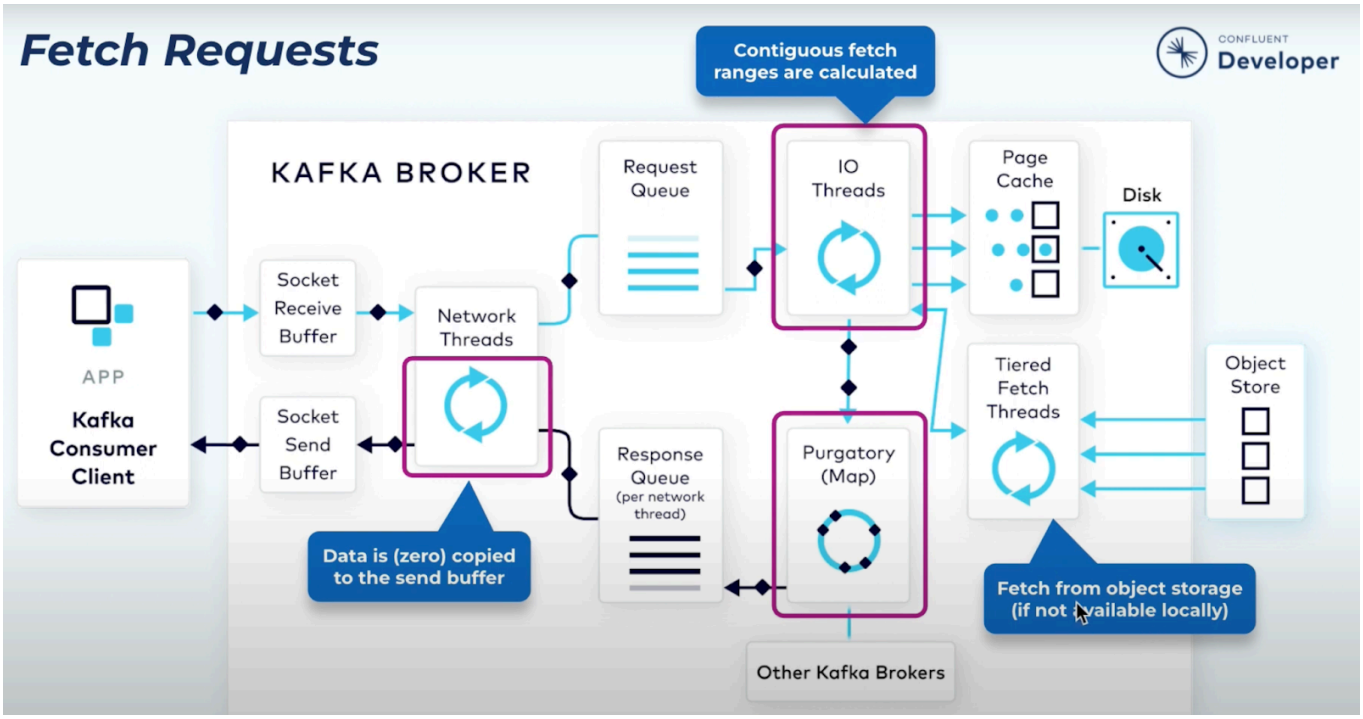
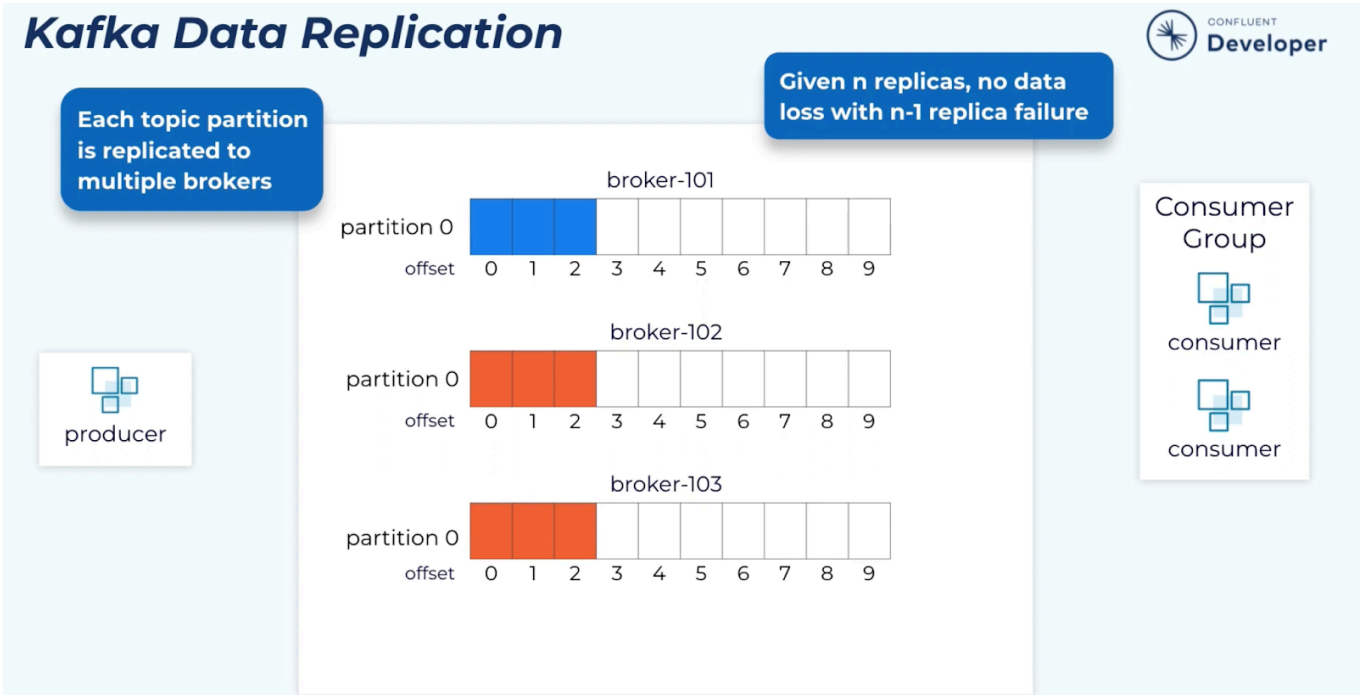
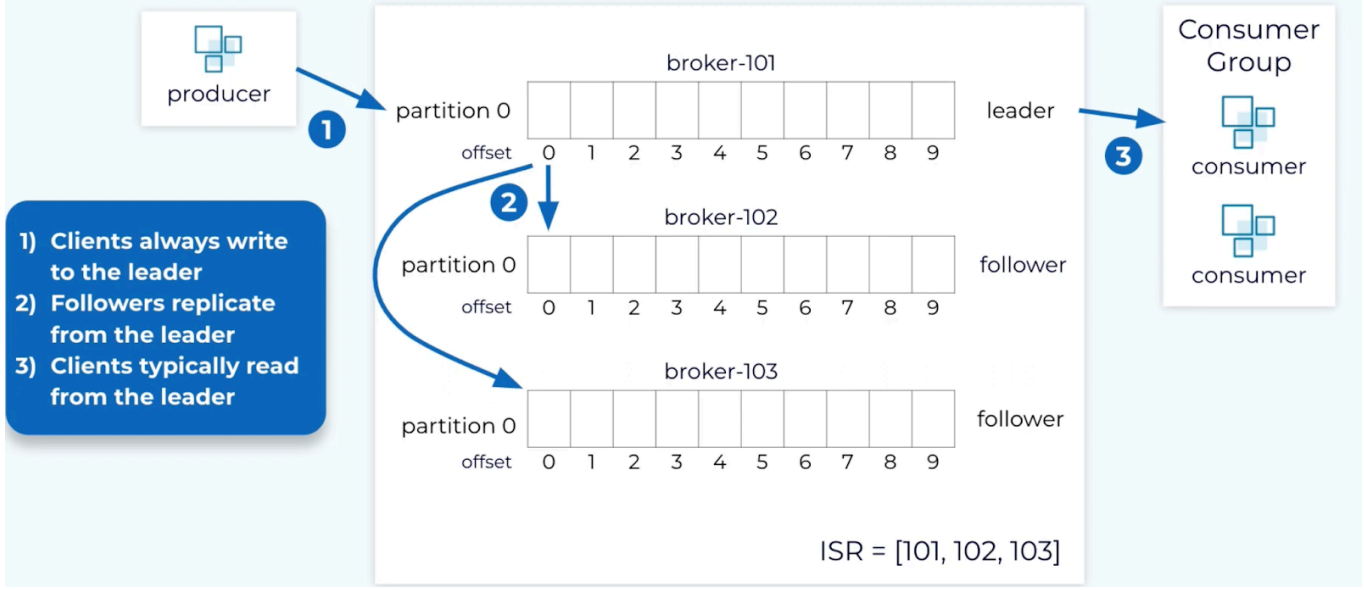
Kafka Architecture



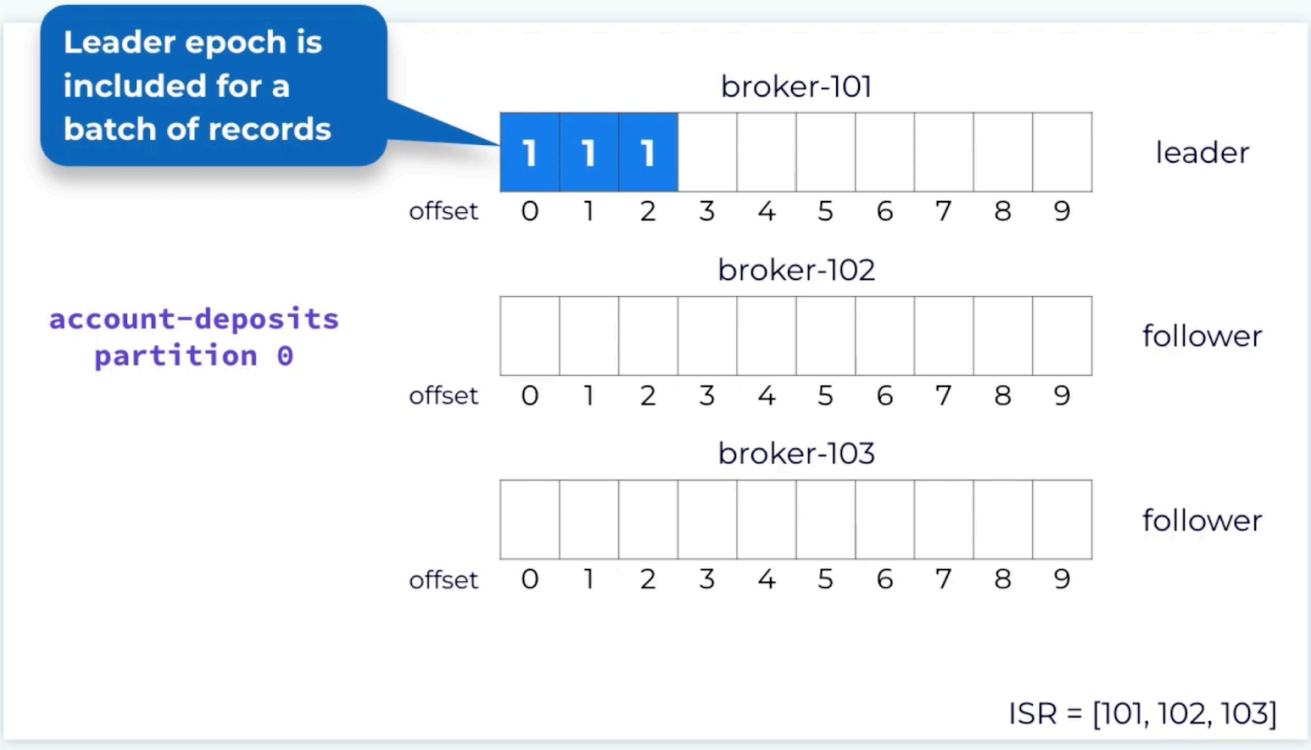
Kafka Data Replication:



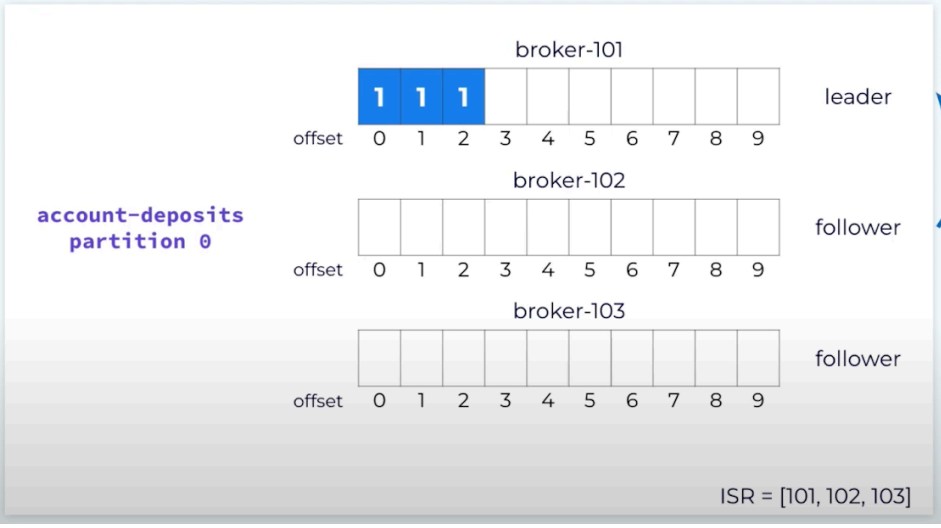
Leader, Follower, and In-Sync Replica (ISR) List



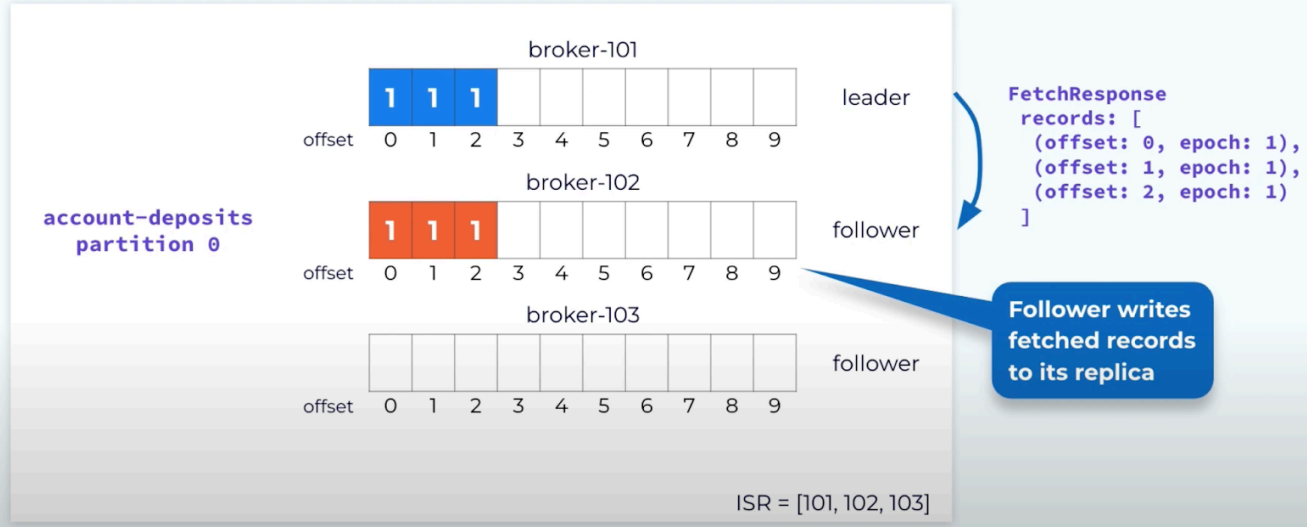
Leader Epoch



Follower Fetch Request

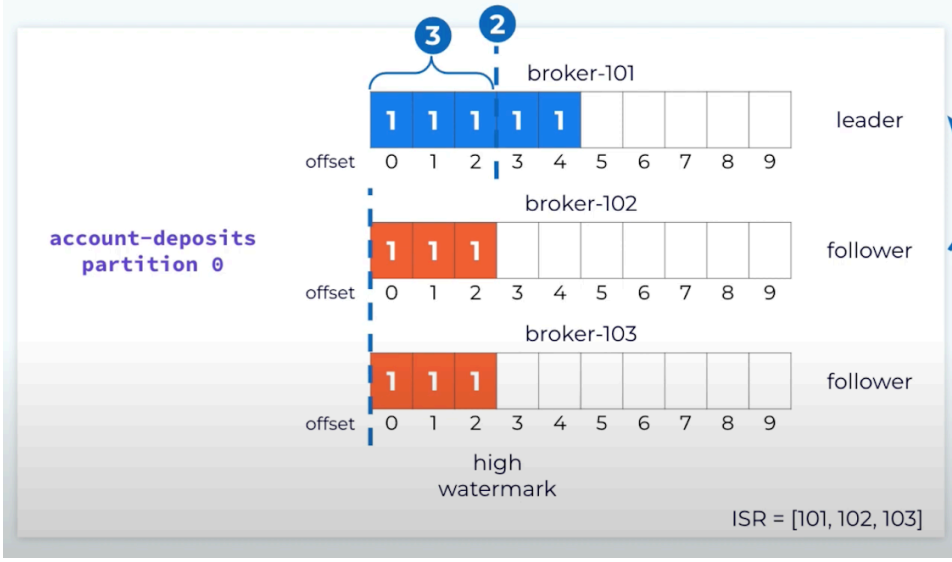


Follower Fetch Response



Committing Partition Offsets

CONFLUENT Developer



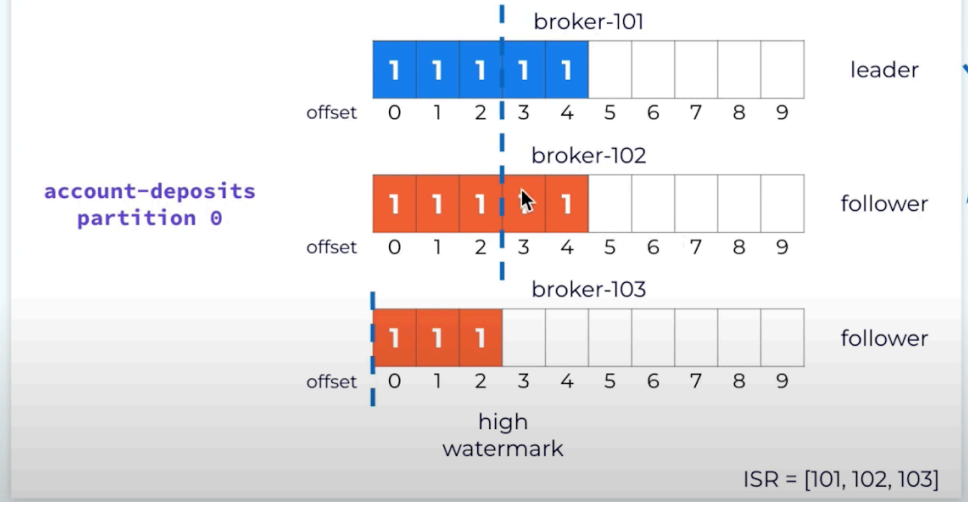
1

FetchRequest
offset: 3

1) Subsequent fetch request implicitly confirms receipt of previously fetched records
2) Leader commits records once all followers in ISR have confirmed receipt
3) Only committed records are exposed to consumers

Advancing the Follower High Watermark

CONFLUENT Developer

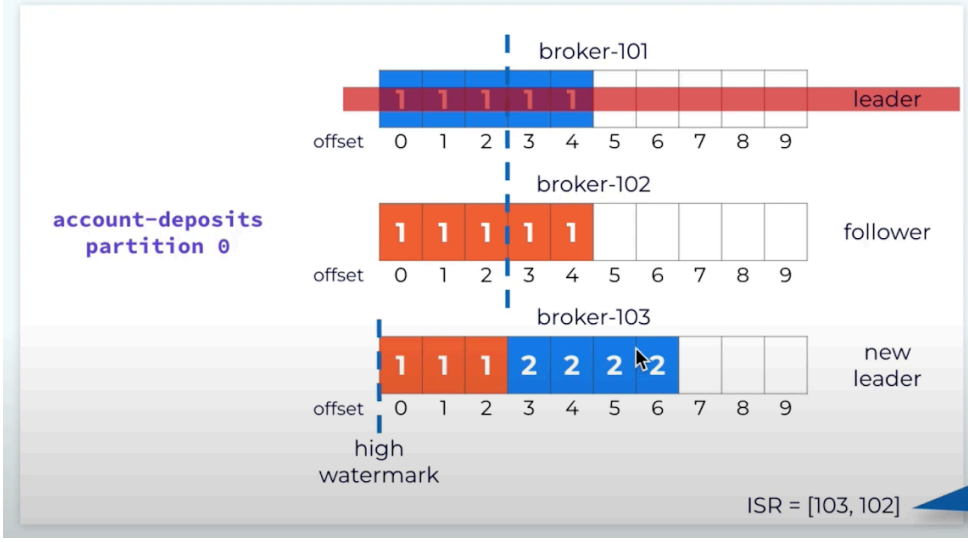



```
FetchResponse
 HighwaterMarkOffset: 3
 records: [
  (offset: 3, epoch: 1),
  (offset: 4, epoch: 1)
 ]
```

Subsequent fetch responses include new high watermark

Handling Leader Failure

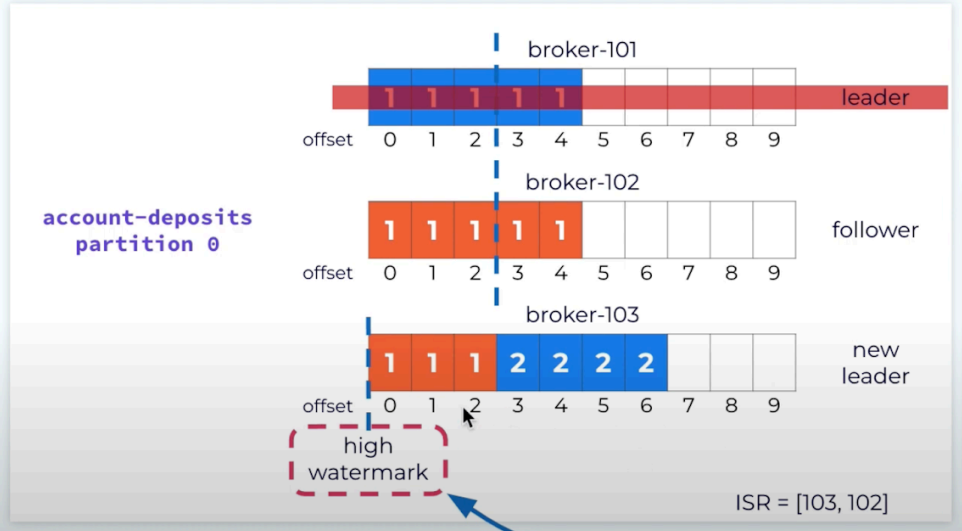
CONFLUENT Developer



New leader elected from ISR and propagated through control plane

Temporary Decreased High Watermark

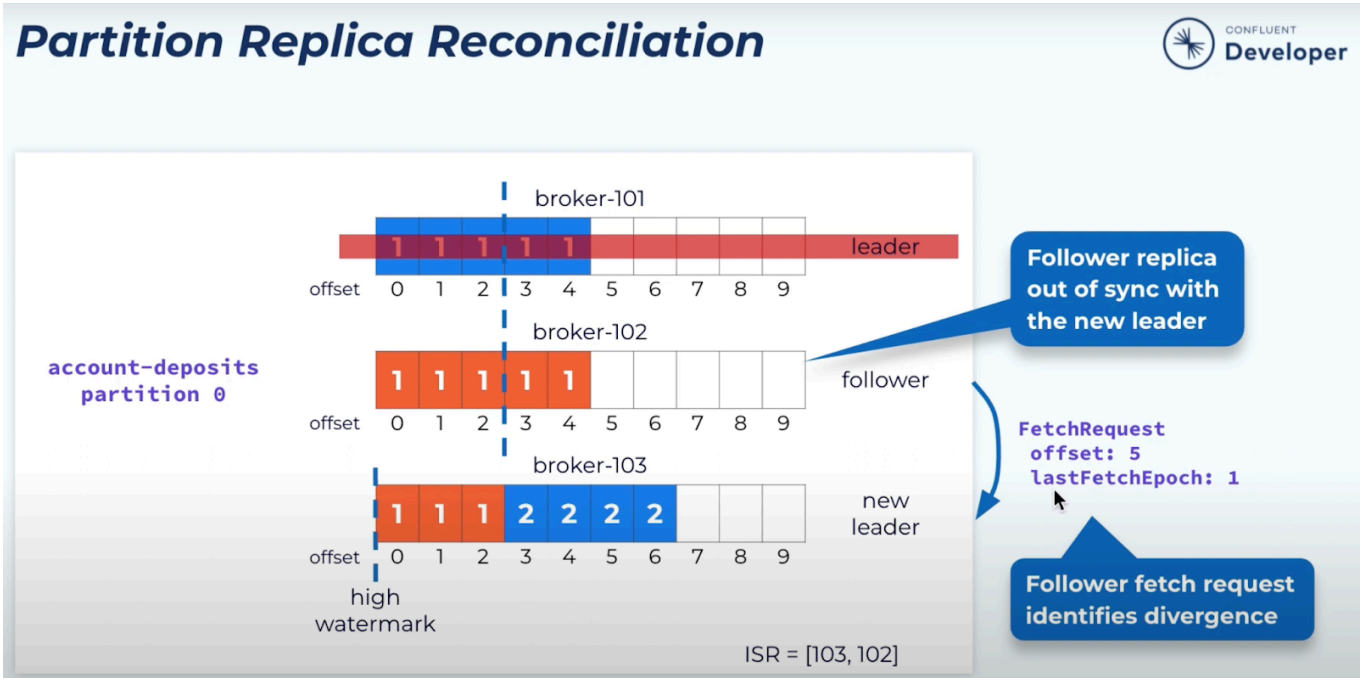
CONFLUENT Developer



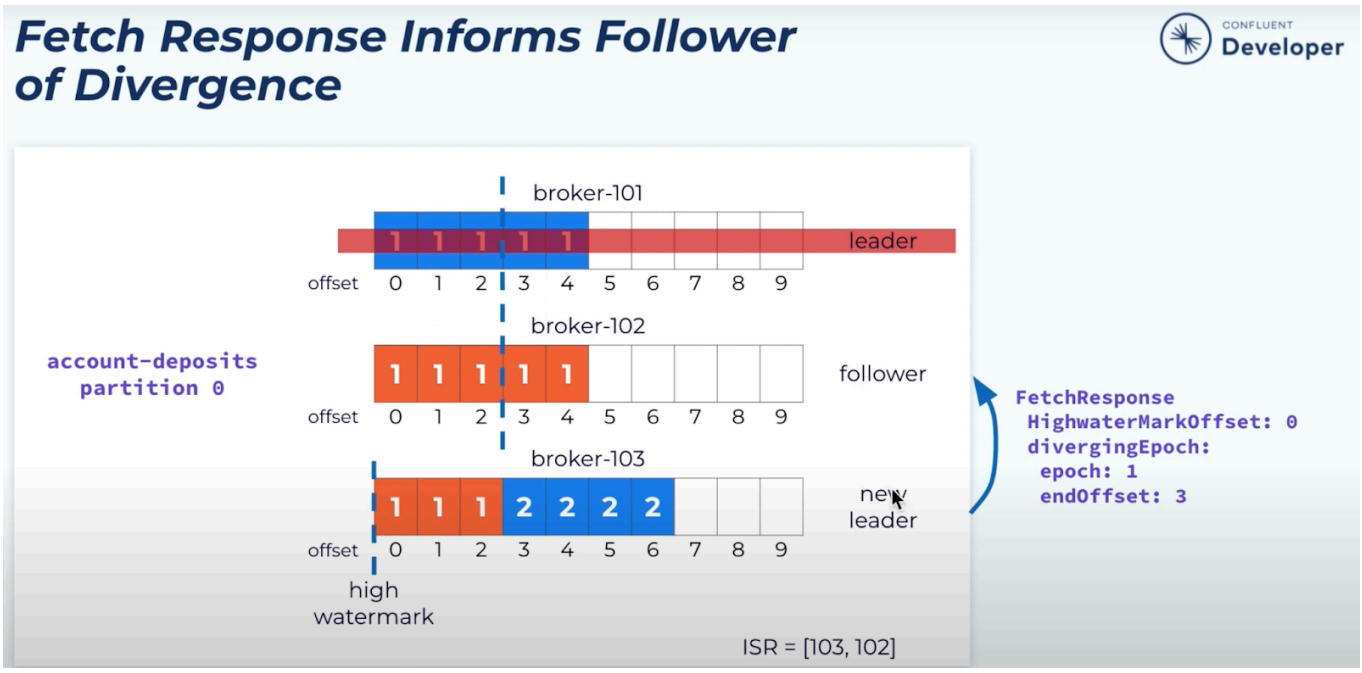
New leader high watermark could be less than true high watermark

- If consumer tries to read data in between, it gets a retriable OFFSET_NOT_AVAILABLE error
- Once high watermark catches up, normal consumption continues

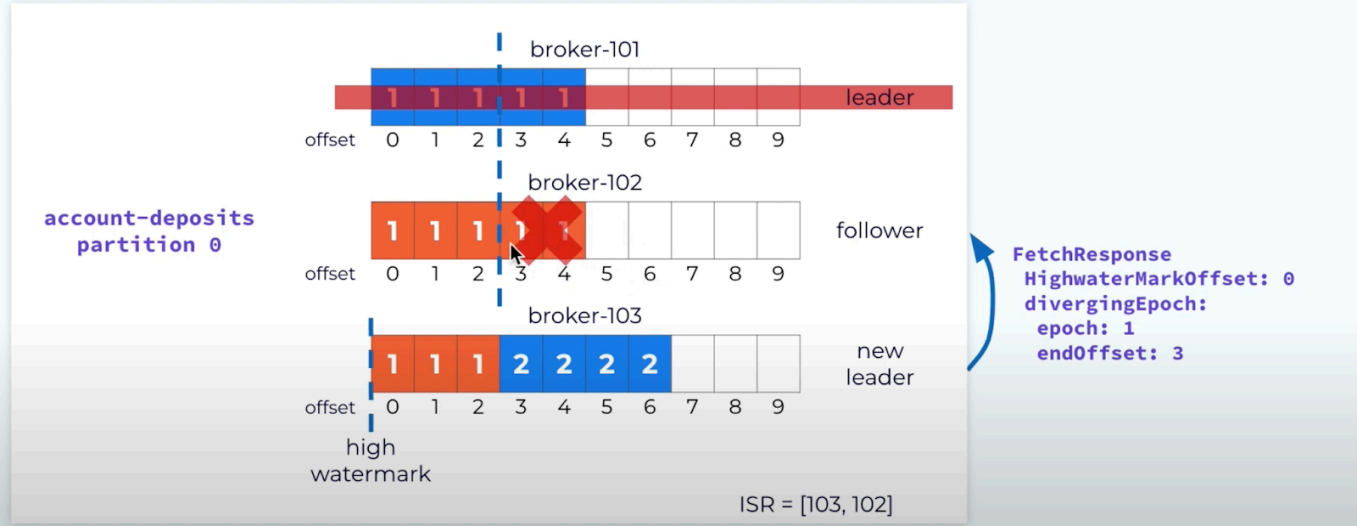
Partition Replica Reconciliation



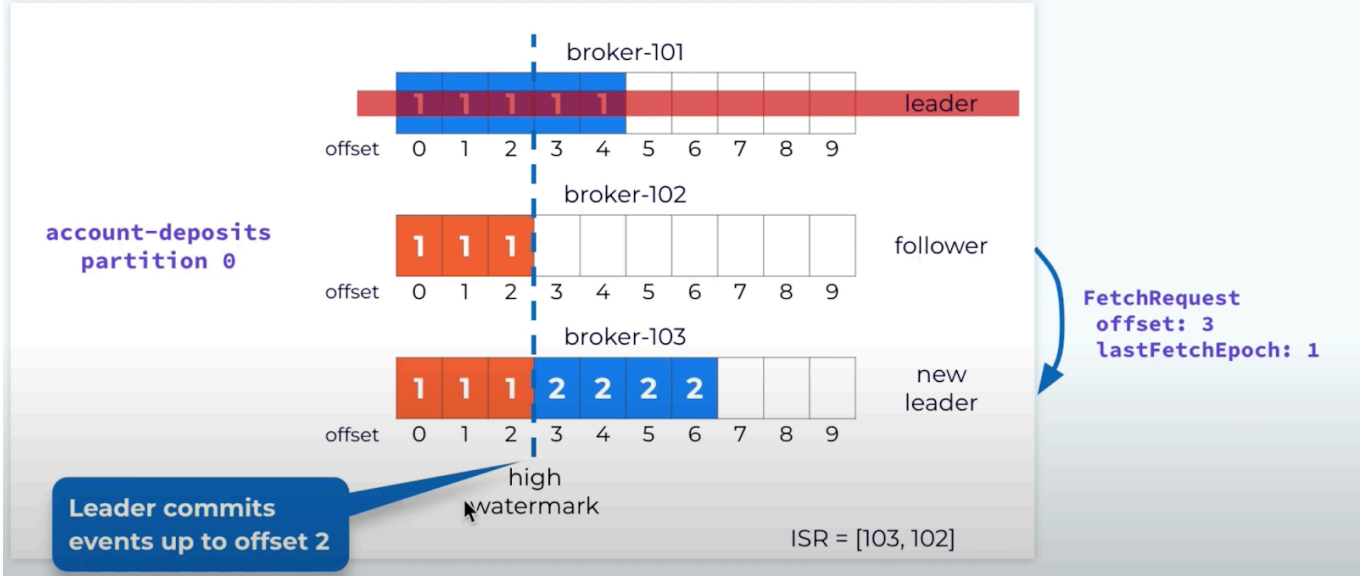
Fetch Response Informs Follower of Divergence



Follower Truncates Log To Match Leader Log

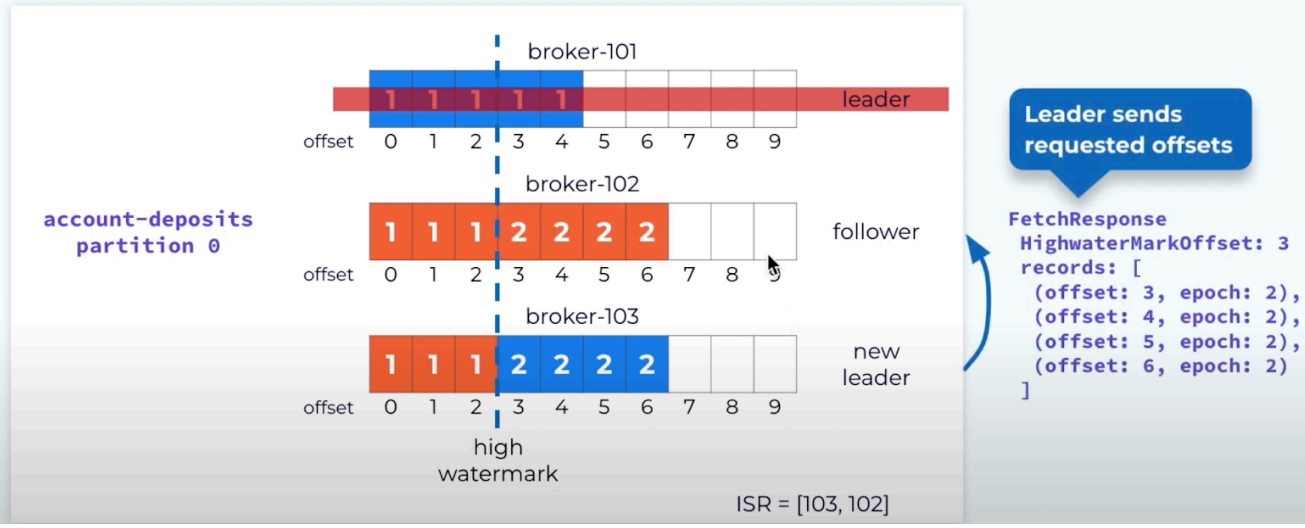


Subsequent Fetch with Updated Offset & Epoch



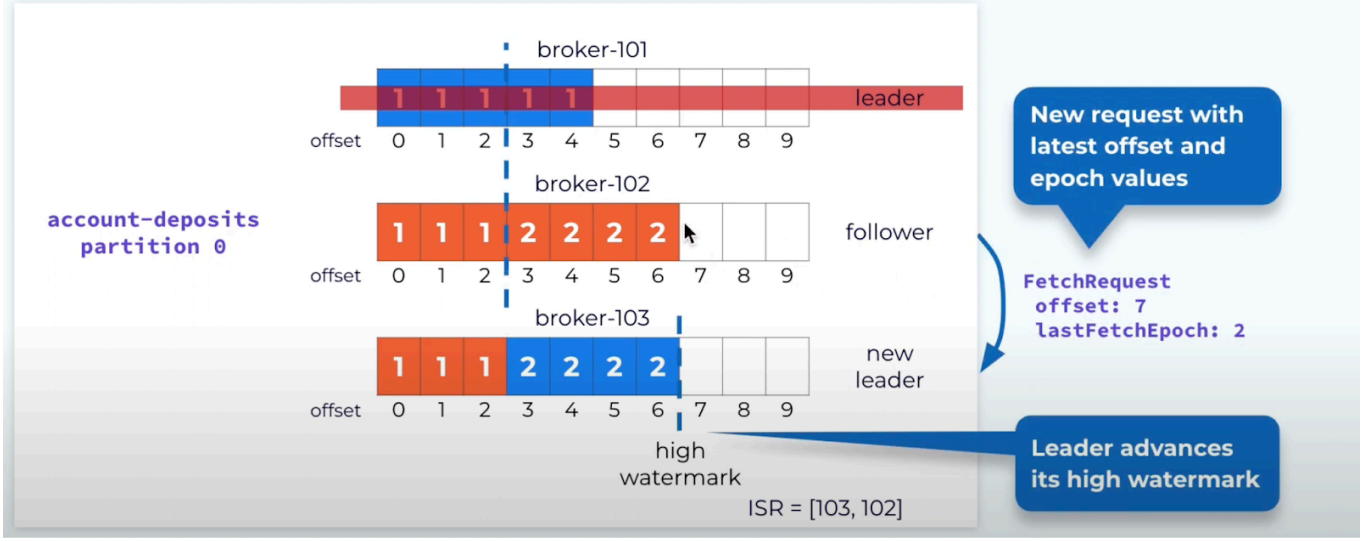
Follower 102 Reconciled

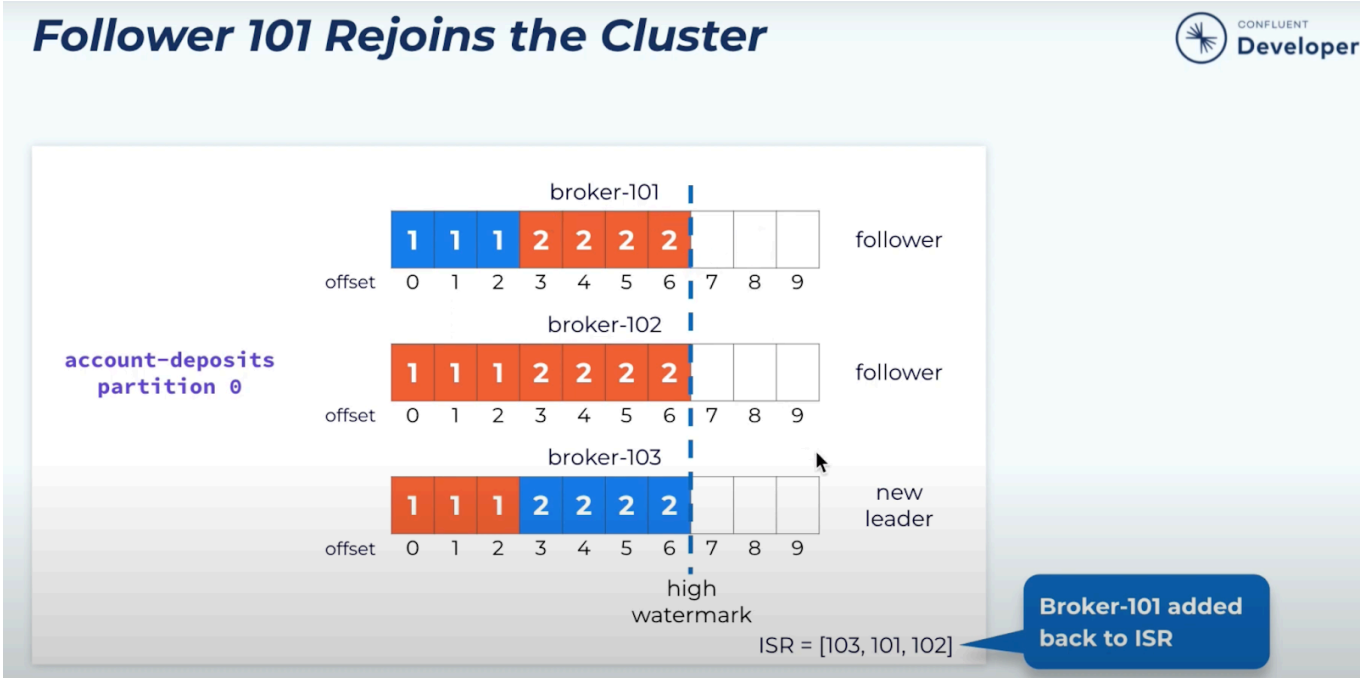
CONFLUENT Developer

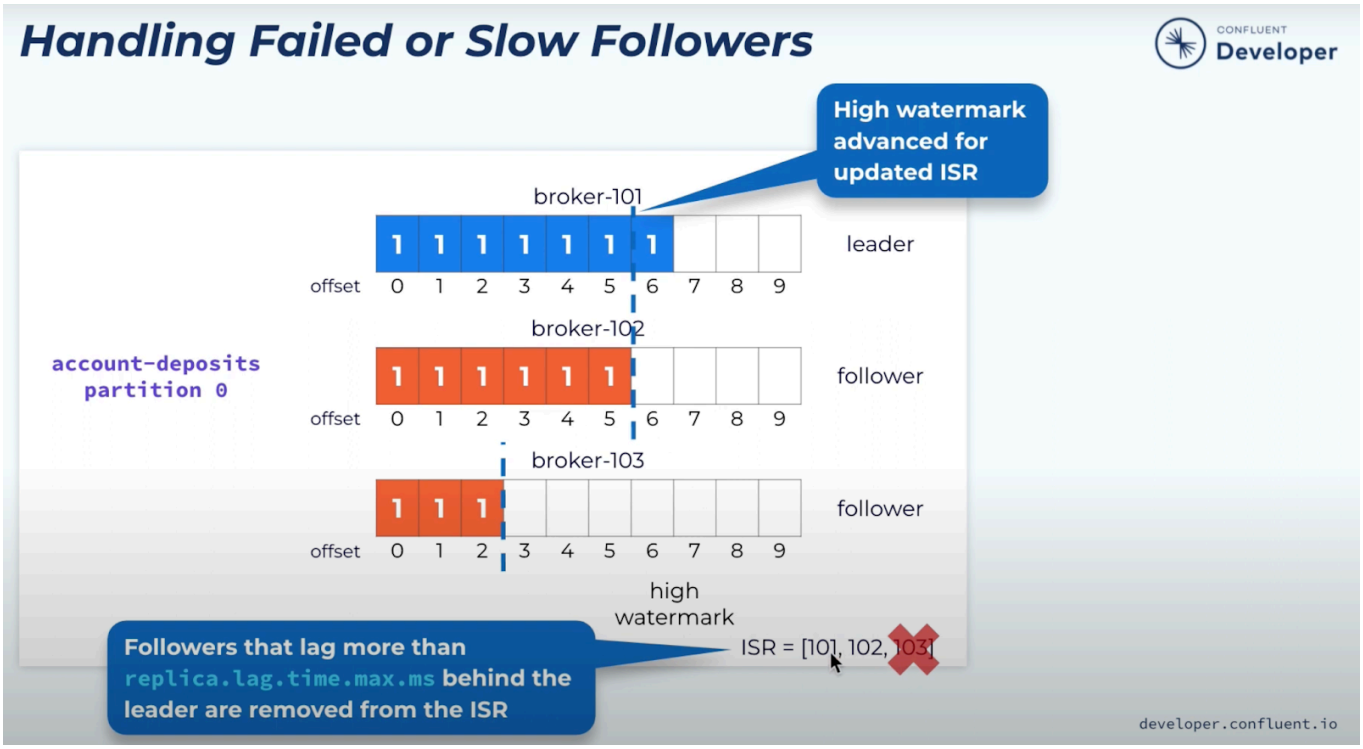


Follower 102 Acknowledges New Records

CONFLUENT Developer

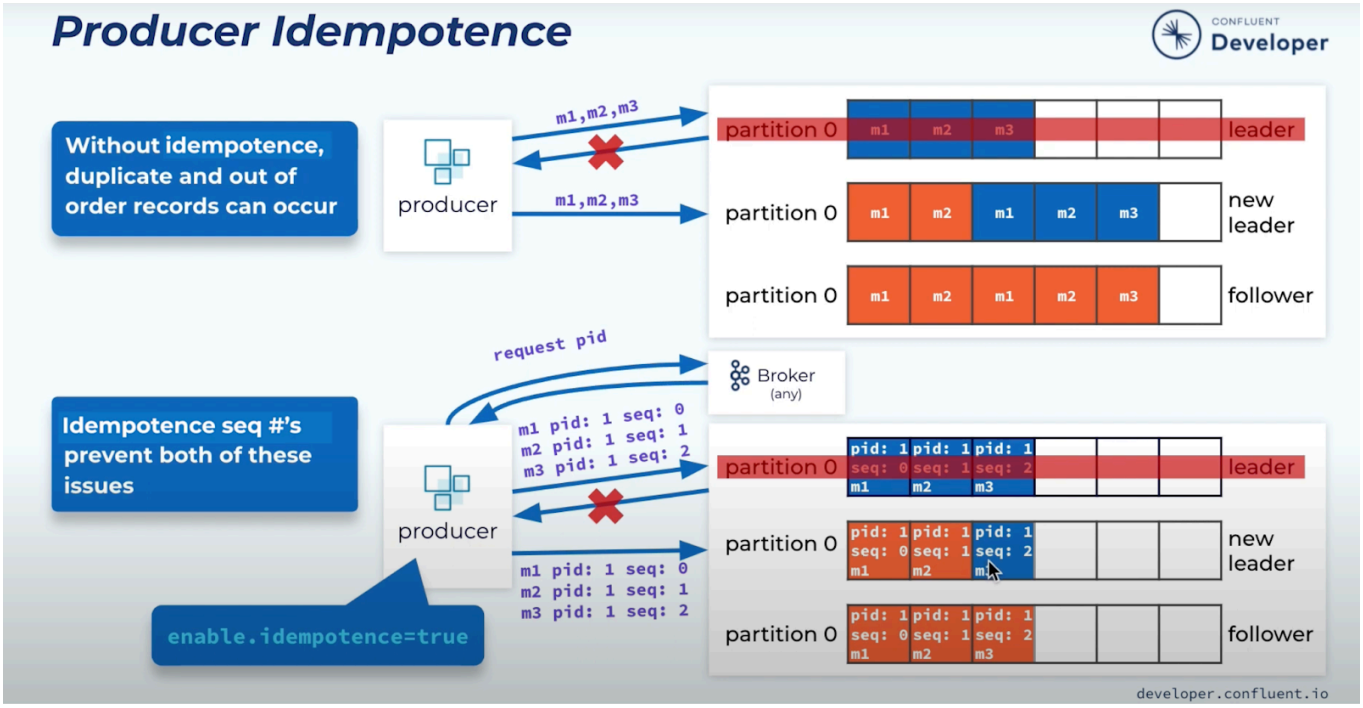




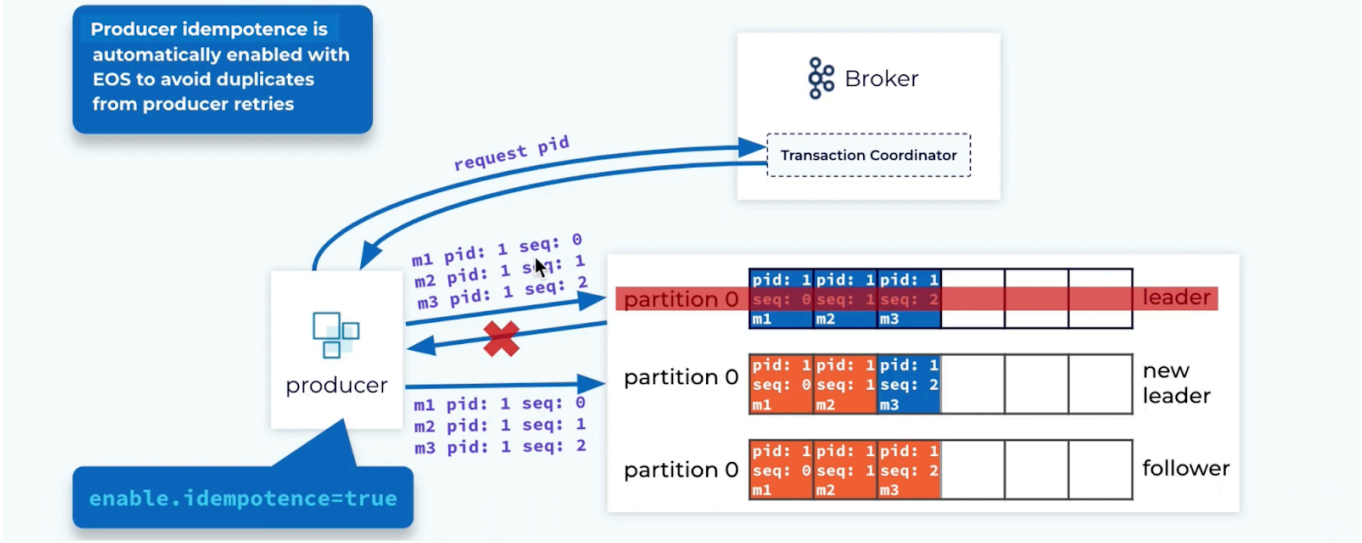


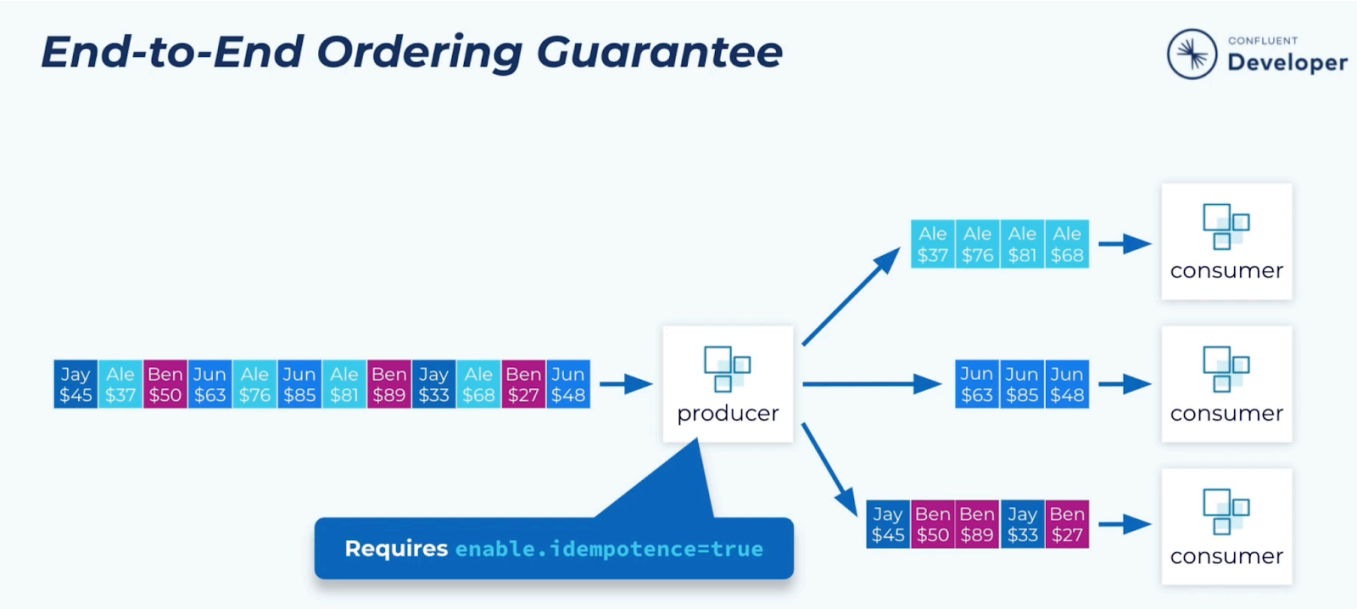
Producer Idempotence:

Producer Idempotence

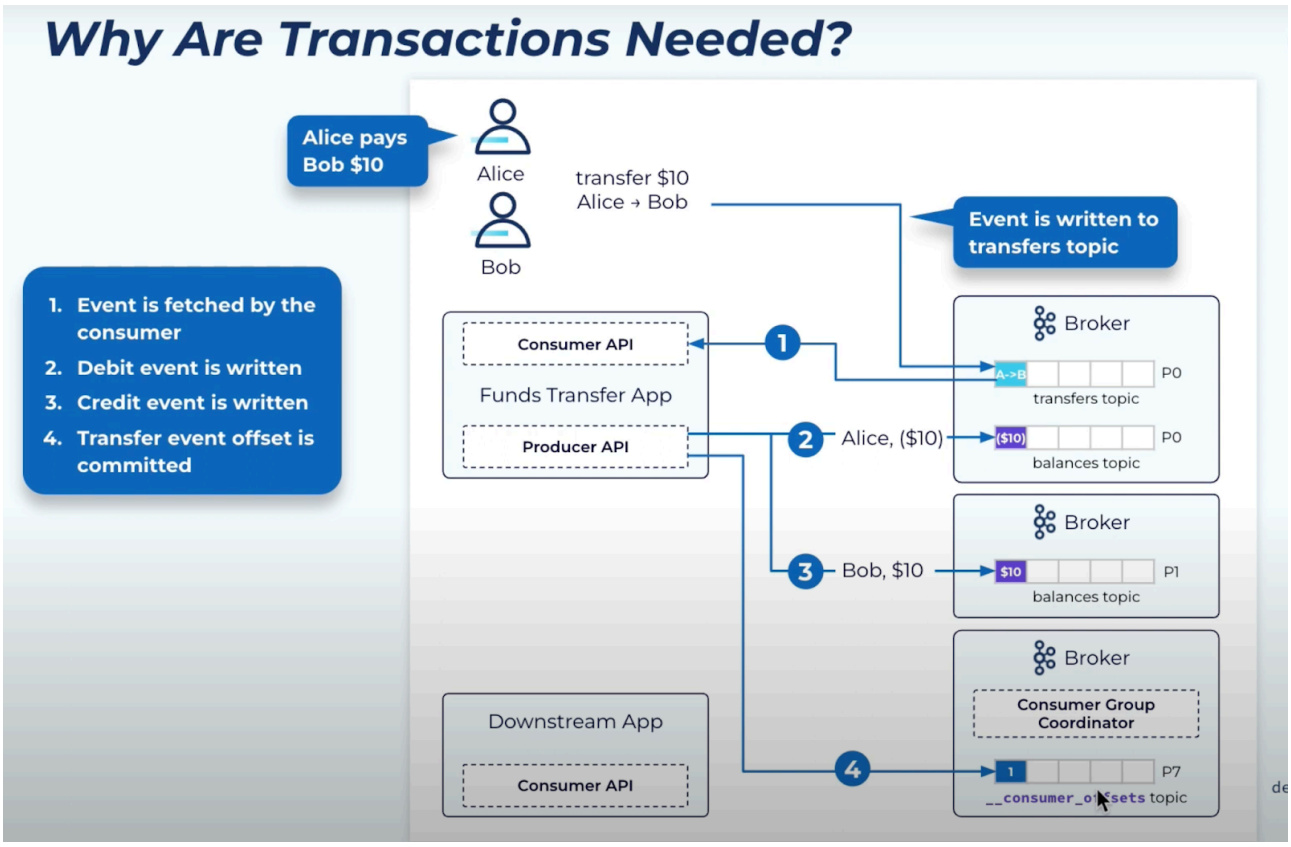


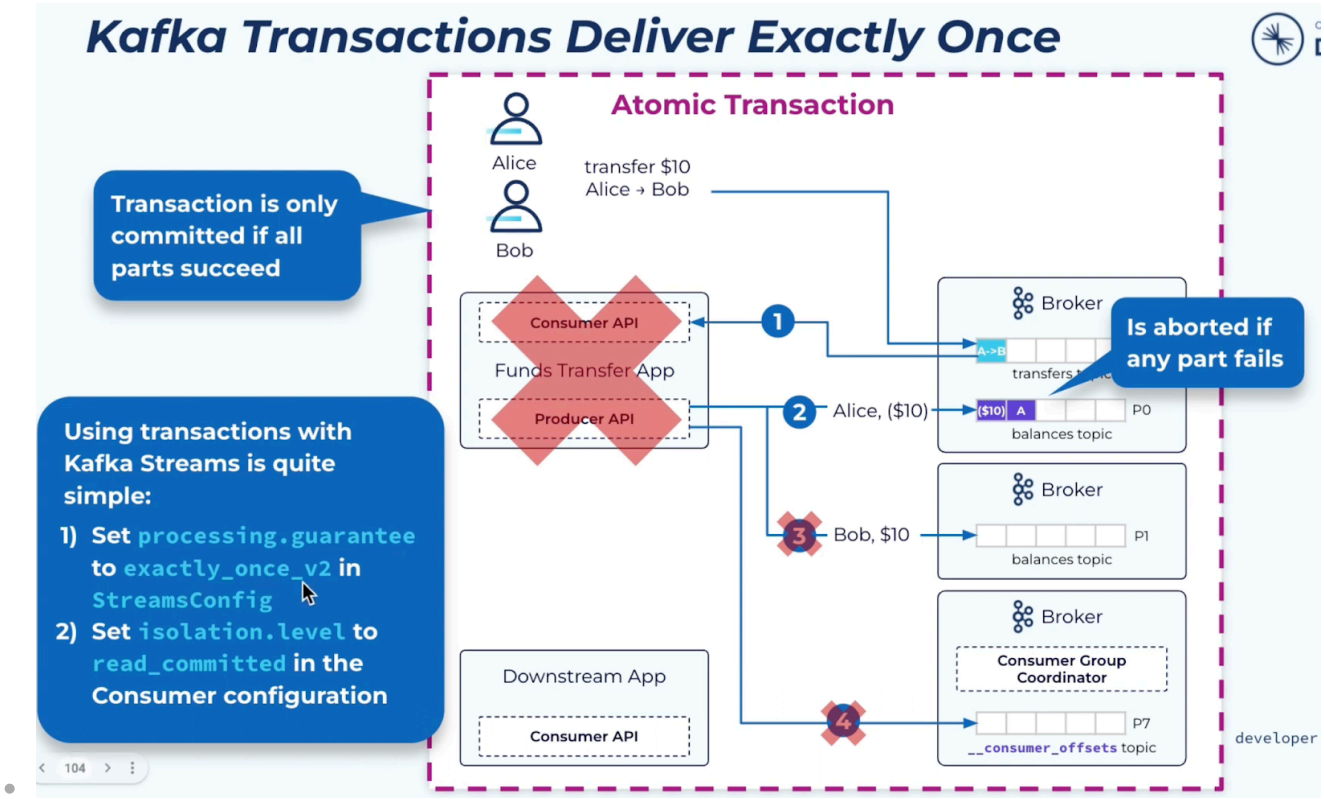
Transactions: Producer Idempotence

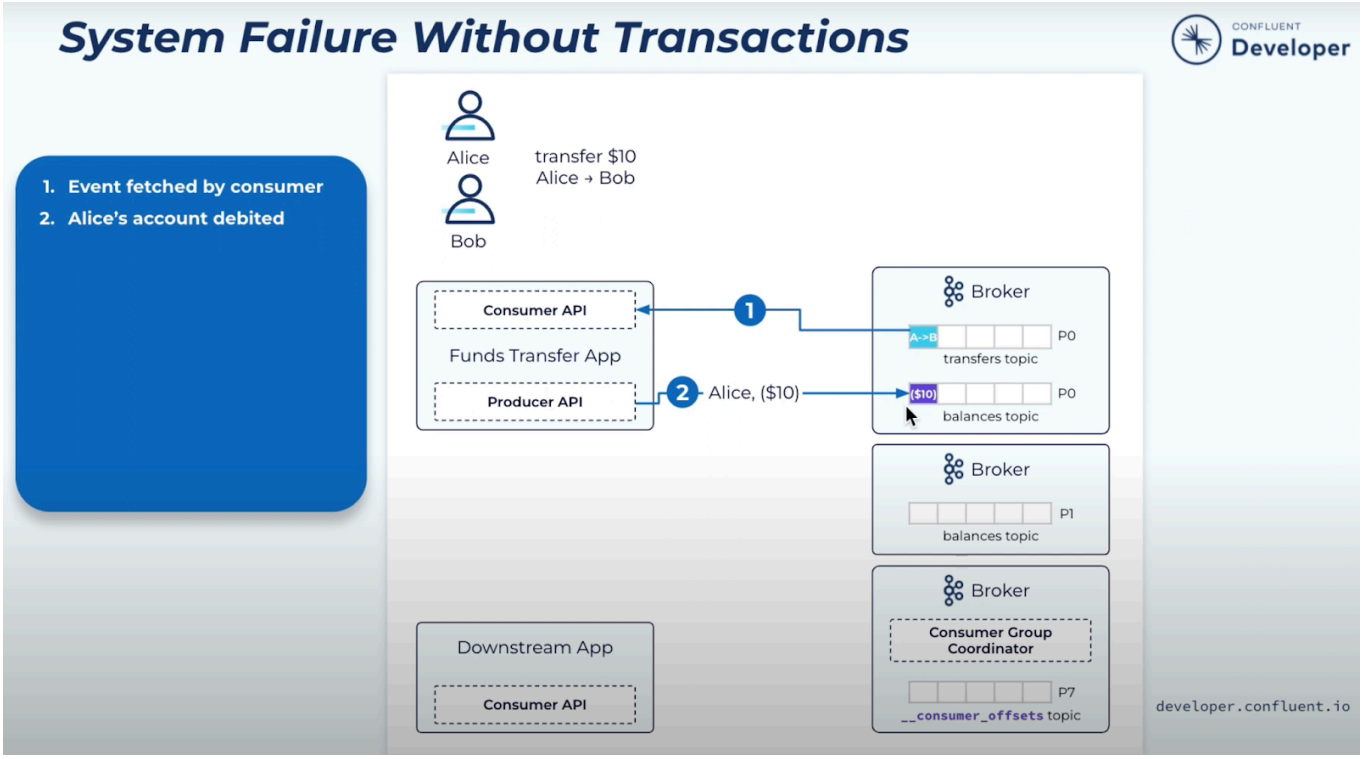




Transactions:




Kafka Transactions Deliver Exactly Once
Atomic Transaction
Alice
Bob
transfer $10
Alice → Bob
Transaction is only committed if all parts succeed
Consumer API
Funds Transfer App
Producer API
Broker
transfers topic
balances topic
Is aborted if any part fails
Alice, ($10)
Bob, $10
P0
P1
P7
Using transactions with Kafka Streams is quite simple:
1) Set processing.guarantee to exactly_once_v2 in StreamsConfig
2) Set isolation.level to read_committed in the Consumer configuration
Downstream App
Consumer API
Consumer Group Coordinator
__consumer_offsets topic
developer


System Failure Without Transactions
CONFLUENT
Developer
1. Event fetched by consumer
2. Alice's account debited
Alice
Bob
transfer $10
Alice → Bob
Consumer API
Funds Transfer App
Producer API
Broker
transfers topic
balances topic
Alice, ($10)
P0
P1
P7
Downstream App
Consumer API
Consumer Group Coordinator
__consumer_offsets topic
developer.confluent.io

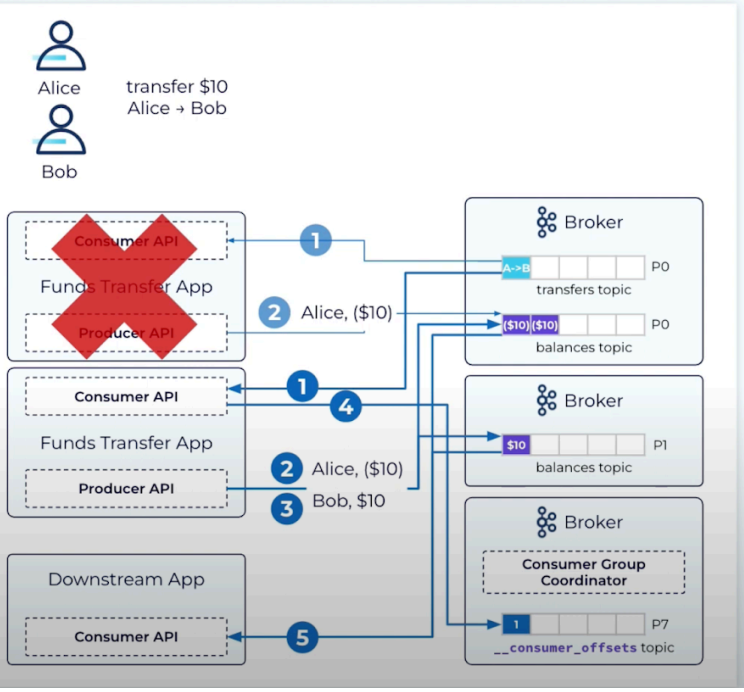
System Failure Without Transactions

CONFLUENT Developer

1. Event fetched by consumer
2. Alice's account debited

Application instance fails without committing offset and new application instance starts

1. **Event fetched by consumer**
2. **Alice's account is debited a second time**
3. **Bob's account is credited**
4. **Consumer offset committed**
5. **Two debit events processed by downstream consumer**

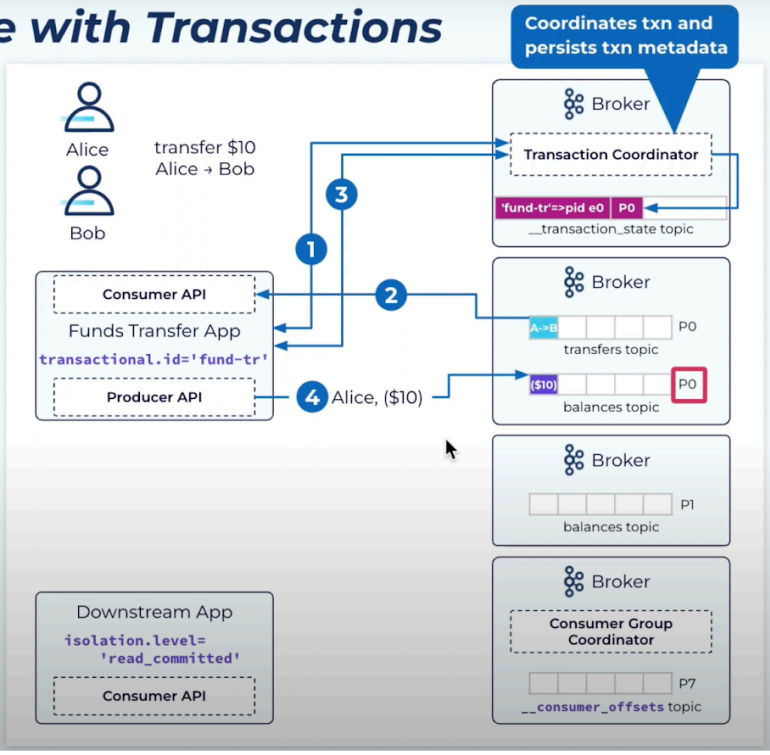


developer.confluent.io

System Failure with Transactions

CONFLUENT Developer

1. **Requests txn ID, is returned PID and txn epoch**
2. **Event fetched by consumer**
3. **Notifies coordinator of partition being written to**
4. **Alice's account debited**



developer.confluent.io

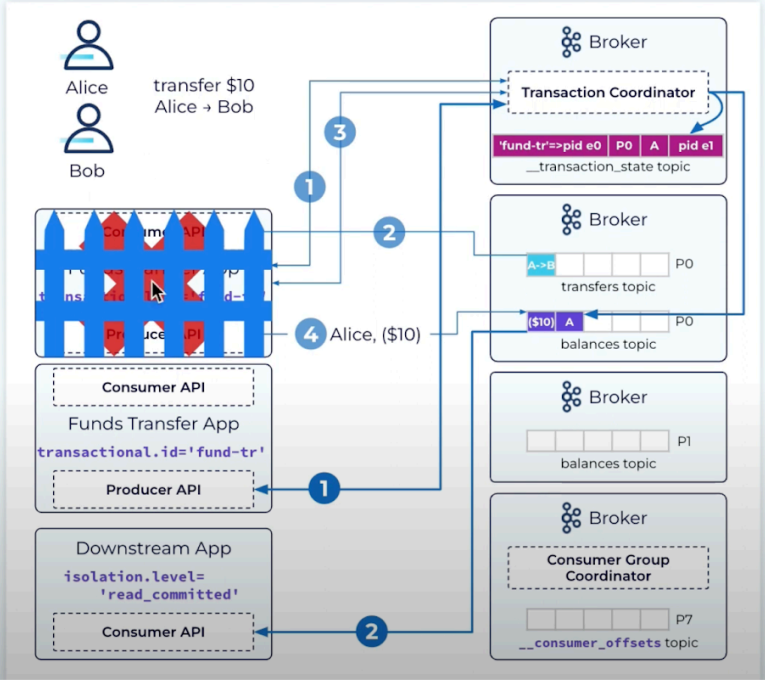
System Failure with Transactions

CONFLUENT Developer

1. Requests txn ID, is returned PID and txn epoch
2. Event fetched by consumer
3. Notifies coordinator of partition being written to
4. Alice's account debited

Application instance fails without committing offset and new application instance starts

1. **New instance requests txn ID**
   a. **Coordinator fences previous instance by aborting pending txn and bumping up epoch**
2. **Downstream consumer with `read_committed` discards aborted events**

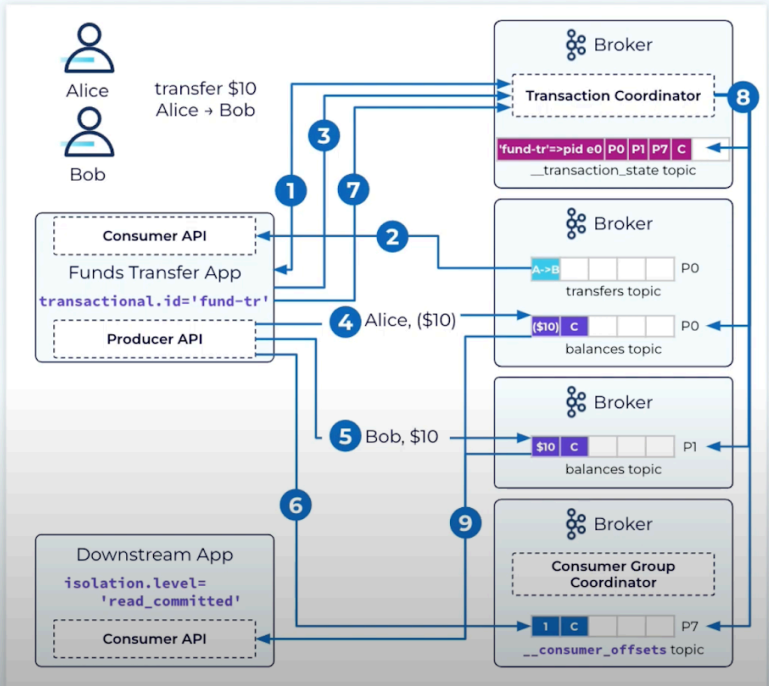


developer.confluent.io

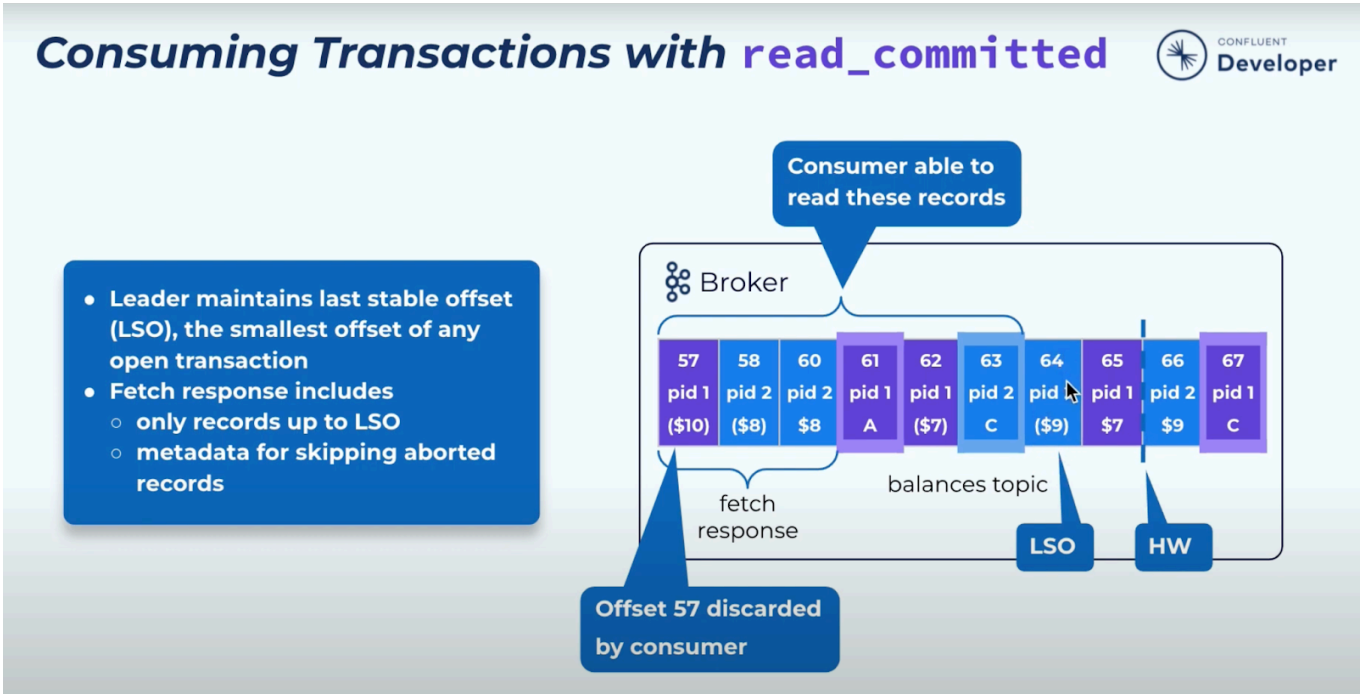
System with Successful Committed Transaction

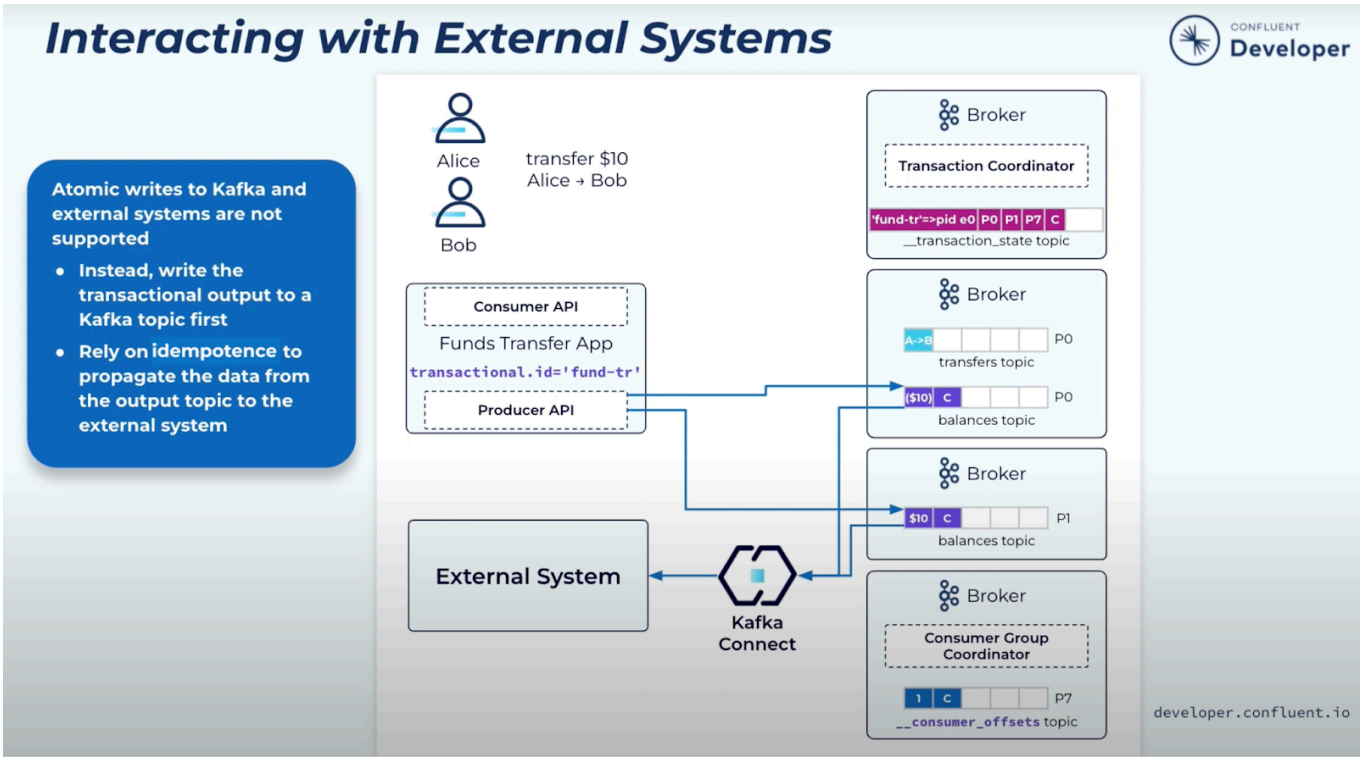
CONFLUENT Developer

1. **Requests txn ID and assigned PID and epoch**
2. **Event fetched by consumer**
3. **Notifies coordinator of partition being written to**
4. **Alice's account debited**
5. **Bob's account credited**
6. **Consumer offset committed**
7. **Notify coordinator that transaction is complete**
8. **Coordinator writes commit markers to p0, p1, p7**
9. **Downstream consumer with `read_committed` processes committed events**



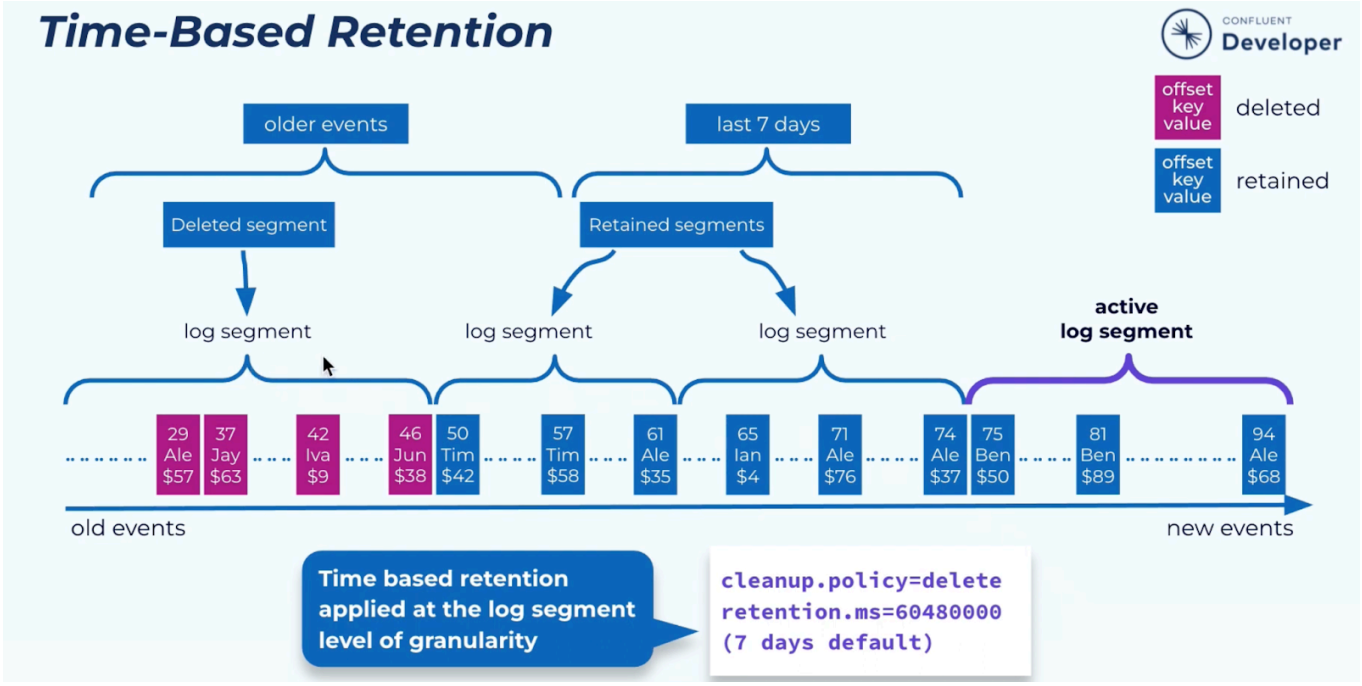
developer.confluent.io


Consuming Transactions with read_committed
CONFLUENT Developer
Leader maintains last stable offset (LSO), the smallest offset of any open transaction
Fetch response includes
only records up to LSO
metadata for skipping aborted records
Consumer able to read these records
Broker
57 pid 1 ($10)
58 pid 2 ($8)
60 pid 2 $8
61 pid 1 A
62 pid 1 ($7)
63 pid 2 C
64 pid 1 ($9)
65 pid 1 $7
66 pid 2 $9
67 pid 1 C
fetch response
balances topic
LSO
HW
Offset 57 discarded by consumer

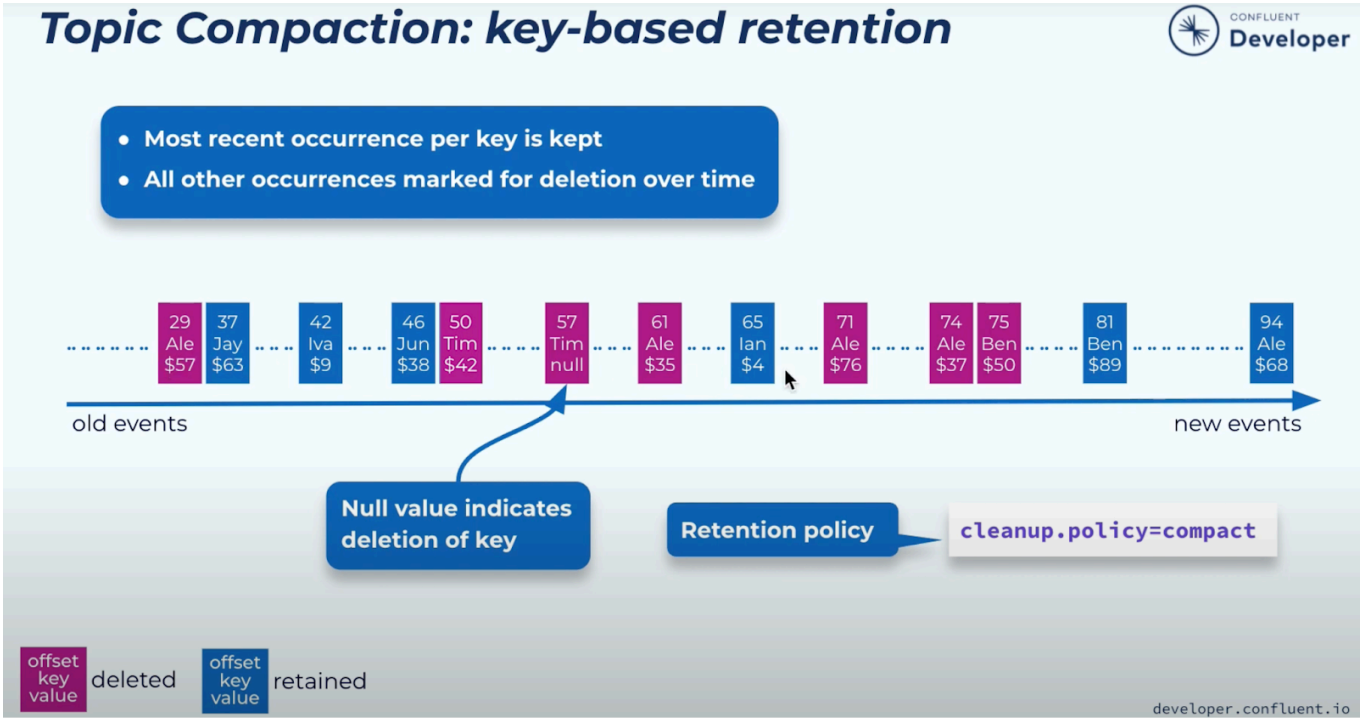

Interacting with External Systems
CONFLUENT Developer
Atomic writes to Kafka and external systems are not supported
Instead, write the transactional output to a Kafka topic first
Rely on idempotence to propagate the data from the output topic to the external system
Alice
Bob
transfer $10
Alice → Bob
Consumer API
Funds Transfer App
transactional.id='fund-tr'
Producer API
External System
Kafka Connect
Broker
Transaction Coordinator
'fund-tr'=>pid e0 P0 P1 P7 C
__transaction_state topic
Broker
A->B
P0
transfers topic
($10) C
P0
balances topic
Broker
$10 C
P1
balances topic
Broker
Consumer Group Coordinator
1 C
P7
__consumer_offsets topic
developer.confluent.io

Retention:

Time-Based Retention



Topic Compaction: key-based retention



Usage and Benefits of Topic Compaction

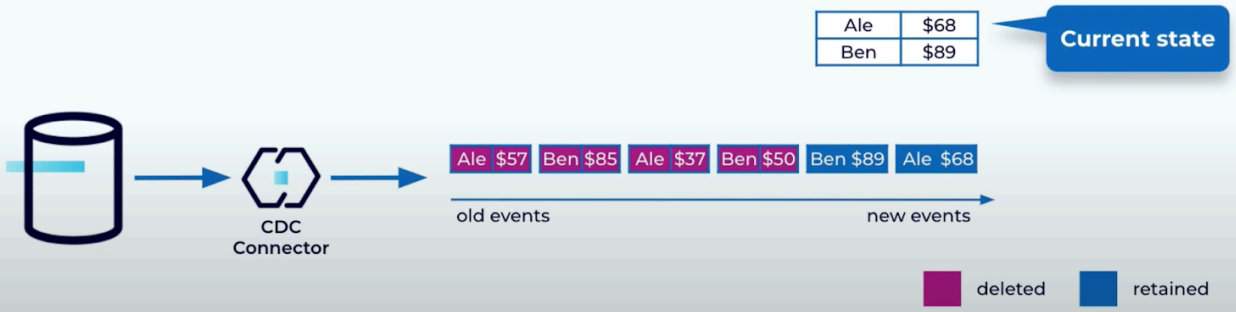
CONFLUENT Developer

Usage:

- Streaming updatable data sets (e.g. profiles, catalogue)
- KSQL state and tables

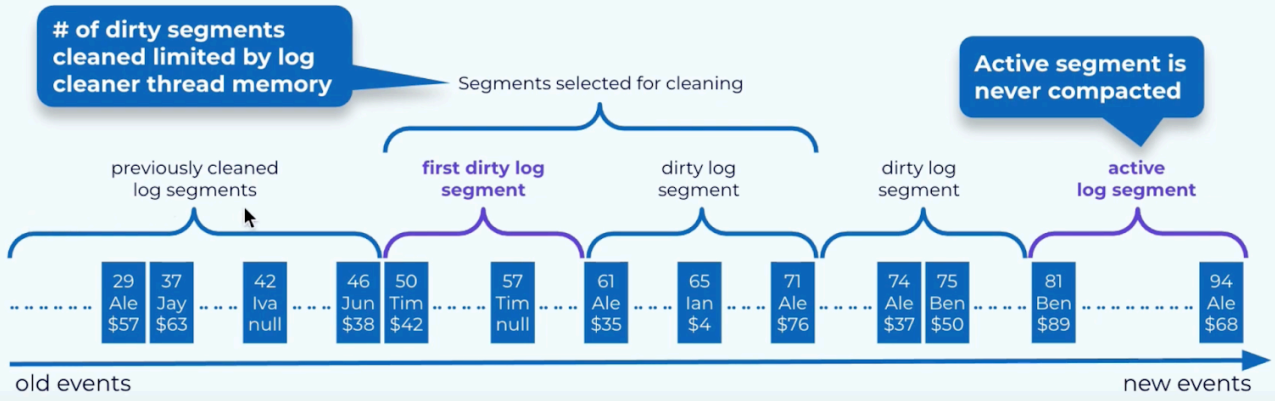
Benefits:

- Keeping latest info with bounded storage
- Allows incremental consumption and bootstrap to be done the same way



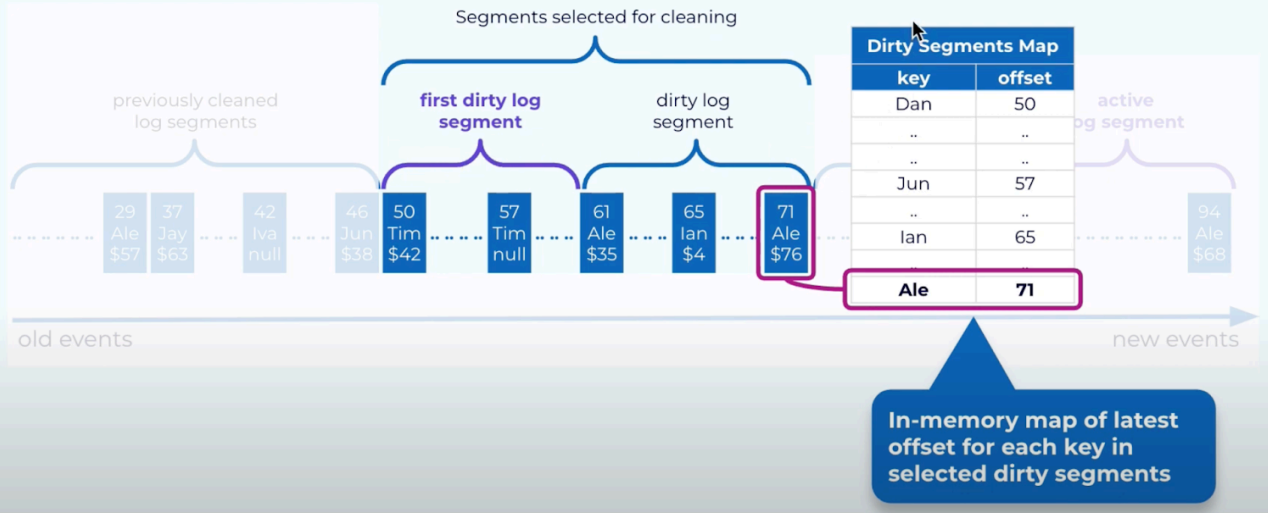
Compaction Process - Segments to Clean

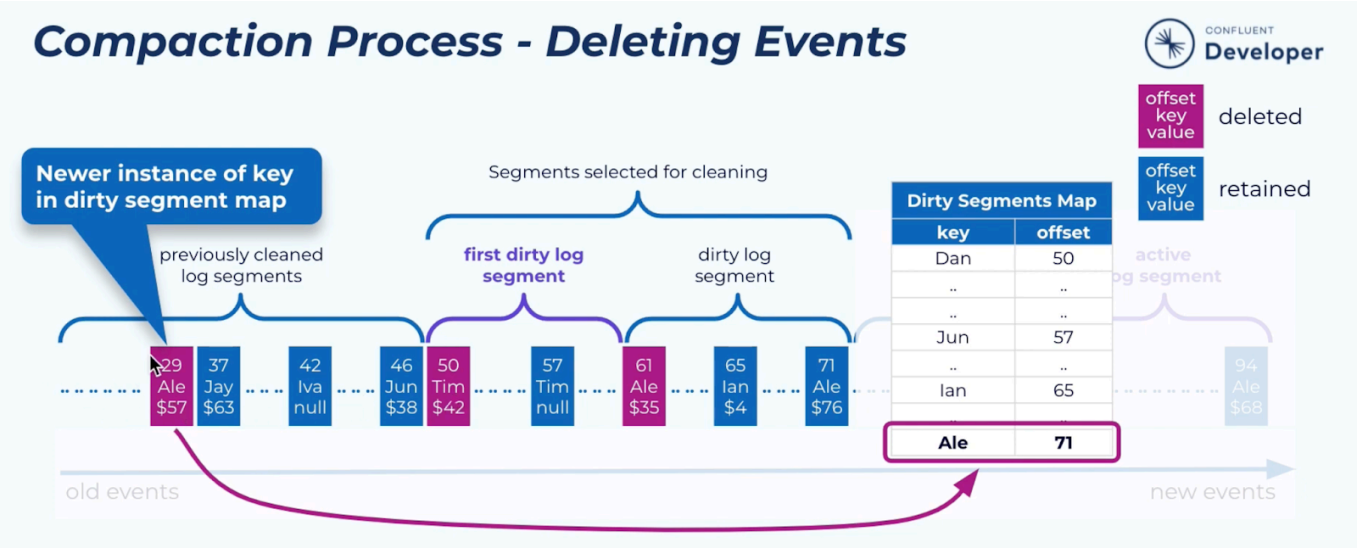
CONFLUENT Developer

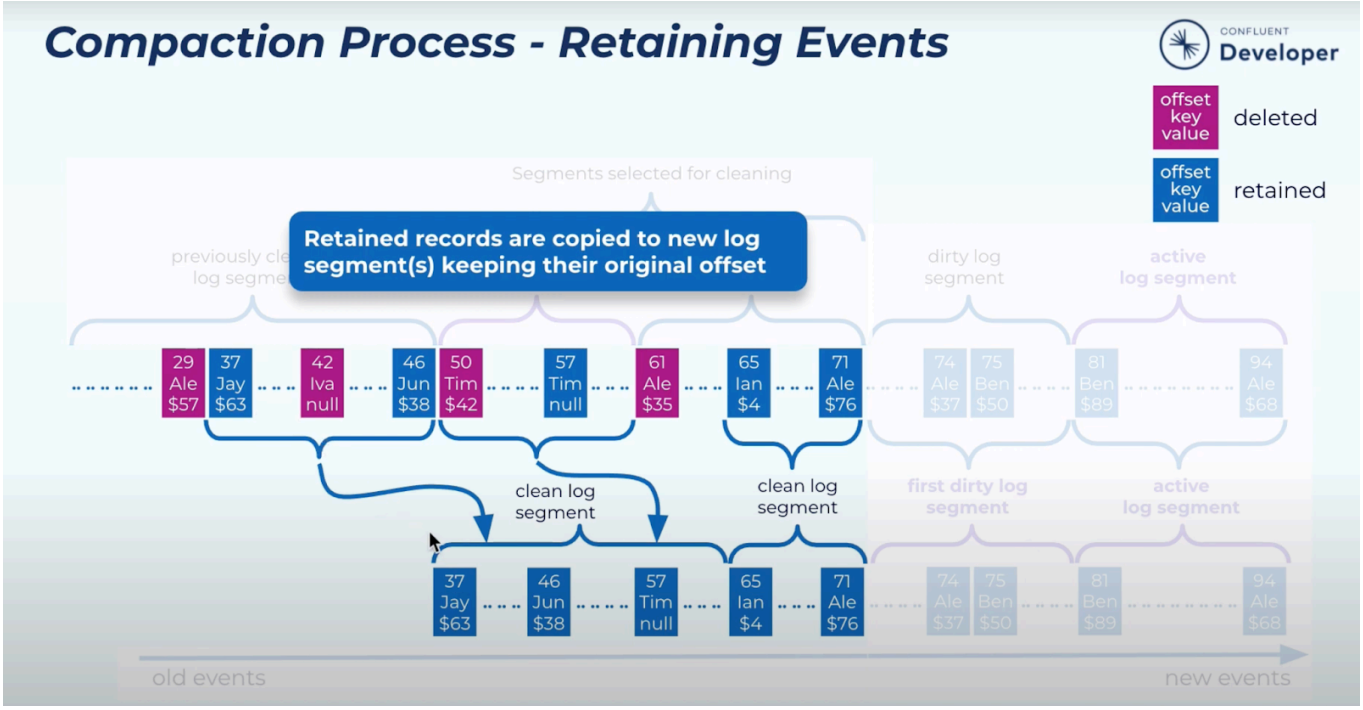


Compaction Process - Build Dirty Segment Map

CONFLUENT Developer

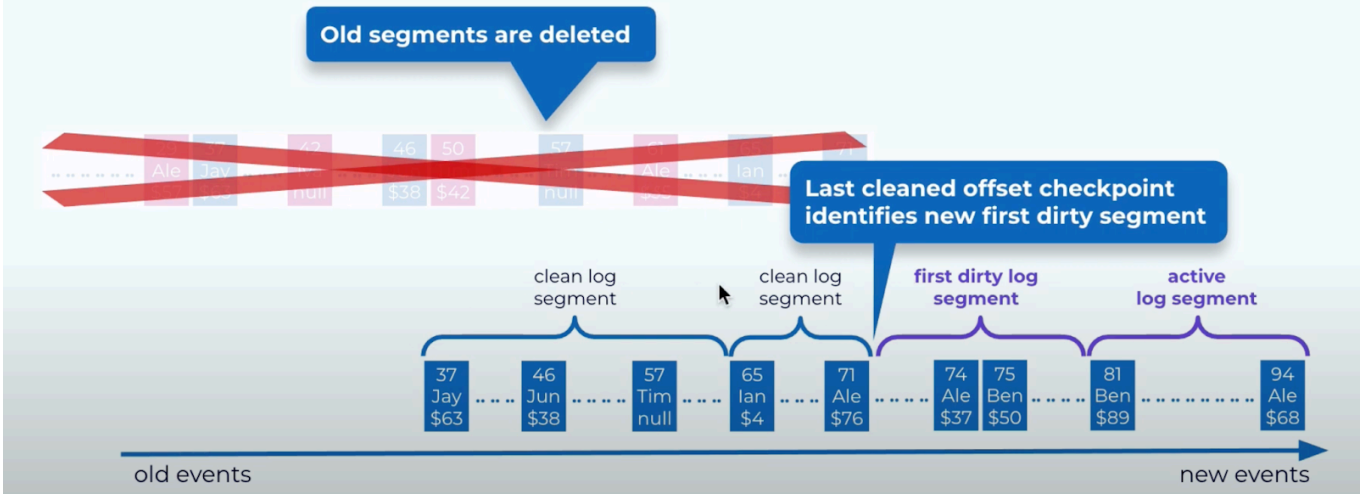



Compaction Process - Deleting Events
CONFLUENT Developer
offset key value
deleted
offset key value
retained
Newer instance of key in dirty segment map
Segments selected for cleaning
previously cleaned log segments
first dirty log segment
dirty log segment
active log segment
Dirty Segments Map
key
offset
Dan
50
Jun
57
Ian
65
Ale
71
29 Ale $57
37 Jay $63
42 Iva null
46 Jun $38
50 Tim $42
57 Tim null
61 Ale $35
65 Ian $4
71 Ale $76
94 Ale $68
old events
new events


Compaction Process - Retaining Events
CONFLUENT Developer
offset key value
deleted
offset key value
retained
Segments selected for cleaning
Retained records are copied to new log segment(s) keeping their original offset
dirty log segment
active log segment
29 Ale $57
37 Jay $63
42 Iva null
46 Jun $38
50 Tim $42
57 Tim null
61 Ale $35
65 Ian $4
71 Ale $76
74 Ale $37
75 Ben $50
81 Ben $89
94 Ale $68
clean log segment
clean log segment
first dirty log segment
active log segment
37 Jay $63
46 Jun $38
57 Tim null
65 Ian $4
71 Ale $76
74 Ale $37
75 Ben $50
81 Ben $89
94 Ale $68
old events
new events

Compaction Process - Replace Old Segments

CONFLUENT Developer

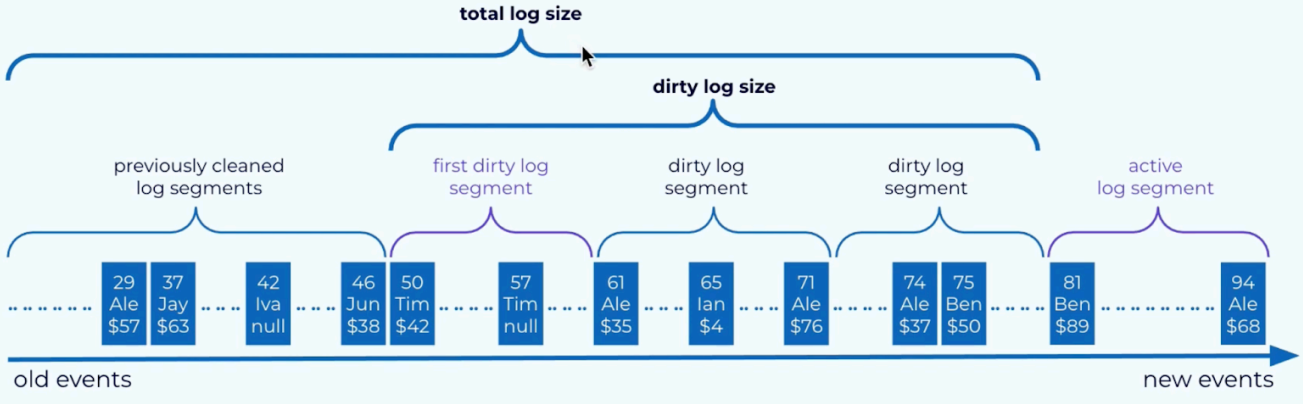


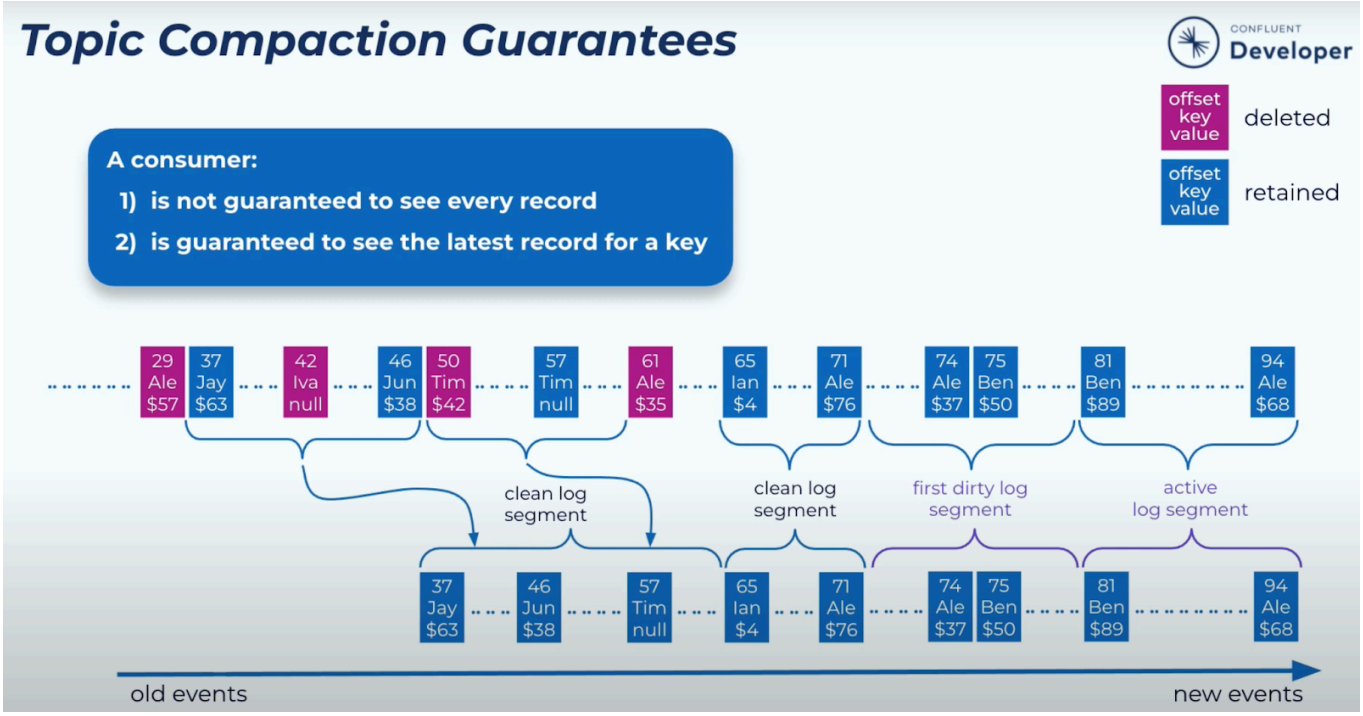
When Compaction is Triggered

CONFLUENT Developer

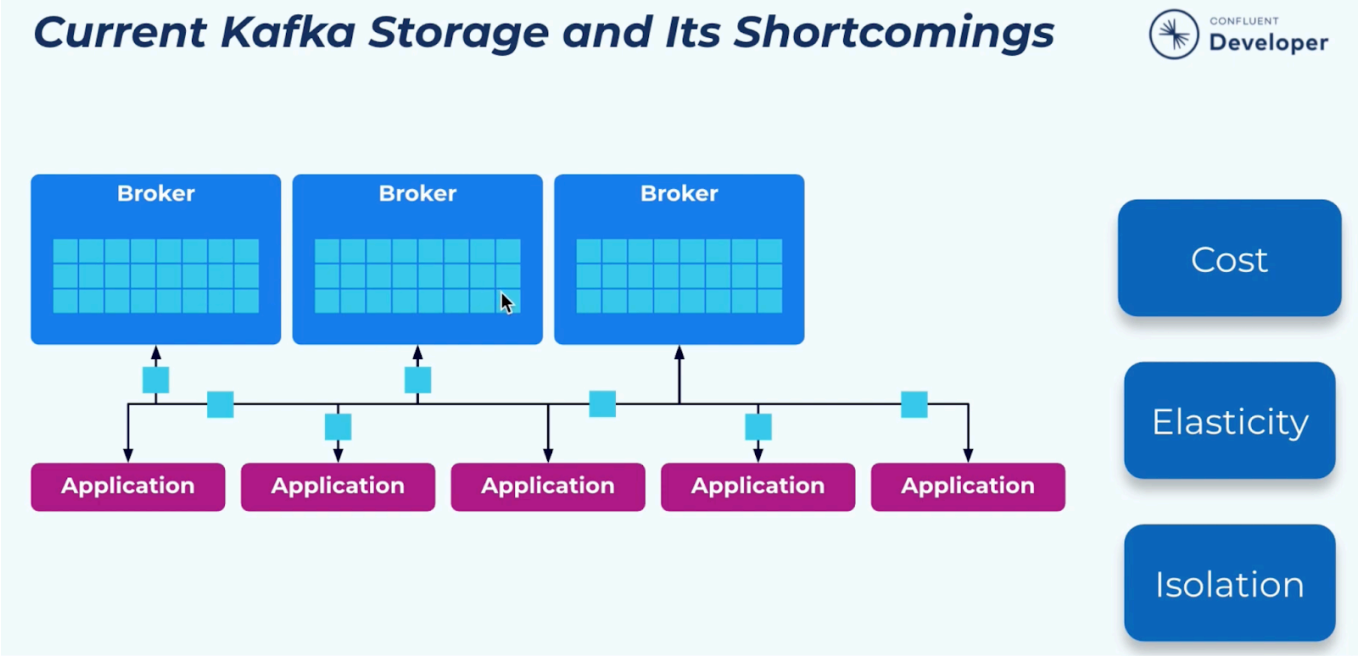
A topic partition is compacted if one of the following is true:

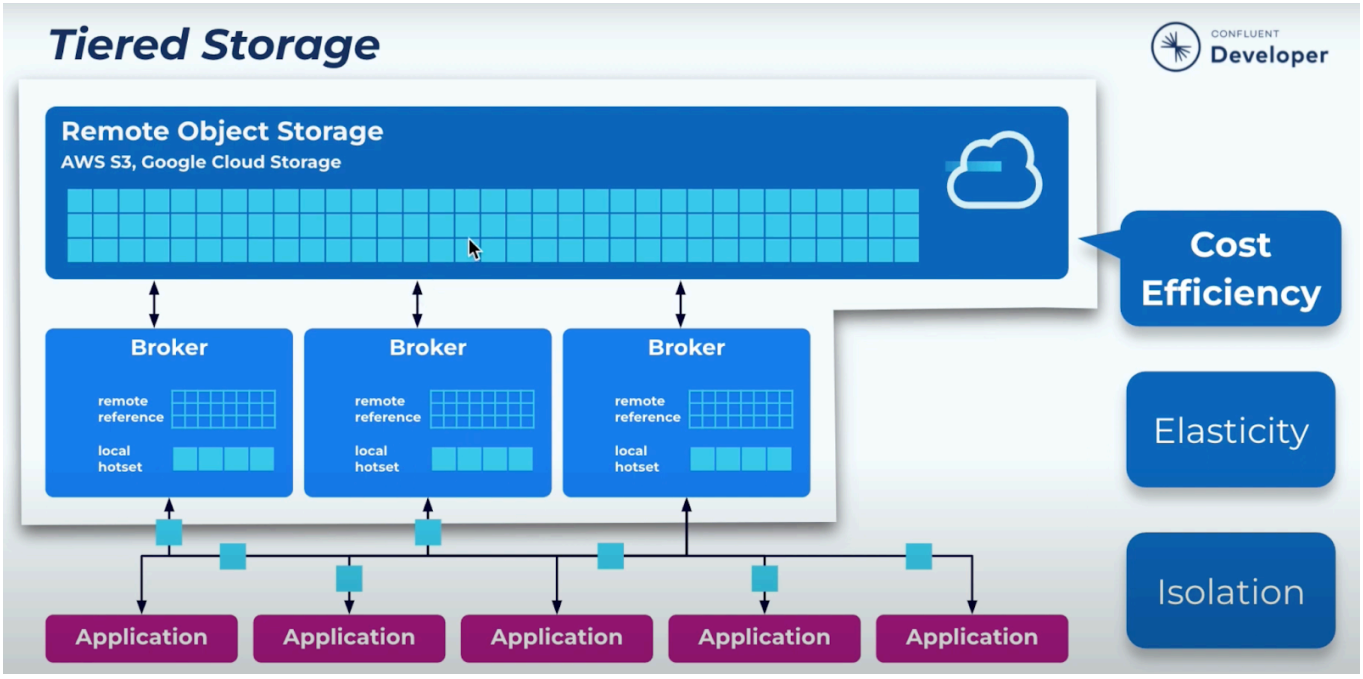
1. `dirty / total > min.cleanable.dirty.ratio` and
   `message timestamp >= current time - min.compaction.lag.ms`
2. `message timestamp < current time - max.compaction.lag.ms`

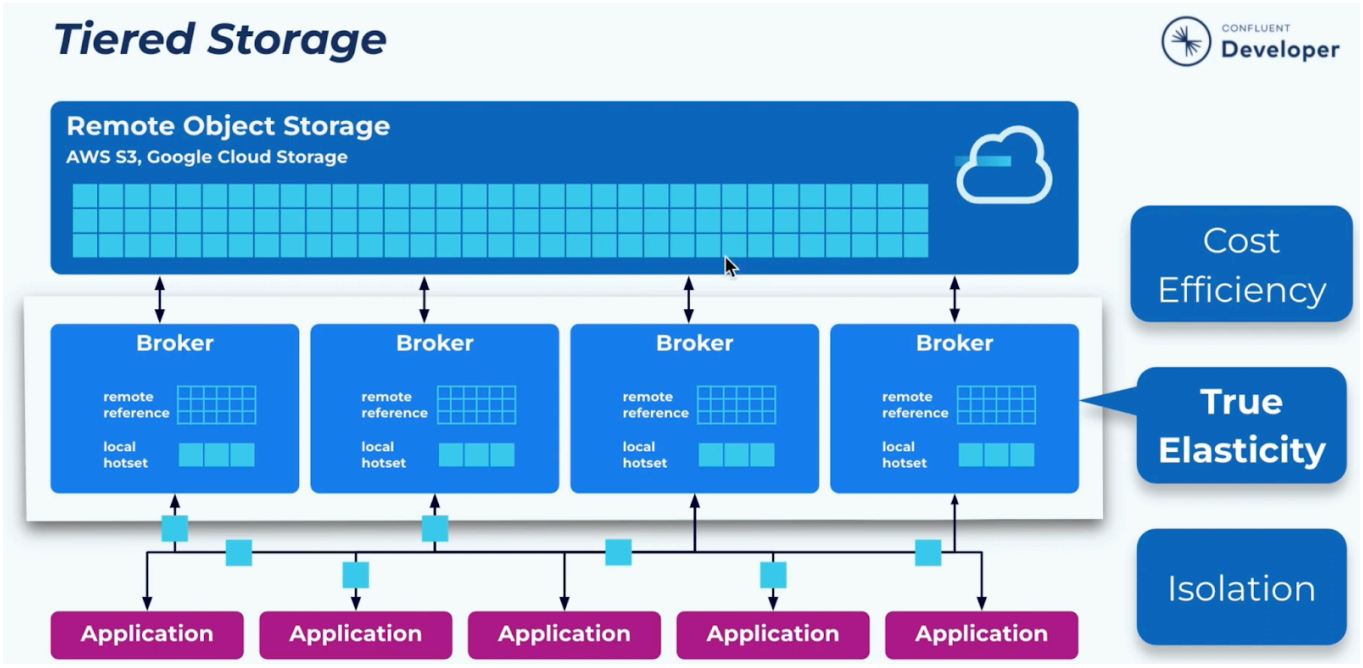




Kafka tiered storage:

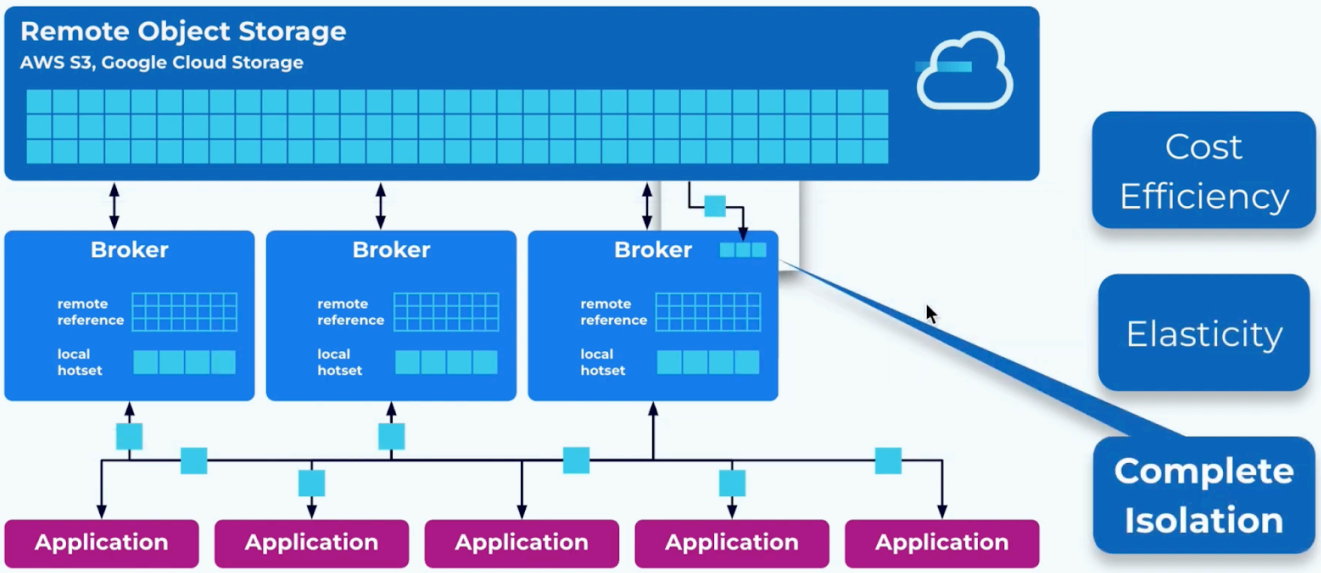



Tiered Storage
CONFLUENT
Developer
Remote Object Storage
AWS S3, Google Cloud Storage
Broker
remote reference
local hotset
Broker
remote reference
local hotset
Broker
remote reference
local hotset
Application
Application
Application
Application
Application
Cost Efficiency
Elasticity
Isolation


Tiered Storage
CONFLUENT
Developer
Remote Object Storage
AWS S3, Google Cloud Storage
Broker
remote reference
local hotset
Broker
remote reference
local hotset
Broker
remote reference
local hotset
Broker
remote reference
local hotset
Application
Application
Application
Application
Application
Cost Efficiency
True Elasticity
Isolation

Tiered Storage

CONFLUENT Developer



Fetching Tiered Data

CONFLUENT Developer