Diagnosing the Structure of Strict-Serializability Violations Across Spanner-like Read-Only Transaction Protocols

Sejong Kim
Korea University
Seoul, Republic of Korea
sejongkim@korea.ac.kr

Yon Dohn Chung
Korea University
Seoul, Republic of Korea
ydchung@korea.ac.kr

ABSTRACT

Stale-read anomalies occur when reads return older values than a stronger consistency model would allow. We study which read-only transactions (ROs) return snapshots that are stale relative to strict serializability (SS), how often this occurs, and how such anomalies arise, using a controlled family of Spanner-like protocols targeting three consistency models weaker than SS: regular sequential serializability (RSS), process-ordered serializability (POS), and serializability (SER). The protocols differ in which prepared conflicting read-write transactions (RWs) an RO can skip rather than wait for.

From execution traces, we build transaction dependency graphs and analyze when RO skips lead to SS-relative stale-read anomalies, which appear as violation cycles in these graphs. Under POS and SER, ROs are far more likely to have at least one skip that is part of such a cycle than under RSS. We identify the key pattern as an anti-regular anti-dependency (ARA) pair: an RO–RW pair in which the RO skips a prepared conflicting RW that is the next committed writer after the version returned by the RO, even though the RW completed before the RO began. Each ARA pair directly forms a violation cycle. More importantly, the anti-dependency edges of ARA pairs often help non-ARA pairs form cycles, making ARA pairs the main source of POS/SER stale-read amplification. RSS suppresses this mechanism by preserving regularity and therefore ruling out ARA pairs. Finally, we associate the formation of ARA pairs with cross-region visibility lag: an RW may complete through its coordinator while remote participants still treat it as prepared, allowing a later RO to skip it.

VLDB Workshop Reference Format:
Sejong Kim and Yon Dohn Chung. Diagnosing the Structure of Strict-Serializability Violations Across Spanner-like Read-Only Transaction Protocols. VLDB 2026 Workshop: Symposium on Consistency Checking Principles (SCCP).

VLDB Workshop Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/sejongk/spanner-like-ro-protocols.

1 INTRODUCTION

Consistency anomalies in storage services are unexpected behaviors visible to users. A stale-read anomaly [15] is one such anomaly: a read returns an old value even though a more recent write should have been visible under a stronger consistency model. For example, a user may add an item to a shopping cart and then refresh the cart, only to find that the item is missing. Stronger consistency can rule out more such anomalies by enforcing more ordering constraints, but usually at a performance cost. Within this trade-off, prior anomaly-measurement studies [15, 21] ask a service-level question: under a weaker consistency model, which anomalies arise relative to a stronger reference model, and how often do they actually occur in practice?

We study this question in Spanner-like protocols for read-only transactions (ROs), focusing on stale snapshots returned by ROs. Under strict serializability (SS) [16], an RO must return a legal snapshot that can be serialized after every transaction that completed before the RO began, i.e., after every transaction that precedes the RO in real time. For ROs, Spanner [4] enforces this requirement by making an RO wait until each shard can serve its reads from a snapshot that reflects all committed read-write transactions (RWs) up to the RO's read timestamp. We ask what happens when this RO wait rule is weakened under consistency models weaker than SS, allowing ROs to *skip*, rather than wait for, some prepared conflicting RWs: *Which ROs return snapshots that are stale relative to SS, how often does this occur, and how do these anomalies arise?*

We analyze a controlled family of Spanner-like RO protocols targeting three consistency models progressively weaker than SS: regular sequential serializability (RSS) [6], process-ordered serializability (POS) [14], and serializability (SER) [16]. Building on Spanner-RSS [6], we implement new POS and SER variants that keep the same Spanner-like substrate but change only the model-specific RO wait/skip condition and the metadata needed to evaluate it. This controlled family lets us isolate how relaxing specific ordering constraints—especially *regularity* [6, 11], which RSS preserves but POS/SER do not—changes the frequency and structure of SS-relative stale-read anomalies.

From each execution trace, we build a transaction dependency graph augmented with real-time order. In this graph, SS-relative stale-read anomalies appear as cycles, each showing that no legal total order can simultaneously satisfy all ordering constraints required by SS. We then diagnose which RO–RW skip pairs (R, W) become cycle-forming and how they do so. A skip pair is *cycle-forming* when two conditions hold. First, R skips W, and W is the next committed writer after the version returned by R on some key, creating a *skip-induced anti-dependency* from R to W. Second, the graph also contains a *return path* from W back to R. Together, the anti-dependency and the return path form a cycle: the anti-dependency requires R to be serialized before W, while the return path forces W before R.

Proceedings of the VLDB Endowment. ISSN 2150-8097.

Our first finding is that skip-induced anti-dependency volume alone does not explain the large increase in cycle-forming pairs in POS and SER. POS and SER produce only 1.5/2.0× as many skip-induced anti-dependency pairs as RSS, but 181/359× as many cycle-forming pairs. This pair-level gap is also reflected at the RO level: 0.02/3.6/7.0% of ROs participate in at least one cycle-forming skip pair under RSS/POS/SER, respectively. Second, we find that the key primitive is an *anti-regular anti-dependency (ARA)* pair: an RO–RW pair with a skip-induced anti-dependency from R to W such that W already precedes R in real time. Each ARA pair is cycle-forming by construction. More importantly, for many non-ARA pairs, the anti-dependency edges of ARA pairs contribute substantially to the return paths that make those non-ARA pairs cycle-forming: in our baseline traces, removing all such edges from the dependency graph leaves 98.3% and 98.7% of non-ARA cycle-forming pairs without a return path in POS and SER, respectively. RSS suppresses this ARA-backed structure by preserving regularity, thereby ruling out ARA pairs. Third, we associate ARA pairs with cross-region visibility lag: after an RW completes through its coordinator, some participant shards may not yet have received or applied the commit decision and can still treat the RW as prepared, allowing later ROs to skip it.

2 BACKGROUND

We summarize the consistency models and the original Spanner transaction protocols [4] that underlie the Spanner-like RO protocols studied in this paper.

2.1 Transactional Consistency Models

Order constraints and conflicts. Given an execution α, we write $T_1 \xrightarrow{\text{RT}} T_2$ when transaction T_1 completes before transaction T_2 starts; this relation is the *real-time order*. Following Spanner-RSS [6], we define *causal order* over actions as the smallest transitive relation containing: (1) process order: within each process, earlier actions precede later actions; (2) message-passing order: a send action precedes its corresponding receive action; and (3) reads-from order: the response of a transaction that writes a version precedes the invocation of a transaction that reads that version. We say that T_1 causally precedes T_2 when T_1's response causally precedes T_2's invocation. We say an RO R *conflicts* with an RW W if R reads at least one key that W writes.

Consistency models. All models considered here require a legal total order of committed transactions, i.e., one in which each read returns the latest preceding committed version of the key. They differ only in the order constraints this total order must respect. SS [16] respects real-time order. RSS [6] respects causal order and the transactional regularity condition: if an RW W precedes a transaction T in real time and T is either another RW or a conflicting RO, then W must precede T in the total order. POS [14] respects each process's order, but not real-time order across processes. SER imposes no additional order constraint beyond legality.

2.2 Spanner

Spanner [4] is a globally distributed database that shards data across Paxos [12] groups and provides strictly serializable transactions using TrueTime [4]. A call to TT.NOW() returns an interval [*earliest*, *latest*] containing the absolute time of the call.

RW Transactions. Spanner executes an RW W using two-phase locking [3] and two-phase commit [5]. Each participant shard s prepares W by acquiring the necessary write locks, durably logging a prepare record, and reporting a prepare timestamp $t_{\text{p},s}(W)$ to the coordinator. After collecting prepare responses, the coordinator chooses a commit timestamp $t_{\text{c}}(W)$ that is no smaller than the participant prepare timestamps and satisfies Spanner's timestamp-ordering rules. The coordinator then logs the commit decision and performs commit-wait before replying to the client. Participants later receive the commit decision, apply W with commit timestamp $t_{\text{c}}(W)$, and release their locks.

RO Transactions. For an RO R, the client chooses $t_{\text{read}}(R) = \text{TT.NOW}().\mathit{latest}$, which is no earlier than R's start time. A shard can serve R only after its Paxos safe time reaches $t_{\text{read}}(R)$, meaning that all Paxos-replicated writes with timestamps at or before $t_{\text{read}}(R)$ have been applied. The transaction manager also prevents R from missing a prepared conflicting RW that may later commit with $t_{\text{c}}(W) \le t_{\text{read}}(R)$, so R waits at shard s for every prepared conflicting RW W such that

$$\text{Wait}_{\text{SS}}(R, W, s) \triangleq t_{\text{p},s}(W) \le t_{\text{read}}(R).$$

3 SPANNER-LIKE RO PROTOCOL FAMILY

We study a controlled family of Spanner-like RO protocols. All variants share the same Spanner RW protocol structure, RO read timestamp selection, Paxos safe-time check, and Spanner-RSS snapshot assembly; they differ only in their model-specific RO wait/skip conditions and the metadata needed to evaluate them. This design isolates how weakening the RO wait condition—and thereby relaxing specific ordering constraints—shapes SS-relative stale-read anomalies.

3.1 Spanner-RSS

Spanner-RSS weakens Spanner's shard-side RO wait condition while preserving RSS. It adds two pieces of metadata. For an RO R, a client-maintained lower bound $t_{\text{min}}(R)$ summarizes causally prior transactions; clients advance the bound after their own transactions complete and when they receive messages carrying causality metadata from other clients. For an RW W, $t_{\text{ee},s}(W)$ is a conservative lower bound on W's client-side end time, used to preserve the conflicting RW→RO real-time order required by regularity.

After performing the same Paxos safe-time check as Spanner, a shard s examines prepared RWs that conflict with R. We say that R *skips* W at s if R observes W as a prepared conflicting RW at s, $\text{Wait}_{\text{SS}}(R, W, s)$ holds, and the skip predicate $\text{Skip}_{\text{RSS}}(R, W, s)$ holds; in that case, s allows R to proceed without waiting for W. Prepared RWs for which Wait_{SS} is false are not counted as skipped. Thus, the wait/skip predicates are

$$\text{Wait}_{\text{RSS}}(R, W, s) \triangleq \text{Wait}_{\text{SS}}(R, W, s) \wedge \neg\text{Skip}_{\text{RSS}}(R, W, s),$$
$$\text{Skip}_{\text{RSS}}(R, W, s) \triangleq t_{\text{p},s}(W) > t_{\text{min}}(R) \wedge t_{\text{ee},s}(W) > t_{\text{read}}(R).$$

The $t_{\text{p},s}(W) \le t_{\text{min}}(R)$ wait clause preserves causal order, while the $t_{\text{ee},s}(W) \le t_{\text{read}}(R)$ clause preserves regularity.

All variants use the Spanner-RSS fast/slow snapshot assembly: if a skipped RW may commit at or before the RO's chosen snapshot timestamp, the client waits for the RW's slow reply and incorporates the RW's writes if needed (Appendix B).

3.2 Spanner-RO-POS and Spanner-RO-SER

We implement Spanner-RO-POS and Spanner-RO-SER as controlled ablation endpoints that further weaken the Spanner-RSS RO wait/skip predicate while keeping the common Spanner-like substrate fixed. The correctness proofs appear in Appendix B.

For Spanner-RO-POS, $t_{\min}(R)$ is a process-order lower bound: clients advance the bound after their own RWs and ROs complete, but do not update it from messages carrying causality metadata from other clients. Spanner-RO-SER does not use $t_{\min}(R)$, and neither variant uses $t_{\mathrm{ee},s}(W)$.

After the Paxos safe-time check, shard s examines prepared RWs that conflict with R. For $M \in \{\mathrm{POS}, \mathrm{SER}\}$, the wait/skip predicates are

$$\mathsf{Wait}_M(R, W, s) \triangleq \mathsf{Wait}_{\mathrm{SS}}(R, W, s) \wedge \neg \mathsf{Skip}_M(R, W, s),$$
$$\mathsf{Skip}_{\mathrm{POS}}(R, W, s) \triangleq t_{\mathrm{p},s}(W) > t_{\min}(R),$$
$$\mathsf{Skip}_{\mathrm{SER}}(R, W, s) \triangleq \mathsf{true}.$$

Thus, POS keeps only the process-order lower-bound clause, while SER skips every prepared conflicting RW for which $\mathsf{Wait}_{\mathrm{SS}}$ holds.

4 ANALYSIS FRAMEWORK

We analyze SS-relative violations in the Spanner-like RO protocol family. Our goal is not to find violations of each variant's target model, but to use the ordering constraints required by SS as a reference for diagnosing the effect of weakening the RO wait condition.

4.1 Dependency Graph

From each trace, we construct an Adya-style Direct Serialization Graph $DSG = (V, E_{\mathrm{DSG}})$ [1], where V contains one vertex for each committed transaction. Using committed write versions and returned read versions, we construct E_{DSG} with write-write (WW) and write-read (WR) dependency edges, and read-write (RW) anti-dependency edges.

To capture the real-time order required by SS, we augment the DSG with real-time (RT) edges. For soundness, we identify a conservative RT edge $T_1 \xrightarrow{\mathrm{RT}} T_2$ only when

$$\widehat{t}_{\mathrm{resp}}(T_1) + \varepsilon < \widehat{t}_{\mathrm{invoc}}(T_2) - \varepsilon,$$

where $\widehat{t}_{\mathrm{invoc}}(T)$ and $\widehat{t}_{\mathrm{resp}}(T)$ are trace-recorded invocation and response timestamps, respectively, and ε is the clock uncertainty bound. This conservative rule guarantees that every added RT edge represents a true real-time precedence relation despite clock uncertainty. Since RT order is transitive and potentially quadratic, we store a reduced set of RT edges using an RT transitive reduction [17]. The reduction preserves RT reachability: for distinct transactions T_1 and T_2, we write $T_1 \xrightarrow{\mathrm{RT}^*} T_2$ when T_1 reaches T_2 through an RT-only path in the reduced graph, and treat each such reachability relation as a single logical edge.

We define and analyze the following logical dependency graph:

$$G = (V, E_{\mathrm{DSG}} \cup E_{\mathrm{RT}^*}), \qquad E_{\mathrm{RT}^*} = \{(T_1, T_2) \mid T_1 \xrightarrow{\mathrm{RT}^*} T_2\}.$$

Because all variants preserve serializability—Spanner-RSS by its RSS guarantee and Spanner-RO-POS/SER by Appendix B—the DSG is acyclic; thus, cycles in G are SS-relative violation witnesses that arise only after adding real-time order. We compute strongly connected components (SCCs) of G using Tarjan's algorithm [19] and use breadth-first search within an SCC when a concrete shortest-hop cycle witness is needed.

4.2 Skip-to-cycle Cascade

We identify which RO–RW pairs (R, W) form SS-relative violation cycles using the following three-stage cascade of filters:

(1) **F1 (Final skip pair).** RO R observes a prepared conflicting RW W at some shard, skips it, and W remains excluded from R's final assembled snapshot.
(2) **F2 (Skip-induced anti-dependency pair).** An F1 pair becomes F2 if the skipped RW W is the next committed writer after the version returned by R on some key, yielding $R \xrightarrow{\mathrm{RW}_{\mathrm{skip}}} W$.
(3) **F3 (Cycle-forming pair).** An F2 pair becomes F3 if it has a *return path*: W reaches R in G, written $W \rightsquigarrow_G R$. Together, the skip-induced anti-dependency and the return path form an SS-relative violation cycle: $R \xrightarrow{\mathrm{RW}_{\mathrm{skip}}} W \rightsquigarrow_G R$.

Operationally, we identify F3 pairs as F2 pairs whose R and W belong to the same non-trivial SCC of G. For each F3 pair, we instantiate a cycle witness by combining the skip-induced anti-dependency $R \xrightarrow{\mathrm{RW}_{\mathrm{skip}}} W$ with a shortest path from W back to R within that SCC.

4.3 Positive Anti-dependency Margin: Necessary Condition for Skip-induced Cycles

For the Spanner-like RO protocol family, we prove that every skip-induced SS-relative violation cycle contains at least one F3 pair with positive *anti-dependency margin (AM)* (i.e., $\mathrm{AM} > 0$); the proof appears in Appendix C. For an F2 pair (R, W), including any F3 pair, we define its AM as

$$\mathrm{AM}(R, W) = \widehat{t}_{\mathrm{invoc}}(R) - \varepsilon - t_{\mathrm{c}}(W).$$

We use AM as a diagnostic statistic rather than a cycle detector: positive AM is necessary at the cycle level, but not sufficient for an individual pair to be cycle-forming.

5 ANALYSIS RESULTS

We now apply the analysis framework from Section 4 to diagnose SS-relative violation cycles in the Spanner-like RO protocol family. All reported counts and percentages are independently rounded to the displayed precision.

5.1 Experimental Setup

We use Retwis [13], following prior Spanner-RSS evaluations [6], because its multi-key RO and RW transactions generate cross-shard conflicts and prepared-RW skip opportunities, while allowing load and contention to be controlled through the number of clients and key-access skew. The workload does not model message passing between clients; thus, the RSS–POS contrast primarily isolates the regularity-preserving $t_{\mathrm{ee},s}(W) \le t_{\mathrm{read}}(R)$ wait clause, rather than causality propagated through messages. Accordingly, our goal is to

Table 1: Skip-to-cycle cascade. Skip is the fraction of all ROs that appear in at least one F1 pair, while the F3 ROs column reports the number of distinct ROs that appear as R in at least one F3 pair, with parentheses showing their fraction of all ROs. F1–F3 count RO–RW pairs, with parentheses showing survival from the previous filter. All models process a similar number of ROs: 1.613–1.617M.

Model	Skip	F1	F2	F3	F3 ROs
RSS	11.4%	197.8K	158.0K (79.8%)	334 (0.2%)	333 (0.02%)
POS	15.8%	283.7K	232.5K (82.0%)	60.4K (26.0%)	57.6K (3.6%)
SER	21.4%	390.1K	323.4K (82.9%)	119.8K (37.0%)	112.5K (7.0%)

isolate the structural effect of relaxing specific ordering constraints, not to estimate workload-independent anomaly rates.

Each experiment runs for 600 s, including a 120 s warm-up, under Zipfian skew 0.9. We run experiments on six machines (64 cores, 327 GB RAM, and a 10 Gbps NIC each), grouped into three emulated regions with two machines per region. We use 3 shards with 3 replicas per shard. Each shard's replicas are co-located within one region, while different shards are placed in different regions. Inter-region round-trip times (RTTs) are 64–130 ms, emulated using Linux TC [8]. The measured clock uncertainty used for RT extraction is below 100 μs, and the TrueTime uncertainty configured for the protocol is 4 ms. For each protocol, we collect more than 2.61M transaction traces (1.001–1.003M RWs and 1.613–1.617M ROs). Additional results under different skew, replica placement, RTT, and RO/RW mix settings are reported in Appendix A.

5.2 F2 Volume Alone Does Not Explain F3 Amplification

RSS, POS, and SER process similar numbers of ROs (1.613/1.615/1.617M). In Table 1, 11.4/15.8/21.4% of all ROs appear in at least one F1 pair under RSS/POS/SER, respectively. After the F2 filter, 158.0K/232.5K/323.4K F2 pairs remain: POS and SER have only 1.5/2.0× as many F2 pairs as RSS. However, the F3 gap is much larger: POS/SER produce 60.4K/119.8K F3 pairs, 181/359× RSS's 334. Thus, F2 volume alone does not explain POS/SER F3 amplification.

The corresponding fractions of distinct ROs that appear as R in at least one F3 pair are 0.02/3.6/7.0% in RSS/POS/SER. Thus, the pair-level amplification is also reflected in a substantially higher fraction of ROs involved in SS-relative stale-read anomalies under POS and SER.

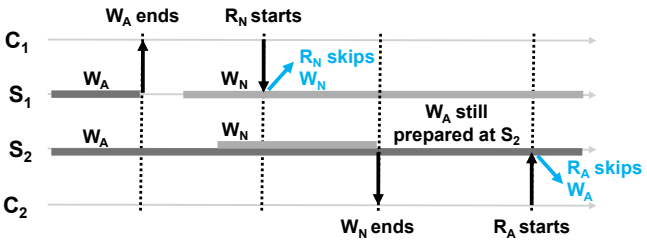
5.3 ARA Pairs Account for a Substantial Share of POS/SER F3 Pairs

We next investigate which F2 pairs become F3 pairs. The key primitive is an *anti-regular anti-dependency (ARA)*: a skip-induced anti-dependency $R \xrightarrow{\text{RW}_{\text{skip}}} W$ whose skipped RW W precedes the skipping RO R in real time, $W \xrightarrow{\text{RT}^*} R$. Thus, an ARA pair is cycle-forming by construction. We call its skip-induced anti-dependency edge an *ARA edge* $R \xrightarrow{\text{RW}_{\text{ARA}}} W$.

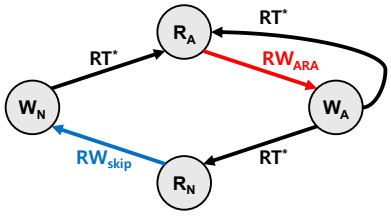
In Figure 1a, (R_A, W_A) is an ARA pair: R_A skips W_A because S_2 still treats W_A as prepared, while W_A precedes R_A in real time.

Table 2: Composition of F2 and F3 pairs. ARA denotes anti-regular anti-dependencies; Non-ARA denotes the remaining F2 pairs. Percentages are normalized within each model and filter.

Model	F2 (ARA / Non-ARA)	F3 (ARA / Non-ARA)
RSS	0 (0.0%) / 158.0K (100.0%)	0 (0.0%) / 334 (100.0%)
POS	28.8K (12.4%) / 203.7K (87.6%)	28.8K (47.7%) / 31.6K (52.3%)
SER	47.4K (14.6%) / 276.1K (85.4%)	47.4K (39.5%) / 72.5K (60.5%)



(a) Concrete execution



(b) Induced dependency graph

Figure 1: Example of ARA/non-ARA pairs and their cycle formation. In (a), client C_1 issues RW W_A and RO R_N, and C_2 issues W_N and R_A; shards S_1 and S_2 coordinate the multi-shard RWs W_A and W_N, respectively. Both pairs (R_A, W_A) and (R_N, W_N) have a skip-induced anti-dependency $R \xrightarrow{\text{RW}_{\text{skip}}} W$: R skips W and returns an older version on some key; in the committed history, W is the next committed writer of that key after the version returned by R. (R_A, W_A) is ARA because $W_A \xrightarrow{\text{RT}^*} R_A$, whereas (R_N, W_N) is non-ARA because W_N does not precede R_N in real time. In (b), (R_A, W_A) is cycle-forming by construction via $W_A \xrightarrow{\text{RT}^*} R_A$; (R_N, W_N) becomes cycle-forming via the return path $W_N \xrightarrow{\text{RT}^*} R_A \xrightarrow{\text{RW}_{\text{ARA}}} W_A \xrightarrow{\text{RT}^*} R_N$.

This pair is cycle-forming by construction, as shown in Figure 1b: $R_A \xrightarrow{\text{RW}_{\text{ARA}}} W_A \xrightarrow{\text{RT}^*} R_A$.

In Table 2, RSS has no ARA pairs in either F2 or F3. In contrast, ARA pairs account for 12.4% of POS F2 pairs and 14.6% of SER F2 pairs; all of them become F3. Although ARA pairs are a minority of F2, they account for a large fraction of F3: 47.7% in POS and 39.5% in SER. RSS's zero ARA count follows directly from regularity: an RO cannot skip a conflicting RW that precedes it in real time. The important contrast is that POS and SER allow only a modest ARA minority at F2, yet these pairs account for a substantial share of F3.

5.4 Non-ARA Pairs Rely on ARA Edges for Return Paths

Non-ARA pairs become F3 much less often than ARA pairs: 203.7K → 31.6K (15.5%) in POS and 276.1K → 72.5K (26.3%) in SER. Because they dominate F2, however, they still make up 52.3% and 60.5% of F3 in POS and SER, respectively (Table 2).

The key question is how these non-ARA pairs (R, W) obtain a cycle-forming return path $W \rightsquigarrow_G R$. We remove all ARA edges from G and, for each non-ARA F3 pair, recompute whether the pair still has a return path. This removes the return path for 98.3% of POS and 98.7% of SER non-ARA F3 pairs; only 1.7% and 1.3% remain cycle-forming, respectively. Concretely, the dominant shortest-hop return-path witness signatures for non-ARA F3 pairs are pure ARA–RT chains: RT^*–RW_{ARA}–RT^*, repeated one or more times. These dominant signatures account for approximately 94.6% and 94.9% of non-ARA witnesses in POS and SER, respectively. Thus, most non-ARA F3 pairs rely on ARA edges in the graph to obtain return paths.

In Figure 1a, (R_N, W_N) is a non-ARA pair: R_N skips W_N, but W_N does not precede R_N in real time. Also, W_A precedes R_N in real time, and W_N precedes R_A in real time. As shown in Figure 1b, (R_N, W_N) becomes cycle-forming via the return path $W_N \xrightarrow{\text{RT}^*} R_A \xrightarrow{\text{RW}_{\text{ARA}}} W_A \xrightarrow{\text{RT}^*} R_N$, which relies on the ARA edge $R_A \xrightarrow{\text{RW}_{\text{ARA}}} W_A$.

Appendix A reinforces this interpretation. Under global replica placement, where each shard's replicas are distributed across regions, ARA's direct share of F3 falls to 19.9/18.3% in POS/SER, yet removing ARA edges removes the return paths for 99.7/99.8% of non-ARA F3 pairs. The same pattern appears under the RO-heavy setting, which uses a 90:10 RO/RW transaction-selection ratio: ARA accounts for only 20.9% of F3 pairs in both POS and SER, while removing ARA edges removes the return paths for 98.6/98.4% of non-ARA F3 pairs, respectively.

5.5 ARA Pairs Are Associated with Cross-region Visibility Lag

ARA can form when an RW has completed through its coordinator but a participant still treats it as prepared. A later RO can then skip this already-completed RW at that participant. The example in Figure 1 illustrates this cross-region visibility-lag structure. For the ARA pair (R_A, W_A), W_A's client C_1 is close to coordinator S_1, while R_A's client C_2 is close to participant S_2. By the time R_A reaches S_2, W_A has already completed through S_1 and returned to C_1, so W_A precedes R_A in real time. However, S_2 still treats W_A as prepared because the commit decision has not yet been applied at S_2. Thus, R_A can skip the already-completed W_A at S_2, creating an ARA pair.

For every per-shard skip observation (R, W, s) associated with an ARA pair, the shard s where R skips W is a participant of W, not W's coordinator. W's client is almost always local to the coordinator (97.4% in both POS and SER), while the participant is far from the coordinator (median RTT 130 ms). Conversely, R's client is usually local to that participant (77.7% in POS, 76.7% in SER). Thus, cross-region shard placement opens an ARA-prone visibility-lag window: the period after W completes through a coordinator near W's client but before the commit decision is applied at remote participants, during which a later RO R can skip W at a remote participant close to R's client.

5.6 AM Provides a Cycle-level Diagnostic

Consistent with the cycle-level positive-AM necessary condition, removing all positive-AM skip-induced anti-dependency edges eliminates all detected skip-induced cycles. ARA pairs have positive median AM: 23.8 ms in POS and 24.0 ms in SER. In contrast, non-ARA F2 pairs have negative median AM in all three models: −34.0/−33.5/−33.0 ms in RSS/POS/SER. Non-ARA F3 pairs shift upward relative to all non-ARA F2 pairs, with medians 30.3/−9.6/−13.0 ms. Since RSS has no ARA pairs, positive-AM witnesses in RSS must come from non-ARA F3 pairs. In POS/SER, non-ARA F3 pairs often have non-positive AM, but their cycle witnesses contain other larger-AM skip-induced anti-dependency edges: the median witness maximum AM is 31.1/50.8/51.6 ms in RSS/POS/SER. These gaps suggest that, for many POS/SER non-ARA F3 pairs, the required positive-AM witness comes from another skip-induced anti-dependency on the witness cycle, consistent with the ARA-backed return paths in Section 5.4.

The example in Figure 1 illustrates this cycle-level role of AM. The non-ARA pair (R_N, W_N) may have small or non-positive AM, yet it becomes cycle-forming through a return path that relies on the ARA pair (R_A, W_A). If W_A completed well before R_A starts while remote participant S_2 still treats W_A as prepared, as in Section 5.5, the ARA pair can supply the larger positive-AM witness for the cycle.

5.7 Implications

The main design lesson is to identify which ordering constraints limit the extent to which skips produce SS-relative stale-read anomalies. Although RSS and POS are not currently widely deployed as database consistency models, the ordering constraints they preserve or relax—such as regularity, process order, and real-time order—are general semantic building blocks that can be combined in future models and protocols. Our controlled RSS/POS/SER family lets us study which ordering constraint matters. In our traces, POS produces fewer F3 pairs and F3 ROs than SER, suggesting that process order helps limit F3 amplification. However, POS still admits ARA pairs and ARA-backed return paths, so process order does not eliminate the dominant amplification mechanism. RSS suggests a useful design point for Spanner-like RO protocols: preserving regularity rules out ARA pairs and keeps both F3-pair and F3-RO counts far lower than in POS/SER. Furthermore, although RSS relaxes RO–RO and non-conflicting RW–RO real-time order, these relaxations alone do not appear sufficient to produce such amplification.

This suggests that selective regularity could be an interesting future protocol direction. For ROs selected based on their issuing users, client regions, or accessed keys, a system could preserve regularity while using weaker wait rules for other ROs. For example, building on consistency-based service-level agreements [20], an application could express consistency as a service-level requirement and require consistency-sensitive ROs to preserve regularity. For such a selected RO R, regularity prevents R from skipping a conflicting RW W that precedes R in real time and returning a snapshot that omits W's write, thereby preventing the corresponding ARA

edge from R to W. Further, ruling out such ARA edges may also reduce stale-read amplification by preventing other non-ARA pairs from becoming cycle-forming.

6 RELATED WORK

Empirical anomaly measurement. Prior work has measured the practical cost of weaker consistency. Lu et al. [15] quantify anomalies in Facebook's eventually consistent TAO by analyzing sampled production traces with respect to stronger local consistency models. Their study asks how often one deployed system's observed executions are disallowed by stronger local consistency models for non-transactional operations; our work instead studies transactional, non-local SS-relative anomalies across a controlled family of Spanner-like RO protocols targeting RSS, POS, and SER.

In our prior work [9], we studied anomaly rarity within Spanner-RSS. An RSS-specific cascade of necessary conditions for cycle formation explained how workload skew, RTT regime, and replica placement affect the rate at which SS-relative anomalies arise. The present work asks a different question across consistency models: how does weakening the RO wait condition beyond RSS—and thereby relaxing additional ordering constraints—amplify SS-relative stale-read anomalies? Using the controlled RSS/POS/SER family, we identify ARA-backed graph structure—direct cycles formed by ARA pairs and return paths for non-ARA pairs supported by ARA edges—as the structural primitive behind this amplification.

Black-box consistency checking. Elle [10] infers Adya-style dependency graphs from client-observed histories using workloads in which reads expose enough version-history information, while Cobra [18] provides scalable black-box serializability verification for transactional key-value stores.

Our goal is different: we do not build a general-purpose black-box checker. Our analysis is deliberately white-box, using protocol traces that expose prepared RWs, skip decisions, timestamps, and the coordinator and participant roles of shards. This lets us attribute SS-relative violation cycles to specific RO–RW skip pairs, classify ARA and non-ARA structures, and connect the formation of ARA pairs to cross-region visibility lag.

Spanner snapshot reads and Spanner-RO-SER. Along with strictly serializable read-only transactions, Spanner [4] provides snapshot reads: they execute without locks at a client-chosen timestamp or at a timestamp selected to satisfy a client-specified staleness bound. Each snapshot read returns a consistent snapshot and can be serialized at that timestamp, but need not respect the real-time order required by SS.

Spanner-RO-SER is not an implementation of this snapshot-read API. Instead, it is a controlled Spanner-RSS-derived RO variant used to isolate the effect of weakening the RO wait condition. It retains the common Spanner-like RO substrate—the RW protocol, RO read timestamp choice, Paxos safe-time check, and snapshot assembly—and changes only the wait rule: it skips every prepared conflicting RW that an SS RO would wait for. It serves as the weakest RO-wait endpoint in our RSS/POS/SER comparison.

7 CONCLUSION

We diagnose SS-relative stale-read anomalies in a controlled family of Spanner-like RO protocols targeting RSS, POS, and SER. In POS/SER, stale-read amplification is not explained by skip-induced anti-dependency volume alone, but by ARA-backed graph structure: ARA pairs form cycles directly, while ARA edges support return paths for many non-ARA pairs. We associate the formation of ARA pairs with cross-region visibility lag at remote participants. RSS prevents this structure by preserving regularity.

Within our controlled family and traces, our findings isolate the ordering constraint that matters for stale-read amplification: process order alone is insufficient, whereas regularity rules out ARA pairs and suppresses the dominant ARA-backed mechanism. This insight may motivate new consistency models and transaction protocols, including selective-regularity designs.

ACKNOWLEDGMENTS

This work was supported in part by the Institute of Information & Communications Technology Planning & Evaluation (IITP)-ICT Creative Consilience Program grant funded by the Korea government (MSIT) (IITP-2026-RS-2020-II201819, RS-2026-25522620); and in part by the Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (RS-2021-NR060143).

REFERENCES

[1] Atul Adya. 1999. *Weak consistency: a generalized theory and optimistic implementations for distributed transactions.* Ph.D. Dissertation. Massachusetts Institute of Technology.
[2] Amazon Web Services. 2026. Amazon Web Services. https://aws.amazon.com/. Accessed: 2026-07-16.
[3] Philip A Bernstein, Vassos Hadzilacos, and Nathan Goodman. 1986. *Concurrency control and recovery in database systems.* Addison-Wesley Longman Publishing Co., Inc., USA.
[4] James C. Corbett, Jeffrey Dean, Michael Epstein, Andrew Fikes, Christopher Frost, J. J. Furman, Sanjay Ghemawat, Andrey Gubarev, Christopher Heiser, Peter Hochschild, Wilson Hsieh, Sebastian Kanthak, Eugene Kogan, Hongyi Li, Alexander Lloyd, Sergey Melnik, David Mwaura, David Nagle, Sean Quinlan, Rajesh Rao, Lindsay Rolig, Yasushi Saito, Michal Szymaniak, Christopher Taylor, Ruth Wang, and Dale Woodford. 2012. Spanner: Google's globally-distributed database. In *Proceedings of the 10th USENIX Conference on Operating Systems Design and Implementation* (Hollywood, CA, USA) *(OSDI'12)*. USENIX Association, USA, 251–264.
[5] Jim Gray. 1978. Notes on Data Base Operating Systems. In *Operating Systems, An Advanced Course*. Springer-Verlag, Berlin, Heidelberg, 393–481.
[6] Jeffrey Helt, Matthew Burke, Amit Levy, and Wyatt Lloyd. 2021. Regular sequential serializability and regular sequential consistency. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*. 163–179.
[7] Jeffrey Helt, Matthew Burke, Amit Levy, and Wyatt Lloyd. 2021. Regular Sequential Serializability and Regular Sequential Consistency. arXiv:2109.08930 [cs.DB] https://arxiv.org/abs/2109.08930
[8] Bert Hubert. [n.d.]. tc(8), Linux manual page. https://man7.org/linux/man-pages/man8/tc.8.html. Accessed: 2026-07-16.
[9] Sejong Kim and Yon Dohn Chung. 2026. Why Are Anomalies Rare in RSS? Diagnosing Strict Serializability Violations in Spanner-RSS. In *2026 56th Annual IEEE International Conference on Dependable Systems and Networks—Supplemental Volume (DSN-S)*. IEEE, 260–261. https://doi.org/10.1109/DSN-S70715.2026.00067
[10] Kyle Kingsbury and Peter Alvaro. 2020. Elle: inferring isolation anomalies from experimental observations. *Proceedings of the VLDB Endowment* 14, 3 (2020), 268–280.
[11] Leslie Lamport. 1986. On interprocess communication: Part ii: Algorithms. *Distributed Computing* 1, 2 (1986), 86–101.
[12] Leslie Lamport. 2019. The part-time parliament. In *Concurrency: The Works of Leslie Lamport*. 277–317.
[13] Costin Leau. 2013. Spring Data Redis-Retwis-J. https://docs.spring.io/spring-data/data-keyvalue/examples/retwisj/current/. Accessed: 2026-07-16.

[14] Haonan Lu, Christopher Hodsdon, Khiem Ngo, Shuai Mu, and Wyatt Lloyd. 2016. The SNOW Theorem and Latency-Optimal Read-Only Transactions. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. 135–150.
[15] Haonan Lu, Kaushik Veeraraghavan, Philippe Ajoux, Jim Hunt, Yee Jiun Song, Wendy Tobagus, Sanjeev Kumar, and Wyatt Lloyd. 2015. Existential consistency: Measuring and understanding consistency at Facebook. In *Proceedings of the 25th Symposium on Operating Systems Principles*. 295–310.
[16] Christos H Papadimitriou. 1979. The serializability of concurrent database updates. *Journal of the ACM (JACM)* 26, 4 (1979), 631–653.
[17] Cheng Tan, Lingfan Yu, Joshua B Leners, and Michael Walfish. 2017. The efficient server audit problem, deduplicated re-execution, and the web. In *Proceedings of the 26th Symposium on Operating Systems Principles*. 546–564.
[18] Cheng Tan, Changgeng Zhao, Shuai Mu, and Michael Walfish. 2020. Cobra: Making transactional key-value stores verifiably serializable. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. 63–80.
[19] Robert Tarjan. 1972. Depth-first search and linear graph algorithms. *SIAM journal on computing* 1, 2 (1972), 146–160.
[20] Douglas B Terry, Vijayan Prabhakaran, Ramakrishna Kotla, Mahesh Balakrishnan, Marcos K Aguilera, and Hussam Abu-Libdeh. 2013. Consistency-based service level agreements for cloud storage. In *Proceedings of the twenty-fourth ACM symposium on operating systems principles*. 309–324.
[21] Hiroshi Wada, Alan D Fekete, Liang Zhao, Kevin Lee, and Anna Liu. 2011. Data Consistency Properties and the Trade-offs in Commercial Cloud Storage: the Consumers' Perspective. In *CIDR*, Vol. 11. 134–143.

A SENSITIVITY ANALYSIS ACROSS SKEW, PLACEMENT, RTT, AND RO/RW MIX

This appendix repeats the analysis under four sensitivity settings, each changing one workload or deployment dimension from the main configuration: lower skew, global replica placement, intra-continent RTT, and an RO-heavy workload mix. All counts and percentages reported in this appendix are independently rounded to the displayed precision.

Takeaway. Across lower skew, global replica placement, intra-continent RTT, and the RO-heavy workload mix, the volume and composition of F3 pairs change substantially. Lower skew reduces skip opportunities; intra-continent RTT shrinks the AM scale and reduces F3 survival; and global replica placement and the RO-heavy mix substantially increase non-ARA F2-to-F3 survival. Despite these differences, the same structural distinction persists: POS/SER show F3 amplification over RSS; most non-ARA return paths in POS/SER remain dependent on ARA edges; and RSS suppresses amplification by ruling out ARA pairs. Thus, the sensitivity results reinforce the design implications discussed in Section 5.7.

Experimental setup. For each protocol, we run the Retwis [13] workload for 600 s, including a 120 s warm-up period. We run experiments on six machines (64 cores, 327 GB RAM, 10 Gbps NIC each), grouped into three emulated regions with two machines per region. We use 3 shards with 3 replicas per shard. For each workload setting, we first measure the peak throughput of all three protocols and set the operating point for each protocol at approximately 70% of its measured peak; because their peak throughputs are similar, we use the same client count across protocols. We evaluate five settings. The baseline uses Zipfian key-access skew 0.9, inter-continent RTTs, and local replica placement: each shard's replicas are co-located within one region, while different shards are placed in different regions. Zipf-0.7 changes only the key-access skew to 0.7. The global setting changes only replica placement: each shard's replicas are distributed across regions, with the leaders of different shards placed in distinct regions. The intra setting changes only the RTT regime to the intra-continent setting in Table 3. The RO-90 setting uses the baseline skew, placement, and RTT regime, but changes the configured RO/RW transaction-selection ratio from 50:50 to 90:10. We do not model message passing in this workload.

Table 3: Wide-area p50 RTTs (ms) measured on AWS [2]. Regions are abbreviated as follows: FFM (Frankfurt), ZH (Zurich), PAR (Paris), VR (Virginia), IR (Ireland), CA (California).

	FFM	ZH	PAR
FFM	0.4		
ZH	10	0.4	
PAR	12	12	0.4

(a) Intra-continent

	VR	IR	CA
VR	0.4		
IR	70	0.4	
CA	64	130	0.4

(b) Inter-continent

Table 3 reports the intra-/inter-continent p50 RTT settings between regions, emulated via Linux TC [8]. The measured clock uncertainty used for RT extraction is below 100 μs. Separately, the emulated TrueTime uncertainty used by the protocol is 4 ms, matching Spanner's reported average under normal conditions [4].

The experiment settings process different numbers of transactions, so raw counts should be compared primarily within the same setting. Within each setting, however, RSS, POS, and SER process similar numbers of ROs: 1.613–1.617M in baseline, 2.333–2.337M in Zipf-0.7, 1.420–1.424M in global, 2.364–2.366M in intra, and 8.065–8.128M in RO-90.

Limitations. Different transaction shapes, access correlations, shard counts, and forms of application-level causality may change the magnitude and composition of cycle-forming pairs. Evaluating these effects under broader workloads and in deployed systems remains future work.

Sanity checks. All settings pass three sanity checks: (1) the DSG alone has no non-trivial SCCs; (2) removing all F2 skip-induced anti-dependency edges removes all non-trivial SCCs from G; and (3) every detected skip-induced cycle contains at least one skip-induced anti-dependency pair with positive AM, as required by Theorem C.4.

F2-to-F3 amplification across settings. Table 4 shows that, within each setting, POS/SER F3 amplification is much larger than the increase in skip-induced anti-dependency volume. In baseline, POS/SER produce only 1.5/2.0× as many F2 pairs as RSS, but 181/359× as many F3 pairs. The same pattern holds in global and RO-90. In global, POS/SER produce only 1.2/1.5× as many F2 pairs as RSS, but 289/648× as many F3 pairs; in RO-90, POS/SER produce only 1.6/1.7× as many F2 pairs as RSS, but 165/198× as many F3 pairs. Under Zipf-0.7, RSS produces almost no F3 pairs, while POS/SER still produce thousands of F3 pairs. This pair-level F3 amplification also appears at the RO level: across all settings, no more than 0.07% of RSS ROs appear as R in an F3 pair, whereas the corresponding fractions reach 13.9/29.2% in POS/SER.

In the intra-continent setting, RSS skip volume drops sharply relative to POS/SER, so POS/SER have 21.9/26.7× as many F2 pairs as RSS. This is consistent with RSS's shard-side t_{ee}-based regularity guard: under the smaller RTT scale, the $t_{\text{ee},s}(W) > t_{\text{read}}(R)$ skip clause is satisfied less often, making RSS skips much rarer. Even

Table 4: Sensitivity of the skip-to-cycle cascade across settings. Skip is the fraction of ROs that appear in at least one F1 pair. F1–F3 count RO–RW pairs; parentheses in the F2 and F3 columns show survival from the previous filter. The F3 ROs column reports the number of distinct ROs that appear as R in at least one F3 pair; parentheses show their fraction of all ROs. Raw counts should be compared primarily within each setting because trace sizes differ across settings.

Setting	Model	Skip	F1	F2	F3	F3 ROs
baseline	RSS	11.4%	197.8K	158.0K (79.8%)	334 (0.2%)	333 (0.02%)
	POS	15.8%	283.7K	232.5K (82.0%)	60.4K (26.0%)	57.6K (3.6%)
	SER	21.4%	390.1K	323.4K (82.9%)	119.8K (37.0%)	112.5K (7.0%)
Zipf-0.7	RSS	1.5%	35.7K	32.5K (91.1%)	4 (0.01%)	4 (< 0.001%)
	POS	2.0%	47.7K	43.9K (92.1%)	6.6K (15.1%)	6.6K (0.3%)
	SER	3.2%	77.3K	71.4K (92.3%)	18.1K (25.3%)	17.9K (0.8%)
global	RSS	28.0%	474.5K	414.7K (87.4%)	789 (0.2%)	781 (0.06%)
	POS	31.4%	548.6K	485.7K (88.5%)	228.4K (47.0%)	197.6K (13.9%)
	SER	38.8%	706.2K	632.9K (89.6%)	511.3K (80.8%)	415.2K (29.2%)
intra	RSS	1.2%	27.8K	20.5K (73.9%)	0 (0.0%)	0 (0%)
	POS	19.4%	513.9K	449.6K (87.5%)	27.0K (6.0%)	26.3K (1.1%)
	SER	23.0%	620.2K	547.8K (88.3%)	51.5K (9.4%)	50.1K (2.1%)
RO-90	RSS	11.8%	1.018M	827.1K (81.3%)	5.2K (0.6%)	5.2K (0.07%)
	POS	17.2%	1.542M	1.286M (83.4%)	866.2K (67.4%)	804.2K (9.9%)
	SER	19.0%	1.718M	1.435M (83.5%)	1.039M (72.4%)	957.8K (11.8%)

Table 5: ARA composition and ARA-edge dependence across settings. Entries under ARA/Non-ARA Share report ARA%/Non-ARA% within F2 or F3. Non-ARA F2→F3 is the survival rate of non-ARA F2 pairs. ARA-dep. F3 is the fraction of non-ARA F3 pairs that are no longer cycle-forming after removing all ARA edges. Dashes denote not applicable.

Setting	Model	ARA/Non-ARA Share		Non-ARA Behavior	
		of F2	of F3	F2→F3	ARA-dep. F3
baseline	RSS	0.0%/100.0%	0.0%/100.0%	0.2%	0.0%
	POS	12.4%/87.6%	47.7%/52.3%	15.5%	98.3%
	SER	14.6%/85.4%	39.5%/60.5%	26.3%	98.7%
Zipf-0.7	RSS	0.0%/100.0%	0.0%/100.0%	0.01%	0.0%
	POS	12.1%/87.9%	80.2%/19.8%	3.4%	99.7%
	SER	18.0%/82.0%	71.1%/28.9%	8.9%	99.9%
global	RSS	0.0%/100.0%	0.0%/100.0%	0.2%	0.0%
	POS	9.3%/90.7%	19.9%/80.1%	41.5%	99.7%
	SER	14.8%/85.2%	18.3%/81.7%	77.5%	99.8%
intra	RSS	0.0%/100.0%	–	0.0%	–
	POS	4.9%/95.1%	81.5%/18.5%	1.2%	79.2%
	SER	7.5%/92.5%	79.8%/20.2%	2.1%	91.3%
RO-90	RSS	0.0%/100.0%	0.0%/100.0%	0.6%	0.0%
	POS	14.1%/85.9%	20.9%/79.1%	62.0%	98.6%
	SER	15.1%/84.9%	20.9%/79.1%	67.5%	98.4%

accounting for this larger F2 gap, however, POS/SER still exhibit non-negligible F3 formation: tens of thousands of their F2 pairs become F3, while RSS produces no F3 pairs in this run.

Table 6: Client-shard co-location and RTTs for ARA observations. C_W is the client of the skipped RW W, S_C is W's coordinator shard, S_P is the shard where RO R skips W, and C_R is R's client. In all ARA observations, S_P is a participant shard of W, not its coordinator. Co-loc. reports the fraction of observations in which the client is co-located with the corresponding shard; RTT columns report median RTTs.

Setting	Model	C_W–S_C		S_C–S_P	C_R–S_P	
		Co-loc.	RTT	RTT	Co-loc.	RTT
baseline	POS	97.4%	<1 ms	130 ms	77.7%	<1 ms
	SER	97.4%	<1 ms	130 ms	76.7%	<1 ms
Zipf-0.7	POS	97.0%	<1 ms	130 ms	81.5%	<1 ms
	SER	97.2%	<1 ms	130 ms	78.1%	<1 ms
global	POS	72.2%	<1 ms	70 ms	46.9%	64 ms
	SER	71.4%	<1 ms	70 ms	47.2%	64 ms
intra	POS	98.0%	<1 ms	12 ms	81.9%	<1 ms
	SER	97.6%	<1 ms	12 ms	82.9%	<1 ms
RO-90	POS	97.7%	<1 ms	130 ms	77.9%	<1 ms
	SER	97.8%	<1 ms	130 ms	77.0%	<1 ms

ARA share varies, but ARA-edge dependence persists. Table 5 shows that F2 pairs are mostly non-ARA across all POS/SER settings. ARA accounts for only 4.9–18.0% of F2 pairs, depending on the setting and model. Because ARA pairs are cycle-forming by construction, however, their share can become much larger at F3. In Zipf-0.7 and intra, ARA accounts for 71.1–81.5% of POS/SER F3 pairs, not because non-ARA F2 pairs are rare, but because non-ARA F2 pairs rarely become F3: only 3.4%/8.9% in Zipf-0.7 and 1.2%/2.1% in intra.

The contrast between Zipf-0.7 and intra is informative. Zipf-0.7 reduces the number of dependency opportunities while retaining the inter-continent RTT regime; the non-ARA F3 pairs that remain are almost entirely ARA-dependent: removing ARA edges leaves 99.7%/99.9% of POS/SER non-ARA F3 pairs without a return path. Intra keeps the higher skew but reduces the RTT scale; there, non-ARA F2→F3 survival is lower, and ARA removal leaves a smaller, but still majority, fraction of non-ARA F3 pairs without a return path: 79.2%/91.3% in POS/SER.

At the opposite extreme are the global and RO-90 settings. ARA accounts for only 18.3–20.9% of POS/SER F3 because non-ARA F2 pairs survive to F3 much more often (41.5–77.5%). Yet removing ARA edges leaves 98.4–99.8% of non-ARA F3 pairs without a return path. Thus, a small direct ARA share does not imply that ARA edges are unimportant; in both settings, ARA's effect appears primarily through its support for non-ARA return paths.

ARA pairs arise from participant-side visibility lag. Table 6 characterizes where ARA pairs arise. Across all settings with ARA pairs, the RO skips the RW at a participant shard rather than at the RW's coordinator. This supports the visibility-lag interpretation: the RW has completed through its coordinator while a participant still treats it as prepared.

Baseline, Zipf-0.7, intra, and RO-90 show the same client-shard distance pattern: the RW client is almost always co-located with the RW coordinator, and the RO client is often co-located with

Table 7: AM diagnostics across settings. All entries are median AM values in milliseconds. The ARA F3, Non-ARA F2, and Non-ARA F3 columns report pair-level AM medians. Witness Max reports, over non-ARA F3 pairs, the median of the maximum AM among skip-induced anti-dependency edges on the chosen shortest-hop logical witness cycle. Dashes denote not applicable.

Setting	Model	ARA	Non-ARA		
		F3	F2	F3	Witness Max
baseline	RSS	–	−34.0	30.3	31.1
	POS	23.8	−33.5	−9.6	50.8
	SER	24.0	−33.0	−13.0	51.6
Zipf-0.7	RSS	–	−34.6	23.1	23.1
	POS	24.4	−31.0	−5.5	48.2
	SER	24.5	−27.9	−5.9	47.9
global	RSS	–	−4.2	99.3	114.9
	POS	108.6	−3.2	13.1	155.7
	SER	111.8	−0.3	5.5	158.3
intra	RSS	–	−9.3	–	–
	POS	6.7	−3.5	2.8	19.3
	SER	7.0	−3.0	2.8	10.2
RO-90	RSS	–	−33.8	30.4	31.5
	POS	23.8	−32.6	−23.8	54.0
	SER	23.9	−32.7	−25.4	54.8

the participant where the skip occurs. In baseline, Zipf-0.7, and RO-90, which use the inter-continent RTT regime, the coordinator-to-participant RTT is 130 ms, the largest RTT in those settings; intra preserves the same co-location pattern at a 12 ms coordinator-to-participant RTT.

Global replica placement exhibits a different client-shard distance pattern: RW-client co-location with the coordinator drops to about 71–72%, RO-client co-location with the participant drops to about 47%, and the RO-client-to-participant RTT rises to 64 ms. Thus, neither strict client-shard co-location nor the largest coordinator-to-participant RTT is necessary for an RO to observe an RW that has already completed through its coordinator as still prepared. This pattern is consistent with additional cross-region replica coordination lengthening the participant-side visibility-lag window before the commit decision is applied.

AM reflects RTT and replica-placement trends. Table 7 reports pair-level AM values and witness-level maximum AM values across settings. The ARA and witness-max columns most directly reflect the RTT and placement trends. Baseline, Zipf-0.7, and RO-90, which share the same inter-continent RTT regime and local replica placement, have similar ARA F3 median AM values around 24 ms and similar POS/SER non-ARA witness-max AM values around 48–55 ms. Intra has a smaller AM scale: ARA F3 median AM is only 6.7–7.0 ms, and the median non-ARA witness max AM is 10.2–19.3 ms. Global shifts these values upward: ARA F3 median AM rises to about 109–112 ms, and non-ARA witness max AM rises to about 155–158 ms in POS/SER.

The non-ARA pair-level columns provide a complementary view. Non-ARA F2 medians remain negative in all settings, and non-ARA F3 medians shift upward relative to non-ARA F2 but are still negative or much smaller than the witness maximum in many POS/SER settings. This suggests that many non-ARA F3 pairs obtain their positive-AM witness from a different skip-induced anti-dependency edge with a larger AM. Together with Table 5, this pattern indicates that their return paths are typically ARA-backed in POS/SER. In contrast, RSS has no ARA pairs, so its rare non-ARA F3 pairs have F3 AM and witness-max AM at similar scales.

B CORRECTNESS OF SPANNER-RO-POS AND SPANNER-RO-SER

We reuse the timestamp-based proof structure of Appendix D of the Spanner-RSS technical report [7]. Our proof sketches identify the arguments that remain unchanged and isolate the model-specific changes for Spanner-RO-POS and Spanner-RO-SER.

Consistent snapshot assembly and snapshot timestamp selection. Skipping prepared RWs independently at different shards can make the initial shard replies fail to form a consistent snapshot. For example, one shard may return a version written by a multi-shard RW W, while another shard skips W because W is still prepared there. To assemble a consistent snapshot, Spanner-RO-POS and Spanner-RO-SER use the same fast/slow reply mechanism as Spanner-RSS.

Each shard first returns a fast reply with committed versions for the requested keys and metadata for skipped prepared RWs. Once all fast replies have arrived, the client computes a fixed snapshot timestamp $t_{\text{snap}}(R)$ from the candidate versions accumulated so far, including any versions obtained from slow replies already received. Following the Spanner-RSS CalculateSnapshotTS rule, the client takes the earliest commit timestamp for each requested key among the candidate versions available for that key, and then takes the maximum of these per-key timestamps. The client keeps $t_{\text{snap}}(R)$ fixed while waiting for slow replies from skipped RWs that could still commit at or before $t_{\text{snap}}(R)$. A slow reply reports whether the skipped RW commits and, if so, its commit timestamp and value. The client incorporates the write if it commits at or before $t_{\text{snap}}(R)$, and repeats until no remaining skipped RW can affect the snapshot.

Serialization timestamp assignment. For an RW W, let $\tau(W) = t_{\text{c}}(W)$. For an RO R, let $t_{\min}(R)$ be the lower bound carried in R's request. For Spanner-RO-POS, this is the client process-order lower bound; for Spanner-RO-SER, it is 0. We assign

$$\tau(R) = \max\left(t_{\text{snap}}(R), t_{\min}(R)\right).$$

Let $t_{\text{invoc}}(T)$ and $t_{\text{resp}}(T)$ denote the invocation and response times of a completed transaction T.

Lemma B.1. *In Spanner-RO-POS and Spanner-RO-SER, suppose T_2 is an RW with serialization timestamp $\tau(T_2)$. For each key read by T_2, the returned value is written by the committed RW T_1 with the greatest serialization timestamp $\tau(T_1)$ such that $\tau(T_1) < \tau(T_2)$.*

Proof sketch. We reuse Lemma D.3 of the Spanner-RSS technical report [7]. The RW transaction protocol is unchanged from Spanner-RSS and is nearly identical to Spanner's RW protocol. Thus, the lemma follows from the Spanner-RSS argument, which relies on the correctness of strict two-phase locking and Spanner's timestamp assignment. □

Lemma B.2. *In Spanner-RO-POS and Spanner-RO-SER, suppose T_2 is an RO with serialization timestamp $\tau(T_2)$ as defined above. For each key read by T_2, the returned value is written by the committed RW T_1 with the greatest serialization timestamp $\tau(T_1)$ such that $\tau(T_1) \leq \tau(T_2)$.*

Proof sketch. If $\tau(T_2) = t_{\text{snap}}(T_2)$, this is exactly the Spanner-RSS snapshot assembly argument in Lemma D.4 of the Spanner-RSS technical report [7]: any skipped prepared RW that could commit at or before $t_{\text{snap}}(T_2)$ is resolved before the final snapshot is returned.

Now suppose $\tau(T_2) = t_{\text{min}}(T_2) > t_{\text{snap}}(T_2)$; this case can occur only for Spanner-RO-POS. Because $t_{\text{min}}(T_2)$ is carried from transactions that completed before T_2 starts, and $t_{\text{read}}(T_2) = \text{TT.NOW}().\mathit{latest}$ is chosen when T_2 starts, we have $t_{\text{min}}(T_2) \leq t_{\text{read}}(T_2)$. We show that no read key has a missing committed write with timestamp in $(t_{\text{snap}}(T_2), t_{\text{min}}(T_2)]$.

Let U be a committed RW that performs such a missing write to a key read by T_2. First suppose U had already committed at the shard when T_2's fast reply was produced. Because the shard has passed the Paxos safe-time check and returns the latest committed version at $t_{\text{read}}(T_2)$, the shard's fast/slow replies received before snapshot selection contain only committed versions whose commit timestamps are at least $t_c(U)$. These versions constitute the candidates used for that key by CalculateSnapshotTS; hence the per-key timestamp for that key is at least $t_c(U)$. Since CalculateSnapshotTS takes the maximum over per-key timestamps, this would force $t_{\text{snap}}(T_2) \geq t_c(U)$, contradicting the assumption that $t_c(U) > t_{\text{snap}}(T_2)$.

Next suppose U was prepared but not yet committed at that shard when the wait/skip decision was evaluated. Since $t_c(U) \leq t_{\text{min}}(T_2)$ and $t_{p,s}(U) \leq t_c(U)$, we have $t_{p,s}(U) \leq t_{\text{min}}(T_2)$. Under Spanner-RO-POS, such a prepared conflicting RW cannot be skipped; the RO waits for it. If U commits, then because $t_c(U) \leq t_{\text{min}}(T_2) \leq t_{\text{read}}(T_2)$, the version returned in the fast reply after waiting reflects U or a later committed write. As in the previous case, this forces $t_{\text{snap}}(T_2) \geq t_c(U)$. If U aborts, it is not a committed missing write. Therefore, any prepared RW X that is skipped and later commits has $t_{p,s}(X) > t_{\text{min}}(T_2)$, and since $t_c(X) \geq t_{p,s}(X)$, it cannot commit at or before $\tau(T_2) = t_{\text{min}}(T_2)$.

Finally, if U had not yet prepared at that shard when T_2's fast reply was produced, the Paxos safe-time check implies that its future prepare timestamp is greater than $t_{\text{read}}(T_2)$. Therefore, since $t_{\text{min}}(T_2) \leq t_{\text{read}}(T_2)$, such a write cannot commit at or before $t_{\text{min}}(T_2)$.

Thus, for the keys read by T_2, the interval $(t_{\text{snap}}(T_2), t_{\text{min}}(T_2)]$ contains no missing committed write. Consequently, the snapshot returned at $t_{\text{snap}}(T_2)$ is also legal at the serialization timestamp $\tau(T_2) = t_{\text{min}}(T_2)$. □

Lemma B.3. *For every completed transaction T,*

$$\tau(T) < t_{\text{resp}}(T).$$

Proof sketch. For an RW, commit-wait ensures that $t_c(T)$ has already passed before the transaction responds. For an RO R, if $t_{\text{snap}}(R) > 0$, then by CalculateSnapshotTS $t_{\text{snap}}(R)$ is the commit timestamp of some candidate version available to the RO client. That version comes from a committed RW whose commit timestamp has passed before R returns; if $t_{\text{snap}}(R) = 0$, the claim is immediate. The incoming $t_{\text{min}}(R)$ was produced by completed earlier transactions at the same client, so by induction it is smaller than R's invocation time. Thus both $t_{\text{snap}}(R)$ and $t_{\text{min}}(R)$ are smaller than $t_{\text{resp}}(R)$, and hence so is $\tau(R)$. □

Lemma B.4. *In Spanner-RO-POS, if completed transactions T_1 and T_2 are issued by the same client and T_1 completes before T_2 starts, then $\tau(T_1) \leq \tau(T_2)$. Furthermore, if both T_1 and T_2 are RW transactions, then $\tau(T_1) < \tau(T_2)$.*

Proof sketch. This is the process-order version of Lemma D.1 in the Spanner-RSS technical report [7]. Spanner-RO-POS uses t_{min} to carry only the client's process-order lower bound. When a client ends a transaction T_1, it advances t_{min} to at least $\tau(T_1)$ and sends this bound with later ROs.

First suppose that T_2 is an RO. Since T_2 is issued after T_1 ends, the lower bound carried by T_2 is at least $\tau(T_1)$. By definition,

$$\tau(T_2) = \max\left(t_{\text{snap}}(T_2), t_{\text{min}}(T_2)\right) \geq t_{\text{min}}(T_2) \geq \tau(T_1).$$

Now suppose that T_2 is an RW. Since T_1 completes before T_2 starts, Lemma B.3 gives $\tau(T_1) < t_{\text{resp}}(T_1) < t_{\text{invoc}}(T_2)$. Spanner's RW timestamp assignment chooses $t_c(T_2)$ after T_2's start, so $\tau(T_1) < \tau(T_2)$.

□

Theorem B.5. *Spanner-RO-POS guarantees process-ordered serializability.*

Proof sketch. We reuse the structure of Theorem D.5 of the Spanner-RSS technical report [7]. We first construct a strict total order $\prec$ over completed transactions using the timestamp τ. First, transactions are ordered by increasing $\tau(T)$. Second, for each timestamp, we form a tie graph over transactions with the same timestamp, using causal-order edges between them. Causal edges are used here only as a tie-breaking rule; POS requires only process-order preservation. Since causal order is acyclic, each tie graph is acyclic. We topologically sort each tie graph, breaking remaining ties arbitrarily.

Note that conflicting RWs receive distinct commit timestamps, so two RWs that write the same key cannot tie at τ. Thus, same-timestamp ties do not create ambiguity about the latest RW writer for any key.

Let S be the sequential history obtained by ordering completed transactions according to $\prec$. We first show that S is legal. If T is an RW, Lemma B.1 shows that each value read by T is written by the greatest-timestamp RW before $\tau(T)$, which is the latest preceding writer in S. If T is an RO, Lemma B.2 shows that each returned value is written by the greatest-timestamp RW at or before $\tau(T)$. If the writer's timestamp is smaller than $\tau(T)$, it precedes T by timestamp order; if the timestamps are equal, the read-from edge is a causal edge in the tie graph, so the topological order places the writer before T. Thus, S is legal.

It remains to show that S respects process order. Consider completed transactions T_1 and T_2 issued by the same client, where T_1 completes before T_2 starts. By Lemma B.4, $\tau(T_1) \leq \tau(T_2)$. If $\tau(T_1) < \tau(T_2)$, then $T_1 \prec T_2$ by timestamp order. Otherwise, $\tau(T_1) = \tau(T_2)$; since process order is included in causal order, the process-order edge from T_1 to T_2 appears in the same timestamp tie graph, and the

topological sort places T_1 before T_2. Hence S respects each client's process order.

Therefore, the execution has a legal serial history that respects process order, so Spanner-RO-POS satisfies POS. □

Theorem B.6. *Spanner-RO-SER guarantees serializability.*

Proof sketch. The same timestamp-order construction used in Theorem B.5, together with Lemmas B.1 and B.2, gives a strict total order and a legal sequential history. Since SER imposes no order constraint beyond the existence of such a history, Spanner-RO-SER satisfies serializability. □

C CORRECTNESS OF POSITIVE CYCLE-LEVEL ANTI-DEPENDENCY MARGIN

For the Spanner-like RO protocol family, we prove that every skip-induced SS-relative violation cycle contains at least one skip-induced RO–RW anti-dependency pair with positive AM.

Dependency timestamp assignment. For the AM proof, we use a dependency timestamp σ, distinct from the serialization timestamp used in Appendix B. For an RW T, let $\sigma(T) = t_c(T)$; for an RO T, let $\sigma(T) = t_{snap}(T)$, the fixed snapshot timestamp used by the final snapshot assembly.

Trace-time bounds and real-time relation. Let $\widehat{t}_{invoc}(T)$ and $\widehat{t}_{resp}(T)$ be the raw invocation and response timestamps recorded in the trace. Let $t^{abs}_{invoc}(T)$ and $t^{abs}_{resp}(T)$ denote the corresponding unobserved absolute event times. We assume the trace-clock error is bounded by ε, so

$$\left|\widehat{t}_{invoc}(T) - t^{abs}_{invoc}(T)\right| \le \varepsilon, \qquad \left|\widehat{t}_{resp}(T) - t^{abs}_{resp}(T)\right| \le \varepsilon.$$

Accordingly, define the conservative event-time bounds

$$\underline{t}_{invoc}(T) = \widehat{t}_{invoc}(T) - \varepsilon, \qquad \overline{t}_{resp}(T) = \widehat{t}_{resp}(T) + \varepsilon.$$

Then

$$\underline{t}_{invoc}(T) \le t^{abs}_{invoc}(T), \qquad t^{abs}_{resp}(T) \le \overline{t}_{resp}(T).$$

In this appendix, RT denotes the full conservative real-time relation:

$$A \xrightarrow{RT} B \quad \text{iff} \quad \overline{t}_{resp}(A) < \underline{t}_{invoc}(B).$$

Equivalently, in terms of raw trace timestamps, this is

$$\widehat{t}_{resp}(A) < \widehat{t}_{invoc}(B) - 2\varepsilon.$$

In Section 4, RT^* denotes the reachability relation induced by the reduced RT graph, which is equivalent to the full conservative RT relation defined above. The proof below reasons about the latter directly.

Definition of AM. For a skip-induced anti-dependency pair (R, W), define

$$AM(R, W) = \underline{t}_{invoc}(R) - t_c(W).$$

A pair has positive AM when this quantity is positive.

Lemma C.1. *For every completed RO R, and for each key read by R, the returned value is written by the committed RW with the greatest commit timestamp $t_c(W)$ such that $t_c(W) \le \sigma(R) = t_{snap}(R)$.*

Proof sketch. This is the Spanner-RSS snapshot-assembly property from Lemma D.4 of the Spanner-RSS technical report [7], restated with $\sigma(R) = t_{snap}(R)$. All variants reuse the same fast/slow snapshot assembly: the client fixes $t_{snap}(R)$, waits for any skipped prepared RW that could still commit at or before this timestamp, incorporates such writes if needed, and finally returns ReadAtTimestamp$(V, t_{snap}(R))$. □

Lemma C.2. *For any data-dependency edge $A \to B$ in the DSG,*

$$\sigma(A) \le \sigma(B).$$

Proof sketch. We consider the three DSG edge types. For edges whose target B is an RW, strict two-phase locking together with Spanner's timestamp assignment places B after the transaction or version that induces the edge. Thus, if $A \to B$ is a WW edge, a WR edge where B is an RW, or an RW anti-dependency where A is an RW, then $\sigma(A) < \sigma(B)$.

It remains to consider edges whose source or target is an RO. For a WR edge where B is an RO, B returns a version written by A. By Lemma C.1, every version returned by B has commit timestamp at most $\sigma(B)$. Hence $\sigma(A) \le \sigma(B)$.

For an RW anti-dependency edge where A is an RO, B is the next committed writer of a key for which A reads an older version. If $\sigma(B) \le \sigma(A)$, then by Lemma C.1, A would have returned B's version or a later version for that key, contradicting the anti-dependency. Hence $\sigma(A) < \sigma(B)$. □

Lemma C.3. *For any completed transaction T,*

$$\sigma(T) < \overline{t}_{resp}(T).$$

Proof sketch. If T is an RW, commit-wait ensures that the commit timestamp has already passed before the transaction responds. Hence

$$\sigma(T) = t_c(T) < t^{abs}_{resp}(T) \le \overline{t}_{resp}(T).$$

If T is an RO and $\sigma(T) = t_{snap}(T) > 0$, then by CalculateSnapshotTS, $t_{snap}(T)$ is the commit timestamp of some candidate version available to the RO client. That version comes from a committed RW whose commit timestamp has already passed before T returns, so

$$\sigma(T) < t^{abs}_{resp}(T) \le \overline{t}_{resp}(T).$$

If T is an RO and $t_{snap}(T) = 0$, the claim is immediate. □

Theorem C.4 (Positive cycle-level AM). *For every skip-induced SS-relative violation cycle C in the DSG augmented with the full conservative RT relation, C contains at least one skip-induced anti-dependency pair (R, W) such that*

$$t_c(W) < \underline{t}_{invoc}(R).$$

Equivalently, C contains at least one skip-induced anti-dependency pair with positive AM.

Proof. We prove the theorem by contradiction. Assume every skip-induced anti-dependency pair (R, W) on C satisfies

$$t_c(W) \ge \underline{t}_{invoc}(R).$$

We assign a tracking value $v(A)$ to each vertex A on C based on its outgoing edge on C:

$$v(A) = \begin{cases} \sigma(A) & \text{if } A\text{'s outgoing edge is a WW/WR/RW edge,} \\ \bar{t}_{\text{resp}}(A) & \text{if } A\text{'s outgoing edge is an } RT \text{ edge.} \end{cases}$$

We traverse C and show that the tracking value never decreases, and that it strictly increases at least once. In Case 5, when an RT edge enters an RO whose next edge is an anti-dependency, we treat the two-edge segment as a single compressed step.

Case 1: $A \to B$ is a data-dependency edge, and B's outgoing edge is a data-dependency edge. Then

$$v(A) = \sigma(A) \le \sigma(B) = v(B)$$

by Lemma C.2.

Case 2: $A \to B$ is a data-dependency edge, and B's outgoing edge is an RT edge. Then

$$v(A) = \sigma(A) \le \sigma(B) < \bar{t}_{\text{resp}}(B) = v(B)$$

by Lemmas C.2 and C.3. This step is strict.

Case 3: $A \to B$ is an RT edge, and B is an RW. Then

$$v(A) = \bar{t}_{\text{resp}}(A) < \underline{t}_{\text{invoc}}(B) \le t_{\text{c}}(B) = \sigma(B).$$

The last inequality follows because Spanner chooses an RW commit timestamp no earlier than the transaction's absolute start time, and $\underline{t}_{\text{invoc}}(B) \le t^{\text{abs}}_{\text{invoc}}(B)$. If B's outgoing edge is a data-dependency edge, then $v(B) = \sigma(B)$. If B's outgoing edge is an RT edge, then $v(B) = \bar{t}_{\text{resp}}(B) > \sigma(B)$ by Lemma C.3. In either case,

$$v(A) < v(B).$$

For Cases 4 and 5, note that an RO performs no writes, so an RO's outgoing edge on C is either an anti-dependency or an RT edge.

Case 4: $A \to B$ is an RT edge, B is an RO, and B's outgoing edge is an RT edge. Then

$$\begin{aligned} v(A) = \bar{t}_{\text{resp}}(A) &< \underline{t}_{\text{invoc}}(B) \le t^{\text{abs}}_{\text{invoc}}(B) \\ &< t^{\text{abs}}_{\text{resp}}(B) \le \bar{t}_{\text{resp}}(B) = v(B). \end{aligned}$$

This step is strict.

Case 5: $A \to B$ is an RT edge, B is an RO, and B's outgoing edge is an anti-dependency $B \xrightarrow{\text{RW}} W$. We compress the two-edge segment

$$A \xrightarrow{\text{RT}} B \xrightarrow{\text{RW}} W$$

and show that $v(A) < v(W)$.

From the RT edge,

$$v(A) = \bar{t}_{\text{resp}}(A) < \underline{t}_{\text{invoc}}(B).$$

Case 5a: $B \xrightarrow{\text{RW}} W$ is skip-induced. By the contradiction hypothesis, the skip-induced anti-dependency pair (B, W) has non-positive AM:

$$t_{\text{c}}(W) \ge \underline{t}_{\text{invoc}}(B).$$

Therefore,

$$v(A) < \underline{t}_{\text{invoc}}(B) \le t_{\text{c}}(W) = \sigma(W).$$

Case 5b: $B \xrightarrow{\text{RW}} W$ is not skip-induced. We first show, by contradiction, that

$$t_{\text{c}}(W) > \underline{t}_{\text{invoc}}(B).$$

Assume instead that

$$t_{\text{c}}(W) \le \underline{t}_{\text{invoc}}(B).$$

Since B chooses $t_{\text{read}}(B) = \text{TT.NOW}().latest$ when it starts, and $\underline{t}_{\text{invoc}}(B)$ is a lower bound on B's absolute invocation time, we have

$$t_{\text{read}}(B) \ge t^{\text{abs}}_{\text{invoc}}(B) \ge \underline{t}_{\text{invoc}}(B).$$

Therefore, $t_{\text{c}}(W) \le t_{\text{read}}(B)$. Because $B \xrightarrow{\text{RW}} W$, RO B read an older version of the key and W is the next committed writer of that key.

There are three subcases. (1) If W's version had already been applied at the shard where B read the key, the candidate versions from the fast/slow replies for that key would have timestamps at least $t_{\text{c}}(W)$, forcing $t_{\text{snap}}(B) \ge t_{\text{c}}(W)$. By Lemma C.1, B would then return W's version or a later version, contradicting the anti-dependency. (2) If W was prepared at that shard, then either B waited for W or B proceeded without waiting. If B waited, the same argument as in (1) applies after the wait. If B proceeded without waiting, then $t_{\text{p},s}(W) \le t_{\text{c}}(W) \le t_{\text{read}}(B)$, so $\text{Wait}_{\text{SS}}(B, W, s)$ holds; since $B \xrightarrow{\text{RW}} W$ is already an anti-dependency, proceeding without waiting makes it skip-induced, contradicting Case 5b. (3) If W was neither applied nor prepared at that shard when B began executing there after the Paxos safe-time check, then a future prepare of W must have timestamp greater than $t_{\text{read}}(B)$. Hence W cannot later commit at or before $t_{\text{read}}(B)$, contradicting the assumption. Thus,

$$t_{\text{c}}(W) > \underline{t}_{\text{invoc}}(B).$$

Therefore,

$$v(A) < \underline{t}_{\text{invoc}}(B) < t_{\text{c}}(W) = \sigma(W).$$

In both Cases 5a and 5b,

$$v(A) < \sigma(W).$$

If W's outgoing edge is a data-dependency edge, then $v(W) = \sigma(W)$. If W's outgoing edge is an RT edge, then $v(W) = \bar{t}_{\text{resp}}(W) > \sigma(W)$ by Lemma C.3. Hence in both cases,

$$v(A) < v(W).$$

Completing the proof. Because all protocols preserve serializability, the DSG is acyclic. Therefore, C contains at least one RT edge. Every RT edge on C falls under Case 3, Case 4, or the compressed Case 5; each of these yields a strict increase. All remaining data-dependency steps on C fall under Case 1 or Case 2 and never decrease the tracking value.

Chaining the inequalities around C, with Case 5 segments compressed, gives

$$v(A_1) \le \cdots < \cdots \le v(A_1),$$

which implies

$$v(A_1) < v(A_1),$$

a contradiction. Therefore, our assumption is false, so C contains at least one skip-induced anti-dependency pair (R, W) with positive AM. □