DEVIEW 2021

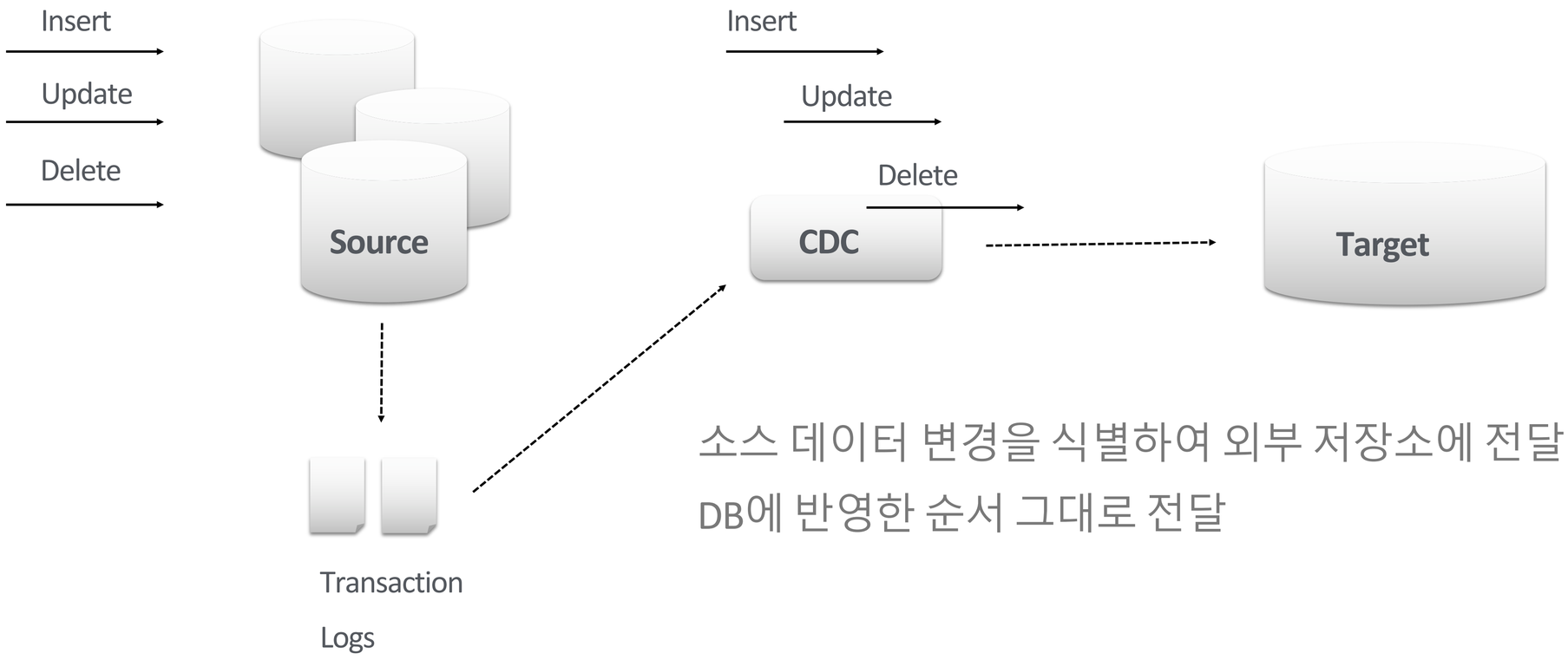
막힌 데이터의 혈을 뚫자!
Pay 플랫폼 CDC 적용 사례

Platform Labs
이혜정 이정윤 이동윤 한우람

CONTENTS

CDC란 무엇인가?

CDC란 무엇인가? Change Data Capture



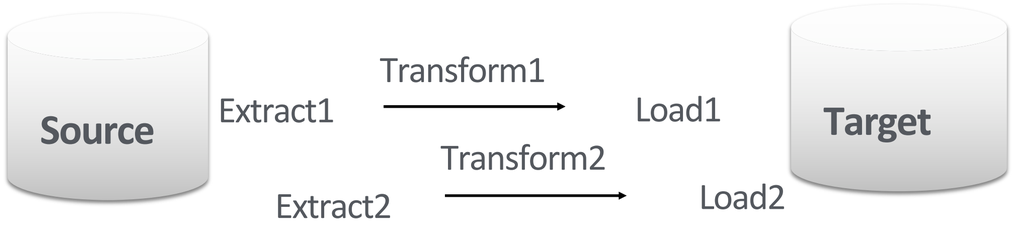
소스 데이터 변경을 식별하여 외부 저장소에 전달

DB에 반영한 순서 그대로 전달

CDC란 무엇인가? CDC가 가진 장점

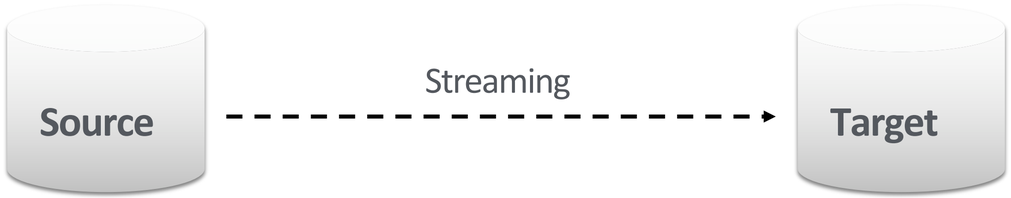
DEVIEW 2021

- Batch 처리



반복적인 ETL 발생

- **CDC**



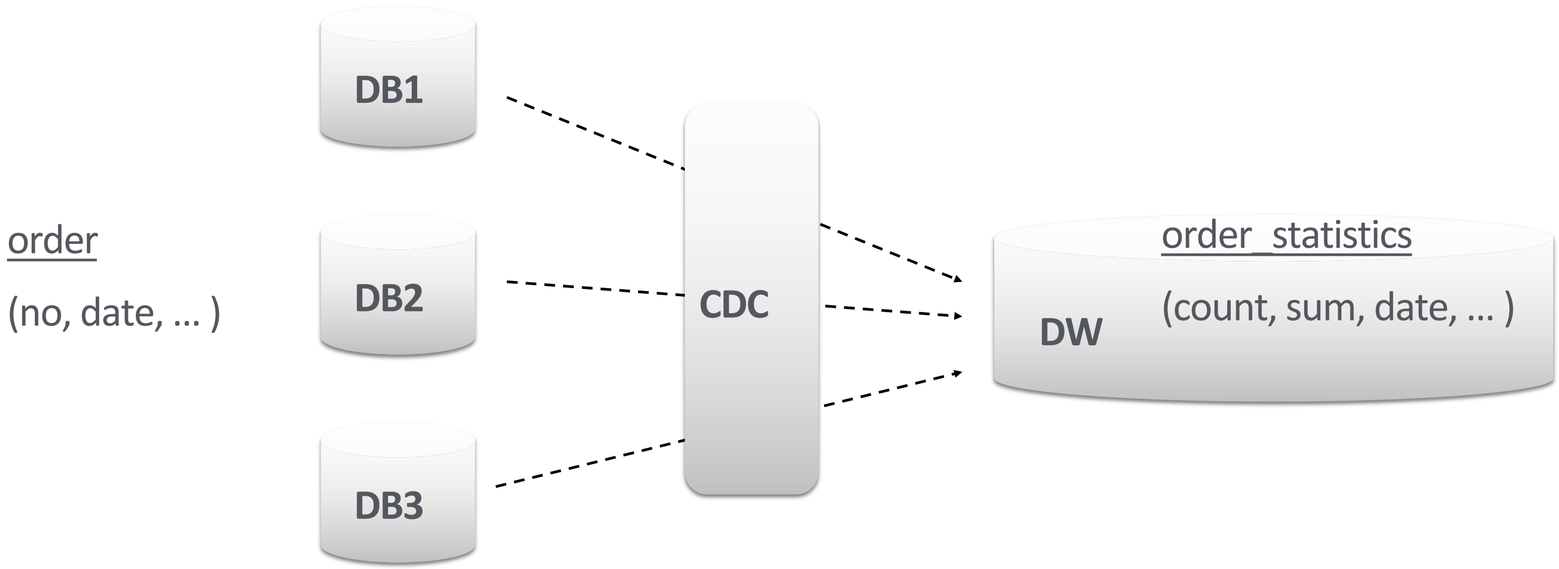
실시간 복제

데이터 정합성 보장

소스 저장소 I/O 및 네트웍 사용 경감

CDC란 무엇인가? CDC로 할 수 있는 것들

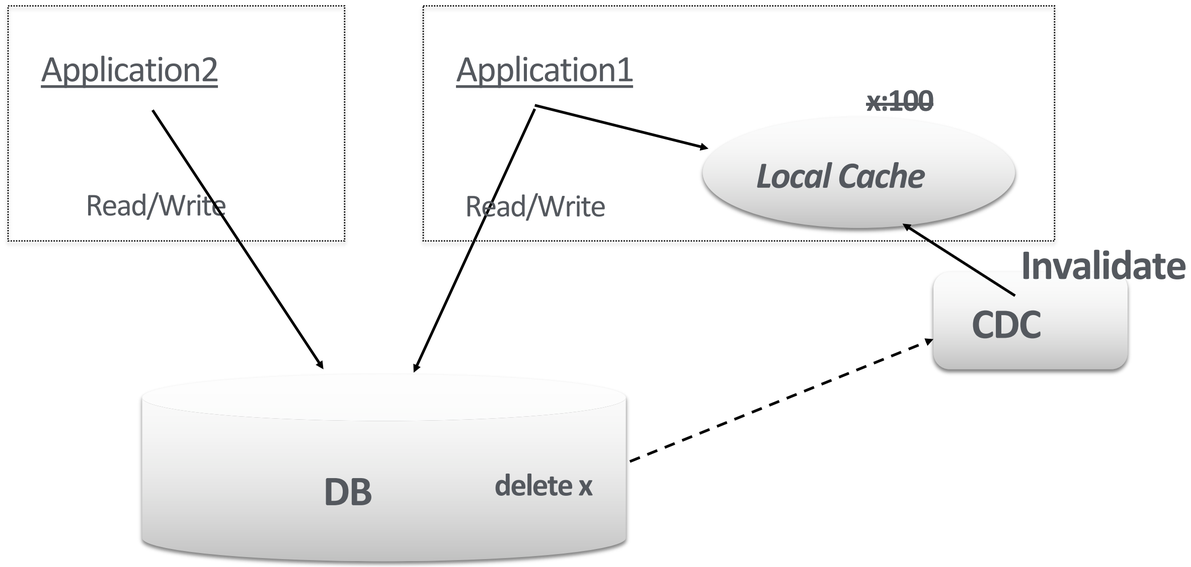
- **Real-time data loading into a Data Warehouse**



다른 스키마 형태로 복제

CDC란 무엇인가? CDC로 할 수 있는 것들

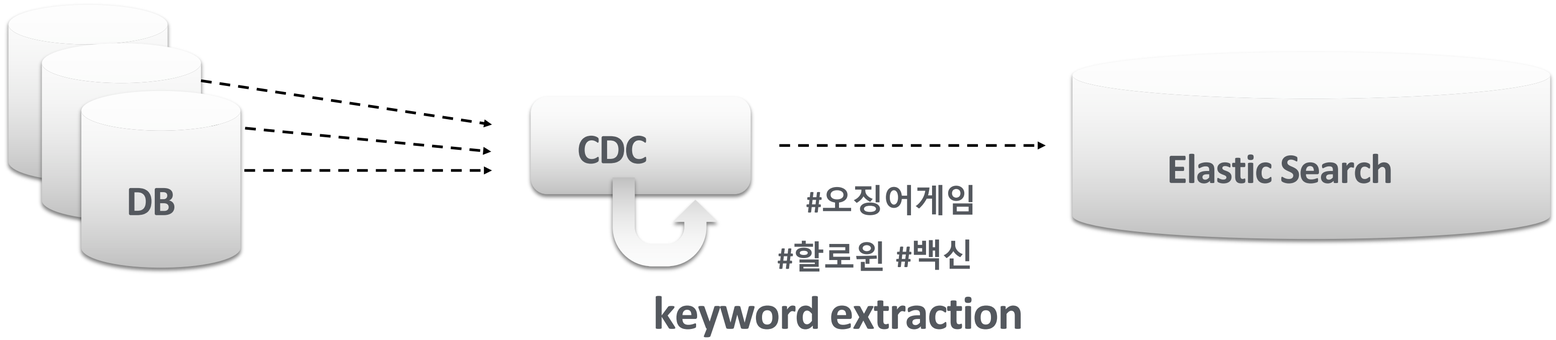
- **Cache invalidation**



원본 DB와 캐시 간의 데이터 일관성 보장

CDC란 무엇인가? CDC로 할 수 있는 것들

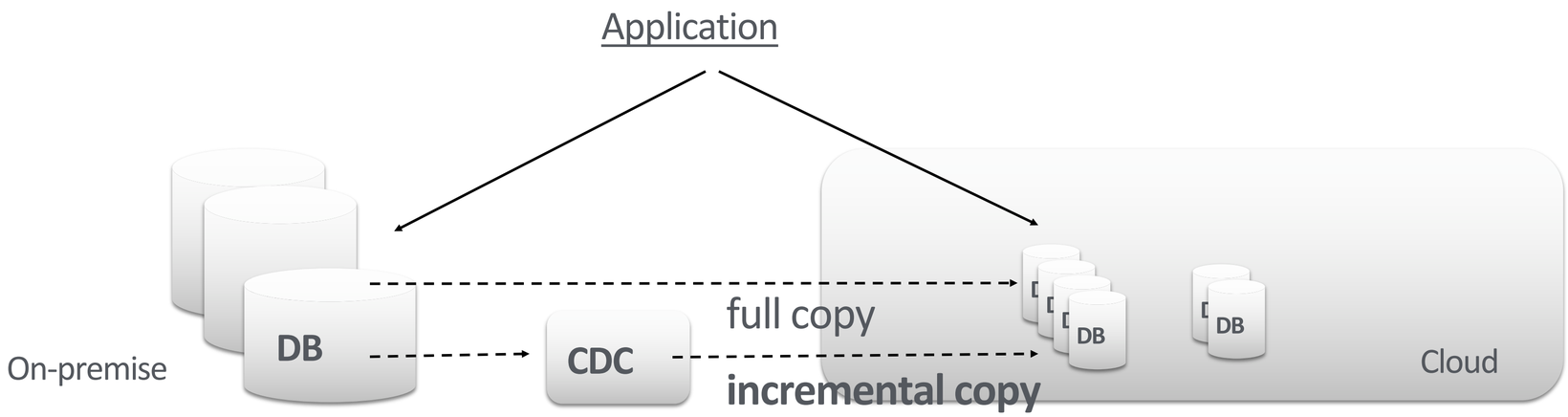
- **Updating a search index such as Elastic Search**



검색 인덱스도 실시간으로 구성

CDC란 무엇인가? CDC로 할 수 있는 것들

- **Synchronization on-premises data to the cloud**



서비스 중단 없이 데이터 이전

Pay 플랫폼에서 CDC의 필요성

Pay 플랫폼에서 CDC의 필요성 서비스 구조의 진화

10배 이상 부하를 받아도 끄떡 없게.. MSA 구조로 전환 해야겠어!

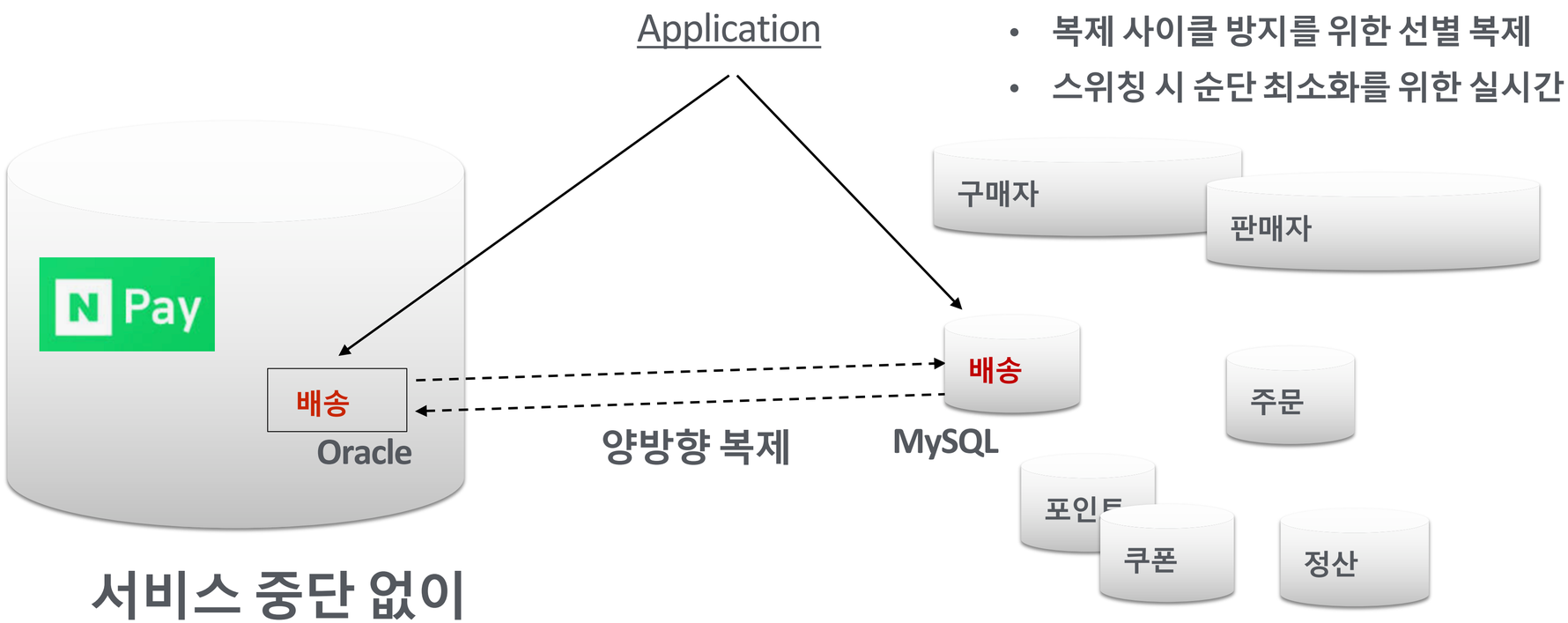
근데 각 도메인 DB는 어떻게 떼어 내지? 도메인 DB간에 데이터 동기화도 해야 돼..

도메인 DB는 scale-out 되게 하고 싶고, 계속 쌓이는 데이터는 비용 효율적으로 저장하고 싶은데?

Pay 플랫폼에서 CDC의 필요성 도메인 DB 떼어 내기

CDC 기술

- 이 기종 간 복제
- 복제 사이클 방지를 위한 선별 복제
- 스위칭 시 순단 최소화를 위한 실시간 복제



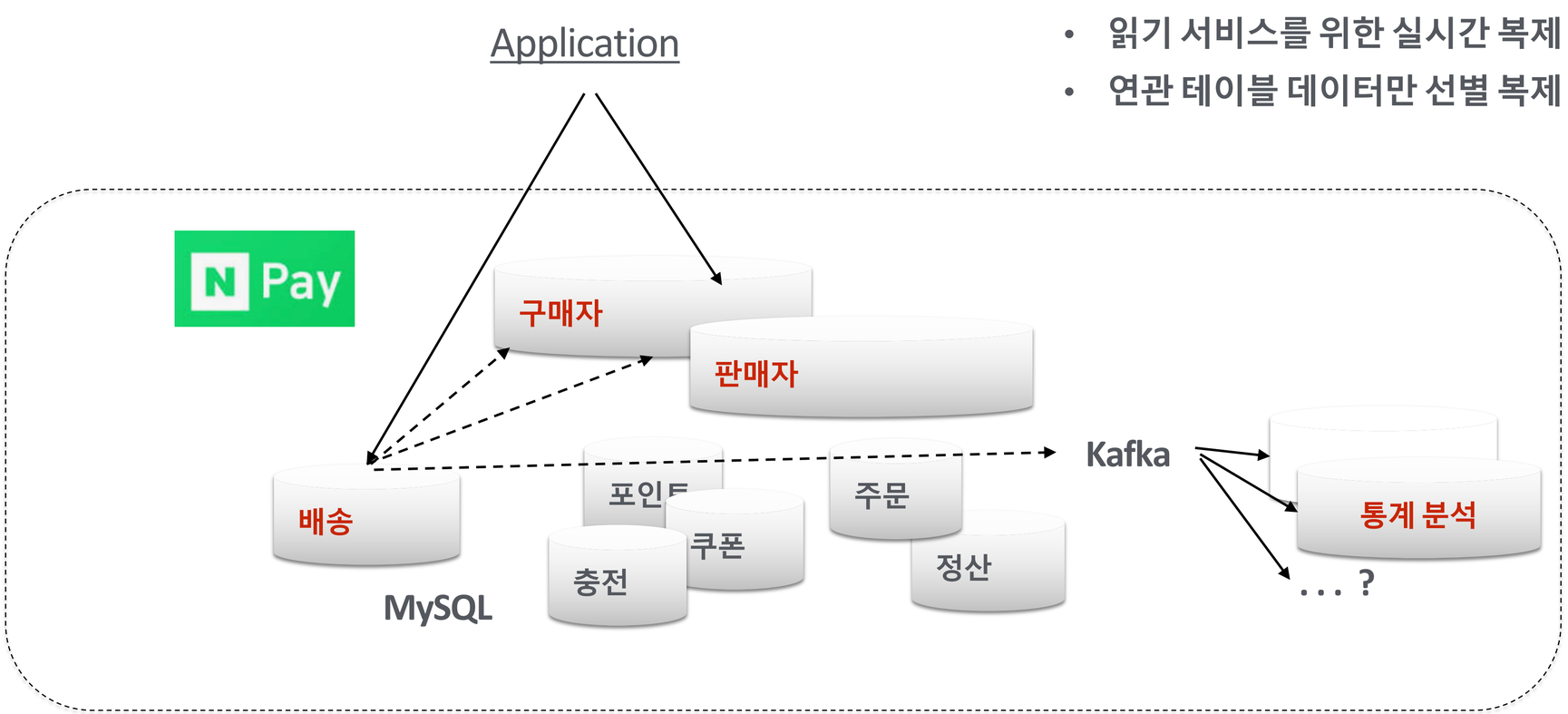
서비스 중단 없이

롤백도 자유 자재로 하려면..

Pay 플랫폼에서 CDC의 필요성 도메인 DB 간 데이터 동기화

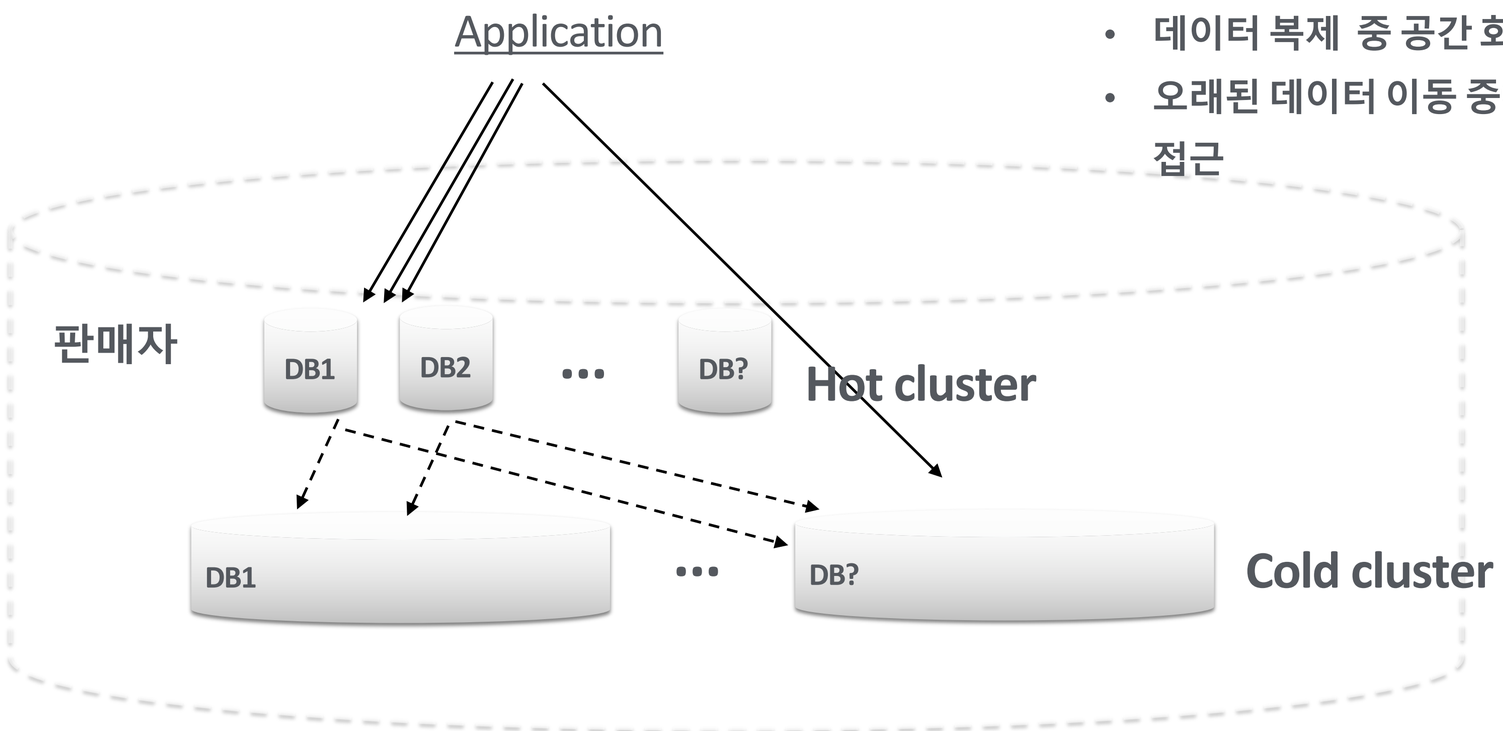
CDC 기술

- 읽기 서비스를 위한 실시간 복제
- 연관 테이블 데이터만 선별 복제



실시간으로 도메인 간 데이터 복제를..

Pay 플랫폼에서 CDC의 필요성 도메인 DB 최적화



CDC 기술

- 데이터 복제 중 공간 회수용 삭제는 필터링
- 오래된 데이터 이동 중에도 끊김 없이 응용 접근

오래된 데이터는 느리지만 싼 저장소에 따로 저장

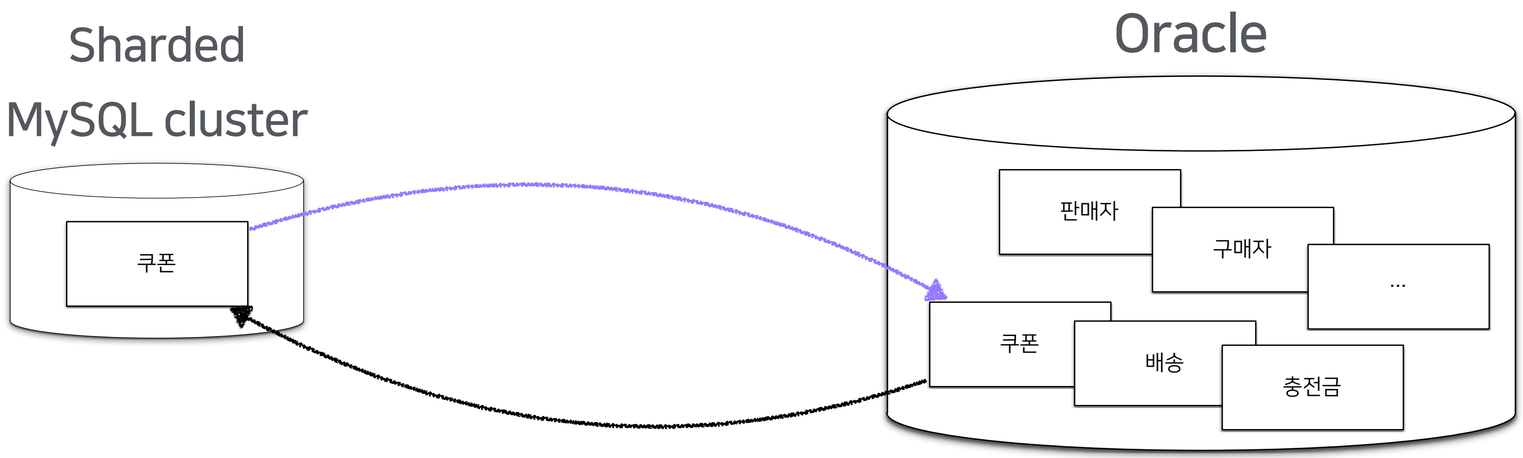
오래 되어도 읽고 쓸 수 있어야 하는데..

DEVIEW 2021

Sharded MySQL Cluster에서의 Change Data Capture

Heterogeneous Replication

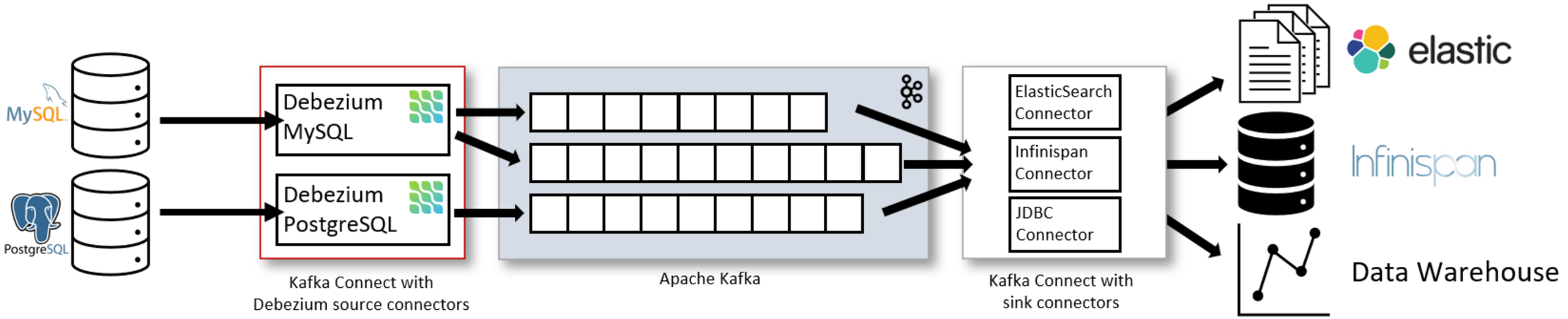
From Sharded MySQL Cluster



Heterogeneous Replication

DEVIEW 2021

Open Source Solution: Debezium



https://debezium.io/documentation/reference/1.6/architecture.html

Heterogeneous Replication

DEVIEW 2021

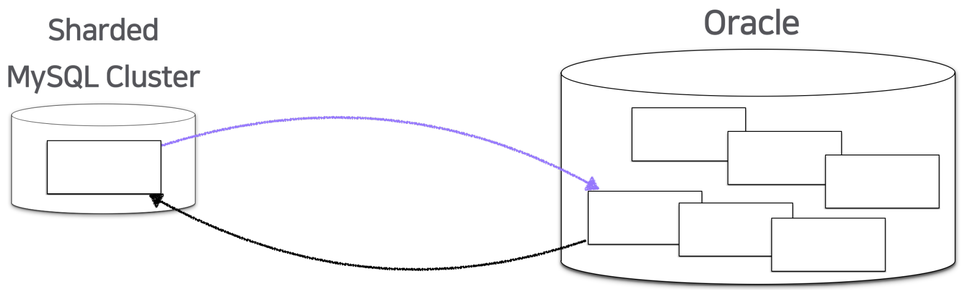
Open Source Solution: Debezium

- Log-Based Change Data Capture
- Snapshot
- Filters
- Masking
- Monitoring
- Single Message Transformations

https://debezium.io/documentation/reference/1.6/features.html

Heterogeneous Replication

From Sharded MySQL Cluster

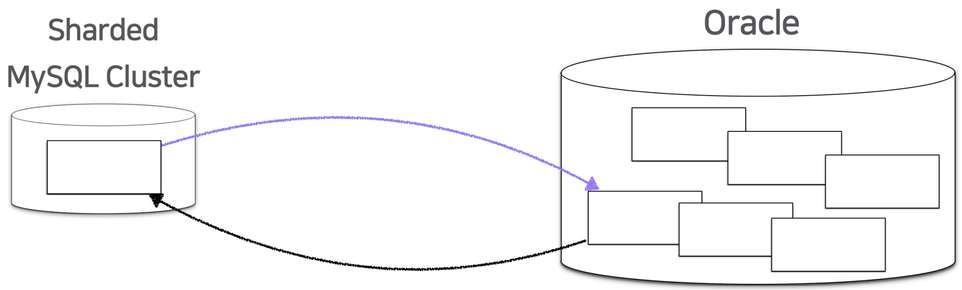


- Sharding을 위한 Logical View + Meta Data 처리
- Shard Rebalance로 인한 내부 부하 및 순서 보장

Heterogeneous Replication

DEVIEW 2021

From Sharded MySQL Cluster + Bidirectional Replication

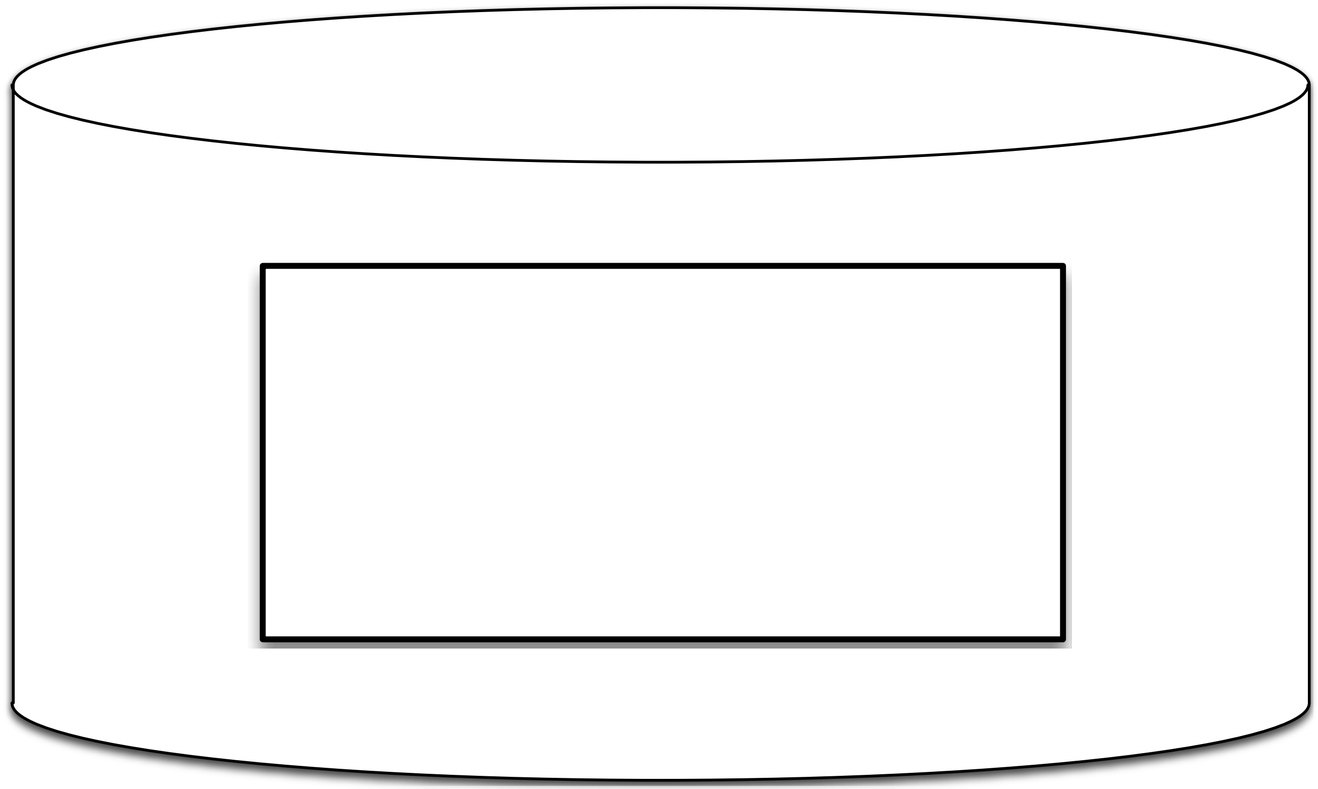


- Sharding을 위한 Logical View + Meta Data 처리
- Shard Rebalance로 인한 내부 부하 및 순서 보장
- 즉각적인 Rollback을 위한 성능 최적화
- 안정적인 복제를 위한 고가용성

CDC in Sharded MySQL Cluster

DEVIEW 2021

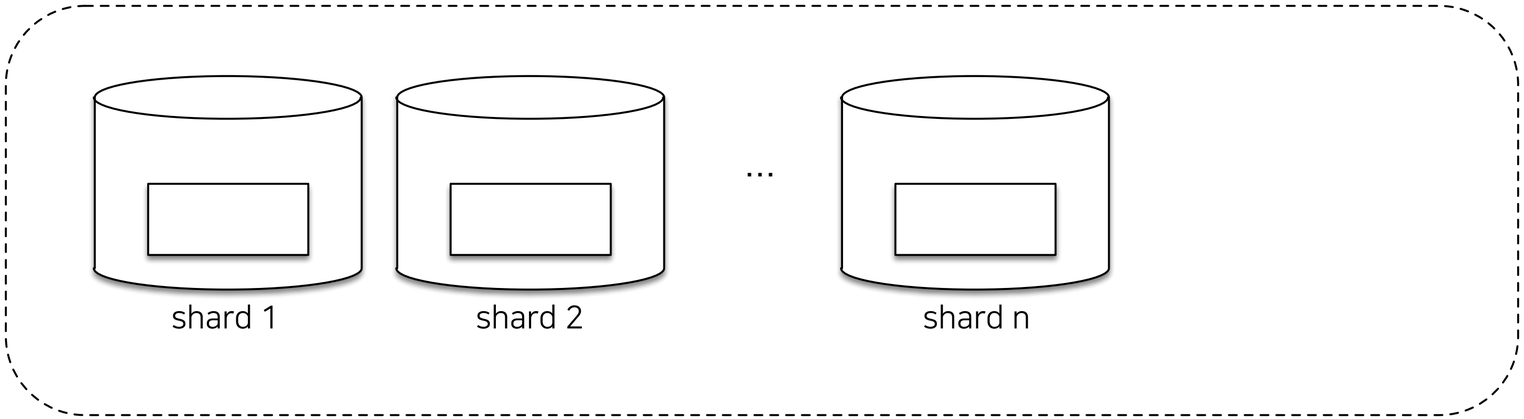
Logical View + Meta Data



CDC in Sharded MySQL Cluster

DEVIEW 2021

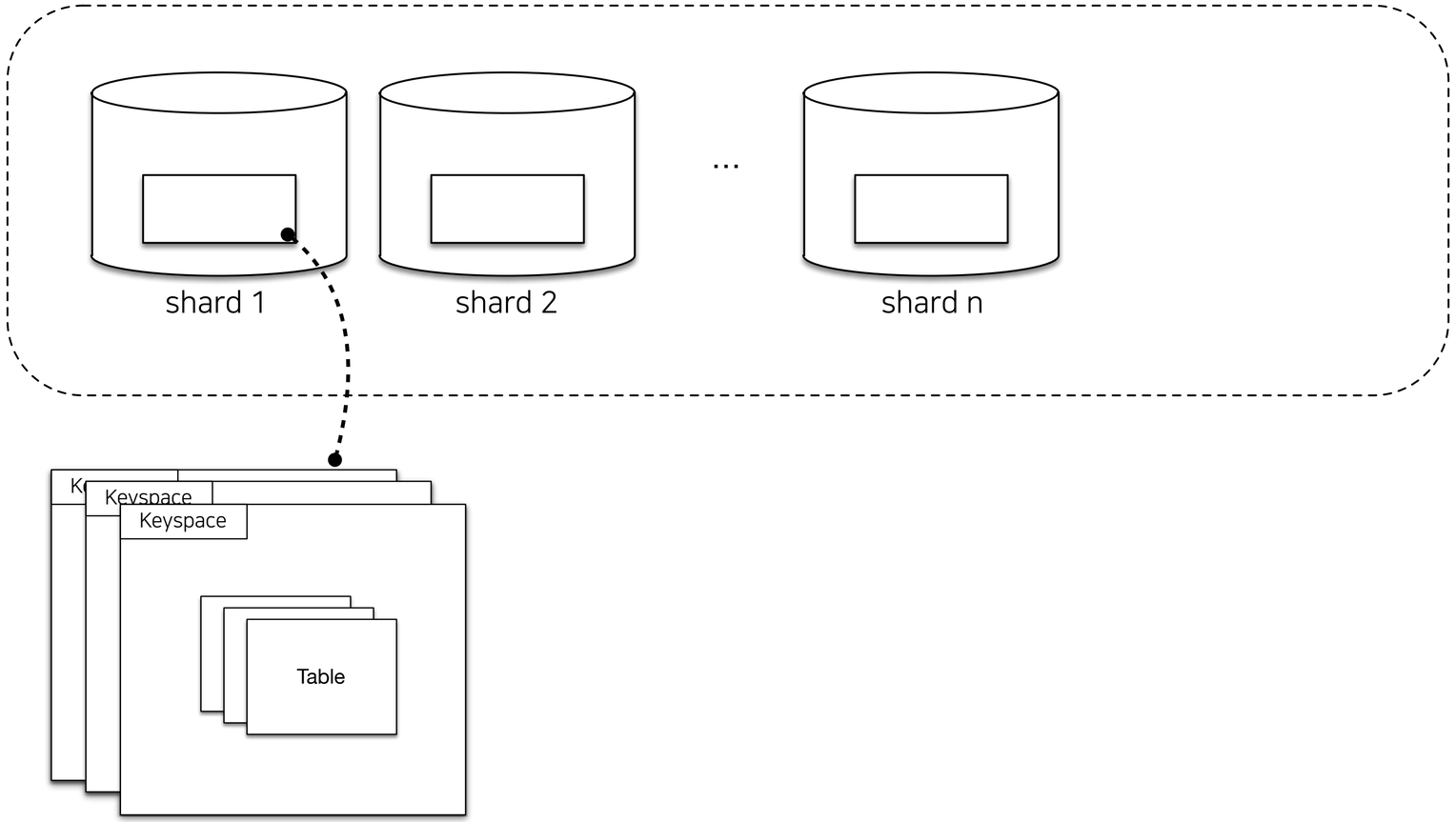
Logical View + Meta Data



CDC in Sharded MySQL Cluster

DEVIEW 2021

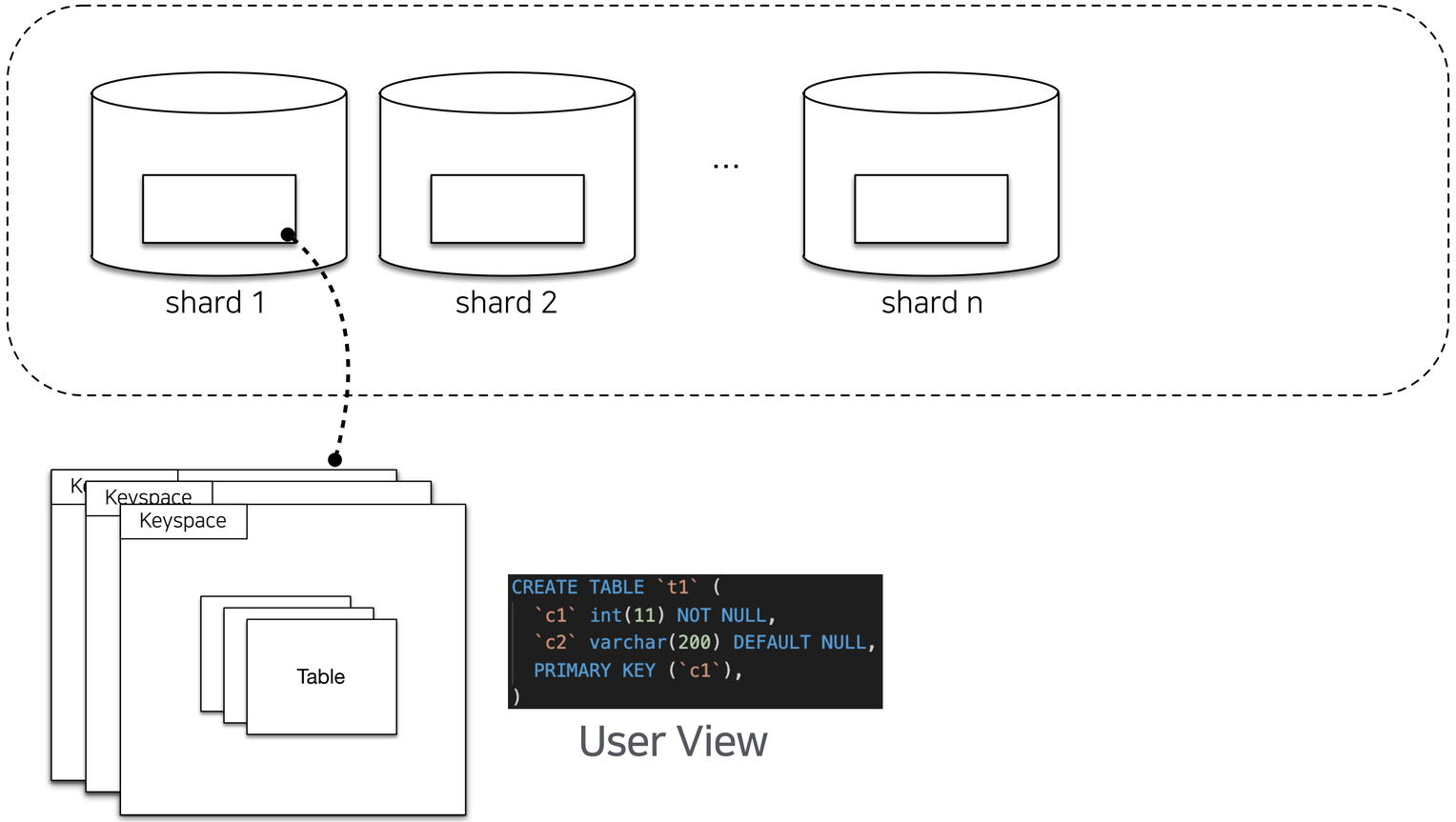
Logical View + Meta Data



CDC in Sharded MySQL Cluster

DEVIEW 2021

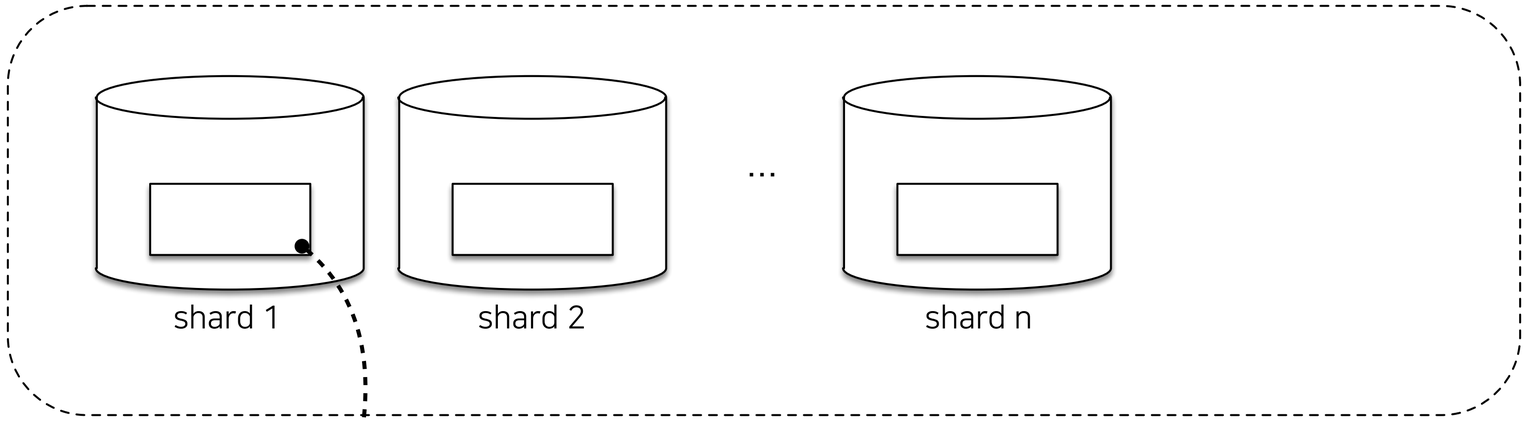
Logical View + Meta Data

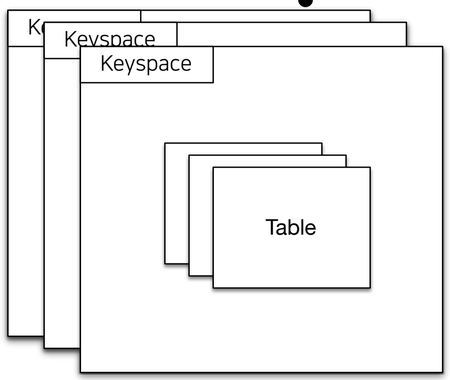


CDC in Sharded MySQL Cluster

DEVIEW 2021

Logical View + Meta Data






```
CREATE TABLE `t1` (
  `c1` int(11) NOT NULL,
  `c2` varchar(200) DEFAULT NULL,
  PRIMARY KEY (`c1`),
)
```

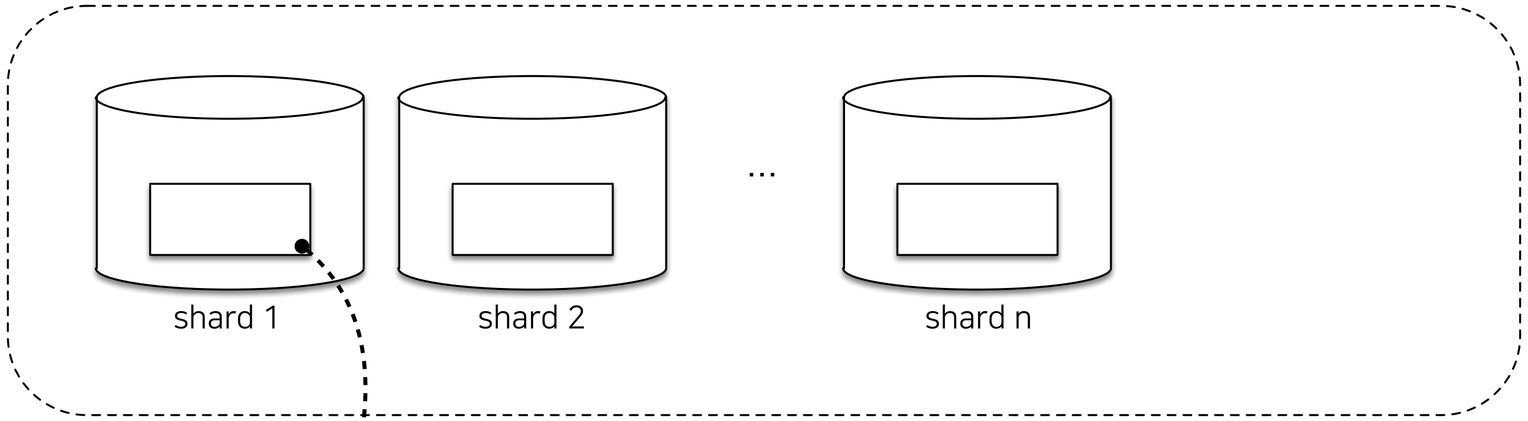
User View

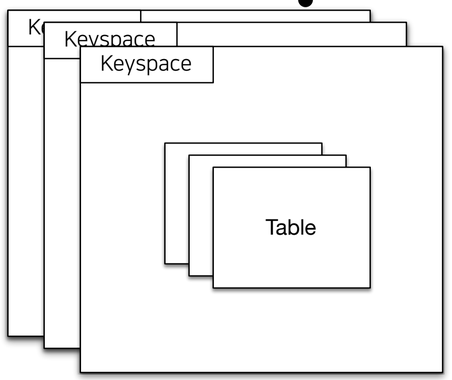
```
CREATE TABLE `xx_ks_t1` (
  `xx_seq` bigint(20) NOT NULL AUTO_INCREMENT,
  `xx_shard_key` varbinary(255) NOT NULL,
  `xx_shard_cgmapid` int(11) DEFAULT NULL,
  `xx_sys_ts` bigint(20) DEFAULT NULL,
  `xx_del_ts` bigint(20) DEFAULT NULL,
  `x_c1` int(11) NOT NULL,
  `x_c2` varchar(200) DEFAULT NULL,
  PRIMARY KEY (`ns_seq`),
  UNIQUE KEY `x_pkidx` (`ns_ck`,`u_c1`),
  ...
)
```

Raw

CDC in Sharded MySQL Cluster

Logical View + Meta Data






```
CREATE TABLE `t1` (
  `c1` int(11) NOT NULL,
  `c2` varchar(200) DEFAULT NULL,
  PRIMARY KEY (`c1`),
)
```

User View

```
CREATE TABLE `xx_ks_t1` (
  `xx_seq` bigint(20) NOT NULL AUTO_INCREMENT,
  `xx_shard_key` varbinary(255) NOT NULL,
  `xx_shard_cgmapid` int(11) DEFAULT NULL,
  `xx_sys_ts` bigint(20) DEFAULT NULL,
  `xx_del_ts` bigint(20) DEFAULT NULL,
  `x_c1` int(11) NOT NULL,
  `x_c2` varchar(200) DEFAULT NULL,
  PRIMARY KEY (`ns_seq`),
  UNIQUE KEY `x_pkidx` (`ns_ck`,`u_c1`),
  ...
)
```

Raw

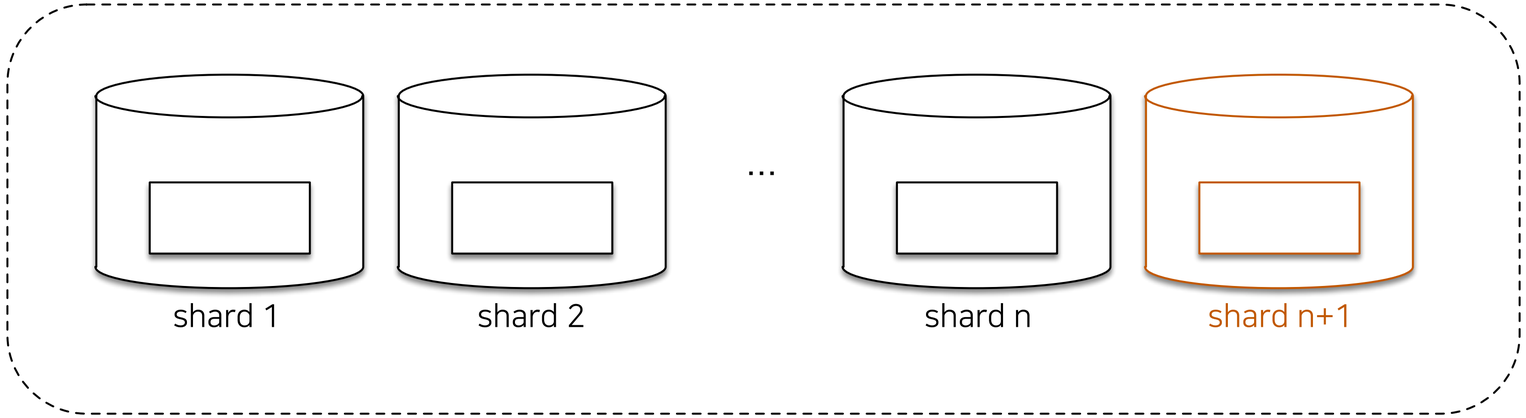
CDC target은
User View의 데이터
변경 이벤트를 기대

```
{
  "c1": 10,
  "c2": "data"
}
```

CDC in Sharded MySQL Cluster

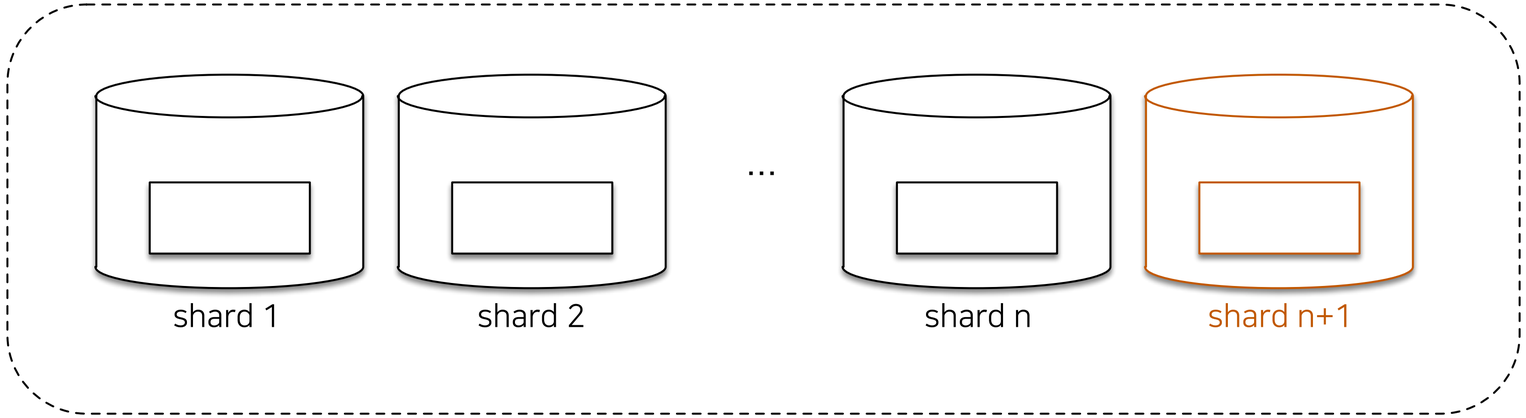
DEVIEW 2021

Rebalance



CDC in Sharded MySQL Cluster

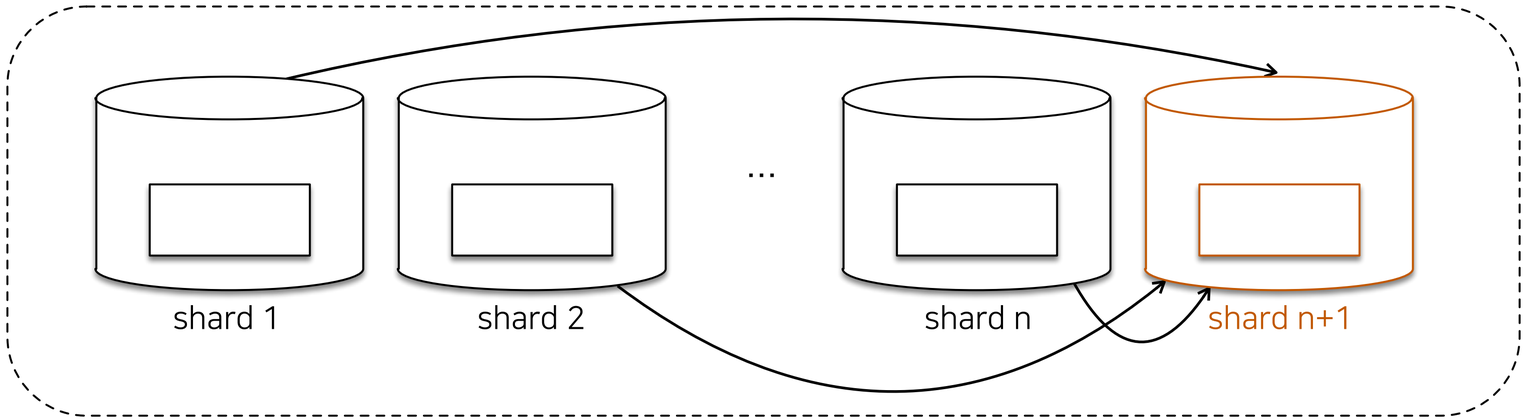
Rebalance로 발생하는 내부 부하 Filtering



CDC in Sharded MySQL Cluster

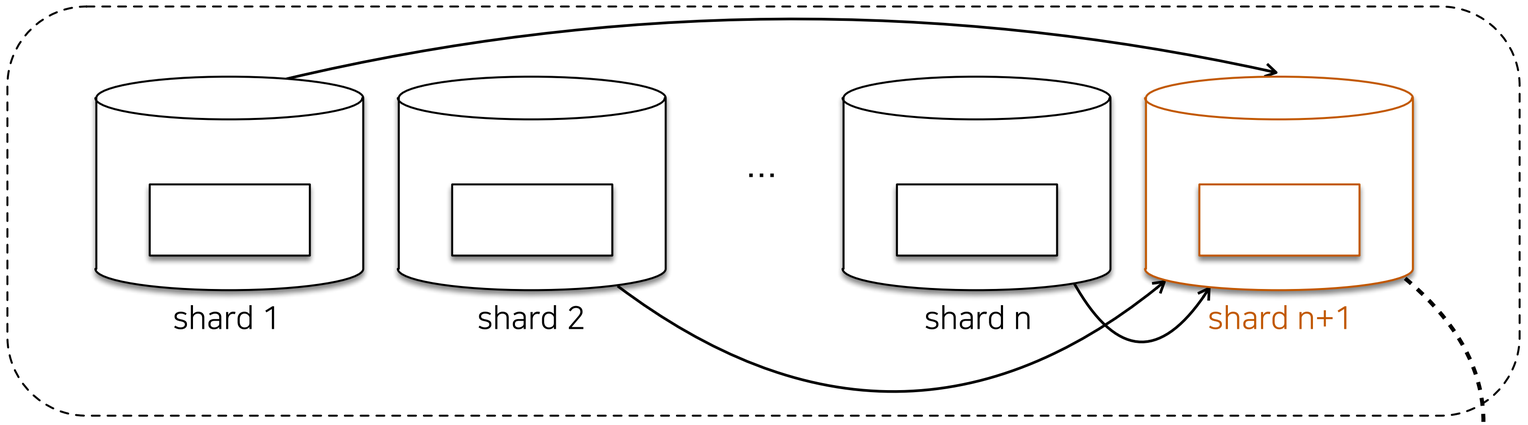
DEVIEW 2021

Rebalance로 발생하는 내부 부하 Filtering

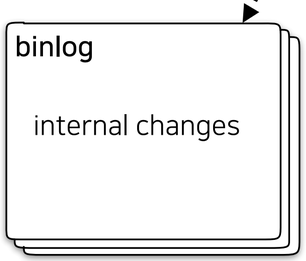


CDC in Sharded MySQL Cluster

Rebalance로 발생하는 내부 부하 Filtering



row의 변경 내용 만으로는 filtering
대상인지 구분할 수 없는 문제

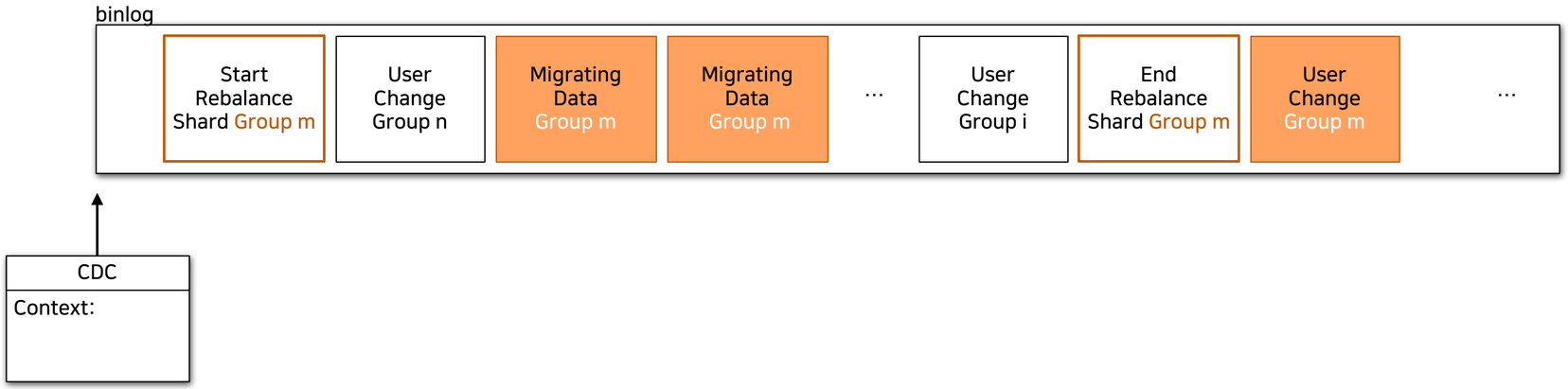


CDC in Sharded MySQL Cluster

DEVIEW 2021

Rebalance로 발생하는 내부 부하 Filtering

binlog 분석을 통한 Context-Aware Filtering

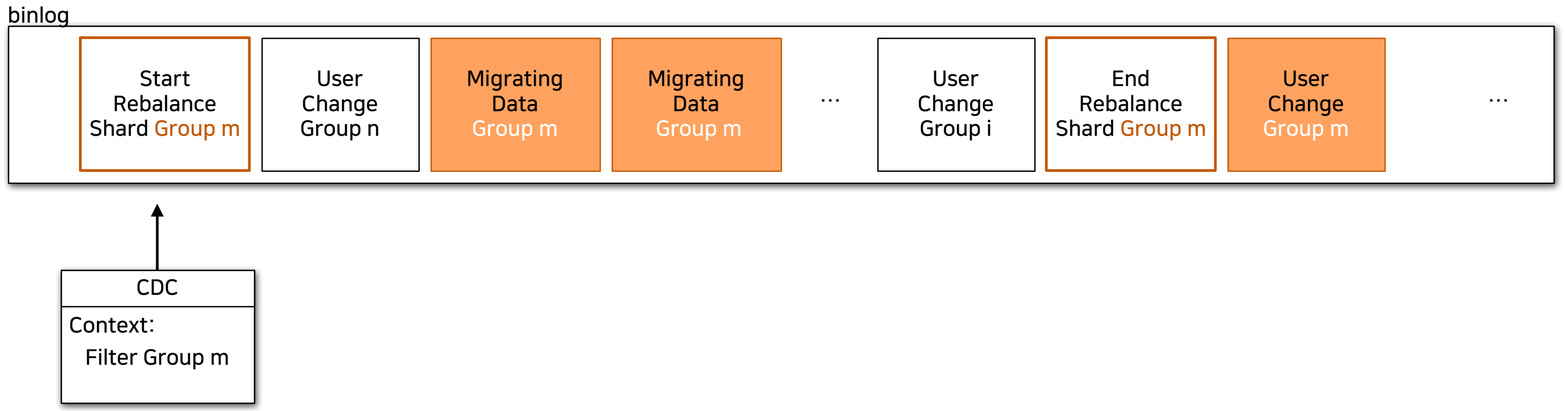


CDC in Sharded MySQL Cluster

DEVIEW 2021

Rebalance로 발생하는 내부 부하 Filtering

binlog 분석을 통한 Context-Aware Filtering

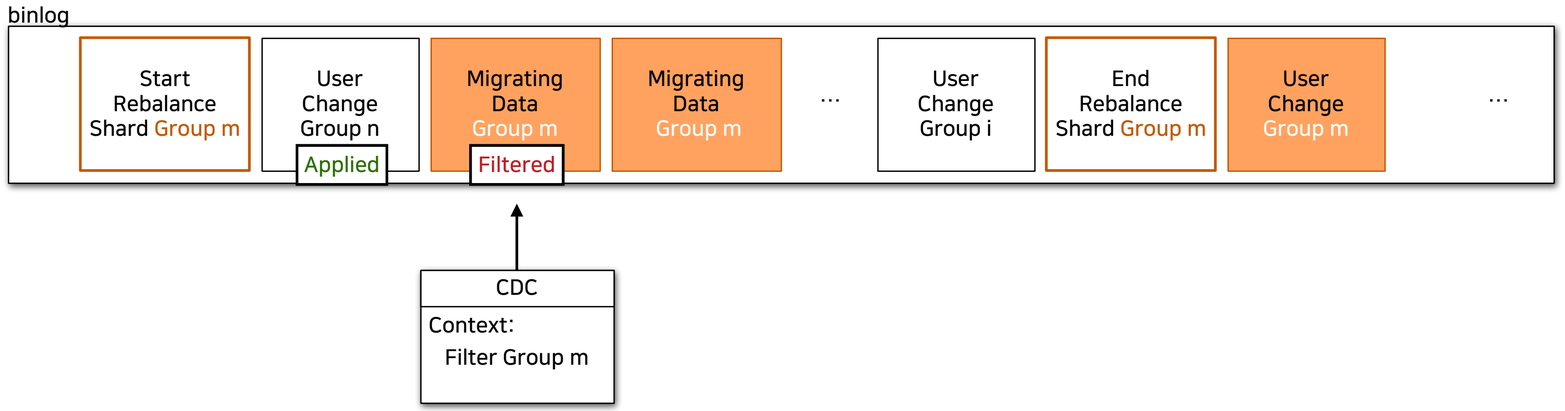


CDC in Sharded MySQL Cluster

DEVIEW 2021

Rebalance로 발생하는 내부 부하 Filtering

binlog 분석을 통한 Context-Aware Filtering

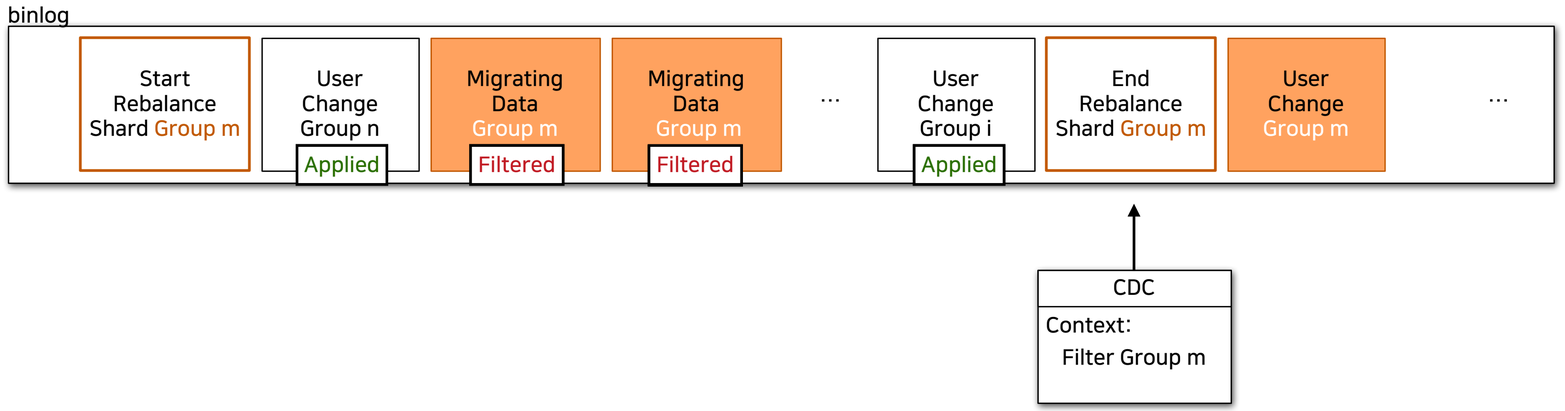


CDC in Sharded MySQL Cluster

DEVIEW 2021

Rebalance로 발생하는 내부 부하 Filtering

binlog 분석을 통한 Context-Aware Filtering

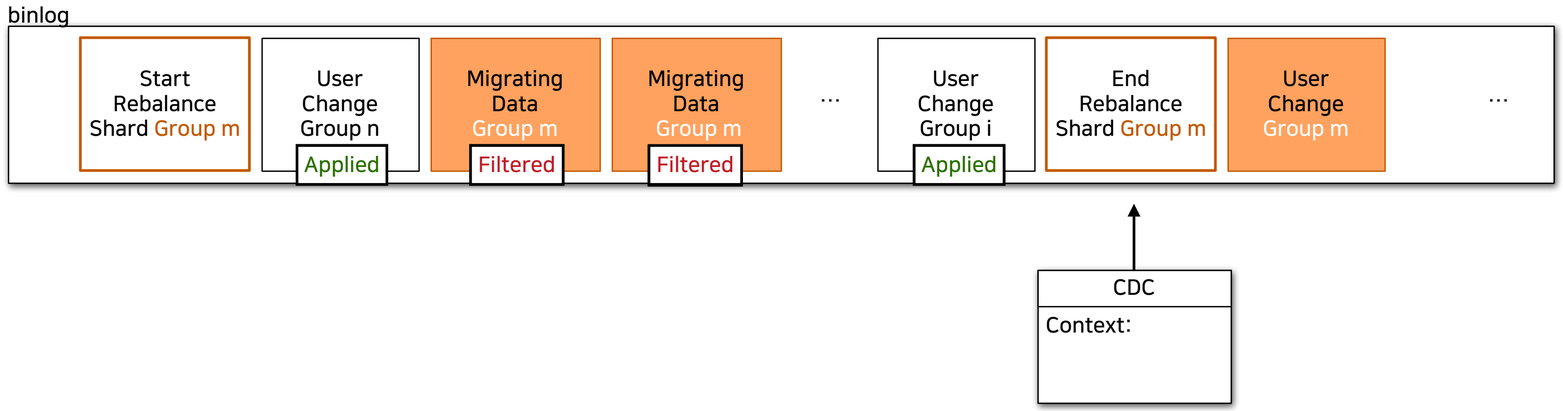


CDC in Sharded MySQL Cluster

DEVIEW 2021

Rebalance로 발생하는 내부 부하 Filtering

binlog 분석을 통한 Context-Aware Filtering

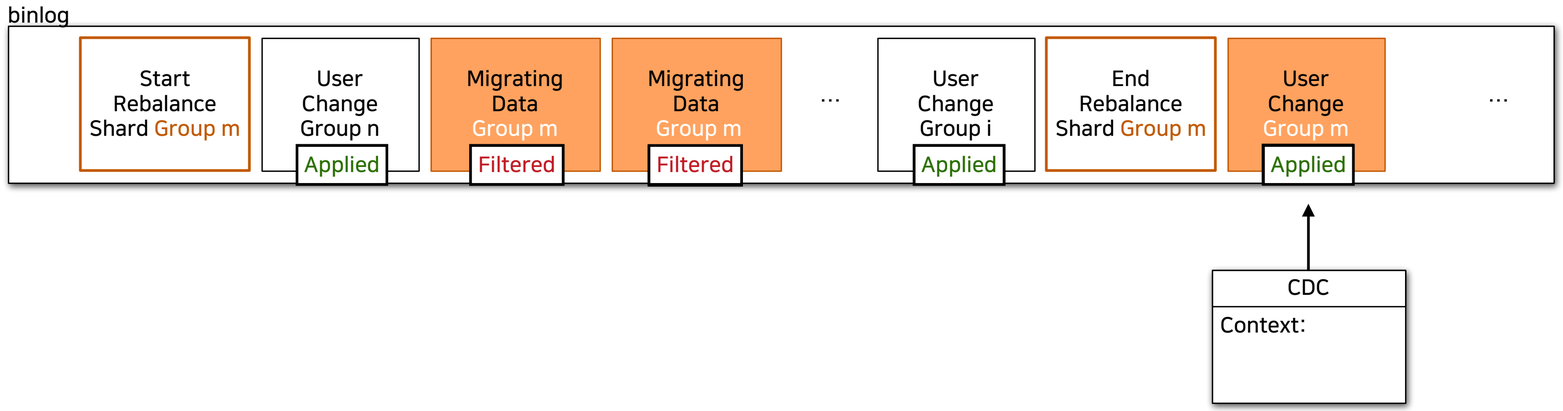


CDC in Sharded MySQL Cluster

DEVIEW 2021

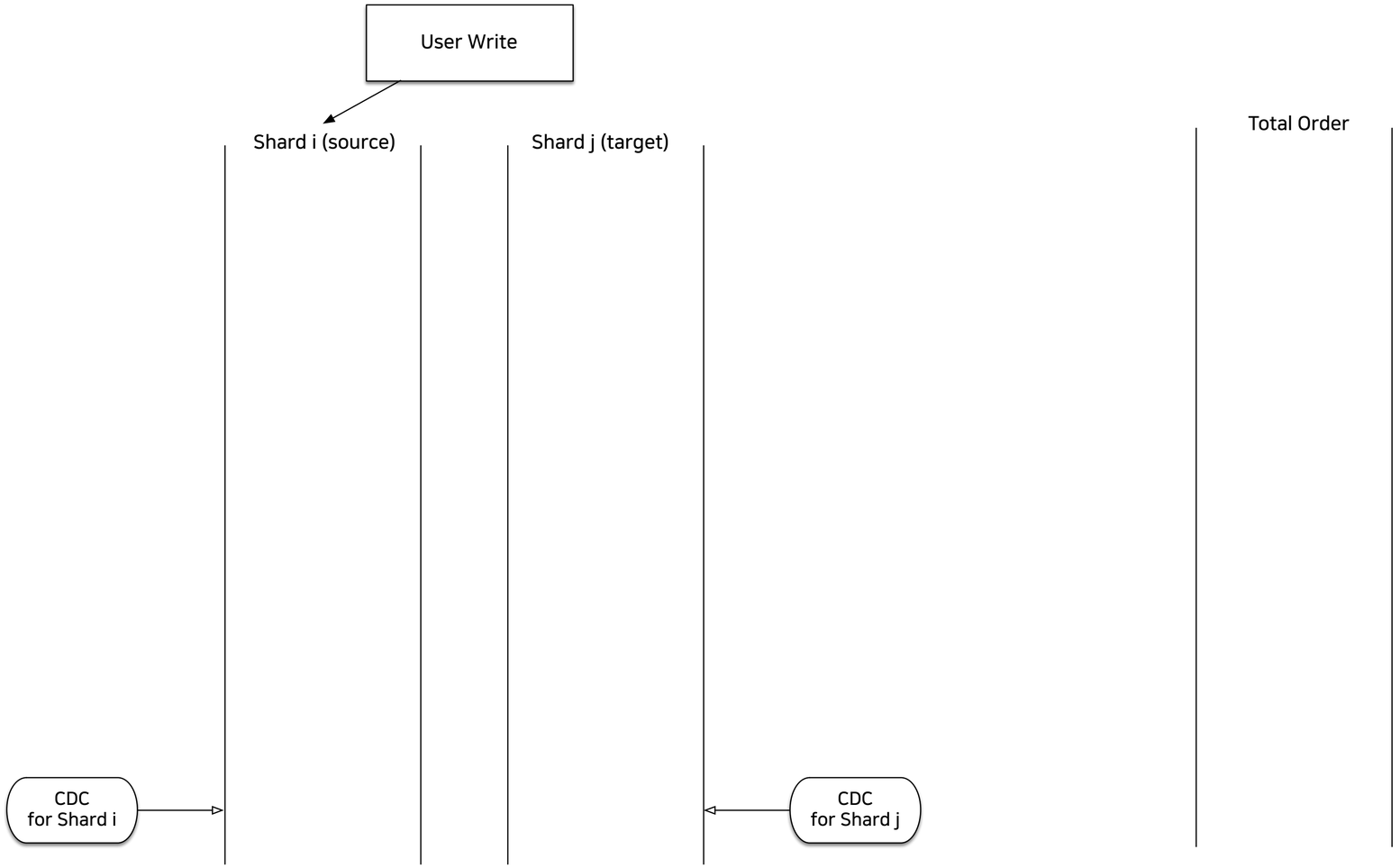
Rebalance로 발생하는 내부 부하 Filtering

binlog 분석을 통한 Context-Aware Filtering



CDC in Sharded MySQL Cluster

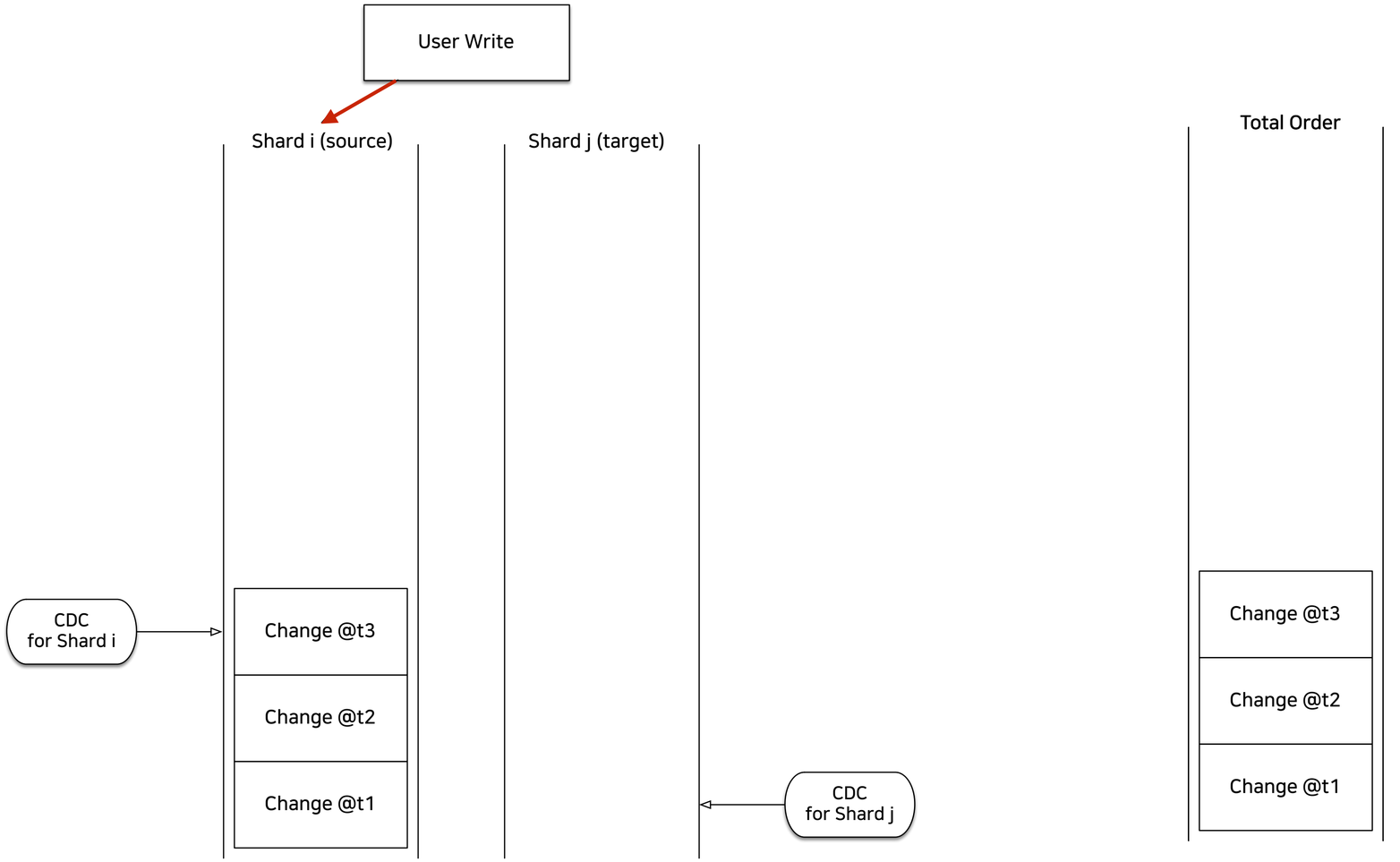
Rebalance 직후 데이터 변경 순서의 역전



각 Shard의 binlog를 독립적으로 처리하는 경우

CDC in Sharded MySQL Cluster

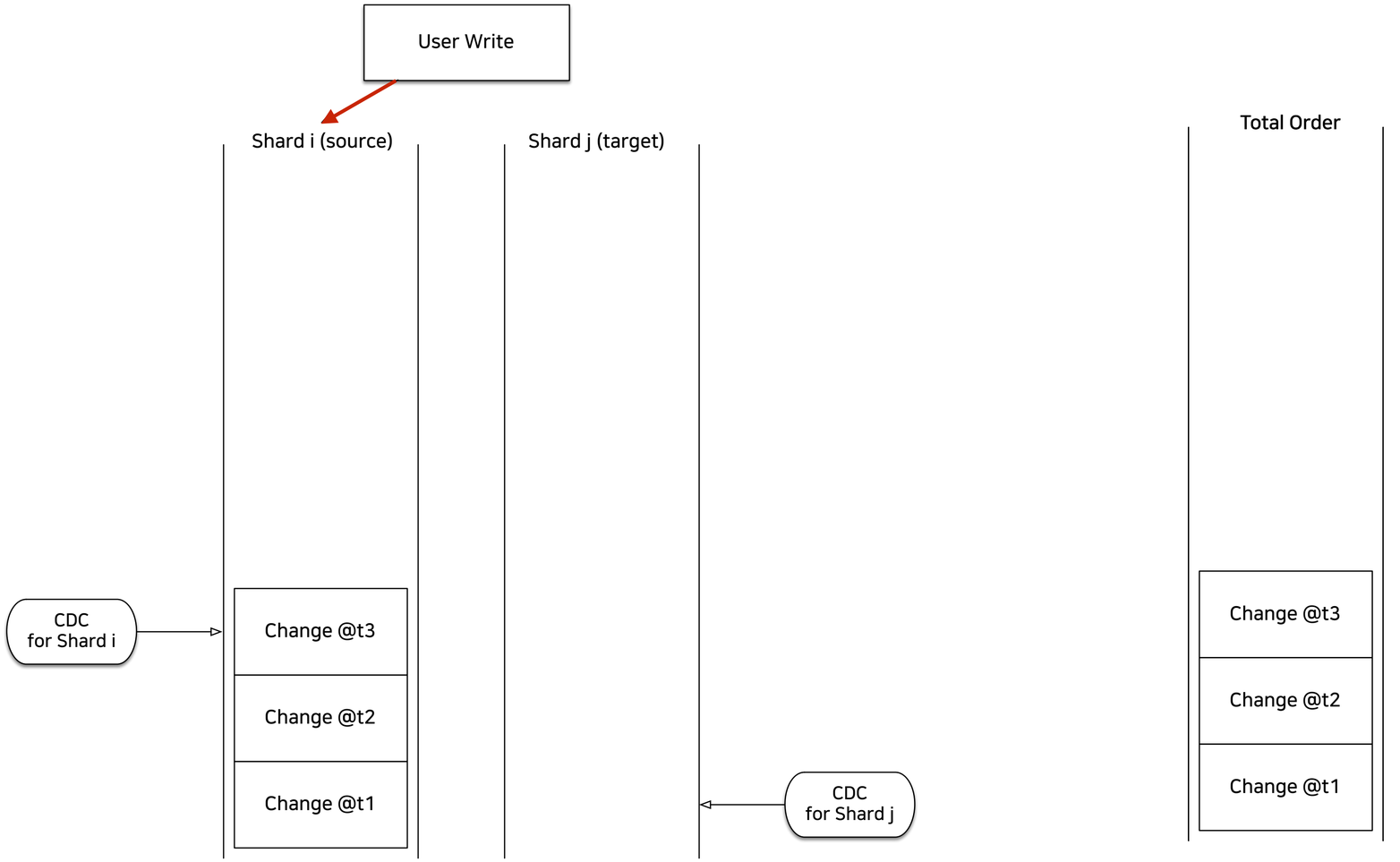
Rebalance 직후 데이터 변경 순서의 역전



각 Shard의 binlog를 독립적으로 처리하는 경우

CDC in Sharded MySQL Cluster

Rebalance 직후 데이터 변경 순서의 역전

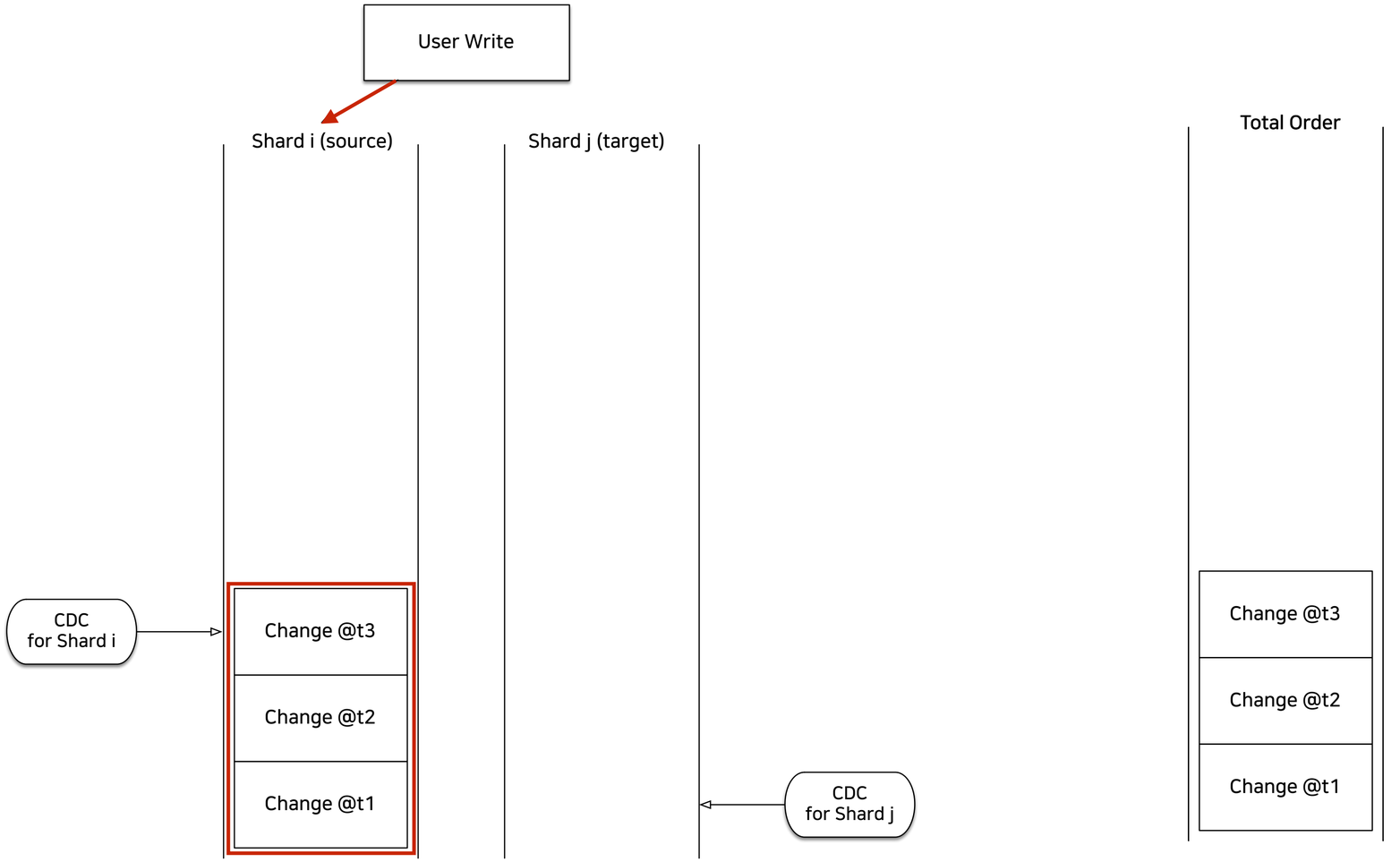


각 Shard의 binlog를 독립적으로 처리하는 경우

CDC in Sharded MySQL Cluster

DEVIEW 2021

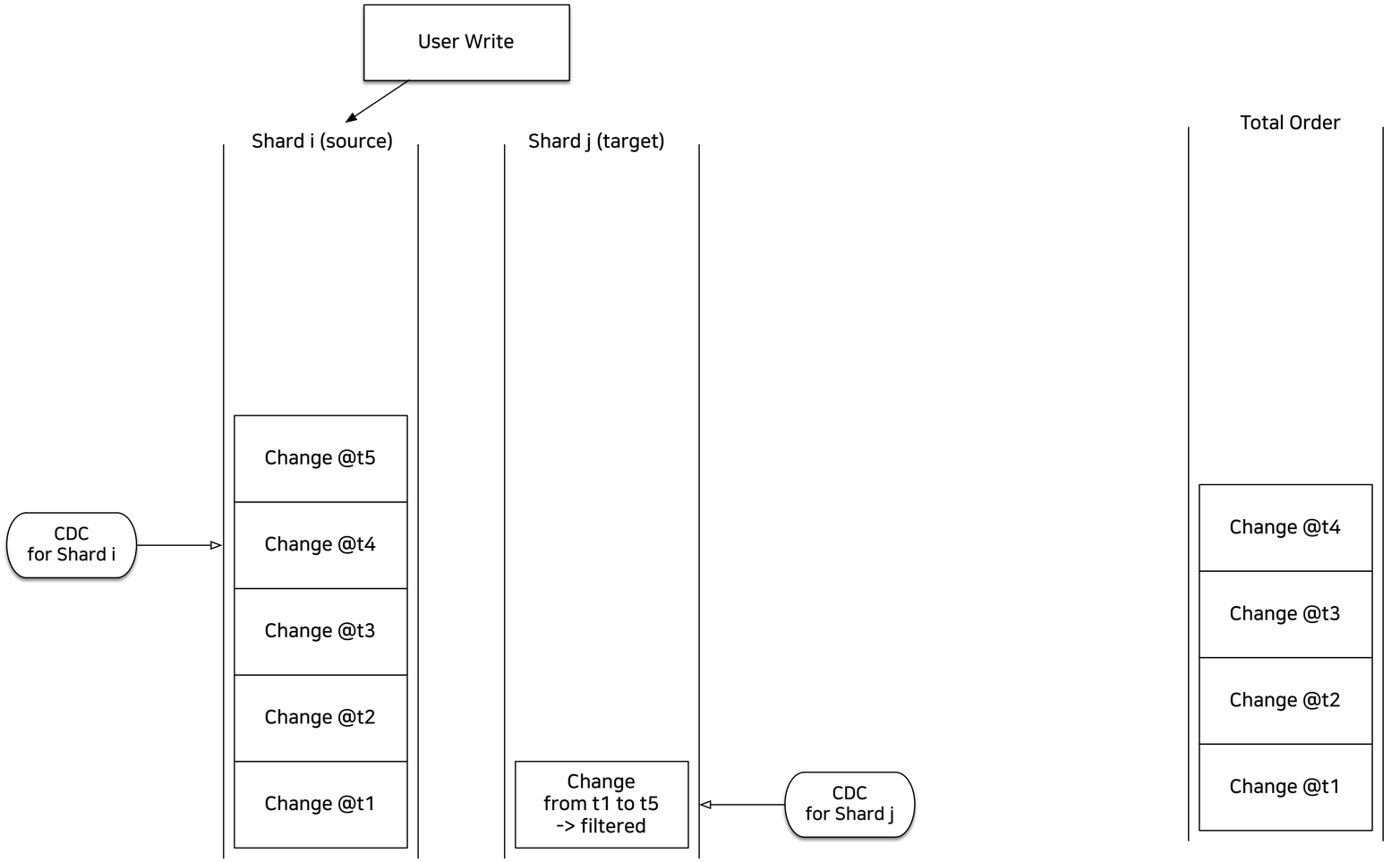
Rebalance 직후 데이터 변경 순서의 역전



각 Shard의 binlog를 독립적으로 처리하는 경우

CDC in Sharded MySQL Cluster

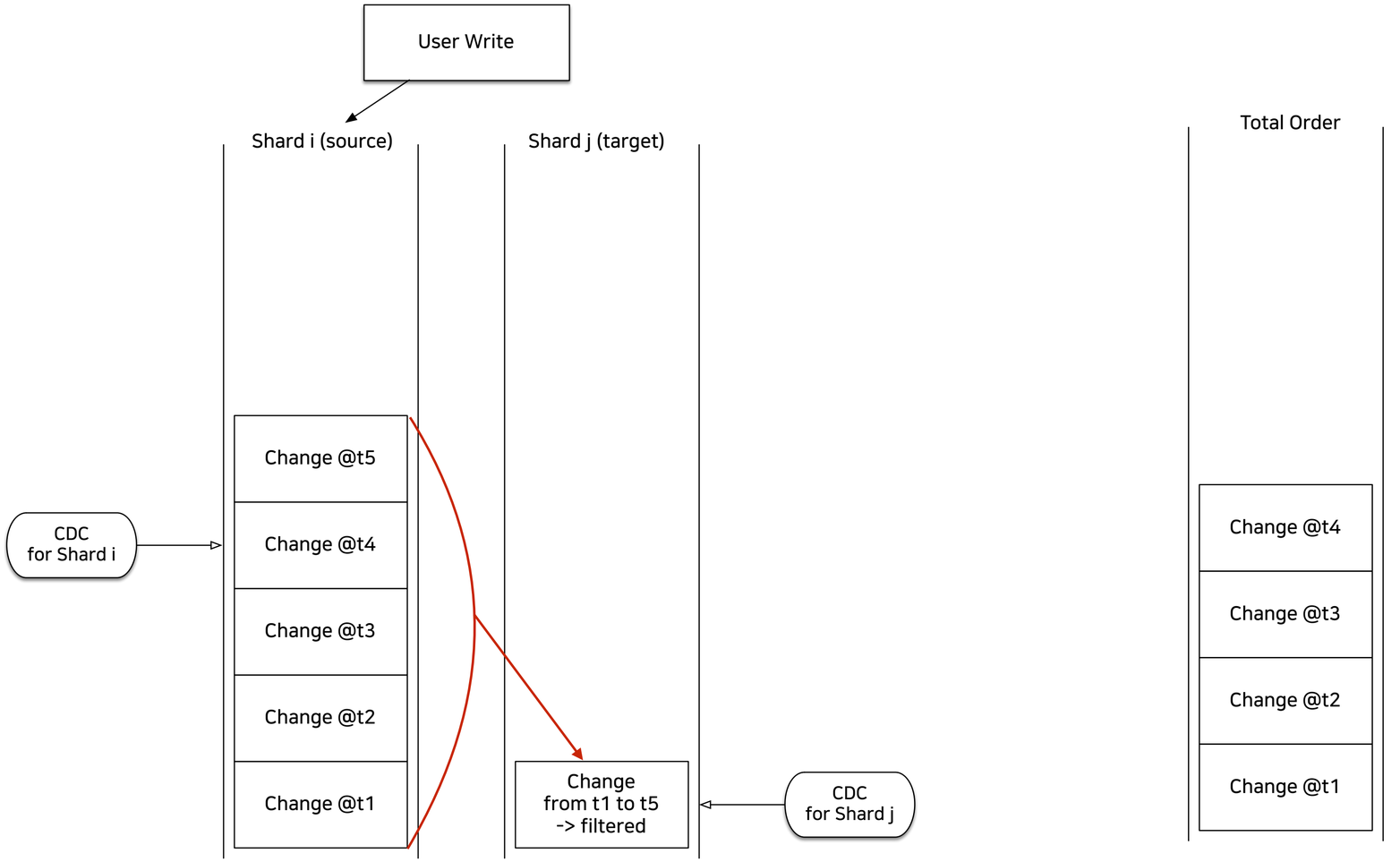
Rebalance 직후 데이터 변경 순서의 역전



각 Shard의 binlog를 독립적으로 처리하는 경우

CDC in Sharded MySQL Cluster

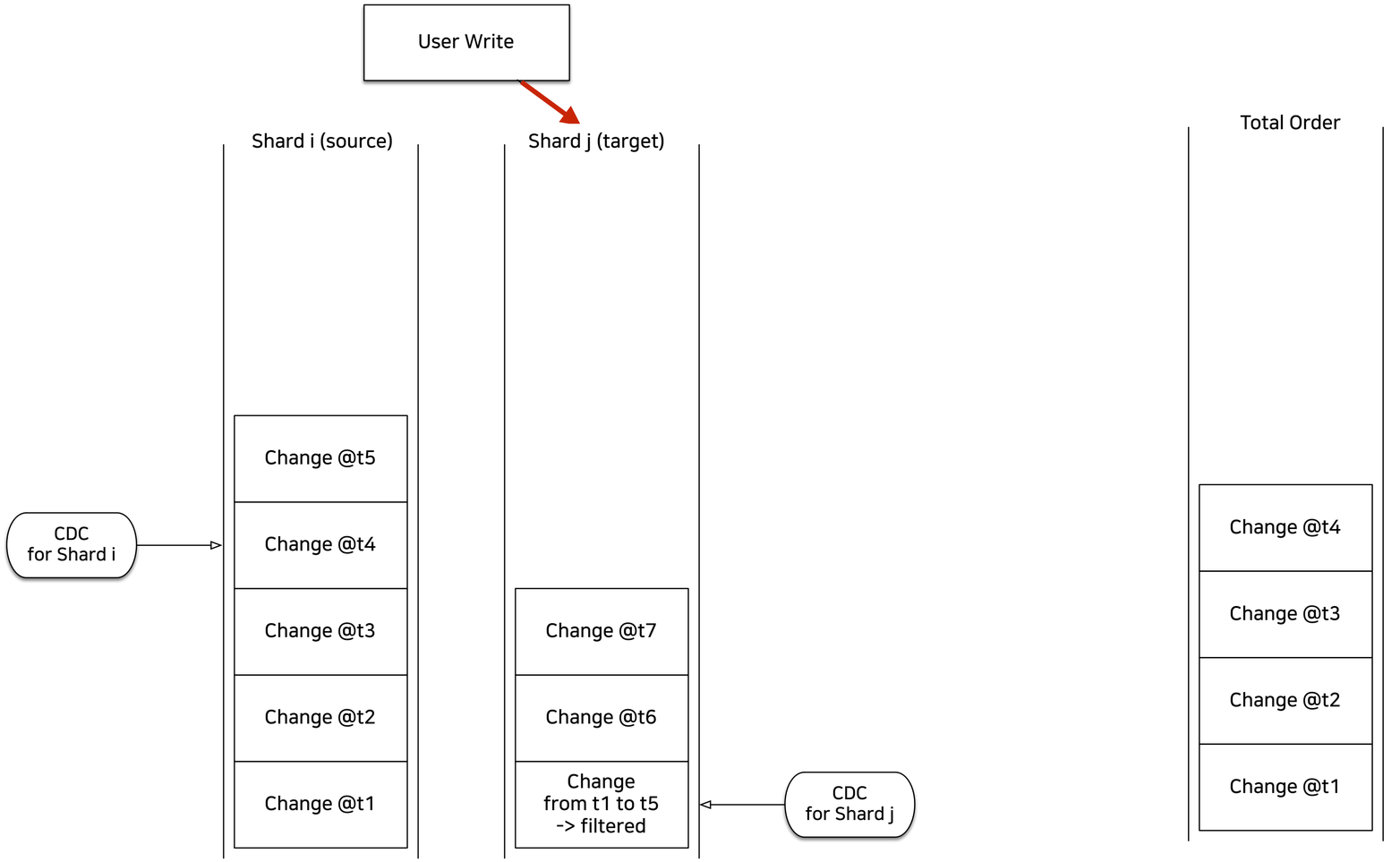
Rebalance 직후 데이터 변경 순서의 역전



각 Shard의 binlog를 독립적으로 처리하는 경우

CDC in Sharded MySQL Cluster

Rebalance 직후 데이터 변경 순서의 역전

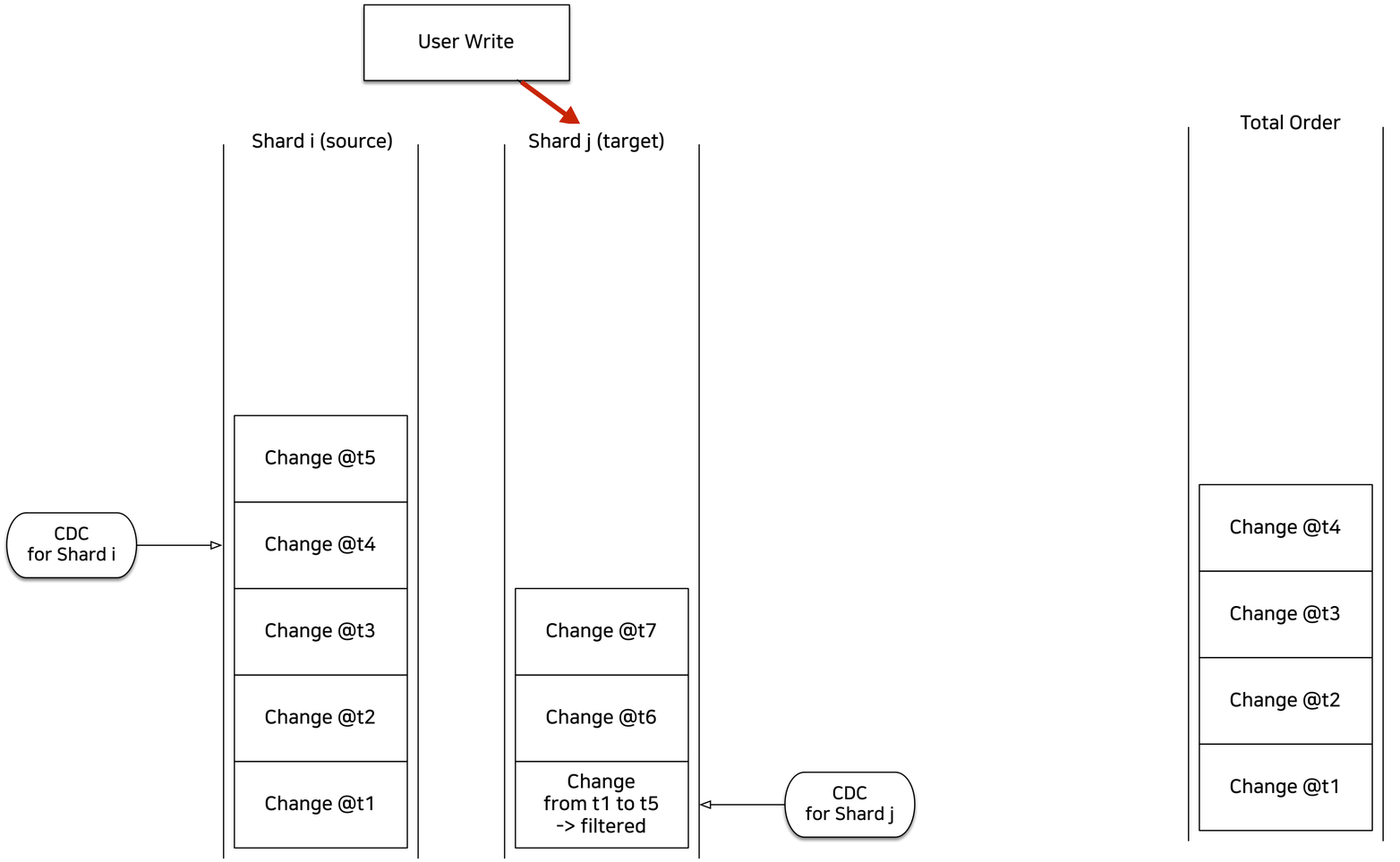


Rebalance를 위한 migration
종료 후 Shard j 를 위한 CDC가
Shard i 를 선행할 경우
Total Ordering에 문제가 발생

각 Shard의 binlog를 독립적으로 처리하는 경우

CDC in Sharded MySQL Cluster

Rebalance 직후 데이터 변경 순서의 역전

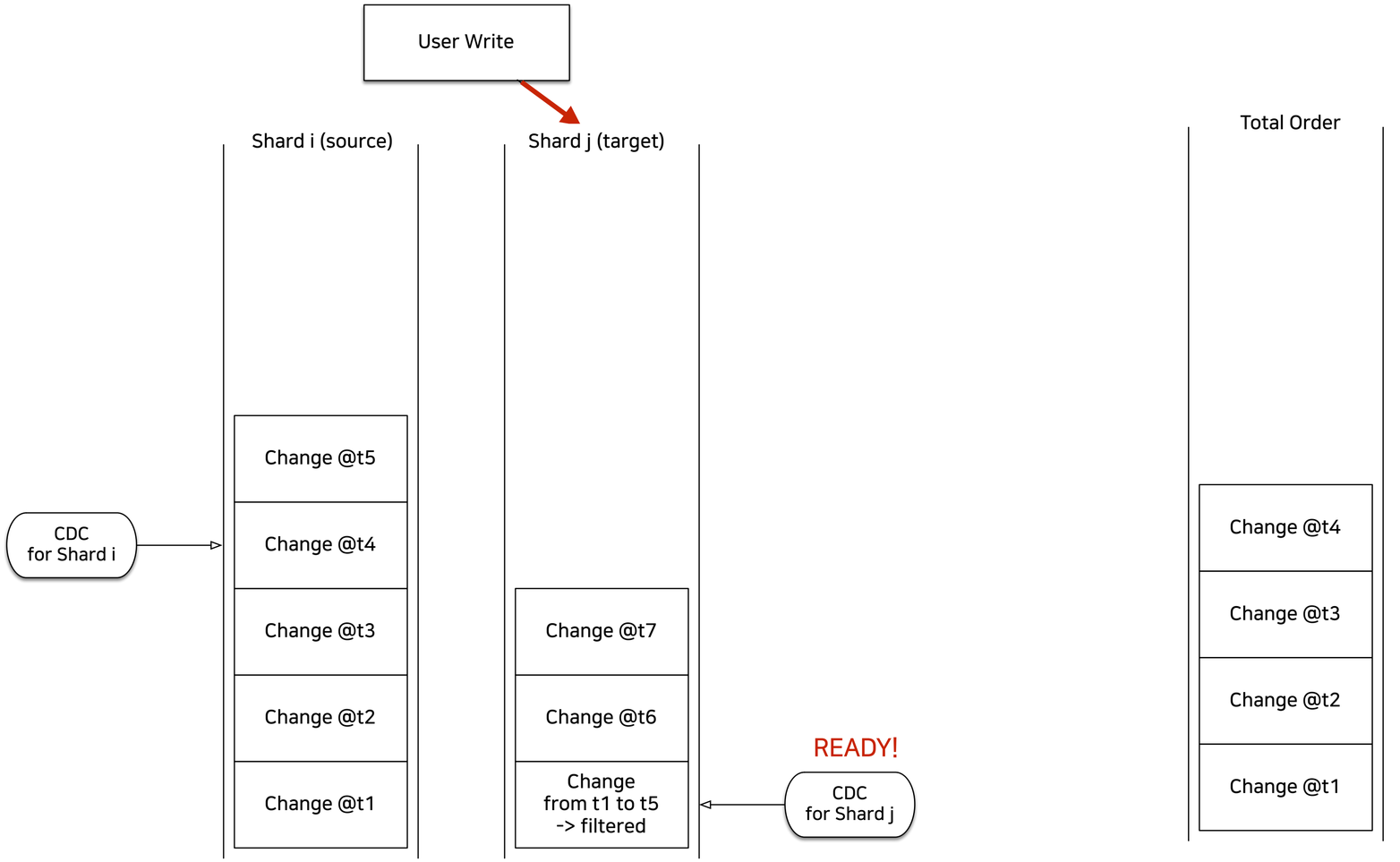


Rebalance를 위한 migration
종료 후 Shard j 를 위한 CDC가
Shard i 를 선행할 경우
Total Ordering에 문제가 발생

각 Shard의 binlog를 독립적으로 처리하는 경우

CDC in Sharded MySQL Cluster

Rebalance 직후 데이터 변경 순서의 역전

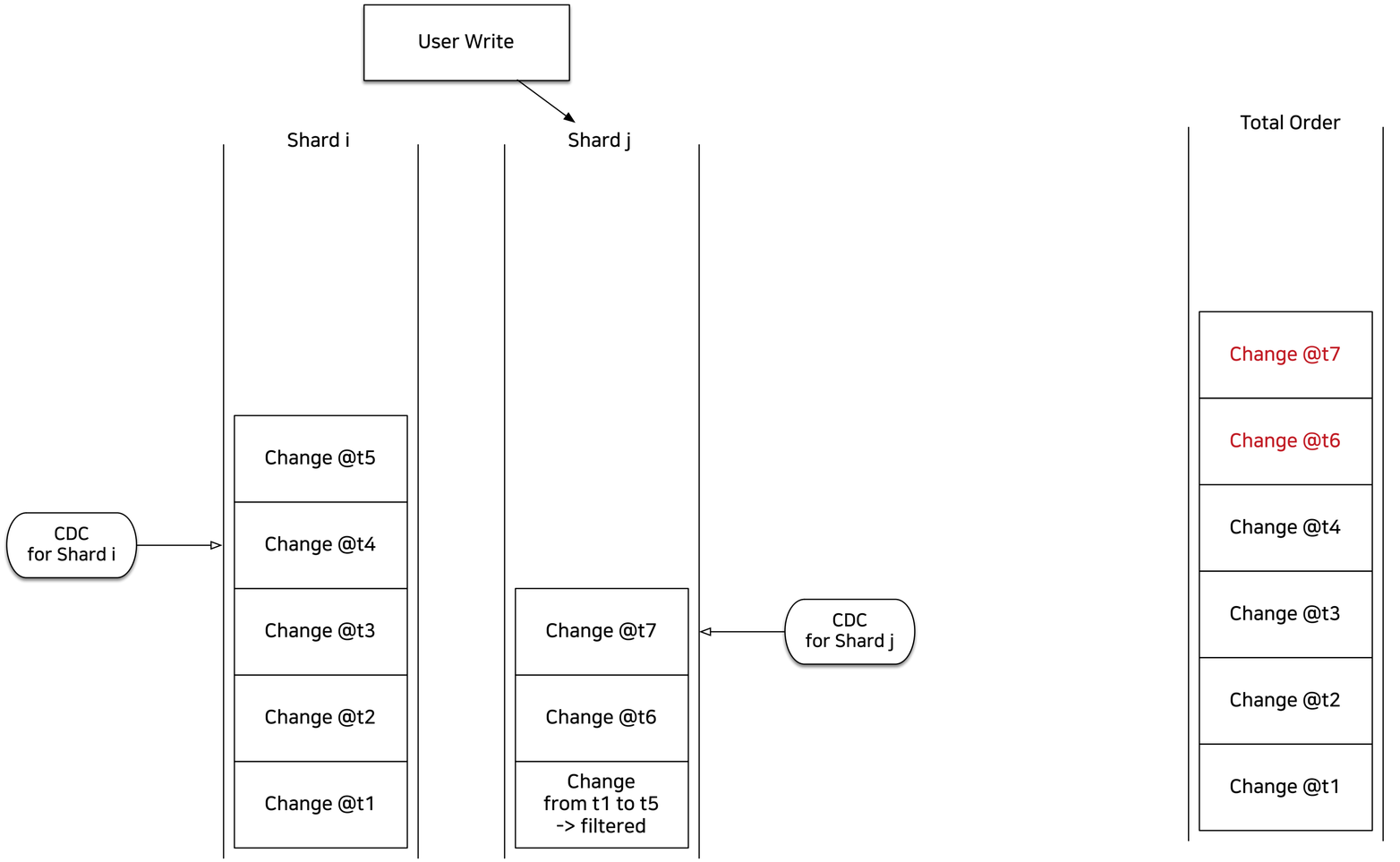


Rebalance를 위한 migration
종료 후 Shard j 를 위한 CDC가
Shard i 를 선행할 경우
Total Ordering에 문제가 발생

각 Shard의 binlog를 독립적으로 처리하는 경우

CDC in Sharded MySQL Cluster

Rebalance 직후 데이터 변경 순서의 역전

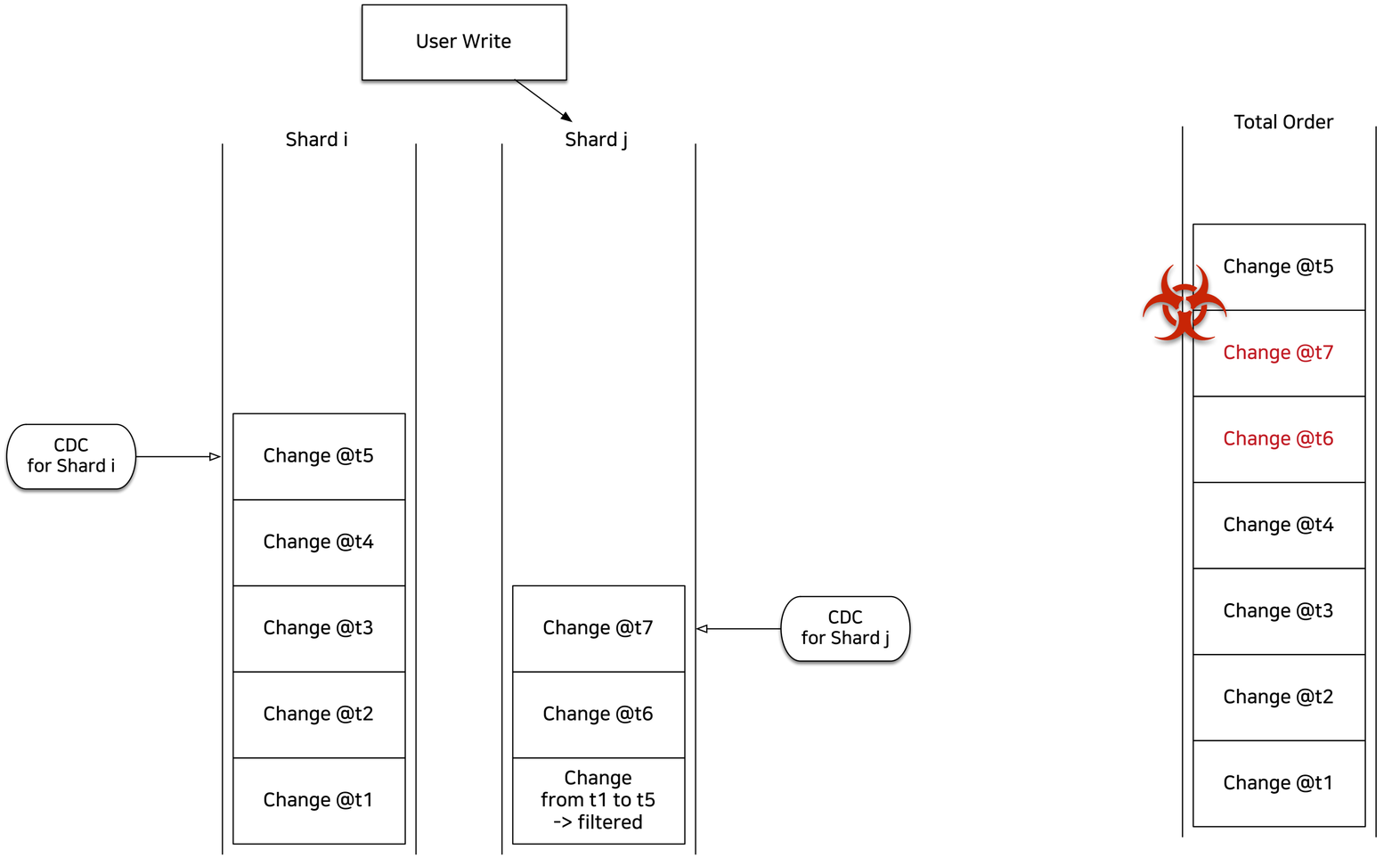


Rebalance를 위한 migration
종료 후 Shard j 를 위한 CDC가
Shard i 를 선행할 경우
Total Ordering에 문제가 발생

각 Shard의 binlog를 독립적으로 처리하는 경우

CDC in Sharded MySQL Cluster

Rebalance 직후 데이터 변경 순서의 역전

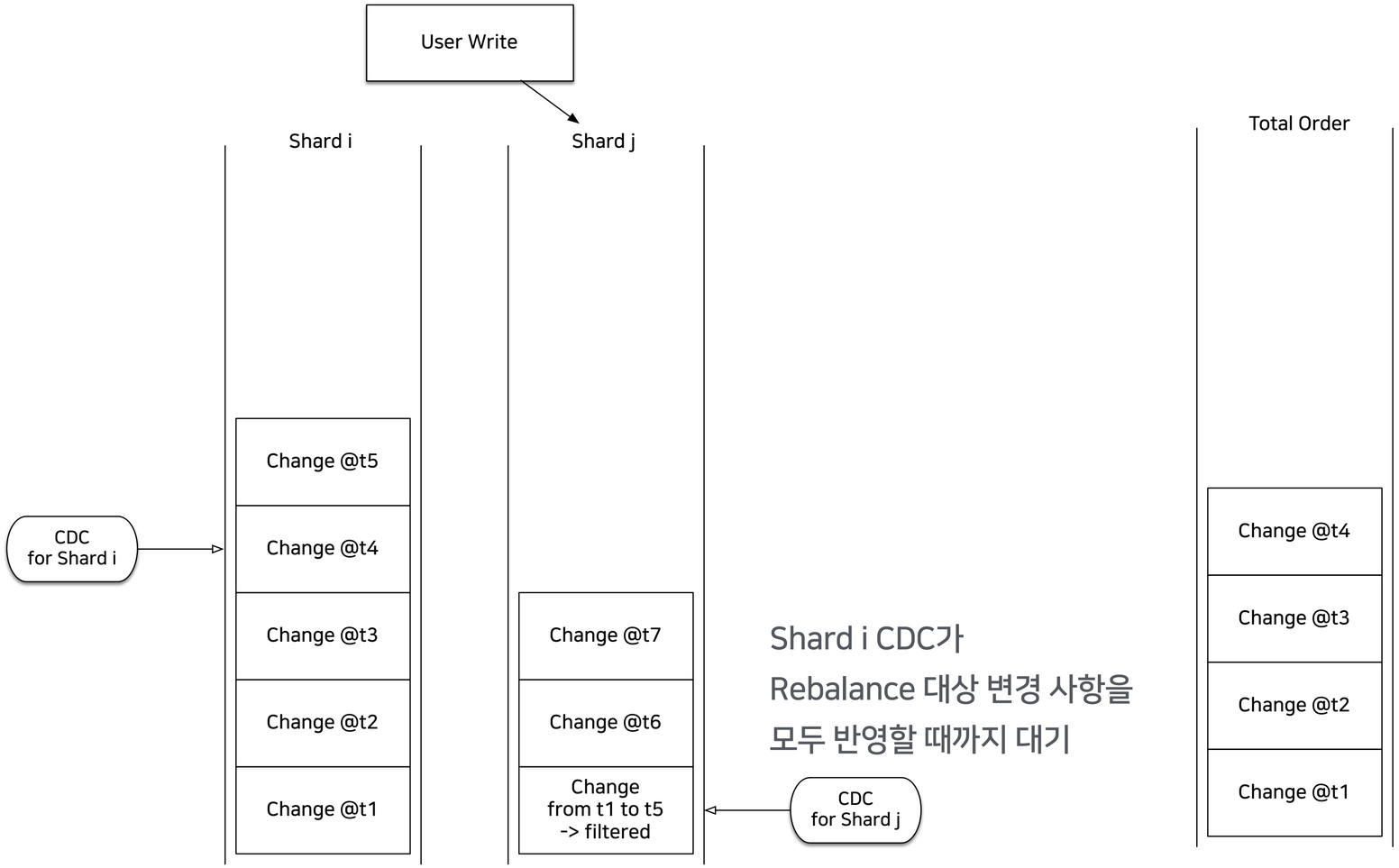


Rebalance를 위한 migration
종료 후 Shard j 를 위한 CDC가
Shard i 를 선행할 경우
Total Ordering에 문제가 발생

각 Shard의 binlog를 독립적으로 처리하는 경우

CDC in Sharded MySQL Cluster

Rebalance 후 데이터 변경 순서의 역전: Agent 간 동기화

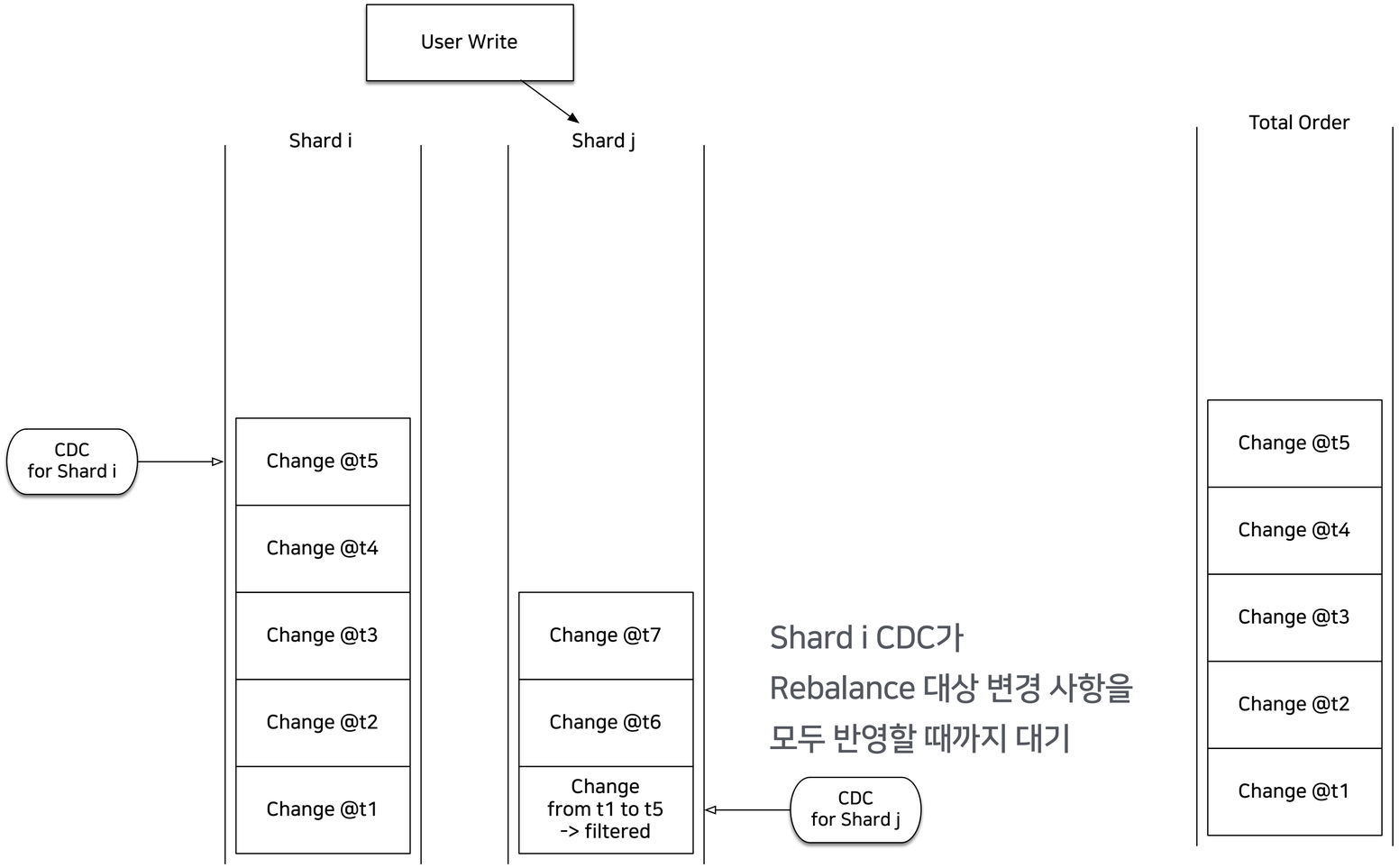


Rebalance 중 CDC agent 간 Schedule 동기화

CDC in Sharded MySQL Cluster

DEVIEW 2021

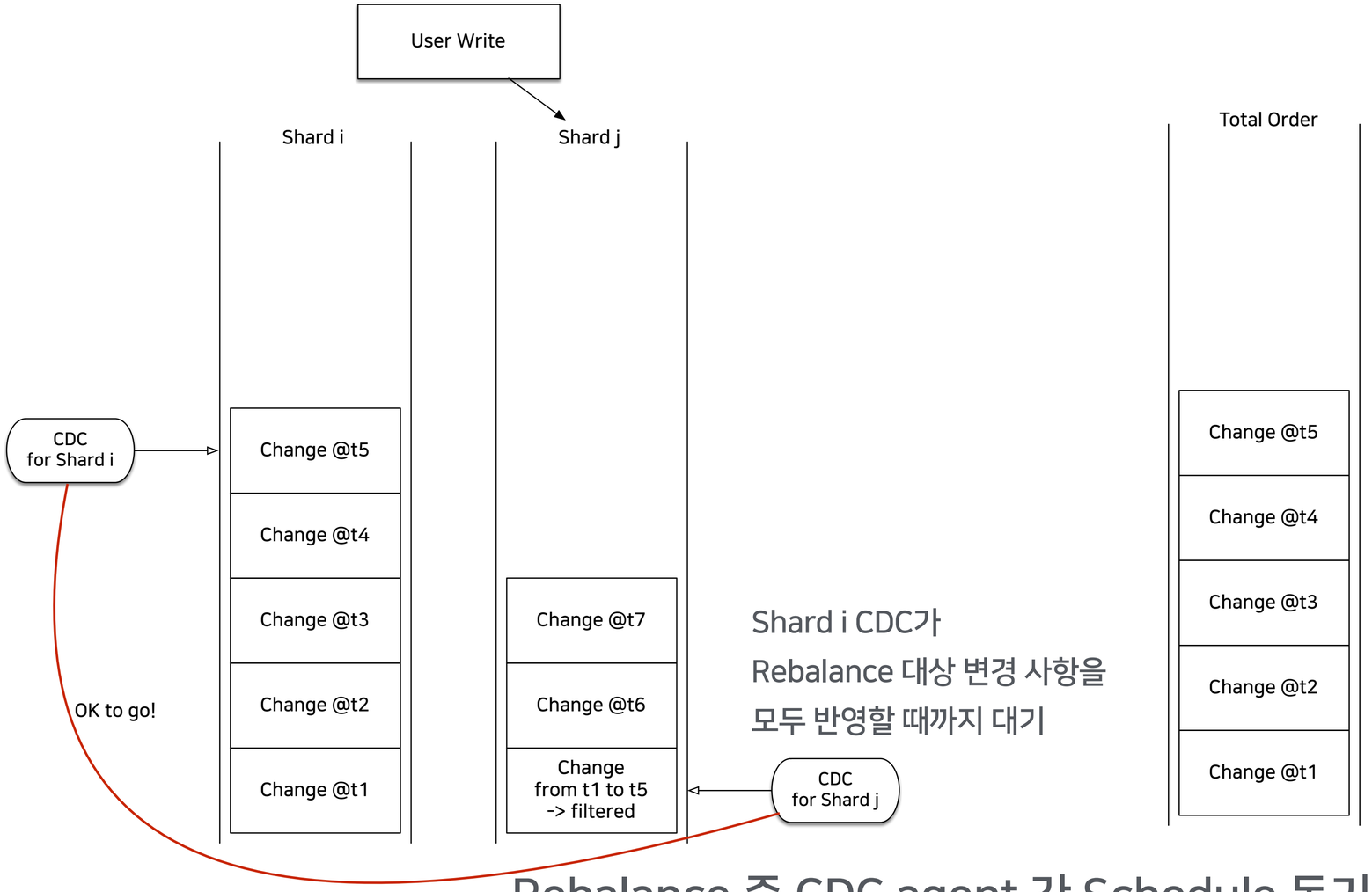
Rebalance 후 데이터 변경 순서의 역전: Agent 간 동기화



Rebalance 중 CDC agent 간 Schedule 동기화

CDC in Sharded MySQL Cluster

Rebalance 후 데이터 변경 순서의 역전: Agent 간 동기화



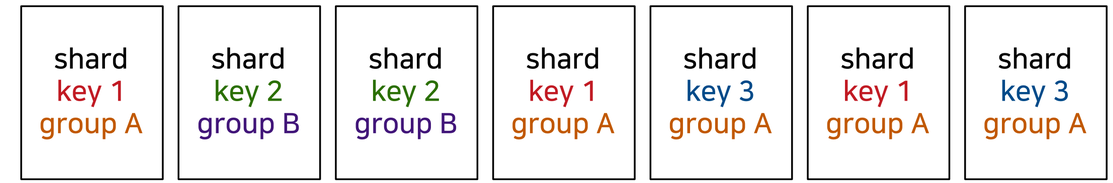
Shard i CDC가
Rebalance 대상 변경 사항을
모두 반영할 때까지 대기

Rebalance 중 CDC agent 간 Schedule 동기화

Performance

Maximize Concurrency

MySQL binglog



Shard Key - 분산의 단위이자 Shard를 특정하는 Key

Shard Group - Shard Key의 Group, Rebalance의 단위

Performance

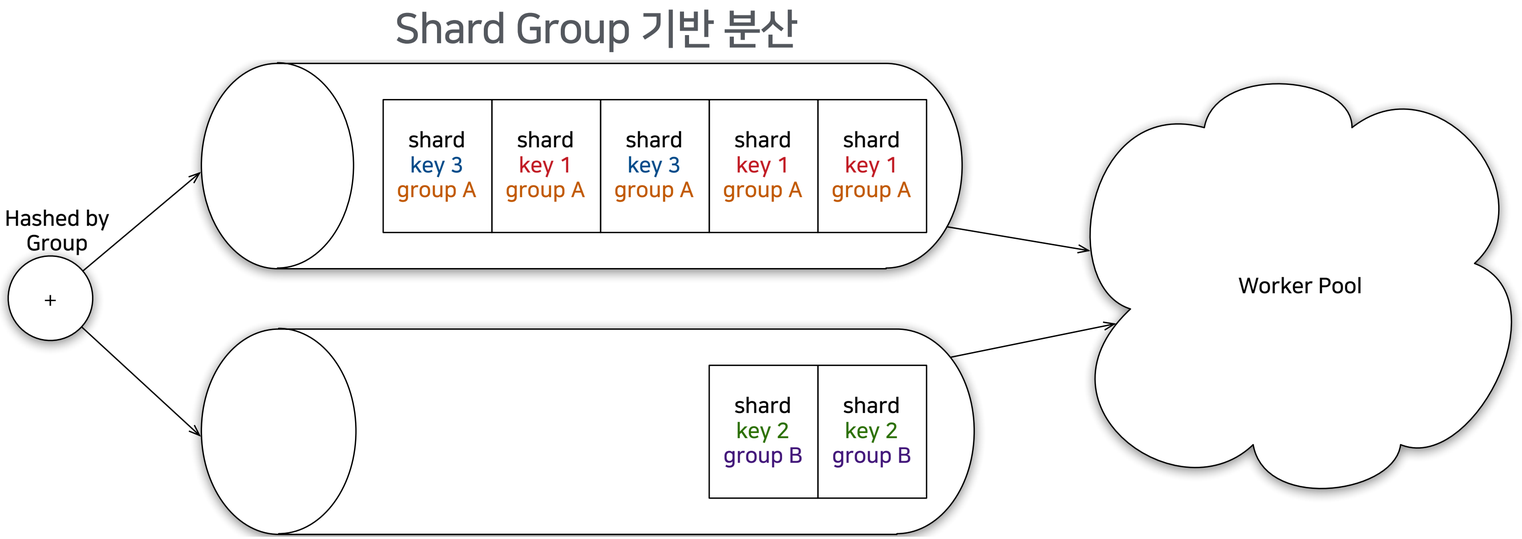
Maximize Concurrency

MySQL binglog

shard key 1 group A	shard key 2 group B	shard key 2 group B	shard key 1 group A	shard key 3 group A	shard key 1 group A	shard key 3 group A	...

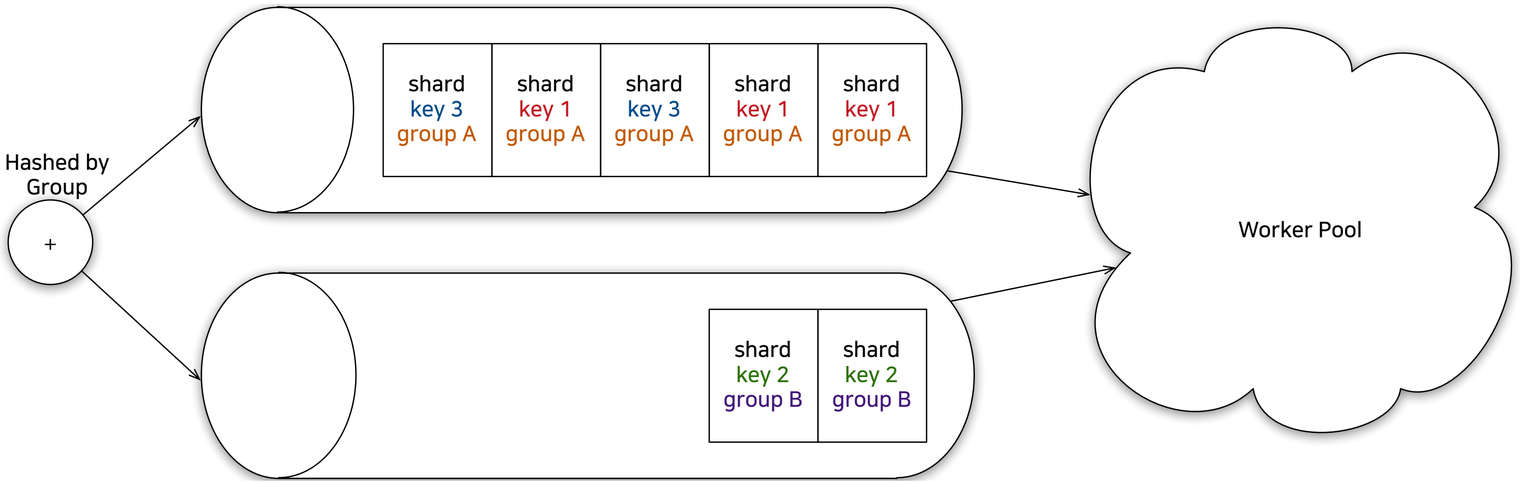
Shard Key - 분산의 단위이자 Shard를 특정하는 Key

Shard Group - Shard Key의 Group, Rebalance의 단위



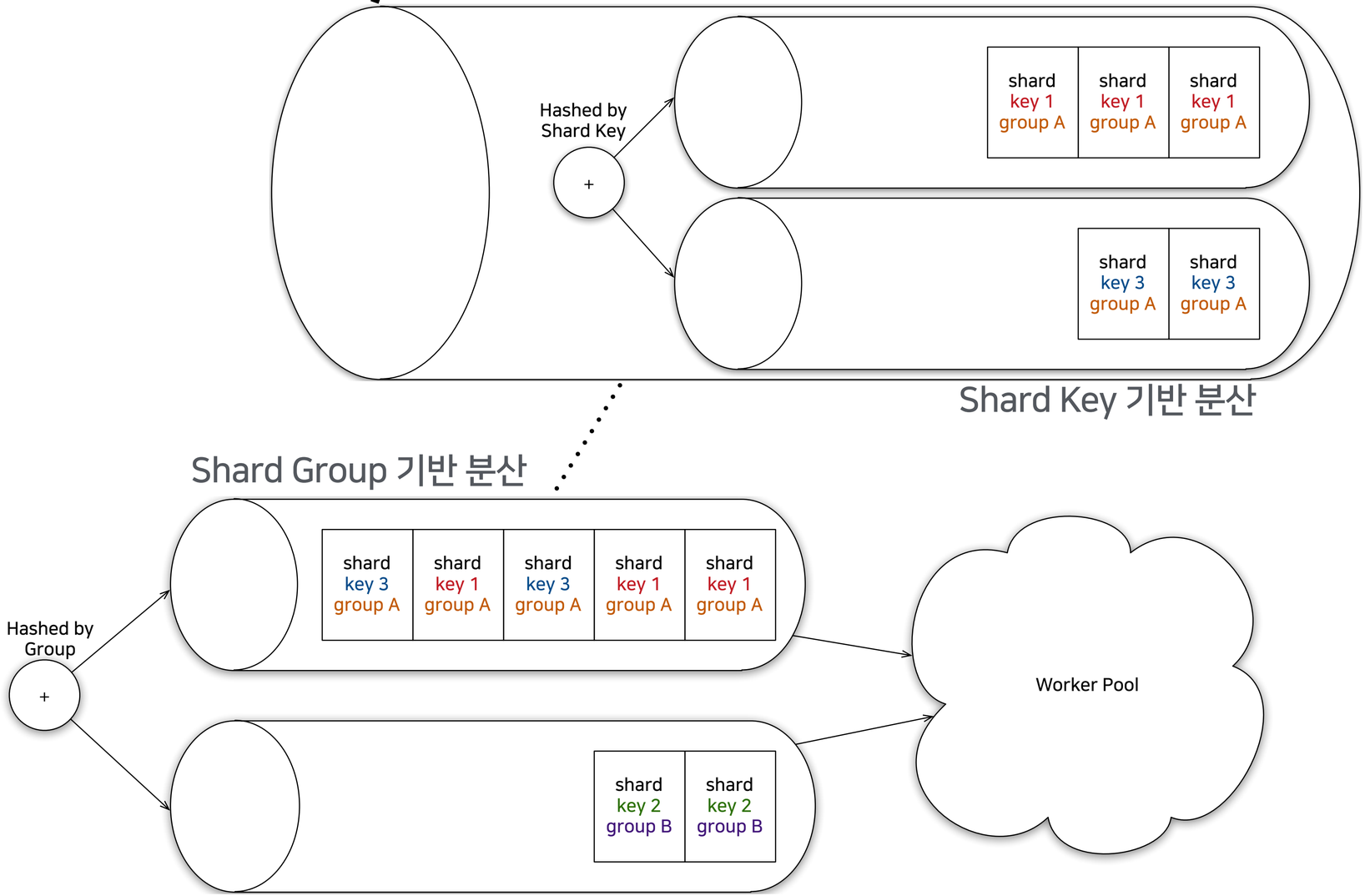
Performance

Maximize Concurrency: Multi-level Queue



Performance

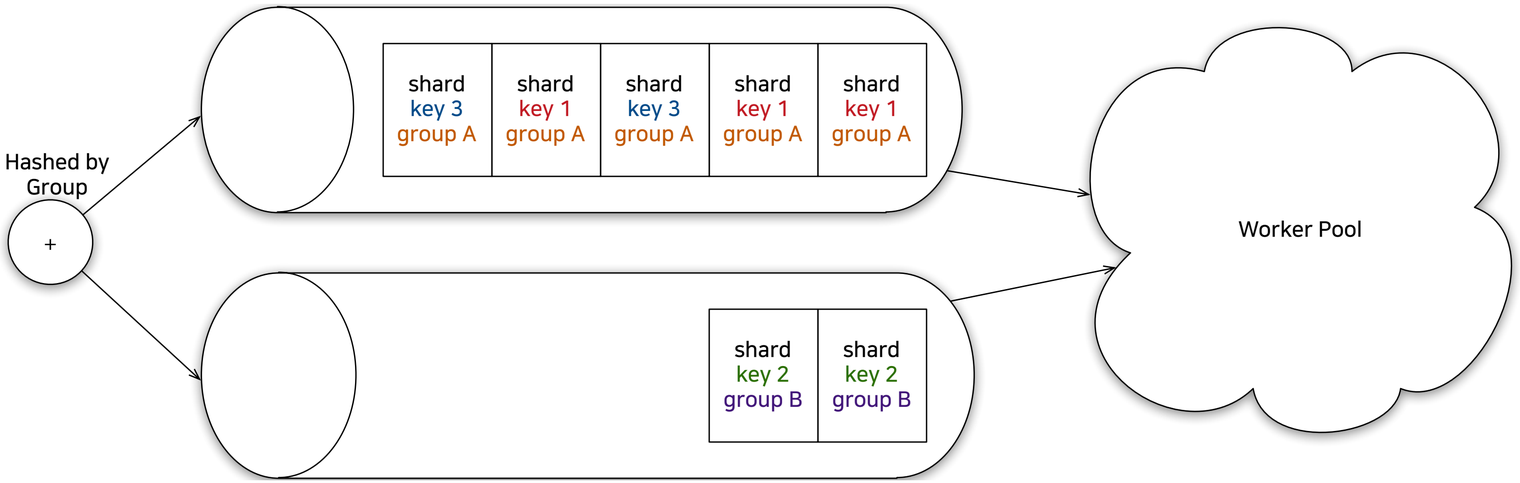
Maximize Concurrency: Multi-level Queue



Performance

Maximize Concurrency: Multi-level Queue

왜 애초에 Shard Key 기반 분산을 하지 않았을까?



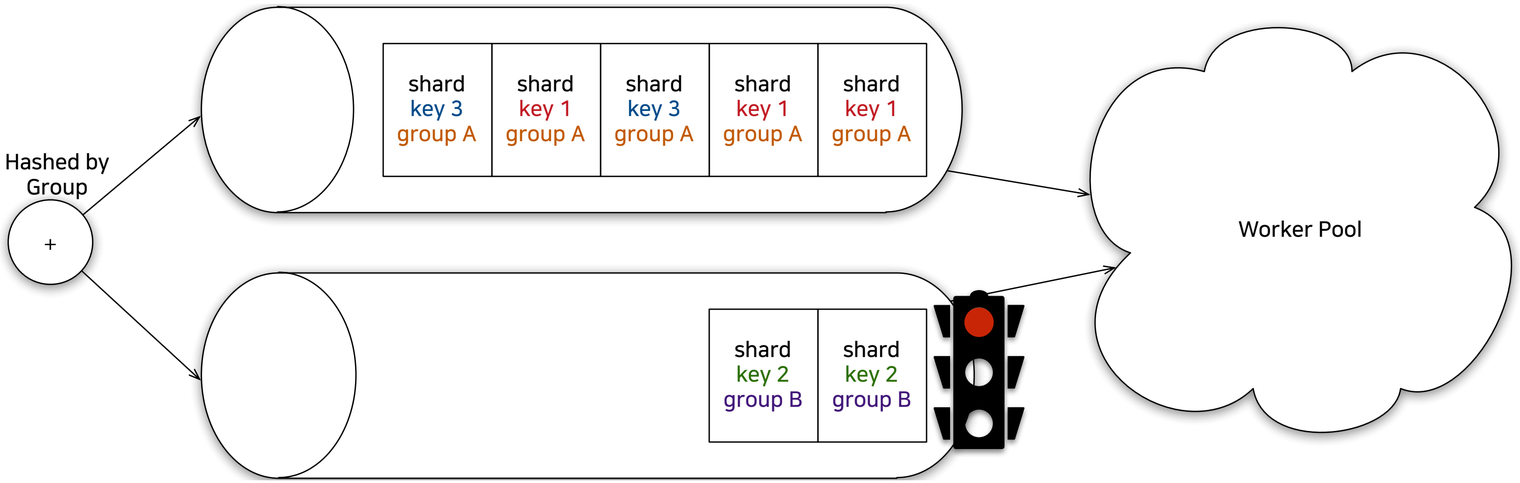
Shard Key - Shard를 특정하는 Key

Shard Group - Shard Key의 Group, Rebalance의 단위

Performance

Maximize Concurrency: Multi-level Queue

왜 애초에 Shard Key 기반 분산을 하지 않았을까?



Shard Rebalance를 위한
CDC 동기화 Blocking

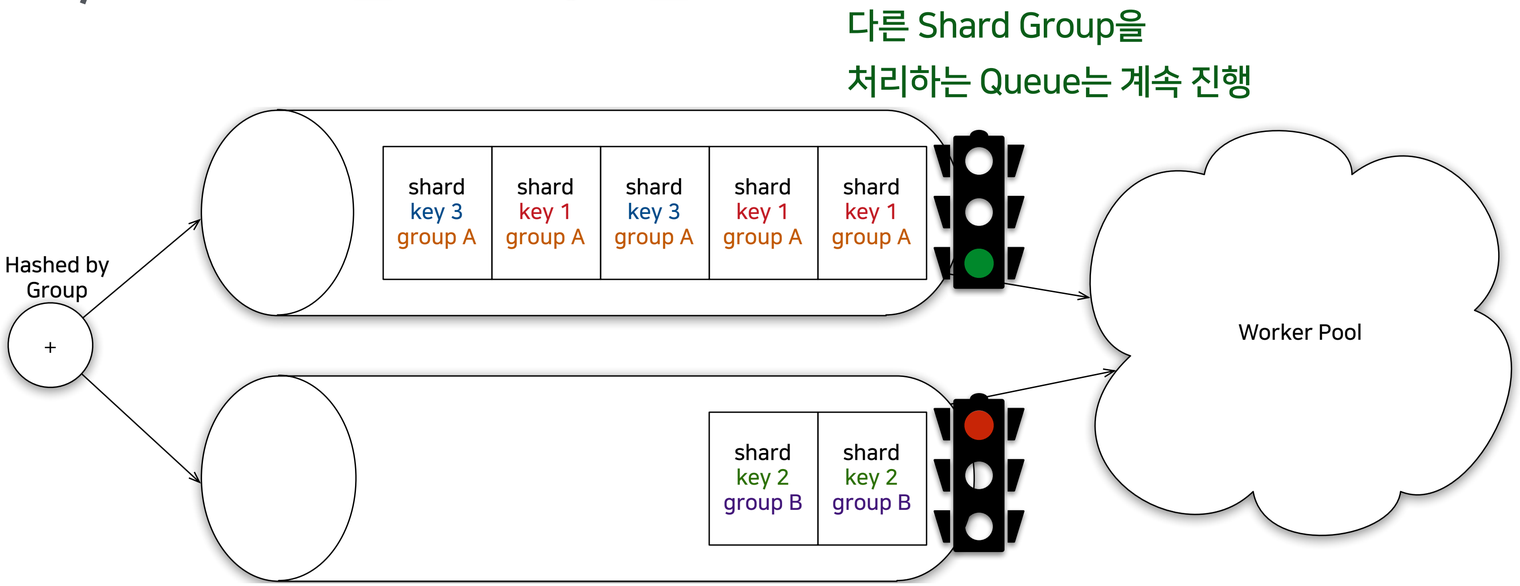
Shard Key - Shard를 특정하는 Key

Shard Group - Shard Key의 Group, Rebalance의 단위

Performance

Maximize Concurrency: Multi-level Queue

왜 애초에 Shard Key 기반 분산을 하지 않았을까?

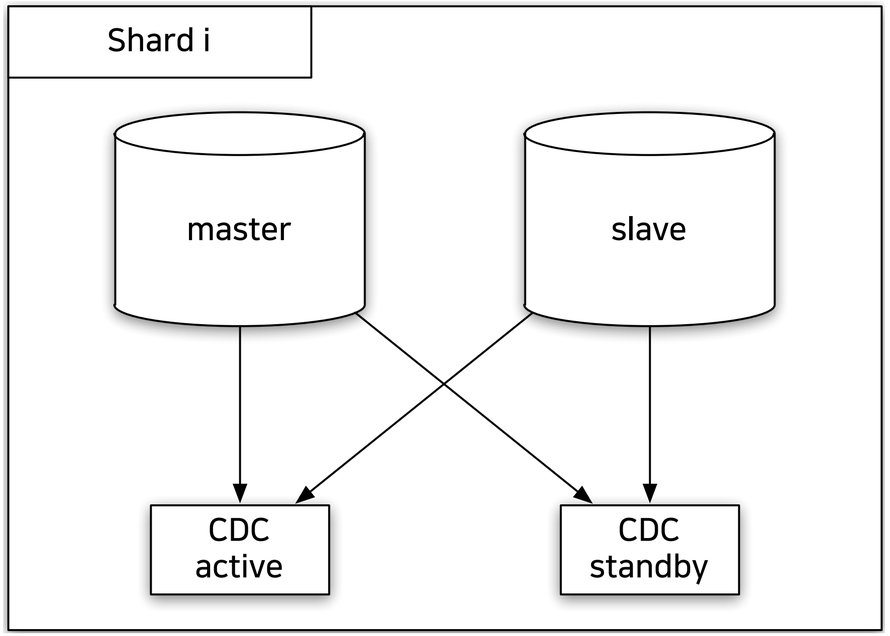


Shard Key - Shard를 특정하는 Key

Shard Group - Shard Key의 Group, Rebalance의 단위

High Availability

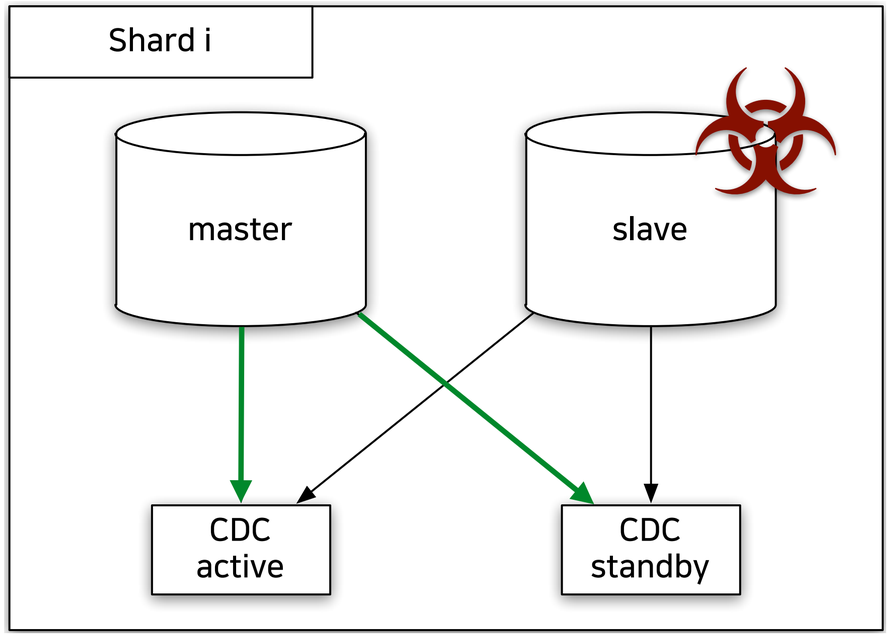
안정적인 복제를 위한 첫걸음



GTID 기반의 active-standby 모델

High Availability

안정적인 복제를 위한 첫걸음

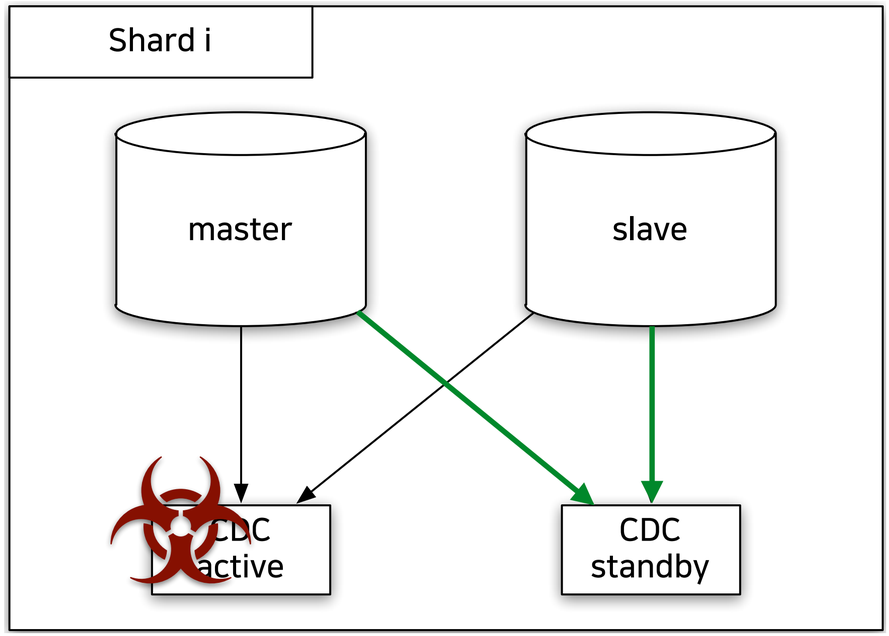


GTID 기반의 active-standby 모델

DB와 CDC agent 어느 구간에서
장애가 발생하여도 즉각적으로
복제를 재개

High Availability

안정적인 복제를 위한 첫걸음

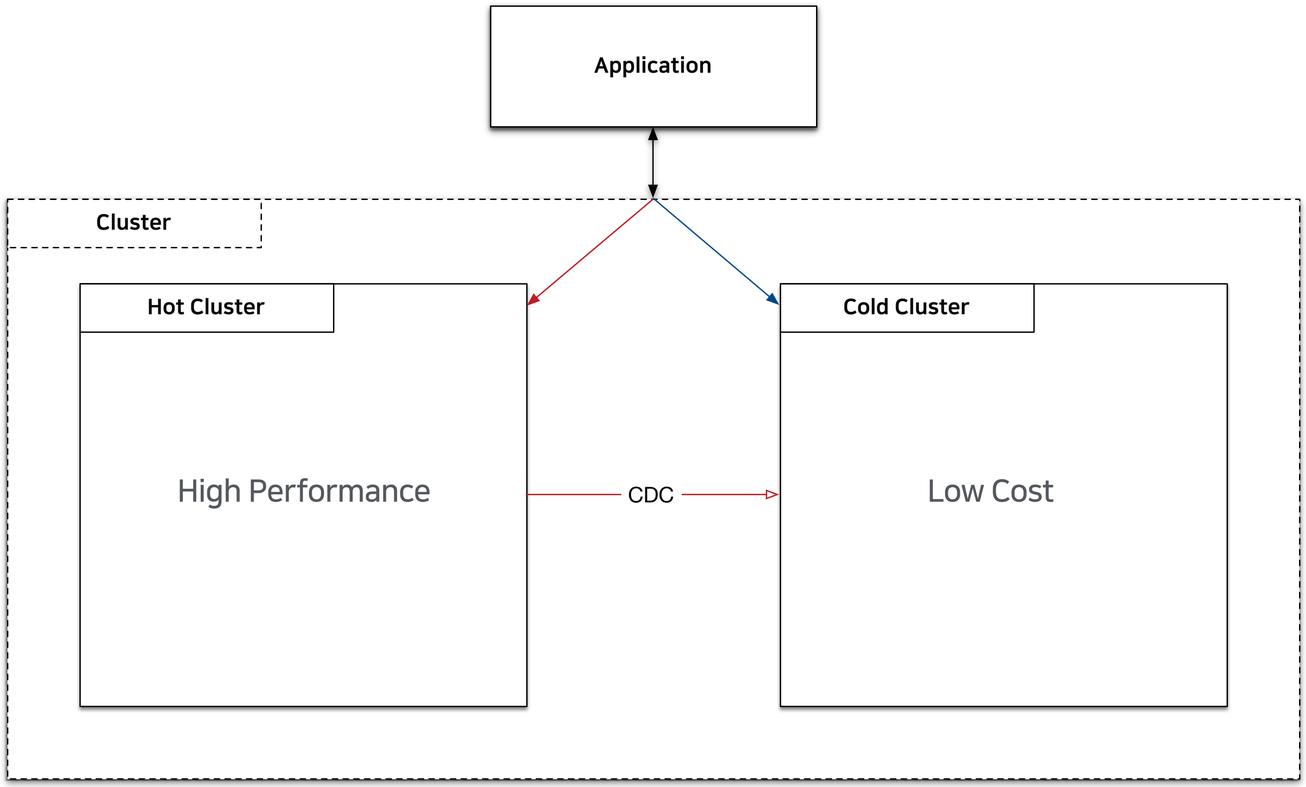


GTID 기반의 active-standby 모델

DB와 CDC agent 어느 구간에서
장애가 발생하여도 즉각적으로
복제를 재개

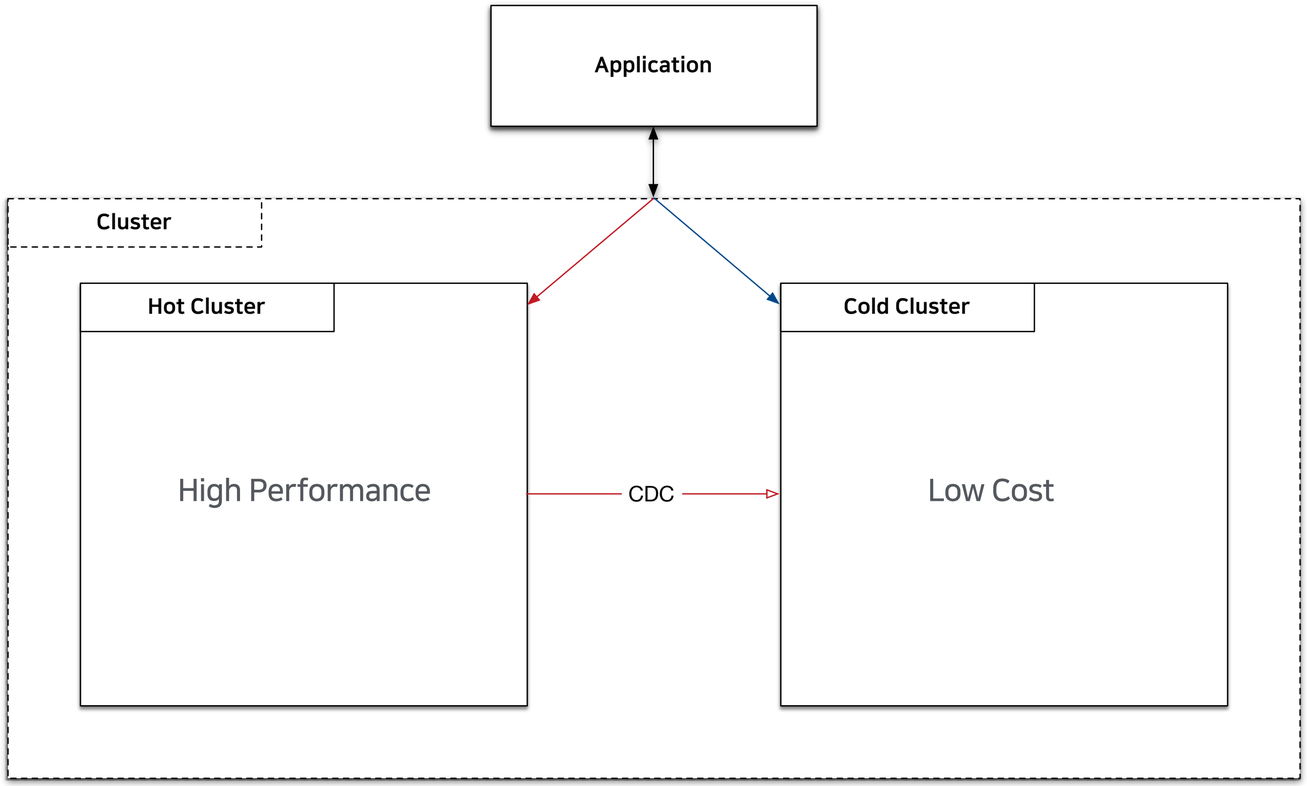
Data Tiering

뜨거운 데이터는 빠르게, 차가운 데이터 값싸게



Data Tiering

CDC를 통한 Hot → Cold 복제



Hot Cluster의 데이터를 Cold Cluster에 복제

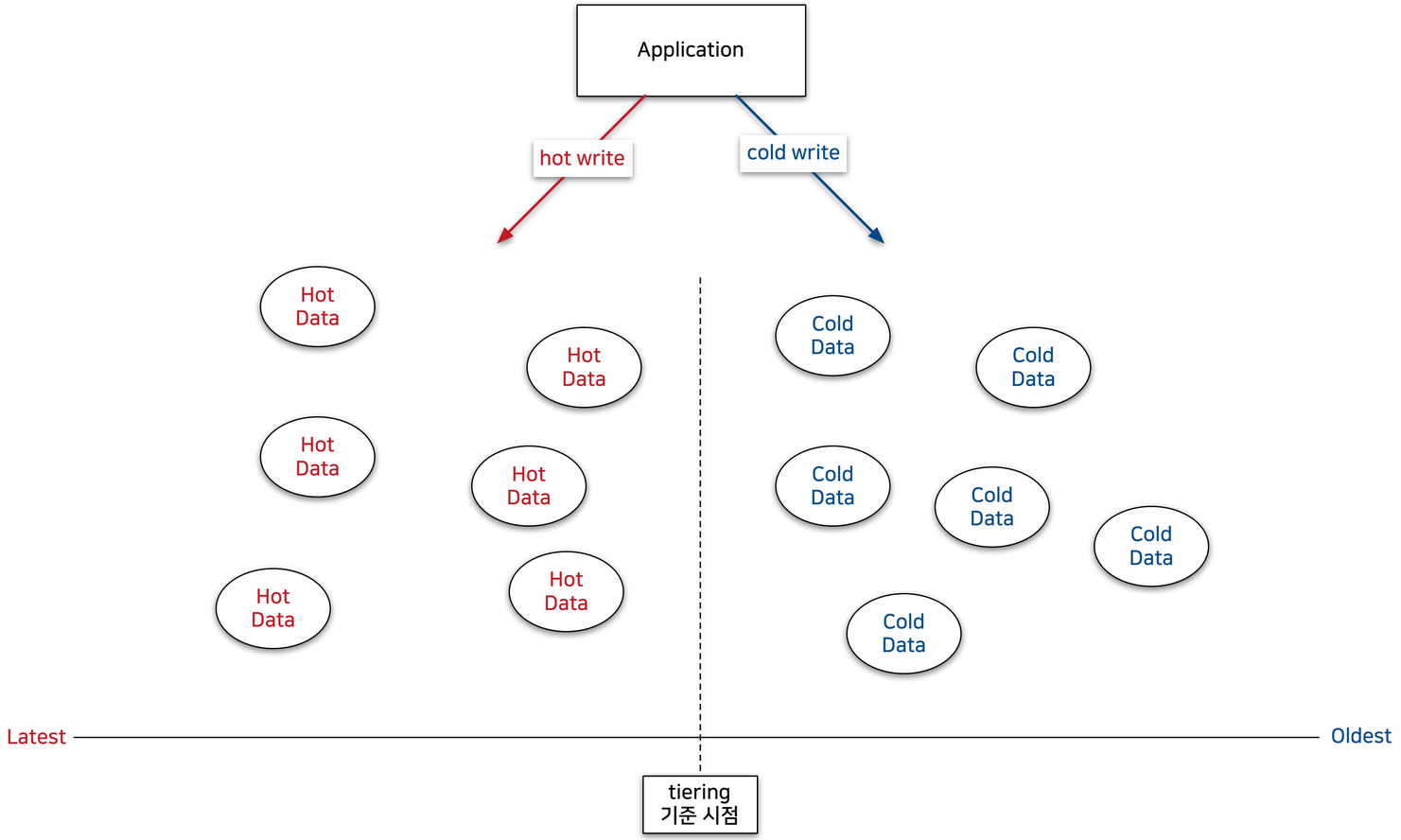
기준 Tiering 시점 이전의 데이터를 Hot에서 삭제,
삭제 부하는 CDC에 의해 필터링

각 테이블은 기준 시점을 적용할 컬럼과 조건 정의

```
CREATE TABLE xxx_order (
  ...
  order_no VARCHAR(20) NOT NULL,
  ...
) TIERING BY order_no < FROM_UNIXTIME(?, '%Y%m%d');
```

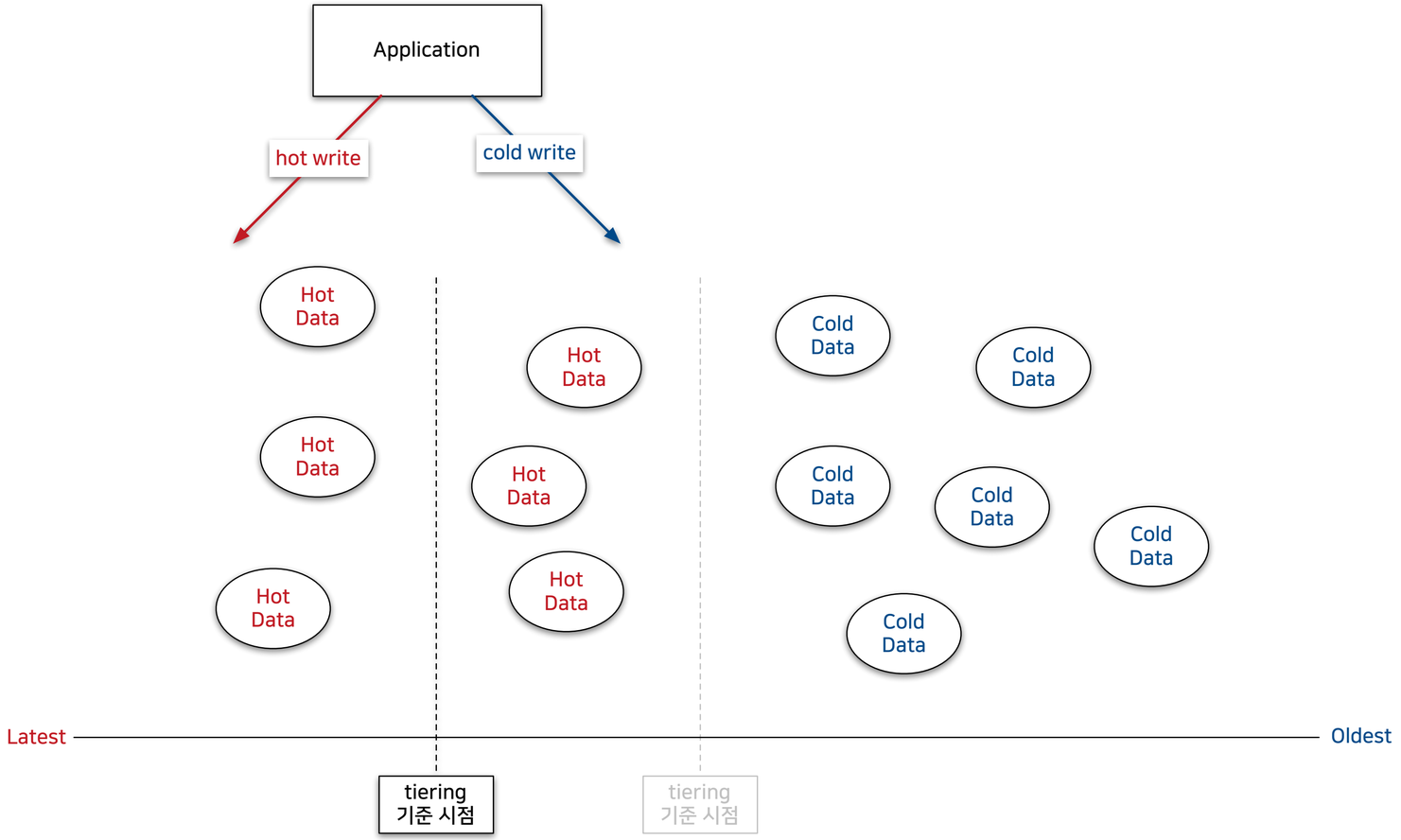
CDC in Data Tiering

Tiering 기준 시점 변경의 어려움



CDC in Data Tiering

Tiering 기준 시점 변경의 어려움

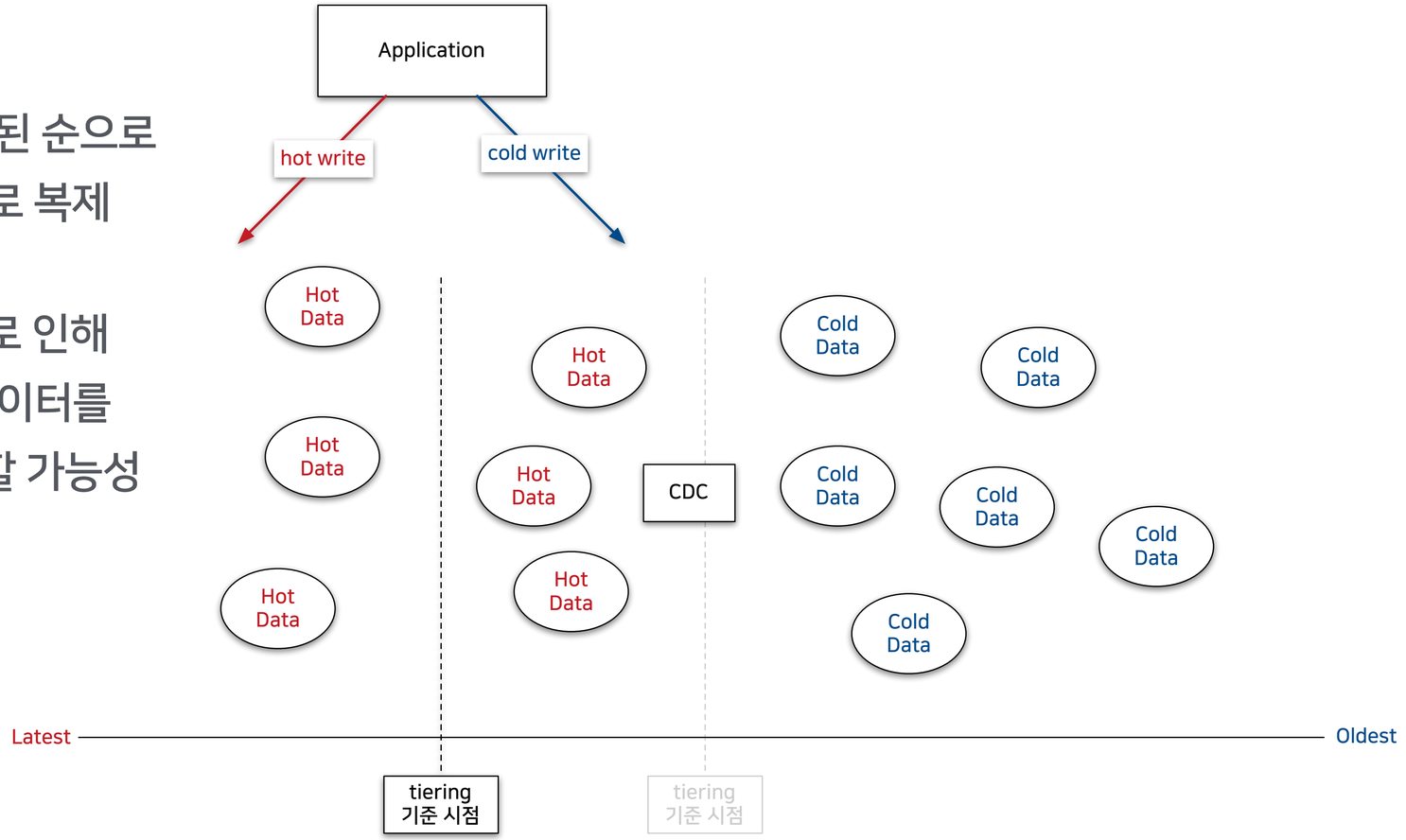


CDC in Data Tiering

Tiering 기준 시점 변경의 어려움

CDC는 binlog에 기록된 순으로
Hot의 데이터를 Cold로 복제

복제의 비동기 특성으로 인해
복제 되지 못한 Hot 데이터를
Cold Cluster에 요청할 가능성

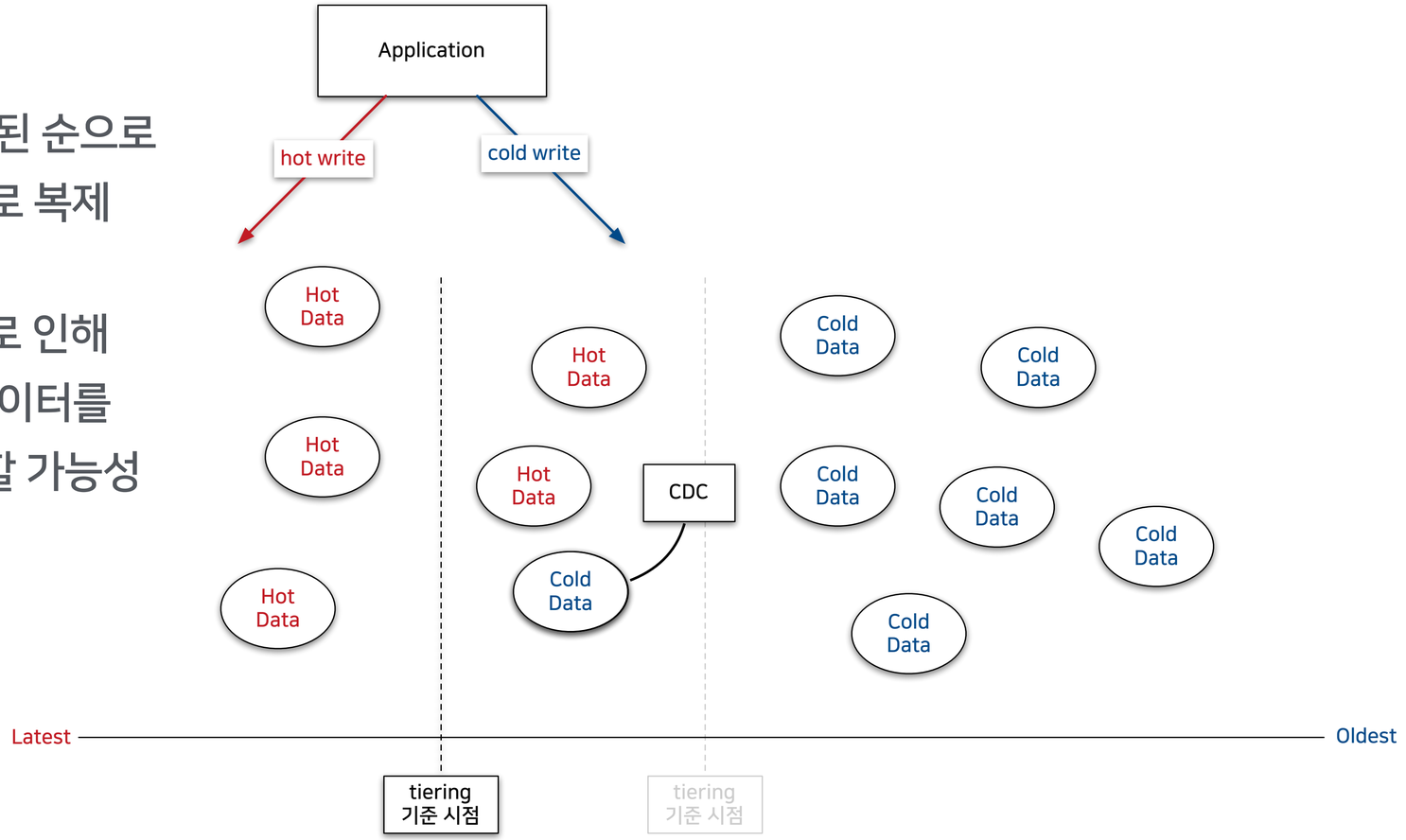


CDC in Data Tiering

Tiering 기준 시점 변경의 어려움

CDC는 binlog에 기록된 순으로
Hot의 데이터를 Cold로 복제

복제의 비동기 특성으로 인해
복제 되지 못한 Hot 데이터를
Cold Cluster에 요청할 가능성

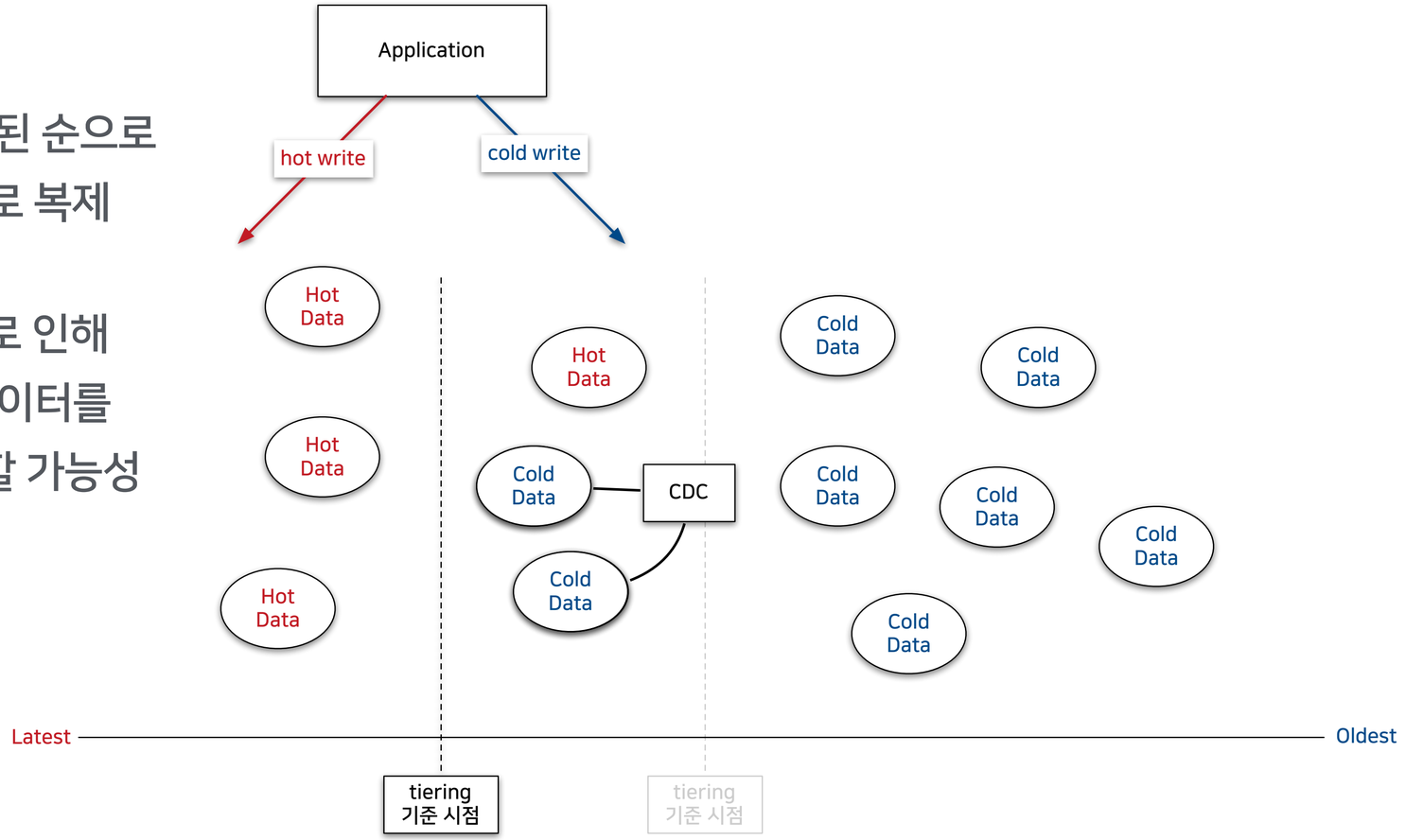


CDC in Data Tiering

Tiering 기준 시점 변경의 어려움

CDC는 binlog에 기록된 순으로
Hot의 데이터를 Cold로 복제

복제의 비동기 특성으로 인해
복제 되지 못한 Hot 데이터를
Cold Cluster에 요청할 가능성

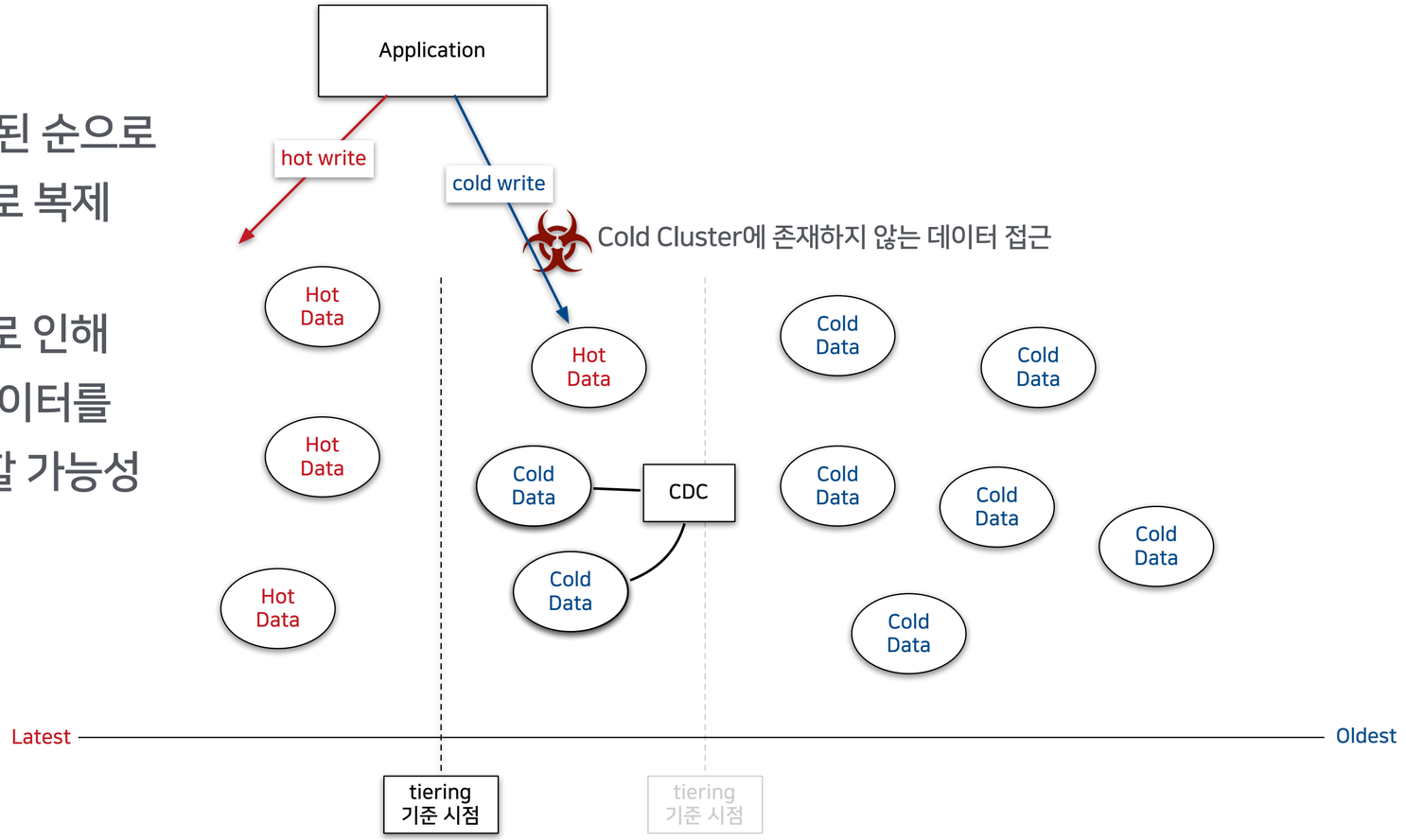


CDC in Data Tiering

Tiering 기준 시점 변경의 어려움

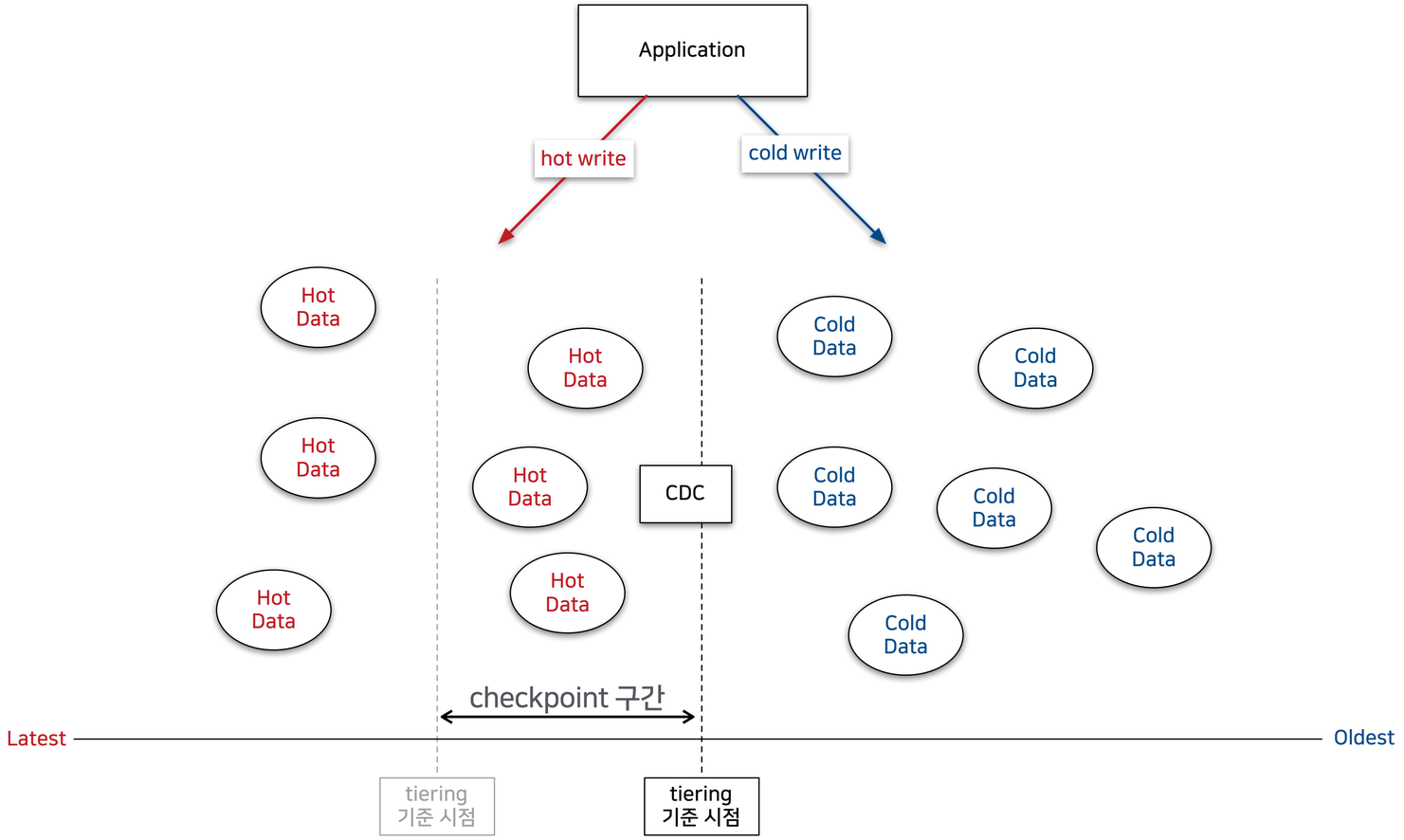
CDC는 binlog에 기록된 순으로
Hot의 데이터를 Cold로 복제

복제의 비동기 특성으로 인해
복제 되지 못한 Hot 데이터를
Cold Cluster에 요청할 가능성



CDC in Data Tiering

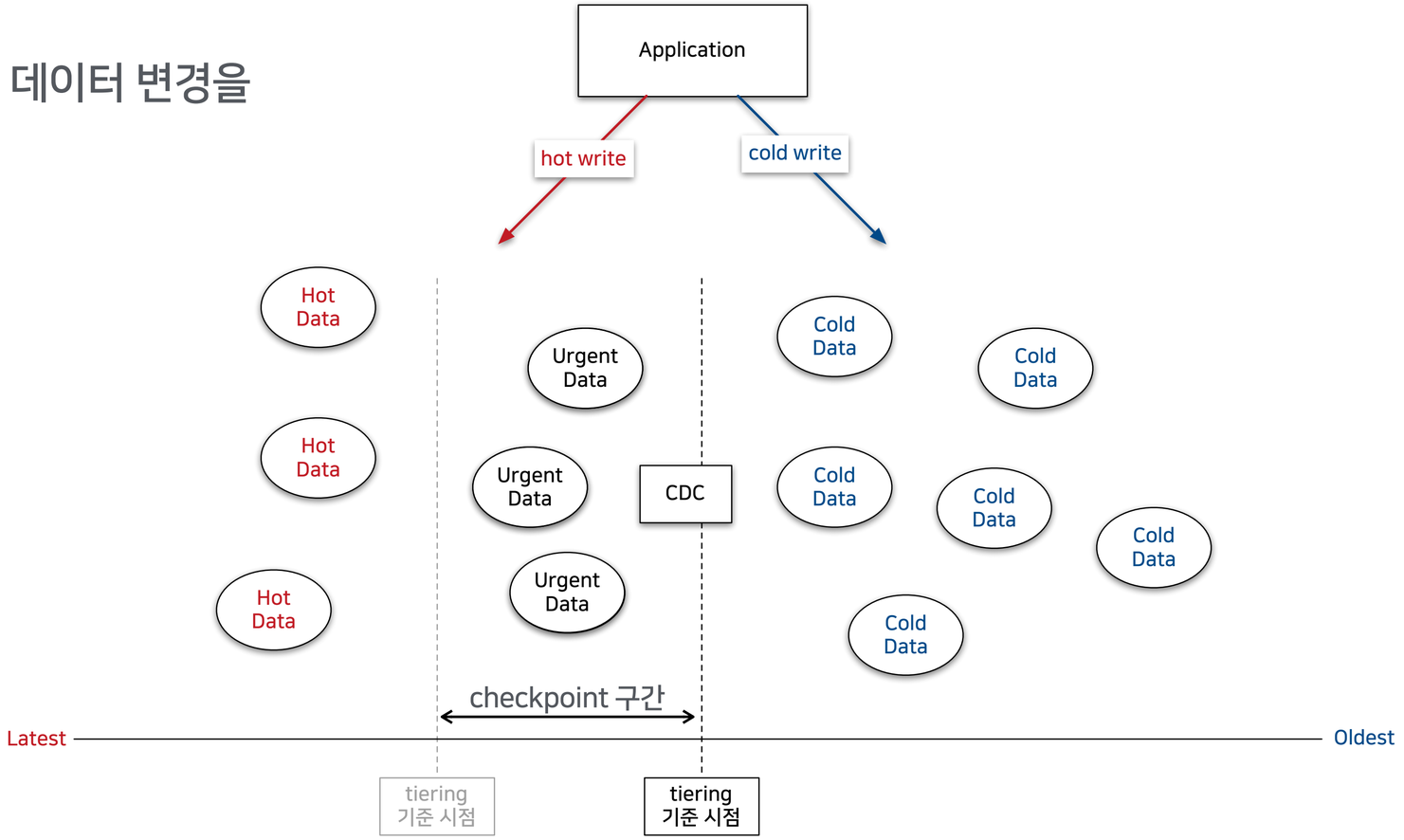
Tiering 기준 시점 변경의 어려움 극복: Checkpoint



CDC in Data Tiering

Tiering 기준 시점 변경의 어려움 극복: Checkpoint

1. checkpoint 구간의 데이터 변경을 urgent로 마킹

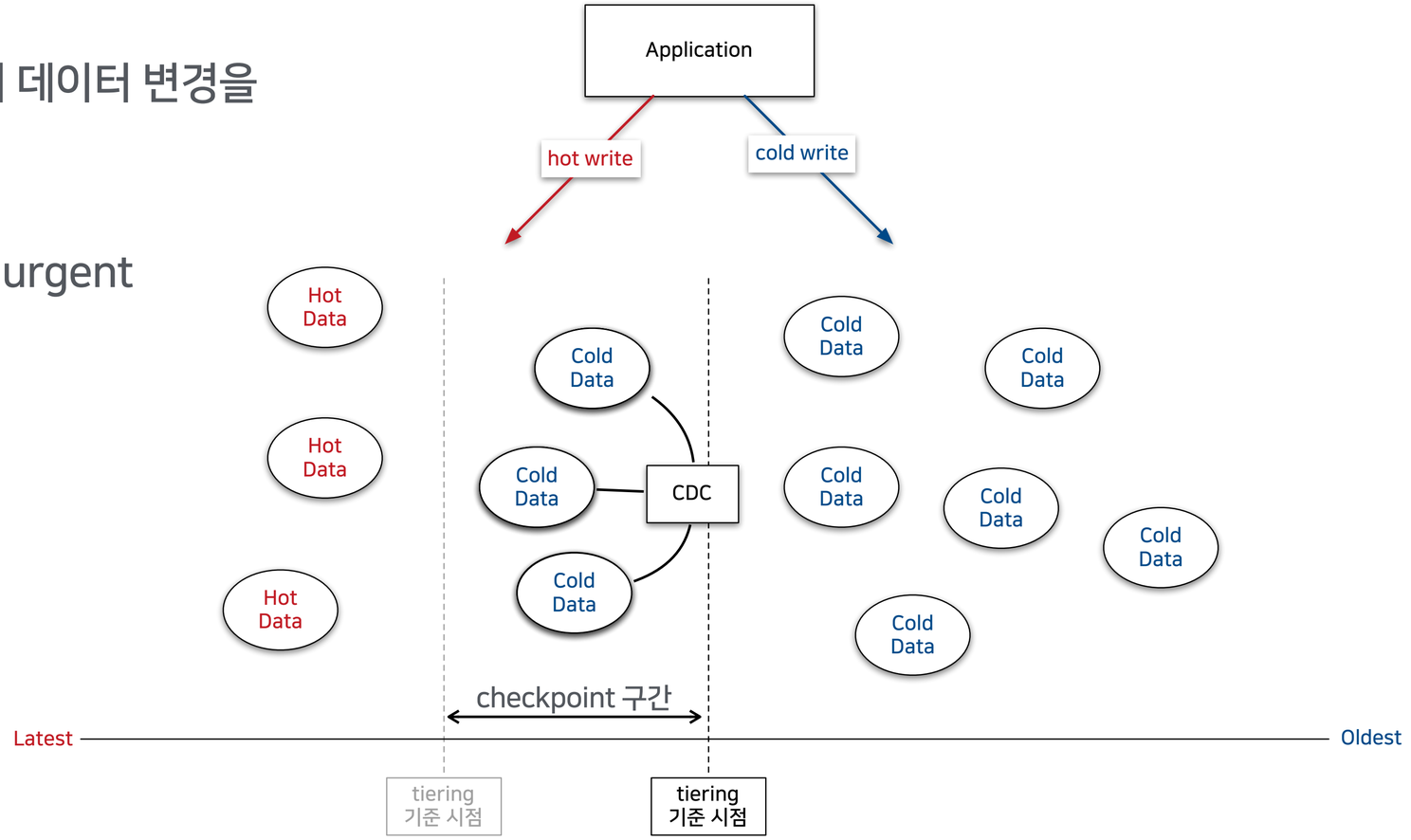


CDC in Data Tiering

Tiering 기준 시점 변경의 어려움 극복: Checkpoint

1. checkpoint 구간의 데이터 변경을 urgent로 마킹

2. binlog를 skip하며 urgent 데이터를 우선 복제

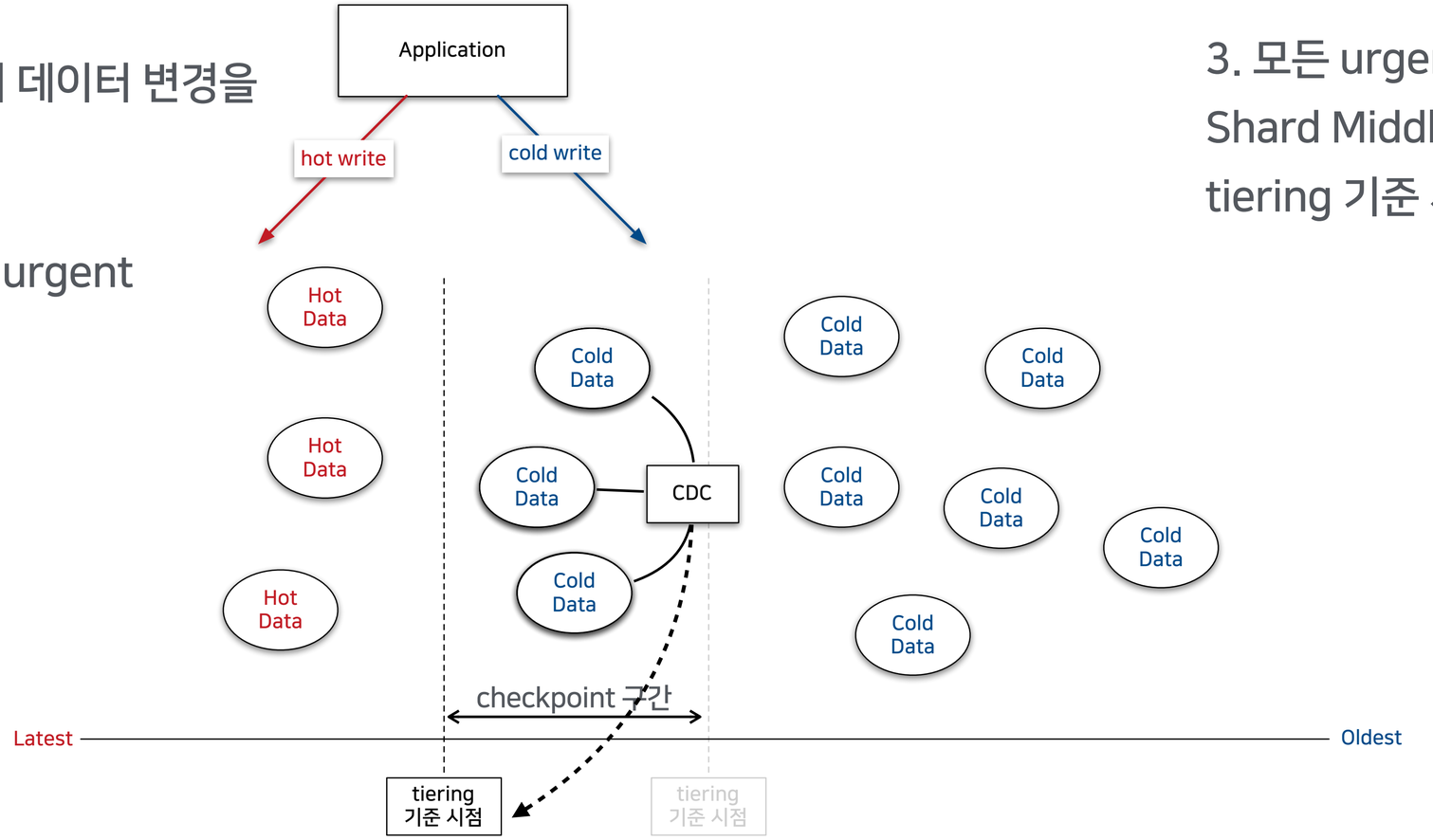


CDC in Data Tiering

Tiering 기준 시점 변경의 어려움 극복: Checkpoint

1. checkpoint 구간의 데이터 변경을 urgent로 마킹

2. binlog를 skip하며 urgent 데이터를 우선 복제



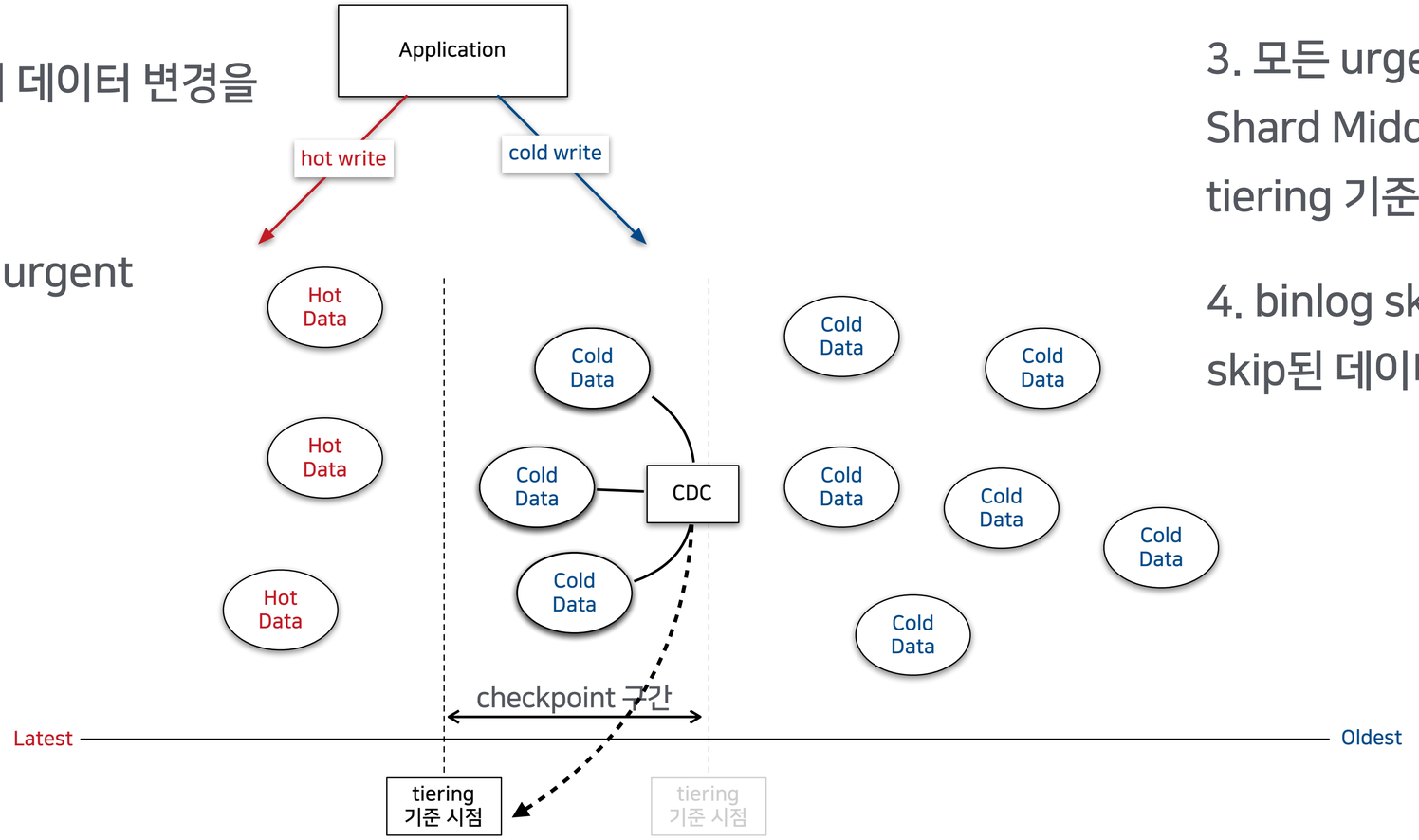
3. 모든 urgent 데이터 복제 완료 시 Shard Middleware에서 tiering 기준 시점 업데이트

CDC in Data Tiering

Tiering 기준 시점 변경의 어려움 극복: Checkpoint

1. checkpoint 구간의 데이터 변경을 urgent로 마킹

2. binlog를 skip하며 urgent 데이터를 우선 복제



3. 모든 urgent 데이터 복제 완료 시 Shard Middleware에서 tiering 기준 시점 업데이트

4. binlog skip 이전 시점으로 복귀하여 skip된 데이터 복제 재개

Change Data Capture

DEVIEW 2021

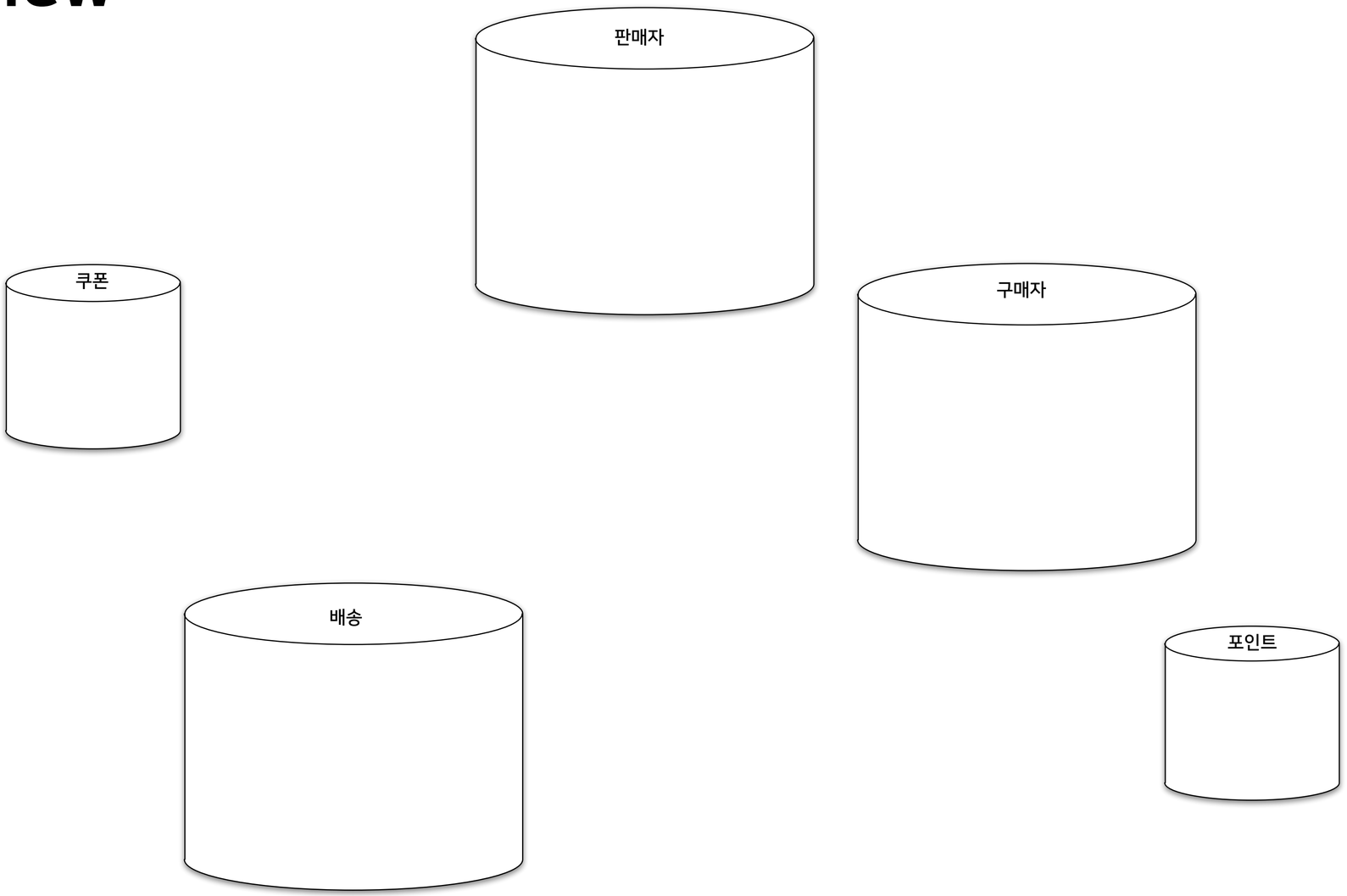
Customizability

- filter
- converter
  - table, column name + conditional value converter
- post-process handler
  - 처리 완료된 row 마킹/삭제
  - extensible fallback
    - 처리 실패에 대한 dump, 에러 마킹
  - 다양한 target
    - sharded cluster, MySQL, Oracle, Kafka
    - trigger message, outbox pattern

Change Data Capture @Naver Pay

DEVIEW 2021

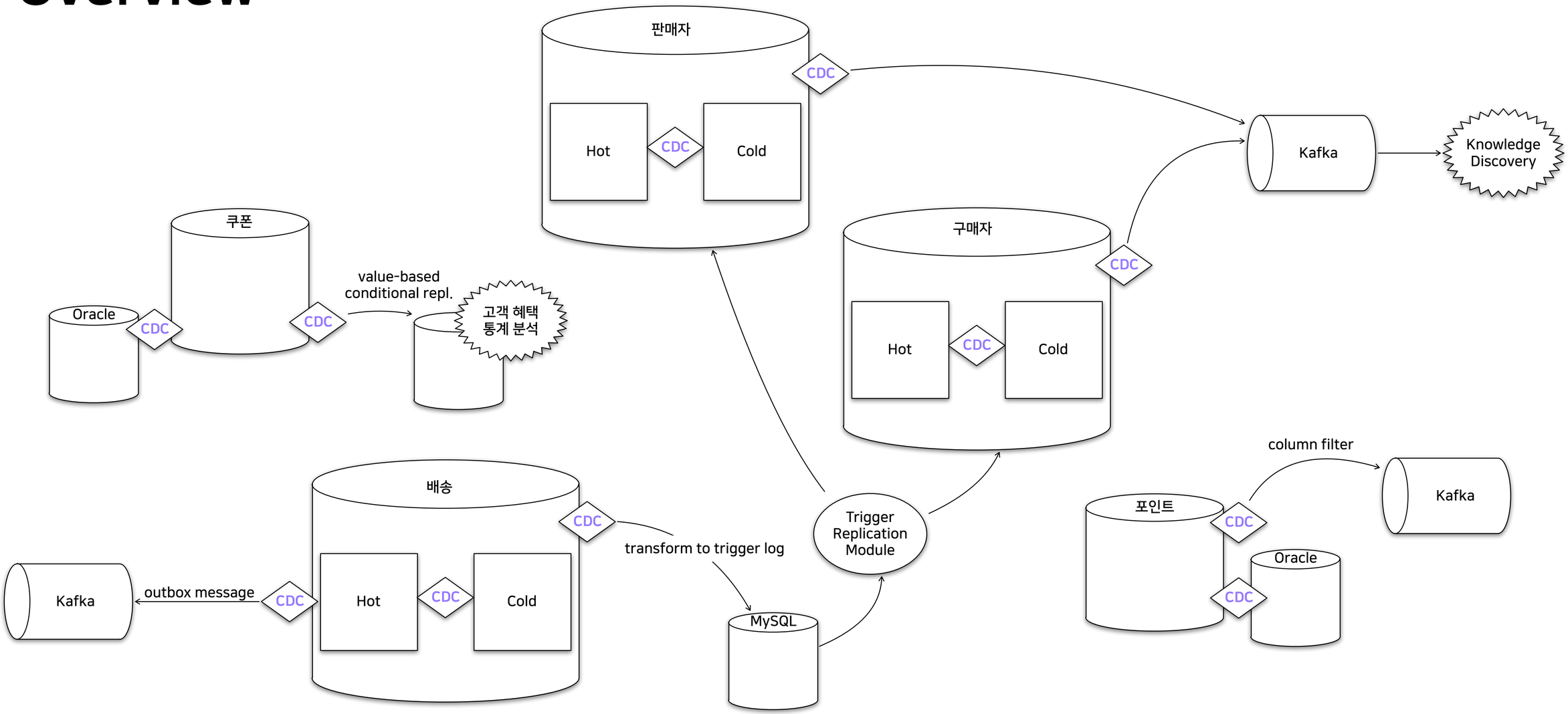
Overview



Change Data Capture @Naver Pay

DEVIEW 2021

Overview



DEVIEW 2021

CDC 기반 Materialized View 로 분산 Data Modeling 문제 해결

Data Modeling Problem on Sharded Cluster

적절한 Shard Key 선정의 어려움

<Unsharded DB>

Merchant ID	Purchaser ID	Order Date	Order Msg
1	100	2021-01-01	null
1	101	2021-01-02	AA
2	101	2021-01-03	BB
2	101	2021-01-04	CC
9	101	2021-02-01	DD

Merchant Client

Purchaser ID 목록?

SELECT UNIQUE {Purchaser ID}

FROM {Table}

WHERE {Merchant ID} = 1

Purchaser Client

Merchant ID 목록?

SELECT UNIQUE {Merchant ID}

FROM {Table}

WHERE {Purchaser ID} = 101

Data Modeling Problem on Sharded Cluster

적절한 Shard Key 선정의 어려움

Shard key

<Sharded DB Cluster>

Shard 1

Merchant ID	Purchaser ID	Order Date	Order Msg
1	100	2021-01-01	null
1	101	2021-01-02	AA

Shard 2

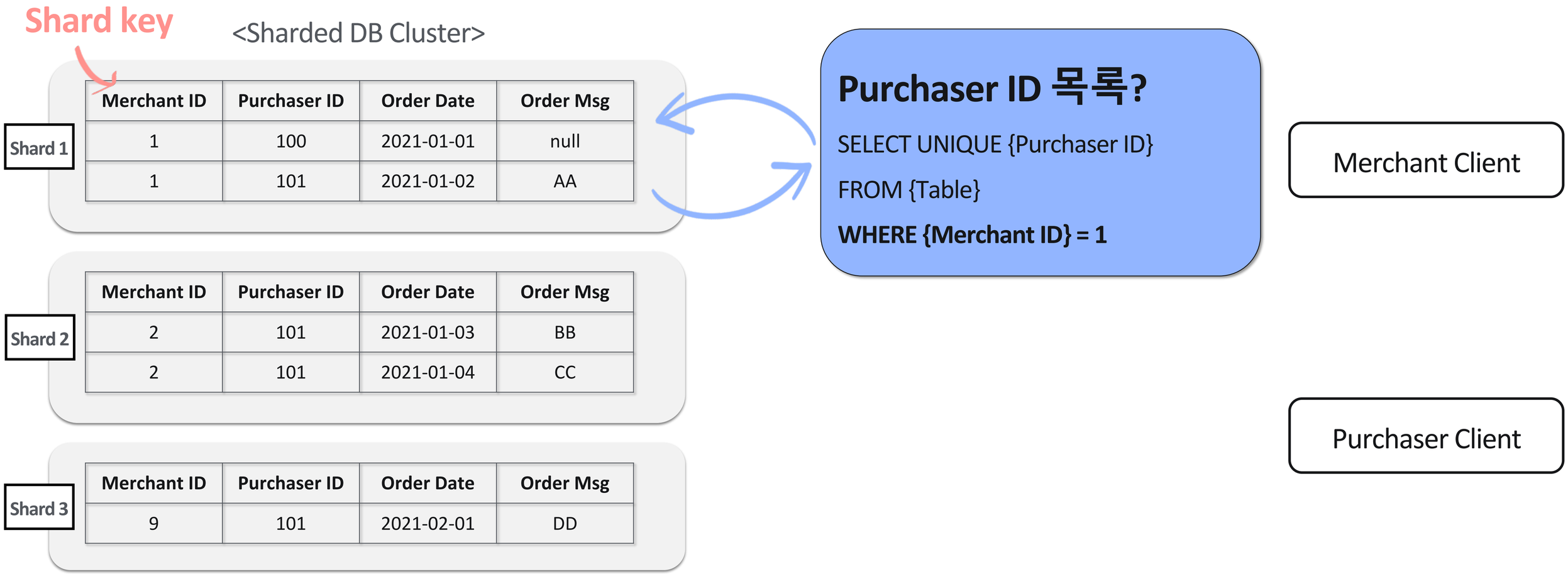
Merchant ID	Purchaser ID	Order Date	Order Msg
2	101	2021-01-03	BB
2	101	2021-01-04	CC

Shard 3

Merchant ID	Purchaser ID	Order Date	Order Msg
9	101	2021-02-01	DD

Data Modeling Problem on Sharded Cluster

적절한 Shard Key 선정의 어려움



Data Modeling Problem on Sharded Cluster

DEVIEW 2021

적절한 Shard Key 선정의 어려움

Shard key

<Sharded DB Cluster>

Shard 1

Merchant ID	Purchaser ID	Order Date	Order Msg
1	100	2021-01-01	null
1	101	2021-01-02	AA

Shard 2

Merchant ID	Purchaser ID	Order Date	Order Msg
2	101	2021-01-03	BB
2	101	2021-01-04	CC

Shard 3

Merchant ID	Purchaser ID	Order Date	Order Msg
9	101	2021-02-01	DD

Merchant Client

Purchaser Client

Merchant ID 목록?

SELECT UNIQUE {Merchant ID}

FROM {Table}

WHERE {Purchaser ID} = 101

Data Modeling Problem on Sharded Cluster

DEVIEW 2021

적절한 Shard Key 선정의 어려움

Shard key

<Sharded DB Cluster>

Shard 1

Merchant ID	Purchaser ID	Order Date	Order Msg
1	100	2021-01-01	null
1	101	2021-01-02	AA

Shard 2

Merchant ID	Purchaser ID	Order Date	Order Msg
2	101	2021-01-03	BB
2	101	2021-01-04	CC

Shard 3

Merchant ID	Purchaser ID	Order Date	Order Msg
9	101	2021-02-01	DD

Merchant Client

Purchaser Client

Merchant ID 목록?

SELECT UNIQUE {Merchant ID}

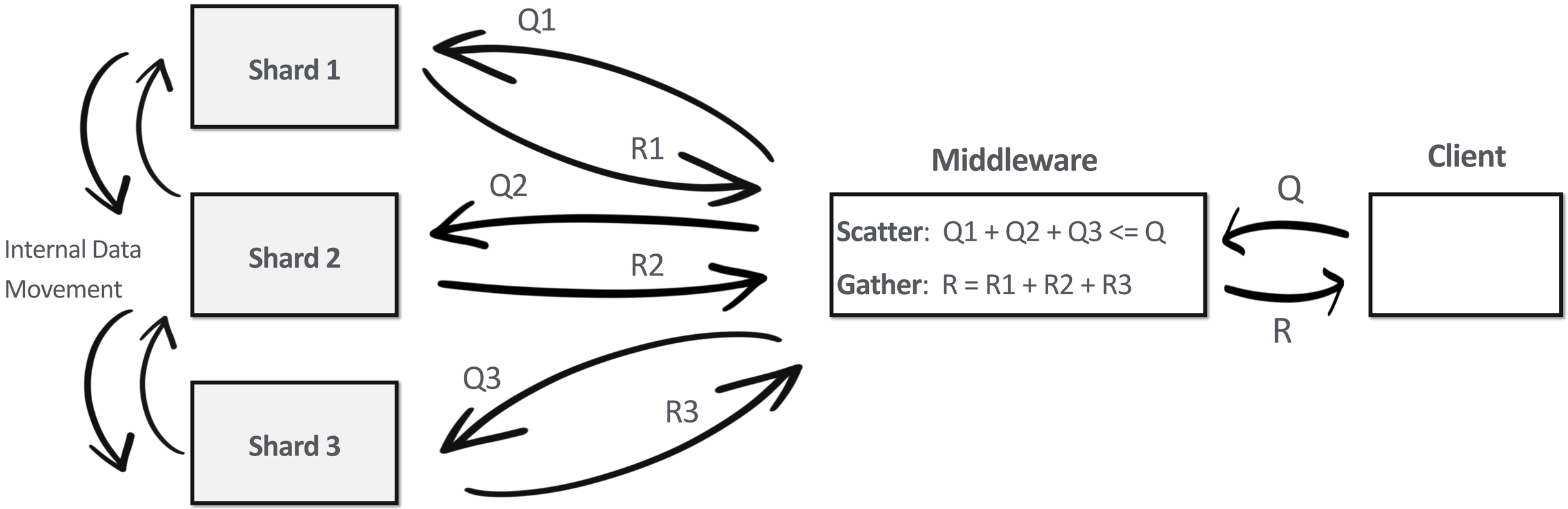
FROM {Table}

WHERE {Purchaser ID} = 101

Data Modeling Problem on Sharded Cluster

Problem: 여러 shard 간 query? *or* **shard key 다른 table 간의 Join?**

Solution: Cross-Shard Query / Cross-Shard Join



Data Modeling Problem on Sharded Cluster

Problem: 여러 shard 간 query? *or* **shard key 다른 table 간의 Join?**

Solution: Cross-Shard Query / Cross-Shard Join

- 단점

> Read 시 전체 Shard 에 쿼리를 수행하고 결과를 취합해야 하므로 느림

> 하나의 쿼리가 많은 internal query 들을 유발

> Join 까지 하는 경우 Shard 간 Internal Data Movement 도 존재

> Workload 가 **Read 가 많은 패턴일 경우 성능상 병목 지점**이 될 확률이 높음

Data Modeling Problem on Sharded Cluster

Another Solution: Dual-Write 기반 Materialized View

- Materialized View?

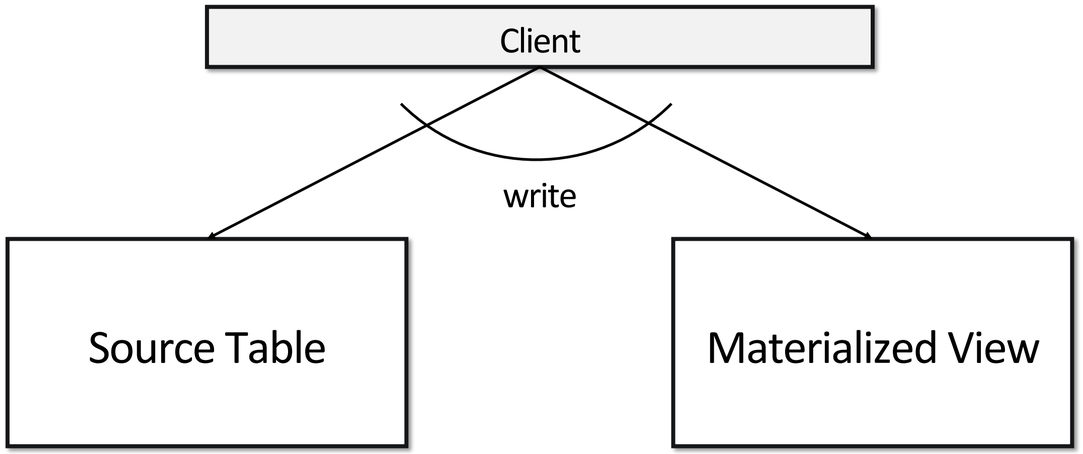
> 쿼리의 결과를 물리적으로 저장하고 있는 database object

> SUM, MIN, MAX, AVG, COUNT 등의 Aggregate Query 를 주로 사용

- Idea

> Source Table 의 Shard Key 를 다르게 하여 Materialized View 로 유지한다면?

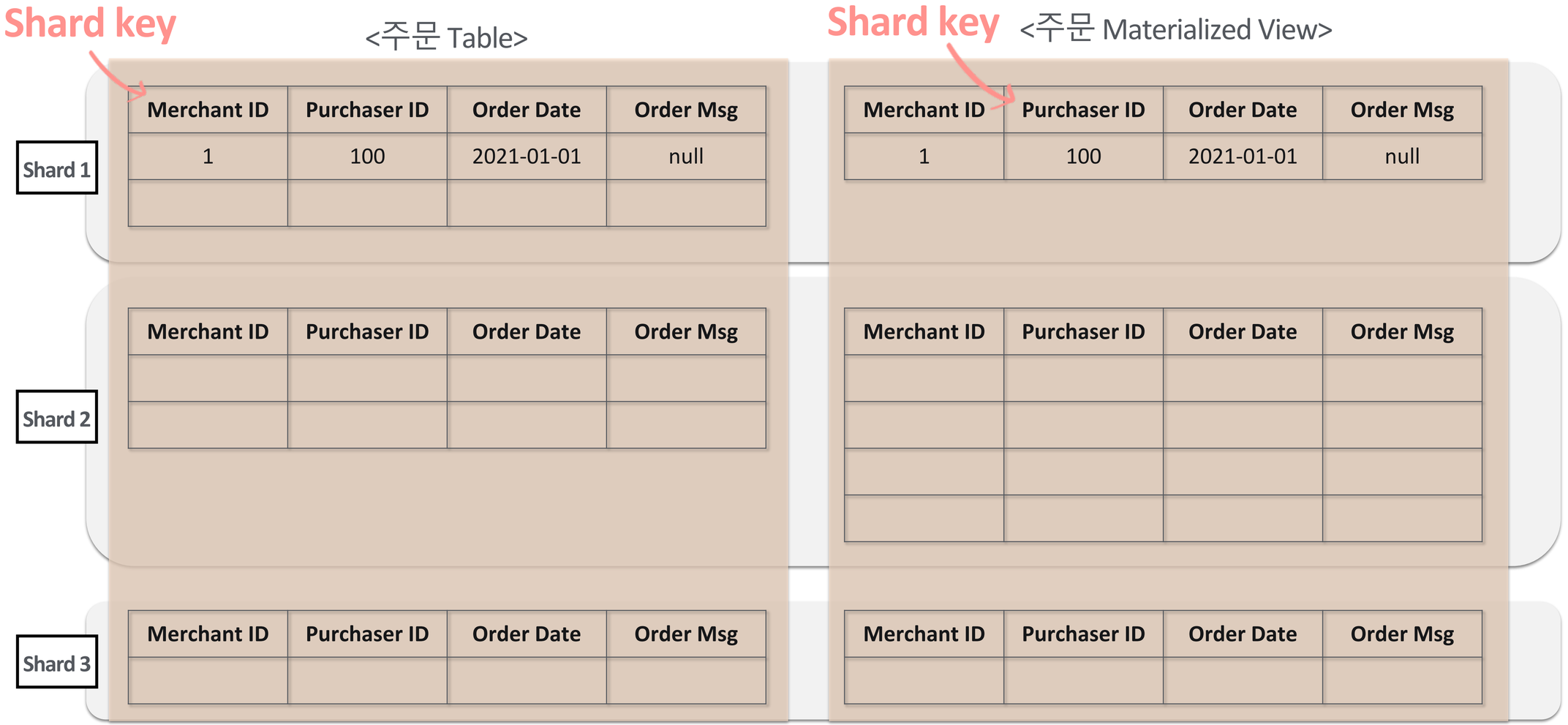
> Source Table 에 write 시, materialized view 에도 write (dual-write)



Data Modeling Problem on Sharded Cluster

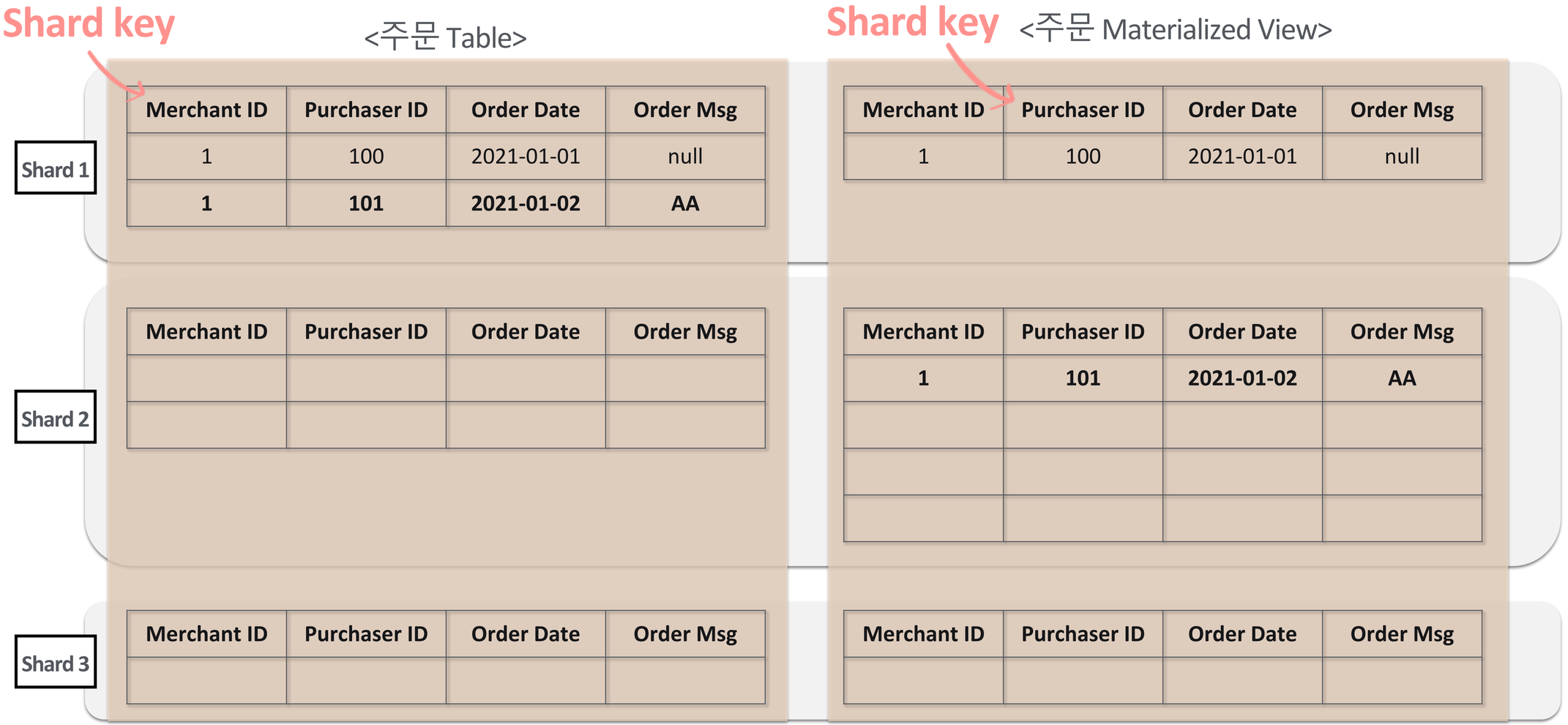
DEVIEW 2021

Another Solution: Dual-Write 기반 Materialized View



Data Modeling Problem on Sharded Cluster

Another Solution: Dual-Write 기반 Materialized View



Data Modeling Problem on Sharded Cluster

Another Solution: Dual-Write 기반 Materialized View

Shard key

<주문 Table>

Shard key

<주문 Materialized View>

Shard 1

Merchant ID	Purchaser ID	Order Date	Order Msg
1	100	2021-01-01	null
1	101	2021-01-02	AA

Merchant ID	Purchaser ID	Order Date	Order Msg
1	100	2021-01-01	null

Shard 2

Merchant ID	Purchaser ID	Order Date	Order Msg
2	**101**	**2021-01-03**	**BB**
2	**101**	**2021-01-04**	**CC**

Merchant ID	Purchaser ID	Order Date	Order Msg
1	101	2021-01-02	AA
2	**101**	**2021-01-03**	**BB**
2	**101**	**2021-01-04**	**CC**

Shard 3

Merchant ID	Purchaser ID	Order Date	Order Msg

Merchant ID	Purchaser ID	Order Date	Order Msg

Data Modeling Problem on Sharded Cluster

Another Solution: Dual-Write 기반 Materialized View

Shard key

<주문 Table>

Shard 1

Merchant ID	Purchaser ID	Order Date	Order Msg
1	100	2021-01-01	null
1	101	2021-01-02	AA

Shard 2

Merchant ID	Purchaser ID	Order Date	Order Msg
2	101	2021-01-03	BB
2	101	2021-01-04	CC

Shard 3

Merchant ID	Purchaser ID	Order Date	Order Msg
9	**101**	**2021-02-01**	**DD**

Shard key

<주문 Materialized View>

Merchant ID	Purchaser ID	Order Date	Order Msg
1	100	2021-01-01	null
1	101	2021-01-02	AA
2	101	2021-01-03	BB
2	101	2021-01-04	CC
9	**101**	**2021-02-01**	**DD**

Merchant ID	Purchaser ID	Order Date	Order Msg

Data Modeling Problem on Sharded Cluster

Another Solution: Dual-Write 기반 Materialized View

- 장점

> Read 시 전체 Shard 에 쿼리하지 않아도 됨 => Read 성능 개선 및 병목 해소

- 단점

> Multiple Shard 에 Write Transaction => Distributed Transaction 지원이 필요함

> 2PC-based Distributed Tx is not scalable

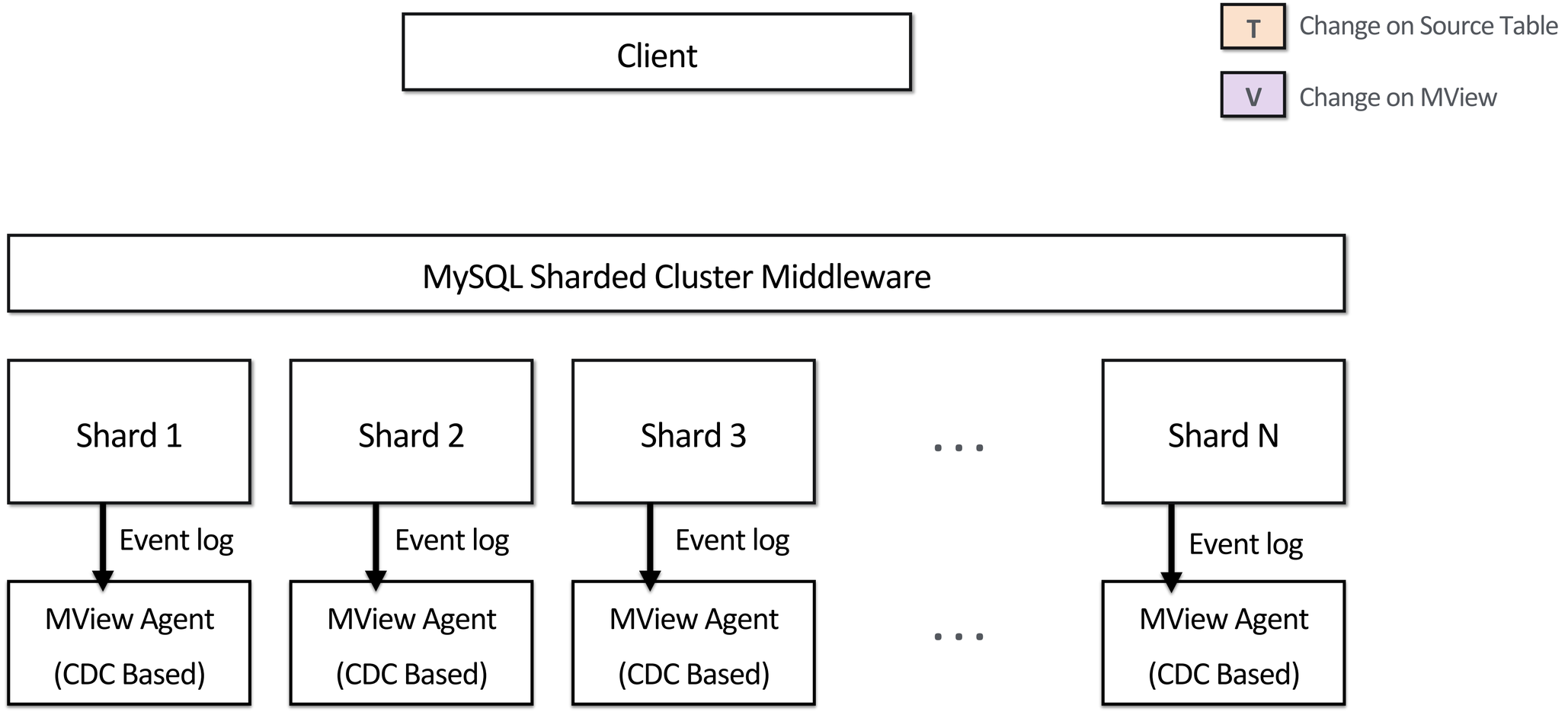
> Deadlock detection, Compensation Query 등 복잡한 추가 처리가 필요

> Data Inconsistency 가 발생했을 때 원인이나 보정이 어려움

CDC-based Materialized View

Our Approach: Log-based CDC 기반 Materialized View

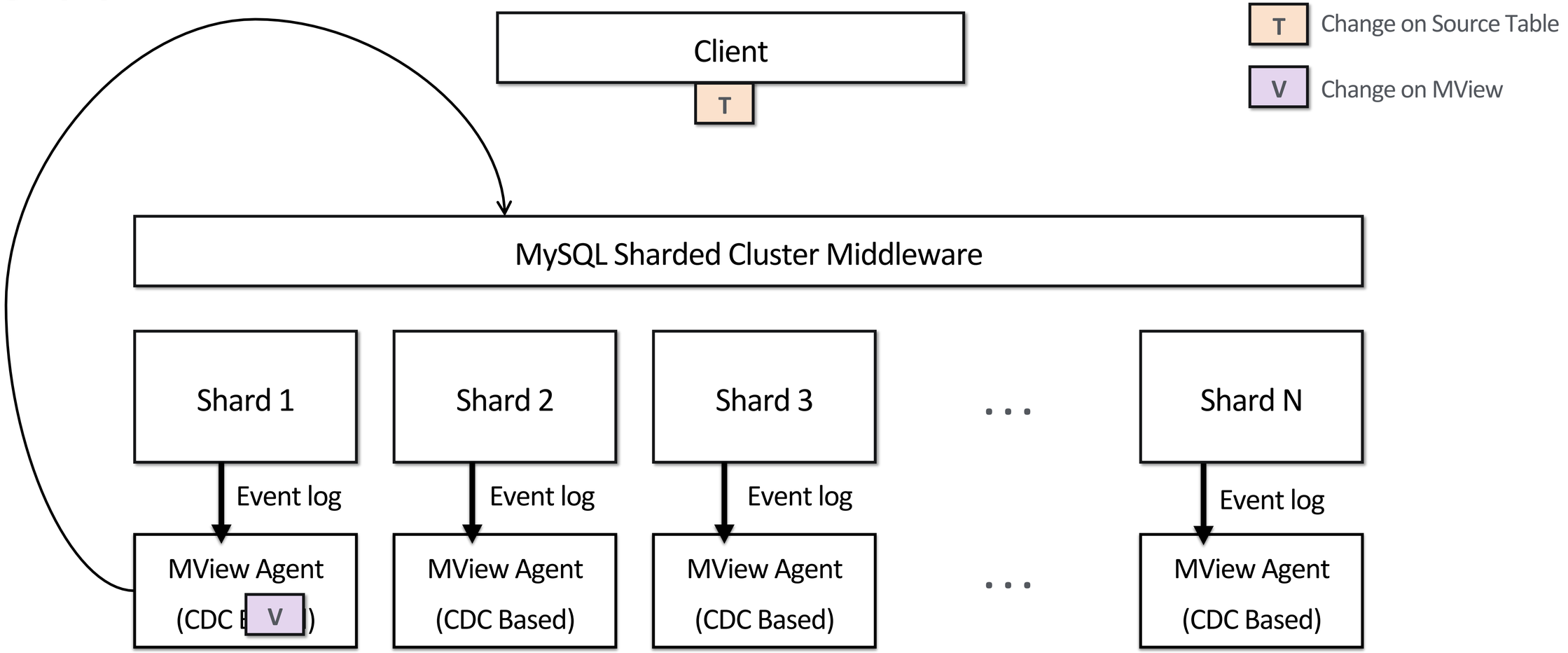
- 동작 구조



CDC-based Materialized View

Our Approach: Log-based CDC 기반 Materialized View

- 동작 구조



CDC-based Materialized View

DEVIEW 2021

Our Approach: Log-based CDC 기반 Materialized View

- 스키마 구조

> 칼럼에 대한 정의 뿐만 아니라 source table 과의 상관관계를 rule 형태로 정의

```
CREATE TABLE `order` (
    `merchant`   INT(11) NOT NULL,
    `purchaser`  INT(11) NOT NULL,
    `order_date` DATE,
    `order_msg`  VARCHAR(200),
    SHARD KEY (`merchant`)
    PRIMARY KEY (`purchaser`)
)
```

```
CREATE VIEW `v_order` (
    `v_merchant`   INT(11) NOT NULL,
    `v_purchaser`  INT(11) NOT NULL,
    `v_order_date` DATE,
    `v_order_msg`  VARCHAR(200),
    SHARD KEY (`v_purchaser`)
    PRIMARY KEY (`v_merchant`)
) AS (
    SELECT `merchant`, `purchaser`, `order_date`, `order_msg`
    FROM `order`
    SHARD KEY `purchaser`
)
```

AS Clause

- Source Table 과 관계 정의

CDC-based Materialized View

DEVIEW 2021

Our Approach: Log-based CDC 기반 Materialized View

- 장점

  > **No cross-shard query, join**: read-heavy workload 에 최적화

  > **No distributed tx required**: data-inconsistency 회피

  > Eventual consistency (최종적 일관성) 만족

  > One source - Multiple view 구조 용이

- 단점

  > 물리적인 저장 공간이 추가로 필요 (max. 2배)

  > No guarantee for timeliness

Online Materialized View Creation

DEVIEW 2021

Problem: Online Materialized View 생성

- 데이터가 들어있는 테이블에 Materialized View 생성

- **추가적으로 고려할 사항들**
  - > **Online DB**: 원본 테이블은 zero read/write downtime
  - > **Side Effect 최소화**: 서비스 중인 다른 View 반영 지연 최소화
  - > **운영 편의성**: 운영자에 의해 멈추거나 재개될 수 있어야함

Online Materialized View Creation

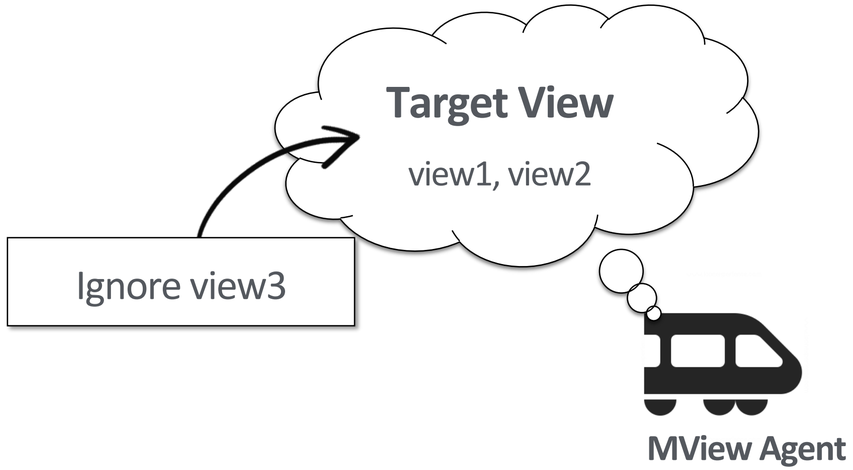
Our Approach: 3 Phase Creation

- 생성 절차를 3 Phase 로 나누어 수행

Online Materialized View Creation

Our Approach: 3 Phase Creation

- **예시) Create mview ‘view3’ from source table ‘table3’** (view 1, view 2: in service)
- **생성 작업 직전 상황**

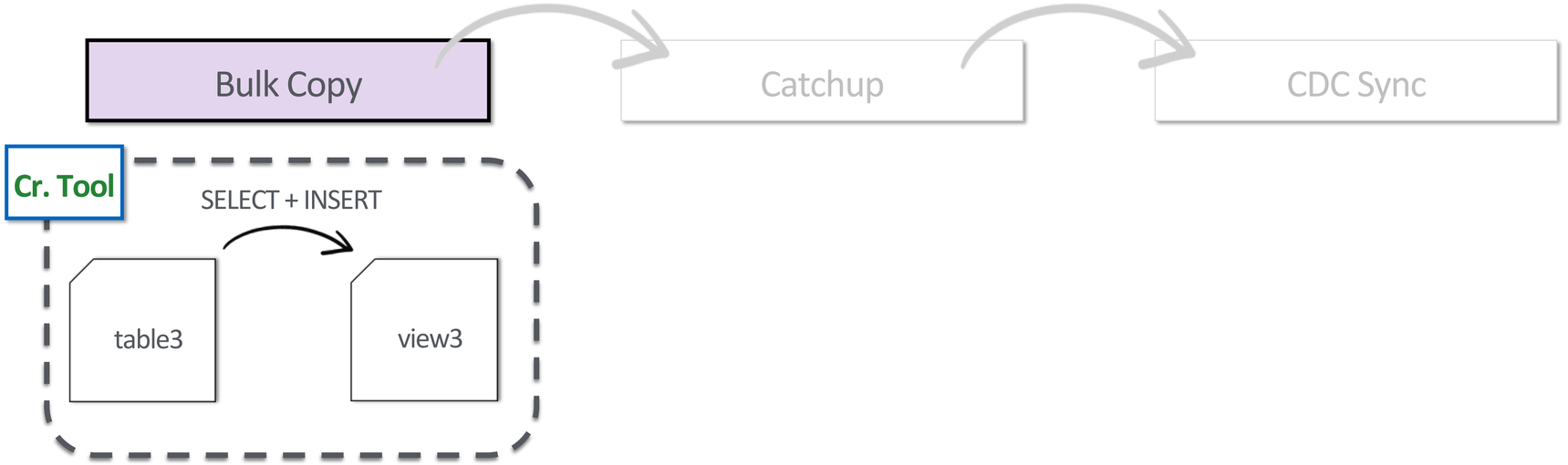


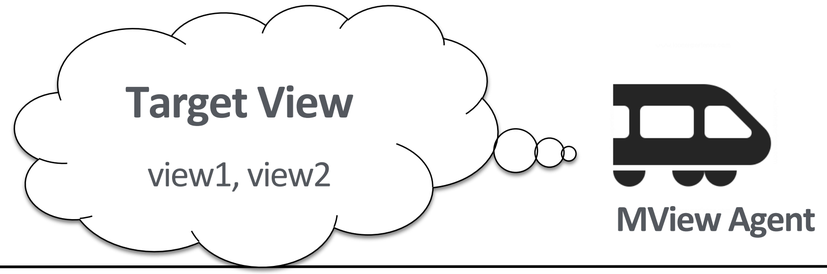
Oldest — Change Stream (=binlog) — Latest

Online Materialized View Creation

Our Approach: 3 Phase Creation

- 예시) Create mview ‘view3’ from source table ‘table3’
- Phase 1



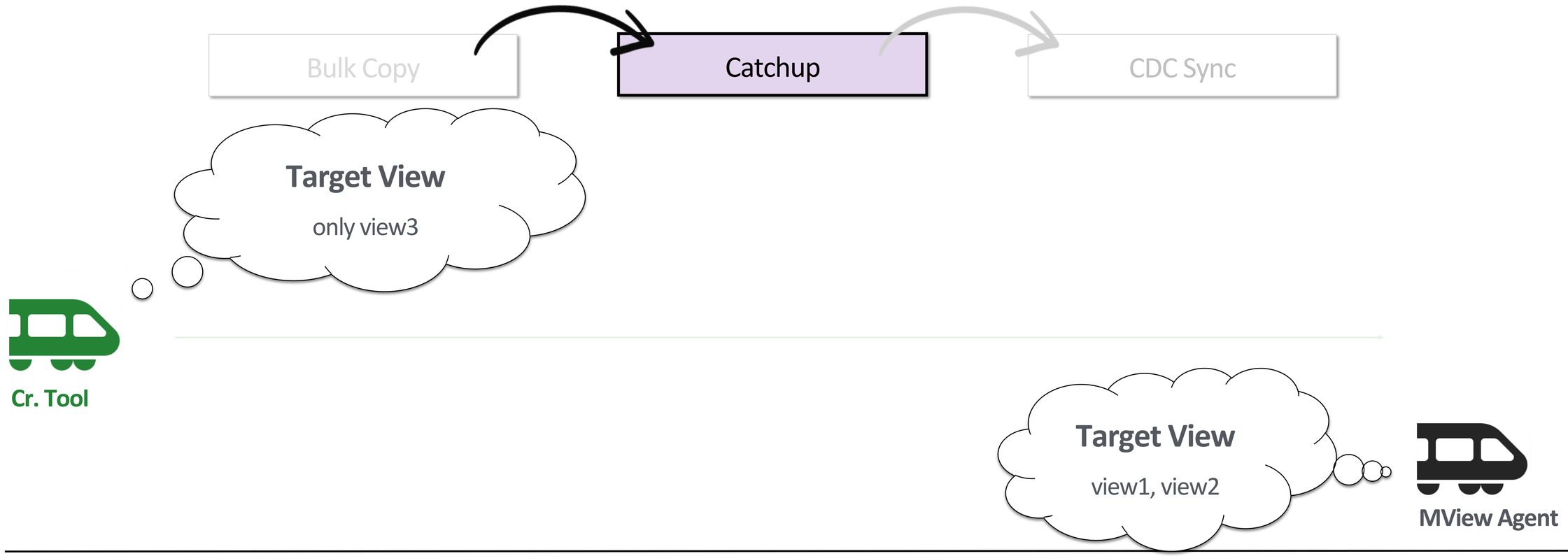


Oldest — Change Stream (=binlog) — Latest

Online Materialized View Creation

Our Approach: 3 Phase Creation

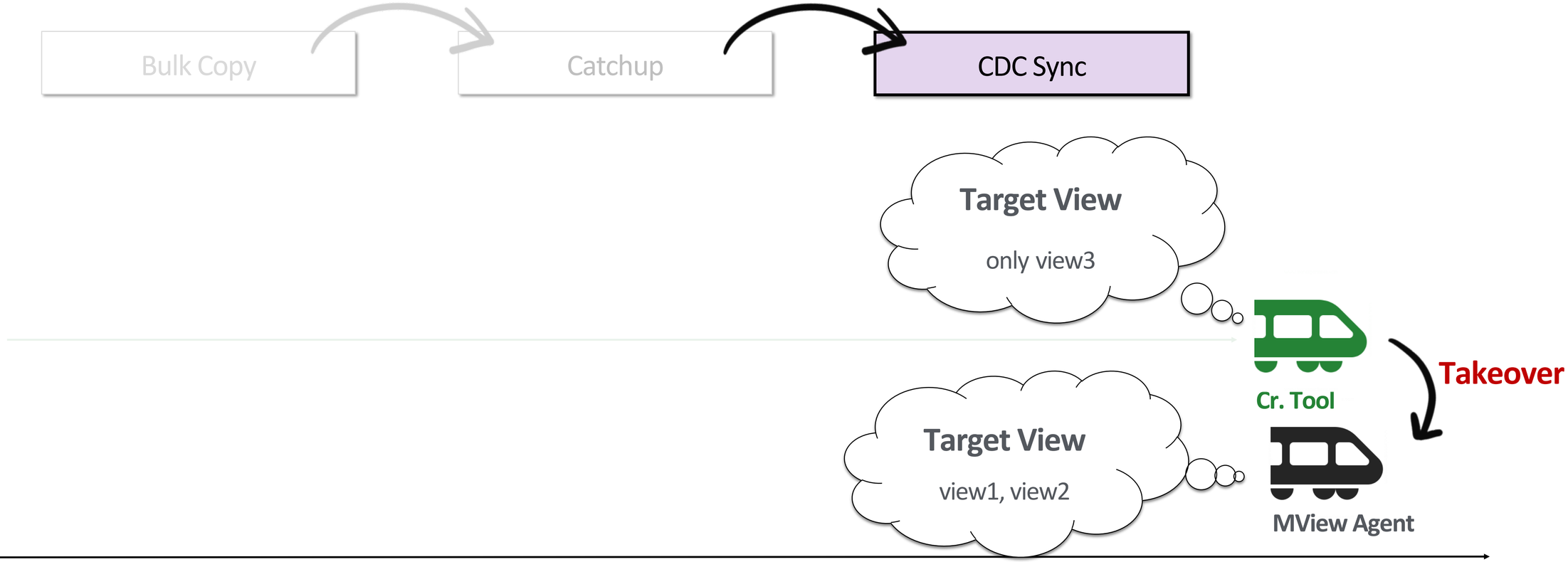
- 예시) Create mview ‘view3’ from source table ‘table3’
- Phase 2: 변경 사항 따라잡기



Online Materialized View Creation

Our Approach: 3 Phase Creation

- 예시) Create mview ‘view3’ from source table ‘table3’
- Phase 3: view 반영 책임을 agent 로 이전



Online Materialized View Creation

Our Approach: 3 Phase Creation

- 예시) Create mview ‘view3’ from source table ‘table3’
- Phase 3: view 반영 책임을 agent 로 이전

Bulk Copy

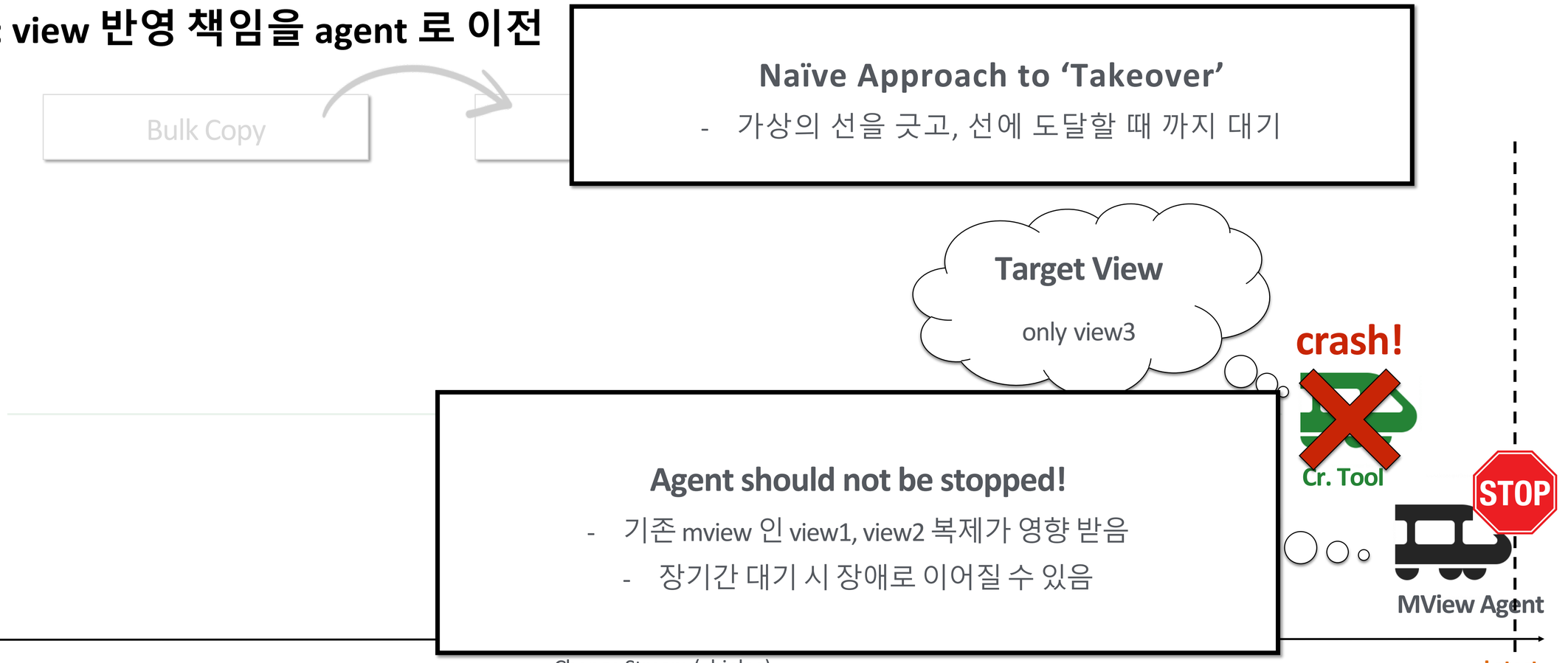
Naïve Approach to ‘Takeover’

- 가상의 선을 긋고, 선에 도달할 때 까지 대기

Online Materialized View Creation

Our Approach: 3 Phase Creation

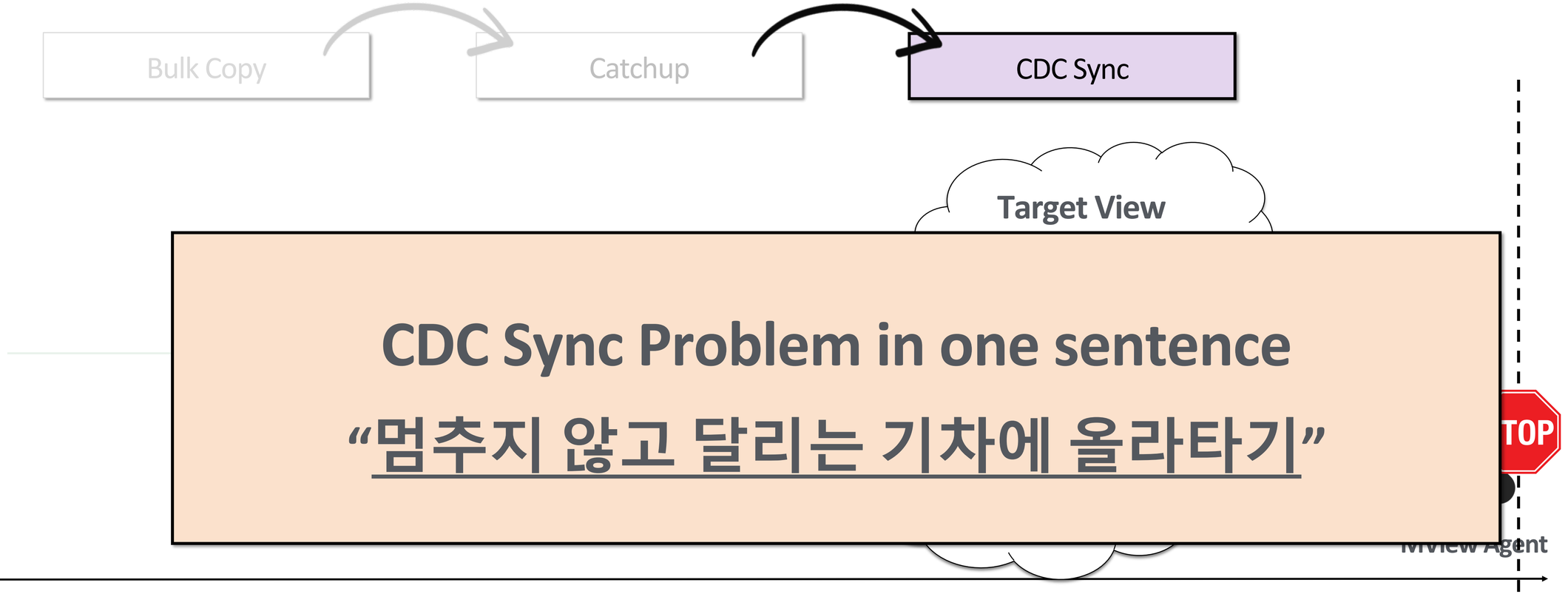
- 예시) Create mview ‘view3’ from source table ‘table3’
- Phase 3: view 반영 책임을 agent 로 이전



Online Materialized View Creation

Our Approach: 3 Phase Creation

- 예시) Create mview ‘view3’ from source table ‘table3’
- Phase 3: view 반영 책임을 agent 로 이전



Online Materialized View Creation

DEVIEW 2021

CDC Sync (“달리는 기차에 올라타는 법”)

- Solution: Change Stream 에 Special Event 를 발행하여 구분

Oldest

Change Stream (=binlog)

Latest

Online Materialized View Creation

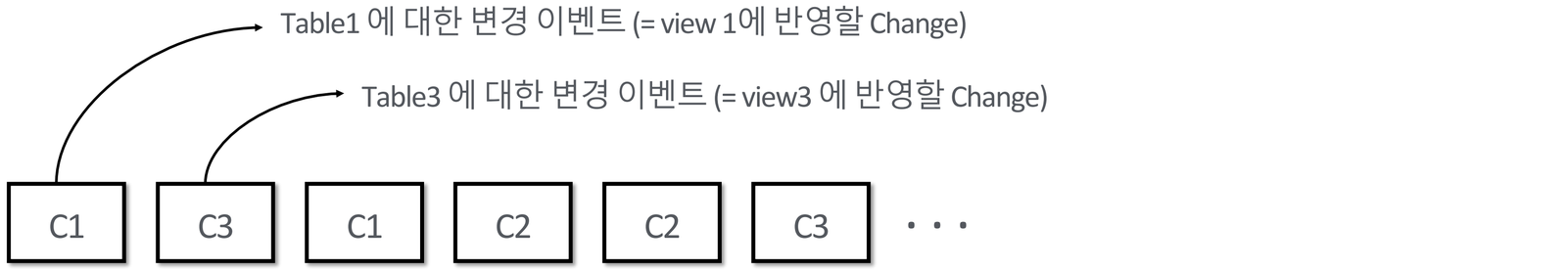
CDC Sync (“달리는 기차에 올라타는 법”)

- **Solution: Change Stream 에 Special Event 를 발행하여 구분**

* Creation Tool

Step 1: READY 이벤트 DB에 발행

Cr. Tool



Online Materialized View Creation

CDC Sync (“달리는 기차에 올라타는 법”)

- Solution: Change Stream 에 Special Event 를 발행하여 구분

* Creation Tool

Step 1: READY 이벤트 DB에 발행

Cr. Tool

Latest - n sec

Change Stream (=binlog)

Latest

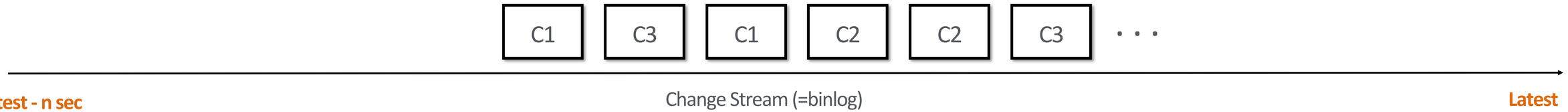
Online Materialized View Creation

CDC Sync (“달리는 기차에 올라타는 법”)

- Solution: Change Stream 에 Special Event 를 발행하여 구분

* Creation Tool

Step 1: READY 이벤트 DB에 발행



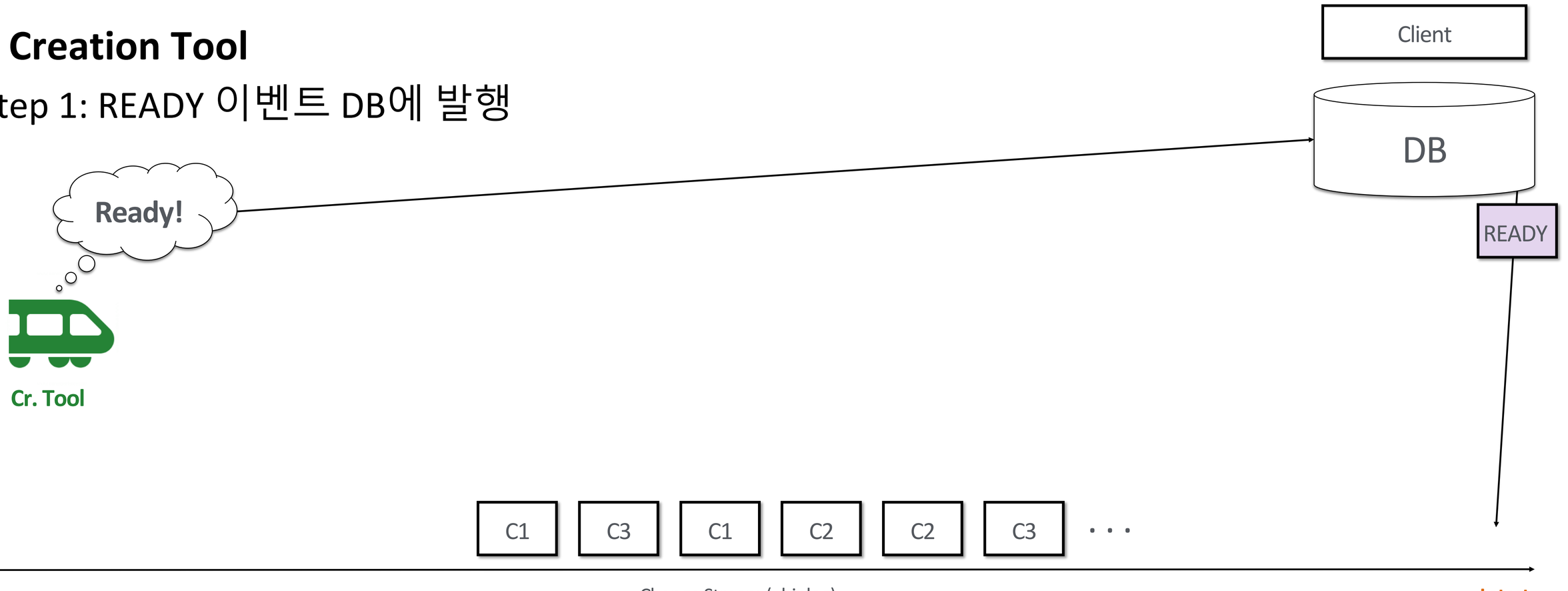
Online Materialized View Creation

CDC Sync (“달리는 기차에 올라타는 법”)

- Solution: Change Stream 에 Special Event 를 발행하여 구분

* Creation Tool

Step 1: READY 이벤트 DB에 발행



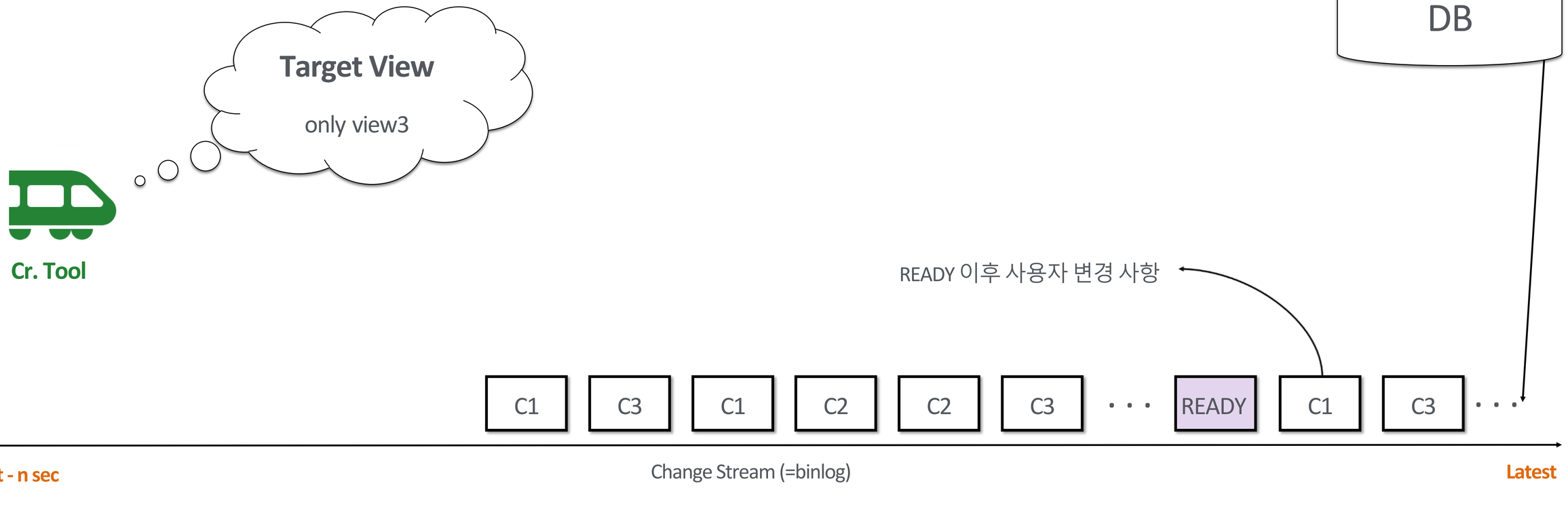
Online Materialized View Creation

CDC Sync (“달리는 기차에 올라타는 법”)

- **Solution: Change Stream 에 Special Event 를 발행하여 구분**

* Creation Tool

Step 2: READY 발견할 때까지는 계속 반영



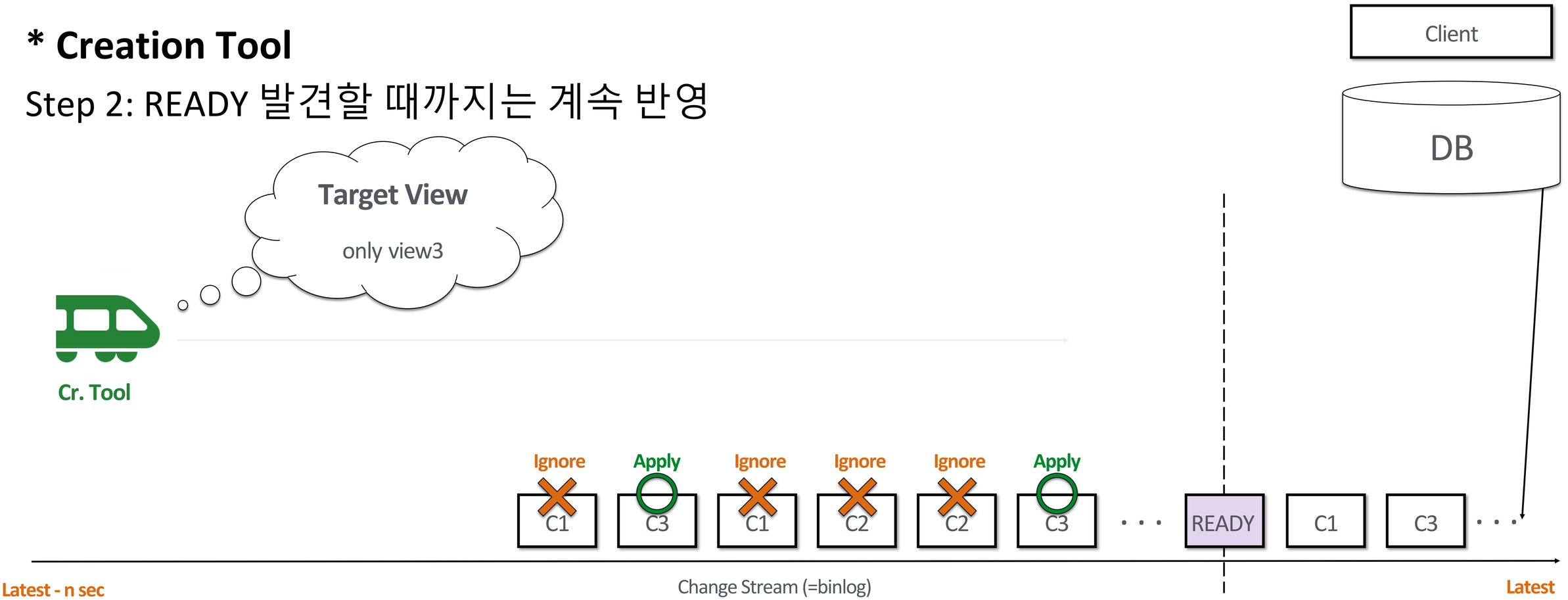
Online Materialized View Creation

CDC Sync (“달리는 기차에 올라타는 법”)

- Solution: Change Stream 에 Special Event 를 발행하여 구분

* Creation Tool

Step 2: READY 발견할 때까지는 계속 반영



Online Materialized View Creation

CDC Sync (“달리는 기차에 올라타는 법”)

- **Solution: Change Stream 에 Special Event 를 발행하여 구분**

* Creation Tool

Step 3: READY 까지 반영 완료를 알리는 SET 발행 후 종료

Online Materialized View Creation

CDC Sync (“달리는 기차에 올라타는 법”)

- **Solution: Change Stream 에 Special Event 를 발행하여 구분**

* Creation Tool

Step 3: READY 까지 반영 완료를 알리는 SET 발행 후 종료

Online Materialized View Creation

CDC Sync (“달리는 기차에 올라타는 법”)

- **Solution: Change Stream 에 Special Event 를 발행하여 구분**

* Creation Tool

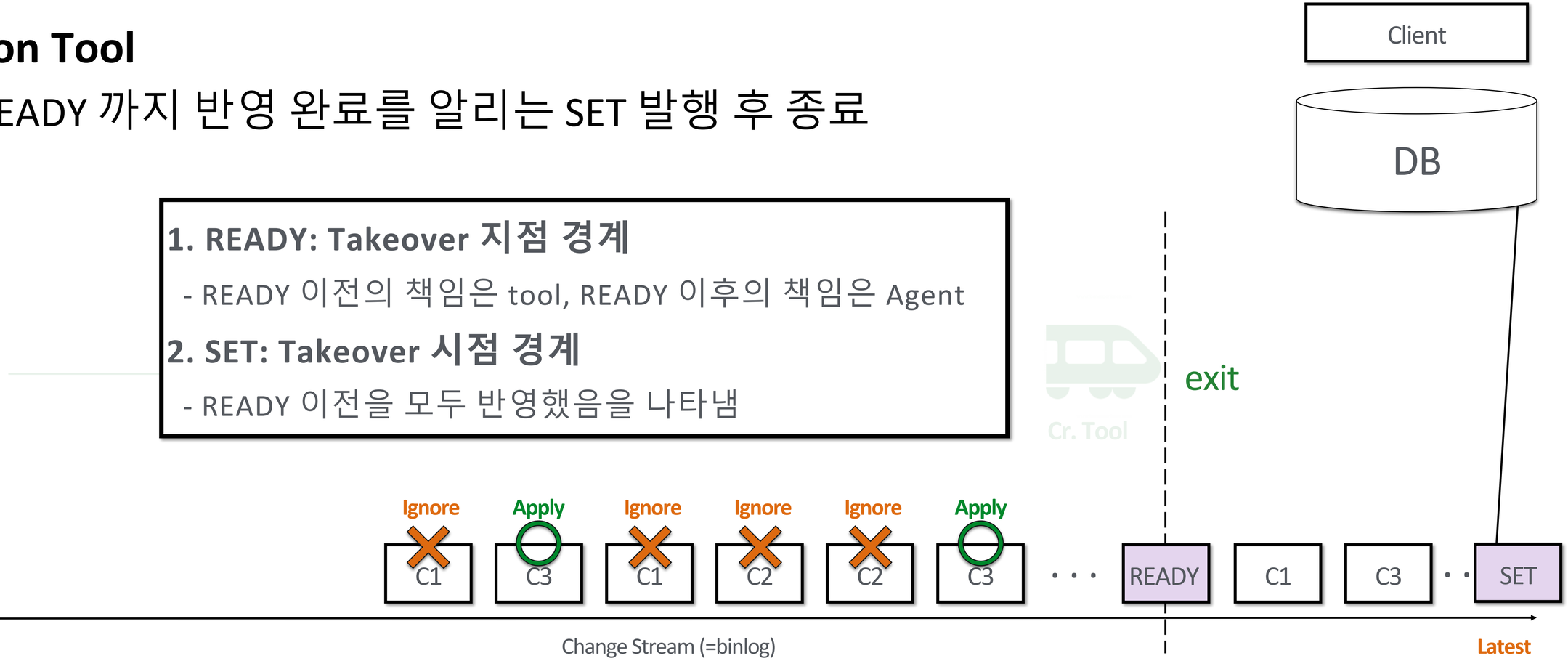
Step 3: READY 까지 반영 완료를 알리는 SET 발행 후 종료

1. READY: Takeover 지점 경계

- READY 이전의 책임은 tool, READY 이후의 책임은 Agent

2. SET: Takeover 시점 경계

- READY 이전을 모두 반영했음을 나타냄



Online Materialized View Creation

CDC Sync (“달리는 기차에 올라타는 법”)

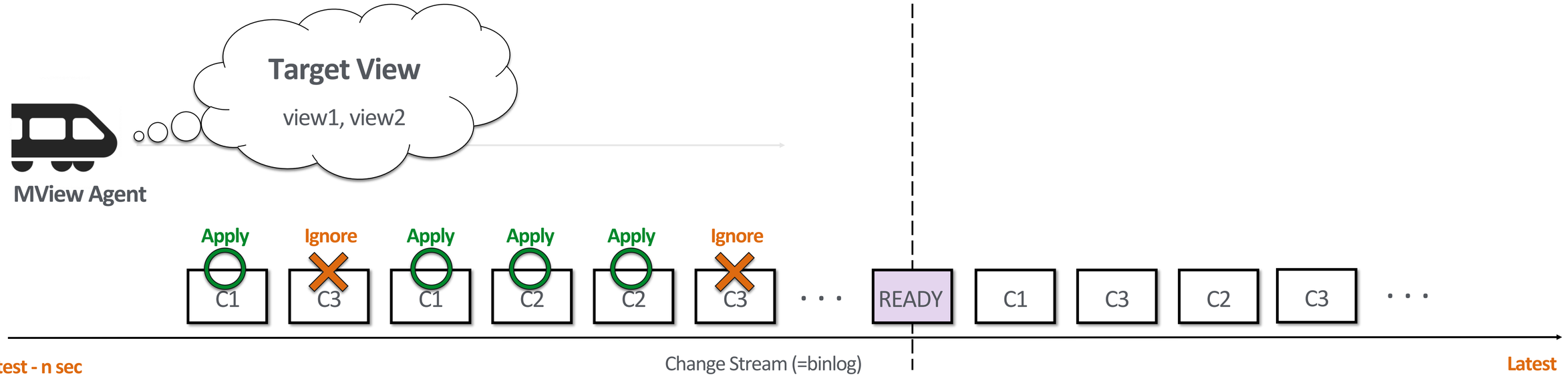
- **Solution: Change Stream 에 Special Event 를 발행하여 구분**

* MView Agent

> Step1: READY 발견 전까지 신규 view 반영 사항 무시

> Step2: SET 발견 전까지 신규 view 반영 Pending 처리

> Step3: SET 발견 후 Pending 처리된 변경 사항 처리 후 신규 view 반영 시작



Online Materialized View Creation

CDC Sync (“달리는 기차에 올라타는 법”)

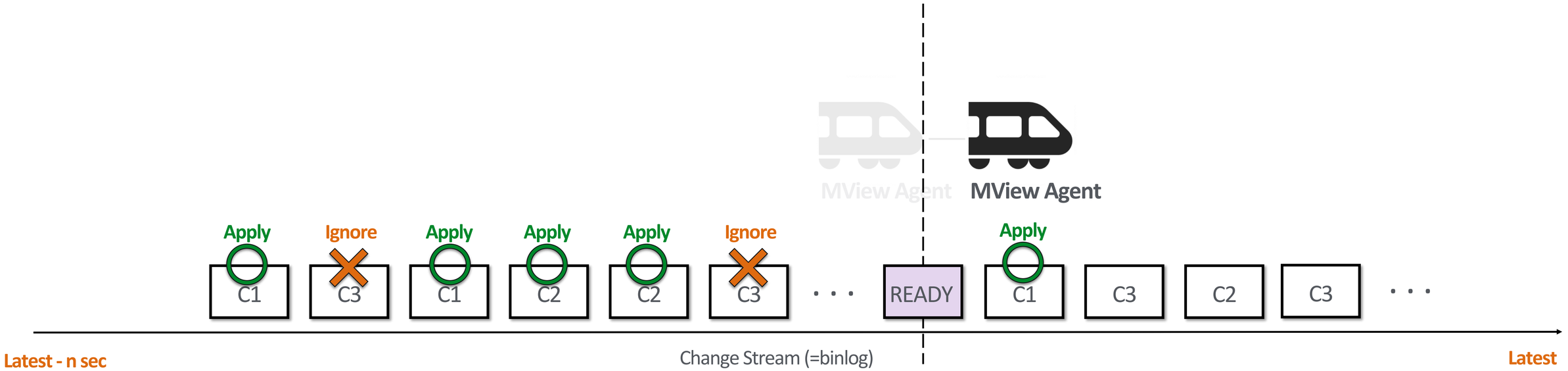
- Solution: Change Stream 에 Special Event 를 발행하여 구분

* MView Agent

> Step1: READY 발견 전까지 신규 view 반영 사항 무시

> Step2: SET 발견 전까지 신규 view 반영 Pending 처리

> Step3: SET 발견 후 Pending 처리된 변경 사항 처리 후 신규 view 반영 시작



Online Materialized View Creation

CDC Sync (“달리는 기차에 올라타는 법”)

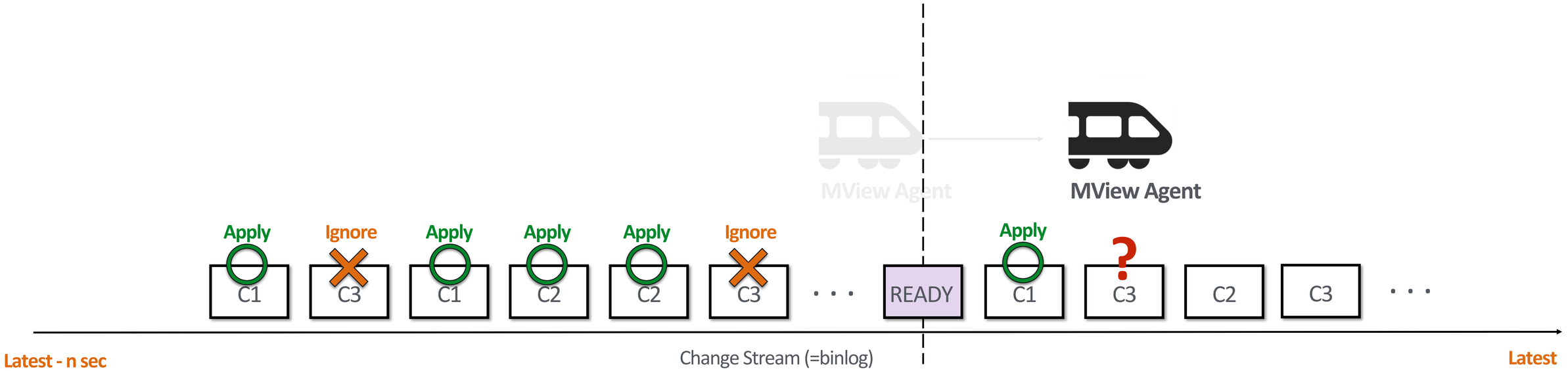
- Solution: Change Stream 에 Special Event 를 발행하여 구분

* MView Agent

> Step1: READY 발견 전까지 신규 view 반영 사항 무시

> Step2: SET 발견 전까지 신규 view 반영 Pending 처리

> Step3: SET 발견 후 Pending 처리된 변경 사항 처리 후 신규 view 반영 시작



Online Materialized View Creation

CDC Sync (“달리는 기차에 올라타는 법”)

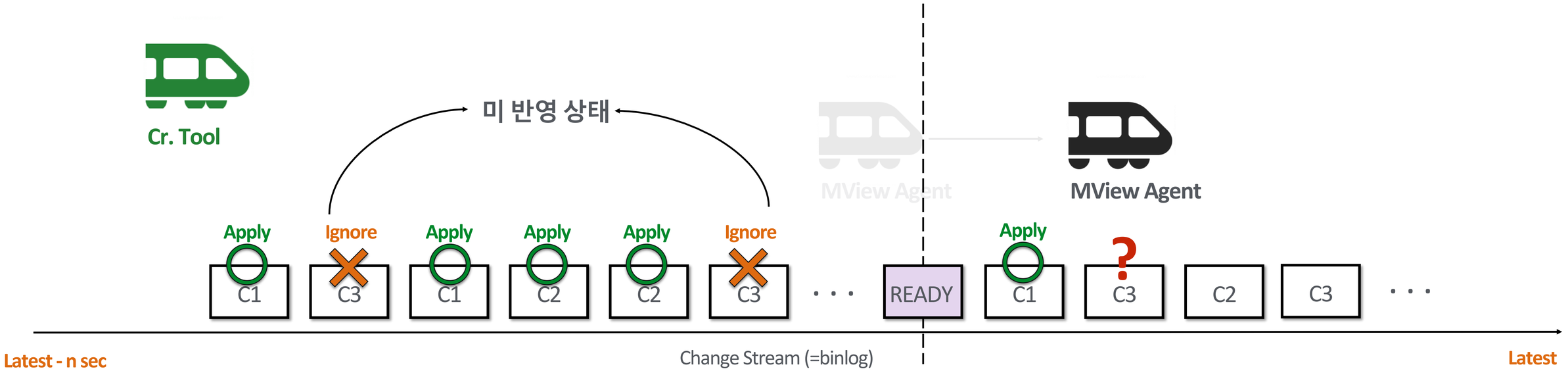
- Solution: Change Stream 에 Special Event 를 발행하여 구분

* MView Agent

> Step1: READY 발견 전까지 신규 view 반영 사항 무시

> Step2: SET 발견 전까지 신규 view 반영 Pending 처리

> Step3: SET 발견 후 Pending 처리된 변경 사항 처리 후 신규 view 반영 시작



Online Materialized View Creation

CDC Sync (“달리는 기차에 올라타는 법”)

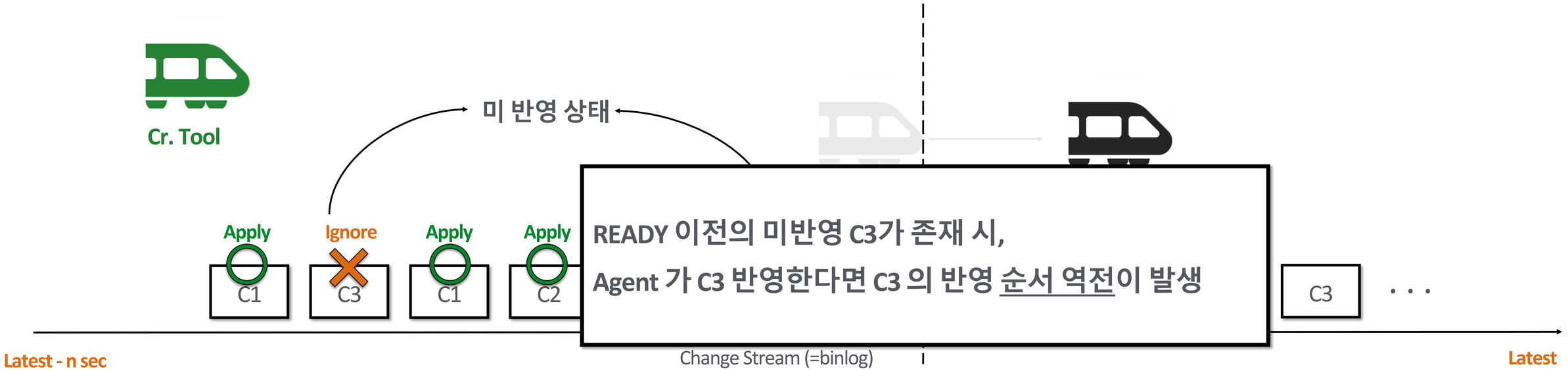
- Solution: Change Stream 에 Special Event 를 발행하여 구분

* MView Agent

> Step1: READY 발견 전까지 신규 view 반영 사항 무시

> Step2: SET 발견 전까지 신규 view 반영 Pending 처리

> Step3: SET 발견 후 Pending 처리된 변경 사항 처리 후 신규 view 반영 시작



Online Materialized View Creation

CDC Sync (“달리는 기차에 올라타는 법”)

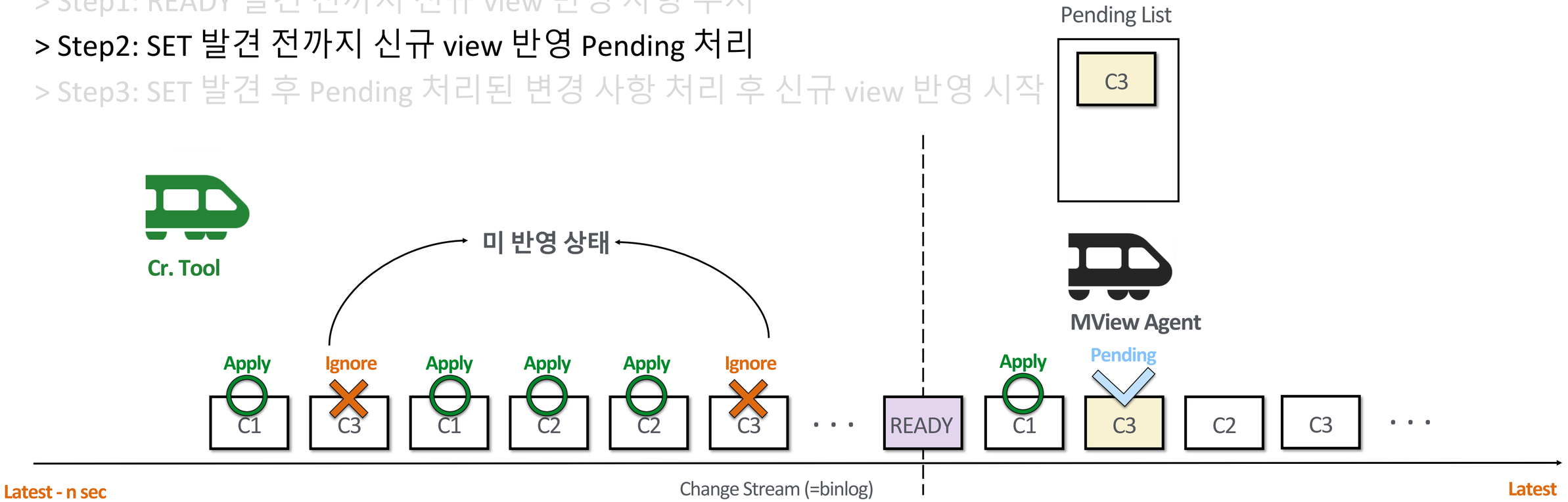
- Solution: Change Stream 에 Special Event 를 발행하여 구분

* MView Agent

> Step1: READY 발견 전까지 신규 view 반영 사항 무시

> Step2: SET 발견 전까지 신규 view 반영 Pending 처리

> Step3: SET 발견 후 Pending 처리된 변경 사항 처리 후 신규 view 반영 시작



Online Materialized View Creation

CDC Sync (“달리는 기차에 올라타는 법”)

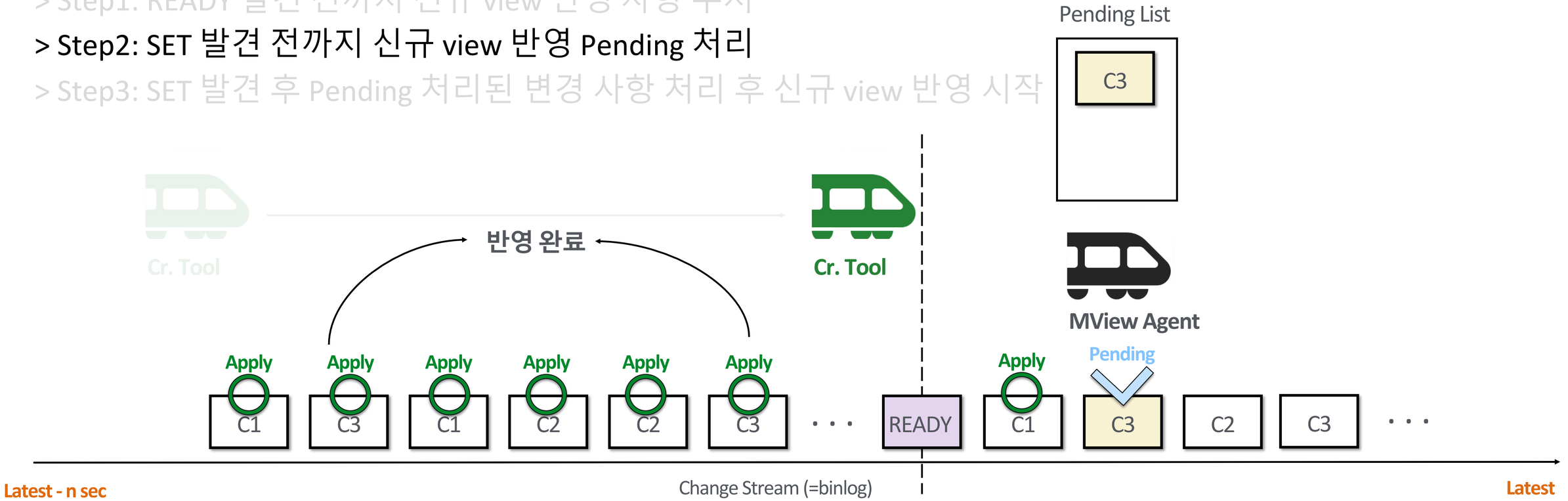
- Solution: Change Stream 에 Special Event 를 발행하여 구분

* MView Agent

> Step1: READY 발견 전까지 신규 view 반영 사항 무시

> Step2: SET 발견 전까지 신규 view 반영 Pending 처리

> Step3: SET 발견 후 Pending 처리된 변경 사항 처리 후 신규 view 반영 시작



Online Materialized View Creation

CDC Sync (“달리는 기차에 올라타는 법”)

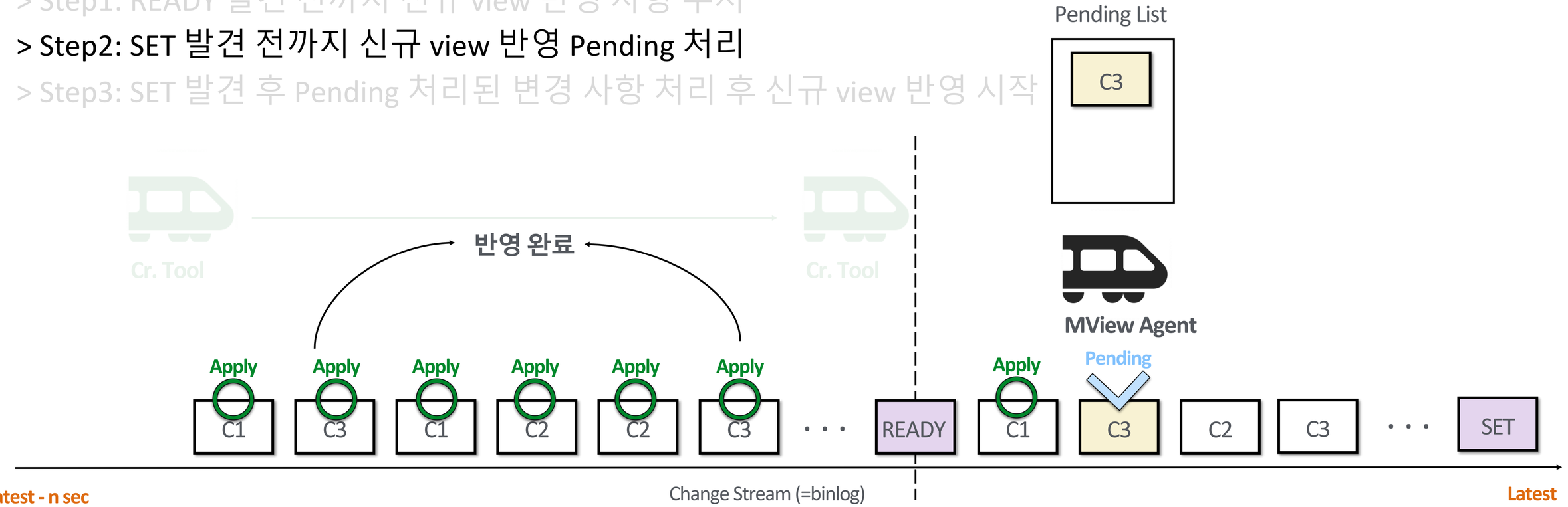
- Solution: Change Stream 에 Special Event 를 발행하여 구분

* MView Agent

> Step1: READY 발견 전까지 신규 view 반영 사항 무시

> Step2: SET 발견 전까지 신규 view 반영 Pending 처리

> Step3: SET 발견 후 Pending 처리된 변경 사항 처리 후 신규 view 반영 시작



Online Materialized View Creation

CDC Sync (“달리는 기차에 올라타는 법”)

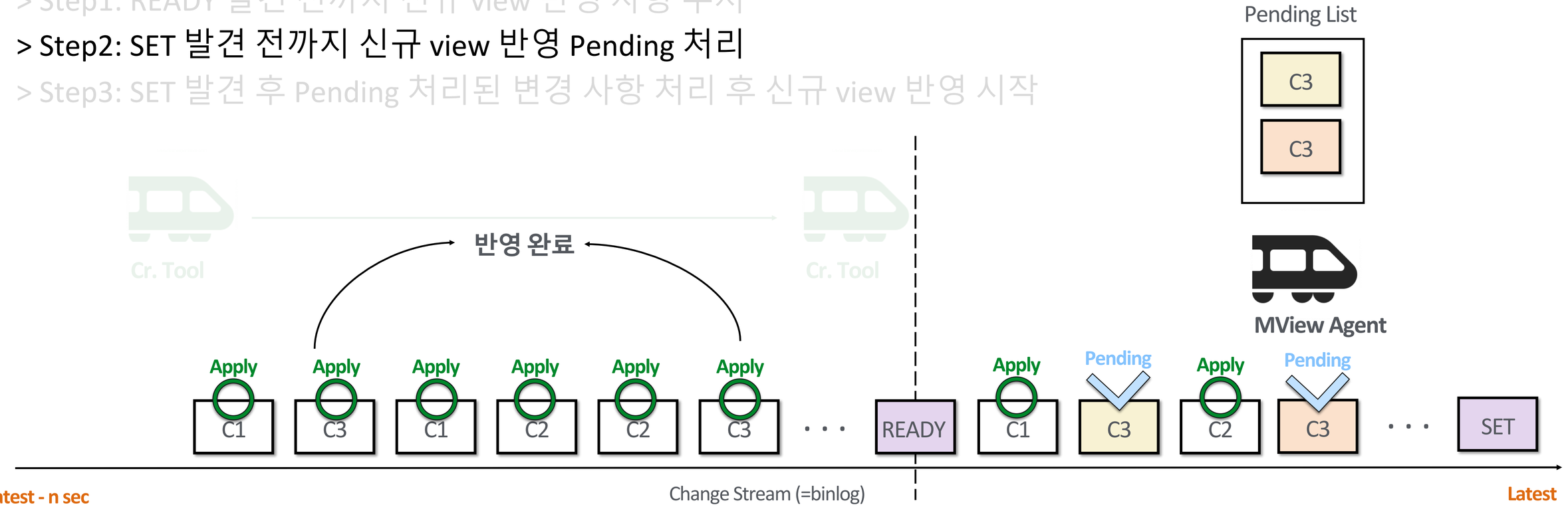
- Solution: Change Stream 에 Special Event 를 발행하여 구분

* MView Agent

> Step1: READY 발견 전까지 신규 view 반영 사항 무시

> Step2: SET 발견 전까지 신규 view 반영 Pending 처리

> Step3: SET 발견 후 Pending 처리된 변경 사항 처리 후 신규 view 반영 시작



Online Materialized View Creation

CDC Sync (“달리는 기차에 올라타는 법”)

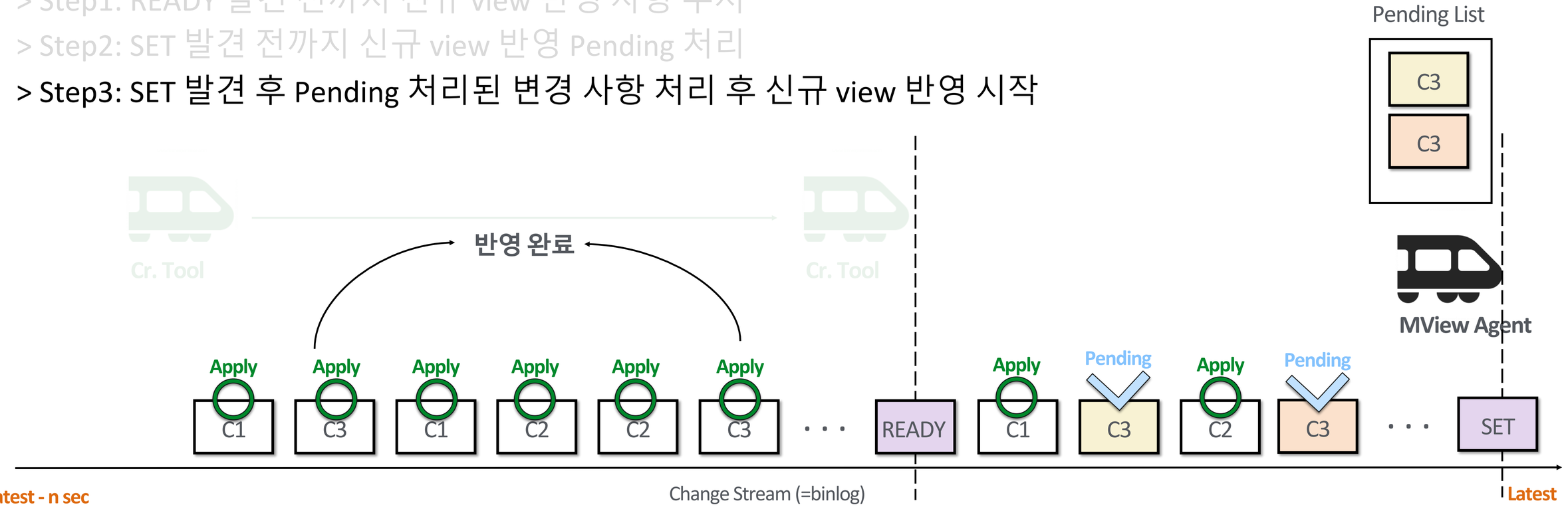
- Solution: Change Stream 에 Special Event 를 발행하여 구분

* MView Agent

> Step1: READY 발견 전까지 신규 view 반영 사항 무시

> Step2: SET 발견 전까지 신규 view 반영 Pending 처리

> Step3: SET 발견 후 Pending 처리된 변경 사항 처리 후 신규 view 반영 시작



Online Materialized View Creation

CDC Sync (“달리는 기차에 올라타는 법”)

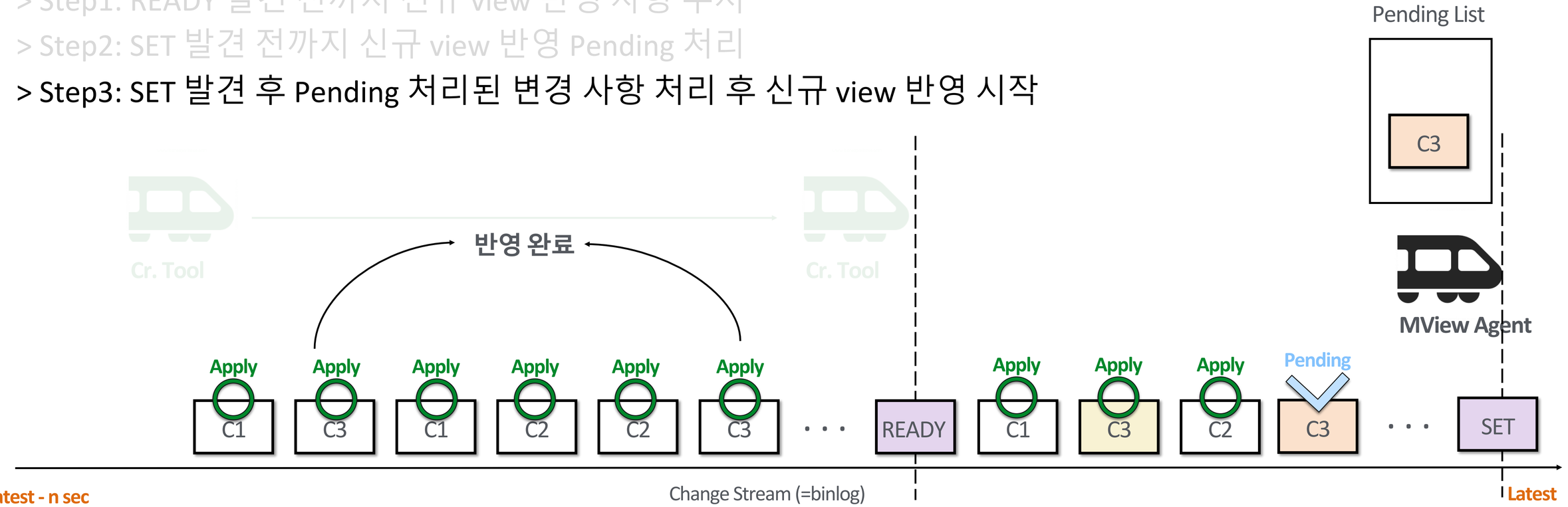
- Solution: Change Stream 에 Special Event 를 발행하여 구분

* MView Agent

> Step1: READY 발견 전까지 신규 view 반영 사항 무시

> Step2: SET 발견 전까지 신규 view 반영 Pending 처리

> Step3: SET 발견 후 Pending 처리된 변경 사항 처리 후 신규 view 반영 시작



Online Materialized View Creation

CDC Sync (“달리는 기차에 올라타는 법”)

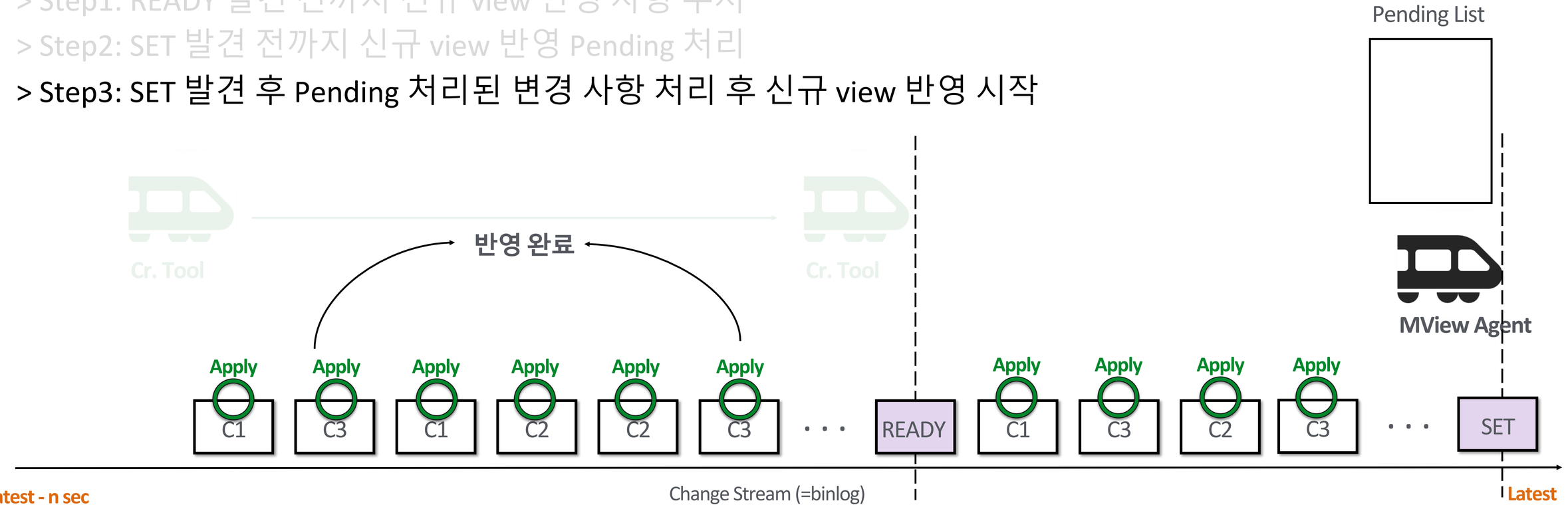
- Solution: Change Stream 에 Special Event 를 발행하여 구분

* MView Agent

> Step1: READY 발견 전까지 신규 view 반영 사항 무시

> Step2: SET 발견 전까지 신규 view 반영 Pending 처리

> Step3: SET 발견 후 Pending 처리된 변경 사항 처리 후 신규 view 반영 시작



Online Materialized View Creation

CDC Sync (“달리는 기차에 올라타는 법”)

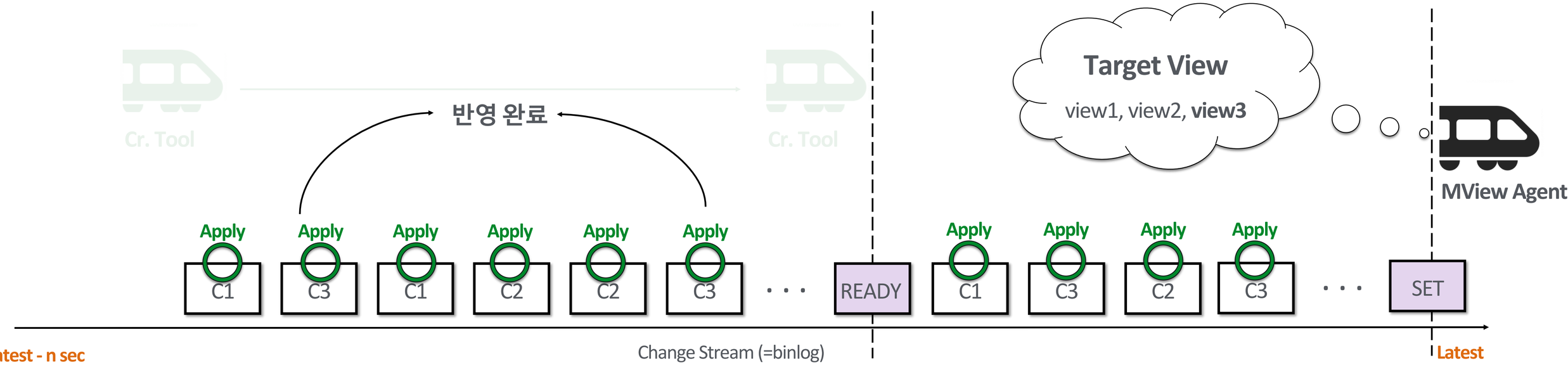
- Solution: Change Stream 에 Special Event 를 발행하여 구분

* MView Agent

> Step1: READY 발견 전까지 신규 view 반영 사항 무시

> Step2: SET 발견 전까지 신규 view 반영 Pending 처리

> Step3: SET 발견 후 Pending 처리된 변경 사항 처리 후 신규 view 반영 시작



Online Materialized View Creation

DEVIEW 2021

CDC Sync (“달리는 기차에 올라타는 법”)

- **Solution: Change Stream 에 Special Event 를 발행하여 구분**

Production System 에 적용된 추가 고려 사항들

> Creation Tool 이 READY 만 발행 후 종료될 경우?

> MView Agent 내 In-Memory Pending List 가 지나치게 커질 경우를 대비한 takeover cancel

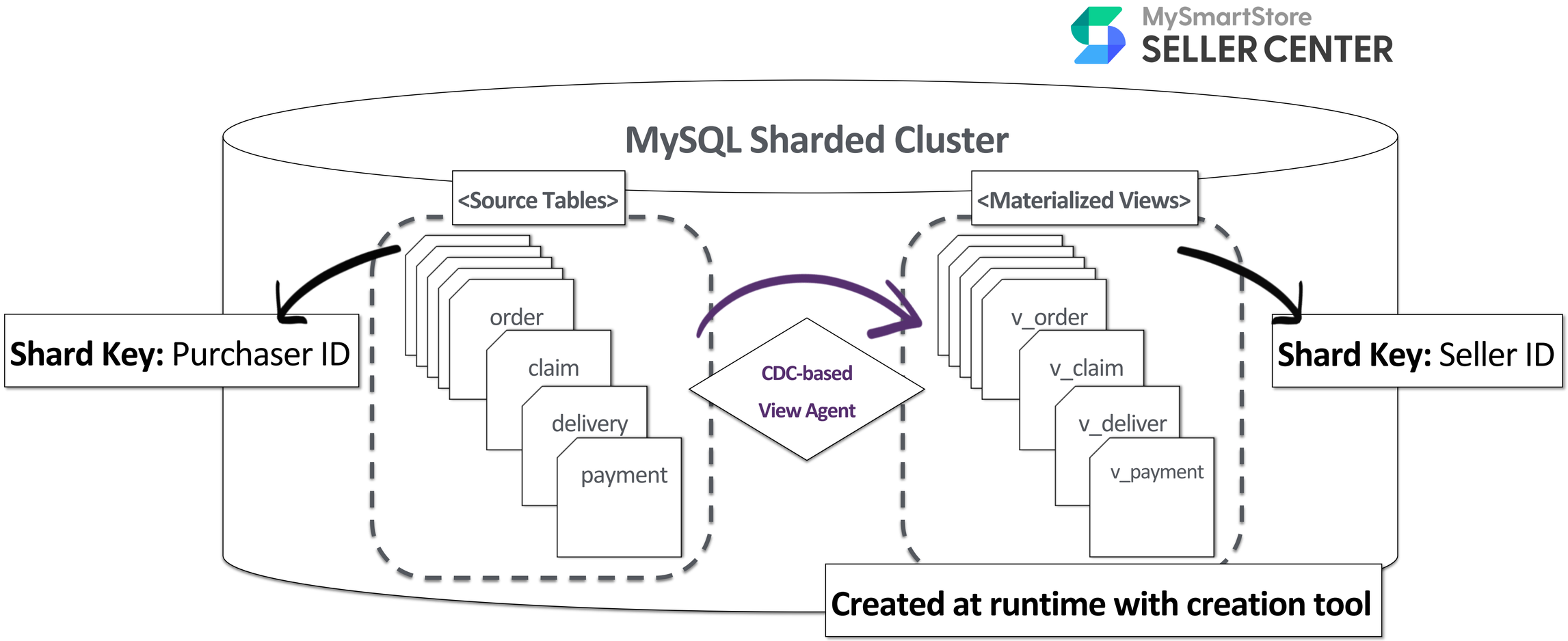
> Sharded Cluster 내 여러 Shard 간에 MView 생성 상태 control 및 조회

> 추가 모니터링 도구들

Materialized View @JP MySmartStore

DEVIEW 2021

Overview

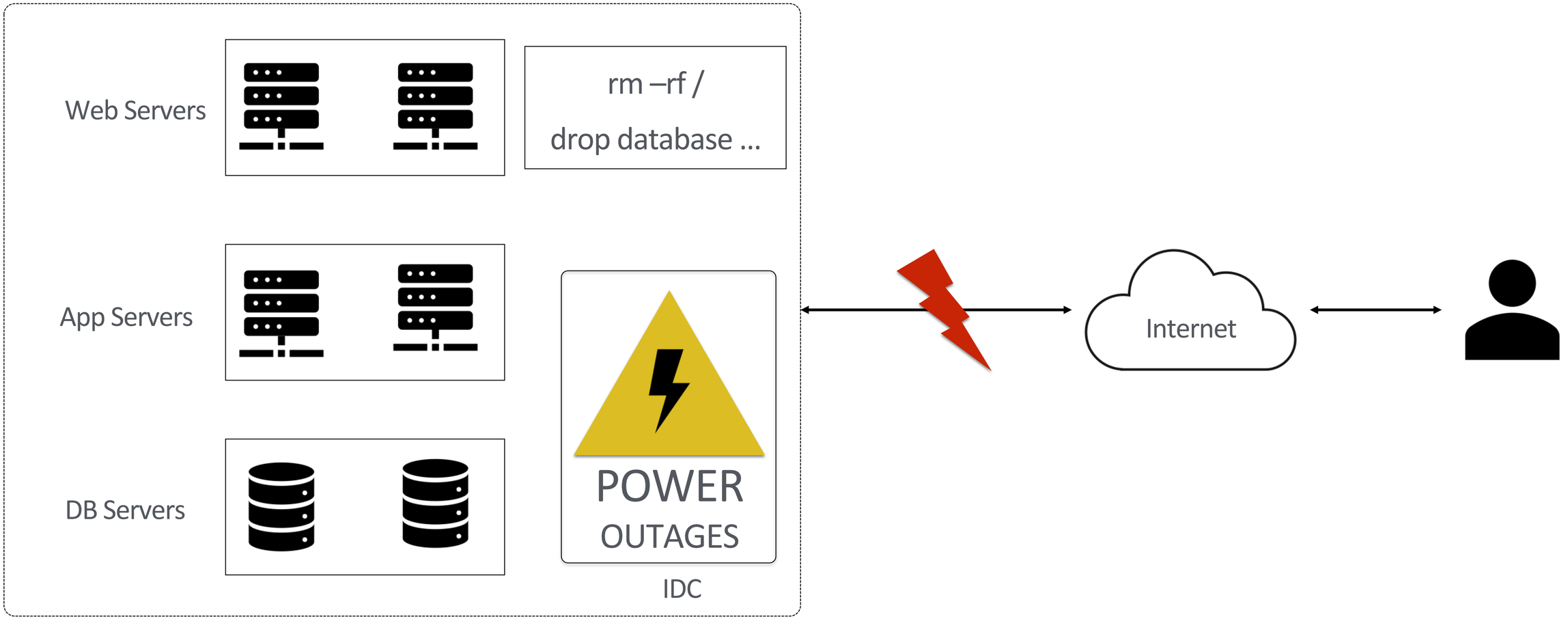


DEVIEW 2021

CDC를 활용한 재해복구 데이터 복제

What is Disaster Recovery?

재해 복구란?

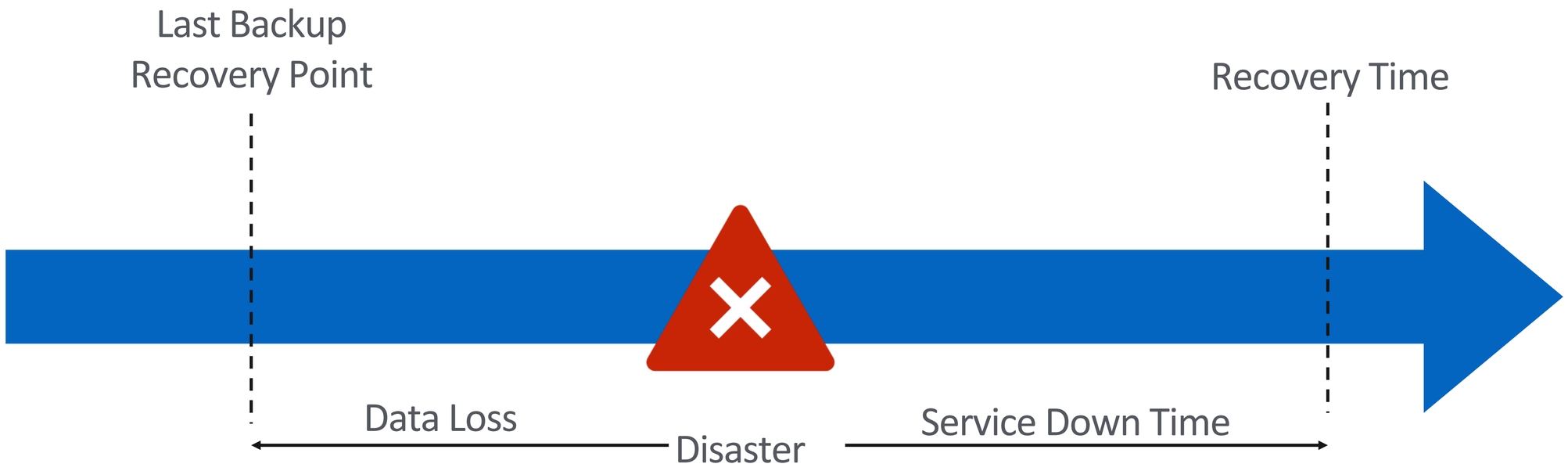


What is Disaster Recovery?

DEVIEW 2021

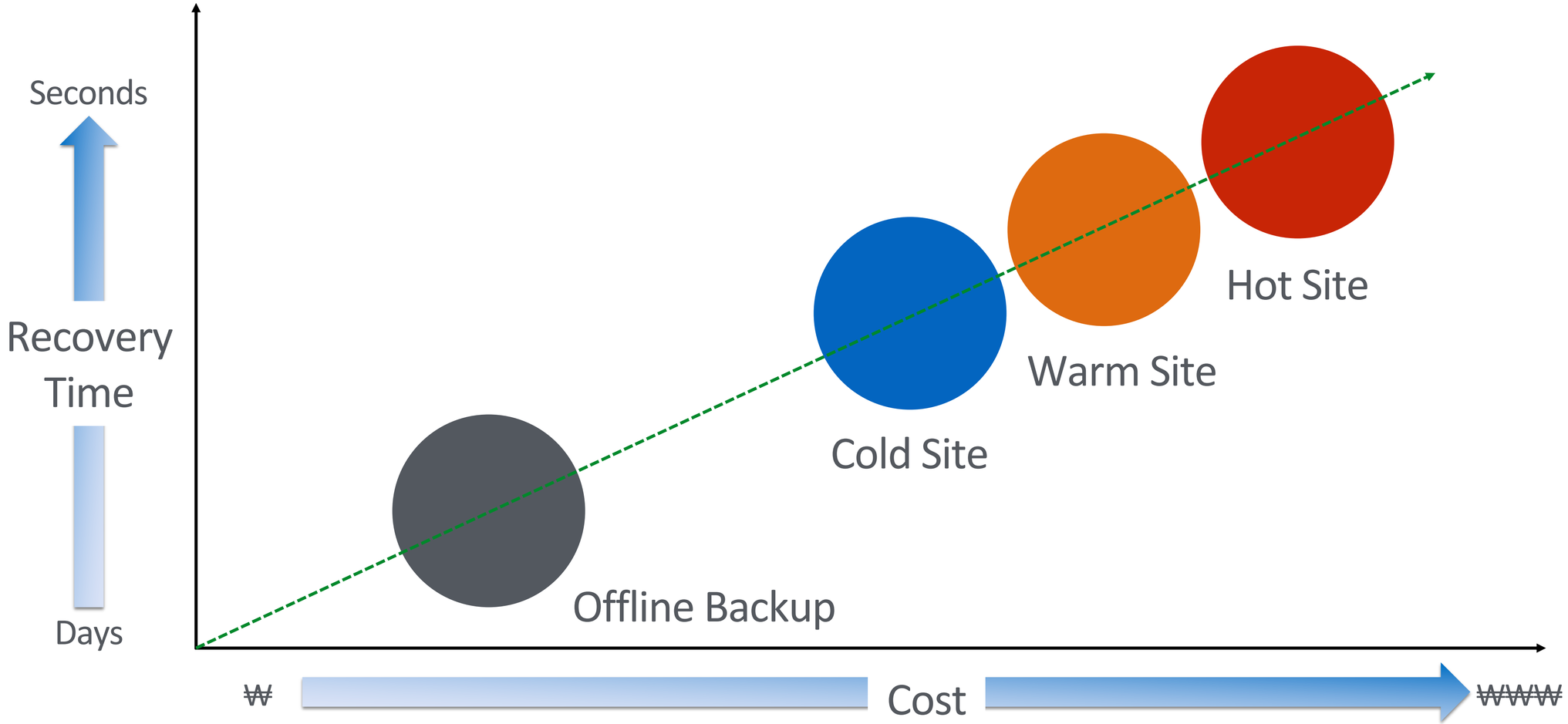
재해 복구 목표

- ✓ 복구 시점 목표
- ✓ 복구 시간 목표



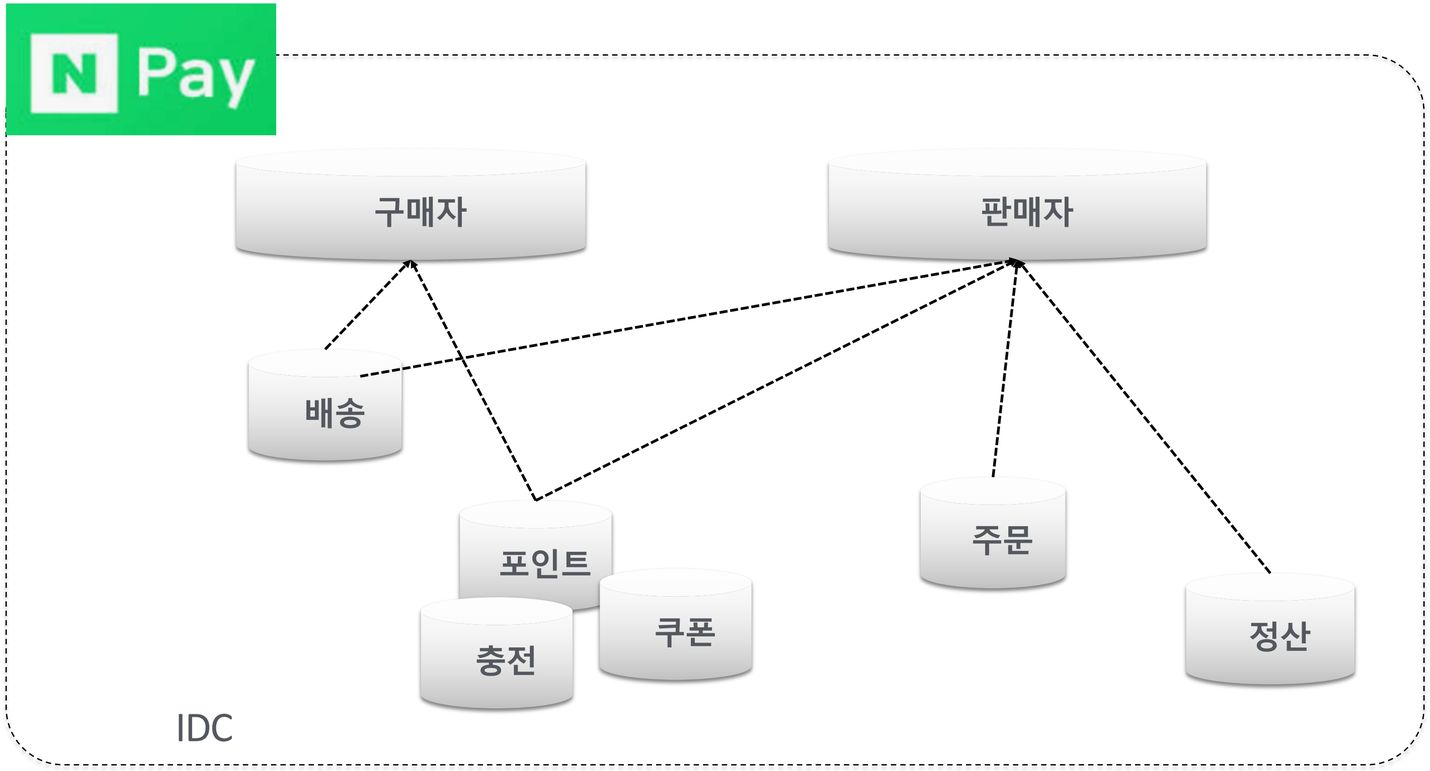
What is Disaster Recovery?

재해 복구 종류



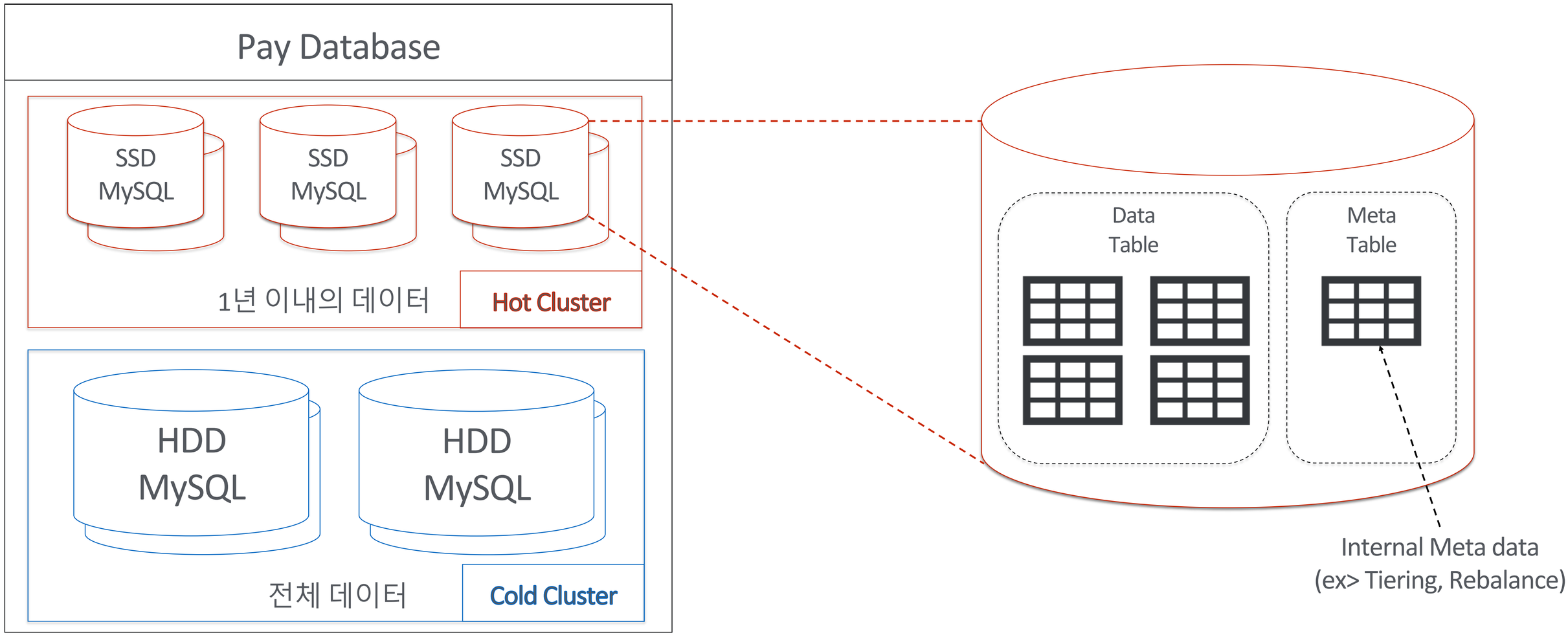
What is Disaster Recovery?

재해 복구 대상



What is Disaster Recovery?

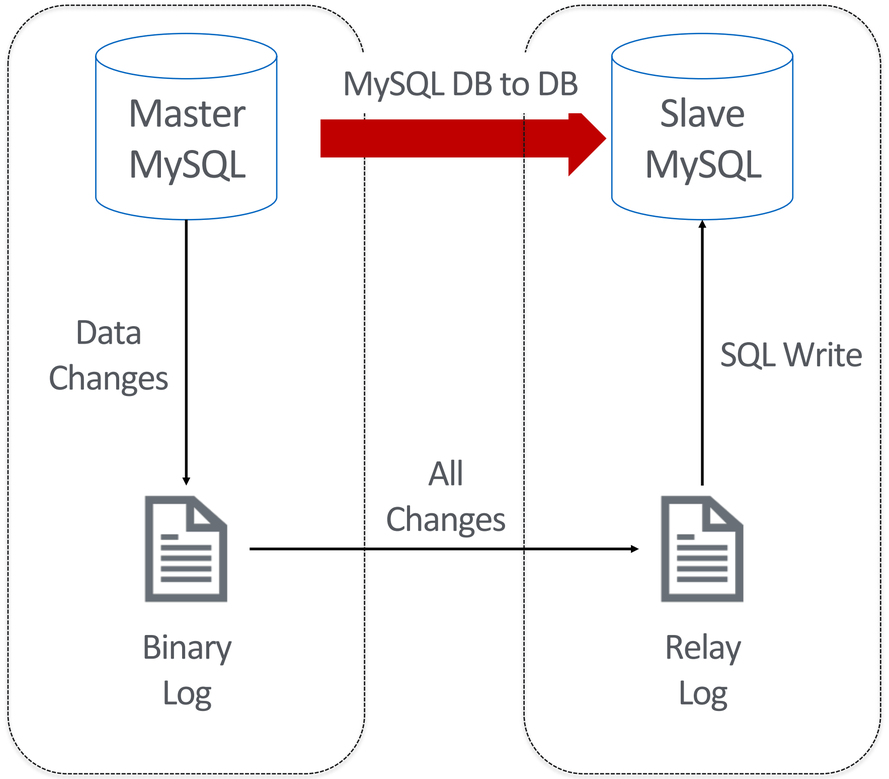
재해 복구 대상 DB 구조



Why Don’t Use MySQL Replication?

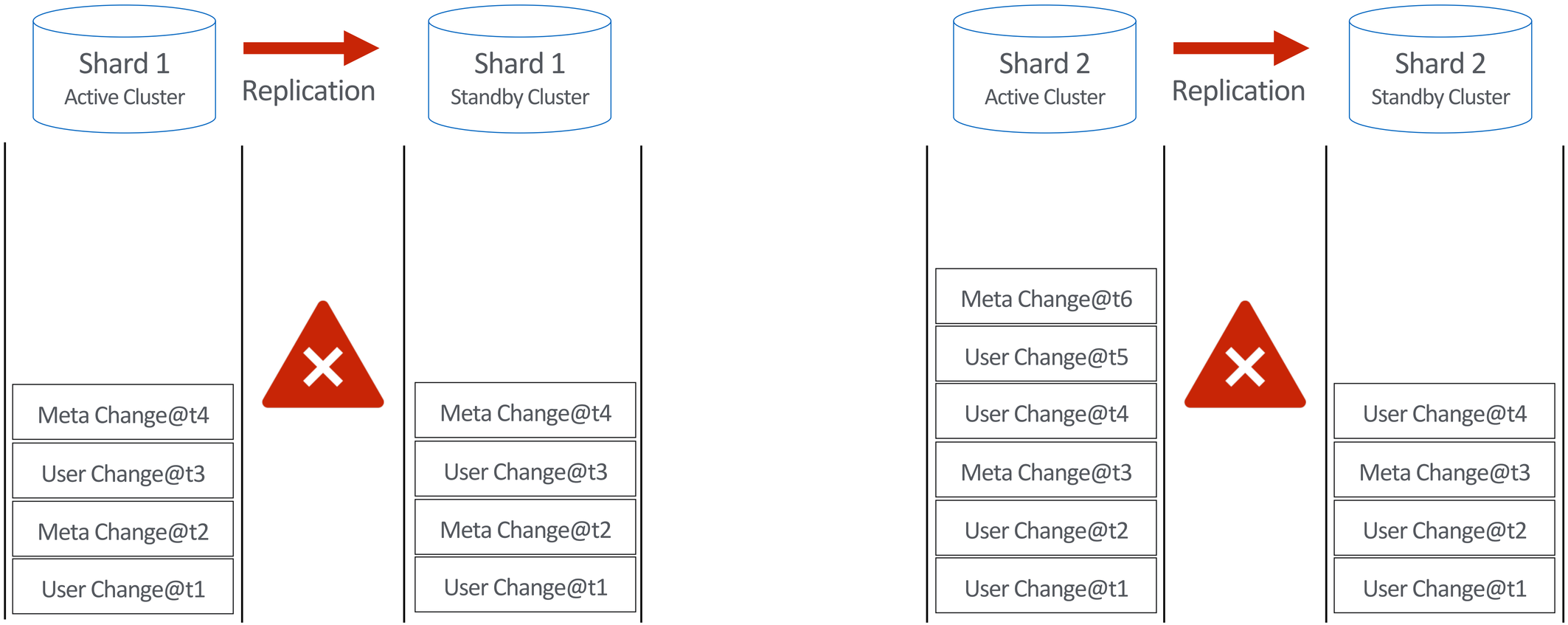
MySQL Replication

- ✓ DB to DB 복제
- ✓ Table의 모든 변경
- ✓ 비동기



Why Don’t Use MySQL Replication?

메타 정보 불일치

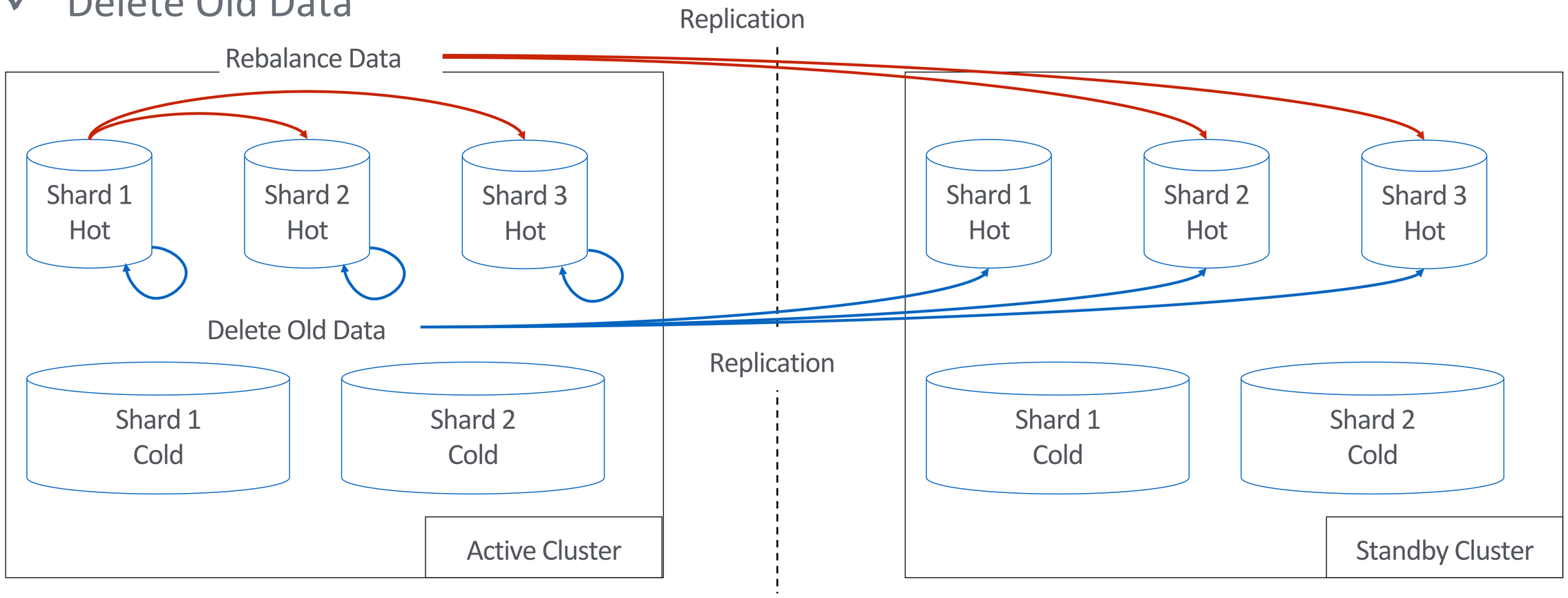


Why Don’t Use MySQL Replication?

DEVIEW 2021

IDC 간 높은 트래픽

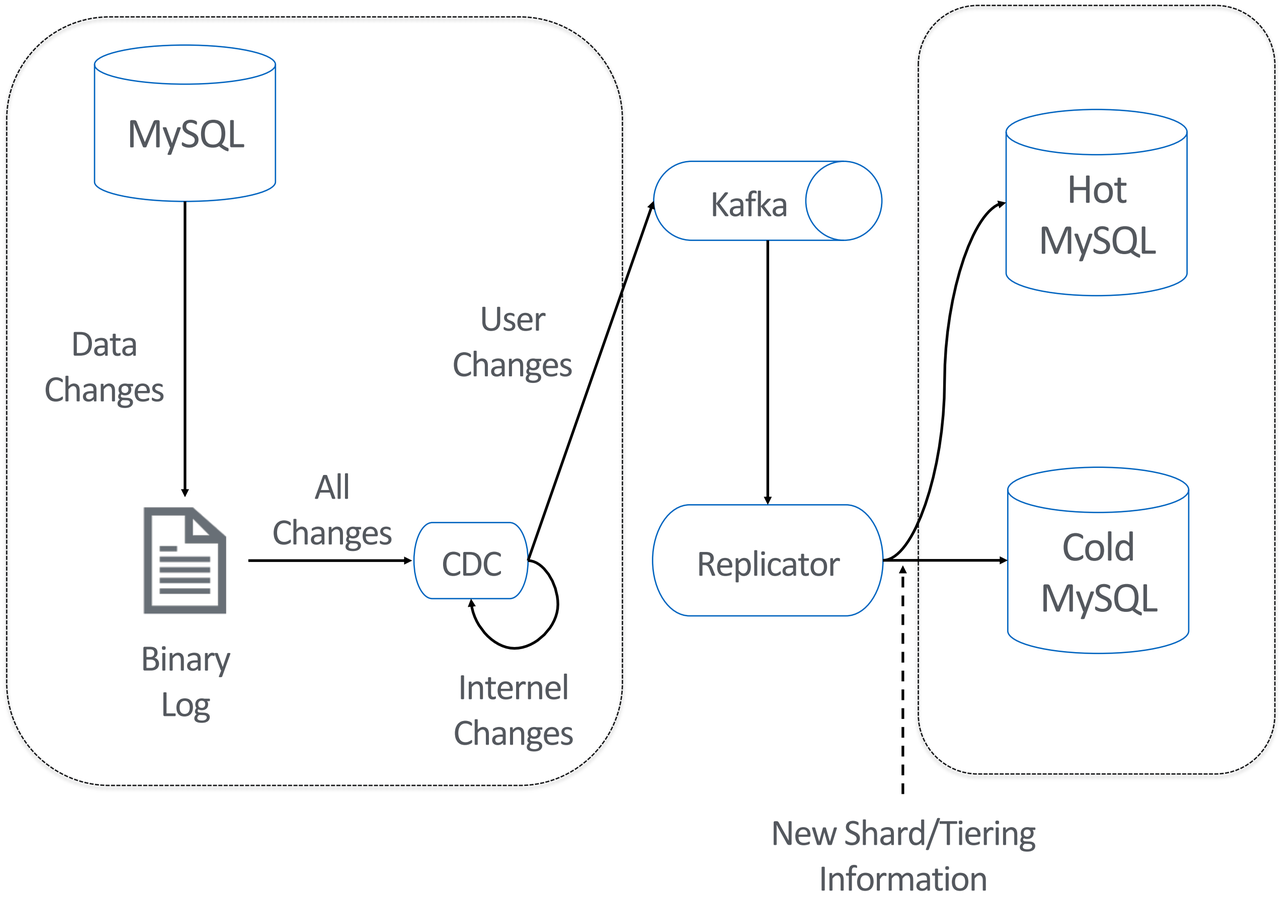
- ✓ Rebalance Data
- ✓ Delete Old Data



Using CDC in Disaster Recovery

Our Approach

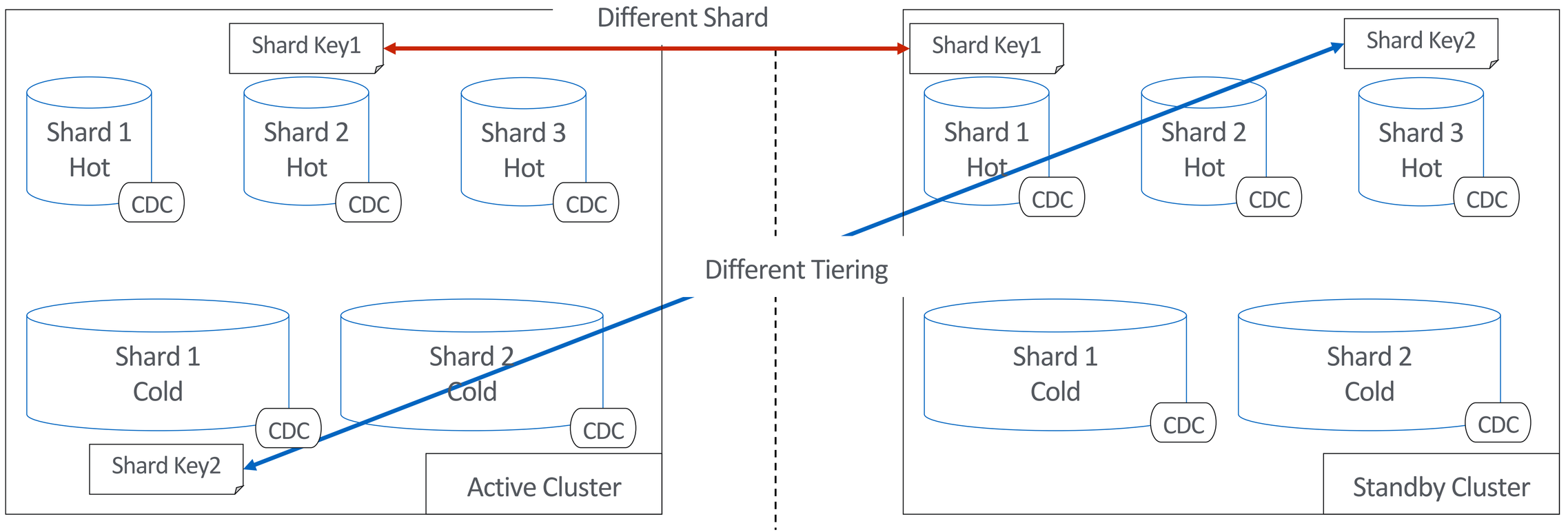
- ✓ 독립적 클러스터 구성
- ✓ 사용자 데이터 변경만 전송
- ✓ 비동기



Using CDC in Disaster Recovery

독립적인 클러스터

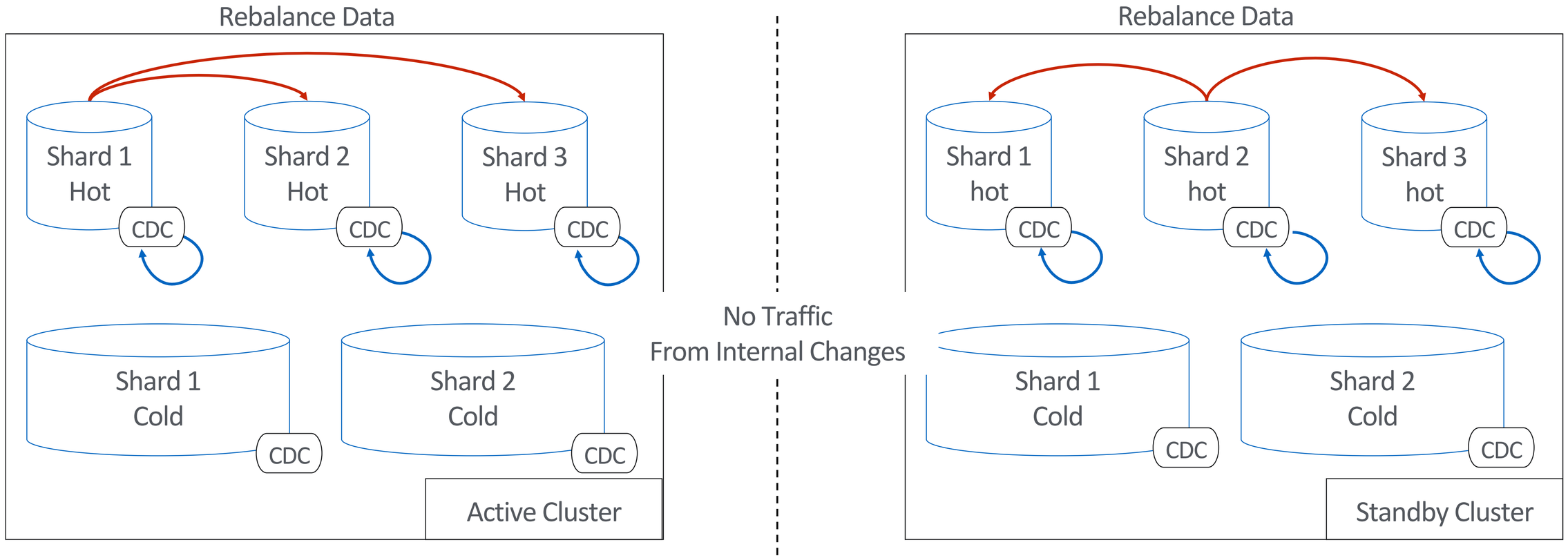
✓ 클러스터 별 메타정보 관리



Using CDC in Disaster Recovery

독립적인 클러스터

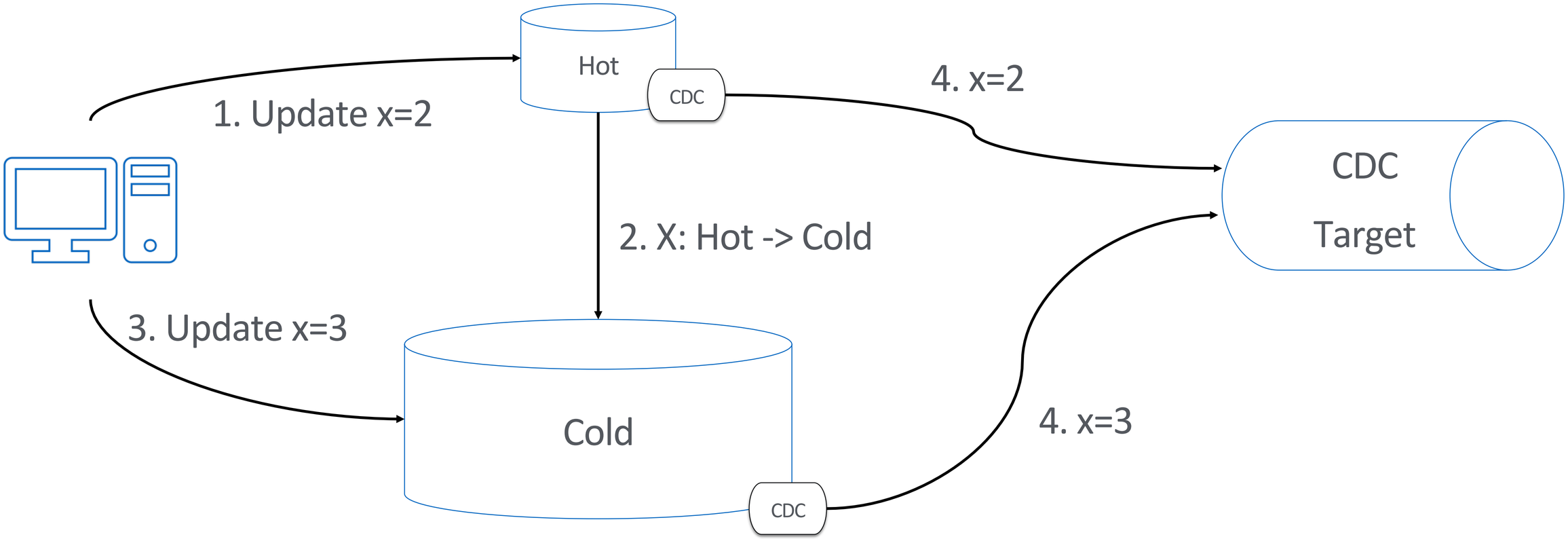
✓ 클러스터 별 내부 데이터 관리 -> IDC간 복제 트래픽 없음



Using CDC in Disaster Recovery

Tiering 환경에서 비동기 복제 시 문제점

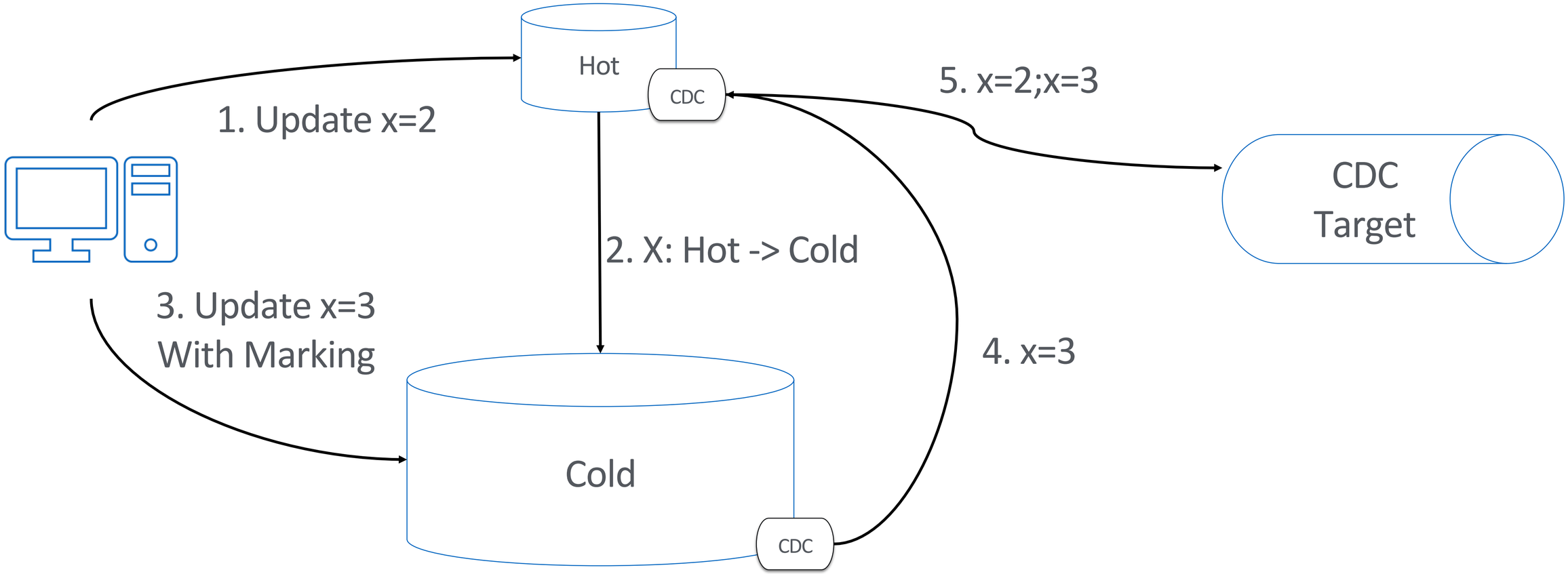
✓ Hot / Cold 에서 동일한 데이터 업데이트 시 순서 문제



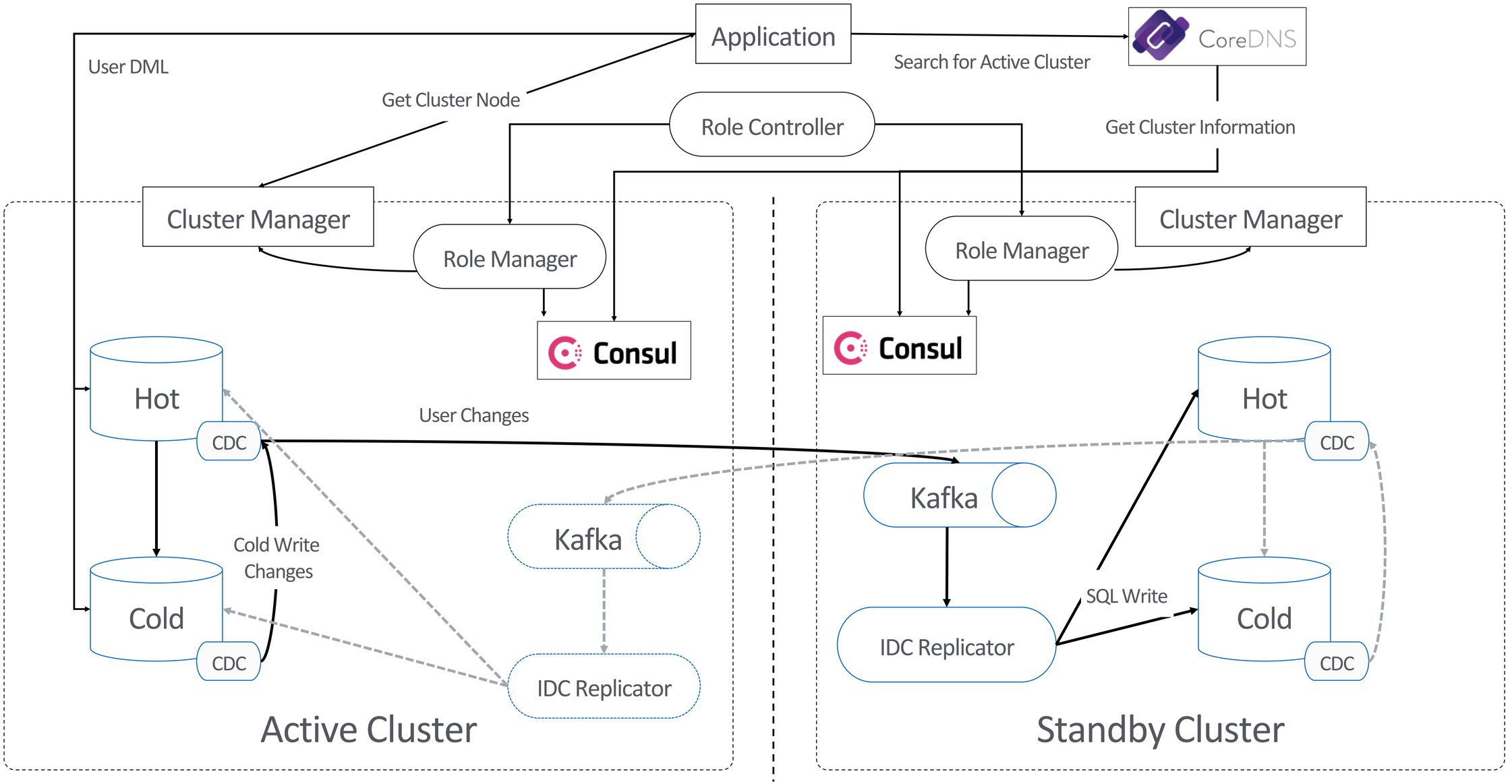
Using CDC in Disaster Recovery

Our Solution

✓ Relay Log : Cold Write -> Hot CDC -> CDC Target



DR Architecture



Thank You