An Approach to Unite Tables and Persistent Queues in One System

Elena Kalinina,
Technical Project Manager,
YDB

Goals of this talk?

1

Talk about the YDB-platform, which unites OLTP processing, work with persistent queues and OLAP processing

2

Demonstrate an approach of uniting tables and persistent queues in one system

3

Dive into our transactions which combine changes in tables and queues in ACID way

YDB – what’s this?

YDB

Transactional Processing

OLTP

- Distributed storage
- Petabytes of data
- Millions of transactions per second

YDB Topics

Persistent queues
(like Apache Kafka)

- Delivery your data between apps
- Exactly once / At least once guarantees
- High loads of gigabytes per second

Analytical Processing

OLAP

- Analytical reports with high performance
- No compromises with availability

YDB is an open source solution published under Apache 2.0 license

YDB platform: main features

- Row-oriented tables for OLTP
- Column-oriented tables for OLAP
- YDB Topics for persistent queues

- Fault-tolerant configuration
  Survives disk, node, rack, or even data center outages
- Automatic disaster recovery
  Minimum latency disruptions for applications
- Horizontal scalability of storage and compute layers

- Rich SQL dialect (YQL)
- ACID transactions

YDB topics — what's this?

YDB Topics is a realization of persistent queues within YDB

Main features

- Reliability
- Work with big amounts of data
  (up to hundreds of gigabytes
  per second, storing petabytes of data)

Based on YDB platform

- Change Data Capture (CDC)
- Transactions with topics and tables

API

- YDB Topic API

  C++ SDK, Java SDK, Python SDK, Go SDK

  All YDB Topics features are supported:
  - Exactly once delivery
  - Transactions tables-topics
  - Topics autopartitioning

- Apache Kafka API

  Now you can use kafka cli, kafka connect,...

  And also integrate with logstash, fluentbit,...

Transactions with Tables and Topics: Examples

Example 1: We need to ”enrich” information about an event with a table data

- Read ”simple” event info from the Topic 1
- Read the reference data from the table
- Write ”rich” event info into the Topic 2

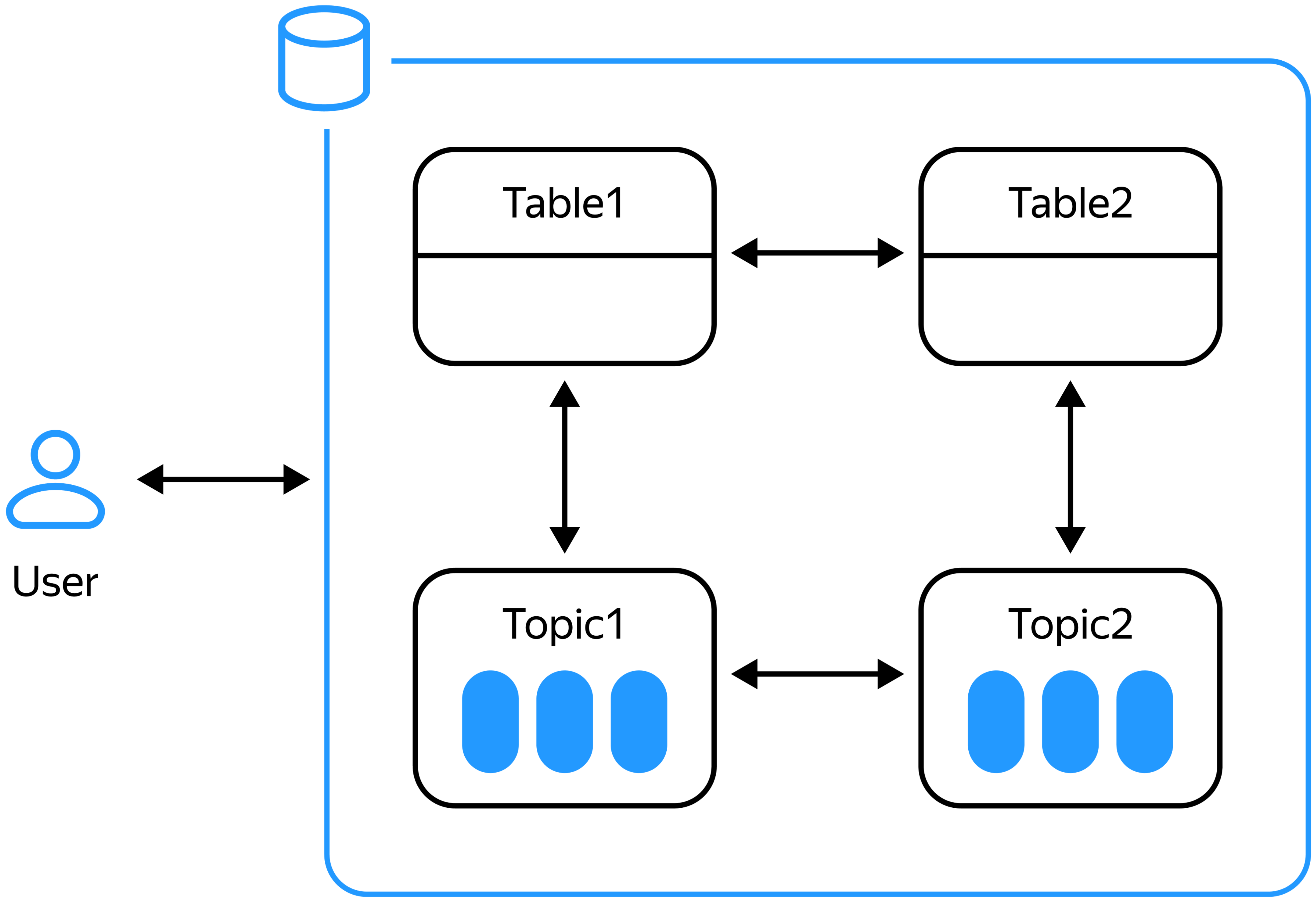
Transactions with Tables and Topics: Examples

Example 2: Resharding task. Input topic has all events and we need to distribute these events between partitions of output topic by some rule.

- Read an event from input topic
- Define output topic partition by event data
- Write an event to the appropriate output topic partition

Transactions with Tables and Topics

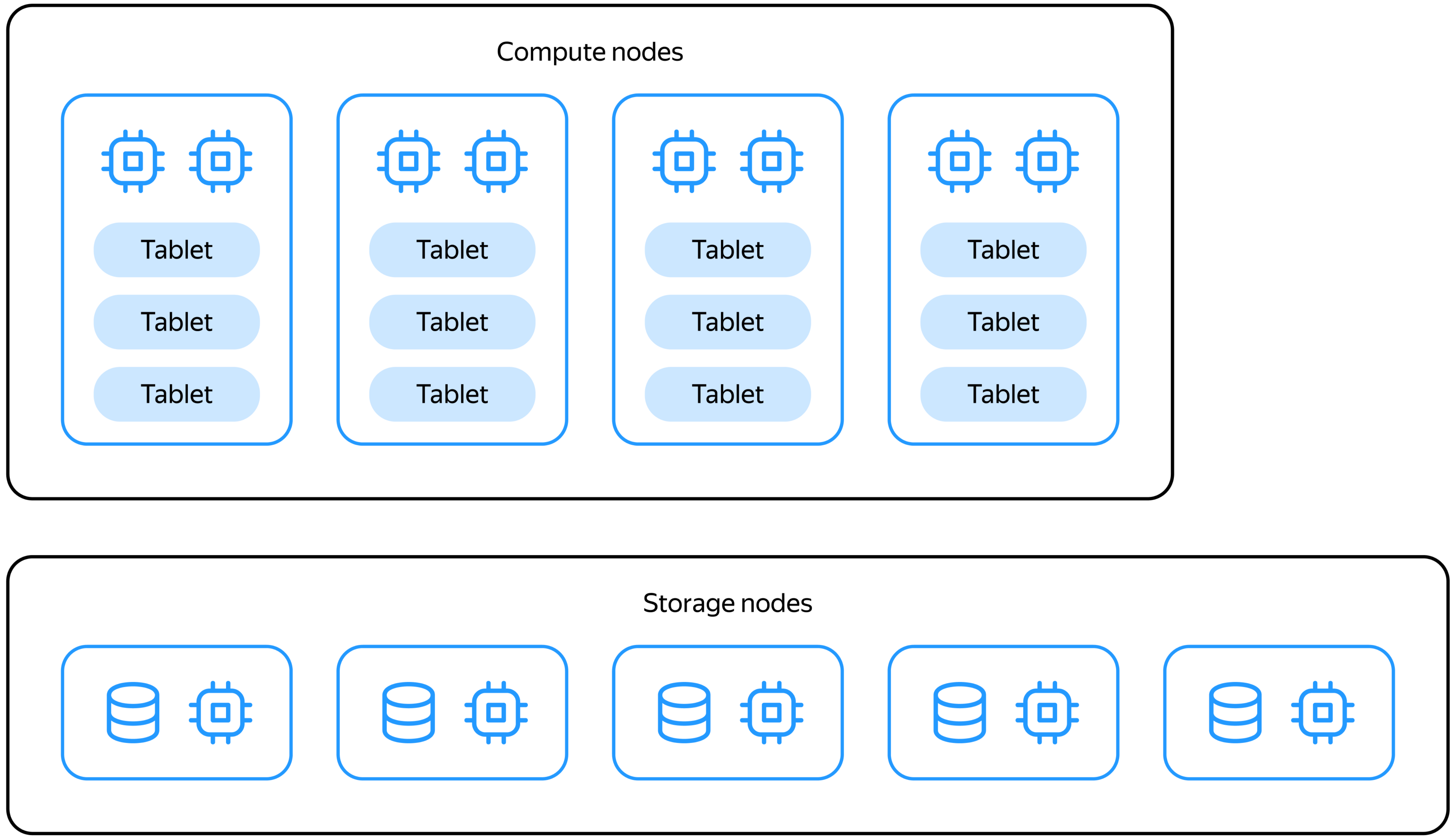
- Read from topic and write to table
- Read from table and write to topic
- Read from one topic and write into another topic
- … And all combinations of these base variants



YDB Platform: Technical aspects

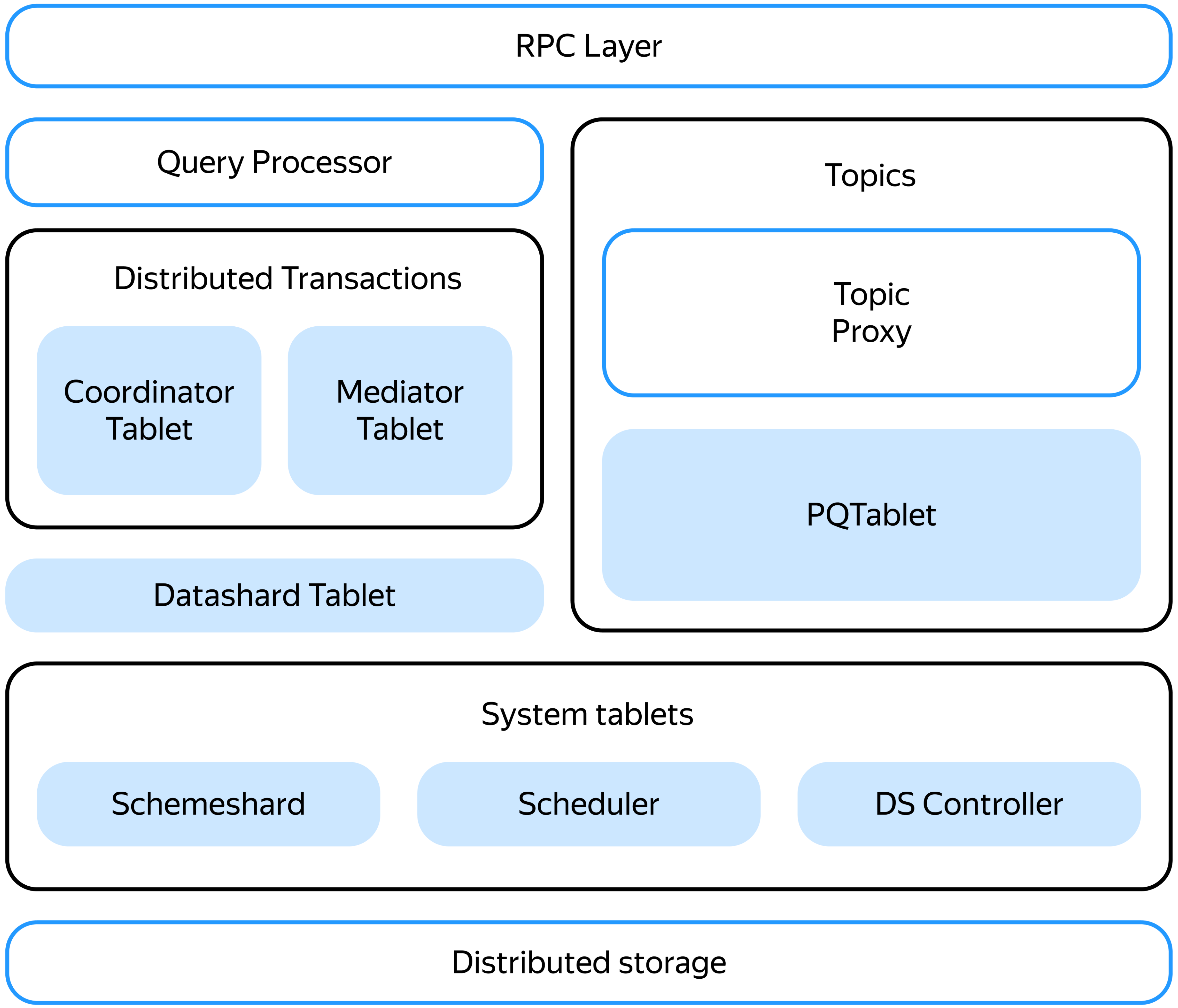
Different Layers for Computing and Storage

- Tablet is a Replicated State Machine which keeps its state in the distributed storage
- Runtimes for Tablets and queries are running on compute nodes
- The data is stored on storage nodes
- YDB moves Tablets between nodes for load balancing



YDB platform components

- Tablet is a Replicated State Machine
- Storage layer is separated from compute layer
- There are different types of Tablets (DataShard Tablet, PQTablet…)
- Actor system for communication



Horizontal scaling: table partitioning

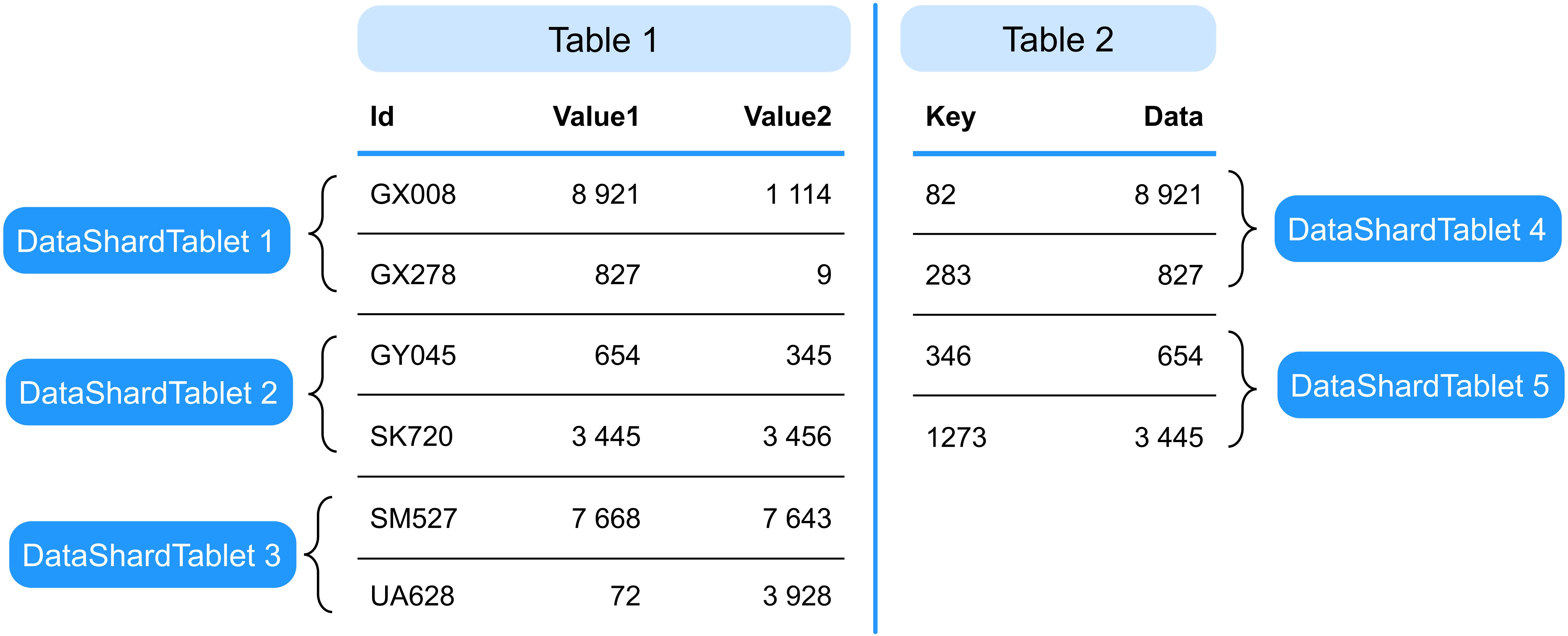


Table 1

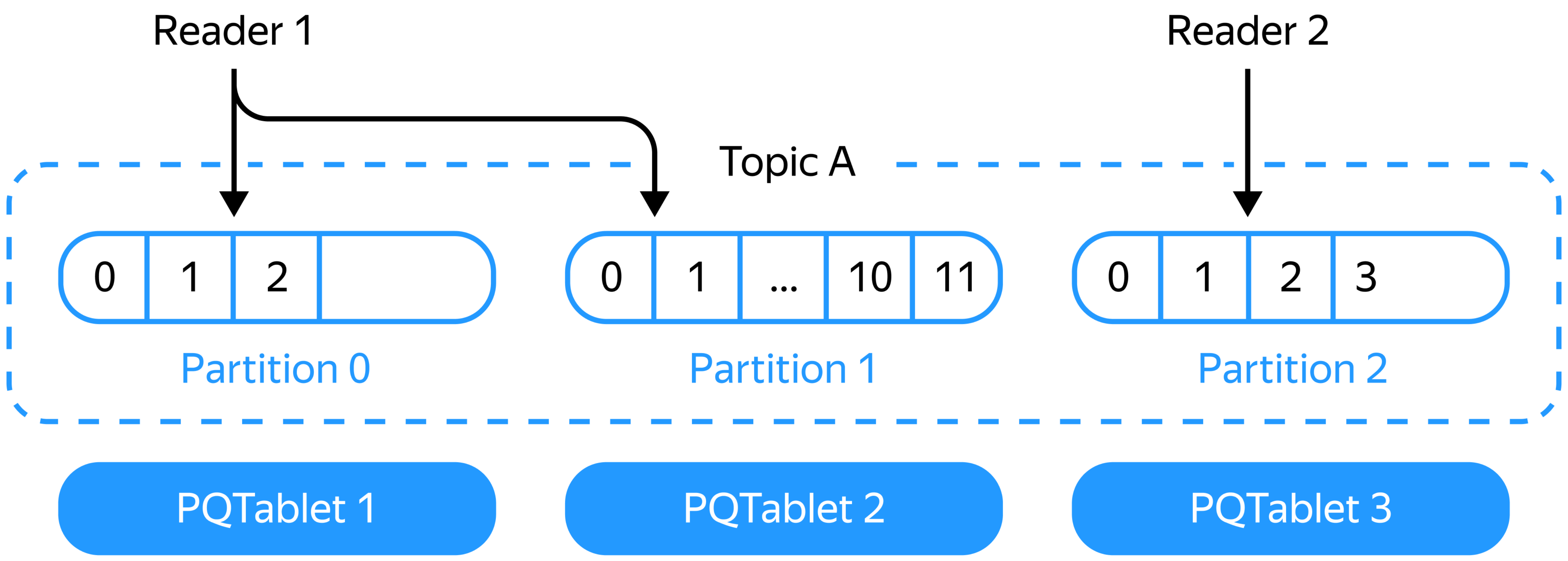
	Id	Value1	Value2
DataShardTablet 1	GX008	8 921	1 114
	GX278	827	9
DataShardTablet 2	GY045	654	345
	SK720	3 445	3 456
DataShardTablet 3	SM527	7 668	7 643
	UA628	72	3 928

Table 2

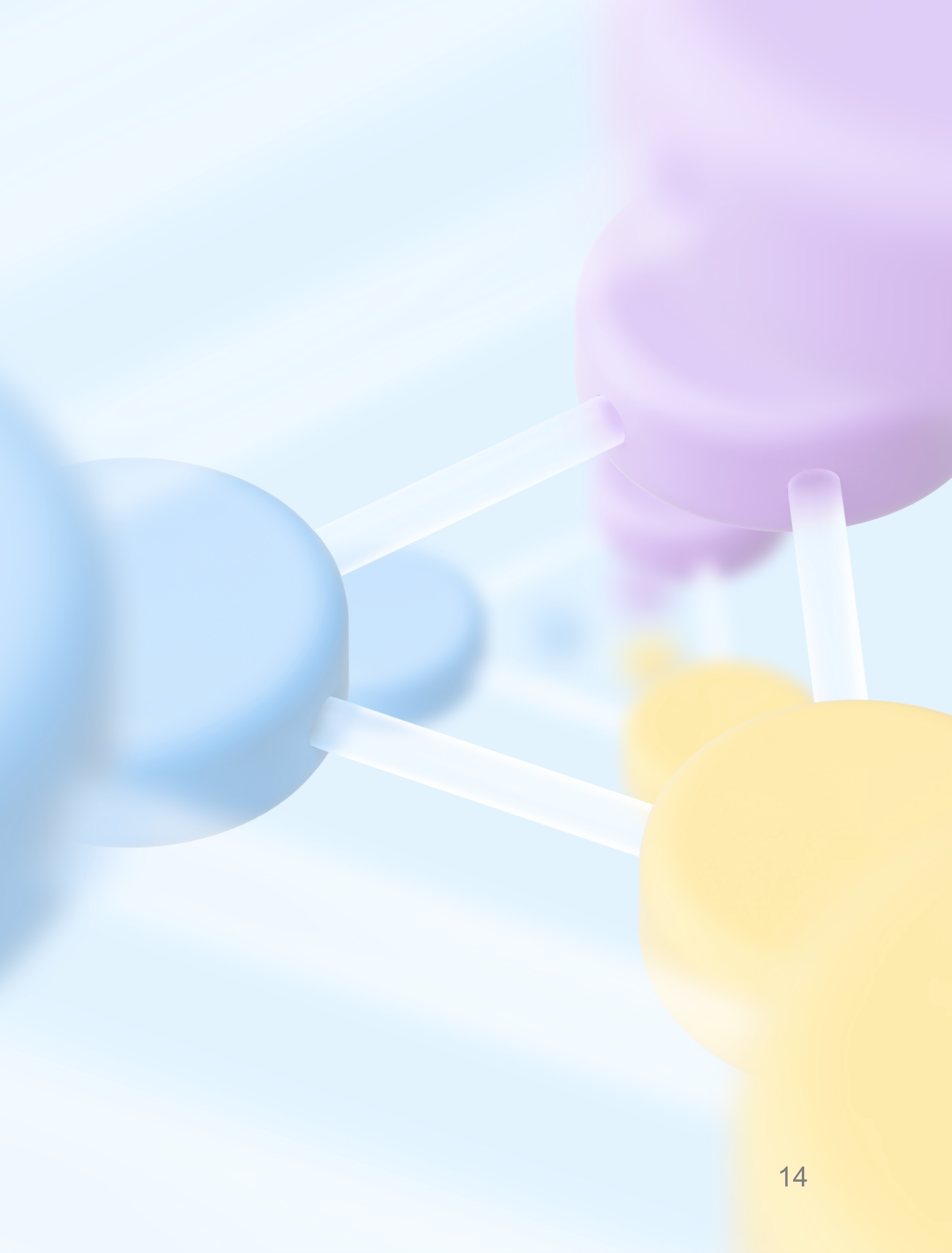
Key	Data	
82	8 921	DataShardTablet 4
283	827	
346	654	DataShardTablet 5
1273	3 445	

YDB topic structure

- User data is grouped into topics
- Topic is divided into partitions
- One partition is a log of messages
- Sequence number of the current message in partition is the offset (offset is a property of the pair partition-reader)
- Every partition is served by one PQTablet



YDB Platform: Transactions with Tables and Topics

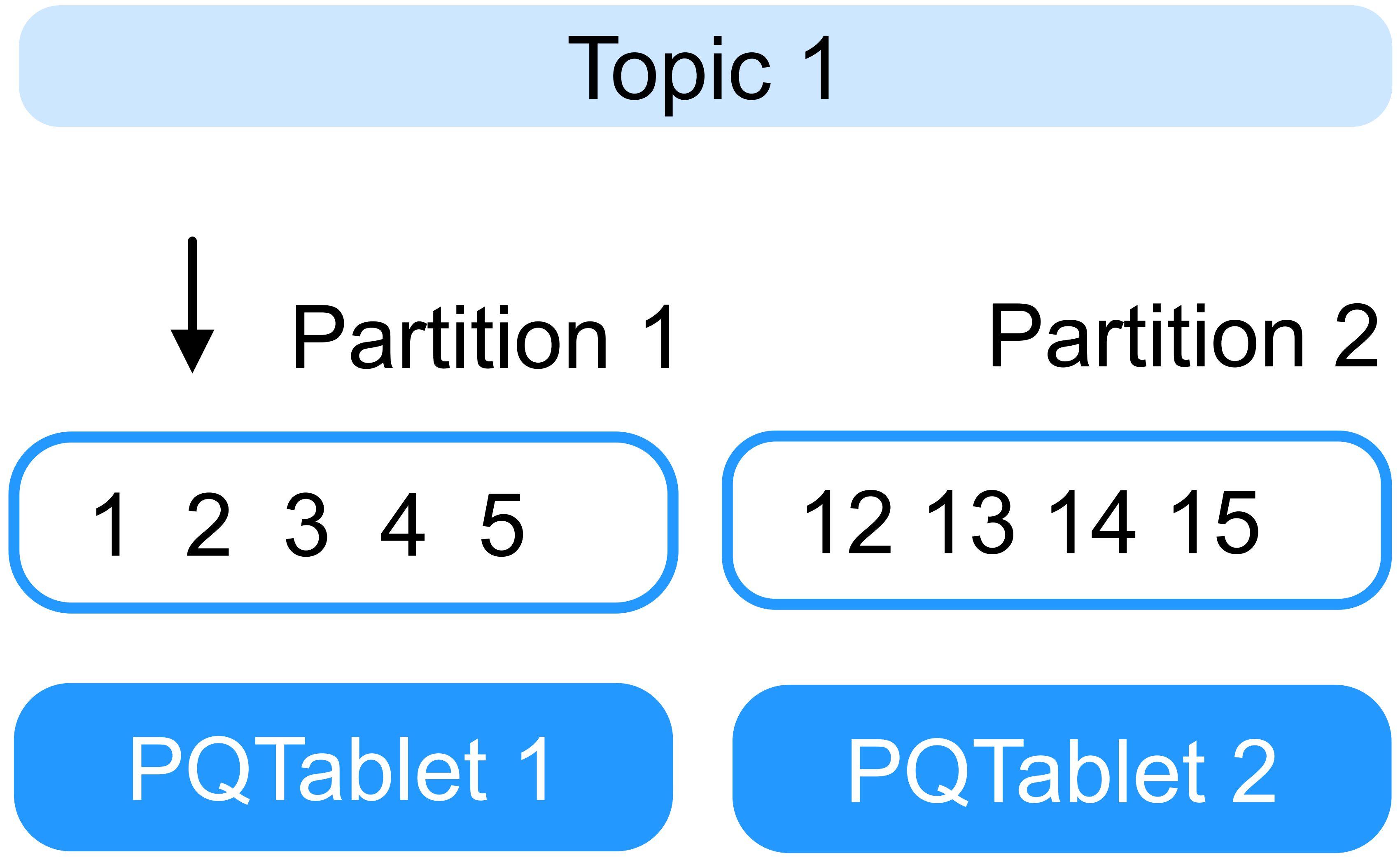


YDB Transactions

Key points:

- Serializable level of isolation by default
- YQL transactions from the User
- Inside YDB:
  - Transactions can be distributed (if applied to several data shards or topic partitions)
  - Distributed transactions are processed with Calvin protocol (plus additional coordinators)

Distributed transaction example



Topic 1

Partition 1

Partition 2

1 2 3 4 5

12 13 14 15

PQTablet 1

PQTablet 2

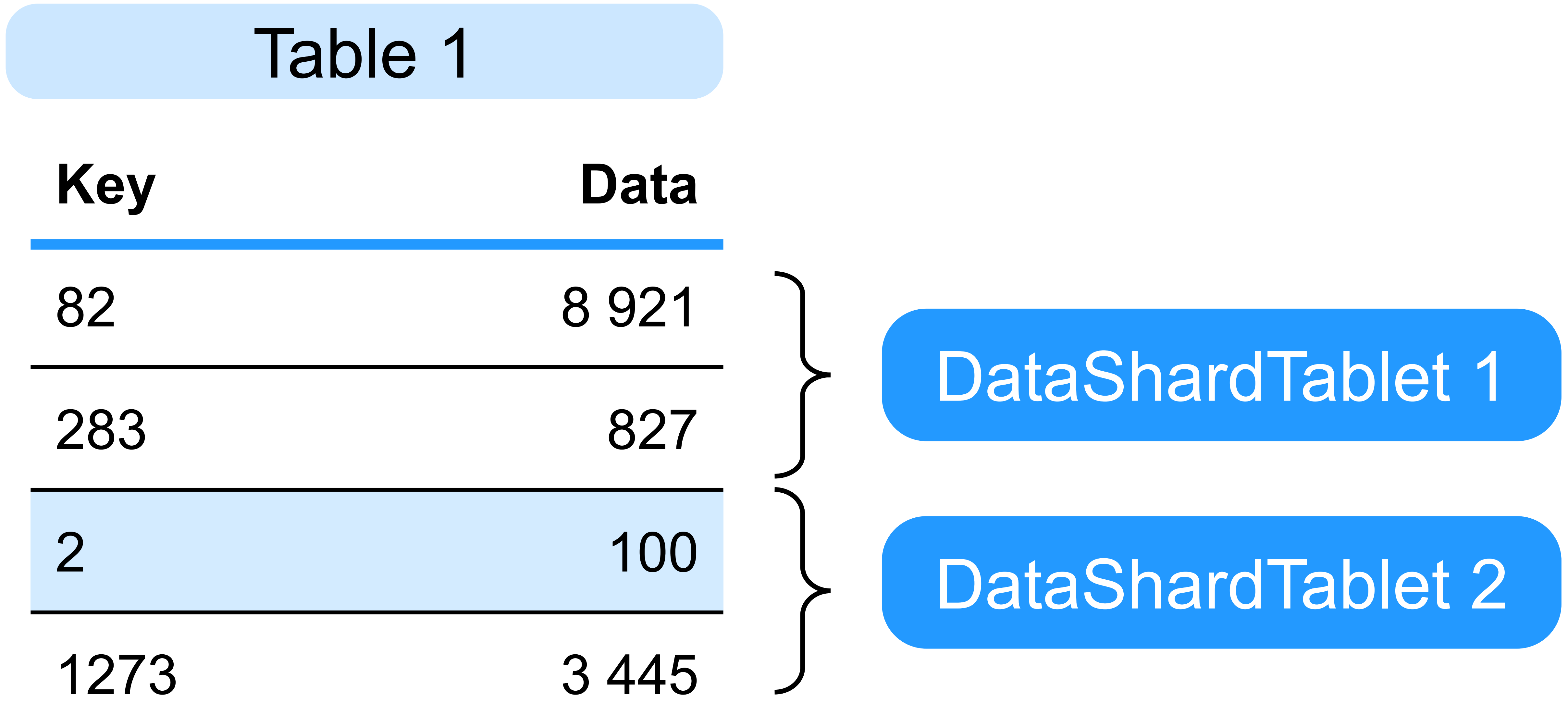


Table 1

Key	Data
82	8 921
283	827
2	100
1273	3 445

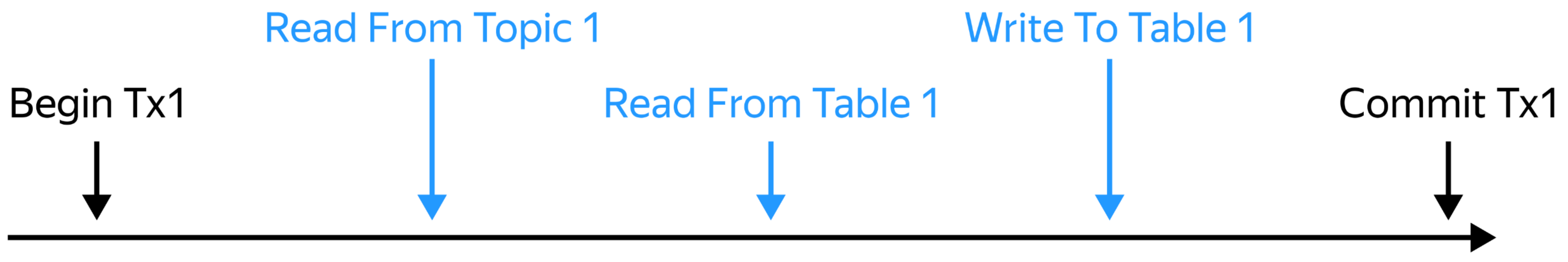
DataShardTablet 1

DataShardTablet 2

```
BEGIN TRANSACTION Tx1;

A = READ 1 MESSAGE FROM Topic1;
B = READ Data FROM Table1 WHERE Key = A;
WRITE INTO Table1: SET Data=B+1 WHERE Key = A;

COMMIT Tx1;
```



Begin Tx1

Read From Topic 1

Read From Table 1

Write To Table 1

Commit Tx1

How to execute distributed transactions

YDB uses Calvin protocol

- Calvin: Fast Distributed Transactions for Partitioned Database Systems by Daniel J. Abadi, Alexander Thomson
- Calvin allows to execute deterministic transactions without locks and conflicts
  - Deterministic transactions know sets of keys for reading/writing

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

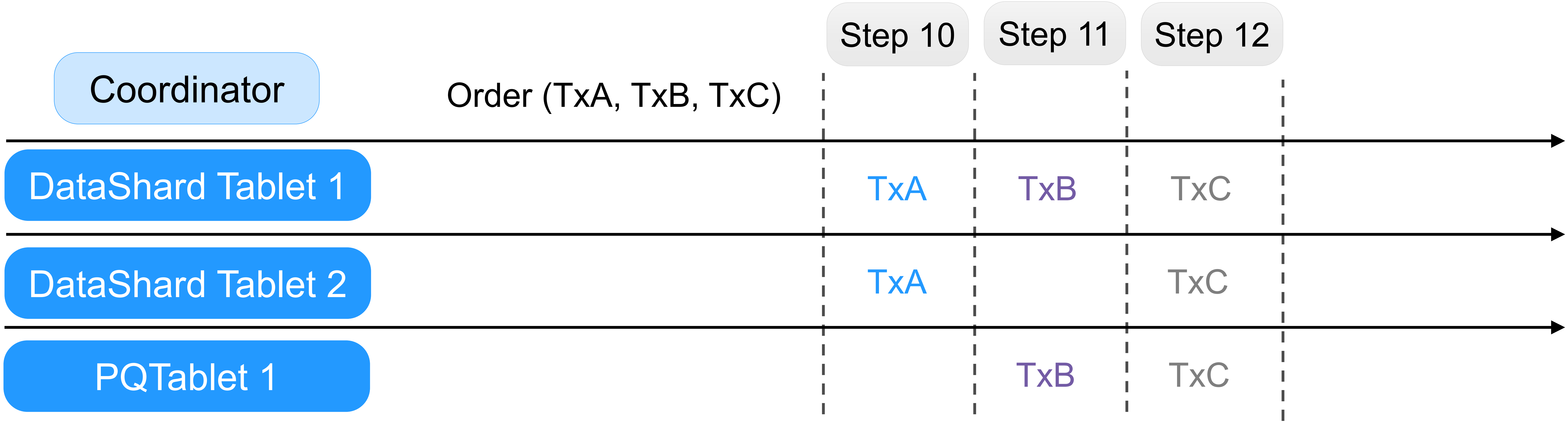
- Calvin can not execute any transaction which is written as SQL query, that's why executing transactions in YDB is bigger than Calvin protokol

How Calvin executes deterministic transactions

Suppose we have these transactions:

TxA (DS1, DS2), TxB (DS1, PQ1), TxC (DS1, DS2, PQ1)

Calvin:
If Coordinator arranges incoming transactions, there will be no conflict between transactions and we'll get serializable isolation



Multistep transactions in YDB

Example of non-deterministic transaction:

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

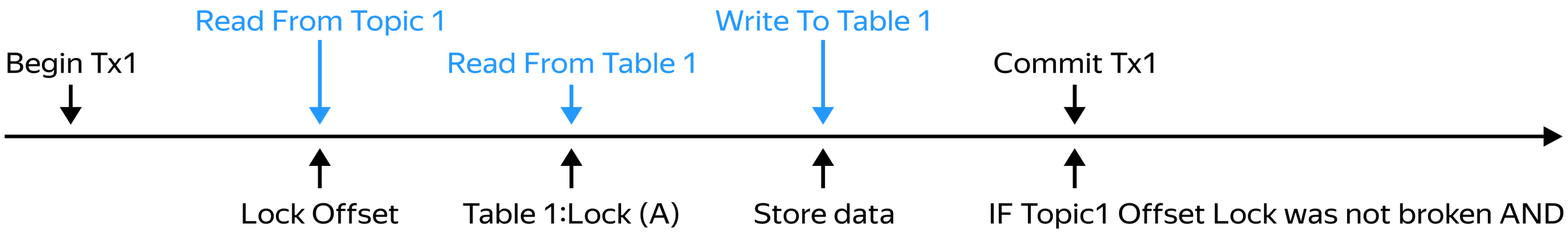
```
1. LOCK(A)
2. LOCK(value(A))
3. LOCK(B)
4. write(C) if LOCKs are not broken
```

We can split a non-deterministic transaction into the sequence of deterministic transactions.

Every step is a deterministic transaction. YDB makes LOCKs on every step. Locks are optimistic. Overall transaction is committed at the end if LOCKs were not broken.

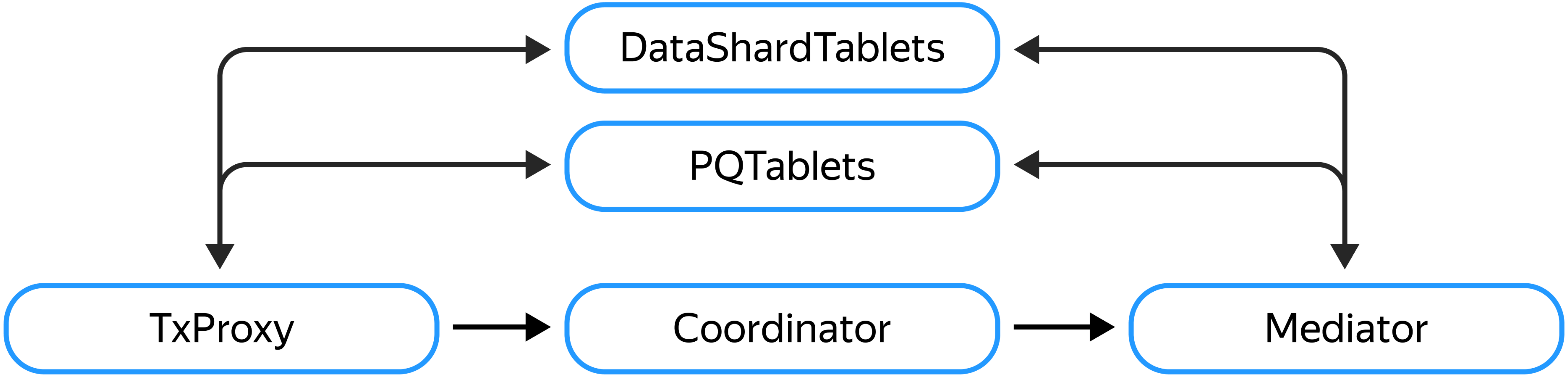
Distributed transaction example

```
BEGIN TRANSACTION Tx1;
A = READ 1 MESSAGE FROM Topic1;
B = READ Data FROM Table1 WHERE Key = A;
WRITE INTO Table1: SET Data=B+1 WHERE Key = A;
COMMIT Tx1;
```



```
IF Topic1 Offset Lock was not broken AND
  Table1 Lock(A) was not broken THEN
 Apply changes (Topic1: Set new offset,
 Table1: Update);
 Return Success;
ELSE
 Return Abort;
```

Transaction: components interaction



Assign txid

Define read-write sets

Define shards

Save "query parts" on shards

Assign stepid

Save step plan

Group transaction by shard

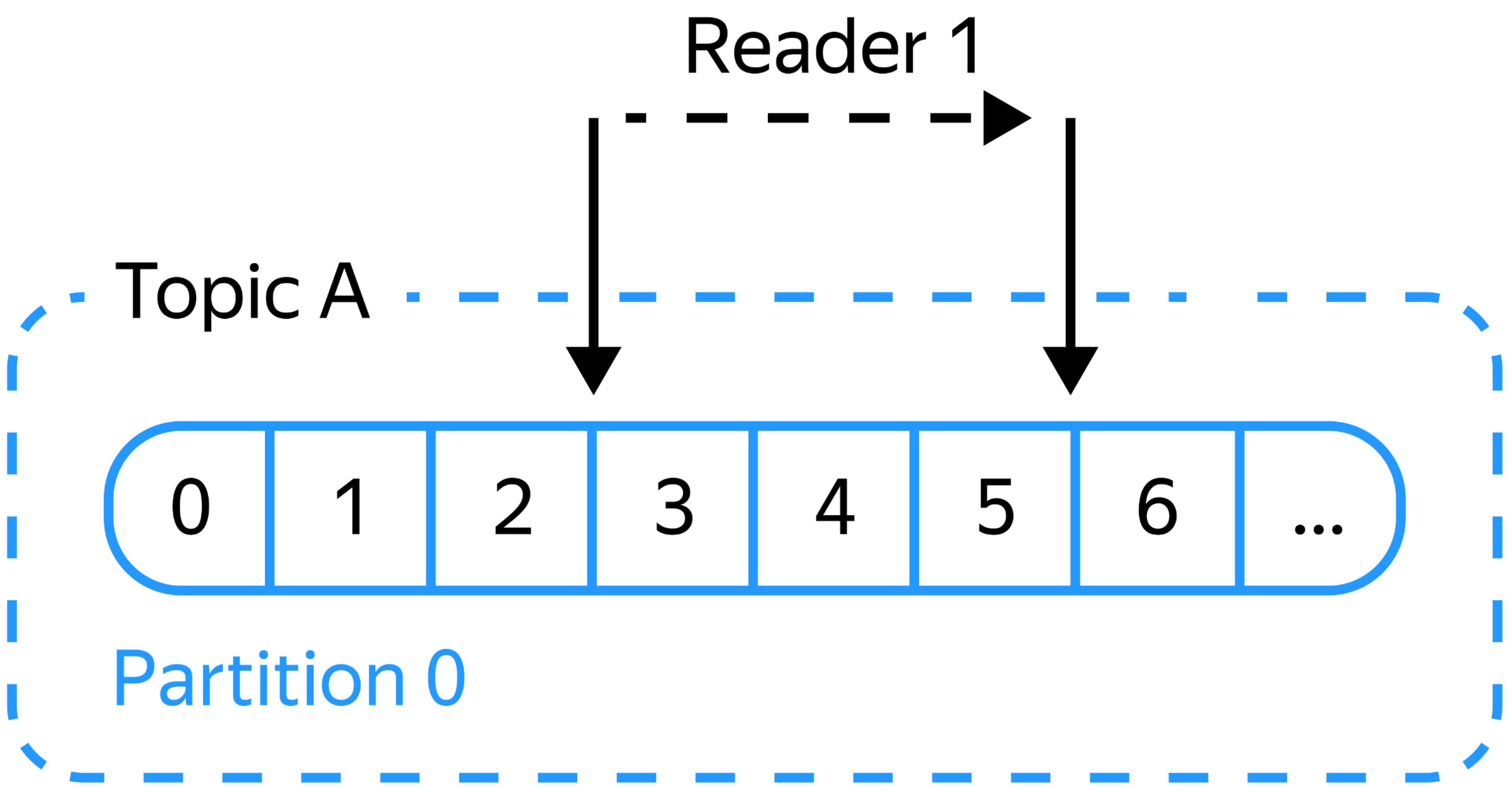
Send step plans to shard

Reading from topic within transaction

Getting data + Moving offset on commit

- Action: Moving offset
- Predicate: Every offset is moved only in one transaction

  So if 2 transactions are reading the same data (1 specific partition), than one of these transactions would be committed, and another would be aborted
- Offsets should be moved in strict order (no skips)



Examples
Topic: Reading

Offset = 3

Begin Tx1

Begin Tx2

…

Read messages 3 – 5 in Tx1

…

Read messages 3 – 10 in Tx2

…

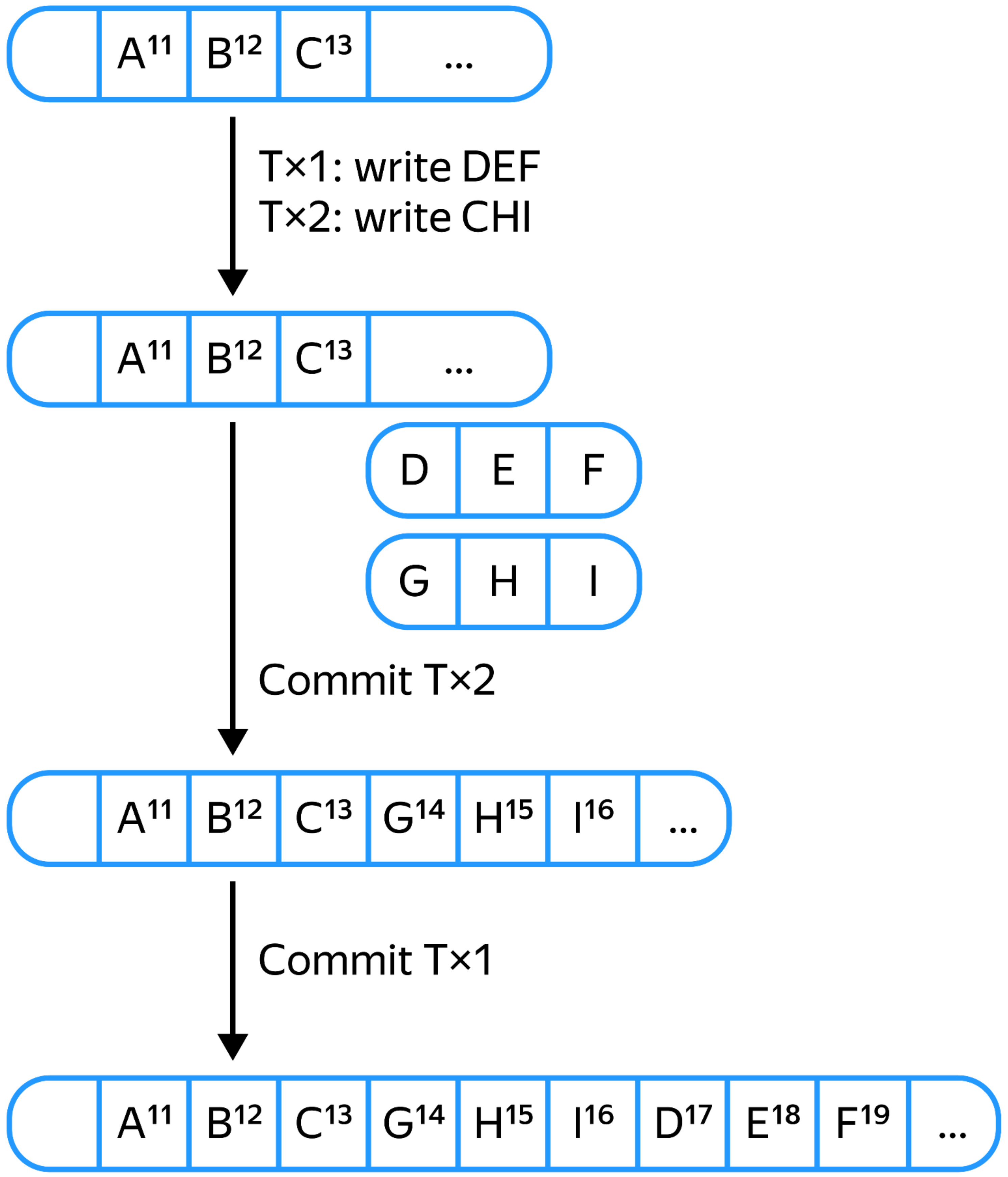
Commit Tx2 Success, Offset = 11

Commit Tx1 Abort, Offset was changed in Tx2

Writing into a topic within transaction

- Action: Writing data
- Predicate: Written data are available for reading only after transaction commit
- So if 2 transactions are committed, than their data are available for reading in order of transactions' commit

Examples
Topic: Writing


A^{11} B^{12} C^{13} ...
T×1: write DEF
T×2: write CHI
A^{11} B^{12} C^{13} ...
D E F
G H I
Commit T×2
A^{11} B^{12} C^{13} G^{14} H^{15} I^{16} ...
Commit T×1
A^{11} B^{12} C^{13} G^{14} H^{15} I^{16} D^{17} E^{18} F^{19} ...

//state of partition before:
messages A, B, C

Begin Tx1

Begin Tx2

…

Write messages D, E, F in Tx1
Write messages G, H, I in Tx2

…

Commit Tx2 Success, partition ABCGHI

…

Commit Tx1 Success, partition ABCGHIDEF

Performance

	Test A	Test B
MessageSize, bytes	10 240	1 000 000
Write speed for 1 writer, messages/s	~102	1
Write time 50 percentile (without transactions), ms	7	16
Write time 50 percentile (with transactions), ms	8	25

Tests configuration

- 100 partitions
- 100 writers
- 100 Mb/s write speed overall
- Commits every second
- 8 servers: 2 CPU Xeon (56 cores), 256 Gb RAM, 4 NVMe 3.2Tb, Net 10Gb/s

Conclusions

★ Now YDB can operate topics and tables within a single transaction

★ It simplifies user code

★ Minimal impact on latency in case of writing small messages

★ CPU usage and system throughput are the same

★ We add ACID guarantees to topic-table operations

Questions?

YDB Community Chat:
t.me/ydb_en

YDB Documentation
ydb.tech/docs/en

YDB Repository
github.com

Elena Kalinina
Technical Project Manager, YDB
t.me/AfinaYndx