Apache Kafka 심화 개념 및 이해

9 Exactly Once Semantics(EOS) 2

Transaction을 구현하기 위해, 몇 가지 새로운 개념들이 도입

- Transaction Coordinator : Consumer Group Coordinator와 비슷하게, 각 Producer에게는 Transaction Coordinator가 할당되며, PID 할당 및 Transaction 관리의 모든 로직을 수행

- Transaction Log : 새로운 Internal Kafka Topic으로써, Consumer Offset Topic과 유사하게, 모든 Transaction의 영구적이고 복제된 Record를 저장하는 Transaction Coordinator의 상태 저장소

- TransactionalId : Producer를 고유하게 식별하기 위해 사용되며, 동일한 TransactionalId를 가진 Producer의 다른 인스턴스들은 이전 인스턴스에 의해 만들어진 모든 Transaction을 재개(또는 중단)할 수 있음

Transaction 관련 파라미터
Broker Configs

Broker Configs

Parameter	설명	Default 값
transactional.id.expiration.ms	• Transaction Coordinator가 Producer TransactionalId로부터 Transaction 상태 업데이트를 수신하지 않고 사전에 만료되기 전에 대기하는 최대 시간(ms)	604800000 (7 days)
transaction.max.timeout.ms	• Transaction에 허용되는 최대 timeout 시간 • Client가 요청한 Transaction 시간이 이 시간을 초과하면 Broker는 InitPidRequest에서 InvalidTransactionTimeout 오류를 반환 • Producer가 Transaction에 포함된 Topic에서 읽는 Consumer를 지연시킬 수 있는 너무 큰 시간 초과를 방지	900000 (15 min)
transaction.state.log.replication.factor	• Transaction State Topic의 Replication Factor	3
transaction.state.log.num.partitions	• Transaction State Topic의 Partition 개수	50
transaction.state.log.min.isr	• Transaction State Topic의 min ISR 개수	2
transaction.state.log.segment.bytes	• Transaction State Topic의 Segment 크기	104857600 bytes

Transaction 관련 파라미터
Producer Configs

Producer Configs

Parameter	설명	Default 값
enable.idempotence	• 비활성화된 경우 Transaction 기능을 사용할 수 없음 • 활성화(true)하고 acks=all, retries > 1, max.inflight.requests.per.connection=1 을 같이 사용해야 함	false
transaction.timeout.ms	• Transaction Coordinator가 진행 중인 Transaction을 사전에 중단하기 전에 Producer의 Transaction 상태 업데이트를 기다리는 최대 시간(ms) • 이 구성 값은 InitPidRequest와 함께 Transaction Coordinator에게 전송 • 이 값이 Broker의 max.transaction.timeout.ms 설정보다 크면 'InvalidTransactionTimeout' 오류와 함께 요청이 실패	60000 (60 sec)
transactional.id	• Transaction 전달에 사용할 TransactionalId • 이를 통해 클라이언트는 새로운 Transaction을 시작하기 전에 동일한 TransactionalId를 사용하는 Transaction이 완료되었음을 보장할 수 있으므로 여러 Producer session에 걸쳐 있는 안정성 의미 체계를 사용할 수 있음 • TransactionalId가 비어있으면(default), Producer는 Idempotent Delivery 만으로 제한 • TransactionalId가 구성된 경우, 반드시 enable.idempotence를 활성화해야 함	없음

Transaction 관련 파라미터
Consumer Configs

Consumer Configs

Parameter	설명	Default 값
isolation.level	• read_uncommitted: Offset 순서로 Commit된 메시지와 Commit되지 않은 메시지를 모두 사용 • read_committed: Non-Transaction 메시지 또는 Commit된 Transaction 메시지만 Offset 순서로 사용	read_uncommitted
enable.auto.commit	• false : Consumer Offset에 대한 Auto Commit 을 Off	true

✓ Consumer가 중복해서 데이터 처리하는 것에 대해 보장하지 않으므로, Consumer의 중복처리는 따로 로직을 작성해야 함(Idempotent Consumer)
✓ 예를 들어, 메시지를 성공적으로 사용한 후 Kafka Consumer를 이전 Offset으로 되감으면 해당 Offset에서 최신 Offset까지 모든 메시지를 다시 수신하게 됨

Transaction Data Flow 관련 예제 소스 코드
Consume 하고 Produce하는 과정을 Transaction으로 처리

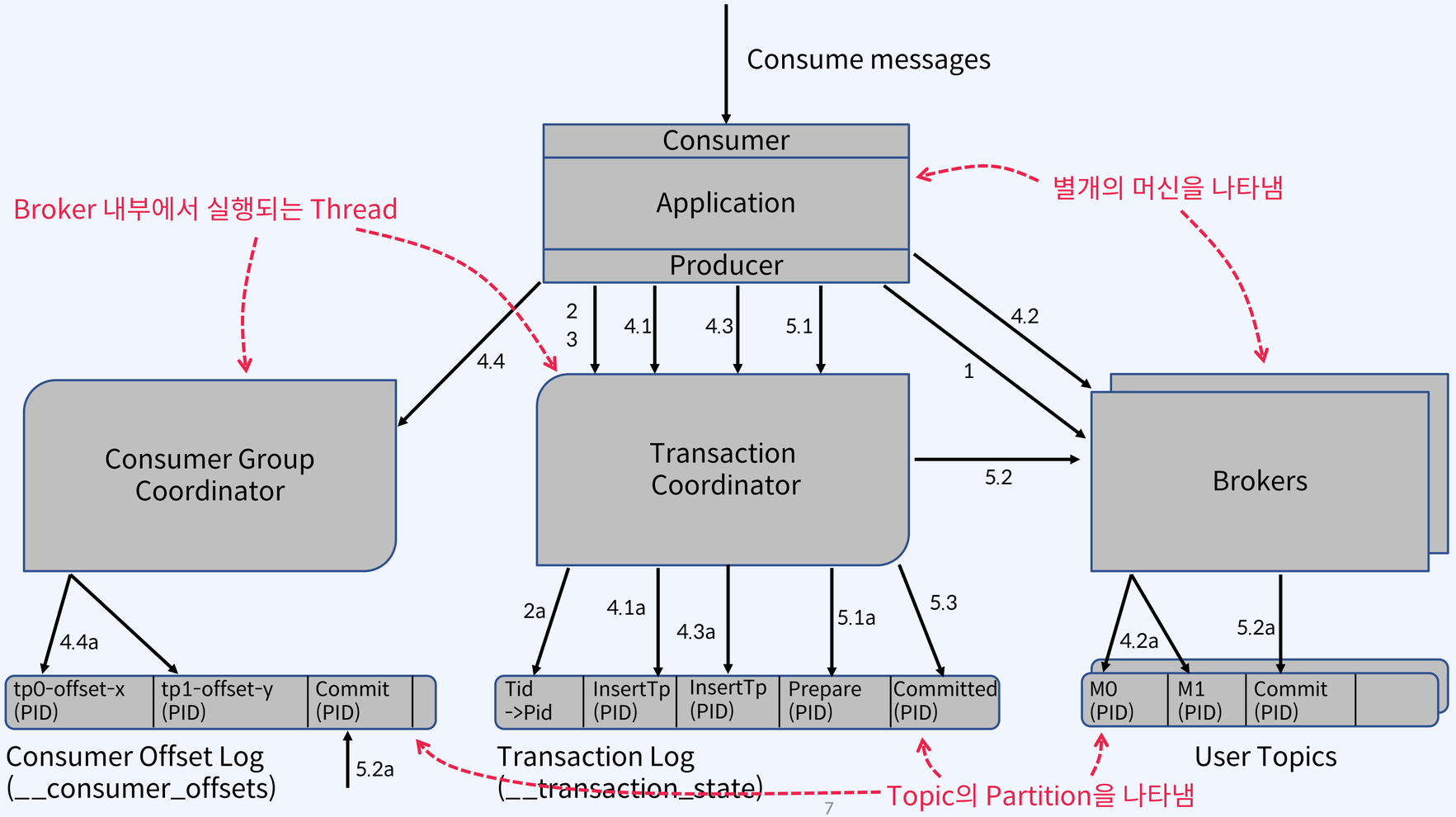
KIP-98 : Exactly Once Delivery and Transactional Messaging
https://cwiki.apache.org/confluence/display/KAFKA/KIP-98+-+Exactly+Once+Delivery+and+Transactional+Messaging

- initTransactions 으로 시작
- poll 로 Source Topic에서 record를 가져옴
- Transaction을 시작
- record로 비즈니스로직 수행 후, 결과 record를 Target Topic으로 send
- sendOffsetsToTransaction을 호출하여 consume(poll)한 Source Topic에 consumer offset을 commit
- commitTransaction 또는 abortTransaction 으로 Transaction Commit 또는 Rollback수행

```
producer.initTransactions();

while(true) {
  ConsumerRecords<String, String> records = consumer.poll(CONSUMER_POLL_TIMEOUT);
  if (!records.isEmpty()) {
    // Start a new transaction. This will begin the process of batching the consumed
    // records as well
    // as an records produced as a result of processing the input records.
    //
    // We need to check the response to make sure that this producer is able to initiate
    // a new transaction.
    producer.beginTransaction();

    // Process the input records and send them to the output topic(s).
    List<ProducerRecord<String, String>> outputRecords = processRecords(records);
    for (ProducerRecord<String, String> outputRecord : outputRecords) {
      producer.send(outputRecord);
    }

    // To ensure that the consumed and produced messages are batched, we need to commit
    // the offsets through
    // the producer and not the consumer.
    //
    // If this returns an error, we should abort the transaction.

    sendOffsetsResult = producer.sendOffsetsToTransaction(getUncommittedOffsets());


    // Now that we have consumed, processed, and produced a batch of messages, let's
    // commit the results.
    // If this does not report success, then the transaction will be rolled back.
    producer.endTransaction();
  }
}
```

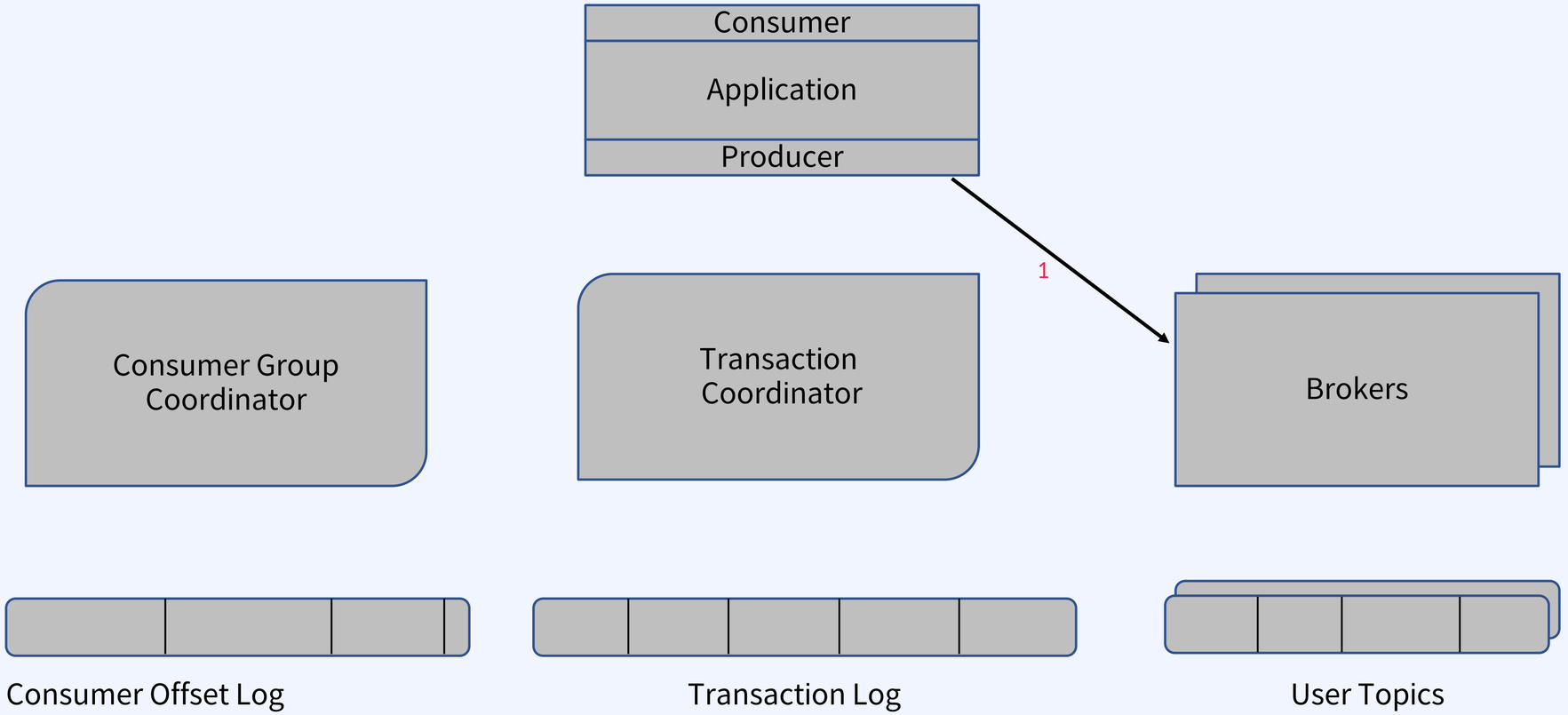
Transaction Data Flow
Transaction 처리 프로세스

KIP-98 : Exactly Once Delivery and Transactional Messaging
https://cwiki.apache.org/confluence/display/KAFKA/KIP-98+-+Exactly+Once+Delivery+and+Transactional+Messaging



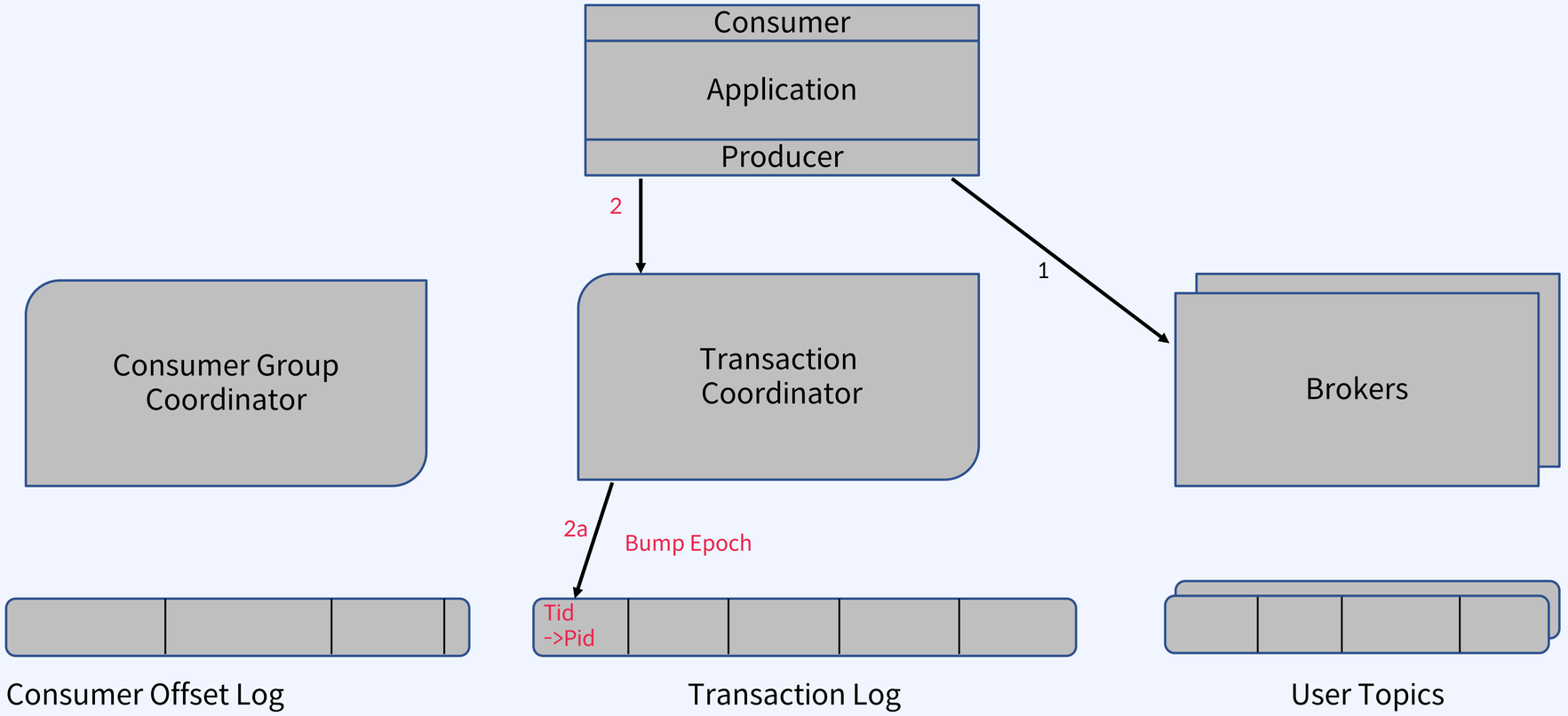
Transaction Data Flow
Transaction 처리 프로세스

1. Transactions Coordinator 찾기
   Producer가 initTransactions()를 호출하여 Broker에게 FindCoordinatorRequest를 보내서 Transaction Coordinator의 위치를 찾음
   Transaction Coordinator는 PID를 할당



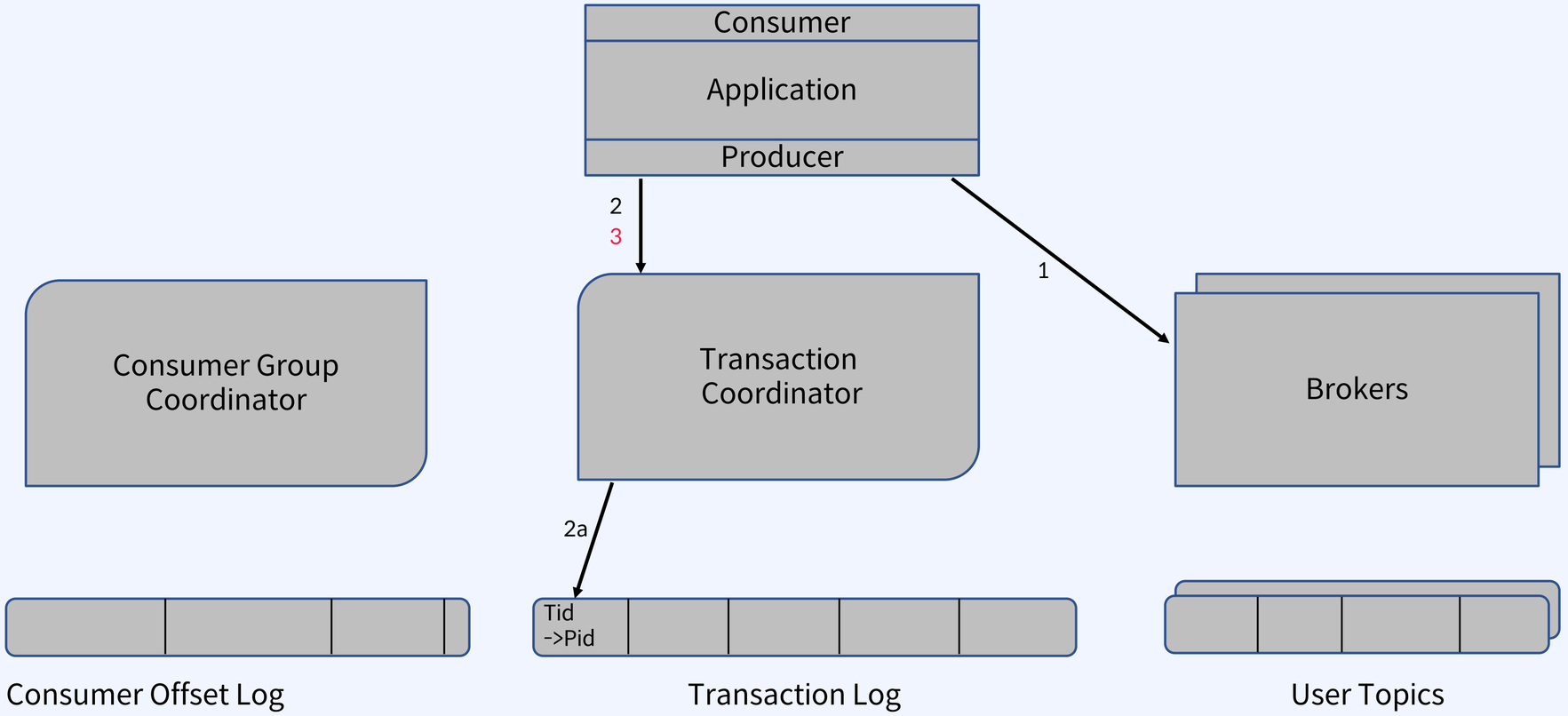
Transaction Data Flow
Transaction 처리 프로세스

2. Producer ID 얻기
Producer가 Transaction Coordinator에게 InitPidRequest를 보내서(TransactionalId를 전달) Producer의 PID를 가져옴
PID의 Epoch를 높여 Producer의 이전 Zombie 인스턴스가 차단되고 Transaction을 진행할 수 없도록 함
해당 PID에 대한 매핑이 2a단계에서 Transaction Log에 기록



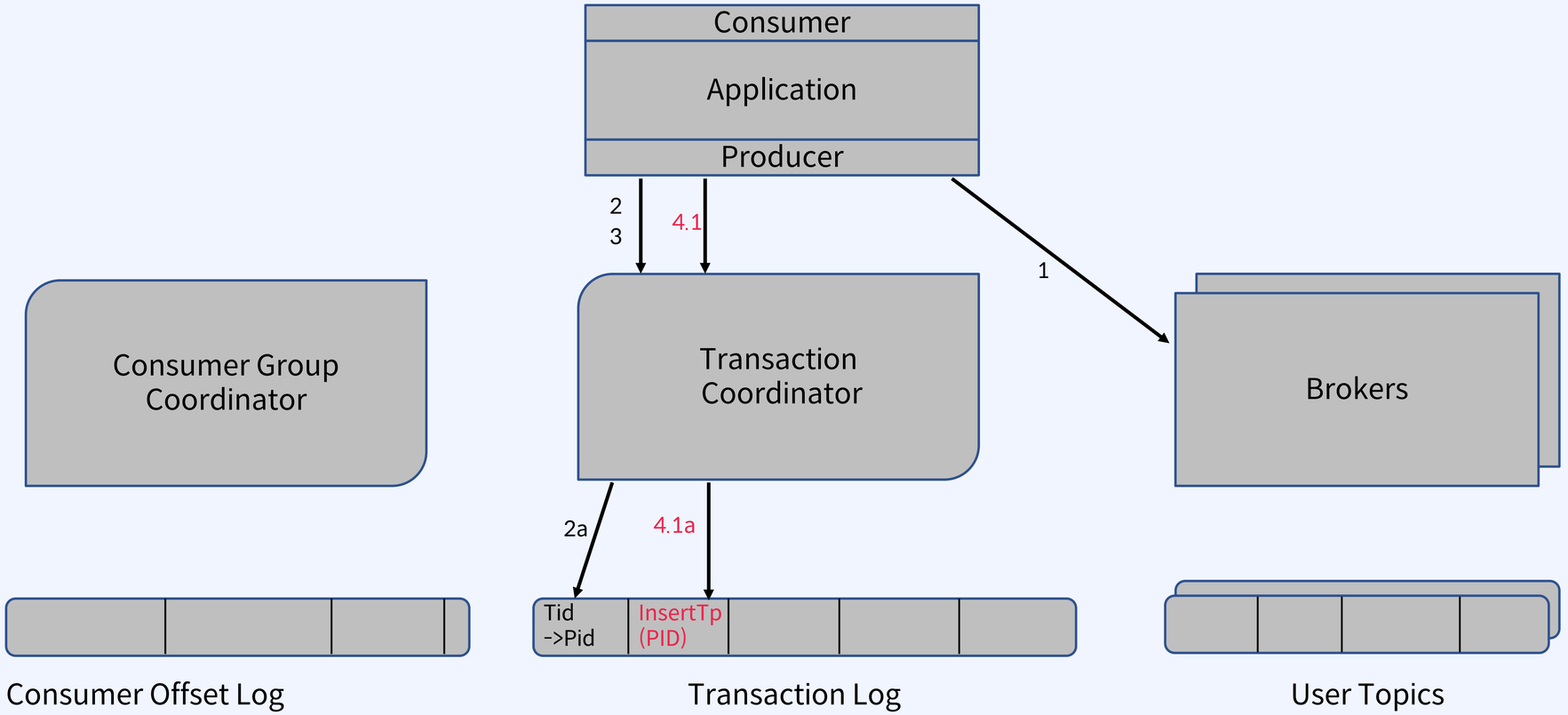
Transaction Data Flow
Transaction 처리 프로세스

3. Transaction 시작
   Producer가 beginTransactions()를 호출하여 새 Transaction의 시작을 알림
   Producer는 Transaction이 시작되었음을 나타내는 로컬 상태를 기록
   첫 번째 Record가 전송될 때까지 Transaction Coordinator의 관점에서는 Transaction이 시작되지 않음



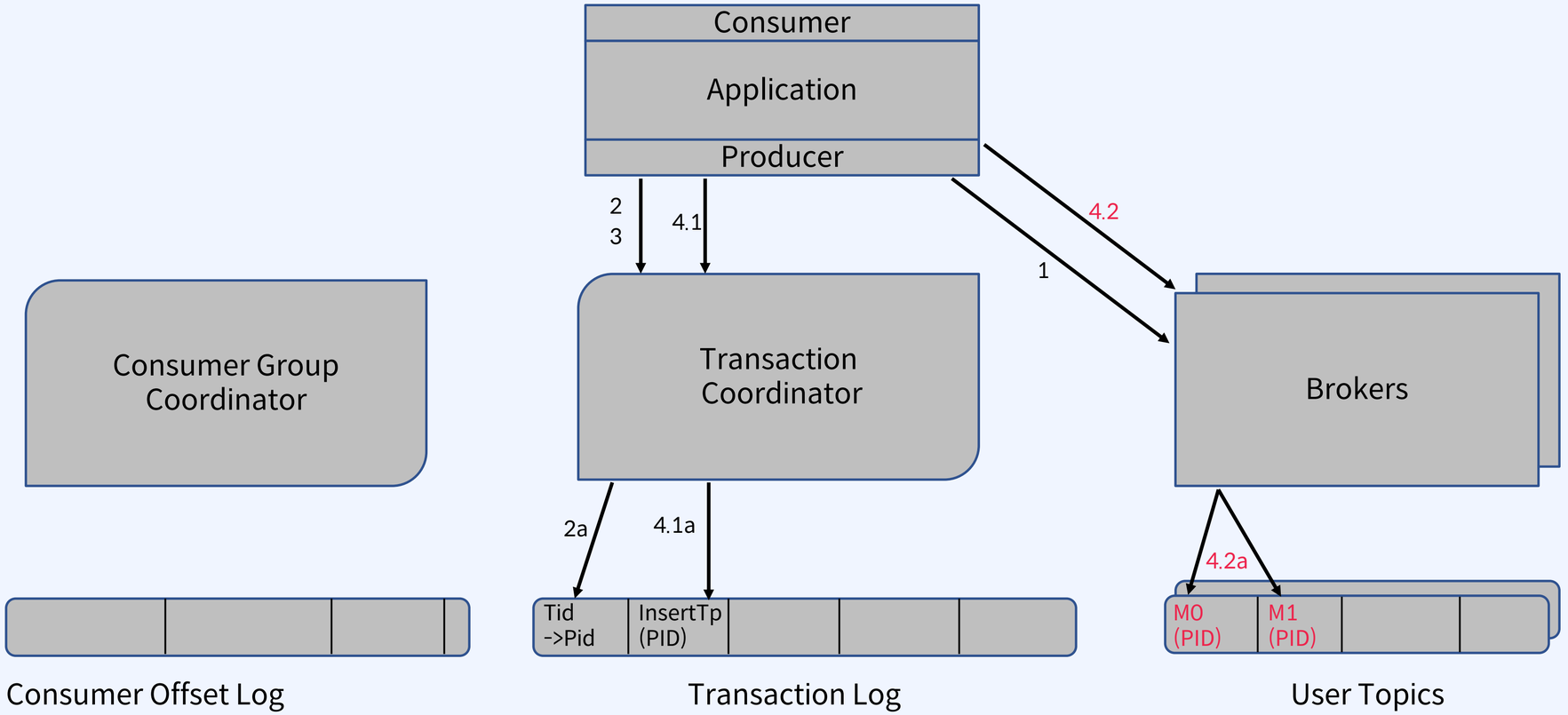
Transaction Data Flow
Transaction 처리 프로세스

4.1. AddPartitionsToTxnRequest
Producer는 Transaction의 일부로 새 TopicPartition이 처음 기록될 때 이 요청을 Transaction Coordinator에게 보냄
이 TopicPartition을 Transaction에 추가하면 Transaction Coordinator가 4.1a 단계에서 기록
Transaction에 추가된 첫 번째 Partition인 경우 Transaction Coordinator는 Transaction Timer도 시작



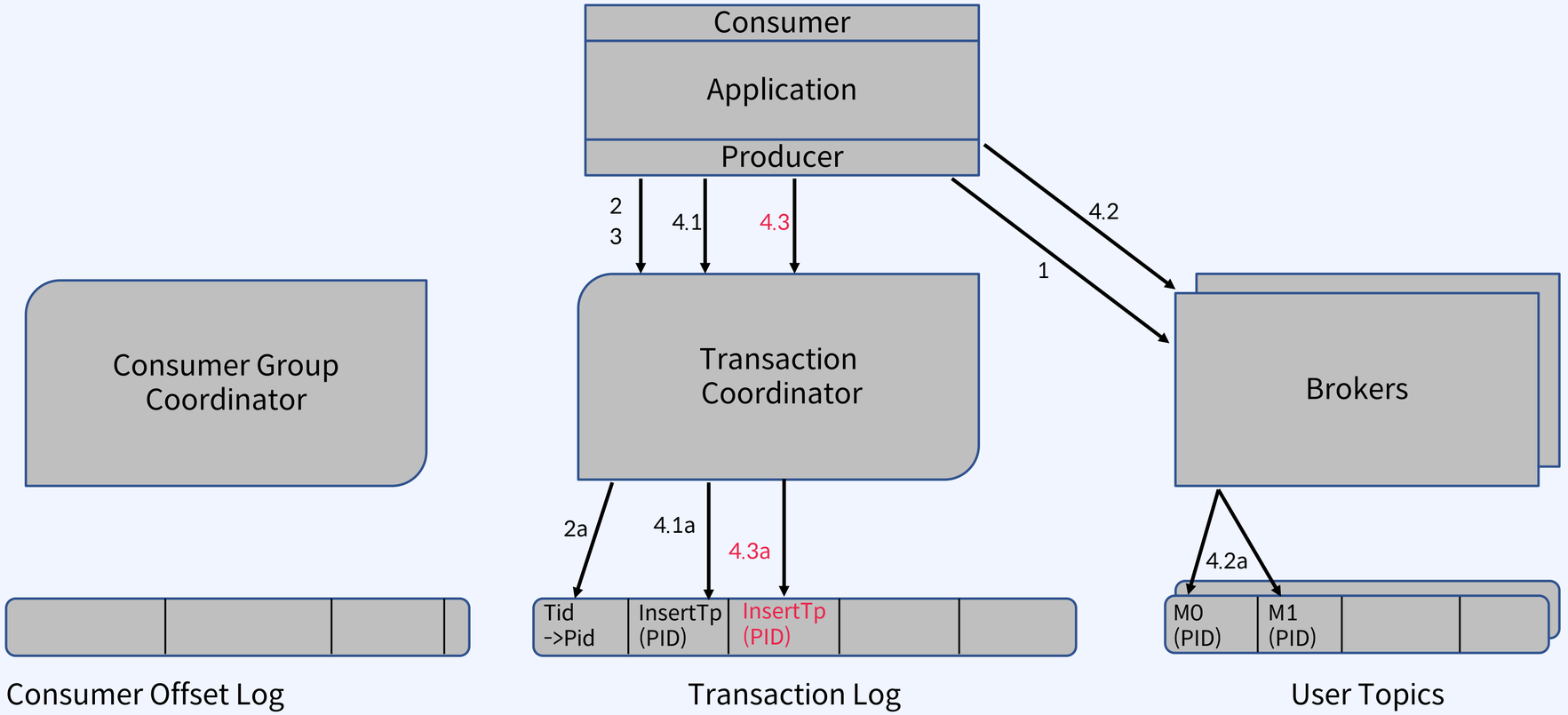
Transaction Data Flow
Transaction 처리 프로세스

4.2. ProduceRequest
Producer는 하나 이상의 ProduceRequests(Producer의 send()에서 시작됨)를 통해 User Topic Partitions에 메시지를 Write
이러한 요청에는 4.2a 에 표시된 대로 PID, Epoch 및 Sequence Number가 포함



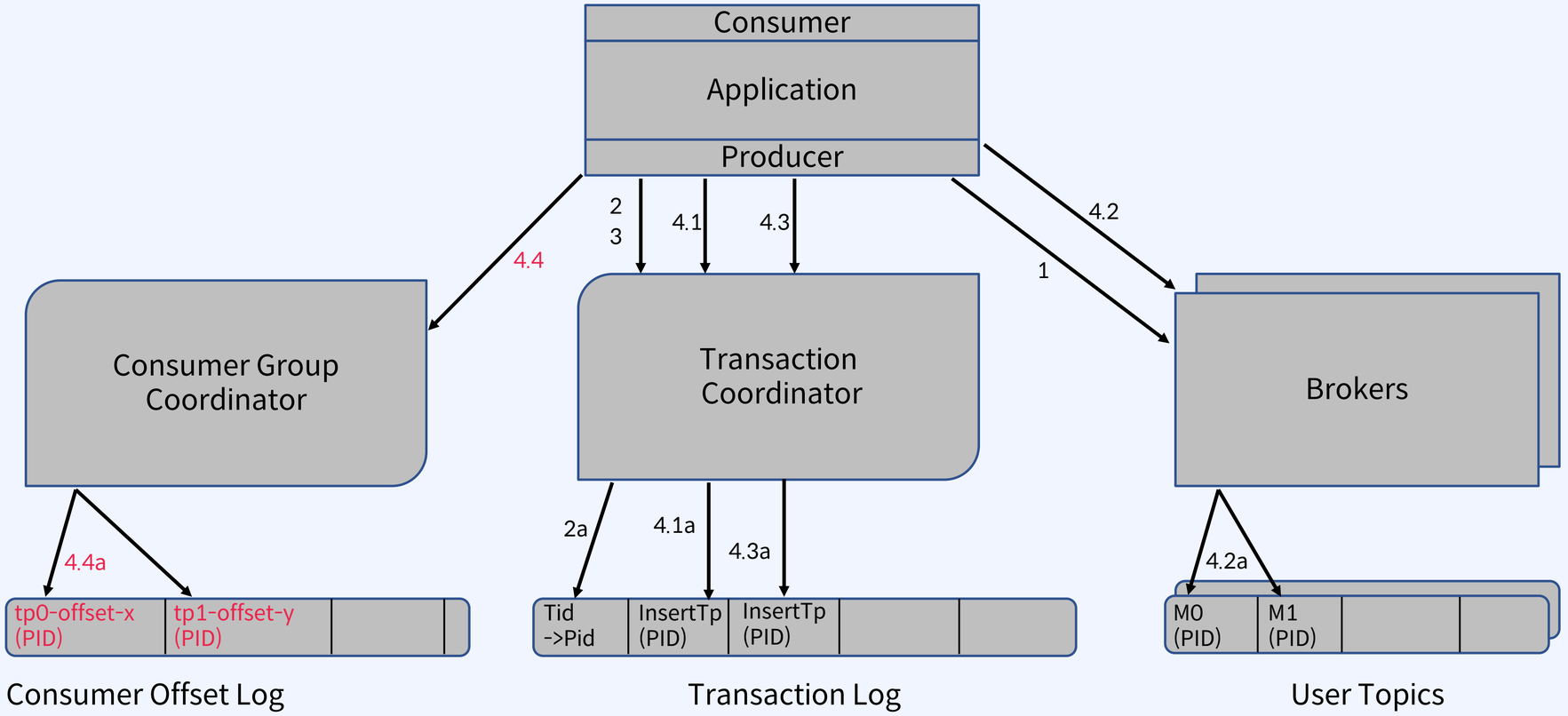
Transaction Data Flow
Transaction 처리 프로세스

4.3. AddOffsetCommitsToTxnRequest
Producer에는 Consume되거나 Produce되는 메시지를 Batch 처리할 수 있는 sendOffsetsToTransaction() 가 있음
sendOffsetsToTransaction 메서드는 groupId가 있는 AddOffsetCommitsToTxnRequests를 Transaction Coordinator에게 보냄
여기서 Transaction Coordinator 는 내부 __consumer_offsets Topic에서 이 Consumer Group에 대한 TopicPartition을 추론함
Transaction Coordinator는 4.3a 단계에서 Transaction Log에 이 Topic Partition의 추가를 기록



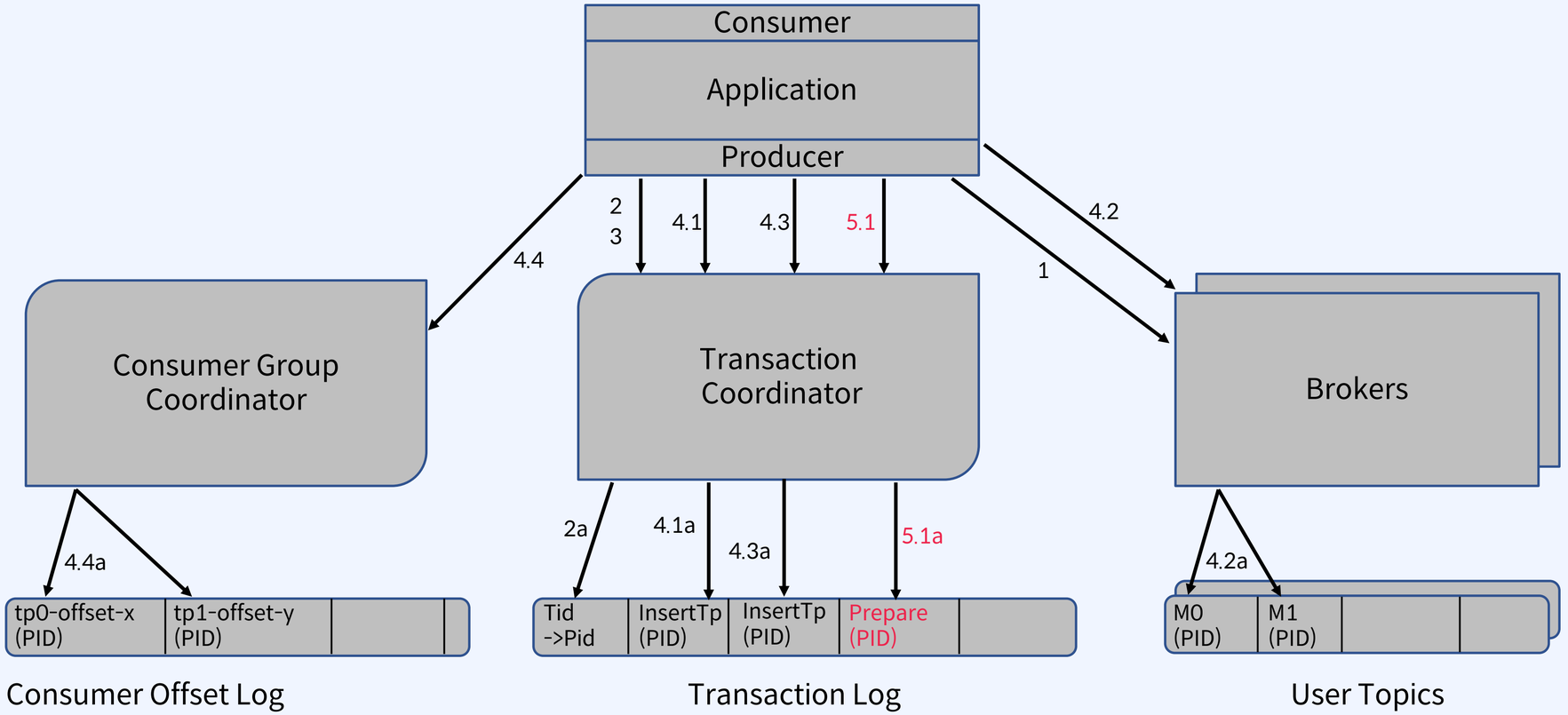
Transaction Data Flow
Transaction 처리 프로세스

4.4. TxnOffsetCommitRequest
Producer는 __consumer_offsets Topic에서 Offset을 유지하기 위해 TxnOffsetCommitRequest를 Consumer Coordinator에게 보냄
Consumer Coordinator는 전송되는 PID 및 Producer Epoch를 사용하여 Producer가 이 요청을 할 수 있는지(Zombie가 아님) 확인
Transaction이 Commit 될 때까지 해당 Offset은 외부에서 볼 수 없음



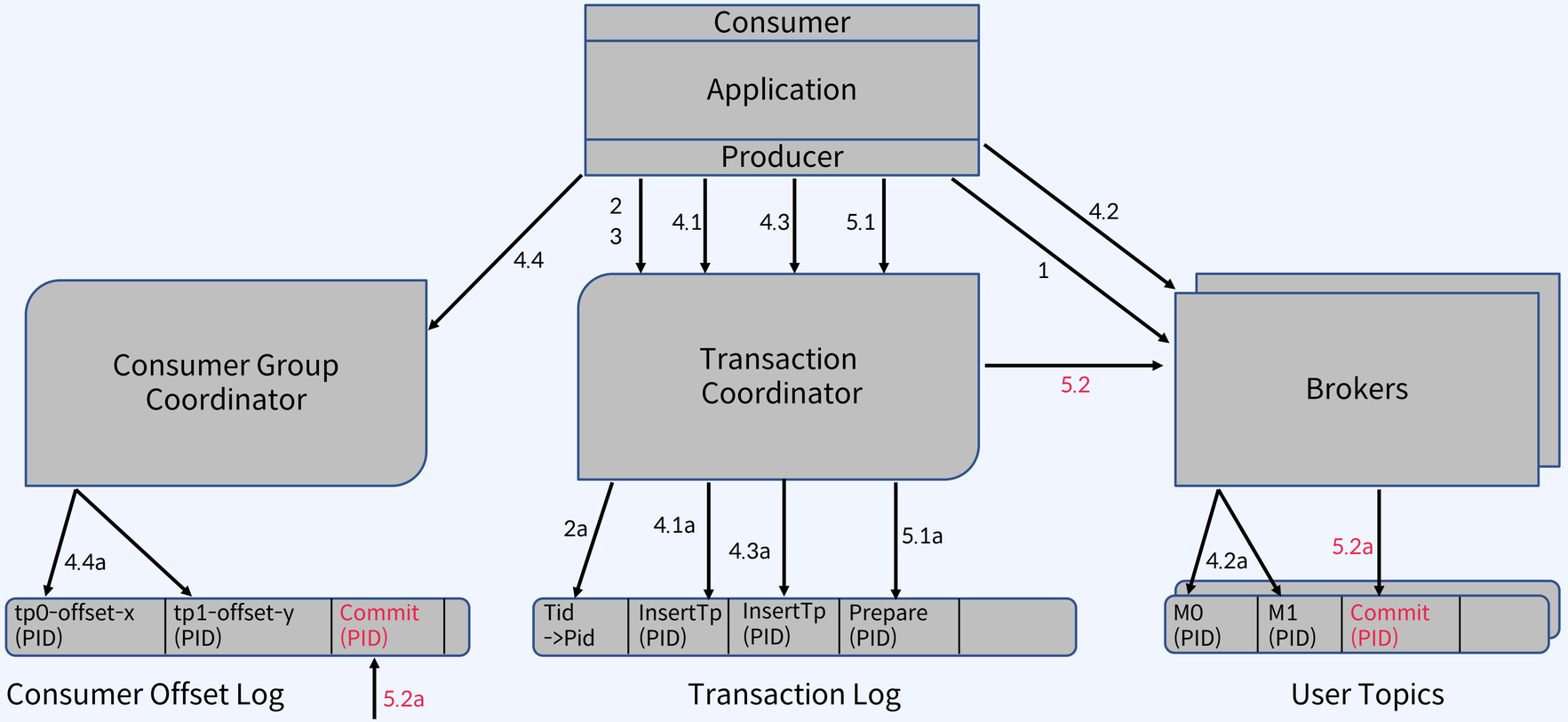
Transaction Data Flow
Transaction 처리 프로세스

5.1. EndTxnRequest
Producer는 Transaction을 완료하기 위해 commitTransaction() 또는 abortTransaction()을 호출
Producer는 Commit되거나 Abort되는지를 나타내는 데이터와 함께 Transaction Coordinator에게 EndTxnRequest를 보냄
Transaction Log에 PREPARE_COMMIT 또는 PREPARE_ABORT 메시지를 write



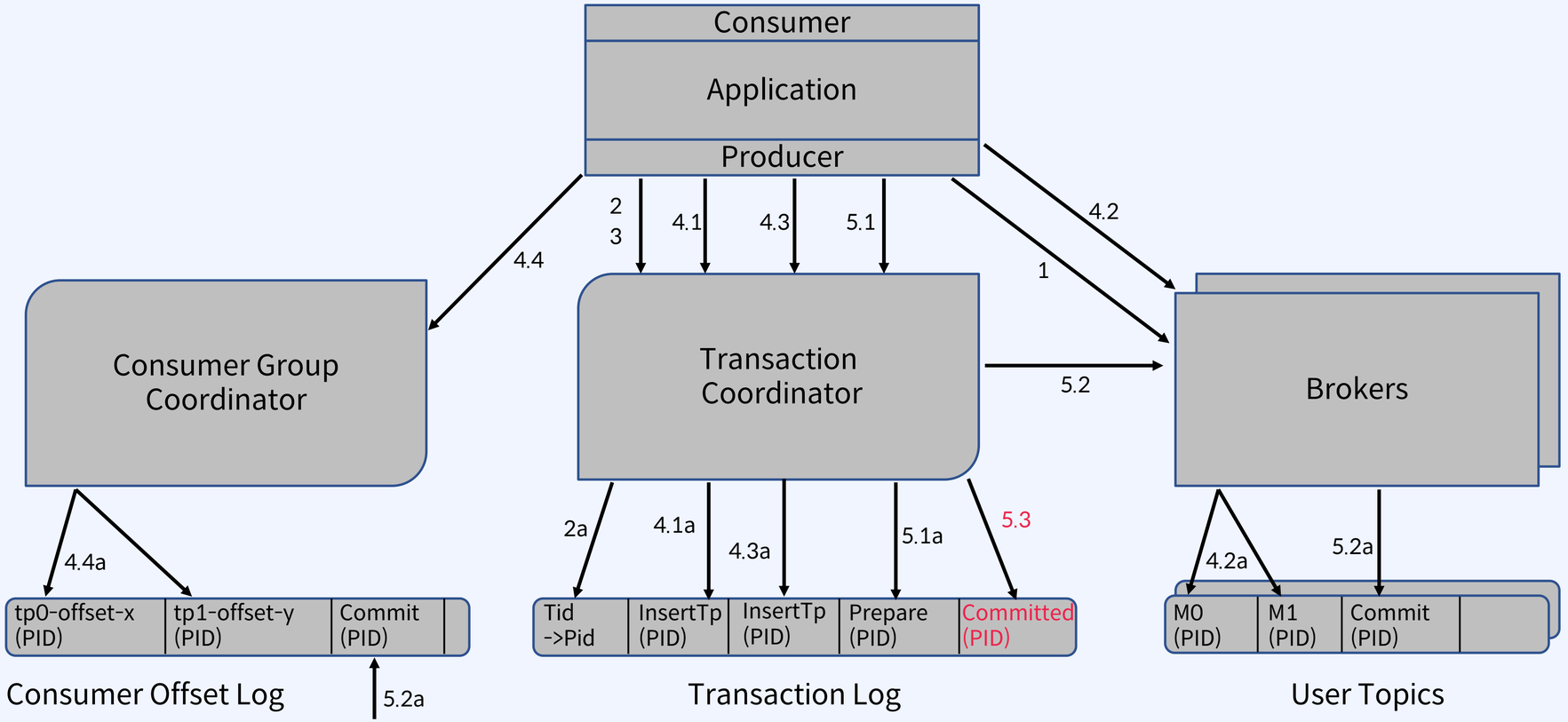
Transaction Data Flow
Transaction 처리 프로세스

5.2. WriteTxnMarkerRequest
Transaction Coordinator가 Transaction에 포함된 각 TopicPartition의 Leader에게 이 요청을 보냄
이 요청을 받은 각 Broker는 COMMIT(PID) 또는 ABORT(PID) 제어 메시지를 로그에 기록
__consumer_offsets Topic에도 Commit (또는 Abort) 가 로그에 기록
Consumer Coordinator는 Commit의 경우 이러한 오프셋을 구체화하거나 Abort의 경우 무시해야 한다는 알림을 받음



Transaction Data Flow
Transaction 처리 프로세스

5.3. Writing the final Commit or Abort Message
Transaction Coordinator는 Transaction이 완료되었음을 나타내는 최종 COMMITTED 또는 ABORTED를 Transaction Log에 기록
이 시점에서 Transaction Log에 있는 Transaction과 관련된 대부분의 메시지를 제거할 수 있음
Timestamp와 함께 완료된 Transaction의 PID만 유지하면 되므로 결국 Producer에 대한 TransactionalId->PID 매핑을 제거할 수 있음



Summary
Transaction

- Transaction 관련 파라미터
- Transaction Data Flow