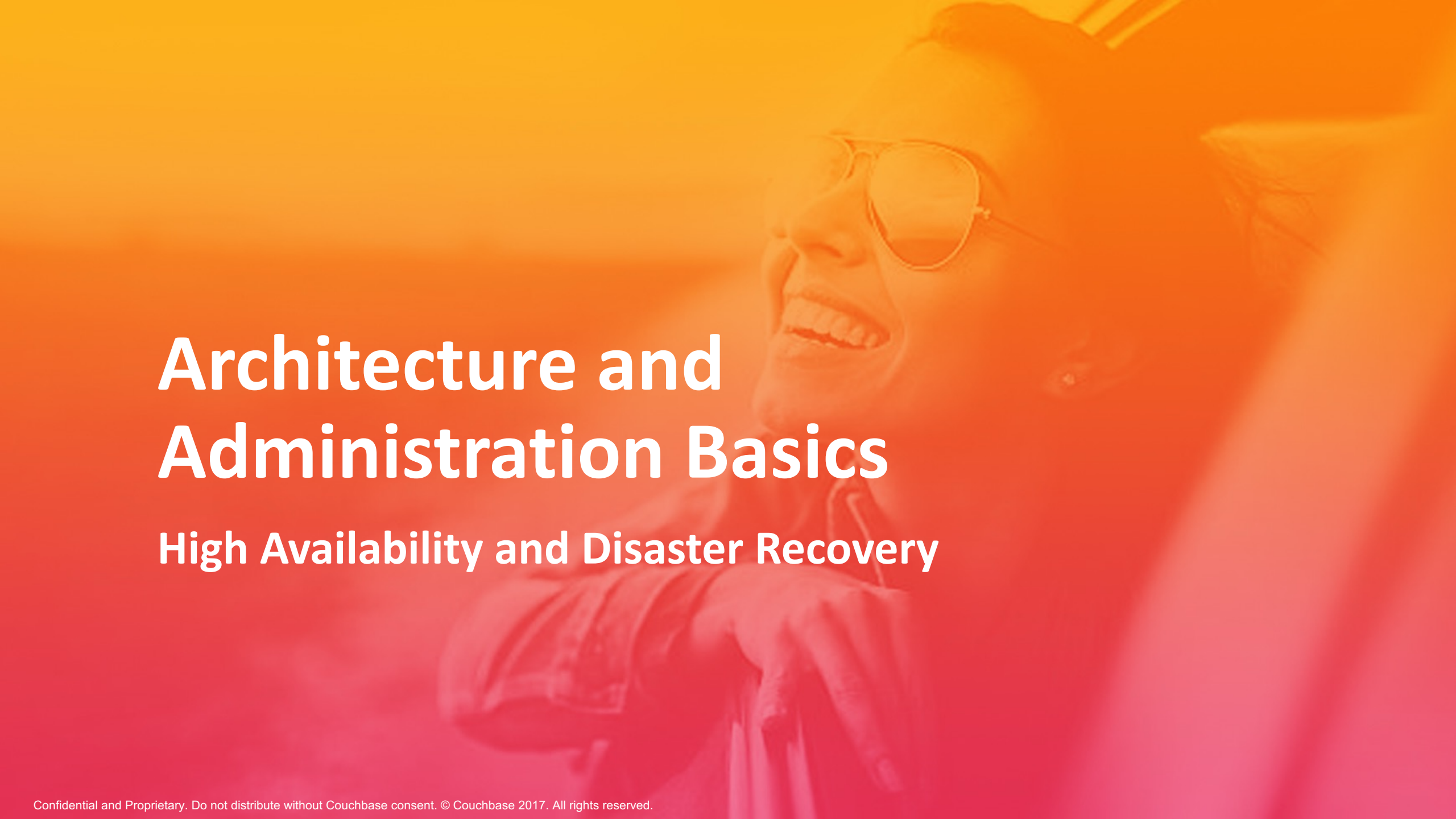


Architecture and Administration Basics

High Availability and Disaster Recovery

1 | Definition of HA and DR

High Availability & Disaster Recovery

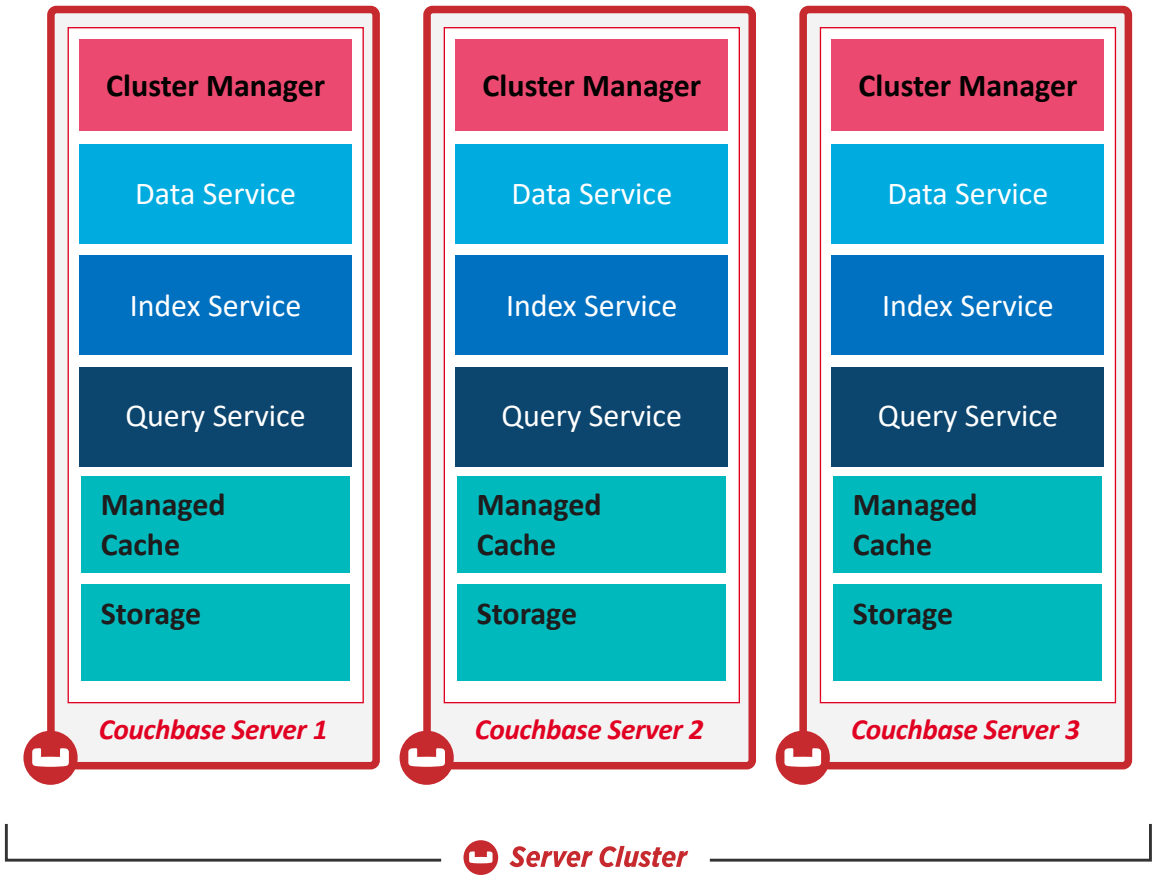
- **High Availability**
  - “Zero Downtime” administration & maintenance
  - Fault tolerance
  - Data & service redundancy
  - Automatic failover
- **Disaster Recovery**
  - A plan for Business Continuity for “mission-critical” applications
  - Multi-state redundancy and failover
  - Data recovery from backup for the worst scenarios

- Disaster Recovery requires a layered approach and overall plan which includes but is not limited to backup, recovery and testing.
- Your HA and DR strategies are dictated by your RTO (Recovery Time Objectives) and your RPO (Recovery Point Objectives)

2

Couchbase: Architectural High Availability

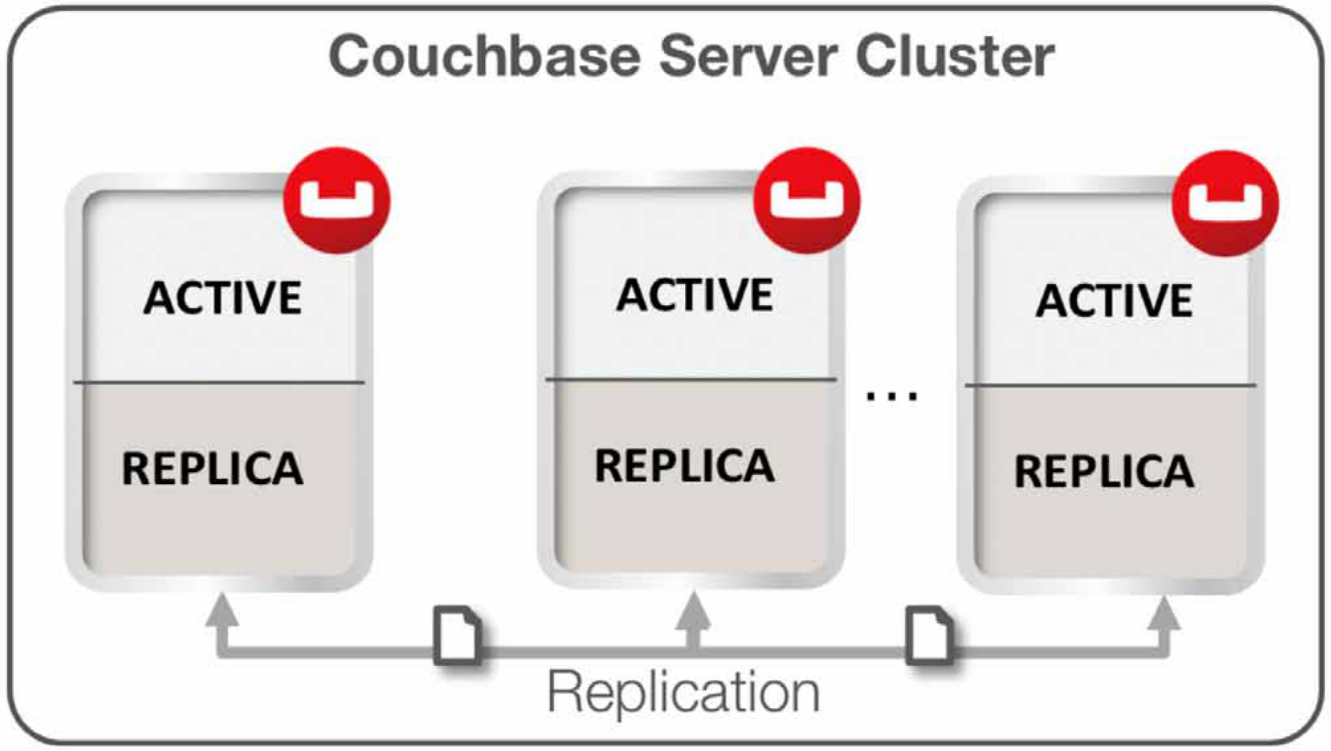
Couchbase Server – Single Node Architecture



- Single node type is the foundation for high availability architecture
- Built-in Redundancy and Failure Detection
- No Single Point of Failure (SPOF)
- Easy scalability

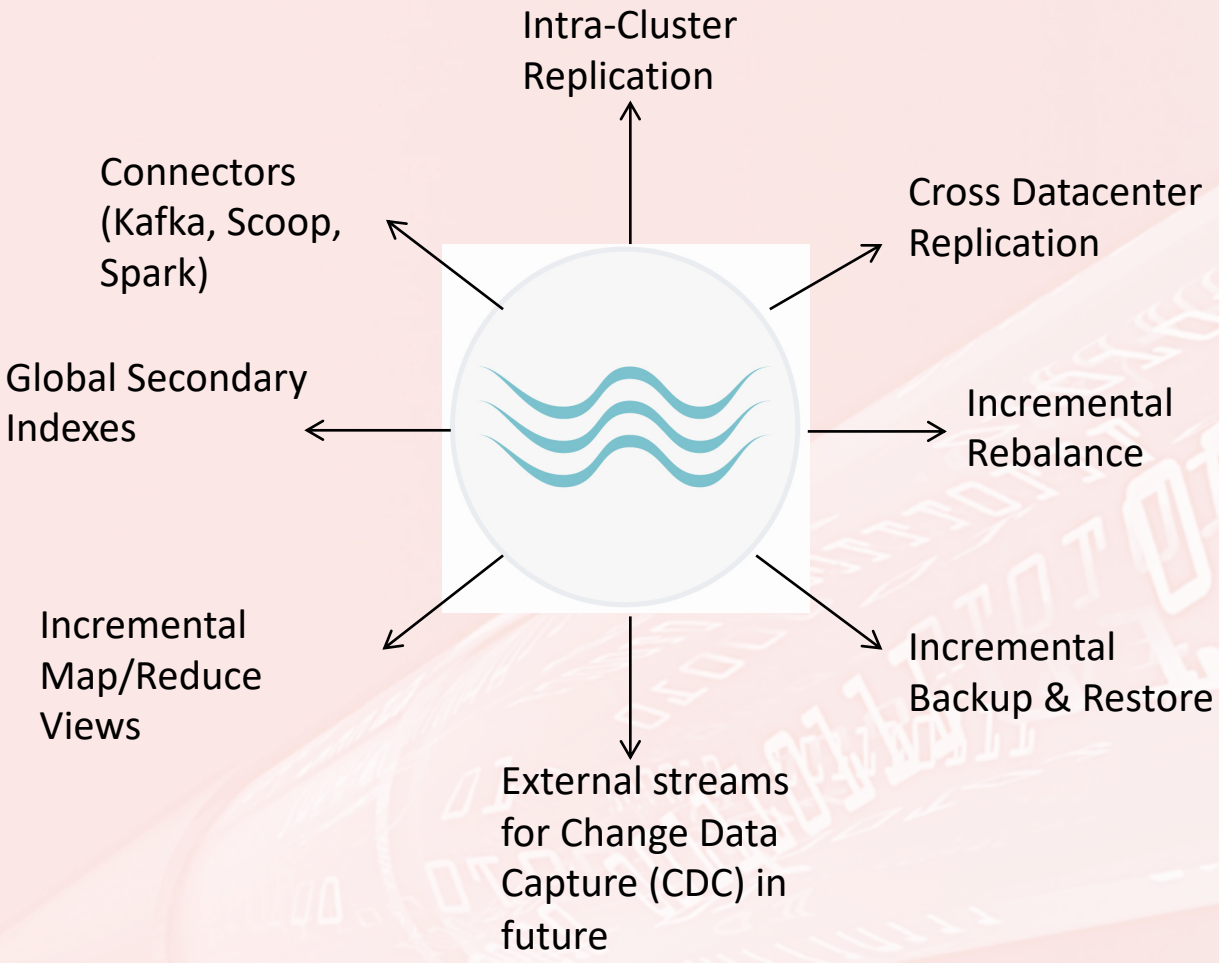
Intra-Cluster Replication – Data Redundancy

Intra-cluster replication is the process of replicating data on multiple servers within a cluster in order to provide data redundancy.



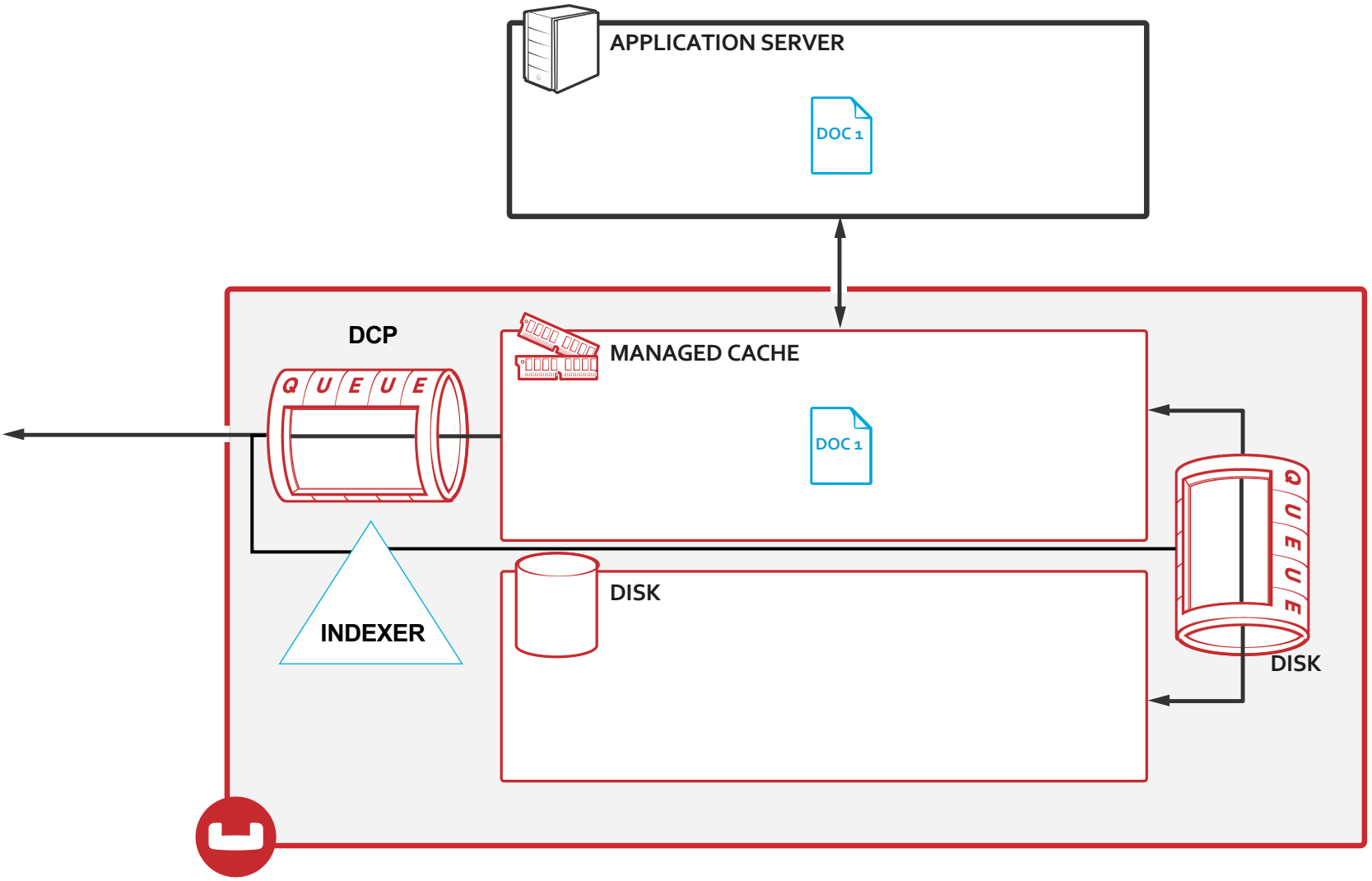
- RAM to RAM replication
- Max of 4 copies of data in a Cluster
- Bandwidth optimized through *de-duplicate, or 'de-dup'* the item

Database Change Protocol – Data Redundancy



- High-Performance, Stream-based Protocol
- Better Resume-ability after blips and failures
- Preserves ordering
- Guarantees consistency

Write Operation – Data Redundancy



- **Tunable Durability:** App gets an ACK when write is successfully in RAM
  - Or RAM+Replicated
  - Or RAM+Persisted
  - Or RAM+Replicated+Persisted
- **DCP based Replication:** writes queued to other nodes
- **Couchstore based Storage:** writes queued for storage

Failover Operation

- Failover *switches-over* to the replicas for a given database
  - Gracefully under node maintenance
  - Immediately under auto-failover
  - Can be triggered manually through the Admin-UI/REST/CLI

- Automatic failover in case of unplanned outages – system failures
  - Can be configured through Admin-UI/REST/CLI
  - Constraints in place to avoid “split-brain” and false positives
    - 5 second delay, multiple heartbeat “pings”
    - Clusters >=3 nodes
    - Only one node down at a time

Autofailover Process

- Failure detection:
  - Failing to reach a node on an operation
  - Regular ping checks among server nodes
- Failover actions:
  - Promoting replicas to active
  - Election of new orchestrator in case of an orchestrator failure
  - Does NOT create missing replicase
- Need rebalance after failover to restore the missing replicas
- Different options after failover:
  - Recover the failed node using full or delta recovery and rebalance
  - Replace the failed node and rebalance
  - Rebalance to a smaller cluster

Recovering from Failover

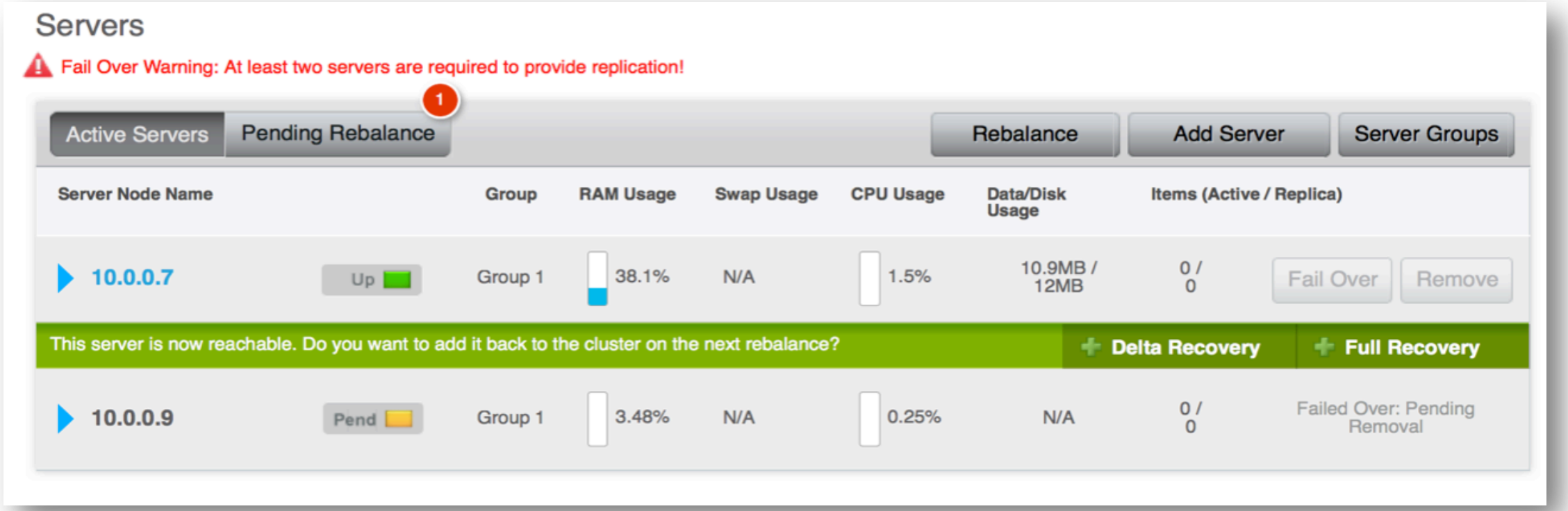
Three options:

1. **Remove the node:**
   - Either Gracefully (Preferred) or Forcefully (e.g. Hard Failures) remove the node from the cluster
   - Rebalance the data across the nodes to ensure a full set of replicas

2. **Full recovery:**
   - e.g. In case of disk loss or corruption
   - Add a failed node into cluster and recover data completely from scratch
   - Allow the node to “fully / completely” catchup on the data from other nodes
   - Typically longer recovery time for large datasets

3. **Delta node recovery:**
   - e.g. In case of transient failure not involving disk loss or corruption
   - Add a failed node into cluster and recover data from where it left off
   - Allow the node to catchup on the data where it left off from other nodes
   - Typically shorter recovery time for large datasets

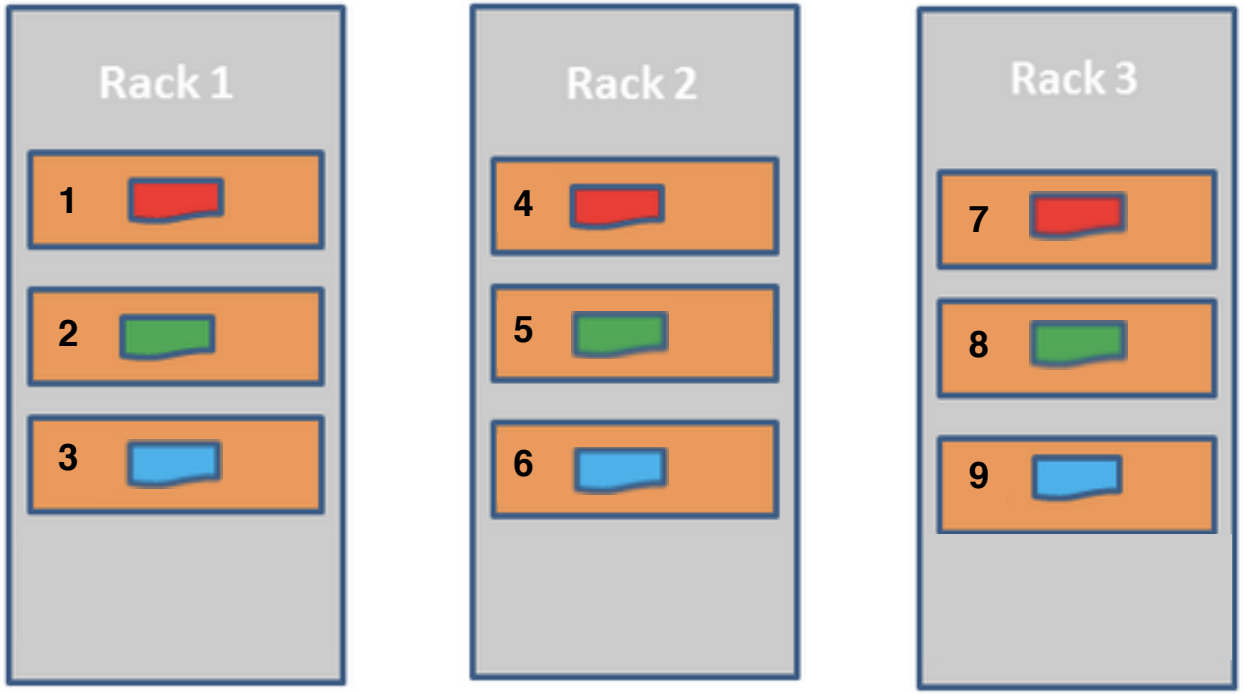
Recovering from Failover: Full / Delta Node Recovery



3 | Rack / Zone Awareness

Rack/Zone Awareness

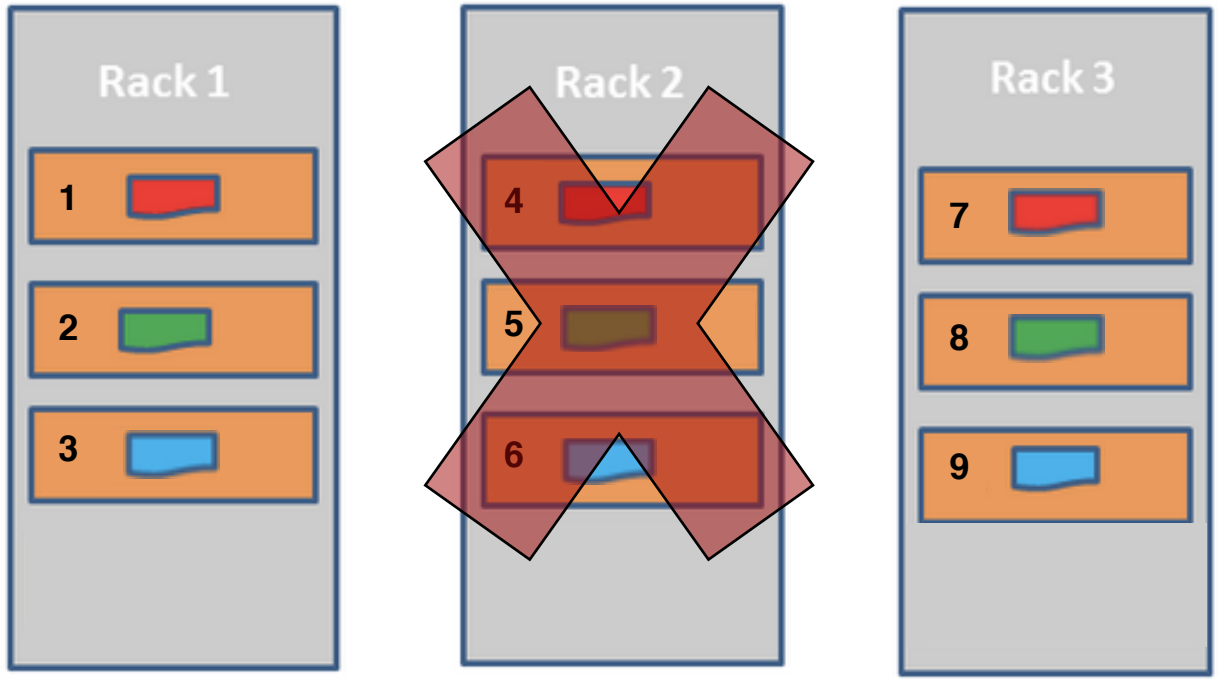
- Informs Couchbase as to the physical distribution of nodes in racks or availability groups.
- Ensures that active and replica vBuckets are distributed across groups



- Servers 1, 2, 3 on Rack 1
- Servers 4, 5, 6 on Rack 2
- Servers 7, 8, 9 on Rack 3
- Cluster has 2 replicas (3 copies of data)
- This is a balanced configuration

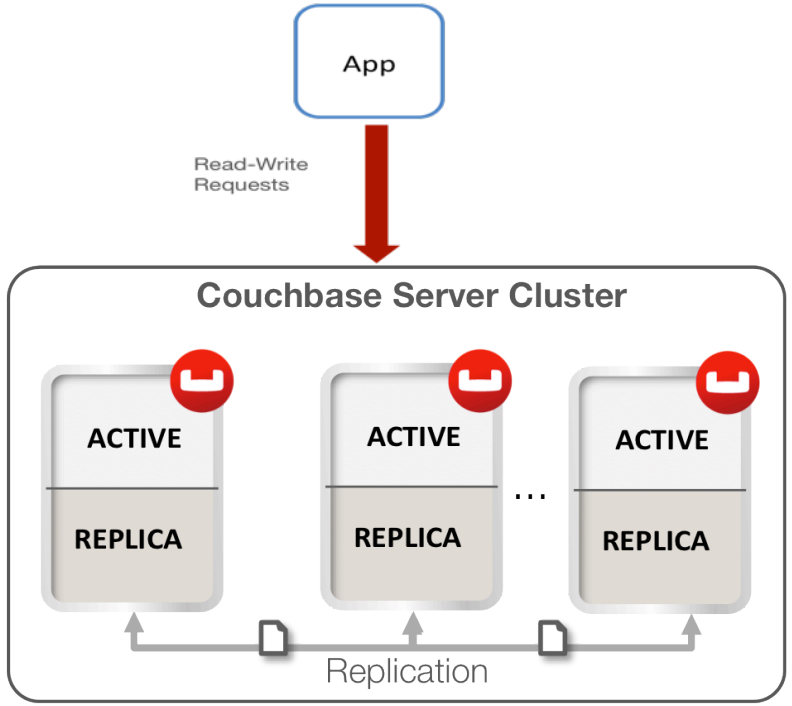
Rack/Zone Awareness

- If an entire rack fails, data is still available
- If an entire cloud availability zone fails, data is still available

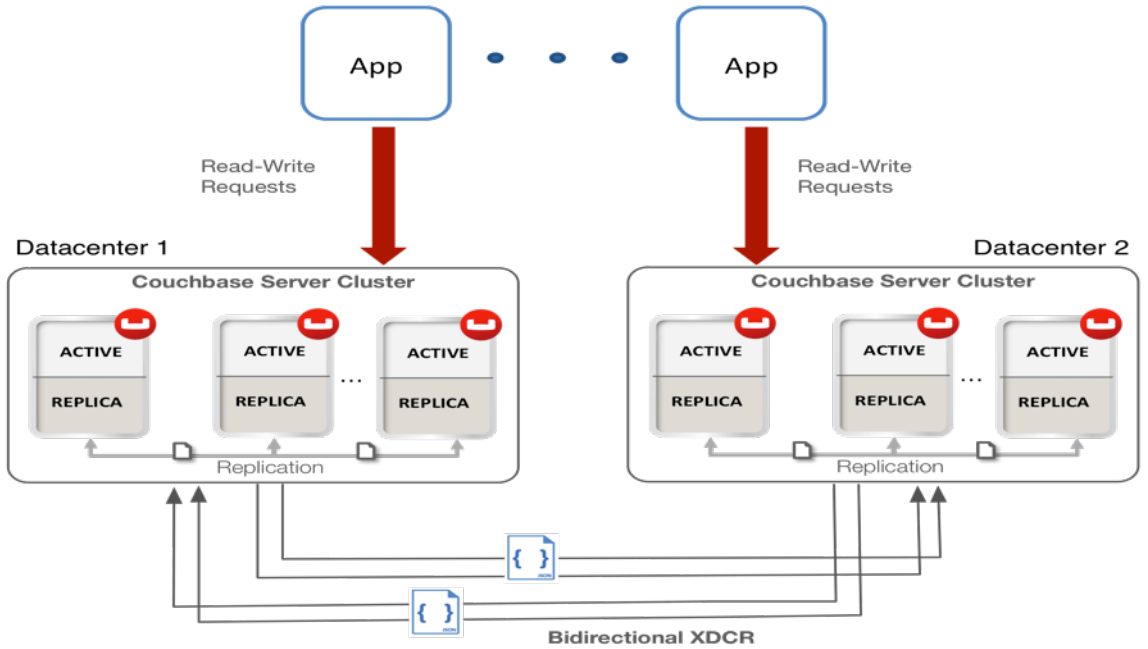


4 | Cross Datacenter Replication (XDCR)

Intra-Cluster vs. Inter-Cluster



VS.

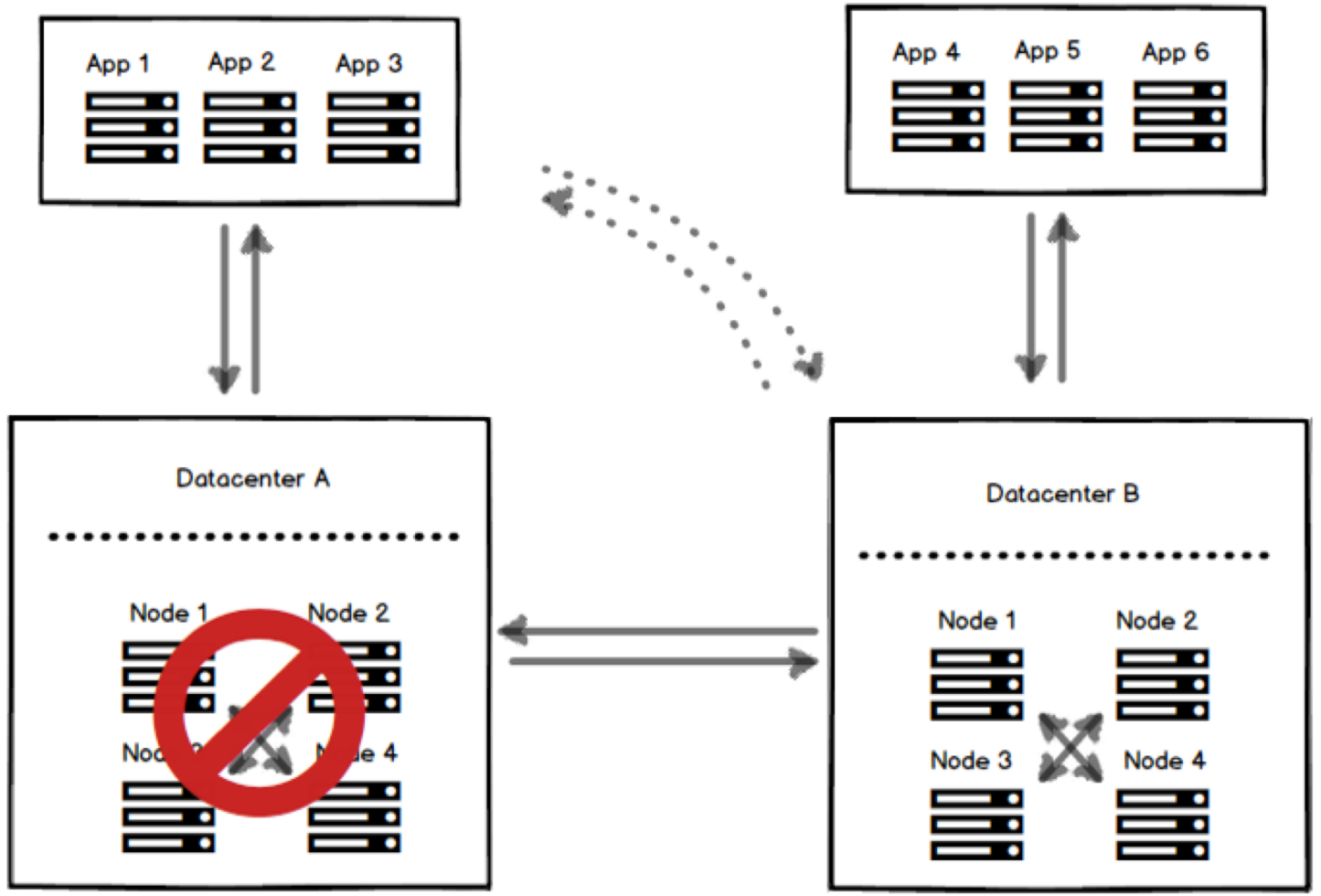


Purposes

- Provide disaster recovery and high availability across data centers
- Support data locality
- For load separation

High Availability - Cross-Cluster FailOver & FailBack

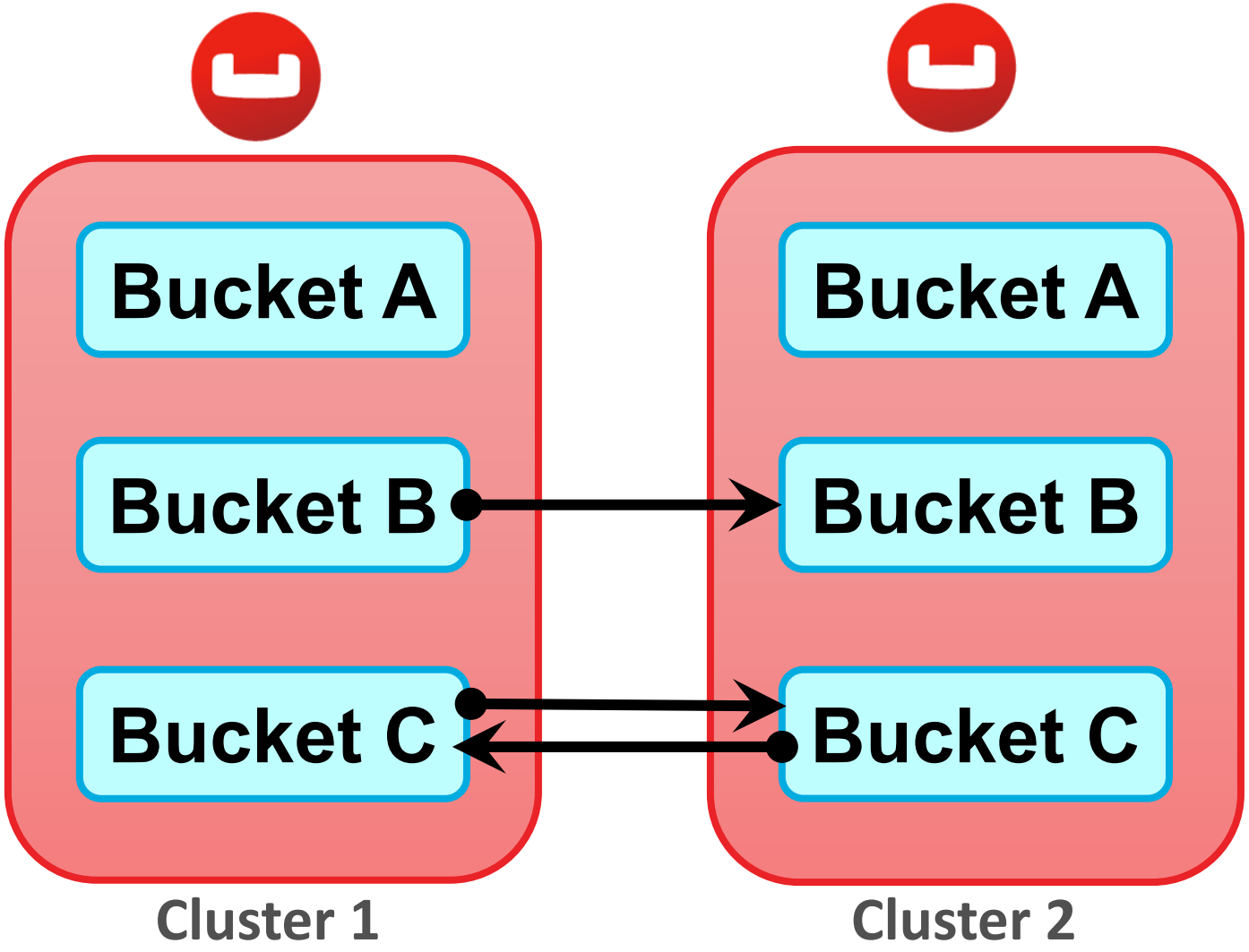
XDCR is used by many customers as an availability solution. Failing over to a secondary cluster following cluster wide or partial failures is critical for customers to be able to provide availability on their database tier.



XDCR: Key Features

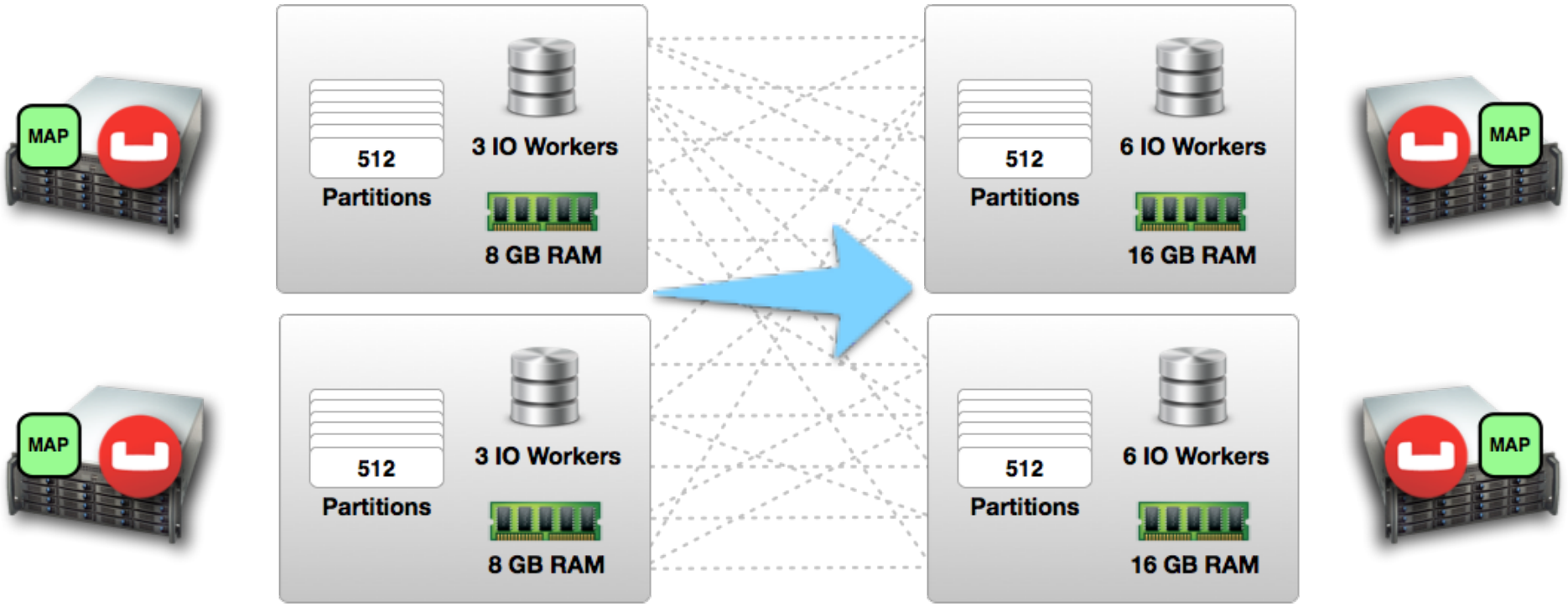
- Configurable at bucket level
- High performing & Scalable
- Resilient
- Supports various topologies and replication schemes, including filtering
- Easy setup of development and test environments

XDCR: From Bucket to Bucket



Cluster-Awareness

- Follows Cluster Map
- Source and destinations can have different number of servers
- Takes topology update into account if e.g. a node of the destination cluster goes down



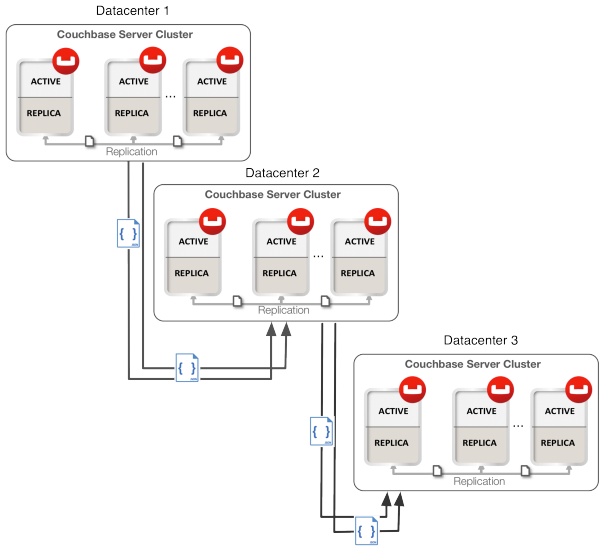
Performance and Scalability

- Asynchronous replication, memory to memory
- Multiple data streams across all shards to move data in parallel
- Replication evenly load balanced across all servers
- Scales by adding more servers
- Compares revisions before transfer

Resiliency

- Replication from all servers to all servers – no single point of failure
- Automatic resume on network errors
- Can be manually paused and resumed
- Source tracks what destination last received via checkpoints
- Resume from last checkpoint

Various Possible Topologies



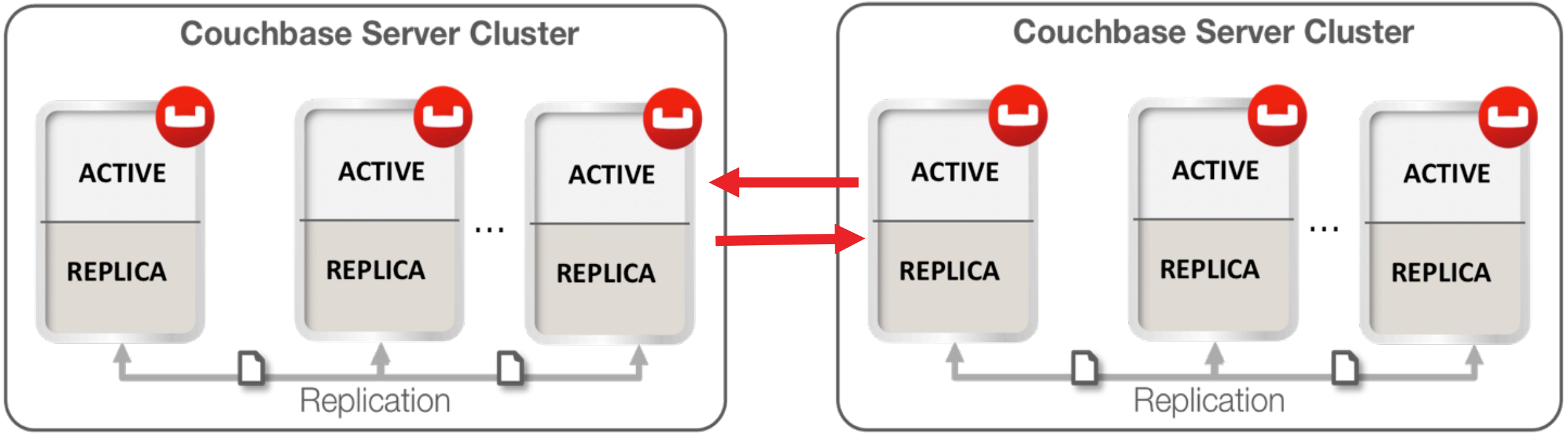
Chain

Data
Aggregation

Filtered
Propagation

Ring, Star, And Many More……

Unidirectional or Bidirectional Replication



Unidirectional (Active – Standby)

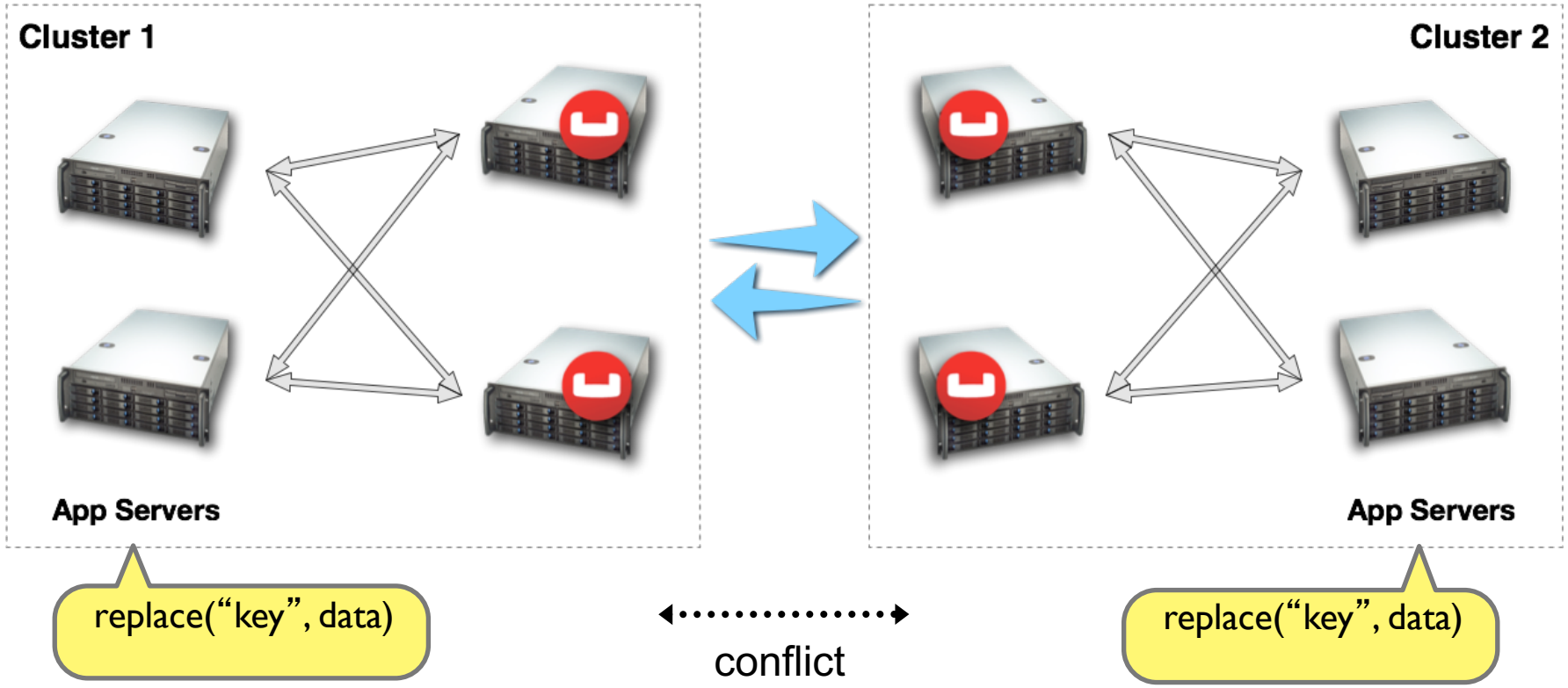
- Hot spare / Disaster Recovery
- Development/Testing copies
- Connectors (Solr, Elasticsearch)
- Integrate to custom consumer

Bidirectional (Active-Active)

- Multiple Active Masters
- Disaster Recovery
- Datacenter Locality

Conflict resolution

- What happens when you write the same key in multiple clusters?



XDCR Conflict Resolution Modes

- **Revision-based Conflict Resolution [Default]:** Current XDCR conflict resolution uses the revision ID (part of document metadata) as the first field to resolve conflicts between two writes across clusters. Revision IDs keep track of the number of mutations to a key, thus the current XDCR conflict resolution can be best characterized as “the most updates wins”.

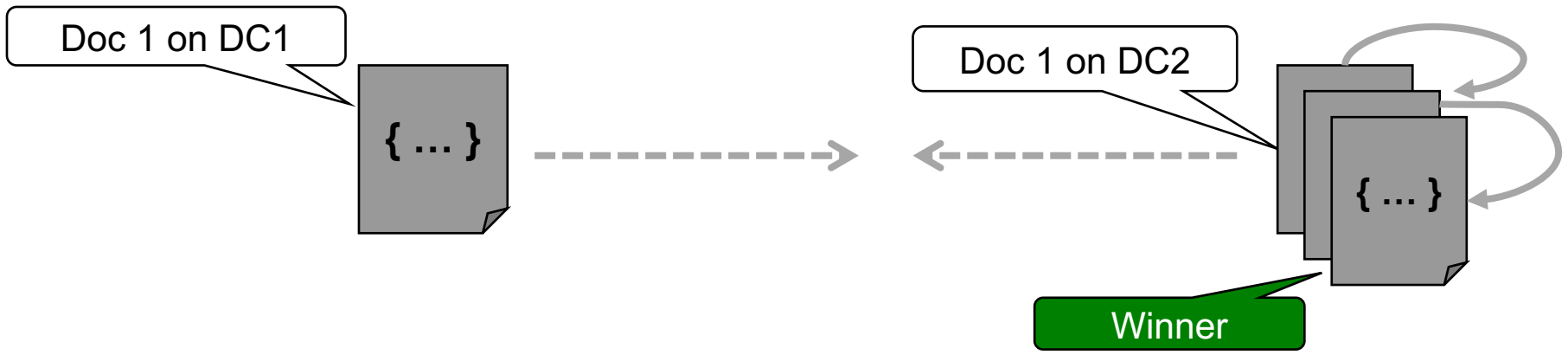
- **Timestamp-based Conflict Resolution [New]:** Timestamp-based conflict resolution uses the hybrid logical clock (part of document metadata) as the first field to resolve conflicts between two writes across clusters. Timestamp has both physical time (NTP) and a logical counter, thus the new XDCR conflict resolution is also known as Last Write Wins (LWW) and is best characterized as “the most recent update wins”.

Revision-Based Conflict Resolution

XDCR is eventually consistent; checks document metadata to resolve conflicts:

1. Numerical sequence (incremented on each mutation)
2. CAS value
3. Expiration (TTL) value

→All clusters will pick the same “winner”



XDCR Use Cases Supported for Conflict Resolution

- Unidirectional Replication
  - Hot spare/ Disaster Recovery
- Bidirectional Replication
  - Datacenter Locality
- High Availability
  - Cross-Cluster FailOver and FailBack

Caution

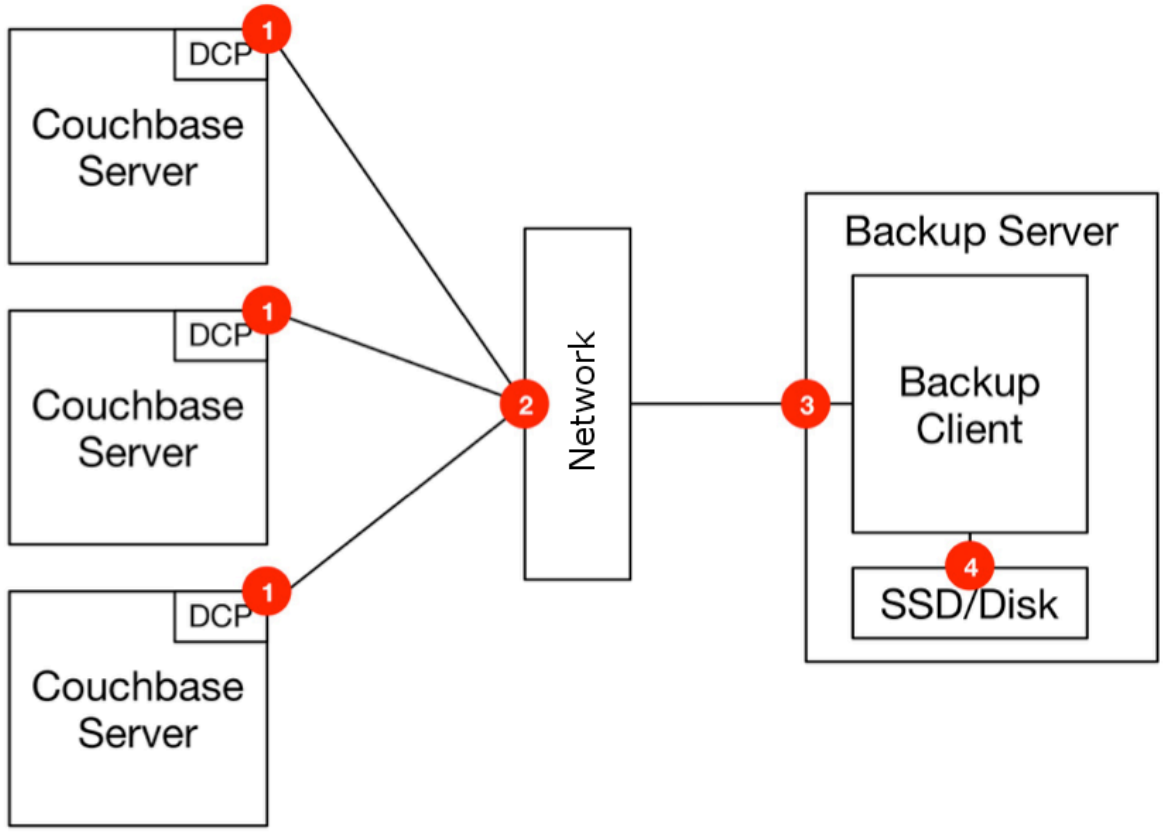
- **Avoid updating the same document in multiple clusters with bi-directional XDCR**
  - Be sure to understand the conflict resolution rules
- **Best Practices**
  - Data Center stickiness
    - Keep users/transactions isolated to a DC
    - Only redirect to another DC in case of major outage
  - Use separate key spaces (e.g. DC prefix) to avoid conflicts on individual documents. Example:
    - dc1::user:a9838-s92-s00
    - dc2::user:293ba-293-922

Advanced Settings

Parameter	Default	Description
Optimistic replication threshold	256	If the size of a document is higher than this threshold then XDCR will send a getMeta request (in batches) from the source cluster to the destination cluster in order to find out if the document needs to be sent over.
Source nozzles per node	2	Controls the parallelism
Target nozzles per node	2	Controls the parallelism
Checkpoint interval	1800	Time in seconds between checkpoints. This defines the amount of data which has to be resent in case of a communication failure.
Batch count	500	Controls the number of documents to be transferred in one batch.
Batch size (kB)	2048	Limits the size of a batch in KB.
Failure retry interval (s)	10	Time in seconds before XDCR retires to resume the replication after a failure.
Filter	None	The filter expression allows you to limit the data which will be sent over the wire by using a regular expression on the document key.

6 | Backup and Restore

Couchbase Backup Architecture

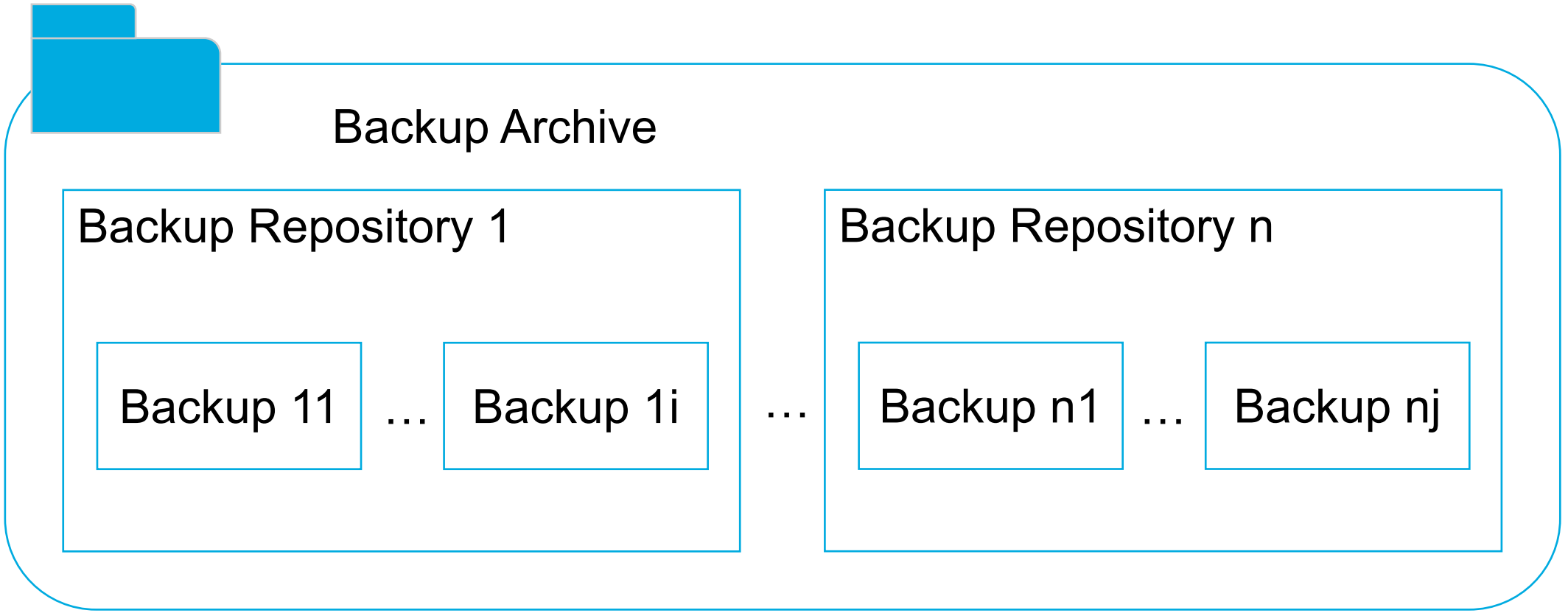


- Backup client on a separate machine
- Streaming from memory
- Transparent and resilient to cluster topology changes
- Backup from memory
- Efficient incremental backups due to DCP

cbbackupmgr Tool

```
cbbackupmgr [<command>] [<args>]

  backup          Backup a Couchbase cluster

  compact         Compact an incremental backup

  config          Create a new backup configuration

  help            Get extended help for a subcommand

  list            List the archive contents

  merge           Merge incremental backups together

  remove          Delete a backup permanently

  restore         Restore an incremental backup
```

Backup Archive Structure



Periodic Merge

- Initial backup is a full one
- Each following backup in the same repository is a new increment
- Increments can be merged into a new full backup

```
Sunday:     Config Backup Repo     (0 backups in backup repo)
Sunday:     Backup 1               (1 backup in backup repo)      Full
Monday:     Backup 2               (2 backups in backup repo)
Tuesday:    Backup 3               (3 backups in backup repo)
Wednesday:  Backup 4               (4 backups in backup repo)     Increment
Thursday:   Backup 5               (5 backups in backup repo)
Friday:     Backup 6               (6 backups in backup repo)
Saturday:   Backup 7               (7 backups in backup repo)
Sunday:     Backup 8               (8 backups in backup repo)
Sunday:     Merge 1-7              (2 backups in backup repo)     Merge
Monday:     Backup 9               (3 backups in backup repo)
Tuesday:    Backup 10              (4 backups in backup repo)
Wednesday:  Backup 11              (5 backups in backup repo)
Thursday:   Backup 12              (6 backups in backup repo)
Friday:     Backup 13              (7 backups in backup repo)
Saturday:   Backup 14              (8 backups in backup repo)
Sunday:     Backup 15              (9 backups in backup repo)
Sunday:     Merge 7-14             (2 backups in backup repo)
```

Full/incremental Backup

- Initial backup is a full one
- Each following backup is a new increment
- A new full one means to create a new repository

```
Sunday:     Config Backup Repo 1   (0 backups in backup repo)
Sunday:     Backup 1               (1 backup in backup repo)      Full
Monday:     Backup 2               (2 backups in backup repo)
Tuesday:    Backup 3               (3 backups in backup repo)
Wednesday:  Backup 4               (4 backups in backup repo)     Increment
Thursday:   Backup 5               (5 backups in backup repo)
Friday:     Backup 6               (6 backups in backup repo)
Saturday:   Backup 7               (7 backups in backup repo)
Sunday:     Config Backup Repo 2   (0 backup in backup repo)
Sunday:     Backup 8               (1 backups in backup repo)     Full
Monday:     Backup 9               (2 backups in backup repo)
Tuesday:    Backup 10              (3 backups in backup repo)
Wednesday:  Backup 11              (4 backups in backup repo)
Thursday:   Backup 12              (5 backups in backup repo)
Friday:     Backup 13              (6 backups in backup repo)
Saturday:   Backup 14              (7 backups in backup repo)
Sunday:     Create Backup Repo 3   (0 backups in backup repo)
Sunday:     Backup 15              (1 backup in backup repo)      Full
```

Full Backup only

- Each new full backup means to create a new repository

```
Sunday:    Config Backup Repo 1   (0 backups in backup repo)
Sunday:    Backup 1               (1 backup in backup repo)      Full
Monday:    Config Backup Repo 2   (0 backups in backup repo)
Monday:    Backup 2               (1 backup in backup repo)
Tuesday:   Config Backup Repo 3   (0 backups in backup repo)      Full
Tuesday:   Backup 3               (1 backup in backup repo)
Wednesday: Config Backup Repo 4   (0 backups in backup repo)
Wednesday: Backup 4               (1 backup in backup repo)
Thursday:  Config Backup Repo 5   (0 backups in backup repo)
Thursday:  Backup 5               (1 backup in backup repo)
Friday:    Config Backup Repo 6   (0 backups in backup repo)
Friday:    Backup 6               (1 backup in backup repo)
Saturday:  Config Backup Repo 7   (0 backups in backup repo)
Saturday:  Backup 7               (1 backup in backup repo)
Sunday:    Config Backup Repo 8   (0 backups in backup repo)
Sunday:    Backup 8               (1 backup in backup repo)
```

Performance Recommendations for 6.0

- Create frequent incremental backups
- Merge incremental backups instead of performing regular full backups
- Run backup with multiple threads, e.g. `-t 16`
- Enable value compression: `--value-compression compressed`
- Backup client is CPU intensive, DO NOT run it on a cluster node
- Run compact command after backup to save disk space