
yugabyteDB
xCluster Replication
Discover the Power and
Flexibility of Async Replication
Build
Meet
Learn

Wei Wang

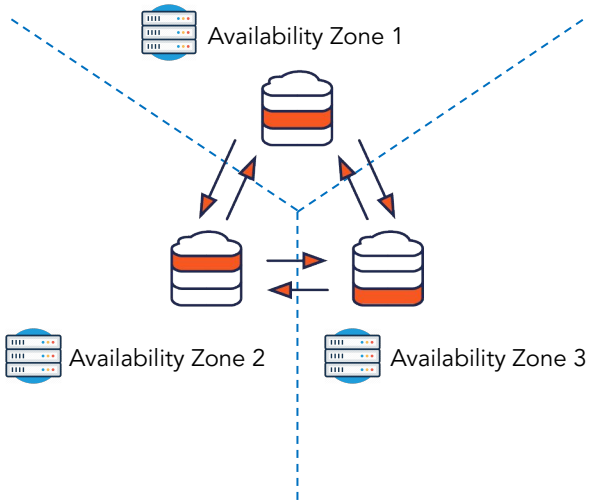
Yugabyte

Michael Spiering

Yugabyte

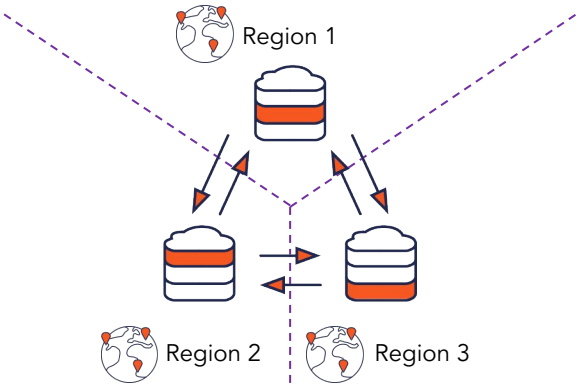
Synchronous Replication: Resilient and strongly consistent

1. Single Region, Multi-Zone



Consistent Across Zones
No WAN Latency But No
Region-Level Failover/Repair

2. Single Cloud, Multi-Region



Consistent Across Regions
with Auto Region-Level
Failover/Repair

Read-only, eventually consistent reads can be achieved through:

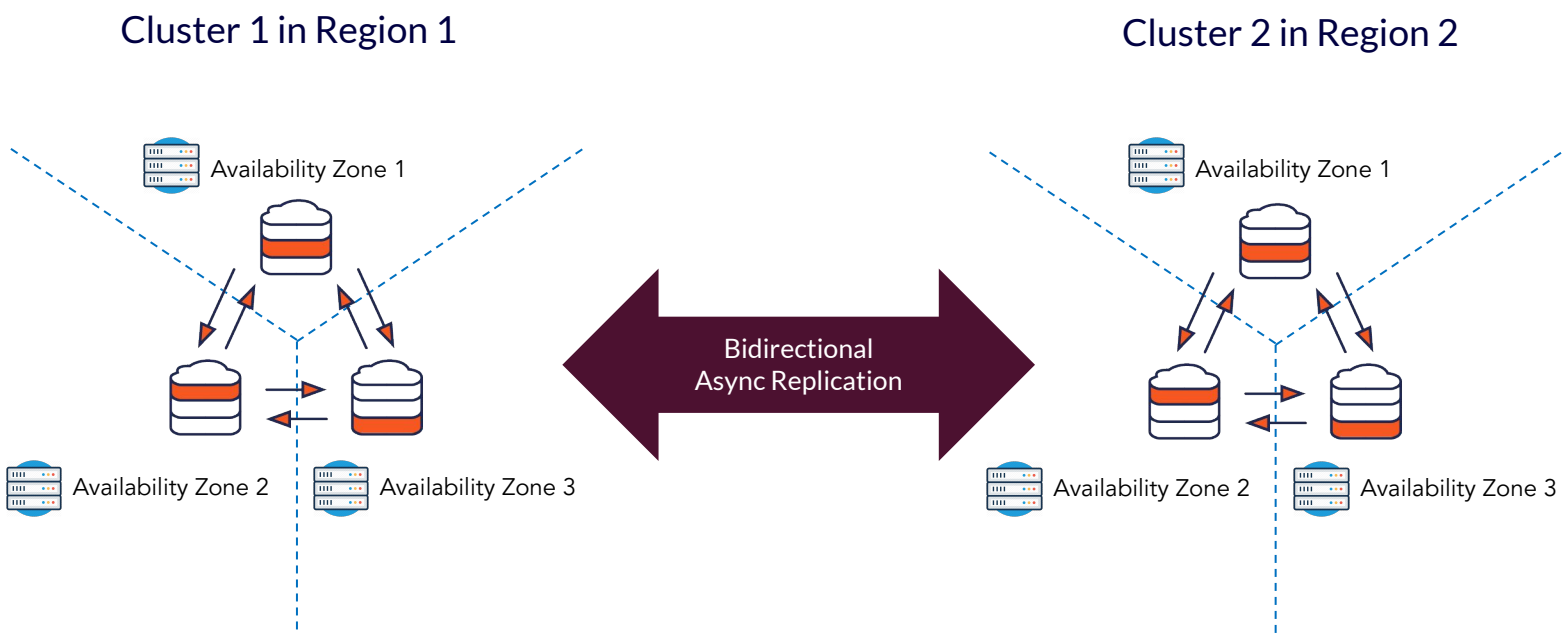
- Follower reads
- Read Replicas

Writes, however, are always consistent (CP database)

- Speed of light is an issue for multi-region deployments

All scenarios require a odd number of failure domains to ensure a quorum can be established

Multi-Cluster Deployments with xCluster Replication



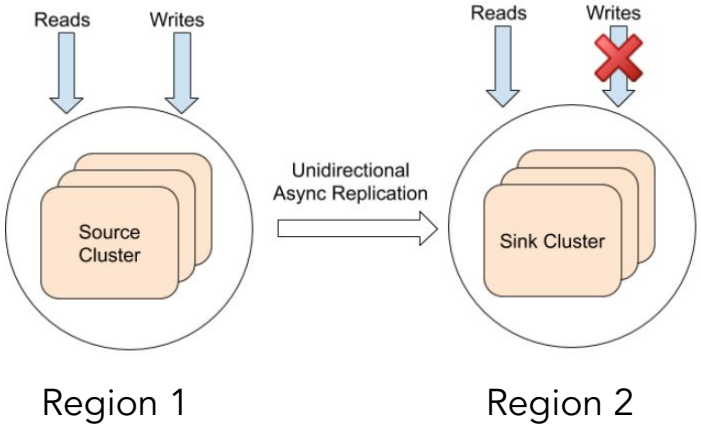
Consistent Across Zones
No Cross-Region Latency for Both Writes & Reads
App Connects to Cluster in Region 2 on Failure

Consistent Across Zones
No Cross-Region Latency for Both Writes & Reads
App Connects to Cluster in Region 1 on Failure

xCluster

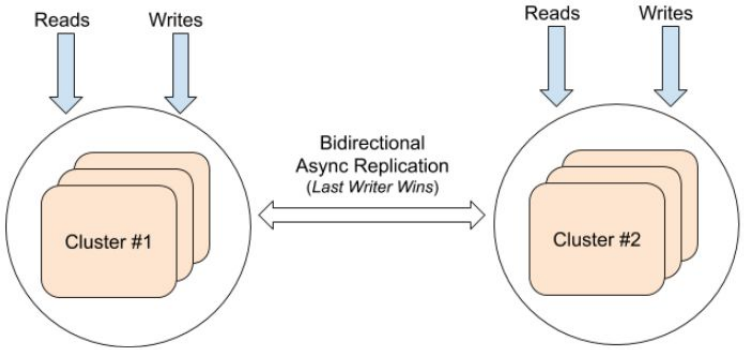
Use Cases

Supported Topologies



Active / Passive setup

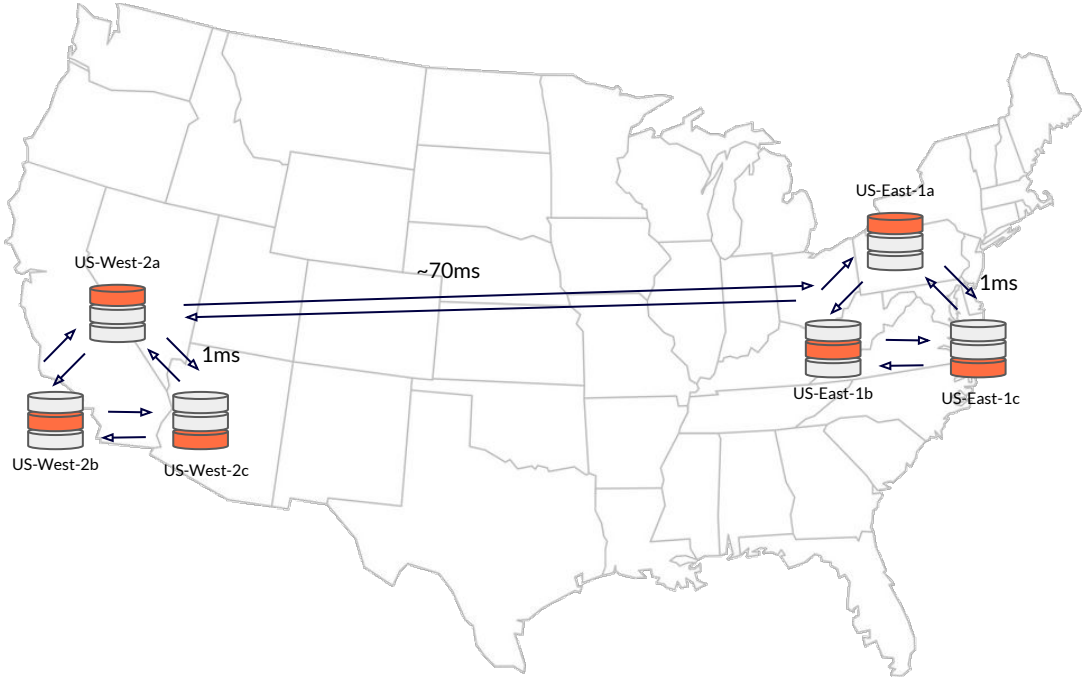
- Ideal for DR purposes
- Low latency reads on both source and sink clusters
- Eventually consistent and timeline consistent
- Live migrate clusters, eg AWS -> GCP



Active / Active setup

- Solves 2 DC problem
- Low Latency reads and write on both clusters
- Last data written wins
- All the trigger/primary key concerns from Gotchas

Major Financial Services Company



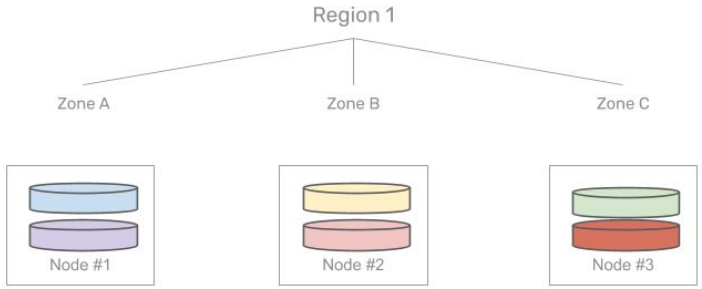
- 2 Data Centers on opposite coasts
- Writes have locality of data (users are always tied to a particular DC)
- Both reads and writes must be fast

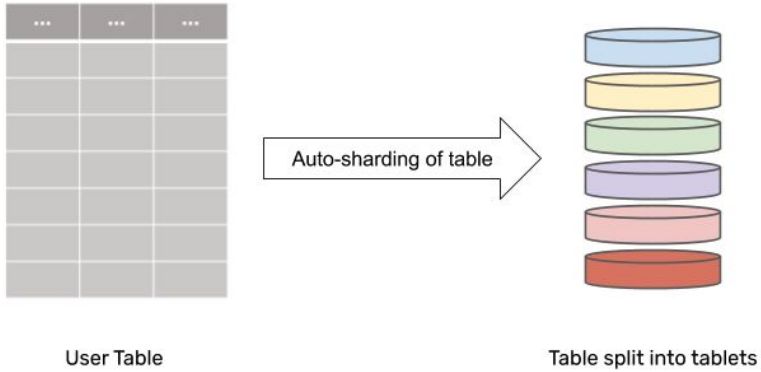
- Either DC should be able to handle all traffic for DR purposes.

xCluster

How Does It Work?

First, some terminology: TABLET

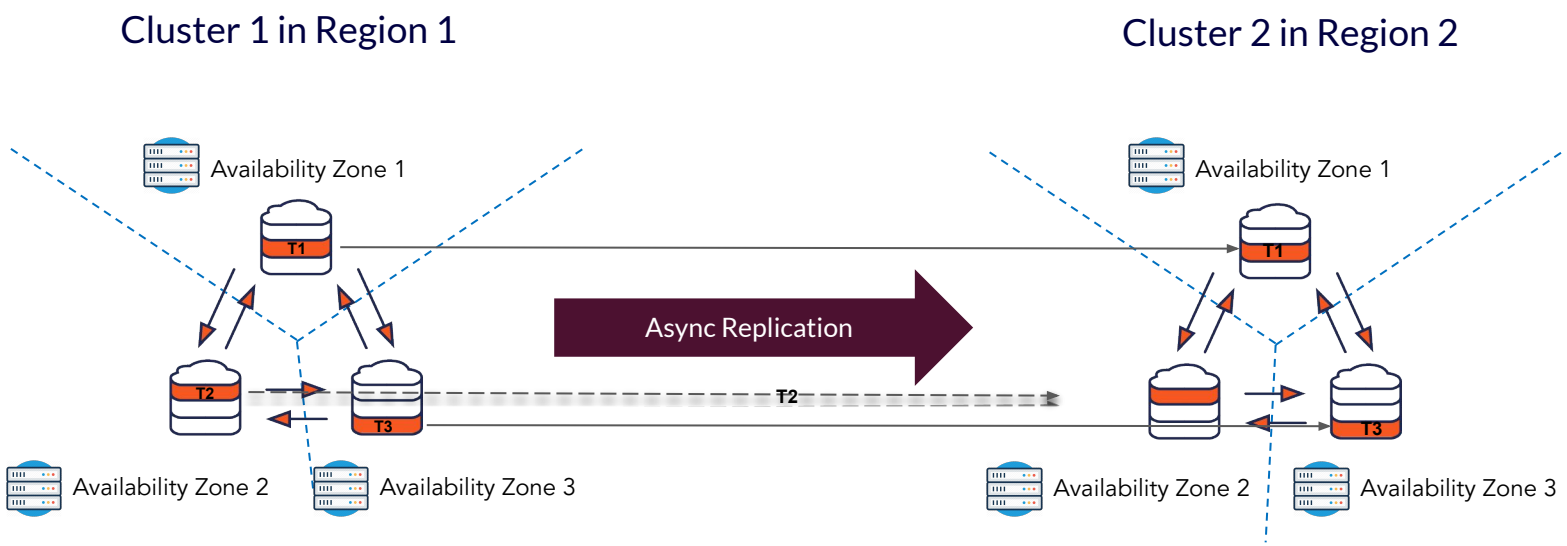




- Assume 3-nodes across zones
- 3 copies of the data => 1 leader, 2 followers

- User tables sharded into tablets
- Tablet = group of rows
- Sharding is transparent to user

xCluster Async Replication

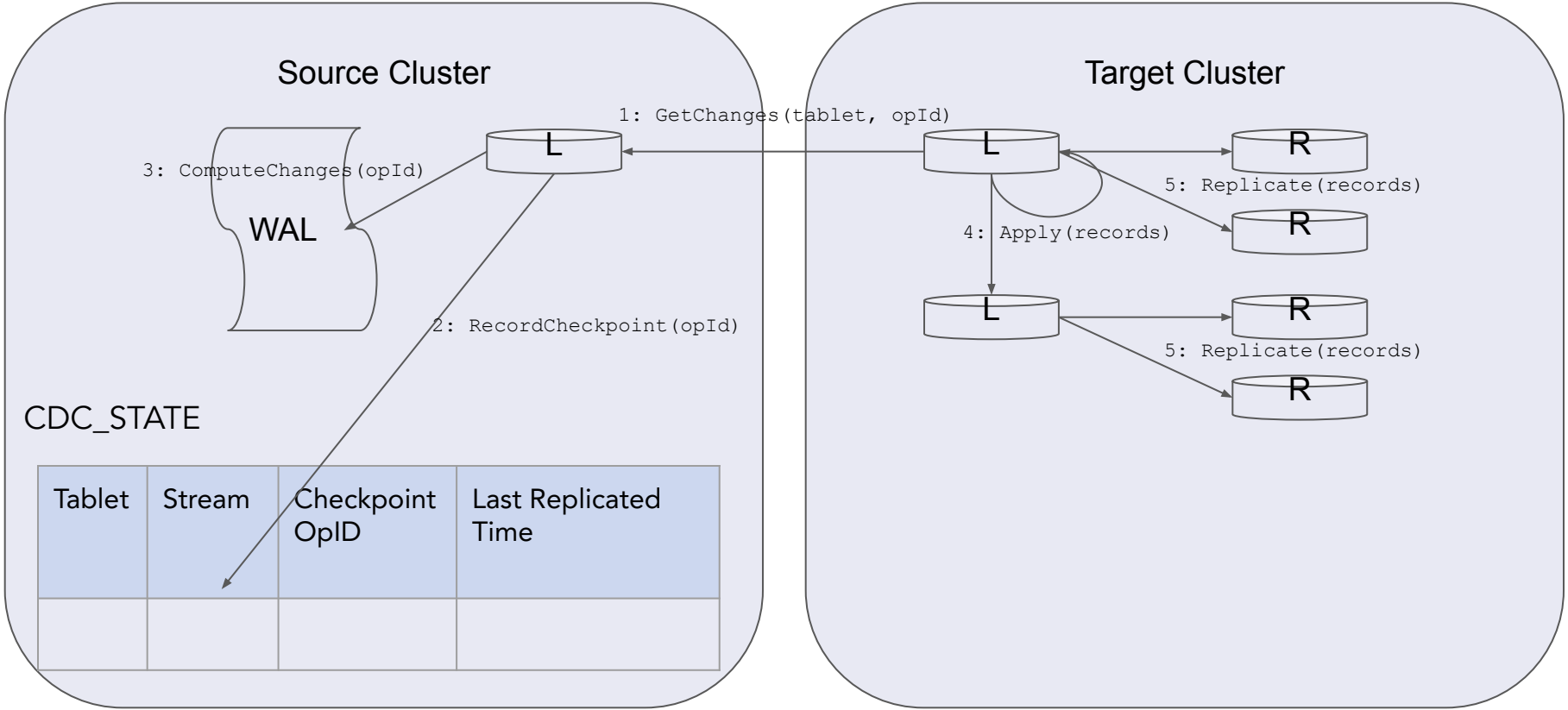


- Topologies of source and sink clusters can be different
- Each tablet is replicated independently
- Writes are batched together for efficient transmission to the sink cluster.
- Destination pulls records from the WAL file on the source cluster and puts them straight into the DocDB storage layer.
    - Hence: **TRIGGERS** are not recommended – they will not be fired on the destination

How XCluster Async Replication works - Setup

Schema Creation	Setup replication on target cluster	Target cluster starts replication process
Run DDL on both clusters	Use the YBA interface that manages the target cluster to setup replication.	1. Validate schema 2. Get source tablet information 3. Setup a poller on target for each source tablet 4. Source computes records not yet sent to target and sends them to target 5. Target poller sends data to the right tablets 6. Target tablets replicate the data using RAFT

How XCluster Async Replication works - Steady state operation



How XCluster Async Replication works - Setup with existing data in source

Bootstrap source and target

1. Backup data on source and record the checkpoint
2. Restore the backup on target

Setup replication on target cluster

Use the YBA interface that manages the target cluster to setup replication.

Target cluster starts replication process

1. Validate schema
2. Get source tablet information
3. Setup a poller on target for each source tablet
4. Source computes records not yet sent to target and sends them to target
5. Target poller sends data to the right tablets
6. Target tablets replicate the data using RAFT

xCluster

What are the problems today?

Gotchas!

- **Active <> Active not production ready for some use cases due to challenges of supporting transactional atomicity(Last Writer Wins)**
  - 2DC, Active <> Passive is well supported topology
- **Avoid setting up replication using CLI if both universes running on the same platform**
  - Giving maturity of platform functionalities such as automatic bootstrapping (2.15.3.0-b64 or later)
  - Inability to monitor via the platform if configuring it via CLI
- **It is highly recommended to use the same version of YBDB to set up xCluster replication**
  - For a software upgrade, Pause replication, the target universe should be upgraded first.
- **Avoid Sequences / unique indexes / triggers**
- **Current day 2 operation challenges such as**
  - DDL changes are not automatically replicated; No support of drop table, truncate table
  - We do not have a way to alert the user if replication is fully broken: plan to expose a API
  - Potential issues with encryption-at-rest, TLS, etc
  - Bootstrapping - backup and restore based. KMS limitations, full database restore

xCluster

2.17 enhancements

xCluster Transactionality

One transaction might update records in 2 different tablets

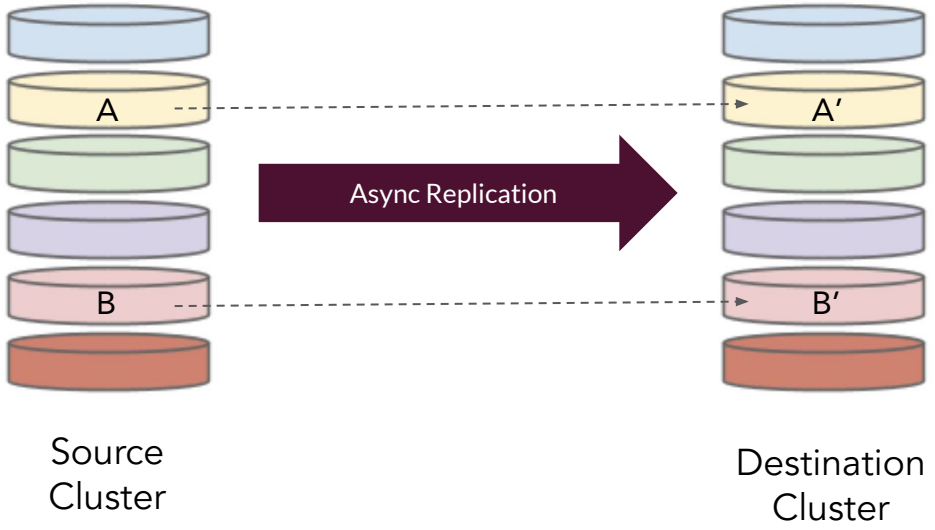
```
begin transaction
    update A
    update B
commit transaction
```

Source sees either:

- Changes to A and B
- Changes to neither A nor B

Destination can see:

- Changes to A and B
- Changes to neither A nor B
- Changes to A but not to B
- Changes to B but not to A



xCluster Global Ordering

Non-transactional updates imply a certain ordering:

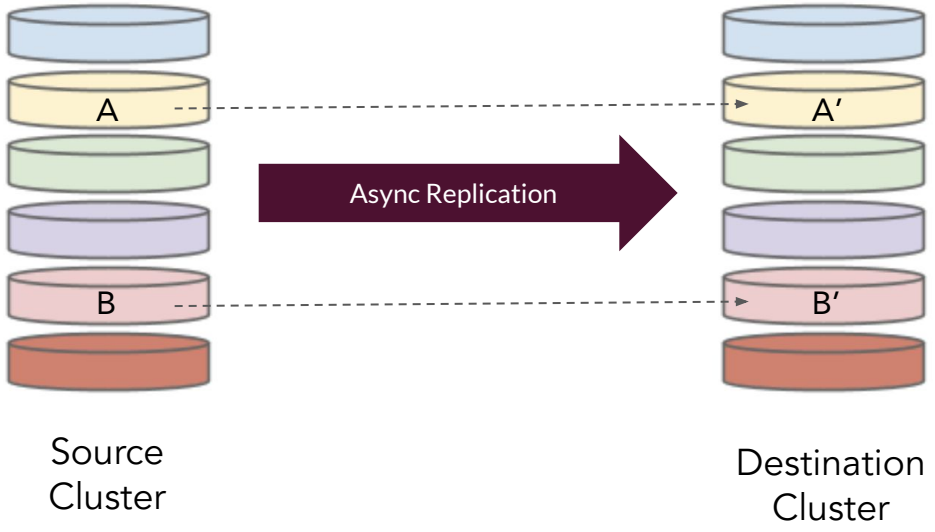
```
update A
update B
```

Source can see:

- Changes to neither A nor B
- Changes to A but not B
- Changes to A and B

Destination can see:

- Changes to A and B
- Changes to neither A nor B
- Changes to A but not to B
- Changes to B but not to A



Other Improvements

- Focus on failing to secondary data center in active/passive setup
  - Planned failover: RPO should be zero
  - Unplanned failover: RPO will be low but non-zero
  - New yb-admin commands for Transactional Consistency and DR Workflows
    1. Change_xcluster_role (ACTIVE vs STANDBY)
    2. Wait_for_replication_drain
    3. Get_xcluster_safe_time
       - get_xcluster_estimated_data_loss

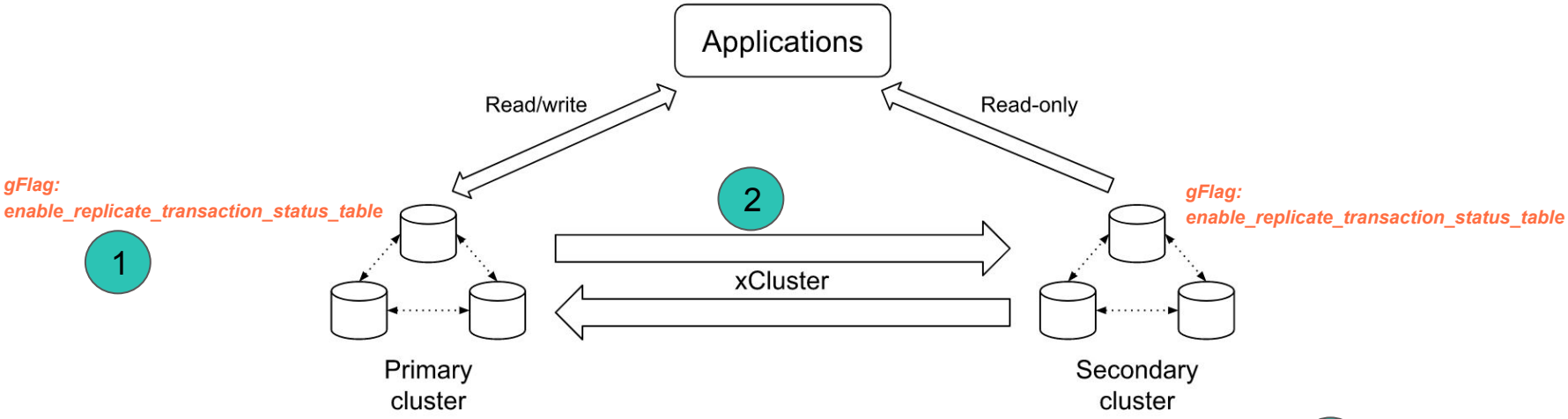
Future:

- Automatic DDL Focus:
  - All schema modifications to existing objects automatically get applied on both sides
  - All new objects get automatically enlisted into replication

xCluster

Demo

Setting up and Operating xCluster Demo: DR Use Case



XCluster Role - ACTIVE

XCluster Role - STANDBY 3

XCluster SafeTime - Consistent time at which Reads are served on a standby universe

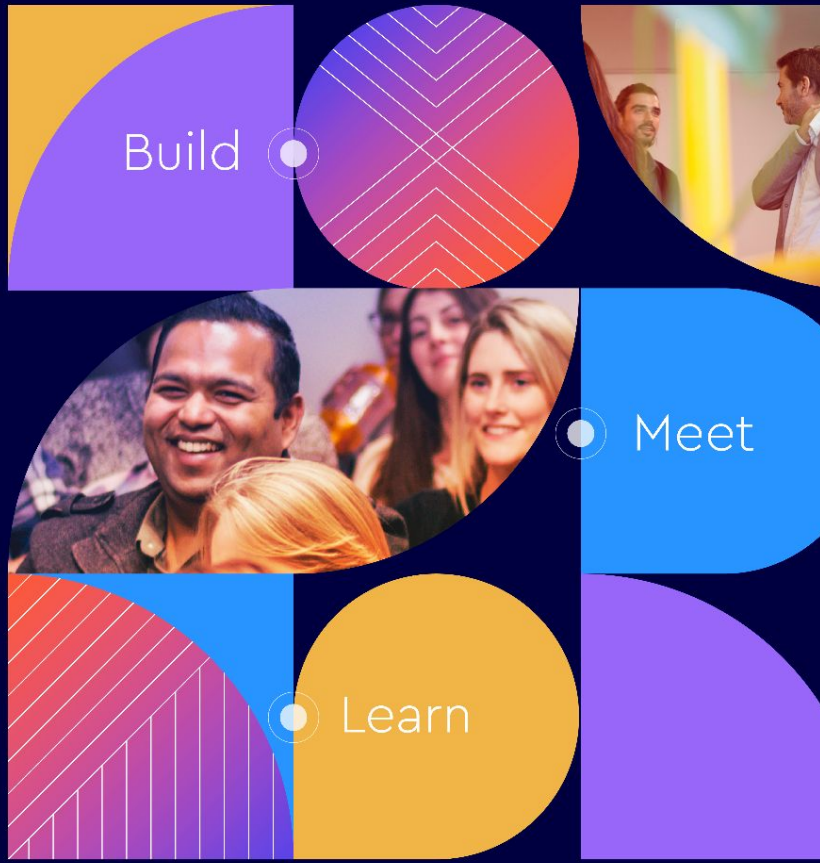
Consistency level - database (Cross-tablet consistency), tablet (similar to today)

xCluster Async Replication - Process Flow

- Call yb-admin.setup_universe_replication for n tables from the target to set up the replication stream
  - Tablet mapping is created(target tablet -> source tablet)
  - Leader of target tablet pulls from the source tablet using CDC_Poller, GetChanges(tablet, opid) polling API (called per tablet)
  - Source side computes all changes since last pull using (OpID) and returns the target
  - Target tablet server applies changes to local rocksdb
  - # of tablets between source and target side can be different
  - Each source tablet uses its own WAL to server the polling request from the target
  - WAL is retained up to the OpId that is needed with a max of 24 hours by default (configurable)
  - The source side maintains a system table CDC_STATE to store metadata about streams
  - Other yb-admin commands: `list_cdc_streams, get_universe_config,delete_cdc_stream`

Tablet	Stream	Checkpoint OpID	LastReplicatedTime

CDC_STATE



Thank You

Join us on Slack:
yugabyte.com/slack

Star us on Github:
github.com/yugabyte/yugabyte-db

Other Improvements

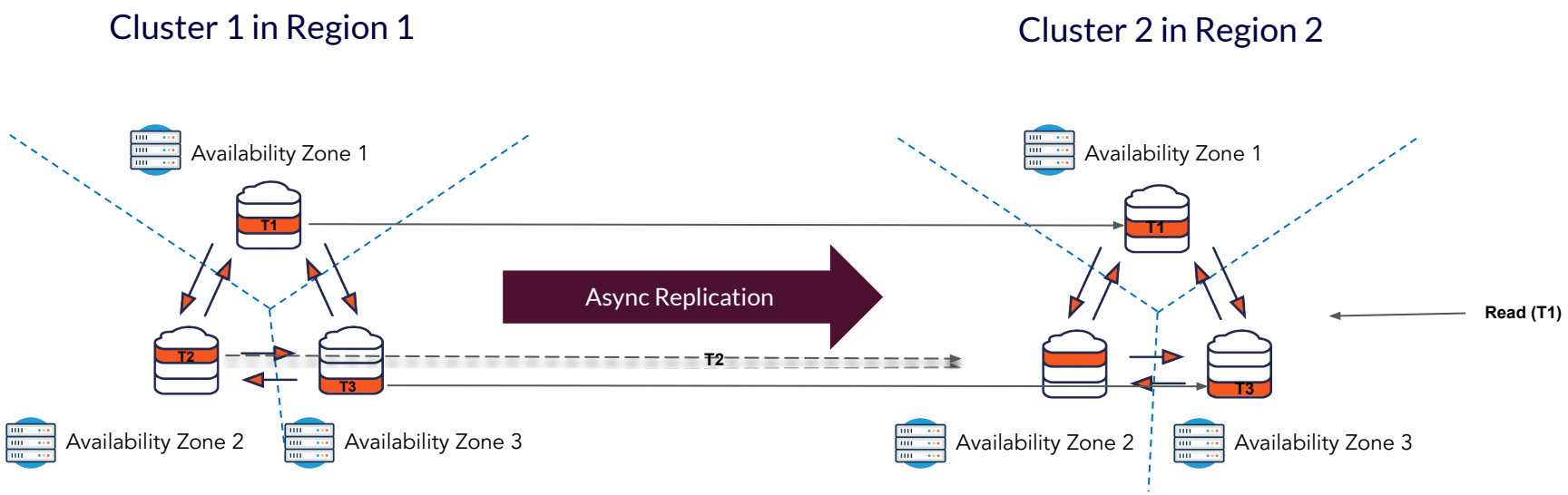
- DDL improvements
  - Support for safely performing ALTER table operations on both source/target without any loss of data or replication errors [Github 11017]
  - Integration of xCluster with Index Backfill [Github 7613]

- “Wait for Replication Drain” API
  - Allows safe cutover from source to target with no data loss [Github 10978]

- Expose APIs to better monitor replication health

- Handling of replication between tables with the same name in different schemas

xCluster Async Replication Transactional Atomicity



- T1, T2, T3 happen in order on Cluster 1.
- T1, T3 reach cluster 2.
- A read on Cluster 2, reads data as of T1 even though T3 is present already in Cluster 2 as T2 has not arrived