

TGIP-31

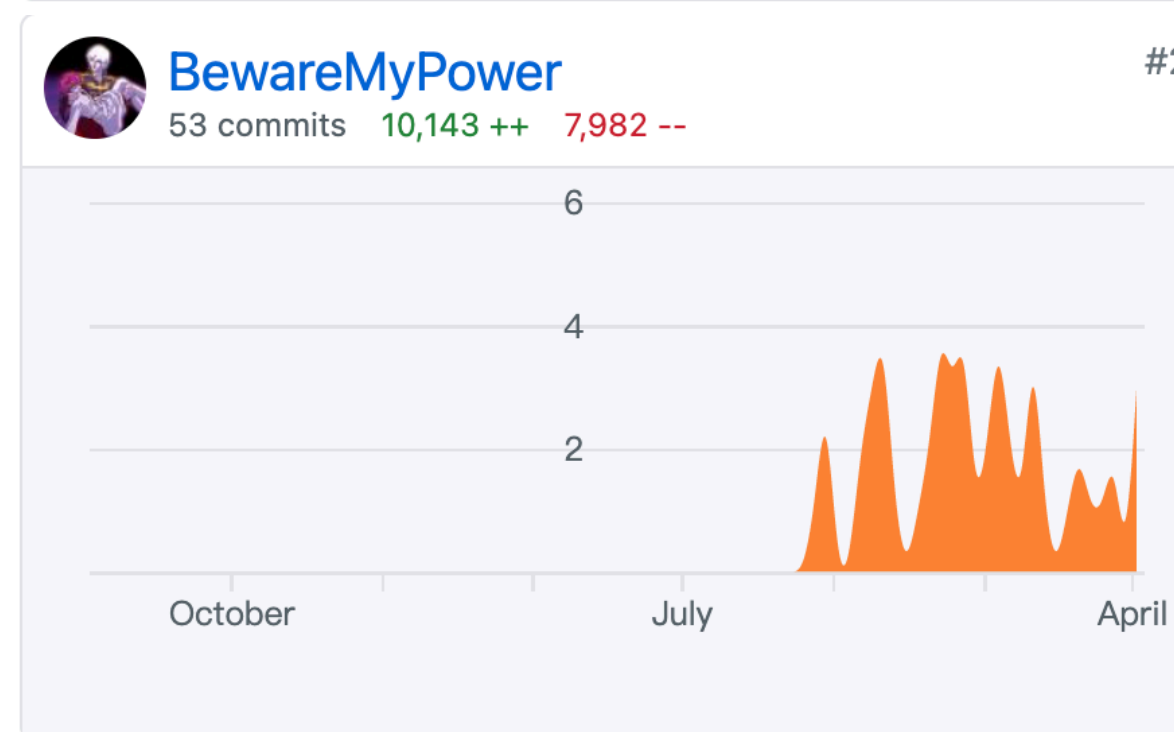
KoP 2.8.0 新特性前瞻

关于我

- StreamNative Software Engineer
- Apache Pulsar Contributor
- KoP Main Contributor

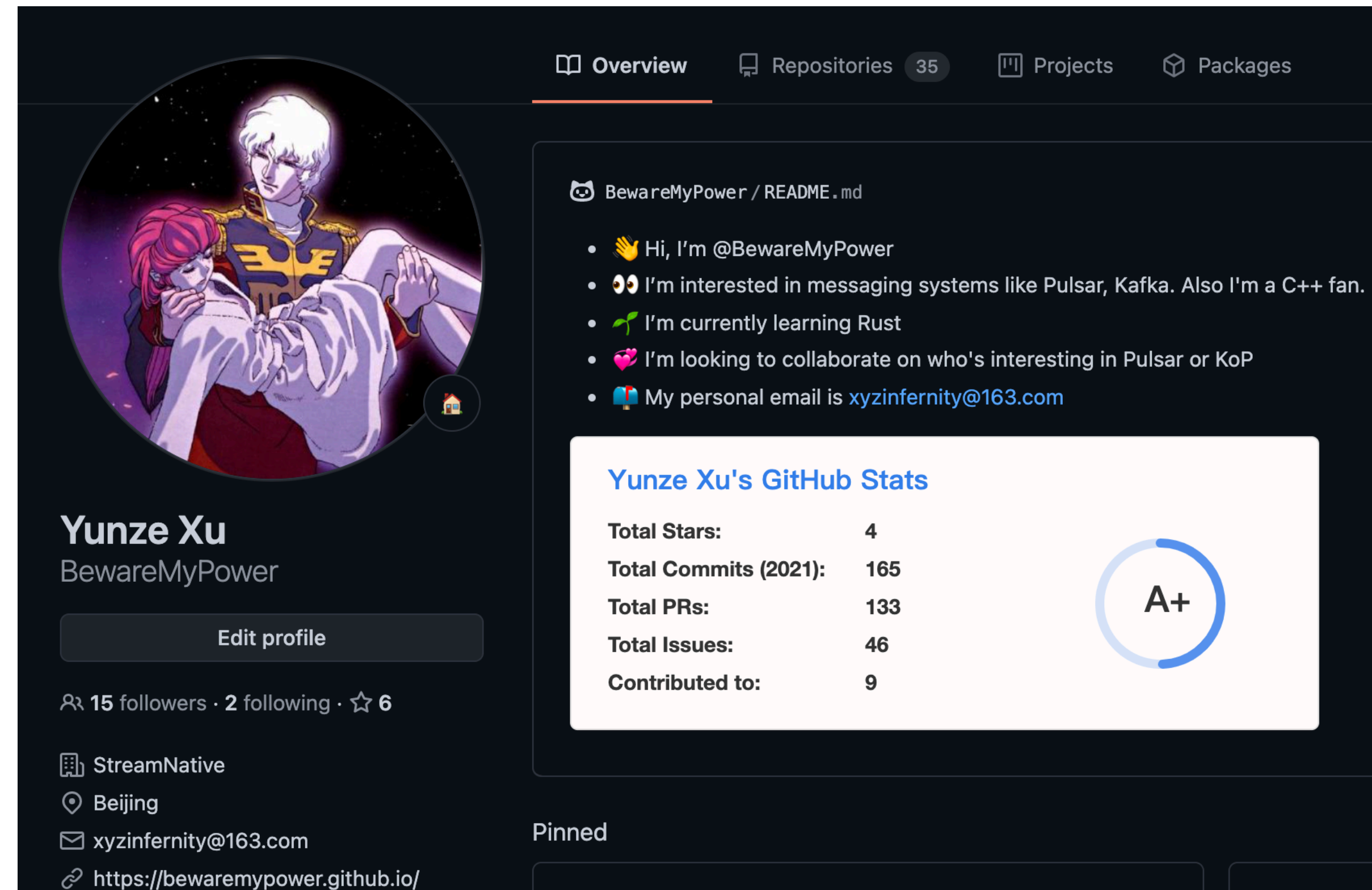


Pulsar



KoP

Until 2021/04/09



GitHub profile of Yunze Xu (BewareMyPower). The profile includes a bio, a list of interests, and a table of GitHub statistics.

Yunze Xu
BewareMyPower

[Edit profile](#)

15 followers · 2 following · 6 stars

StreamNative
Beijing
xyzinfernity@163.com
<https://bewaremypower.github.io/>

BewareMyPower / README.md

- 🙋 Hi, I'm @BewareMyPower
- 🐼 I'm interested in messaging systems like Pulsar, Kafka. Also I'm a C++ fan.
- 🌱 I'm currently learning Rust
- 💖 I'm looking to collaborate on who's interesting in Pulsar or KoP
- 📧 My personal email is xyzinfernity@163.com

Yunze Xu's GitHub Stats

Total Stars:	4
Total Commits (2021):	165
Total PRs:	133
Total Issues:	46
Contributed to:	9

A+

KoP 版本号

Apache Pulsar:

- Major release, 比如 2.7.0
- Minor release, 比如 2.7.1

StreamNative Pulsar (<https://github.com/streamnative/pulsar>)

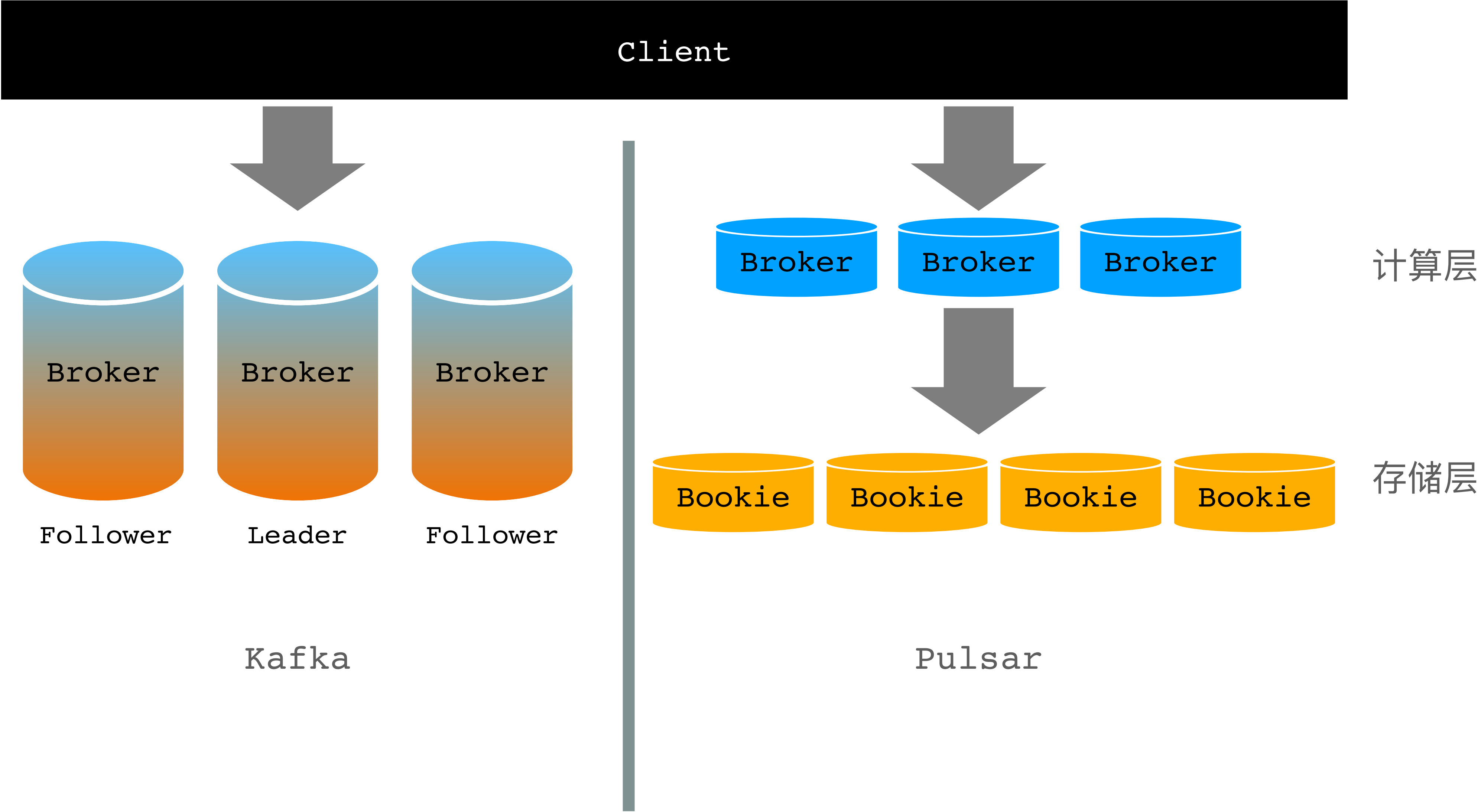
- Daily release, 比如 2.8.0-rc-202104032208
- Weekly release, 比如 2.7.1.2

从 2.6.2.0 起, KoP 的版本号和 Pulsar 一致, master 分支会不定期更新依赖的 SN Pulsar。

概览

1. 为什么需要 KoP?
2. KoP 的基本实现。
3. KoP 2.8.0-SNAPSHOT 版本的近期进展。
4. 近期计划

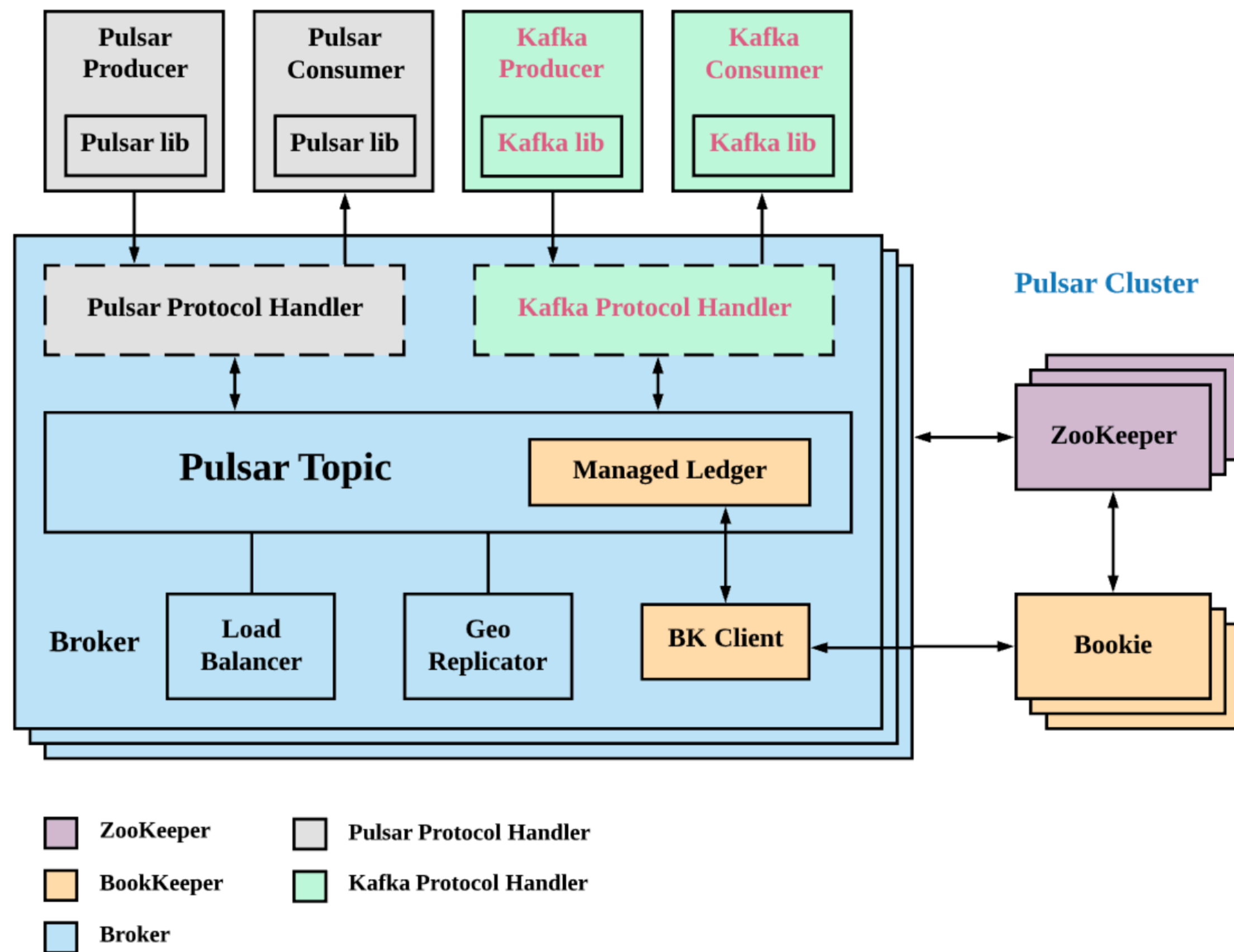
Kafka vs Pulsar



从 Kafka 迁移到 Pulsar

- 推动业务更换客户端?
 - 麻烦。
 - Pulsar adaptors? 看起来不错，可惜我不是用的 Java 客户端。
 - 我不嫌麻烦，但我只会 PHP。
- 用户直接使用了 Kafka 连接器（近百种）连接到外部系统怎么办？
- 用户使用外部系统的连接器连接到 Kafka 怎么办？

KoP (Kafka on Pulsar)



Pulsar Architecture from 2.5.0

如何使用?

1. 将 Protocol Handler 的 NAR 包放入 protocols 目录
2. 对 broker.conf 或 standalone.conf 添加相应配置

```
messagingProtocols=kafka  
# Protocol handler's configs  
# ...
```

支持的客户端:

- Java >= 1.0
- C/C++: librdkafka
- Golang: sarama
- NodeJS:
- 其他基于 rdkafka 的客户端

Protocol Handler

```
public interface ProtocolHandler extends AutoCloseable {

    /** Returns the unique protocol name. For example, `kafka-v2` for protocol handler for Kafka v2 protocol. */
    String protocolName();

    /** Verify if the protocol can speak the given <tt>protocol</tt>. ...*/
    boolean accept(String protocol);

    /** Initialize the protocol handler when the protocol is constructed from reflection. ...*/
    void initialize(ServiceConfiguration conf) throws Exception;

    /** Retrieve the protocol related data to advertise as part of ...*/
    String getProtocolDataToAdvertise();

    /** Start the protocol handler with the provided broker service. ...*/
    void start(BrokerService service);

    /** Create the list of channel initializers for the ports that this protocol handler ...*/
    Map<InetSocketAddress, ChannelInitializer<SocketChannel>> newChannelInitializers();

    @Override
    void close();
}
```

BrokerService 掌控每个 broker 的一切资源

- 连接的 producers, subscriptions
- 持有的 topic 及其对应的 managed ledgers
- 内置的 admin 和 client
- ...

Broker 启动流程



Topic & Partition

Kafka

```
/**
 * A topic name and partition number
 */
public final class TopicPartition implements Serializable {
    private static final long serialVersionUID = -613627415771699627L;

    private int hash = 0;
    private final int partition;
    private final String topic;
```

Pulsar

`persistent://public/default/my-topic-partition-0`

- 是否持久化
- 租户
- 命名空间
- 主题
- 分区编号

Topic & Partition

```
# 默认租户
kafkaTenant=public
# 默认命名空间
kafkaNamespace=default
# 禁止自动创建 non-partitioned topic
allowAutoTopicCreationType=partitioned
```

- my-topic => persistent://public/default/my-topic
- Tenant-0/Namespace-0/Topic-0 => persistent://Tenant-0/Namespace-0/Topic-0
- xxx/my-topic => **invalid topic name**
- persistent://Tenant-0/Namespace-0/Topic-0 => persistent://Tenant-0/Namespace-0/Topic-0

```
/**
 * KopTopic maintains two topic name, one is the original topic name, the other is the full topic name used in Pulsar.
 * We shouldn't use the original topic name directly in KoP source code. Instead, we should
 * 1. getOriginalName() when read a Kafka request from client or write a Kafka response to client.
 * 2. getFullName() when access Pulsar resources.
 */
public class KopTopic {

    @Getter
    private final String originalName;
    @Getter
    private final String fullName;
```

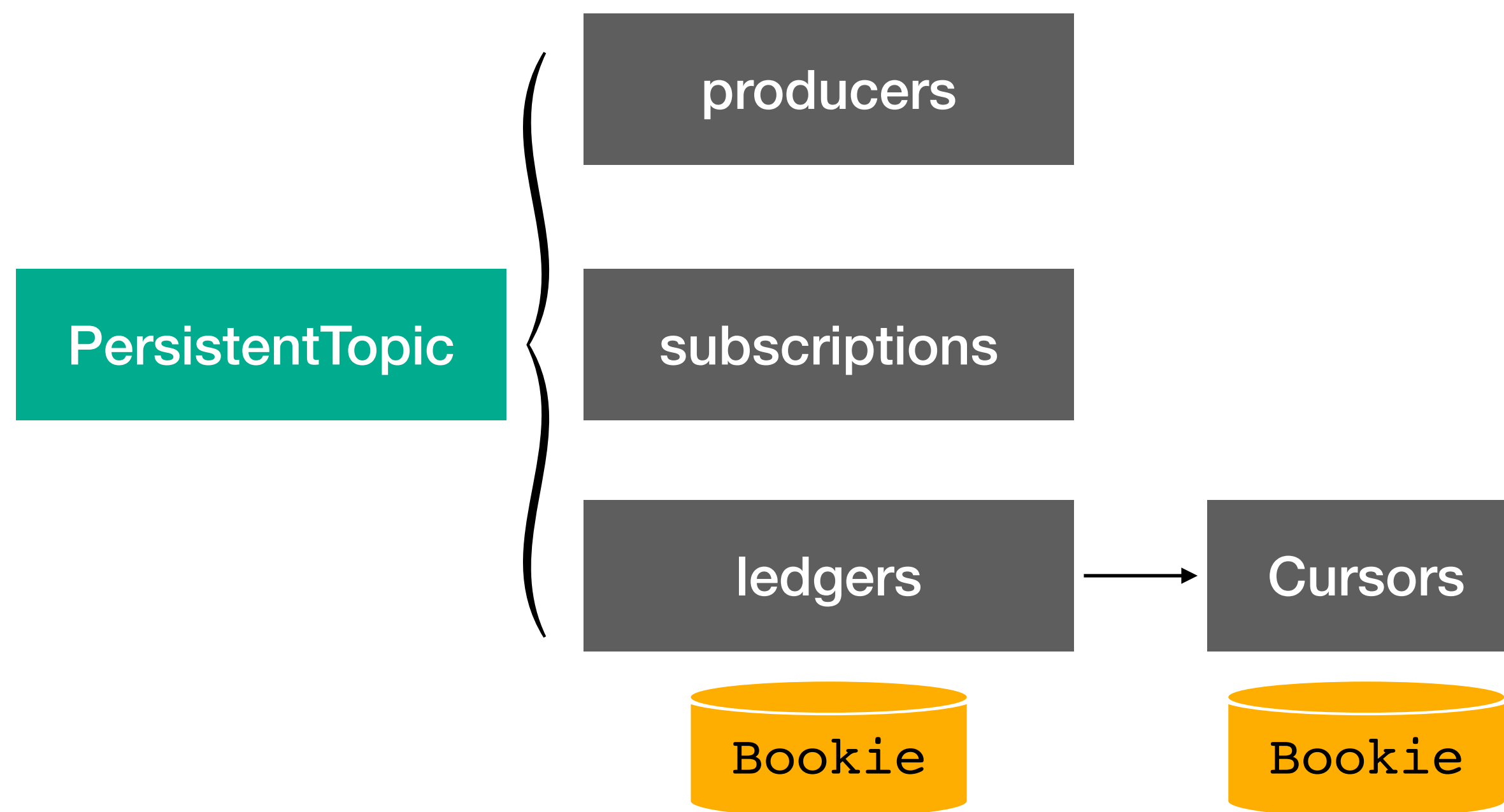
Produce & Fetch 请求

PRODUCE 请求:

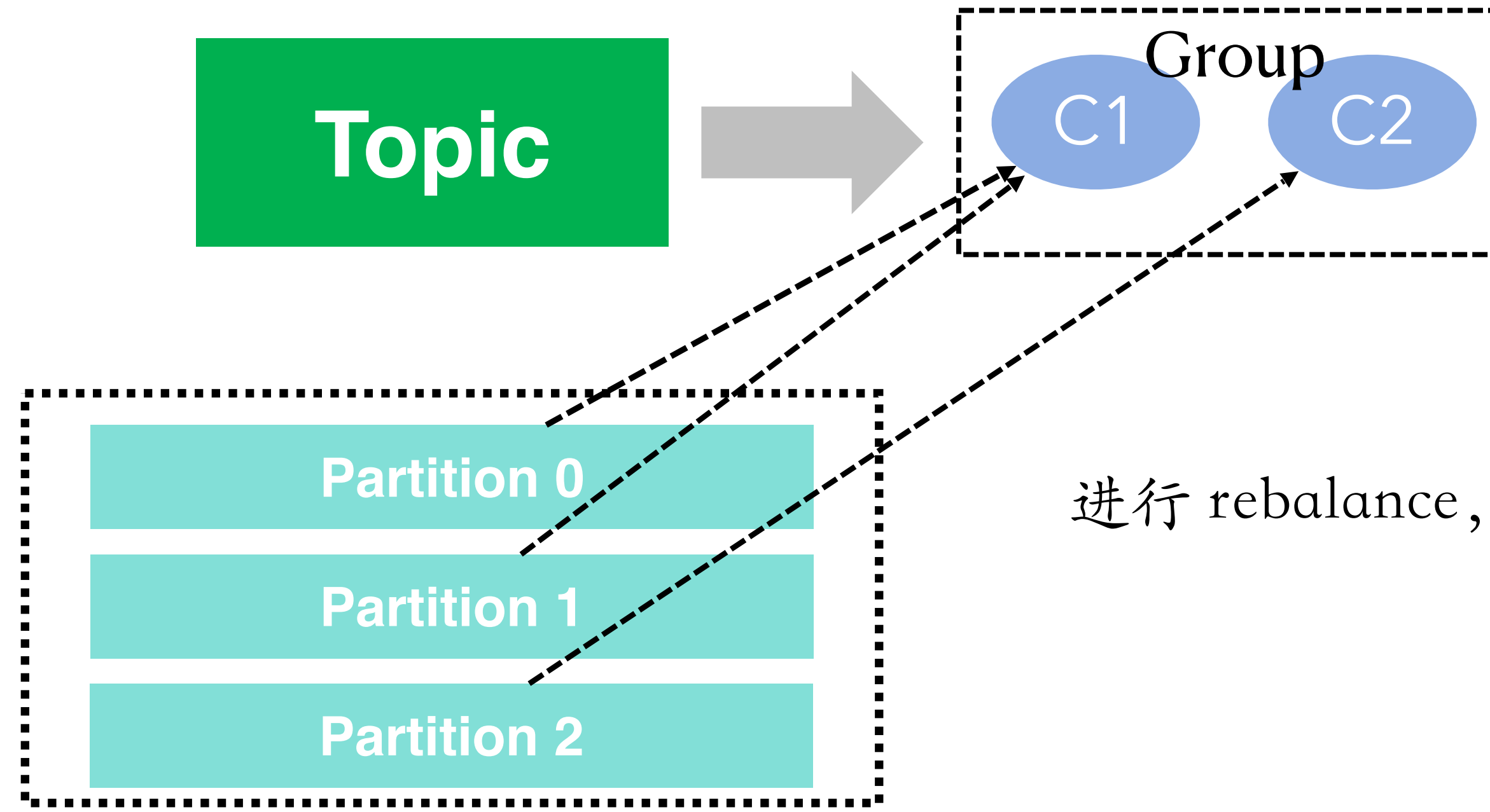
1. 通过 topic 名字找到 PersistentTopic 对象 (内含 **ManagedLedger**) 。
2. 对消息格式进行转换。
3. 异步写入消息到 bookie。

FETCH 请求:

1. 通过 topic 名字找到 PersistentTopic 对象。
2. 通过 offset 找到对应的 **ManagedCursor**。
3. 从 ManagedCursor 对应位置读取 entry。
4. 对 entry 格式进行转换后将消息返回给客户端。



Group Coordinator



进行 rebalance, 决定 partition 和 group 的映射关系。

当 consumer 加入（订阅）一个 group 时：

1. 发送 JoinGroup 请求，通知 broker 有新的消费者加入。
2. 发送 SyncGroup 请求用于 partition 的分配。

Group Coordinator

特殊 topic: __consumer_offsets

```
kafkaMetadataTenant="public"
kafkaMetadataNamespace="__kafka"
offsetsTopicNumPartitions=8
```

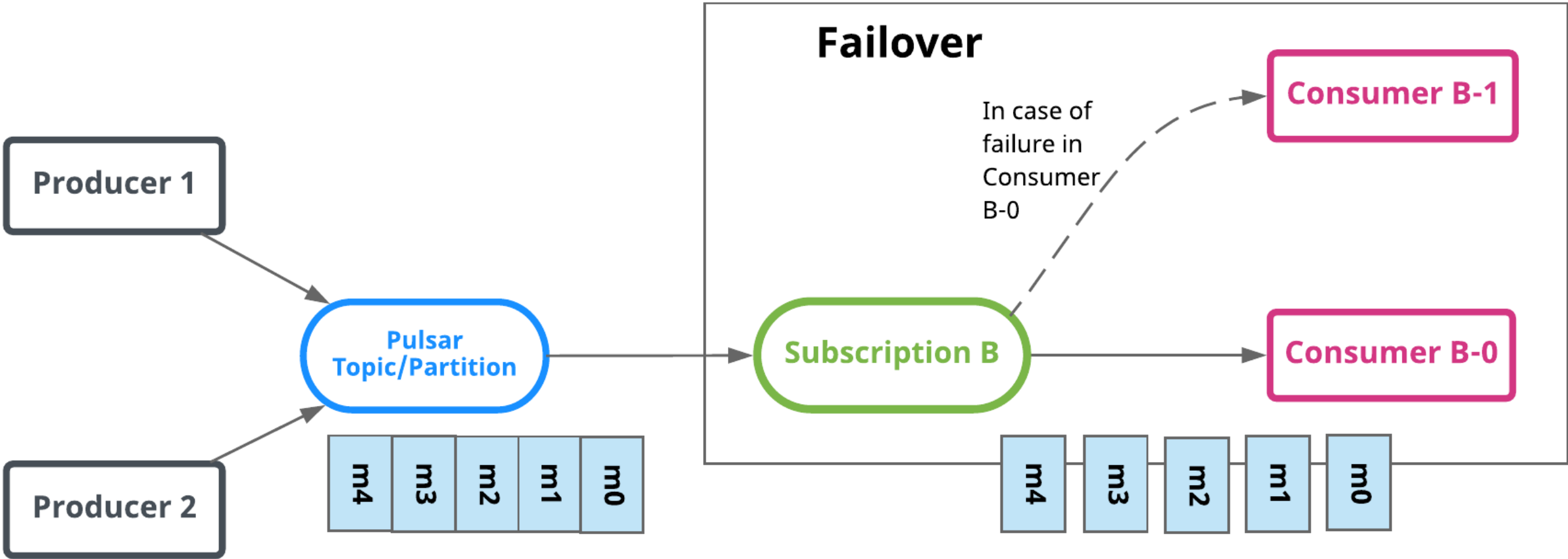
GroupMetadataManager

offsetProducers

offsetReaders

Write/Read __consumer_offsets

Kafka group 等价于 Pulsar Failover subscription

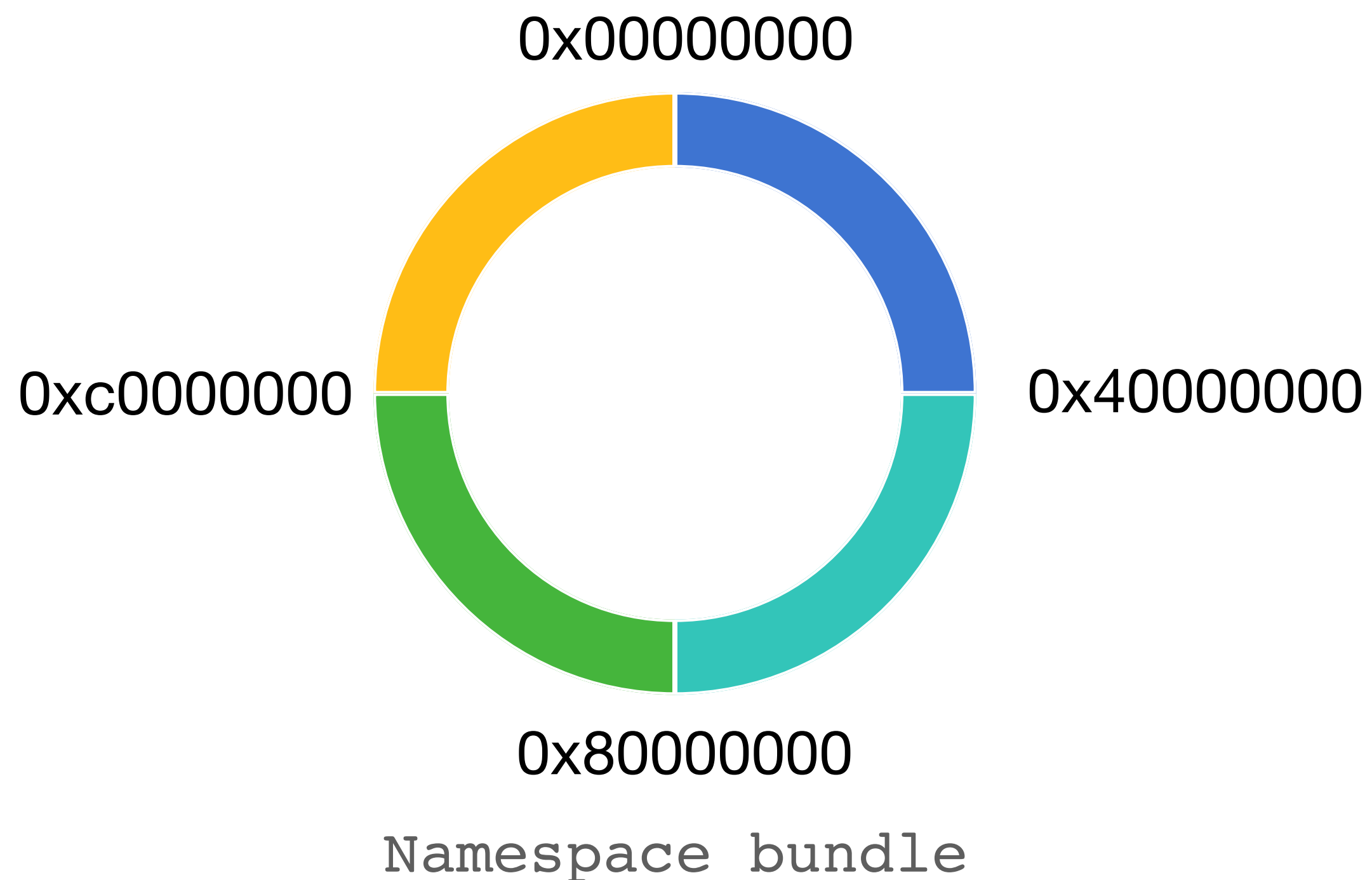


OffsetAcker

consumers

Acknowledge cumulatively

Group Coordinator



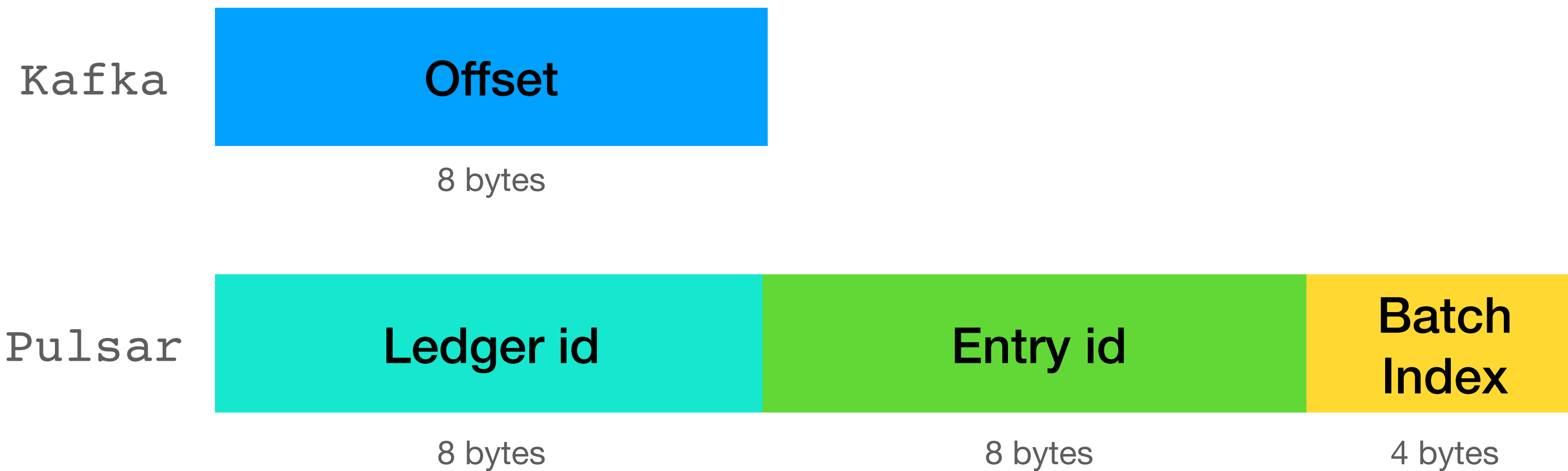
- 每台 broker 都拥有 (own) 一些 bundle range。
- Topic 会按名字哈希到其中一个 bundle range, 这个 range 的 owner broker 就是 topic 的 owner broker。
- Bundle 可能分裂, broker 也有可能挂掉, 因此 bundle ownership 会发生改变。

因此, KoP 注册了 bundle ownership 的 listener 用于通知 group coordinator。

Kafka offset

Before KoP 2.8.0

- Kafka offset 是一个 64 位整型，用来标识消息存储的位置。
- Pulsar 使用 MessageId 来标识消息位置。



KoP 之前的分配策略: 20 bits + 32 bits + 12bits

Kafka offset

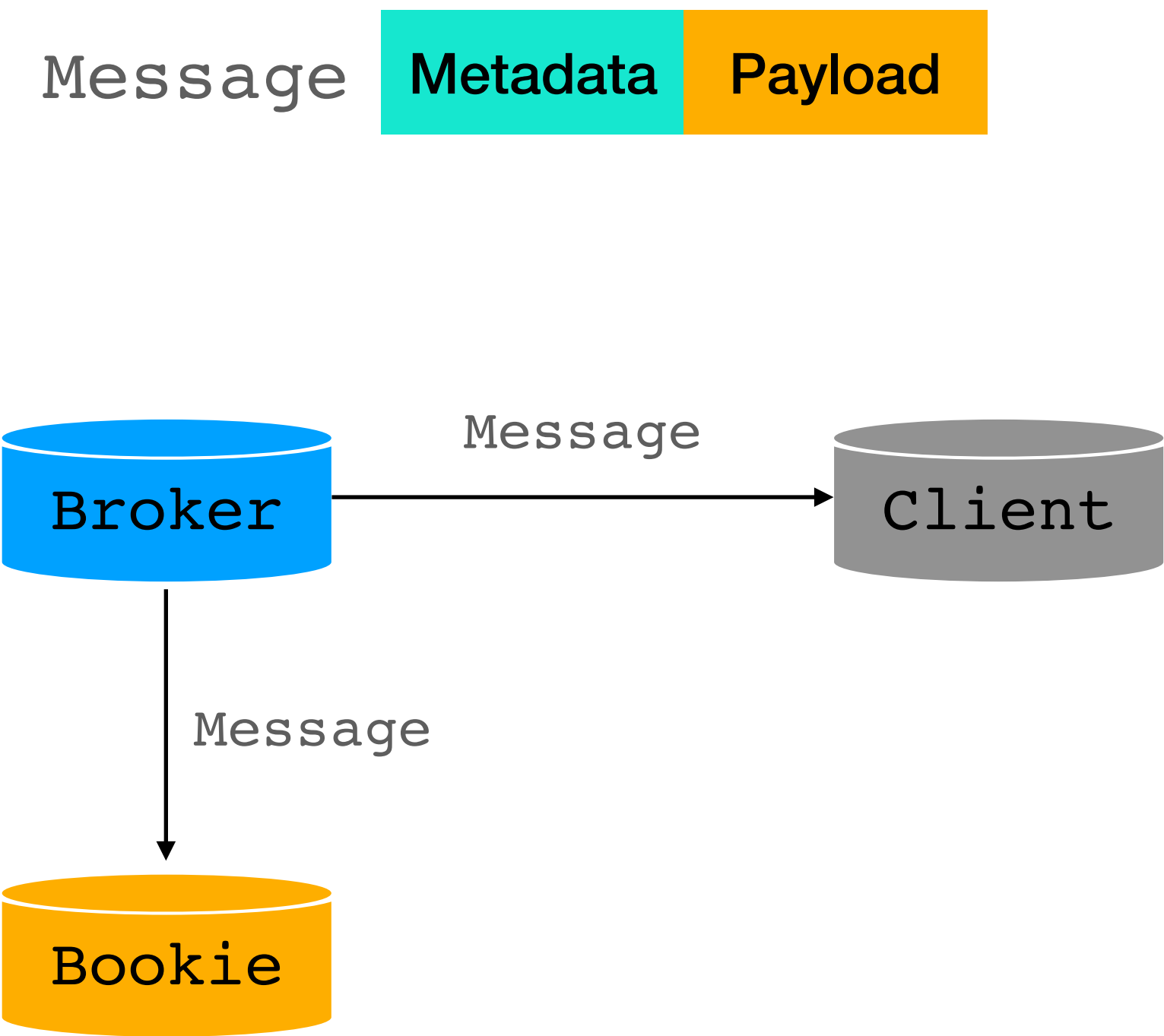
MessageId 拆分的问题

Offset (64 bits) = LedgerId (20 bits) + EntryId (32 bits) + BatchIndex (12bits)

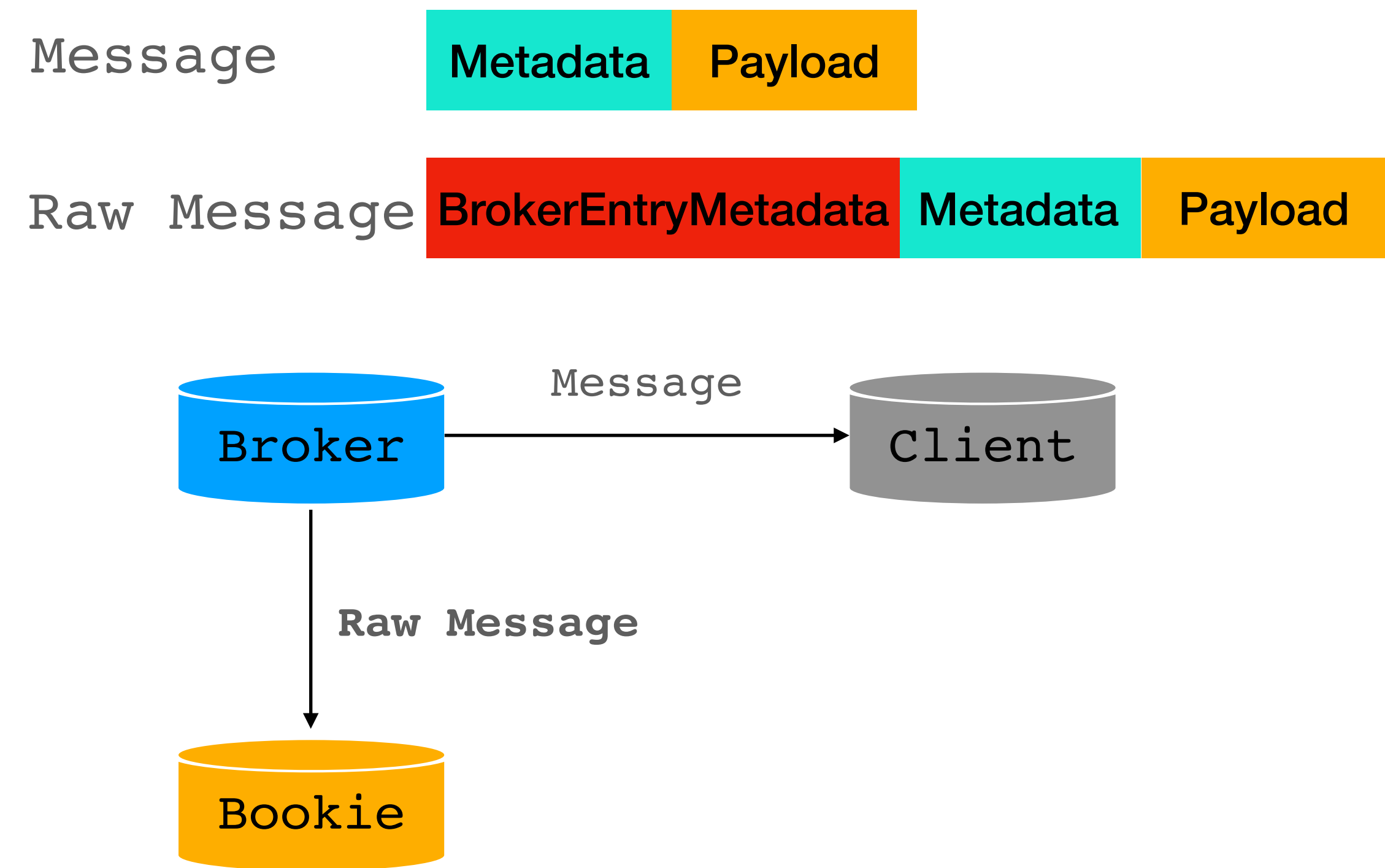
- 难以提出『合适』的分配方案
- 从 cursor 读取 entry 时只能一个一个读取，否则可能导致 Maximum offset delta exceeded
- 有些第三方组件（比如 Spark）依赖于连续 offset 的功能

Kafka offset

PIP 70 BrokerEntryMetadata



Before



After

Kafka offset

PIP 70 BrokerEntryMetadata

```
// metadata added for entry from broker
message BrokerEntryMetadata {
    optional uint64 broker_timestamp = 1;
    optional uint64 index = 2;
}
```

ManagedLedger 中使用 interceptor

```
private boolean beforeAddEntry(OpAddEntry addOperation) {
    // if no interceptor, just return true to make sure addOperation will be initiate()
    if (managedLedgerInterceptor == null) {
        return true;
    }
    try {
        managedLedgerInterceptor.beforeAddEntry(addOperation, addOperation.getNumberOfMessages());
        return true;
    }
}
```

Broker 中将 BrokerEntryMetadata 添加到 entry 首部，并传递给 BrokerEntryMetadataInterceptor

```
@Override
public OpAddEntry beforeAddEntry(OpAddEntry op, int numberOfMessages) {
    if (op == null || numberOfMessages <= 0) {
        return op;
    }
    op.setData(Commands.addBrokerEntryMetadata(op.getData(), brokerEntryMetadataInterceptors, numberOfMessages));
    return op;
}
```

KoP 2.8.0 连续 offset

基于 BrokerEntryMetadata 很容易实现连续 offset:

- FETCH 请求: 直接读 bookie, 解析 BrokerEntryMetadata 即可。
- PRODUCE 请求: 将 ManagedLedger 传入异步写 bookie 的上下文, 从 ManagedLedger 的 interceptor 中拿到 offset。 (此外 <https://github.com/apache/pulsar/pull/9257> 已支持在回调中直接获取 BrokerEntryMetadata)
- COMMIT_OFFSET 请求: 对于 __consumer_offsets, 原封不动写入 topic。对于 Pulsar 的 cumulative acknowledgement, 则对 ManagedLedger 进行二分查找。

```
// metadata added for entry from broker
message BrokerEntryMetadata {
    optional uint64 broker_timestamp = 1;
    optional uint64 index = 2;
}
```

因此 KoP 2.8.0 必须配置:

```
brokerEntryMetadataInterceptors=org.apache.pulsar.common.intercept.AppendIndexMetadataInterceptor
```

Message encode & decode

Before 2.8.0

生产：

1. 解析 MemoryRecords，重新构造 Pulsar 的 MessageMetadata。
2. 解压缩，解 batch，为所有单条消息重新构造 Pulsar 的 Message。
3. 同 Pulsar 客户端的操作。

消费：

1. 解析 Pulsar 消息的元数据，从而构造 MemoryRecordsBuilder。
2. 通过 MessageId 反算出 offset。
3. 解压缩，解 batch，将所有单条消息及其 offset 传入 appendWithOffset 方法。

Message encode & decode

如果不兼容 Pulsar 客户端呢？

☒ Kafka Producer & Kafka Consumer

☒ Pulsar Producer & Pulsar Consumer

☐ Kafka Producer & Pulsar Consumer

☐ Pulsar Producer & Kafka Consumer

生产的时候，直接将 MemoryRecords 内部的 ByteBuffer 写入 bookie 即可。

消费的时候呢？

其实还额外给 Metadata 加了个 property
`entry.format=kafka`

Message encode & decode

ManagedCursor

```
void asyncReadEntries(int numberOfEntriesToRead, ReadEntriesCallback callback, Object ctx,
                      PositionImpl maxPosition);
```

```
interface ReadEntriesCallback {
    void readEntriesComplete(List<Entry> entries, Object ctx);

    void readEntriesFailed(ManagedLedgerException exception, Object ctx);
}
```

简单粗暴的做法：继续用 MemoryRecordsBuilder 把所有 Entry 拼起来。

实际发现开销还是比较大，追根溯源看到 appendWithOffset 其实会对每条消息重新计算校验和。

```
public static long write(DataOutputStream out,
                        byte magic,
                        long timestamp,
                        ByteBuffer key,
                        ByteBuffer value,
                        CompressionType compressionType,
                        TimestampType timestampType) throws IOException {
    byte attributes = computeAttributes(magic, compressionType, timestampType);
    long crc = computeChecksum(magic, attributes, timestamp, key, value);
    write(out, magic, crc, attributes, timestamp, key, value);
    return crc;
}
```

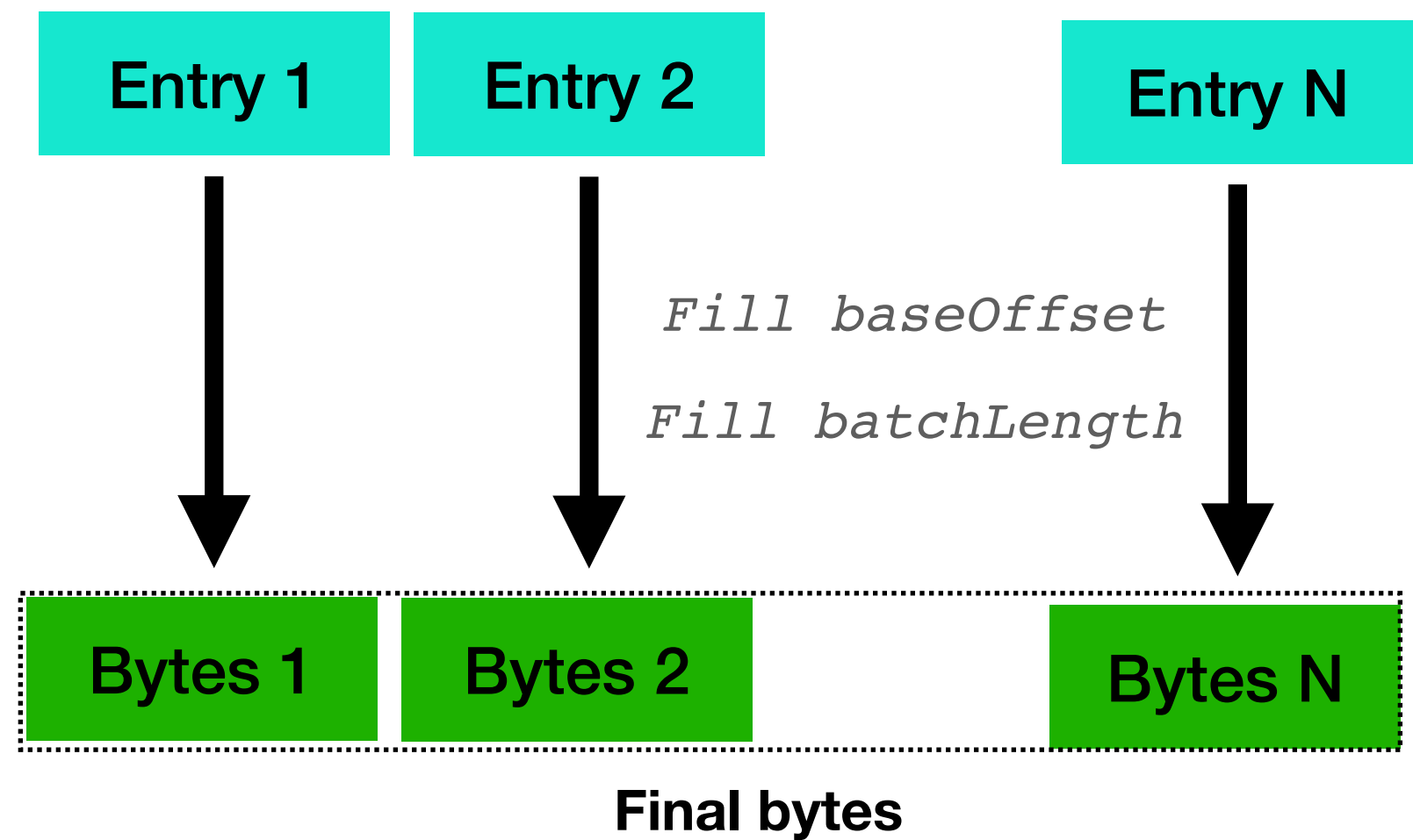

Message encode & decode

5.3.1 Record Batch

The following is the on-disk format of a RecordBatch.

```
1  baseOffset: int64
2  batchLength: int32
3  partitionLeaderEpoch: int32
4  magic: int8 (current magic value is 2)
5  crc: int32
6  attributes: int16
7    bit 0~2:
8      0: no compression
9      1: gzip
10     2: snappy
11     3: lz4
12     4: zstd
13    bit 3: timestampType
14    bit 4: isTransactional (0 means not transactional)
15    bit 5: isControlBatch (0 means not a control batch)
16    bit 6~15: unused
17  lastOffsetDelta: int32
18  firstTimestamp: int64
19  maxTimestamp: int64
20  producerId: int64
21  producerEpoch: int16
22  baseSequence: int32
23  records: [Record]
```

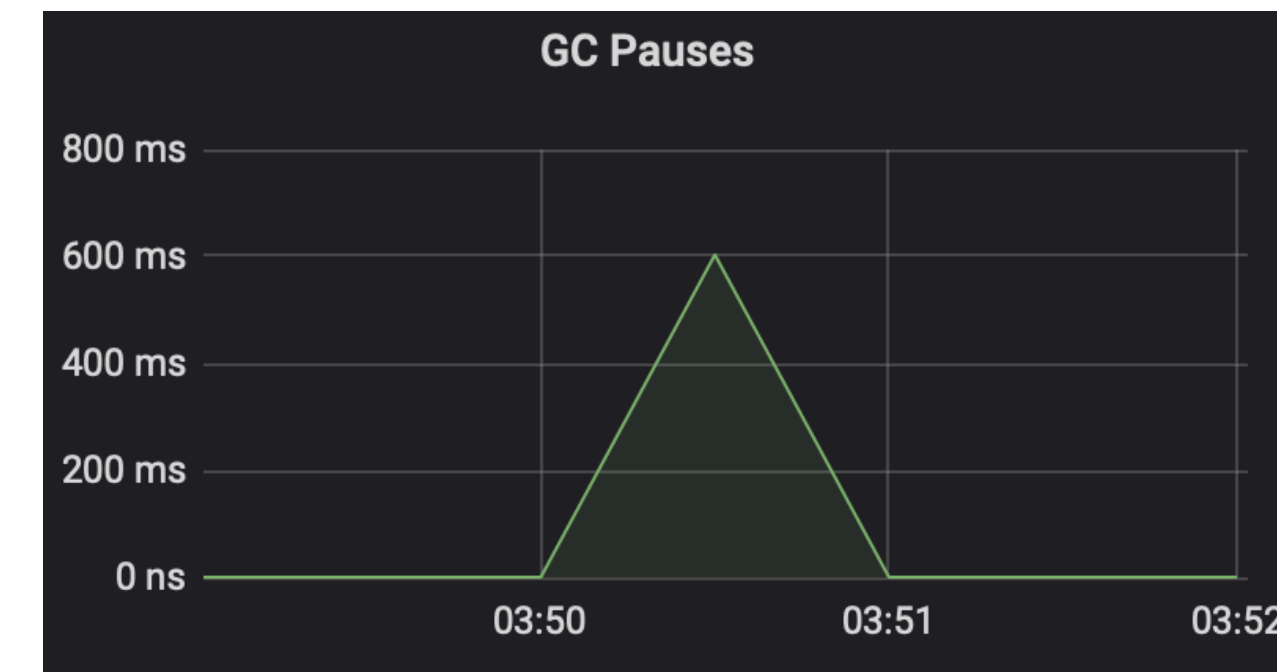
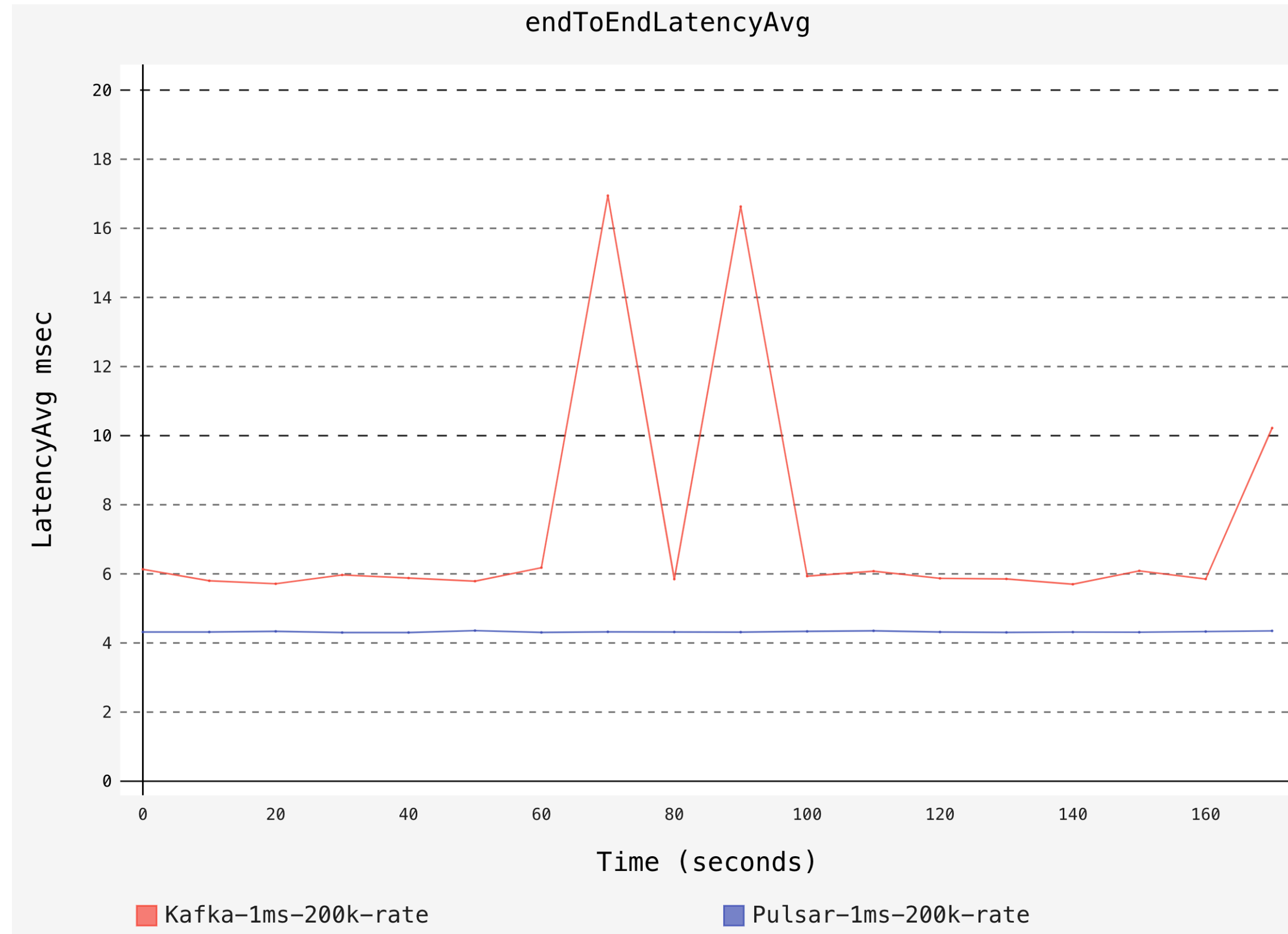
appendWithOffset 简化版



若不想兼容 Pulsar 和 Kafka 客户端互操作，可以添加如下配置：

entryFormat=kafka

性能测试 (WIP)



16 分区, batch.size=1KB, batch.ms=1, 200k * 1KB msg/s。

<https://github.com/BewareMyPower/openmessaging-benchmark/commits/bewaremypower/deploy-kop>

性能测试（WIP）



- HandleProduceRequest：PRODUCE 请求的处理开始，到这次请求所有消息全部成功写入 bookie。
- ProduceEncode：对 Kafka 消息编码的时间。
- MessageQueuedLatency：从每个分区的消息排队开始，到准备异步发送的时间。
- MessagePublish：单个分区的信息从异步发送开始，到成功写入 bookie 的时间。

KoP Authentication

Before 2.8.0

KoP 对 authentication 的支持仅限于 SASL/PLAIN 机制，它基于 Pulsar 的 JSON Web Token 认证，在 broker 的基本配置之外，只需要额外配置

```
saslAllowedMechanisms=PLAIN
```

用户端则需要输入 namespace 和 token 作为 JAAS 配置的用户名和密码。

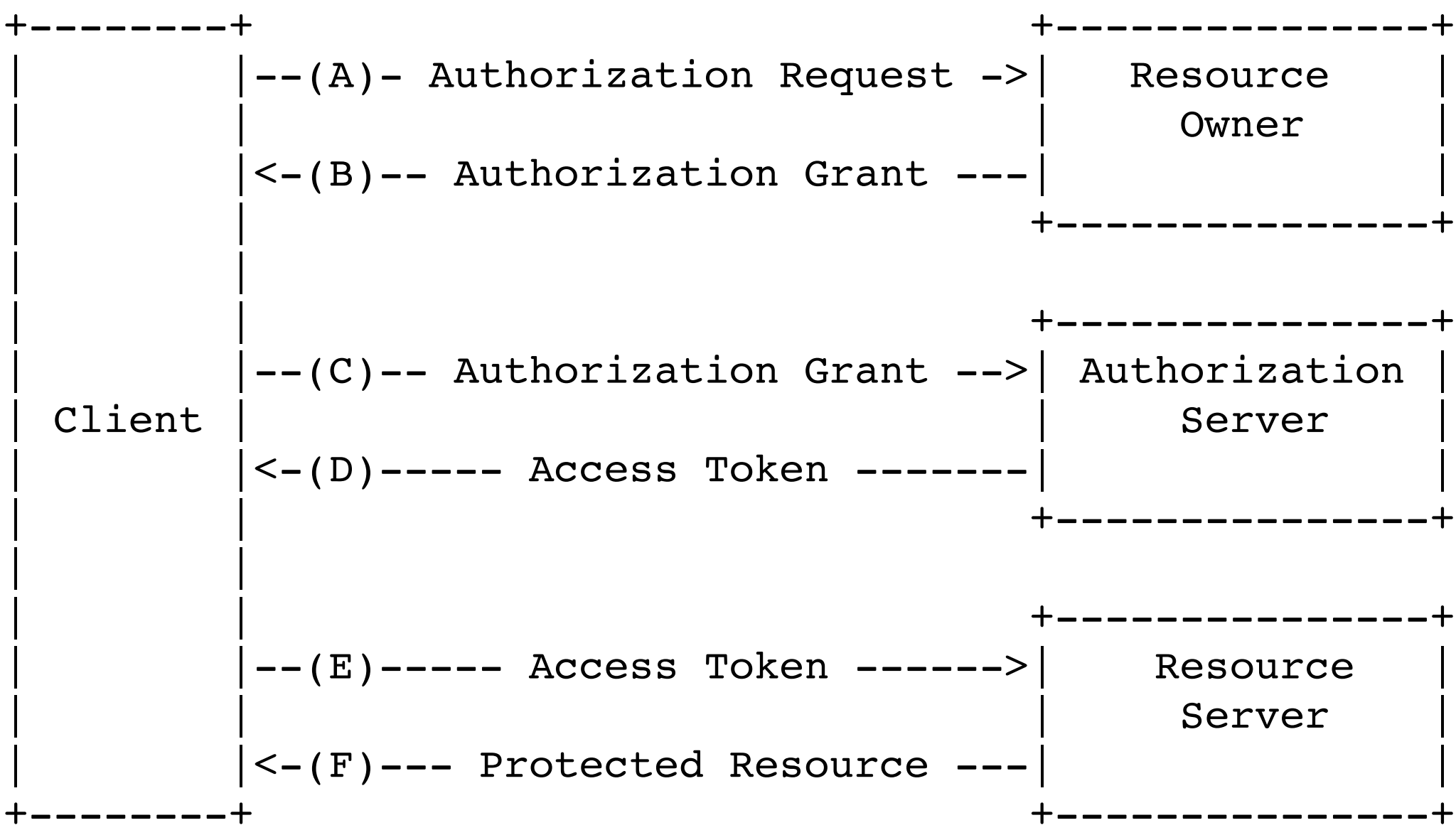
```
security.protocol=SASL_PLAINTEXT # or security.protocol=SASL_SSL if SSL connection is used  
sasl.mechanism=PLAIN  
sasl.jaas.config=org.apache.kafka.common.security.plain.PlainLoginModule \  
required username="public/default" password="token:xxx";
```

KoP Authentication

支持 OAuth 2.0

KoP 2.8.0 支持 OAuth 2.0 进行认证，也就是 SASL/OAUTHBEARER 机制。

```
saslAllowedMechanisms=OAUTHBEARER
```



KoP Authentication

支持 OAuth 2.0

类似 Kafka，KoP 也需要在 broker 端配置 Server Callback Handler 用于 token 验证：

- `kopOAuth2AuthenticateCallbackHandler`：handler 类
- `kopOAuth2ConfigFile`：配置文件路径

KoP 提供了一种实现类，它基于 Pulsar broker 配置的 `AuthenticationProvider` 进行验证，因此配置文件中仅需配置 `auth.validate.method=<method>`。

KoP Authentication

支持 OAuth 2.0

对于 Kafka 客户端，KoP 提供了一种 Login Callback Handler 实现。

```
sasl.login.callback.handler.class=io.streamnative.pulsar.handlers.kop.security.oauth.OauthLoginCallbackHandler
security.protocol=SASL_PLAINTEXT # or security.protocol=SASL_SSL if SSL connection is used
sasl.mechanism=OAUTHBEARER
sasl.jaas.config=org.apache.kafka.common.security.oauthbearer.OAuthBearerLoginModule \
  required oauth.issuer.url="https://accounts.google.com" \
  oauth.credentials.url="file:///path/to/credentials_file.json" \
  oauth.audience="https://broker.example.com";
```

Kafka Java 客户端 OAuth 2.0 认证

Java

You can use the factory method to configure authentication for Pulsar Java client.

```
String issuerUrl = "https://dev-kt-aa9ne.us.auth0.com";
String credentialsUrl = "file:///path/to/KeyFile.json";
String audience = "https://dev-kt-aa9ne.us.auth0.com/api/v2/";

PulsarClient client = PulsarClient.builder()
    .serviceUrl("pulsar://broker.example.com:6650/")
    .authentication(
        AuthenticationFactoryOAuth2.clientCredentials(issuerUrl, credentialsUrl, audience))
    .build();
```

 Copy

Pulsar Java 客户端 OAuth 2.0 认证

KoP 2.8.0 其他进展

- 移植了 Kafka 的 Transaction Coordinator

若想启用 transaction，需要添加如下配置：

```
enableTransactionCoordinator=true
```

```
brokerId=<id>
```

- 基于 PrometheusRawMetricsProvider 添加了 KoP 自定义的 metrics。
- 暴露 advertised listeners，从而支持 Envoy Kafka Filter 进行代理。
- 完善对 Kafka AdminClient 的支持。

近期计划

在 KoP 2.8.0 正式发布之前：

1. 添加更详细的 metrics。
2. 排查压测过程中内存持续增长以及 full GC 的问题。
3. 进行更为系统的性能测试。
4. 处理社区近期反馈的问题。