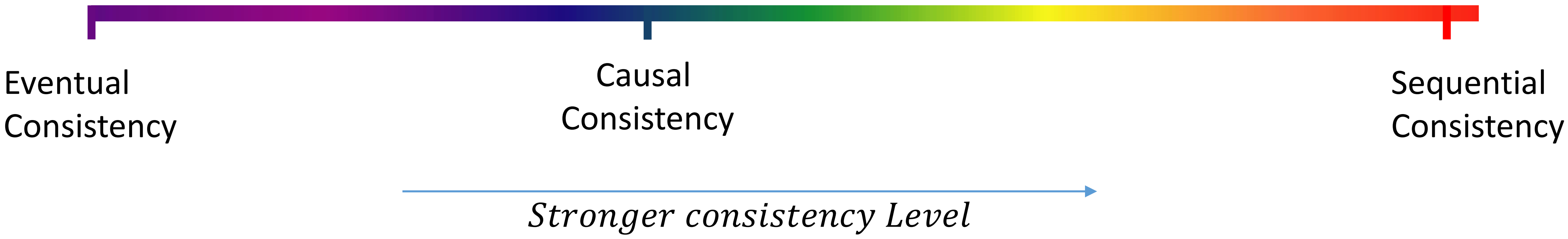
GentleRain: Cheap and Scalable Causal Consistency with Physical Clocks

Jiaqing Du | Calin Iorgulescu | Amitabha Roy | Willy Zwaenepoel
École polytechnique fédérale de Lausanne (EPFL)

Published in the Proceedings of the 5th ACM Symposium on Cloud Computing, 2014

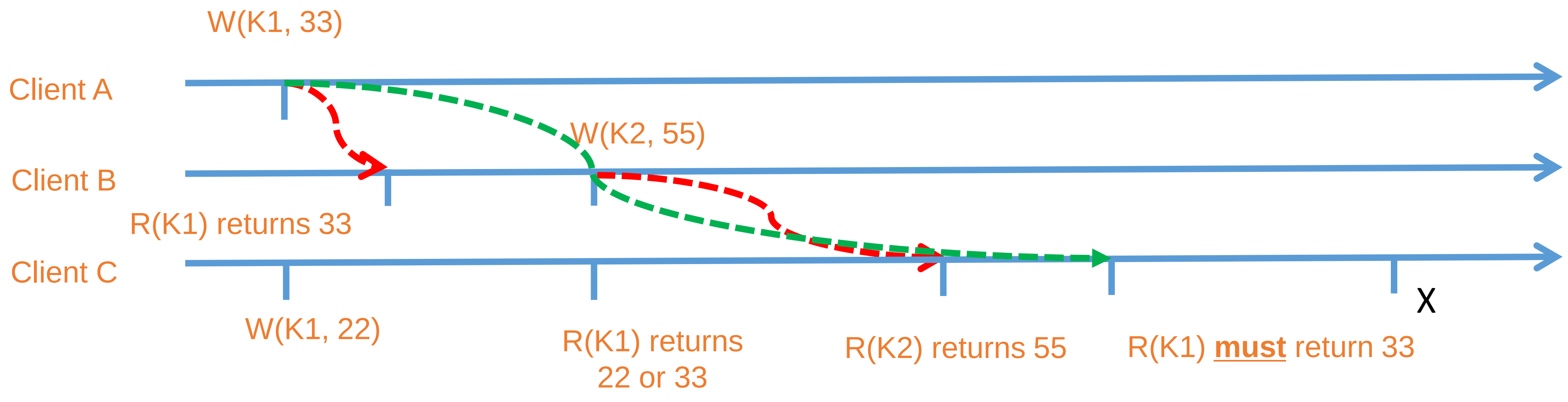
Presented By: Aditya Rastogi

What is Causal Consistency ?



- From the point of view of a client : If a certain version of a data item is visible, then all of its causal dependencies (all versions that *happen before* this version) are also visible.
- Operations that are causally related (*happens before relationship*) are seen by every client in the same order.

Example



- Social Network Updates
- Order of display of unrelated status updates does not matter. (concurrent events)
- But Comments in response to a post must not appear beofore that post! (causally related events)

GentleRain

- A Geo-Replicated data store
- Provides Causal Consistency
- Motivation: No need for dependency check messages, use a single physical timestamp instead
- Benefit: Achieve greater throughput
- Tradeoff: Delayed visibility of updates at remote replicas

System Model

- N partitions containing keys assigned by hash value
- Each partition replicated by M replicas (datacenters)
- Servers with physical clocks with monotonically increasing timestamps
- *Put(key,val)* : Create / modify key
- *Get(key)* : Get value for the key
- *Sn_read(keys)* : Returns a causally consistent snapshot containing values for all the *keys.*
- *Ro_trx(keys)* : Returns values for a causally consistent read only transaction. Values previously seen by the client must also be returned

GentleRain Protocol

- Timestamp all updates with physical clock value at originating server
- Local updates are immediately visible
- Remote updates are visible only when older than a global timestamp determined by Global Stable Timestamp (GST)
- All updates across different partitions and replicas totally ordered by update timestamp

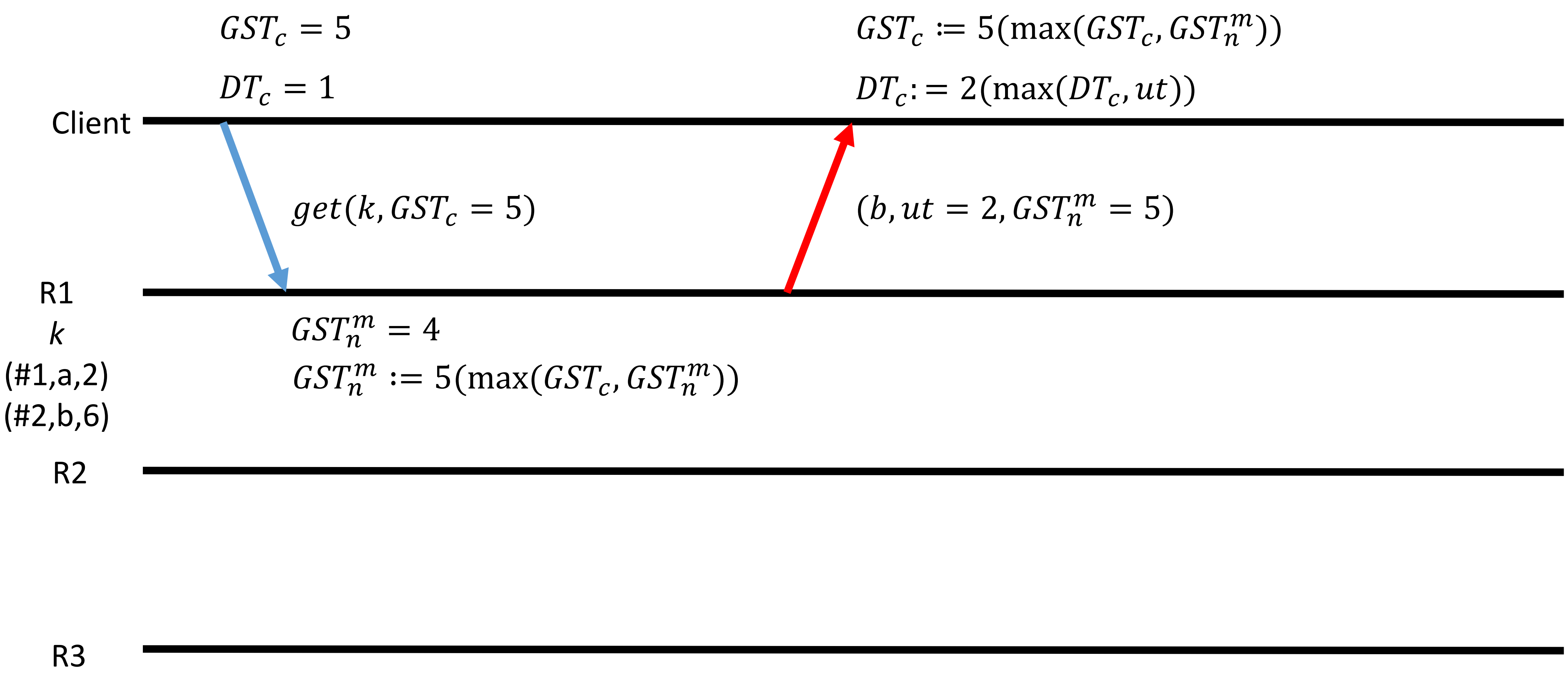
Client and Server States

- Client
  - Dependency Time DT_c: latest update timestamp across all items accessed by client
  - GST_c : Client's knowledge of Global Stable Time.
- Server
  - Version Vector $VV_n^m[1..M]$: Physical timestamp vector at m^{th} replica of n^{th} partition(key).
  - Local Stable Time LST_n^m : Minimum element of VV_n^m at a partition.
  - Global Stable Time GST_n^m : Lower bound of minimum LST of *all* partitions(keys) *within the datacenter*.
  - Each item maintained as a tuple <key, value, update_timestamp, source_id>, list of versions maintained.
  - Messages sent out in update timestamp and clock order.

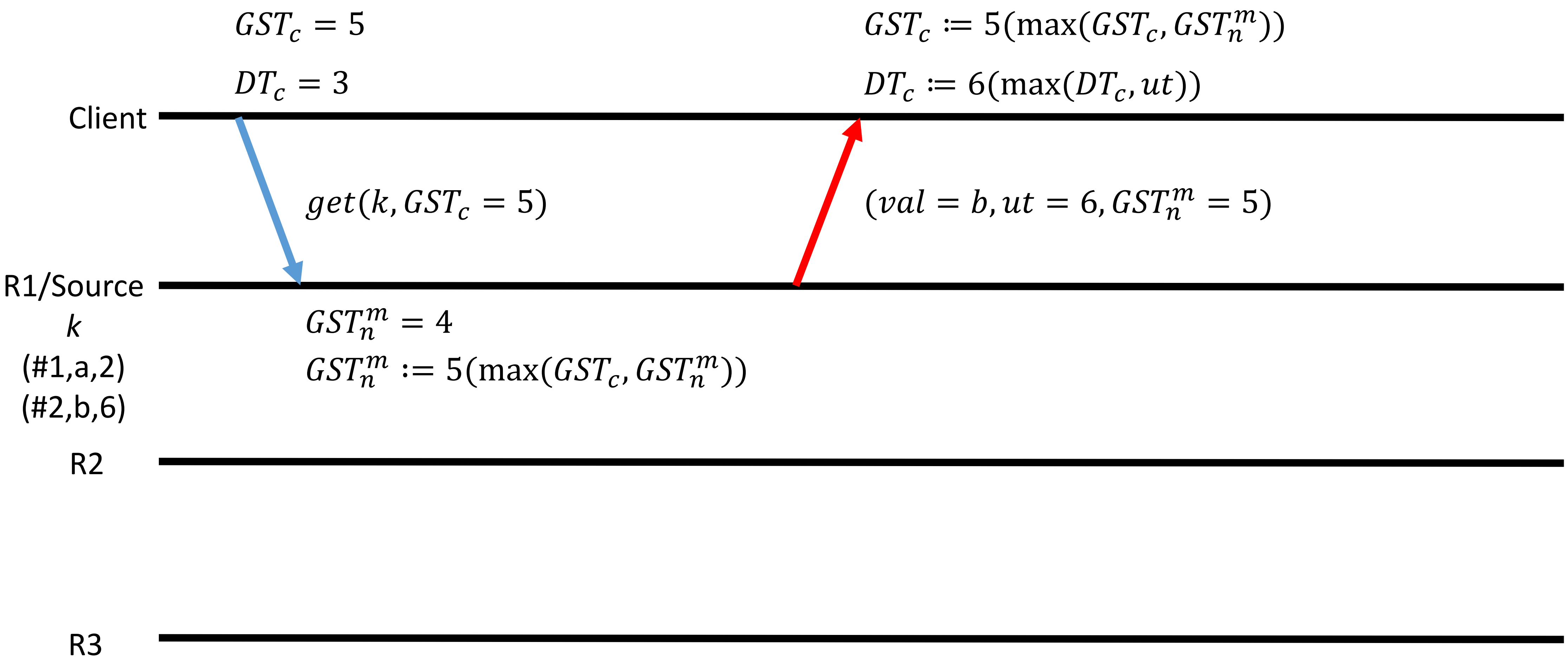
Understanding GST

- Intuitively, serves as a cutoff time for causally consistent reads.
- All remote reads are return values with update timestamp < GST
- Guarantees that if at a certain partition the GST value is T, then all partitions(keys) have received all updates with update timestamp less than GST.

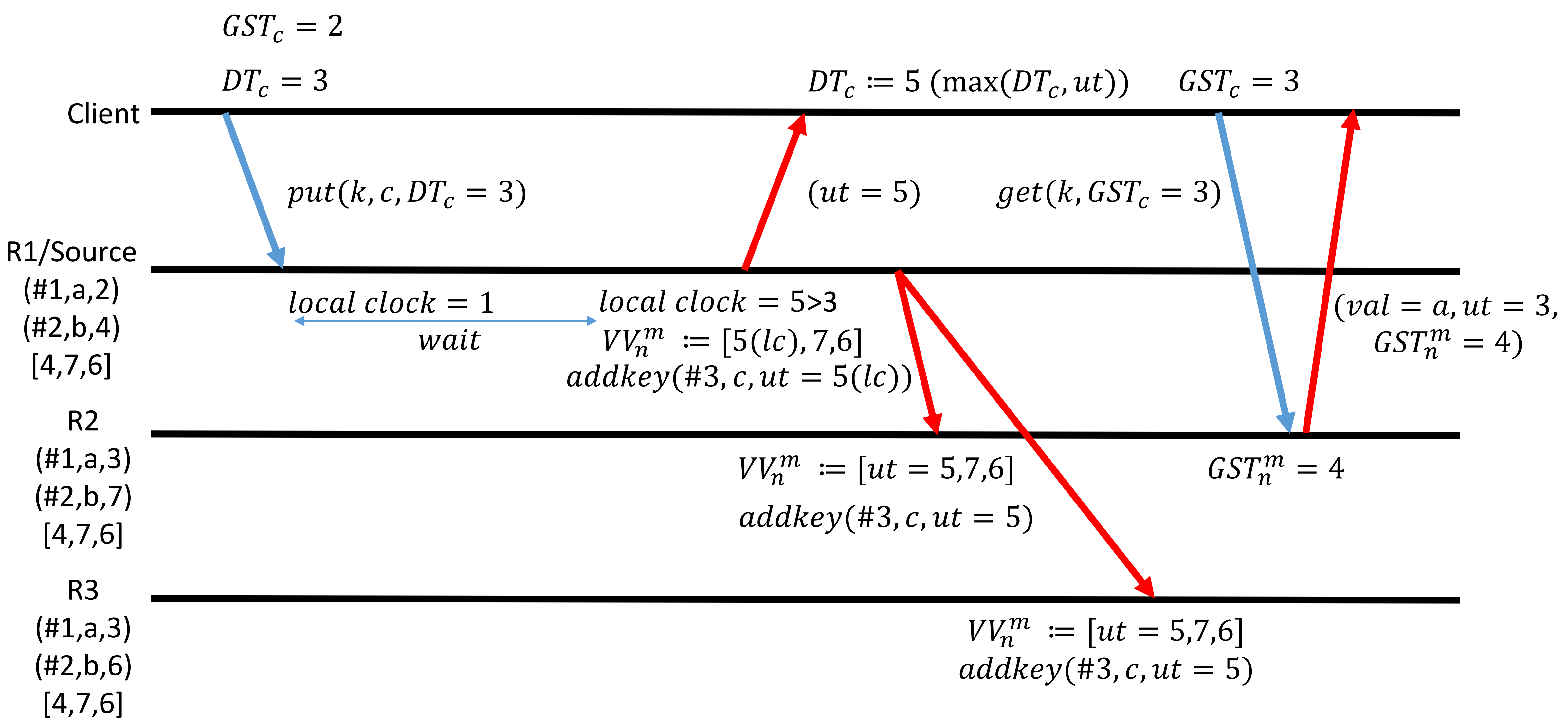
Get Operation (Non-Local Reads)



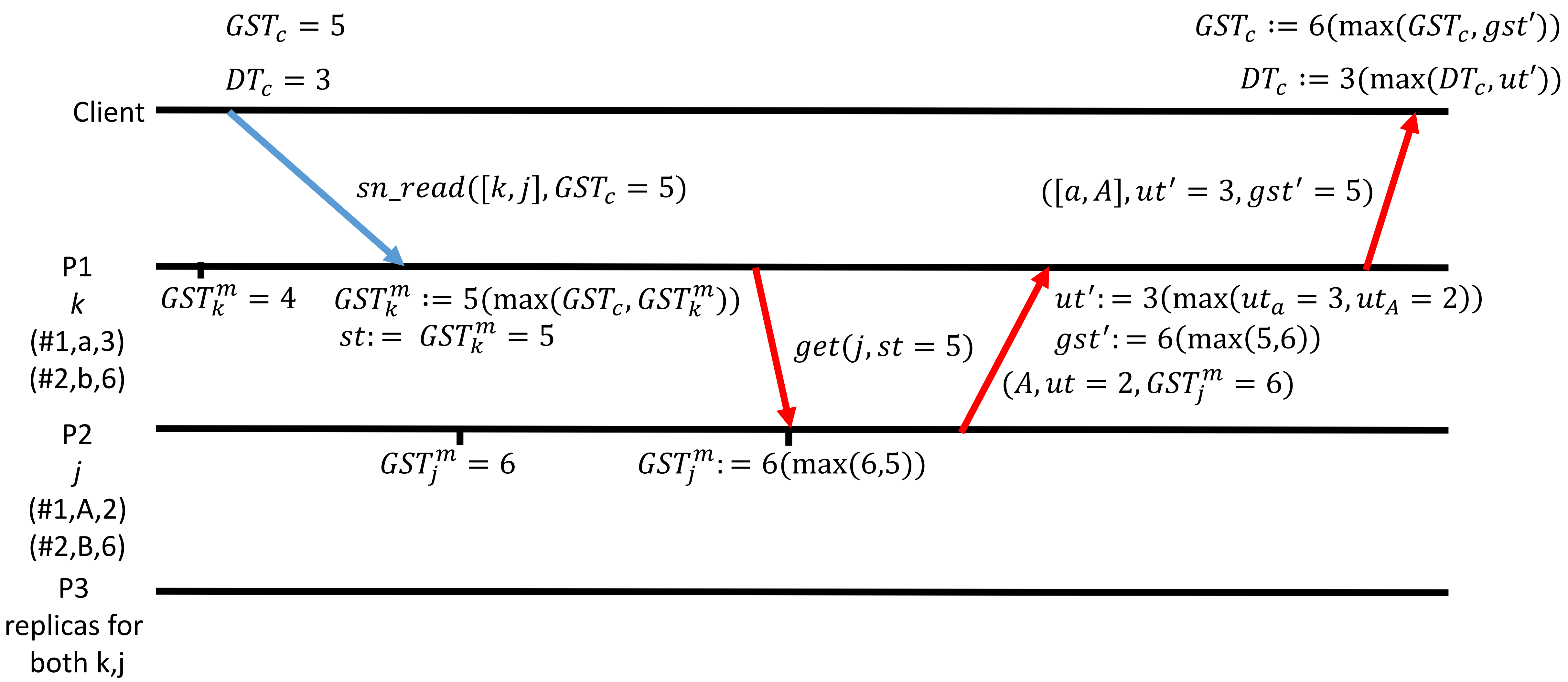
Get Operation (Local Reads)



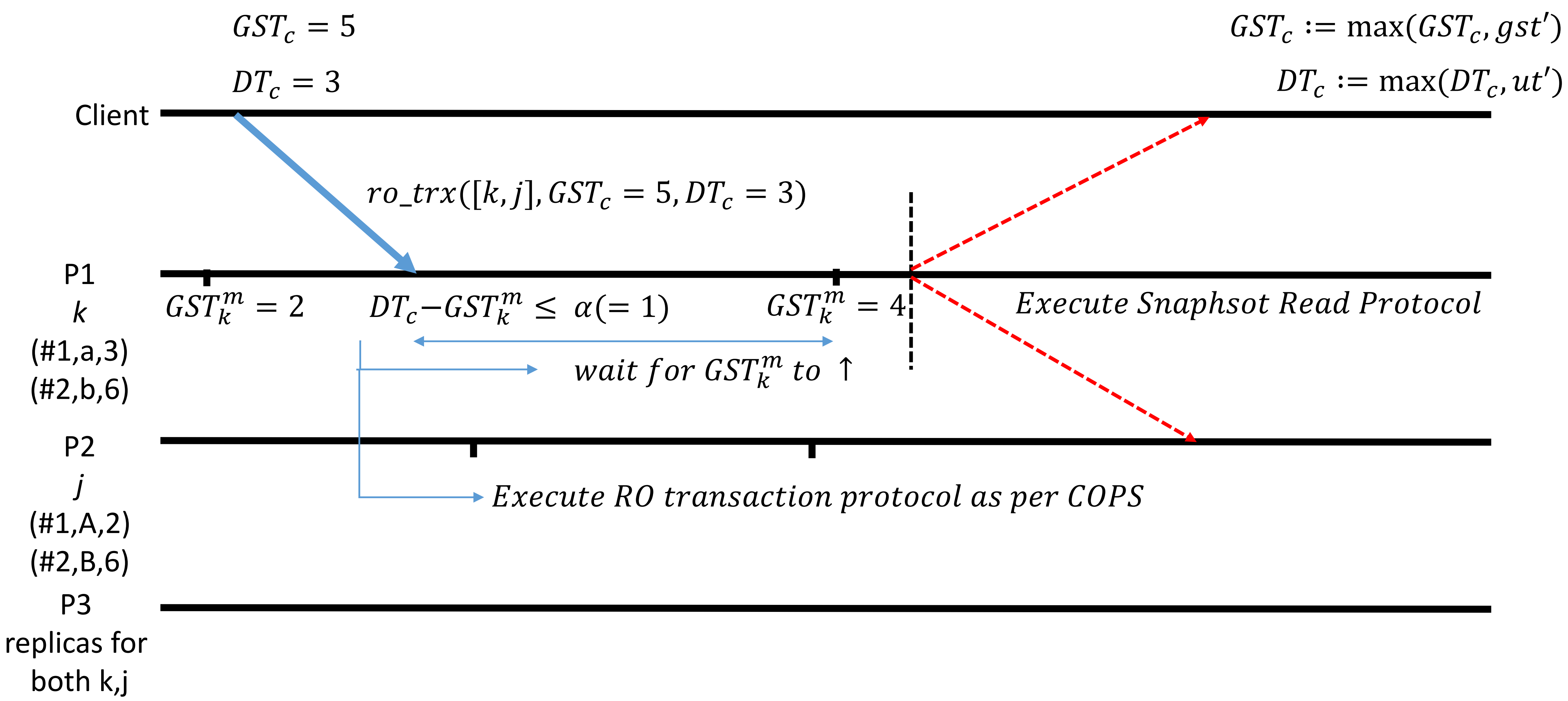
Put Operation



Snapshot Read (Across Partitions)



Read-Only Transactions



GST Derivation

- GST_n^m at a server is the lower bound on the minimum LST_n^m of all partitions(keys) within the *same datacenter*. i.e.
$$GST_n^m = \min(LST_k^m) \; \forall \, k \in N$$
- Periodically computed for partitions(keys) within *same datacenter.*
- For efficient derivation of GST_n^m at a datacenter, spanning tree built over all partitions in the datacenter.
- Leaf nodes push GST_n^m up the tree, root communicates the min GST_n^m back.
- Message complexity = $O(N)$, time taken = $2 * RTT * logN$.

Heartbeats

- If a partition (key) does not receive frequent updates its VV_n^m will not advance $\rightarrow$ LST_n^m will not advance $\rightarrow$ GST_n^m will not advance !
- To solve this :
- Periodically update VV_n^m at each partition(key)
- Set $VV_n^m[m] := local\ clock$ at replica m
- Broadcast local clock to all replicas, using piggybacking on failure detector heartbeats.
- At replica $k \neq m$ set $VV_n^m[k] :=$ clock from heartbeat of replica k

Garbage Collection

- Partitions within the same datacenter periodically exchange snapshot timestamp of oldest active snapshot read.
- If a partition does not have any active snapshot read, it sends out GST.
- Partitions choose minimum timestamp of all such snapshot timestamps for garbage collection.
- Keep only the latest item versions just before this timestamp , discard earlier versions.

Conflict Detection

- Remember, even in causal ordering, you can have concurrent events !
- Conflict happens when causally unrelated updates to same key are done at two different replicas.
- Updates that need to be replicated carry *update time* and *source replica id* of previous version.
- Replicate operation at a server applied only if the previous version at server = previous version in replicate message.
- Otherwise conflict reported to client which dictates the order of conflicting updates in a *consistent manner across servers.*

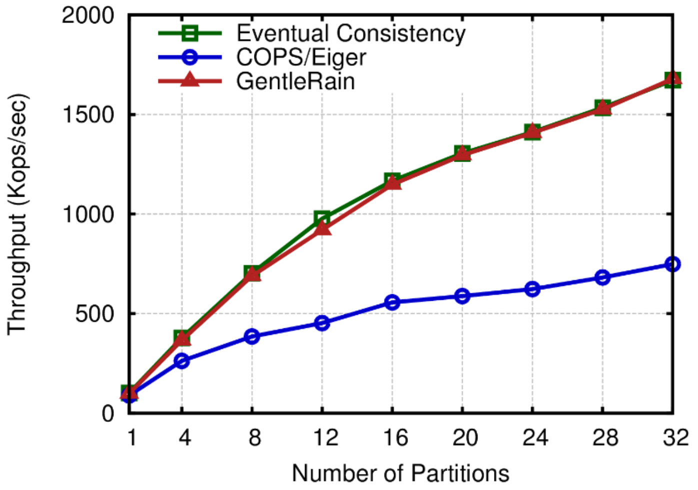
Why Physical Clocks?

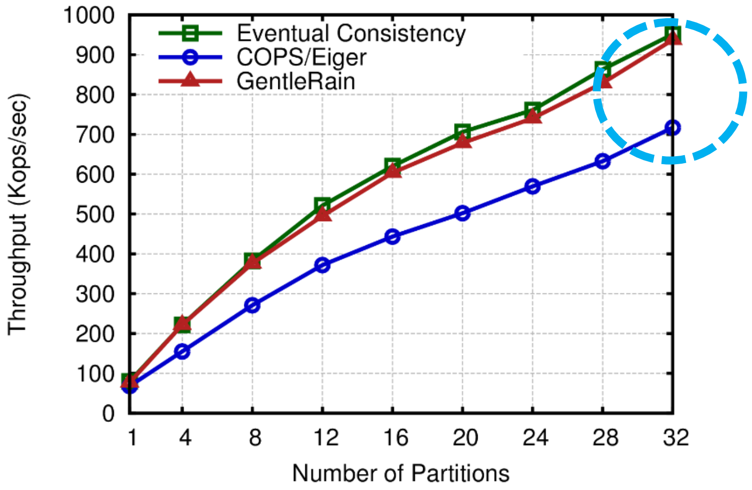
- System can be causally consistent even if we use logical clocks.
- However, logical clocks only updated when update is made.
- But Partitions(keys) can receive updates at different frequencies.
- If a partition (key) does not receive frequent updates its VV_n^m will not advance → LST_n^m will not advance → GST_n^m will not advance !
- Hence, loosely synced (using NTP) physical clocks used as timestamps for updates.

Results

- System Evaluated in terms of throughput and remote update visibility
- Compared to data stores providing Eventual Consistency and Causal Consistency
- Each partition replicated at three Amazon EC2 datacenters – Oregon (O), Ireland (I) and Virginia(V)

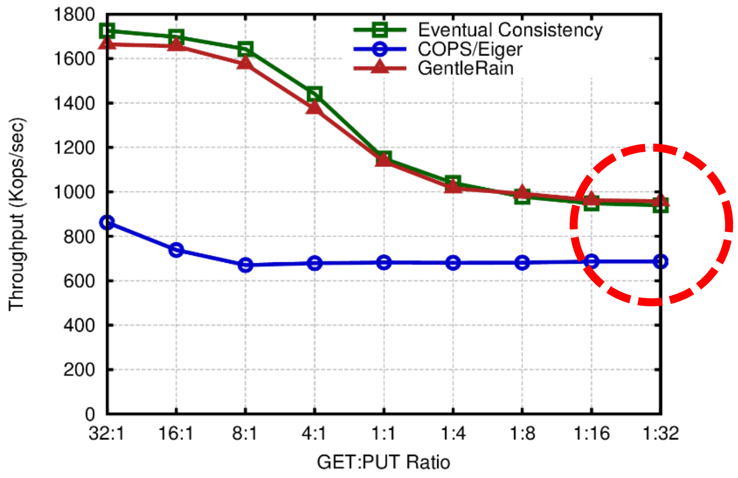
Results - Throughput

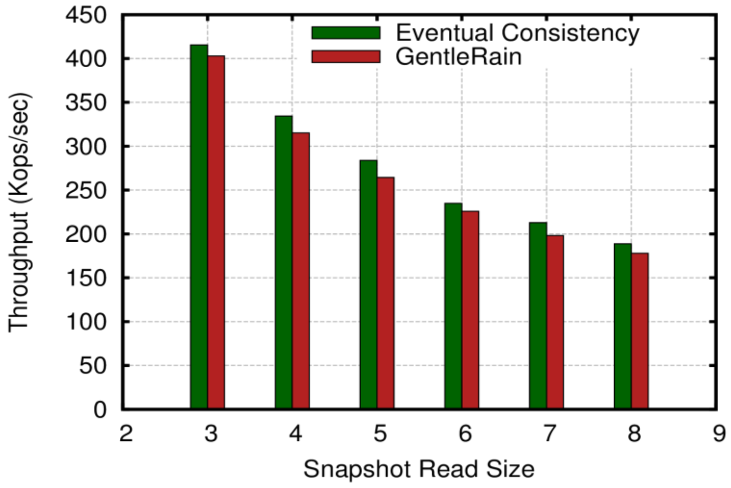




- Left: Read a randomly selected item from every partition and update a randomly selected item at one partition.
  - Much better throughput than COPS which needs to send dep-check messages to all partitions
- Right: Update a randomly selected item in each partition in round-robin fashion
  - GAP in throughput smaller due to lesser no of dep-check messages in COPS

Results - Throughput

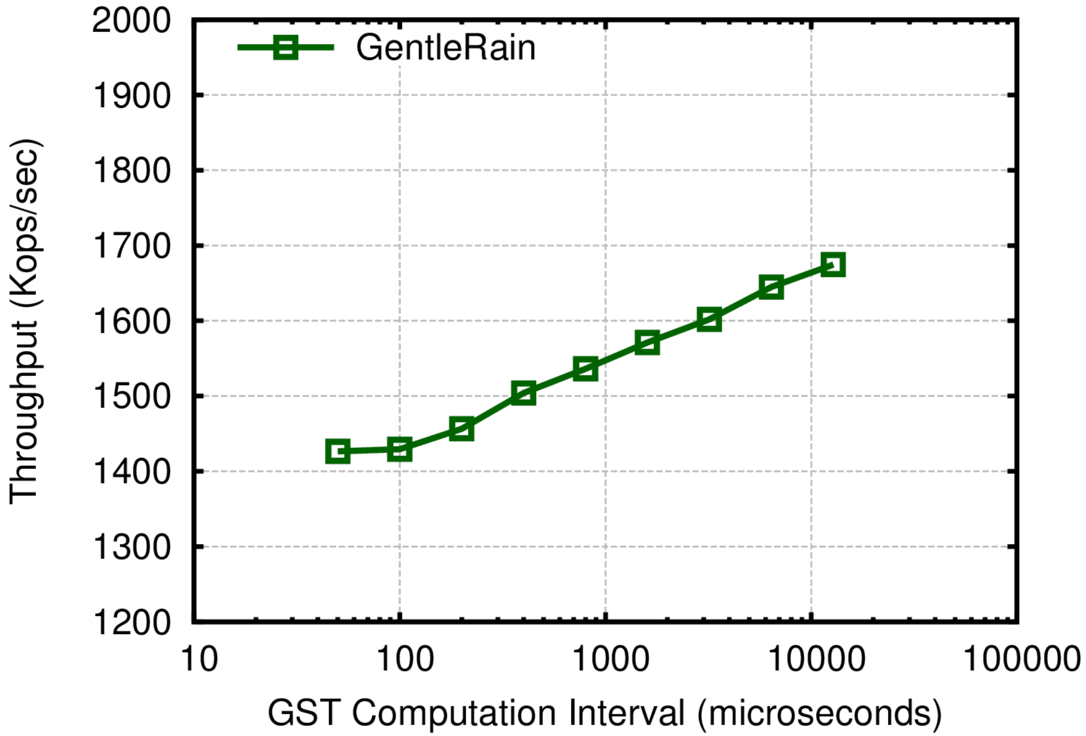




- Left: Read N randomly selected items from randomly selected partitions and write one random item to each of M randomly selected partitions.
  - GAP in throughput narrows as COPS does not need to track a lot of dependencies.
- Right: Causally Consistent snapshot reads in GentleRain and reads in Eventually Consistent systems. Nearly identical throughput.

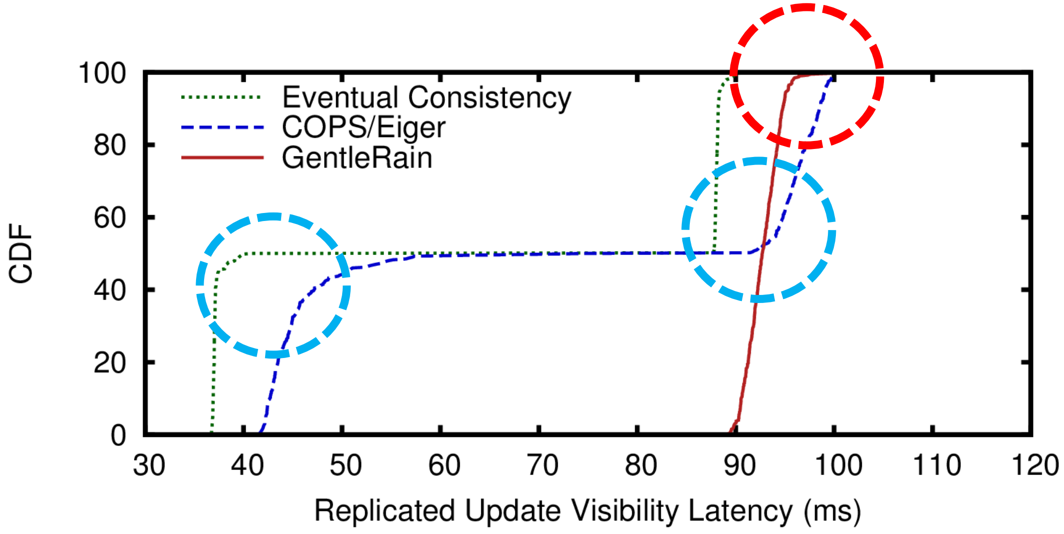
Results – Impact of GST update

- Increasing the time between GST updates leads to marginal increase in Throughput.
- Increase of 256x in GST causes increase of only 1.15x in throughput.
- GST message exchange traffic contained within datacenter.



Results – Update Visibility Latency

- Measured as the time difference between physical update time at the origin replica and the time when update becomes visible at remote replica.
- Updates originating at I(50%) and V(50%) and later made visible at O
- COPS Update Visibility equal to network travel time.
- Gentle Rain Update Visibility equal to longest network travel time (between O & I) + GST update time



Pros

- Throughput comparable to Eventually consistent data stores .
- Idea of using physical clocks instead of logical – system built on top of existing clock sync protocols like NTP
- Message size and bandwidth savings through elimination of dependency check messages.
- Conflict detection

Improvements

- Biggest Drawback : Getting GST to make adequate progress across datacenters
  - Network Partitions across datacenters : Datacenters Excluded from GST calculation
  - Machine Failures : Duplicate stable copies
  - Heartbeat piggybacking more of a workaround , not reliable
- Without GST updates remote update visibility impacted.
- Tree model of dissemination susceptible to failures
- Parameters
  - How frequently should heartbeats be sent out ?
  - How recent writes supported for serving read only transactions (α) ?
- Lack of negative / failure scenario experiments. What is the impact when GST update does not happen at all ?

Related Work

- Spanner
  - Serializable transactions with external consistency.
  - Relies on synchronized GPS and atomic clocks to bound time uncertainty
  - Relies on the
- COPS
  - Used as baseline for comparison
  - Implements causal consistency in partitioned replicated datastore.
  - Causal dependencies recorded for an update are sent with update replication messages
  - At remote datacenter, causal dependencies are verified by sending dep-check messages to other partitions

Your Questions

- What happens when a datacenter is partitioned, what happens on rejoins ?
- Clock skew may impact visibility of updates ?
- With failure of root nodes within datacenters, how would GST be computed ?
- How consistency is maintained among replicas in the same data center. Is an update installed only after approval from all local replicas ?