FAUNA

FaunaDB

Global Consistency at High Throughput

Matt Freels, CTO and Cofounder
HPTS 2017

FaunaDB: Summary

- OLTP database (soft-realtime or interactive workloads)
- Written in Scala, runs on JVM
- NoSQL, hybrid data model
- eDSL-based query language (closer to general purpose PL)
- Multi-key ACID transactions
- Multi-region replication
- Cloud-native (network, hardware assumptions)

Functional Transaction Language

```
Create(Class("users"),
  Obj("data" -> Obj("name" -> "Matt"))
)
```

```
Let {
  val set = Match(Index("users_by_name"), "Matt")
  Paginate(set)
}
```

Data Model

- Instances organized by class
- Indexes are first-class
  - Map instances to partitioned, sorted sets of tuples
- Immutable, temporal by default

Underlying Algorithms

Calvin

Used for distributed multi-key transaction resolution

- Strict serializability
- Well-suited for multi-region environment
- Local region snapshot reads
- No special hardware requirements
- Modular architecture

Underlying Algorithms

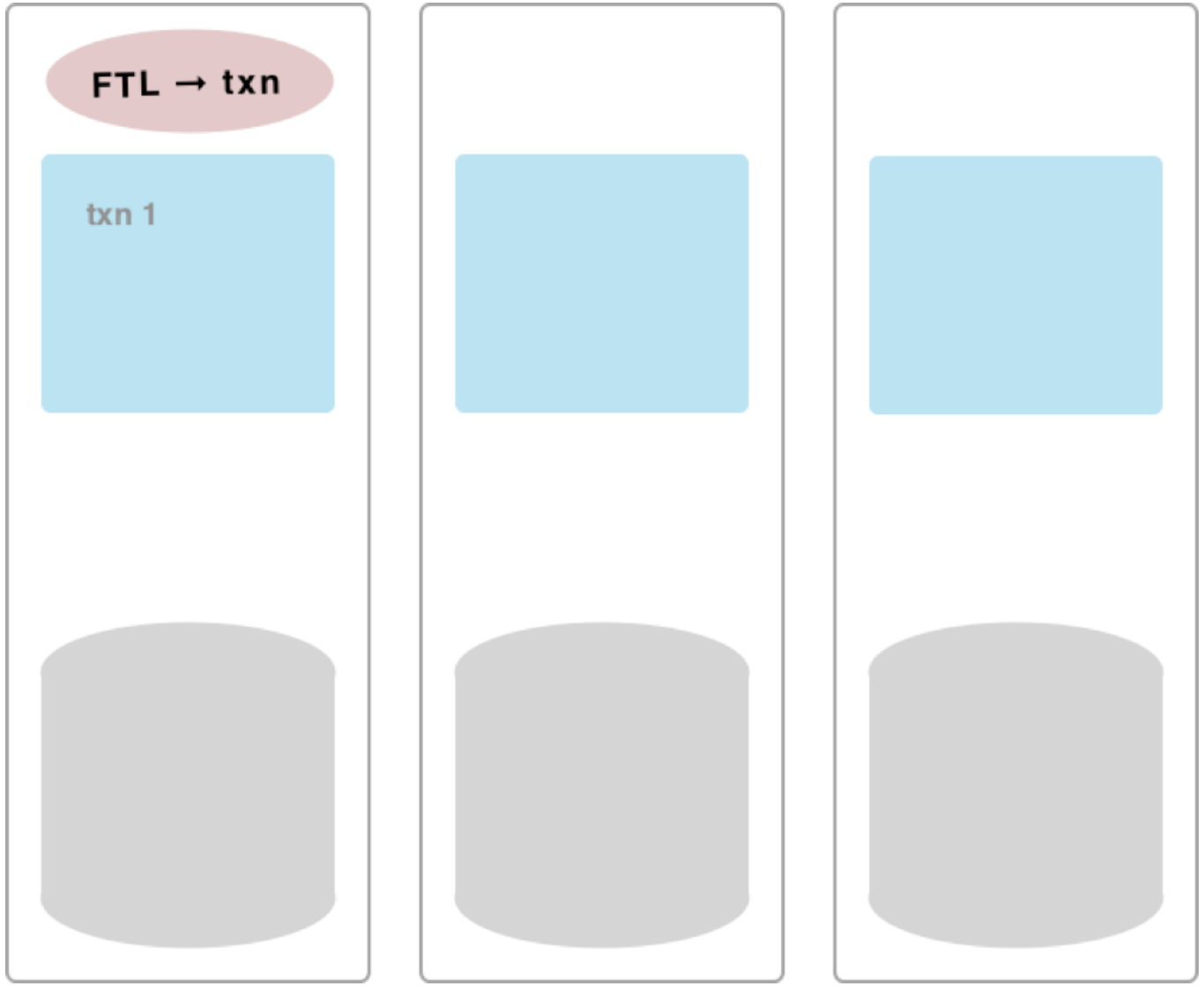
Calvin

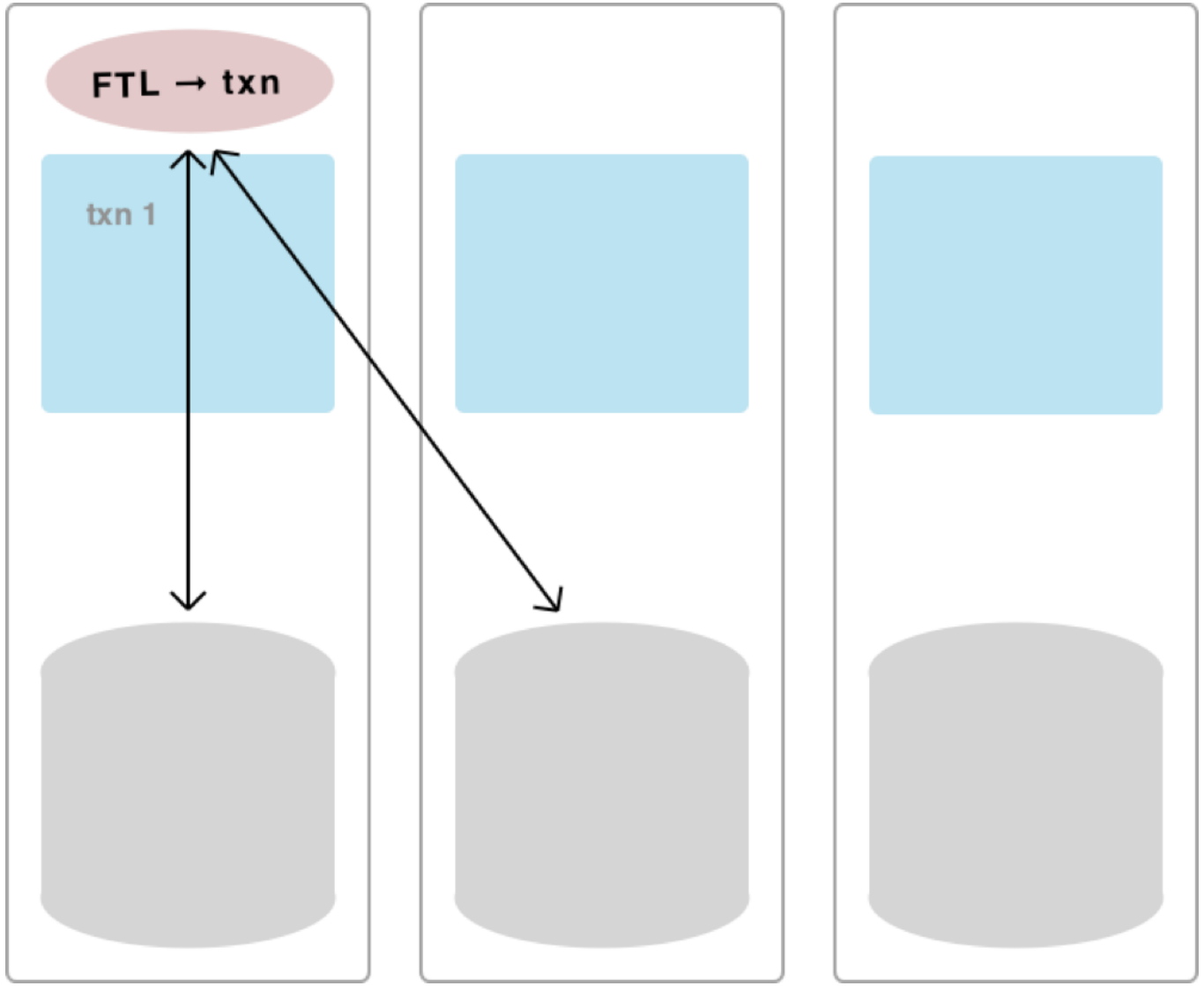
- Transactions == pure functions
  - predetermined read-write set
- Partitioned log determines transaction order
  - Order reflected in transaction timestamps
- Data partitions apply transactions independently by region
  - Temporal storage allows for looser coupling
  - Last applied transaction timestamp determines safe snapshot reads

Life of a Transaction

```
Let {
  val matt = Get(Match(Index("users_by_name"), "Matt"))

  Update(
    Select("ref", matt),
    Obj("data" -> Obj("status" -> "At HPTS")
  )
}
```





Life of a Transaction

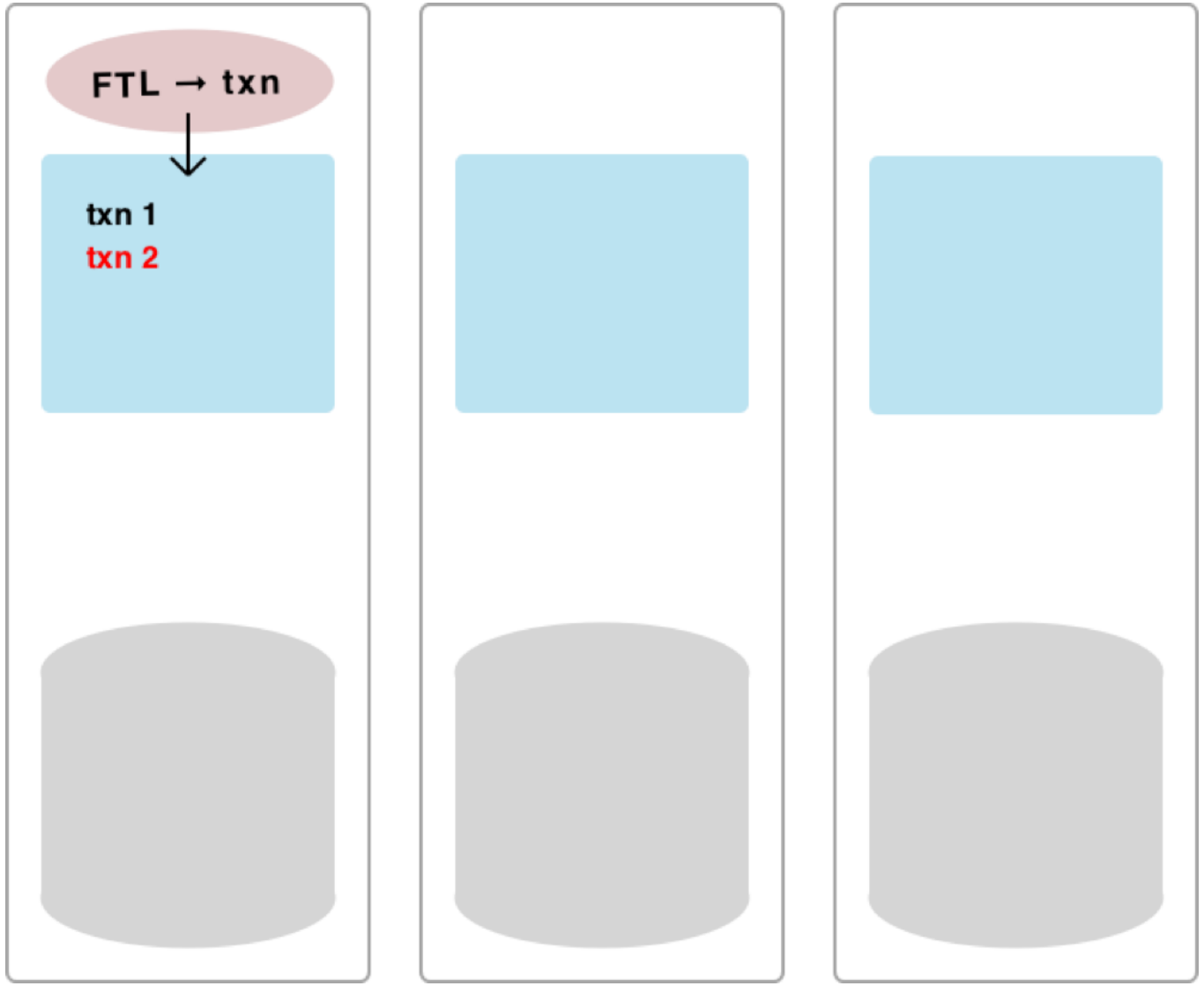
Snapshot time: `t12`

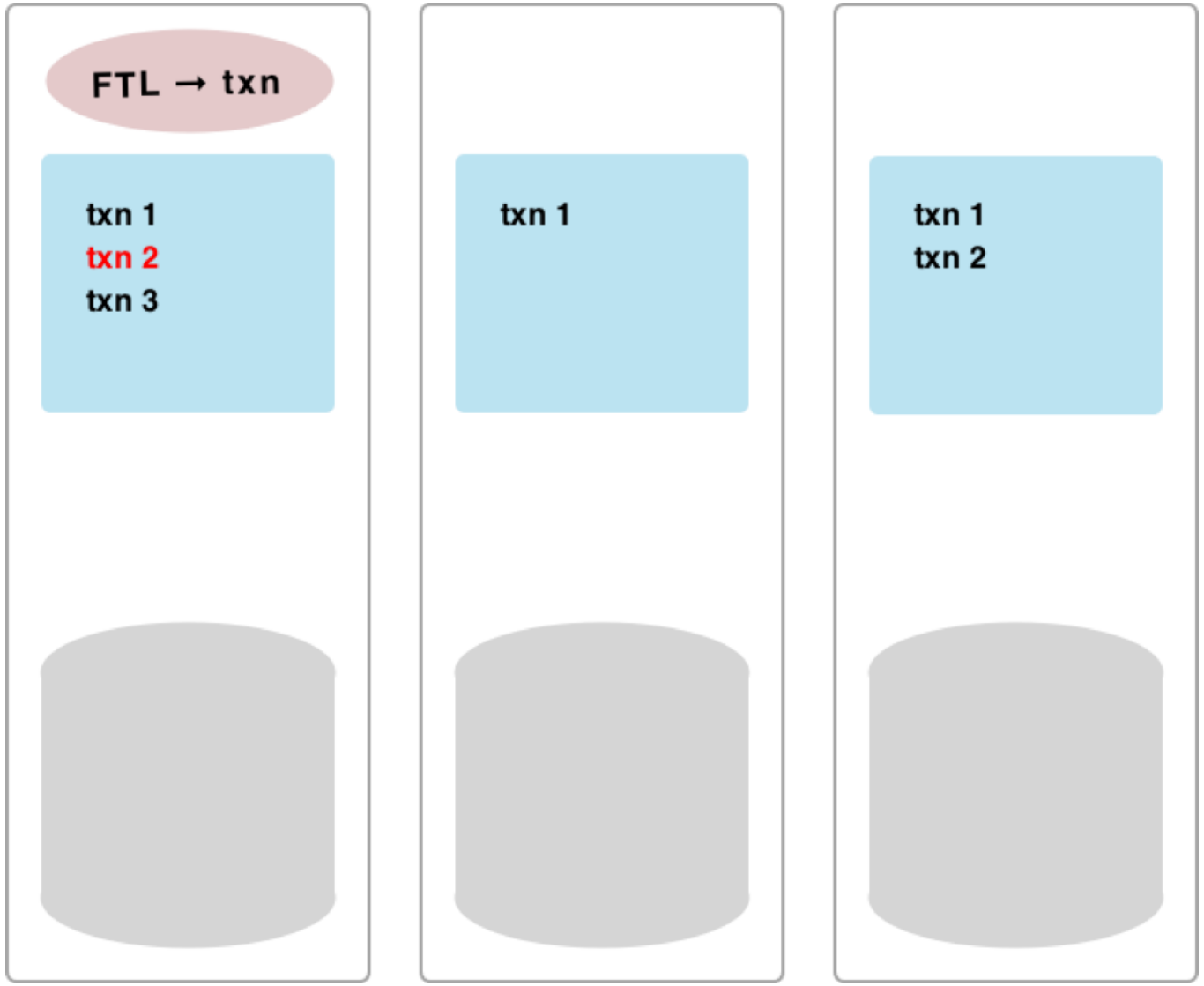
Reads:

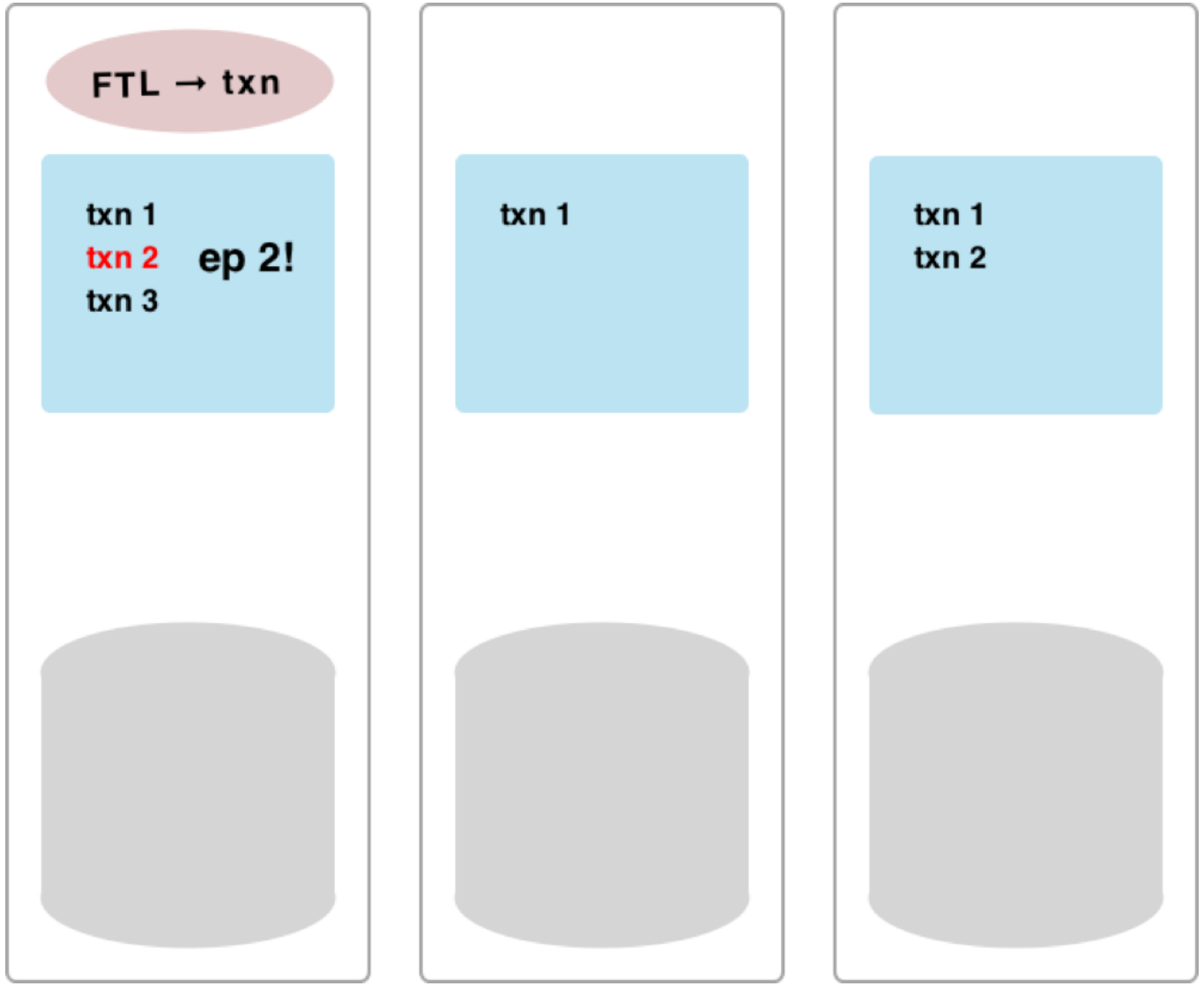
```
[Match(Index("users_by_name"), "Matt")), t6]
["users/123", t6]
```

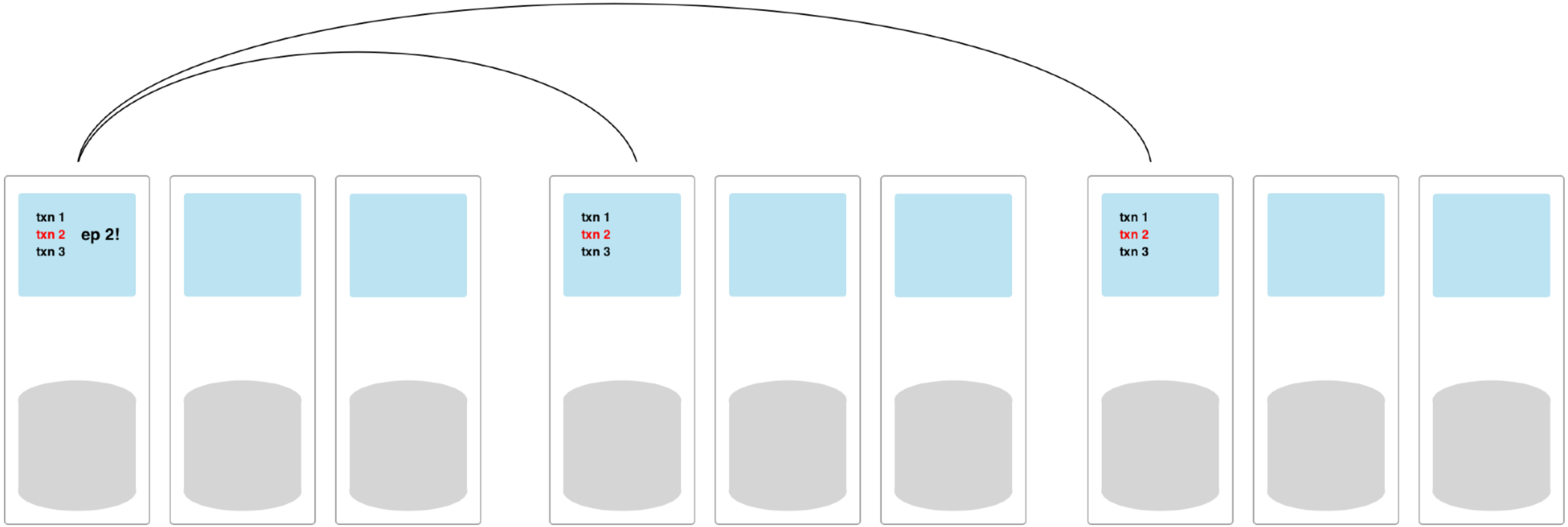
Writes:

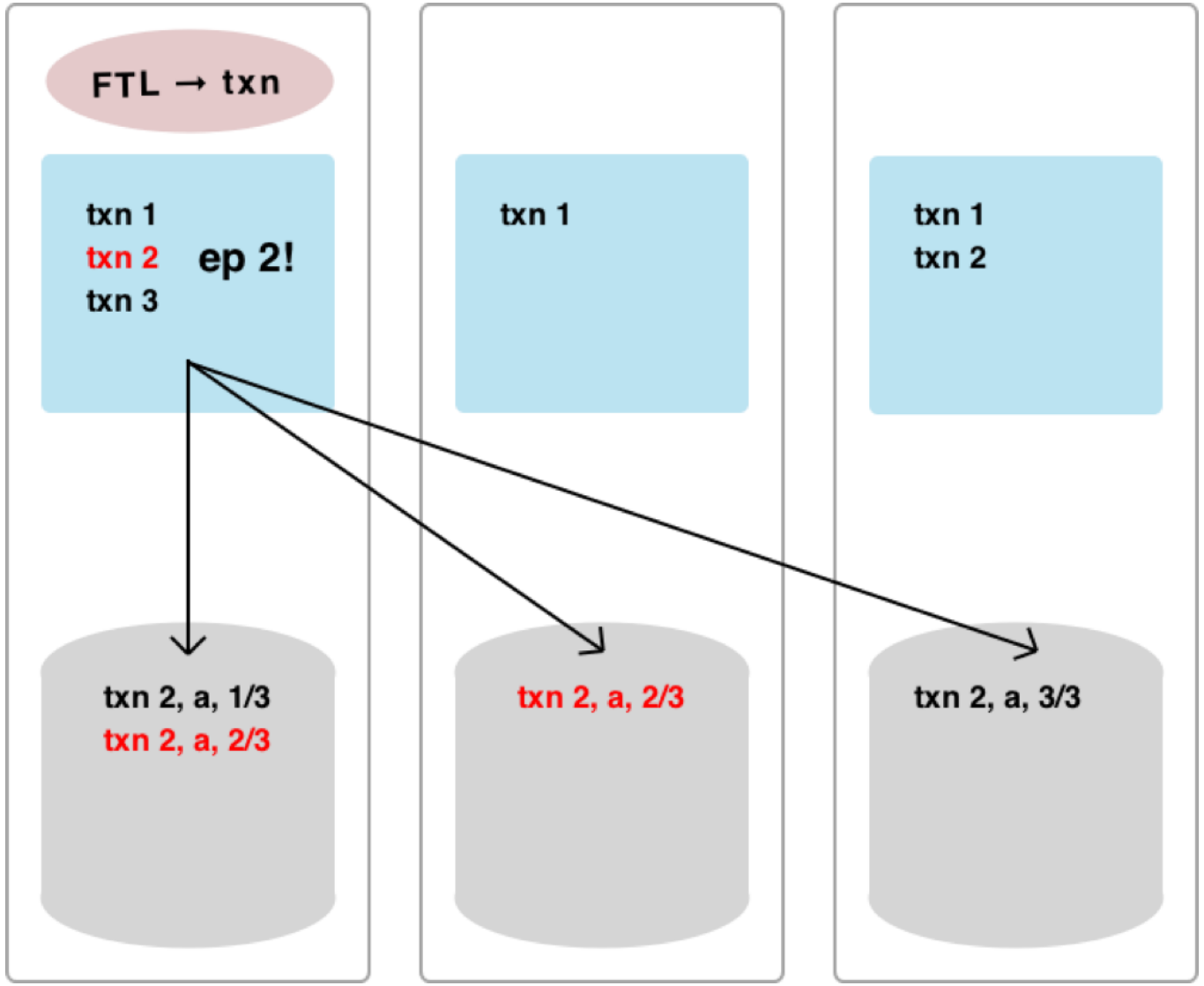
```
["users/123", { "data": { "name": "Matt", "status": "At HPTS" } }]
```

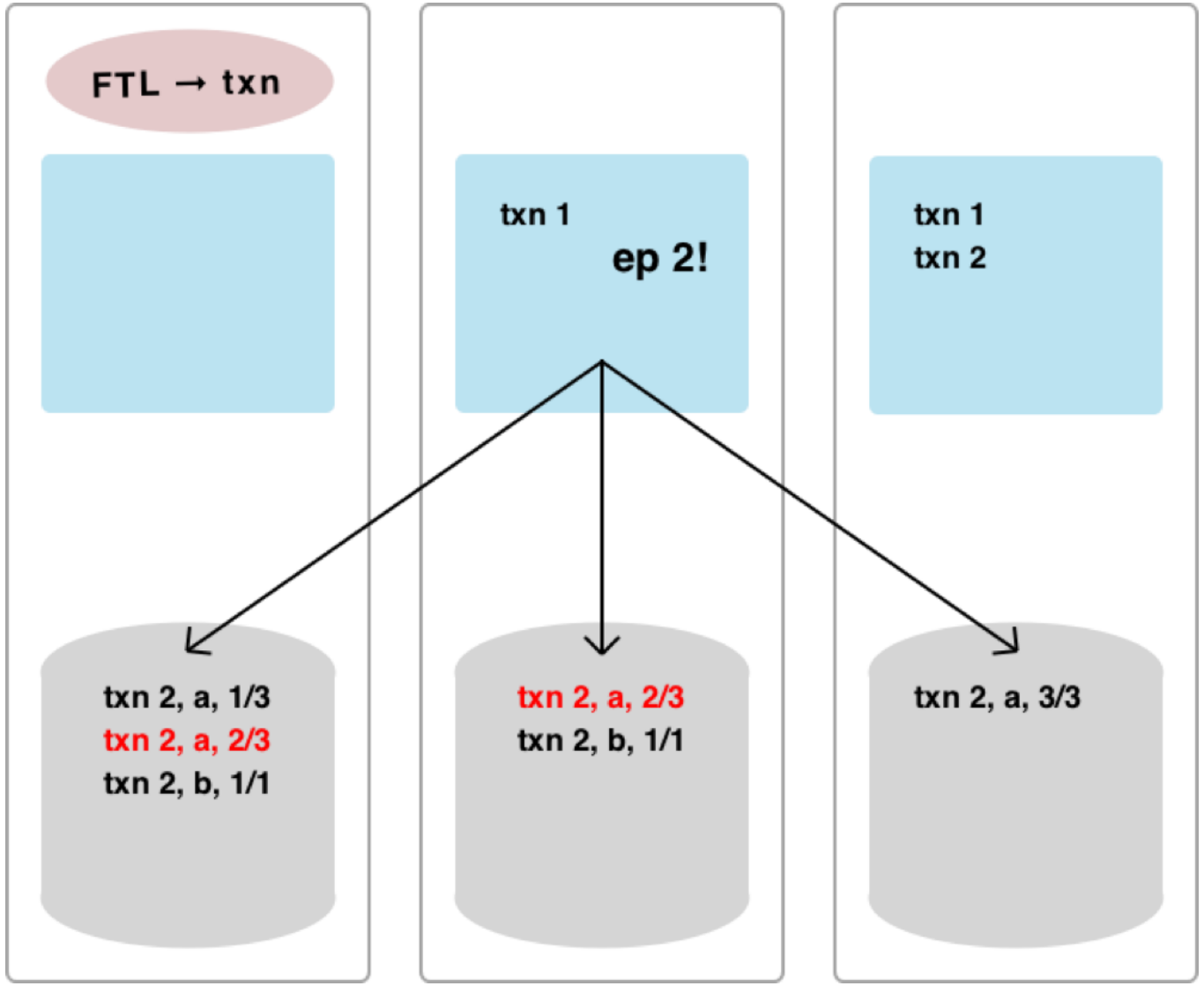


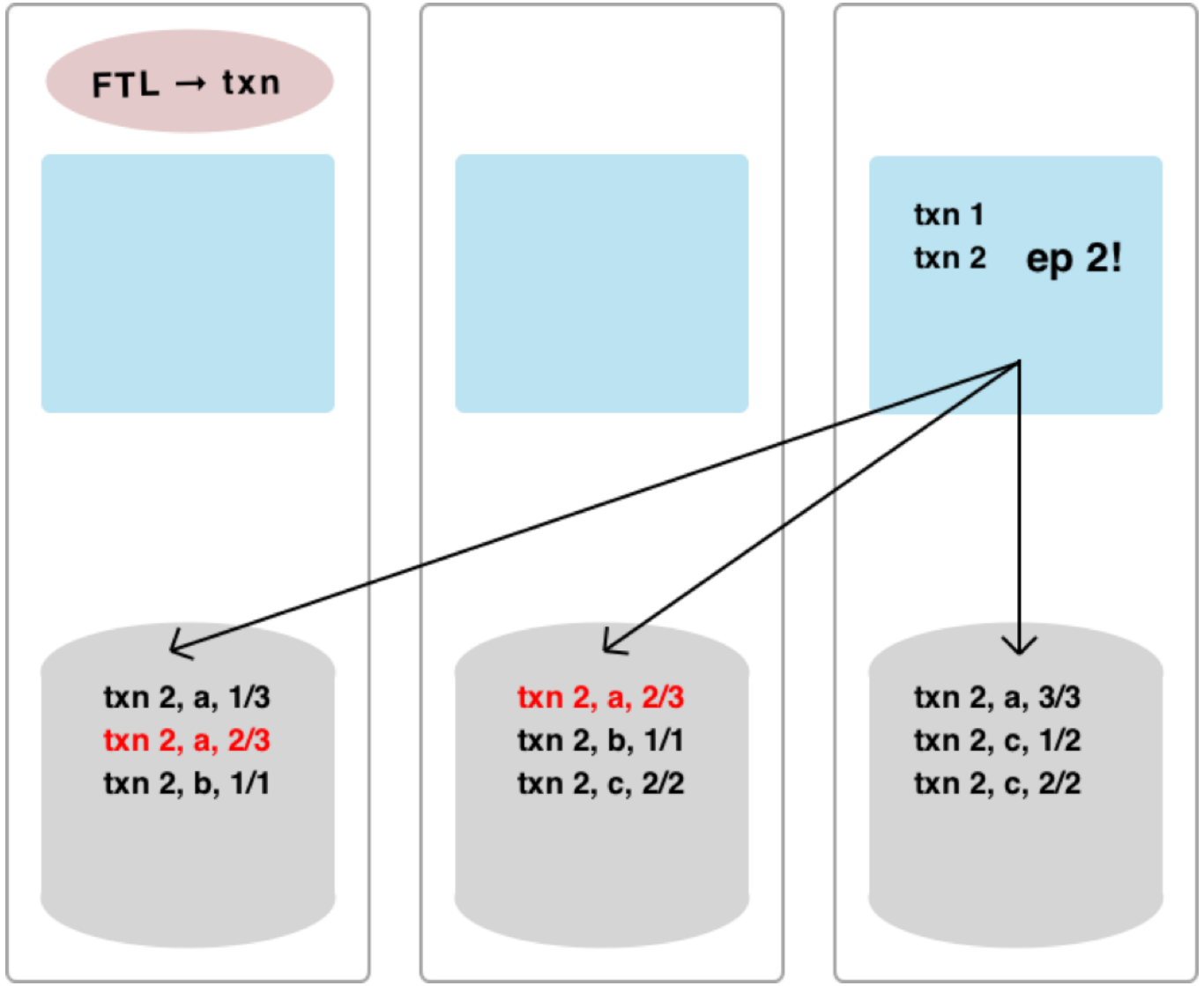


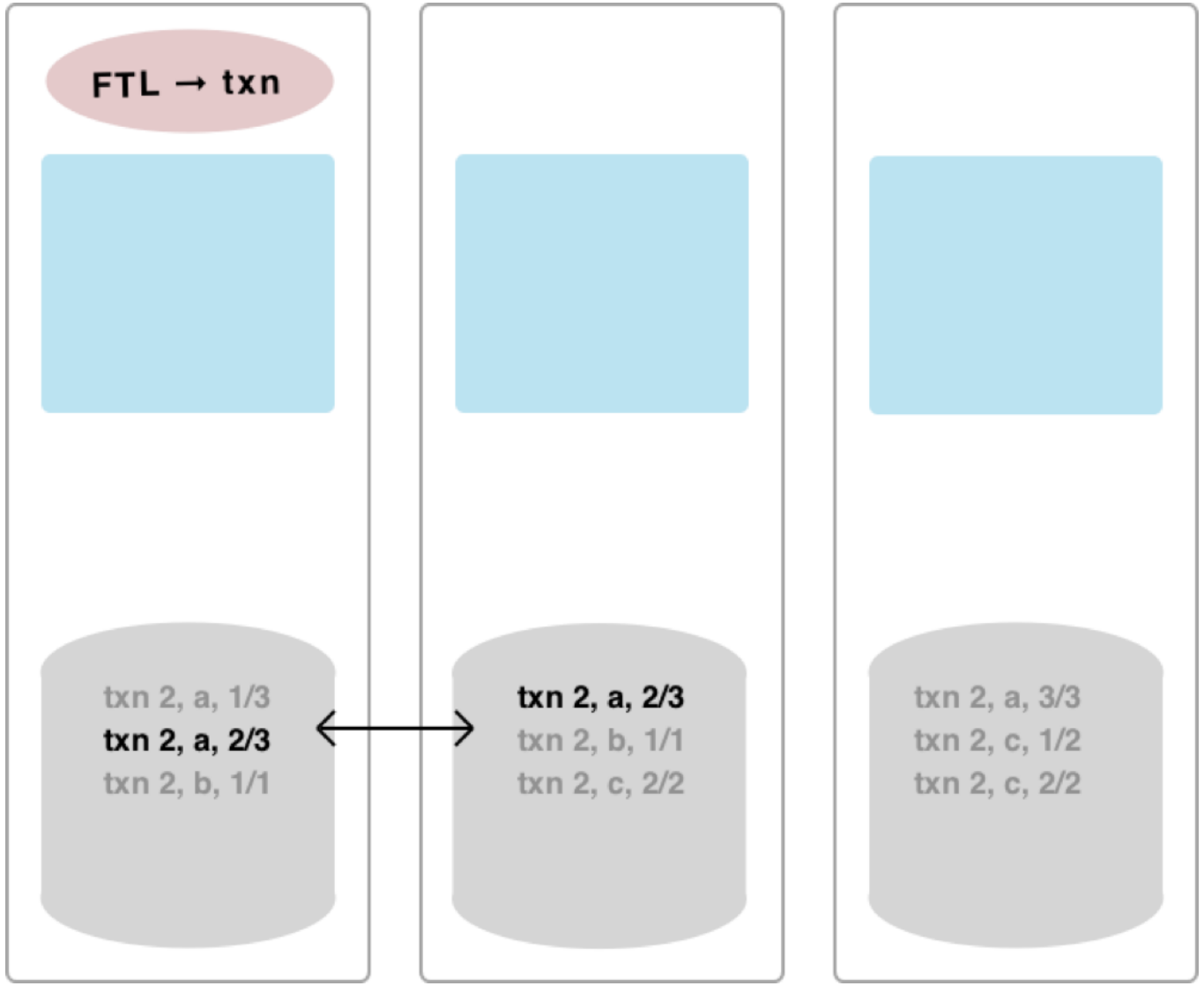












Life of a Transaction

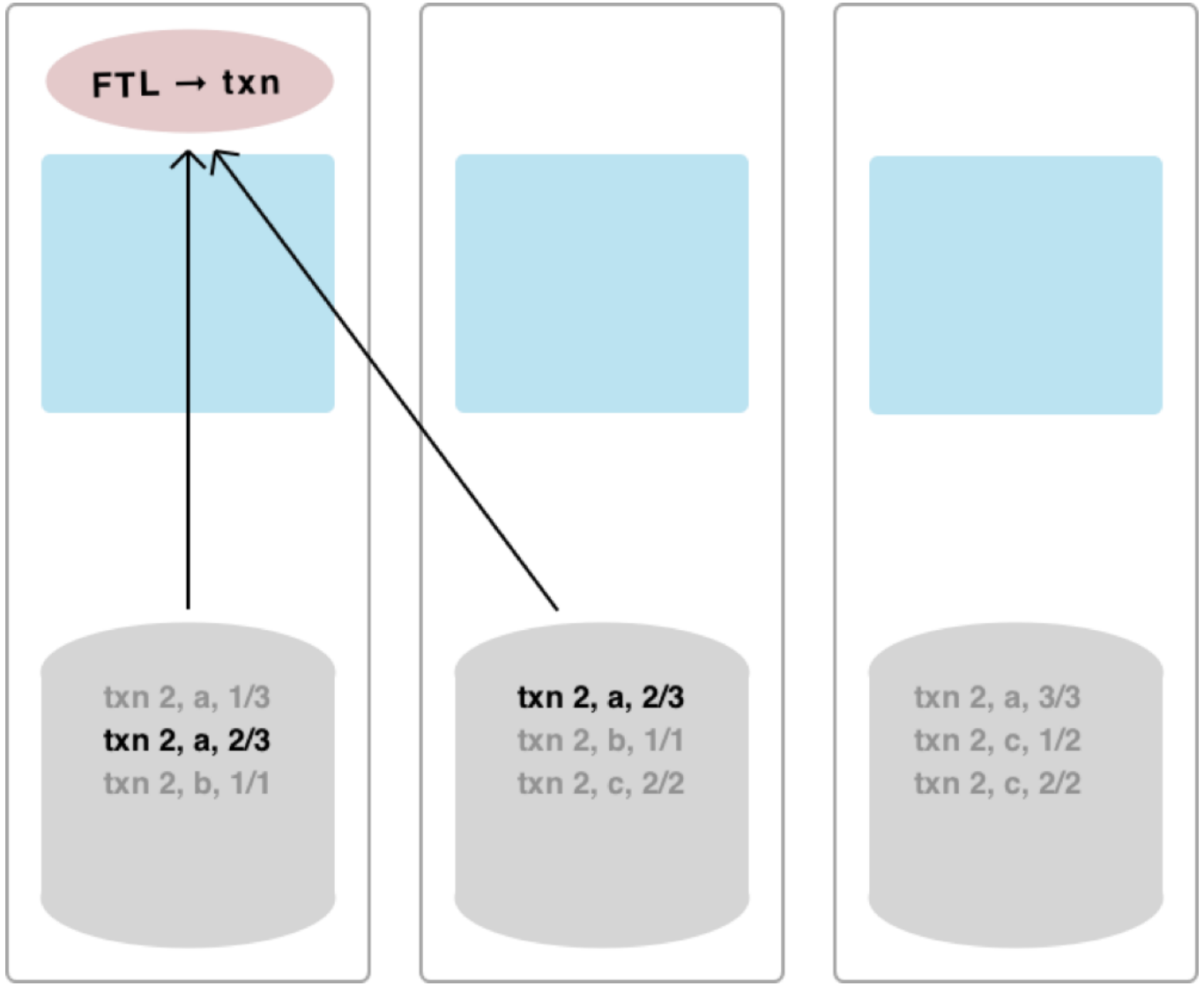
Transaction time: `(2, a, 2/3) → (2, 2/6) → t25`

Reads:

```
[Match(Index("users_by_name"), "Matt")), t6]
["users/123", t6]
```

Writes:

```
["users/123", { "data": { "name": "Matt", "status": "At HPTS" } }]
```



Read-only Transactions

- Relaxed consistency by default (Still serializable)
- Transaction timestamp acts as a causal token
- Snapshot read clock informed by transaction apply latency

Underlying Algorithms

RAFT

- Core internal replication primitive
- Used to replicate:
  - Shared cluster state
  - Calvin transaction log

Underlying Algorithms

RAFT

Improvements over reference implementation:

- Pipelined appends
- Better election protocol resiliency
  - Minority cannot trigger an election
  - Faster elections (not timeout-dependent)
- Reduced WAN communication overhead
  - Allow reads from any replica
  - Broadcasted acceptance & peer-based commit

Performance Characteristics

- 500-5000 write TPS/node, depending on fanout
- 100-200ms write latencies in global cluster
- Degree of contention between transactions affects throughput, higher rate of transaction failure/retry
- Theoretical maximum number of log partitions (ergo throughput) per database

Future Work

- Selectable read consistency (strictly serializable reads)
- Selectable write consistency (tentative response)
- Richer transaction semantics (fewer aborts under contention)
- Transaction reordering
- Generalized at-most-once transactions

Thank You!

Contact:
matt@fauna.com