Complex Rate-Limiting is easy with Bucket4j

Speaker Maxim Bartkov


maxgalayoutop@gmail.com


@MaximBartkov

About the speaker

- More than 8 years in Java
- Co-Author Bucket4j library
- Co-Author book "Spring Rest Building Java Microservices and Cloud Applications"
- Author of scientific publications
- Write articles for Java community

What this talk is about

What this talk is about

- What rate-limiting is

What this talk is about

- What rate-limiting is
- Why sometimes we should use rate-limiting on application-level

What this talk is about

- What rate-limiting is
- Why sometimes we should use rate-limiting on application-level
- Main areas to use it

What this talk is about

- What rate-limiting is
- Why sometimes we should use rate-limiting on application-level
- Main areas to use it
- Why Bucket4j

Map legend

Bandwidth == Limit

Multi-Bandwidth == Set of limits

Application level == In java code

What rate-limiting is

The main areas to use rate-limiting

- To protect our system from external requests

The main areas to use rate-limiting

- To protect our system from external requests
- To realize contract requirements (using external API as an example)

The main areas to use rate-limiting

Contract requirements

	BASIC	BUSINESS	PREMIUM
/API/FRIENDS			
/API/USERS/{ID}			
/API/POSTS			

The main areas to use rate-limiting

Contract requirements

	BASIC	BUSINESS	PREMIUM
/API/FRIENDS	1000/hour 10000/day		
/API/USERS/{ID}	120/hour 1200/day		
/API/POSTS	300/hour 2000/day		

The main areas to use rate-limiting

Contract requirements

	BASIC	BUSINESS	PREMIUM
/API/FRIENDS	1000/hour 10000/day	2000/hour 20000/day	
/API/USERS/{ID}	120/hour 1200/day	240/hour 2400/day	
/API/POSTS	300/hour 2000/day	600/hour 4000/day	

The main areas to use rate-limiting

Contract requirements

	BASIC	BUSINESS	PREMIUM
/API/FRIENDS	1000/hour 10000/day	2000/hour 20000/day	4000/hour 40000/day
/API/USERS/{ID}	120/hour 1200/day	240/hour 2400/day	720/hour 7200/day
/API/POSTS	300/hour 2000/day	600/hour 4000/day	1800/hour 12000/day

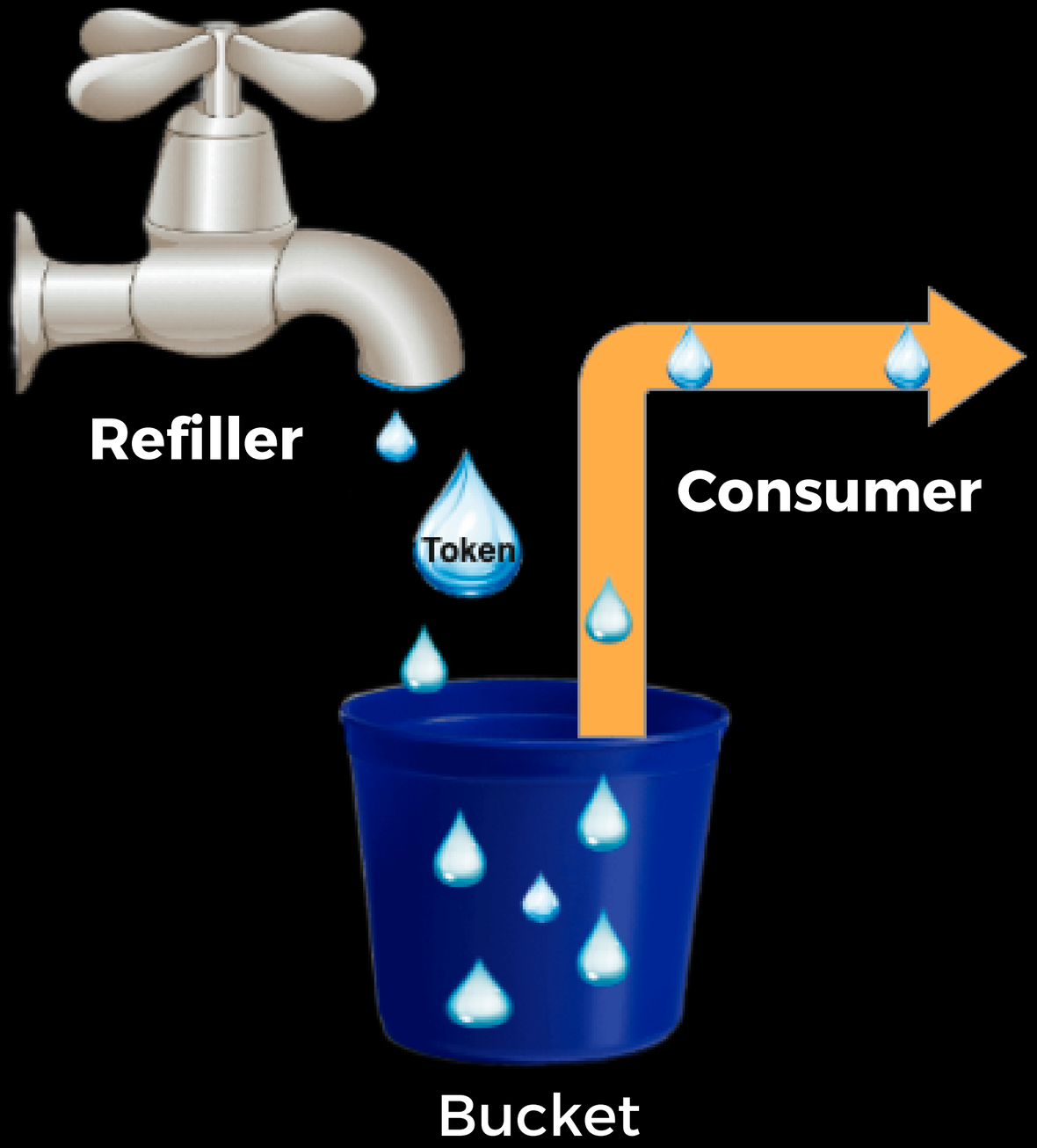
The main areas to use rate-limiting

- To protect our system from external requests
- To realize contract requirements (using external API as an example)
- To recognize fraud/anomaly detection

The main areas to use rate-limiting

- To protect our system from external requests
- To realize contract requirements (using external API as an example)
- To recognize fraud/anomaly detection
- To protect an external system from us

The Token Bucket algorithm



The Token Bucket - code example

```
private final long capacity;
private long availableTokens;

private final long nanosToGenerationToken;
private long lastRefillNanotime;

public TokenBucketExample(long permits, Duration period) {
    this.nanosToGenerationToken = period.toNanos() / permits;
    this.lastRefillNanotime = System.nanoTime();
    this.capacity = permits;
    this.availableTokens = permits;
}
```

The Token Bucket - code example

```
synchronized public boolean tryConsume(int permits) {
    refill();
    if(availableTokens < permits) {
        return false;
    } else {
        availableTokens -= permits;
        return true;
    }
}
```

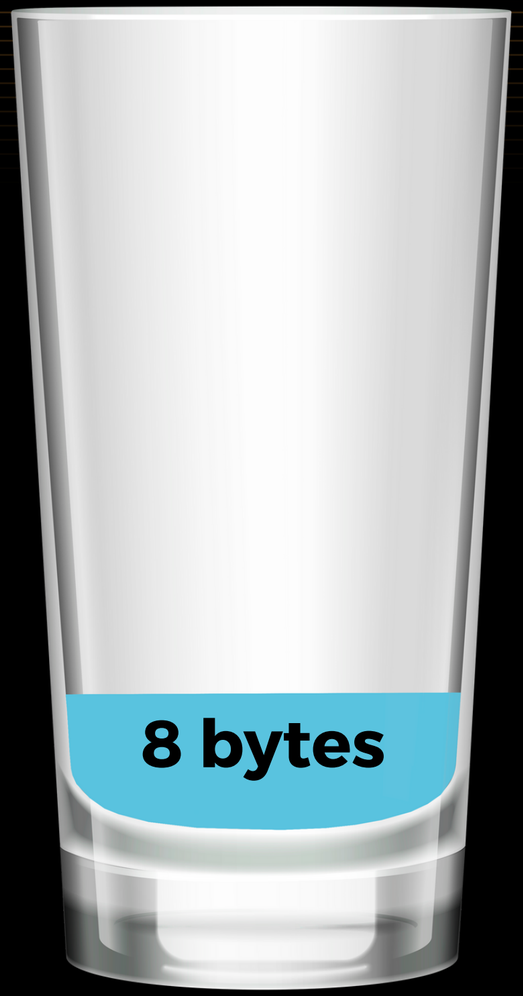
The Token Bucket - code example

```
private void refill() {
    long now = System.nanoTime();
    long nanosSinceLastRefill = now - lastRefillNanotime;
    if(nanosSinceLastRefill <= nanosToGenerationToken) {
        return;
    }
    long tokensSinceLastRefill = nanosSinceLastRefill / nanosToGenerationToken;
    availableTokens = Math.min(capacity, availableTokens + tokensSinceLastRefill);
    lastRefillNanotime += tokensSinceLastRefill * nanosToGenerationToken;
}
```

The main advantage of the Token Bucket algorithm

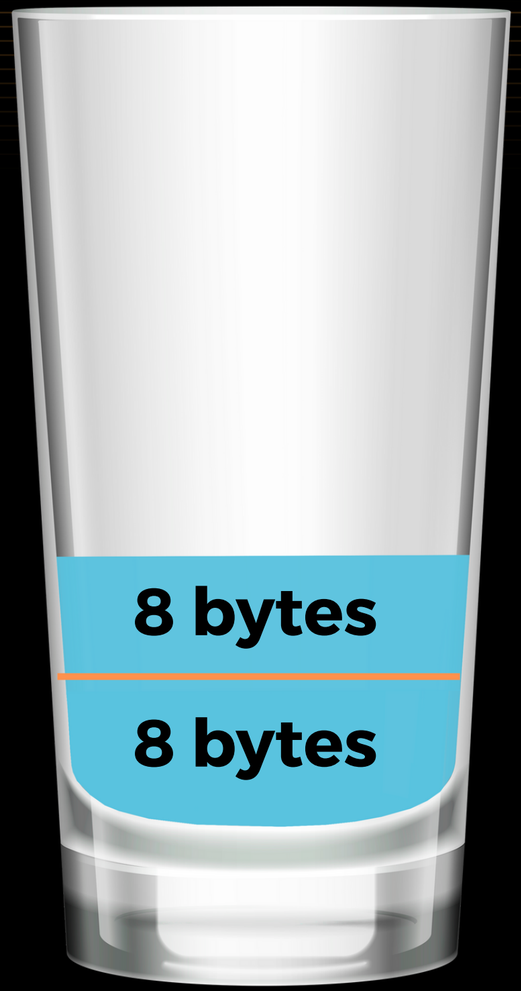
The main advantage

The main advantage



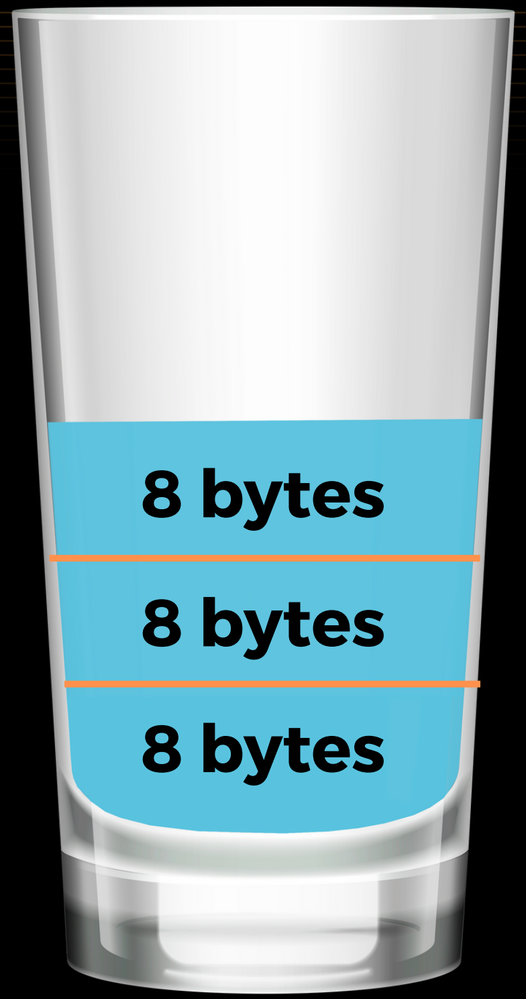
Volume of Bucket (maximum possible count of tokens) - 8 bytes (long)

The main advantage



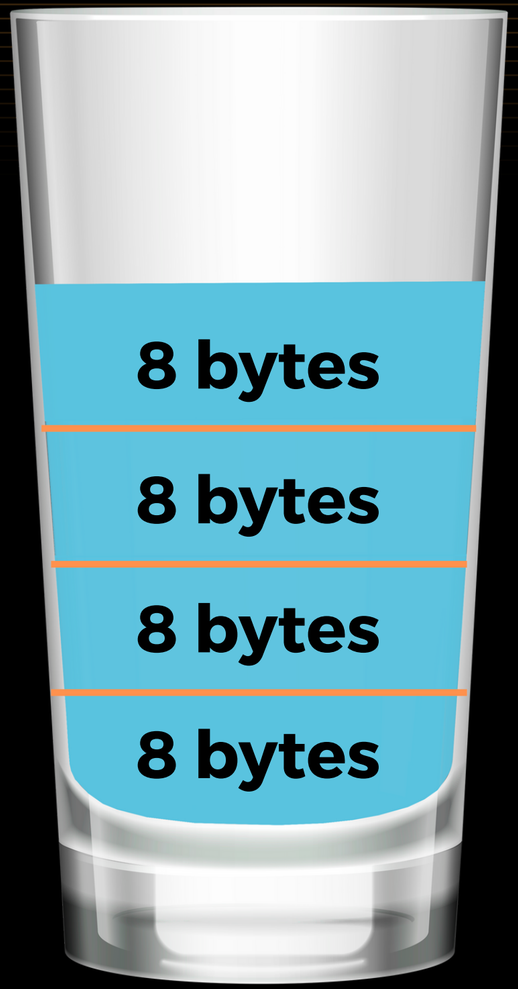
Current count of tokens in a bucket - 8 bytes (long)

The main advantage



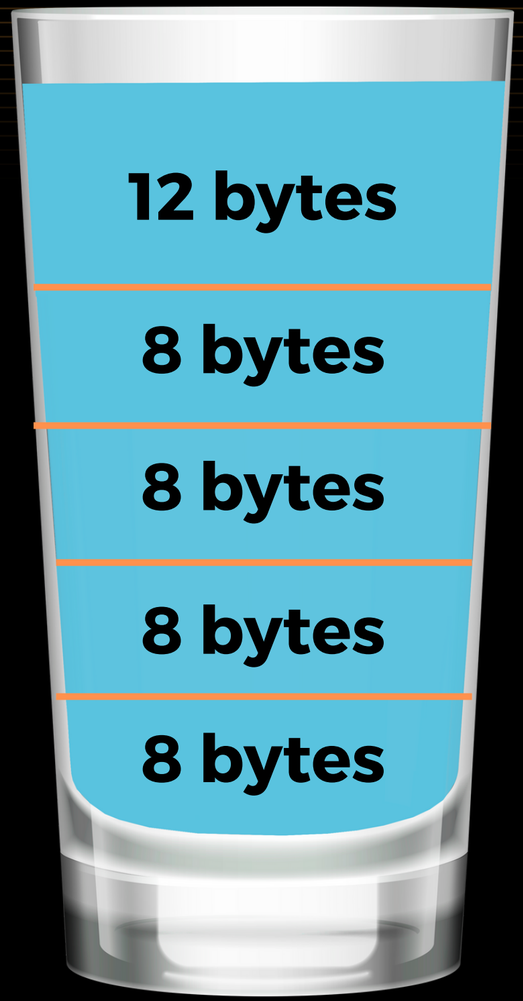
Count of nanoseconds for generating a new token - 8 bytes (long)

The main advantage



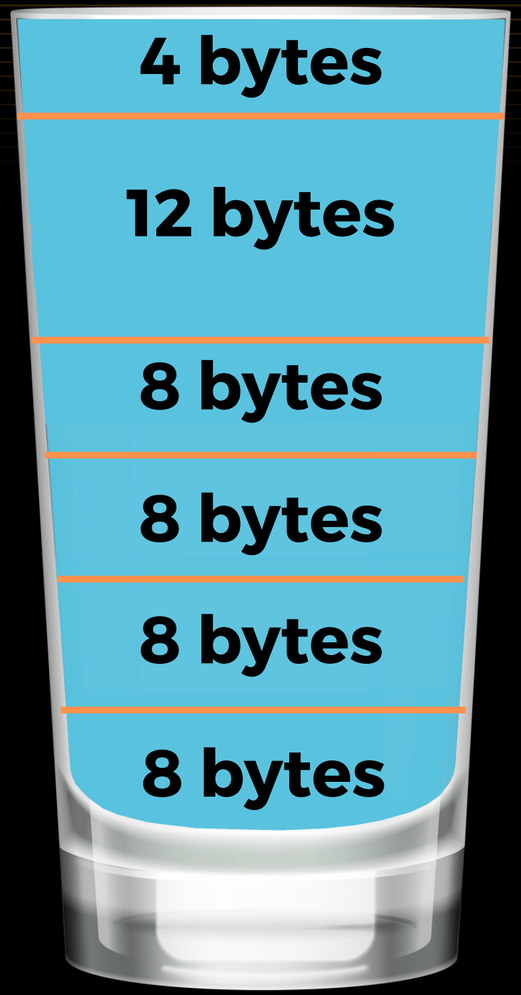
Last time refill in nanoseconds - 8 bytes (long)

The main advantage



Header of object - 12 bytes (default object weight for 64-bit JDK)

The main advantage



Padded to a multiple of 8 - **4 bytes** (an object should have a weighted multiple of 8 bytes in 64-bit JDK)

The main advantage

- Volume of Bucket (maximum possible count of tokens) - 8 bytes (long)
- Current count of tokens in a bucket - 8 bytes (long)
- Count of nanoseconds for generating a new token - 8 bytes (long)
- Last time refill in nanoseconds - 8 bytes (long)
- Header of object - 12 bytes (default object weight for 64-bit JDK)
- Padded to a multiple of 8 - 4 bytes (an object should have a weighted multiple of 8 bytes in 64-bit JDK)

In total: 48 bytes

"Why should we think about memory in Java?"

Rate-Limiting on Application level ?

What ???

Situations to use on application level

Situations to use on application level

- To manage limits from Java

Situations to use on application level

- To manage limits from Java
- To implement the complex limits

Situations to use on application level

- To manage limits from Java
- To implement the complex limits
- Rebalance limits

Situations to use on application level

- To manage limits from Java
- To implement the complex limits
- Rebalance limits
- Monitoring limits

Situations to use on application level

- To manage limits from Java
- To implement the complex limits
- Rebalance limits
- Monitoring limits
- Simple application

Situations to use on application level

- To manage limits from Java
- To implement the complex limits
- Rebalance limits
- Monitoring limits
- Simple application
- For receiving detailed info about limits

About the Bucket4j library

Bucket4j - the most popular library in Java-World for realizing rate-limiting features

Bucket4j

Throttle all the things!

- Every month Bucket4j downloads up to 250,000 times from Maven Central.
- Contained in 3500 dependencies on GitHub.
- Used in Kubernetes Java client, JHipster, Atlassian, Twitch, e.t.c

The Token Bucket - code example

Distributed facilities

- Asynchronous API
- Apache Ignite integration
- Hazelcast integration
- JCache integration
- Infinispan integration
- Oracle Coherence integration
- Redis integration
- DynamoDB integration
- MySQL integration
- PostgreSQL integration
- Framework to implement work with your custom database

Basic functionality (Main Rate-Limiting API)

- Opportunity to work with multi-bandwidth management
- Specifying initial amount of tokens
- Turning-off the refill greediness
- Blocking API
- Scheduler API

Advanced

- Monitoring API
- Diagnostic API
- On-the-fly configuration replacement API
- Modeling time API
- Batching API

How to start work with Bucket4j

```xml
<dependency>
    <groupId>com.github.vladimir-bukhtoyarov</groupId>
    <artifactId>bucket4j-core</artifactId>
    <version>7.4.0</version>
</dependency>
```

Simple Code Example

```
public class SimpleExample {

    public static void main(String args[]) {

        Bandwidth oneConsumePerMinuteLimit = Bandwidth.simple(1, Duration.ofMinutes(1));

        Bucket bucket = Bucket.builder()
                .addLimit(oneConsumePerMinuteLimit)
                .build();

        bucket.tryConsume(1); //true
        bucket.tryConsume(1); //false

    }
```

Multi-Bandwidth Code Example

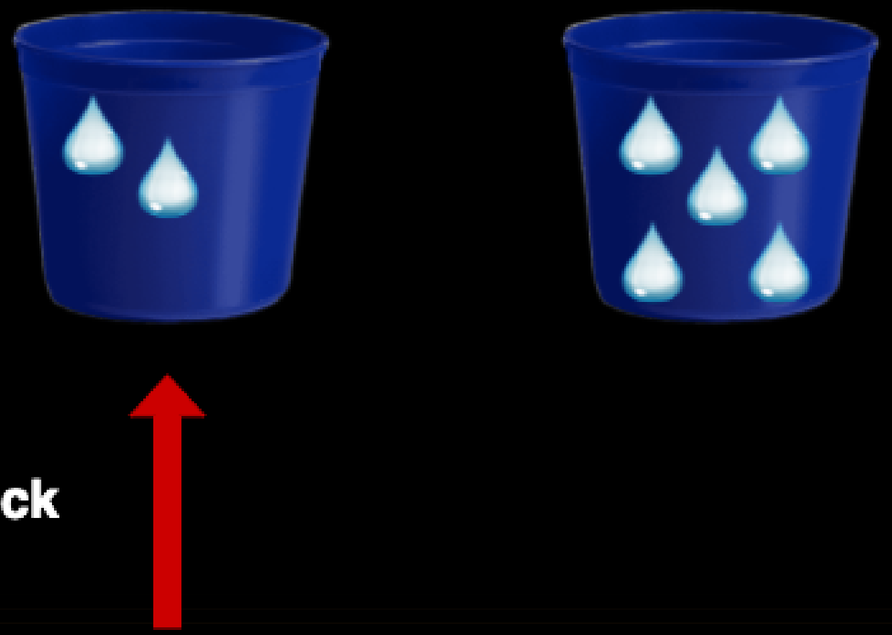
```java
public class MultiBandwidthExample {

    public static void main(String[] args) throws InterruptedException {

        Bandwidth nineConsumesPerMinuteLimit = Bandwidth.simple(9, Duration.ofMinutes(1));
        Bandwidth fiveConsumesPerSecondLimit = Bandwidth.simple(5, Duration.ofSeconds(1));

        Bucket bucket = Bucket.builder()
                .addLimit(nineConsumesPerMinuteLimit)
                .addLimit(fiveConsumesPerSecondLimit)
                .build();

        bucket.tryConsume(5); //true
        Thread.sleep(1000L);
        bucket.tryConsume(5); //false
        bucket.getAvailableTokens(); //4
    }
```

Improvement of the algorithm

Improvement of the algorithm

Improvement of the algorithm

Improvement of the algorithm

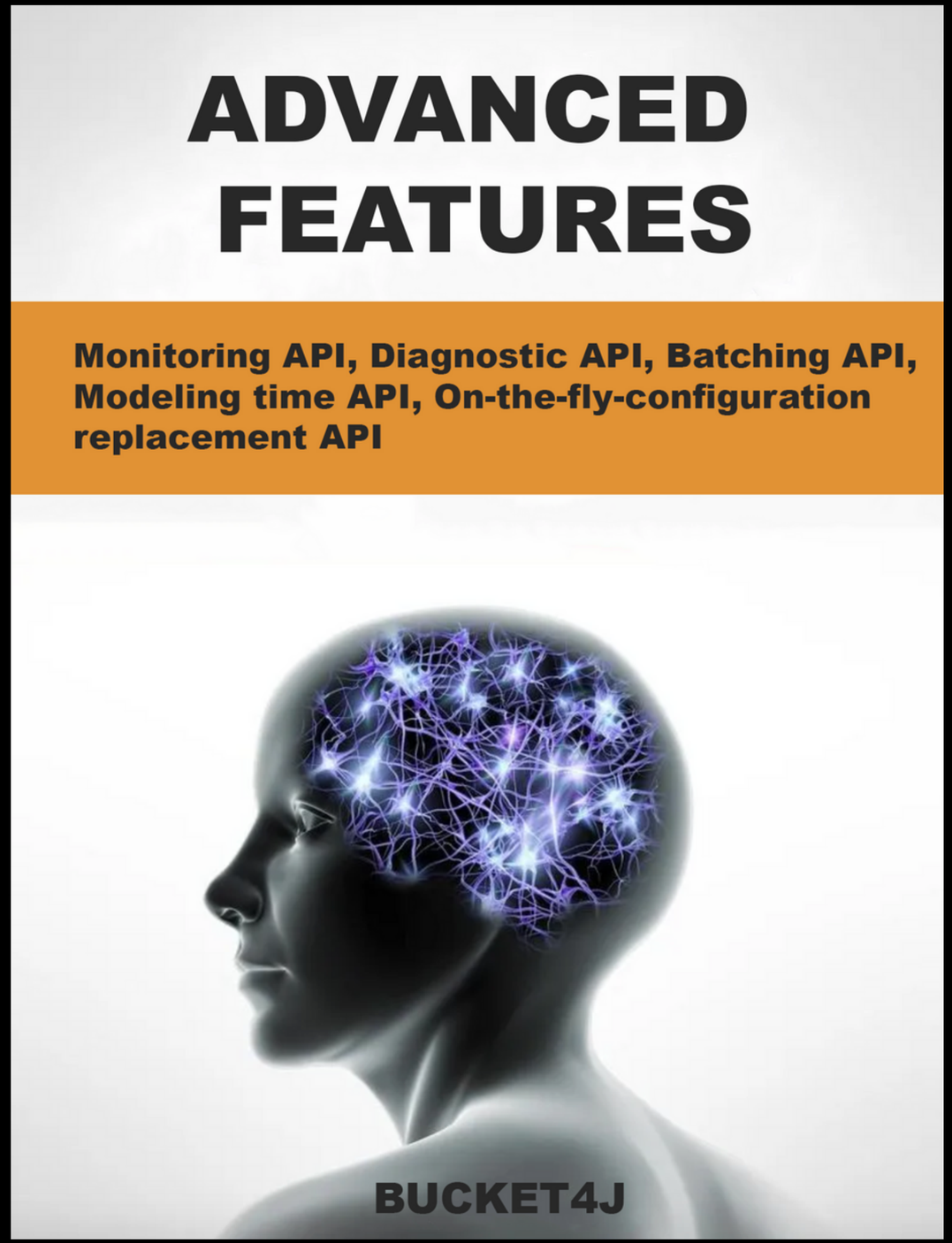


Improvement of the algorithm

Distributed Code Example

```
public static void main(String[] args) {
    HazelcastInstance hazelcastInstance = HazelcastClient.newHazelcastClient(hazelcastConfig());
    IMap<String, byte[]> bucketsMap = hazelcastInstance.getMap("bucket-map");
    ProxyManager<String> proxyManager = new HazelcastProxyManager<>(bucketsMap);

    BucketConfiguration configuration = BucketConfiguration.builder()
            .addLimit(Bandwidth.simple(9, Duration.ofMinutes(1)))
            .addLimit(Bandwidth.simple(5, Duration.ofSeconds(1)))
            .build();
    Bucket bucket = proxyManager.builder().build("key", configuration);
    bucket.tryConsume(5); //true
    bucket.tryConsume(5); //false

}

public static ClientConfig hazelcastConfig() {
    ClientConfig clientConfig = new ClientConfig();
    //Configuration of your client for Hazelcast instance...
    return clientConfig;
}
```

Advanced Features



Monitoring API

Monitoring API

- Consumed tokens

Monitoring API

- Consumed tokens
- Rejected tokens

Monitoring API

- Consumed tokens
- Rejected tokens
- Delayed nanos (Scheduler API)
- Parked nanos (Blocking API)
- On interrupted (Blocking API)

Monitoring API

Code example

```java
@Bean
public Bucket listenerBucket(MeterRegistry meterRegistry) {
    Bucket bucket = Bucket.builder().addLimit(Bandwidth.simple( capacity: 10, Duration.ofMinutes(1))).build();
    BucketListener listener = new BucketListener() {
        Counter onConsumedCounter = meterRegistry.counter( name: "rate-limiter" + "onConsumed");
        Counter onRejectedCounter = meterRegistry.counter( name: "rate-limiter" + "onRejected");
        @Override
        public void onConsumed(long tokens) {
            onConsumedCounter.increment(tokens);
        }

        @Override
        public void onRejected(long tokens) {
            onRejectedCounter.increment(tokens);
        }

        @Override
        public void onParked(long nanos) {

        }

        @Override
        public void onInterrupted(InterruptedException e) {

        }

        @Override
        public void onDelayed(long nanos) {

        }
    };
    return bucket.toListenable(listener);
}
```

Monitoring API

How it works in practice

On-the-fly configuration replacement

On-the-fly configuration replacement

Use cases

On-the-fly configuration replacement

Use cases

- Change limits of tariffications

On-the-fly configuration replacement

Use cases

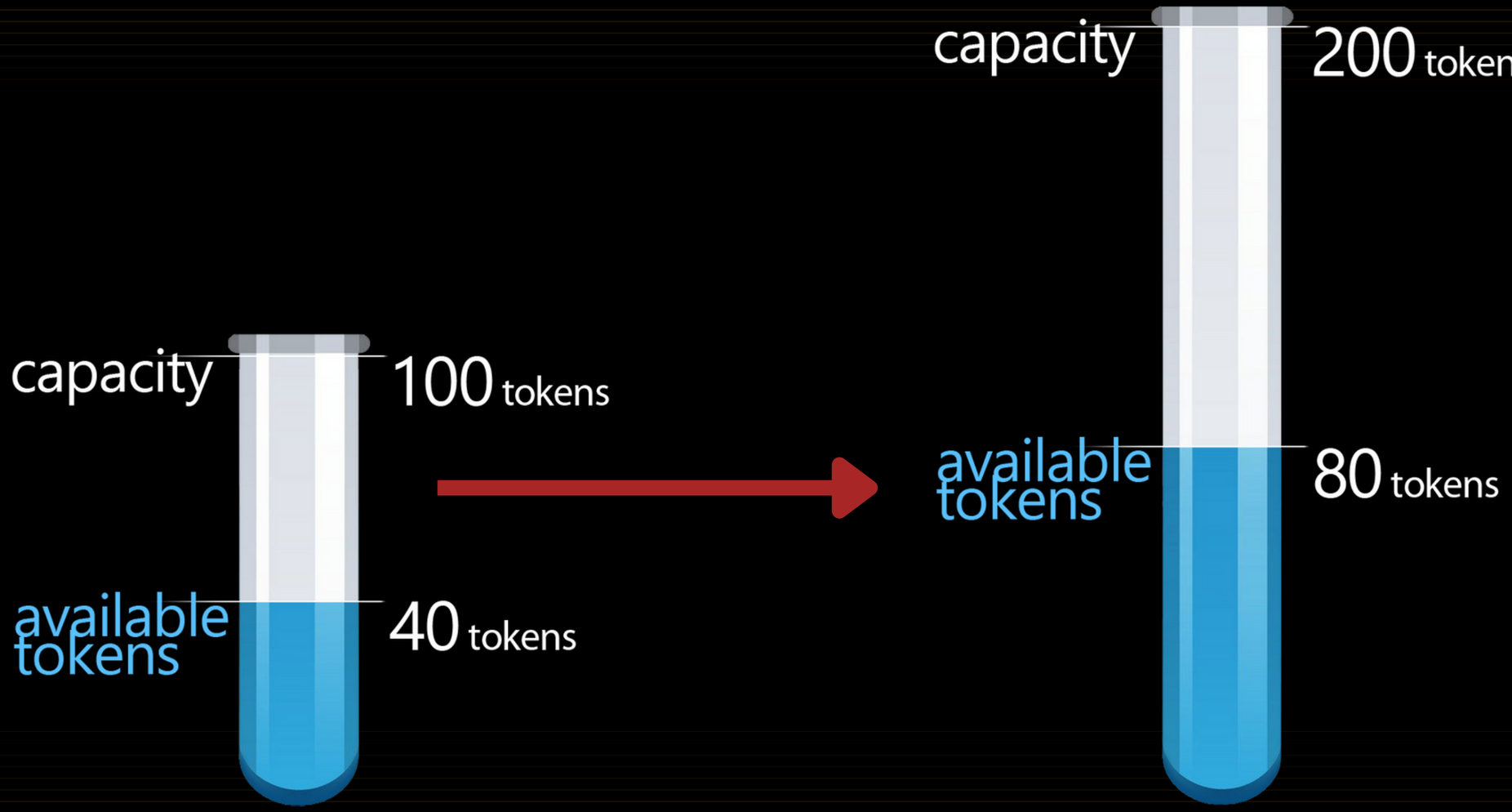
- Change limits of tariffications
- Change limits to call on external API

On-the-fly configuration replacement

Types of strategies

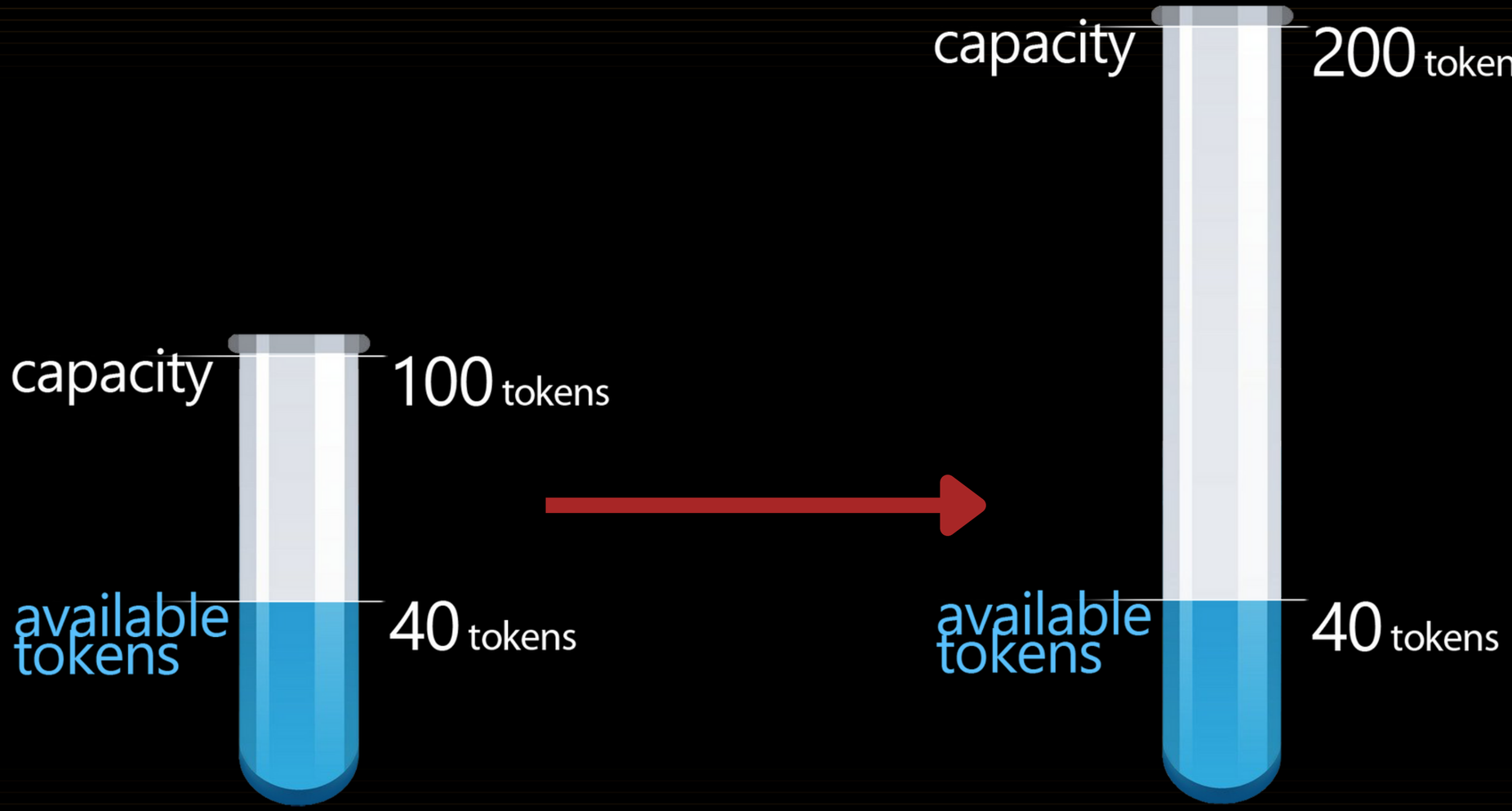
On-the-fly configuration replacement

PROPORTIONALLY



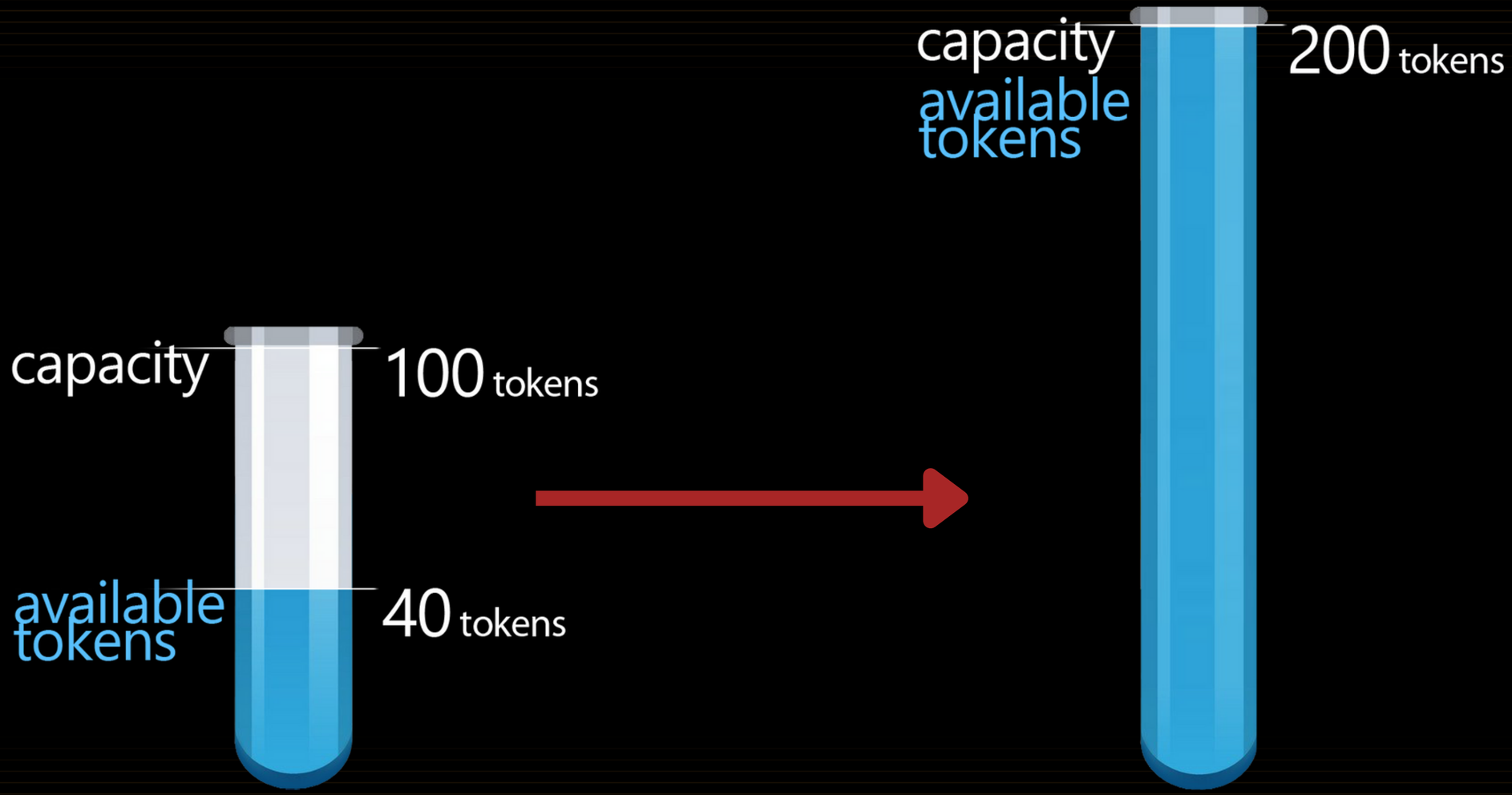
On-the-fly configuration replacement

AS_IS



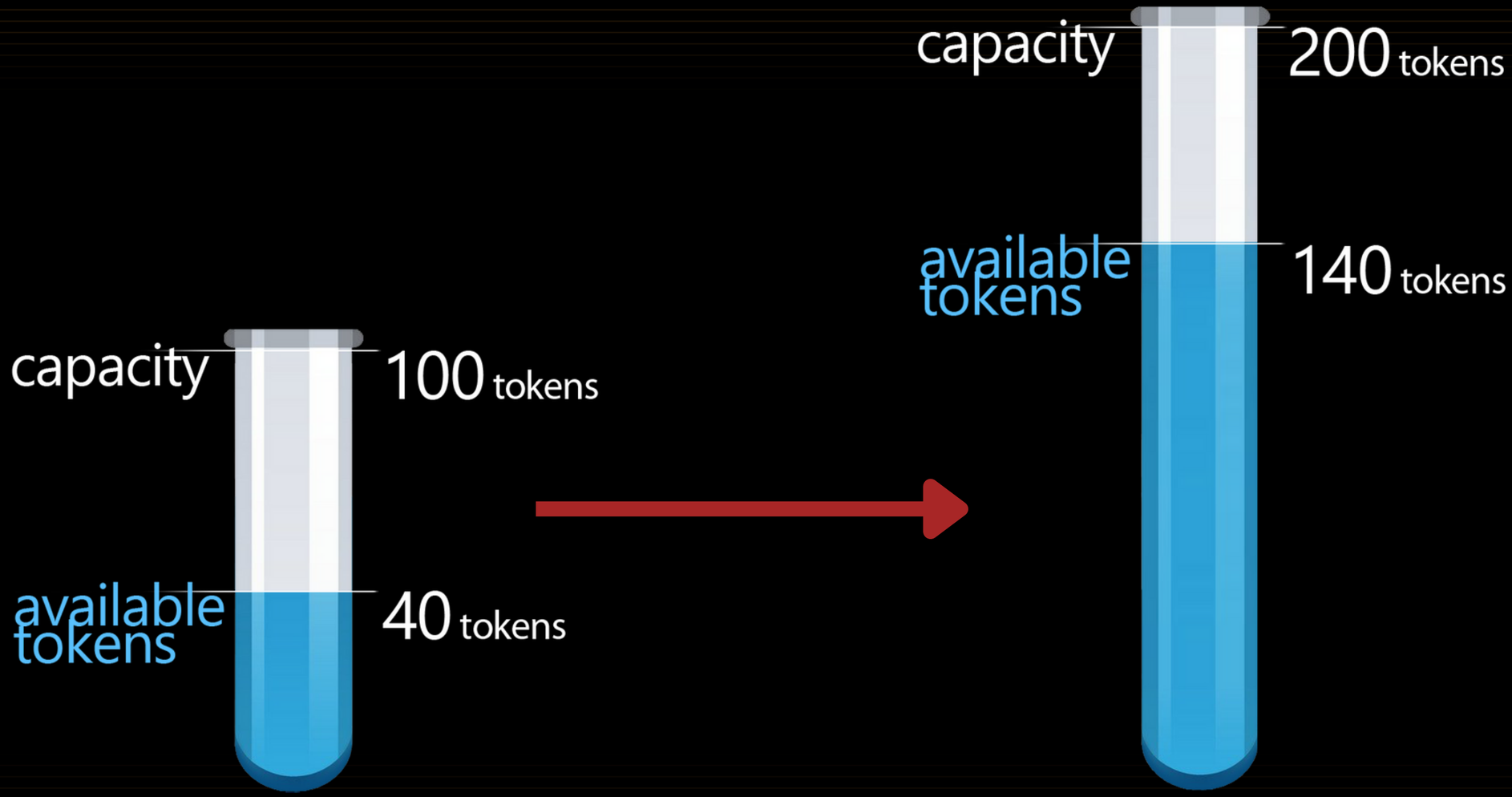
On-the-fly configuration replacement

RESET



On-the-fly configuration replacement

ADDITIVE



On-the-fly configuration replacement

Code example

```
public static void main(String[] args) {
    Bucket bucket = Bucket.builder()
            .addLimit(Bandwidth.simple( capacity: 10000, Duration.ofHours(1)).withId("business-limit"))
            .addLimit(Bandwidth.simple( capacity: 10, Duration.ofSeconds(1)).withId("technical-limit"))
            .build();
    //....
    BucketConfiguration newConfiguration = BucketConfiguration.builder()
            .addLimit(Bandwidth.simple( capacity: 5000, Duration.ofHours(1)).withId("business-limit"))
            .addLimit(Bandwidth.simple( capacity: 100, Duration.ofSeconds(10)).withId("technical-limit"))
            .build();
    bucket.replaceConfiguration(newConfiguration, TokensInheritanceStrategy.PROPORTIONALLY);
}
```

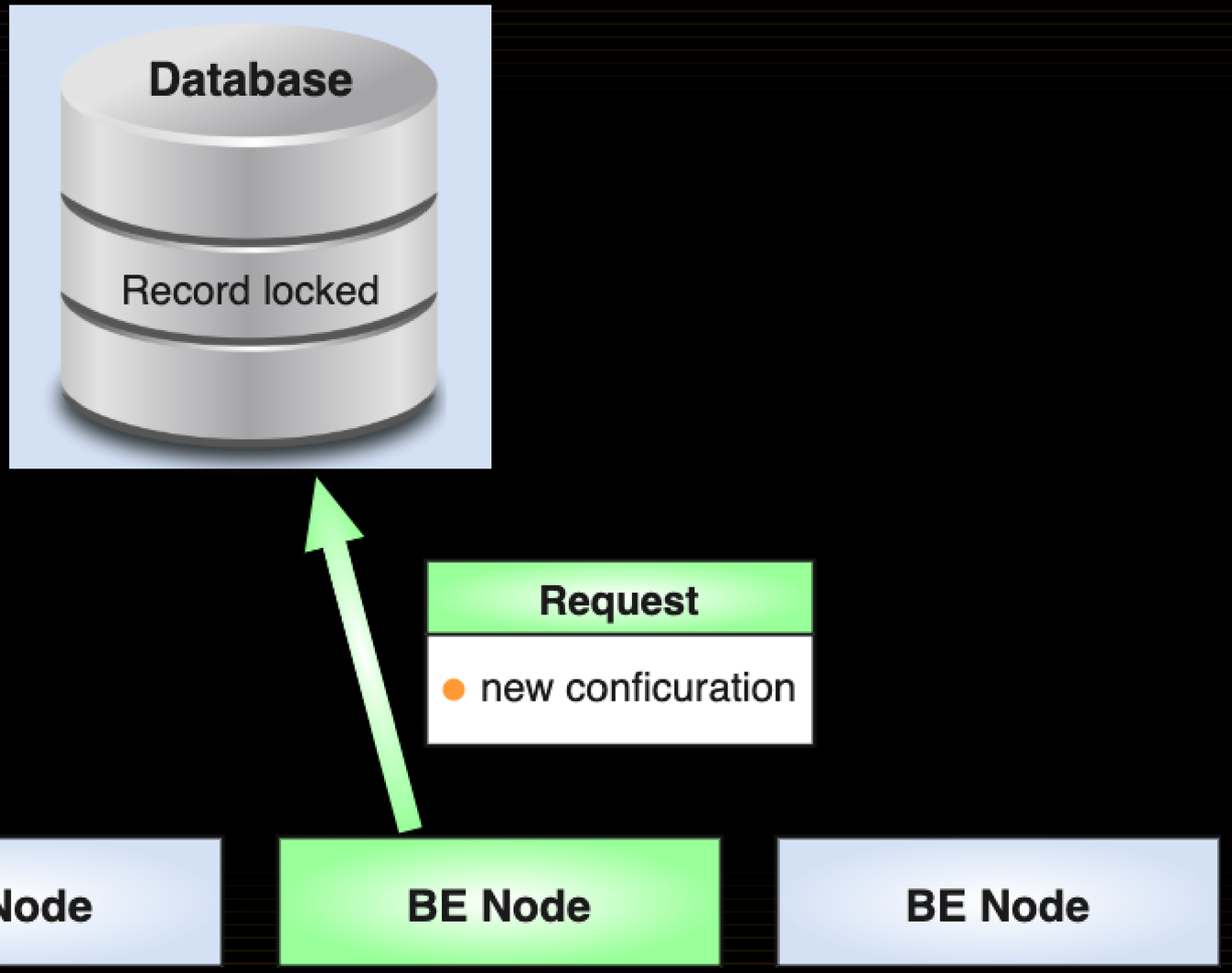
On-the-fly configuration replacement

How it works

BE Node | BE Node | BE Node | BE Node

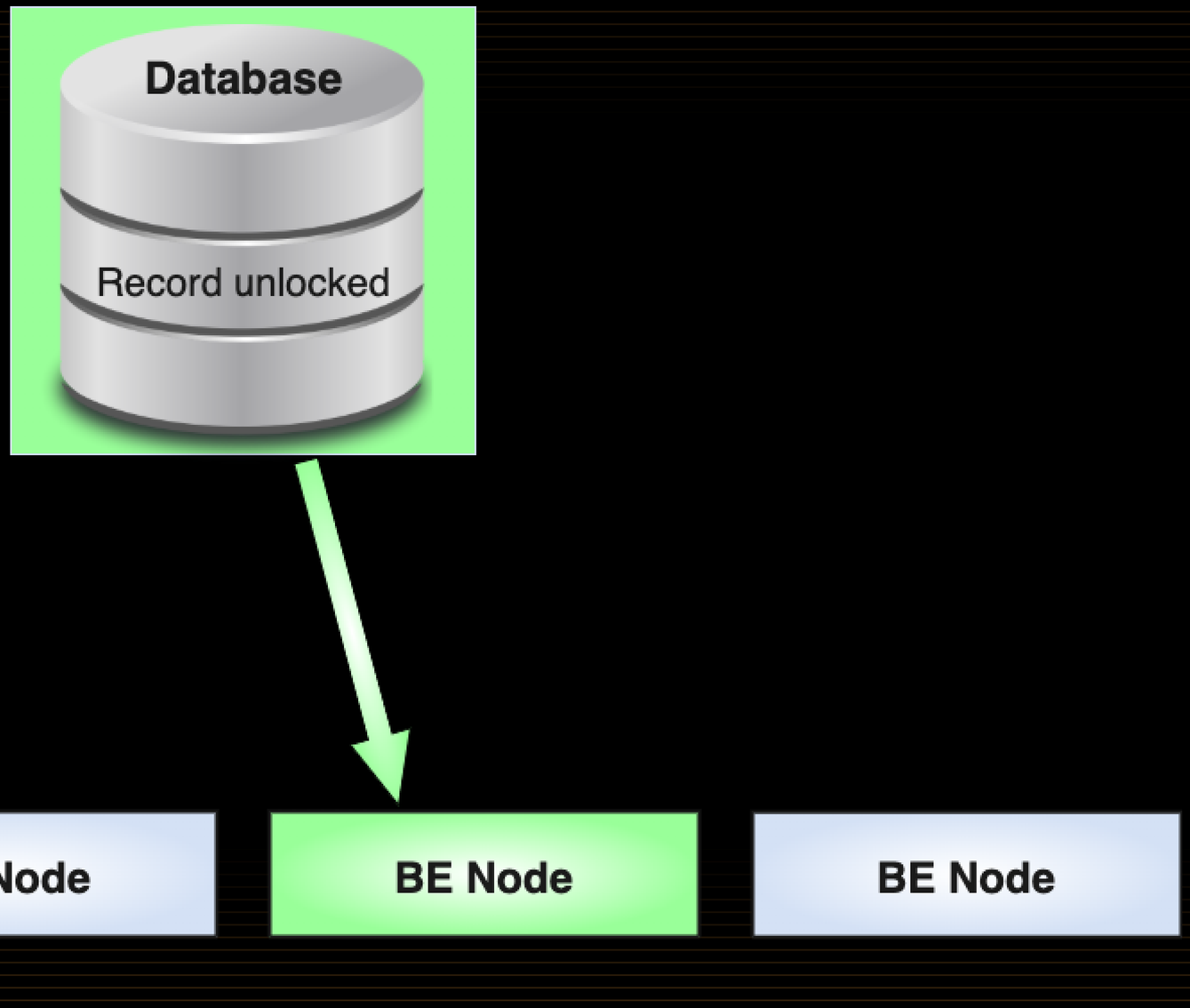
On-the-fly configuration replacement

How it works



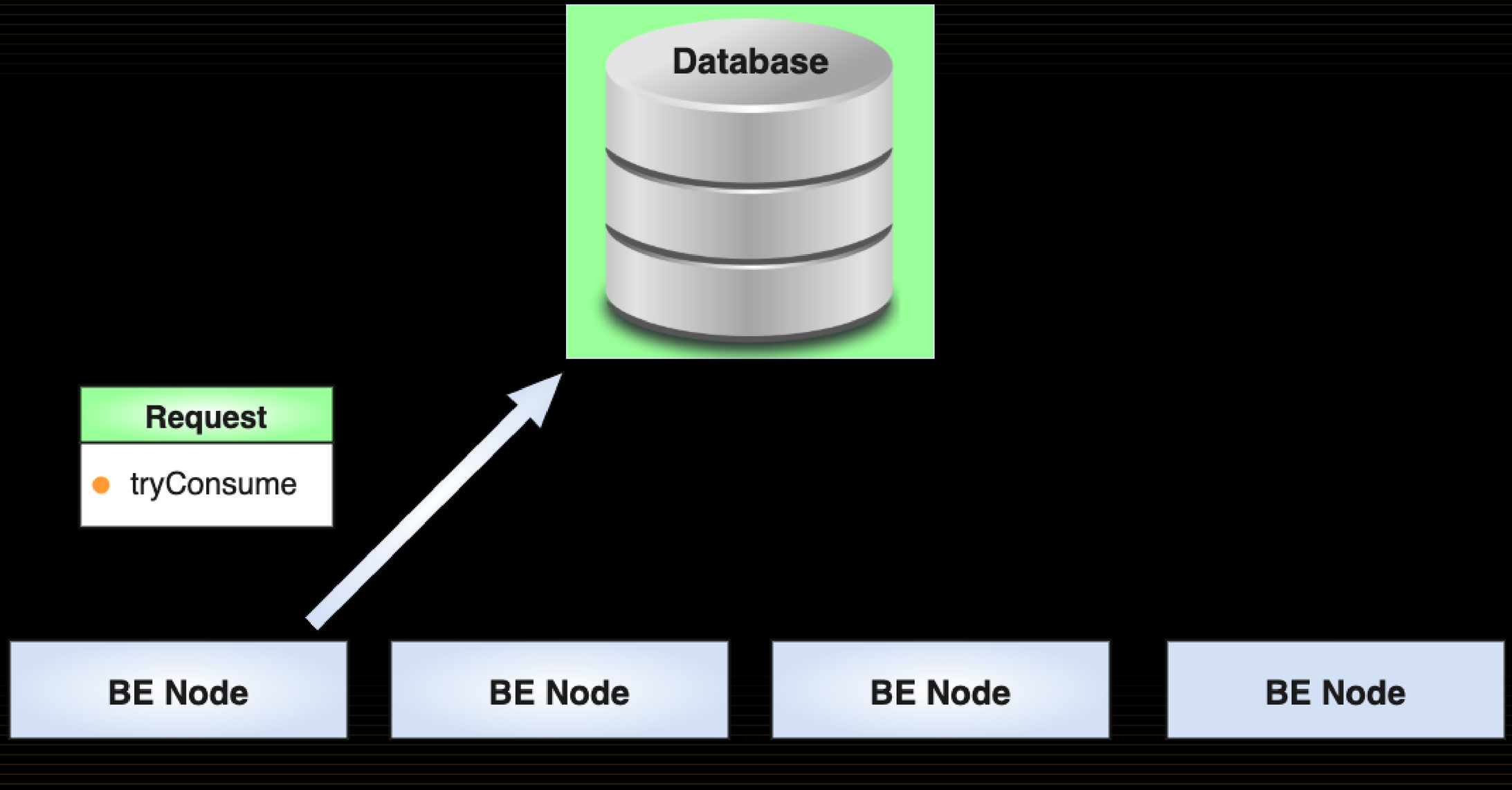
On-the-fly configuration replacement

How it works



On-the-fly configuration replacement

How it works



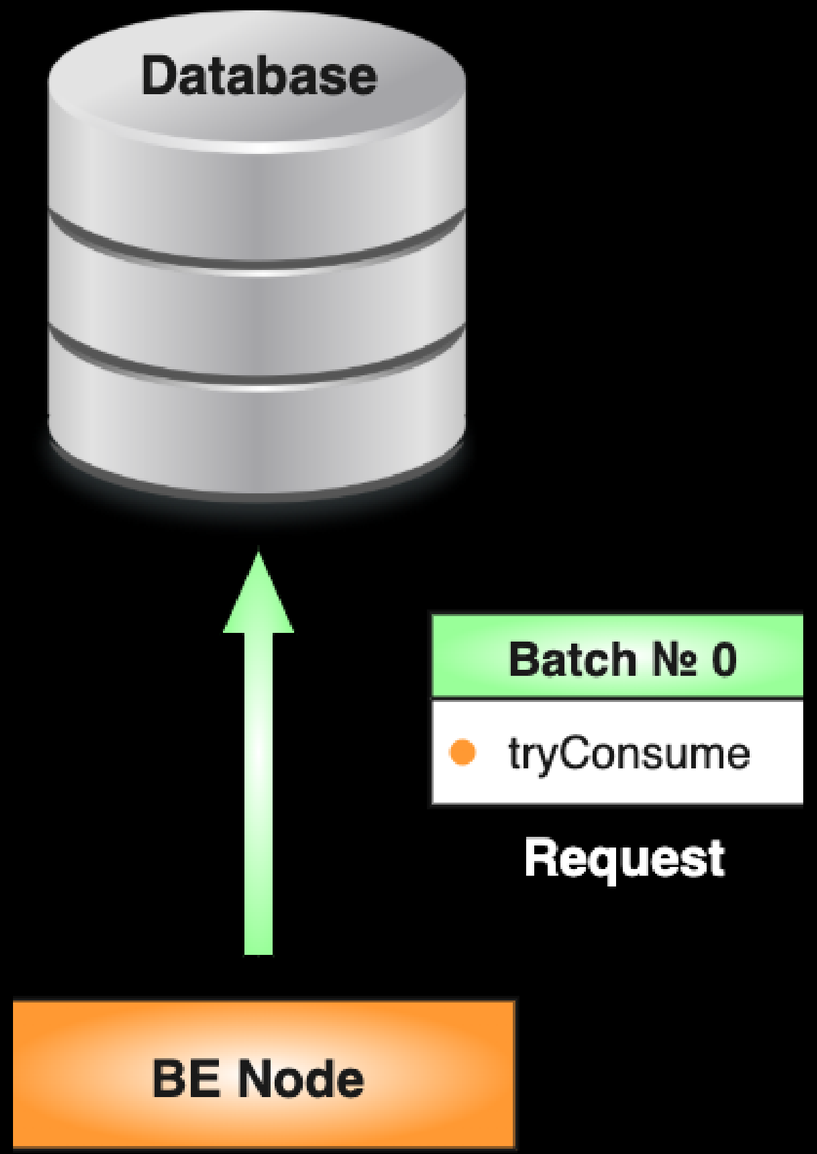
On-the-fly configuration replacement

How it works in practice

Batching API

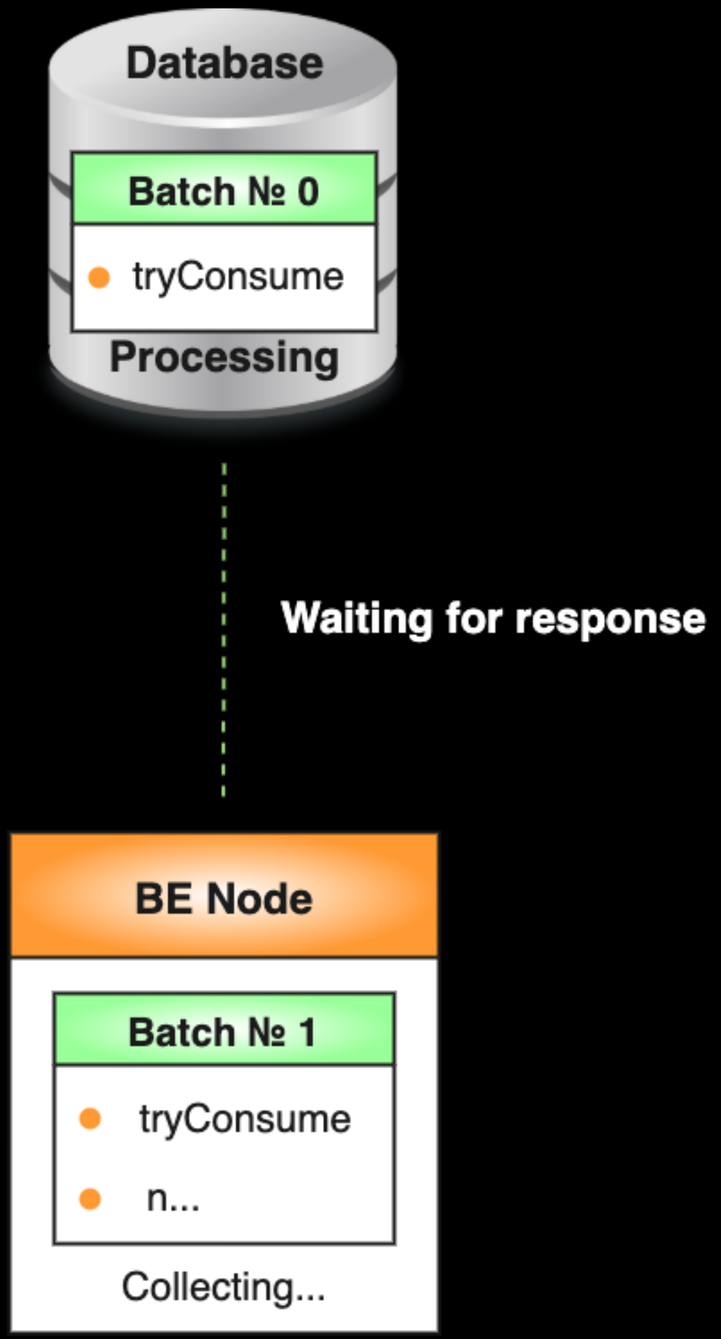
Batching API

What it is



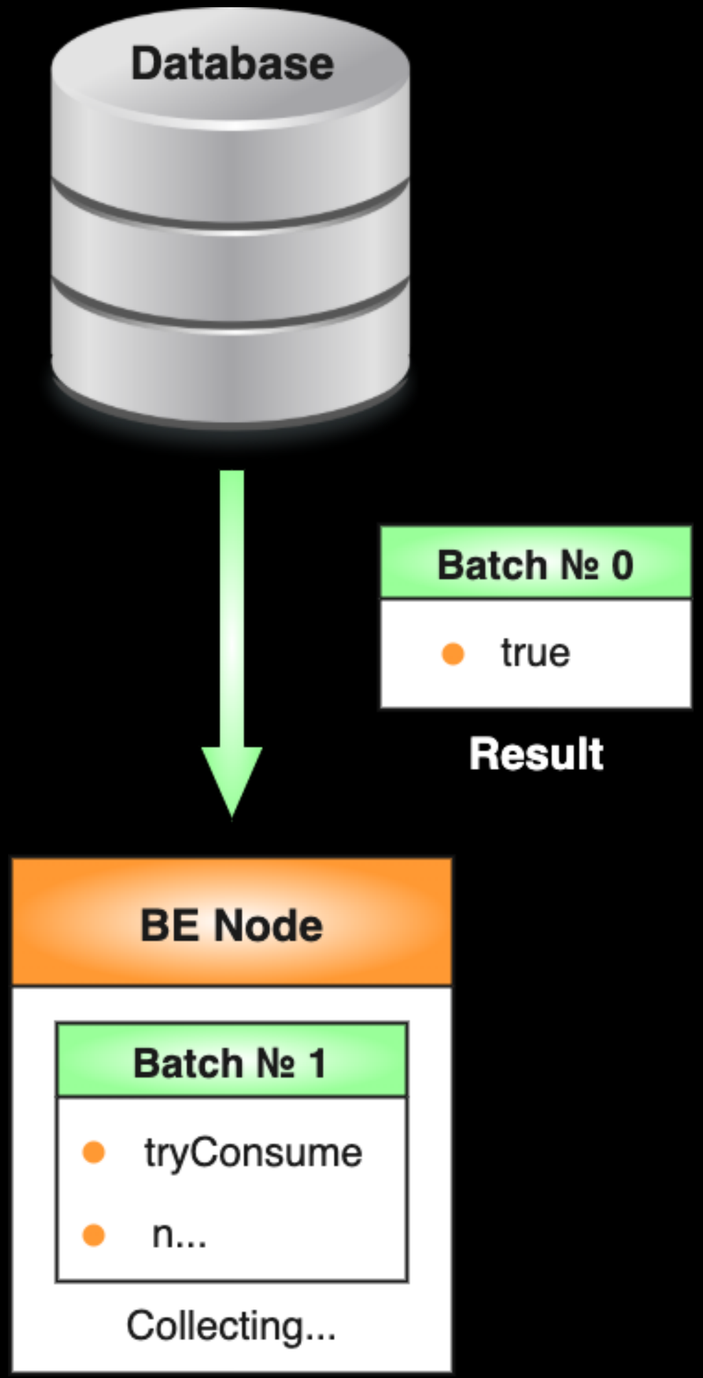
Batching API

What it is



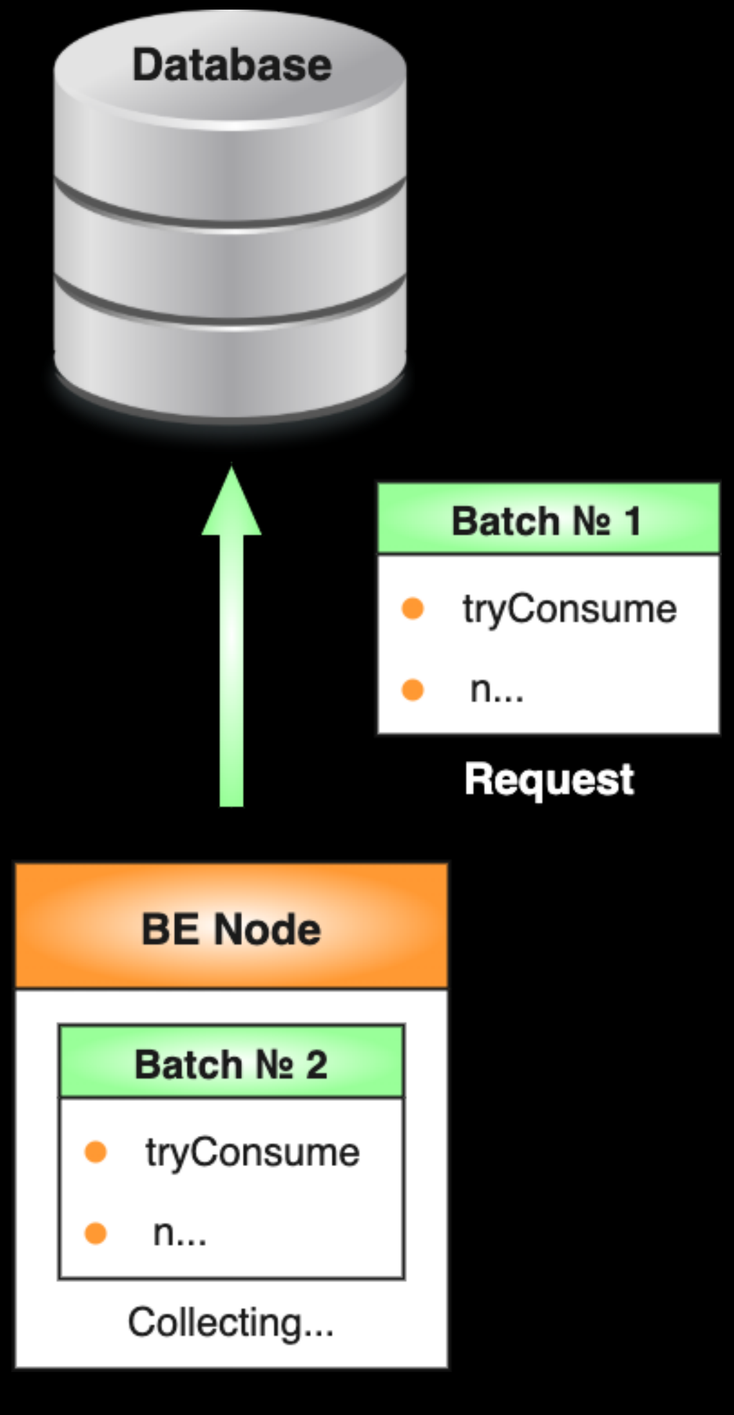
Batching API

What it is



Batching API

What it is



Where it need

- **High Load calls by one key**

Batching API

Strategies

- Batching
- Delaying
- Predicting

Code example

```
Bucket bucket = proxyManager.builder()
        .withOptimization(Optimizations.batching())
        .build( key: "13", configuration);
```

How it works in practice

Conclusions

- Rate-Limiting - it's easy!

Conclusions

- Rate-Limiting - it's easy!
- Rate-Limiting is much easier with Bucket4j

Conclusions

- Rate-Limiting - it's easy!
- Rate-Limiting is much easier with Bucket4j
- Rate-Limiting on application level - it's normal!

Thanks for your attention!

Speaker Maxim Bartkov


maxgalayoutop@gmail.com


@MaximBartkov