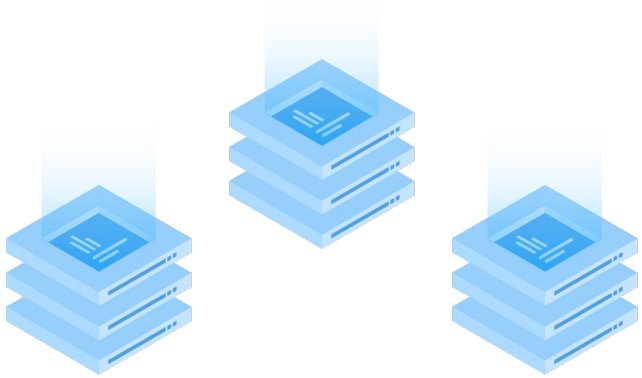
Head First
Distributed Transaction

LiuTang@PingCAP



About me

- Chief Engineer of PingCAP
- Open sources: go-mysql, go-mysql-elasticsearch, LedisDB, etc.

Agenda

1. ACID
2. Consistency Model
3. Practice

PingCAP

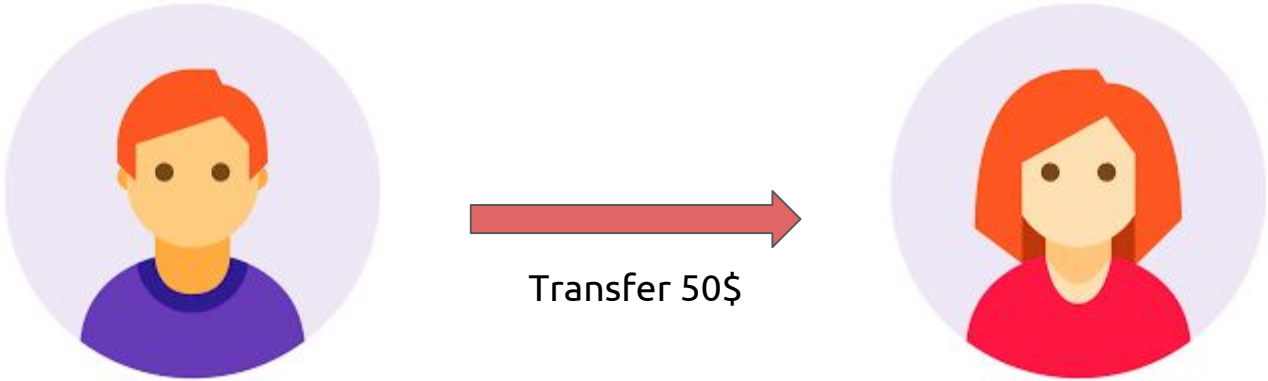
Appetizer

Transaction

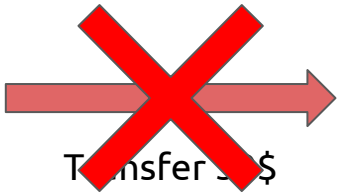
Bob
100$

Alice
100$

Transaction



Transaction



Bob
50$

Alice
100$

Transaction

```
mysql> begin;

mysql> update account set balance = balance - 50 where name = “Bob”;

mysql> update account set balance = balance + 50 where name = “Alice”;

mysql> commit;
```

Transaction

Wiki:

A transaction symbolizes **a unit of work** performed within a database management system (or similar system) against a database, and treated in a coherent and reliable way independent of other transactions. A transaction generally represents any change in a database.

A distributed transaction is a database transaction in which **two or more network hosts** are involved. Usually, hosts provide transactional resources, while the transaction manager is responsible for creating and managing a global transaction that encompasses all operations against such resources.

PingCAP

ACID

ACID

- Atomicity
- Concistency
- Isolation
- Durability

Isolation Levels

- Read Uncommitted
- Read Committed
- Cursor Stability
- Monotonic Atomic View
- Repeated Read
- Snapshot
- Serializable

Anomalies - Dirty Write(P0)

T1: write x = y = 1

T2: write x = y = 2

Expect: x = 1, y = 1, but Got: x = 2, y = 1

T1	Wx(1)				Wy(1)	C
T2		Wx(2)	Wy(2)	C		

Anomalies - Dirty Read(P1)

x = y = 50, must ensure: x + y = 100

T1: x transfers 40 to y

T2 reads x = 10, 10 + 50 != 100

T1	Wx(10)			Wy(90)	C
T2		Rx(10)	Ry(50)		C

Anomalies - Fuzzy Read(P2)

T1: read x = 50

T2: write x = 100 and commit

T1: read x = 100

T1	Rx(50)			Rx(100)
T2		Wx(100)	C	

Anomalies - Phantom(P3)

T1: read in range (a, d) -> a, b, d

T2: write c

T1 read again -> a, b, c, d

T1	R(a, b, d)			R(a, b, c, d)
T2		W(c)	C	

Anomalies - Lost Update(P4)

T2: write x = 100

T1: write x = 200

T2 x = 100 is lost

T1	Rx(50)			Wx(200)	C
T2		Wx(100)	C		

Anomalies - Cursor Lost(P4C)

T1: Cursor Read x = 50

T2: write x = 100

T1: write x = 200

T2 x = 100 is lost

T1	RCx(50)			Wx(200)	C
T2		Wx(100)	C		

Anomalies - Read Skew(A5A)

x = y = 50, must ensure: x + y = 100

T2: x transfers 40 to y

T1: reads y = 90, 50 + 90 != 100

T1	Rx(50)				Ry(90)
T2		Wx(10)	Wy(90)	C	

Anomalies - Write Skew(A5B)

x = y = 50, precondition: x + y = 100

T1: write x = 60

T2: write y = 60

T1	Rx(50)	Ry(50)	Wx(60)	
T2	Rx(50)	Ry(50)		Wy(60)

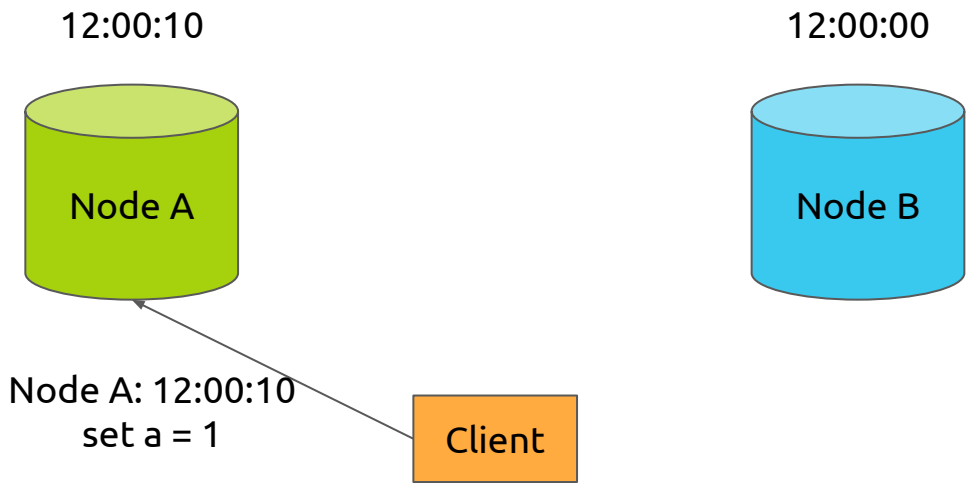
	P0	P1	P2	P3	P4C	P4	A5A	A5B
Read Uncommitted	NP	P	P	P	P	P	P	P
Read Committed	NP	NP	P	P	P	P	P	P
Cursor Stability	NP	NP	SP	P	NP	SP	P	SP
Repeated Read	NP	NP	NP	P	NP	NP	NP	NP
Snapshot	NP	NP	NP	SP	NP	NP	NP	P
Serializable	NP	NP	NP	NP	NP	NP	NP	NP

NP: Not Possible
P: Possible
SP: Sometimes Possible

For distributed transaction
Serializable is not enough
We need to care more...

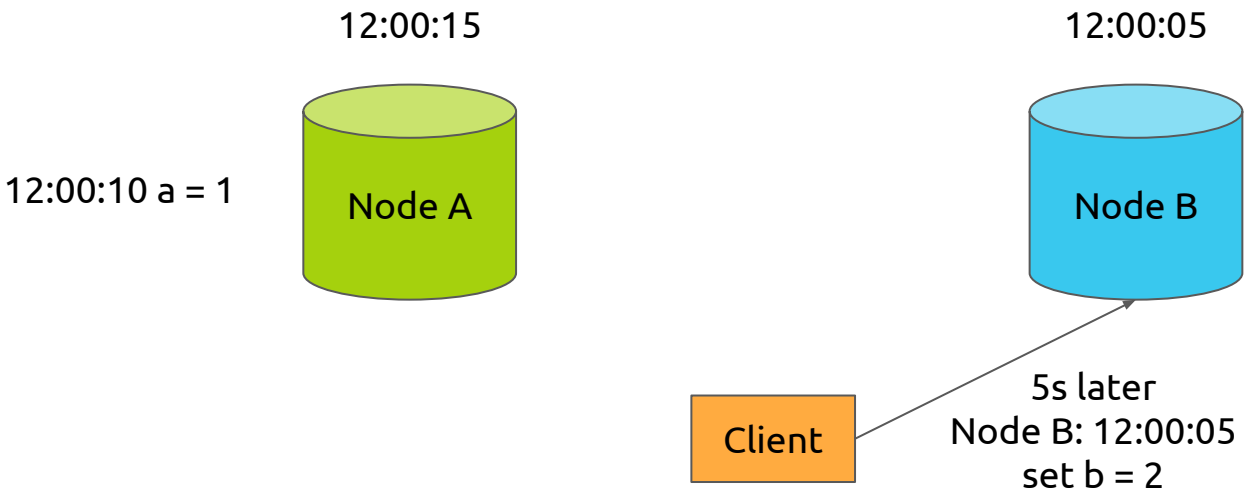
Time is important...

Node A's clock is 10s faster than Node B



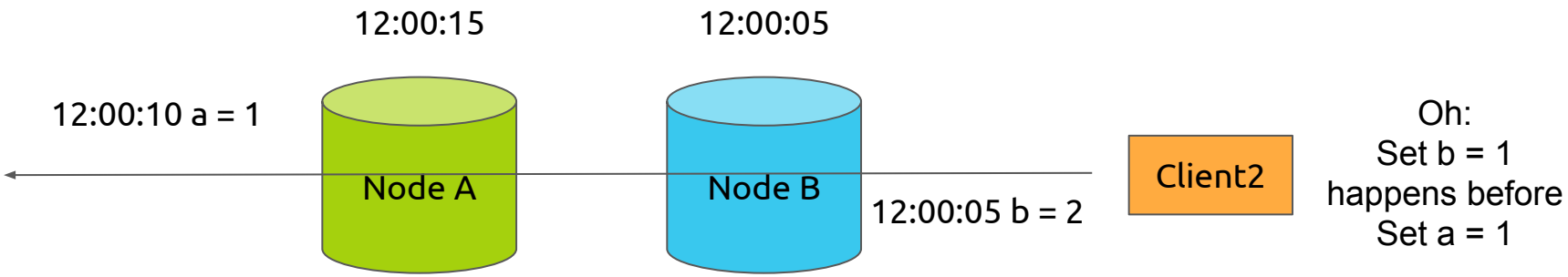
Time is important...

Node A's clock is 10s faster than Node B



Time is important...

Node A's clock is 10s faster than Node B



Time

- TrueTime
- HLC
- Timestamp Oracle

PingCAP

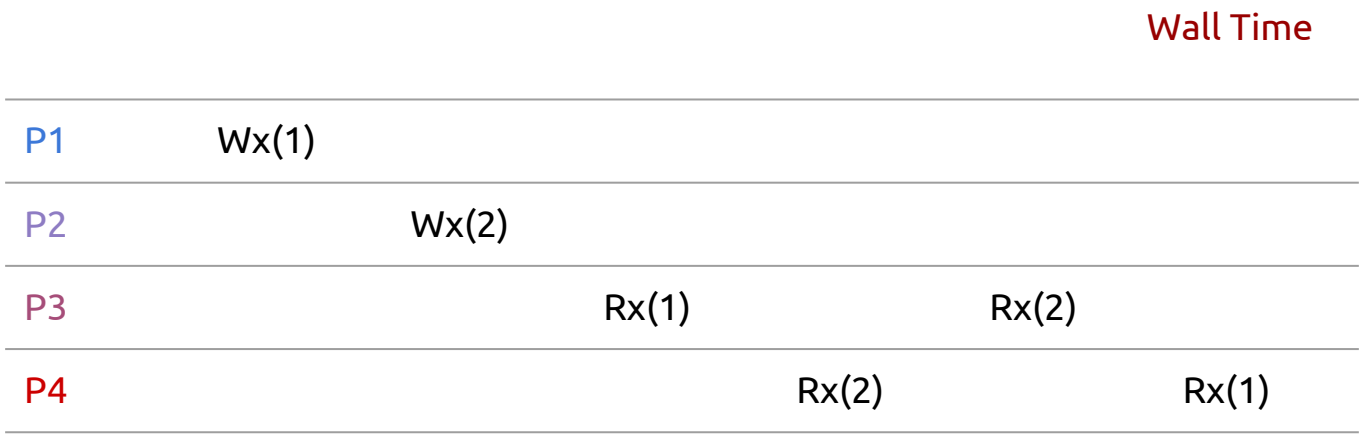
Consistency Model

Basic

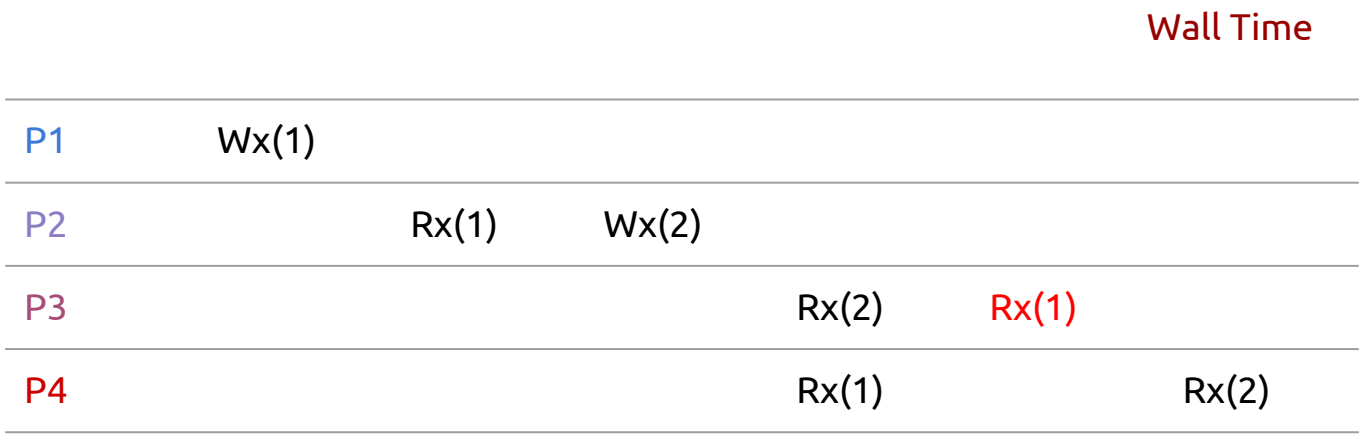
- Writes Follow Reads
- Monotonic Reads
- Monotonic Writes
- Read Your Writes

PRAM(Pipeline Random Access Memory)

Causal



Causal



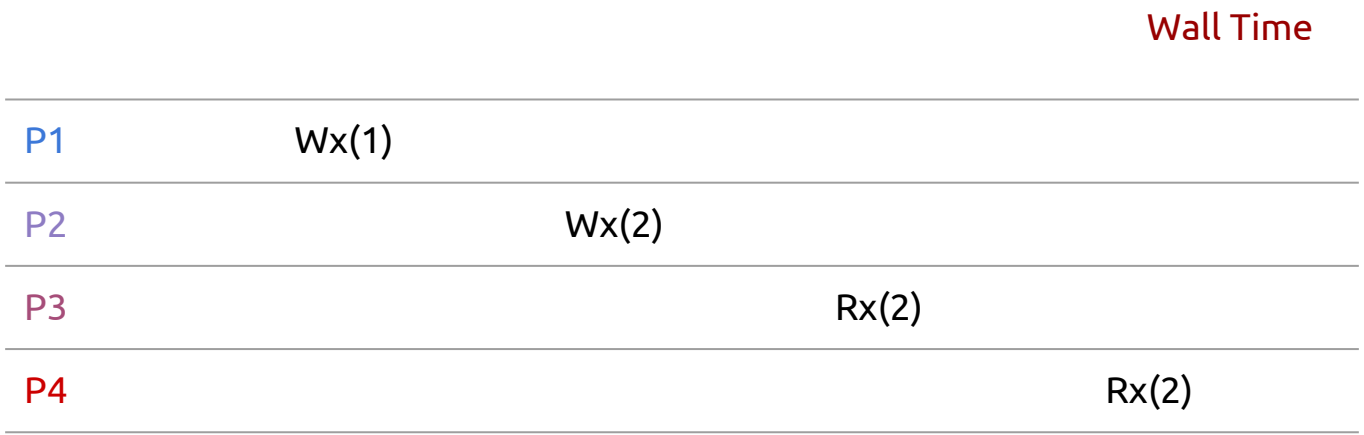
Sequential

					Wall Time
P1	Wx(1)				
P2		Wx(2)			
P3			Rx(1)		Rx(2)
P4				Rx(2)	Rx(2)

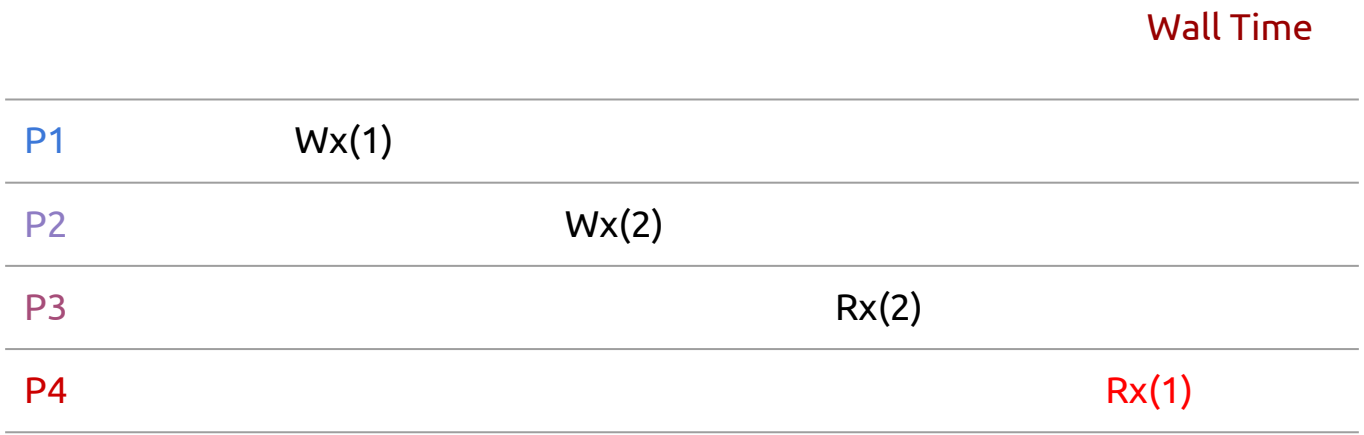
Sequential

					Wall Time
P1	Wx(1)				
P2		Wx(2)			
P3			Rx(2)		Rx(1)
P4				Rx(2)	Rx(2)

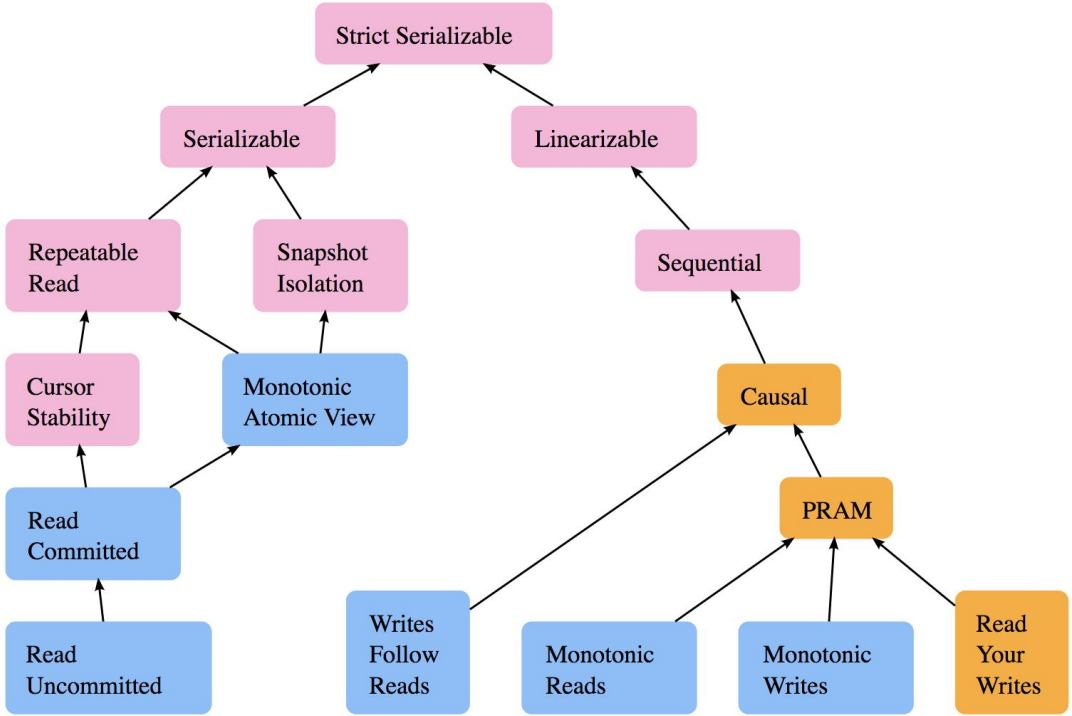
Linearizable



Linearizable



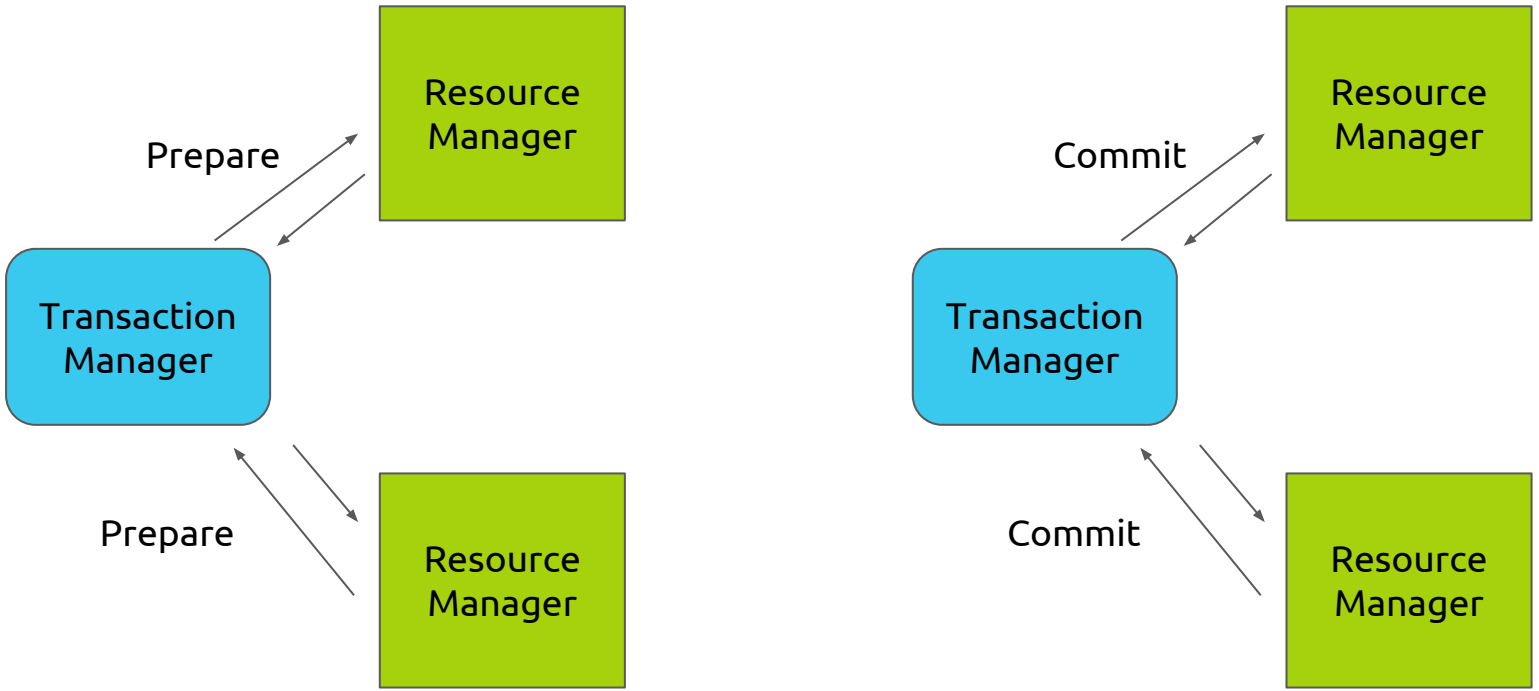
Whole Picture



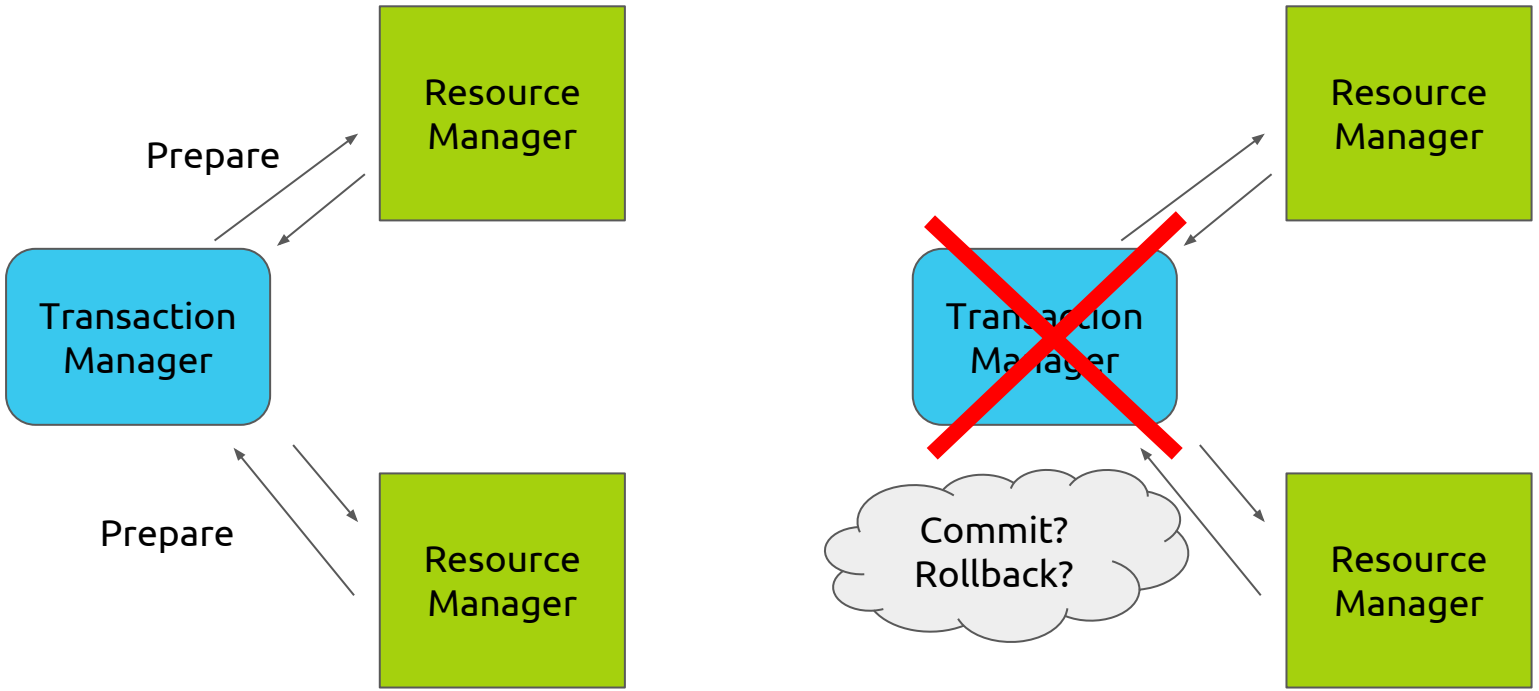
PingCAP

Practice

Two-Phase Commit



Two-Phase Commit



Percolator

- Optimized two phase commit
- MVCC
- Optimistic Commit
- Snapshot Isolation
- Timestamp Oracle

Percolator

- 3 Column Families
  - Data: key_startTs -> value
  - Lock: key -> meta
  - Write: key_commitTs -> startTs
- Primary lock / Secondary locks
- Prewrite + Commit

Transaction Model

Bob wants to transfer 7$ to Alice

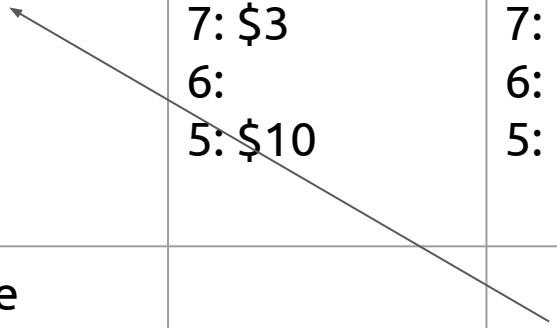
Key	Bal: Data	Bal: Lock	Bal: Write
Bob	6: 5: $10	6: 5:	6: data @ 5 5:
Alice	6: 5: $2	6: 5:	6: data @ 5 5:

Transaction Model

Key	Bal: Data	Bal: Lock	Bal: Write
Bob	7: $3 6: 5: $10	7: I am Primary 6: 5:	7: 6: data @ 5 5:
Alice	6: 5: $2	6: 5:	6: data @ 5 5:

Transaction Model

Key	Bal: Data	Bal: Lock	Bal: Write
Bob	7: $3 6: 5: $10	7: I am Primary 6: 5:	7: 6: data @ 5 5:
Alice	7: $9 6: 5: $2	7:Primary@Bob.bal 6: 5:	7: 6: data @ 5 5:



Transaction Model (commit point)

Key	Bal: Data	Bal: Lock	Bal: Write
Bob	8: 7: $3 6: 5: $10	8: 7: ~~I am Primary~~ 6: 5:	8: data @ 7 7: 6: data @ 5 5:
Alice	8: 7: $9 6: 5: $2	8: 7:Primary@Bob 6: 5:	8: 7: 6: data @ 5 5:

Transaction Model

Key	Bal: Data	Bal: Lock	Bal: Write
Bob	8: 7: $3 6: 5: $10	8: 7: 6: 5:	8: data @ 7 7: 6: data @ 5 5:
Alice	8: 7: $9 6: 5: $2	8: 7:~~Primary@Bob~~ 6: 5:	8: data @ 7 7: 6: data @ 5 5:

Transaction Model

Key	Bal: Data	Bal: Lock	Bal: Write
Bob	8: 7: $3 6: 5: $10	8: 7: 6: 5:	8: data @ 7 7: 6: data @ 5 5:
Alice	8: 7: $9 6: 5: $2	8: 7: 6: 5:	8: data @ 7 7: 6: data @ 5 5:

Thank You !

源创会 No.82 TiDB 交流群

该二维码7天内(11月19日前)有效，重新进入将更新