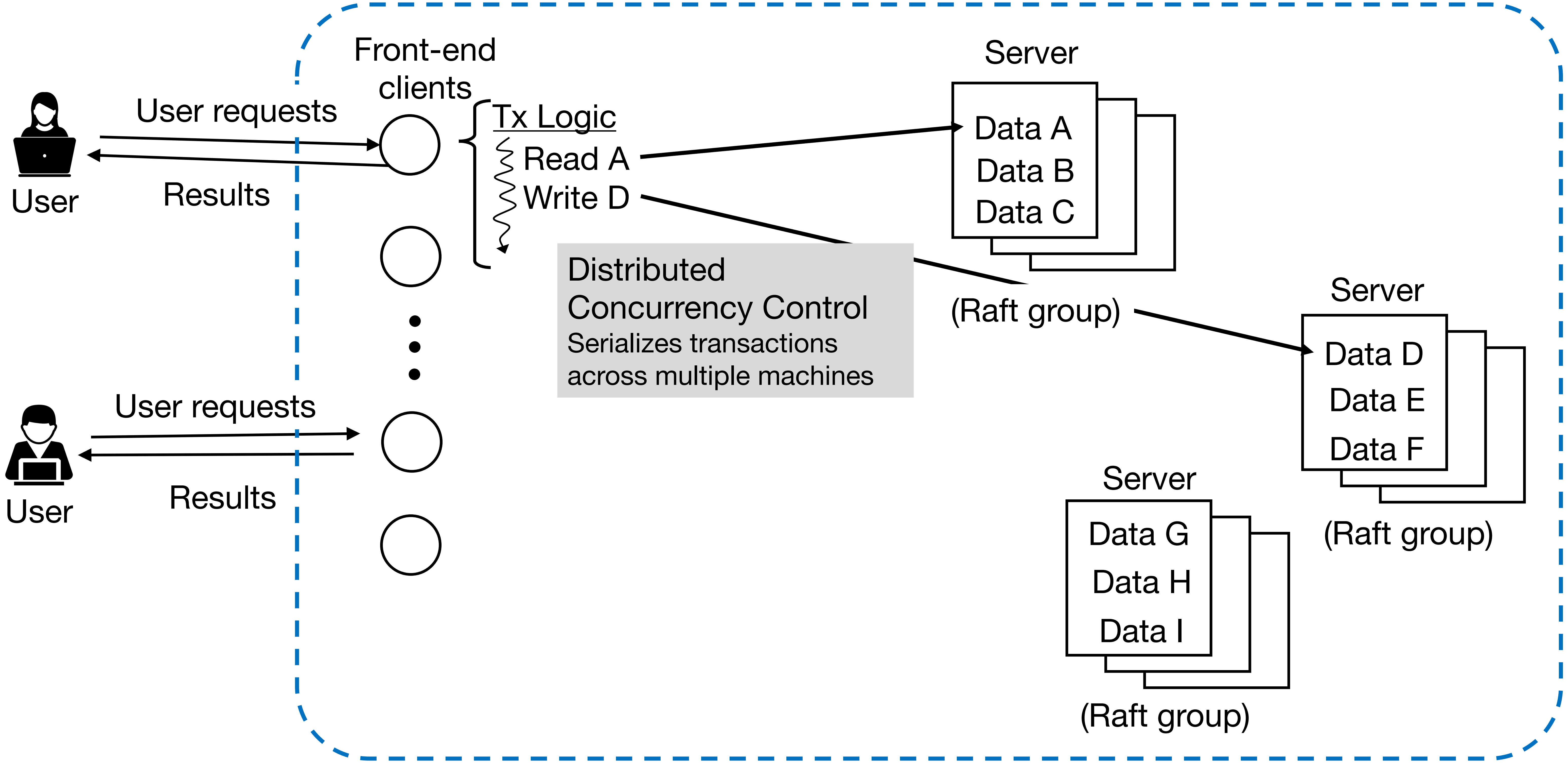
Accelerating Skewed Workloads With Performance Multipliers in the TurboDB Distributed Database

Jennifer Lam[a]

Jeffrey Helt[a] Wyatt Lloyd[a] Haonan Lu[b]

[a]Princeton University [b]University at Buffalo

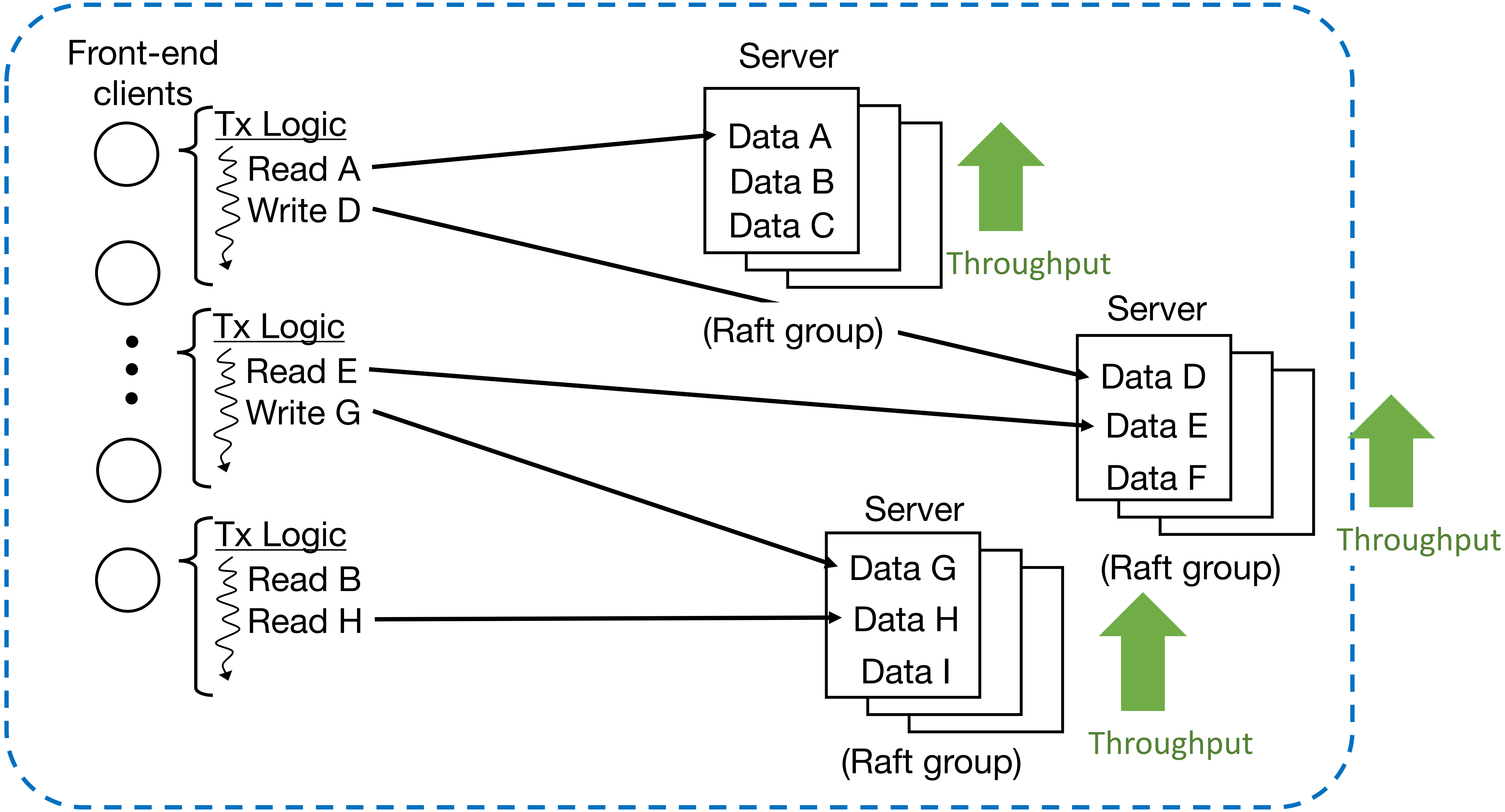
Distributed Databases Overview



Distributed Databases Enables Large Scale Applications

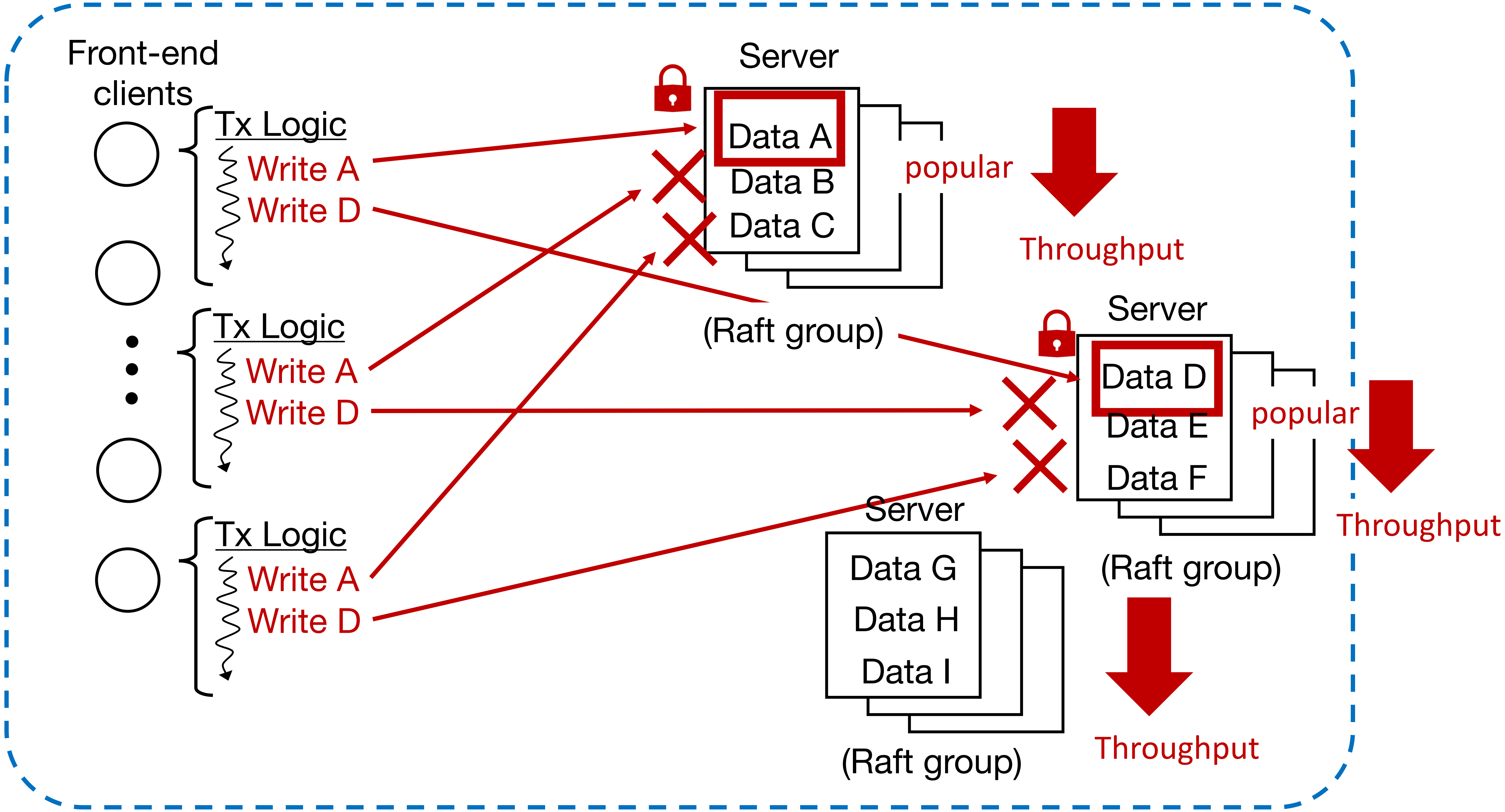
Scale capacity!

Scale throughput for **uniform** workload!



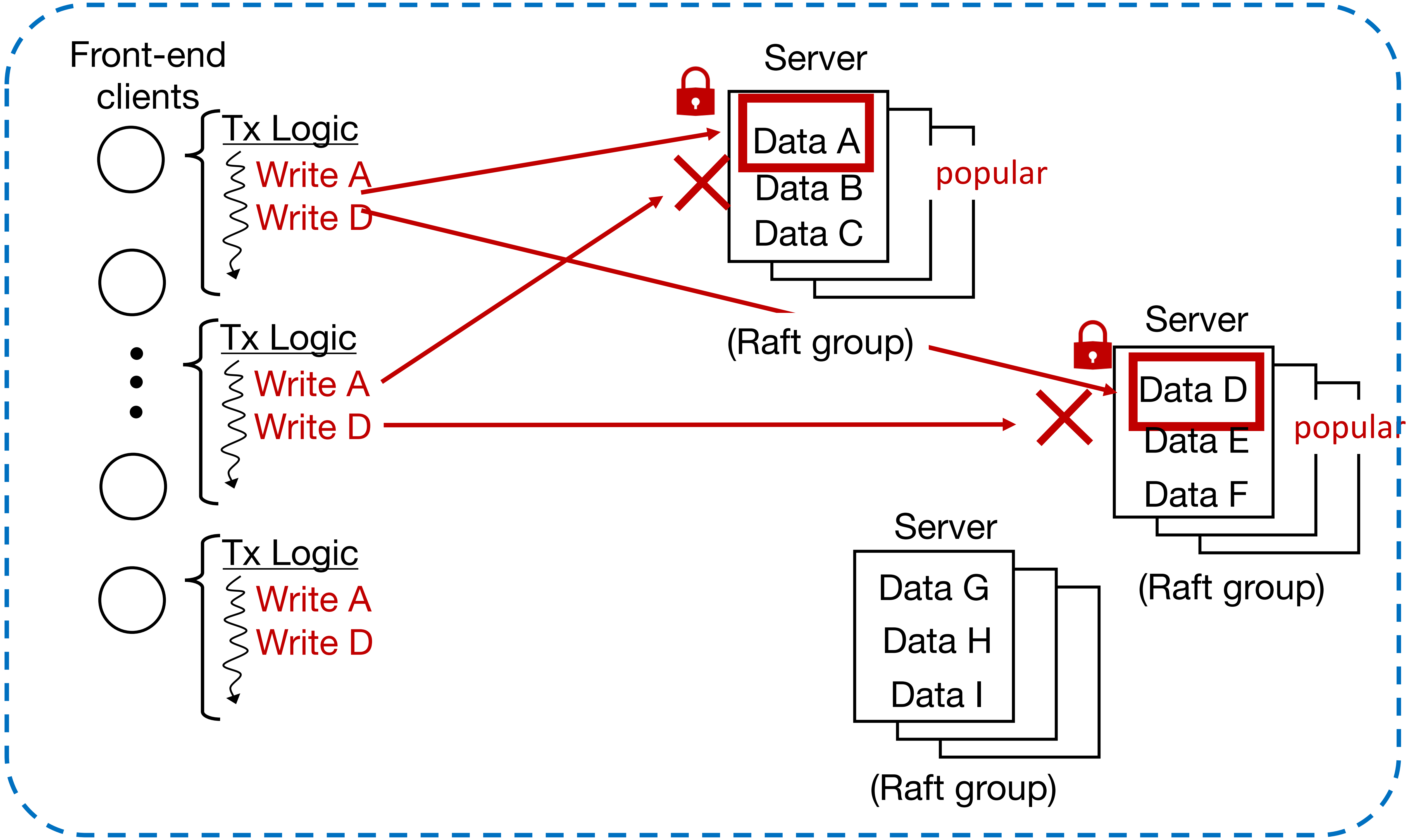
Distributed Databases Challenged by Skewed Workloads

Contention leads to excessive aborts and retries that **degrade system performance**.



Data Sharding Exacerbates Skew's Negative Impact

Culprit: cross-node coordination. Transactions incur network latency.

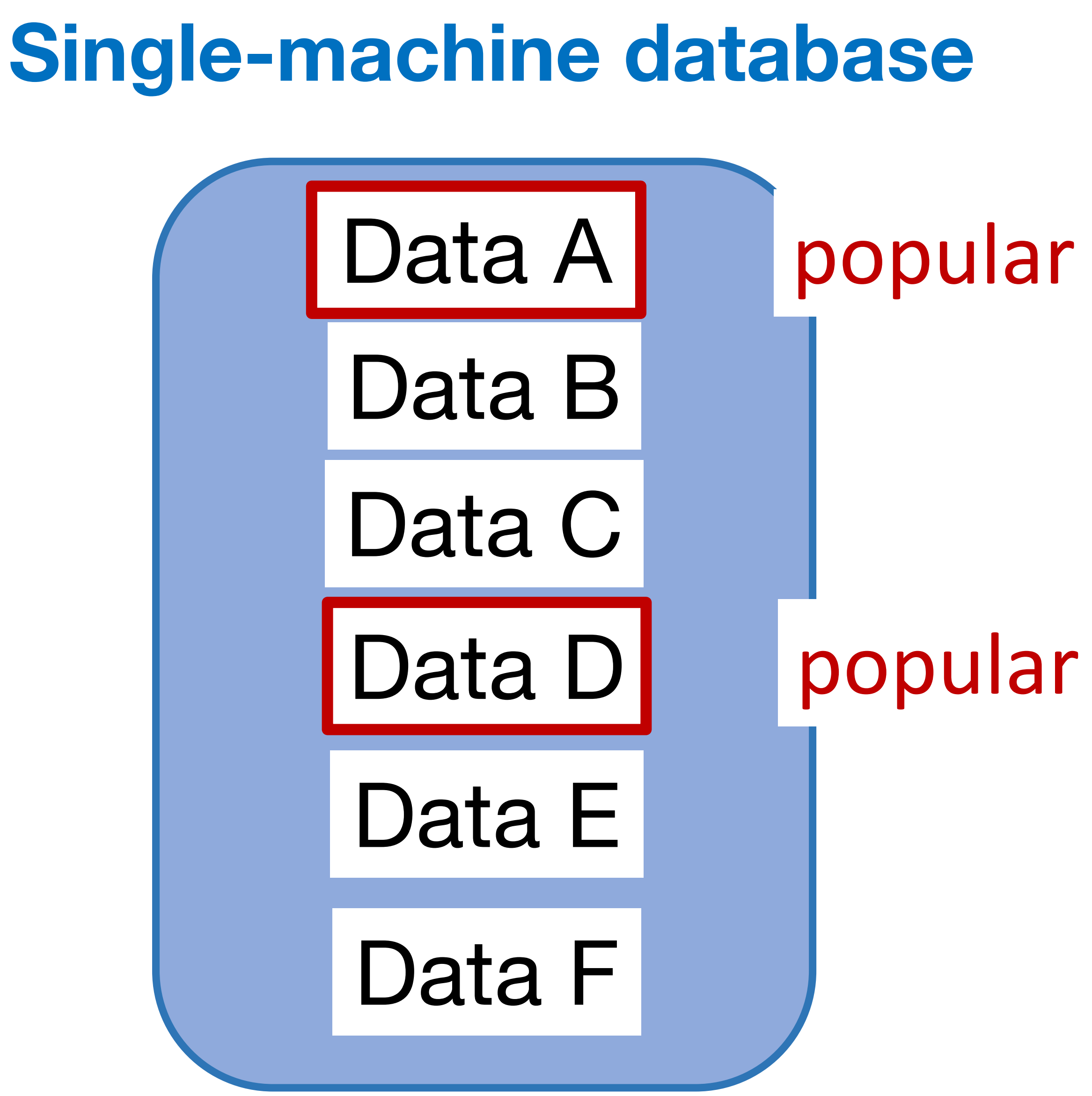


Longer latencies than local transactions → longer transaction lifetimes → likely to conflict.

Single-Machine Databases Better Handle Skew

Single machine databases: centralize data on one server.

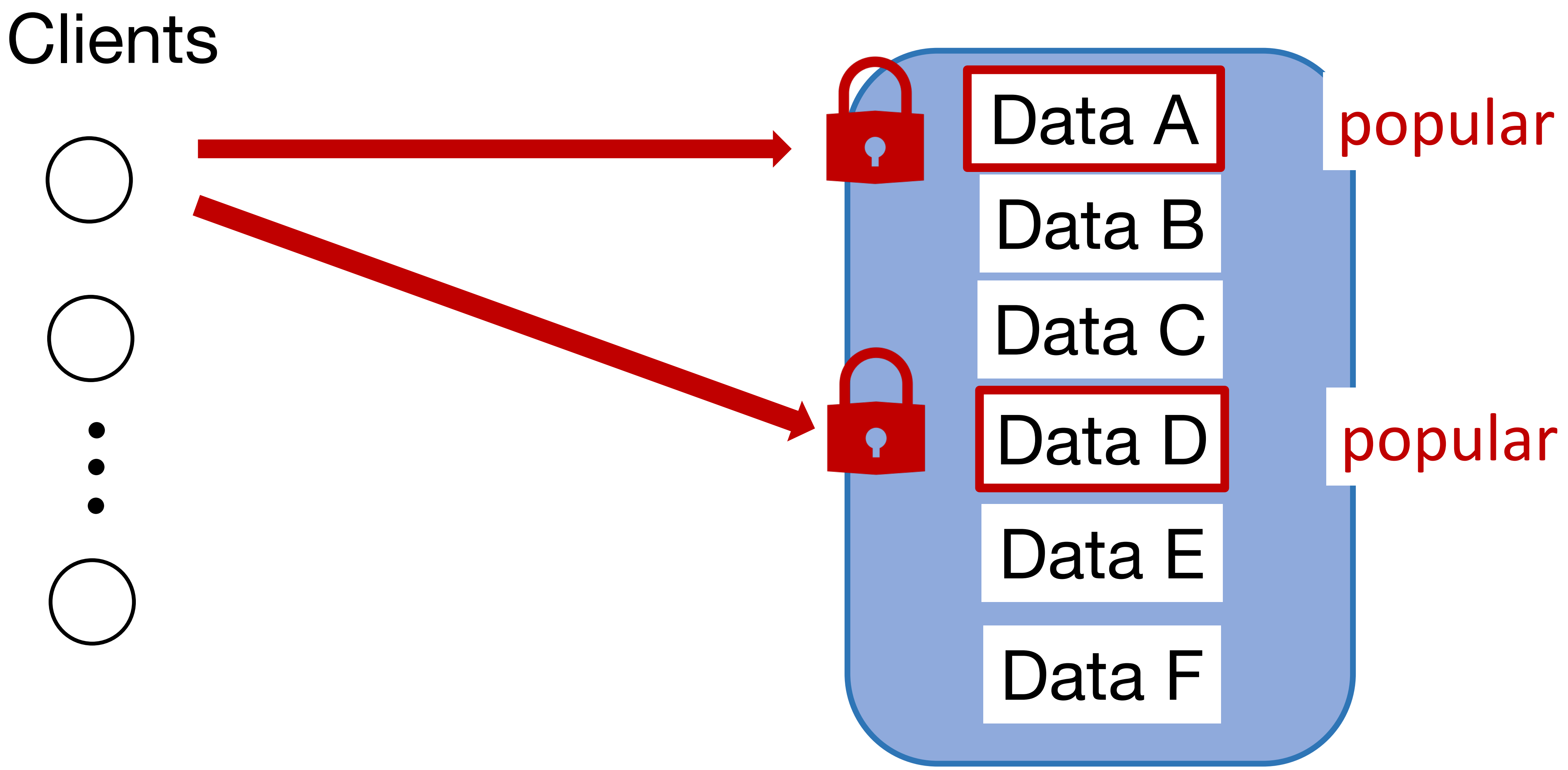
- No cross-node coordination → transactions do not incur network latency.



Single-Machine Databases Leverage Performance Multipliers

Performance multipliers: properties of single-machine databases that benefit its performance.

- **Does not apply to distributed databases.**

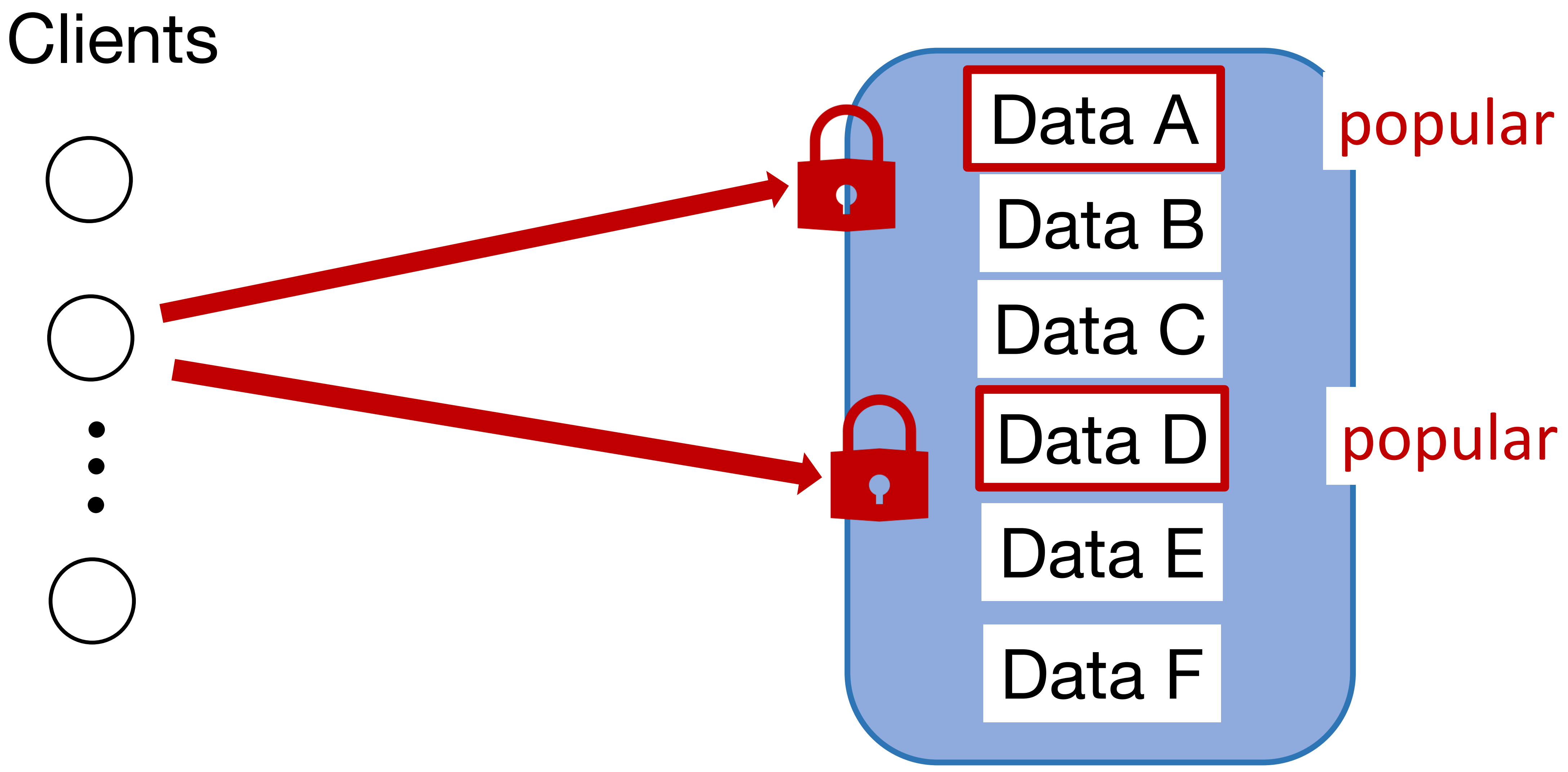


One-stop execution: all transaction requests **only locally** access database.

Single-Machine Databases Leverage Performance Multipliers

Performance multipliers: properties of single-machine databases that benefit its performance.

- **Does not apply to distributed databases.**

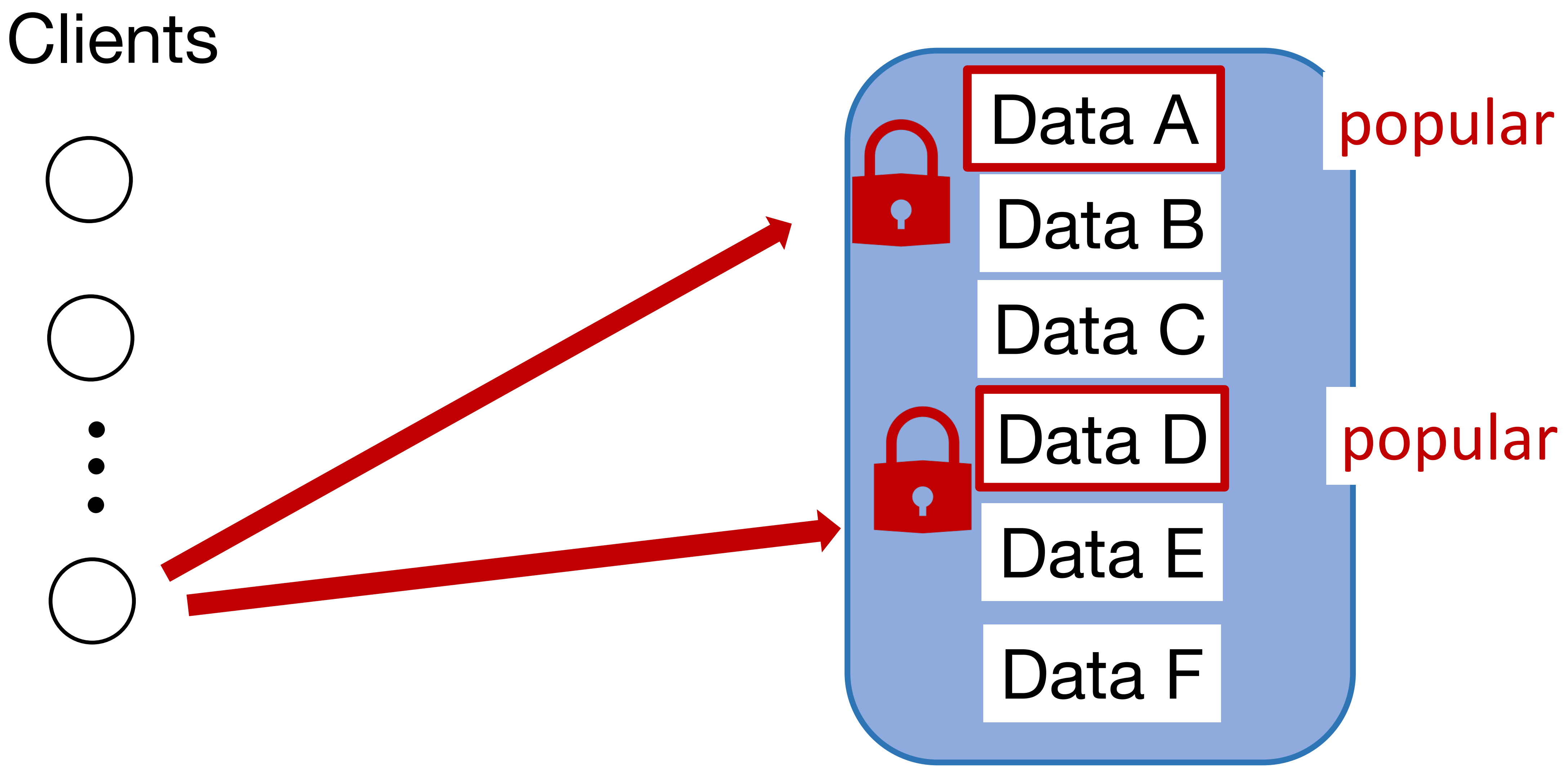


One-stop execution: all transaction requests **only locally** access database.

Single-Machine Databases Leverage Performance Multipliers

Performance multipliers: properties of single-machine databases that benefit its performance.

- **Does not apply to distributed databases.**



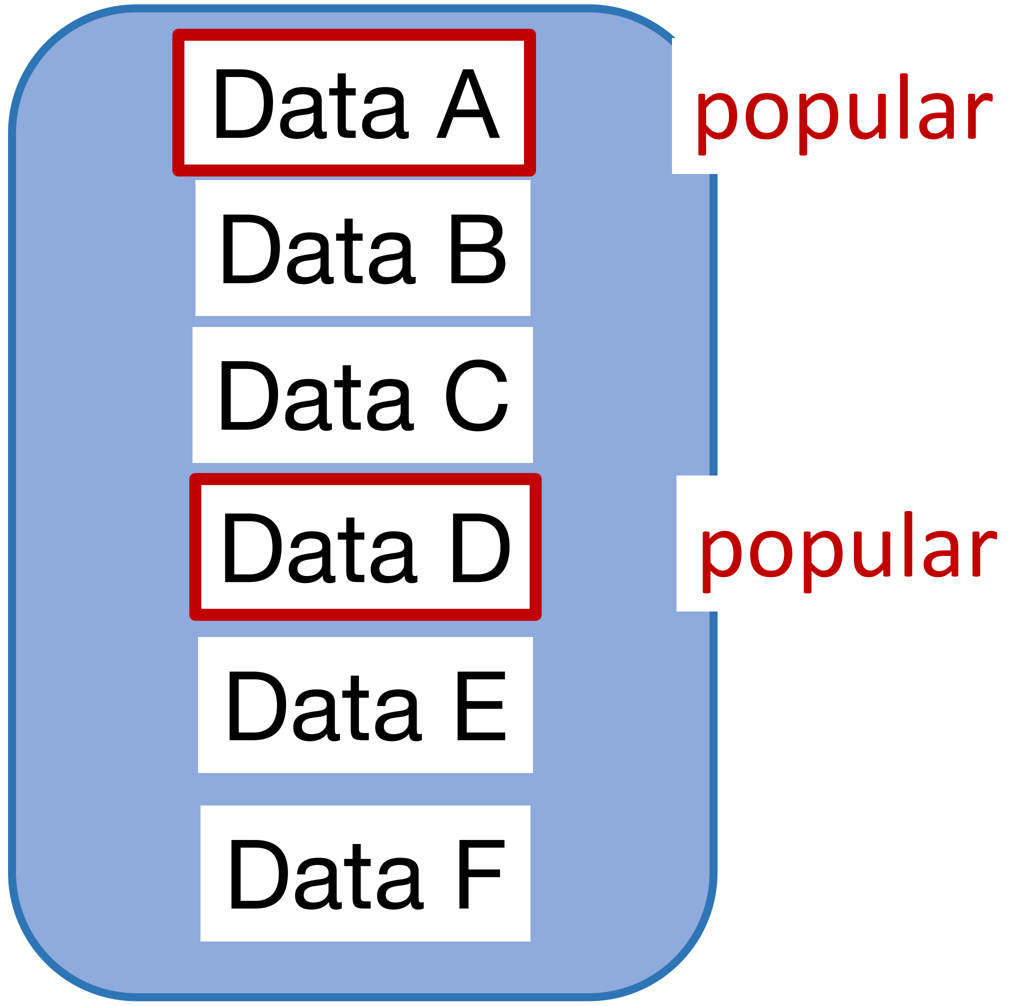
One-stop execution: all transaction requests **only locally** access database.

Single-Machine Databases Leverage Performance Multipliers

Performance multipliers: properties of single-machine databases that benefit its performance.

- **Does not apply distributed databases.**

✓ Global database-wide techniques, e.g. Silo [OSDI '14].
✓ Targeting local bottlenecks, e.g. MVTL [PODC '18].



One-stop execution: all transaction requests **only locally** access database.

Local concurrency control techniques: performance optimizations exploit data being central to one machine.

Distributed vs. Single-Machine Databases

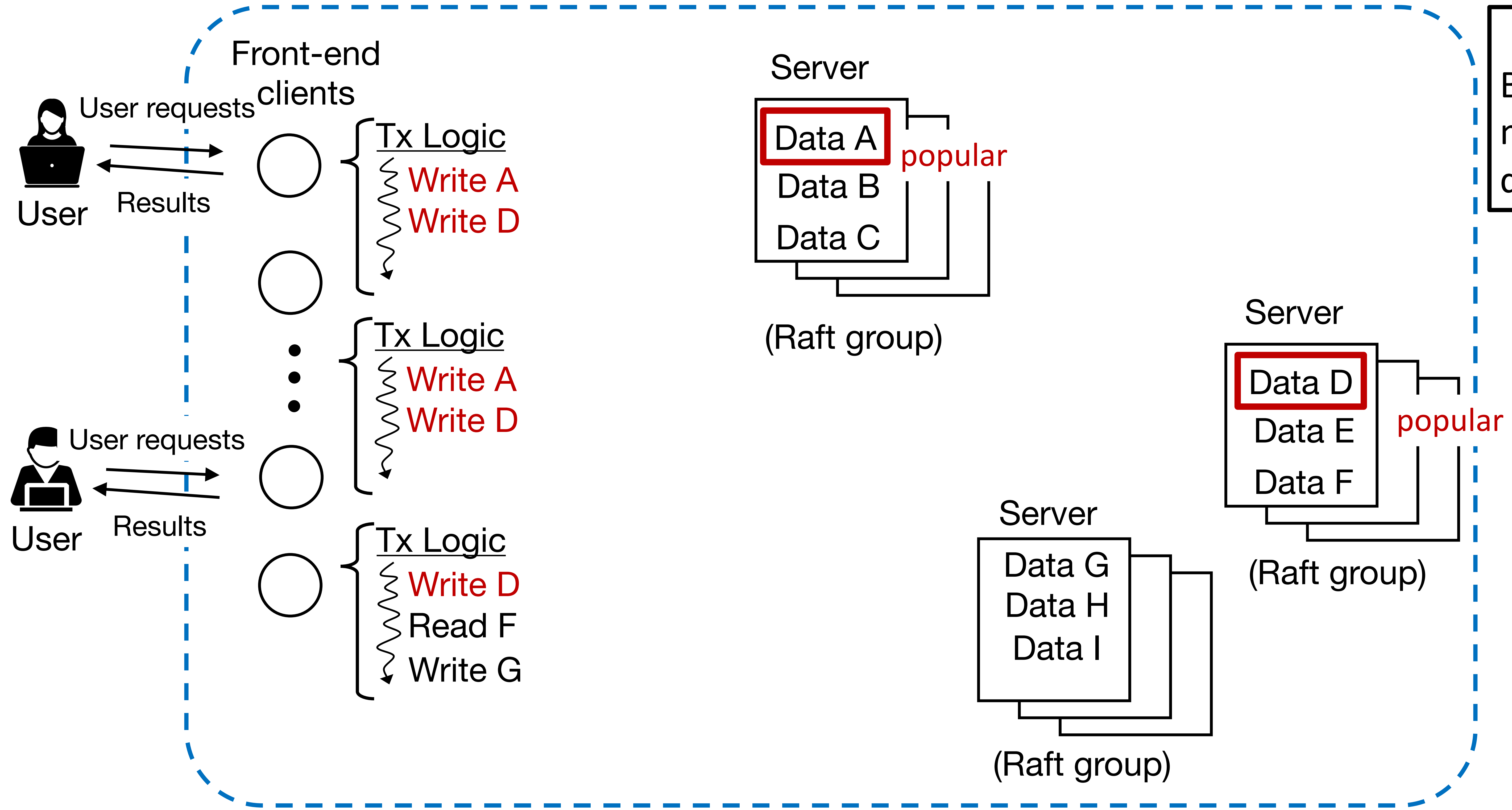
	Scales capacity	Handles skewed workloads
Distributed databases	✓	✗
Single-machine databases	✗	✓
Ideal	✓	✓

TurboDB

A new hybrid architecture that integrates
a single-machine database into a distributed database.

The distributed database scales storage capacity while
the single-machine database "turbocharges" performance
with its performance multipliers.

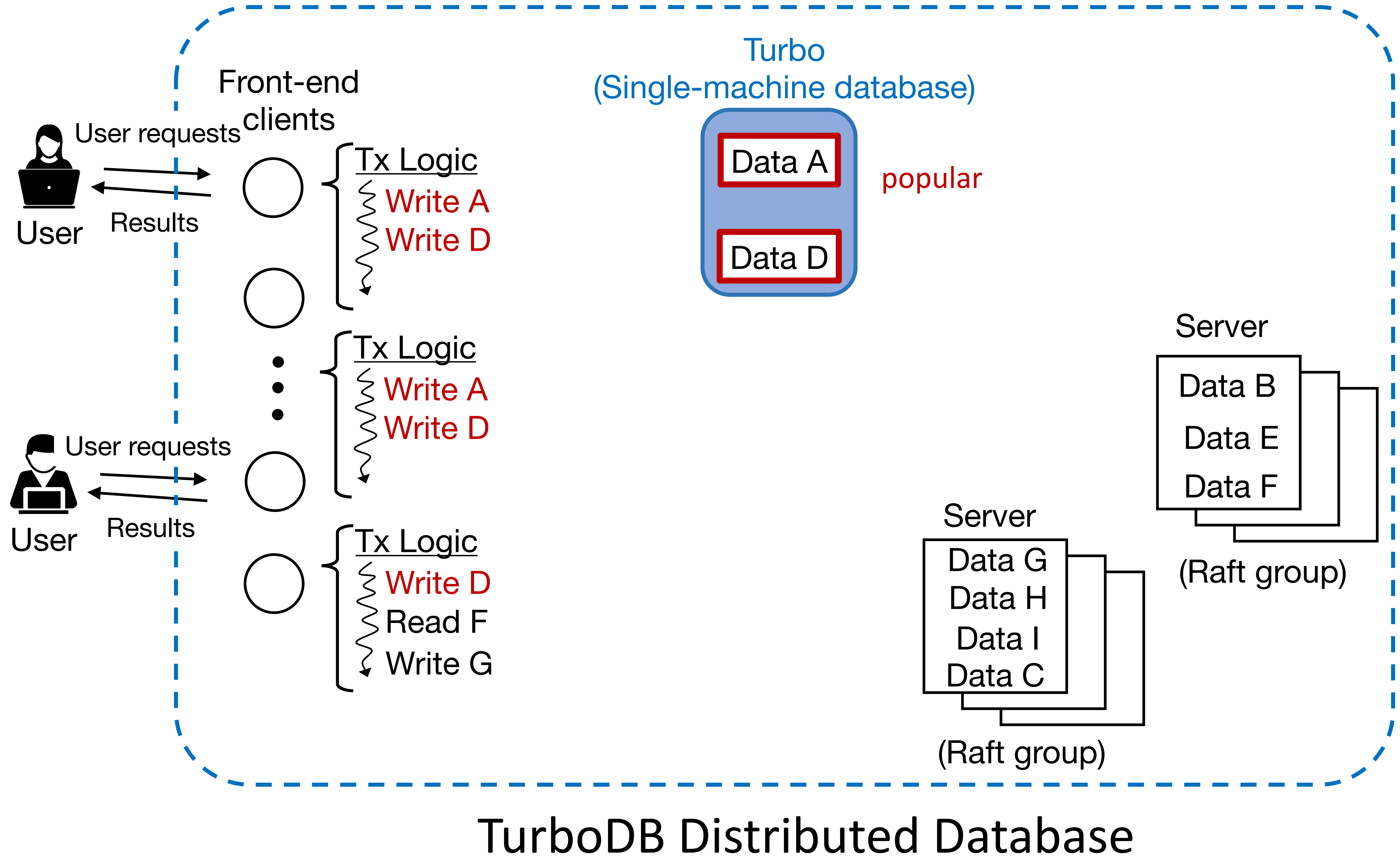
TurboDB's Hybrid Architecture



TurboDB Distributed Database

Core challenge
Bringing the performance multipliers to bear on skewed, distributed workloads.

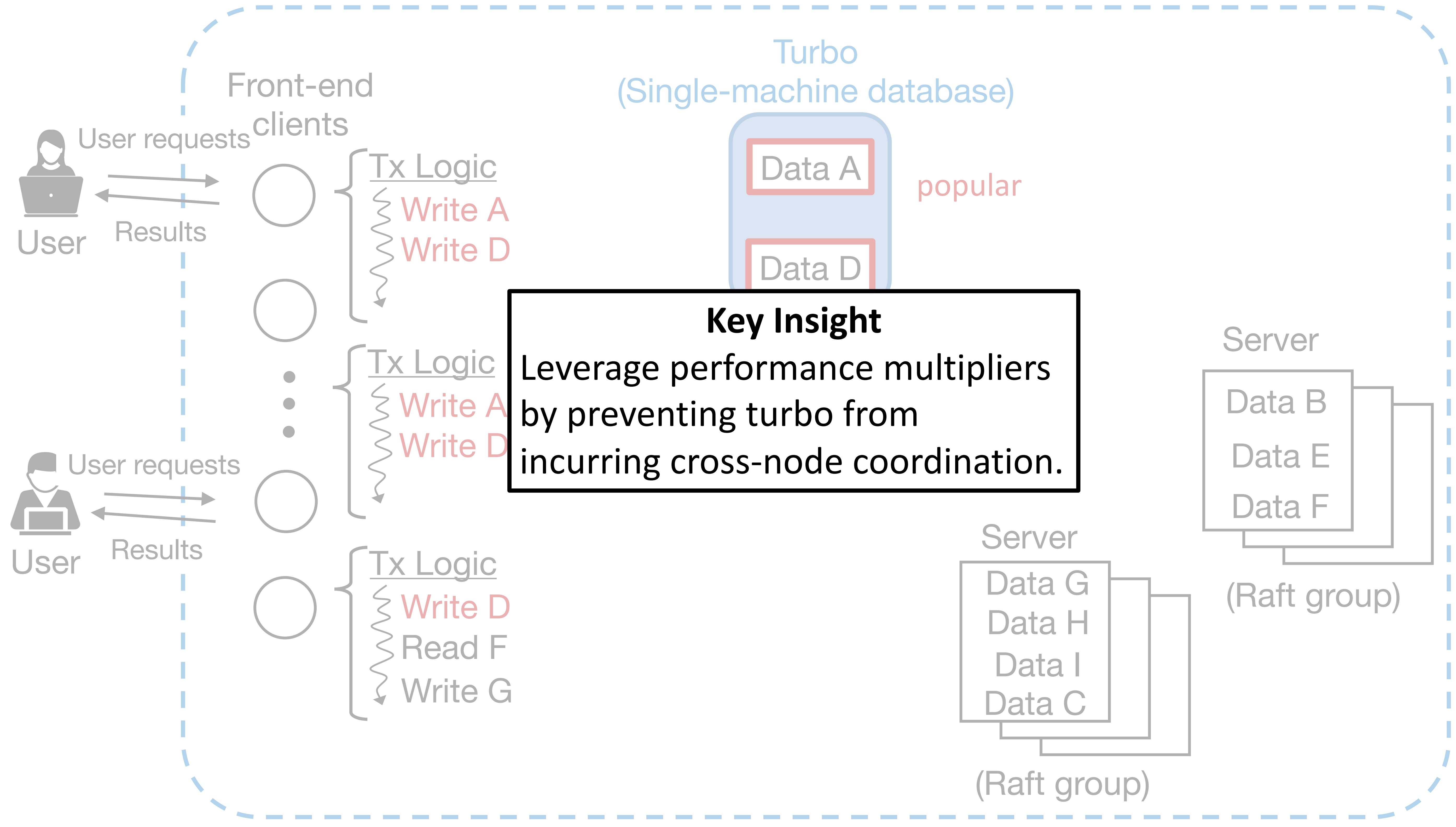
TurboDB's Hybrid Architecture



Core challenge

Bringing the performance multipliers to bear on skewed, distributed workloads.

Challenges Unique to TurboDB

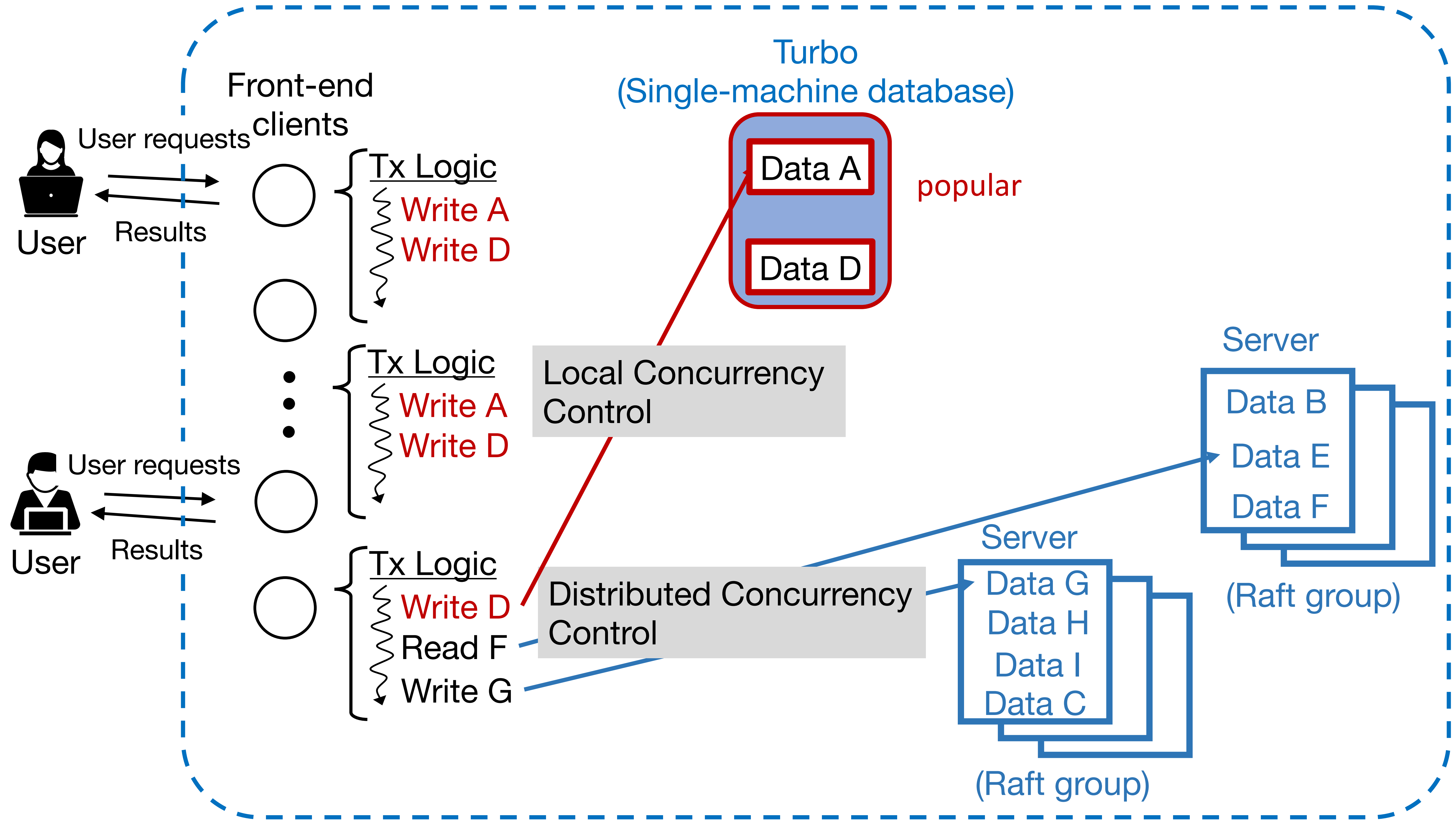


TurboDB Distributed Database

Key Insight
Leverage performance multipliers by preventing turbo from incurring cross-node coordination.

Core challenge
Bringing the performance multipliers to bear on skewed, distributed workloads.

Challenges Unique to TurboDB



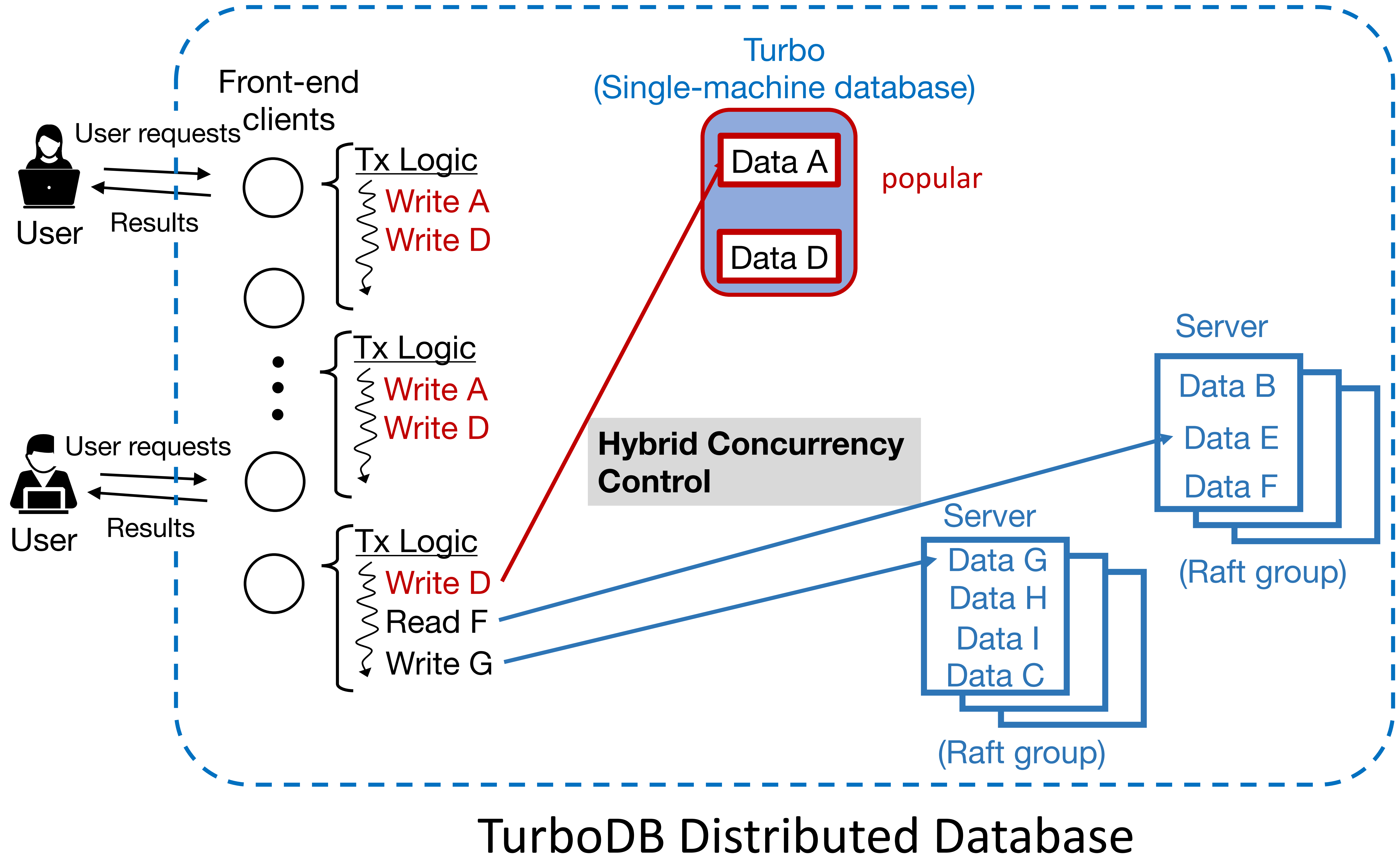
TurboDB Distributed Database

Core challenge

Leverage performance multipliers on distributed databases.

#1: Two concurrency controls.

Challenges Unique to TurboDB

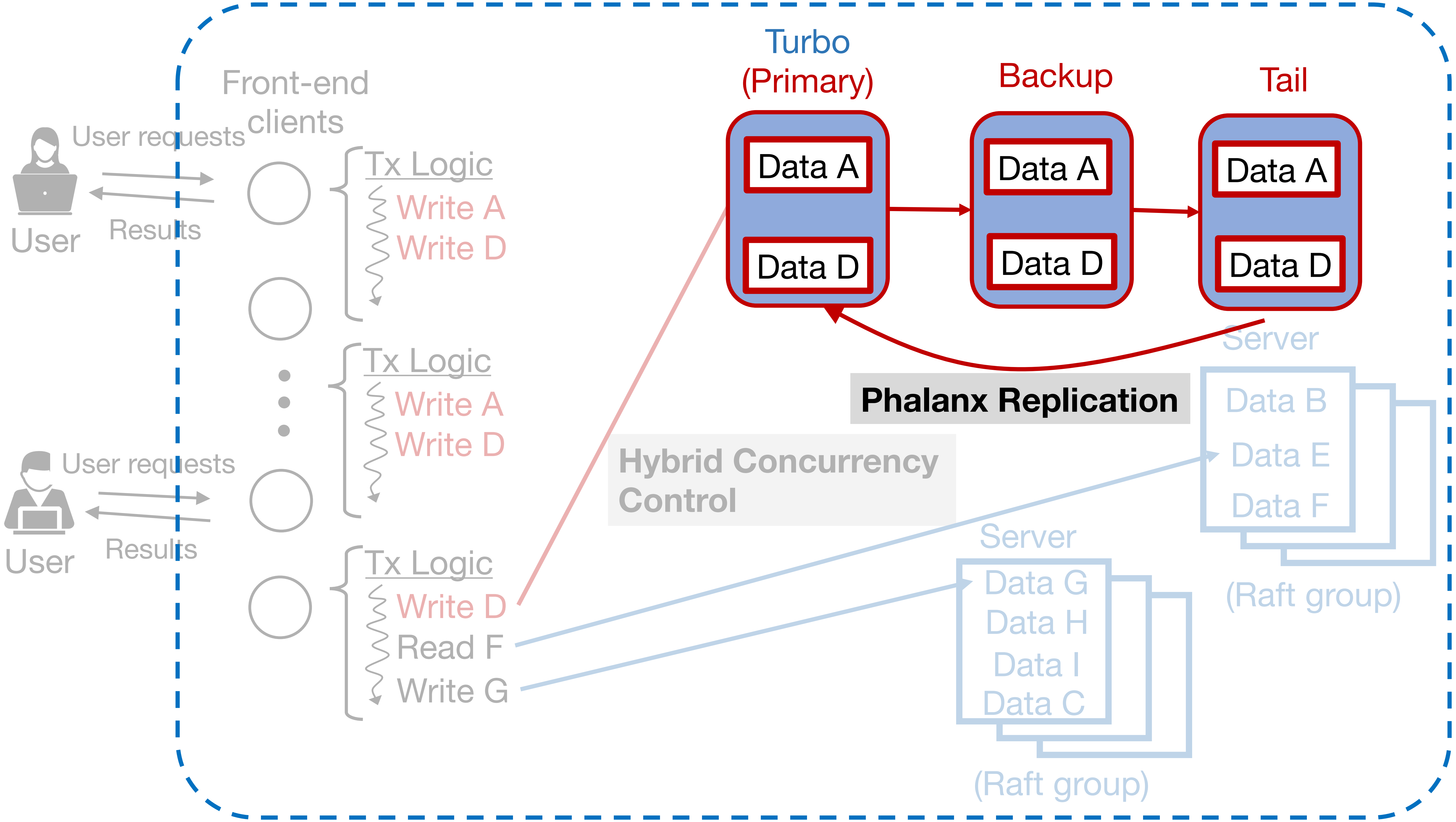


Core challenge

Leverage performance multipliers on distributed databases.

#1: Two concurrency controls.
Hybrid Concurrency Control (HCC)

Challenges Unique to TurboDB



TurboDB Distributed Database

Core challenge

Leverage performance multipliers on distributed databases.

#1: Two concurrency controls.
Hybrid Concurrency Control (HCC)

#2: Turbo fault tolerance.
Phalanx Replication

Hybrid Concurrency Control (HCC) Intuition

HCC Goal. Orchestrate two independent database's concurrency control protocols:

(Correctness) Both can totally order transactions in the same serial schedule.

(Performance) Turbo can execute its part **without cross-node coordination**.

Correctness: HCC Enforces Process-Order Serializability

Process-ordered serializability = total order + process order.

Total order: txns are **assigned a single timestamp** across both local and distributed concurrency control protocols.

- Both protocols serialize txns by timestamp, so both converge on the same total order.

Process order: to respect process order, clients generate timestamps.

Performance: HCC Runs Turbo as a Standalone Database

Performance Goal: prevent turbo from incurring cross-node coordination.

Solution: Run turbo as a standalone, single-machine database **unaware of its role in hybrid architecture**.

Abstract away txns as turbo's "clients."

→

Txns encapsulate turbo requests as **isolated, single-machine database transactions**.

→

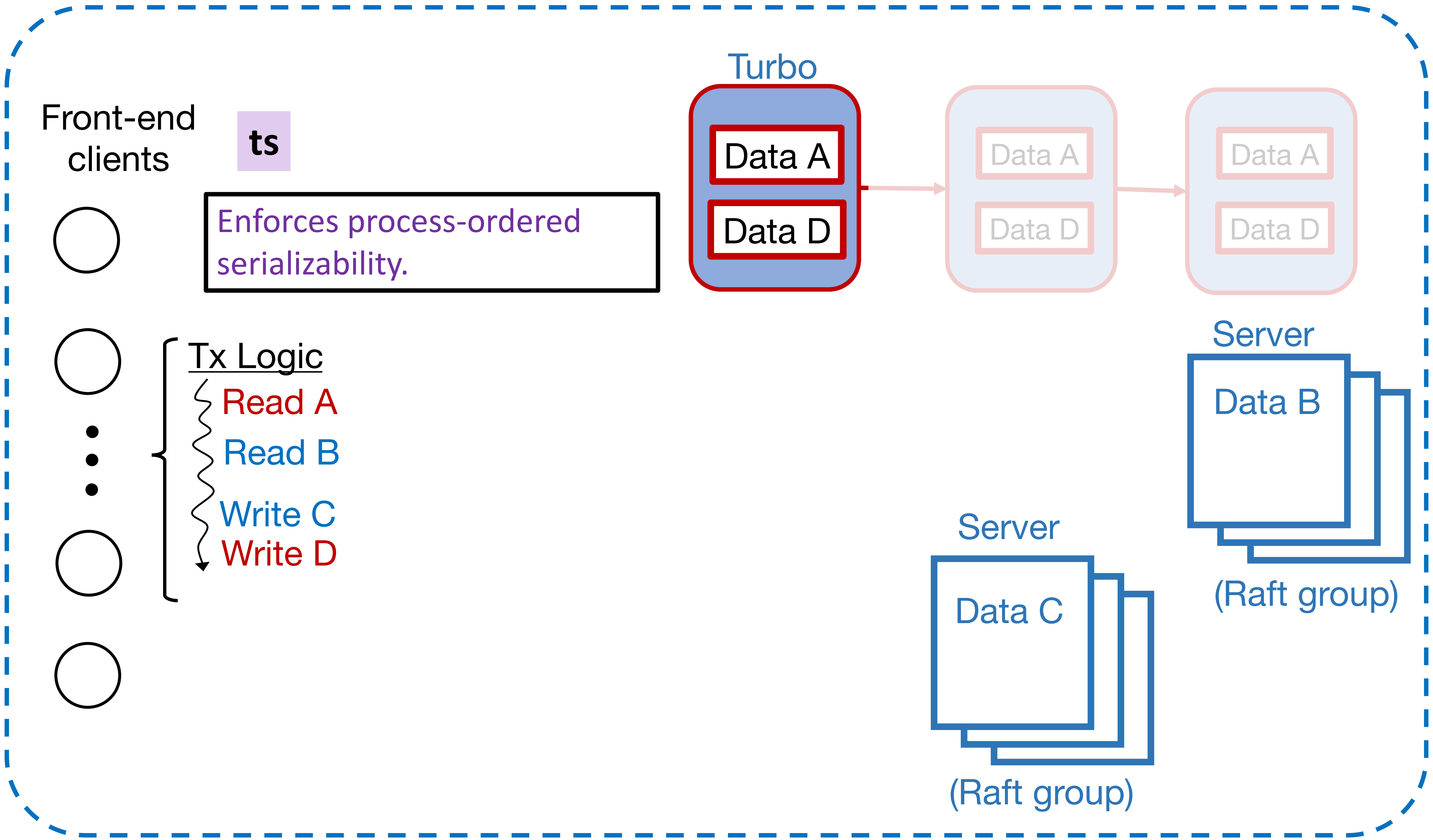
Transactional Atomicity Issue: The turbo can't match up all requests from the same txn.

Key Insight #1: Each distributed transaction limited to sending the turbo…

- **One read-write single-machine database txn** of update operations.
- **Multiple read-only single-machine database txns** of read operations.

Key Insight #2: When turbo commits (aborts) isolated read-write txn, servers mirror decision.

HCC Supports General Transactions



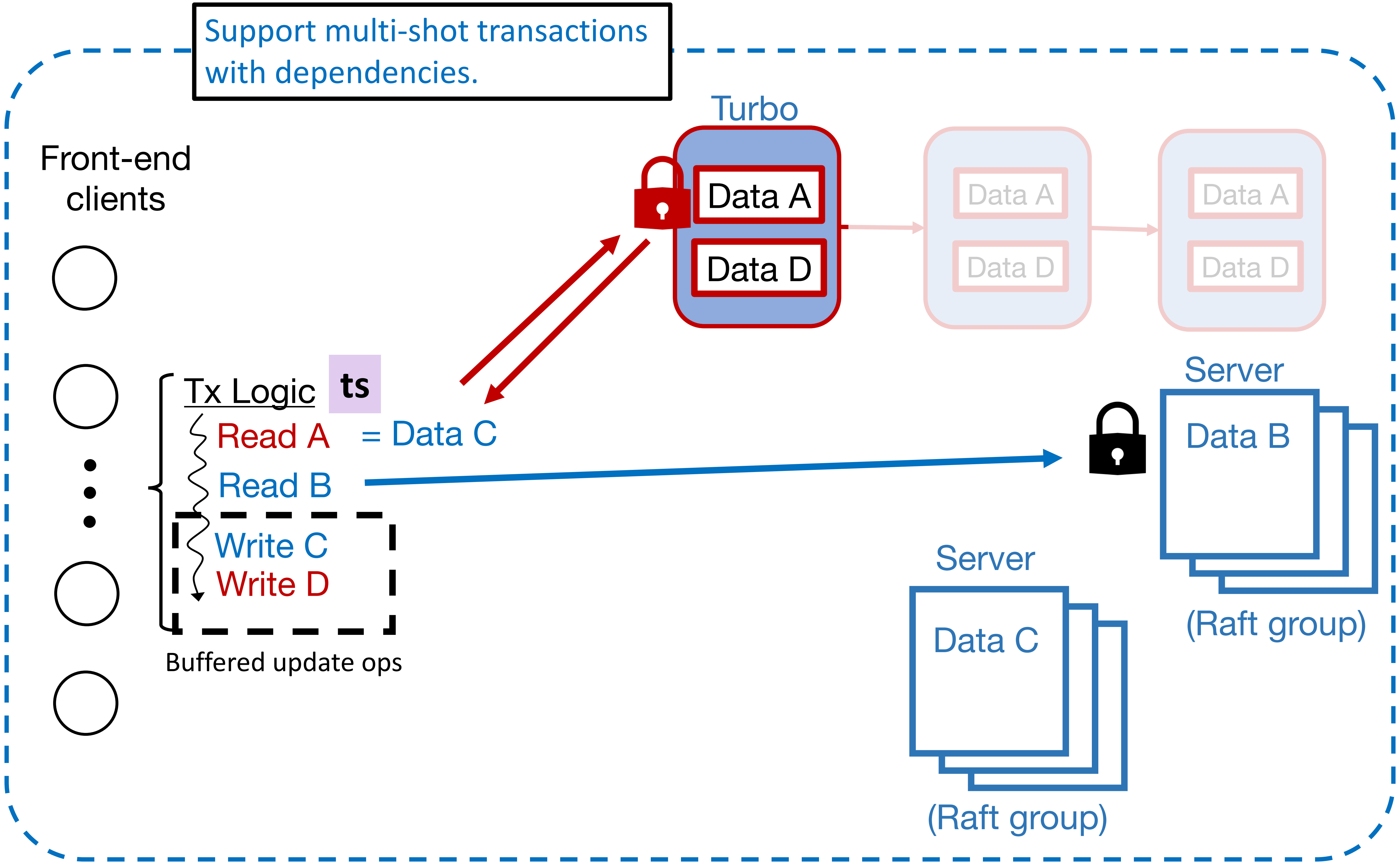
HCC Protocol

Execute Phase.

1. Client assigns ***timestamp ts***.

Finale-Commit Phase

HCC Supports General Transactions



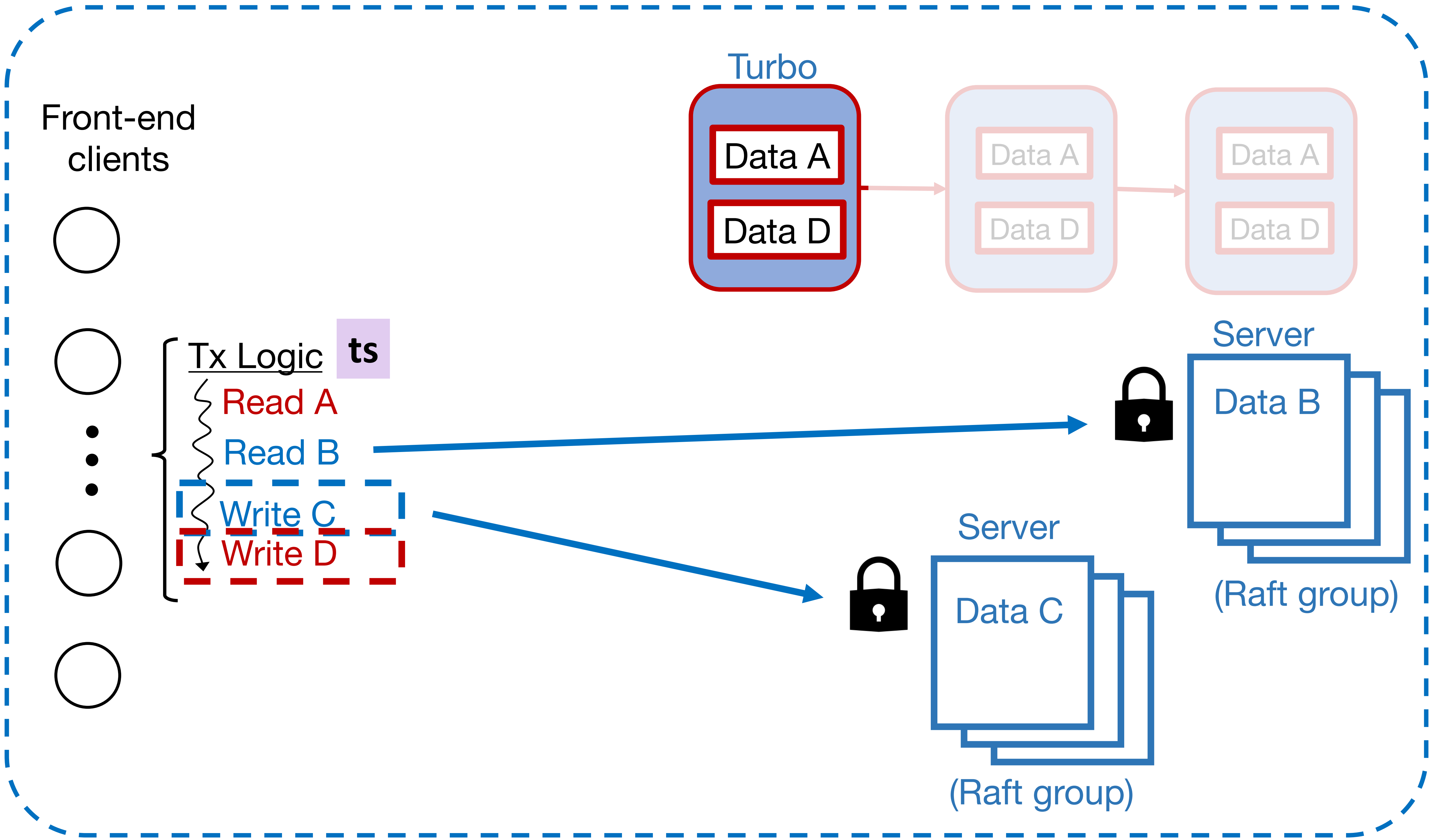
HCC Protocol

Execute Phase.

1. Client assigns ***timestamp ts***.
2. Client locally buffers update ops; sends (a) read ops to servers, and (b) standalone read-only requests to turbo.

Finale-Commit Phase

HCC Supports General, Multi-shot Transactions



HCC Protocol

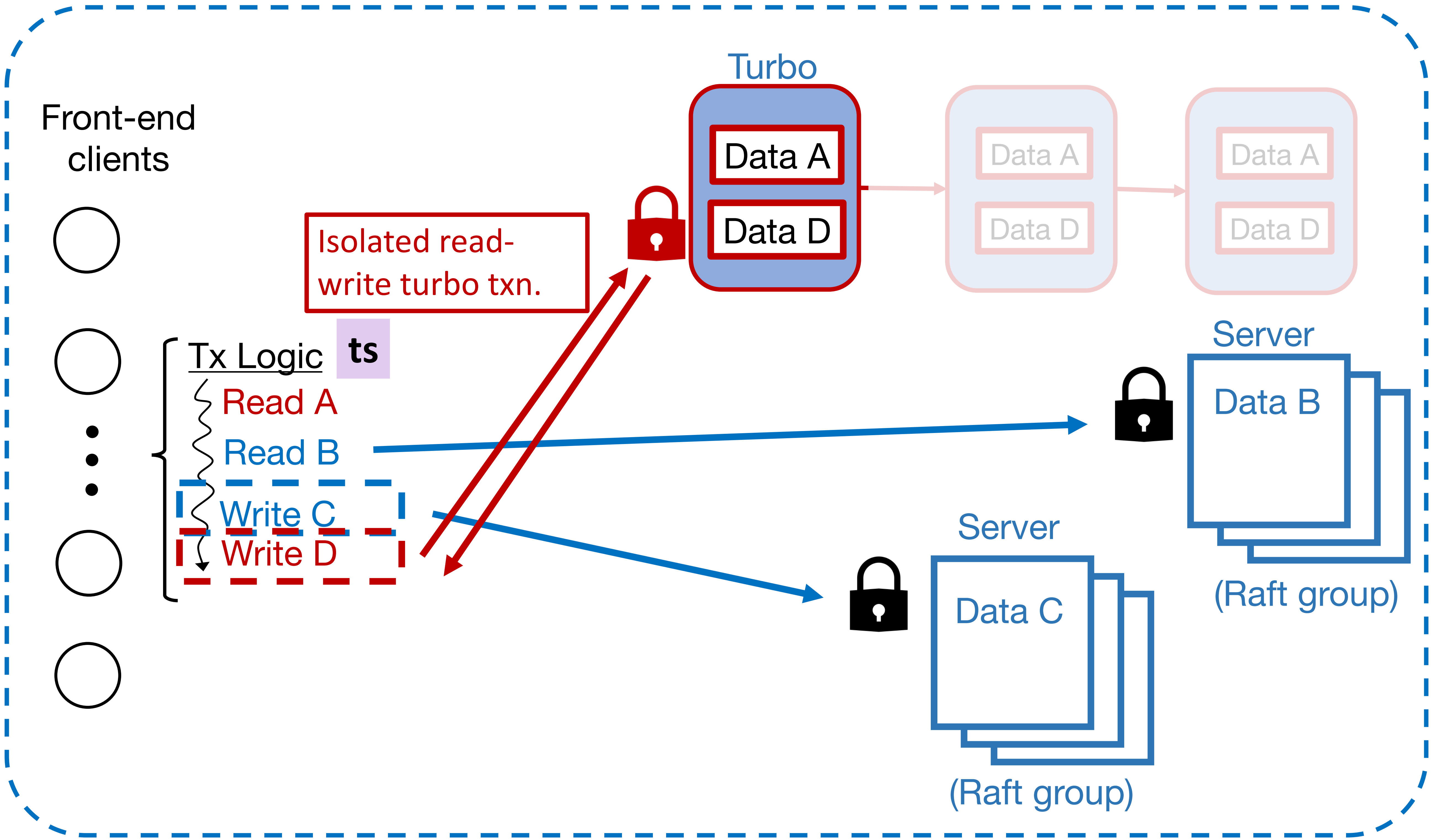
Execute Phase.

1. Client assigns ***timestamp ts***.
2. Client locally buffers update ops; sends (a) read ops to servers, and (b) standalone read-only requests to turbo.

Finale-Commit Phase (serial two-step commit)

3. Client sends buffered cool key updates to servers. Servers prepare to commit.

HCC Supports General, Multi-shot Transactions



HCC Protocol

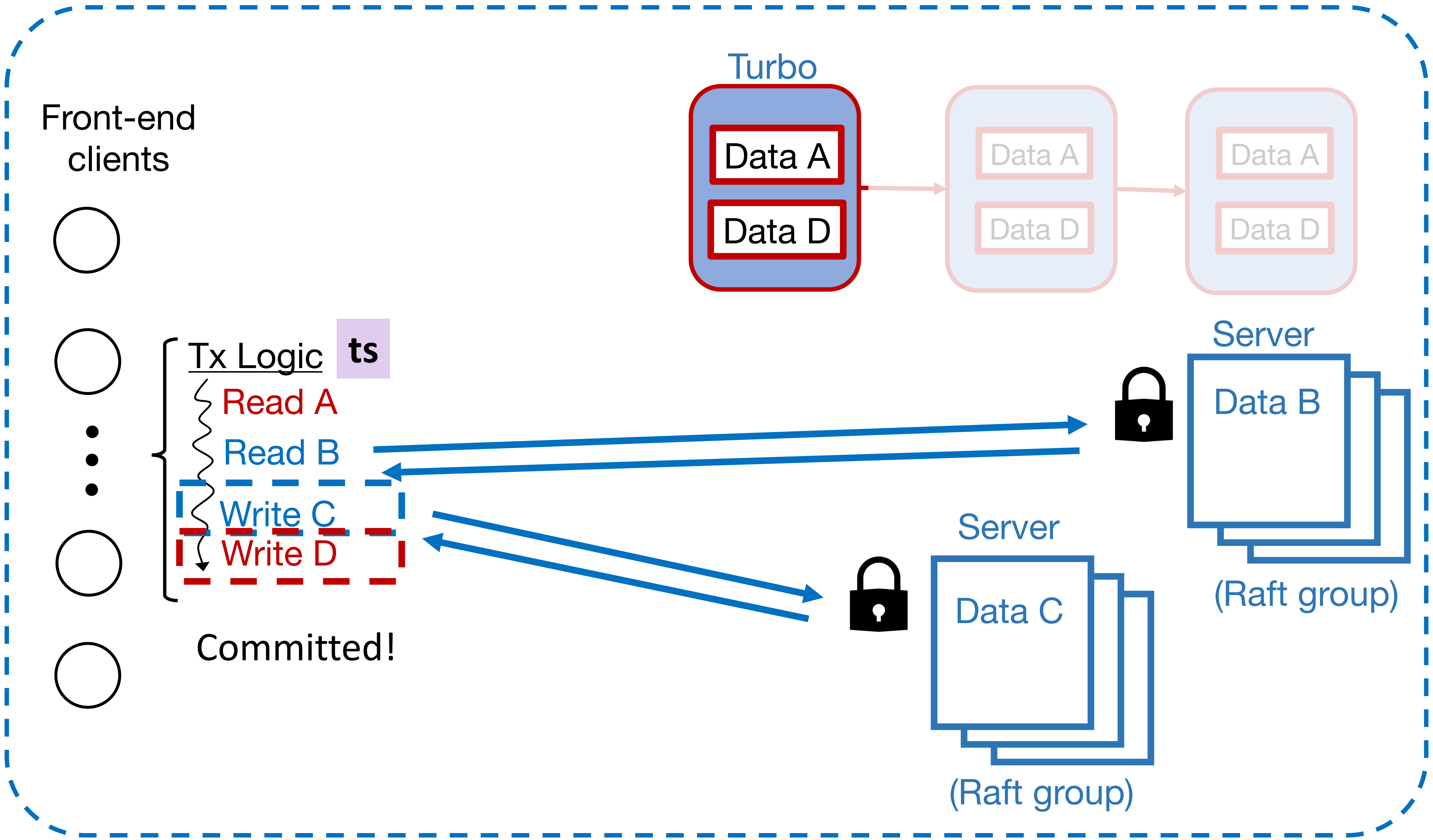
Execute Phase.

1. Client assigns ***timestamp ts***.
2. Client locally buffers update ops; sends (a) read ops to servers, and (b) standalone read-only requests to turbo.

Finale-Commit Phase (serial two-step commit)

3. Client sends buffered cool key updates to servers. Servers prepare to commit.
4. Client sends single read-write request to turbo. Turbo commits (aborts), and servers mirror decision.

HCC Supports General, Multi-shot Transactions



HCC Protocol

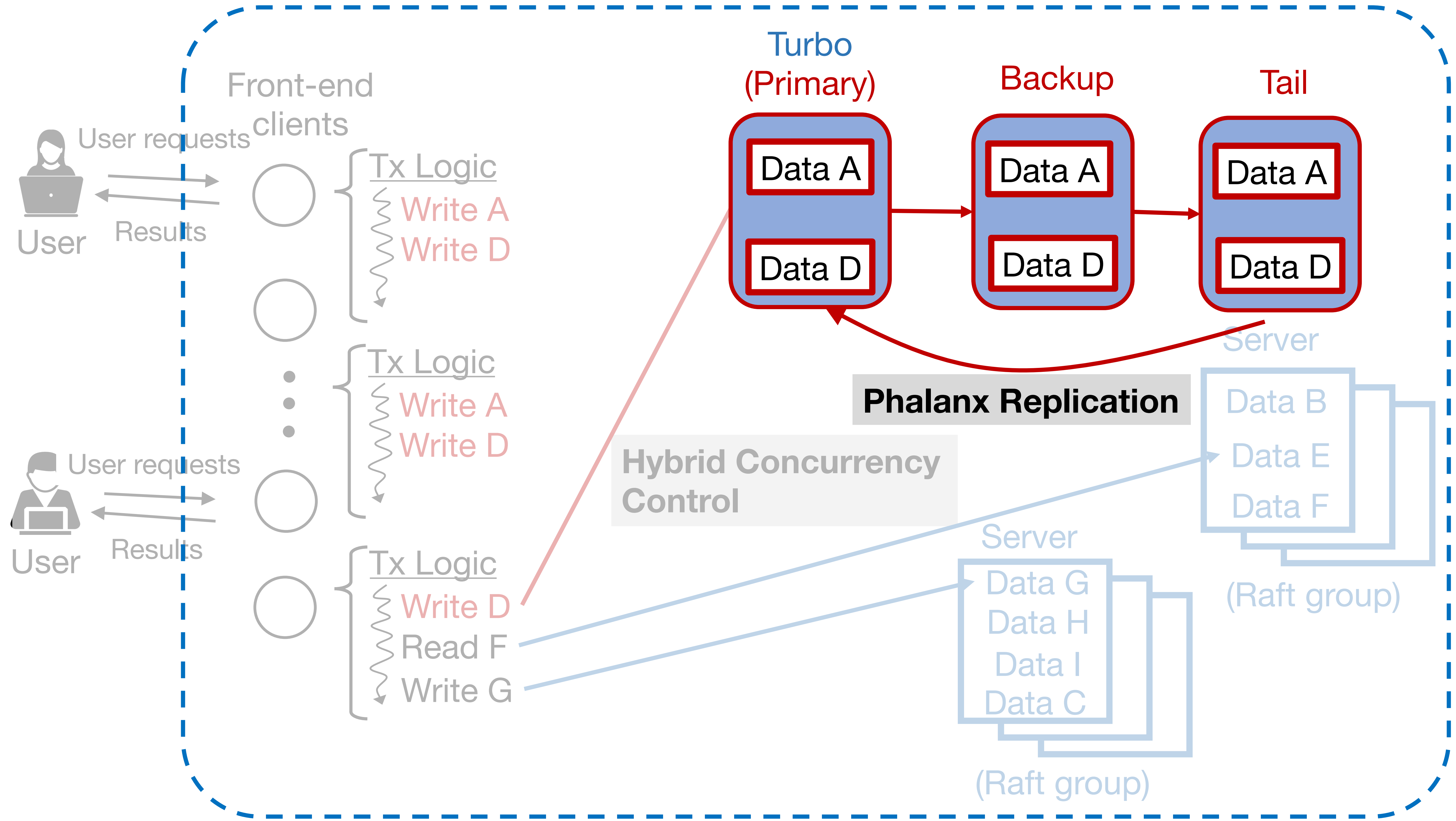
Execute Phase.

1. Client assigns ***timestamp ts***.
2. Client locally buffers update ops; sends (a) read ops to servers, and (b) standalone read-only requests to turbo.

Finale-Commit Phase (serial two-step commit)

3. Client sends buffered cool key updates to servers. Servers prepare to commit.
4. Client sends single read-write request to turbo. Turbo commits (aborts), and servers mirror decision.

Phalanx is a Turbo-Specific Replication Protocol



TurboDB Distributed Database

Phalanx Replication: Tolerating Turbo Failures

Phalanx Protocol Goal: correctly replicate the turbo's data, but without replication latency penalizing single-machine performance multipliers.

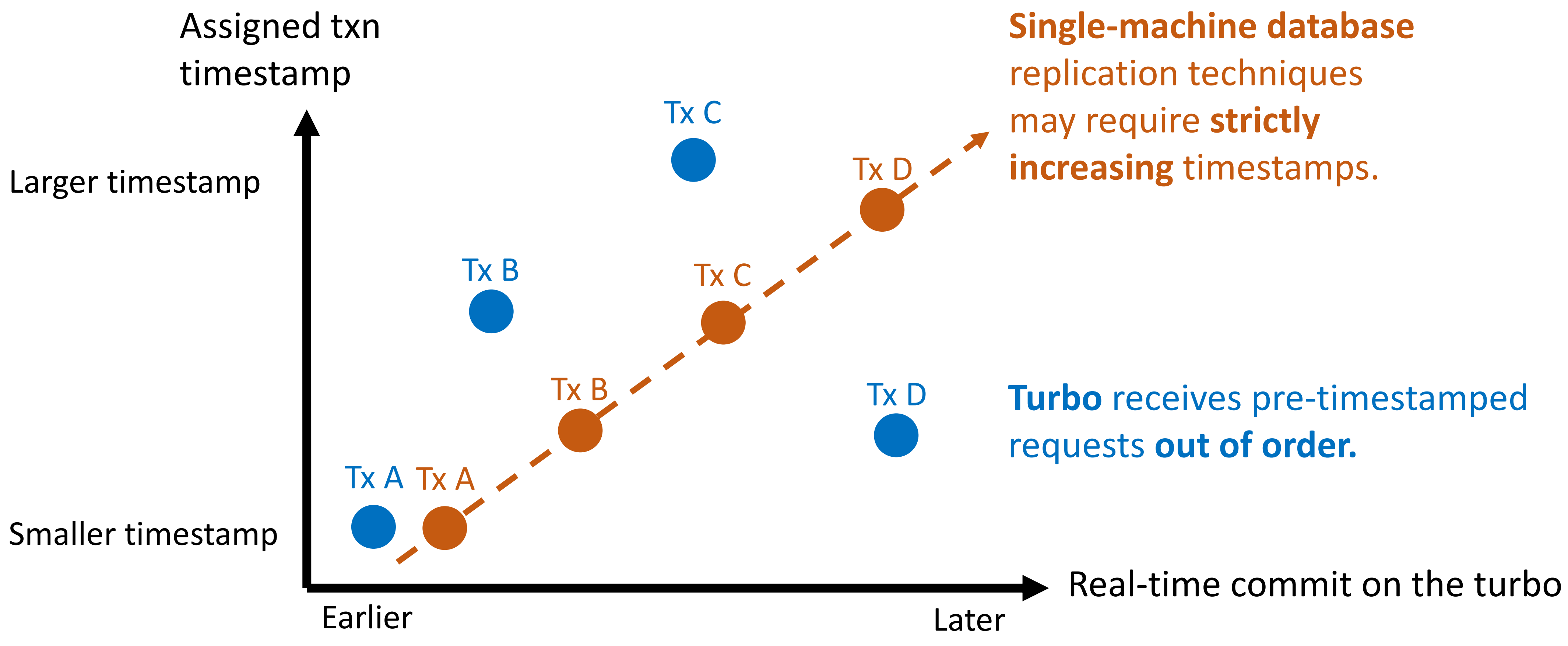
Intuition (from existing work): decouple replication from transaction execution.

- After committing transaction, turbo primary makes it visible *before and during* its replication, buffering transaction's results in the meantime.

Phalanx Returns Committed Transactions in Correct Order

Subtle issue: turbo's performance cannot tolerate returning buffered, committed transactions in timestamp order.

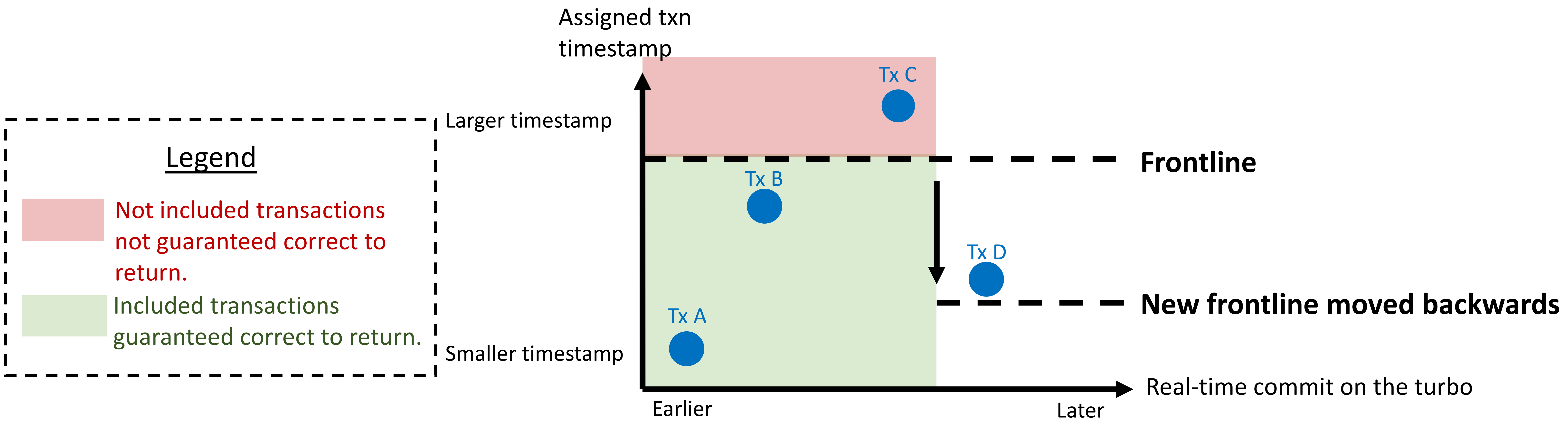
Solution: Frontline mechanism returns committed transactions in correct order without blocking progress.



Phalanx's Frontline Moves Forward and Backwards

Frontline definition: global threshold timestamp. Represents a snapshot of the turbo where all committed transactions can be correctly returned.

Frontline returns committed transactions in correct order by **selectively obeying timestamp order**.



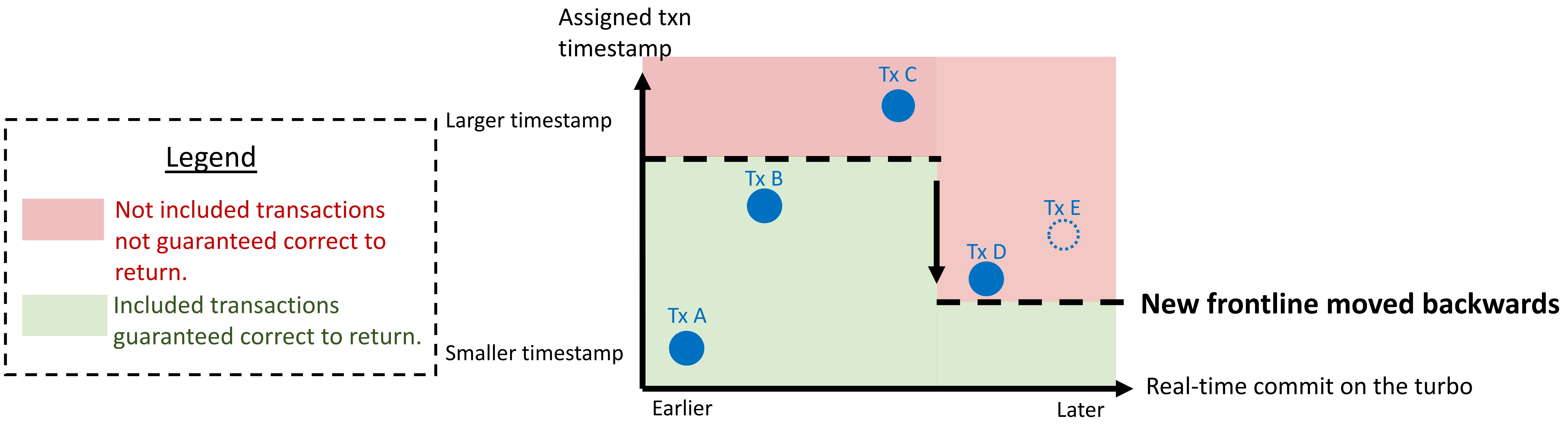
Frontline's Backward Movement is Correct

Prevents newly committed, un-replicated txns from being prematurely returned.

Does not revoke correctness of previously returned transactions.

1. Committed transaction is replicated.
2. If transaction depends on any prior transactions, those are also replicated.*

*HCC guarantees that if txn B depends on txn A, then txn B.ts > txn A.ts.



Implementation

Built on CockroachDB [SIGMOD ’20] and Cicada [SIGMOD ’17].

Baseline: CockroachDB.

Workloads:

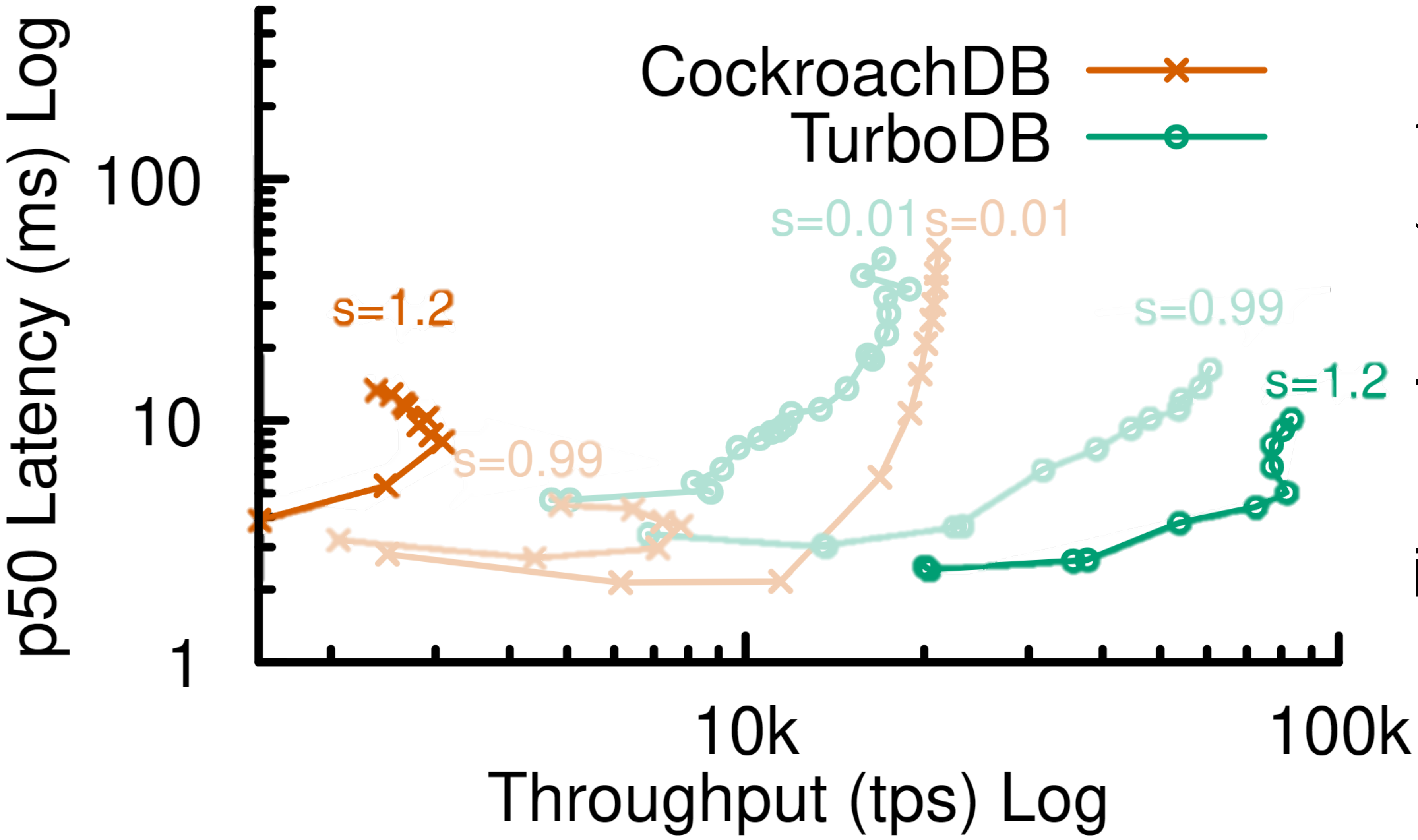
- YCSB+T.
- TPC-C New-Order transactions.
- Varying skew and read-to-write ratios.

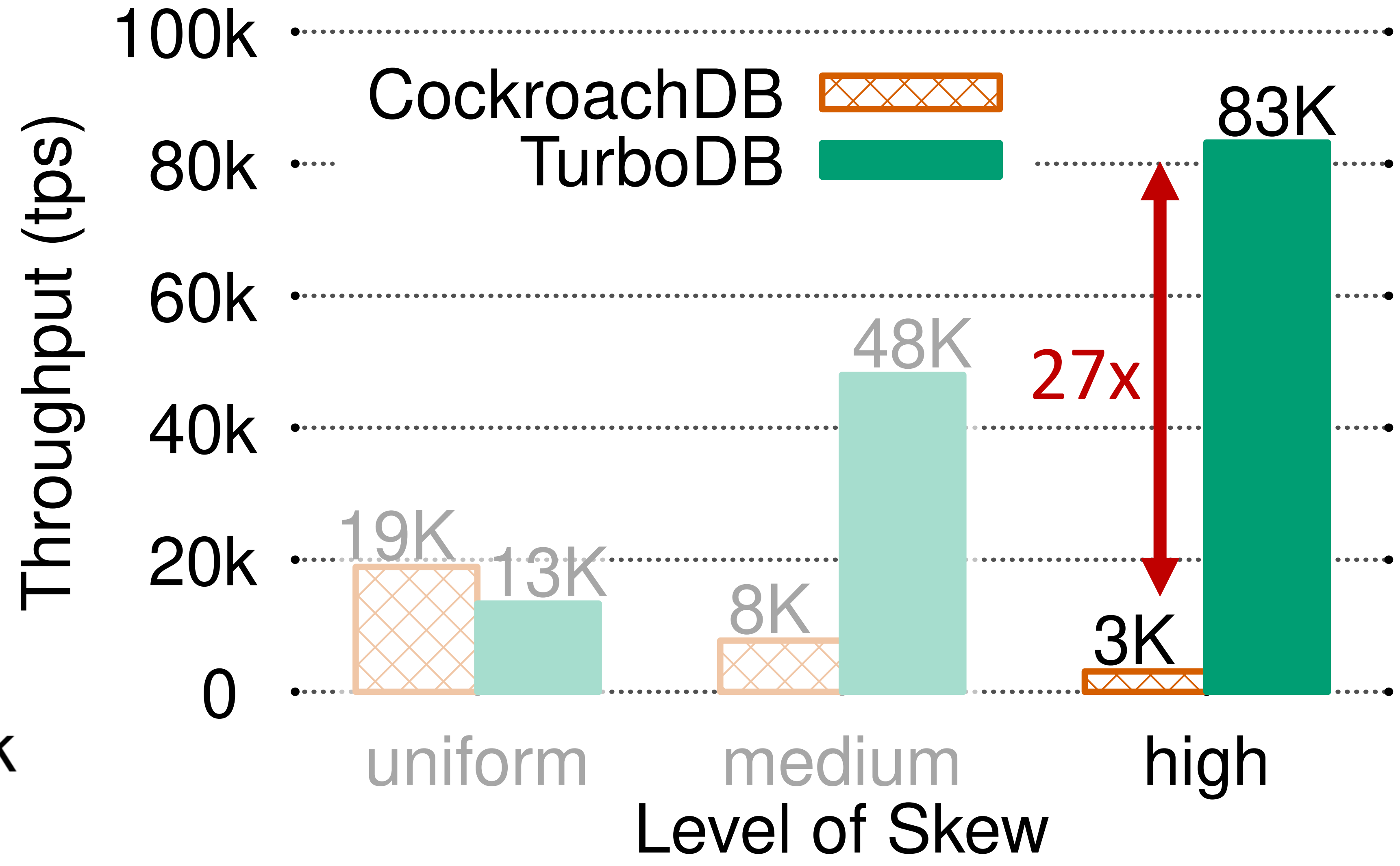
Performance metrics: throughput, latency, and scalability.

Evaluation

YCSB+T (95% reads, 5% updates): transaction size of 10 unique keys.

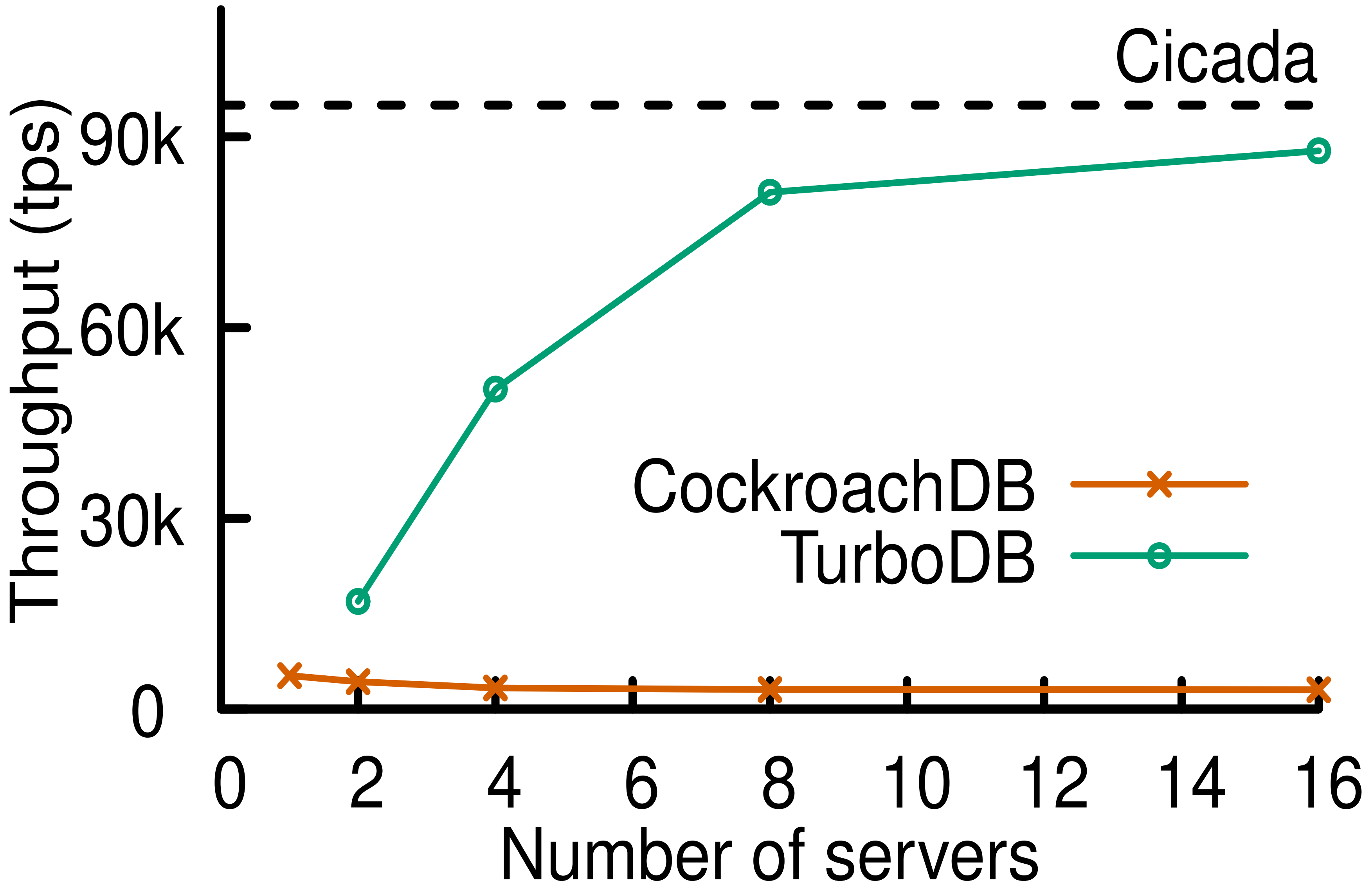
Cicada stores 40M most popular keys (of 160M total keys).





Evaluation

Scalability (YCSB+T) up to 16 nodes.



High skew (*s* = *1.2*)

Conclusion

TurboDB: a distributed database designed for skewed workloads.

A novel, hybrid database architecture.

- Integrates a single-machine database to "turbocharge" the overall performance.
- Leverages the turbo's performance multipliers.

Specialized designs for challenges unique to hybrid architecture.

- Hybrid Concurrency Control (HCC) ensures process-ordered serializability.
- Phalanx Replication tolerates turbo failures.

Implementation and evaluation of TurboDB.

- Up to an order of magnitude improvement under skewed workloads.
- Code: https://github.com/princeton-sns/TurboDB

Thank you!

Backup Slides

Determining and Migrating Popular Keys

Determine key popularity with per-key queries-per-second (QPS) count.

- Promote keys with highest QPS to turbo.

Custom migration protocol.

- Transaction deletes keys from servers and inserts them into turbo.
- Assumes distribution does not rapidly change (i.e. diurnal workloads).

Migration protocol runs during system warmup, but not evaluation experiments.