

Moving Apache Ignite into Production: Best Practices for Distributed Transactions

Ivan Rakov
June 10, 2020



June 10, 2020



Ivan Rakov

- Leader of the data-consistency development team, GridGain Systems
- Apache® Ignite™ Committer

Ignite transactions subsystem



cacheConfiguration.setCacheAtomicityMode(TRANSACTIONAL);

- Cross-partition
- Cross-cache
- Multi-record
- Multi-node
- Full ACID guarantees
- Failover-safe (still ACID if some of participant nodes fail)

Agenda



- API overview
- Isolation and concurrency
- What happens under the hood
 - Transaction flow and commit protocol
 - TX recovery
- Tips for moving safely into production
 - Recommendations to avoid delays
 - Troubleshooting

Agenda



- API overview
- Isolation and concurrency
- What happens under the hood
 - Transaction flow and commit protocol
 - TX recovery
- Tips for moving safely into production
 - Recommendations to avoid delays
 - Troubleshooting

Entry point to transactions API



ignite.transactions();

Main facade for transactions API

Transaction start



transactions.txStart(concurrency, isolation, timeout, txSize);

- concurrency = PESSIMISTIC, OPTIMISTIC
- isolation = READ_COMMITTED, REPEATABLE_READ, SERIALIZABLE
- timeout = transaction to be rolled back after the specified period
- txSize = expected number of affected entries
- The transaction is started and is attached to the current thread.
- All cache operations that are executed in the current thread are within the transaction.

Transaction start



transactions.txStart(concurrency, isolation, timeout, txSize);

```
try (Transaction tx = transactions.txStart()) {  
    cache1.put(k, v);  
    cache2.remove(k);  
    // ^ two operations within the same transaction  
}
```


Current transaction retrieval



transactions.tx();

Returns the transaction instance that is attached to the current thread

```
try (Transaction tx = transactions.txStart()) {  
    foo();  
}  
  
public void foo() {  
    Transaction tx = transactions.tx();  
    tx.setRollbackOnly();  
}
```

Transaction commit



tx.commit();

- The transaction commit is performed.
- Data is not changed, unless commit is called.

```
try (Transaction tx = transactions.txStart()) {  
    cache.put(k1, v1);  
    cache.put(k2, v2);  
  
    tx.commit();  
}
```

Transaction close



tx.close();

- Transaction detached from the current thread
- Transaction rolled back, if it wasn't committed

```
try (Transaction tx = transactions.txStart()) {  
    cache.put(k1, v1);  
    cache.put(k2, v2);  
}  
// Transaction is AutoCloseable:  
// close is called, transaction gets rolled back
```

Transaction rollback



tx.setRollbackOnly();

- Makes rollback the only possible outcome of transaction
- Affects state, if called before commit();

tx.rollback();

Performs greedy and synchronous transaction rollback

Implicit transactions



tx.implicit();

- Returns true, if the transaction was started implicitly—without txStart();
- cache.putAll(map);
 - Starts implicit transactions

Suspending and resuming transactions




tx.suspend();

Detaches the transaction from the current thread

tx.resume();

Attaches the suspended transaction to the current thread

```
Thread 1:  
tx = transactions.txStart();  
tx.suspend();
```



```
Thread 2:  
tx.resume();  
tx.commit();
```

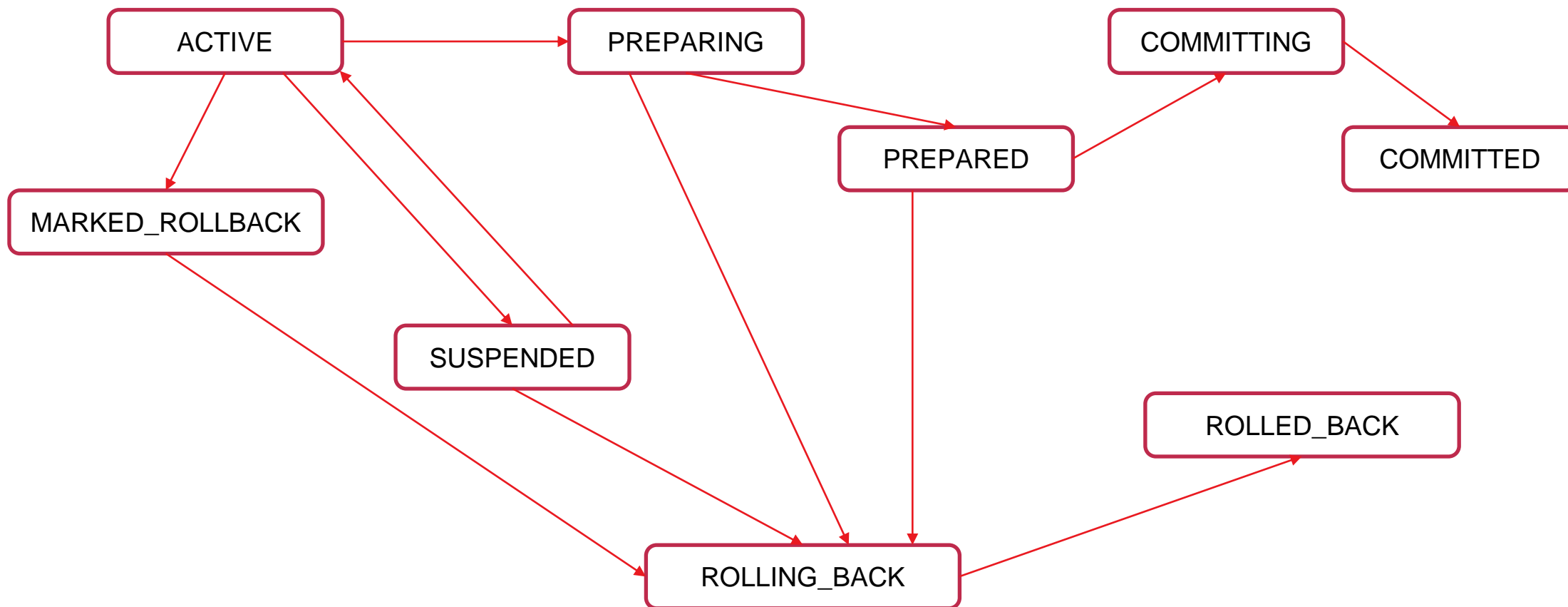
Transaction states



tx.state();

Transaction state	Comment
ACTIVE	transactional activity in progress
SUSPENDED	transaction activity paused, but can be resumed from any thread
MARKED_ROLLBACK	transactional activity in progress, but commit isn't possible
PREPARING	commit preparing
PREPARED	commit prepared
COMMITTING	commit in progress, data storage being updated
COMMITTED	commit finished
ROLLING_BACK	rollback in progress
ROLLED_BACK	rollback finished

Transaction state flow



Exceptions to check



- TransactionRollbackException—on automatic rollback
- TransactionTimeoutException—on timeout
- TransactionDeadlockException—on key-level deadlock
- TransactionOptimisticException—on optimistic lock failure
- ClusterTopologyException—on primary node fail

Agenda



- API overview
- Isolation and concurrency
- What happens under the hood
 - Transaction flow and commit protocol
 - TX recovery
- Tips for moving safely into production
 - Recommendations to avoid delays
 - Troubleshooting

OPTIMISTIC, READ_COMMITTED



Use case 1: atomic batch update

```
try (Transaction tx = txs.txStart(OPTIMISTIC, READ_COMMITTED)) {  
    txCache.putAll(batch);  
    tx.commit();  
}  
  
// equal actions  
  
txCache.putAll(batch);
```

PESSIMISTIC, READ_COMMITTED



Use case 2: exclusive update

```
try (Transaction tx = txs.txStart(PESSIMISTIC, READ_COMMITTED)) {  
    lastUserExecutionCache.put(user1, UUID.randomUUID());  
    // all similar operations with user1 are locked  
  
    transfersCache.put(newTransferId, transfer);  
    auditCache.put(newAuditId, audit);  
  
    tx.commit();  
}
```

PESSIMISTIC, REPEATABLE_READ



Use case 3: safe payment processing

```
try (Transaction tx = txs.txStart(PESSIMISTIC, REPEATABLE_READ)) {  
    int balance1 = cache.get(acc1);  
    int balance2 = cache.get(acc2);  
  
    cache.put(acc1, balance1 - 100);  
    cache.put(acc2, balance2 + 100);  
  
    tx.commit();  
}
```

Snapshot isolation



Snapshot isolation is a guarantee that all reads made in a transaction will see a consistent snapshot of the database.

- Unavailable in Ignite transactions
- Supported in MVCC mode (since 2.7, beta)
- Capable of being emulated

OPTIMISTIC, SERIALIZABLE



Use case 4: emulation of snapshot isolation

```
try (Transaction tx = txs.txStart(OPTIMISTIC, SERIALIZABLE)) {
    Map<UserId, User> map = cache.getAll(usersFromCity);
    // Optimistic shared locks are acquired for all users

    boolean noAmbassador = !map.values().stream().anyMatch(User::isAmbassador);

    if (noAmbassador) {
        User u = map.get(newAmbassadorId);
        u.setStatus(CITY_AMBASSADOR);
        cache.put(userId, u);
    }

    tx.commit();
    // Guarantees that only one user from city can be promoted to ambassador
}
```

Transactional caches



	OPTIMISTIC / READ_COMMITTED*	PESSIMISTIC / READ_COMMITTED	PESSIMISTIC / REPEATABLE_READ	OPTIMISTIC / SERIALIZABLE
ACID guarantees	✓	✓	✓	✓
Locks on write (exclusive update possible)	x	✓	✓	✓ (optimistic lock)
Locks on read (payment processing possible)	x	x	✓	✓ (optimistic lock)
Forced rollback on concurrent update is possible	x	x	x	✓ (if optimistic locking fails)
Automatic resolution of deadlocks that are caused by the application	x	x	x	✓

* Batch putAll also has OPTIMISTIC / READ_COMMITTED guarantees

Agenda



- API overview
- Isolation and concurrency
- What happens under the hood
 - Transaction flow and commit protocol
 - TX recovery
- Tips for moving safely into production
 - Recommendations to avoid delays
 - Troubleshooting

Transaction flow and commit protocol



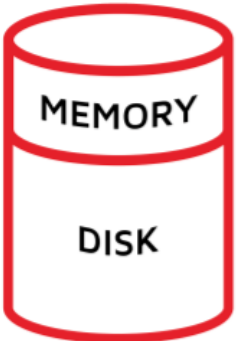
```
try (Transaction tx = txStart()) {
```



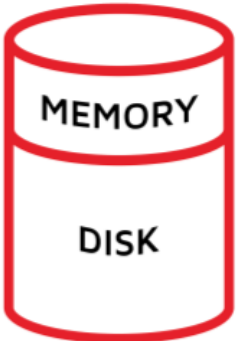
CLIENT

TX state	ACTIVE
Enlisted entries	
Locked keys	
Written WAL	

TX state	?
Enlisted entries	
Locked keys	
Written WAL	

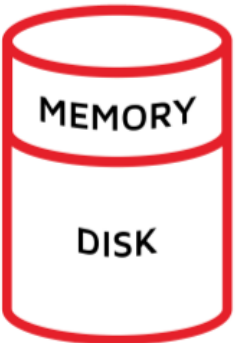


 NODE
primary



 NODE
primary

TX state	?
Enlisted entries	
Locked keys	
Written WAL	



 NODE
backup

TX state	?
Enlisted entries	
Locked keys	
Written WAL	

TX state	?
Enlisted entries	
Locked keys	
Written WAL	



 NODE
backup

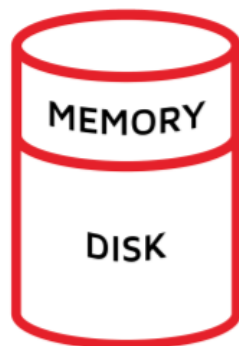
```
try (Transaction tx = txStart()) {  
    cache.put(k1, v1);
```



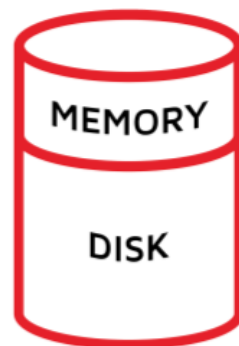
CLIENT

TX state	ACTIVE
Enlisted entries	(k1, v1)
Locked keys	
Written WAL	

TX state	?
Enlisted entries	
Locked keys	
Written WAL	

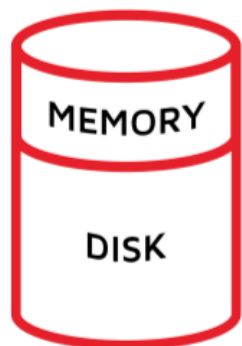


 NODE
primary



 NODE
primary

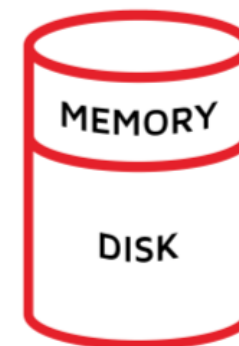
TX state	?
Enlisted entries	
Locked keys	
Written WAL	



 NODE
backup

TX state	?
Enlisted entries	
Locked keys	
Written WAL	

TX state	?
Enlisted entries	
Locked keys	
Written WAL	



 NODE
backup

```
try (Transaction tx = txStart()) {
    cache.put(k1, v1);
```

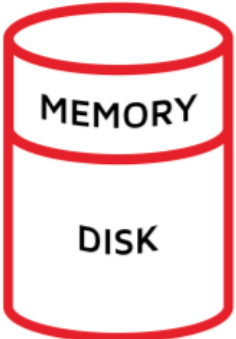


CLIENT

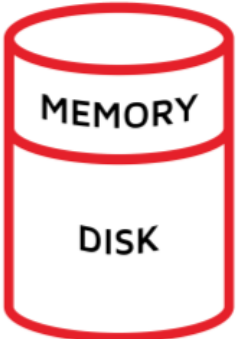
lock request

TX state	ACTIVE
Enlisted entries	(k1, v1)
Locked keys	
Written WAL	

TX state	ACTIVE
Enlisted entries	
Locked keys	
Written WAL	

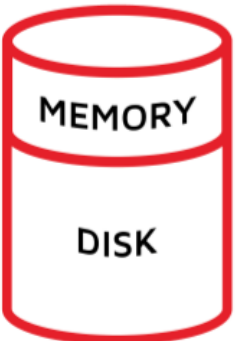


 NODE
primary



 NODE
primary

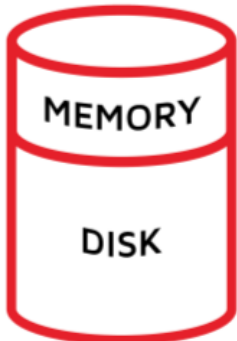
TX state	?
Enlisted entries	
Locked keys	
Written WAL	



 NODE
backup

TX state	?
Enlisted entries	
Locked keys	
Written WAL	

TX state	?
Enlisted entries	
Locked keys	
Written WAL	



 NODE
backup

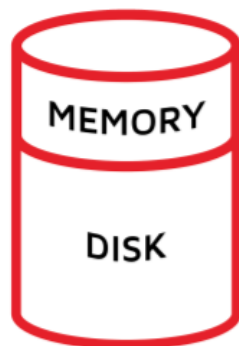
```
try (Transaction tx = txStart()) {
    cache.put(k1, v1);
```



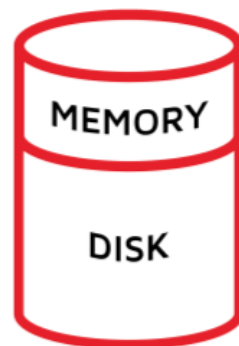
CLIENT

TX state	ACTIVE
Enlisted entries	(k1, v1)
Locked keys	
Written WAL	

TX state	ACTIVE
Enlisted entries	
Locked keys	k1
Written WAL	

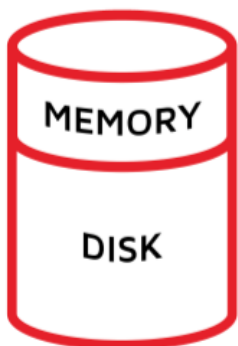


 NODE
primary



 NODE
primary

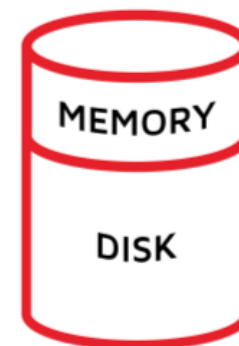
TX state	?
Enlisted entries	
Locked keys	
Written WAL	



 NODE
backup

TX state	?
Enlisted entries	
Locked keys	
Written WAL	

TX state	?
Enlisted entries	
Locked keys	
Written WAL	



 NODE
backup

```
try (Transaction tx = txStart()) {  
    cache.put(k1, v1);
```

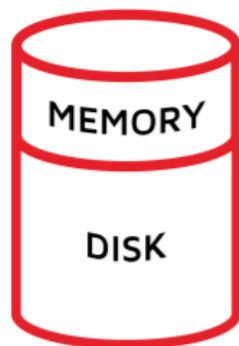


CLIENT

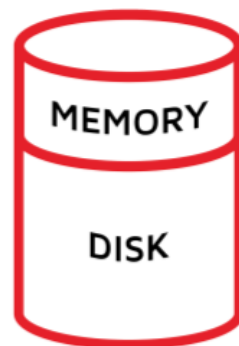
lock response

TX state	ACTIVE
Enlisted entries	(k1, v1)
Locked keys	
Written WAL	

TX state	ACTIVE
Enlisted entries	
Locked keys	k1
Written WAL	

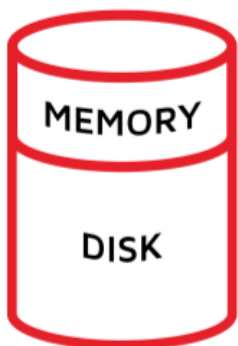


NODE
primary



NODE
primary

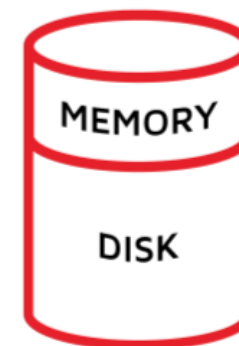
TX state	?
Enlisted entries	
Locked keys	
Written WAL	



NODE
backup

TX state	?
Enlisted entries	
Locked keys	
Written WAL	

TX state	?
Enlisted entries	
Locked keys	
Written WAL	



NODE
backup

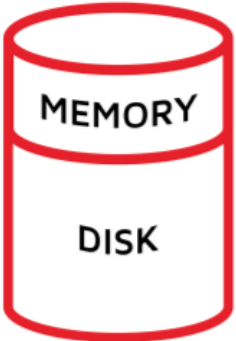
```
try (Transaction tx = txStart()) {  
    cache.put(k1, v1); ✓  
}
```



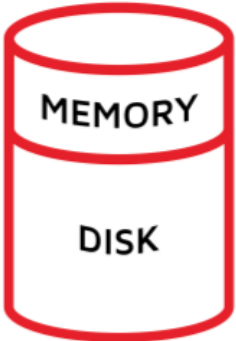
CLIENT

TX state	ACTIVE
Enlisted entries	(k1, v1)
Locked keys	
Written WAL	

TX state	ACTIVE
Enlisted entries	
Locked keys	k1
Written WAL	

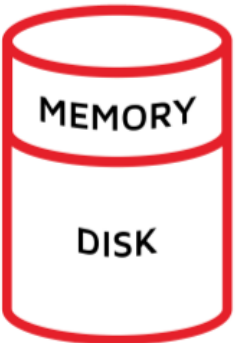


 NODE
primary



 NODE
primary

TX state	?
Enlisted entries	
Locked keys	
Written WAL	



 NODE
backup

TX state	?
Enlisted entries	
Locked keys	
Written WAL	

TX state	?
Enlisted entries	
Locked keys	
Written WAL	



 NODE
backup


```
try (Transaction tx = txStart()) {  
    cache.put(k1, v1); ✓
```

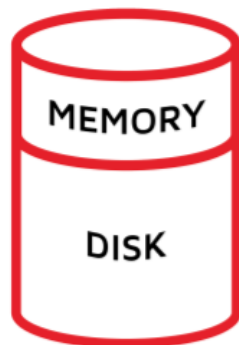
```
    cache.put(k2, v2); ⚙
```



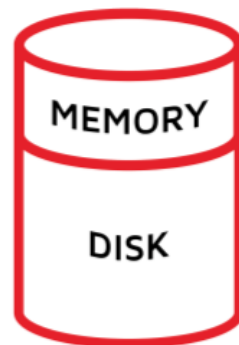
CLIENT

TX state	ACTIVE
Enlisted entries	(k1, v1), (k2, v2)
Locked keys	
Written WAL	

TX state	ACTIVE
Enlisted entries	
Locked keys	k1
Written WAL	

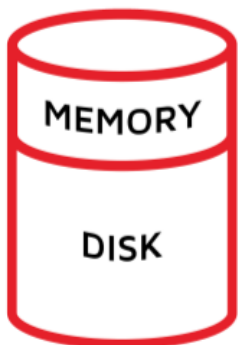


 NODE
primary



 NODE
primary

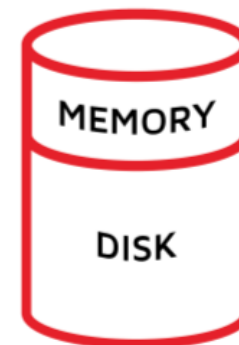
TX state	?
Enlisted entries	
Locked keys	
Written WAL	



 NODE
backup

TX state	?
Enlisted entries	
Locked keys	
Written WAL	

TX state	?
Enlisted entries	
Locked keys	
Written WAL	



 NODE
backup

```
try (Transaction tx = txStart()) {  
    cache.put(k1, v1); ✓
```

```
    cache.put(k2, v2); ⚙
```

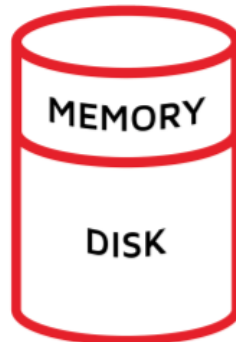


CLIENT

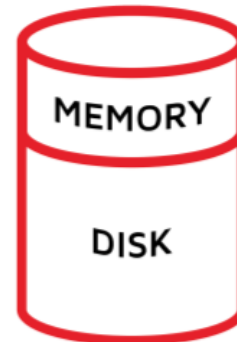
TX state	ACTIVE
Enlisted entries	(k1, v1), (k2, v2)
Locked keys	
Written WAL	

lock request

TX state	ACTIVE
Enlisted entries	
Locked keys	k1
Written WAL	

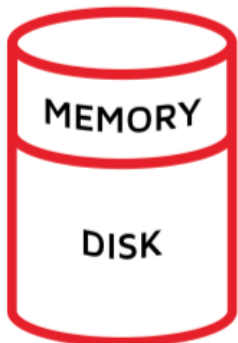


 NODE
primary



 NODE
primary

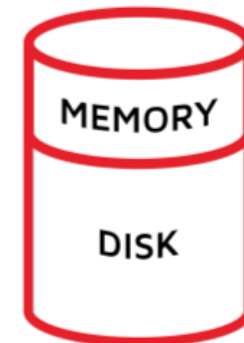
TX state	ACTIVE
Enlisted entries	
Locked keys	
Written WAL	



 NODE
backup

TX state	?
Enlisted entries	
Locked keys	
Written WAL	

TX state	?
Enlisted entries	
Locked keys	
Written WAL	



 NODE
backup

```
try (Transaction tx = txStart()) {  
    cache.put(k1, v1); ✓
```

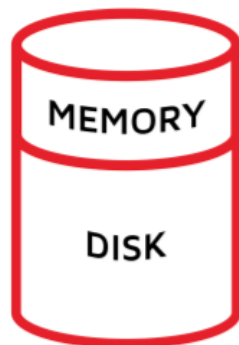
```
    cache.put(k2, v2); ⚙
```



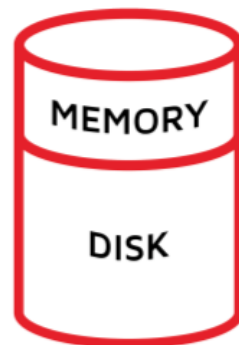
CLIENT

TX state	ACTIVE
Enlisted entries	(k1, v1), (k2, v2)
Locked keys	
Written WAL	

TX state	ACTIVE
Enlisted entries	
Locked keys	k1
Written WAL	

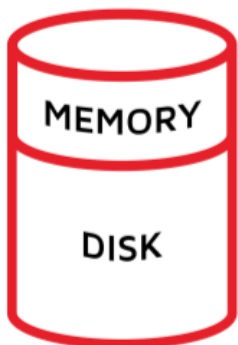


 NODE
primary



 NODE
primary

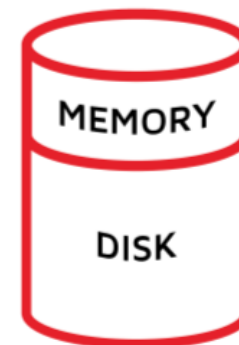
TX state	ACTIVE
Enlisted entries	
Locked keys	k2
Written WAL	



 NODE
backup

TX state	?
Enlisted entries	
Locked keys	
Written WAL	

TX state	?
Enlisted entries	
Locked keys	
Written WAL	



 NODE
backup

```
try (Transaction tx = txStart()) {  
    cache.put(k1, v1); ✓
```

```
    cache.put(k2, v2); ⚙
```

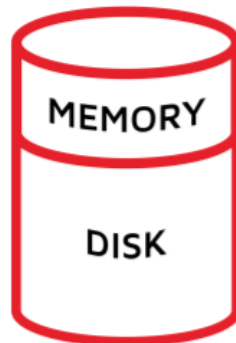


CLIENT

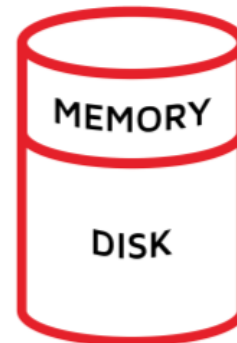
TX state	ACTIVE
Enlisted entries	(k1, v1), (k2, v2)
Locked keys	
Written WAL	

lock response

TX state	ACTIVE
Enlisted entries	
Locked keys	k1
Written WAL	

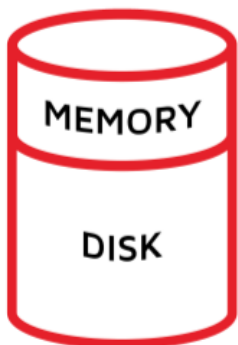


 NODE
primary



 NODE
primary

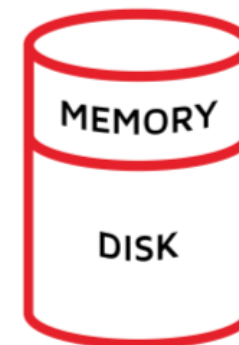
TX state	ACTIVE
Enlisted entries	
Locked keys	k2
Written WAL	



 NODE
backup

TX state	?
Enlisted entries	
Locked keys	
Written WAL	

TX state	?
Enlisted entries	
Locked keys	
Written WAL	



 NODE
backup

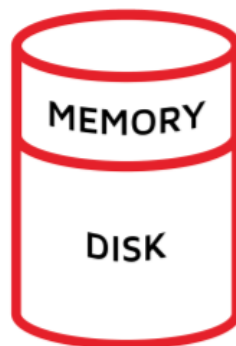
```
try (Transaction tx = txStart()) {
    cache.put(k1, v1); ✓

    cache.put(k2, v2); ✓
}
```

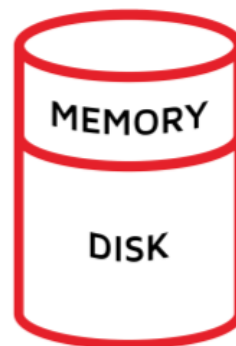


TX state	ACTIVE
Enlisted entries	(k1, v1), (k2, v2)
Locked keys	
Written WAL	

TX state	ACTIVE
Enlisted entries	
Locked keys	k1
Written WAL	

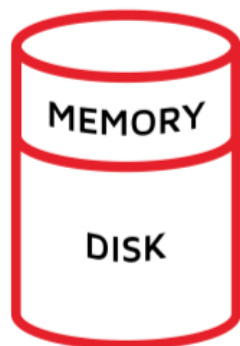


 NODE
primary



 NODE
primary

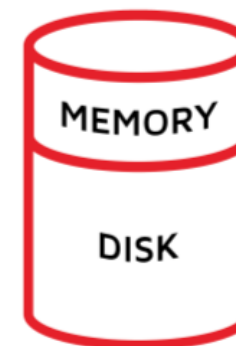
TX state	ACTIVE
Enlisted entries	
Locked keys	k2
Written WAL	



 NODE
backup

TX state	?
Enlisted entries	
Locked keys	
Written WAL	

TX state	?
Enlisted entries	
Locked keys	
Written WAL	



 NODE
backup

```
try (Transaction tx = txStart()) {
    cache.put(k1, v1); ✓

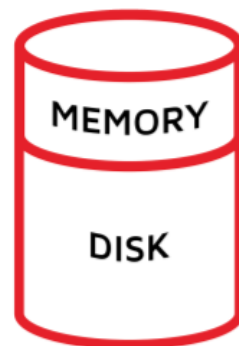
    cache.put(k2, v2); ✓

    tx.commit(); ⌛
}
```

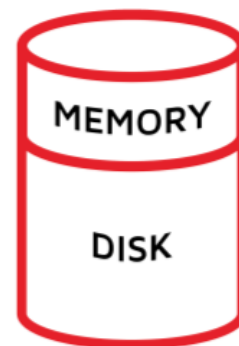


TX state	PREPARING
Enlisted entries	(k1, v1), (k2, v2)
Locked keys	
Written WAL	

TX state	ACTIVE
Enlisted entries	
Locked keys	k1
Written WAL	

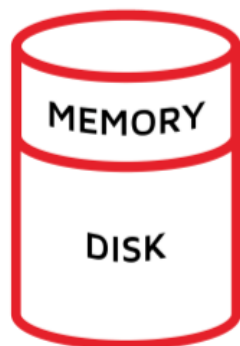


 NODE
primary



 NODE
primary

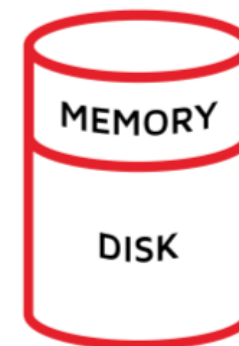
TX state	ACTIVE
Enlisted entries	
Locked keys	k2
Written WAL	



 NODE
backup

TX state	?
Enlisted entries	
Locked keys	
Written WAL	

TX state	?
Enlisted entries	
Locked keys	
Written WAL	



 NODE
backup

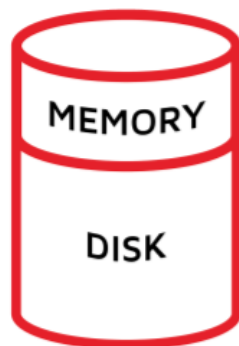
```
try (Transaction tx = txStart()) {
    cache.put(k1, v1); ✓
    cache.put(k2, v2); ✓
    tx.commit(); ⌛ prepare request
}
```



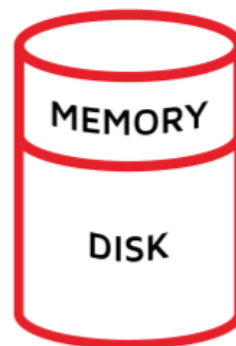
CLIENT

TX state	PREPARING
Enlisted entries	(k1, v1), (k2, v2)
Locked keys	
Written WAL	

TX state	PREPARING
Enlisted entries	(k1, v1)
Locked keys	k1
Written WAL	

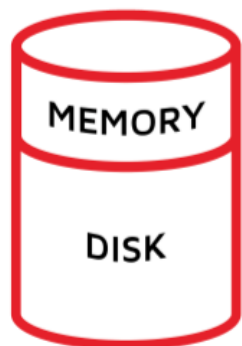


 NODE
primary



 NODE
primary

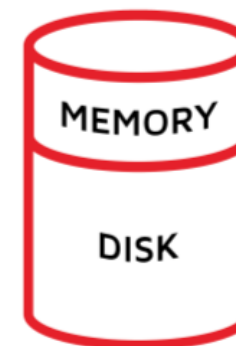
TX state	ACTIVE
Enlisted entries	
Locked keys	k2
Written WAL	



 NODE
backup

TX state	?
Enlisted entries	
Locked keys	
Written WAL	

TX state	?
Enlisted entries	
Locked keys	
Written WAL	



 NODE
backup

```
try (Transaction tx = txStart()) {
    cache.put(k1, v1); ✓

    cache.put(k2, v2); ✓

    tx.commit(); ⌛
}
```

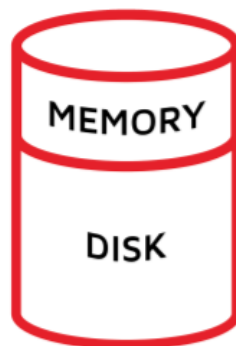


CLIENT

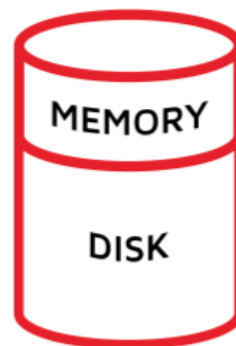
TX state	PREPARING
Enlisted entries	(k1, v1), (k2, v2)
Locked keys	
Written WAL	

prepare request

TX state	PREPARING
Enlisted entries	(k1, v1)
Locked keys	k1
Written WAL	

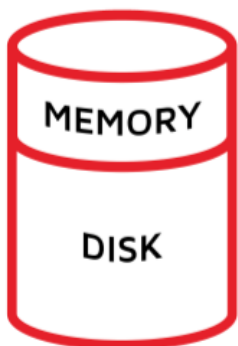


 NODE
primary



 NODE
primary

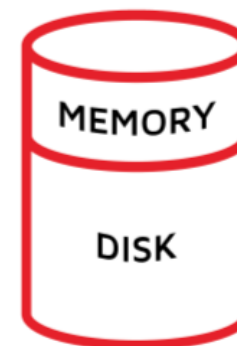
TX state	PREPARING
Enlisted entries	(k2, v2)
Locked keys	k2
Written WAL	



 NODE
backup

TX state	?
Enlisted entries	
Locked keys	
Written WAL	

TX state	?
Enlisted entries	
Locked keys	
Written WAL	



 NODE
backup


```
try (Transaction tx = txStart()) {
    cache.put(k1, v1); ✓

    cache.put(k2, v2); ✓

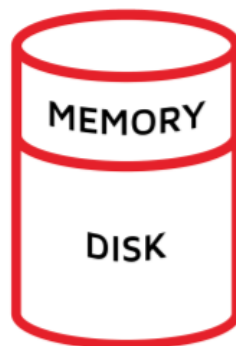
    tx.commit(); 🌀
}
```




CLIENT

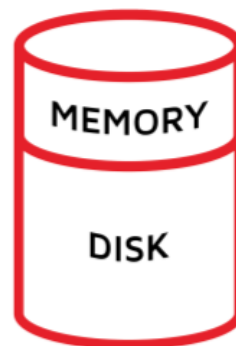
TX state	PREPARING
Enlisted entries	(k1, v1), (k2, v2)
Locked keys	
Written WAL	

TX state	PREPARING
Enlisted entries	(k1, v1)
Locked keys	k1
Written WAL	



 NODE

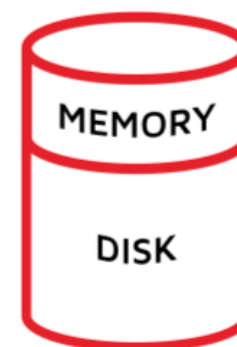
primary



 NODE

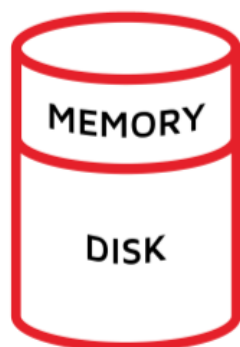
primary

TX state	PREPARING
Enlisted entries	(k2, v2)
Locked keys	k2
Written WAL	



 NODE

backup



 NODE

backup

prepare request

prepare request

TX state	PREPARED
Enlisted entries	(k1, v1)
Locked keys	k1
Written WAL	

TX state	PREPARED
Enlisted entries	(k2, v2)
Locked keys	k2
Written WAL	

```
try (Transaction tx = txStart()) {
    cache.put(k1, v1); ✓

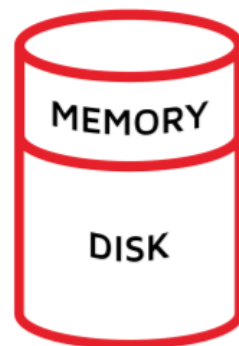
    cache.put(k2, v2); ✓

    tx.commit(); 🌀
}
```

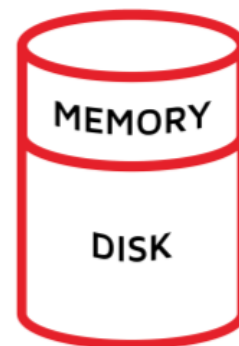


TX state	PREPARING
Enlisted entries	(k1, v1), (k2, v2)
Locked keys	
Written WAL	

TX state	PREPARED
Enlisted entries	(k1, v1)
Locked keys	k1
Written WAL	

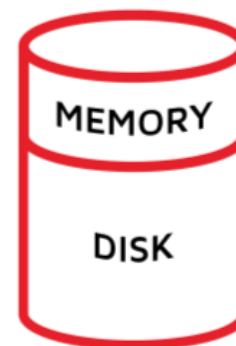


NODE



NODE

TX state	PREPARED
Enlisted entries	(k2, v2)
Locked keys	k2
Written WAL	



NODE

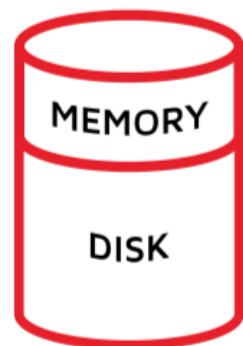
backup

prepare response

primary

primary

prepare response



NODE

backup

TX state	PREPARED
Enlisted entries	(k1, v1)
Locked keys	k1
Written WAL	

TX state	PREPARED
Enlisted entries	(k2, v2)
Locked keys	k2
Written WAL	

```
try (Transaction tx = txStart()) {
    cache.put(k1, v1); ✓
    cache.put(k2, v2); ✓
    tx.commit(); ⌛
}
```



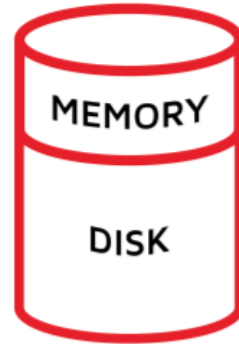
CLIENT

TX state	PREPARED
Enlisted entries	(k1, v1), (k2, v2)
Locked keys	
Written WAL	

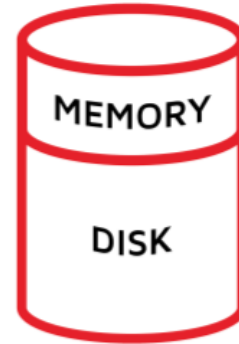
prepare response

prepare response

TX state	PREPARED
Enlisted entries	(k1, v1)
Locked keys	k1
Written WAL	

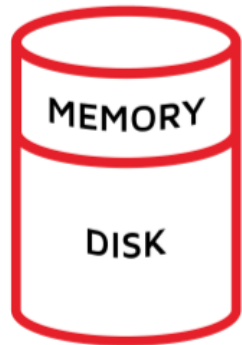


 NODE
primary



 NODE
primary

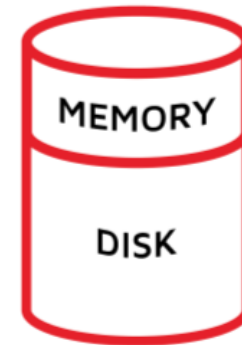
TX state	PREPARED
Enlisted entries	(k2, v2)
Locked keys	k2
Written WAL	



 NODE
backup

TX state	PREPARED
Enlisted entries	(k1, v1)
Locked keys	k1
Written WAL	

TX state	PREPARED
Enlisted entries	(k2, v2)
Locked keys	k2
Written WAL	



 NODE
backup

```
try (Transaction tx = txStart()) {
    cache.put(k1, v1); ✓

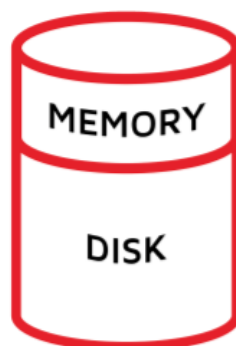
    cache.put(k2, v2); ✓

    tx.commit(); 🌀
}
```

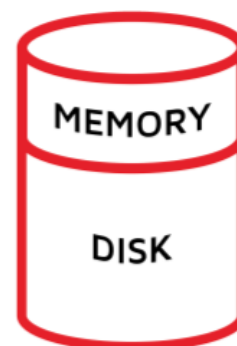


TX state	COMMITTING
Enlisted entries	(k1, v1), (k2, v2)
Locked keys	
Written WAL	

TX state	PREPARED
Enlisted entries	(k1, v1)
Locked keys	k1
Written WAL	

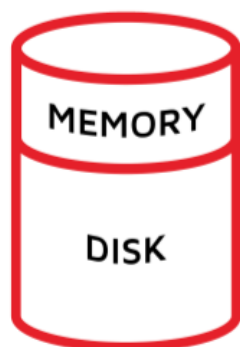


 NODE
primary



 NODE
primary

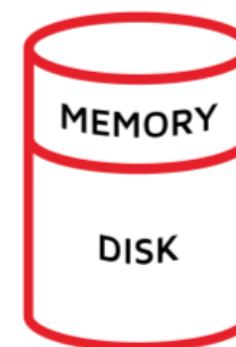
TX state	PREPARED
Enlisted entries	(k2, v2)
Locked keys	k2
Written WAL	



 NODE
backup

TX state	PREPARED
Enlisted entries	(k1, v1)
Locked keys	k1
Written WAL	

TX state	PREPARED
Enlisted entries	(k2, v2)
Locked keys	k2
Written WAL	



 NODE
backup

```
try (Transaction tx = txStart()) {
    cache.put(k1, v1); ✓
    cache.put(k2, v2); ✓
    tx.commit(); ⌛
}
```



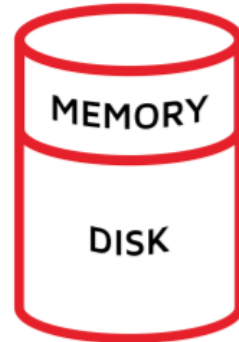
CLIENT

TX state	COMMITTING
Enlisted entries	(k1, v1), (k2, v2)
Locked keys	
Written WAL	

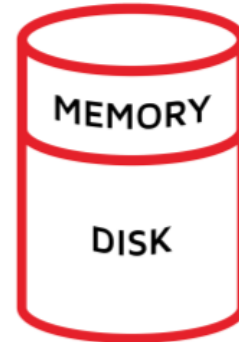
commit request

commit request

TX state	COMMITTING
Enlisted entries	(k1, v1)
Locked keys	k1
Written WAL	

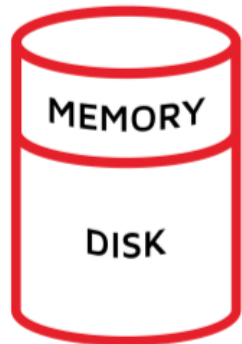


 NODE
primary



 NODE
primary

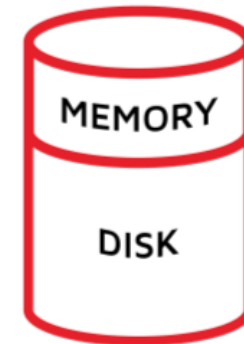
TX state	COMMITTING
Enlisted entries	(k2, v2)
Locked keys	k2
Written WAL	



 NODE
backup

TX state	PREPARED
Enlisted entries	(k1, v1)
Locked keys	k1
Written WAL	

TX state	PREPARED
Enlisted entries	(k2, v2)
Locked keys	k2
Written WAL	



 NODE
backup

```
try (Transaction tx = txStart()) {
    cache.put(k1, v1); ✓

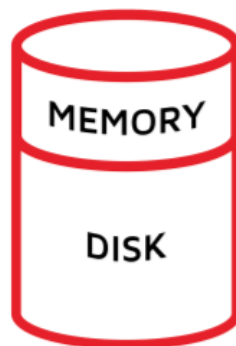
    cache.put(k2, v2); ✓

    tx.commit(); 🌀
}
```

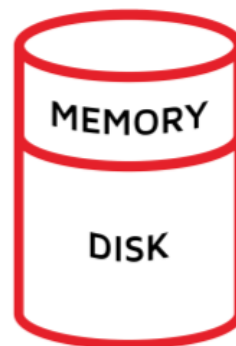


TX state	COMMITTING
Enlisted entries	(k1, v1), (k2, v2)
Locked keys	
Written WAL	

TX state	COMMITTING
Enlisted entries	(k1, v1)
Locked keys	
Written WAL	(k1, v1)

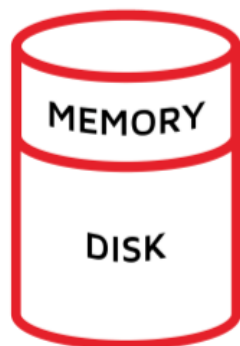


 NODE
primary



 NODE
primary

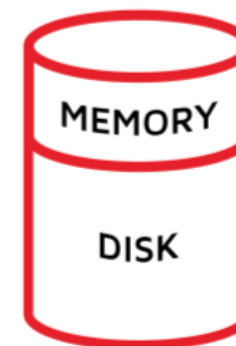
TX state	COMMITTING
Enlisted entries	(k2, v2)
Locked keys	
Written WAL	(k2, v2)



 NODE
backup

TX state	PREPARED
Enlisted entries	(k1, v1)
Locked keys	k1
Written WAL	

TX state	PREPARED
Enlisted entries	(k2, v2)
Locked keys	k2
Written WAL	



 NODE
backup

```
try (Transaction tx = txStart()) {
    cache.put(k1, v1); ✓

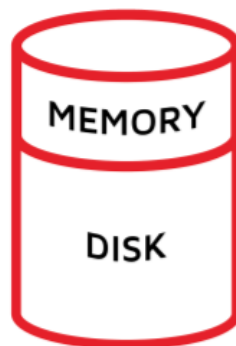
    cache.put(k2, v2); ✓

    tx.commit(); 🌀
}
```

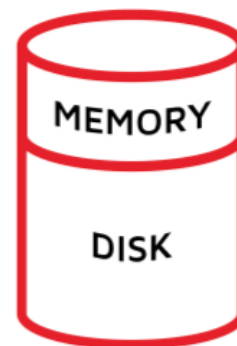


TX state	COMMITTING
Enlisted entries	(k1, v1), (k2, v2)
Locked keys	
Written WAL	

TX state	COMMITTING
Enlisted entries	(k1, v1)
Locked keys	
Written WAL	(k1, v1)

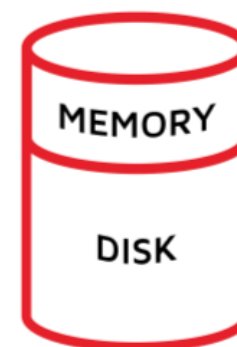


 **NODE**
primary



 **NODE**
primary

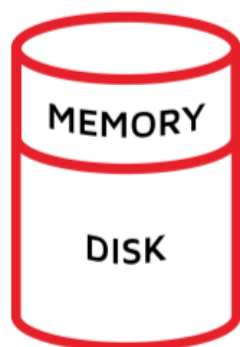
TX state	COMMITTING
Enlisted entries	(k2, v2)
Locked keys	
Written WAL	(k2, v2)



 **NODE**
backup

commit request

commit request



 **NODE**
backup

TX state	COMMITTING
Enlisted entries	(k1, v1)
Locked keys	k1
Written WAL	

TX state	COMMITTING
Enlisted entries	(k2, v2)
Locked keys	k2
Written WAL	

```
try (Transaction tx = txStart()) {
    cache.put(k1, v1); ✓

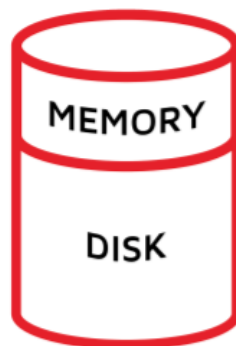
    cache.put(k2, v2); ✓

    tx.commit(); ⌛
}
```

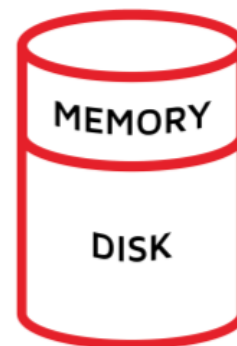


TX state	COMMITTING
Enlisted entries	(k1, v1), (k2, v2)
Locked keys	
Written WAL	

TX state	COMMITTING
Enlisted entries	(k1, v1)
Locked keys	
Written WAL	(k1, v1)

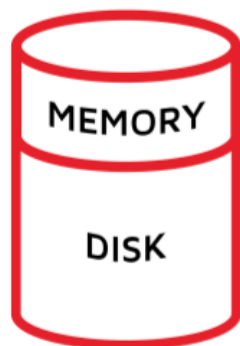


 NODE
primary



 NODE
primary

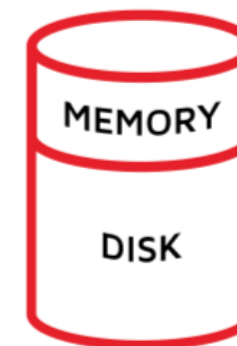
TX state	COMMITTING
Enlisted entries	(k2, v2)
Locked keys	
Written WAL	(k2, v2)



 NODE
backup

TX state	COMMITTED
Enlisted entries	(k1, v1)
Locked keys	
Written WAL	(k1, v1)

TX state	COMMITTED
Enlisted entries	(k2, v2)
Locked keys	
Written WAL	(k2, v2)



 NODE
backup


```
try (Transaction tx = txStart()) {
    cache.put(k1, v1); ✓

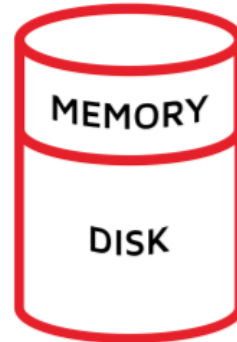
    cache.put(k2, v2); ✓

    tx.commit(); 🌀
}
```

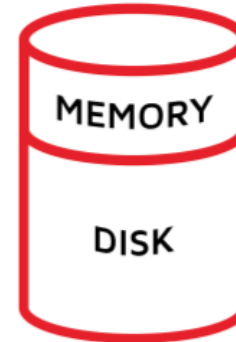


TX state	COMMITTING
Enlisted entries	(k1, v1), (k2, v2)
Locked keys	
Written WAL	

TX state	COMMITTED
Enlisted entries	(k1, v1)
Locked keys	
Written WAL	(k1, v1)

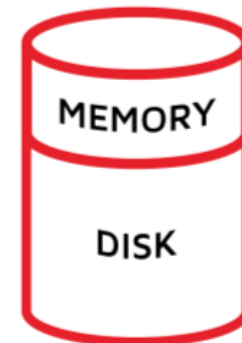


 **NODE**
primary

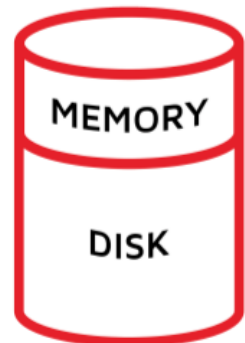


 **NODE**
primary

TX state	COMMITTED
Enlisted entries	(k2, v2)
Locked keys	
Written WAL	(k2, v2)



 **NODE**
backup



 **NODE**
backup

commit response

commit response

TX state	COMMITTED
Enlisted entries	(k1, v1)
Locked keys	
Written WAL	(k1, v1)

TX state	COMMITTED
Enlisted entries	(k2, v2)
Locked keys	
Written WAL	(k2, v2)

```
try (Transaction tx = txStart()) {
    cache.put(k1, v1); ✓
    cache.put(k2, v2); ✓
    tx.commit(); 🌀
}
```



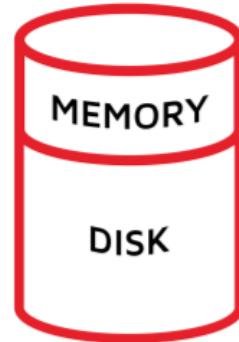
CLIENT

TX state	COMMITTED
Enlisted entries	(k1, v1), (k2, v2)
Locked keys	
Written WAL	

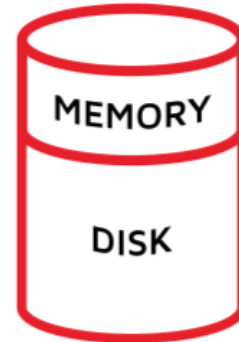
commit response

commit response

TX state	COMMITTED
Enlisted entries	(k1, v1)
Locked keys	
Written WAL	(k1, v1)

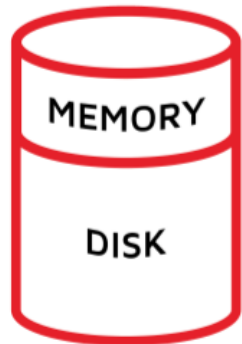


 NODE
primary



 NODE
primary

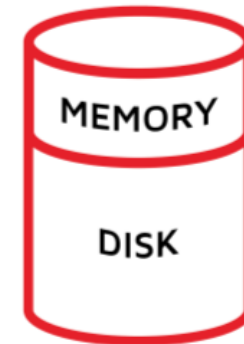
TX state	COMMITTED
Enlisted entries	(k2, v2)
Locked keys	
Written WAL	(k2, v2)



 NODE
backup

TX state	COMMITTED
Enlisted entries	(k1, v1)
Locked keys	
Written WAL	(k1, v1)

TX state	COMMITTED
Enlisted entries	(k2, v2)
Locked keys	
Written WAL	(k2, v2)



 NODE
backup

```
try (Transaction tx = txStart()) {
    cache.put(k1, v1); ✓

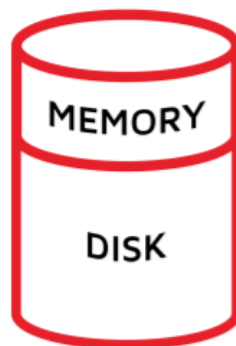
    cache.put(k2, v2); ✓

    tx.commit(); ✓
}
```

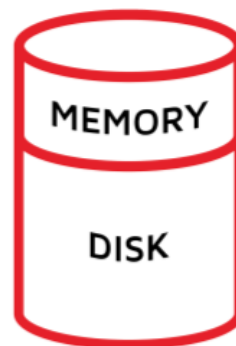


TX state	COMMITTED
Enlisted entries	(k1, v1), (k2, v2)
Locked keys	
Written WAL	

TX state	COMMITTED
Enlisted entries	(k1, v1)
Locked keys	
Written WAL	(k1, v1)

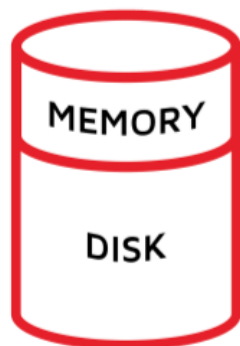


 NODE
primary



 NODE
primary

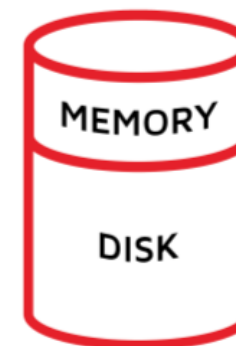
TX state	COMMITTED
Enlisted entries	(k2, v2)
Locked keys	
Written WAL	(k2, v2)



 NODE
backup

TX state	COMMITTED
Enlisted entries	(k1, v1)
Locked keys	
Written WAL	(k1, v1)

TX state	COMMITTED
Enlisted entries	(k2, v2)
Locked keys	
Written WAL	(k2, v2)



 NODE
backup

Transaction flow and commit protocol: reference



- Before commit, updates are enlisted on only the initiator node.
- Locks are acquired on the primary node.
- On first primary access, the transaction is mapped to a topology version.
- On commit, a two-phase commit procedure is started.
- In the prepare phase, locks for all data are acquired on primaries and backups.
- In the commit phase, data is changed in storage, and locks are released.

Transactions interaction with PME



Partition Map Exchange (PME) is a distributed, partition-state consolidation protocol that is triggered on every topology change.

- Reference: Partition Map Exchange under the hood
 - <https://cwiki.apache.org/confluence/display/IGNITE/%28Partition+Map%29+Exchange+--+under+the+hood>
- Behavior: PME inflicts stop-the-world pause on all user loads.

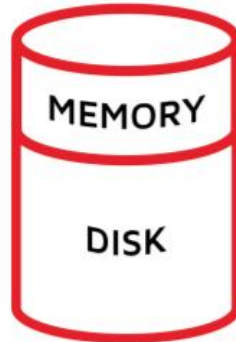
```
try (Transaction tx = txStart()) {
```



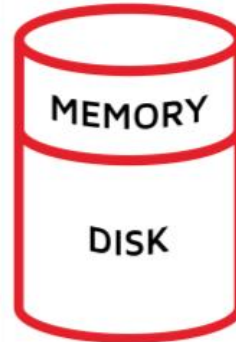
CLIENT

TX state

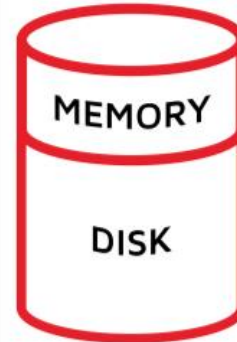
ACTIVE



 NODE
primary



 NODE
primary



 NODE
primary



CLIENT

topology version = (2, 0)

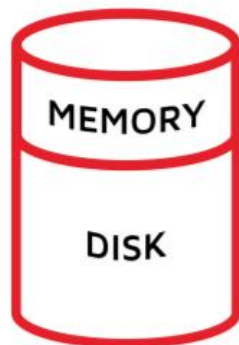
```
try (Transaction tx = txStart()) {  
    cache.put(k1, v1); ✓
```



CLIENT

TX state

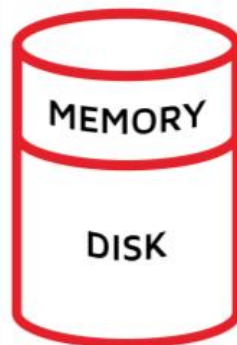
ACTIVE (mapped 2,0)



NODE

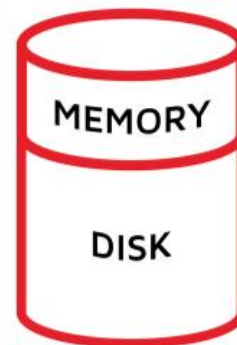
primary

k1 locked ✓



NODE

primary



NODE

primary



CLIENT

topology version = (2, 0)

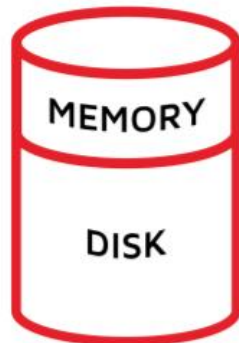
```
try (Transaction tx = txStart()) {  
    cache.put(k1, v1); ✓
```



CLIENT

TX state

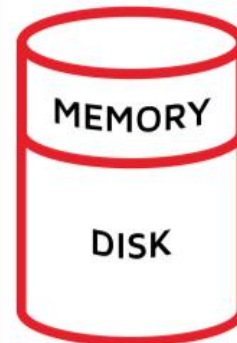
ACTIVE (mapped 2,0)



NODE
primary
k1 locked ✓



NODE
primary



NODE
primary



CLIENT

topology version = (3, 0)


```
try (Transaction tx = txStart()) {  
    cache.put(k1, v1); ✓
```

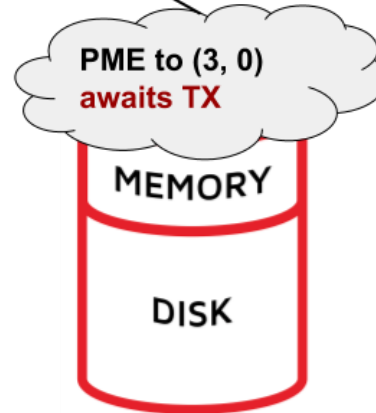


CLIENT

TX state

ACTIVE (mapped 2,0)

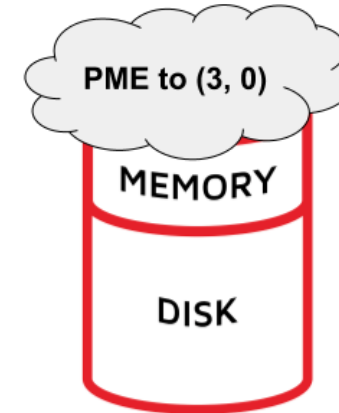
blocked by



NODE
primary
k1 locked ✓



NODE
primary



NODE
primary



CLIENT

topology version = (3, 0)

```
try (Transaction tx = txStart()) {  
    cache.put(k1, v1); ✓
```



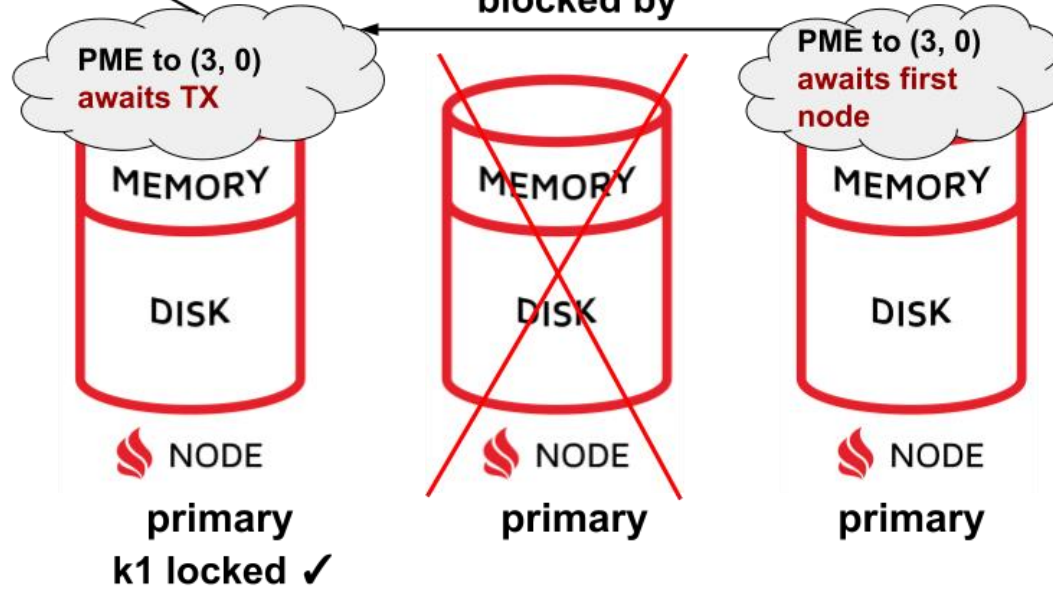
CLIENT

TX state

ACTIVE (mapped 2,0)

blocked by

blocked by



CLIENT

topology version = (3, 0)

```
try (Transaction tx = txStart()) {  
    cache.put(k1, v1); ✓
```

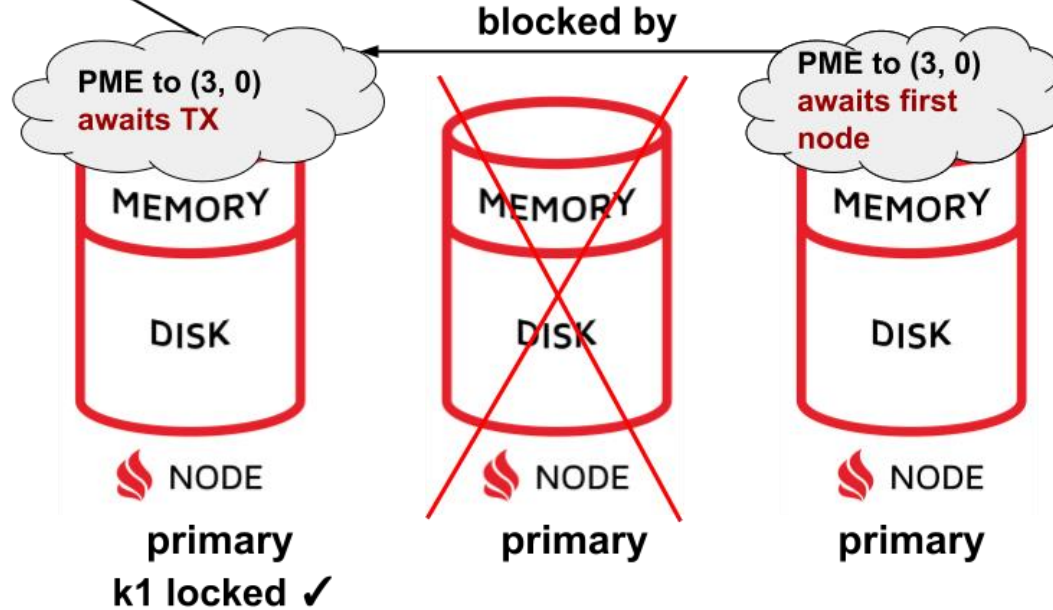


CLIENT

TX state

ACTIVE (mapped 2,0)

blocked by



TX state

ACTIVE



CLIENT

```
try (Transaction tx = txStart()) {
```

topology version = (3, 0)

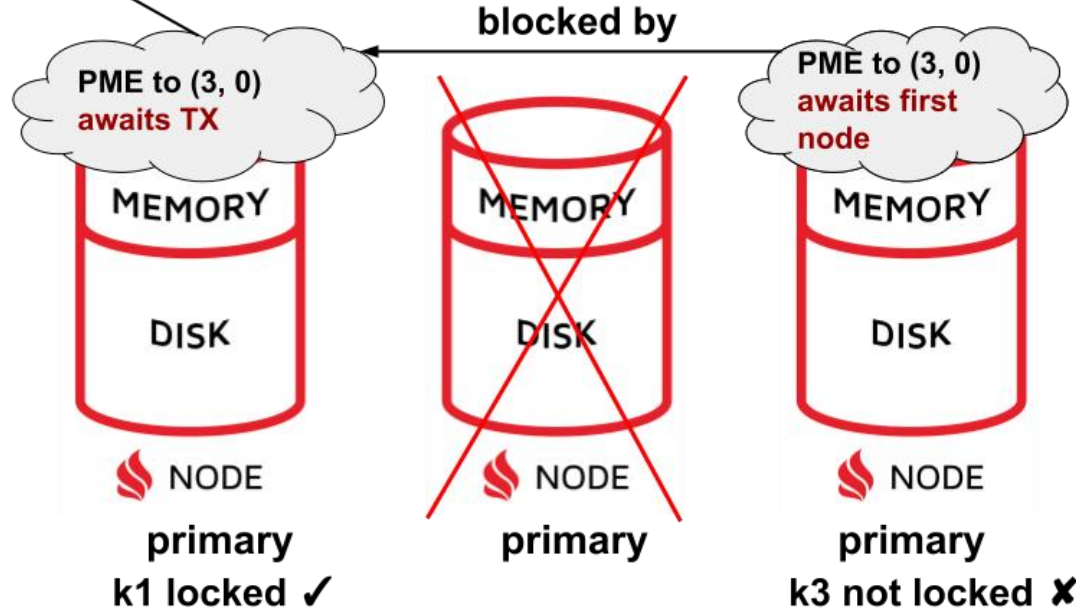
```
try (Transaction tx = txStart()) {
    cache.put(k1, v1); ✓
```



CLIENT

TX state	ACTIVE (mapped 2,0)
----------	---------------------

blocked by



TX state	ACTIVE (mapped 3,0)
----------	---------------------



CLIENT

```
try (Transaction tx = txStart()) {
    cache.put(k3, v3); ⌛
```

topology version = (3, 0)

```
try (Transaction tx = txStart()) {  
    cache.put(k1, v1); ✓
```

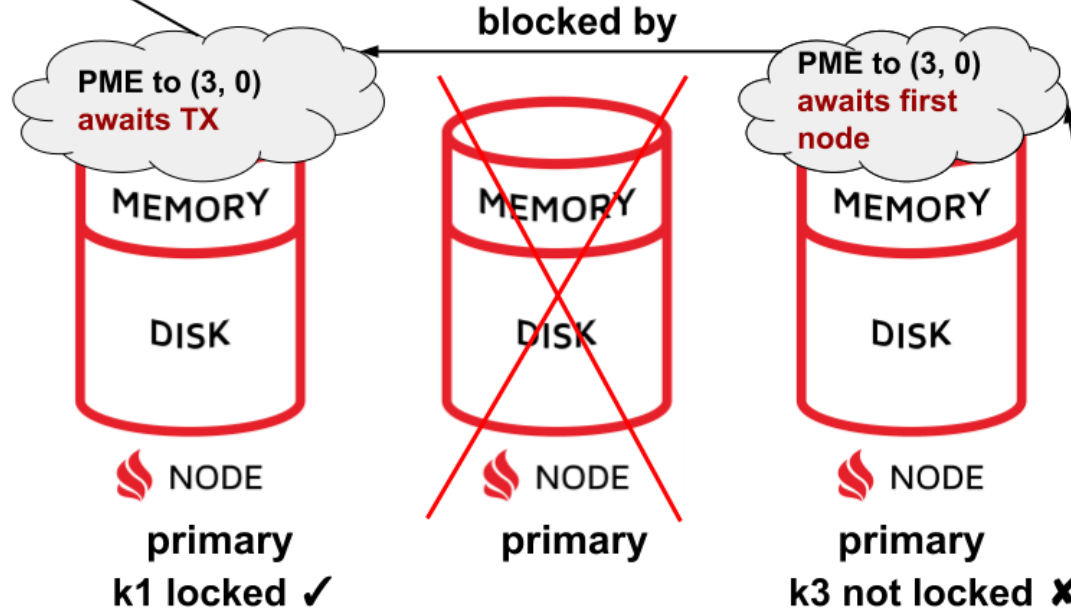


CLIENT

TX state

ACTIVE (mapped 2,0)

blocked by



blocked by

TX state

ACTIVE (mapped 3,0)



CLIENT

```
try (Transaction tx = txStart()) {  
    cache.put(k3, v3); ⚙️  
    // awaits PME
```

topology version = (3, 0)

Transaction interaction with PME



PME is a distributed, partition-state consolidation protocol that is triggered on every topology change.

- PME is triggered every time topology changes.
- PME to topVer=X can't finish if there are TXs mapped to $Y < X$.
- TXs can't map to topVer=X if PME isn't finished.

Caution: PME while long TX is in progress freezes all successive TXs

TX recovery



TX recovery is a safe transaction finish protocol that is used if a coordinator node or a primary node fails.

```
try (Transaction tx = txStart()) {
    cache.put(k1, v1); ✓

    cache.put(k2, v2); ✓

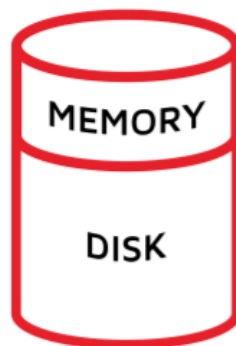
    tx.commit(); ⌛
}
```



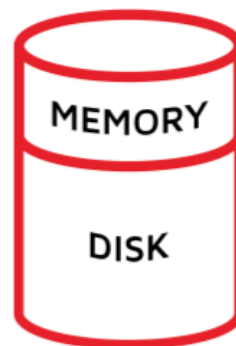
CLIENT

TX state	COMMITTING
Enlisted entries	(k1, v1), (k2, v2)
Locked keys	
Written WAL	

TX state	PREPARED
Enlisted entries	(k1, v1)
Locked keys	k1
Written WAL	

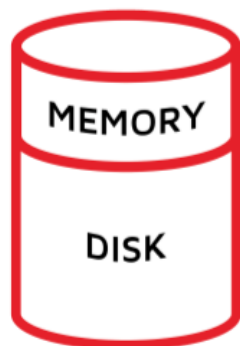


 NODE
primary



 NODE
primary

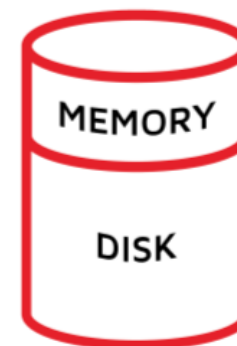
TX state	PREPARED
Enlisted entries	(k2, v2)
Locked keys	k2
Written WAL	



 NODE
backup

TX state	PREPARED
Enlisted entries	(k1, v1)
Locked keys	k1
Written WAL	

TX state	PREPARED
Enlisted entries	(k2, v2)
Locked keys	k2
Written WAL	



 NODE
backup


```
try (Transaction tx = txStart()) {
    cache.put(k1, v1); ✓

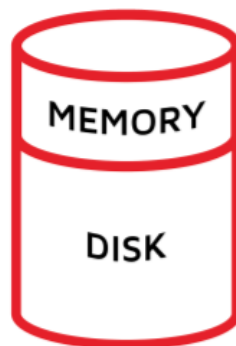
    cache.put(k2, v2); ✓

    tx.commit(); 🌀
}
```

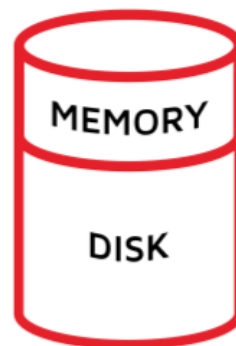


TX state	COMMITTING
Enlisted entries	(k1, v1), (k2, v2)
Locked keys	
Written WAL	

TX state	PREPARED
Enlisted entries	(k1, v1)
Locked keys	k1
Written WAL	

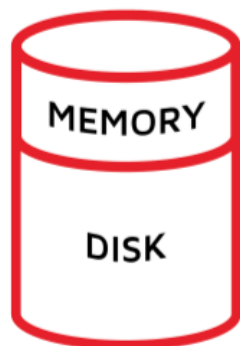


 NODE
primary



 NODE
primary

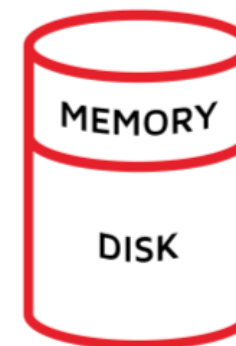
TX state	PREPARED
Enlisted entries	(k2, v2)
Locked keys	k2
Written WAL	



 NODE
backup

TX state	PREPARED
Enlisted entries	(k1, v1)
Locked keys	k1
Written WAL	

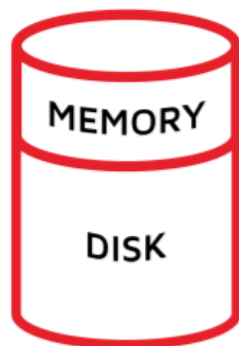
TX state	PREPARED
Enlisted entries	(k2, v2)
Locked keys	k2
Written WAL	



 NODE
backup

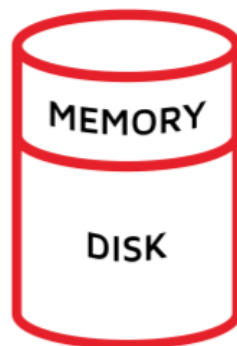
TX state	PREPARED
Enlisted entries	(k1, v1)
Locked keys	k1
Written WAL	

???



 NODE
primary

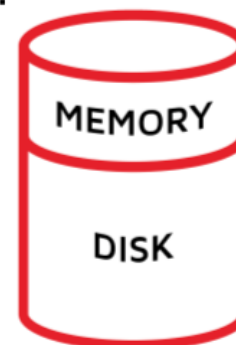
???



 NODE
primary

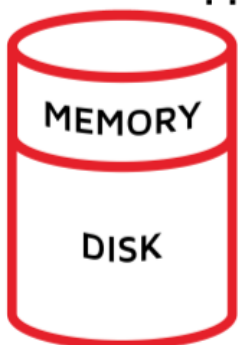
TX state	PREPARED
Enlisted entries	(k2, v2)
Locked keys	k2
Written WAL	

???



 NODE
backup

???

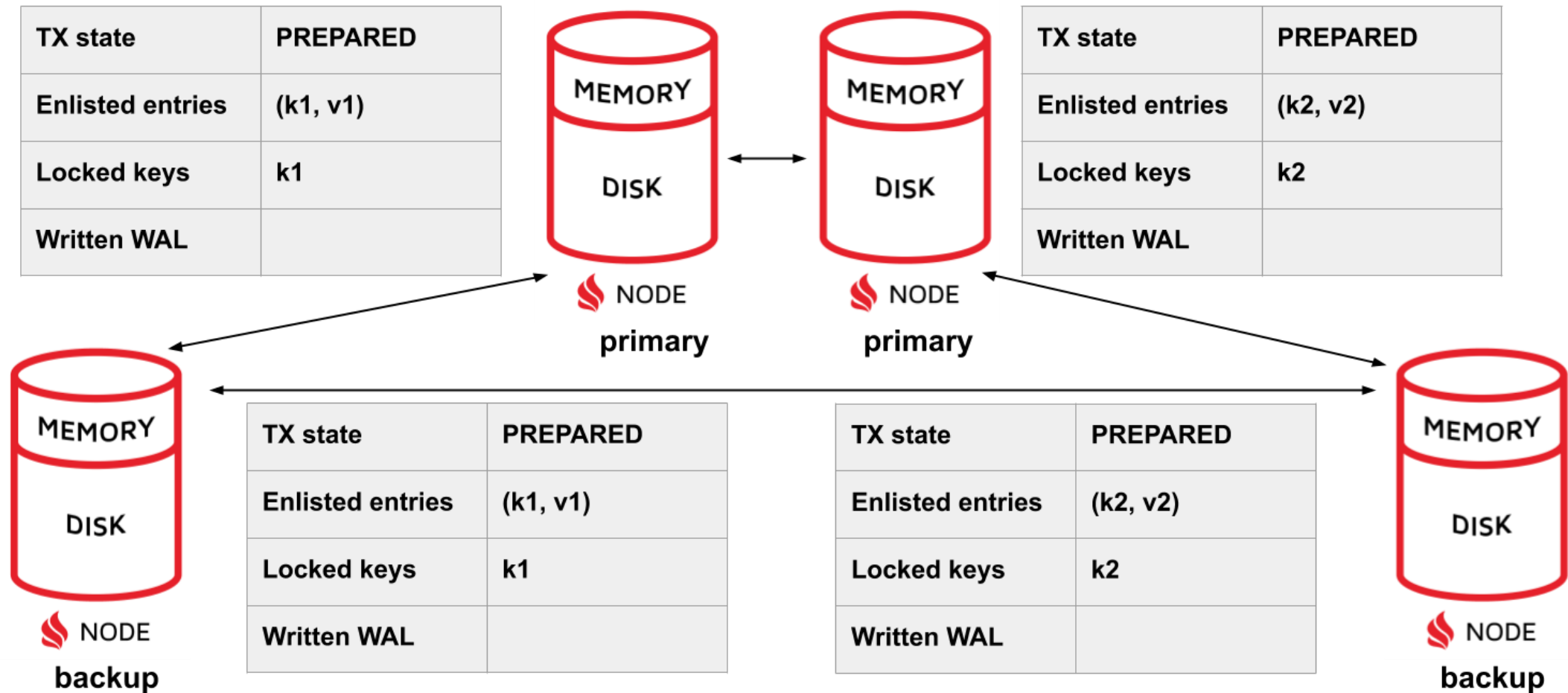


 NODE
backup

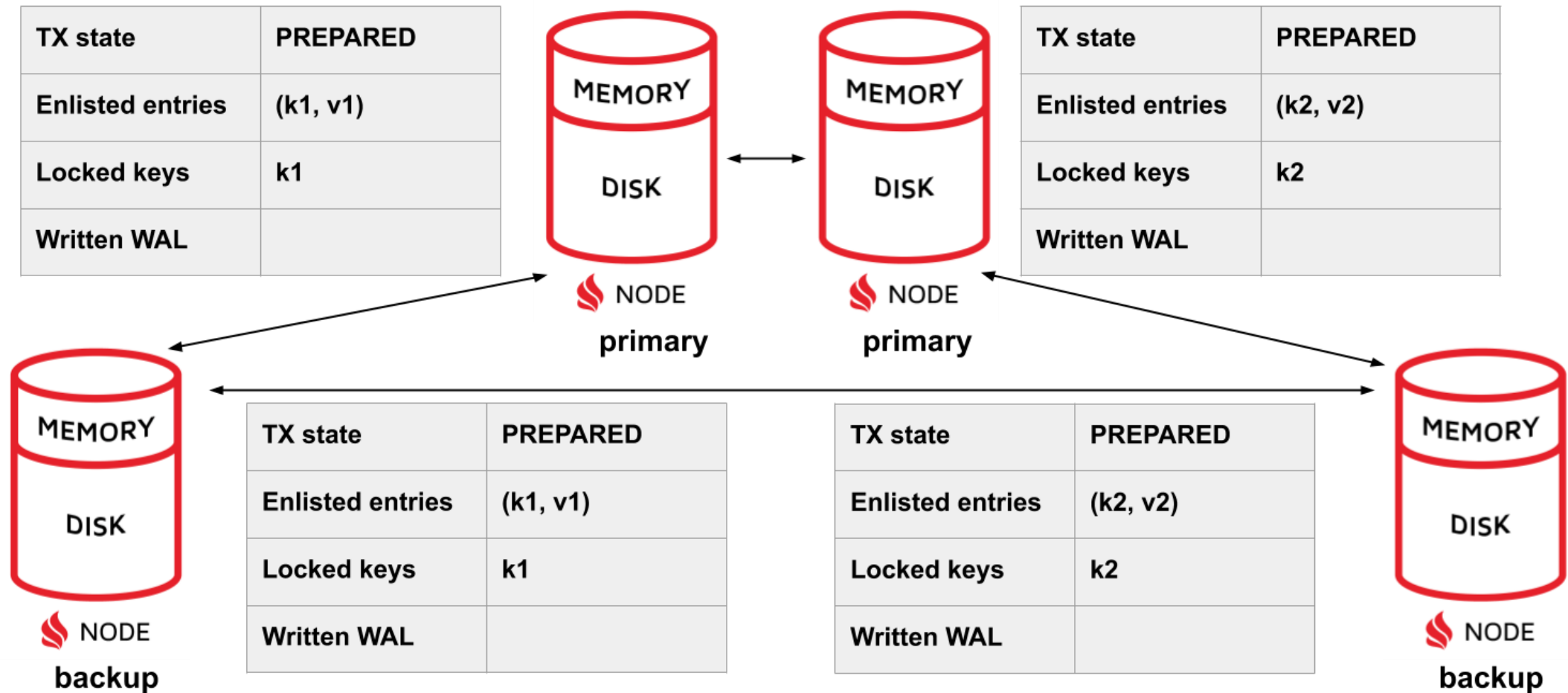
TX state	PREPARED
Enlisted entries	(k1, v1)
Locked keys	k1
Written WAL	

TX state	PREPARED
Enlisted entries	(k2, v2)
Locked keys	k2
Written WAL	

TX recovery
messages
between all
participants



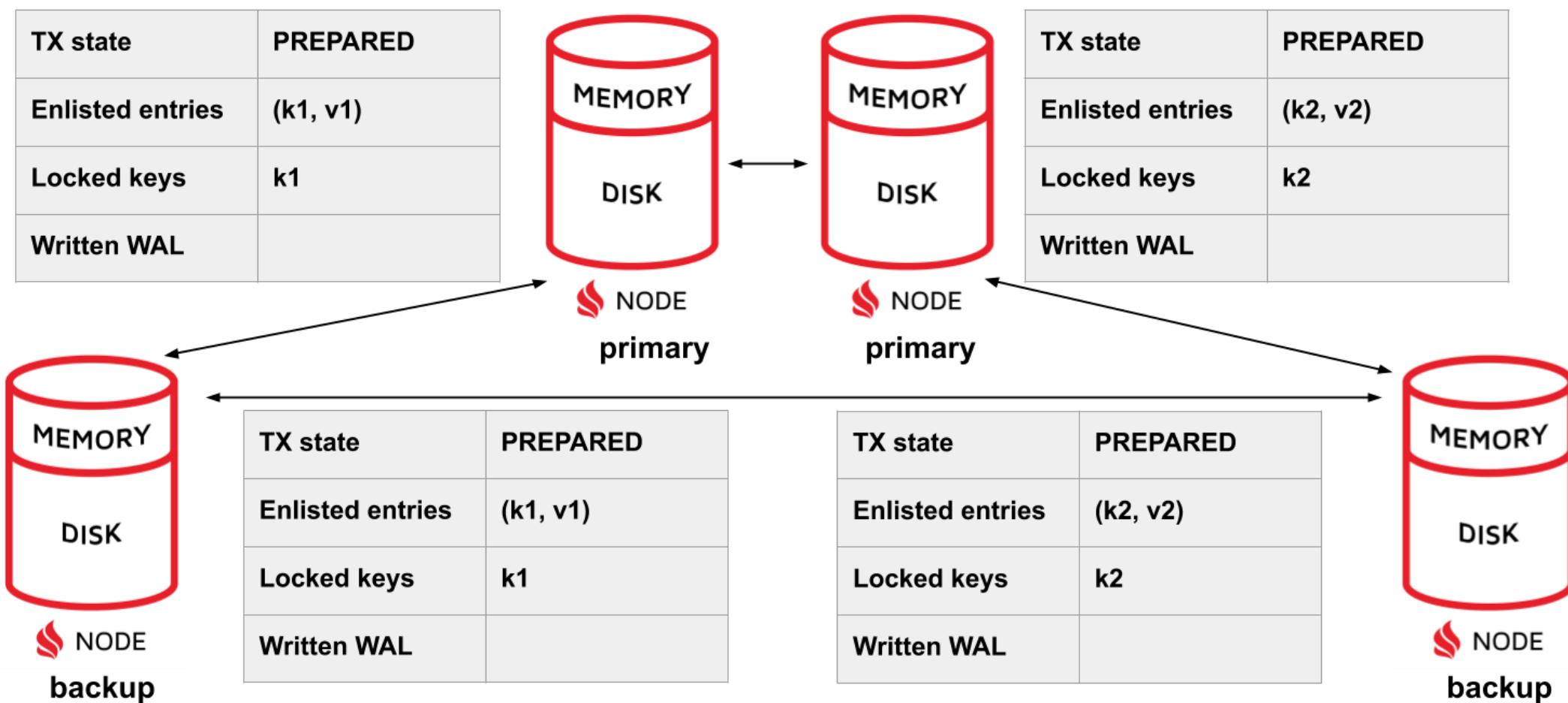
Node	TX state
primary 1	PREPARED
primary 2	PREPARED
backup 1	PREPARED
backup 2	PREPARED



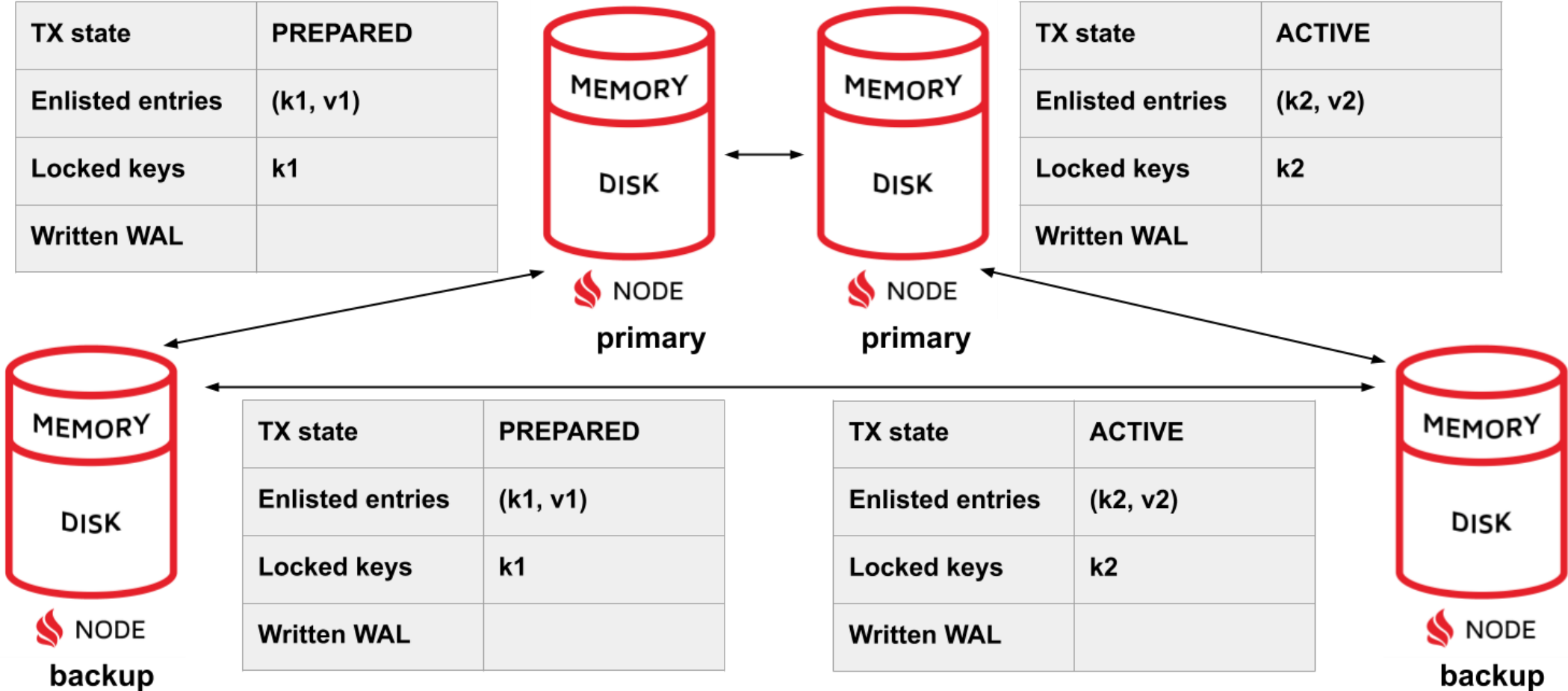
Node	TX state
primary 1	PREPARED
primary 2	PREPARED
backup 1	PREPARED
backup 2	PREPARED

resolution

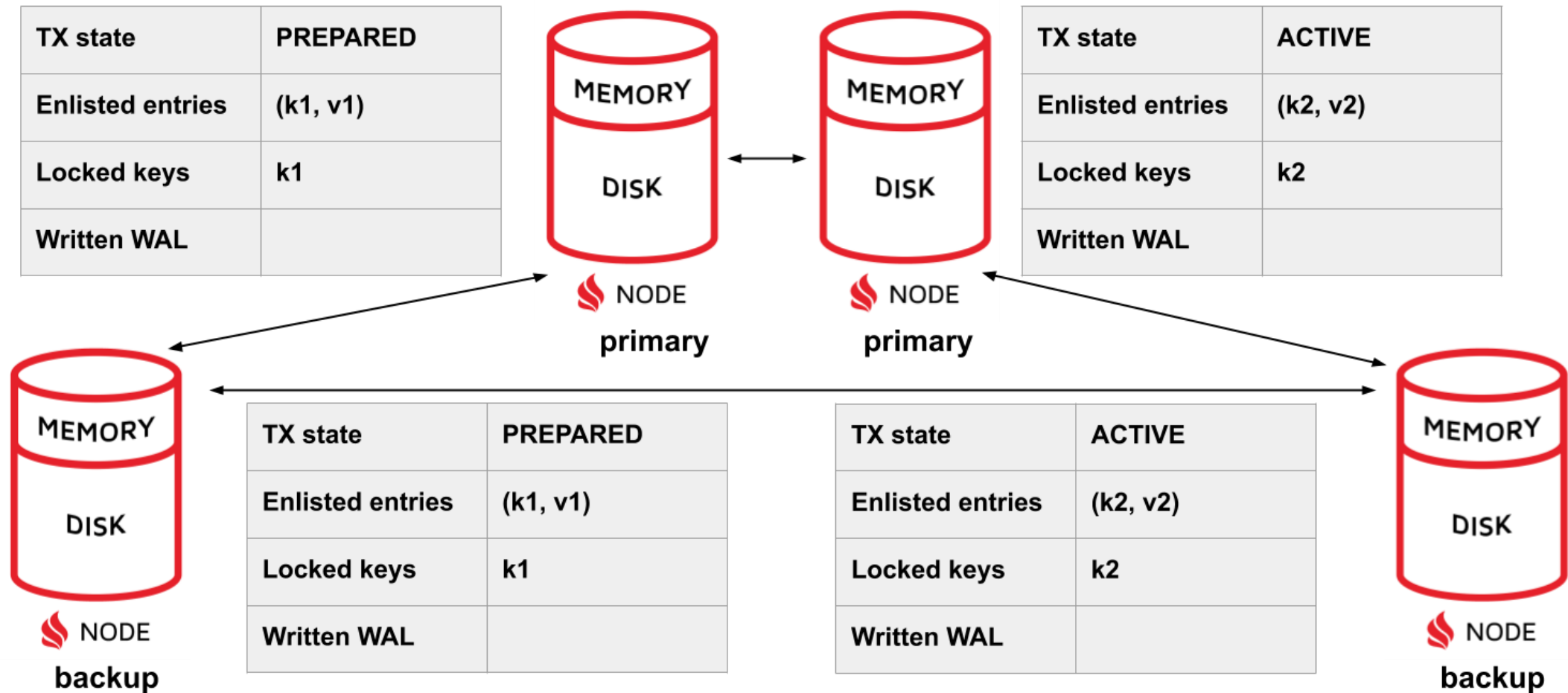
TX is at least PREPARED on all participants.
Safe commit is possible.



TX recovery
messages
between all
participants



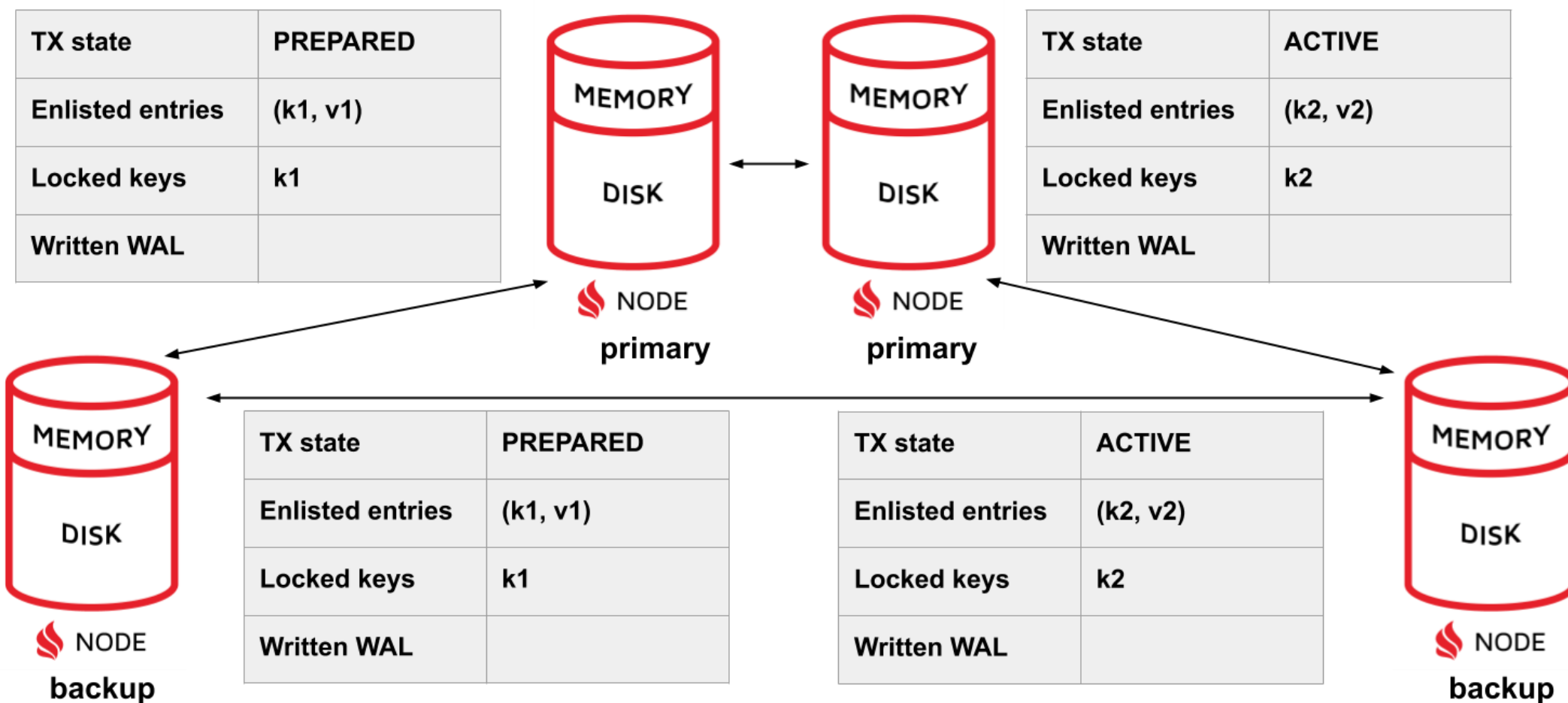
Node	TX state
primary 1	PREPARED
primary 2	ACTIVE
backup 1	PREPARED
backup 2	ACTIVE



Node	TX state
primary 1	PREPARED
primary 2	ACTIVE
backup 1	PREPARED
backup 2	ACTIVE

resolution

TX is not PREPARED on some participants.
No data is updated anywhere.
Safe rollback is possible.



TX recovery: reference



- When a participant node crashes, distributed TX recovery starts.
- All participant nodes exchange TX state.
- If all nodes are PREPARED and above, TX is committed.
- If a node is below PREPARED, TX is rolled back.
- Invariants:
 - If TX has changed data somewhere, TX is PREPARED or above on all nodes.
 - If TX is below PREPARED somewhere, TX hasn't changed data anywhere.

Agenda



- API overview
- Isolation and concurrency
- What happens under the hood
 - Transaction flow and commit protocol
 - TX recovery
- Tips for moving safely into production
 - Recommendations to avoid delays
 - Troubleshooting

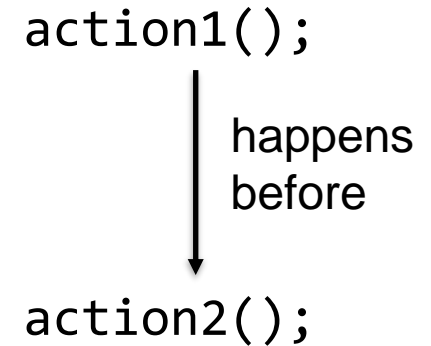
Pitfall: key contention



- Key locks are exclusive.
- Concurrent transactions that access the same key queue on the key.

```
try (Transaction tx1 = txs.txStart(PESSIMISTIC, REPEATABLE_READ)) {  
    cache.put(hotKey, val1);  
  
    action1();  
}
```

```
try (Transaction tx2 = txs.txStart(PESSIMISTIC, REPEATABLE_READ)) {  
    cache.put(hotKey, val2);  
  
    action2();  
}
```



- TX latency increases, all of the load freezes in case of PME.

Pitfall: heavy activities inside the transaction



```
try (Transaction tx = txs.txStart(PESSIMISTIC, REPEATABLE_READ)) {  
    cache.put(k, v);  
  
    callToExternalSlowService(); // Keys and topology are locked  
}
```

- Dependent load may freeze for the duration of the long call.
- In the case of PME, all of the load freezes.

Pitfall: key-level deadlock



```
Thread 1:
try (Transaction tx = txs.txStart(PESSIMISTIC, REPEATABLE_READ)) {
    cache.put(k1, v1);
    cache.put(k2, v2);
}

Thread 2:
try (Transaction tx = txs.txStart(PESSIMISTIC, REPEATABLE_READ)) {
    cache.put(k2, v2);
    cache.put(k1, v1);
}
```

Deadlock risk on k1 and k2 key locks

Avoiding pitfalls



Avoiding pitfalls: timeouts



- `txTimeout`
Required for production use
- `txConfiguration.setDefaultTxTimeout(timeout);`
Applied for transactions that do not have an explicit timeout specification
- `txConfiguration.setTxTimeoutOnPartitionMapExchange();`
Used to specify a shorter timeout, in case PME is in progress

Avoiding pitfalls: deadlocks



- `txConfiguration.setDefaultTxTimeout(timeout);`
Deadlock detection is triggered only on timeout
- Preserve global order of key access

```
try (Transaction tx = txs.txStart(PESSIMISTIC, REPEATABLE_READ)) {  
    cache.putAll(new TreeMap<>(updateMap));  
}
```

- Use OPTIMISTIC, SERIALIZABLE if other options are not possible

Troubleshooting: reading logs



Reading logs: long-running transactions



Logs report on long-running transactions.

```
[2020-06-08 03:26:00,858][WARN ][grid-timeout-worker-#23%internal.TransactionsMXBeanImplTest0%][root]
First 10 long running transactions [total=1]
[2020-06-08 03:26:00,859][WARN ][grid-timeout-worker-#23%internal.TransactionsMXBeanImplTest0%][root]
>>> Transaction [startTime=03:26:00.447, curTime=03:26:00.855, systemTime=0, userTime=408,
tx=GridNearTxLocal [mappings=IgniteTxMappingsImpl [], nearLocallyMapped=false,
colocatedLocallyMapped=false, needCheckBackup=null, hasRemoteLocks=false, trackTimeout=false,
systemTime=0, systemStartTime=0, prepareStartTime=0, prepareTime=0, commitOrRollbackStartTime=0,
```

Reading logs: TX deadlock



Logs provide detailed information about TX deadlock.

Deadlock detected:

K1: TX1 holds lock, TX2 waits lock.
K2: TX3 holds lock, TX1 waits lock.
K3: TX4 holds lock, TX3 waits lock.
K4: TX2 holds lock, TX4 waits lock.

Transactions:

TX1 [txId=GridCacheVersion [topVer=203056358, order=1591576409356, nodeOrder=2], nodeId=09ec0592-f9d1-4824-8dd3-5991a3300001, threadId=1335]
TX2 [txId=GridCacheVersion [topVer=203056358, order=1591576409357, nodeOrder=3], nodeId=e8d66155-873c-406b-a696-9c846fa00002, threadId=1331]
TX3 [txId=GridCacheVersion [topVer=203056358, order=1591576409358, nodeOrder=1], nodeId=8598c47f-55f2-467d-b838-985a7e600000, threadId=1333]
TX4 [txId=GridCacheVersion [topVer=203056358, order=1591576409357, nodeOrder=1], nodeId=8598c47f-55f2-467d-b838-985a7e600000, threadId=1336]

Keys:

K1 [key=30, cache=cache]
K2 [key=20, cache=cache]
K3 [key=11, cache=cache]
K4 [key=18, cache=cache]

Reading logs: heavy activities inside the transaction



Logs provide stack trace of the TX owner thread on the initiator node.

```
[2020-06-08 03:37:42,127][WARN ][grid-timeout-worker-#598%internal.TransactionsMXBeanImplTest0%][root]
Dumping the near node thread that started transaction [xidVer=GridCacheVersion [topVer=203056662,
order=1591576659549, nodeOrder=1], nodeId=9fdd51f0-12d4-40d8-99f7-4a12ecc00000]
Stack trace of the transaction owner thread:
Thread [name="test-runner-#576%internal.TransactionsMXBeanImplTest%", id=620, state=TIMED_WAITING,
blockCnt=1, waitCnt=7]
    at java.lang.Thread.sleep(Native Method)
    at o.a.i.i.util.IgniteUtils.sleep(IgniteUtils.java:8023)
    at o.a.i.testframework.GridTestUtils.waitForCondition(GridTestUtils.java:1969)
    at
o.a.i.i.TransactionsMXBeanImplTest.checkLongOperationsDumpTimeoutViaTxMxBean(TransactionsMXBeanImplTest
.java:297)
    at
o.a.i.i.TransactionsMXBeanImplTest.testLongOperationsDumpTimeoutPositive(TransactionsMXBeanImplTest.jav
a:131)
    at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
```

Reading logs: PME is blocked by transaction



Logs report if PME can't finish due to an ongoing transaction.

```
[2020-06-08 03:45:39,090][WARN ][exchange-worker-#185%persistence.IgnitePdsTxCacheRebalancingTest1%]  
[diagnostic] Failed to wait for partition release future [topVer=AffinityTopologyVersion [topVer=7,  
minorTopVer=0], node=b4bf7fbe-8250-4f1b-9501-b2b78c600001]  
[2020-06-08 03:45:39,124][WARN ][exchange-worker-#185%persistence.IgnitePdsTxCacheRebalancingTest1%]  
[diagnostic] Pending transactions:  
[2020-06-08 03:45:39,125][WARN ][exchange-worker-#185%persistence.IgnitePdsTxCacheRebalancingTest1%]  
[diagnostic] >>> [txVer=AffinityTopologyVersion [topVer=4, minorTopVer=1], exchWait=true,  
tx=GridNearTxLocal [mappings=IgniteTxMappingsImpl [], nearLocallyMapped=false,  
colocatedLocallyMapped=false, needCheckBackup=null, hasRemoteLocks=true, trackTimeout=false,  
systemTime=13593100, systemStartTime=924183821725900, prepareStartTime=0, prepareTime=0,  
commitOrRollbackStartTime=0, commitOrRollbackTime=0,  
txDumpsThrottling=o.a.i.i.processors.cache.transactions.IgniteTxManager$TxDumpsThrottling@6f26bf1c,
```

TX tracing: coming soon



- Reports activity on all nodes as OpenCensus spans to the tracing system
- Allows tracing of the complete transaction flow
- Enables tracing of transactions selectively by label

```
ignite.transactions().withLabel("end-of-day clearing");
```



Dashboard



Alerting



Tracing



SQL



Baseline



Snapshots



Clusters

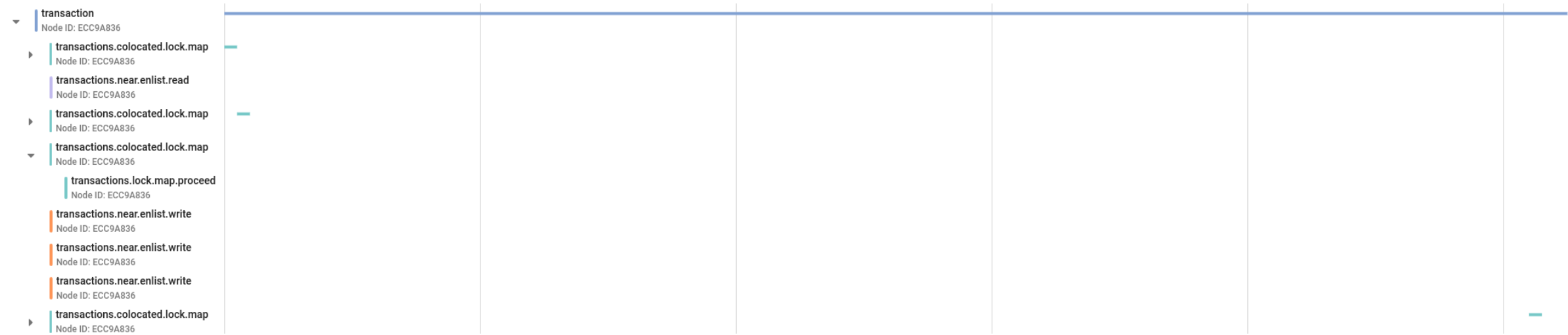


Admin

Name	Start Time ↓	Duration	Total Spans	Details
transaction	Jun 8, 12:07:11.221	107 ms	29	
transaction	Jun 8, 12:07:10.563	106 ms	41	
transaction	Jun 8, 12:07:08.542	109 ms	41	
transaction	Jun 8, 12:07:07.855	114 ms	41	
transaction	Jun 8, 12:07:06.116	116 ms	41	
discovery.node.join.request	Jun 8, 12:07:00.111	58 ms	9	
discovery.custom.event	Jun 8, 12:06:48.841	4 ms	3	Message Class: ChangeGlobalStateFinishMessage
discovery.custom.event	Jun 8, 12:06:48.827	7 ms	4	Message Class: DistributedMetaStorageCasMessage
discovery.custom.event	Jun 8, 12:06:48.519	38 ms	4	Message Class: DistributedMetaStorageUpdateMessage
discovery.custom.event	Jun 8, 12:06:48.512	384 ms	5	Message Class: ChangeGlobalStateMessage
discovery.node.join.request	Jun 8, 12:06:48.403	102 ms	7	

transaction

Trace Start June 8, 2020 at 12:07:10.563 GMT+3 Duration 106 ms Depth 6 Total Spans 41



TX tracing: reference



- IEP-48: Tracing
 - <https://cwiki.apache.org/confluence/display/IGNITE/IEP-48%3A+Tracing>
- Tracing documentation on GridGain dev portal
 - <https://www.gridgain.com/docs/control-center/latest/tracing>



Thanks for your attention!

Questions?

e-mail: irakov@gridgain.com

public list for discussions: user@ignite.apache.org