PostgreSQL Lock Management

BEYOND THE DEADLOCK

Greg Dostatni DBA @ Command Prompt, Inc

Presenting on April 18, 2024 at Postgres Conference

COMMAND PROMPT, INC.

INTRODUCTION: WHAT IS THIS ABOUT?

Database

Case studies

lmgr internals

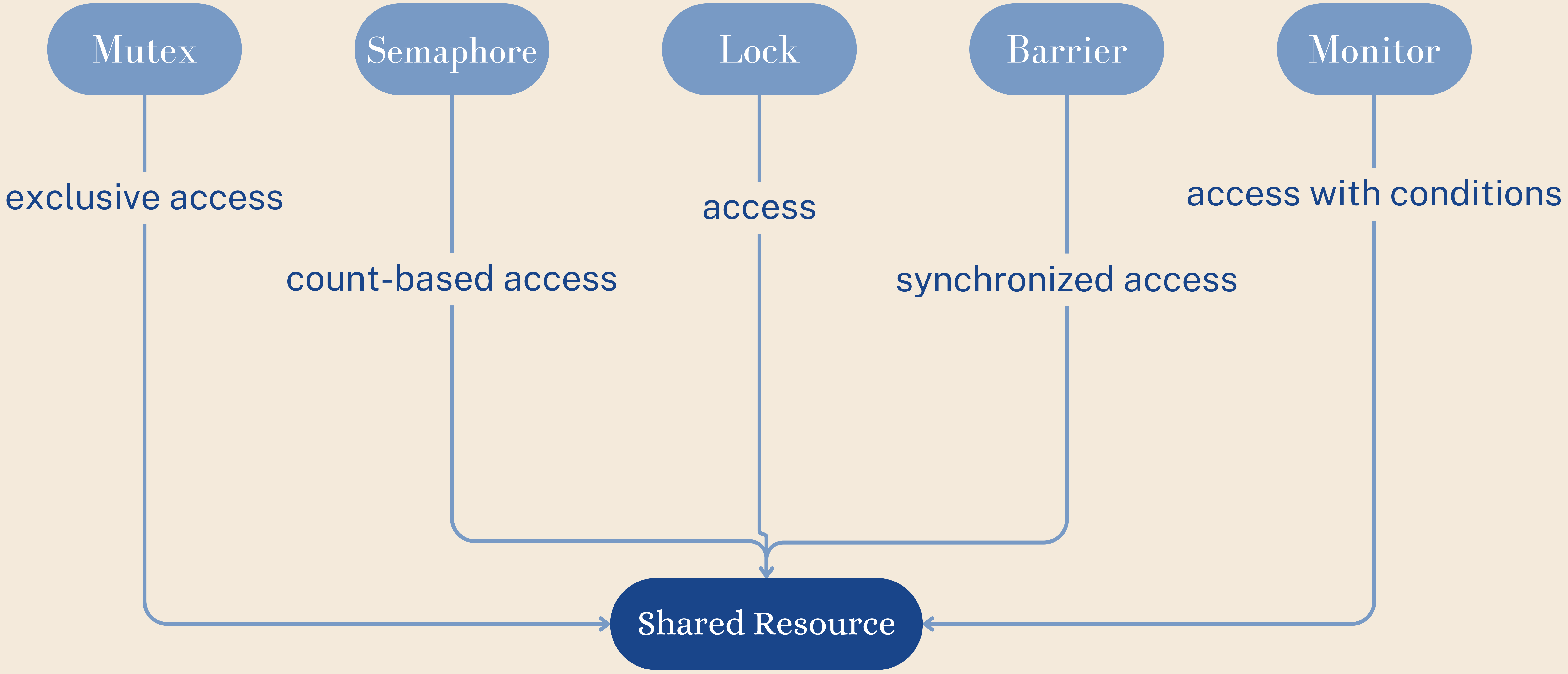
~~Deadlocks~~

COMMAND PROMPT, INC.

LOCK MANAGER 101

COMMAND PROMPT, INC.

INTRODUCTION: WHAT ARE LOCKS USED FOR?



COMMAND PROMPT, INC.

INTRODUCTION: MULTI VERSION CONCURRENCY CONTROL

Benefits

Concurrency

Consistency and Isolation

Optimistic locking

Implementation

Row versioning

Visibility rules

Considerations

Bloat

Transaction ID wraparound

Performance

COMMAND PROMPT, INC.

INTRODUCTION: TYPES OF LOCKS

Regular Locks

SIReadLocks

Spinlocks

LWLocks

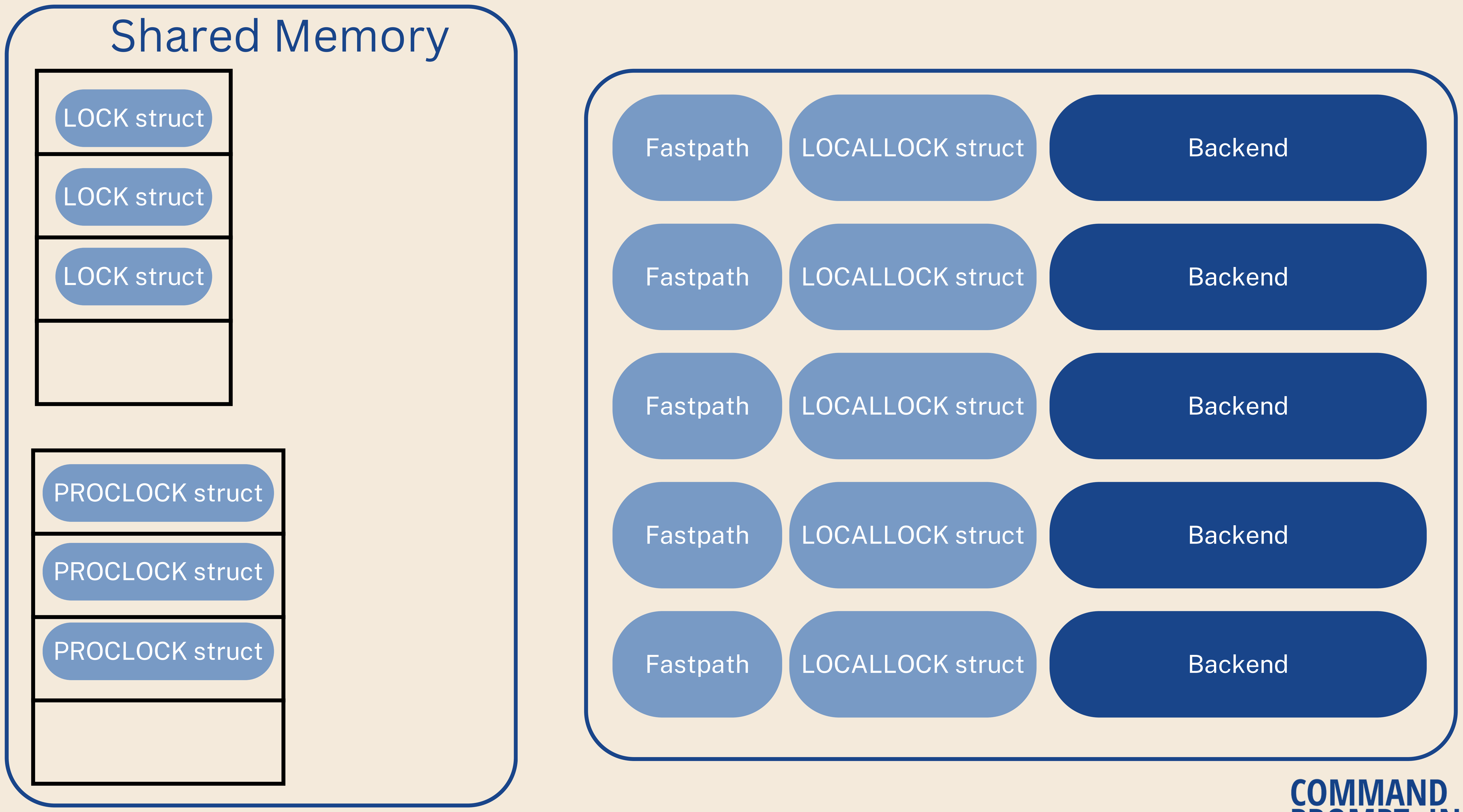
COMMAND PROMPT, INC.

LOCKS: TYPES AND INTERACTIONS

	Existing Lock Mode							
Requested	AS	RS	RE	SUE	S	SRW	E	AE
Access Share								X
Row Share							X	X
Row Exclusive					X	X	X	X
Share Update Exclusive				X	X	X	X	X
Share			X	X		X	X	X
Share Row Exclusive			X	X	X	X	X	X
Exclusive		X	X	X	X	X	X	X
Access Exclusive	X	X	X	X	X	X	X	X

COMMAND PROMPT, INC.

INTRODUCTION: PG LOCK MANAGER

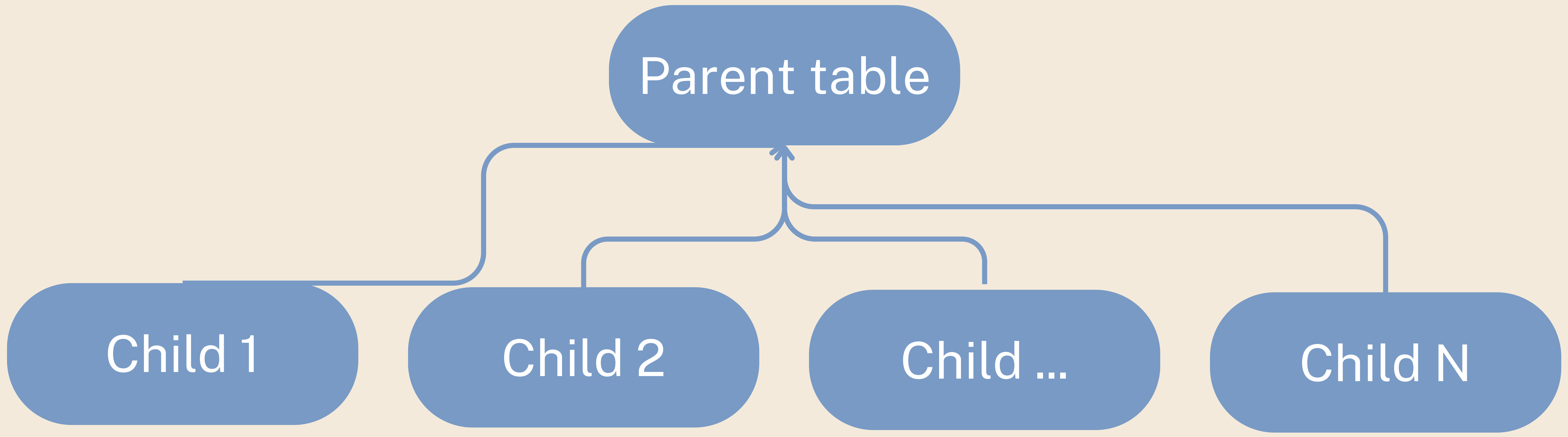


COMMAND PROMPT, INC.

REGULAR LOCKS

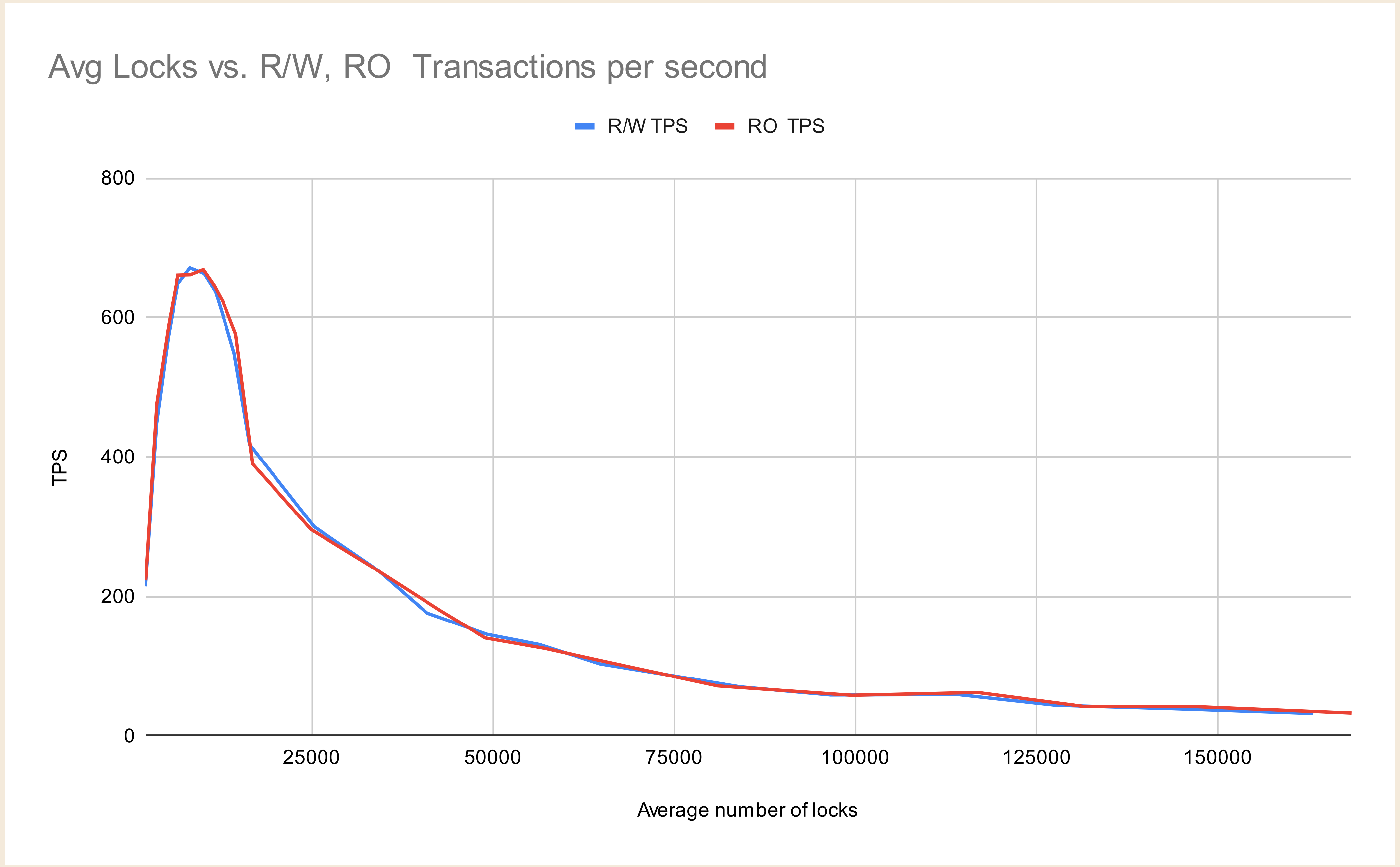
COMMAND PROMPT, INC.

CASE STUDY 1: INHERITANCE



COMMAND PROMPT, INC.

CASE STUDY 1: INHERITANCE



COMMAND PROMPT, INC.

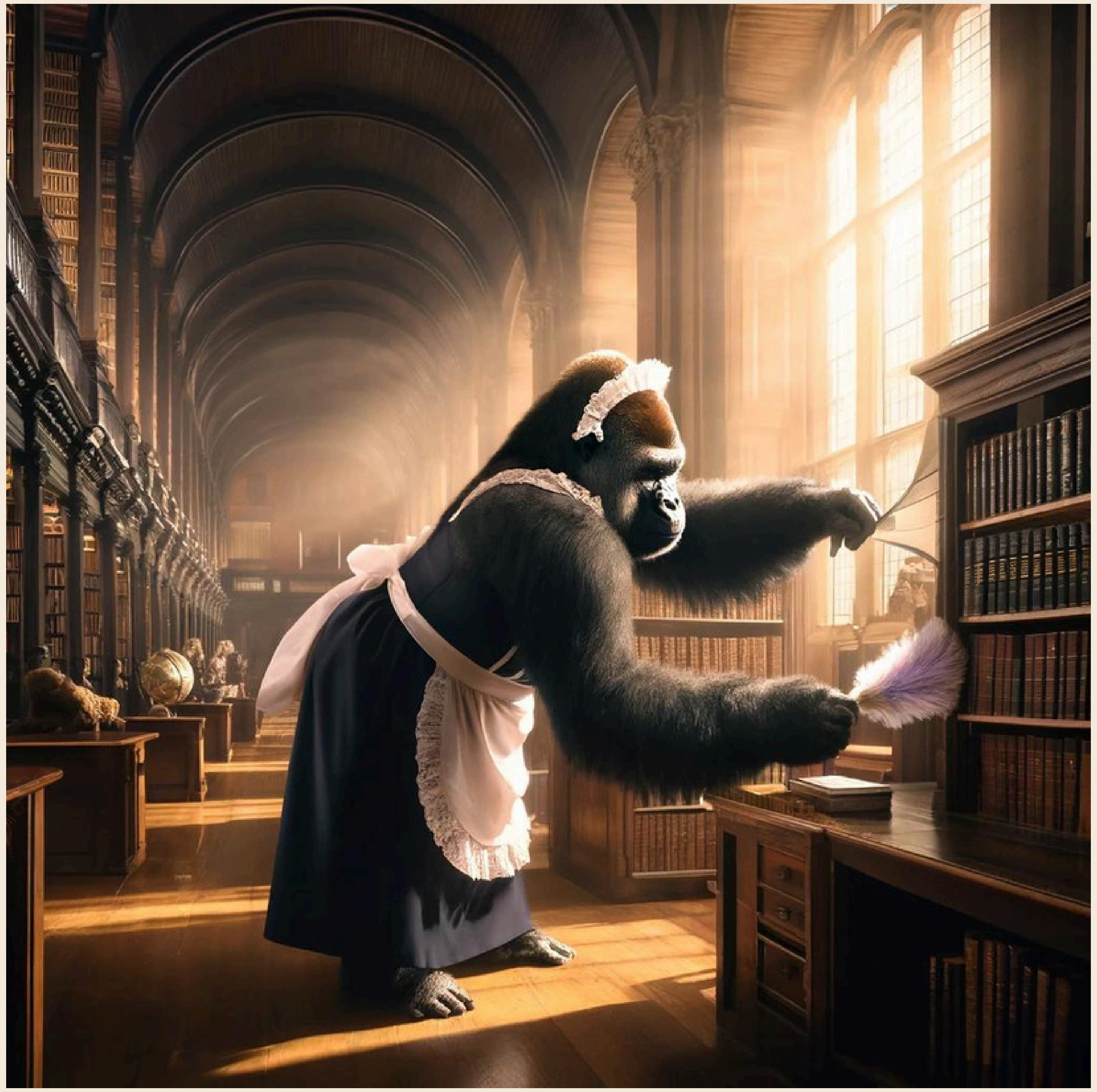
CASE STUDY 1: CONVERT TO NATIVE

```
BEGIN TRANSACTION;

ALTER TABLE <parent>
RENAME TO <parent_old>;

CREATE TABLE <parent>
    (
        LIKE <parent_old>
        INCLUDING INDEXES
        INCLUDING COMMENTS
        INCLUDING CONSTRAINTS
    ) PARTITION BY
RANGE(<partition_key>);
```

```
For every child table:

    ALTER TABLE <child> NO INHERIT
<parent>;

    ALTER TABLE <parent>
        ATTACH PARTITION <child>
        FOR VALUES FROM (<from>) TO
(<to>);

DROP TABLE <parent>;
COMMIT;
```

COMMAND PROMPT, INC.

CASE STUDY 2: NATIVE PARTITIONING



Cleanup / Archiving

- Delete Records
  - Slow, high IO, possible table bloat
- Detach and drop partition
  - Fast, but requires an exclusive lock on parent
  - Use lock_timeout to fail gracefully and try again
- Truncate partition
  - Reclaim space without locking parent
  - Detach is still required

COMMAND PROMPT, INC.

CASE STUDY 2: NATIVE PARTITIONING

```
CREATE PROCEDURE del_records
    (plimit integer, psleep decimal)
LANGUAGE PLPGSQL
AS $$
DECLARE
    _r record; _count integer; _ids int[];
BEGIN

    LOOP
       SELECT array_agg(<pkey>)
       INTO _ids FROM (
            SELECT <pkey> from <table> limit plimit
            ) sub ;

     EXIT WHEN array_length(_ids, 1) IS NULL;
```

```
        BEGIN
            DELETE FROM <table> where pkey =
ANY(_ids);
        COMMIT;
        END;

        PERFORM pg_sleep(psleep);

    END LOOP;

END; $$ ;
```

COMMAND PROMPT, INC.

LOCKS: INVESTIGATION

```
SELECT
   a.query,
    array_agg(
         DISTINCT l.relation::regclass::text || ':' ||
                         l.mode
      ) AS locks
FROM
    pg_stat_activity a
JOIN
    pg_locks l ON l.pid = a.pid
WHERE
    a.state <> 'idle' AND
    l.relation IS NOT NULL  AND
    a.query NOT ILIKE '%pg_stat_activity%'
GROUP BY a.query
ORDER BY a.query;
```

Run in separate session outside of transaction

Not in production

```
BEGIN;
    <query>
    -- execute query on right
    -- before rollback
ROLLBACK;
```

```
      post_tags    AccessExclusiveLock,
 post_tags_pkey    AccessExclusiveLock,
      post_tags    ShareLock,
           tags    AccessExclusiveLock,
      tags_pkey    AccessExclusiveLock,
           tags    ShareLock
```

COMMAND PROMPT, INC.

LOCKS: MONITORING

Configuration

log_lock_waits
deadlock_timeout
max_locks_per_transaction
lock_timeout

Realtime

pg_locks
pg_stat_activity

pg_blocking_pids(PID)

Questions

1. Which transaction is blocked?
2. Which transaction is doing the blocking?
3. Which objects are locked the most?
4. Which locks is this query trying to acquire?
5. What is the average waiting time for a lock right now?

COMMAND PROMPT, INC.

LIGHTWEIGHT LOCKS

COMMAND PROMPT, INC.

CASE STUDY 3: SUBTRANSACTIONS

```
BEGIN;
...
EXCEPTION
    WHEN ... THEN ... ;
END;
```

```
BEGIN;
...
SAVEPOINT s1;
...
SAVEPOINT s2;
...
COMMIT;
```

COMMAND PROMPT, INC.

CASE STUDY 3: SUBTRANSACTIONS - MULTIXACT IDS

https://buttondown.email/nelhage/archive/notes-on-some-postgresql-implementation-details/

Row level locks are stored on disk in the header of the tuple.
Multixact ID represents an immutable set of locking transaction IDs stored in a global MultiXact store.
Any access to MuliXact is controlled by a single global LWLock.

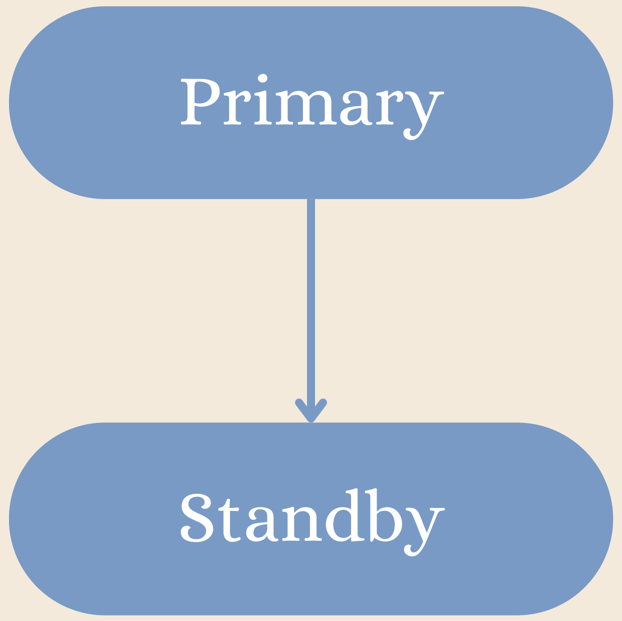
Taking out an exclusive lock on a row, but performing the work in subtransaction results in the use of multixact IDs.

```
SELECT [some row] FOR UPDATE;
SAVEPOINT save;
UPDATE [the same row];
```

COMMAND PROMPT, INC.

CASE STUDY 3: SUBTRANSACTIONS - SLRU OVERFLOW

https://postgres.ai/blog/20210831-postgresql-subtransactions-considered-harmful



Primary:
Subtransactions issuing updates
Ongoing long-running transaction

Standby:
Selects dealing with the same tuples.
pg_subtrans

https://gitlab.com/postgres-ai/postgresql-consulting/tests-and-benchmarks/-/issues/21

COMMAND PROMPT, INC.

LWLOCKS: LIGHTWEIGHT LOCKS

Access to shared memory structures

Meant for internal processes

Provide shared and exclusive lock modes

COMMAND PROMPT, INC.

LWLOCKS: PG_STAT_ACTIVITY

Waiting

Client
Extension
IO
IPC
Timeout

Locking

Buffer Pin
Lock
LWLock
Extension

LWLock events

LockManager
Buffer*
MultiXact*
WAL*

TAKEAWAYS

Lock Manager is awesome

It is a complex system

Problems may be difficult to diagnose

Read the source

COMMAND PROMPT, INC.

MONITORING RESOURCES

PostgreSQL documentation and source

https://www.postgresql.org/docs/16/runtime-config-developer.html

https://www.postgresql.org/docs/16/monitoring-stats.html#WAIT-EVENT-TABLE

wiki.postgresql.org/wiki/Lock_Monitoring

perf top

https://github.com/jnidzwetzki/pg-lock-tracer

https://github.com/lesovsky/pgcenter

```
perf record --call-graph dwarf -F 500 -a sleep 5
perf report --no-children --sort comm,symbol
```

COMMAND PROMPT, INC.

QUESTIONS?

COMMAND PROMPT, INC.

THANK YOU!

COMMAND PROMPT, INC.