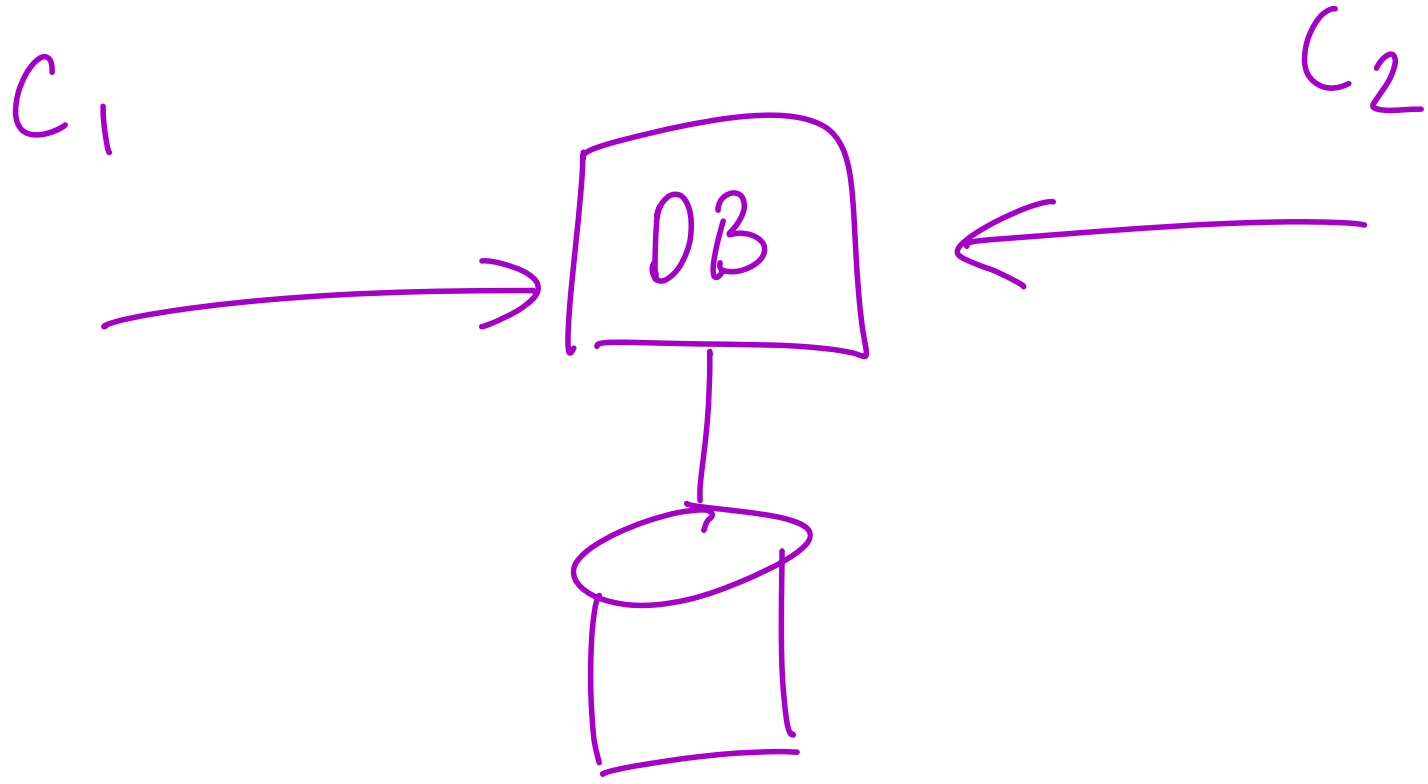
CSE P 552

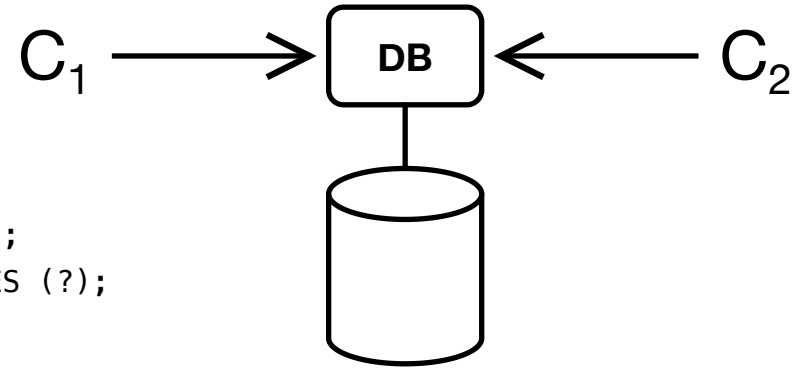
Distributed Systems

Week 8



Transaction = group of operations

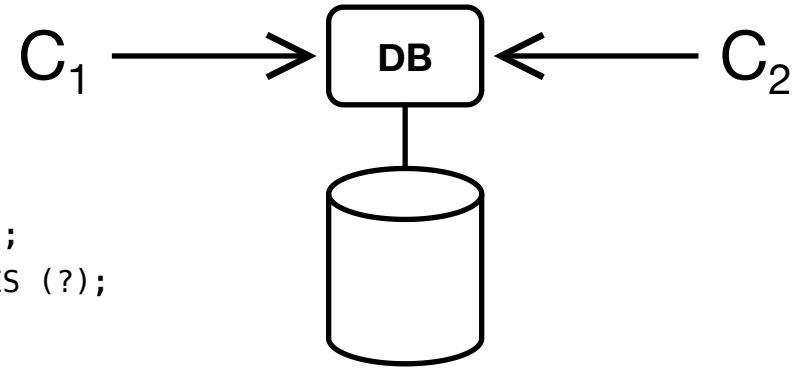
Lab 4.3: Distributed Transactions




```
BEGIN;
SELECT tix FROM events;
UPDATE events SET tix = ?;
INSERT INTO holders VALUES (?);
COMMIT;
```

```
BEGIN;
SELECT tix FROM events;
ROLLBACK;  // aka ABORT
```

```
boolean buy(DB db, String email) {
  db.execute("BEGIN");
  int tix = db.query("SELECT tix FROM events");
  if (tix > 0) {
    db.execute("UPDATE events SET tix = ?", tix - 1);
    db.execute("INSERT INTO holders VALUES (?)", email);
    db.execute("COMMIT");
    return true;
  } else {
    db.execute("ROLLBACK");  // aka ABORT
    return false;
  }
}
```




```
BEGIN;
SELECT tix FROM events;
UPDATE events SET tix = ?;
INSERT INTO holders VALUES (?);
COMMIT;
```

```
BEGIN;
SELECT tix FROM events;
UPDATE events SET tix = ?;
INSERT INTO holders VALUES (?);
COMMIT;
```

```
boolean buy(DB db, String email) {
  db.execute("BEGIN");
  int tix = db.query("SELECT tix FROM events");
  if (tix > 0) {
    db.execute("UPDATE events SET tix = ?", tix - 1);
    db.execute("INSERT INTO holders VALUES (?)", email);
    db.execute("COMMIT");
    return true;
  } else {
    db.execute("ROLLBACK");  // aka ABORT
    return false;
  }
}
```

events

tix
1

C_1

```
BEGIN;

SELECT tix FROM events;  → 1

UPDATE events SET tix = ?;
INSERT INTO holders VALUES (?);
COMMIT;
```

C_2

```
BEGIN;

SELECT tix FROM events;  → 1

UPDATE events SET tix = ?;
INSERT INTO holders VALUES (?);
COMMIT;
```

Concurrency Control

- preventing concurrent txns from interfering w/ each other

Properties of txns

Atomic — all or nothing
Consistent — invariant maintenance
Isolated — concurrency control
Durability — commit ⇒ data safe on disk

2 phase Locking (2PL)

- spec: Serializability (S)
  - equivalent to a serial execution
    - Serial = all ops of a txn are next to eachother in the history
      = interleaving of txns not just ops
- vs linearizability (L)
  - S is about groups of ops, L is single ops

- L has "real time" constraint

- Strict Serializability = S + real time constraint

2PL

- read an object: acquire a read lock on that object
- write ——— write lock
- hold locks til end of txn
  - acquire phase + release phase

events

tix
1

C_1

```
BEGIN;

SELECT tix FROM events;  → 1

UPDATE events SET tix = ?;
INSERT INTO holders VALUES (?);
COMMIT;
```

C_2

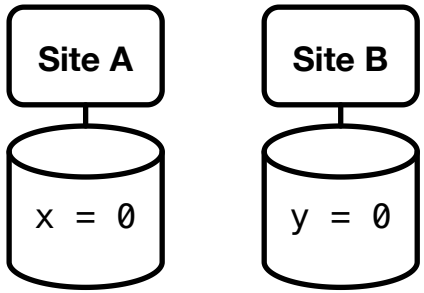
```
BEGIN;

SELECT tix FROM events;  → 1

UPDATE events SET tix = ?;
INSERT INTO holders VALUES (?);
COMMIT;
```

$r_1(tix) \rightarrow 1 \quad r_2(tix) \rightarrow 1 \quad w_1(tix, 0) \quad c_1 \quad w_2(tix, 0) \quad c_2$

$$r_1(tix) \rightarrow 1 \quad r_2(tix) \rightarrow 1 \quad w_1(tix, 0) \quad c_1 \quad w_2(tix, 0) \quad c_2$$

C_1

```
BEGIN;
UPDATE kv SET v = 1 WHERE k = 'x';
UPDATE kv SET v = 1 WHERE k = 'y';
COMMIT;
```



C_2

```
BEGIN;
SELECT v FROM kv WHERE k = 'x';
SELECT v FROM kv WHERE k = 'y';
COMMIT;
```

Lab 4 part 2 design: small groups, 20 minutes

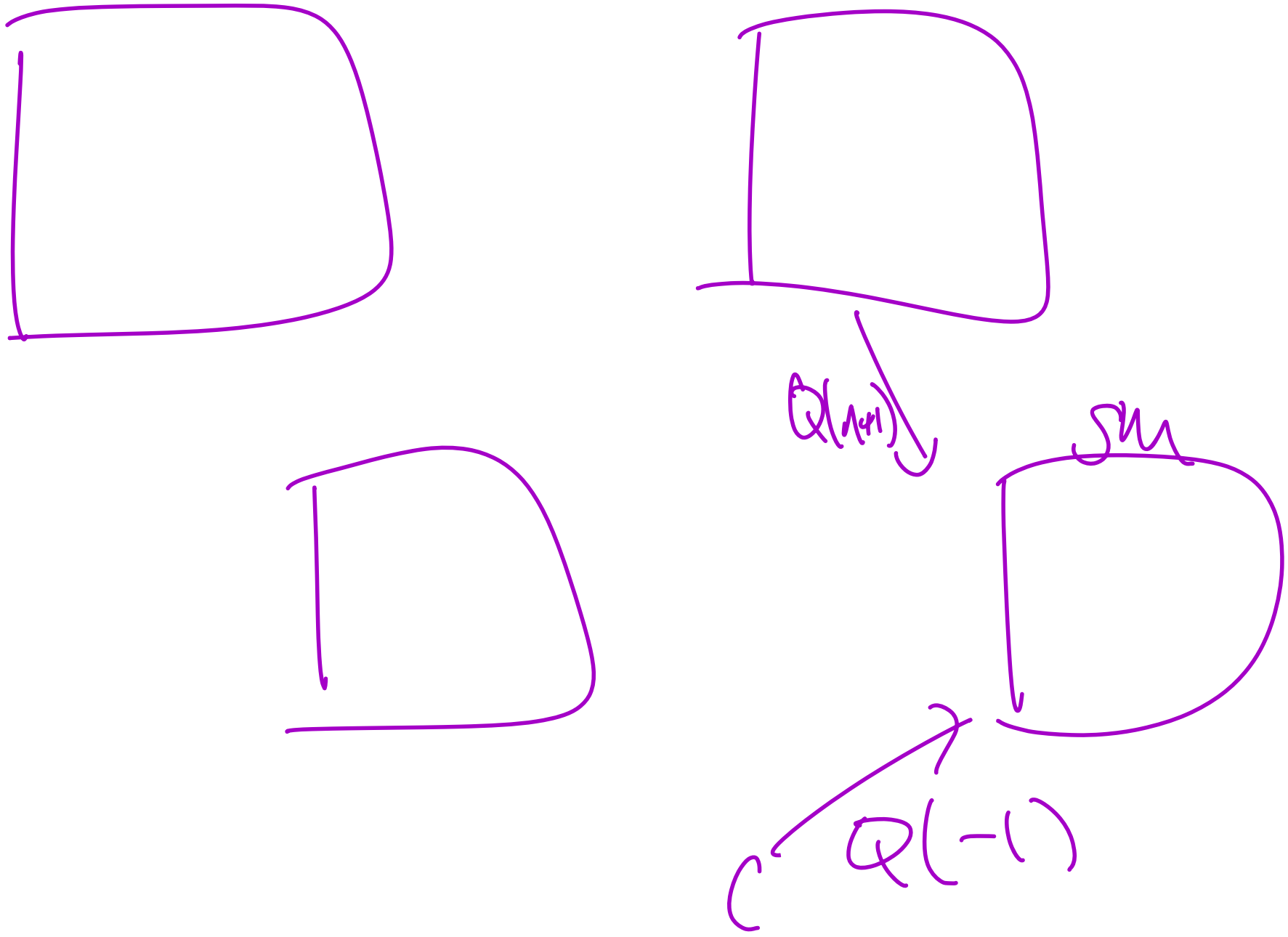
Groups of 3 or 4.

- Introduce yourselves.
- Share which questions you addressed. Any particularly nice insights?
- Share any remaining things you are unsure about.

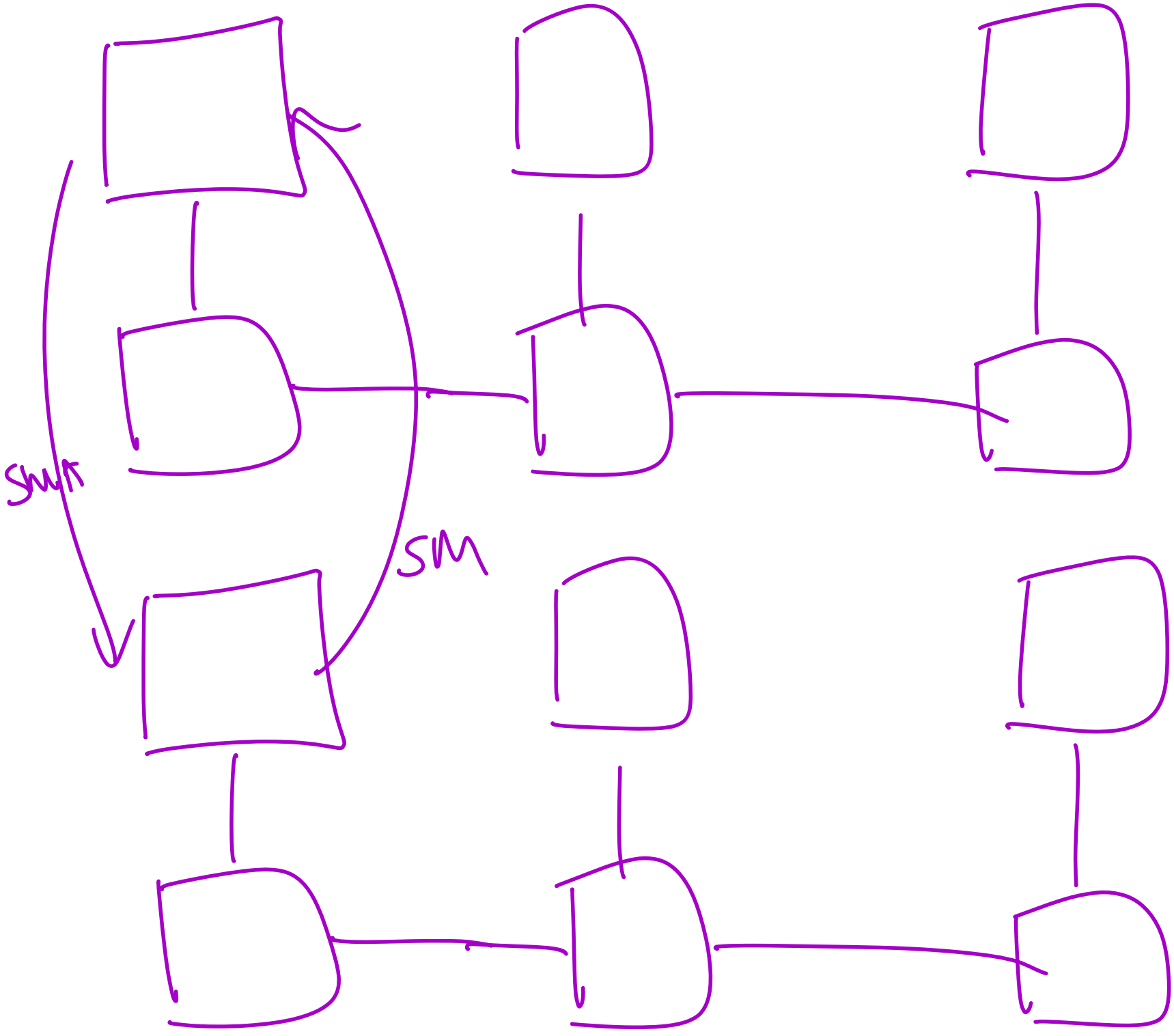
Discussion questions if you run out:

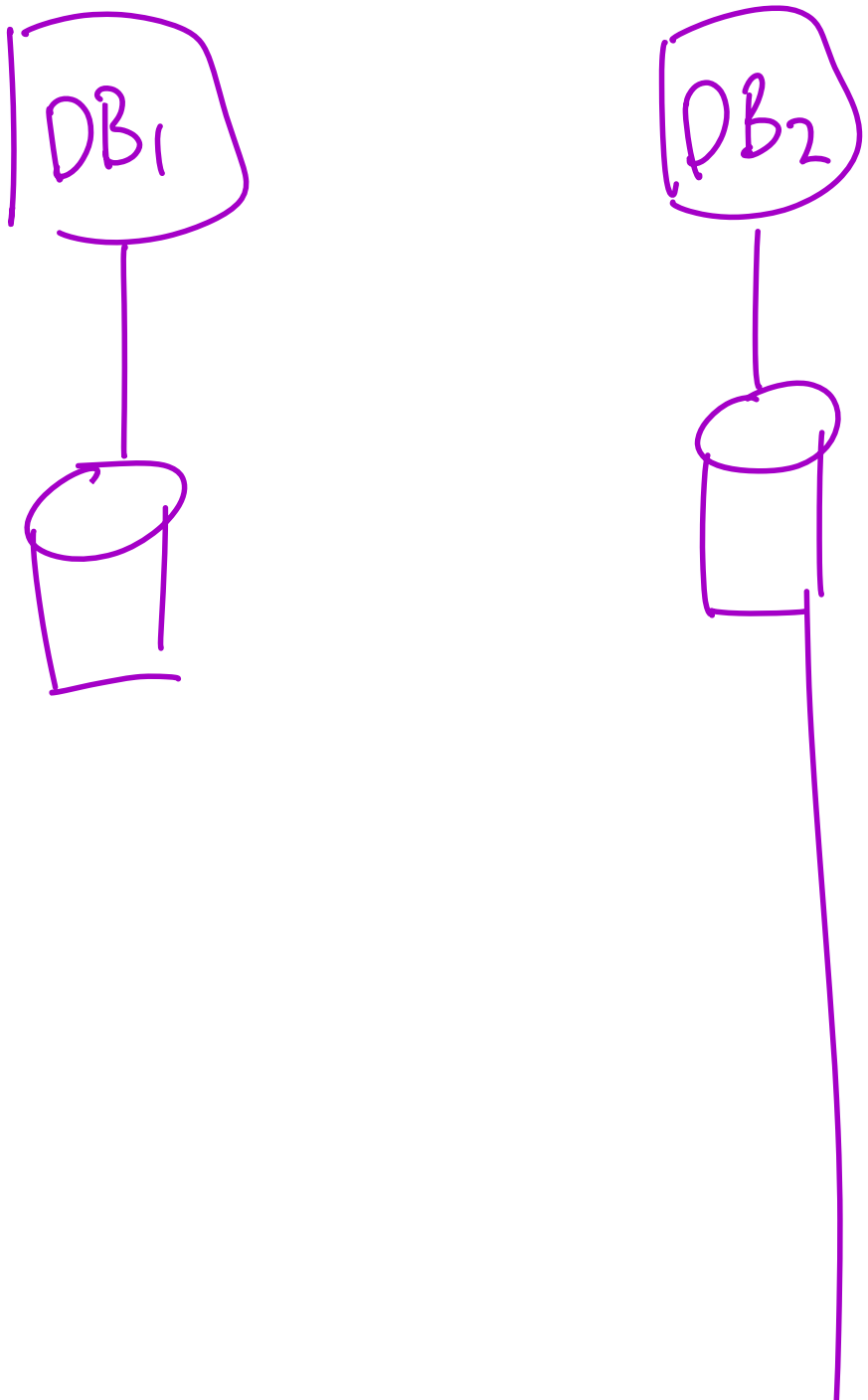
1. During reconfig, can we serve requests to keys that are not moving?
2. What actions are (not) safe for a group to take without using Paxos?
3. Is it possible a KV-operation is chosen during reconfiguration for a key that is moving? How to prevent this or handle this?
4. Suppose the Paxos leader for a group changes. What actions does the new leader need to takeover from the old leader?
5. Anything else you think is interesting!

We'll debrief as a class.


O(1,1)
SM
C
O(-1)

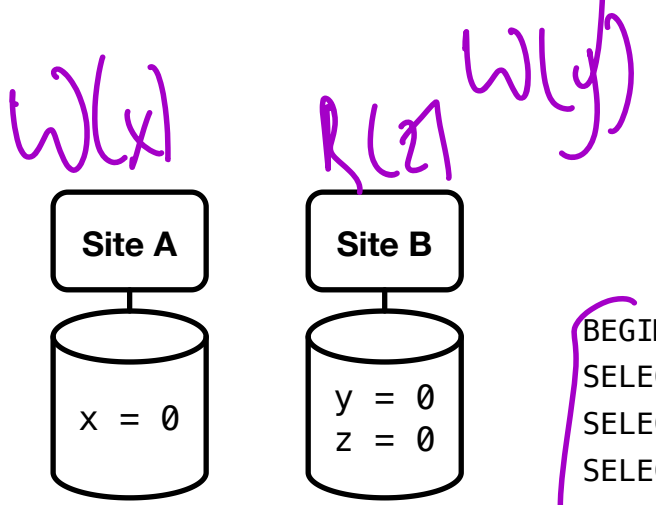

id ...
config(10, ...)
SM


SM
SM


DB1
DB2

C_1

```
BEGIN;
UPDATE kv SET v = 1 WHERE k = 'x';
UPDATE kv SET v = 1 WHERE k = 'y';
COMMIT;
```

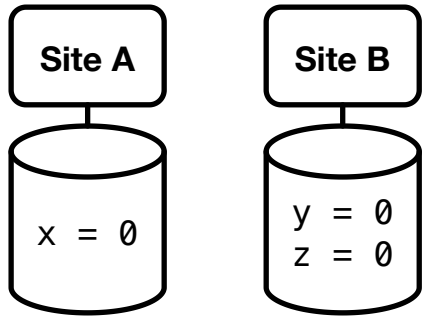


C_2

```
BEGIN;
SELECT v FROM kv WHERE k = 'z';
SELECT v FROM kv WHERE k = 'x';
SELECT v FROM kv WHERE k = 'y';
COMMIT;
```

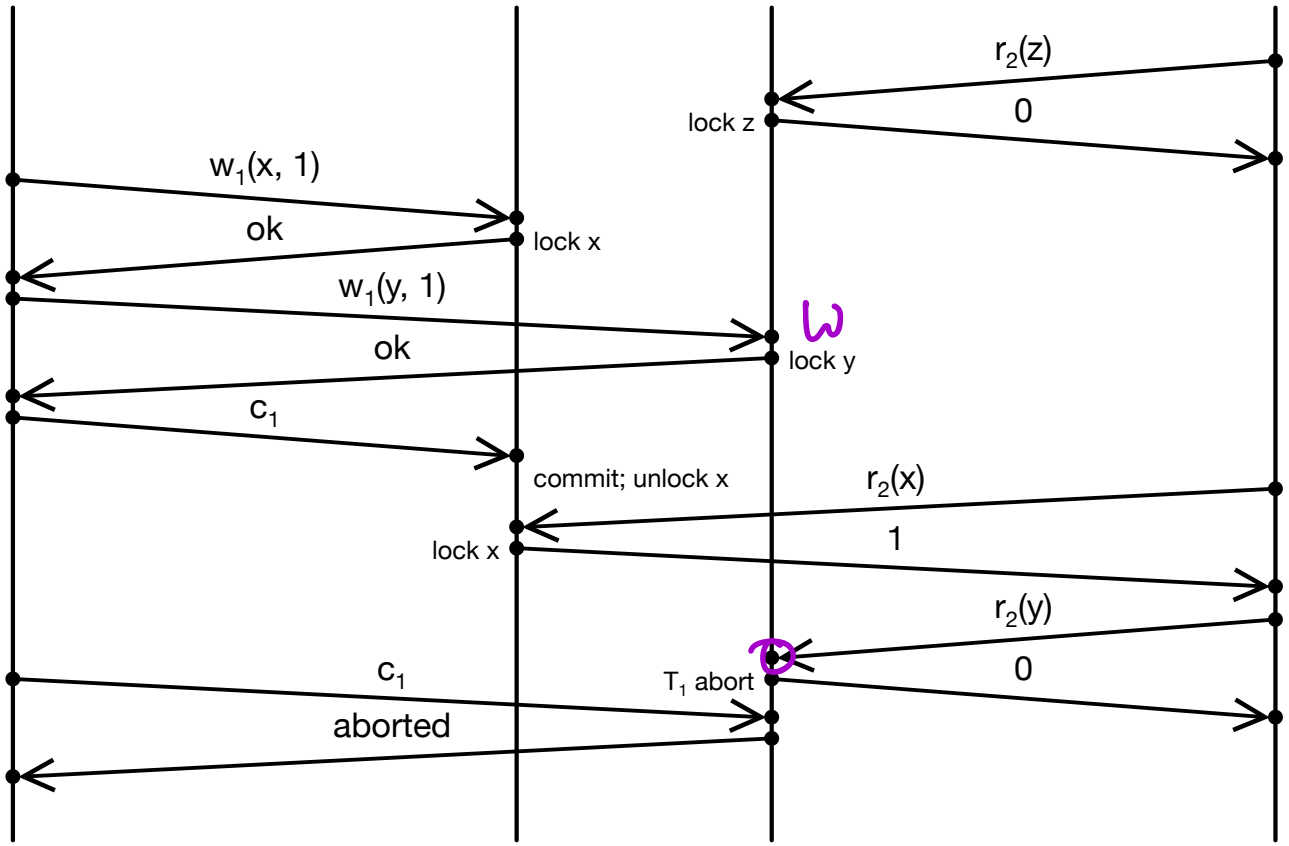
C_1

```
BEGIN;
UPDATE kv SET v = 1 WHERE k = 'x';
UPDATE kv SET v = 1 WHERE k = 'y';
COMMIT;
```



C_2

```
BEGIN;
SELECT v FROM kv WHERE k = 'z';
SELECT v FROM kv WHERE k = 'x';
SELECT v FROM kv WHERE k = 'y';
COMMIT;
```



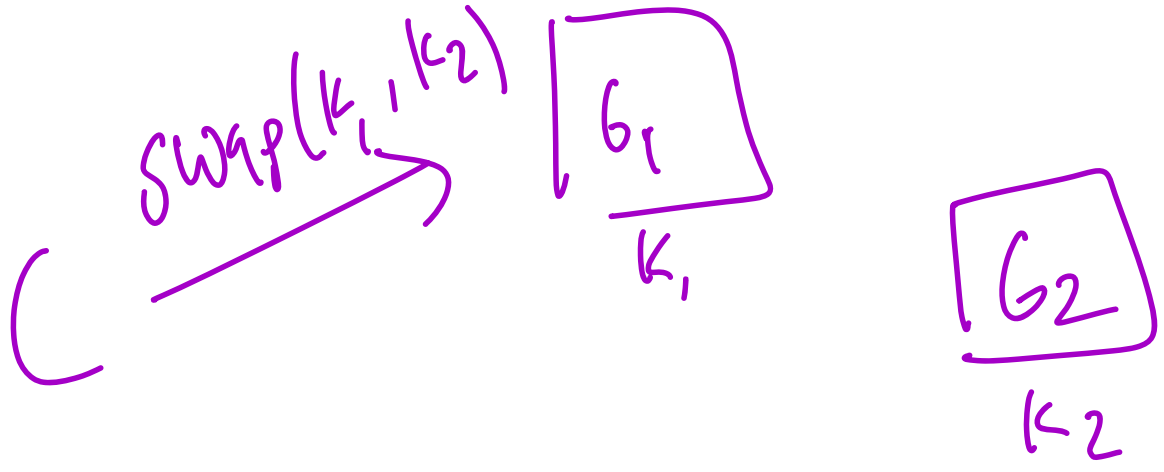
x = 1, y = 0 for good

2 phase commit (2PC)

- "participants" — sites | groups relevant to txn (have data)
- "coordinator" — distinguished group in charge of the txn
  - also a participant

Lab 4.3 ops

- MultiGet(Set<Key>)
- MultiPut(Map<Key, Value>)
- Swap(k_1, k_2)



Swap(K_1, K_2)

$K_1 \mapsto G_1$

$K_2 \mapsto G_2$

G_1 Coord

