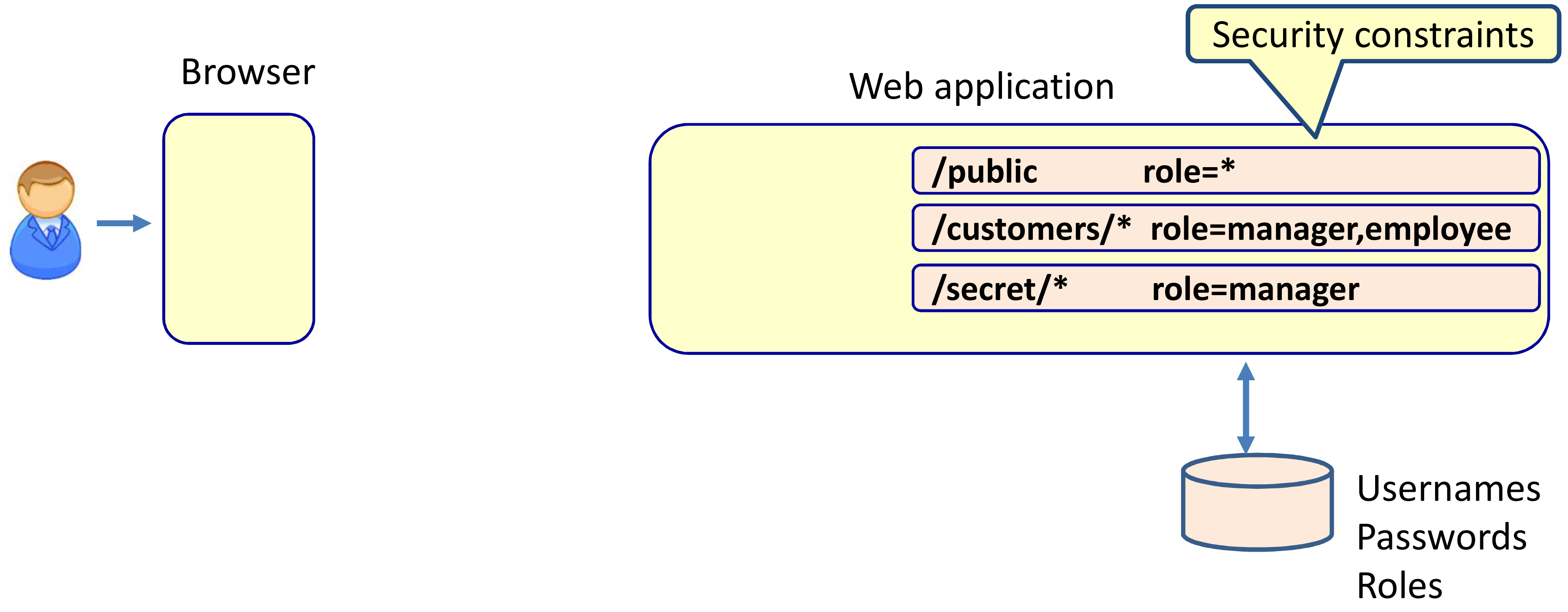
# SECURITY

# Aspects of security

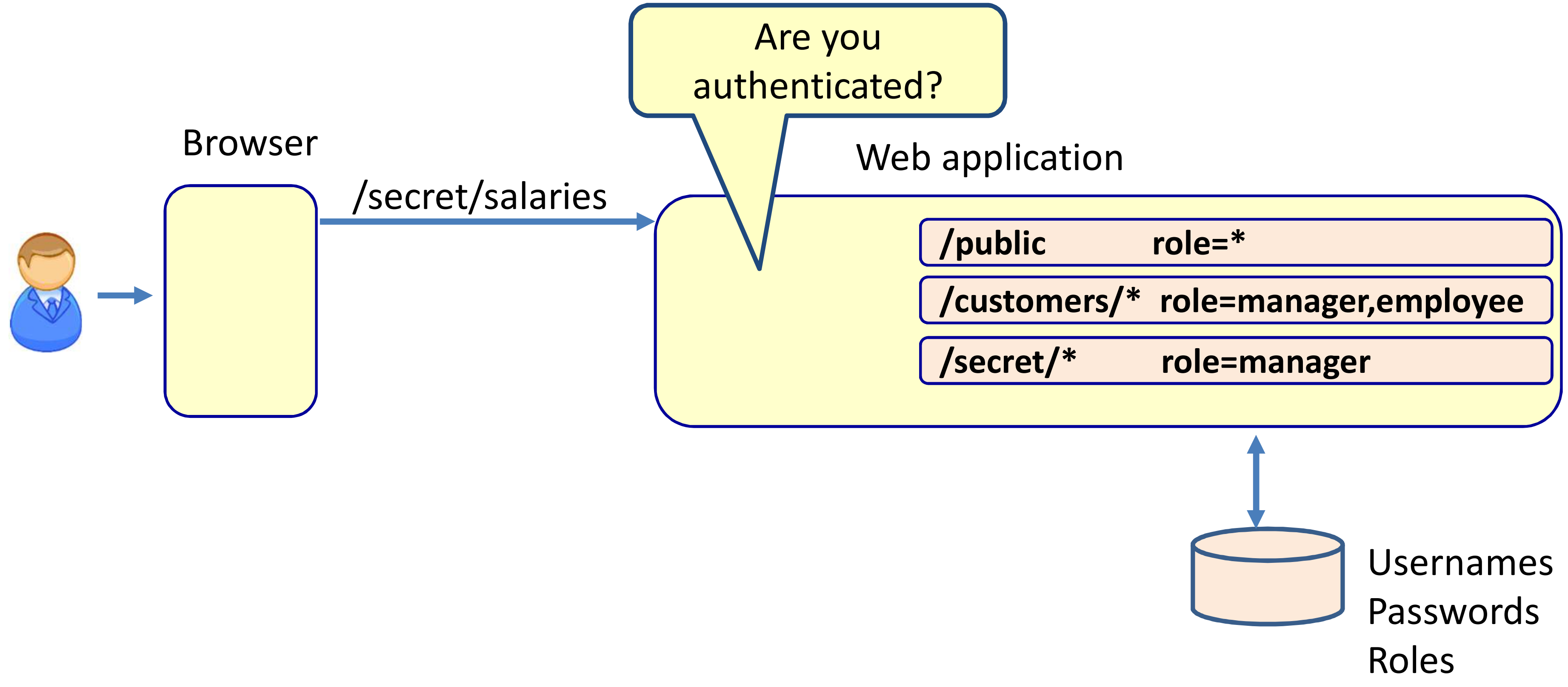
- Authentication: are you who you say you are?
  - Login with username/password
- Authorization: what are you allowed to do?
  - Make url's and/or methods secure
- Confidentiality: No one may look into this request/response
  - Encryption
- Data integrity: No one may change this request/response
  - Encryption, hashcode,…

## SECURING A WEB APPLICATION

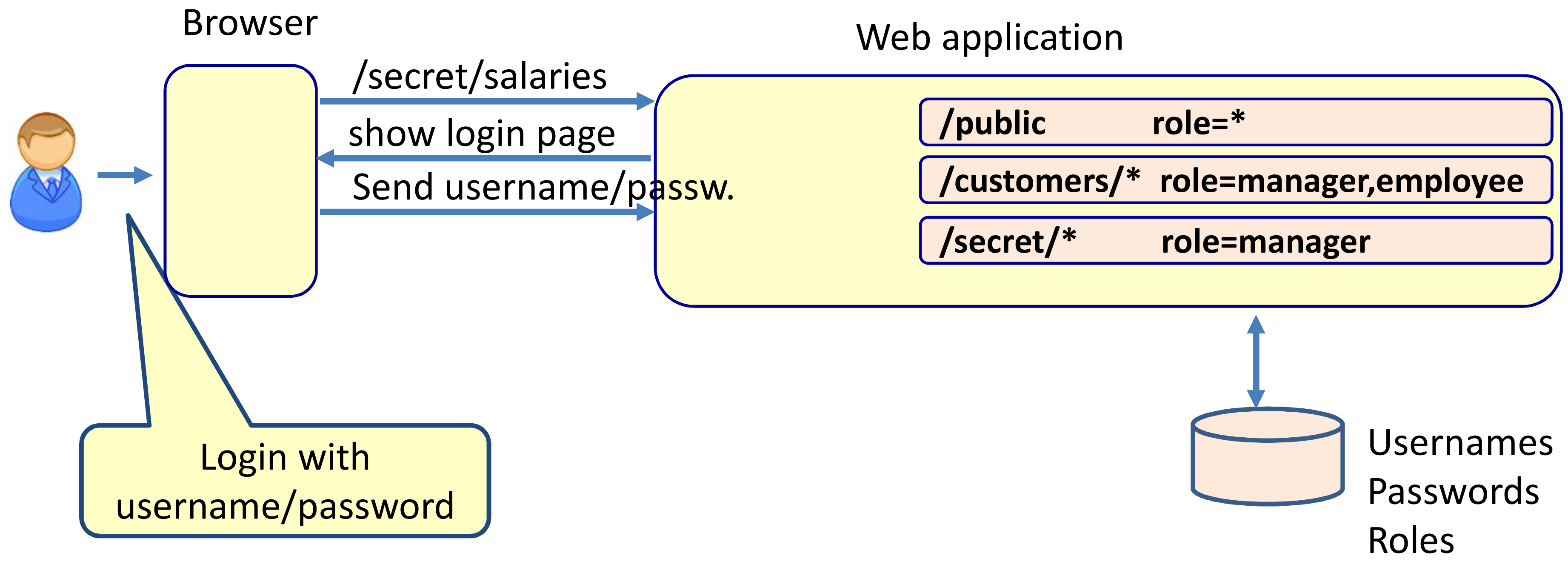
# Web security



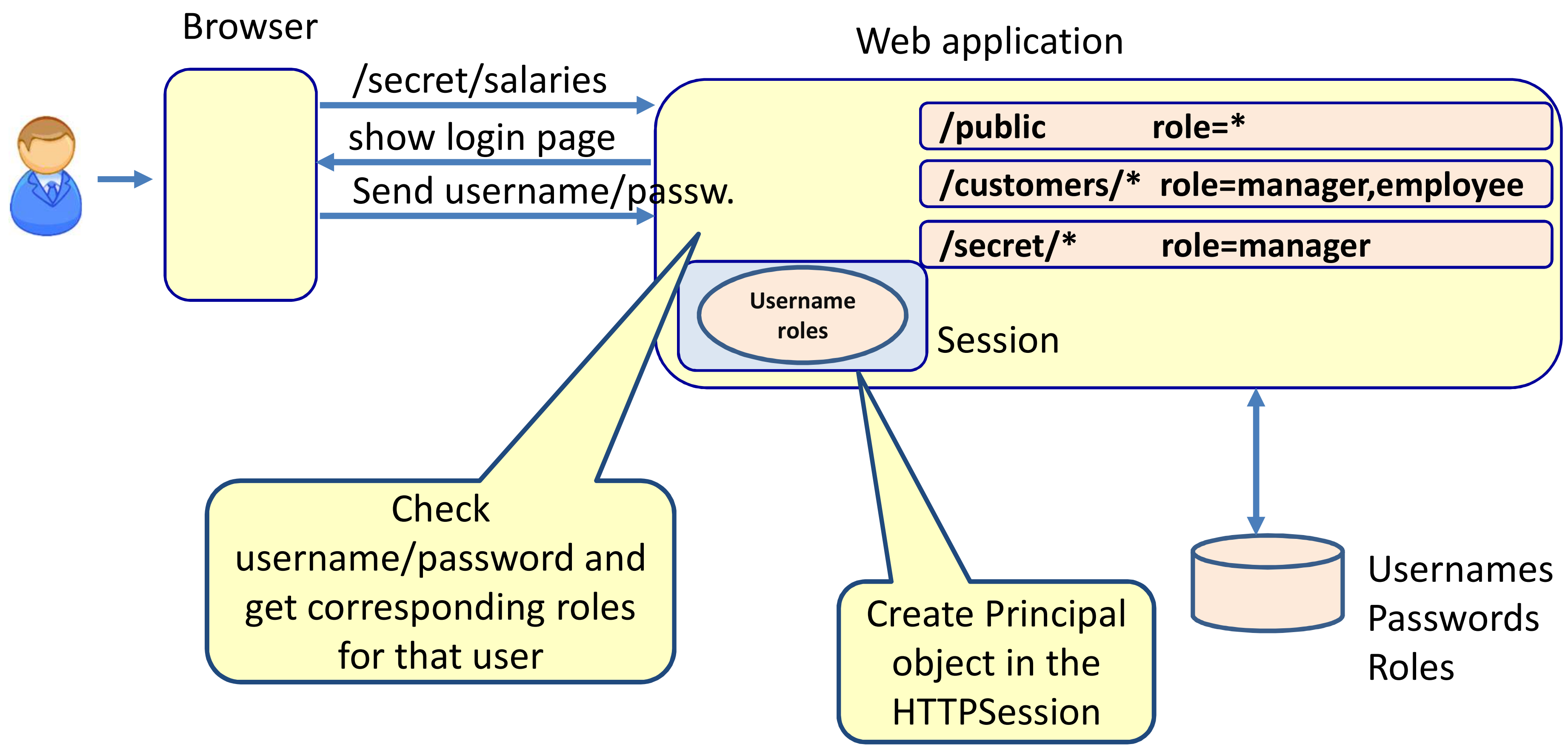
# Web security



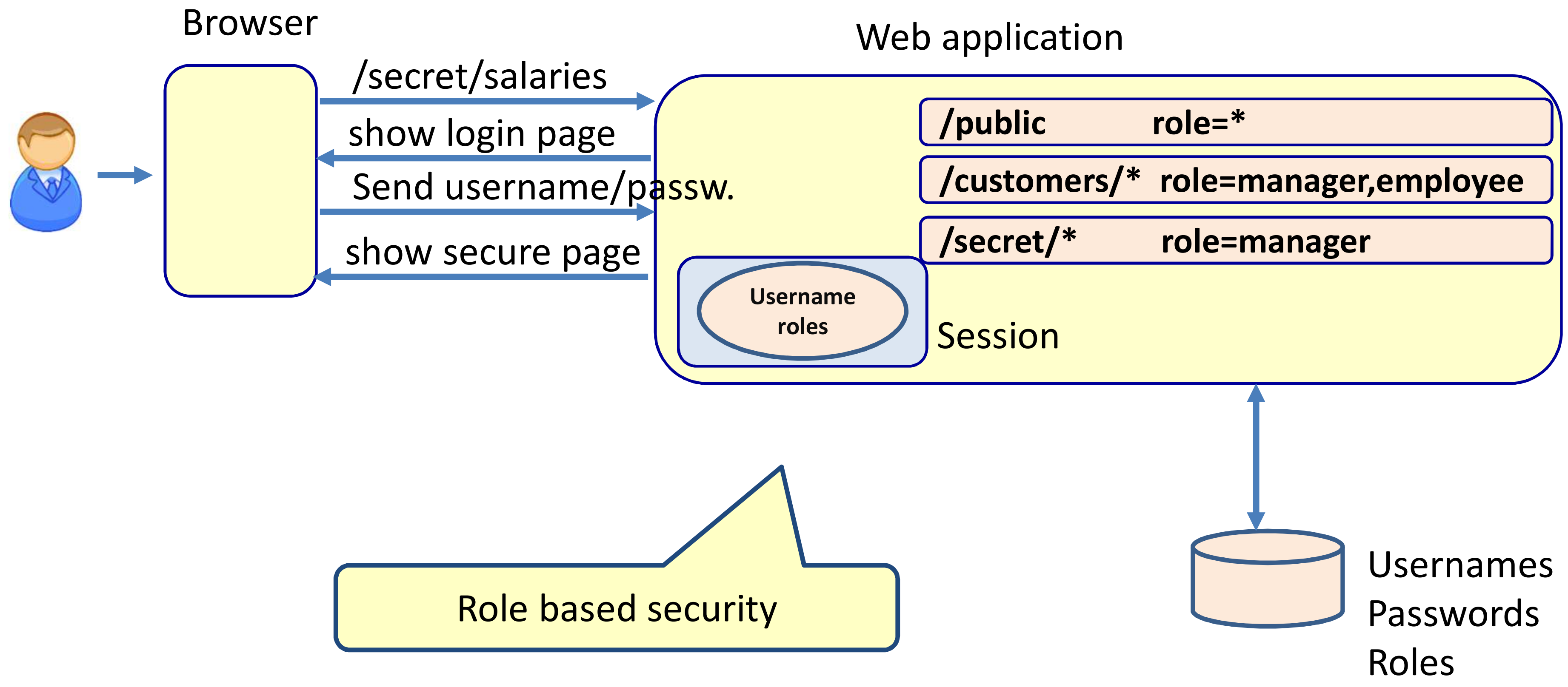
# Web security



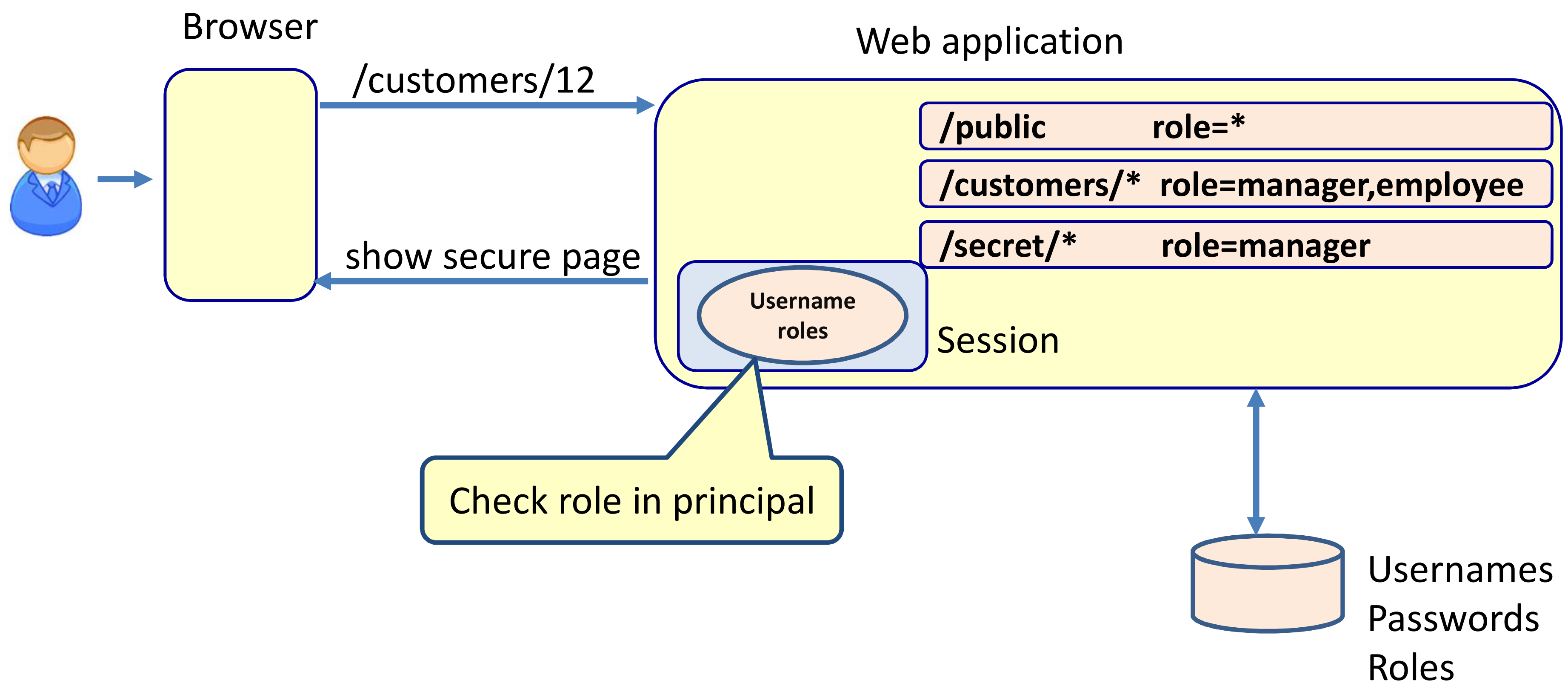
# Web security


Browser
/secret/salaries
show login page
Send username/passw.
Web application
/public role=*
/customers/* role=manager,employee
/secret/* role=manager
Username roles
Session
Check username/password and get corresponding roles for that user
Create Principal object in the HTTPSession
Usernames Passwords Roles

# Web security

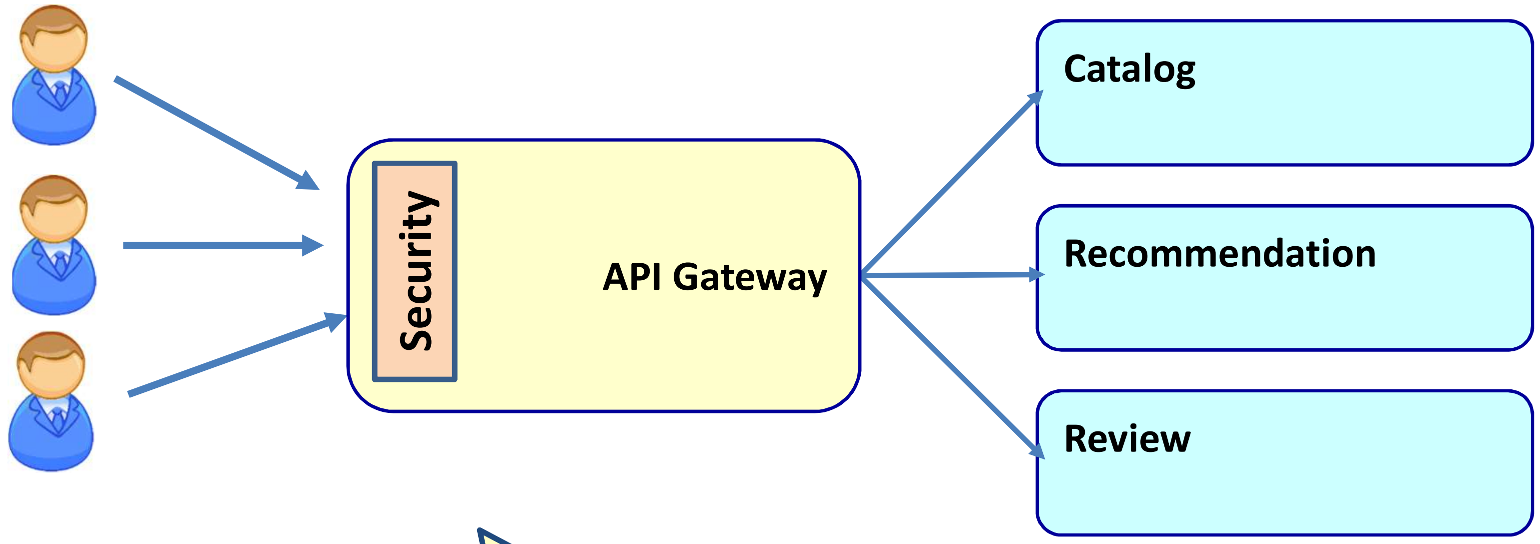


# Web security



## SECURE THE MICROSERVICE ARCHITECTURE

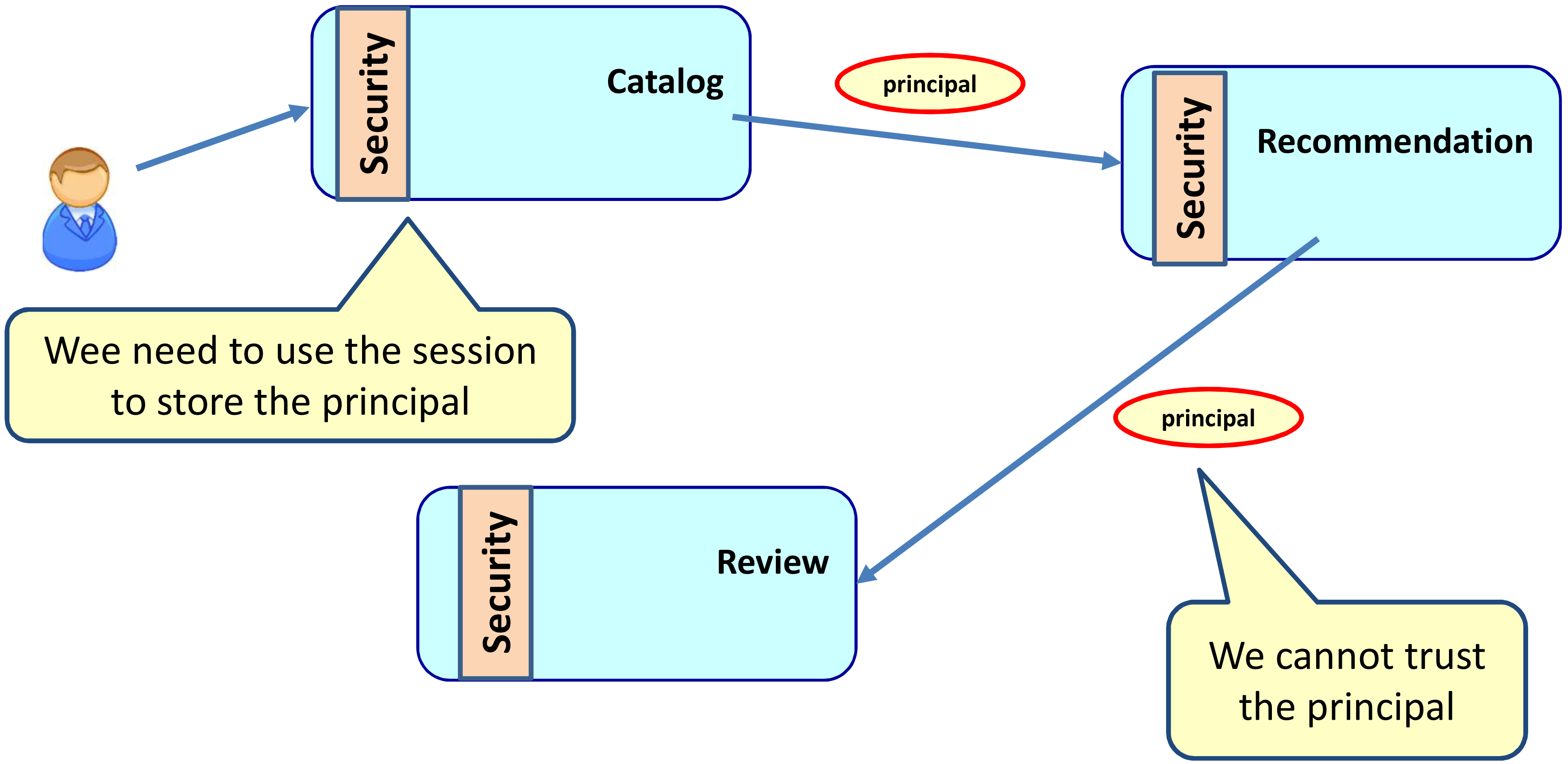
# Secure the API gateway



Client can bypass the API gateway

Does not prevent us from inside attacks

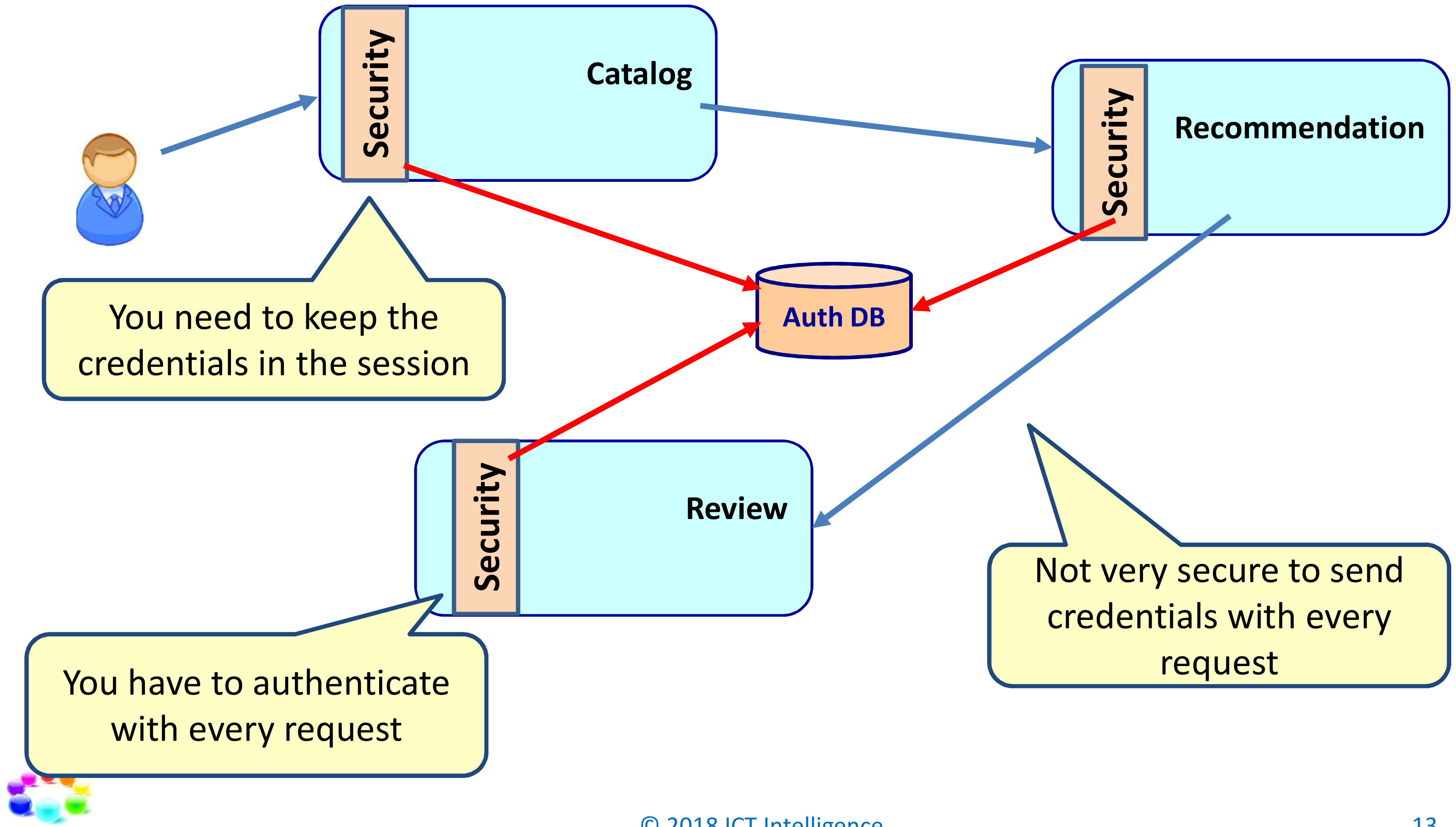
# Send principal with every request



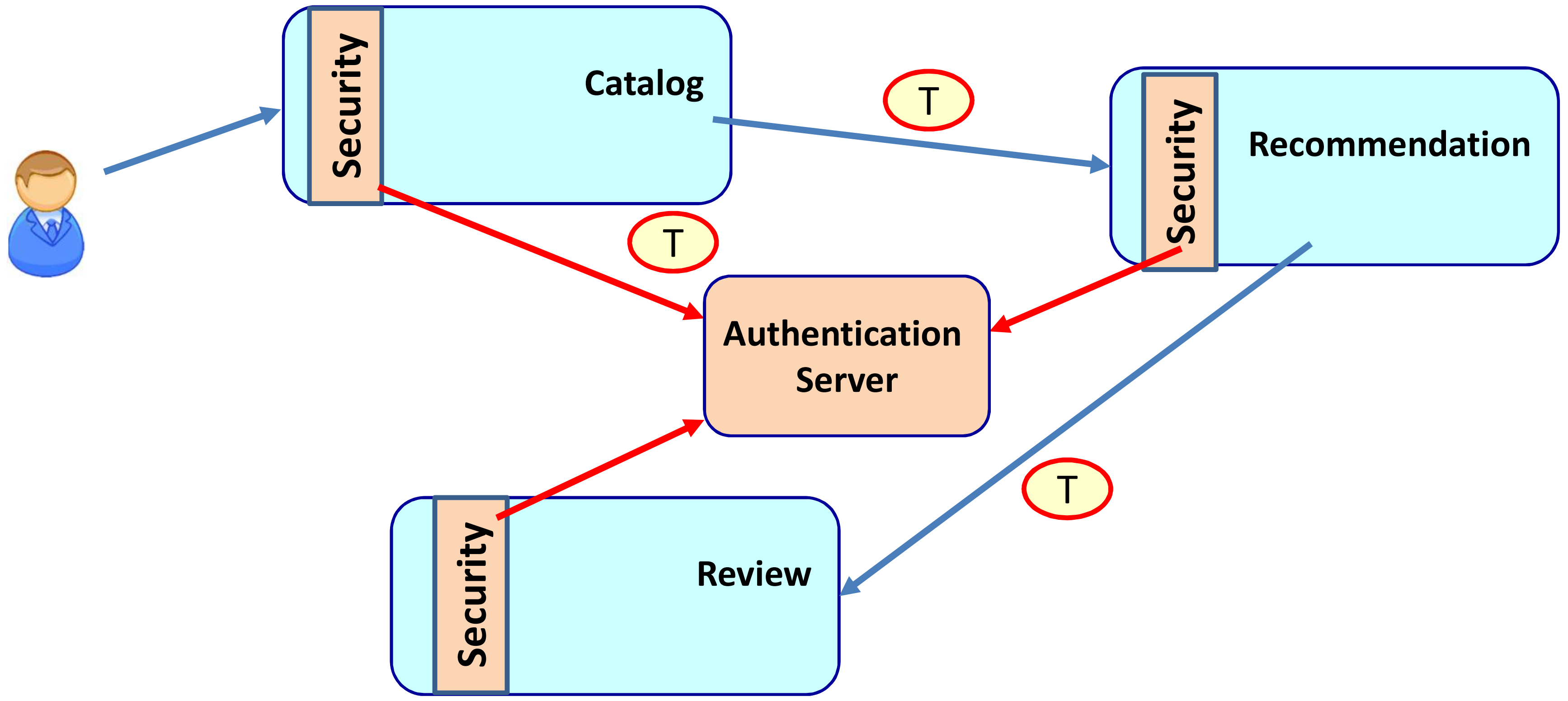
Wee need to use the session to store the principal

We cannot trust the principal

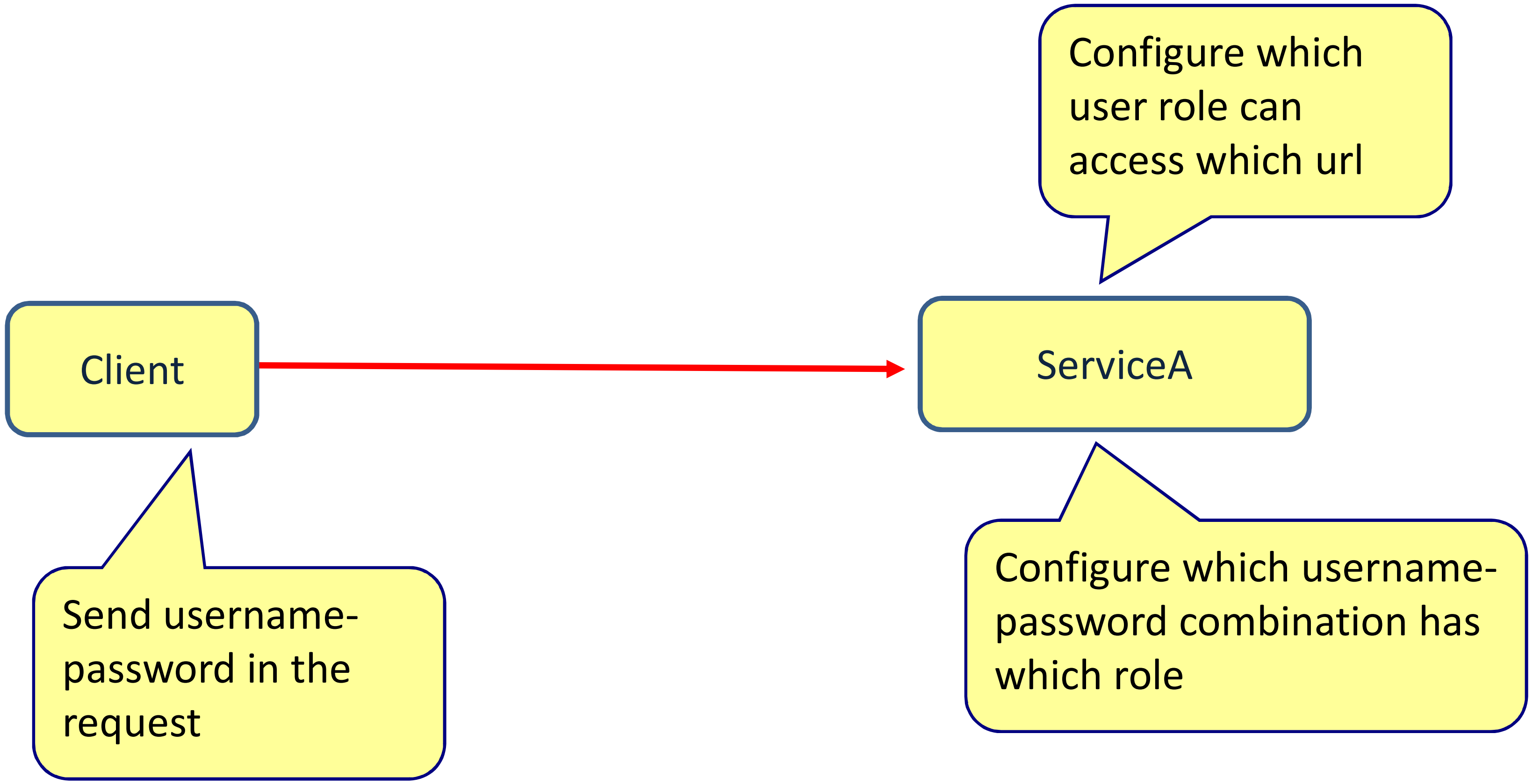
# Send userid/password with every request


Security
Catalog
Security
Recommendation
Auth DB
Security
Review
You need to keep the credentials in the session
Not very secure to send credentials with every request
You have to authenticate with every request

# OAuth2: Token based security



# Securing a REST service


Configure which user role can access which url
Client
ServiceA
Send username-password in the request
Configure which username-password combination has which role

# REST Server

```
@Configuration
@EnableWebSecurity
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {
    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http
            .authorizeRequests()
                .antMatchers("/productinfo").permitAll()
                .antMatchers("/salaryinfo").hasAnyRole("MANAGER")
                .and()
            .httpBasic();
    }

    // create users
    @Autowired
    public void configureGlobal(AuthenticationManagerBuilder auth) throws Exception {

        auth.inMemoryAuthentication()
                .withUser("manager").password("{noop}pass").roles("MANAGER");
    }
}
```

Configure which user role can access which url

In-memory users

Configure which username-password combination has which role

# Getting security info from the database

```
@Configuration
@EnableWebSecurity
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {

  @Autowired
  DataSource dataSource;

  @Autowired
  public void configAuthentication(AuthenticationManagerBuilder auth) throws
      Exception {

    auth.jdbcAuthentication().dataSource(dataSource)
      .usersByUsernameQuery(
           "select username,password, enabled from users where username=?")
      .authoritiesByUsernameQuery(
           "select username, role from user_roles where username=?");
  }

...
```

# REST Client

```
@Component
public class SecureRestClient {
  @Autowired
  private RestOperations restTemplate;
  private String serverUrl = "http://localhost:8080/";

  public void showProductInfo() {
    String productInfo= restTemplate.getForObject(serverUrl+"/productinfo", S     g.class);
    System.out.println("Receiving: "+productInfo);
  }
  public void showSalaryInfo() {
    HttpEntity<String> request = new HttpEntity<String>(createHeaders("manager","pass"));
    ResponseEntity<String> response = restTemplate.exchange(serverUrl+"/salaryinfo",
                                  HttpMethod.GET, request, String.class);
    String salaryInfo = response.getBody();
    System.out.println("Receiving: "+salaryInfo);
  }

  public HttpHeaders createHeaders(String username, String password) {
    HttpHeaders headers = new HttpHeaders();

    String auth = username + ":" + password;
    String encodedAuth =
      Base64.getEncoder().encodeToString(auth.getBytes(Charset.forName("US-ASCII")));
    String authHeader = "Basic " + encodedAuth;
    headers.set("Authorization", authHeader);
    return headers;
  }
}
```

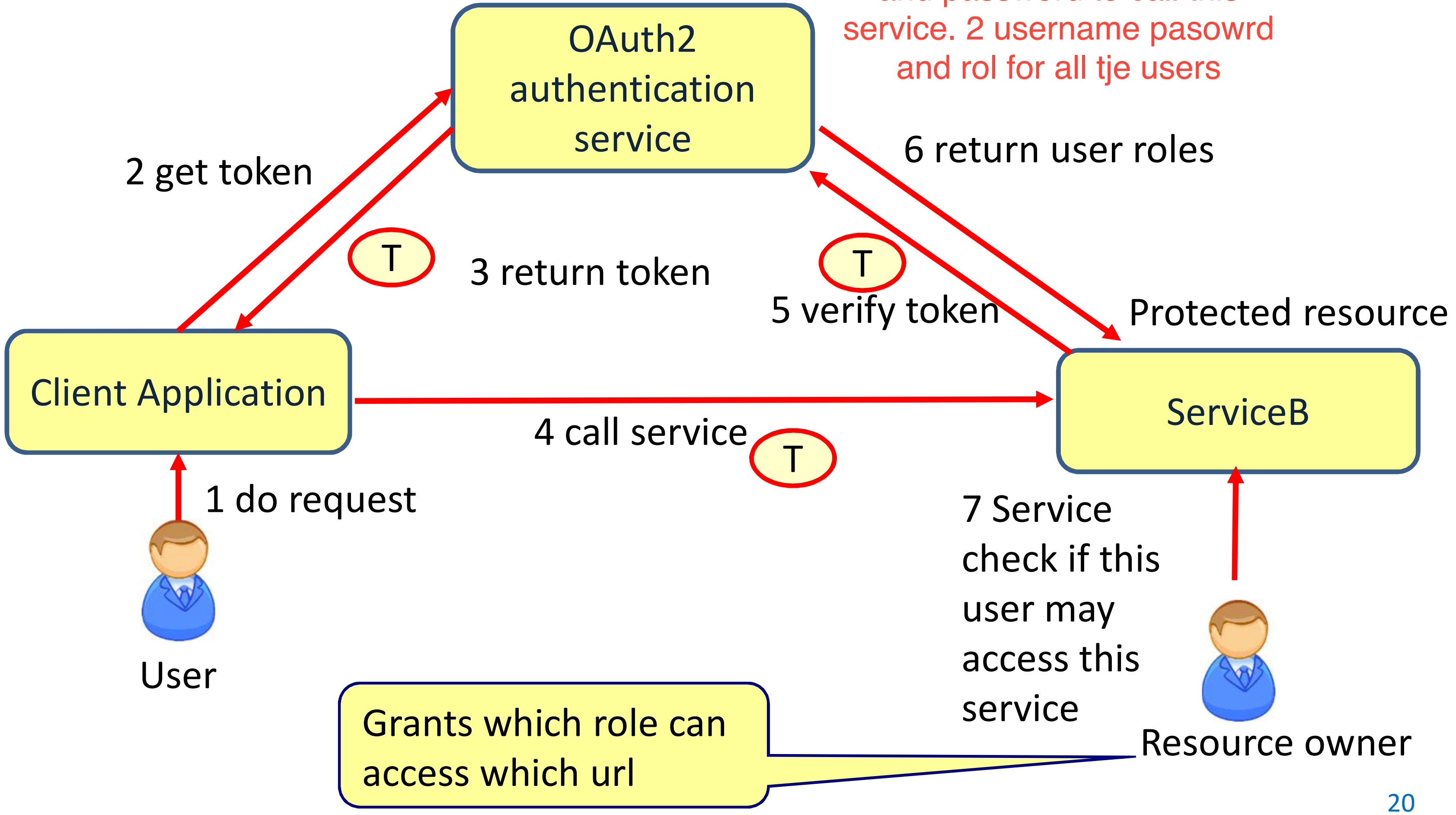
Add username and password as a header

## OAUTH2

# How does OAuth2 work

- Token based authentication and authorization framework

configure 2 things, 1 username and password to call this service. 2 username pasowrd and rol for all tje users



## OAUTH2 AUTHENTICATION SERVICE

# The authentication service

```java
@SpringBootApplication
@RestController
@EnableResourceServer
@EnableAuthorizationServer
public class AuthenticationServiceApplication {

  public static void main(String[] args) {
    SpringApplication.run(AuthenticationServiceApplication.class, args);
  }

  @RequestMapping(value = { "/user" }, produces = "application/json")
  public Map<String, Object> user(OAuth2Authentication user) {
    Map<String, Object> userInfo = new HashMap<>();
    userInfo.put("user", user.getUserAuthentication().getPrincipal());
    userInfo.put("authorities",
      AuthorityUtils.authorityListToSet(user.getUserAuthentication().getAuthorities()));
    return userInfo;
    }

}
```

# OAuth2 configuration

```java
@Configuration
public class OAuth2Config extends AuthorizationServerConfigurerAdapter {
    @Autowired
    private AuthenticationManager authenticationManager;

    @Autowired
    private UserDetailsService userDetailsService;

    @Override
    public void configure(ClientDetailsServiceConfigurer clients) throws Exception {
        clients.inMemory()
                .withClient("theClient")
                .secret("{noop}thisissecret")
                .authorizedGrantTypes("refresh_token", "password", "client_credentials")
                .scopes("webclient", "mobileclient");
    }
    @Override
    public void configure(AuthorizationServerEndpointsConfigurer endpoints) throws
           Exception {
      endpoints
        .authenticationManager(authenticationManager)
        .userDetailsService(userDetailsService);
    }
}
```

# Web security configuration

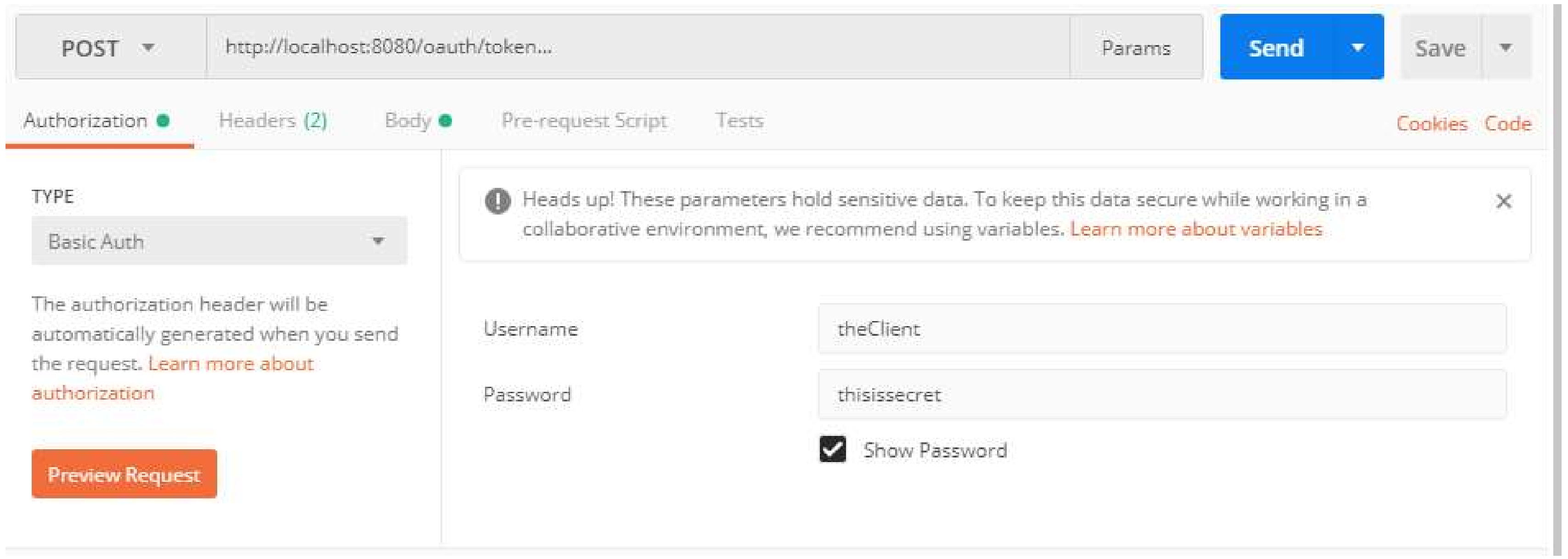
```java
@Configuration
public class WebSecurityConfigurer extends WebSecurityConfigurerAdapter {
  @Override
  @Bean
  public AuthenticationManager authenticationManagerBean() throws Exception {
    return super.authenticationManagerBean();
  }

  @Override
  @Bean
  public UserDetailsService userDetailsServiceBean() throws Exception {
    return super.userDetailsServiceBean();
  }

  @Override
  protected void configure(AuthenticationManagerBuilder auth) throws Exception {
    auth.inMemoryAuthentication()
        .withUser("john").password("{noop}password1").roles("USER")
        .and()
        .withUser("frank").password("{noop}password2").roles("USER", "MANAGER");
  }
}
```

# The configuration

**application.yml**

```
server:
  port: 8080
```

# Retrieve a token



http://localhost:8080/oauth/token...

TYPE

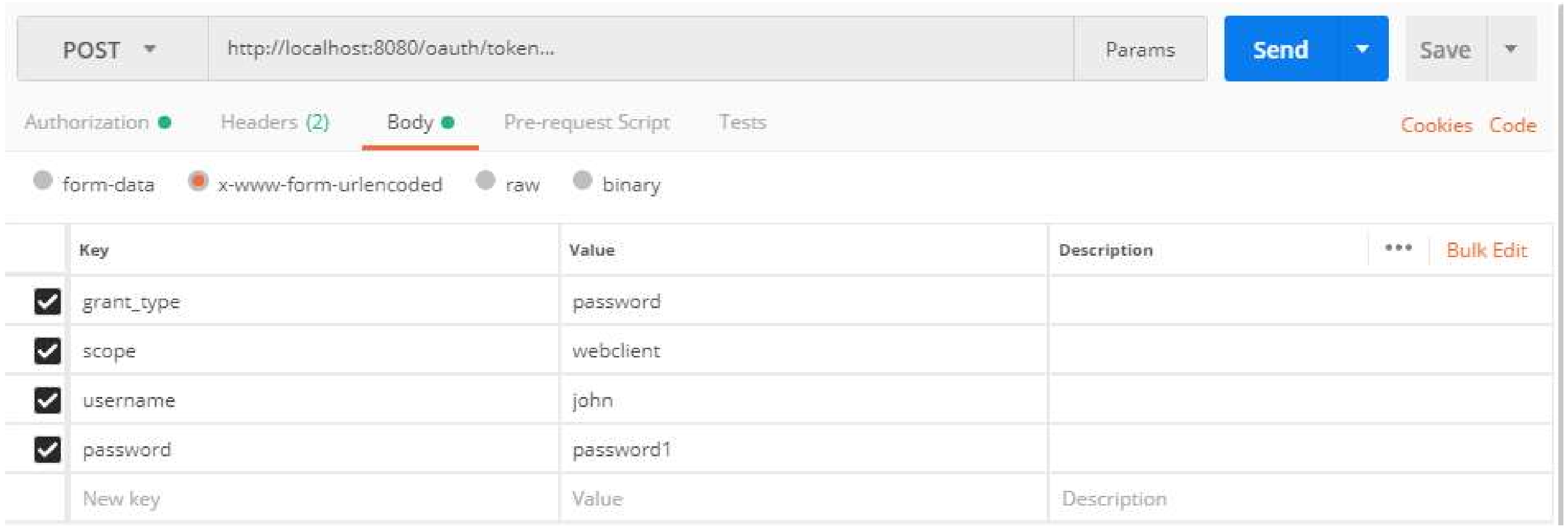
Basic Auth

The authorization header will be automatically generated when you send the request. Learn more about authorization

Heads up! These parameters hold sensitive data. To keep this data secure while working in a collaborative environment, we recommend using variables. Learn more about variables

| | |
|---|---|
| Username | theClient |
| Password | thisissecret |

✓ Show Password

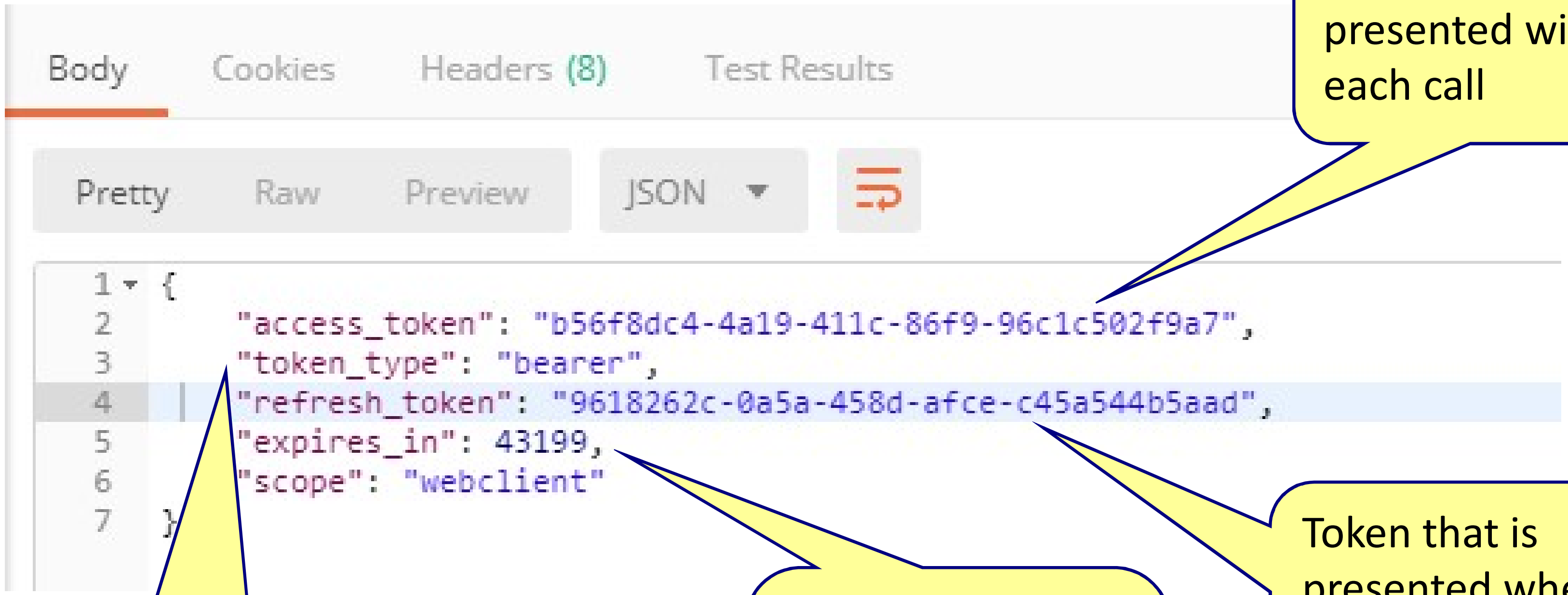


http://localhost:8080/oauth/token...

form-data [✓ x-www-form-urlencoded] raw binary

| | Key | Value | Description |
|---|---|---|---|
| ✓ | grant_type | password | |
| ✓ | scope | webclient | |
| ✓ | username | john | |
| ✓ | password | password1 | |
| | New key | Value | Description |

# Returned payload



```
{
    "access_token": "b56f8dc4-4a19-411c-86f9-96c1c502f9a7",
    "token_type": "bearer",
    "refresh_token": "9618262c-0a5a-458d-afce-c45a544b5aad",
    "expires_in": 43199,
    "scope": "webclient"
}
```

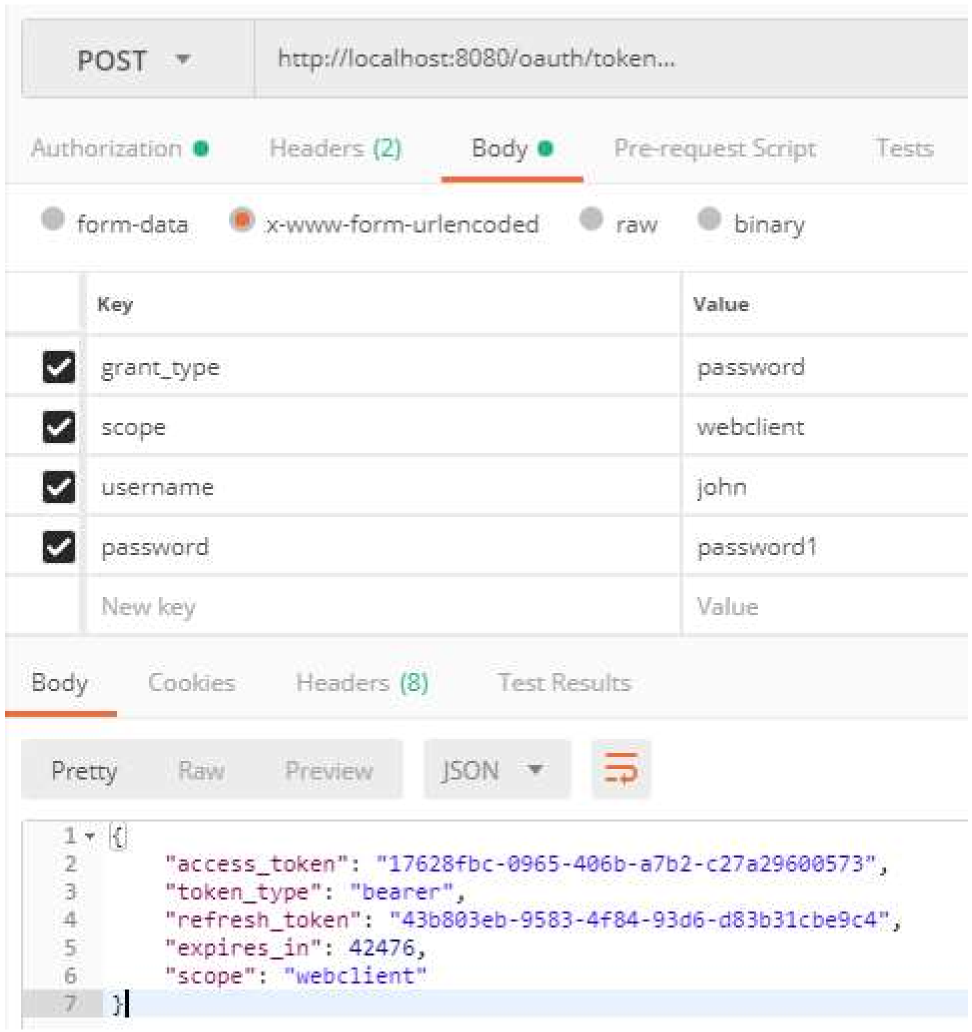
Access_token is presented with each call

Oauth supports multiple token types

Number of seconds before the token expires Spring default value is 12 hours

Token that is presented when you refresh an expired token

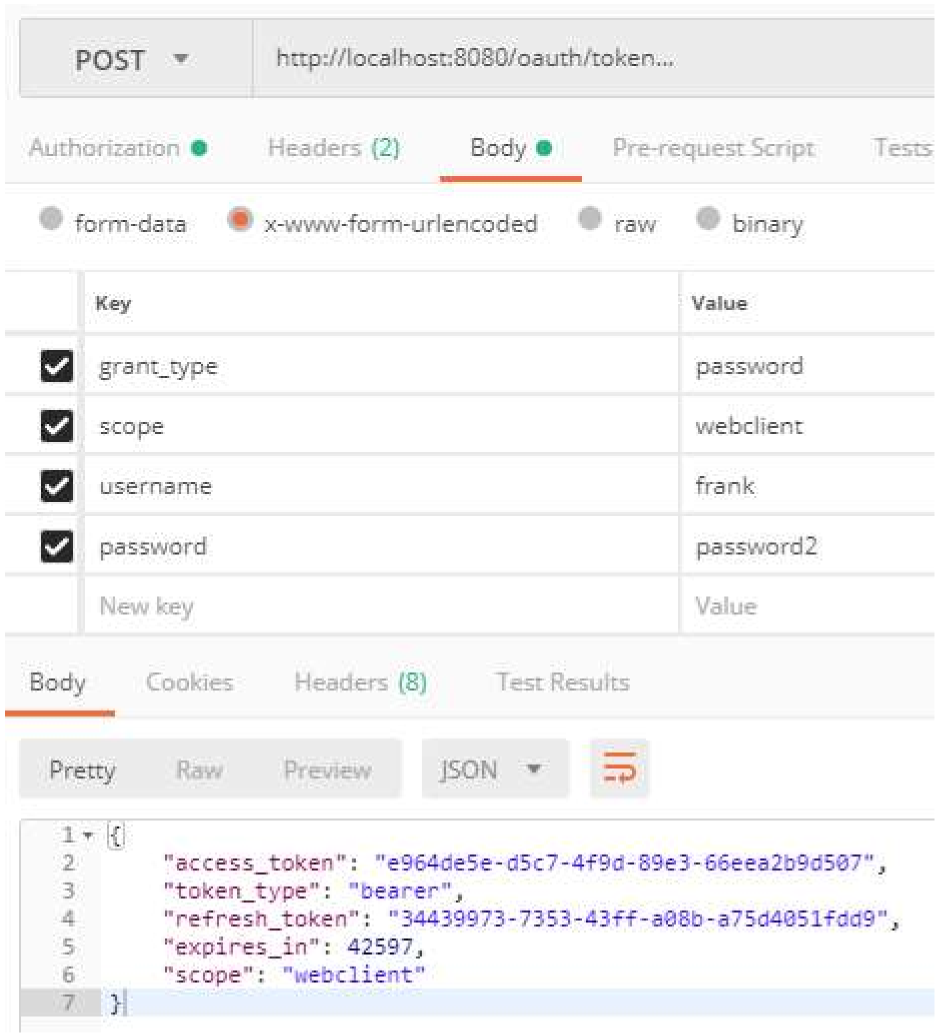
# Token for John and Frank



http://localhost:8080/oauth/token...

| Key | Value |
|---|---|
| ✓ grant_type | password |
| ✓ scope | webclient |
| ✓ username | john |
| ✓ password | password1 |
| New key | Value |

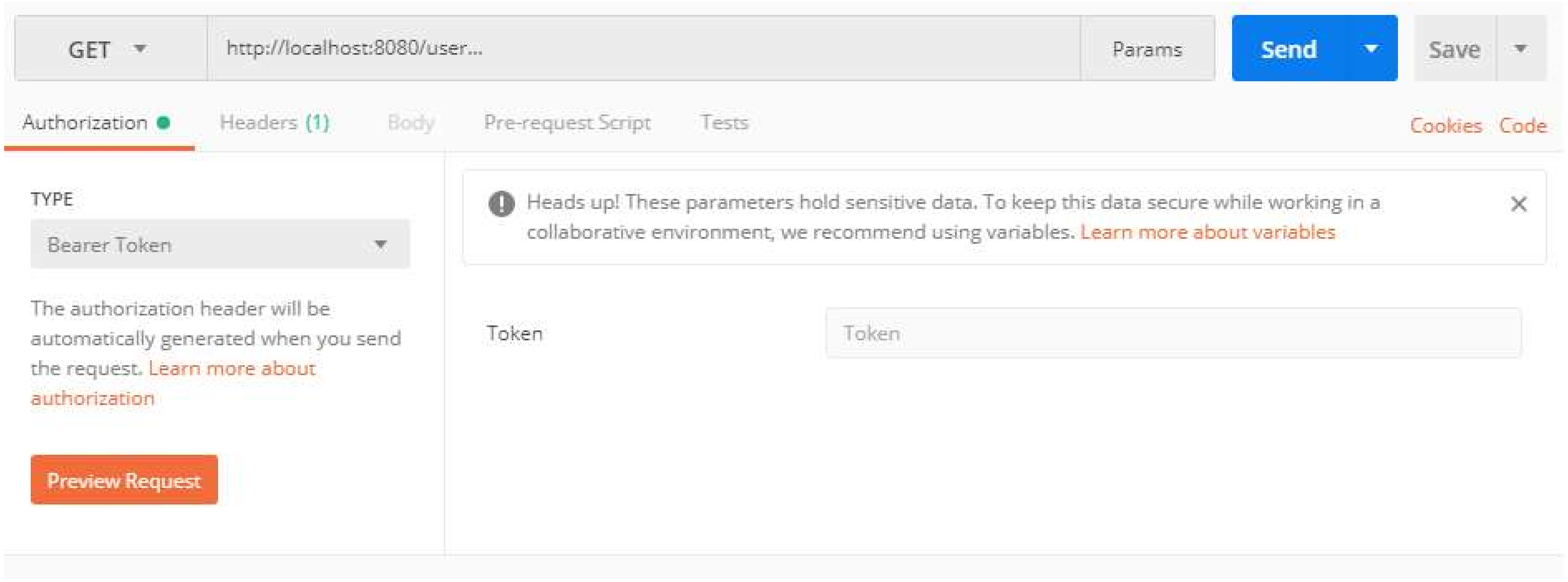
```
{
    "access_token": "17628fbc-0965-406b-a7b2-c27a29600573",
    "token_type": "bearer",
    "refresh_token": "43b803eb-9583-4f84-93d6-d83b31cbe9c4",
    "expires_in": 42476,
    "scope": "webclient"
}
```



http://localhost:8080/oauth/token...

| Key | Value |
|---|---|
| ✓ grant_type | password |
| ✓ scope | webclient |
| ✓ username | frank |
| ✓ password | password2 |
| New key | Value |

```
{
    "access_token": "e964de5e-d5c7-4f9d-89e3-66eea2b9d507",
    "token_type": "bearer",
    "refresh_token": "34439973-7353-43ff-a08b-a75d4051fdd9",
    "expires_in": 42597,
    "scope": "webclient"
}
```



GET http://localhost:8080/user... Params Send Save

Authorization ● | Headers (1) | Body | Pre-request Script | Tests | Cookies | Code

TYPE

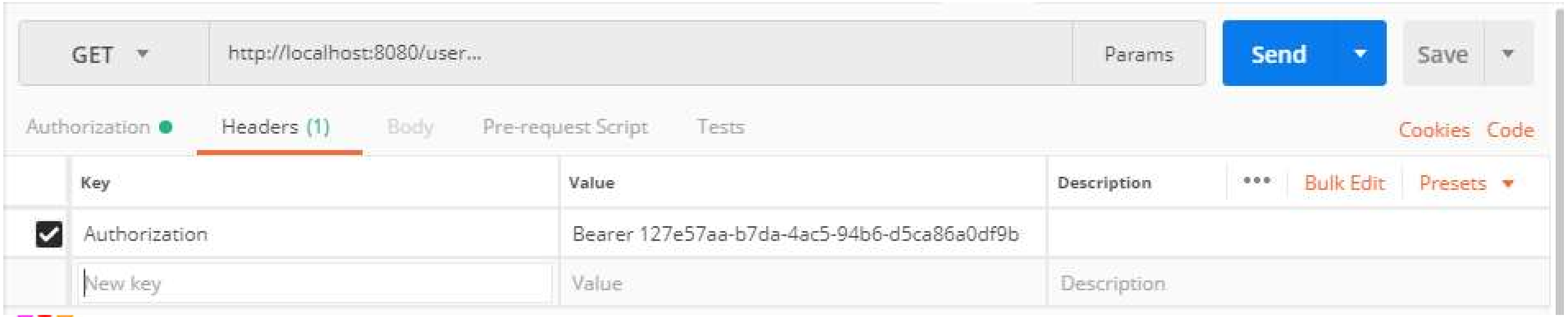
Bearer Token

The authorization header will be automatically generated when you send the request. Learn more about authorization

Preview Request

Heads up! These parameters hold sensitive data. To keep this data secure while working in a collaborative environment, we recommend using variables. Learn more about variables

Token: Token

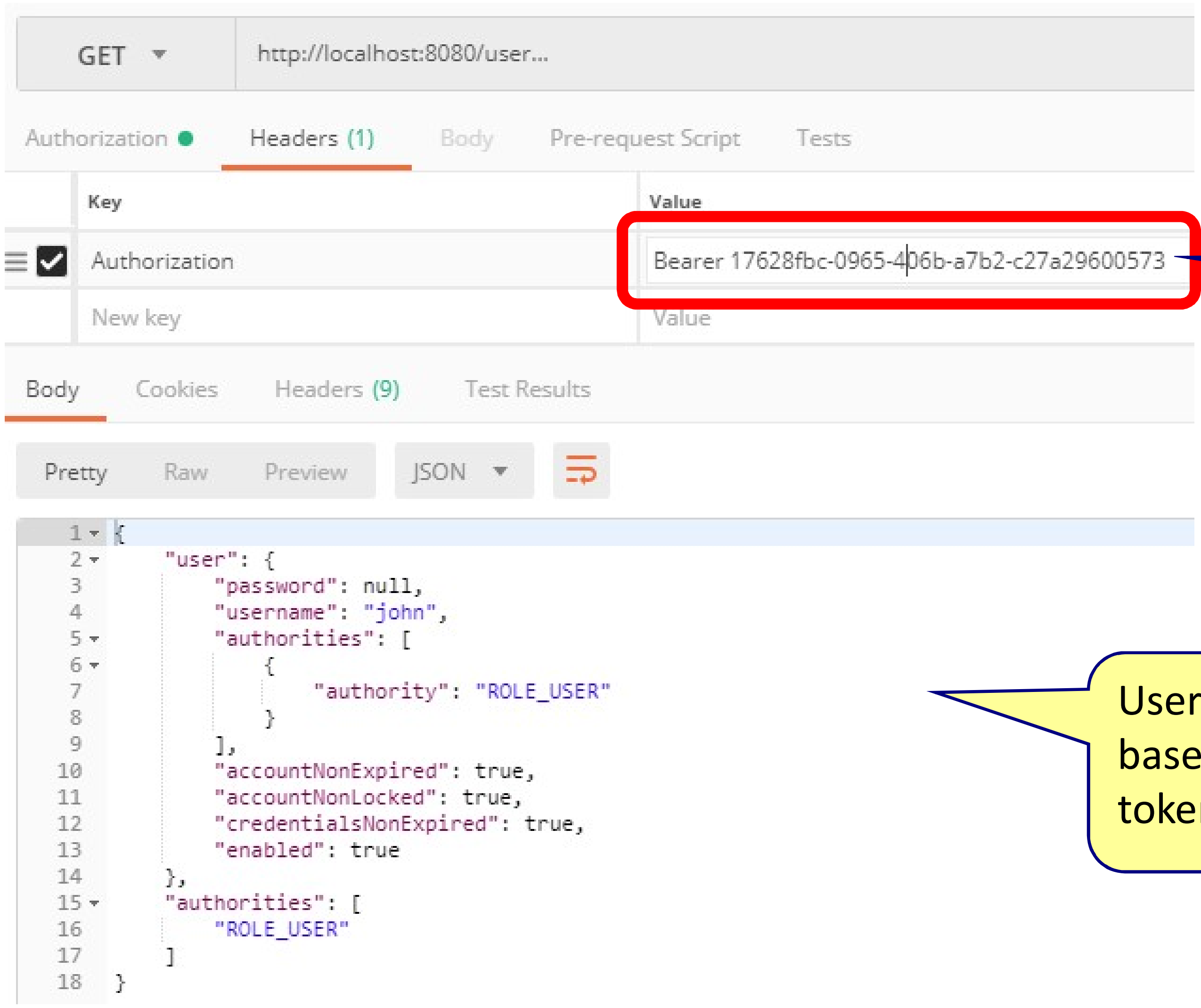


GET http://localhost:8080/user... Params Send Save

Authorization ● | Headers (1) | Body | Pre-request Script | Tests | Cookies | Code

| | Key | Value | Description |
|---|---|---|---|
| ✓ | Authorization | Bearer 127e57aa-b7da-4ac5-94b6-d5ca86a0df9b | |
| | New key | Value | Description |

# Get user information



http://localhost:8080/user...

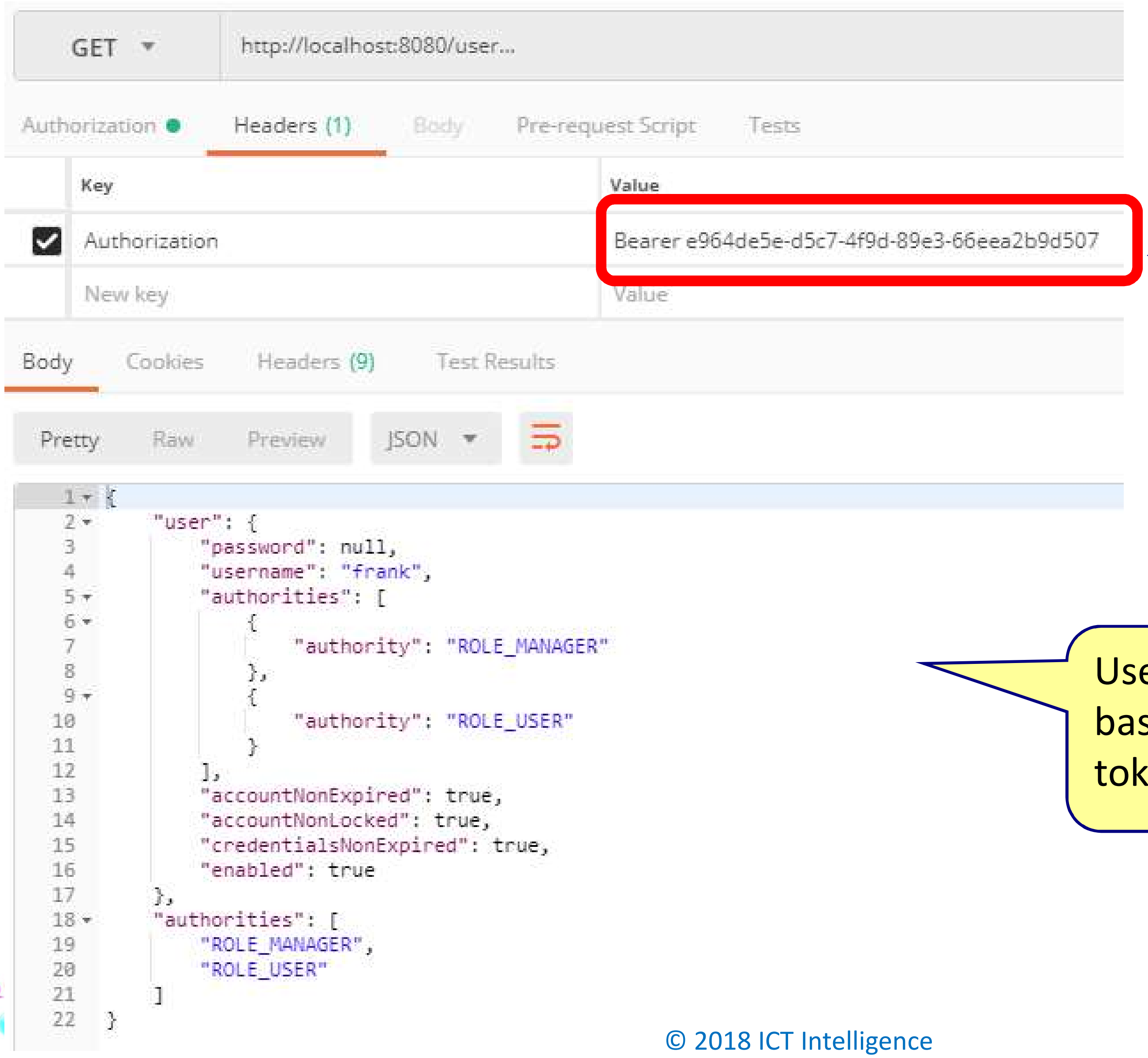
| Key | Value |
|---|---|
| Authorization | Bearer 17628fbc-0965-406b-a7b2-c27a29600573 |

```
{
    "user": {
        "password": null,
        "username": "john",
        "authorities": [
            {
                "authority": "ROLE_USER"
            }
        ],
        "accountNonExpired": true,
        "accountNonLocked": true,
        "credentialsNonExpired": true,
        "enabled": true
    },
    "authorities": [
        "ROLE_USER"
    ]
}
```

Token for John

User information based on OAuth token

# Get user information



```
http://localhost:8080/user...
```

Authorization: Bearer e964de5e-d5c7-4f9d-89e3-66eea2b9d507

```
{
    "user": {
        "password": null,
        "username": "frank",
        "authorities": [
            {
                "authority": "ROLE_MANAGER"
            },
            {
                "authority": "ROLE_USER"
            }
        ],
        "accountNonExpired": true,
        "accountNonLocked": true,
        "credentialsNonExpired": true,
        "enabled": true
    },
    "authorities": [
        "ROLE_MANAGER",
        "ROLE_USER"
    ]
}
```

Token for Frank

User information based on OAuth token

## A SECURE APPLICATION

# A secure application

```
@SpringBootApplication
public class SecureServiceAApplication {

  public static void main(String[] args) {
    SpringApplication.run(SecureServiceAApplication.class, args);
  }
}
```

```
@Configuration
@EnableResourceServer
public class ResourceServerConfig extends ResourceServerConfigurerAdapter {
    @Override
    public void configure(HttpSecurity http) throws Exception {
        http
                .authorizeRequests()
                .antMatchers("/name").permitAll()
                .antMatchers("/salary").hasRole("MANAGER")
                .antMatchers("/phone").hasRole("USER")
                .anyRequest()
                .authenticated();
    }
}
```

# The controller

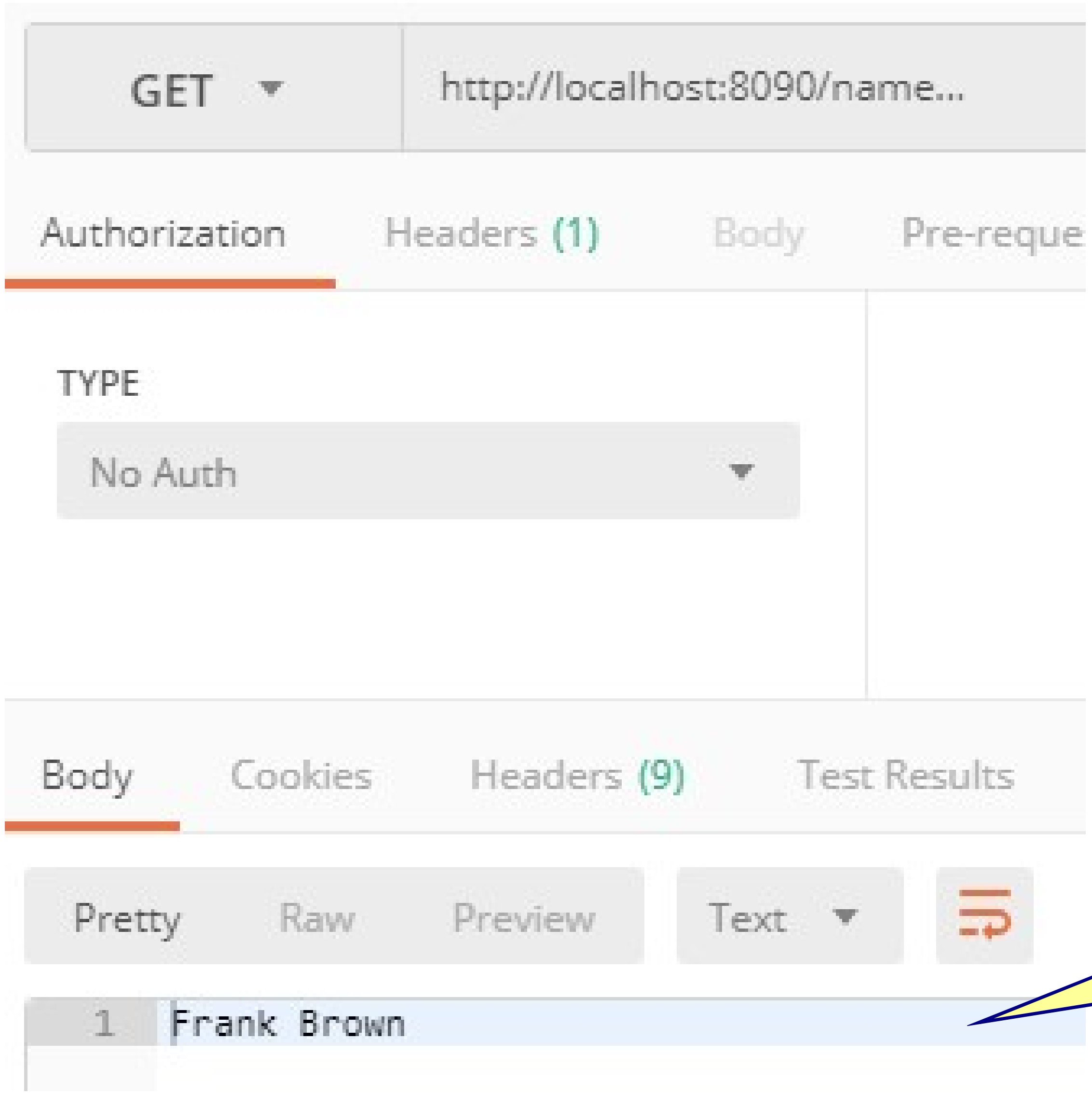
```
@RestController
public class Controller {

  @GetMapping("/name")
  public String getName() {
    return "Frank Brown";
  }

  @GetMapping("/salary")
  public String getGetSalary() {
    return "95.000";
  }

  @GetMapping("/phone")
  public String getPhone() {
    return "645322899";
  }
}
```

# The configuration

**application.yml**

```
server:
  port: 8090

security:
  oauth2:
    client:
      accessTokenUri: http://localhost:8080/oauth/token
      userAuthorizationUri: http://localhost:8080/oauth/authorize
      clientId: theClient
      clientSecret: thisissecret
    resource:
      userInfoUri: http://localhost:8080/user
```

# Get the user name


GET http://localhost:8090/name...

Authorization | Headers (1) | Body | Pre-reque

TYPE

No Auth

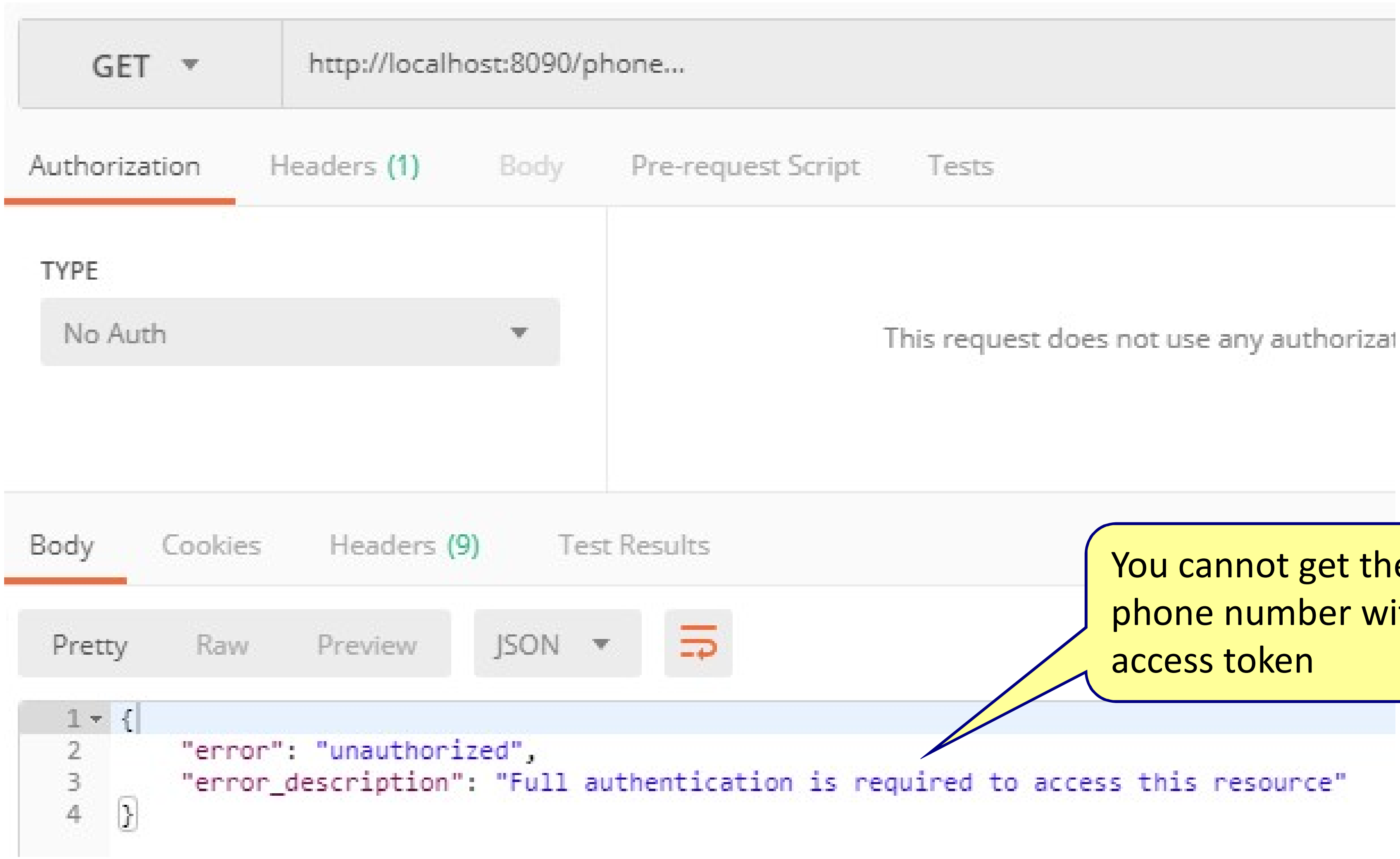
Body | Cookies | Headers (9) | Test Results

Pretty | Raw | Preview | Text

```
Frank Brown
```

Everyone can get the phone number without authorization

# Get the phone number



http://localhost:8090/phone...

TYPE

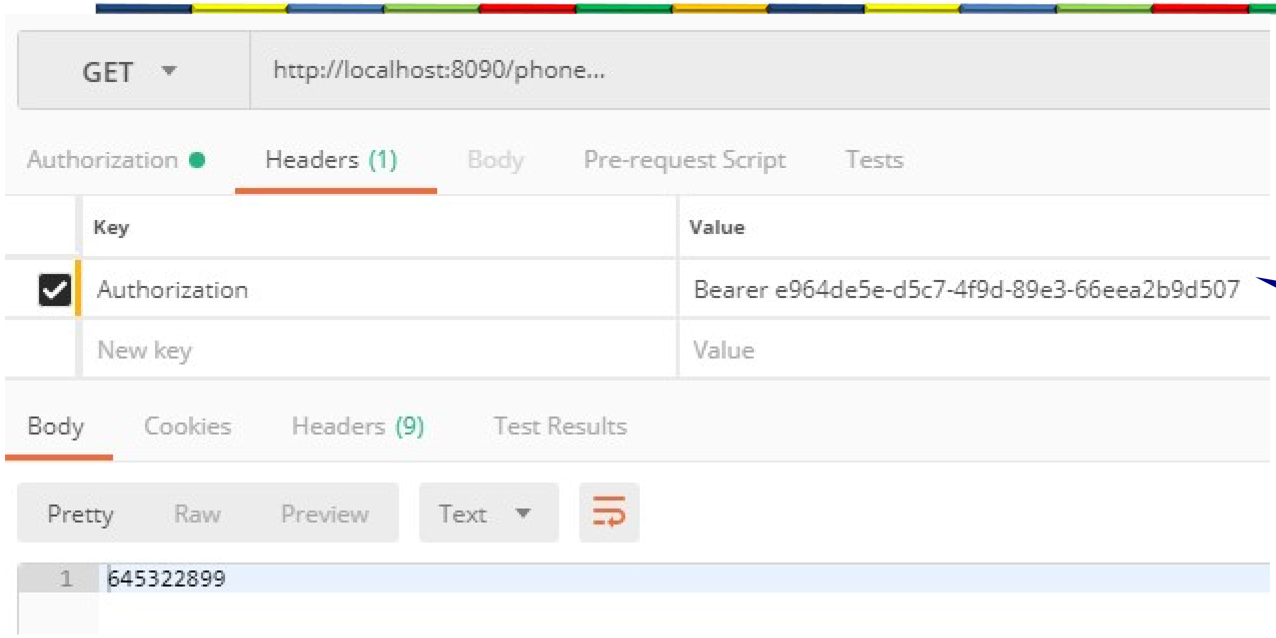
No Auth

This request does not use any authorizat

```
{
    "error": "unauthorized",
    "error_description": "Full authentication is required to access this resource"
}
```

You cannot get the phone number without access token

# Get the phone number



GET http://localhost:8090/phone...

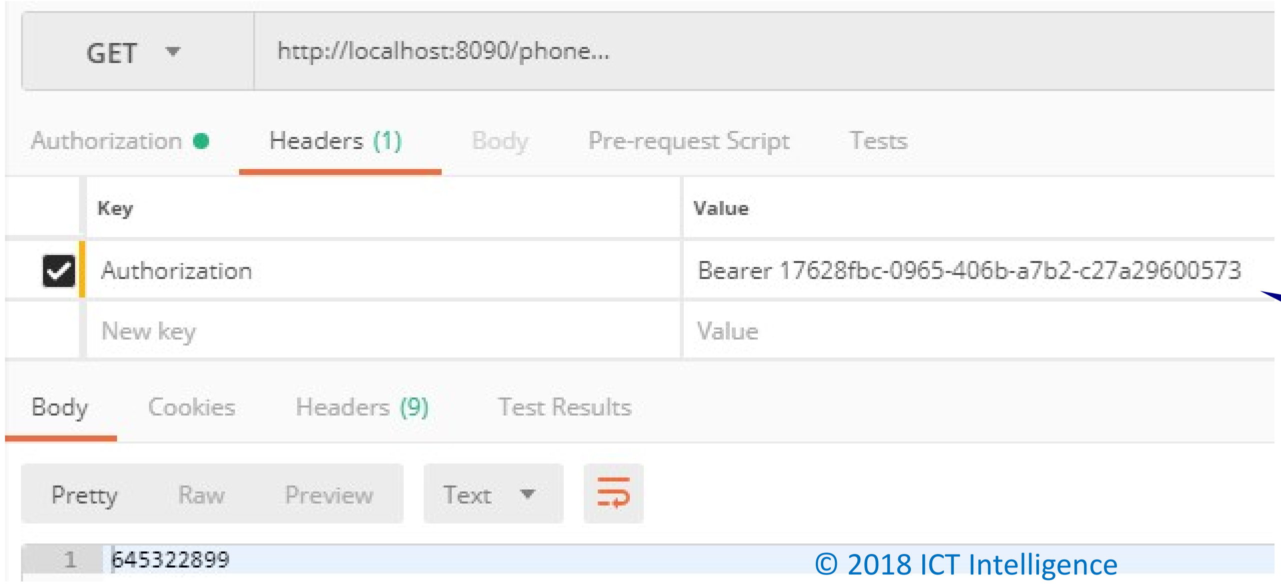
Authorization ● | Headers (1) | Body | Pre-request Script | Tests

| ✓ | Key | Value |
|---|---|---|
| ✓ | Authorization | Bearer e964de5e-d5c7-4f9d-89e3-66eea2b9d507 |
| | New key | Value |

Body | Cookies | Headers (9) | Test Results

Pretty | Raw | Preview | Text

```
645322899
```



GET http://localhost:8090/phone...

Authorization ● | Headers (1) | Body | Pre-request Script | Tests

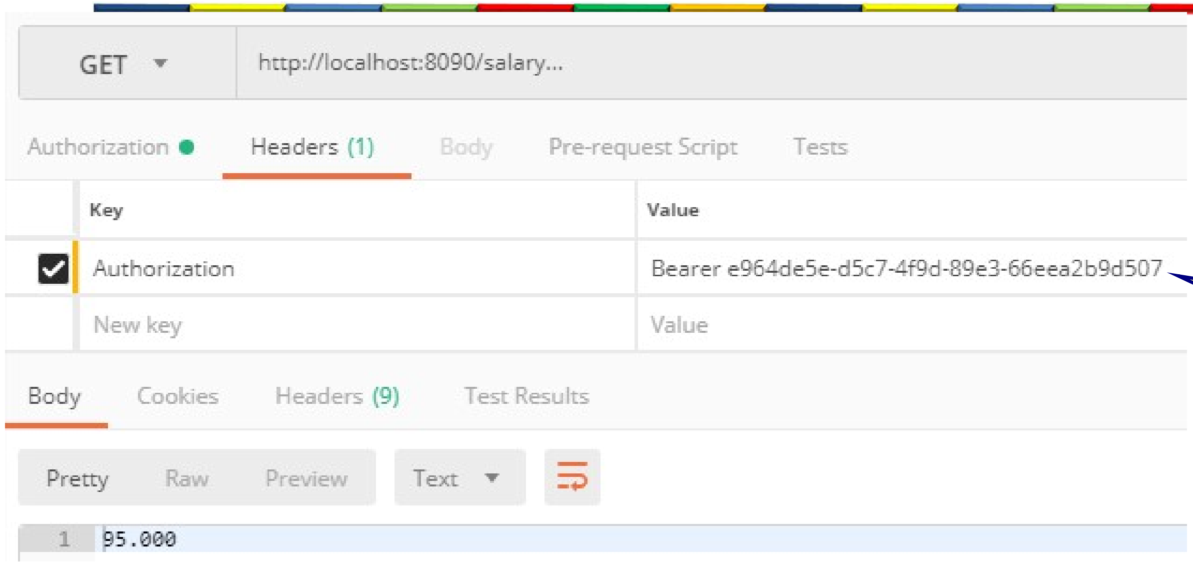
| ✓ | Key | Value |
|---|---|---|
| ✓ | Authorization | Bearer 17628fbc-0965-406b-a7b2-c27a29600573 |
| | New key | Value |

Body | Cookies | Headers (9) | Test Results

Pretty | Raw | Preview | Text

```
645322899
```

# Get the salary



GET http://localhost:8090/salary...

Authorization ● Headers (1) Body Pre-request Script Tests

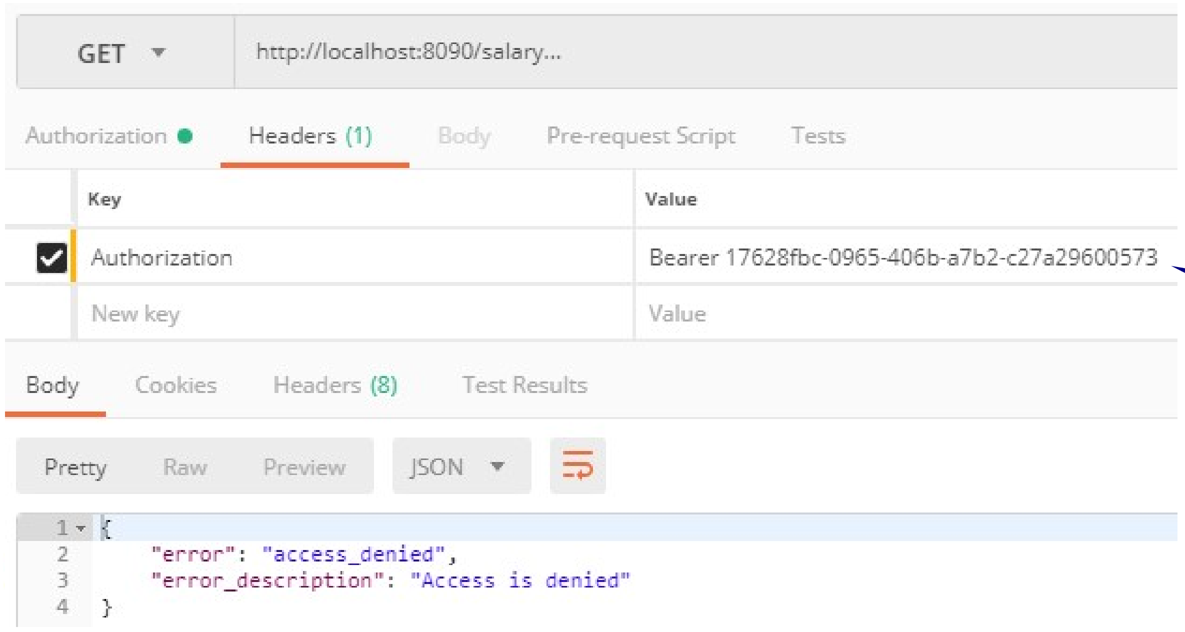
| Key | Value |
|---|---|
| ✓ Authorization | Bearer e964de5e-d5c7-4f9d-89e3-66eea2b9d507 |
| New key | Value |

Body Cookies Headers (9) Test Results

Pretty Raw Preview Text

```
95.000
```

Frank can get the salary (role=MANAGER)



GET http://localhost:8090/salary...

Authorization ● Headers (1) Body Pre-request Script Tests

| Key | Value |
|---|---|
| ✓ Authorization | Bearer 17628fbc-0965-406b-a7b2-c27a29600573 |
| New key | Value |

Body Cookies Headers (8) Test Results

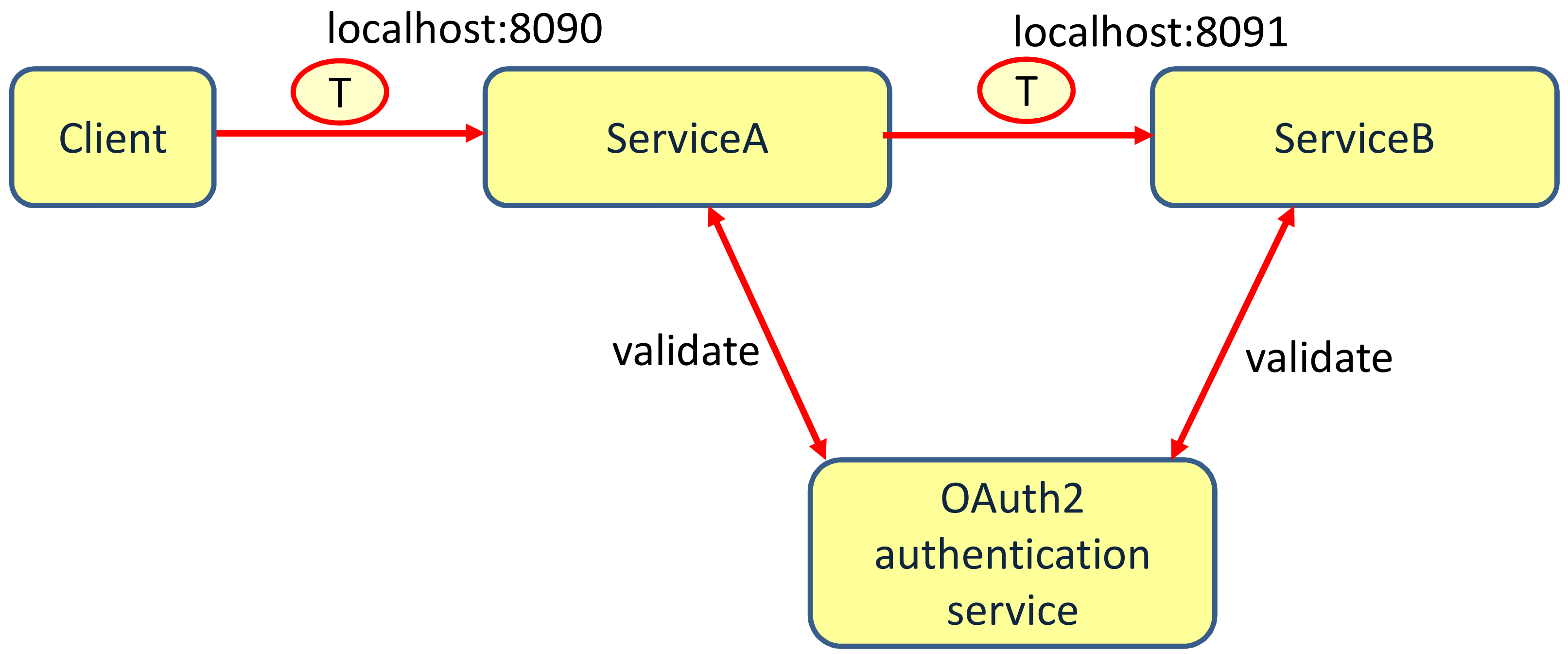
Pretty Raw Preview JSON

```
{
    "error": "access_denied",
    "error_description": "Access is denied"
}
```

John cannot get the salary (role = USER)

## PROPAGATING THE TOKEN

# Propagate the token



# Secure application B

```java
@SpringBootApplication
public class SecureServiceBApplication {

  public static void main(String[] args) {
    SpringApplication.run(SecureServiceBApplication.class, args);
  }
}
```

```java
@Configuration
@EnableResourceServer
public class ResourceServerConfig extends ResourceServerConfigurerAdapter {
    @Override
    public void configure(HttpSecurity http) throws Exception {
        http
                .authorizeRequests()
                .antMatchers("/publicinfo").permitAll()
                .antMatchers("/userinfo").hasRole("USER")
                .antMatchers("/managerinfo").hasRole("MANAGER")
                .anyRequest()
                .authenticated();
    }
}
```

## The controller

```
@RestController
public class Controller {

  @GetMapping("/publicinfo")
  public String getPublicInfo() {
    return "This is public info";
  }

  @GetMapping("/userinfo")
  public String getUserInfo() {
    return "This info is for users";
  }

  @GetMapping("/managerinfo")
  public String getManagerInfo() {
    return "This info is for managers";
  }
}
```

## Secure application A

```
@SpringBootApplication
public class SecureServiceAApplication {

  public static void main(String[] args) {
    SpringApplication.run(SecureServiceAApplication.class, args);
  }

  @Bean
  public OAuth2RestTemplate oAuth2RestTemplate(OAuth2ClientContext
        oAuth2ClientContext, OAuth2ProtectedResourceDetails details) {
    return new OAuth2RestTemplate(details, oAuth2ClientContext);
  }
}
```

## Secure application A

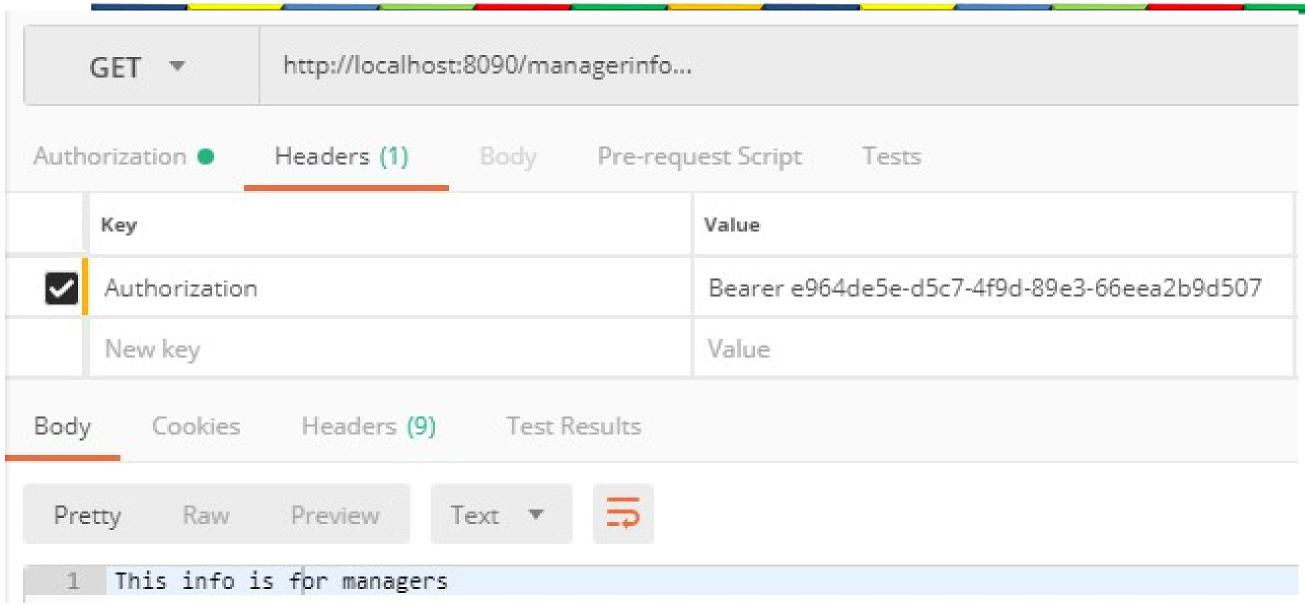
```
@Configuration
@EnableResourceServer
public class ResourceServerConfig extends ResourceServerConfigurerAdapter
{
    @Override
    public void configure(HttpSecurity http) throws Exception {
        http
                .authorizeRequests()
                .antMatchers("/name").permitAll()
                .antMatchers("/salary").hasRole("MANAGER")
                .antMatchers("/phone").hasRole("USER")
                .antMatchers("/publicinfo").permitAll()
                .antMatchers("/managerinfo").hasRole("MANAGER")
                .antMatchers("/userinfo").hasRole("USER")
                .anyRequest()
                .authenticated();
    }
}
```

## The controller2

```
@RestController
public class Controller2 {
  @Autowired
  OAuth2RestTemplate restTemplate;

  @GetMapping("/publicinfo")
  public String getPublicInfo() {
    return restTemplate.getForObject("http://localhost:8091/publicinfo", String.class);
  }

  @GetMapping("/userinfo")
  public String getUserInfo() {
    return restTemplate.getForObject("http://localhost:8091/userinfo", String.class);
  }

  @GetMapping("/managerinfo")
  public String getManagerInfo() {
    return restTemplate.getForObject("http://localhost:8091/managerinfo", String.class);
  }
}
```

OAuth2RestTemplate

# One services calls another service



GET http://localhost:8090/managerinfo...

Authorization ● Headers (1) Body Pre-request Script Tests

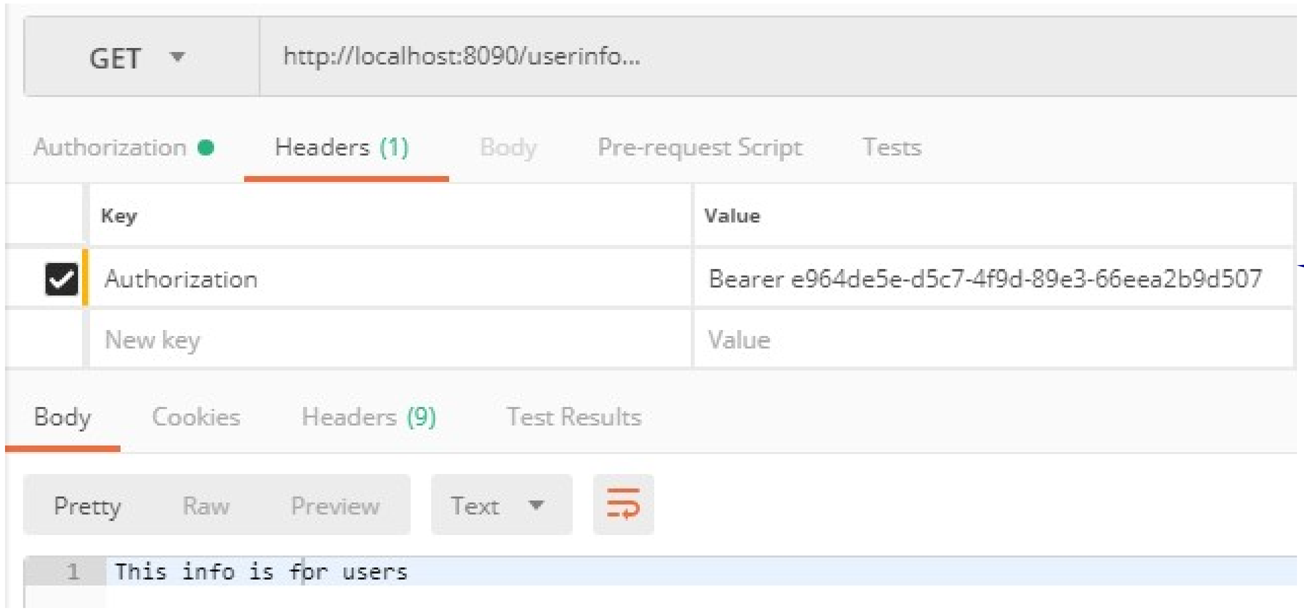
| Key | Value |
|---|---|
| ✓ Authorization | Bearer e964de5e-d5c7-4f9d-89e3-66eea2b9d507 |
| New key | Value |

Body Cookies Headers (9) Test Results

Pretty Raw Preview Text

```
This info is for managers
```

Frank can get all info



GET http://localhost:8090/userinfo...

Authorization ● Headers (1) Body Pre-request Script Tests

| Key | Value |
|---|---|
| ✓ Authorization | Bearer e964de5e-d5c7-4f9d-89e3-66eea2b9d507 |
| New key | Value |

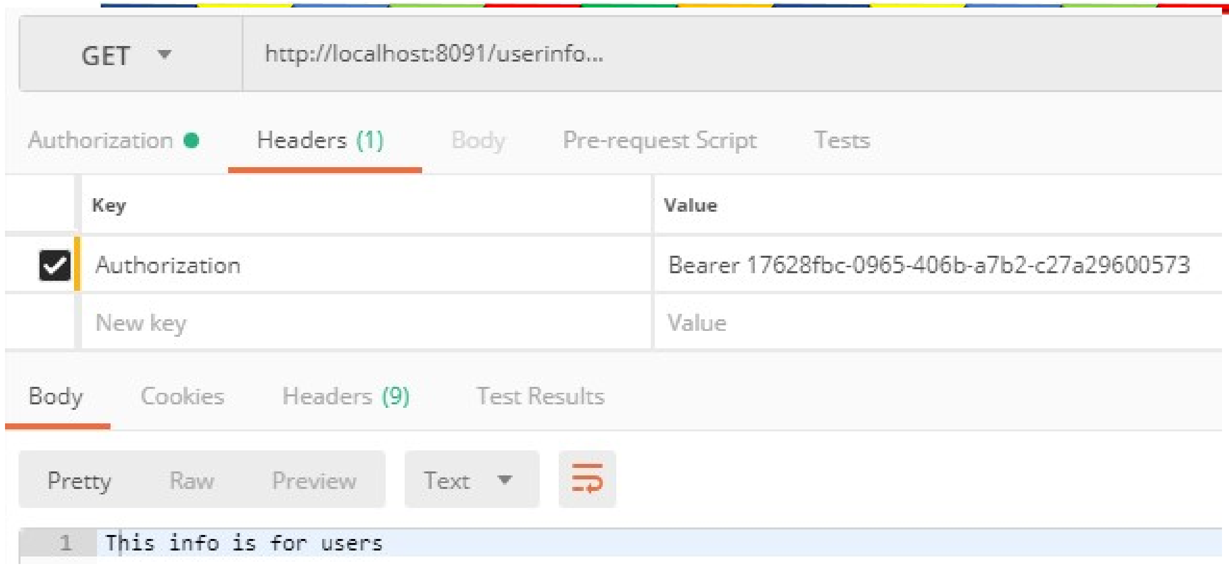
Body Cookies Headers (9) Test Results

Pretty Raw Preview Text

```
This info is for users
```

Frank can get all info

# One services calls another service



GET http://localhost:8091/userinfo...

Authorization ● | Headers (1) | Body | Pre-request Script | Tests

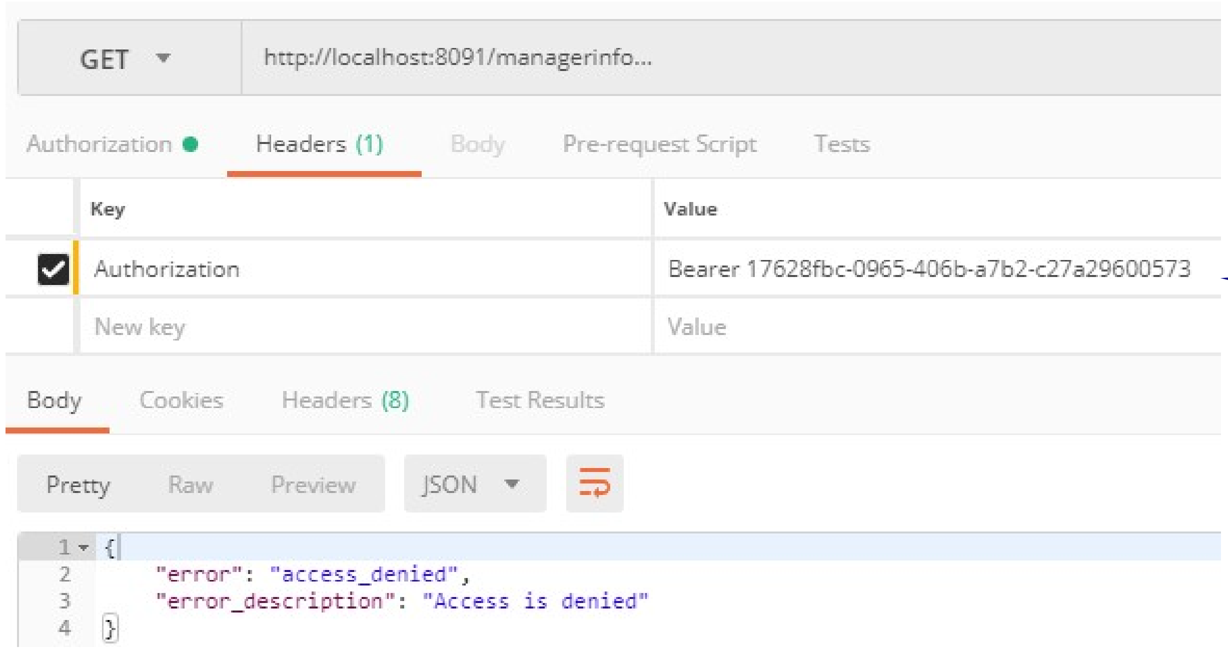
| | Key | Value |
|---|---|---|
| ✓ | Authorization | Bearer 17628fbc-0965-406b-a7b2-c27a29600573 |
| | New key | Value |

Body | Cookies | Headers (9) | Test Results

Pretty | Raw | Preview | Text

```
This info is for users
```

John can get user info



GET http://localhost:8091/managerinfo...

Authorization ● | Headers (1) | Body | Pre-request Script | Tests

| | Key | Value |
|---|---|---|
| ✓ | Authorization | Bearer 17628fbc-0965-406b-a7b2-c27a29600573 |
| | New key | Value |

Body | Cookies | Headers (8) | Test Results

Pretty | Raw | Preview | JSON

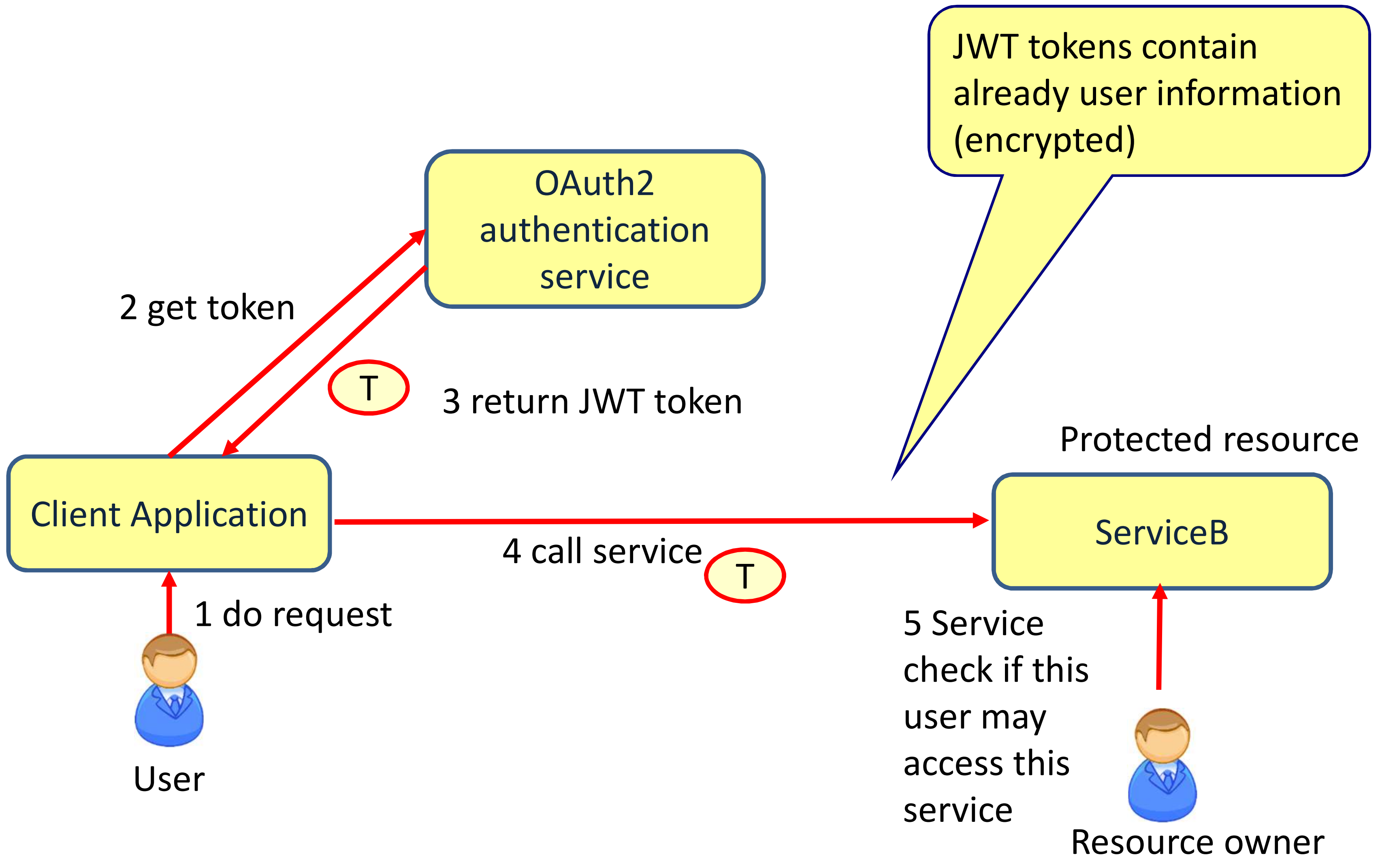
```
{
    "error": "access_denied",
    "error_description": "Access is denied"
}
```

John cannot get managerinfo

# JWT tokens

- Oauth2 is a token based authorization framework bus does provide a standard for tokens
- JWT (JavaScript Web Tokens) provides a standard structure for OAuth tokens
  - Small
  - Cryptographically signed
  - Self contained
  - Extensible

# JWT tokens


JWT tokens contain already user information (encrypted)
OAuth2 authentication service
2 get token
T
3 return JWT token
Protected resource
Client Application
ServiceB
4 call service
T
1 do request
5 Service check if this user may access this service
User
Resource owner

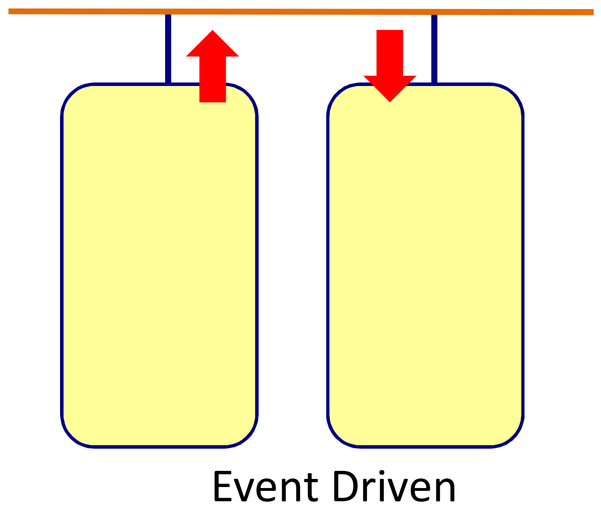
## Main point

- With OAuth2 and JWT we can make microservices secure without providing security credentials to the services, and without storing information into sessions

- By transcending into Pure Consciousness one gets access to all intelligence of creation.

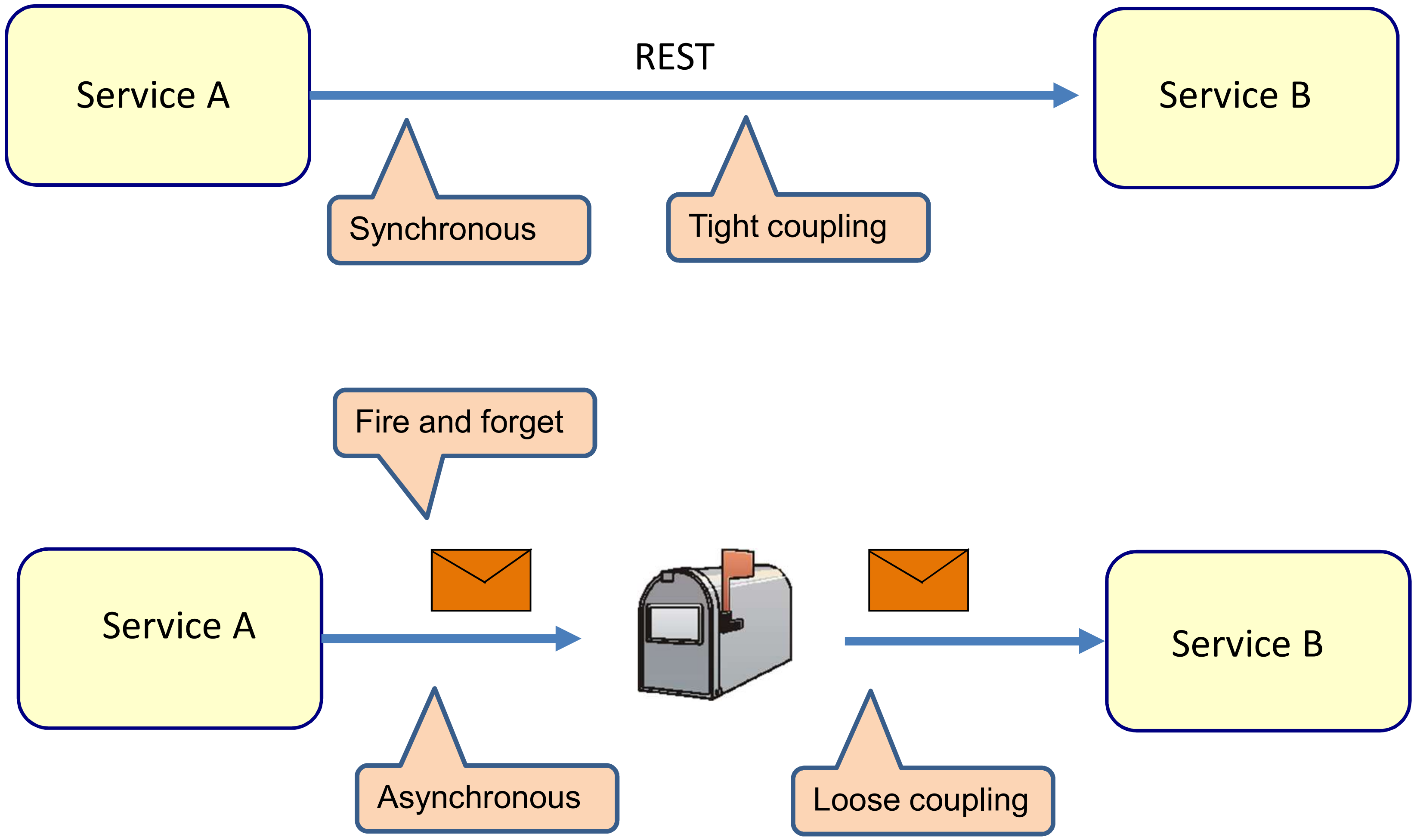
## EVENT DRIVEN ARCHITECTURE

# Event Driven Architecture (EDA)

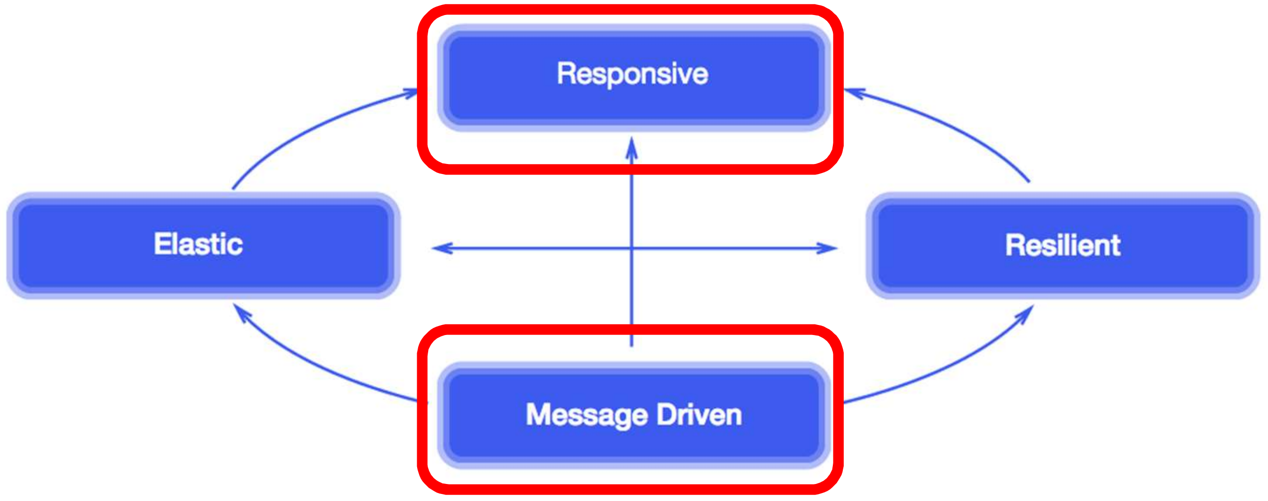


- Applications publish events
- Applications subscribe to events

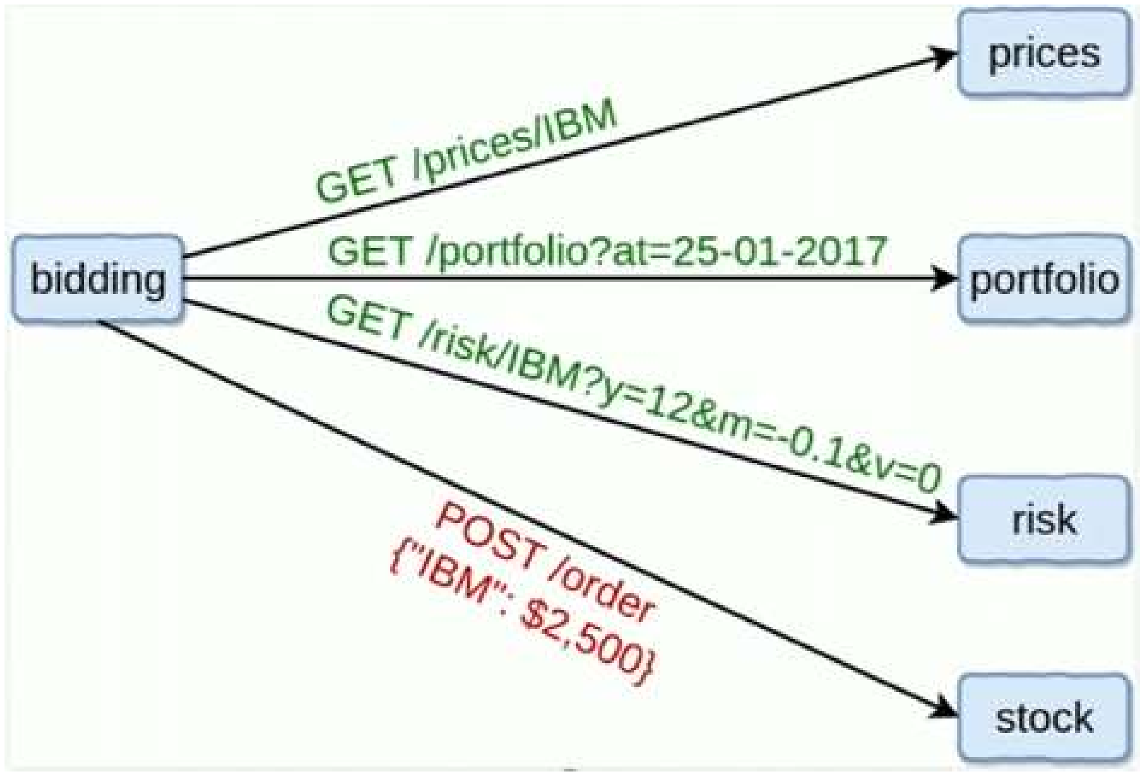
# REST vs. messaging



# Reactive applications

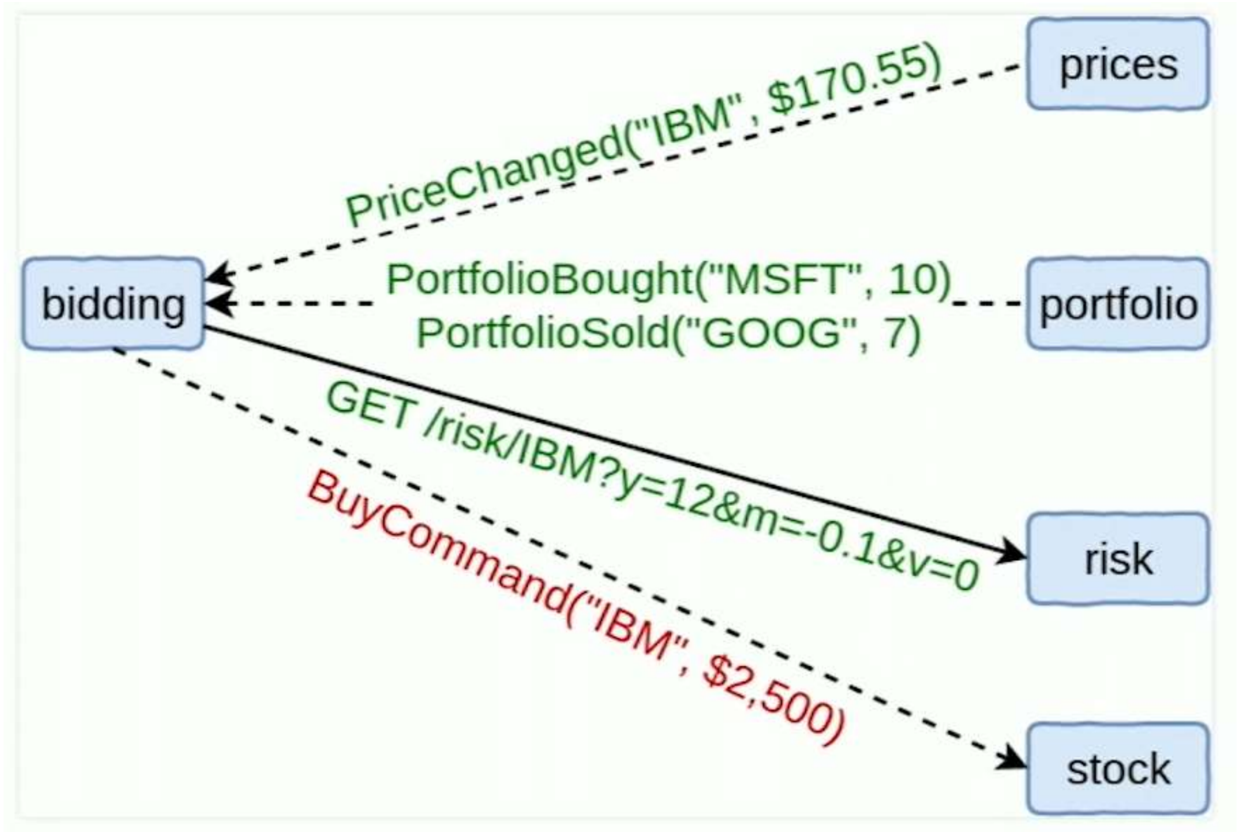


## Synchronous architecture



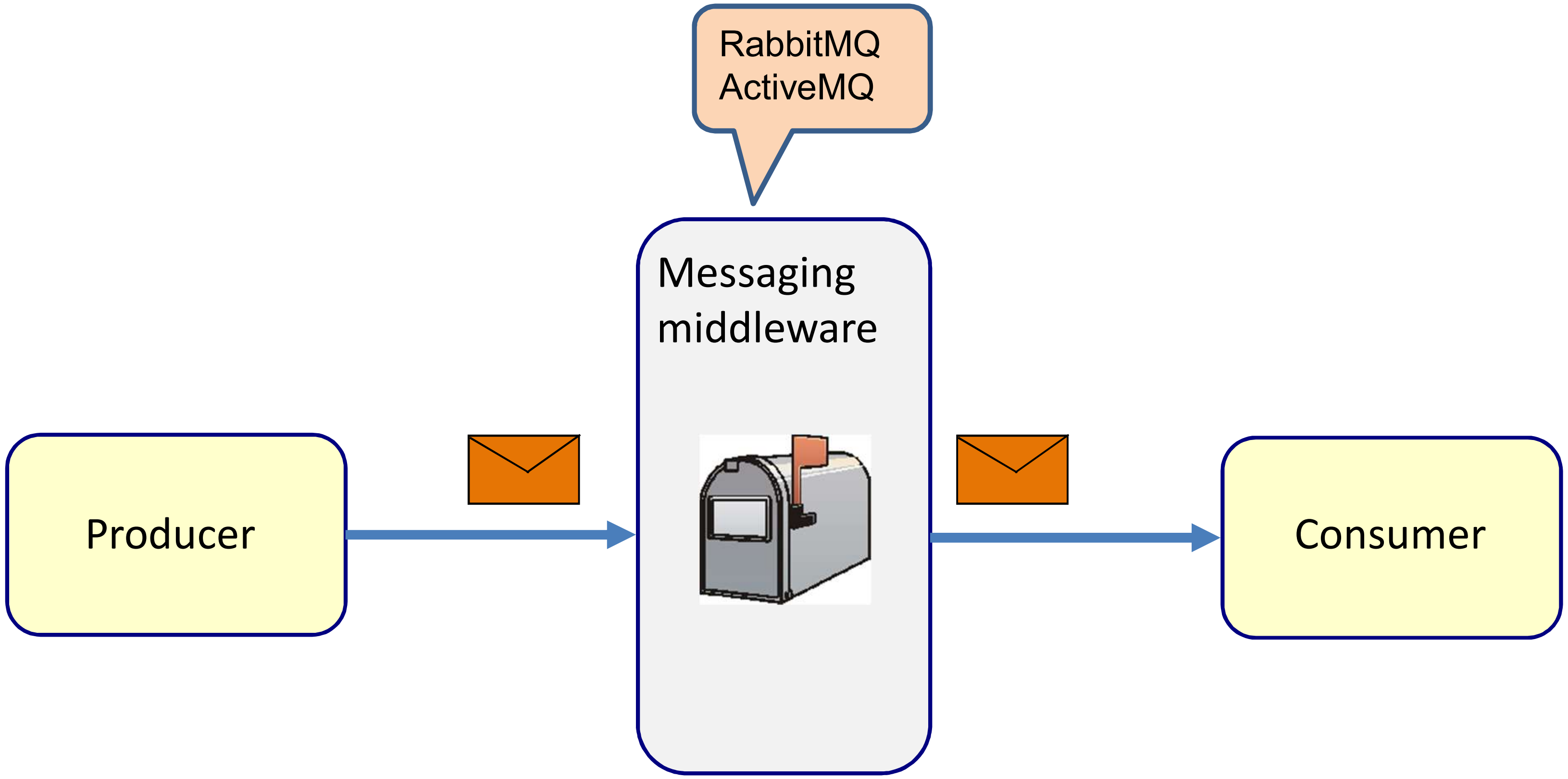
- Slow
- If one of the components go down, we cannot order
- Temporal coupling
  - Two services have to exist at the same time to perform some functionality

## Event driven architecture

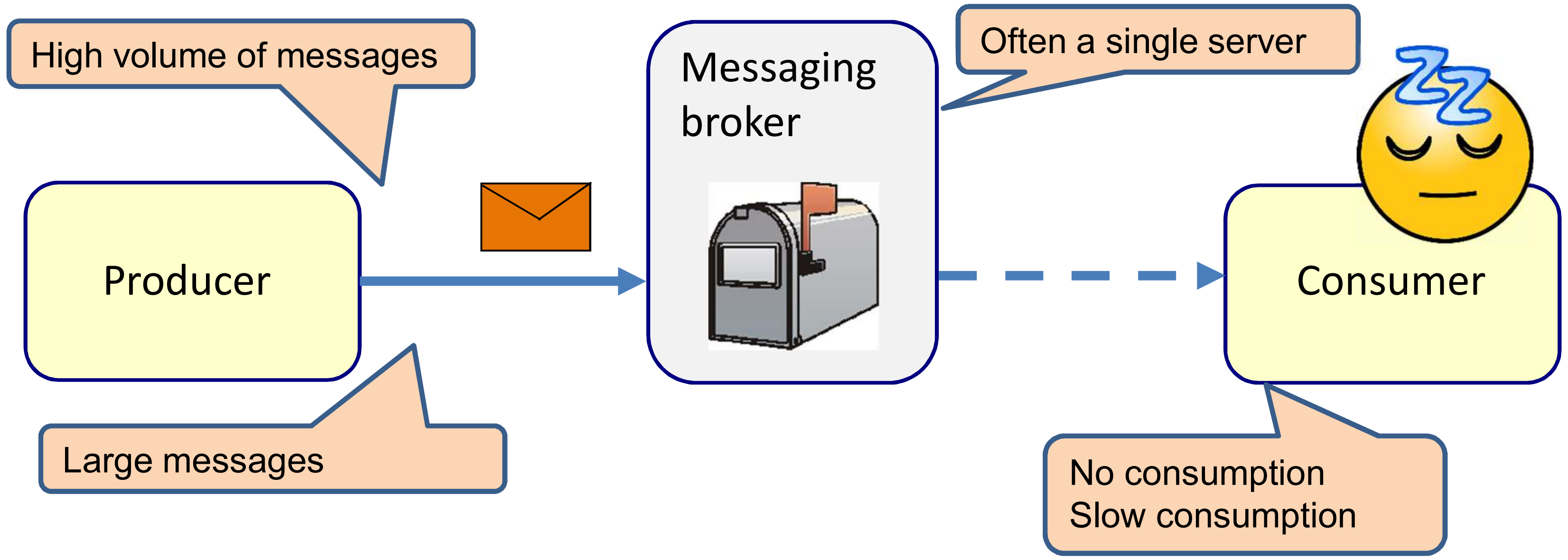

prices
PriceChanged("IBM", $170.55)
bidding
PortfolioBought("MSFT", 10)
PortfolioSold("GOOG", 7)
portfolio
GET /risk/IBM?y=12&m=-0.1&v=0
risk
BuyCommand("IBM", $2,500)
stock

- Much faster
- Less dependencies

# Traditional Messaging Systems

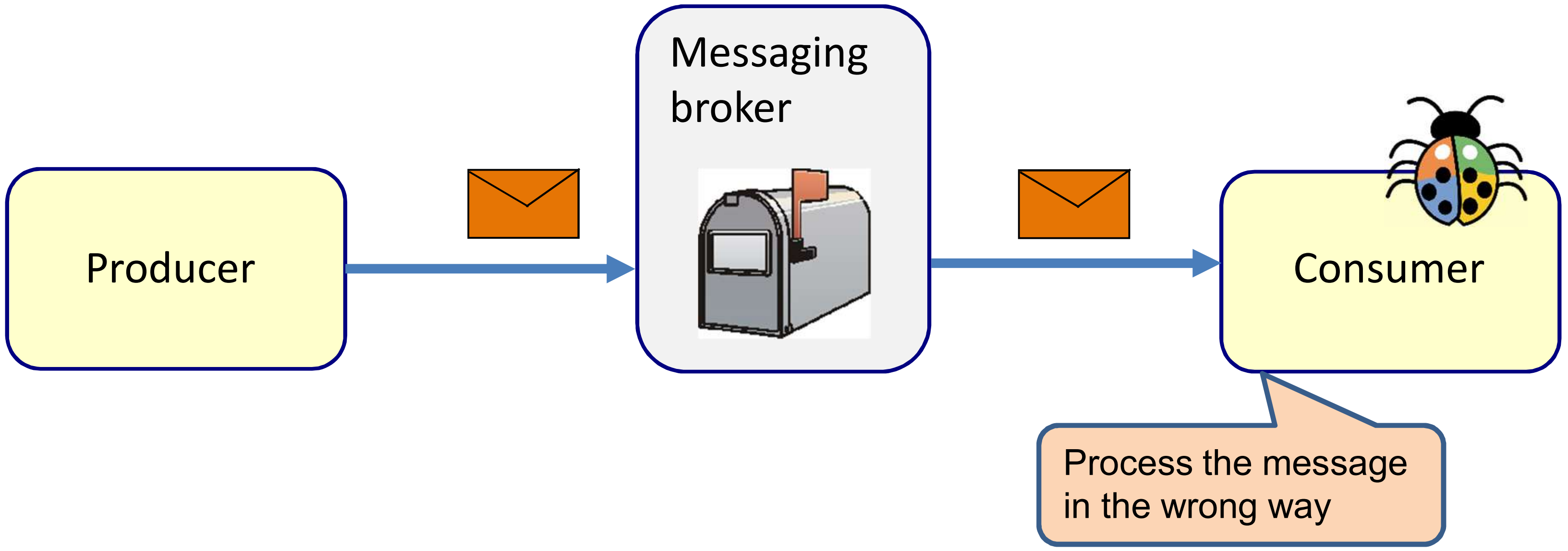


# Problems with traditional messaging middleware



- If the consumer is temporally not available (or very slow) the message middleware has to store the messages
  - This restricts the volume of messages and the size of the messages
  - Eventually the message broker will fail

# Problems with traditional messaging middleware



- If the consumer has a bug, and handles the messages incorrectly, then the messages are gone.
  - Not fault-tolerant

message broker

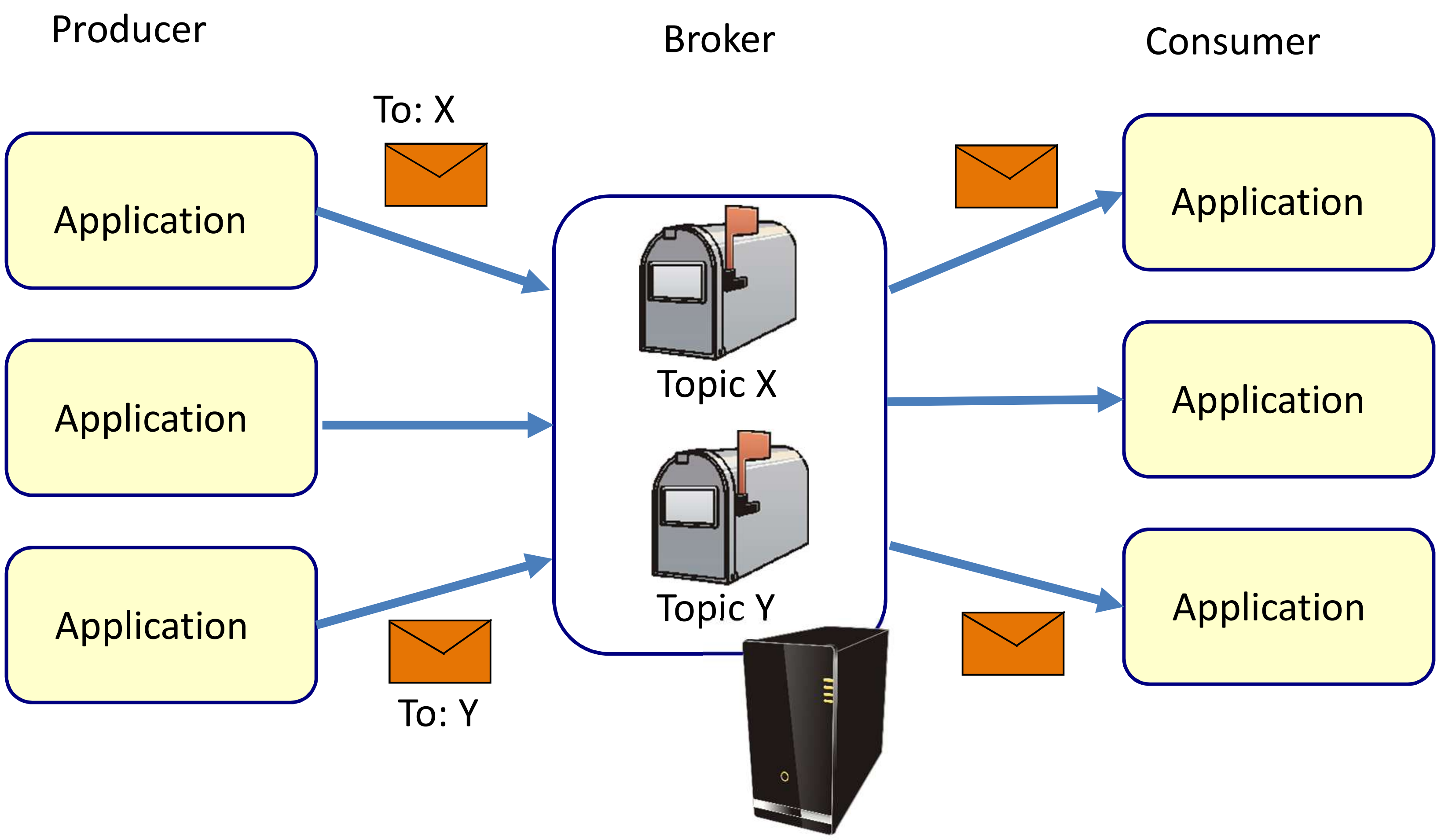
# Apache Kafka

- Created by Linked In
- Characteristics
  - High throughput
  - Distributed
  - Unlimited scalable
  - Fault-tolerant
    - Reliable and durable
  - Loosely coupled Producers and Consumers
  - Flexible publish-subscribe semantics

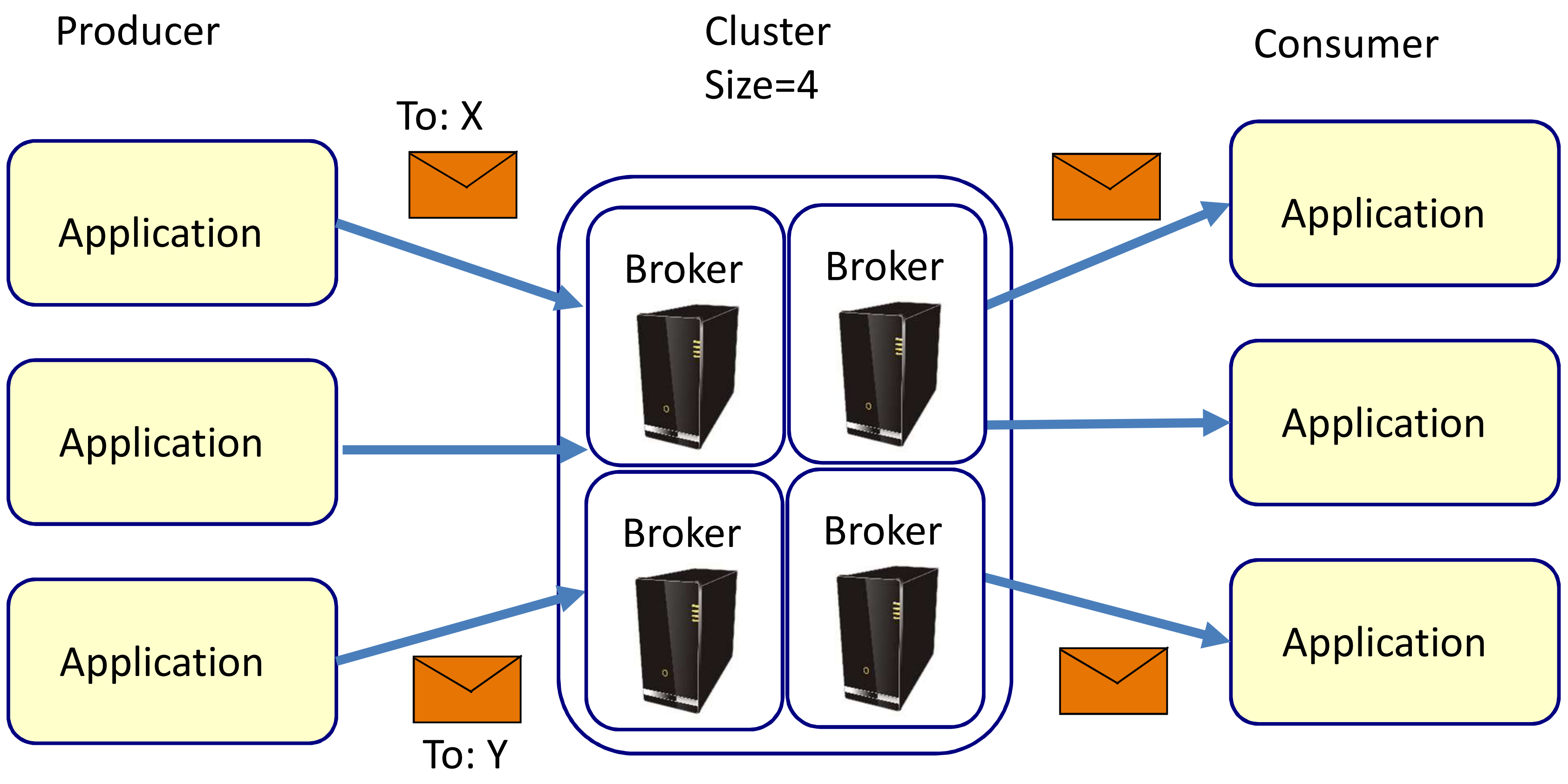
High Volume:
- Over 1.4 trillion messages per day
- 175 terabytes per day

High Velocity:
- Peak 13 million messages per second
- 2.75 gigabytes per second

# Kafka


Producer
Broker
Consumer
To: X
Application
Application
Application
Topic X
Topic Y
To: Y
Application
Application
Application

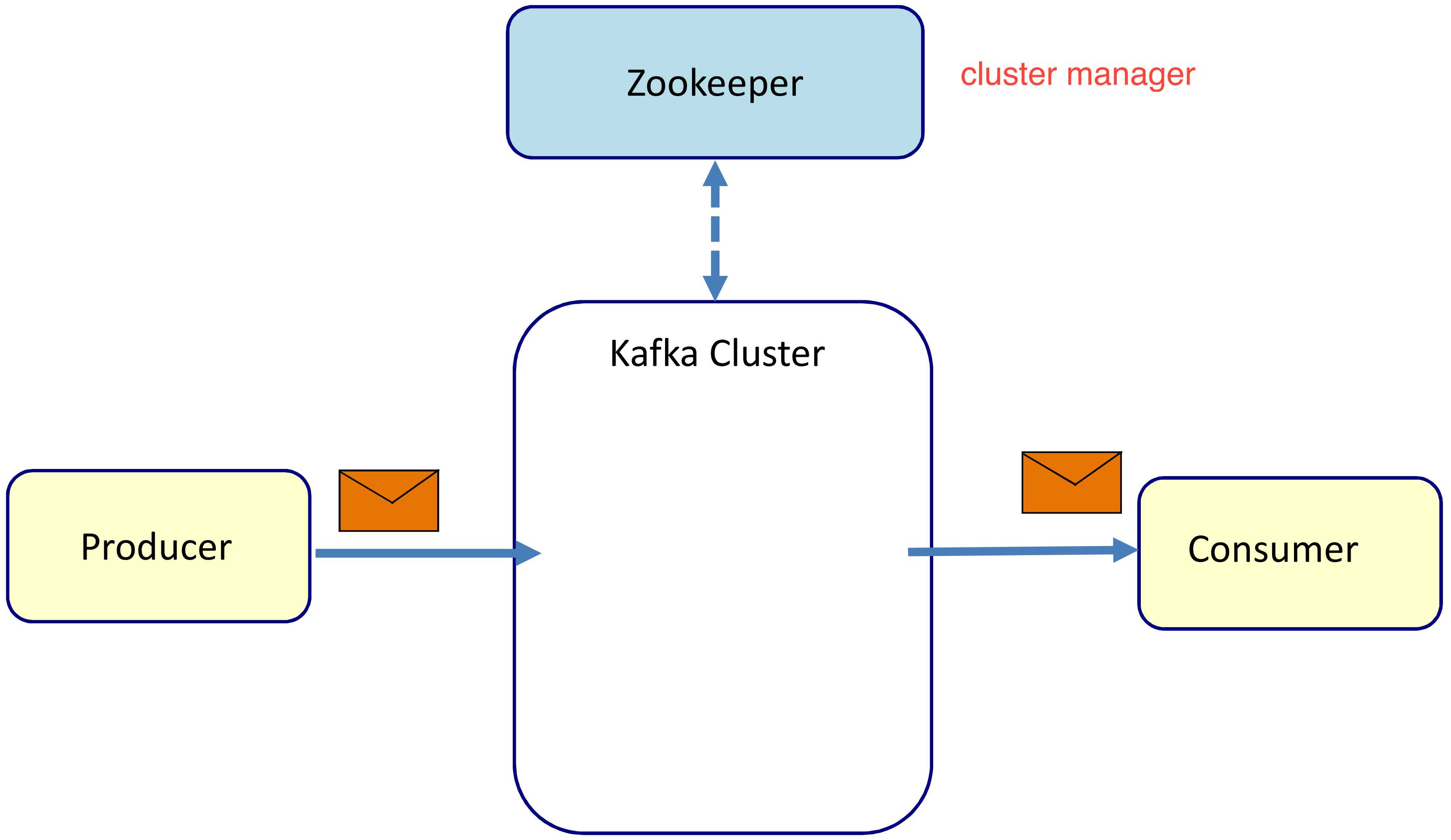
# Cluster of Brokers



## Apache Zookeeper

- Maintains metadata about a cluster of distributed nodes
  - Configuration information
  - Heath status
  - Group membership

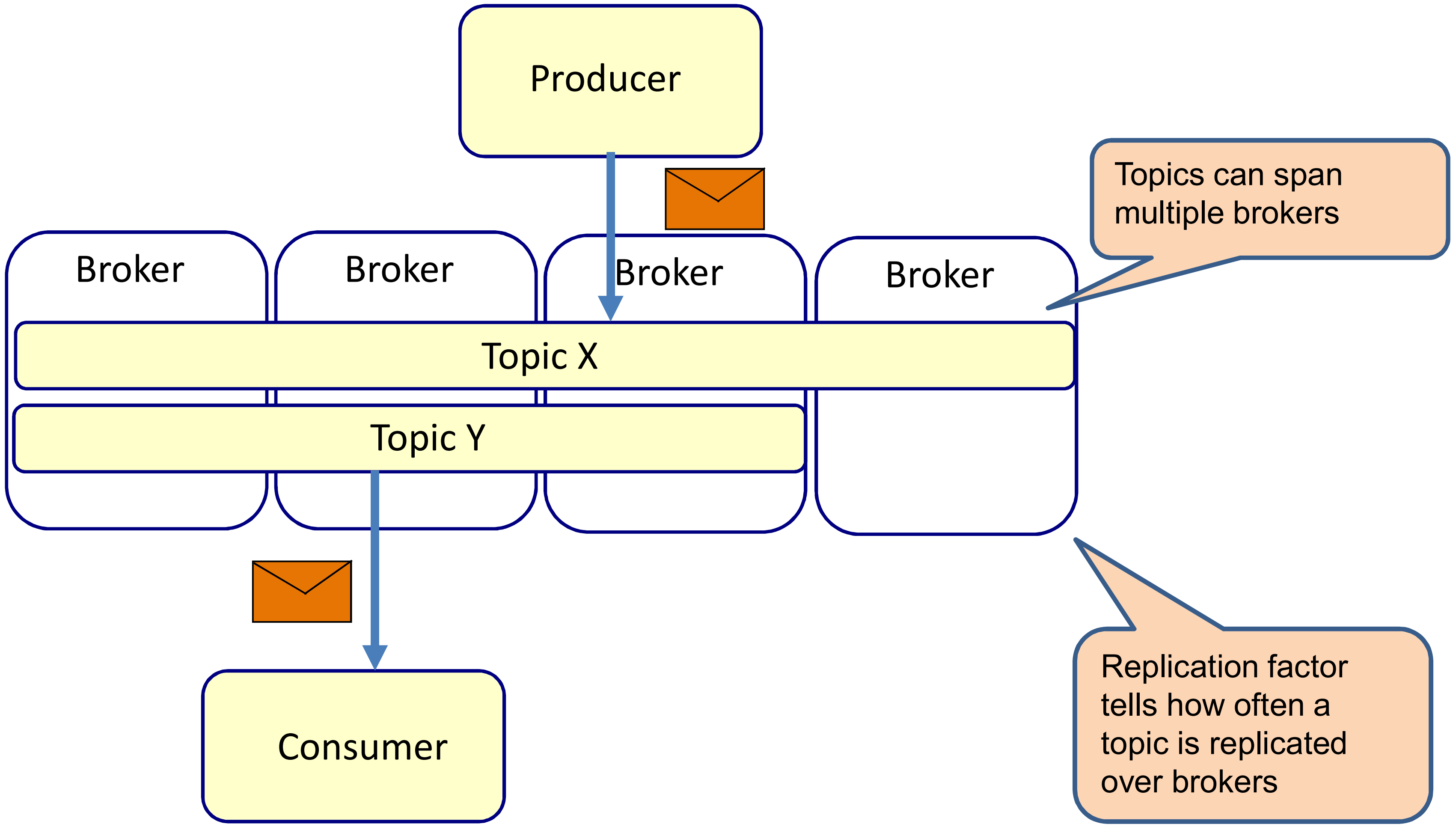
# Kafka distributed architecture



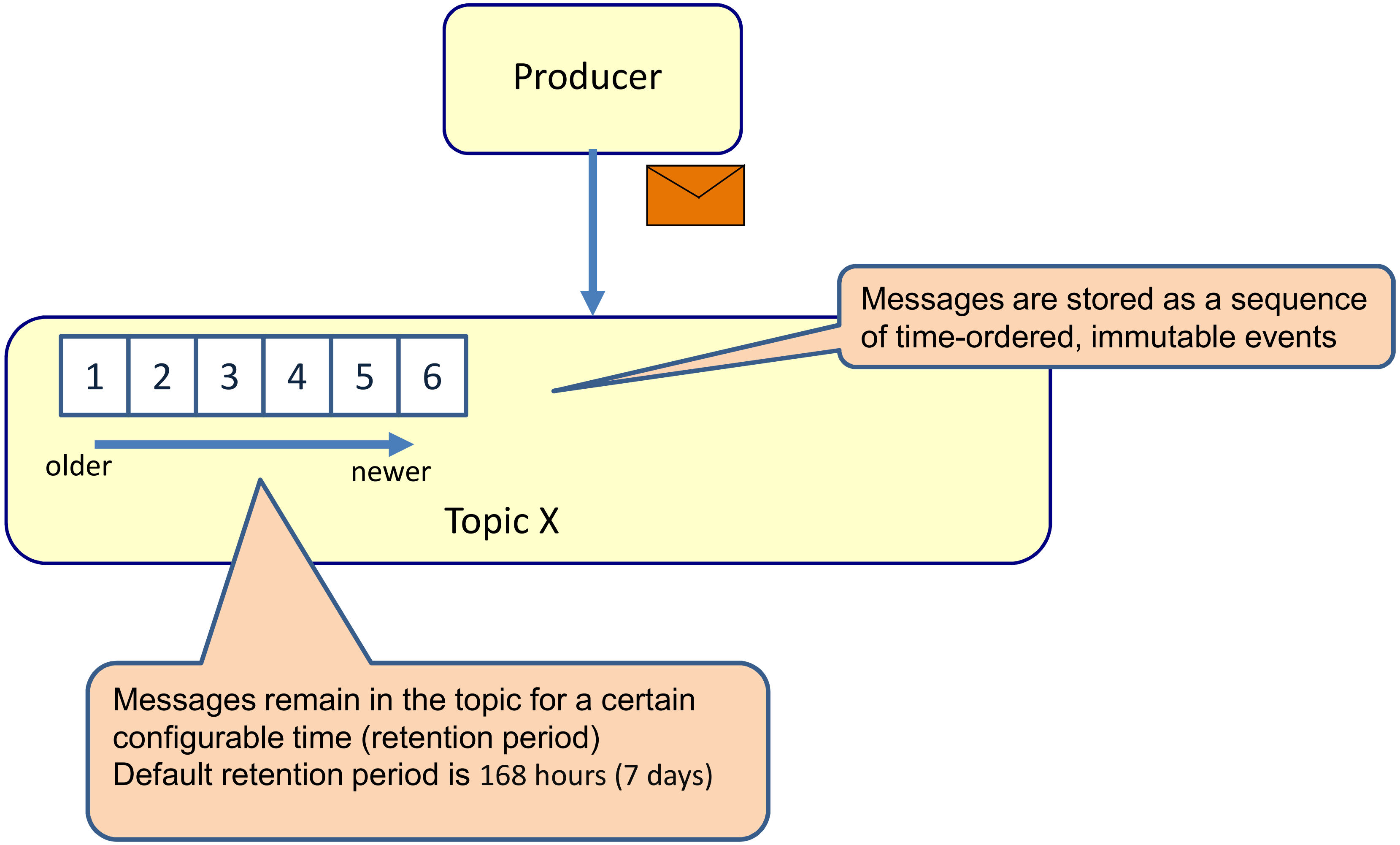
## TOPIC

# Topics

channel


Producer
Broker
Broker
Broker
Broker
Topic X
Topic Y
Consumer
Topics can span multiple brokers
Replication factor tells how often a topic is replicated over brokers

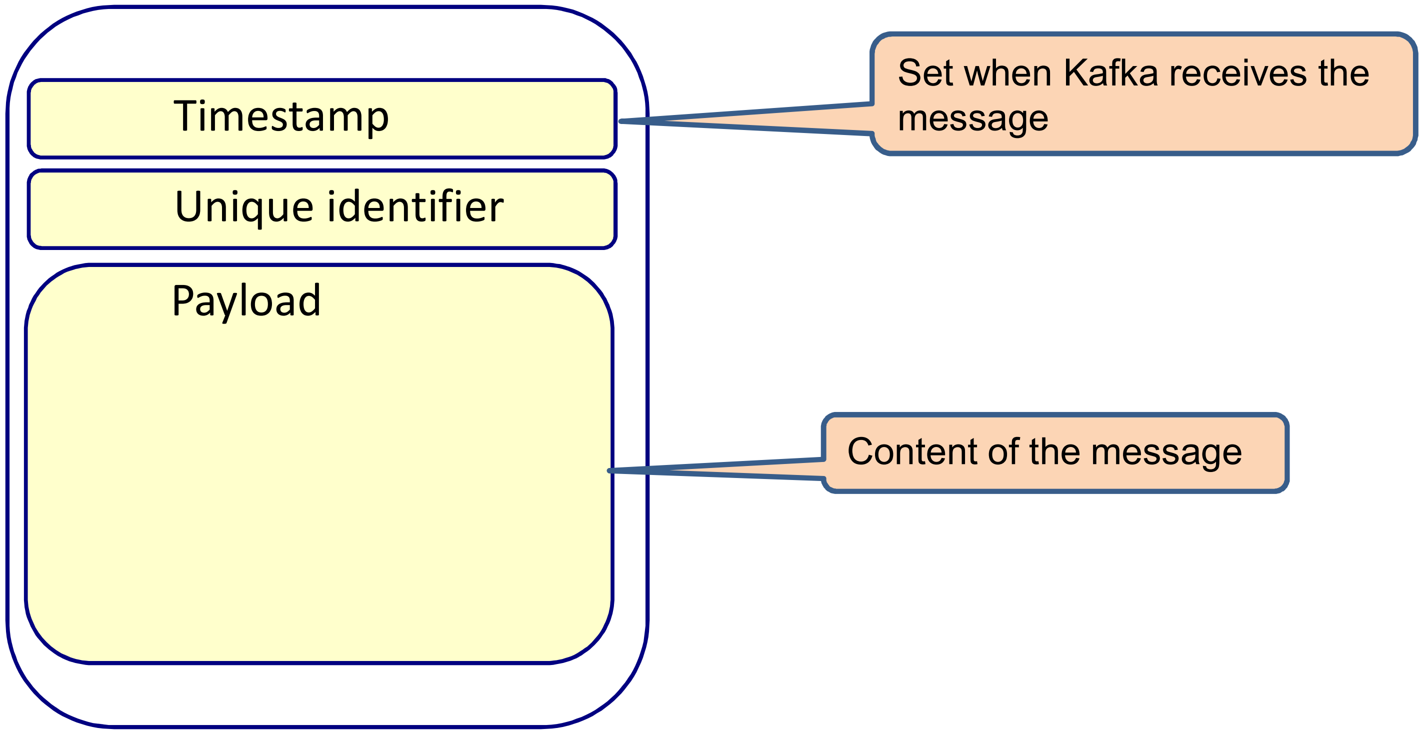
# Event sourcing



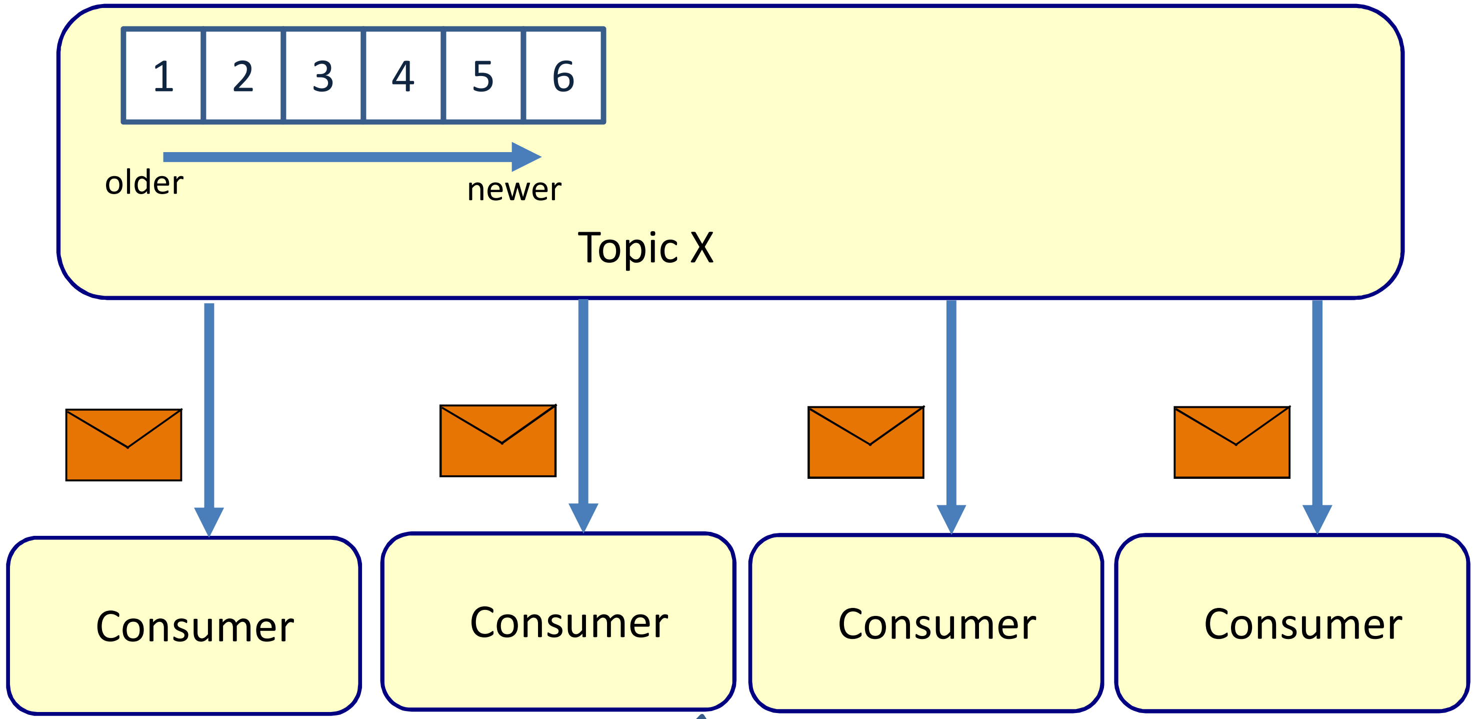
Messages are stored as a sequence of time-ordered, immutable events

Messages remain in the topic for a certain configurable time (retention period)
Default retention period is 168 hours (7 days)

# Message


Timestamp
Unique identifier
Payload
Set when Kafka receives the message
Content of the message

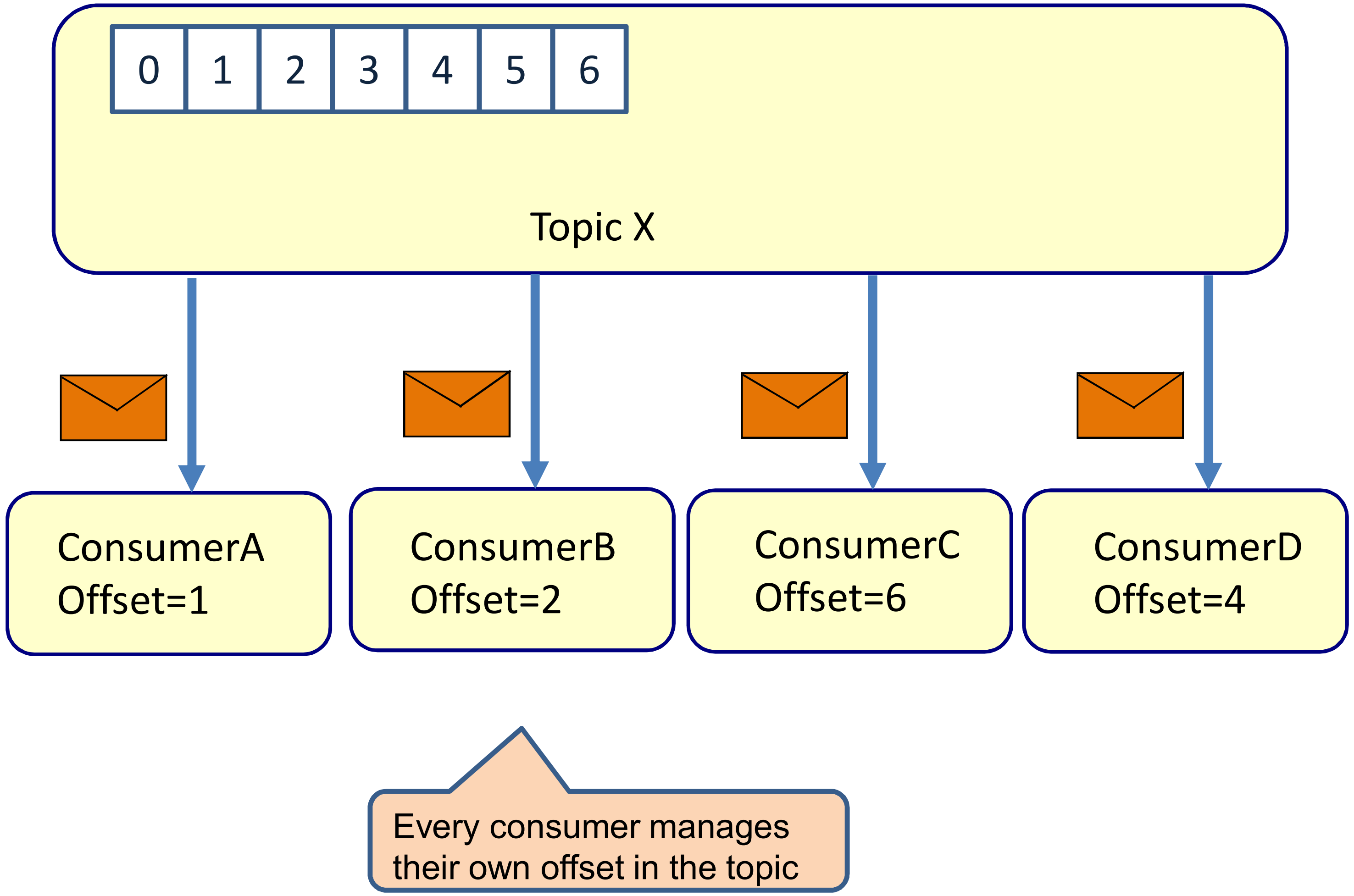
# Consumers of a Topic



For Kafka, it does not matter

1. How many consumers want to access the topic
2. If the consumer(s) have bugs
3. If the consumer(s) are down

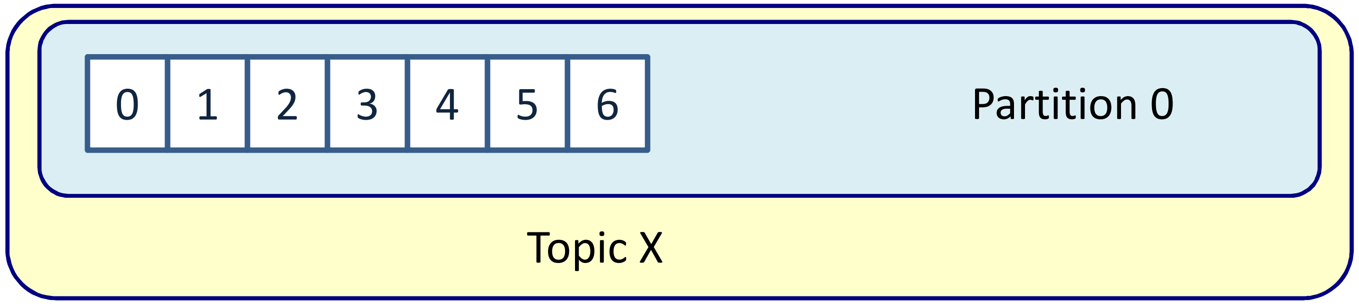
# Offset


0
1
2
3
4
5
6
Topic X
ConsumerA Offset=1
ConsumerB Offset=2
ConsumerC Offset=6
ConsumerD Offset=4
Every consumer manages their own offset in the topic

## Partition

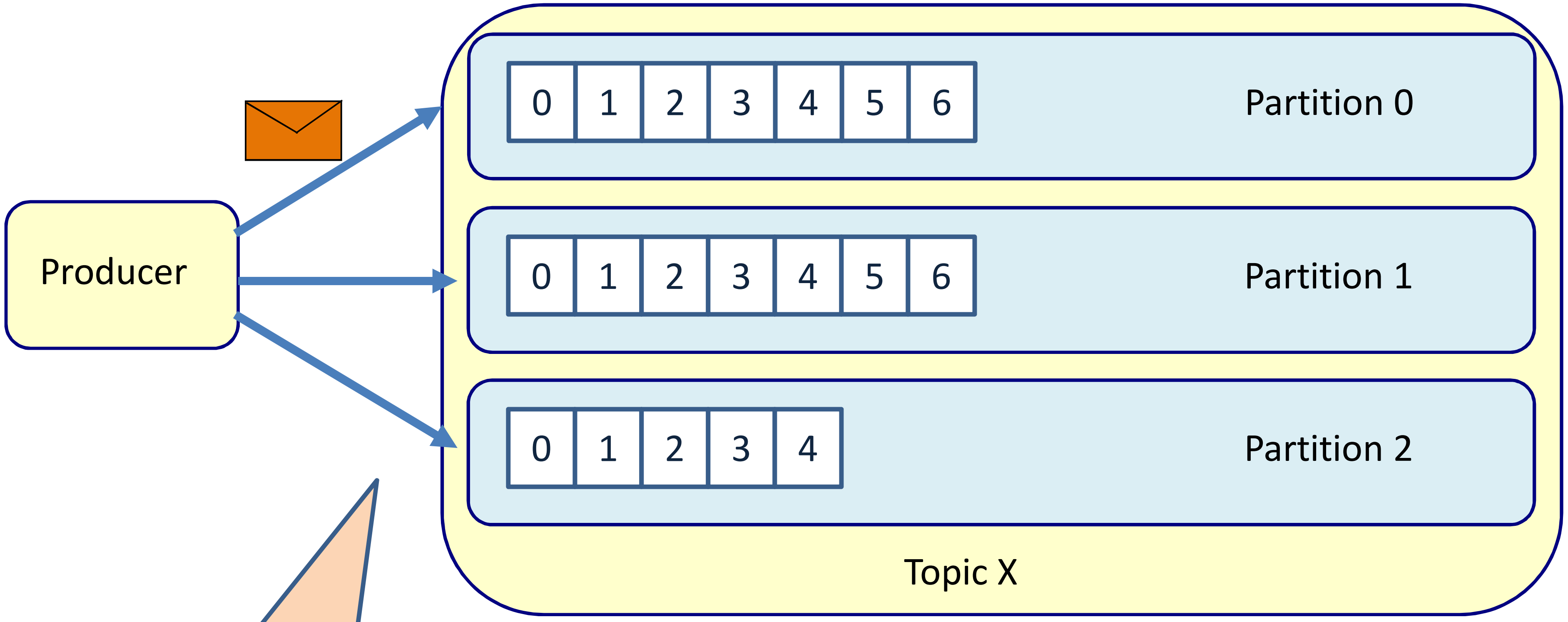
- Each topic has one or more partitions
  - This is configurable
- Each partition is maintained on 1 or more brokers

## 1 partition



- Each partition must fit on 1 broker

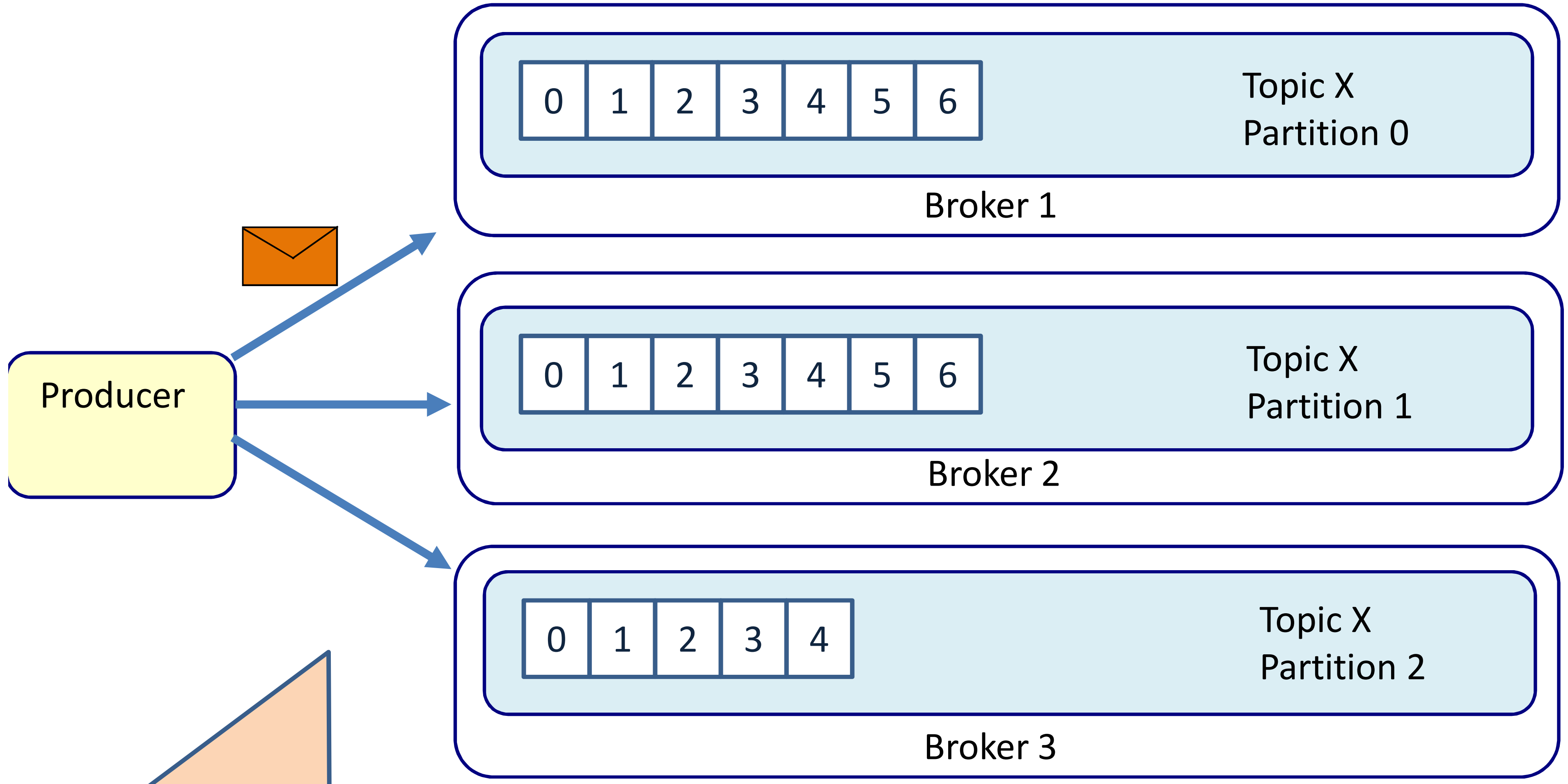
# 3 partitions



Each partition receives different messages

There is no message order between partitions
If this is important
- Use a single partition
- Let the consumer manage the order

# Scale out partitions



Increases
- Scalability (number of messages and their size)
- Higher levels of throughput (parallelism)

## Main point

- Event driven architecture has many advantages over synchronous architectures
- All events in creation has its source in the abstract Unified Field.

## SPRING BOOT KAFKA

## Kafka producer

```
@Service
public class Sender {
    @Autowired
    private KafkaTemplate<String, String> kafkaTemplate;

    @Value("${app.topic.greetingtopic}")
    private String topic;

    public void send(String message){
    System.out.println("sending message="+message+" to topic="+ topic);
        kafkaTemplate.send(topic, message);
    }
}
```

# Kafka consumer

```
@Service
public class Receiver {

    @KafkaListener(topics = "${app.topic.greetingtopic}")
    public void receive(@Payload String message,
                        @Headers MessageHeaders headers) {
      System.out.println("received message="+ message);
      headers.keySet().forEach(key -> System.out.println(key+" : "+ headers.get(key)));
    }

}
```

## The configuration

**application.properties**

```
spring.kafka.bootstrap-servers=localhost:9092
spring.kafka.consumer.group-id= gid
spring.kafka.consumer.auto-offset-reset= earliest
spring.kafka.consumer.key-deserializer=
org.apache.kafka.common.serialization.StringDeserializer
spring.kafka.consumer.value-deserializer=
org.apache.kafka.common.serialization.StringDeserializer
spring.kafka.producer.key-serializer=
org.apache.kafka.common.serialization.StringSerializer
spring.kafka.producer.value-serializer=
org.apache.kafka.common.serialization.StringSerializer

app.topic.greetingtopic= greetingtopic

logging.level.root= ERROR
org.springframework= ERROR
```

# The application

```java
@SpringBootApplication
@EnableKafka
public class KafkaProjectApplication implements CommandLineRunner {

    public static void main(String[] args) {
        SpringApplication.run(KafkaProjectApplication.class, args);
    }

    @Autowired
    private Sender sender;

    @Override
    public void run(String... strings) throws Exception {
        sender.send("Spring Kafka and Spring Boot Configuration Example");
    }
}
```

# Connecting the parts of knowledge with the wholeness of knowledge

1. OAuth2 is a token based authorization framework that allows us to secure microservices
2. Kafka is a distributed, scalable, fault-tolerant message broker that can handle millions of transactions per second

---

3. **Transcendental consciousness** is the never changing field at the basis of all change.
4. **Wholeness moving within itself:** In Unity Consciousness, the eternal and universal creative activity that maintains the universe is realized as the self-referral dynamics of one's own consciousness.