

# Inside PostgreSQL Shared Memory

BRUCE MOMJIAN



POSTGRESQL is an open-source, full-featured relational database. This presentation gives an overview of the shared memory structures used by Postgres.

<https://momjian.us/presentations>



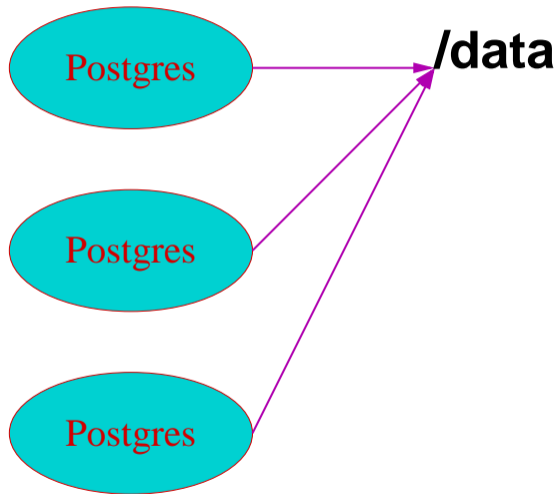
*Creative Commons Attribution License*

*Last updated: April 2024*

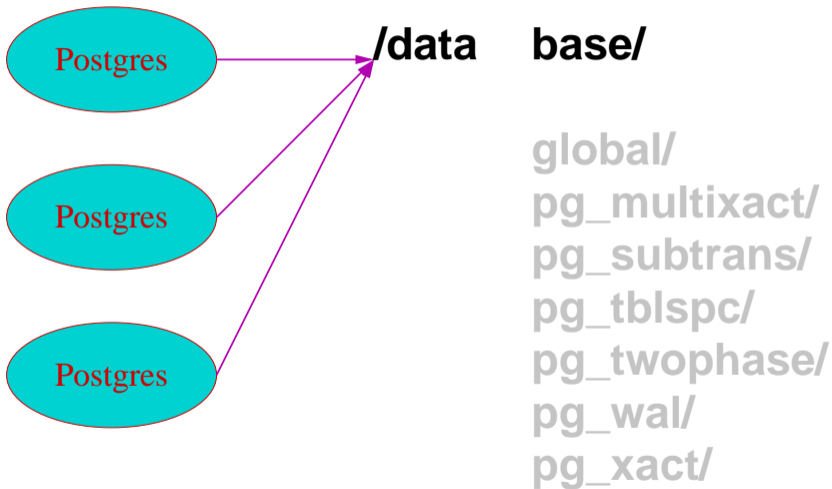
# Outline

1. File storage format
2. Shared memory creation
3. Shared buffers
4. Row value access
5. Locking
6. Other structures

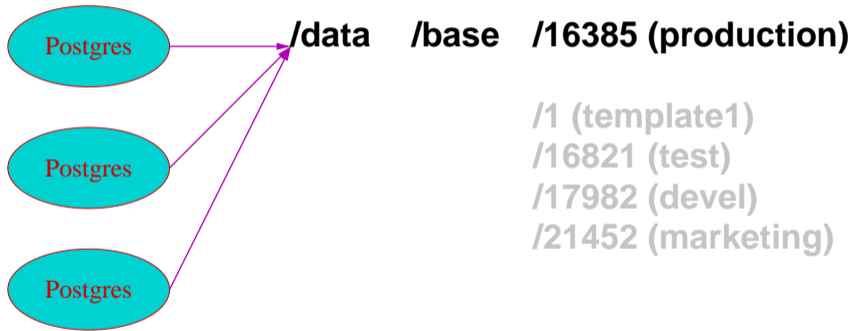
# File System /data



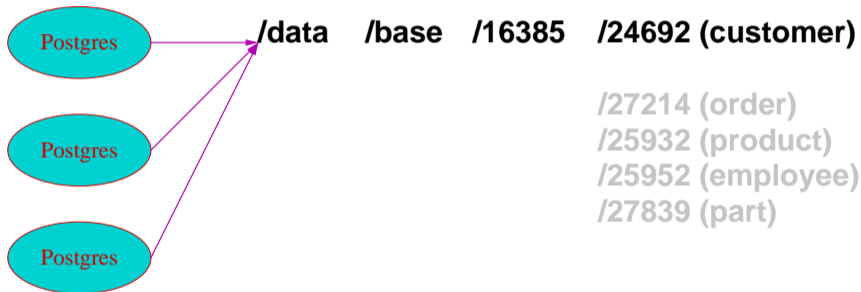
# File System /data/base



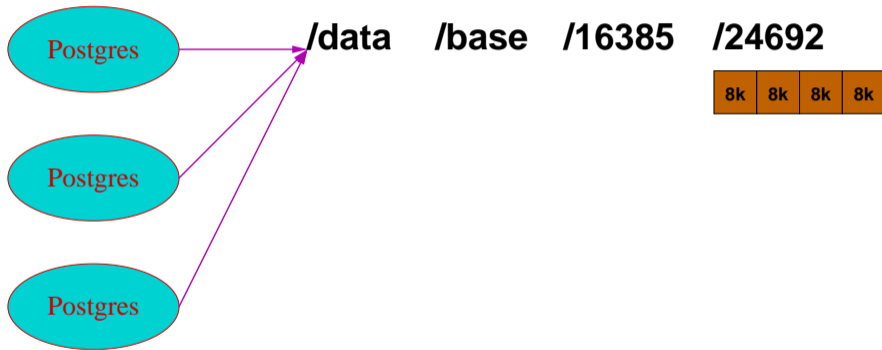
# File System /data/base/db



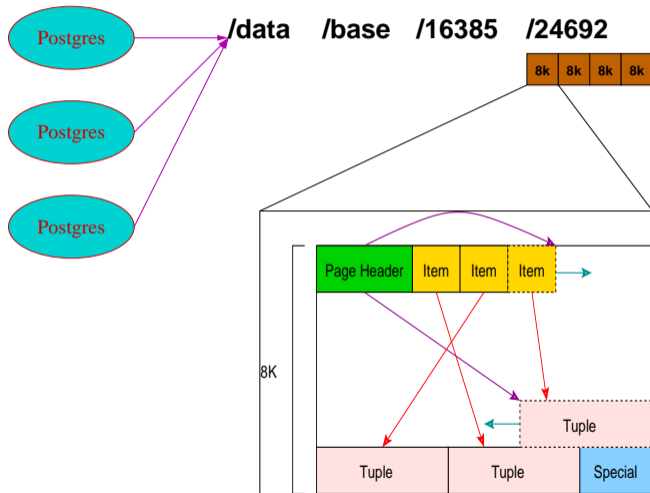
# File System /data/base/db/table



# File System Data Pages

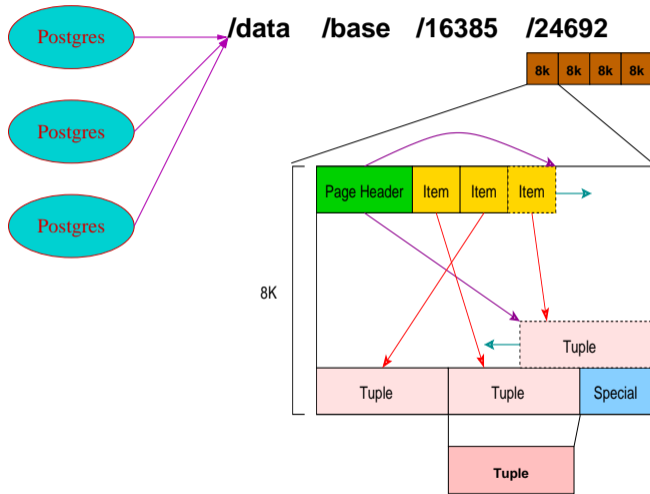


# Data Pages



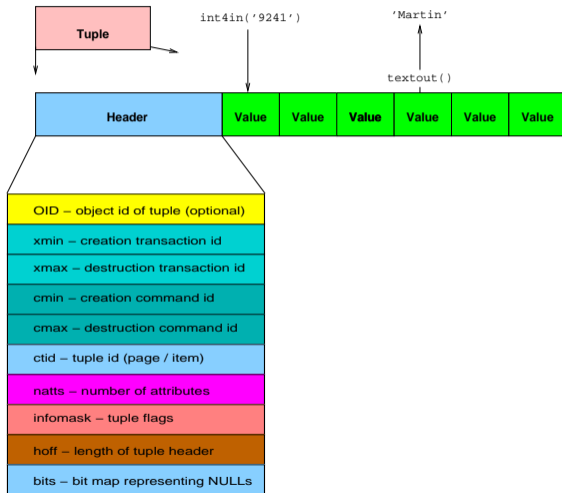


# File System Block Tuple



<https://stormatics.tech/blogs/postgresql-internals-part-2-understanding-page-structure>

# File System Tuple



# Tuple Header C Structures

```
typedef struct HeapTupleFields
{
    TransactionId t_xmin;          /* inserting xact ID */
    TransactionId t_xmax;         /* deleting or locking xact ID */

    union
    {
        CommandId t_cid;         /* inserting or deleting command ID, or both */
        TransactionId t_xvac;     /* VACUUM FULL xact ID */
    } t_field3;
} HeapTupleFields;

typedef struct HeapTupleHeaderData
{
    union
    {
        HeapTupleFields t_heap;
        DatumTupleFields t_datum;
    } t_choice;

    ItemPointerData t_ctid;       /* current TID of this or newer tuple */

    /* Fields below here must match MinimalTupleData! */

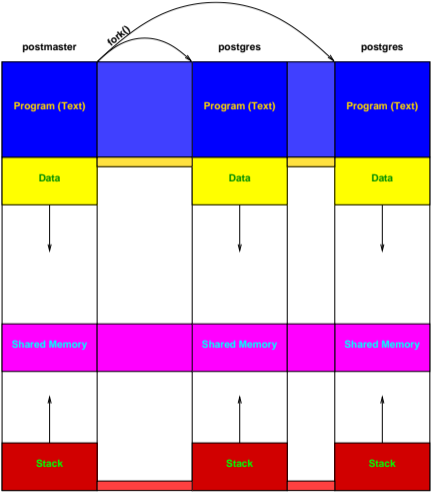
    uint16 t_infomask2;          /* number of attributes + various flags */
    uint16 t_infomask;           /* various flag bits, see below */
    uint8 t_hoff;                /* sizeof header incl. bitmap, padding */

    /* ^ - 23 bytes - ^ */

    bits8 t_bits[1];             /* bitmap of NULLs -- VARIABLE LENGTH */

    /* MORE DATA FOLLOWS AT END OF STRUCT */
} HeapTupleHeaderData;
```

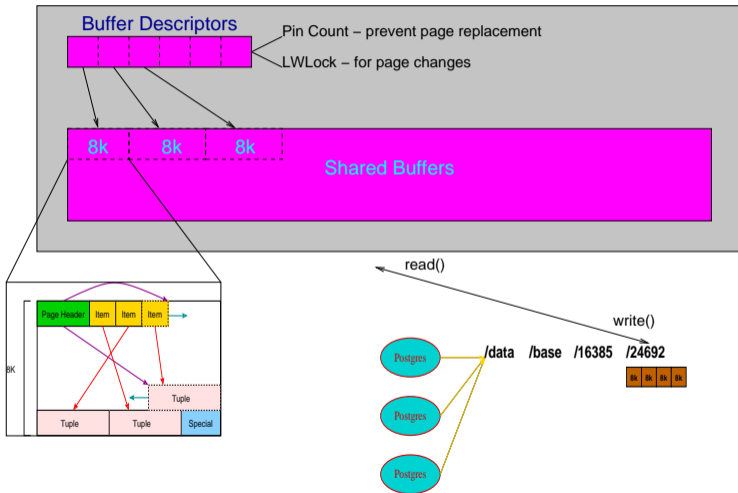
# Shared Memory Creation



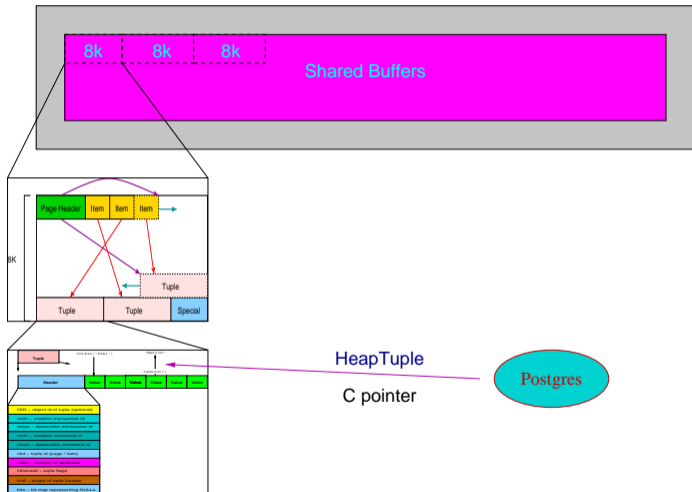
# Shared Memory



# Shared Buffers



# HeapTuples



# Finding A Tuple Value in C

```
Datum
nocachegetattr(HeapTuple tuple,
               int attnum,
               TupleDesc tupleDesc,
               bool *isnull)
{
    HeapTupleHeader tup = tuple->t_data;
    Form_pg_attribute *att = tupleDesc->attrs;

    {
        int i;

        /*
         * Note - This loop is a little tricky. For each non-null attribute,
         * we have to first account for alignment padding before the attr,
         * then advance over the attr based on its length. Nulls have no
         * storage and no alignment padding either. We can use/set
         * attcacheoff until we reach either a null or a var-width attribute.
         */
        off = 0;
        for (i = 0;; i++) /* loop exit is at "break" */
        {
            if (HeapTupleHasNulls(tuple) && att_isnull(i, bp))
                continue; /* this cannot be the target att */

            if (att[i]->attlen == -1)
                off = att_align_pointer(off, att[i]->attalign, -1,
                                       tp + off);
            else
                /* not varlena, so safe to use att_align_nominal */
                off = att_align_nominal(off, att[i]->attalign);

            if (i == attnum)
                break;

            off = att_addlength_pointer(off, att[i]->attlen, tp + off);
        }
    }

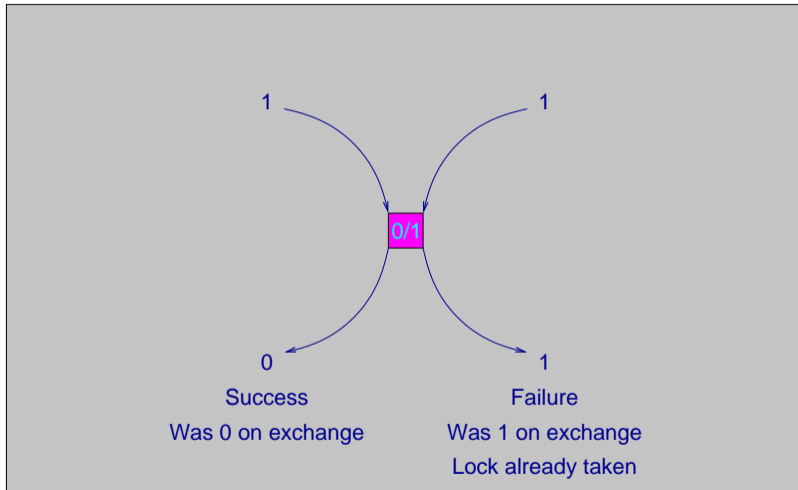
    return fetchatt(att[attnum], tp + off);
}
```



# Value Access in C

```
#define fetch_att(T,attbyval,attlen) \  
( \  
    (attbyval) ? \  
        ( \  
            (attlen) == (int) sizeof(int32) ? \  
                Int32GetDatum(*((int32 *) (T))) \  
            : \  
            ( \  
                (attlen) == (int) sizeof(int16) ? \  
                    Int16GetDatum(*((int16 *) (T))) \  
                : \  
                ( \  
                    AssertMacro((attlen) == 1), \  
                    CharGetDatum(*((char *) (T))) \  
                ) \  
            ) \  
        ) \  
    ) \  
    : \  
    PointerGetDatum((char *) (T)) \  
)
```

# Test And Set Lock Can Succeed Or Fail



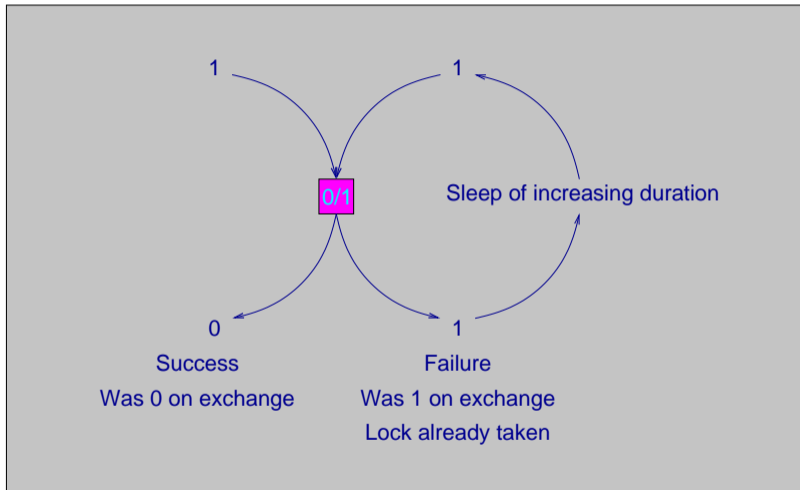
<https://en.wikipedia.org/wiki/Test-and-set>

# Test And Set Lock x86 Assembler

```
static __inline__ int
tas(volatile slock_t *lock)
{
    register slock_t _res = 1;

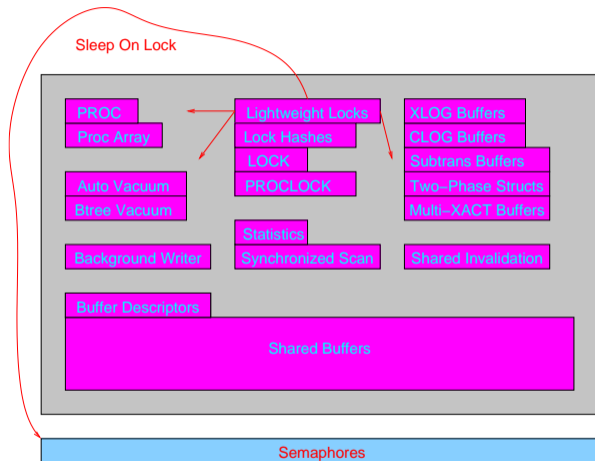
    /*
     * Use a non-locking test before asserting the bus lock. Note that the
     * extra test appears to be a small loss on some x86 platforms and a small
     * win on others; it's by no means clear that we should keep it.
     */
    __asm__ __volatile__(
        "    cmpb    $0,%1    \n"
        "    jne    1f        \n"
        "    lock   \n"
        "    xchgb  %0,%1    \n"
        "1: \n"
        "+q"(_res), "+m"(*lock)
        :
        :
        : "memory", "cc");
    return (int) _res;
}
```

# Spin Lock Always Succeeds



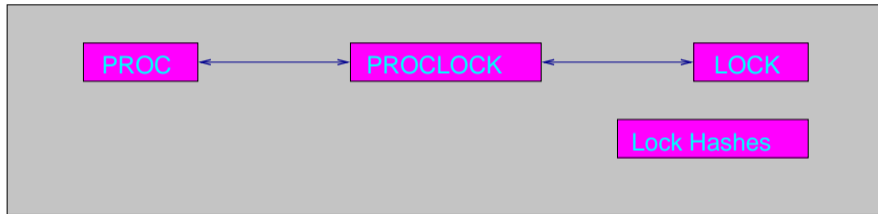
Spinlocks are designed for short-lived locking operations, like access to control structures. They are not be used to protect code that makes kernel calls or other heavy operations.

# Light Weight Locks

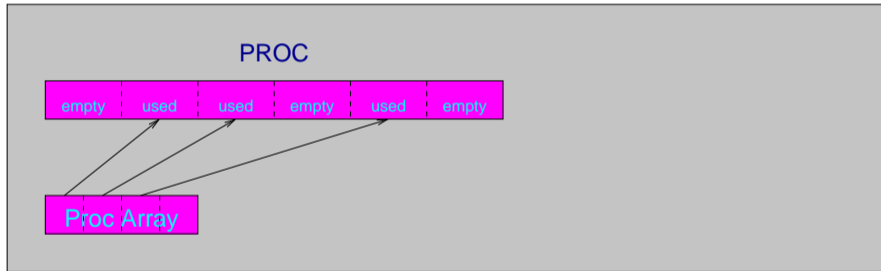


Light weight locks attempt to acquire the lock, and go to sleep on a semaphore if the lock request fails. Spinlocks control access to the light weight lock control structure.

# Database Object Locks



# Proc



# Other Shared Memory Structures





# Conclusion



<https://momjian.us/presentations>

[https://www.flickr.com/photos/john\\_getchel/](https://www.flickr.com/photos/john_getchel/)