ACID

WHAT MAKES SQL DATABASES.. SQL DATABASES

- SQL language - declarative query language
- ACID
  - Atomicity
  - Consistency
  - Isolation (Transaction isolation)
  - Durability

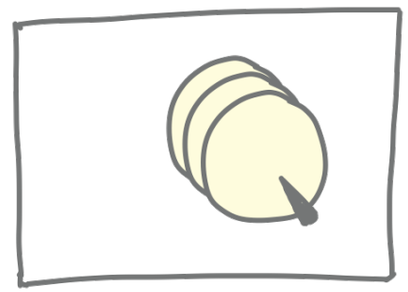
DURABILITY

DURABILITY

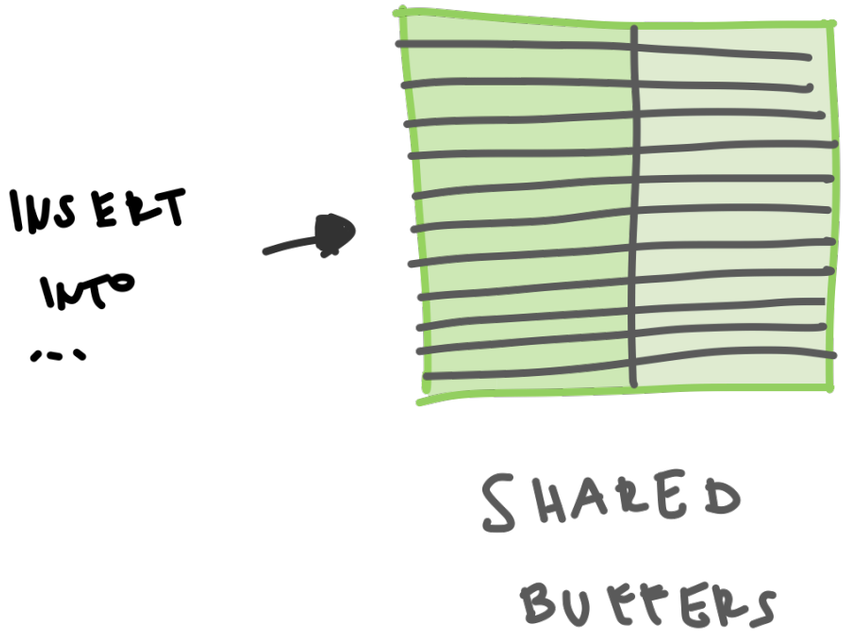
- = Saving data to persistent storage (disk drive)
- .. is complicated

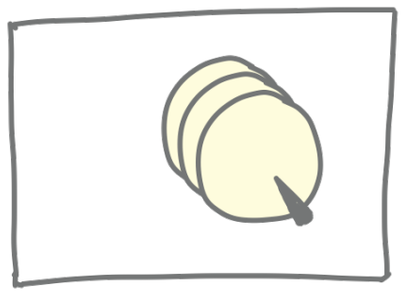
CACHE ONION

INSERT
INTO
...



CACHE ONION





SHARED BUFFERS

- Specialized PostgreSQL cache
- Transactions don't operate on raw data - only on data in cache area
- When the data ("page") not in cache, database loads it from disk

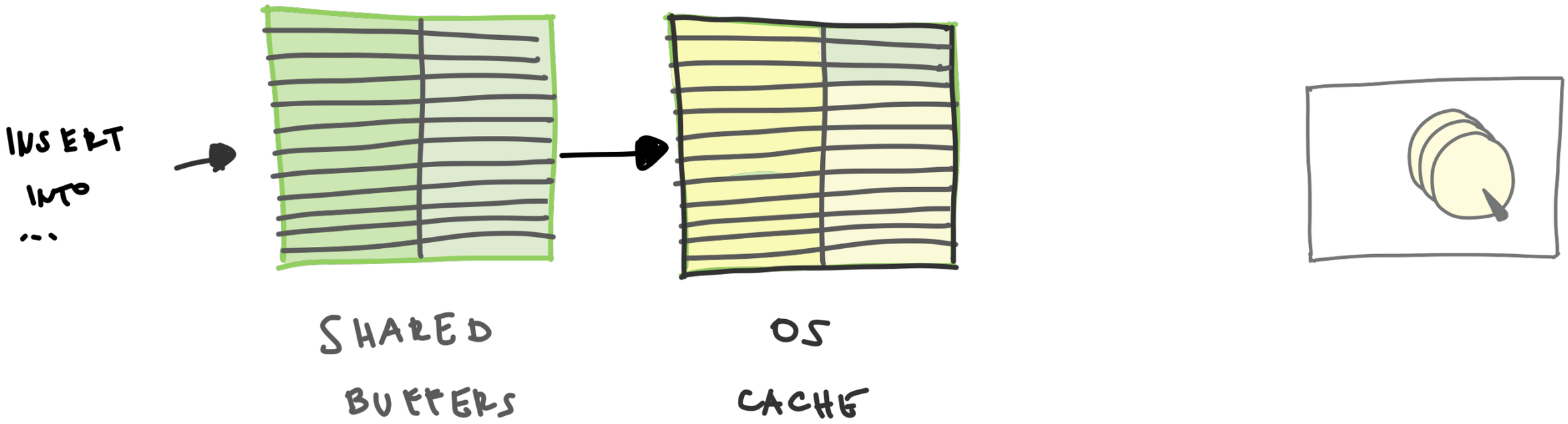
```
CREATE EXTENSION pg_buffercache;

SELECT c.relname, b.bufferid, b.isdirty
FROM pg_class c
JOIN pg_buffercache b ON
b.relfilenode=c.relfilenode
JOIN pg_database d
ON (b.reldatabase=d.oid AND
d.datname=current_database())
WHERE c.relname not like 'pg%';
```

```
CREATE TABLE test(id serial, name text);

INSERT INTO test(name) VALUE('tomas');

-- pg_buffercache select
```

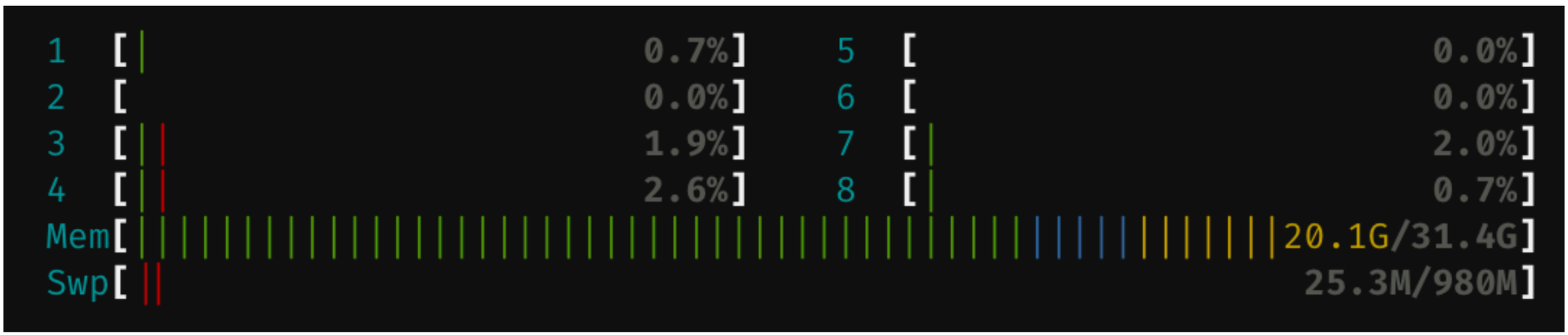
```
   relname    | bufferid    |    isdirty
--------------+-------------+-------------
 test         | 358         | t
```

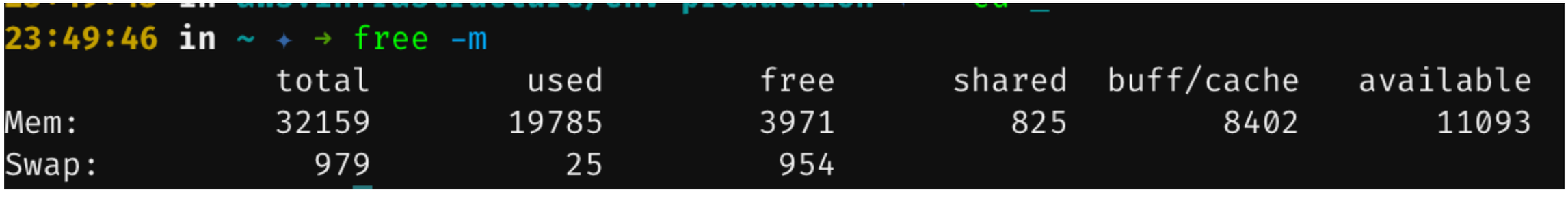
```
CHECKPOINT;

-- pg_buffercache select

   relname   | bufferid   |   isdirty
-------------+------------+-----------
 test        | 358        | f
```

CACHE ONION



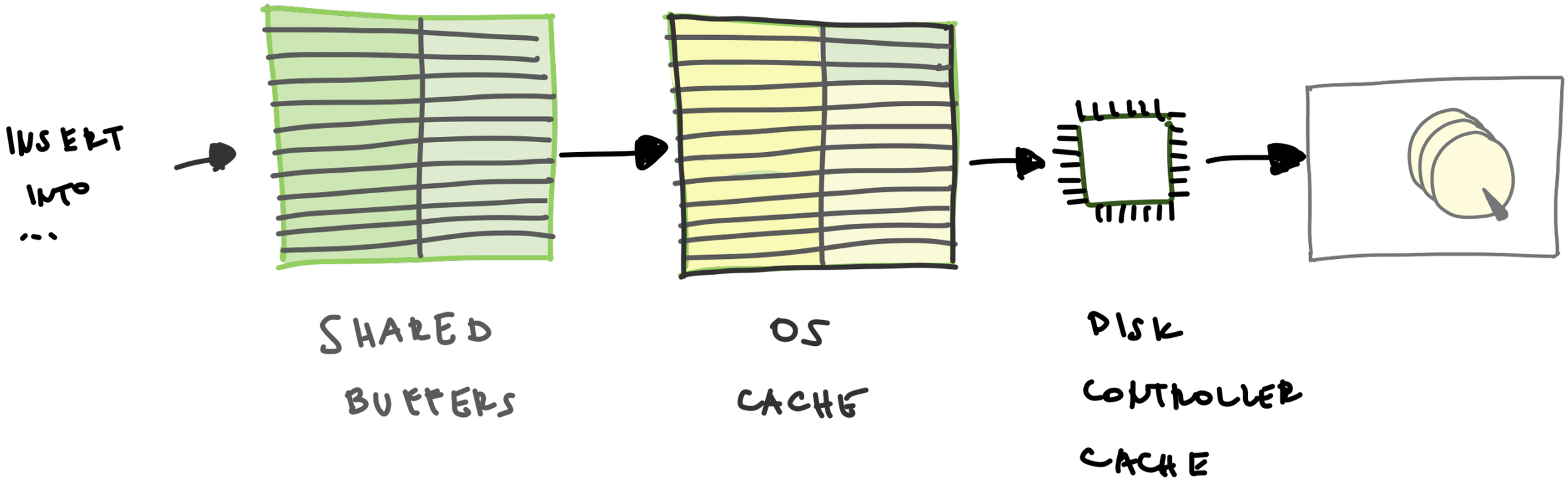
OS FILESYSTEM CACHE





```
23:49:46 in ~ ✦ → free -m
               total        used        free      shared  buff/cache   available
Mem:           32159       19785        3971         825        8402       11093
Swap:            979          25         954
```

CACHE ONION

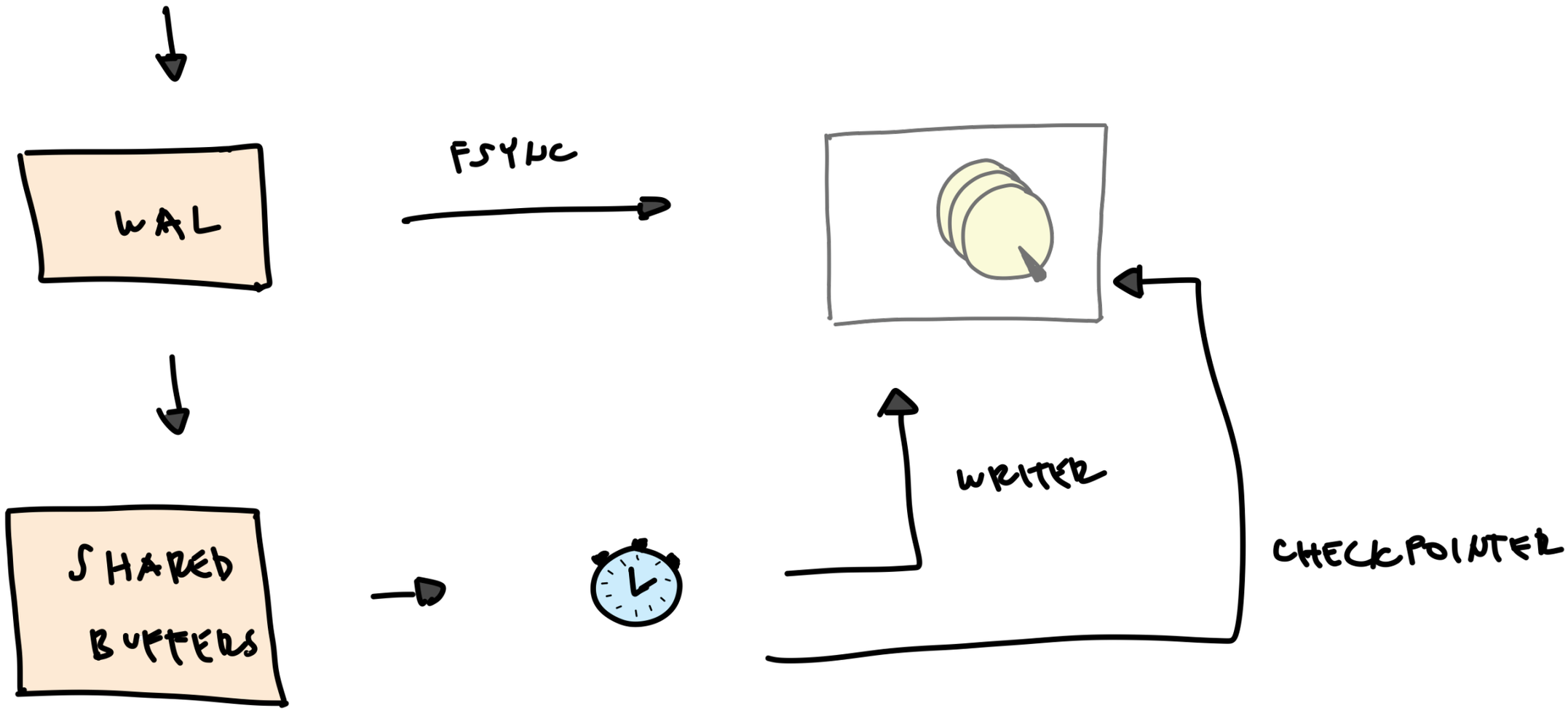


FSYNC

Database can be *reasonably* sure that the data is safely persisted on disk only after doing fsync.

```
fsync() transfers ("flushes") all modified in-core data of (i.e.,
   modified buffer cache pages for) the file referred to by the file
   descriptor fd to the disk device (or other permanent storage device)
   so that all changed information can be retrieved even if the system
   crashes or is rebooted.  This includes writing through or flushing a
   disk cache if present.  The call blocks until the device reports that
   the transfer has completed.
```

INSERT INTO ... VALUES ...



WAL

- Write Ahead Log
- On data modification
  - write WAL, fsync WAL
  - write data pages (to shared buffers)

CHECKPOINTER

- At least N seconds, or when WAL size > X do a checkpoint
- Checkpoint = write dirty pages from shared buffers to disk
- Checkpoint = a mark that WAL is in sync with persistent storage
- Important for recovery

(BACKGROUND) WRITER PROCESS

- Periodically write dirty pages from shared buffers to disk
- Smart, e.g.
  - tries not to write "hot" pages too frequently
- Helps spread checkpointer IO load

POSTGRESQL PROCESSES

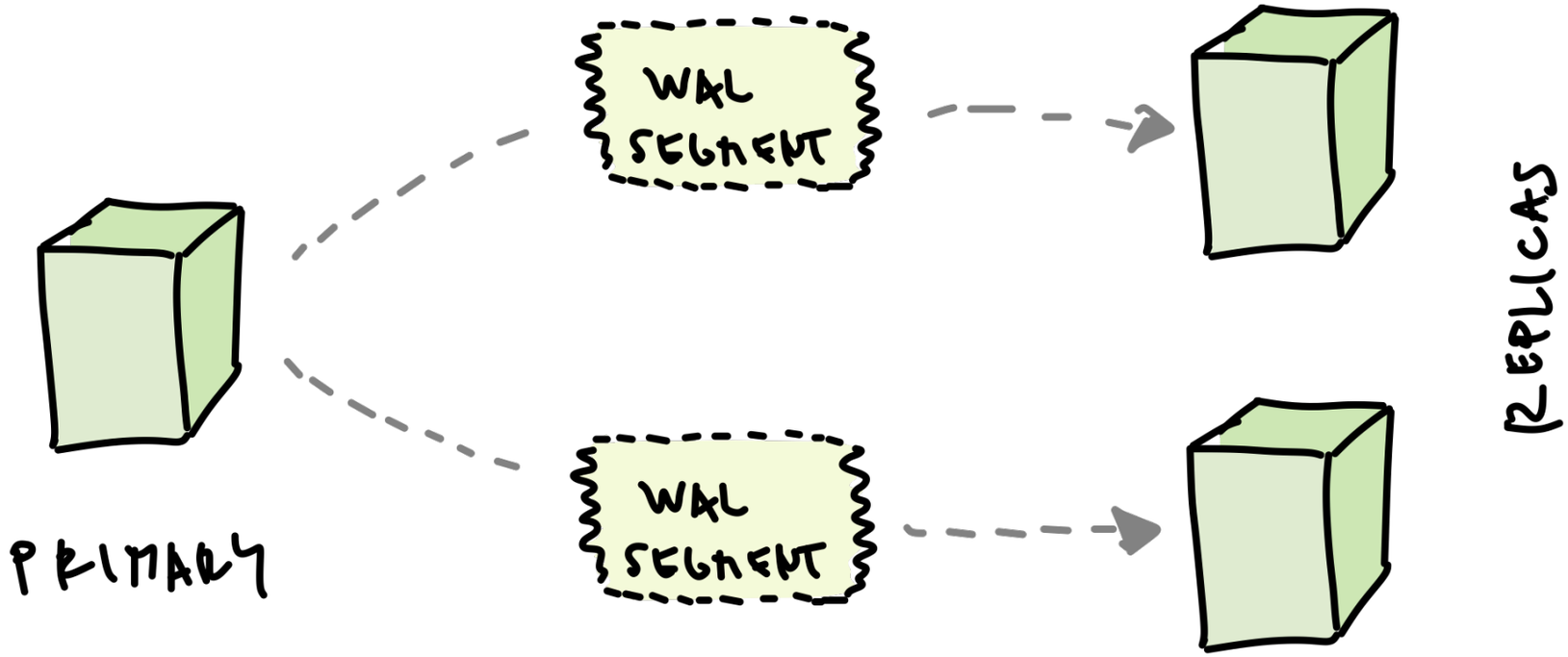
```
00:40:00 in ~ → ps auxf | grep postgres
systemd+ 30175  0.0  0.0 223664 19876 ?        Ss   nov16   0:00      \_ postgres
systemd+ 30326  0.0  0.0 223852  7380 ?        Ss   nov16   0:00          \_ postgres: checkpointer process
systemd+ 30327  0.0  0.0 223664  5452 ?        Ss   nov16   0:00          \_ postgres: writer process
systemd+ 30328  0.0  0.0 223664  3960 ?        Ss   nov16   0:00          \_ postgres: wal writer process
systemd+ 30329  0.0  0.0 224500  6696 ?        Ss   nov16   0:00          \_ postgres: autovacuum launcher process
systemd+ 30330  0.0  0.0  79476  4100 ?        Ss   nov16   0:00          \_ postgres: stats collector process
systemd+ 30414  0.0  0.0 228632 16748 ?        Ss   nov16   0:00          \_ postgres: postgres oz 172.17.0.1(34620) idle
```

RECOVERY

- WAL = append only
- Roll forward recovery (REDO)
  - Find last checkpoint
  - Replay WAL from that checkpoint

REPLICATION

- Ship WAL segments to read-only replicas
- Hot standby/master-slave replication
- Replication lag / eventual consistency



DATA-SAFETY TRADEOFFS

- Data safety vs. DML performance
- Normal operation
  - write WAL
  - fsync WAL
  - write data
- asynchronous commit
  - write WAL
  - write data

ASYNCHRONOUS COMMIT

- In the event of crash, recent transactions may be lost
- Acceptable trade-off for many applications

BENCHMARK

(You will do this in labs)

```
insert    919.723  (± 4.0%) i/s
```

with synchronous_commit = off

```
insert      6.968k (± 4.4%) i/s
```

TRANSACTION ISOLATION

TRANSACTION ISOLATION LEVELS

- Read uncommited
  - Enables dirty read
- **Read commited** - default
  - Non-repeatable reads
- Repeatable reads
  - Lock selected data for writing
  - Range locks are not acquired, phantom reads are possible
- Serializable
  - Also acquire range locks

READ COMMITED

- Each transaction sees the database as it was at the time that transaction started
- Requires a database snapshot
  - .. or a more clever solution

MVCC

- Concurrency control mechanism - avoid locks (contentions)
- Data is not overwritten - all modifications create new versions
- Transactions create snapshots of the data
  - Inexpensive - all row versions exist at the same time, with visibility governed by xmin and xmax
- Periodic vacuum process cleans up old version, which are no longer visible by any running transaction

MVCC

- Each row (version)
  - xmin - ID of the transaction that *created* the row version
  - xmax - ID of the transaction that *deleted* the row version

- Every snapshot also includes xip - list of transactions in progress

- Low-level updates are not possible
  - Always create a new (version of the) row
    - Mark the old version as deleted

```
oz=# select txid_current();
 txid_current
--------------
          733
(1 row)

oz=# select txid_current();
 txid_current
--------------
          734
(1 row)

oz=# select txid_current();
 txid_current
--------------
          735
(1 row)
```

```
oz=# begin;
BEGIN
oz=# select txid_current();
 txid_current
--------------
          736
(1 row)

oz=# select txid_current();
 txid_current
--------------
          736
(1 row)
```

```
oz=# begin;
BEGIN
oz=# insert into test(name) values('pdt');
INSERT 0 1
oz=# select xmin, xmax, * from test;
 xmin | xmax | id | name
------+------+----+------
  739 |    0 | 34 | pdt
(1 row)

oz=# select txid_current();
 txid_current
--------------
          739
(1 row)

oz=# commit;
```

SESSION #1

```
oz=# begin;
BEGIN
oz=# update test set name = 'oop';
UPDATE 1
oz=# select xmin, xmax, * from test;
 xmin | xmax | id | name
------+------+----+------
  741 |    0 | 34 | oop
(1 row)

oz=# select txid_current();
 txid_current
--------------
          741
(1 row)
```

SESSION #2

```
oz=# select xmin, xmax, * from test;
 xmin | xmax | id | name
------+------+----+------
  739 |  741 | 34 | pdt
(1 row)
```

BACK TO SESSION #1

```
commit;
```

AND BACK TO SESSION #2

```
oz=# select xmin, xmax, * from test;
 xmin | xmax | id | name
------+------+----+------
  741 |    0 | 34 | oop
(1 row)
```

```
create extension pageinspect;

SELECT * FROM
heap_page_items(get_raw_page('test', 0));
```

```
oz=# create extension pageinspect;
CREATE EXTENSION
oz=#
SELECT * FROM heap_page_items(get_raw_page('pg_class', 0));
oz=#
SELECT * FROM heap_page_items(get_raw_page('test', 0));
 lp | lp_off | lp_flags | lp_len | t_xmin | t_xmax | t_field3 | t_ctid | t_infomask2 | t_infomask | t_hoff | t_bits | t_oid
----+--------+----------+--------+--------+--------+----------+--------+-------------+------------+--------+--------+-------
  1 |   8160 |        1 |     32 |    739 |    741 |        0 | (0,3)  |       16386 |       1282 |     24 |        |
  2 |   8128 |        1 |     32 |    740 |      0 |        0 | (0,2)  |       32770 |      10754 |     24 |        |
  3 |   8096 |        1 |     32 |    741 |      0 |        0 | (0,3)  |       32770 |      10498 |     24 |        |
(3 rows)
```

VACUUM PROCESS

- vacuum
  - cleanup snapshots that are no longer visible to any transaction
- vacuum full
  - rewrites data to new file - allows OS to reclaim disk space
- vacuum analyze
  - runs vacuum and also collects data statistics for query planner

WHAT YOU SHOULD KNOW

- Understand how caching impacts data durability
- Understand WAL, checkpoints, recovery process and replication
- Understand the tradoffs of using WAL vs. no WAL, understand why WAL is "slow" (fsync)
- Understand that MVVC tracks row versions and maintains visibility boundaries