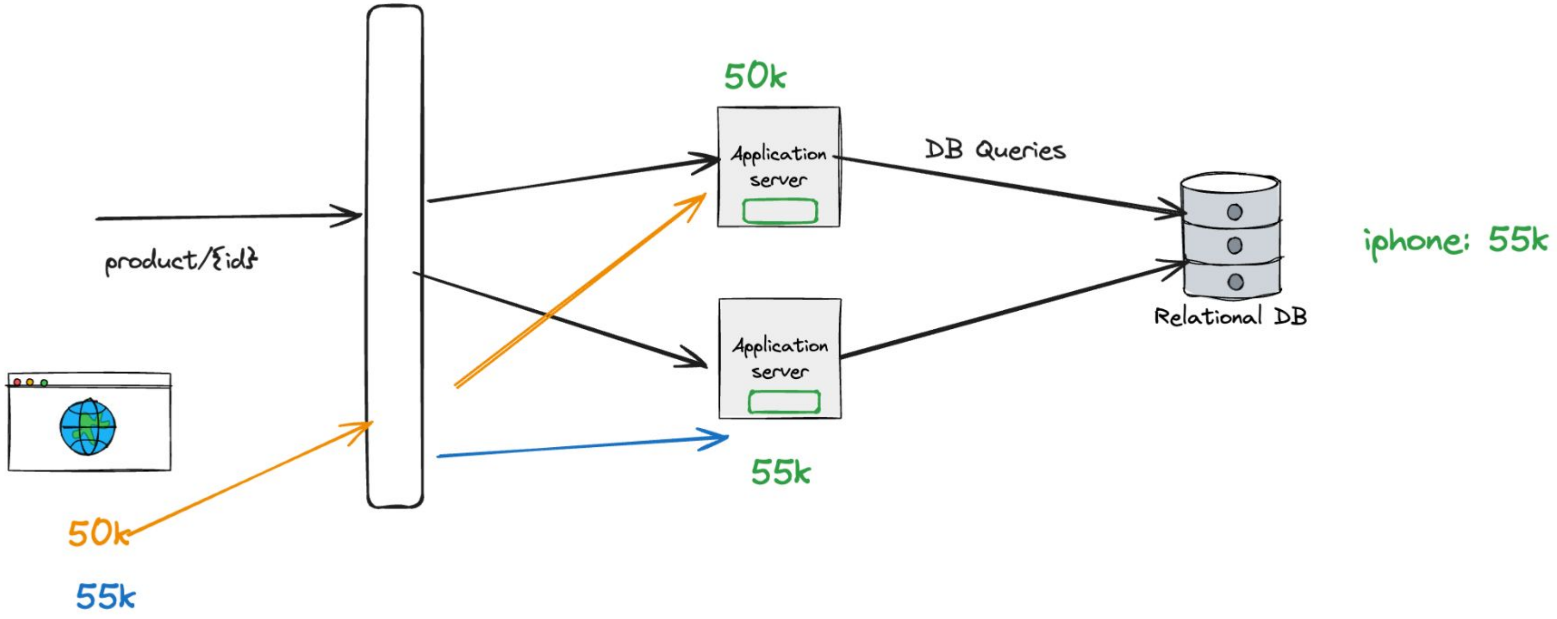
Cache Miss.

Cache Eviction policy. LRU, LFU

Local Cache

- Resides in application server only.
- JVM cache (HashMap)
- In case of of multiple instance each instance will have its own cache.
- Fast
- Limited size.
- Consistency Issue.

Local Cache Consistency Issue



Global Cache (Distributed Cache)

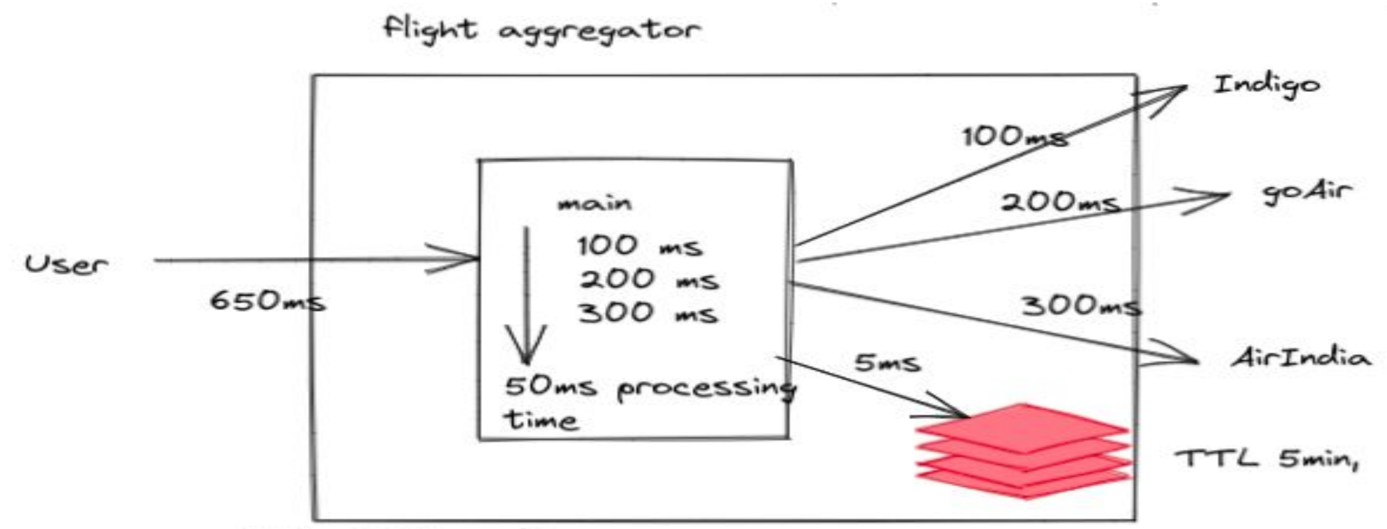
Runs on separate server or servers.

All application instance uses this cache only.

Data is more consistent as compare to local cache.

Example: Redis, Memcache, Aerospike

Optimizing Fight Aggregator



Using 3 threads:
In 300ms all data will available.
API Response time(With multithreading): 350ms

With Redis: 55ms (Cache hit)

360ms (Cache Miss)