Synchronous Log Shipping Replication

Takahiro Itagaki and Masao Fujii
NTT Open Source Software Center

PGCon 2008

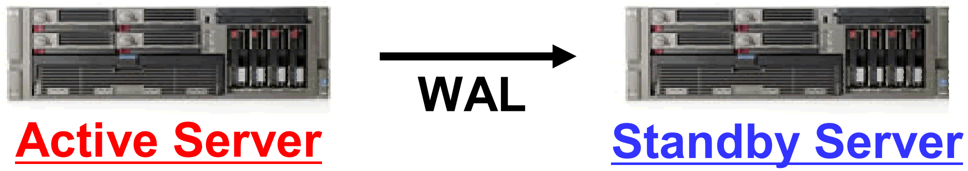
Agenda

- Introduction: What is this?
  - Background
  - Compare with other replication solutions
- Details: How it works
  - Struggles in development
- Demo
- Future work: Where are we going?
- Conclusion

What is this?

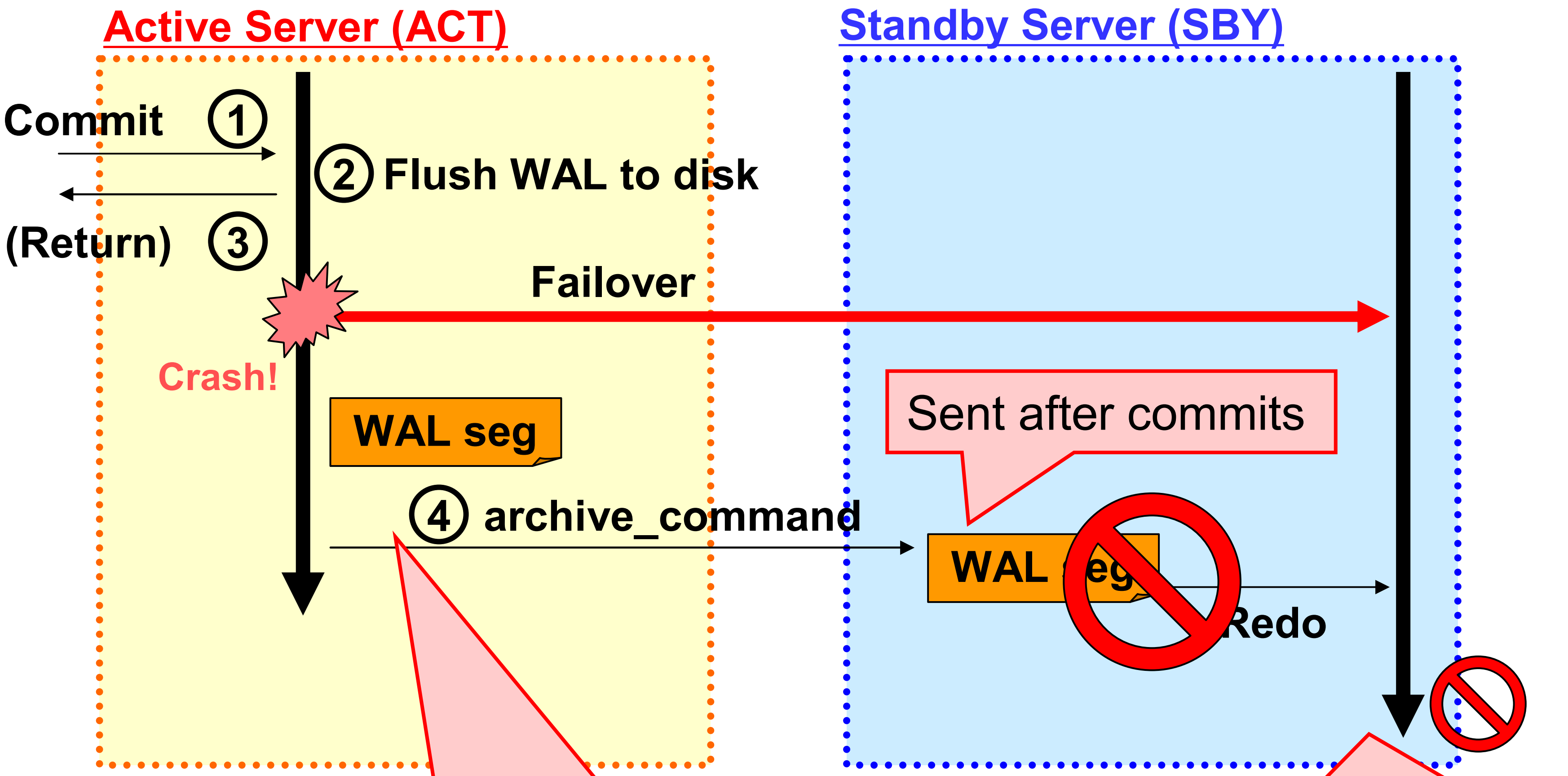
What is this?

- Successor of warm-standby servers
  - Replication system using WAL shipping.
    - using Point-in-Time Recovery mechanism
  - However, no data loss after failover because of synchronous log-shipping.



- Based on PostgreSQL 8.2 with a patch and including several scripts
  - Patch: Add two processes into postgres
  - Scripts: For management commands

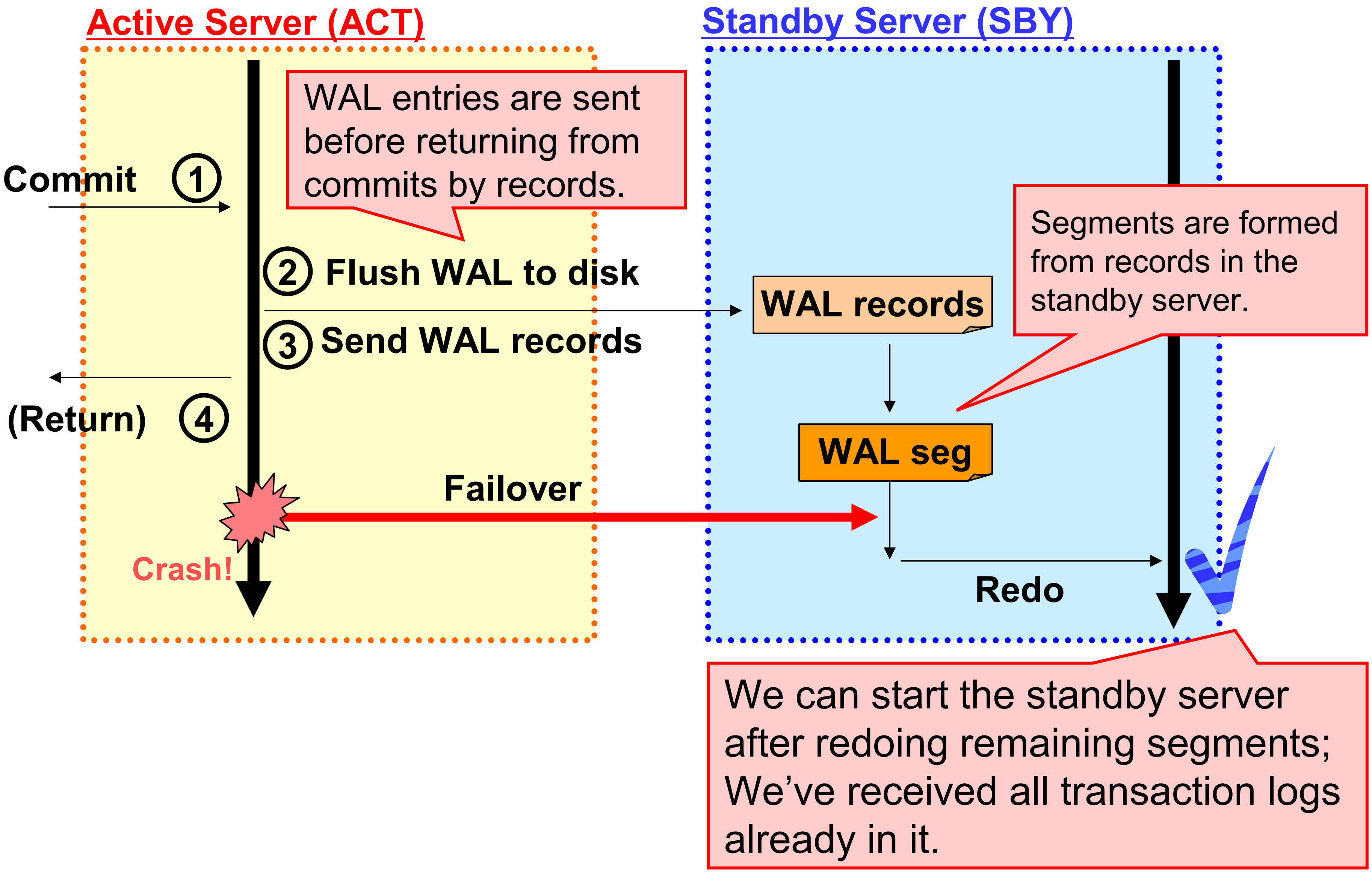
Warm-Standby Servers (v8.2~)



The last segment is not available in the standby server if the active crashes before archiving it.

We need to wait for remounting active's storage on the standby server, or we wait the active's reboot.

Synchronous Log Shipping Servers



WAL entries are sent before returning from commits by records.

Segments are formed from records in the standby server.

We can start the standby server after redoing remaining segments; We've received all transaction logs already in it.

Background: Why new solution?

- We have many migration projects from Oracles and compete with them with postgres.
  - So, we hope postgres to be **SUPERIOR TO ORACLE!**
- Our activity in PostgreSQL 8.3
  - Performance stability
    - Smoothed checkpoint
  - Usability; Ease to tune server parameters
    - Multiple autovacuum workers
    - JIT bgwriter – automatic tuning of bgwriter
- **Where are alternatives of RAC?**
  - Oracle Real Application Clusters

Background: Alternatives of RAC

- Oracle RAC is a multi-purpose solution
  - … but we don’t need all of the properties.
- In our use:
  - No downtime <- **Very Important**
  - No data loss <- **Very Important**
  - Automatic failover <- **Important**
  - Performance in updates <- **Important**
  - Inexpensive hardware <- **Important**
  - Performance scalability <- **Not important**
- Goal
  - Minimizing system downtime
  - Minimizing performance impact in updated-workloads

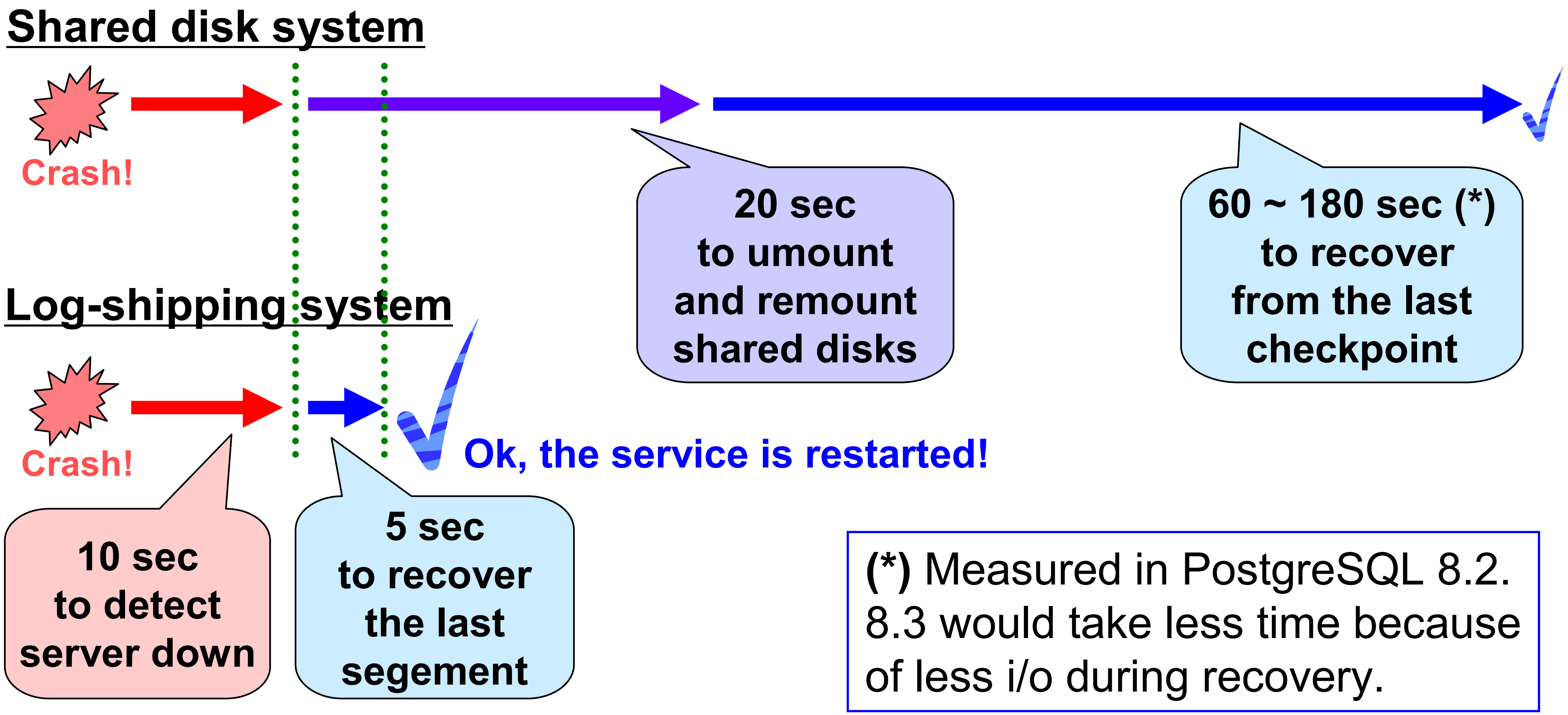
Compare with other replication solutions

	No data loss	No SQL restriction	Failover	Performance scalability	Update performance	How to copy?
Log Shipping	OK	OK	Auto, Fast	No	Good	Log
warm-standby	NG	OK	Manual	No	Async	
Slony-I	NG	OK	Manual	Good	Async	Trigger
pgpool-II	OK	NG	Auto, Hard to re-attach	Good	Medium	SQL
Shared Disks	OK	OK	Auto, Slow	No	Good	Disk

- Log Shipping is excellent except performance scalability.
- Also, Re-attaching a repaired server is simple.
  - Just same as normal hot-backup procedure
    - Copy active server's data into standby and just wait for WAL replay.
  - No service stop during re-attaching

Compare downtime with shared disks

- Cold standby with shared disks is an alternative solution
  - but it takes long time to failover in heavily-updated load.
  - Log-shipping saves time for mounting disks and recovery.



Shared disk system

Crash!
20 sec to umount and remount shared disks
60 ~ 180 sec (*) to recover from the last checkpoint

Log-shipping system

Crash!
Ok, the service is restarted!
10 sec to detect server down
5 sec to recover the last segement

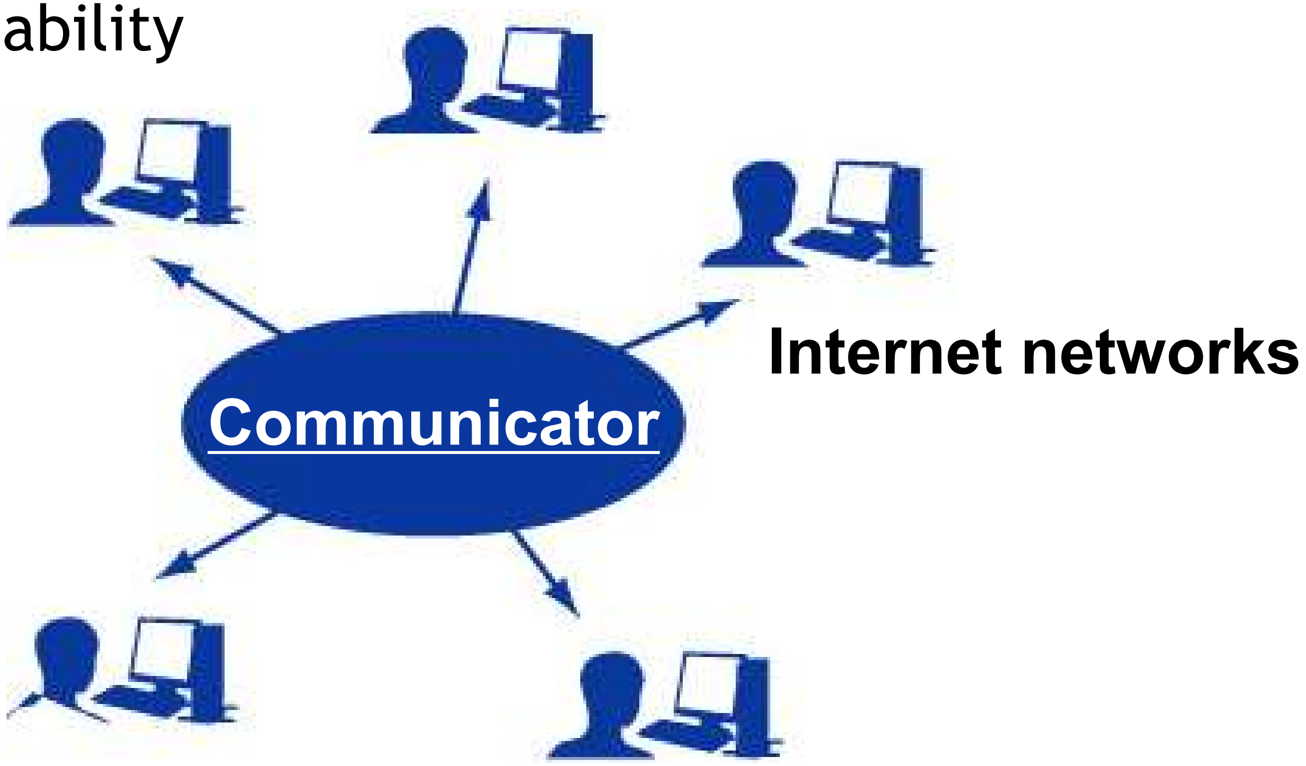
(*) Measured in PostgreSQL 8.2.
8.3 would take less time because of less i/o during recovery.

Advantages and Disadvantages

- Advantages
  - Synchronous
    - No data loss on failover
  - Log-based (Physically same structure)
    - No functional restrictions in SQL
    - Simple, Robust, and Easy to setup
  - Shared-nothing
    - No Single Point of Failure
    - No need for expensive shared disks
  - Automatic Fast Failover (within 15 seconds)
    - “Automatic” is essential not to wait human operations
  - Less impact against update performance (less than 7%)
- Disadvantages
  - No performance scalability (for now)
  - Physical replication. Cannot use for upgrading purposes.

Where is it used?

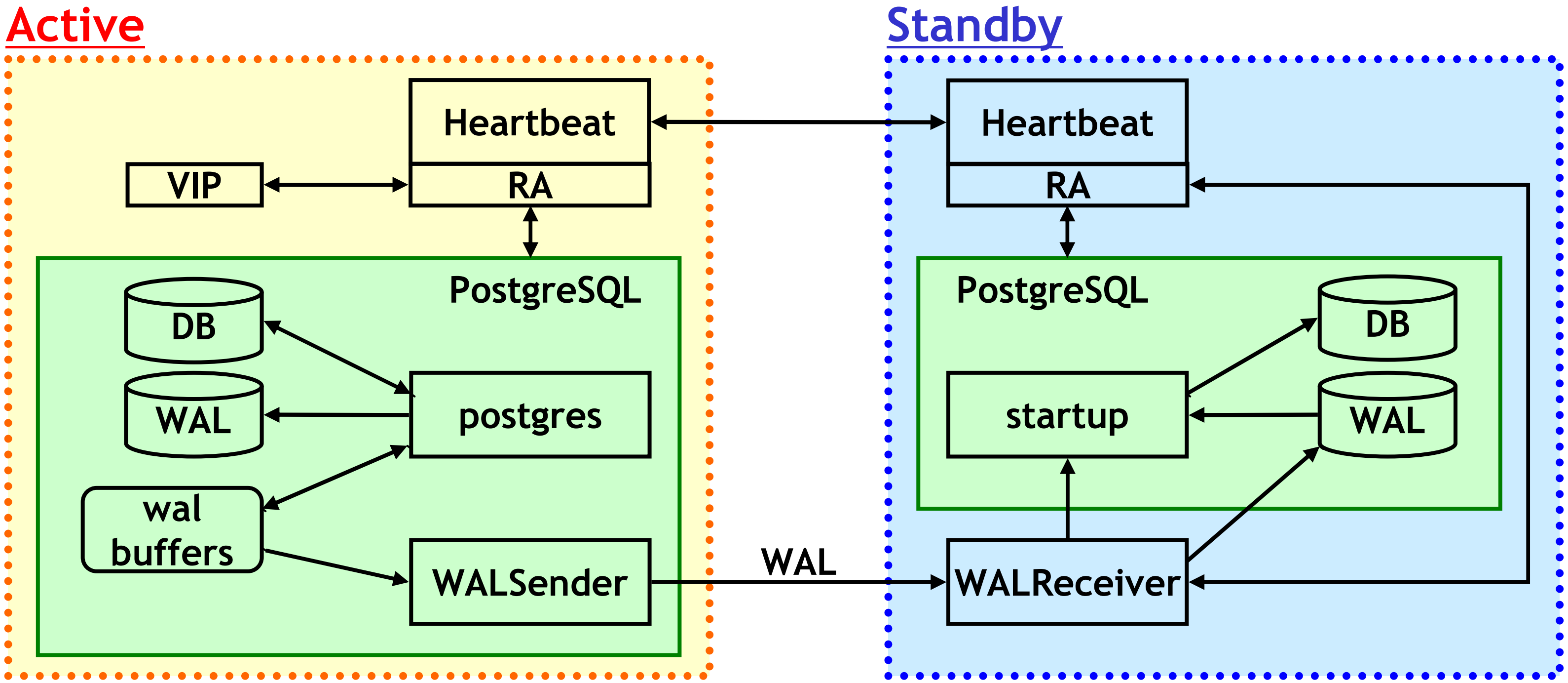
- Interactive teleconference management package
  - Commercial service in active
  - Manage conference booking and file transfer
  - Log-shipping is an optional module for users requiring high availability



How it works

System overview

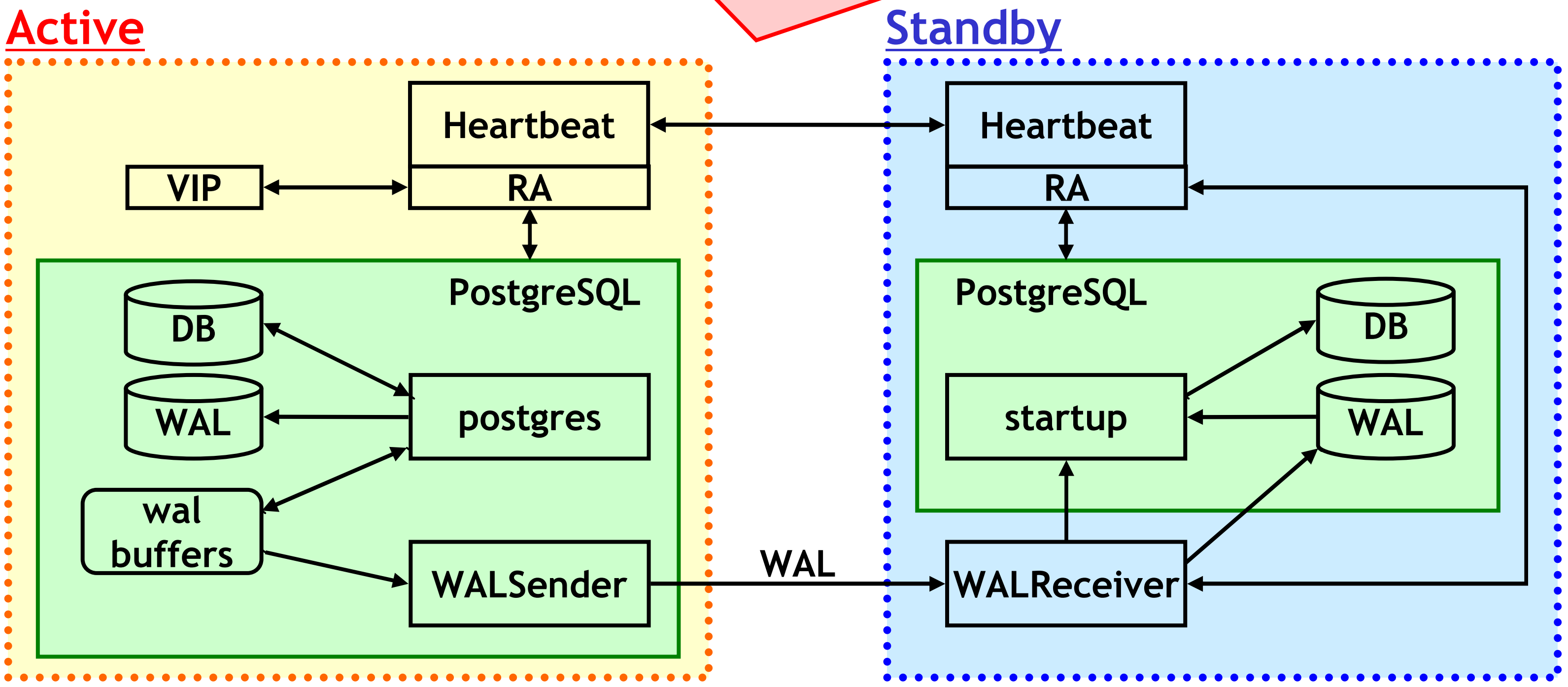
- Based on PostgreSQL 8.2, 8.3(under porting)
- WALSender
  - New child process of postmaster
  - Reads WAL from walbuffers and sends WAL to WALReceiver
- WALReceiver
  - New daemon to receive WAL
  - Writes WAL to disk and communicates with startup process
- Using Heartbeat 2.1
  - Open source high-availability software manages the resources via resource agent(RA)
  - Heartbeat provides a virtual IP(VIP)



System overview

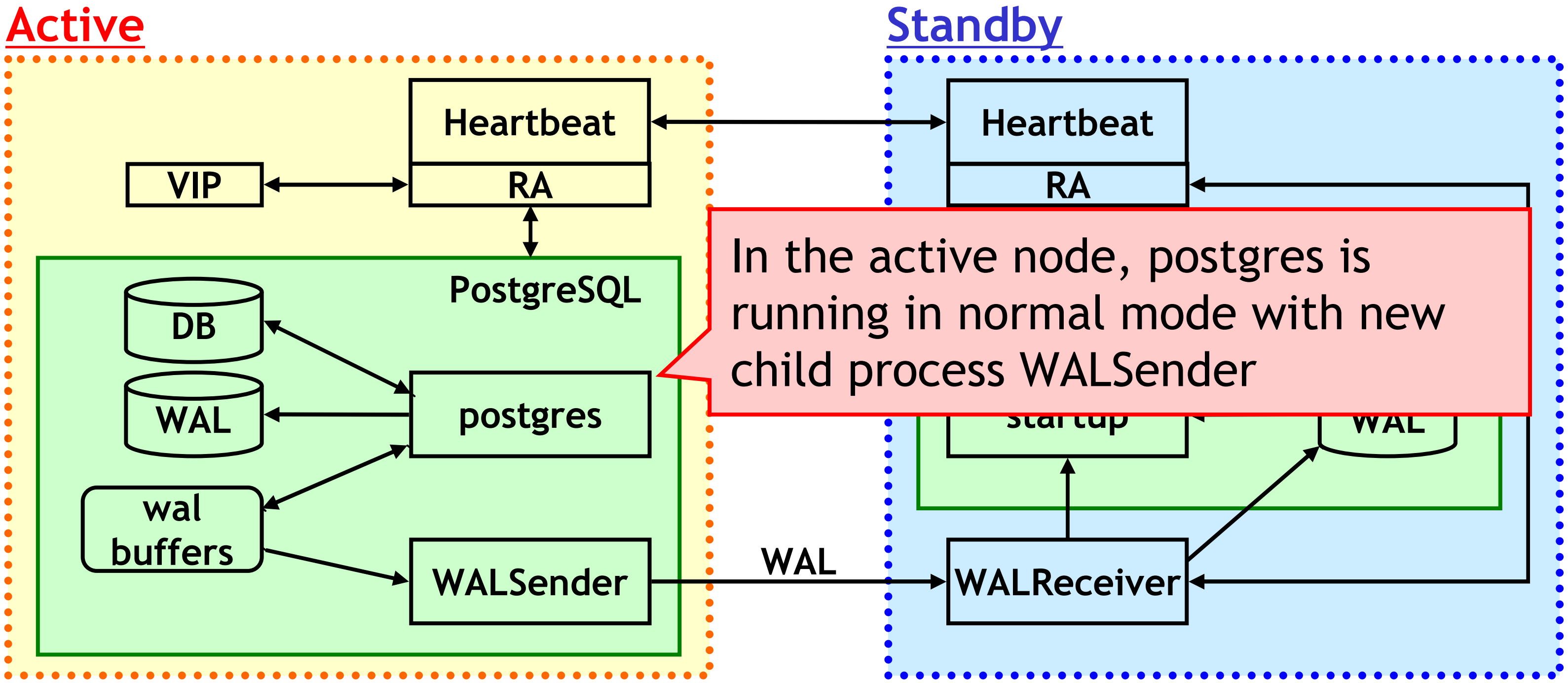
- Based on PostgreSQL 8.2, 8.3(under porting)
- WALSender
  - New child process of postmaster
  - Reads WAL from walbuffers and sends WAL to WALReceiver
- WALReceiver
  - New daemon to receive WAL
  - Writes WAL to disk and communicates with startup process
- Using Heartbeat 2.1
  - Open source high-availability nt(RA)
  - Heartbeat provides a virtual

In our replicator, there are two nodes, active and standby



System overview

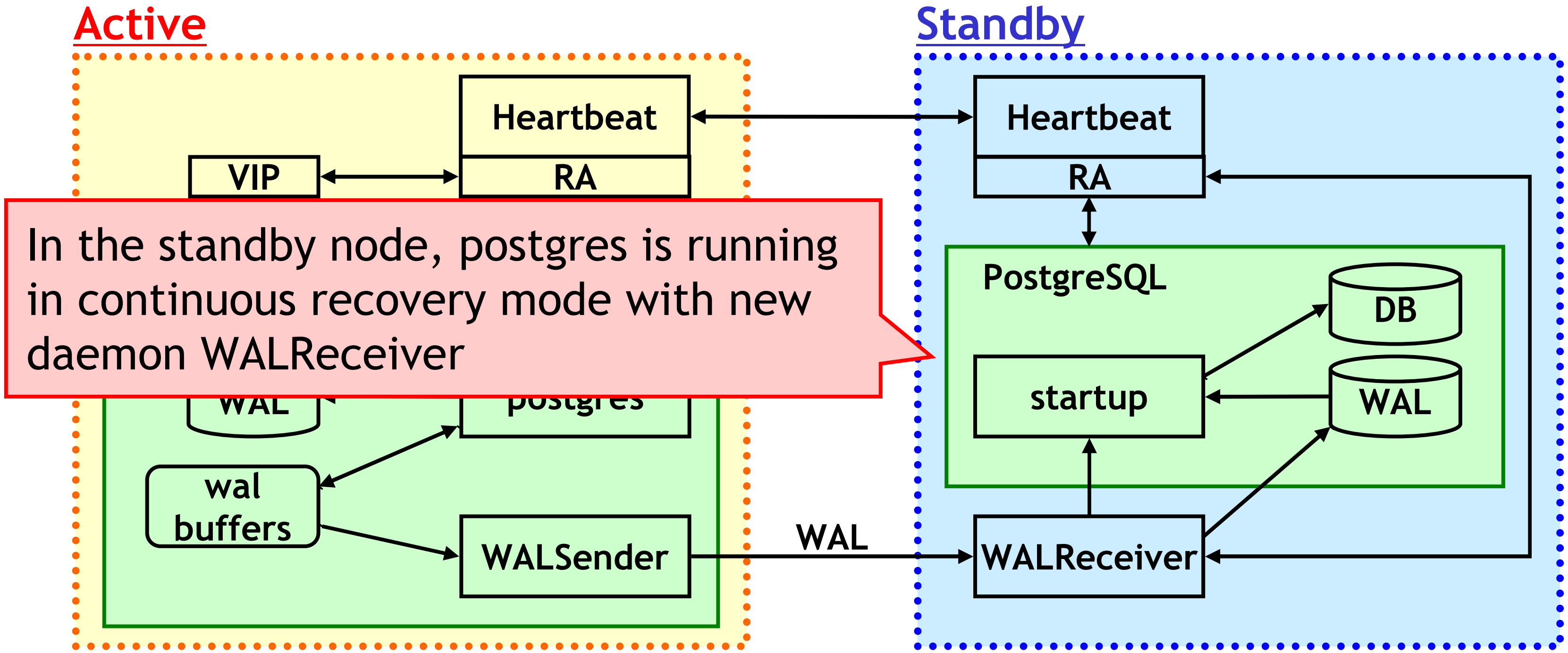
- Based on PostgreSQL 8.2, 8.3(under porting)
- WALSender
  - New child process of postmaster
  - Reads WAL from walbuffers and sends WAL to WALReceiver
- WALReceiver
  - New daemon to receive WAL
  - Writes WAL to disk and communicates with startup process
- Using Heartbeat 2.1
  - Open source high-availability software manages the resources via resource agent(RA)
  - Heartbeat provides a virtual IP(VIP)



In the active node, postgres is running in normal mode with new child process WALSender

System overview

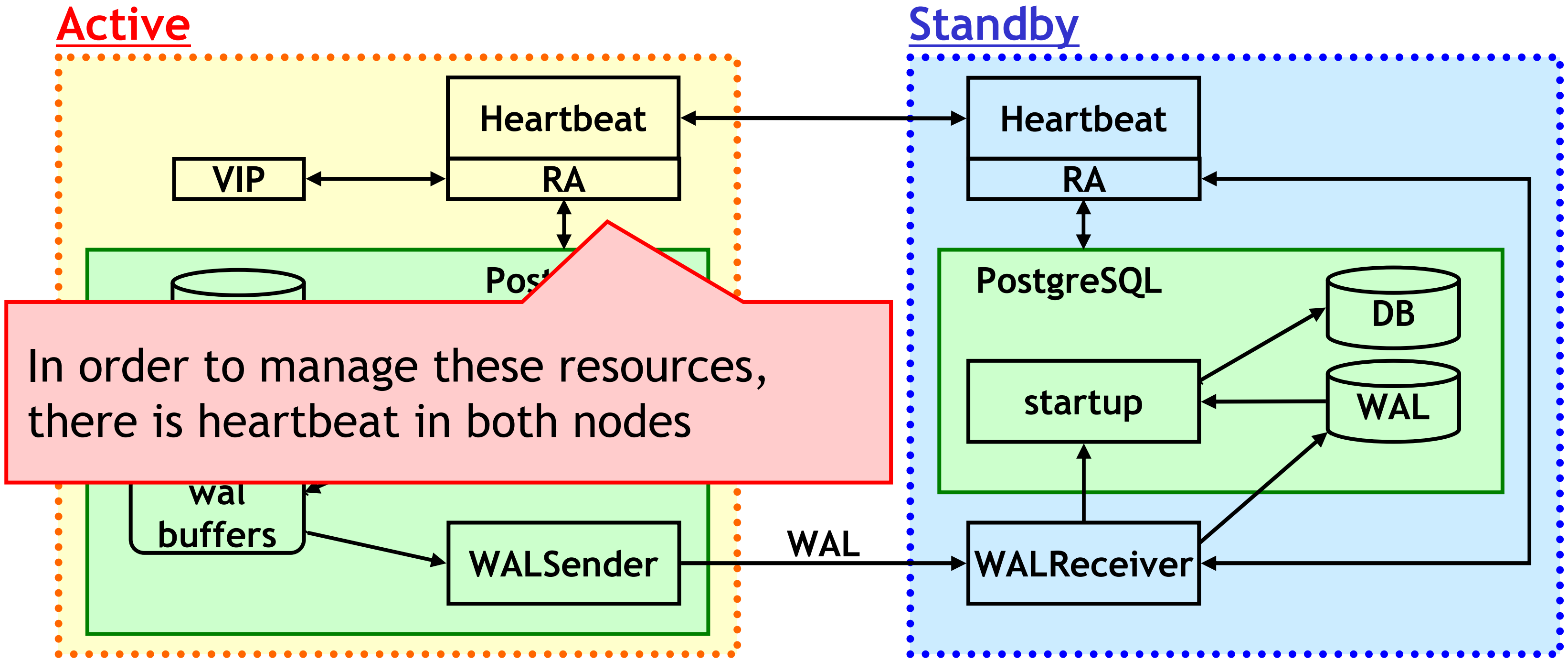
- Based on PostgreSQL 8.2, 8.3(under porting)
- WALSender
  - New child process of postmaster
  - Reads WAL from walbuffers and sends WAL to WALReceiver
- WALReceiver
  - New daemon to receive WAL
  - Writes WAL to disk and communicates with startup process
- Using Heartbeat 2.1
  - Open source high-availability software manages the resources via resource agent(RA)
  - Heartbeat provides a virtual IP(VIP)



In the standby node, postgres is running in continuous recovery mode with new daemon WALReceiver

System overview

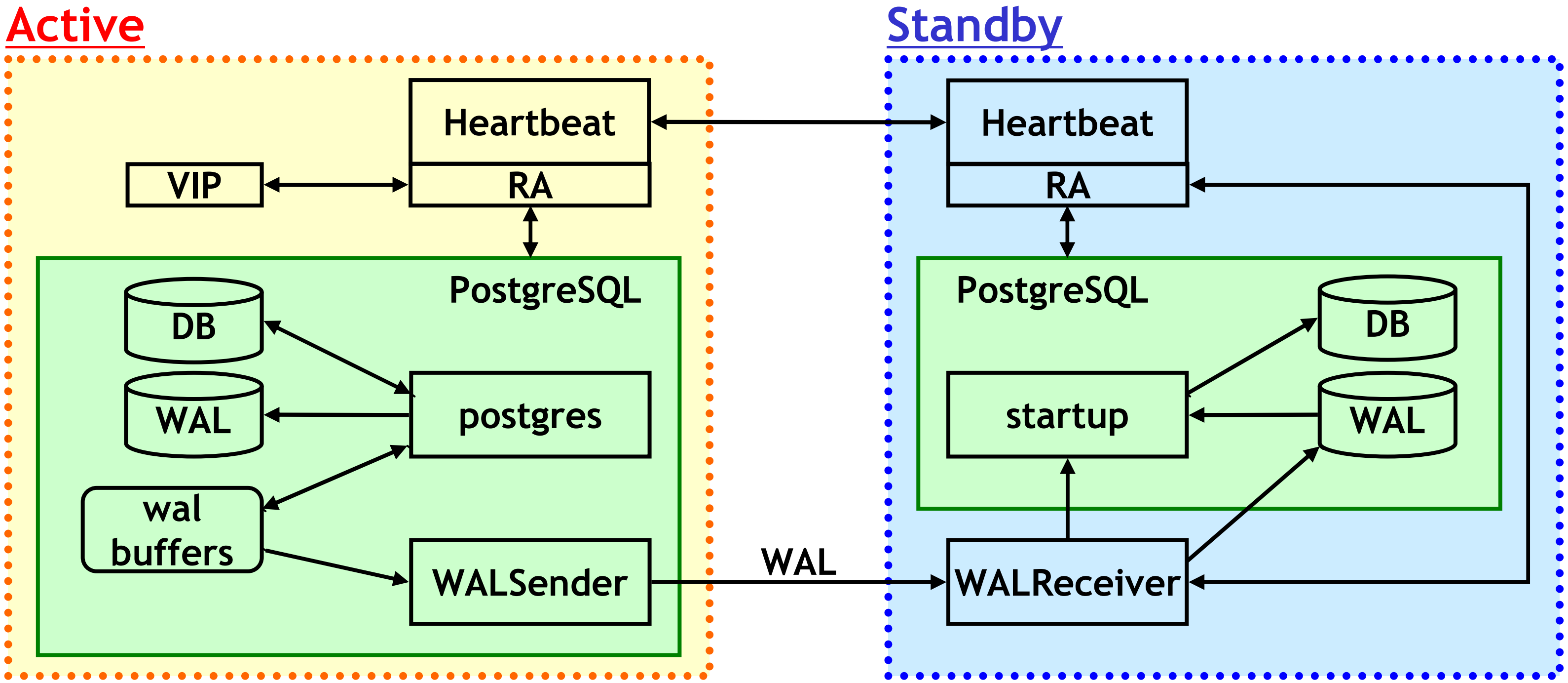
- Based on PostgreSQL 8.2, 8.3(under porting)
- WALSender
  - New child process of postmaster
  - Reads WAL from walbuffers and sends WAL to WALReceiver
- WALReceiver
  - New daemon to receive WAL
  - Writes WAL to disk and communicates with startup process
- Using Heartbeat 2.1
  - Open source high-availability software manages the resources via resource agent(RA)
  - Heartbeat provides a virtual IP(VIP)



In order to manage these resources, there is heartbeat in both nodes

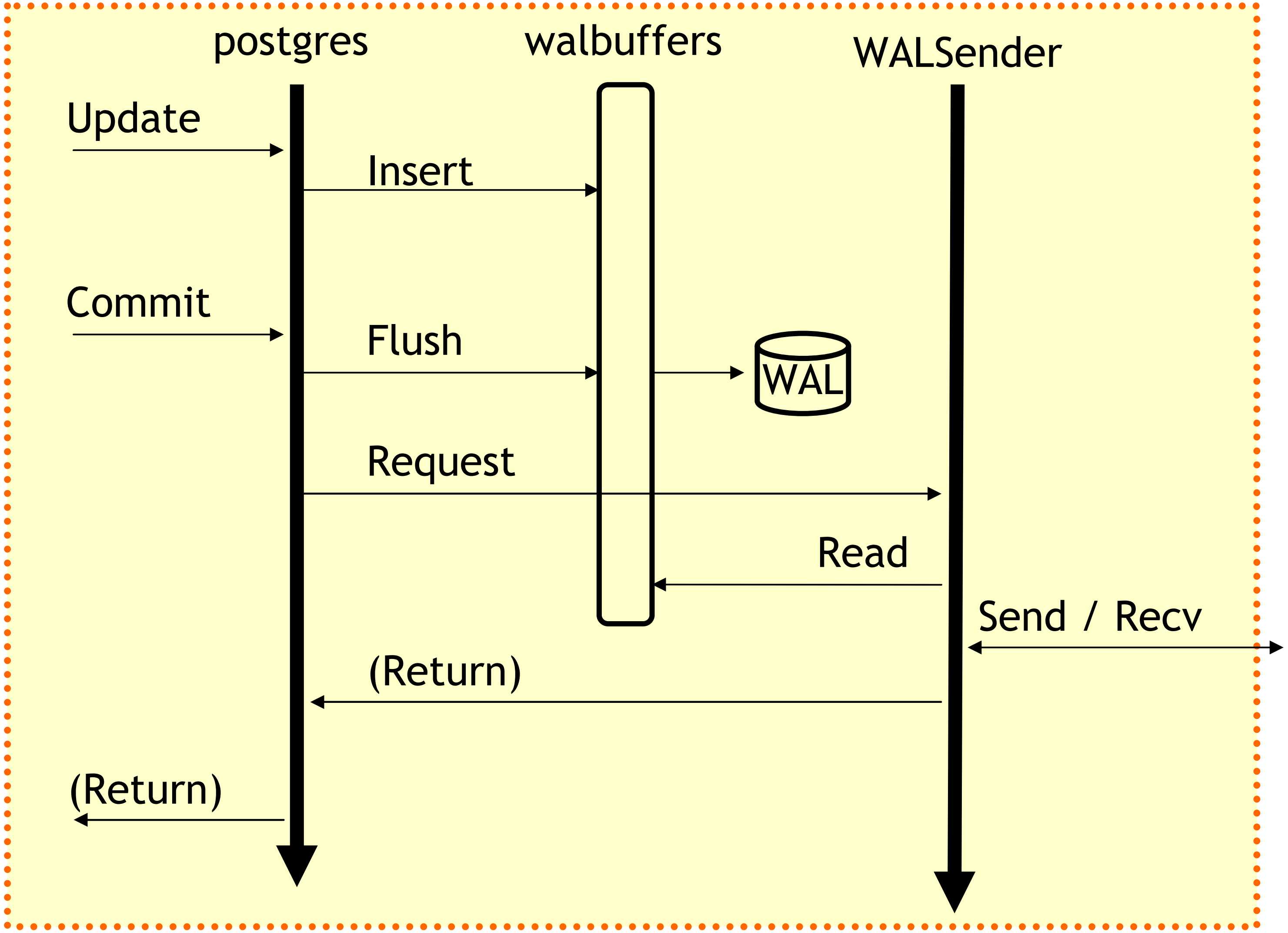
System overview

- Based on PostgreSQL 8.2, 8.3(under porting)
- WALSender
  - New child process of postmaster
  - Reads WAL from walbuffers and sends WAL to WALReceiver
- WALReceiver
  - New daemon to receive WAL
  - Writes WAL to disk and communicates with startup process
- Using Heartbeat 2.1
  - Open source high-availability software manages the resources via resource agent(RA)
  - Heartbeat provides a virtual IP(VIP)

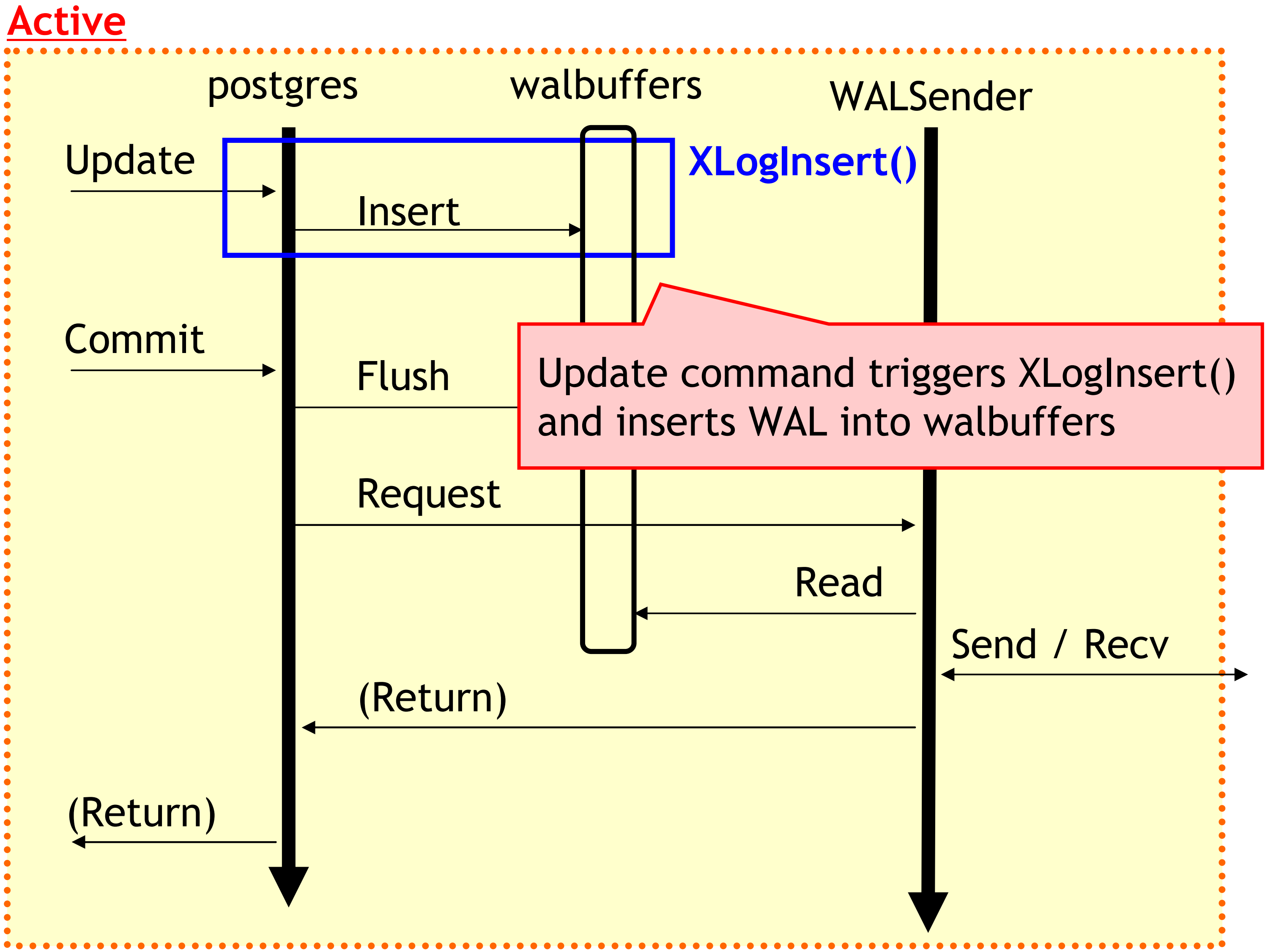


WALSender

Active



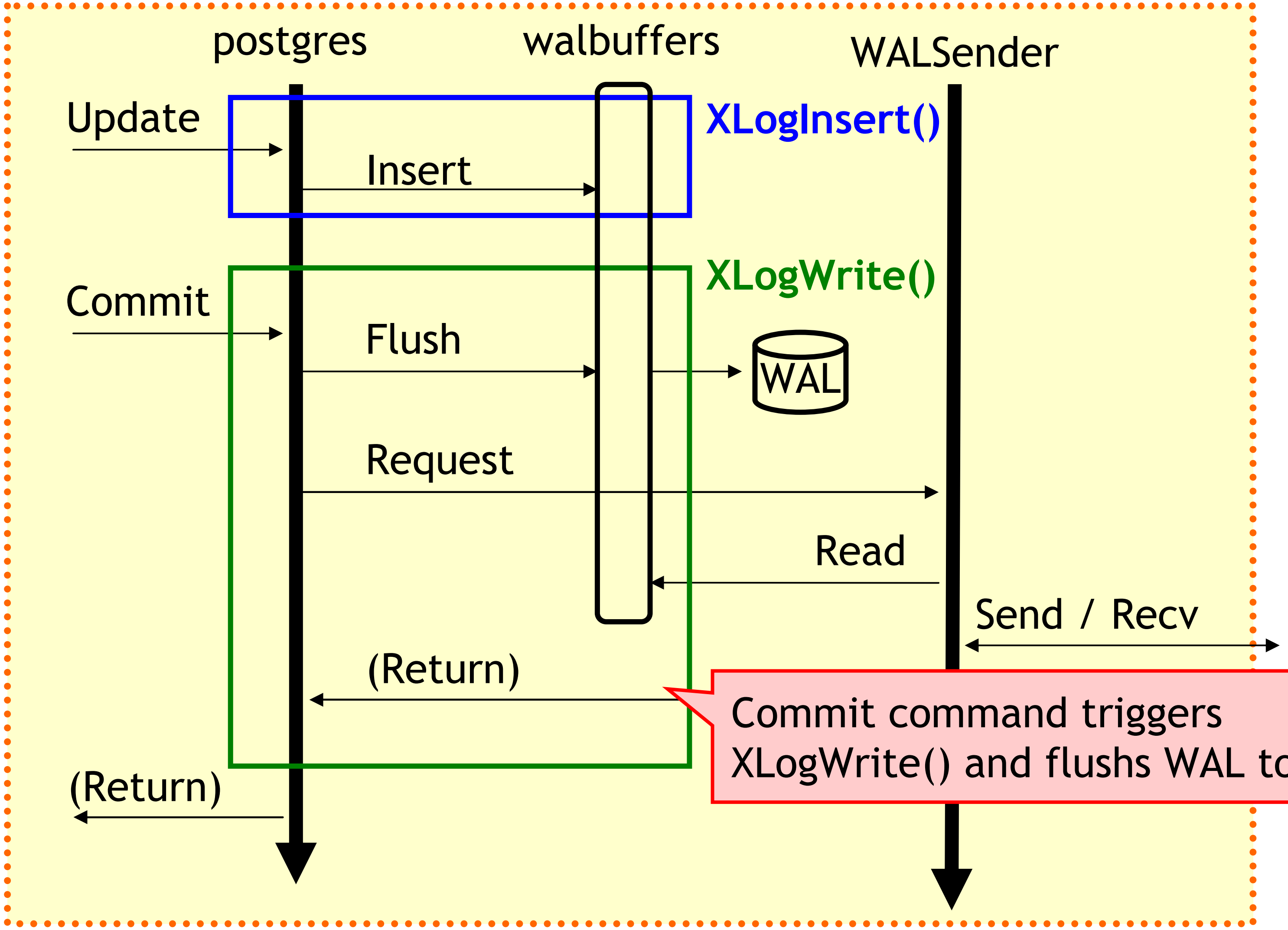
WALSender



Update command triggers XLogInsert() and inserts WAL into walbuffers

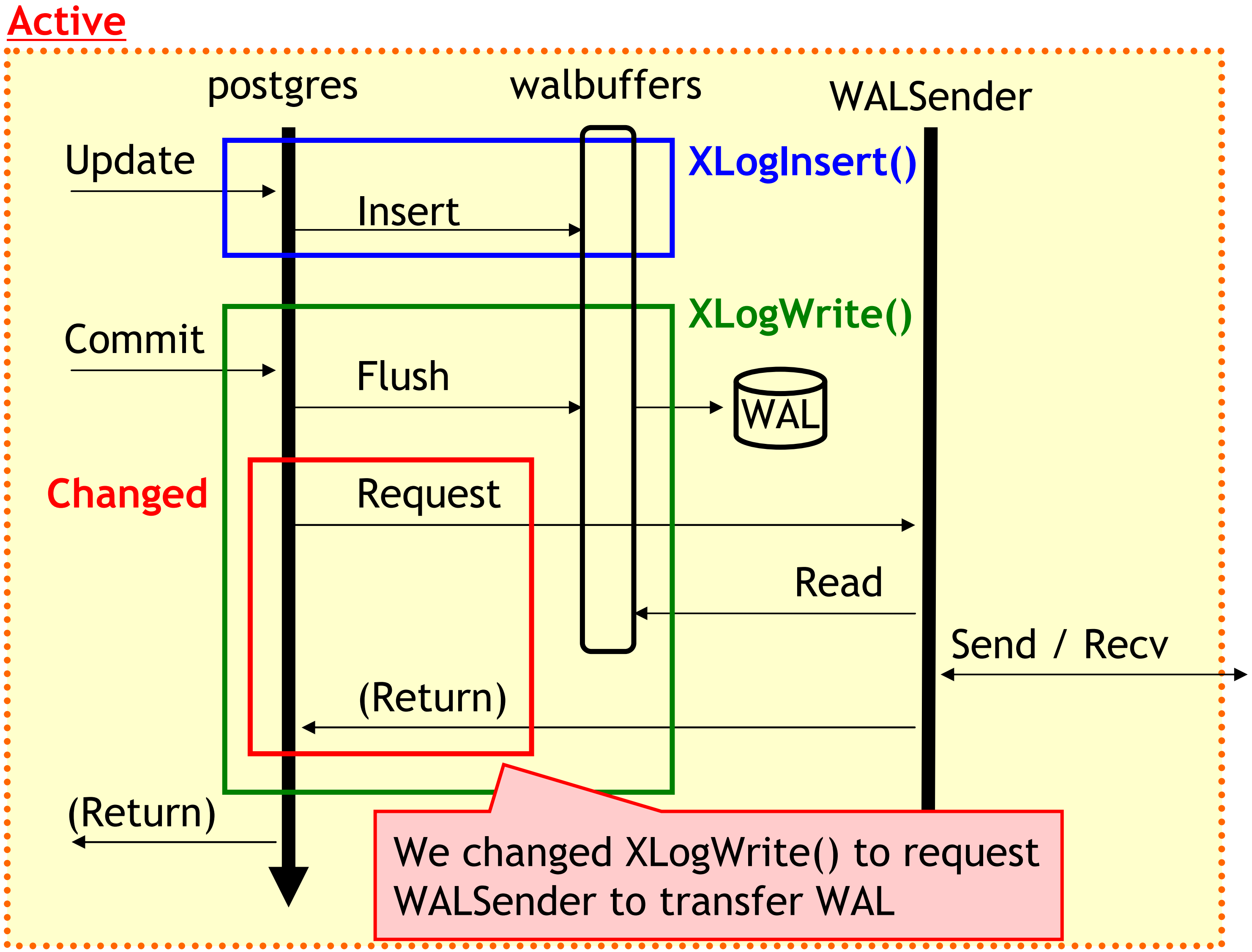
WALSender

Active



Commit command triggers XLogWrite() and flushes WAL to disk

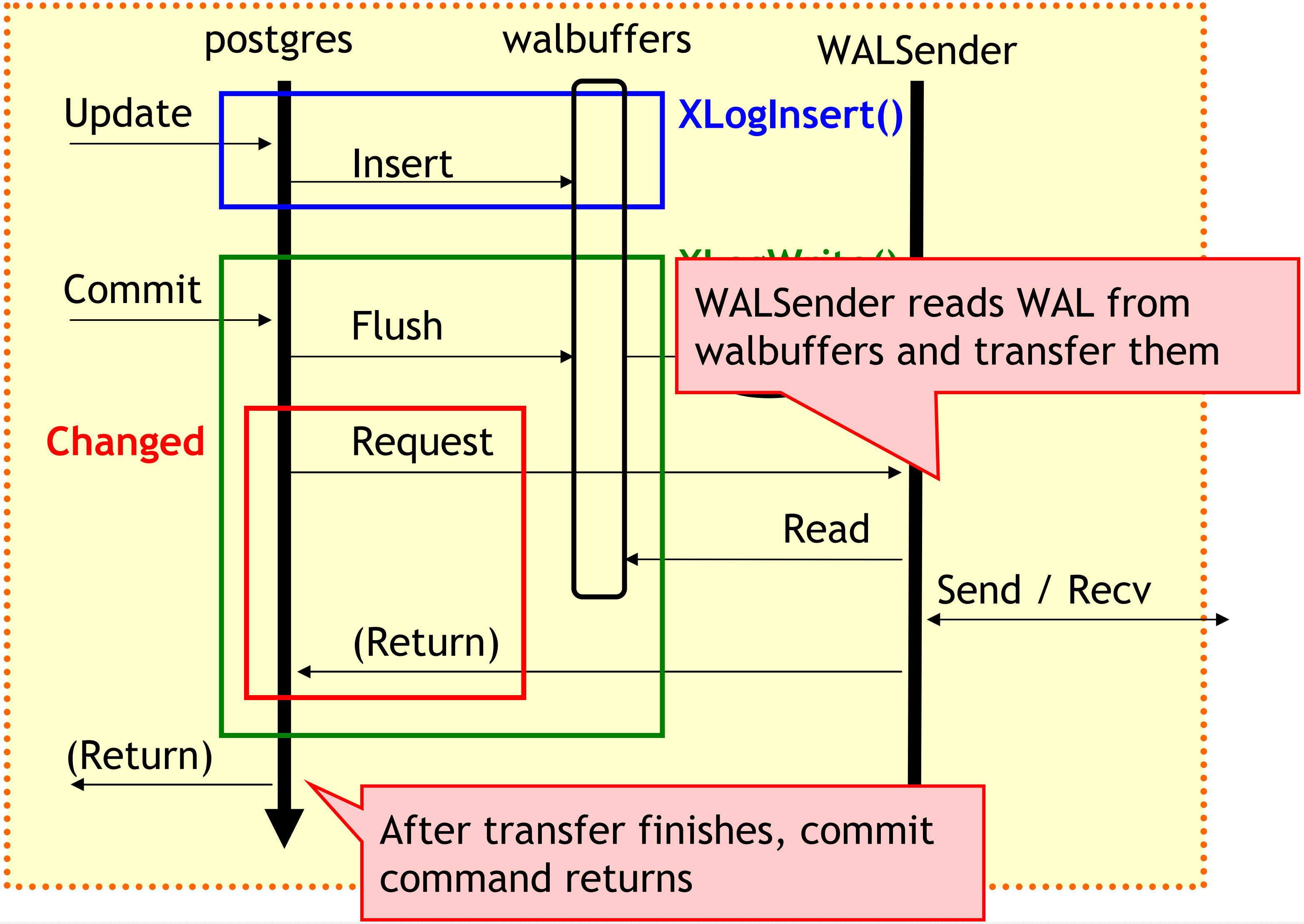
WALSender



We changed XLogWrite() to request WALSender to transfer WAL

WALSender

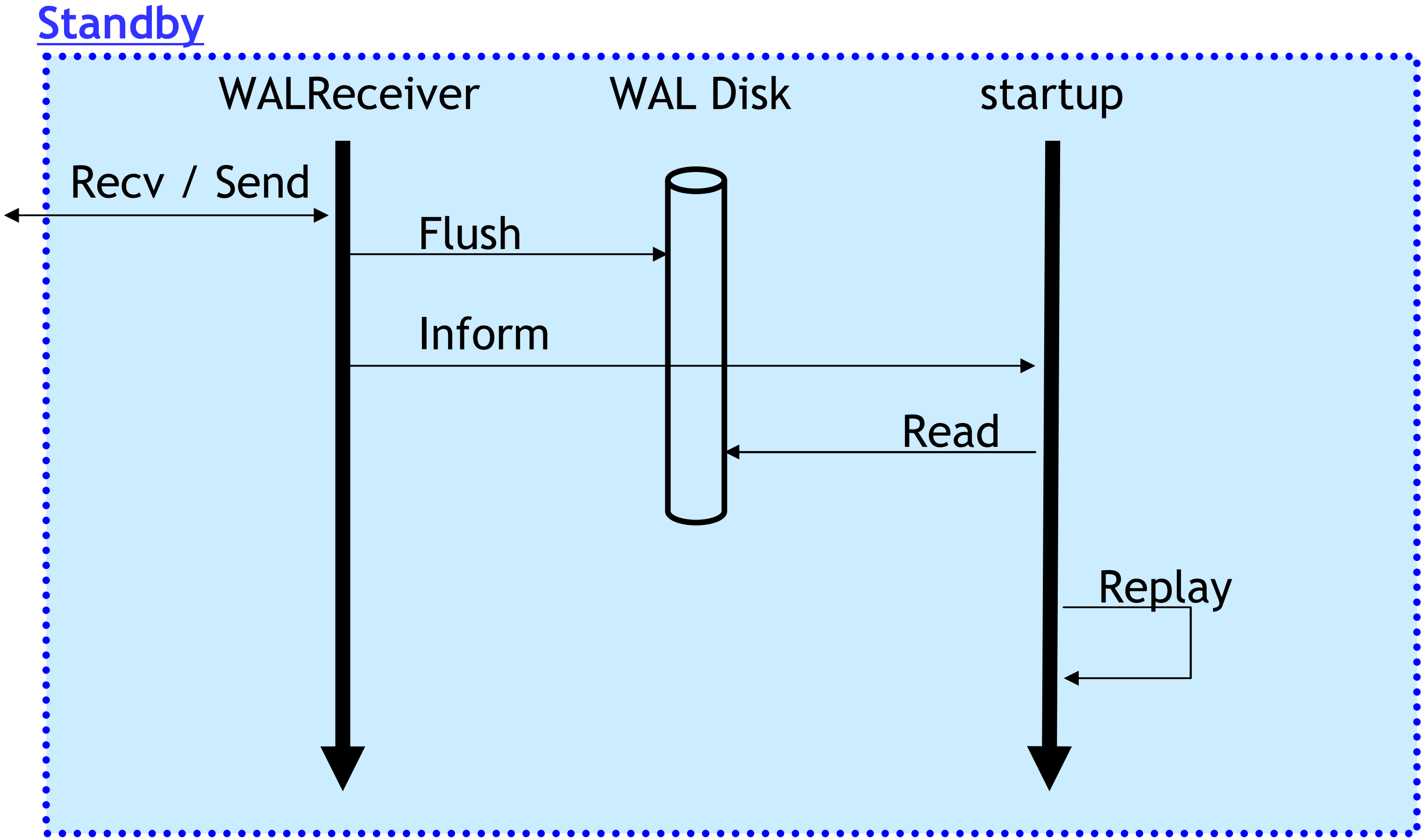
Active



WALSender reads WAL from walbuffers and transfer them

After transfer finishes, commit command returns

WALReceiver



Standby

WALReceiver

WAL Disk

startup

Recv / Send

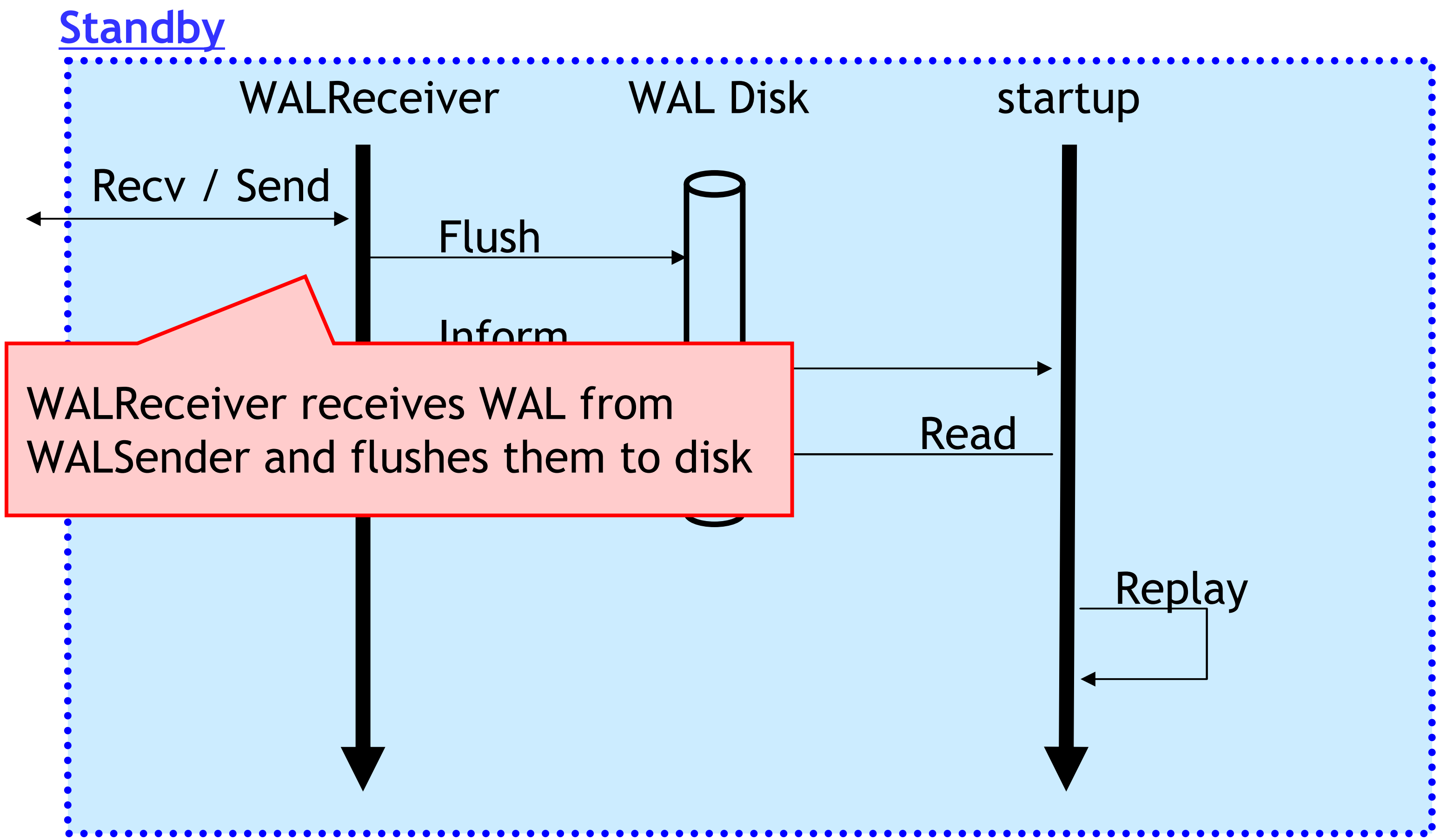
Flush

Inform

Read

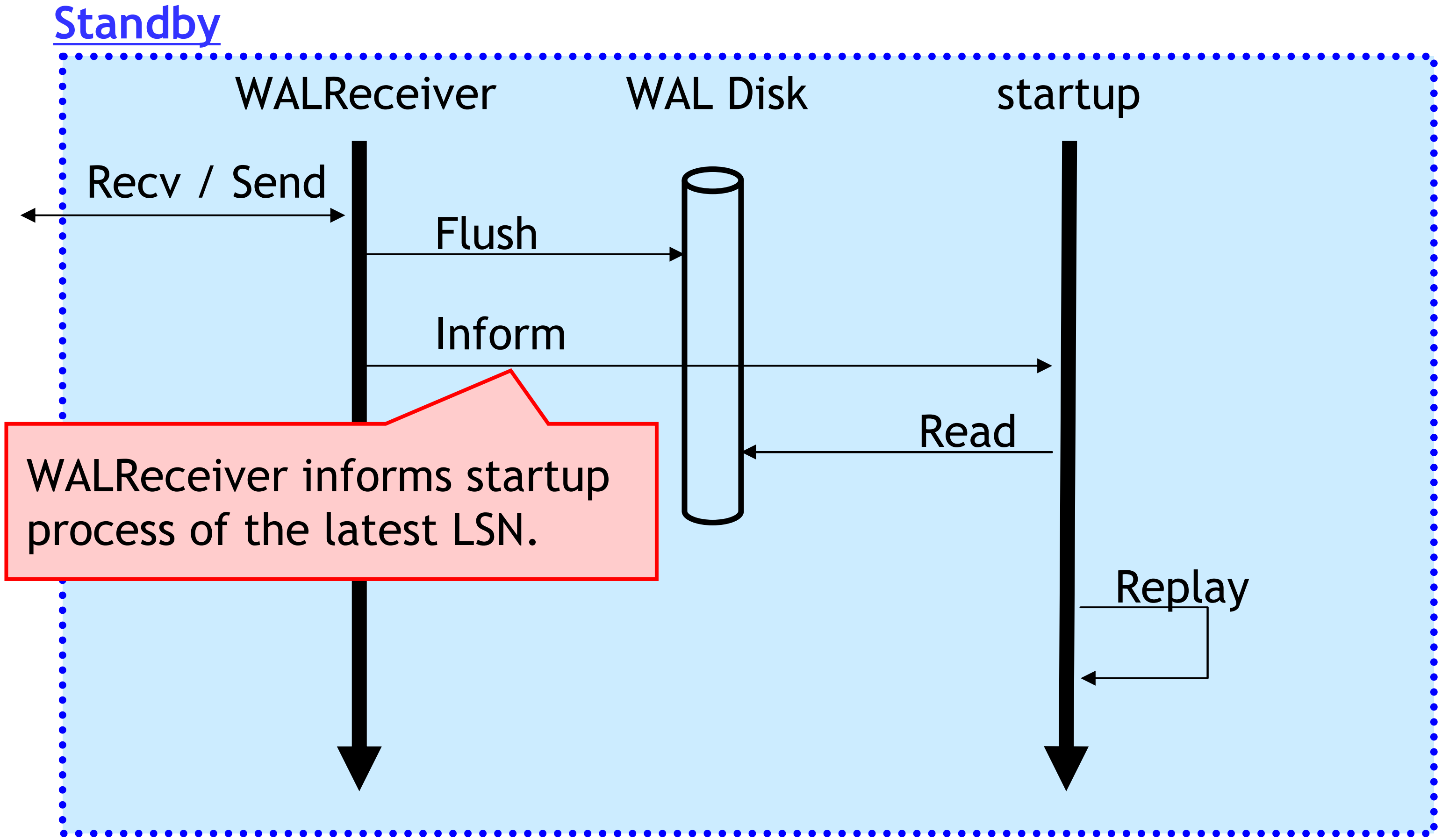
Replay

WALReceiver



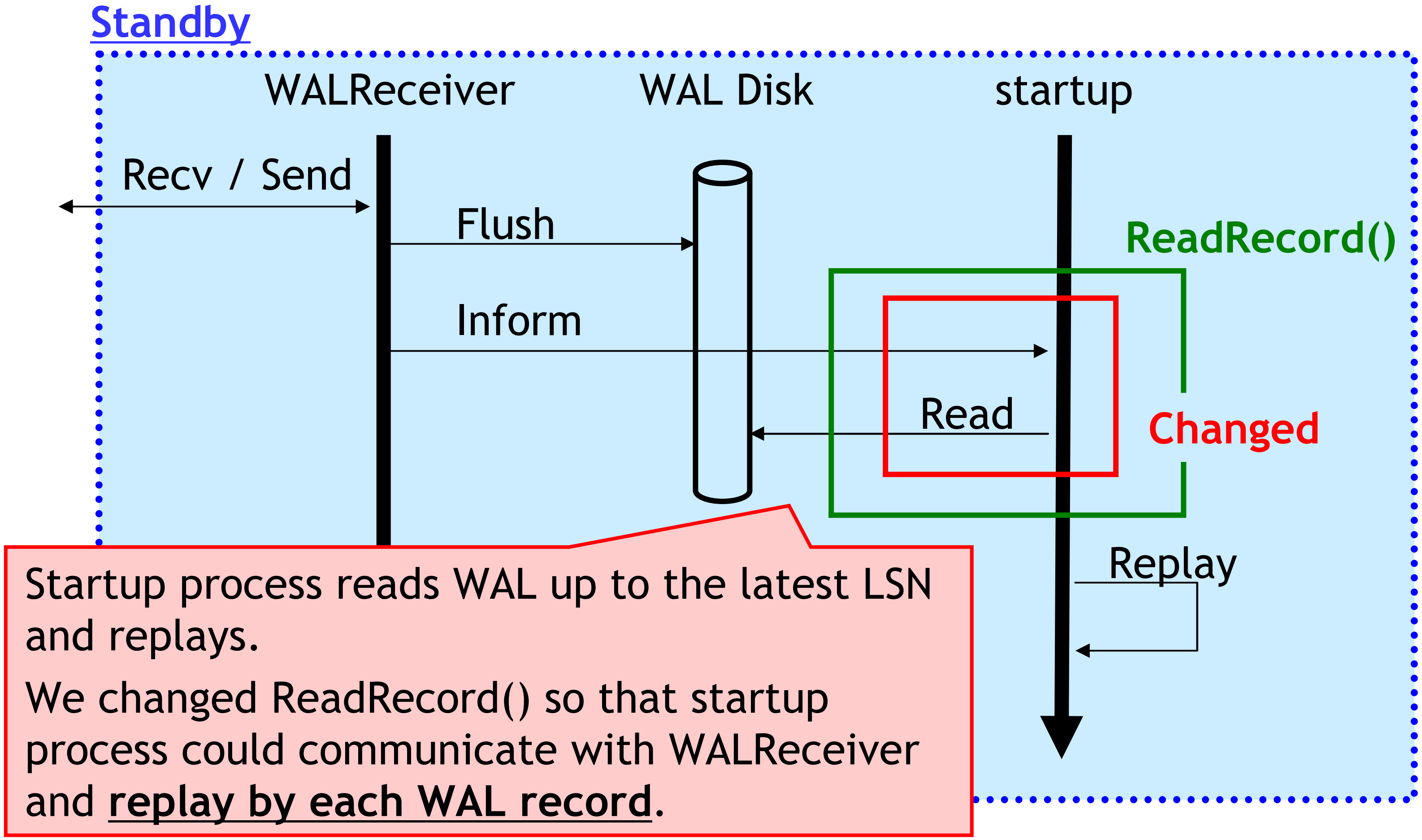
WALReceiver receives WAL from WALSender and flushes them to disk

WALReceiver



WALReceiver informs startup process of the latest LSN.

WALReceiver



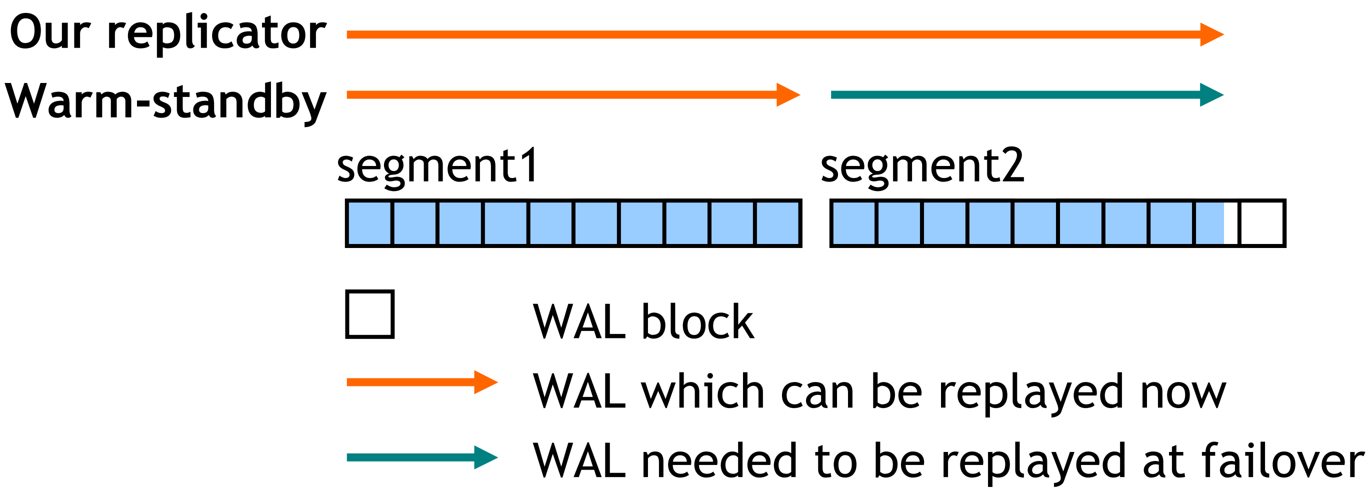
Startup process reads WAL up to the latest LSN and replays.

We changed ReadRecord() so that startup process could communicate with WALReceiver and **replay by each WAL record**.

Why replay by each WAL record?

- Minimize downtime
- Shorter delay in read-only queries (at the standby)

	Our replicator	Warm-Standby
Replay by each	WAL record	WAL segment
Needed to be replayed at failover	a few records	the latest one segment
Delay in read-only queries	shorter	longer

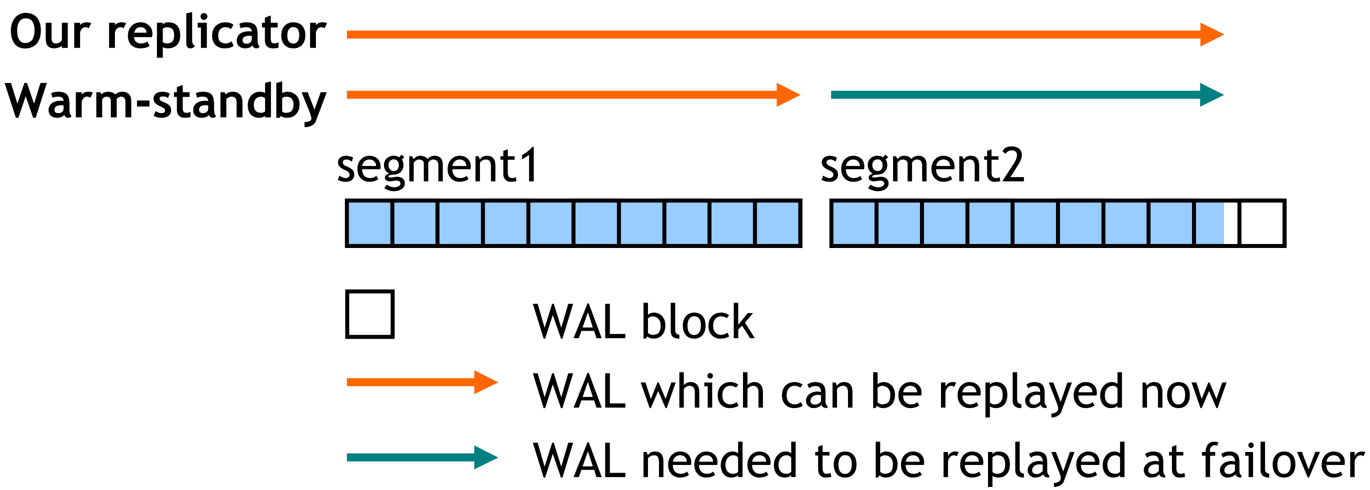

Our replicator
Warm-standby
segment1
segment2
WAL block
WAL which can be replayed now
WAL needed to be replayed at failover

Why replay by each WAL record?

- Minimize downtime
- Shorte

In our replicator, because of replay by each **WAL record**, the standby only has to replay **a few records** at failover

	Our replicator	Warm-Standby
Replay by each	WAL record	WAL segment
Needed to be replayed at failover	a few records	the latest one segment
Delay in read-only queries	shorter	longer



Our replicator

Warm-standby

segment1

segment2

□ WAL block

WAL which can be replayed now

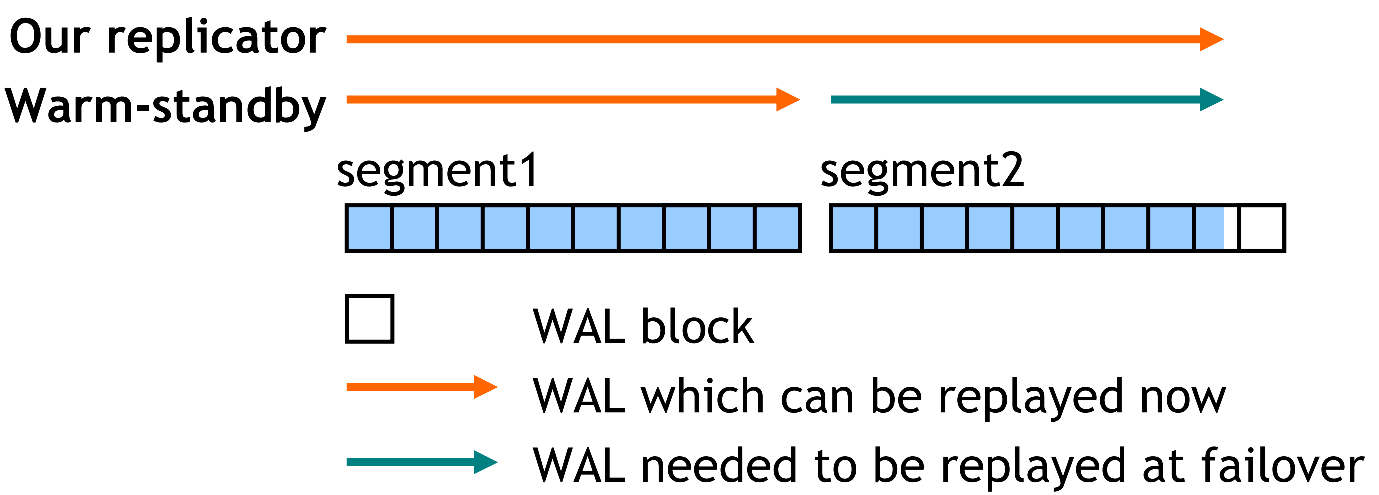
WAL needed to be replayed at failover

Why replay by each WAL record?

- Minimize do
- Shorter dela

On the other hand, in warm-standby, because of replay by each **WAL segment**, the standby has to replay **the latest one segment**

	Our replicator	Warm-Standby
Replay by each	WAL record	WAL segment
Needed to be replayed at failover	a few records	the latest one segment
Delay in read-only queries	shorter	longer



□ WAL block

WAL which can be replayed now

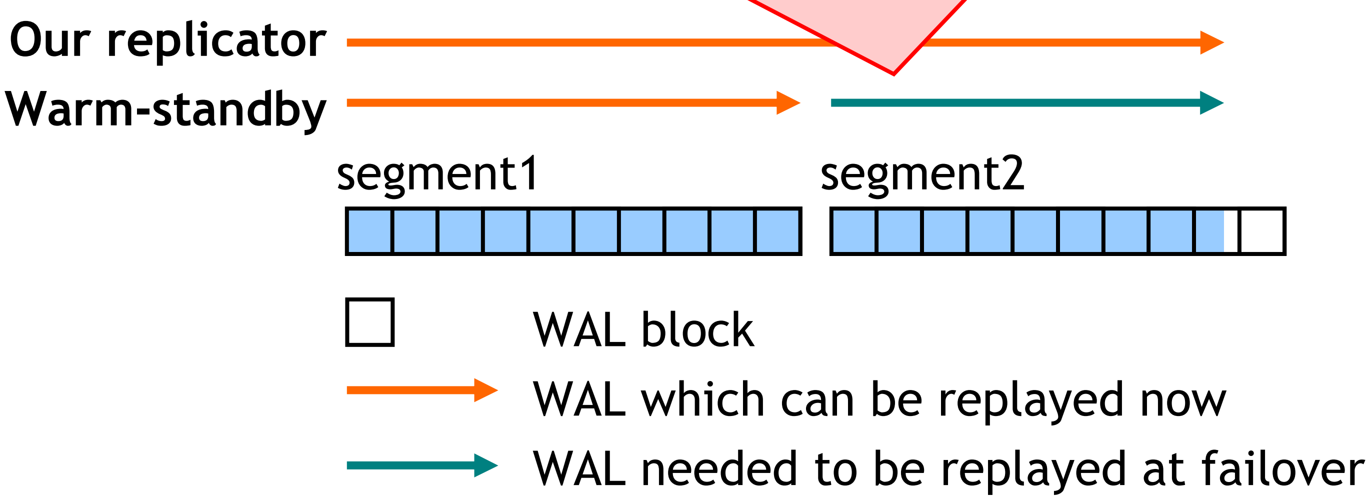
WAL needed to be replayed at failover

Why replay by each WAL record?

- Minimize downtime
- Shorter delay in read-only queries (at the standby)

	Our replicator	Warm-Standby
Replay by each	WAL record	WAL segment
Needed to be replayed at failover	a few records	the latest one segment
Delay in read-only queries		

In this example, warm-standby needed to replay most 'segment2' at failover.

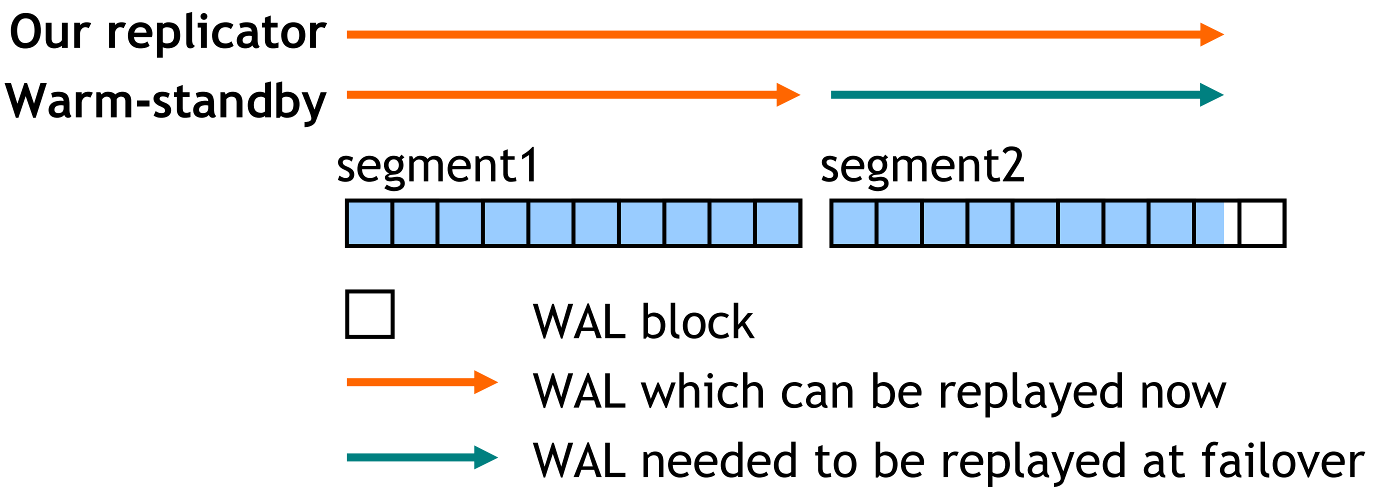

Our replicator
Warm-standby
segment1
segment2
WAL block
WAL which can be replayed now
WAL needed to be replayed at failover

Why replay by each WAL record?

- Minimize downtime
- nly queries (at the standby)

And, in our replicator, because of replay by each WAL record, delay in read-only queries is shorter

	Our replicator	Warm-Standby
Replay by each	WAL record	WAL segment
Needed to be replayed failover	a few records	the latest one segment
Delay in read-only queries	shorter	longer

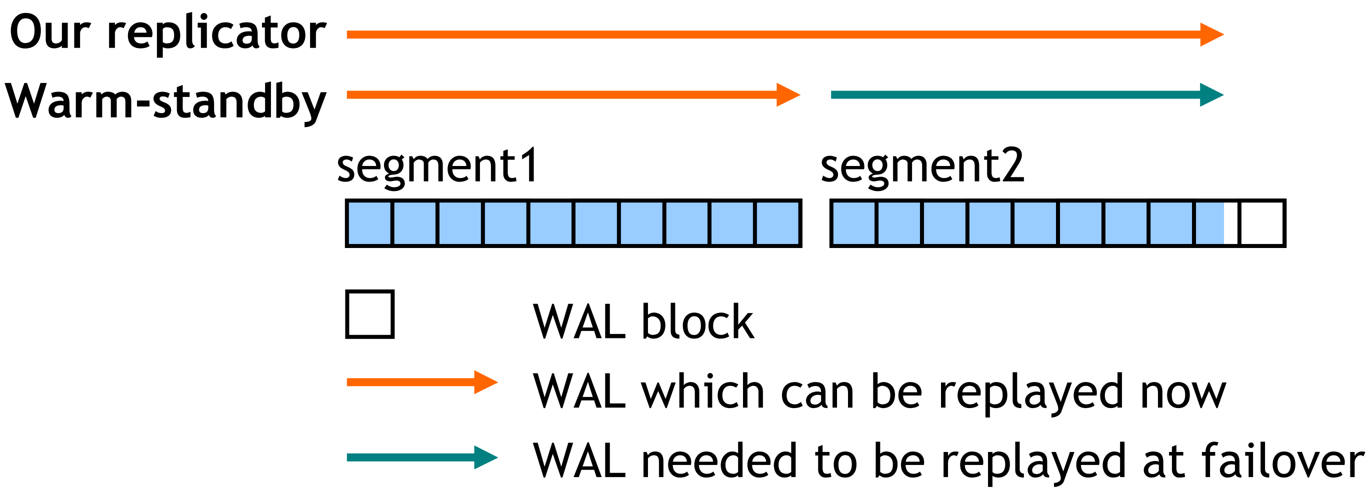

Our replicator
Warm-standby
segment1
segment2
WAL block
WAL which can be replayed now
WAL needed to be replayed at failover

Why replay by each WAL record?

- Minimize downtime
- Shorter delay in read-only queries (at the standby)

Therefore, we implemeted replay by each WAL record

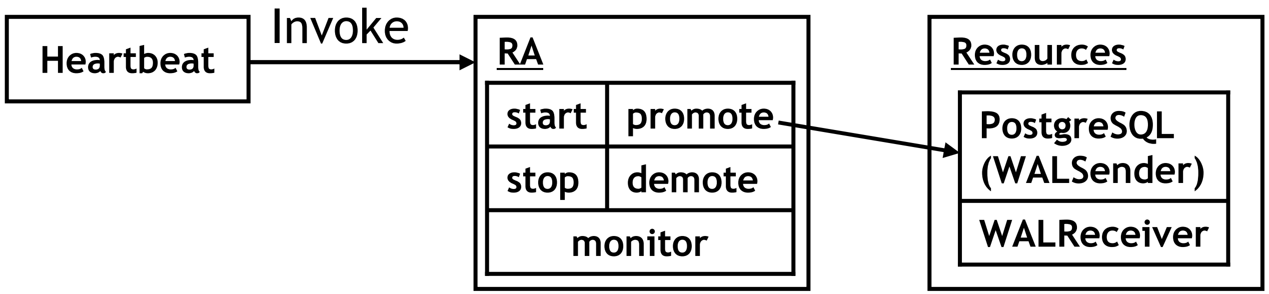
	or	Warm-Standby
Replay by ea		WAL segment
Needed to be replayed at failover	a few records	the latest one segment
Delay in read-only queries	shorter	longer


Our replicator
Warm-standby
segment1
segment2
WAL block
WAL which can be replayed now
WAL needed to be replayed at failover

Heartbeat and resource agent

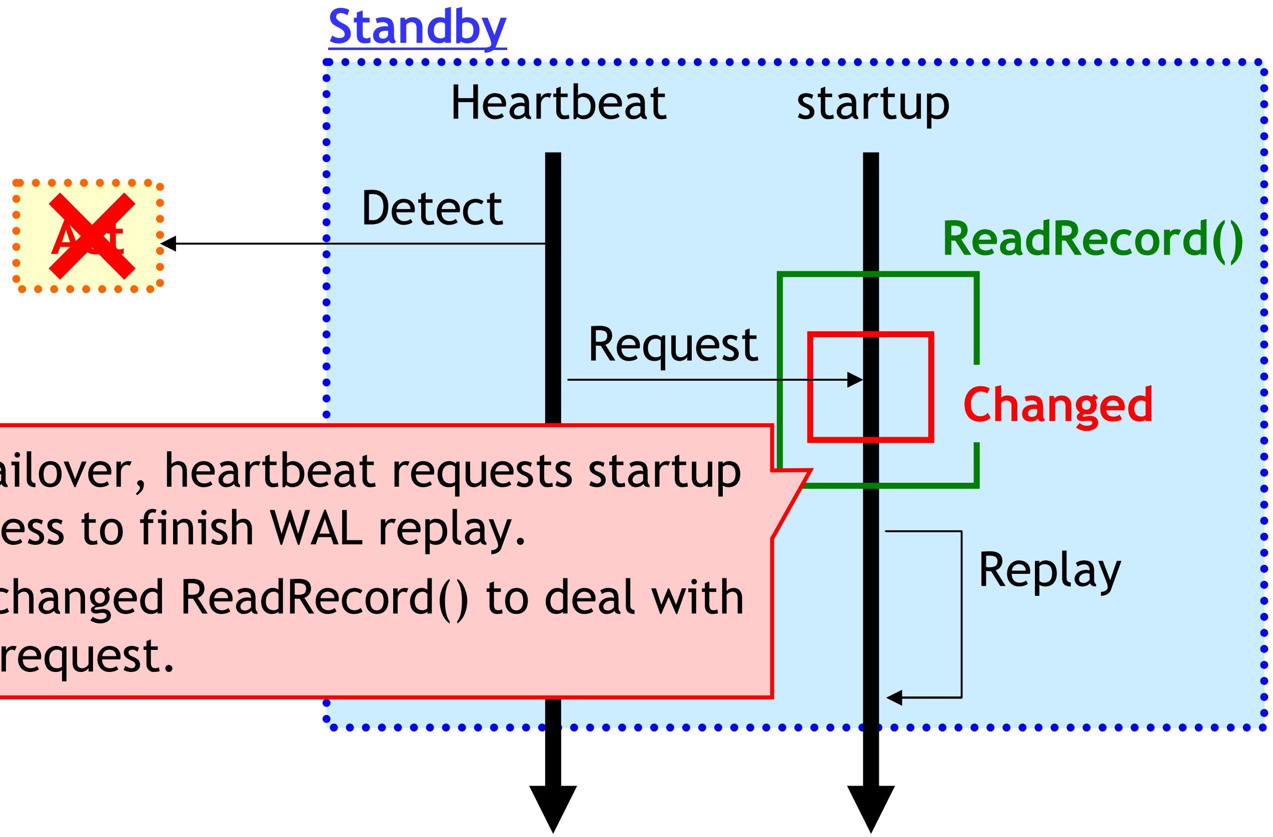
- Heartbeat needs resource agent (RA) to manage PostgreSQL(with WALSender) and WALReceiver as a resource
- RA is an executable providing the following feature

Feature	Description
start	start the resources as standby
promote	change the status from standby to active
demote	change the status from active to standby
stop	stop the resources
monitor	check if the resource is running normally



Failover

- Failover occurs when heartbeat detects that the active node is not running normally
- After failover, clients can restart transactions only by reconnecting to virtual IP provided by Heartbeat

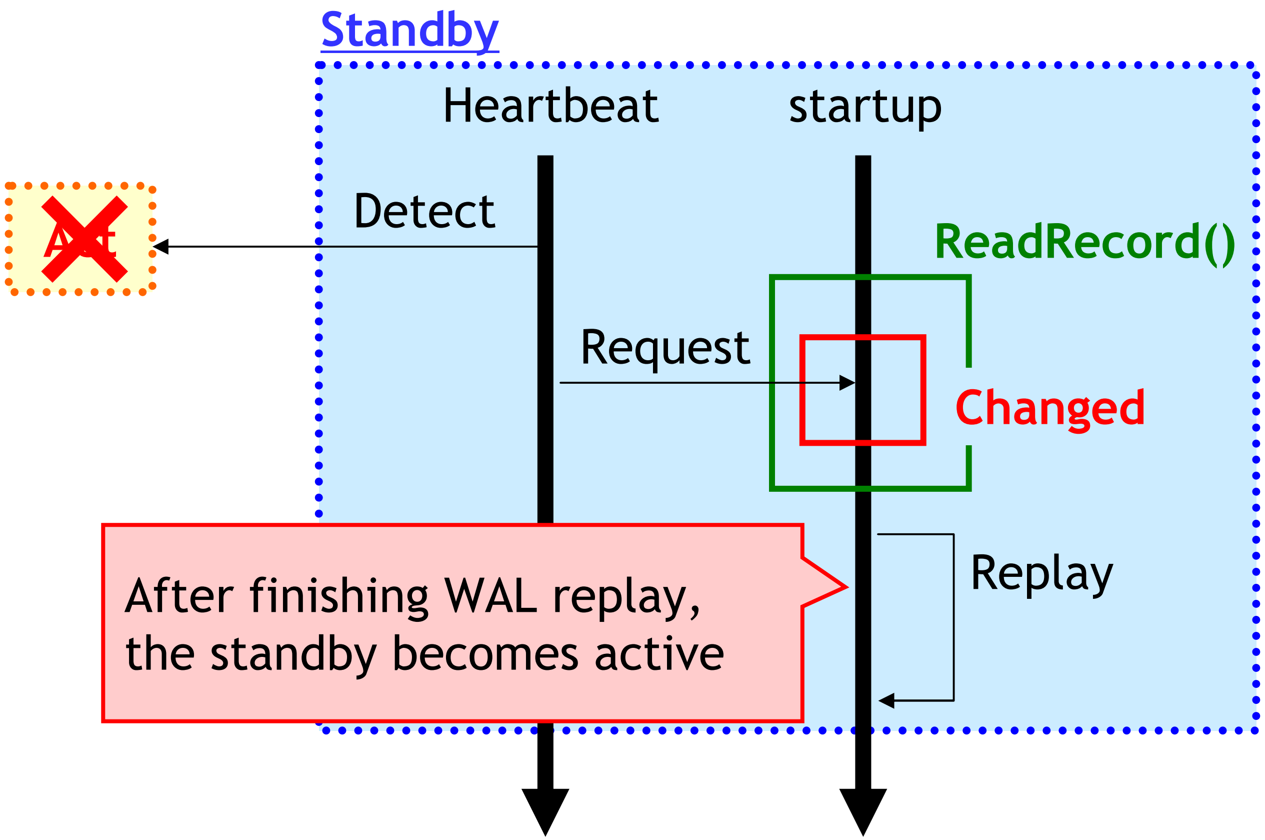


At failover, heartbeat requests startup process to finish WAL replay.

We changed ReadRecord() to deal with this request.

Failover

- Failover occurs when heartbeat detects that the active node is not running normally
- After failover, clients can restart transactions only by reconnecting to virtual IP provided by Heartbeat

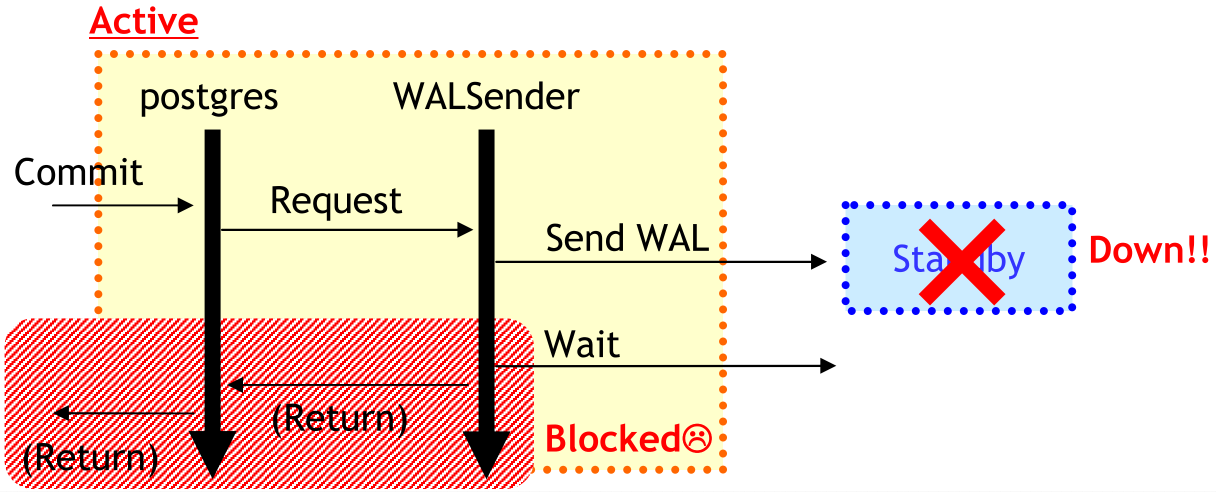


After finishing WAL replay, the standby becomes active

Struggles in development

Downtime caused by the standby down

- The active down triggers a failover and causes downtime
- Additionally, **the standby down** might also cause downtime
  - WALSender waits for the response from the standby after sending WAL
  - So, when the standby down occurs, **unless WALSender detects the failure**, WALSender is blocked
  - i.e. WALSender keeps waiting for the response which never comes
- How to detect
  - Timeout notification is needed to detect
  - Keepalive, but it doesn't work occasionally on Linux (Linux bug!?)
  - Original timeout

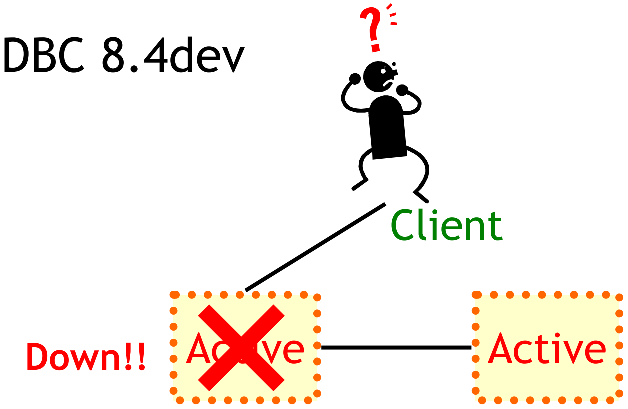


Downtime caused by clients

- Even if the database finishes a failover immediately, downtime might still be long **by clients reason**
  - Clients wait for the response from the database
  - So, when a failover occurs, **unless clients detect a failover**, they can't reconnect to the new active and restart the transaction
  - i.e. clients keeps waiting for the response which never comes

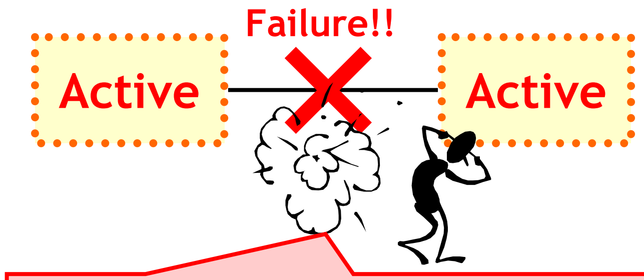
- How to detect
  - Timeout notification is needed to detect
  - Keepalive
    - Our setKeepAlive patch was accepted in JDBC 8.4dev
  - Socket timeout
  - Query timeout

We want to implement these timeouts!!



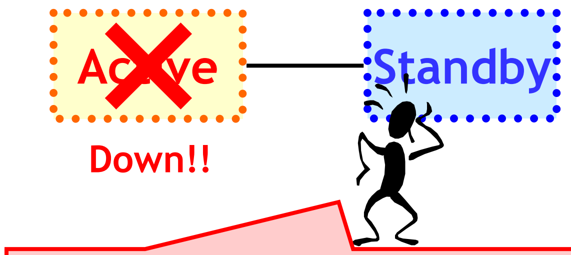
Split-brain

- High-availability clusters must be able to handle split-brain
- Split-brain causes data inconsistency
  - Both nodes are active and provide the virtual IP
  - So, clients might update inconsistently each node
- Our replicator also causes split-brain unless the standby can distinguish network failure from the active down



If network failure are mis-detected as the active down, the standby becomes active even if the other active is still running normally.

This is split-brain scenario.



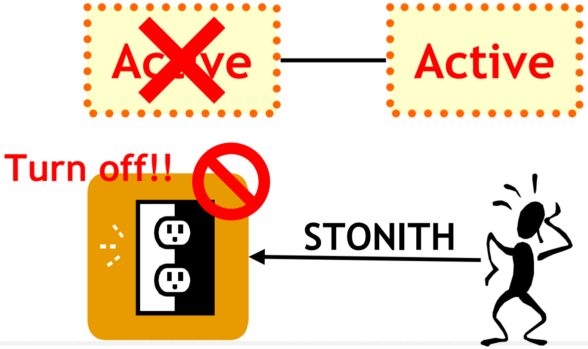
If the active down are mis-detected as network failure, a failover doesn't start even if the other active is down.

This scenario is also problem though split-brain doesn't occur.

Split-brain

- How to distinguish
  - Combining the following solution

1. Redundant network between two nodes
   - The standby can distinguish unless all networks fail
2. STONITH(Shoot The Other Node In The Head)
   - Heartbeat's default solution for avoiding split-brain
   - STONITH always forcibly turns off the active when activating the standby
   - Split-brain doesn't occur because the active node is always only one



What delays the activation of the standby

- In order to activate the standby immediately, recovery time at failover must be short!!
- In 8.2, recovery is very slow☹
  - A lot of WAL needed to be replayed at failover might be accumulated
  - Another problem: disk full failure might happen
- In 8.3, reocvery is fast☺
  - Because of avoiding unnecessary reads
  - But, there are still two problems

What delays the activation of the standby

1. Checkpoint during recovery
   - It took 1min or more (in the worst case) and occupied 21% of recovery time
   - What is worse is that WAL replay is blocked during checkpoint
     - Because only startup process performs both checkpoint and WAL replay

   -> Checkpoint delays recovery...☹

- [Just idea] bgwriter during recovery
   - Leaving checkpoint to bgwriter, and making startup process concentrate on WAL replay

What delays the activation of the standby

2. Checkpoint at the end of recovery
   - Activation of the standby is blocked during checkpoint

   -> Downtime might take 1min or more...☹

- [Just idea] Skip of the checkpoint at the end of recovery
  - But, postgres works fine if it fails before at least one checkpoint after recovery?
  - We have to reconsider why checkpoint is needed at the end of recovery

!!! Of course, because recovery is a critical part for DBMS, more careful investigation is needed to realize these ideas

How we choose the node with the later LSN

- When starting both two nodes, we should synchronize from the node with the later LSN to the other
  - But, it's unreliable to depend on server logs (e.g. heartbeat log) or a human memory in order to choose the node

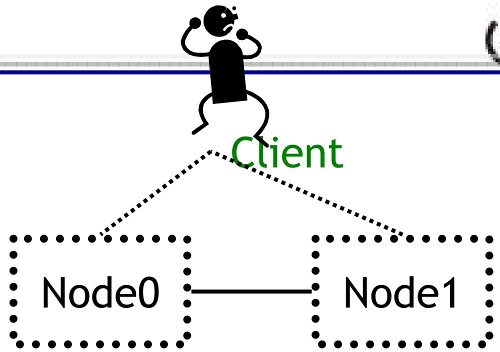
- We choose the node from WAL which is most reliable
  - Find the latest LSN from WAL files in each node by using our original tool like xlogdump and compare them

Bottleneck

- Bad performance after failover
  - No FSM
  - A little commit hint bits in heap tuples
  - A little dead hint bits in indexes

Demo

Demo

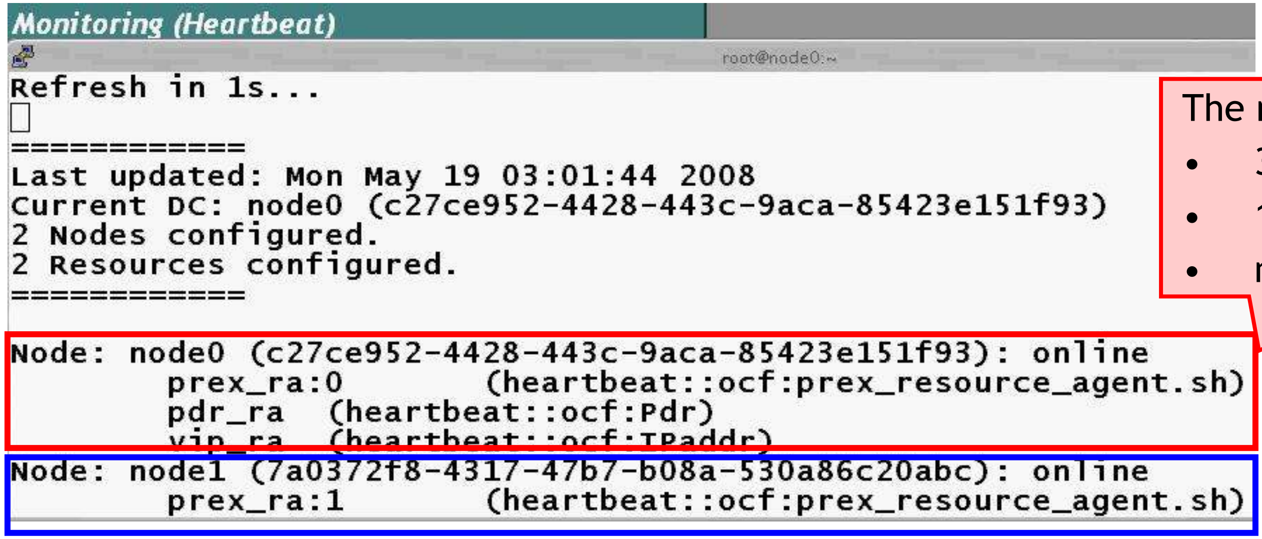


Environment

- 2 nodes, 1 client

How to watch

- there are two kind of terminals
- the terminal at the top of the screen displays the cluster status



```
Refresh in 1s...

============
Last updated: Mon May 19 03:01:44 2008
Current DC: node0 (c27ce952-4428-443c-9aca-85423e151f93)
2 Nodes configured.
2 Resources configured.
============

Node: node0 (c27ce952-4428-443c-9aca-85423e151f93): online
        prex_ra:0       (heartbeat::ocf:prex_resource_agent.sh)
        pdr_ra  (heartbeat::ocf:Pdr)
        vip_ra  (heartbeat::ocf:IPaddr)
Node: node1 (7a0372f8-4317-47b7-b08a-530a86c20abc): online
        prex_ra:1       (heartbeat::ocf:prex_resource_agent.sh)
```

The node with
- 3 lines is active
- 1 line is standby
- no line is not started yet

active

standby

- the other terminal is for operation
  - Client
  - Node0
  - Node1

Demo

Operation

1. start only node0 as the active
2. createdb and pgbench -i (from client)
3. online backup
4. copy the backup from node0 to node1
5. pgbench -c2 -t2000
6. start node1 as the standby during pgbench -> synchronization starts
7. killall -9 postgres (in active node0) -> failover occurs

Future work

- Where are we going? -

Where are we going?

- We’re thinking to make it Open Source Software.
  - To be a multi-purpose replication framework
  - Collaborators welcome.
- TODO items
  - For 8.4 development
    - Re-implement WAL-Sender and WAL-Receiver as extensions using two new hooks
    - Xlogdump to be an official contrib module
  - For performance
    - Improve checkpointing during recovery
    - Handling un-logged operations
  - For usability
    - Improve detection of server down in client library
    - Automatic retrying abundant transactions in client library

For 8.4 : WAL-writing Hook

- Purpose
  - Make WAL-Sender to be one of general extensions
    - WAL-Sender sends WAL records before commits
- Proposal
  - Introduce “WAL-subscriber model”
  - “WAL-writing Hook” enables to replace or filter WAL records just before they are written down to disks.
- Other extensions using this hook
  - “Software RAID” WAL writer for redundancy
    - Writes WAL into two files for durability (it might be a paranoia…)
  - Filter to make a bitmap for partial backup
    - Writes changed pages into on-disk bitmaps
  - …

For 8.4 : WAL-reading Hook

- Purpose
  - Make WAL-Receiver to be one of general extensions
    - WAL-Receiver redo in each record, not in each segment
- Proposal
  - "WAL-reading Hook" enables to filter WAL records during they are read in recovery.
- Other extensions using this hook
  - Read-ahead WAL reader
    - Read a segment at once and pre-fetch required pages that are not a full-page-writes and not in shared buffers
  - …

Future work : Multiple Configurations

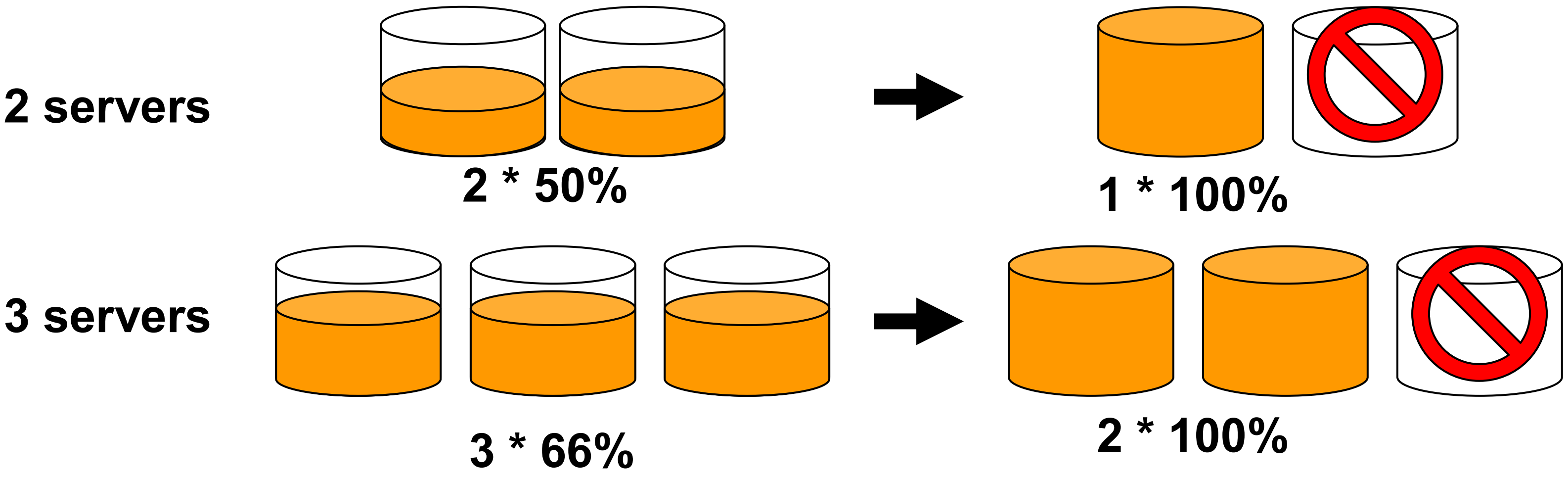
- Supports several synchronization modes
  - One configuration is not fit all,
    but one framework could fit many uses!

No.	Configuration	Before/After Commit in ACT			
		Send to SBY	Flush in ACT	Flush in SBY	Redo in SBY
1	Speed	After	After	After	After
2	Speed + Durability	After	Before	After	After
3	HA + Speed	Before	After	After	After
4	HA + Durability	Before	Before	After	After
5	HA + More durability	Before	Before	Before	After
6	Synchronous Reads in SBY	Before	Before	Before	Before

Now → 4

Future work : Horizontal scalability

- Horizontal scalability is not our primary goal, but for potential users.
- Postgres TODO: “Allow a warm standby system to also allow read-only statements” helps us.
- NOTE: We need to support 3 or more servers if we need both scalability and availability.



Conclusion

- Synchronous log-shipping is the best for HA.
  - A direction of future warm-standby
  - Less downtime, No data loss, and Automatic failover.
- There remains rooms for improvements.
  - Minimize downtime and performance scalability.
  - Improvements for recovery also helps Log-shipping.

- We’ve shown requirements, advantages, and remaining tasks.
  - It has potential to improvements, but requires some works to be more useful solution
  - We’ll make it open source! Collaborators welcome!

Fin.

Contact

itagaki.takahiro@oss.ntt.co.jp
fujii.masao@oss.ntt.co.jp