

PostgreSQL replication

a hands-on tutorial



WiFi

1) Instructions here

covered

- basic asynchronous
- configuration
- tools and monitoring
- file-based
- failover & fail-back
- synchronous
- cascading
- query cancel/lag

plus 9.4 material

- replication slots
- logical decoding

not covered

- DR planning
- 3rd-party tools
- application design
- non-binary replication
- Point In Time Recovery



**what is
binary replication?**

**vagrant up
or docker run
now**

replication terms

master / slave

master / standby

master / replica

primary / secondary

primary / replica

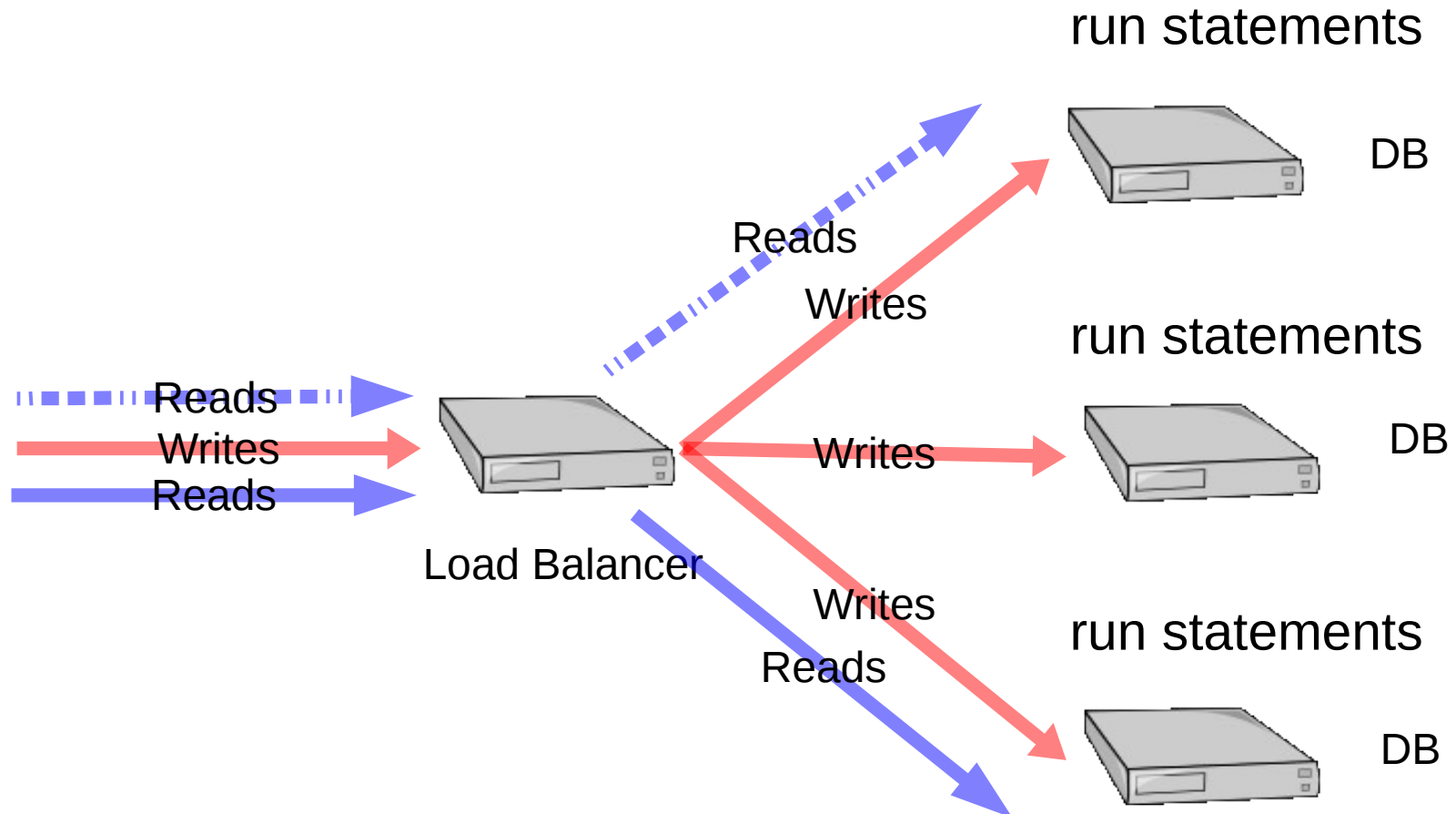
replication mechanisms

1. statement
2. row
3. binary

replication mechanisms

1. queries
2. rows
3. data pages

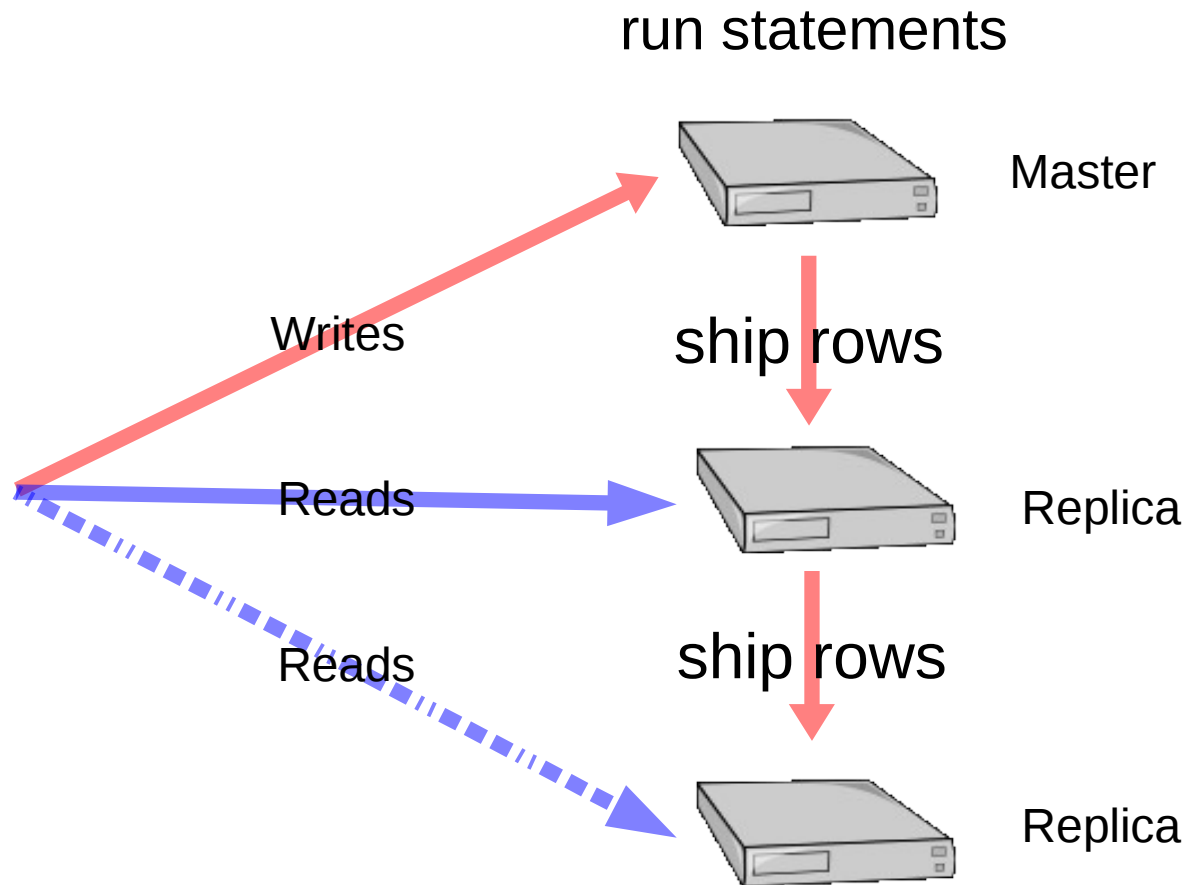
statement replication



statement replication

- pgPool2 replication
- GridSQL
- C-JDBC
- Continuent
- DBI::Multiplex
- original MySQL replication

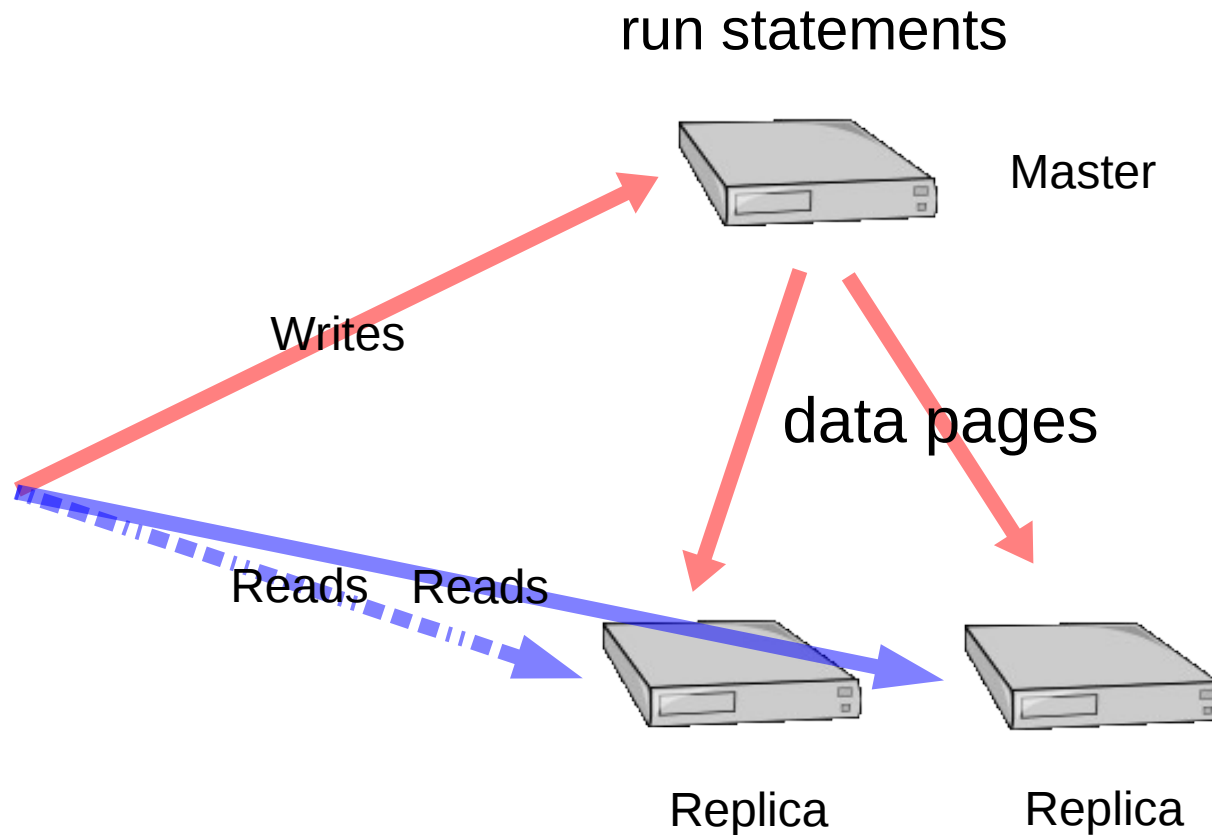
row-based replication



row-based replication

- Slony-I
- Londiste
- Bucardo
- new MySQL replication
- upcoming 9.4 replication

binary replication



DRBD for PostgreSQL

(only much much faster)

also called ...

- **streaming replication**
 - refers to the ability to stream new data pages over a network connection
- **hot standby**
 - refers to the ability of standbys to run read-only queries while in standby mode

advantages

- low administration
- low overhead on master
- non-invasive
- low-latency
- good for large DBs

disadvantages

- need to replicate the whole server
- no writes of any kind on replicas
- some things not replicated
- query cancel

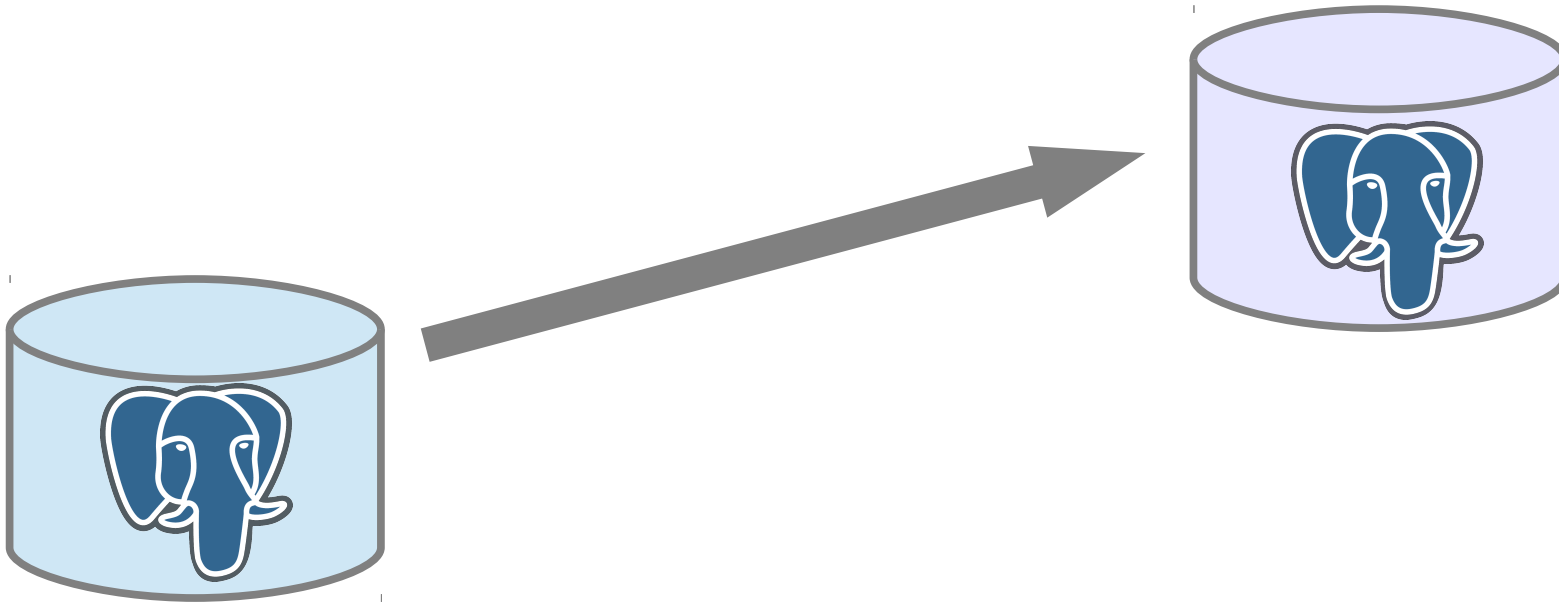
A photograph of a small, clear stream flowing through a wooded area. The stream is bordered by a low, rustic stone wall made of flat, grey stones. The water is shallow and clear, reflecting the surrounding greenery. The stream is surrounded by dense foliage, including tall grasses and various trees. The scene is captured from a slightly elevated angle, looking down the length of the stream.

streaming asynchronous replication

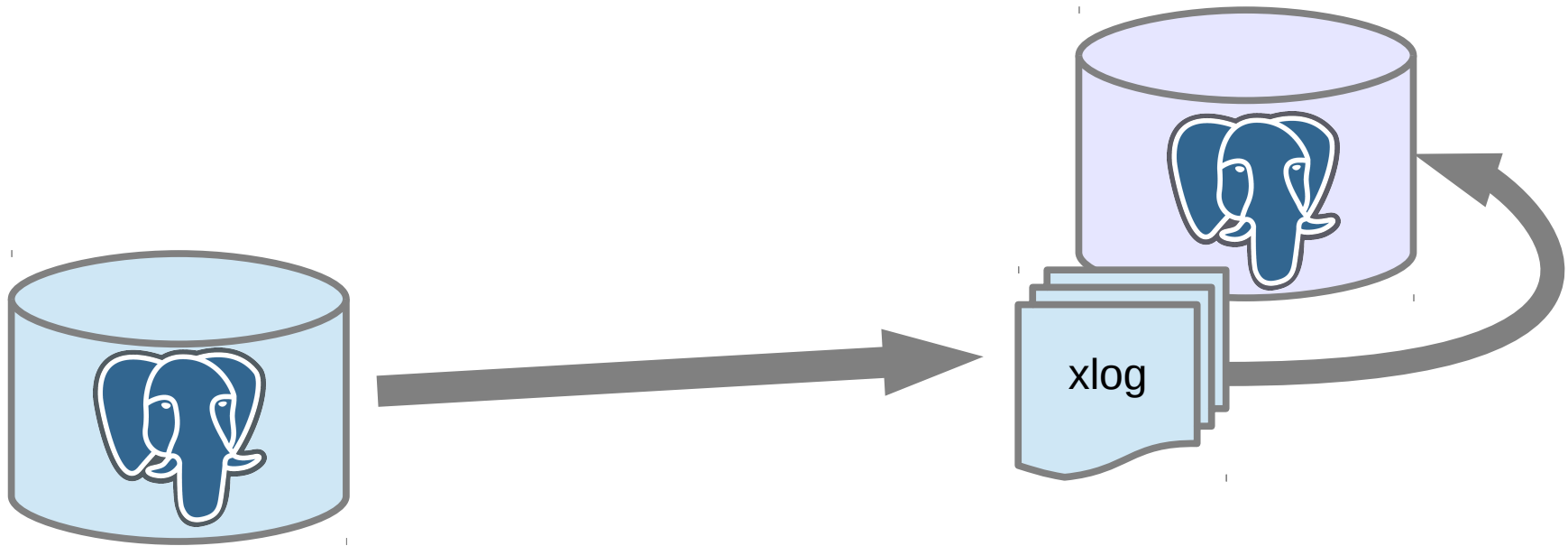
more terms

- **recovery**
 - binary replication came from binary backup, i.e. Point In Time **Recovery**
- **snapshot, clone**
 - taking a moment-in-time copy of a running database server
- **standalone**
 - a lone read-write server, neither master nor replica

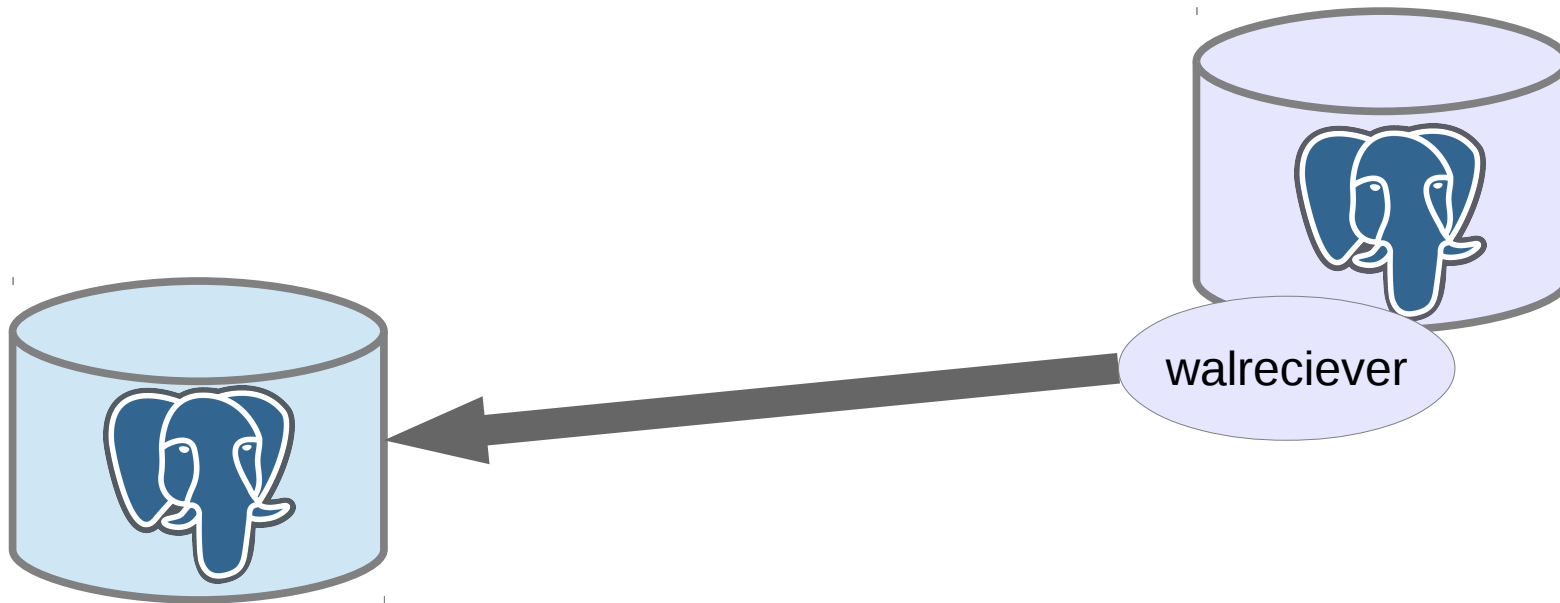
how it works



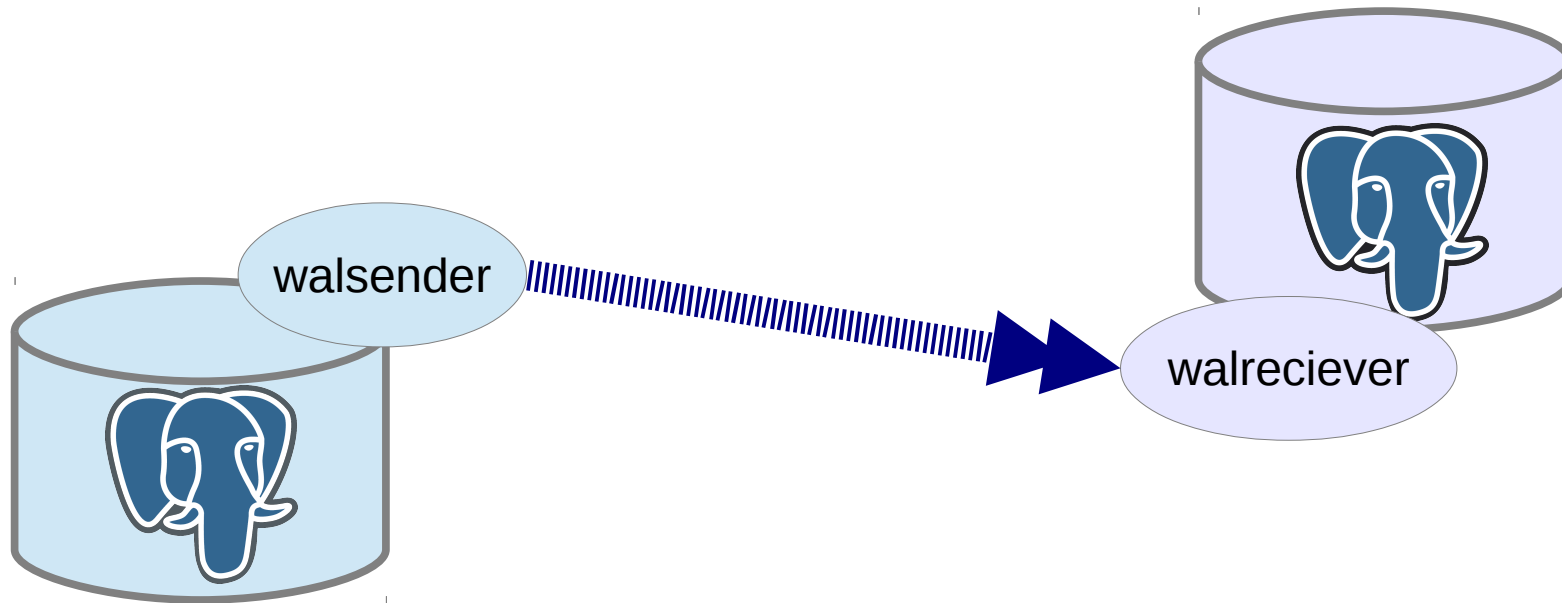
how it works



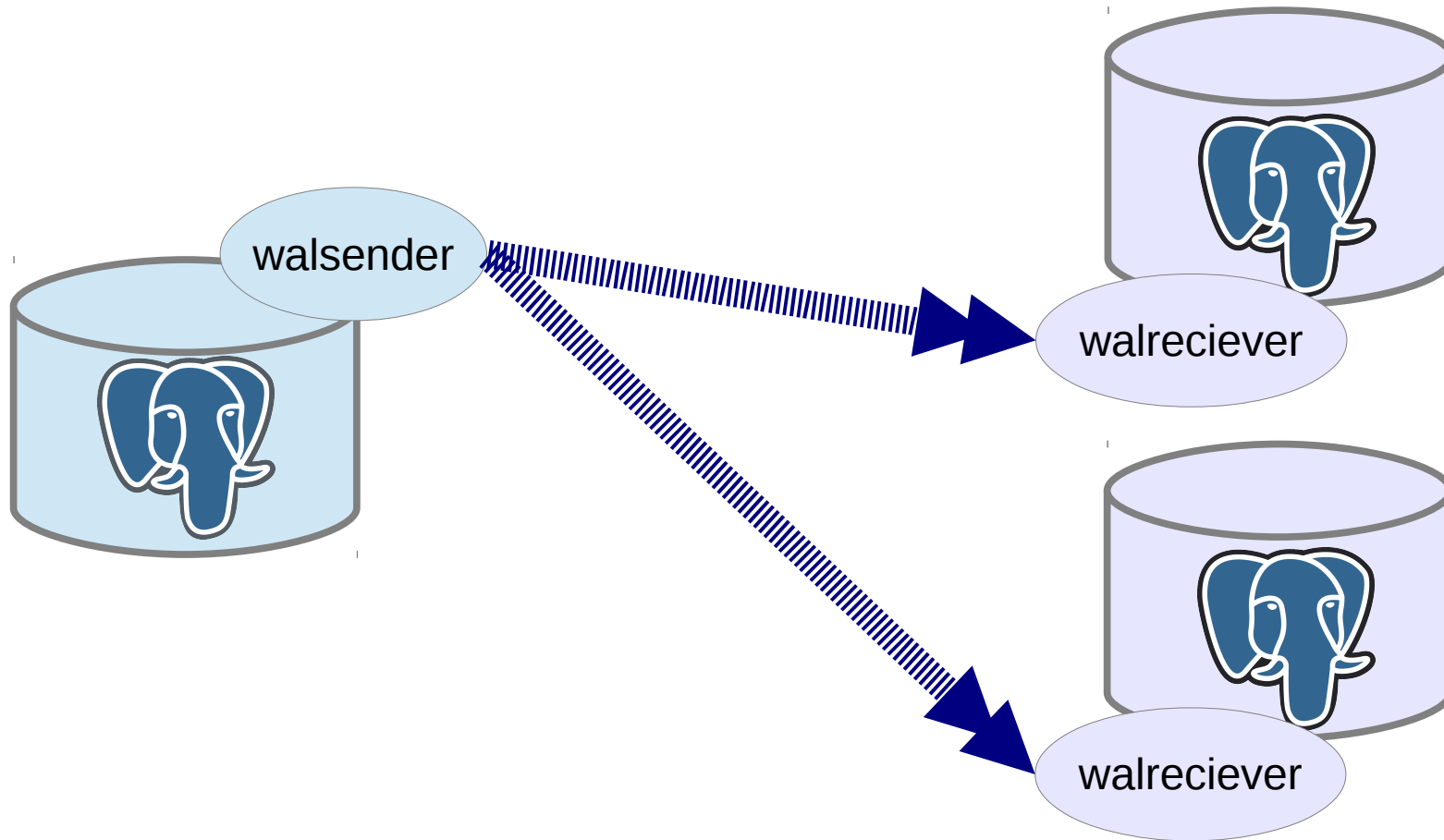
how it works



how it works

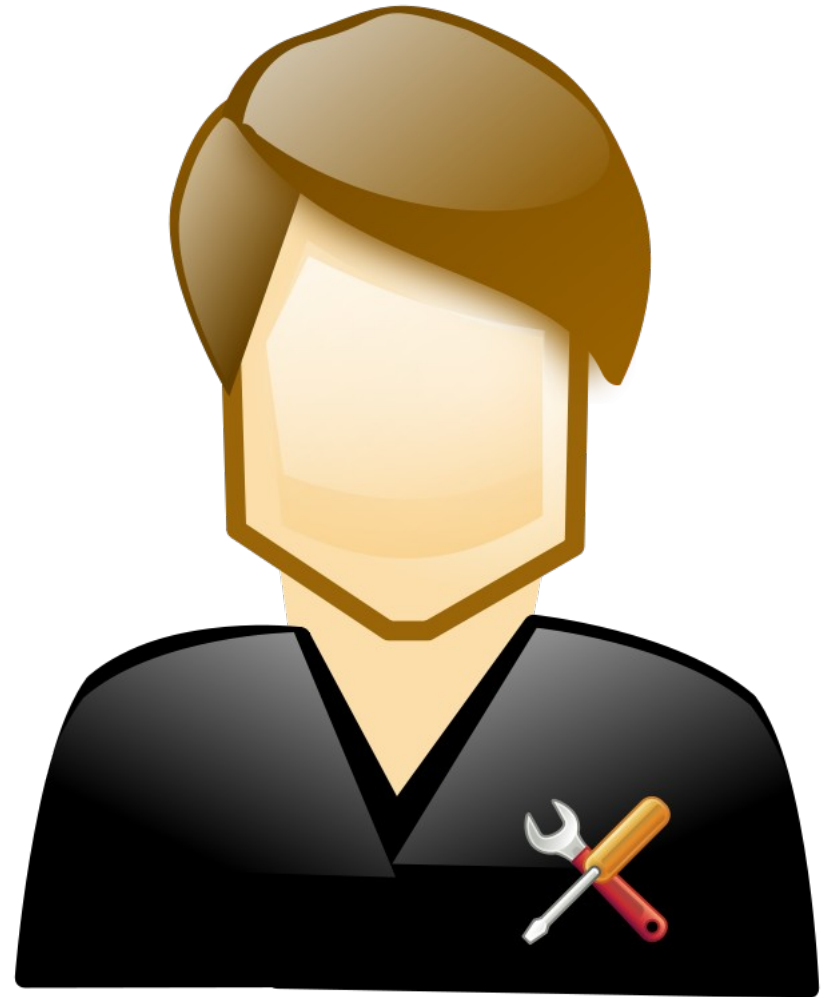


how it works



**streaming
async replication
exercise**

**administering
replication**



configuration files

- postgresql.conf
 - same settings for master, replica
- recovery.conf
 - presence turns on replication
 - must be in \$PGDATA

views & functions

- process list
- pg_stat_replication
- pg_is_in_recovery()
- pg_xlog* functions

administration exercise

permissions & security

- A. replication permission
- B. pg_hba.conf
- C. max_wal_senders
- D. firewall/network

security exercise

replicating extensions

1. install package/libraries master
2. install package/libraries on each replica
3. install extension into database

replicating PostGIS

1. install PostGIS libraries on new replica
2. clone to new replica
3. start replication

upgrading PostGIS

1. upgrade PostGIS libraries on master
2. upgrade PostGIS libraries on replicas
3. run `ALTER EXTENSION UPDATE` on master

replication & upgrades

1. declare downtime
2. stop replication
3. upgrade a replica
4. run tests
5. failover
6. upgrade the master

unreplicated stuff

- unlogged tables
- temporary tables
- LISTEN/NOTIFY
 - (might get fixed)



cloning



cloning requirements

copy a point-in-time snapshot

or

copy all database files, plus all
transaction logs between
beginning and end of copy

downtime cloning

1. shut down PostgreSQL
2. copy all files
3. bring up master
4. bring up replica

FS snapshots

1. use ZFS, LVM or SAN
2. take point-in-time snapshot
3. mount snapshot on replica
4. bring up replica

pg_basebackup

- command-line tool for cloning
- copies over \$PGPORT
 - no ssh needed
- also copies required logs
- requires streaming replication
- no compression, incremental

pg_basebackup stream

pg_basebackup -X stream

- prevents clone from falling behind
- requires a 2nd connection



archiving replication



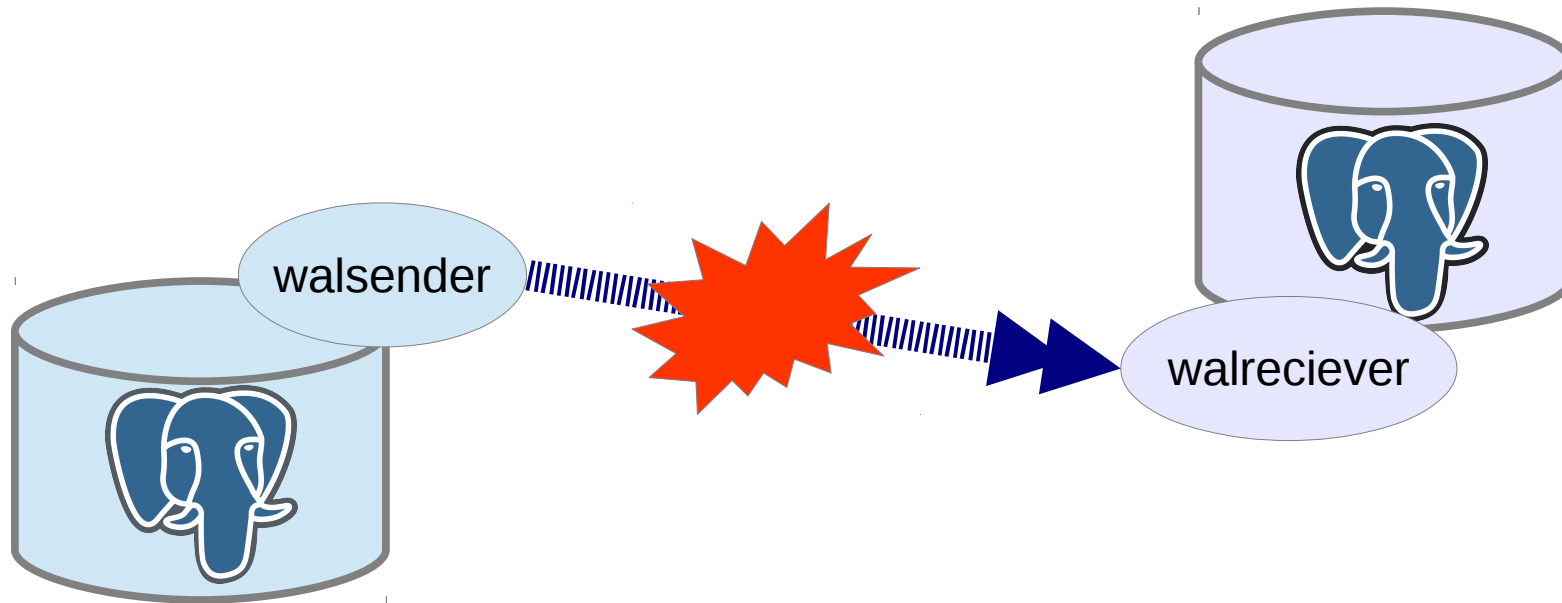
archiving replication

1. set up archiving
2. start archiving
3. `pg_start_backup('label')`
4. rsync all files
5. `pg_stop_backup()`
6. bring up replica

reasons to archive

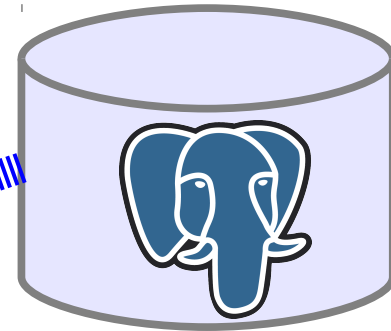
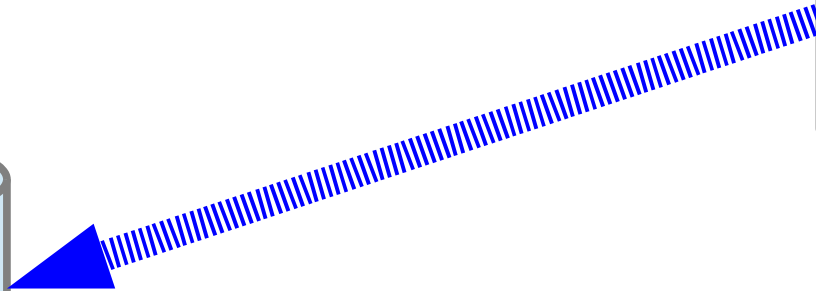
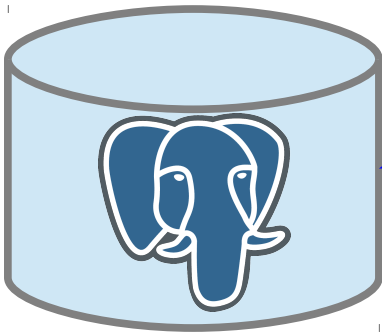
- replica out-of-sync
- combine with PITR or DR
- very erratic connection to master
- need remastering before 9.3

falling behind



falling behind

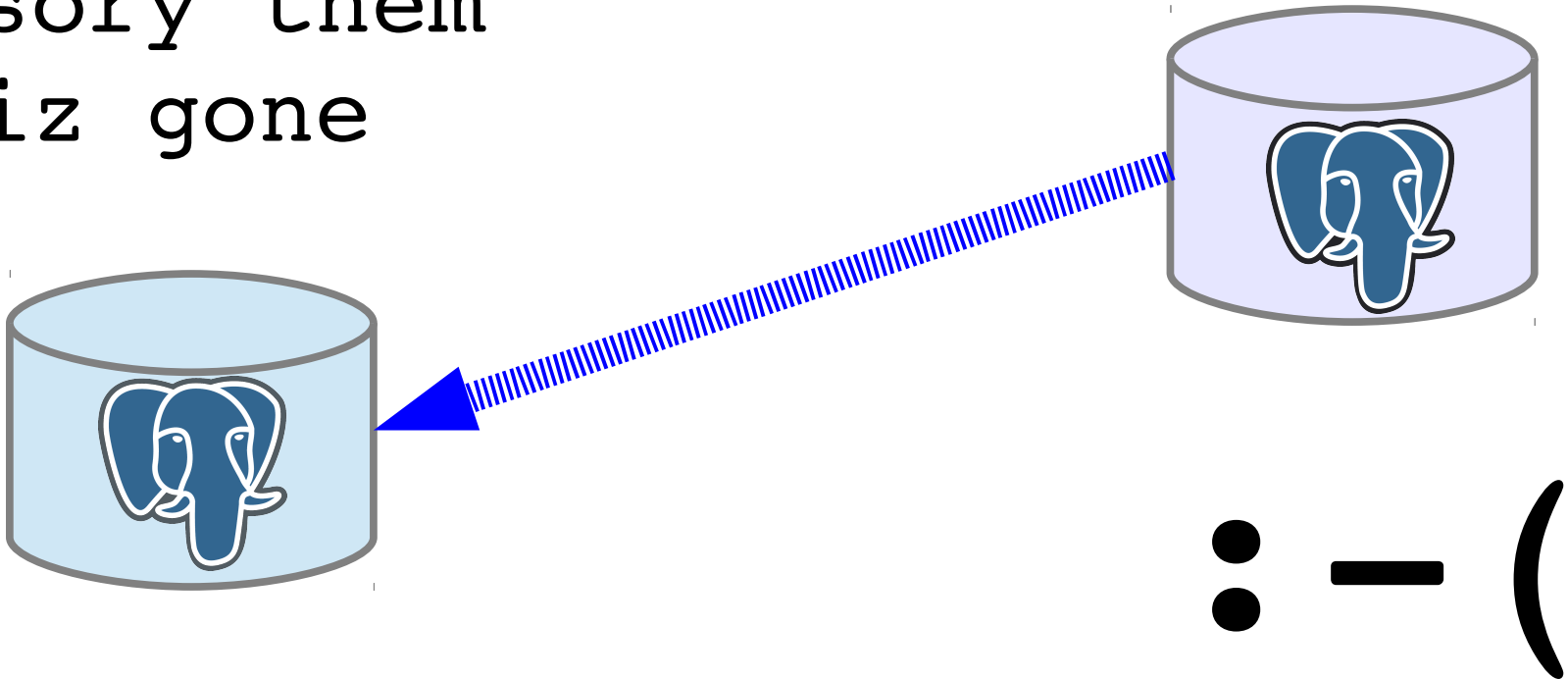
sorry them
iz gone



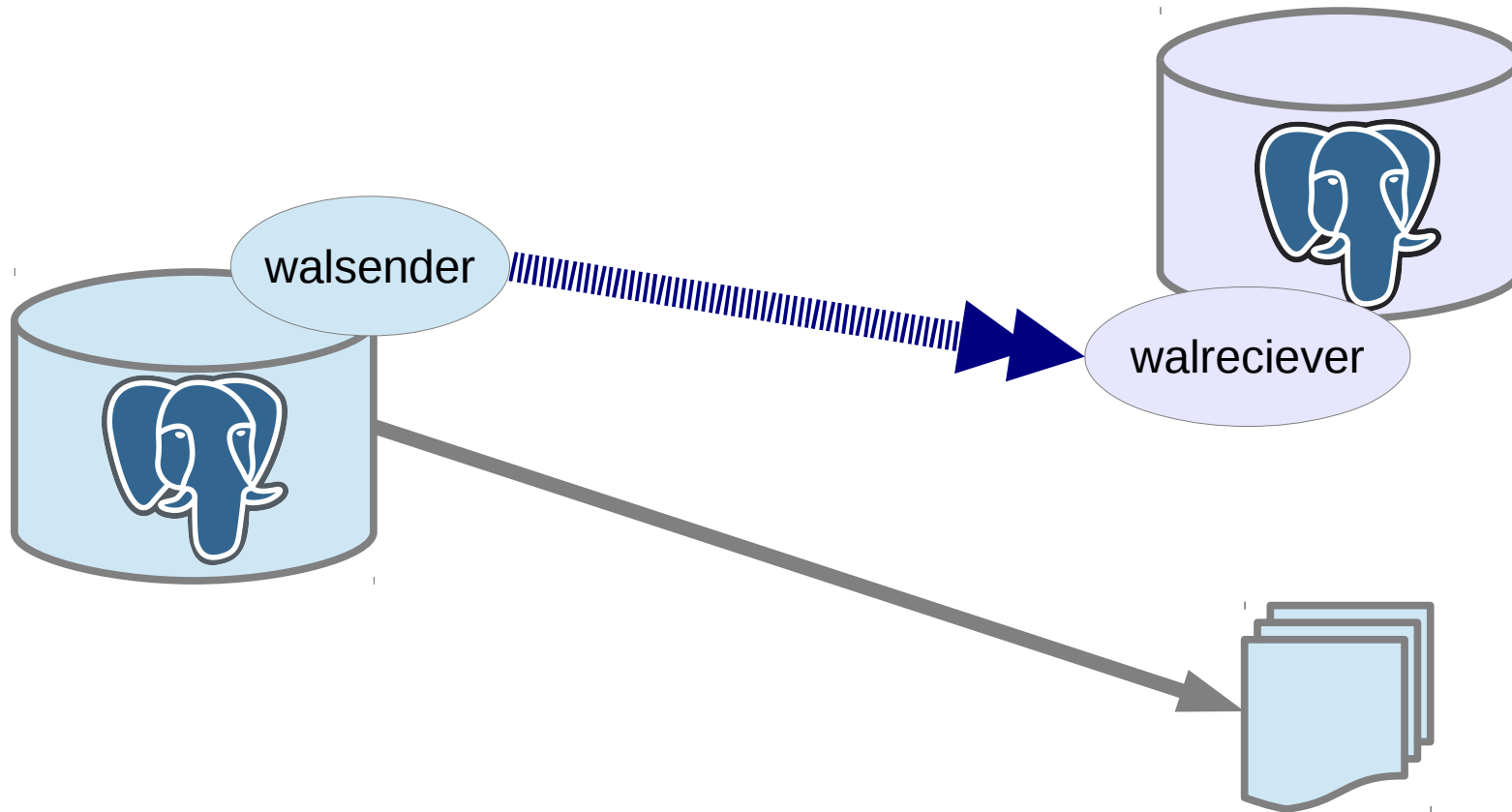
can I haz
some old
logs plz?

falling behind

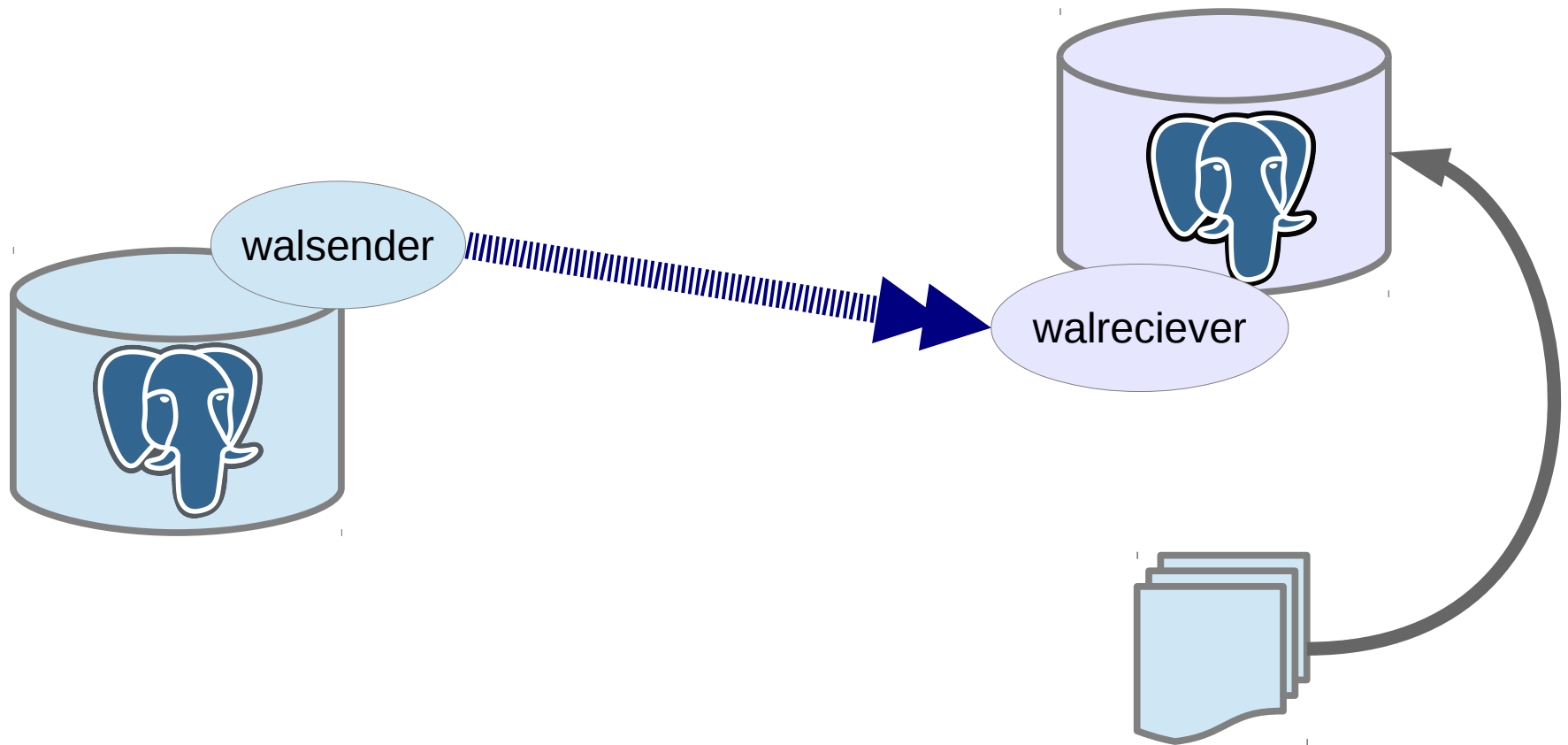
sorry them
iz gone



falling behind



falling behind



other catchup options

- wal_keep_segments
 - keeps extra xlog on the master
 - creates master load
- replication slots
 - will cover later in this tutorial

archiving hands-on

archiving tips

- use a script which handles copy failure
- use a shared drive
- put archive on a partition
- monitor for archive growth
- compression

failover, failback & remastering



more terms

- **failover, promotion**
 - making a replica into a master/standalone
- **failback**
 - returning the original master to master status
- **remastering**
 - designating a new master in a group of servers

replica promotion

- pg_ctl promote
- trigger file
- rm recovery.conf & restart

**the trouble
with fallback:
timelines, sync
& split-brain**



**hands-on
failover & failback**

failover has 3 parts

1. failing over the database
2. failing over the connections
3. STONITH

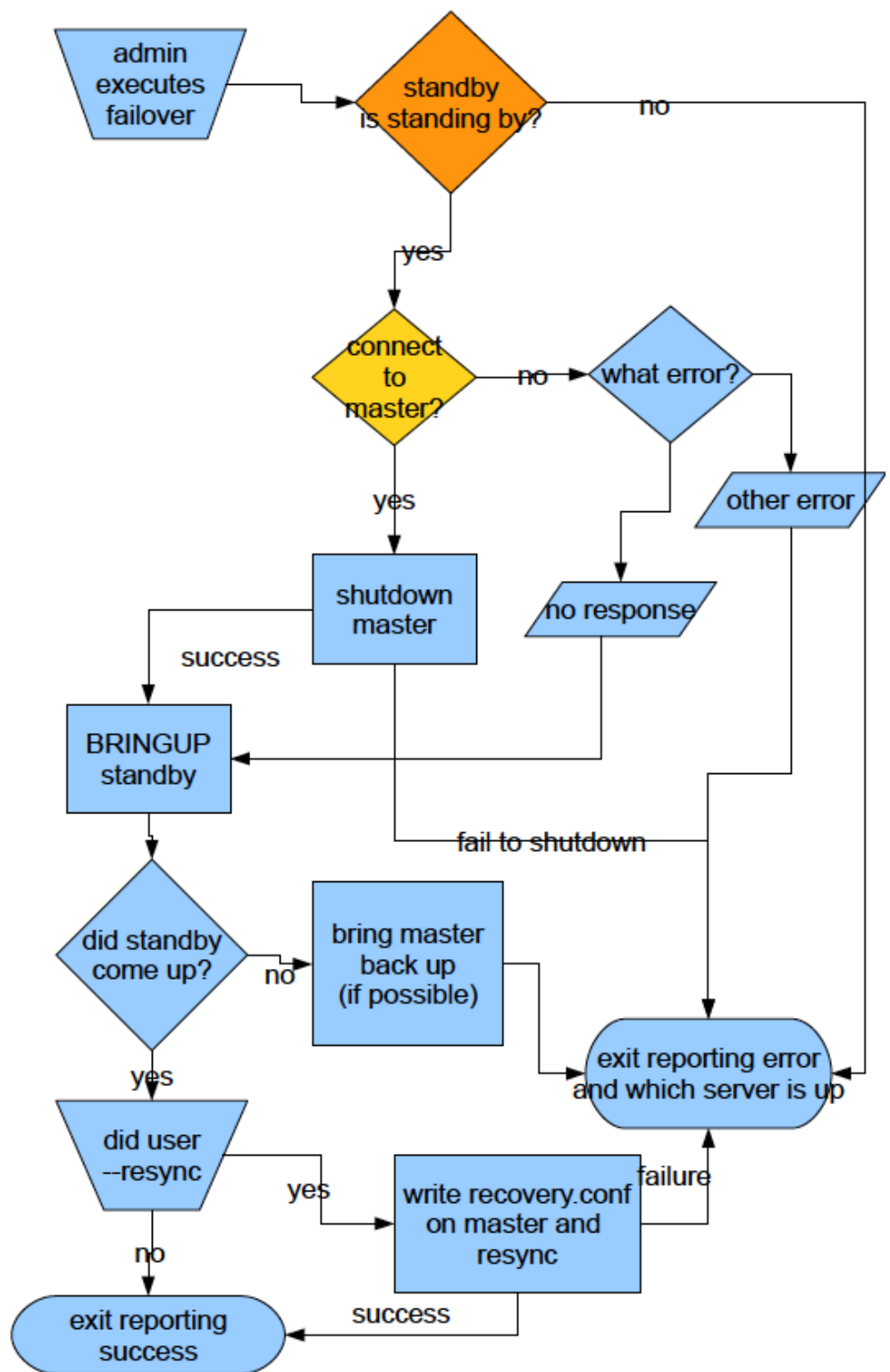
manual failover

- advantages:
 - easy to set up
 - fewer accidental failovers
- disadvantages:
 - downtime
 - being woken up at 3am

automated failover

- advantages:
 - low downtime
 - sleep through the night
- disadvantages:
 - hard to set up correctly
 - need broker server
 - accidentally triggered failovers

automated failover logic



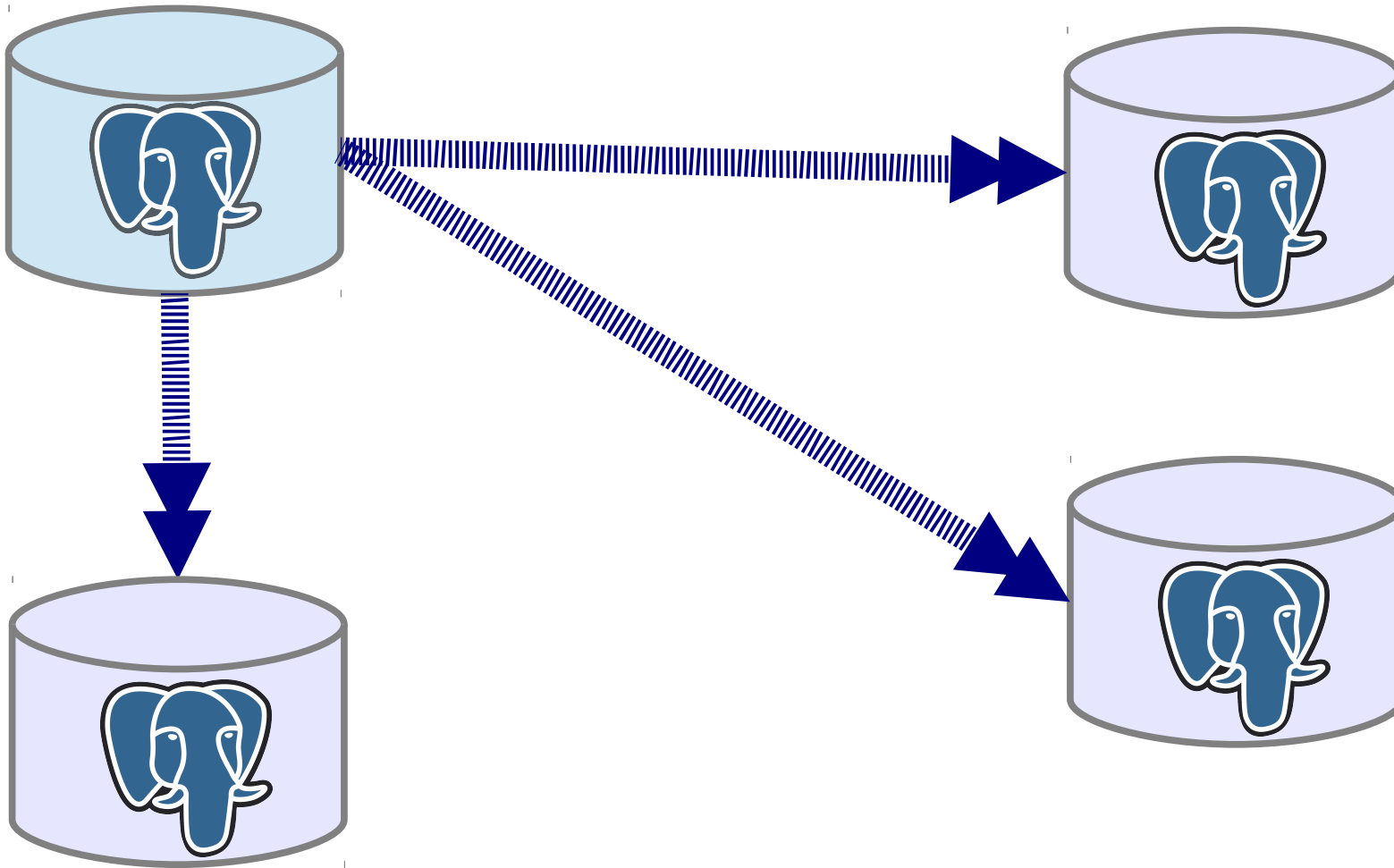
STONITH

- use corosync/VIP
- use connection failover
- use peer broker server

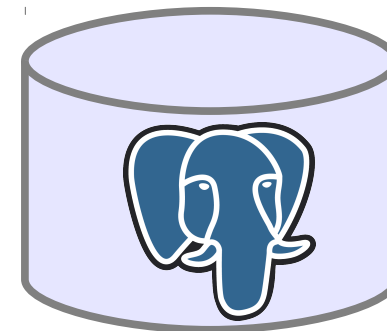
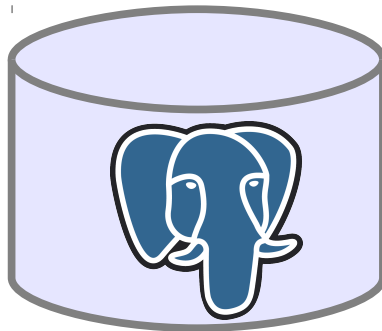
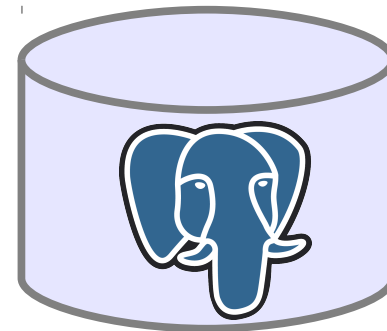
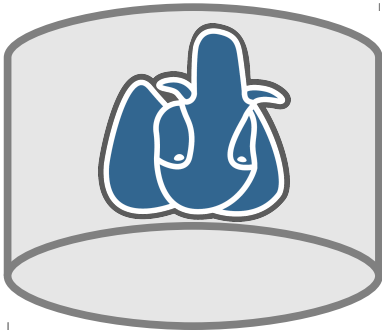


www.handyrep.org

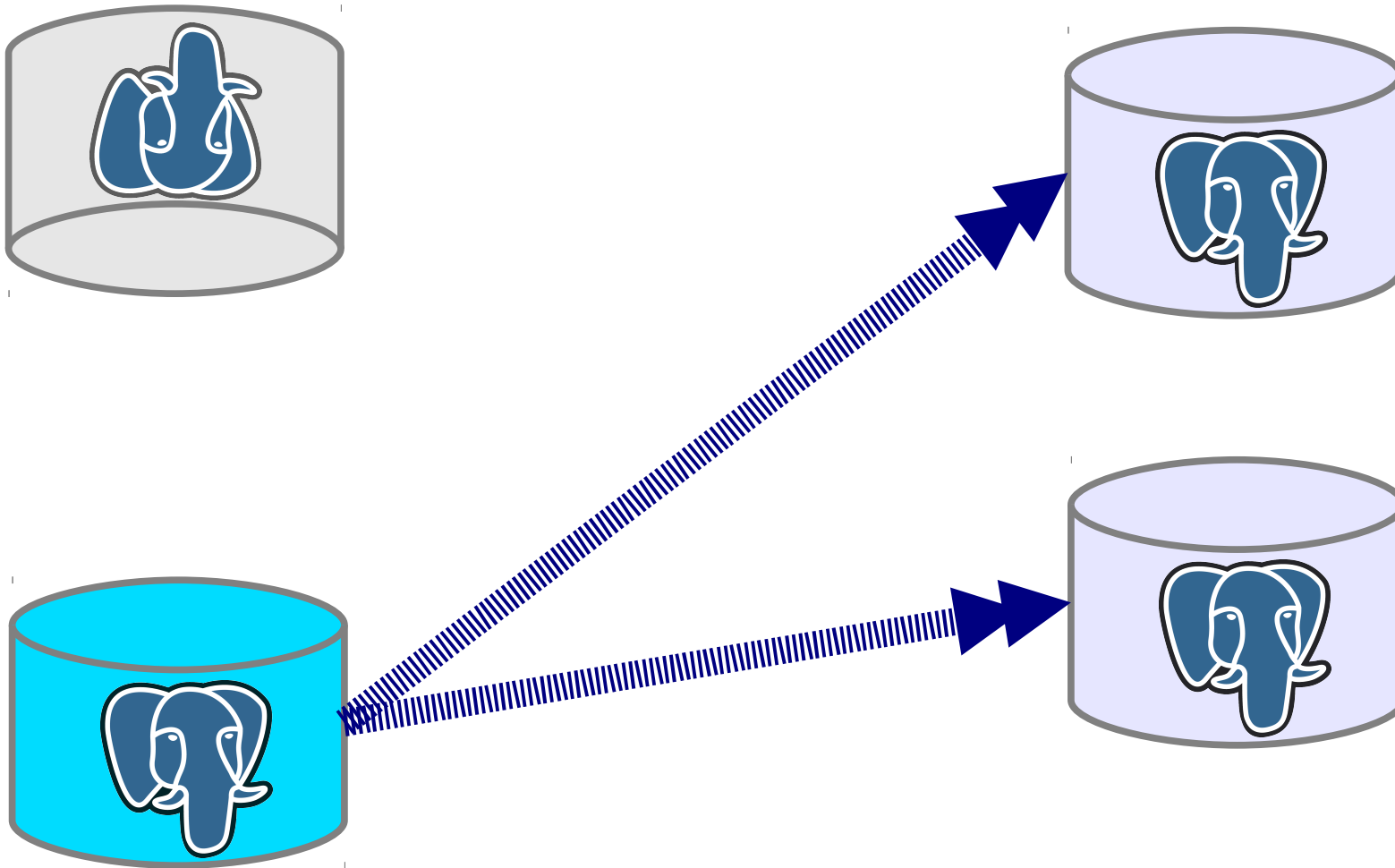
Remastering 9.3+



Remastering 9.3+



Remastering 9.3+



remastering

- need the replica which is “furthest ahead”
- measure both receive point and replay point
- need 9.3 for “streaming-only” remastering

replay lag

```
SELECT  
pg_xlog_location_diff(  
    pg_last_xlog_receive_location(),  
    pg_last_xlog_replay_location()  
);
```

4294967296

replication lag & query cancel



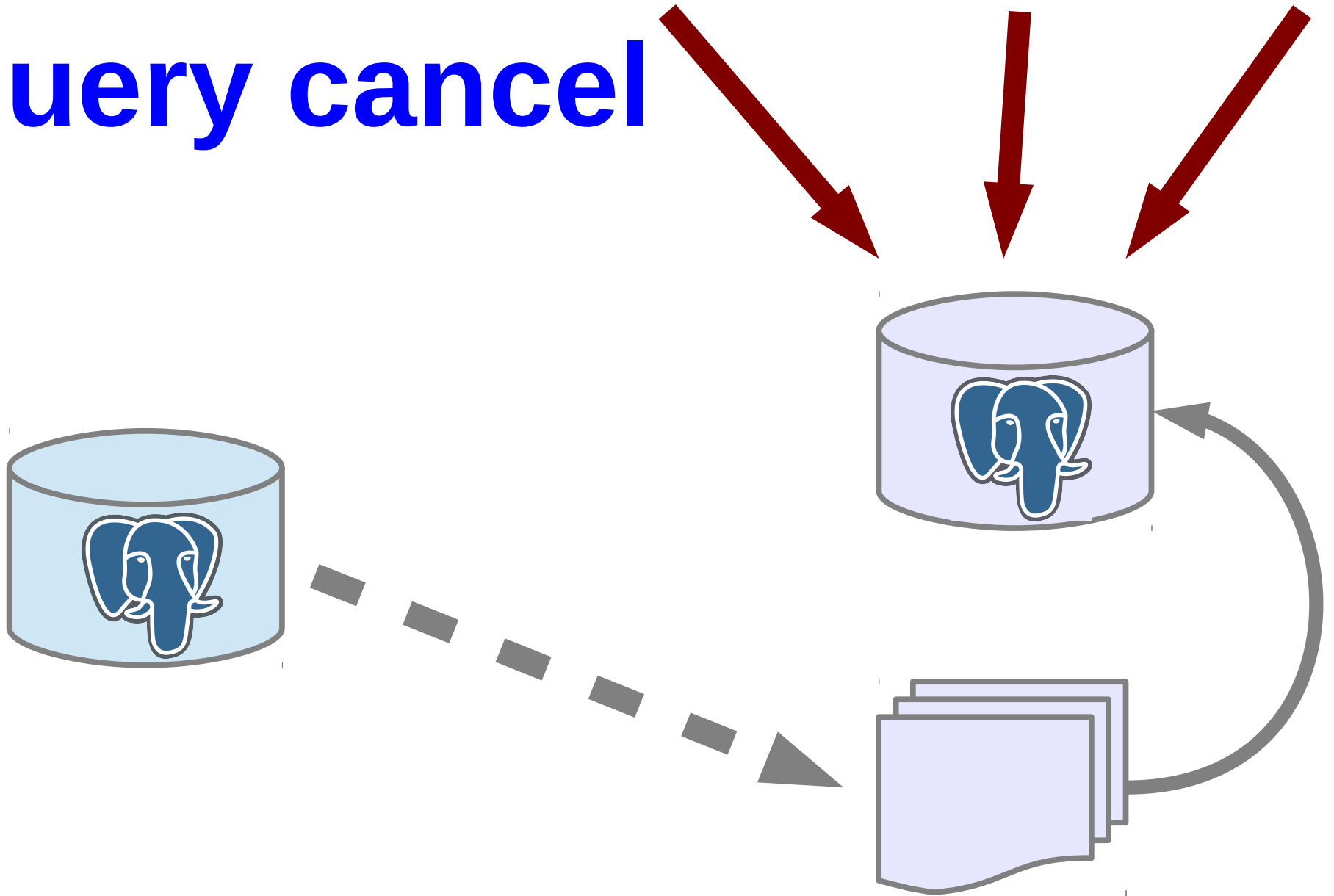
reasons for lag

- network delay
 - speed of light
- replica too busy
- file operations block
 - VACUUM
 - DROP TABLE

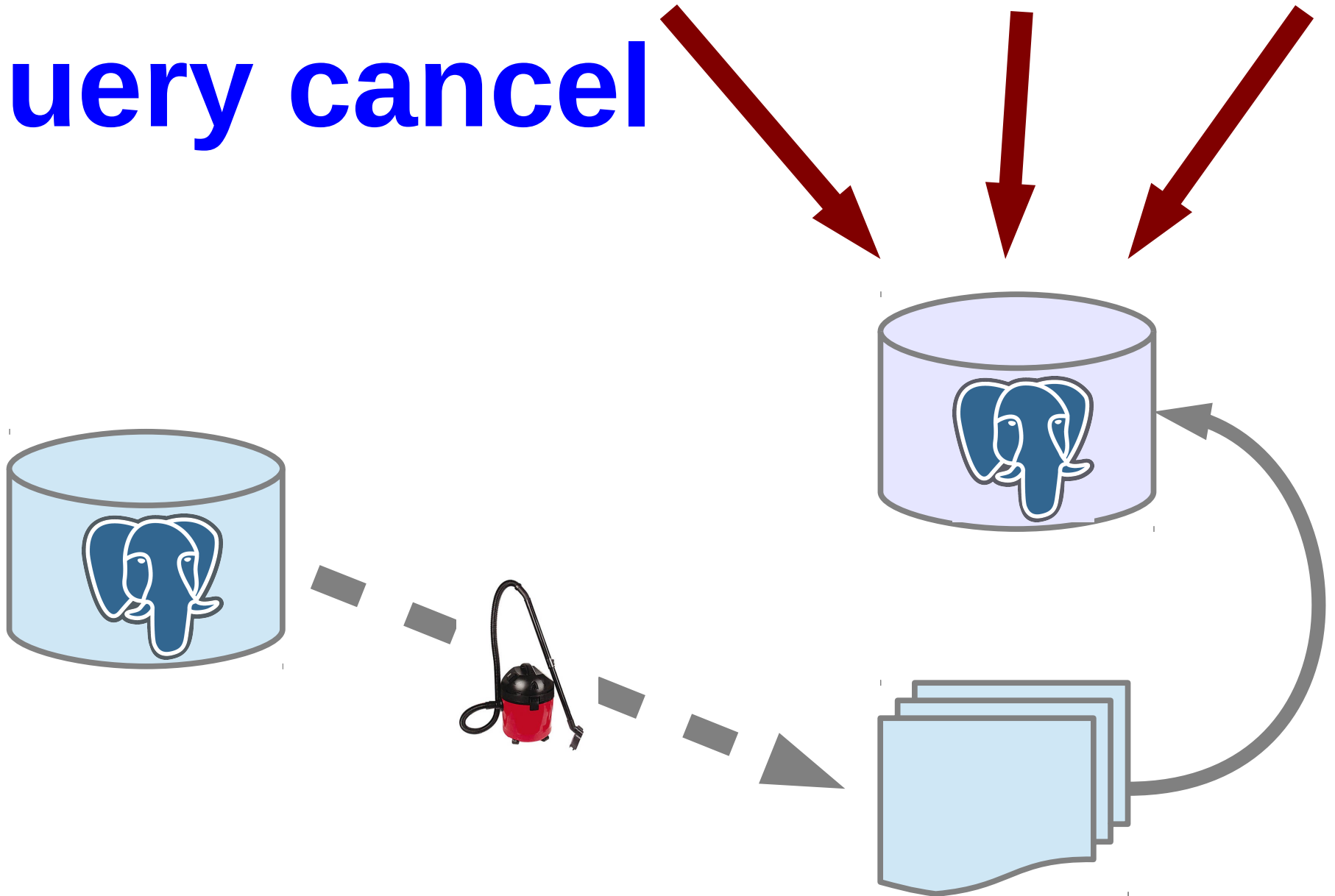
replication lag issues

- inconsistency (if load-balancing)
- query cancel
 - applications need to retry queries
- catch-up speed
- burden on master

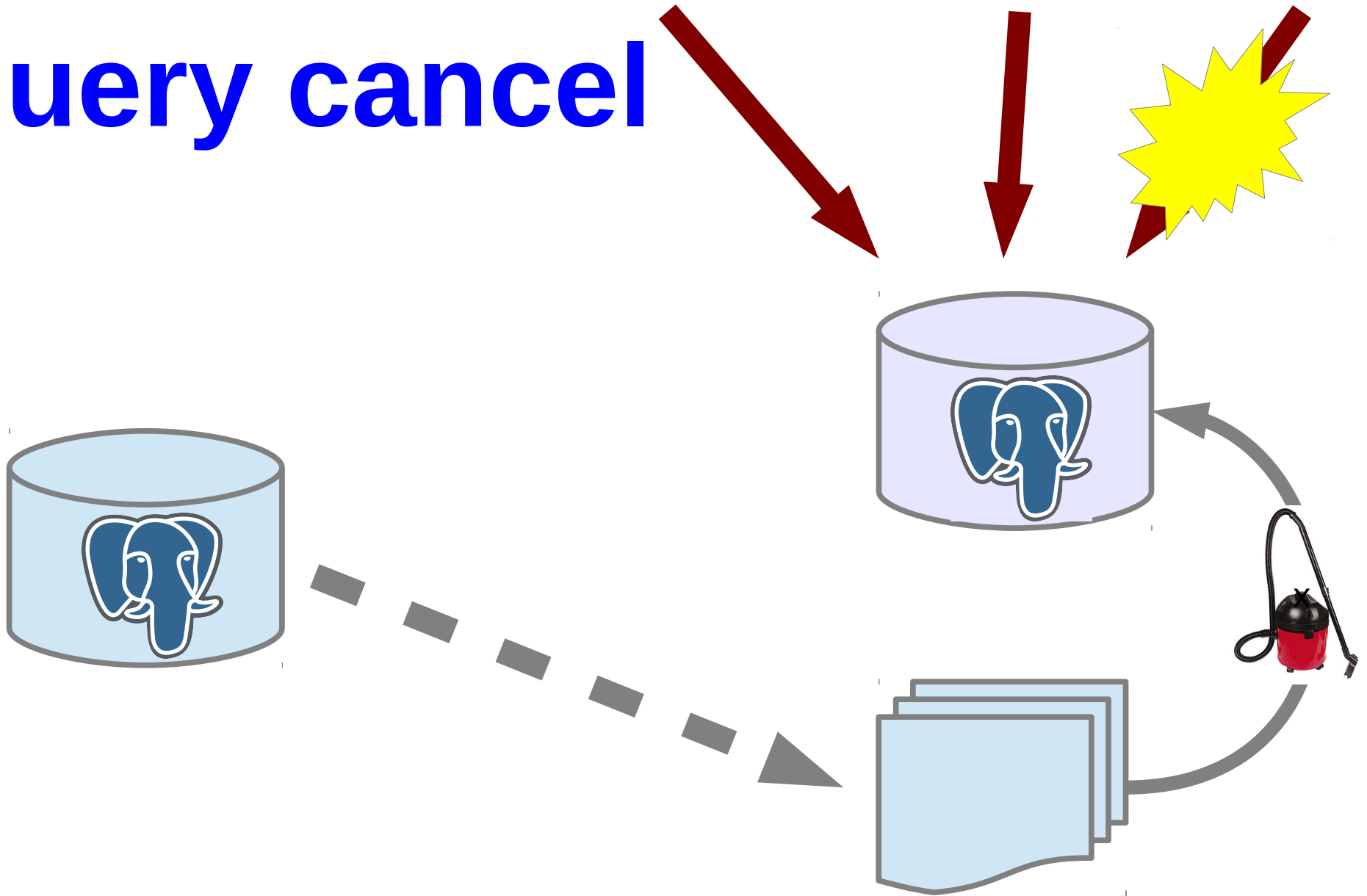
query cancel



query cancel

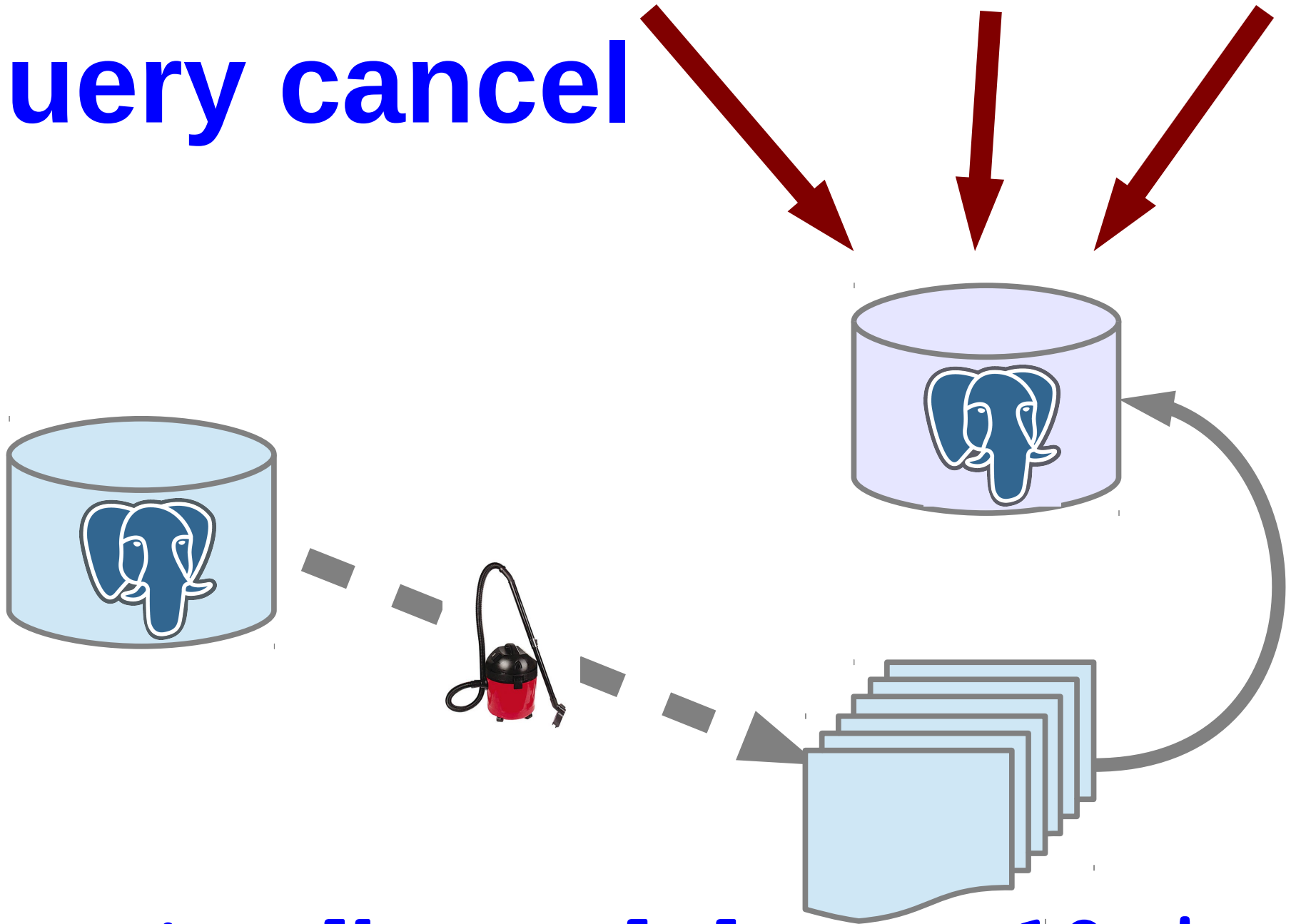


query cancel



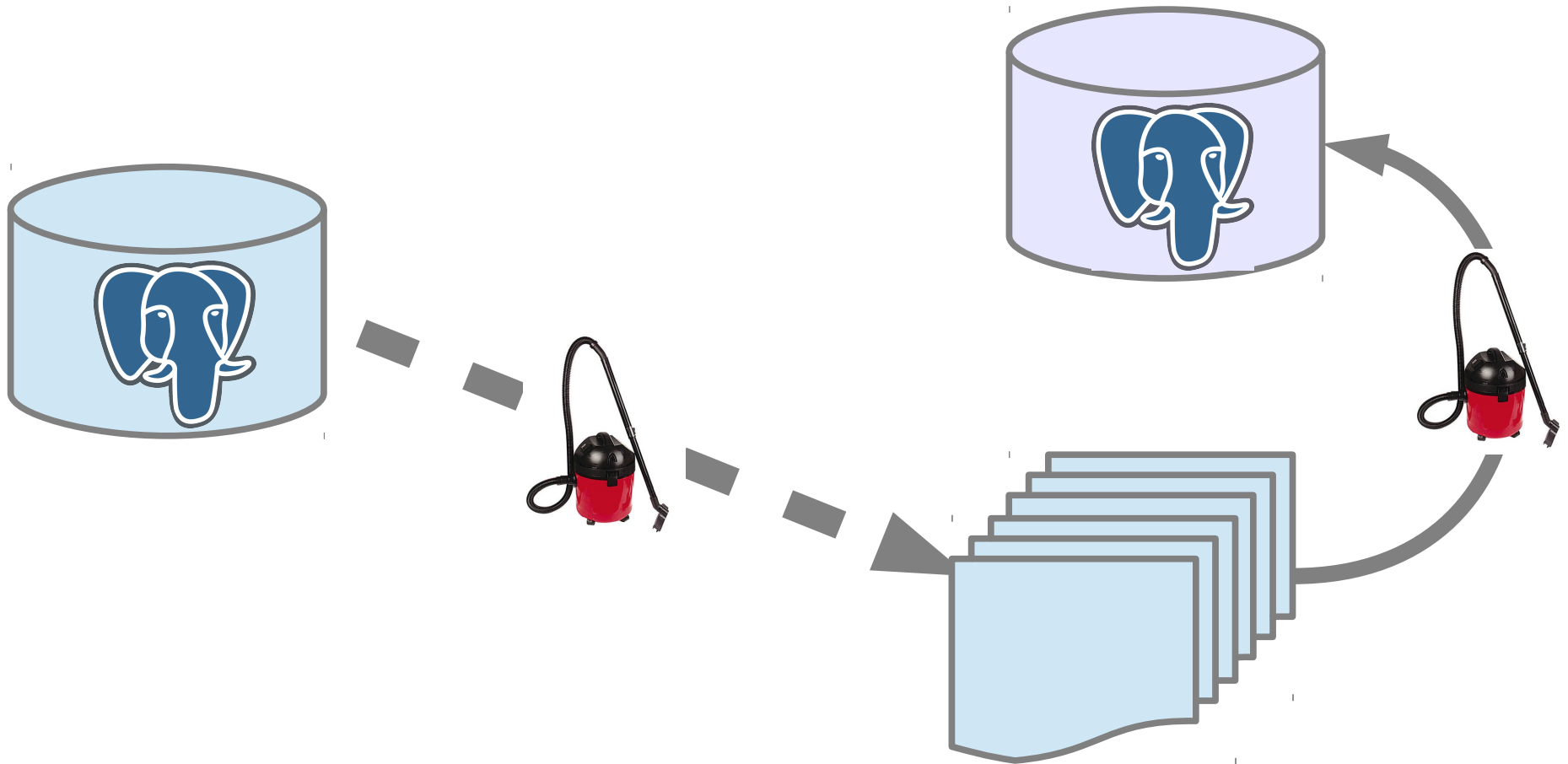
ERROR: canceling
statement due to conflict
with recovery
DETAIL: User query might
have needed to see row
versions that must be
removed.

query cancel



$\text{max_standby_}_\text{delay} = 10\text{min}$

max_standby_delay



lag vs. cancel

- increase *_delay → fewer cancels, more lag
- decrease *_delay → less lag, more cancels

lag vs. cancel

- hot failover server: low *_delay
- reporting server: high *_delay
- load-balancing:
moderate *_delay

cancel-causing

- DROP database;
- DROP table/index;
- REINDEX;
- VACUUM;

other ways

- vacuum_defer_cleanup_age
 - delays *all* vacuuming
 - set to ~~ 1000
- hot_standby_feedback
 - feedback from standbys to master on what's visible

**configuring lag
and cancel
hands-on**

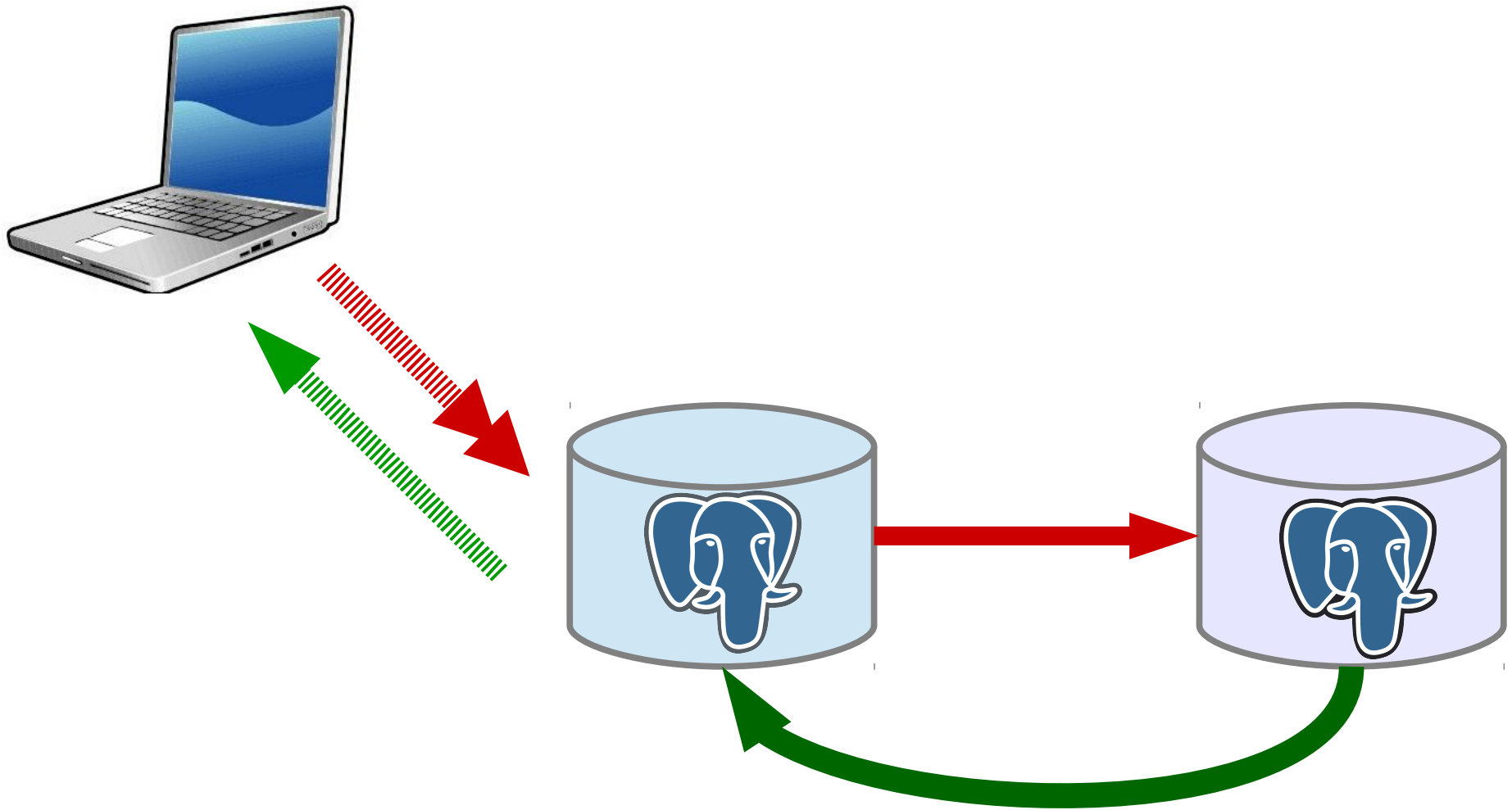


**synchronous
replication**

what synch rep does

guarantee against data loss

how it works



what it doesn't

- enforce global consistency
 - master can be behind
 - replica snapshot can be behind
- help availability

**“I would rather be down
than potentially lose
data.”**

how to synch rep

1. pick one (or a pool) of servers to be your synch replicas
2. change application_name
3. change master's postgresql.conf
4. reload

Postgres specialities

- implements only 1-redundant model
- synch is *per-transaction*
 - not per replica
 - synch only important transactions

synchronous_commit

setting	disk	replica memory	replica disk
off	no	no	no
local	yes	no	no
remote_write	yes	yes	no
on	yes	yes	yes


synch rep
hands-on

synch rep design

- 1 replica is synch replica
- several asynch replicas
- load-balance to asynch only
- always failover to synch replica

synch rep monitoring

- monitor critically:
 - synch rep downtime
 - synch replication speed
- script disabling synch rep
 - if replica is down

A photograph of a large, multi-tiered waterfall cascading down a mossy rock face into a pool of water. The water is white and frothy as it falls, and the surrounding rocks are covered in green moss and vegetation. The scene is lush and natural.

cascading replication

how to cascade

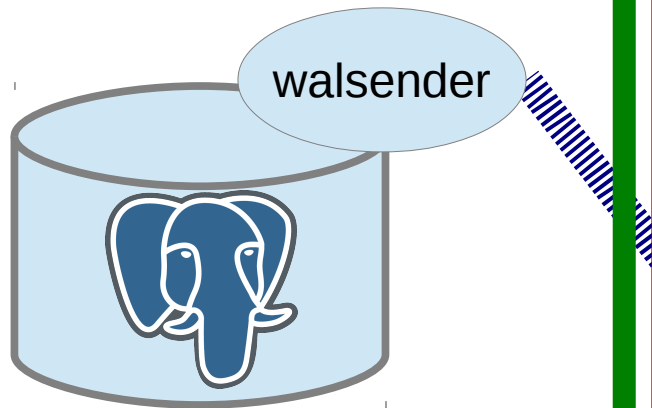
1. have master & replica
2. clone the master or the replica
3. point primary_conninfo to the replica
4. bring up the new replica

why to cascade

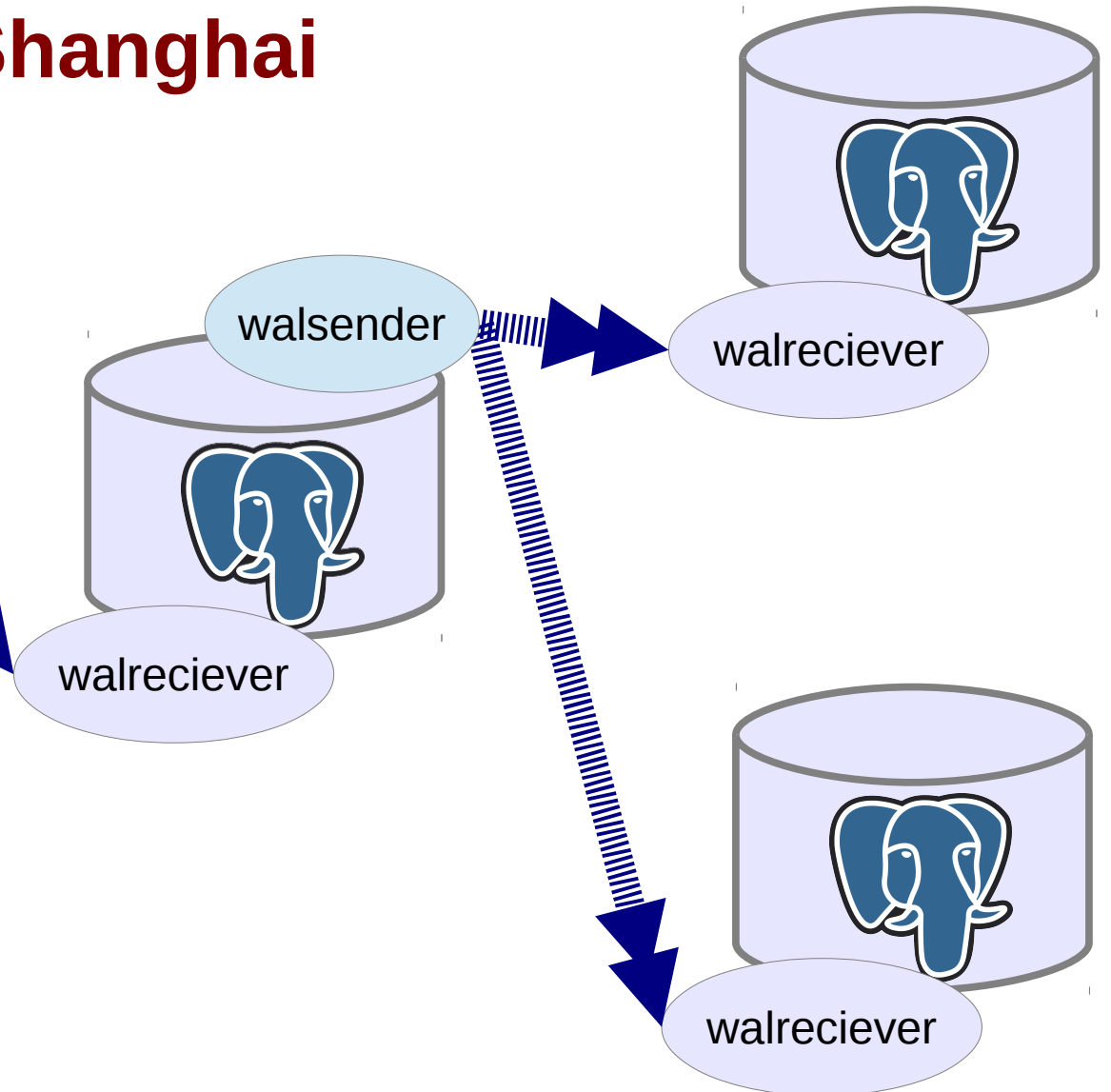
- limit connections to master
- don't clone master
- know which replica is ahead

why to cascade

Phoenix



Shanghai



why not cascade?

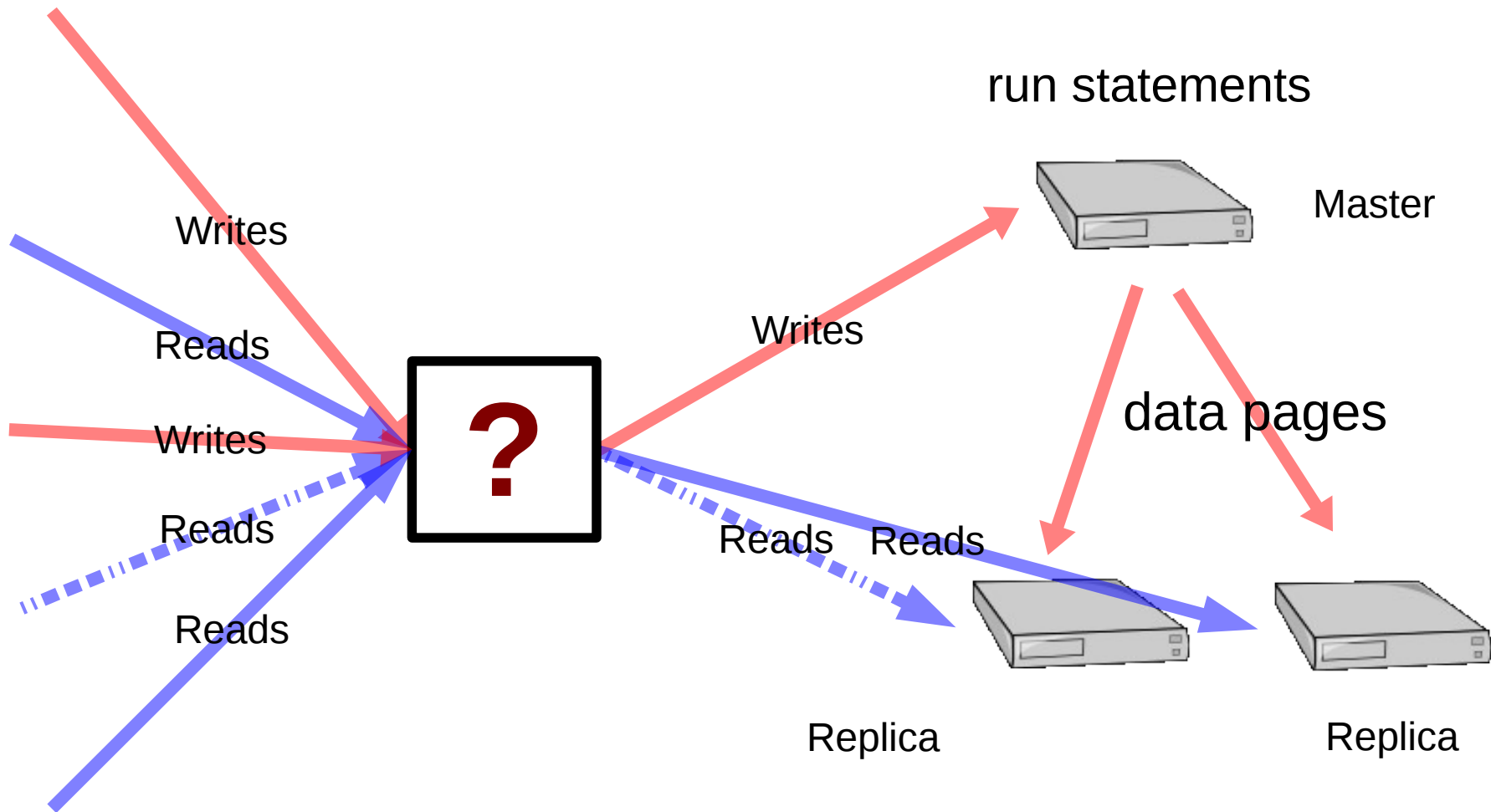
- complexity
- cycles
- increases SPOFs

**hands-on
cascading**

load balancing



load-balancing?



why load-balance?

- get some use out of the replica
- scale-out
- be ready for failover
- run special workloads
(reporting)

why not load-balance?

- complexity
- inconsistency
- limitations
- additional SPOFs
- not needed for performance

inconsistency

- lag between master & replica
- defeats read-then-write-then-read
- django: read-then-write (fortunately)
- otherwise: implement “sticky”

application LB

1. use autocommit

- django: @xact or @atomic

2. create “rw” and “ro” databases

3. route connections

- django: set up django router which directs writes & reads

network LB

1. same as application, plus:
2. set up virtual IPs
 - using Zeus, HAproxy, Cisco, etc.
3. use VIPs to load-balance read traffic
4. use VIPs to fail over
 - optional: auto-failover

pgPool2

- connects to all servers
- separates reads/writes by parsing queries
- manages failover
- not actually a pooler
 - despite name

why not pgPool2?

- complicated
 - very hard to configure correctly
 - documentation is terrible
- failover logic not great
 - and hard to change

pgBouncer

- pooler & redirector
- redirect read and write connections
- works with manual & scripted failover

pgBouncer

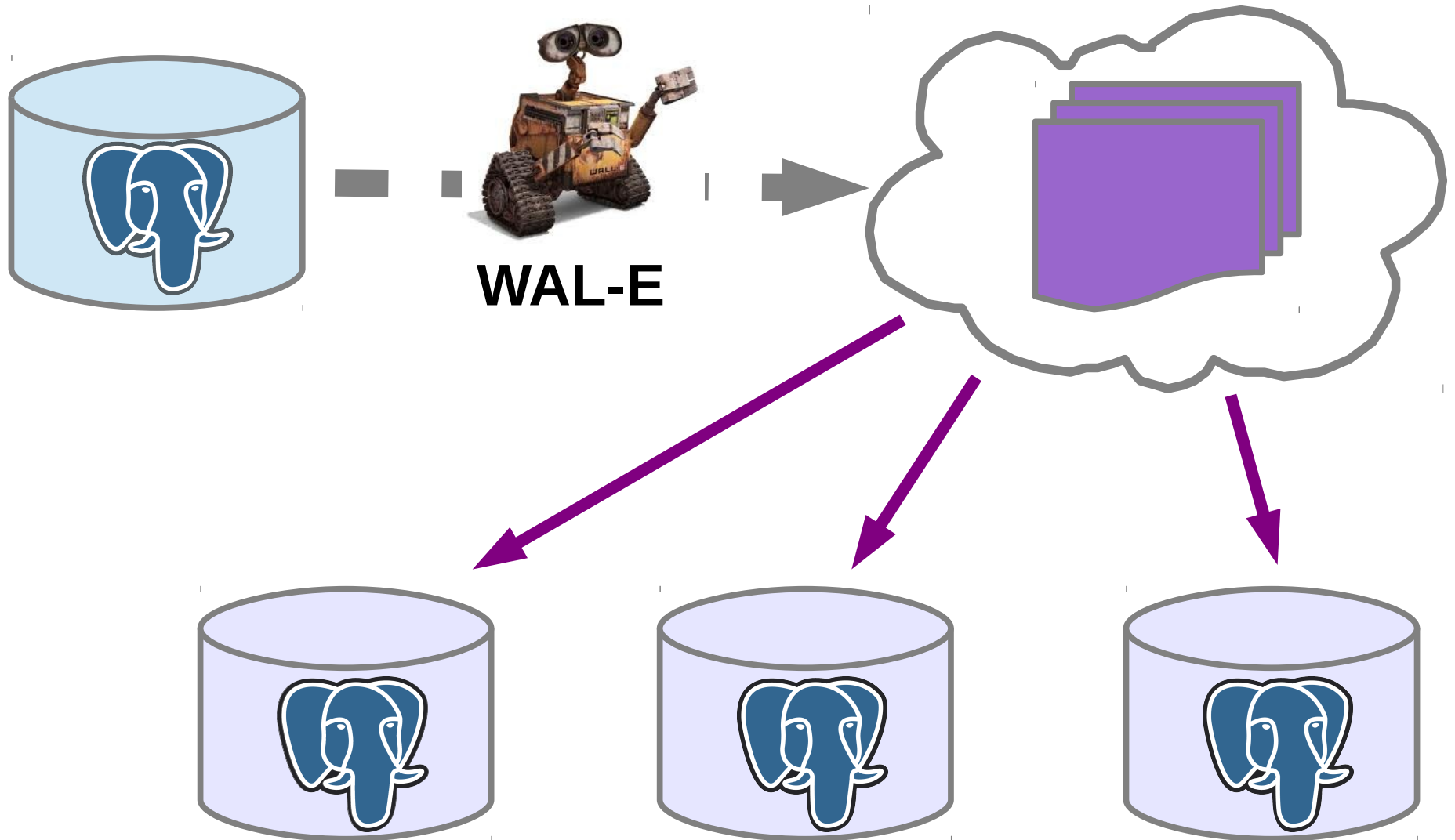
load-balancing

exercise

An aerial photograph showing a vast expanse of white, fluffy clouds against a clear blue sky. The clouds are dense and cover most of the lower two-thirds of the image. The text 'replication in the cloud' is overlaid in the center in a bold blue font.

**replication
in the
cloud**

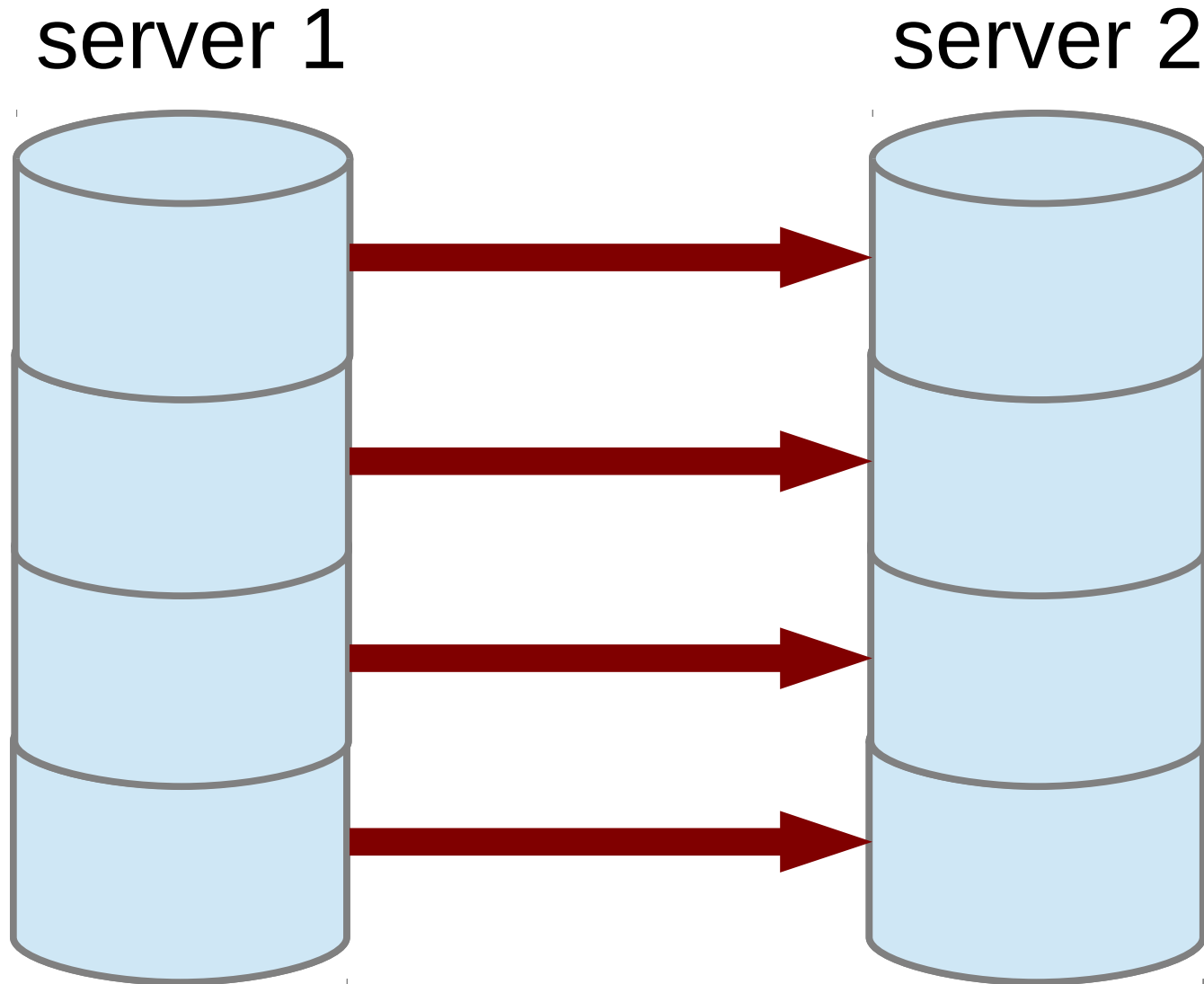
use a shared archive



ephemeral replicas

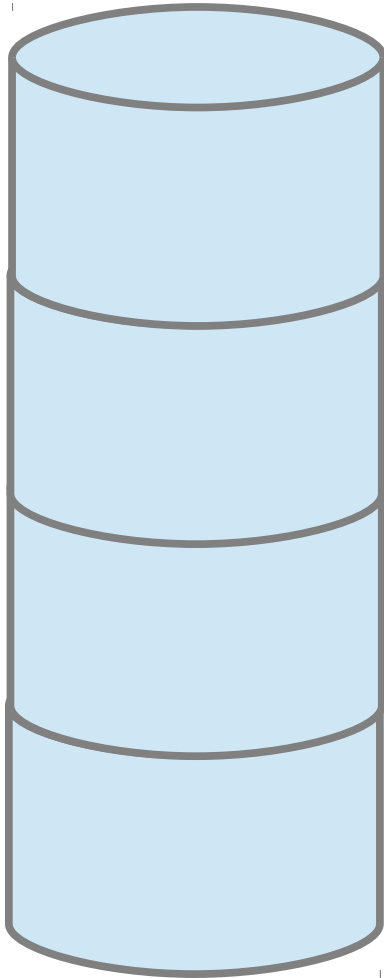
- no sync to disk
- do not recover from crash
 - spin up a replacement instead
- turn off all logging/disk
 - fsync off, bgwriter off,
full_page_writes off

sharding and replication

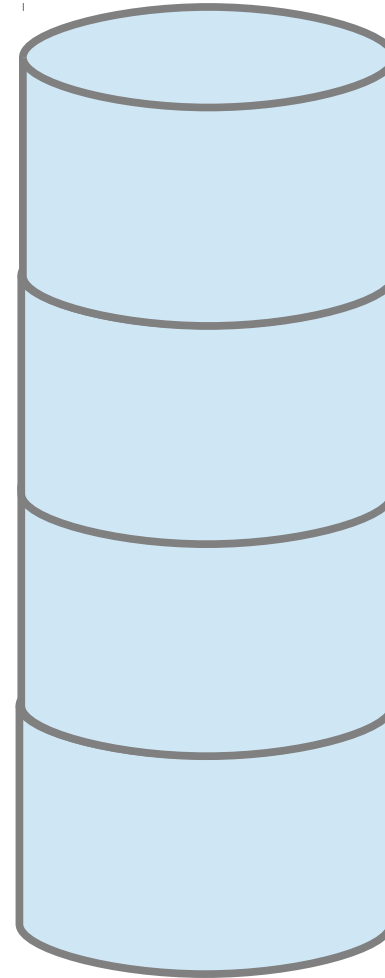


sharding and replication

server 1



server 2



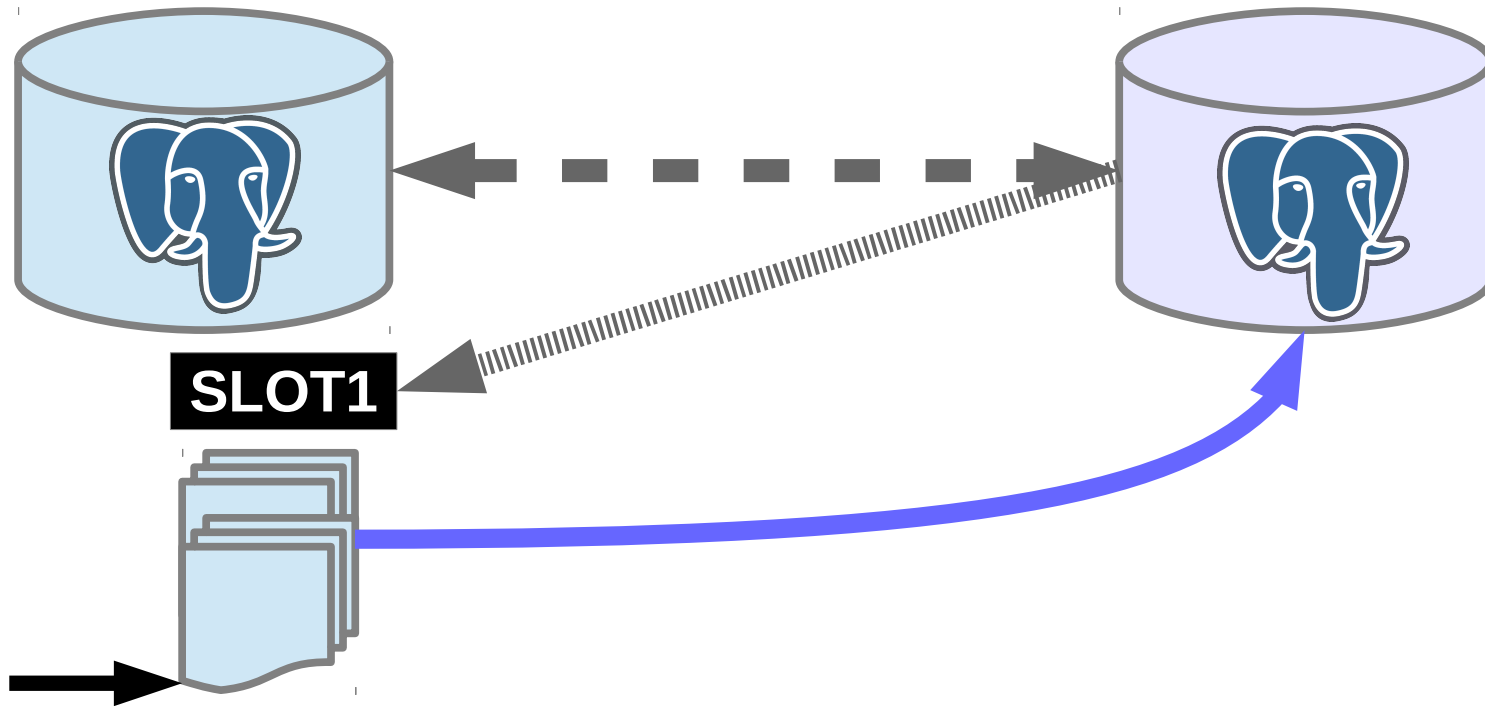
9.4 Replication



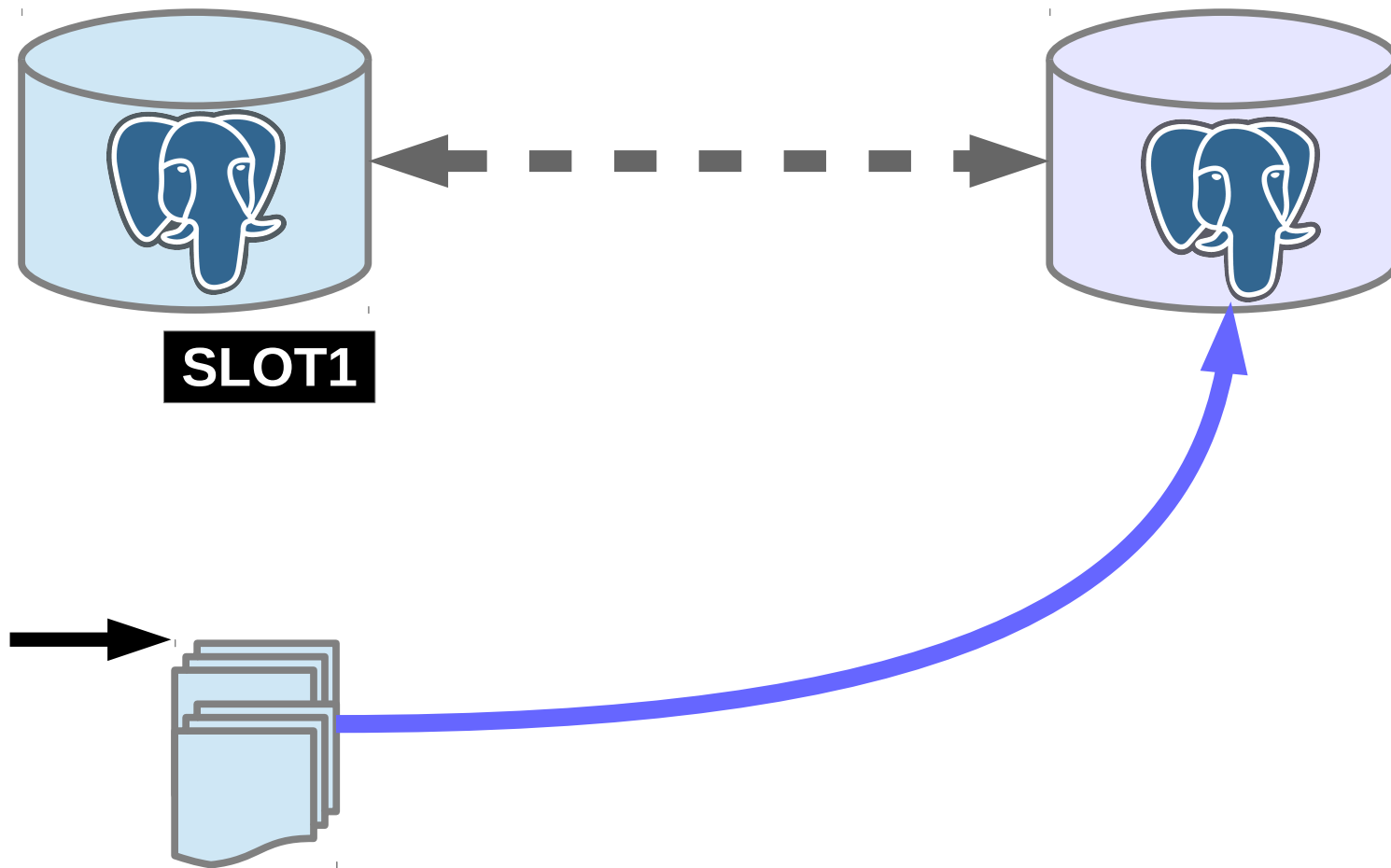
9.4: Replication Slots

- no more wal_keep_segments
- instead assign each replica a “slot”
- master will keep the logs they need
 - but monitor!

replication slots



replication slots



replication slots



9.4: Logical Decoding

- convert binary stream to row-based replication
- permits “bi-directional” rep.
 - and other custom replication
 - and cross-version replication
- will require external tools

also 9.4: delay

- `recovery_min_apply_delay`
 - delay for applying new data, in seconds
 - window for catching mistakes
 - do not use with `hot_standby_feedback`

replication slots exercise

questions?

- github.com/jberkus/pgReplicationTutorial
- **Josh Berkus:** josh@pgexperts.com
 - PGX: www.pgexperts.com
 - Blog: www.databasesoup.com
- **Upcoming Events**
 - pgDaySF @ FOSS4GNA
 - pgCon: Ottawa June 20th



Copyright 2014 PostgreSQL Experts Inc. Released under the Creative Commons Share-Alike 3.0 License. All images are the property of their respective owners. The WAL-E image is the property of Disney Inc. and is use here as parody.

