Omid: A Transactional Framework for HBase

Francisco Perez-Sorrosal

Ohad Shacham

Hadoop Summit SJ

June 29th, 2016

Outline

- Background
- Basic Concepts
- Use cases
- Architecture
- Transaction Management
- High Availability
- Performance
- Summary

Background

- New Big data apps → new requirements:
  - Low-latency
  - Incremental data processing
  - e.g. Percolator

- Multiple clients updating same data concurrently
  - **Problem**: Conflicts/Inconsistencies may arise
  - **Solution**: **Transactional Access to Data**

Background

- Transaction → Abstract UoW to manage data with certain guarantees
  - ACID
  - Relational databases
- Big data → NoSQL datastores → Transactions in NoSQL
  - Hard to Scale
    - Data partition
    - Data replication
  - Relaxed Guarantees:
    - e.g. Atomicity, Consistency

Omid is a…

- **Flexible**
- **Reliable**
- **High Performant**
- **Scalable**

…***OLTP framework*** that allows **BigData** apps to execute **ACID transactions** on top of **HBase**

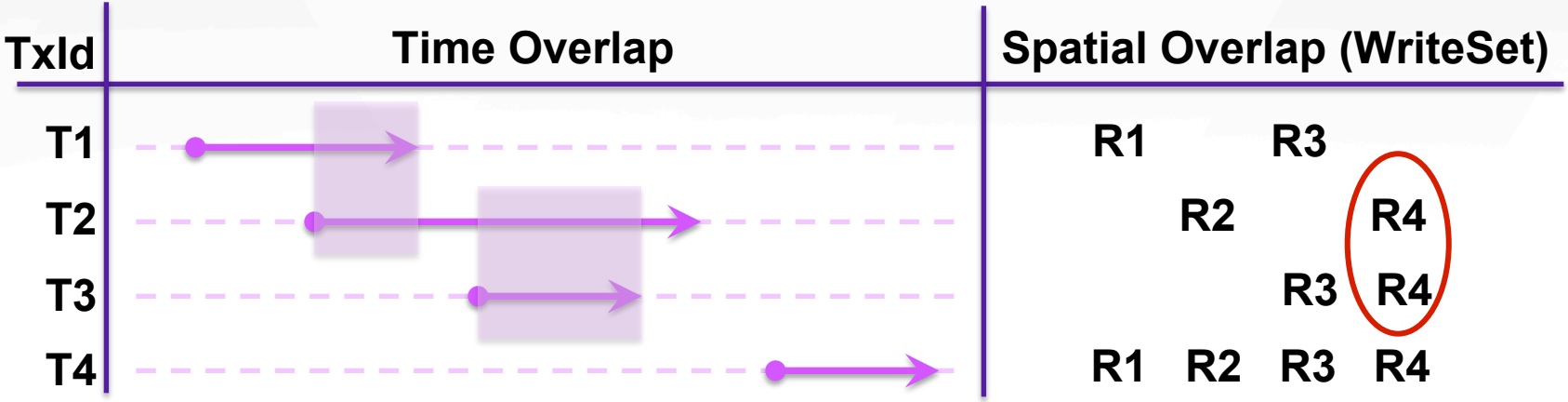
YAHOO!

Why use Omid?

- **Simplifies development** of apps requiring **consistency**
  - Multi-row/multi-table transactions on HBase
  - Simple & well-known interface
- **Good performance & reliability**
- **Lock-free**
- **Snapshot Isolation**
- **HBase is a blackbox**
  - No HBase code modification
  - No changes on table schemas
- **Used successfully at Yahoo**

YAHOO!

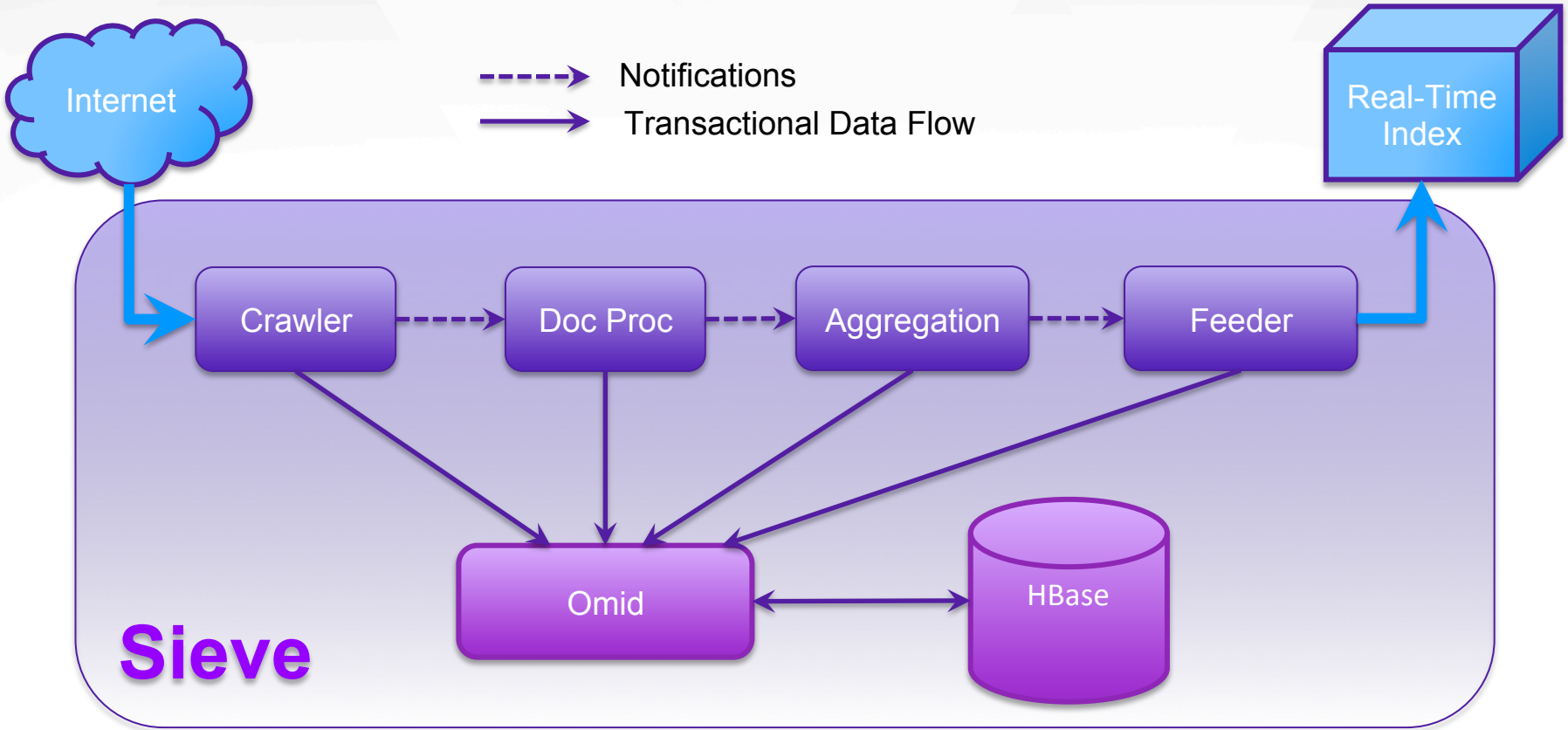
Snapshot Isolation



- Transaction T2 overlaps in time with T1 & T3, but spatially:
  - T1 ∩ T2 = ∅
  - T2 ∩ T3 = { R4 } Transactions T2 and T3 conflict
- Transaction T4 does not have conflicts

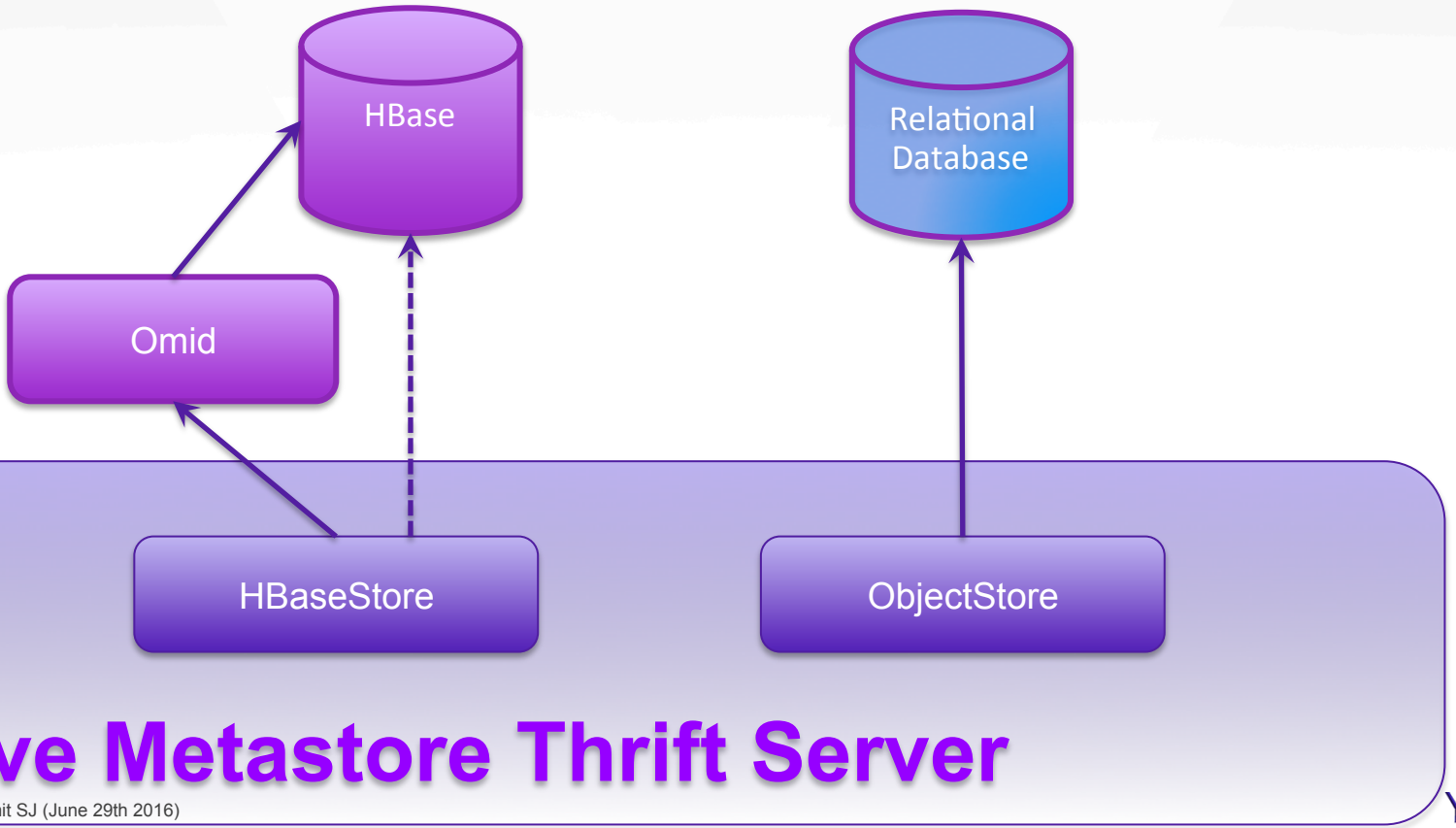
YAHOO!

Use Cases: Sieve @ Yahoo

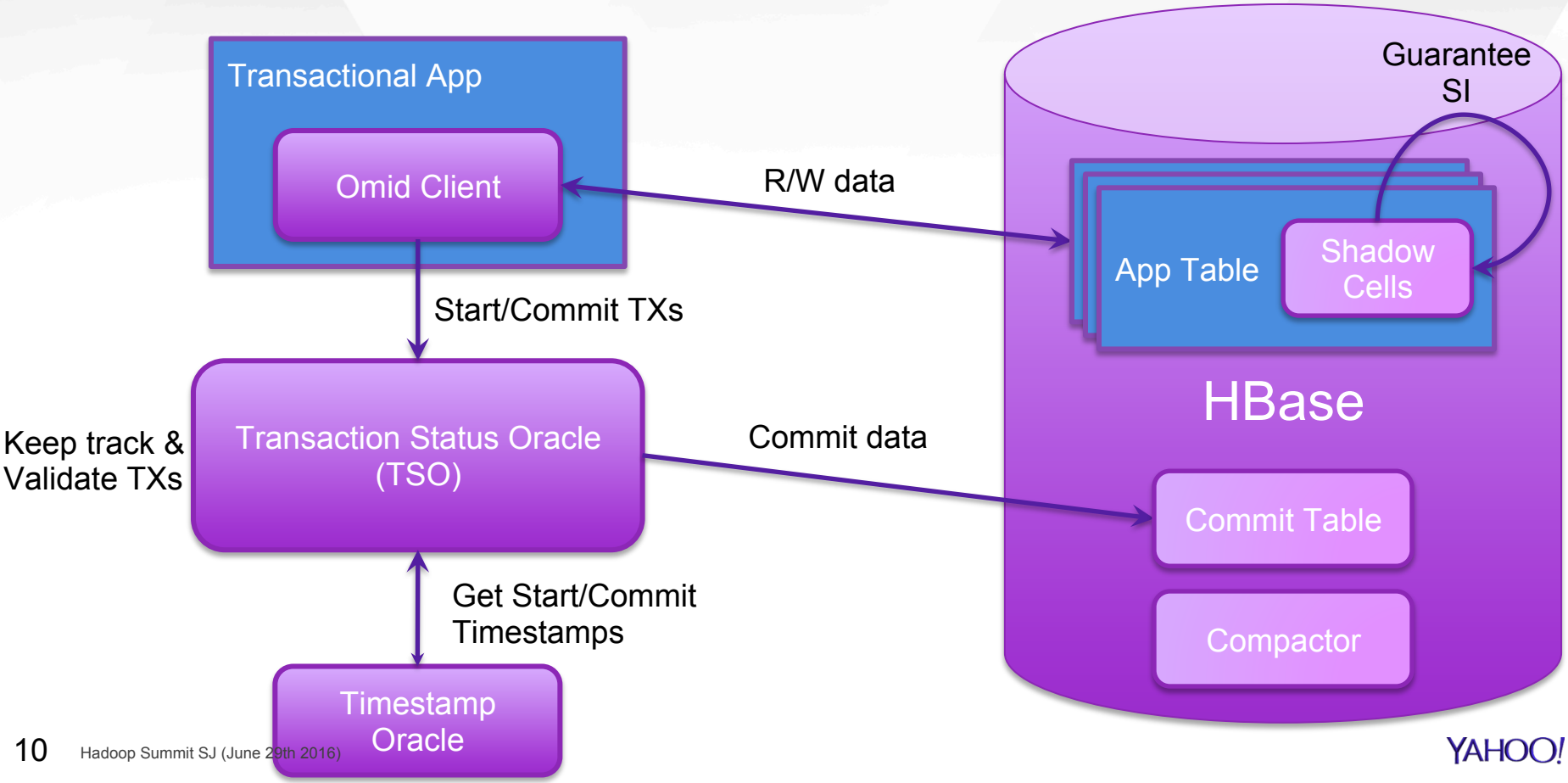


YAHOO!

Use Cases:


HBase
Relational Database
Omid
HBaseStore
ObjectStore
Hive Metastore Thrift Server

Architectural Components



YAHOO!

Client APIs

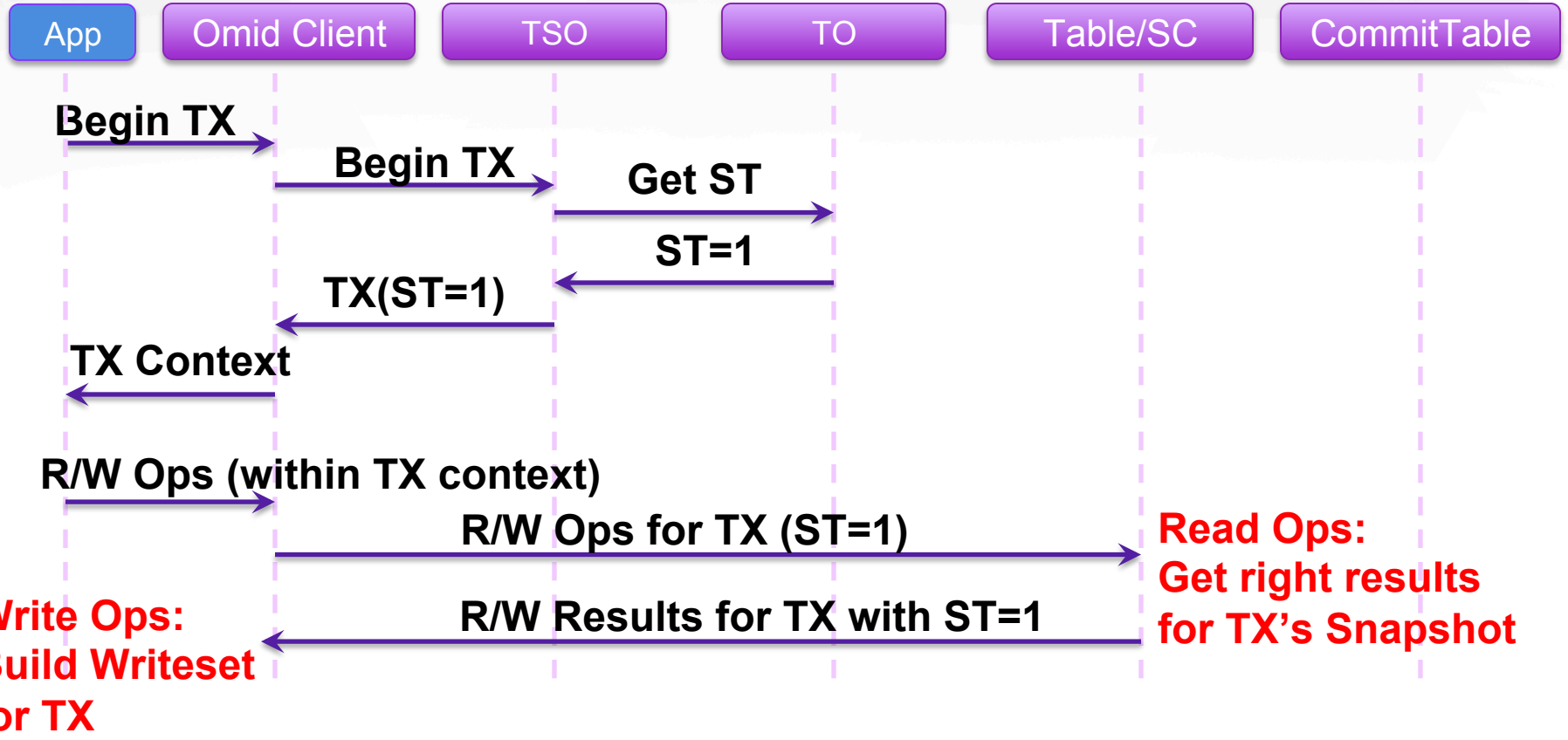
- Transaction Manager → Create Transactional contexts

```
Transaction begin();
void commit(Transaction tx);
void rollback(Transaction tx);
```

- Transactional Tables (TTable) → Data access

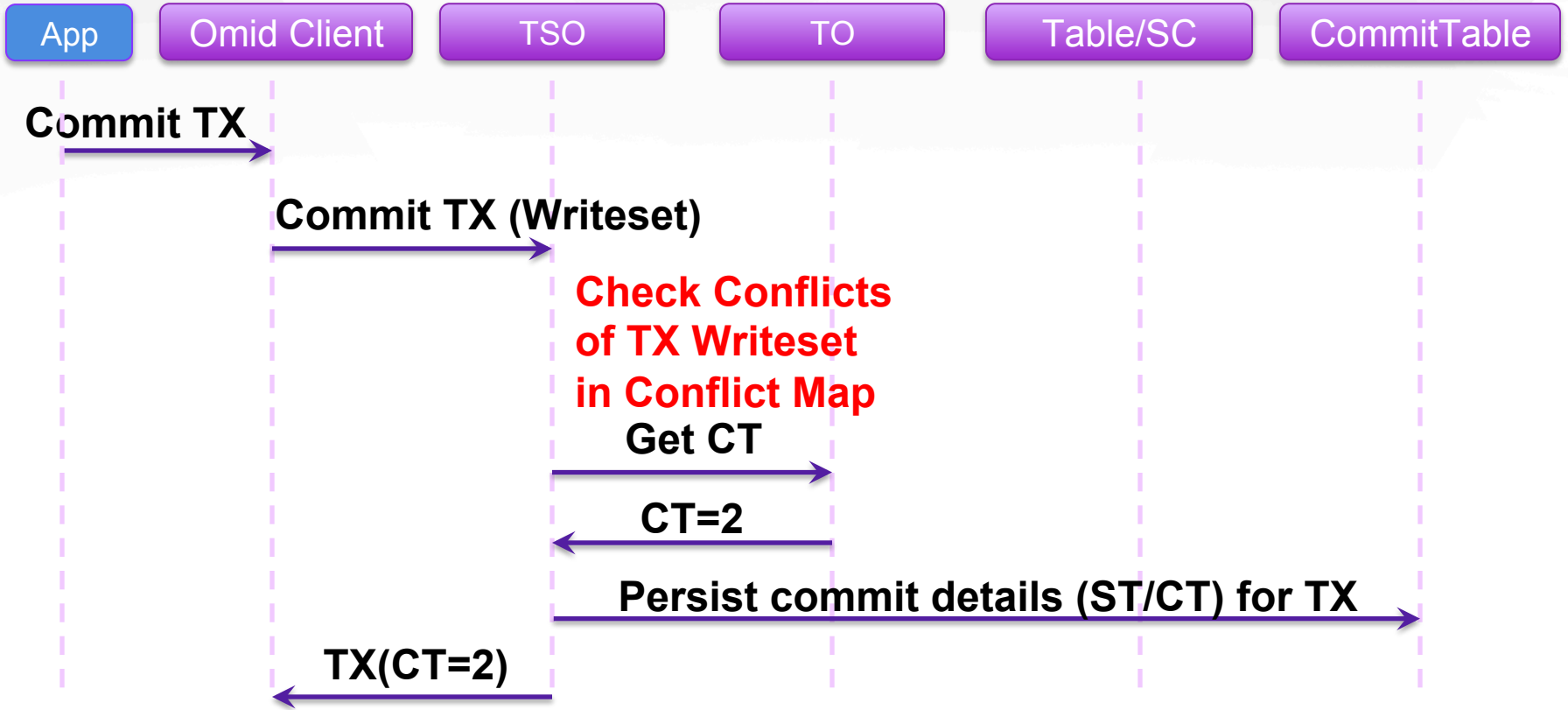
```
Result get(Transaction tx, Get g);
void put(Transaction tx, Put p);
ResultScanner getScanner(Transaction tx, Scan s);
```

TX Management (Begin TX phase)



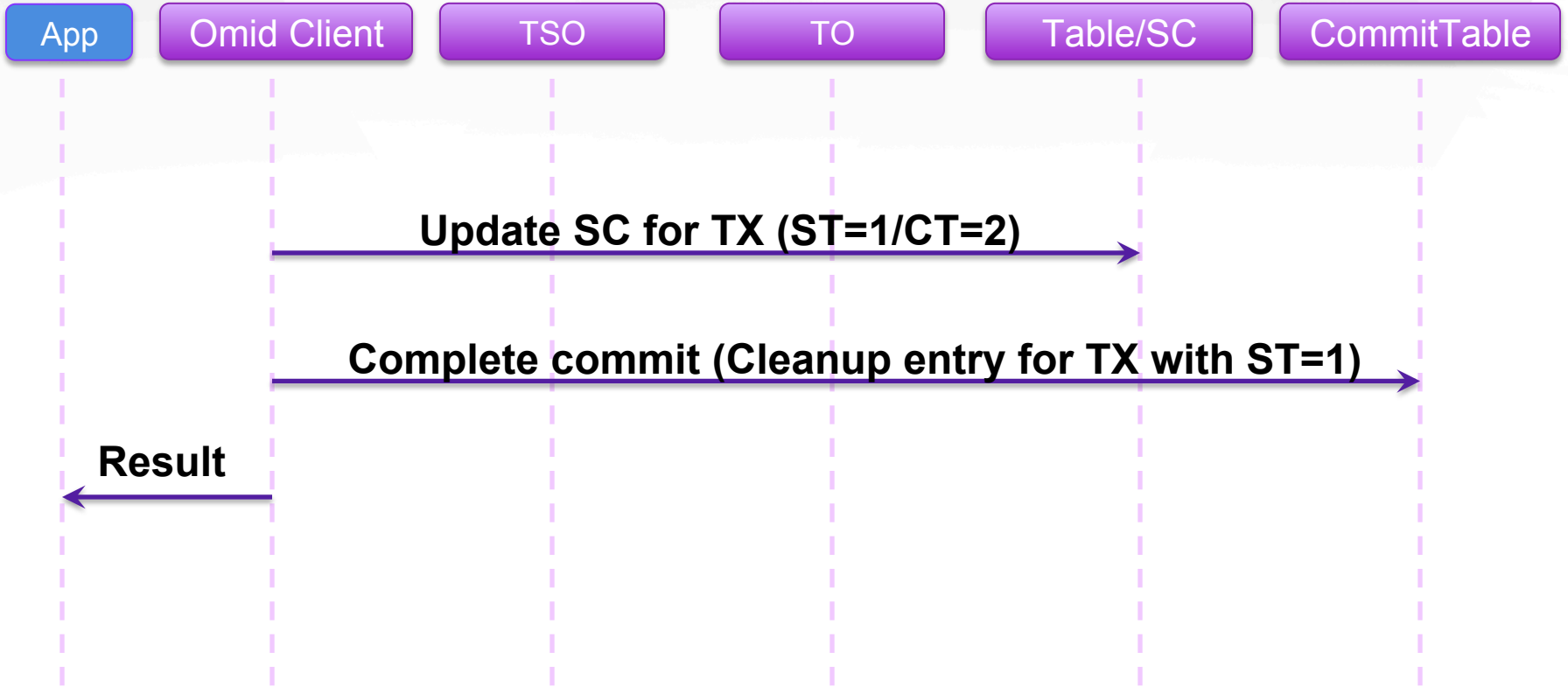
YAHOO!

TX Management (Commit TX Phase)



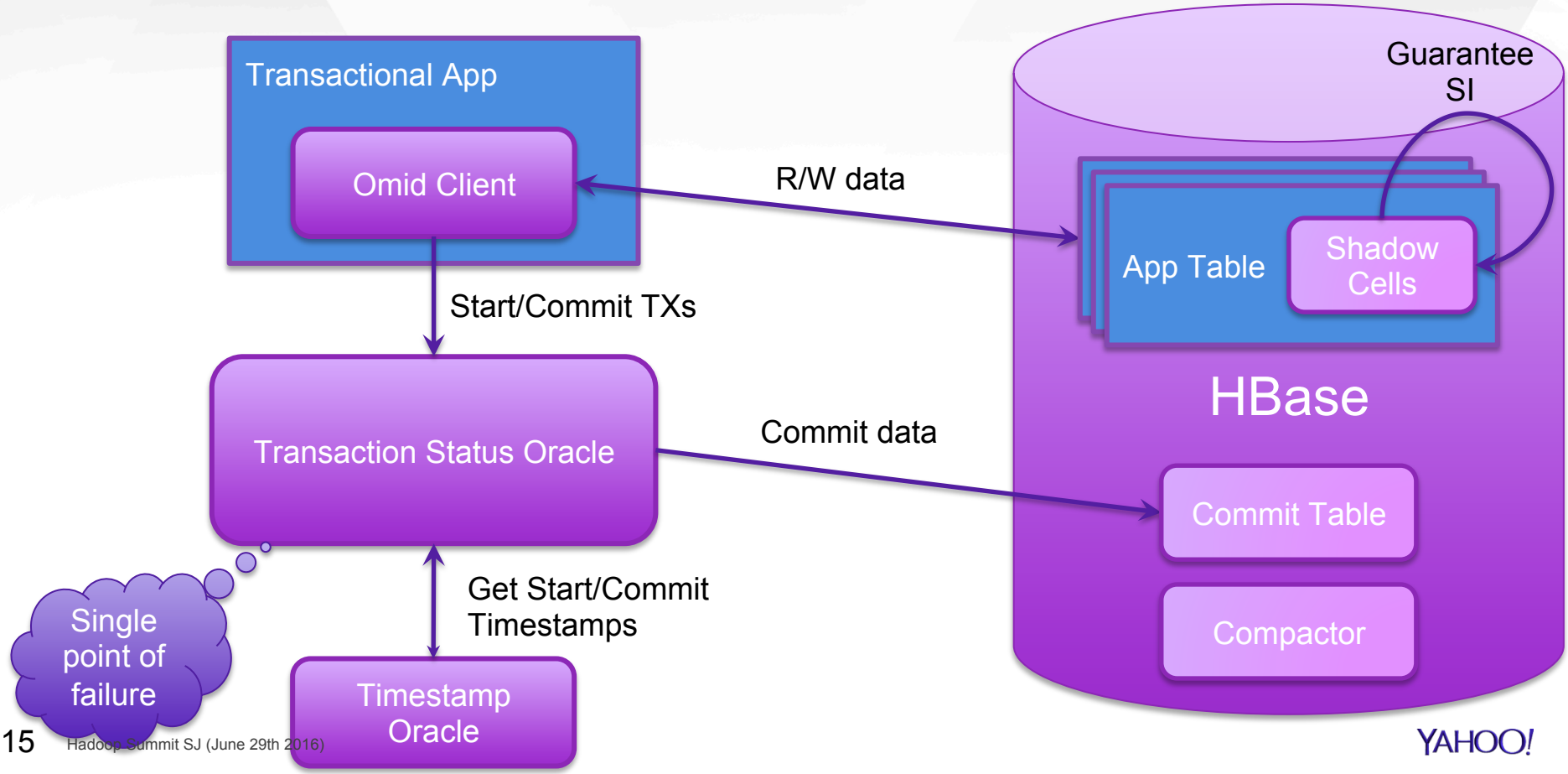
YAHOO!

TX Management (Complete TX Phase)



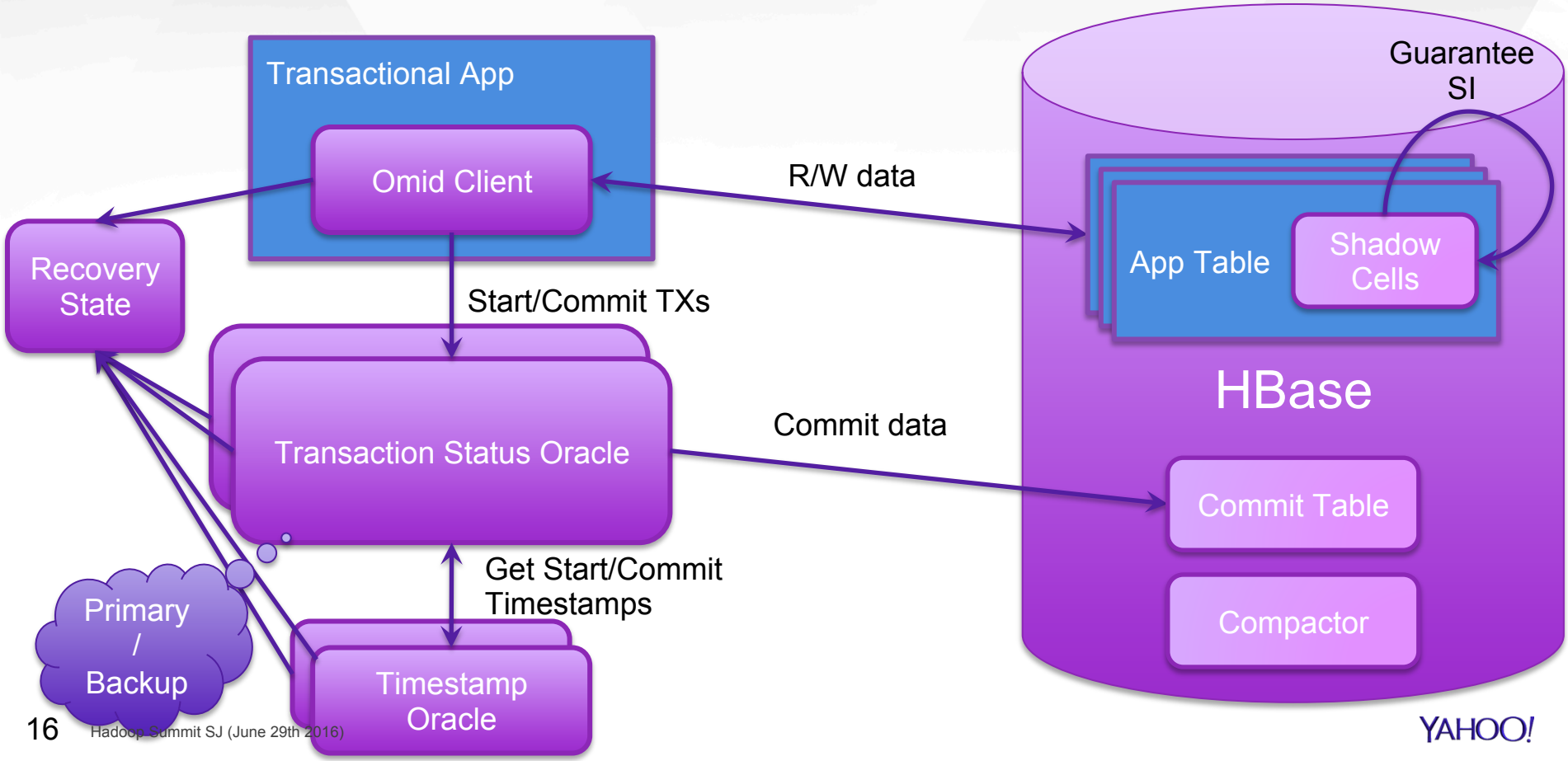
YAHOO!

High Availability



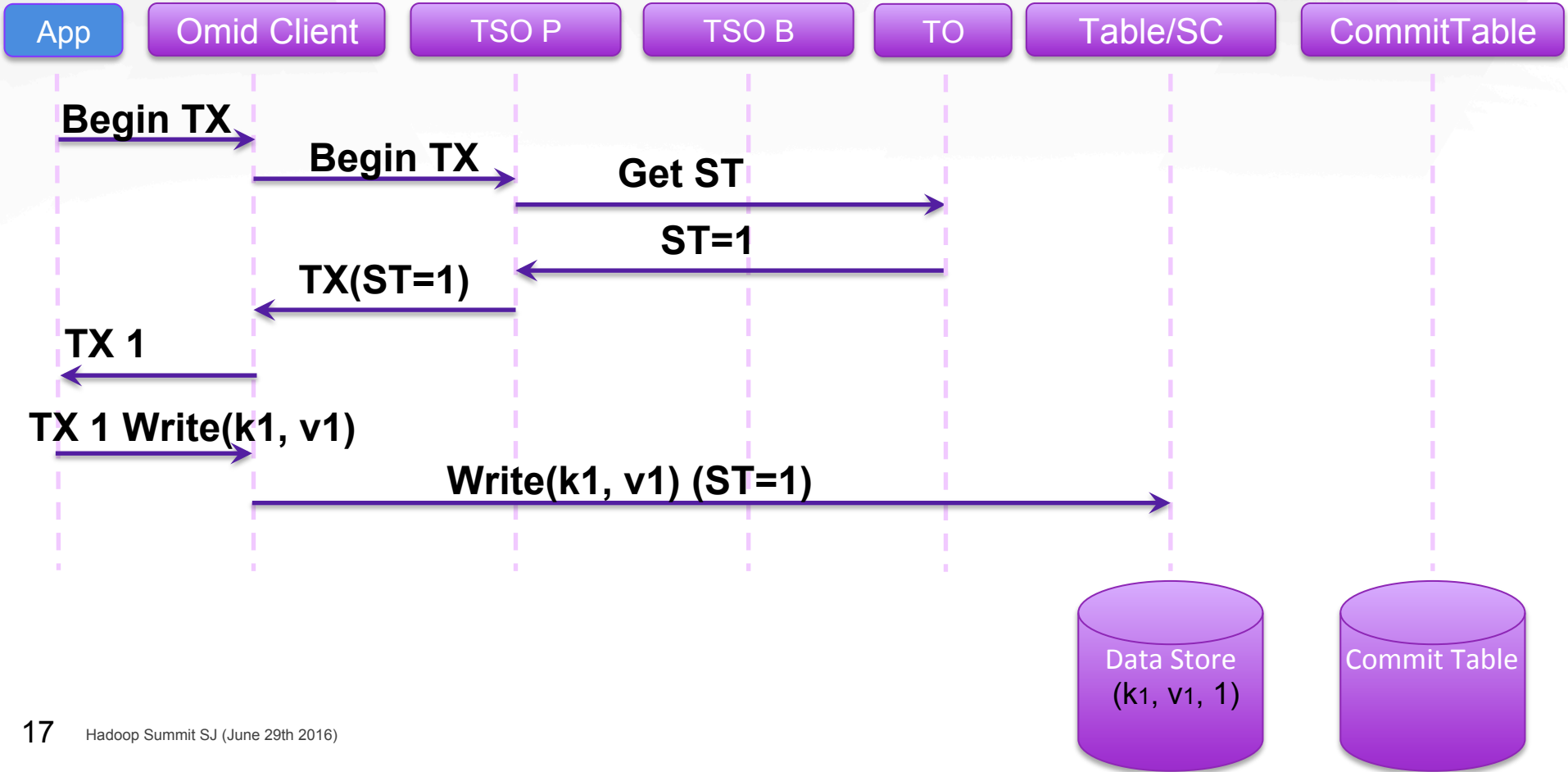
YAHOO!

High Availability

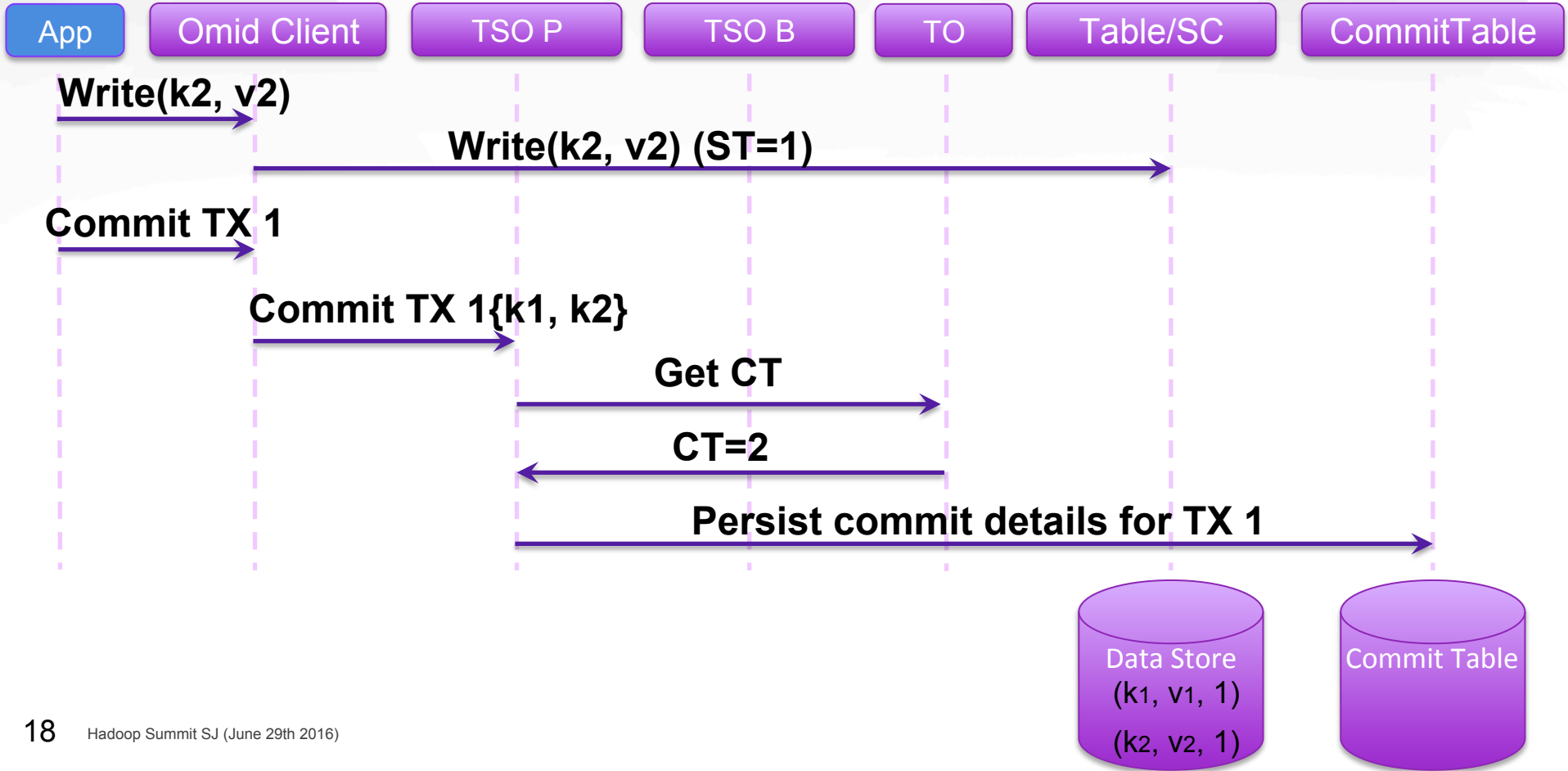


YAHOO!

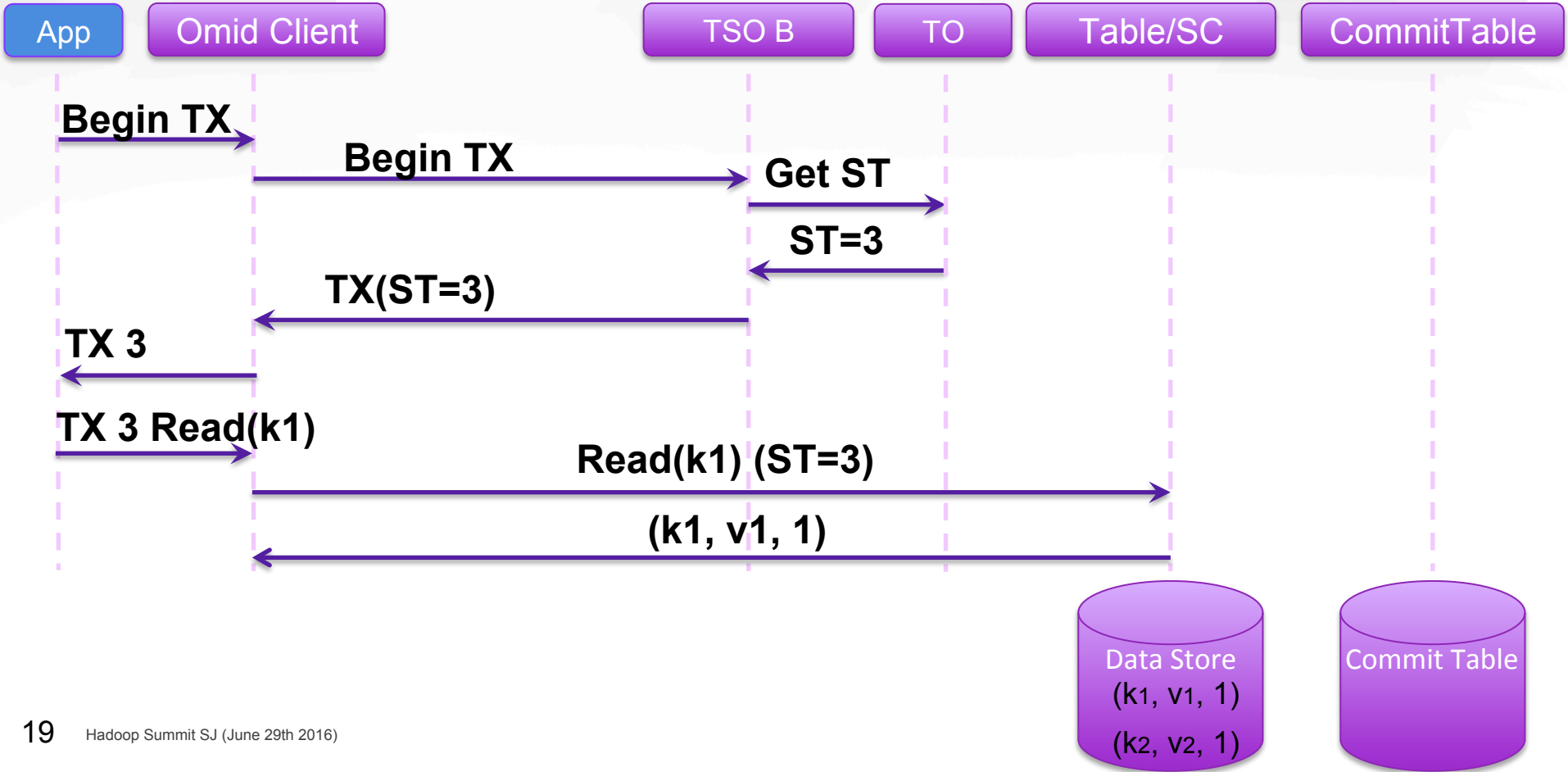
High Availability – Failing Scenario



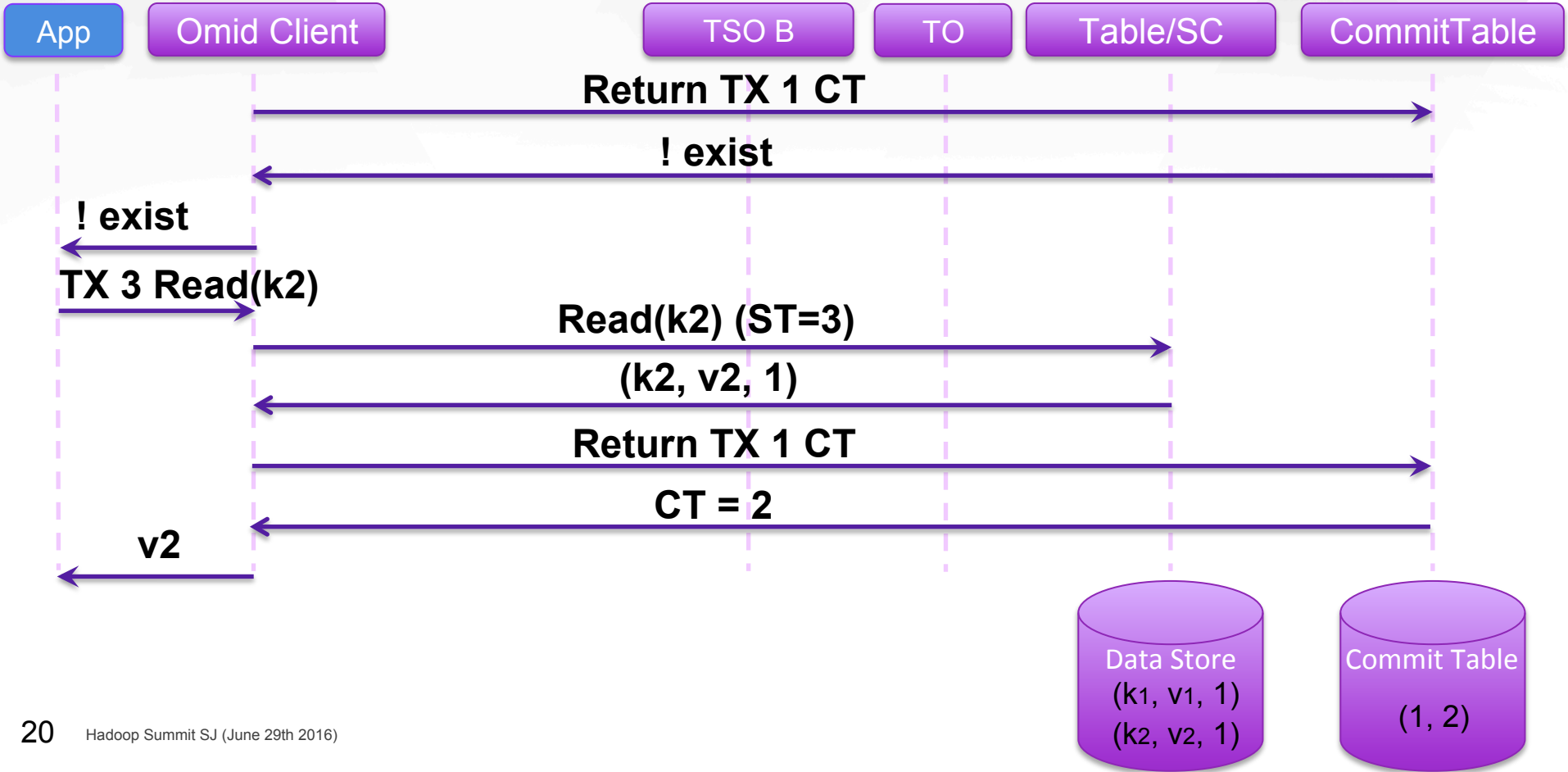
High Availability – Failing Scenario



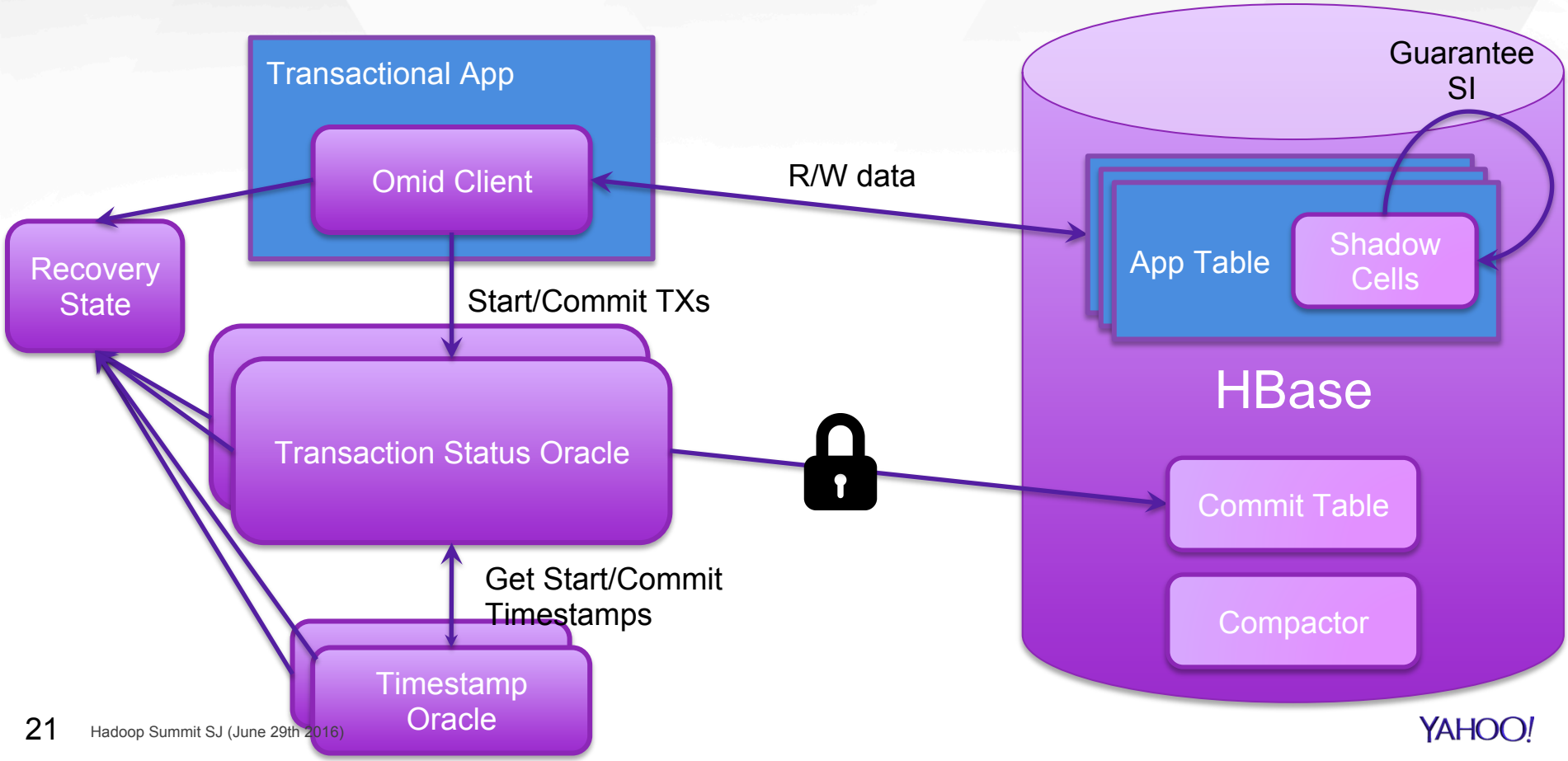
High Availability – Failing Scenario



High Availability – Failing Scenario

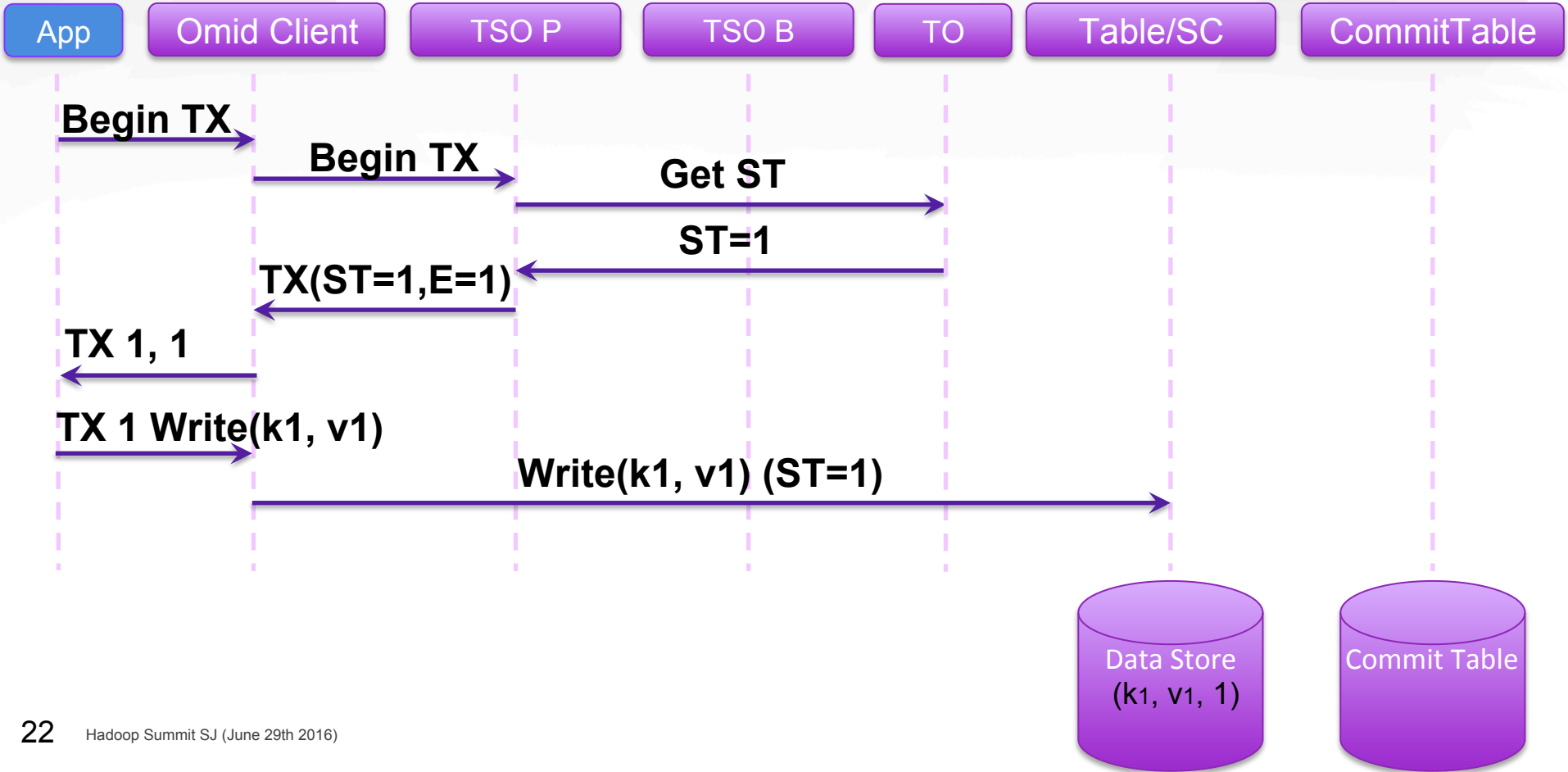


High Availability

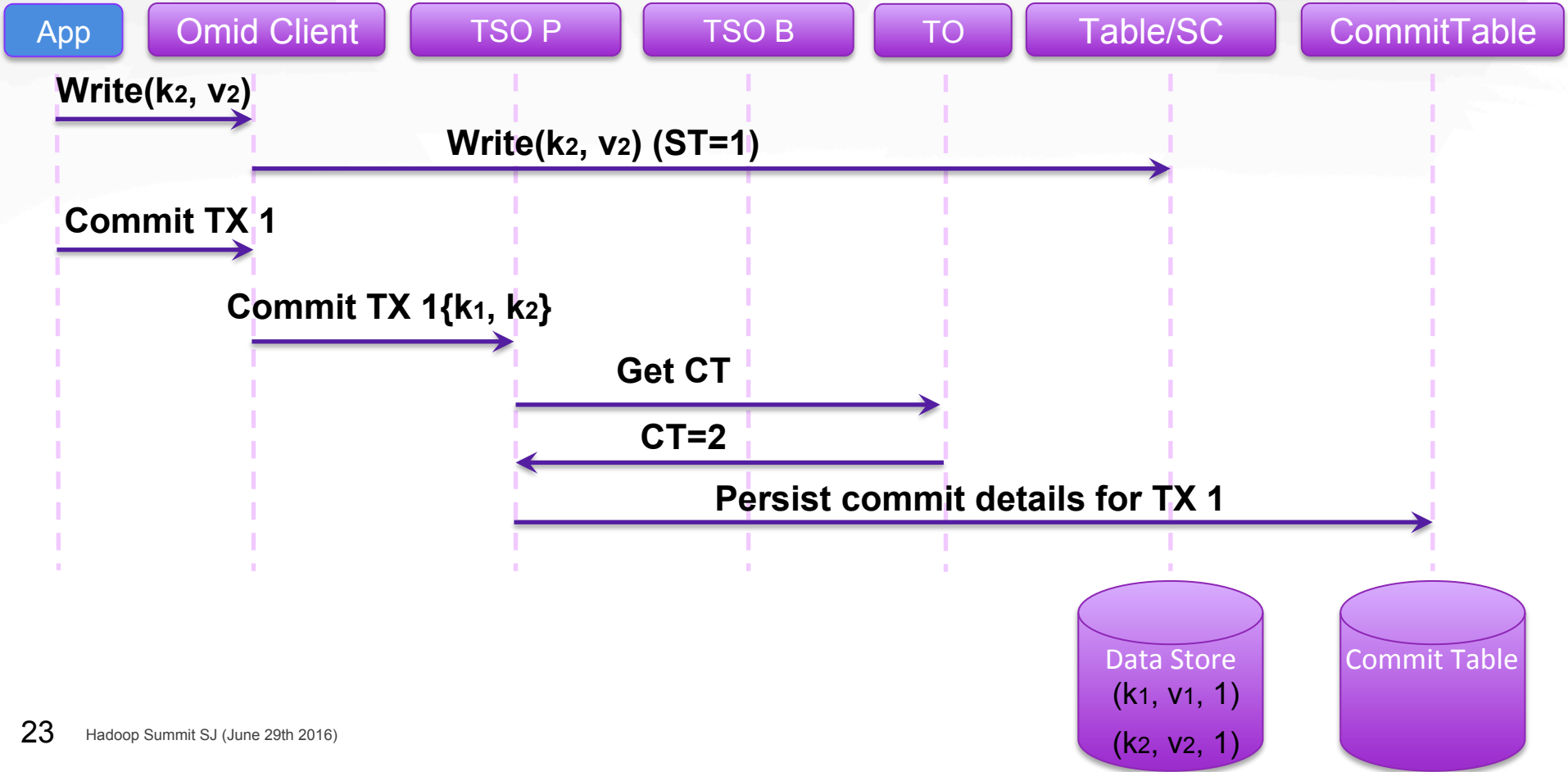


YAHOO!

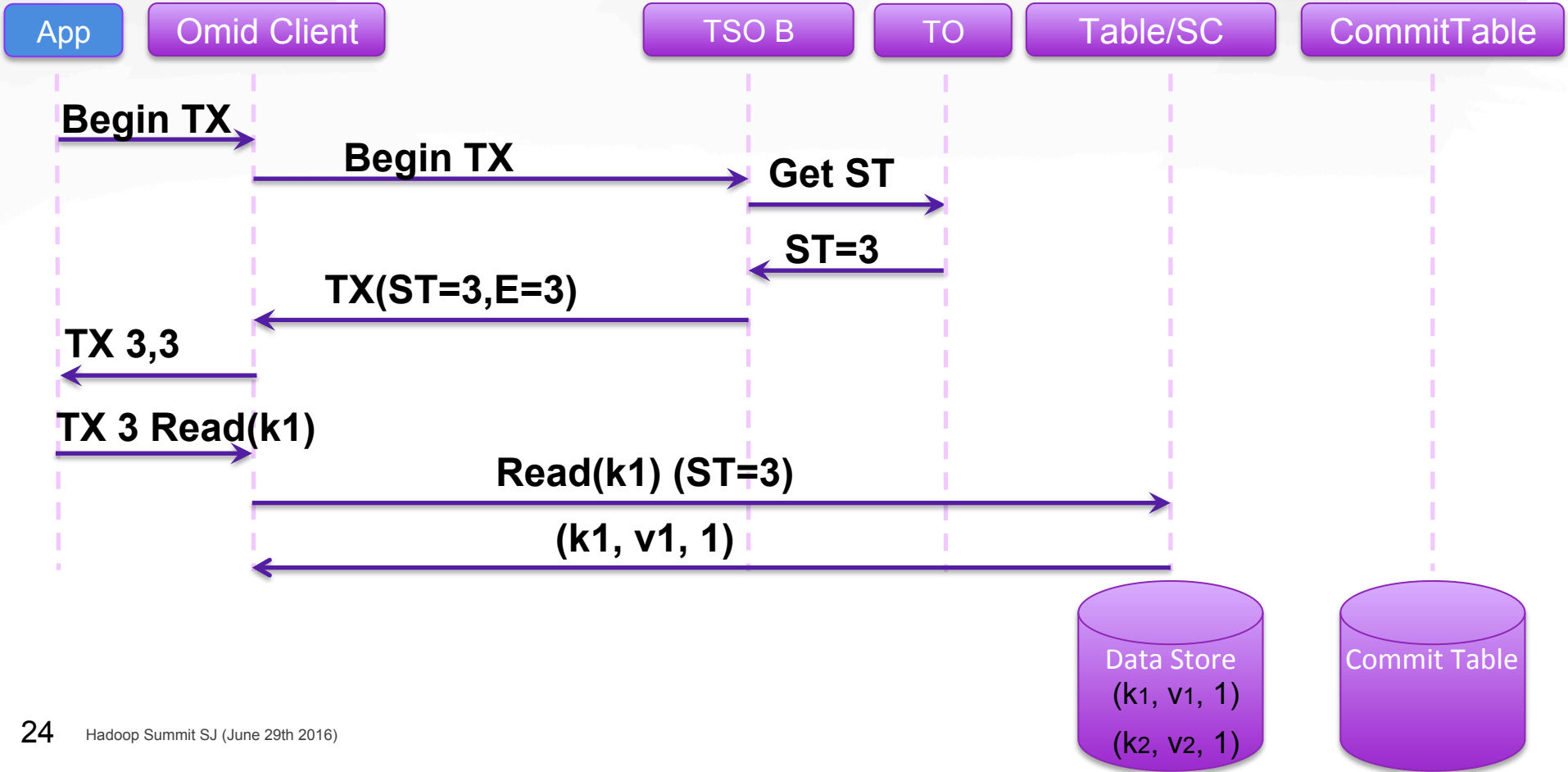
High Availability – Solution



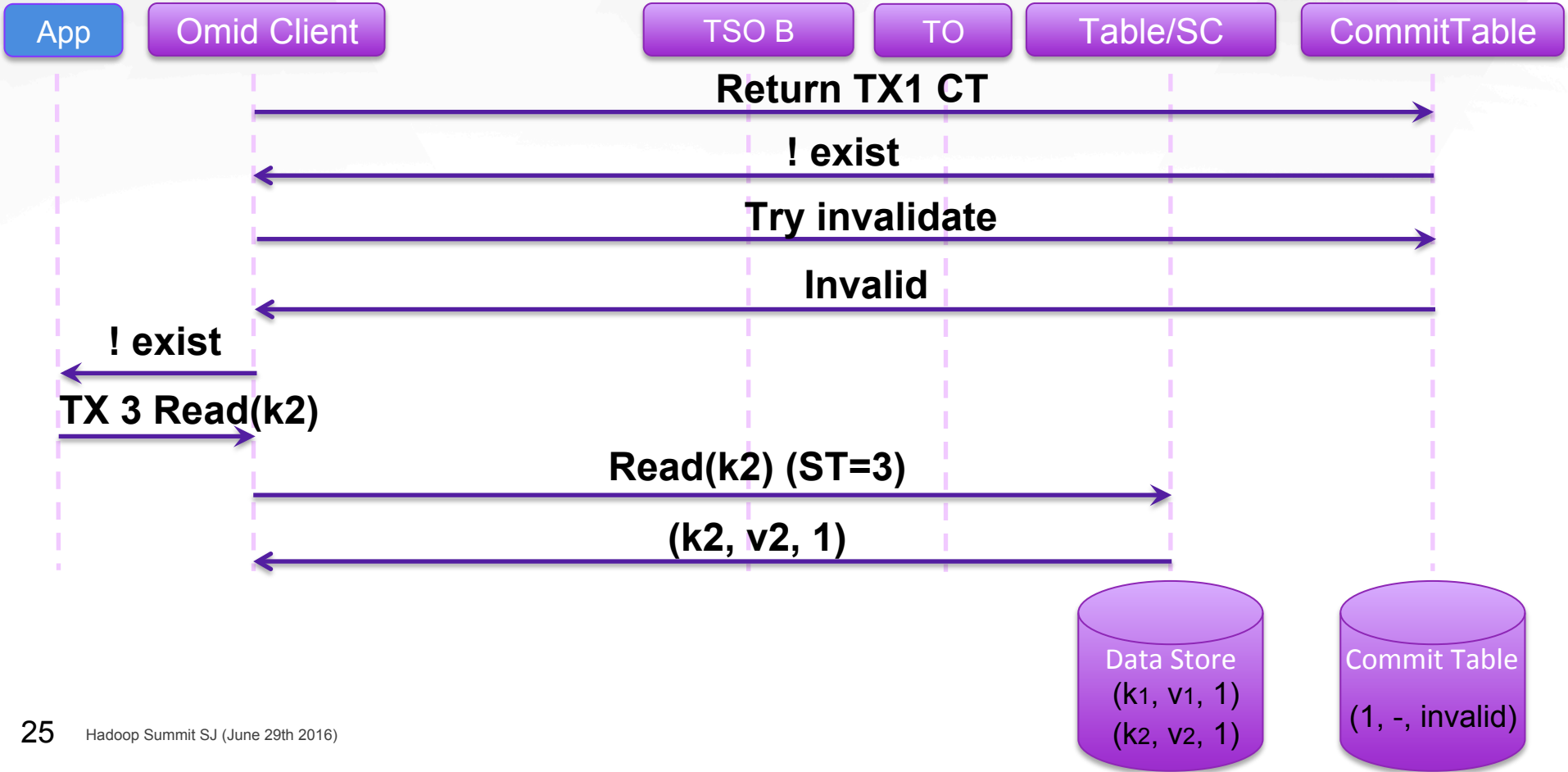
High Availability – Solution



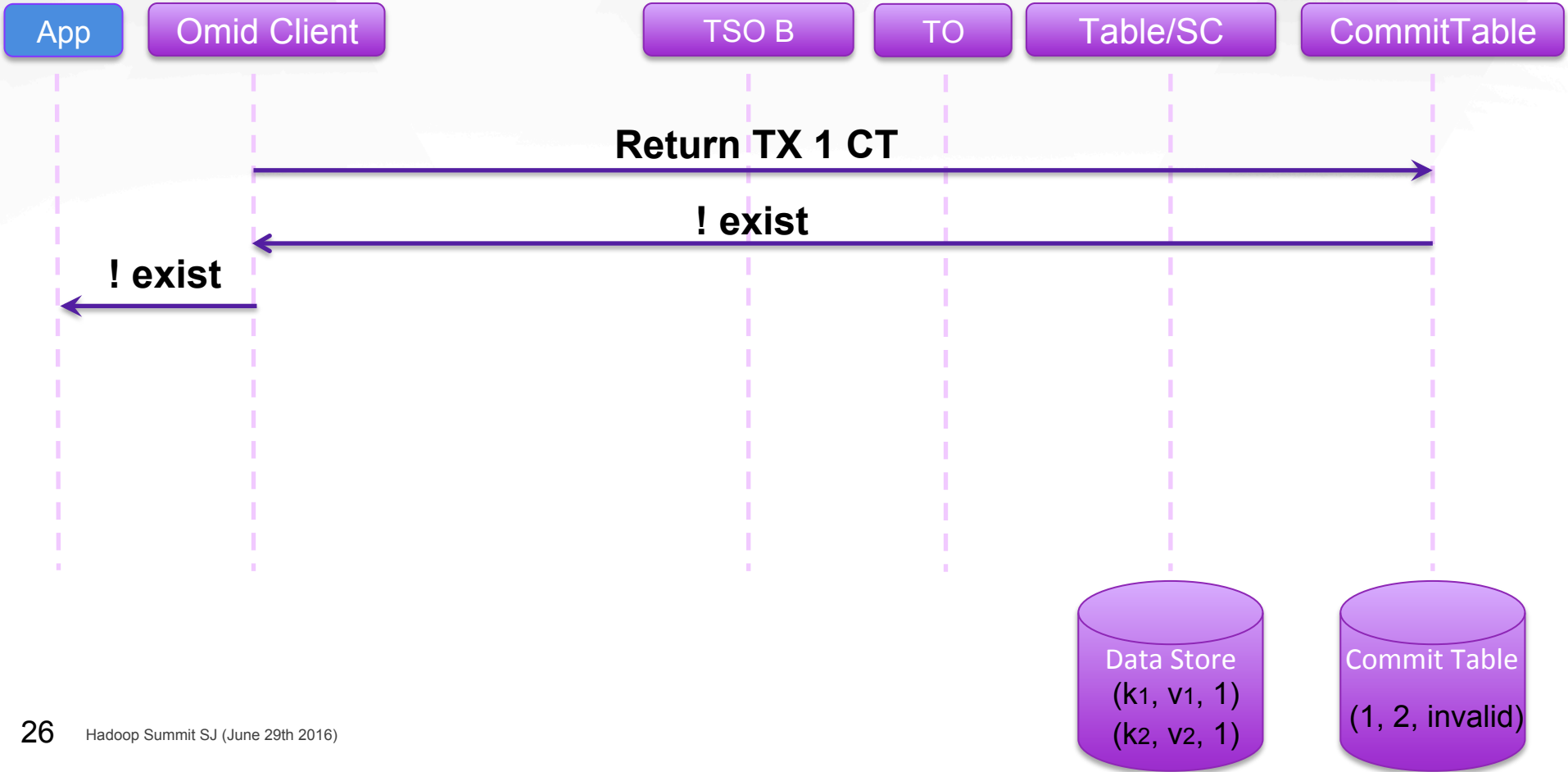
High Availability – Solution



High Availability – Solution



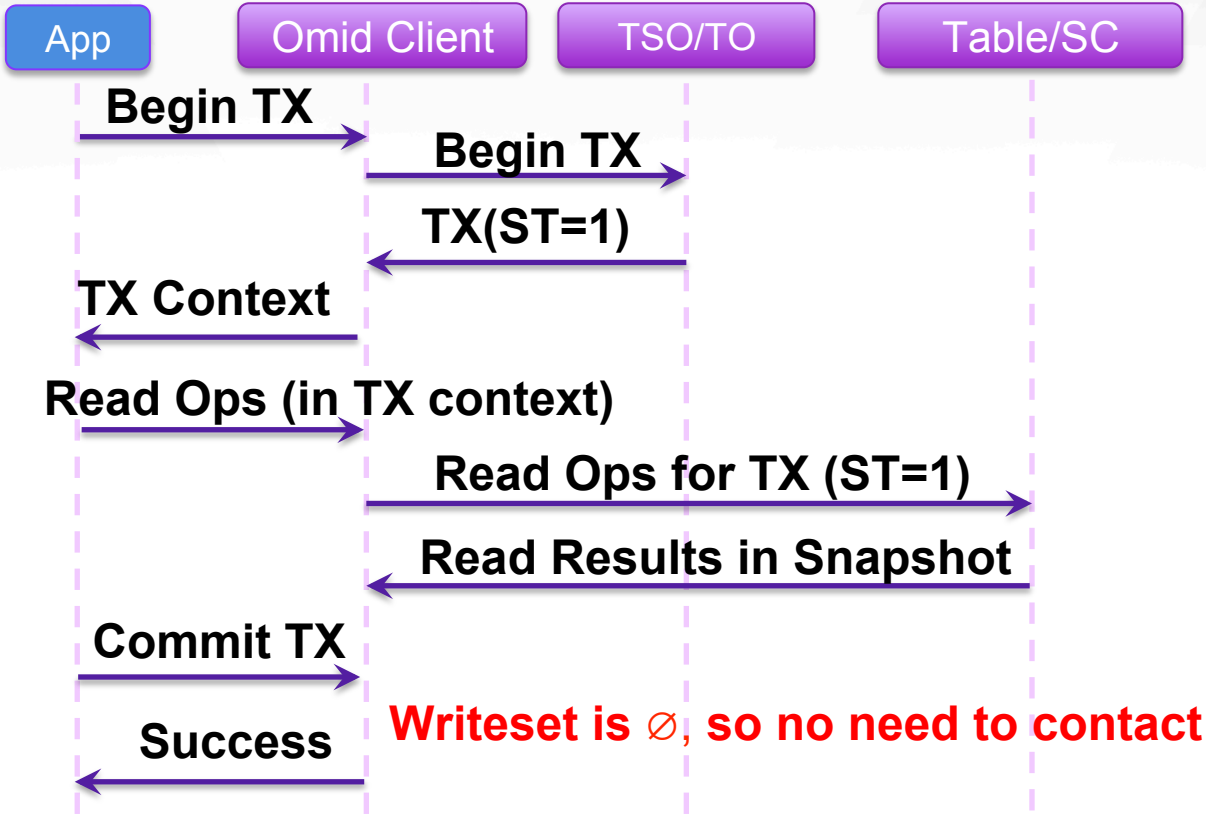
High Availability – Solution



High Availability

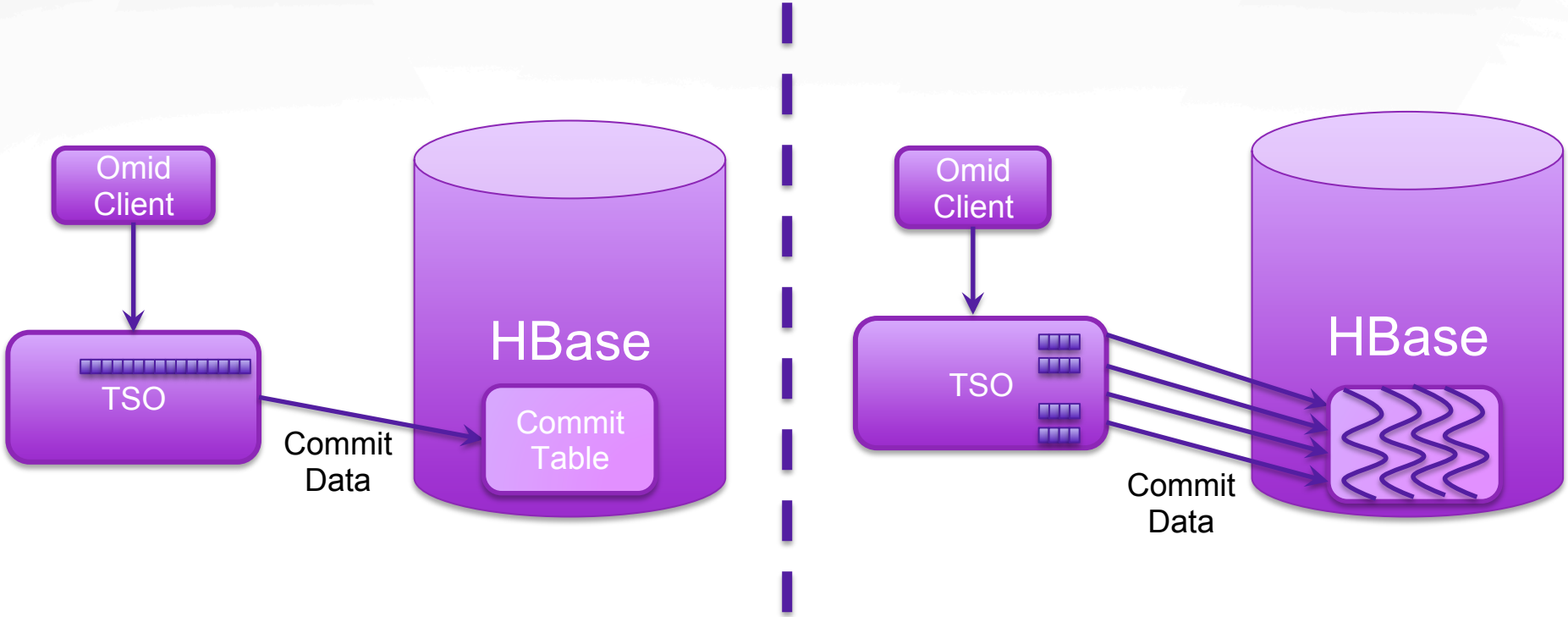
- **No runtime overhead** in mainstream execution
  - Minor overhead after failover
- TSO uses **regular writes**
- **Leases** for leader election
  - Lease status check before/after writing to Commit Table

Perf. Improvements: Read-Only Txs



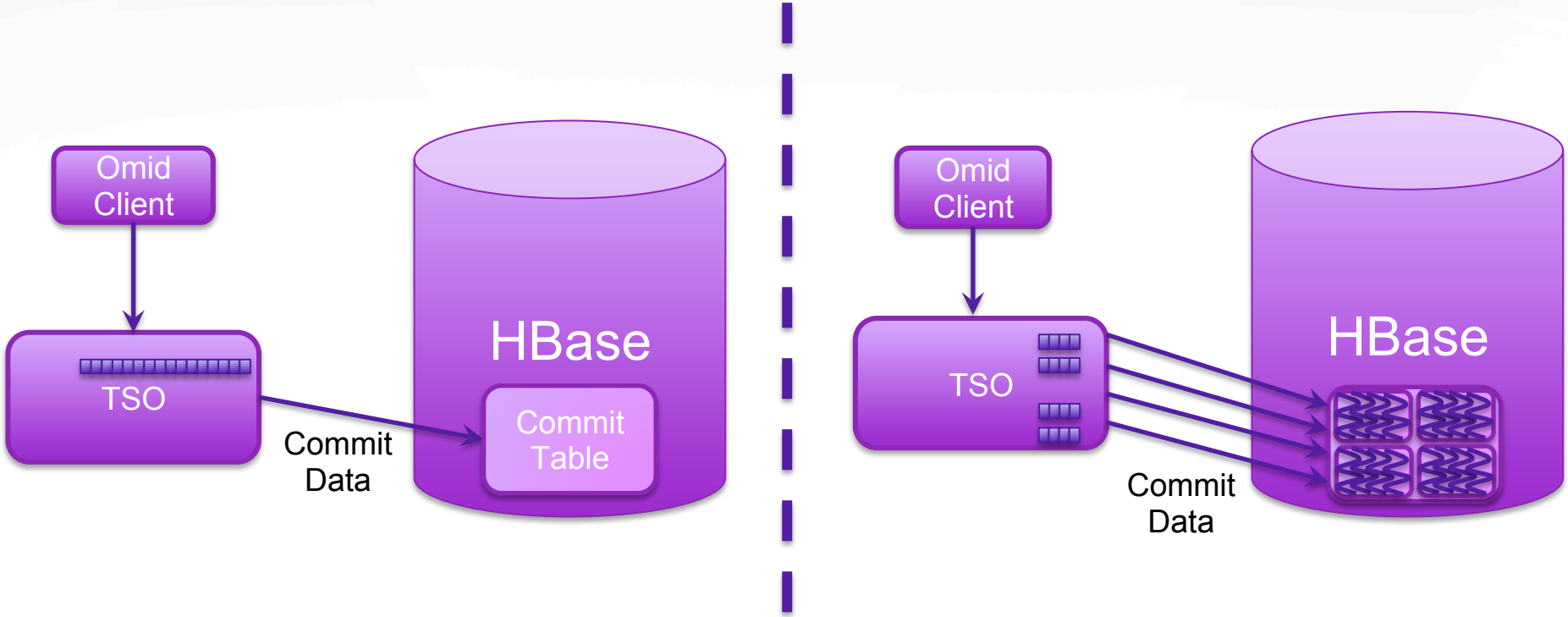
YAHOO!

Perf. Improvements: Commit Table Writes



YAHOO!

Perf. Improvements: Commit Table Writes



YAHOO!

Omid Throughput with Improvements

YAHOO!

Summary

- **Transactions in NoSQL**
  - Use cases in incremental big data processing
  - *Snapshot Isolation*: Scalable consistency model
- **Omid**
  - Web-scale TPS for HBase
  - Reliable and performant
  - Battle-tested
  - 

    http://omid.incubator.apache.org/

YAHOO!

Questions?

@ApacheOmid

Apache Omid Incubator Page