How We Doubled System Read Throughput with Only 26 Lines of Code

Presented by Minghua Tang

About me

- Minghua Tang
- Interested in databases, storage systems (and Civilization)
- R&D, PingCAP
- Github ID: @5kbpers

Agenda

- What’s TiKV
- How Follower Read was built
- General use cases
- Q&A

Part 1 - What’s TiKV

TiKV is ...

- a **distributed transactional key-value** database originally created by PingCAP as the underlying storage engine for TiDB

- based on the design of **Google Spanner** and **HBase**, but simpler to manage and without dependencies on any distributed file system

- a **CNCF** incubating project with 7.6 K GitHub Stars and 246 Contributors

TiKV offers ...

- Key-Value store

- Get(Key)
- Put(Key, Value)
- Delete(Key)
- Scan(StartKey)

Infra build upon TiKV:

TiDB (SQL like), Tedis (Redis like), etc

TiKV offers ...

- Key-Value store
- Cloud native

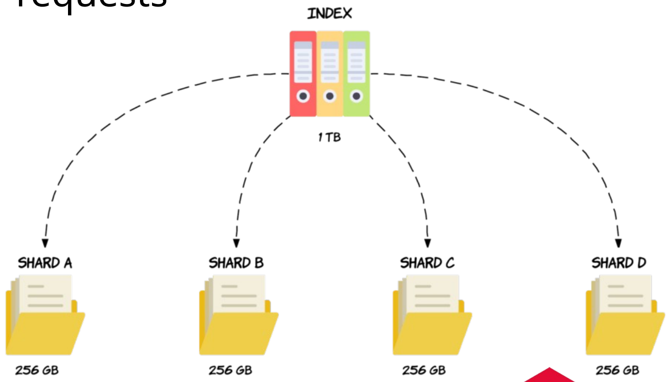
You can run TiKV across physical, virtual, container, and cloud environments

TiKV offers ...

- Key-Value store
- Cloud native
- Horizontal scalability

Deploy more TiKV instances to **scale out**:

- **Scale Storage** to store petabytes of data
- **Scale Performance** to handle more requests

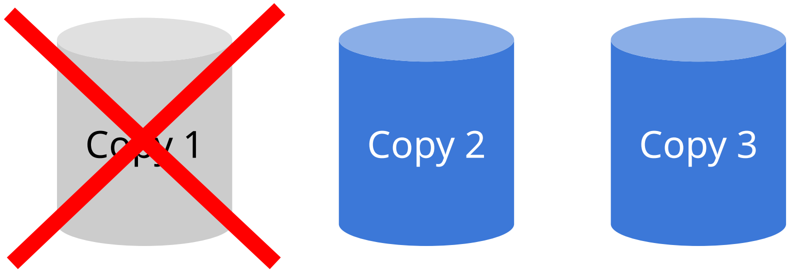


TiKV

TiKV offers ...

- Key-Value store
- Cloud native
- Horizontal scalability
- High availability

Replicate and store data in multiple **distant physical locations** to provide redundancy in case of data center failures.



TiKV

TiKV offers ...

- Key-Value store
- Cloud native
- Horizontal scalability
- High availability
- Dynamic membership

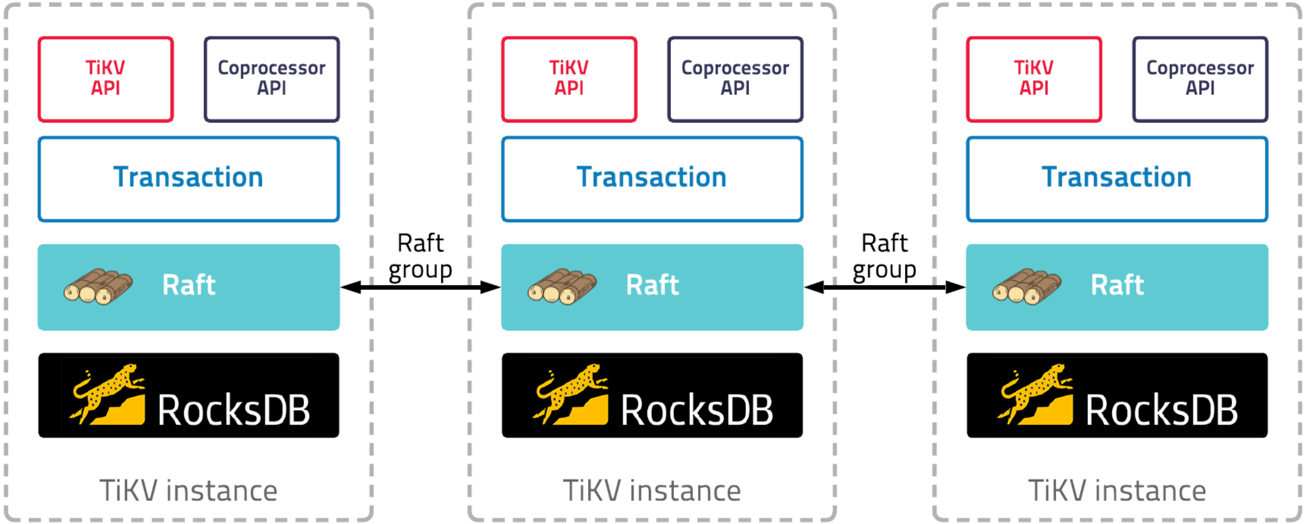
Grow or **shrink** TiKV clusters dynamically, without the need for downtime

TiKV offers ...

- Key-Value store
- Cloud native
- Horizontal scalability
- High availability
- Dynamic membership
- Transactional

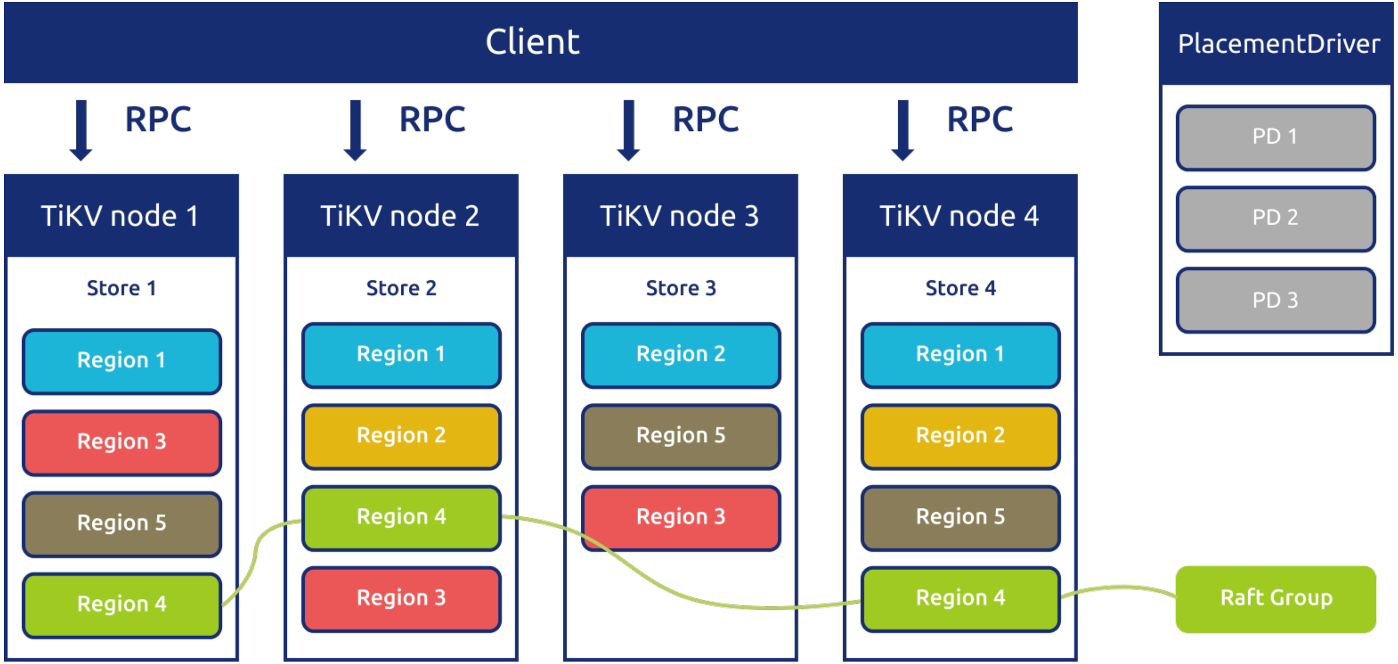
Provides externally consistent distributed transactions (**ACID**) to operate over multiple Key-Value pairs.

System architecture

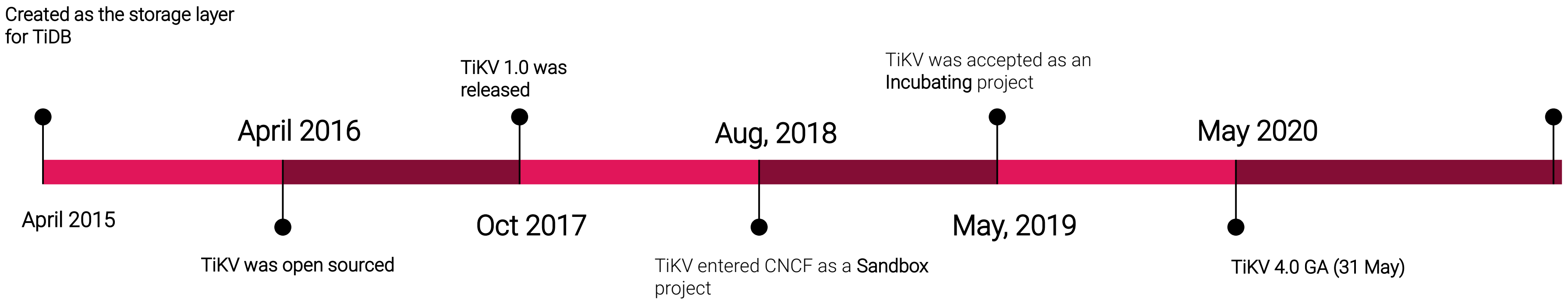


TiKV

System architecture



TiKV Timeline



Part 2 - How Follower Read was built

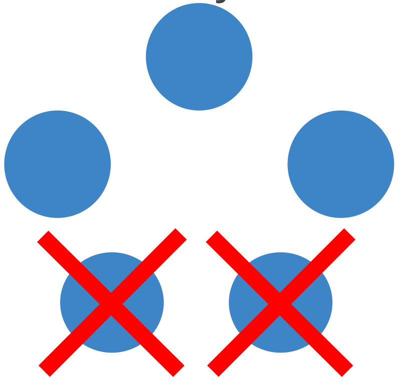
Why Follower Read ?

- By default, only the leader in a Region handled heavy workloads
- Question: how to reduce the load on the leader and scale out efficiently?
- Follower Read: let followers serve read requests

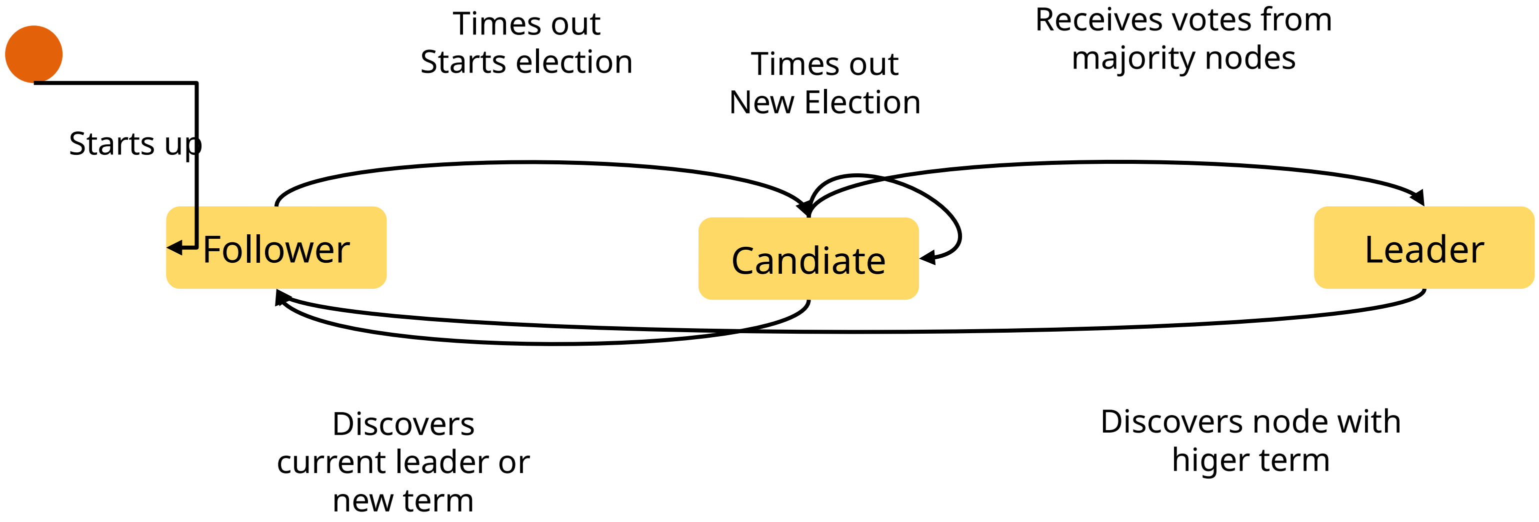
TiKV

Raft Consensus Algorithm

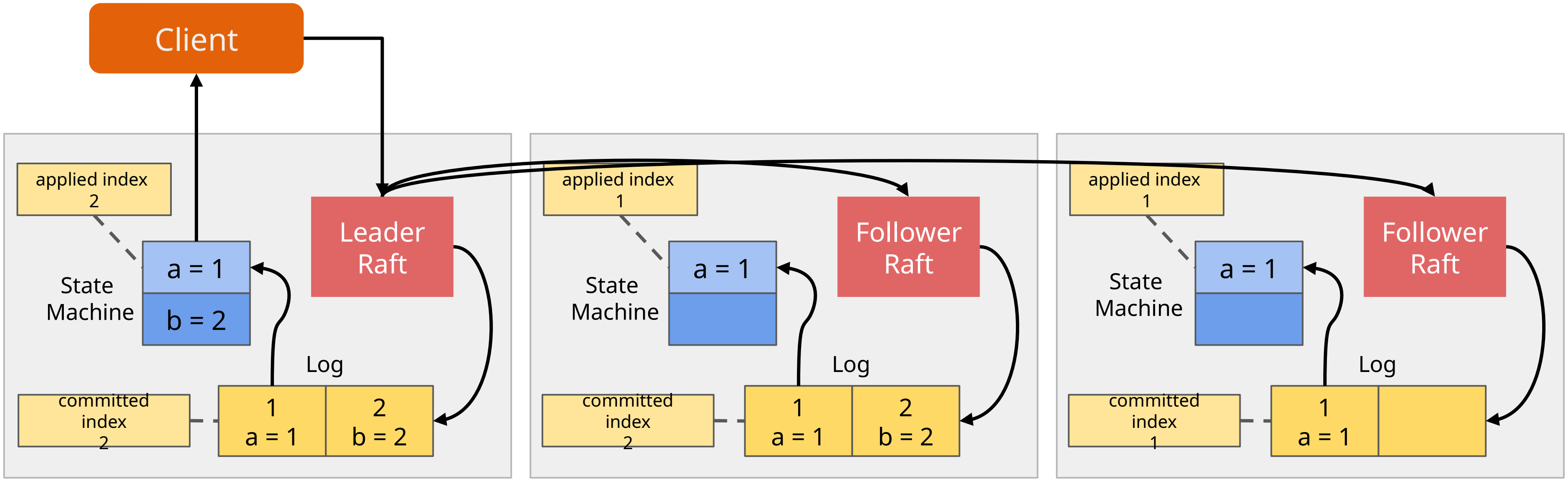
- What is consensus?
  - Agreement on shared state
  - Recovers from server failures autonomously
    - Minority of servers fail: no problem
    - Majority fail: lose availability, retain consistency



Server States

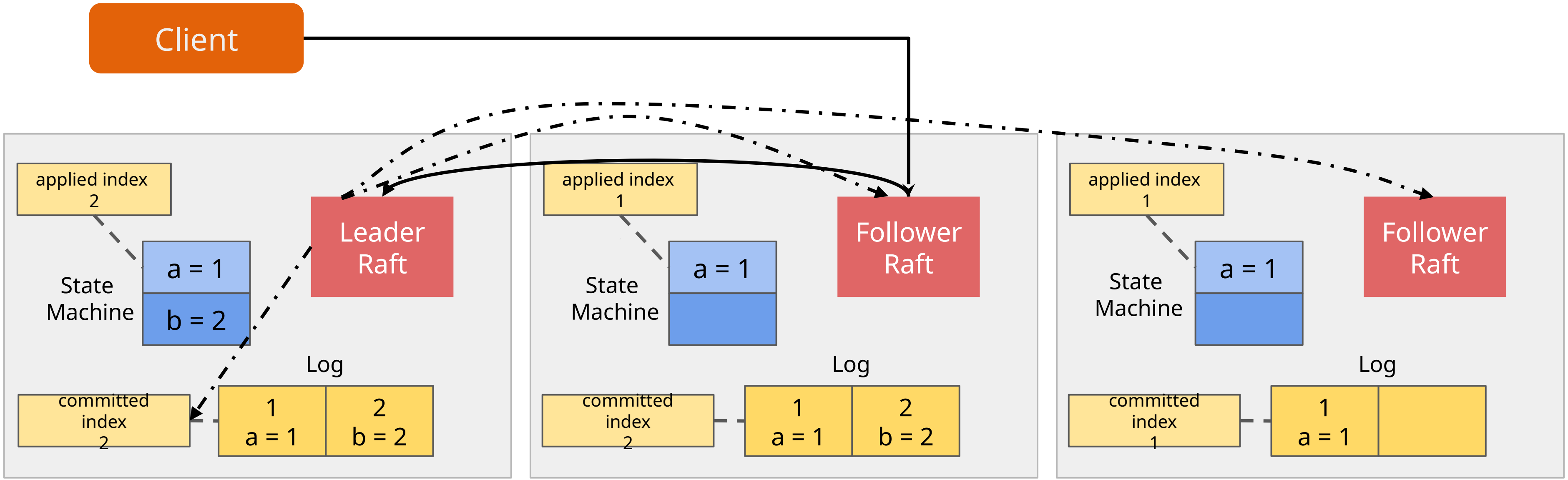


Log replication



- **Commit:** Replicate logs to a majority of replicas. The progress was recorded by **committed index (only available on leader)**
- **Apply:** Execute commands inside logs in the state machine. The progress was recorded by **applied index**
- **Note:** follower applied index != leader applied index != leader committed index

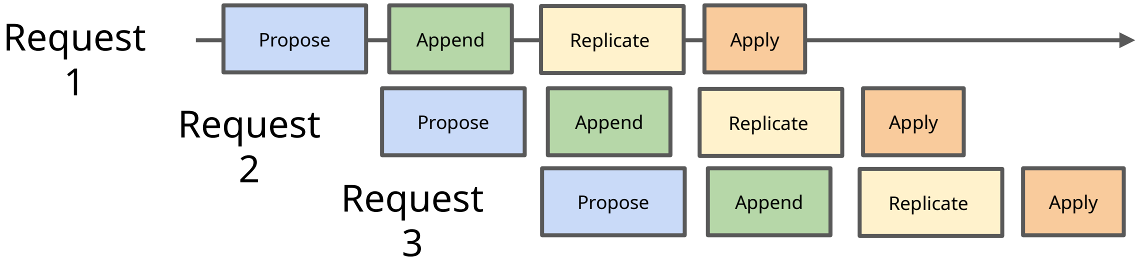
Read Index



- **Steps:**
  1. Follower requests a ReadIndex from leader
  2. Leader reads its committed index and broadcasts a message for confirming its liveness
  3. Leader returns the committed index to follower
- **Optimazation:** Lease Read

Follower Read

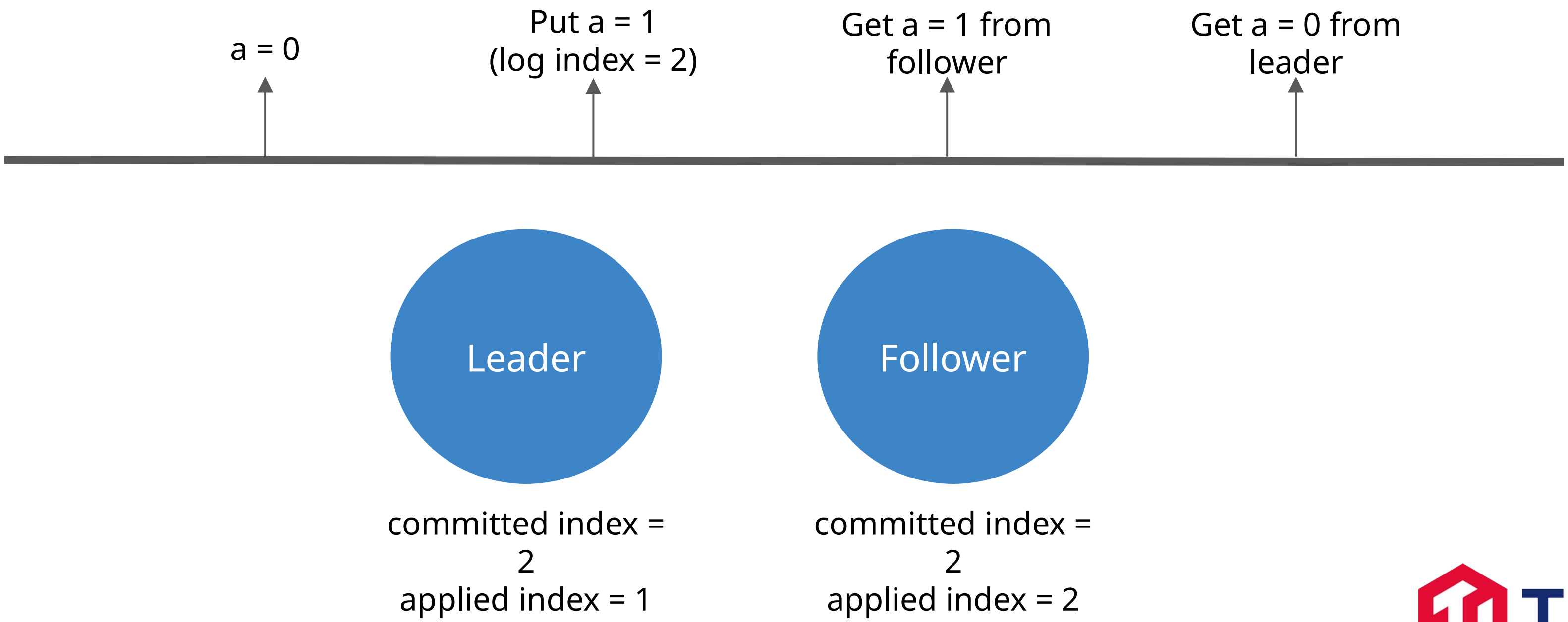
- Two steps
  - request leader committed index through ReadIndex
  - read states locally in state machine of the follower
- Exception:
  - TiKV implements pipelined raft, “apply” is executed asynchronously
    - leader may apply slower than followers
  - Then what if follow applied index > leader applied index?



follower applied index > leader applied index

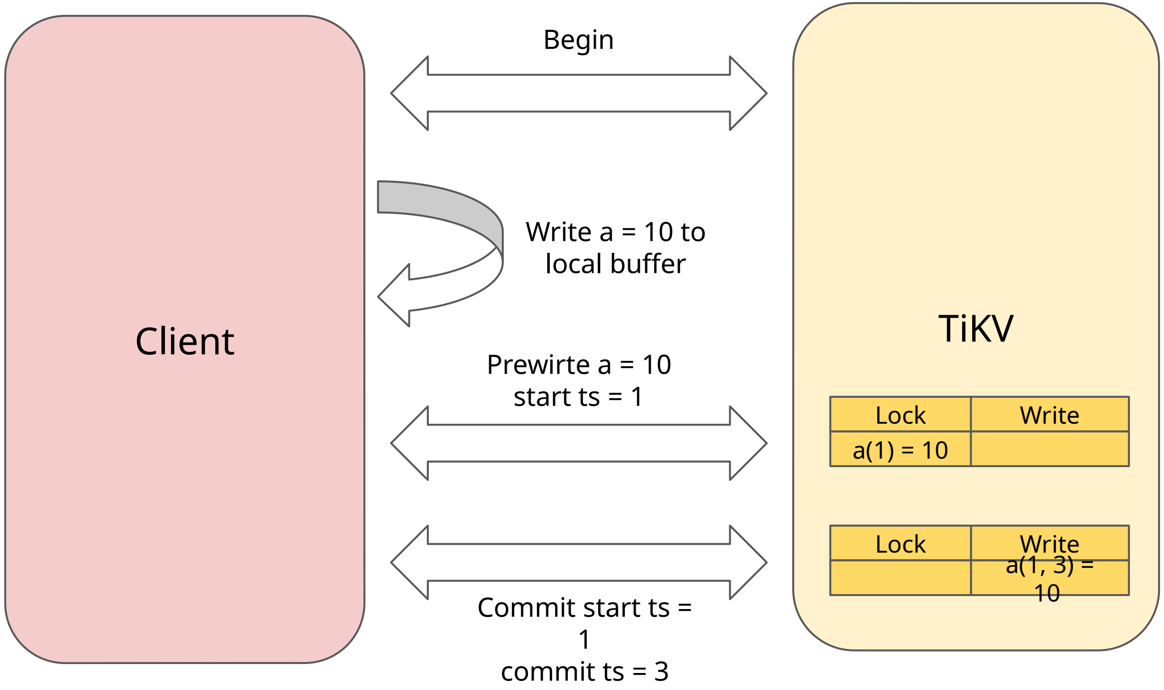
Break linearizability !!!

But snapshot isolation is still ok.



TiKV

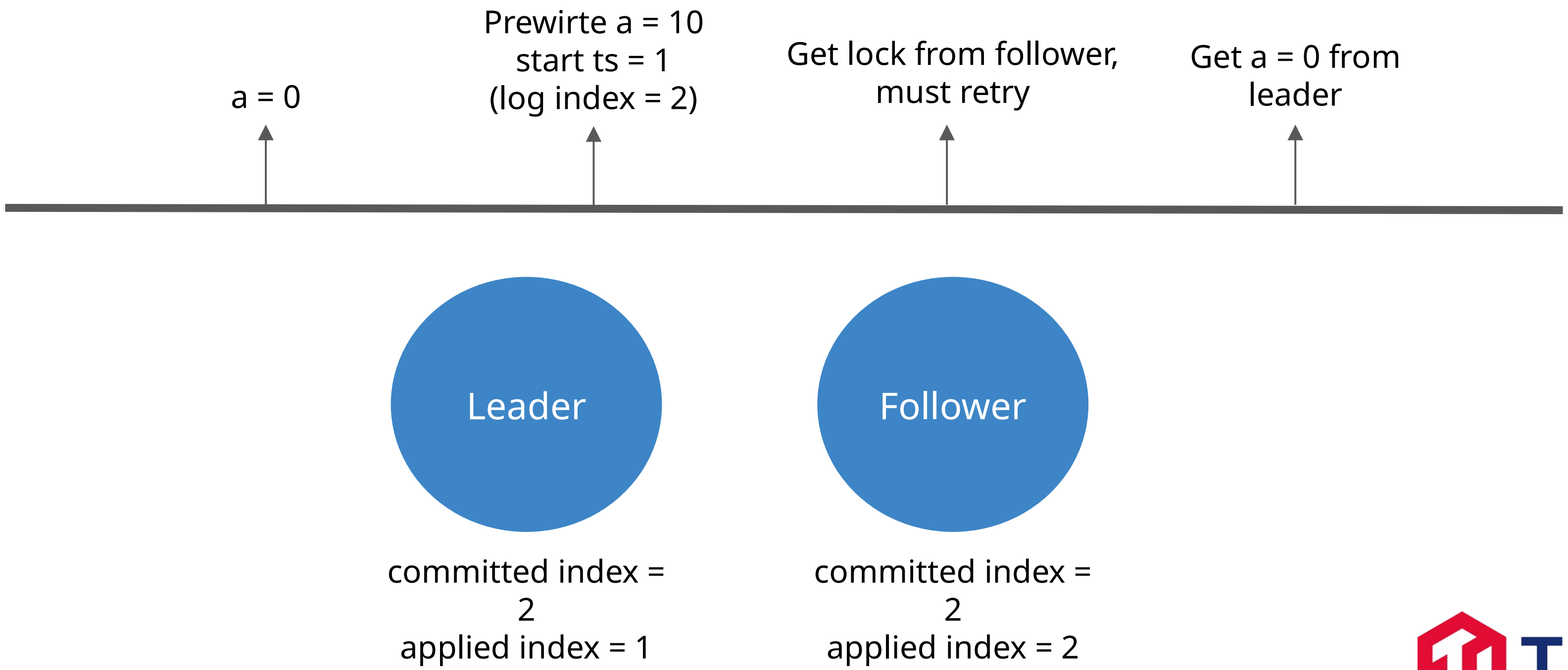
Tranactions in TiKV



TiKV

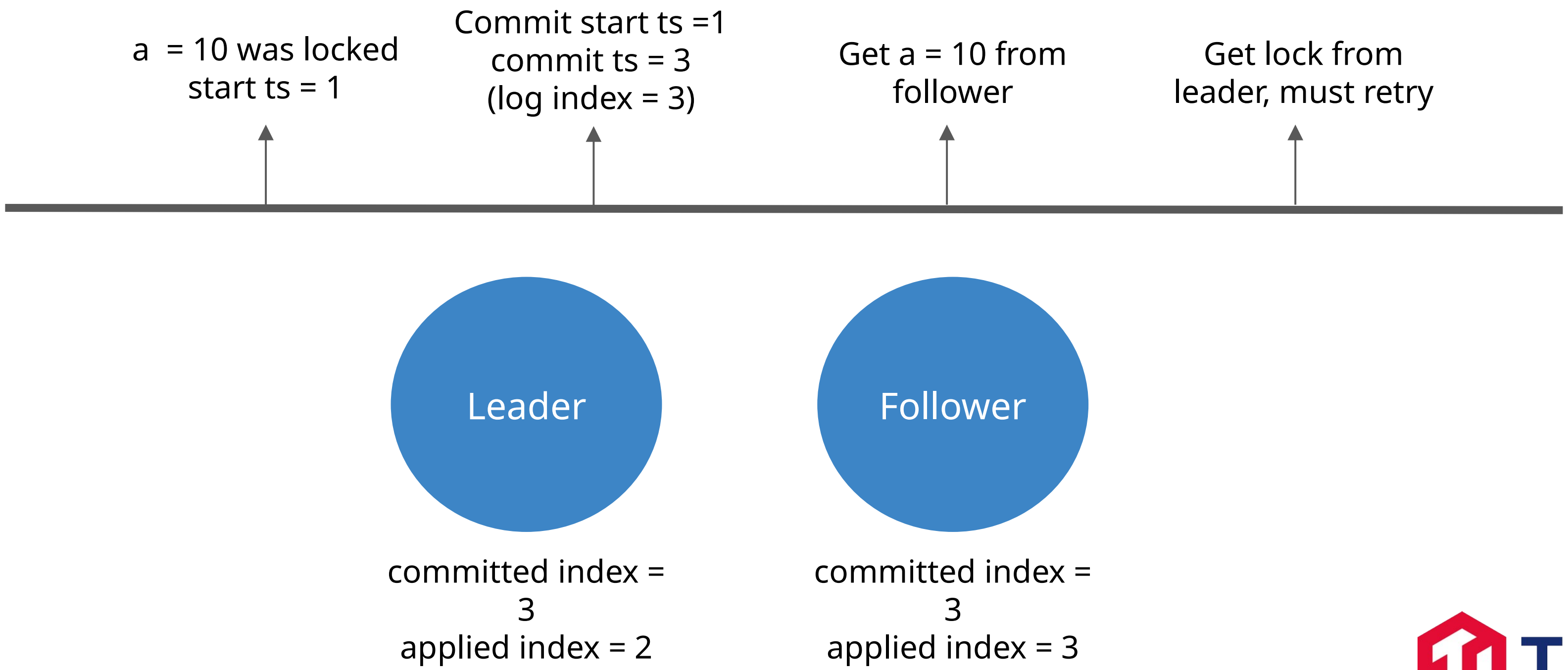
Snapshot Isolation ?

Snapshot isolation is still ok.



Snapshot Isolation ?

Snapshot isolation is still ok.

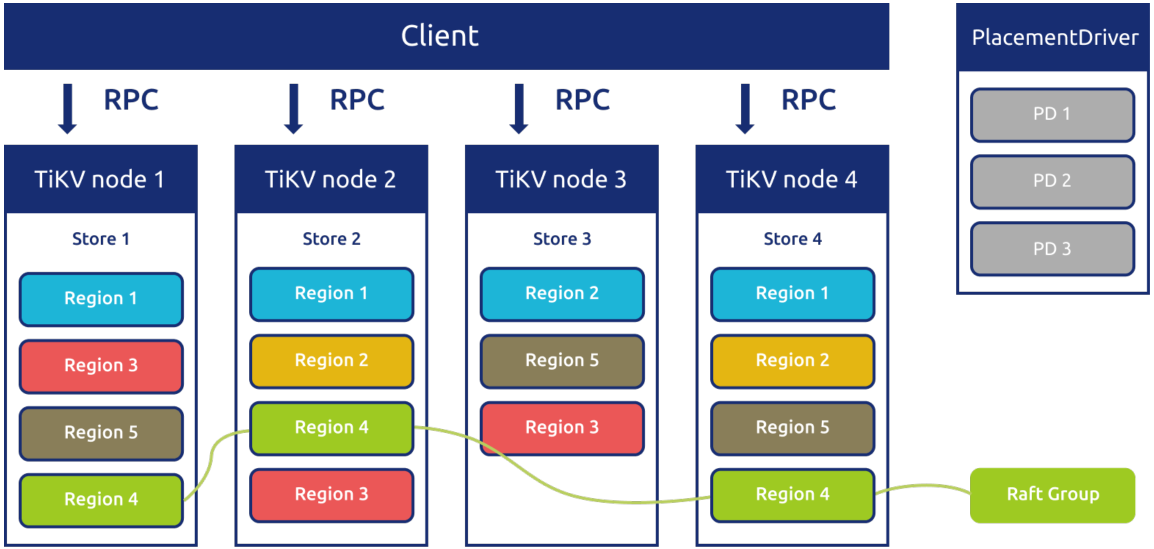


TiKV

Part 3 - General use cases

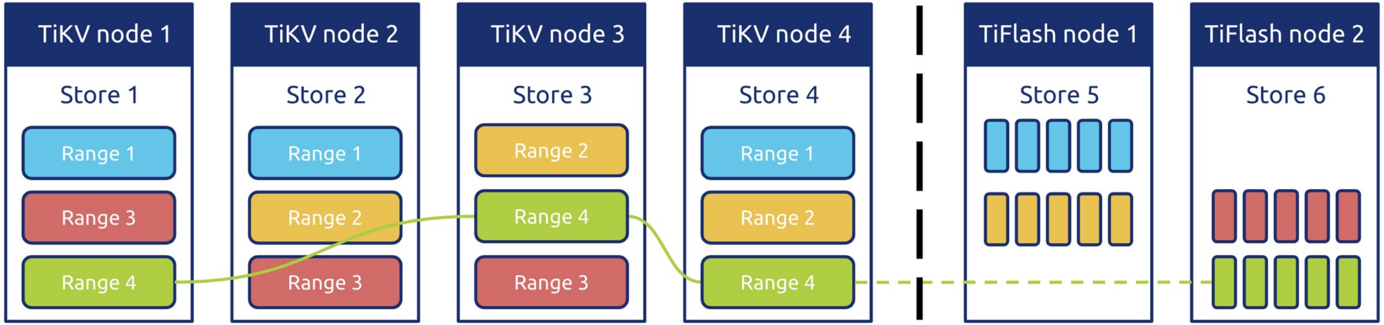
General use cases

- **Note: Generally Follower Read is not helpful for performance**
    - TiDB is a multi Raft service, leaders are balanced on stores



General use cases

- **Note: Generally Follower Read is not helpful for performance**
  - TiDB is a multi Raft service, leaders are balanced on stores
- Use case 1: Build a HTAP system
  - Performing read on a column store (TiFlash) is much faster than on TiKV



General use cases

- **Note: Generally Follower Read is not helpful for performance**
  - TiDB is a multi Raft service, leaders are balanced on stores
- Use case 1: Build a HTAP system
  - Performing read on a column store (TiFlash) is much faster than on TiKV
- Use case 2: Read cross multiple data centers
  - Performing read on nearlier data center

Number of scan keys	QPS	P99 latency for TiKV	P99 latency for the client
10	3,110	43 ms	115 ms
100	314	450 ms	7,888 ms
200	158	480 ms	13,180 ms
500	63	500 ms	23,630 ms
1,000	31	504 ms	34,693 ms
1,500	8	507 ms	50,220 ms

Read from leader in another data centor

Number of scan keys	QPS	P99 latency for TiKV	P99 latency for the client
10	3,110	43 ms	115 ms
100	314	450 ms	7,888 ms
200	158	480 ms	13,180 ms
500	63	500 ms	23,630 ms
1,000	31	504 ms	34,693 ms
1,500	8	507 ms	50,220 ms

Read from follower in the same data centor

TiKV

General use cases

- **Note: Generally Follower Read is not helpful for performance**
  - TiDB is a multi Raft service, leaders are balanced on stores
- Use case 1: Build a HTAP system
  - Performing read on a column store (TiFlash) is much faster than on TiKV
- Use case 2: Read cross multiple data centers
  - Performing read on nearlier data center
- Use case 3: Scale out for the read performance
  - Elastically add a store in which places raft learners for improving read performance

Number of scan keys	QPS	P99 latency for TiKV	P99 latency for the client
10	18,865	31 ms	33 ms
100	4,233	58 ms	267 ms
200	2,321	94 ms	550 ms
500	1,008	130 ms	1,455 ms
1,000	480	330 ms	3,228 ms
1,500	298	450 ms	6,438 ms

Leader Read

Number of scan keys	QPS	P99 latency for TiKV	P99 latency for the client
10	15,021	31 ms	34 ms
100	3,859	62 ms	272 ms
200	2,186	120 ms	560 ms
500	947	243 ms	1,305 ms
1,000	450	480 ms	3,189 ms
1,500	277	763 ms	5,058 ms

Follower Read

Thanks!