LOCKS IN POSTGRES

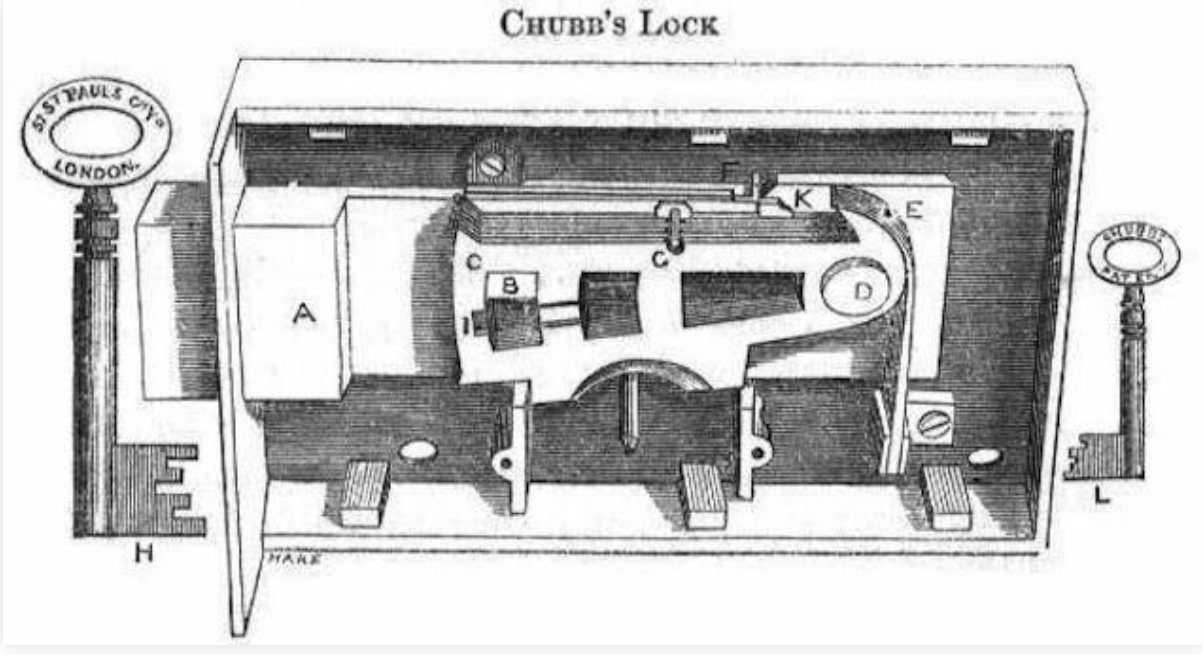
Abhijit Menon-Sen

2ndQuadrant

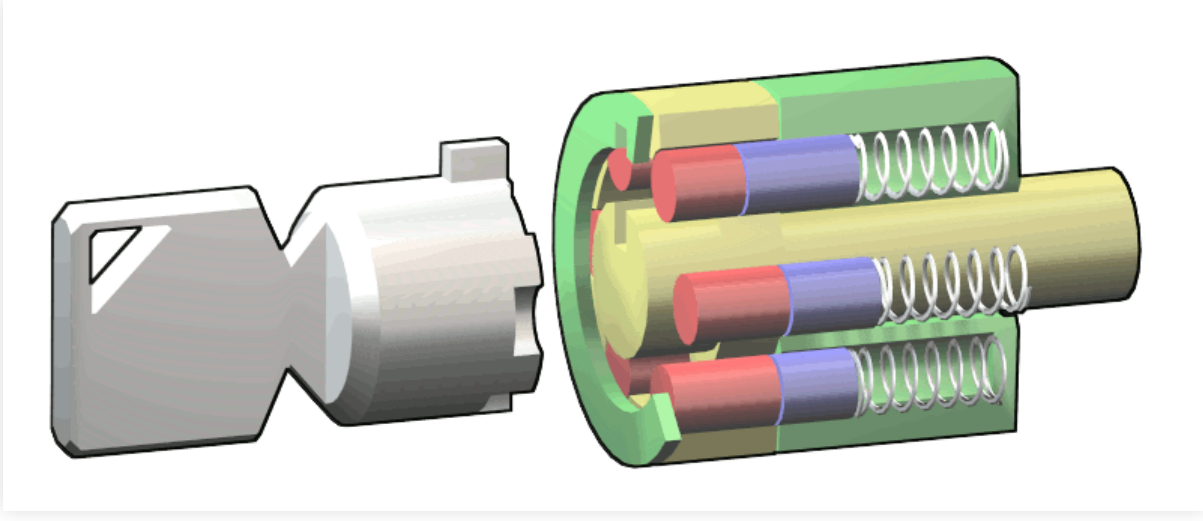
WHAT ARE LOCKS?

WHAT ARE LOCKS?

WHAT ARE LOCKS?



WHAT ARE LOCKS?



WHAT ARE LOCKS?

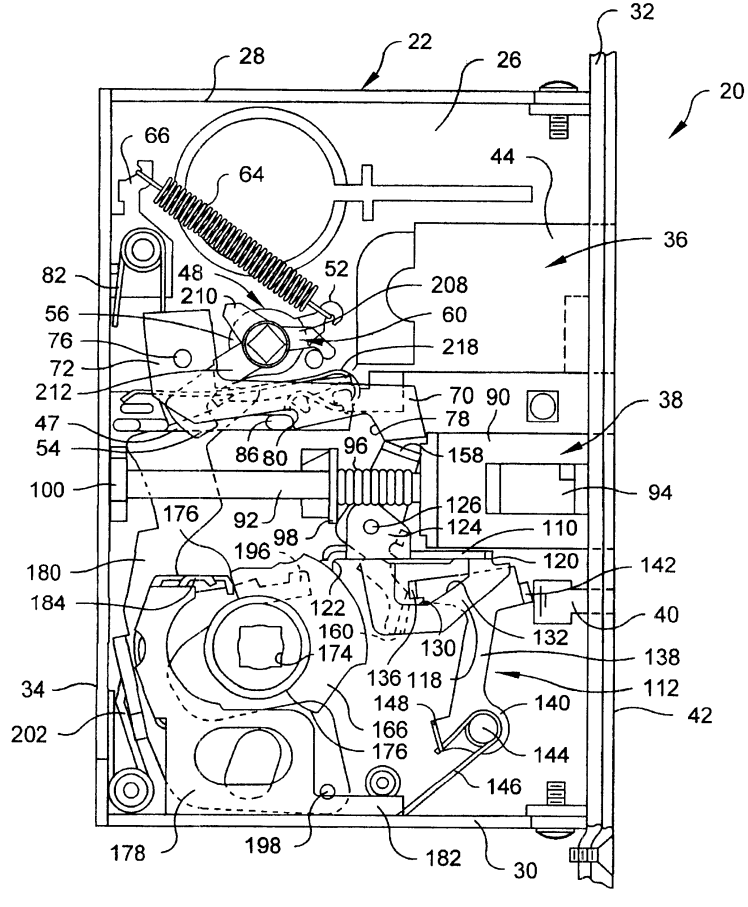


Fig. 6

WHAT ARE LOCKS?

Different kinds

Different applications

Different complexity

WHAT ARE LOCKS?

Different kinds

Different applications

Different complexity

They all do the same thing

WHAT ELSE ARE LOCKS?

Not a security mechanism

Used for synchronisation

Multiple concurrent processes

Do something one at a time

WHAT ELSE ARE LOCKS?

Not a security mechanism

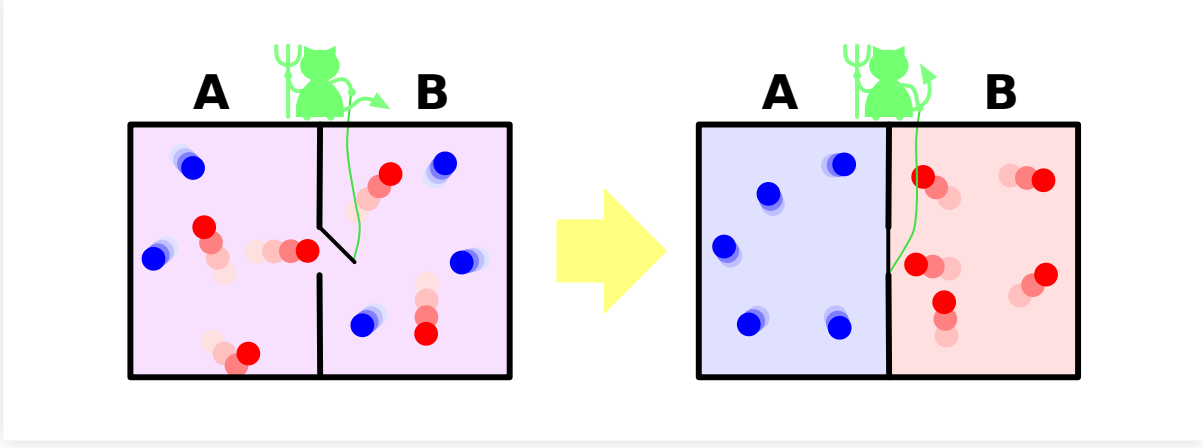
Used for synchronisation

Multiple concurrent processes

Do something one at a time

Challenging to think about

WHAT ARE LOCKS?



LOCKS AS BOTTLENECKS

Narrow bridges

Accident sites

Construction zones

LOCKS AS BOTTLENECKS

Narrow bridges

Accident sites

Construction zones

Limited analogies

WHAT DO LOCKS DO?

Control concurrency

Bank balance example

WHAT DO LOCKS DO?

Control concurrency

Bank balance example

Decrease throughput

WHAT DO LOCKS DO?

Control concurrency

Bank balance example

Decrease throughput

Needed for correctness

WHAT DO LOCKS DO?

Control concurrency

Bank balance example

Decrease throughput

Needed for correctness

Being wrong is faster

LOCK LIFETIME

Big locks = big bottlenecks

😄😄☹☹☹☹☹

LOCK LIFETIME

Big locks = big bottlenecks

Big enough locks = no concurrency

LOCK SCOPE

Few big locks = less throughput

LOCK SCOPE

Few big locks = less throughput

Many small locks = more mistakes

Many small locks = more overhead

LOCK SCOPE

Few big locks = less throughput

Many small locks = more mistakes

Many small locks = more overhead

So 😑 😑 😑 anyway

LOCK STRENGTH

Exclusive vs. shared

Multiple readers OK

One writer at a time

LOCK STRENGTH

Exclusive vs. shared

Multiple readers OK

One writer at a time

Thanks, MVCC

CONSIDERATIONS

What to lock

CONSIDERATIONS

What to lock

When to lock

When to release

CONSIDERATIONS

What to lock

When to lock

When to release

How strongly to lock

CONSIDERATIONS

What to lock

When to lock

When to release

How strongly to lock

Reduce all the things!

LOCK TYPES

Regular locks

Advisory locks

Predicate locks

Lightweight locks

Spinlocks

~~Memory barriers~~

```
ams=> select pg_backend_pid();
 pg_backend_pid
----------------
         230693
(1 row)

ams=> select 16395::regclass;
 regclass
----------
 x
(1 row)
```

```
ams=> select pid,locktype,mode,granted from pg_locks;
  pid   |  locktype  |      mode        | granted
--------+------------+------------------+---------
 230693 | relation   | AccessShareLock | t
 230693 | virtualxid | ExclusiveLock   | t
```

```
locktype           | text
database           | oid
relation           | oid
page               | integer
tuple              | smallint
virtualxid         | text
transactionid      | xid
classid            | oid
objid              | oid
objsubid           | smallint
virtualtransaction | text
pid                | integer
mode               | text
granted            | boolean
fastpath           | boolean
```

PG_LOCKS

Who: virtualtransaction, pid

How: locktype, mode, granted

What: database, relation, page, tuple, virtualxid, transactionid, classid, objid, objsubid

EXPLICIT LOCKS

LOCK TABLE x IN xxx MODE

NOWAIT to not block

EXPLICIT LOCKS

LOCK TABLE x IN xxx MODE

NOWAIT to not block

SELECT … FOR UPDATE

SKIP LOCKED

EXPLICIT LOCKS

LOCK TABLE x IN xxx MODE

NOWAIT to not block

SELECT … FOR UPDATE

SKIP LOCKED

Same locks as Postgres

LOCK MODES

ACCESS SHARE (SELECT)

...

ACCESS EXCLUSIVE (DROP TABLE)

OTHER LOCK MODES

ROW SHARE (SELECT FOR UPDATE)

ROW EXCLUSIVE (UPDATE/DELETE/INSERT)

SHARE UPDATE EXCLUSIVE (VACUUM)

SHARE (CREATE INDEX)

SHARE ROW EXCLUSIVE (ALTER TABLE)

EXCLUSIVE (materialised views)

ROW-LEVEL LOCKS

FOR UPDATE

FOR NO KEY UPDATE

FOR SHARE

FOR KEY SHARE

SESSION 1

```
$ psql -q
ams=> begin;
ams=> select pg_backend_pid();
 pg_backend_pid
----------------
         247196
(1 row)
```

SESSION 2

```
$ psql -q
ams=> begin;
ams=> select pg_backend_pid();
 pg_backend_pid
----------------
         247170
(1 row)
```

```
ams=> select pid,locktype,mode,                    \
        database,relation,page,tuple,granted    \
        from pg_locks order by pid;
-[ RECORD 1 ]-------------
pid       | 247170
locktype  | relation
mode      | AccessShareLock
database  | 16384
relation  | 11645
page      |
tuple     |
granted   | t
```

```
-[ RECORD 2 ]-------------
pid       | 247170
locktype  | virtualxid
mode      | ExclusiveLock
database  |
relation  |
page      |
tuple     |
granted   | t
```

```
-[ RECORD 3 ]-------------
pid       | 247196
locktype  | virtualxid
mode      | ExclusiveLock
database  |
relation  |
page      |
tuple     |
granted   | t
```

LOCK WAIT

```
ams=> update x set a = a * 2;
ams=>

…

ams=> select * from x for update;
```

```
  pid   |         locktype-mode         | db-relation |  p,t
--------+-------------------------------+-------------+-------
 247170 | virtualxid/ExclusiveLock      |             |
 247170 | relation/AccessShareLock      | 16384/11645 |
 247170 | transactionid/ExclusiveLock   |             |
 247170 | relation/RowExclusiveLock     | 16384/16395 |
 247196 | transactionid/ShareLock       |             |
 247196 | relation/RowShareLock         | 16384/16395 |
 247196 | virtualxid/ExclusiveLock      |             |
 247196 | tuple/AccessExclusiveLock     | 16384/16395 | [0,1]
```

LOCK RELEASE

```
ams=> commit;
ams=>

…

ams=> select * from x for update;
 a
---
 2
 4
(2 rows)
```

```
  pid   |           locktype-mode           | db-relation | p,t
--------+-----------------------------------+-------------+-----
 247170 | relation/AccessShareLock          | 16384/11645 |
 247170 | virtualxid/ExclusiveLock          |             |
 247196 | relation/RowShareLock             | 16384/16395 |
 247196 | virtualxid/ExclusiveLock          |             |
 247196 | transactionid/ExclusiveLock       |             |
```

FIND BLOCKERS

pg_locks has the info

Conflicts not obvious

Use pg_blocking_pids() (9.5+)

```
ams=> select pid,locktype,mode from pg_locks \
        where not granted;
  pid   |   locktype    |   mode
--------+---------------+-----------
 245562 | transactionid | ShareLock
(1 row)

ams=> select * from pg_blocking_pids(245562);
 pg_blocking_pids
------------------
 {245547}
(1 row)
```

```
ams=> select pid,locktype,mode from pg_locks \
        where pid=245547;
  pid   |   locktype    |       mode
--------+---------------+-------------------
 245547 | relation      | AccessShareLock
 245547 | relation      | RowExclusiveLock
 245547 | virtualxid    | ExclusiveLock
 245547 | transactionid | ExclusiveLock
(4 rows)
```

DROP TABLE

```
ams=> SET lock_timeout = '3s';
SET
ams=> drop table x;
ERROR:  canceling statement due to lock timeout
```

DROP TABLE

```
  pid   |         locktype-mode          | db-relation
--------+--------------------------------+-------------
 247532 | relation/AccessExclusiveLock | 16384/16395
```

CREATE INDEX

```
ams=> CREATE INDEX xa ON x(a);
CREATE INDEX
ams=> drop index xa;
ERROR:  canceling statement due to lock timeout
```

PG_STAT_ACTIVITY

```
ams=> select pid,wait_event_type,wait_event,state \
        from pg_stat_activity where state='active';
  pid   | wait_event_type | wait_event | state
--------+-----------------+------------+--------
 247532 | Lock            | relation   | active
 247170 |                 |            | active
(2 rows)
```

CREATE INDEX

CREATE INDEX needs SHARE

Like SELECT

DROP INDEX needs AEL

CREATE INDEX

CREATE INDEX needs SHARE

Like SELECT

DROP INDEX needs AEL

DDL locking is complex

ADVISORY LOCKS

Not used by postgres

Meant for applications

64-bit integer lock keys

Lifetime: session, transaction

Mode: exclusive, shared

SESSION-LEVEL LOCKS

`SELECT pg_advisory_lock(42)`

`ExclusiveLock`: one holder

Released when session ends

(Or pg_advisory_unlock)

TRANSACTION LOCKS

`SELECT pg_advisory_xact_lock(42)`

Still an `ExclusiveLock`

Released when transaction ends

(No explicit unlock)

SHARED LOCKS

Mode `ShareLock`

Multiple concurrent holders

Conflicts with ExclusiveLock

`pg_advisory_lock_shared(42)`

`pg_advisory_xact_lock_shared(42)`

Same lifetime rules

NON-BLOCKING

`pg_try_advisory_xxx()` functions

Same modes, same lifetimes

Don't wait to acquire lock

Return true/false immediately

Worker can do something else

PG_LOCKS

locktype = 'advisory'

mode = Exclusive/Shared

Key in classid/objid/objsubid

LOCKING ORDER

A: acquire 1, acquire 3

B: acquire 3, acquire 1

LOCKING ORDER

A: acquire 1, acquire 3

B: acquire 3, acquire 1

A: ☹

B: ☹

LOCKING ORDER

A: acquire 1, acquire 3

B: acquire 3, acquire 1

A: ☹

B: ☹

Postgres: 😄

DEADLOCKS

Not just advisory locks

Postgres: 😟

Complex cycles

DEADLOCKS

Not just advisory locks

Postgres: 😟

Complex cycles

Ordered acquisition

DEADLOCK DETECTOR

Expensive

Aborts waiting transactions

CONFIGURATION

lock_timeout (acquisition)

deadlock_timeout (detection)

log_lock_waits

PREDICATE LOCKS

Serializable Snapshot Isolation (SSI)

`SERIALIZABLE` isolation level

Transactions execute as if in order

Roll back conflicting transactions

(Application can retry)

PREDICATE LOCKS

Locks used to detect conflicts

They never block anything

Lock escalation: tuple, page, table

PREDICATE LOCKS

Locks used to detect conflicts

They never block anything

Lock escalation: tuple, page, table

`backend/storage/lmgr/README-SSI`

LOW-LEVEL LOCKS

Less overhead

For internal use

No deadlock detection

Not available to users

No pg_locks entries

LWLOCKS

Lightweight locks

Shared memory access

LWLockAcquire/Release

Mode: LW_SHARED/EXCLUSIVE

Released on errors

SPINLOCKS

Low-level lock

Fast acquisition

Very fast release

No queued waiters

No error handling

EXAMPLES

EXAMPLES

LWLocks

XidGenLock

WALWriteLock

LockMgrLock

EXAMPLES

LWLocks

XidGenLock

WALWriteLock

LockMgrLock

Spinlocks

(Used by LWLocks)

XLogCtl->info_lck

REPLICA LOCKING

One process ('startup')

Backends take weak locks

No deadlocks

AccessExclusive locks

Query cancels

SUMMARY

Look at pg_locks

SUMMARY

Look at pg_locks

(WHERE NOT granted)

SUMMARY

Look at pg_locks

(WHERE NOT granted)

backend/storage/lmgr/README

QUESTIONS?

THANK YOU

In solidarity with the victims of communal violence in Delhi