

QCon⁺ 案例研习社

海量长连接消息推送系统实践

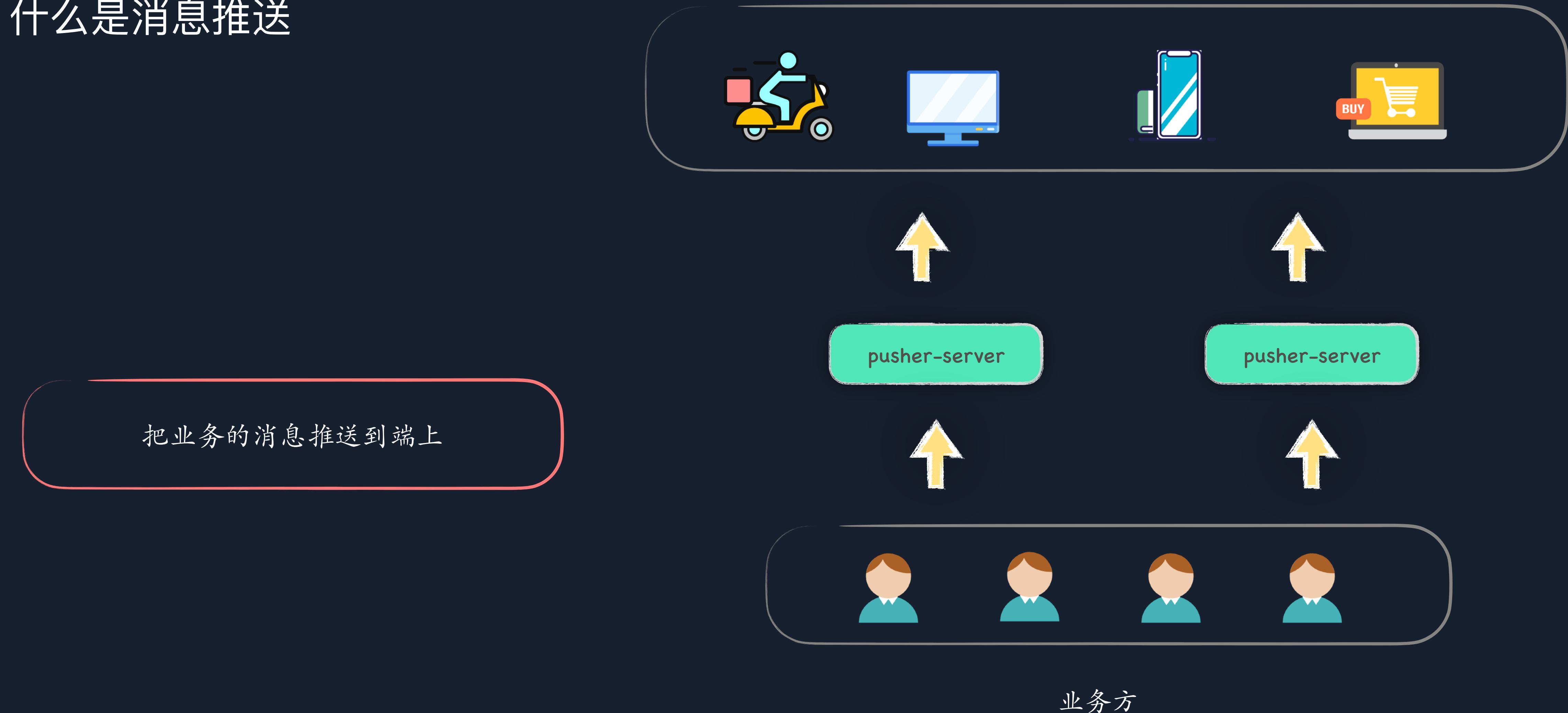
芮峰云 rfyiamcool

目录

- 1 长连接推送**
- 2 架构设计**
- 3 性能优化**
- 4 深度优化**
- 5 总结**

长连接推送

什么是消息推送

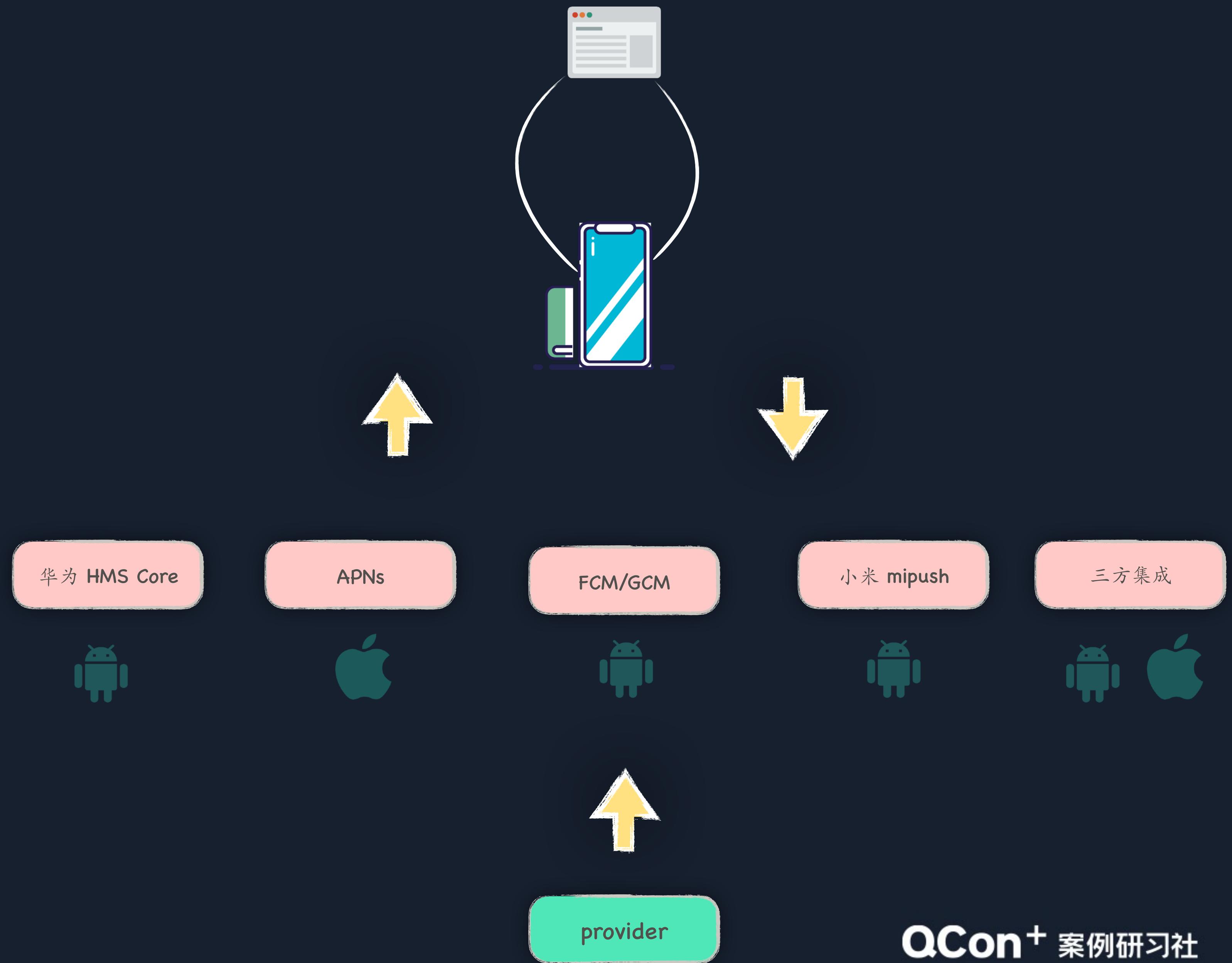


长连接推送

厂商系统推送

- * 缺点

- * 限制繁多
- * 功能计费
- * 更适合做离线弹窗提醒
- * 消息及时性不满足
- * 消息可靠性不满足
- *



长连接推送

难点挑战

- * 当前量级
 - * 超百万的长连接接入
 - * 每天近 100 亿条消息
 - * 高峰值时每秒 70w+ 条消息
- * 目标
 - * 如何尽量保证消息的低延迟
 - * 如何保证系统的高并发
 - * 如何保证消息的可靠性
 - * 如何保证系统的可用性
- *



架构设计

选型

- * 语言

- * golang

- * 中间件

- * kafka

- * redis



- * 接入协议

- * websocket

- * 兼容性很好

- * 前端很友好

- * quic

- * 弱网络友好



- * 序列化协议

- * json

- * protobuf

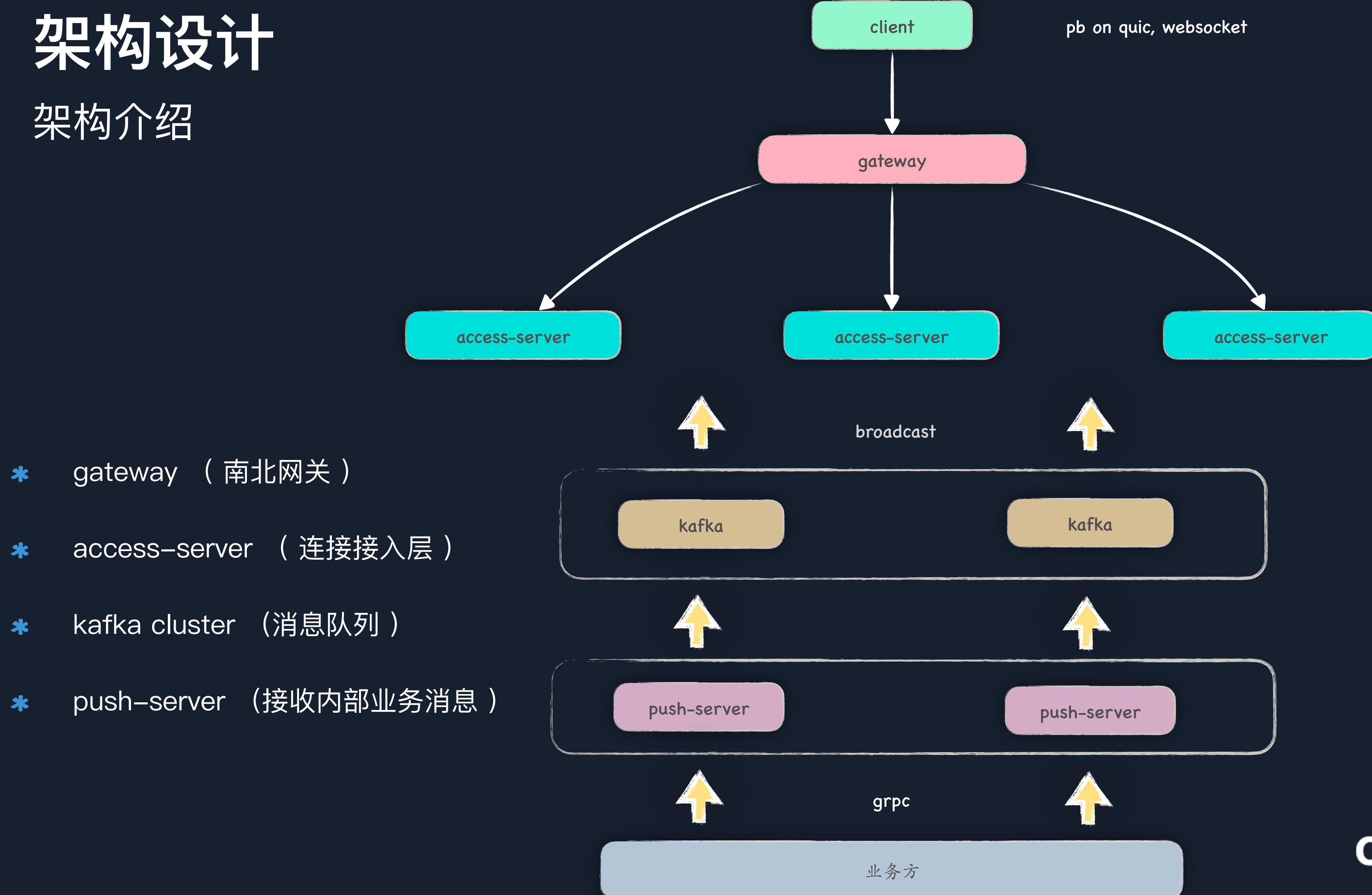
- * 更快

- * 更省内存

- * 更规范

架构设计

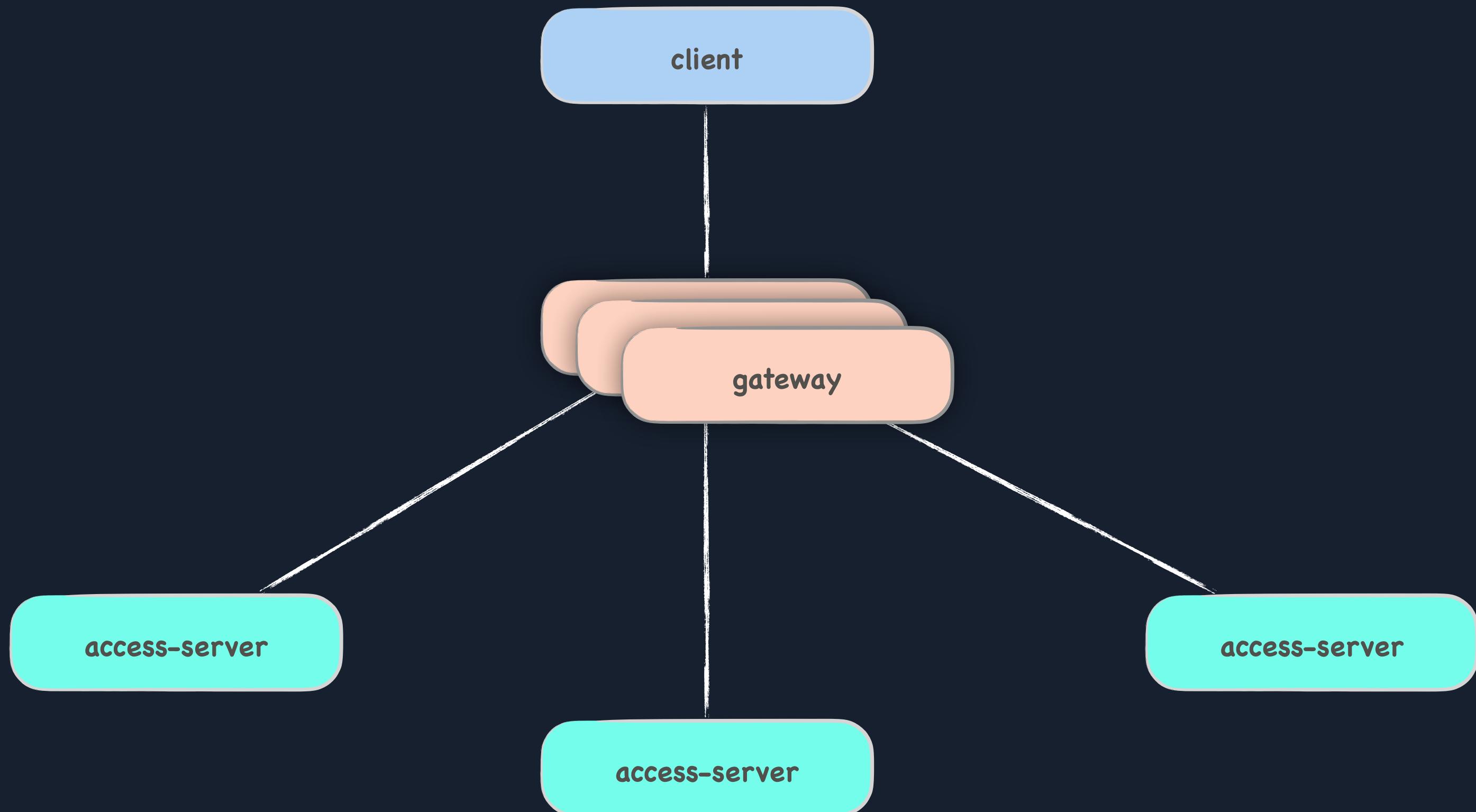
架构介绍



架构设计

gateway

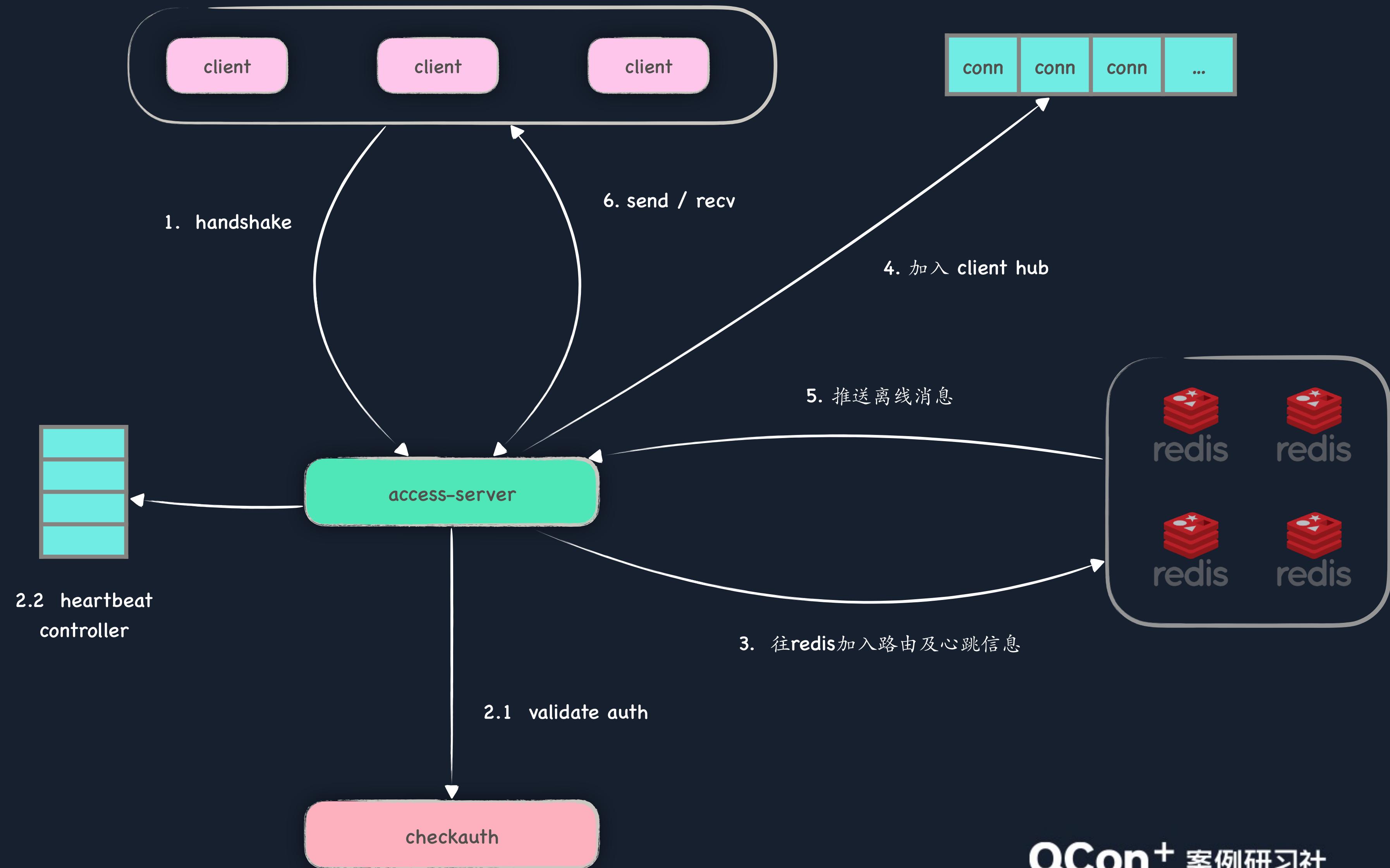
- * 南北流量总入口
- * 流量调度
- * 应用策略
- * 异地多活
- * TLS 卸载



架构设计

access-server

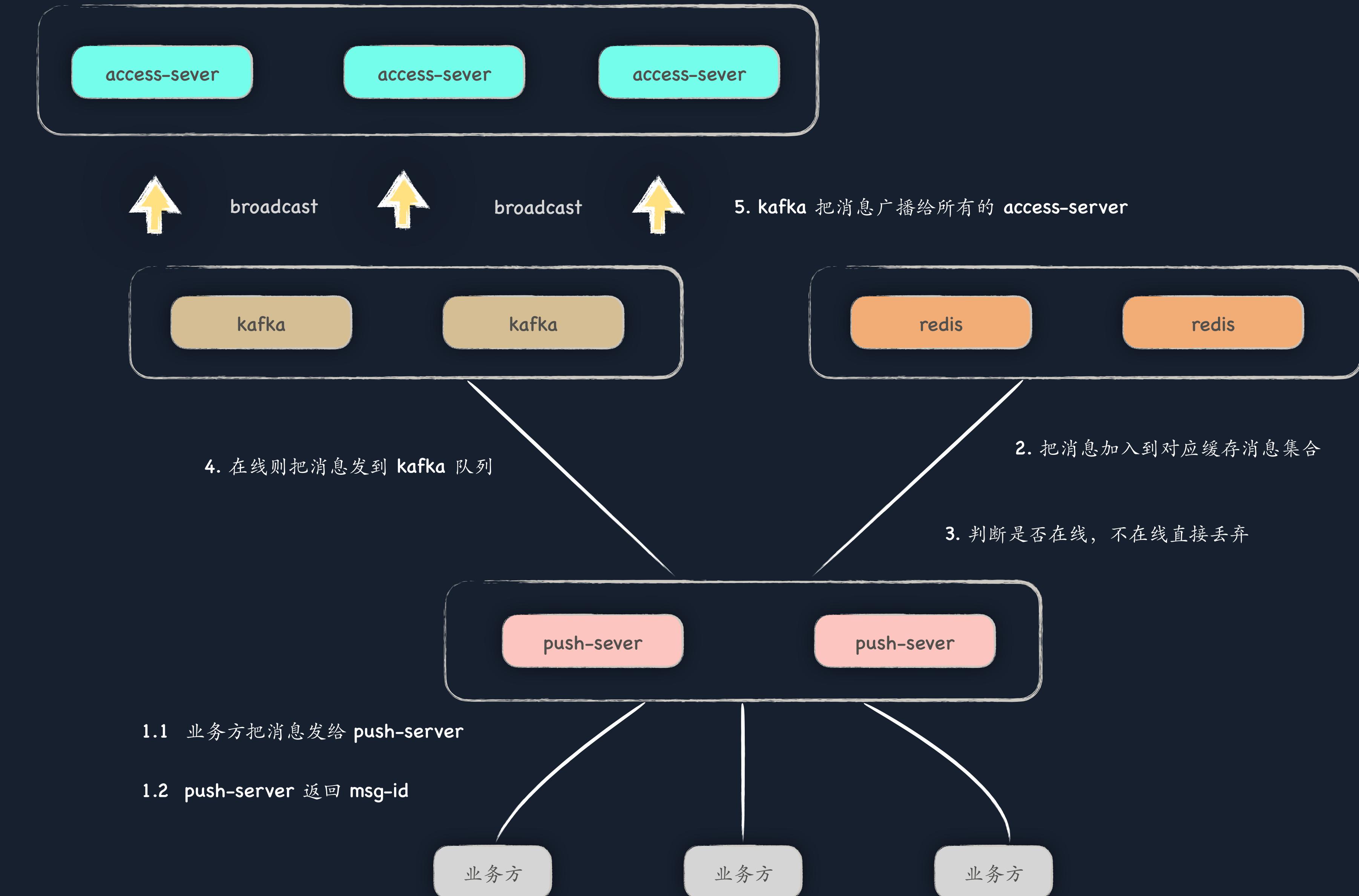
- * 多协议客户端接入
- * 握手包进行鉴权
- * 消息下发及ack确认
- * 连接心跳管理
- * 注册客户端信息到redis
- * 从redis拉取离线消息
- * 用户长连接管理
- * 未确认消息延迟重发
- * ...



架构设计

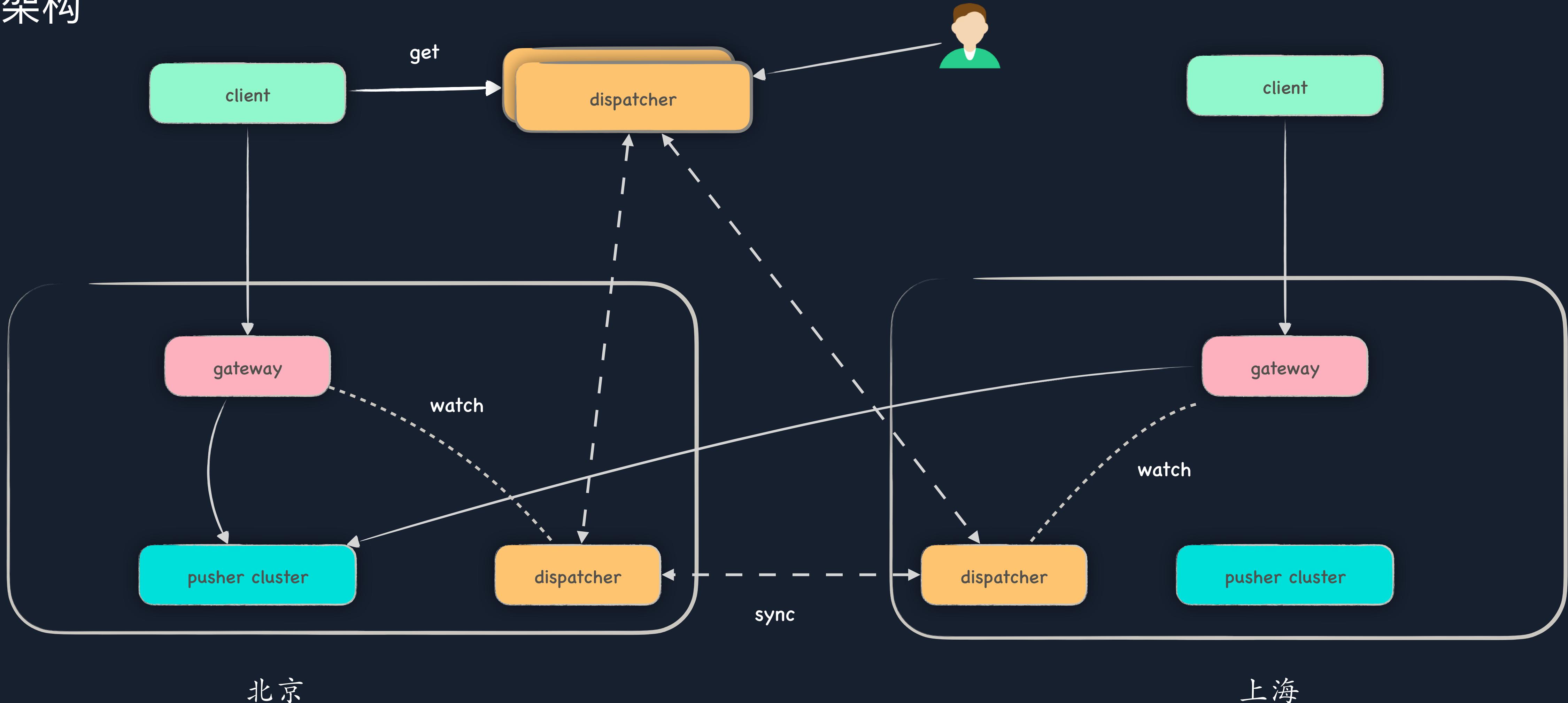
push-server

- * 接口鉴权
- * 格式检验
- * 为业务消息加入唯一ID
- * 先把消息加入缓存列表
- * 判断是否最近在线
- * 在线
 - * 把消息发给 kafka 队列
- * 不在线
 - * 走离线逻辑



架构设计

多活架构



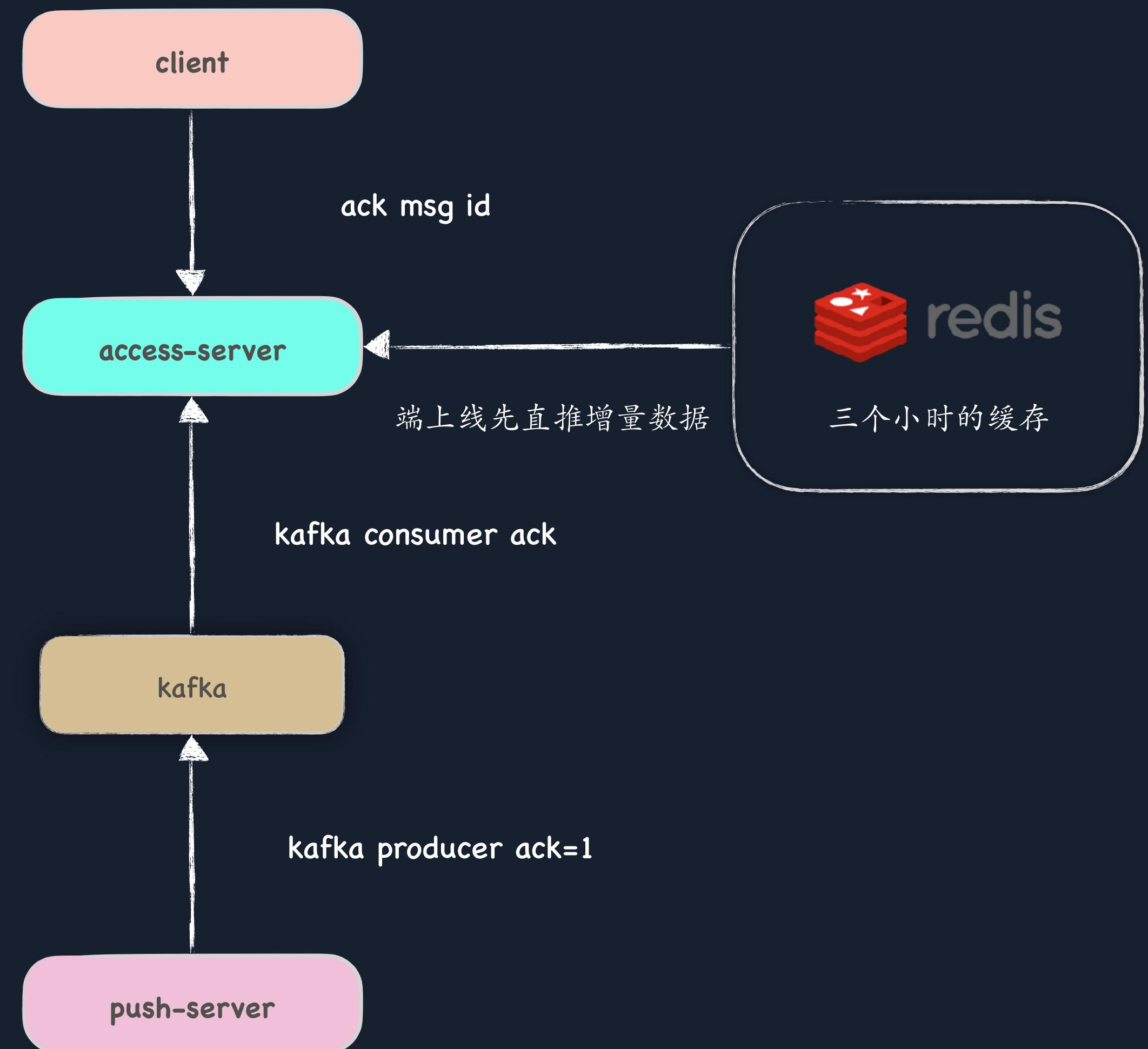
北京

上海

架构设计

消息的可靠性

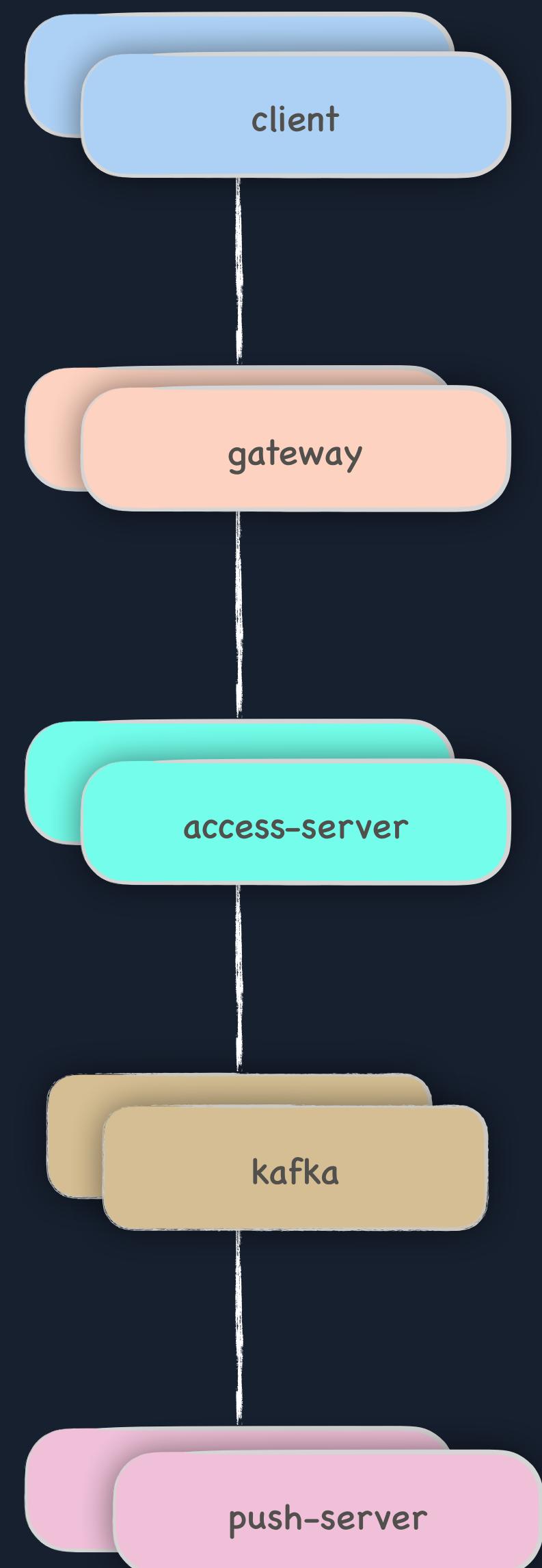
- * 离线消息机制
 - * 客户端优先拉取离线消息
- * 消息的 ack 确认机制
 - * 只要服务端没收到 ack, 那么就发起重传
- * 安全的操作 kafka
 - * producer 折中选择 ack=1
 - * consumer 手动 ack 模式



架构设计

系统高可用

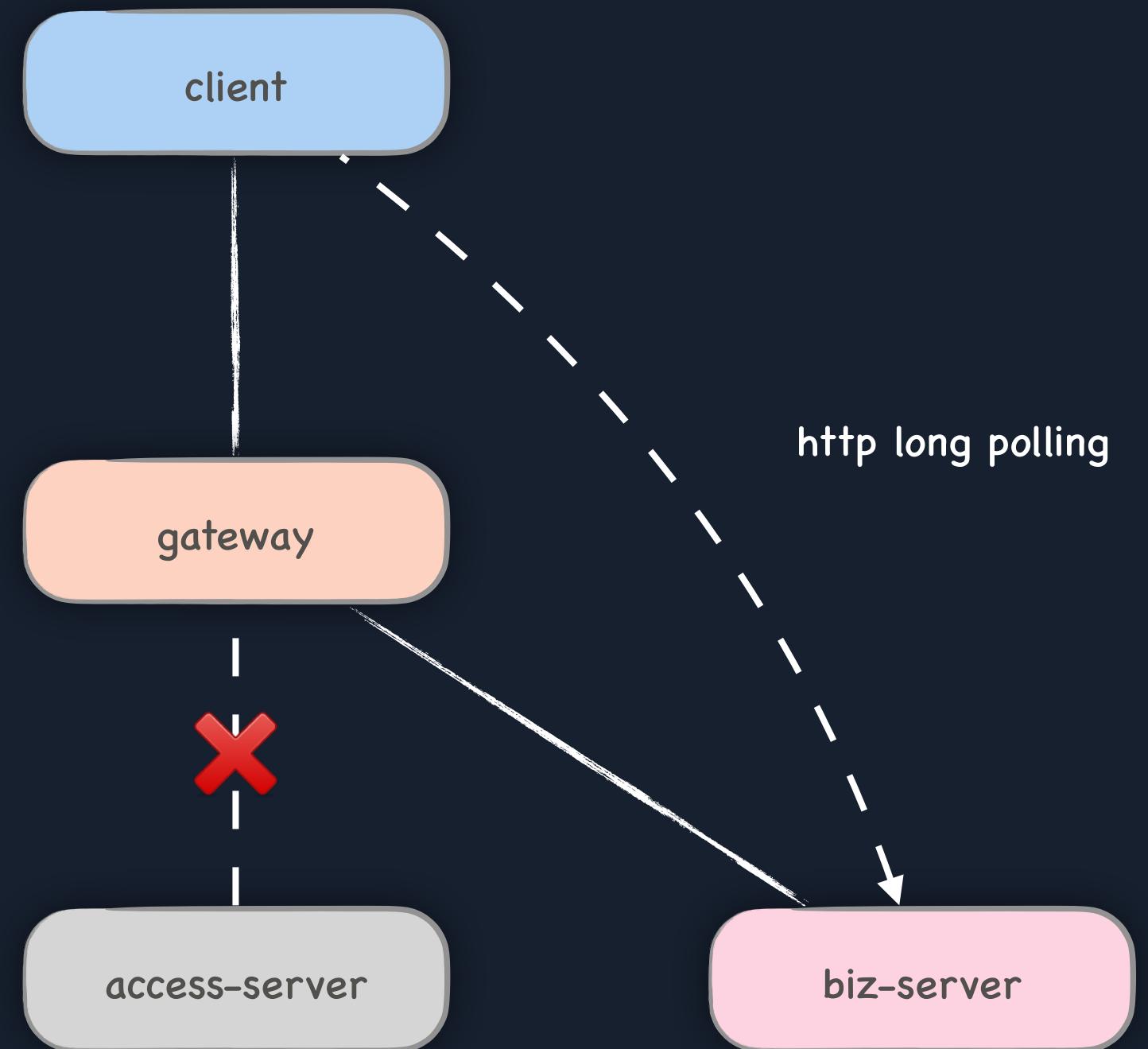
- * gateway 可横向扩展, 通过 dns/dispatcher 轮询
- * access-server 可以横向扩展
- * push-server 可以横向扩展
- * 当 redis cluster 异常时, 所有条件都为 true !!!
- * 当 kafka cluster 异常时, 直推给所有 access-server



架构设计

客户端

- * 客户端握手时需要声明业务类型、APP类型、版本、套件等
 - * 客户端需要解决幂等、难见的乱序问题
 - * 客户端如连不通 pusher，则走业务方长短轮询接
 - * 探测线路质量，弱网走 quic 协议
-
- * 支持自定义压缩
 - * 减少客户端的流量
 - * lz4 > zstd > gzip
 - * 通常在 10 倍的压缩率



架构设计

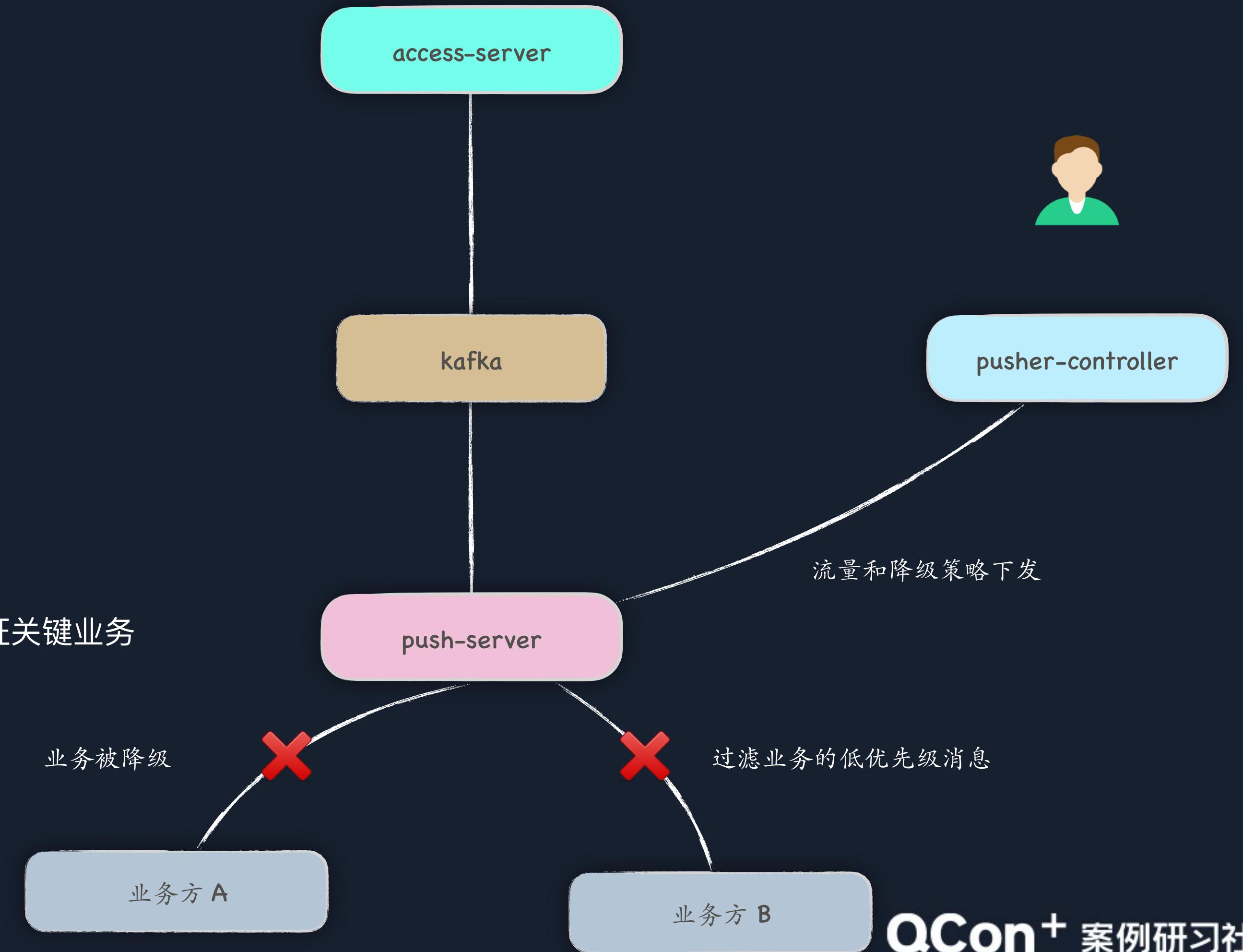
流量管理和降级

- * 流量管理的几种策略

- * 根据业务字段进行过滤
- * 根据业务内部的消息等级进行过滤
- * 根据业务的优先级进行过滤

- * 服务降级处理

- * 当出现流量过大时，对低优先级业务进行降级，需保证关键业务
- * 通知被降级的客户端做兜底处理



性能调优

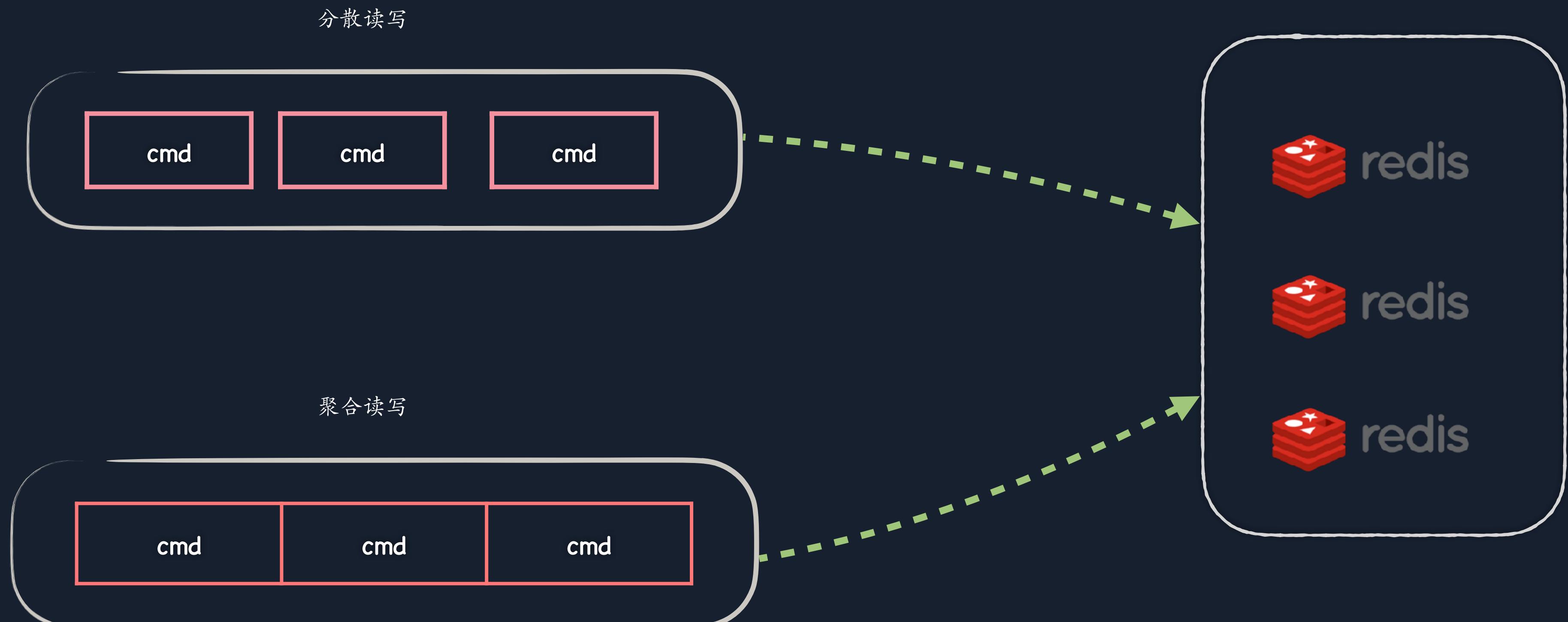
提高 redis 吞吐

* 不使用管道

* qps 13w

* 使用管道

* qps 50w +

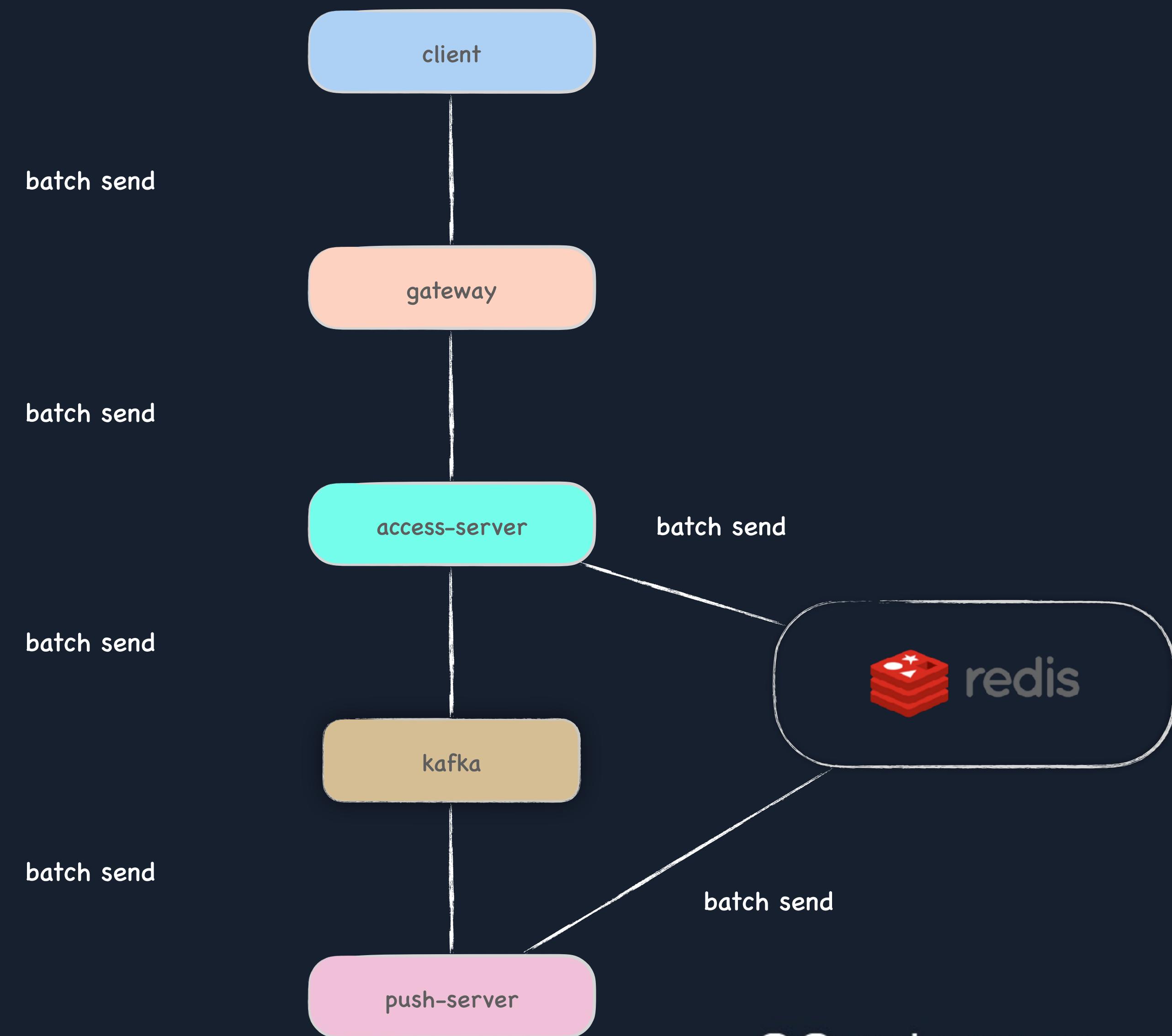


性能调优

批量写入

- * 批量写入的优点
 - * 减少 syscall
 - * 减少软硬中断
 - * 减少网络 RTT
 - * 长文本压缩效果更好 (1/10)

只有触发阈值才启用批量 !



性能调优

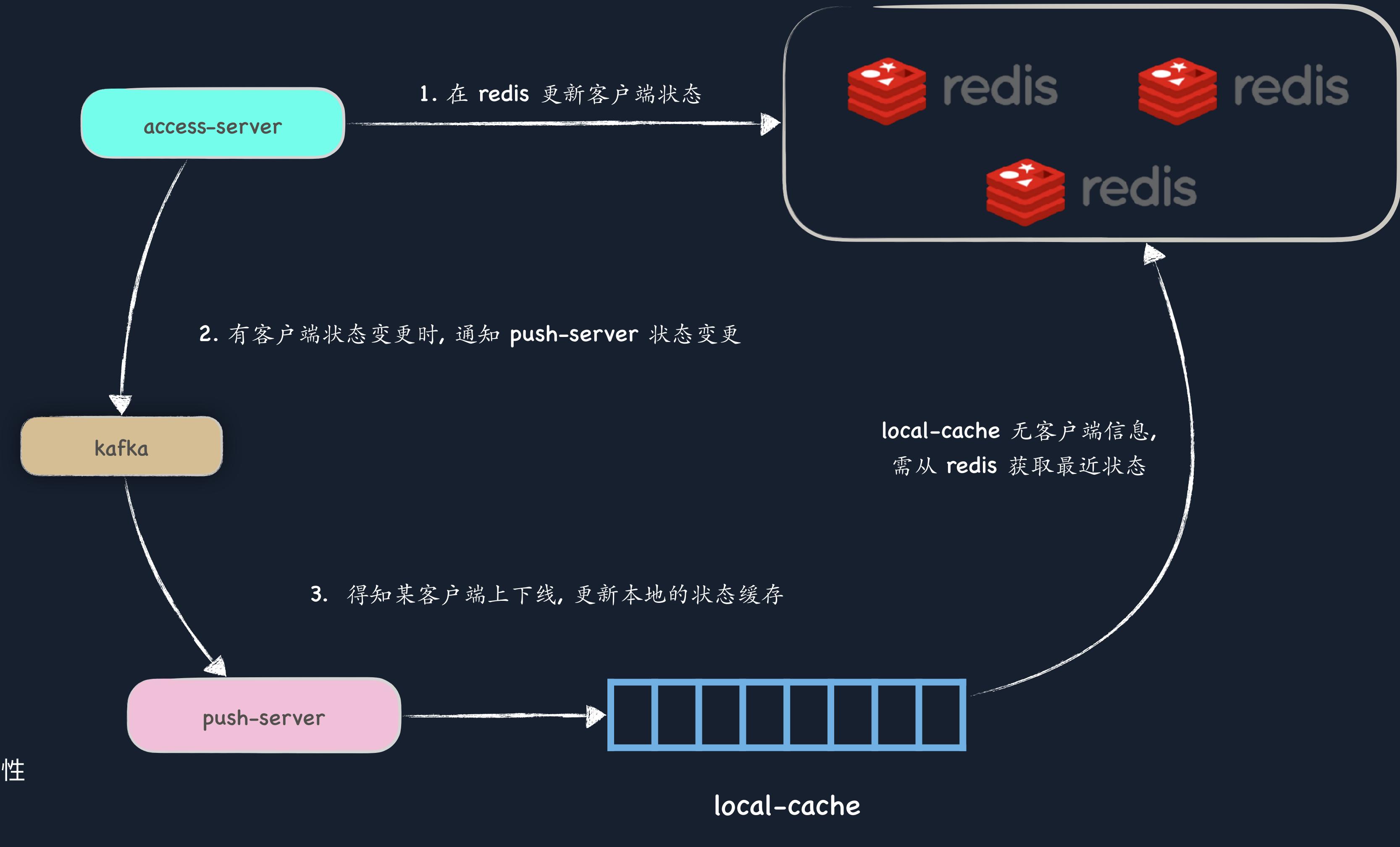
缓存优化

* 优化目的

- * 减少时延
- * 减少系统调用开销
- * 减少 redis 开销

* 做法

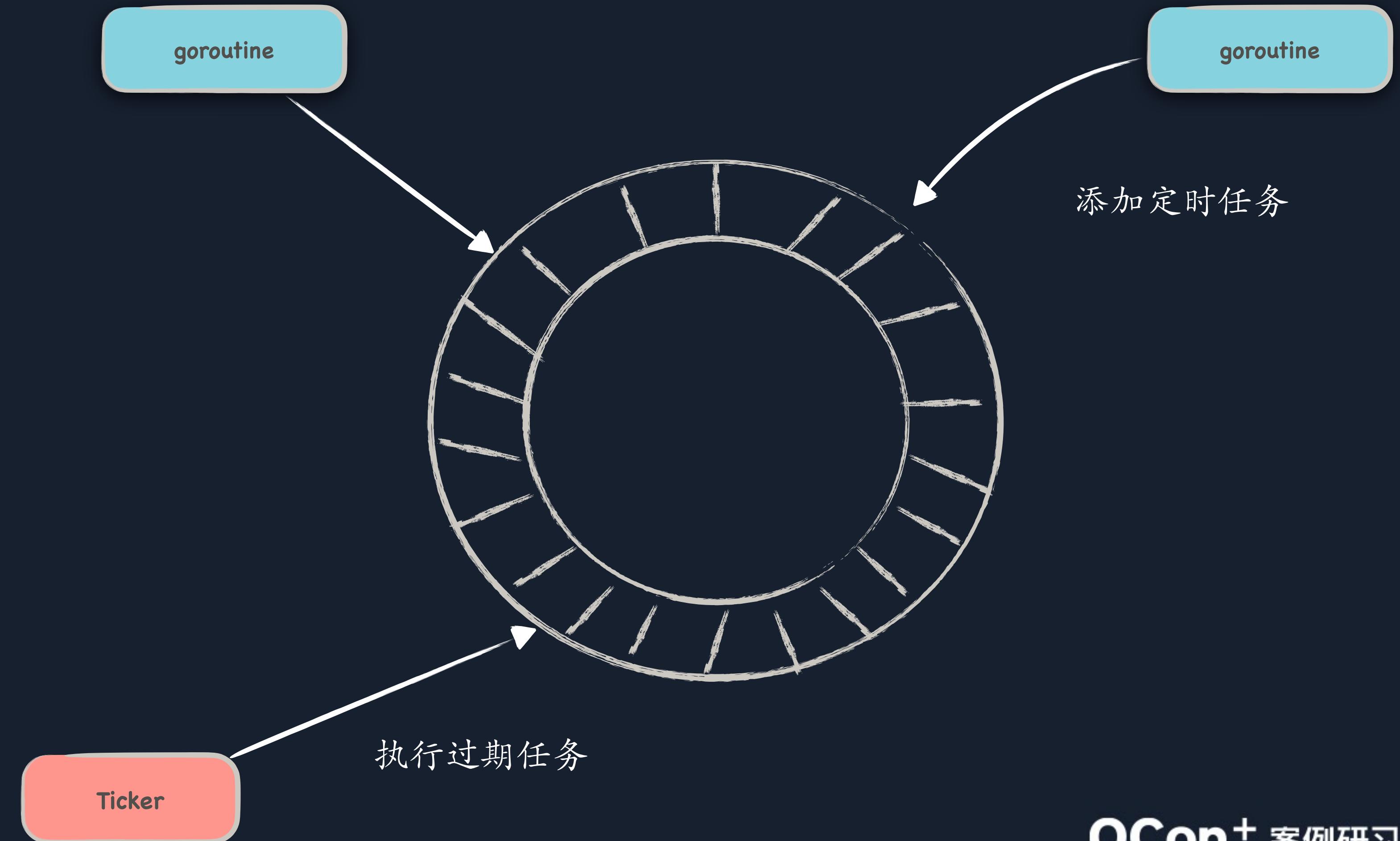
- * 优先从本地缓存中获取客户端状态
- * 无缓存, 再从 redis 获取状态
- * 再反向本地更新缓存
- * 兜底优化
 - * 通过 cache ttl 和 kafka 通知实现最终一致性
 - * 如 redis 异常, 直接往 kafka 里怼数据
 - * 状态不明时, 做法同上



性能调优

定时器优化

- * 大量的定时器场景
 - * 聚合批量写入
 - * 主动心跳检测
- * 时间轮定时器的优点
 - * 区分阻塞/非阻塞的回调, 减少定时器引发的调度开销
 - * 减少细精度定时器对时间堆的增减开销

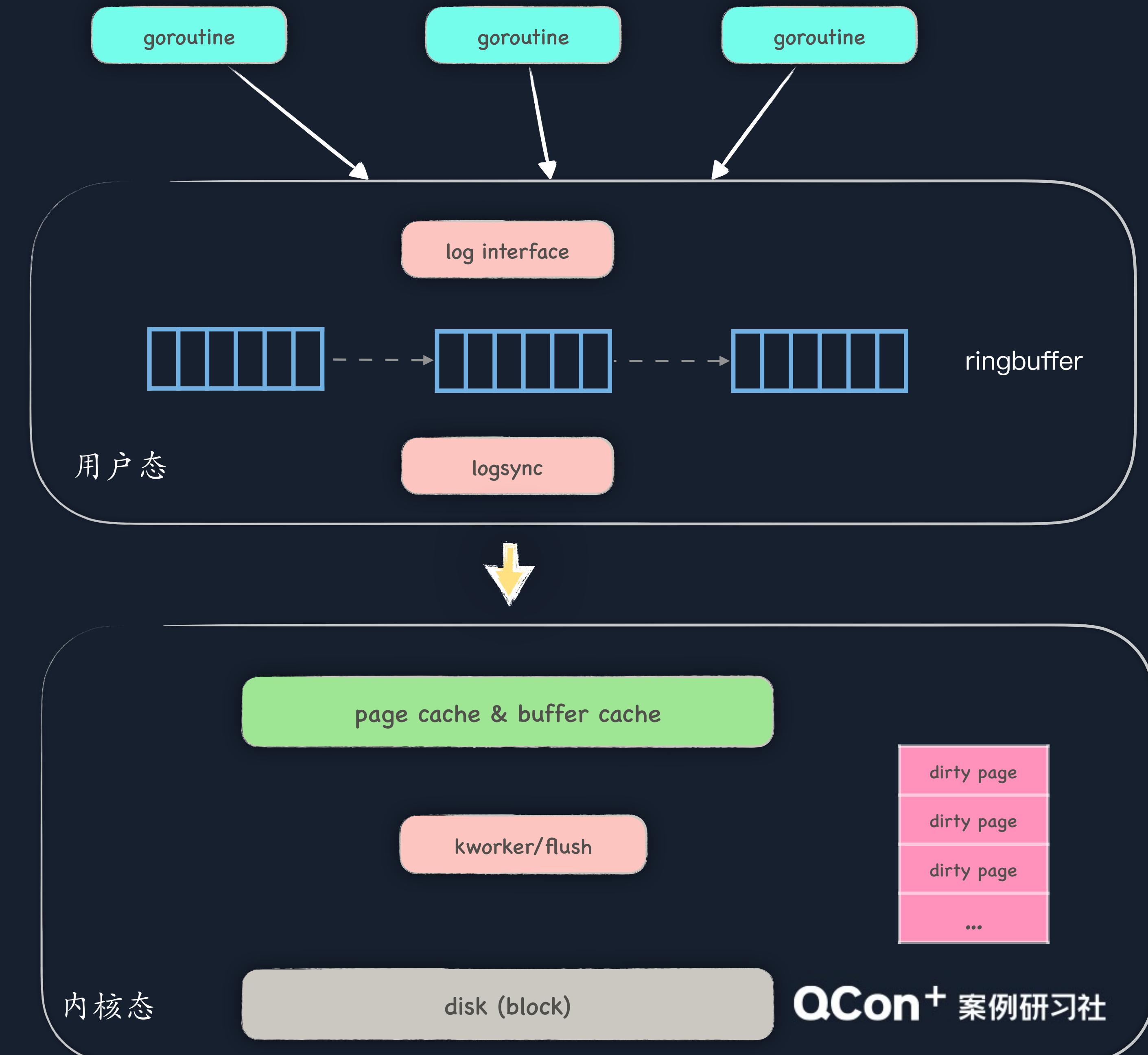
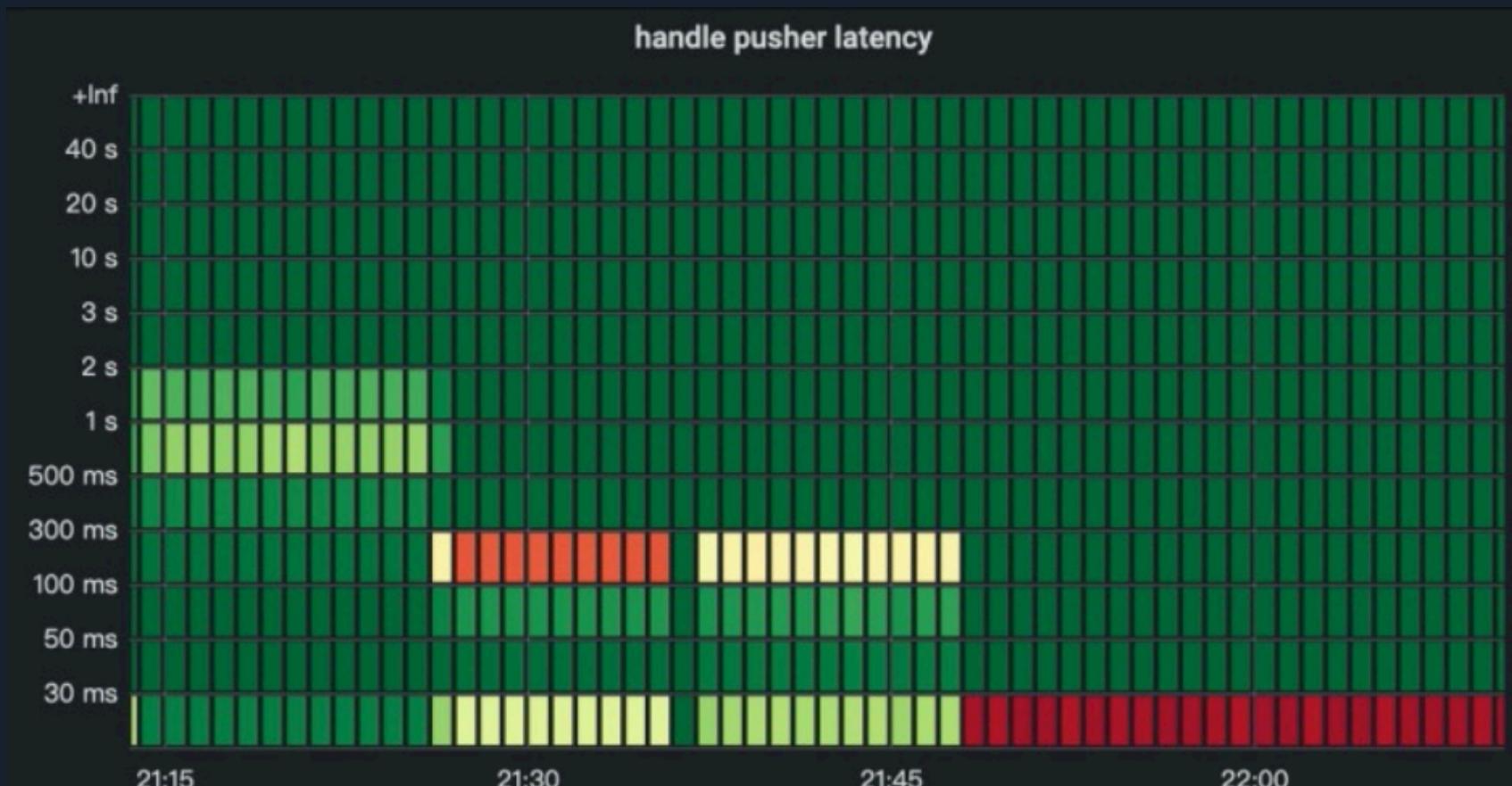


性能调优

Disk IO

- * 业务高峰期打日志引发的问题
 - * disk io 阻塞引发抢占, 线程数暴涨
 - * disk io 阻塞引发消息的高时延
- * 解决方法

- * 写日志到 ringbuffer 队列中, 由一个异步 io flush 协程去消费
- * 磁盘强烈建议使用高 iops 的 ssd



深度优化

netpoll vs rawepoll

* go netpoll 当然很爽 !!!

* 使用标准net库实现的网络库，可同步编码方式写代码！

* 以同步方式写异步代码 !!! 底层自动实现非阻塞io !!!

* goroutine-per-connection 模式

* (长连接推送场景) 通常开读写两个协程

* 业务层通过channel交互数据！



单机 20w 个连接产生 40w个协程 ?

业务读取处理

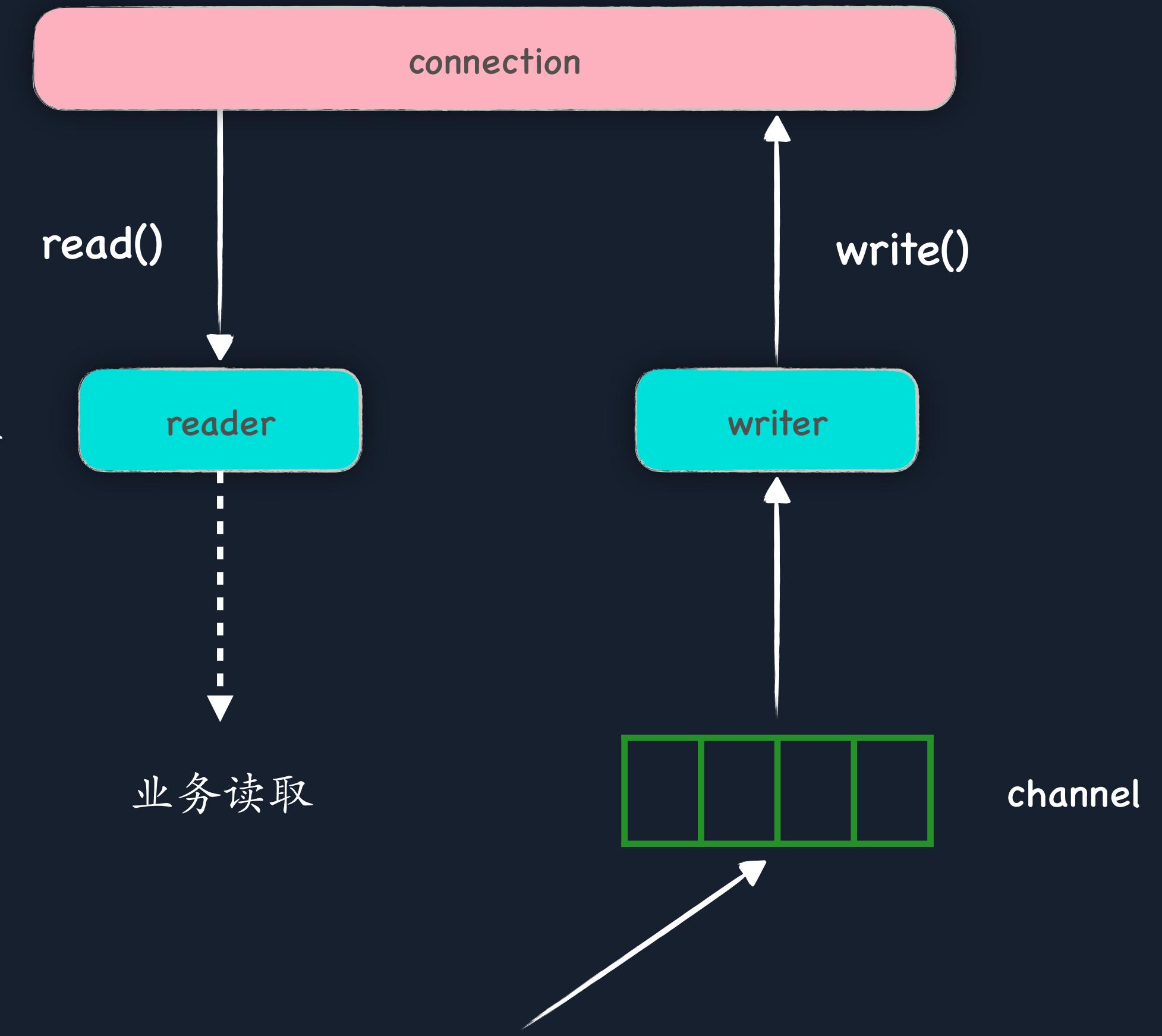
reader

业务读取

writer

channel

业务写入



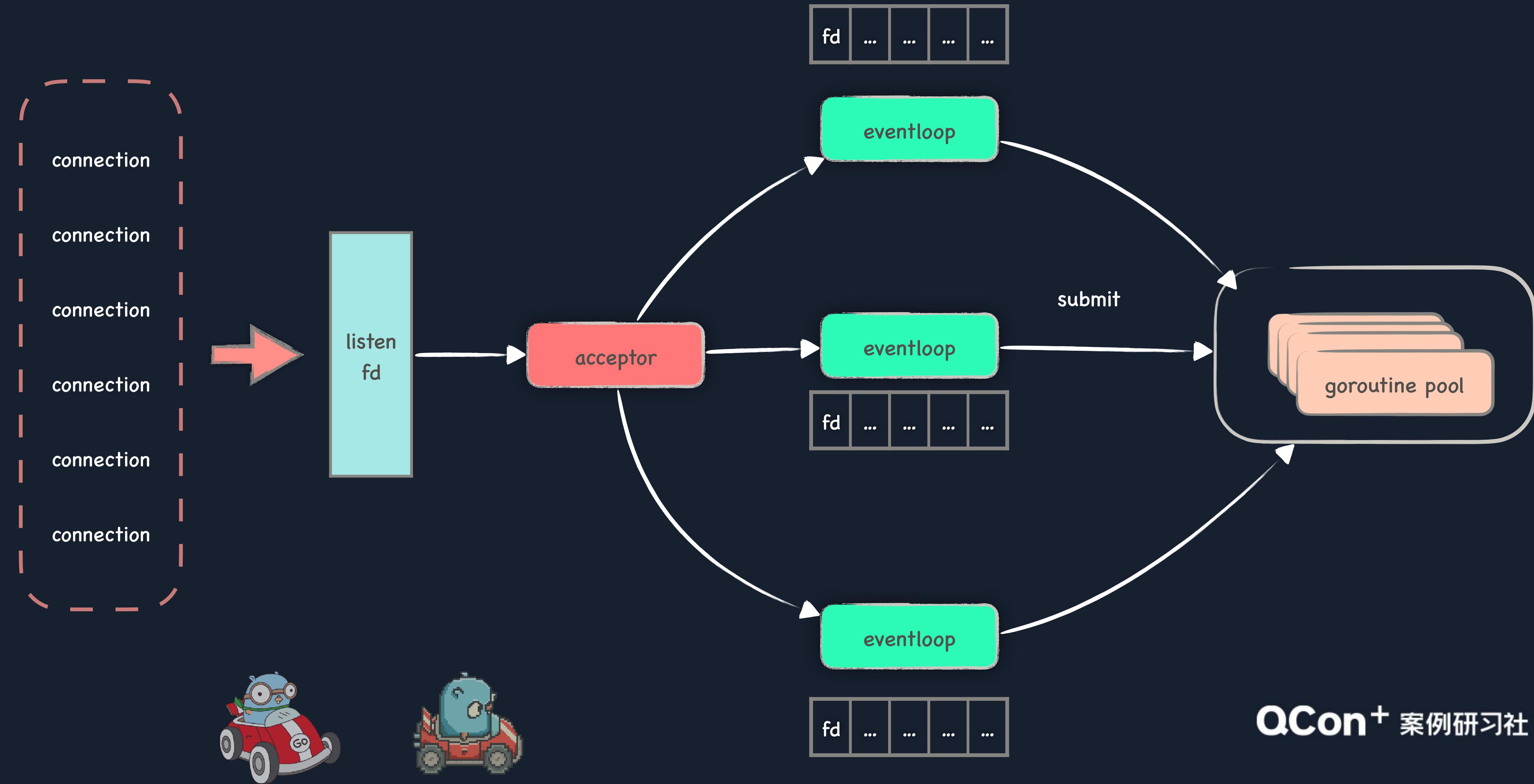
深度优化

netpoll vs rawepoll



深度优化

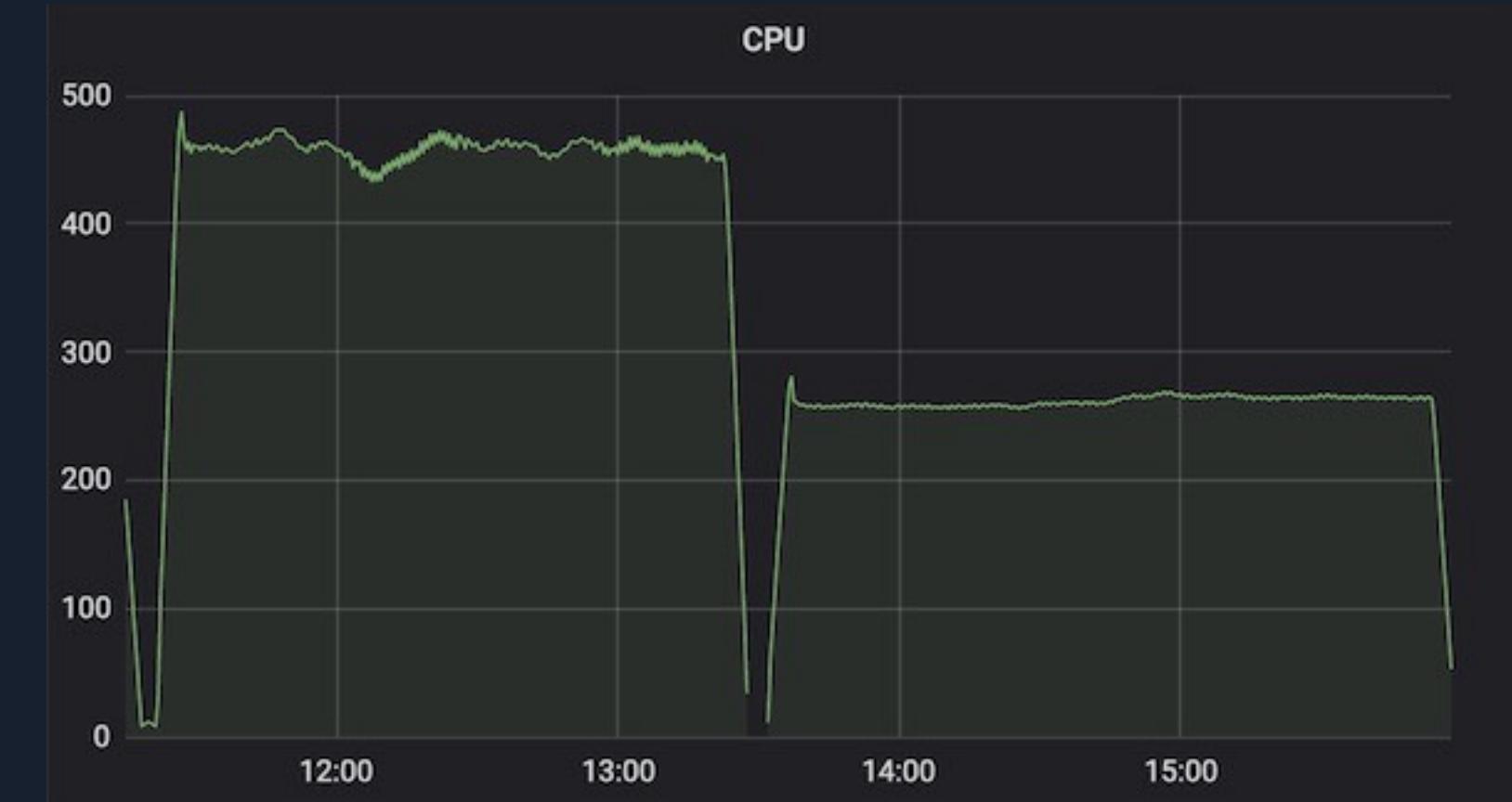
rawepoll



深度优化

netpoll vs rawepoll

- * 压测数据
- * 20w connections
- * produce 20w msgs per 1s
- * one batch per 50 msgs
- * rawepoll 为什么压测下表现更好？
 - * 节省了海量协程产生的调度开销
 - * 减少内存及 gc 扫描开销



深度优化

消息的decode优化

- * access-server 默认收到全量的消息, 在高峰期时, 反序列化实在太大 !!!
- * push-server 把设备ID相关字段放到 kafka header 中
- * access-server 先 kafka header 进行设备过滤
- * 设备 ID 在当前 access-server 连接集合中
 - * 反序列化 kafka value 的消息
 - * 发送消息数据
- * 否则, 直接丢弃 !

减少无用的 decode !!!

kafka message

```
// For implementation reasons
// be set to true in its config
type SyncProducer interface {
    // SendMessage produces a single message
    // succeeded or failed.
    // of the produced message.
    SendMessage(msg *ProducerMessage) error

    // SendMessages produces multiple messages
    // in the sequence specified by the
    // can succeed and fail.
    // SendMessages will return errors for
    // pass-through data.
    SendMessages(msgs []*ProducerMessage) error

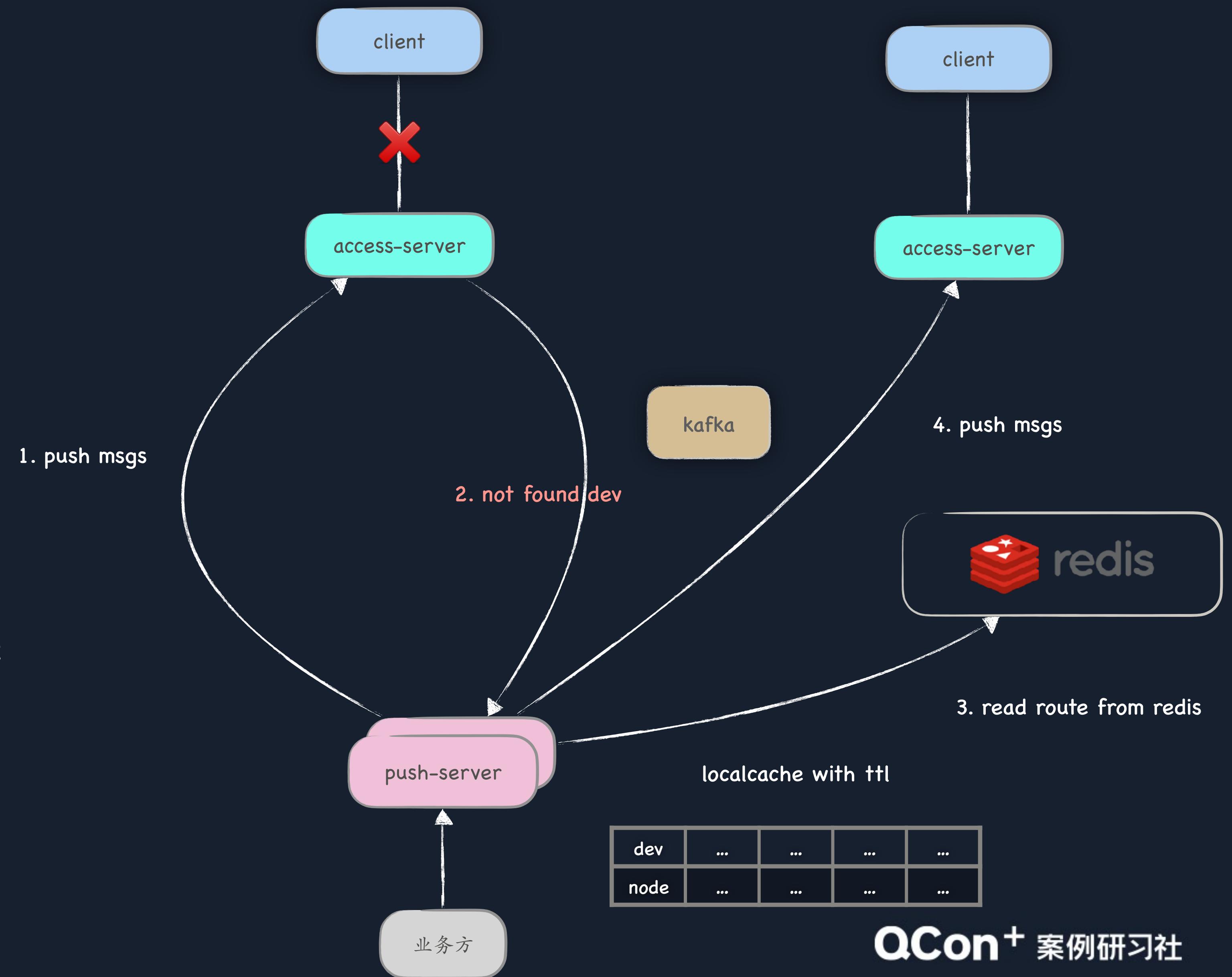
    // Close shuts down the producer; you must call this function before a
    // object passes out of scope, as it may otherwise leak memory.
    // You must call this before calling Close on the underlying client.
    Close() error
}
```

深度优化

消息的惊群问题

- * push-server 根据路由来直推消息
- * push-server 本地缓存路由关系
- * 当出现多次直推失败时，回退到 kafka 广播逻辑

惊群最大的性能优化点 !!!



总结页

- * 可靠性

- * 消息的 ack 确认机制

- * 离线消息机制

- * kafka ack 确认机制

- * 高可用

- * 各个节点为无状态

- * 可横向扩容

- * 高并发

- * 本地缓存

- * 批量操作

- * rawepoll

- * 高吞吐的关键优化

- * 批量

- * 批量

- * 批量

- * ...

出门在外,一定要保护好自己的
头发 !!!

QCon⁺ 案例研习社

THANKS

QCon+ 案例研习社