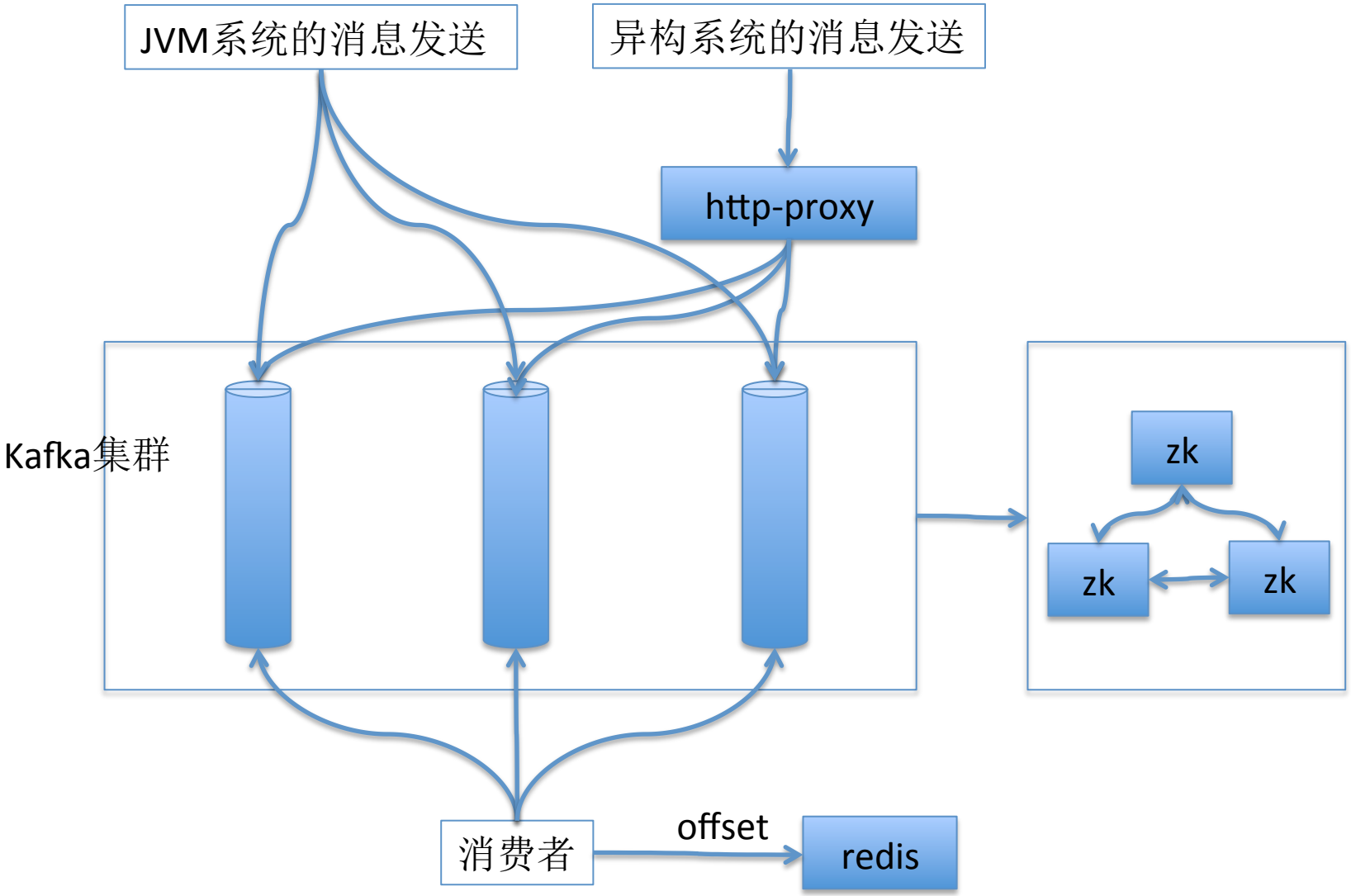
Kafka的使用经验


JVM系统的消息发送
异构系统的消息发送
http-proxy
Kafka集群
zk
zk
zk
消费者
offset
redis

Kafka的使用经验

- **基于SimpleConsumer的封装**
  - offset存于redis而非zookeeper
  - 允许一组消费端串行消费某一分区的消息

- 运维体会
  - 相对稳定，投入精力少（一个人10~20%精力）
  - 业务消息规模未达到亿级之前，可采取简单方式管理，topic只采用单分区

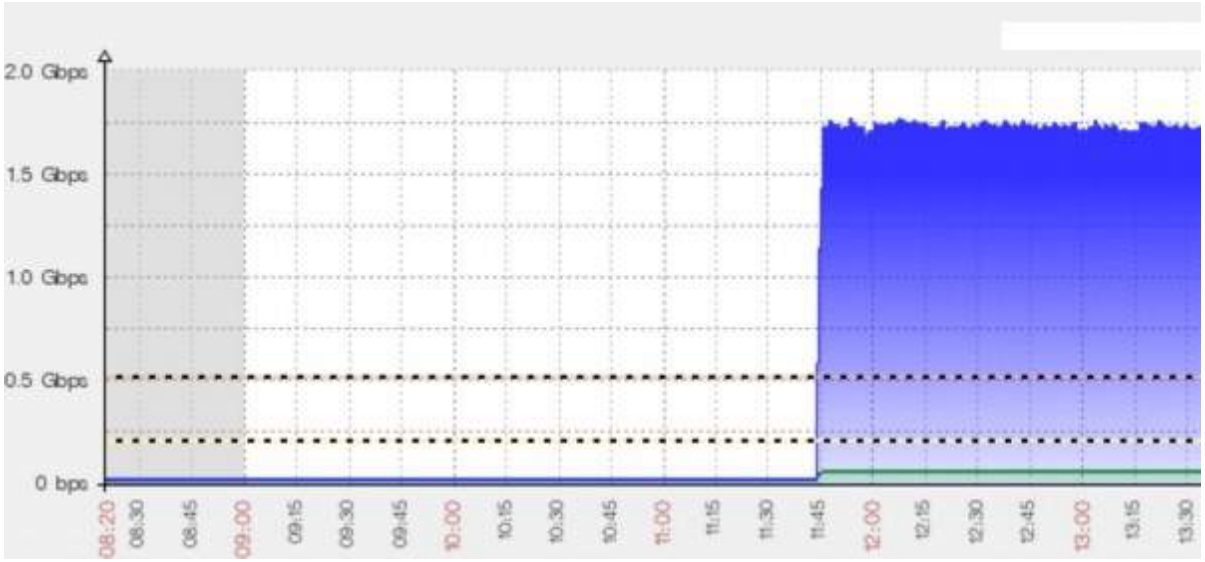
Kafka的使用经验

- 运维案例

  1）同步或异步都可能出现重复发送

  2）msgSize, replicaSize，fetchSize 的参数不一致问题

  3) 测试环境流量异常的问题

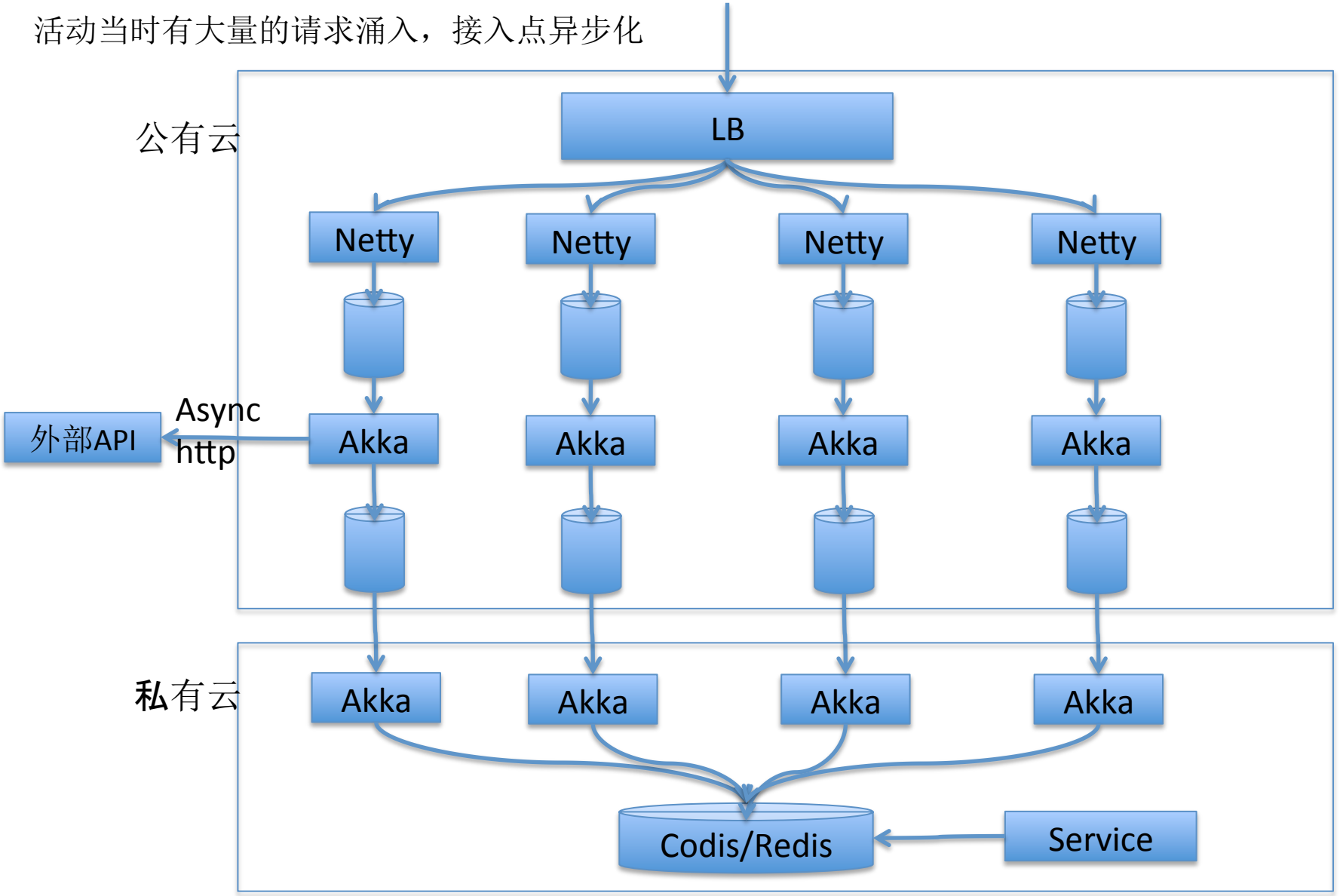


Akka的使用经验

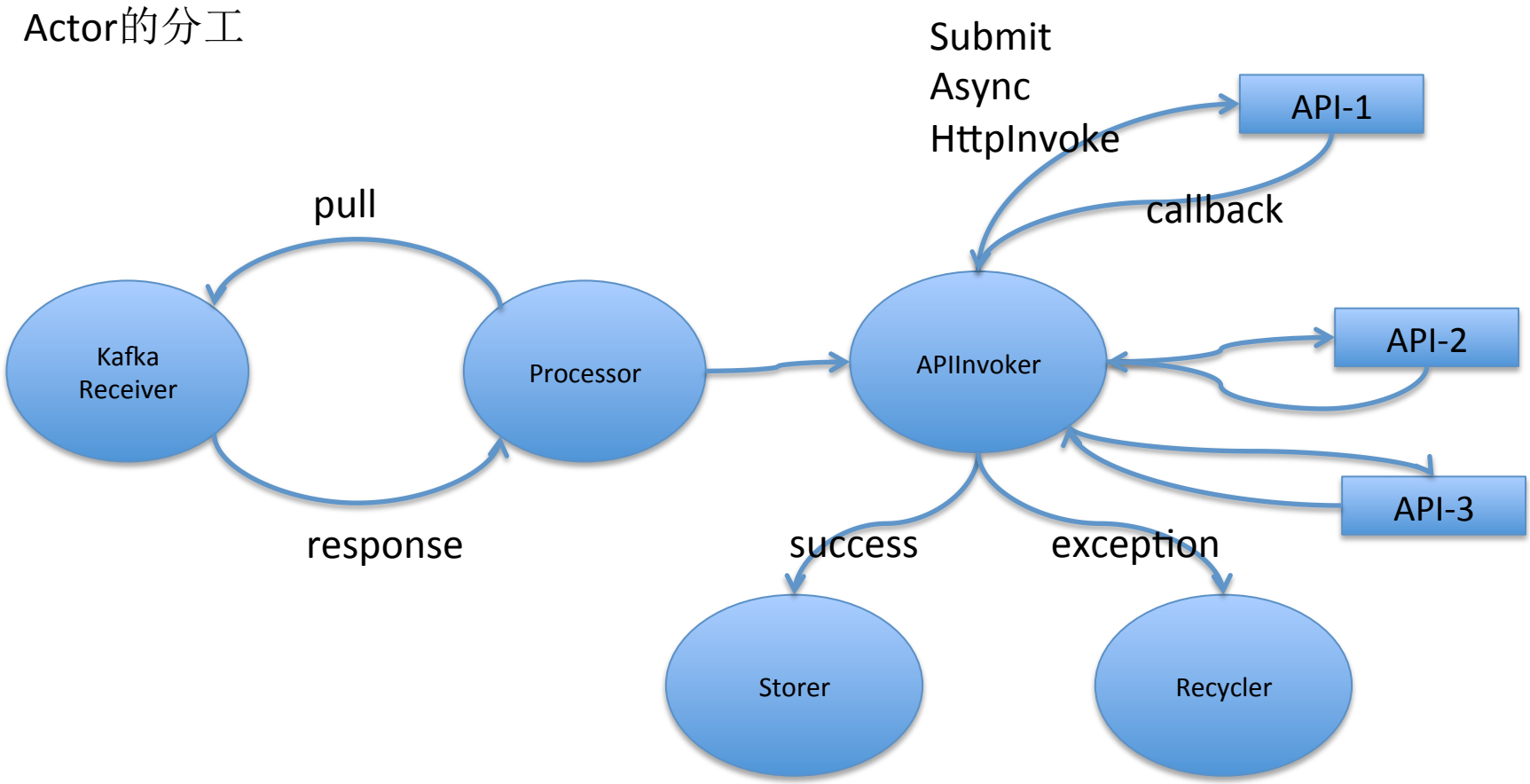
- 数据处理中大量使用
- 未使用持久化和集群等复杂特性
- 介绍一个简单的场景

基于混合云的引流

活动当时有大量的请求涌入，接入点异步化



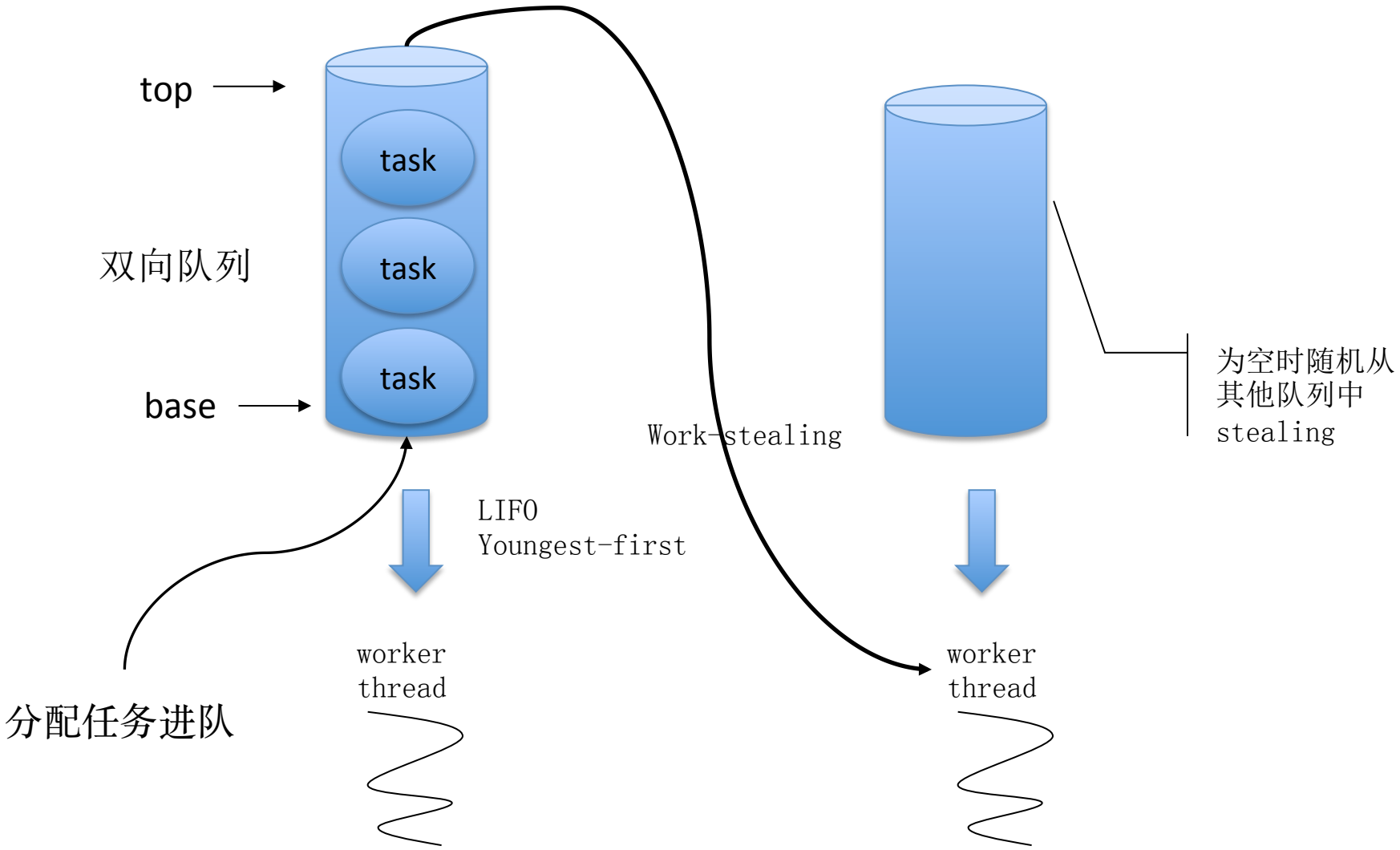
基于混合云的引流


Actor的分工
Submit
Async
HttpInvoke
API-1
callback
pull
Kafka
Receiver
Processor
APIInvoker
API-2
API-3
response
success
exception
Storer
Recycler

Akka的使用经验

- 无锁的世界
- 天然的扩展性
- 底层还是基于线程池的实现
- ForkJoin 效率很高，容易把CPU跑满

work-stealing （工作窃取）的实现



Akka的使用经验

- 与Kafka很搭
  - 分布式的管道与命令
- Akka 与 Spring 的整合
  - 对象/Actor 创建的问题
  - Akka Extension

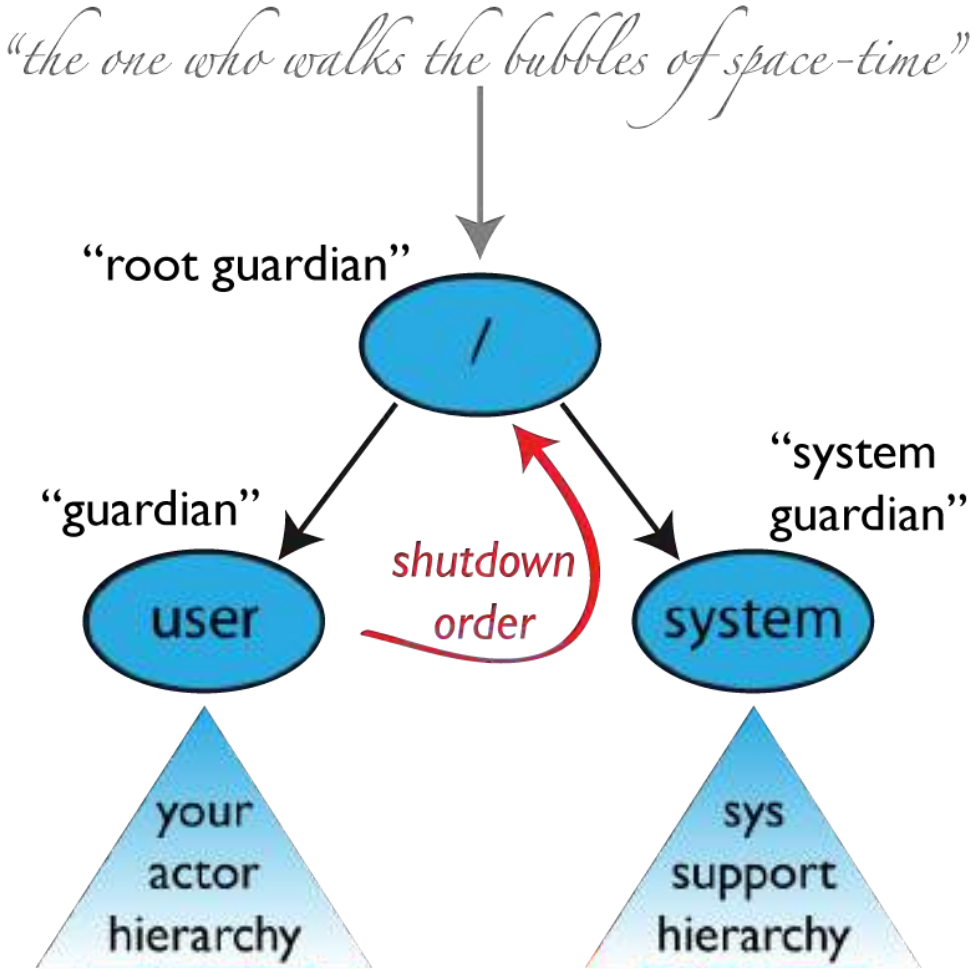
Akka的使用经验

- 实践小结
  - Actor职责尽可能单一
  - 避免阻塞
  - Supervisor和错误处理
  - Push vs Pull
  - WaterMark
  - 邮箱监控

Akka常用模式

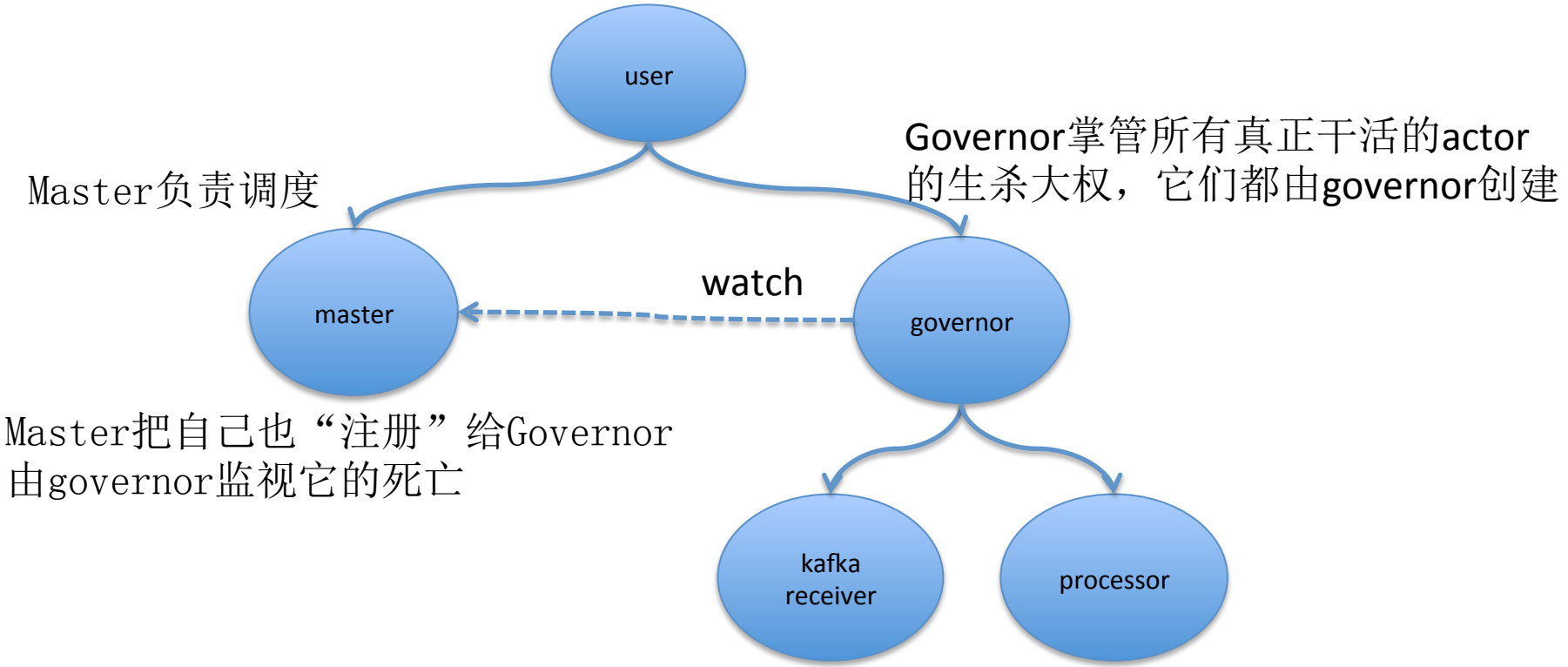
- 替身模式(cameo pattern)
  - Pattern.ask 就是一个例子
- Extra pattern
- Circuit Breaker
- WaterMark

优雅(顺序)关闭的问题


"the one who walks the bubbles of space-time"
"root guardian"
/
"guardian"
"system guardian"
shutdown order
user
system
your actor hierarchy
sys support hierarchy

优雅(顺序)关闭的问题

Governor： Terminator模式的变种



优雅(顺序)关闭的问题

提供一个 Governor的模板，实现顺序停止子Actor

```
abstract class ContextManager(signal: Semaphore) extends Actor {

 // 顺序的停止所有子actor
  protected def stopAll(kids: List[OrderedActorRef]): Future[Any] = {
    kids match {
      case first :: Nil =>
        gracefulStop(first.actor, stopTimeout).flatMap { _ => Future { AllDead } }
      case first :: rest =>
        gracefulStop(first.actor, stopTimeout).flatMap { _ => stopAll(rest) }
      case Nil =>
        Future { AllDead }
    }
  }
…
```

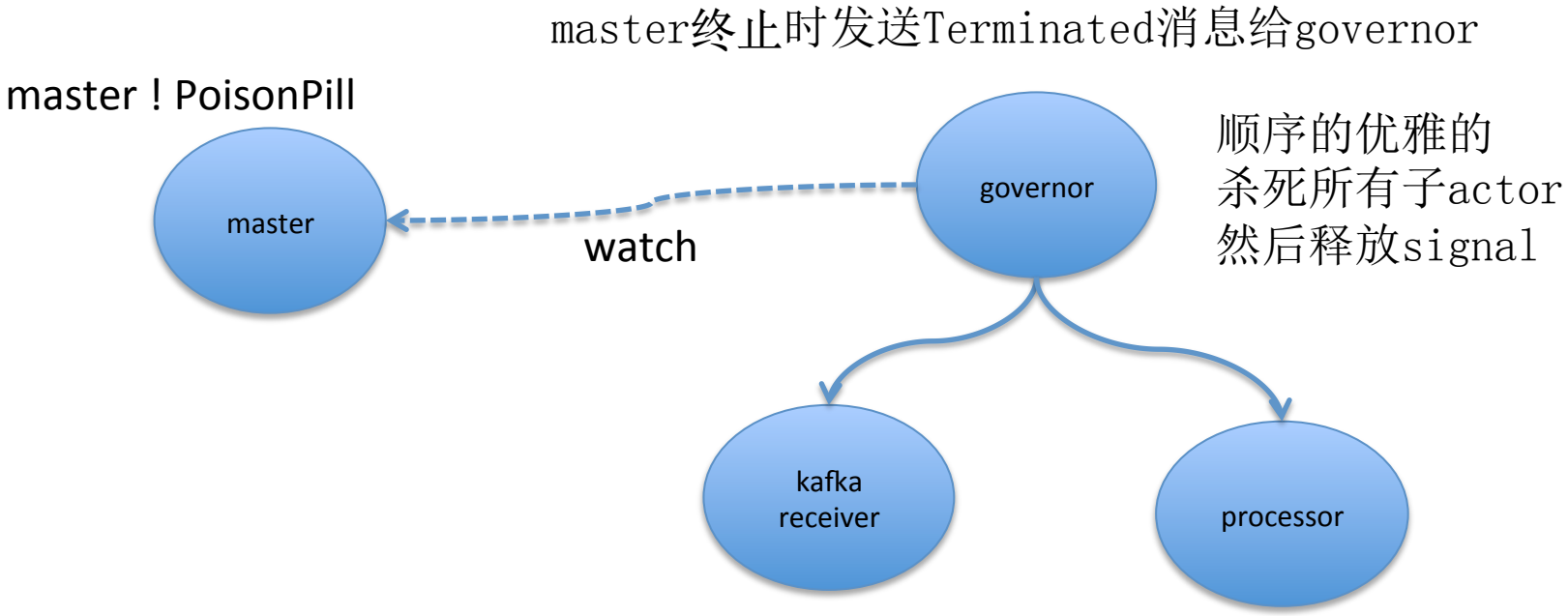
优雅(顺序)关闭的问题

实现 Governor 时，指定每个子Actor的关闭顺序，
比如下面在shutdown时先停止receiver再停止processor最后停止storer

```
class Governor(signal: Semaphore) extends ContextManager(signal) {
  override def createActors() {
    val ext = SpringExtension(context.system)
    // 1) receiver
    val receiver = context.actorOf(props(ext, "serialConsumer"), "kafkaReceiver”)
    setOrderNo(receiver, 0)
    // 2) processor
    val processor = context.actorOf(props(ext, "processor").withRouter(getProcessorRouter), "processor”)
    setOrderNo(processor, 1)
    // 3) storer
    val storer = context.actorOf(props(ext, "storer").withRouter(getStorerRouter), "storer”)
    setOrderNo(storer, 2)
  }
....
}
```

在系统关闭的逻辑里(shutdownhook)，master发给自己毒药丸，并等待 signal，signal满足后整个akka系统退出

优雅(顺序)关闭的问题


master终止时发送Terminated消息给governor
master ! PoisonPill
master
watch
governor
顺序的优雅的
杀死所有子actor
然后释放signal
kafka
receiver
processor