2ndQuadrant

Advanced PostgreSQL Professional Services

Replication Replication Replication

Simon Riggs
2nd Quadrant
simon@2ndQuadrant.com

2ndQuadrant

Advanced PostgreSQL Professional Services

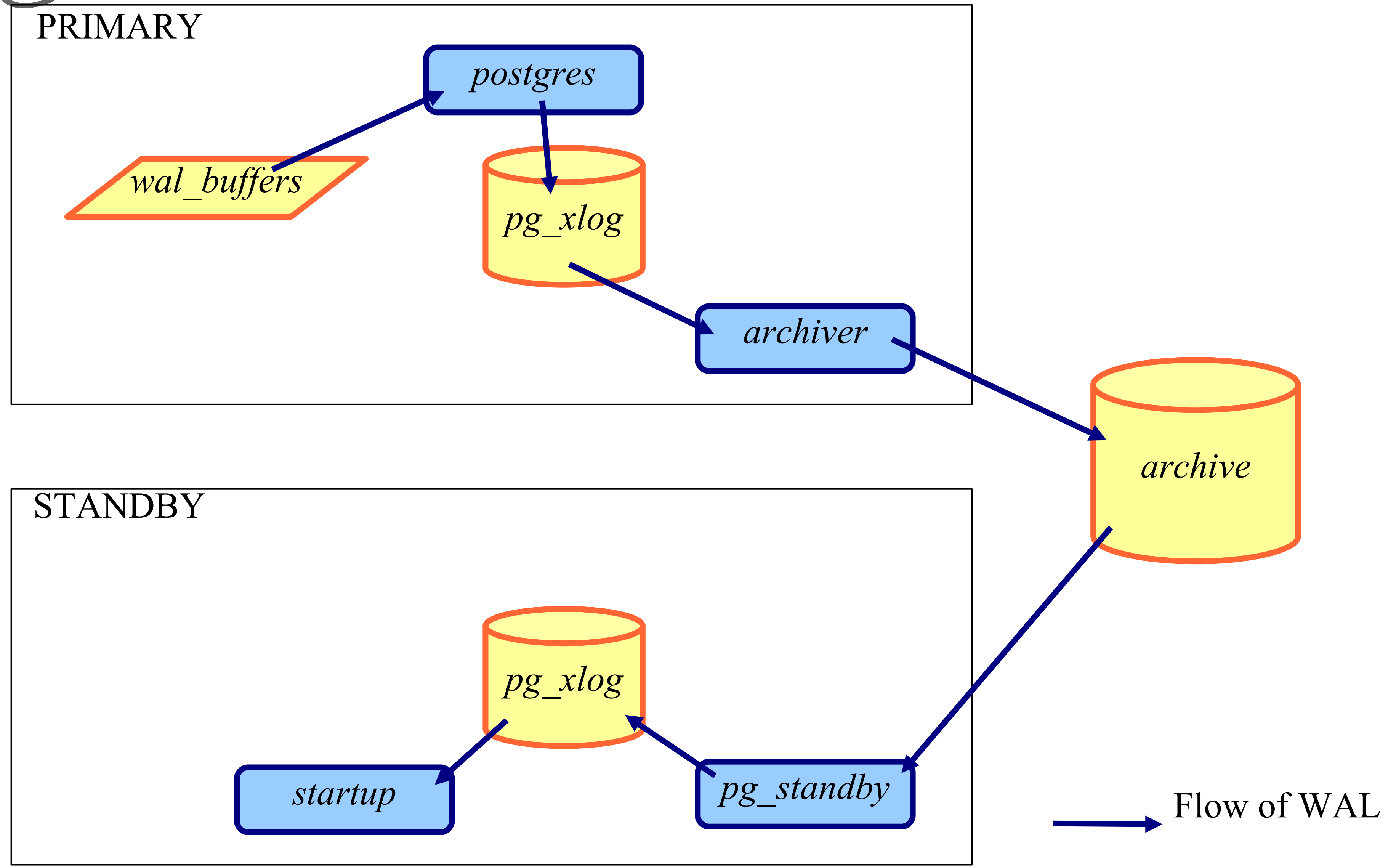
~~Replication~~
Replication
Replication

Simon Riggs
2nd Quadrant
simon@2ndQuadrant.
com

Topics

- Streaming Replication
- Hot Standby
- Futures
- Scorecard
- Conclusion

File-based Log Shipping (8.2/8.3)



PRIMARY

postgres

wal_buffers

pg_xlog

archiver

archive

STANDBY

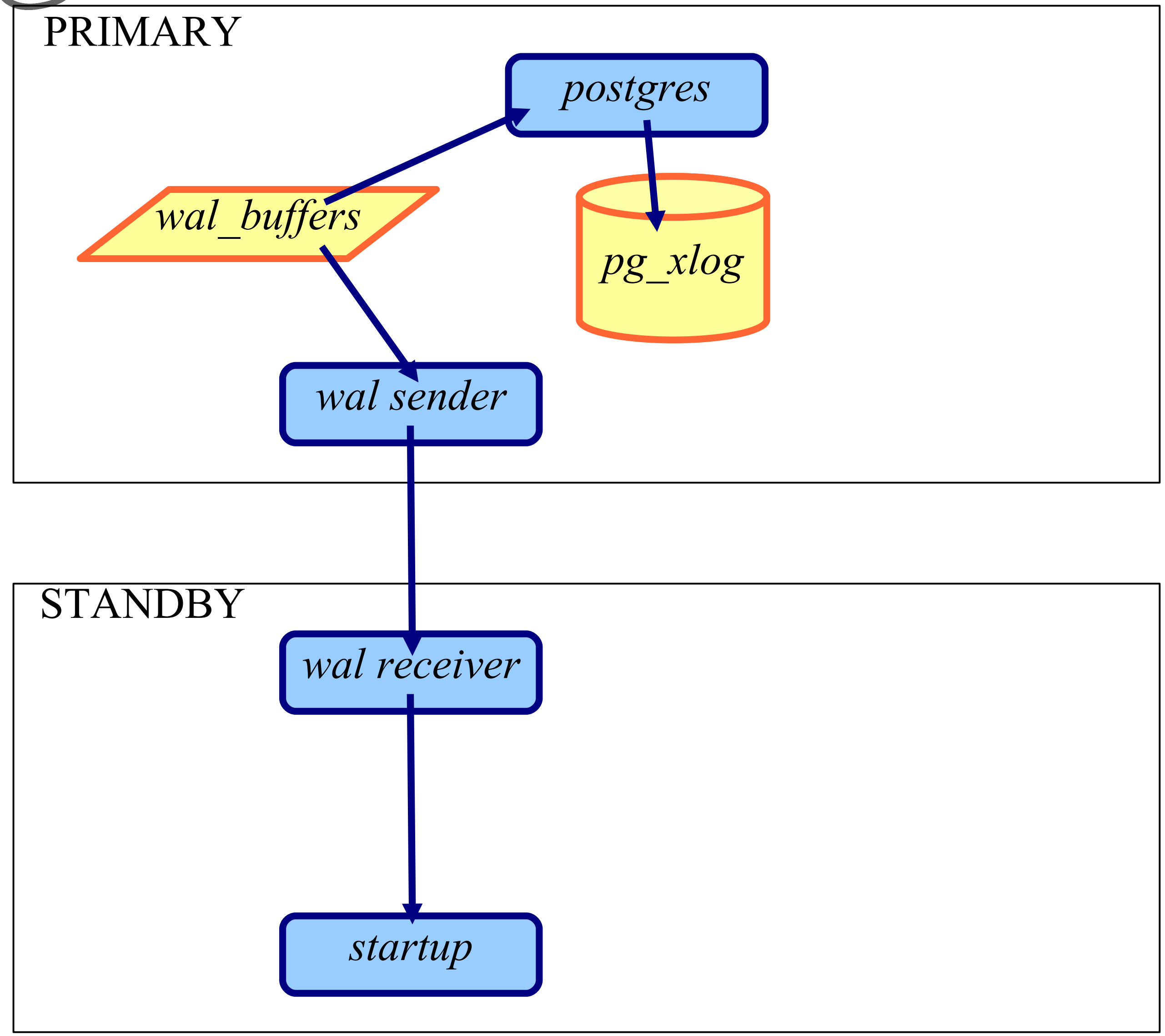
pg_xlog

startup

pg_standby

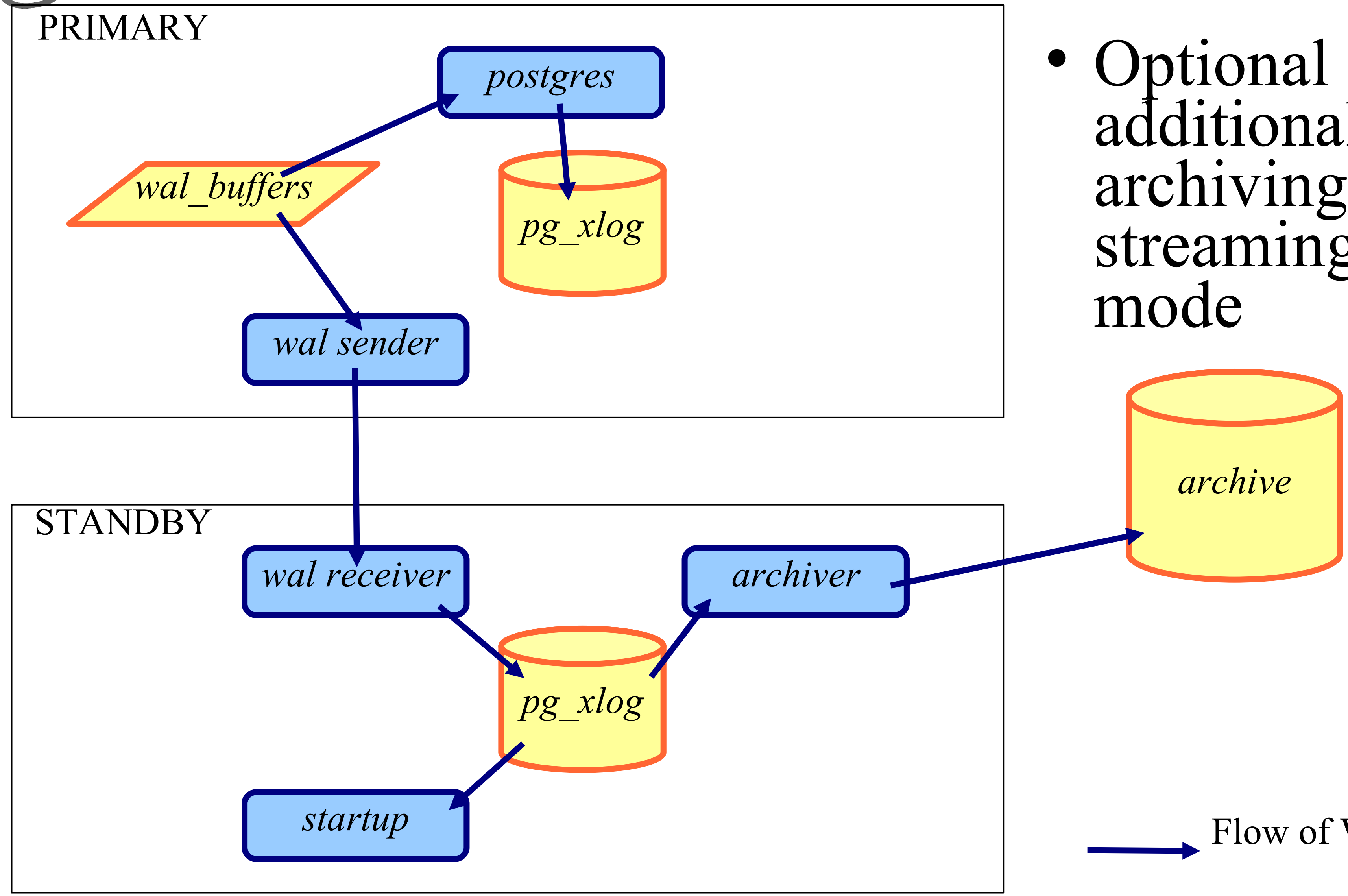
Flow of WAL

Sync Replication [Stream mode]



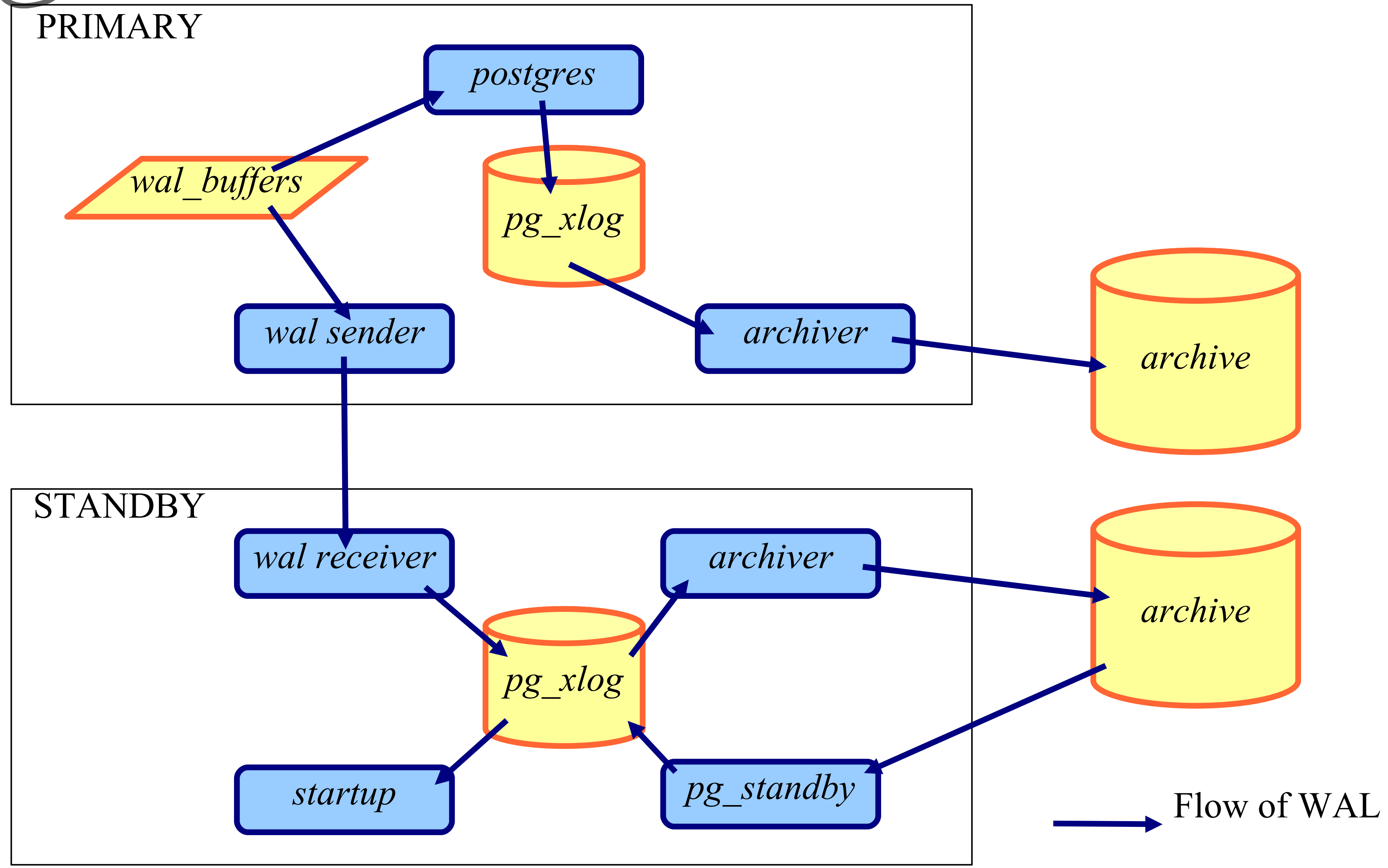
- “Intuitive” design

Sync Replication [Archiving]

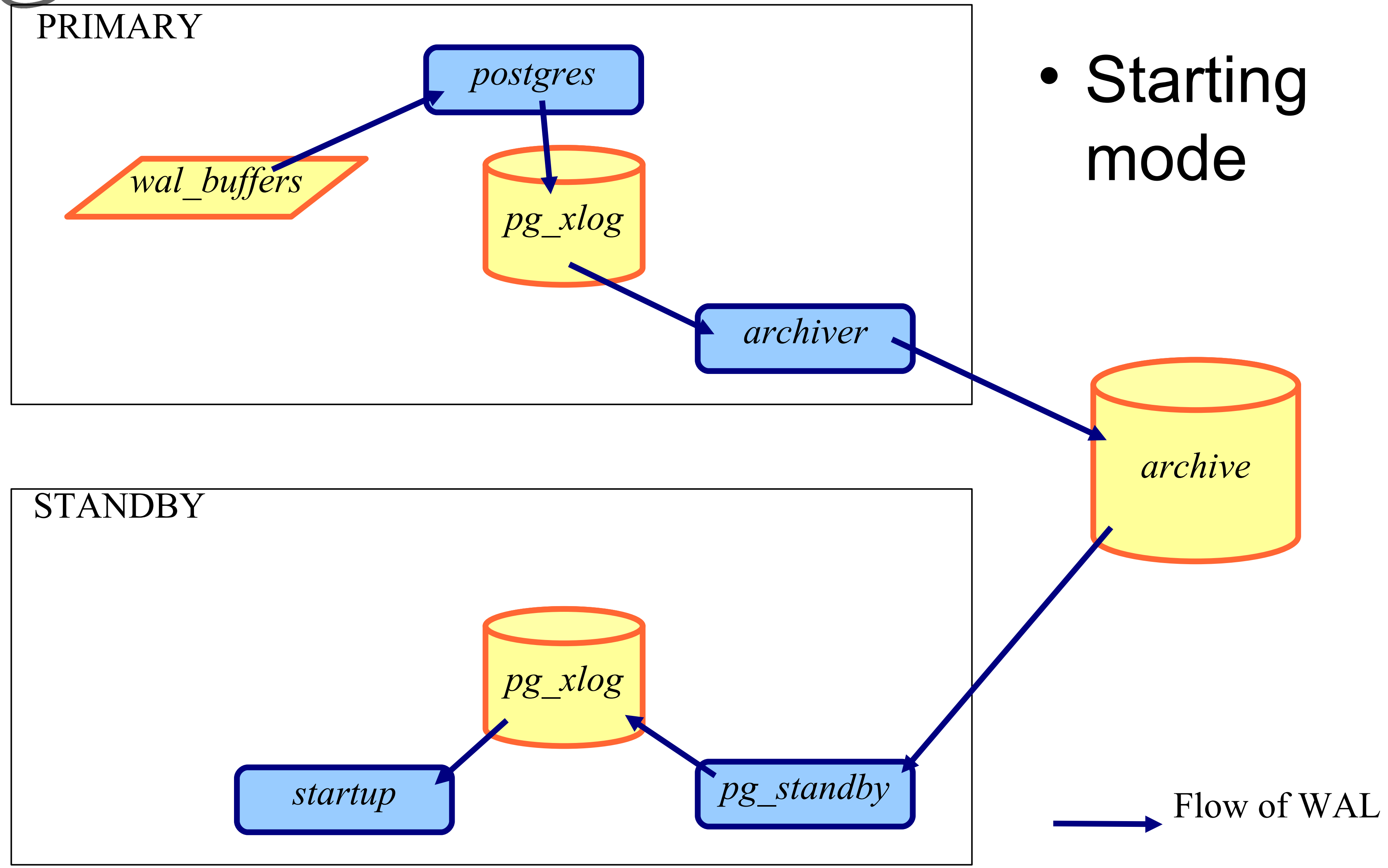


- Optional additional archiving in streaming mode

Sync Replication [As at 1/11]

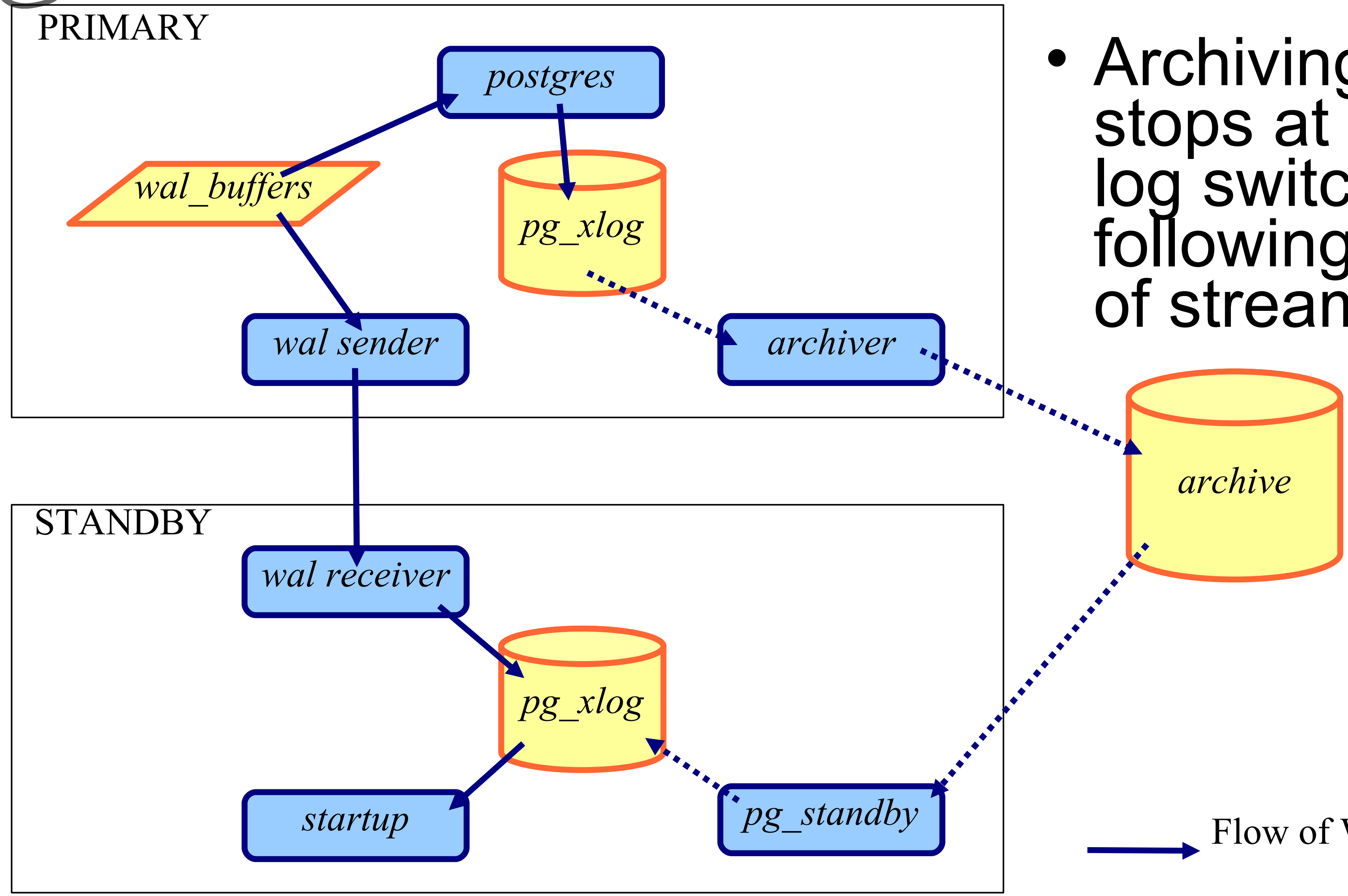


Sync Replication [File mode]



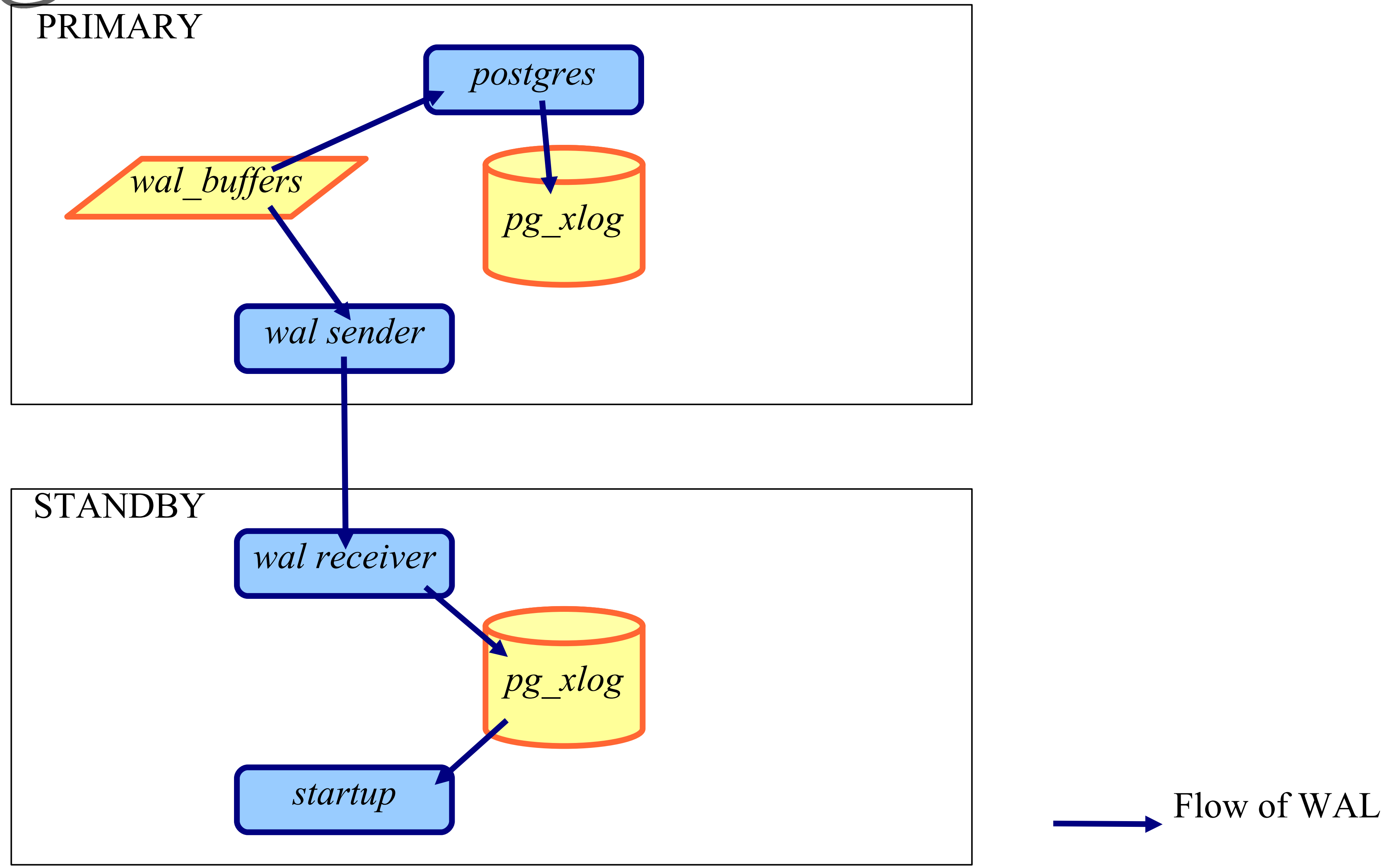
- Starting mode

Sync Replication [Switching]

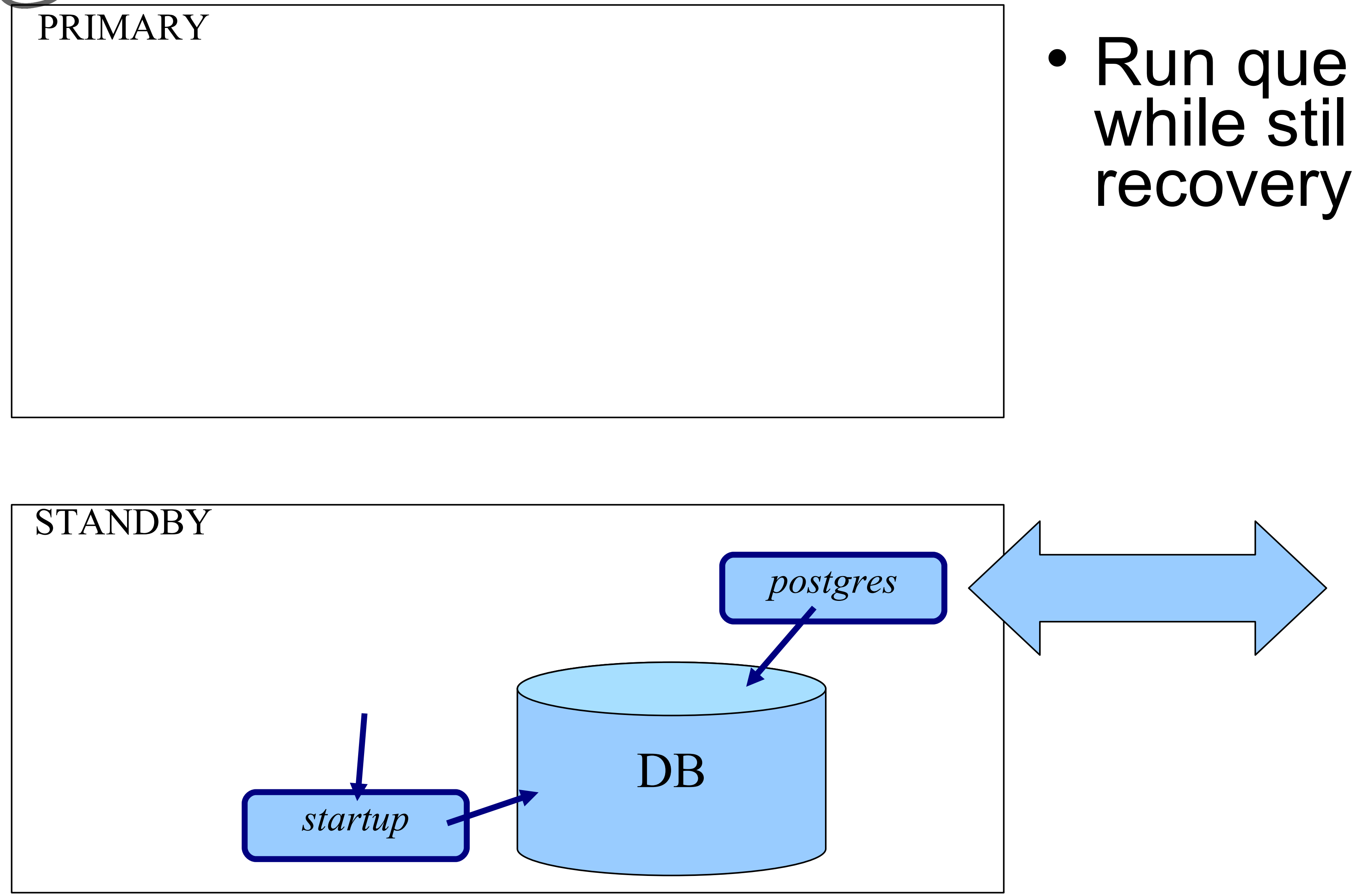


- Archiving stops at next log switch following start of streaming

Sync Replication [Stream mode]



Hot Standby



- Run queries while still in recovery

Problem #1: Transactions

- How do we run transactions when we can't allocate new TransactionIds? (Xids)
- Florian Pflug solved the transaction problem in 8.3: Read-only transactions never allocate Xids
- Florian's early analysis of these problems made an eventual solution feasible

Problem #2: Conflicts

- Recovery may need to do things like ALTER TABLE. Standby queries could be reading the table we wish to alter.
- Recovery needs to clean “old” data out, when primary system runs HOT or VACUUM. Standby queries might **need** to see data even **after** it has been removed
- Recovery may need to drop tablespaces or databases that we are currently using.

Conflict Resolution

- Wait-then-cancel
  - Startup process waits up to max_standby_delay, then issues a cancel, backend will then
    - immediate ERROR if AccessExclusiveLock request
    - defer ERROR until we see a block with a recent LSN
- Pause recovery
- Linkback session
  - Connect to primary with dblink and hold open a serializable transaction, so we never receive any cleanup records on standby

Deferred Buffer Conflicts

- Each proc maintains a small cache (8) of relations that it **may** have conflicts with
- If query reads buffer for that relation we check LSN of buffer to see if it is later than conflict LSN
- When cache overflows, we cancel query for **any** relation if buffer is lately modified
- Idle sessions and idle in transaction sessions **never** cause **buffer** conflicts
- Reality is that very active OLTP sites will have many conflicts and so will require planning

Problem #3: Snapshots

- PostgreSQL MVCC requires that we have a snapshot when we read user tables
- We cannot make sense of tuple xids otherwise
- Options
  - Get a snapshot from primary
  - Build a snapshot from WAL information

Incomplete Information

• BEGIN;	• (nothing)
• INSERT ...	• INSERT ...
• LOCK TABLE ...	• (nothing)
• SAVEPOINT s1;	• (nothing)
• INSERT ...	• INSERT ...
• COMMIT;	• COMMIT ...

Locking

- On Standby only locks allowed are AccessShareLocks. AccessShareLocks only conflict with AccessExclusiveLocks. Users cannot cause deadlocks...
- So the **only** locks we care to track are AccessExclusiveLocks
- Advisory Locks are allowed on standby, but advisory locks on primary are **not** propagated
- Locks are held by Standby process by proxy

Catalog Info Cacheing

- Current xlog code does not use relcache
- Queries **need** relcache
- New relcache usage mode “send_only”:
  can publish invalidations but never reads them
- Need to invalidate flat files

Unobserved Xids

- Snapshots **must** contain record of all running transactions, otherwise we can violate MVCC
- Xids are assigned in sequence, but don't arrive in order because of block locking
- Xids can be assigned recursively in some cases, so no theoretical limit on unobserved xids
- Limit recursive assignment with new WAL record type. Rarely called, though limits number of unobserved xids to 2* max_connections

Subtransaction Marking

- We must store unobserved xids in snapshot
- No room because of subxid cache overflow
- Change logic of XidInMVCCSnapshot so we look in subxid cache **and** pg_subtrans
- **Now** we can fit unobserved xids in snapshot
- ⇒Can optimise pg_subtrans inserts so that they **never happen at all** unless we have > 64 subtransactions on current transaction

Atomic SubXids

- Do we need to mark clog at subcommit?
- After much analysis: No
- Re-arrange clog changes so that they all happen at commit/abort
- ⇒No new WAL records required!
- ⇒Optimise clog updates for large numbers of subtransactions
- ⇒Avoid need to update clog at subcommit

New WAL records

- Recursive Xid Assignment – if ever
- Running Xact Set – 1 per checkpoint
- AccessExclusiveLocks – 1 per lock
- Relcache Invalidation – 1 per invalidation
- Vacuum Cleanup Info – 1 per VACUUM

WAL Record Enhancements

- Each WAL record has 4 extra bytes
  - No extra space required on 64-bit systems
- Changed WAL records
  - No changes to main Insert, Update, Delete paths
  - Commit
  - Btree Vacuum

Performance

- Primary
  - < 0.1% impact from additional WAL
  - Subtransactions substantially improved: +0-5% typical
  - ~Zero impact on scalability
  - No increase in WAL volume
- Standby
  - 2% CPU impact, but we're I/O bound anyway
  - Some additional I/O on btree index vacuums (can be tuned away)
  - Bgwriter active: +10-30%
  - Queries slightly slower than normal

Recovery Control

⏸ - pg_recovery_pause()
⏭ pg_recovery_pause_xid()
pg_recovery_pause_timestamp()
or recovery_starts_paused (recovery.conf)

⏩ - pg_recovery_continue()
▶ pg_recovery_advance(n)
■ pg_recovery_stop()

- pg_is_in_recovery()
pg_current_recovery_target()
pg_last_recovered_xid()
pg_last_recovered_xlog_location()
pg_last_recovered_xact_timestamp()

Conflicts & Usability

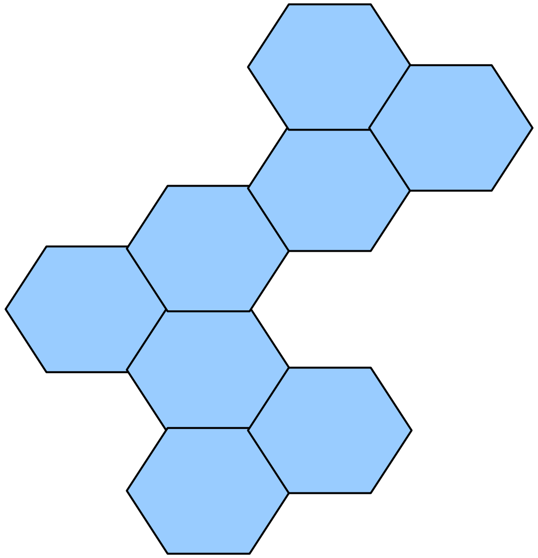
- Conflicts will cause some discussion
- No form of replication or clustering is free from performance or other side-effects
- First release
- Happy to tweak during beta, or fix in 8.5+

Project Overview

- Touches more than 80 files
- More than 10,000 lines
- ~6 man months, including significant testing from 5 staff in 2ndQuadrant, led by Gianni Ciolli
- 18 bugs in code of Nov 1
  - Around 50% found by code inspection
- > 30 changes and enhancements as a result of refactoring, review and discussion

Futures

- SQL/MED
  - Query routing by design, by workload
- Multiple streaming standby servers
  - Each with different missions
- Create a “Hive” of databases
  - Sharing data
  - Sharing queries
  - Loose coupling provides
    - Robust bulkheads in Hive to prevent loss of service
    - Minimise impact of changes between systems



FOSDEM Scorecard

sync repl	>15	✗
hot standby	15	✓
global restore points	15	✗
recovery parallelism	13	✓
xlogdump	11	✓
WAL compression	10	✓
include/exclude objects	7	✗
logical log based replication	5	✗
dropped table cache	2	✗

Conclusion

- Postgres is quickly becoming the *best* database
- Keep the dream alive
- Prioritise
- Act with urgency
- Do Big Things

2ndQuadrant

Advanced PostgreSQL Professional Services

PostgreSQ
L