

极客大学 Java 进阶训练营

第 5 课

Netty 原理与 API 网关



KimmKing

Apache Dubbo/ShardingSphere PMC

个人介绍

Apache Dubbo/ShardingSphere PMC

前某集团高级技术总监/阿里架构师/某银行北京研发中心负责人

阿里云 MVP、腾讯 TVP、TGO 会员

10 多年研发管理和架构经验

熟悉海量并发低延迟交易系统的设计实现

目录

1. 再谈谈什么是高性能*
2. Netty 如何实现高性能*
3. Netty 网络程序优化
4. 典型应用：API 网关
5. 自己动手实现 API 网关*
6. 第 5 课总结回顾与作业实践

1. 再谈谈什么是高性能

什么是高性能？

大家思考一下，什么是高性能？

高并发用户 (Concurrent Users)

高吞吐量 (Throughput)

低延迟 (Latency)

~ 容量

什么是高性能?

大家思考一下，什么是高性能?

```
○ kimmking@JRA1W1PF227YSH: ~
kimmking@JRA1W1PF227YSH:~$ wrk -c 40 -d 30s --latency http://localhost:8088
Running 30s test @ http://localhost:8088/api/hello
  2 threads and 40 connections
  Thread Stats      Avg      Stdev      Max  +/- Stdev
    Latency      1.69ms  765.69us  16.95ms  73.87%
    Req/Sec      8.15k    1.14k   13.09k  79.17%
  Latency Distribution
    50%      1.57ms
    75%      2.07ms
    90%      2.64ms
    99%      4.05ms
  487205 requests in 30.07s, 58.17MB read
  Requests/sec: 16204.49
  Transfer/sec: 1.93MB
```

高并发用户 (Concurrent Users)

什么是高性能?

大家思考一下，什么是高性能?

```
○ kimmking@JRA1W1PF227YSH: ~
kimmking@JRA1W1PF227YSH:~$ wrk -c 40 -d 30s --latency http://localhost:8088
Running 30s test @ http://localhost:8088/api/hello
  2 threads and 40 connections
  Thread Stats      Avg      Stdev      Max  +/- Stdev
    Latency      1.69ms  765.69us  16.95ms  73.87%
    Req/Sec      8.15k    1.14k    13.09k  79.17%
  Latency Distribution
    50%      1.57ms
    75%      2.07ms
    90%      2.64ms
    99%      4.05ms
  487205 requests in 30.07s, 58.17MB read
Requests/sec: 16204.49
Transfer/sec:  1.95MB
```

高吞吐量 (Throughput)

什么是高性能?

大家思考一下，什么是高性能?

```
kimmking@JRA1W1PF227YSH: ~
kimmking@JRA1W1PF227YSH:~$ wrk -c 40 -d 30s --latency http://localhost:8088
Running 30s test @ http://localhost:8088/api/hello
  2 threads and 40 connections
Thread Stats      Avg      Stdev     Max  +/- Stdev
  Latency      1.69ms  765.69us  16.95ms  73.87%
  Req/Sec      8.15k    1.14k   13.09k  79.17%
Latency Distribution
  50%      1.57ms
  75%      2.07ms
  90%      2.64ms
  99%      4.05ms
  487205 requests in 30.07s, 58.17MB read
Requests/sec: 16204.49
Transfer/sec: 1.93MB
```

低延迟 (Latency)

高性能的另一面

如果实现了高性能，有什么副作用呢？

1、系统复杂度 $\times 10$ 以上

2、建设与维护成本++++

3、故障或 BUG 导致的破坏性 $\times 10$ 以上

应对策略

稳定性建设（混沌工程）：

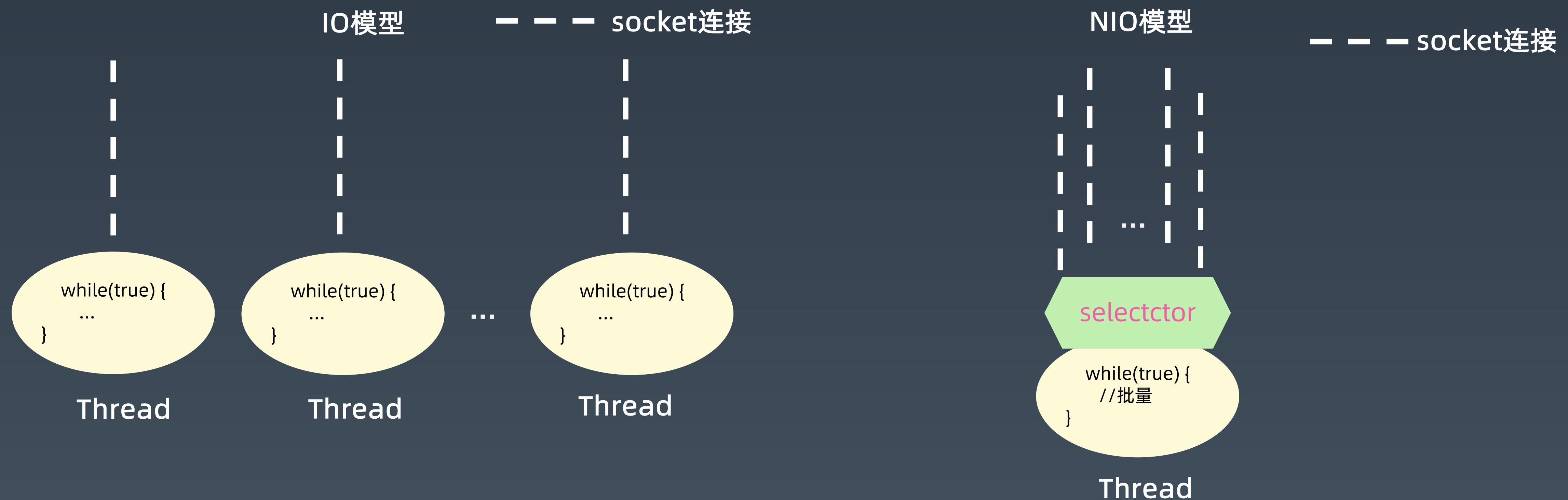
1、容量

2、爆炸半径

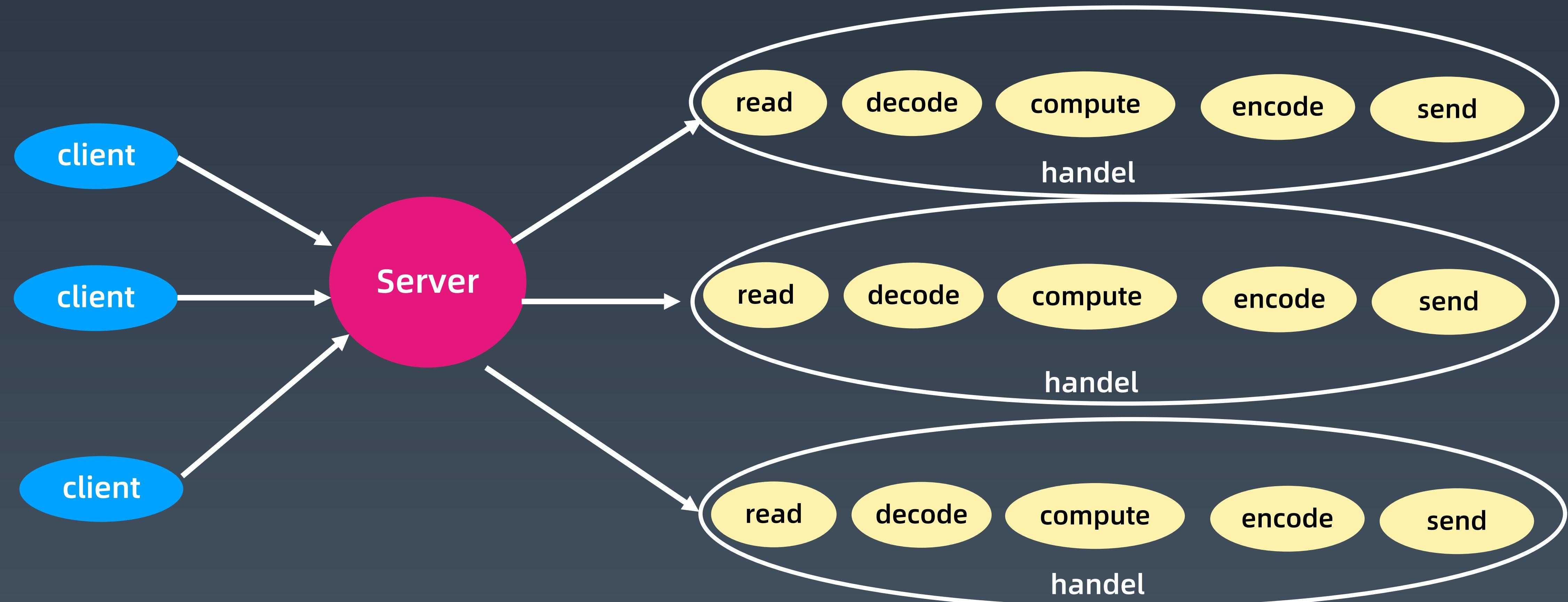
3、工程方面积累与改进

2. Netty 如何实现高性能

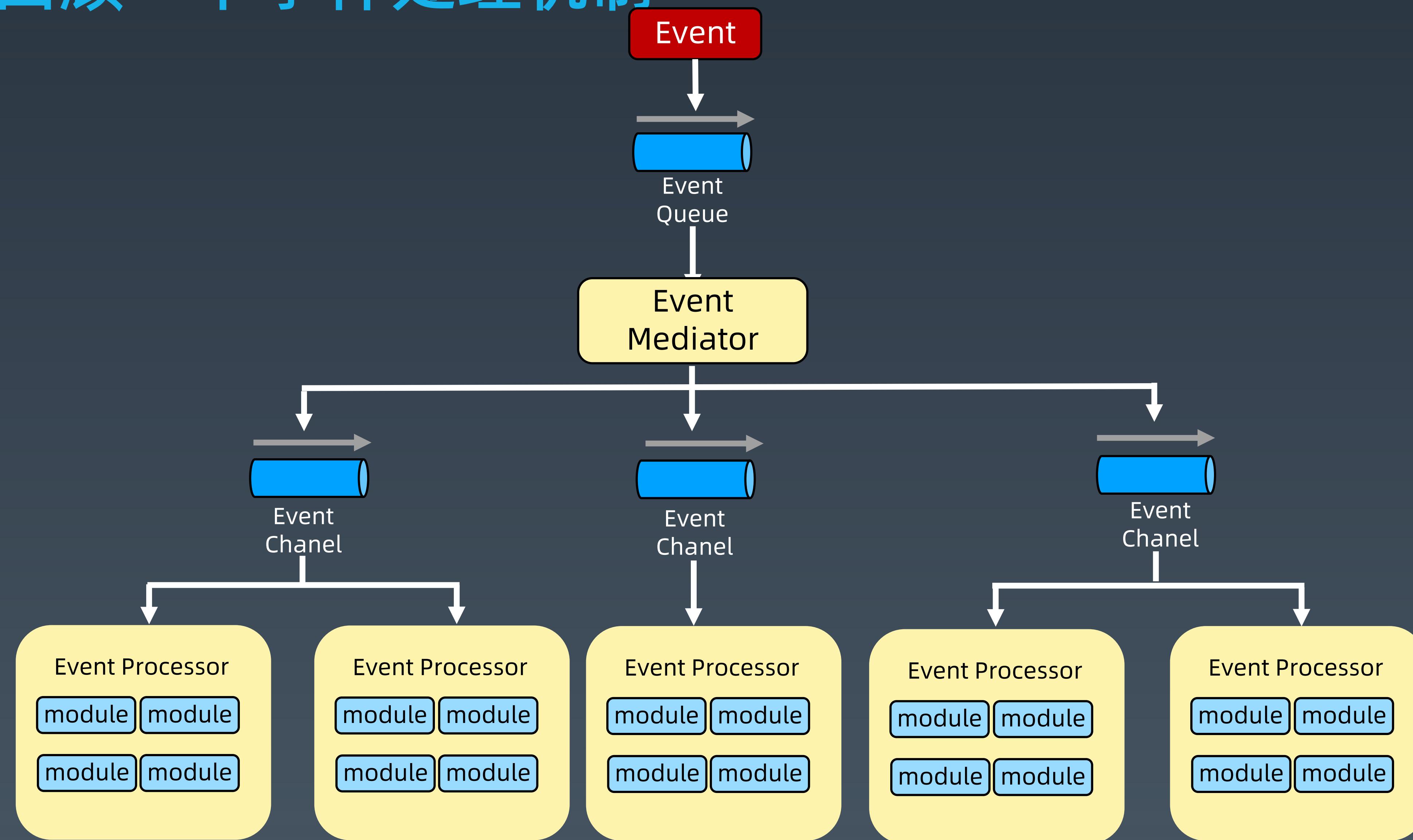
从 Socket IO 到 NIO



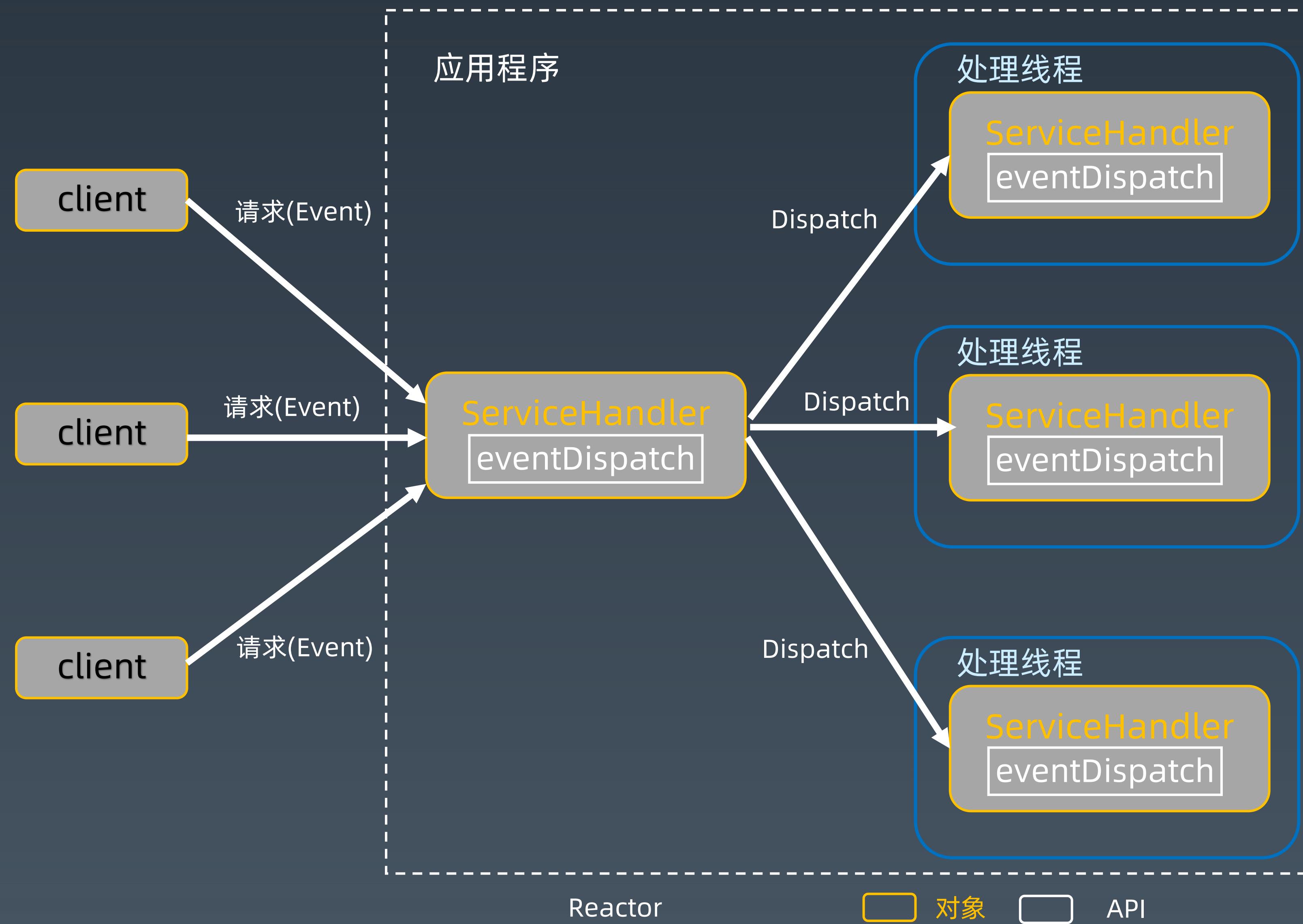
从 Socket IO 到 NIO--BIO多线程模型



回顾一下事件处理机制



从事件处理机制到 Reactor 模型

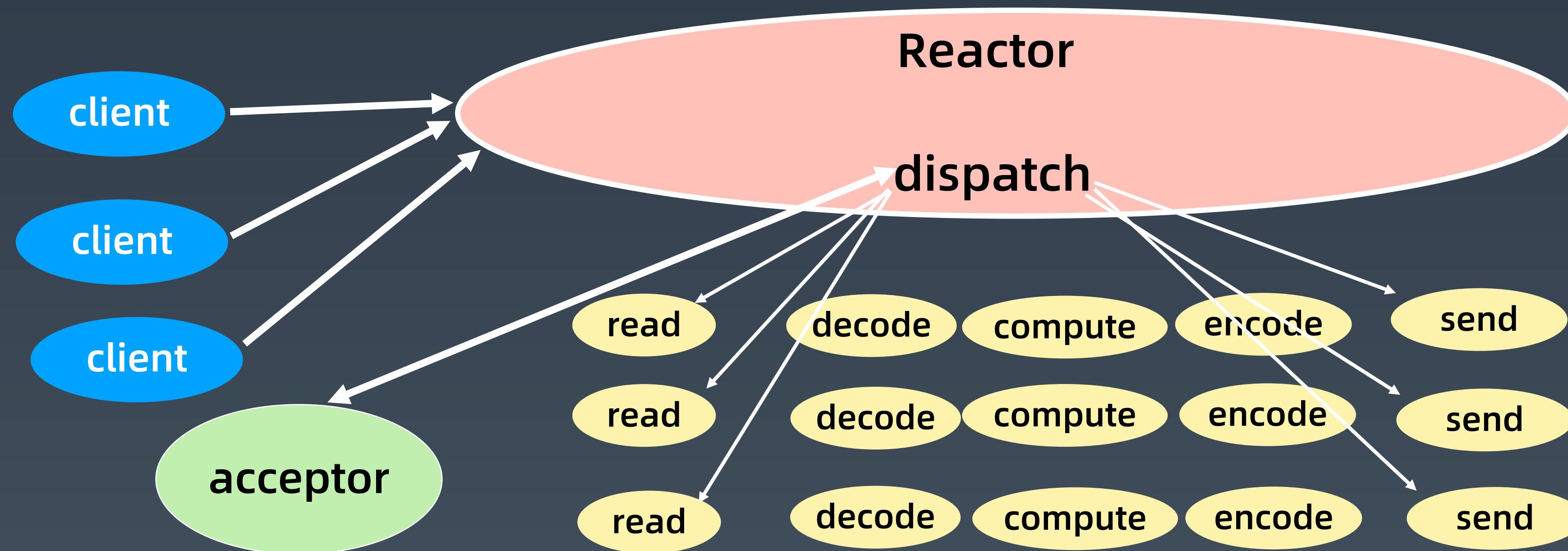


Reactor 模式首先是事件驱动的，有一个或者多个并发输入源，有一个 Service Handler 和多个 EventHandlers。

这个 Service Handler 会同步的将输入的请求多路复用的分发给相应的 Event Handler。

从 Reactor 模型到 Netty NIO--01

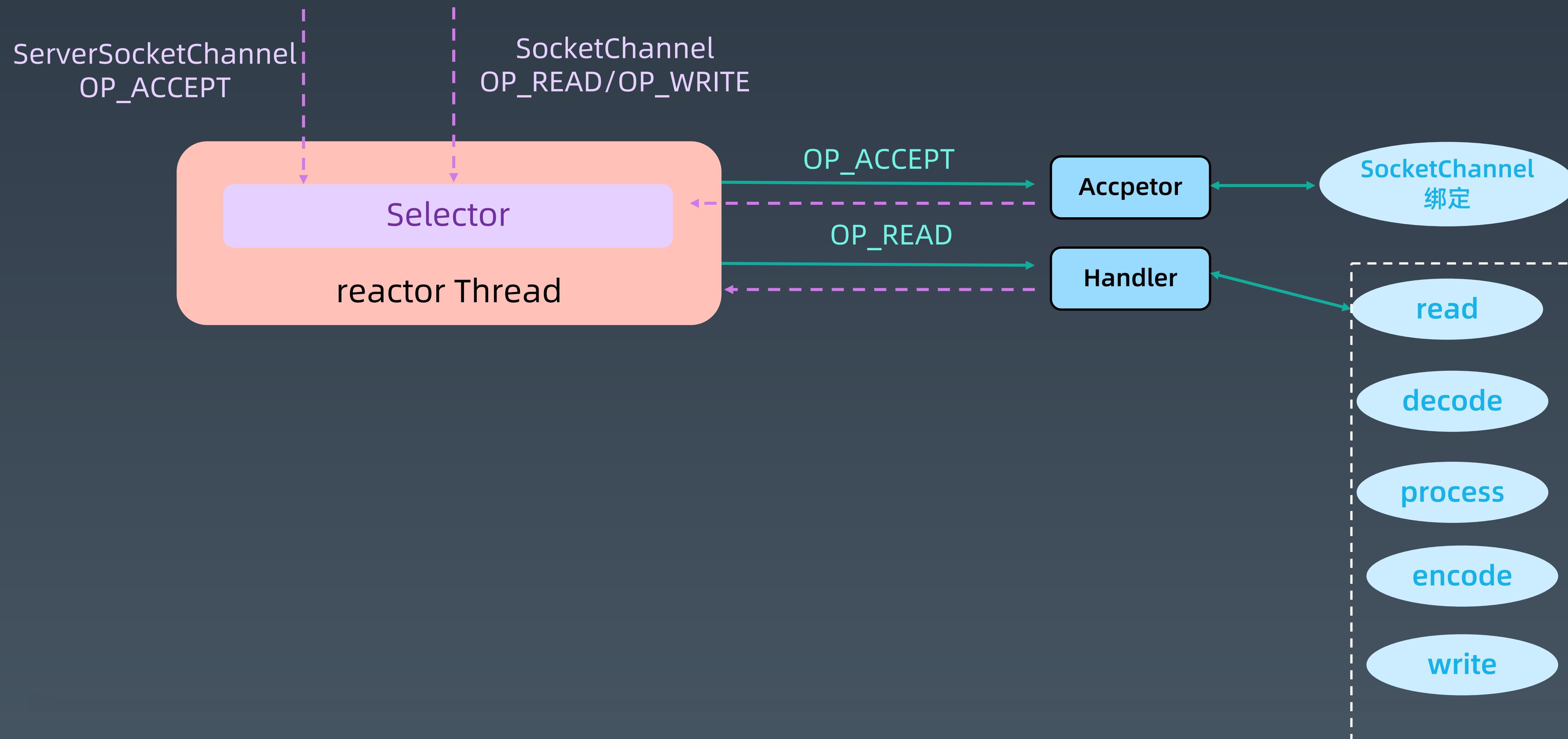
Reactor单线程模型



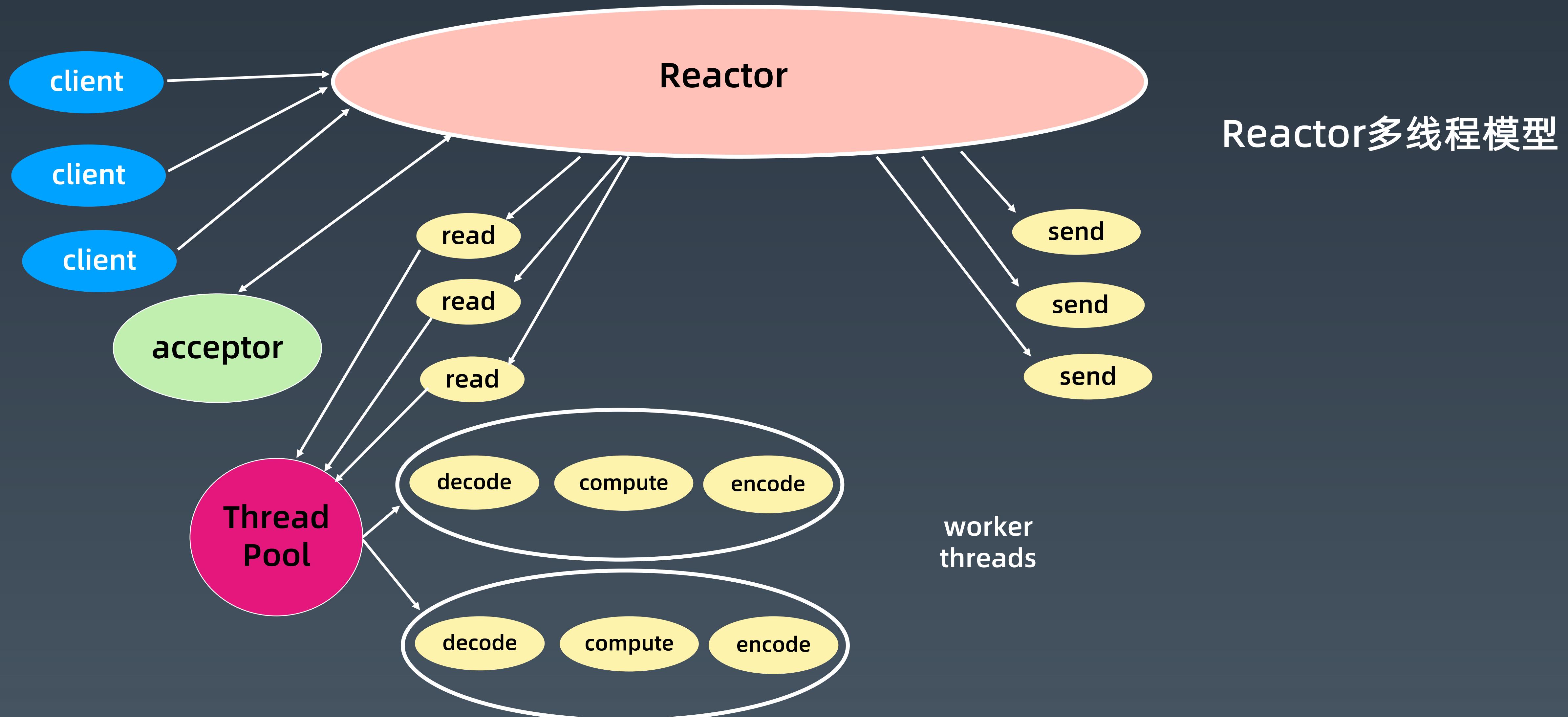
Doug Lea 《Scalable IO in Java》

从 Reactor 模型到 Netty NIO--01

Reactor单线程模型

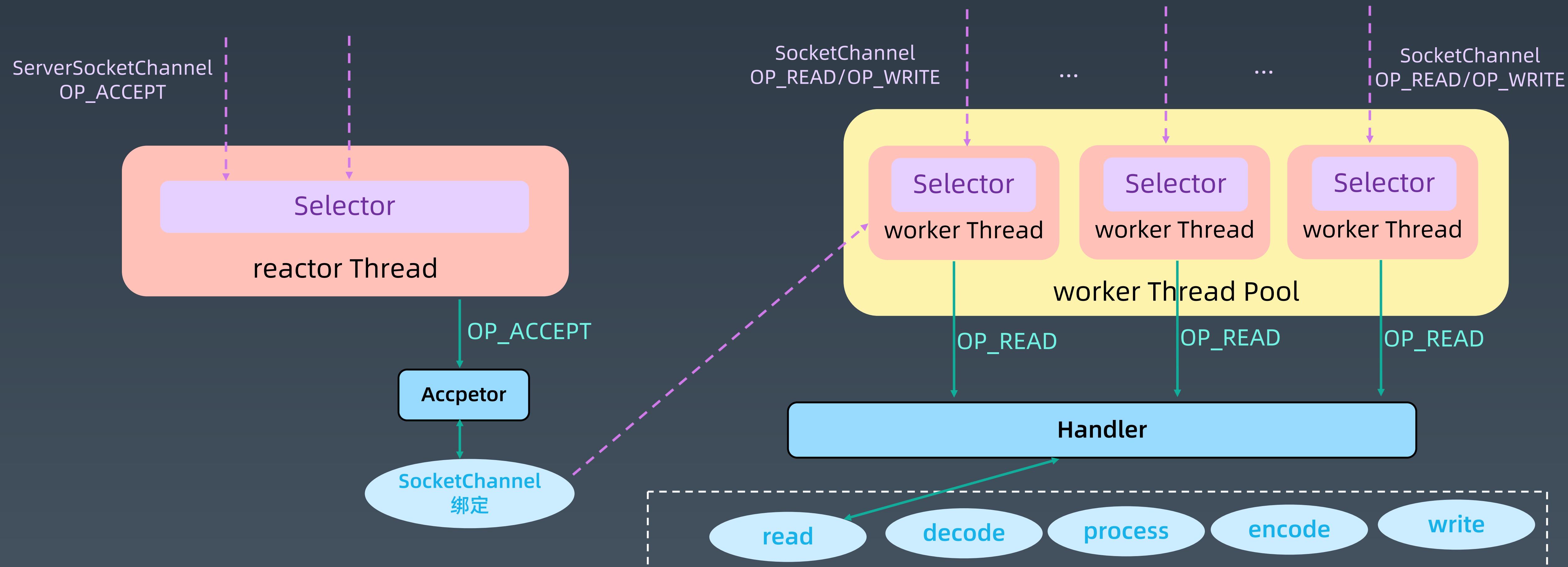


从 Reactor 模型到 Netty NIO--02



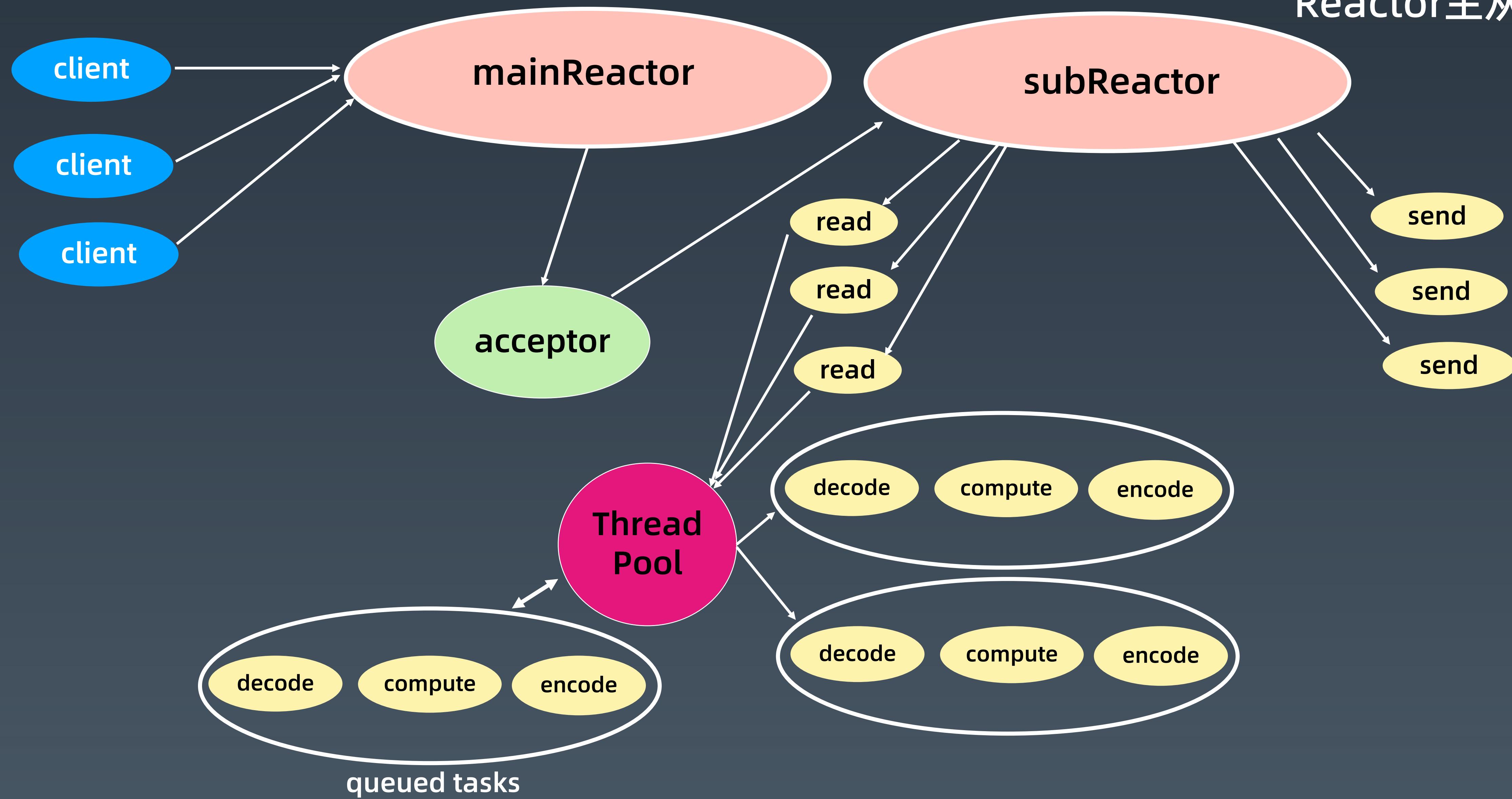
从 Reactor 模型到 Netty NIO--02

Reactor多线程模型



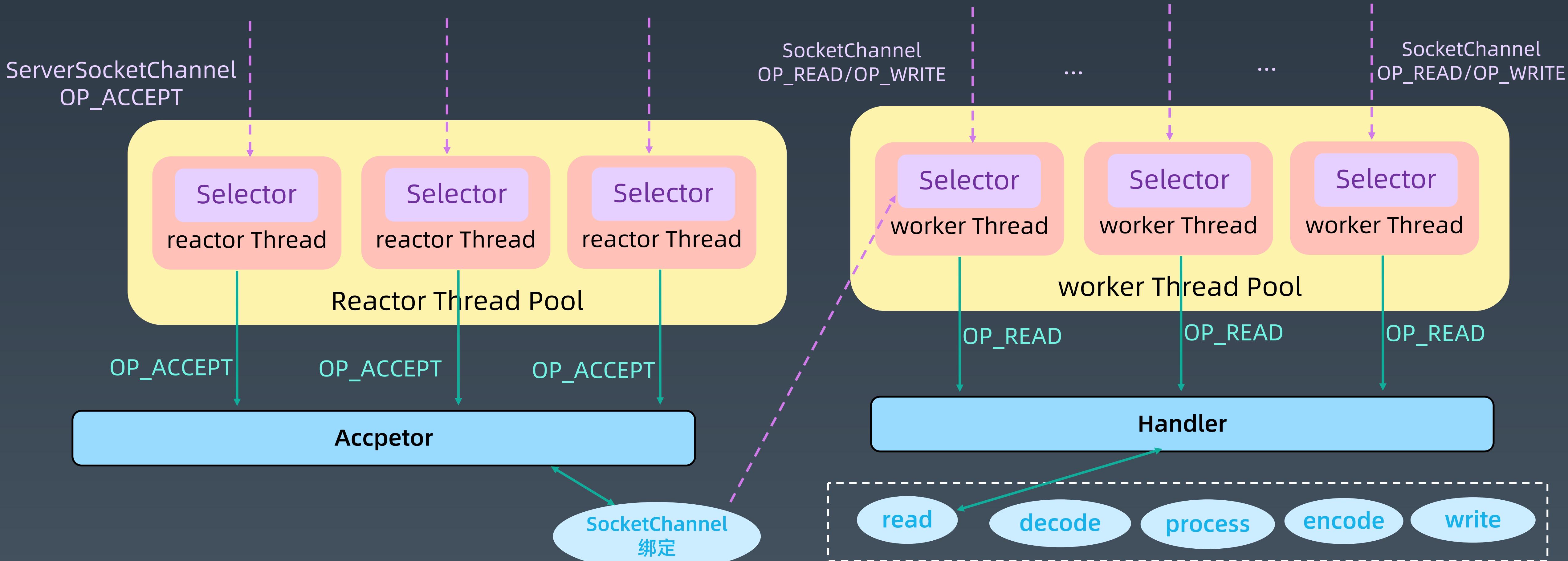
从 Reactor 模型到 Netty NIO--03

Reactor主从模型



从 Reactor 模型到 Netty NIO--03

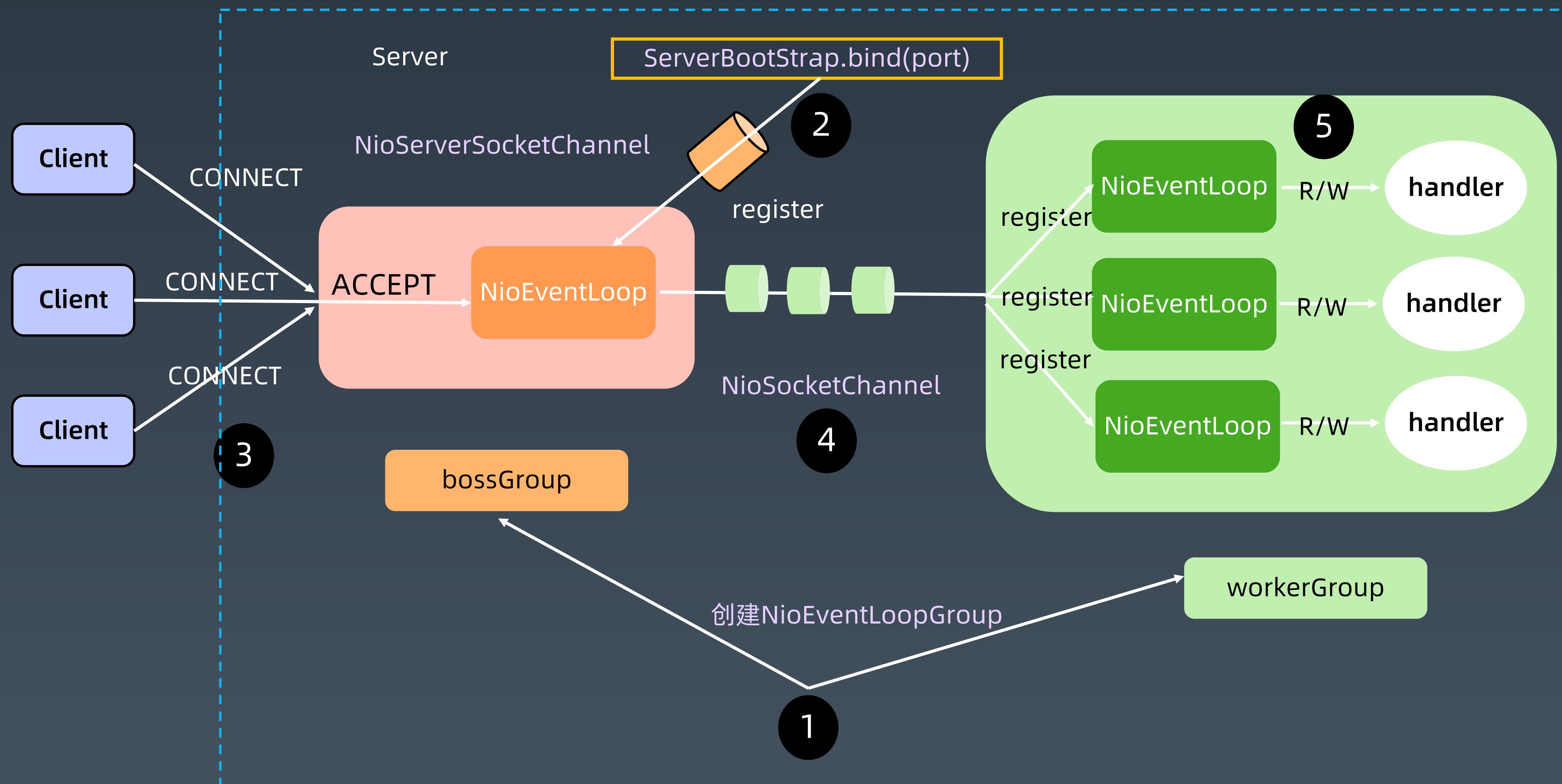
Reactor主从模型



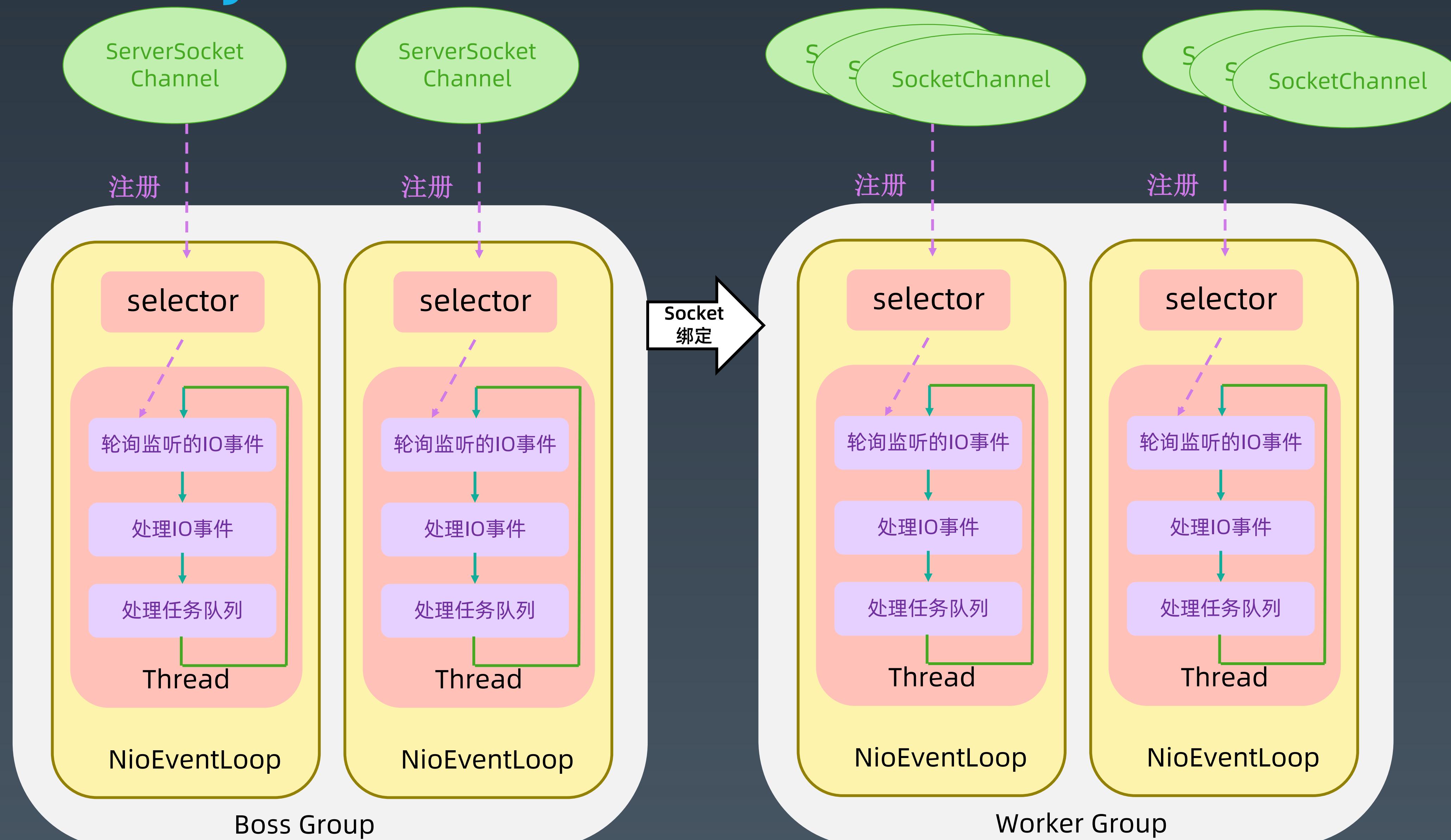
Netty对三种模式的支持

Reactor 单线程模式	<pre>EventLoopGroup eventGroup = new NioEventLoopGroup(1); ServerBootstrap serverBootstrap = new ServerBootstrap(); serverBootstrap.group(eventGroup);</pre>
非主从 Reactor 多线程模式	<pre>EventLoopGroup eventGroup = new NioEventLoopGroup(); ServerBootstrap serverBootstrap = new ServerBootstrap(); serverBootstrap.group(eventGroup);</pre>
主从 Reactor 多线程模式	<pre>EventLoopGroup bossGroup = new NioEventLoopGroup(); EventLoopGroup workerGroup = new NioEventLoopGroup(); ServerBootstrap serverBootstrap = new ServerBootstrap(); serverBootstrap.group(bossGroup, workerGroup);</pre>

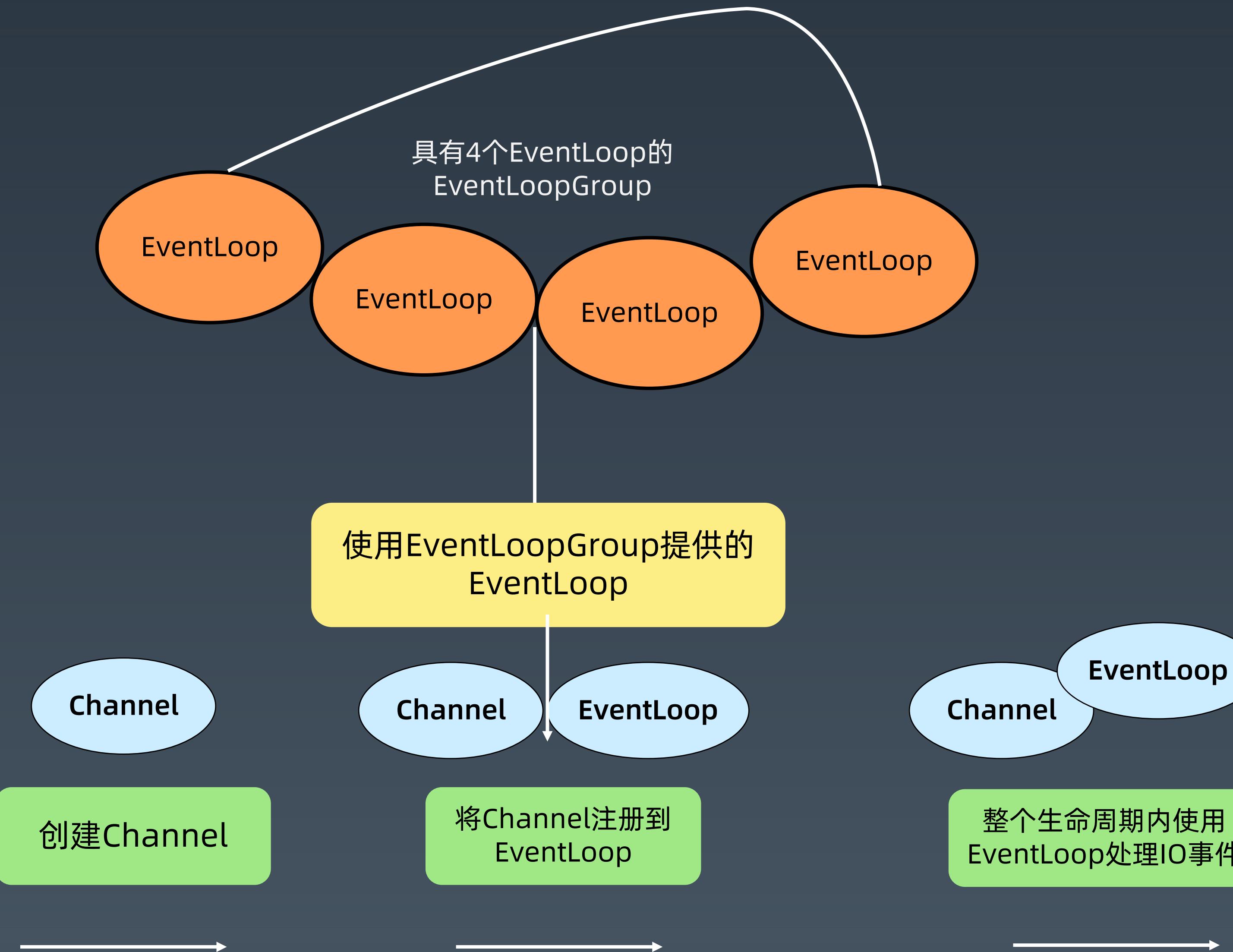
Netty启动和处理流程



Netty线程模式

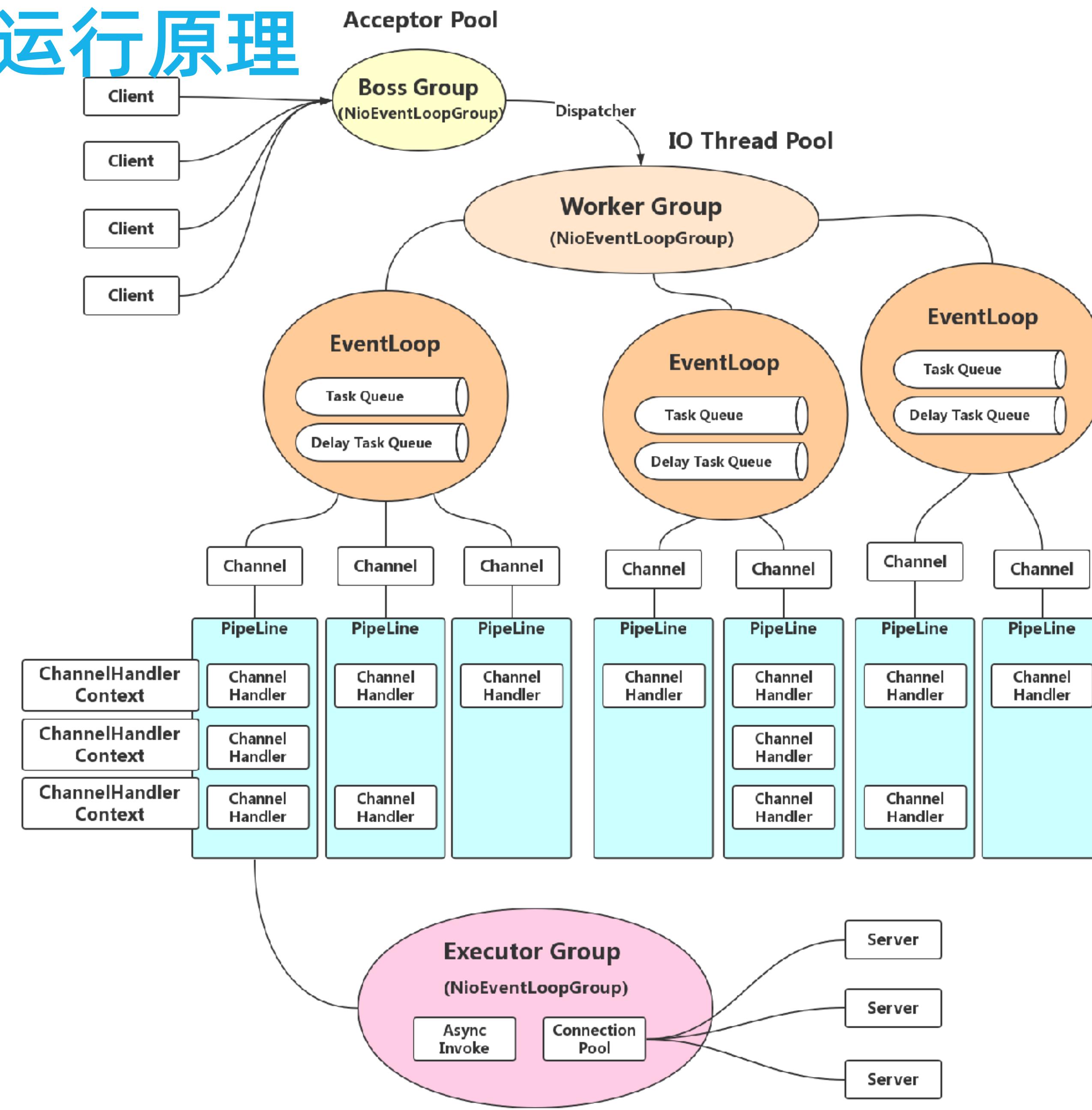


Netty核心对象



到底什么是
EventLoop

Netty 运行原理



关键对象

Bootstrap: 启动线程, 开启 socket

EventLoopGroup

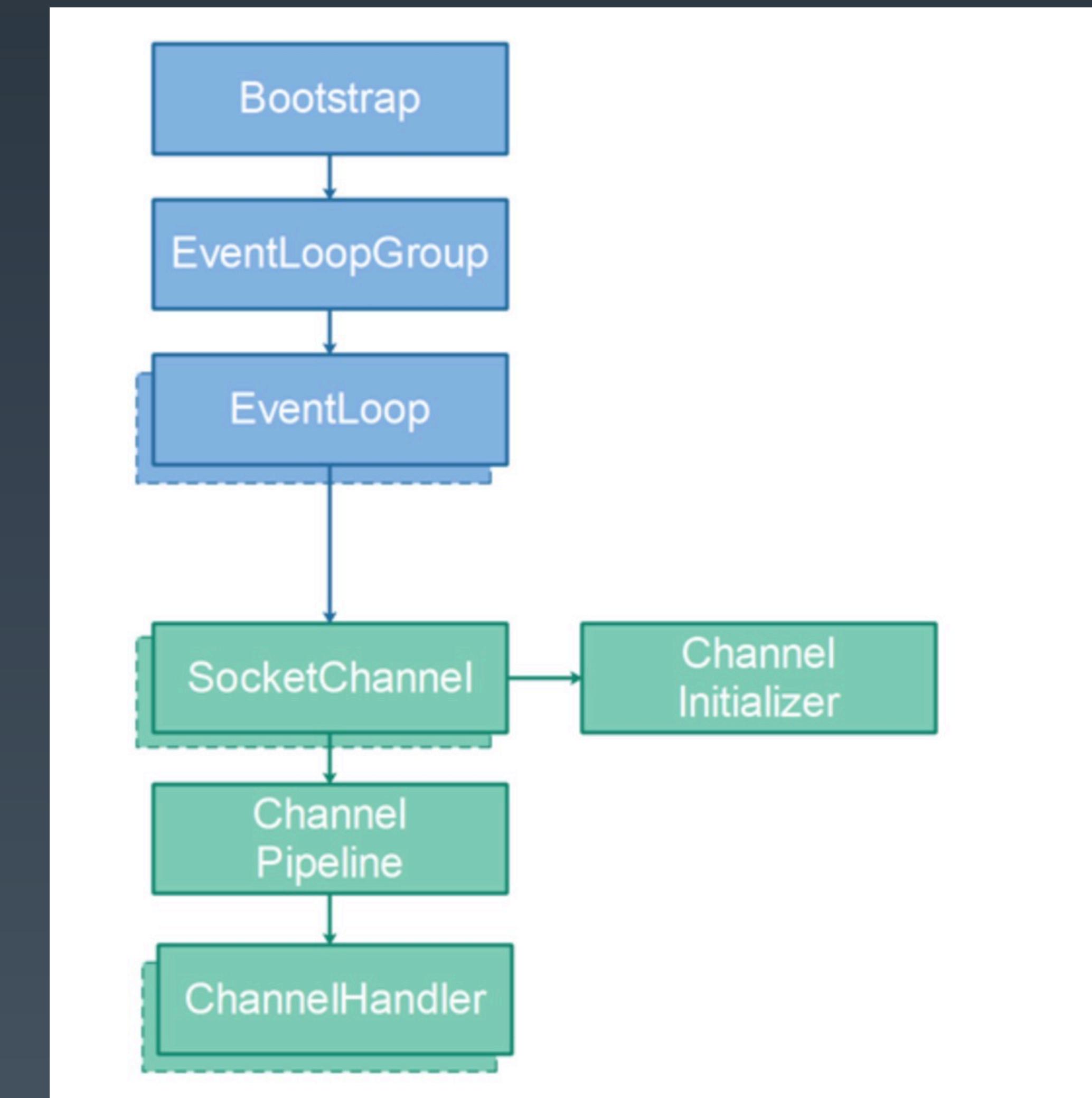
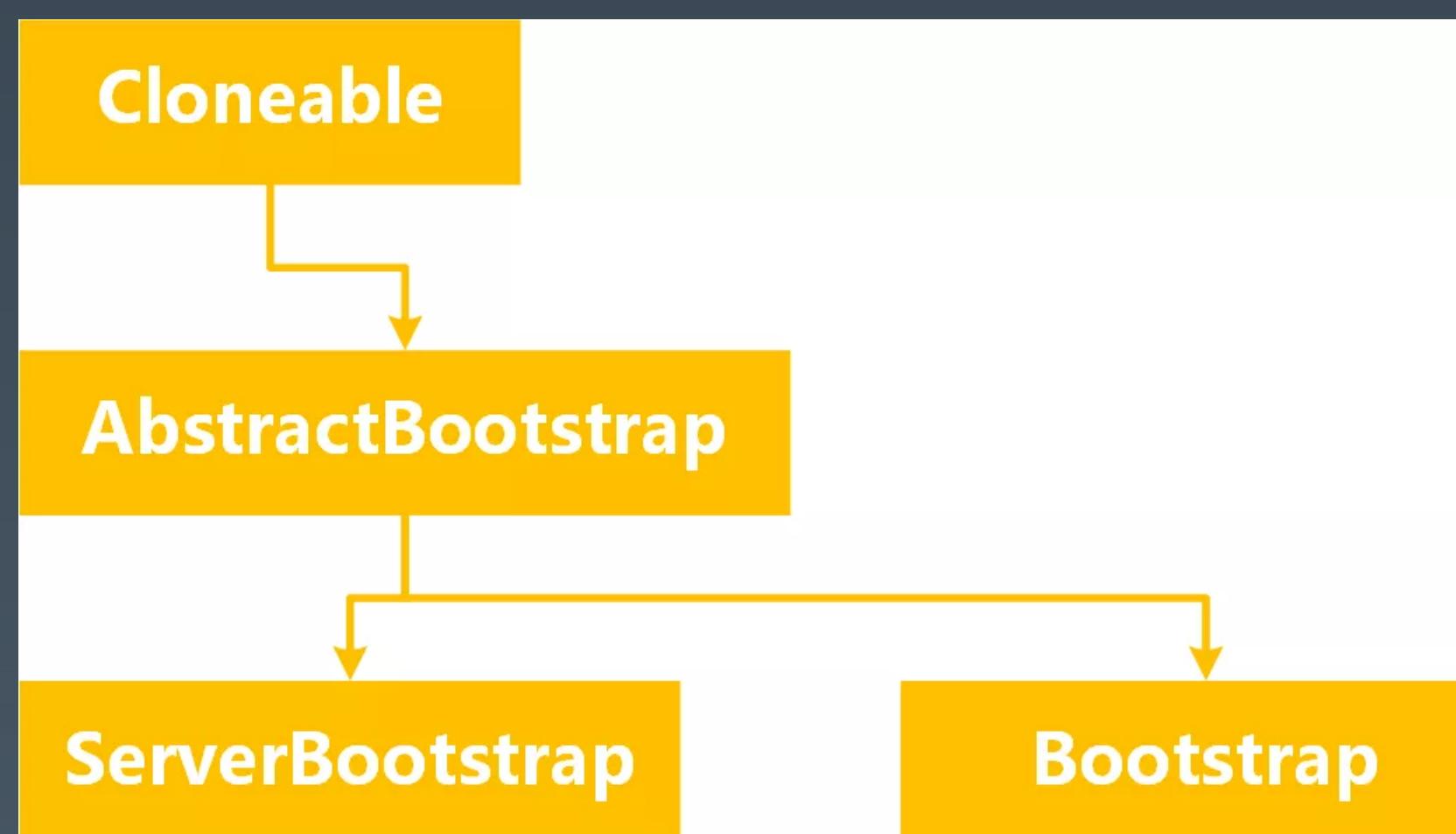
EventLoop

SocketChannel: 连接

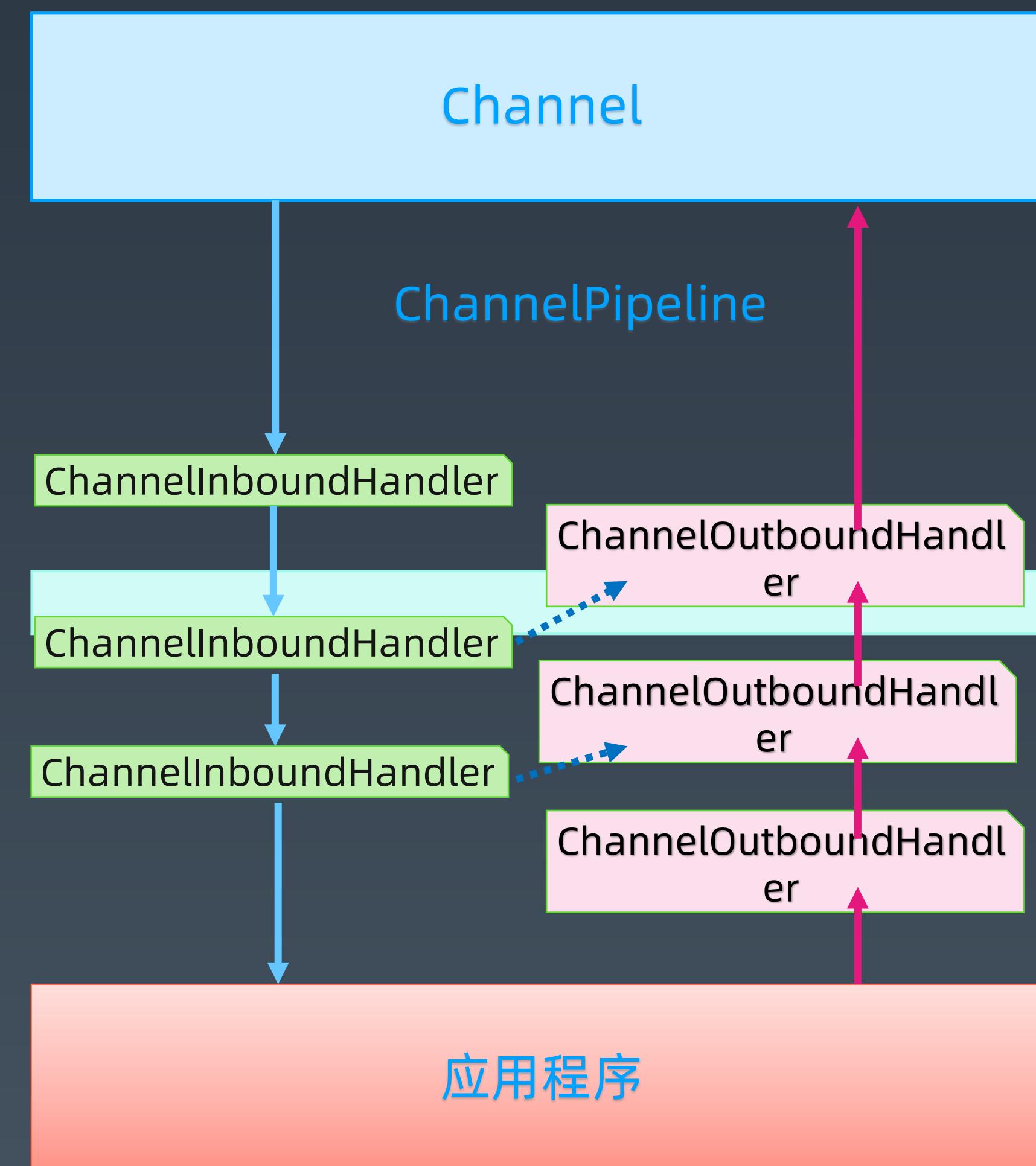
ChannelInitializer: 初始化

ChannelPipeline: 处理器链

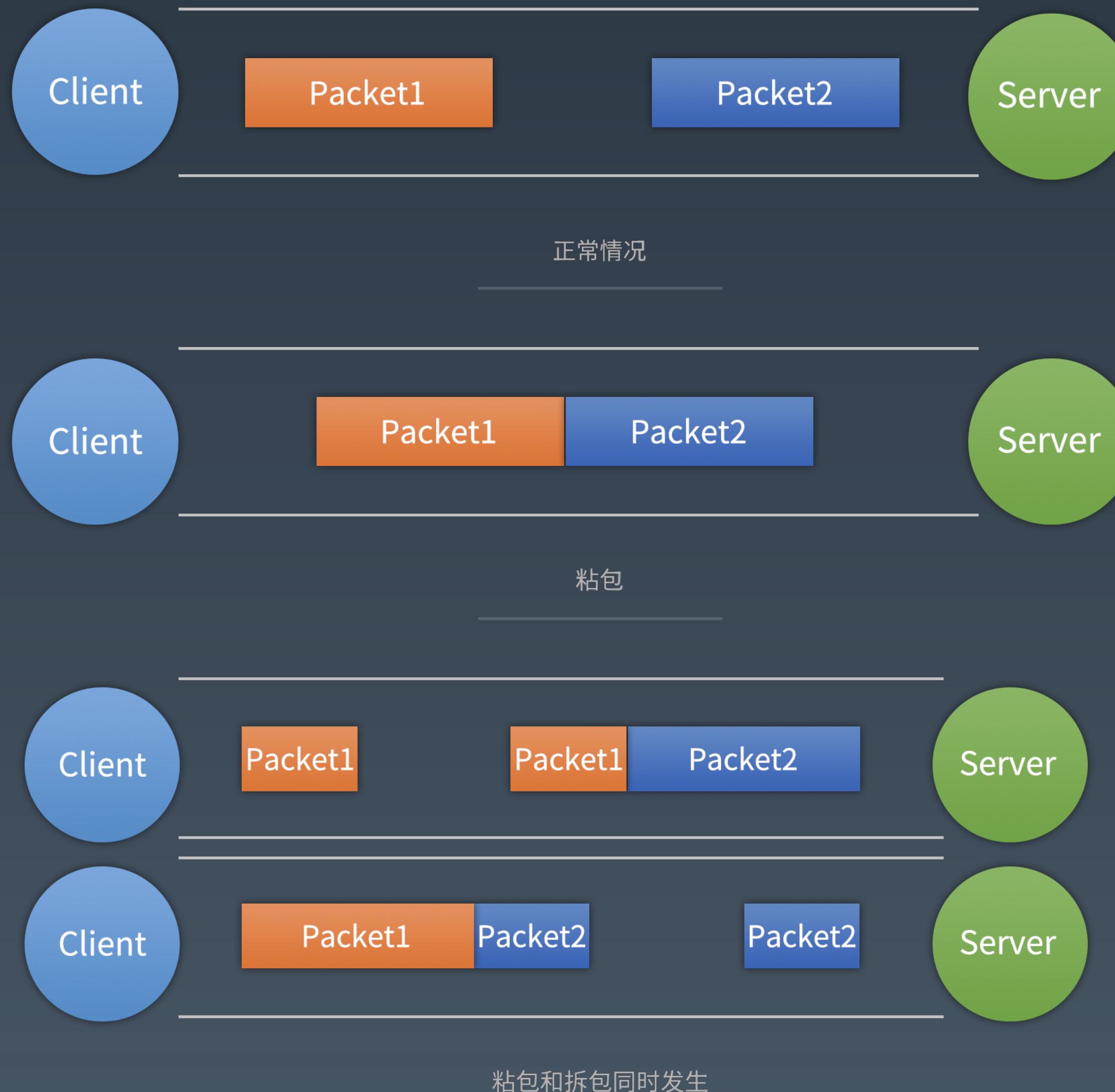
ChannelHandler: 处理器



ChannelPipeline



粘包与拆包



都是人为问题

ByteToMessageDecoder 提供的一些常见的实现类：

1. **FixedLengthFrameDecoder**: 定长协议解码器，我们可以指定固定的字节数算一个完整的报文
2. **LineBasedFrameDecoder**: 行分隔符解码器，遇到\n 或者\r\n，则认为是一个完整的报文
3. **DelimiterBasedFrameDecoder**: 分隔符解码器，分隔符可以自己指定
4. **LengthFieldBasedFrameDecoder**: 长度编码解码器，将报文划分为报文头/报文体
5. **JsonObjectDecoder**: json 格式解码器，当检测到匹配数量的“{”、“}”或“[”、“]”时，则认为是一个完整的 json 对象或者 json 数组

Nagle 与 TCP_NODELAY

网络拥堵与 Nagle 算法优化
TCP_NODELAY

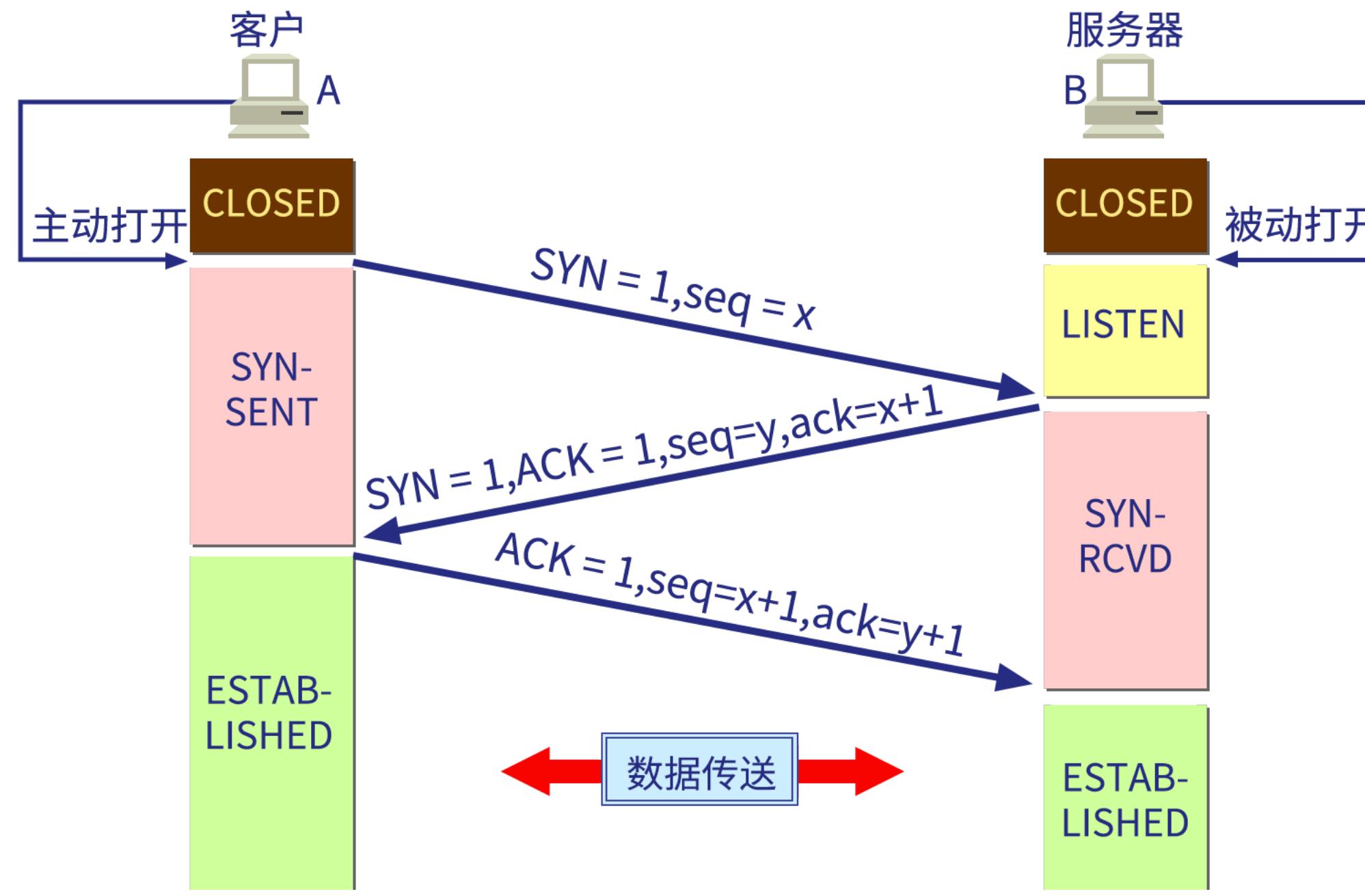
优化条件：
- 缓冲区满
- 达到超时

MTU: Maximum Transmission Unit 最大传输单元
MSS: Maximum Segment Size 最大分段大小, 为 $MTU - 20(IP) - 20(TCP)$

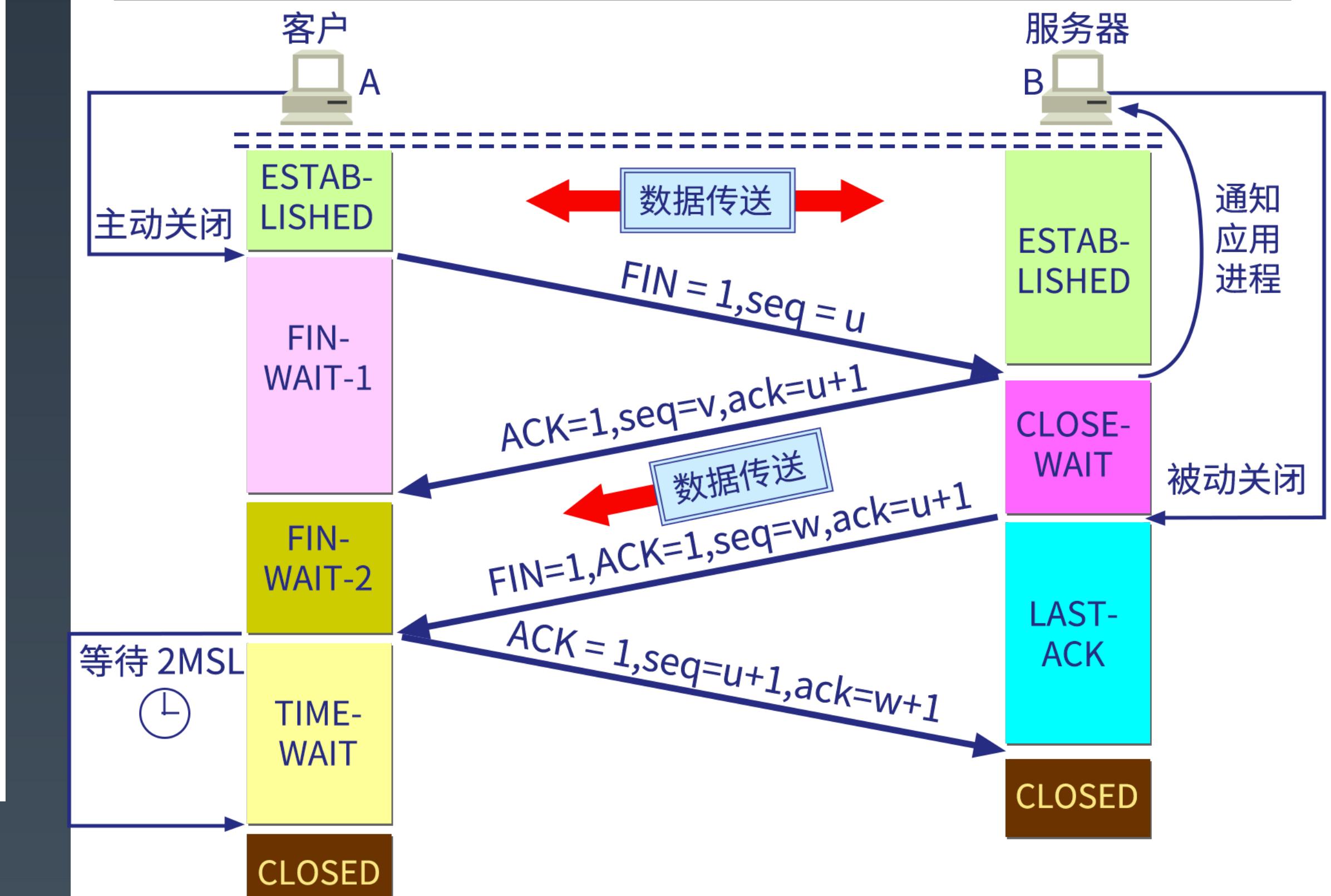
$MTU = 1500 \text{ Byte}$
 $MSS = 1460 \text{ Byte}$

连接优化

用三次握手建立TCP连接的各状态



TCP连接必须经过时间 2MSL后才真正释放掉。



注意 TCP 与 UDP 区别

Linux上MSL默认1分钟，

windows上默认为2分钟。

Netty 优化

1、不要阻塞 EventLoop

2、系统参数优化

ulimit -a, /proc/sys/net/ipv4/tcp_fin_timeout, TcpTimedWaitDelay

3、缓冲区优化

SO_RCVBUF/SO_SNDBUF/SO_BACKLOG/ REUSEXXX

4、心跳周期优化

心跳机制与短线重连

5、内存与 ByteBuffer 优化

DirectBuffer 与 HeapBuffer

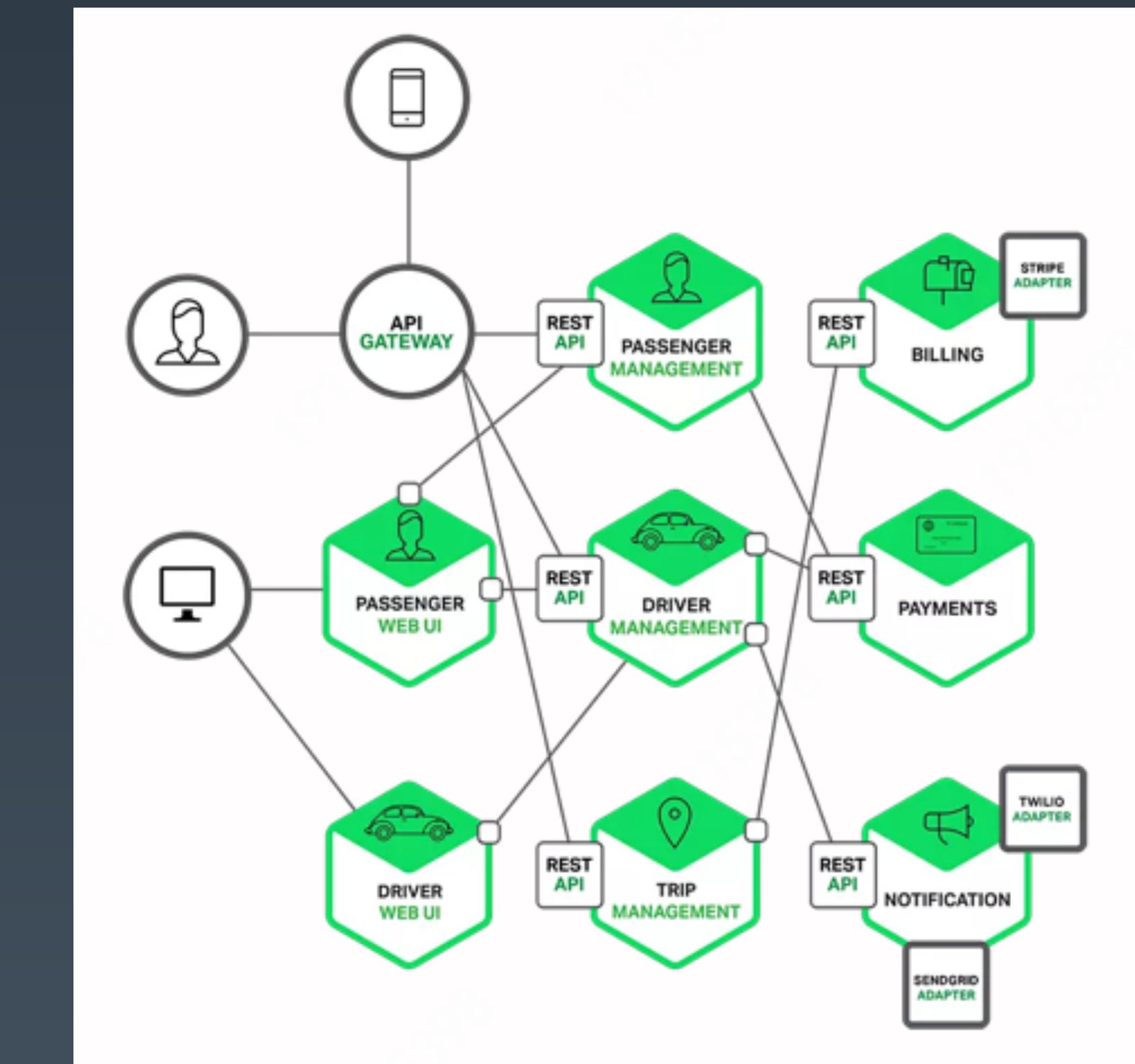
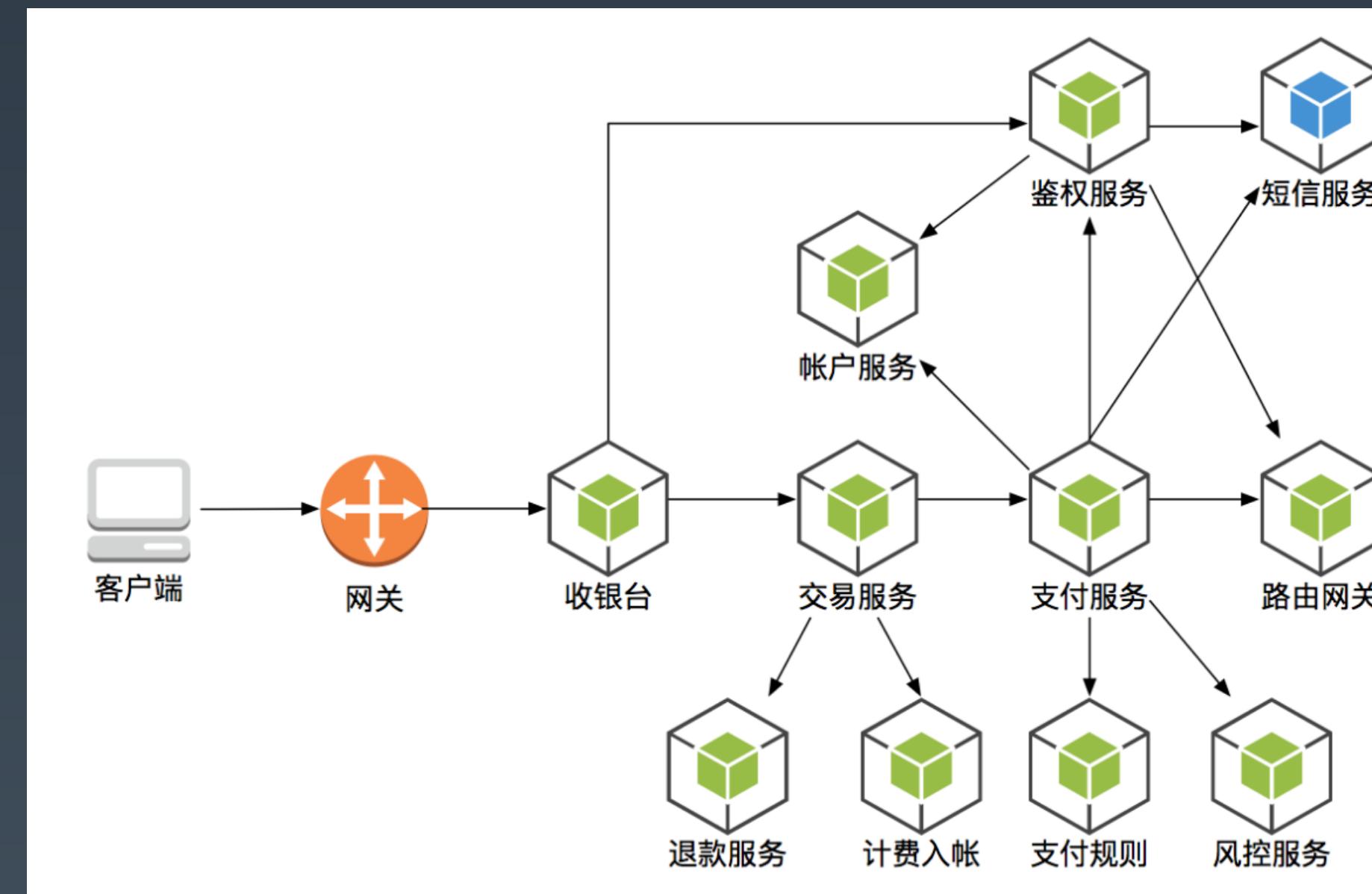
6、其他优化

- ioRatio
- Watermark
- TrafficShaping

4. 典型应用：API 网关

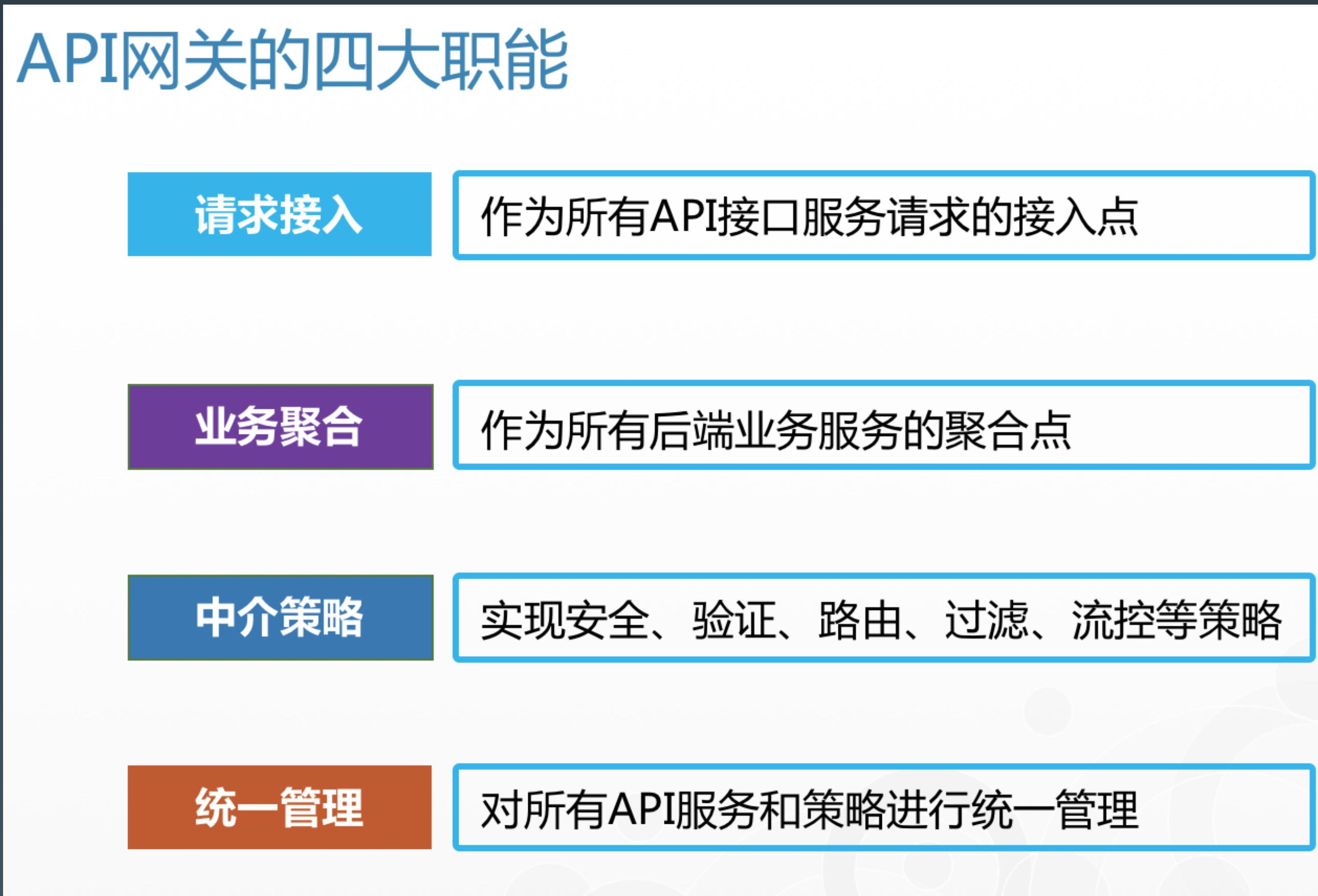
典型应用：API 网关

网关的结构和功能？



典型应用：API 网关

网关的结构和功能？

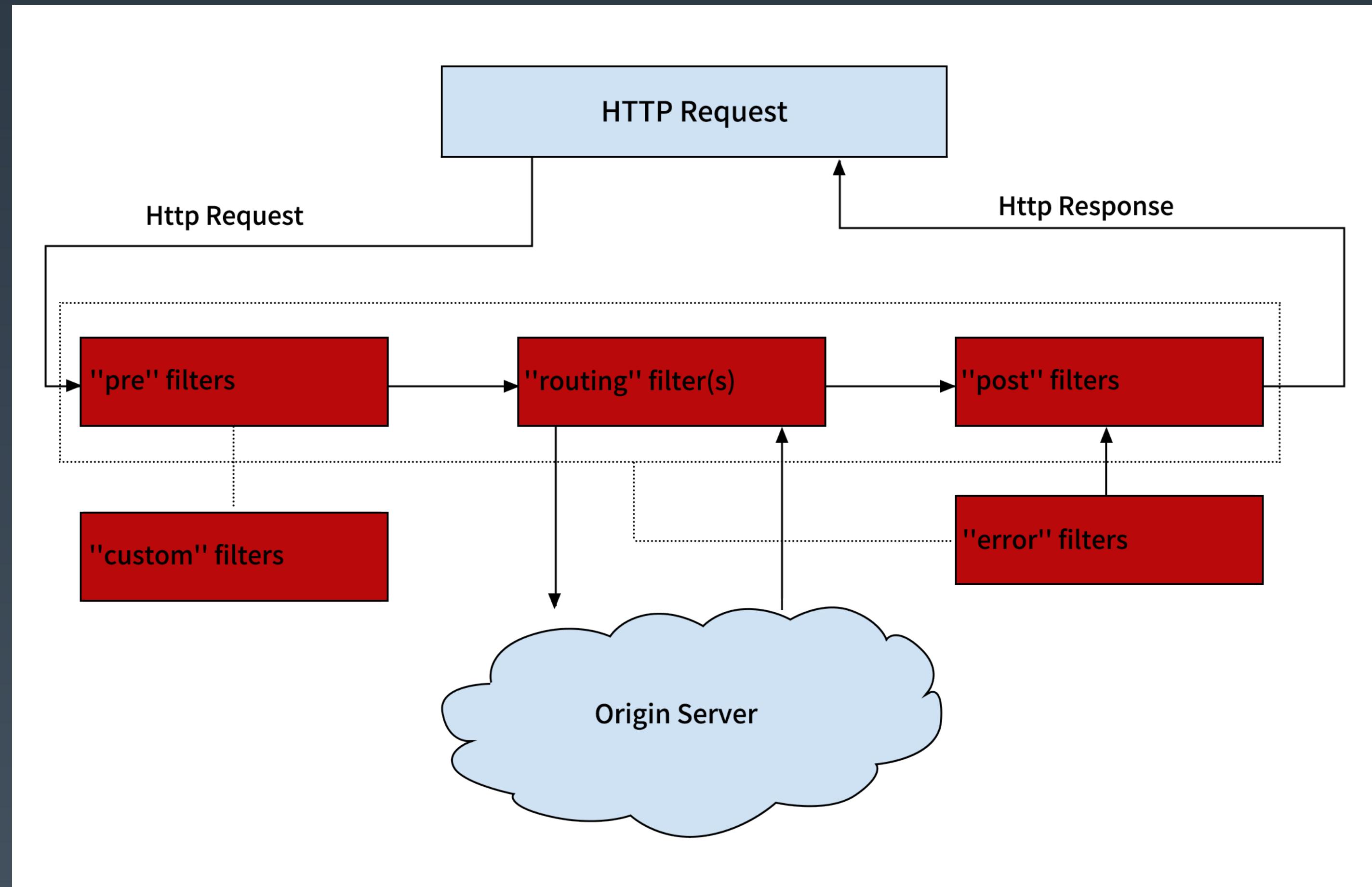


典型应用：API 网关

网关的分类



典型应用：API 网关

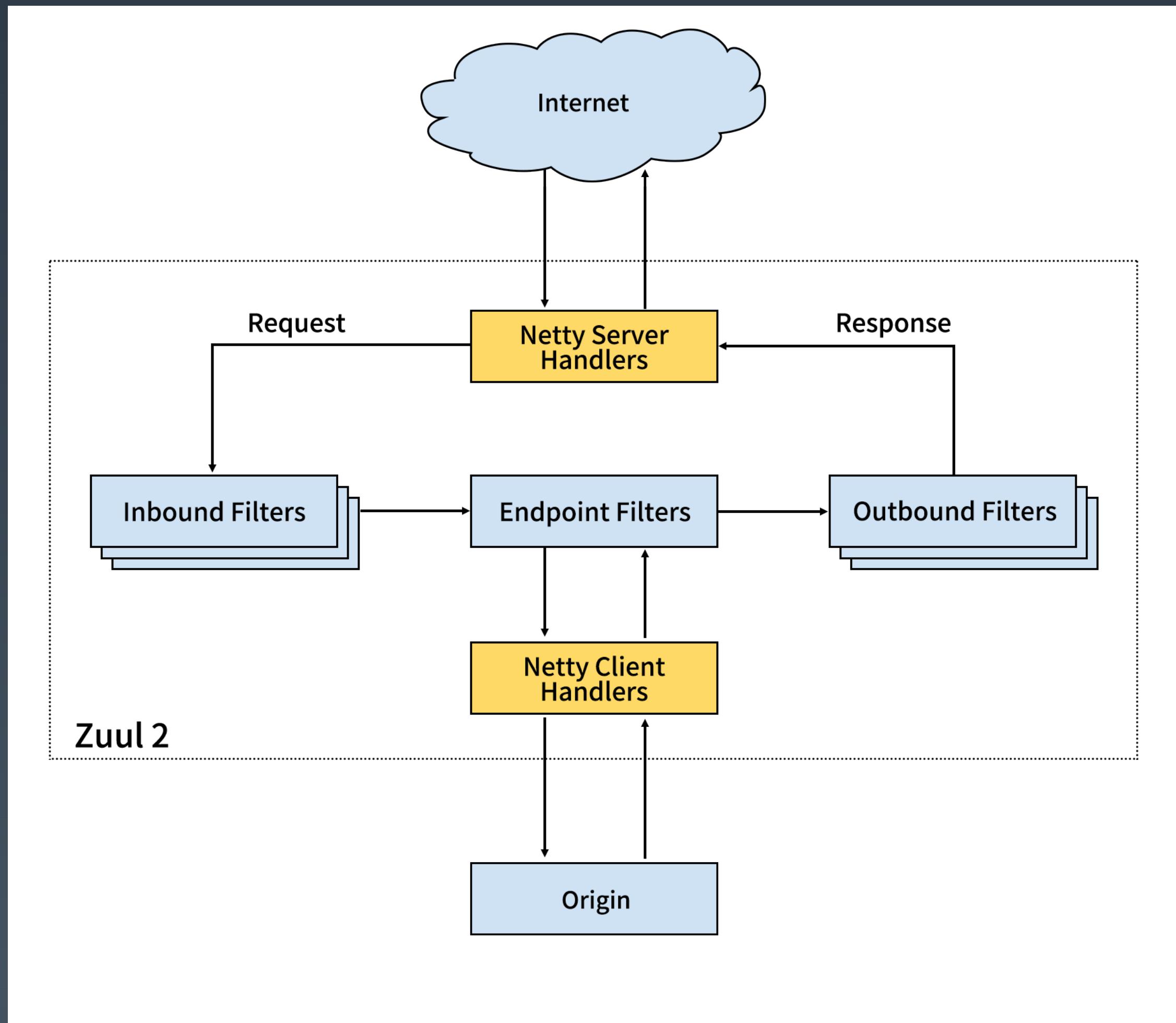


Zuul 是 Netflix 开源的 API 网关系统，它的主要设计目标是动态路由、监控、弹性、和安全。

Zuul 的内部原理可以简单看做是很多不同功能 filter 的集合，最主要的就是 pre、routing、post 这三种过滤器，分别作用于调用业务服务 API 之前的请求处理、直接响应、调用业务服务 API 之后的响应处理。

典型应用：API 网关

Zuul2



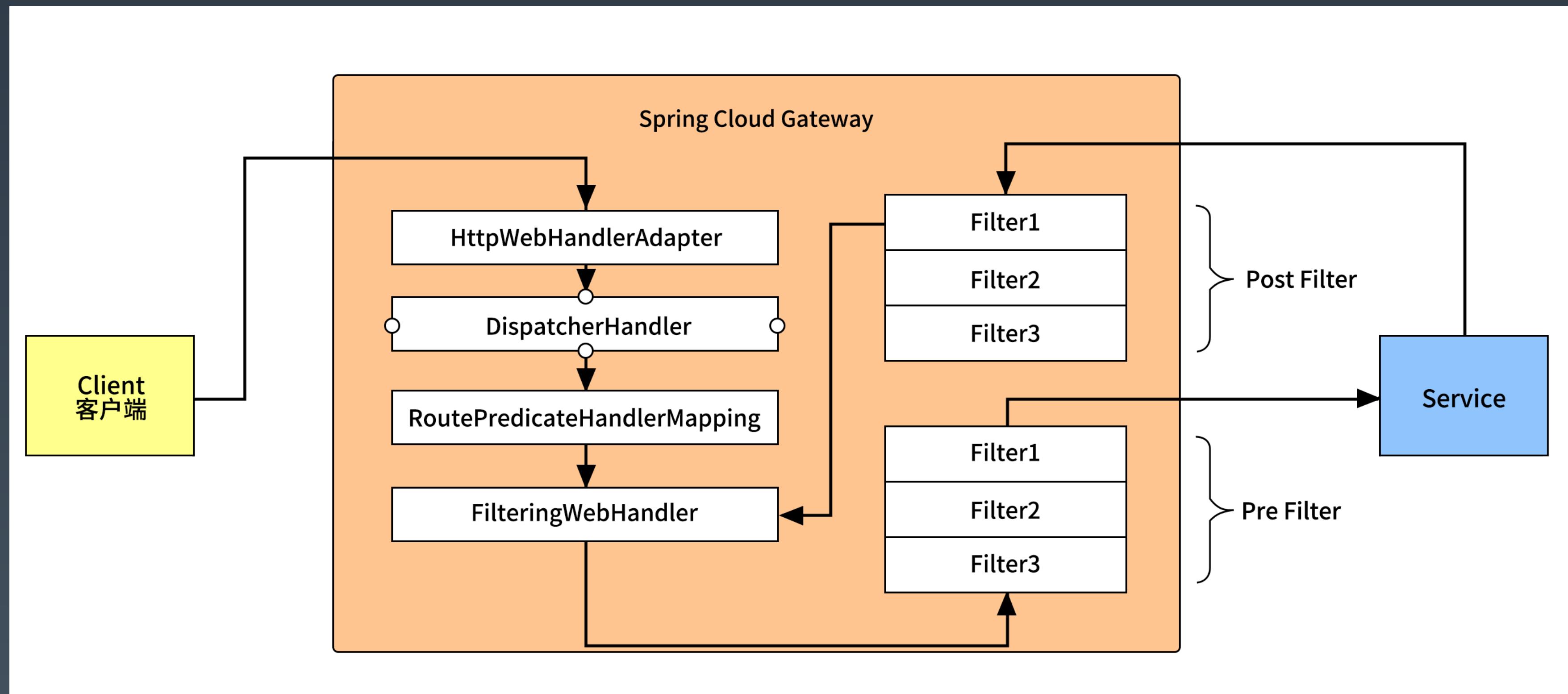
Zuul 2.x 是基于 Netty 内核重构的版本。

核心功能：

1. Service Discovery
2. Load Balancing
3. Connection Pooling
4. Status Categories
5. Retries
6. Request Passport
7. Request Attempts
8. Origin Concurrency Protection
9. HTTP/2
10. Mutual TLS
11. Proxy Protocol
12. GZip
13. WebSockets

典型应用：API 网关

Spring Cloud Gateway



典型应用：API 网关

网关对比



性能非常好，适合流量网关



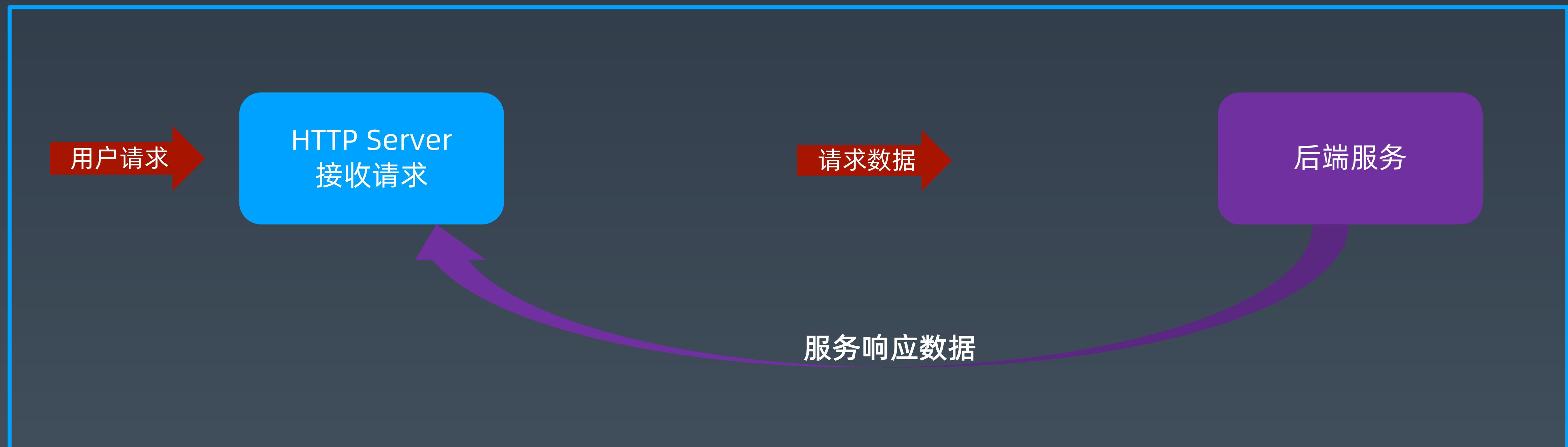
扩展性好，适合业务网关，二次开发



5. 自己动手实现 API 网关

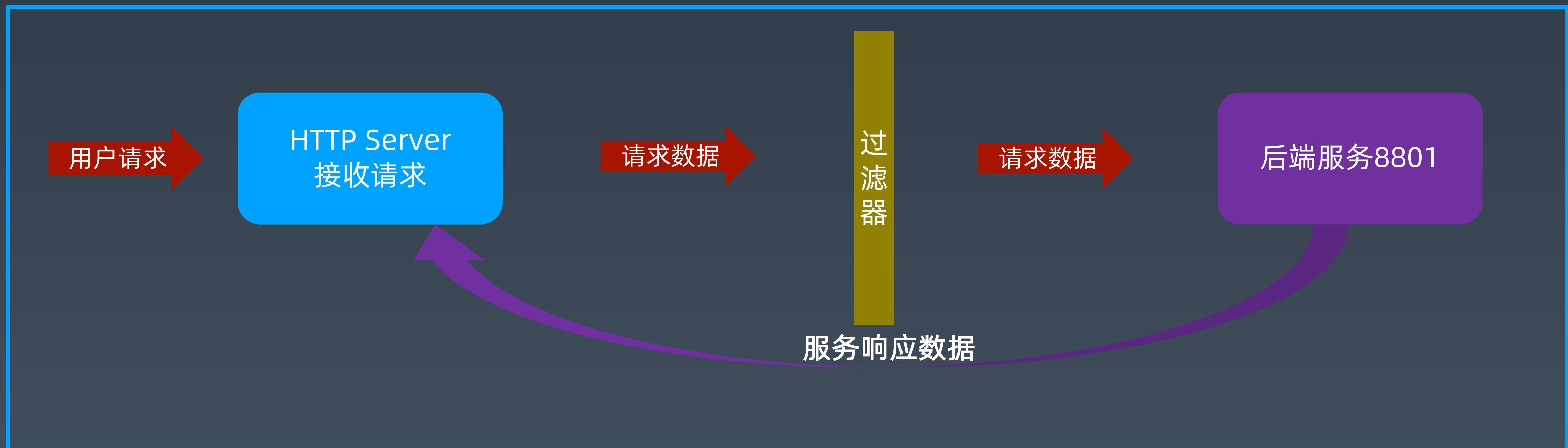
自己动手实现 API 网关

最简单的网关--gateway 1.0



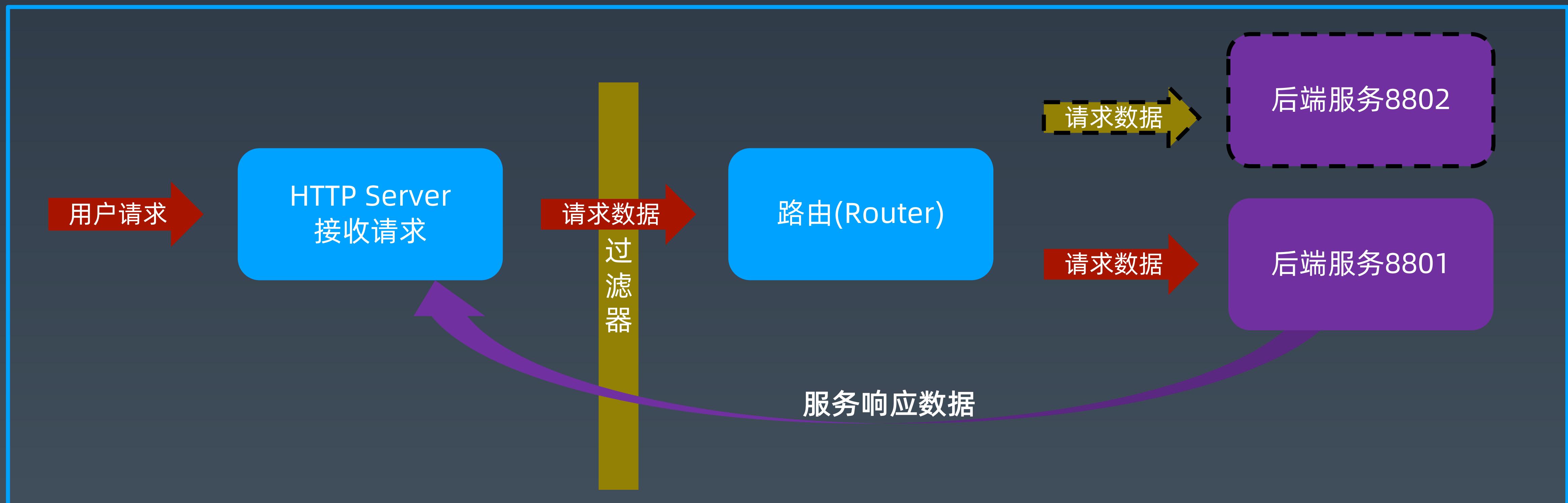
自己动手实现 API 网关

最简单的网关--gateway 2.0



自己动手实现 API 网关

最简单的网关--gateway 3.0



自己动手实现 API 网关

架构设计

设计：技术复杂度与业务复杂度

抽象：概念理清、正确命名

组合：组件之间的相互关系

第5节课总结回顾

再谈谈什么是高性能

Netty 如何实现高性能

Netty 网络程序优化

典型应用：API 网关

自己动手实现 API 网关

第5节课作业实践

1、按今天的课程要求，实现一个网关，

基础代码可以 fork: <https://github.com/kimmking/JavaCourseCodes>

02nio/nio02 文件夹下

实现以后，代码提交到 Github。

1) 周三作业：（必做）整合你上次作业的httpClient/okhttp；

2) 周三作业（可选）：使用netty实现后端http访问（代替上一步骤）；

3) 周日作业：（必做）实现过滤器 ~

4) 周日作业（可选）：实现路由

THANKS! |  极客大学