Remus: Efficient Live Migration for Distributed Databases with Snapshot Isolation

Junbin Kang, Le Cai, Feifei Li, Xingxuan Zhou, Wei Cao, Songlu Cai, Daming Shao

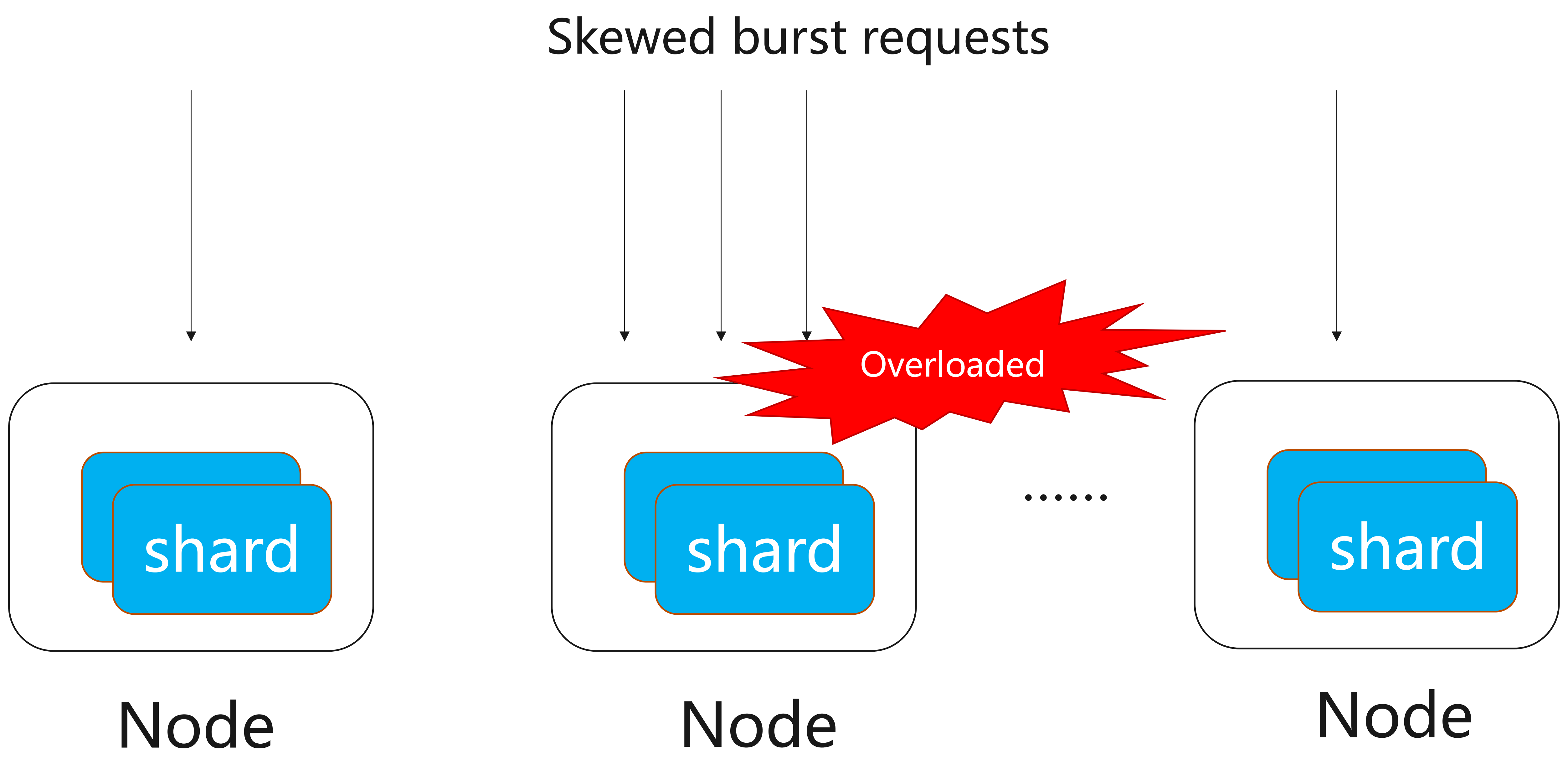
Alibaba Group

Shared-nothing databases on the cloud

- Cost efficiency
  - ✓ on-demand resources provision
  - ✓ Elasticity at workload of high concurrency

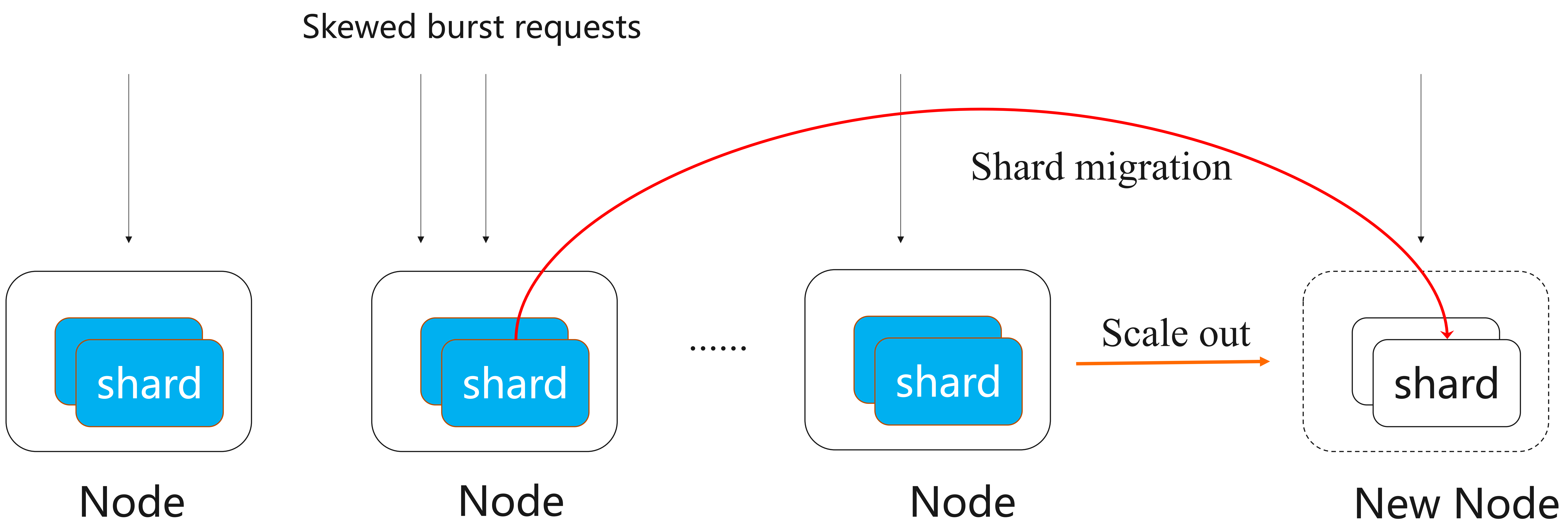
- Dynamic workload
  - ✓ Burst requests (e.g., double 11 shopping festival)
  - ✓ Skewed access and hotspots also change over time

- Challenge: static sharding is hard to react to dynamic workloads on the cloud

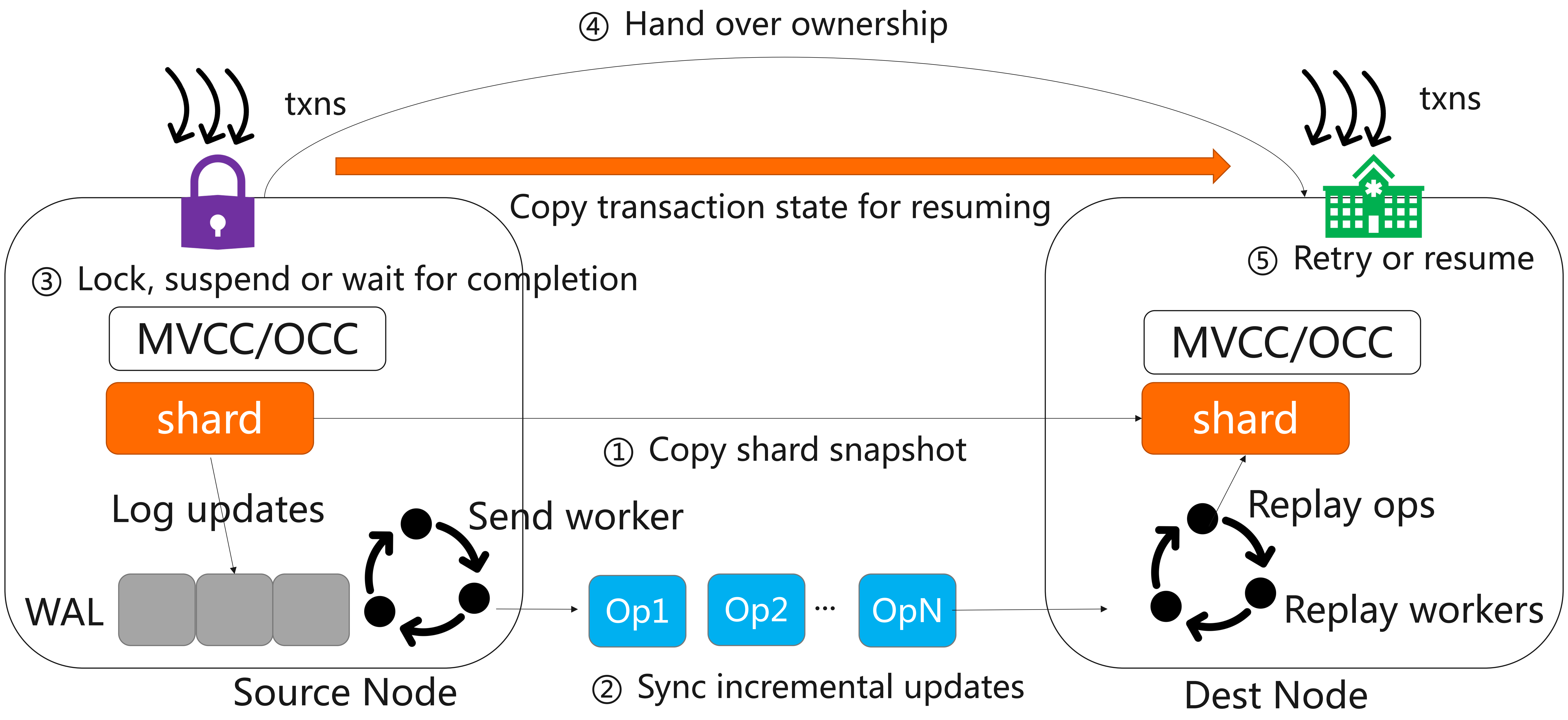


Live migration: key to offer elasticity with load balance

- Provisioning more VMs under peak loads
- Un-provisioning some VMs under light loads to save costs
- Migrating shards from overloaded nodes to the others for load balance.

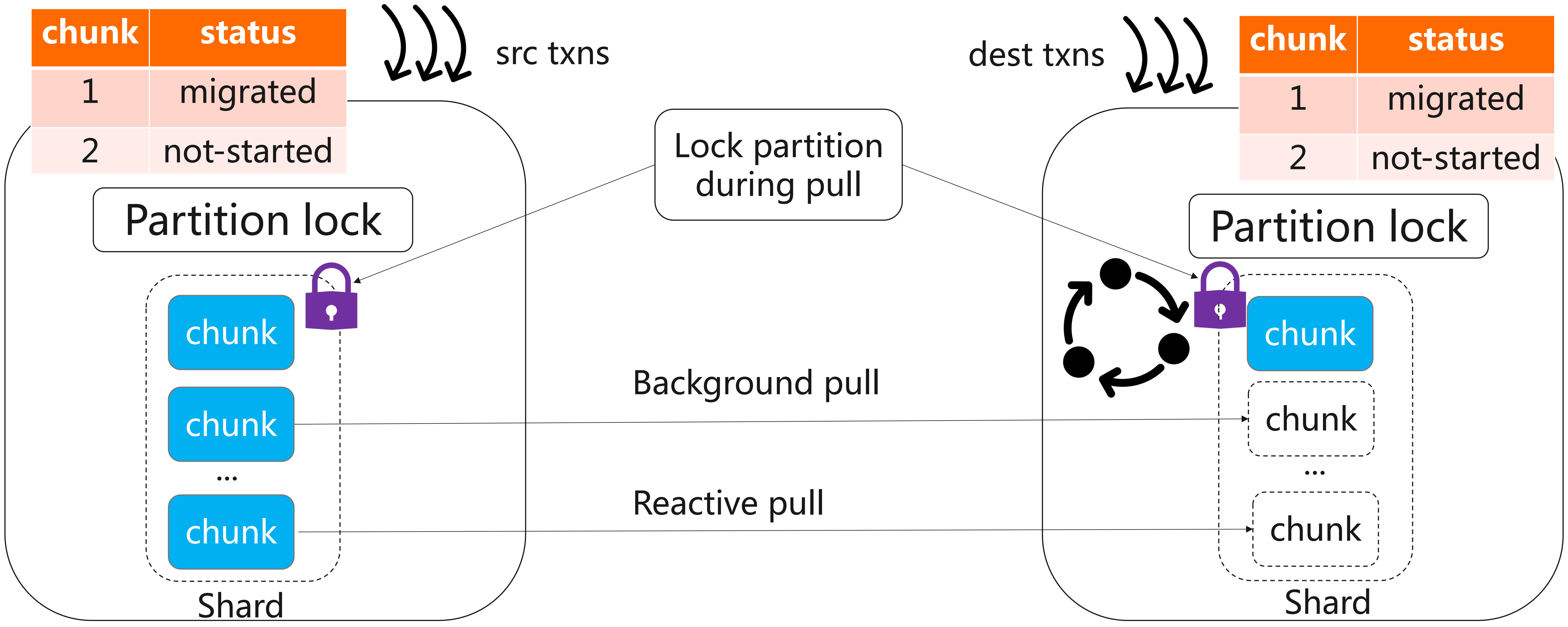


Existing approaches: push-migration



- Existing push-migration incurs transaction aborts or significant downtime
  - ✓ Lock-and-abort [Citus, SIGMOD '21]: lock the shard and abort blocked transactions after handover
  - ✓ Suspend-and-resume [Albatross, VLDB '11]: suspend src txns, copy transaction state and resume txns on destination
  - ✓ Wait-and-remaster [DynaMast, ICDE' 20]: suspend routing and wait for exsiting txns to completion on the source before handover

Existing approaches: pull-migration



- The state-of-the-art pull-migration [Squall, SIGMOD '15]:
  - ✓ Use chunk status table to track each chunk's migration status
  - ✓ Pull chunks on demand by accessing transactions and in the background by workers
  - ✓ Leverage partition locks in H-Store to maintain consistency for on-the-fly pulls
- Source transaction would fail if its accessing chunk is migrated -> transaction aborts
- Partition locking would incur significant throughput drops and latency increases

Challenge #1: costs of live migration

- Existing approaches often incur some costs:
  - ✓ Failed transactions (e.g., Squall [SIGMOD '15] , Zephyr [SIGMOD '11], Citus [SIGMOD '21])
  - ✓ Service downtime (e.g., Citus, Albatross [VLDB '11], DynaMast [ICDE' 20])
  - ✓ Performance impact in throughput and latency (e.g., Squall, Citus, Zephyr, Albatross)

- Challenge: theses migration costs may violate the strict SLA on the cloud
  - ✓ Alibaba Cloud SLA definition [1]: Monthly Uptime Percentage=100%-Average Error Rate
  - ✓ Failed transactions from migration may result in SLA violation on Alibaba Cloud
  - ✓ 99.95% SLA means: for 10k TPS, no more than 5 failed txns per second from migration
  - ✓ Latency sensitive applications such as online games require even more strict SLO guarantee
    - For example, > 100 ms tail latency may severely affect users' game experiences.

[1] https://www.alibabacloud.com/help/en/legal/latest/database-management-service-level-agreement

Challenge #2 live migration under hybrid workloads

阿里云

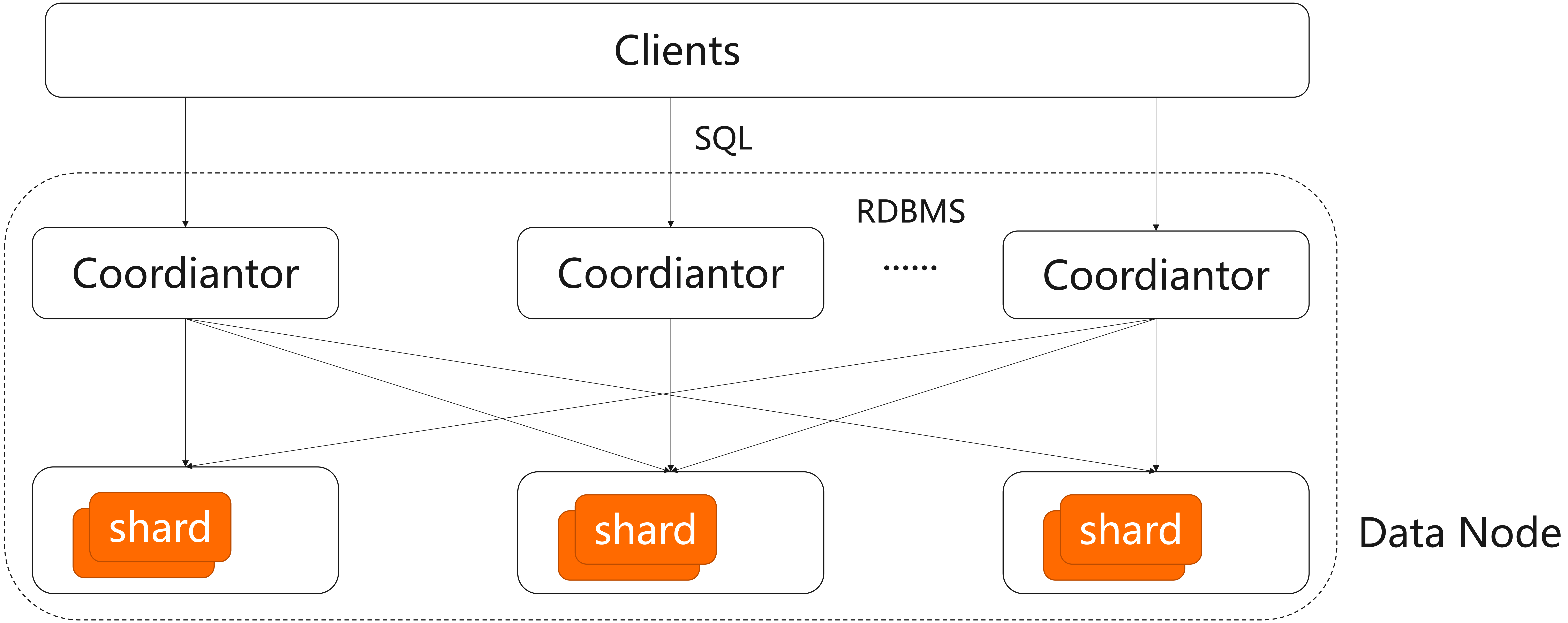
- Customers may run hybrid workloads on their cloud database
  - ✓ Short OLTP transactions, e.g., stored procedures and client-interactive transactions
  - ✓ Long lived transactions (LLT), e.g., analytic queries, batch inserts and a mixed of them for ETL
  - ✓ Hybrid workloads of OLTP and LLT are common in HTAP, IoT and HSAP [VLDB '21] scenarios
    - Real-time queries over continuously ingested data for BI reports or ML models

- Challenge: migration costs may be amplified under hybrid workloads
  - ✓ Failed transactions may lead to huge restart costs for long-lived transactions
  - ✓ Analytic queries may lead to a lengthy downtime for suspend-and-resume (Albatross [VLDB '11]) and wait-and-remaster (DynaMast [ICDE' 20])
  - ✓ Interactive transactions make internal restarts for failed transactions impossible

Contributions

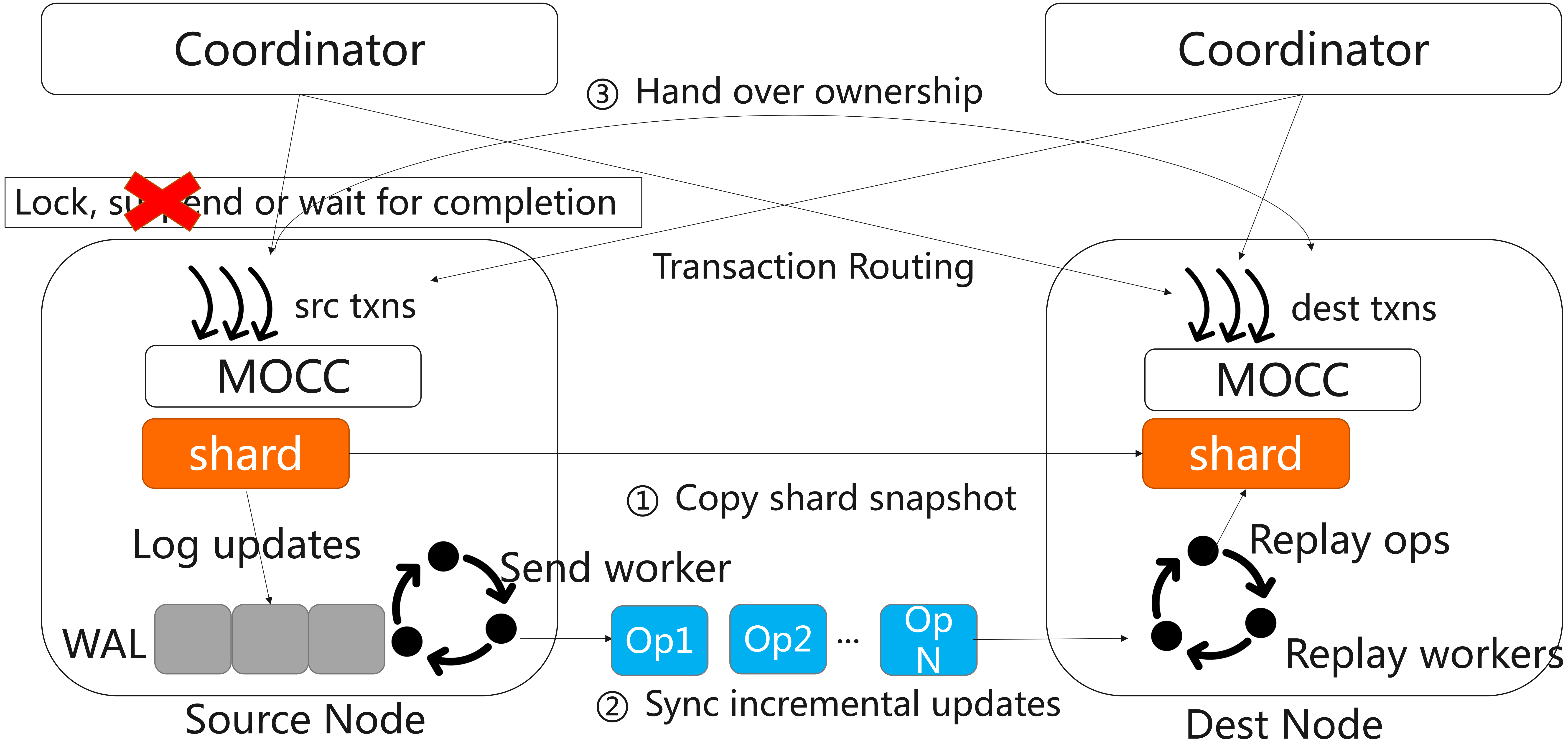
- Designed a live migration under SI (snapshot isolation) with zero service interruption and marginal performance degradation
- Implemented in PolarDB for PostgreSQL (distributed version)
- Evaluated state-of-art approaches under a broad spectrum of workloads

Target system (PolarDB for PG)



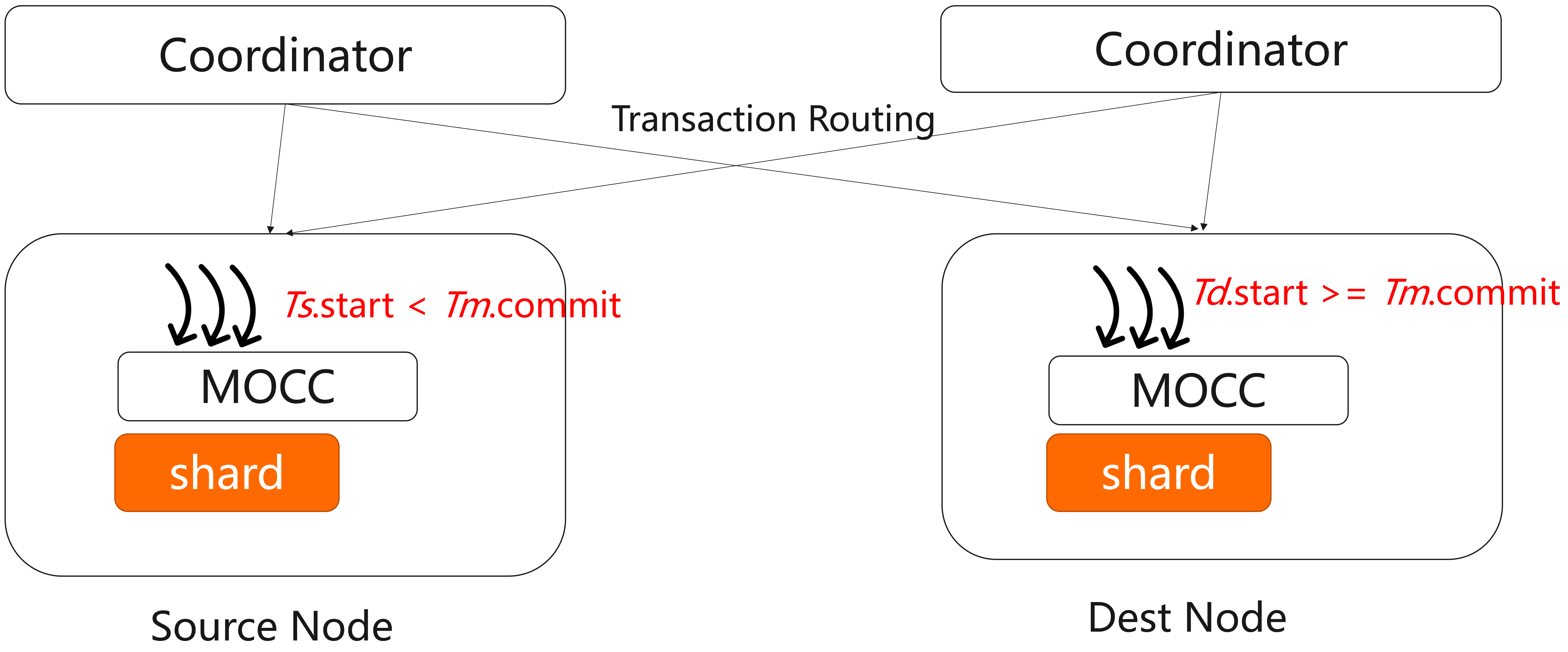
- Multi-coordinator architecture for scaling throughput
- Two-phase commit (2PC) for atomicity
- Distributed snapshot isolation
  - ✓ Timestamp ordering based MVCC
  - ✓ Global/Decentralized timestamp coordination

Overview



- Remove the lock-and-suspension step and avoid interruption or suspension
- Source transactions: active transactions on the soured node starting before hand-over
- Destination transaction: transactions starting on the destination node after hand-over
- Dual execution: utilizing ordered diversion and MOCC
  - ✓ allow both to run concurrently with consistency and snapshot isolation

Ordered diversion



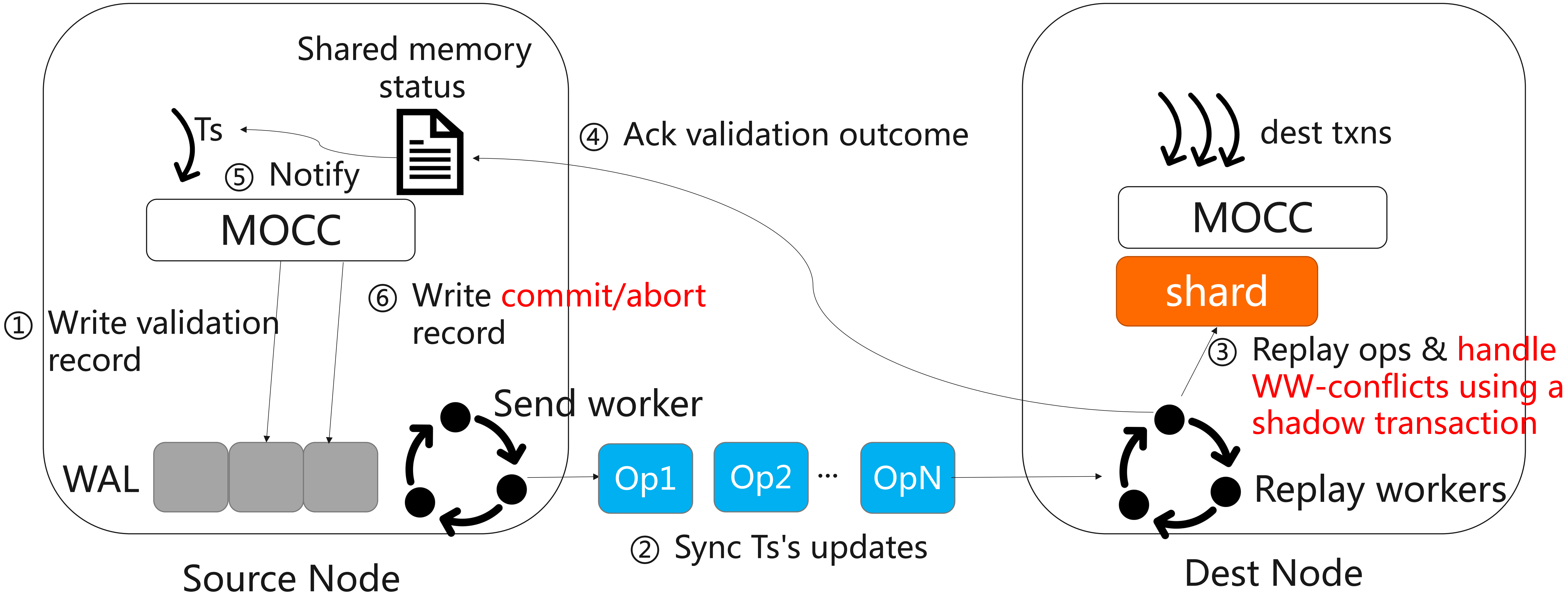
Ts: source transaction

Td: destination transaction

Tm: shard ownership hand-over transaction

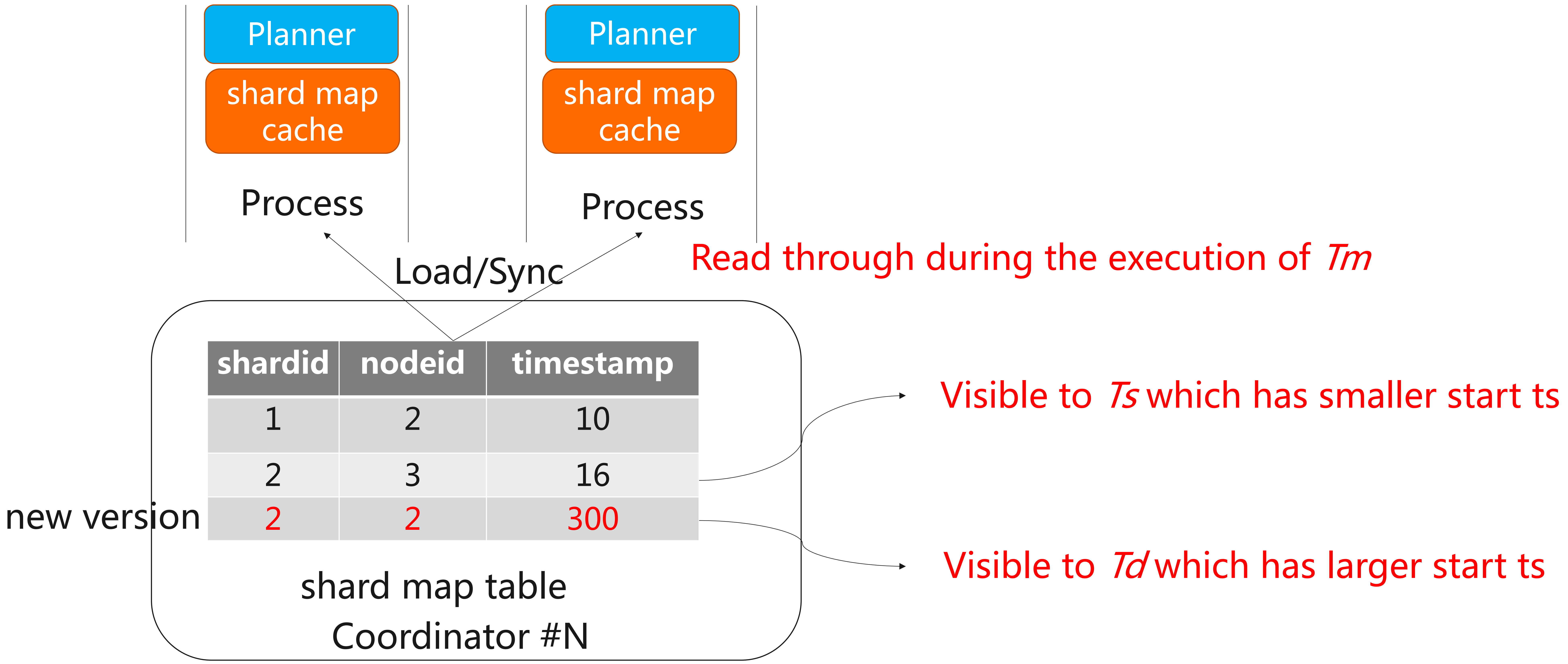
- Global timestamp ordering: *Td* starts after *Tm* commits
- *Td*.commit_timestamp > *Td*.start_timestamp >= *Tm*.commit_timestamp > *Ts*.start_timestamp
- *Td*'s updates are invisible to *Ts* under snapshot isolation (SI)
- Unidirectional synchronization: only updates of source transactions propagated to the destination
  - ✓ We minimize sync overhead
  - ✓ Only source transactions experience sync latency

Multi-version optimistic concurrency control



- Changing to sync propagation mode: source transaction cannot commit until its changes are propagated and validated on the destination
- Source and destination transactions follow MOCC:
  - ✓ local CC based on MVCC
  - ✓ cross-node CC based on OCC
- Distributed source transaction combines 2PC with MOCC's two stage commit

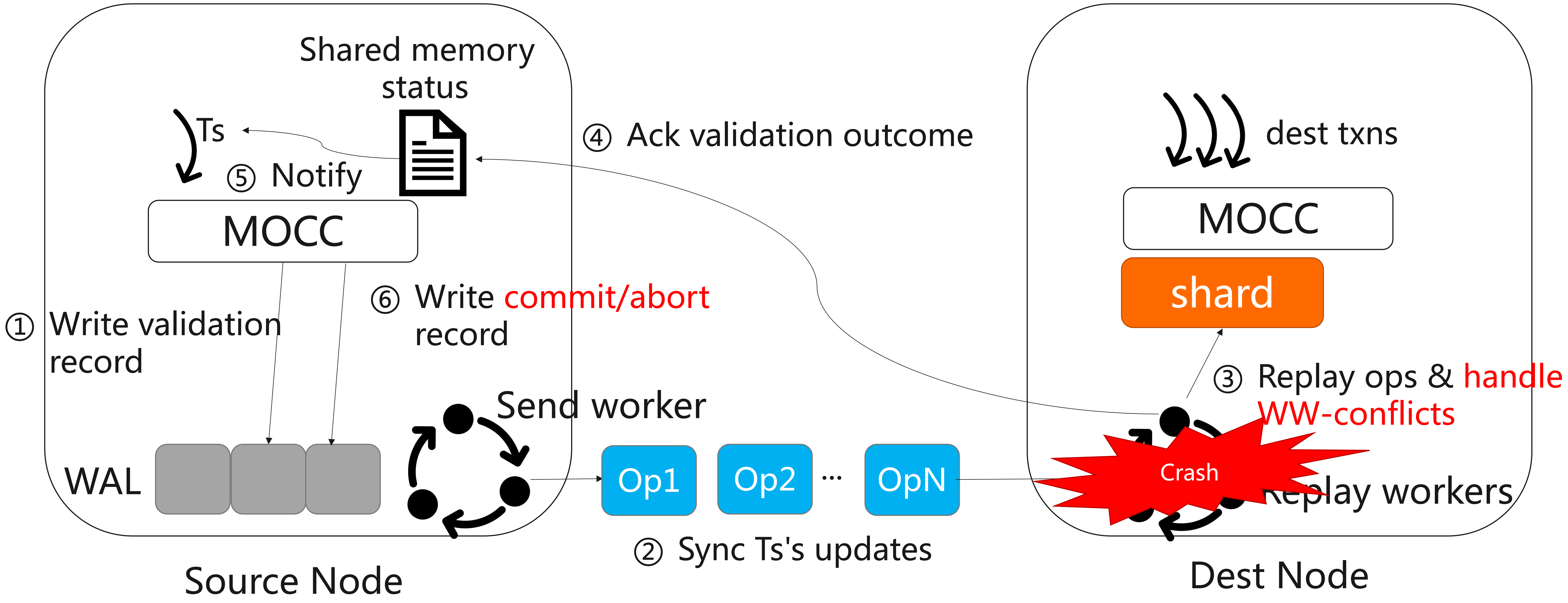
Consistency of shard map cache



shardid	nodeid	timestamp
1	2	10
2	3	16
2	2	300

- Retain transaction semantics between shard map cache and its MVCC table
  - ✓ Each process builds a shard map cache to speed up shard-location when routing transactions (*T1*)
  - ✓ Planner may read stale shard map entries from the cache even if *T1*'s start >= *Tm*.commit
  - ✓ We adopt a read-through strategy to make sure planner can see the appropriate version in cache

Crash recovery



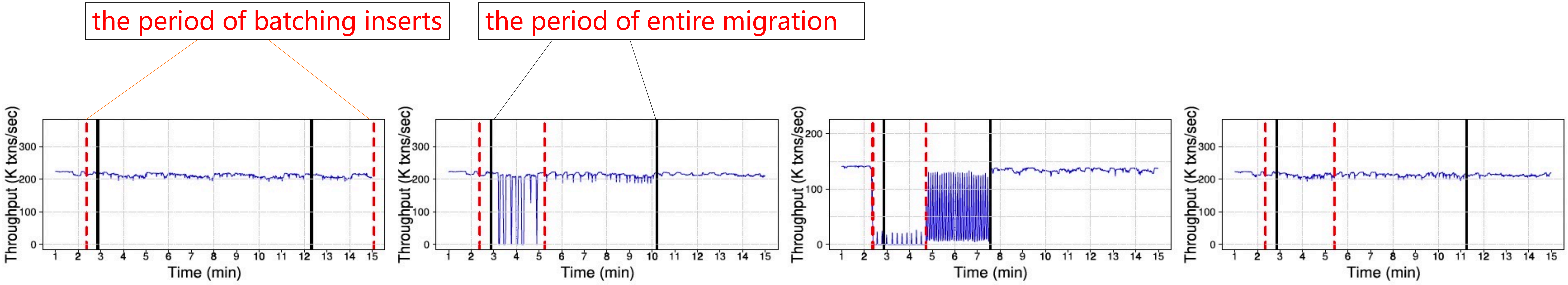
- Crash may happen on source, destination or both nodes during migration
- Check migration status to recover unfinished progress
  - ✓ If entering dual execution, check each pair of shadow and source transactions to complete unfinished transactions

Evaluation

阿里云

- Experiments were conducted on Alibaba Cloud using a 6-node database
- Workloads: TPC-C, YCSB and hybrid workloads
  - ✓ Hybrid workload A: a hybrid of batching inserts and YCSB
    - ➢ Simulate IoT and real-time analytics scenarios
  - ✓ Hybrid workload B: a hybrid of analytic queries and YCSB
    - ➢ Simulate HTAP scenarios
- Elasticity scenarios: cluster-consolidation, scale-out and load balance
- Compared baselines
  - ✓ Pull migration: Squall
  - ✓ Push migration: Lock-and-abort, wait-and-remaster

Cluster Consolidation under Hybrid workload A



(a) Lock-and-abort (b) Wait-and-remaster (c) Squall (d) Remus

YCSB throughput

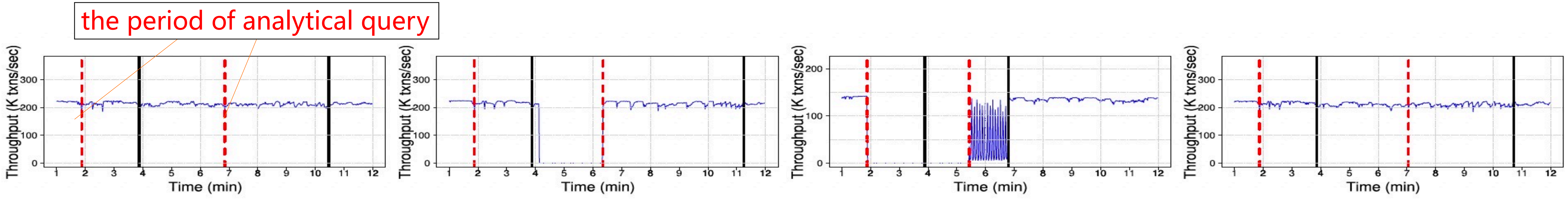
	Lock-and-abort	Wait-and-remaster	Squall	Remus
Abort Ratio During Consolidation	97%	0%	13%	0%
Avg. Throughput During/Before Consolidation	1.8/59	59/59	67/80	55/59

Table 2: The batch insert throughput (K tuples/s) under hybrid workload A (Ingested tuple size: 1KB).

- ✓ Due to failed transactions from migration, the throughput of batching insert for Lock-and-abort is only 1/30 of Remus during consolidation
- ✓ There are significant YCSB throughput fluctuations for Wait-and-remaster and Squall

Under Hybrid workload B & YCSB only

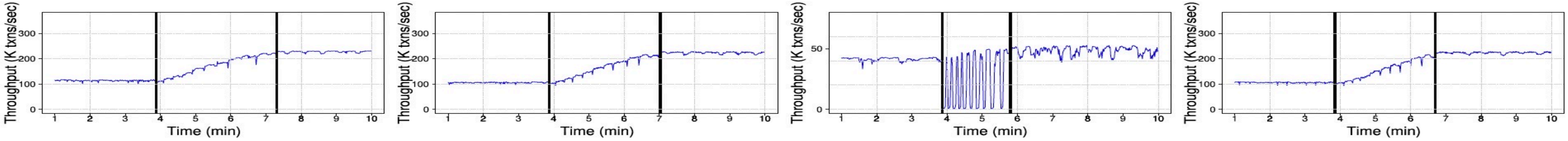
Cluster consolidation under Hybrid workload B



(a) Lock-and-abort (b) Wait-and-remaster (c) Squall (d) Remus

YCSB throughput

Load balance under YCSB only

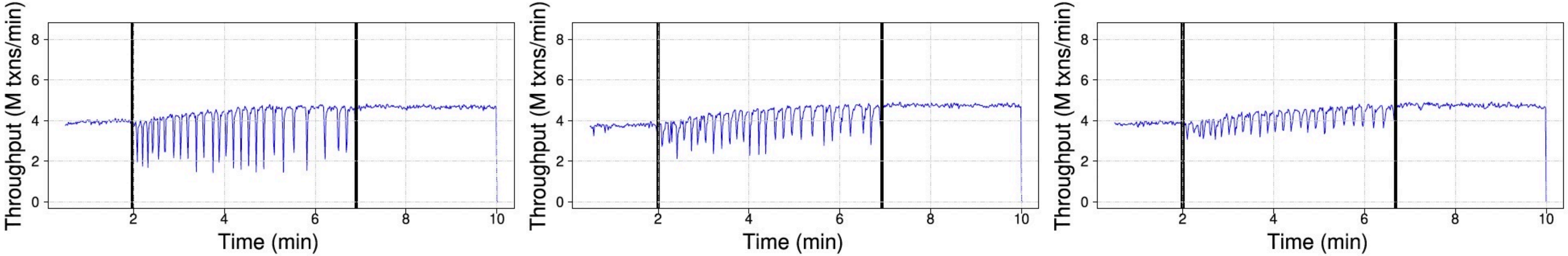


(a) Lock-and-abort (b) Wait-and-remaster (c) Squall (d) Remus

YCSB throughput

- ✓ significant YCSB throughput fluctuations for Squall
- ✓ The YCSB throughput of wait-and-remaster and Squall drops to zero during the execution of analytical query

Cluster Scale-out under TPC-C



(a) **Lock-and-abort** (b) **Wait-and-remaster** (c) **Remus**

TPC-C throughput

- ✓ Remus introduces much smaller throughput variation
- ✓ The lock downtime for ownership handover in lock-and-abort leads to significant throughput fluctuations

Latency increase compared to lock-and-abort

阿里云

Workload	*Remus*	*lock-and-abort*	Txn Latency
Hybrid A	1.9	27	2.1
Hybrid B	1.7	33	2.1
Load balancing	6.6	51	2.8
Scale-out	4.1	94	4-15

Avg. latency increase in ms

- ✓ The avg. latency increase in Remus is about an order of magnitude smaller than that in Lock-and-abort
- ✓ The latency increase in Lock-and-abort includes:
  - ➢ the time to lock the migrating shards and replay all remaining final updates
  - ➢ the time to update the shard map table across coordinators using 2PC

Conclusion

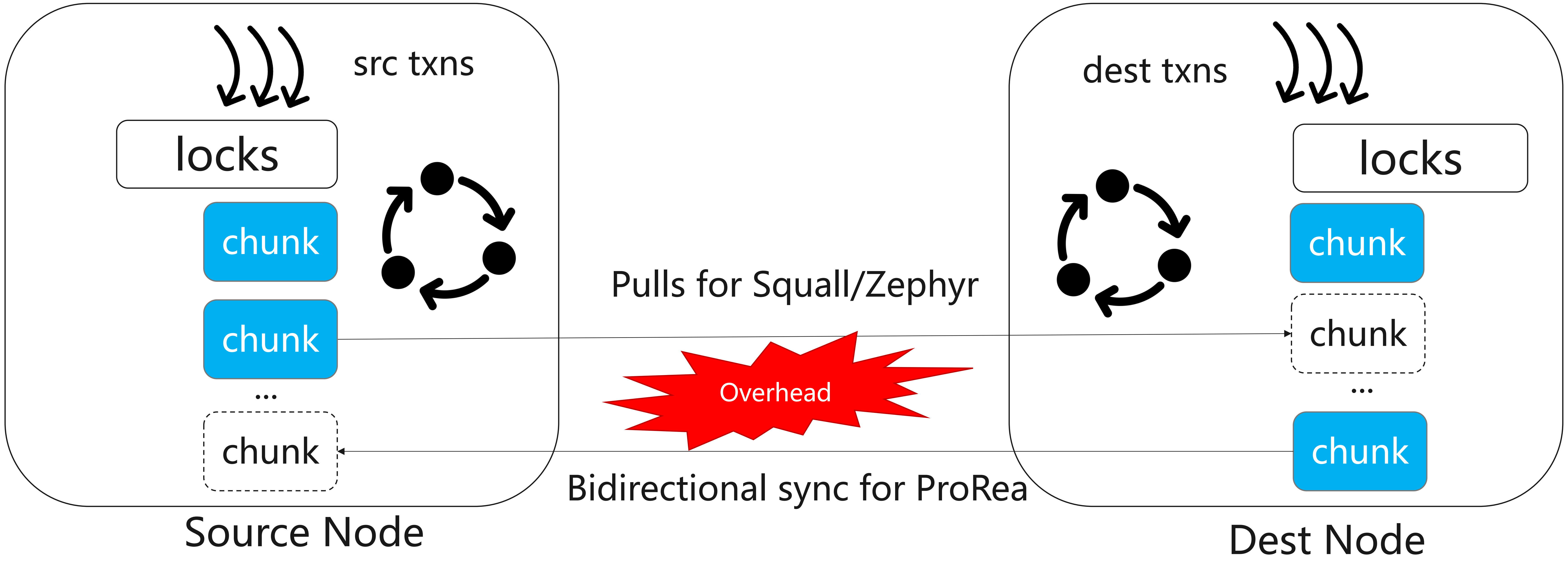
阿里云

- Compared to state-of-the-art approaches, Remus achieves following advantages under a wide variety of workloads:
  - ✓ zero transaction abort
  - ✓ zero downtime
  - ✓ marginal performance impact in terms of both throughput and latency

阿里云

Design challenge for dual execution

- Challenge for dual execution： How to maintain consistency at a low overhead
  - ✓ Squall adopts partition locking -> large overhead & failed txns
  - ✓ Zephyr uses frozen index + page locking to synchronize -> large overhead & failed txns
  - ✓ ProRea [EDBT '13] synchronizes pages between sites -> large overhead
  - ✓ MgCrab [VLDB '19] uses determinism to synchronize -> not general
- A good design should avoid the use of locking and bidirectional syncing



Ordered diversion

Migrate shard 2 from node 3 to node 2

Tm

Update shardmap set nodeid = 2 where shardid = 2

2PC

shardid	nodeid	timestamp
1	2	10
2	3	16
2	2	300

new version

shard map table

Coordinator #1

......

shardid	nodeid	timestamp
1	2	10
2	3	16
2	2	300

shard map table

Coordinator #N

- Adopt multi-versioning shard map table + timestamp ordering protocols to achieve this
  - ✓ Planner uses running transaction's start timestamp to read shard map entries for routing
  - ✓ We use a distributed transaction *Tm* to update shard map table across coordinators
  - ✓ Existing timestamp ordering protocols (e.g., Google Percolator [OSDI '10]) can be leveraged to guarantee routing consistency among multiple coordinators