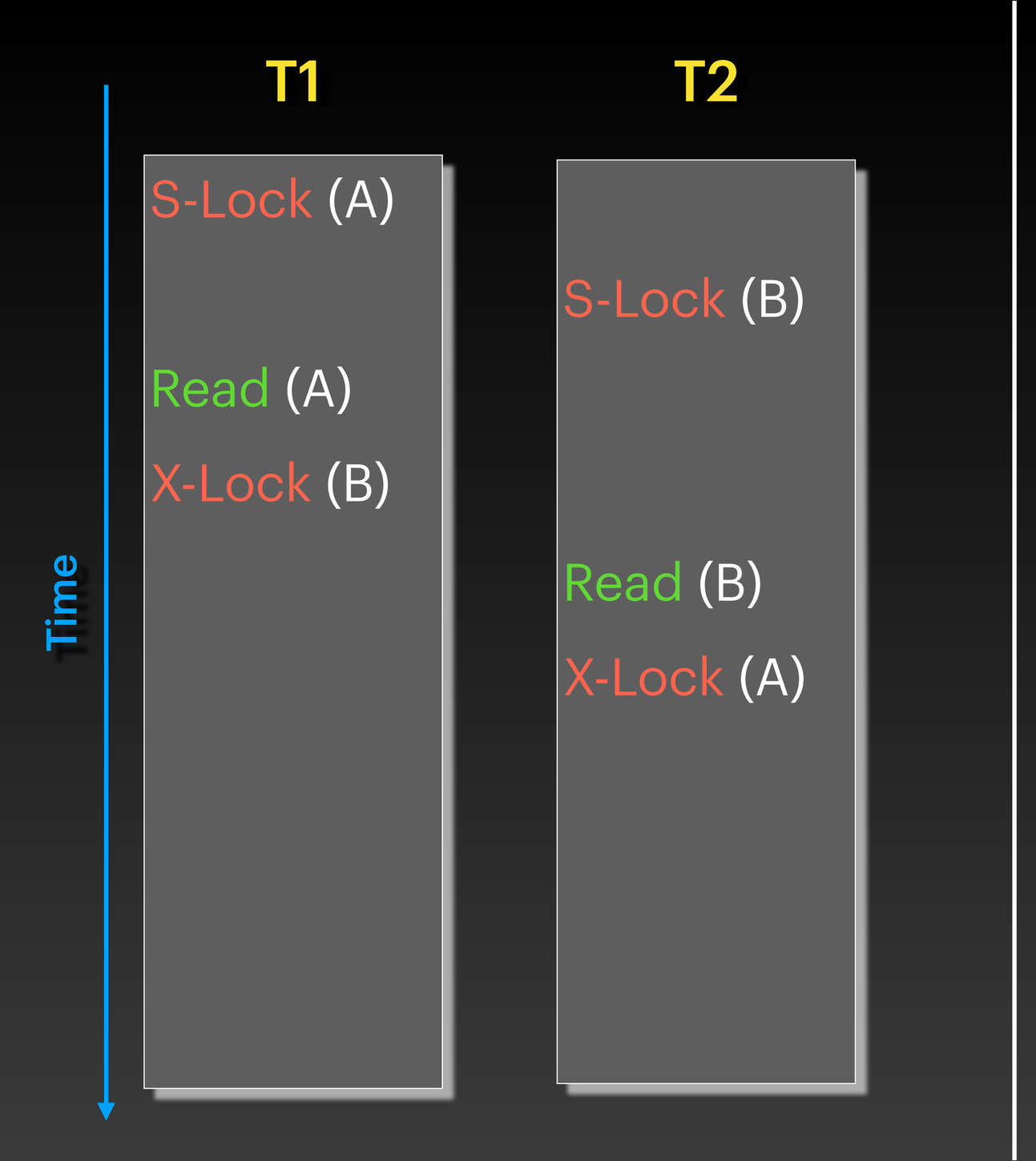
TECH VAULT

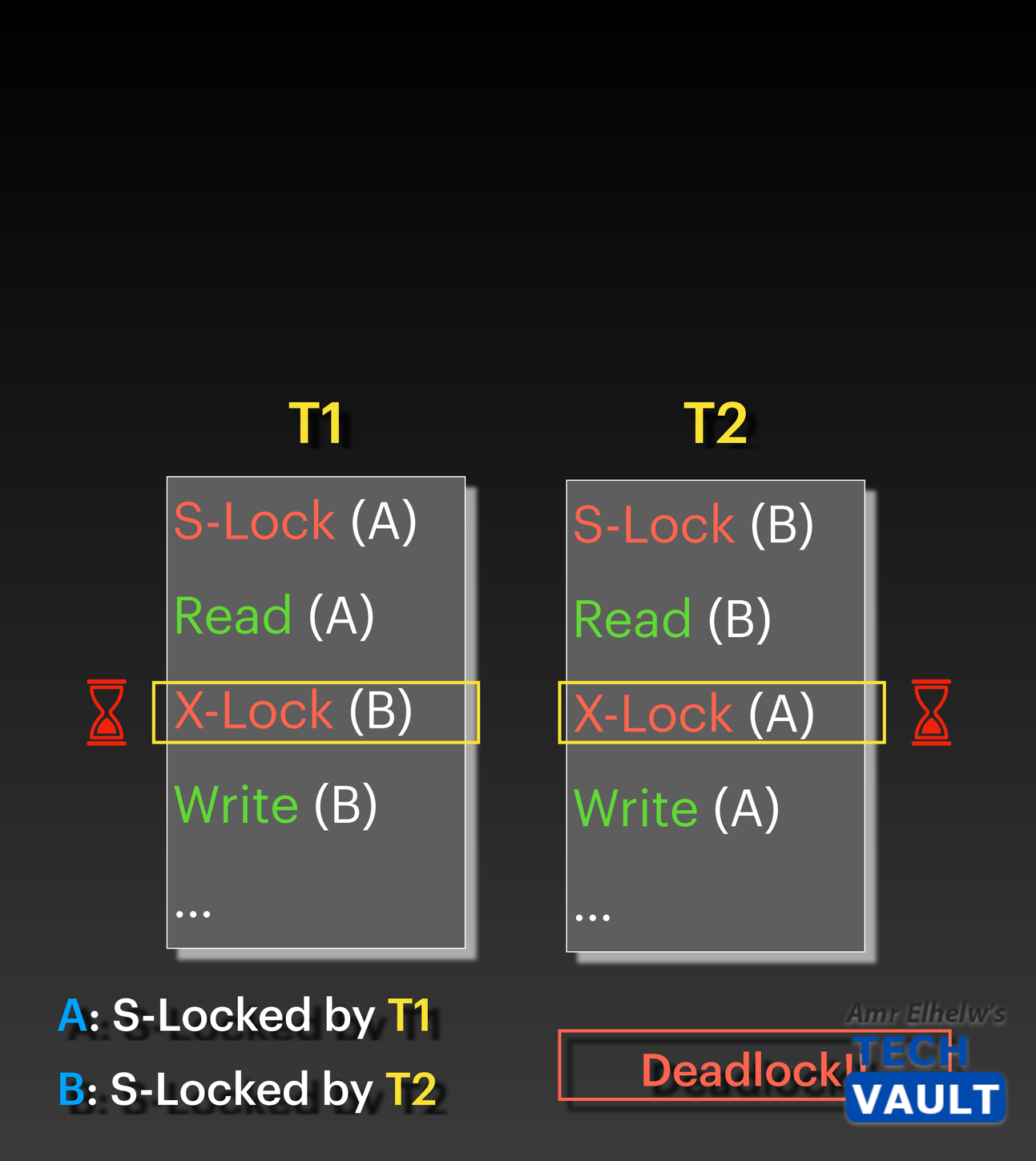
CONCURRENCY CONTROL

Two-Phase Locking

Part (2)

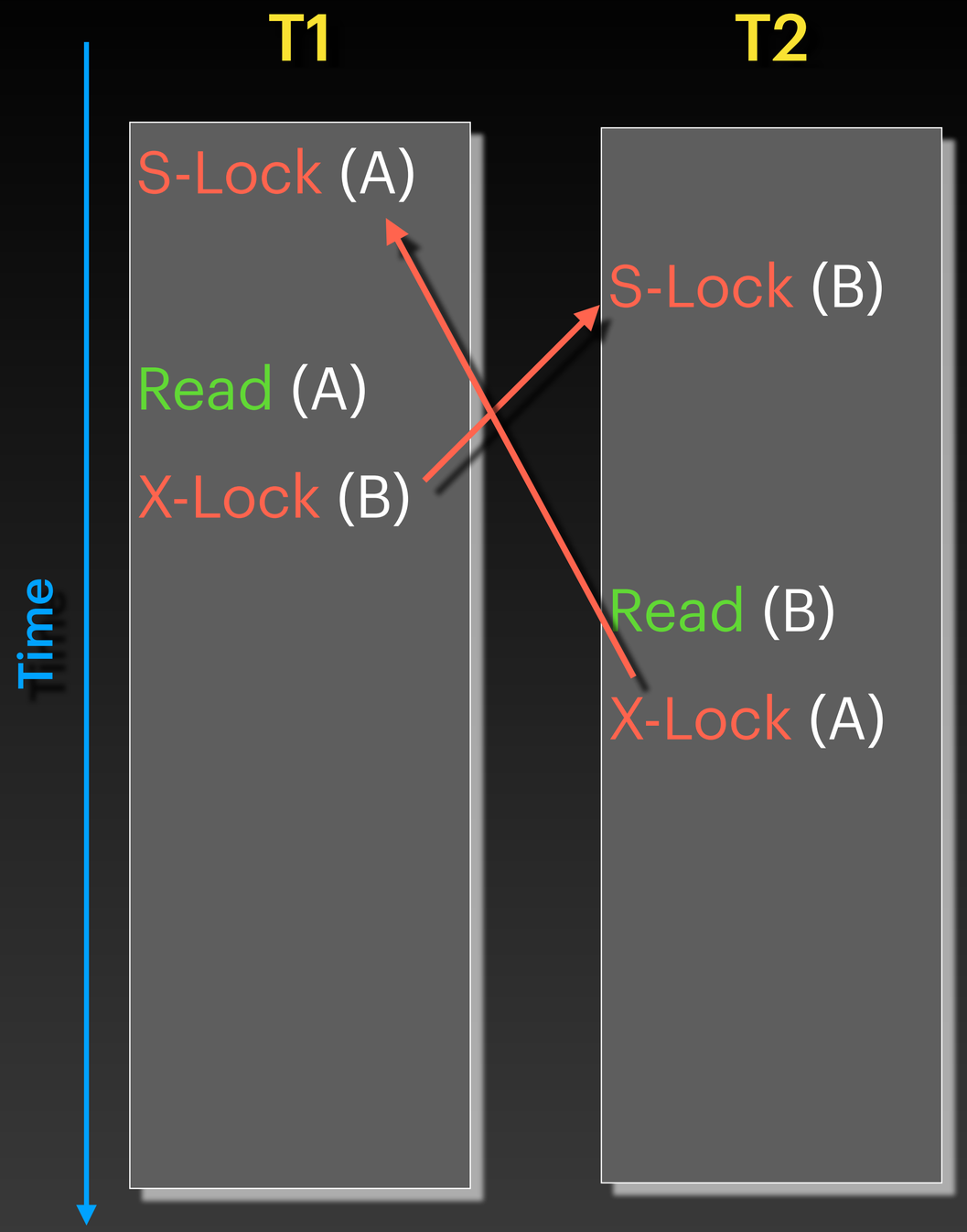
AMR ELHELW


T1
T2
S-Lock (A)
Read (A)
X-Lock (B)
S-Lock (B)
Read (B)
X-Lock (A)
Time

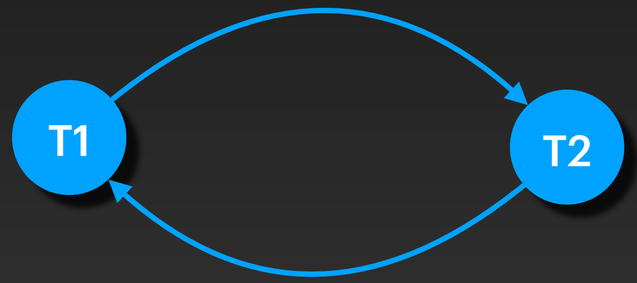

T1
T2
S-Lock (A)
Read (A)
X-Lock (B)
Write (B)
...
S-Lock (B)
Read (B)
X-Lock (A)
Write (A)
...
A: S-Locked by T1
B: S-Locked by T2
Deadlock!
Amr Elhelw's
TECH
VAULT

DeadLock Detection

- Uses “waits-for” graph
  - Each node is a transaction
  - Edge from T_i to T_j if T_i is waiting for a lock held by T_j.
  - Cycle = deadlock!!



"Waits-for" Graph



Deadlock!!

Victim Selection

Possible criteria

- Newest (by timstamp)
- Least number of writes done
- Most number of locked items
- Least number of rollbacks (avoid starvation)
- ...

DeadLock Prevention

- **Method 1:** Use Conservative 2PL
  - Transaction request all locks at start.
  - If any lock is unavailable, don’t acquire any locks, wait, then retry.

DeadLock Prevention

- **Method 2:** Timestamp-based
  - Each transaction has a timestamp *ts*
    - *T1* is “older” than *T2* if *ts(T1) < ts(T2)*
  - If a transaction requests a lock held by another transaction —> Either *block* or *abort* one of the two, based on the timestamps

DeadLock Prevention

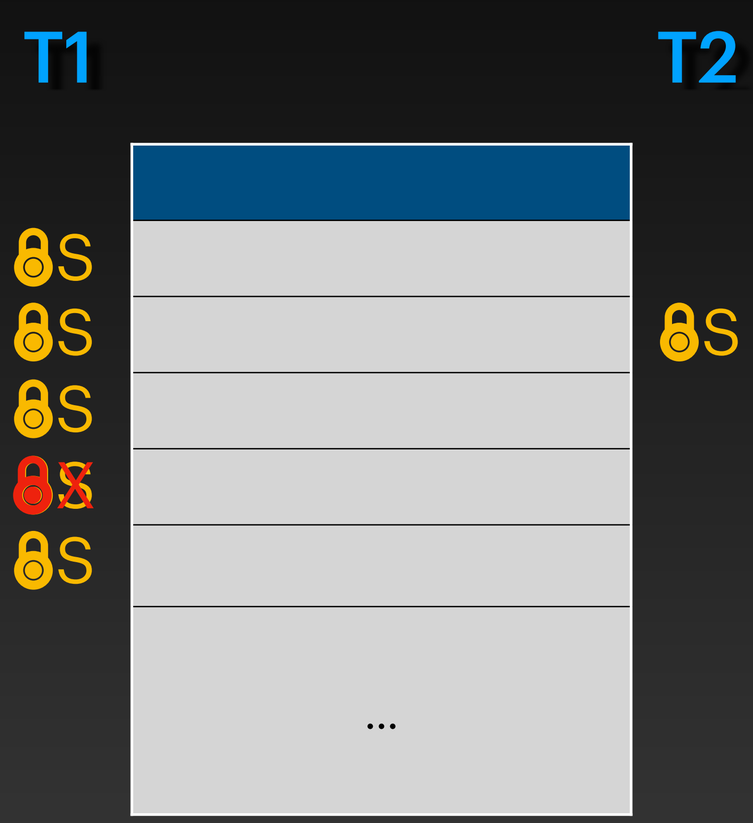
- **Method 2:** Timestamp-based
  - T_{hld} holds lock, T_{req} requests lock
  - **Wait-Die**
    - If T_{req} older than T_{hld} —> T_{req} waits, else abort T_{req}
  - **Wound-Wait**
    - If T_{req} older than T_{hld} —> abort T_{hld}, else T_{req} waits

Lock Hierarchy

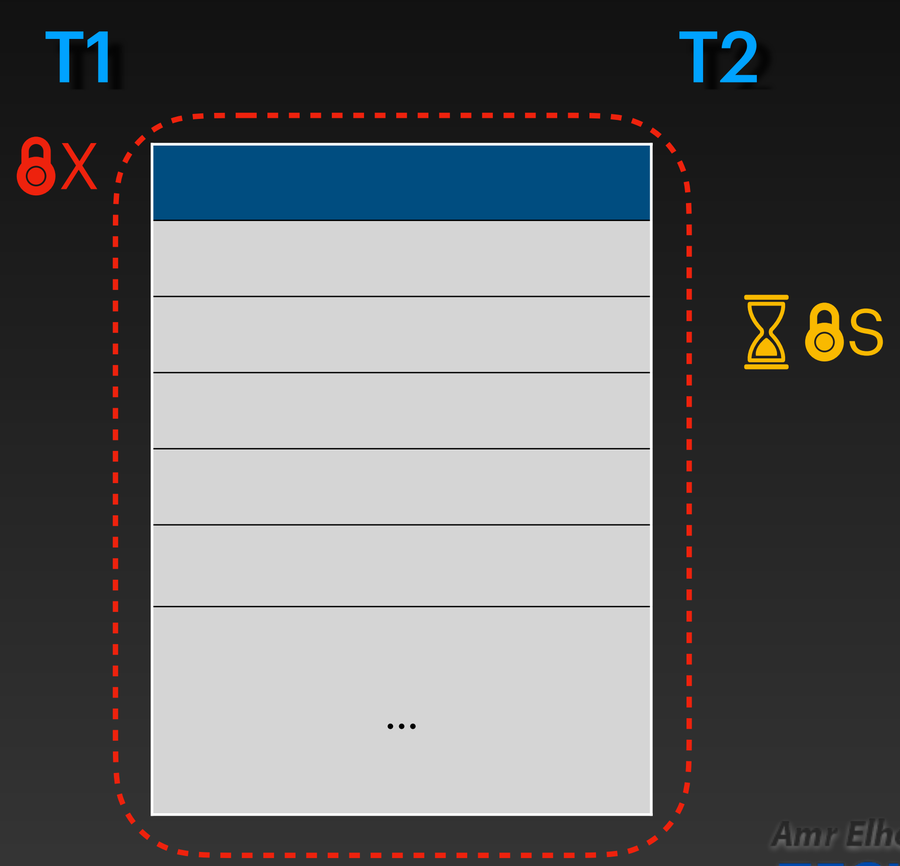
- Database (not very common)
  - Table (very common)
    - Page (common)
      - Row (very common)
        - Attribute (rare)

T1: scan all table and then update one row

T2: read a single row



Too many locks - expensive



Bad concurrency

Intent Lock

- A lock acquired on a higher level object to allow the transaction to lock a lower level object

- The DBMS (lock manager) internally manages all intent locks. The user does not have direct control over them

Intent Lock Types

- Intent Shared lock (IS)
  - Intent to get S-lock(s) at a lower level
- Intent eXclusive lock (IX)
  - Intent to get X-lock(s) at a lower level
- Shared + Intent eXclusive lock (SIX)
  - S-lock at higher level (and all its descendants) + intent to get X-lock(s) at a lower level

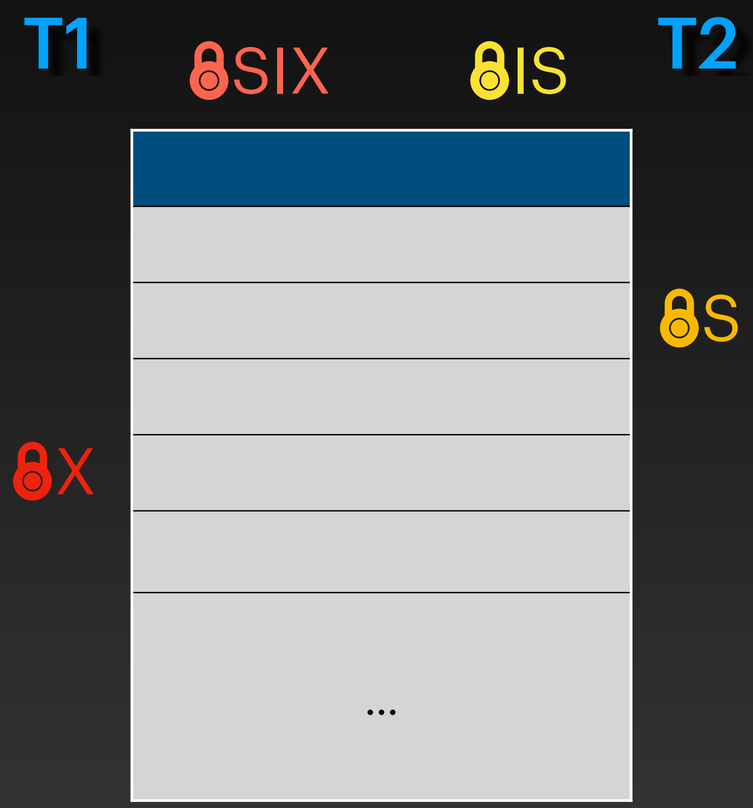
Compatibility Matrix

T2 Requests

T1 Holds	X	S	IS	IX	SIX
X	✗	✗	✗	✗	✗
S	✗	✓	✓	✗	✗
IS	✗	✓	✓	✓	✓
IX	✗	✗	✓	✓	✗
SIX	✗	✗	✓	✗	✗

T1: scan all table and then update one row

T2: read a single row



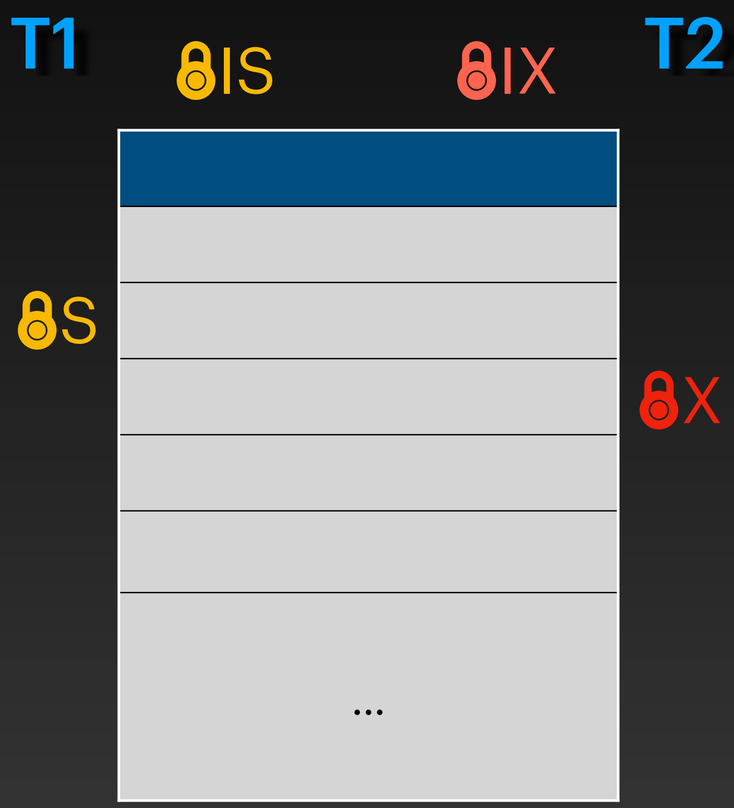
Transaction requests

Transaction holds

	X	S	IS	IX	SIX
X	✗	✗	✗	✗	✗
S	✗	✓	✓	✗	✗
IS	✗	✓	✓	✓	✓
IX	✗	✗	✓	✓	✗
SIX	✗	✗	✓	✗	✗

T1: read a single row

T2: update a single (different) row



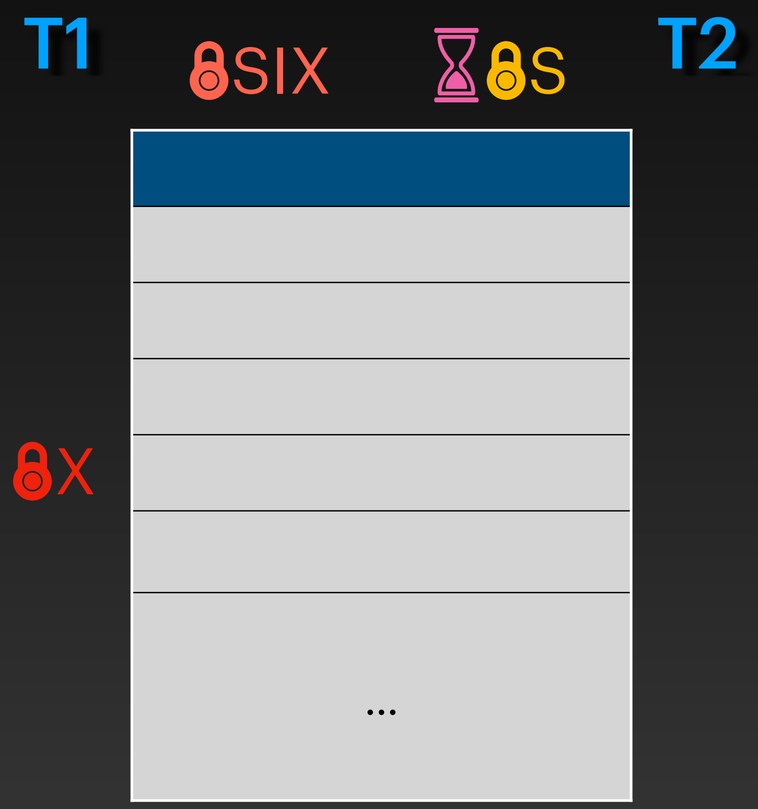
Transaction requests

Transaction holds

	X	S	IS	IX	SIX
X	✗	✗	✗	✗	✗
S	✗	✓	✓	✗	✗
IS	✗	✓	✓	✓	✓
IX	✗	✗	✓	✓	✗
SIX	✗	✗	✓	✗	✗

T1: scan all table and then update one row

T2: scan all table



Transaction requests

Transaction holds

	X	S	IS	IX	SIX
X	✗	✗	✗	✗	✗
S	✗	✓	✓	✗	✗
IS	✗	✓	✓	✓	✓
IX	✗	✗	✓	✓	✗
SIX	✗	✗	✓	✗	✗

MySQL

- Table-level locking

```
LOCK TABLE customers READ;
LOCK TABLES customers READ, items WRITE;
UNLOCK TABLES;
```

- Row-level locking

```
SELECT * FROM users WHERE NAME = 'Jones' FOR SHARE;
SELECT * FROM t WHERE i = 2 FOR UPDATE;
```

PostgreSQL

- Table-level locking

```
LOCK TABLE customers IN <lockmode> MODE;
```

  <lockmode> can be:

```
ACCESS SHARE | ROW SHARE | ROW EXCLUSIVE | SHARE UPDATE EXCLUSIVE | SHARE |
SHARE ROW EXCLUSIVE | EXCLUSIVE | ACCESS EXCLUSIVE
```

- Row-level locking

```
SELECT ... FOR <mode>;
```

  <mode> can be:

```
UPDATE | NO KEY UPDATE | SHARE | KEY SHARE
```

- More info and compatibility matrices

  https://www.postgresql.org/docs/current/explicit-locking.html