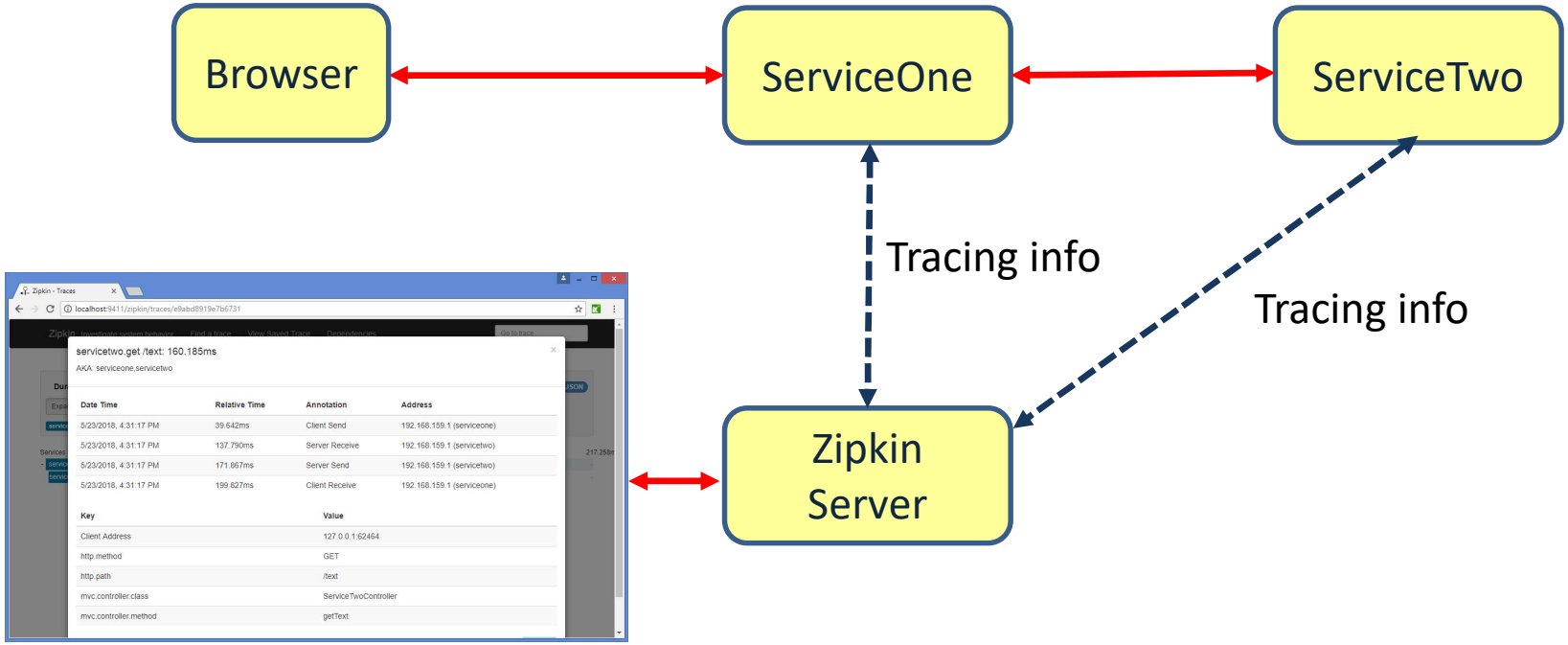
Lesson 9

MICROSERVICES

DISTRIBUTED TRACING: ZIPKIN

Distributed Tracing

- One central place where one can see the end-to-end tracing of all communication between services



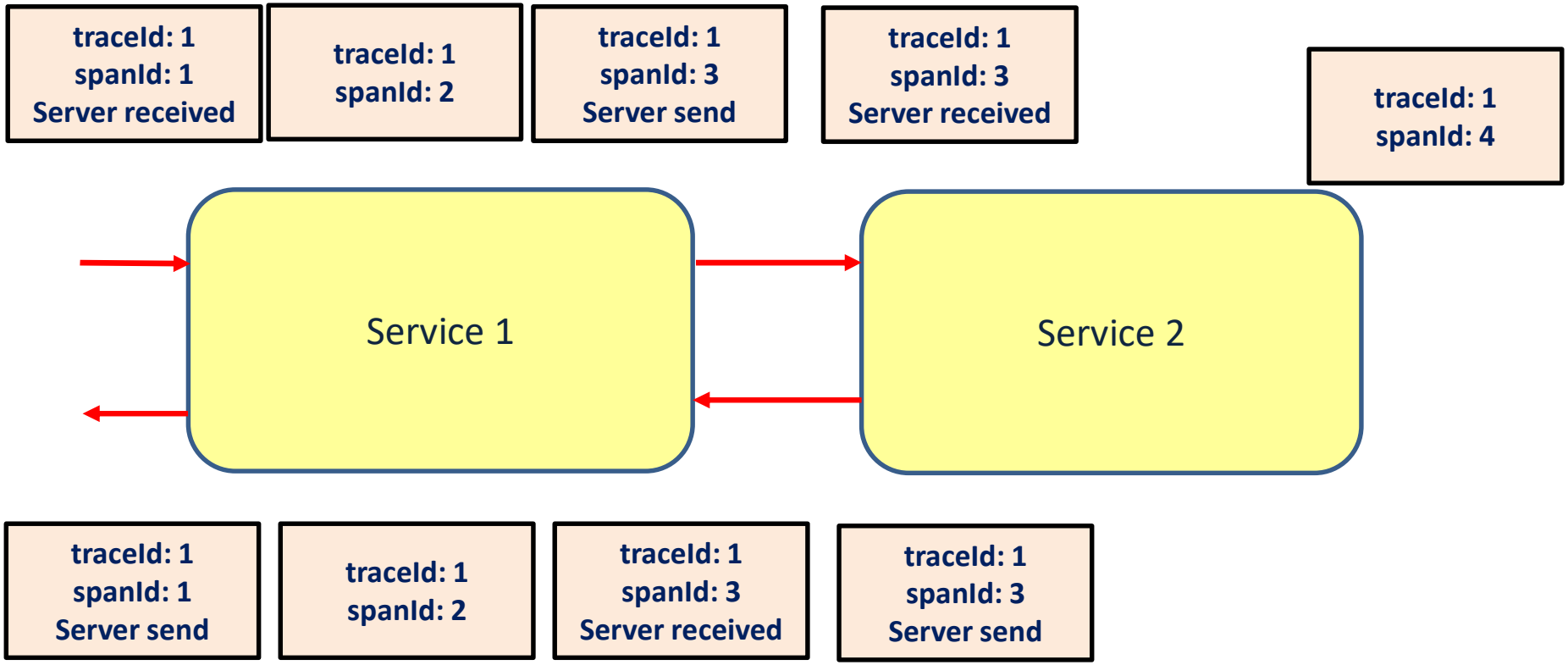
Spring cloud Sleuth

- Adds unique id's to a request so we can trace the request
  - Span id: id for an individual operation
  - Trace id: id for a set of spans

- Also embeds these unique id's to log messages

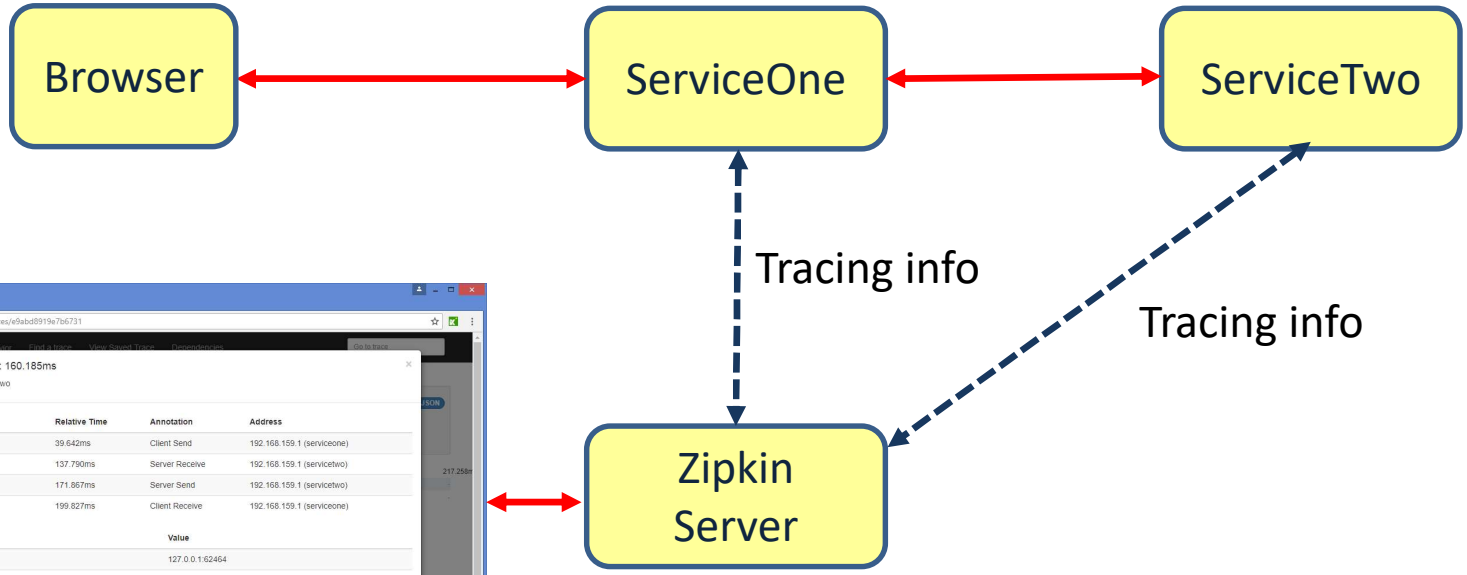
Spring cloud Sleuth

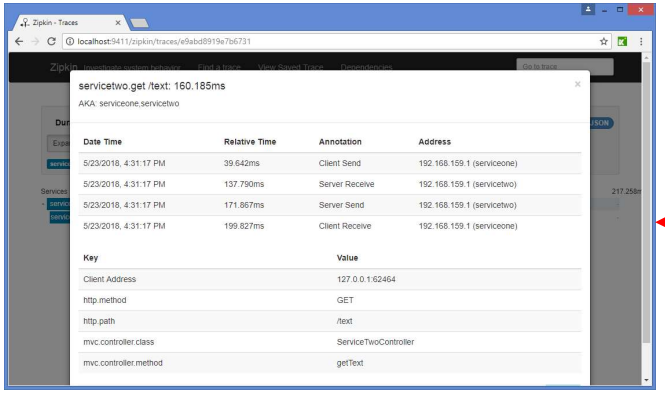
- Span: an individual operation
- Trace: a set of spans



Zipkin

- Centralized tracing server
  - Collects tracing information
- Zipkin console shows the data





Service1

```
@SpringBootApplication
public class Service1Application {
  public static void main(String[] args) {
    SpringApplication.run(Service1Application.class, args);
  }
}
```

```
@RestController
public class ServiceOneController {

  @Autowired
  RestTemplate restTemplate;

  @RequestMapping("/text")
  public String getText() {
    String service2Text = restTemplate.getForObject("http://localhost:9091/text",
                                                    String.class);
    return "Hello "+ service2Text;
  }

  @Bean
  RestTemplate getRestTemplate() {
    return new RestTemplate();
  }
}
```

Service1

application.yml

```yaml
server:
  port: 9090

spring:
  application:
    name: ServiceOne
  zipkin:
    base-url: http://localhost:9411/

  sleuth:
    sampler:
      probability: 1 #100% (default = 10%)
```

Service1

pom.xml

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-web</artifactId>
</dependency>
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-sleuth</artifactId>
</dependency>
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-zipkin</artifactId>
  <version>2.2.3.RELEASE</version>
</dependency>
```

Service2

```
@SpringBootApplication
public class Service2Application {
  public static void main(String[] args) {
    SpringApplication.run(Service2Application.class, args);
  }
}
```

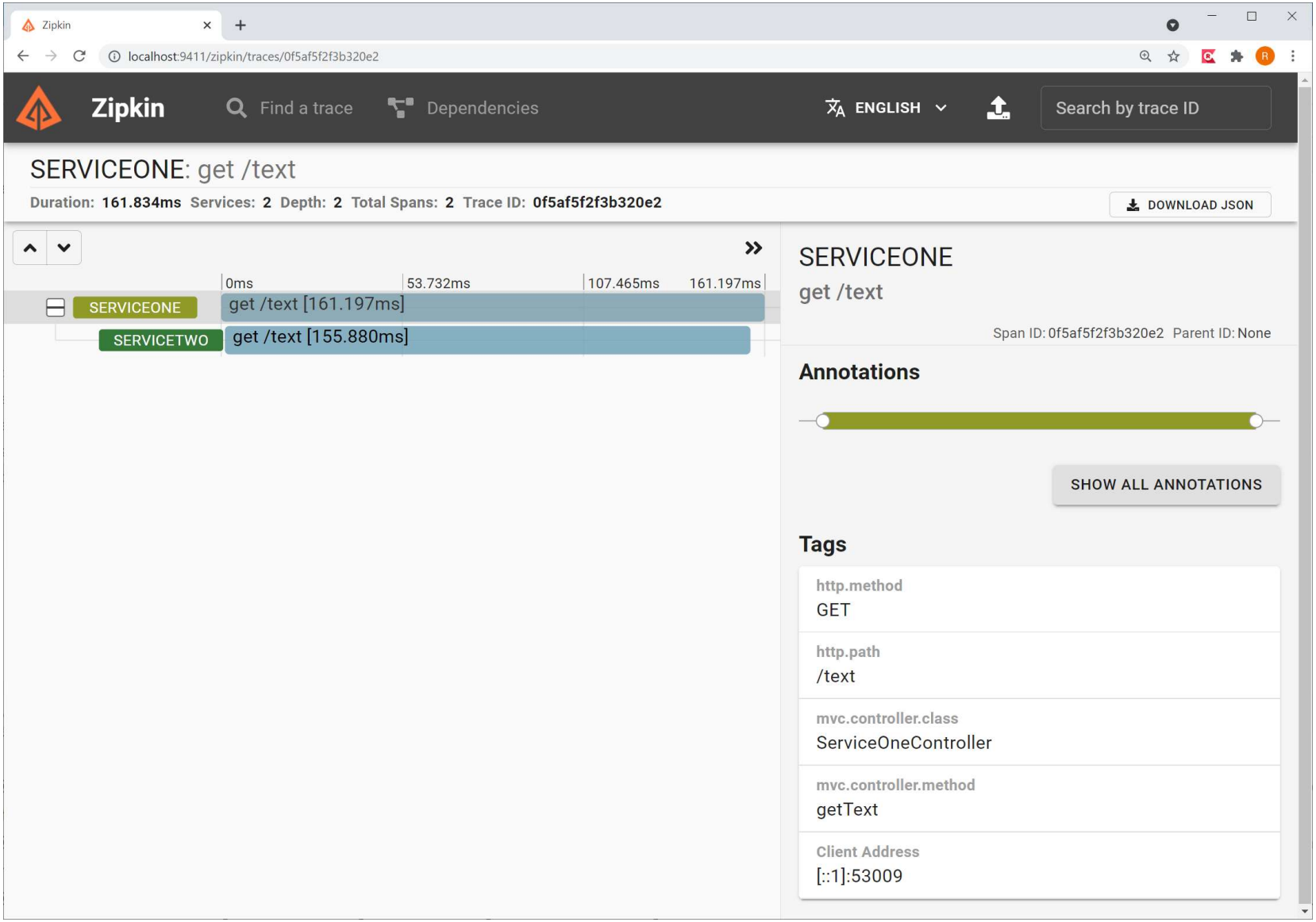
```
@RestController
public class ServiceTwoController {

  @RequestMapping("/text")
  public String getText() {
    return "World";
  }
}
```

Service2

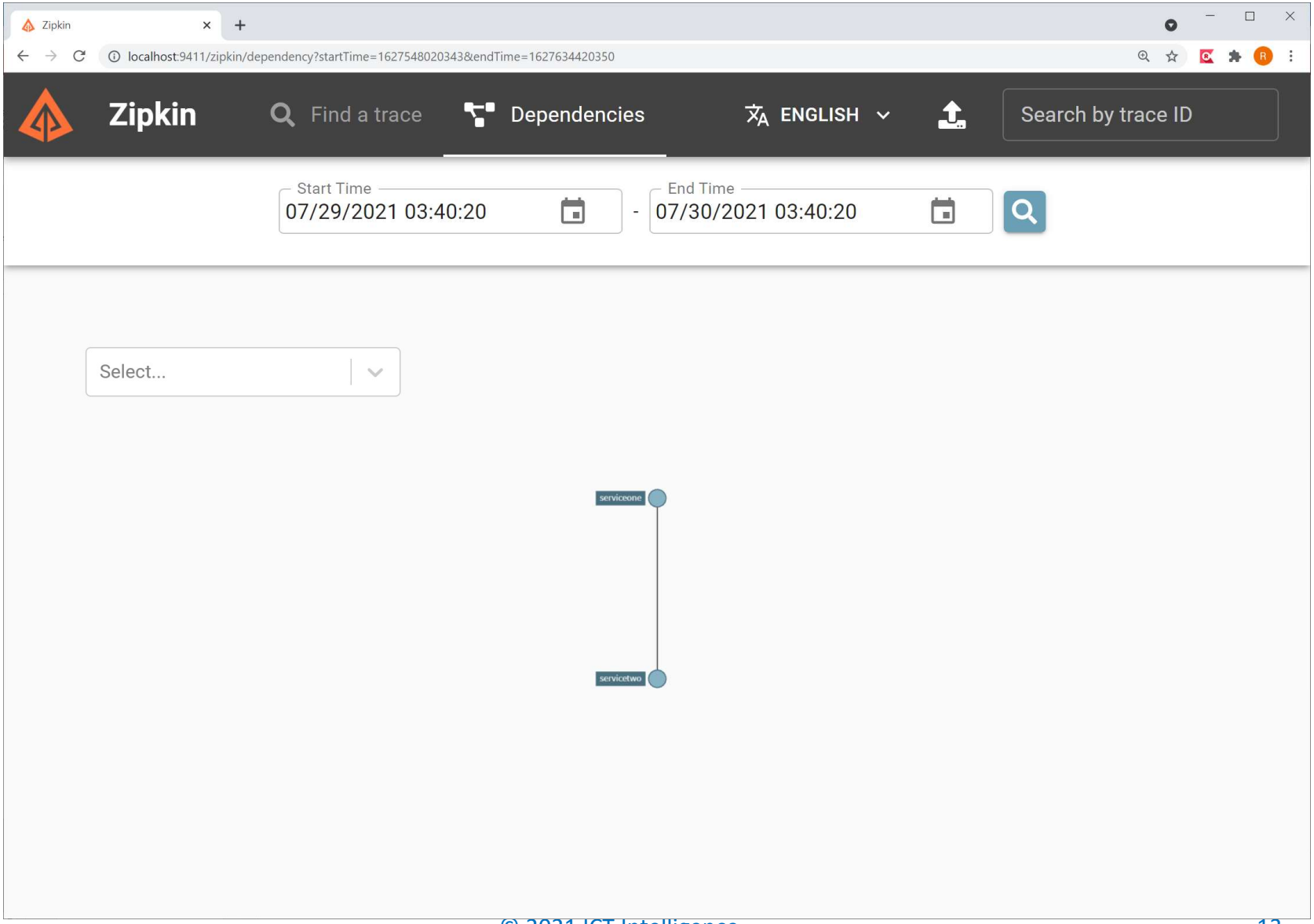
application.yml

```yaml
server:
  port: 9091

spring:
  application:
    name: ServiceTwo
  zipkin:
    base-url: http://localhost:9411/

sleuth:
  sampler:
    probability: 1 #100% (default = 10%)
```

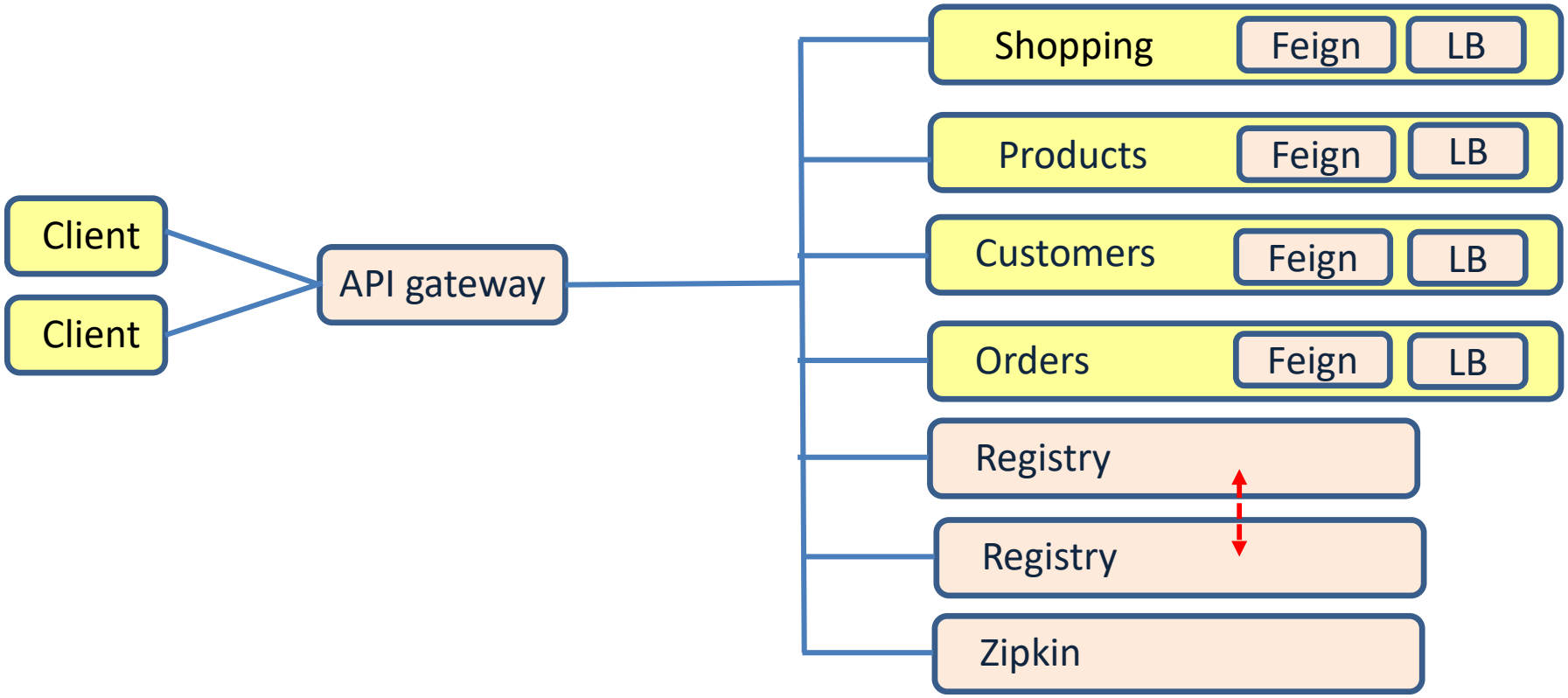
Zipkin console



Zipkin console



Implementing microservices



Challenges of a microservice architecture

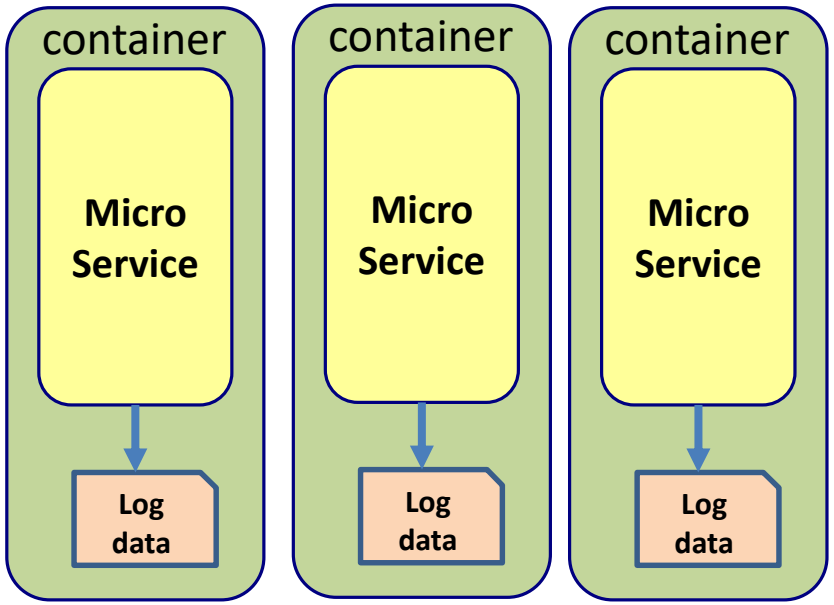
Challenge	Solution
Complex communication	Feign Registry API gateway
Performance	
Resilience	Registry replicas Load balancing between multiple service instances
Security	
Transactions	
Following the process	
Keep data in sync	
Keep interfaces in sync	
Keep configuration in sync	
Monitor health of microservices	
Follow/monitor business processes	Zipkin

Main point

- We need zipkin in order to monitor and debug service-to-service communication
- The Unified Field is the field of perfection

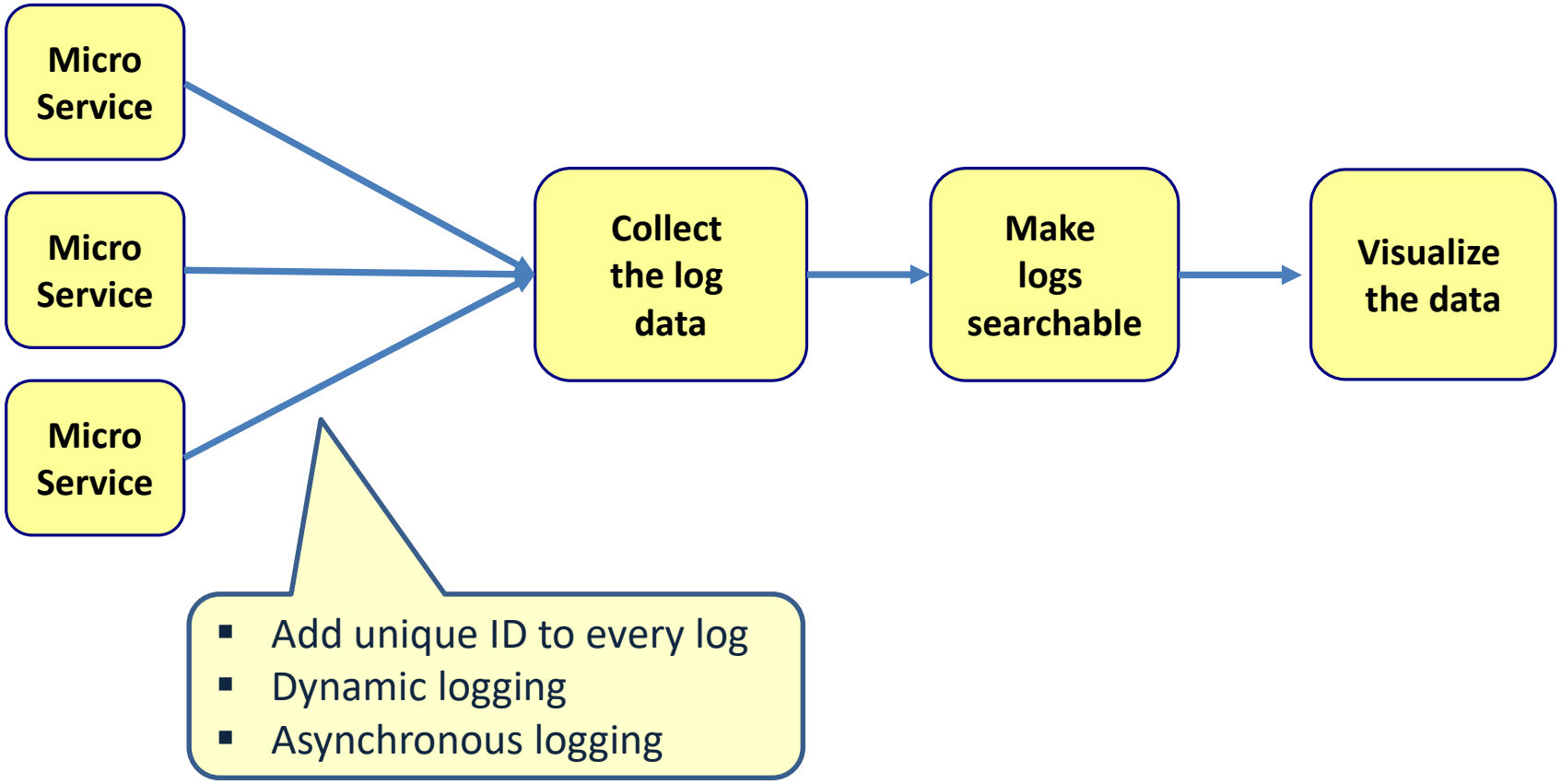
DISTRIBUTED LOGGING
ELK STACK

The need for centralized logging



- Local logging does not work
  - Containers come and go
  - Containers have no fixed address

Microservice logging architecture



ELK stack

Micro Service

Micro Service

Micro Service

Collect the log data

Make logs searchable

Visualize the data

logstash

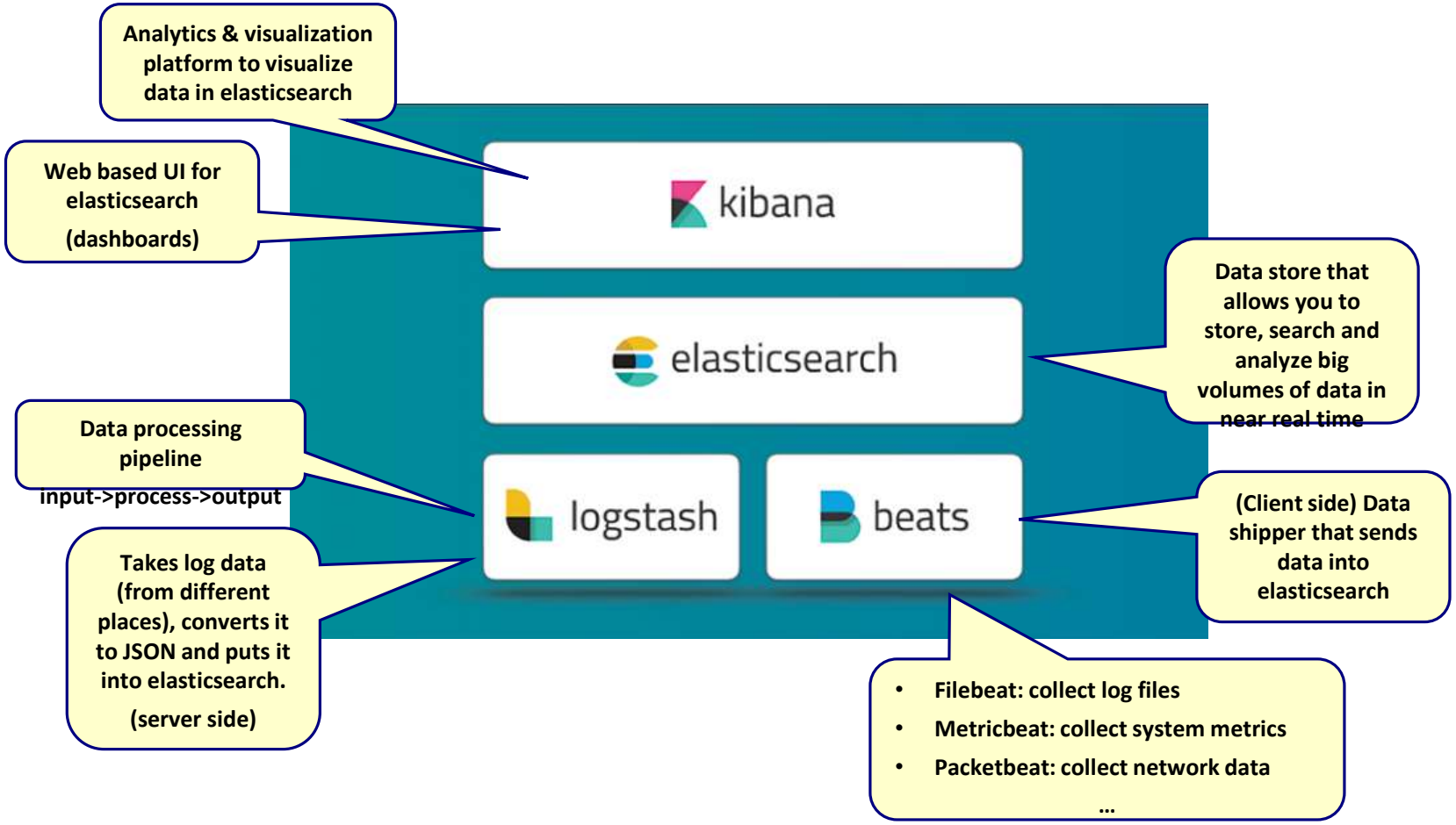
elasticsearch

kibana

Collect logs
Transform
Time normalization

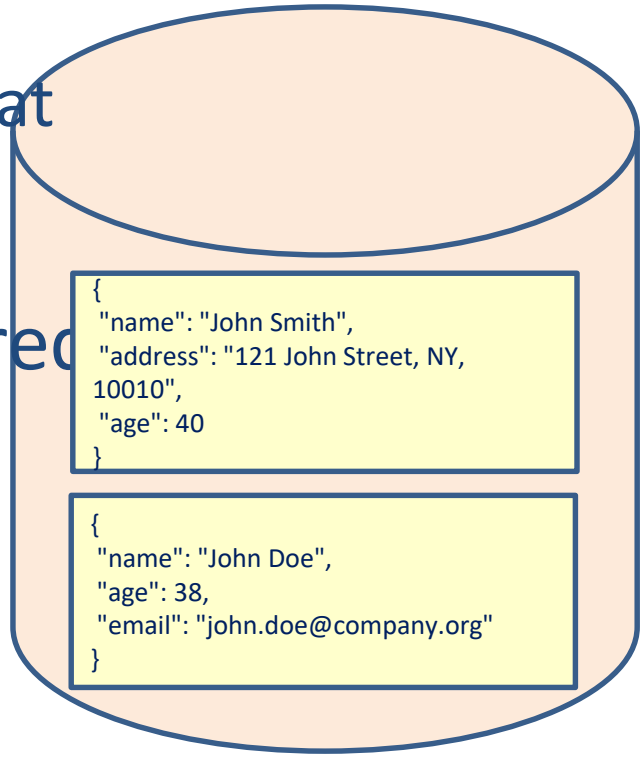
Schema less search DB
Highly scalable
Fast searching

Elastic stack components



What is Elasicsearch?

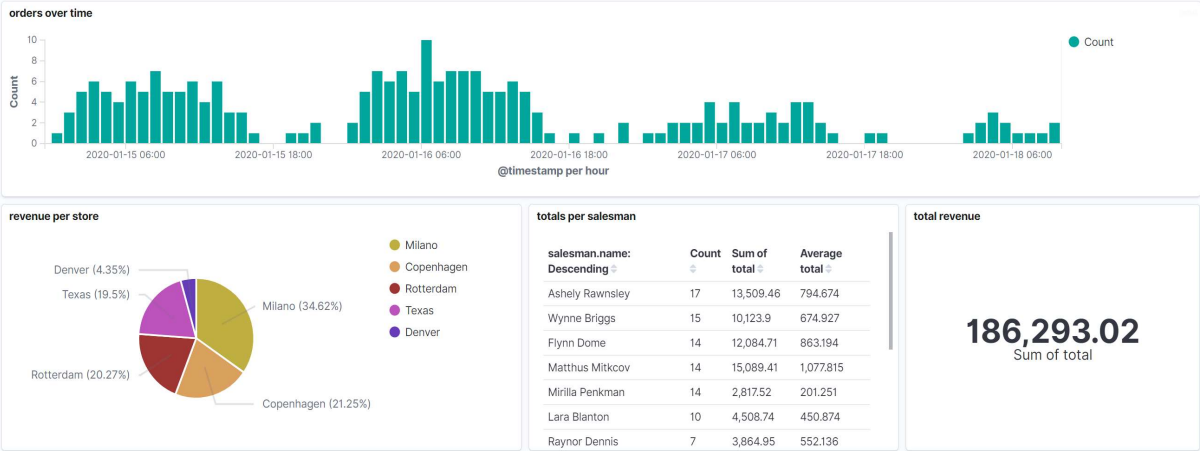
- Database
  - Data is stored as documents
  - Data is structured in JSON format
- Full text search engine
- Analytics platform for structured



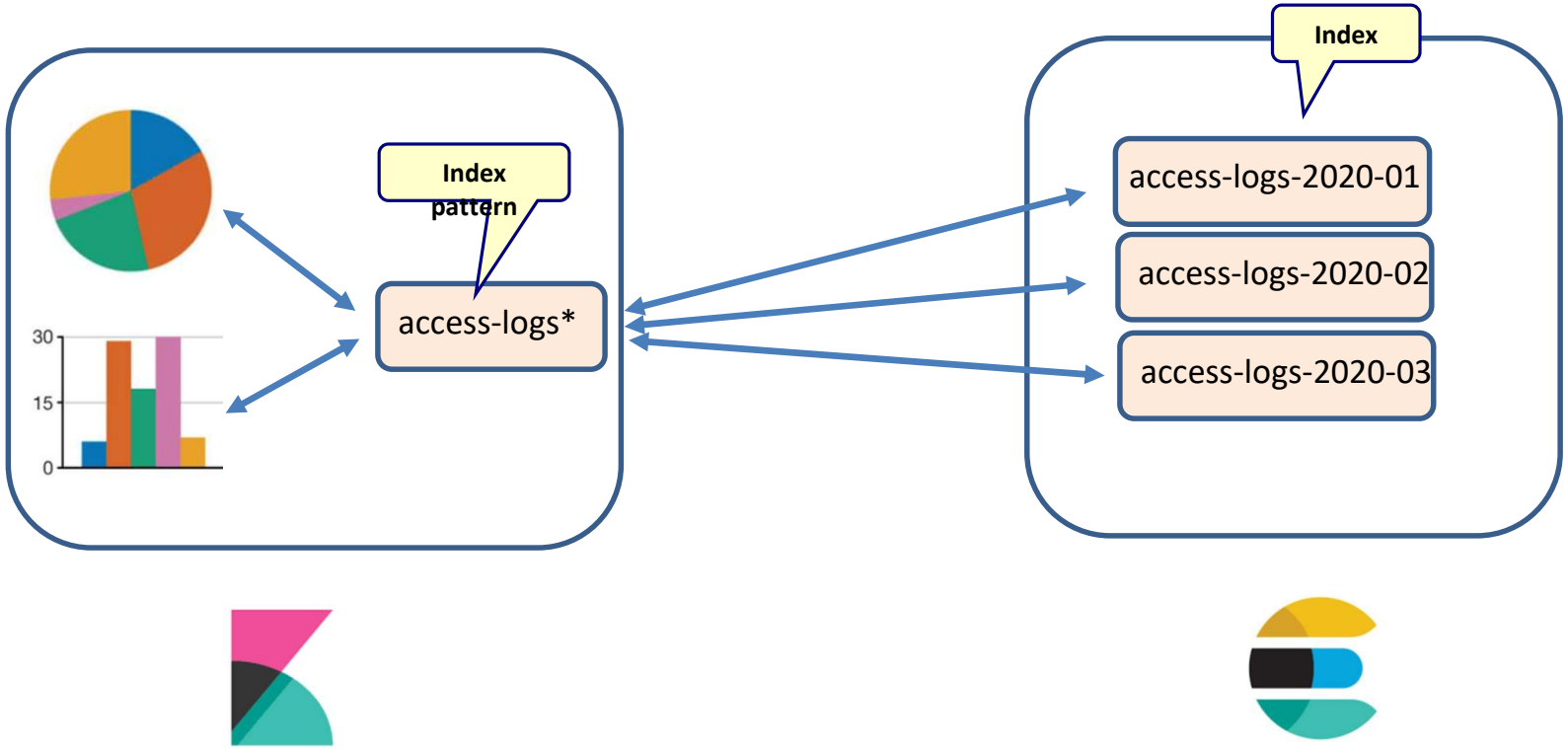
KIBANA

Kibana

- Web UI on top of elasticsearch
- Has its own Kibana query language (KQL)
- Objects (Queries, visualizations, dashboards, etc.) are saved in elasticsearch

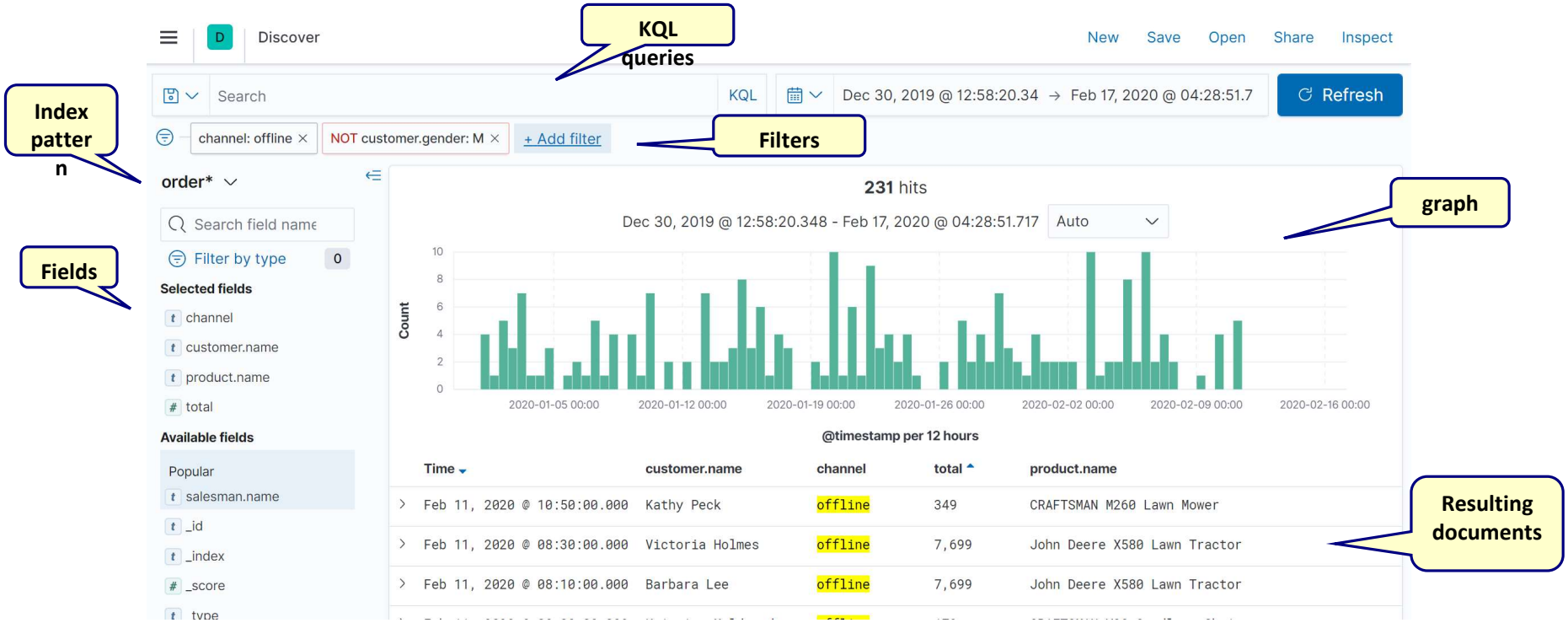


Index patterns

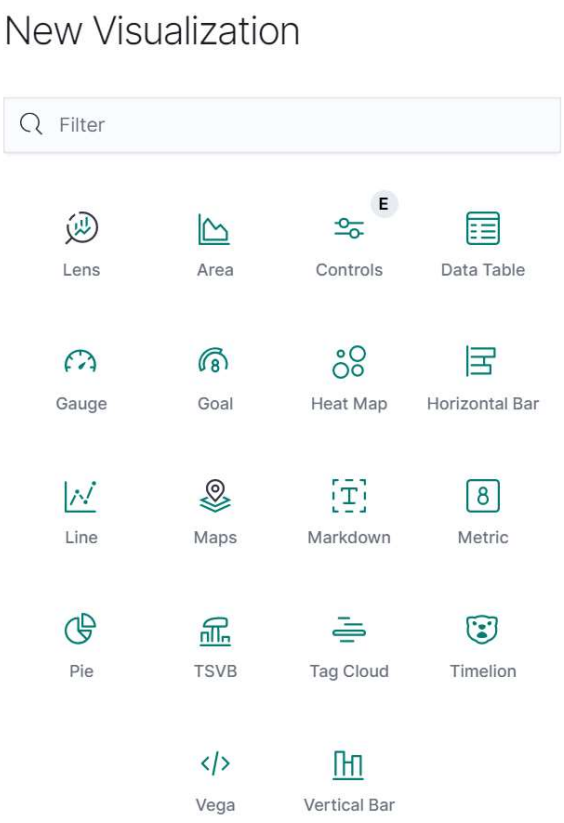


Discover app

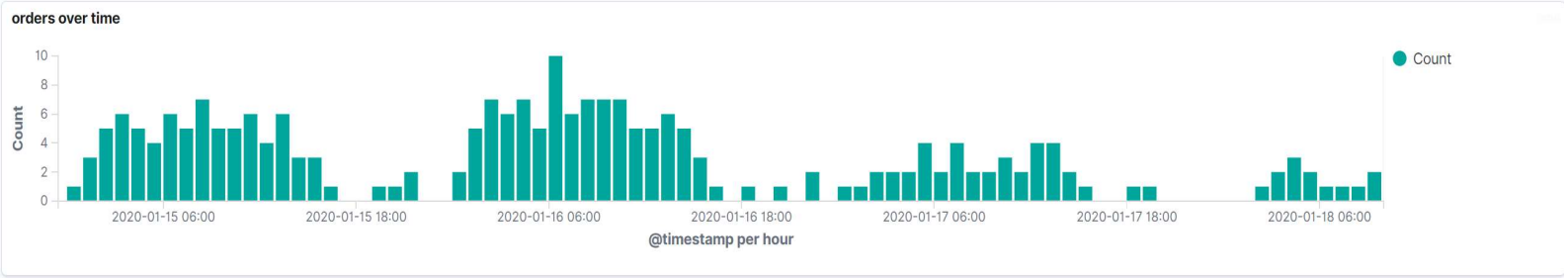
- Good for exploring and analyzing data

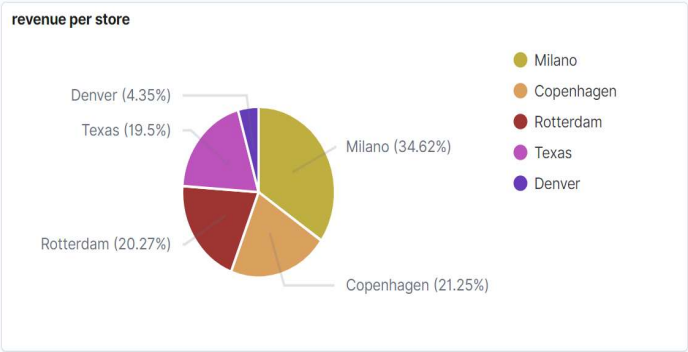


Visualizations



Dashboard





totals per salesman

salesman.name: Descending	Count	Sum of total	Average total
Ashely Rawnsley	17	13,509.46	794.674
Wynne Briggs	15	10,123.9	674.927
Flynn Dome	14	12,084.71	863.194
Matthus Mitkcov	14	15,089.41	1,077.815
Mirilla Penkman	14	2,817.52	201.251
Lara Blanton	10	4,508.74	450.874
Raynor Dennis	7	3,864.95	552.136

total revenue

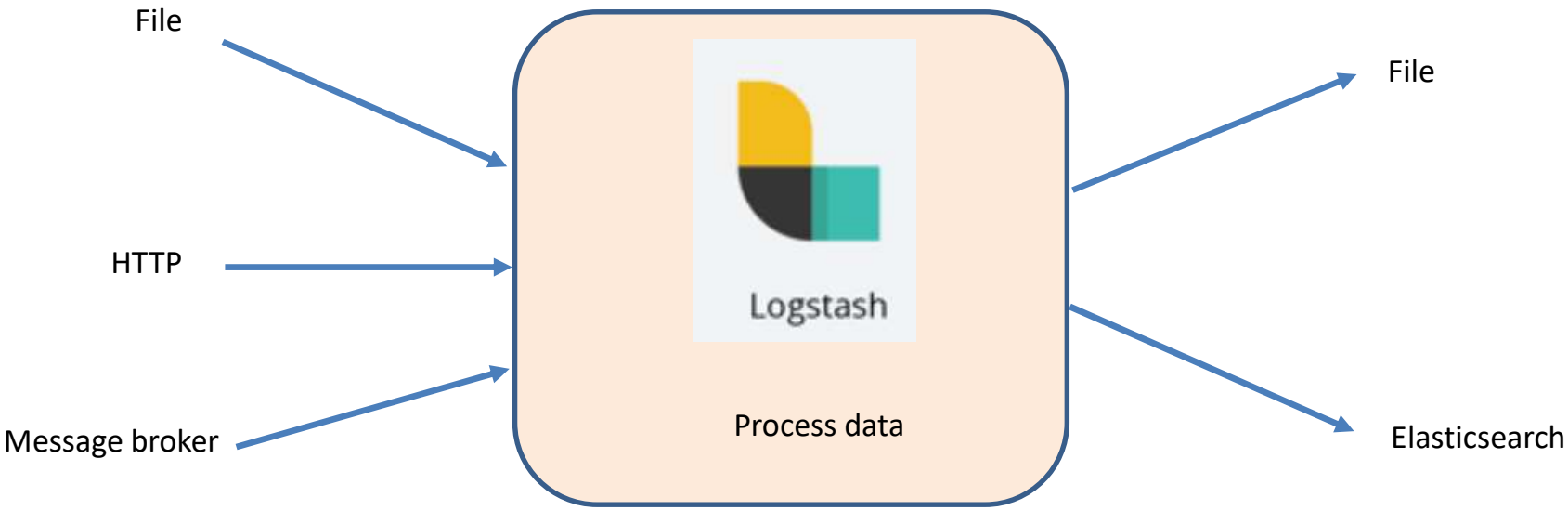
186,293.02
Sum of total

Contains visualizations

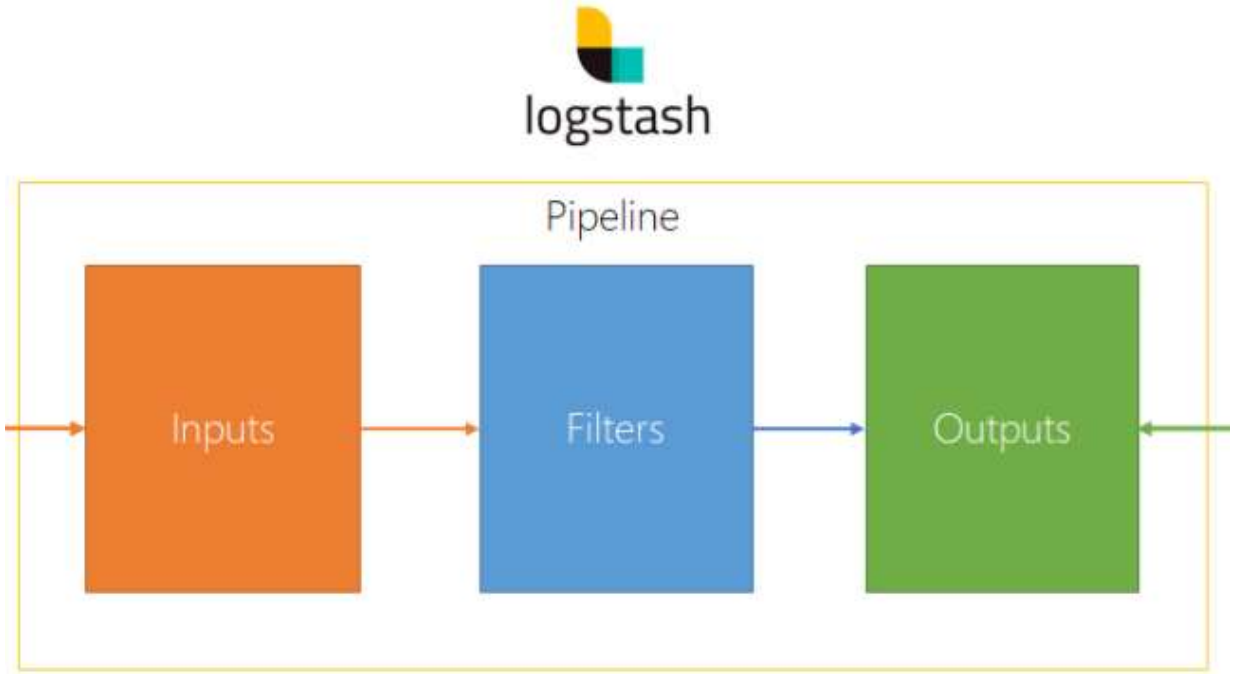
LOGSTASH

Logstash

- Event processing engine



How does Logstash work?



Logstash configuration

```
input {
   ...
}

filter {
   ...
}

output {
   ...
}
```

- **file**: This streams log events from a file
- **redis**: This streams events from a redis instance
- **stdin**: This streams events from standard input
- **syslog**: This streams syslog messages over the network
- **ganglia**: This streams ganglia packets over the network via udp
- **lumberjack**: This receives events using the lumberjack protocol
- **eventlog**: This receives events from Windows event log
- **s3**: This streams events from a file from an s3 bucket
- **elasticsearch**: This reads from the Elasticsearch cluster based on results of a search query

- **date**: This is used to parse date fields from incoming events, and use that as Logstash timestamp fields, which can be later used for analytics
- **drop**: This drops everything from incoming events that matches the filter condition
- **grok**: This is the most powerful filter to parse unstructured data from logs or events to a structured format
- **multiline**: This helps parse multiple lines from a single source as one Logstash event
- **dns**: This filter will resolve an IP address from any fields specified
- **mutate**: This helps rename, remove, modify, and replace fields in events
- **geoip**: This adds geographic information based on IP addresses that are retrieved from `Maxmind` database

- **file**: This writes events to a file on disk
- **e-mail**: This sends an e-mail based on some conditions whenever it receives an output
- **elasticsearch**: This stores output to the Elasticsearch cluster, the most common and recommended output for Logstash
- **stdout**: This writes events to standard output
- **redis**: This writes events to redis queue and is used as a broker for many ELK implementations
- **mongodb**: This writes output to mongodb
- **kafka**: This writes events to Kafka topic

Logstash configuration

input.txt

```
Hello world
```

pipeline.conf

```
input {
  file {
    path => "C:/elasticsearchtraining/temp/input.txt"
    start_position => "beginning"
  }
}

output {
  stdout {
    codec => rubydebug
  }
  file {
    path => "C:/elasticsearchtraining/temp/output.txt"
  }
}
```

output.txt


```
{
"host":"DESKTOP-BVHRK6K",
"@version":"1",
"path":"C:/elasticsearchtraining/temp/input.txt",
"message":"Hello world\r",
"@timestamp":"2021-01-16T13:52:32.726Z"
}
```


Anytime this file changes, read from this file

Write the output to the console

Write the output to the specified file

Logstash configuration

input.txt

```
Hi there
```

pipeline.conf

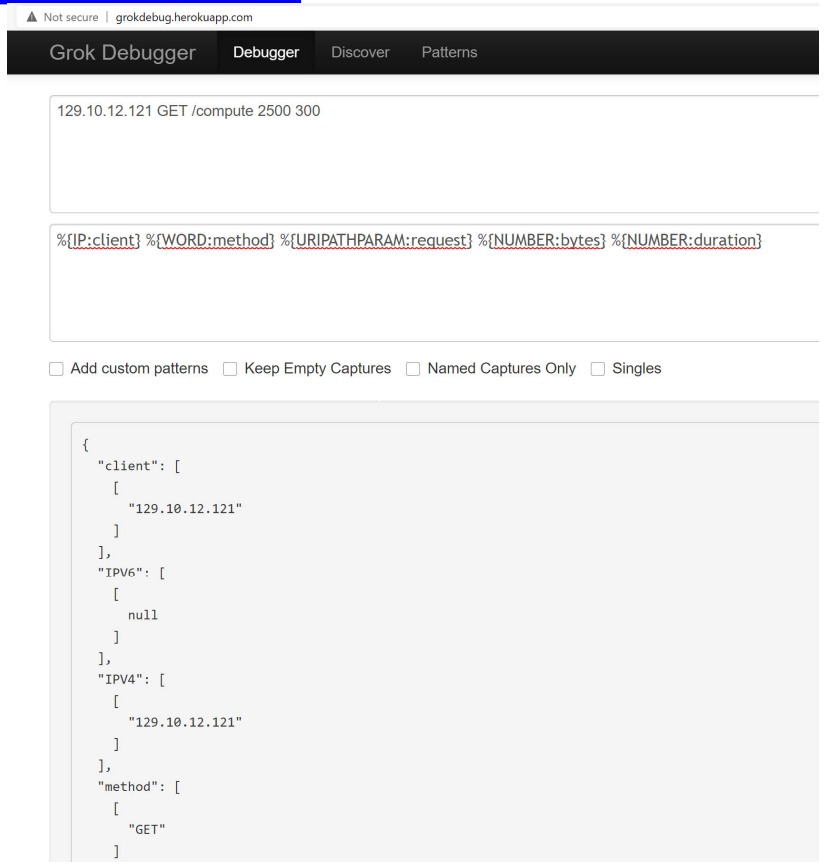
```
input {
 file {
  path => "C:/elasticsearchtraining/temp/input.txt"
        start_position => "beginning"
 }
}

filter {
  mutate {
    uppercase => ["message"]
  }
}

output {
 stdout {
  codec => rubydebug
 }
 file {
  path => "C:/elasticsearchtraining/temp/output.txt"
 }
}
```

output.txt

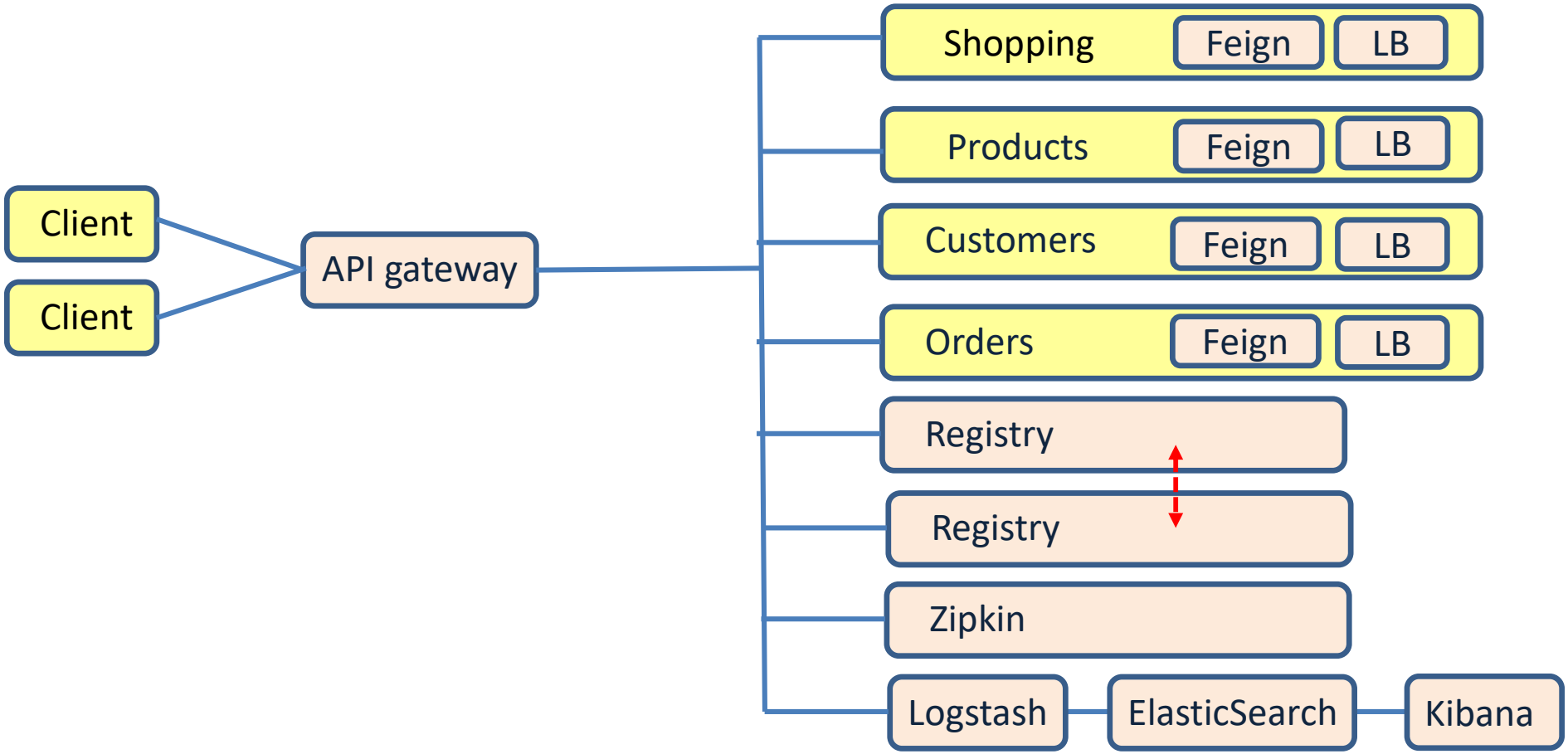

```
{
"path":"C:/elasticsearchtraining/temp/input.txt",
"message":"HI THERE\r",
"host":"DESKTOP-BVHRK6K",
"@version":"1",
"@timestamp":"2021-01-16T14:17:10.537Z"
}
```

Testing grok

- http://grokdebug.herokuapp.com/



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
Security	
Transactions	
Following the process	
Keep data in sync	
Keep interfaces in sync	
Keep configuration in sync	
Monitor health of microservices	
Follow/monitor business processes	Zipkin ELK

RESILIENCE

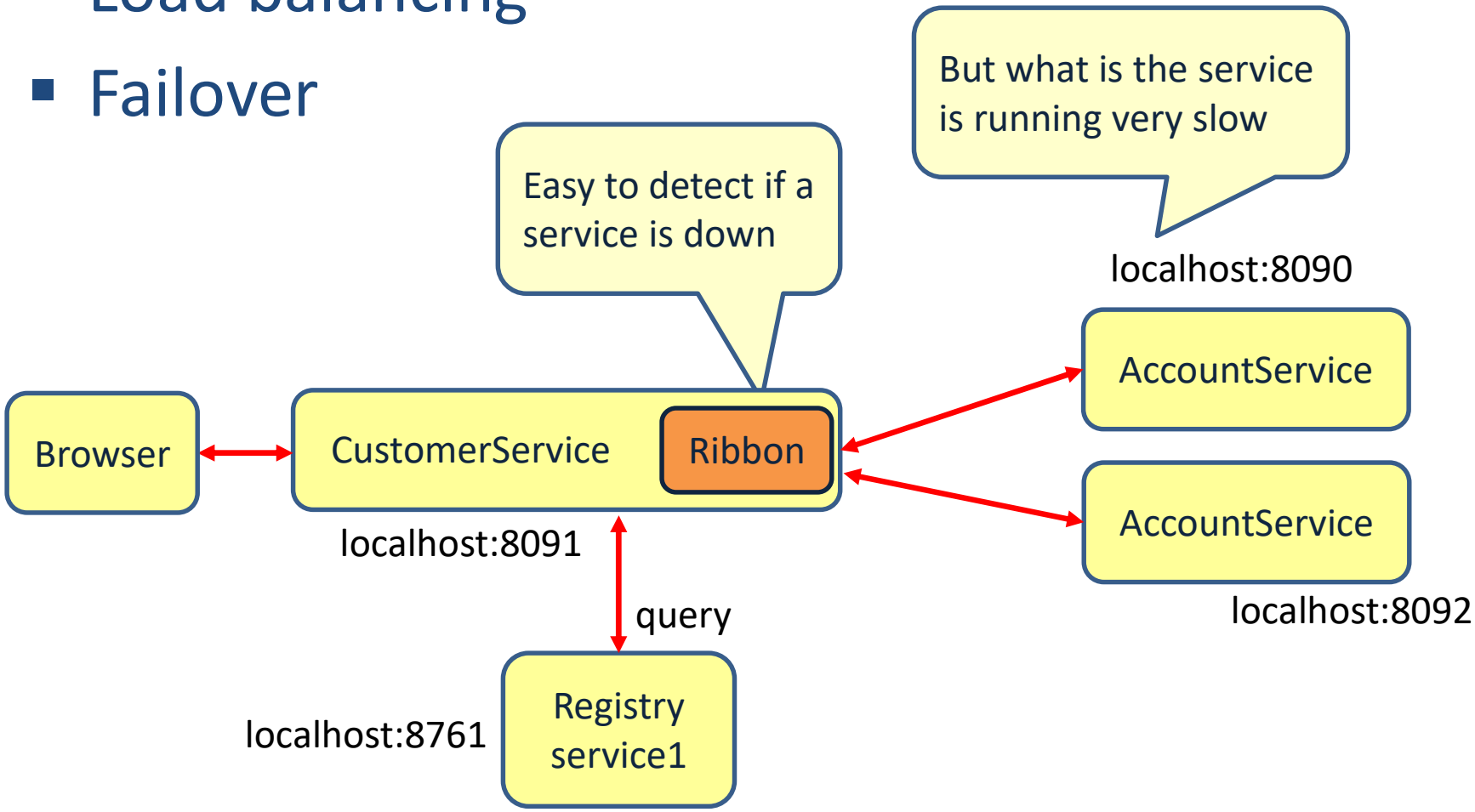
The ability to recover from failures

Fallacies of distributed computing

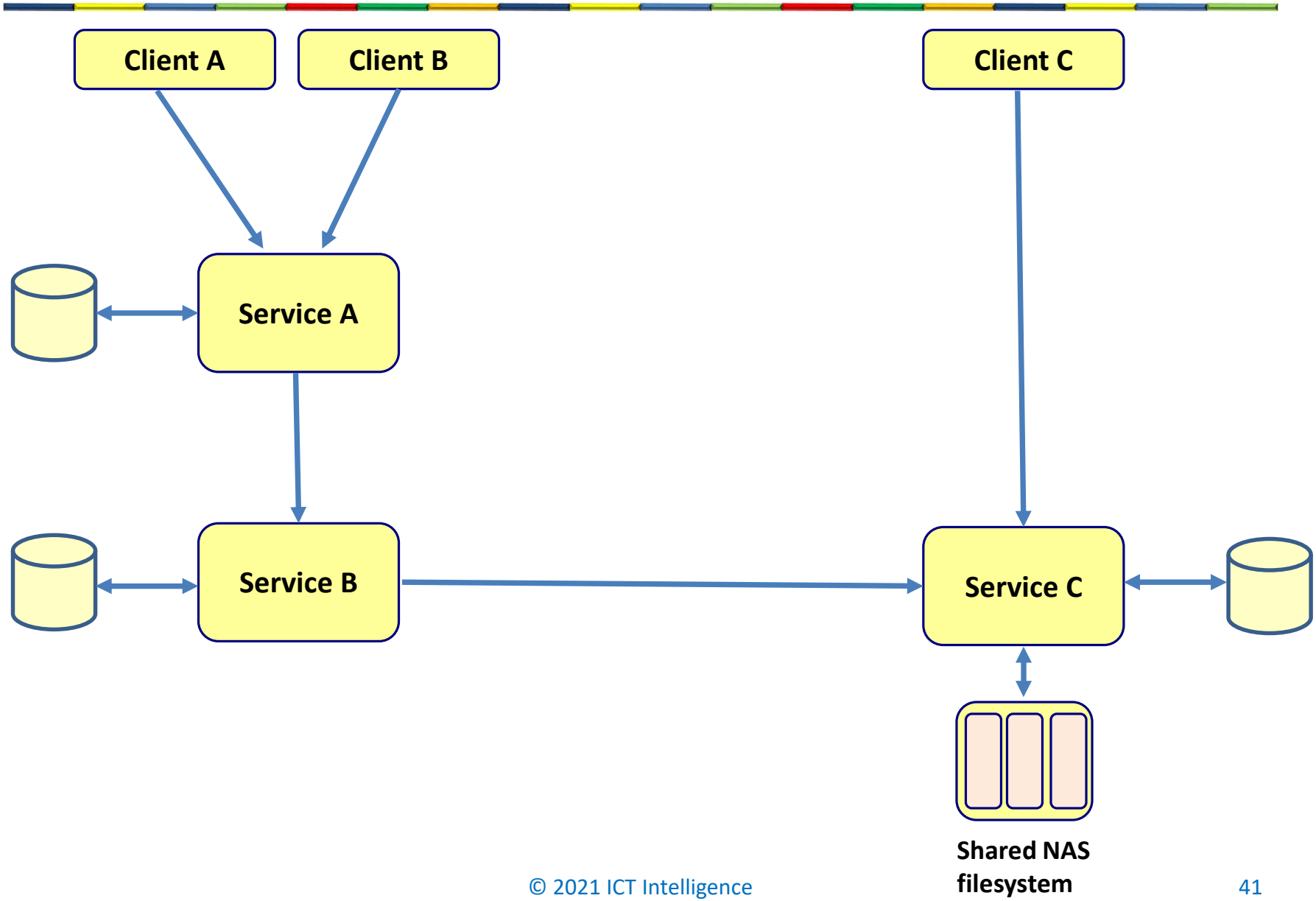
- The network is reliable
- Latency is zero
- Bandwidth is infinite
- The network is secure
- Topology doesn’t change
- There is one administrator
- Transport cost is zero
- The network is homogeneous

Clustering

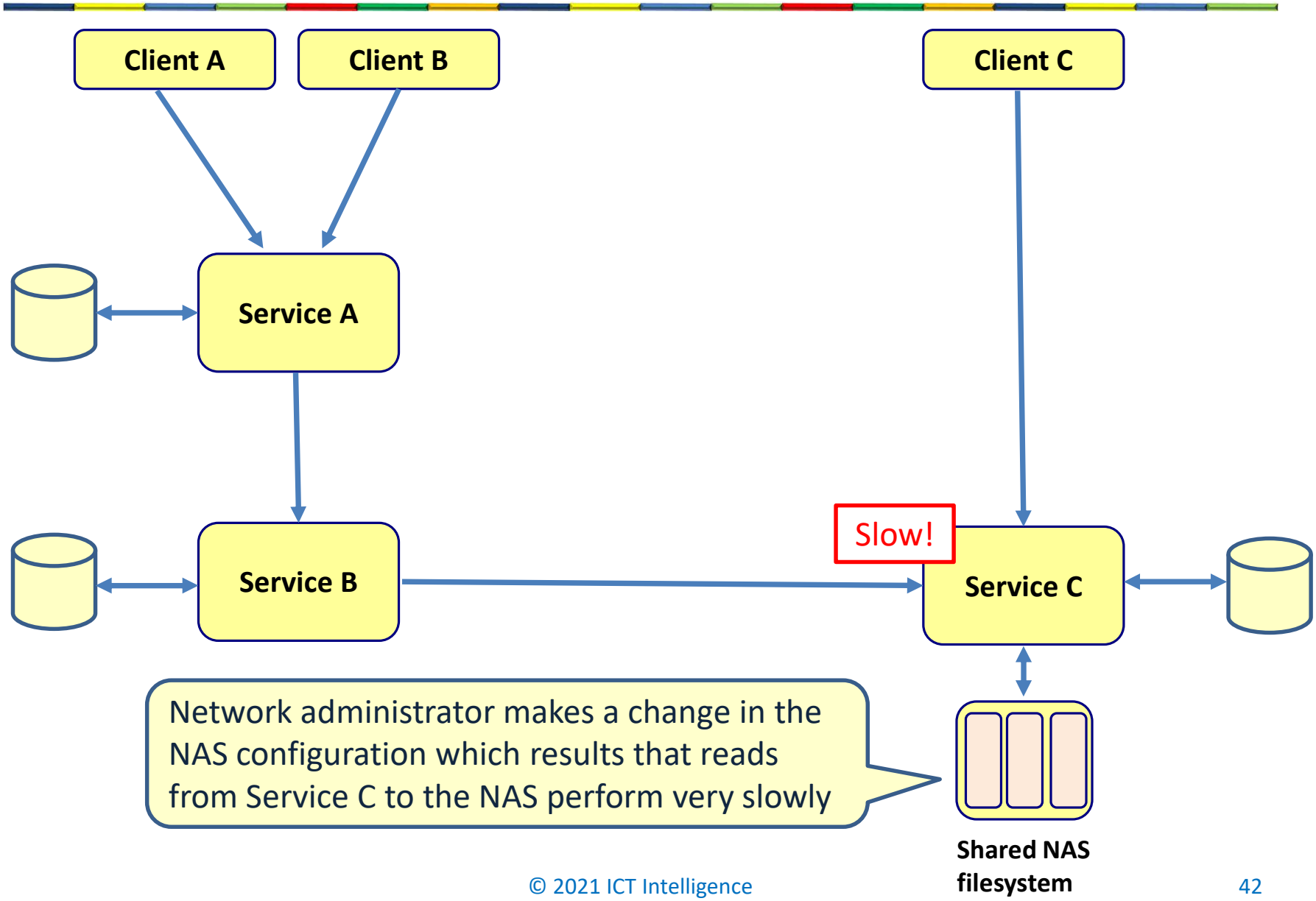
- Load balancing
- Failover



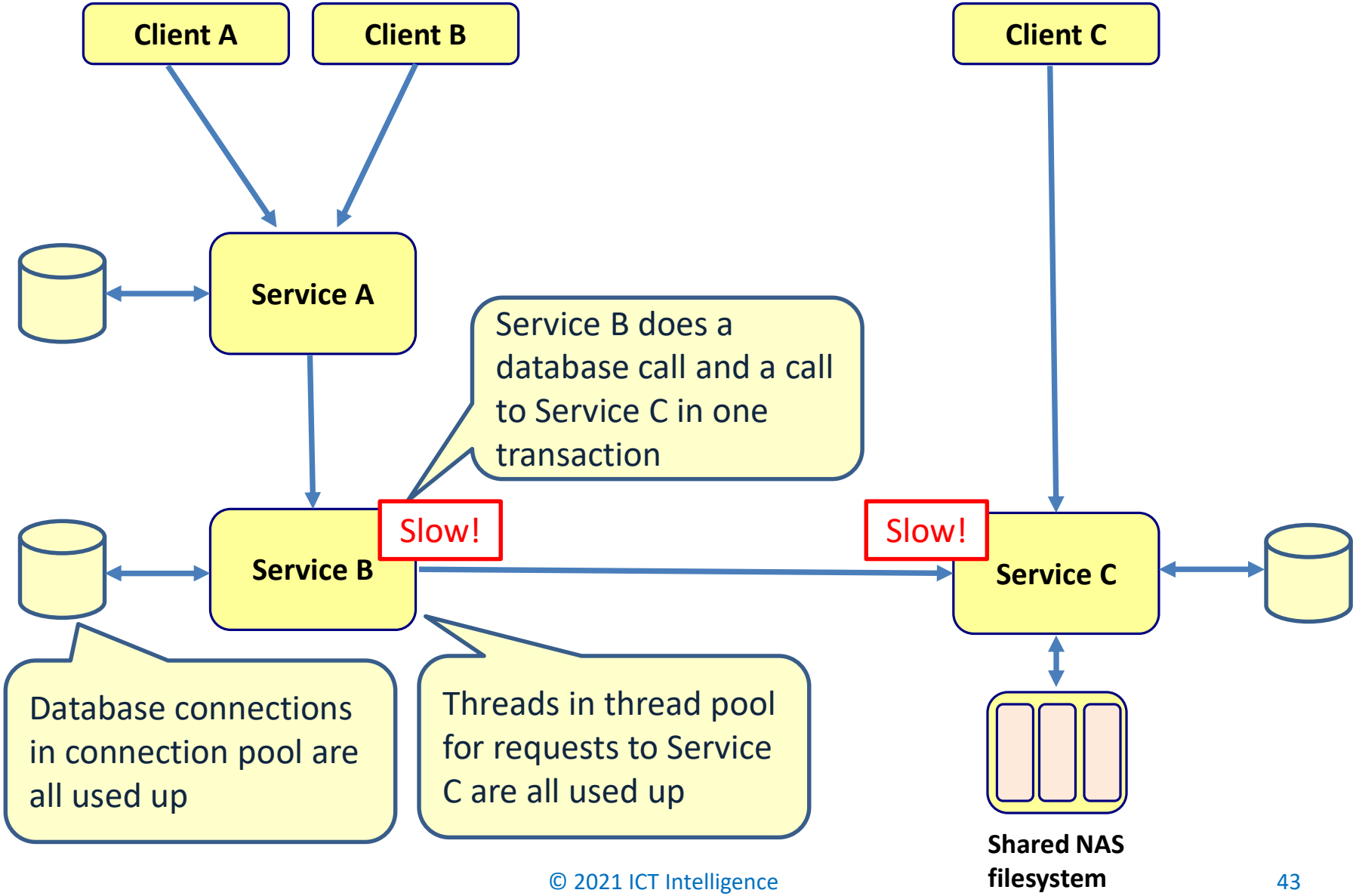
Example



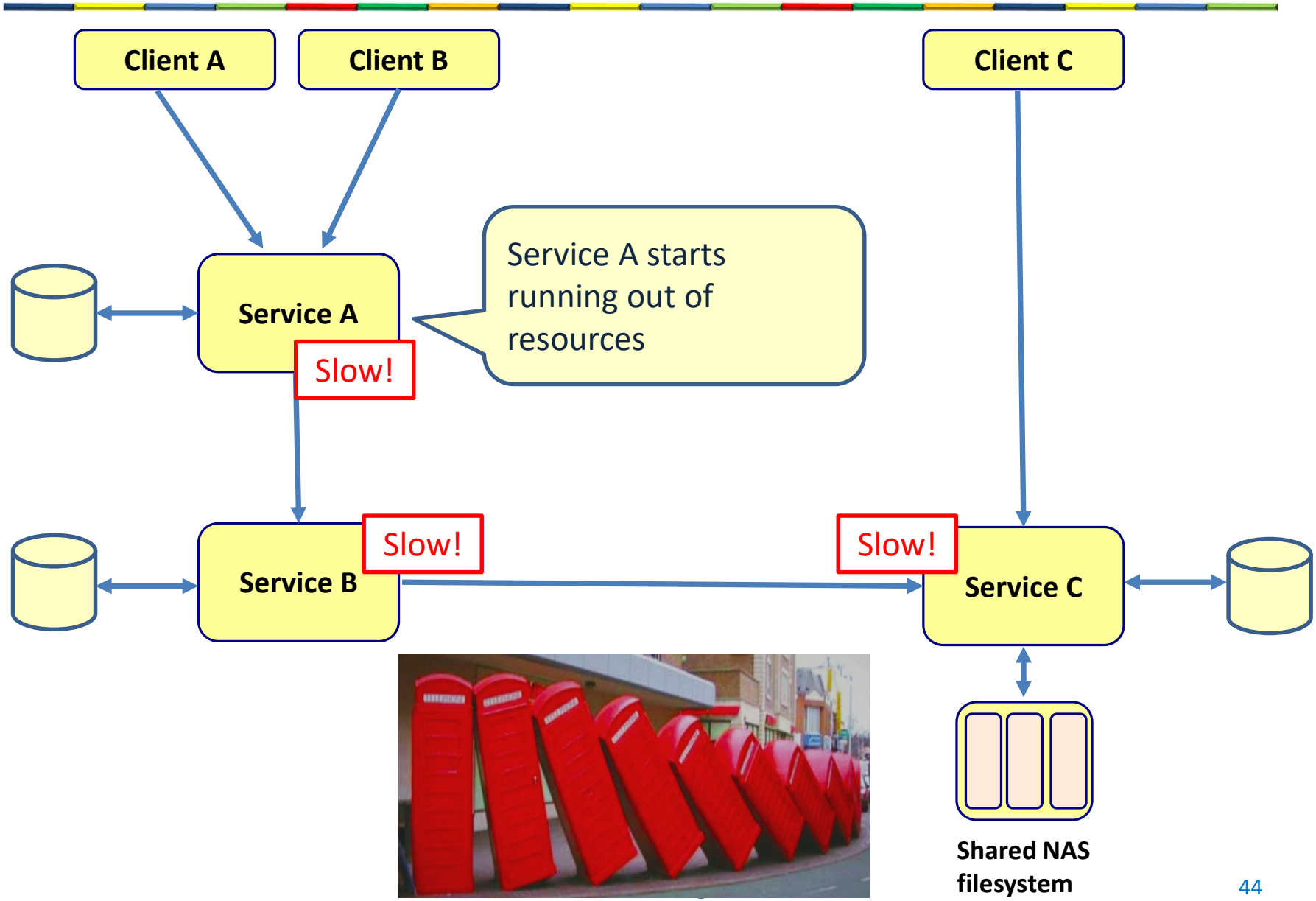
Example



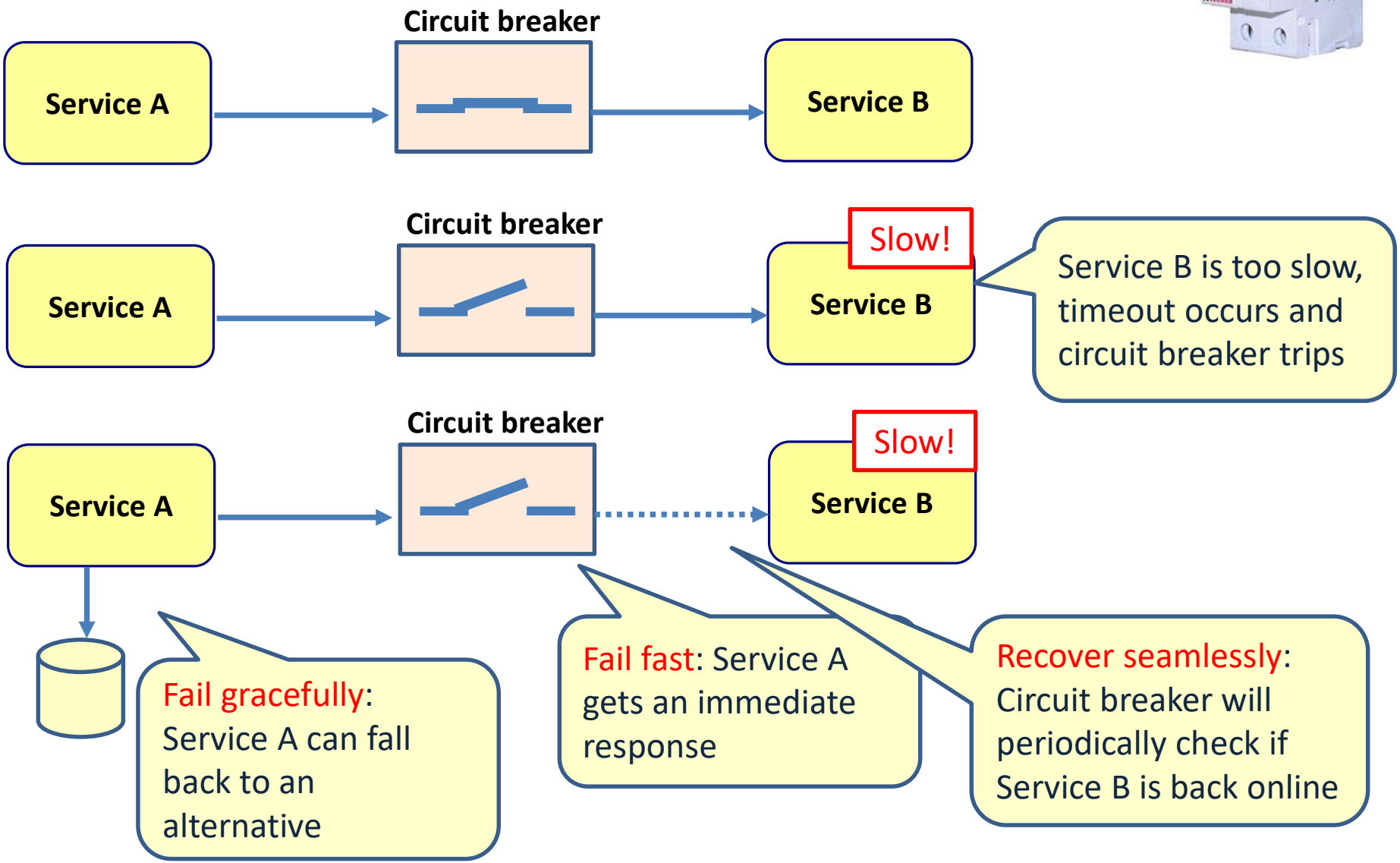
Example



Example


Client A
Client B
Client C
Service A
Slow!
Service A starts running out of resources
Service B
Slow!
Slow!
Service C
Shared NAS filesystem

Circuit breaker



Main point

- A circuit breaker takes care that not the whole microservice architecture gets slow when one service becomes slow.
- Every relative part of creation is connected at the level of pure consciousness.

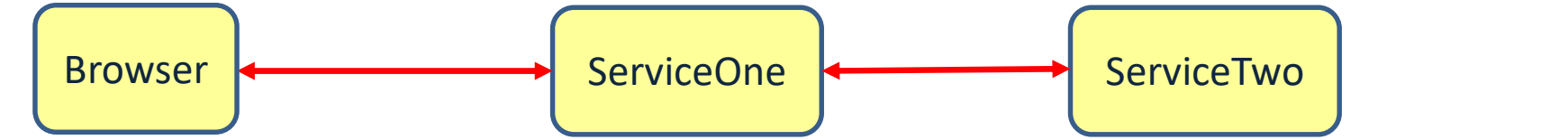
RESILIENCE: HYSTRIX

Hystrix dependency

pom.xml

```
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-netflix-hystrix</artifactId>
</dependency>
```

Enable Hystrix




```
@SpringBootApplication
@EnableCircuitBreaker
public class ServiceOneApplication {

  public static void main(String[] args) {
    SpringApplication.run(Service2Application.class, args);
  }
}
```

Enable Hystrix

```
@SpringBootApplication
@EnableCircuitBreaker
public class ServiceTwoApplication {

  public static void main(String[] args) {
    SpringApplication.run(Service2Application.class, args);
  }
}
```

Enable Hystrix

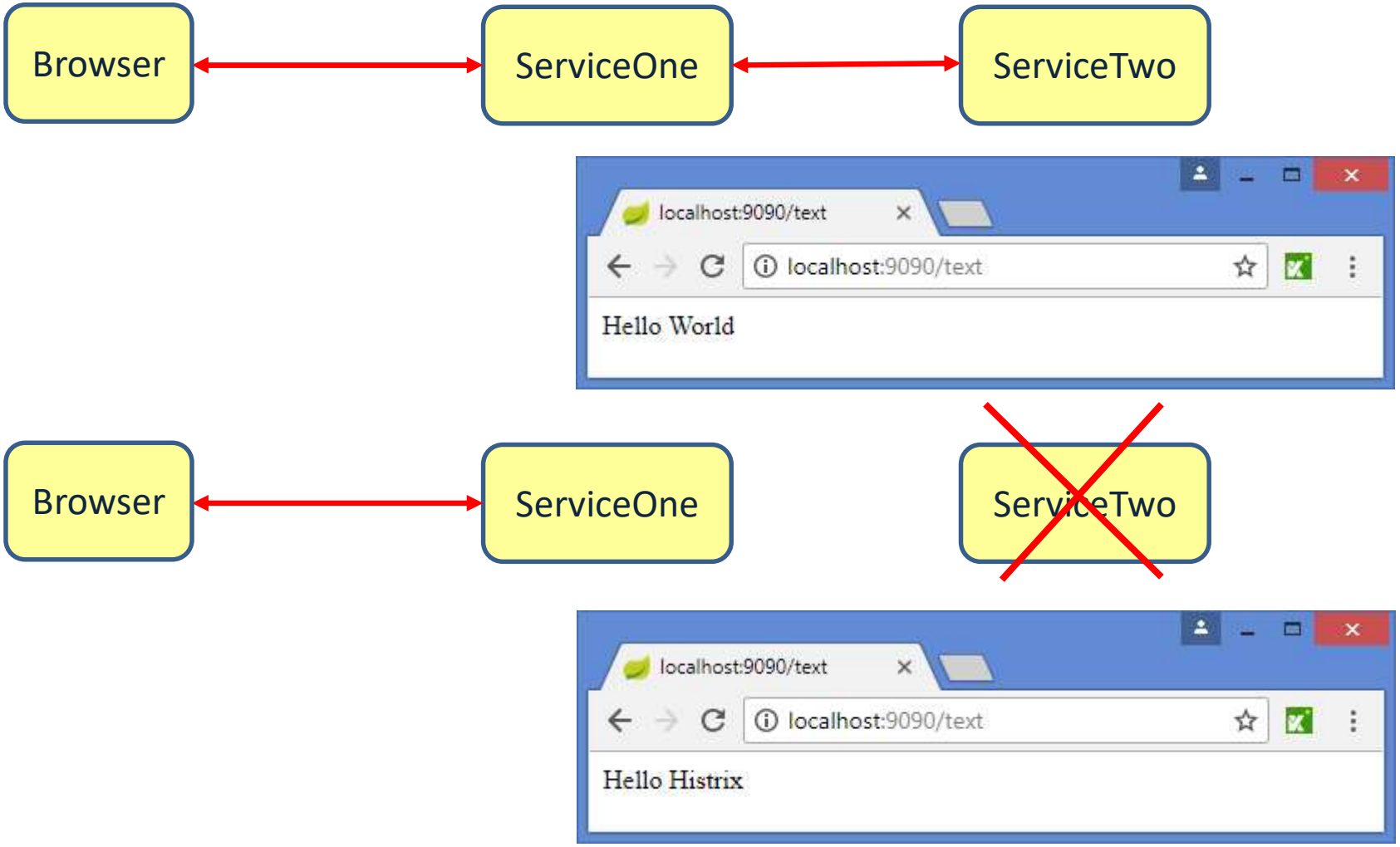
Using the circuit breaker

```
public class ServiceOneController {

  @Autowired
  RestTemplate restTemplate;

  @RequestMapping("/text")
  @HystrixCommand(fallbackMethod = "getTextFallback")
  public String getText() {
    String service2Text = restTemplate.getForObject("http://localhost:9091/text",
                                                    String.class);
    return "Hello "+ service2Text;
  }

  public String getTextFallback() {
   return "Hello Histrix";
  }

  @Bean
  RestTemplate getRestTemplate() {
    return new RestTemplate();
  }
}
```

If this method throws an exception or takes longer than 2 seconds, call the fallback method

Fallback method

Using Histrix



Setting the timeout

```
public class ServiceOneController {

  @Autowired
  RestTemplate restTemplate;

  @RequestMapping("/text")
  @HystrixCommand(fallbackMethod = "getTextFallback", commandProperties=
  {@HystrixProperty(name="execution.isolation.thread.timeoutInMilliseconds",
                    value="4000")})
  public String getText() {
    String service2Text = restTemplate.getForObject("http://localhost:9091/text",
                                                    String.class);
    return "Hello "+ service2Text;
  }

  public String getTextFallback() {
   return "Hello Histrix";
  }

  @Bean
  RestTemplate getRestTemplate() {
    return new RestTemplate();
  }
}
```

Set timeout to 4 seconds

Setting the timeout

```
@RestController
public class ServiceTwoController {

  @RequestMapping("/text")
  public String getText() throws InterruptedException {
    Thread.sleep(5000);
    return "World";
  }
}
```

Sleep of 5 seconds

RESILIENCE: RESILIENCE4J

dependency

pom.xml

```
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-circuitbreaker-resilience4j</artifactId>
  <version>1.0.2.RELEASE</version>
</dependency>
```

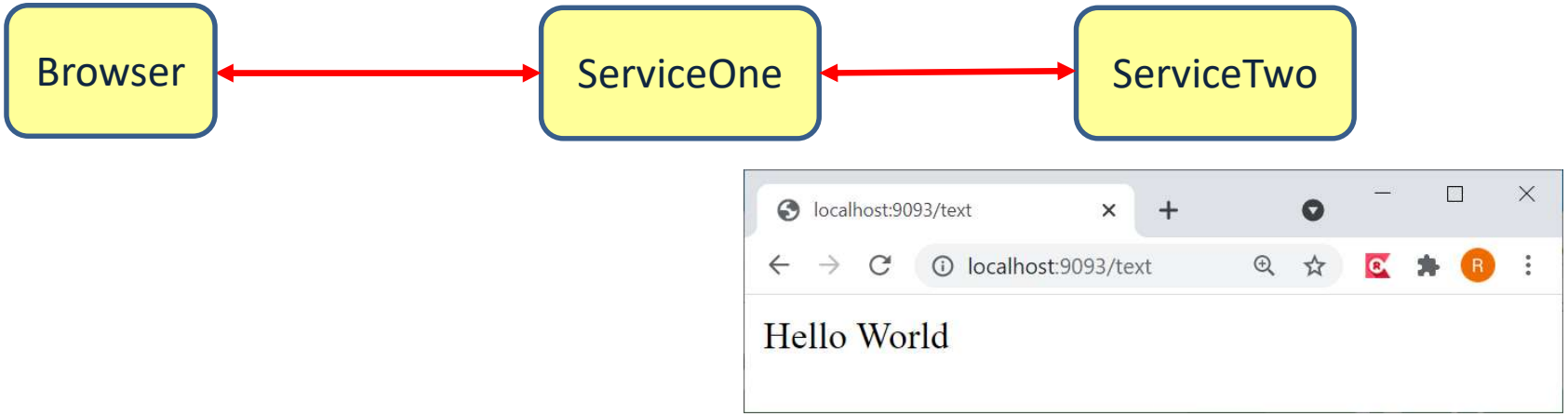
Using the circuit breaker

```
@RestController
public class ServiceOneController {
   @Autowired
   RestTemplate restTemplate;

   @Autowired
   private CircuitBreakerFactory circuitBreakerFactory;

   @RequestMapping("/text")
   public String getText() {
      CircuitBreaker circuitBreaker = circuitBreakerFactory.create("circuitbreaker");

      String service2Text = circuitBreaker.run(() -> restTemplate.getForObject("http://localhost:9091/text",
            String.class), throwable -> getFallbackName());
      return "Hello "+ service2Text;
   }

   private String getFallbackName() {
      return "Fallback World";
   }

   @Bean
   RestTemplate getRestTemplate() {
      return new RestTemplate();
   }
}
```

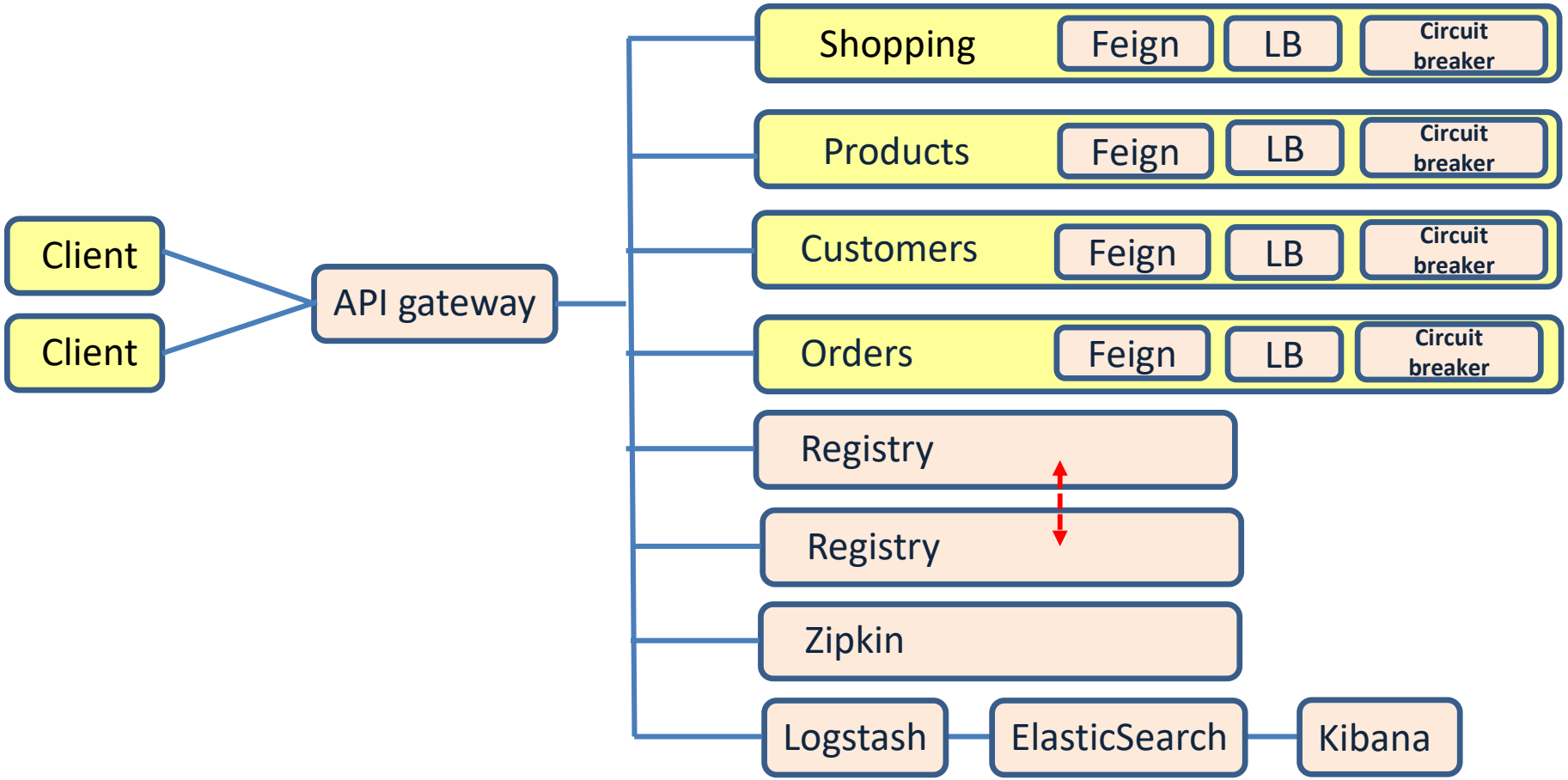
If this method throws an exception or takes longer than 2 seconds, call the fallback method

Fallback method

Using Resilience4J



Implementing microservices

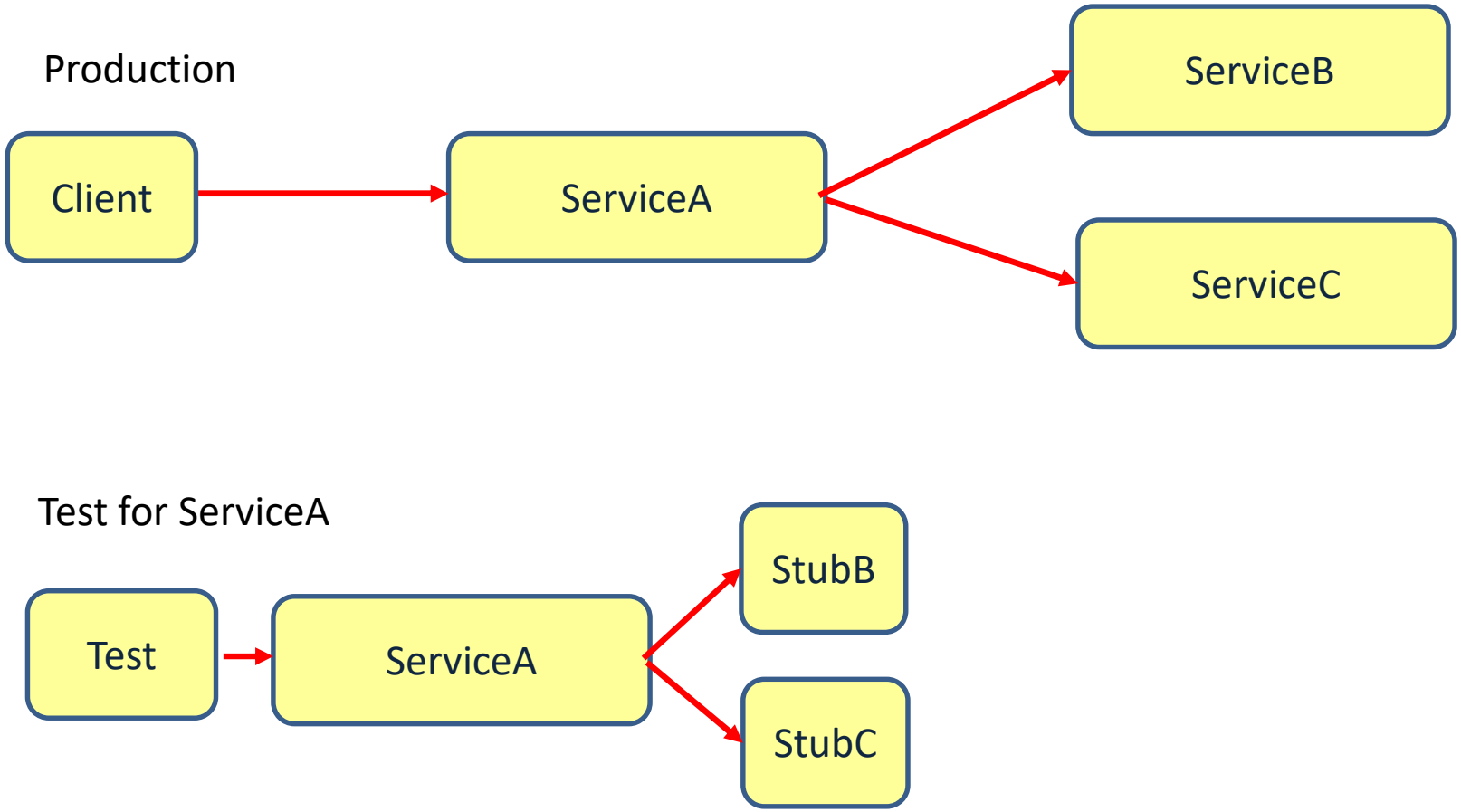


Challenges of a microservice architecture

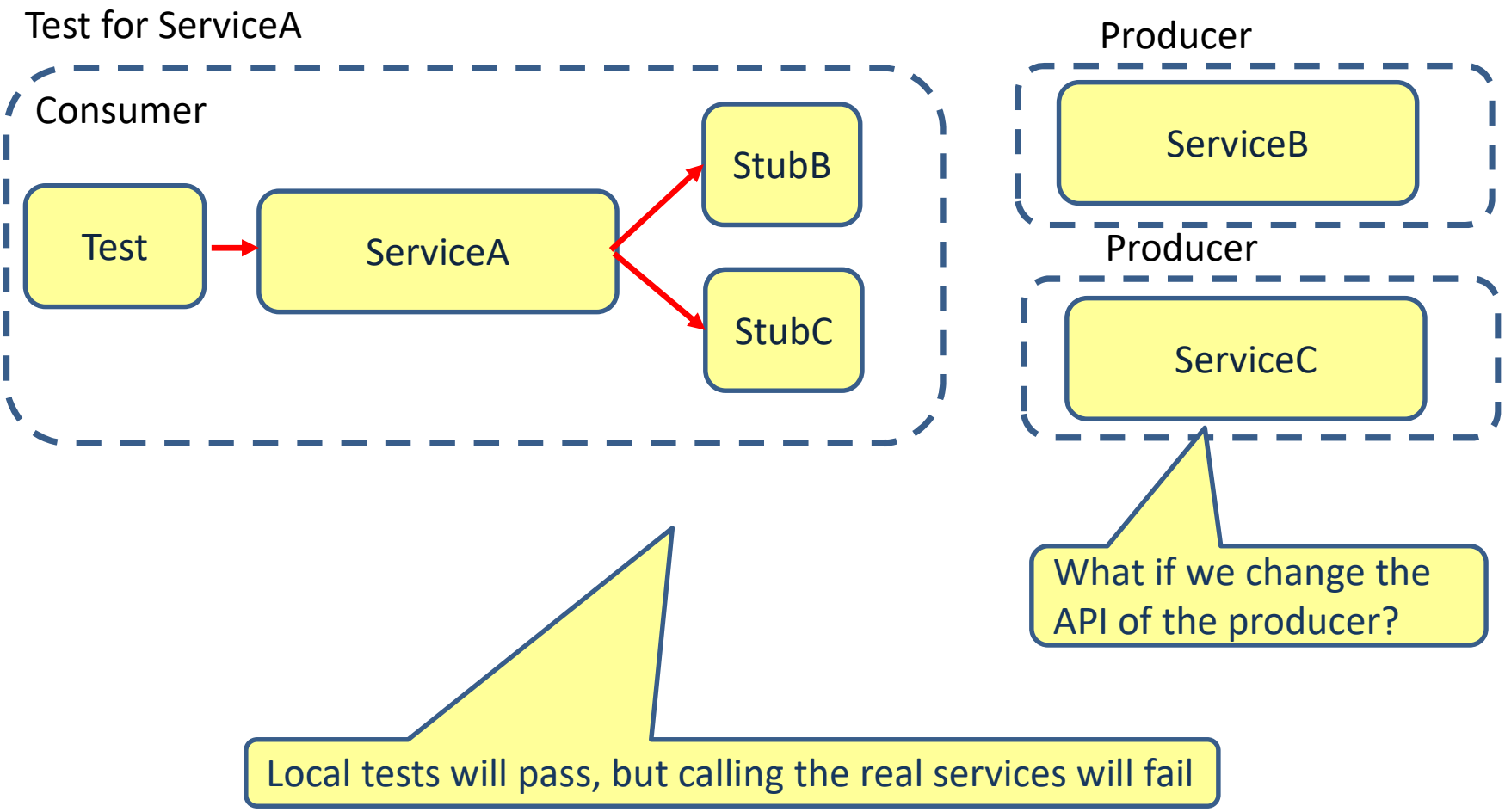
Challenge	Solution
Complex communication	Feign Registry API gateway
Performance	
Resilience	Registry replicas Load balancing between multiple service instances Circuit breaker
Security	
Transactions	
Following the process	
Keep data in sync	
Keep interfaces in sync	
Keep configuration in sync	
Monitor health of microservices	
Follow/monitor business processes	Zipkin ELK

SPRING CLOUD CONTRACT

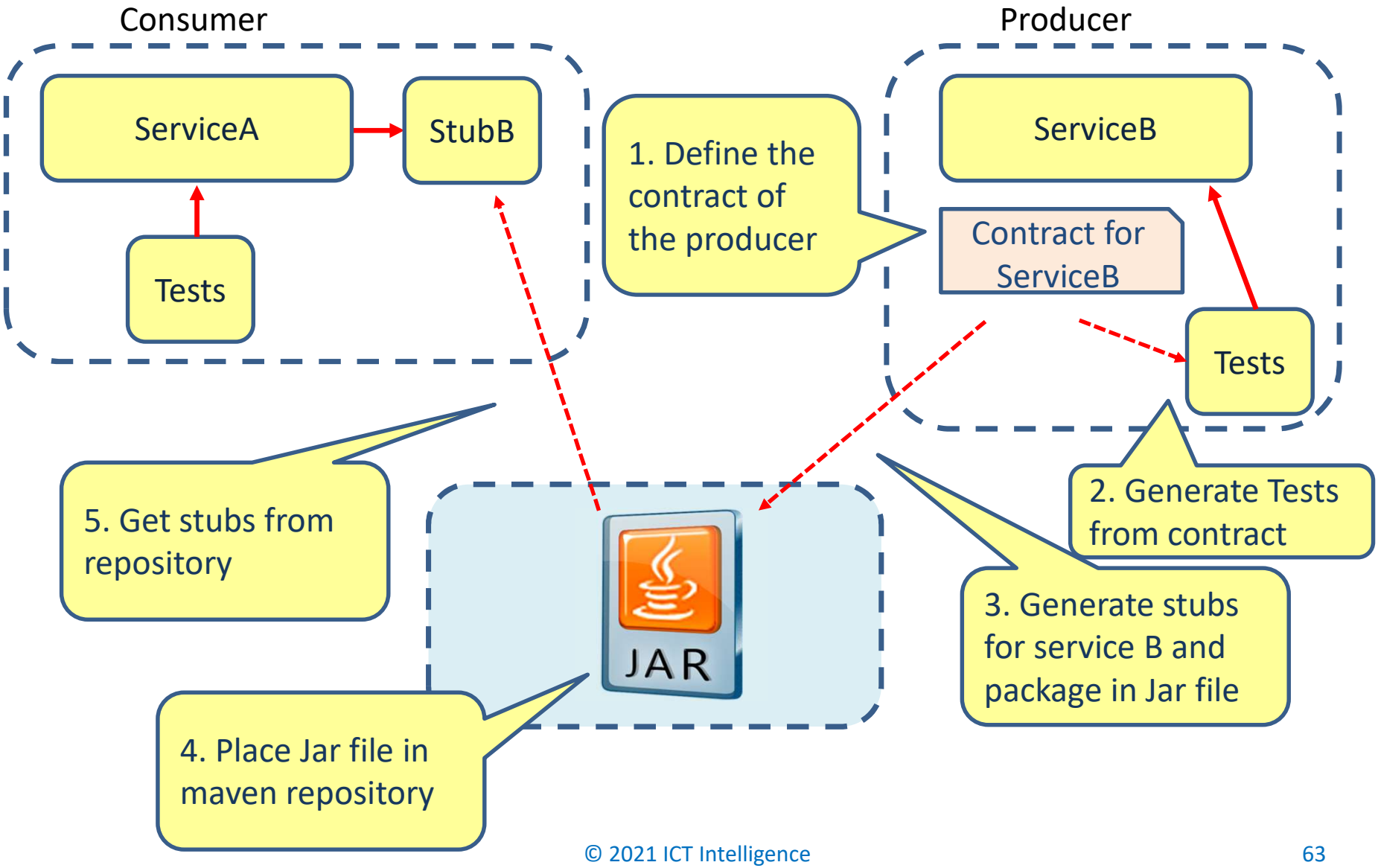
How to test microservices



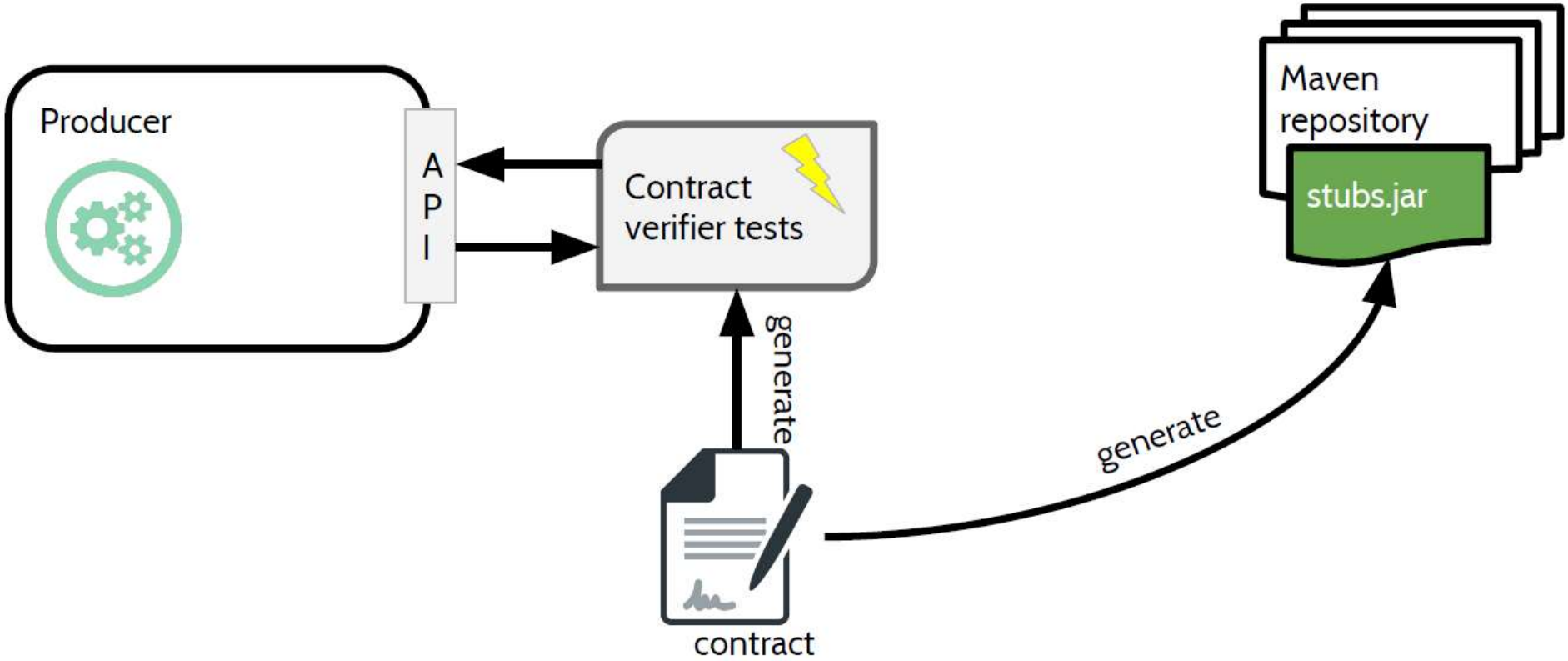
Stubs live at the consumer



Spring cloud contracts



Producer


Producer
A
P
I
Contract
verifier tests
generate
contract
generate
Maven
repository
stubs.jar

Producer maven configuration

```
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-contract-verifier</artifactId>
  <scope>test</scope>
</dependency>


<plugin>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-contract-maven-plugin</artifactId>
  <version>2.2.2.RELEASE</version>
  <extensions>true</extensions>
  <configuration>
    <baseClassForTests>service.BaseTestClass</baseClassForTests>
    <testFramework>JUNIT5</testFramework>
  </configuration>
</plugin>
```

Producer

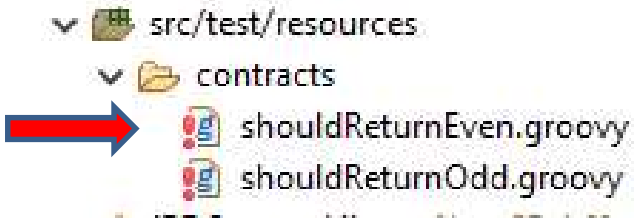
```
@RestController
public class EvenOddController {

    @GetMapping("/validate")
    public String evenOrOdd(@RequestParam("number") Integer number) {
        return number % 2 == 0 ? "Even" : "Odd";
    }
}
```

```
@SpringBootApplication
public class EvenoddServiceApplication {

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

Producer contract 1

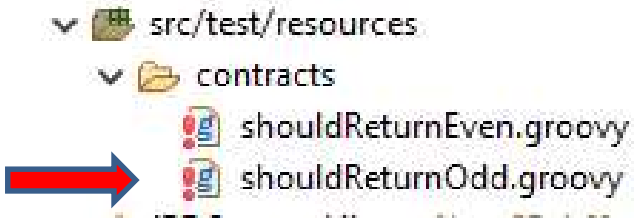
```
import org.springframework.cloud.contract.spec.Contract

Contract.make {
  description "should return even when number input is even"
  request{
    method GET()
    url("/validate") {
      queryParameters {
        parameter("number", "2")
      }
    }
  }
  response {
    body("Even")
    status 200
  }
}
```

Contract in groovy



Producer contract 2

```
import org.springframework.cloud.contract.spec.Contract

Contract.make {
  description "should return odd when number input is odd"
  request {
    method GET()
    url("/validate") {
      queryParameters {
        parameter("number", "1")
      }
    }
  }
  response {
    body("Odd")
    status 200
  }
}
```

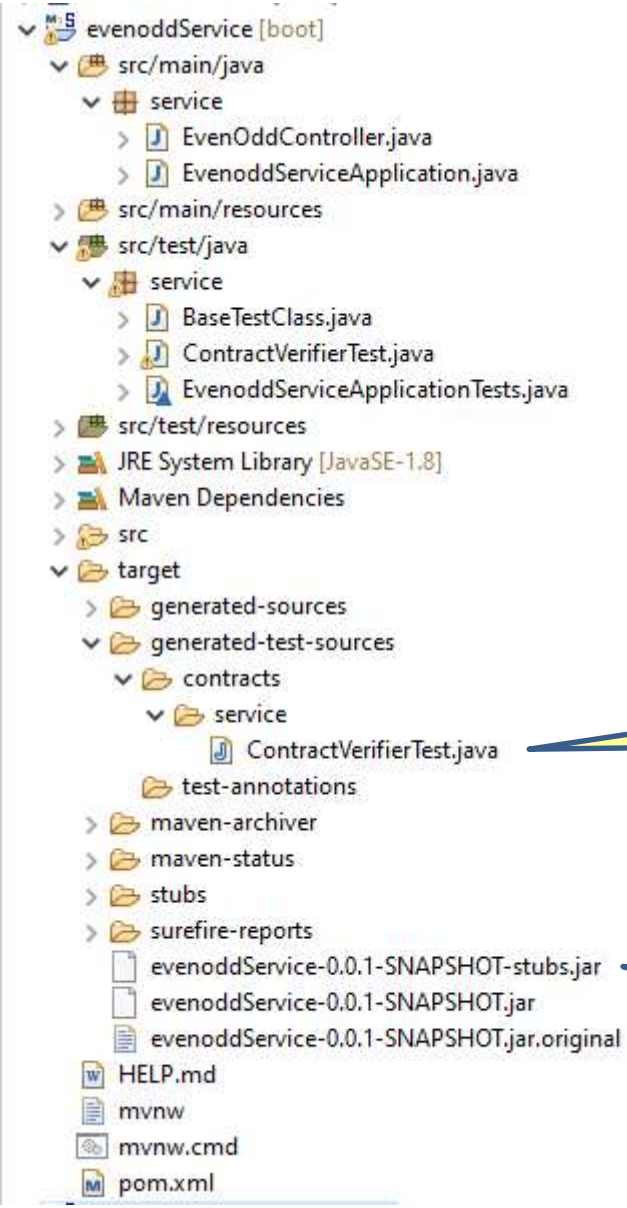


Producer: base test class

```
@SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.MOCK)
@DirtiesContext
@AutoConfigureMessageVerifier
public class BaseTestClass {

    @Autowired
    private EvenOddController evenOddController;

    @BeforeEach
    public void setup() {
        StandaloneMockMvcBuilder standaloneMockMvcBuilder
          = MockMvcBuilders.standaloneSetup(evenOddController);
        RestAssuredMockMvc.standaloneSetup(standaloneMockMvcBuilder);
    }
}
```

This is the base class for all to be generated test classes

After running **maven install**



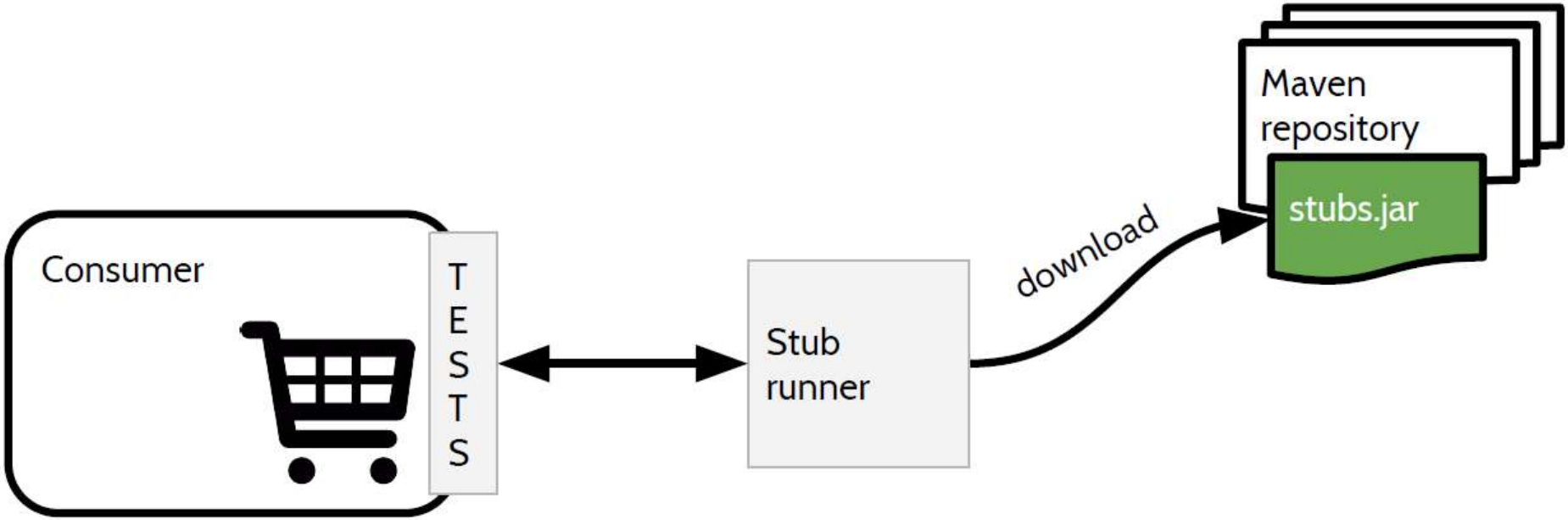
Generated test class based on the contract

Generated stub classes to be used by the consumer. This jar will be placed in the local maven repository

Spring cloud contract DSL

```
import org.springframework.cloud.contract.spec.Contract
Contract.make {
  description("GET employee with id=1")
  request {
    method 'GET'
    url '/employee/1'
  }
 response {
    status 200
    body("""
    {
    "id": "1",
    "fname": "Jane",
    "lname": "Doe",
    "salary": "123000.00",
    "gender": "M"
    }
    """)
    headers {
      contentType(applicationJson())
    }
  }
}
```

Consumer




```
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-contract-stub-runner</artifactId>
  <version>2.2.2.RELEASE</version>
  <scope>test</scope>
</dependency>
```

Consumer

```
@RestController
public class MathController {

    @Autowired
    private RestTemplate restTemplate;

    @GetMapping("/calculate")
    public String checkOddAndEven(@RequestParam("number") Integer number) {
        HttpHeaders httpHeaders = new HttpHeaders();
        httpHeaders.add("Content-Type", "application/json");

        ResponseEntity<String> responseEntity = restTemplate.exchange(
          "http://localhost:8090/validate?number=" + number,
          HttpMethod.GET,
          new HttpEntity<>(httpHeaders),
          String.class);

        return responseEntity.getBody();
    }
}
```

Consumer

```
@SpringBootApplication
public class MathServiceApplication {

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

  @Bean
  RestTemplate restTemplate() {
    return new RestTemplate();
  }
}
```

Consumer test

Get the stubs from the local repository

```
@SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.MOCK)
@AutoConfigureMockMvc
@AutoConfigureJsonTesters
@AutoConfigureStubRunner(stubsMode = StubRunnerProperties.StubsMode.LOCAL,
        ids = "com.acme:evenoddService:+:stubs:8090")
public class MathControllerIntegrationTest {

    @Autowired
    private MockMvc mockMvc;

    @Test
    public void given_WhenPassEvenNumberInQueryParam_ThenReturnEven() throws Exception {
        mockMvc.perform(MockMvcRequestBuilders.get("/calculate?number=2")
          .contentType(MediaType.APPLICATION_JSON))
          .andExpect(status().isOk())
          .andExpect(content().string("Even"));
    }

    @Test
    public void given_WhenPassOddNumberInQueryParam_ThenReturnOdd() throws Exception {
        mockMvc.perform(MockMvcRequestBuilders.get("/calculate?number=1")
          .contentType(MediaType.APPLICATION_JSON))
          .andExpect(status().isOk())
          .andExpect(content().string("Odd"));
    }
}
```

Consumer test

```
@SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.MOCK)
@AutoConfigureMockMvc
@AutoConfigureJsonTesters
@AutoConfigureStubRunner(stubsMode = StubRunnerProperties.StubsMode.LOCAL,
        ids = "com.acme:evenoddService:+:stubs:8090")
public class MathControllerIntegrationTest {
```

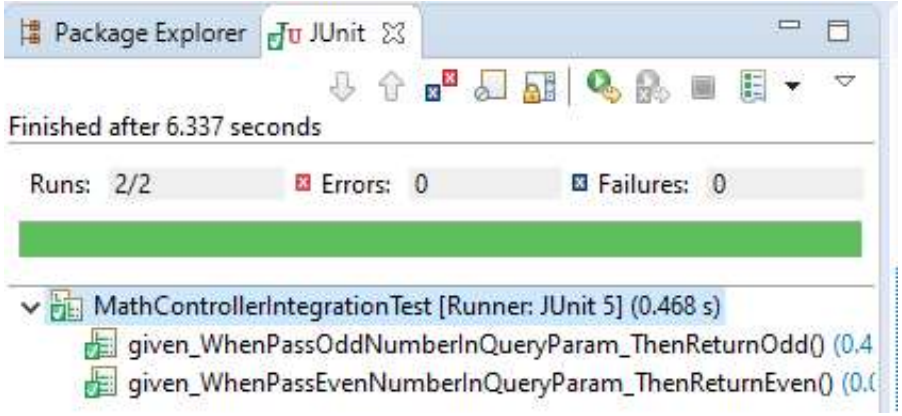
Group id

Artifact id

Version + means latest version

stubs

Port number to run the stubs on



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	
Following the process	
Keep data in sync	
Keep interfaces in sync	Spring cloud contract
Keep configuration in sync	
Monitor health of microservices	
Follow/monitor business processes	Zipkin ELK