

Kresil

Kotlin Resilience

Kotlin Multiplatform library for fault-tolerance
with Ktor Integration

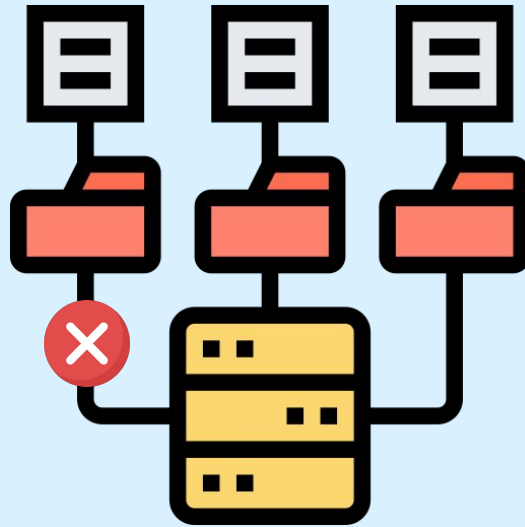


ISEL
INSTITUTO SUPERIOR DE
ENGENHARIA DE LISBOA

Supervisor: Prof. Pedro Félix
Author: Francisco Engenheiro - 49428
Project and Seminary
BSc in Computer Science and Engineering
Summer 2023/2024

Resilience in Distributed Systems

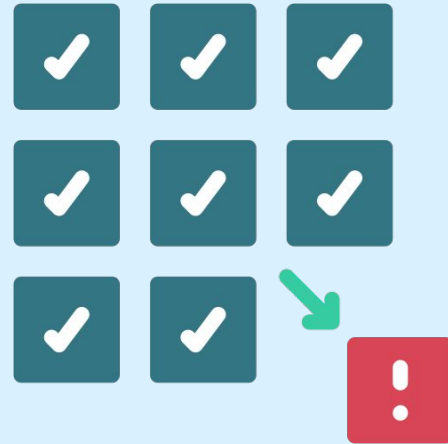
If failure is inevitable, what can be done?



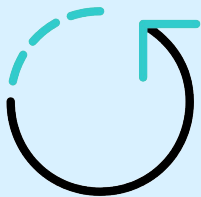
Fault-Tolerance Service

In the presence of faults:

- **Maintains all or part of its functionality**
- **Provides an alternative**



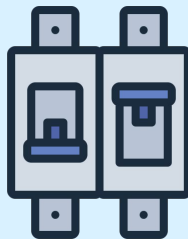
Resilience Mechanisms



Retry - Repeats failed executions



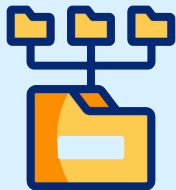
Rate Limiter - Limits executions/period



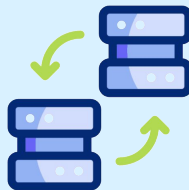
Circuit Breaker - Temporarily blocks possible failures



Time Limiter - Limits duration of execution



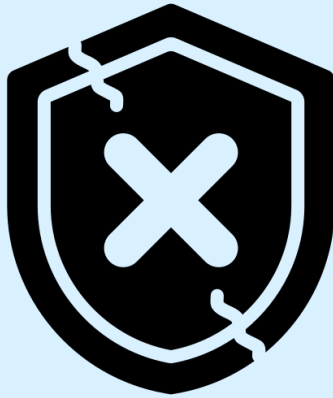
Cache - Memorizes a successful result



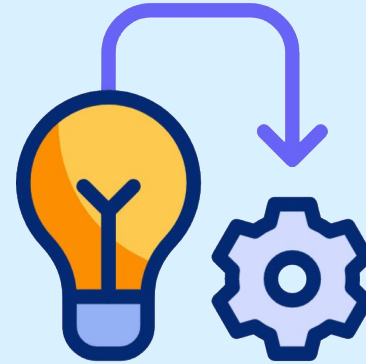
Fallback - Defines an action to fallback on failure

And many more...

Resilience Types

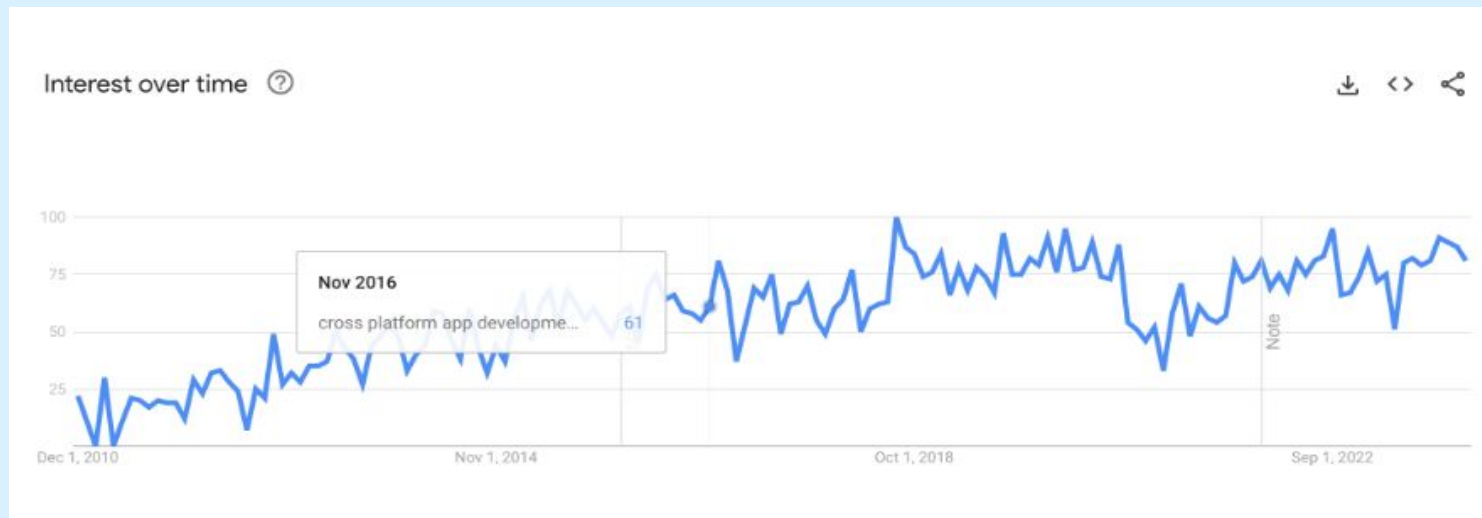


Reactive - Mitigates
impact from failures



Proactive - Prevents
failures from happening

Multipatform Development Interest

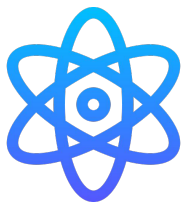


Font:

<https://www.jetbrains.com/help/kotlin-multiplatform-dev/cross-platform-frameworks.html>

Problem

Multiplatform Technologies



React Native



Flutter

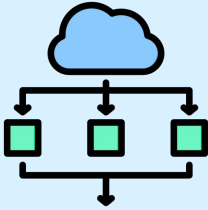


**Kotlin
Multiplatform**

**No libraries that provide
resilience mechanisms in a
multiplatform context**



Multiplatform Considerations



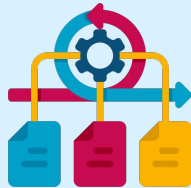
**Concurrency
Model**



**Time
Management**



**Synchronous
vs
Asynchronous**

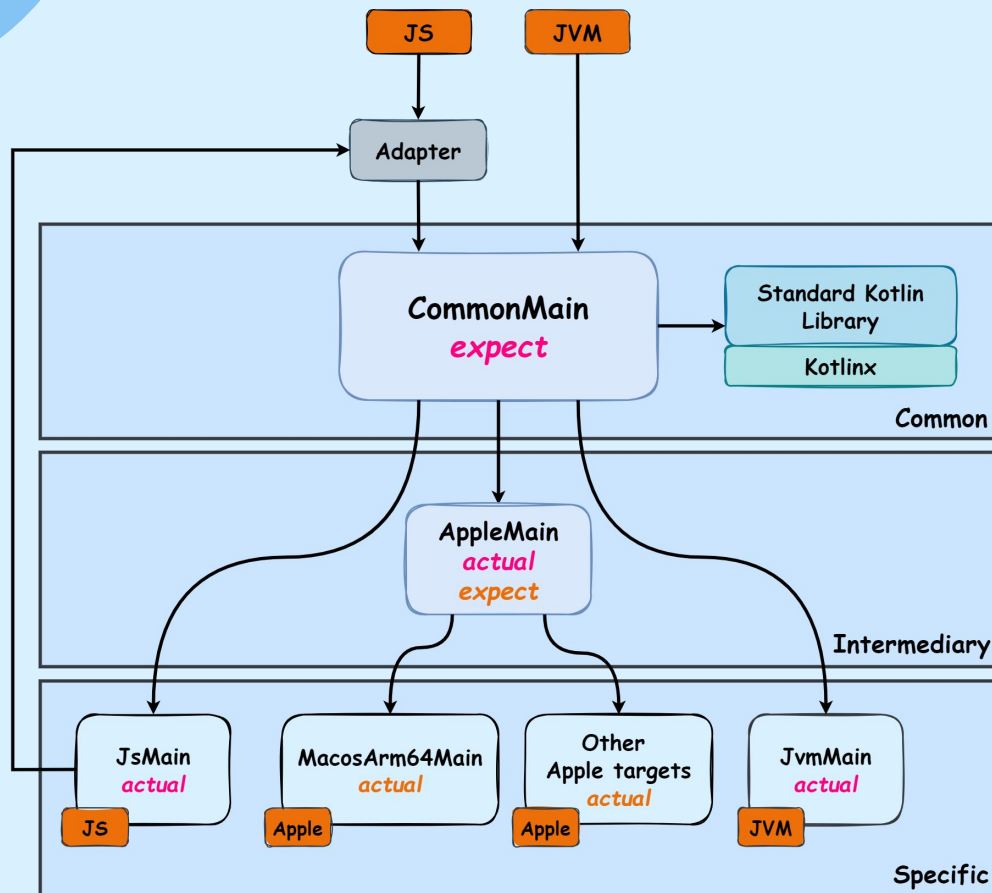


Mocking



Logging

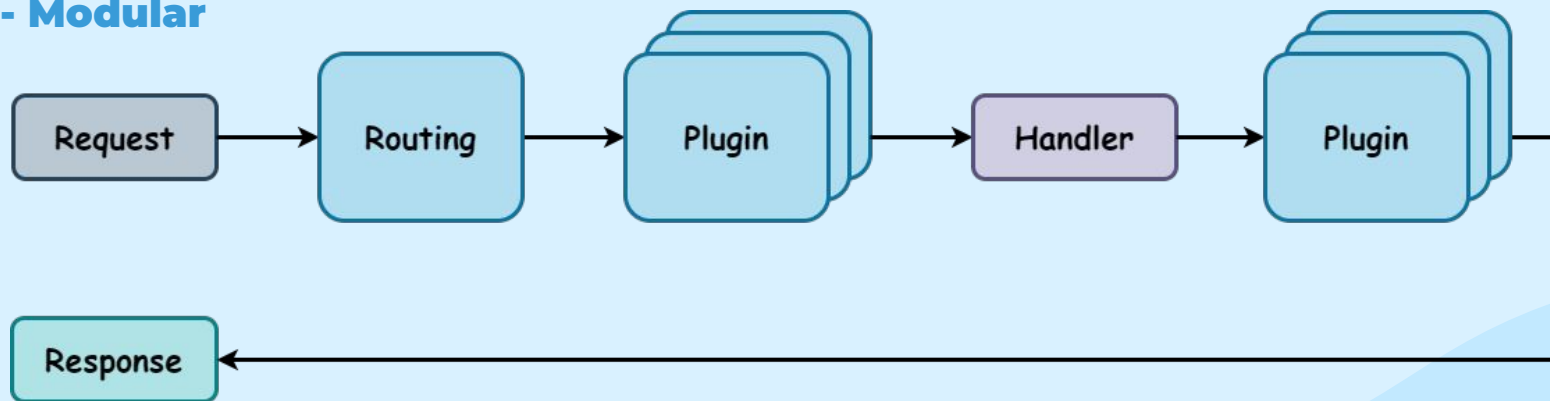
Kotlin Multiplatform Architecture





Ktor Framework

- Built with Kotlin Multiplatform;
- Only known asynchronous server and client development framework
- Based on the coroutines concurrency model;
- Modular



Existing Solutions

Platform-Specific



Resilience4j



Netflix's Hystrix



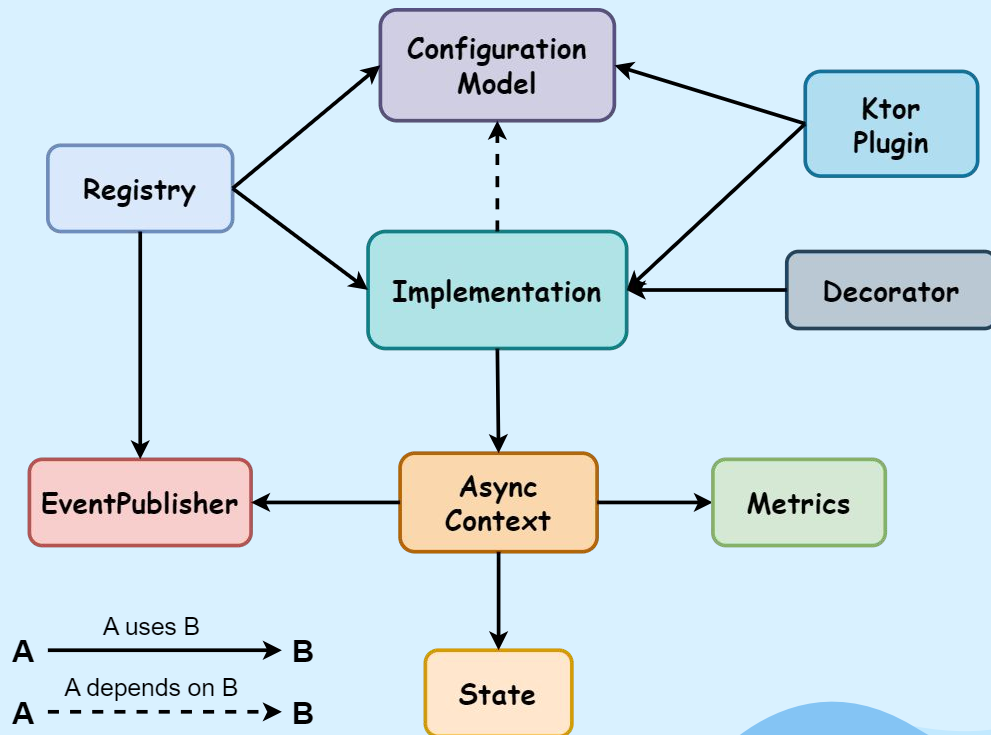
Multiplatform



Arrow



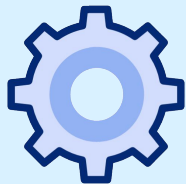
Mechanism Model



Mechanism Configuration

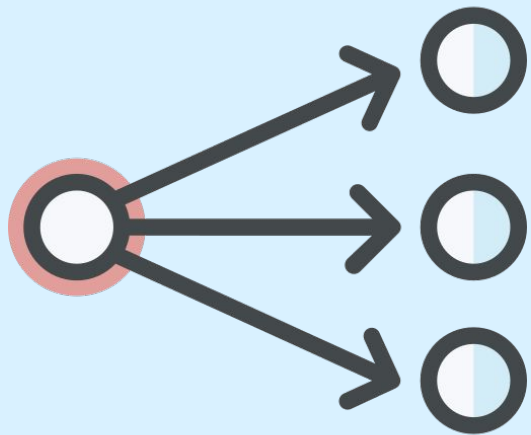
```
internal fun <TBuilder: ConfigBuilder<TConfig>, TConfig> mechanismConfigBuilder(  
    builder: TBuilder,  
    configure: TBuilder.() -> Unit  
) : TConfig = builder.apply(configure).build()
```

Policies - Define
the mechanism
behaviour



```
val config: RetryConfig = retryConfig {  
    maxAttempts = 3  
    retryIf { it is WebServiceException }  
    constantDelay(3.seconds)  
}  
val retry = Retry(config)  
// or: val retry = Retry() // uses default config
```

Event Listeners

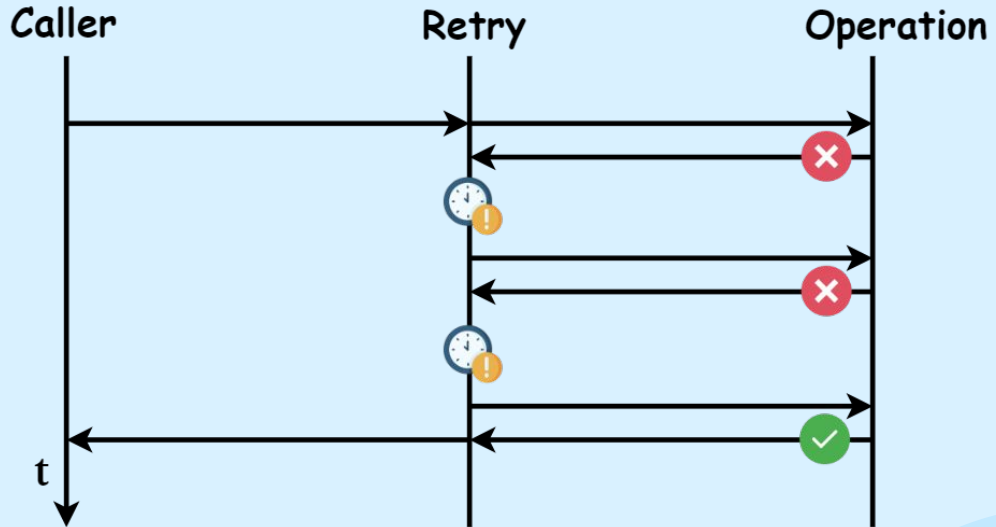


Each mechanism:

- Provides listeners for **specific** and **undiscriminated** events
- Uses asynchronous coroutine primitive **Flow**
- Supports **cancellation** of registered listeners

Retry Mechanism

- **Reactive mechanism**
- **Retries operation on failure**
- **Addresses transient faults**



Delay Handling

Delay Strategy



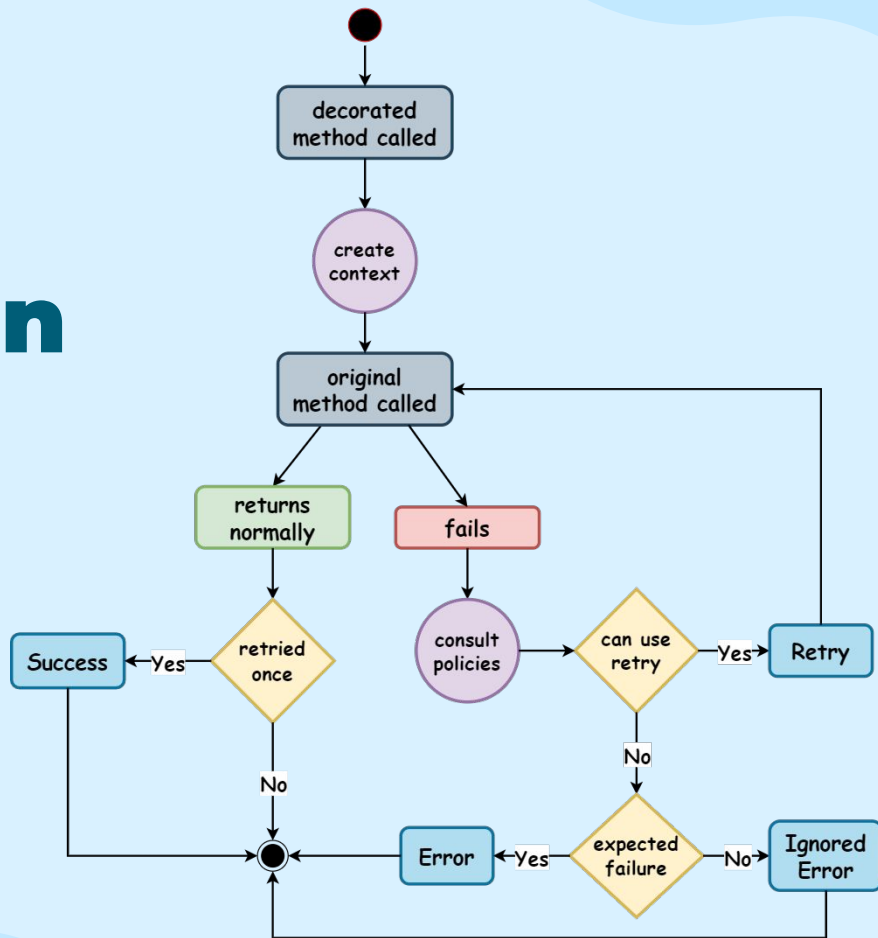
Options

- **No delay**
- **Constant delay**
- **Linear delay**
- **Exponential delay**
- **Custom Delay**

Delay Provider



Retry Execution Flow

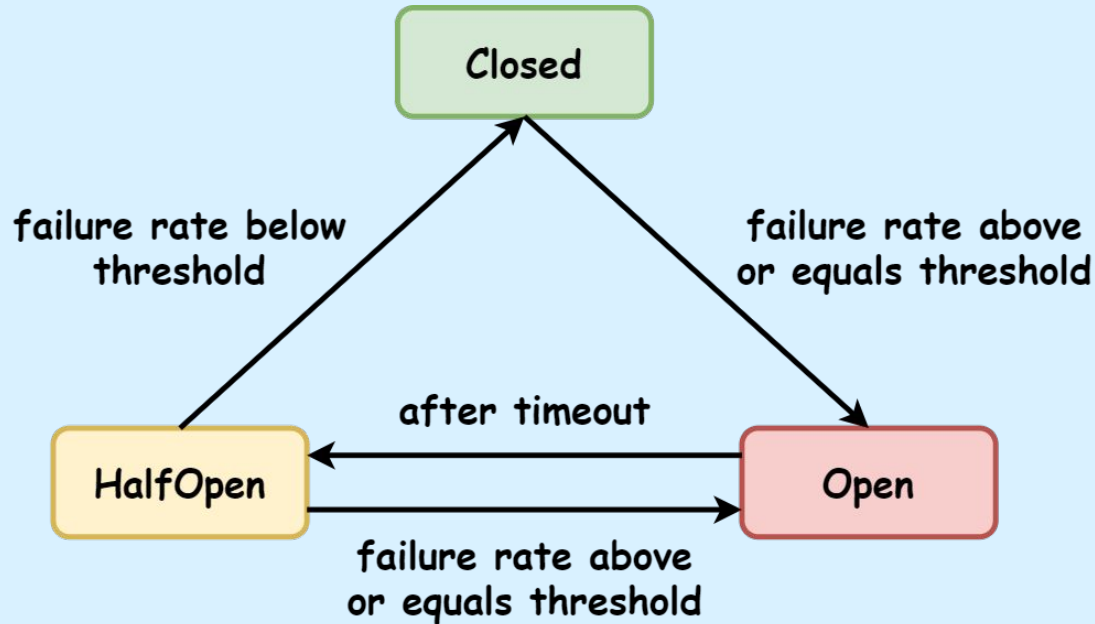


Circuit Breaker Mechanism

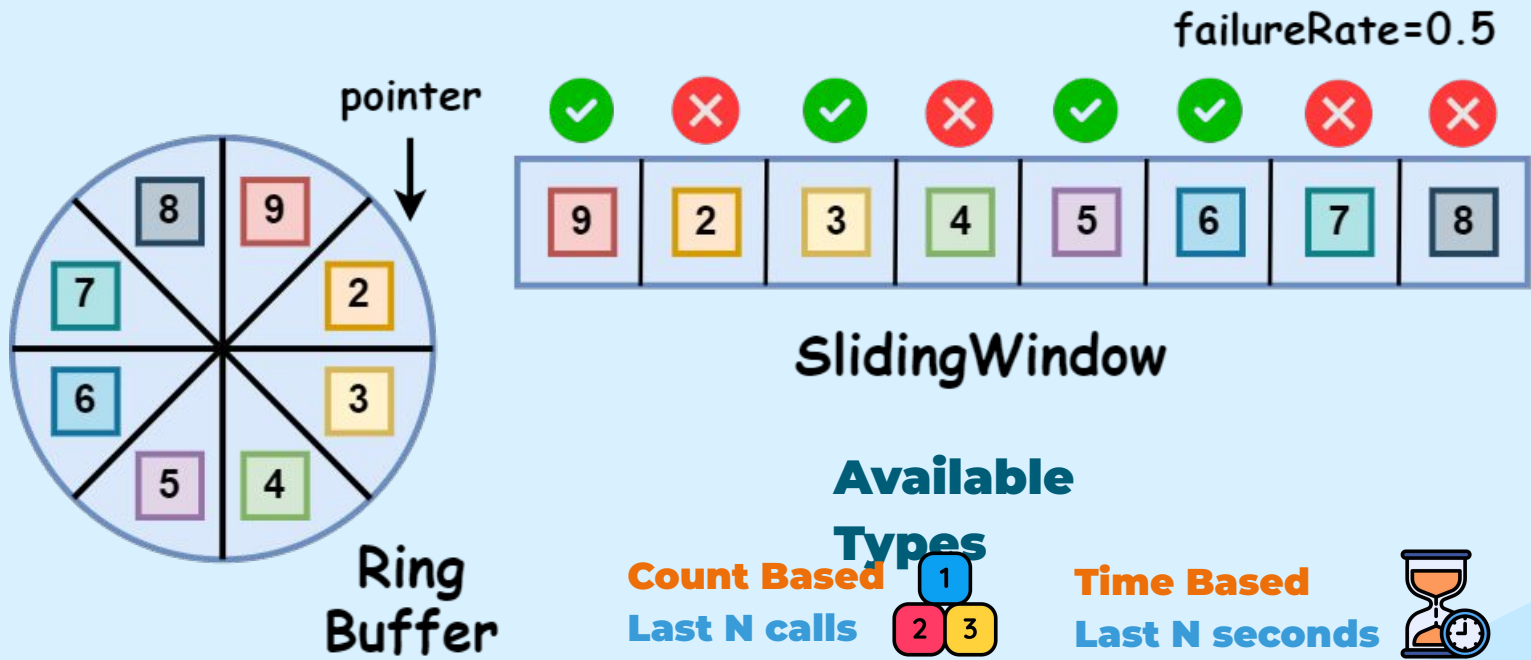
- **Reactive** resilience mechanism
- **Protects** component from **overload/failure**
- **Short-circuits** requests **when** **component misbehaves**
- **Monitors** system health



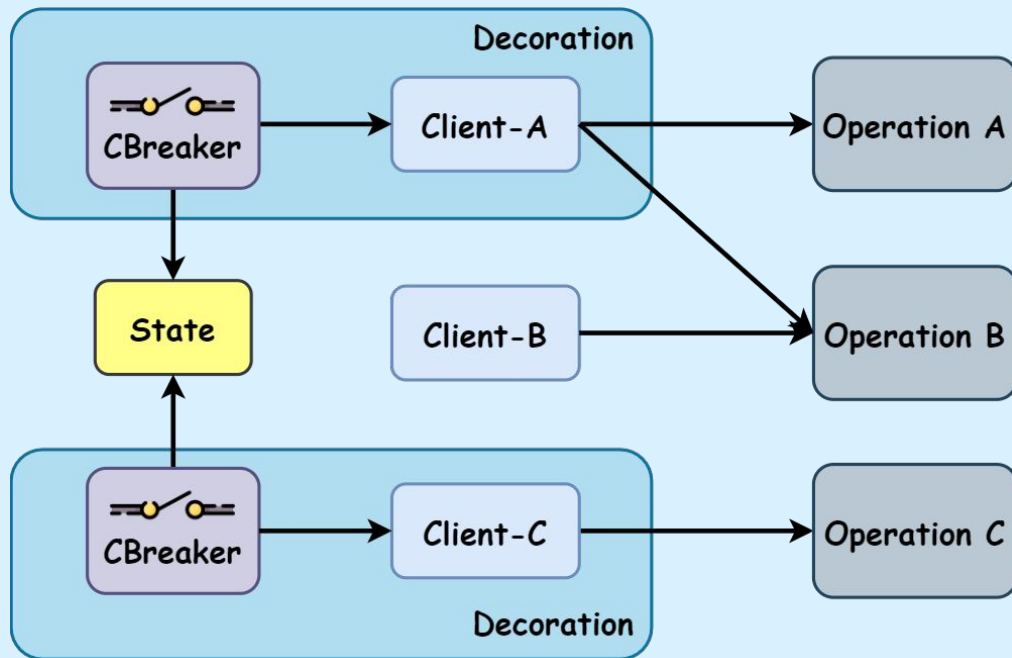
Circuit Breaker States



Circuit Breaker Sliding Window

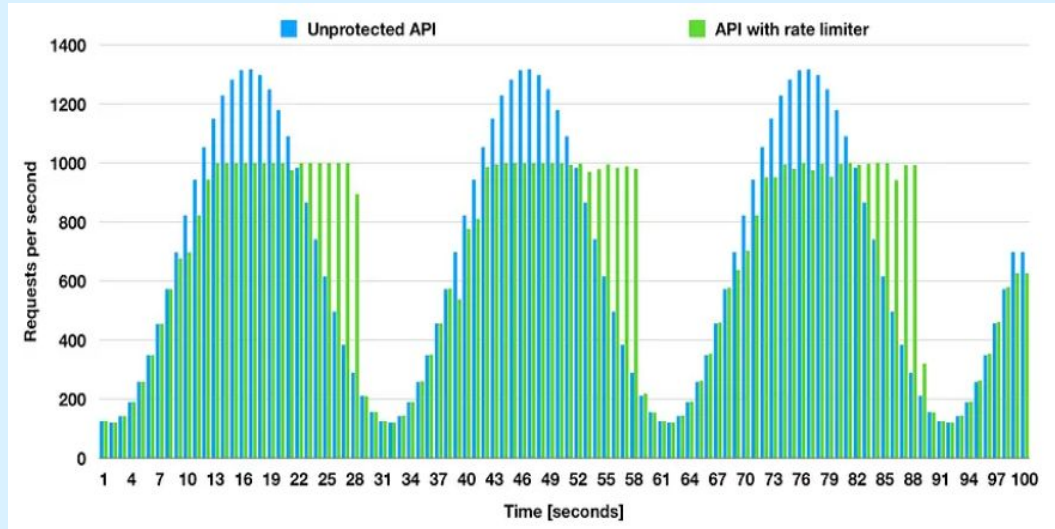


Circuit Breaker Decoration

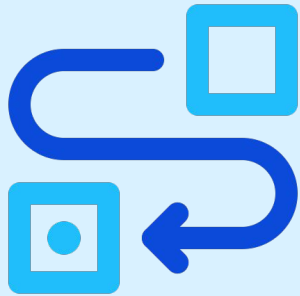


Rate Limiter Mechanism

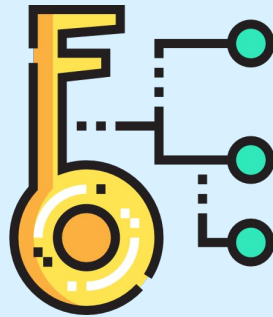
- **Proactive** mechanism
- **Limits requests to a component**
- **Could be bound to a time unit**
- **Protects systems from overloading**



Types of Rate Limiting



Total Requests



Key-based

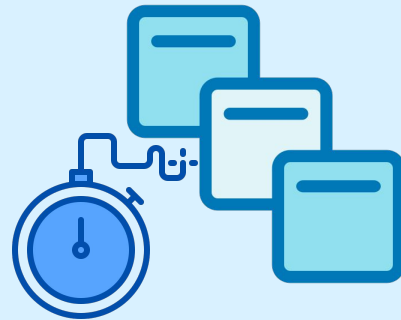
Rate Limiting Exceeded



Reject: Immediately deny the request and return an error response message

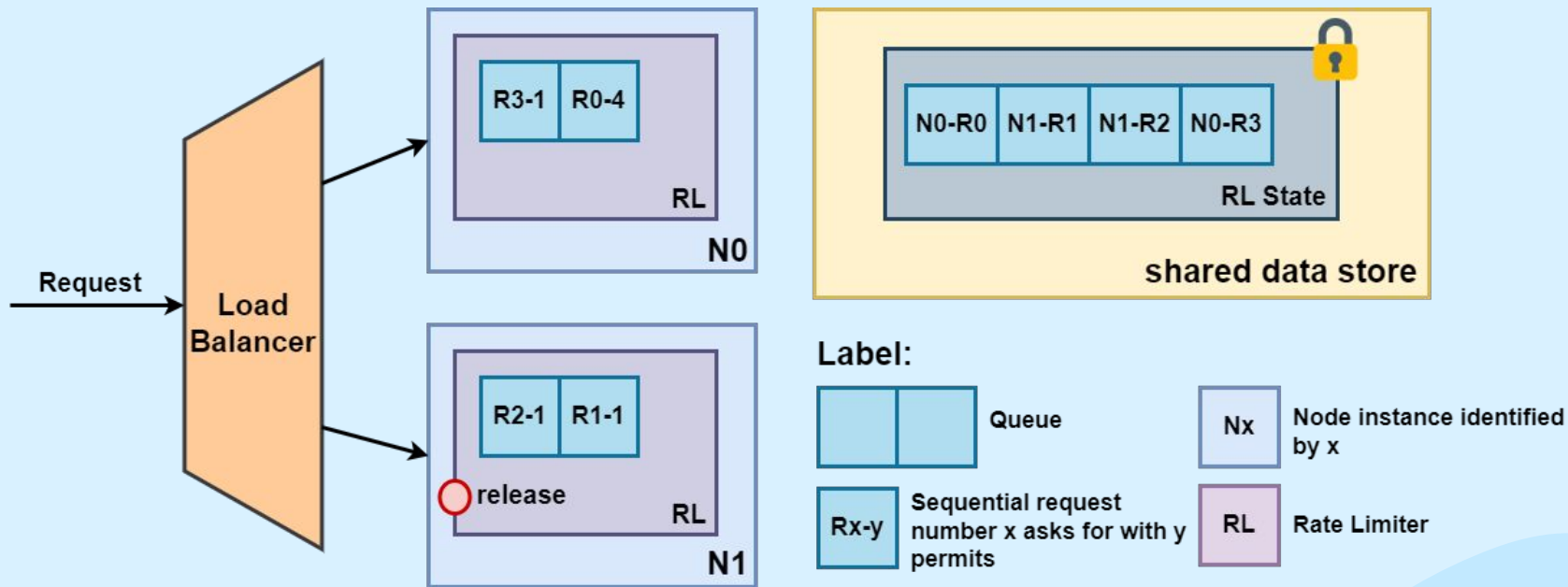


Wait: Place the request in a queue to be processed later when the rate limit allows

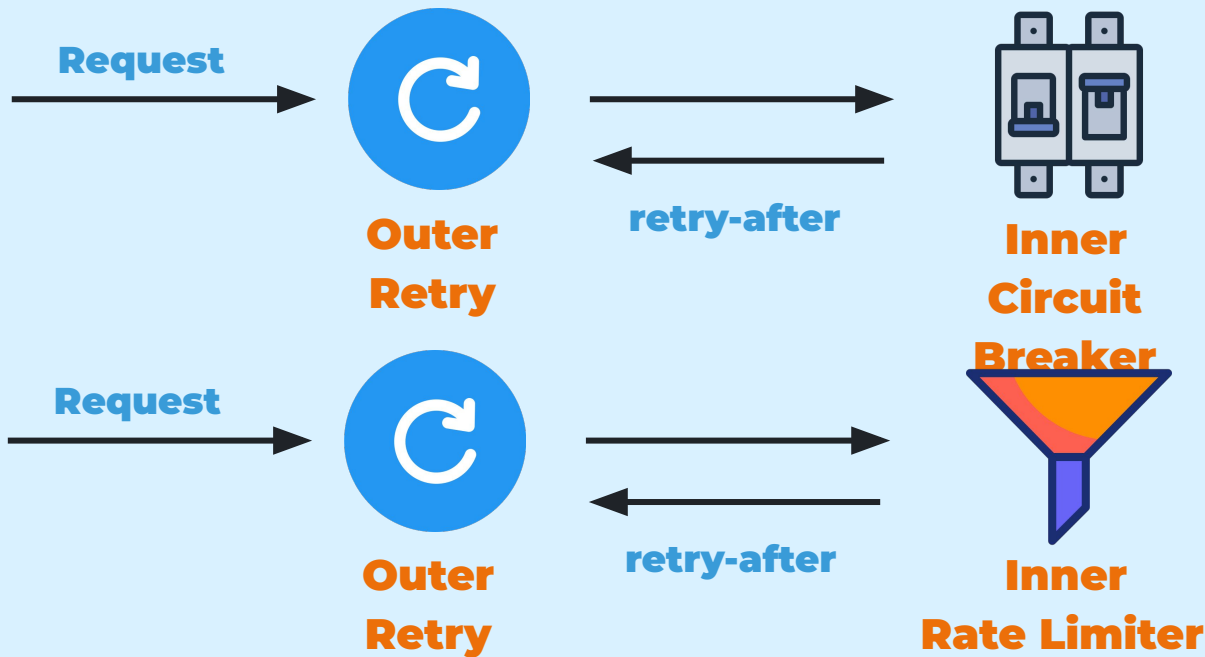


Both: Place request in the queue and reject after timeout expires

Distributed Rate Limiting



Mechanism Combination



API Documentation

The screenshot shows the API documentation for the `CircuitBreaker` class in the `kresil-lib / kresil.circuitbreaker` package. The left sidebar lists the package structure, with `CircuitBreaker` selected. The main content area displays the class name `CircuitBreaker` and its Scala code signature:

```
class CircuitBreaker(  
  val config: CircuitBreakerConfig = defaultCircuitBreakerConfig()  
): FlowEventListenerImpl<CircuitBreakerEvent>
```

A "(source)" link is provided for the code. Below the code, a descriptive paragraph explains the `Circuit Breaker` as a **reactive** resilience mechanism. It details how it monitors system health, short-circuits requests upon detecting failures, allows test requests after a timeout, and resumes normal operation based on test results. It also mentions that the circuit breaker is initialized with a configuration that defines its behavior through pre-configured policies.

Below the text, a state machine diagram illustrates the transitions between the `Closed` and `Open` states. The diagram shows that the `Closed` state transitions to the `Open` state when the "failure rate exceeds or equals threshold". From the `Open` state, a transition labeled "after timeout" returns the circuit breaker to the `Closed` state. The `Closed` state also has a self-loop labeled "failure rate exceeds or equals threshold".

Link: <https://kresil.github.io/kresil/>

Plugin Demos Architecture



Server



Client

Retry Plugin Demo



**Client
With Retry**



**Client
Without
Retry**

Overview:

- Server allows for adjustment of error rate
- Error rate affects request failure probability
- Clients display request error rate

Objective: Demonstrate improved success rate to a unreliable server



Server

Retry Plugin Demo

The screenshot shows a web browser window with the title "Kresil Ktor Retry Plugin" and the address bar showing "localhost:8081". The page has a light blue header with a logo and the title. Below the header, there are two main sections: "Client with Retry" and "Client without Retry". In the center, there is a control panel with a text input for "Enter error rate (0.0 - 1.0)", a "Set Error Rate" button, and a status indicator showing "Current Server Error Rate: 0.0".

Kresil Ktor Retry Plugin

Client with Retry

- Total Requests: 6
- Total Errors: 0
- Client Error Rate: 0

Enter error rate (0.0 - 1.0)

Set Error Rate

Current Server Error Rate: 0.0

Client without Retry

- Total Requests: 6
- Total Errors: 0
- Client Error Rate: 0

Circuit Breaker Plugin Demo



Client

Overview:

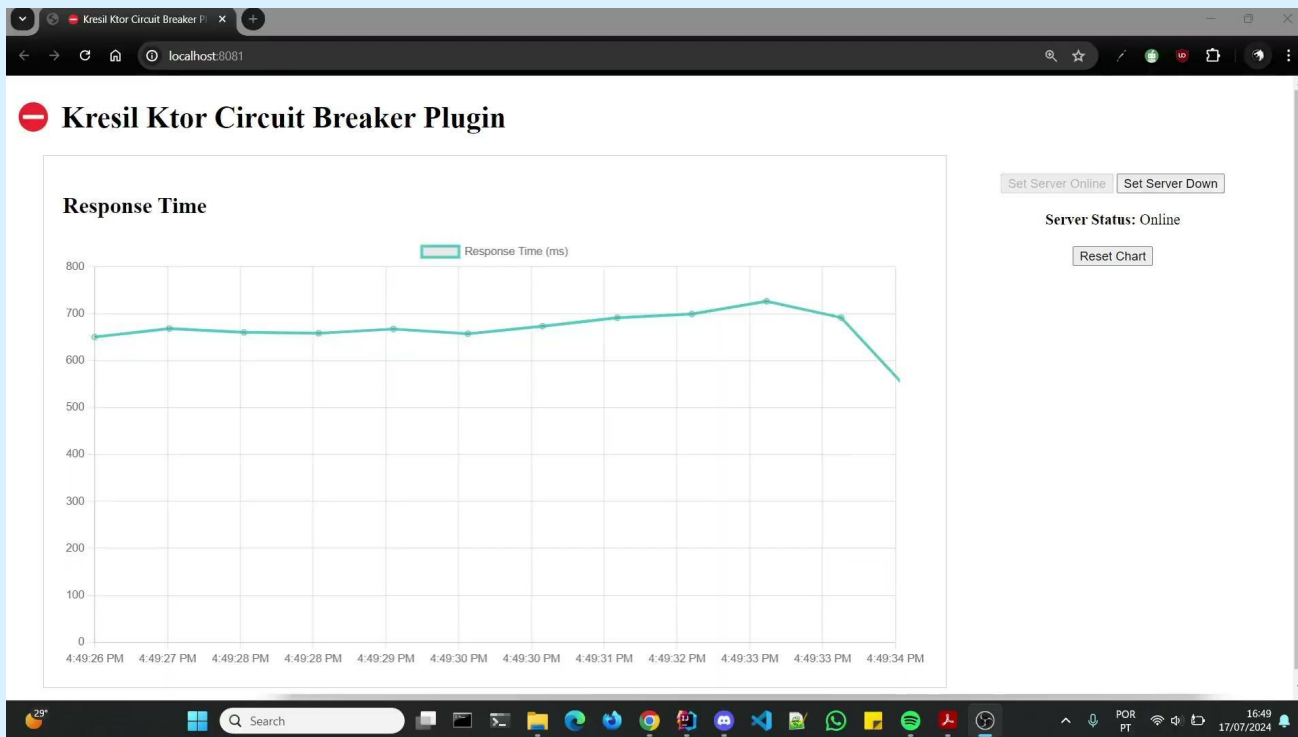
- Server allows configuration of response behavior (OK/NOK)
- Client monitors response time from server



Server

Objective: Minimize request response time to a considered failing component (fail-fast)

Circuit Breaker Plugin Demo



Rate Limiter Plugin Demo



Client

Overview:

- Server has rate-limited and unlimited routes
- Client plots requests made to both routes over a specific time unit

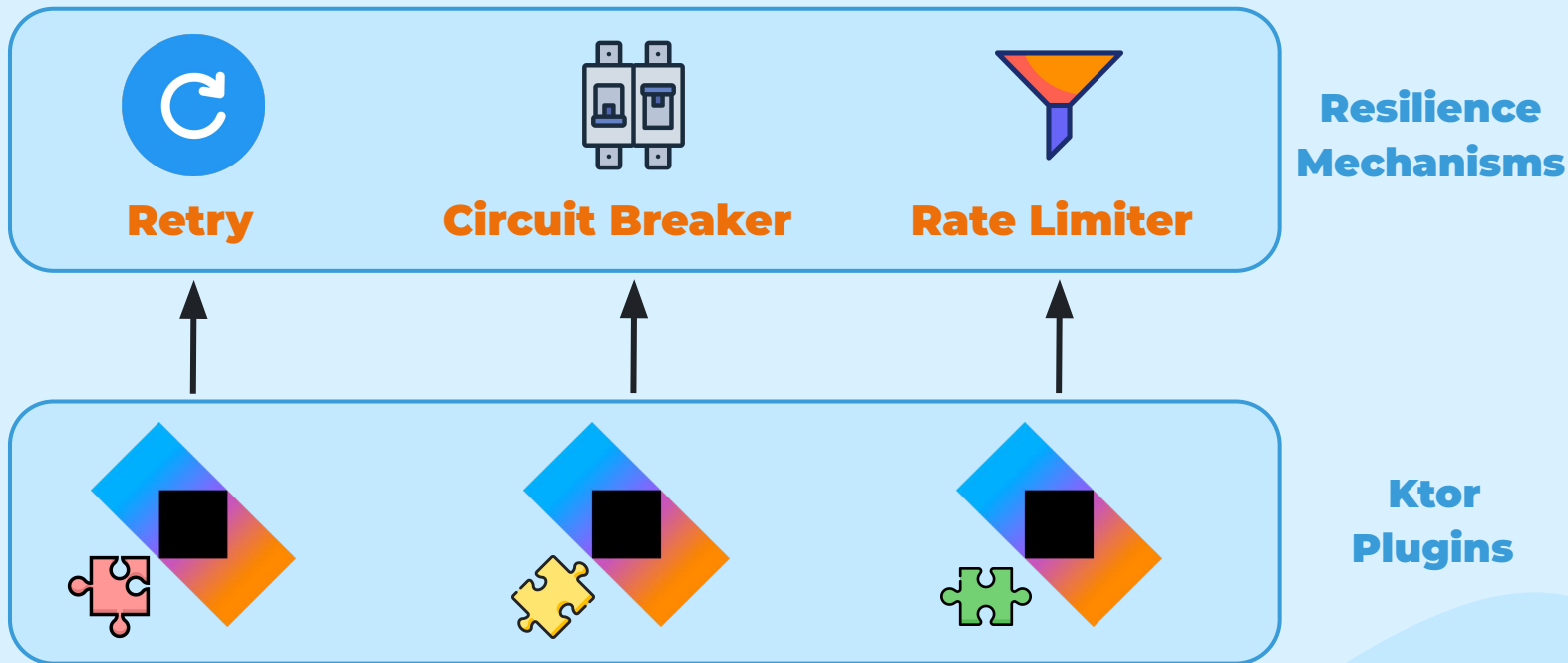
Objective: Observe rate-limited requests do not exceed a predefined limit; while unrated do



Server

Rate Limiter Plugin Demo

Conclusions



Software Engineering Practices

The screenshot displays a Kanban board for a project named 'Main Project'. The board is organized into four columns: Planning, Todo, In Progress, and In Review. Each column contains tasks with descriptions, assignees, and status tags.

Planning (14)
Tasks in active brainstorming. Subject to change or be deleted

- kresil #65**: Define semantic commit messages to use (workflow)
- Draft**: Implement thread-safe Metrics in Mechanisms (after Circuit Breaker and Rate Limiter implementations and plugins)
- Draft**: Publish a release to Maven Central

Todo (2)
Tasks that haven't been started

- kresil #60**: Rate Limiter: provide a demo using a shared data store (demonstration)
- Draft**: Rate Limiter: update documentation

In Progress (6)
Tasks actively being worked on

- ps #36**: Develop final presentation slides (demonstration)
- kresil #75**: Circuit Breaker: alter demo behaviour (demonstration)
- kresil #67**: Circuit Breaker: add exception handler (enhancement)

In Review (1)
Completed tasks enter this phase for evaluation

- kresil #52**: Rate Limiter: check Resilience4j implementation and interoperability with Kotlin (research)