The Magic of Hot Streaming Replication

BRUCE MOMJIAN,
ENTERPRISEDB

October, 2010

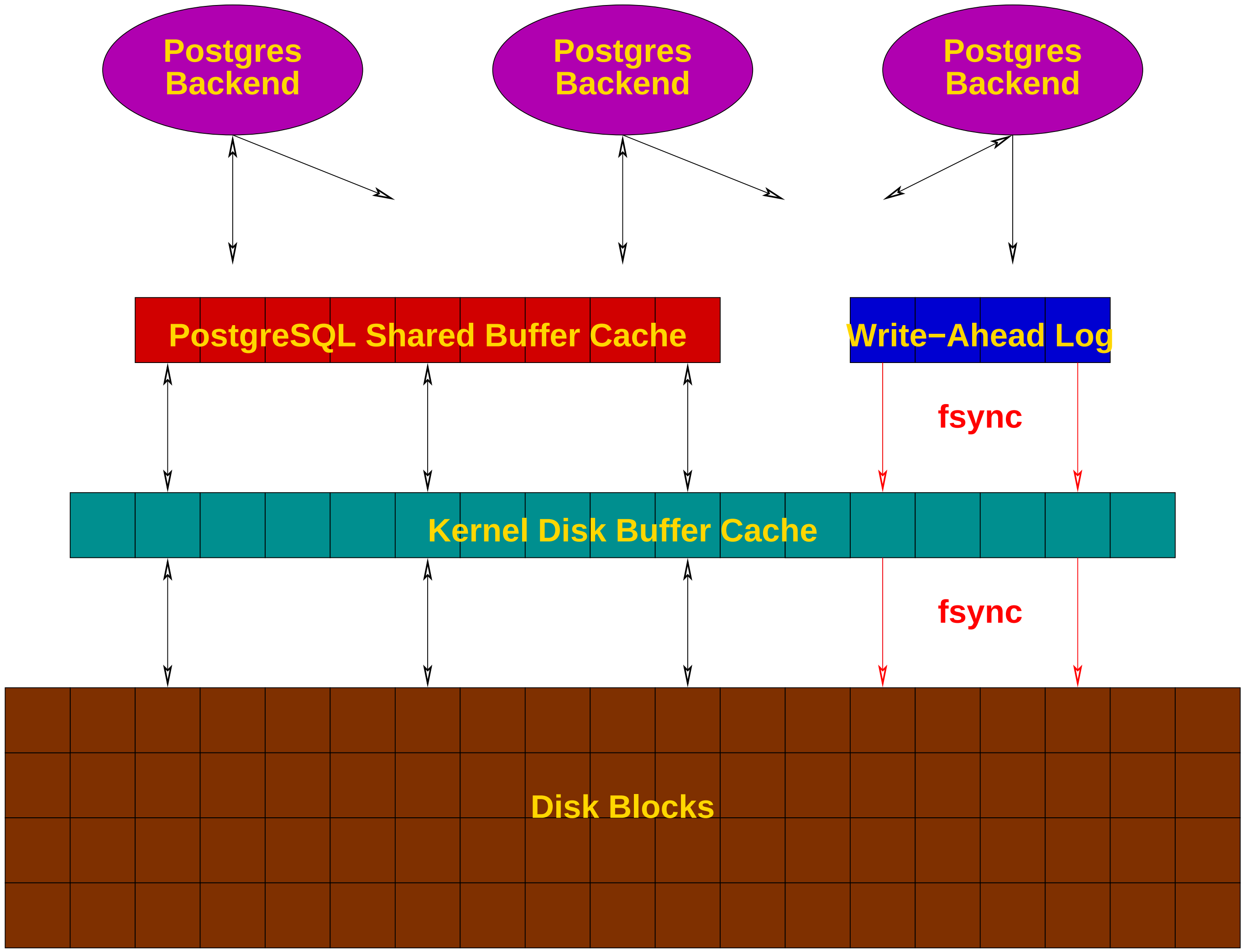
Abstract

POSTGRESQL 9.0 offers new facilities for maintaining a current standby server and for issuing read-only queries on the standby server. This tutorial covers these new facilities.

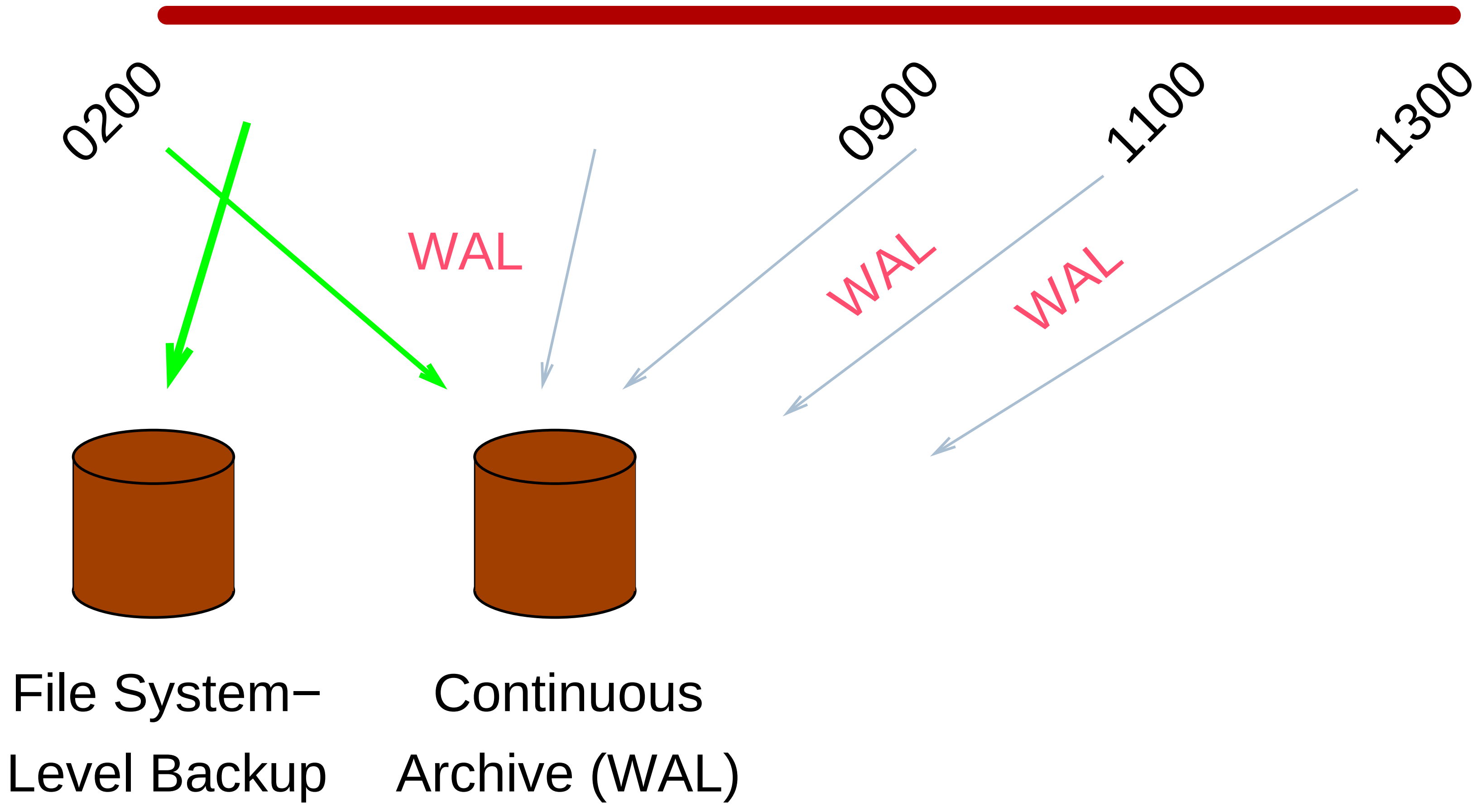
Introduction

- How does WAL combined with a disk image enable standby servers? (review)
- How do you configure continuous archiving?
- How do you configure a streaming, read-only server?
- Multi-Server complexities
- Primary/Standby synchronization complexities

Write-Ahead Logging (xlog)



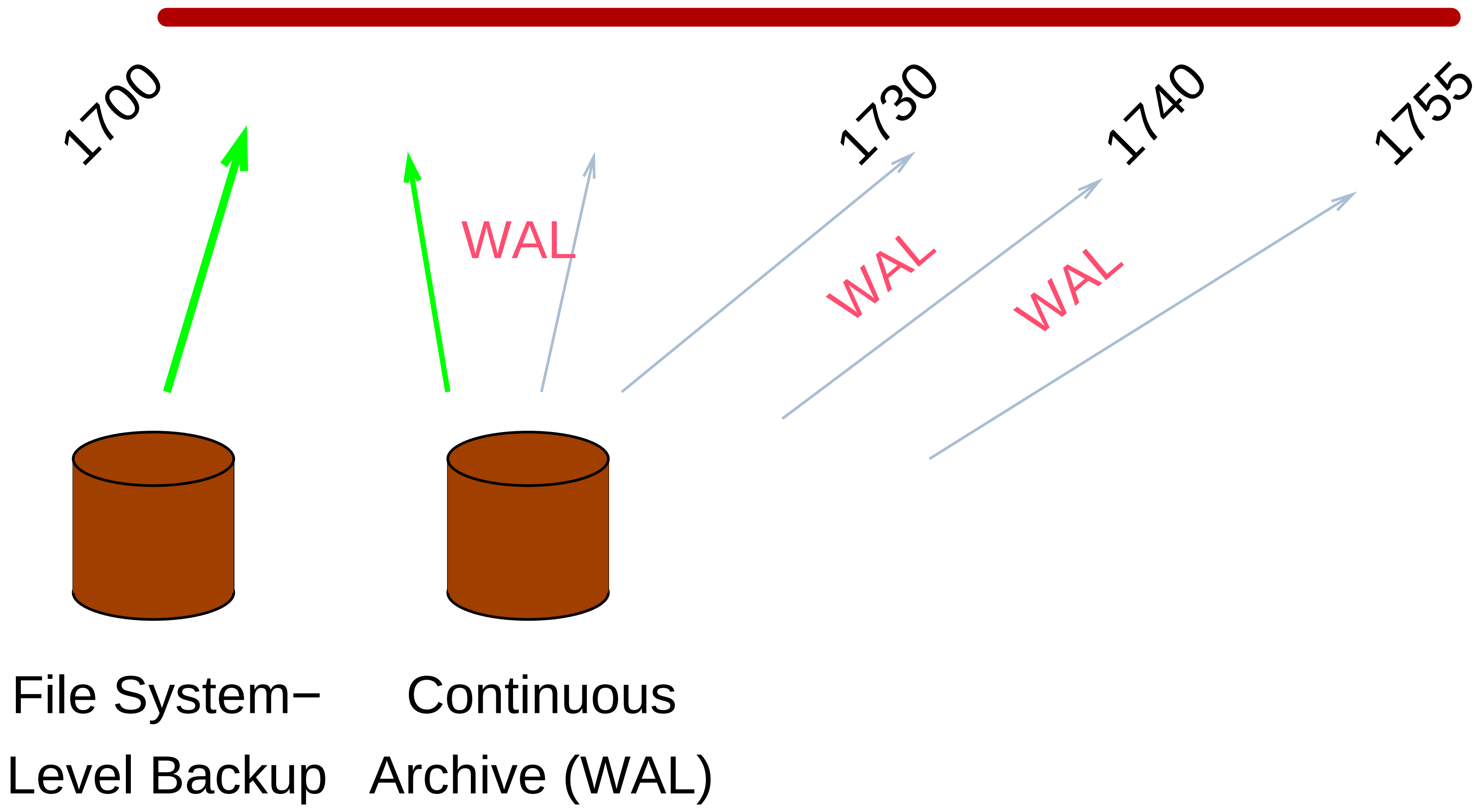
Pre-9.0 Continuous Archiving / Point-In-Time Recovery (PITR)



PITR Backup Procedures

1. `archive_command = 'cp -i %p /mnt/server/pgsql/%f < /dev/null'`
2. `SELECT pg_start_backup('label');`
3. Perform file system-level backup (can be inconsistent)
4. `SELECT pg_stop_backup();`

PITR Recovery



PITR Recovery Procedures

1. Stop postmaster
2. Restore file system-level backup
3. Make adjustments as outlined in the documentation
4. Create recovery.conf
5. `restore_command = 'cp /mnt/server/pgsql/%f %p'`
6. Start the postmaster

Disadvantages

- Only complete 16MB files can be shipped
- *archive_timeout* can be used to force more frequent shipping (this increases archive storage requirements)
- No queries on the standby

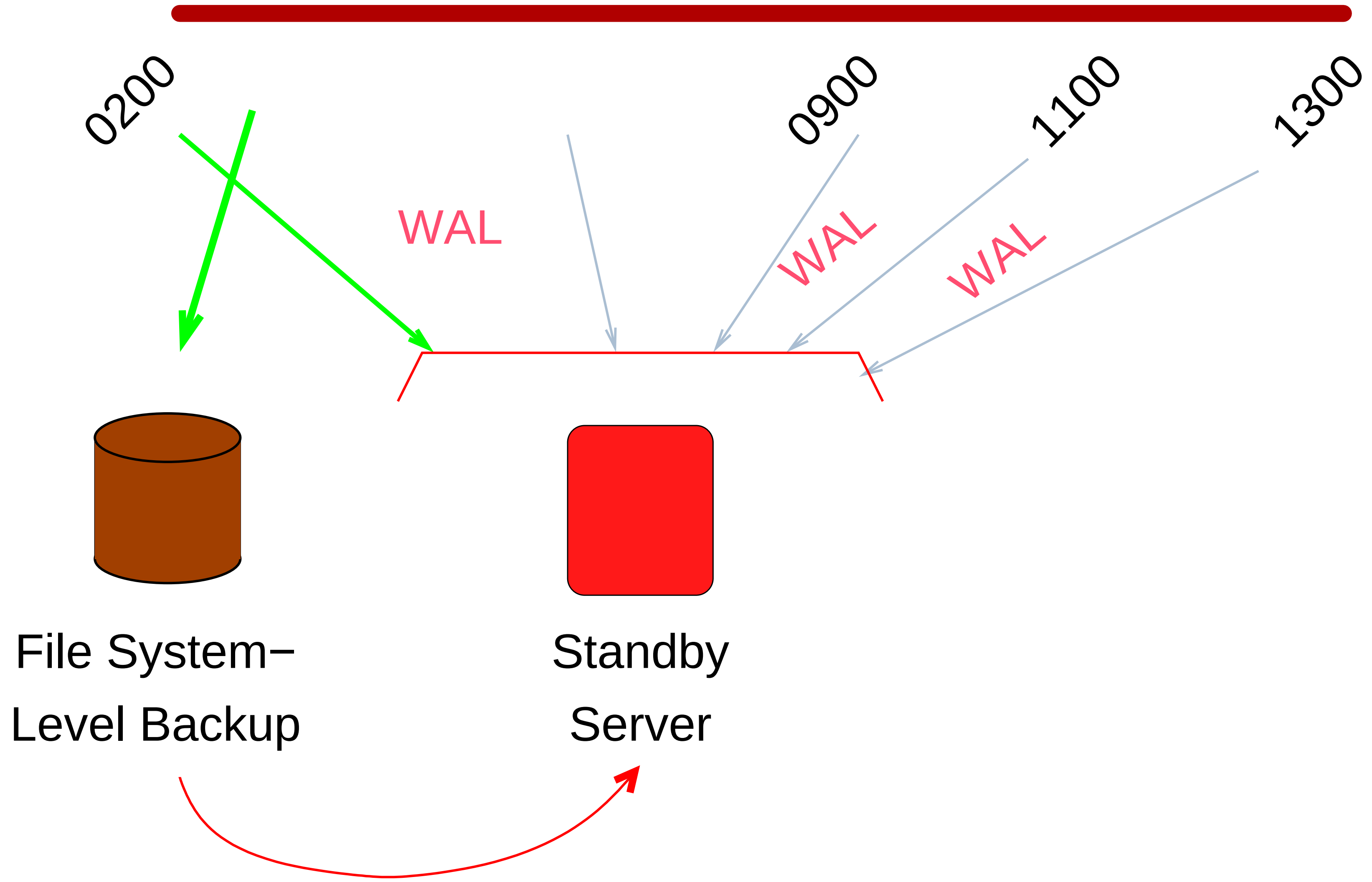
9.0 Streaming Replication / Hot Standby

- Changes are streamed to the standby, greatly reducing log shipping delays
- Standby can accept read-only queries

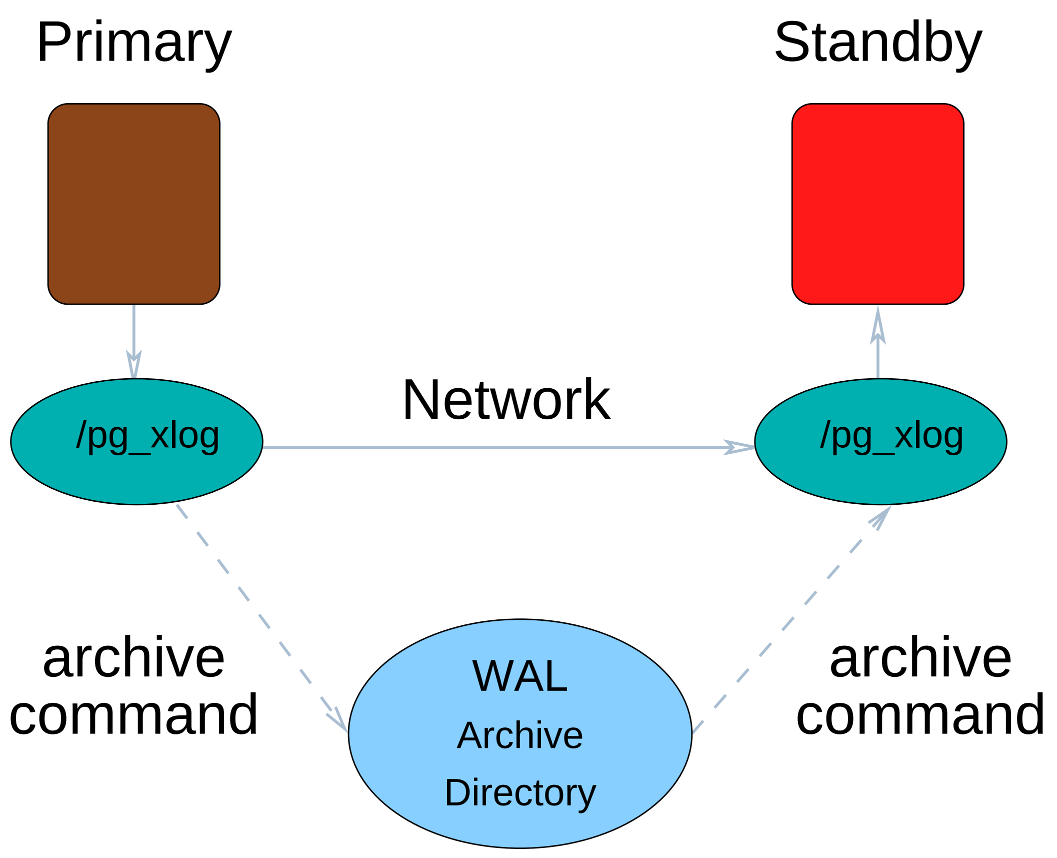
Streaming Replication Differs from PITR

- File system backup is restored immediately on the standby server
- WAL files are streamed to the slave
- WAL files can also be archived if point-in-time recovery (PITR) is desired

How Does Streaming Replication Work?



Live Streaming Replication



Enable Streaming to the Standby

Enable the proper WAL contents:

```
wal_level = hot_standby
```

Retain WAL files needed by the standby:

```
wal_keep_segments = 50
```

Enable the ability to stream WAL to the standby:

```
max_wal_senders = 1
```

Enable Standby Connection Permissions

Add permission for replication to *pg_hba.conf:*

```
host    replication    all    127.0.0.1/32    trust
```

Start the primary server:

```
pg_ctl -l /u/pg/data/server.log start
```

Perform a WAL-Supported File System Backup

Start *psql* and issue:

```
SELECT pg_start_backup('testing');
```

Copy the database */u/pg/data* to a new directory, */u/pg/data2:*

```
cp -p -R /u/pg/data /u/pg/data2
```

Dash-p preserves ownership. The copy is inconsistent, but that is okay (WAL replay will correct that).

Signal the backup is complete from *psql:*

```
SELECT pg_stop_backup();
```

Configure the Standby

Remove /data2/postmaster.pid so the standby server does not see the primary server's pid as its own:

```
rm /u/pg/data2/postmaster.pid
```

(This is only necessary because we are testing with the primary and slave on the same computer.)

Edit *postgresql.conf* on the standby and change the port to 5433

```
port = 5433
```

Enable hot standby in *postgresql.conf:*

```
hot_standby = on
```

Configure the Standby For Streaming Replication

Create *recovery.conf:*

```
cp /u/pg/share/recovery.conf.sample /u/pg/data2/recovery.conf
```

Enable streaming in *recovery.conf:*

```
standby_mode = 'on'
primary_conninfo = 'host=localhost port=5432'
```

Start the standby server:

```
PGDATA=/u/pg/data2 pg_ctl -l /u/pg/data2/server.log start
```

Test Streaming Replication and Hot Standby

```
$ psql -p 5432 -c 'CREATE TABLE streamtest(x int)' postgres
$ psql -p 5433 -c '\d' postgres
           List of relations
 Schema |    Name    | Type  |  Owner
--------+------------+-------+----------
 public | streamtest | table | postgres
(1 row)

$ psql -p 5432 -c 'INSERT INTO streamtest VALUES (1)'  postgres
INSERT 0 1
$ psql -p 5433 -c 'INSERT INTO streamtest VALUES (1)'  postgres
ERROR:  cannot execute INSERT in a read-only transaction
```

Additional Complexities

- Multi-server permissions
- Stream from /pg_xlog and the continuous archive directory if *archive_mode* is enabled on the primary

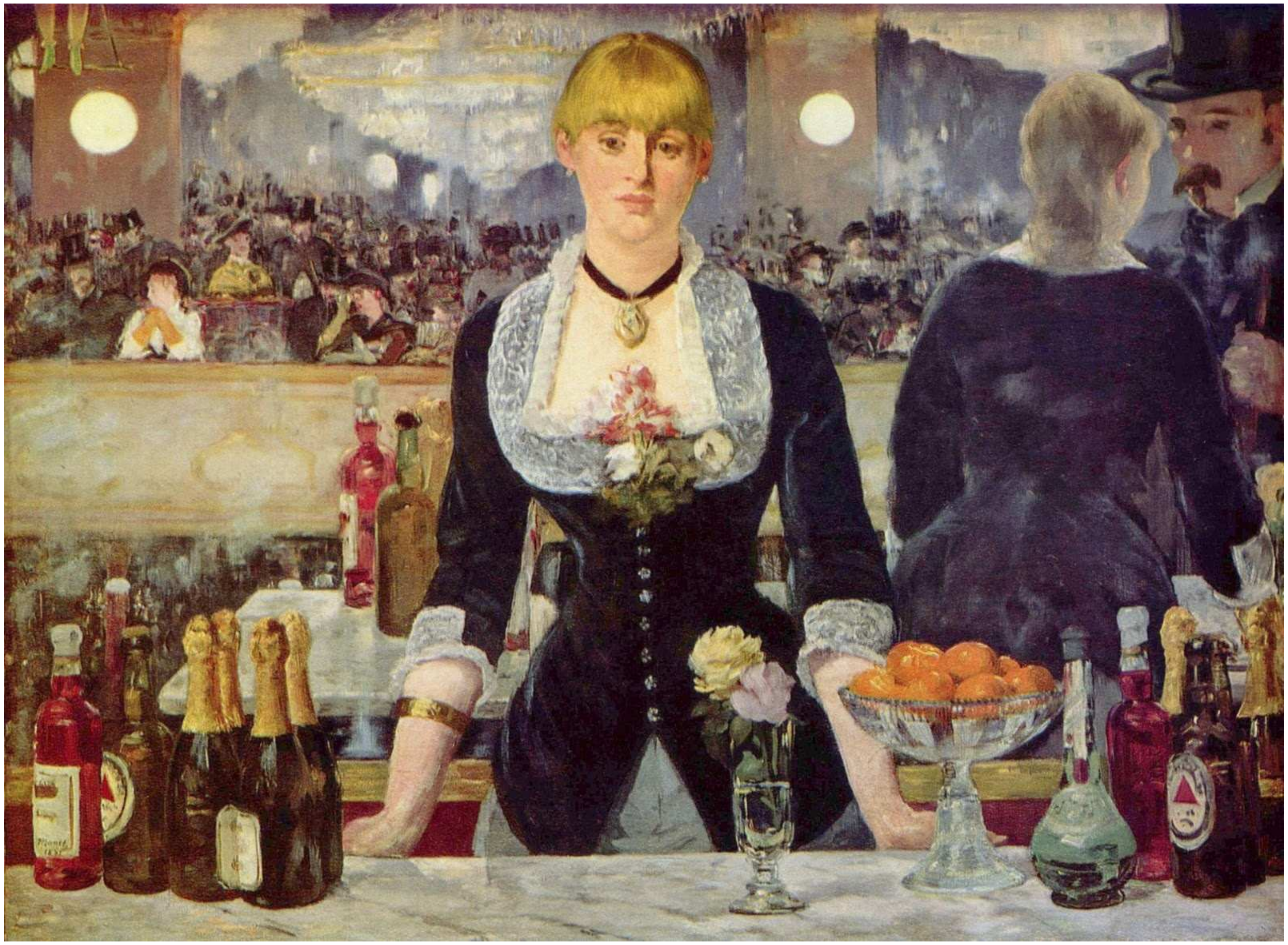
Primary/Standby Synchronization Issues

The primary server can take actions that cause long-running queries on the standby to be cancelled. Specifically, the cleanup of unnecessary rows that are still of interest to long-running queries on the standby can cause long-running queries to be cancelled on the standby. Standby query cancellation can be minimized in two ways:

1. Delay cleanup of old records on the primary with *vacuum_defer_cleanup_age* in *postgresql.conf.*

2. Delay application of WAL logs on the standby with *max_standby_delay* in p*ostgresql.conf.* The default is 30 seconds; -1 causes application to delay indefinitely to prevent query cancellation. This also delays changes from appearing on the standby and can lengthen the time required for failover to the slave.

Conclusion



Manet, Bar at Folies Bergère