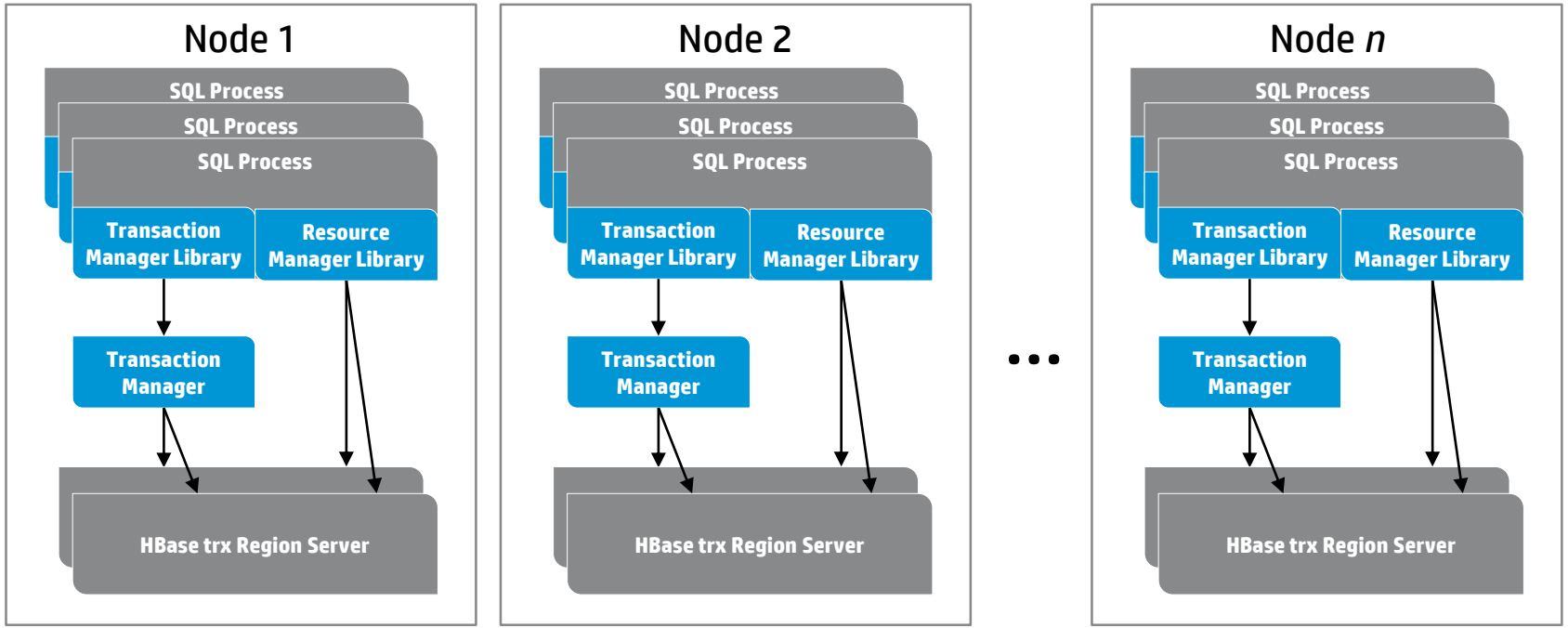
Trafodion Distributed Transaction Management

June 2014

Trafodion Distributed Transaction Management

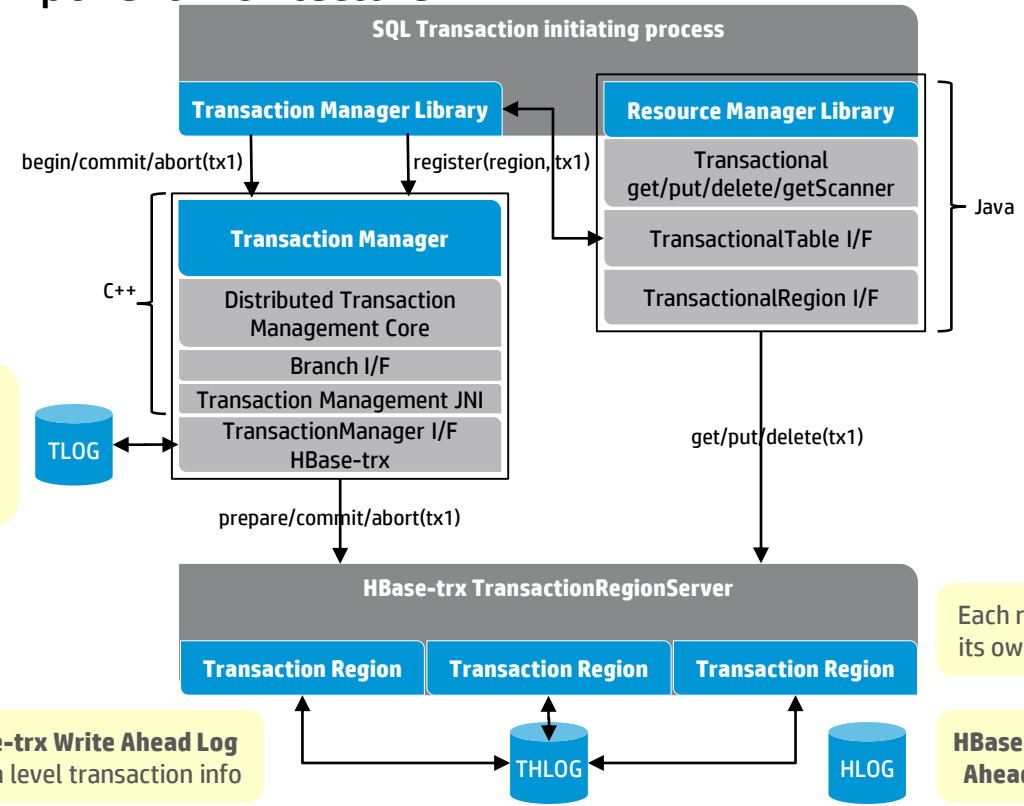
Scalable Architecture



HBase trx extends HBase classes

Trafodion Distributed Transaction Management

Peeling the onion – Component Architecture



Transaction Manager Log
Meta info/state about transaction
(more than one region in a txn)
HBase table(s)

Each region server has its own THLOG & HLOG

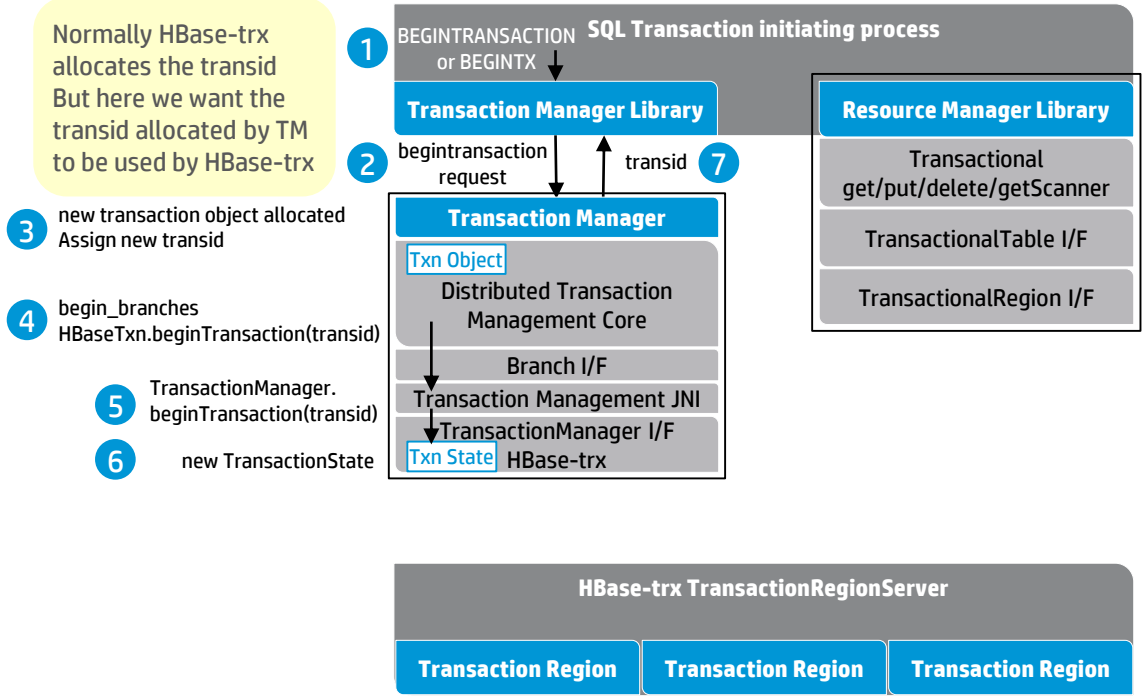
HBase-trx Write Ahead Log
Region level transaction info

HBase Write Ahead Log

Trafodion Distributed Transaction Management

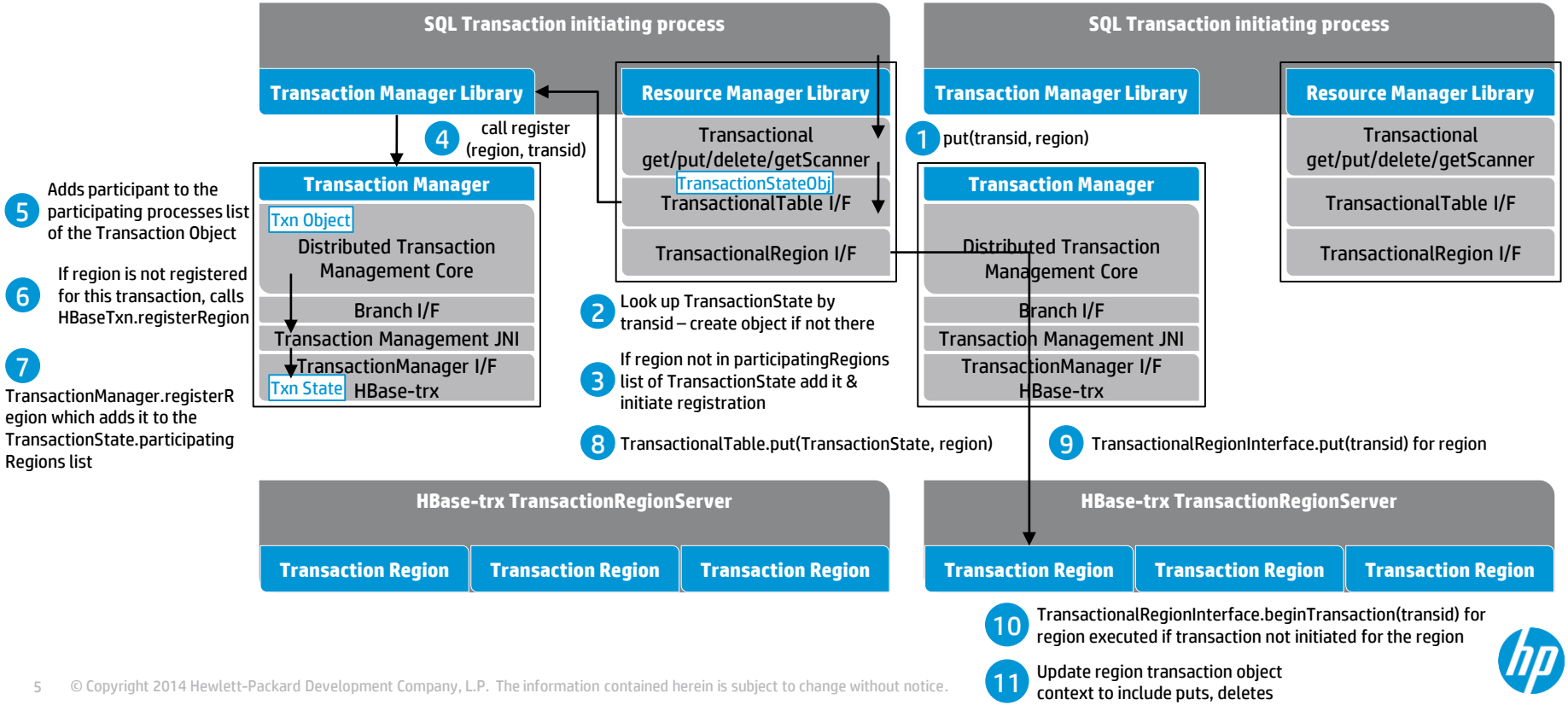
BEGIN WORK – BEGINTRANSACTION

Normally HBase-trx allocates the transid But here we want the transid allocated by TM to be used by HBase-trx



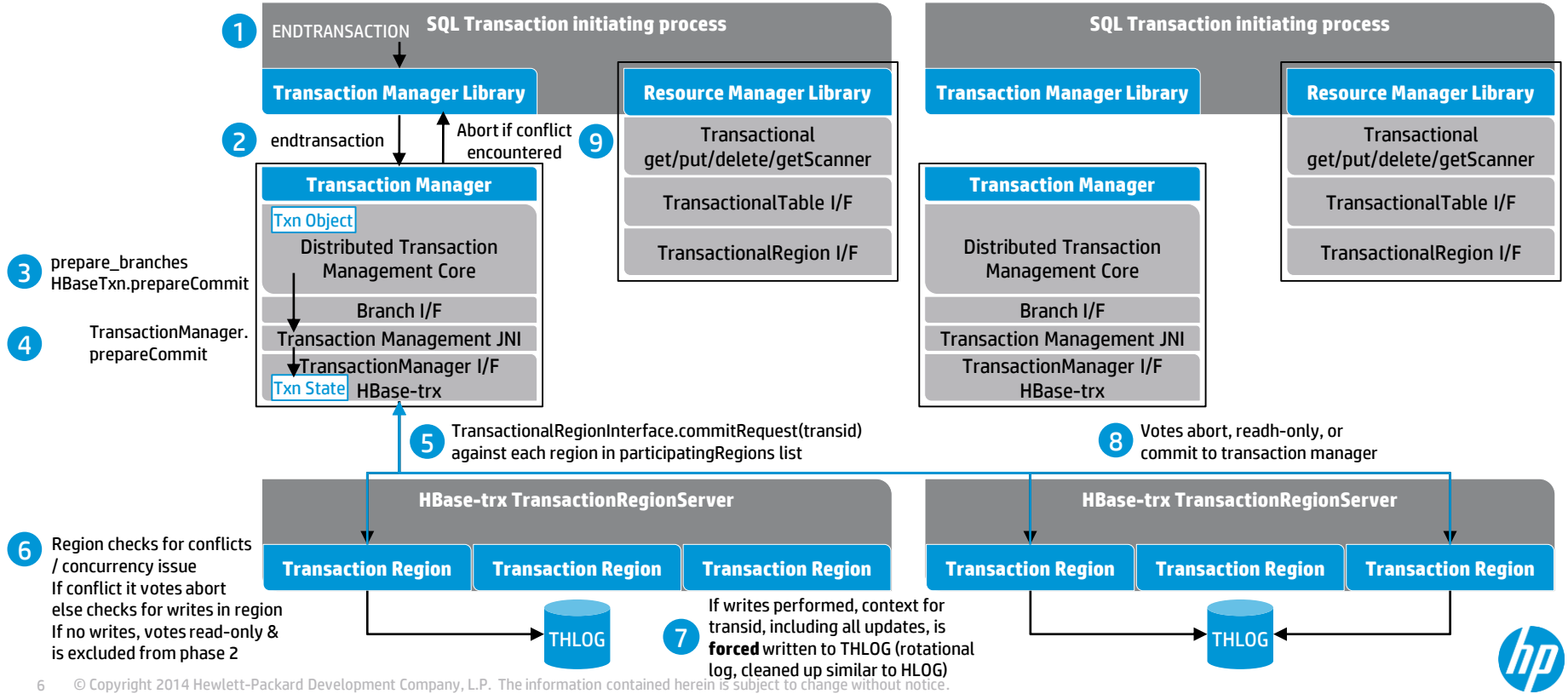
Trafodion Distributed Transaction Management

get / put / delete



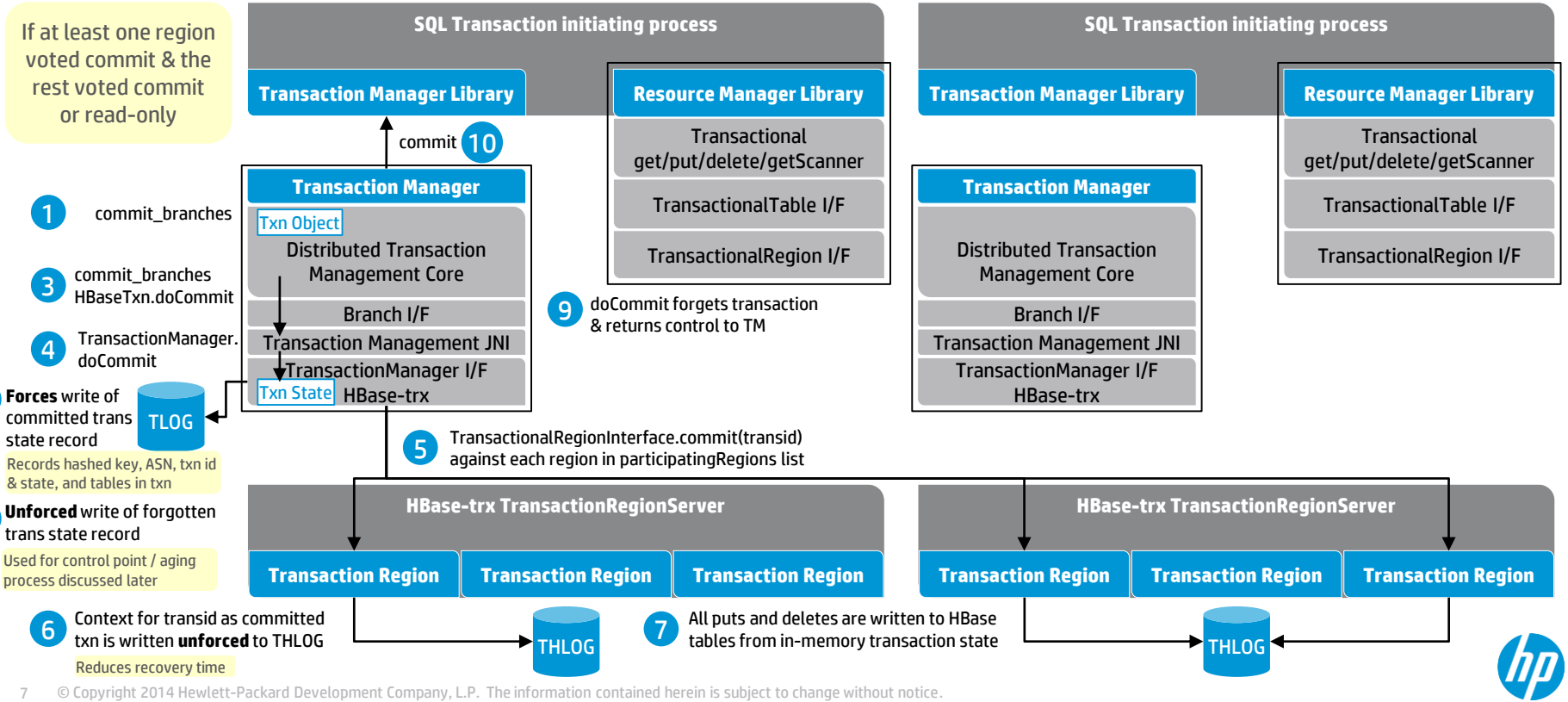
Trafodion Distributed Transaction Management

COMMIT WORK – ENDTRANSACTION – Phase 1, prepare



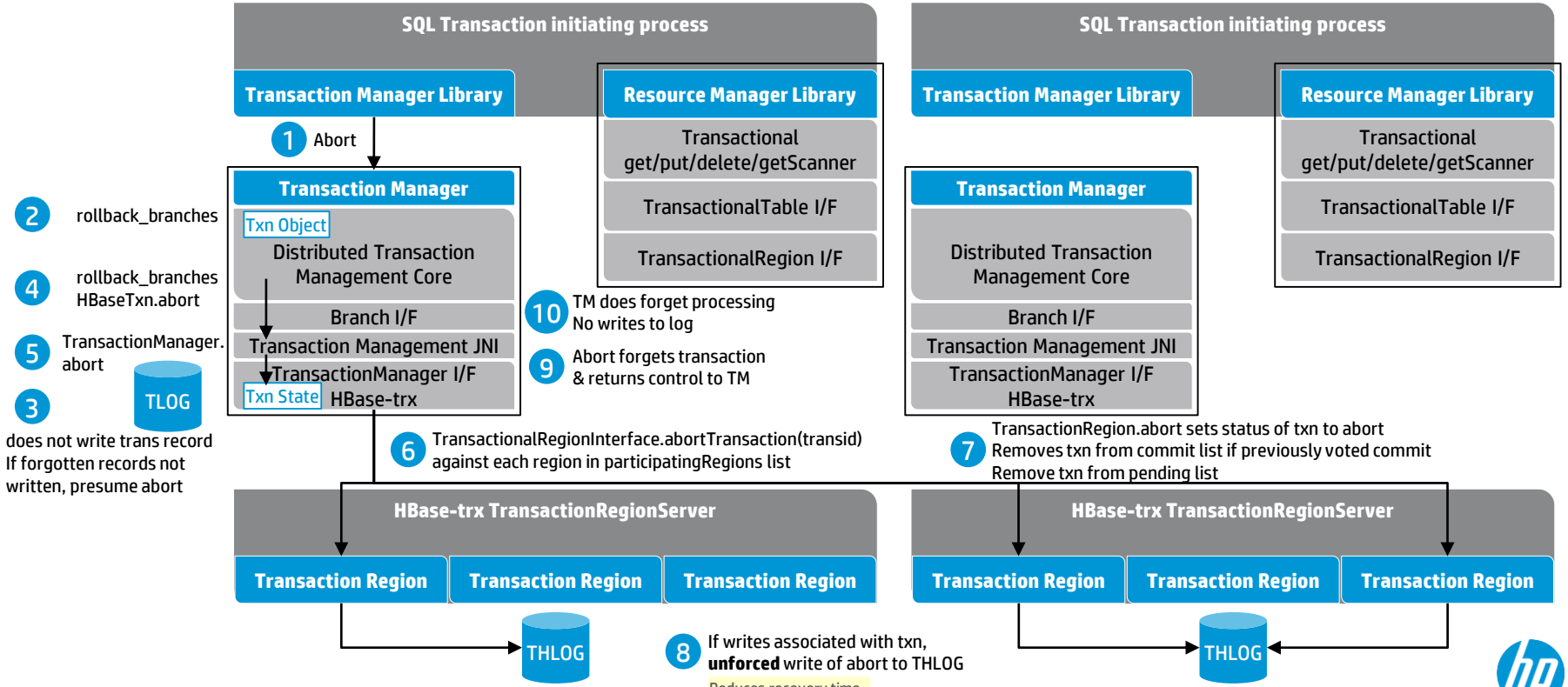
Trafodion Distributed Transaction Management

COMMIT WORK – ENDTRANSACTION – Phase 2, commit



Trafodion Distributed Transaction Management

ABORT WORK – ABORTTRANSACTION



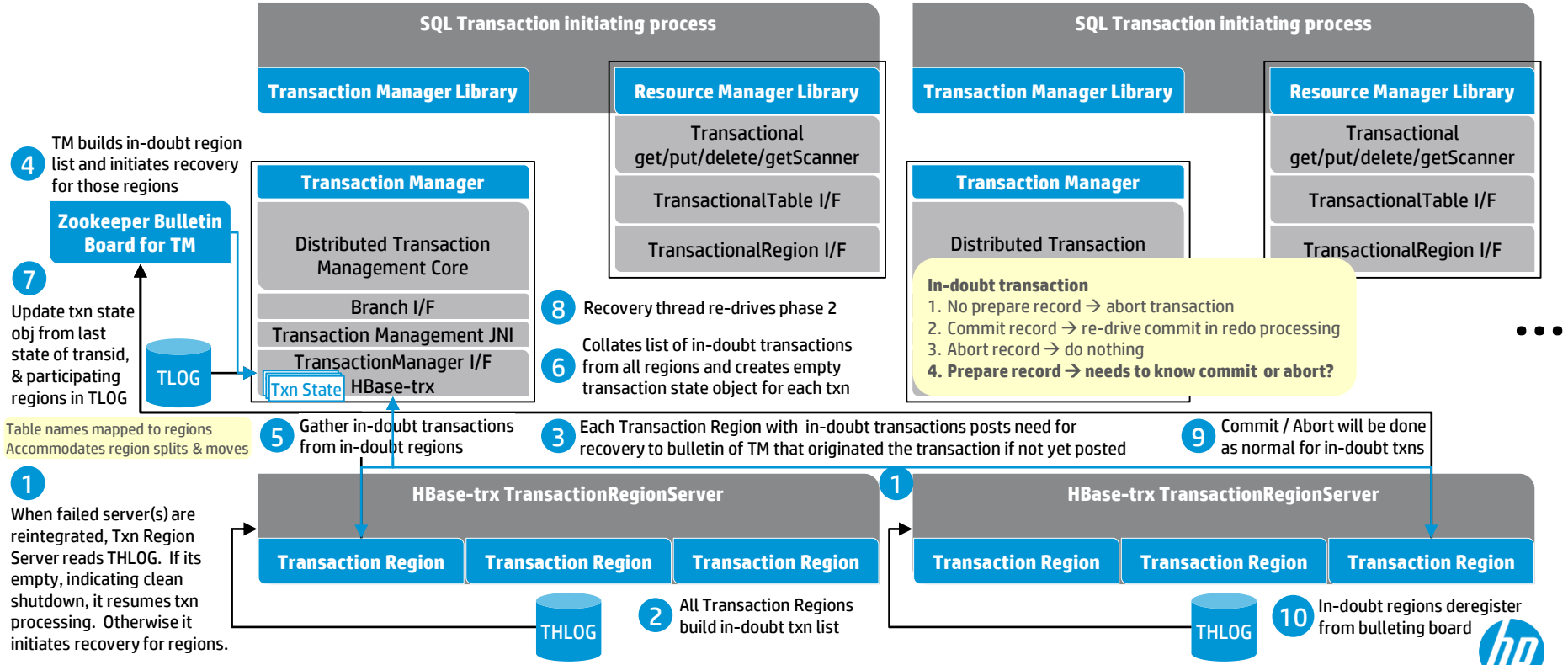
Trafodion Distributed Transaction Management

Transaction Recovery – timing of failure

State at failure	Action at failure	Failed region(s) start up
Updates are in region's memory	Initiate abort processing	No recovery needed
Prepare: Some regions complete	Initiate abort processing	If region had not completed Prepare phase before failure, no recovery needed. If it had, then abort processing initiated.
Prepare: All regions complete	Initiate abort processing	Abort processing for all regions that had updates
Commit: Some regions complete	Initiate commit processing	Apply updates from THLOG to tables if not committed

Trafodion Distributed Transaction Management

Transaction Recovery



TM disables transactions during recovery

Trafodion Distributed Transaction Management

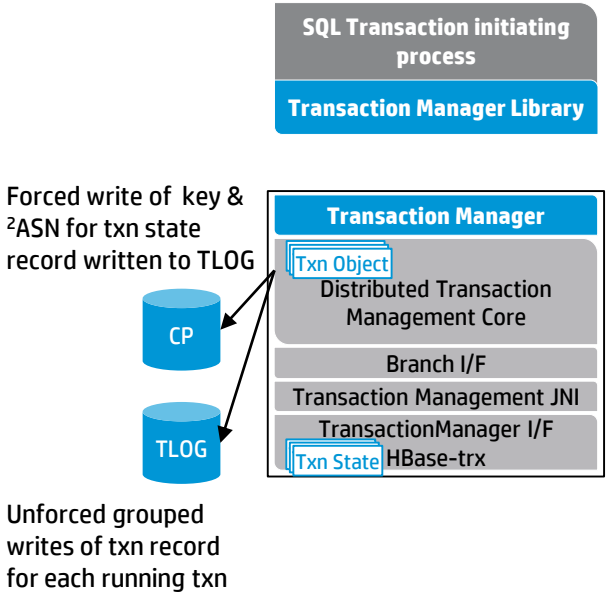
Transaction Recovery – Control Point processing

Used to re-drive transaction recovery

- After server or cluster failure to ensure data consistency given in-flight operations at time of failure
- Control points written to Control Point table at a configurable [1]interval, currently set at 2 minutes
- Re-drives of transaction commits (redo) driven by recovery needs of regions

Used for aging

- TLOG entries prior to 2 control points can be aged out and only two entries maintained in control point table
- Commit records rewritten to log, along with writing txn state of all running transactions, until all commits received and forgotten record written



[1]control point interval is represented by two consecutive control point records where the first record defines the ASN from the start of the interval and the second defines the ASN at the end of the record

[2]monotonically increasing value indicating the audit write sequence number

Trafodion Distributed Transaction Management

Addressing the requirements

Current State

- HBase-trx uses some derived classes for RegionServer, Region and HLogSplitter. These use HBase-internal interfaces.
- Trafodion has Transaction Manager (TM) processes, written in C++, supporting:
  - Recovery on client / region server crash
  - Transactions involving multiple HBase clients (transaction propagation)
- Clients can use a new, transactional client derived from regular HBase client
- HBase-trx as single jar file, used by region server, TM & clients. Compatible with most HBase 0.94.x versions.

Goals

- Optional, no penalty if not used
- Very low overhead – tight integration with the region server helps
- Avoid additional processes
- Avoid non-Java code
- Avoid version incompatibilities

How to achieve goals

- Implement as coprocessors
- Provide Java implementation of TM code for the following functionality, if needed:
  - Txn propagation
  - Manageability
  - Recovery

Trafodion Distributed Transaction Management

Future opportunities

Optimizations

For single region transactions

Transaction flow

Combining THLOG with HLOG

Isolation support

Repeatable read

Snapshot isolation

Serialized snapshot isolation

Row Locking paradigm for certain tables

Tables involved in long running transactions

Pessimistic locking for highly concurrent operations

High Availability

Recovery from region server, TM, or node failure
Accommodate Region Splitting & Balancing

Manageability

Transaction object info/metrics via REST APIs

Thank you