JavaEE平台技术

JavaEE Platform Technologies

——微服务体系结构

Micro service Architecture

邱明 博士

厦门大学信息学院

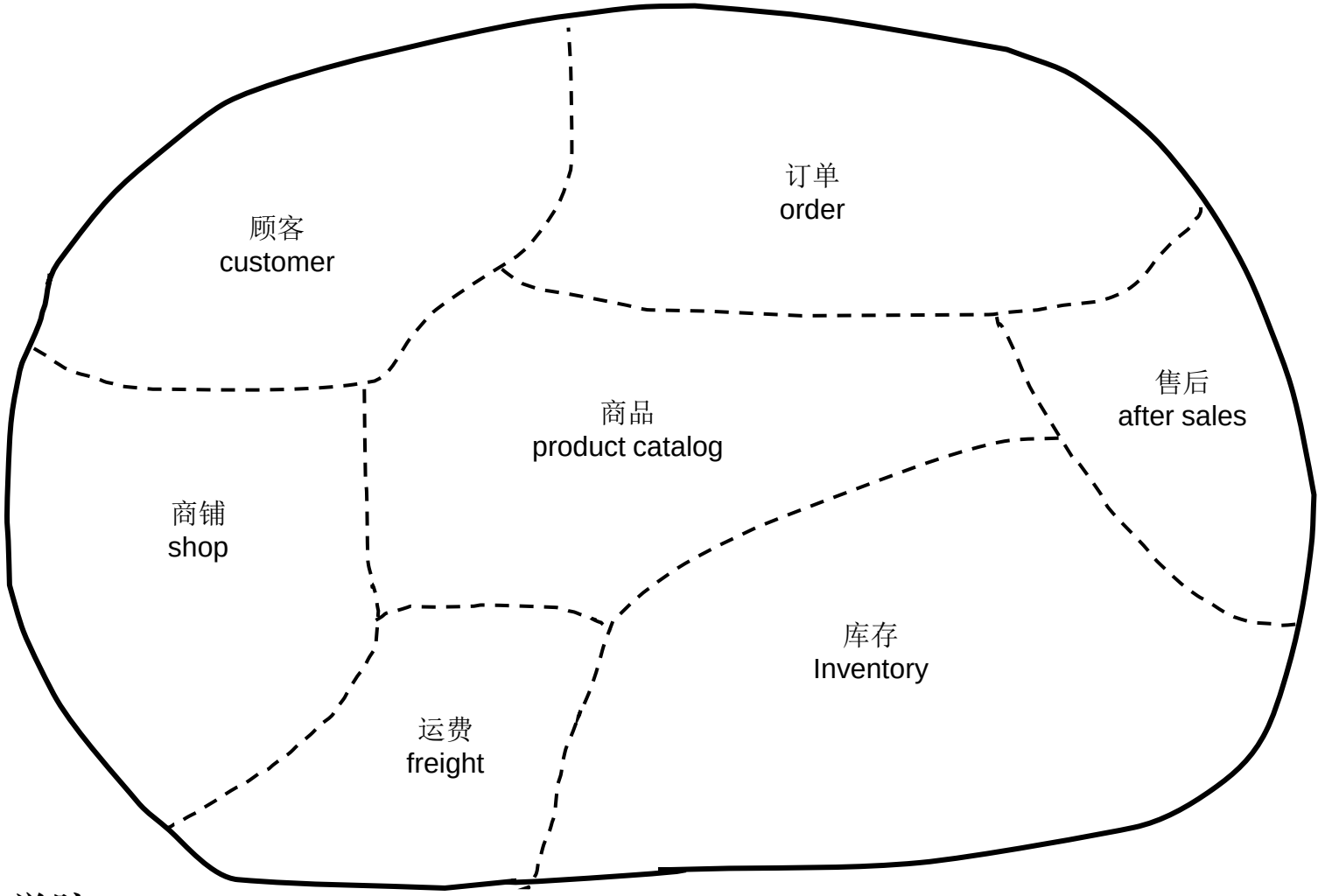
mingqiu@xmu.edu.cn

2023年秋季学期

1.微服务的定义
Microservice Definition

- 微服务是考虑围绕着业务领域组件来创建应用，这些应用可独立地进行开发、管理和加速。
  - 单一职责：每一个微服务应该都是单一职责的，一个微服务解决一个业务问题。应用系统是一组微服务的集合。
  - 松散耦合：服务与服务之间应该采用轻量级的通信（HTTP、REST和JSON）相互连接
  - 独立存在：每一个微服务拥有独立的数据结构和数据库，数据只有自己可以处理。每一个服务可以独立编译打包部署。

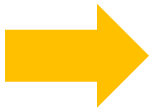
1.微服务的定义
Microservice Definition

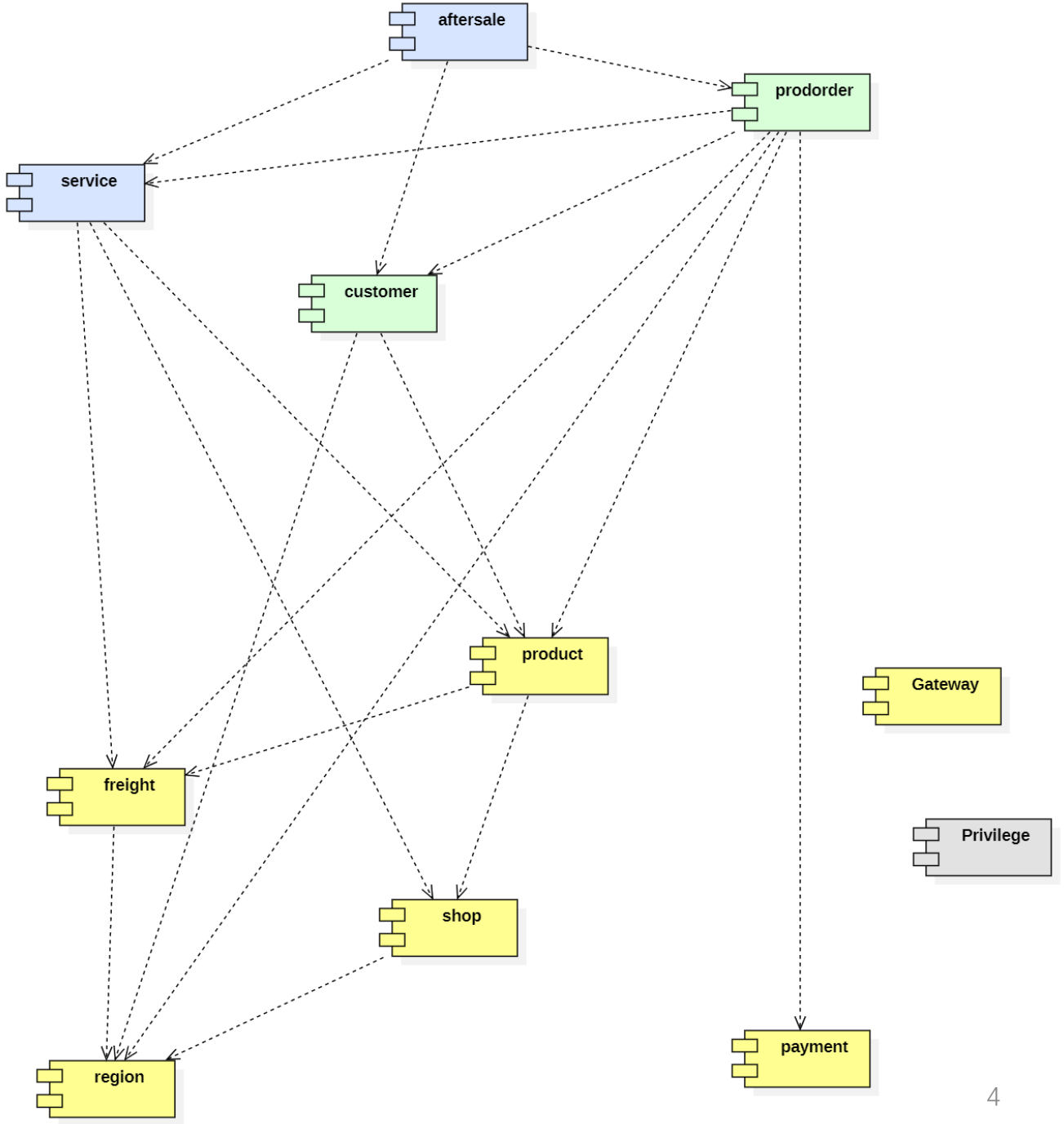


1.微服务的定义
Microservice Definition

- 分解业务问题
- 确定服务的粒度
- 定义服务的接口

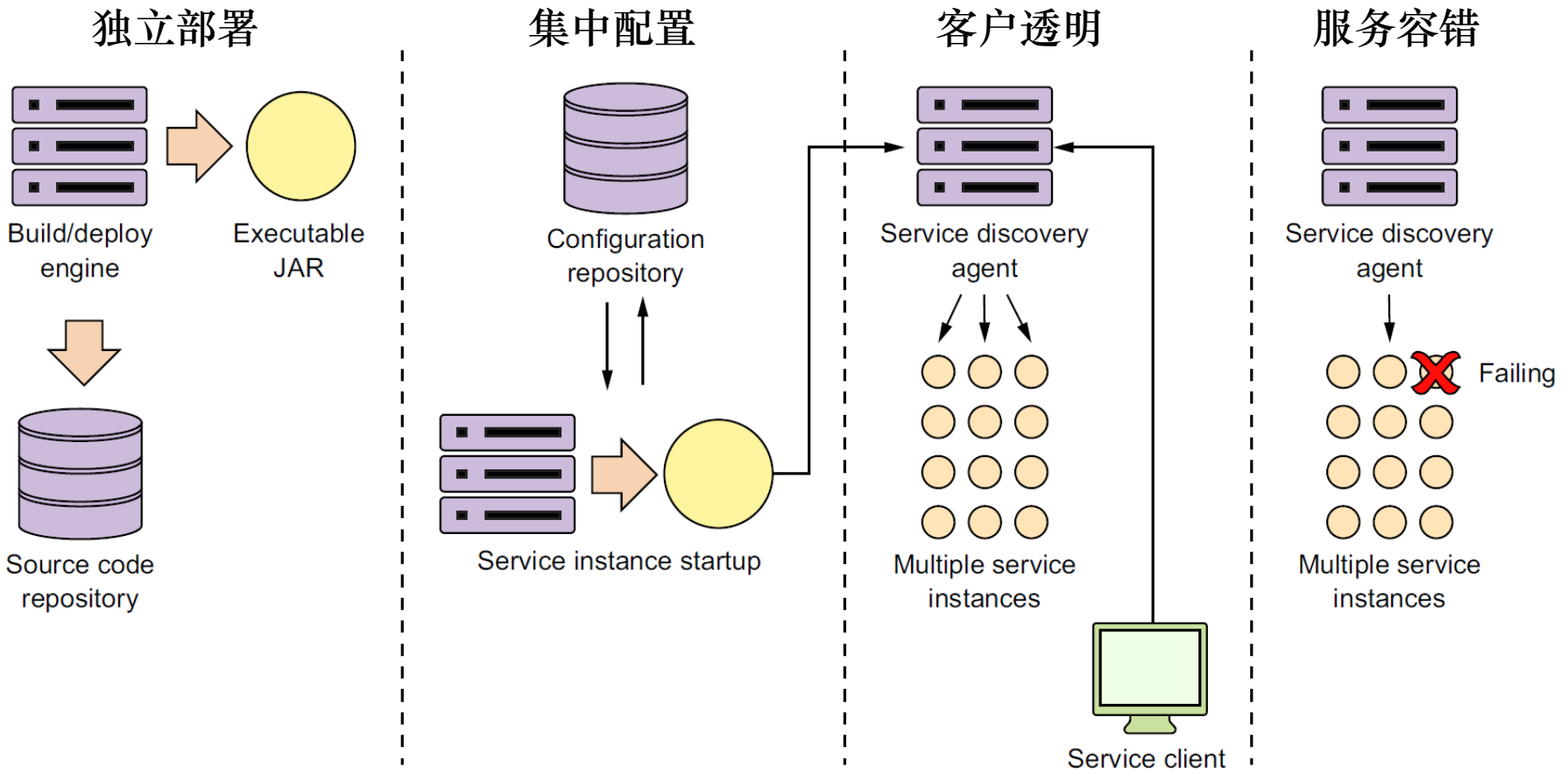
OOMALL





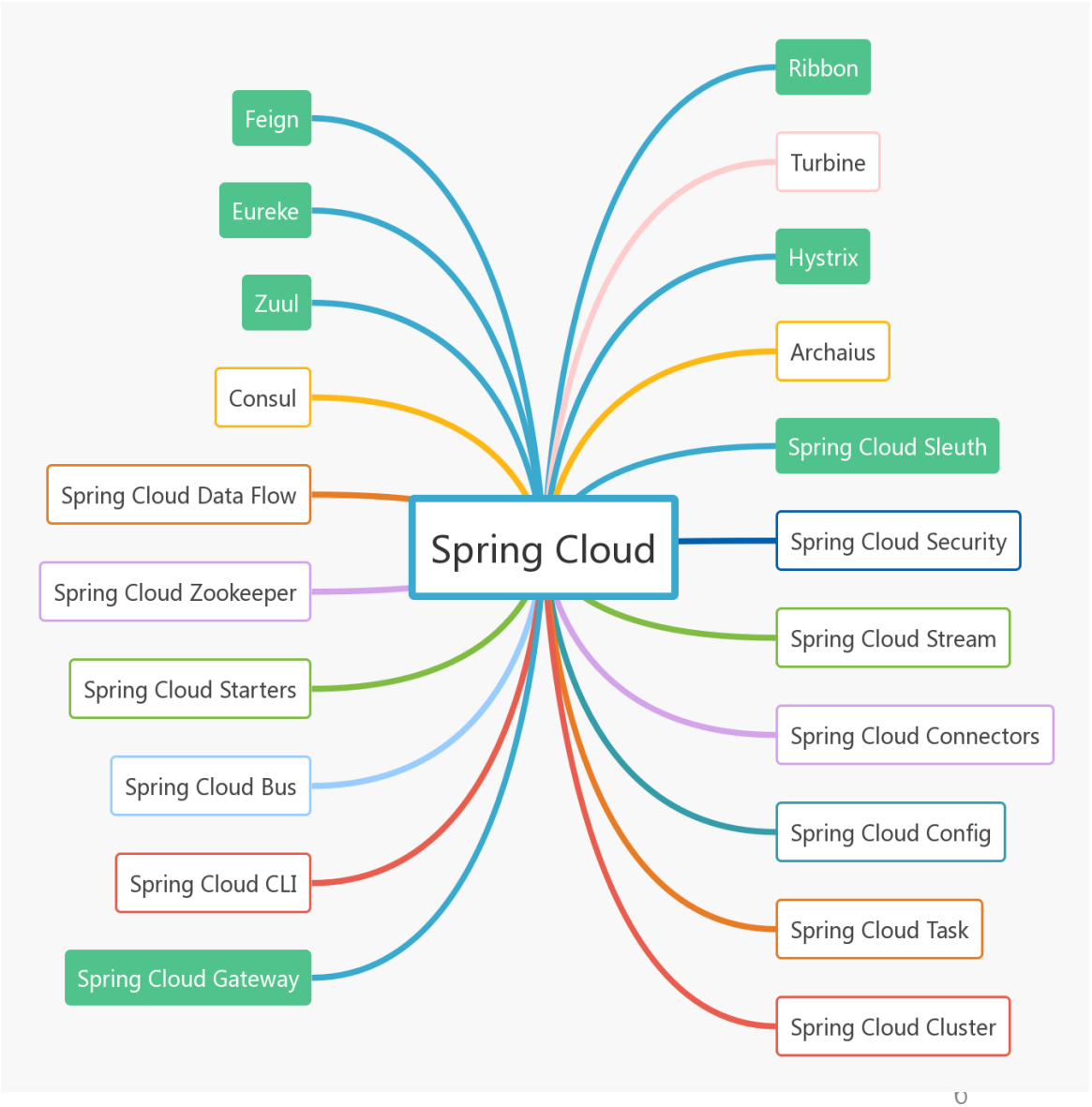
1.微服务的定义
Microservice Definition

- 微服务的四项基本原则



1.微服务的定义
Microservice Definition

- Spring Cloud
  - spring-cloud-gateway：服务网关
  - spring-cloud-openfeign：服务间调用
  - spring-cloud-consul：配置中心
  - Eureka：注册中心
  - resilience4j：限流熔断器



1.微服务的定义
Microservice Definition

- Spring Cloud Alibaba
  - Nacos：配置管理、服务发现
  - Sentinel：熔断限流器
  - Seata：分布式事务

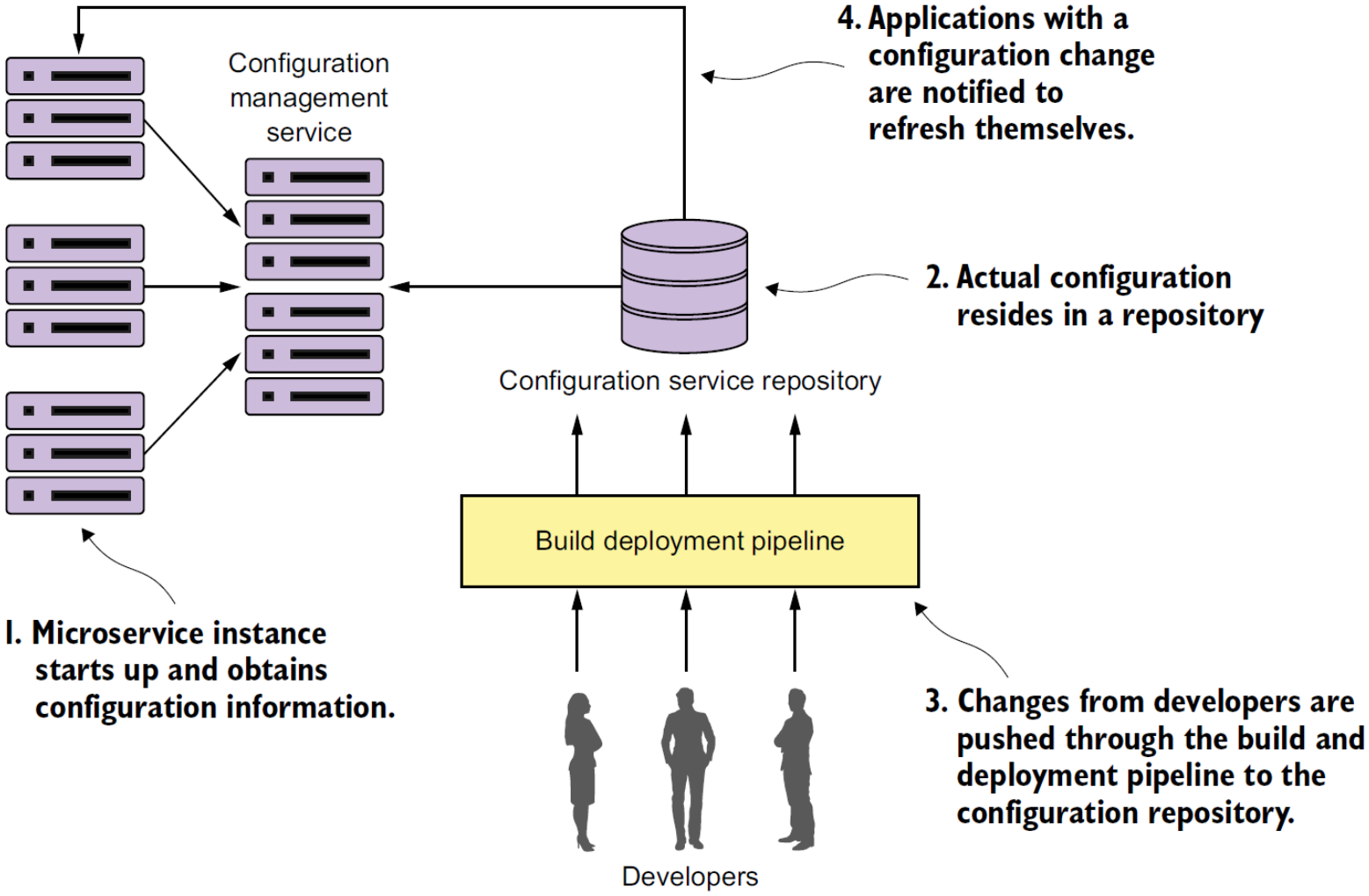
2.集中配置
Configuration Management

- 传统配置方式的局限
  - 安全性：配置跟随源代码保存在代码库中，容易造成配置泄漏
  - 时效性：修改配置，需要重启服务才能生效
  - 局限性：无法支持动态调整：例如日志开关、功能开关

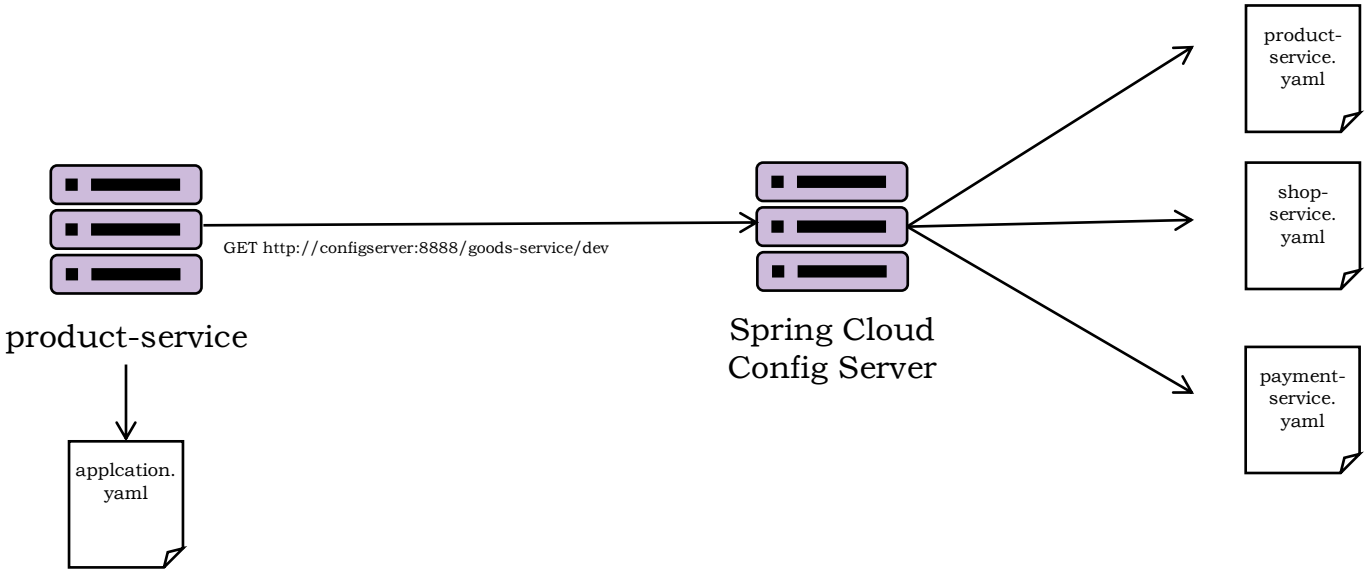
2.集中配置
Configuration Management

- 集中配置的要求
  - 配置信息应该不需要与服务实例在同一台服务器上，可以在服务启动时从集中的配置服务器上读取。
  - 配置服务应该以REST服务方式提供
  - 配置信息应该集中存储在几台服务器上
  - 配置服务应该提供高容错性

2.集中配置
Configuration Management

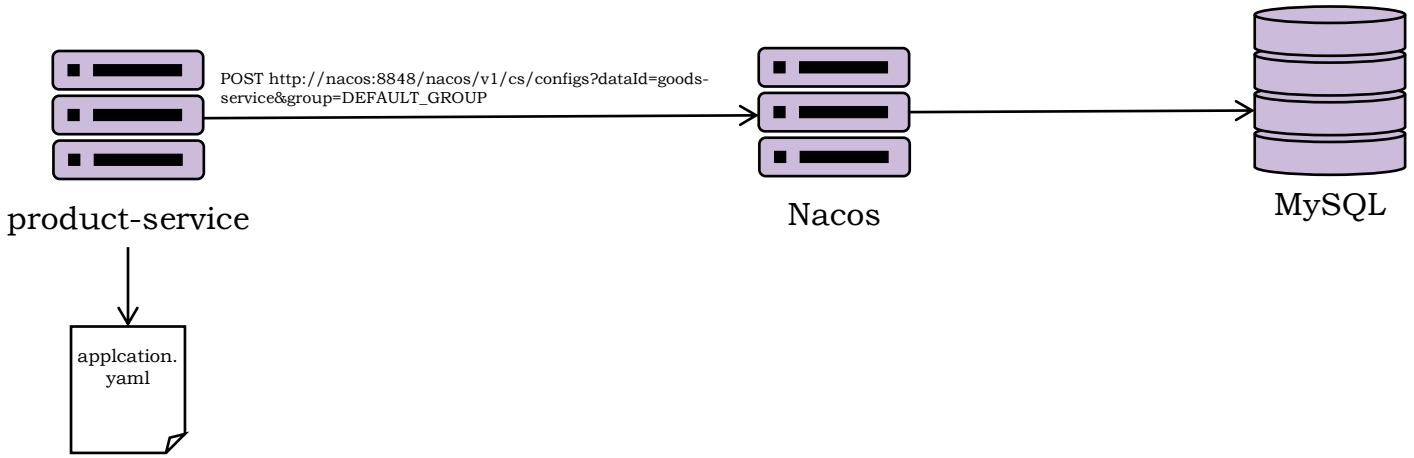


2.集中配置
Configuration Management



Spring Cloud Config Server

2.集中配置
Configuration Management



Nacos

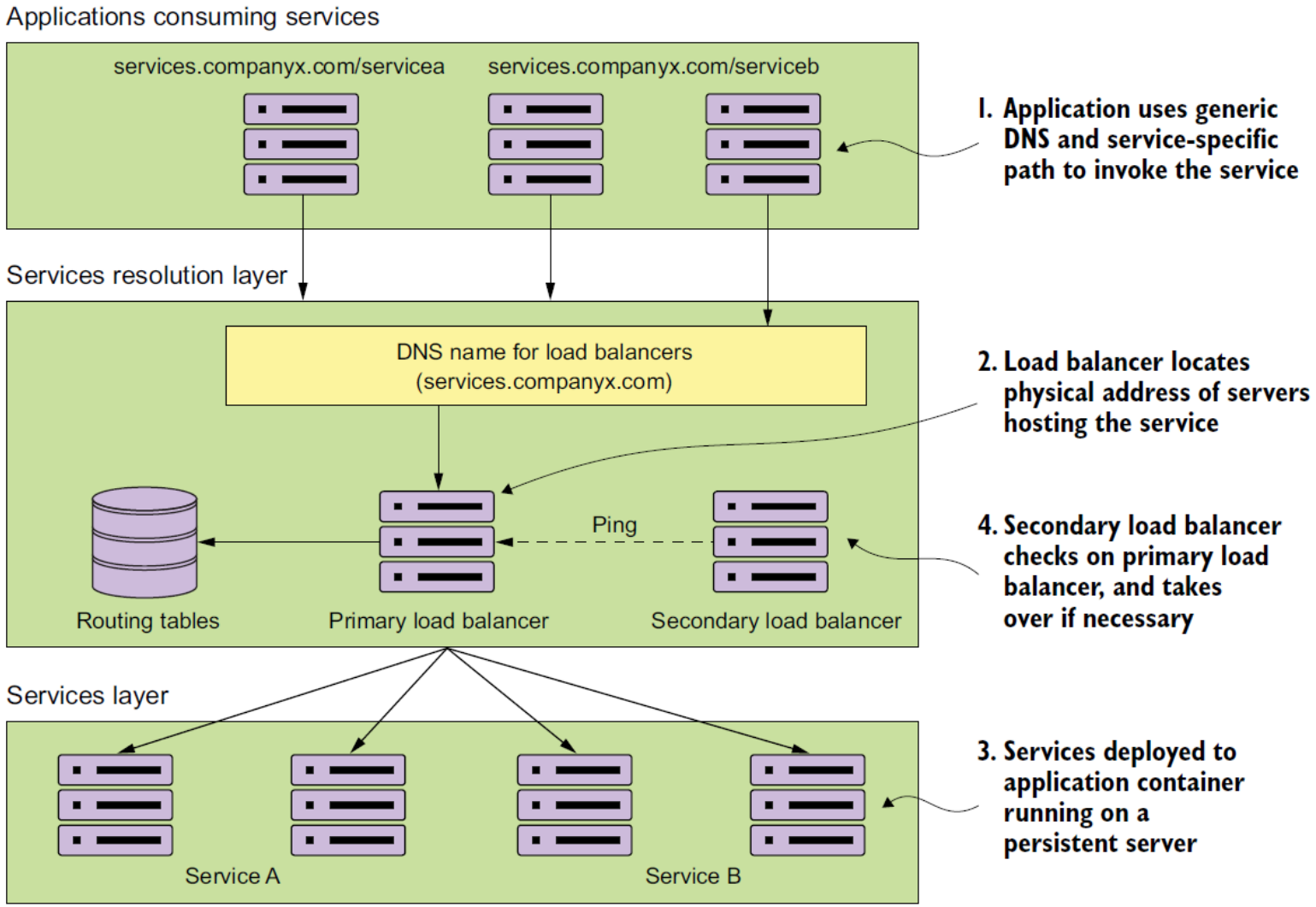
2.集中配置
Configuration Management

- Spring cloud bootstrap
  - 当配置信息集中在配置服务器，用什么定义配置服务器？

```yaml
spring:
  cloud:
    nacos:
      config:
        server-addr: nacos:8848
        file-extension: yaml
    inetutils:
      ignored-interfaces: eth.*
      preferred-networks: 10.0.1.;192.168.31.
  application:
    name: shop-service
```

3.服务发现与容错
Service Discovery and Fault-tolerant

- 传统方式
  - DNS
  - 动态负载平衡



3.服务发现与容错
Service Discovery and Fault-tolerant

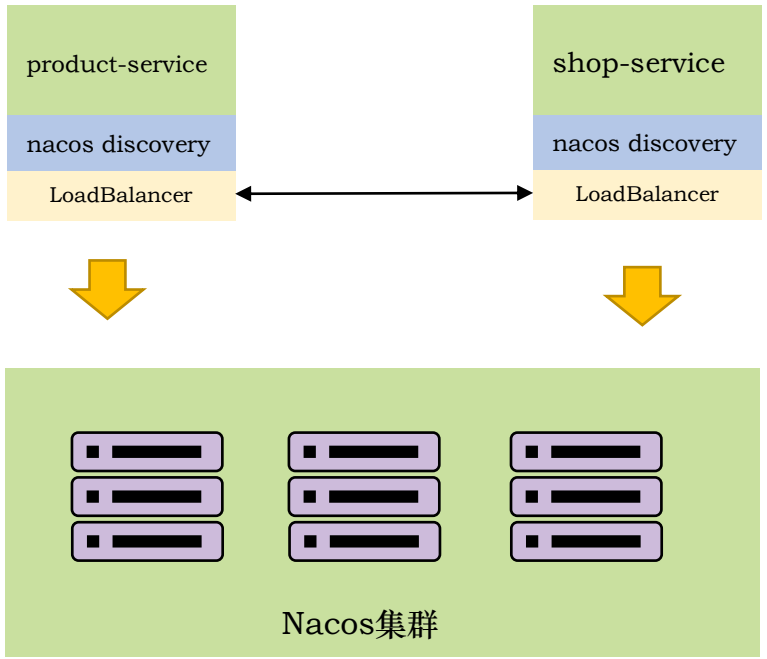
- 传统方式的弱点
  - 负载平衡服务器是整个系统的关键点，需要保持高可用性
  - 水平扩展能力有限，负载平衡服务器是系统的性能瓶颈
  - 服务的配置信息是静态管理的

3.服务发现与容错
Service Discovery and Fault-tolerant

- 服务发现的要求
  - 高可用性，支持采用多节点提供服务
  - 多节点间可共享服务信息
  - 支持动态负载平衡
  - 支持容错，即使服务发现集群全体失效，所有的微服务依然能利用缓存的服务信息继续工作
  - 自动恢复，能检测出失效的服务

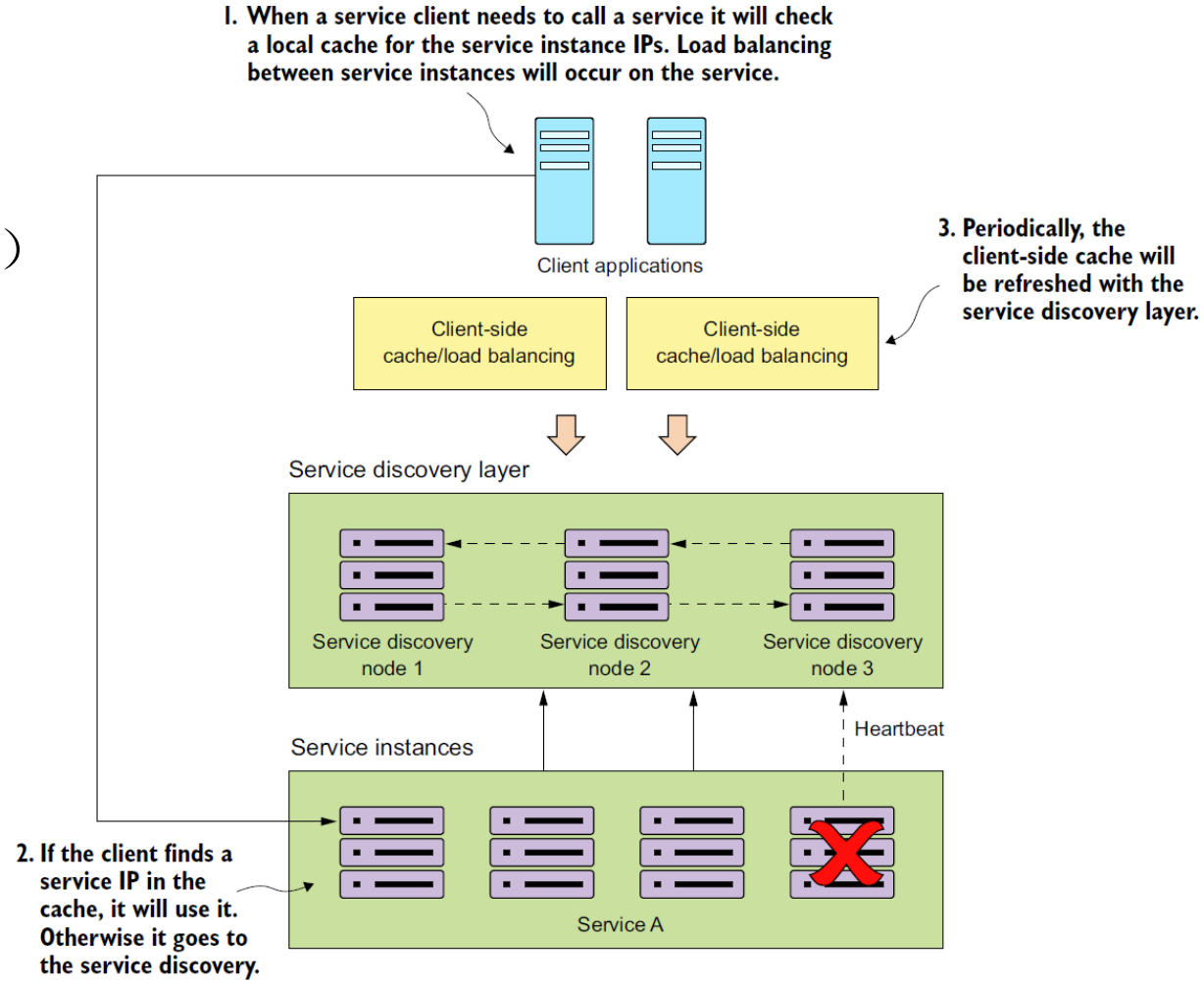
3.服务发现与容错
Service Discovery and Fault-tolerant

- Nacos的服务注册与发现



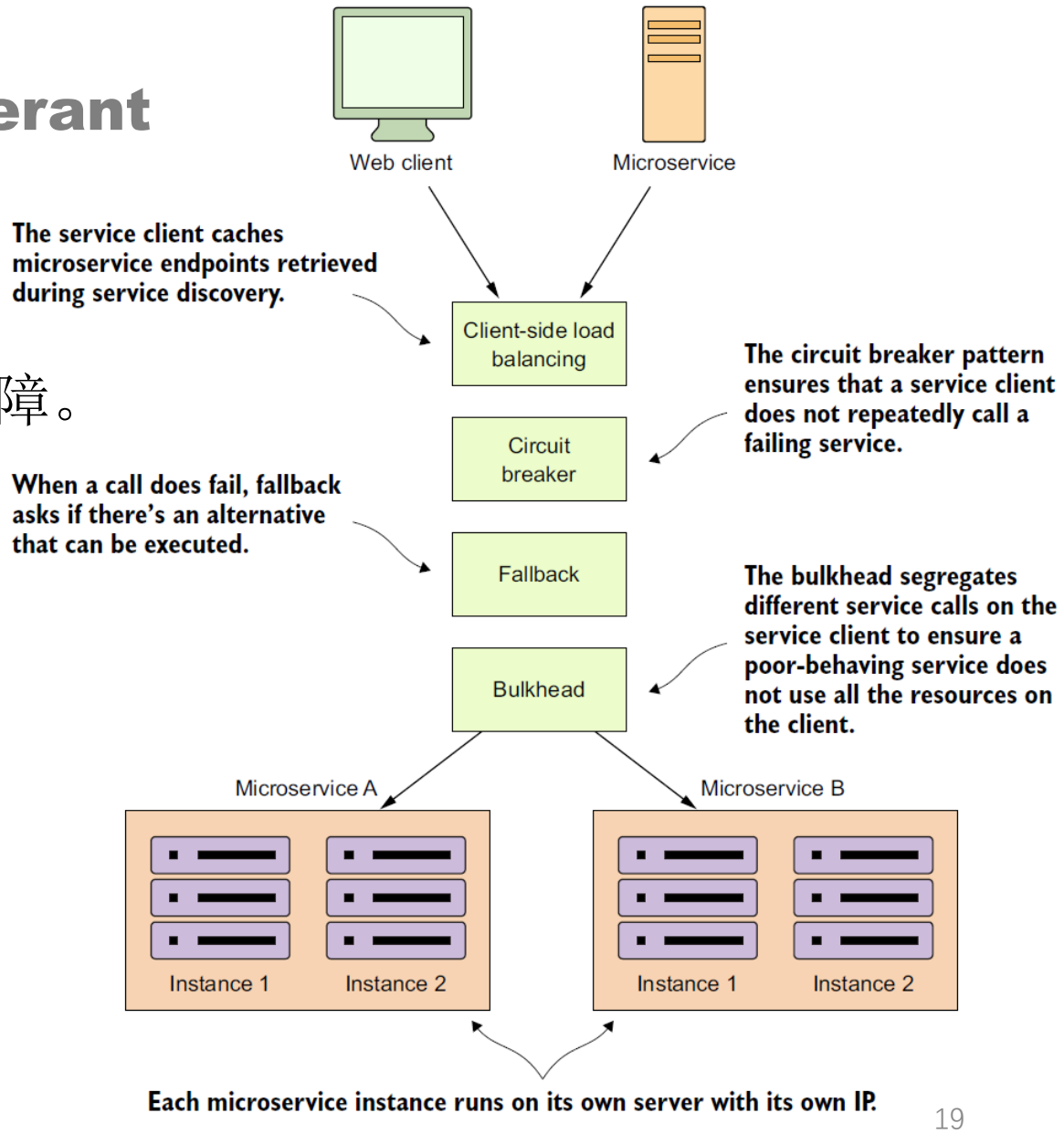
3.服务发现与容错
Service Discovery and Fault-tolerant

- 客户端负载均衡
  - Spring Cloud Load Balancer
    - 轮询（RoundRobinLoadBalancer）
    - 随机（RandomLoadBalancer）



3.服务发现与容错
Service Discovery and Fault-tolerant

- Circuit breaker：熔断器
  - Sentinel
  - 在复杂的分布式系统中阻止级联故障。
  - 快速失败，快速恢复。
  - 实时监控、警报和操作控制



4.服务调用
Service Invocation

- Openfeign
  - Spring 声明式Web Service客户端
  - 通过SpringMVC的注解实现Web Service调用

```
@FeignClient("shop-service")
public interface ShopMapper {

    @GetMapping("/shops/{id}")
    InternalReturnObject<Shop> getShopById(@PathVariable Long id);

    @GetMapping("/shops/{shopId}/templates/{id}")
    InternalReturnObject<Template> getTemplateById(@PathVariable Long shopId, @PathVariable Long id);

}
```

4.服务调用
Service Invocation

- Openfeign
  - Spring 声明式Web Service客户端
  - 通过SpringMVC的注解实现Web Service调用

```
@FeignClient("shop-service")
public interface ShopMapper {

    @GetMapping("/shops/{id}")
    InternalReturnObject<Shop> getShopById(@PathVariable Long id);

    @GetMapping("/shops/{shopId}/templates/{id}")
    InternalReturnObject<Template> getTemplateById(@PathVariable Long shopId, @PathVariable Long id);

}
```

4.服务调用
Service Invocation

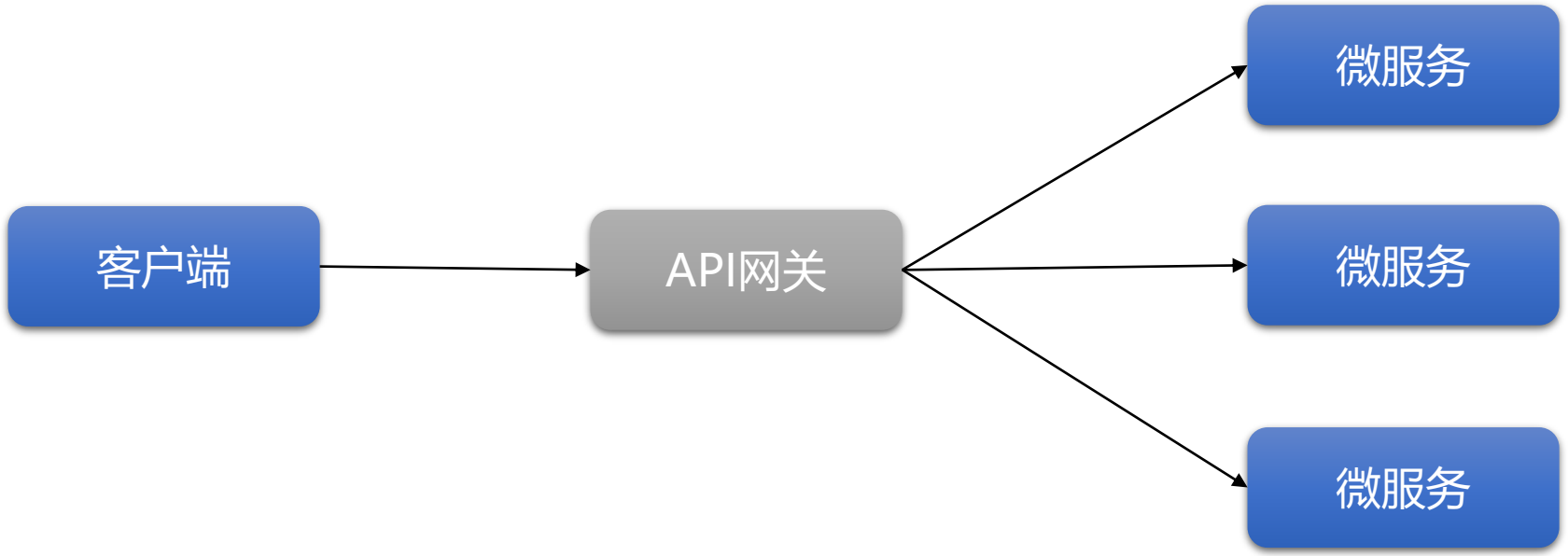
- gPRC

```
syntax = "proto3";
package greeting.v3;
import "google/api/annotations.proto";

option go_package = "github.com/garystafford/pb-greeting/gen/go/greeting/v3";

message Greeting {
  string id = 1;
  string service = 2;
  string message = 3;
  string created = 4;
  string hostname = 5;
}

message GreetingRequest {
  Greeting greeting = 1;
}

message GreetingResponse {
  repeated Greeting greeting = 1;
}

service GreetingService {
  rpc Greeting (GreetingRequest) returns (GreetingResponse) {
    option (google.api.http) = {
        get: "/api/greeting"
    };
  }
}
```

5.API网关
API Gateway

- 如果部署到多台服务器上，这些微服务如果都暴露给客户，是非常难以管理的，系统需要有一个唯一的出口。
- API网关是一个服务，是系统的唯一出口。API网关封装了系统内部的微服务，为客户端提供一个定制的API。
- 客户端只需要调用网关接口，就可以调用到实际的微服务，实际的服务对客户不可见，并且容易扩展服务。

5.API网关
API Gateway



5.API网关
API Gateway

- SpringCloud Gateway
  - 基于WebFlux框架实现的，底层使用了高性能的Reactor模式通信框架Netty。
  - 提供统一的路由方式
  - 基于 Filter 链的方式提供了网关基本的功能

5.API网关
API Gateway

- Filter（过滤器）
  - 使用它拦截和修改请求，并且对上游的响应，进行二次处理。
- Predicate（断言）
  - 使用Java 8 的 Predicate来匹配来自 HTTP 请求的任何内容
- Route（路由）
  - 网关配置的基本组成模块，一个Route模块由一个 ID，一个目标 URI，一组断言和一组过滤器定义。如果断言为真，则路由匹配，目标URI会被访问。

5.API网关
API Gateway

- 客户端向 Spring Cloud Gateway 发出请求。然后在 Gateway Handler Mapping 中找到与请求相匹配的路由，将其发送到 Gateway Web Handler。Handler 再通过指定的过滤器链来将请求发送到我们实际的服务执行业务逻辑，然后返回。过滤器之间用两种颜色分开是因为过滤器可能会在发送代理请求之前（“pre”）或之后（“post”）执行业务逻辑。