Parallel Asynchronous Replication between YDB Database Instances

Andrey Fomichev,
Head of YDB

Ilnaz Nizametdinov,
YDB Senior Software Developer

1. Intro and Problem Statement
2. YDB Architecture in 5 minutes
3. An Approach to Asynchronous Replication in YDB
4. Dealing with multiple logs
5. Distributed Transactions in YDB
6. Globally ordered log and consistency

YDB

YDB — Open-source Distributed SQL Database

Database Distributed SQL means

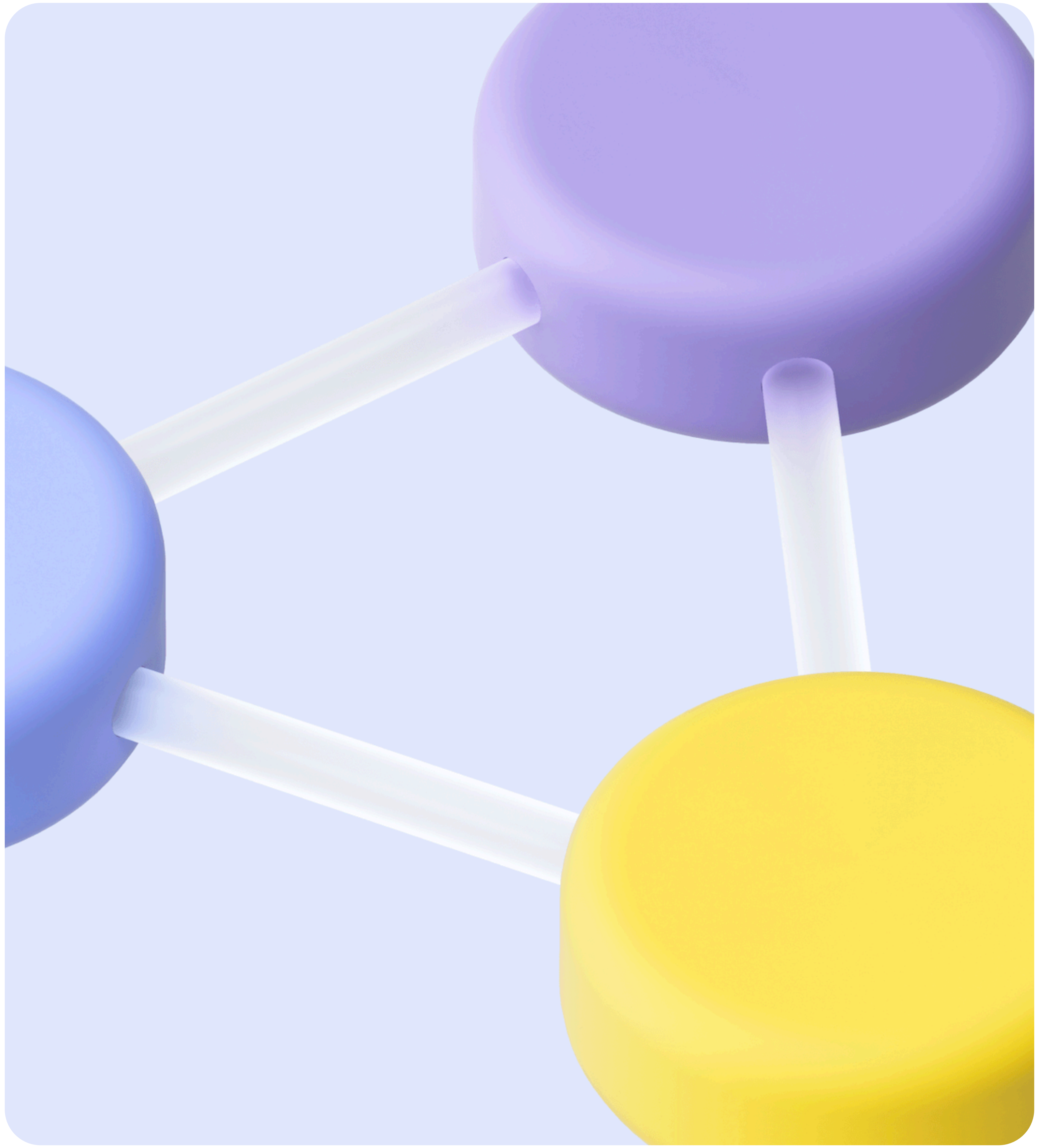
Multiple servers

Strongly consistent Relational database

Open Source

Apache 2.0 License

clck.ru/rdFQw



YDB Facts

Consistency& Serializable transaction execution

CAP-theorem, prefer CP

Serializable transaction isolation level

Highly available

Runs in multiple Availability Zones (AZ)

Survives AZ plus rack failure w/o human intervention, available for read/

Mission critical database

Works for projects with 24x7 requirements

No maintenance windows required

Platform

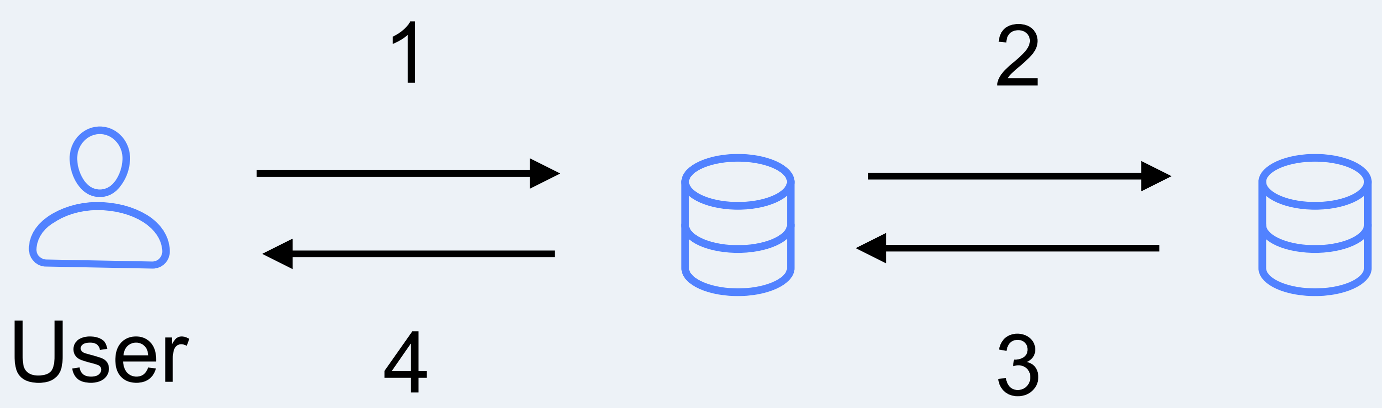
Topics, block store, time series, etc

Mostly OLTP workload

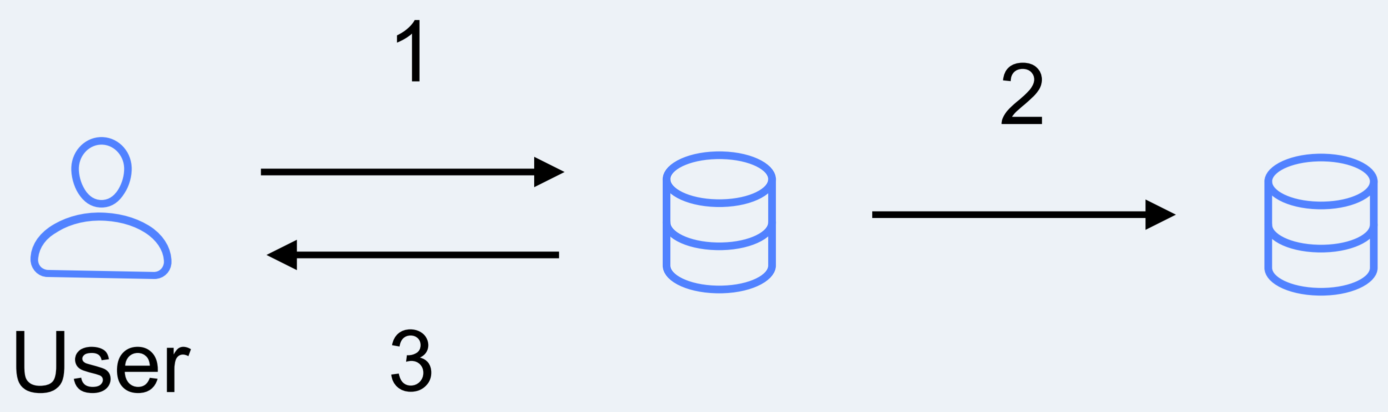
Column store plus ETL are in progress

Synchronous vs Asynchronous Replication in a Database

Synchronous Replication



Asynchronous Replication



Synchronous vs Asynchronous Replication in YDB

Different types of replication are possible even in single database installation

YDB is a database with strict consistency, so by default the replication is synchronous

Nevertheless asynchronous replication is also available in YDB — so called read replicas; they are used for

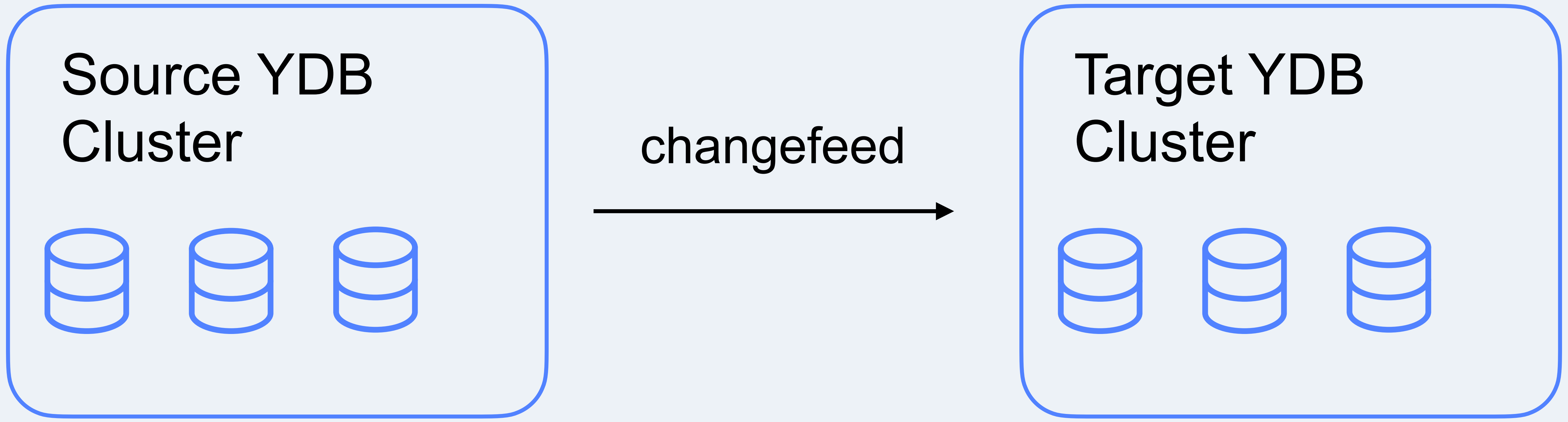
Read workload scalability

Even infinitely scalable database may have problems, if you would like to perform 1M rps for a single key

Latency optimization for read queries, i.e. read replicas may run in every AZ to avoid cross-AZ read queries

Comes with relaxed guarantees

Asynchronous Replication between YDB instances



Why do We Need Asynchronous Replication between YDB instances?

Disaster Recovery, hot Standby

Even fault tolerant systems may experience availability issues

Recovery from backup may be unacceptably time consuming process

Regional clusters

Spread YDB cluster over continents introduce high write latency

Different load patterns

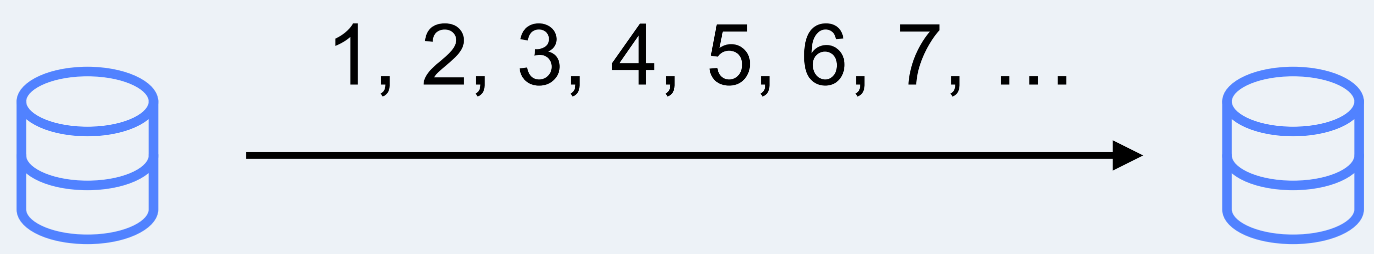
OLTP and OLAP load patterns

Regulatory Compliance (GDPR)

User table per country, replicate to other regions

What guarantees are expected from asynchronous replication?

Strong ordering



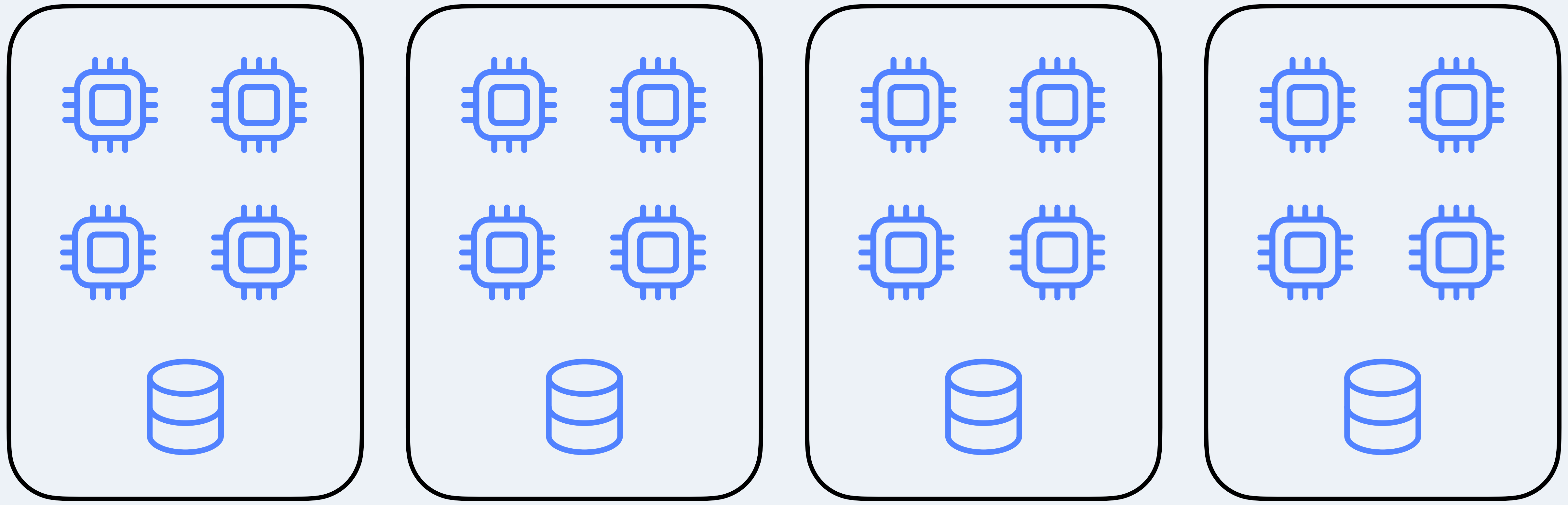
Global consistency

Adequate delays

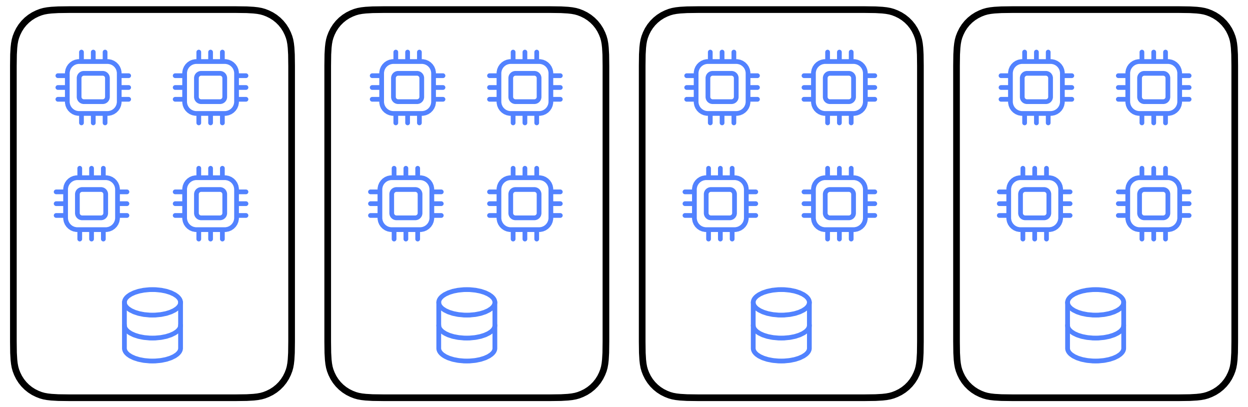
1. Intro and Problem Statement
2. YDB Architecture in 5 minutes
3. An Approach to Asynchronous Replication in YDB
4. Dealing with multiple logs
5. Distributed Transactions in YDB
6. Globally ordered log and consistency

Share Nothing Architecture

Cluster of nodes, share nothing architecture, commodity hardware



Tables and Queries



SQL Query

ID	Value1	Value2
GX008	8921	1114
GX278	827	9
GY045	654	345
SK720	3445	3456
SM527	7668	7643
UA628	72	3928

Key	Data
82	8921
283	827
346	654
1273	3445

Tables are sorted by primary key

Table Partitioning

All tables data are split into partitions, partitions are stored in Tablets

DataShard	ID	Value1	Value2
DataShard	GX008	8921	1114
	GX278	827	9
DataShard	GY045	654	345
	SK720	3445	3456
DataShard	SM527	7668	7643
	UA628	72	3928

Key	Data	DataShard
82	8921	DataShard
283	827	
346	654	DataShard
1273	3445	

Inside Tablet (1)

Tablet is a core part of YDB

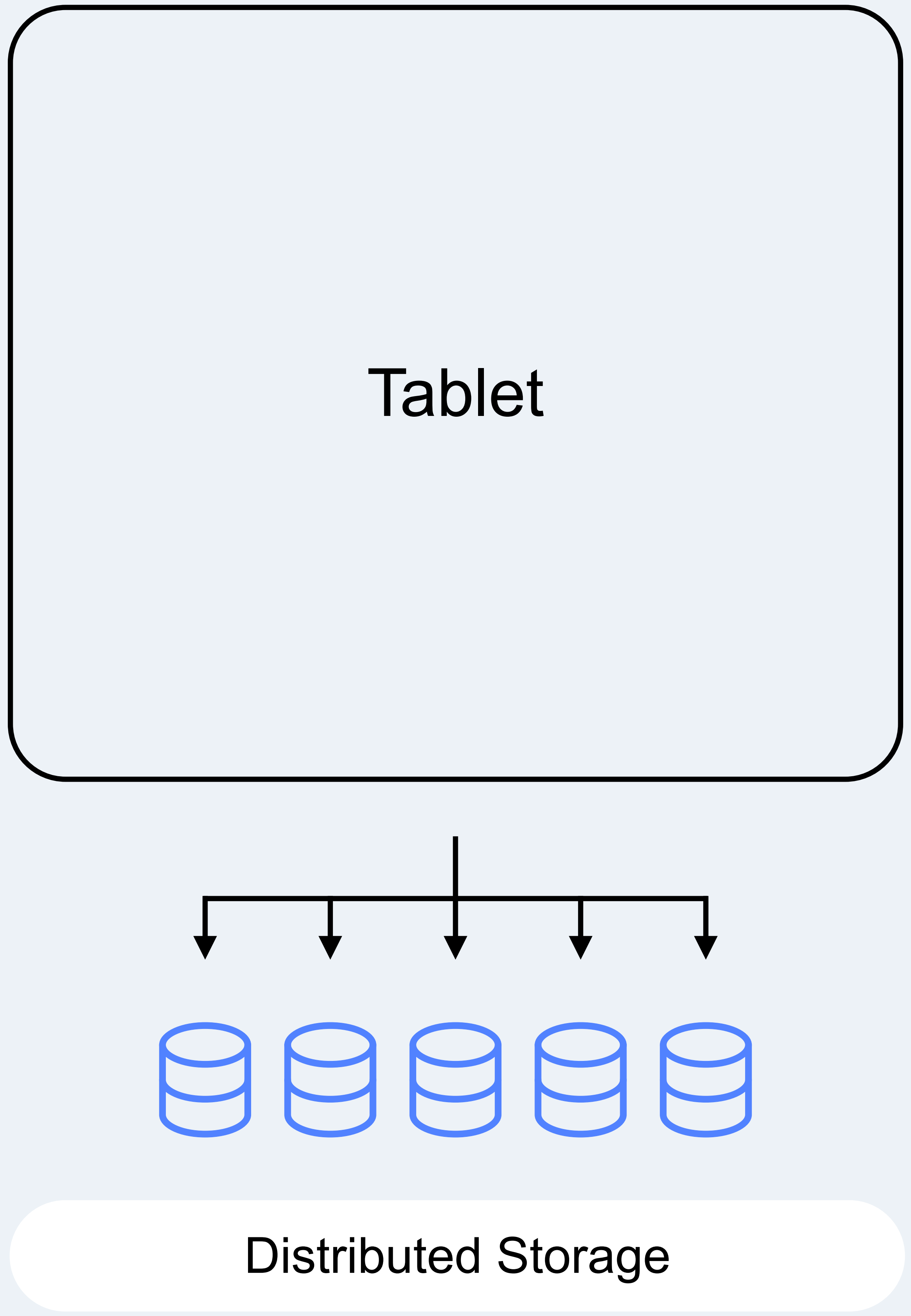
It provides API to the upper level, for instance

- Insert row
- Delete row
- Read row

You can think about tablet as an adapter to the data stored in Distributed Storage

- Tablet usually has volatile data cache
- On "update" operation tablet writes a record to the log
- table can die because of node failure
  or other reasons and run at another node

Technically, tablet implementation is a set of C++ classes



Inside Tablet (2)

Replication State Machine

- Writes a log of changes
- Recovers from log on tablet crash
- Provides guarantees analogous to RAFT and Paxos

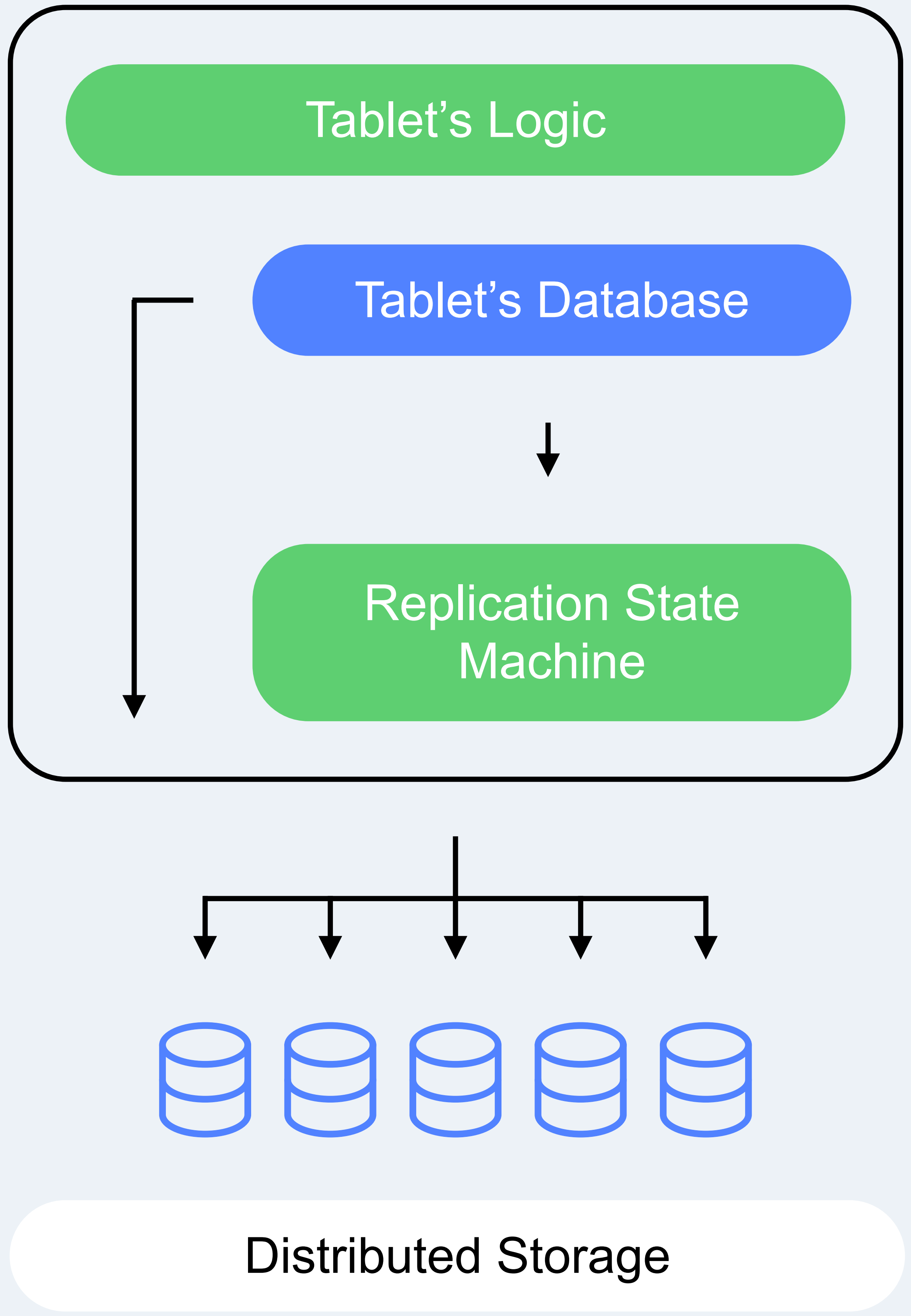
Tablet’s Database

- Data is organised as an LSM-tree
  (Log Structured Merge Tree)
- Guarantees ACID properties for the data it is in charge

Tablet’s Logic is specific for the Tablet type

- Can implement different APIs
- Can be active component

Distributed Storage provides reliable data storage with redundancy



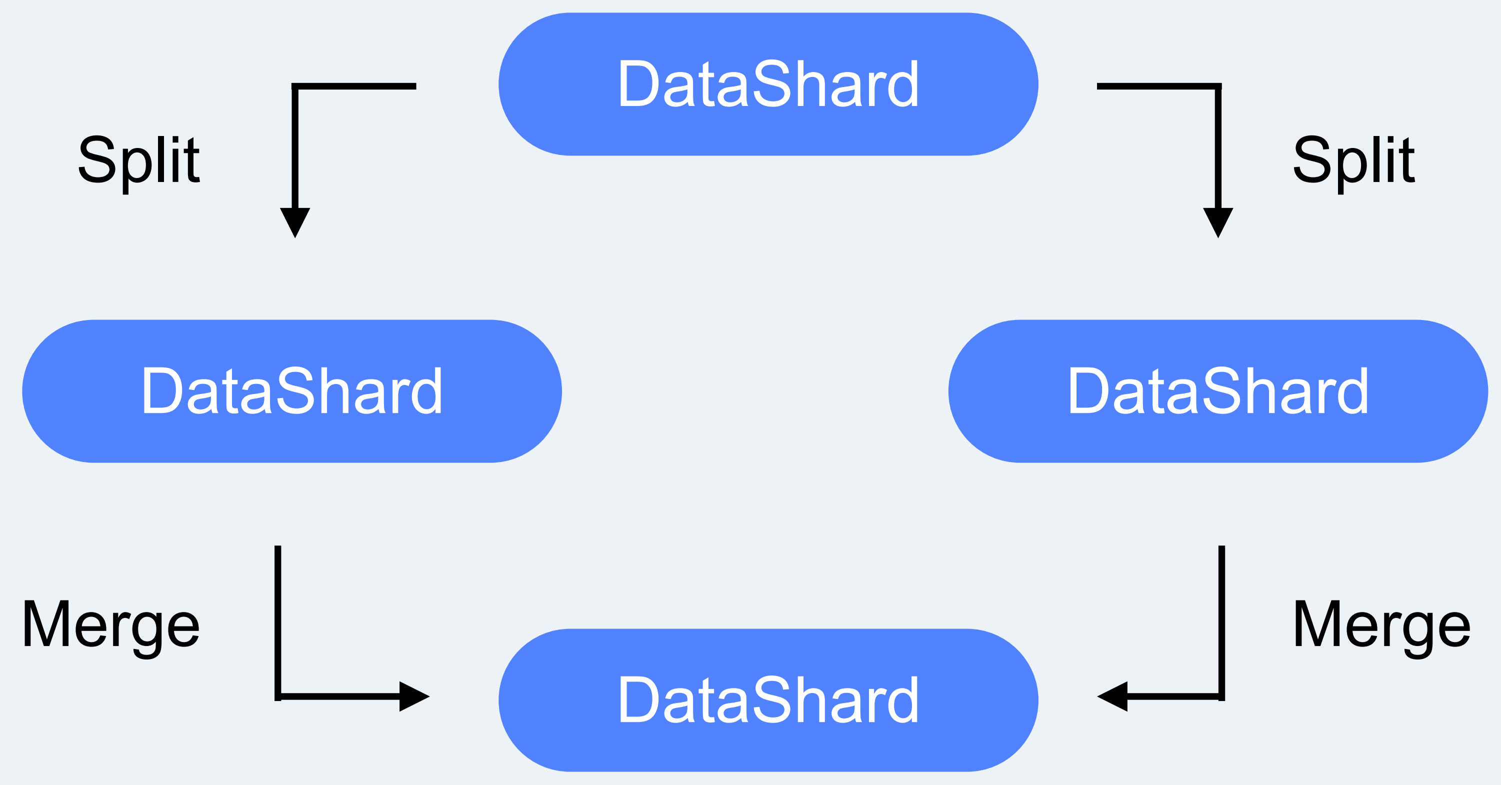
DataShard Split/Merge

Data in tables and DataShard lives it own live

- Key range can grow and become too large for one tablet
- Key range can decrease, so we get too many small tablets

DataShards can automatically

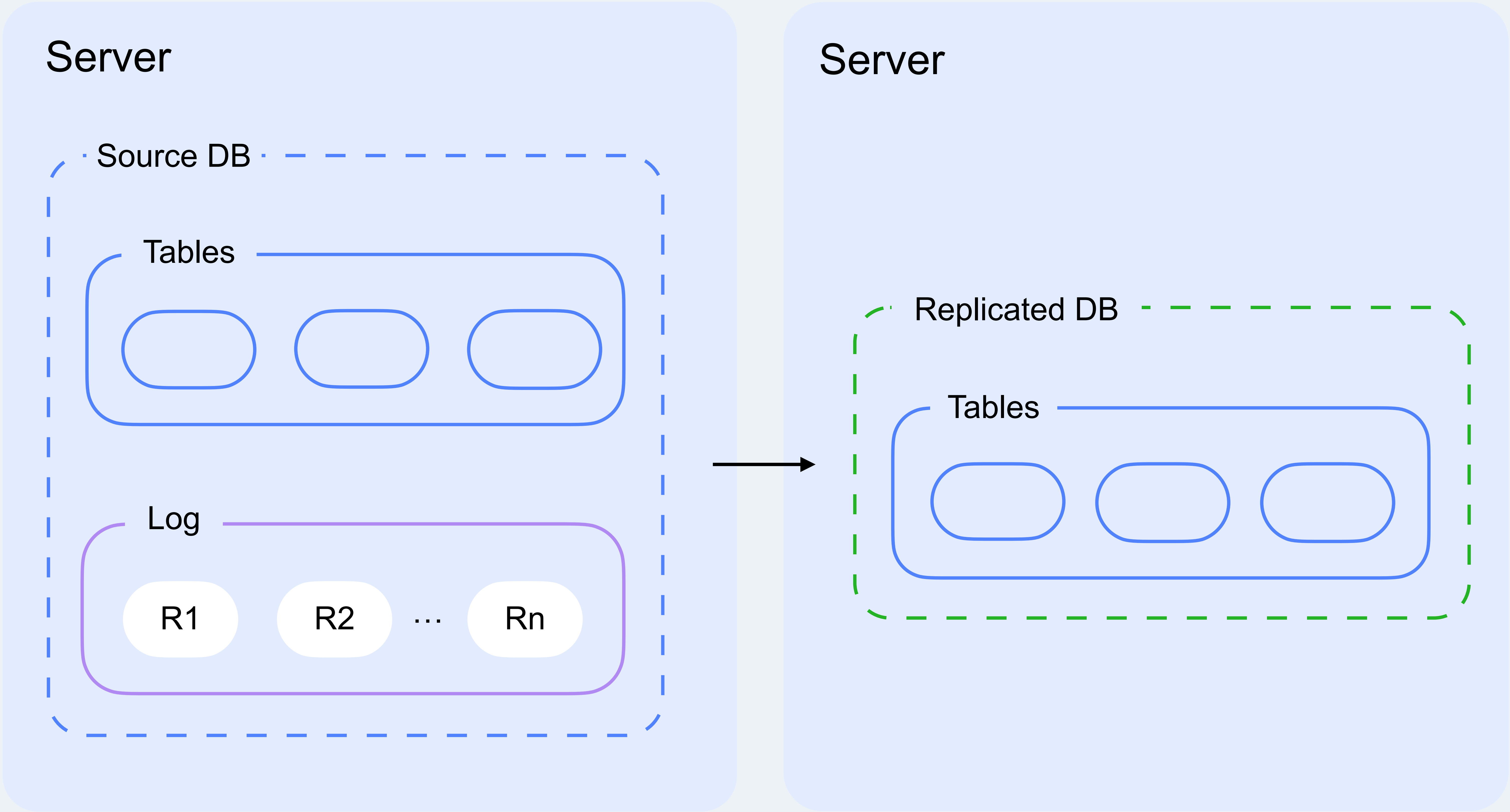
- Split on multiple DataShards
- Merge with their neighbours to form a larger DataShard



1. Intro and Problem Statement
2. YDB Architecture in 5 minutes
3. An Approach to Asynchronous Replication in YDB
4. Dealing with multiple logs
5. Distributed Transactions in YDB
6. Globally ordered log and consistency

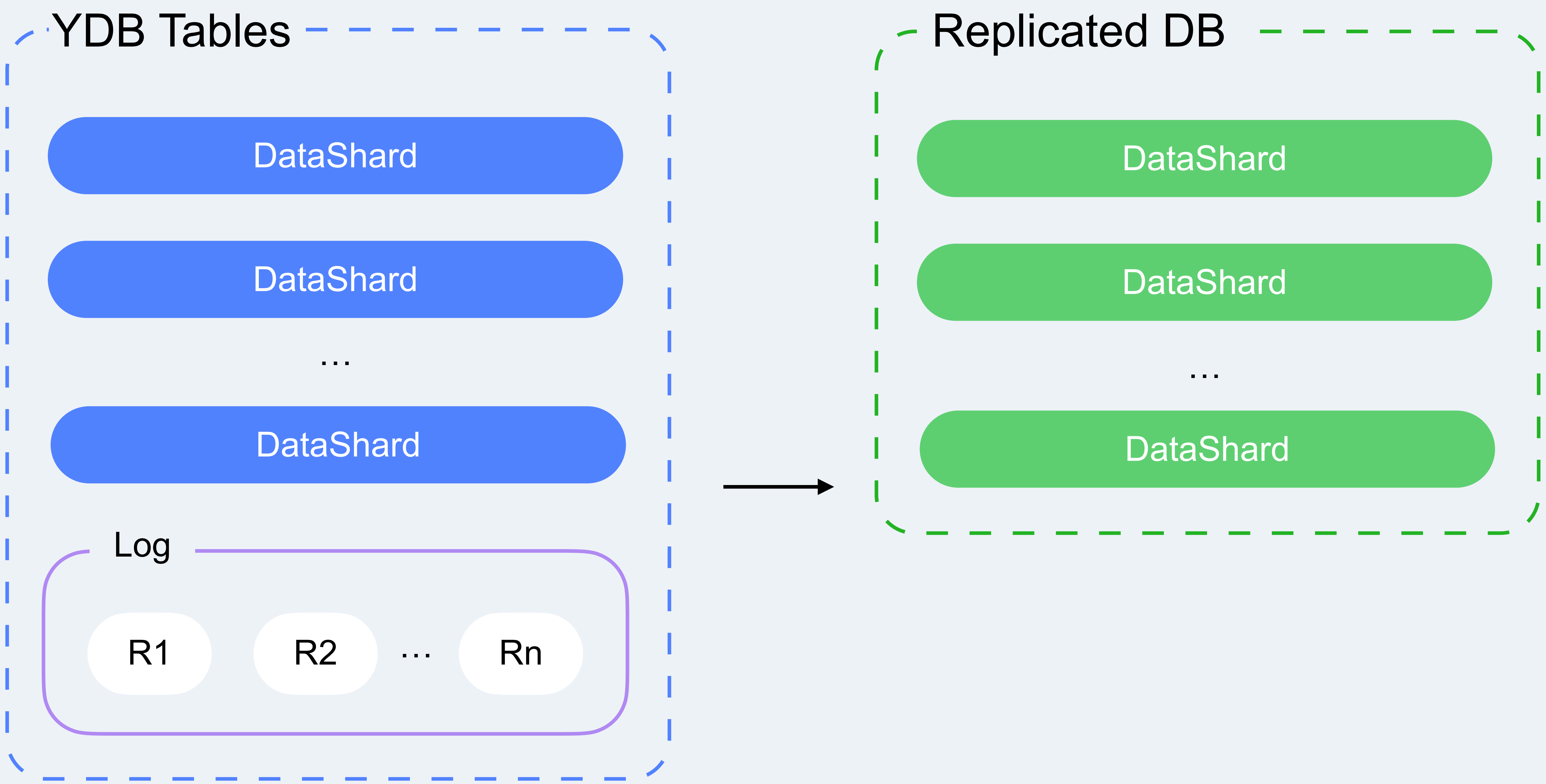
Asynchronous replication between relational DBs

- DB is hosted on a single server
- There is a lot of tables in DB
- But single log (e.g. binlog)



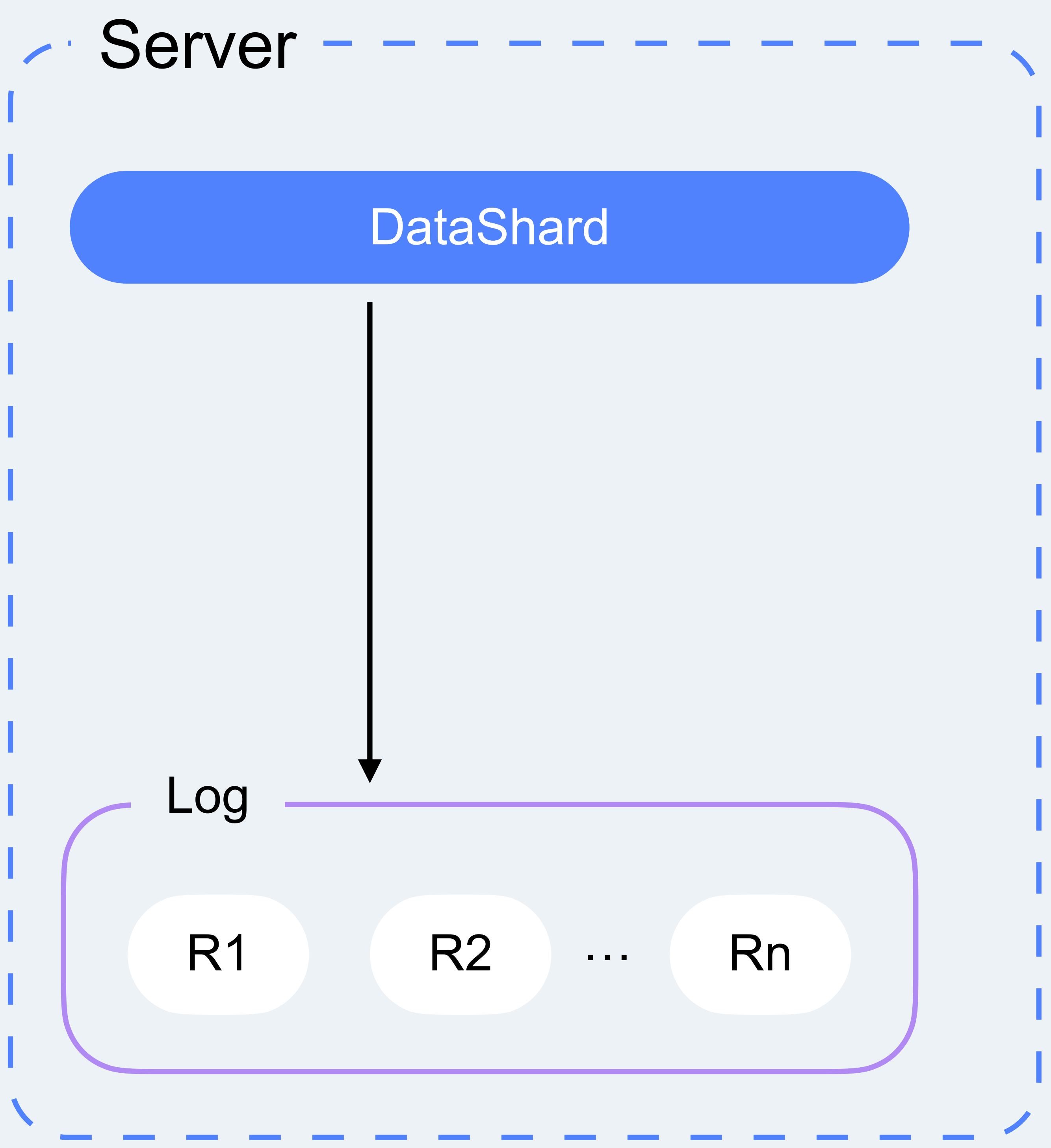
Asynchronous replication between YDB DBs

- DB is hosted on multiple servers
- Each table is a set of DataShards
- Can there be a single log?

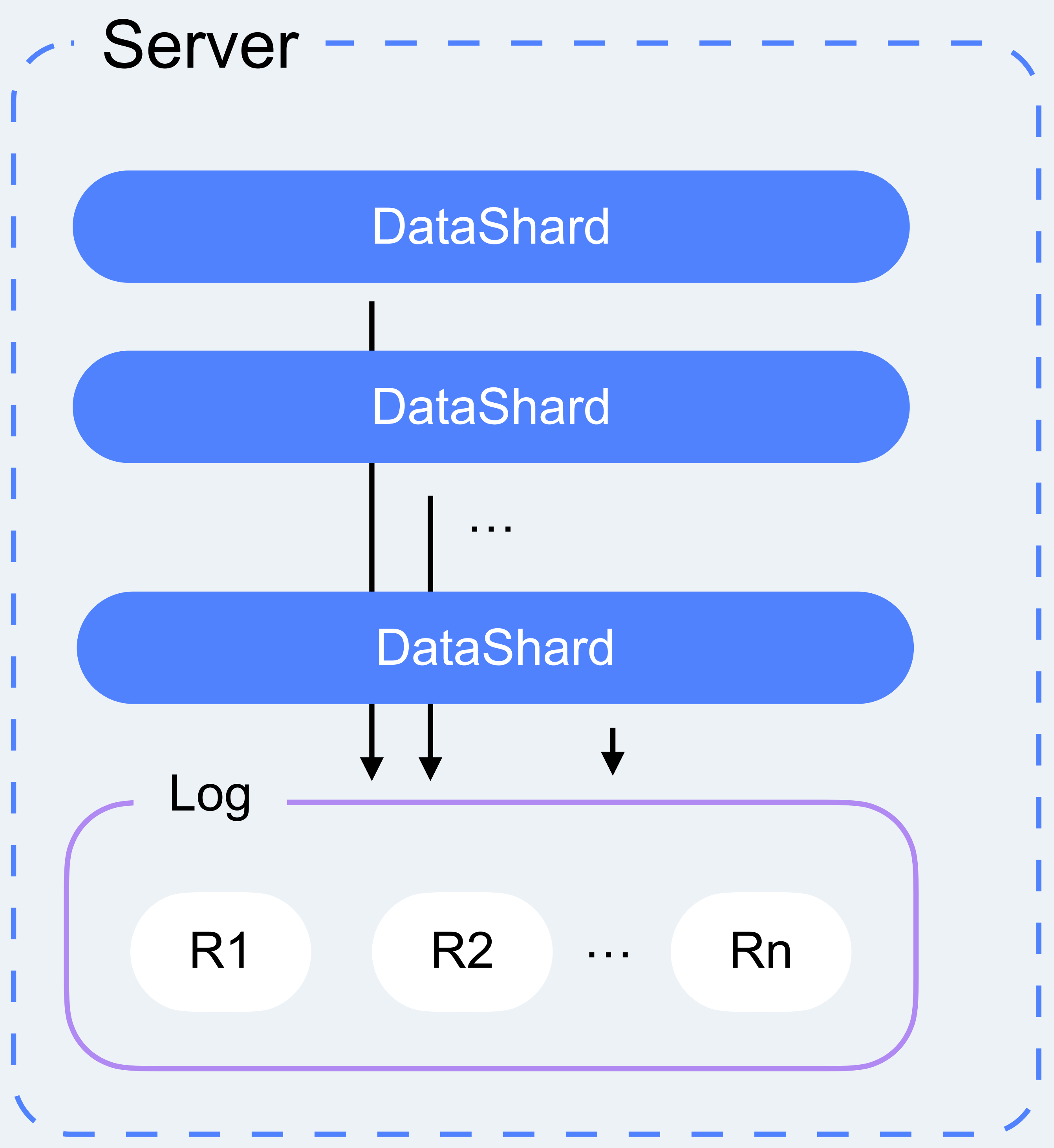


…

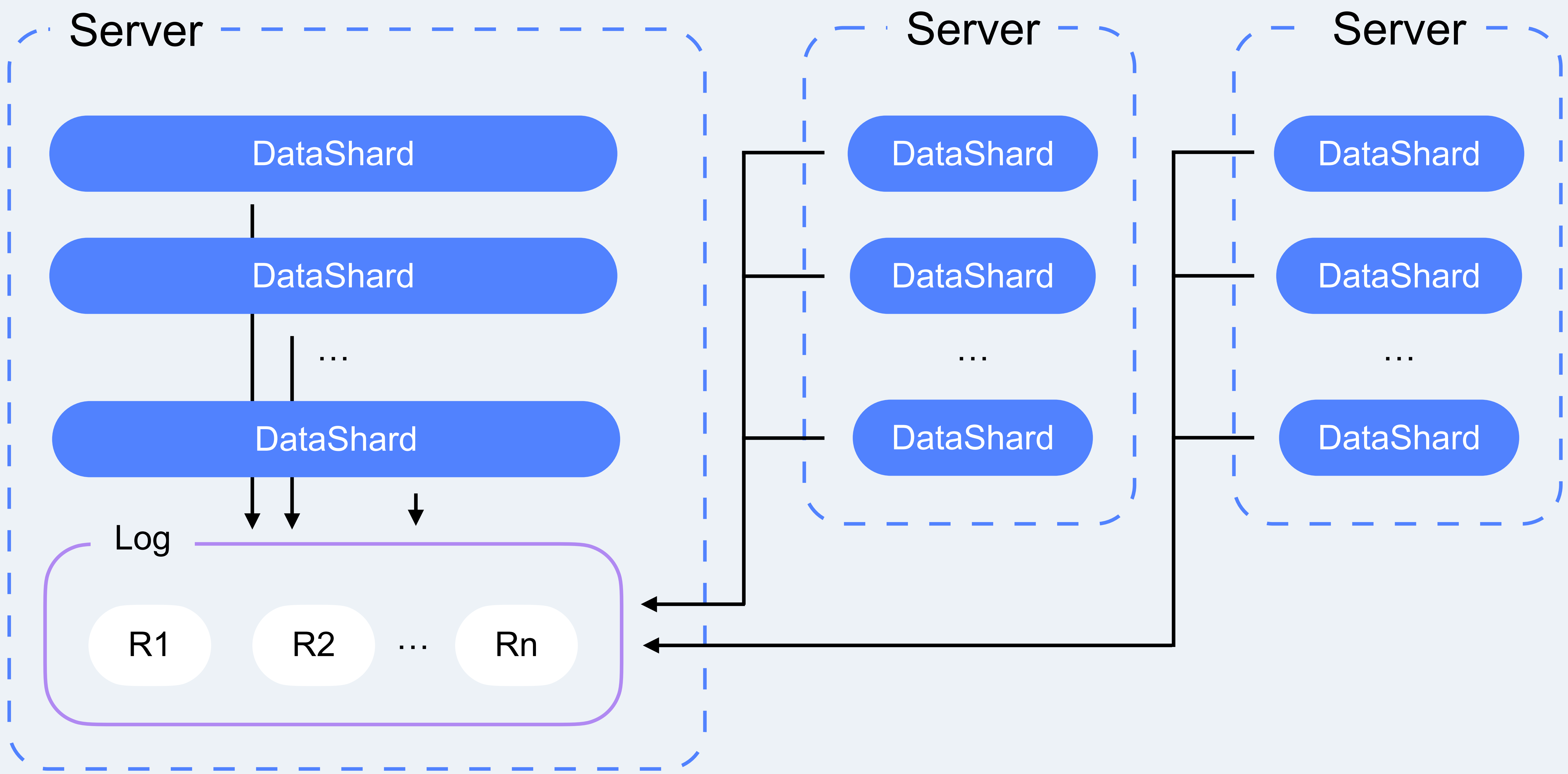
Can there be a single log?



Can there be a single log?



Can there be a single log?



Design decision for asynchronous replication

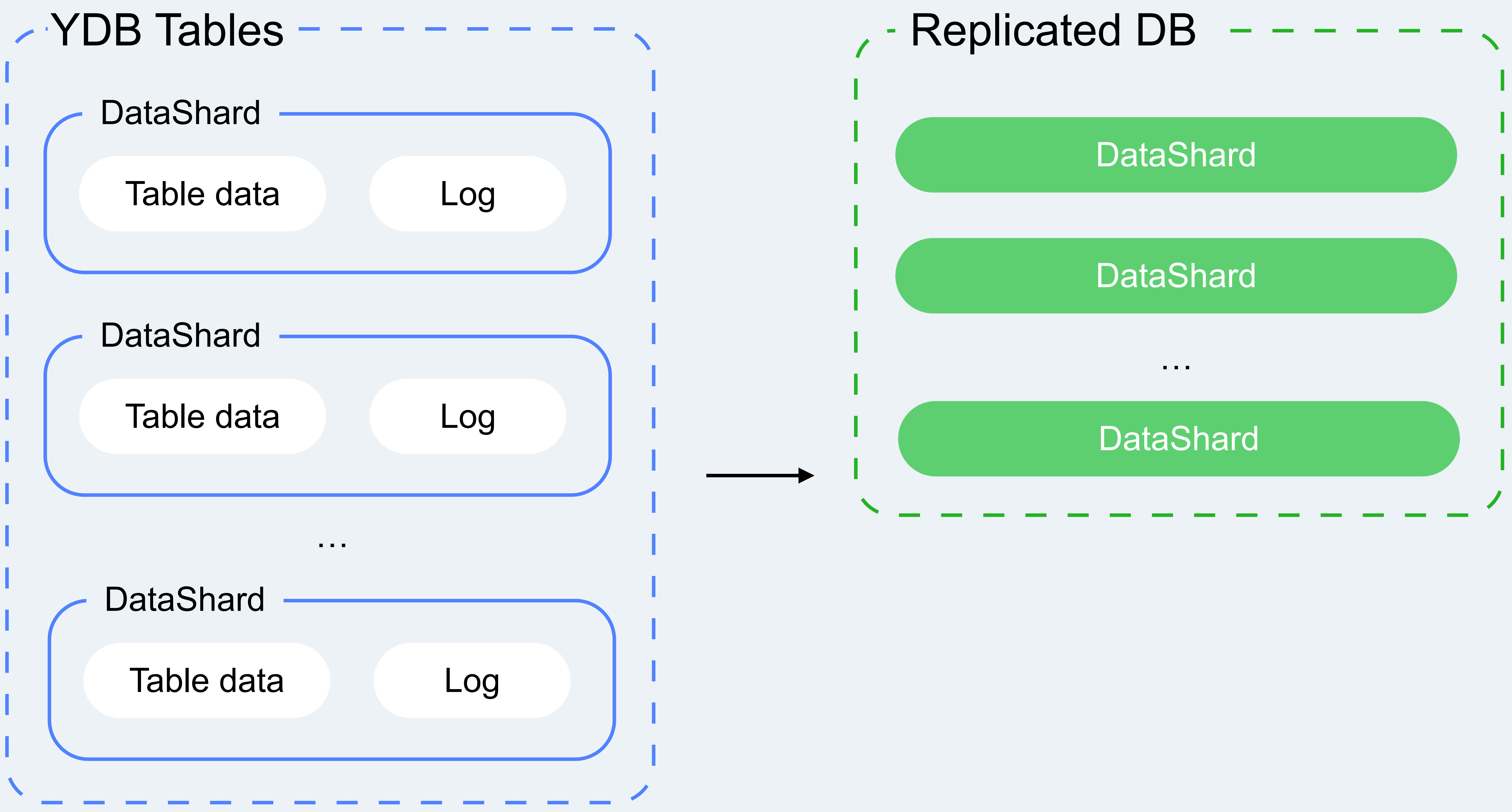
The log is a FIFO data structure

A single log has scalability limitations

So we need multiple logs

Asynchronous replication between YDB DBs

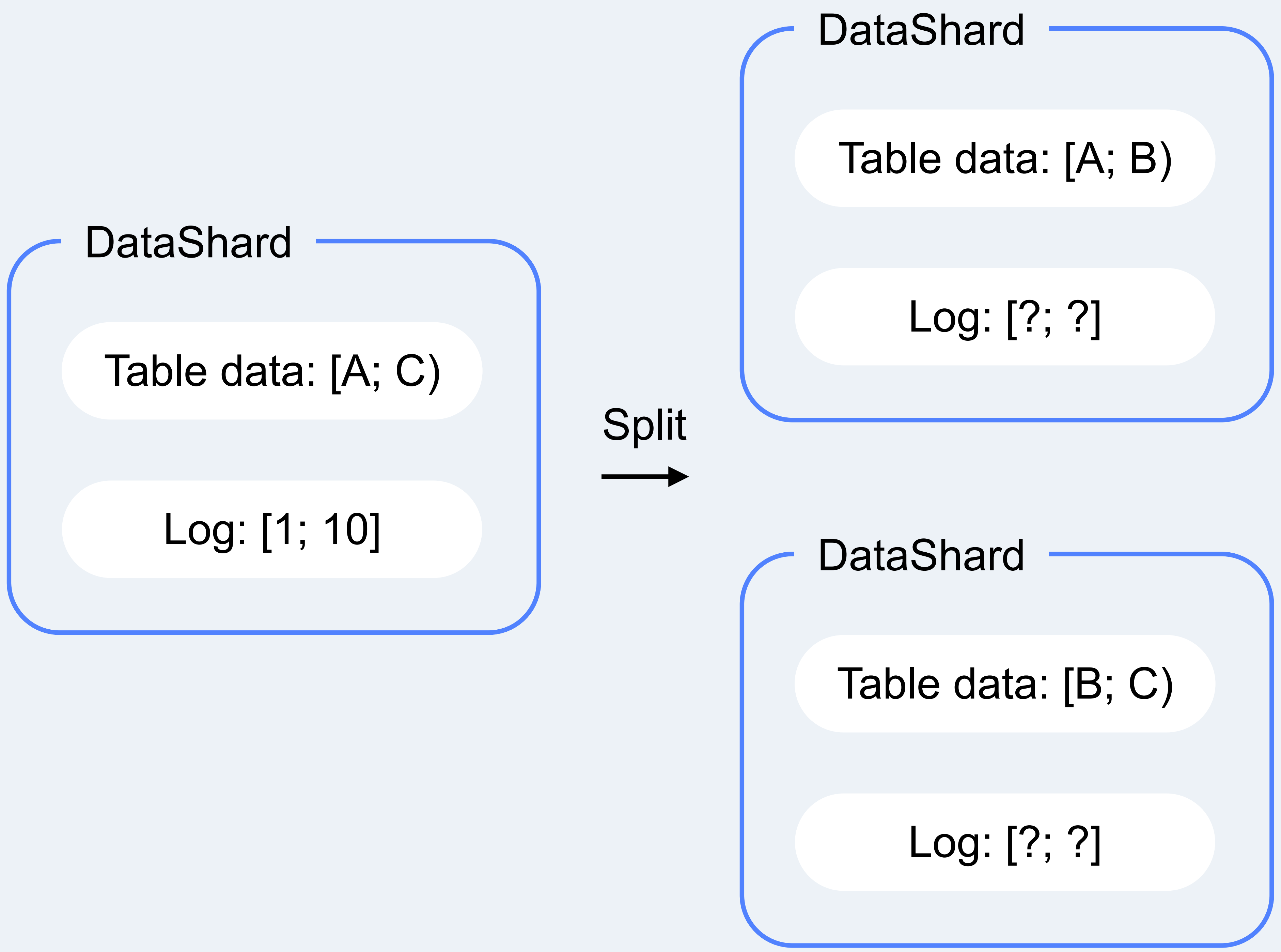
- Each DataShard has its own log
- Normally this log is small
- But it can growth in case of connectivity issues



...

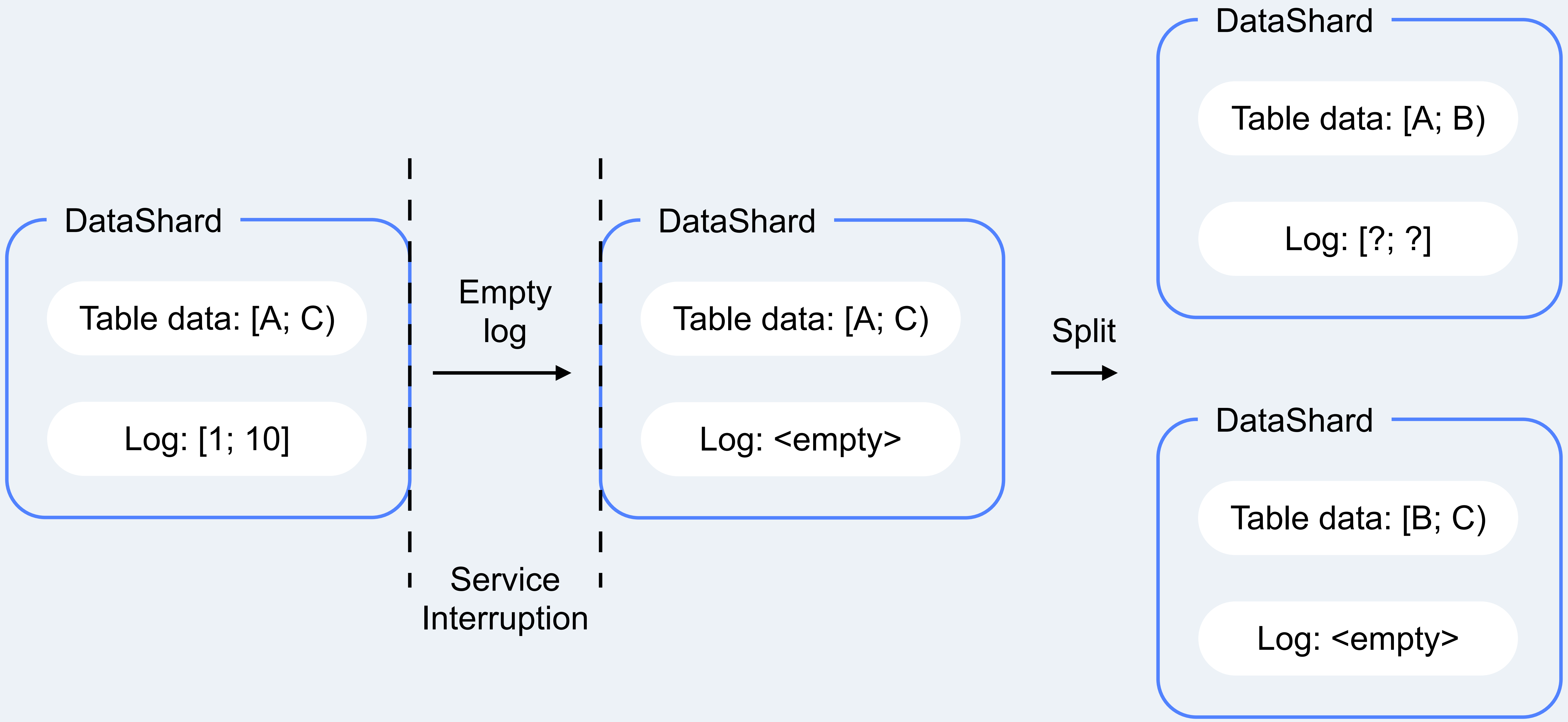
Log growth causes problems

- Table data is sorted by PK
- Log is sorted in the order in which the changes appeared
- Splitting the log is non-trivial



Log growth causes problems

- Instead of splitting the log, it possible to empty log before split
- But this will cause a service interruption
- Its duration depends on the size of the log



Problems of asynchronous replication

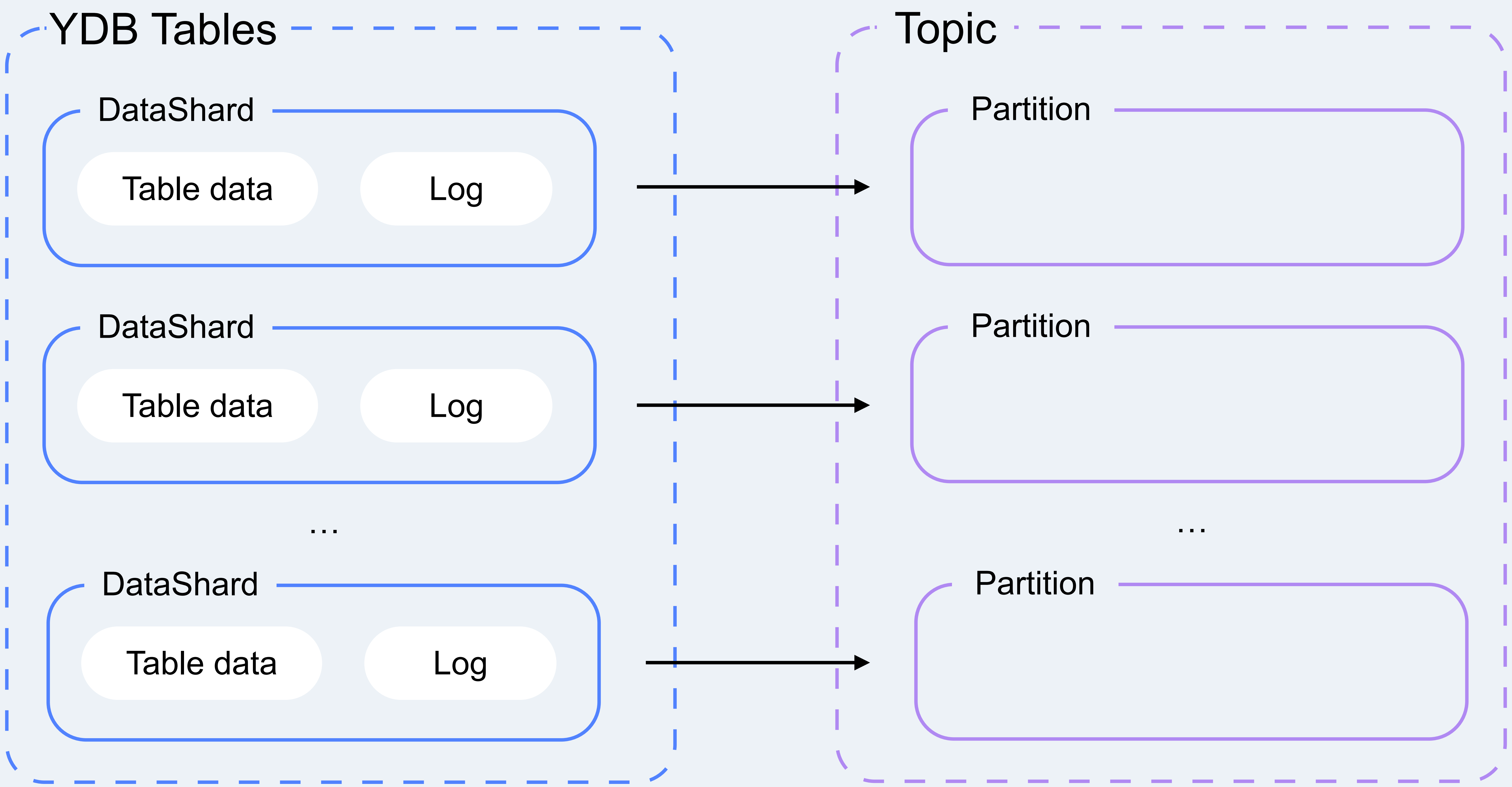
A lot of DataShards each with its own log

Connectivity issues will cause the log to growth

Large log will cause service interruption during DataShard split

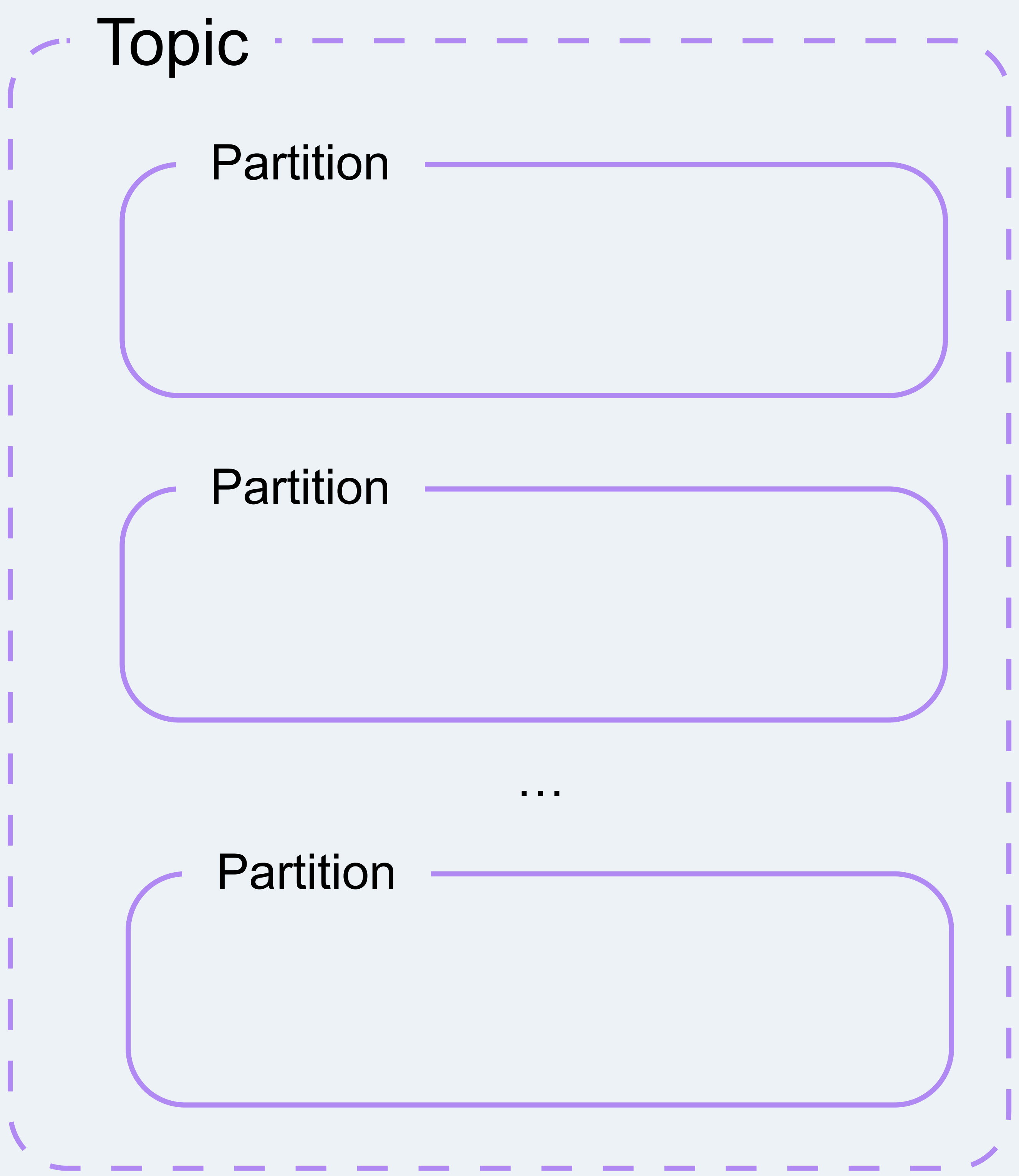
Transferring log to specialized storage

- Specialized storage solves the log growth problem
- DataShard log remains small
- Storage's partitions can store logs for a long time



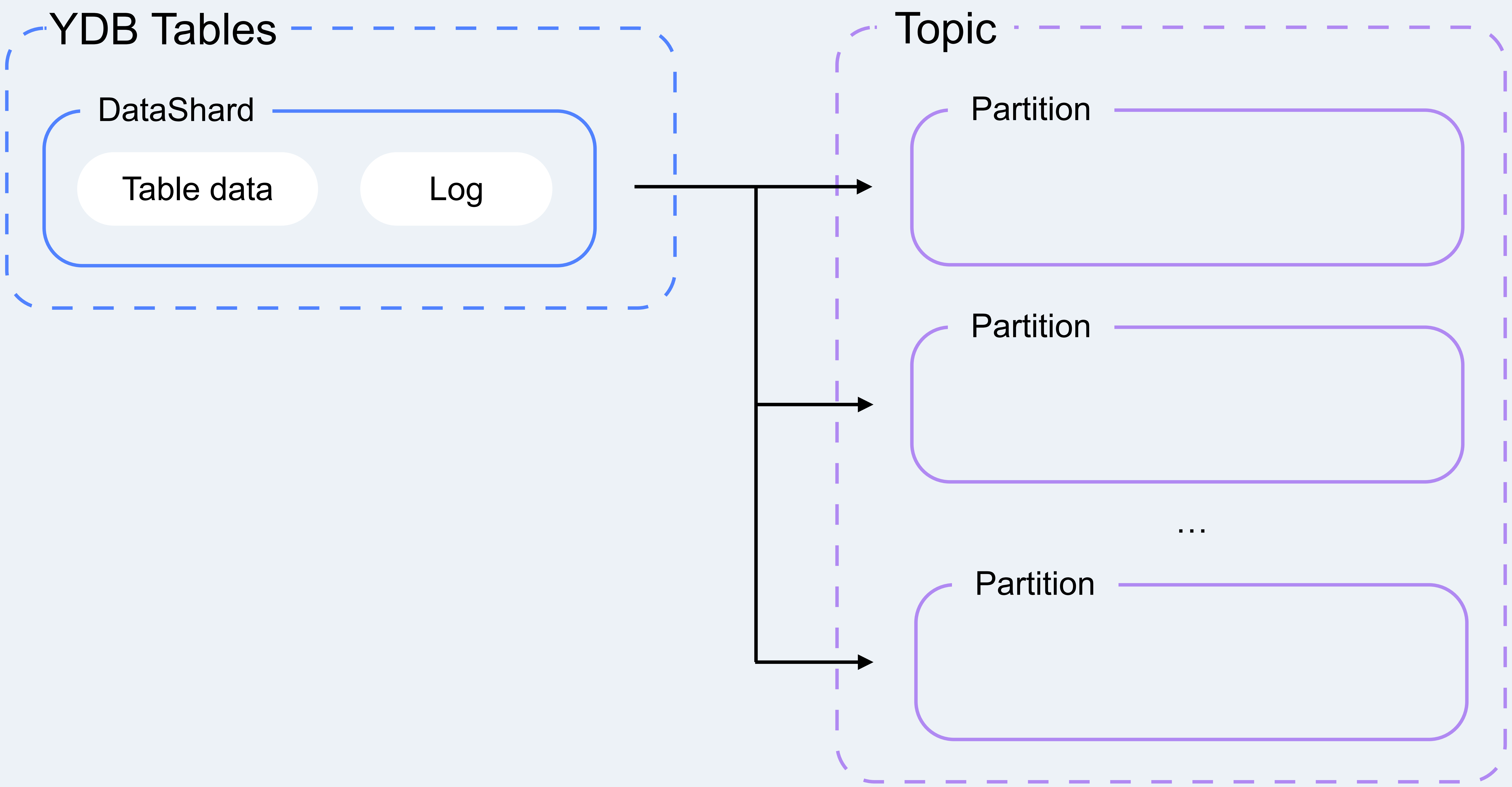
Topic — log storage in YDB

- Topic — implementation of Kafka-like topic in YDB
- Topic partition is another type of YDB tablet



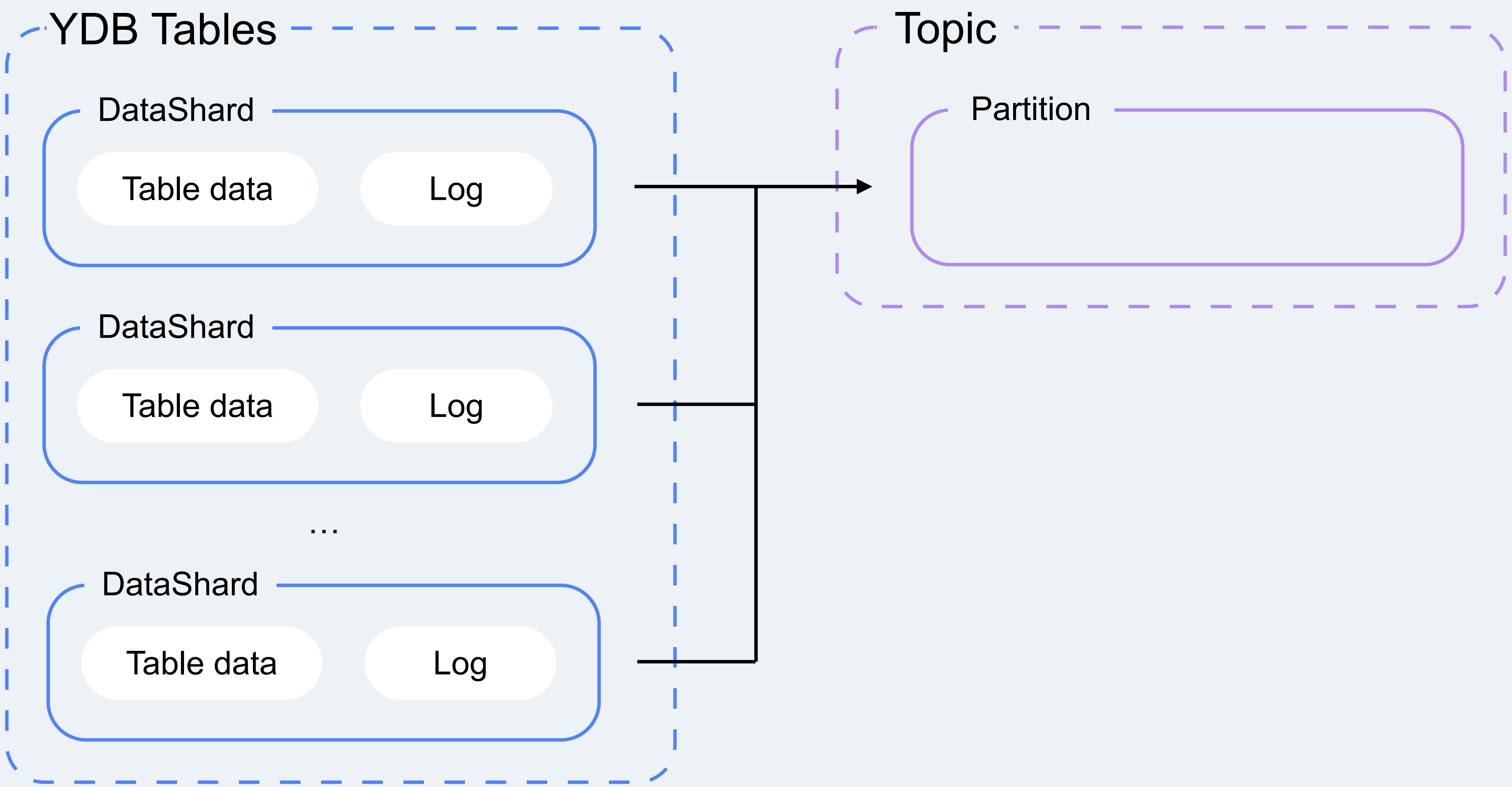
Additional benefits of using Topic

- Few (one) DataShards can generate large log
- E.g. frequently updated small set of keys
- A lot of partitions are required to store such log



Additional benefits of using Topic

- Vice versa, a lot of DataShards can generate a small log
- Few (one) partitions is enough to store such log



1. Intro and Problem Statement
2. YDB Architecture in 5 minutes
3. An Approach to Asynchronous Replication in YDB
4. Dealing with multiple logs
5. Distributed Transactions in YDB
6. Globally ordered log and consistency

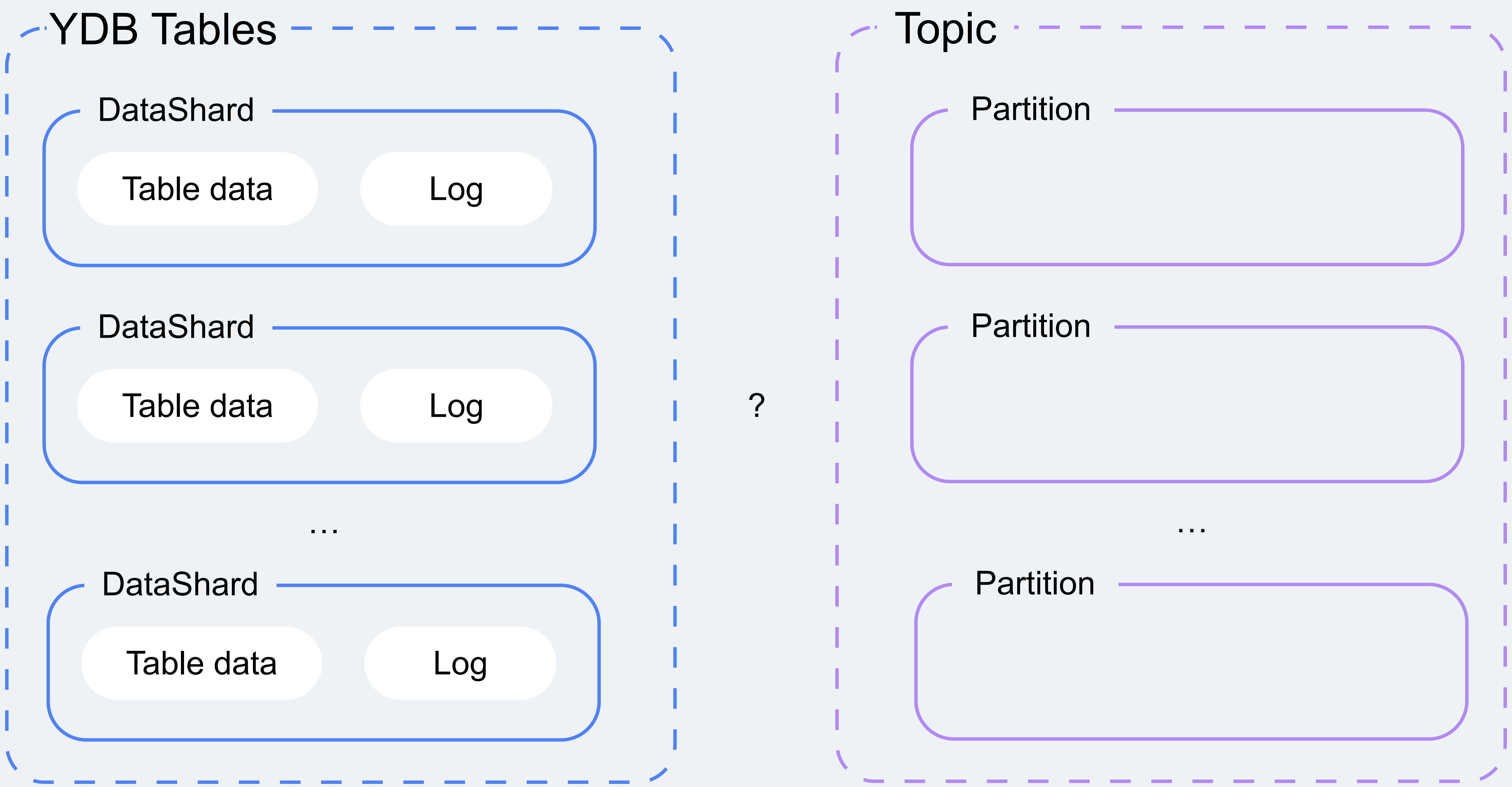
How to transfer log to Topic?

How are DataShards and Topic partitions related?

How to write and read globally ordered log in Topic partition?

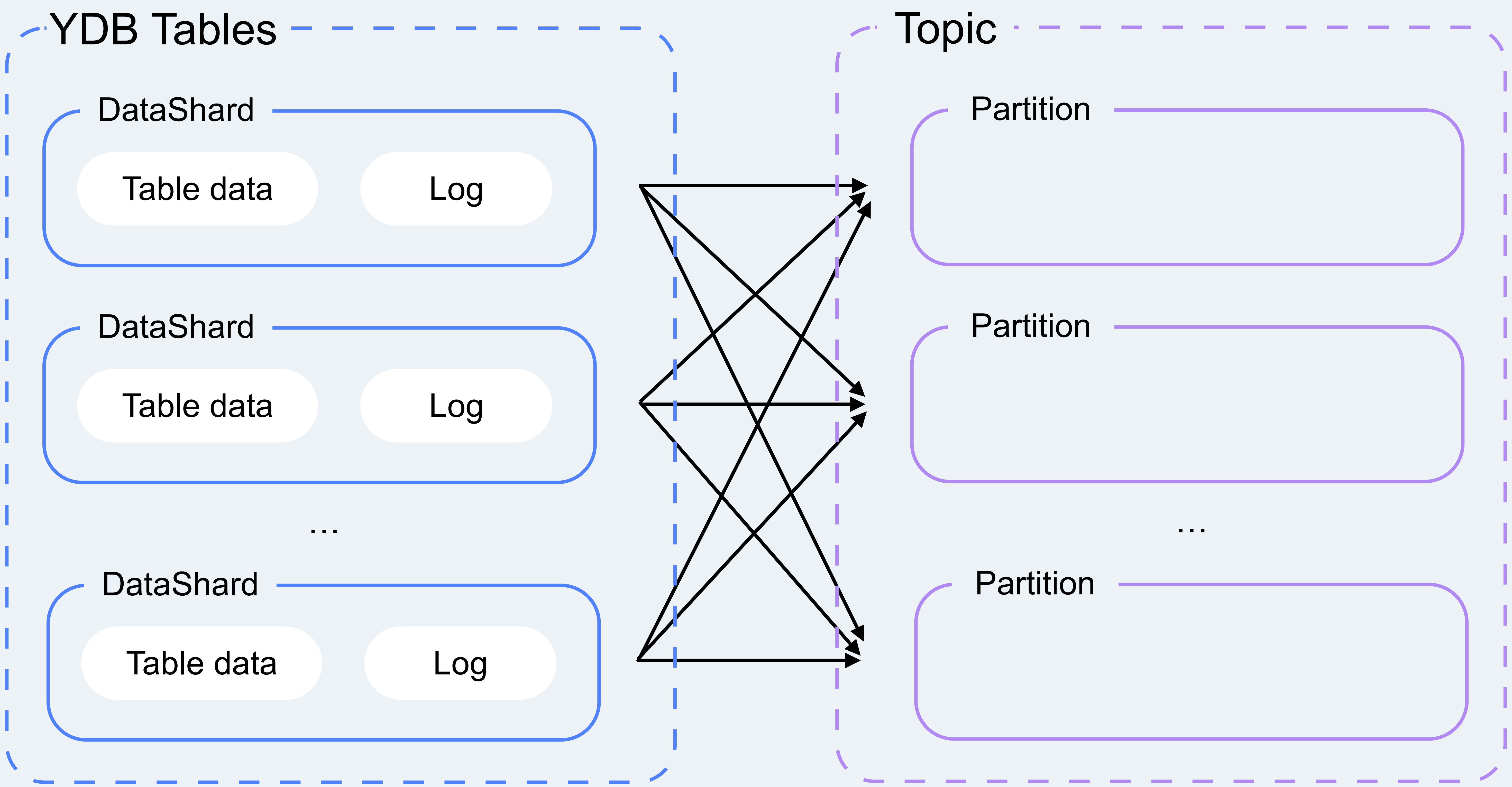
DataShard-Topic partition relation

- Random
- N:M (1:1, 1:M, N:1)
- Consistent hashing



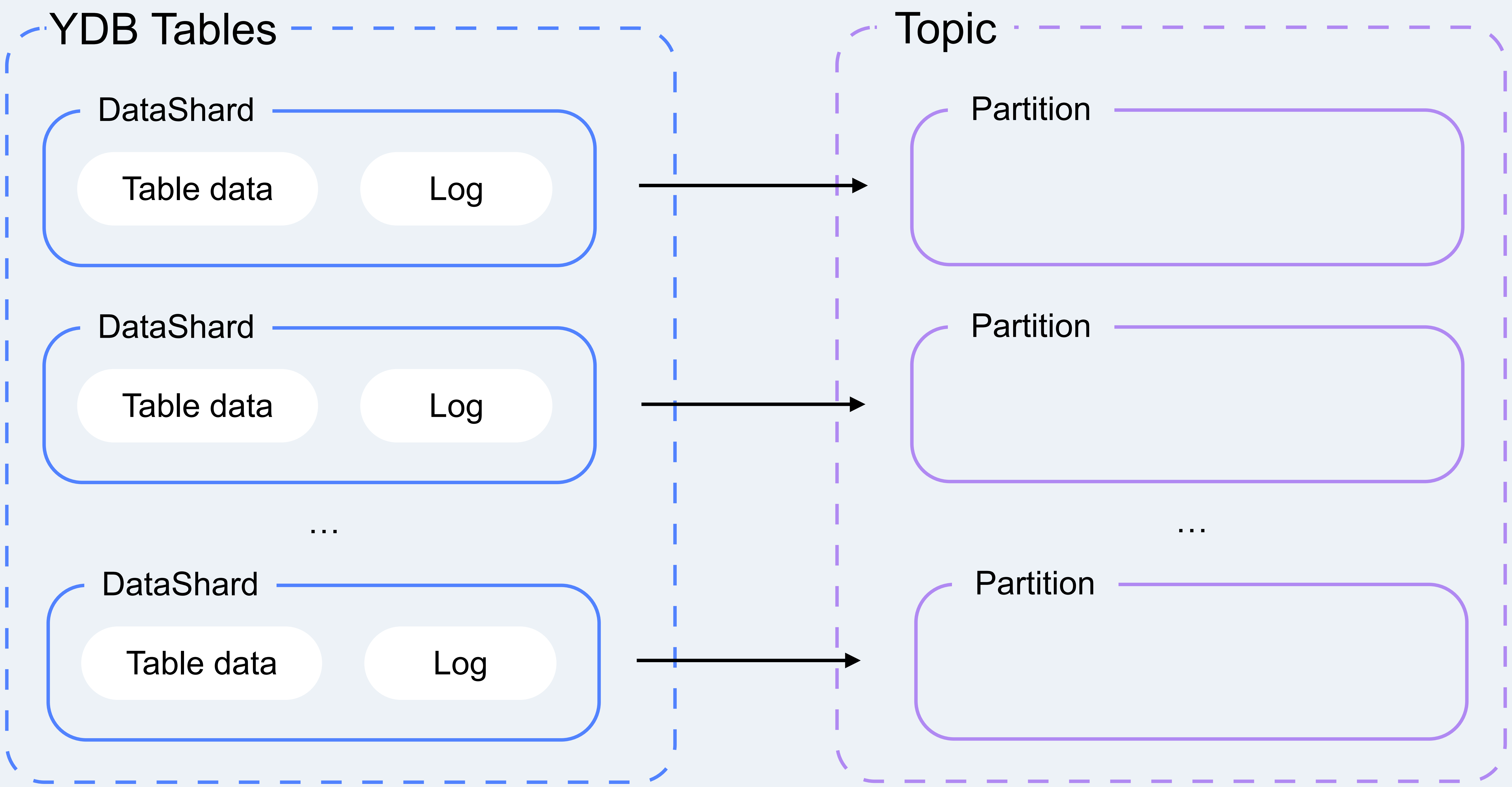
DataShard-Topic partition relation

- > Random
- N:M (1:1, 1:M, N:1)
- Consistent hashing



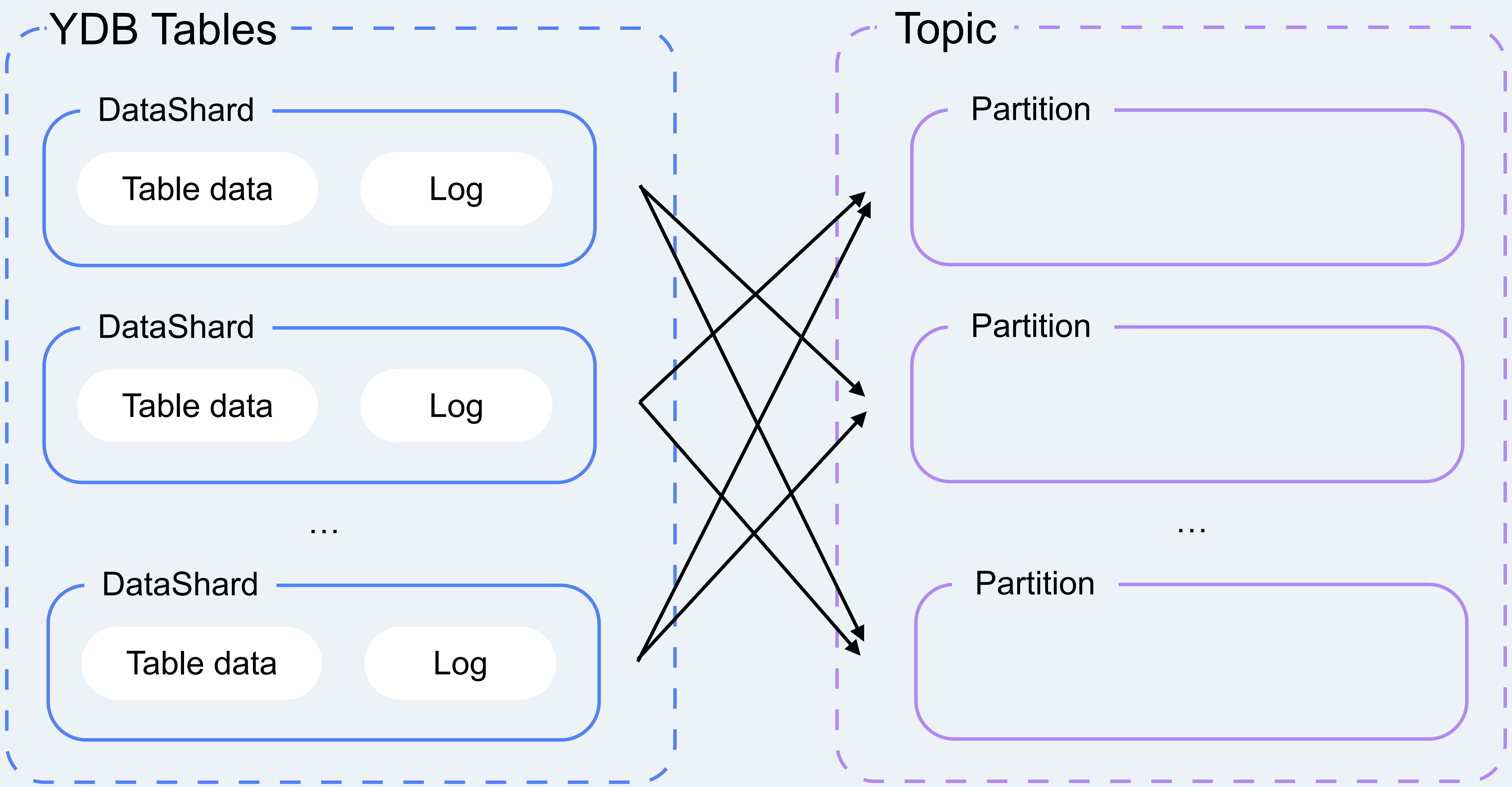
DataShard-Topic partition relation

- Random
- > N:M (1:1, 1:M, N:1)
- Consistent hashing



DataShard-Topic partition relation

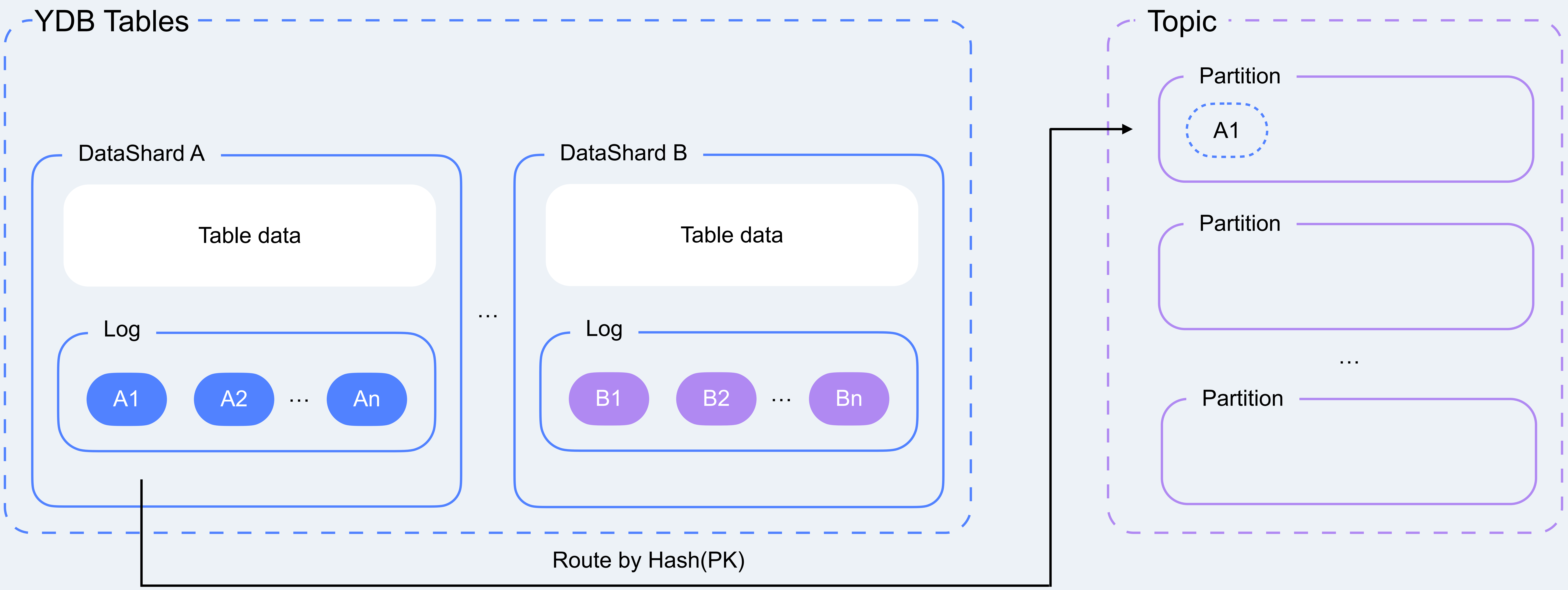
- Random
- N:M (1:1, 1:M, N:1)
- > Consistent hashing



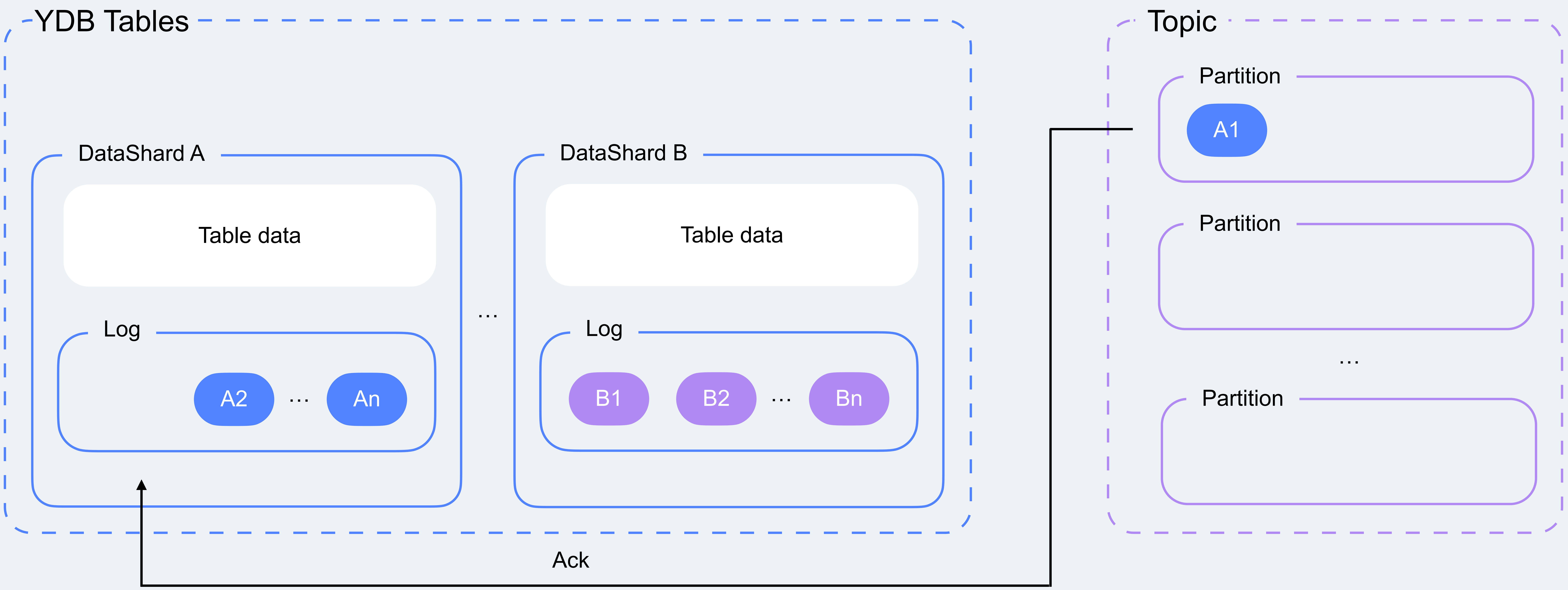
Why consistent hashing?

- Each modification of specific row appears in the same Topic partition
- Provides ordered row-level modifications (as well as N:M)
- Easier to implement than N:M

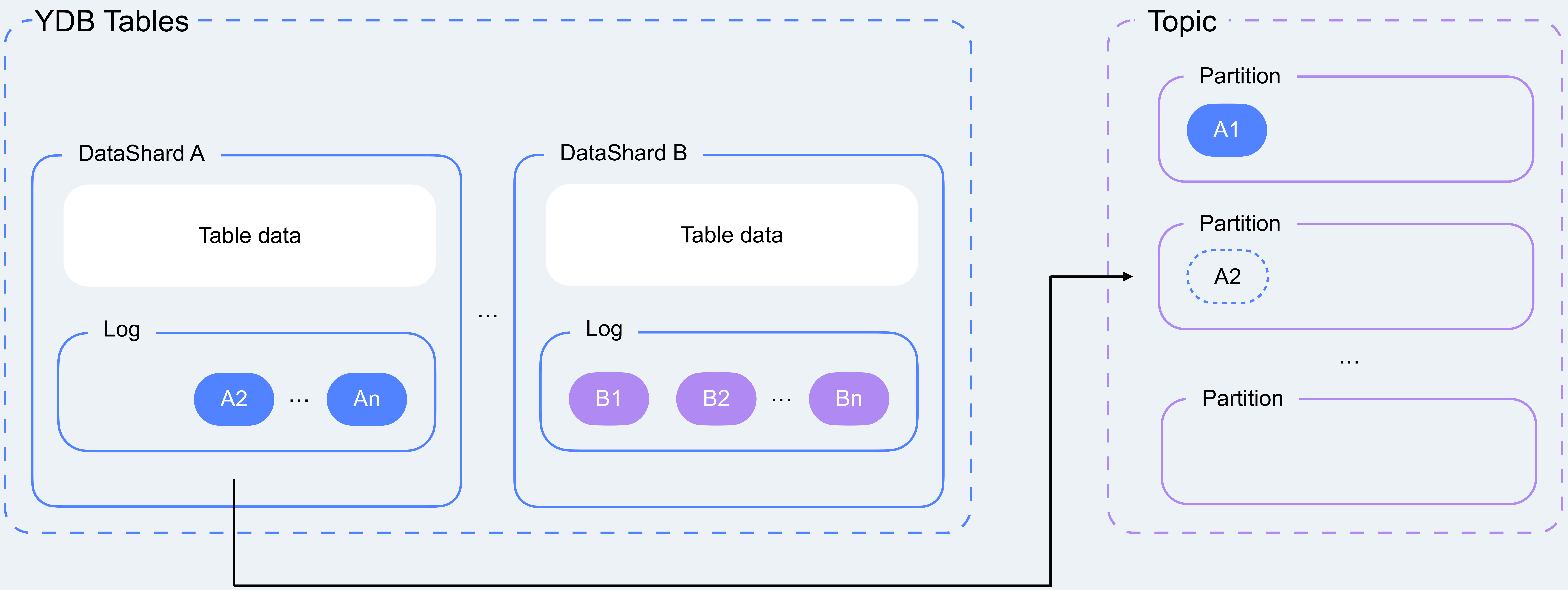
How to write to Topic?



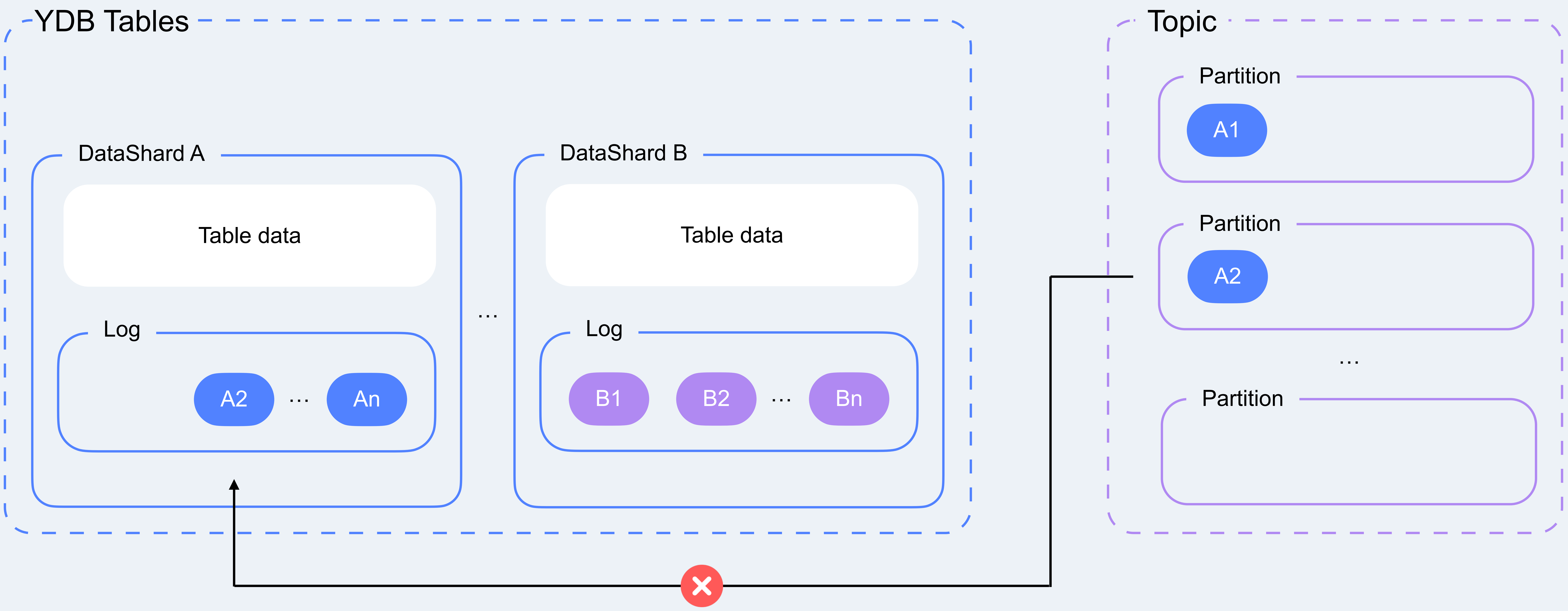
How to write to Topic?



How to write to Topic?



How to write to Topic?



Delivery problems

Tablets (DataShards, Topic partitions) can restart due to

Cluster updates

Hardware failures

Balancing

Connectivity issues (inside DB)

Consequences of delivery problems

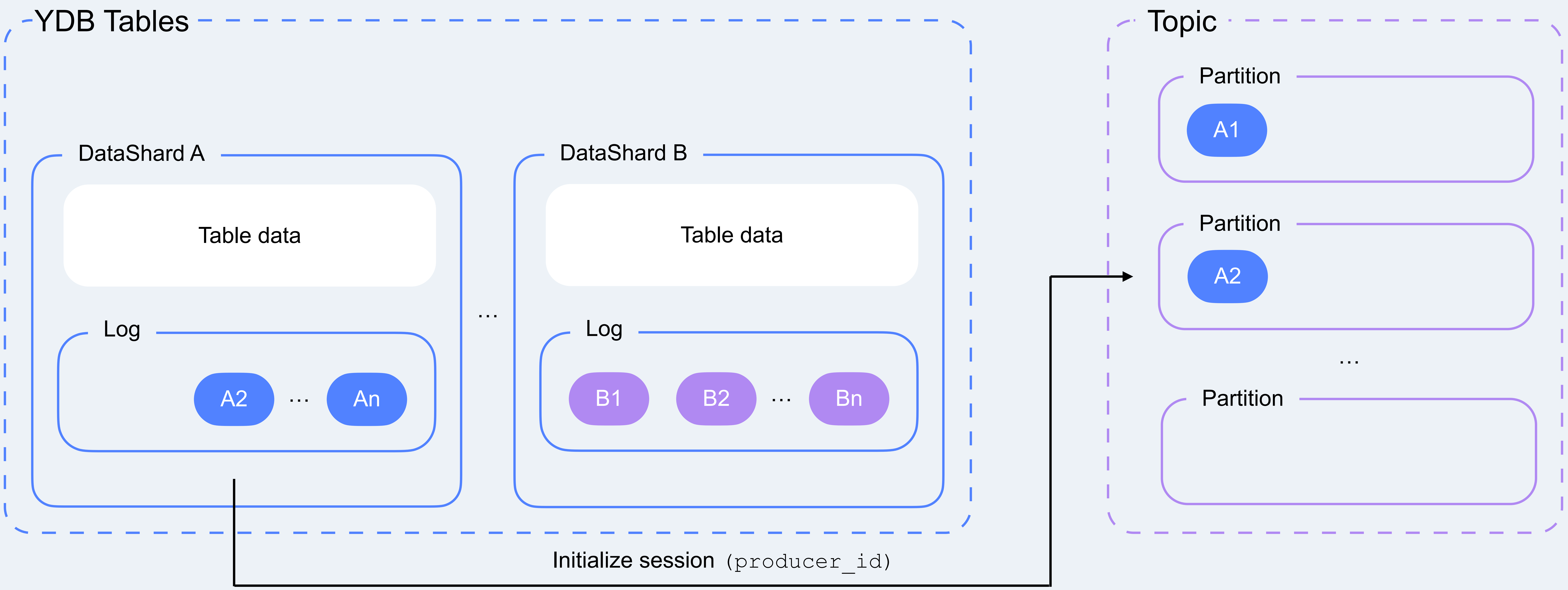
Duplicates (send, lost ack, resend)

Potential log growth at DataShards

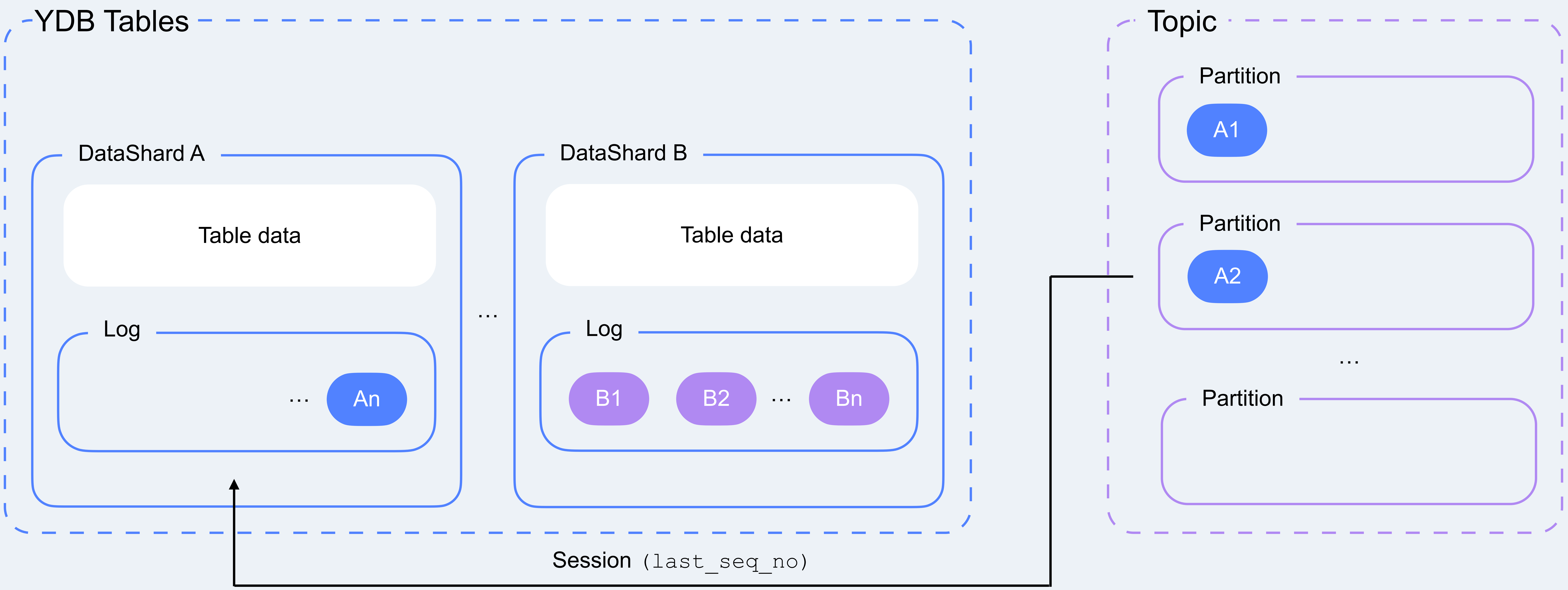
Delivery guarantees

- Each producer (DataShard) has its own `producer_id`
- Each log record from specific producer is identified by monotonic sequence number `seq_no`
- `(producer_id, seq_no)` pair allows to deduplicate records and achieve exactly-once guarantee

Write session



Write session



Log growth prevention

DataShard controls size of its log

When the log size reaches the limit, DataShard activates backpressure mechanism (until the log gets smaller)

Tablets normally restore availability quickly, so backpressure is a last resort

Transferring log to Topic

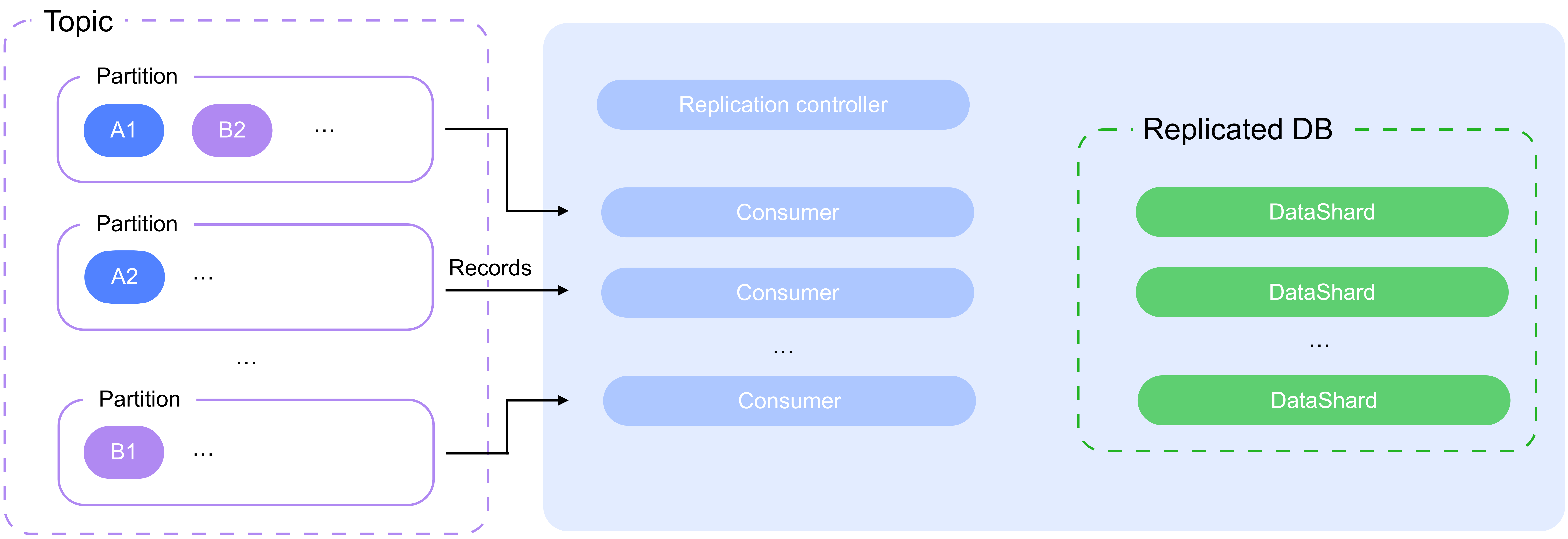
Log records routed by hash from table's primary key

For each row that is modified in a YDB table, the log records appear in the same Topic partition as the actual modifications to the row

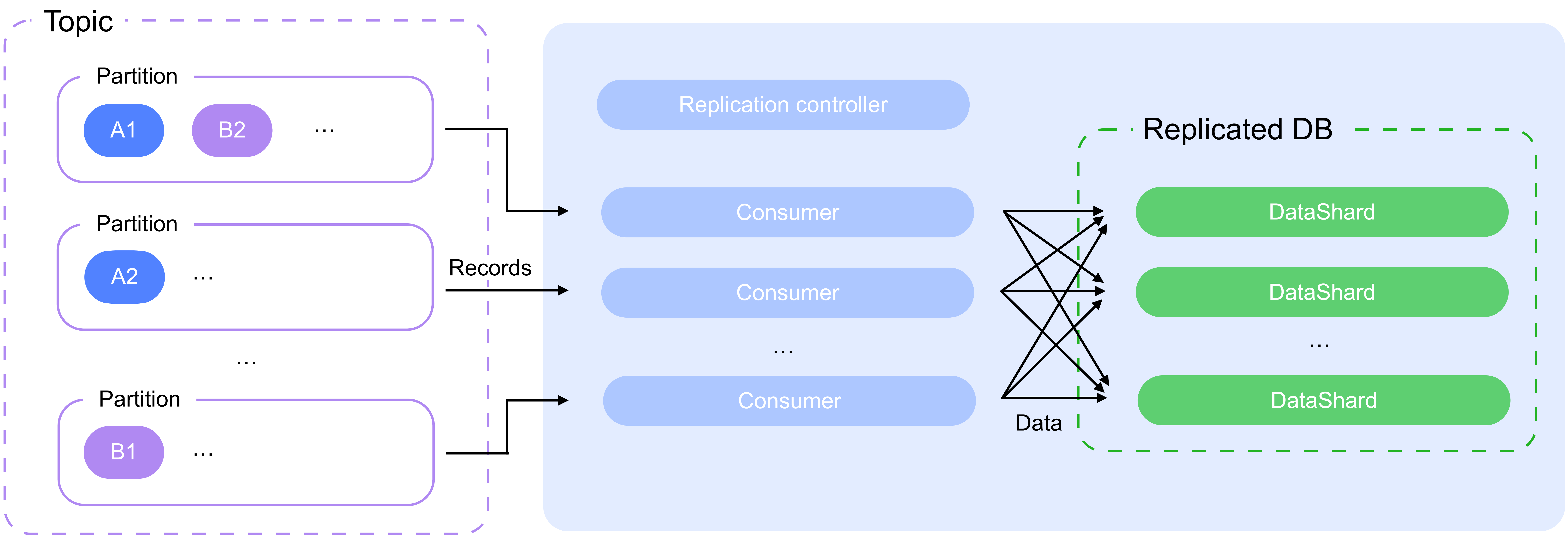
Exactly-once guarantee

Size of DataShard log is still reasonable

Replication from Topic



Replication from Topic



Replication from Topic

- Replication controller creates a consumer for each Topic partition
- Consumer reads the partition log and writes data to a set of DataShards (primary key routing)
- Controller periodically receives and remembers consumer's progress

1. Intro and Problem Statement
2. YDB Architecture in 5 minutes
3. An Approach to Asynchronous Replication in YDB
4. Dealing with multiple logs
5. Distributed Transactions in YDB
6. Globally ordered log and consistency

Distributed Transaction Example

ID	Value1	Value2
GX008	8921	1114
GX278	827	9
GY045	654	345
SK720	3445	3456
SM527	7668	7643
UA628	72	3928

Key	Data
82	8921
283	827
346	654
1273	3445

```
UPDATE table1 SET Value1=3845 WHERE Id="GY045"
UPDATE table2 SET Data=Data+1 WHERE Key=346;
COMMIT;
```

How to Implement Distributed Transactions?

2PC (Two-phase Commit)

The most standard way to implement distributed transactions

Disadvantages: low throughput on high contention

YDB adapts Calvin protocol for distributed transaction processing

Calvin: Fast Distributed Transactions for Partitioned Database Systems by Daniel J. Abadi, Alexander Thomson

Calvin allows nonblocking execution of deterministic transactions

Calvin itself is not enough to execute arbitrary transaction, so YDBs transaction processing is more than just Calvin

What a Deterministic Transaction is?

Deterministic transaction knows
it read/write set

```
read A
read B
write C = value(A) + value(B)
```

What a Deterministic Transaction is?

Deterministic transaction knows
it read/write set

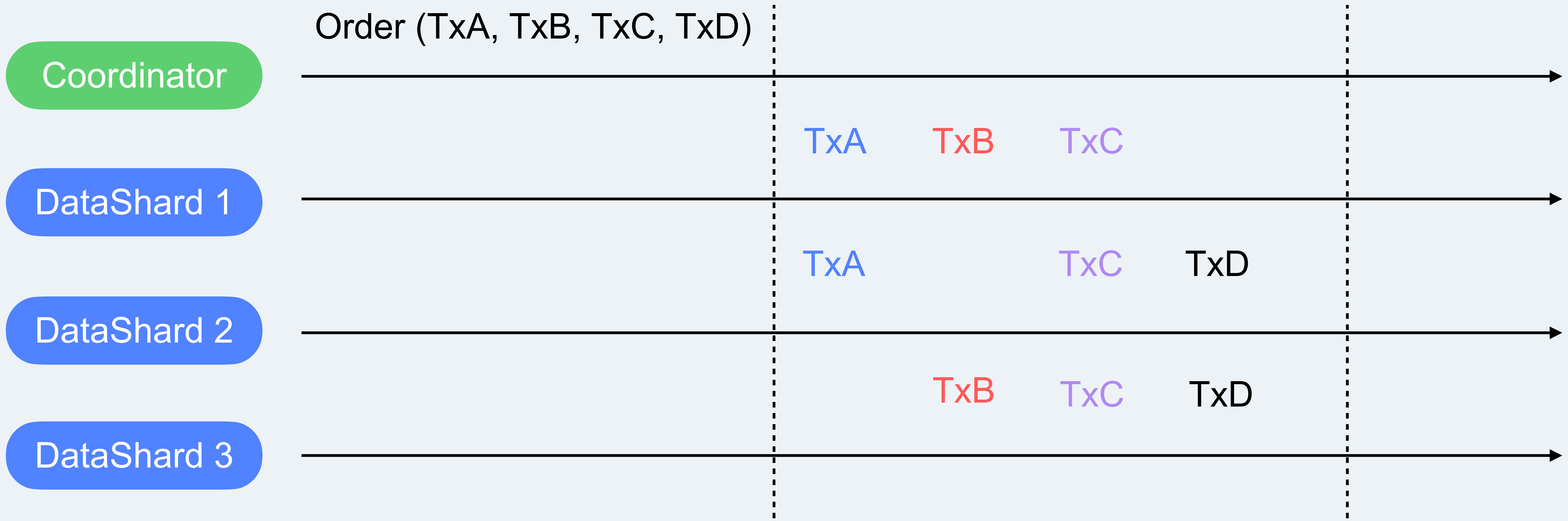
```
read A
read B
write C = value(A) + value(B)
```

Not all transactions are deterministic.
Example of none-deterministic transaction

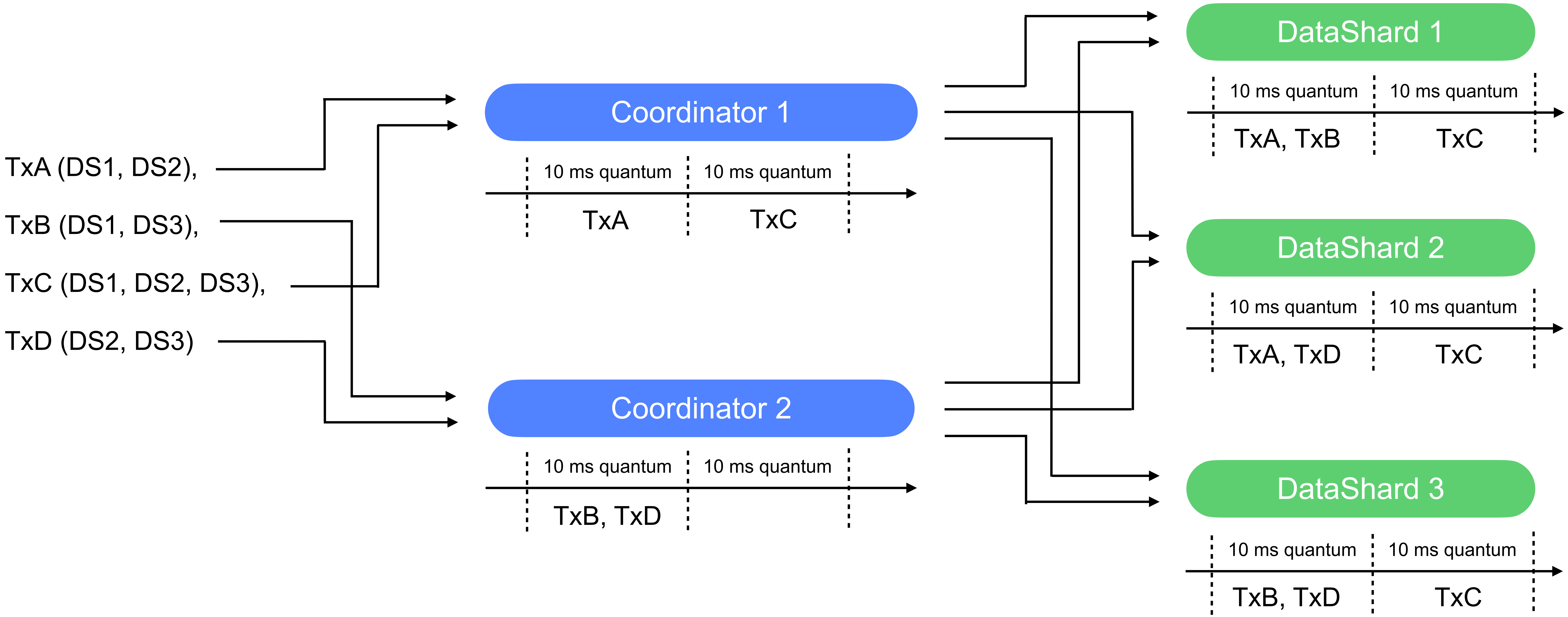
```
read A

read value(A)
read B
write C = value(value(A)) + value(B)
```

How Calvin Executes Deterministic Transactions?

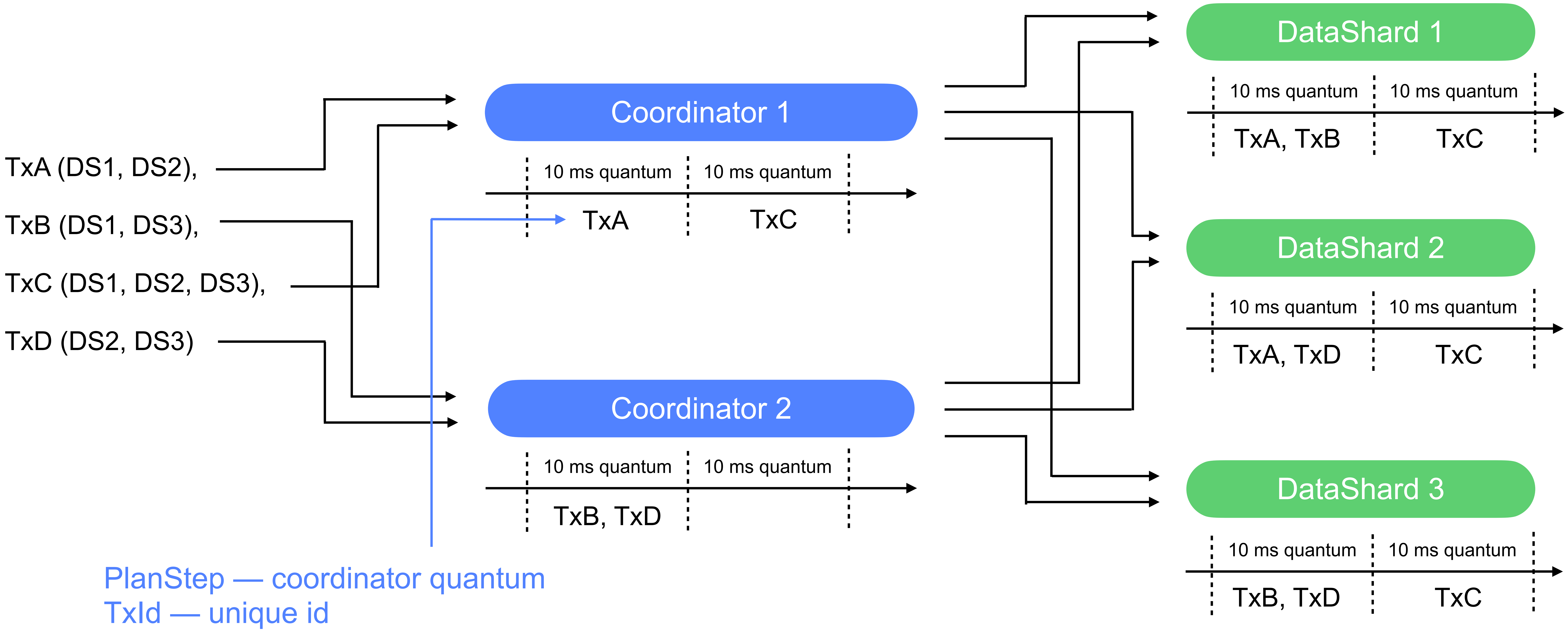
Say, we have incoming transactions: TxA(DS1, DS2), TxB(DS1, DS3), TxC(DS1, DS2, DS3), TxD(DS2,DS3). Calvin: If Coordinator arranges incoming transactions, then there will be no conflicts and we will get serializable isolation



YDB’s Multiple Coordinators

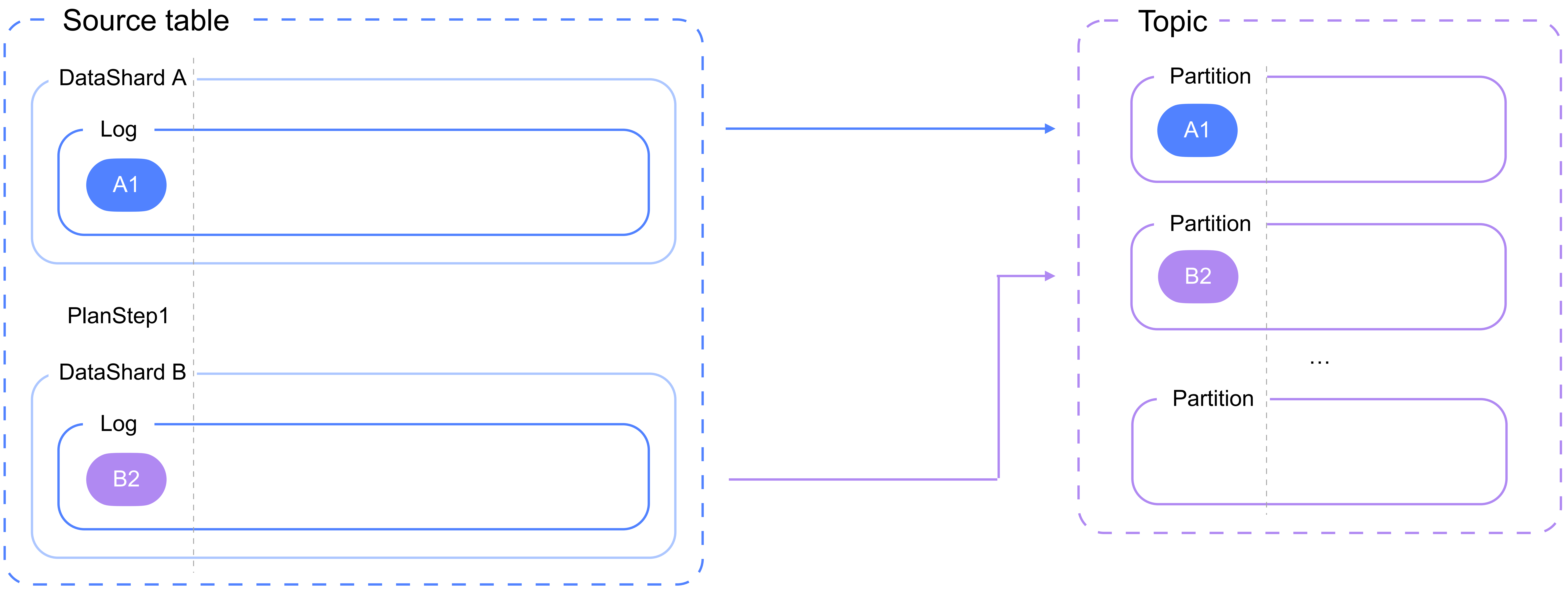


YDB’s Multiple Coordinators

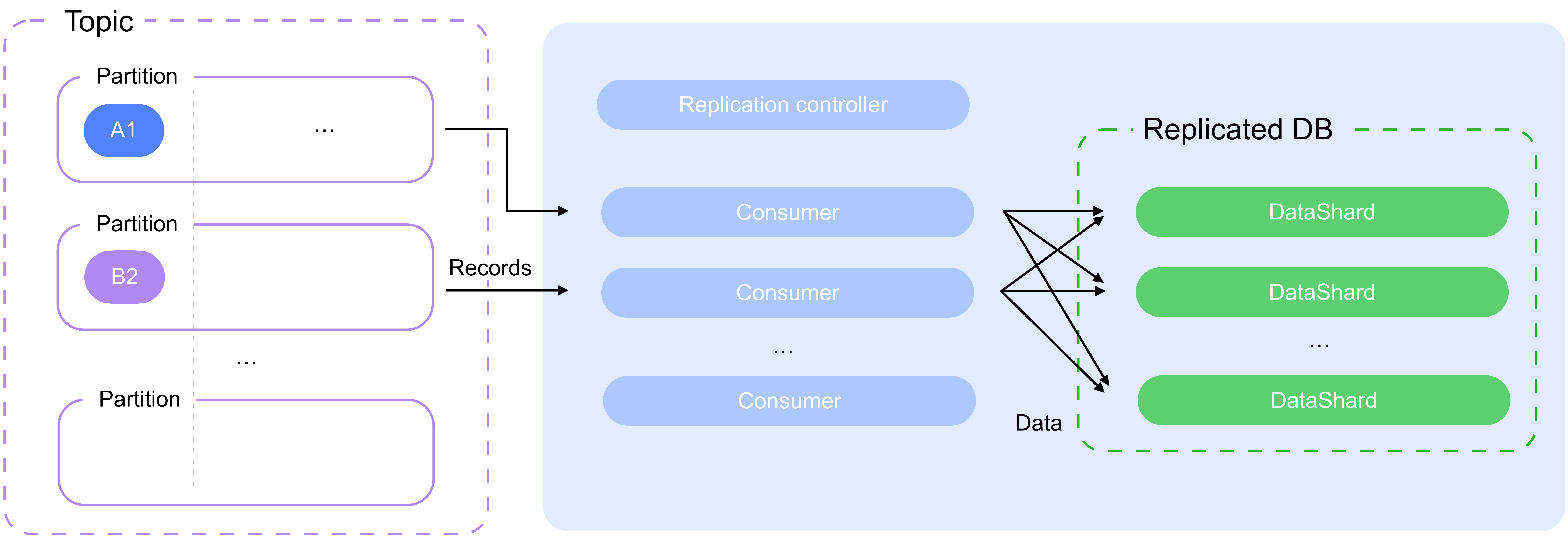


1. Intro and Problem Statement
2. YDB Architecture in 5 minutes
3. An Approach to Asynchronous Replication in YDB
4. Dealing with multiple logs
5. Distributed Transactions in YDB
6. Globally ordered log and consistency

How to write globally ordered log to Topic?



Will it be consistent?

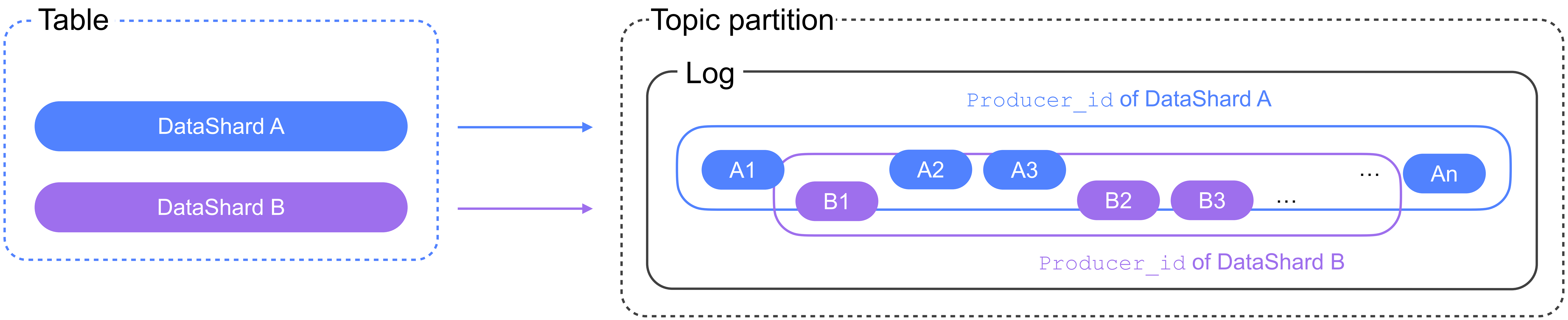


How to achieve consistency?

Client on replicated-side must be sure that he has received all records for a certain PlanStep

Therefore, the client needs to know the list of DataShards (producers) that write to the Topic

More about producers



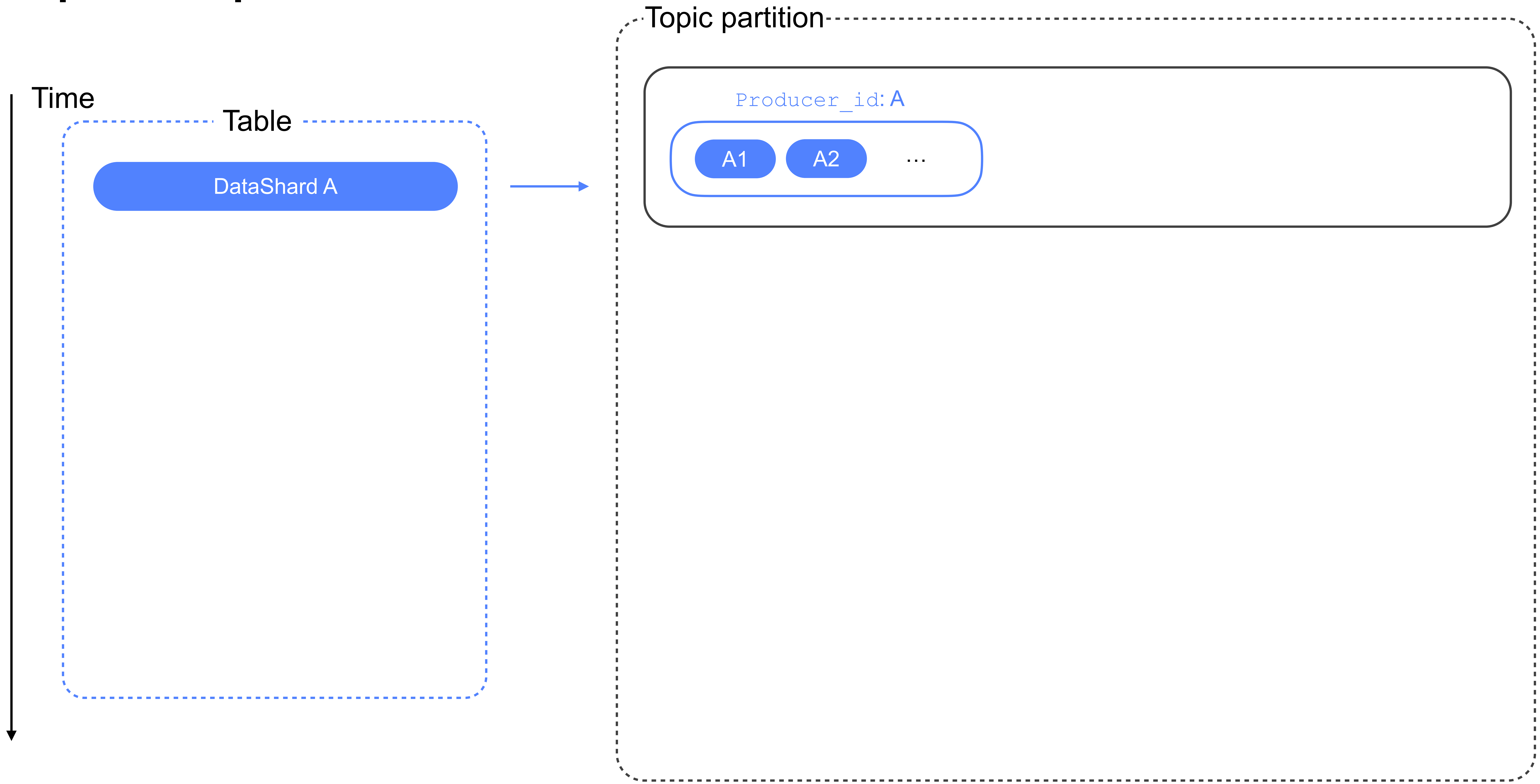
More about producers

The Topic partition log consists of records of all its producers (DataShards)

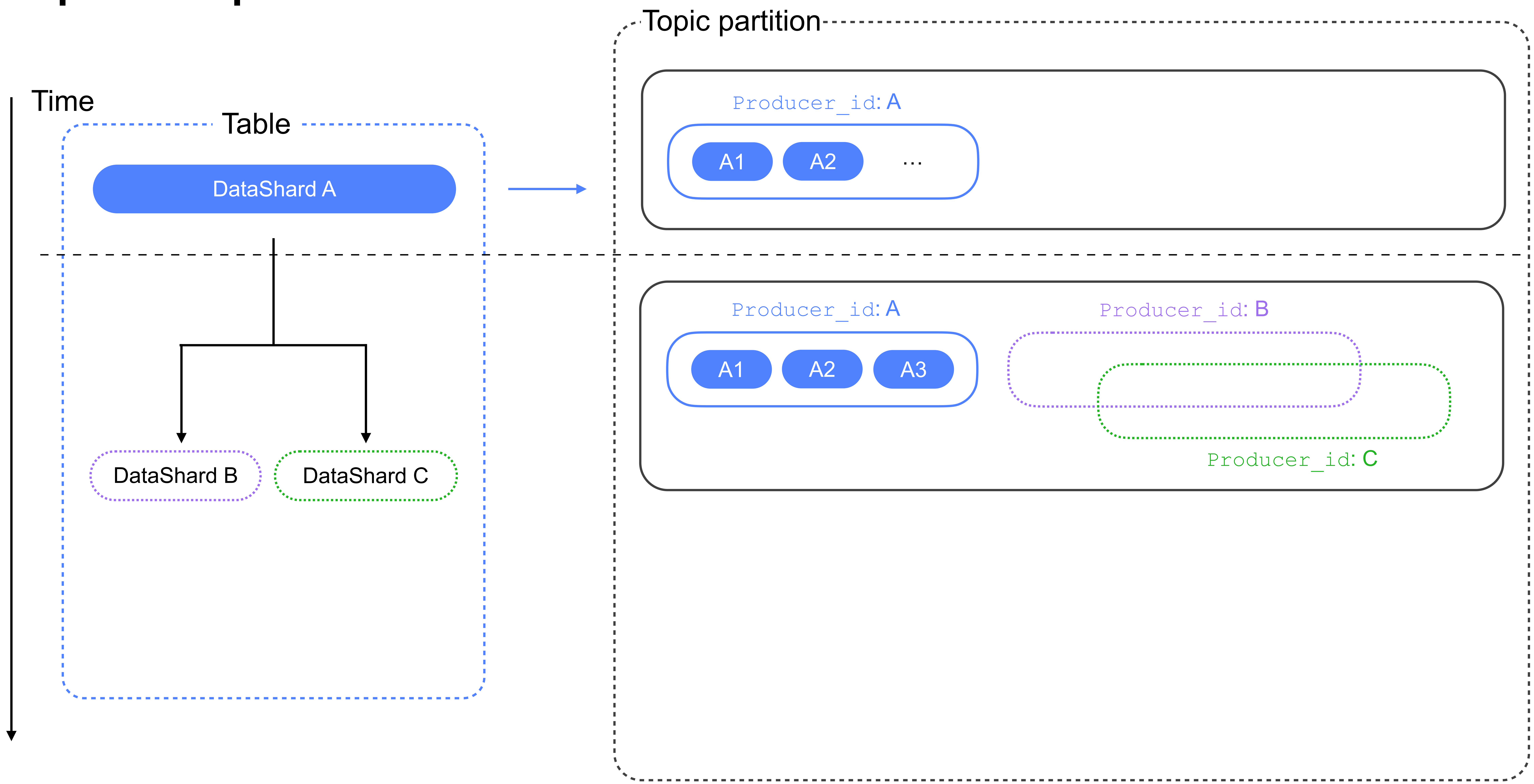
Producer has its own `producer_id`

Topic partition knows list of producers at any moment of time

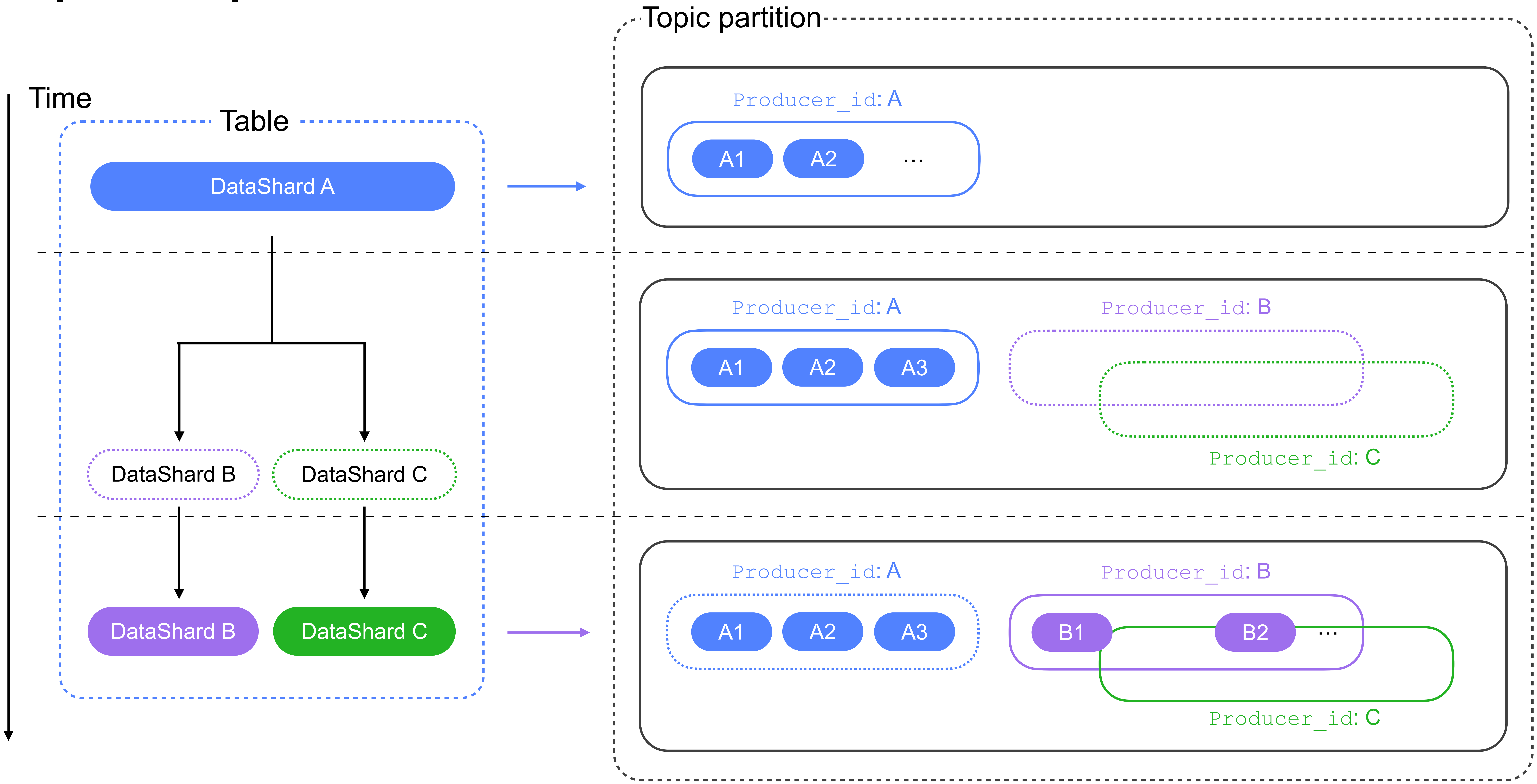
Split of producer



Split of producer



Split of producer

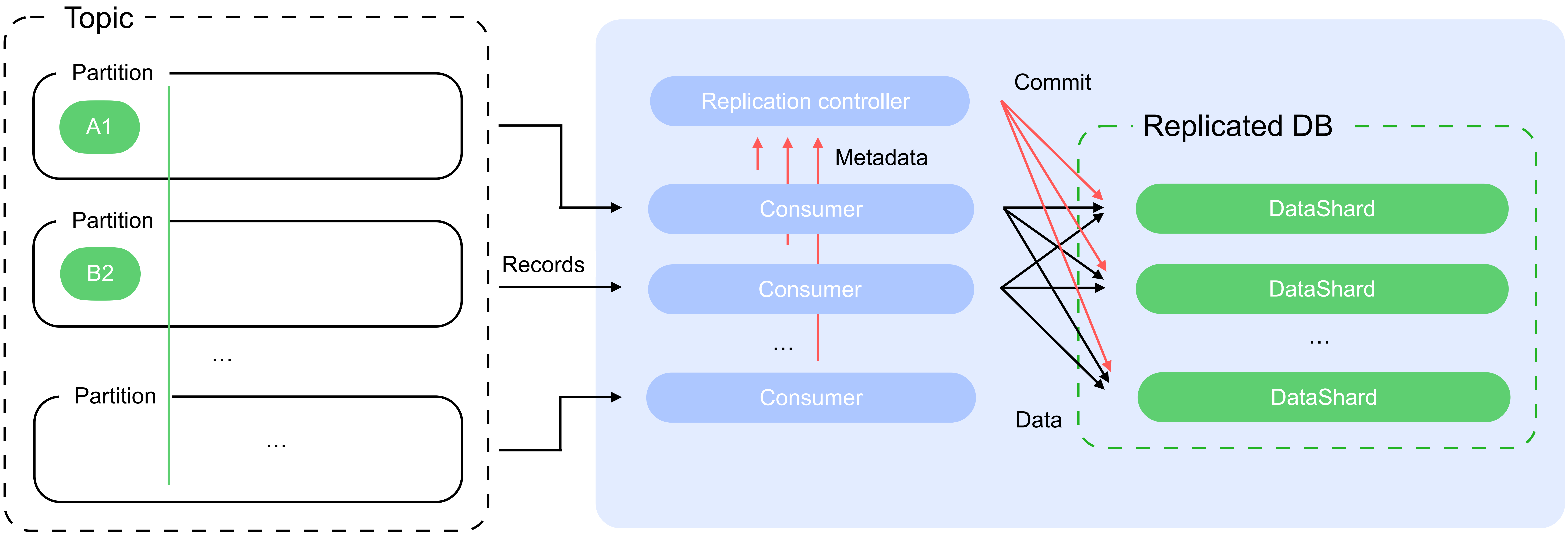


List of producers

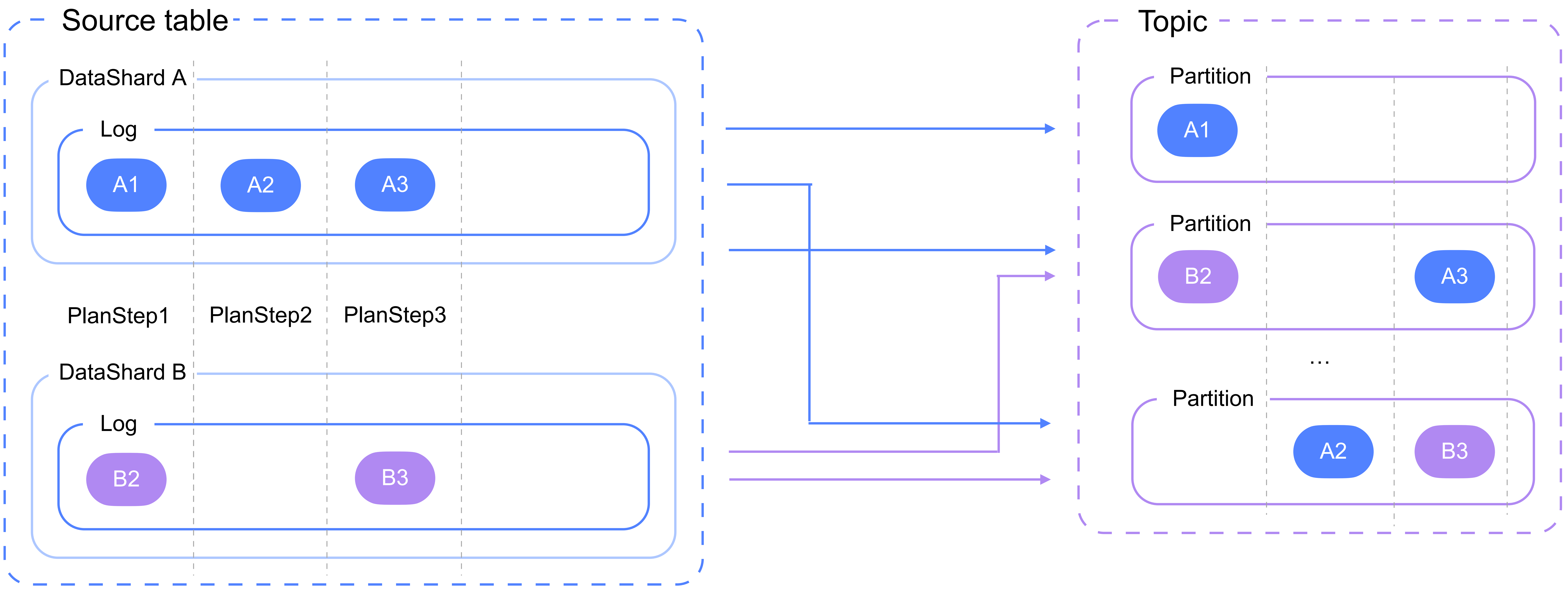
The list of producers is known, kept up to date (during split or merge) and available to clients

This information helps to determine whether all records for a certain PlanStep have been received or not

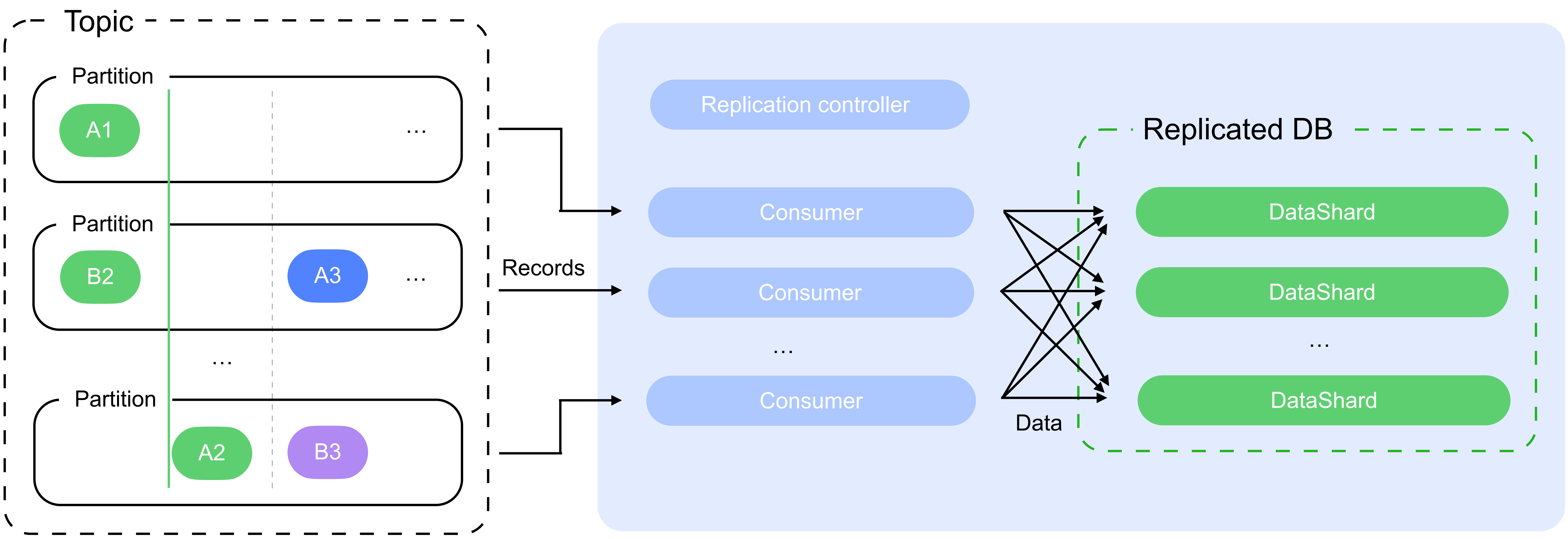
Now it's consistent



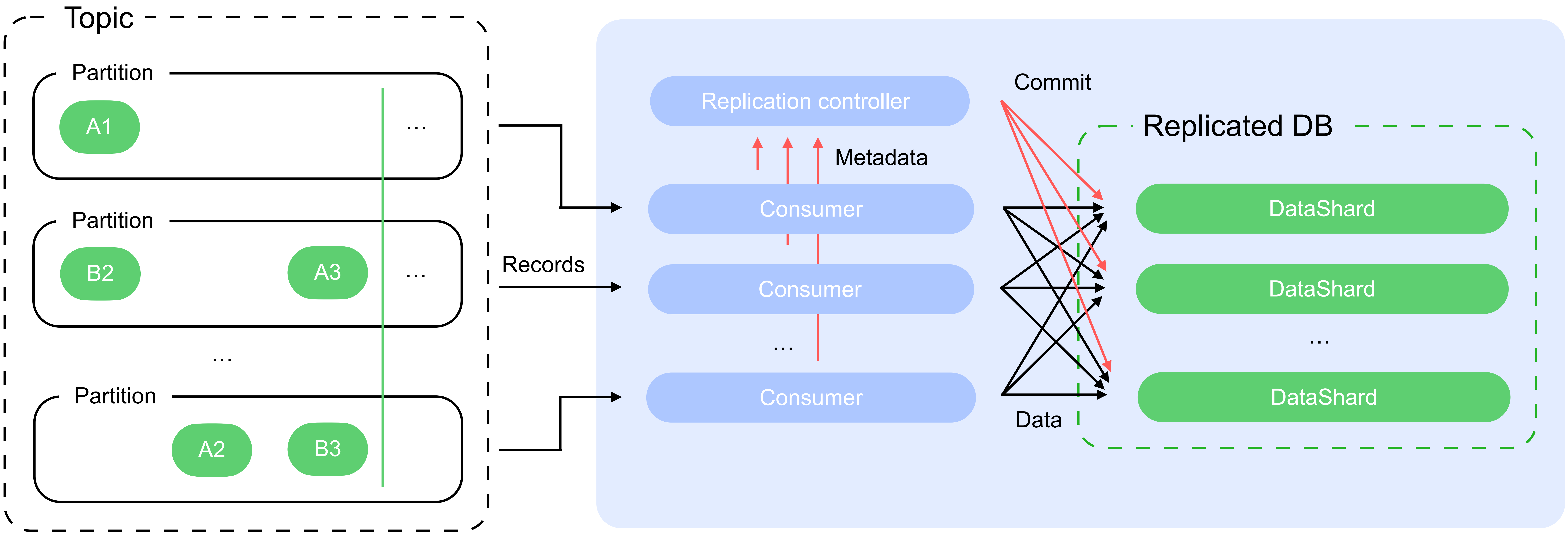
What if nothing has changed?



What if nothing has changed?



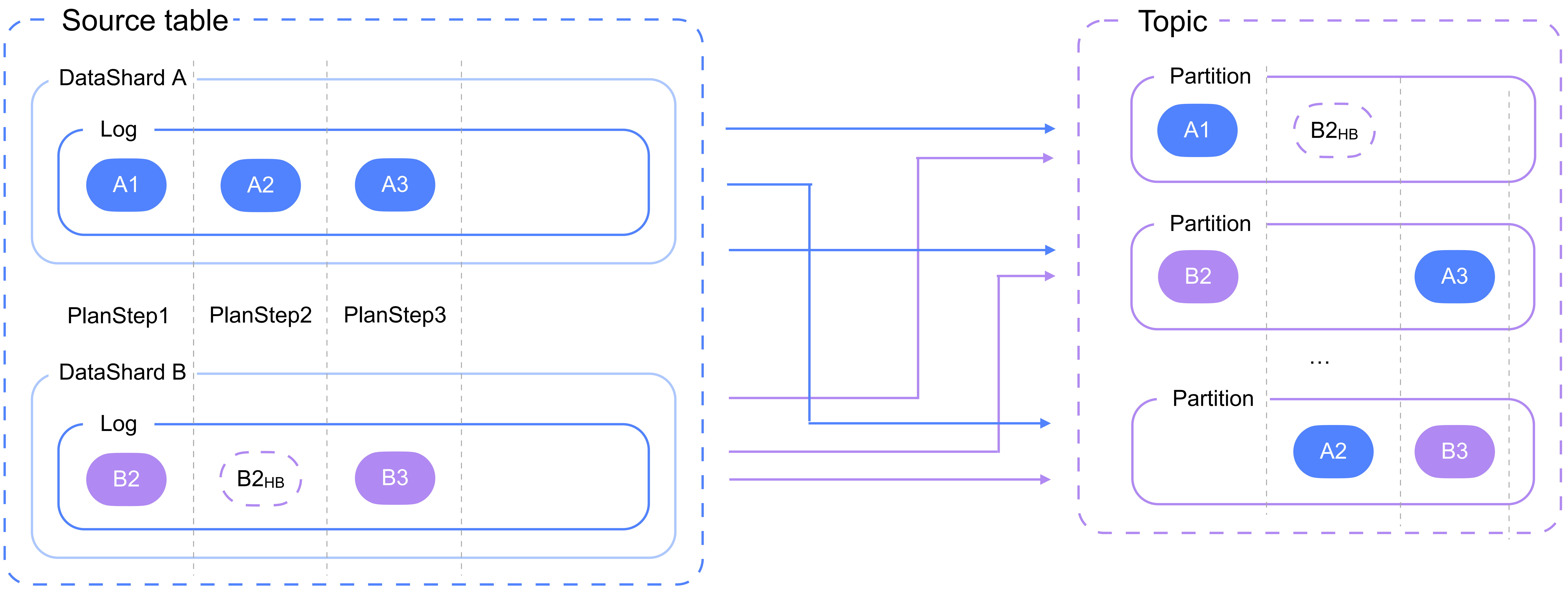
What if nothing has changed?



Gaps in the log

- Gaps do not allow to promote edge of commited data
- We have to wait until changes occur in the all DataShards
- Replication delays are increasing

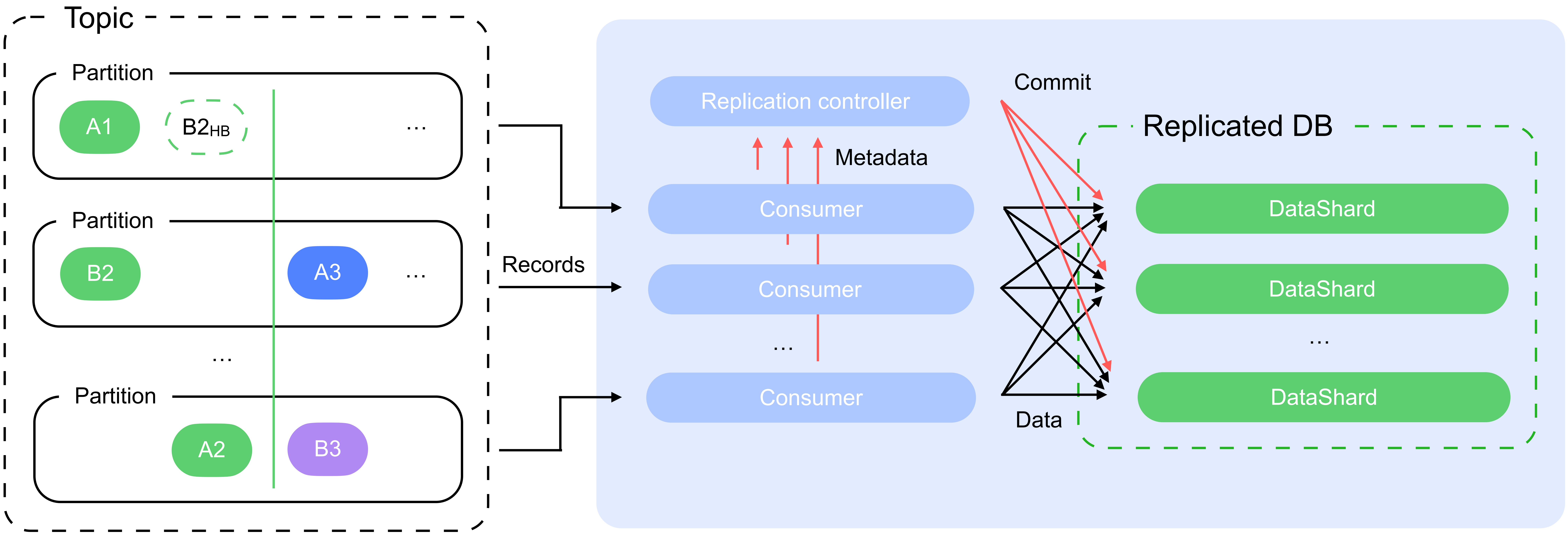
If nothing has changed, just send something



Sending heartbeats

- DataShard should send records every PlanStep: with data (something happened), or heartbeat (nothing has changed)
- Heartbeat can be sent to any Topic partition
- Heartbeats help to promote edge of commited data

Commited edge promotion using heartbeats



Conclusion

https://ydb.tech

@YDBPlatform

@YDBPlatform

@yandexdatabase_ru

Scalable multi-level log

Small log at DataShards

Large log at Topic partitions

Available as a part of Change Data Capture

Global consistency with adequate delays

Records sorted in global time by PlanStep

Heartbeats help to get rid of gaps in global time

Will be available in the next major version

Topic Partitions Split/Merge

Elastic topics

Automatically adjust number of partitions like tables do

Next steps, contributions are welcome