Distributed transaction

T_1: Transfer

```
x = GET(x)
y = GET(y)
PUT(x, x-10)
PUT(Y, y+10)
```

T_2: Audit

```
x = GET(x)
y = GET(y)
print(x+y)
```

ACID guarantees:

- Atomicity: All parts of txn execute or none (x's balance decreases, y's balance does not increase)
- Consistency: Preserves invariants. (eg. x's balance >0)
- Durability: Txn's effect are not lost (even if servers restart)

Isolation

T_1: Transfer

```
x = GET(x)
y = GET(y)
PUT(x, x-10)
PUT(Y, y+10)
```

→ what if T_2 executes here? Prints 190

T_2: Audit

```
x = GET(x)
y = GET(y)
print(x+y)
```

what if T_1 executes here? Prints 210

- Serializability

T_1 T_2 x:90 y:110 P200

or

T_2 T_1

Strict serializability

If T_2 started after T_1 committed, then.

T_1 T_2

- T_2 Stale Read

Rx100 Ry100 P200

T_2 Rx100 Ry100 Wx90 Wy110

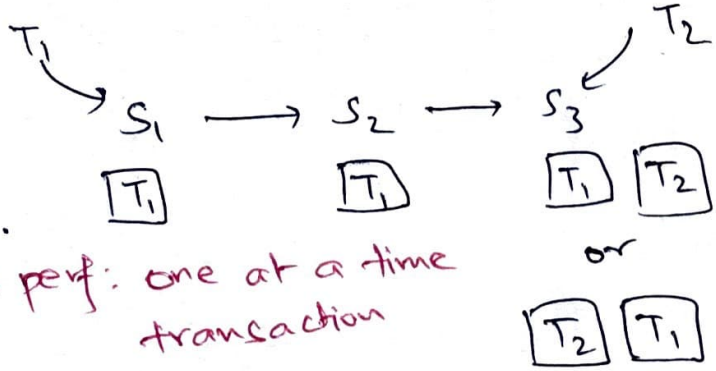
Serializable but not linearizable.

T_2 Rx90 Ry100 P190

T_1 Rx100 Ry100 Wx90 Wy110

Linearizable but not serializable

- Chain Replication?



Bad perf: one at a time transaction

Challenge: data does not fit on a single machine (eg. X and Y)

- Sharding

$A_1 \rightarrow A_2 \rightarrow A_3$ M-X	$B_1 \rightarrow B_2 \rightarrow B_3$ Y-L

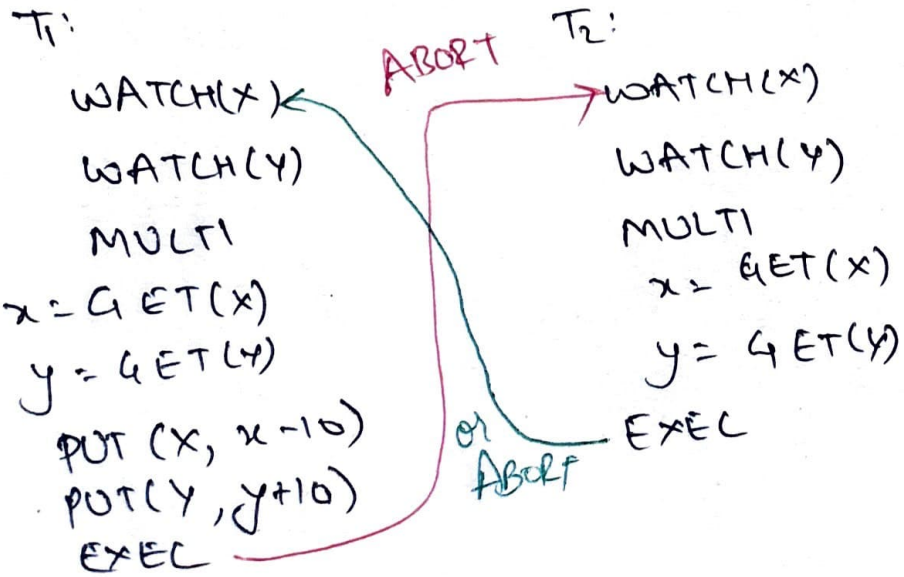
- Might also shard for performance.

Transfer from M→N, F→G can happen in parallel on separate machines.

- Isolation → Concurrency control (optimistic, pessimistic, multi version)

  Atomicity → Atomic commit (2 phase commits)

- Optimistic concurrency control (Redis)



- Pessimistic concurrency control

T_1	T_2
rlock (X,Y)	rlock (X,Y)
x = GET(X)	x = GET(X)
y = GET (Y)	y = GET (Y)
wlock (X,Y)	release (X,Y)
PUT (X, x+10)	Print (x+y)
PUT (Y, y-10)	
release (X,Y)	

- optimistic
  - If high conflicts, keep aborting and restarting

- Perscimistic
  - unnecessary locking if no conflicts
  - Deadlock avoidance is needed

- Wound wait deadlock avoidance.

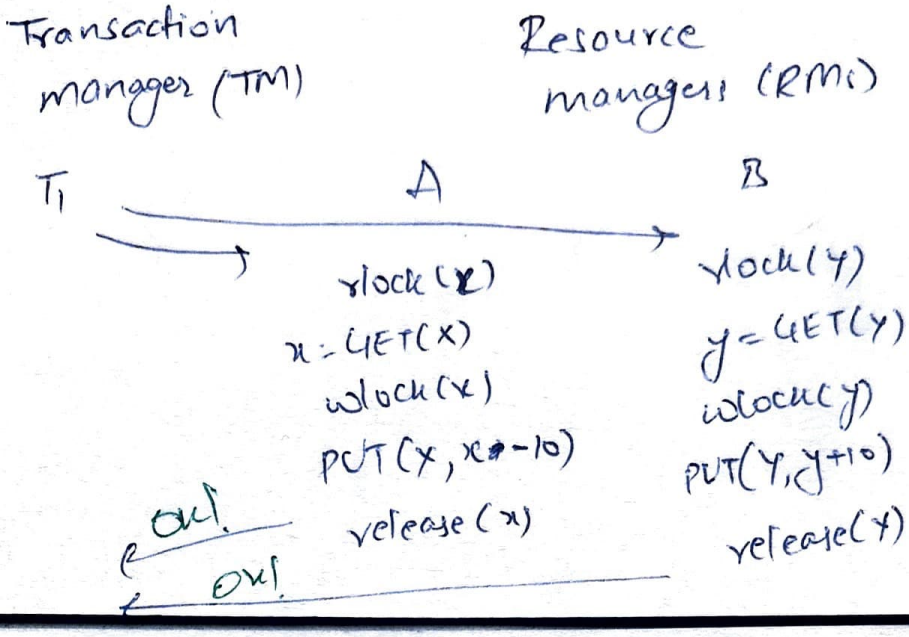
T_1: rlock (X,Y) ← T_2: rlock (X,Y)

wlock (X,Y) → deadlock! ← wlock (X,Y)

release (X,Y) release (X,Y)

First to wlock, aborts other transaction.

- Bad commit protocol



- what can go wrong?
  - Not enough money in x
  - y account does not exist
  - A or B crashes before receiving msgs
  - Network fails
  - TM crashes after sending txn to A but before " " " B

- Safety?

Atomic commits: Everyone commits or everyone aborts.

Keep aborting forever?

- Liveness
  - If no failures, A, B can commit, then commit
  - If failures, reach a conclusion ASAP

- R/W transactions can be thought of as 2 phases

```
T1: rlock(x,y)
    x = GET(x)
    assert(x>10)
    y = GET(y)
    wlock(x,y)
    PUT(x, x+10)
    PUT(y, y+10)
    release(x,y)
```

Prepare phase: Read all values. Take all locks. No writes!

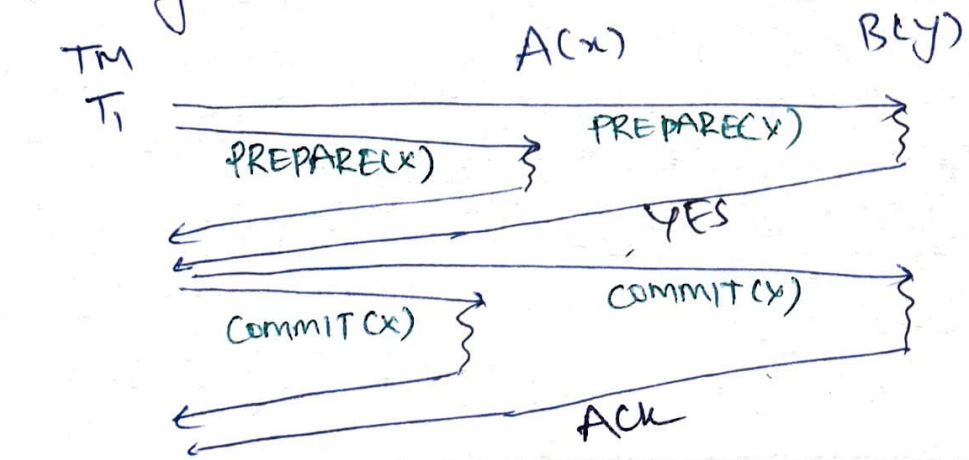
Commit phase: Write and release locks. Write both x,y or none. Must be atomic.

```
PREPARE(x):
    rlock(x)
    x = GET(x)
    assert(x>10)
    wlock(x)
    return x.

COMMIT(x)
    PUT(x, x+10)
    release(x)

ABORT(x)
    release(x)
```

- Atomicity
  Two phase Commits

Transaction manager

Resource managers



- Why does it give atomic commits?
  - TM can send commit only if it has heard Yes from all RMs.
    - All or nothing
  - Ex: If B cannot wlock(y), it replies NO
    - ⇒ TM abort transaction.

- B crashes before sending YES to TM.
  - TM timeouts and unilaterally aborts.
  - or n/w lost YES message

- B crashes after sending YES to TM
  - TM sends commit to A.
  - B restarts
    - must remember it was in middle of Txn. $PREPARE_{T_1}(y)$ i.e, wlock(y)
    - TM keeps retrying commit(T_1)
    - Is B guaranteed to get wlock(y)?

WAL $Prepare_{T_1}(y) \rightarrow$ Yes

- What if TM restarts before sending prepare to B?
  - Send prepare again.
  - B prepares
  - A should remember it was already prepared and reply YES

- What if TM restarts after prepares?
  - If participant had replied Yes, it is blocked. waiting for commit/abort
  - After restart, TM must commit/abort all pending transactions

- TM log
  - <Txn ID> <details>
  - <commit> <Txn ID>

    or

    <abort> <Txn ID>

Why is it ok to not log -

- Sent prepare to A?
  - can just resend prepares
- received yes from A?
  - can unilaterally abort
- Sent commit to A?
  - can just resend commit.

RM/
Participant log

- <Txn ID> <details> <prepared>
- " <committed>
  or <aborted>

Safety -

- No commit unless everyone says yes
- RM Cannot back out after saying Yes
  across restarts

Liveness

- Not live if TM crashes forever after prepare (or becomes unreachable)