Lesson 11

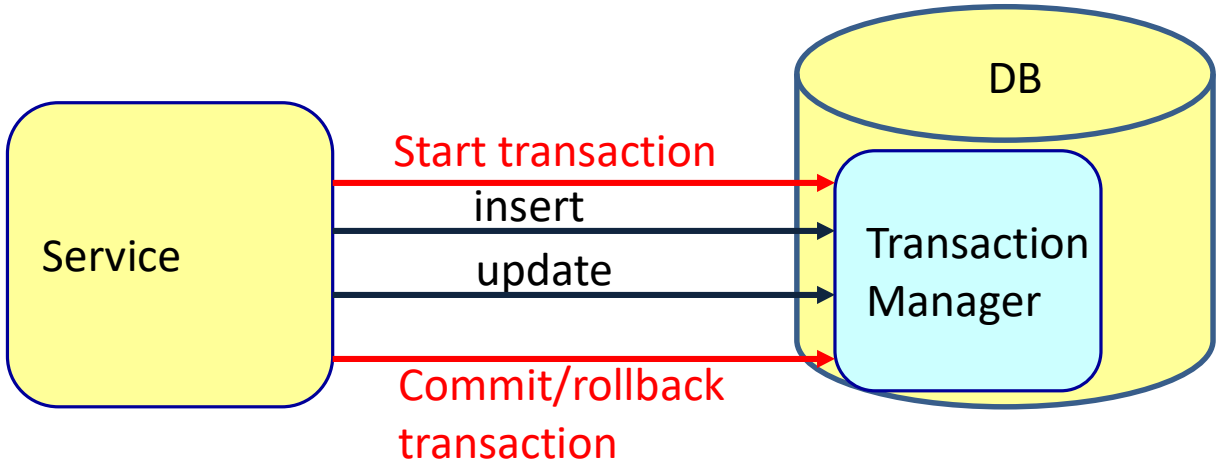
MICROSERVICES

TRANSACTIONS

Transactions

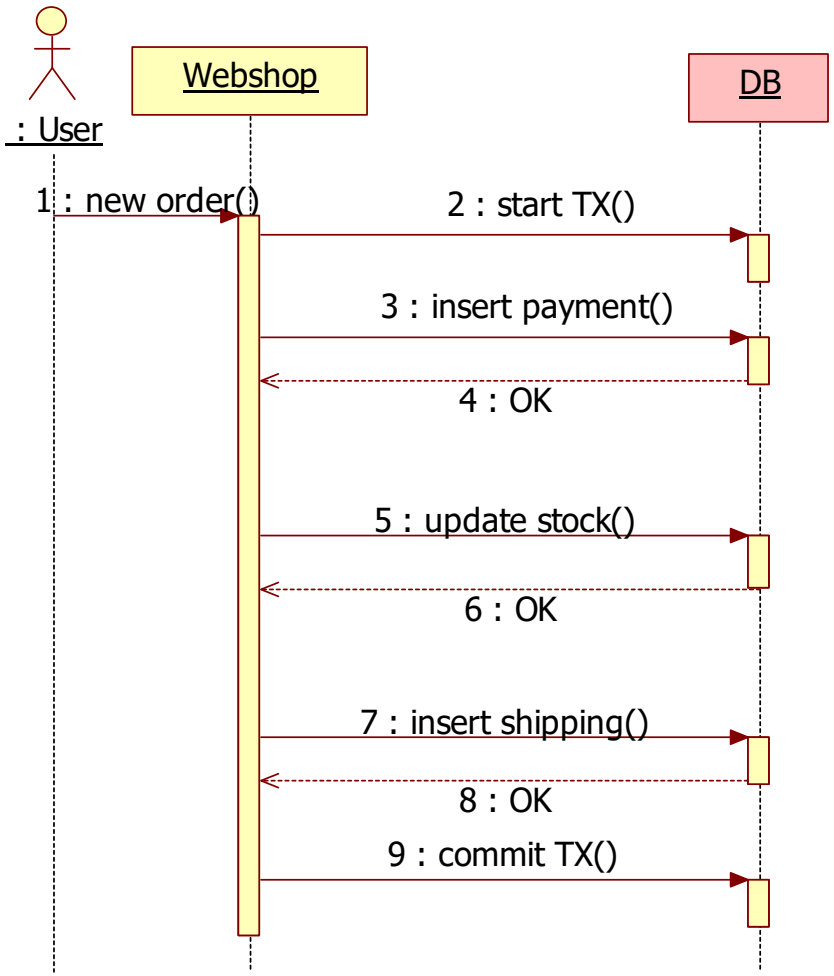
- A Transaction is a unit of work that is:
  - **ATOMIC:** The transaction is considered a single unit, either the entire transaction completes, or the entire transaction fails.
  - **CONSISTENT:** A transaction transforms the database from one consistent state to another consistent state
  - **ISOLATED:** Data inside a transaction can not be changed by another concurrent processes until the transaction has been committed
  - **DURABLE:** Once committed, the changes made by a transaction are persistent

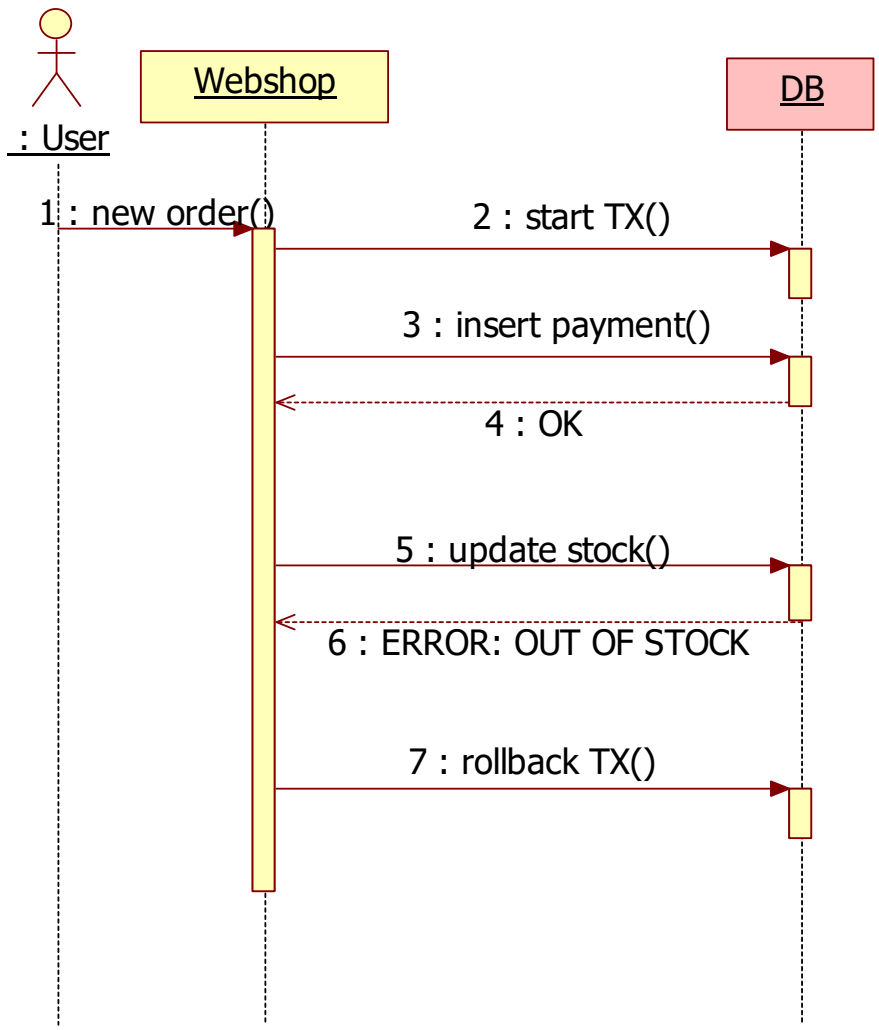
Local transaction



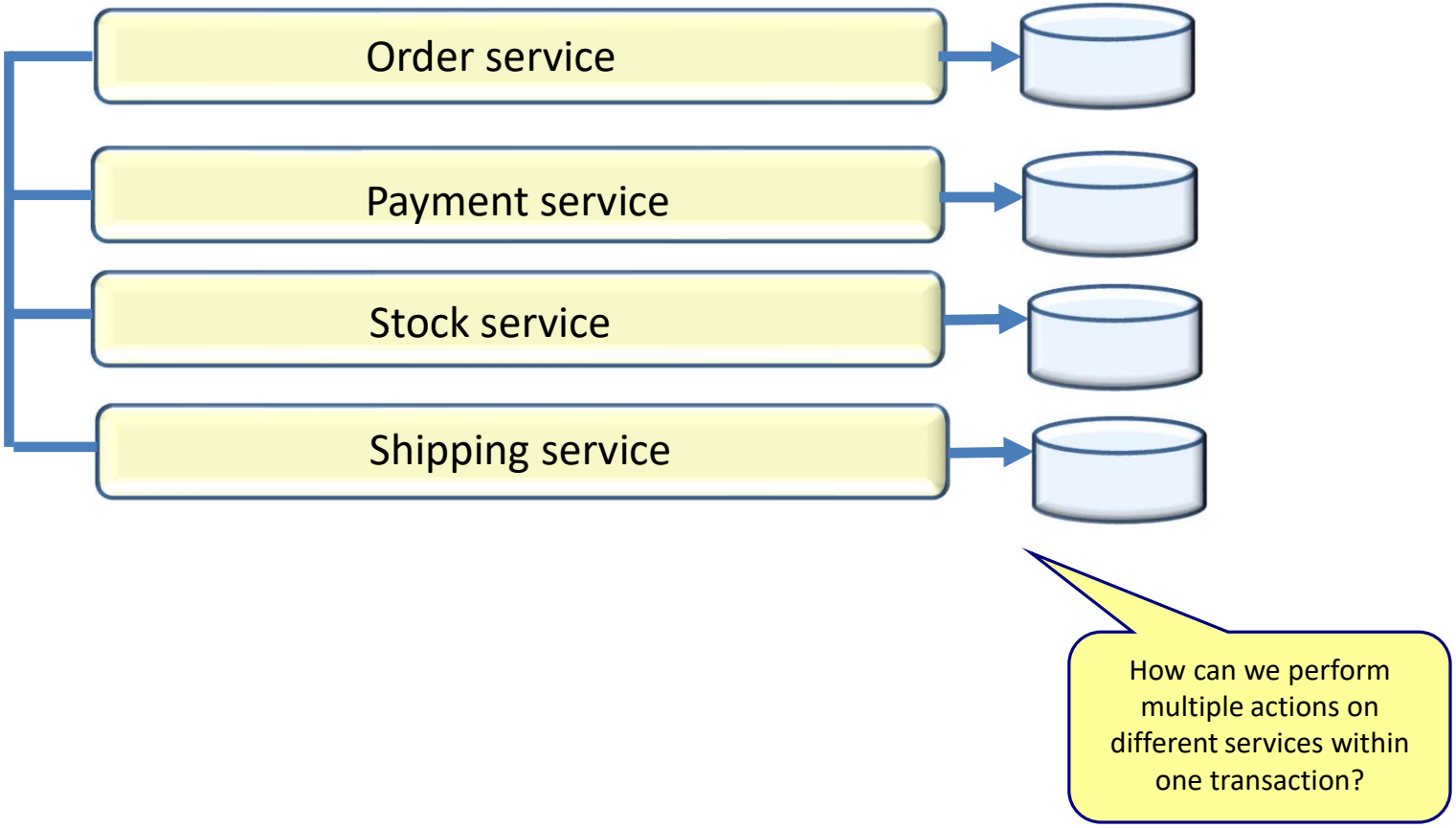
- The transaction is managed by the database
  - Simple
  - Fast
- Always try to keep transaction boundaries within a service

Local transaction



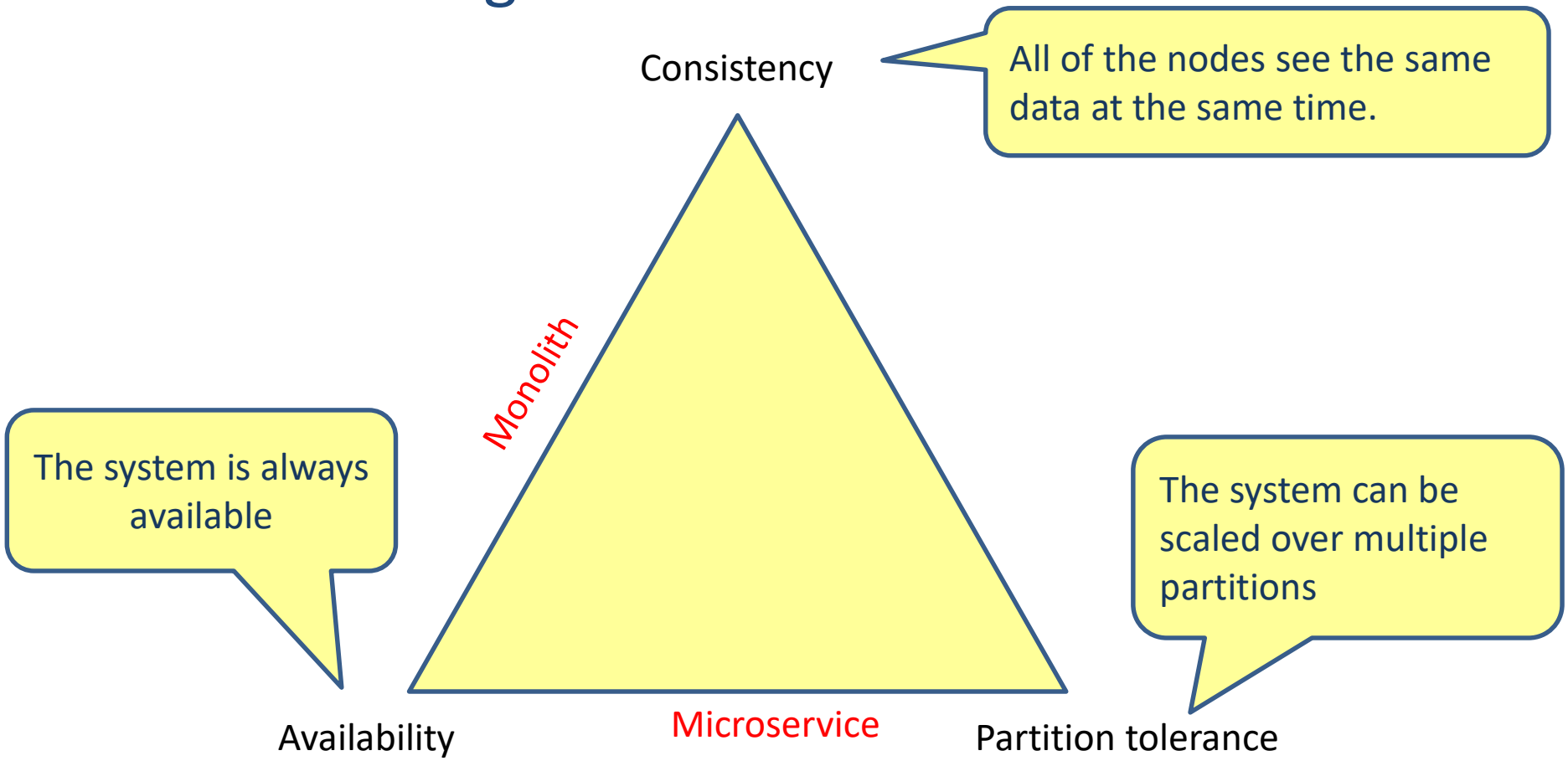


Distributed transactions

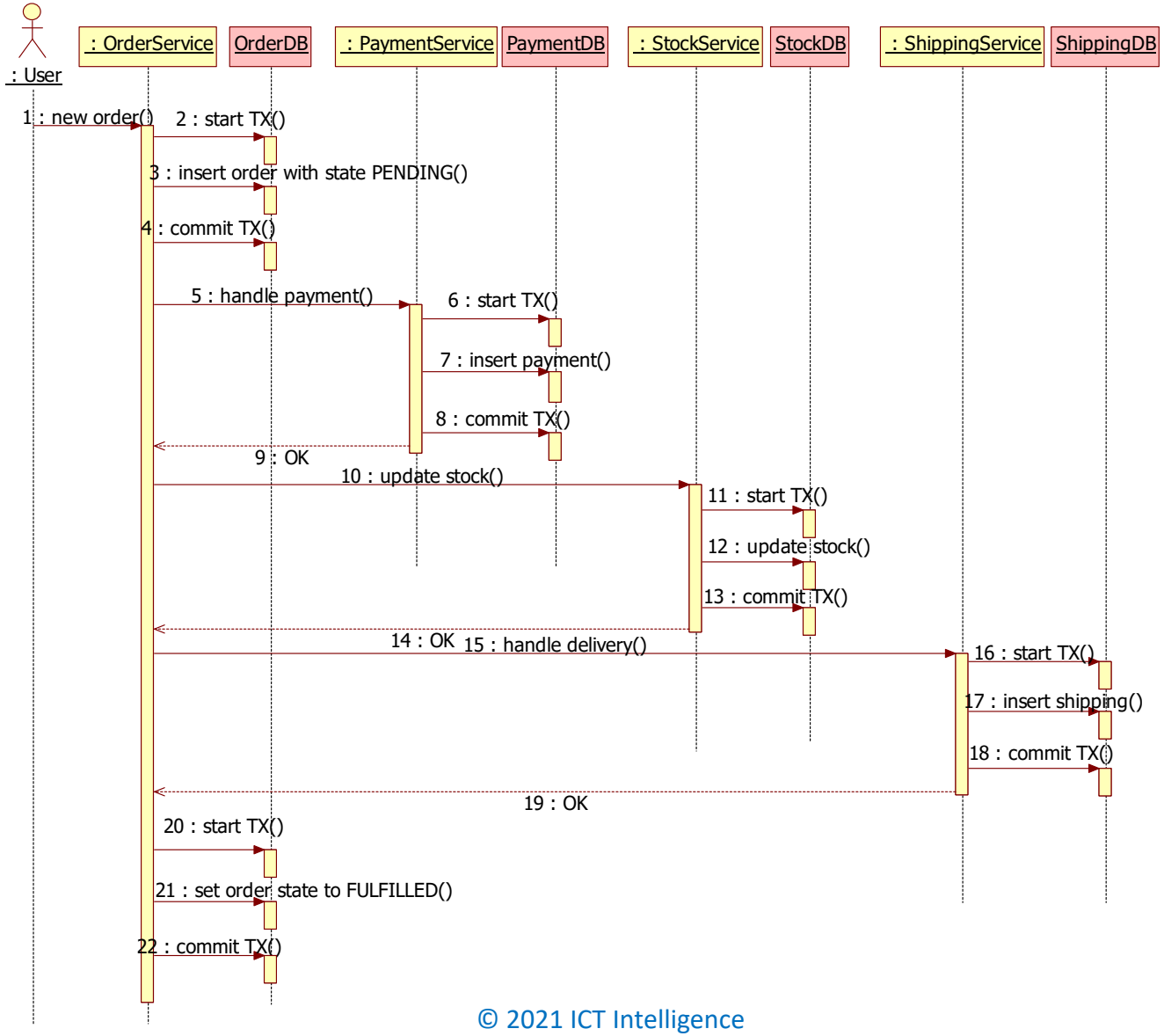


Brewer's CAP Theorem

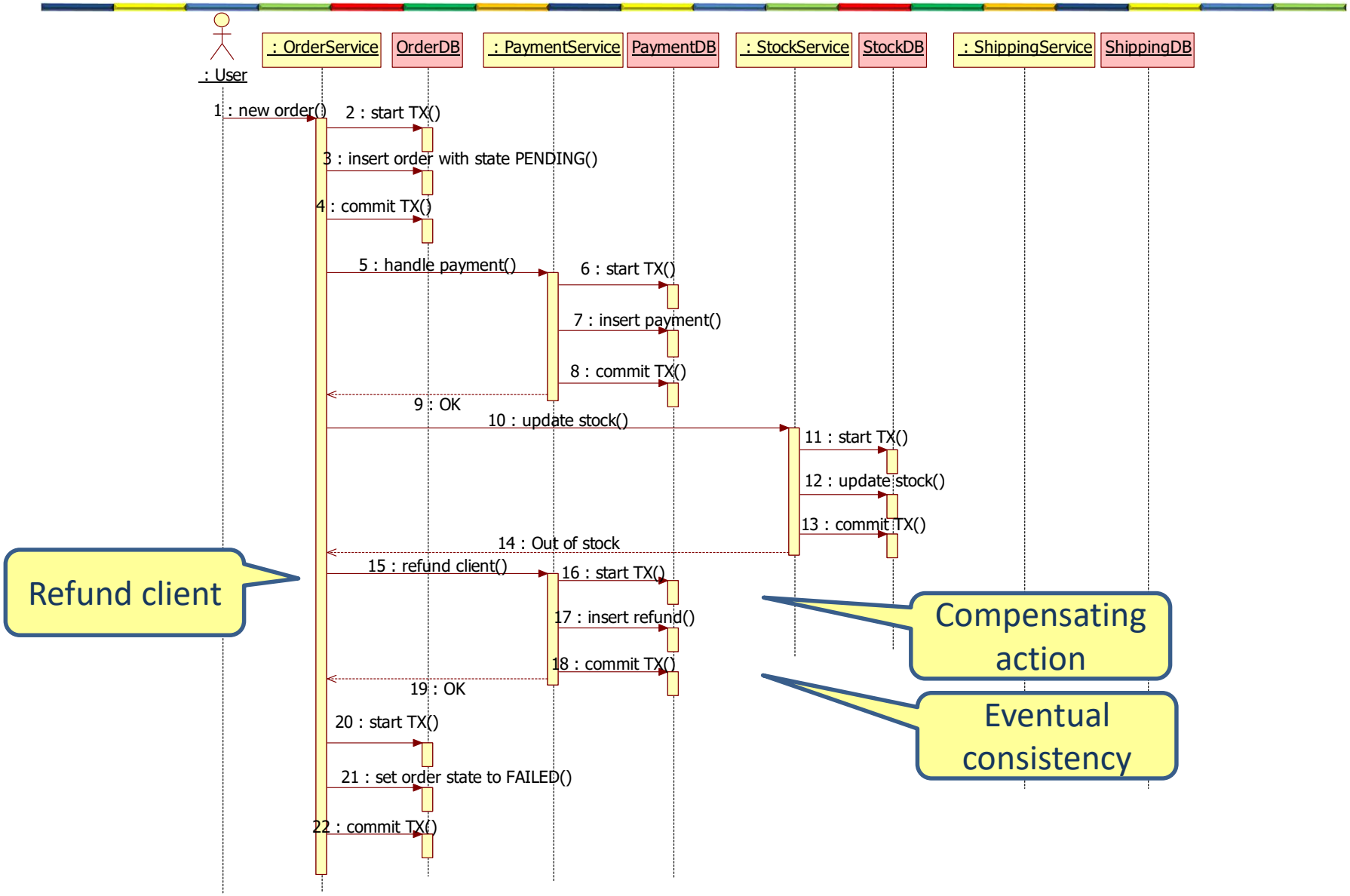
- A distributed system can support only two of the following characteristics



Distributed transaction (SAGA)



Distributed transaction (SAGA)



Challenges of a microservice architecture

Challenge	Solution
Complex communication	Feign Registry API gateway
Performance	
Resilience	Registry replicas Load balancing between multiple service instances Circuit breaker
Security	
Transactions	Compensating transactions Eventual consistency
Keep data in sync	Publish-subscribe data change event
Keep interfaces in sync	Spring cloud contract
Keep configuration in sync	Config server
Monitor health of microservices	ELK + beats
Follow/monitor business processes	Zipkin ELK

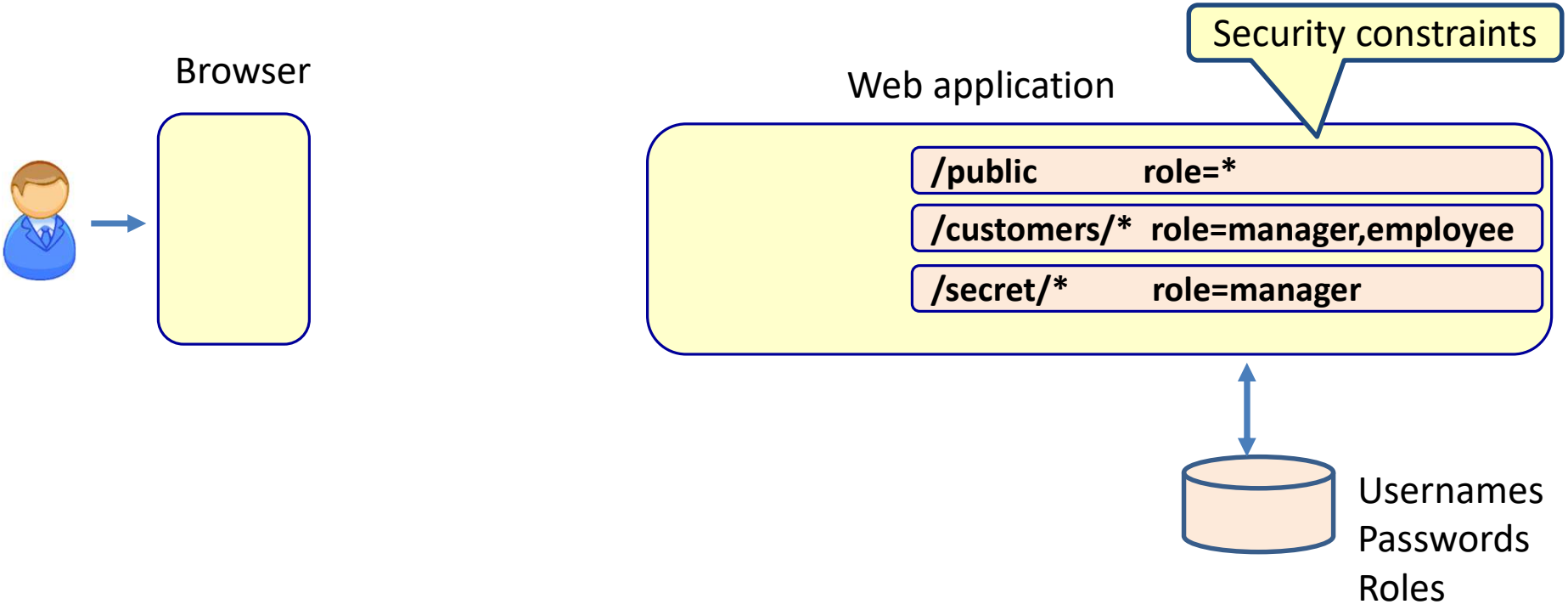
SECURITY

Aspects of security

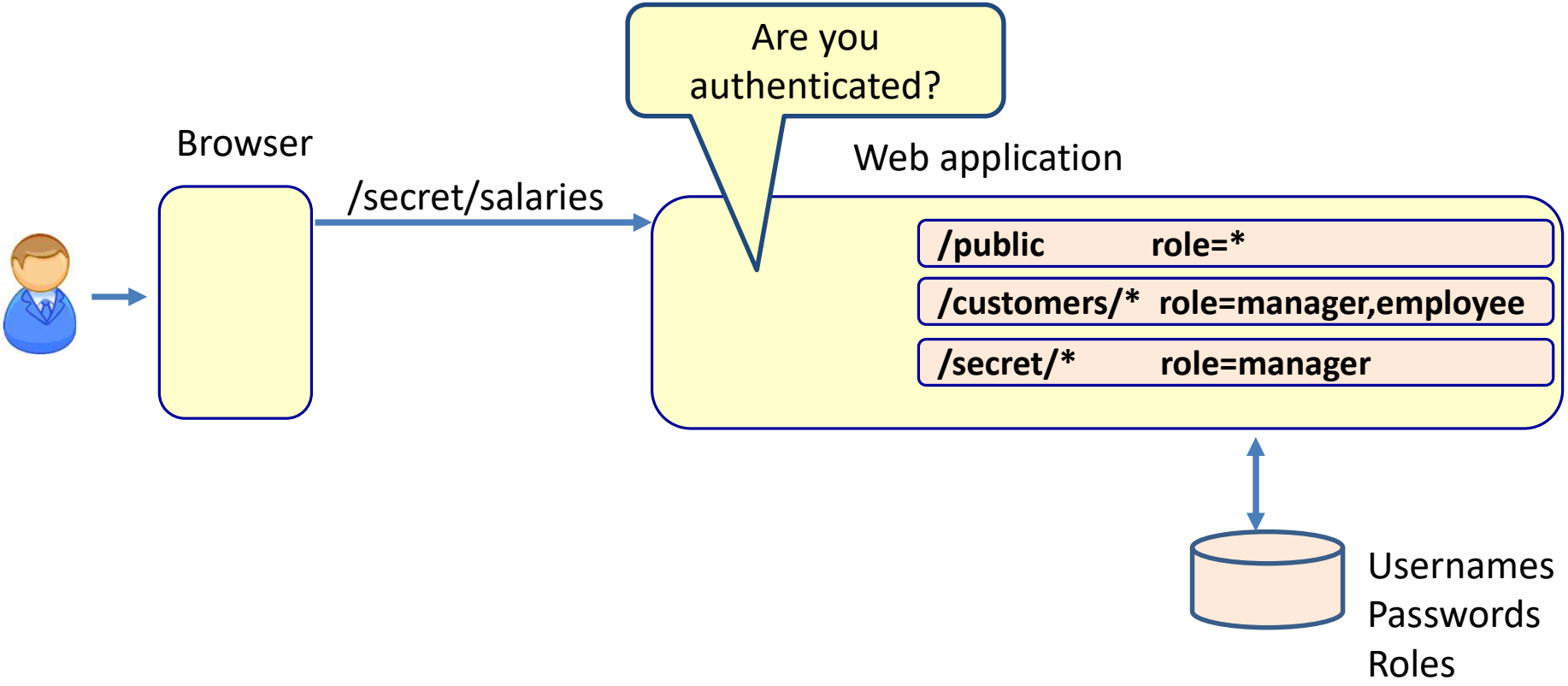
- Authentication: are you who you say you are?
  - Login with username/password
- Authorization: what are you allowed to do?
  - Make url's and/or methods secure
- Confidentiality: No one may look into this request/response
  - Encryption
- Data integrity: No one may change this request/response
  -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

SECURING A WEB APPLICATION

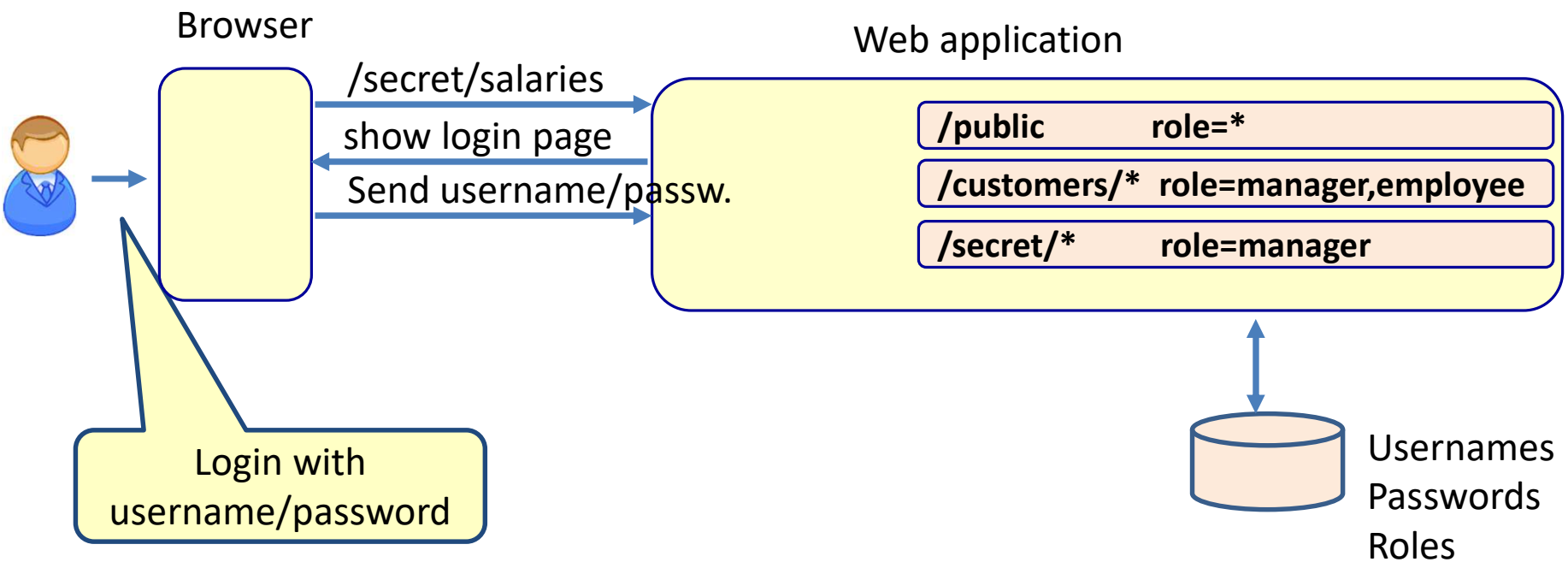
Web security



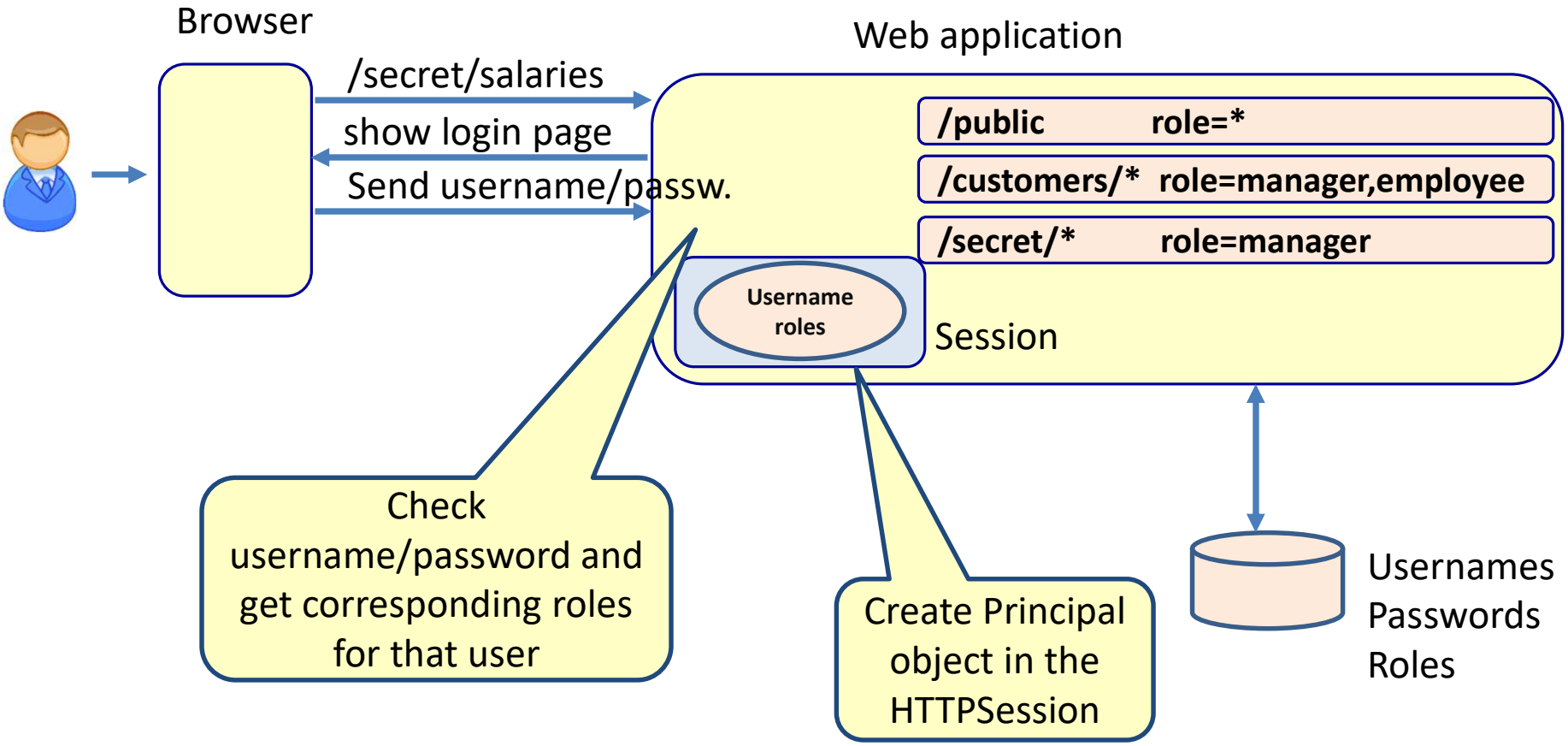
Web security



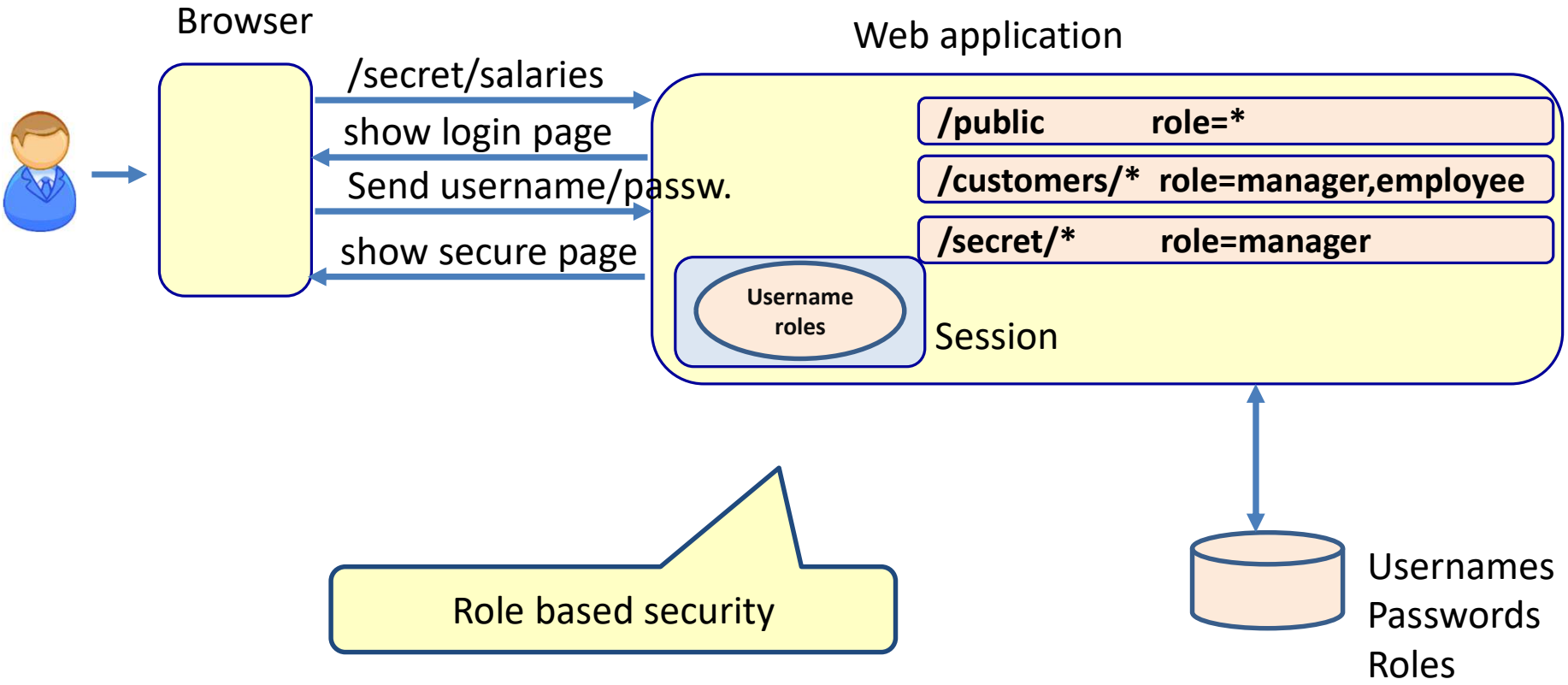
Web security



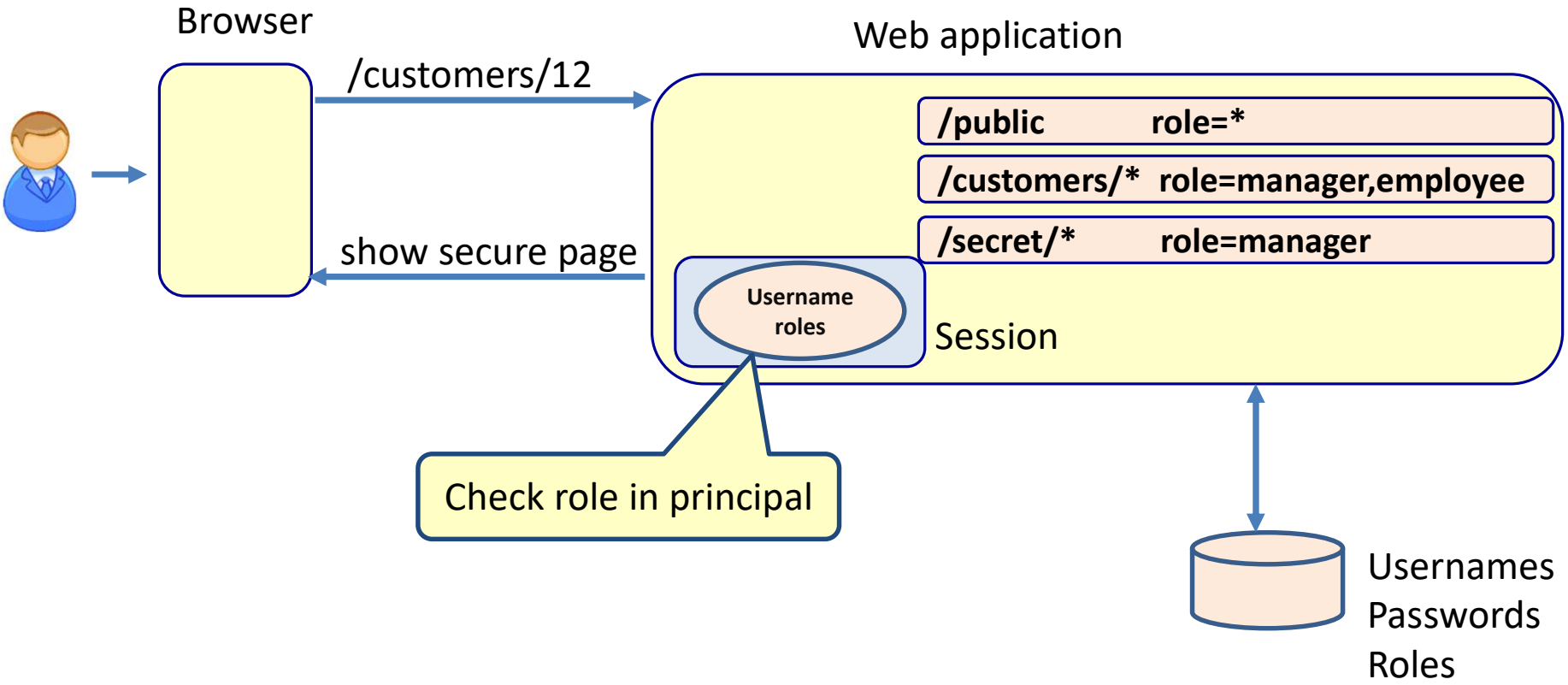
Web security



Web security



Web security



Spring boot security

```
@Configuration
@EnableWebSecurity
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {
    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http
            .authorizeRequests()
                .antMatchers("/**").hasAnyRole("USER")
                .and()
            .formLogin();
    }

    // create users
    @Autowired
    public void configureGlobal(AuthenticationManagerBuilder auth) throws Exception {
        auth.inMemoryAuthentication()
                .withUser("user").password("{noop}pass").roles("USER");
    }
}
```

Give the role USER access to all pages

Use the default login form

Add an in-memory user

{noop} means do not use a password encoder

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 security: Server

```
@RestController
public class MyController {

    @GetMapping("/productinfo")
    public String getProductInfo() {
        return "productinfo";
    }

    @GetMapping("/salaryinfo")
    public String getSalaryInfo() {
        return "salaryinfo";
    }
}
```

Rest security: 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");
    }
}
```

Basic authentication

Rest security: Client

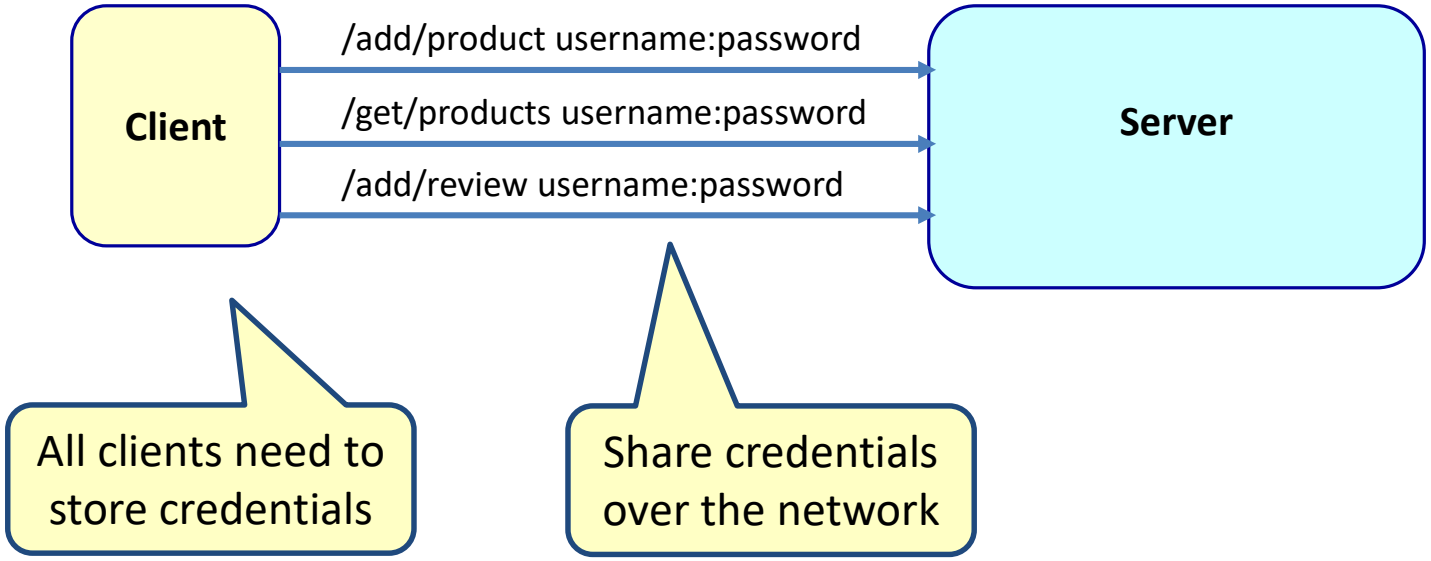
```
@Component
public class SecureRestClient {
  @Autowired
  private RestOperations restTemplate;
  private String serverUrl = "http://localhost:8080/";

  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 to the header

SECURE THE MICROSERVICE ARCHITECTURE

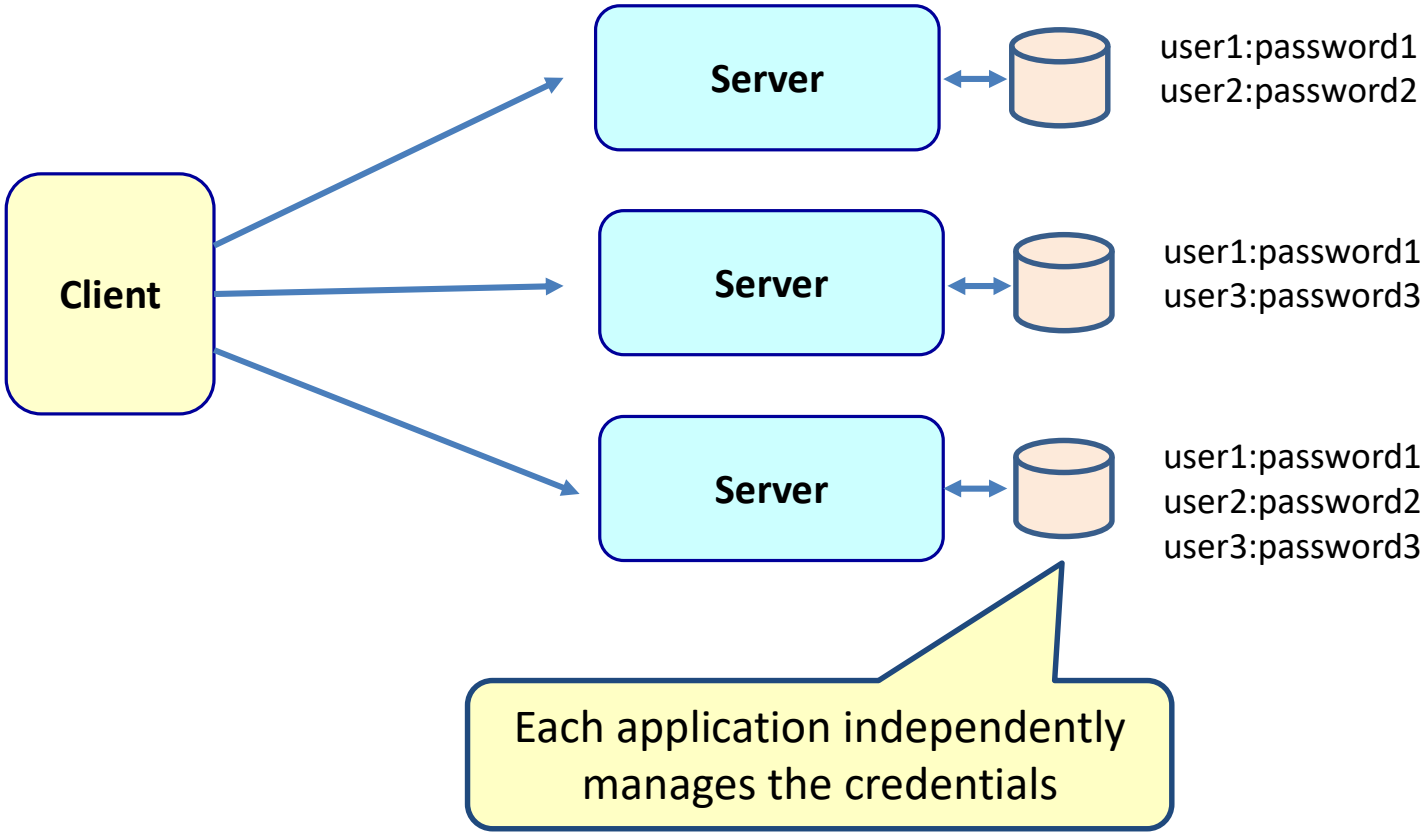
Problems with HTTP basic authentication

- We have to send credentials for every request



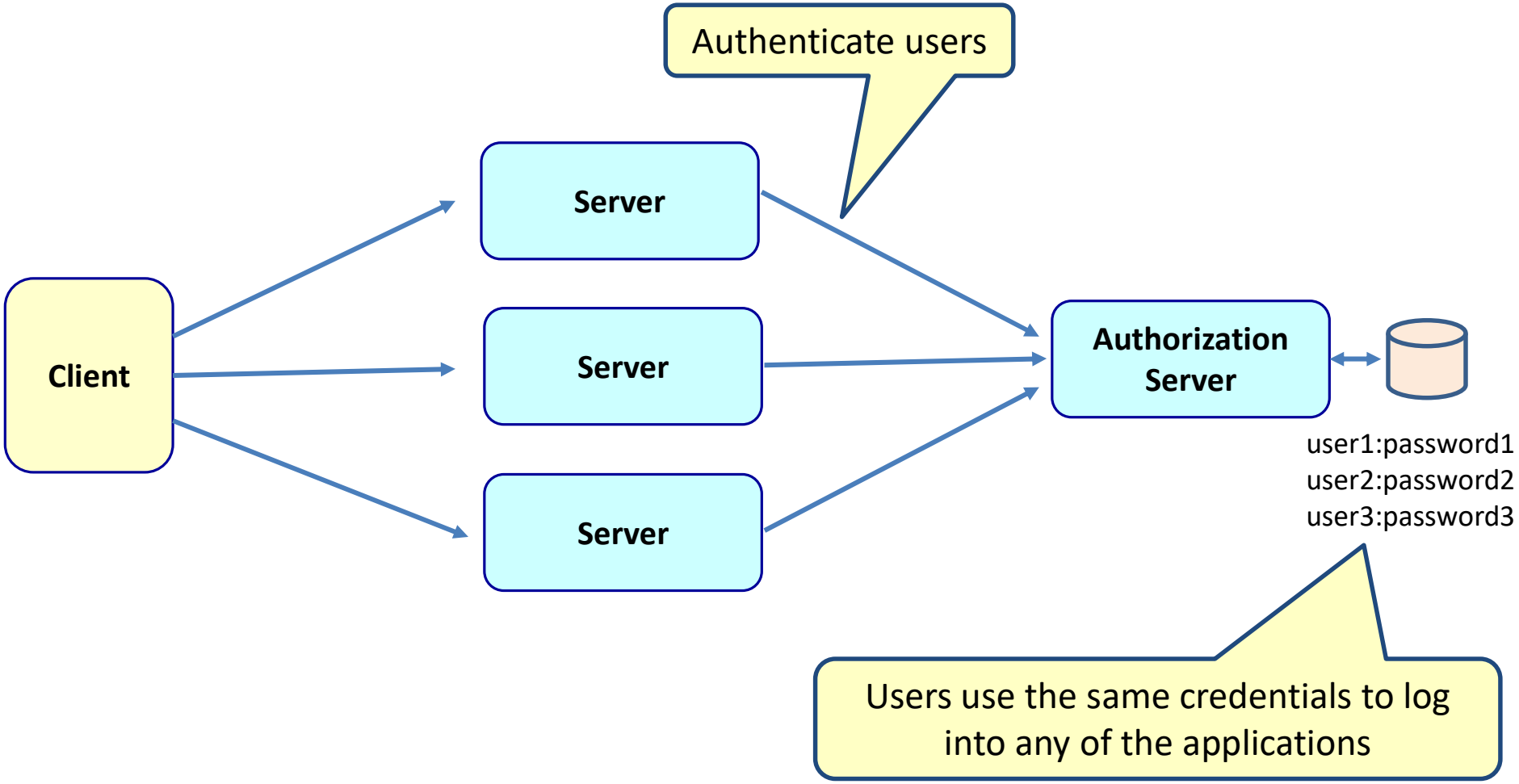
Problems with HTTP basic authentication

- Every system needs to manage these credentials

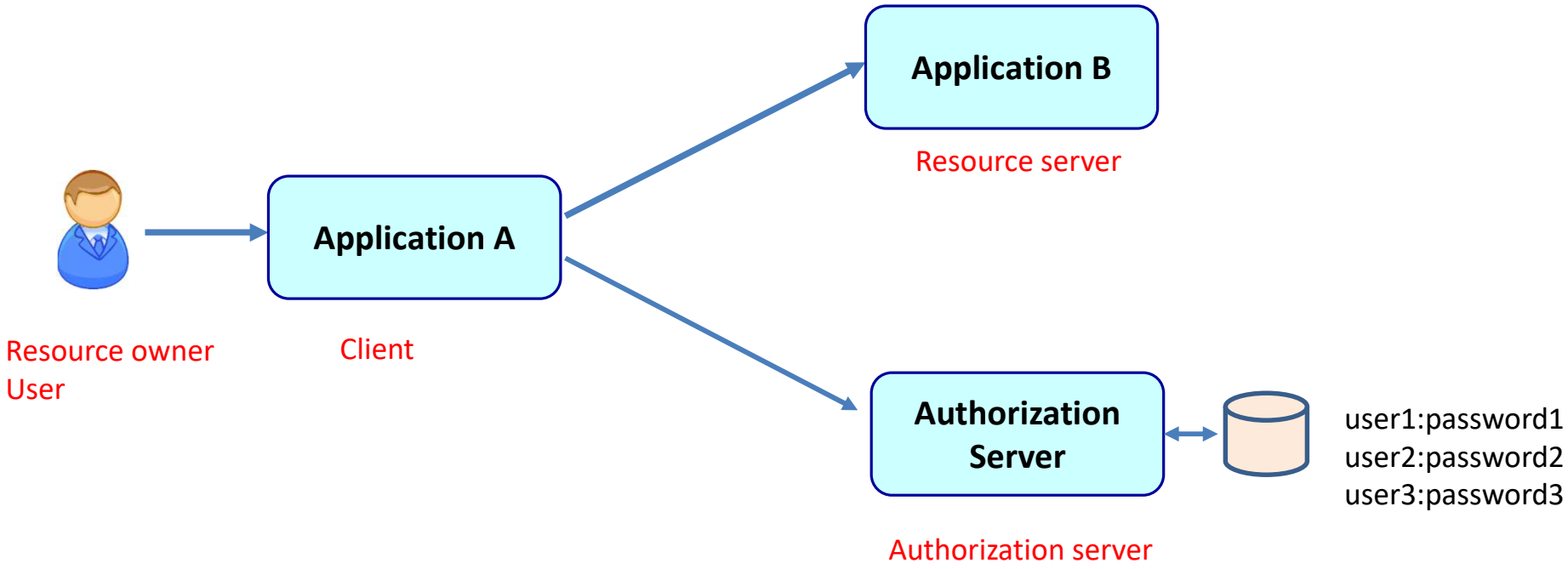


Better solution

- Authorization server



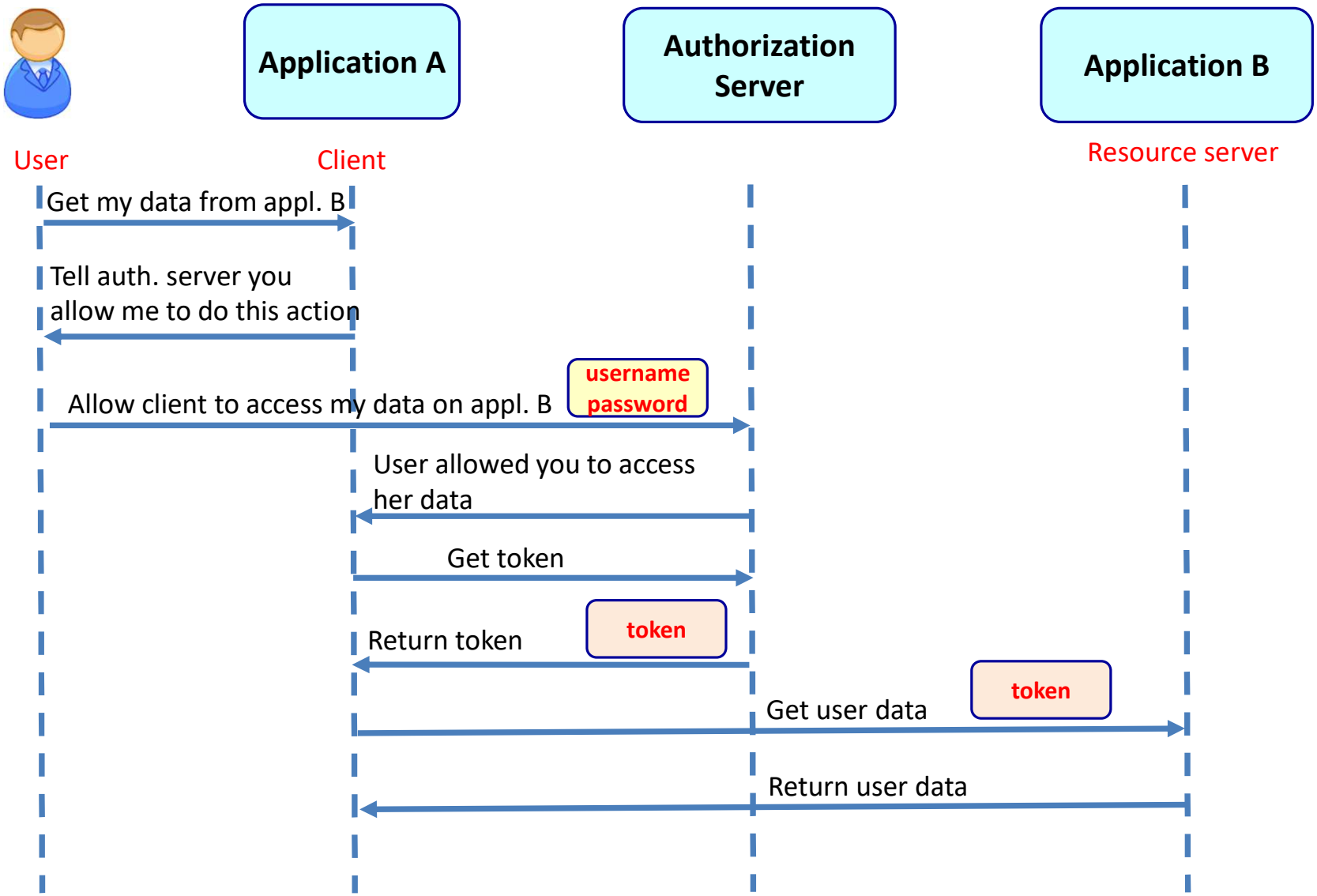
Parts of OAuth 2



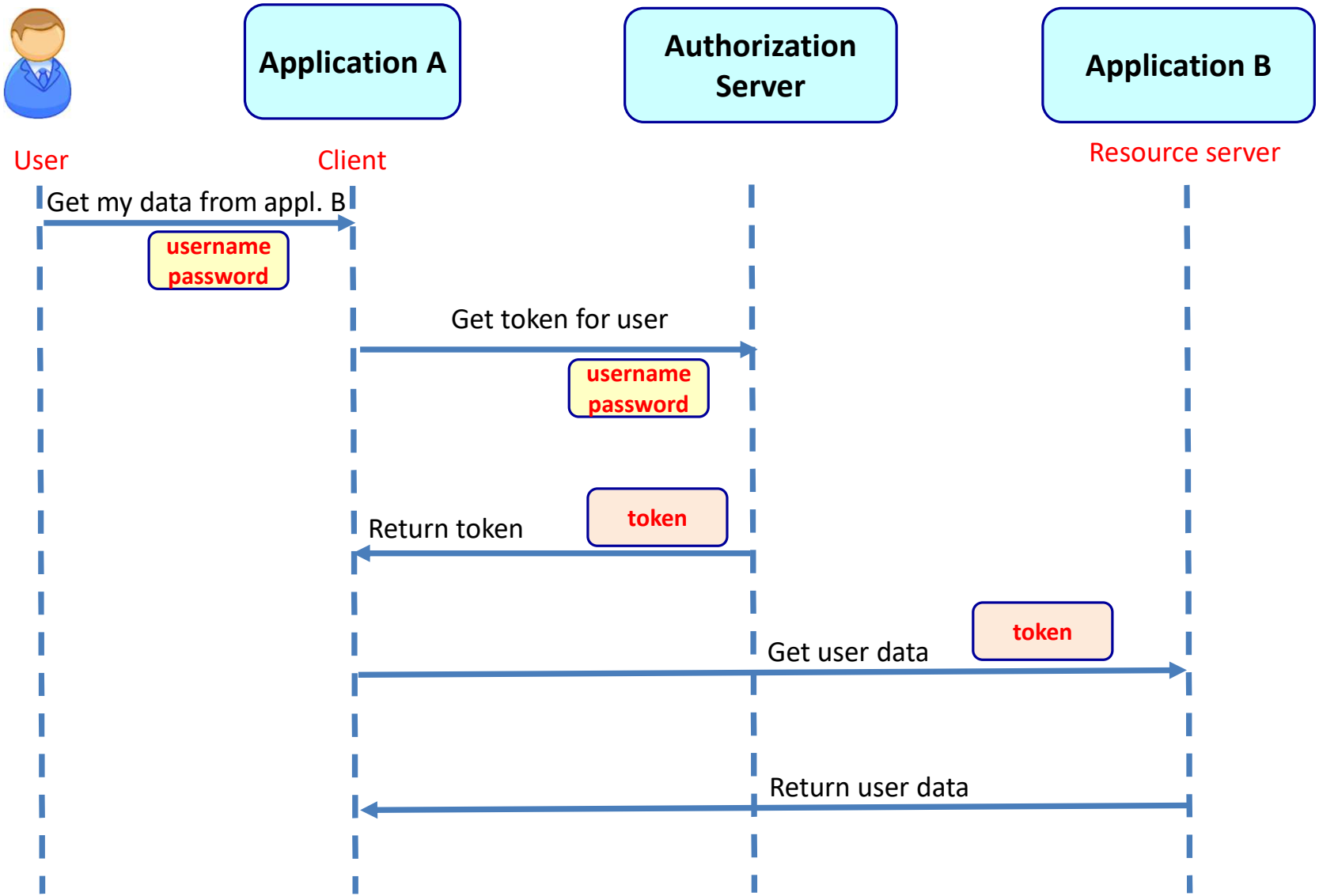
OAuth 2 grants

- Authorization code
- Password
- Client credentials
- Refresh token

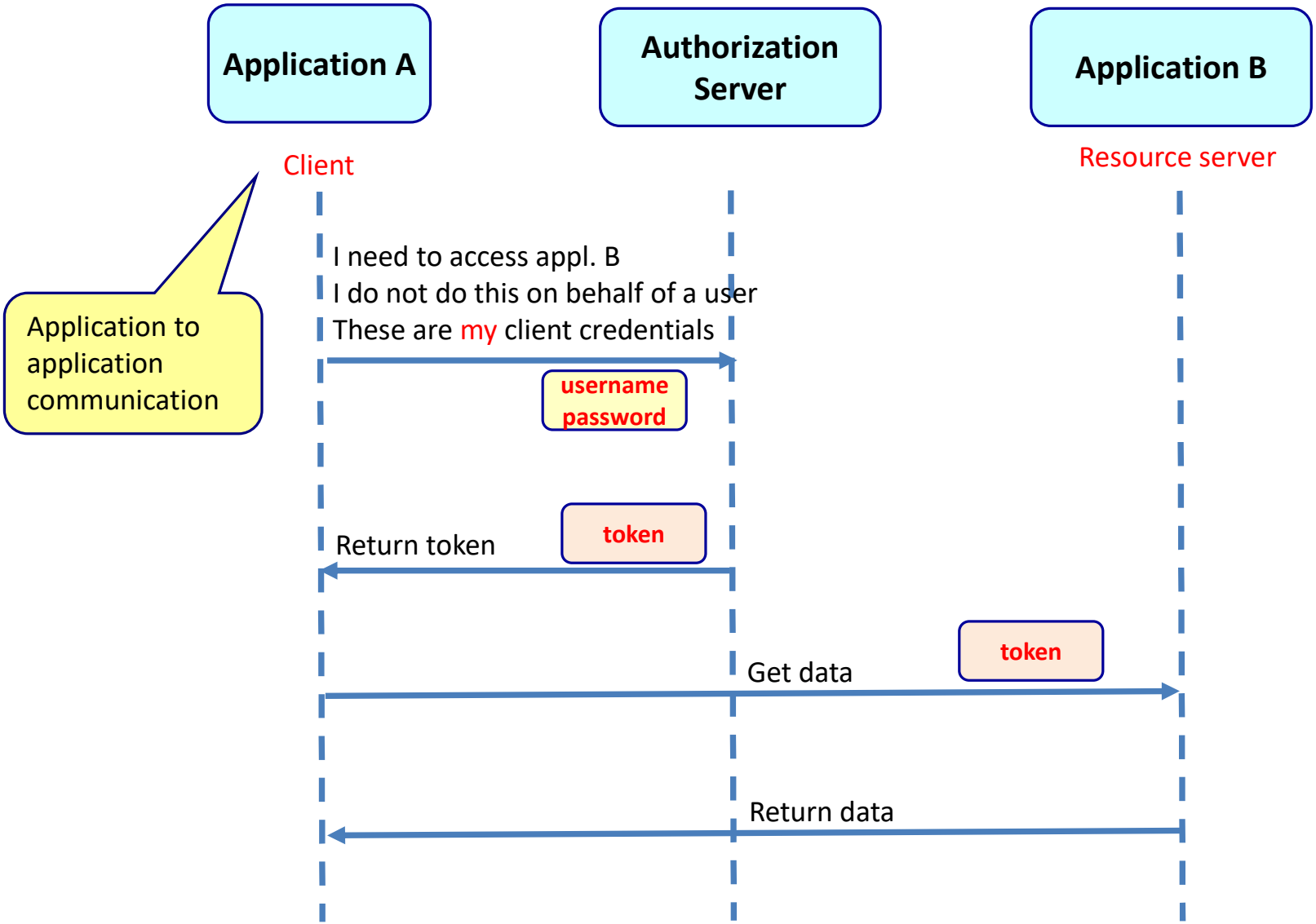
Authorization code grant



Password grant



Client credentials grant



Token

- Tokens can expire
  - Avoid tokens that don't expire
- Why refresh token?
  - User does not have to login again after token is expired
  - Client does not need to store user credentials

Body Cookies Headers (8) Test Results

Pretty Raw Preview JSON

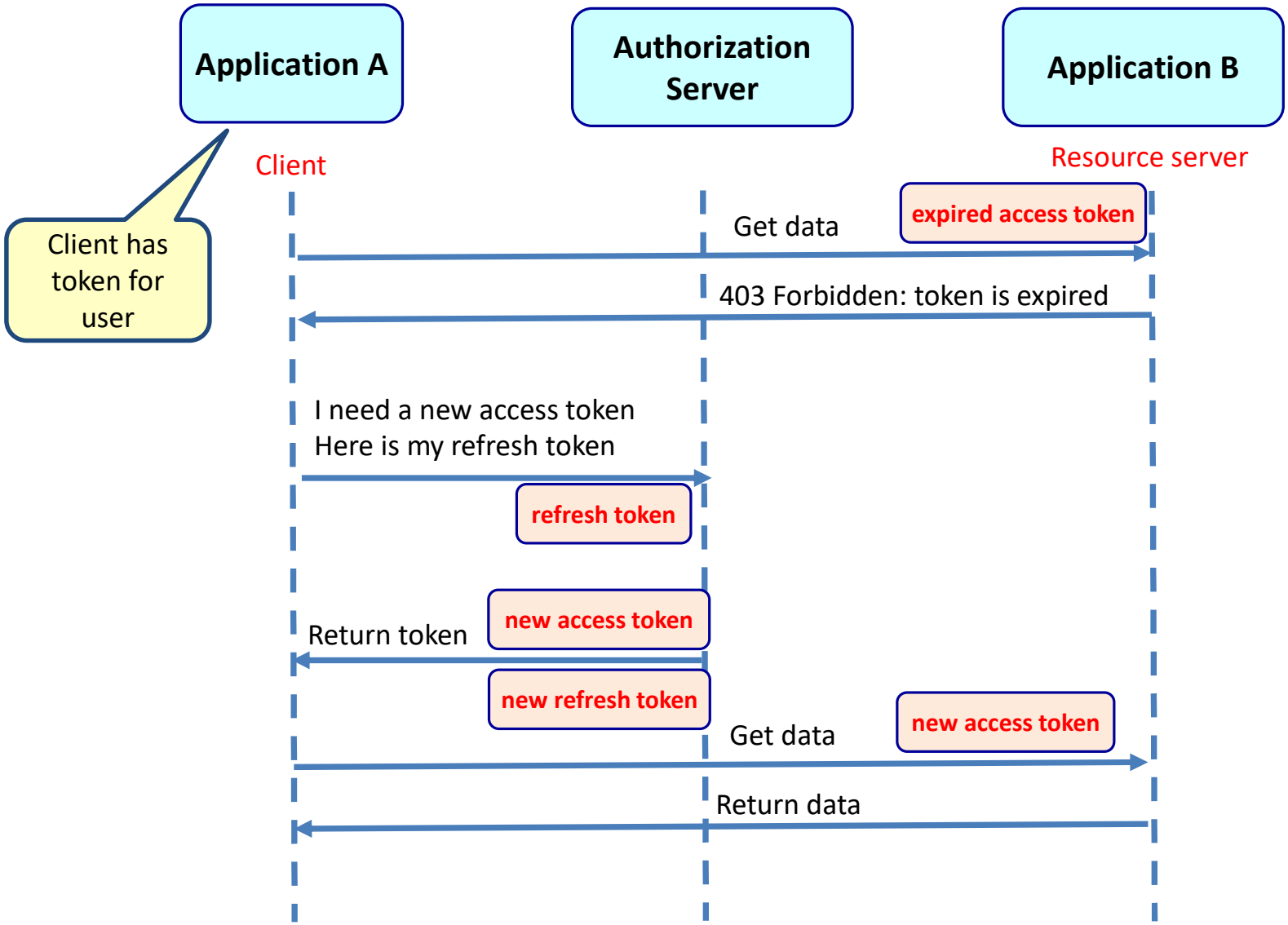

```
1 {
2     "access_token": "b56f8dc4-4a19-411c-86f9-96c1c502f9a7",
3     "token_type": "bearer",
4     "refresh_token": "9618262c-0a5a-458d-afce-c45a544b5aad",
5     "expires_in": 43199,
6     "scope": "webclient"
7 }
```


Access token is presented with each call

Refresh token

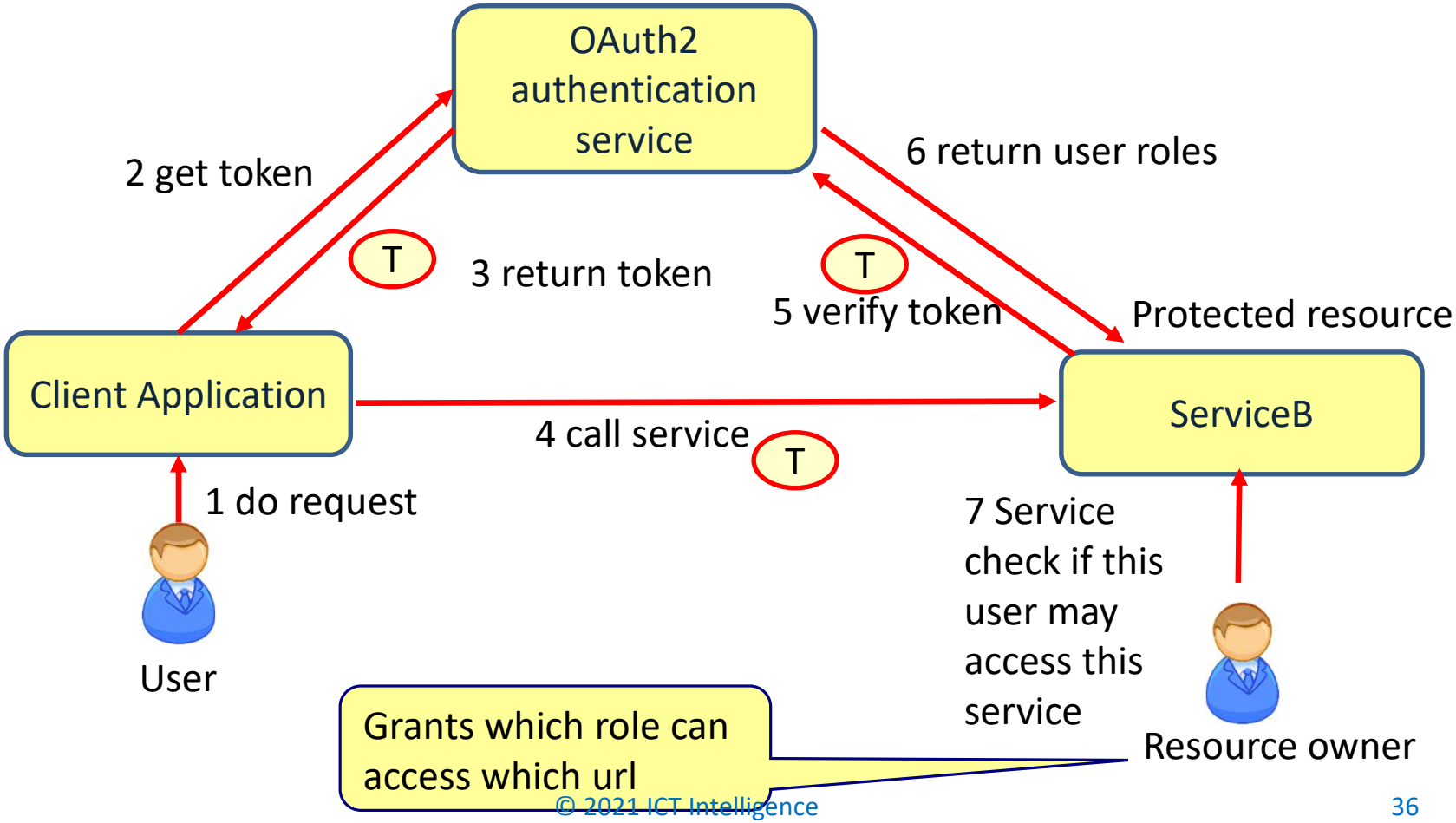
Number of seconds before the token expires in seconds
Spring default value is 12 hours

Refresh token grant



How does OAuth2 work

- Token based authentication and authorization framework



OAUTH2 AUTHORIZATION SERVICE

OAuth2 authorization server

Authorization server

username	password
theClient	thisissecret

user	password	roles
john	password1	USER
frank	password2	USER,MANAGER

Username-password to retrieve a token

Users of your microservices

The authorization service

```
@SpringBootApplication
@RestController
@EnableResourceServer
@EnableAuthorizationServer
public class AuthorizationServiceApplication {

  public static void main(String[] args) {
    SpringApplication.run(AuthorizationServiceApplication.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

```yaml
server:
  port: 8080
```

Retrieve a token

Authentication server

username	password
theClient	thisissecret

user	password	roles
john	password1	USER
frank	password2	USER,MANAGER

POST localhost:8080/oauth/token

Basic authentication

username	theClient
password	thisissecret

grant_type	password
scope	webclient
username	john
password	password1

Token

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
☑	grant_type	password		
☑	scope	webclient		
☑	username	john		
☑	password	password1		
	New key	Value	Description	

Returned payload

Body Cookies Headers (8) Test Results

Pretty Raw Preview JSON


```
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 }
```


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

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

Authorization Headers (2) Body Pre-request Script Tests

form-data x-www-form-urlencoded raw binary

Key	Value
grant_type	password
scope	webclient
username	john
password	password1
New key	Value

Body Cookies Headers (8) Test Results

Pretty Raw Preview JSON


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


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

Authorization Headers (2) Body Pre-request Script Tests

form-data x-www-form-urlencoded raw binary

Key	Value
grant_type	password
scope	webclient
username	frank
password	password2
New key	Value

Body Cookies Headers (8) Test Results

Pretty Raw Preview JSON


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

Retrieve user information

Authentication server

username	password
theClient	thisissecret

user	password	roles
john	password1	USER
frank	password2	USER,MANAGER

GET localhost:8080/user

Authorization	Bearer <Token>

User name
User roles

Get user information

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

Authorization Headers (1) Body Pre-request Script Tests

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

Token for John

Body Cookies Headers (9) Test Results

Pretty Raw Preview JSON


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


User information based on OAuth token

Get user information

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

Authorization Headers (1) Body Pre-request Script Tests

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

Token for Frank

Body Cookies Headers (9) Test Results

Pretty Raw Preview JSON


```
{
    "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"
    ]
}
```

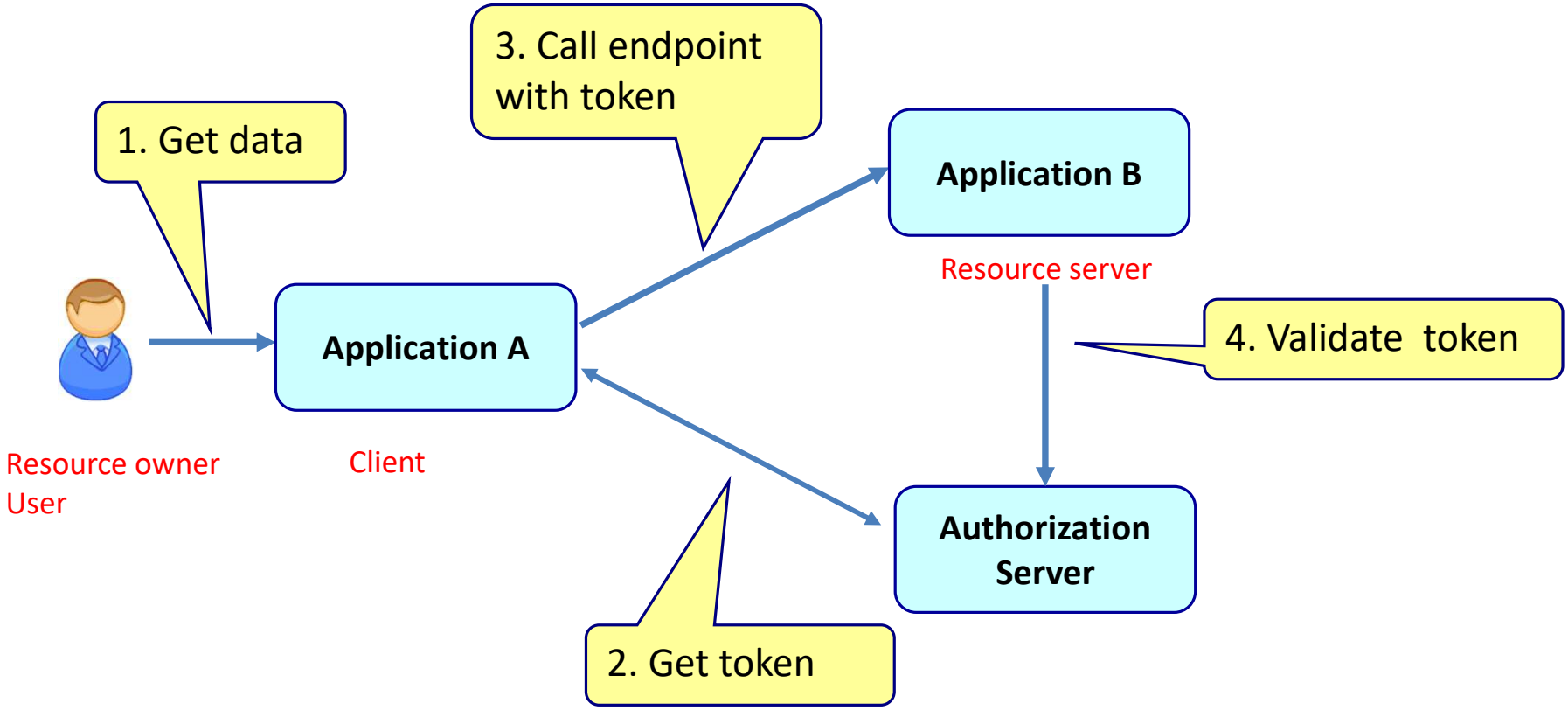

User information based on OAuth token

A SECURE APPLICATION

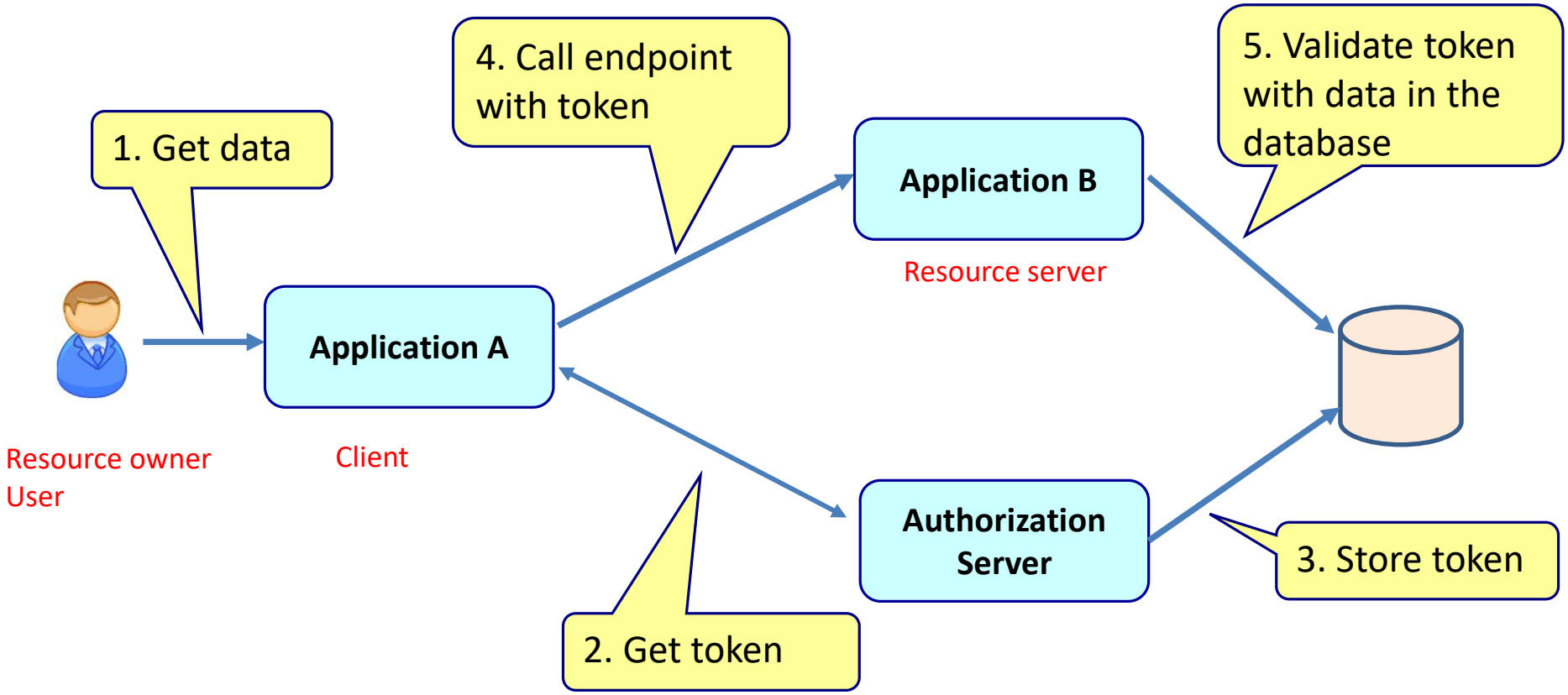
How does the resource server validate a token?

1. Resource server directly calls the authorization server
2. Use a common database (blackboarding)
3. Use JSON Web Tokens (JWTs)

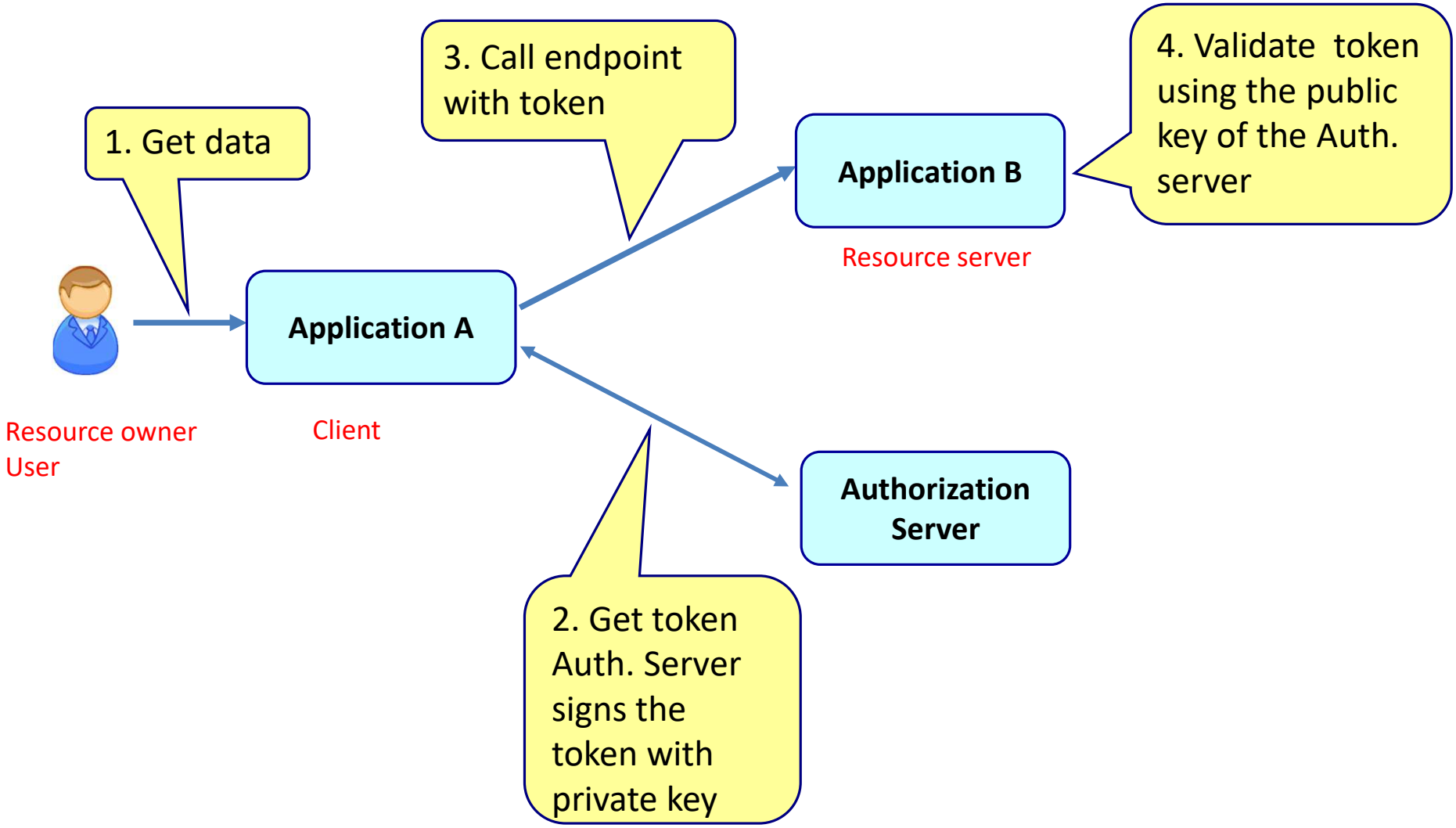
1. Resource server directly calls the authorization server



2. Use a common database



3. Use JSON Web Tokens (JWTs)



A secure application

Authentication server

username	password
theClient	thisissecret

user	password	roles
john	password1	USER
frank	password2	USER,MANAGER

GET localhost:8080/user

Authorization	Bearer <Token>

User name
User roles

Secure application

URL	Roles
/phone	USER, MANAGER
/Salary	MANAGER

A secure application

```java
@SpringBootApplication
public class SecureServiceAApplication {

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

```java
@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

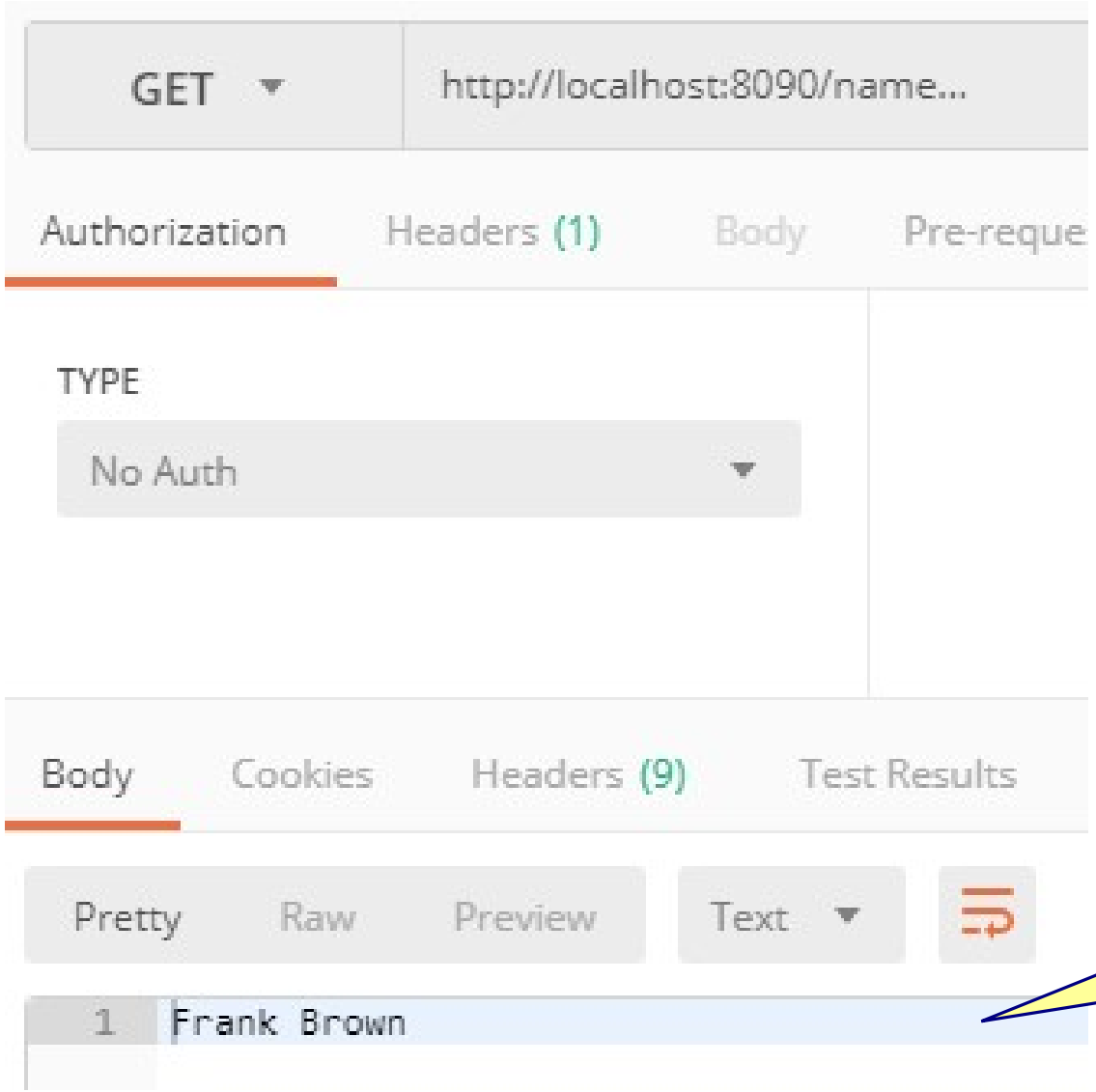
```
@RestController
public class Controller {

  @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

```yaml
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



Everyone can get the user name without authorization

Get the phone number

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

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

TYPE

No Auth

This request does not use any authorizat

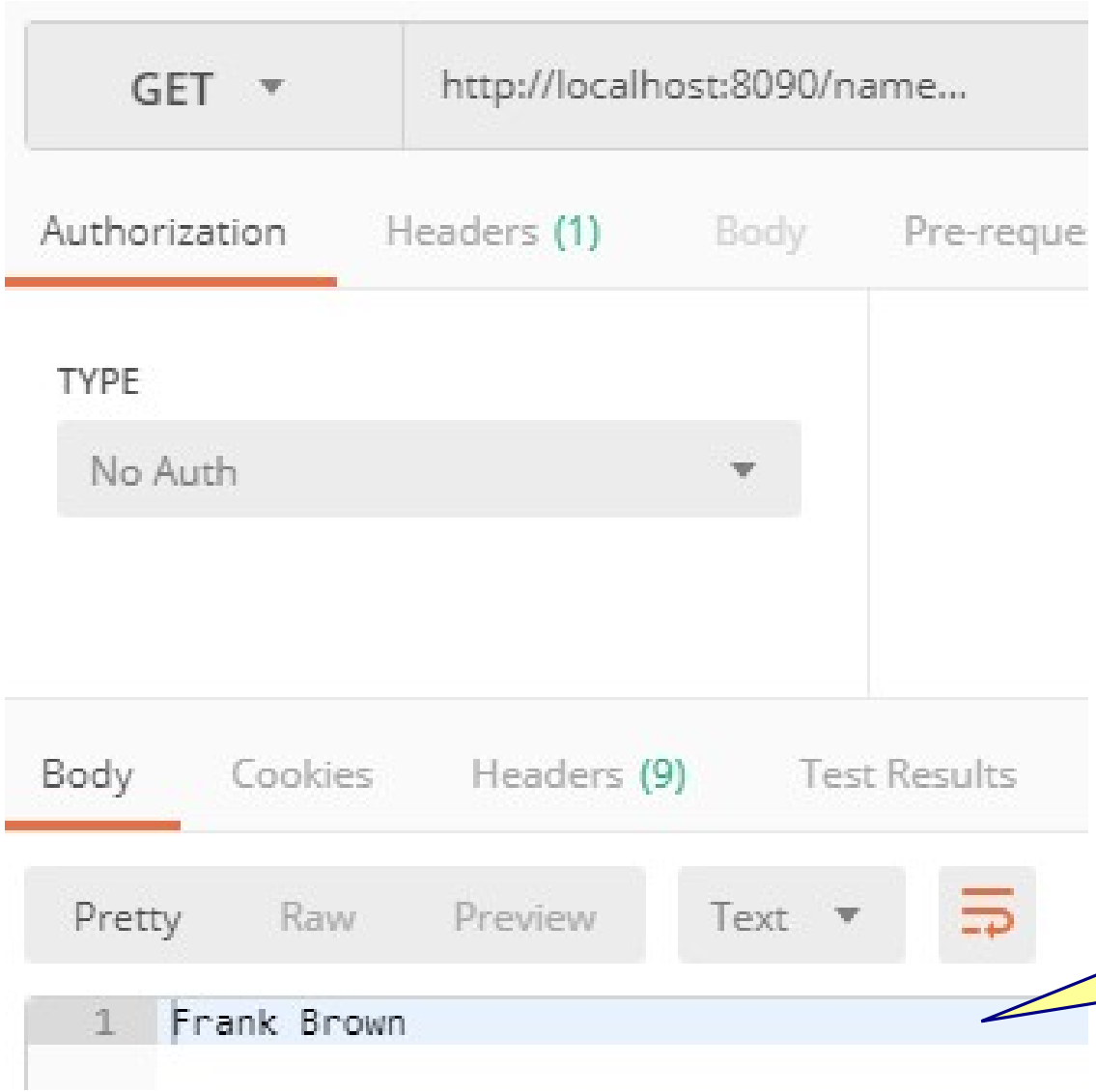
Body | Cookies | Headers (9) | Test Results

Pretty | Raw | Preview | JSON

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

You cannot get the phone number without access token

Get the user name



Everyone can get the phone number without authorization

Get the phone number

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

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

TYPE

No Auth

This request does not use any authorizat

Body | Cookies | Headers (9) | Test Results

Pretty | Raw | Preview | JSON

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

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

```
1 645322899
```

Frank can get the phone number

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

```
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
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
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 }
```

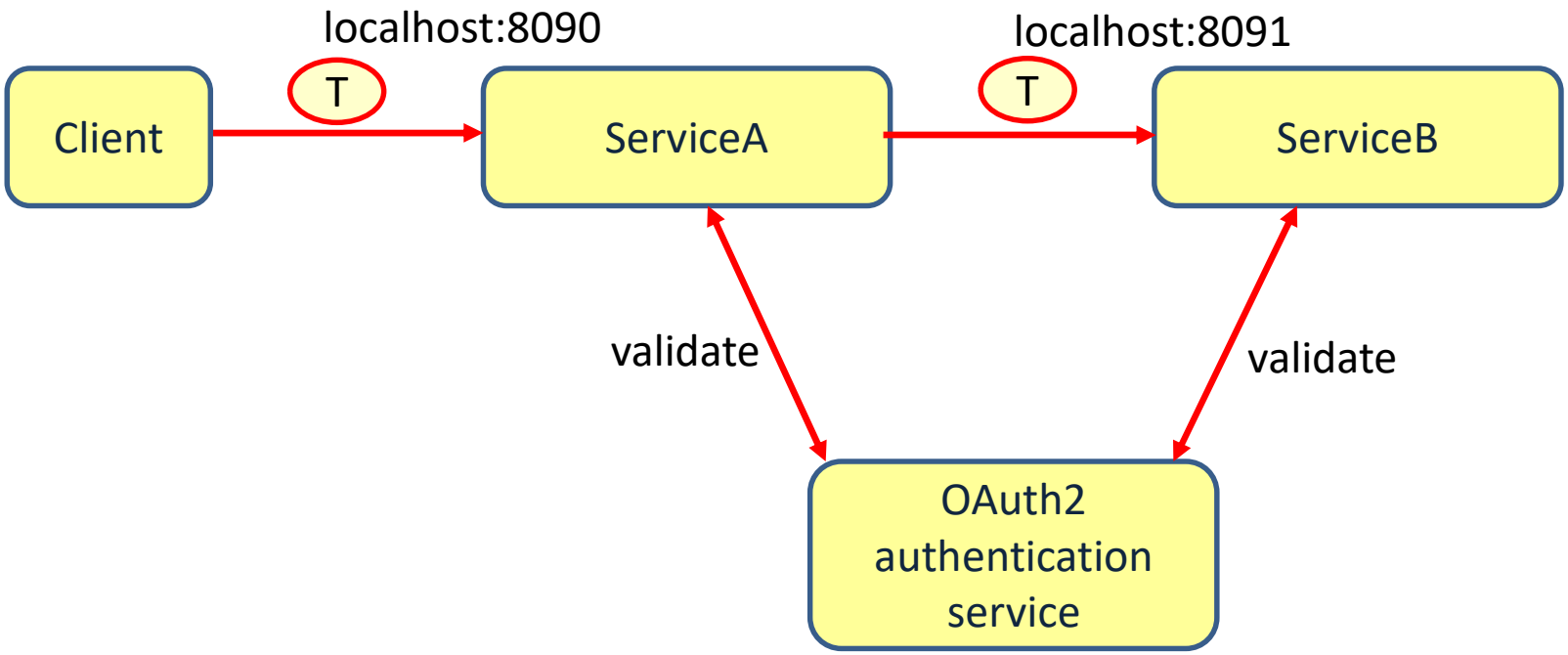
John cannot get the salary (role = USER)

Main point

- To implement security in a microservice architecture we use token based security (OAuth2). *The human nervous system is able to access that most basic field of pure consciousness which is the source of all the laws of Nature.*

PROPAGATING THE TOKEN

Propagate the token



Secure application B

```
@RestController
public class Controller {


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

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

}
```

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);
  }
}
```

OAuth2RestTemplate

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")
                .anyRequest()
                .authenticated();
    }
}
```

The controller2

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

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

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

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

OAuth2RestTemplate

One services calls another service

GET http://localhost:8090/salary Send

Params Auth Headers (10) Body Pre-req. Tests Settings Cookies

Headers 9 hidden

	KEY	VALUE	DESCRI
☑	Authorization	Bearer 4f9e70ec-834d-4068-9cd5-b...	
	Key	Value	Description

Bulk Edit Presets

Body Cookies Headers (8) Test Results 403 Forbidden 102 ms 333 B Save Response

Pretty Raw Preview Visualize JSON

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

John cannot get salary info

One services calls another service

GET http://localhost:8090/salary Send

Params Auth Headers (10) Body Pre-req. Tests Settings Cookies

Headers 9 hidden

	KEY	VALUE	DESCRI	ooo	Bulk Edit	Presets
☑	Authorization	Bearer 37d7c231-9882-4c13-9a4d-6...				
	Key	Value	Description			

Body Cookies Headers (9) Test Results 200 OK 463 ms 303 B Save Response

Pretty Raw Preview Visualize Text

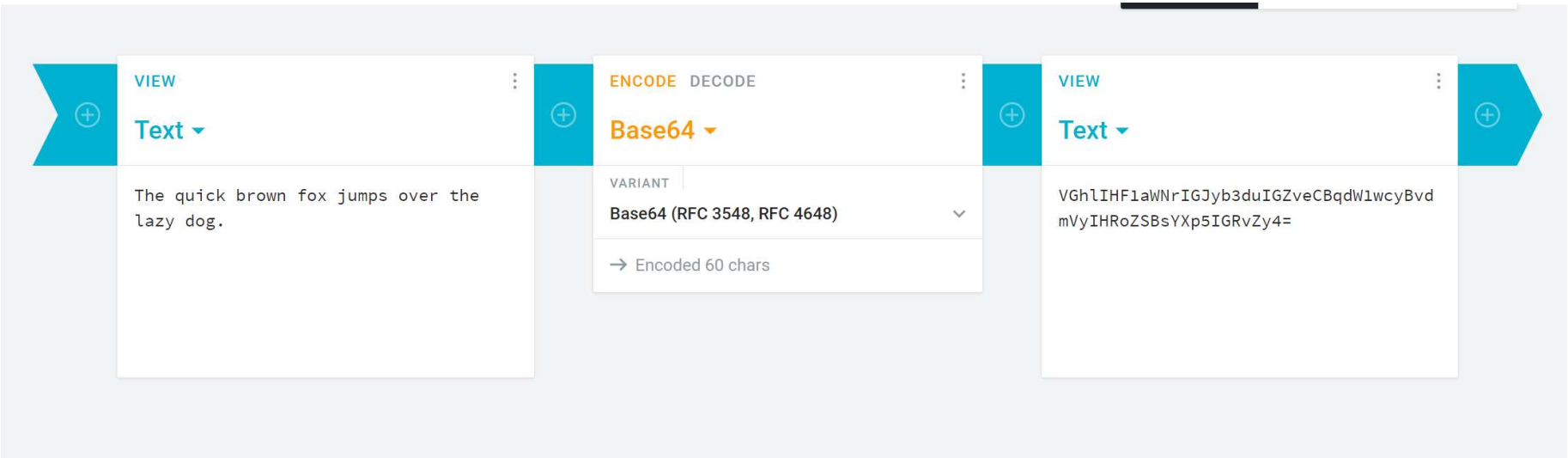
```
1  95.000
```

Frank can get salary info

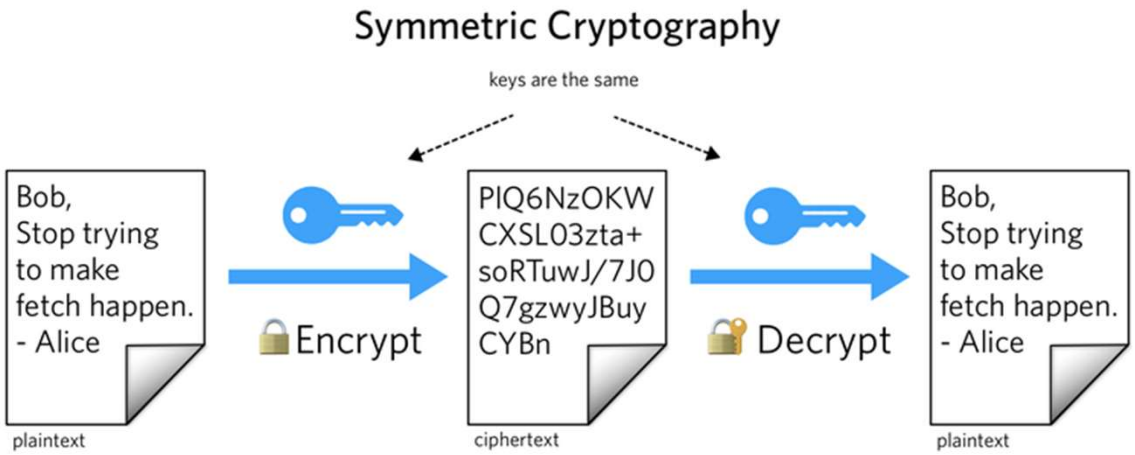
JWT

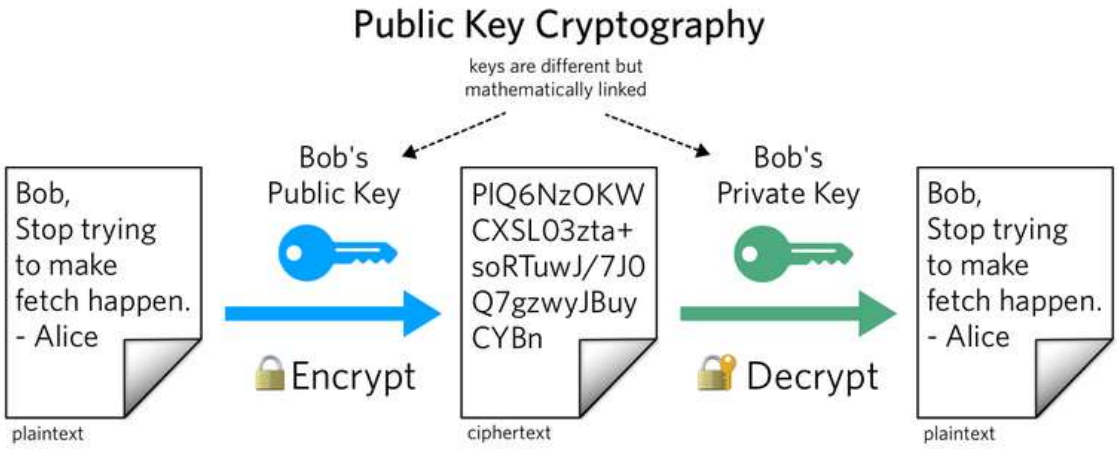
Base64 encoding

- Transform data into transport save format
  - Characters that HTTP understands
- Not encryption (not secure)
- cryptii.com

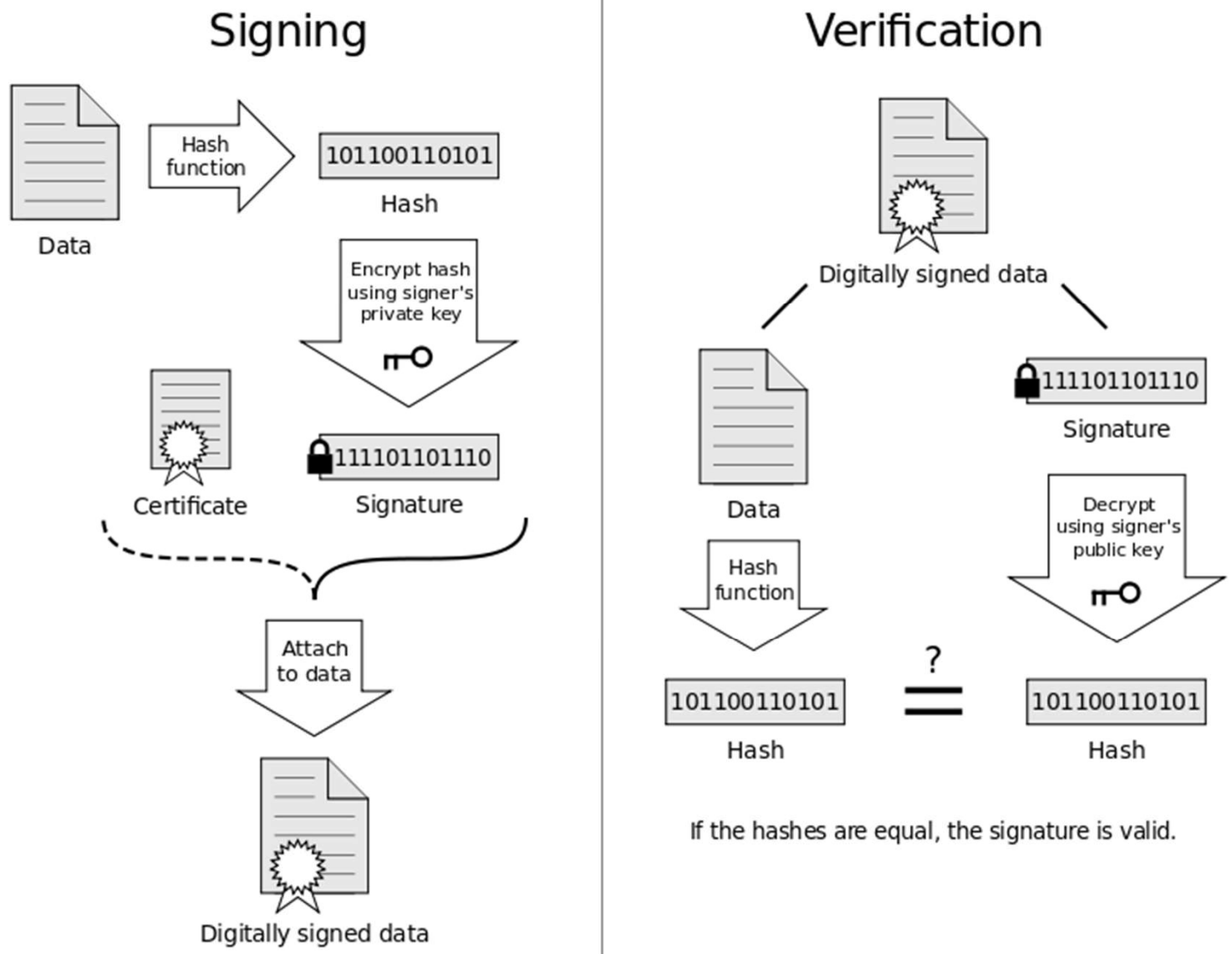


Cryptography





Digital signature



JWT

- JSON Web Token

- HTTPSession: keep user information (username, role) on the server (in the session)
- JWT: keep user information (username, role) on the client (in a token)

jwt.io

Not secure, not encrypted content!

Algorithm HS256

Encoded PASTE A TOKEN HERE


```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.ey
JzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6Ikpva
G4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKx
wRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c
```


JWT token

Decoded EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

Signature algorithm

PAYLOAD: DATA

```
{
  "sub": "1234567890",
  "name": "John Doe",
  "iat": 1516239022
}
```

Subject (often userid)

Issued at

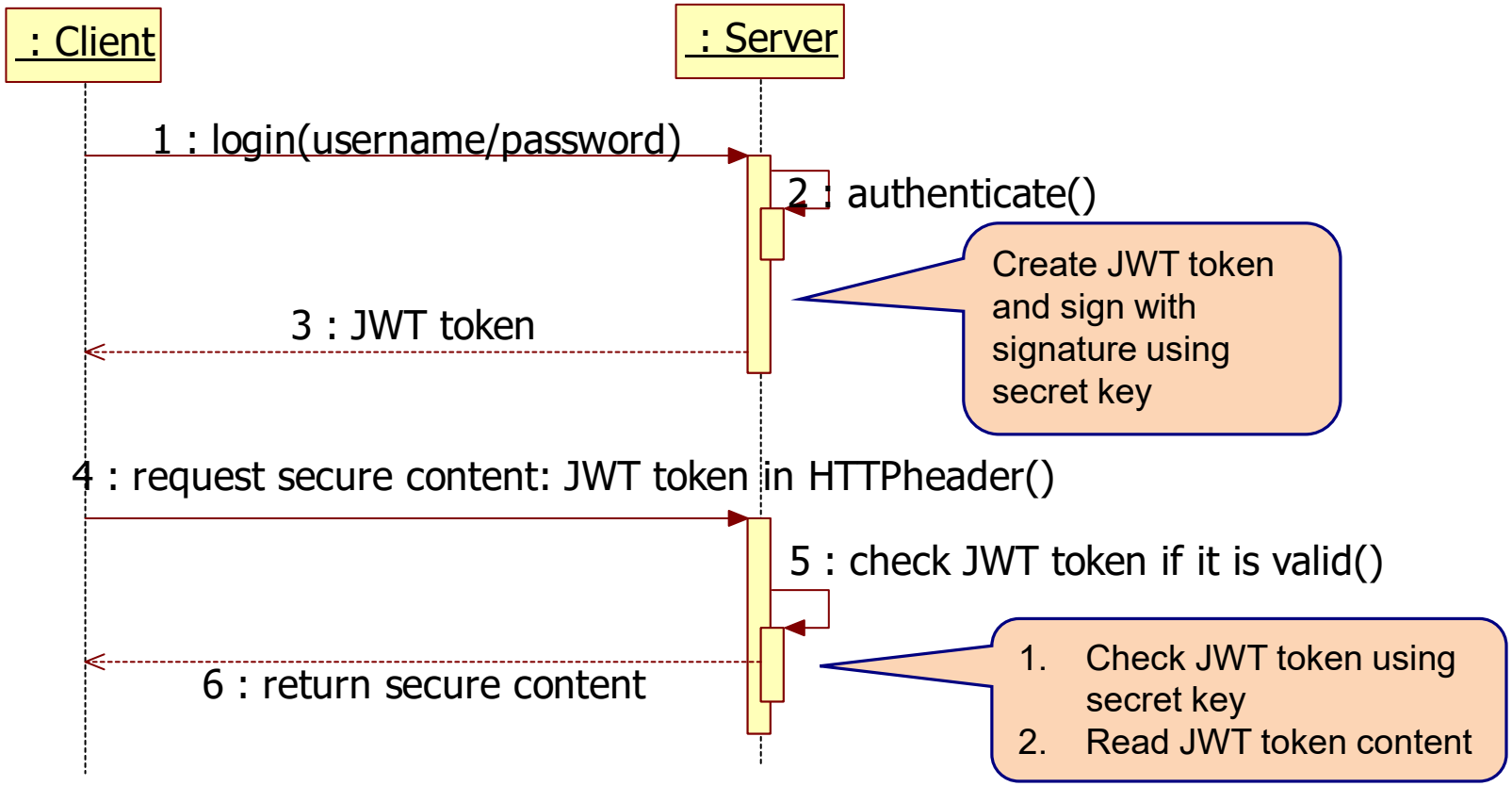
You can add anything you want in the token

VERIFY SIGNATURE

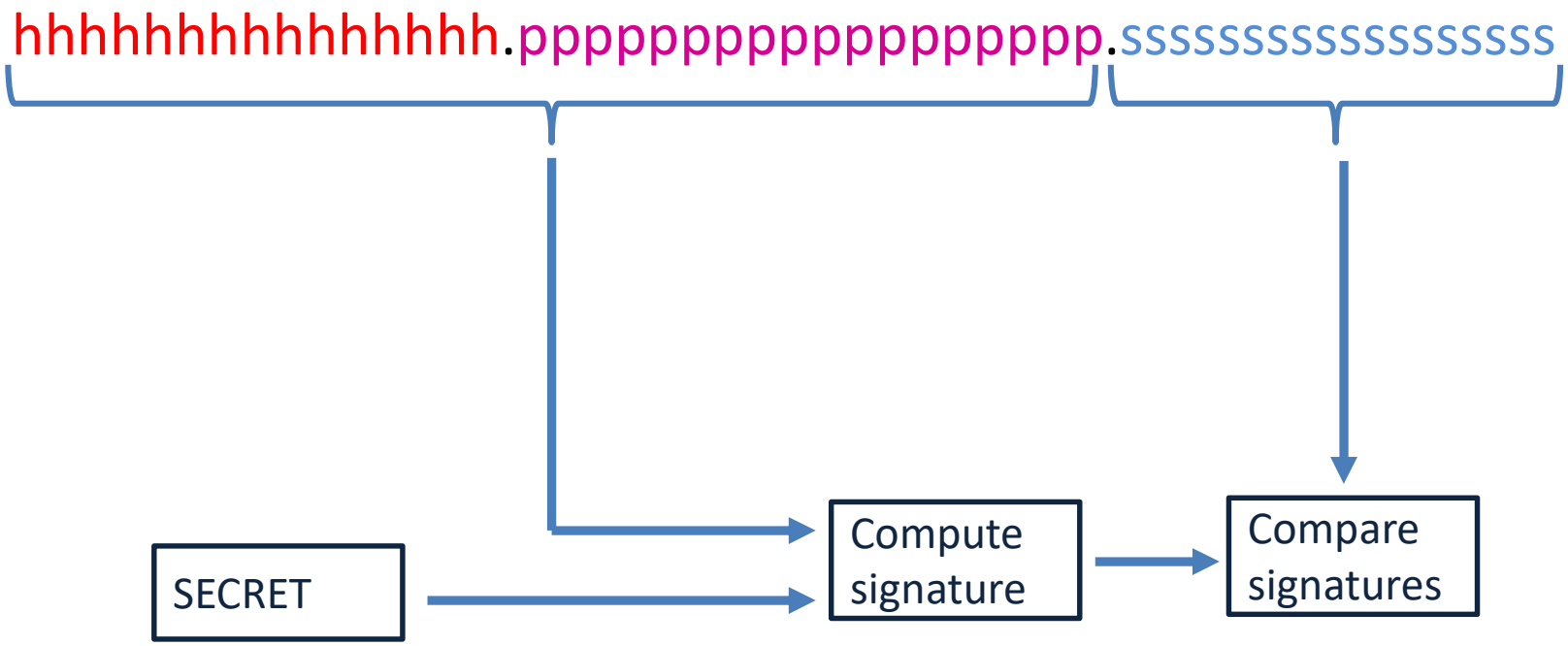
```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  your-256-bit-secret
) ☐ secret base64 encoded
```

Secret key

Using JWT


: Client
: Server
1 : login(username/password)
2 : authenticate()
Create JWT token and sign with signature using secret key
3 : JWT token
4 : request secure content: JWT token in HTTPheader()
5 : check JWT token if it is valid()
1. Check JWT token using secret key
2. Read JWT token content
6 : return secure content

Check JWT signature



Get JWT token content

hhhhhhhhhhhhhhh.pppppppppppppppppppp.sssssssssssssssssss

Base 64 encode

Base 64 encode

```
{
    .....
    ....
}
```

header

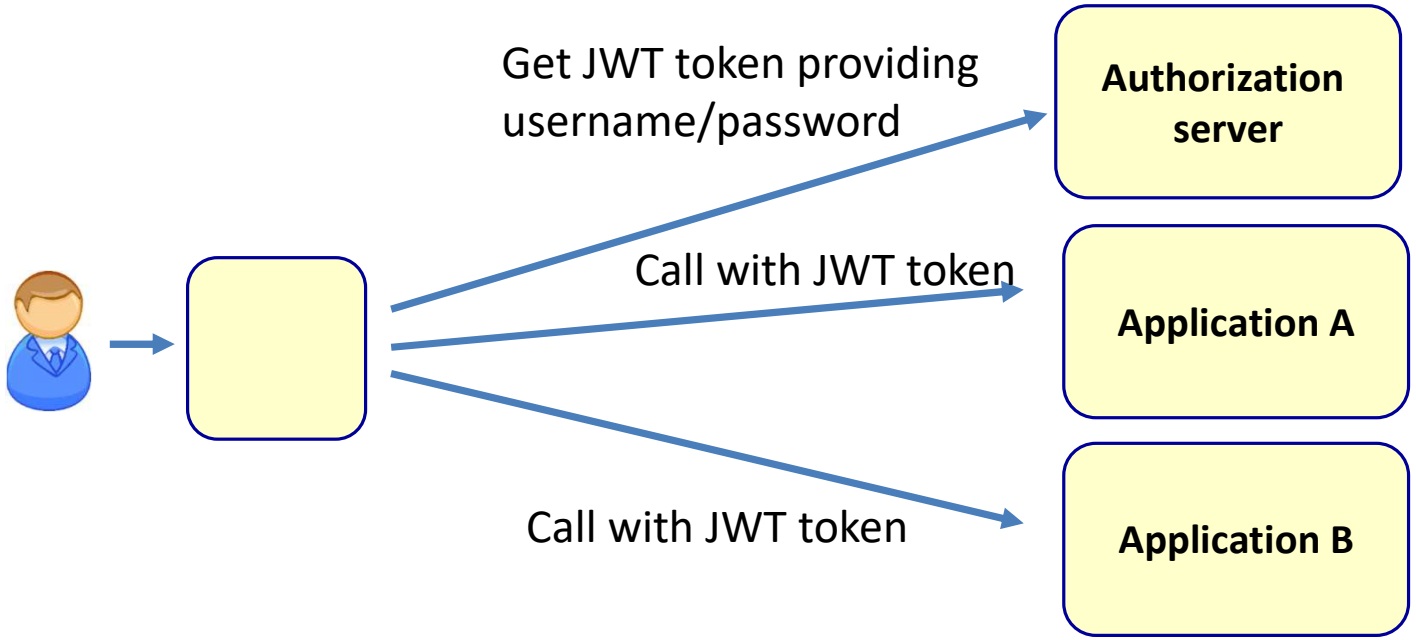
```
{
    .....
    ....
}
```

payload

JWT

- Never place secure content in a JWT token
  - JWT token is only signed, not encrypted
- What if someone steals the JWT token
  - Use HTTPS
  - Use token expiration
  - Server maintains list of blacklisted JWT's

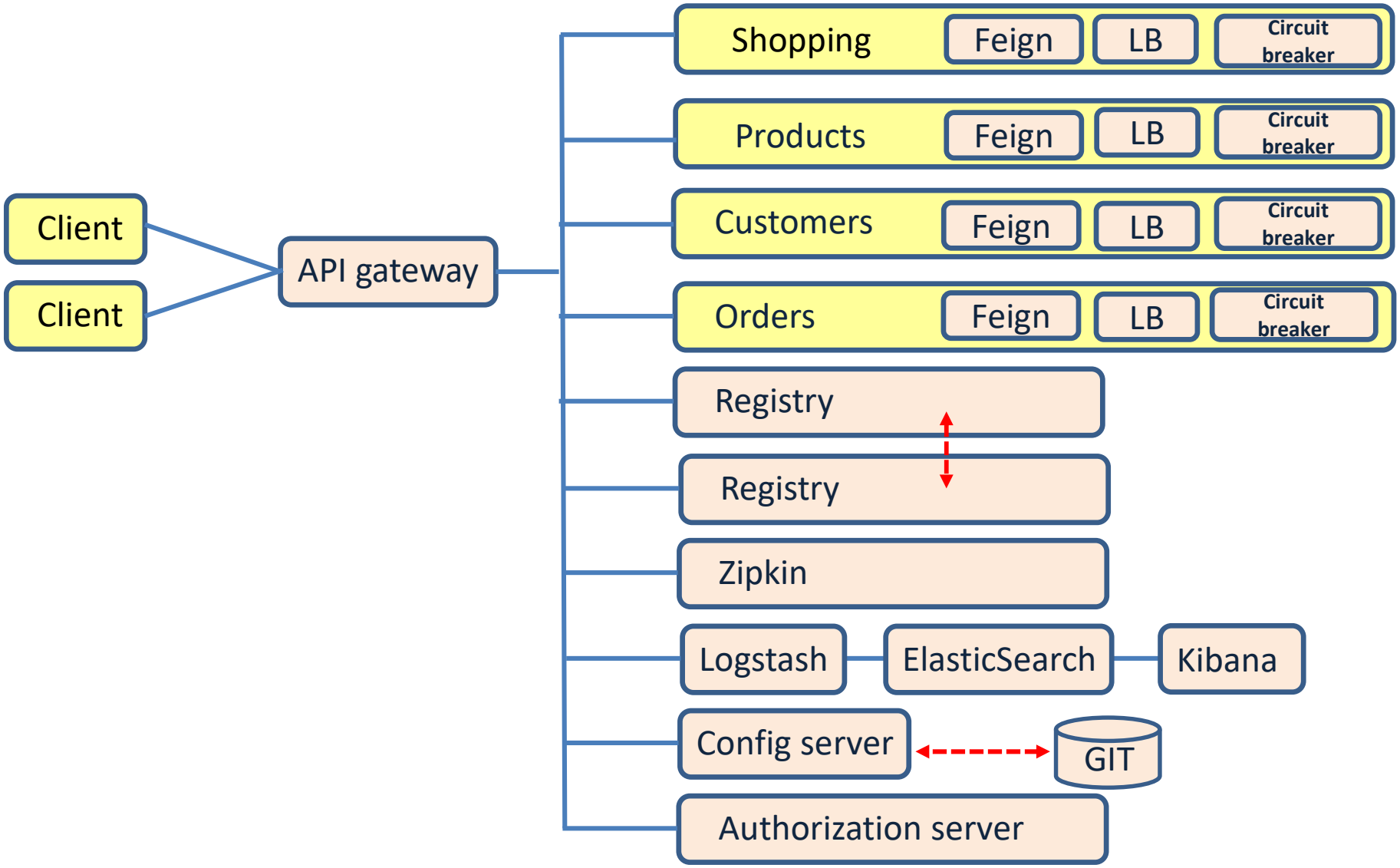
OAuth2 + JWT


Get JWT token providing
username/password
Authorization
server
Call with JWT token
Application A
Call with JWT token
Application B

Main point

- With JWT a token receiver can verify the correctness of the token using the signature within the token. *The TM technique is the key to transcend and access pure consciousness.*

Implementing microservices



Challenges of a microservice architecture

Challenge	Solution
Complex communication	Feign Registry API gateway
Performance	
Resilience	Registry replicas Load balancing between multiple service instances Circuit breaker
Security	Token based security (OAuth2) Digitally signed (JWT) tokens
Transactions	Compensating transactions Eventual consistency
Keep data in sync	Publish-subscribe data change event
Keep interfaces in sync	Spring cloud contract
Keep configuration in sync	Config server
Monitor health of microservices	ELK + beats
Follow/monitor business processes	Zipkin ELK