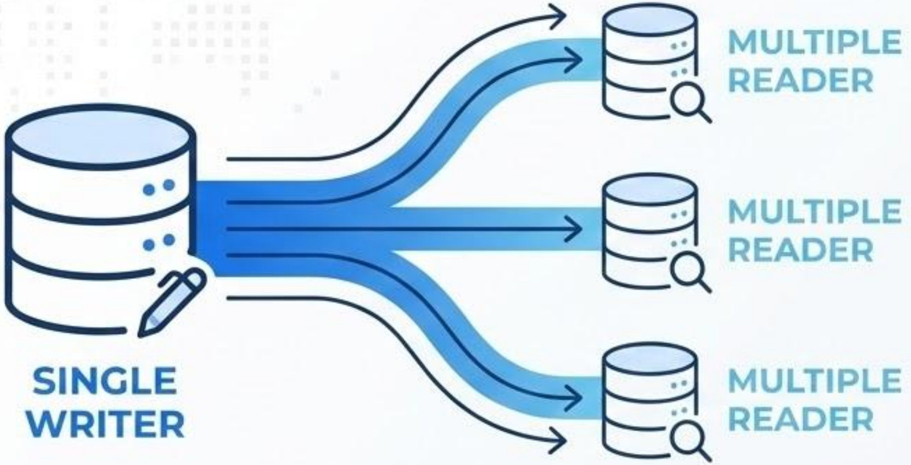
Multi-Writer CDC Challenges

Sunny Bains

PingCAP

FOSDEM 2026

Traditional Replication



- Single writer multiple reader
- Ordering of events is implicit

Examples

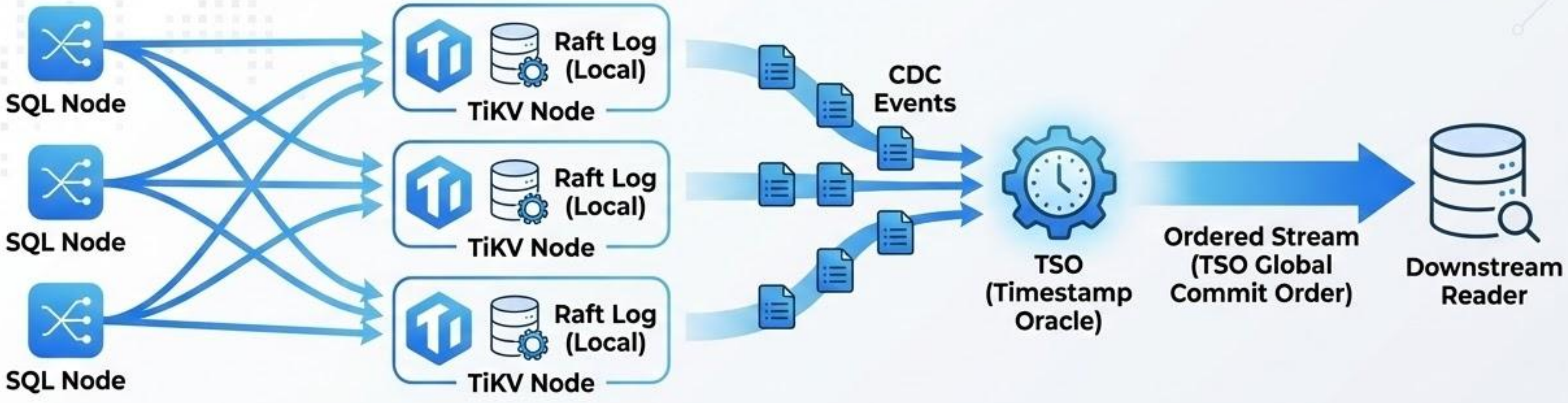
MySQL binlog

PostgreSQL WAL replication

TiDB CDC

Distributed Write Architecture

Global TSO Ordering & Commit



- Multiple SQL Nodes write to multiple TiKV nodes.
- Distributed transactions can touch multiple TiKV nodes.
- The Raft log stores the replicated state machine and distributed transactions.
- The Raft log is per TiKV node not a global log.

- CDC Events have to be read from the Raft logs and then ordered according to the global commit order based on the commit TSO when propagating the CDC.
- Downstream readers must see events in the TSO global order in which they were committed when reading from the CDC stream.

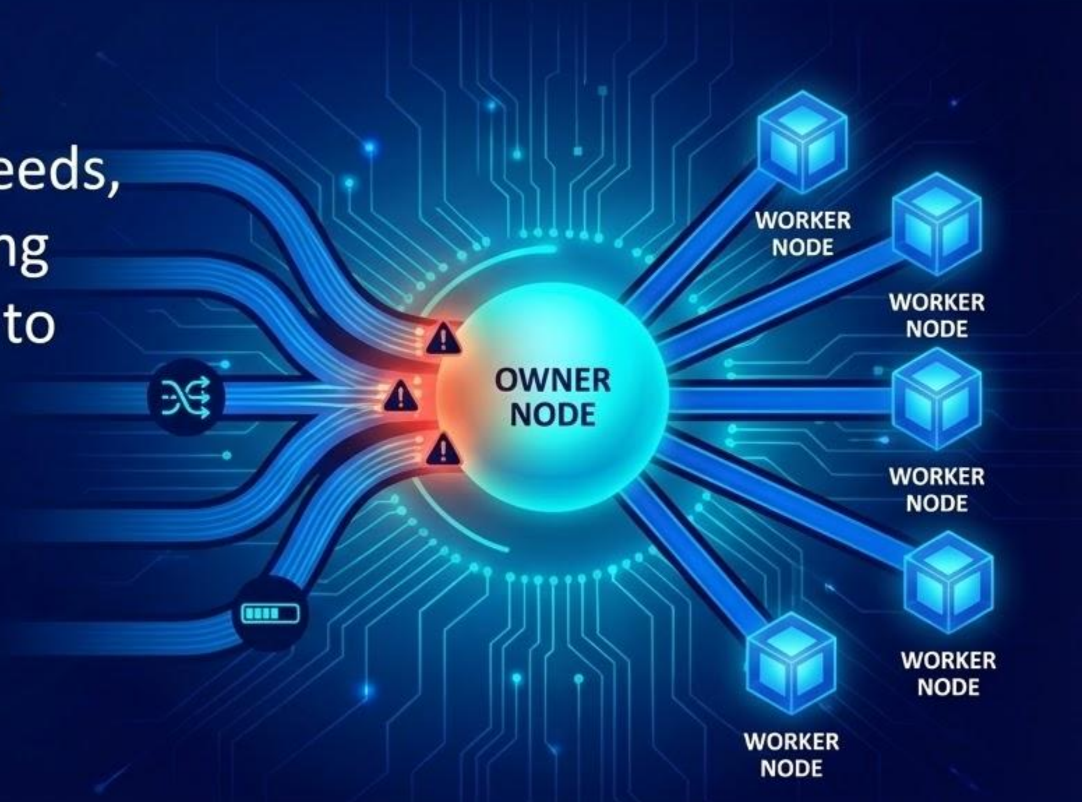
The Ordering Guarantee

The execution order is: DML → DDL → DML. TiCDC ensures schema correctness by coordinating execution.

PingCAP Docs

PREVIOUS ARCHITECTURE

- Centralized Owner node
- Owner handles changefeeds, table progress, scheduling
- Workers tightly coupled to Owner logic



MOTIVATION & BACKGROUND

- Growing cluster scale exposes architectural limits
- Owner-centric design becomes a bottleneck
- Increasing complexity impacts stability and operability
- Cloud-native and multi-tenant support are constrained

Scaling Challenges with High Region Count

1 Million Regions across 3 TiKV Nodes
- ~333K regions per TiKV

Concurrent gRPC Streams
- ~333K per TiCDC capture

ResolvedTs Messages
- ~333K every tick interval

This Creates:

Memory Pressure
Each Delegate/Resolver holds state

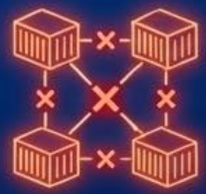
CPU Overhead
Processing millions of small messages

Network Amplification
ResolvedTs fanout

Real-World Symptoms (From TiKV issues):

⚠️ CDC endpoint CPU 100% with just 800 ops/s (#9981)
⚠️ TiKV OOM from buffered CDC events (#8168, #9996)
⚠️ Resolver memory unbounded growth (#15412)

CLOUD-NATIVE CHALLENGES



- Difficult multi-tenancy

- No fine-grained resource control

- High operational complexity

SCALABILITY LIMITATIONS

- Owner is a single-point bottleneck
- ~100k tables, ~400 changefeeds, ~10 nodes
- ~700MB/s throughput, ~3 DDLs/sec

Old architecture vs new Architecture

The “Classic” TiCDC architecture faced scaling challenges that were addressed in v8.5.4-r1.

Previous limitations

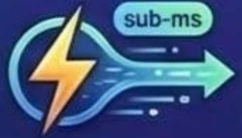
- **The 50ms Timer Trap:**
  “Owner” node’s 50ms polling loop capped DDL speed (~3/sec), introducing a “lag floor.”

- **Stateful/Stateless Mixing:**
  Tightly coupled components. Node failure caused massive “Stop-the-world” latency spikes.

- **Large Transaction Buffering:**
  GB-sized transactions overwhelmed Sorter memory, leading to OOM errors or disk thrashing.

Ongoing & Future Improvements

- **Event-Driven Architecture:**
  Moving to a purely event-driven model for sub-millisecond processing and higher DDL throughput.



- **Task Splitting:**
  Splitting single table replication across multiple nodes for "hotspot" (ultra-high write volume) handling.


Hotspot Handling

- **Decoupled Log Service:**
  Dedicated stateful “Log Service” for sorting/storage, leaving stateless “Processors” to push data, enabling faster scale-out.


Faster Scale-Out

CORE DESIGN PRINCIPLES

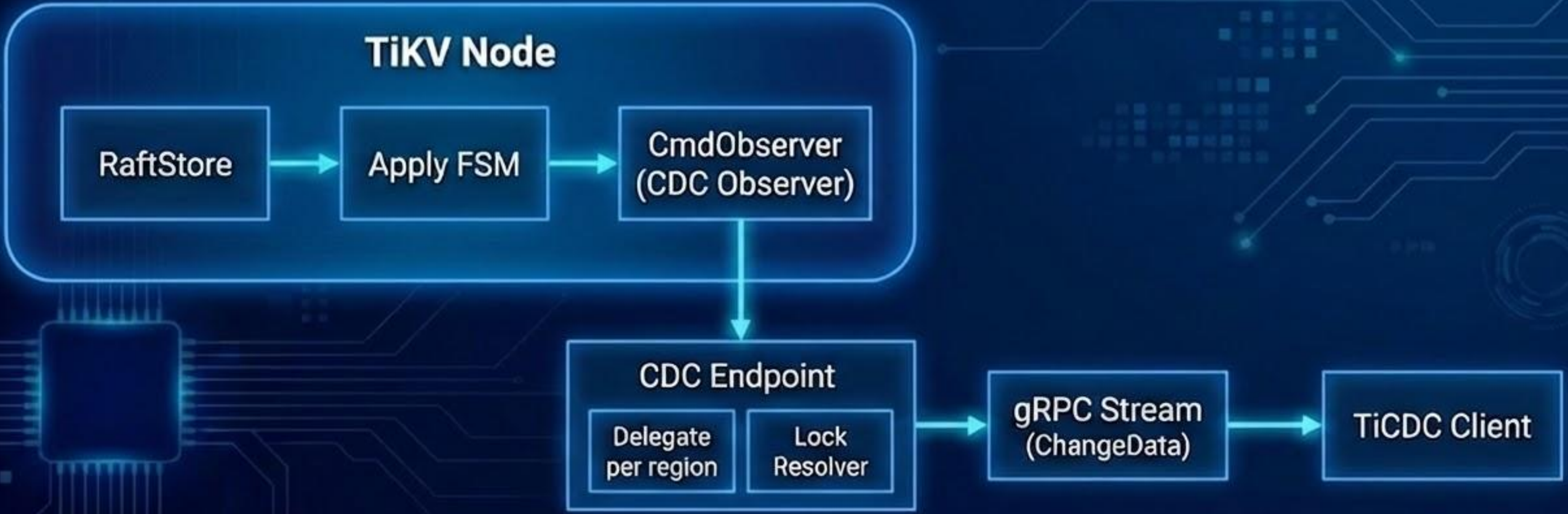
Decentralization

Event-driven processing

Clear separation of concerns

NEW ARCHITECTURE COMPONENTS

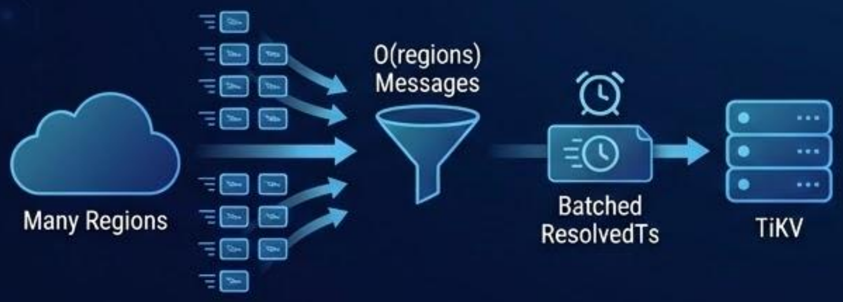
How CDC Hooks into RaftStore (Coprocessor Observer Pattern)



CDC doesn't intercept Raft messages.
Instead, it uses the **Coprocessor Observer** pattern.

Batched ResolvedTs & Multiplexed Streams (Optimizations)

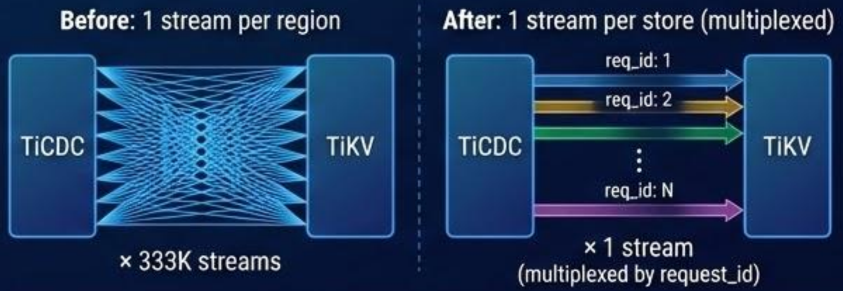
1. Batched ResolvedTs (Network Optimization)



Reduces message count from O(regions) to O(1) per interval.

2. Multiplexed Streams (Connection Optimization)

Single gRPC stream handles multiple regions



The `request_id` field in ChangeDataRequest enables this multiplexing.

Additional Optimizations

3. Region Merge (B-Tree)

4. Memory Quota & Backpressure

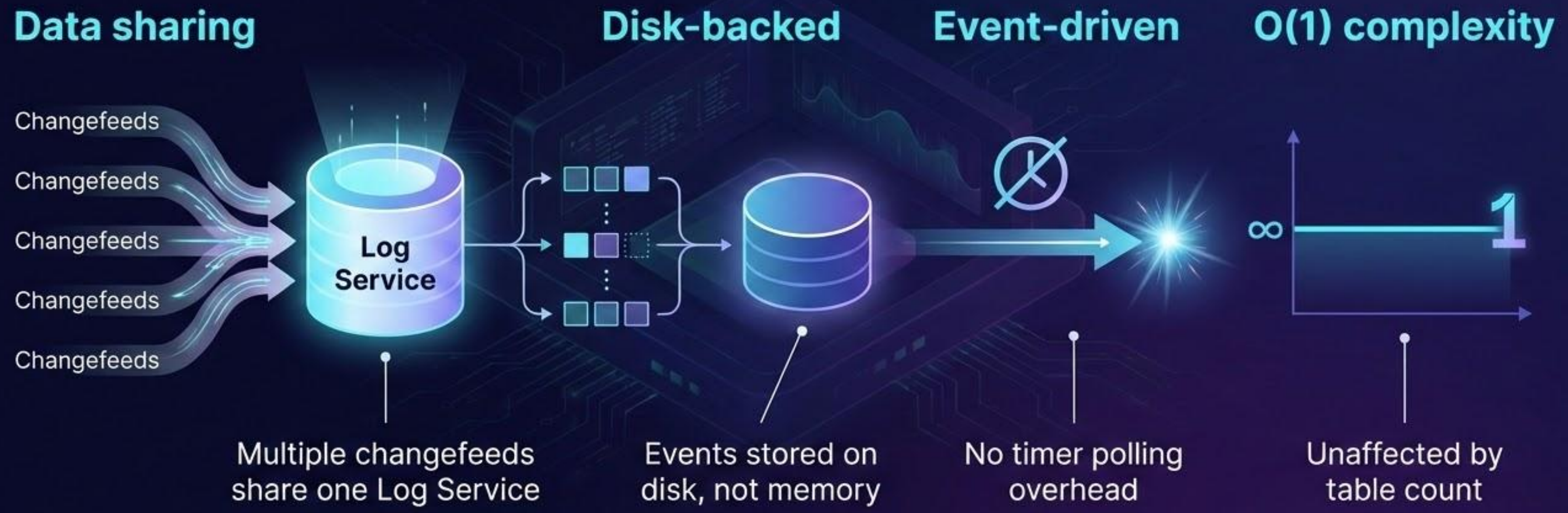
When quota exhausted → backpressure → slow down event generation.

5. Resolved-by-Raft

Avoids tracking every lock; uses Raft's applied index.

TiCDC New Architecture (v8.5+)

The new architecture fundamentally redesigns for scale:



Mounter's Role

Sits between the sorter and sink in the data flow:

TiKV → Puller → Sorter → **Mounter** → Sink → Downstream

What It Does

Decodes raw KV pairs

Deserializes TiKV CDC low-level key-value changes into structured row data.

Schema resolution

Maps raw bytes to column names/types using table schema. Handles mid-stream DDL changes.

Constructs row change events

Produces a canonical representation with table, operation, new, and old column values.

Formats for downstream consumption

Prepares data in formats the sink understands (e.g., SQL, JSON/Avro).

Why It's Separate

- Decoupling encoding logic allows Sorter to work on opaque bytes (faster, simpler) and Sink to receive clean, structured events.
- The mounter is a **compute-heavy** component (lots of deserialization).

Technical Deep Dive: The Heart of TiCDC

A. The Resolved TS (Timestamp) Mechanism

- **Watermark Tracking**: Each TiKV node maintains a "Resolved TS" (no earlier writes guaranteed).
- **Global Aggregation**: TiCDC pulls Resolved TS from all Regions.
- **The Bar**: Calculates Minimum Resolved TS as the "safe-to-emit" line.

B. Event Sorting and Transaction Reconstruction

- **Sorter**: Buffers and flushes interleaved logs in strict chronological order (often via Pebble).
- **Mounter**: Transforms raw "Key:Value" bytes into RowChangedEvent using schema snapshot.

Cluster Startup Workflow

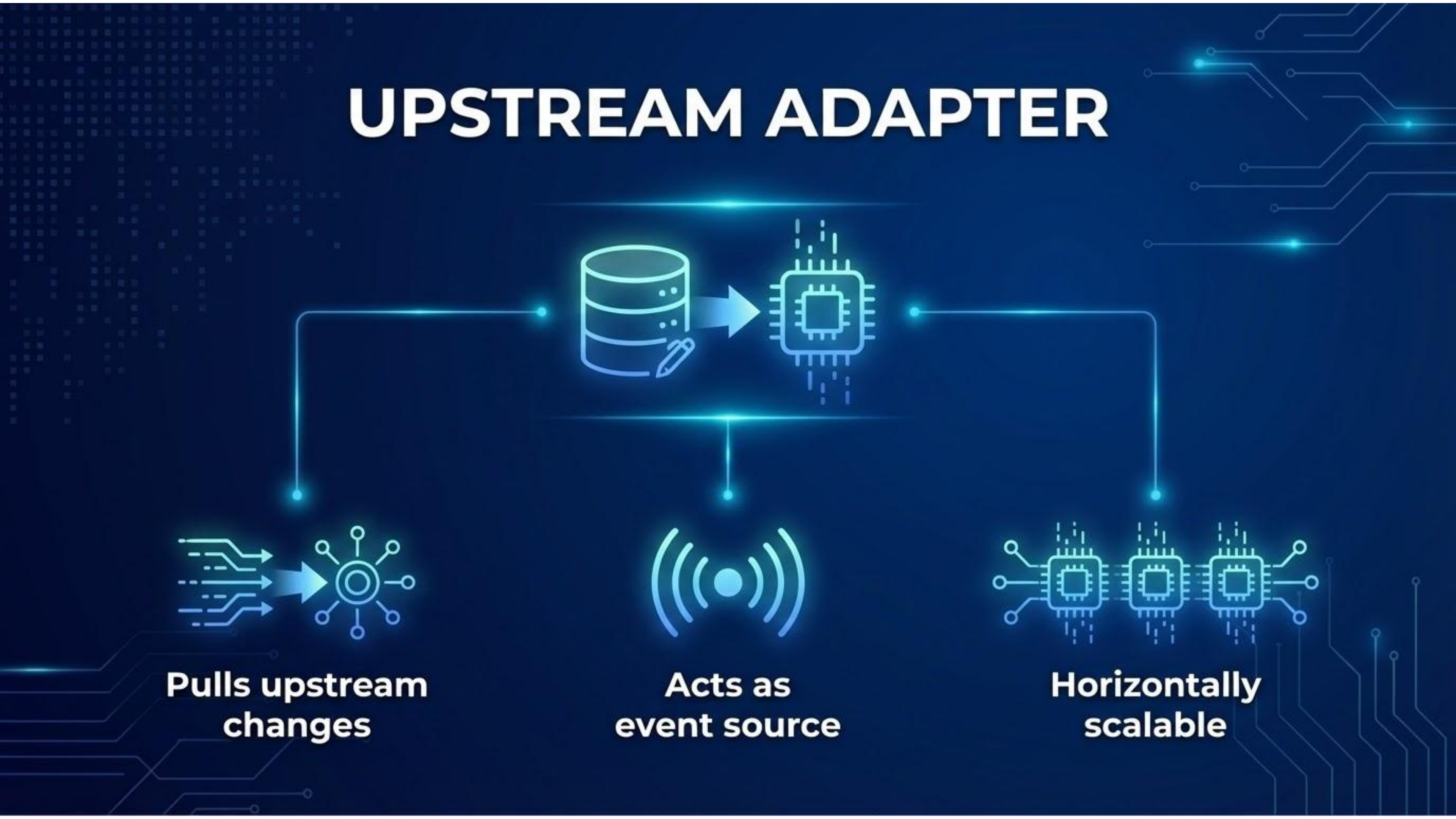
Instance Join / Leave

Special Scenarios

Network isolation handling

Partitioned tables

Upgrade / downgrade compatibility


UPSTREAM ADAPTER
Pulls upstream changes
Acts as event source
Horizontally scalable


LOG SERVICE
Local disk event storage
Table Stream Manager
Event Store
Schema Store

DOWNSTREAM ADAPTER

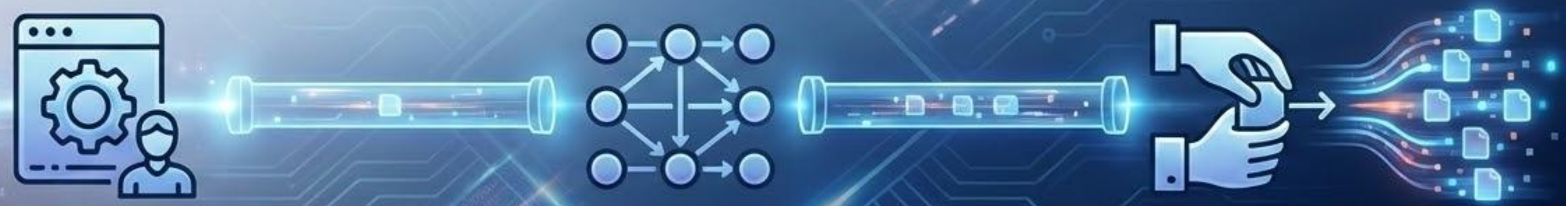

COORDINATOR
ETCD
Schedules changefeeds
Manages metadata in ETCD
No data-plane responsibilities

New Changefeed Creation

Persistence & Metadata

New Changefeed Creation

- Create **maintainer**
- Create **dispatchers**
- Fetch and push events



Large Changefeeds & Tables

More changefeeds supported

Large tables can be split

Transaction integrity preserved

Third-Party Sink Support

Coordinator vs Maintainer

Failure Handling

Crash recovery via metadata

Automatic state restoration

ETCD-based leadership

Distributed consensus

Brain-split prevention

Quorum-based stability

Summary

https://github.com/pingcap/ticdc