# 微服务下的契约管理

祁兮
Technical Principal
ThoughtWorks 思特沃克
May 8, 2018

ORACLE CODE
developer.oracle.com

Live for the Code

ORACLE

# 目录

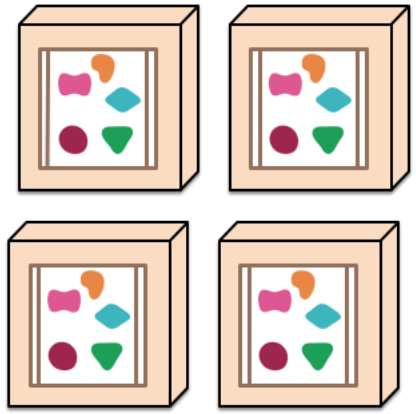
- 微服务架构
- 契约测试
- 契约管理

# 微服务架构

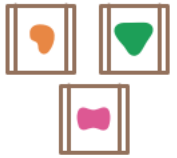
# 微服务架构

*A monolithic application puts all its functionality into a single process...*

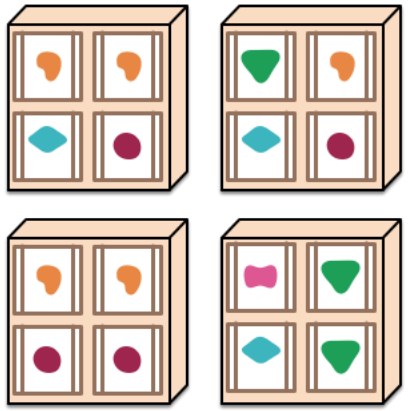
*... and scales by replicating the monolith on multiple servers*



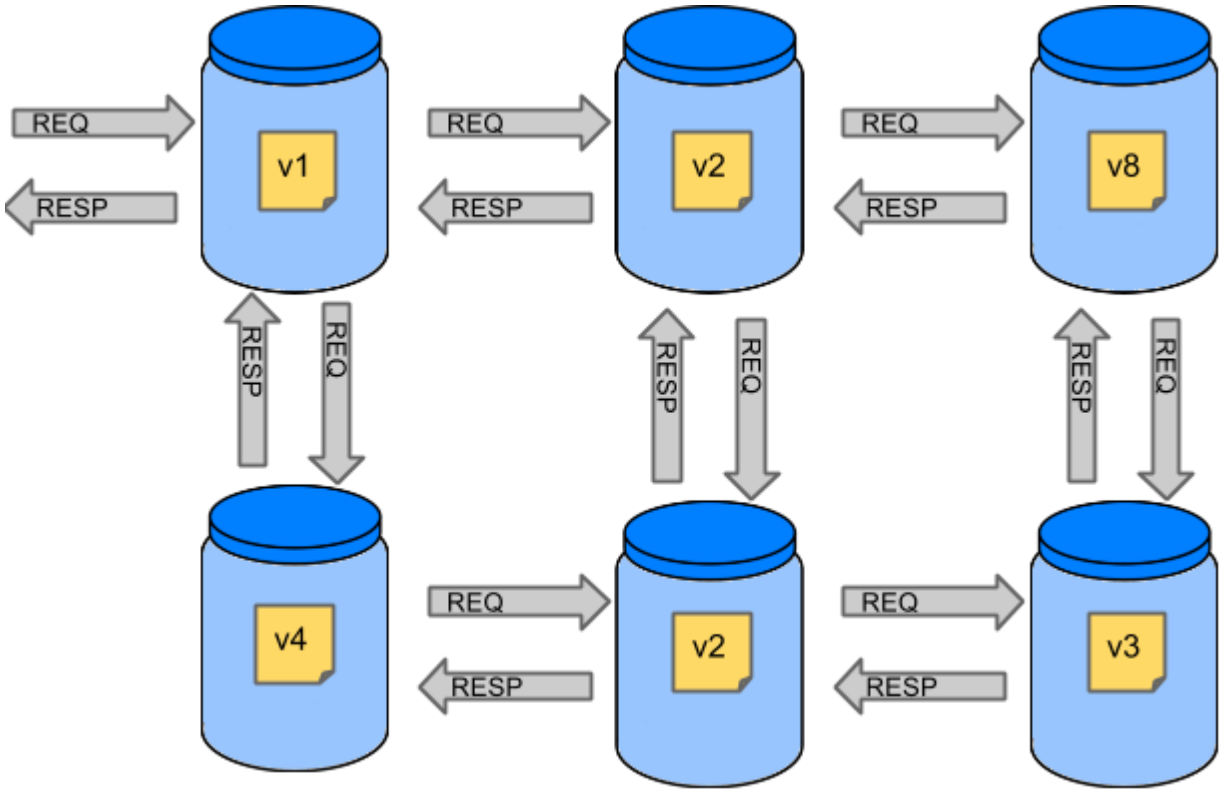
*A microservices architecture puts each element of functionality into a separate service...*



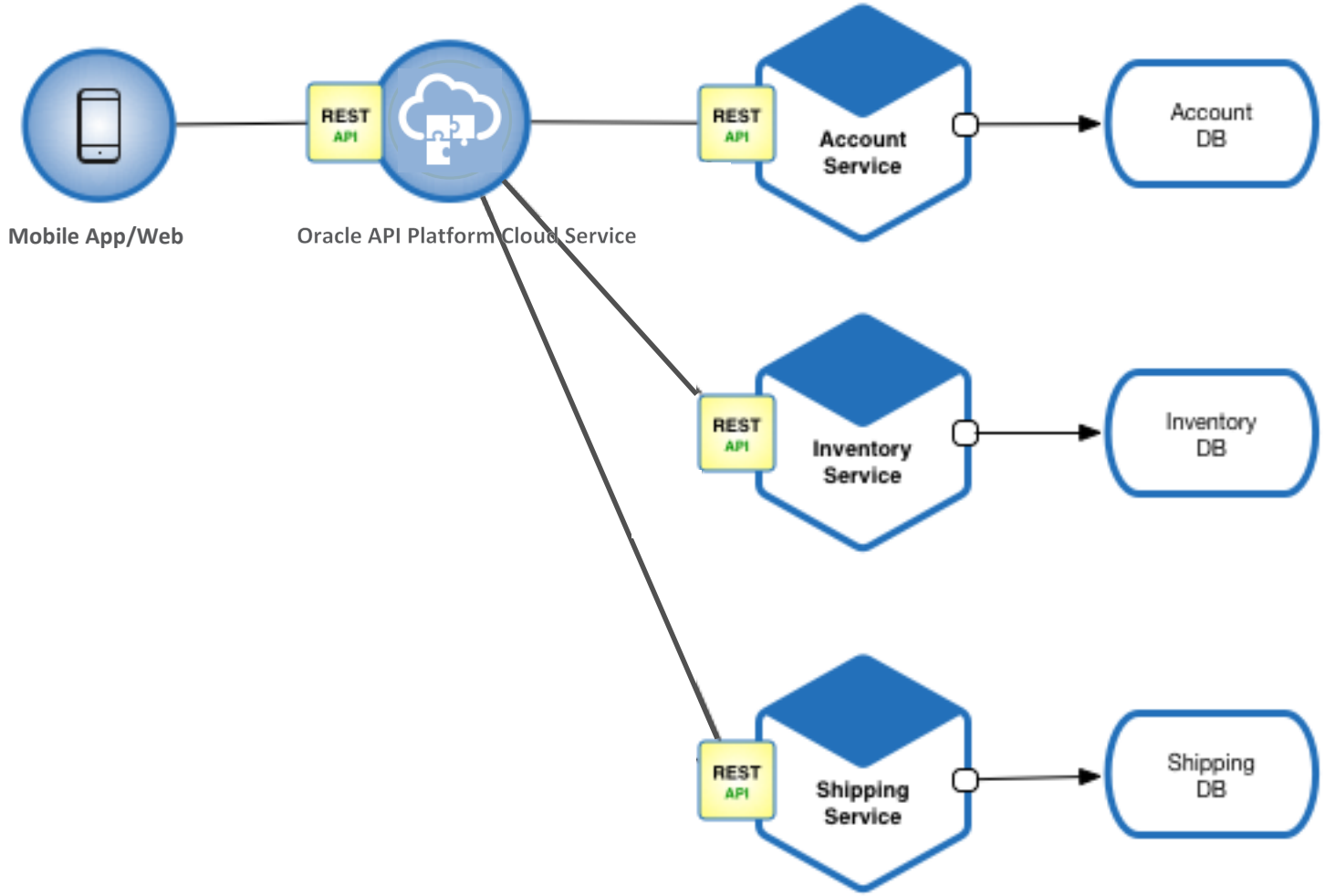
*... and scales by distributing these services across servers, replicating as needed.*



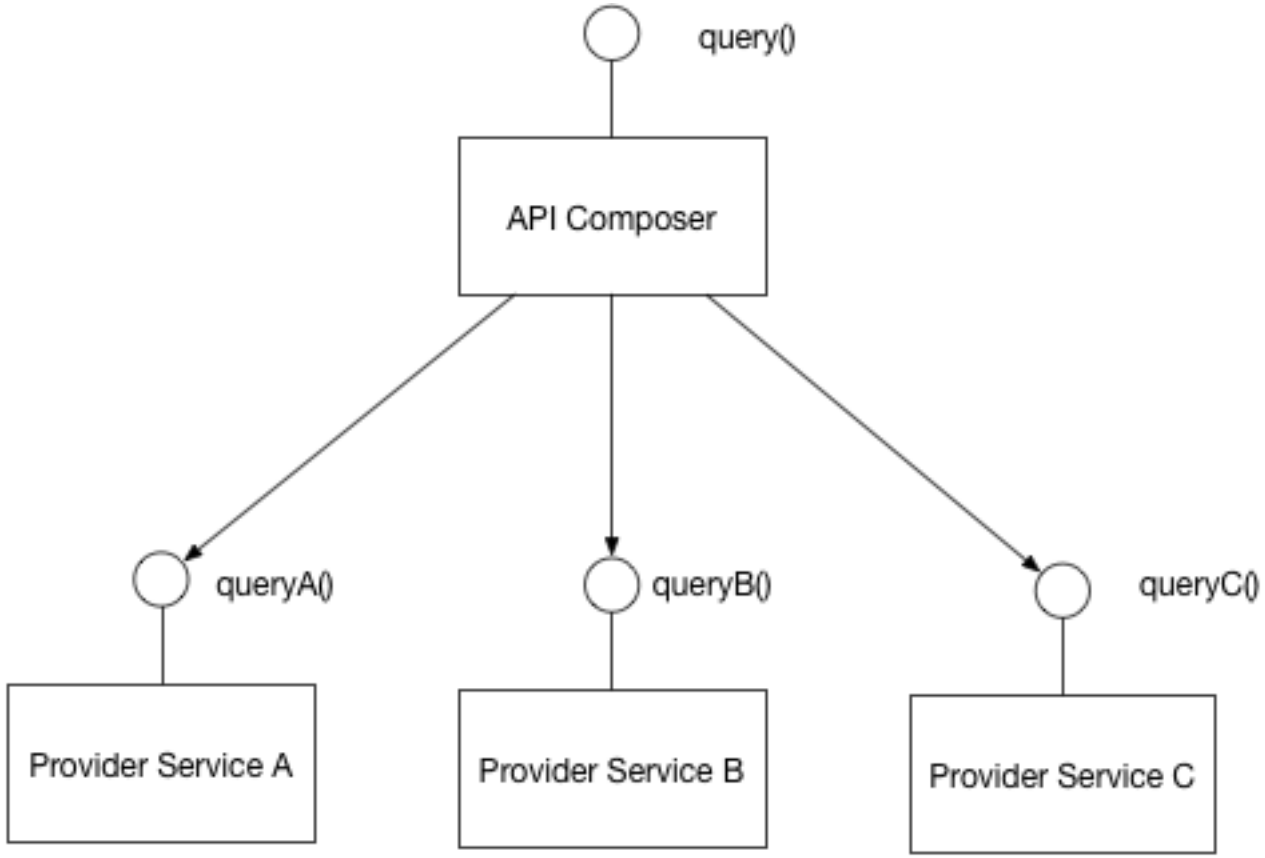
# 服务间调用



# 包含前端的微服务架构

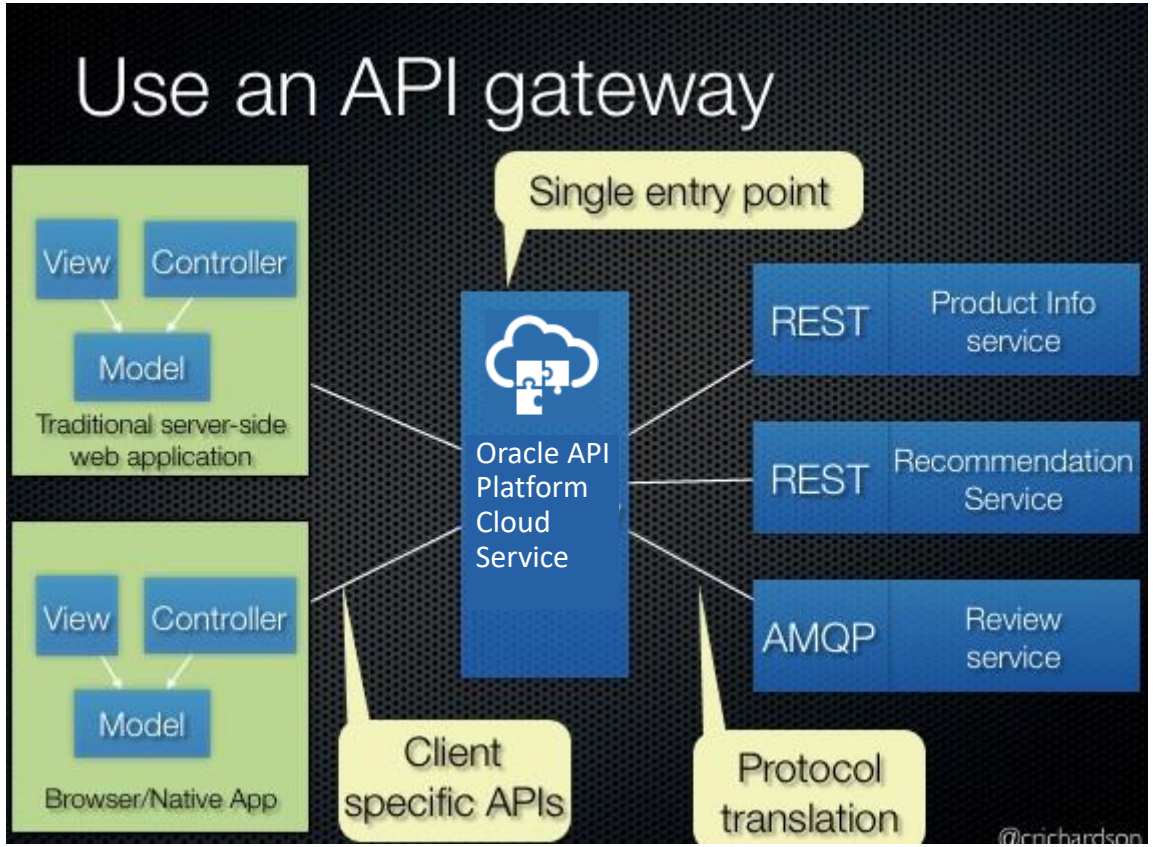


# API Composition

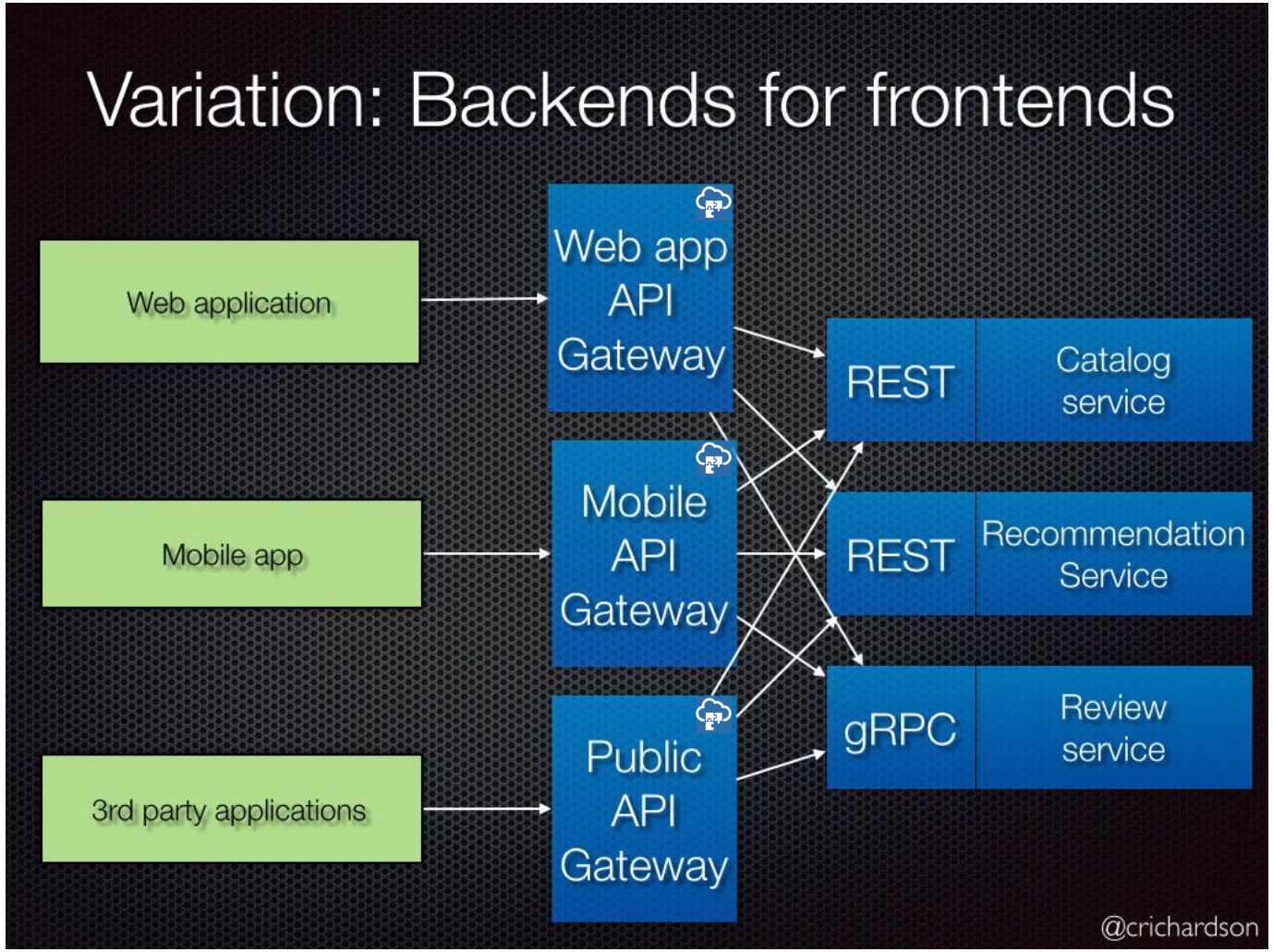


# 多个前端

## 不同的前端需要不同的接口



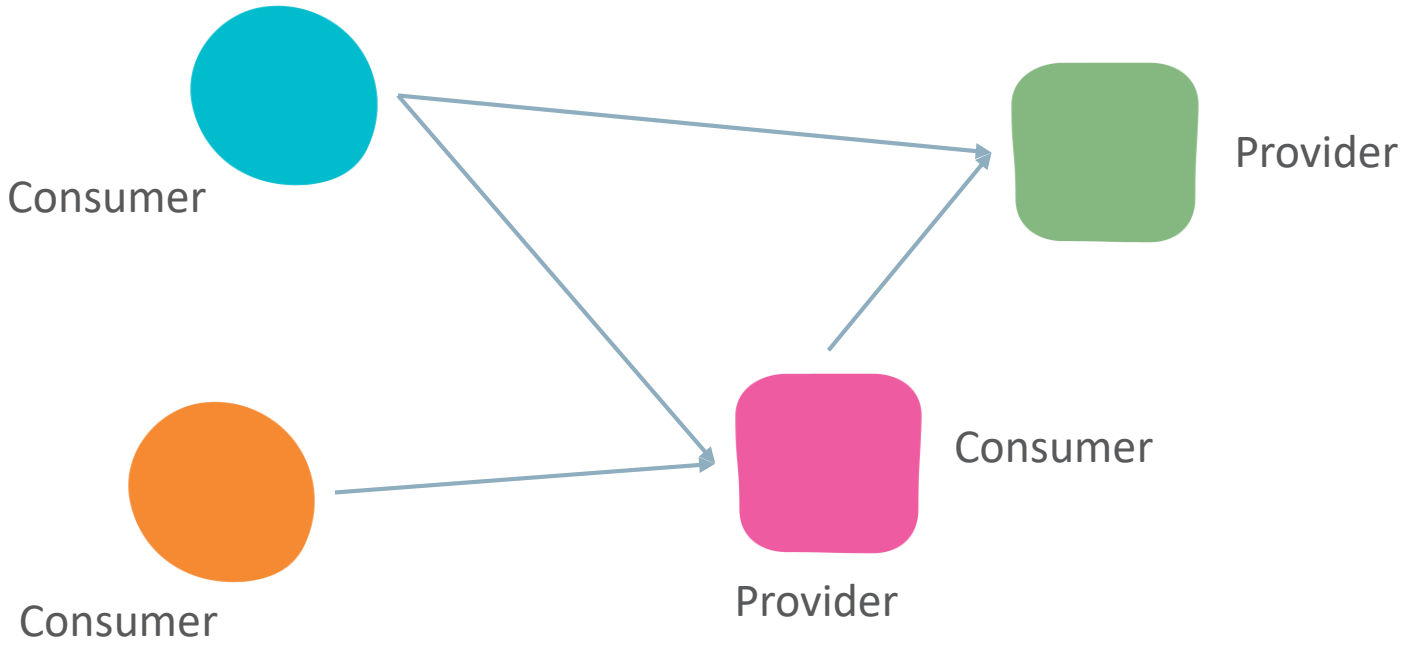
# Backend for Front-End



# 契约测试

# 接口管理

## 复杂的调用关系



# 接口管理怎么做？

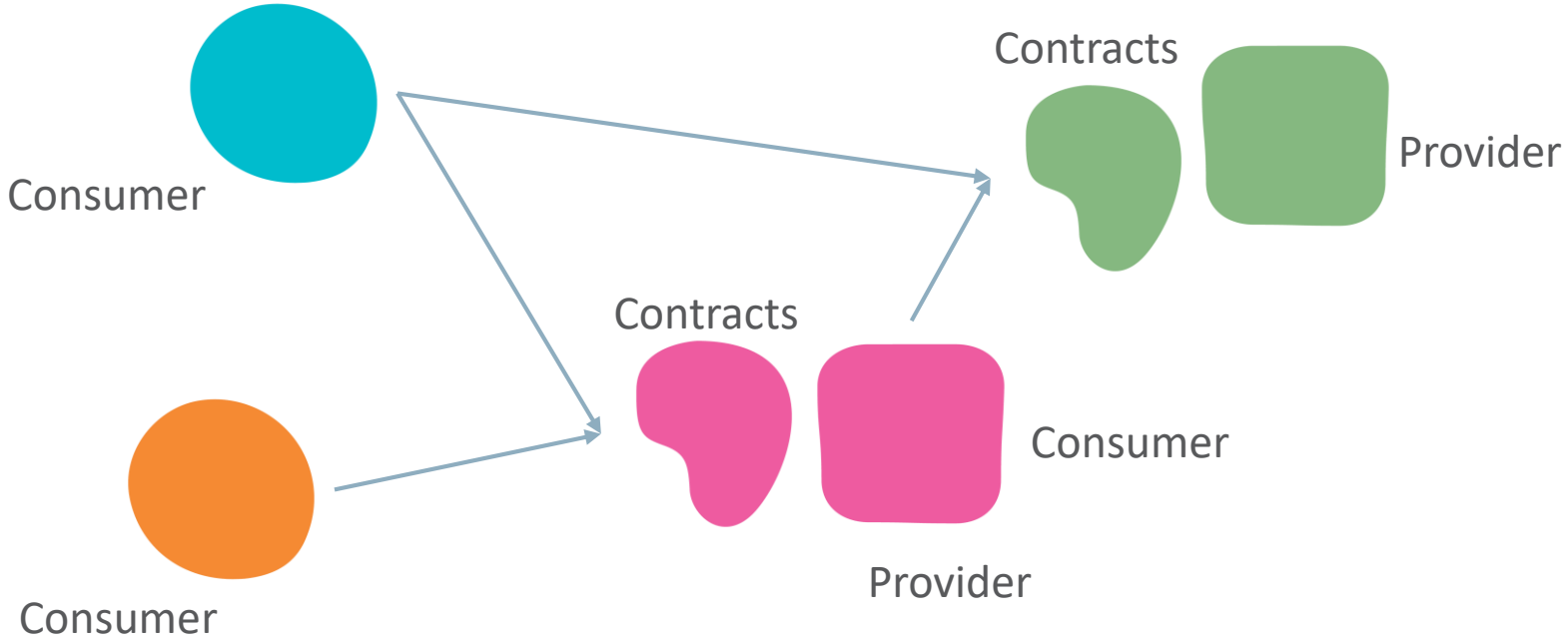
- 越来越多的服务，越来越多的接口，越来越多的调用
- 一个接口变化了，哪些功能会被影响、需要测试？
- 生产环境和开发环境接口有哪些差别？微服务可以独立交付么？

# 前后端分离带来的问题

- 由于前端组件依赖于后端提供的接口，在开发过程中前端需要等待后端接口开发完成才能开始开发。
- 而在项目中我们希望一个功能从开发开始到开发结束的时间尽可能短，所以通常都是前端和后端的同时开始开发这个功能。
- 那么问题来了：在后端接口开发完成前，前端怎么才能正常开发，不被阻塞呢？

# 引入契约

在开卡时前后端会一起约定后端需要提供哪几个接口，每个接口是什么样子。有了这些约定，也就是契约，前后端就可以各自开发，然后集成。
把前端看作**Consumer**，后端看作**Provider**。契约适用于所有**Consumer**和**Provider**集成的情况



# 集成测试的问题

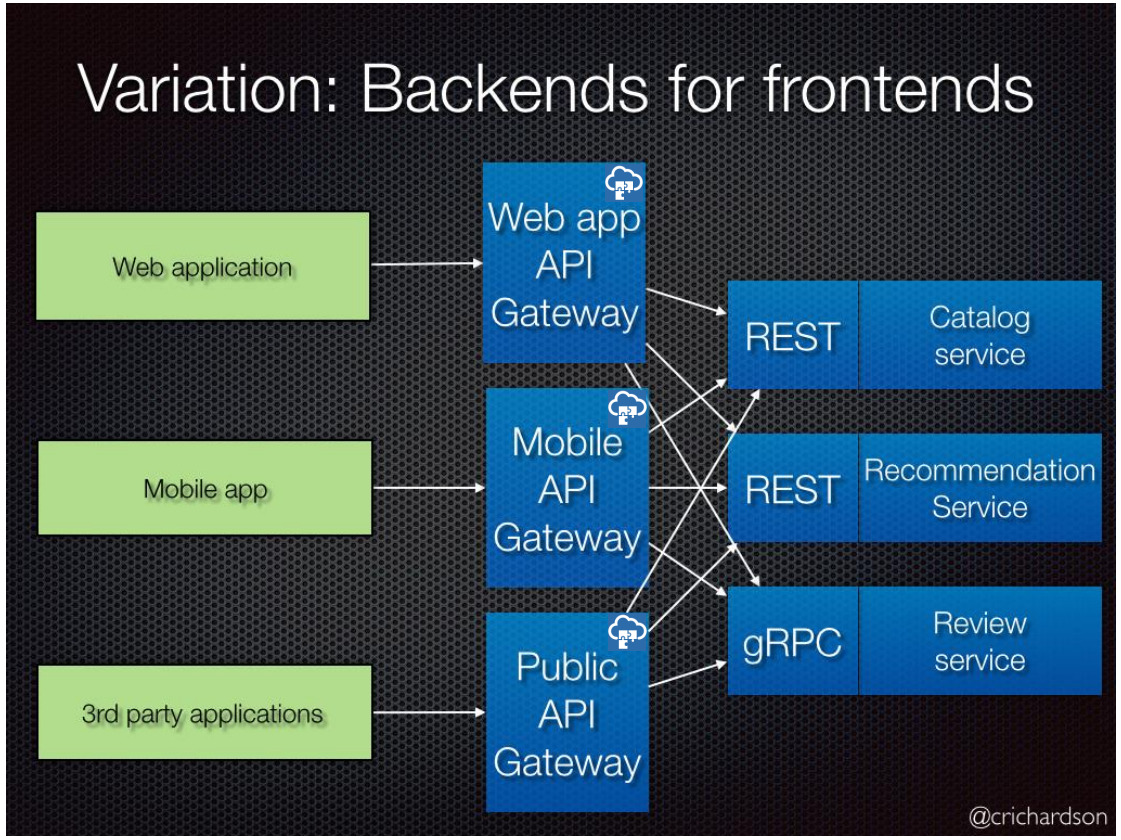
- 随着业务的变化，接口也会发生变化，而从可能会破坏Consumer功能。
- 应该有某种测试覆盖这个场景，当有功能被破坏时，这种测试就会失败，从而驱动我们修复功能。
- 这个满足我们需求的测试是什么样的？测试的SUT（System Under Test）是什么？
- 这个测试既要覆盖Provider接口，也要覆盖Consumer代码。

# 集成测试的问题

- E2E测试是否可以满足我们的需求？

# E2E测试?

## system test/end-to-end test?



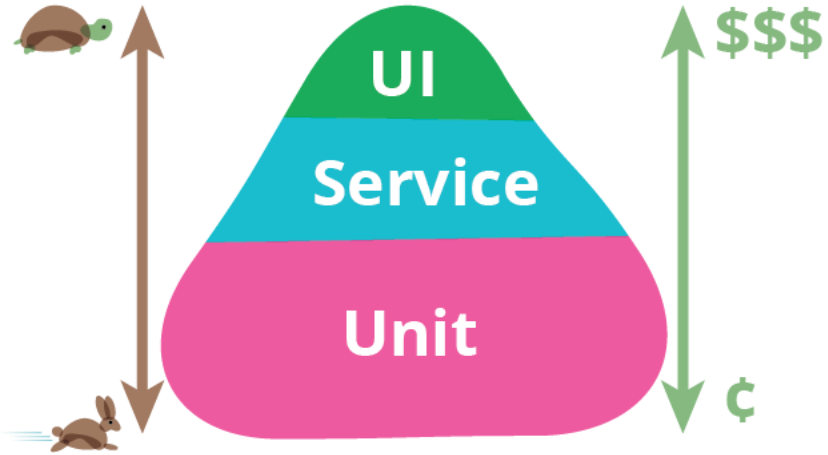
# Integrated Tests Are A Scam

You write integrated tests because you can’t write perfect unit tests.

-- J.B. Rainsberger

# 测试金字塔

**If you get a failure in a high level test, not just do you have a bug in your functional code, you also have a missing or incorrect unit test.**



# Narrow Integration Tests

- exercise only that portion of the code in my service that talks to a separate service
- uses test doubles of those services, either in process or remote
- thus consist of many narrowly scoped tests, often no larger in scope than a unit test (and usually run with the same test framework that's used for unit tests)

# 引入契约测试

- Consumer契约测试用来保证Consumer功能不因契约变化而被破坏
- Provider契约测试用来保证Provider代码是符合契约的
- 为了方便Consumer开发，契约可以运行成Mock Server
- 契约使用DSL定义

# Consumer契约测试

## Spring Cloud Contract

```
org.springframework.cloud.contract.spec.Contract.make {
        request {
                method 'PUT'
                url '/fraudcheck'
                body([ "clientId": $(regex('[0-9]{10}')), loanAmount: 99999] )
                headers { contentType('application/json') }
        }
        response {
                status OK()
                body([ fraudCheckStatus: "FRAUD", "rejectionReason": "Amount too high" ] )
                headers { contentType('application/json') }
        }
}
```

# Consumer契约测试

## Spring Cloud Contract

```
@Test
public void shouldBeRejectedDueToAbnormalLoanAmount() {
        LoanApplication application = new LoanApplication(new Client("1234567890"),
99999);
        LoanApplicationResult loanApplication = service.loanApplication(application);
        assertThat(loanApplication.getLoanApplicationStatus()).isEqualTo(LoanApplicationS
tatus.LOAN_APPLICATION_REJECTED);
        assertThat(loanApplication.getRejectionReason()).isEqualTo("Amount too high");
}


ResponseEntity<FraudServiceResponse> response =
restTemplate.exchange("http://localhost:" + port + "/fraudcheck", HttpMethod.PUT, new
HttpEntity<>(request, httpHeaders), FraudServiceResponse.class);
```

# Consumer契约测试

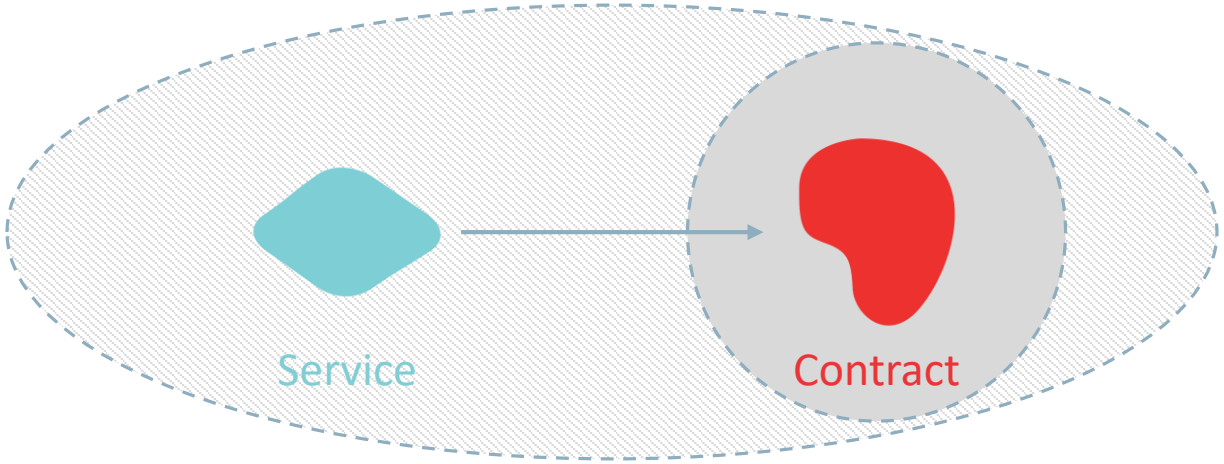
## Mest

```
let mest = new Mest()

let results: IDiff = mest.localCompareInterface('FraudCheckResponse.ts', {
  status: FRAUD,
  rejectionReason: 'Amount too high'
})

expect(results).toEqual({
  diff: {
    local: ['fraudCheckStatus'],
    remote: ['status']
  }
})
```

# Consumer契约测试的SUT




Mest SUT

Spring Cloud Contract SUT

# Mock Server

## Spring Cloud Contract

```
@SpringBootApplication
@AutoConfigureWireMock
public class ContractsApplication {
}
```

```
wiremock:
  server:
    files:
    stubs:
      - file:build/stubs/**/mappings/**/*.json
    port: 6565
```

# Provider契约测试

## Spring Cloud Contract

```
public class FraudBase {
     @Before
     public void setup() {
          RestAssuredMockMvc.standaloneSetup(new
FraudCheckController(stubbedService()));
     }
     private FraudCheckService stubbedService() {
     }
```

# Provider契约测试

## Spring Cloud Contract生成的测试代码

```
@Test
public void validate_shouldMarkClientAsFraud() throws Exception {
        MockMvcRequestSpecification request = given()
                .header("Content-Type", "application/vnd.fraud.v1+json")
                .body("{\"clientId\":\"1234567890\",\"loanAmount\":99999}");
        ResponseOptions response = given().spec(request)
                .put("/fraudcheck");
        assertThat(response.statusCode()).isEqualTo(200);
        assertThat(response.header("Content-Type")).matches("application/vnd.fraud.v1.json.*");
        DocumentContext parsedJson = JsonPath.parse(response.getBody().asString());
        assertThatJson(parsedJson).field("['fraudCheckStatus']").matches("[A-Z]{5}");
        assertThatJson(parsedJson).field("['rejectionReason']").isEqualTo("Amount too high");
}
```

# Provider契约测试

## Moscow

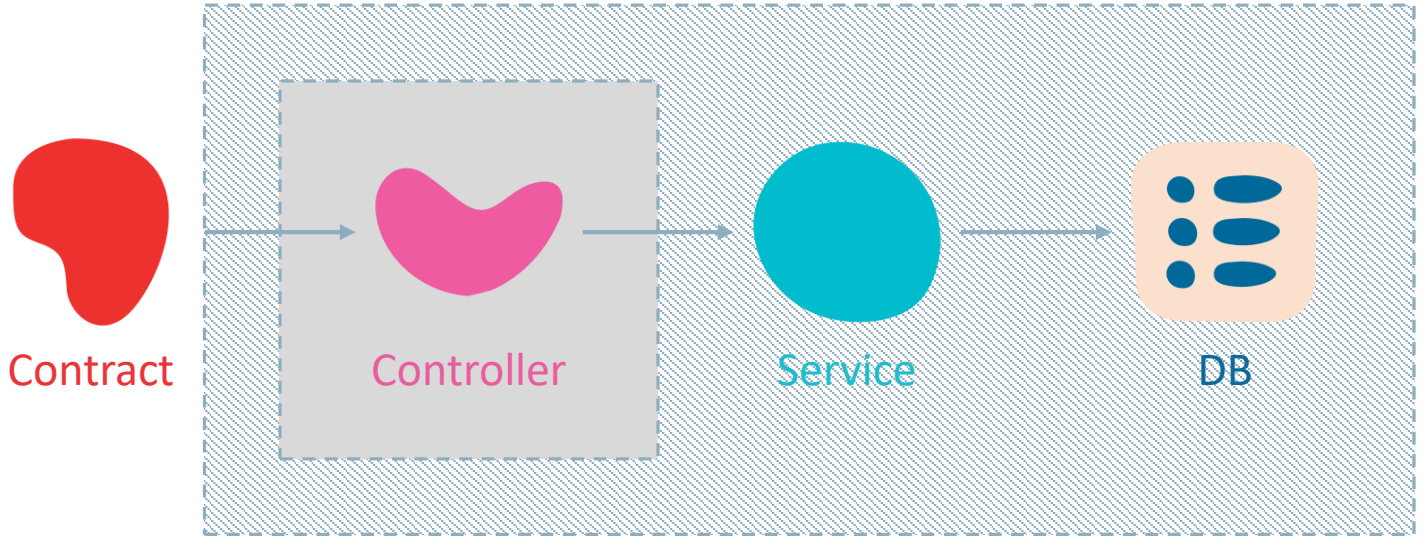
```
{
    "description": "should_mark_client_as_fraud",
    "request": {
        "method": "PUT",
        "uri": "/fraudcheck",
        "json": { "clientId": "1234567890", "loanAmount": 99999 }
    },
    "response": {
        "status": 200,
        "headers": { "content-type": "application/json" },
        "json": { "fraudCheckStatus": "FRAUD", "rejectionRreason": "Amount too high" }
    }
}
```

# Provider契约测试

**Moscow**

```
@RunWith(SpringJUnit4ClassRunner.class)
@SpringBootTest(webEnvironment = RANDOM_PORT)
public class FraudCheckApiTest extends ApiTestBase {
    @Test
    public void should_mark_client_as_fraud() {
        assertContract();
    }
}
```

# 后端契约测试的SUT




Spring Cloud Contract SUT

Moscow SUT

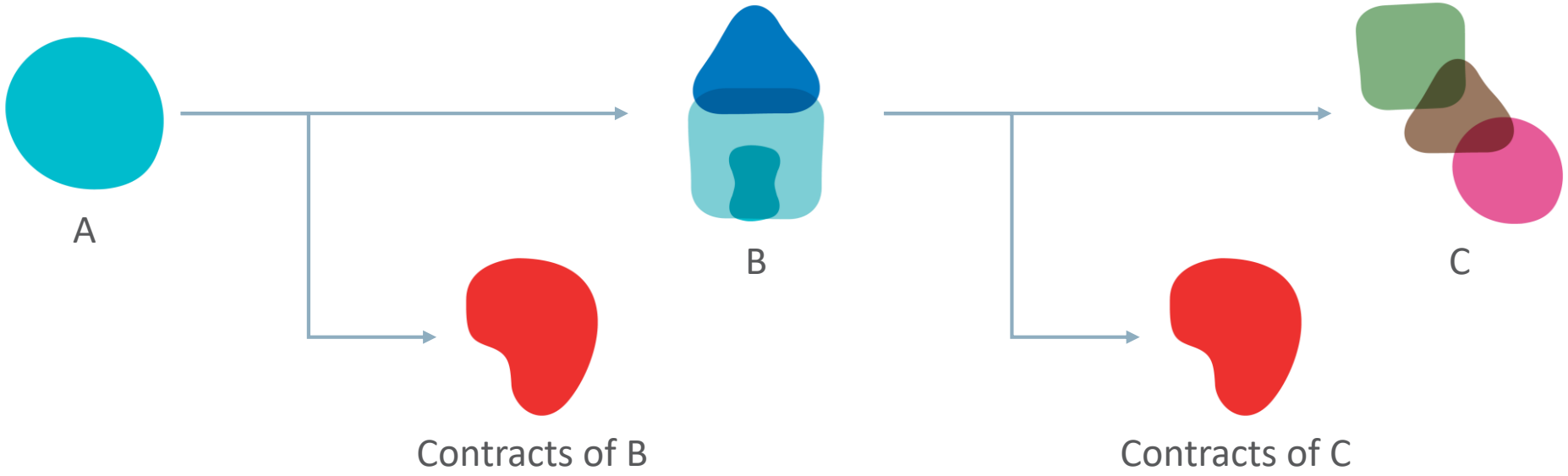
# 附加价值 - TDD

- 你的团队做到测试驱动开发了吗？
- 先写契约，再写实现，实现了（契约）测试驱动（Provider）开发
- 更容易做到的TDD

# 契约管理

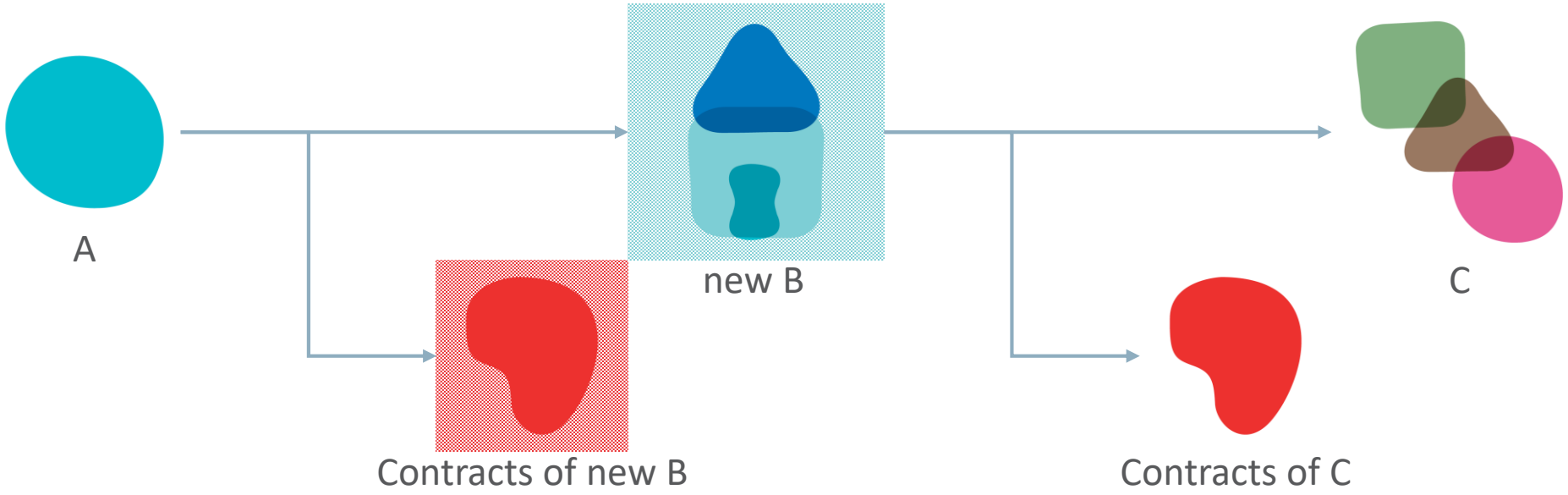
# 契约版本

## 单个服务能否独立交付？

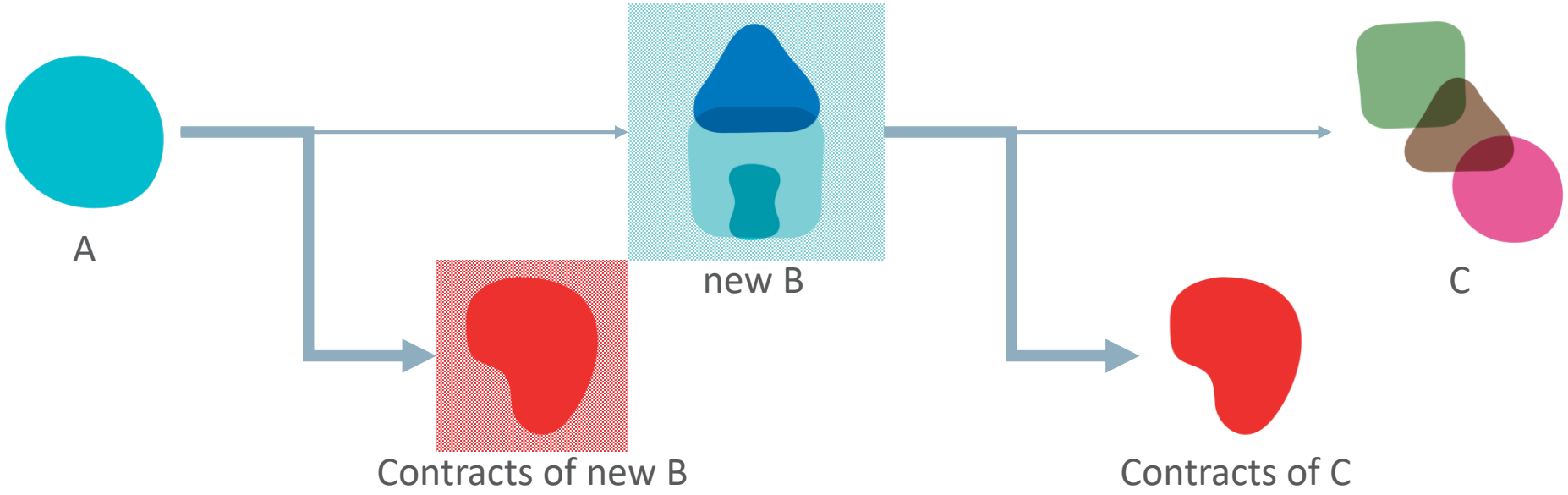


# 契约版本

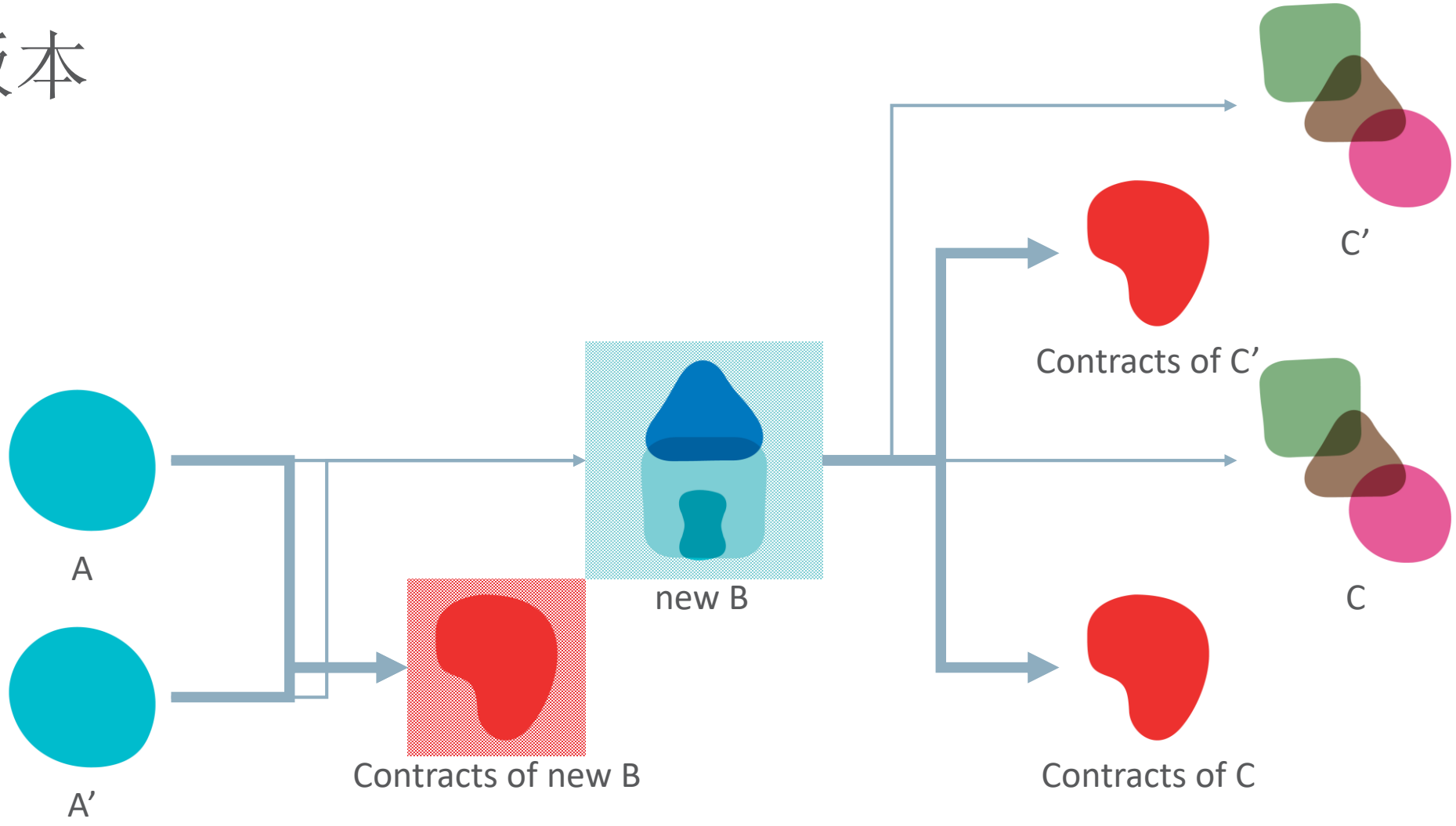
## 升级B服务，B->new B



# 契约版本



# 契约版本



# 契约文档

## Swagger/RAML

- 基于Example的契约，其包含的信息量超过基于Schema的契约
- 可以将基于Example的契约转换成基于Schema的契约
- 可以使用类似Oracle API Catalog工具把Schema转换在线文档，支持在线调用

# 为什么不直接使用Swagger、 RAML做契约测试？

- 缺少测试需要的数据
- 不支持返回不同响应

# Review契约文档

- 契约是接口的描述，接口是服务对外暴露的粗粒度的功能，接口封装了服务能力。
- 通过Review契约文档，可以观察到接口设计以及架构设计。过细粒度的接口定义往往表明服务划分不合理。

# Q & A

# 做好契约管理，你最终将获得什么？

- 不依赖其他服务的，轻量级契约（集成）测试
- 顺畅的前后端开发体验
- 基于契约测试的TDD
- 微服务独立部署交付
- 被自动化测试覆盖的海量微服务接口
- 友好的接口文档网站，帮助架构师Review

# 参考资料

- https://martinfowler.com/articles/microservices.html
- https://martinfowler.com/bliki/IntegrationTest.html
- https://martinfowler.com/bliki/ContractTest.html
- http://microservices.io/patterns/microservices.html
- http://microservices.io/patterns/apigateway.html
- http://blog.thecodewhisperer.com/permalink/integrated-tests-are-a-scam
- http://cloud.spring.io/spring-cloud-static/spring-cloud-contract/2.0.0.RC1/single/spring-cloud-contract.html
- https://github.com/phodal/mest
- https://github.com/macdao/moscow
- [你的微服务敢独立交付么？](https://www.jianshu.com/p/625476437c22)

# Thank You

## 期待为您的创新服务

ThoughtWorks