Omid

Efficient Transaction Management and Incremental Processing for HBase

Daniel Gómez Ferro

21/03/2013

YAHOO!

About me

- Research engineer at Yahoo!
- Main Omid developer
- Apache S4 committer
- Contributor to:
  - ZooKeeper
  - HBase

Outline

- Motivation
- Goal: transactions on HBase
- Architecture
- Examples
- Future work

Motivation

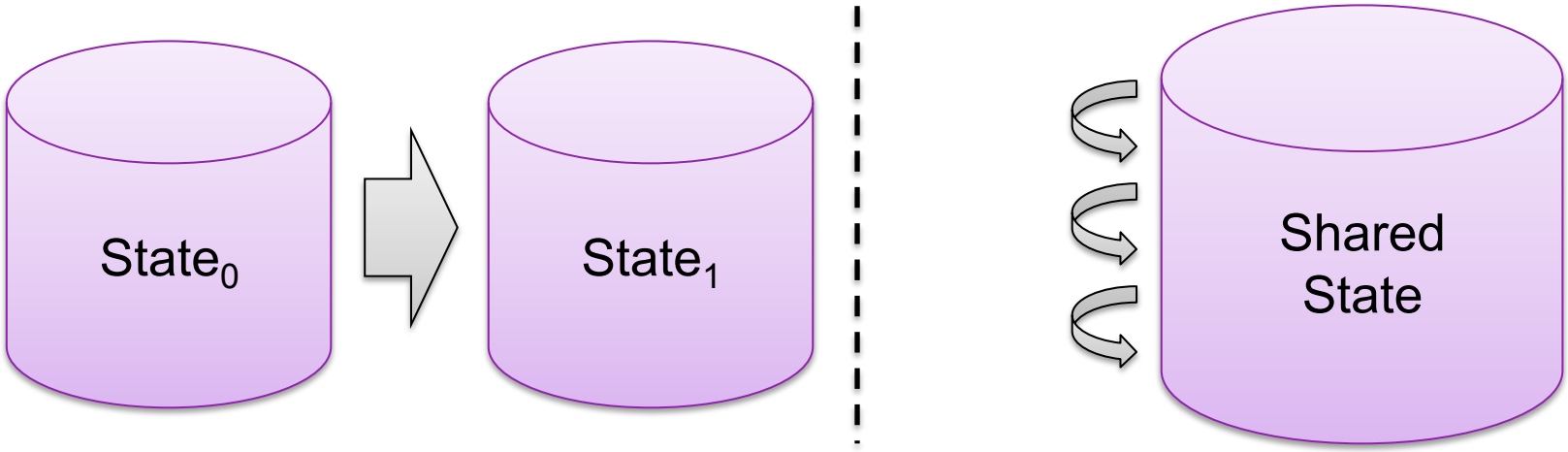
- Traditional DBMS:
  - SQL
  - Transactions
  - ~~Scalable~~
- NoSQL stores:
  - ~~SQL~~
  - ~~Transactions~~
  - Scalable
- Some use cases require
  - Scalability
  - Consistent updates

Motivation

- Requested feature
  - Support for transactions in HBase is a common request
- Percolator
  - Google implemented transactions on BigTable
- Incremental Processing
  - MapReduce latencies measured in hours
  - Reduce latency to seconds
  - Requires transactions

Incremental Processing

Offline MapReduce	Online Incremental Processing
Fault tolerance Scalability	Fault tolerance Scalability Shared state



Using Percolator, Google dramatically reduced crawl-to-index time

Transactions on HBase

Omid

- ‘Hope’ in Persian
- Optimistic Concurrency Control system
  - Lock free
  - Aborts transactions at commit time
- Implements Snapshot Isolation
- Low overhead
- http://yahoo.github.com/omid

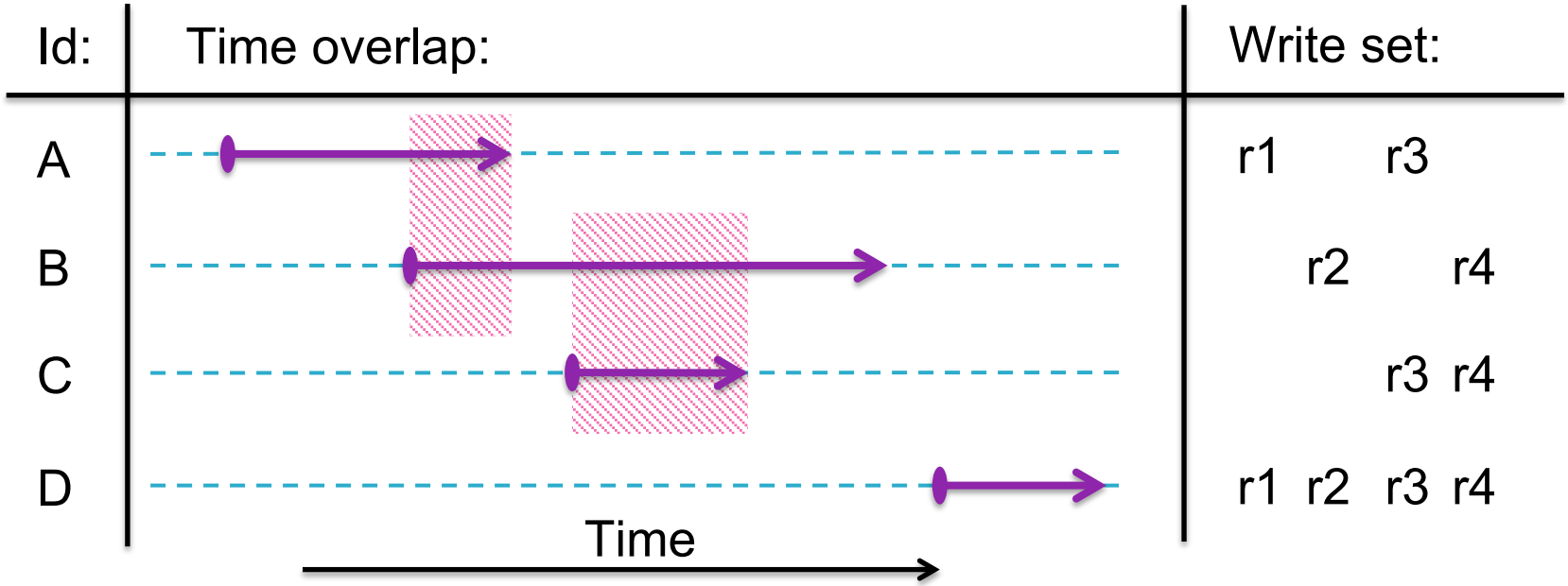
Snapshot Isolation

- Transactions read from their own snapshot
- Snapshot:
  - Immutable
  - Identified by creation time
  - Values committed before creation time
- Transactions conflict if two conditions apply:

A) Overlap in time

B) Write to the same row

Transaction conflicts

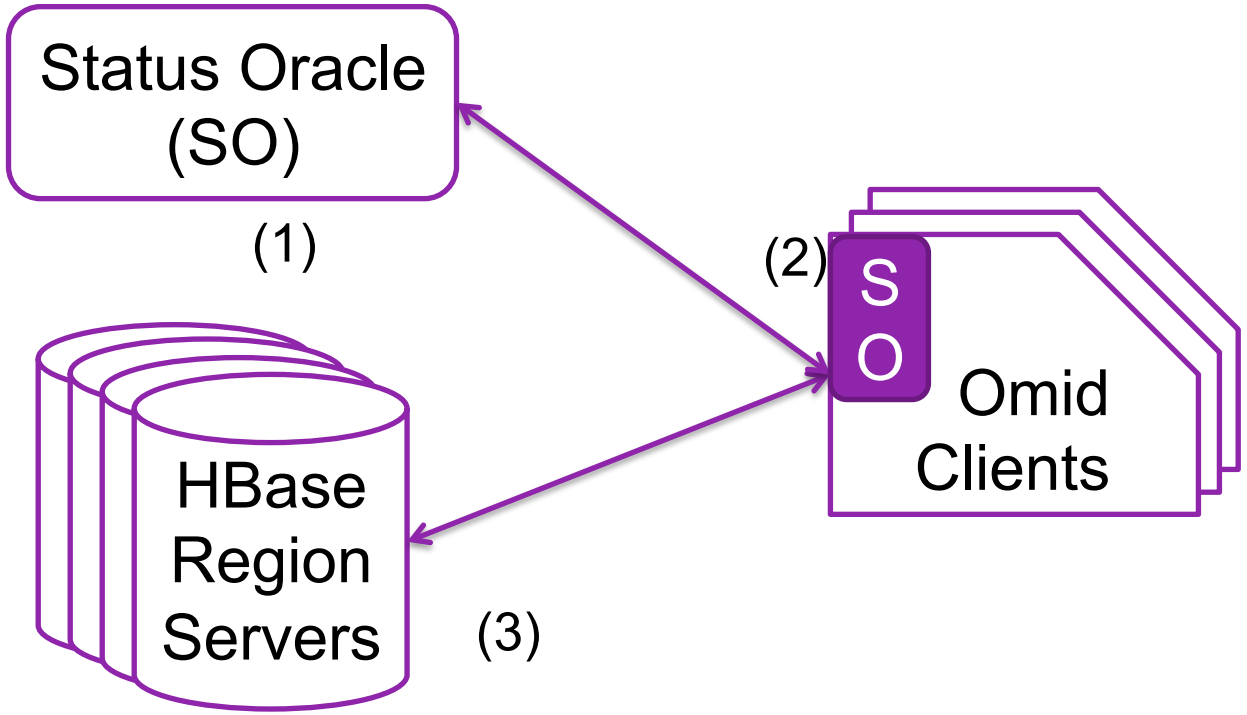


- Transaction B overlaps with A and C
  - B ∩ A = ∅
  - B ∩ C = { r4 }
  - B and C **conflict**
- Transaction D **doesn't conflict**

Omid Architecture

Architecture

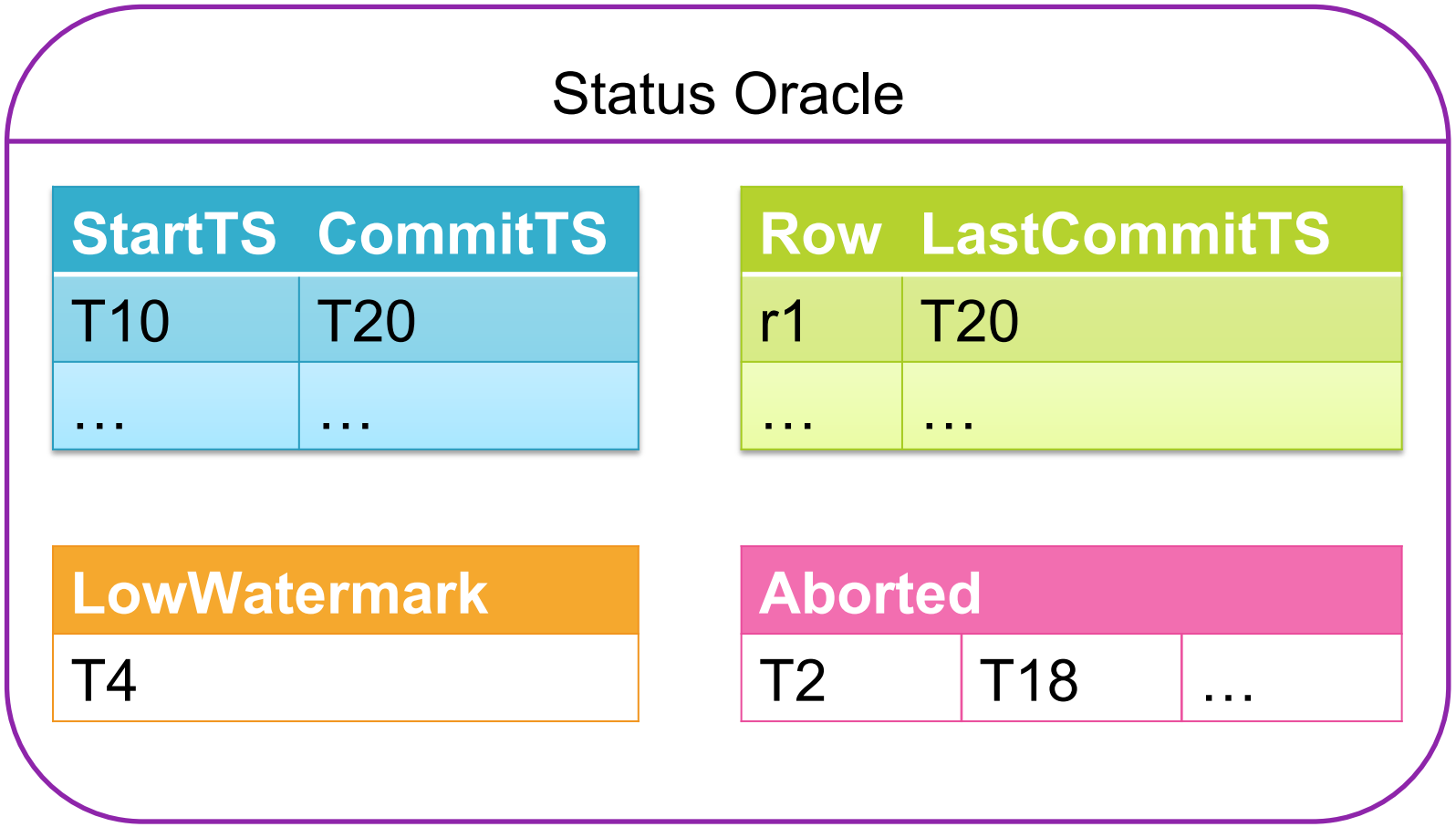
1. Centralized server
2. Transactional metadata replicated to clients
3. Store transaction ids in HBase timestamps



Status Oracle

- Limited memory
  - Evict old metadata
  - … **maintaining consistency guarantees**!
- Keep **LowWatermark**
  - Updated on metadata eviction
  - **Abort** transactions older than LowWatermark
  - Necessary for Garbage Collection

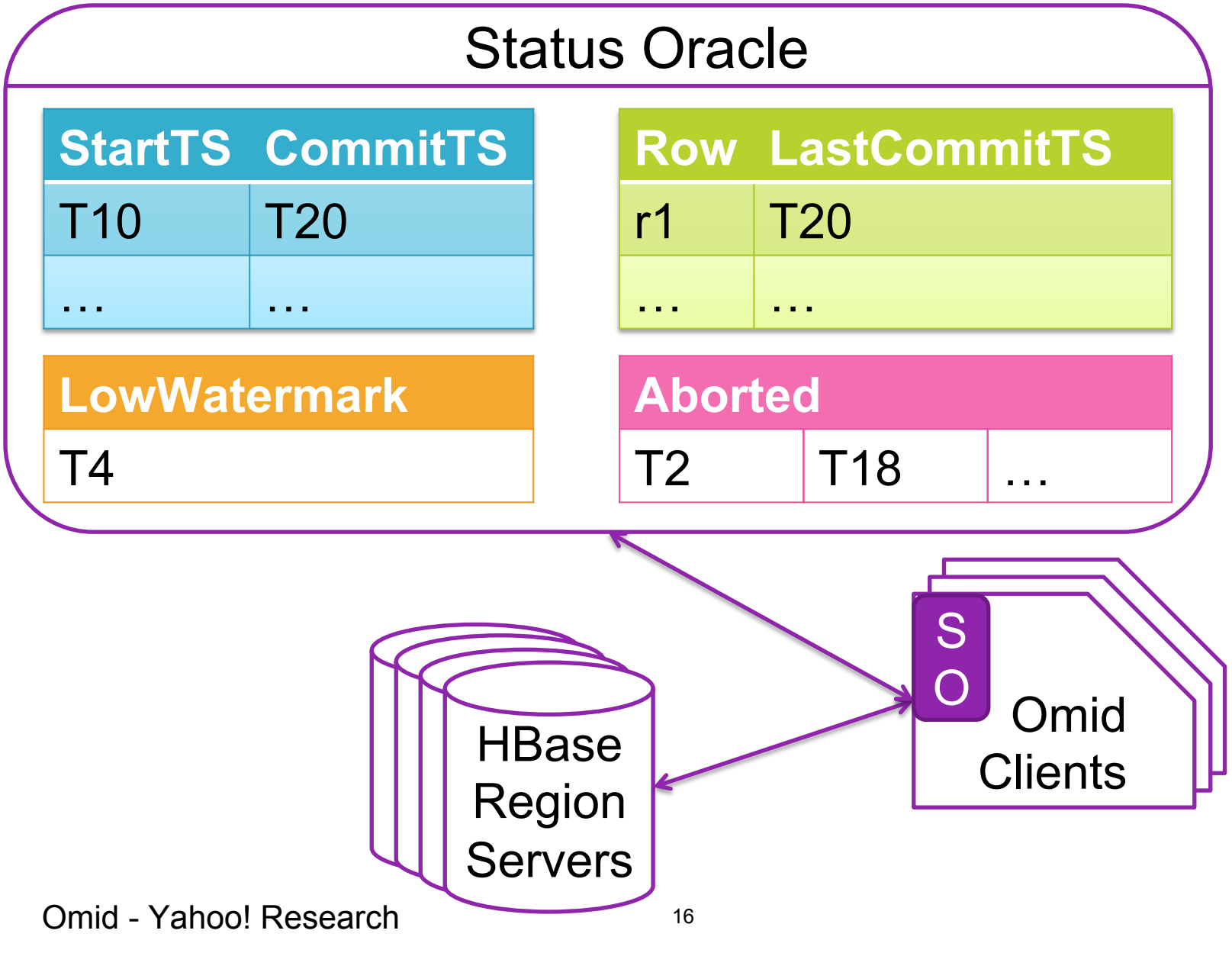
Transactional Metadata



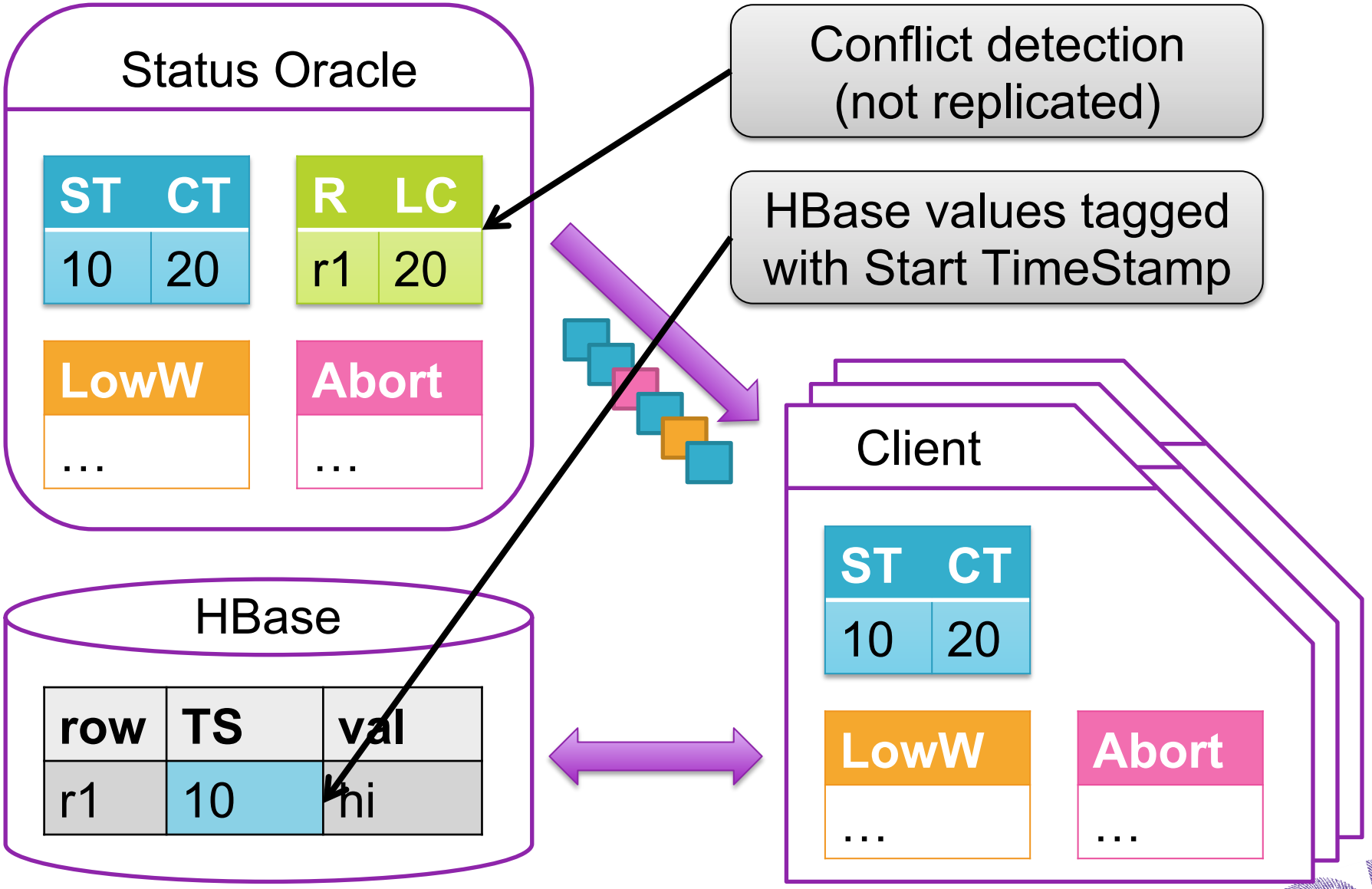
Metadata Replication

- Transactional **metadata is replicated to clients**
- Status Oracle not in critical path
  - **Minimize overhead**
- Clients guarantee Snapshot Isolation
  - Ignore data not in their snapshot
  - Talk directly to HBase
  - Conflicts resolved by the Status Oracle
- Expensive, but scalable up to 1000 clients

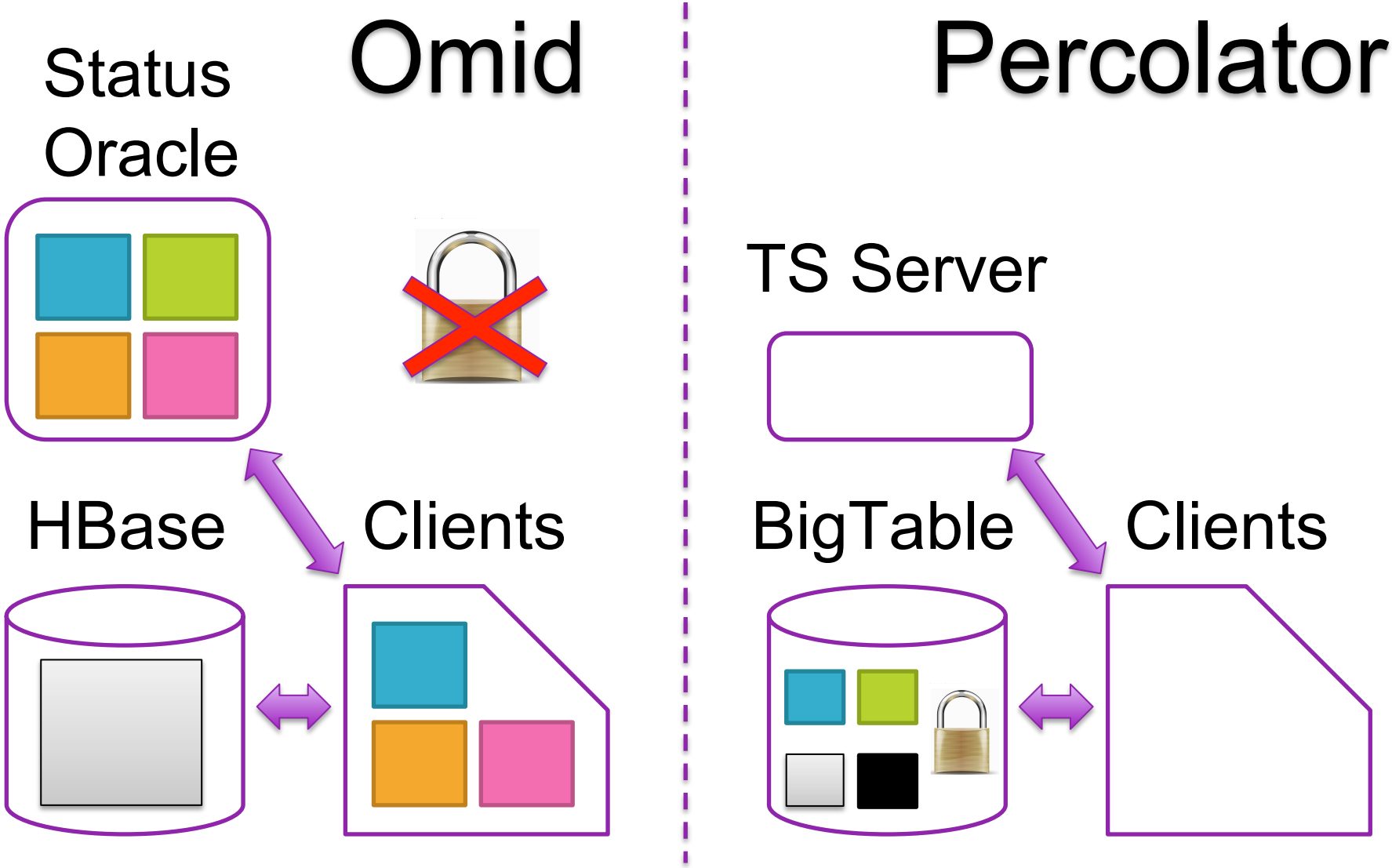
Architecture recap



Architecture + Metadata



Comparison with Percolator



Simple API

- TransactionManager:

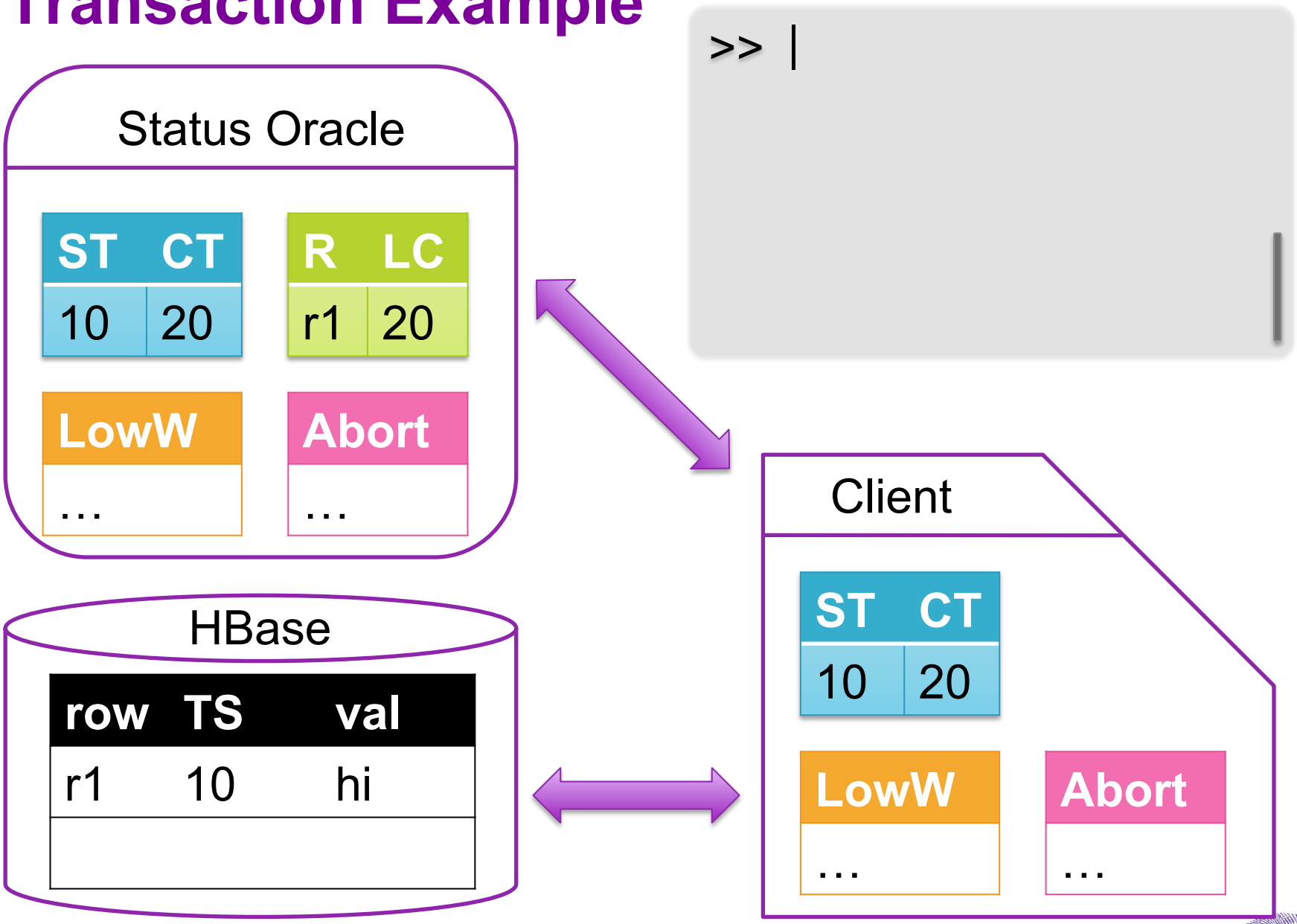
```
Transaction begin();
void commit(Transaction t) throws RollbackException;
void rollback(Transaction t);
```

- TTable:

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

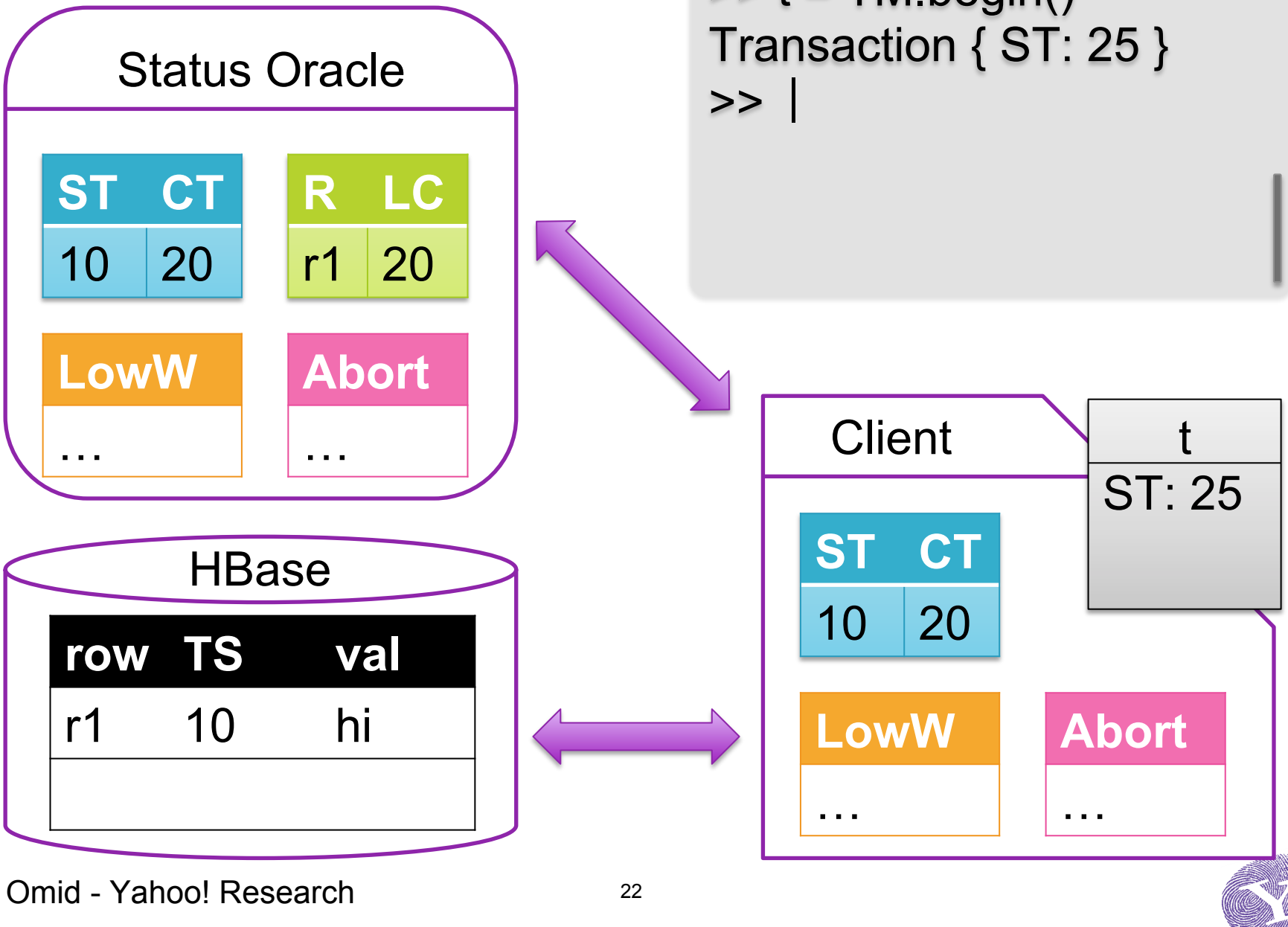

Example Transaction

Transaction Example



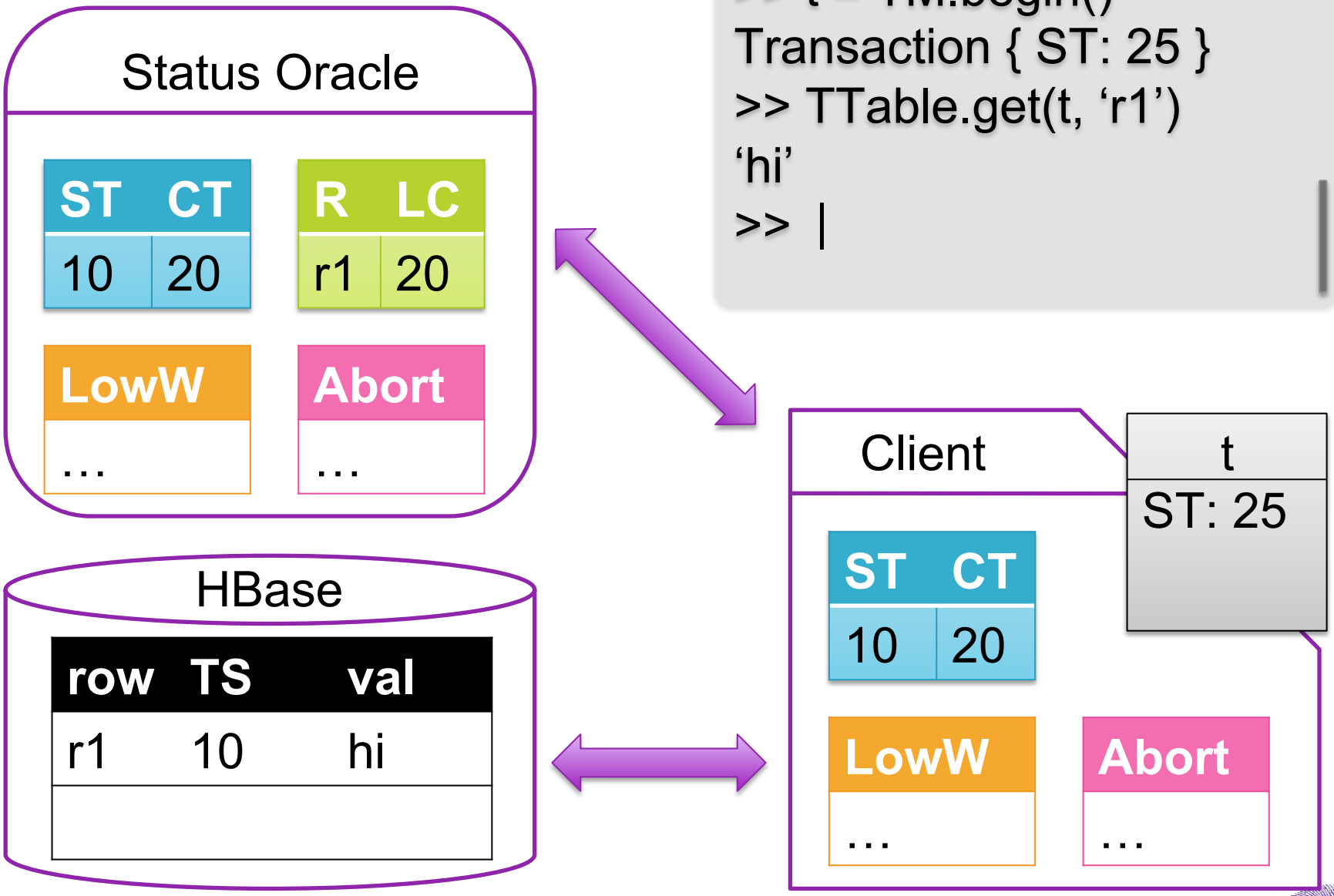
Transaction Example

```
>> t = TM.begin()
Transaction { ST: 25 }
>>
```



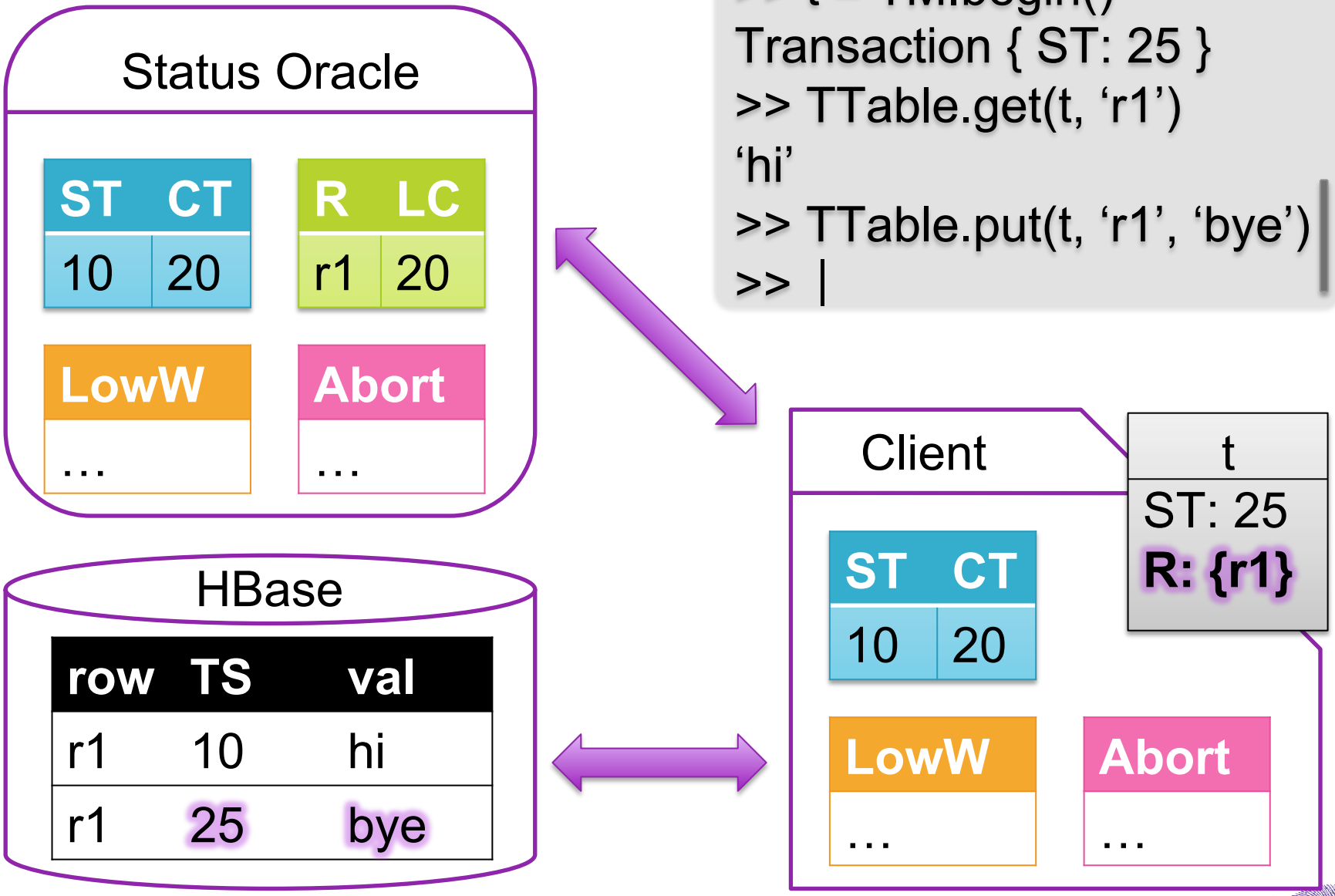
Transaction Example

```
>> t = TM.begin()
Transaction { ST: 25 }
>> TTable.get(t, 'r1')
'hi'
>>
```



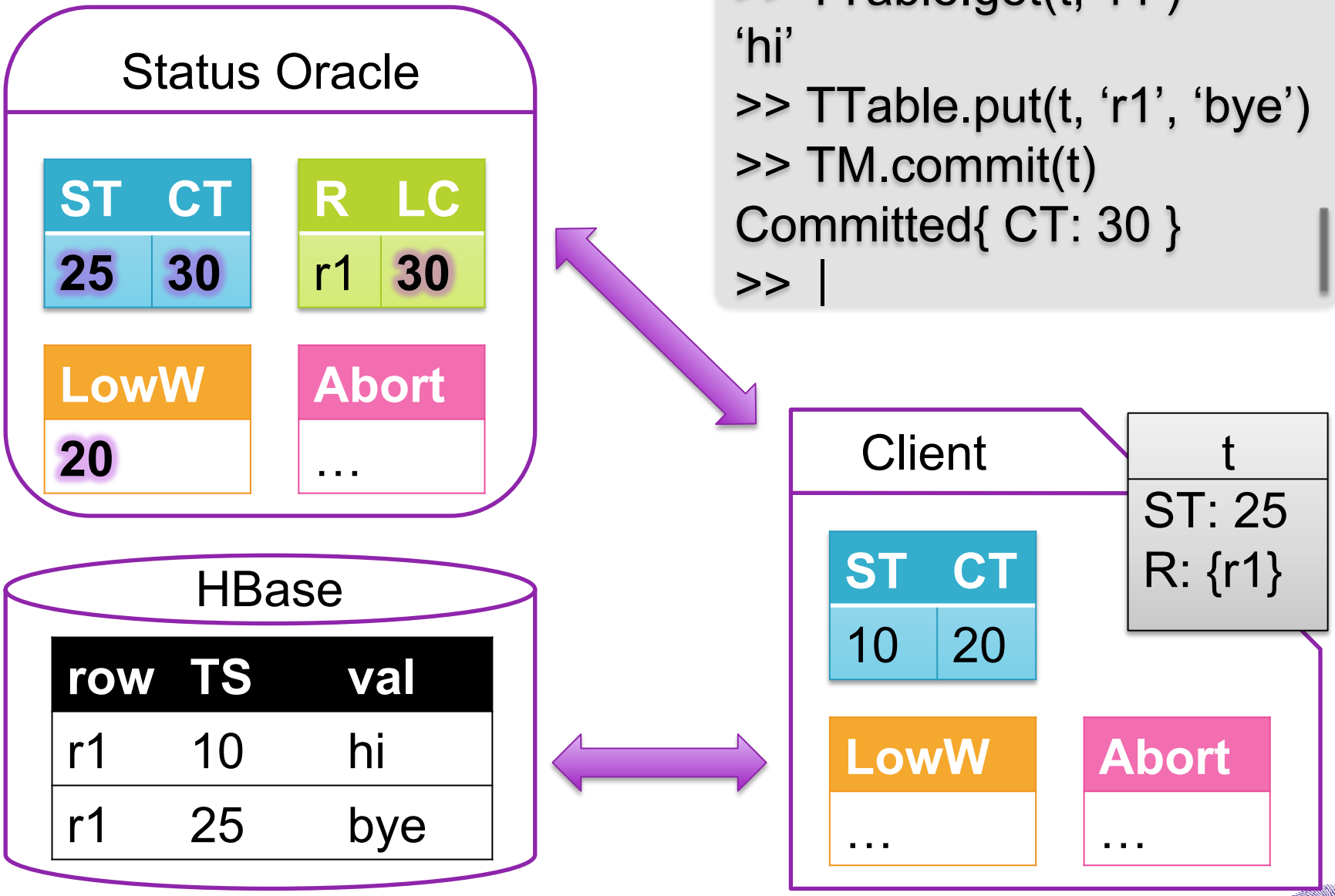
Transaction Example

```
>> t = TM.begin()
Transaction { ST: 25 }
>> TTable.get(t, 'r1')
'hi'
>> TTable.put(t, 'r1', 'bye')
>> |
```

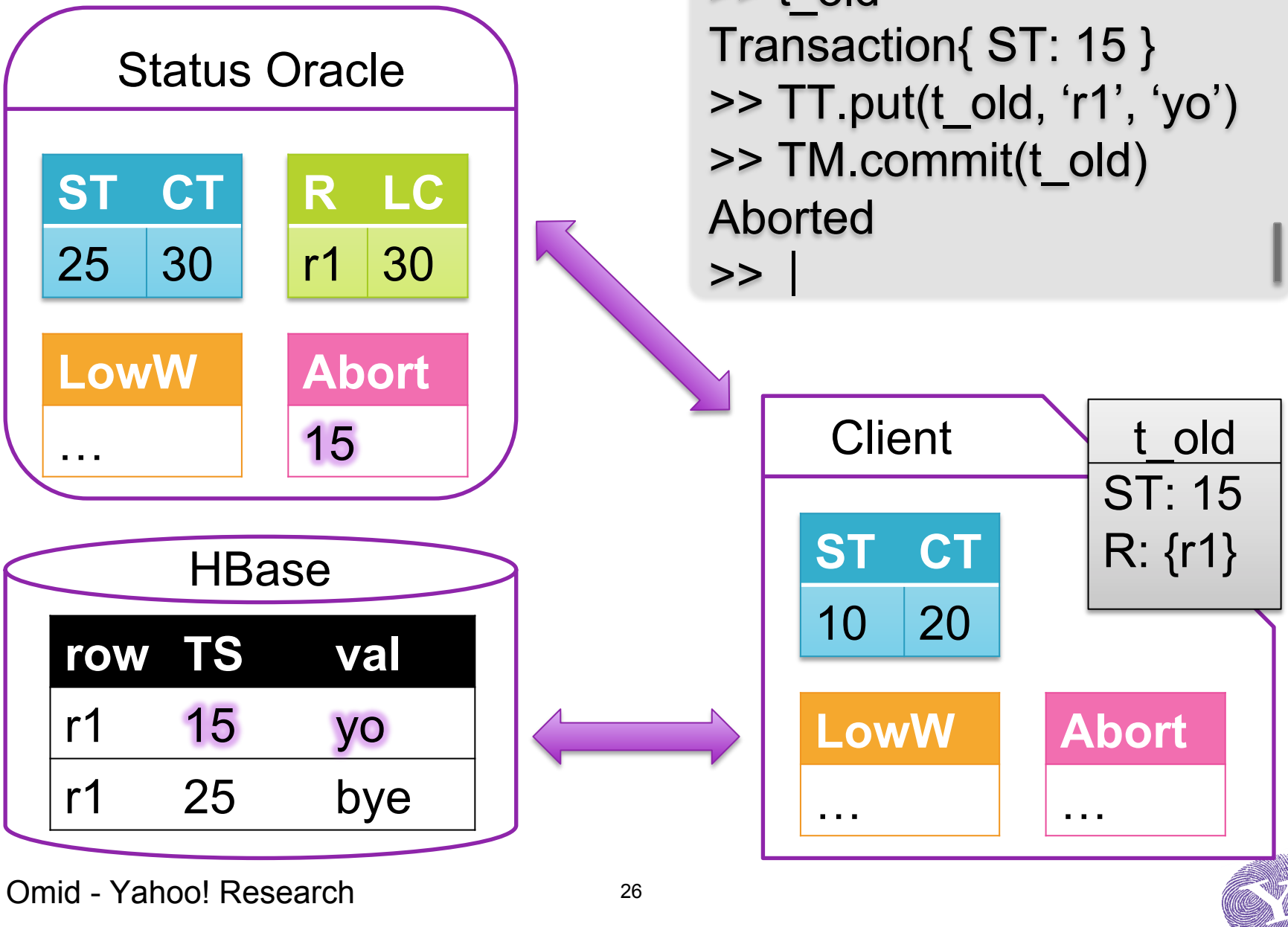


Transaction Example

```
>> TTable.get(t, 'r1')
'hi'
>> TTable.put(t, 'r1', 'bye')
>> TM.commit(t)
Committed{ CT: 30 }
>>
```



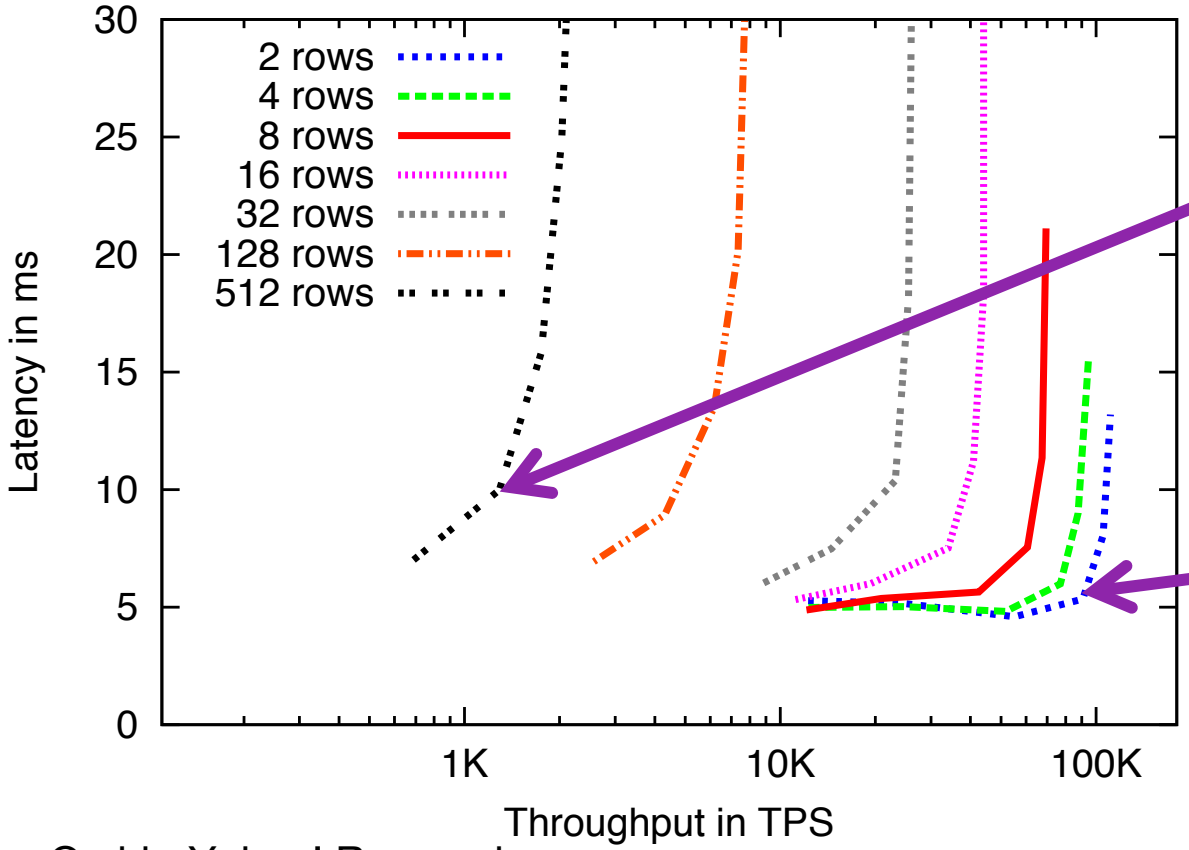
Transaction Example



Centralized Approach

Omid Performance

- Centralized server = bottleneck?
  - Focus on good performance
  - In our experiments, it never became the bottleneck

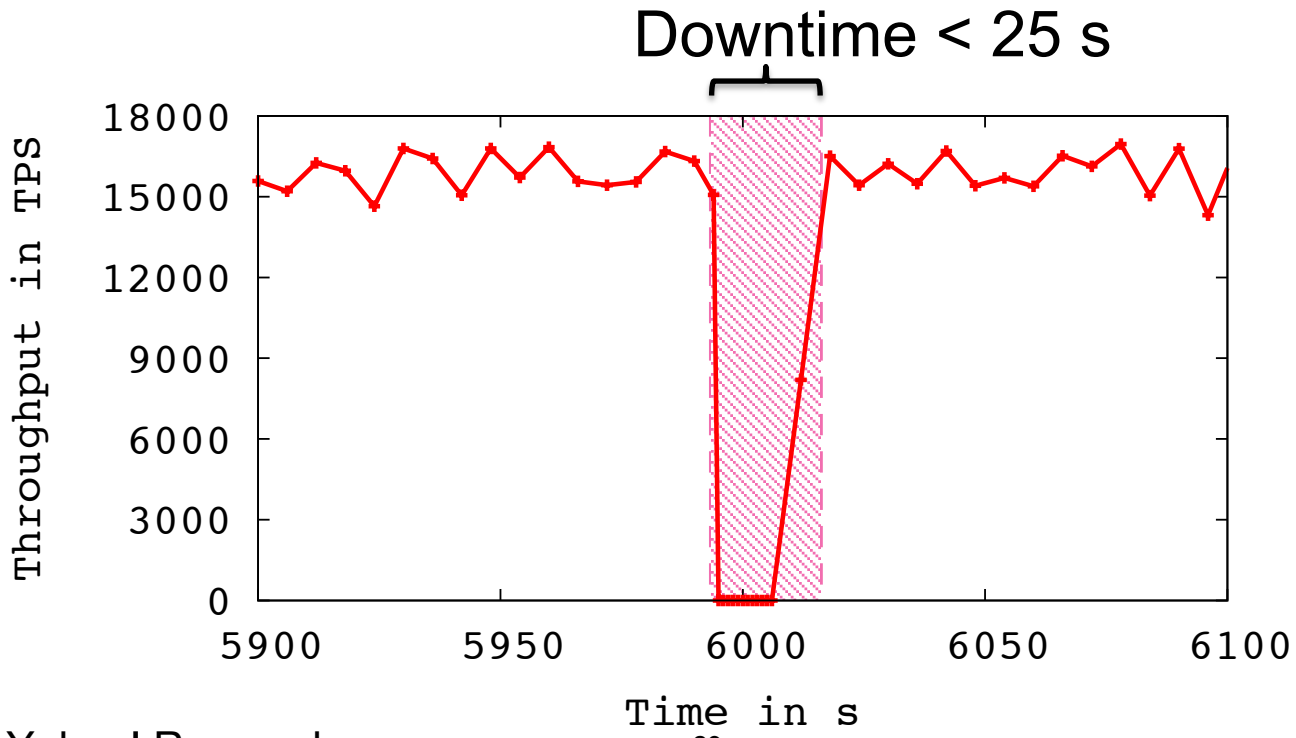


512 rows/t
1K TPS
10 ms

2 rows/t
100K TPS
5 ms

Fault Tolerance

- Omid uses BookKeeper for fault tolerance
  - BookKeeper is a distributed Write Ahead Log
- Recovery: reads log's tail (bounded time)



Example Use Case

News Recommendation System

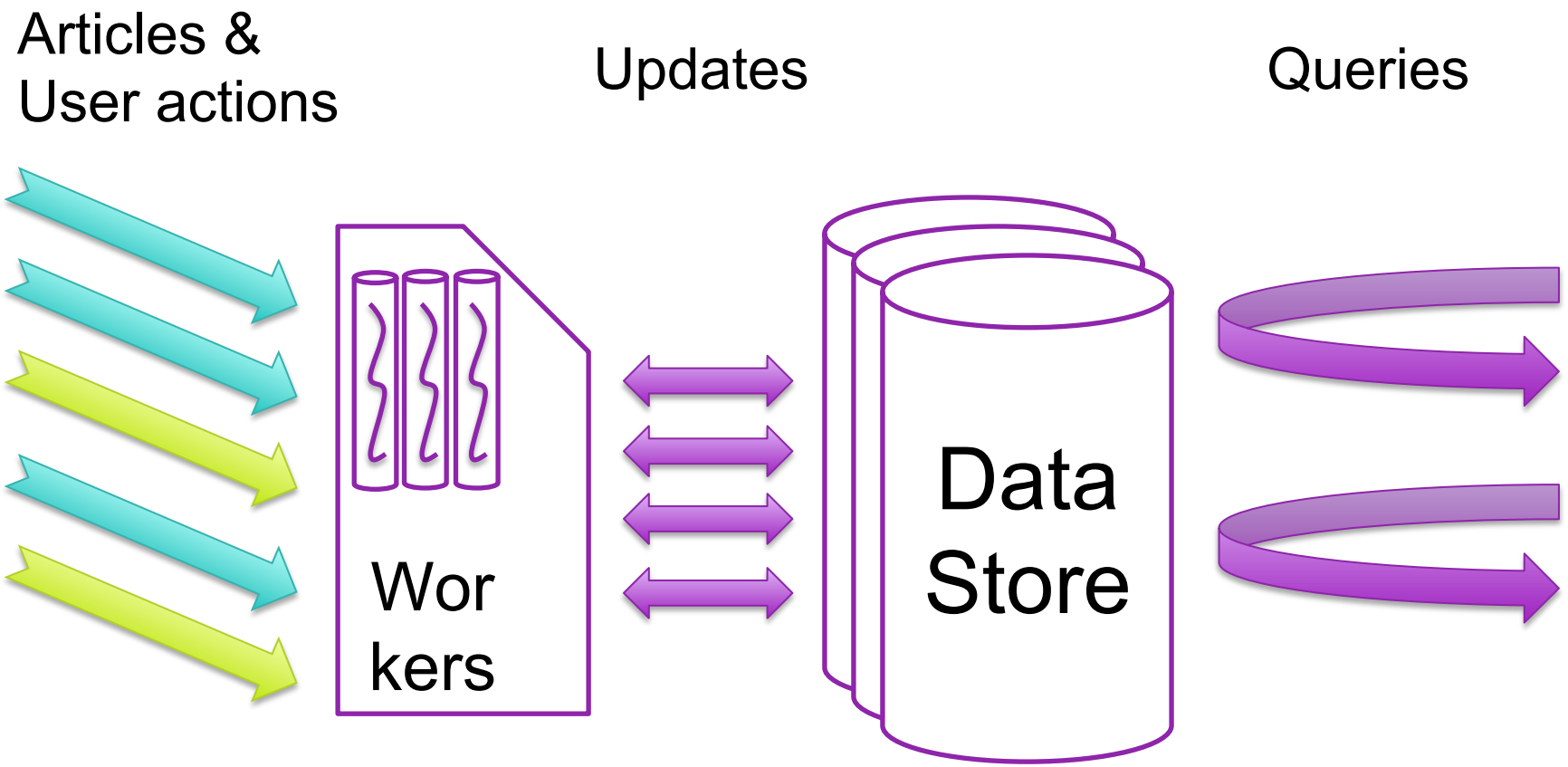
News Recommendation System

- Model-based recommendations
  - Similar users belong to a cluster
  - Each cluster has a trained model
- New article
  - Evaluate against clusters
  - Recommend to users
- User actions
  - Train model
  - Split cluster

Transactions Advantages

- All operations are **consistent**
  - Updates and queries
- Needed for a recommendation system?
  - We **avoid corner cases**
  - Implementing existing algorithms is **straightforward**
- Problems we solve
  - Two operations concurrently splitting a cluster
  - Query while cluster splits
  - …

Example overview



Other use cases

- Update indexes incrementally
  - Original Percolator use case
- Generic graph processing
- Adapt existing transactional solutions to HBase

Future Work

Current Challenges

- Load isn't distributed automatically
- Complex logic requires big transactions
  - More likely to conflict
  - More expensive to retry
- API is low level for data processing

Future work

- **Framework** for Incremental Processing
  - Simpler API
  - Trigger-based, easy to decompose operations
  - Auto scalable
- Improve user friendliness
  - Debug tools, metrics, etc
- Hot failover

Questions?

All time contributors

Ben Reed

Flavio Junqueira

Francisco Pérez-Sorrosal

Ivan Kelly

Matthieu Morel

Maysam Yabandeh

Partially supported by

CumuloNimbo project (ICT-257993)

funded by the European Commission

Try it!

yahoo.github.com/omid

Backup

Commit Algorithm

- Receive commit request for txn_i
- Set CommitTimestamp(txn_i) <= new timestamp
- For each modified row:
  - if LastCommit(row) > StartTimestamp(txn_i) => **abort**
- For each modified row:
  - Set LastCommit(row) <= CommitTimestamp(txn_i)