PostgreSQL replication

a hands-on tutorial

PGX POSTGRESQL EXPERTS, INC.

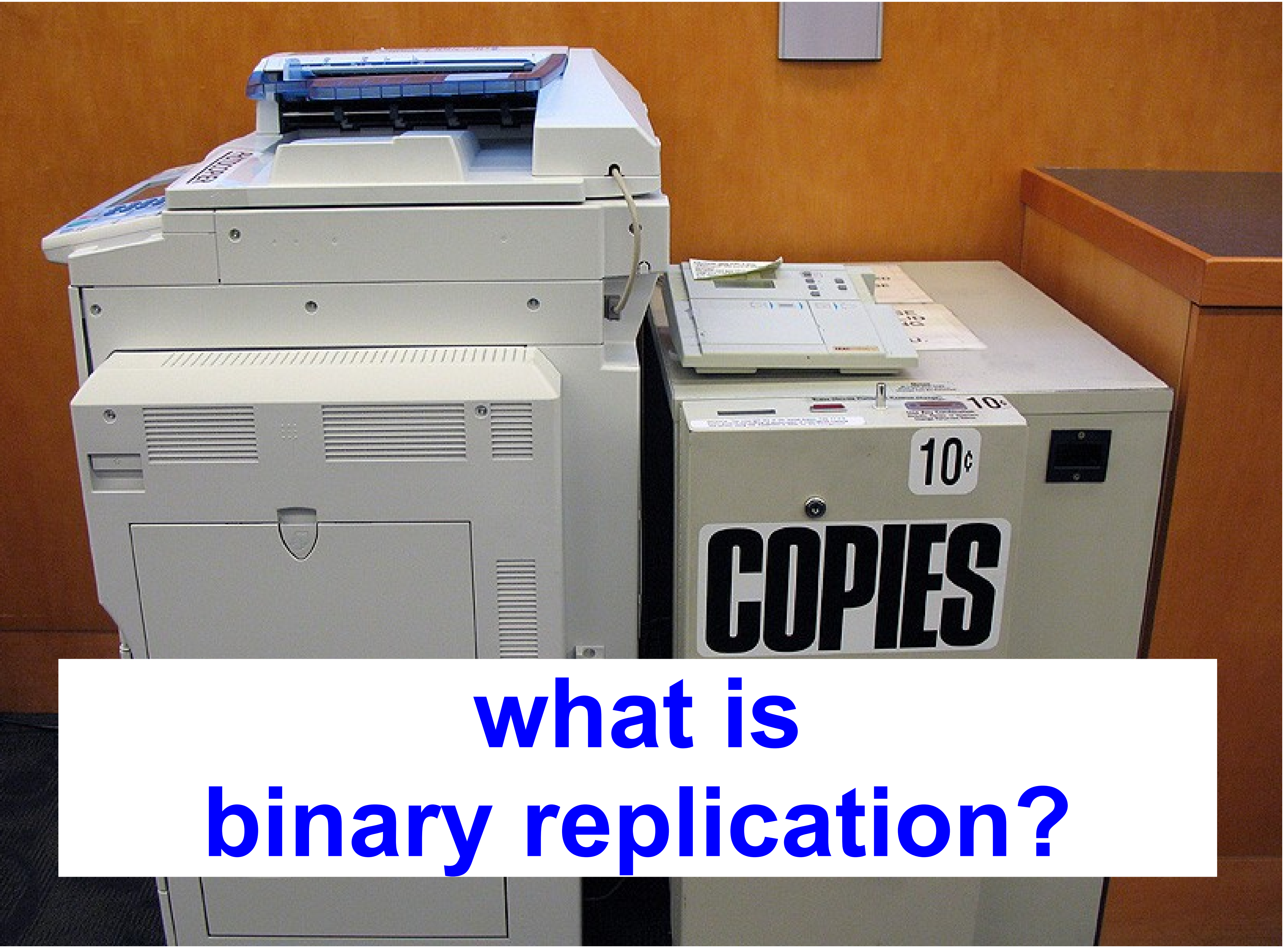
Josh Berkus
PostgreSQL Experts, Inc.
FOSS4G 2014

covered

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

not covered

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



what is
binary replication?

vagrant up now

replication terms

master / slave

master / standby

master / replica

primary / secondary

primary / replica

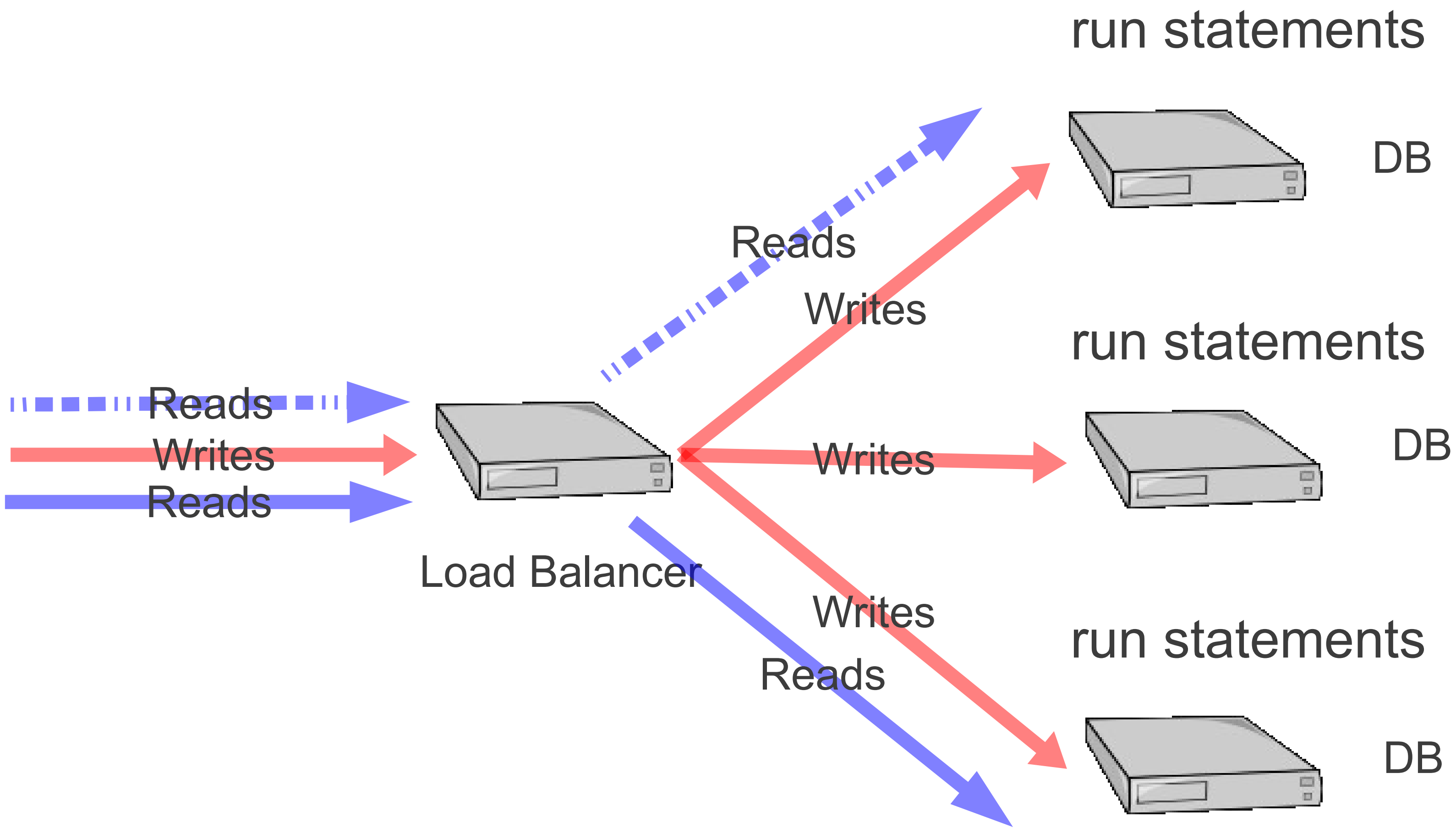
replication mechanisms

1. statement
2. row
3. binary

replication mechanisms

1. queries
2. rows
3. data pages

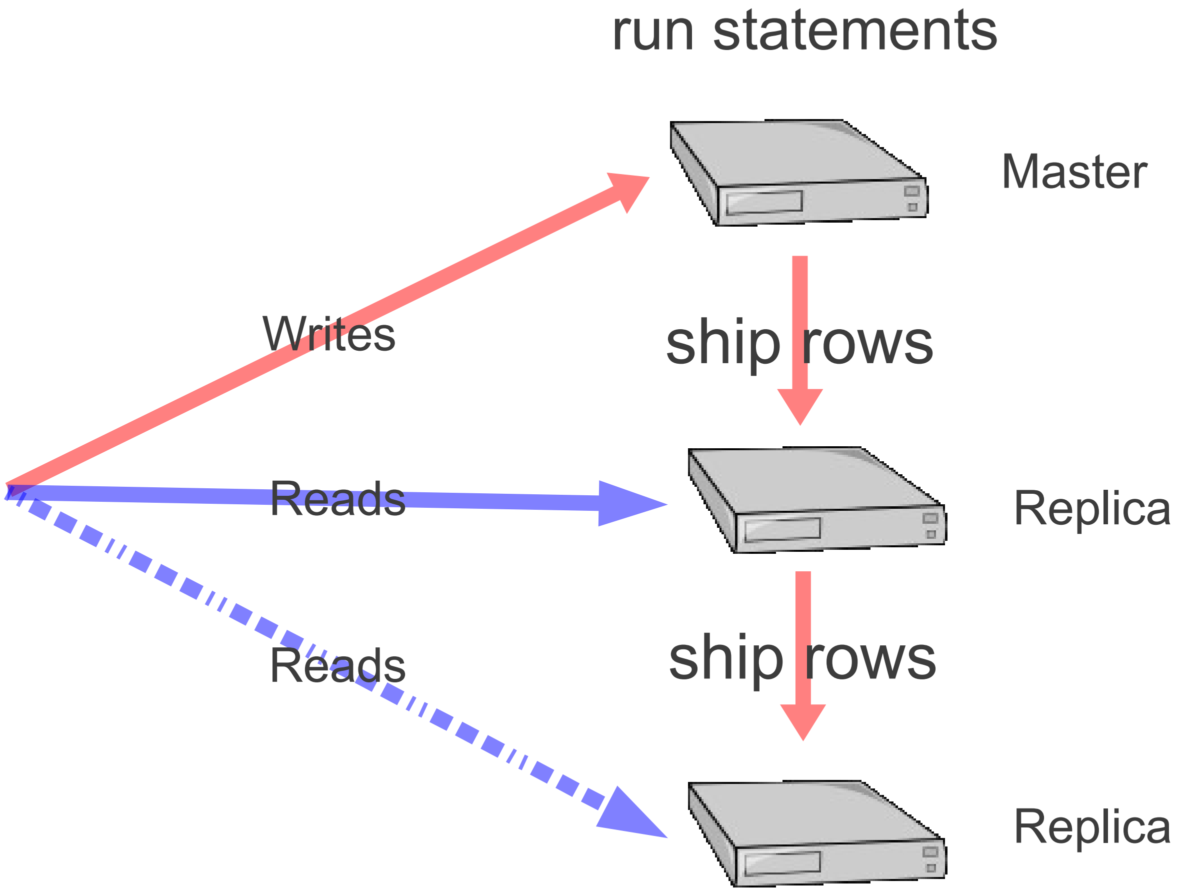
statement replication



statement replication

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

row-based replication



run statements

Master

Writes

ship rows

Reads

Replica

ship rows

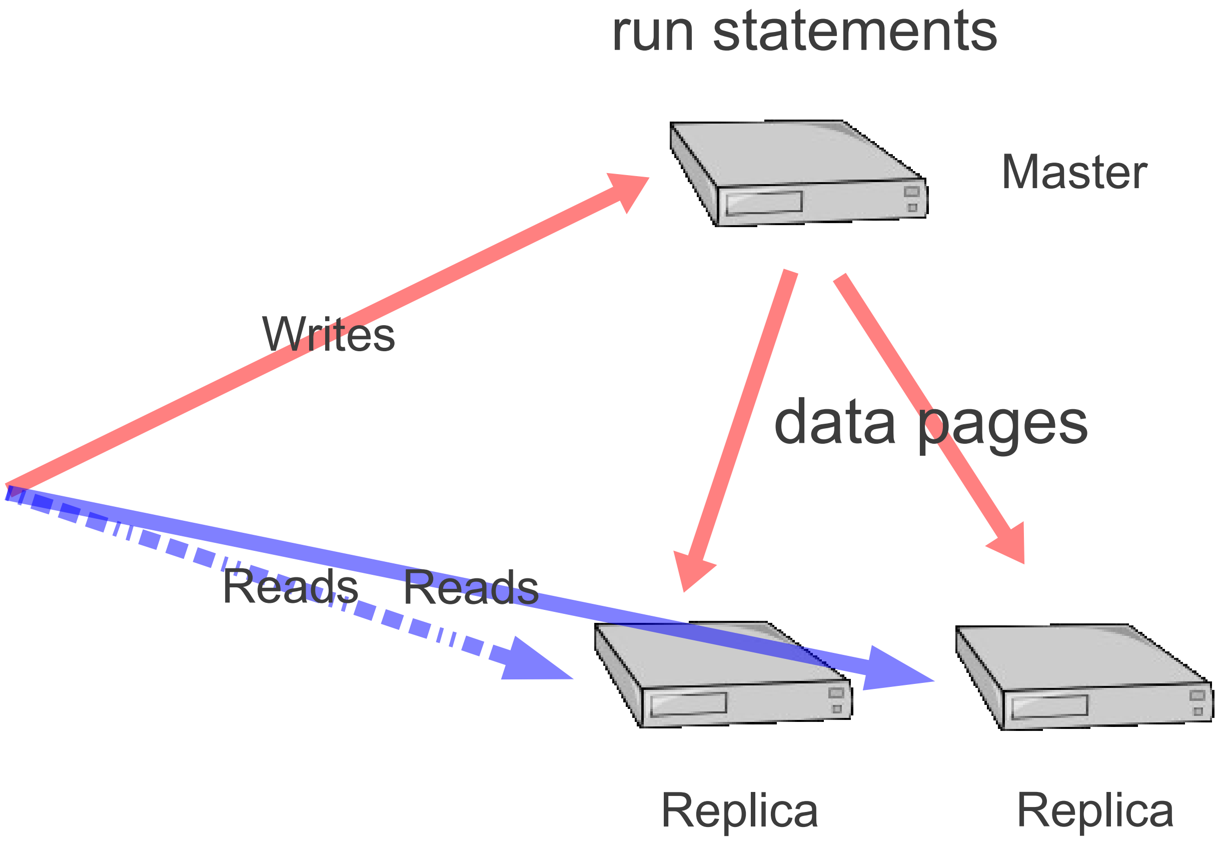
Reads

Replica

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

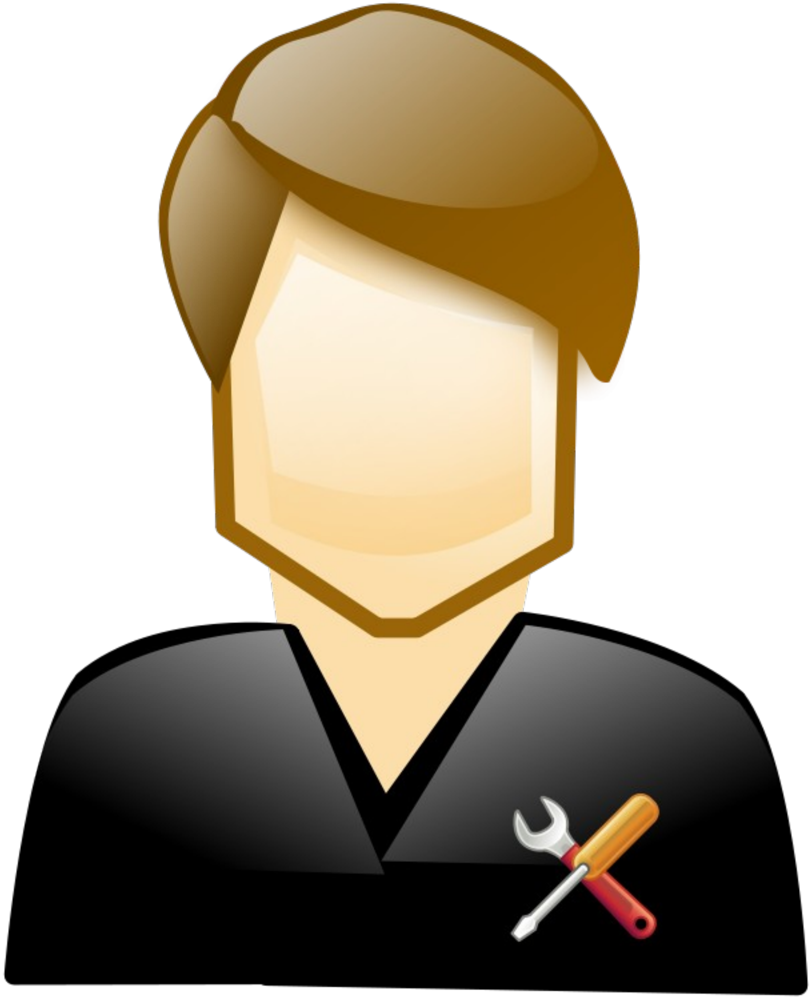
streaming asynchronous replication

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

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

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

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

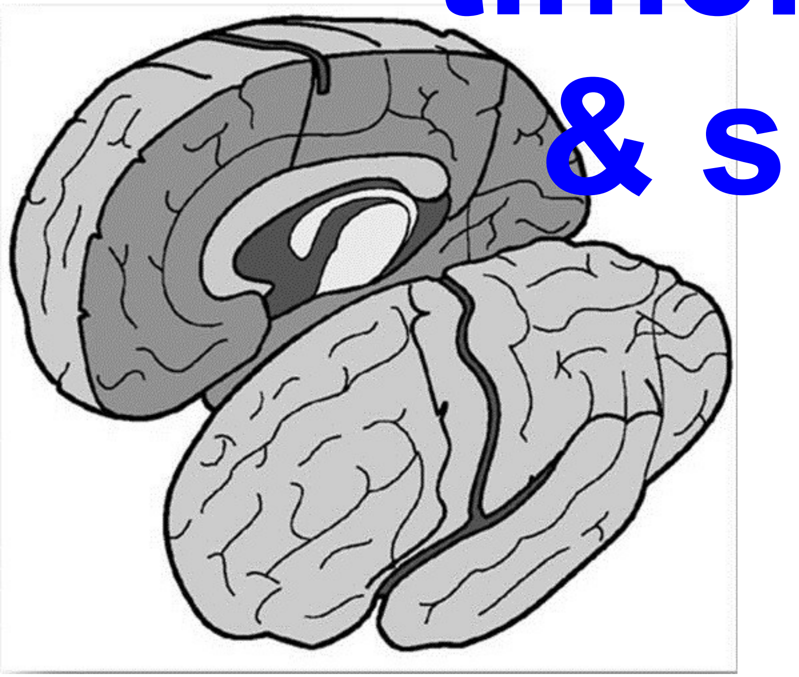
more terms

- **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 failback: 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

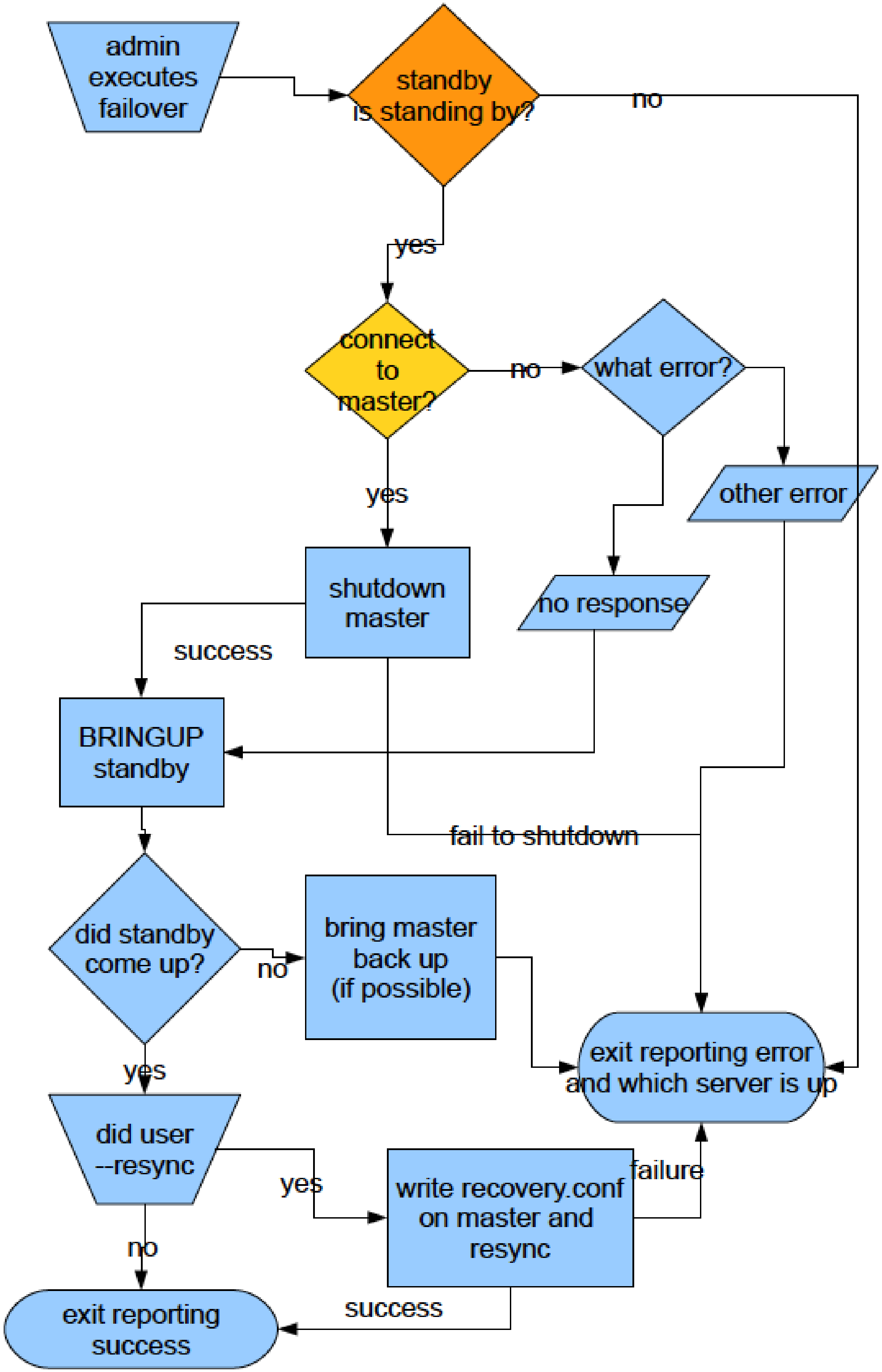
manual failover

- 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

HandyRep

www.handyrep.org

remastering

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

remastering scripts

- search “database soup remastering”
  - http://www.databasesoup.com/2012/10/determining-furthest-ahead-replica.html
  - https://gist.github.com/jberkus/3850644

replication lag
&
query cancel

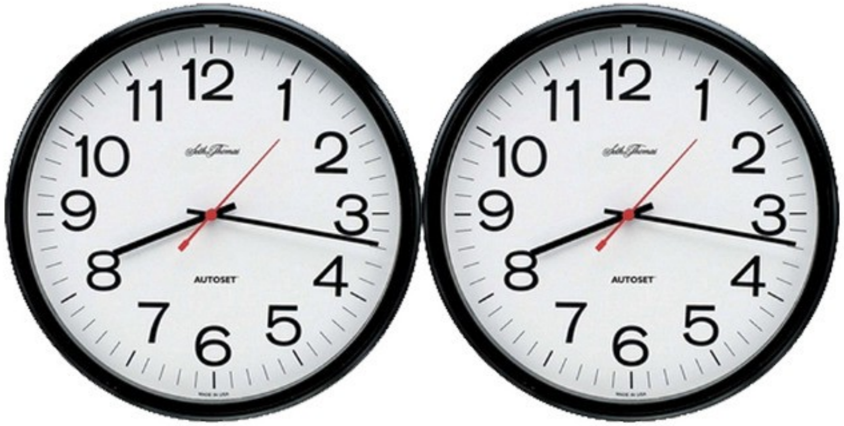
reasons for lag

- 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

configuring lag hands-on



synchronous replication

what synch rep does

guarantee against data loss

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 asych 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

cascading replication

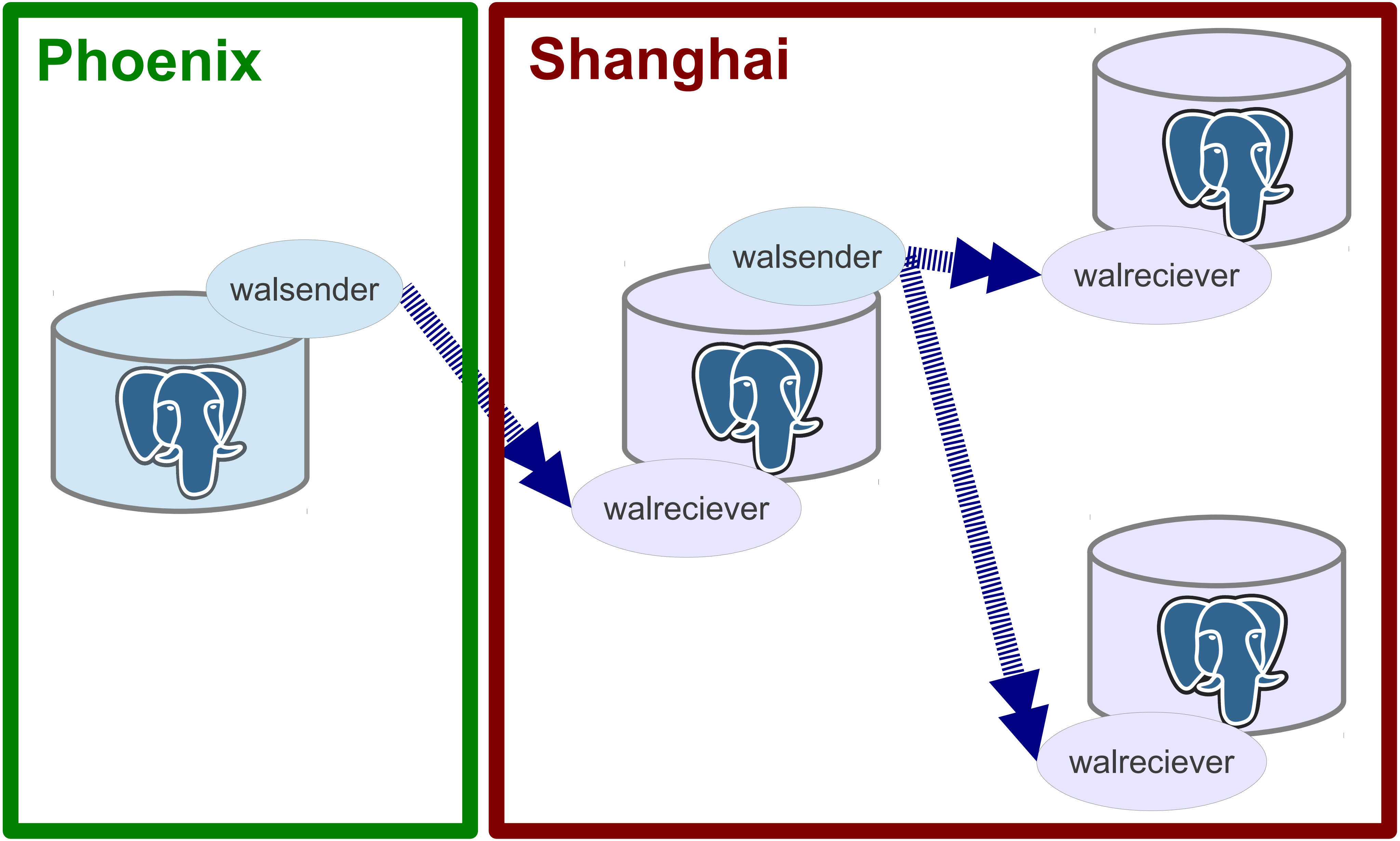
how to cascade

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



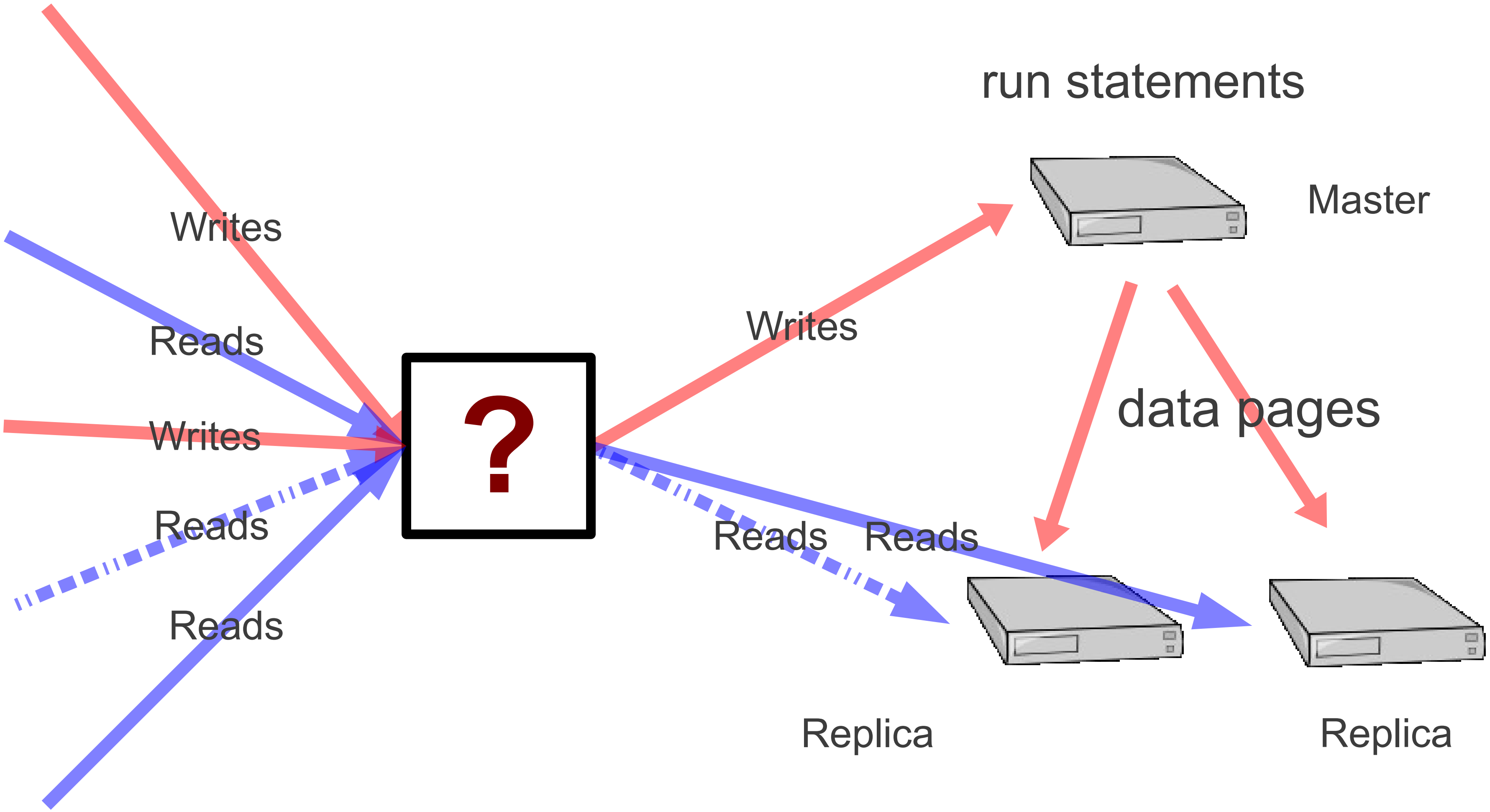
why not cascade?

- 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

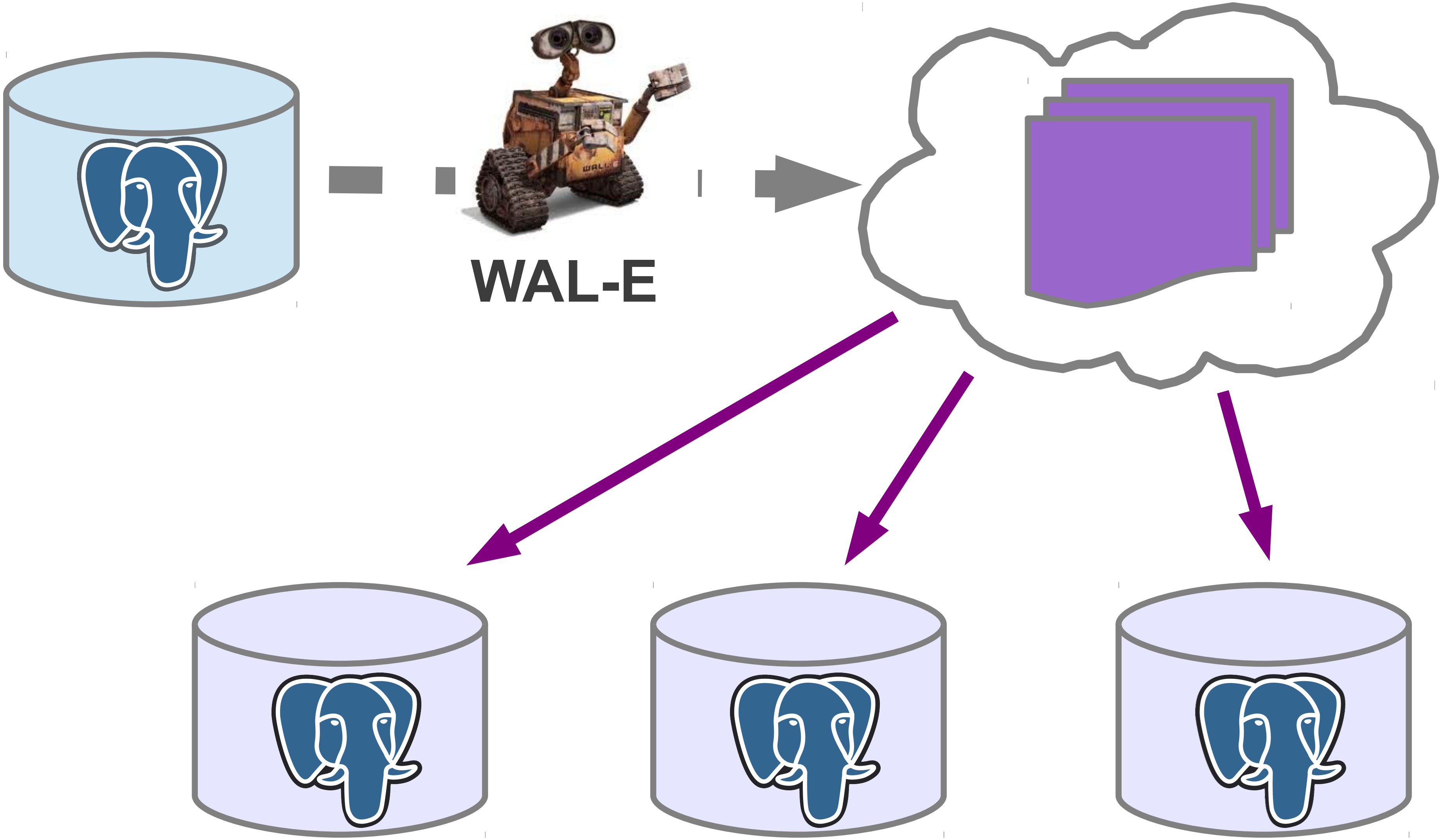
pgBouncer

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

pgBouncer load-balancing exercise

replication in the cloud

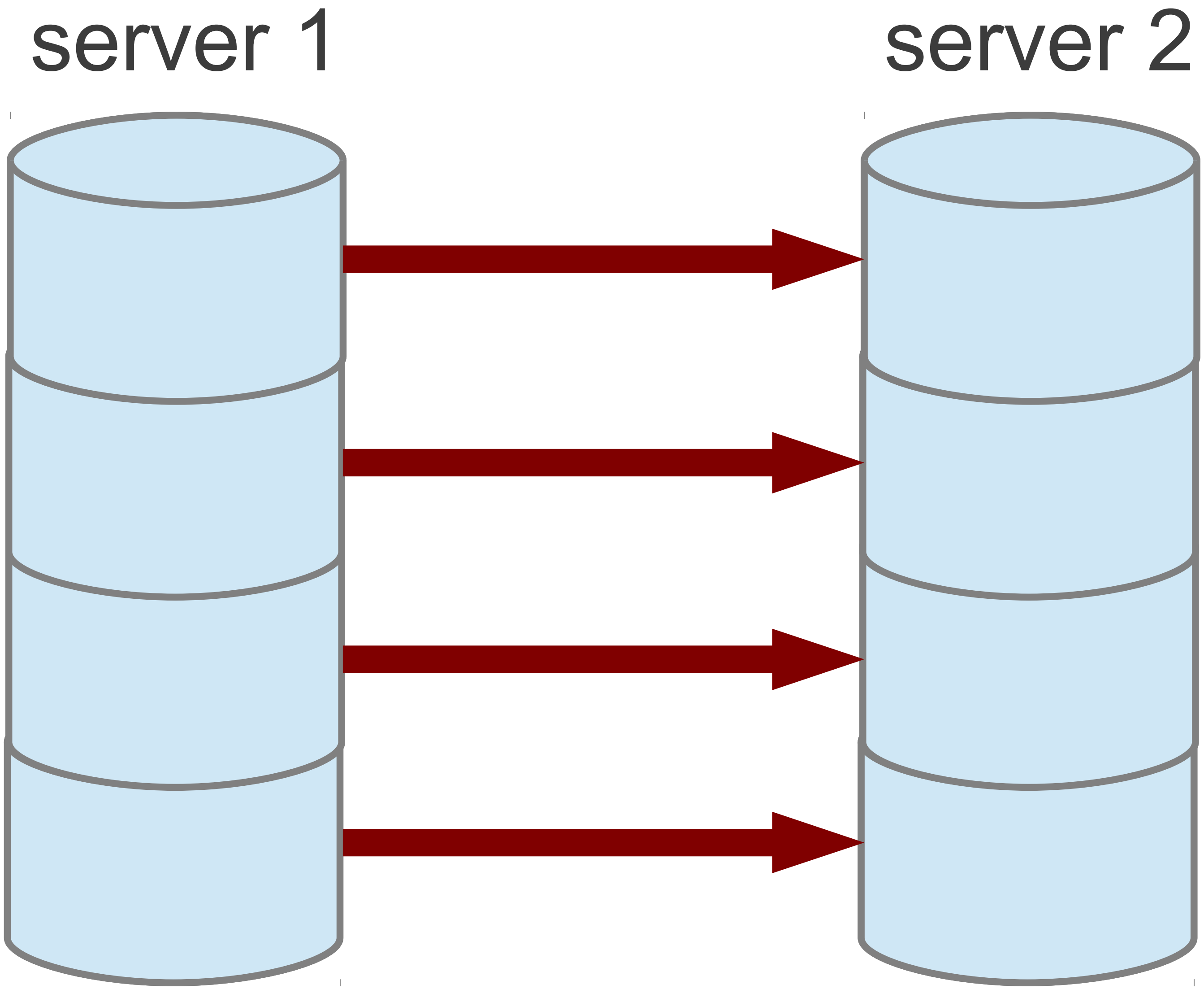
use a shared archive



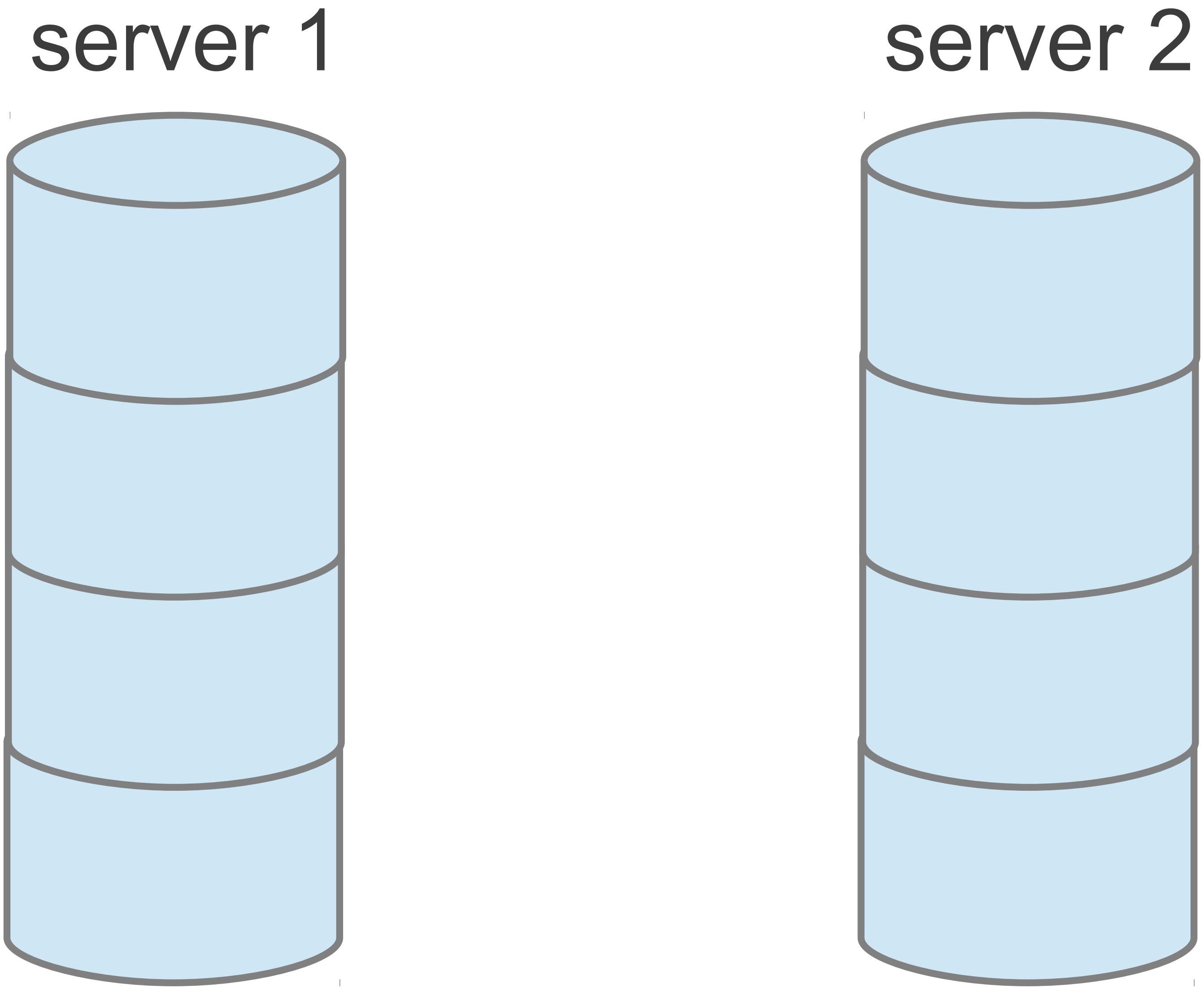
ephemeral replicas

- 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!

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

questions?

- github.com/jberkus/pgReplicationTutorial
- Josh Berkus: josh@pgexperts.com
  - PGX: www.pgexperts.com
  - Blog: www.databasesoup.com
- Upcoming Events
  - Postgres Open: Chicago, one week!
  - pg.EU: Madrid, October 23

PGX POSTGRESQL EXPERTS, INC.