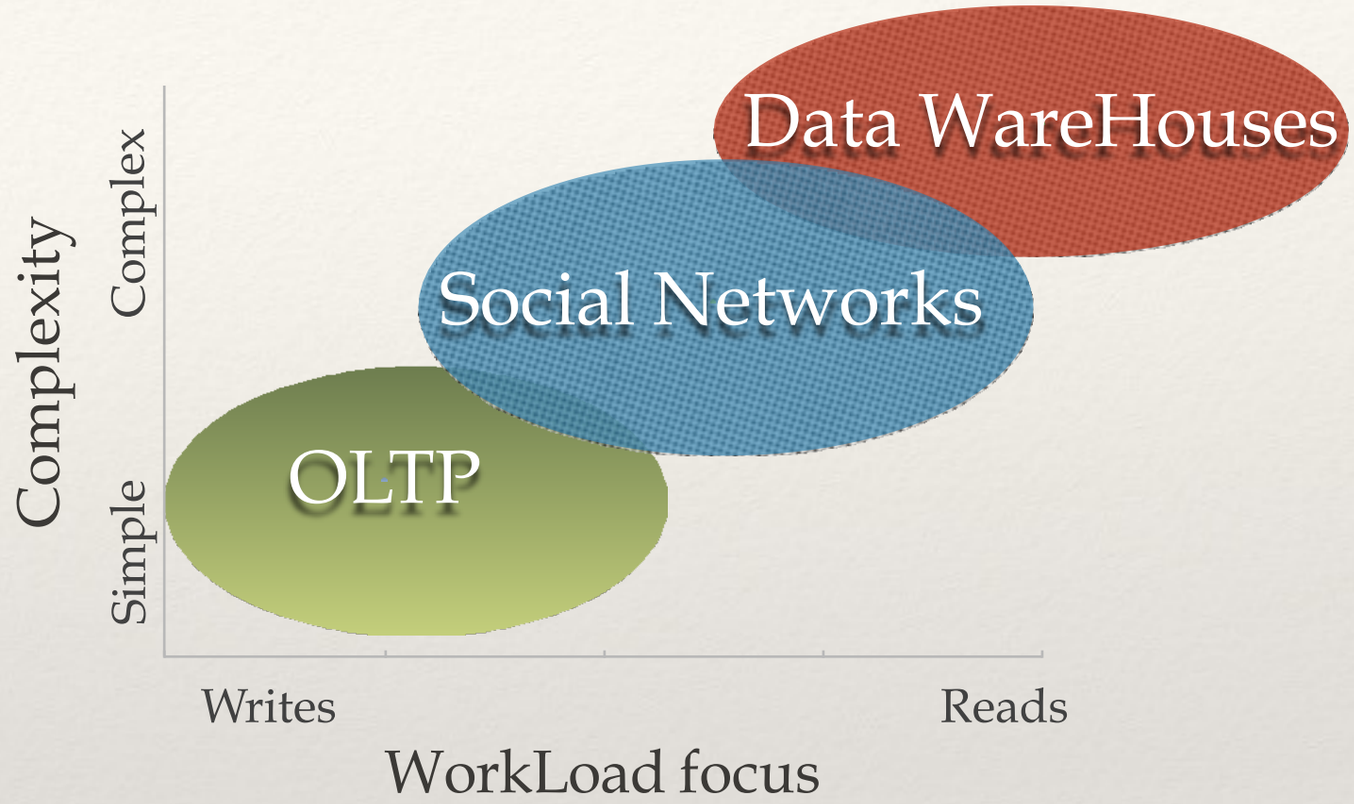
NewSQL Overview

History of SQL

- Relational Model in 1970
  - disk-oriented
  - rows
  - sql



- "One size fits all" doesn't work:
  - Column-oriented data warehouses for OLAP.
  - Key-Value storages, Document storages

Column-oriented DBMS

- Store content by column rather than by row

John	Smith	20
Joe	Smith	30
Alice	Adams	50

John:001; Joe:002; Alice:003.

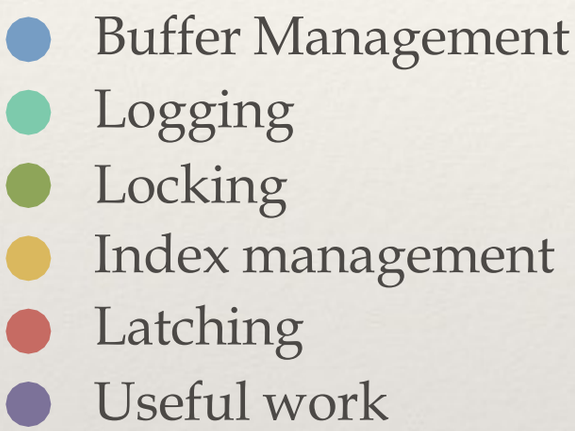
Smith:001,002; Adams:003.

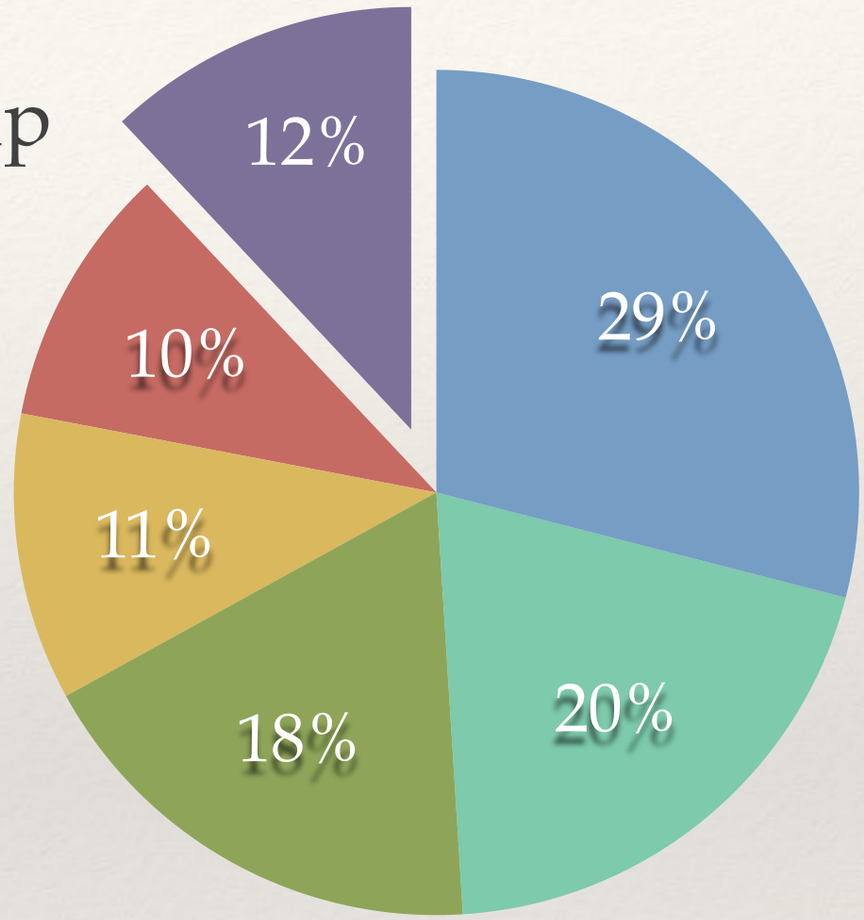
20:001; 30:002; 50:003.

- Efficient in hard disk access
- Good for sparse and repeated data
- Higher data compression
- More reads/writes for large records with a lot of fields
- Better for relatively infrequent writes, lots of data throughput on reads (OLAP, analytic requests).

Traditional DBMS overheads

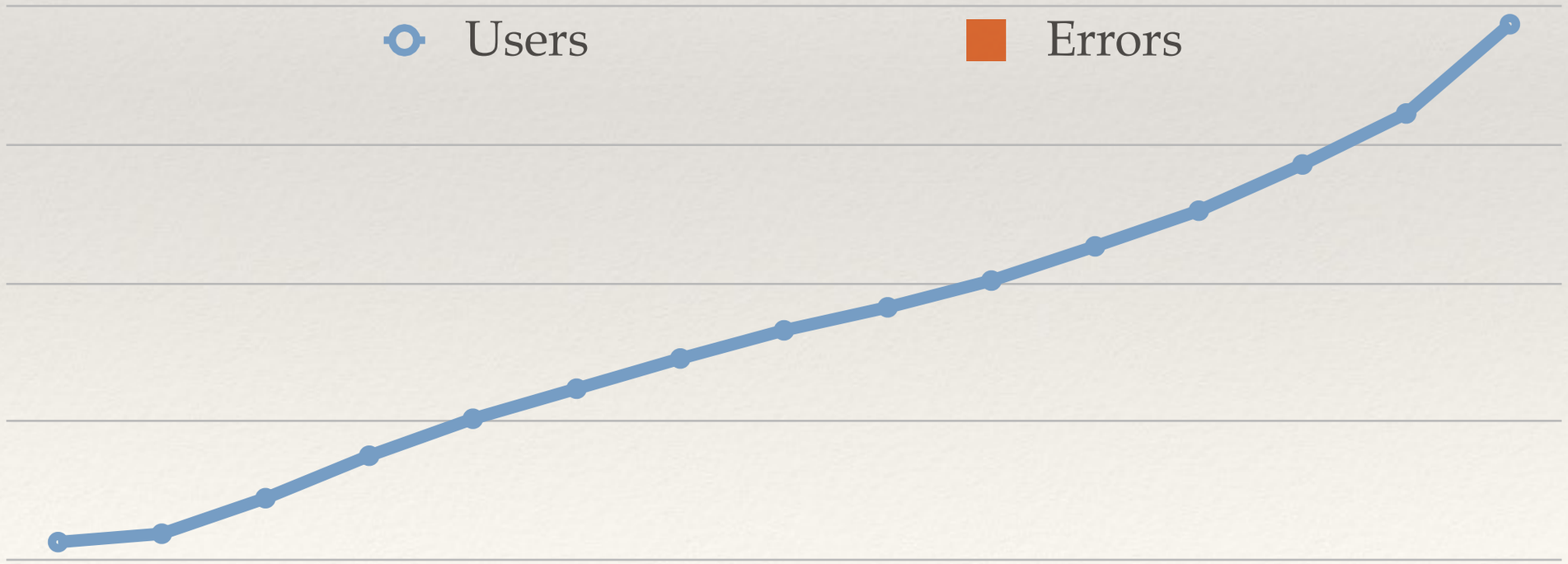
by Stonebraker & research group





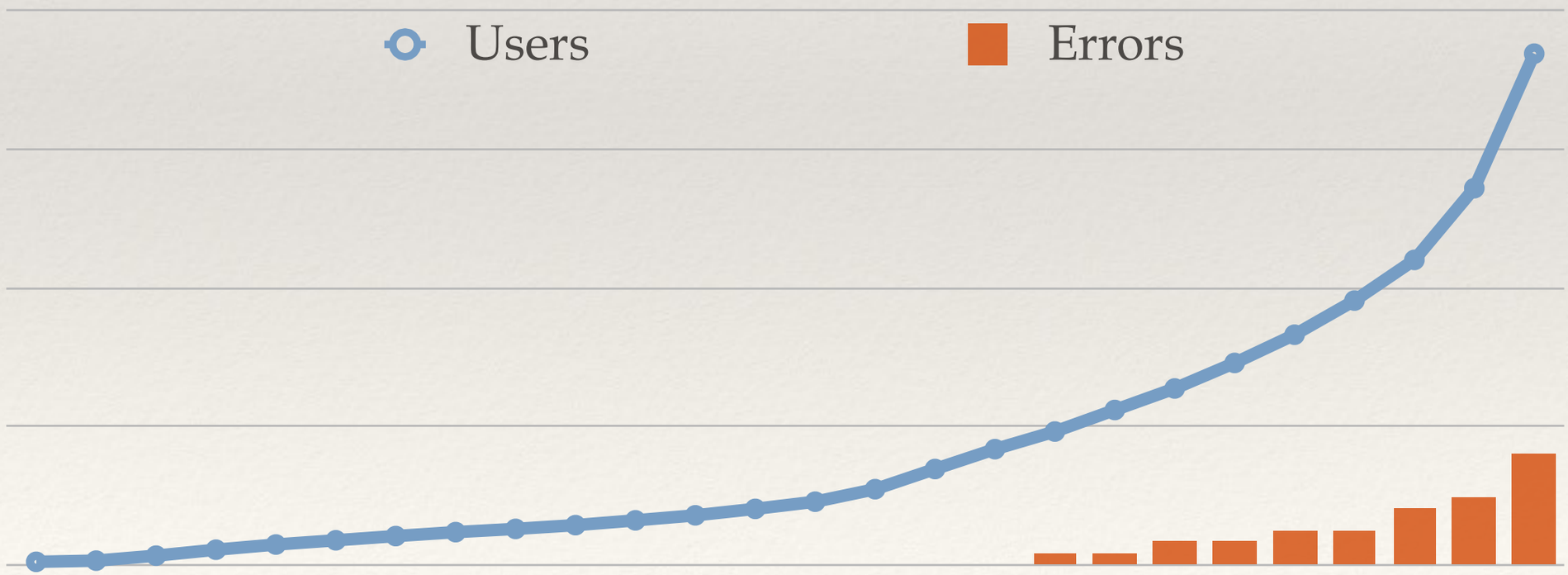
Startups lifecycle

- Start: no money, no users, open source



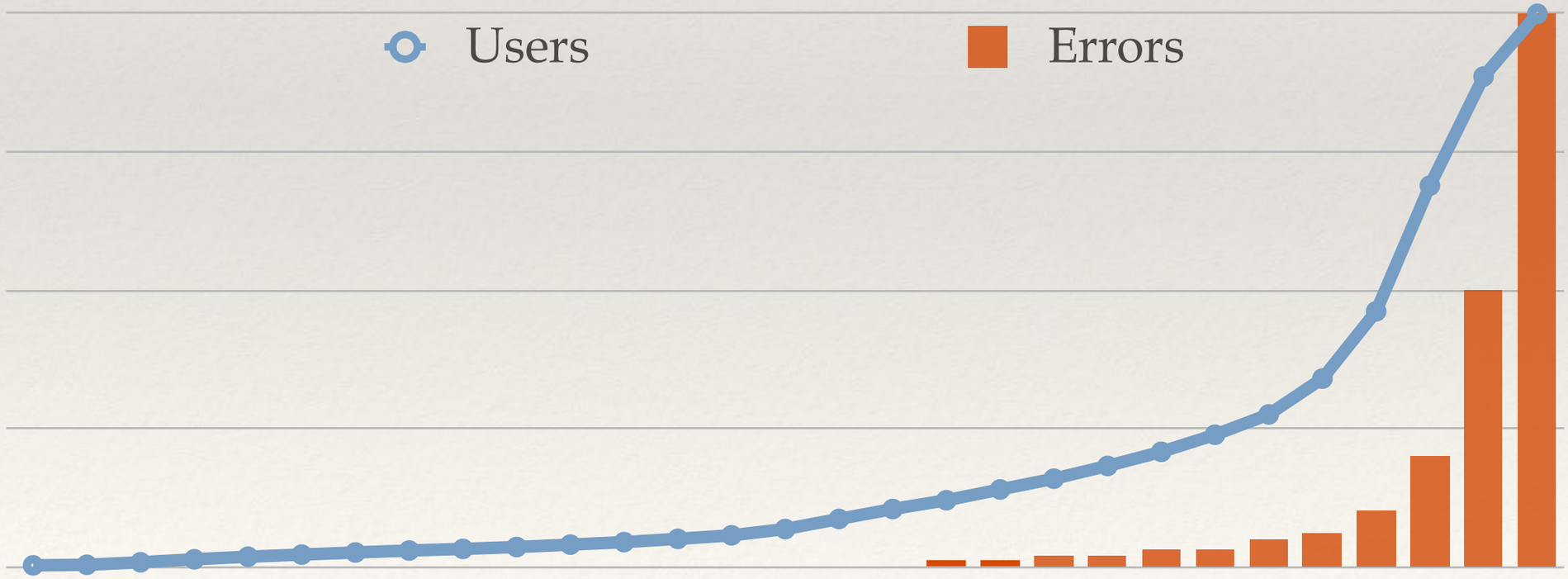
Startups lifecycle

- Start: no money, no users, open source
- Middle: more users, storage optimization



Startups lifecycle

- Start: no money, no users, open source
- Middle: more users, storage optimization
- Final: plenty of users, storage failure



Application-level sharding

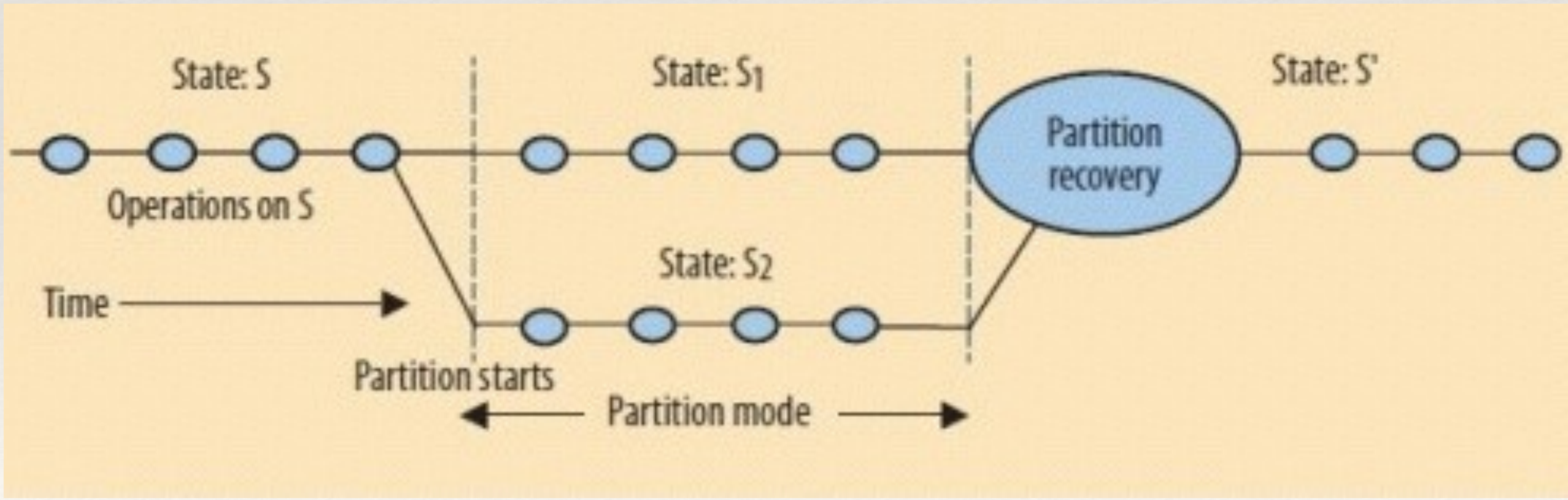
- Middleware + single-node DBMS
- Additional application-level logic
- Difficulties with cross-sharding transactions
- More servers to maintain and more components

NoSQL

- CAP: **c**onsistency, **a**vailability, **p**artitioning
- ACID: **a**tomicity, **c**onsistency, **i**solation, **d**urability

NoSQL

- ‘P’ in CAP is not discrete
- Managing partitions: detection, limitations in operations, recovery



NoSQL

- CAP: first ‘A’, then ‘C’: finer control over availability
- Horizontal scaling
- Not a “relational model”, custom API
- Schemaless
- Types: Key-Value, Document, Graph, …

NoSQL

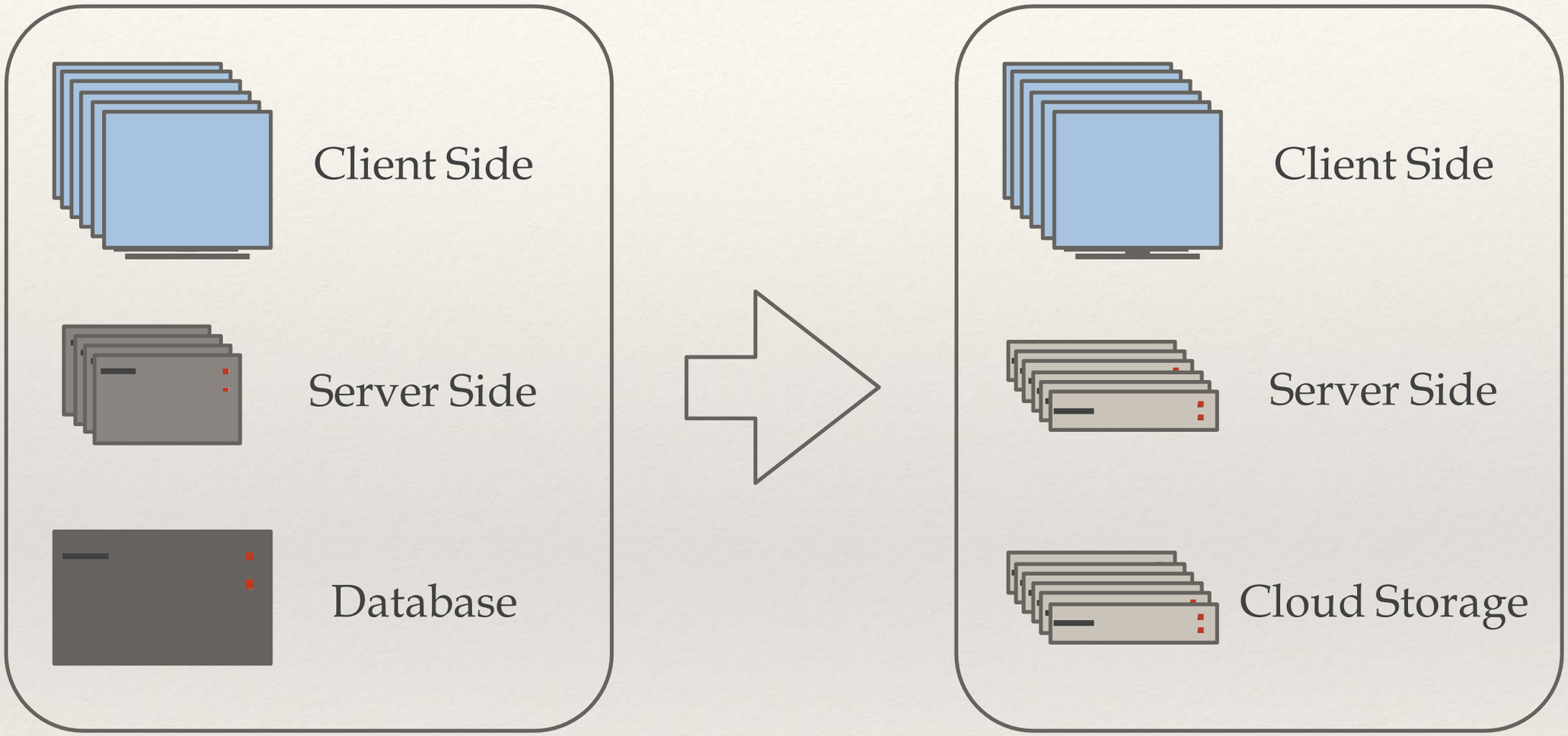
"NoSQL DBMSs cause their developers to spend too much time writing code to handle inconsistent data and that transactions provide a useful abstraction that is easier for humans to reason about."

Google

New requirements

- Large scale systems, with huge and growing data sets
- Information is frequently generated by devices
- High concurrency requirements
- Usually, data model with some relations
- Often, transactional integrity

Trends: architecture change



Consistency, transactions: Database

Storage optimization: Database

Scalability: Client Side

Consistency, transactions: Cloud

Storage optimization: Cloud

Scalability: All levels

NewSQL: definition

"A DBMS that provide the same scalable performance of NoSQL for online transaction processing (OLTP) read- write workloads while still maintaining ACID guarantees for transactions."

Andrew Pavlo & Matthew Aslett

NewSQL: definition

- *SQL as the primary interface*
- *ACID support for transactions*
- *Non-locking concurrency control*
- *High per-node performance*
- *Scalable, shared nothing architecture*

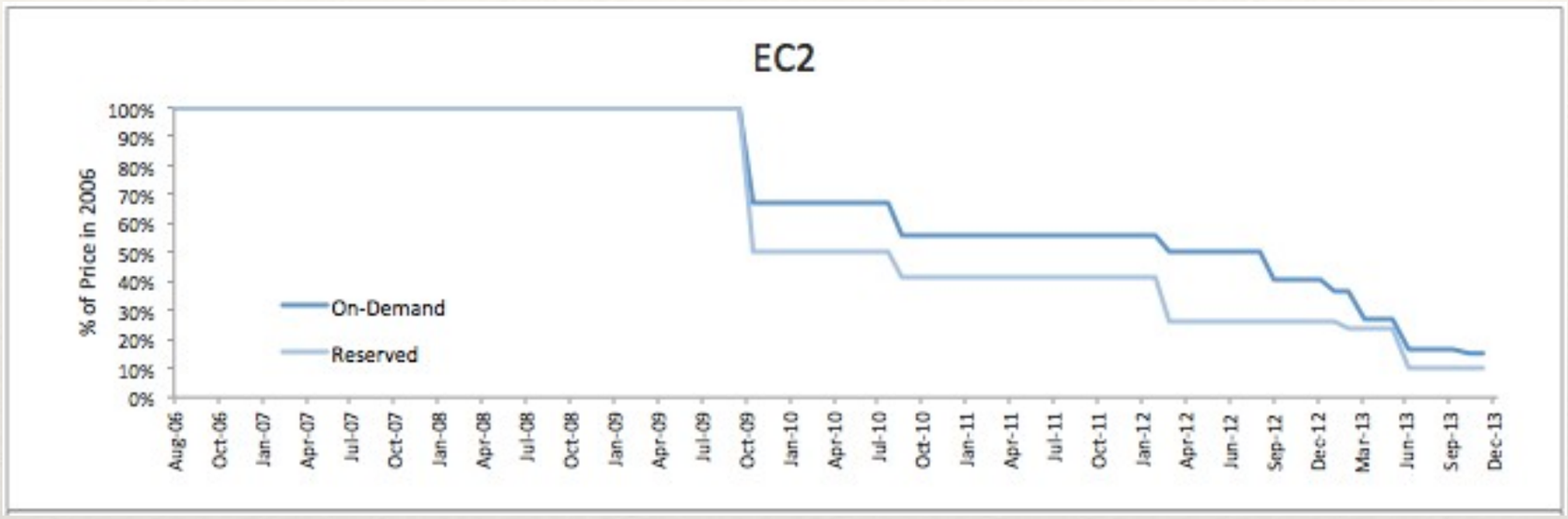
Michael Stonebraker

Shared nothing architecture

- No single point of failure
- Each node is independent and self-sufficient
- No shared memory or disk
- Scale infinitely
- Data partitioning
- Slow multi-shards requests

In-memory storage: price

Amazon price reduction

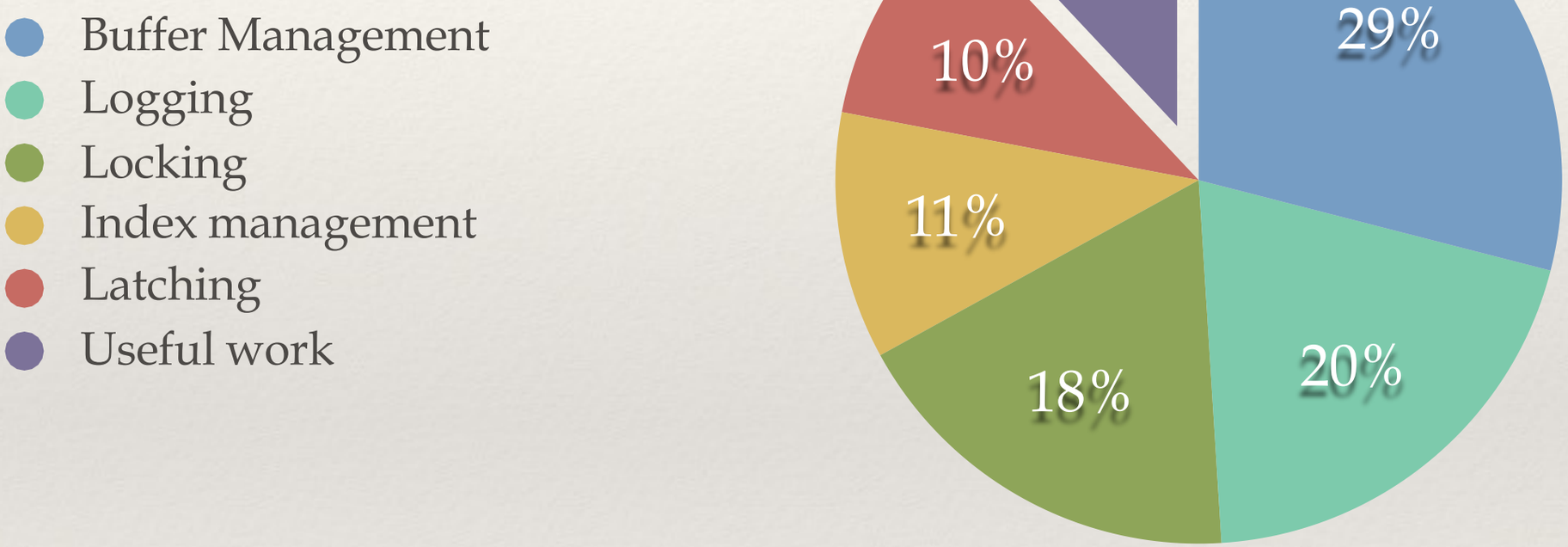


Current price for 1TB (~4 instances of 'r3.8xlarge' type)

	on-demand	3Y-reserved plan
per hour	11.2 $	3.9 $
per month	8.1K $	2.8K $
per year	97K $	33,7K $

Traditional DBMS overheads

by Stonebraker & research group



"Removing those overheads and running the database in main memory would yield orders of magnitude improvements in database performance"

In-memory storage

- High throughput
- Low latency
- No Buffer Management
- If serialized, no Locking or Latching

NewSQL: categories

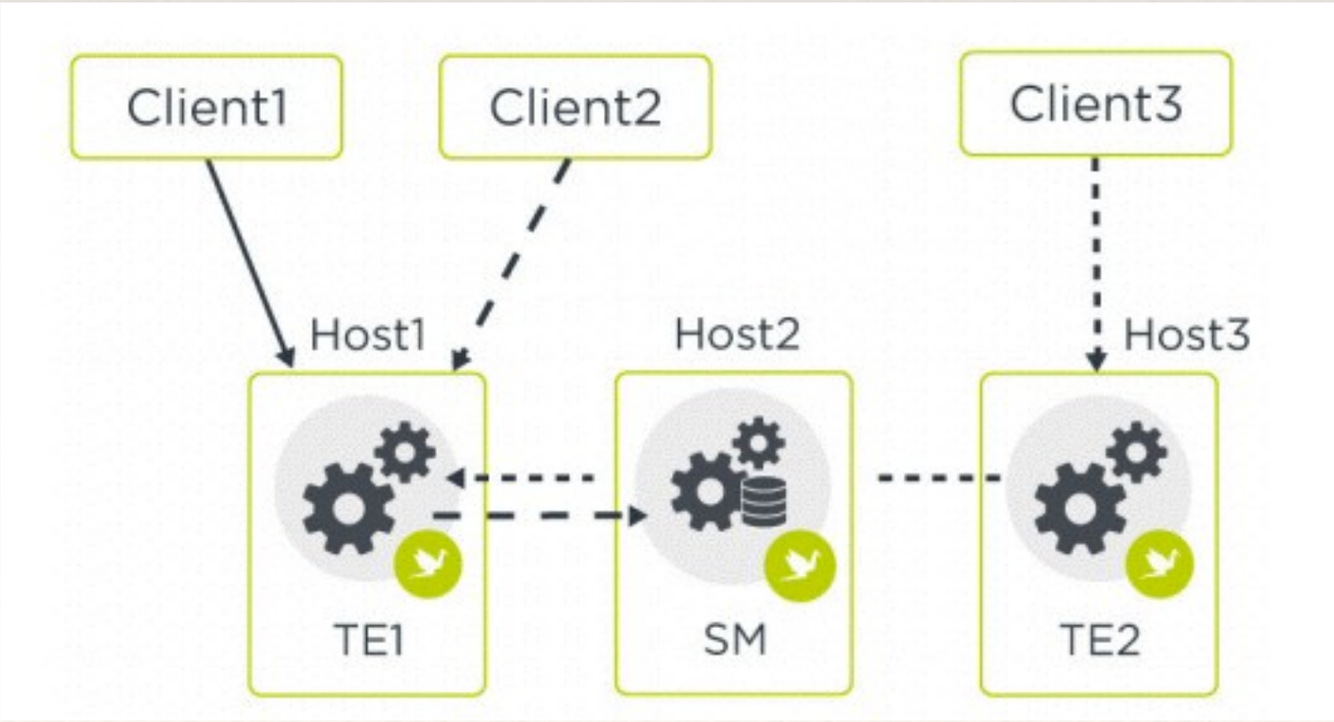
- **Novel new systems**: Spanner, NuoDB, VoltDB, Clustrix, MemSQL , SAP HANA, Hyper
- **Sharding Middleware**: dbShards, ScaleBase
- **Database-as-a-service**: Amazon Aurora, ClearDB

NuoDB

- Multi-tier architecture:
- Administrative: managing, stats, cli, web-ui
- Transactional: ACID except ‘D’, cache
- Storage: key-value store (‘D’ from ACID)

NuoDB

- Everything is an ‘Atom’
- Peer-to-peer communication, encrypted sessions
- MVCC + Append-only storage

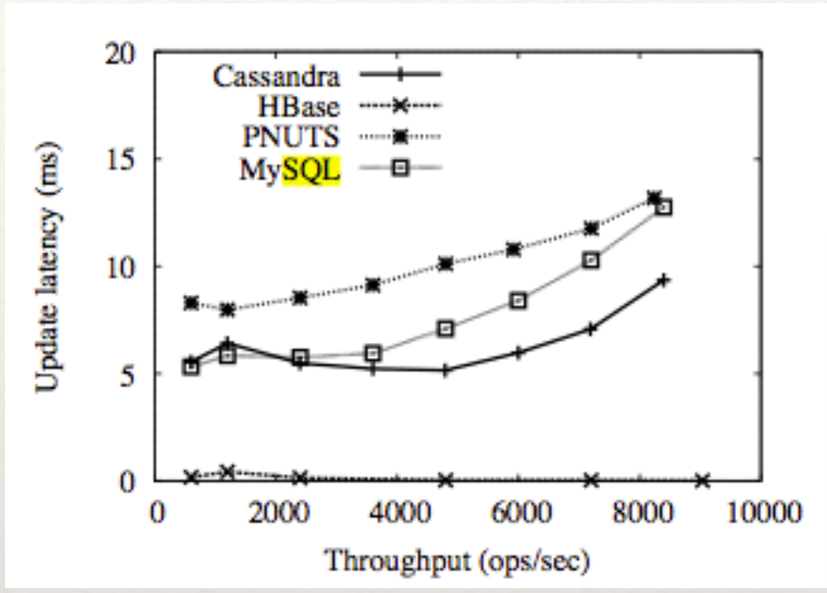


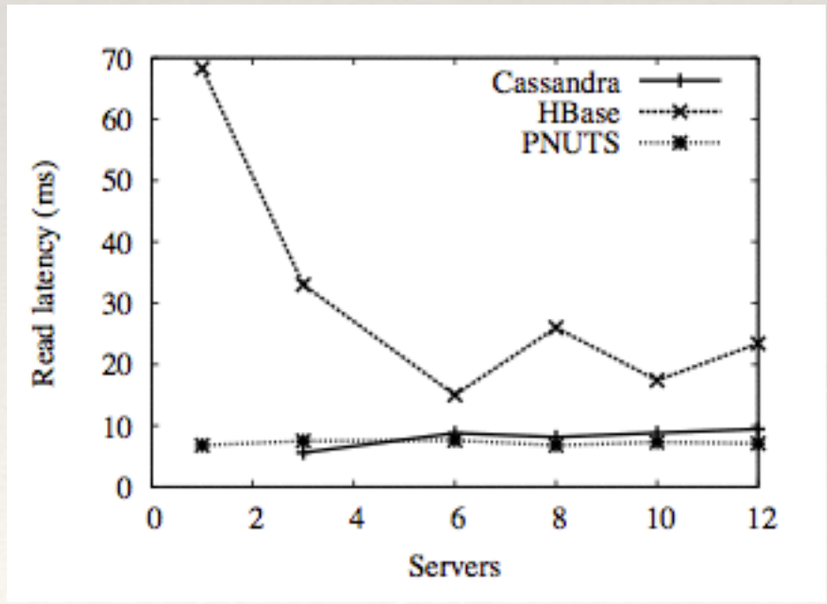
NuoDB: Partitioning

- Dynamic Partitioning
- Retrieve all of the atoms and broadcast changes
- Heuristics “Pin”

YCSB

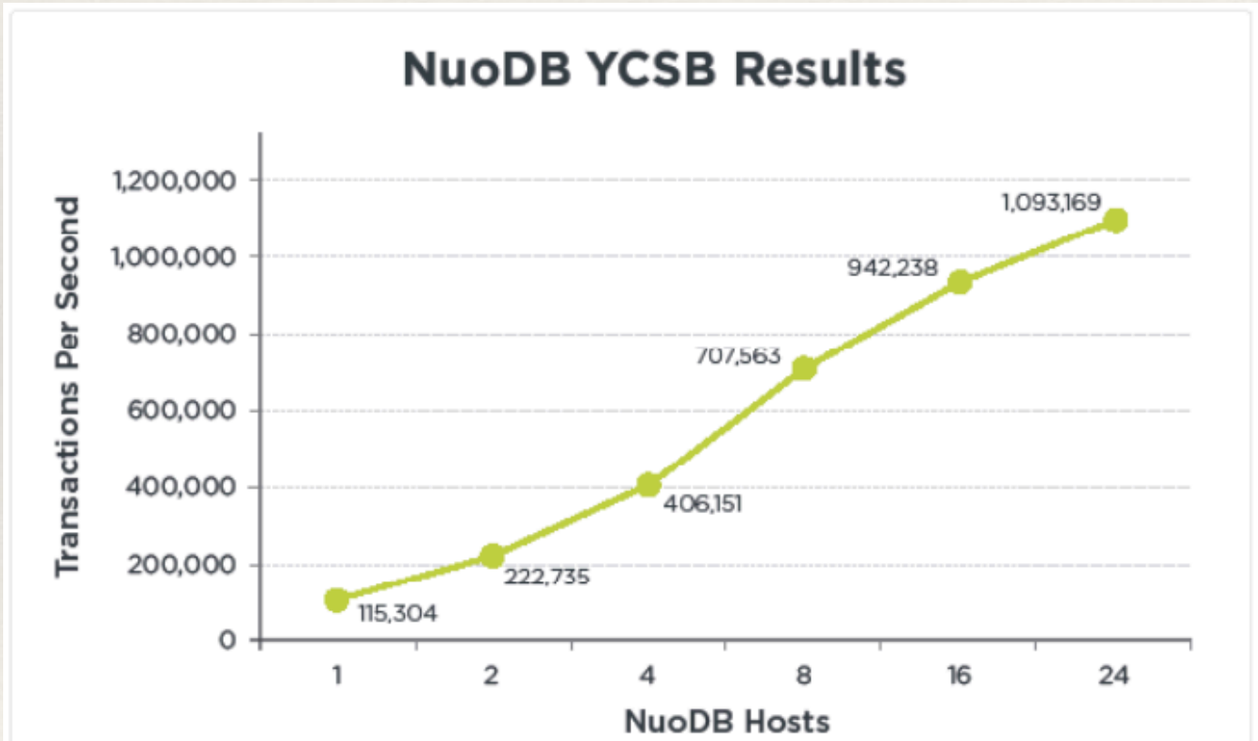
- Yahoo Cloud Serving Benchmark
- Key-value: insert/read/update/scan
- Measures:
  - Performance: latency/throughput
  - Scaling: elastic speedup

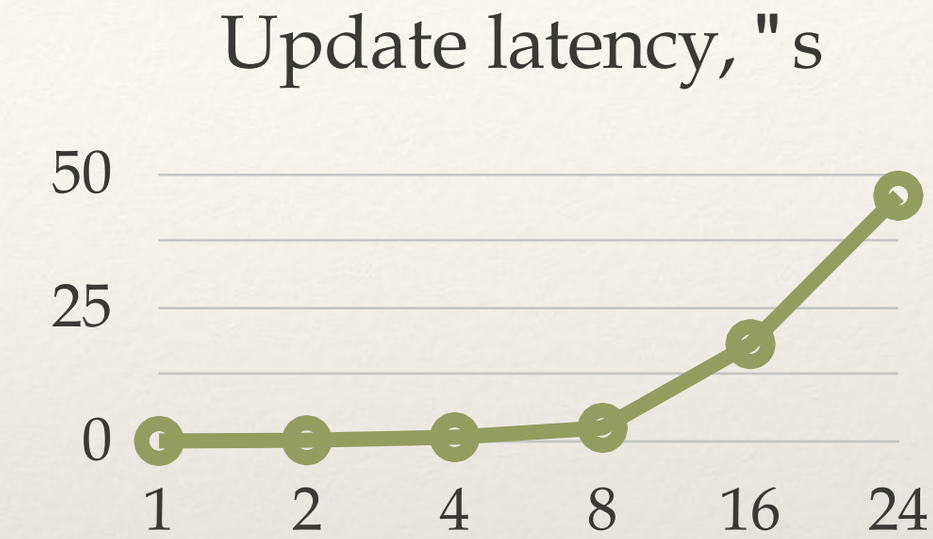


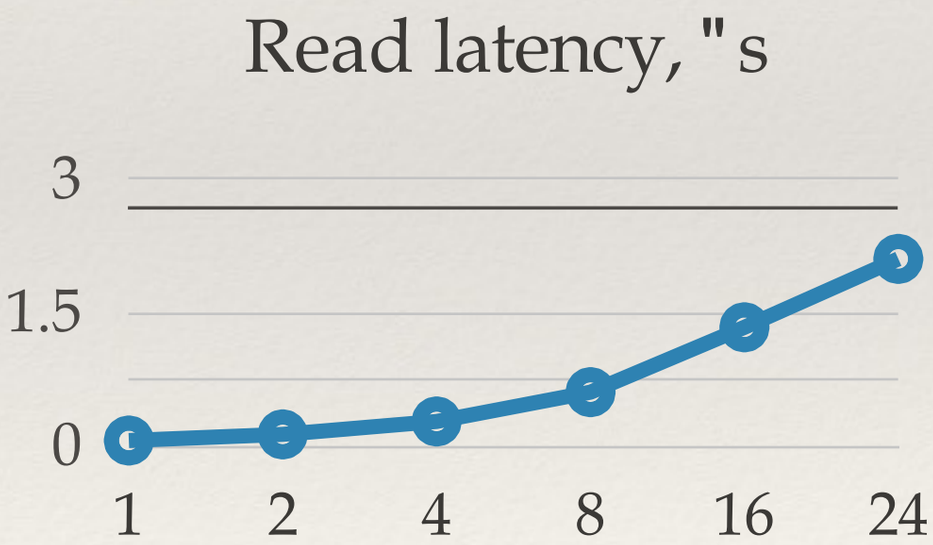


NuoDB: YCSB

Number of NuoDB Database Hosts	Transactions per Second (TPS)	Update Latency in µs	Read Latency in µs
1	115,304	91	66
2	222,735	222	135
4	406,151	788	280
8	707,563	2,538	609
16	942,238	18,205	1,337
24	1,093,169	45,988	2,105







5% updates, 95% reads

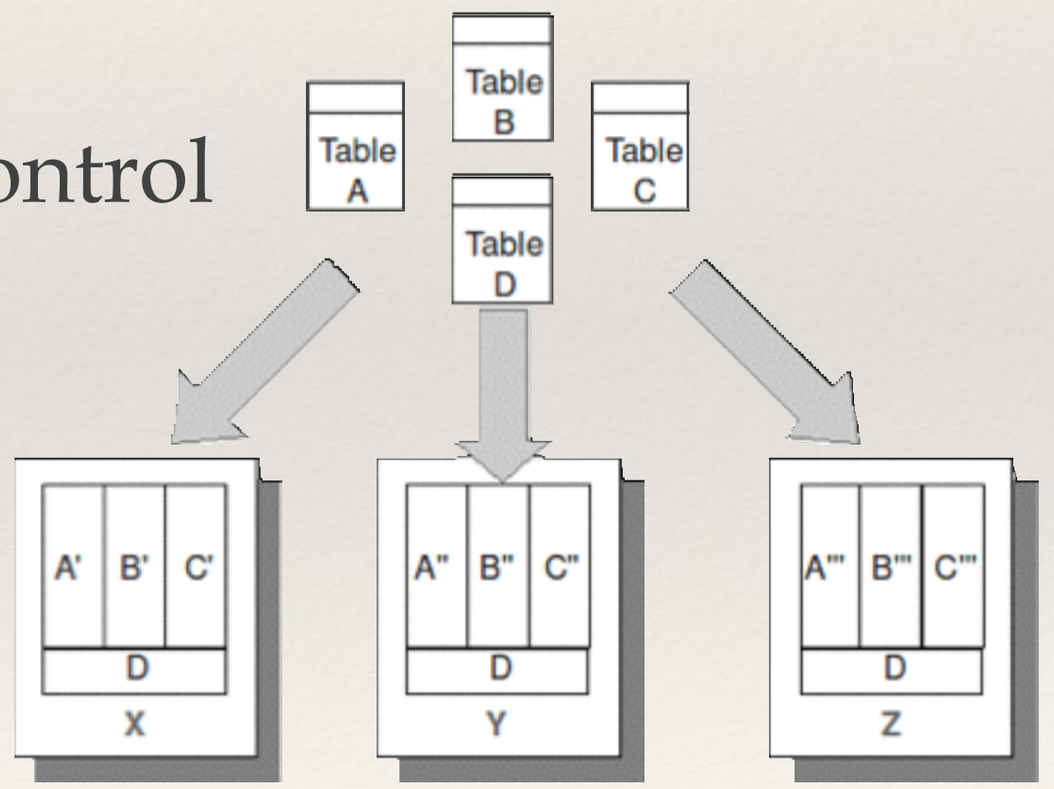
Hosts: 32GB, Xeon 8 cores, 1TB HDD, 1Gb LAN

VoltDB

- In-memory storage
- Stored procedure interface, async/sync proc execution
- Serializing all data access
- Horizontal partitioning
- Multi-master replication

VoltDB

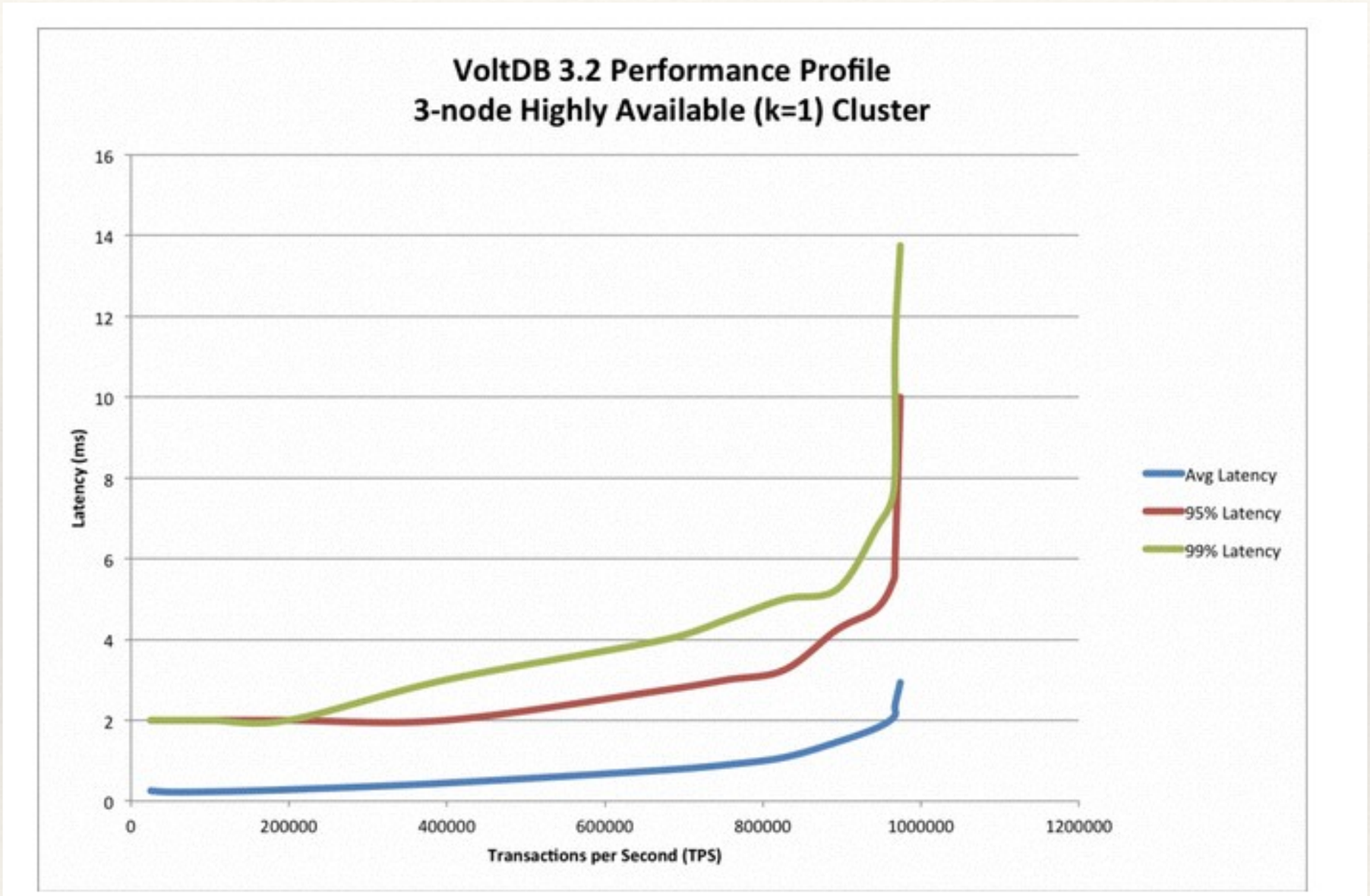
- Open-source, community edition is under GPLv3.
- Java + C++
- Partitioning and Replication control



VoltDB: CC

- Not MVCC, TO Concurrency Control
- Schedule transactions to execute **one-at-a-time** at each partition
- Partition-based concurrency control
- Hybrid architecture

VoltDB: key-value bench

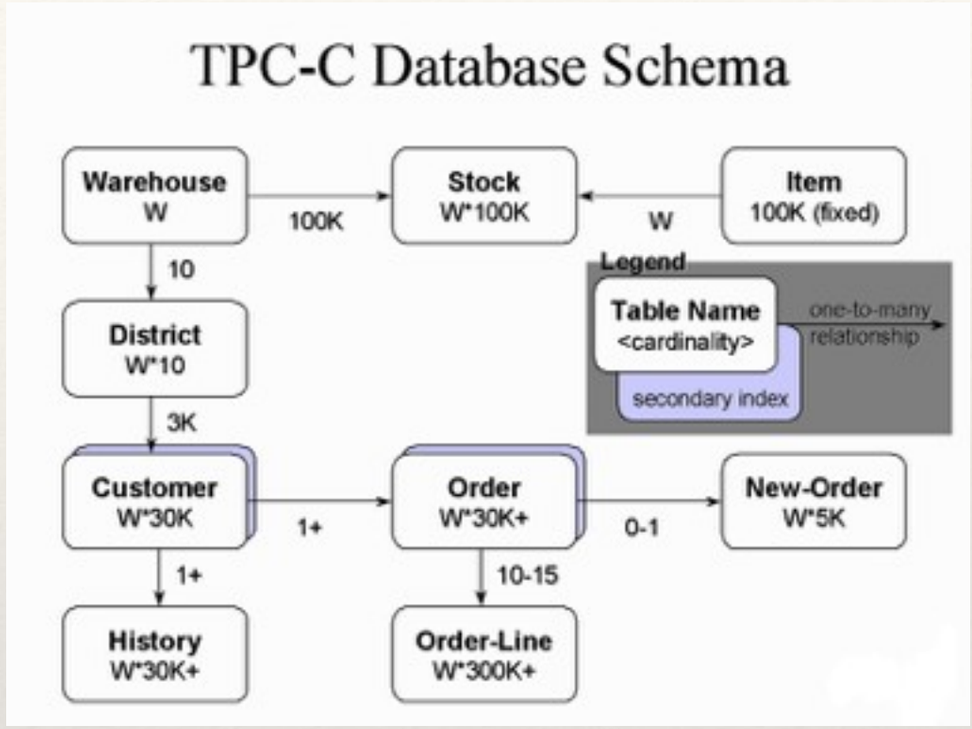


90%reads, 10%writes

3 nodes: 64GB, dual 2.93GHz intel 6 core processors

TPC-C

- Online Transaction Processing (OLTP) benchmark
- 9 types of tables
- 5 concurrent transactions of different complexity
- Productivity measured in “new-order transaction”

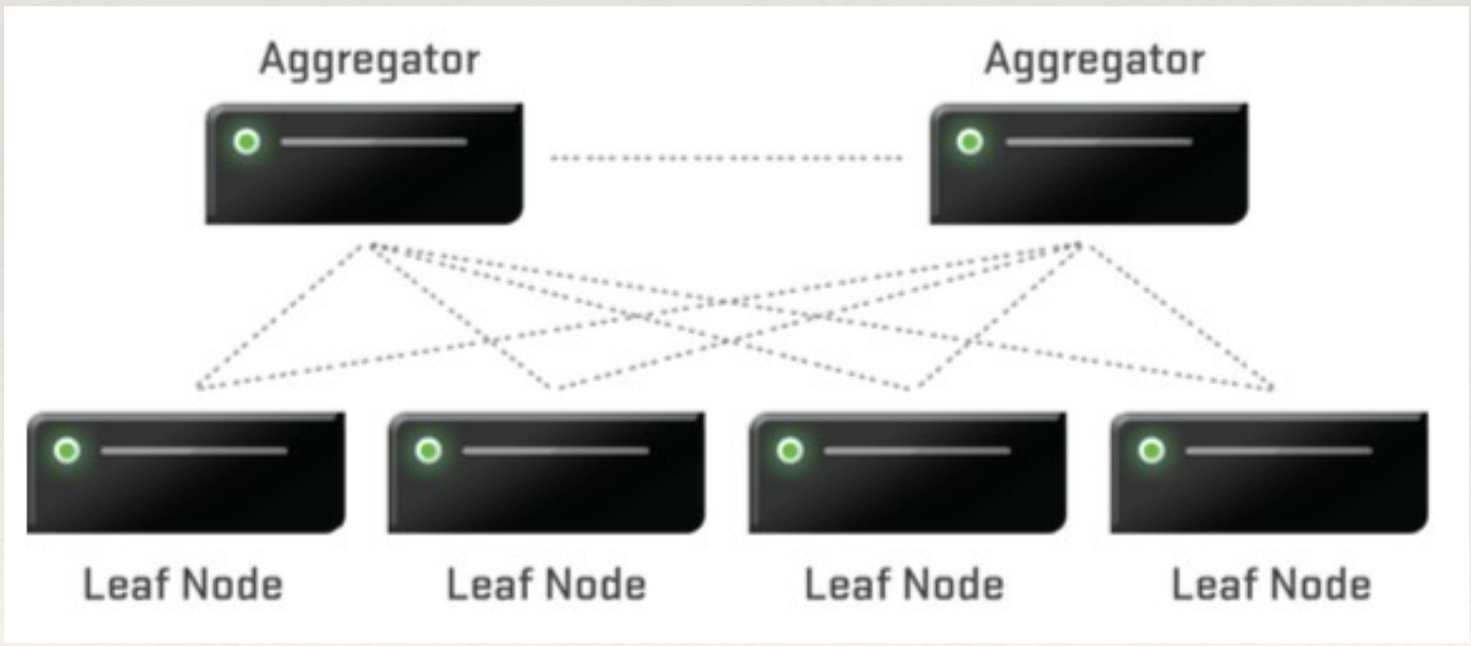


MemSQL

- In-Memory Storage for OLTP
- Column-oriented Storage for OLAP
- Compiled Query Execution Plans (+cache)
- Local ACID transactions (no global txs for distributed)
- Lock-free, MVCC
- Fault tolerance, automatic replication, redundancy (=2 by default)
- [Almost] no penalty for replica creation

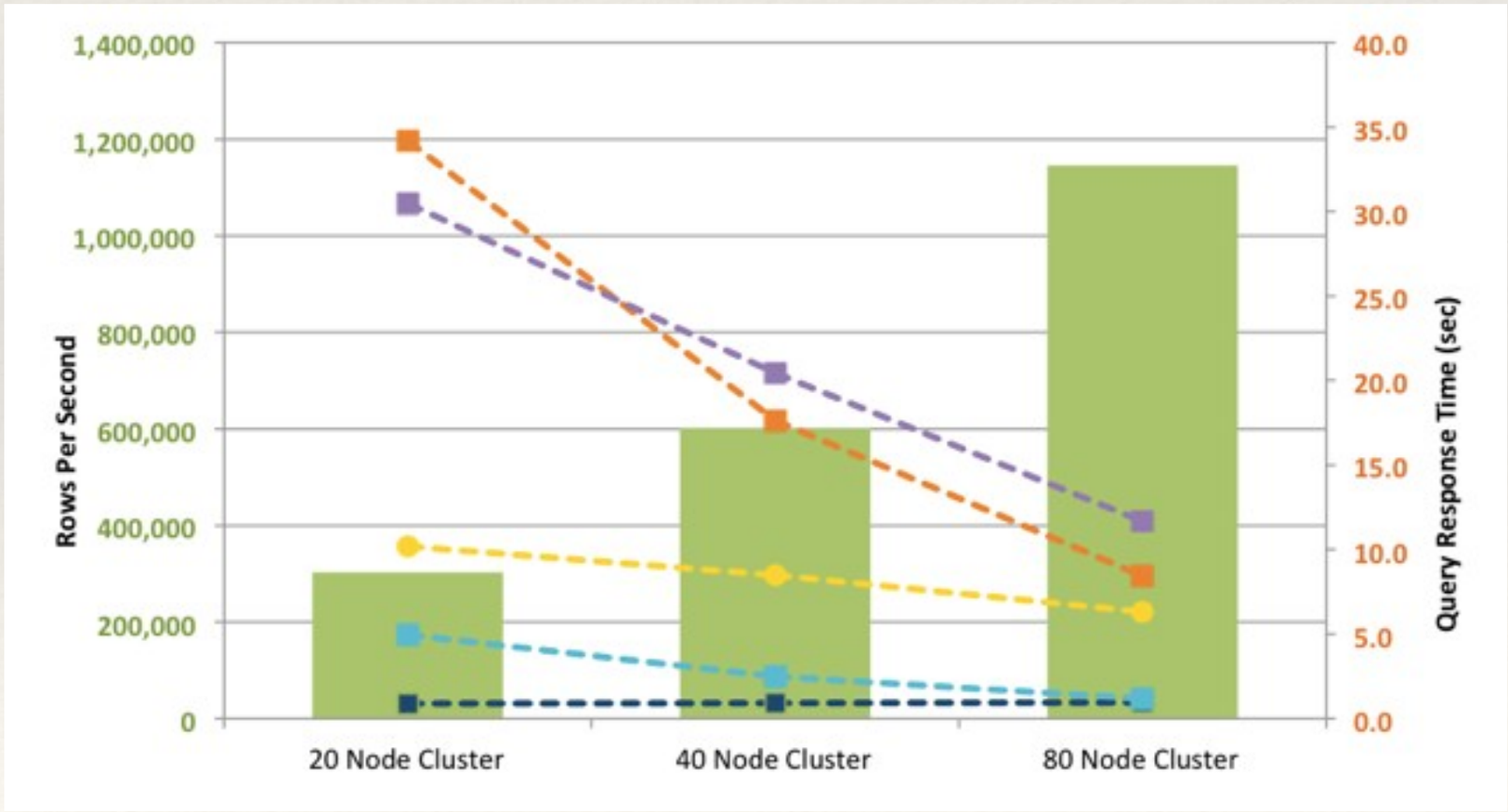
MemSQL

- Two-tiered shared-nothing architecture
  - Aggregators for query routing
  - Leaves for storage and processing
- Integration:
  - SQL
  - MySQL protocol
  - JSON API



MemSQL: Performance

- Adapted TPC-H
- OLAP Reads & OLTP writes simultaneously
- AWS EC2 VPC



Overview

	Max Isolation	Scalable	Open Source	Free to try	Language
PostgreSQL	S	?	Yes	Yes	C
NuoDB	CR	Yes	No	<5 domains	C++
VoltDB	S	Yes	Yes	Yes (wo HA)	Java/C++
ScaleDB	RC	Yes	No	?	?
ClustrixDB	RR	Yes	No	Trial (via email req)	C ?
FoundationDB	S	Yes	Partly	<6 processes	Flow(C++)
MemSQL	RC	Yes	No	?	C++

S: Serializable, RR: Repeated Read, RC: Read Committed, CR: Consistent Read

Conclusions

- NewSQL is an established trend with a number of options
- Hard to pick one because they're not on a common scale
- No silver bullet
- Growing data volume requires ever more efficient ways to store and process it

Spanner

What is Spanner ?

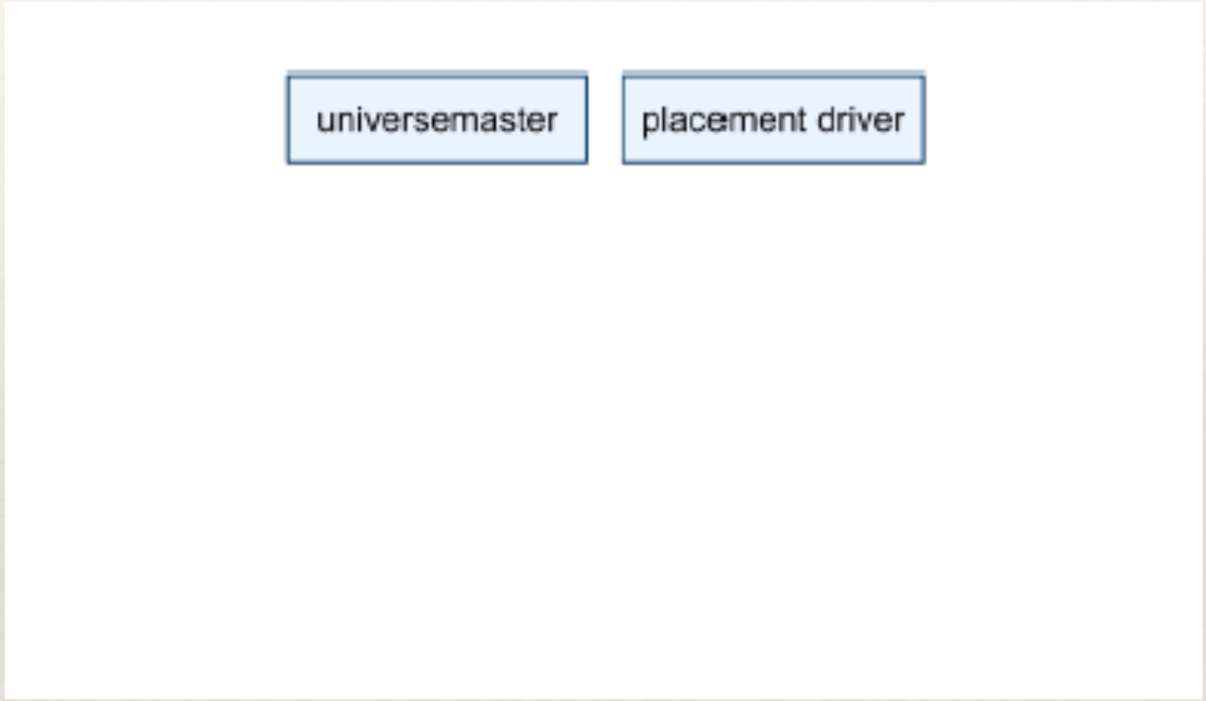
- Globally distributed multi-version database
  - General-purpose transactions (ACID)
  - SQL-like query language
  - Schematized semi-relational tables
- Currently running in production
  - Storage for Google's F1 adv.backend data
  - Replaced a sharded MySQL database

Overview

- Lock-free distributed read transactions
- Global external consistency of distributed transactions
  - Same as linearizability: if a transaction T_1 commits before another transaction T_2 starts, then T_1's commit timestamp is smaller than T_2's.
- Used technologies: concurrency control, replication, 2PC and 2PL
- The key technology: TrueTime service

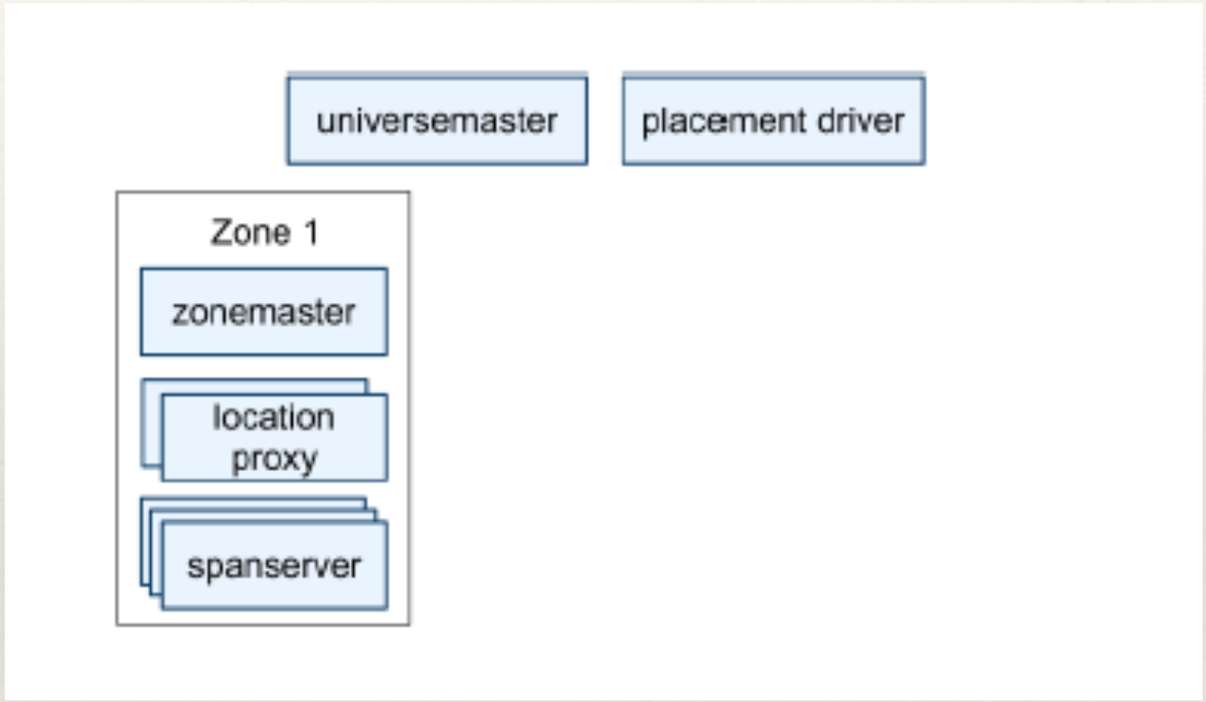
Spanner server organization

- A Spanner deployment is called an *universe*
- It have two singletons: the universe master and the placement driver



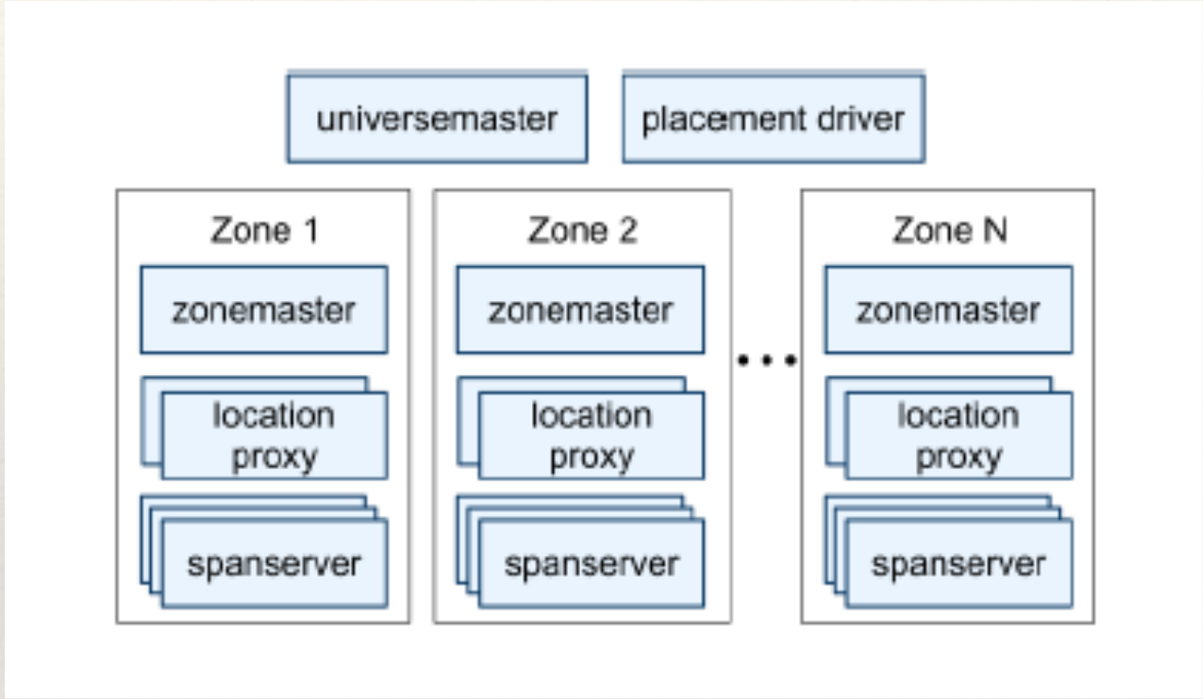
Spanner server organization

- A Spanner deployment is called an *universe*
- It have two singletons: the universe master and the placement driver
- Can have up to several thousands spanservers

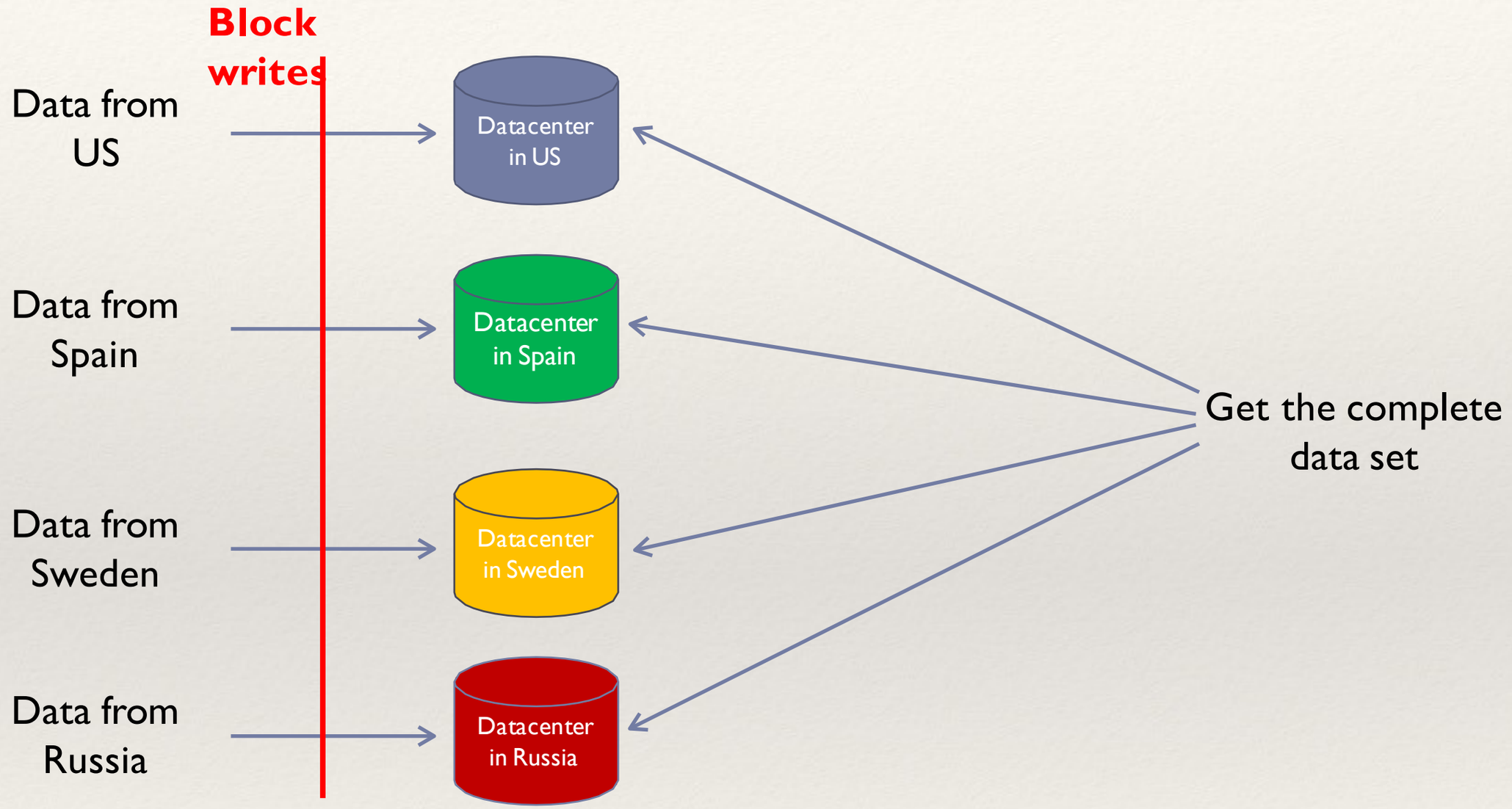


Spanner server organization

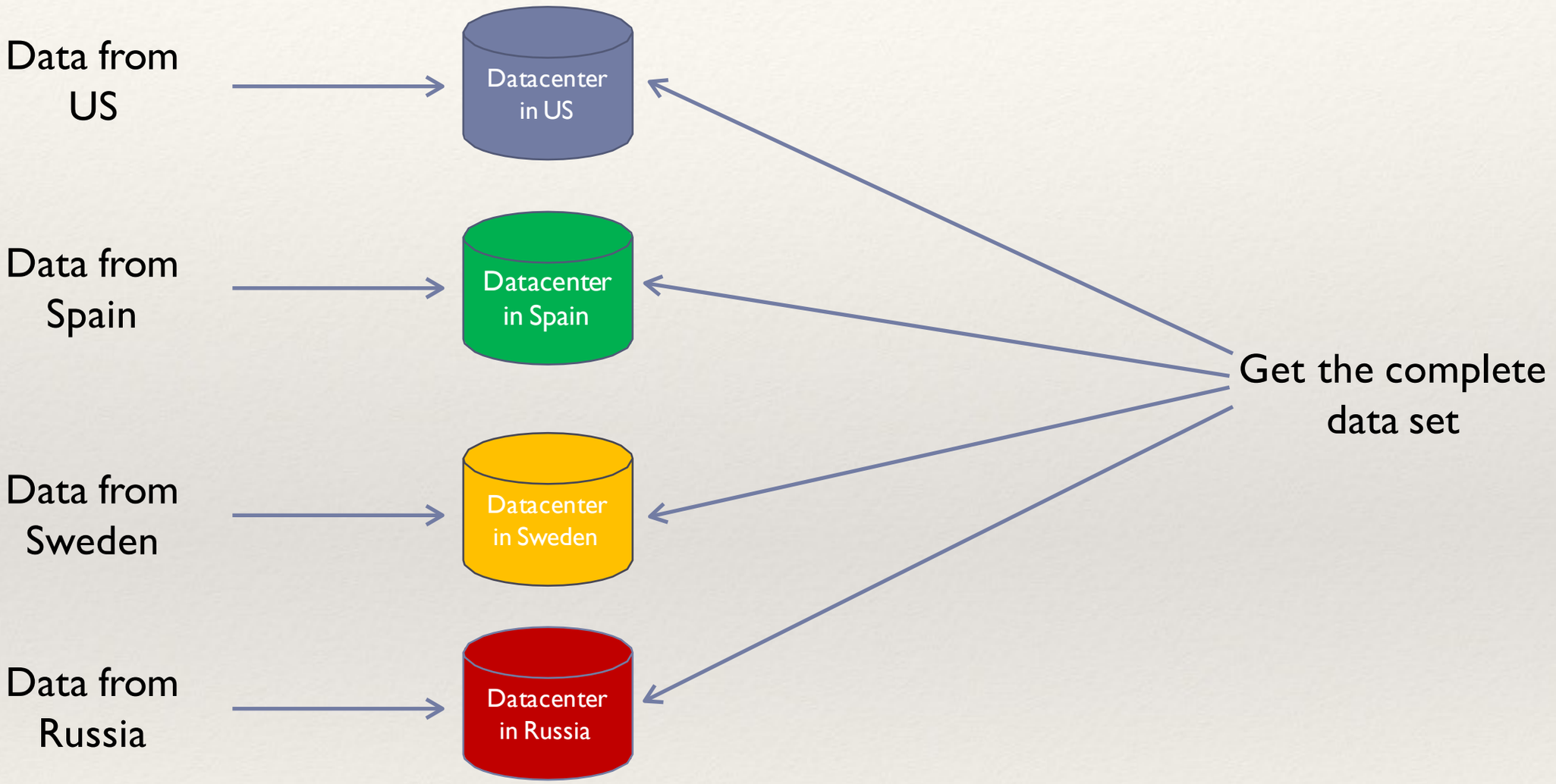
- A Spanner deployment is called an *universe*
- It have two singletons: the universe master and the placement driver
- Can have up to several thousands spanservers
- Organized as a set of zones



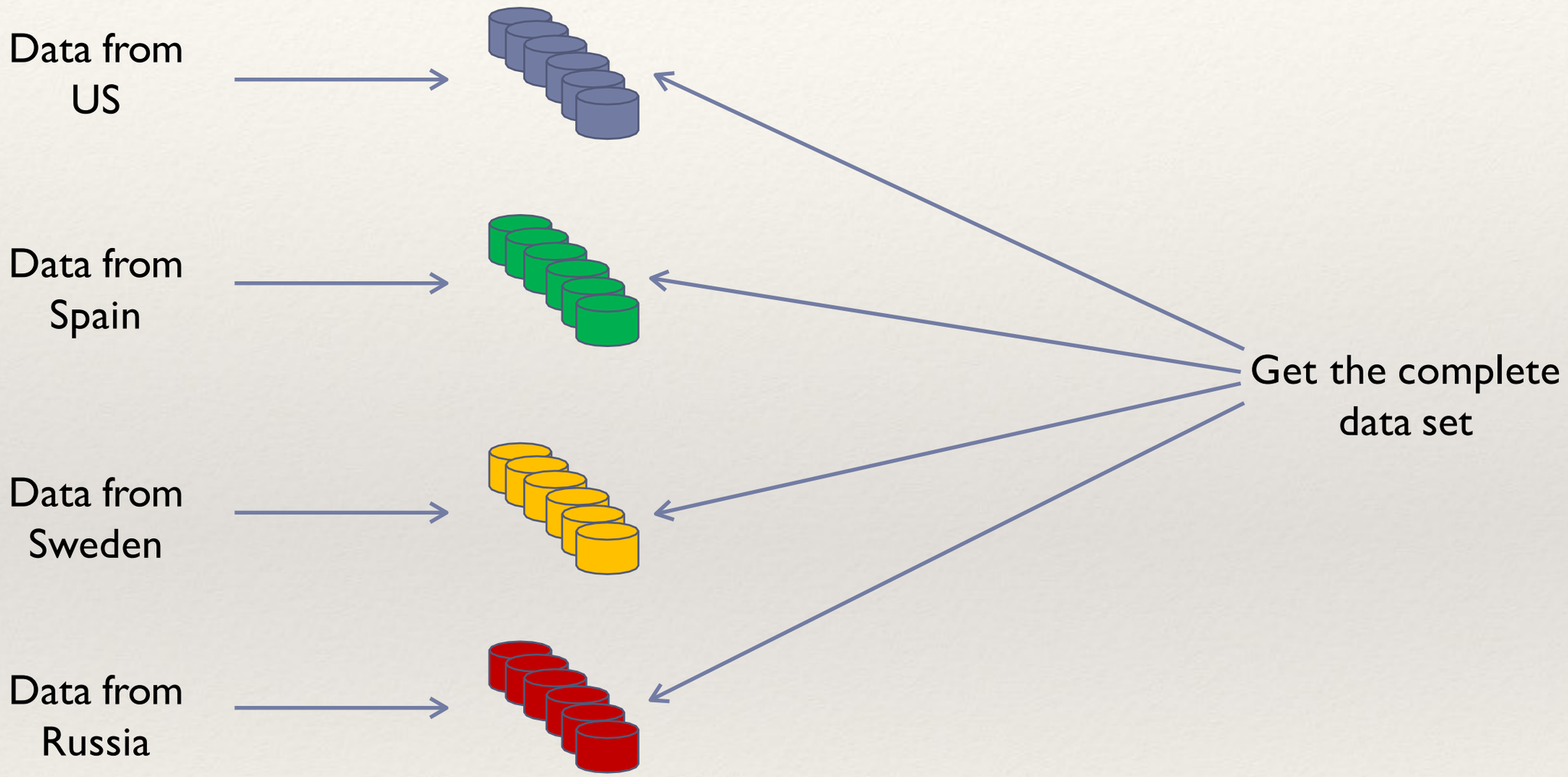
Serving data from multiple datacenters



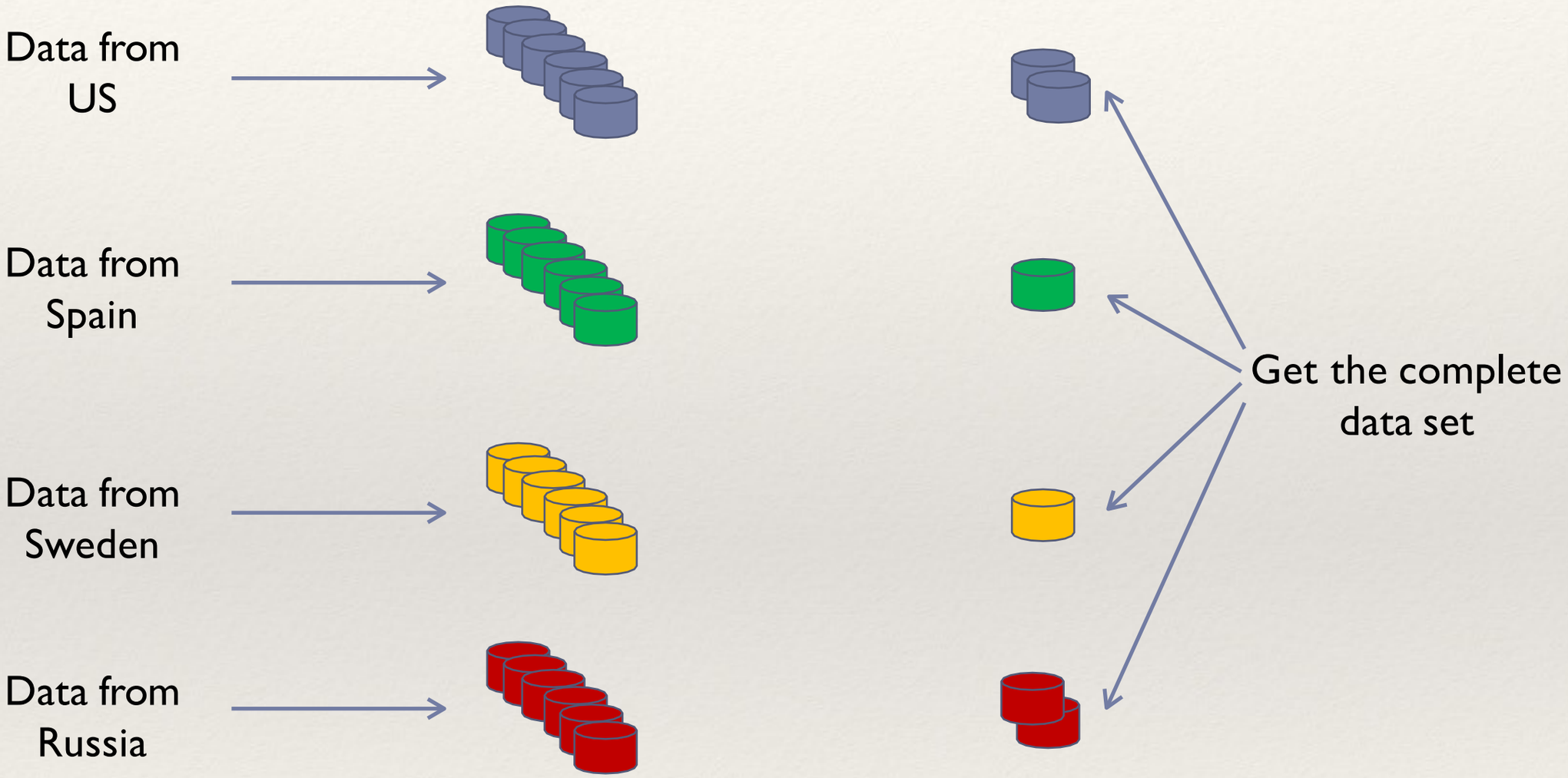
Serving data from multiple datacenters



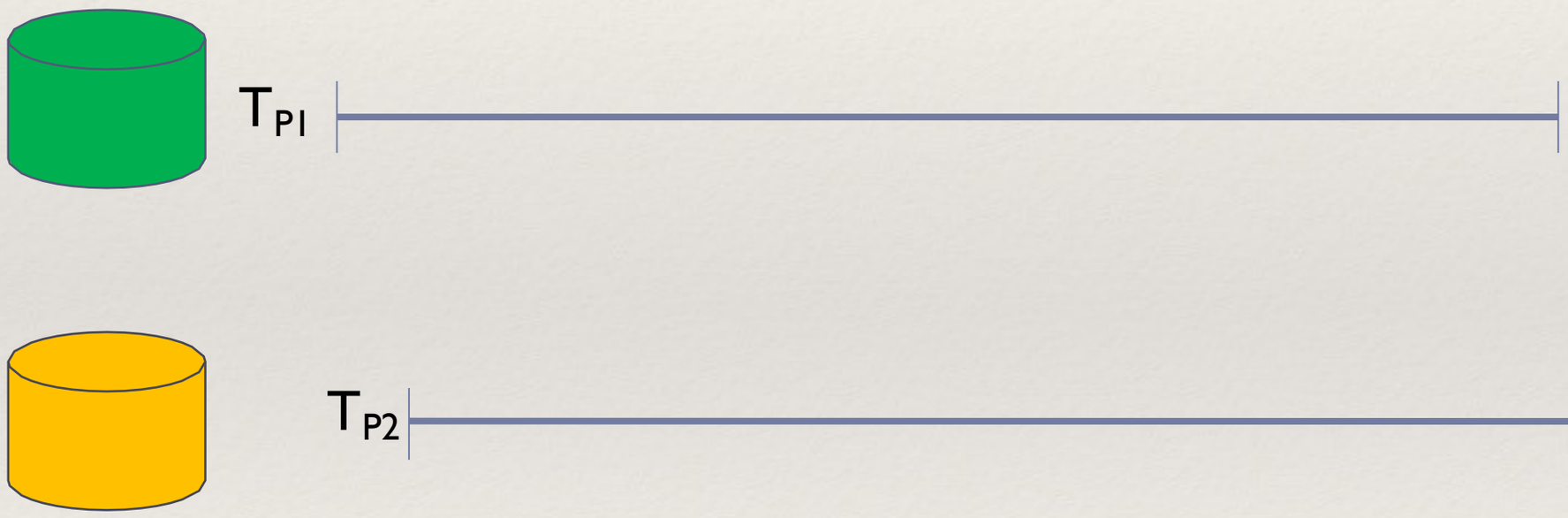
Serving data from multiple datacenters



Serving data from multiple datacenters

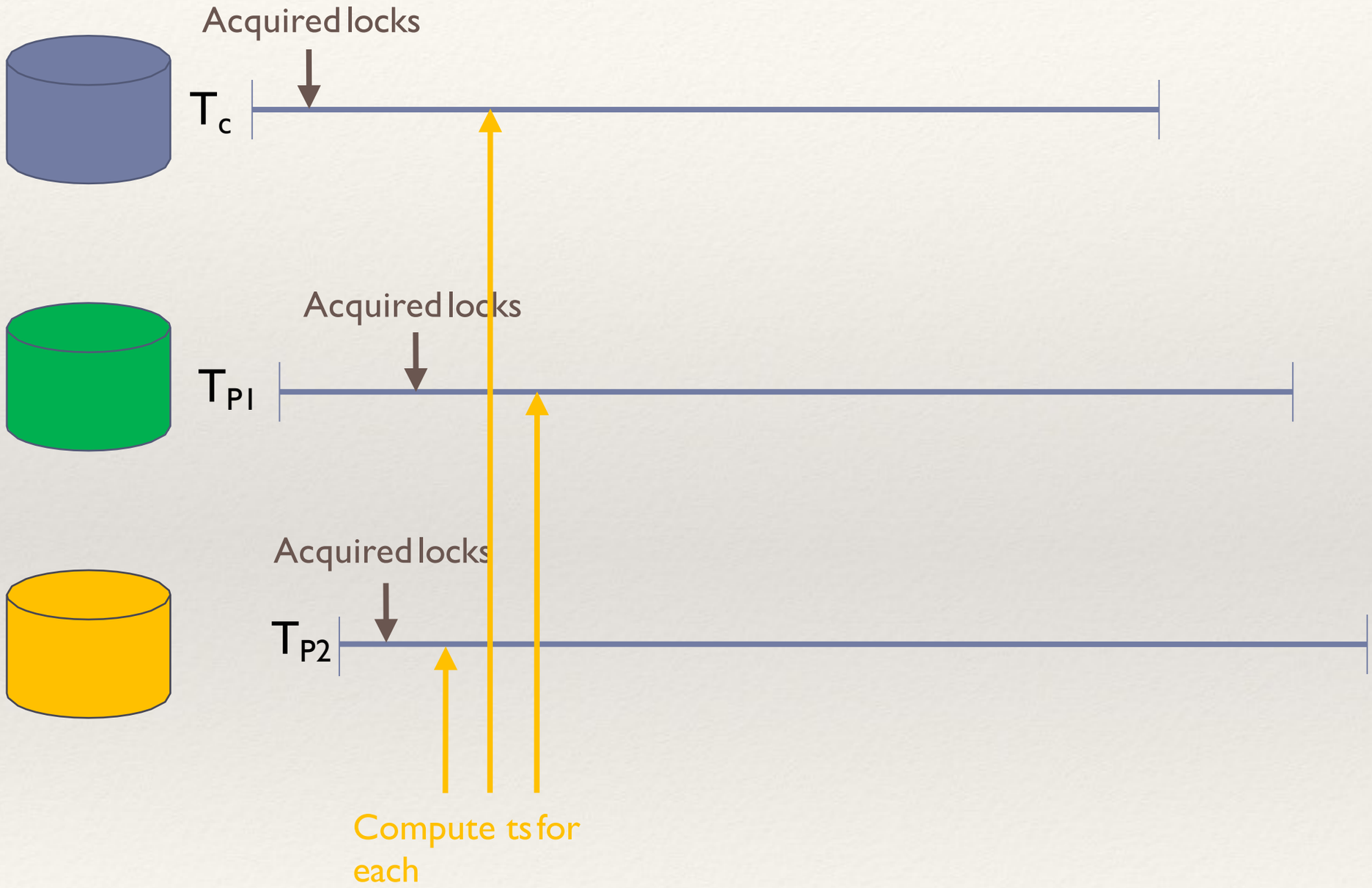


Transaction example



Transaction example

Transaction example



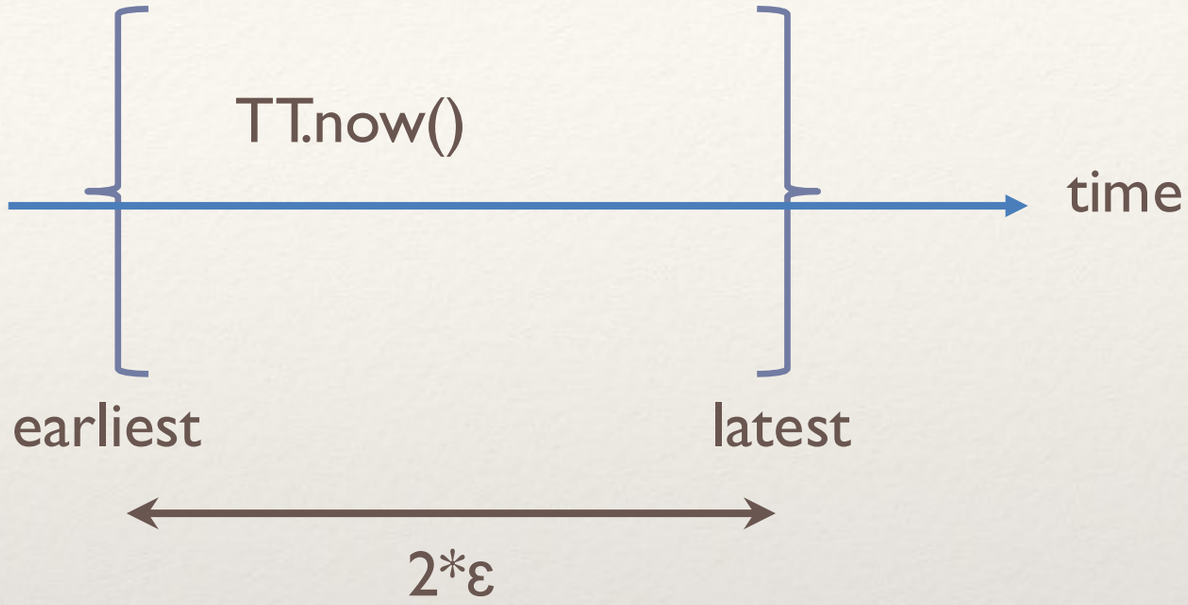
True Time API

- Provides an absolute time denoted as „Global wall-clock time“.
- Has bounded uncertainty ε,which varies between 1 to 7 ms over each poll interval
- Values derived from the worst case local-clock drift scenario

Method	Returns
TT.now()	TTinterval:[earliest, latest]
TT.after(t)	true if t has definitely passed
TT.before(t)	true if t has definitely not arrived

True Time API

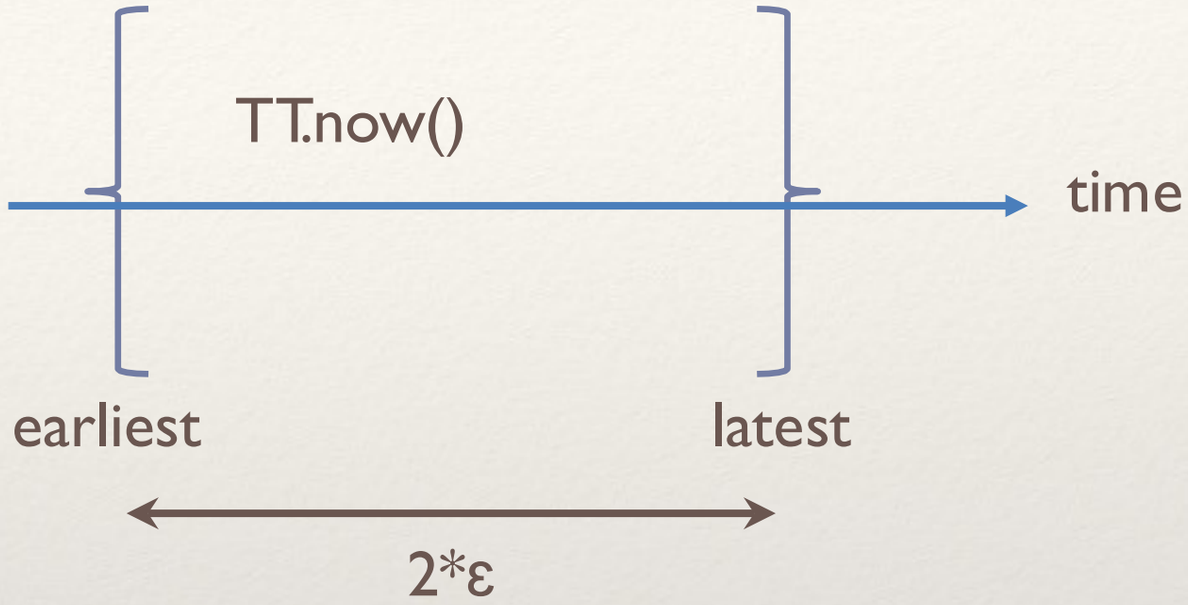
- Provides an absolute time denoted as „Global wall-clock time“.
- Has bounded uncertainty ε, which varies between 1 to 7 ms over each poll interval
- Values derived from the worst case local-clock drift scenario



Method	Returns
TT.now()	TTinterval:[earliest, latest]
TT.after(t)	true if t has definitely passed
TT.before(t)	true if t has definitely not arrived

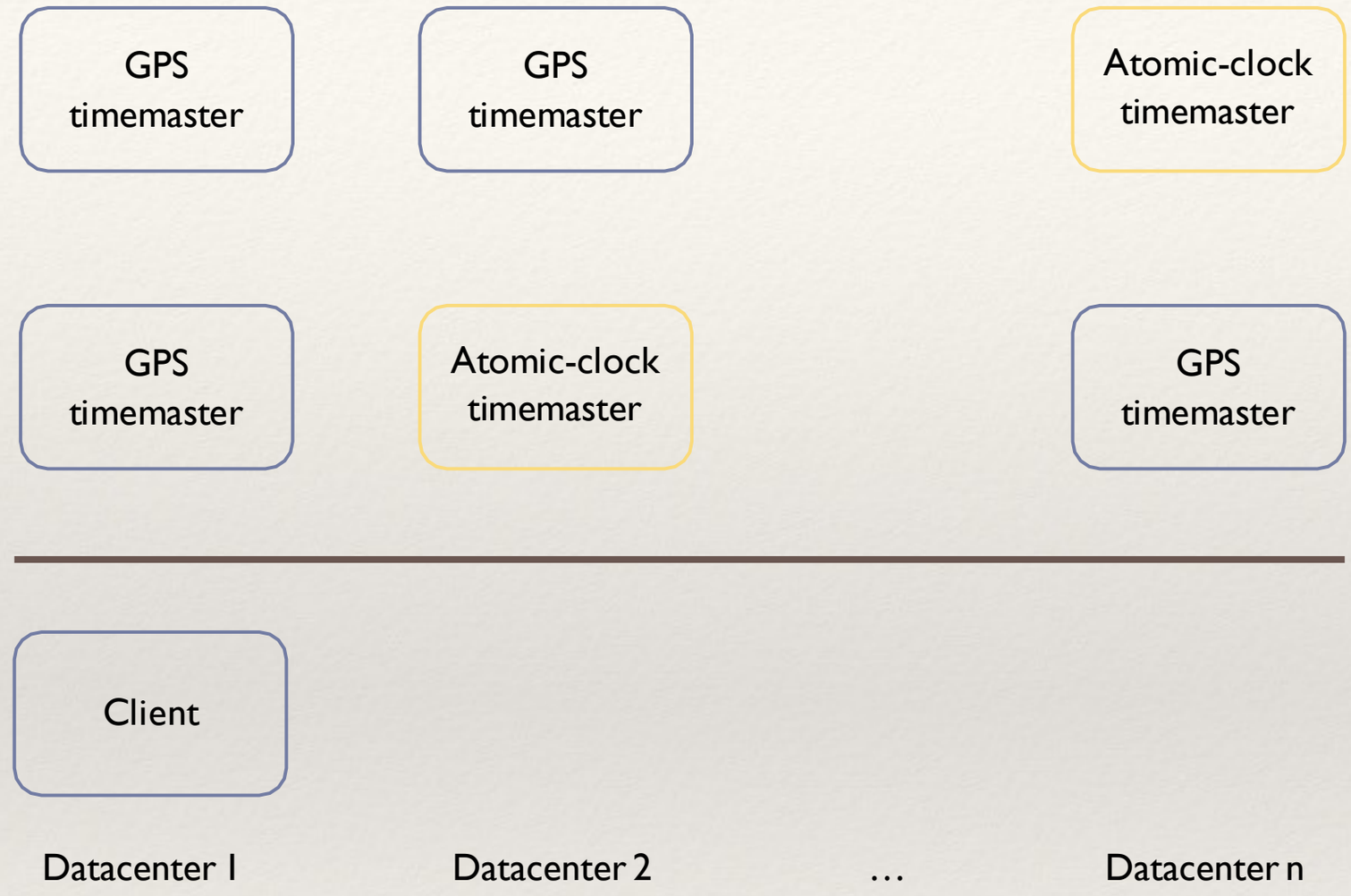
True Time API

- Provides an absolute time denoted as „Global wall-clock time“.
- Has bounded uncertainty ε, which varies between 1 to 7 ms over each poll interval
- Values derived from the worst case local-clock drift scenario
- Magic number: 200 µs/s



Method	Returns
TT.now()	TTinterval:[earliest, latest]
TT.after(t)	true if t has definitely passed
TT.before(t)	true if t has definitely not arrived

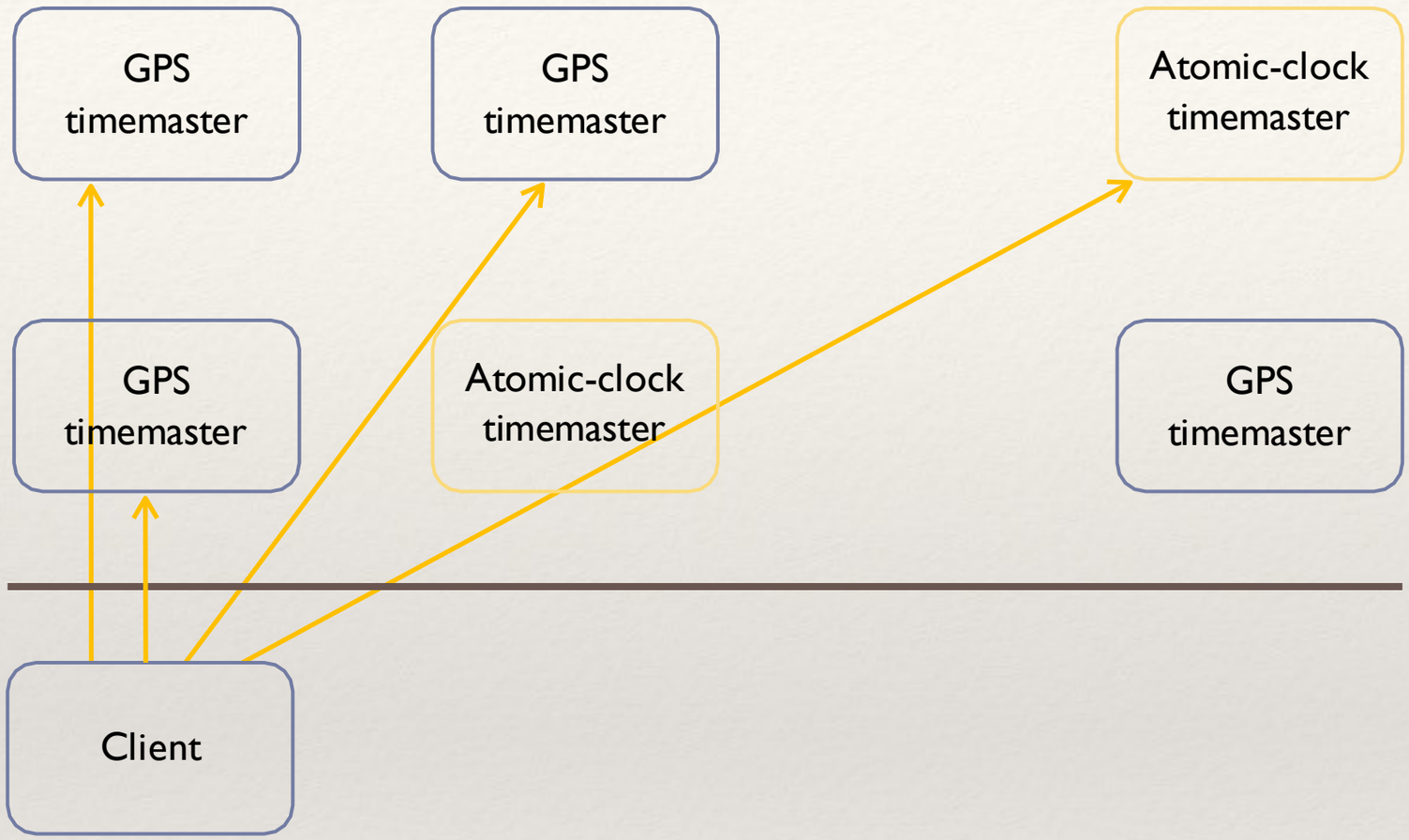
True Time Architecture



now = reference now + local-clock offset

ε = reference ε + worst-case local-clock drift

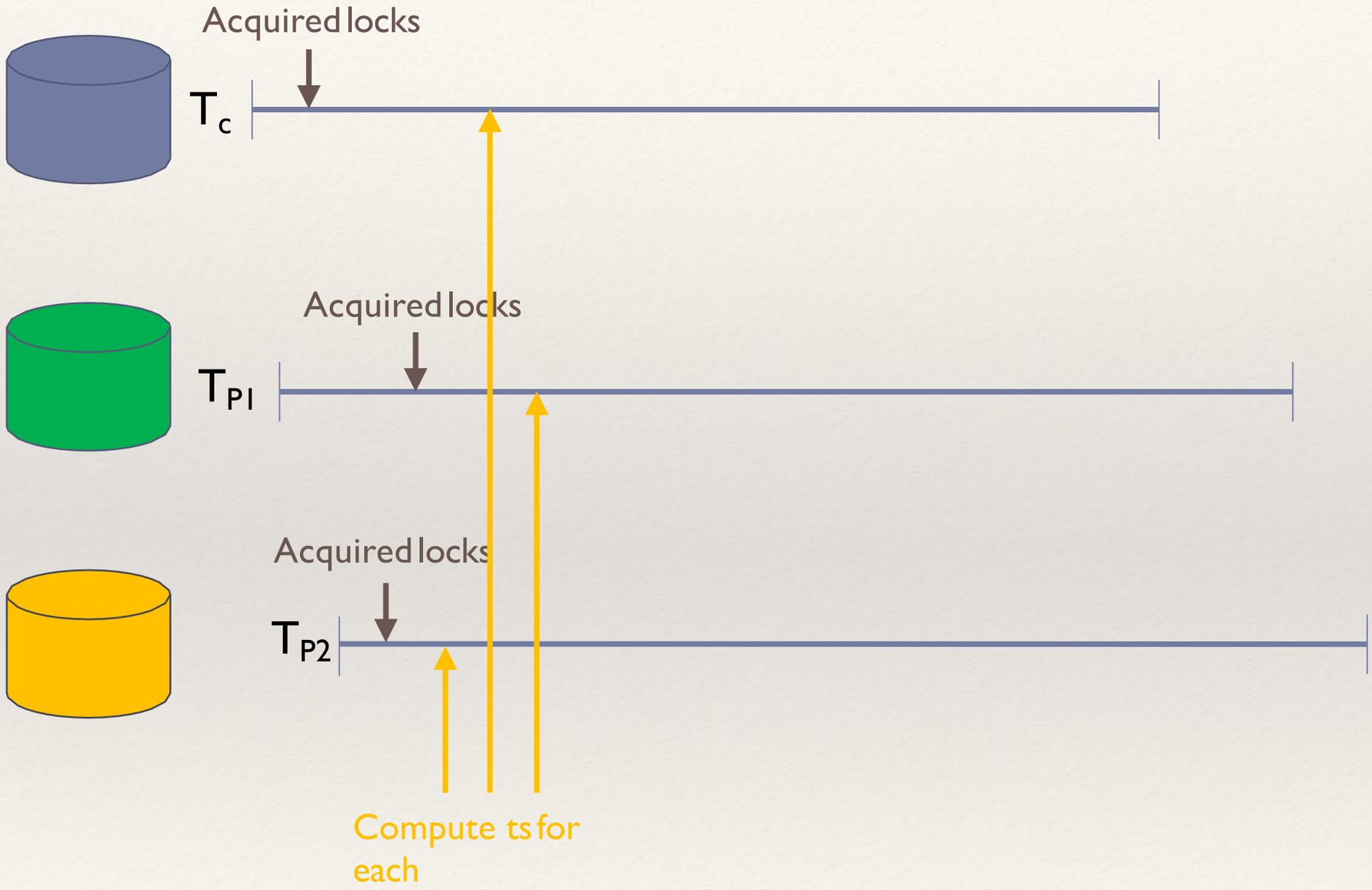
True Time Architecture



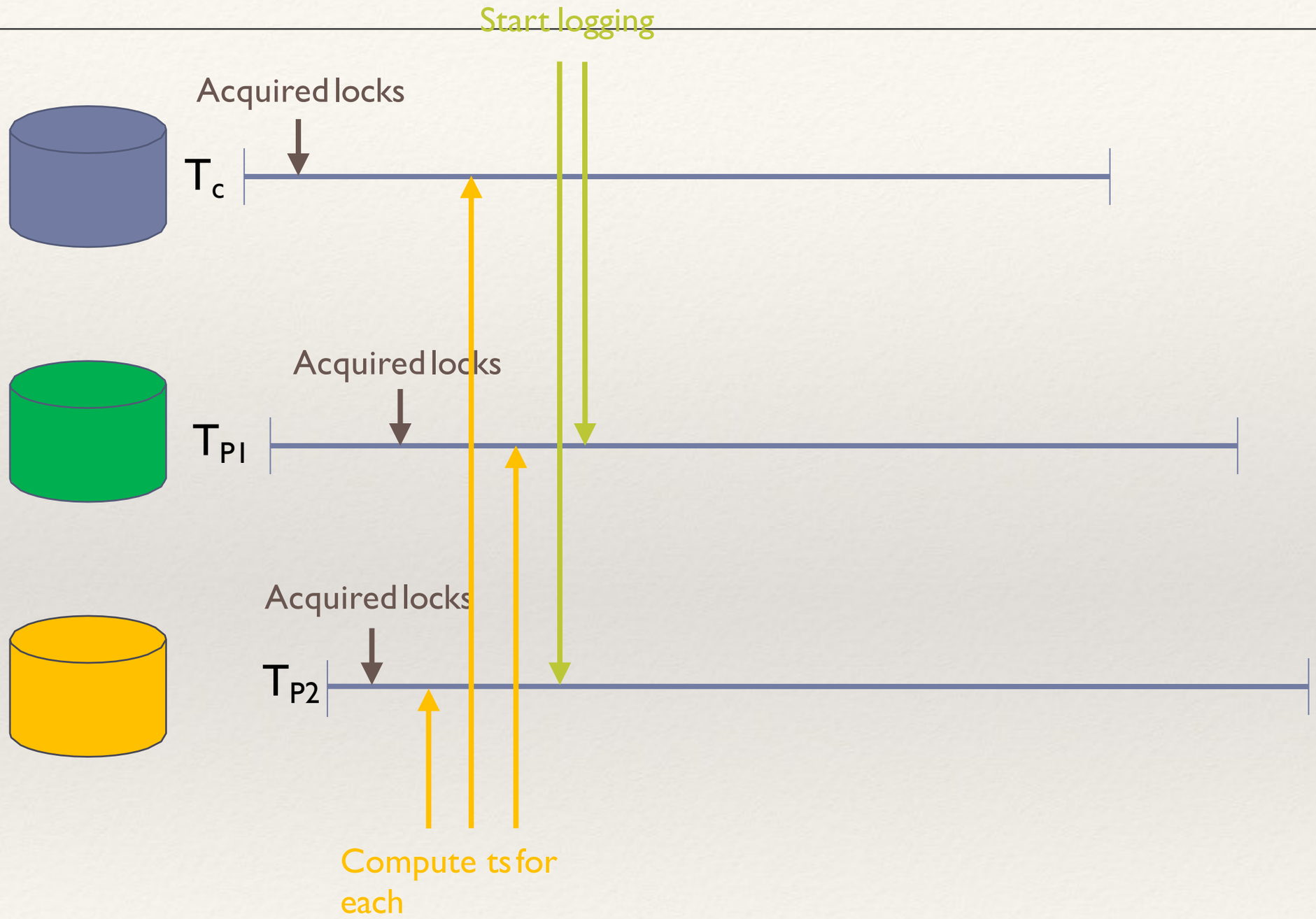
now = reference now + local-clock offset

ε = reference ε + worst-case local-clock drift

Transaction example

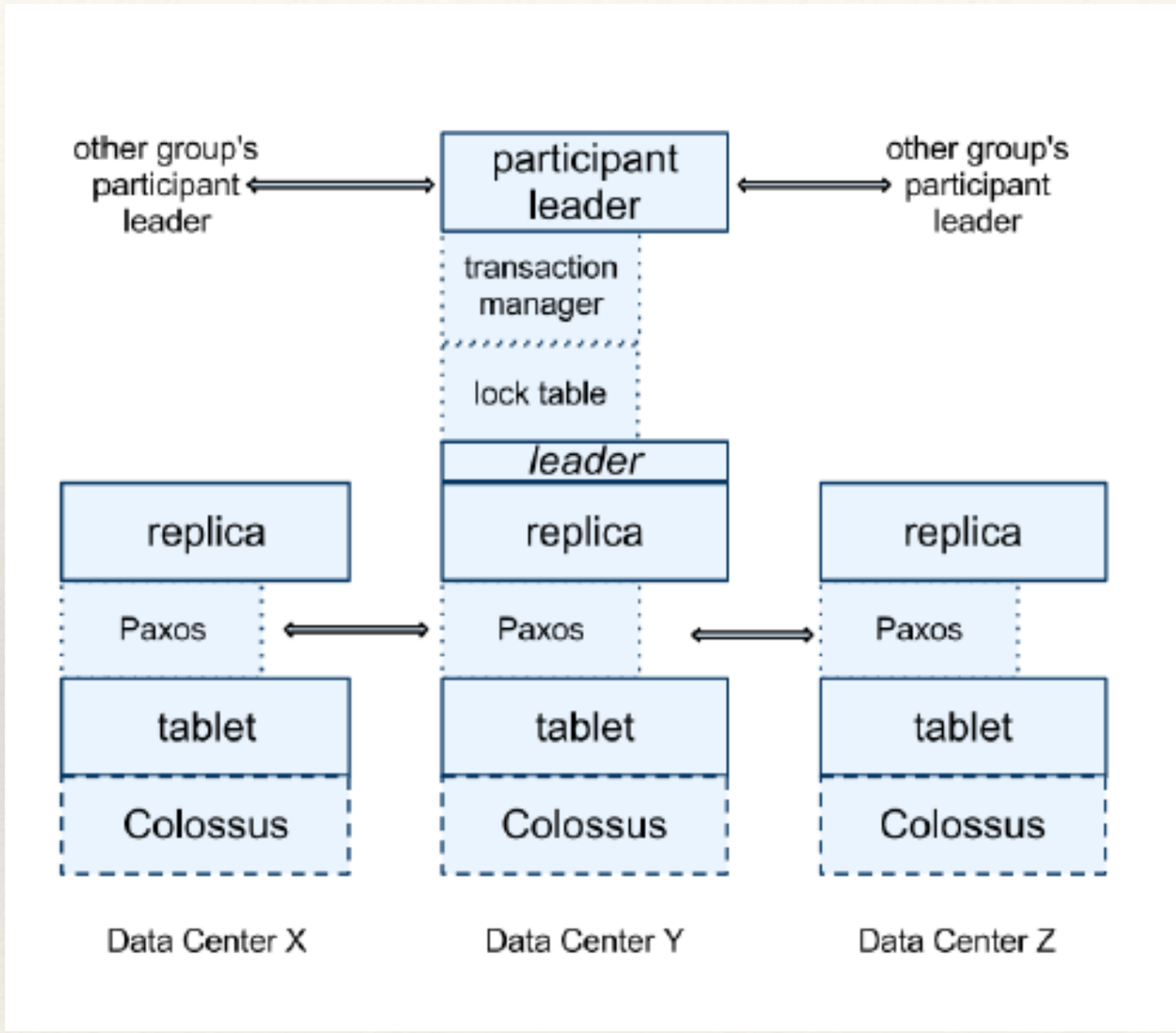


Transaction example



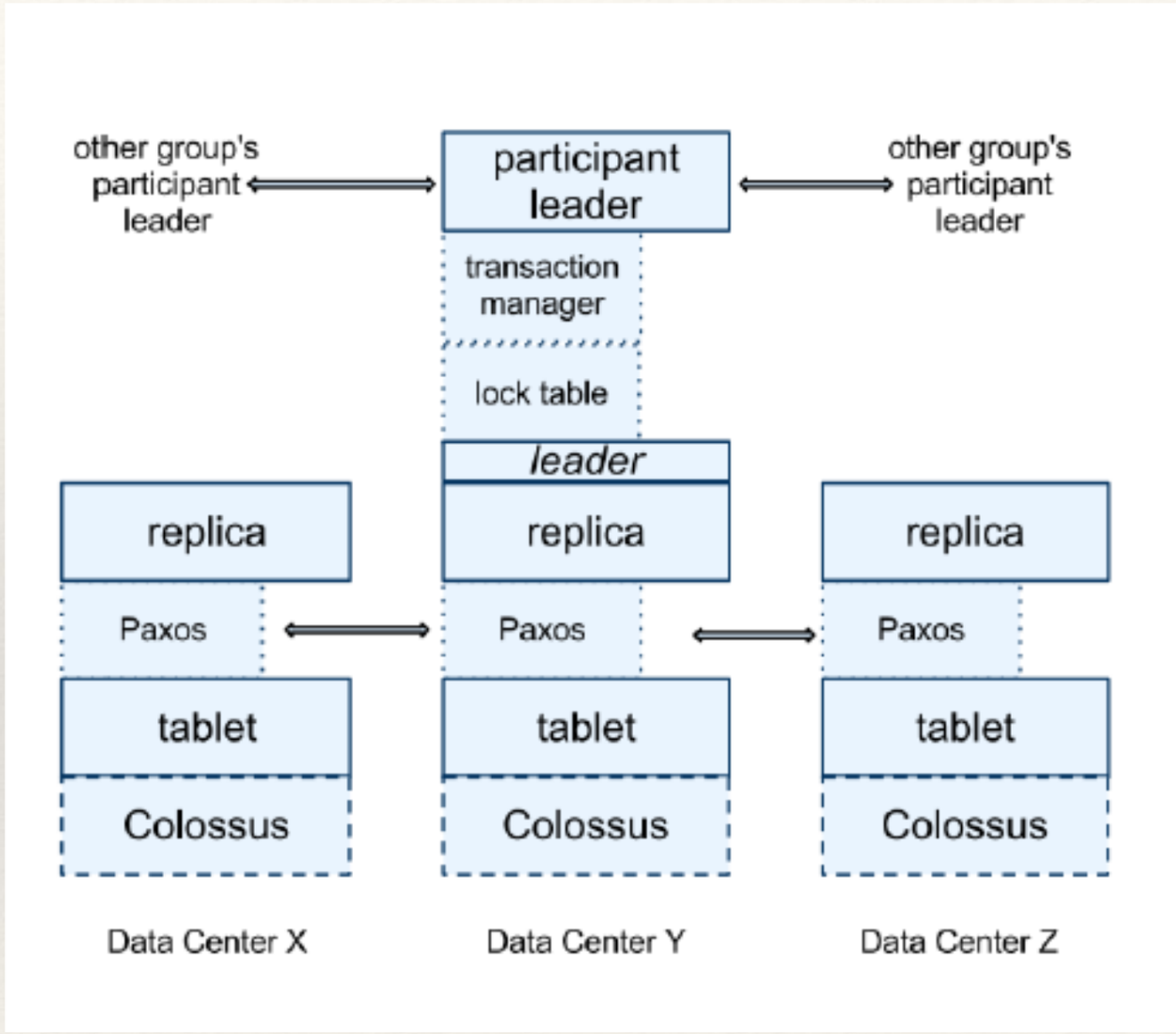
Spanserver software stack

- Tablet implements mappings:
  (key, timestamp) -> string

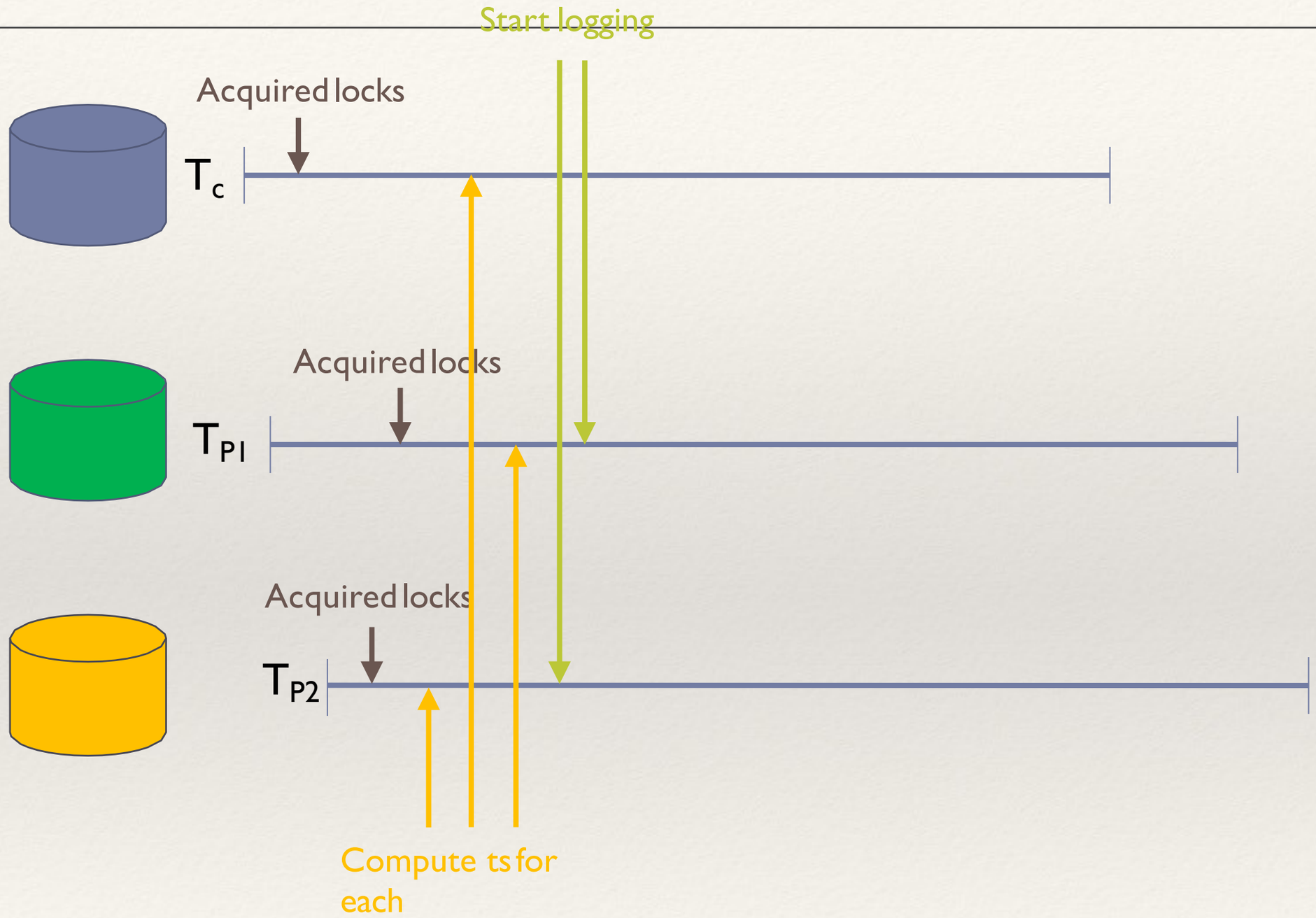


Spanserver software stack

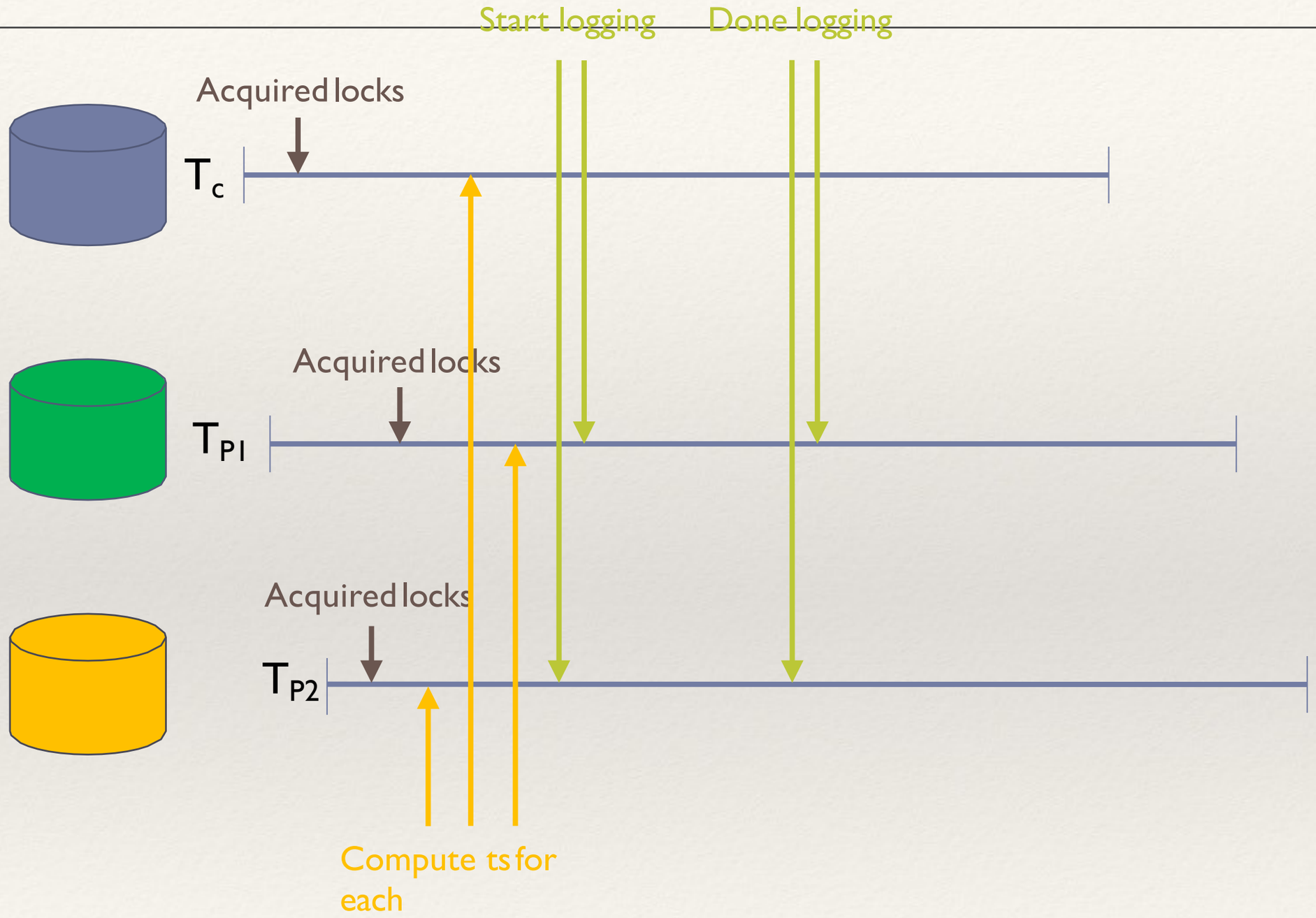
- Tablet implements mappings: (key, timestamp) -> string
- The Paxos state machine for replication support
- Writes initiate the Paxos protocol at the leader
- Focuses on long-lived transactions



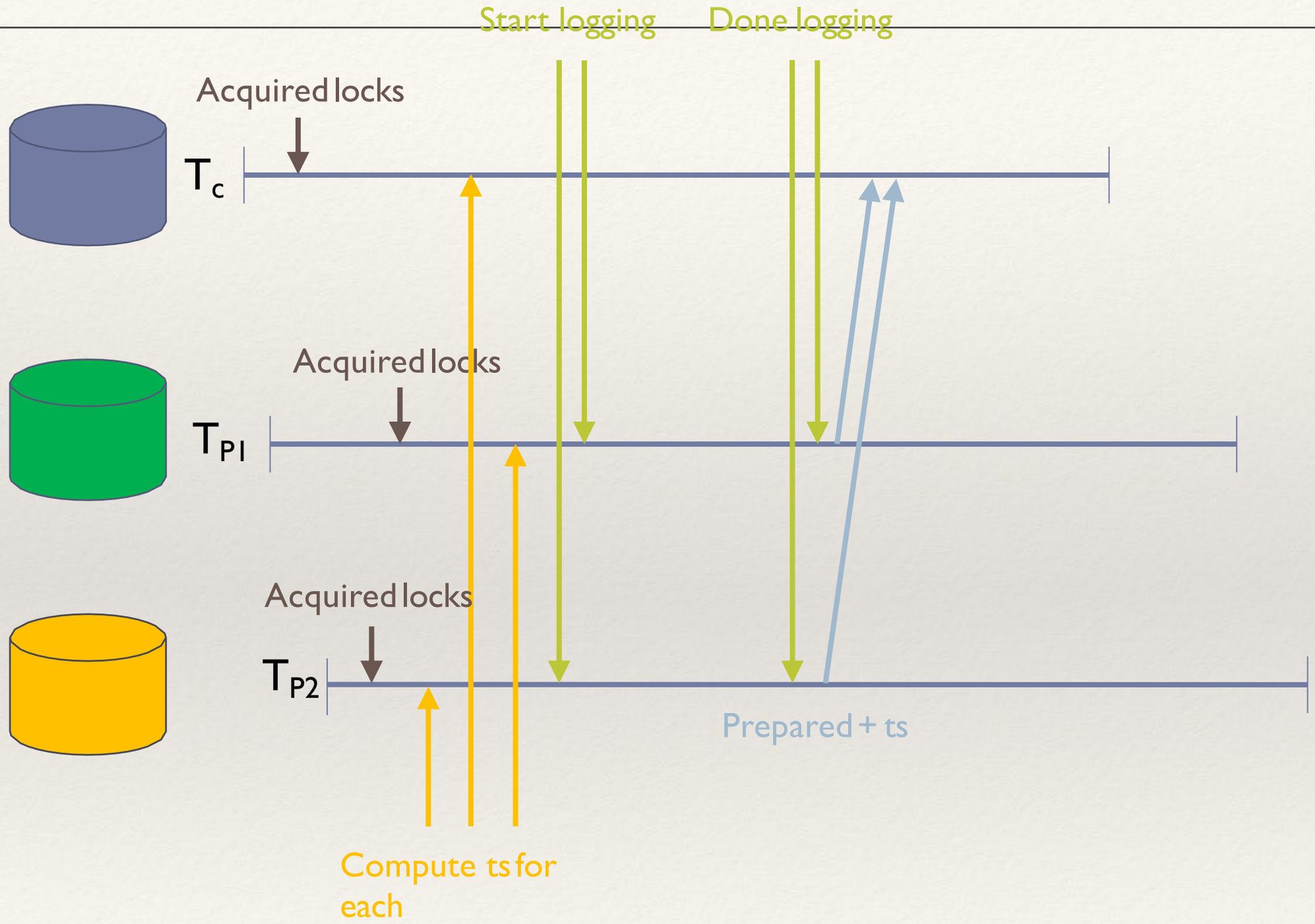
Transaction example



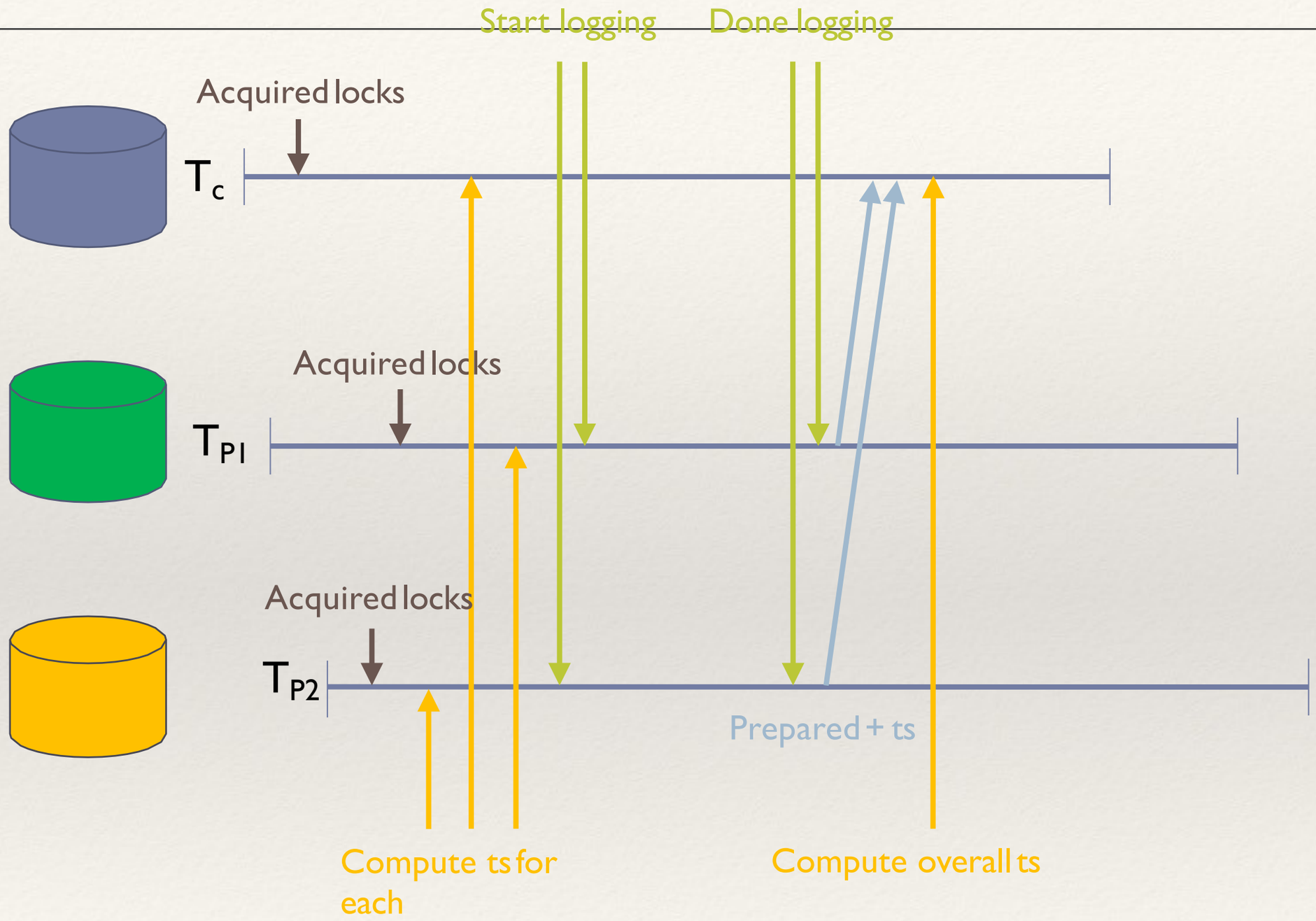
Transaction example



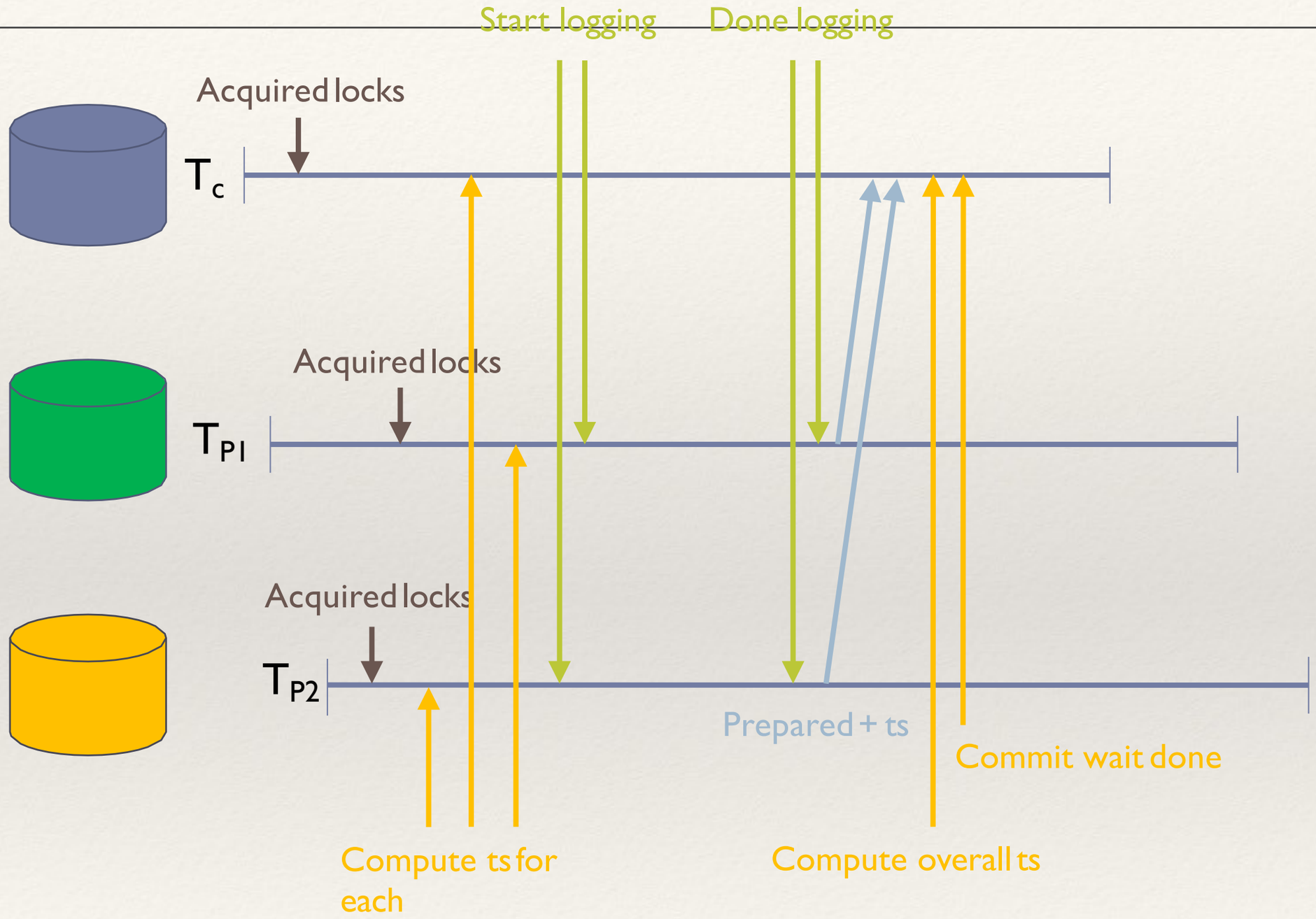
Transaction example



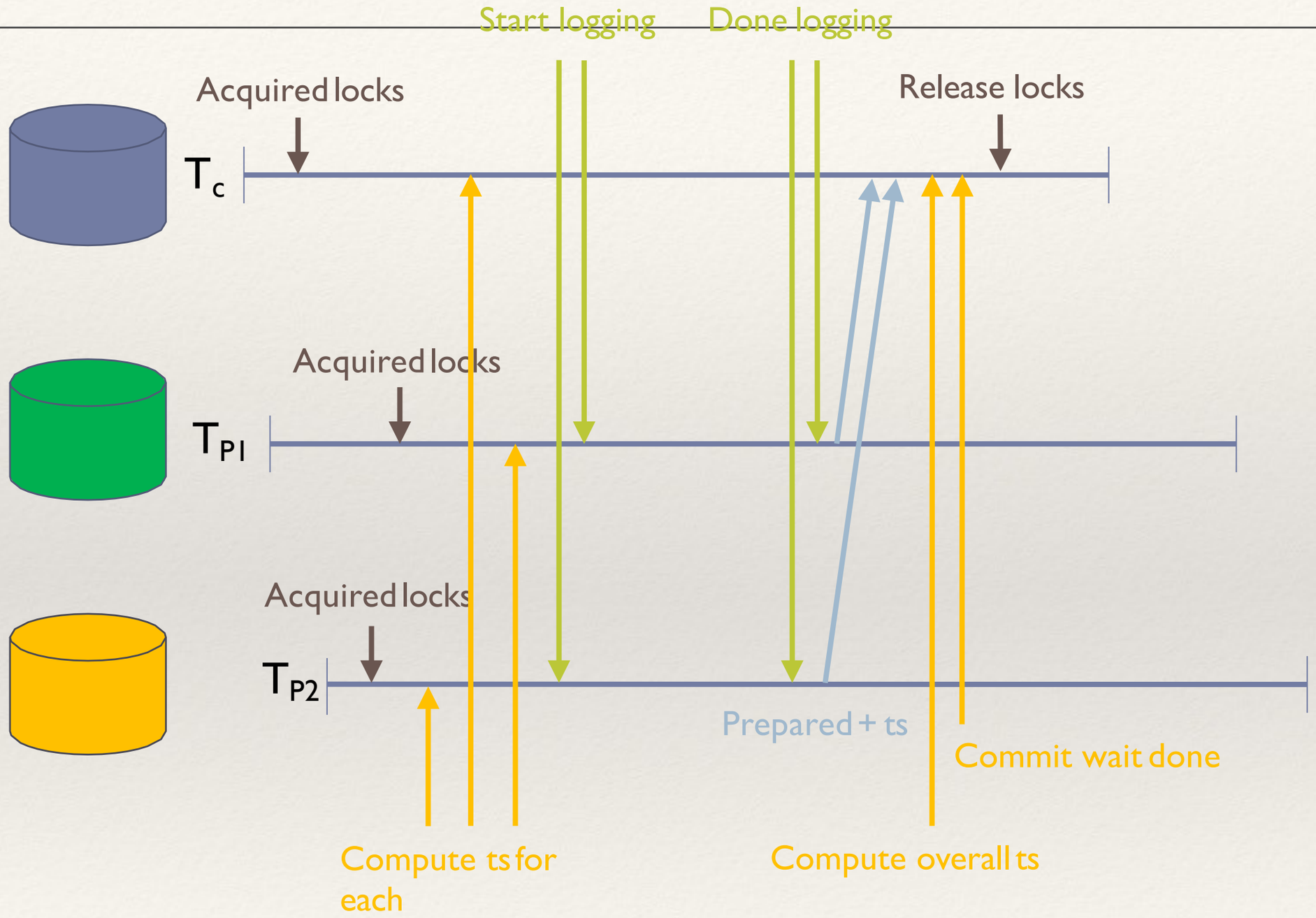
Transaction example



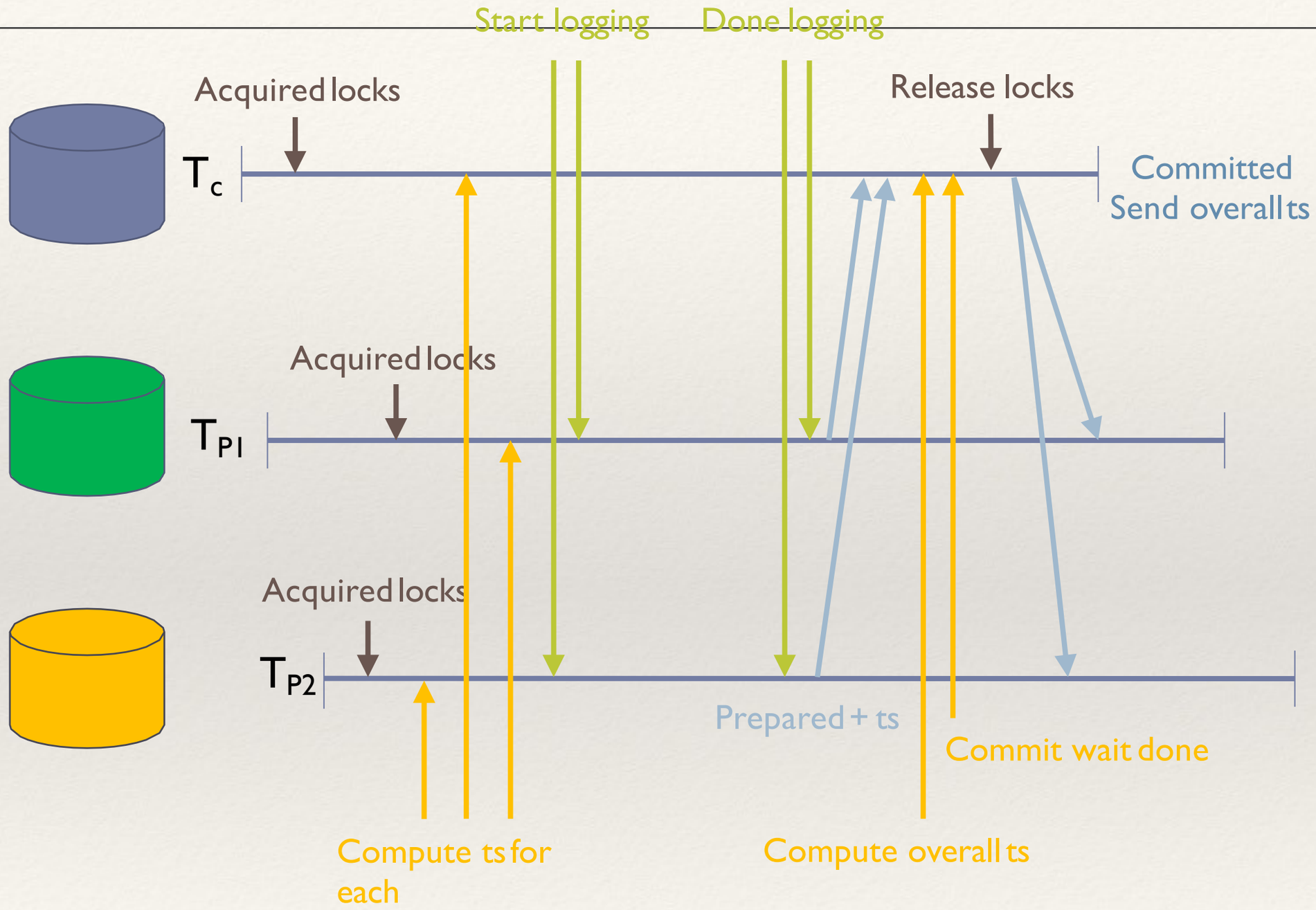
Transaction example



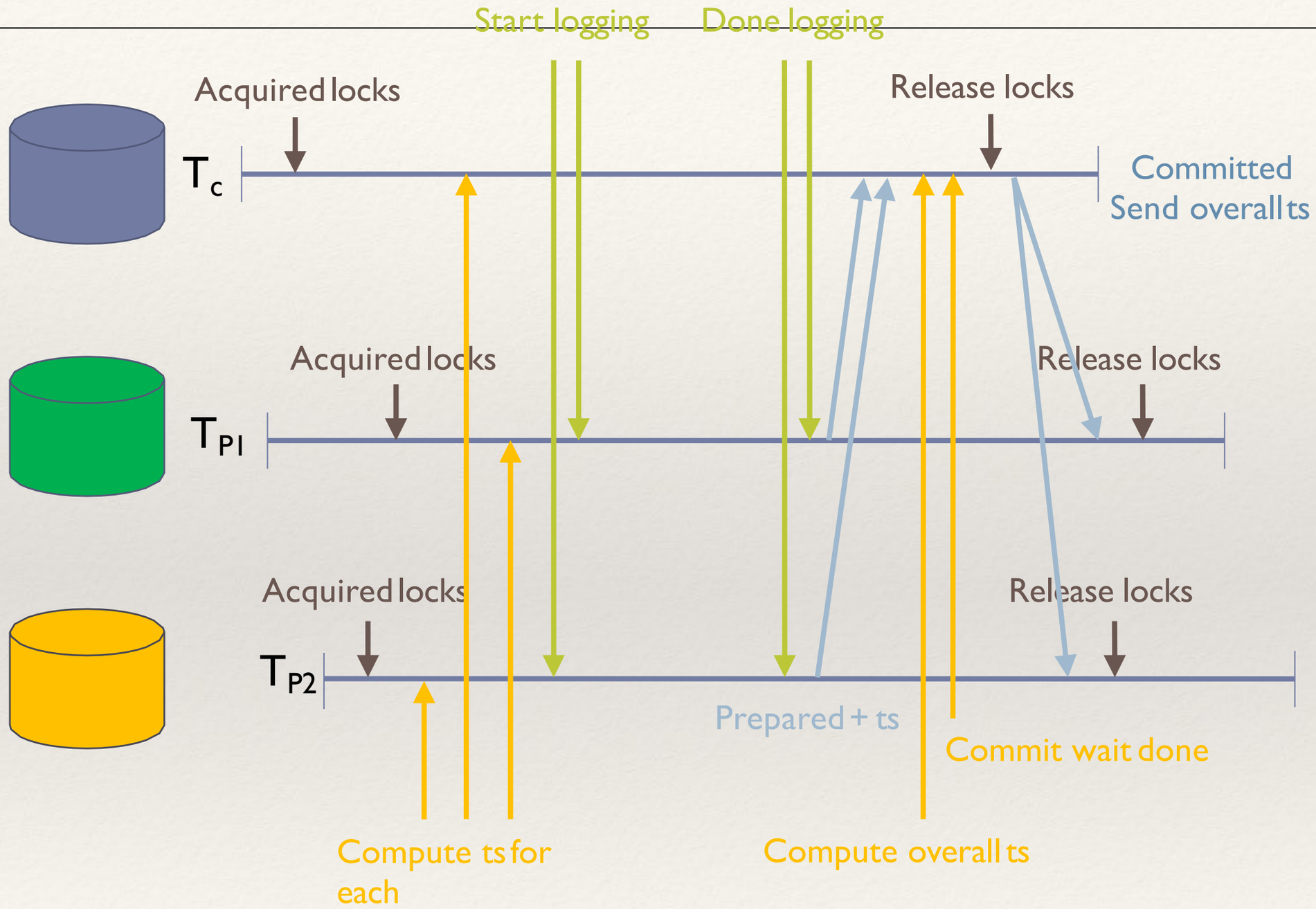
Transaction example



Transaction example



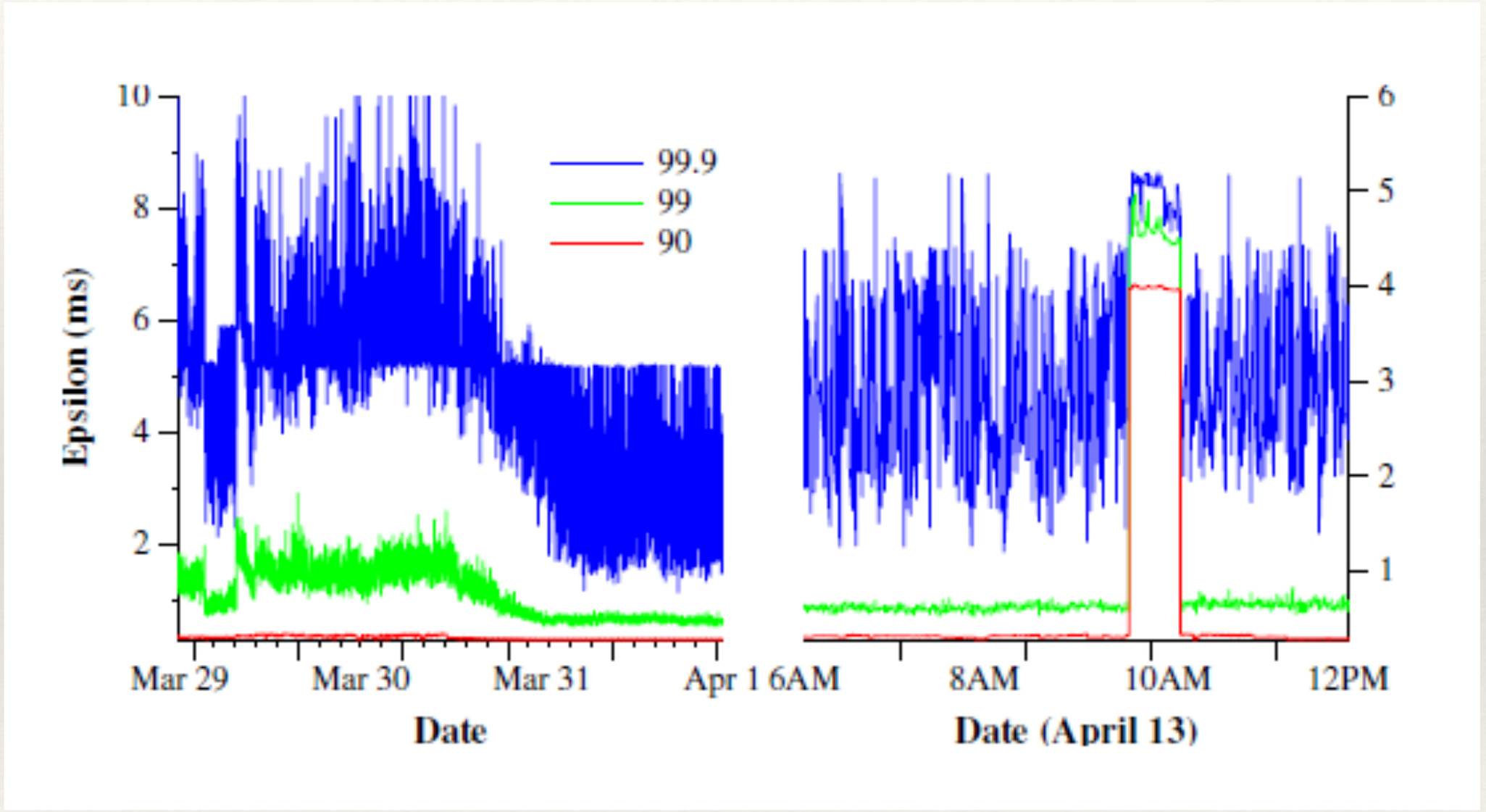
Transaction example



Additional uncovered bits

- Supports atomic schema changes
- Non-blocking snapshot reads in the past
- How to read at the present time
- Paxos protocol restriction
  - Does not support in-Paxos configuration changes

Evaluation: TrueTime uncertainty



Distribution of TrueTime ε values, sampled right after time-slave daemon polls the time masters

Evaluation: F1 study case

# fragments	# directories
1	>100M
2-4	341
5-9	5336
10-14	232
15-99	34
100-500	7

operation	latency (ms)		count
	mean	std dev	
all reads	8,7	376,4	21,5B
single-site commit	72,3	112,8	31,2M
multi-site commit	103,0	52,2	32,1M

Evaluation: F1 study case

# fragments	# directories
1	>100M
2-4	341
5-9	5336
10-14	232
15-99	34
100-500	7

Distribution of directory-fragment counts

operation	latency (ms)		count
	mean	std dev	
all reads	8,7	376,4	21,5B
single-site commit	72,3	112,8	31,2M
multi-site commit	103,0	52,2	32,1M

Evaluation: F1 study case

# fragments	# directories
1	>100M
2-4	341
5-9	5336
10-14	232
15-99	34
100-500	7

Distribution of directory-fragment counts

Perceived operation latencies (over 24 hour course)

operation	latency (ms)		count
	mean	std dev	
all reads	8,7	376,4	21,5B
single-site commit	72,3	112,8	31,2M
multi-site commit	103,0	52,2	32,1M

Evaluation: Microbenchmarks

replicas	latency (ms)			throughput (Kops/sec)		
	write	read-only transactions	snapshot read	write	read-only transactions	snapshot read
1D	9,4±0,6	-	-	4,0±0,3	-	-
1	14,4±1,0	1,4±0,1	1,3±0,1	4,1±0,05	10,9±0,4	13,5±0,1
3	13,9±0,6	1,3±0,1	1,2±0,1	2,2±0,5	13,8±3,2	38,5±0,3
5	14,4±0,4	1,4±0,05	1,3±0,04	2,8±0,3	25,3±5,2	50,0±1,1

Evaluation: Microbenchmarks

replicas	latency (ms)			throughput (Kops/sec)		
	write	read-only transactions	snapshot read	write	read-only transactions	snapshot read
1D	9,4±0,6	-	-	4,0±0,3	-	-
1	14,4±1,0	1,4±0,1	1,3±0,1	4,1±0,05	10,9±0,4	13,5±0,1
3	13,9±0,6	1,3±0,1	1,2±0,1	2,2±0,5	13,8±3,2	38,5±0,3
5	14,4±0,4	1,4±0,05	1,3±0,04	2,8±0,3	25,3±5,2	50,0±1,1

participants	latency (ms)	
	mean	99th percentile
1	17,0±1,4	75,0±34,9
2	24,5±2,5	87,6±35,9
5	31,5±6,2	104,5±52,2
10	30,0±3,7	95,6±25,4
25	35,5±5,6	100,4±42,7
50	42,7±4,1	93,7±22,9
100	71,4±7,6	131,2±17,6
200	150,5±11,0	320,3±35,1

Conclusion

- The first service to provide global externally consistent multi-version database

- Relies on novel time API (TrueTime)

- Improvements introduced over previous services