The Wonderful World of WAL

BRUCE MOMJIAN

The write-ahead log, or WAL, provides key features of Postgres, which are covered in this presentation.

https://momjian.us/presentations

Last updated: March 2026

Outline

1. Inside the write-ahead log (WAL)
2. Crash recovery
3. Point-in-time recovery
4. Physical replication
5. Logical replication
6. Replication slots

1. Inside the Write-Ahead Log (WAL)

The Postgres write-ahead log, or WAL, is basically a change-log for the database. It enables several important Postgres features: crash recovery, point-in-time recovery, and binary and logical replication.

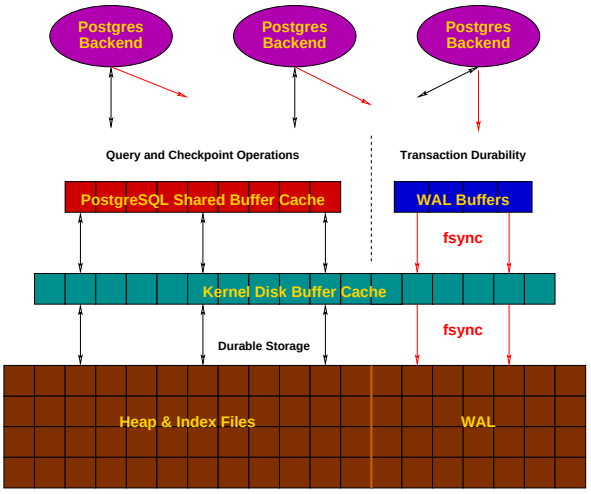
https://www.postgresql.org/docs/current/wal-intro.html

WAL History

- 2001, PG 7.1: Write-ahead log
- 2005, PG 8.0: Point-in-time recovery
- 2010, PG 9.0: Streaming physical replication
- 2014, PG 9.4: Replication slots
- 2017, PG 10: Logical replication

https://bucardo.org/postgres_all_versions

Normal WAL Operation



https://www.postgresql.org/docs/current/wal-reliability.html

Install pg_walinspect

```
CREATE EXTENSION IF NOT EXISTS pg_walinspect;
```

All queries used in this presentation are available at https://momjian.us/main/writings/pgsql/wal.sql.

https://www.postgresql.org/docs/current/pgwalinspect.html

Create Function to Remap *resource_manager*

```
-- adjust resource_manager for historical names
CREATE FUNCTION remap_resource_manager(resource_manager TEXT)
RETURNS TEXT AS
$$
        SELECT  CASE resource_manager
                WHEN 'CLOG' THEN 'Pg_xact'
                WHEN 'XLOG' THEN 'Pg_wal'
                ELSE resource_manager
                END
$$ LANGUAGE SQL;
```

Pg_walinspect with *resource_manager*

```
-- adjust resource_manager for historical names
WITH wal_records AS (
        SELECT  remap_resource_manager(resource_manager) AS res_manager,
                COUNT(*) AS count,
                SUM(record_length) as size
        FROM pg_get_wal_records_info('0/01000000', 'FFFFFFFF/FFFFFFFF')
        GROUP BY 1
)
SELECT  res_manager,
        count,
        round(count * 100.0 / (SUM(count) OVER ()), 1) AS "%",
        round(size * 100.0 / (SUM(size) OVER ()), 1) AS "size%"
FROM wal_records
ORDER BY res_manager;
```

Output of Pg_walinspect with *resource_manager*

res_manager	count	%	size%
Btree	15118	57.7	12.9
Database	3	0.0	0.0
Heap	5638	21.5	15.9
Heap2	2686	10.3	17.1
Pg_wal	1263	4.8	52.0
Pg_xact	1	0.0	0.0
RelMap	1	0.0	0.0
Standby	414	1.6	0.2
Storage	310	1.2	0.1
Transaction	756	2.9	1.7

https://www.postgresql.org/docs/current/pgwalinspect.html

Pg_walinspect with added *record_type*

```
-- add record_type
WITH wal_records AS (
        SELECT  remap_resource_manager(resource_manager) AS res_manager,
                record_type,
                COUNT(*) AS count,
                SUM(record_length) as size
        FROM pg_get_wal_records_info('0/01000000', 'FFFFFFFF/FFFFFFFF')
        GROUP BY 1, 2
)
SELECT  res_manager,
        record_type,
        count,
        round(count * 100.0 / (SUM(count) OVER ()), 1) AS "%",
        round(size * 100.0 / (SUM(size) OVER ()), 1) AS "size%"
FROM wal_records
ORDER BY res_manager, record_type;
```

Pg_walinspect: Btree

res_manager	record_type	count	%	size%
Btree	DEDUP	50	0.2	0.0
Btree	DELETE	9	0.0	0.0
Btree	INSERT_LEAF	14888	56.8	11.9
Btree	INSERT_UPPER	43	0.2	0.0
Btree	NEWROOT	30	0.1	0.0
Btree	SPLIT_L	15	0.1	0.4
Btree	SPLIT_R	38	0.1	0.5
Btree	VACUUM	45	0.2	0.0

Pg_walinspect: Database

```
 res_manager |     record_type      | count |  %   | size%
-------------+----------------------+-------+------+-------
 Database    | CREATE_FILE_COPY     |     2 |  0.0 |   0.0
 Database    | CREATE_WAL_LOG       |     1 |  0.0 |   0.0
```

Pg_walinspect: Heap

res_manager	record_type	count	%	size%
Heap	DELETE	37	0.1	0.0
Heap	HOT_UPDATE	393	1.5	0.4
Heap	INPLACE	211	0.8	0.8
Heap	INSERT	4326	16.5	9.1
Heap	INSERT+INIT	156	0.6	1.5
Heap	LOCK	258	1.0	2.8
Heap	UPDATE	239	0.9	1.1
Heap	UPDATE+INIT	18	0.1	0.2

Pg_walinspect: Heap2

res_manager	record_type	count	%	size%
Heap2	MULTI_INSERT	1665	6.4	2.6
Heap2	MULTI_INSERT+INIT	52	0.2	0.6
Heap2	PRUNE_ON_ACCESS	269	1.0	0.1
Heap2	PRUNE_VACUUM_CLEANUP	49	0.2	0.0
Heap2	PRUNE_VACUUM_SCAN	256	1.0	0.3
Heap2	VISIBLE	395	1.5	13.4

“Heap2” is the same as “Heap” and is used to support additional heap record types.

Pg_walinspect: Pg_wal

res_manager	record_type	count	%	size%
Pg_wal	CHECKPOINT_ONLINE	4	0.0	0.0
Pg_wal	CHECKPOINT_REDO	4	0.0	0.0
Pg_wal	CHECKPOINT_SHUTDOWN	3	0.0	0.0
Pg_wal	FPI	1106	4.2	42.5
Pg_wal	FPI_FOR_HINT	142	0.5	9.5
Pg_wal	NEXTOID	4	0.0	0.0

Pg_walinspect: Pg_xact

res_manager	record_type	count	%	size%
Pg_xact	ZEROPAGE	1	0.0	0.0

Pg_walinspect: RelMap

res_manager	record_type	count	%	size%
RelMap	UPDATE	1	0.0	0.0

Pg_walinspect: Standby

res_manager	record_type	count	%	size%
Standby	INVALIDATIONS	139	0.5	0.1
Standby	LOCK	271	1.0	0.1
Standby	RUNNING_XACTS	4	0.0	0.0

Pg_walinspect: Storage

res_manager	record_type	count	%	size%
Storage	CREATE	310	1.2	0.1

Pg_walinspect: Transaction

res_manager	record_type	count	%	size%
Transaction	COMMIT	756	2.9	1.7

Create Table

```
-- lsn (log sequence numbers) values represent locations in the WAL
SELECT pg_current_wal_lsn() AS start_lsn
\gset

-- PRIMARY KEY creates an index
CREATE TABLE wal_test (x INTEGER PRIMARY KEY);
```

WAL Records for CREATE TABLE

```
-- uses psql variable start_lsn
WITH wal_records AS (
        SELECT  remap_resource_manager(resource_manager) AS res_manager,
                record_type,
                COUNT(*) AS count,
                SUM(record_length) as size
        FROM pg_get_wal_records_info(:'start_lsn', 'FFFFFFFF/FFFFFFFF')
        GROUP BY 1, 2
)
SELECT  res_manager,
        record_type,
        count,
        round(count * 100.0 / (SUM(count) OVER ()), 1) AS "%",
        round(size * 100.0 / (SUM(size) OVER ()), 1) AS "size%"
FROM wal_records
ORDER BY res_manager, record_type;
```

WAL Records for CREATE TABLE

res_manager	record_type	count	%	size%
Btree	INSERT_LEAF	50	63.3	41.9
Heap	INPLACE	2	2.5	6.3
Heap	INSERT	7	8.9	17.2
Heap2	MULTI_INSERT	9	11.4	19.4
Heap2	PRUNE_ON_ACCESS	5	6.3	3.1
Pg_wal	FPI	1	1.3	1.7
Standby	LOCK	2	2.5	1.0
Storage	CREATE	2	2.5	1.0
Transaction	COMMIT	1	1.3	8.4

INSERT

```
SELECT pg_current_wal_lsn() AS start_lsn
\gset

INSERT INTO wal_test VALUES (1);
```

WAL Records for INSERT

```
WITH wal_records AS (
        SELECT  remap_resource_manager(resource_manager) AS res_manager,
                record_type,
                COUNT(*) AS count,
                SUM(record_length) as size
        FROM pg_get_wal_records_info(:'start_lsn', 'FFFFFFFF/FFFFFFFF')
        GROUP BY 1, 2
)
SELECT  res_manager,
        record_type,
        count,
        round(count * 100.0 / (SUM(count) OVER ()), 1) AS "%",
        round(size * 100.0 / (SUM(size) OVER ()), 1) AS "size%"
FROM wal_records
ORDER BY res_manager, record_type;
```

WAL Records for INSERT

```
 res_manager | record_type | count |  %   | size%
-------------+-------------+-------+------+-------
 Btree       | INSERT_LEAF |     1 | 25.0 |  25.9
 Btree       | NEWROOT     |     1 | 25.0 |  36.4
 Heap        | INSERT+INIT |     1 | 25.0 |  23.9
 Transaction | COMMIT      |     1 | 25.0 |  13.8
```

Update

```
SELECT pg_current_wal_lsn() AS start_lsn
\gset

UPDATE wal_test SET x = 2;
```

WAL Records for UPDATE

```
WITH wal_records AS (
        SELECT  remap_resource_manager(resource_manager) AS res_manager,
                record_type,
                COUNT(*) AS count,
                SUM(record_length) as size
        FROM pg_get_wal_records_info(:'start_lsn', 'FFFFFFFF/FFFFFFFF')
        GROUP BY 1, 2
)
SELECT  res_manager,
        record_type,
        count,
        round(count * 100.0 / (SUM(count) OVER ()), 1) AS "%",
        round(size * 100.0 / (SUM(size) OVER ()), 1) AS "size%"
FROM wal_records
ORDER BY res_manager, record_type;
```

WAL Records for UPDATE

```
 res_manager | record_type | count |  %   | size%
-------------+-------------+-------+------+-------
 Btree       | INSERT_LEAF |     1 | 33.3 |  38.3
 Heap        | UPDATE      |     1 | 33.3 |  41.3
 Transaction | COMMIT      |     1 | 33.3 |  20.4
```

DELETE

```
SELECT pg_current_wal_lsn() AS start_lsn
\gset

DELETE FROM wal_test;
```

WAL Records for DELETE

```
WITH wal_records AS (
        SELECT  remap_resource_manager(resource_manager) AS res_manager,
                record_type,
                COUNT(*) AS count,
                SUM(record_length) as size
        FROM pg_get_wal_records_info(:'start_lsn', 'FFFFFFFF/FFFFFFFF')
        GROUP BY 1, 2
)
SELECT  res_manager,
        record_type,
        count,
        round(count * 100.0 / (SUM(count) OVER ()), 1) AS "%",
        round(size * 100.0 / (SUM(size) OVER ()), 1) AS "size%"
FROM wal_records
ORDER BY res_manager, record_type;
```

WAL Records for DELETE

```
 res_manager | record_type | count |  %   | size%
-------------+-------------+-------+------+-------
 Heap        | DELETE      |     1 | 50.0 |  61.4
 Transaction | COMMIT      |     1 | 50.0 |  38.6
```

Drop Table

```
SELECT pg_current_wal_lsn() AS start_lsn
\gset

DROP TABLE wal_test;
```

WAL Records for DROP TABLE

```
WITH wal_records AS (
        SELECT  remap_resource_manager(resource_manager) AS res_manager,
                record_type,
                COUNT(*) AS count,
                SUM(record_length) as size
        FROM pg_get_wal_records_info(:'start_lsn', 'FFFFFFFF/FFFFFFFF')
        GROUP BY 1, 2
)
SELECT  res_manager,
        record_type,
        count,
        round(count * 100.0 / (SUM(count) OVER ()), 1) AS "%",
        round(size * 100.0 / (SUM(size) OVER ()), 1) AS "size%"
FROM wal_records
ORDER BY res_manager, record_type;
```

WAL Records for DROP TABLE

res_manager	record_type	count	%	size%
Heap	DELETE	21	84.0	57.6
Heap2	PRUNE_ON_ACCESS	1	4.0	2.9
Standby	LOCK	2	8.0	4.3
Transaction	COMMIT	1	4.0	35.2

INSERT a Second Row

```
CREATE TABLE wal_test (x INTEGER PRIMARY KEY);
INSERT INTO wal_test VALUES (1);

SELECT pg_current_wal_lsn() AS start_lsn
\gset

INSERT INTO wal_test VALUES (2);
```

WAL Records for Second INSERT

```
WITH wal_records AS (
        SELECT  remap_resource_manager(resource_manager) AS res_manager,
                record_type,
                COUNT(*) AS count,
                SUM(record_length) as size
        FROM pg_get_wal_records_info(:'start_lsn', 'FFFFFFFF/FFFFFFFF')
        GROUP BY 1, 2
)
SELECT  res_manager,
        record_type,
        count,
        round(count * 100.0 / (SUM(count) OVER ()), 1) AS "%",
        round(size * 100.0 / (SUM(size) OVER ()), 1) AS "size%"
FROM wal_records
ORDER BY res_manager, record_type;
```

WAL Records for Second INSERT

```
 res_manager | record_type | count |  %   | size%
-------------+-------------+-------+------+-------
 Btree       | INSERT_LEAF |     1 | 33.3 |  40.8
 Heap        | INSERT      |     1 | 33.3 |  37.6
 Transaction | COMMIT      |     1 | 33.3 |  21.7
```

Detailed WAL Records for Second INSERT

```
\x on

SELECT *
FROM pg_get_wal_records_info(:'start_lsn', 'FFFFFFFF/FFFFFFFF')
ORDER BY start_lsn;
```

WAL Record #1

```
-[ RECORD 1 ]----+-----------------------------------------------
start_lsn        | 0/01BB8540
end_lsn          | 0/01BB8580
prev_lsn         | 0/01BB8518
xid              | 766
resource_manager | Heap
record_type      | INSERT
record_length    | 59
main_data_length | 3
fpi_length       | 0
description      | off: 2, flags: 0x00
block_ref        | blkref #0: rel 1663/16384/16399 fork main blk 0
```

WAL Record #2

```
-[ RECORD 2 ]----+-----------------------------------------------
start_lsn        | 0/01BB8580
end_lsn          | 0/01BB85C0
prev_lsn         | 0/01BB8540
xid              | 766
resource_manager | Btree
record_type      | INSERT_LEAF
record_length    | 64
main_data_length | 2
fpi_length       | 0
description      | off: 2
block_ref        | blkref #0: rel 1663/16384/16403 fork main blk 1
```

WAL Record #3

```
-[ RECORD 3 ]----+-----------------------------------------------
start_lsn        | 0/01BB85C0
end_lsn          | 0/01BB85E8
prev_lsn         | 0/01BB8580
xid              | 766
resource_manager | Transaction
record_type      | COMMIT
record_length    | 34
main_data_length | 8
fpi_length       | 0
description      | 2025-10-31 19:00:00.978944-04
block_ref        |
```

More Detailed WAL Record #1

```
SELECT *
FROM pg_get_wal_block_info(:'start_lsn', 'FFFFFFFF/FFFFFFFF')
ORDER BY start_lsn
LIMIT 1;
```

More Detailed WAL Record #1

```
-[ RECORD 1 ]-----+-----------------------
start_lsn         | 0/01BB8540
end_lsn           | 0/01BB8580
prev_lsn          | 0/01BB8518
block_id          | 0
reltablespace     | 1663
reldatabase       | 16384
relfilenode       | 16399
relforknumber     | 0
relblocknumber    | 0
xid               | 766
resource_manager  | Heap
record_type       | INSERT
record_length     | 59
main_data_length  | 3
block_data_length | 10
block_fpi_length  | 0
block_fpi_info    |
description       | off: 2, flags: 0x00
block_data        | \x0100000818000200000
block_fpi_data    |
```

Logical Replication Adds WAL Record #4

```
ALTER SYSTEM SET wal_level = 'logical';

$ pg_ctl restart

SELECT pg_current_wal_lsn() AS start_lsn
\gset

INSERT INTO wal_test values (3);

\x on

SELECT *
FROM pg_get_wal_records_info(:'start_lsn', 'FFFFFFFF/FFFFFFFF')
ORDER BY start_lsn;
```

Logical Replication Adds WAL Record #4

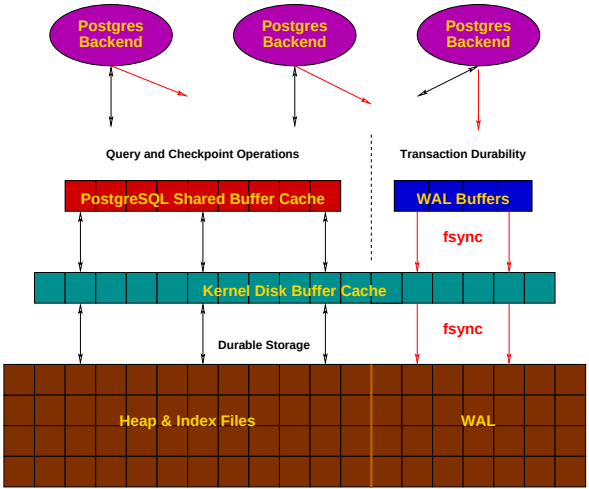
```
-[ RECORD 1 ]----+-----------------------------------------------------------
start_lsn        | 0/01BB86D0
end_lsn          | 0/01BBA090
prev_lsn         | 0/01BB8698
xid              | 0
resource_manager | XLOG
record_type      | FPI_FOR_HINT
record_length    | 6561
main_data_length | 0
fpi_length       | 6512
description      |
block_ref        | blkref #0: rel 1663/16384/1249 fork main blk 16 (FPW); …
```

Set Table to UNLOGGED

```
ALTER TABLE wal_test SET UNLOGGED;

SELECT pg_current_wal_lsn() AS start_lsn
\gset

INSERT INTO wal_test values (4);

\x on

SELECT *
FROM pg_get_wal_records_info(:'start_lsn', 'FFFFFFFF/FFFFFFFF')
ORDER BY start_lsn;
ERROR:  could not find a valid record after 0/01BD2FF8
```

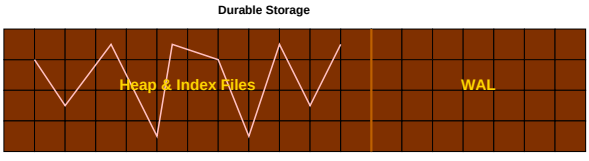
https://www.postgresql.org/docs/current/sql-createtable.html#SQL-CREATETABLE-UNLOGGED

2. Crash Recovery: Normal Operation

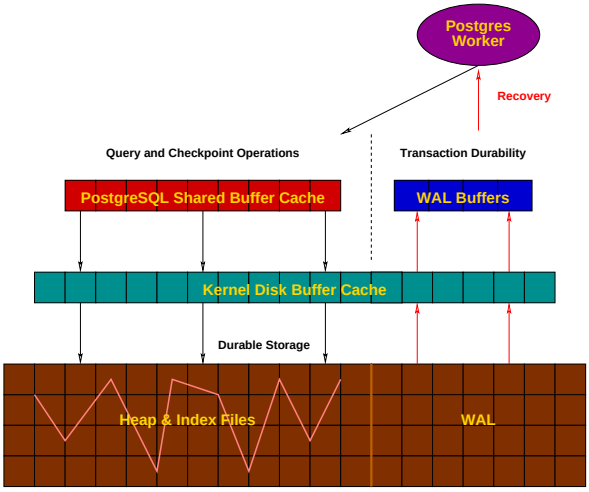


https://www.postgresql.org/docs/current/wal-reliability.html
https://www.cybertec-postgresql.com/en/postgresql-recovery-internals/

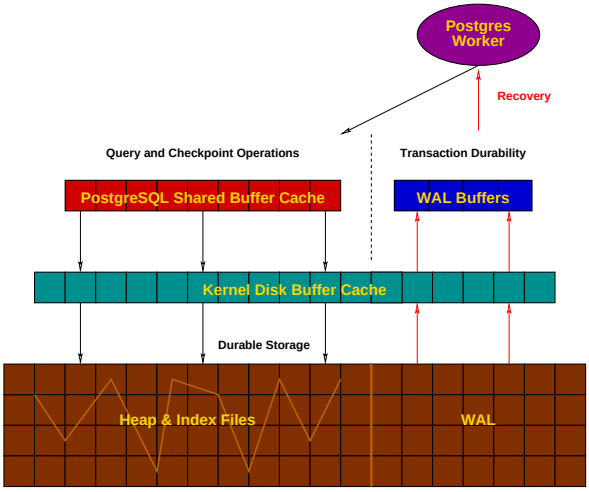
Crash



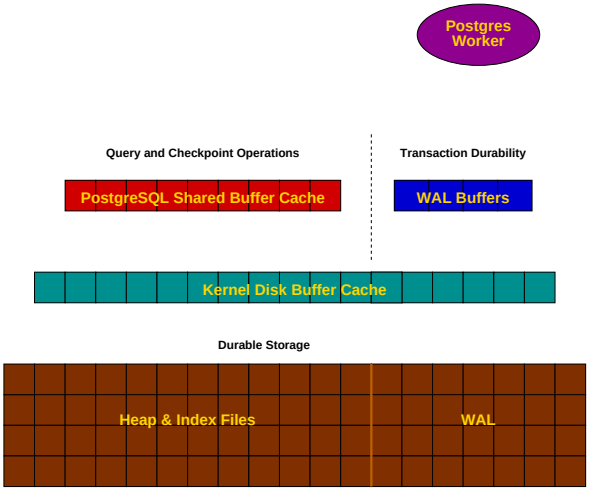
Crash Recovery



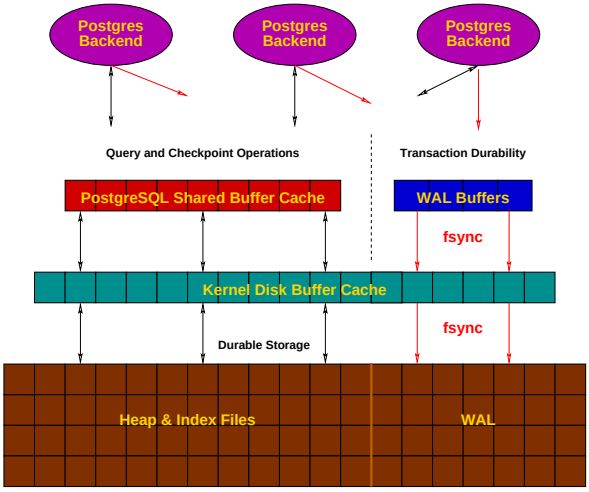
Crash Recovery



Successful Crash Recovery



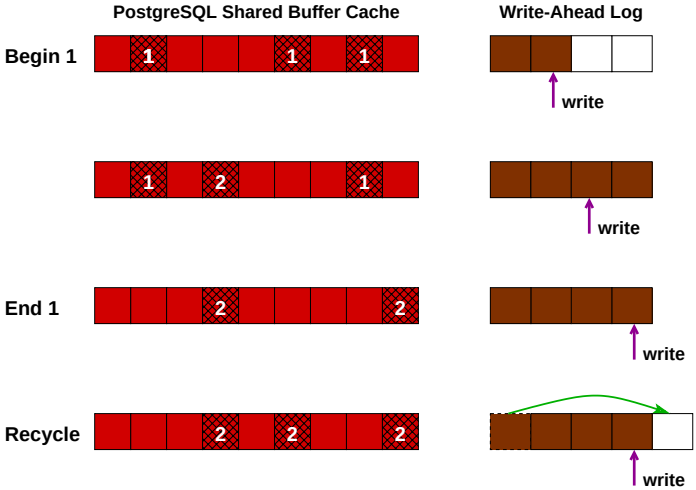
Normal Operation



Storage of Newly Inserted Row

```
CREATE TABLE crash_test (message TEXT);

INSERT INTO crash_test VALUES ('PostgreSQL is awesome.');

$ # only in WAL
$ grep -lR --text 'PostgreSQL is awesome\.' "$PGDATA"
/u/pgsql/data/pg_wal/000000010000000000000001

$ sleep "$(postgres -C checkpoint_timeout)"

$ # after checkpoint, in WAL and heap storage
$ grep -lR --text 'PostgreSQL is awesome\.' "$PGDATA"
/u/pgsql/data/base/16384/16385
/u/pgsql/data/pg_wal/000000010000000000000001
```

Checkpoint



https://www.postgresql.org/docs/current/wal-configuration.html

Simulate a Crash

```
INSERT INTO crash_test VALUES ('Long live relational systems!');

$ # only in WAL
grep -lR --text 'Long live relational systems!' "$PGDATA"
/u/pgsql/data/pg_wal/000000010000000000000001

$ # simulate a crash
$ pg_ctl -m immediate stop
waiting for server to shut down.... done
server stopped

$ # only in WAL
grep -lR --text 'Long live relational systems!' "$PGDATA"
/u/pgsql/data/pg_wal/000000010000000000000001
```

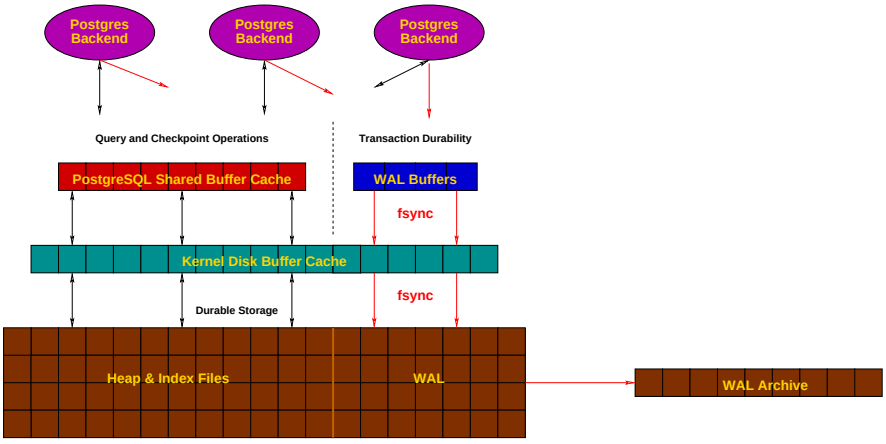
Crash Recovery

```
$ # perform crash recovery
$ pg_ctl start
waiting for server to start....
… LOG:  starting PostgreSQL 19devel …
… LOG:  listening on IPv4 address "127.0.0.1", port 5432
… LOG:  listening on Unix socket "/tmp/.s.PGSQL.5432"
… LOG:  database system was interrupted; last known up at …
… LOG:  database system was not properly shut down; automatic recovery in progress
… LOG:  redo starts at 0/01BFCC38
… LOG:  invalid record length at 0/01BFCE98: expected at least 24, got 0
… LOG:  redo done at 0/01BFCE60 system usage: …
… LOG:  checkpoint starting: end-of-recovery fast wait
… LOG:  checkpoint complete: wrote 1 buffers (0.0%), …
… LOG:  database system is ready to accept connections
 done
server started
```

Successful Crash Recovery

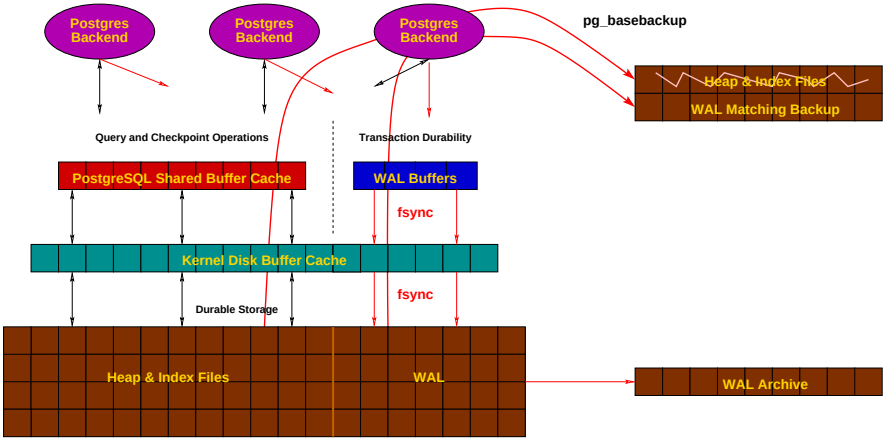
```
$ # in WAL and heap storage
$ grep -lR --text 'Long live relational systems!' "$PGDATA"
/u/pgsql/data/base/16384/16385
/u/pgsql/data/pg_wal/000000010000000000000001

-- and visible in the database
SELECT * FROM crash_test;
            message
------------------------------
 PostgreSQL is awesome.
 Long live relational systems!
```

3. Point-in-Time Recovery (PITR): Continuous Archiving

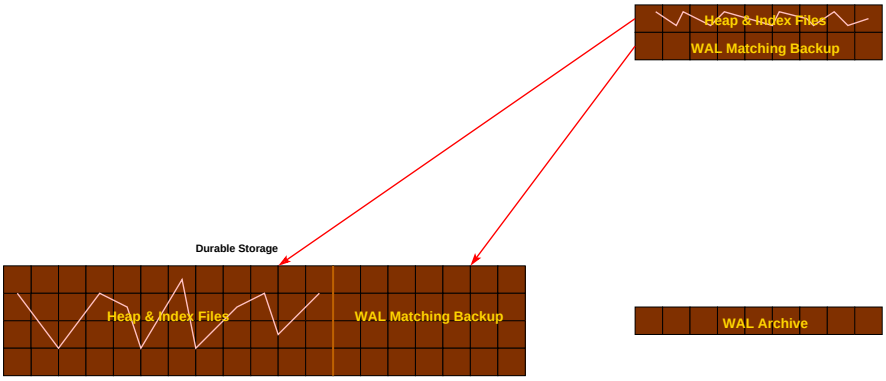


https://www.postgresql.org/docs/current/continuous-archiving.html

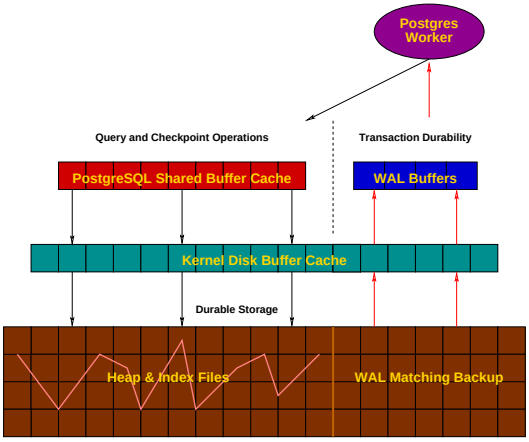
File System Backup



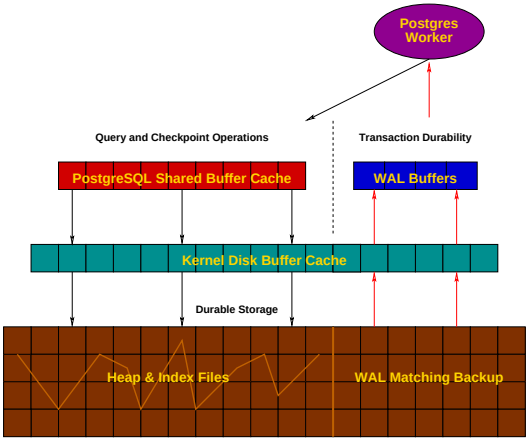
Point-in-Time Recovery: File System Restore



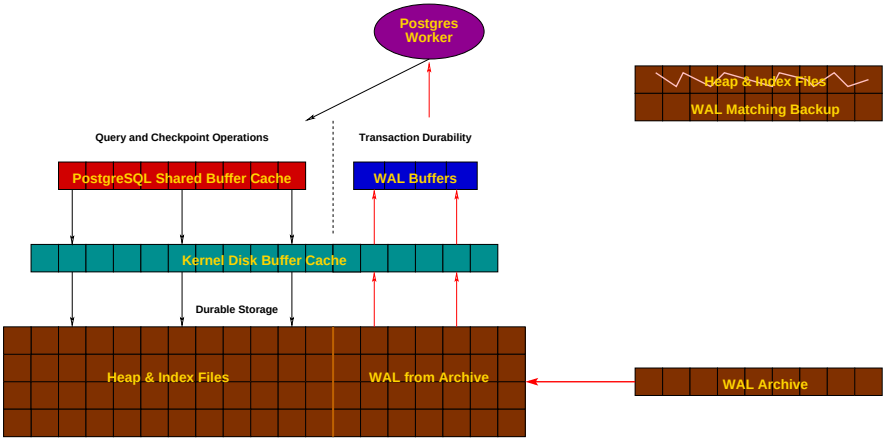
File System Repair & WAL Replay



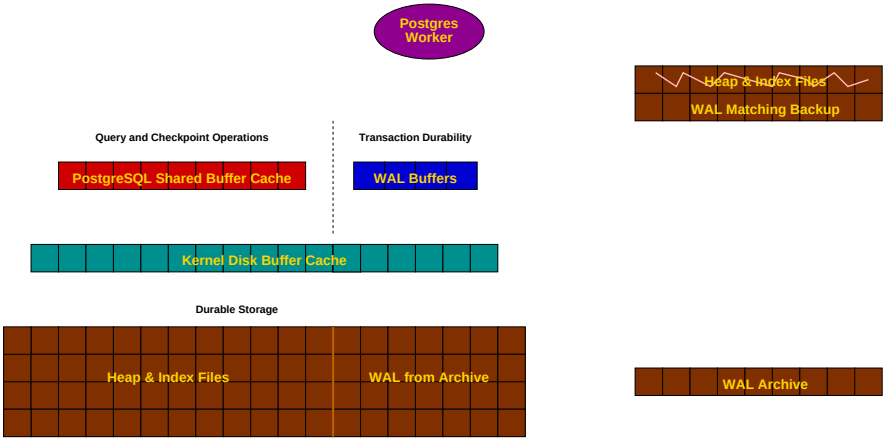
File System Repair & WAL Replay



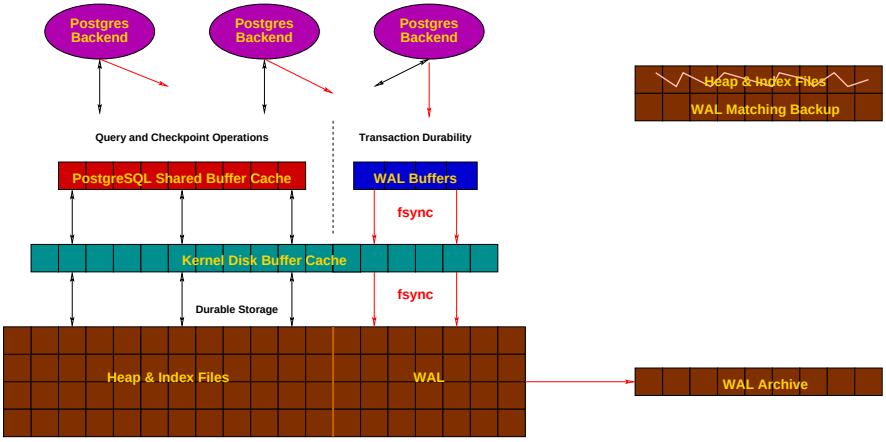
WAL Replay


Postgres Worker
Heap & Index Files
WAL Matching Backup
Query and Checkpoint Operations
Transaction Durability
PostgreSQL Shared Buffer Cache
WAL Buffers
Kernel Disk Buffer Cache
Durable Storage
Heap & Index Files
WAL from Archive
WAL Archive

Successful Point-in-Time Recovery



Normal PITR Operation



Set Up Continuous Archiving

```
-- The operating system user is "postgres".
\! whoami
postgres

-- create archive directory, in the local file system
-- for demonstration purposes
\! mkdir --mode 0700 /u/postgres/archive

-- normally postgresql.conf would be modified
ALTER SYSTEM SET archive_mode = on;

ALTER SYSTEM SET archive_command =
    'test ! -f "/u/postgres/archive/%f" &&
     cp "%p" "/u/postgres/archive/%f"';

$ # restart server
$ pg_ctl restart
waiting for server to shut down.... done
server stopped
```

Test Continuous Archiving

```
SELECT pg_ls_dir('/u/postgres/archive');
 pg_ls_dir
-----------

-- force a new WAL file, which will be immediately archived
SELECT pg_switch_wal();
 pg_switch_wal
---------------
 0/01BACE70

SELECT pg_ls_dir('/u/postgres/archive');
         pg_ls_dir
--------------------------
 000000010000000000000001
```

Test Continuous Archiving Row Propagation

```
CREATE TABLE pitr_test (message TEXT);

INSERT INTO pitr_test VALUES ('Pre-WAL-switch message');

\! grep -lR --text 'Pre-WAL-switch message' "$PGDATA"
/u/pgsql/data/pg_wal/000000010000000000000002

-- not in archive
\! grep -lR --text 'Pre-WAL-switch message' /u/postgres/archive

-- force a new WAL file, which will be immediately archived
SELECT pg_switch_wal();
 pg_switch_wal
---------------
 0/020687B0

-- now in WAL archive
\! grep -lR --text 'Pre-WAL-switch message' /u/postgres/archive
/u/postgres/archive/000000010000000000000002
```

Perform PITR Backup

```
INSERT INTO pitr_test VALUES ('Pre-backup message');

$ pg_basebackup --checkpoint fast -D /u/postgres/backup

-- in WAL archive
\! grep -lR --text 'Pre-backup message' /u/postgres/archive
/u/postgres/archive/000000010000000000000003

-- in heap backup file
\! grep -lR --text 'Pre-backup message' /u/postgres/backup
/u/postgres/backup/base/16384/16389
```

Test Post-PITR Backup Row Propagation

```
INSERT INTO pitr_test VALUES ('Post-backup message');

\! grep -lR --text 'Post-backup message' /u/postgres/archive

SELECT pg_switch_wal();
 pg_switch_wal
---------------
 0/09000338

\! grep -lR --text 'Post-backup message' /u/postgres/archive
/u/postgres/archive/000000010000000000000005

-- It is not in the backup.
\! grep -lR --text 'Post-backup message' /u/postgres/backup
```

Restore PITR

```
$ pg_ctl stop
$ rm -r "$PGDATA"
$ cp -p -R /u/postgres/backup "$PGDATA"
$ # simulates ALTER SYSTEM when the server is down
$ echo "restore_command = 'cp \"/u/postgres/archive/%f\" \"%p\"'" \
    >> "$PGDATA"/postgresql.auto.conf
$ touch "$PGDATA"/recovery.signal
$ pg_ctl start

SELECT * FROM pitr_test;
        message
------------------------
 Pre-WAL-switch message
 Pre-backup message
 Post-backup message
```

Continuous Archiving Is Still Active

```
SHOW archive_mode;
 archive_mode
--------------
 on

SHOW archive_command;
             archive_command
----------------------------------------
 test ! -f "/u/postgres/archive/%f" && .
.cp "%p" "/u/postgres/archive/%f"
```

This happened because the *postgresql.auto.conf* file was restored from the backup.

A New Timeline in *pg_wal*

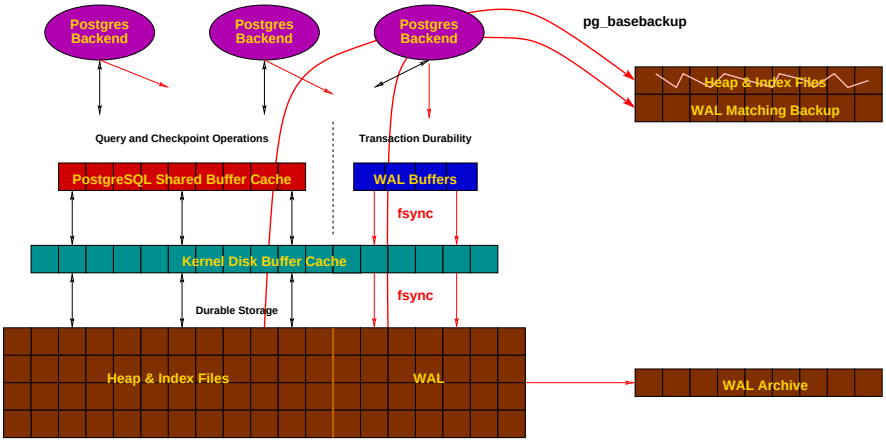
```
SELECT pg_ls_dir('./pg_wal')
ORDER BY 1;
        pg_ls_dir
--------------------------
 0000000200000000000000007
 0000000200000000000000008
 0000000200000000000000009
 000000020000000000000000A
 00000002.history
 archive_status
 summaries
```

https://www.postgresql.org/docs/current/continuous-archiving.html#BACKUP-TIMELINES

A New Timeline in the Archive

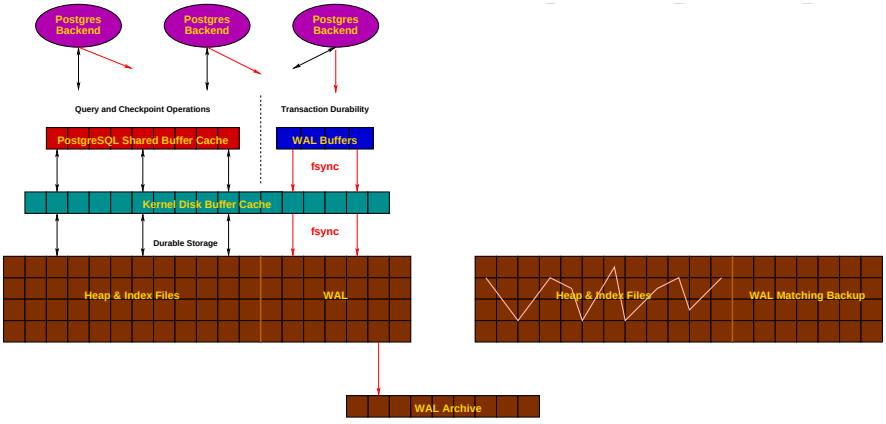
```
SELECT pg_switch_wal();
 pg_switch_wal
---------------
 0/07004088

SELECT pg_ls_dir('/u/postgres/archive')
ORDER BY 1;
                 pg_ls_dir
-----------------------------------------
 000000010000000000000001
 000000010000000000000002
 000000010000000000000003
 000000010000000000000004
 000000010000000000000004.00000028.backup
 000000010000000000000005
 000000010000000000000006
 000000020000000000000007
 00000002.history
```

4. Physical Replication: File System Backup

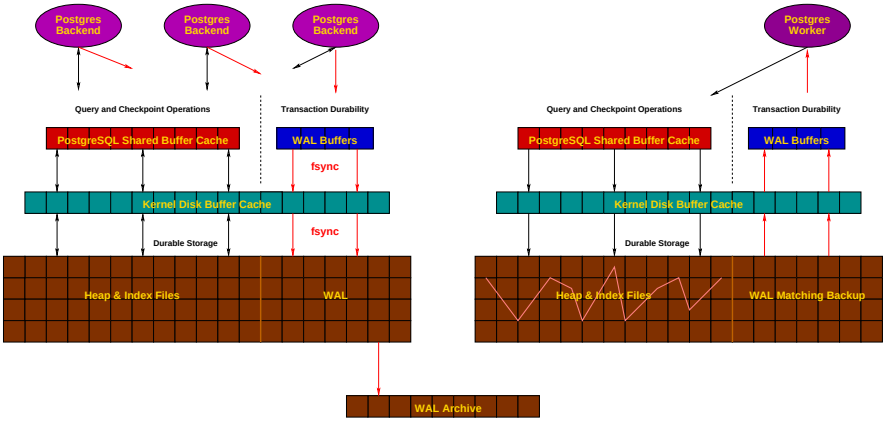


https://www.postgresql.org/docs/current/warm-standby.html

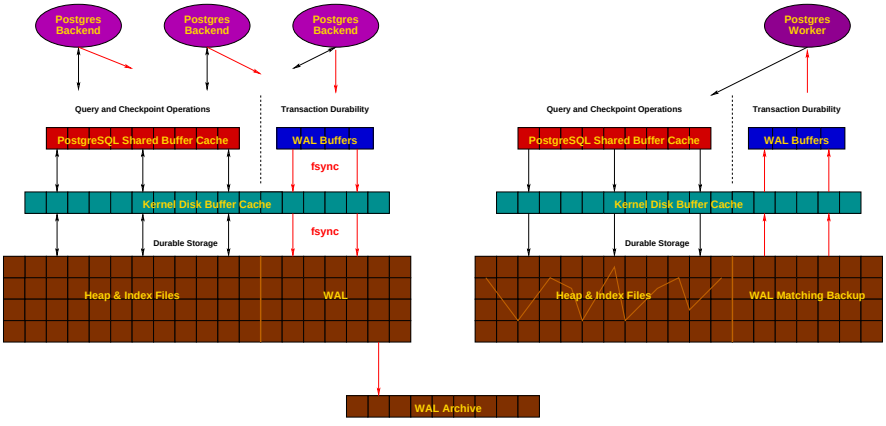
Physical Replication: Setup



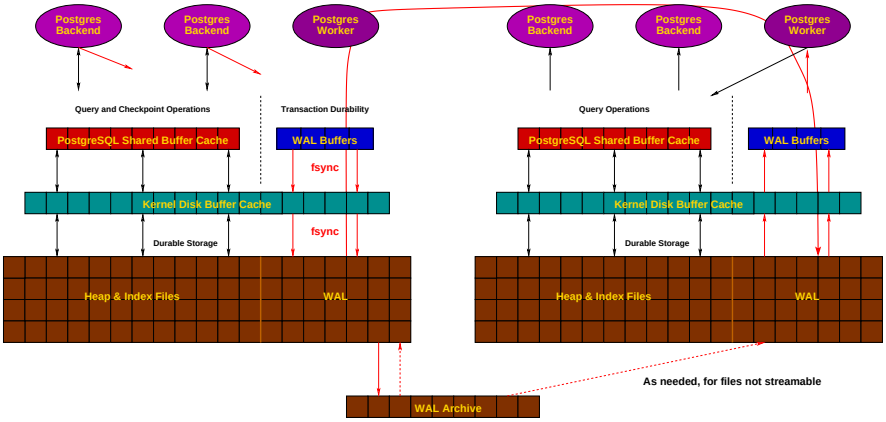
File System Repair & WAL Replay



File System Repair & WAL Replay



Normal Streaming Replay Operation



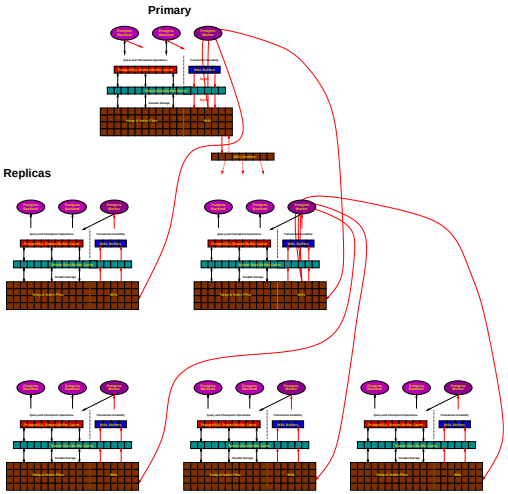
Set Up Physical Replication

```
$ pg_basebackup --checkpoint fast -D /u/postgres/replica

$ # overwrite restored file contents
$ echo "restore_command = 'cp \"/u/postgres/archive/%f\" \"%p\"'" \
    > "$PGDATA"/postgresql.auto.conf

$ echo "primary_conninfo = 'port=5432'" \
    >> /u/postgres/replica/postgresql.auto.conf

$ echo "port=5433" >> /u/postgres/replica/postgresql.auto.conf

$ touch /u/postgres/replica/standby.signal

$ echo 'local  replication  postgres  trust' >> "$PGDATA"/pg_hba.conf'

$ pg_ctl reload
$ pg_ctl -D /u/postgres/replica start
```

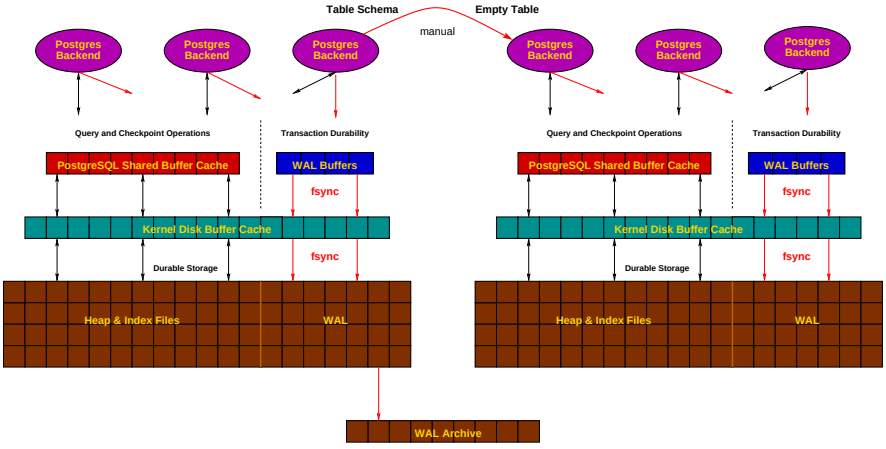
Physical Replication

Primary	Replica
`CREATE TABLE phys_rep_test (message TEXT);`	
`INSERT INTO phys_rep_test VALUES ('one row');`	
	`SELECT * FROM phys_rep_test;`
	one row
`UPDATE phys_rep_test SET message = upper(message);`	
	`SELECT * FROM phys_rep_test;`
	ONE ROW
`DROP TABLE phys_rep_test;`	
	`SELECT * FROM phys_rep_test;`
	ERROR:

Streaming Hierarchy

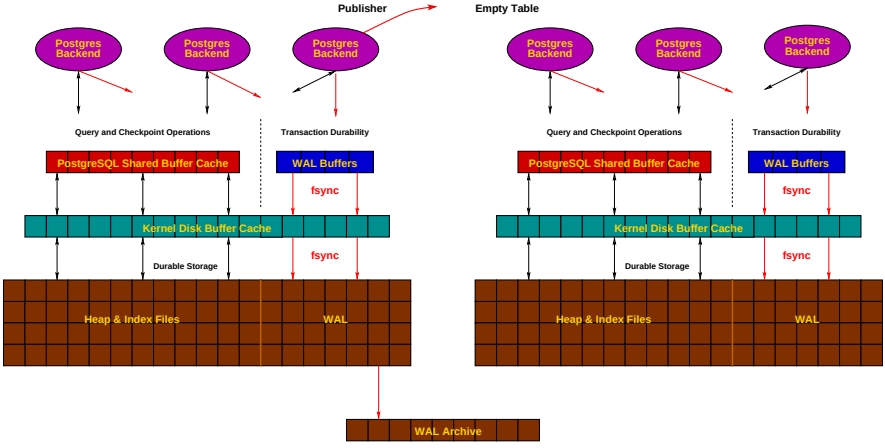


5. Logical Replication: Create Empty Table

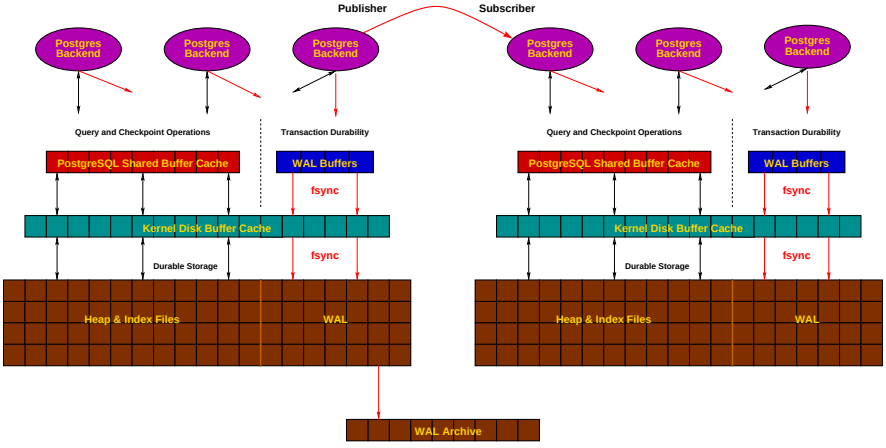


https://www.postgresql.org/docs/current/logical-replication.html
https://www.youtube.com/watch?v=Stx13CLchVs

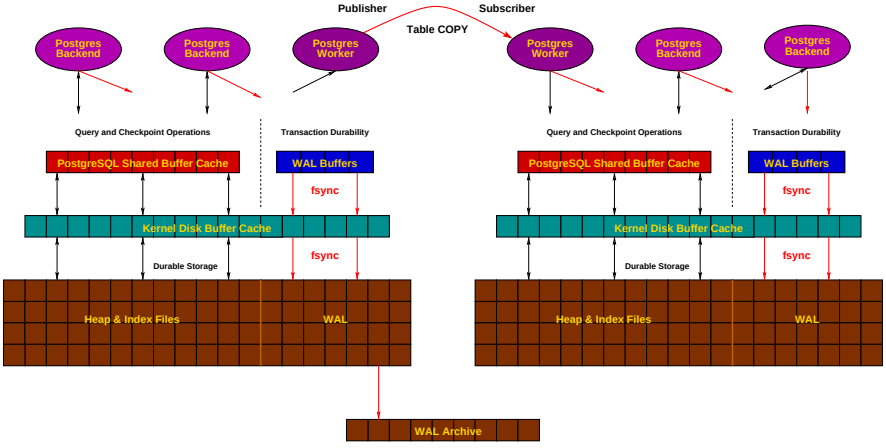
Publisher


Publisher
Empty Table
Postgres Backend
Postgres Backend
Postgres Backend
Postgres Backend
Postgres Backend
Postgres Backend
Query and Checkpoint Operations
Transaction Durability
Query and Checkpoint Operations
Transaction Durability
PostgreSQL Shared Buffer Cache
WAL Buffers
PostgreSQL Shared Buffer Cache
WAL Buffers
fsync
fsync
Kernel Disk Buffer Cache
Kernel Disk Buffer Cache
Durable Storage
fsync
Durable Storage
fsync
Heap & Index Files
WAL
Heap & Index Files
WAL
WAL Archive

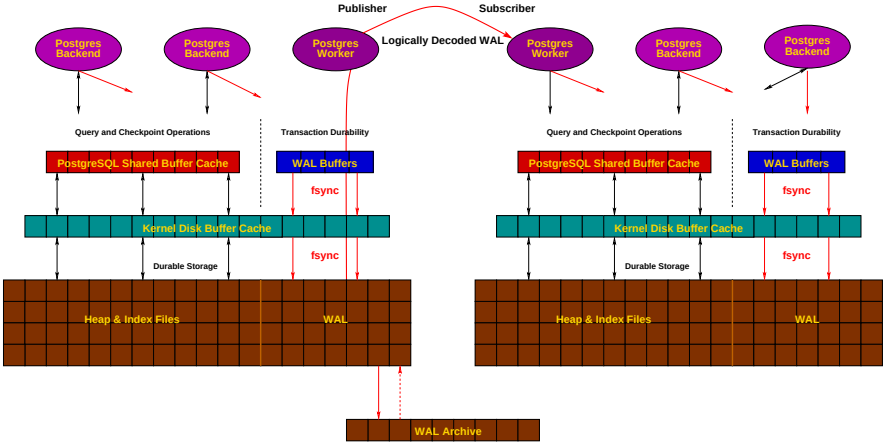
Subscriber


Publisher
Subscriber
Postgres Backend
Postgres Backend
Postgres Backend
Postgres Backend
Postgres Backend
Postgres Backend
Query and Checkpoint Operations
Transaction Durability
Query and Checkpoint Operations
Transaction Durability
PostgreSQL Shared Buffer Cache
WAL Buffers
PostgreSQL Shared Buffer Cache
WAL Buffers
fsync
fsync
Kernel Disk Buffer Cache
Kernel Disk Buffer Cache
Durable Storage
fsync
Durable Storage
fsync
Heap & Index Files
WAL
Heap & Index Files
WAL
WAL Archive

Populate Table



Streaming Logical Changes



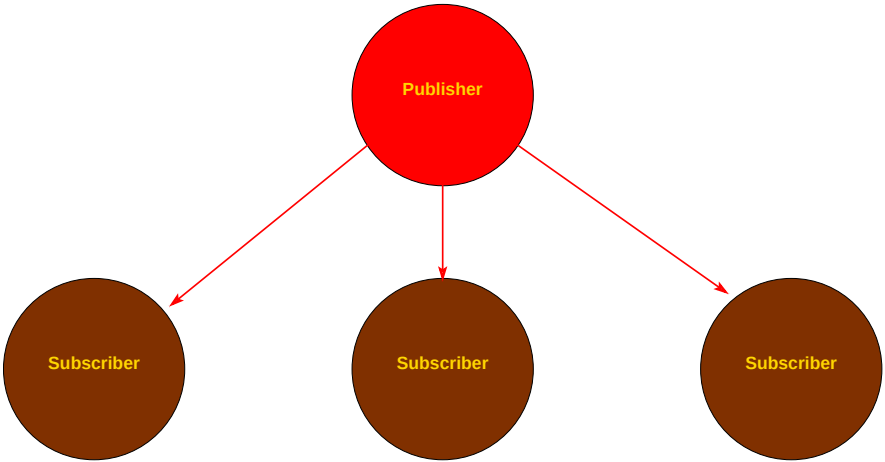
Set Up Logical Replication

```
$ pg_basebackup --checkpoint fast -D /u/postgres/replica

$ # overwrite restored file contents
$ echo "wal_level = 'logical'" > "$PGDATA"/postgresql.auto.conf

$ echo "port=5433" >> /u/postgres/replica/postgresql.auto.conf

$ pg_ctl restart
$ pg_ctl -D /u/postgres/replica start
```

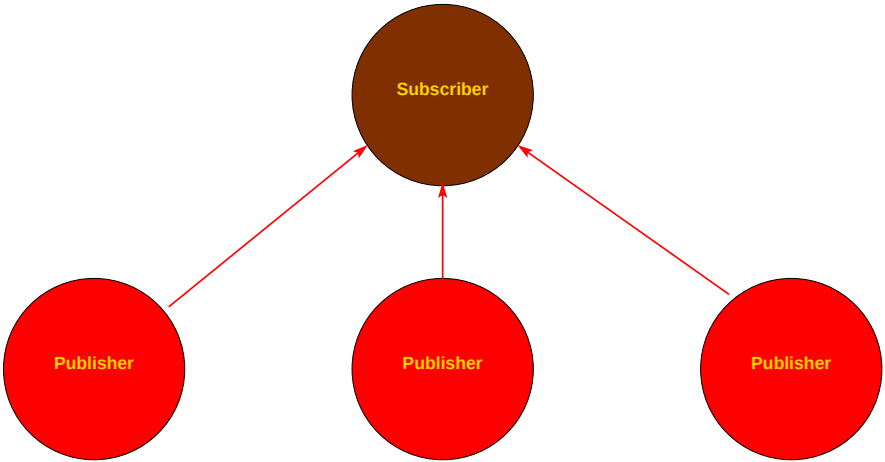
Set Up Table Replication

Primary	Replica
`CREATE TABLE logic_rep_test (message TEXT PRIMARY KEY);`	
`CREATE PUBLICATION pub1 FOR TABLE logic_rep_test;`	
	`CREATE TABLE logic_rep_test (message TEXT PRIMARY KEY);`
	`CREATE SUBSCRIPTION sub1 CONNECTION 'port=5432 dbname=test' PUBLICATION pub1;`
`INSERT INTO logic_rep_test VALUES ('one row');`	
	`SELECT * FROM logic_rep_test;`
	one row
`UPDATE logic_rep_test SET message = upper(message);`	
	`SELECT * FROM logic_rep_test;`
	ONE ROW

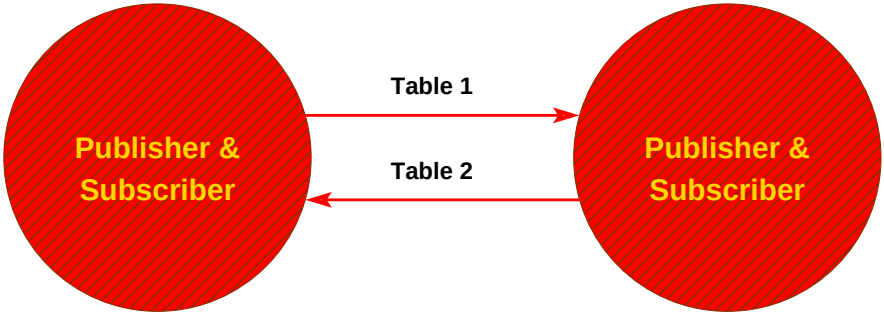
Broadcast



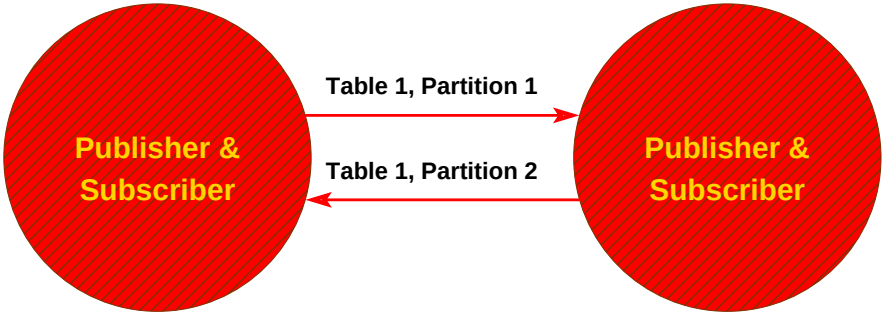
Aggregate



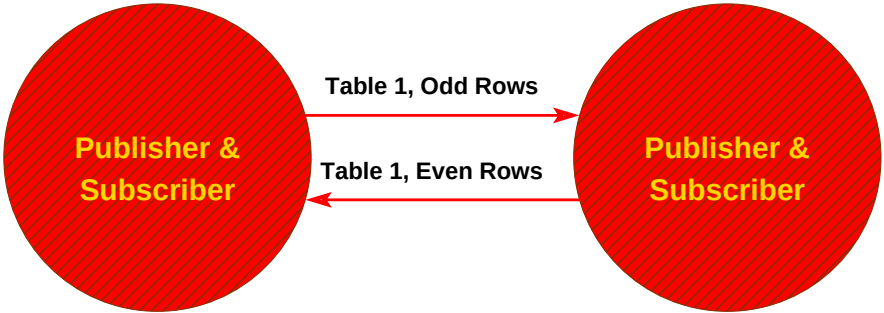
Bidirectional Tables



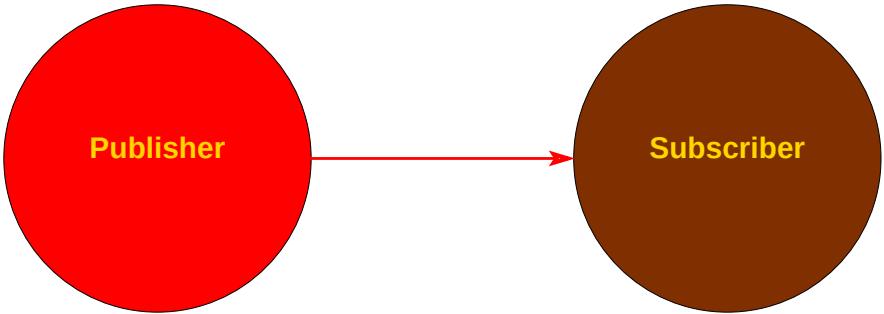
Bidirectional Partitions



Bidirectional Rows

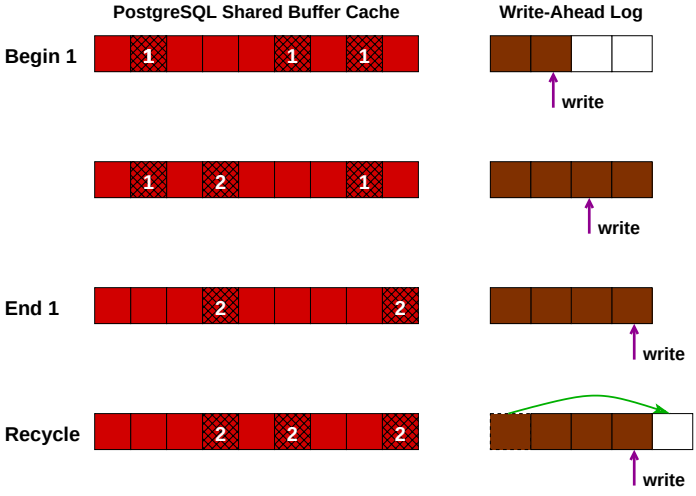


Mismatched Major Versions



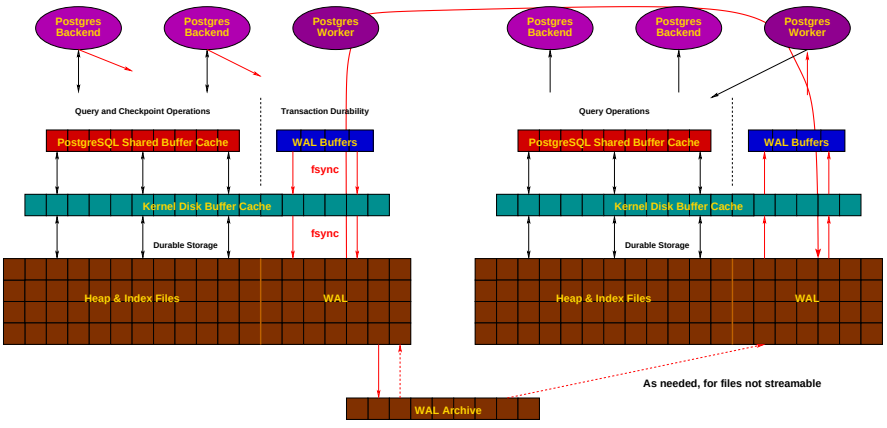
PG Major Version **!= PG Major Version**

6: Replication Slots: Checkpoint Removes WAL

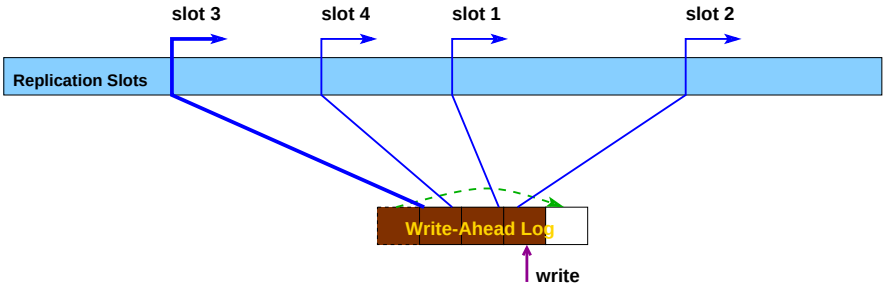


https://www.postgresql.org/docs/current/wal-configuration.html

Keep WAL for Physical Streaming



Replication Slots Retain WAL



https://www.postgresql.org/docs/current/warm-standby.html#STREAMING-REPLICATION-SLOTS
https://hevodata.com/learn/postgresql-replication-slots/

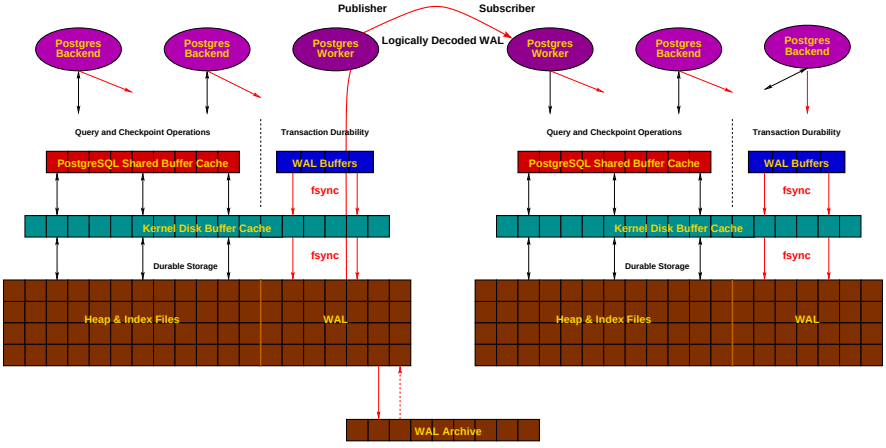
Physical Replication Slot

```
$ # create slot on primary
$ psql -c "
SELECT *
FROM pg_create_physical_replication_slot(
   'streaming_replica_slot'
);" test

$ # record slot on replica
$ echo "primary_slot_name = 'streaming_replica_slot'
    > "$PGDATA"/postgresql.auto.conf
 $ pg_ctl -D /u/postgres/replica reload
```

or

```
$ # Allow the primary to restore WAL files needed for streaming.
$ echo "restore_command = 'cp \"/u/postgres/archive/%f\" \"%p\"'" \
    >> "$PGDATA"/postgresql.auto.conf
$ pg_ctl reload
```

Keep WAL for Logical Streaming



Logical Replication Slot

Subscribers automatically create logical replication slots on the publisher.

```
\x on

SELECT * FROM pg_replication_slots;
-[ RECORD 1 ]-------+-----------
slot_name           | sub1
plugin              | pgoutput
slot_type           | logical
datoid              | 16384
database            | test
temporary           | f
active              | t
active_pid          | 2488029
xmin                |
catalog_xmin        | 770
restart_lsn         | 0/030244B0
confirmed_flush_lsn | 0/030244E8
wal_status          | reserved
safe_wal_size       |
two_phase           | f
two_phase_at        |
inactive_since      |
conflicting         | f
invalidation_reason |
failover            | f
synced              | f
```

Conclusion

https://www.flickr.com/photos/daaggg/