confluent cloud
Apache Kafka as a Service

Confluent Cloud

kafka

kakfa的设计与实现

LinkedIn

- xiaorui.cc

CONFLUENT

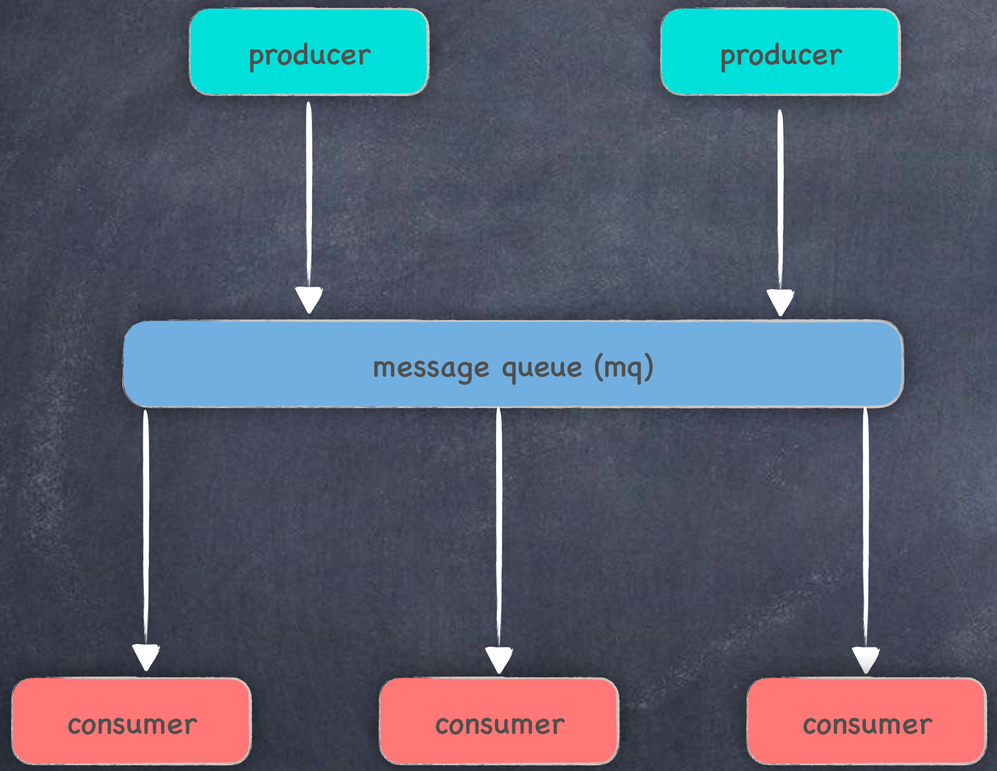
- github.com/rfyiamcool

APACHE kafka®
A distributed streaming platform

confluent

kafka 介绍

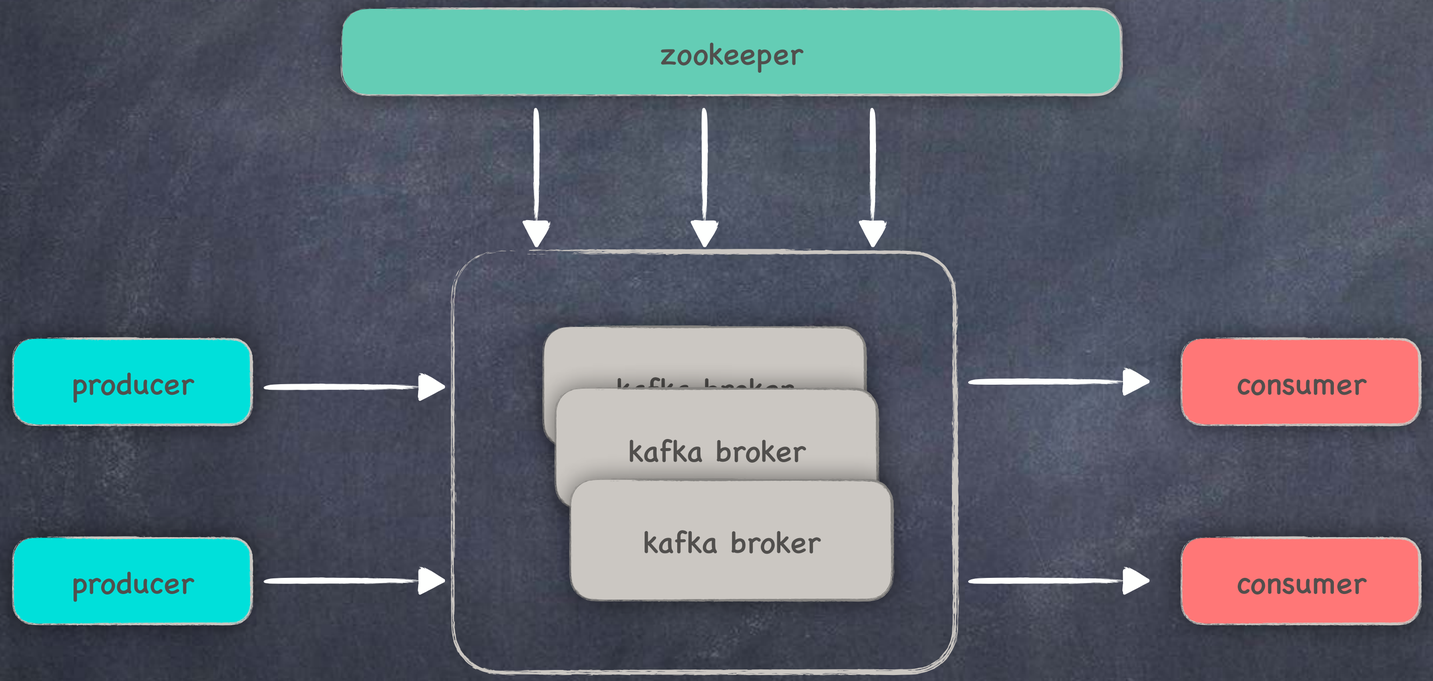
message queue ?



- 异步通信
- 解耦
- 削峰
- 发布订阅
- 可恢复性
- 顺序排队
- ...

老生常谈 . . .

what is kafka ?

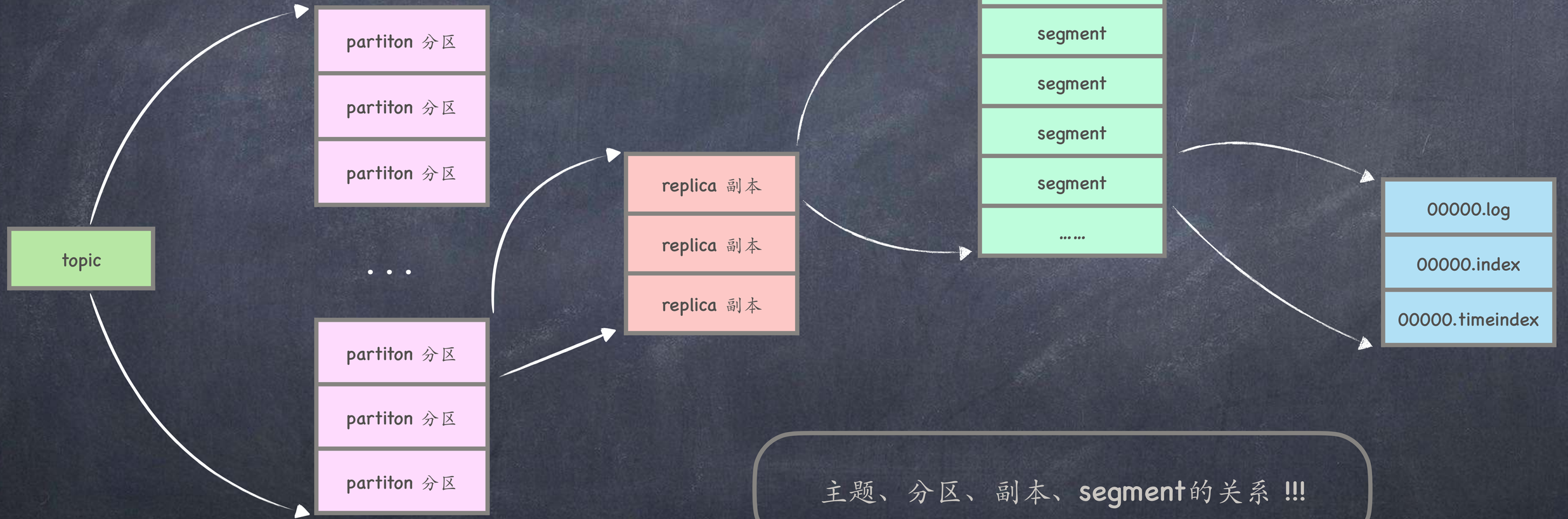


- 优点
  - 高吞吐
  - 消息持久化
  - 负载均衡
  - 故障转移
  - 伸缩性
  - …

kafka naming

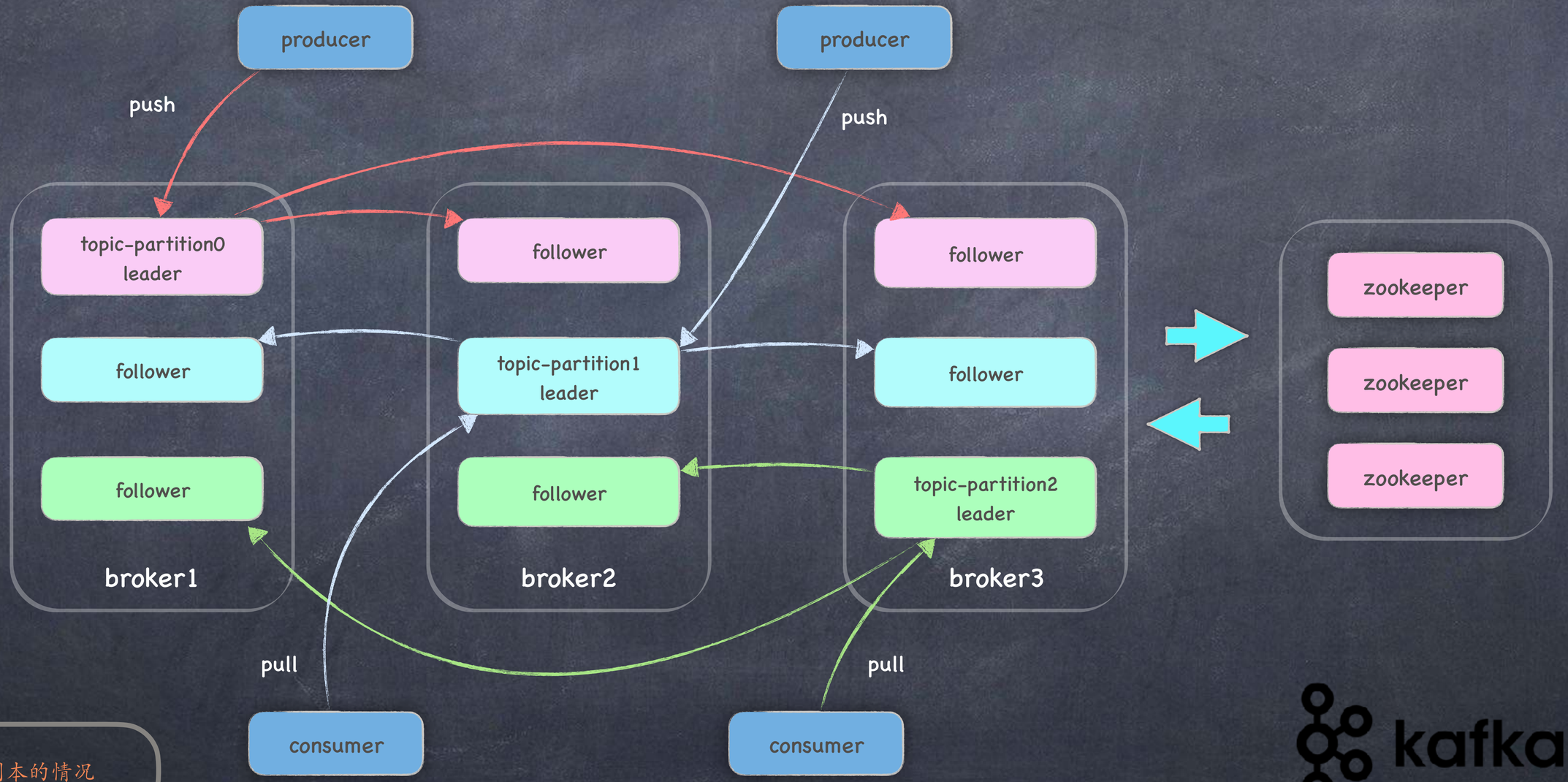
- broker
  - 一个节点就是一个broker，一个Kafka集群由一个或多个broker组成。
- producer
  - 负责向kafka broker发送消息(push方式)
- consumer
  - 负责从kafka broker拉去消息(pull方式), 多个消费者组成一个consumer group
- topic
  - 消息的分类，同一类消息一个topic
- partition
  - 同一个topic上面的消息，进行分区来存储

view



主题、分区、副本、segment的关系 !!!

design



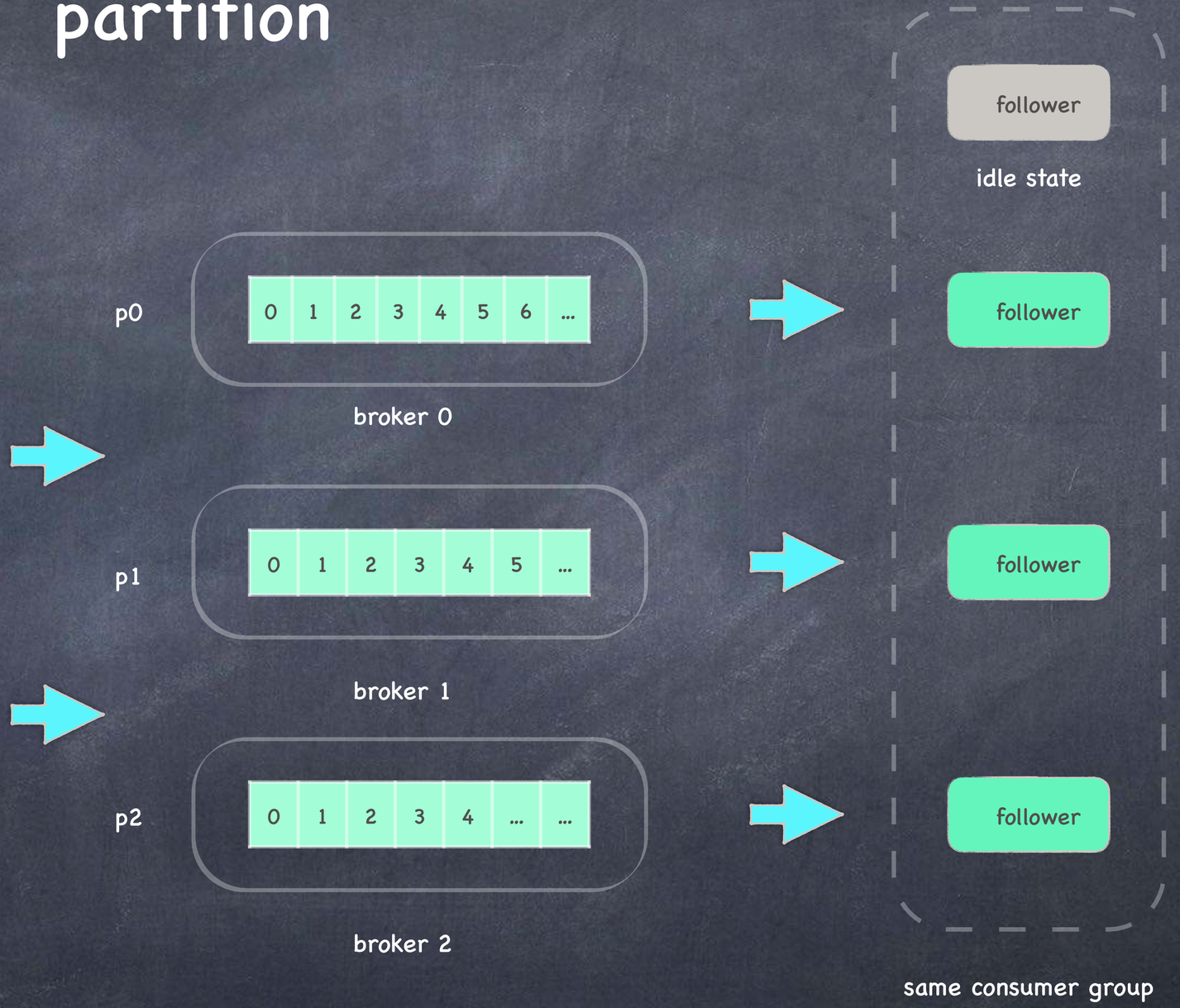
一个topic三个分区三副本的情况

kafka

partition

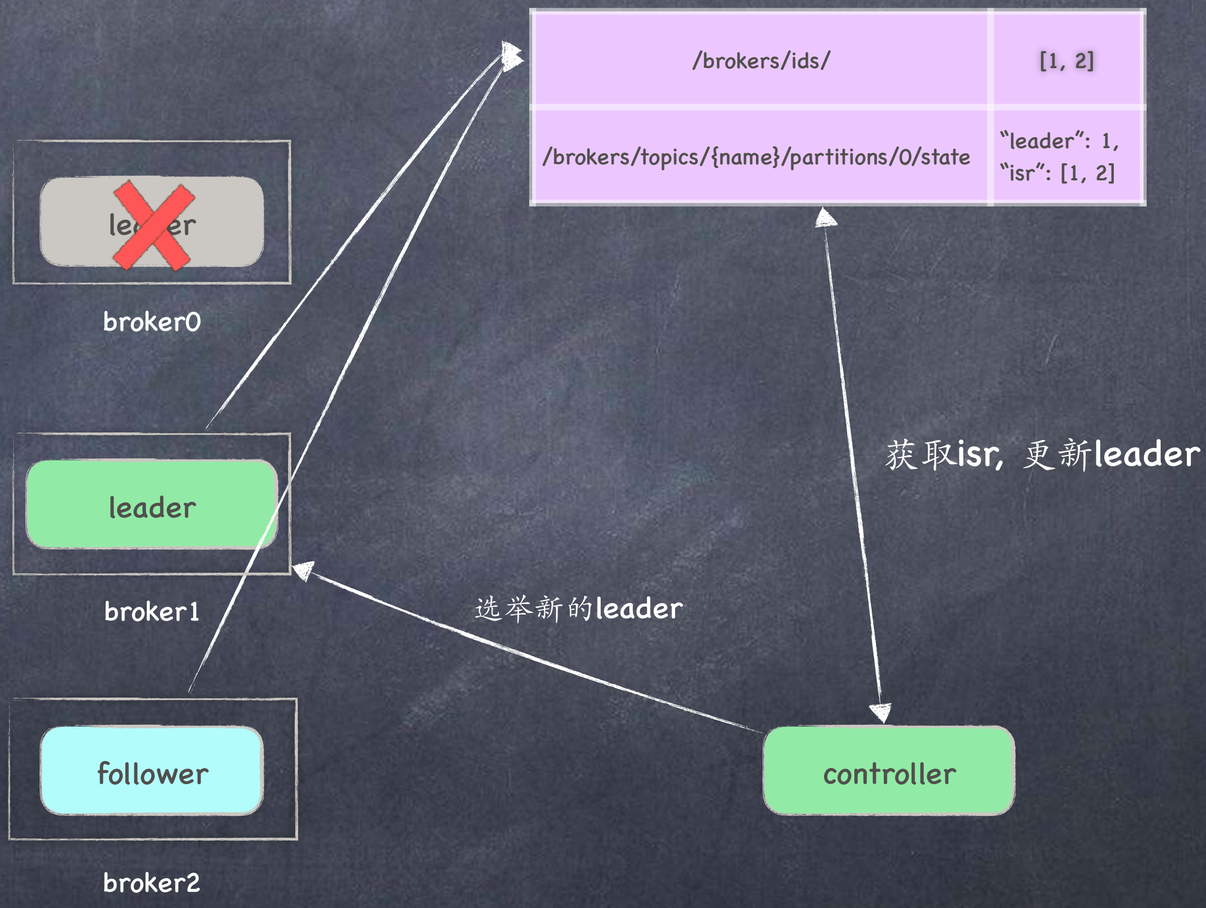
- 一个 topic 下的 partiton 分布到多个 broker 上
  - 提高客户端的并行度，提高效率
  - 提高吞吐率，减轻单机负载压力
  - 分布式存储，减轻单机存储压力
  - 提高水平扩展性，可扩容分区数
  - …

一个 partiton 为一个并发维度！！！



zookeeper

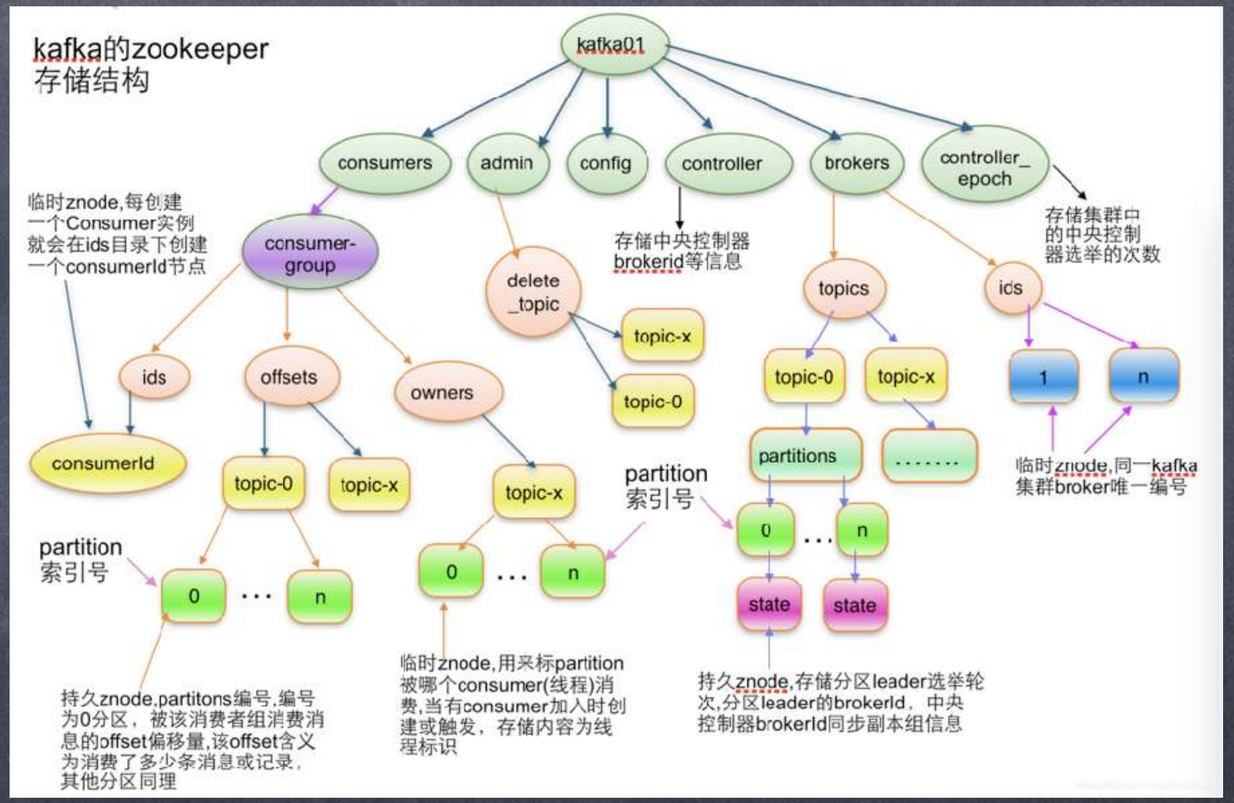
- zookeeper 作用
  - broker 的上线、下线处理
  - 新创建的 topic 或已有 topic 的分区扩容，处理分区副本的分配、leader 选举
  - 管理所有副本的状态机和分区的状态机，处理状态机的变化事件
  - topic 删除、副本迁移、leader 切换等处理



比如，某个broker掉线，leader的选举

zookeeper

- /brokers/ids/[0 - N]
  - 记录 broker 服务器节点
  - 不同的 broker 必须使用不同的 broker ID 进行注册
  - 每个 broker 就会将自己的 ip 地址和端口信息记录到节点
  - 临时节点, crash后会被清理
- /borkers/topics/[topic]
  - 记录topic的分区及broker的对应信息
- /consumers/[group_id]/ids/[consumer_id]
  - 消费者负载均衡
- /consumers/[group_id]/owners/[topic]/[broker_id-partition_id]
  - 消费者跟broker和分区的映射关系
- /controller
  - 身为 控制器 的broker id
- . . .



2

kafka 为什么快？

why fast !!!

kafka

- 磁盘顺序 io 写入 （ O_APPEND ）
- 分布式分布 partition
- 高效的日志分段及索引设计
- mmap
- 充分利用 page cache

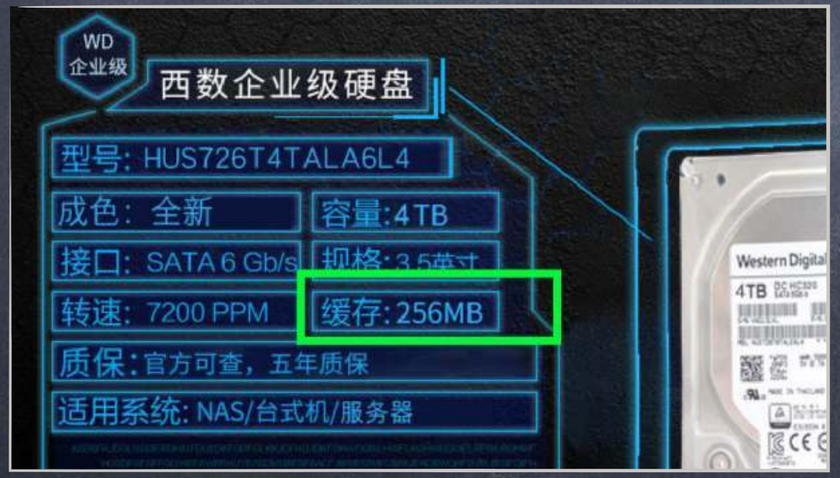
- reactor 网络模型
- sendfile
- 消息压缩
- 批量操作
- 时间轮
- 支持多Disk Drive
- ...

机械硬盘

每一圈为一个磁道，每个框为一个扇区 !!!

如何访问在第五磁道的第3扇区数据 ???

- 机械硬盘通常为每分钟 7200, 10000 转
- IOPS = 1000ms / (寻道时间 + 旋转延迟 + 数据传输时间)
- 寻道时间
  - 磁头移动到指定磁道需要的时间
  - 耗时在 3 - 15 ms （通常选定 3 ms）
- 旋转延迟
  - 磁头定位到某一磁道的扇区所需要的时间
  - 7200 rpm = 4.17 ms
  - 10000 rpm = 3 ms
  - 旋转简单计算（分钟）
    - 最坏耗时 （等一圈）
      - 60 x 1000ms ÷ 7200 = 8.33 ms
    - 平均耗时 （等半圈）
      - 60 x 1000ms ÷ 7200/2 = 4.17 ms
- 传输时间
  - 从磁盘读出或者写入经历的时间 （通常在计算时省略）



大多数硬盘有缓存机制 !!!

- 寻道
  - 7200转/分平均物理寻道时间是10.5ms
  - 10000转/分平均物理寻道时间是7ms
  - 15000转/分平均物理寻道时间是5ms
- 旋转时延
  - 60*1000/7200/2 = 4.17ms
  - 60*1000/10000/2 = 3ms
  - 60*1000/15000/2 = 2ms
- IOPS
  - 7200 rpm = 140
  - 10000 rpm = 167
  - 150000 rpm = 200

ssd

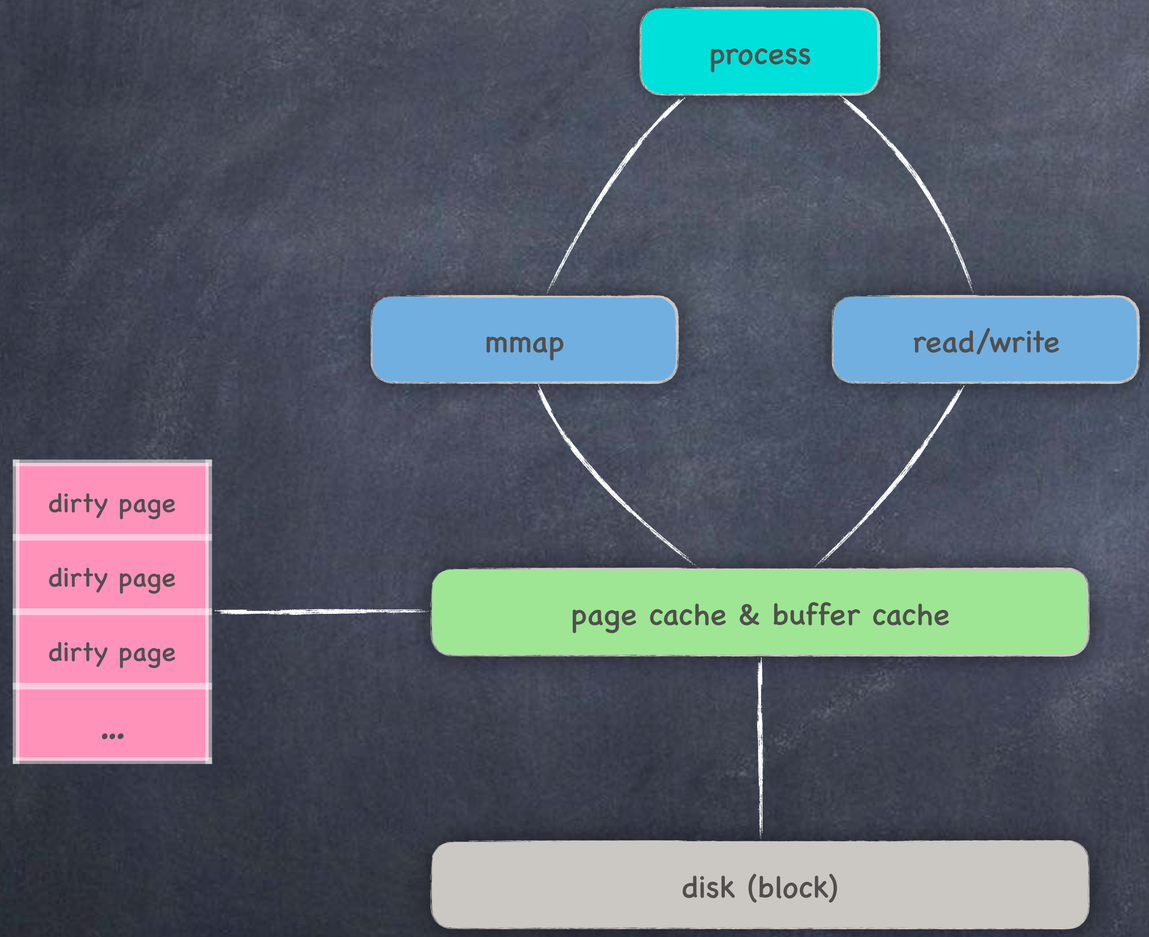
消费级 SSD

- iops
  - 顺序读 60w iops
  - 顺序写 50w iops
- bd
  - 顺序读 3500 mb/s
  - 顺序写 3200 mb/s

企业级 SSD

- iops
  - 顺序读 48w iops
  - 顺序写 40w iops
- bd
  - 顺序读 2900 mb/s
  - 顺序写 1400 mb/s

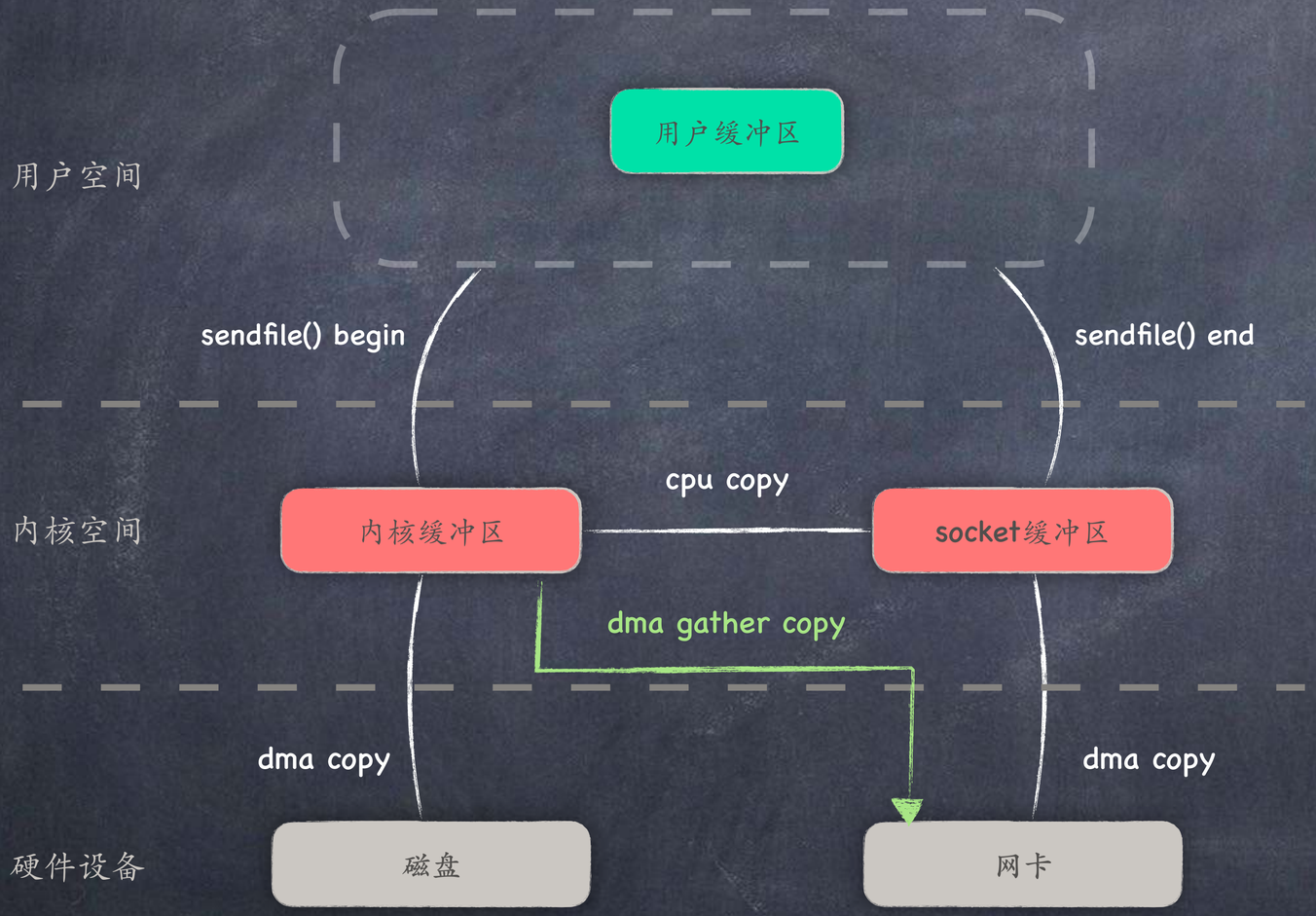
page cache



- file o_append 日志追加模式
- kafka 强制写盘 （官方不建议调整）
  - log.flush.interval.messages
  - log.flush.interval.ms
- dirty_background_ratio = 10 %
  - 脏页率超过指标开始 Flush Dirty PageCache
- dirty_ratio = 20 %
  - 阻塞所有的写操作来进行Flush
- dirty_writeback_centisecs = 500 // 5s
  - 检查是否需要 flush
- dirty_expire_centisecs = 3000 // 30s
  - 把超过 30s 的脏页写到磁盘里

kafka

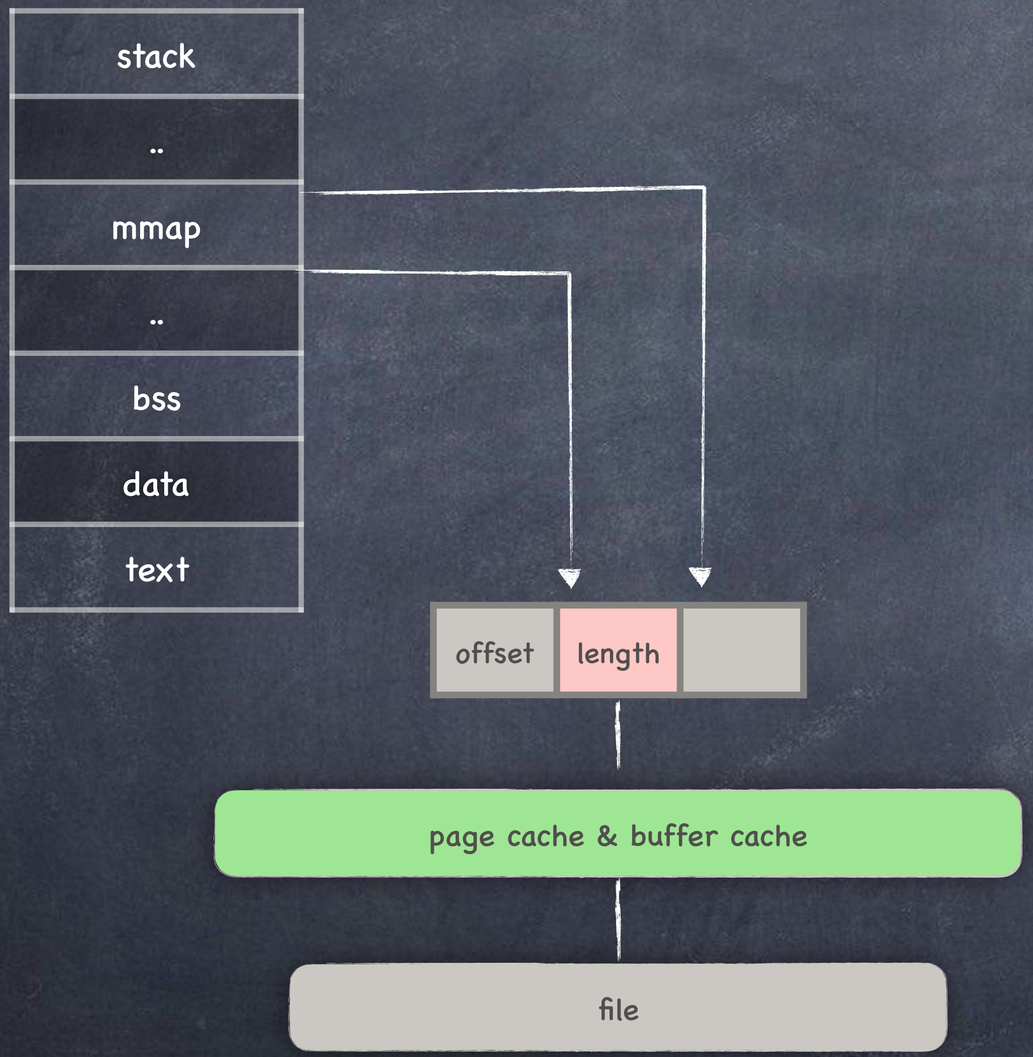
sendfile



- sendfile
  - 用户进程发起sendfile系统调用
  - 内核基于DMA Copy将文件数据从磁盘拷贝到内核缓冲区
  - 内核将内核缓冲区中的文件描述信息(文件描述符，数据长度)拷贝到Socket缓冲区
  - 如硬件支持 gather copy 模式，则可以直接把文件描述符信息复制到网卡
  - 用户进程sendfile系统调用完成并返回
- 减少 copy次数，减少系统调用次数，减少上下文次数

kafka

mmap



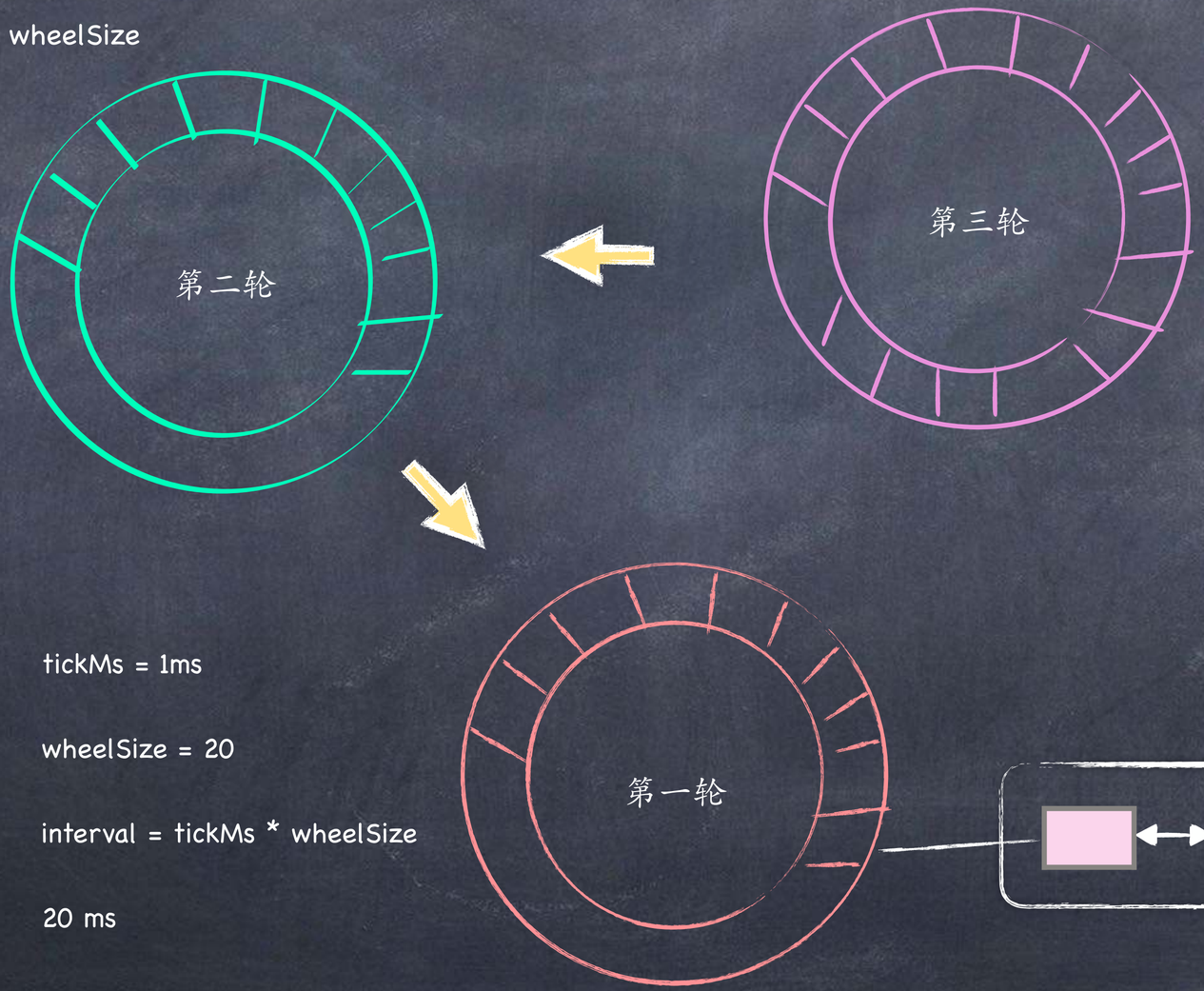

```
    file, err := os.OpenFile("my.db", os.O_RDWR|os.O_CREATE, 0644)
    if err != nil {
        panic(err)
    }
    defer file.Close()

    ...

    b, err := syscall.Mmap(int(file.Fd()), 0, size, syscall.PROT_WRITE|syscall.PROT_READ,
syscall.MAP_SHARED)
    if err != nil {
        panic(err)
    }

    for index, bb := range []byte("Hello world") {
        b[index] = bb
    }

    err = syscall.Munmap(b)
    if err != nil {
        panic(err)
    }
```

* 文件直接映射到进程虚拟地址空间
* 像操作内存一样操作文件, 不在使用 file api
* 减少用户空间到内核空间的拷贝

kafka

时间轮

- tickMs = 20 ms
- wheelSize = 20
- interval = tickMs * wheelSize
- 400 ms



- tickMs = 400 ms
- wheelSize = 20
- interval = tickMs * wheelSize
- 8000 ms

- tickMs = 1ms
- wheelSize = 20
- interval = tickMs * wheelSize
- 20 ms

日志布局

- partiton
  - segment
    - index
    - timeindex
    - log
  - 二分查找

topic

partiton

partiton

...

delete from front

appends

00000000.index
00000000.timeindex
00000000.log

00001111.index
00001111.timeindex
00001111.log

00002222.index
00002222.timeindex
00002222.log

segment

segment

segment

00001111.index

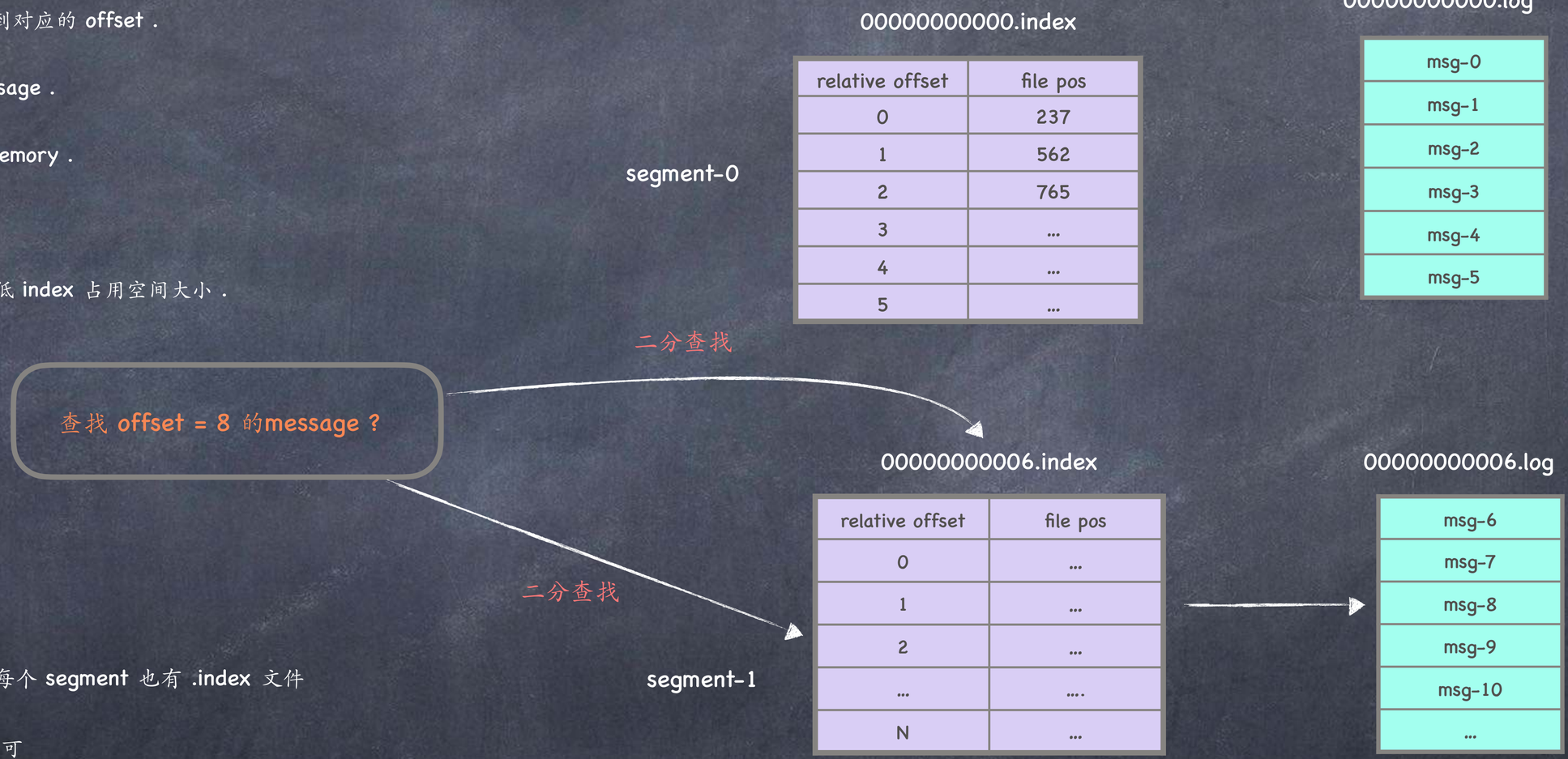
relative offset	file pos
100	7291
200	10333
300	14999
...	....
N	pos

00001111.log

msg
Msg1111 - value
...
Msg1411 - value
...
N

segment

- segment
  - 通过 二分查找 找到对应的 segment .
  - 再通过 二分查找 从 index 找到对应的 offset .
  - 通过 log 可以快速seek到 message .
  - 且 index 元数据全部映射到 memory .
  - offset 为 相对 偏移量 .
  - 索引文件是稀疏索引，大幅降低 index 占用空间大小 .
- partition 为什么要分 segment ?
  - 插入效率，消息追加模式
  - 查找效率，文件名本就索引，每个 segment 也有 .index 文件
  - 清理删除效率，删掉旧文件即可
  - mmap 内存大小



kafka

timeindex

- 目的
  - 通过时间选择offset
  - 基于时间删除数据

00000000000.timeindex

timestamp	offset
1613808236	5
1613808239	10
1613808257	16
1613808266	22
1613808277	27
...	...

1. 根据时间戳找到 不大于 该时间戳对应的偏移量 .

00000000000.index

relative offset	file pos
6	...
14	...
18	...
...	....
N	...

2. 根据查询的偏移量找到 不大于 偏移量的物理文件位置

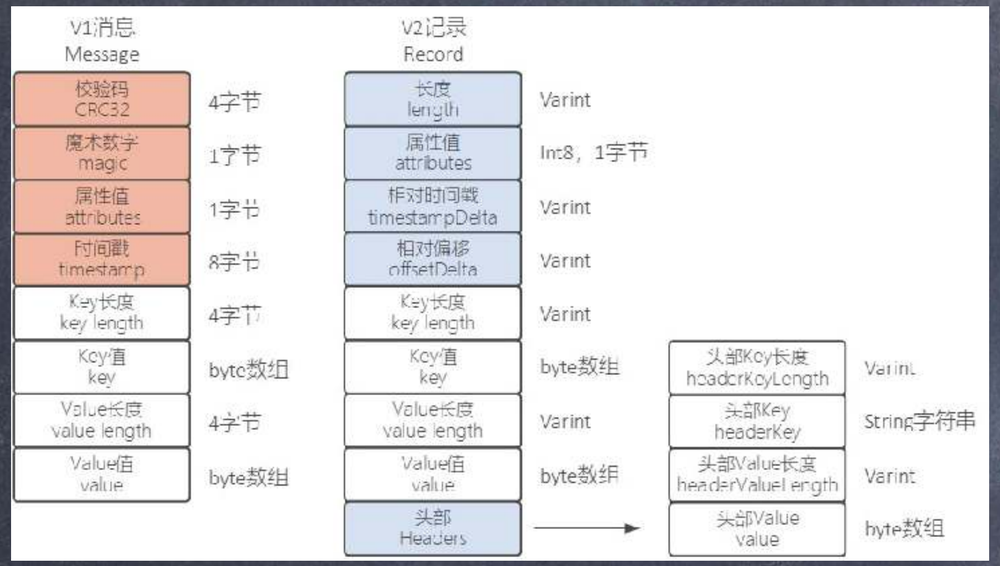
3. 从物理文件中找到对应的消息 .

00000000000.log

msg
msg
msg
msg
...
...
msg
msg
msg
msg

kafka

message




```
length: varint            # 长度
attributes: int8          # 属性值
    bit 0~7: unused
timestampDelta: varint    # 时间戳
offsetDelta: varint       # 偏移
keyLength: varint         # Key长度
key: byte[]               # Key
valueLen: varint          # Value长度
value: byte[]             # Value
Headers => [
    headerKeyLength: varint       # header的Key长度
    headerKey: String             # header的Key
    headerValueLength: varint     # header的Value长度
    Value: byte[]                 # header的Value
]
```

- 具有更好的压缩效果
  - 类似 protobuf 的 varint
  - 类似 Facebook Gorilla 的相对值

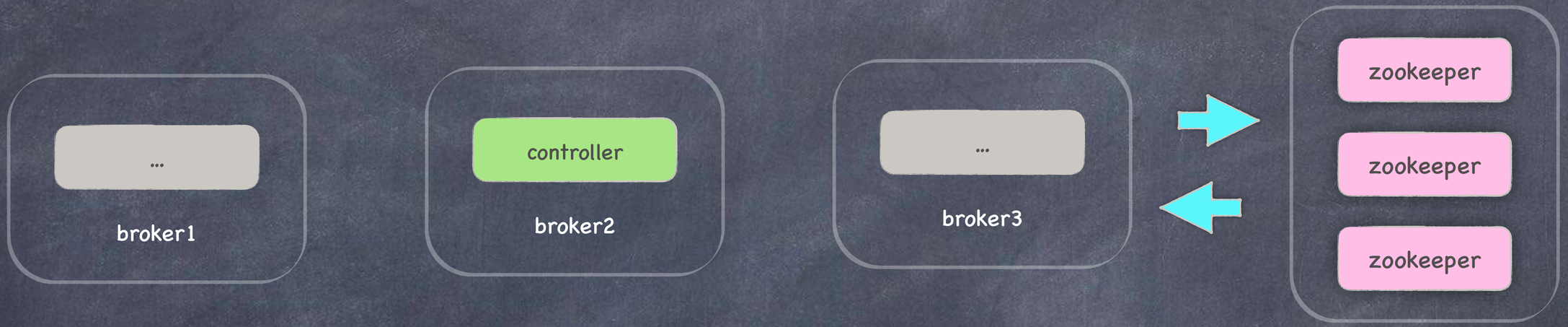
v2

kafka

3

基础实现

controller

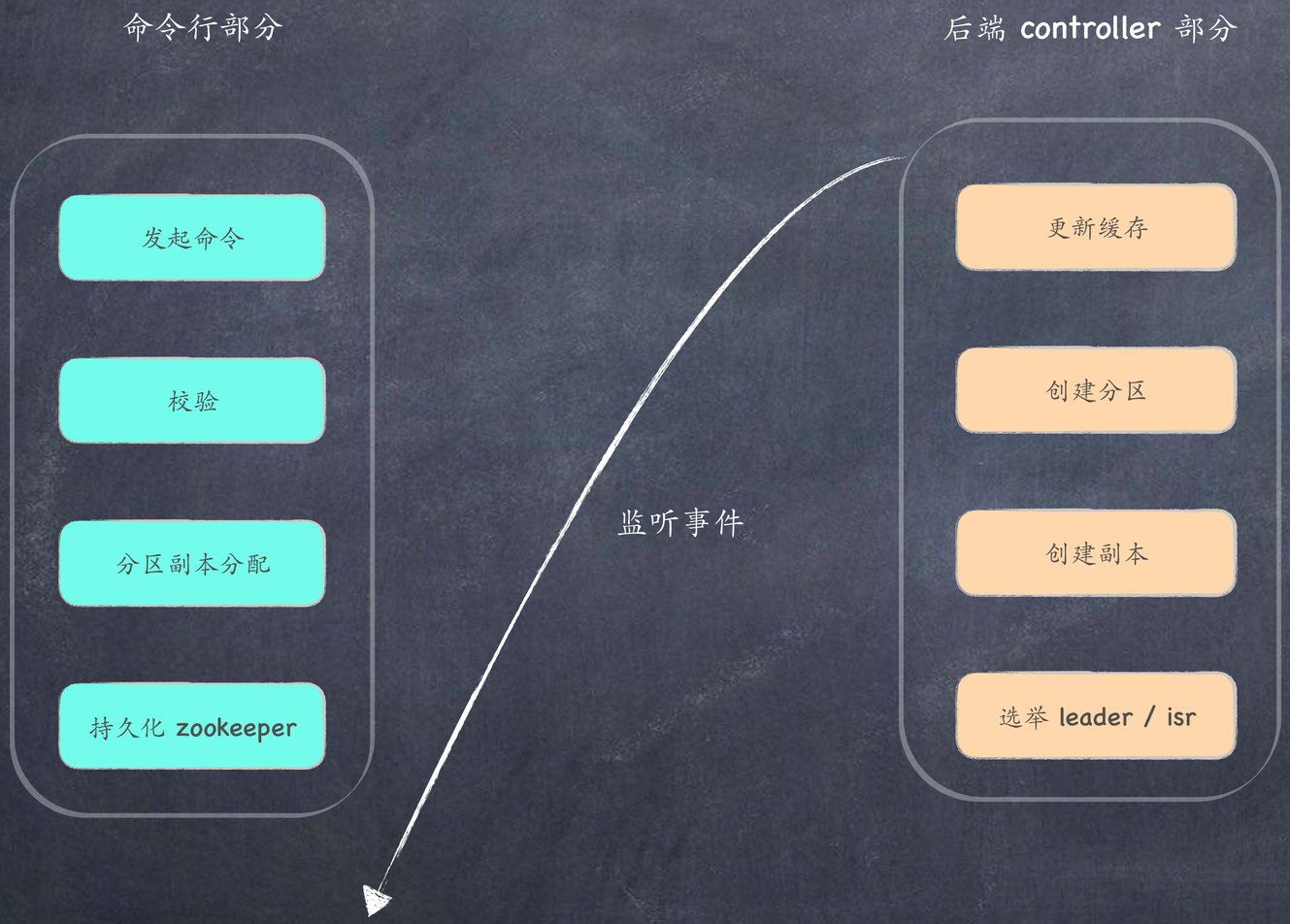


- 谁可以先在 "/controller" 目录创建临时节点，谁就是 controller .
- kafka 通过 zookeeper 选出 controller 管理集群 .
- broker 主要存消息体，zookeeper 存元数据 .

- 主要功能
  - broker 上下线管理
  - topic 管理
  - partition 管理
  - 其他元数据管理

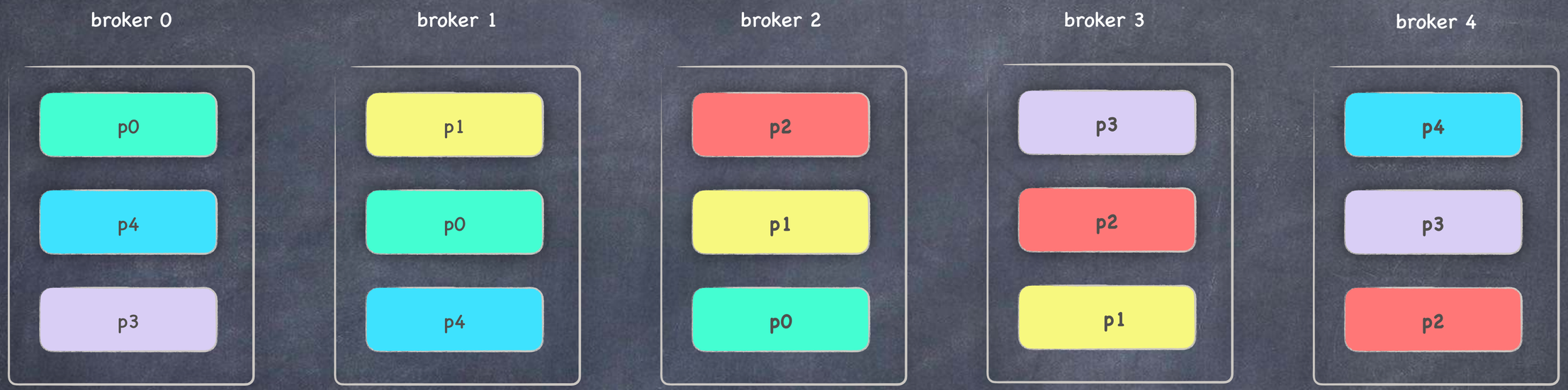
kafka

create topic



- client
  - 确定分区副本的分配方案
    - 就是每个分区的副本都分配到哪些broker上 .
  - 创建zookeeper节点
    - 把这个方案写入/brokers/topics/<topic>节点下 .
- controller
  - 创建分区
  - 创建副本
  - 为每个分区选举leader、ISR
  - 更新各种缓存

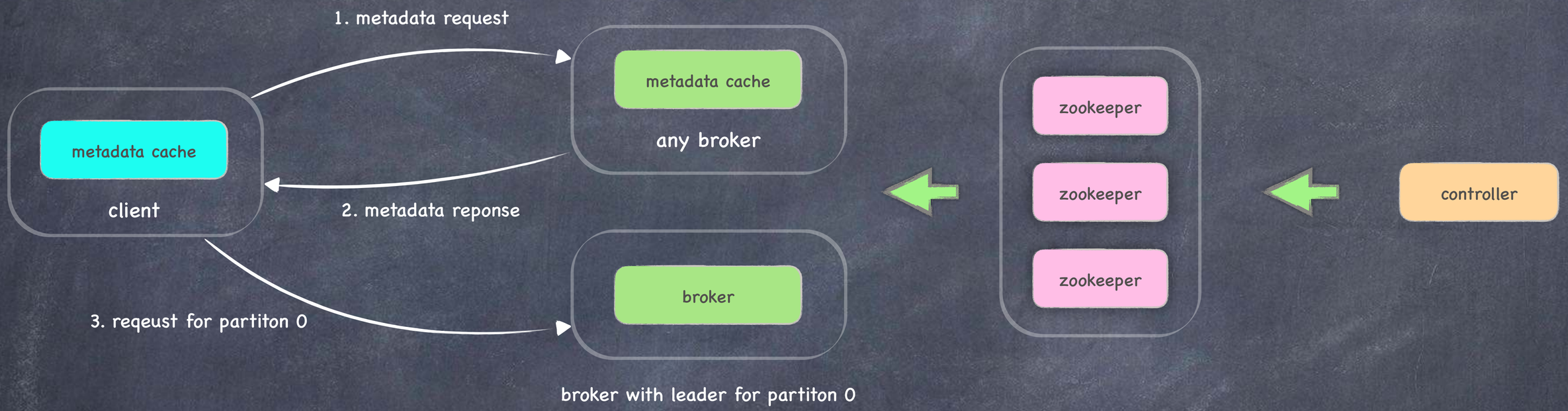
副本分布



- 副本分配原则
  - 副本分配到不同的 Broker 上
  - 分区的第一个副本通常是 leader

- 分配算法
  - 将所有 Broker和待分配的 Partition 排序
  - 随机选取选择一个分配的位置起点
  - 将第 i 个 Partition 分配到第(i mod n)个 Broker 上 (这个就是 Leader)
  - 将第 i 个 Partition 的第 j 个 Replica 分配到第 (i + j) mode n 个 Broker 上

metadata

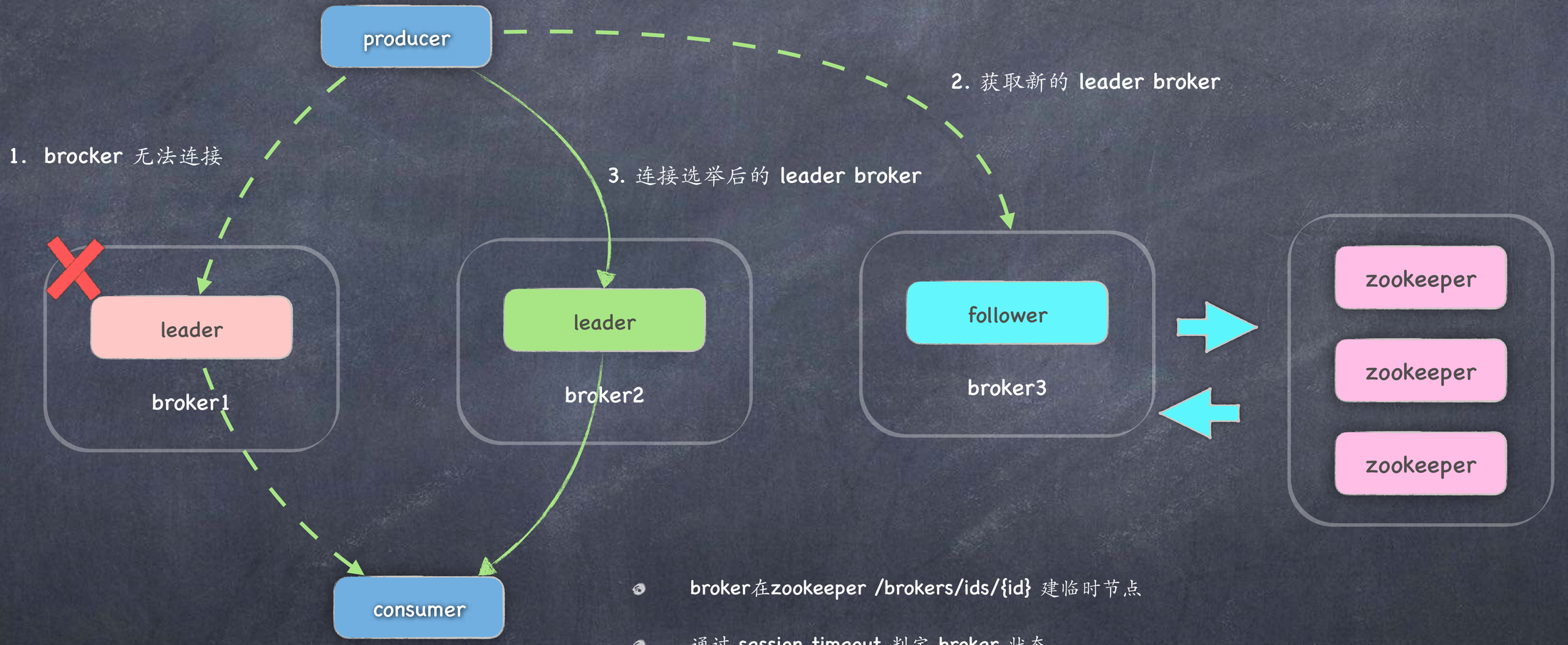


- client 通过 metadata 从任意 broker 知晓集群信息
  - Kafka中存在哪些主题？
  - 每个主题有几个分区及副本分布情况？
  - Leader 分区所在的 broker 地址及端口？
  - 每个 broker 的地址及端口是多少？
  - 总之，整个集群的元数据都可获取！

client: 该去哪里去读写？

- clienet 什么时候更新 metadata？
  - 在往Kafka发送请求是收到 Not a Leader 异常
  - meta_data.max.age.ms 过期后

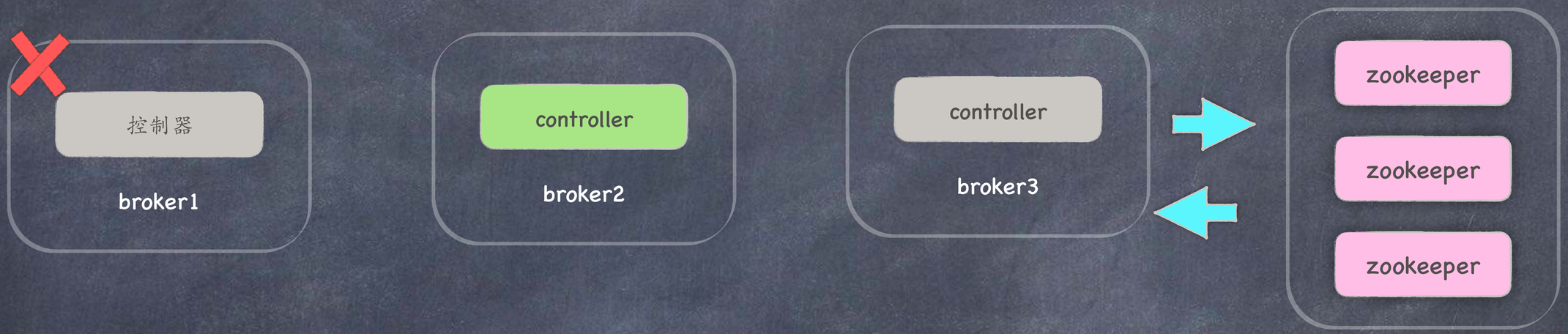
broker failover



- broker在zookeeper /brokers/ids/{id} 建临时节点
- 通过 session timeout 判定 broker 状态
- 由 controller 控制选举 leader
- 选举范围在 isr 的副本集合里

kafka

controller failover

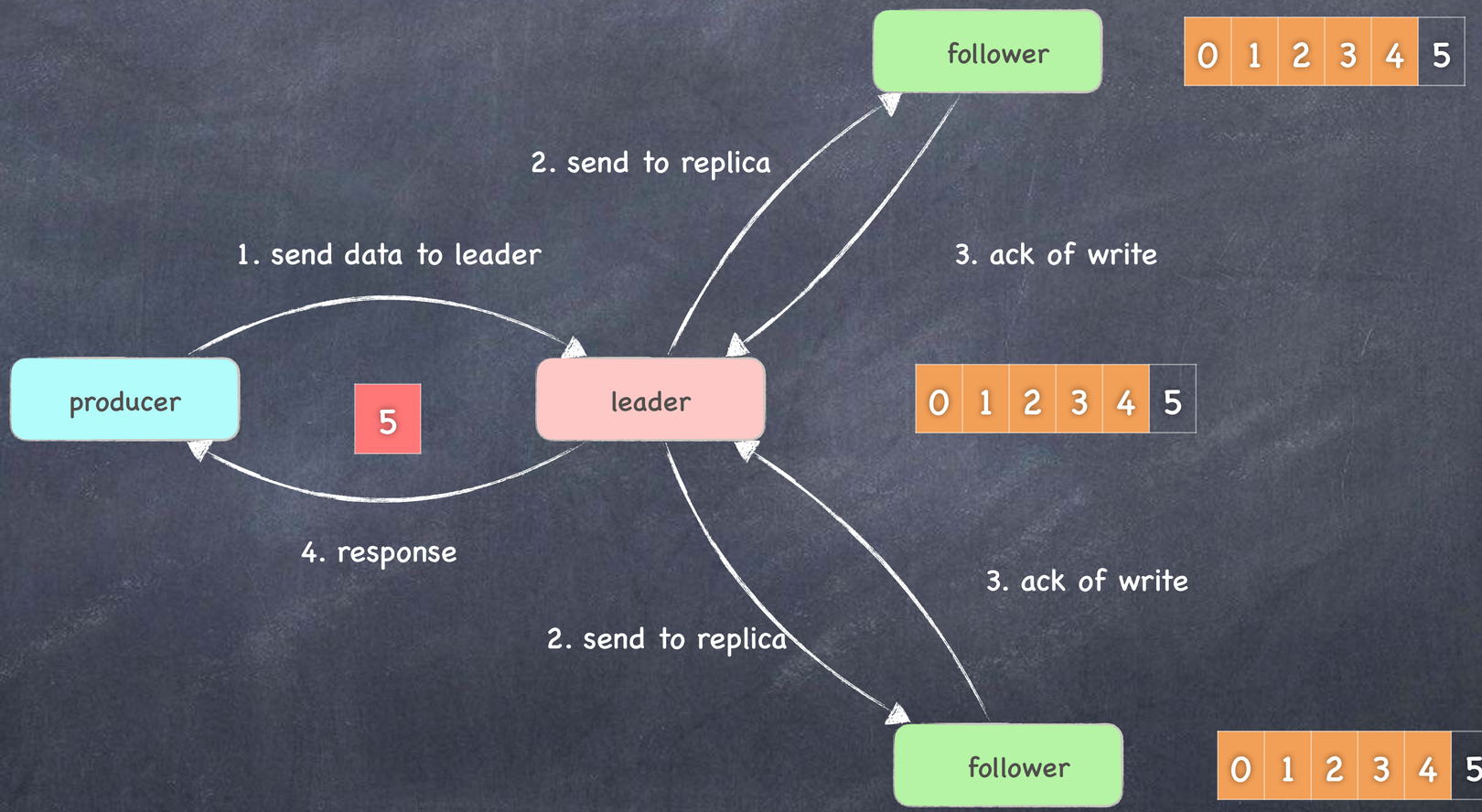


- 谁可以先在 "/controller"目录创建节点，谁就是 controller .
- Controller 主要是作为 master 来仲裁 partition 的 leader 选举，并维护 partition 和 replicas 的状态机，以及相应的 zk 的 watcher 注册 .

kafka

safe producer

- acks = 0 (no ack)
  - 达到最大的吞吐
  - 只管发送，无需关注 broker 回复报文
- acks = 1 (leader ack)
  - 默认模式
  - 需要 leader 写入后的回复确认
  - 由 leader 异步传输给其他 replicas
- acks = all (replicas ack)
  - 需要 leader 与 replicas的 确认
  - 满足 in-sync replicas 副本数
  - 增加 latency 时延

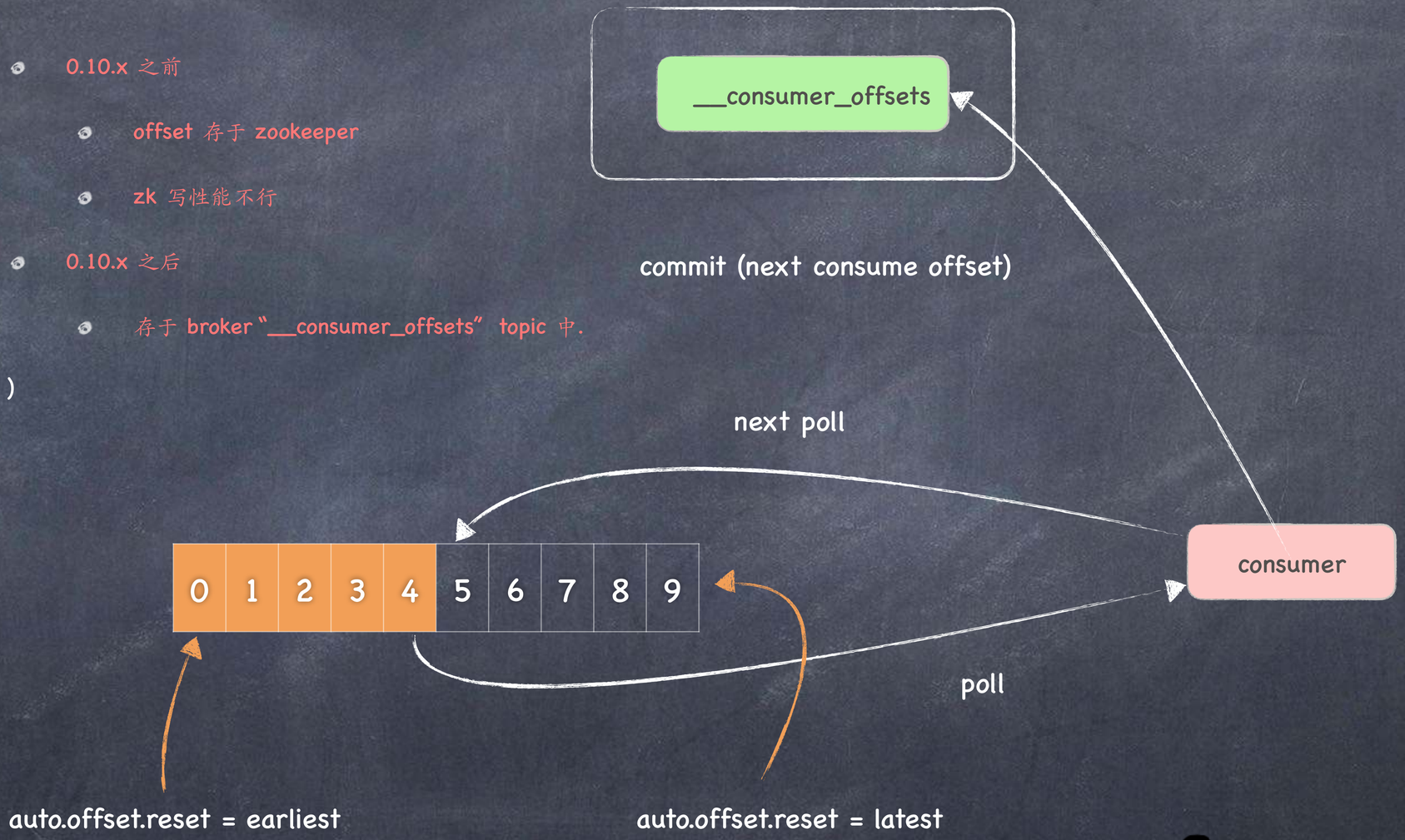


kafka

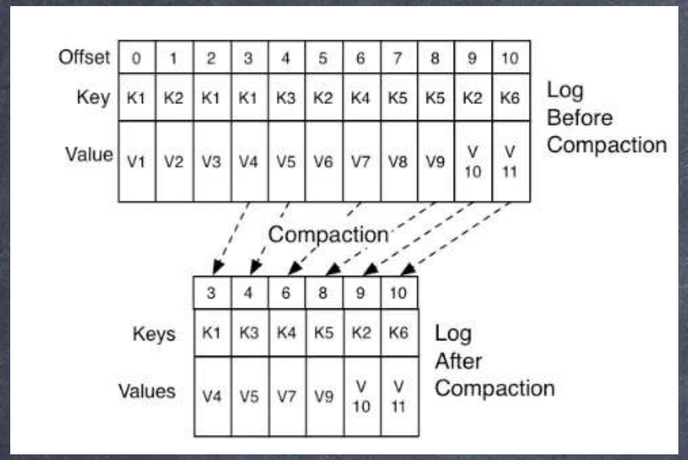
commit & offset

- question
  - 消息丢失
  - 消息重复
- 自动提交 （defualt）
  - enable.auto.commit = true
  - auto.commit.interval.ms = 5000 （default）
  - consumer.close()
- 手动提交
  - enable.auto.commit = false
  - commitSync() 同步阻塞提交
  - commitAsync() 不会失败重试
- 指定 offset
- 指定 时间戳

- 0.10.x 之前
  - offset 存于 zookeeper
  - zk 写性能不行
- 0.10.x 之后
  - 存于 broker “__consumer_offsets” topic 中.



consumer_offsets



- key
  - version, group, topic, partition
- Value
  - version, offset, metadata
  - commit_timestamp, expire_timestamp

- 当有消费者第一次消费kafka数据时就会自动创建 .
- 内置的 __consumer_offsets 分区数为 50 （default）.
- consumer_groupName.hashCode % __consumer_offset.partition.num
- Log Cleaner 线程会定时清理无用的日志 .
- 通过 Compaction 压缩清理过期的消息，同样的 key 消息只保留最新的 .

Kafka Log Cleanup

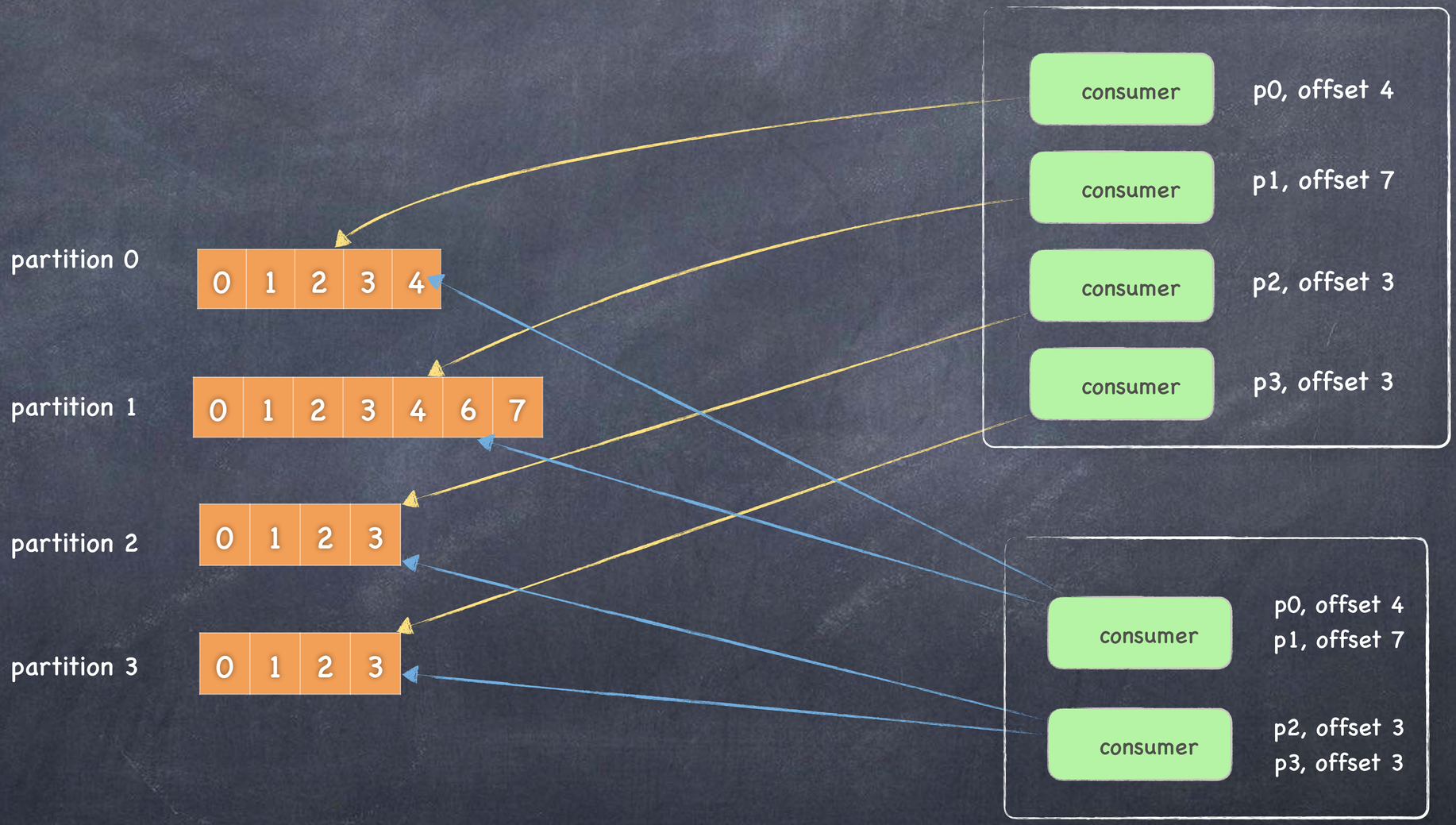
- delete
  - Kafka中所有用户创建的topics，默认均为此策略
  - log.retention.hours=168
    - 默认的保留最长时间是7天
  - log.retention.bytes=-1
    - 指定每个 partition 的日志大小，默认 -1 为无限制
- compact
  - topic __consumer_offsets 默认为此策略

- log.cleaner.enable = true (default)
- when
  - log.retention.check.interval.ms = 300000 // 5m
- log.segment.bytes // 1GB
  - 单个 log 段超过空间限制，则创建新的log段
- log.roll.hours // 7d
  - 日志超过 7d 则进行轮转切分日志

consumer group

- 面向全局，而非单个 topic
- consumer group 之间为广播
- consumer group 内部为单播
- 多余的consumer则空闲阻塞
- 单个consumer也可订阅多个分区

- 消费者分区分配策略
  - range （default）
  - roundrobin
  - Sticky

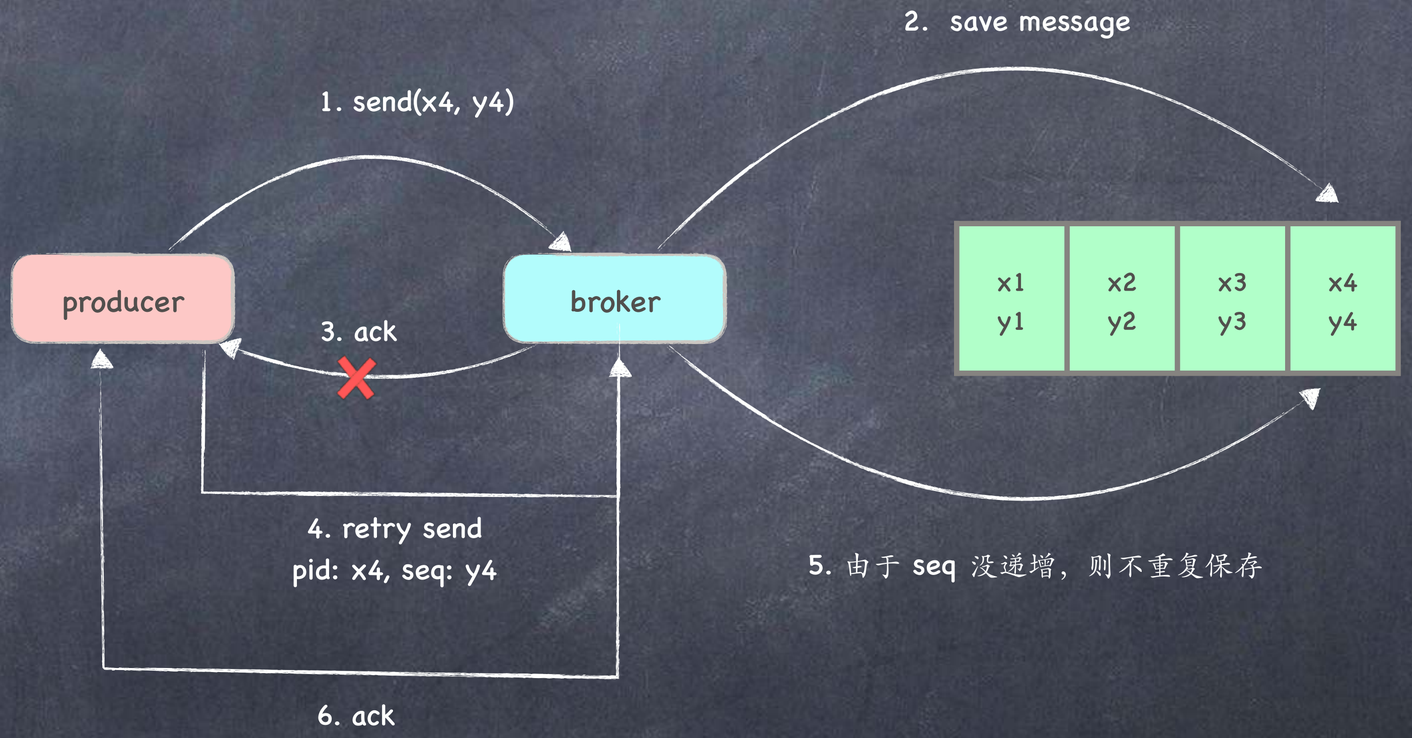


4

高级实现

生产者幂等性

- 生产者幂等性
  - producer ID
    - 每个producer初始化时，分配唯一的pid .
  - sequence number
    - producer 发送数据的每个 Topic 和 Partition 都对应一个从0开始单调递增的 SequenceNumber 值 .
- how to enable ?
  - enable.idempotence = true



消费者幂等性

- 利用数据库的唯一约束实现幂等
- 去重表
- 利用 redis 实现去重
- token机制
- ...

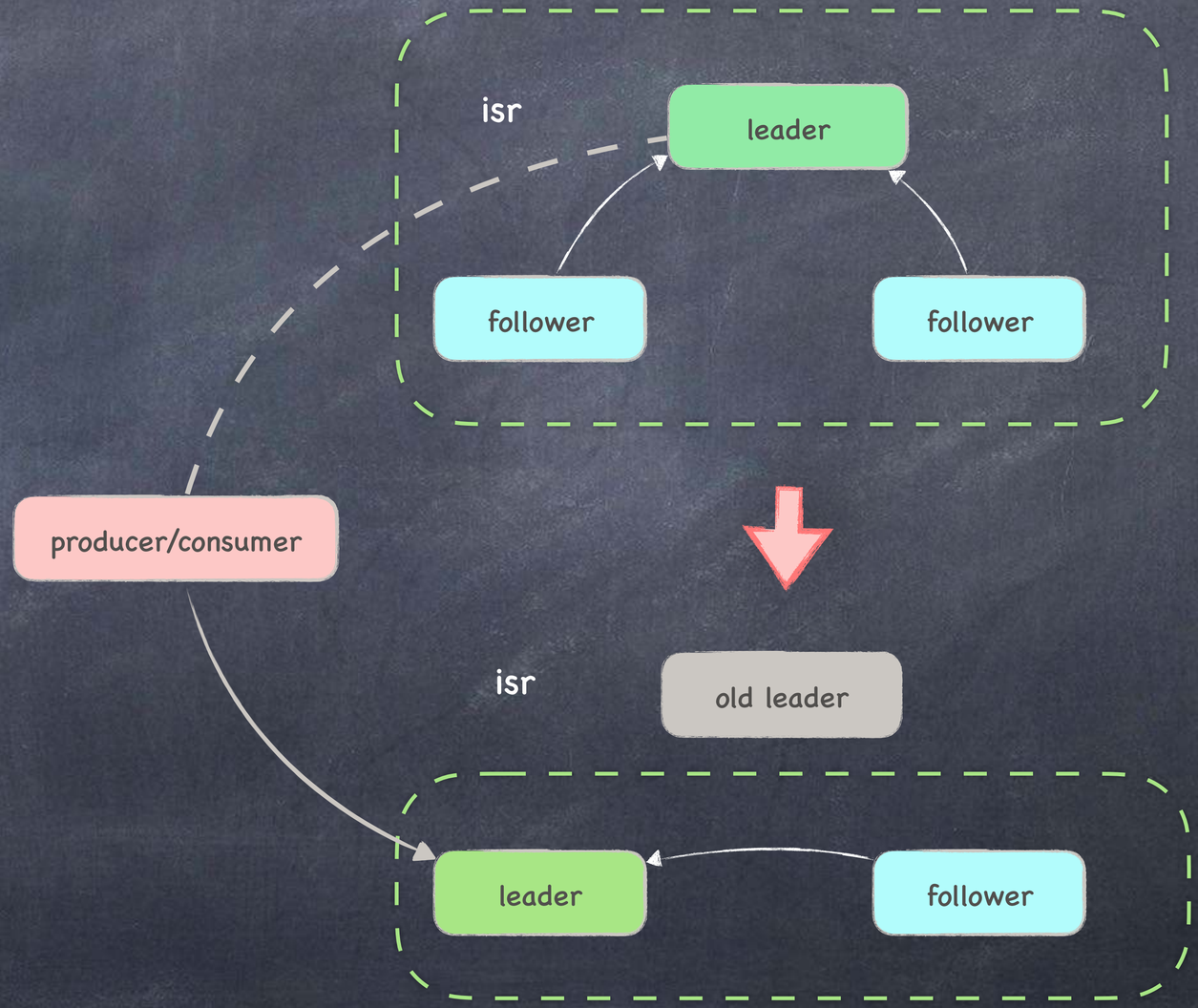
需要业务方搞定重复消息的幂等性 !!!

副本同步列表（isr）

- isr (in-sync Replica)
  - isr 就是 leader 会维护一个与其基本保持一定同步的 replica 列表
  - 每个 partition 都会有一个 isr .
  - 如果 follower 超过一定时间未发起数据复制请求，则被 leader 踢出 isr .
  - 当 isr 中所有 replica 都向 leader 发送 ack 时，leader才commit到本地 .
- 满足 isr 条件
  - 副本节点必须能与zookeeper保持会话 （心跳机制）
  - 副本跟 leader 保持一定的同步（ 满足 replica.lag.time.max.ms ）
- AR 所有已分配的副本（Assigned Replicas ）
  - AR = ISR + OSR
- ISR 在同步的副本（In Sync Replicas ）
  - 跟 leader 保持一定程度同步的副本
- OSR 非同步的副本（Out-of-Sync Replied ）
  - 同步滞后太多，超过阈值从 isr 踢出的副本
  - 新副本也属于 osr
- 0.9.0 之前
  - replica.lag.max.messages // 4000
  - 如瞬时生产的消息太多，导致 follower 无法及时消费，也要被踢出 isr ？？？
- 0.9.0 后
  - replica.lag.time.max.ms // 10 s
  - 允许 follower 副本不同步消息的最大时间值
  - “ Configuration parameter replica.lag.max.messages was removed ”

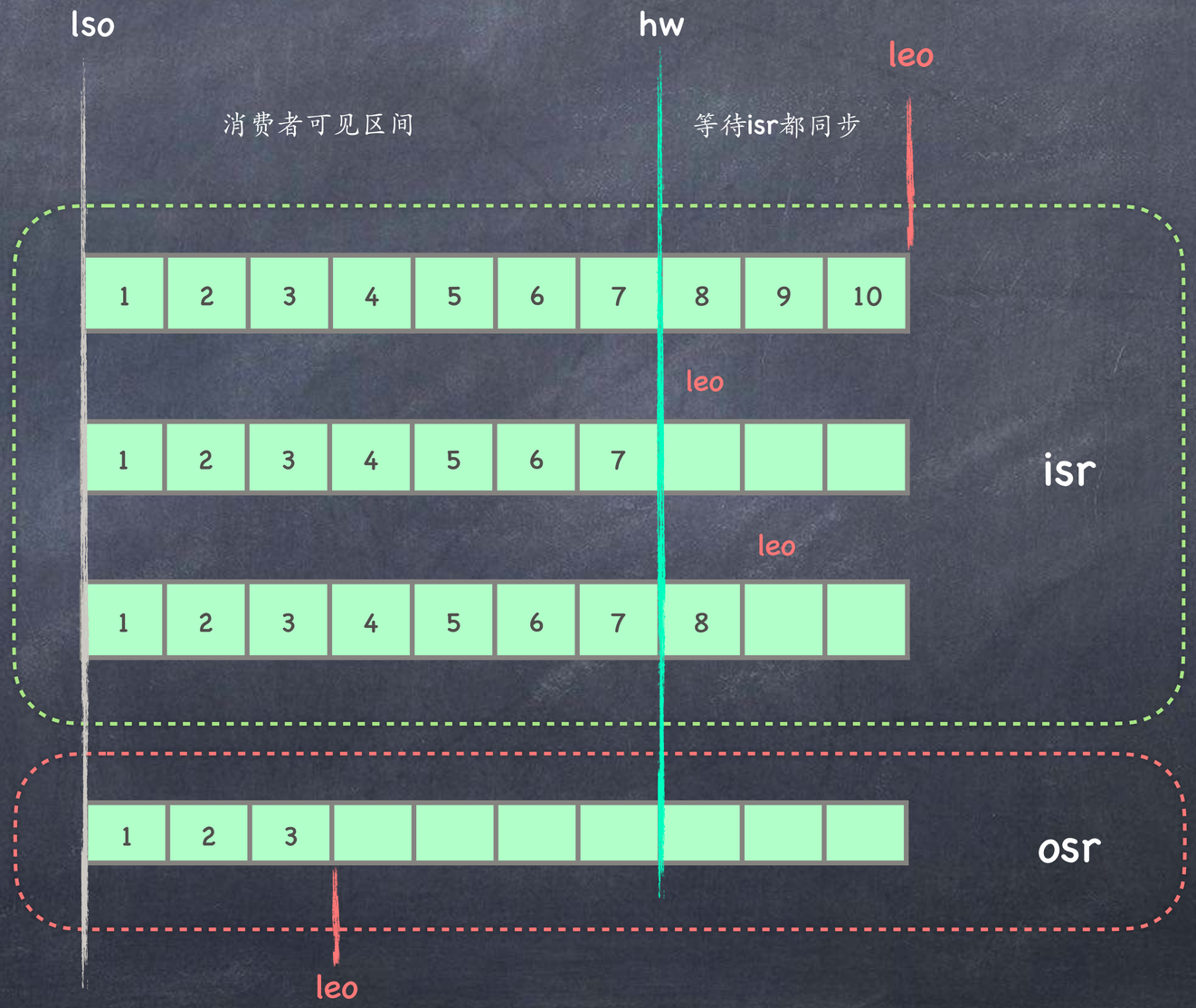
副本机制

- 多副本的好处
  - 提高可用性 ！
- 副本设计
  - leader 负责读写，follower 不能进行对外的读写
  - follower 只是同步 leader 数据，不能配合 leader 读写分离
  - leader 挂了会从 AR 集合中选举 leader
    - unclean.leader.election.enable = true
      - isr 为空也能选举
    - unclean.leader.election.enable = false (默认)
      - 只能从 isr 选举



副本机制

- LSO
  - 消费者只能消费到LSO(LastStableOffset)的位置
- HW 高水位
  - 消费者只能拉取到这个offset之前的消息
  - ISR集合中最小的 LEO 即为分区的 HW
- LEO （Log End Offset）
  - 表示了当前日志文件中下一条待写入消息的offset
  - ISR 集合中的每个副本都会维护自身的 LEO
- LW 低水位
  - 表示 AR 集合中最小的 logStartOffset 值



coordinator

1. 每个 consumer 都会发送 JoinGroup 请求到 coordinator 服务上 .
2. coordinator 从一个 consumer group 中取出一个 consumer 作为 leader coordinator把consumer group 情况发送给这个 leader
3. leader 会负责制定消费方案
4. 通过 SyncGroup 发送给 coordinator
5. 接着 coordinator 就把消费方案下发给所有的 consumer，他们会从指定的分区的 leader broker 开始进行 socket 连接和进行消息的消费

- 每个 kafka broker 都有一个 group coordinator 服务
- 目的
  - offset 位移管理
  - consumer rebalance
- how to select coordinator ?
  - __consumer_offsets 默认为 50 分区
  - “group.id” .hash() % offsets.topic.num.partitions
  - 再通过分区找到对应broker的coordinator

每个 consumer group 都会选择一个 coordinator
他负责监控整个消费组里的各个分区的心跳，以及判断是否宕机和开启rebalance .

kafka

rebalance

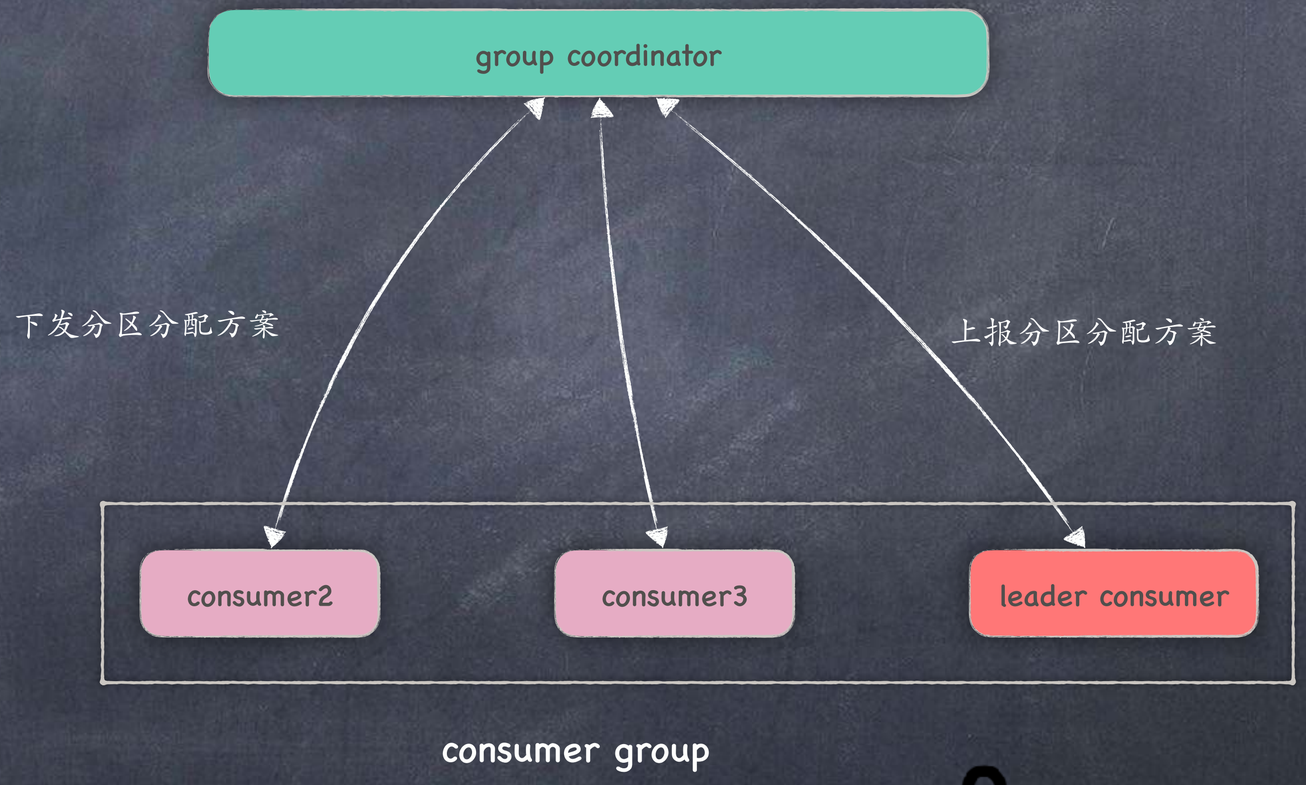
- 避免频繁 rebalance ?
  - session.timeout.ms 超时时间
    - 默认 10 s
  - heartbeat.interval.ms 心跳间隔
    - 默认 3 s
    - session.timeout.ms >= 3 * heartbeat.interval.ms
  - max.poll.interval.ms
    - 默认 5 min
    - 每隔多长时间去拉取消息，超过阈值则被 coordinator 判定死忙，被踢出 consumer group，进一步 rebalance

- 目的
  - 重新均衡消费者，尽量达到最公平的分配
- 触发时机
  - 组成员数量发生变化
  - 订阅主题数量发生变化
  - 订阅主题的分区数发生变化
- 影响
  - 可能重复消费
    - consumer还未提交offset，被rebalance
  - 集群不稳定
    - 只有做变动，就发生一次 rebalance
  - 影响消费速度
    - 频繁的 rebalance 影响消费速度

kafka

rebalance

- coordinator 会选定一个消费者为 leader, 收集所有成员的订阅信息，然后根据这些信息，制定具体的分区消费分配方案 .
- 通常第一个发送 JoinGroup 请求的消费者提升为 leader .
- 流程
  - join group (消费者加入组)
    - 由 coordinator 选定为主或通告谁是主, 及当前的消费者集合
    - 上报自己要消费的 topic
  - sync group (等待领导者分配方案)
    - 接收由 coordinator下发的leader的分配方案



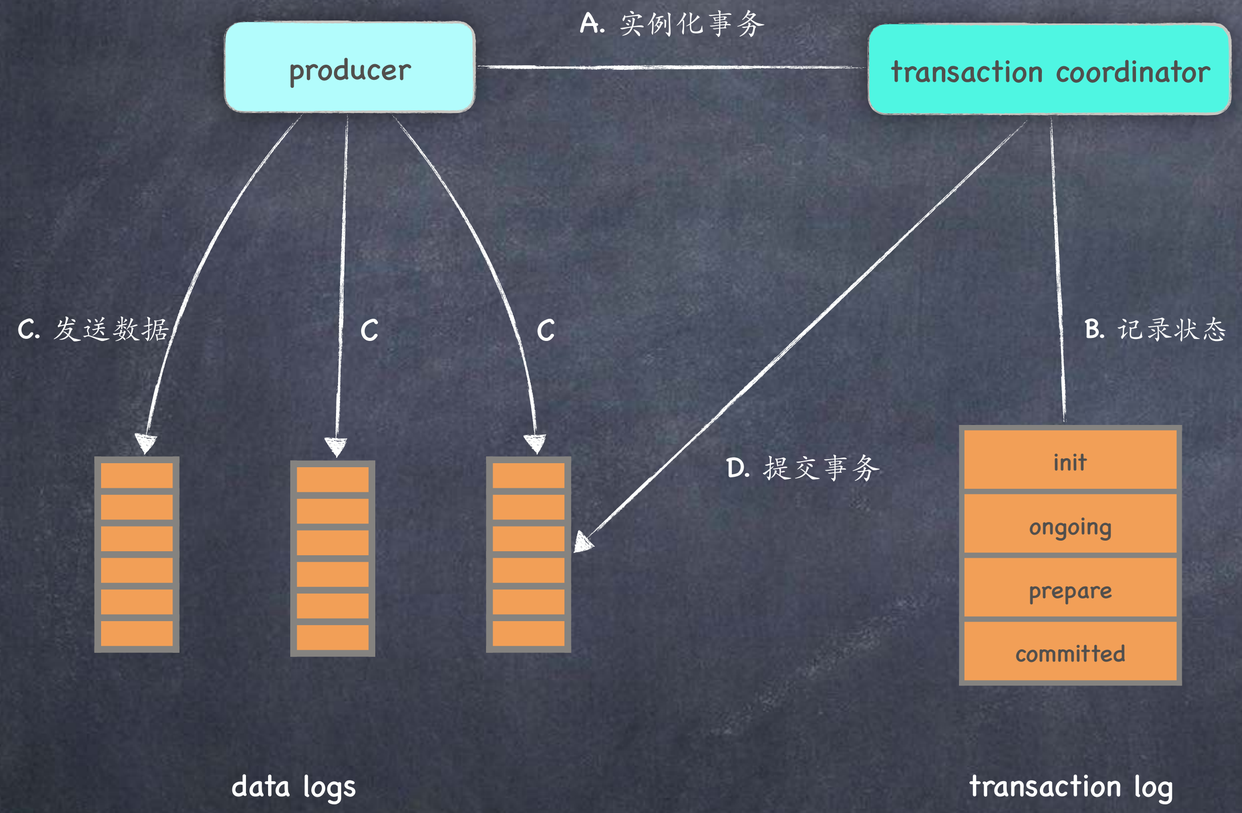
kafka

消息传递语义

- At most once 最多一次
  - 消息可能会丢，但绝不会重复传递
- At least once 最少一次
  - 消息绝不会丢，但可能会重复传递
- Exactly once 恰好一次
  - 每条消息只会被精确地传递一次，既不会多，也不会少

kafka

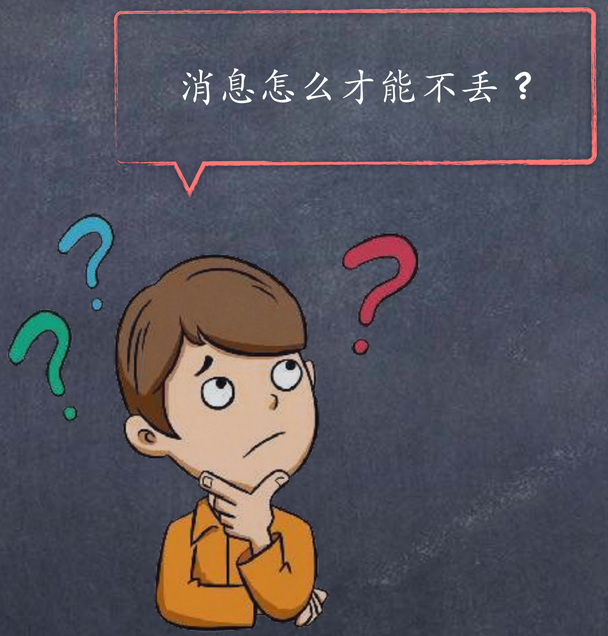
事务消息




```java
KafkaProducer producer = createKafkaProducer(
  "bootstrap.servers", "vps.xiaorui.cc:9092",
  "transactional.id", "my-transactional-id");

producer.initTransactions();  // 初始化事务
producer.beginTransaction();  // 开始事务
producer.send("outputTopic", "message1");
producer.send("outputTopic", "message2");
producer.commitTransaction(); // 提交事务
producer.abortTransaction();  // 回滚事务
```

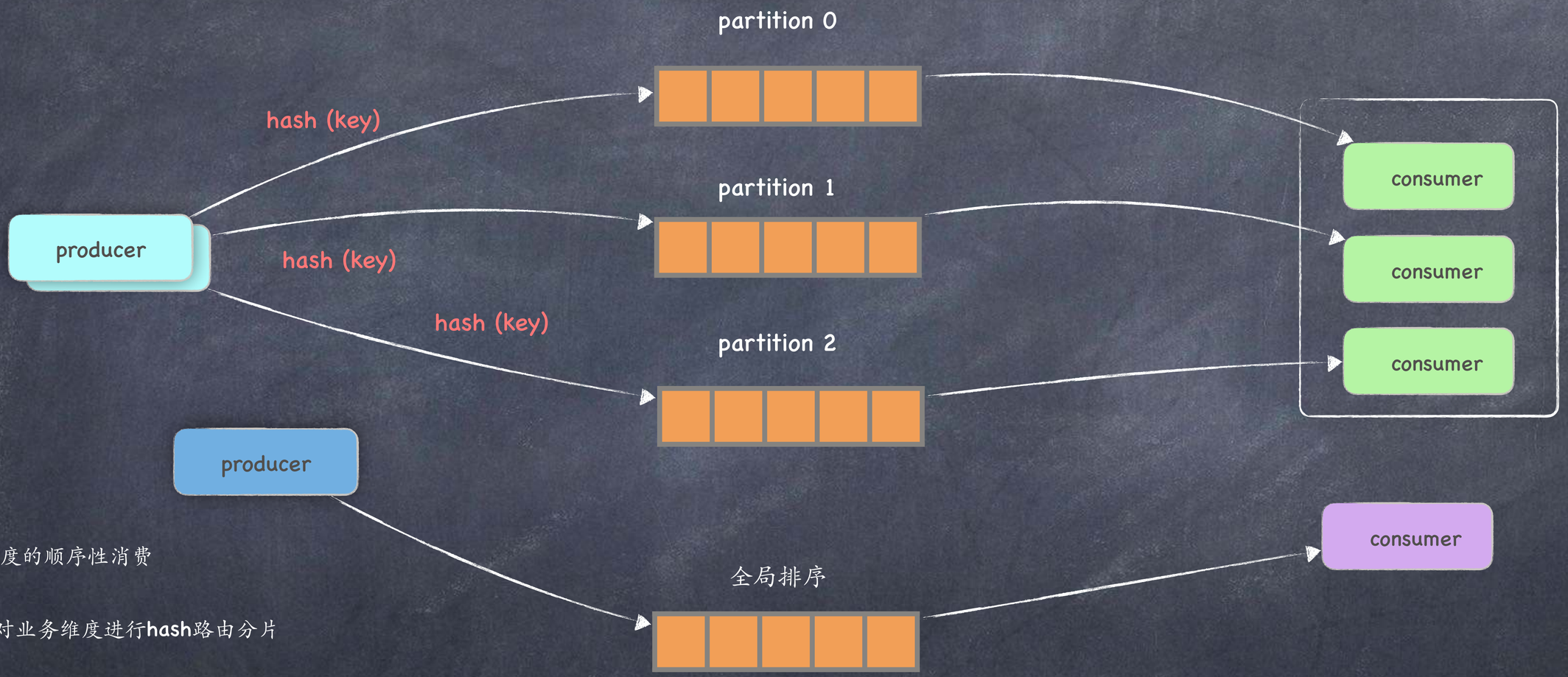
通过事务发送多分区消息，要么成功，要么失败 !!!

消息的可靠性



- producer
  - acks = all
  - 同步数据到 isr 副本
- kafka broker
  - min.insync.replicas
  - 落盘策略，使脏页就是落盘
- consumer
  - 改为手动提交

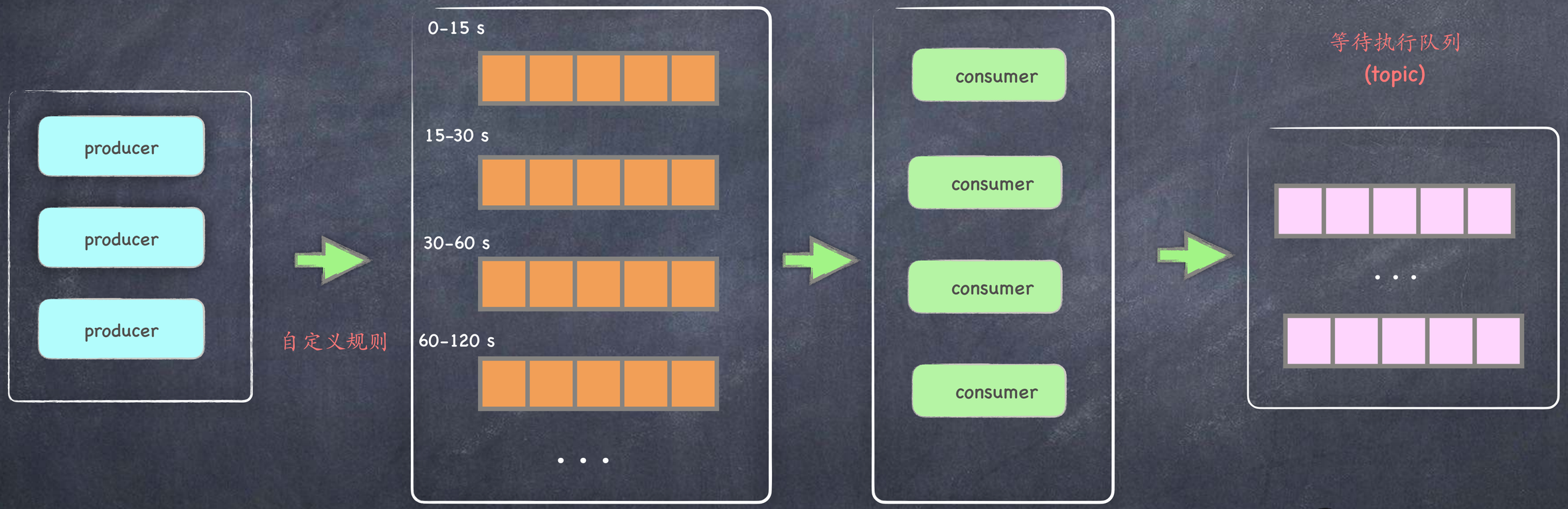
消息的顺序性



- 业务维度的顺序性消费
  - 对业务维度进行hash路由分片
- 全局顺序性消费
  - 设定单个 partition 分片

kafka

延迟队列


producer
producer
producer
自定义规则
0-15 s
15-30 s
30-60 s
60-120 s
consumer
consumer
consumer
consumer
等待执行队列
(topic)

kafka

5

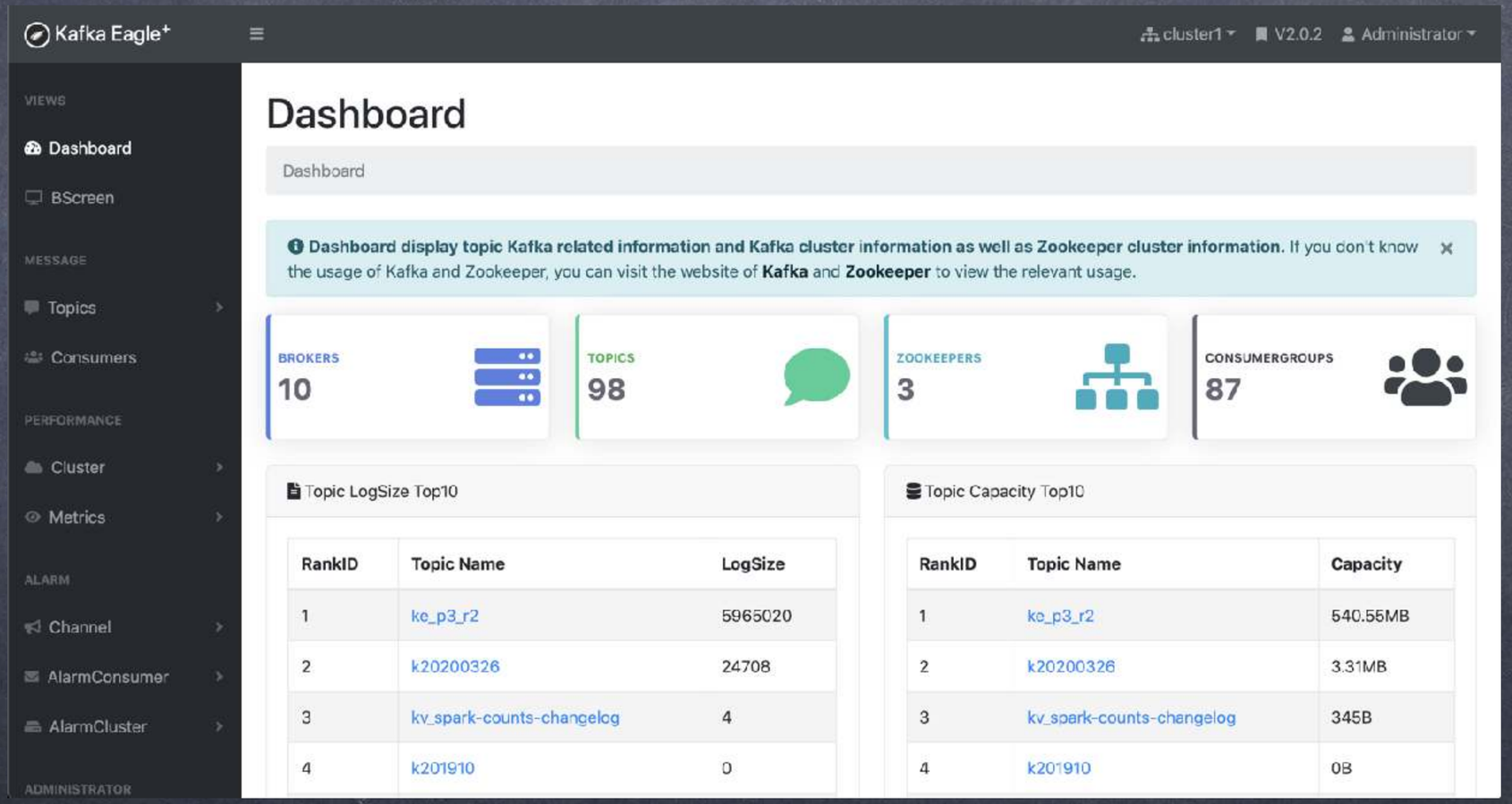
ops

tools

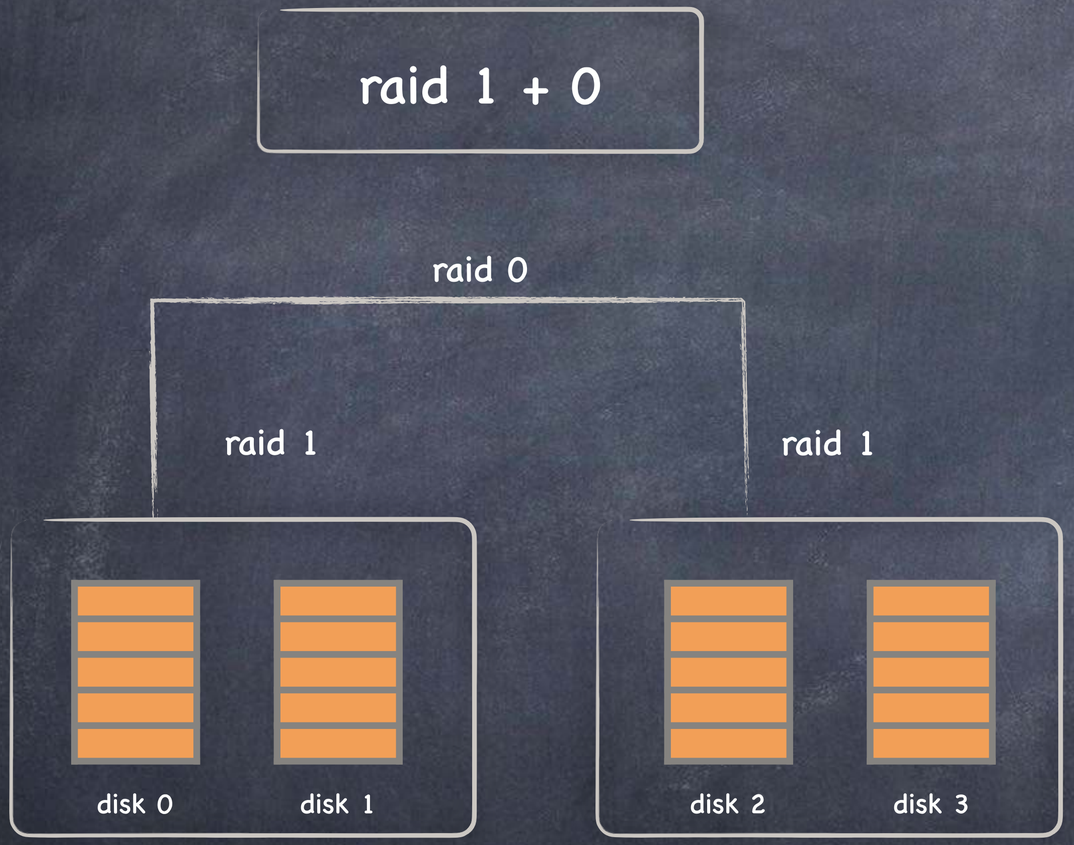
- kafka-manager
- kafka Eagle
- 自研监控面板
- ...

- topic 管理
  - bin/kafka-topics.sh
  - 增删改查
- 消费组管理
  - bin/kafka-consumer-groups.sh
- 生产消息
  - bin/kafka-console-producer.sh
- 消费消息
  - bin/kafka-console-consumer.sh
- ...

tools


Kafka Eagle+
cluster1
V2.0.2
Administrator
VIEWS
Dashboard
BScreen
MESSAGE
Topics
Consumers
PERFORMANCE
Cluster
Metrics
ALARM
Channel
AlarmConsumer
AlarmCluster
ADMINISTRATOR
Dashboard
Dashboard
Dashboard display topic Kafka related information and Kafka cluster information as well as Zookeeper cluster information. If you don't know the usage of Kafka and Zookeeper, you can visit the website of Kafka and Zookeeper to view the relevant usage.
BROKERS
10
TOPICS
98
ZOOKEEPERS
3
CONSUMERGROUPS
87
Topic LogSize Top10
RankID
Topic Name
LogSize
1
ke_p3_r2
5965020
2
k20200326
24708
3
kv_spark-counts-changelog
4
4
k201910
0
Topic Capacity Top10
RankID
Topic Name
Capacity
1
ke_p3_r2
540.55MB
2
k20200326
3.31MB
3
kv_spark-counts-changelog
345B
4
k201910
0B

存储规划



- raid 10
  - raid 0 + raid 1
  - Linkedin Kafka 也是该方案
  - 资源利用率低，单机在 50 %
  - 如两个副本，则有4倍的冗余
- JBOD
  - log.dirs = /data1, /data2, ...
  - 磁盘利用率高
  - 利用 kafka failover 保证可用性

要么不用 RAID，要么用 RAID10

vs other mq

对比

RabbitMQ

kafka

Apache RocketMQ

PULSAR

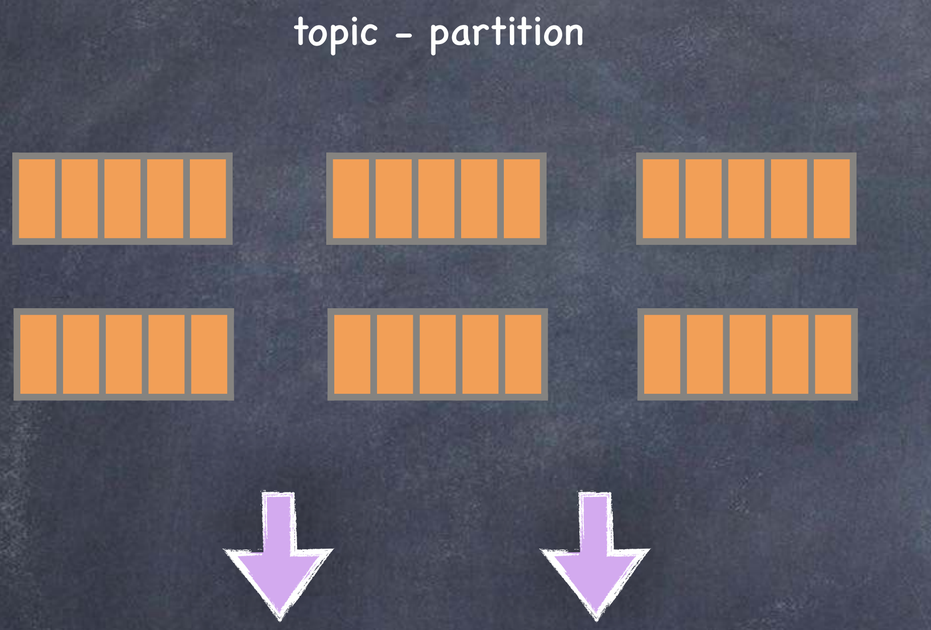
- rabbitmq 
- nsq
- nats streaming
- kafka
- rocketmq
- pulsar mq

rabbitmq

	rabbitmq	kafka
优先级队列	支持	可以改造
延迟队列 TTL	支持	可以改造
私信队列	支持	可以改造
重试队列	可以改造	可以改造
广播模式	支持	支持
回溯消息	不支持	支持
堆积持久化	支持	支持
消息轨迹	插件支持	可以改造
顺序性	不好支持	分区有序
生产幂等	不支持	支持
集群可用性	可以	很强
路由	可以	可以改造
....	....	....

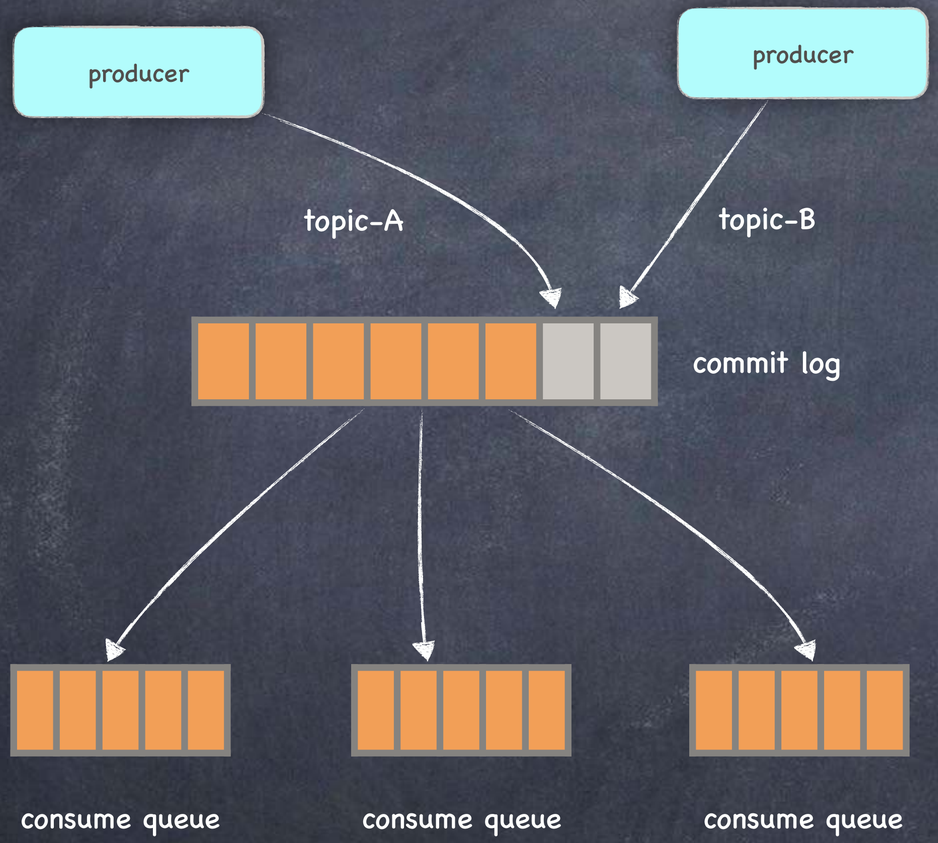
- rabbitmq 功能丰富，简单易用
- 集群性能
  - kafka 吞吐可压到 几十万 以上 .
  - rabbitmq 吞吐在 万 级别.

kafka 瓶颈



- kafka 如果分区太多？（单机超过 64 分区）
  - load 负载加大
  - 消息延迟也加大
- 一个分区一个文件。
- 单个文件按照 append 写是顺序 io。
- 分区多文件多，那么局部的顺序写会退化到随机 io。

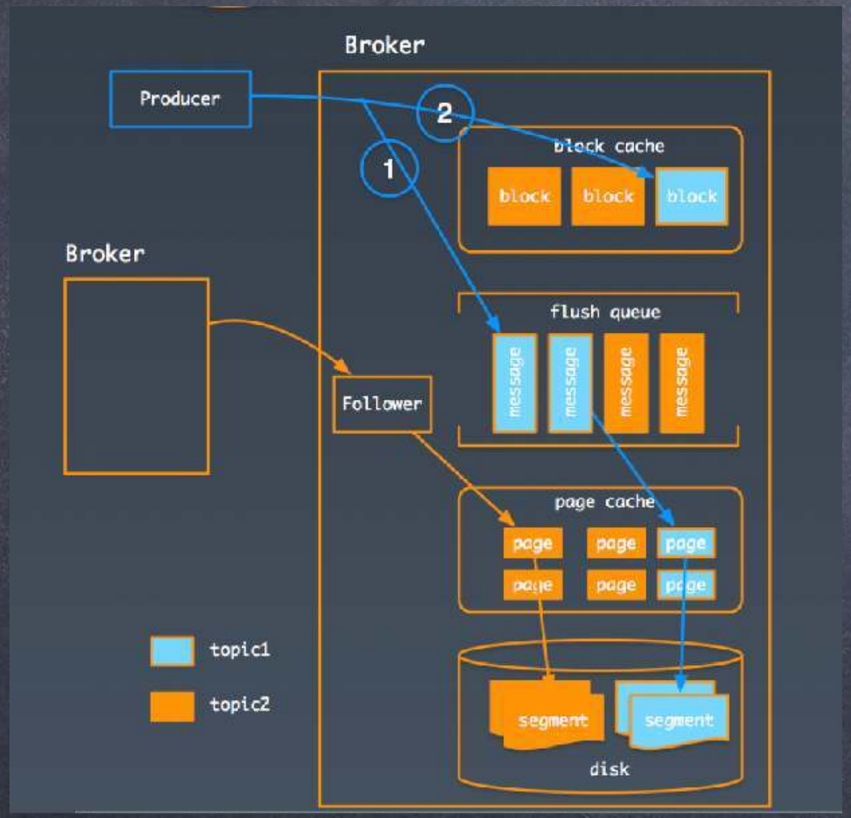
顺序 io 优化



- rocketmq
  - 消息数据在 commit log 文件中
  - consume queue存有位置信息
  - queue 每条数据仅占 20byte
  - consumerQueue切分多个，每个为5.8mb左右
  - queue内 30w 元素仅占用 5.8 mb 空间
  - msgID = broker IP + Port + CommitLog Offset

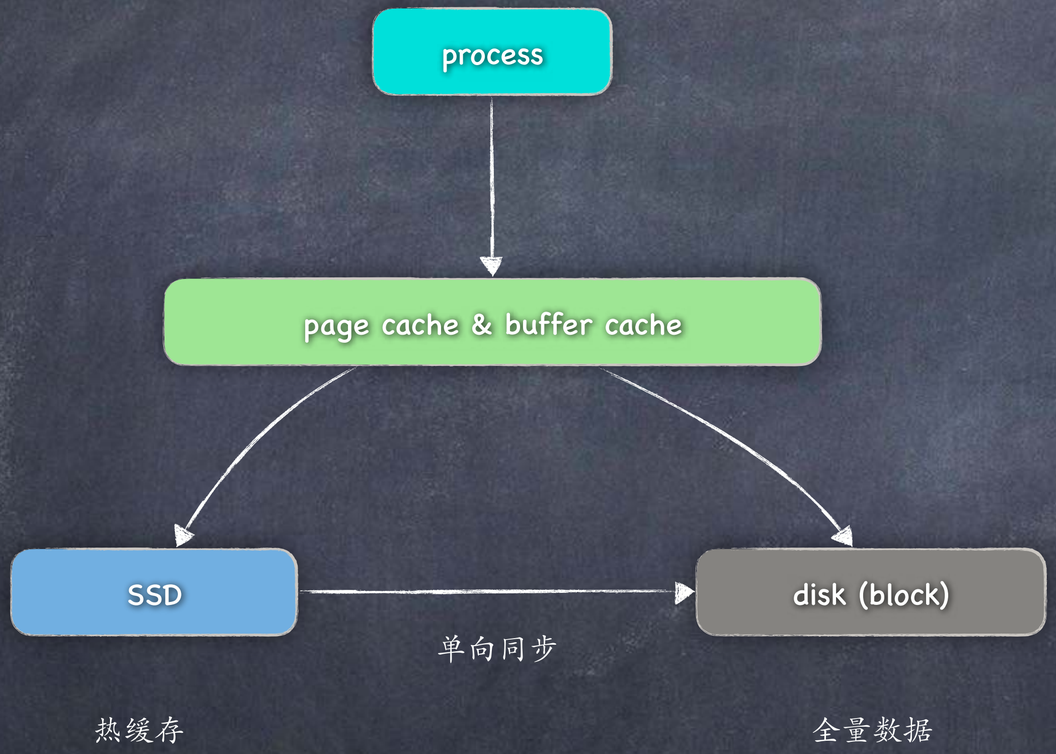
8byte	4byte	8byte
commitlog offset	size	tag hashcode

缓存污染优化



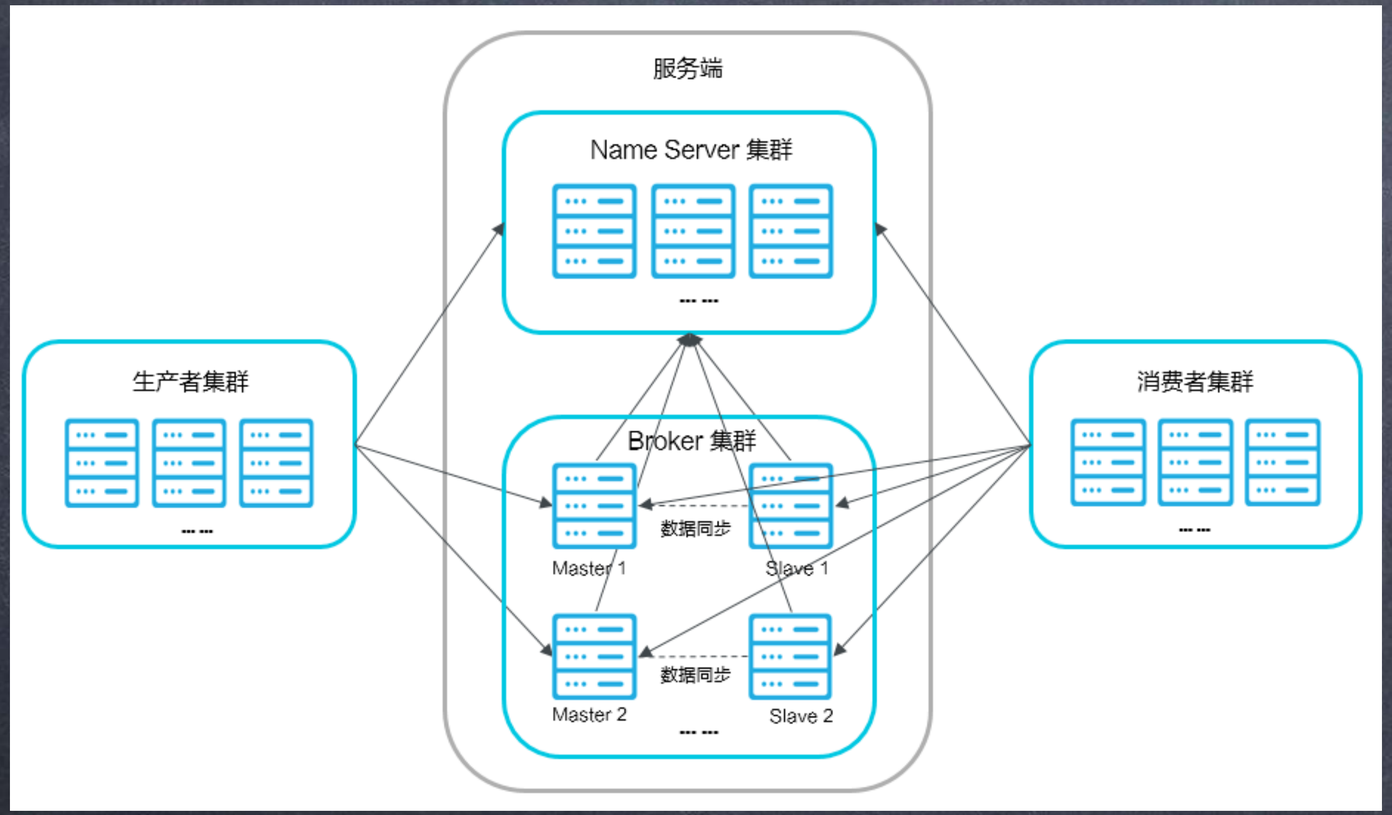
- 快手方案
- 造成污染的原因
  - broker 的 follower 的写操作造成缓存污染
  - consumer 旧数据读取挤掉新近的写缓存
- 同步写自定义cache，异步写磁盘 .
  - 自定义cache比 page cache 优先级高，更加自由可控 .
  - 减少触发同步写盘，减少时延 .
  - 缓存 producer 的写操作到缓存中
  - follower 还是以前操作
- 结论
  - 在绝大数是少量对接的数场景下， consumer 读基本都可命中缓存 .
  - 如 cache 没命中，则还是以前的套路，从 page cache 和 磁盘中读取 .

缓存污染优化



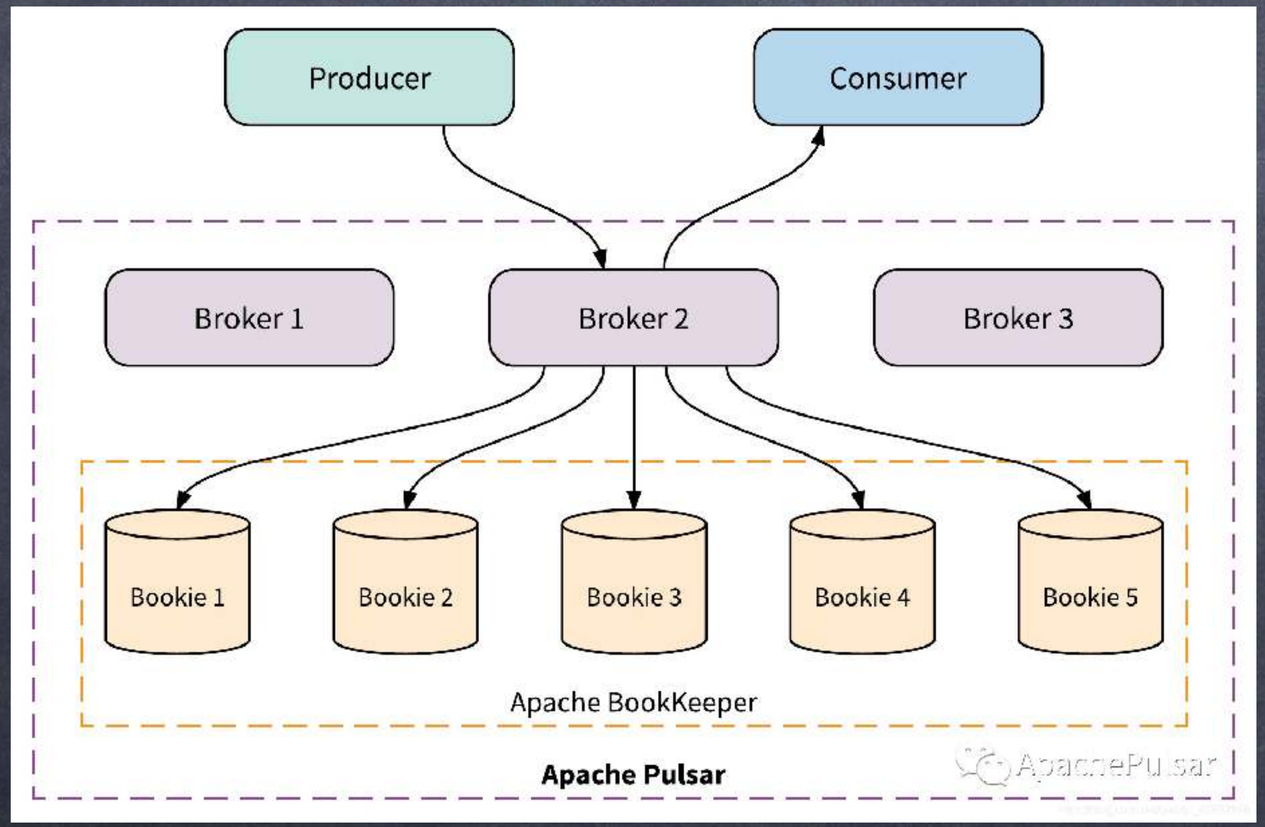
- 美团和京东方案
- 还是解决 page cache 不可控问题
- 利用 SSD 超高的IOPS做热缓存
- 方案
  - facebook flashcache
  - 自定义 ssd 缓存

rocketmq



- 收敛日志到commit log
- 事务消息
- 支持单机上万的队列数
- 消息过滤
- 消息查询，消息追踪
- 延迟消息
- ...

pulsar



- pulsar采用存储计算分离的架构
- 使用 bookkeeper 做消息的存储
- 支持多租户
- 支持多中心
- . . .

“ Q&A ”

- xiaorui.cc