

# SECURITY



# Aspects of security

---

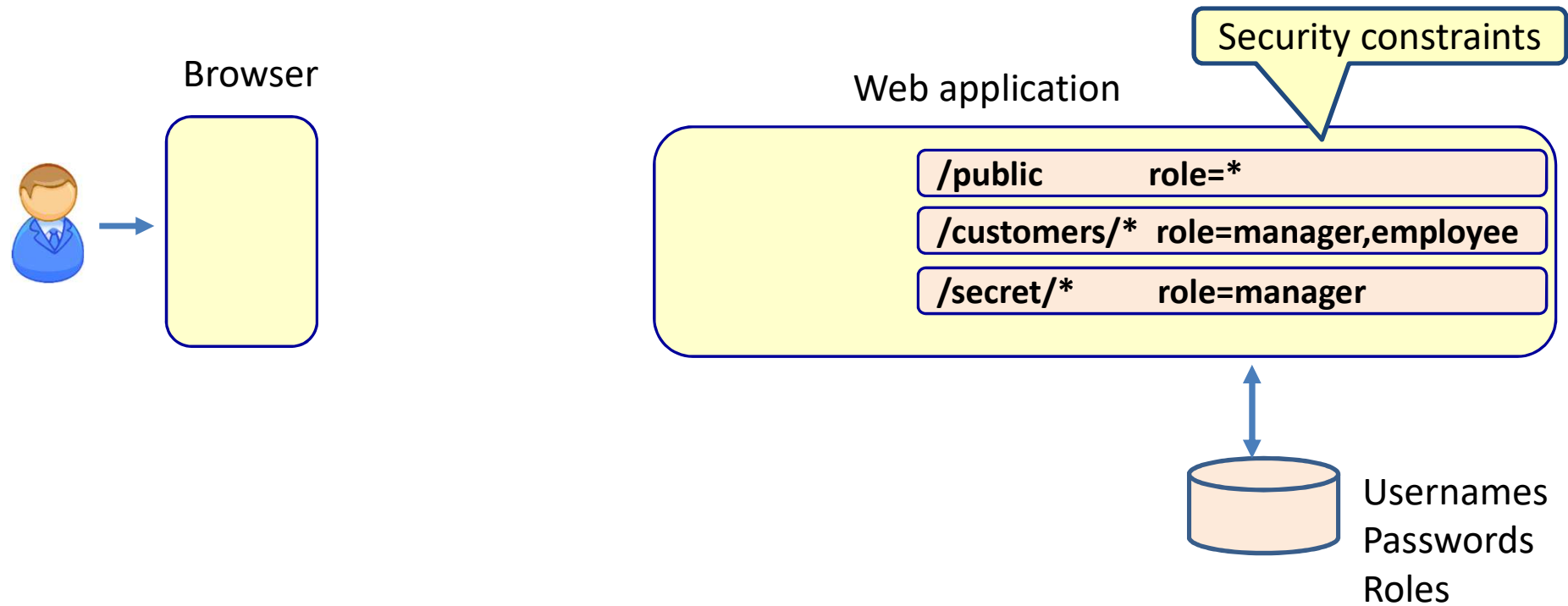
- Authentication: are you who you say you are?
  - Login with username/password
- Authorization: what are you allowed to do?
  - Make url's and/or methods secure
- Confidentiality: No one may look into this request/response
  - Encryption
- Data integrity: No one may change this request/response
  -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



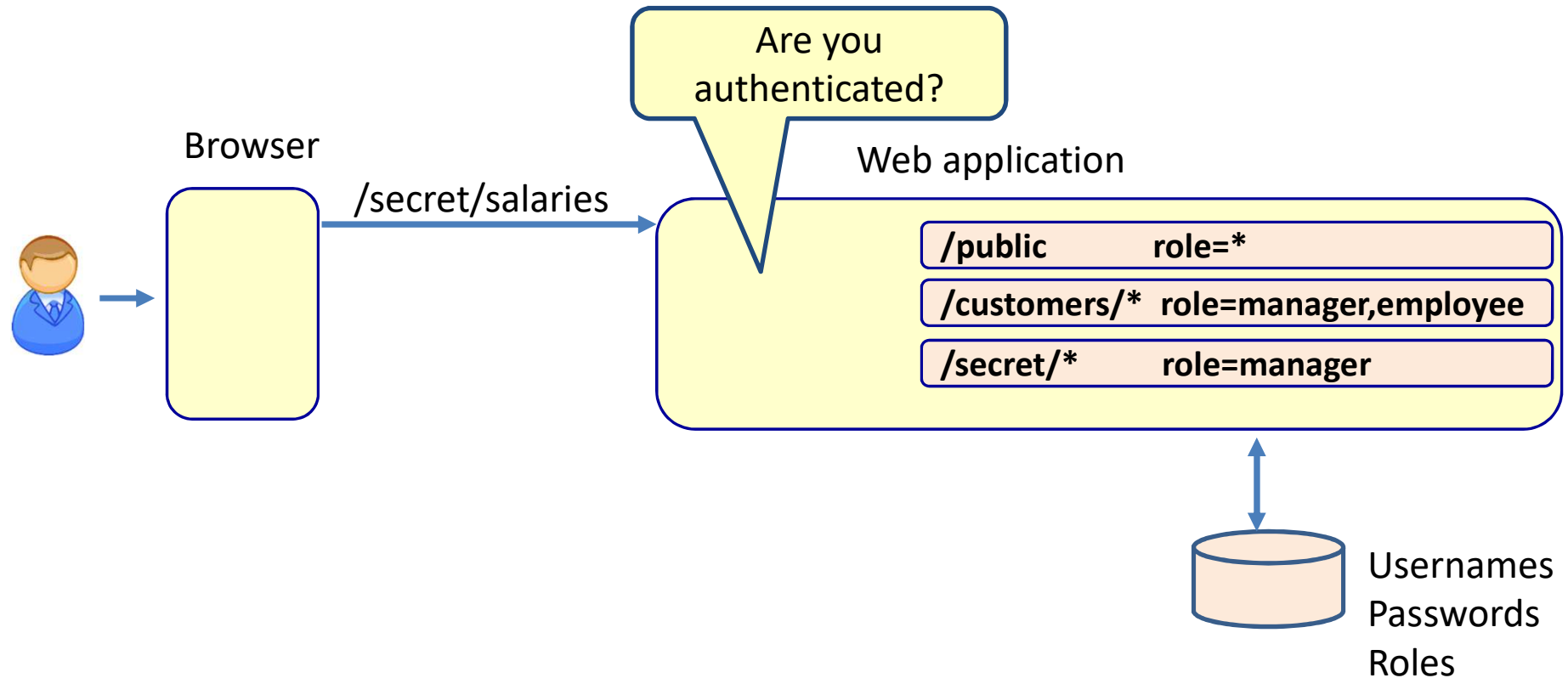
# SECURING A WEB APPLICATION



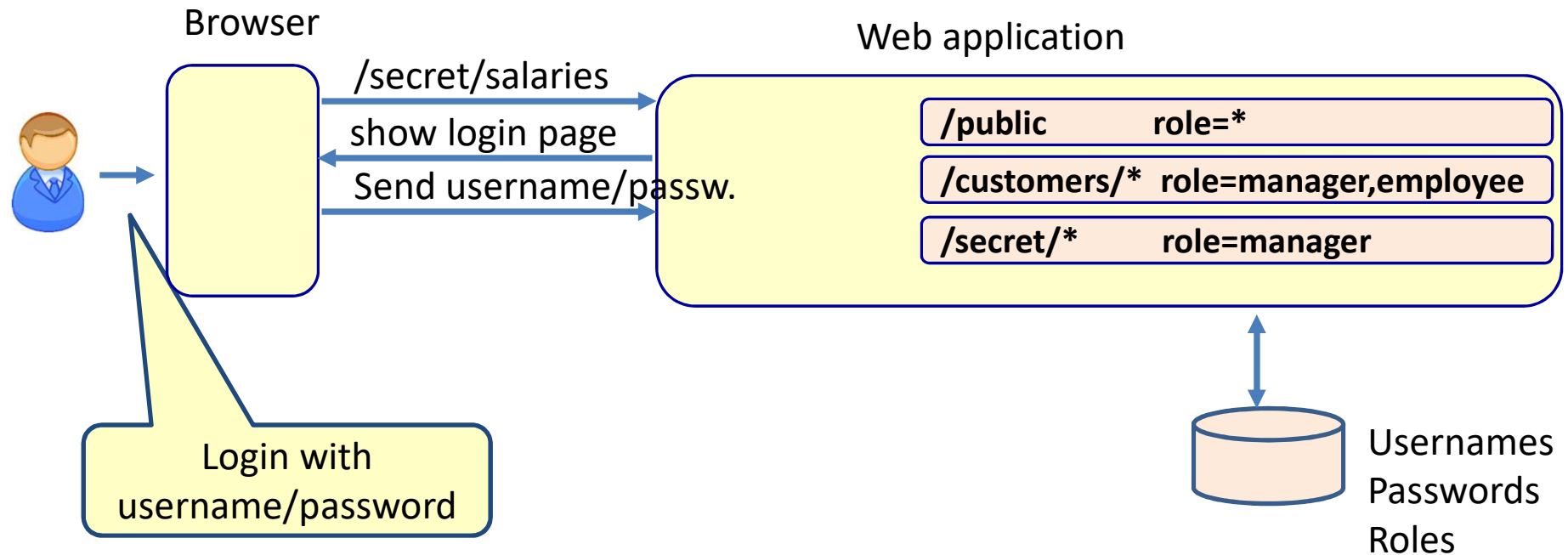
# Web security



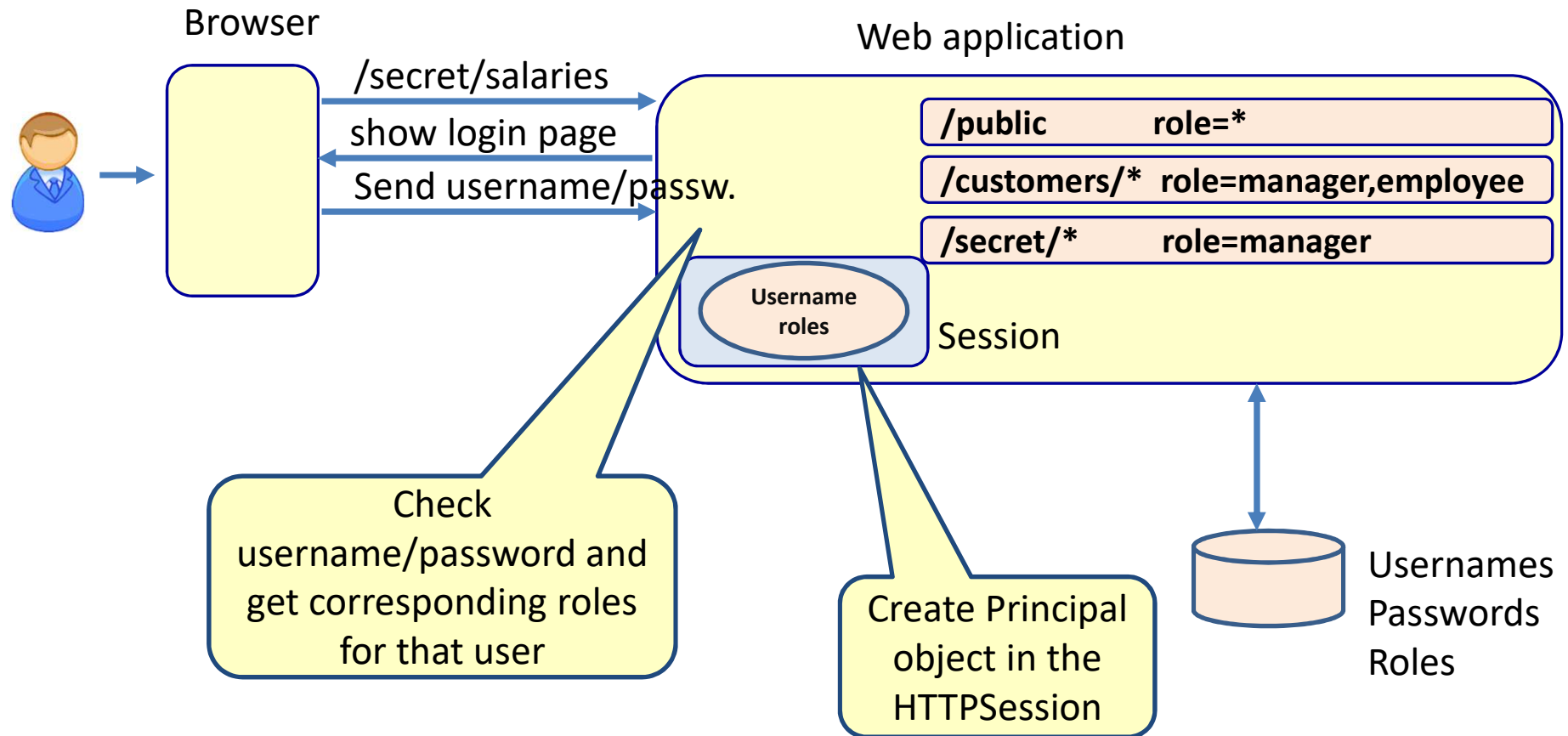
# Web security



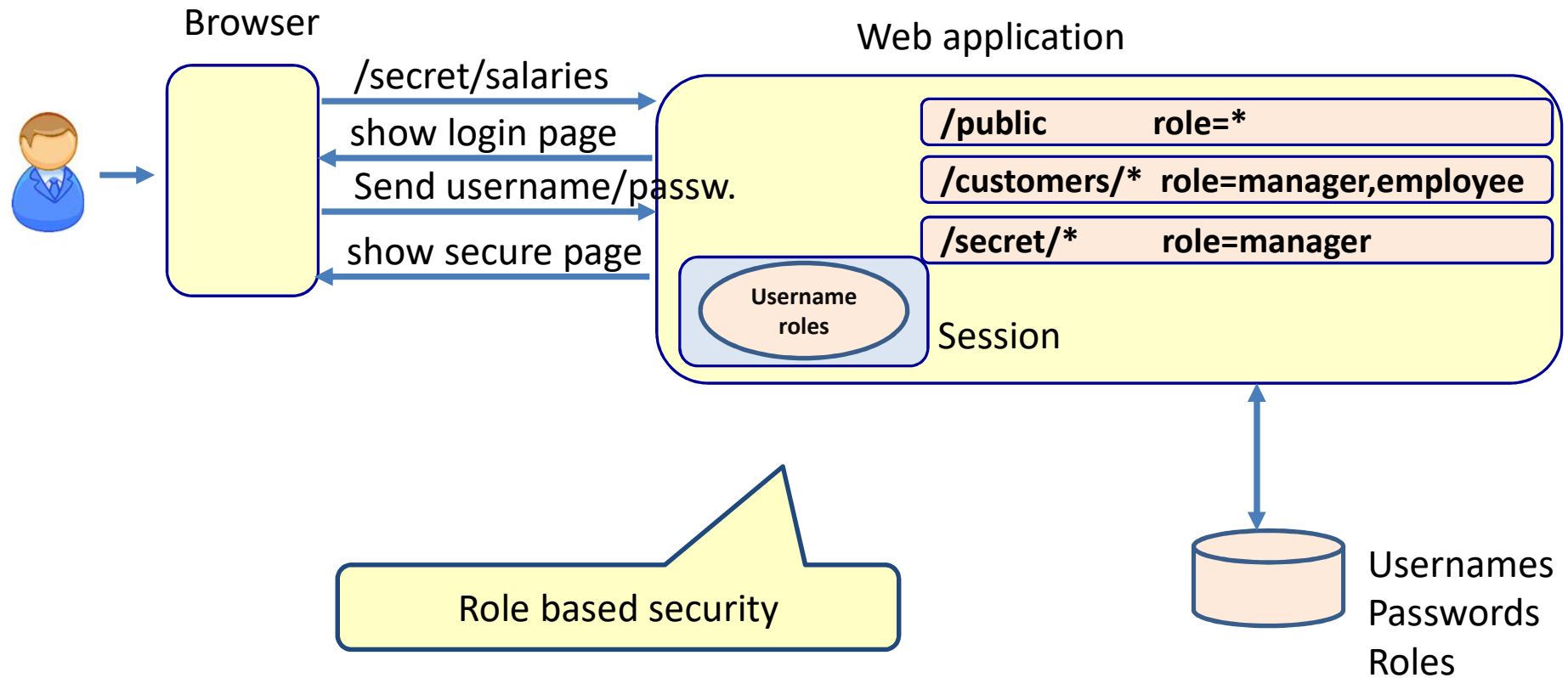
# Web security



# Web security

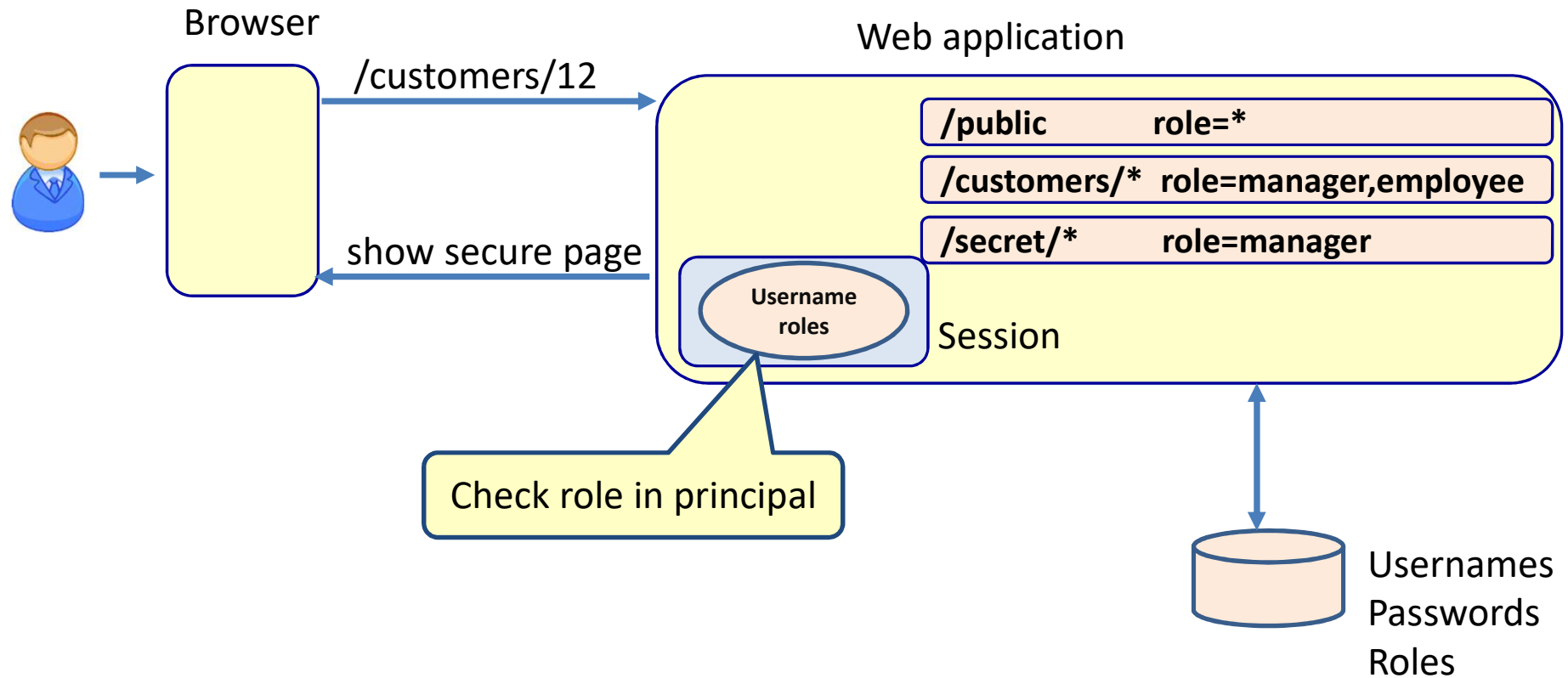


# Web security





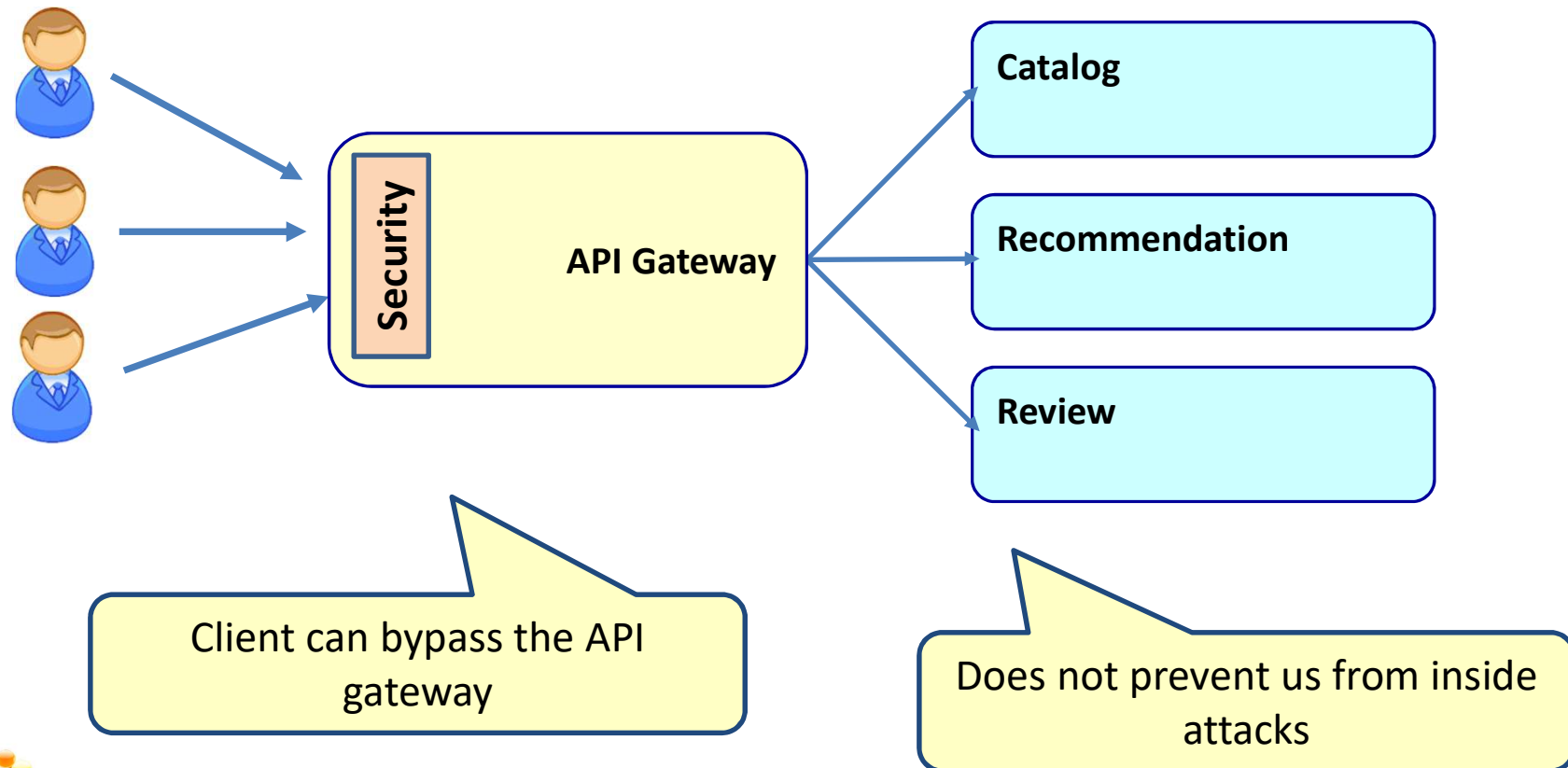
# Web security



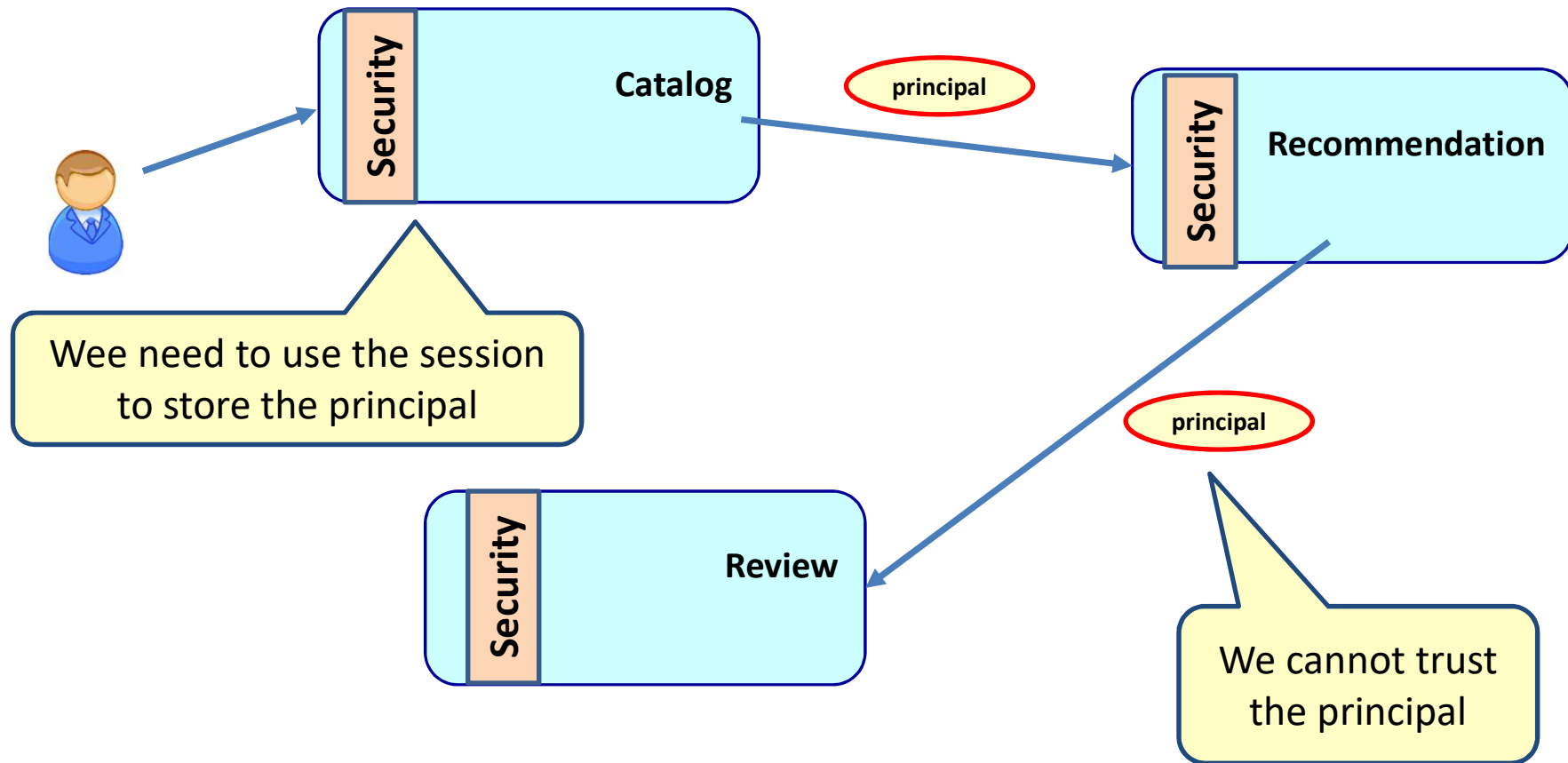
# **SECURE THE MICROSERVICE ARCHITECTURE**



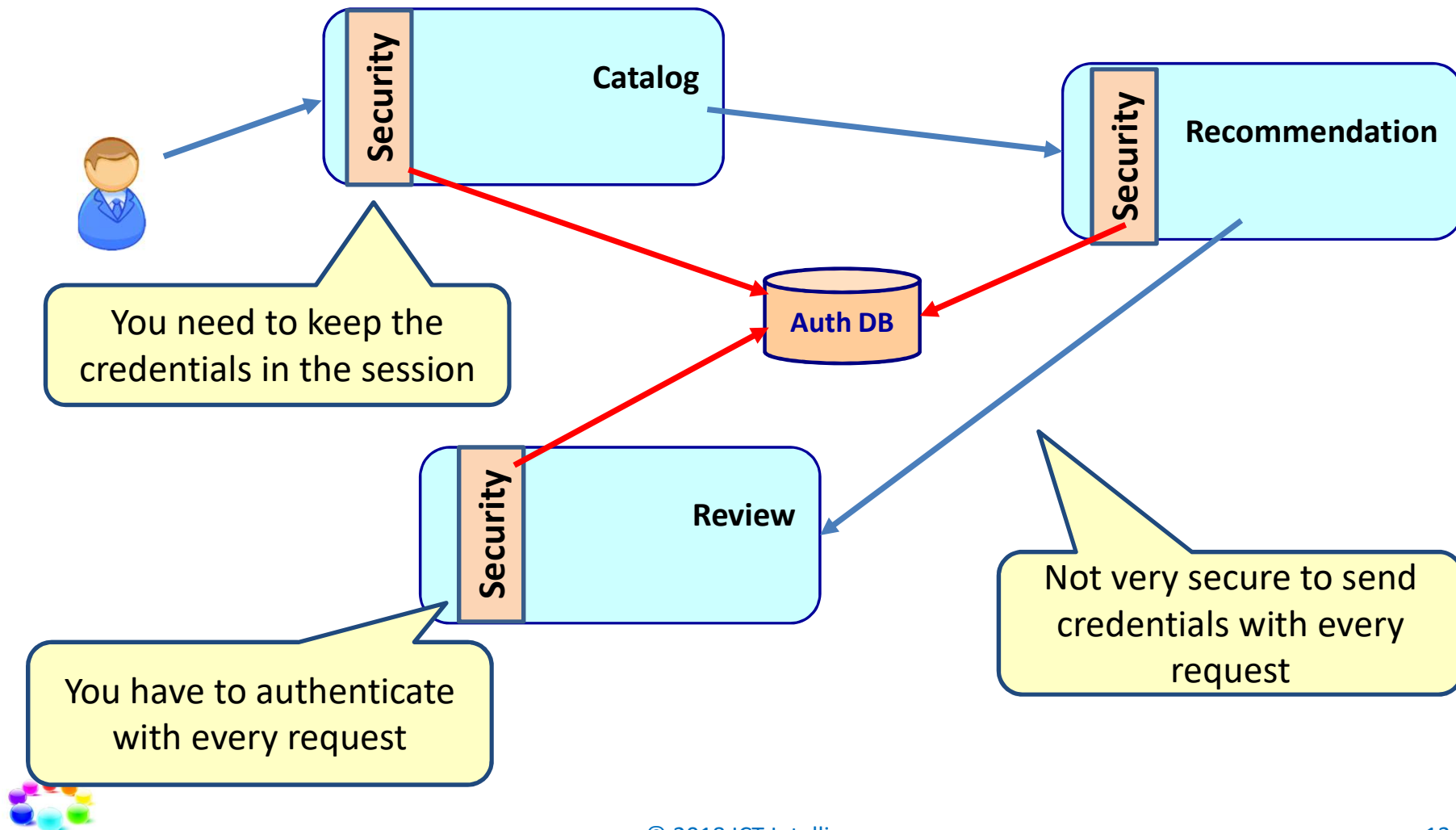
# Secure the API gateway



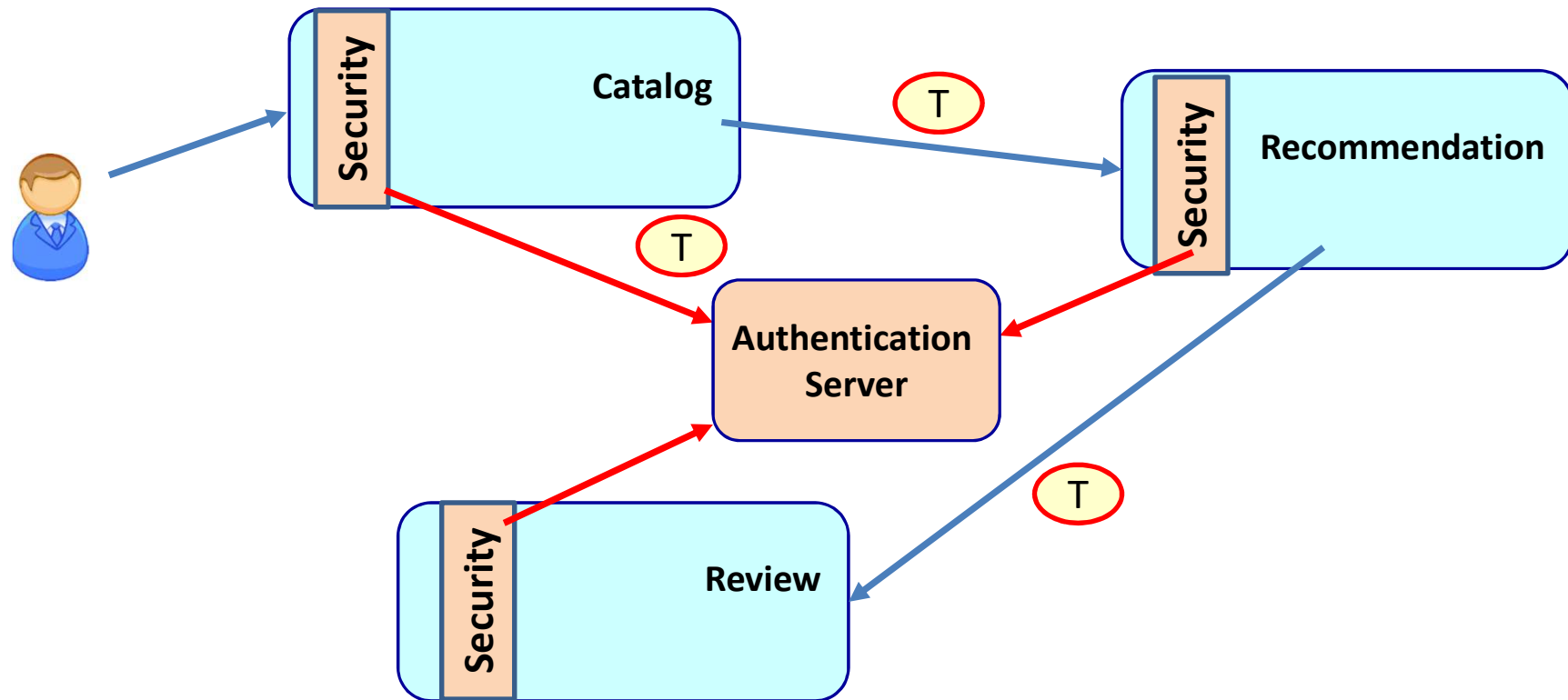
# Send principal with every request



# Send userid/password with every request

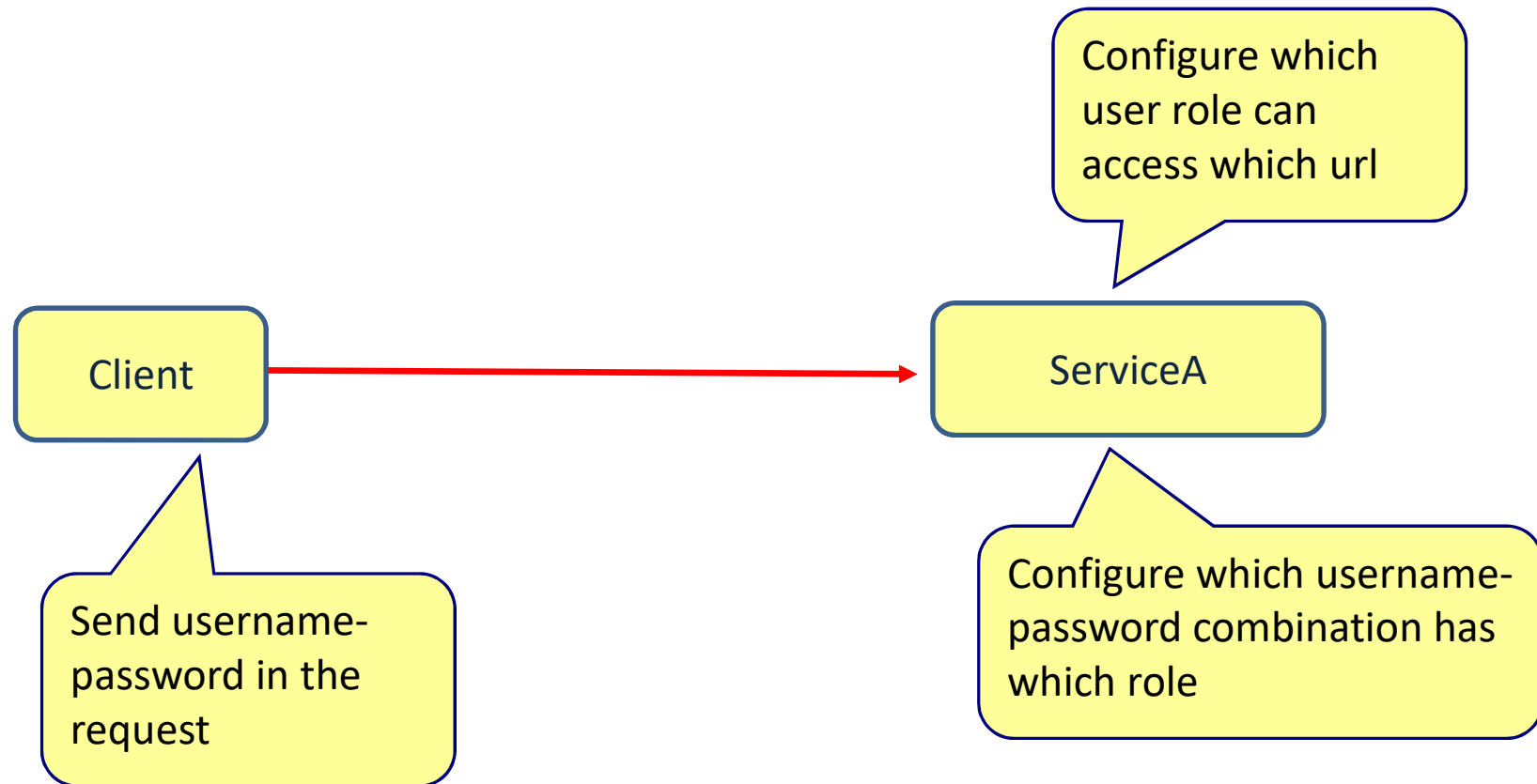


# OAuth2: Token based security



# Securing a REST service

---



# 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", String.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

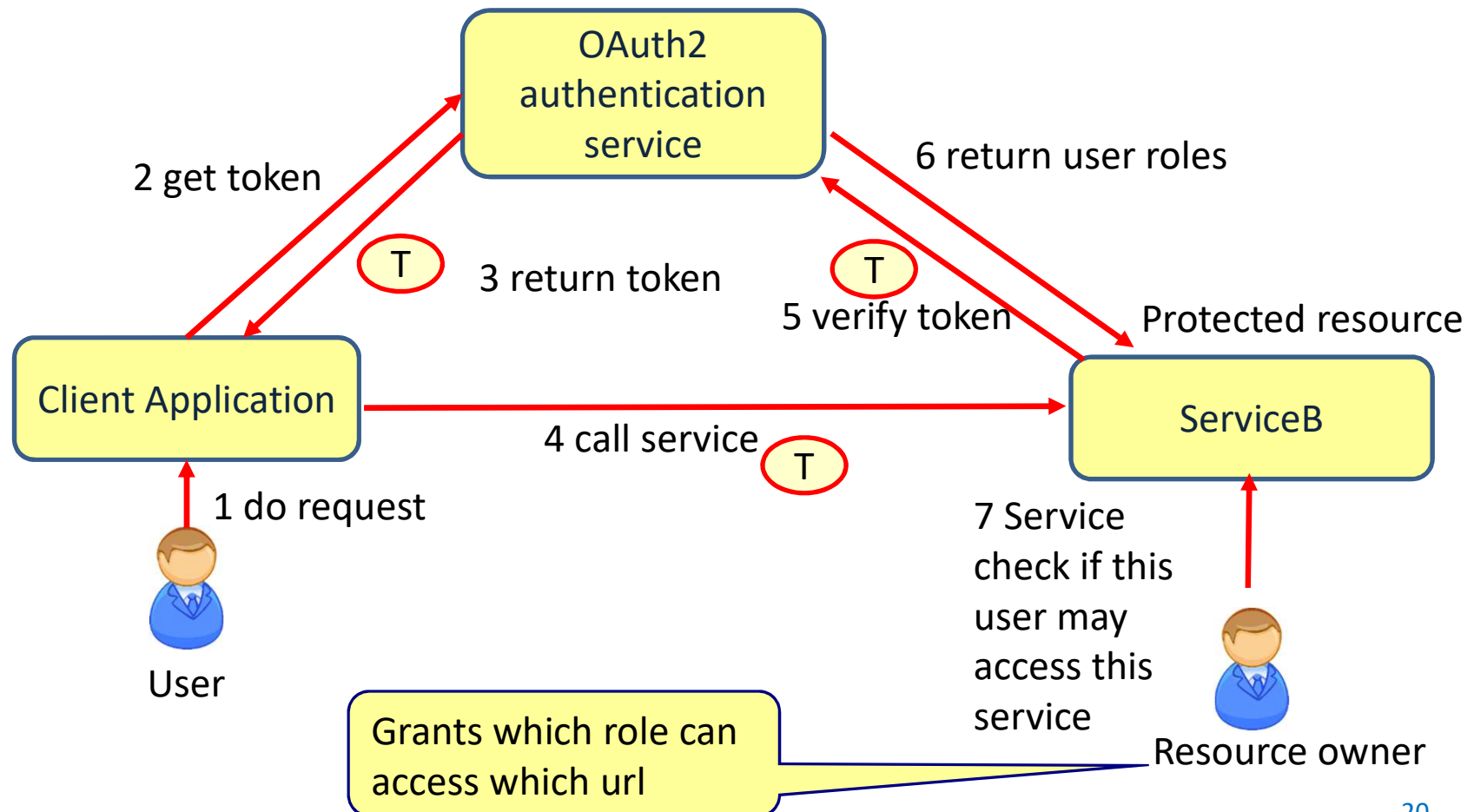


# **OAUTH2**



# How does OAuth2 work

- Token based authentication and authorization framework



# **OAUTH2 AUTHENTICATION SERVICE**



# The authentication service

```
@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

```
@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

```
@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

POST

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

Params

Send

Save

Authorization

Headers (2)

Body

Pre-request Script

Tests

Cookies

Code

TYPE

Basic Auth

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](#)

Username

theClient

Password

thisissecret

☒ Show Password

POST

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

Params

Send

Save

Authorization

Headers (2)

Body

Pre-request Script

Tests

Cookies

Code

☐ form-data

☒ x-www-form-urlencoded

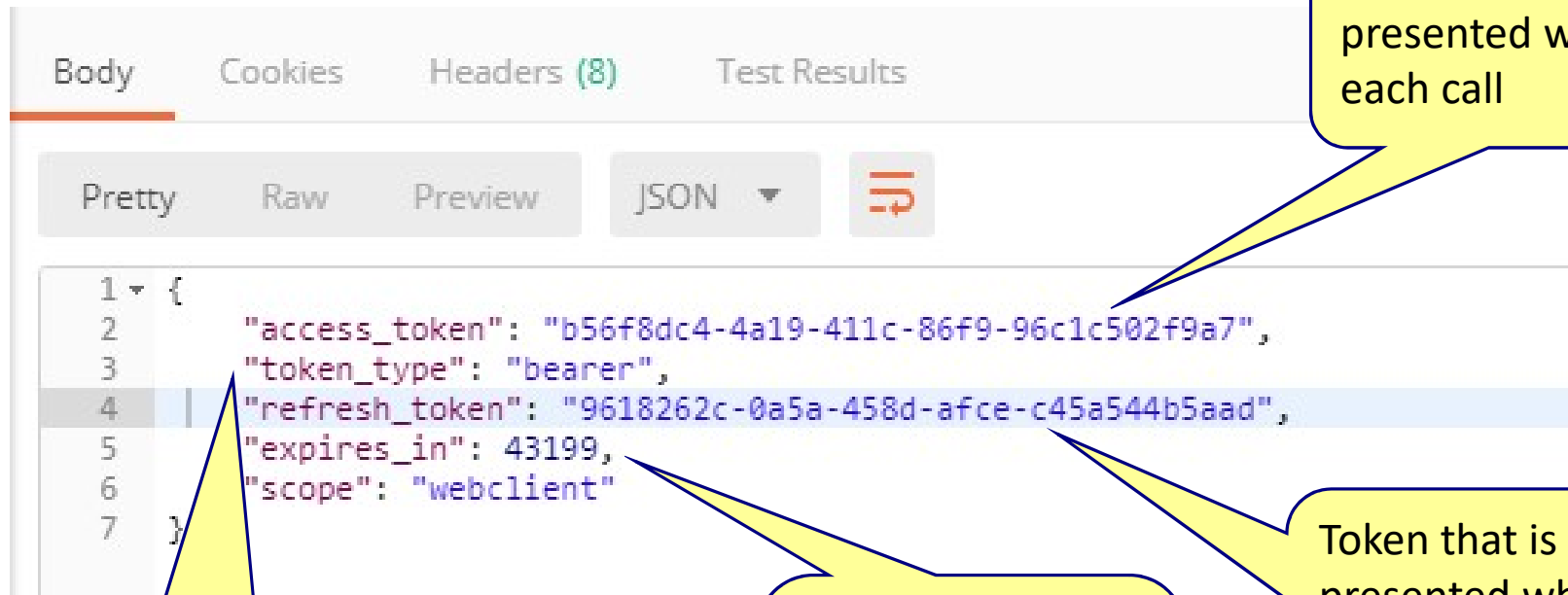
☐ raw

☐ binary

	Key	Value	Description	...	Bulk Edit
<input checked="" type="checkbox"/>	grant_type	password			
<input checked="" type="checkbox"/>	scope	webclient			
<input checked="" type="checkbox"/>	username	john			
<input checked="" type="checkbox"/>	password	password1			
	New key	Value	Description		



# Returned payload



```
1 {
2   "access_token": "b56f8dc4-4a19-411c-86f9-96c1c502f9a7",
3   "token_type": "bearer",
4   "refresh_token": "9618262c-0a5a-458d-afce-c45a544b5aad",
5   "expires_in": 43199,
6   "scope": "webclient"
7 }
```

Oauth supports multiple token types


Access\_token is presented with each call

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

Token that is presented when you refresh an expired token







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

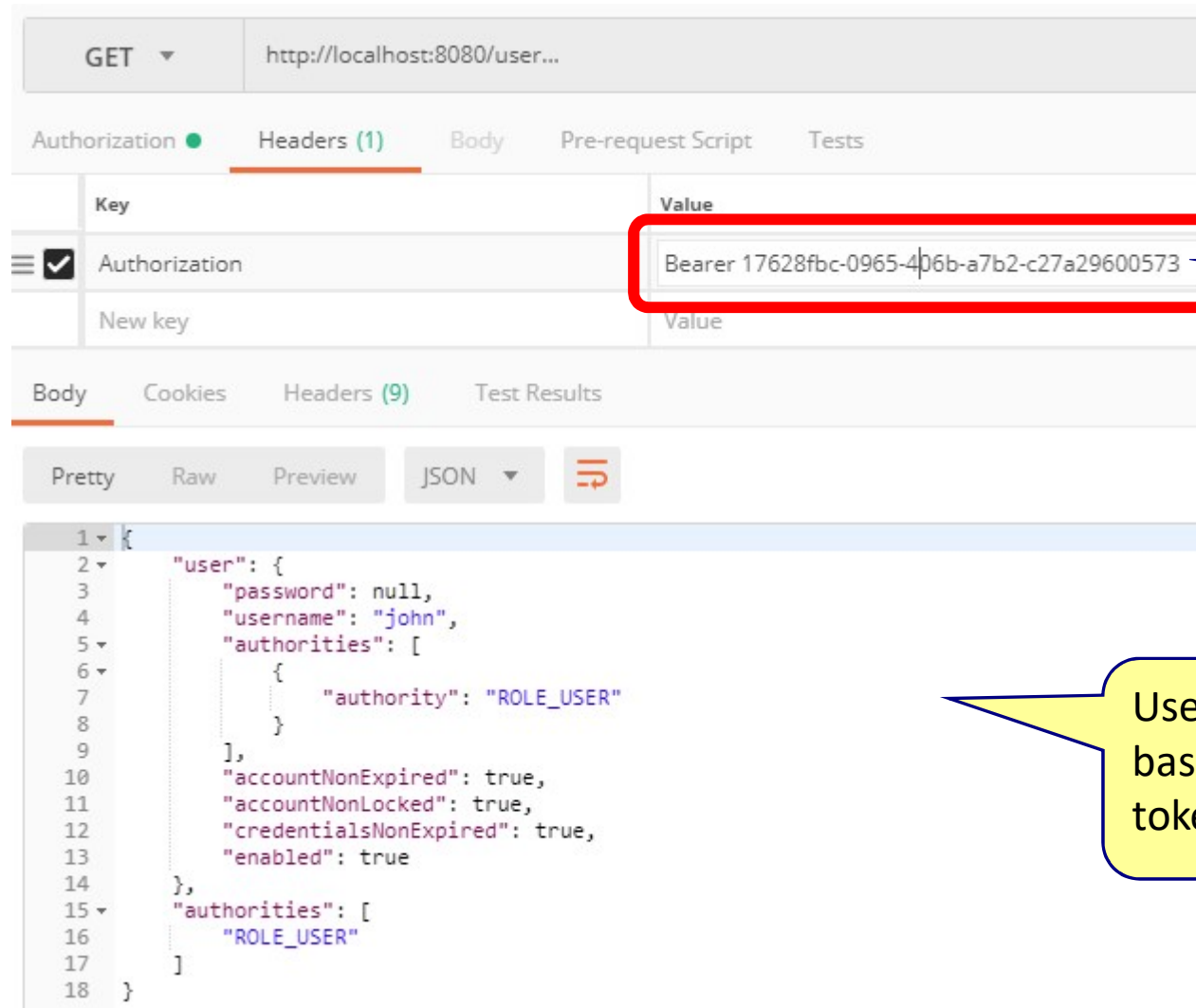
GET ▾
http://localhost:8080/user...
Params
Send ▾
Save ▾

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

	Key	Value	Description	...	Bulk Edit	Presets ▾
<input checked="" type="checkbox"/>	Authorization	Bearer 127e57aa-b7da-4ac5-94b6-d5ca86a0df9b				
	New key	Value	Description			



# Get user information



The screenshot displays a REST client interface. The top section shows a GET request to `http://localhost:8080/user...`. The 'Headers' tab is selected, showing an 'Authorization' header with the value 'Bearer 17628fbc-0965-406b-a7b2-c27a29600573'. A red box highlights this value, with a callout pointing to it. Below the headers, the 'Body' tab is selected, showing a JSON response. The JSON structure includes a 'user' object with fields like 'password', 'username', 'authorities', 'accountNonExpired', 'accountNonLocked', 'credentialsNonExpired', and 'enabled'. A callout points to the 'user' object in the JSON.

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

```
1 {
2   "user": {
3     "password": null,
4     "username": "john",
5     "authorities": [
6       {
7         "authority": "ROLE_USER"
8       }
9     ],
10    "accountNonExpired": true,
11    "accountNonLocked": true,
12    "credentialsNonExpired": true,
13    "enabled": true
14  },
15  "authorities": [
16    "ROLE_USER"
17  ]
18 }
```

Token for John

User information  
based on OAuth  
token



# Get user information

The screenshot displays a web browser's developer tools interface. At the top, a GET request is shown to the URL `http://localhost:8080/user...`. The 'Headers' tab is selected, showing a table with one header: 'Authorization' with a value of 'Bearer e964de5e-d5c7-4f9d-89e3-66eea2b9d507'. This value is highlighted with a red rectangle, and a yellow callout bubble points to it with the text 'Token for Frank'. Below the headers, the 'Body' tab is selected, showing a JSON response in 'Pretty' format. The JSON object contains user details for 'frank', including roles 'ROLE\_MANAGER' and 'ROLE\_USER', and various status flags. A yellow callout bubble points to the JSON body with the text 'User information based on OAuth token'.

Key	Value
<input checked="" type="checkbox"/> Authorization	Bearer e964de5e-d5c7-4f9d-89e3-66eea2b9d507
New key	Value

```
1 {
2   "user": {
3     "password": null,
4     "username": "frank",
5     "authorities": [
6       {
7         "authority": "ROLE_MANAGER"
8       },
9       {
10        "authority": "ROLE_USER"
11      }
12    ],
13    "accountNonExpired": true,
14    "accountNonLocked": true,
15    "credentialsNonExpired": true,
16    "enabled": true
17  },
18  "authorities": [
19    "ROLE_MANAGER",
20    "ROLE_USER"
21  ]
22 }
```

# 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 getSalary() {
        return "95.000";
    }

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



# The configuration

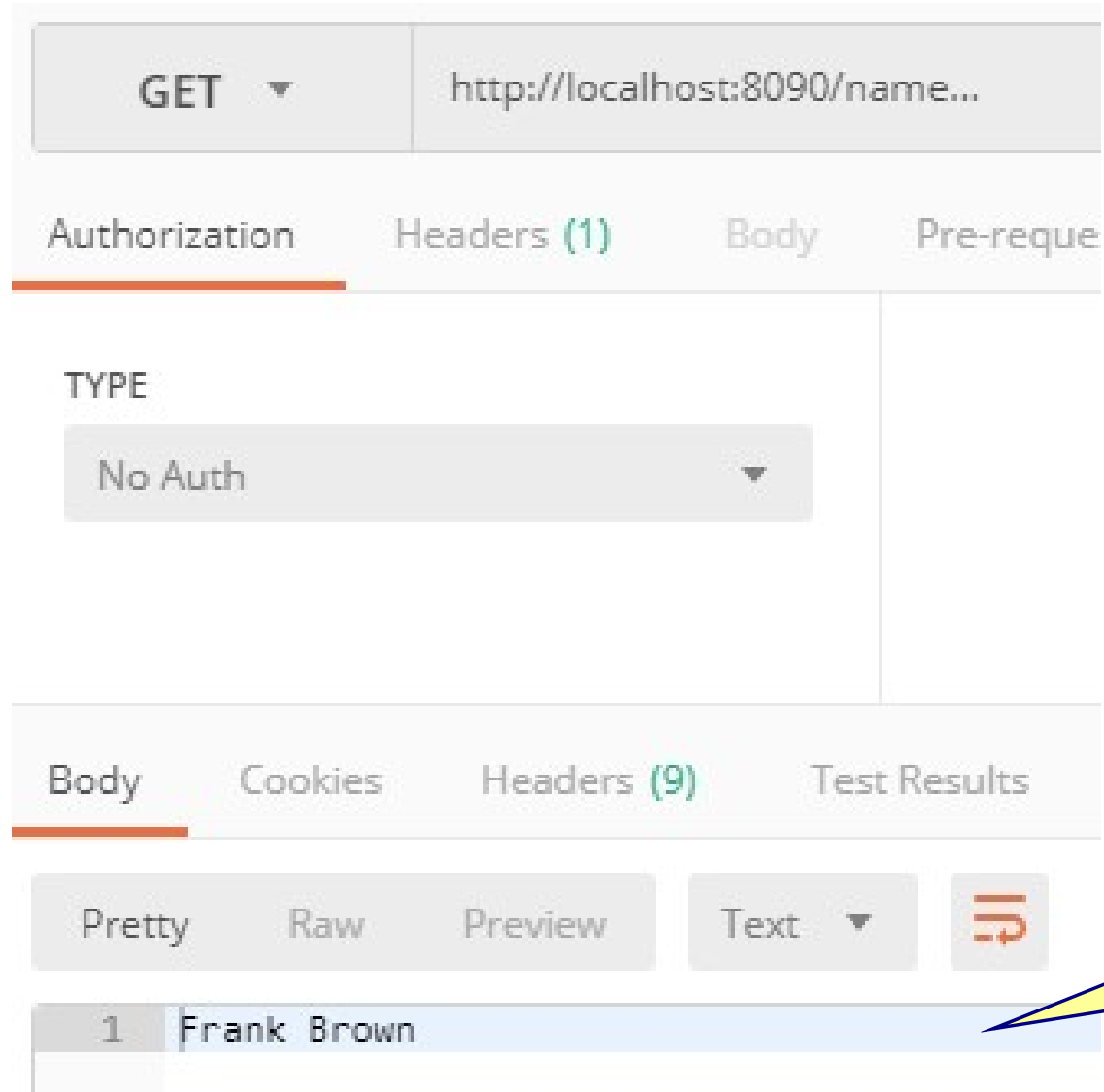
---

## 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



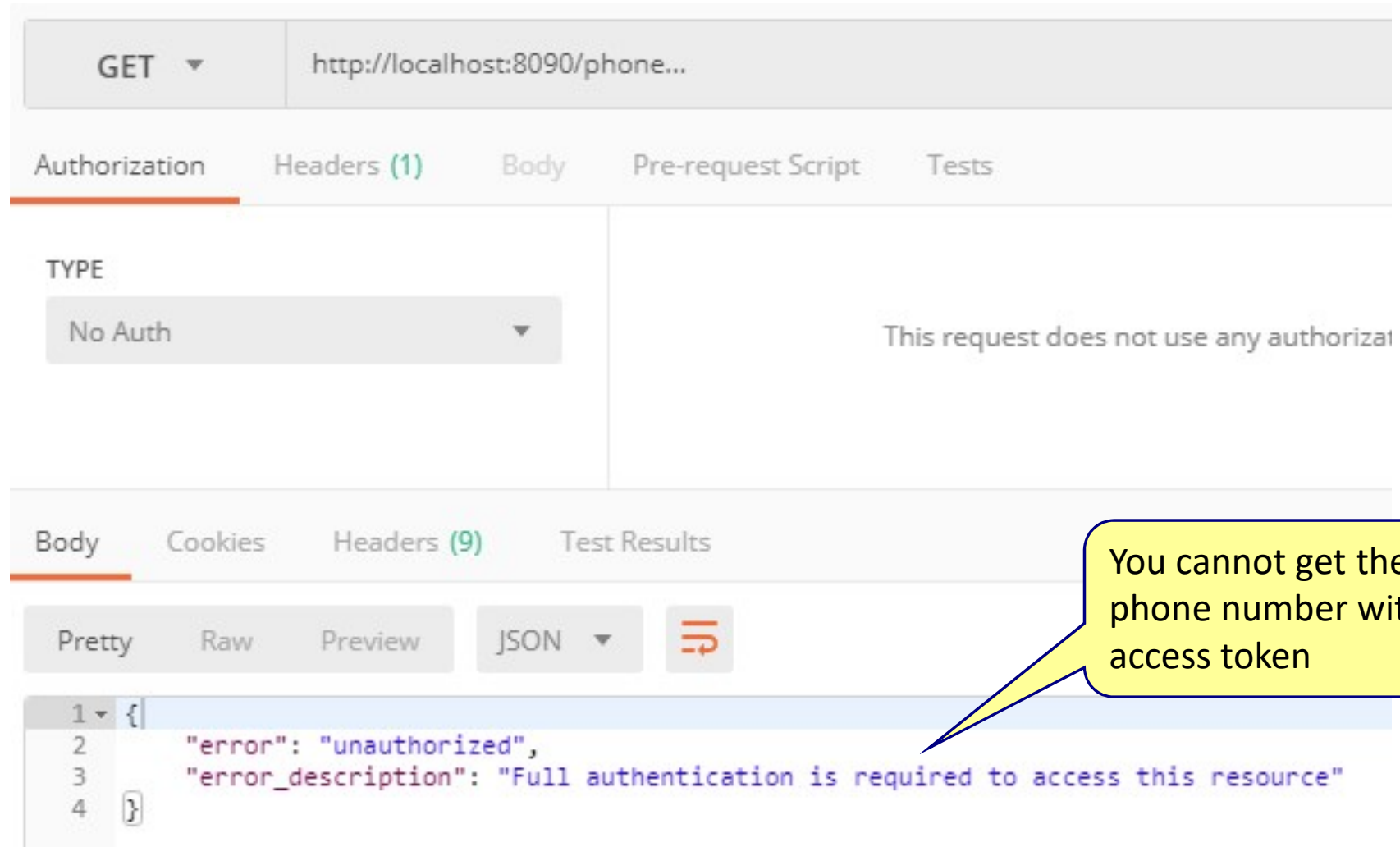
The screenshot shows a web browser's developer tools interface. At the top, a GET request is shown to the URL `http://localhost:8090/name...`. Below this, the 'Authorization' tab is selected, showing 'No Auth' under the 'TYPE' dropdown. The 'Body' tab is also visible, showing the response body 'Frank Brown'. The 'Headers' tab is selected, showing 9 headers. The 'Test Results' tab is also visible. The response body is displayed in a table with one row: '1 Frank Brown'. A yellow callout bubble points to the response body with the text: 'Everyone can get the phone number without authorization'.

TYPE
No Auth

Body
1 Frank Brown



# Get the phone number



The screenshot shows a REST client interface with a GET request to `http://localhost:8090/phone...`. The 'Authorization' tab is selected, showing 'No Auth' and a message: 'This request does not use any authorization'. The 'Body' tab is also visible, showing a JSON response in 'Pretty' format:

```
1 {  
2   "error": "unauthorized",  
3   "error_description": "Full authentication is required to access this resource"  
4 }
```

A yellow callout bubble points to the response body with the text: '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
<input checked="" type="checkbox"/> Authorization	Bearer e964de5e-d5c7-4f9d-89e3-66eea2b9d507
New key	Value

Body Cookies Headers (9) Test Results

Pretty Raw Preview Text

```
1 645322899
```

Frank can get the phone number

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

Authorization Headers (1) Body Pre-request Script Tests

Key	Value
<input checked="" type="checkbox"/> Authorization	Bearer 17628fbc-0965-406b-a7b2-c27a29600573
New key	Value

Body Cookies Headers (9) Test Results

Pretty Raw Preview Text

```
1 645322899
```

John can get the phone number

# Get the salary

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

Authorization Headers (1) Body Pre-request Script Tests

Key	Value
<input checked="" type="checkbox"/> Authorization	Bearer e964de5e-d5c7-4f9d-89e3-66eea2b9d507
New key	Value

Body Cookies Headers (9) Test Results

Pretty Raw Preview Text

```
1 95.000
```

Frank can get the salary  
(role=MANAGER)

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

Authorization Headers (1) Body Pre-request Script Tests

Key	Value
<input checked="" type="checkbox"/> Authorization	Bearer 17628fbc-0965-406b-a7b2-c27a29600573
New key	Value

Body Cookies Headers (8) Test Results

Pretty Raw Preview JSON

```
1 {  
2   "error": "access_denied",  
3   "error_description": "Access is denied"  
4 }
```

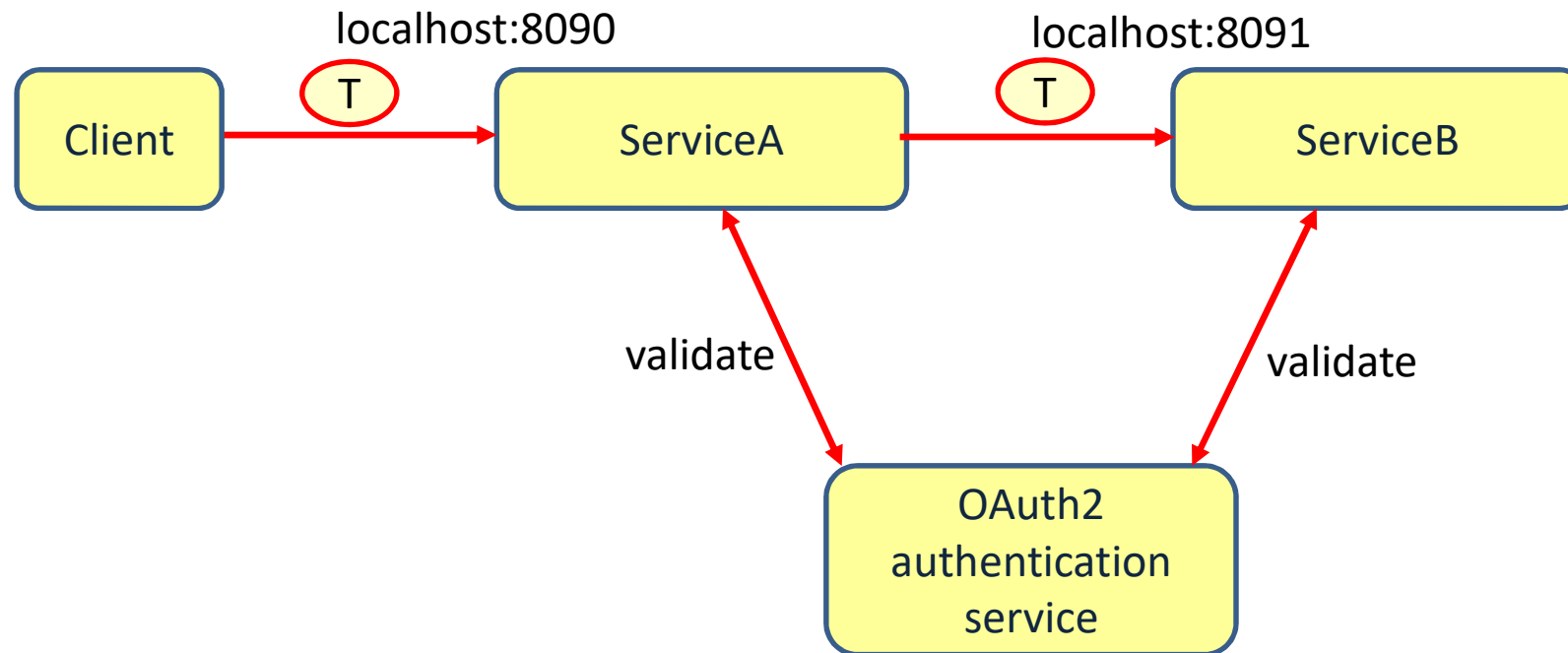
John cannot get the  
salary (role = USER)

# PROPAGATING THE TOKEN





# Propagate the token



# Secure application B

```
@SpringBootApplication
public class SecureServiceBApplication {

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

```
@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
<input checked="" type="checkbox"/> Authorization	Bearer e964de5e-d5c7-4f9d-89e3-66eea2b9d507
New key	Value

Body Cookies Headers (9) Test Results

Pretty Raw Preview Text

1 This info is for managers

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

Authorization Headers (1) Body Pre-request Script Tests

Key	Value
<input checked="" type="checkbox"/> Authorization	Bearer e964de5e-d5c7-4f9d-89e3-66eea2b9d507
New key	Value

Body Cookies Headers (9) Test Results

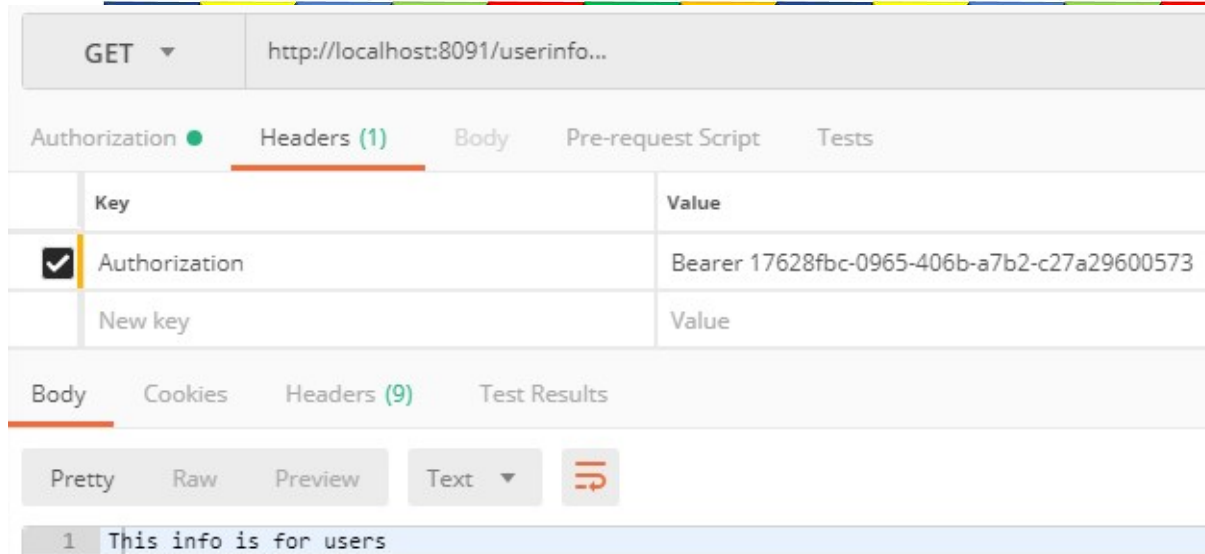
Pretty Raw Preview Text

1 This info is for users

Frank can get all info

Frank can get all info

# One services calls another service



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

Authorization Headers (1) Body Pre-request Script Tests

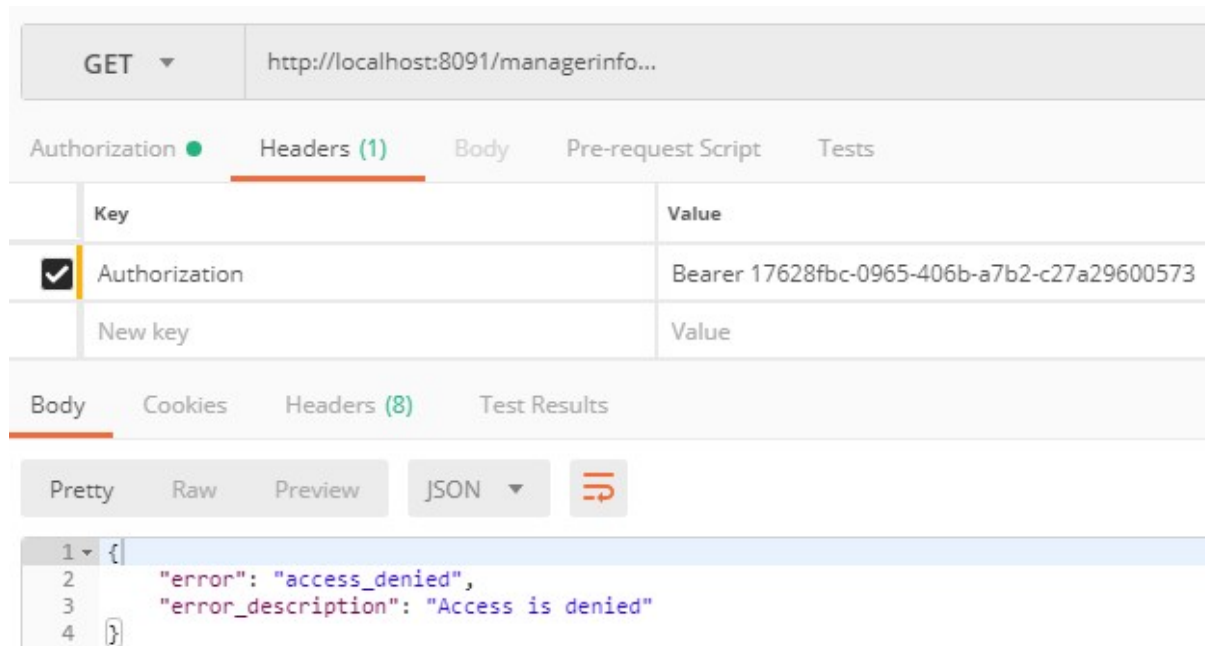
Key	Value
Authorization	Bearer 17628fbc-0965-406b-a7b2-c27a29600573
New key	Value

Body Cookies Headers (9) Test Results

Pretty Raw Preview Text

```
1 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

```
1 {
2   "error": "access_denied",
3   "error_description": "Access is denied"
4 }
```

John cannot get managerinfo



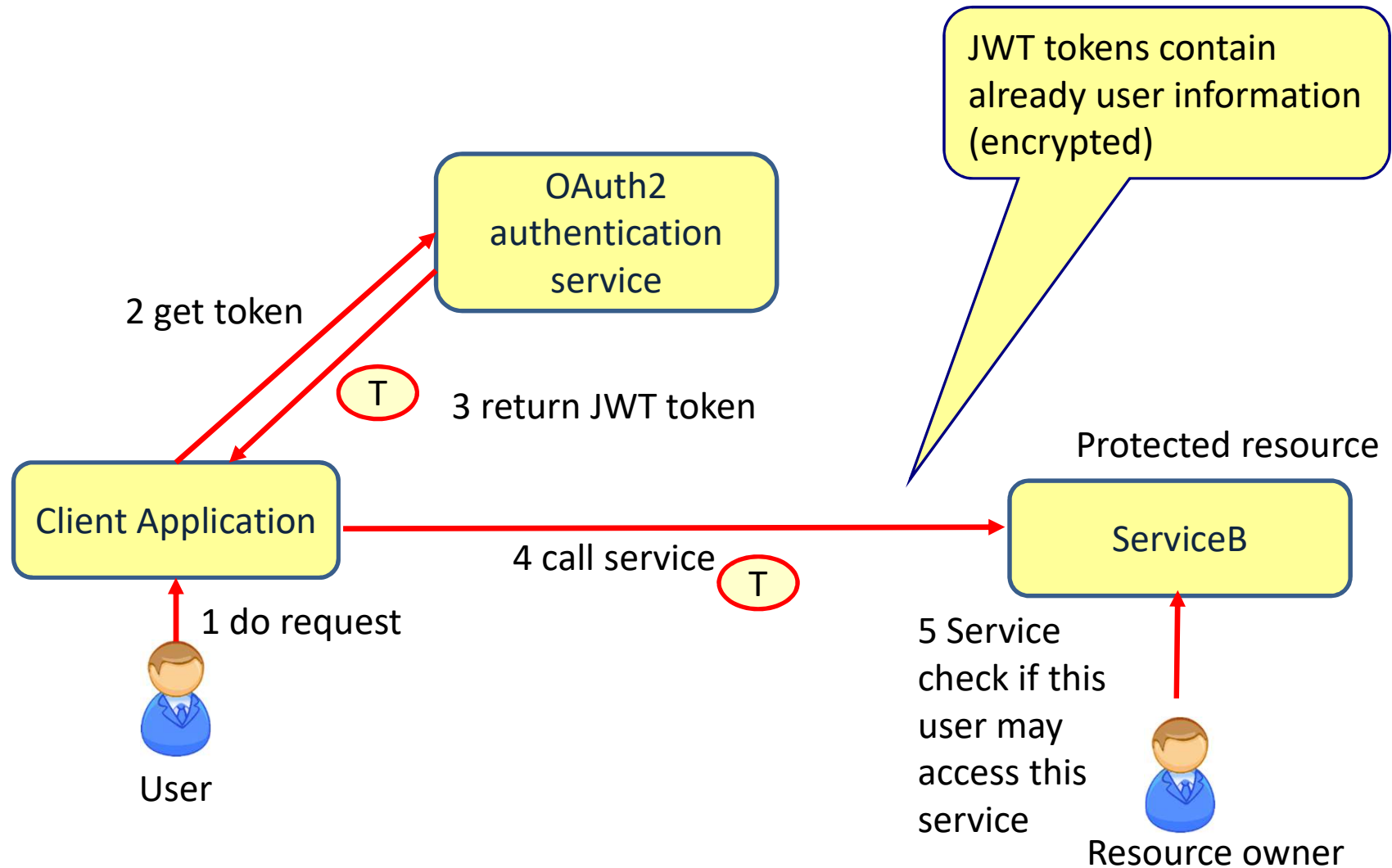
# JWT tokens

---

- OAuth2 is a token based authorization framework but 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



# 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.

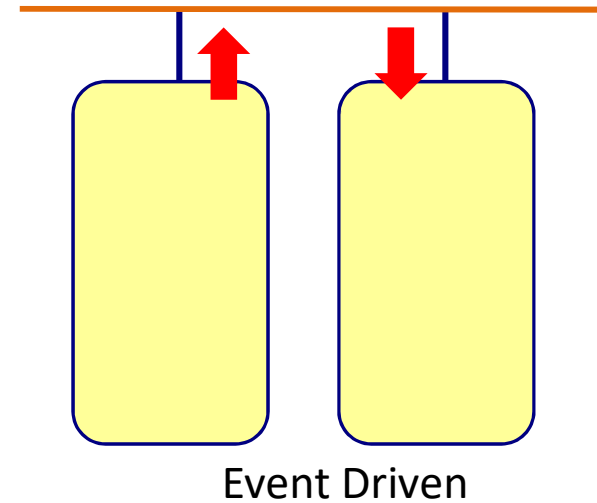


# EVENT DRIVEN ARCHITECTURE



# Event Driven Architecture (EDA)

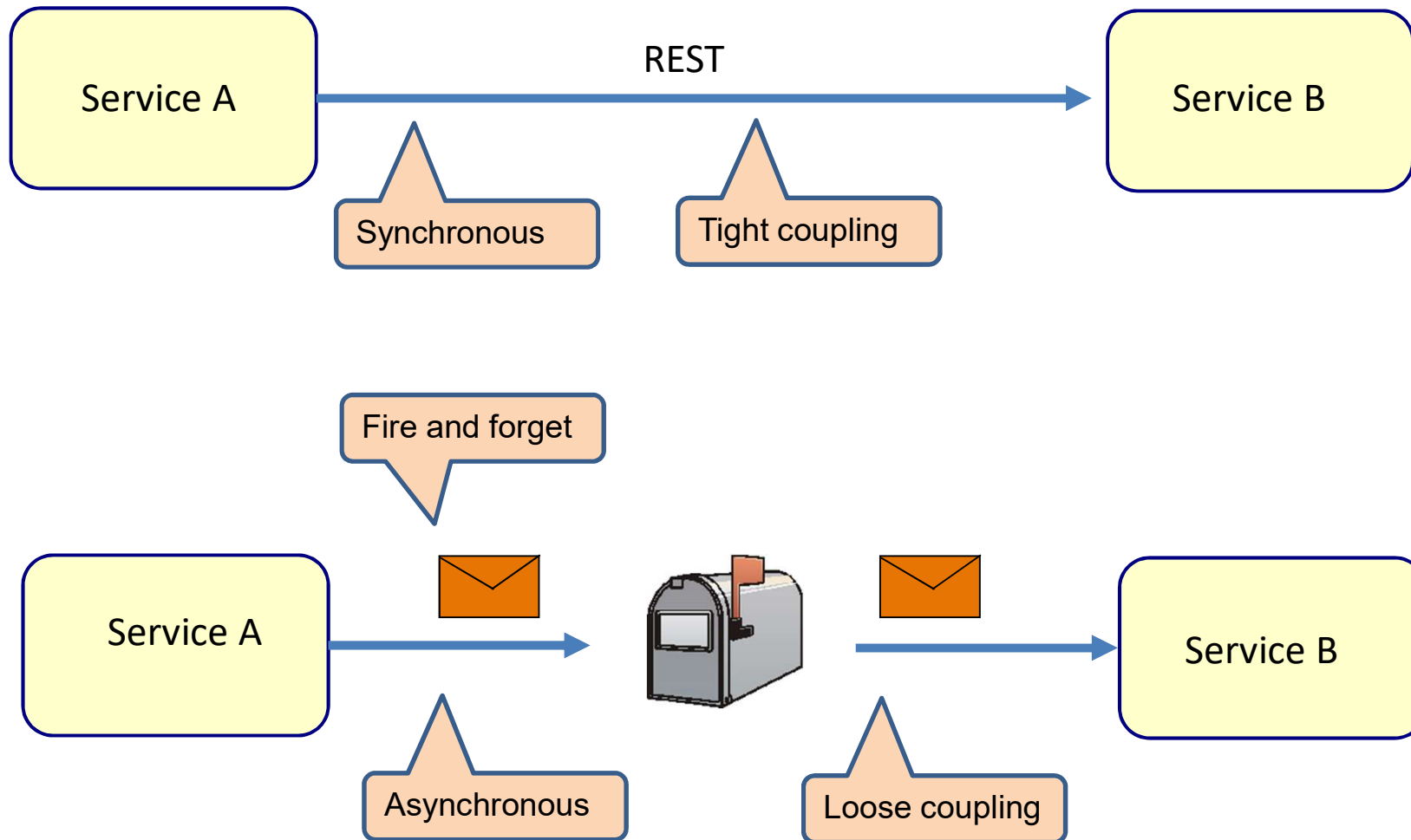
---



- Applications publish events
- Applications subscribe to events

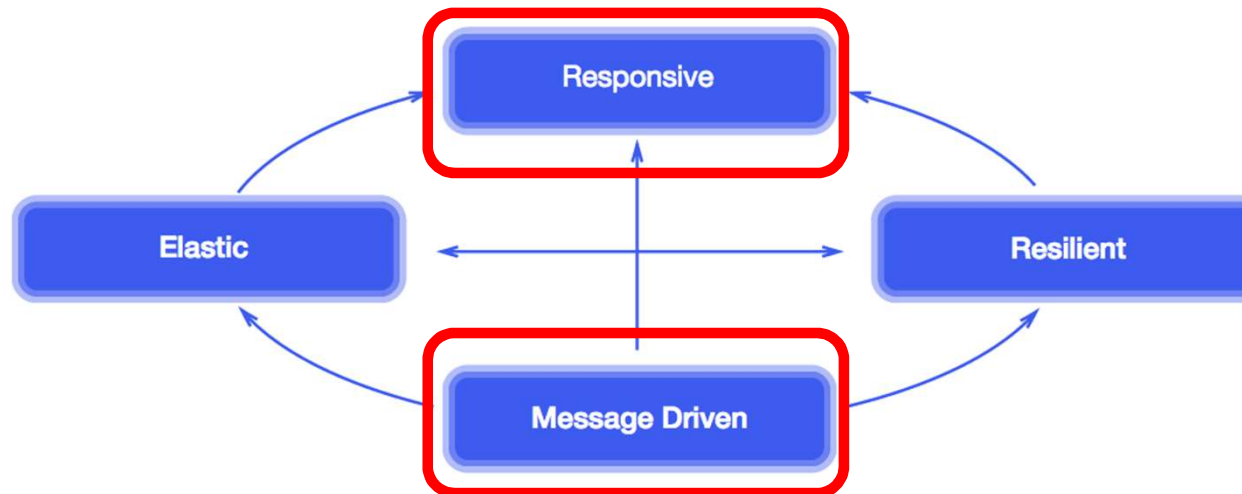


# REST vs. messaging

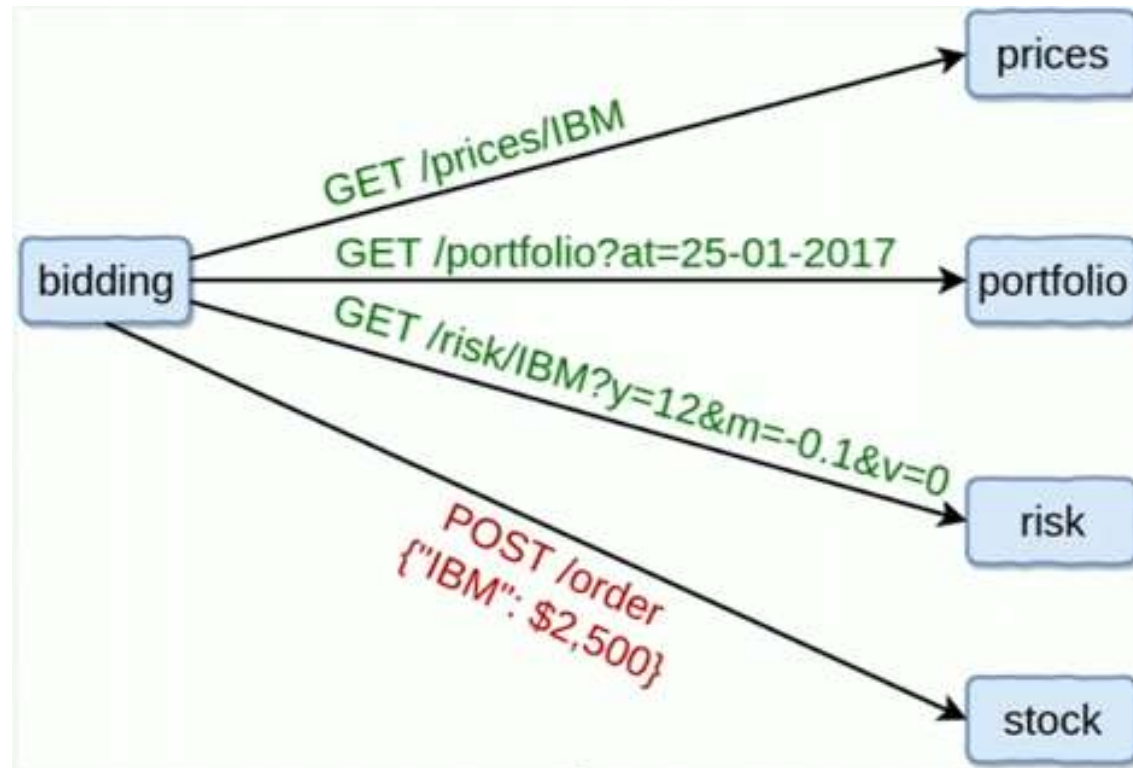


# Reactive applications

---



# Synchronous architecture

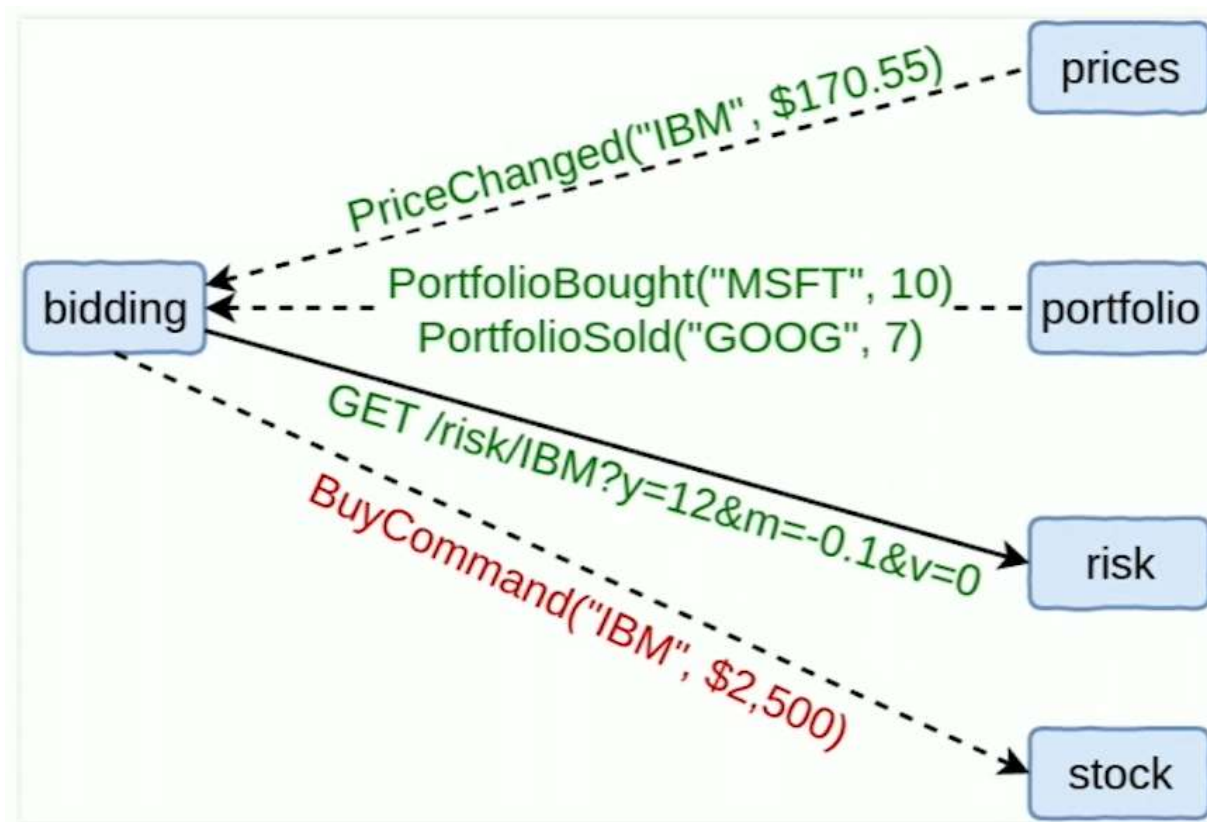


- Slow
- If one of the components go down, we cannot order
- Temporal coupling
  - Two services have to exist at the same time to perform some functionality





# Event driven architecture

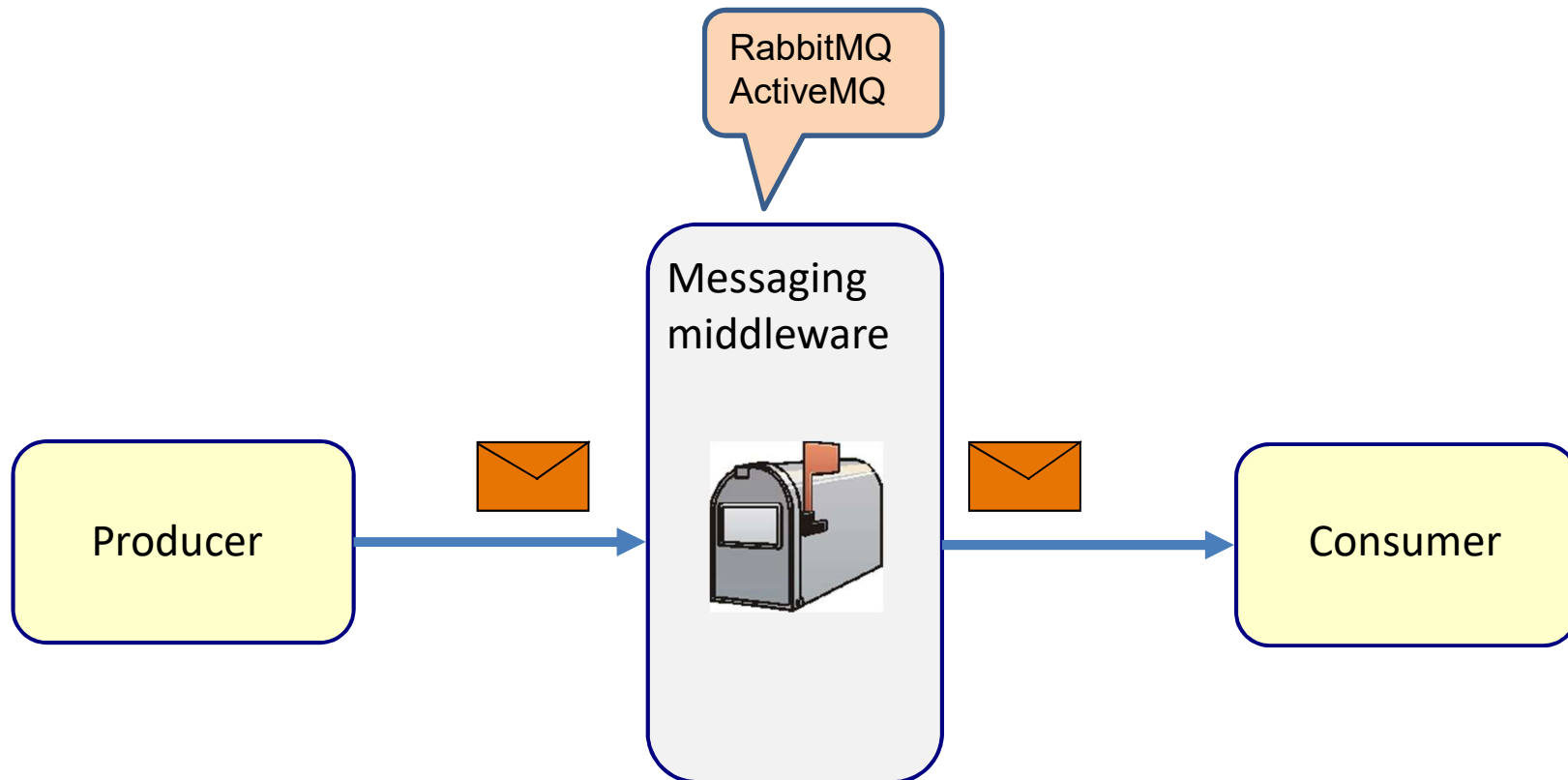


- Much faster
- Less dependencies

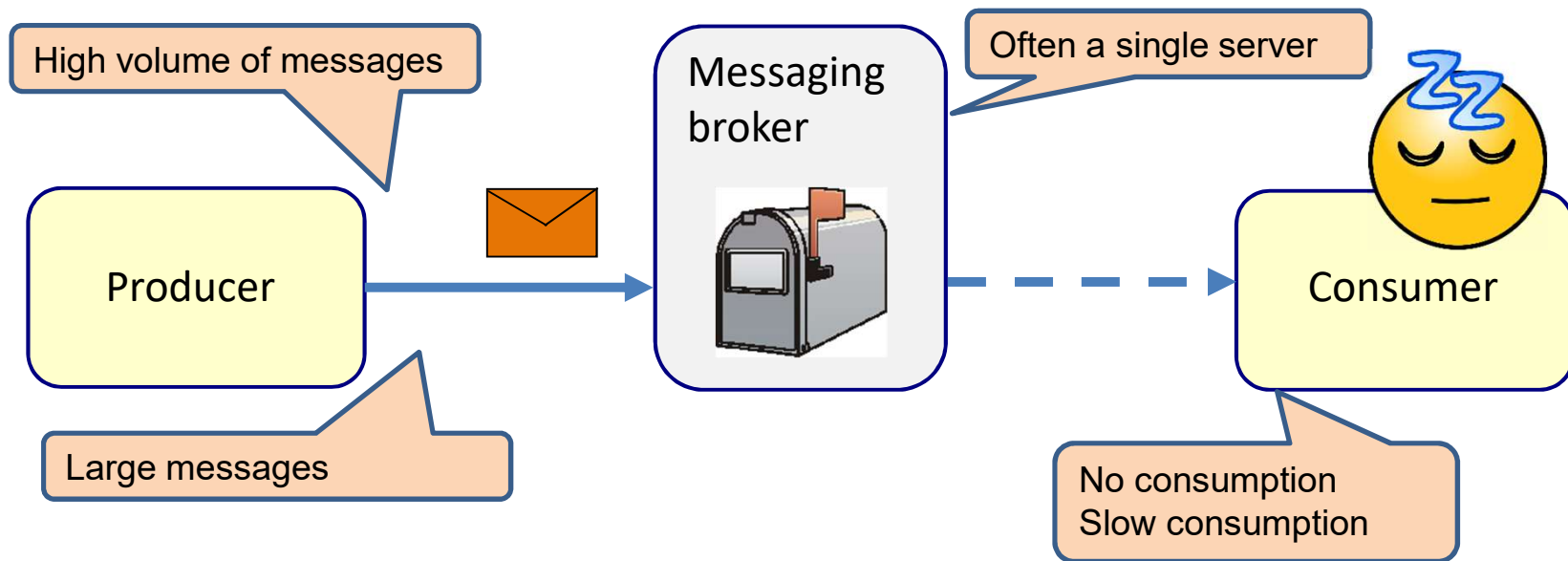


# Traditional Messaging Systems

---



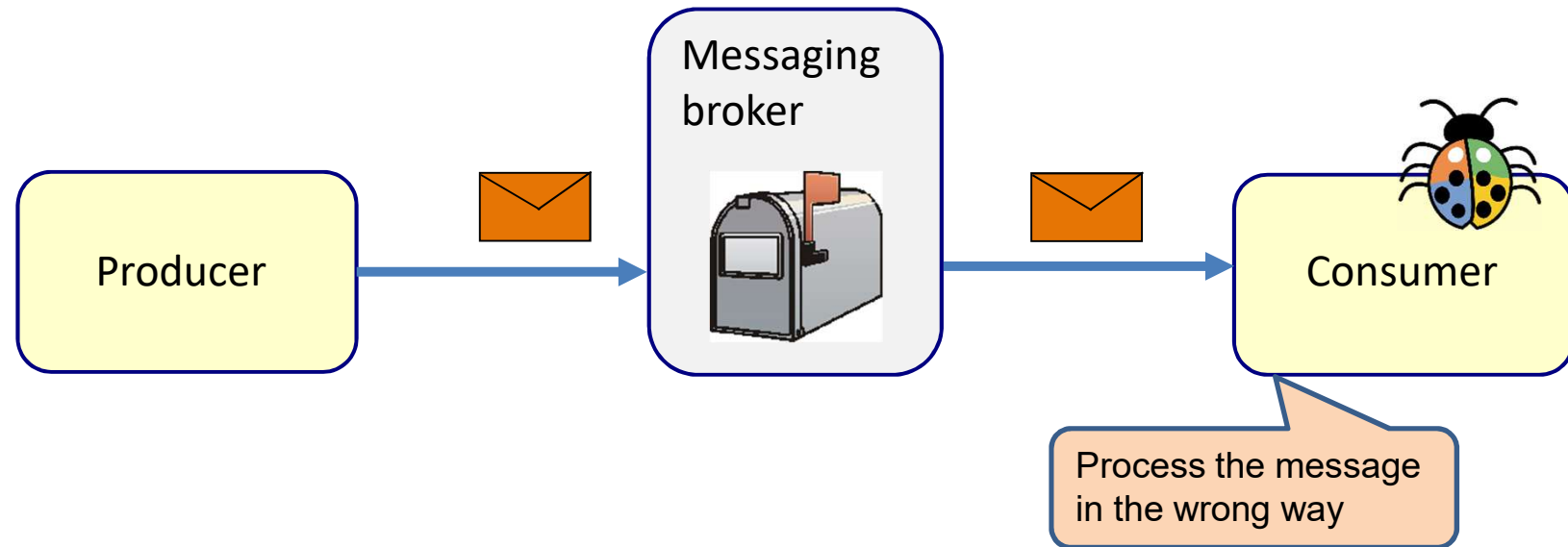
# Problems with traditional messaging middleware



- 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



# 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:

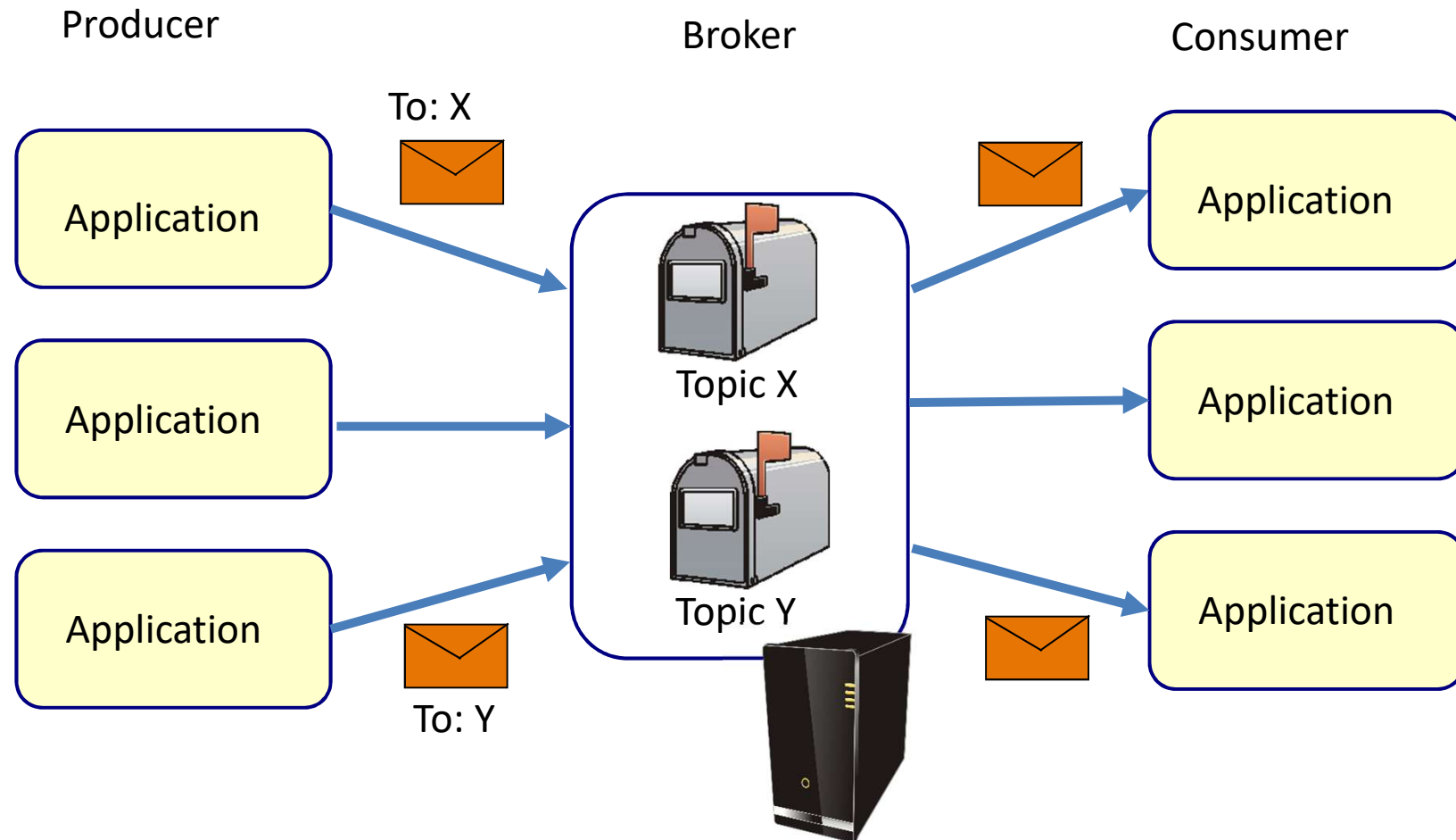
- Over 1.4 trillion messages per day
- 175 terabytes per day

High Velocity:

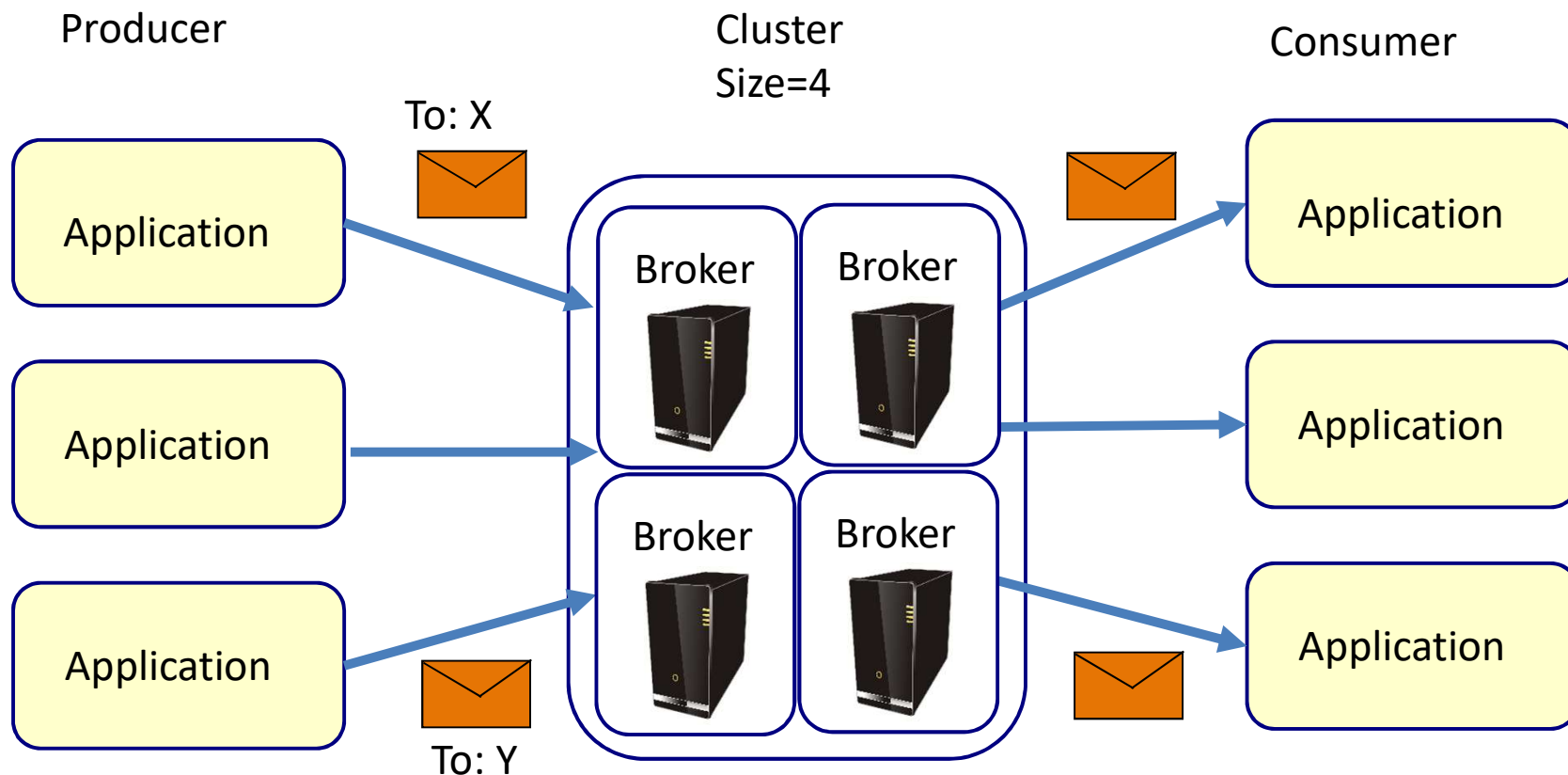
- Peak 13 million messages per second
- 2.75 gigabytes per second



# Kafka



# Cluster of Brokers



# Apache Zookeeper

---

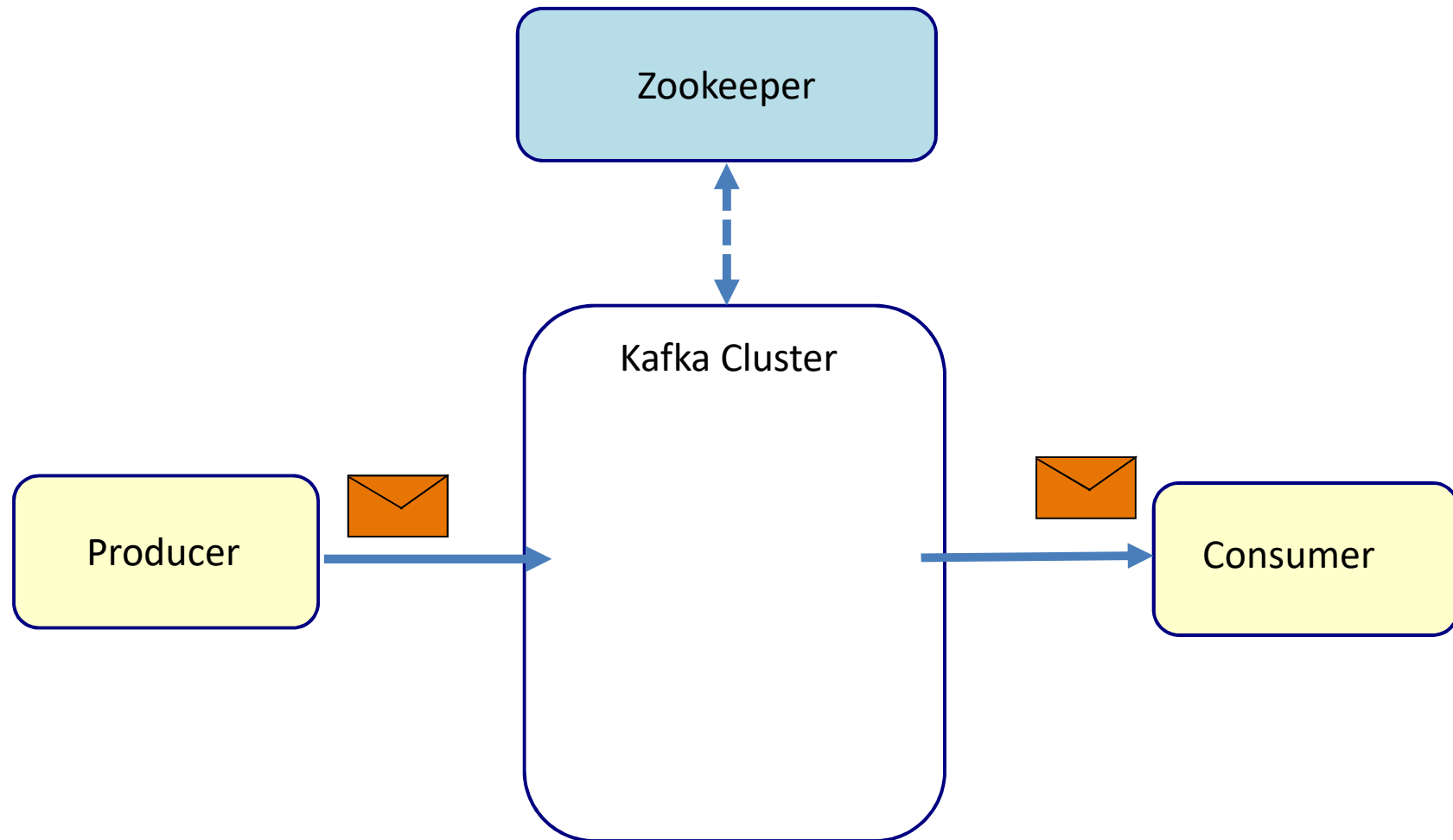
- Maintains metadata about a cluster of distributed nodes
  - Configuration information
  - Health status
  - Group membership





# Kafka distributed architecture

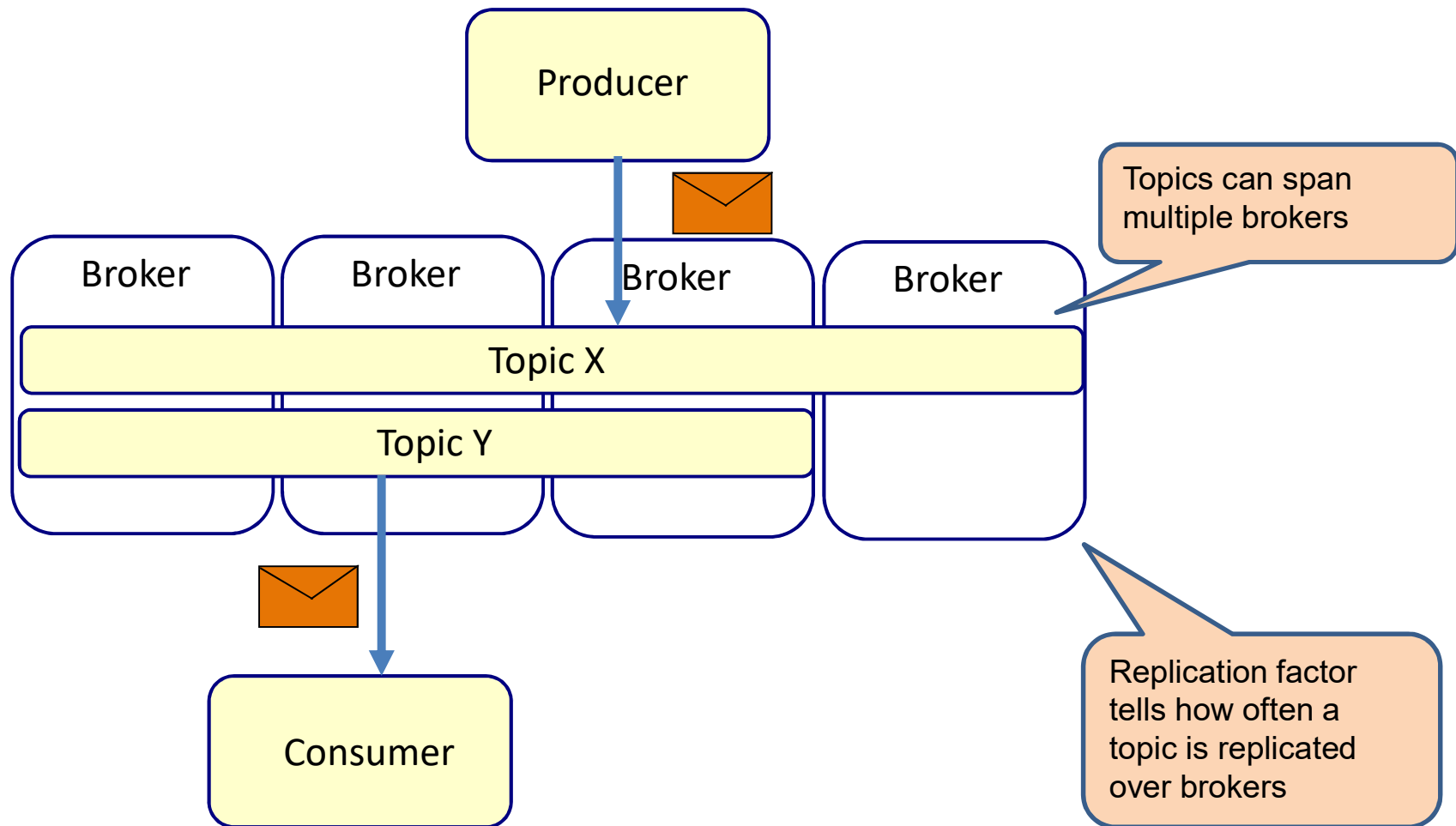
---



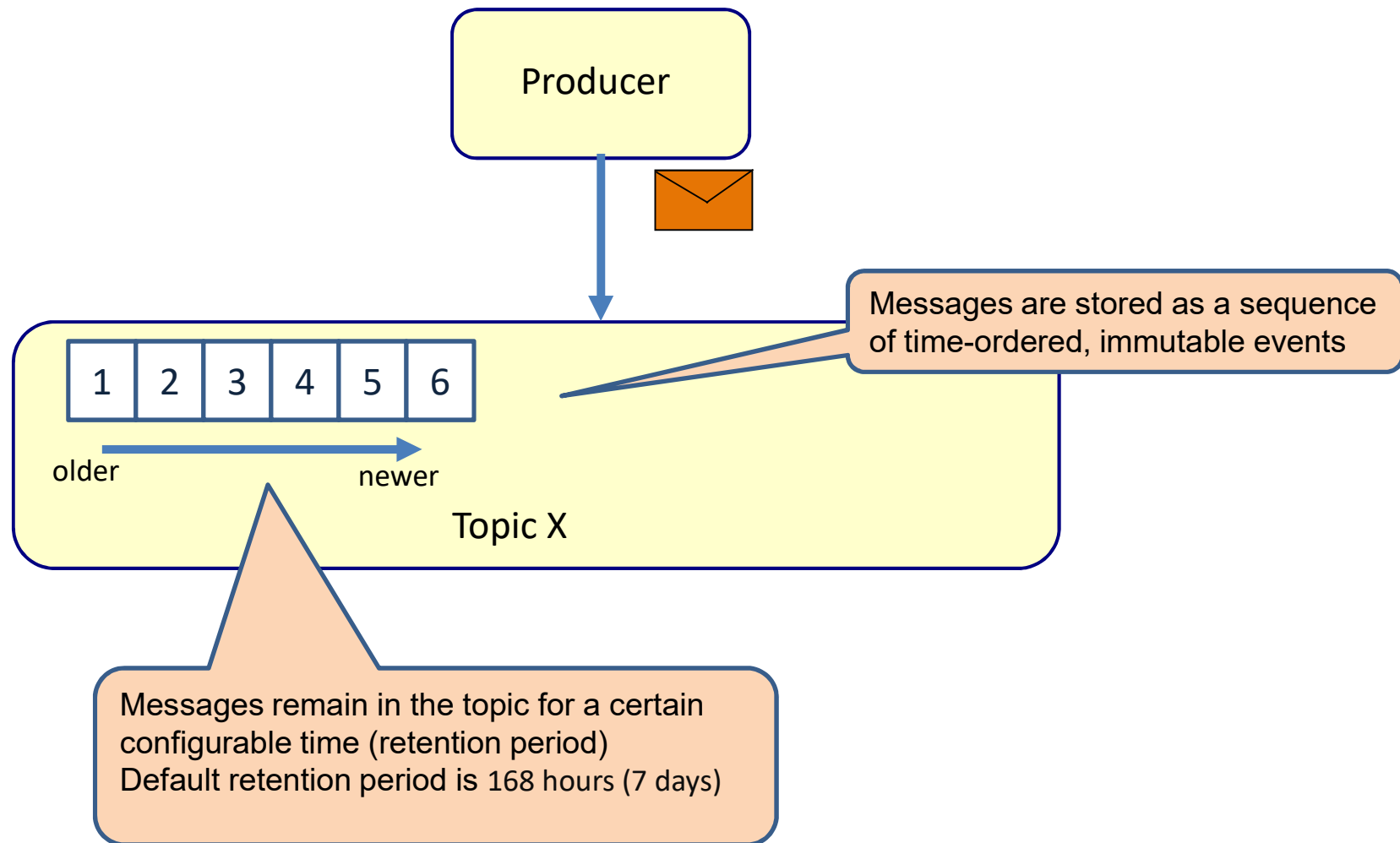
# TOPIC



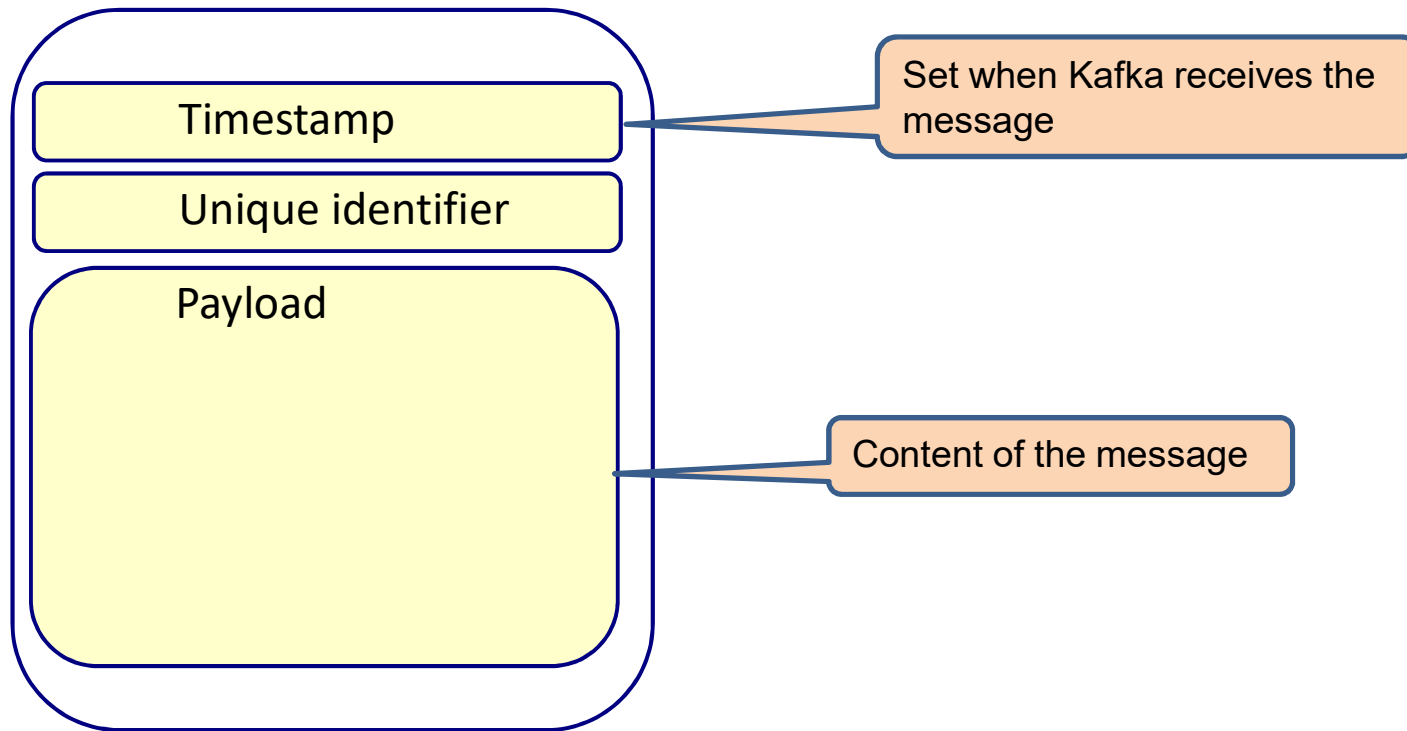
# Topics



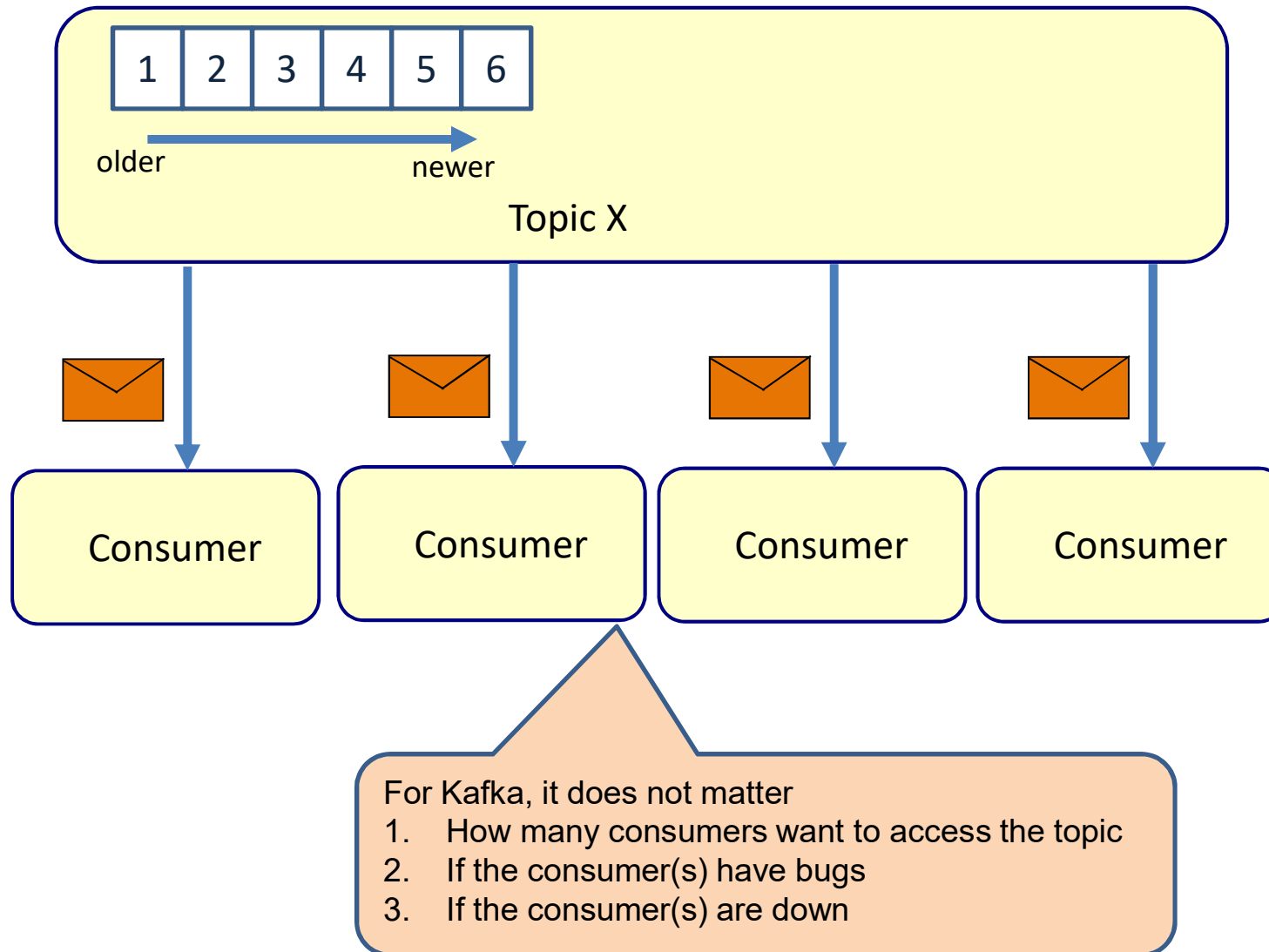
# Event sourcing



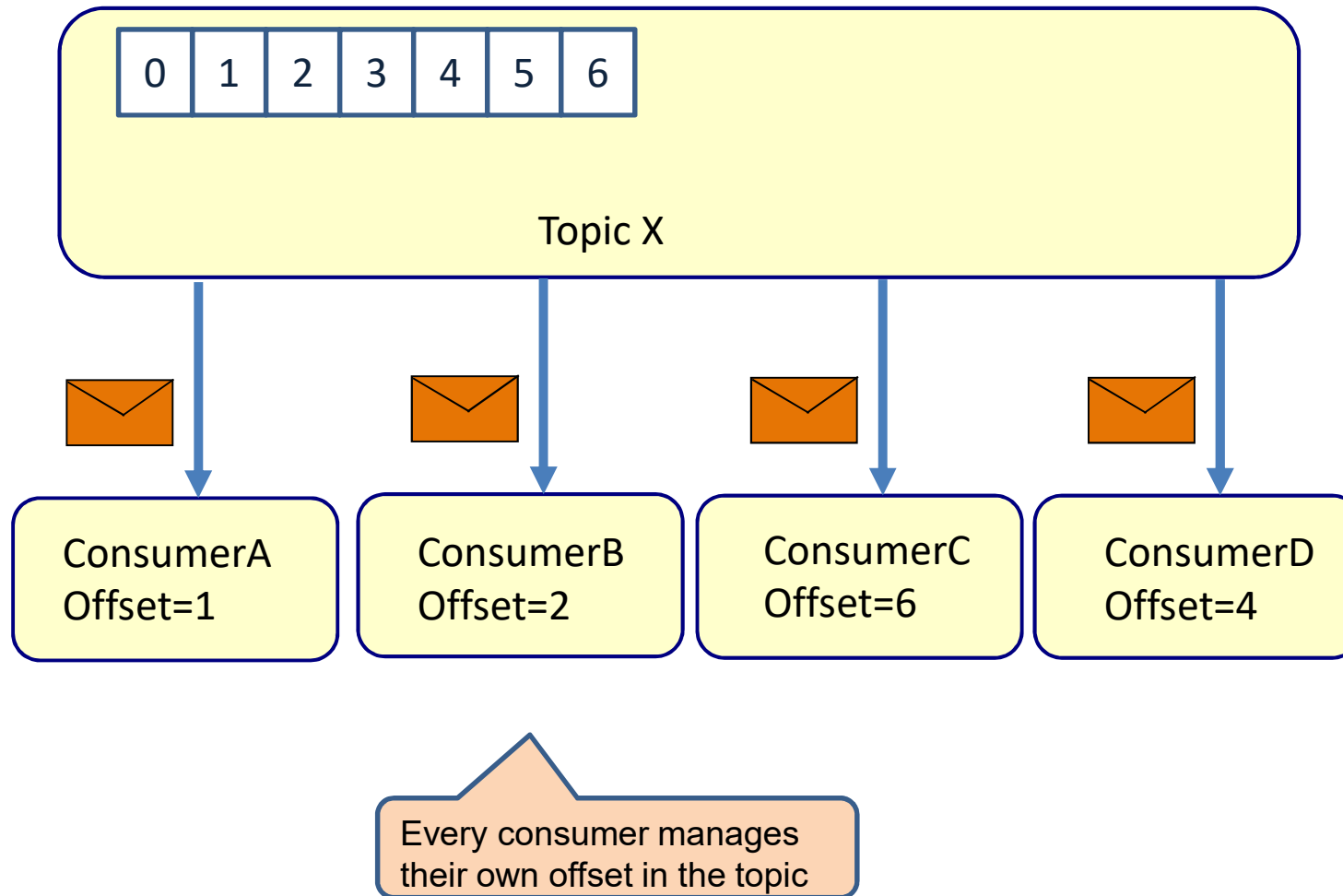
# Message



# Consumers of a Topic



# Offset



# Partition

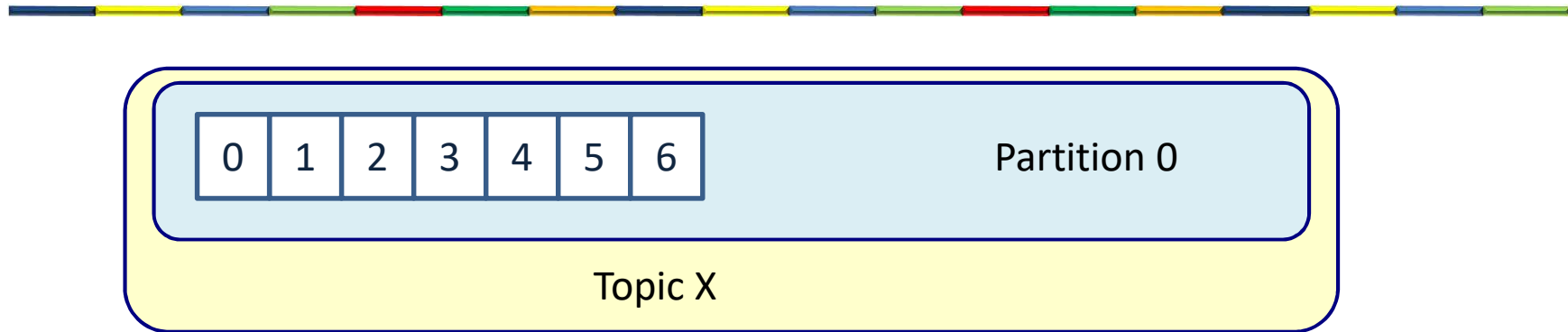


- Each topic has one or more partitions
  - This is configurable
- Each partition is maintained on 1 or more brokers





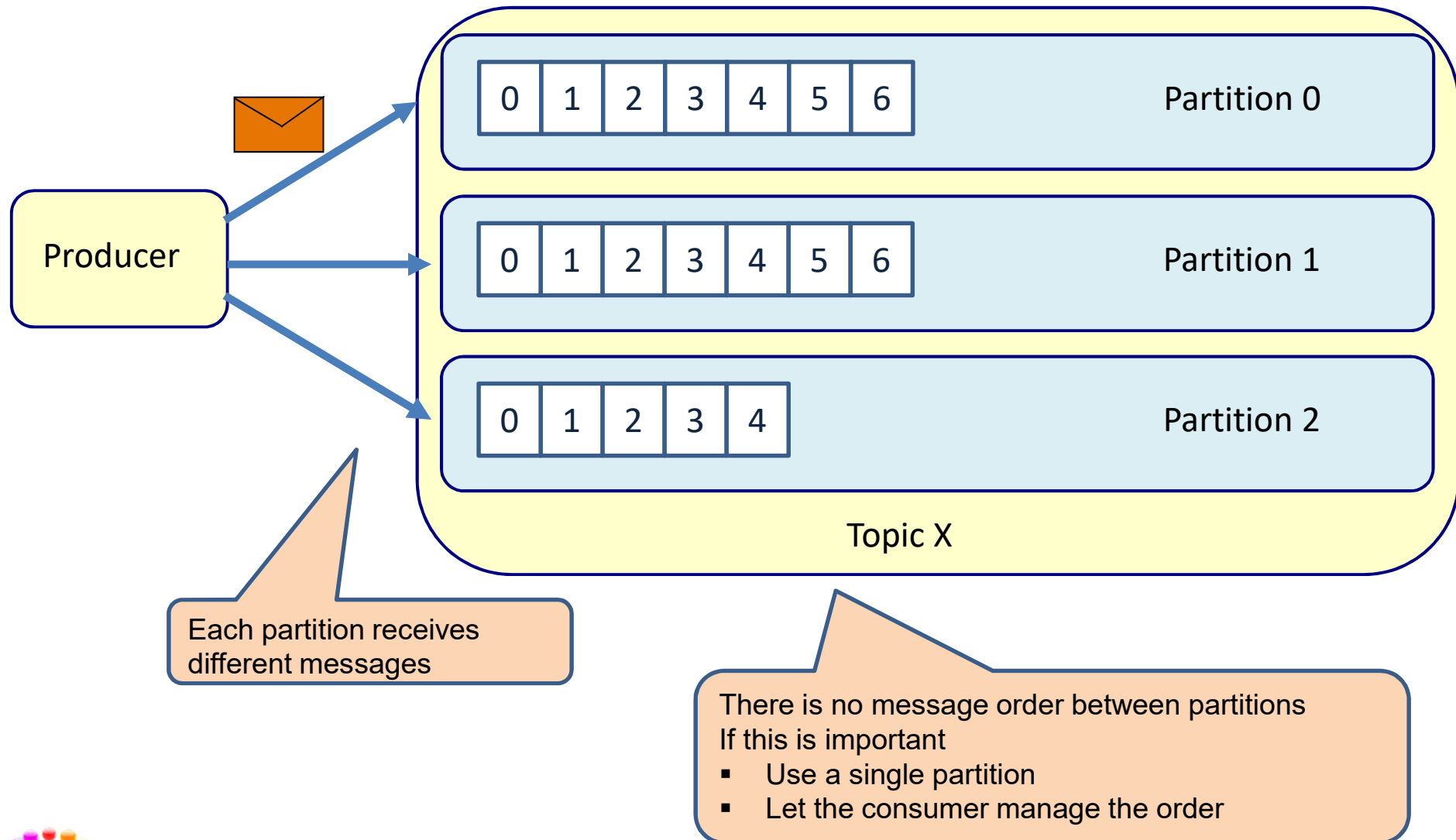
# 1 partition



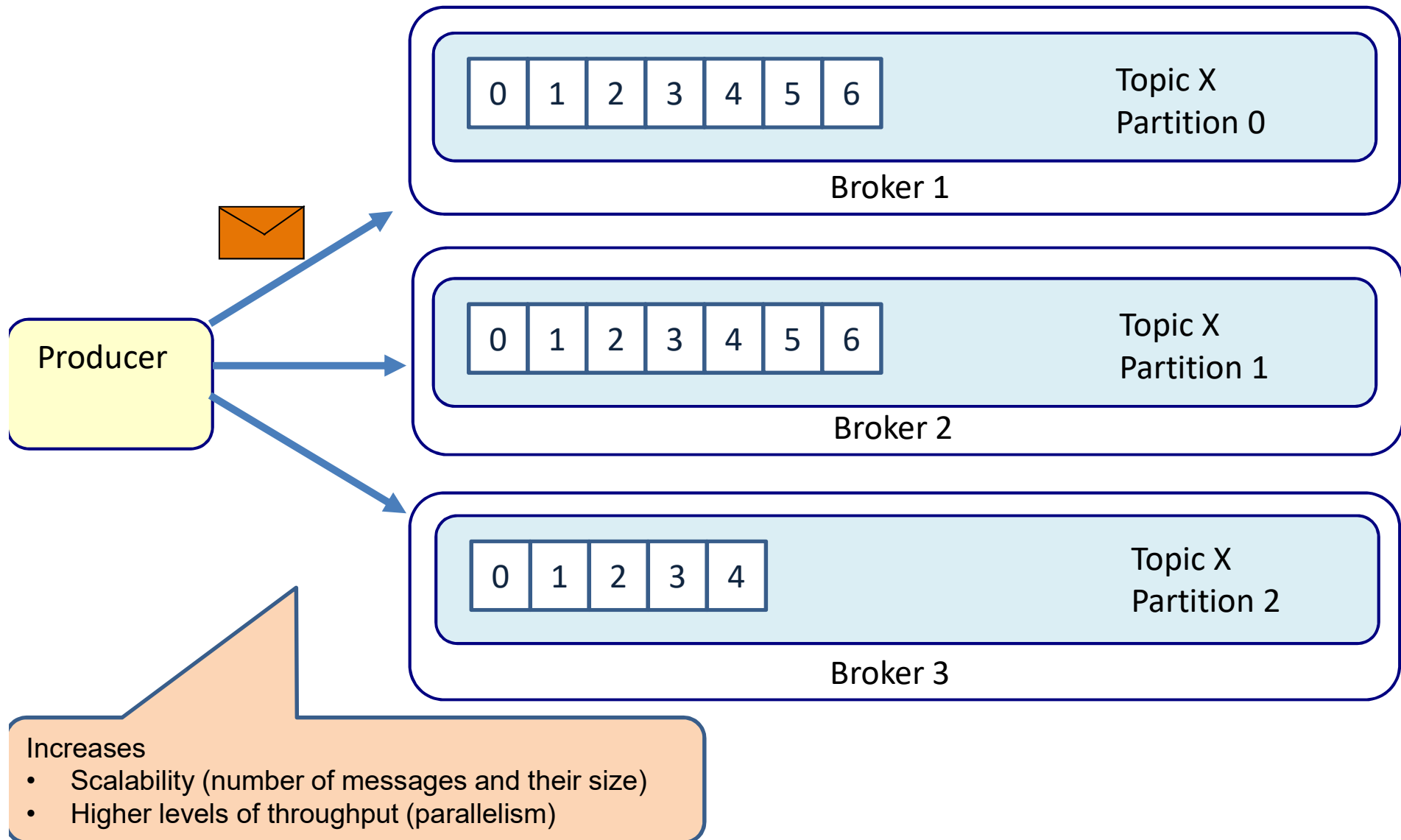
- Each partition must fit on 1 broker



# 3 partitions



# Scale out partitions



# 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

```
@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.

