

# The Accidental DBA

*a guide for  
the perplexed*



# covered in this talk

- installation
- extensions
- updates and upgrades
- configuration
- connections & security
- backups
- replication
- VACUUM
- query management

# not covered

- older versions ( $< 9.1$ )
- schema design
- query rewriting
- indexes
- testing
- application stuff

# Exercises & Slides

- [www.pgexperts.com/tutorials.html](http://www.pgexperts.com/tutorials.html)
- exercises are Vagrant VM
  - you needed to install this last week

**“You know Linux,  
right?  
You're in charge  
of the database  
now.”**

**“Efficiency”**

**“DevOps”**

**“Cloud”**

**“Growth Opportunity”**

**“We're not going to  
hire a DBA”**

# Y U no DBA?

- 1.limited budgets
- 2.shortage of operational staff
- 3.cheaper OSS databases

... you are the DBA now.



[Home](#) → [Documentation](#) → [Manuals](#) → [PostgreSQL 9.2](#)

This page in other versions: [9.2](#) / [9.1](#) / [9.0](#) / [8.4](#) | Unsupported versions: [8.3](#) / [8.2](#) / [8.1](#) / [8.0](#) / [7.4](#) / [7.3](#) / [7.2](#) / [devel](#)

## PostgreSQL 9.2.4 Documentation

### The PostgreSQL Global Development Group

[Copyright](#) © 1996-2013 The PostgreSQL Global Development Group

---

#### Table of Contents

##### [Preface](#)

[What is PostgreSQL?](#)

[A Brief History of PostgreSQL](#)

[Conventions](#)

[Further Information](#)

[Bug Reporting Guidelines](#)

#### I. [Tutorial](#)

1. [Getting Started](#)

2. [The SQL Language](#)

3. [Advanced Features](#)

#### II. [The SQL Language](#)

4. [SQL Syntax](#)

5. [Data Definition](#)

6. [Data Manipulation](#)

7. [Queries](#)

8. [Data Types](#)

9. [Functions and Operators](#)

10. [Type Conversion](#)

11. [Indexes](#)

12. [Full Text Search](#)

13. [Concurrency Control](#)

14. [Performance Tips](#)





**Oh My God,  
We're All Going To Die.**









# vagrant up now

```
cd /dir/to/accidentalDBA/vagrant/  
vagrant up
```



# Installation

# Linux: use packages!

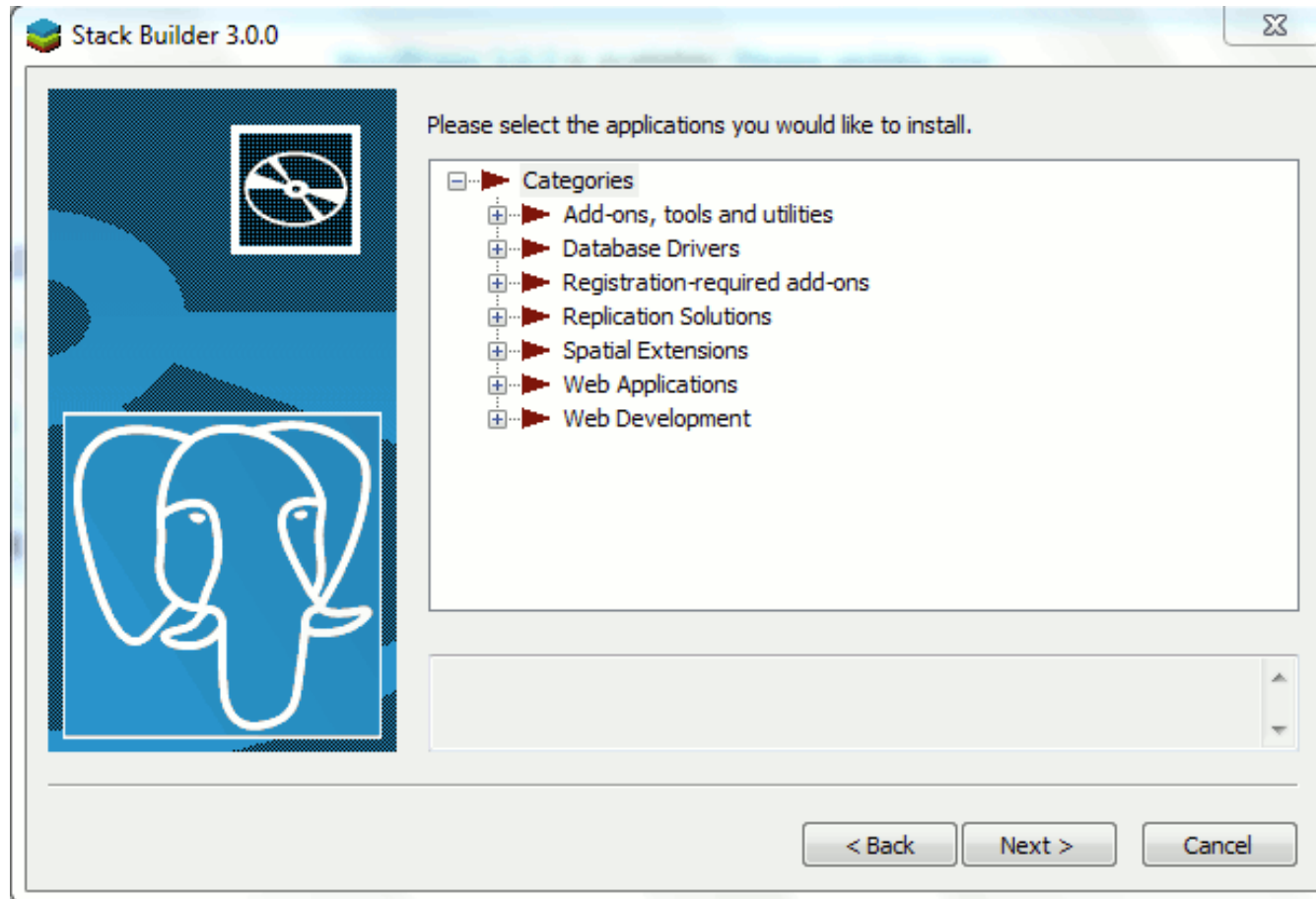
- version not important?
- use the ones that come with your distro
  - Red Hat, Centos, SciLinux
  - Debian, Ubuntu
  - SuSE



# Linux: use packages!

- need the latest version?
- alternate packages
  - Red Hat: [yum.postgresql.org](http://yum.postgresql.org)
  - Ubuntu: [apt.postgresql.org](http://apt.postgresql.org)
  - SuSE: build service
  - Debian: [apt.postgresql.org](http://apt.postgresql.org)

# Windows/OSX



# Windows/OSX

- use the graphical installer
  - from EnterpriseDB
  - “wizard” GUI
- also installs optional components
  - pgAdmin
  - Extensions

# other platforms

- Packages available:
  - Solaris 10/11
  - OpenSolaris/Illumos
  - FreeBSD
  - OpenBSD
  - NetBSD
- No Packages:
  - HP-UX
  - AIX
  - Solaris 9
  - Tablets
  - “Home” Windows

# get logged in

```
vagrant ssh
```

```
sudo su -
```

```
tmux
```

# packages exercise

```
less  
/etc/apt/sources.list.d/pgdg.list  
apt-cache search postgresql
```

# create data directory

- \$PGDATA is where the database files live
- most packages create it
- if not, use “initdb” to create it
  - pick a suitable location!

```
initdb -D /db/9.2/main
```

# initdb exercise

```
su - postgres  
mkdir test  
initdb -D test  
cd test  
ls -l
```





**starting & stopping**

# 3 commands

service / initdb scripts

- normal use, recommended

pg\_ctl

- for custom configurations
- must be run as “postgres”

postgres

- only for control script-writers



# 4 states

1. start
2. stop
3. restart
4. reload

# start & stop

- what it says on the tin
- except ...
  - “smart” stop: wait for all connections
  - “fast” stop: force disconnect
  - “immediate” stop: just like “kill”

# restart

- shuts down and restarts postgres
- breaks all connections
- required for:
  - changes to memory, connections
  - changes to archive\_mode
  - changes to logging\_collector

# reload

- signals the postmaster to reload files
- does not break connections
- works for
  - changes to security (pg\_hba.conf)
  - most changes to logging
  - changes to defaults

# start/stop exercise

```
service postgresql start
```

```
service postgresql restart
```

```
service postgresql reload
```

```
service postgresql stop
```



# start/stop exercise

```
su - postgres
```

```
pg_ctl -D /etc/postgresql/9.2/main  
start
```

```
pg_ctl -D /etc/postgresql/9.2/main  
-m fast stop
```

```
exit
```

```
service postgresql start
```

```
Sidney-Stratton:~ josh$ psql libdata
psql (9.1.1)
Type "help" for help.
```

```
libdata=# \dt
```

### List of relations

Schema	Name	Type	Owner
public	books	table	libdata
public	branches	table	libdata
public	copies	table	libdata
public	copy_history	table	libdata
public	copy_status	table	libdata

# the psql command line

# pgAdmin

The screenshot displays the pgAdmin 3 interface. The Object browser on the left shows a tree structure of server groups, including 'serveur 1 (localhost:5432)' which contains databases like 'b1', 'b2', and 'benchs'. The main window shows the 'Properties' tab for the 'pgbench\_accounts' table, listing various attributes such as Name, OID, Owner, Tablespace, ACL, Of type, Primary key, Rows (estimated), Fill factor, Rows (counted), and Inherits tables.

Below the main window, the SQL Editor is visible with the query: `select * from pg_stat_activity`. The Output pane at the bottom shows the results of the query, displaying a table with columns: datid, oid, datname, procid, usesysid, username, application\_name, client\_addr, client\_port, and back\_time. The results are as follows:

	datid	oid	datname	procpid	usesysid	username	application_name	client_addr	client_port	back_time
1	11874	postgres	12482	10	guillaume		:::1	49467	2010	
2	33018	b1	12483	10	guillaume		:::1	49468	2010	
3	16384	benchs	12485	10	guillaume		:::1	49469	2010	
4	16384	benchs	12490	10	guillaume		:::1	49470	2010	

The Properties dialog for the 'pgbench\_accounts' table is also shown, displaying fields for Name, OID, Owner, Tablespace, Of type, Fill Factor, Has OIDs, and Comment. The 'Name' field is highlighted, and the 'Owner' is set to 'guillaume'.

# psql exercise

```
su - postgres
```

```
psql libdata
```

```
\?
```

```
\h create table
```

```
\dt
```

```
\d+ copies
```



**extensions**

# extensions?

- extensions add extra functionality
  - like Python/Perl modules, Ruby Gems, etc.
- need to be installed separately
  - some come with PostgreSQL packages
  - some need to be installed from PGXN or source
- handled very differently before 9.1

# libdata extensions

Extension	Purpose	Source
plpgsql	<i>Procedural language for triggers and functions. Automatically included 9.0 and later.</i>	core
citext	<i>Case-insensitive text data type</i>	contrib
isn	<i>Product scan code data type. Used in libdata for ISBNs</i>	contrib

# installing extensions

1. install the binary files
  - using packages, PGXN, or source
  - installs to postgres “share” directory
  - a few extensions don't have binaries
2. install the extension in each database where it's used



# extensions exercise

```
\dx
```

```
create extension hstore;
```

```
select * from pg_extension;
```

```
select 'fname => josh,  
       lname => berkus'::hstore;
```



**updates  
&  
upgrades**

🏠 [InfoWorld Home](#) / [Security](#) / [News](#) / PostgreSQL updates address high-risk...

APRIL 05, 2013

# PostgreSQL updates address high-risk vulnerability, other issues

**VMware also releases fixes for its PostgreSQL-based vFabric Postgres database product**

By Lucian Constantin | IDG News Service

 [Print](#) |  [Add a comment](#)



3



More

The PostgreSQL developers released updates for all major branches of the popular open-source database system on Thursday in order to address several vulnerabilities, including a high-risk one that could allow attackers to crash the server, modify

# major vs. minor

**9.2** == a major version

- requires an *upgrade* from 9.1.8
- contains features not in 9.1
- requires testing and planned downtime

**9.1.9** == a minor version

- is a minor *update* from 9.1.8.
- can (and should) be applied immediately

# minor updates

- come out ~~ every 2 months
- contain only bugfixes
  - security hole patches
  - data loss prevention
  - fix server crashes
- no new or changed features
  - *occasional* documented breakage

**update promptly**  
**update often**

# update procedure

- 1.schedule 5 minute downtime
- 2.download packages
- 3.shut down postgresql
- 4.install packages
- 5.restart postgresql
- 6.restart application

# major upgrades

- come out once per year
- have many new features
  - and sometimes break stuff which used to work
- require extensive testing with your application
- require significant downtime to upgrade



# upgrade procedures

- dump & reload
  - use pg\_dump & pg\_restore on database
  - most reliable way
  - “cleans up” database in process
  - best with small databases
  - can take a long, long time

# upgrade procedures

- pgUpgrade
  - upgrade “in place”
  - much faster
  - does not “clean up” database
  - sometimes doesn't work
    - issues with extensions

**EOL after  
5 years**

# upgrading extensions

*not possible before 9.1*

1. upgrade Postgres
2. install new extension binaries
3. run upgrade script in each DB

```
ALTER EXTENSION hstore  
UPDATE;
```



**configuration**

# configuration

1. Hardware
2. OS/FS
3. PostgreSQL

# use good hardware

- databases use *all* the hardware
  - RAM, CPU, IO
  - disk can be very important
    - DB larger than RAM
    - write-heavy database
- *the database cannot outperform bad hardware*

**put the database  
on its own server  
(or virtual server)**



# cloud servers

- cloud server performance sucks
  - especially IO
- make sure you have enough RAM to cache the *whole* database

# Linux configuration

1. turn the OOM killer off
2. turn reclaim\_zone\_files off
3. use XFS or Ext4 for database files
  1. use “noatime,nodiratime”
4. increase shmmax, shmall
  1. so that you can raise shared\_buffers
  2. not required for 9.3beta, yaaaay!

# tmux new window

```
<ctrl>b, c
```

```
<ctrl>b, n
```

```
<ctrl>b, n
```

# kernel params

```
less  
/etc/sysctl.d/30-postgresql-shm.conf
```

- Q: what would it take for 1GB?

# BSD/Solaris Config.

- Use ZFS
  - decrease block size to 8K
- increase shmmax/shmall on BSD
- may need to mess with ulimits
  - on very busy systems
  - Postgres will give you errors

# Windows/OSX

*optimization not possible*

# the xlog

- xlog == WAL
  - where transactions are recorded
- best on its own drive/volume
  - write-only
  - writes synchronously
  - response time paramount
- create volume, then link pg\_xlog dir

# xlog dir

```
<ctrl>b,n
```

```
cd 9.2/main/pg_xlog
```

```
ls -lh
```



# postgresql.conf

**parameters  
you care  
about:**

10 to 25

**parameters  
you don't care  
about:**

206 to 207

# postgresql.conf

```
cd  
/etc/postgresql/9.2/main  
$EDITOR postgresql.conf
```

- editors available:
  - joe, jmacs, vi, nano

# postgresql.conf

```
psql
show max_connections;
show all;
\x

select * from
pg_settings;
```



**connections  
&  
security**

# network

- local connections: UDP, if possible
  - faster than TCP/IP
- other servers: port 5432
  - make sure it's open on the firewall!
- on the cloud? use SSL
  - secure your connections
  - PITA to set up, though

# max\_connections

`"ERROR: connection limit  
exceeded for non-superusers"`

- postgresql.conf
- increase number of connections
- good up to about  $20 + 10 \times \text{cores}$
- keep needing to increase it?  
something is wrong

# security

```
"FATAL: password authentication
failed for user "wwwuser"
```

- Postgres users & passwords
  - CREATE/ALTER USER
  - “group” ROLES
- Or: use LDAP, GSSAPI or PAM

# create user

```
psql
```

```
create user 'bench'  
password 'benchmark';
```

```
\du
```

```
\q
```



# host-based access

```
"FATAL: no pg_hba.conf entry for host  
"192.168.0.1", user "chaos", database  
"chaosLRdb", SSL off"
```

- pg\_hba.conf
- access control list:
  - database/user/host address
  - like iptables for Postgres
- change config and reload

# pg\_hba.conf

```
<ctrl>b,n
```

```
$EDITOR /etc/postgresql  
    /9.2/main/pg_hba.conf
```

```
service postgresql  
reload
```

# pg\_hba.conf

```
<ctrl>b,n
```

```
psql
```

```
\q
```

```
psql -U bench
```

```
\q
```

# **.pgpass**

```
cd ~  
cp /setup/postgres/.pgpass .  
chmod 700 .pgpass  
less .pgpass  
psql -U bench
```

# three tips for security

1. don't expose the postgres server/port to the internet
2. don't allow users to connect as the superuser (postgres)
3. use the strongest authentication which is practical

# connection pooling

- pgbouncer
  - event-based pooler
  - separate package
  - on DB server, or
  - app server, or
  - 3<sup>rd</sup> “bouncer” server

# pgbench

- simple “benchmark” utility
  - based on “Wconsin” benchmark
  - ships in postgresql-contrib
  - useful for testing for really bad OS/hardware issues
  - also for demos

# setting up pgbench

```
createdb bench
```

```
pgbench -U bench -i -s 10
```



# pgbouncer

```
<ctrl>b,n
```

```
less /etc/pgbouncer/pgbouncer.ini
```

```
less /etc/pgbouncer/userlist.txt
```

```
service pgbouncer start
```

```
<ctrl>b,n
```

```
psql -U bench -p 6432
```

```
\q
```

# pgbouncer

```
cd /setup/pgbench  
less runbench_pool.sh  
./runbench_pool.sh  
<ctrl>b,n  
su - postgres  
psql
```

# pgbouncer

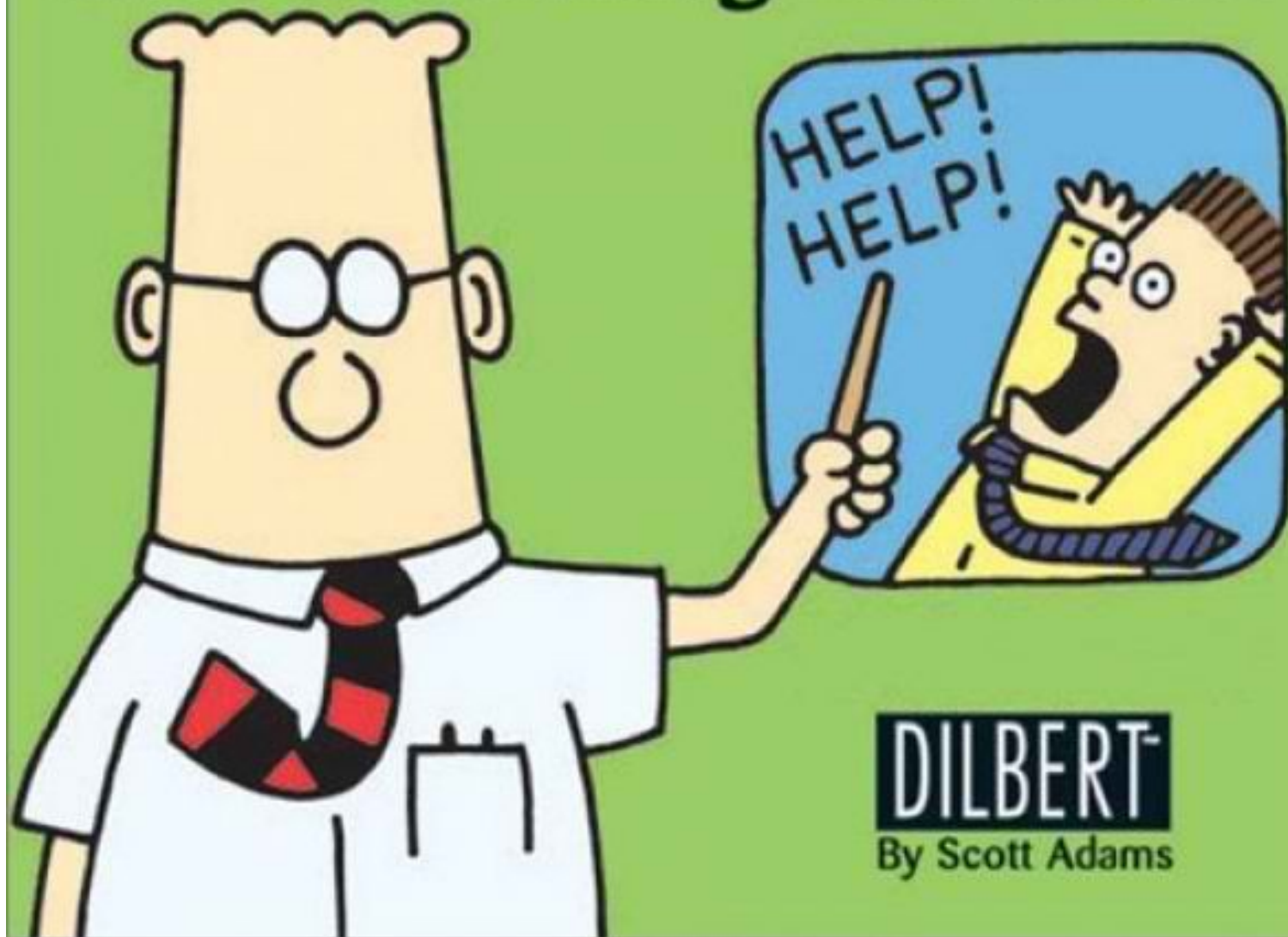
```
select * from pg_stat_activity;  
\q  
<ctrl>b,n  
<ctrl>c
```



**backups**

```
2012-01-27 18:00:44 MSK FATAL:  
    invalid page header in block  
    311757 of relation  
    base/26976/27977  
2012-01-27 18:00:44 MSK CONTEXT:  
    xlog redo insert:  
    rel 1663/26976/27977;  
    tid 311757/44  
2012-01-27 18:00:44 MSK LOG:  
    startup process (PID 392)  
    exited with exit code 1  
2012-01-27 18:00:44 MSK LOG:  
    aborting startup due  
    to startup process failure
```

# Our Disaster Recovery Plan Goes Something Like This...



**DILBERT**  
By Scott Adams

# three methods

A) pg\_dump

B) PITR

C) filesystem snapshot

# pg\_dump

- “logical” backup
  - portable
  - compressed
  - works for upgrades
- good for small databases
- use -Fc
  - custom binary format



# pg\_restore

- used to reload pg\_dumps
  - parallel mode for fast loading
  - extract specific tables, schema
  - extract to an alternate database

# pgdump

```
cd ~
```

```
pg_dump -Fc -v
```

```
-f backup/libdata.dump
```

```
libdata
```

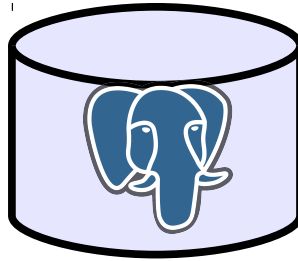
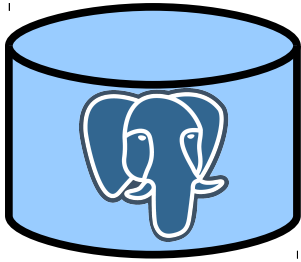
```
ls -lh backup/libdata.dump
```

# pgrestore

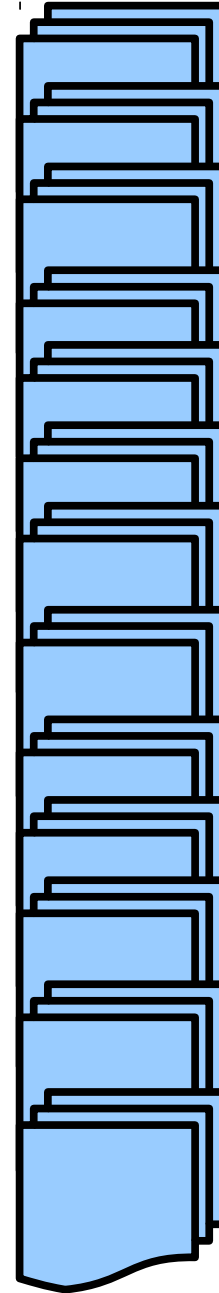
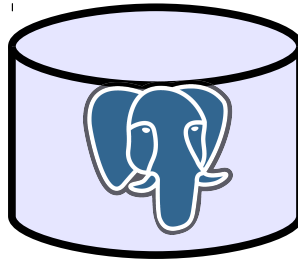
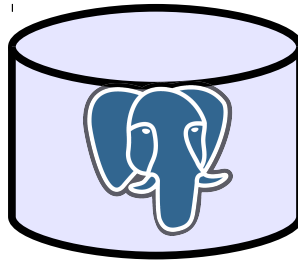
```
pg_restore -l backup/libdata.dump  
createdb libdata2  
pg_restore -v -d libdata2  
        backup/libdata.dump  
psql libdata2  
\dt  
\q
```

# PITR

- “Point-In-Time Recovery”
- “binary” and “continuous” backup
  - take snapshot of DB files
  - accumulate logfile copies
- good for large databases
- can combine with replication



DROP TABLE  
circulation;



# PITR - PITA

- can be difficult to set up & monitor
- use tools:
  - Barman
  - OmniPITR
  - WAL-E (for AWS)

# PITR

```
<ctrl>b,n
```

```
exit
```

```
$EDITOR /etc/postgresql/9.2/main  
/postgresql.conf
```

```
$EDITOR /etc/postgresql/9.2/main  
/pg_hba.conf
```

```
service postgresql restart
```

```
less /setup/postgres/archive_logs.sh
```

# PITR

```
<ctrl>b,n
```

```
cd ~
```

```
pg_basebackup -x -P -D 9.2/replica
```

```
cd wal_archive
```

```
ls -lh
```



# filesystem snapshot

- works with snapshotting filesystems
  - ZFS, BTRFS
  - LVM
  - Some SANs
- must be real snapshot
  - coherent
  - point-in-time

# have a DR plan

- ways you can lose data
  - recovery time
  - acceptable data loss
- how to recover from lost data
  - detailed steps
  - verification

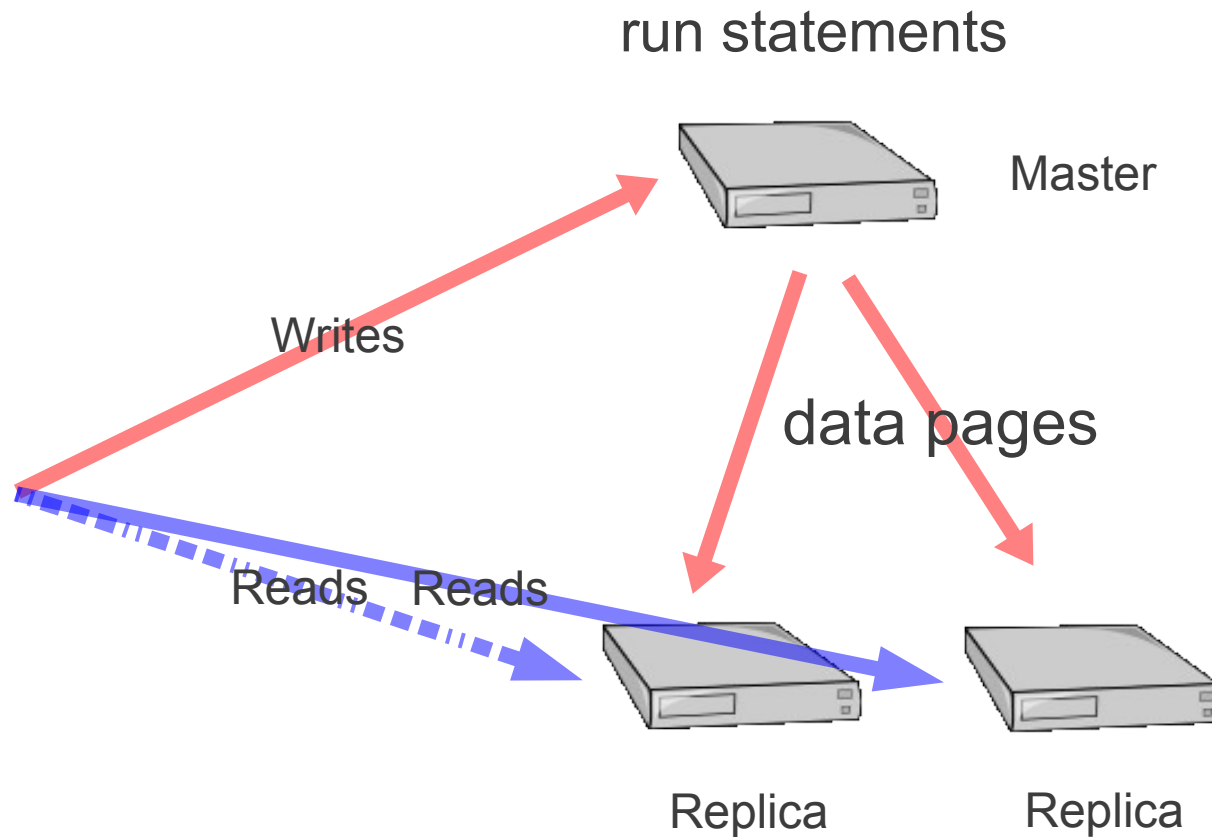


**replication**

# use replication for ...

- **availability:** have an “always on” failover server
- **load-balancing:** offload traffic
  - especially reporting workloads
- **security:** provide users read-only access

# 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
  - not much incremental cost
- non-invasive
  - no extra tables/triggers
  - no primary key requirements
  - no limitations on statements



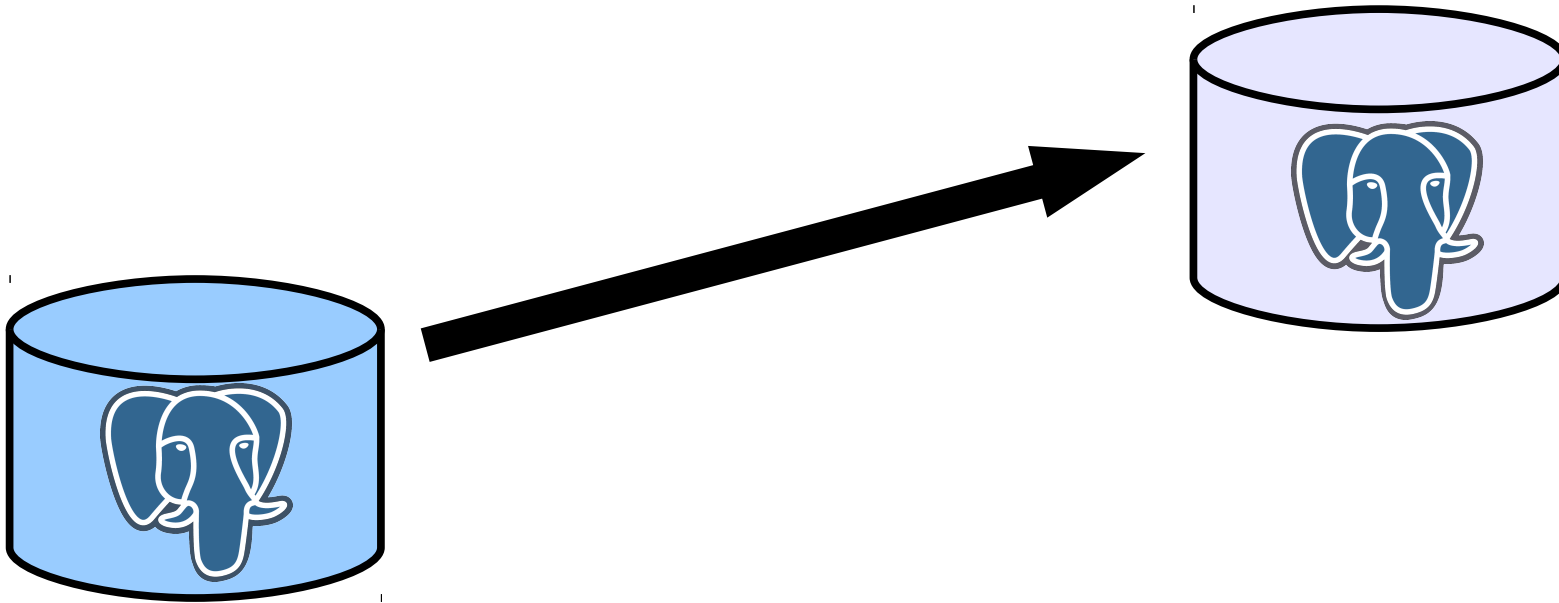
# advantages

- low-latency
- no synchronization issues
- good for very large databases
- works with external disk tools
  - virtual machine cloning
  - filesystem snapshots
- supports disconnected replication

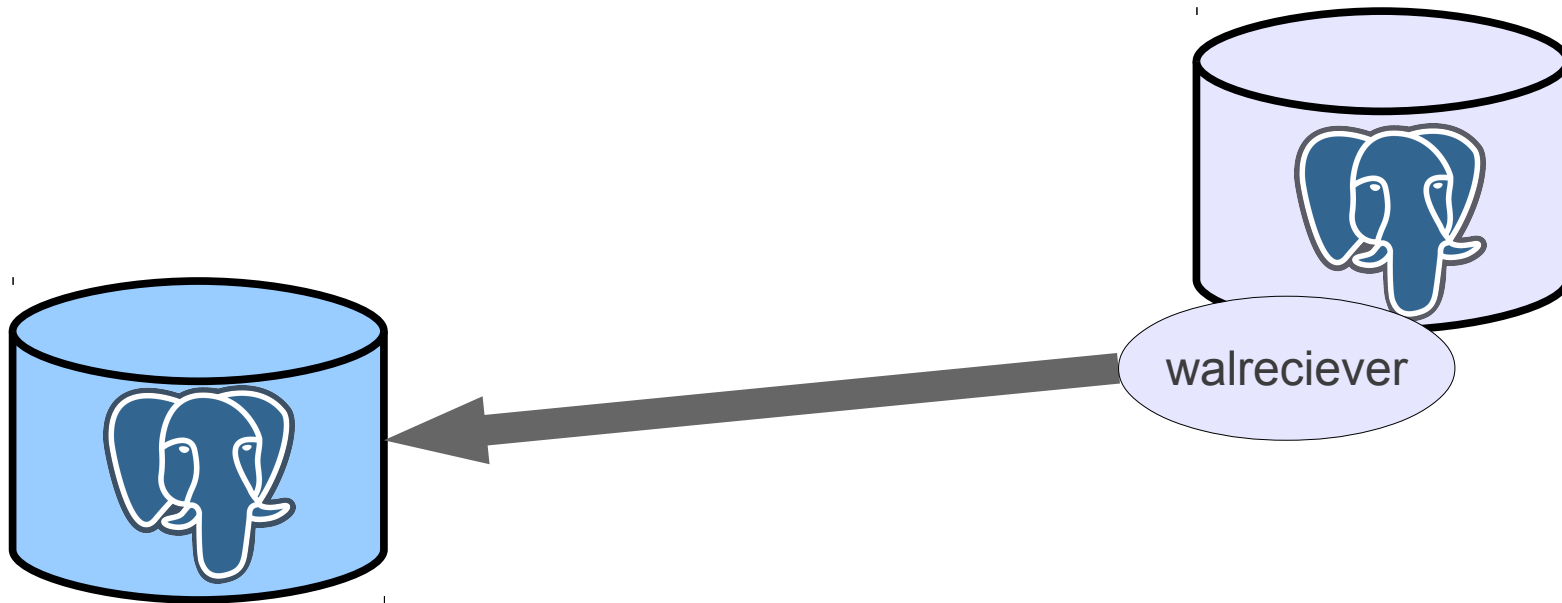
# disadvantages

- need to replicate the whole server
  - not individual databases
  - not specific tables
- no writes of any kind on replicas
- some things not replicated
  - temp & unlogged tables
  - LISTEN/NOTIFY
- query cancel (described later)

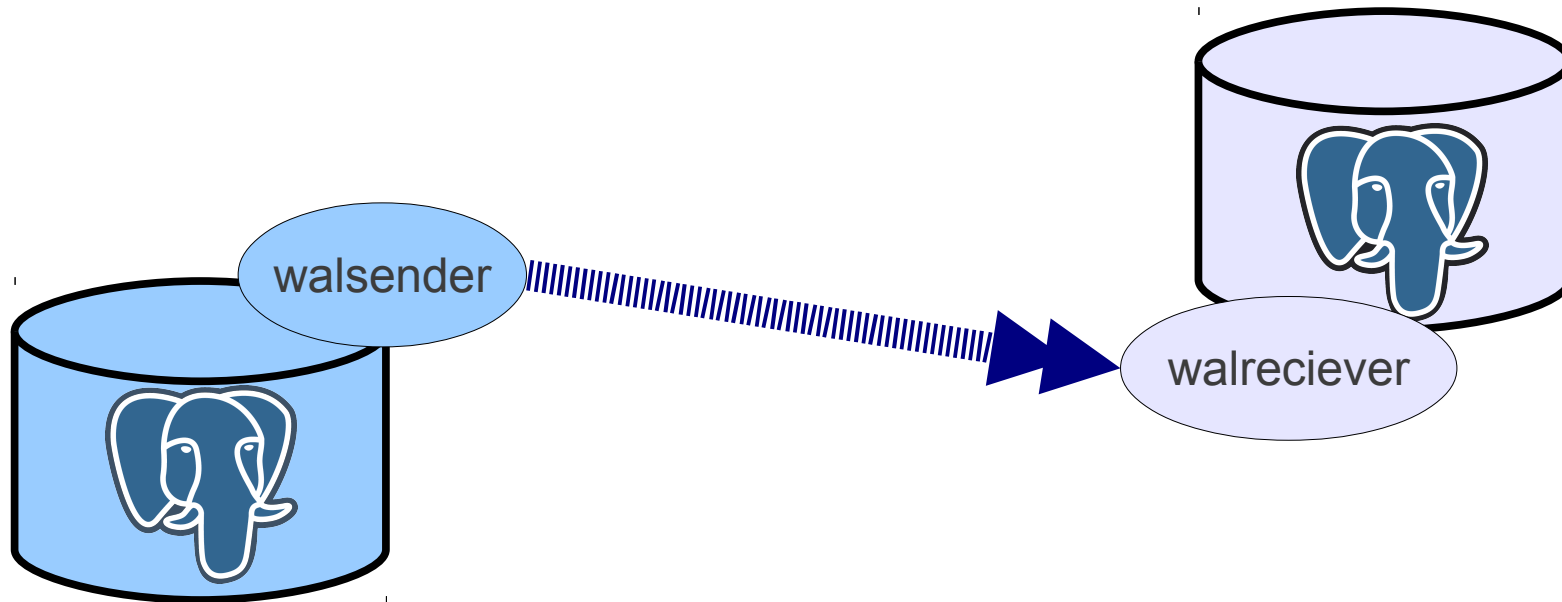
# how it works



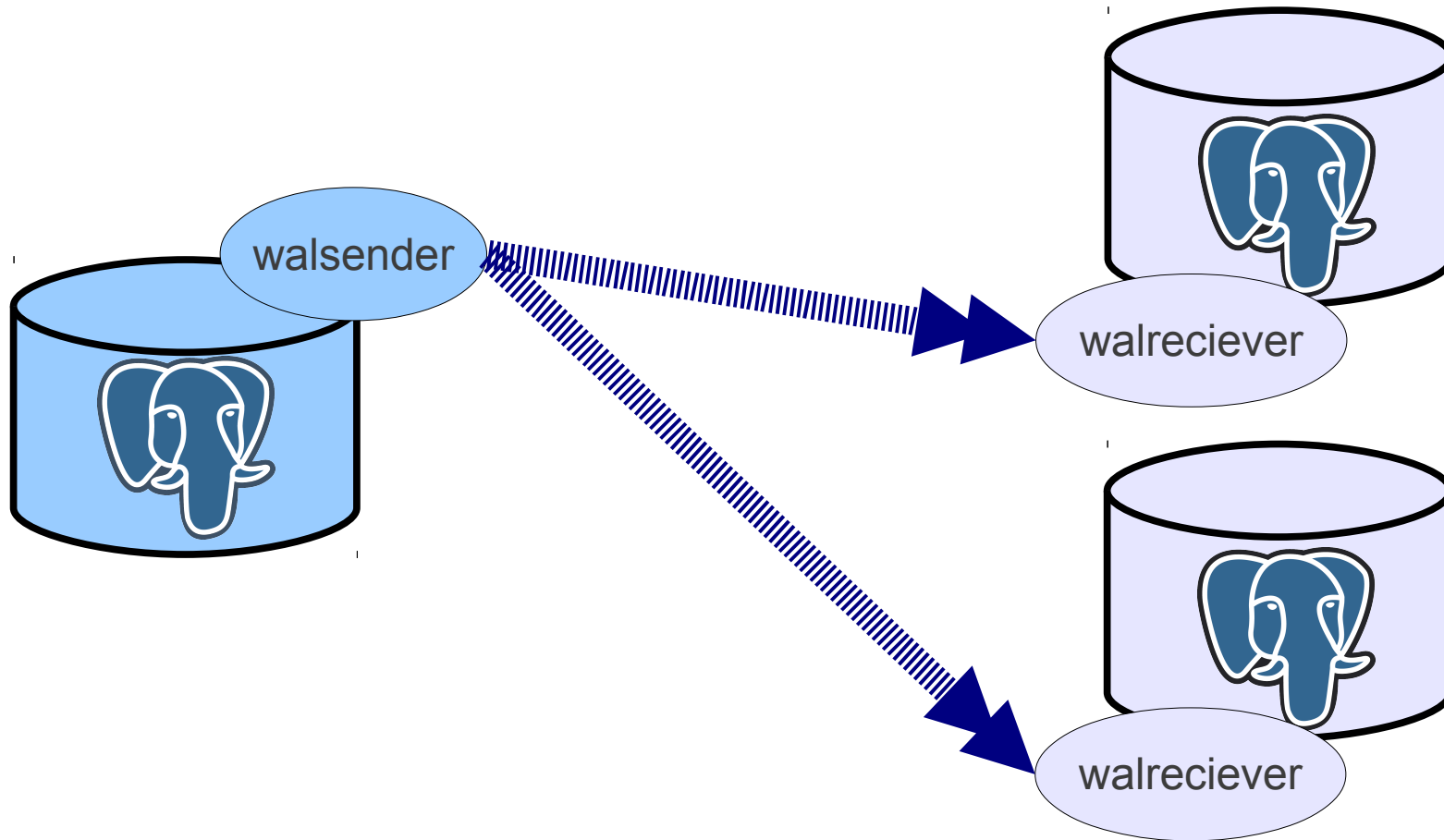
# how it works



# how it works



# how it works



# “recovery”

- historical term
  - because replication grew out of PITR
- used to refer to the replica being “in recovery mode”, i.e no write queries
- all over parameter and file names
  - recovery.conf
  - recovery\_target
  - pg\_is\_in\_recovery()

# replication

```
<ctrl>b,n  
cd ~/9.2/replica  
cp -p -r  
/setup/postgres/archive/* .  
less recovery.conf  
pg_ctl -D . start
```



# replication

```
psql -p 5433
```

```
select pg_is_in_recovery();
```

```
\q
```

```
psql -p 5432
```

```
select * from
```

```
    pg_stat_replication;
```

```
\q
```

# failover

```
pg_ctl -D . promote
```

```
psql -p 5433
```

```
select * from pg_is_in_recovery();
```

```
\q
```

```
pg_ctl -D . stop
```

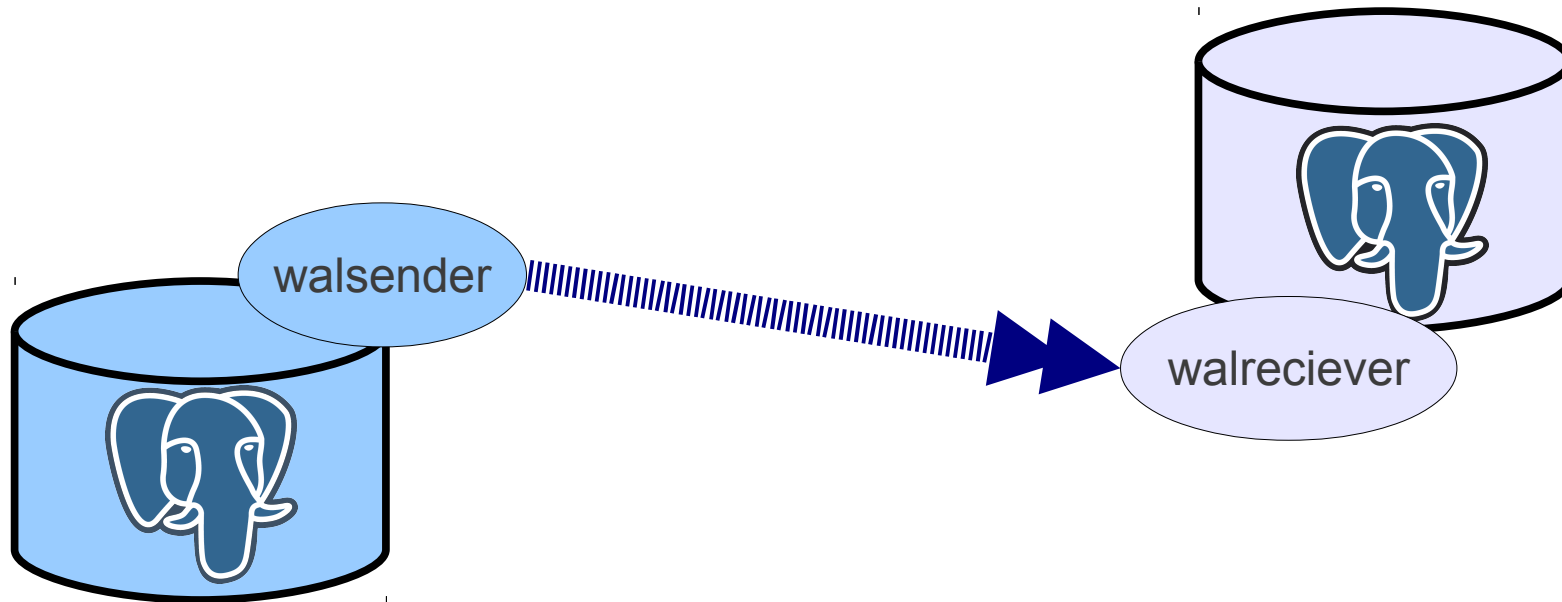
# other cloning methods

- rsync
- filesystem snapshot
  - (on shared storage)
- VM cloning
  - (if rapid)

# creating a snapshot

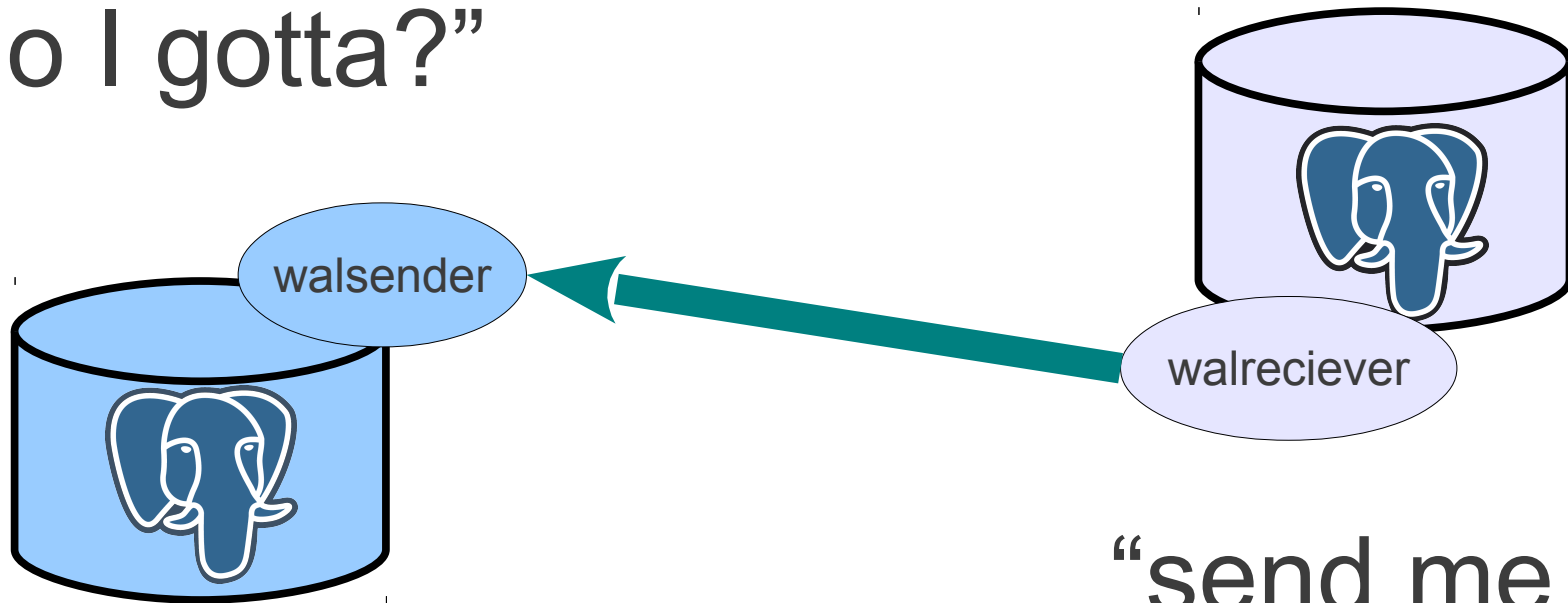
1. make sure you are replicating transaction logs to the replica
2. `pg_start_backup('replication')`
3. `rsync` / snapshot / clone
4. `pg_stop_backup()`

# replica out of sync



# replica out of sync

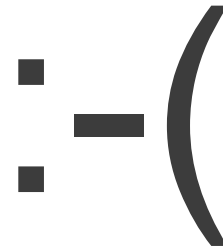
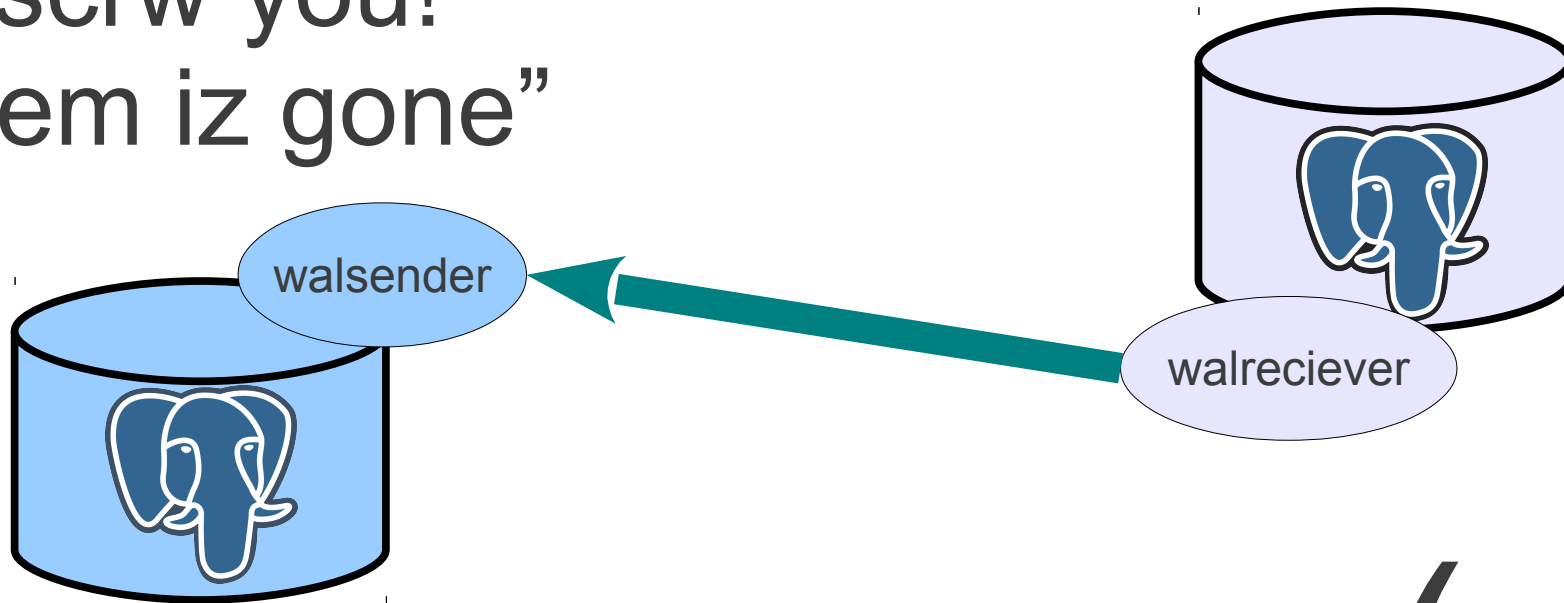
“do I gotta?”



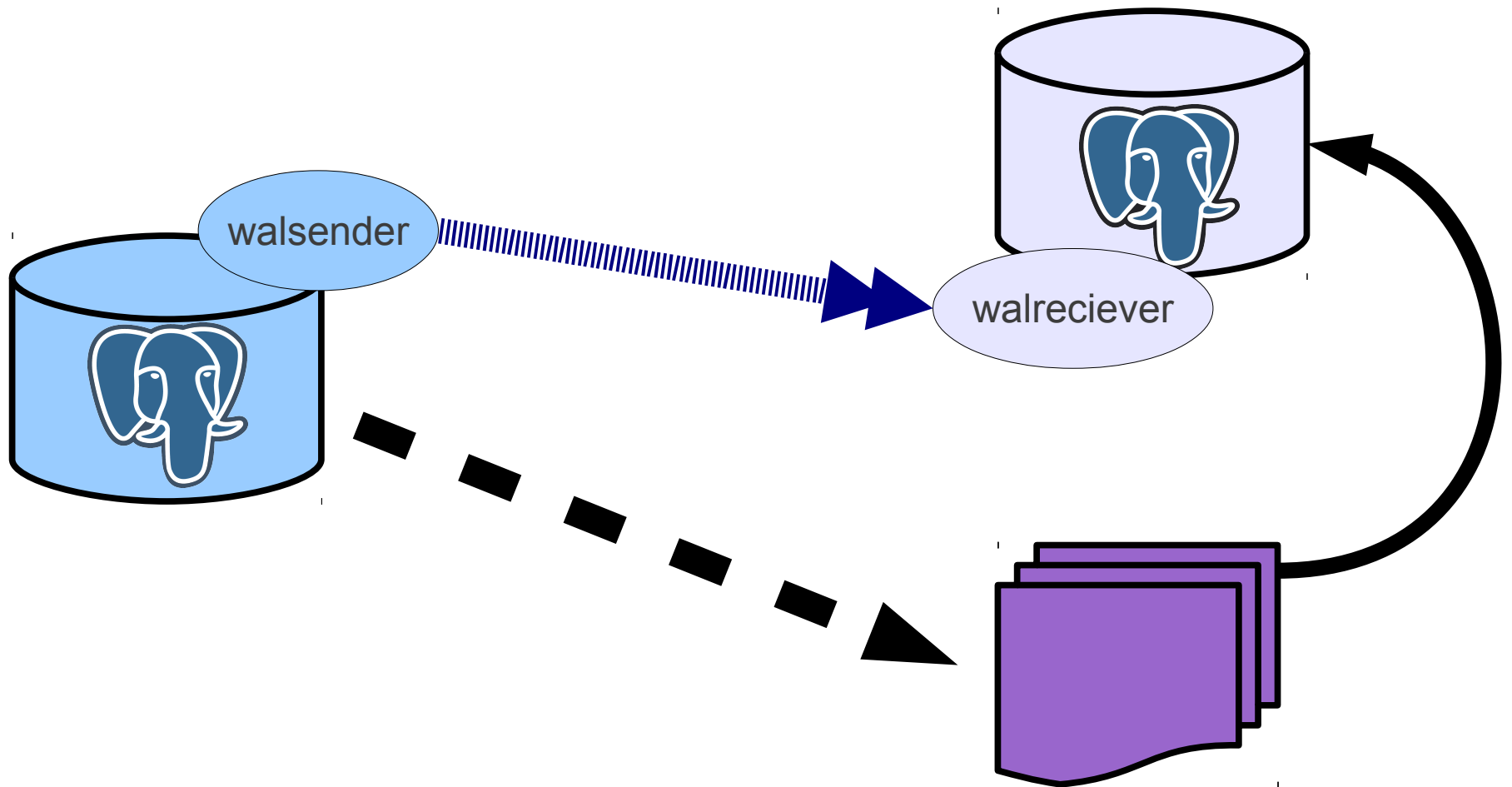
“send me  
vry old  
logfiles pls”

# replica out of sync

“scrw you!  
them iz gone”

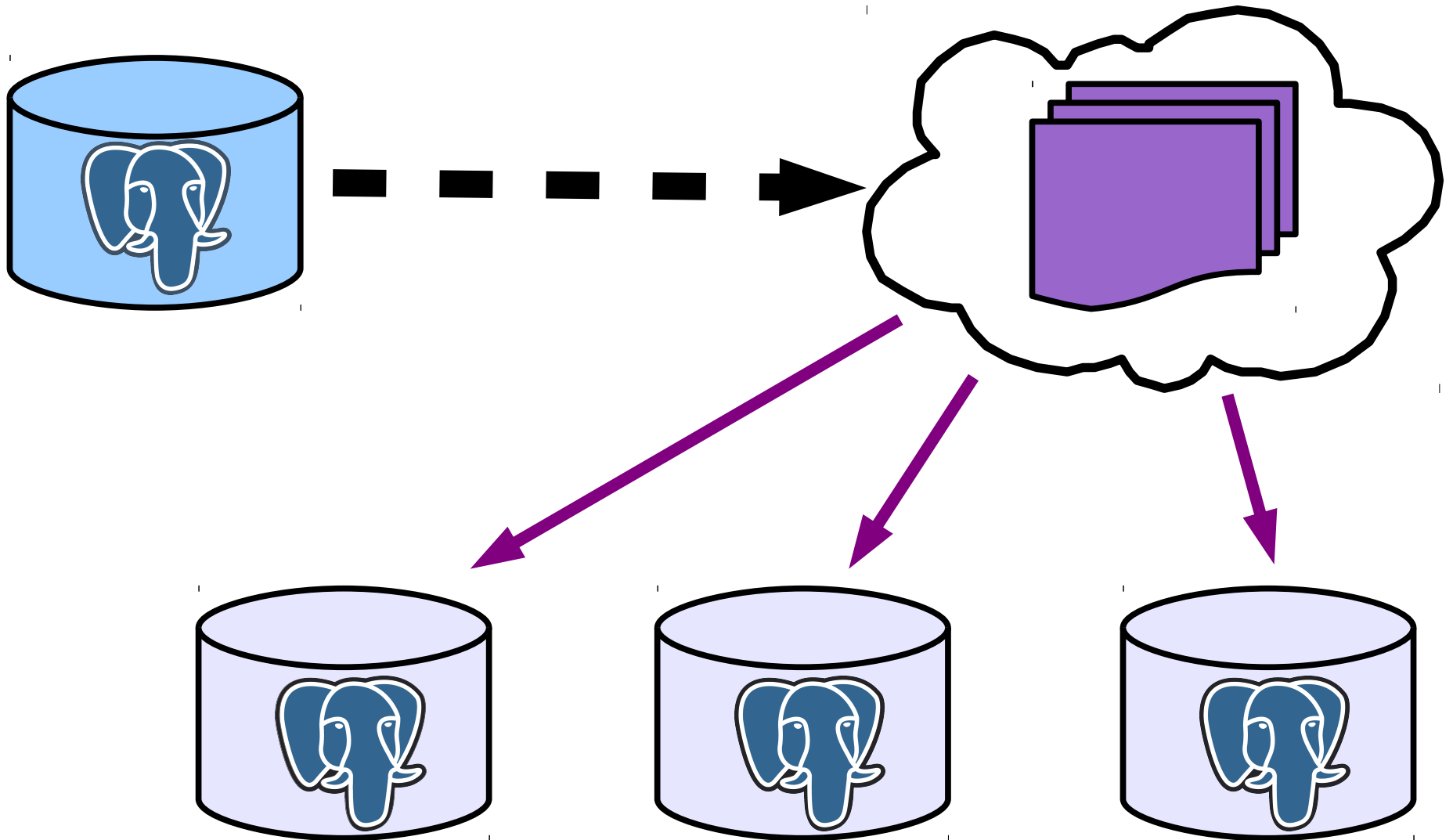


# log file backup





# log file backup: cloud



# failover vs. load-balancing

load causes replicas to fall behind

your replica can work for:

- rapid failover
- offloading work

but not both!

# lag vs. query cancel

- decide what your application can accept
  - inconsistent lagged read data
  - cancel-and-retry queries
- different settings for different replicas
  - ephemeral web replicas: short delay
  - reporting replicas: long delay
- issue common to all M-S replication

# other replication

- synchronous replication
- Slony-I
- Londiste
- Bucardo



**monitoring**

# use your favorite tool

ganglia, collectd, Hyperic, OpenNMS, OpenView, whatever ....

- nagios check\_postgres.pl
  - broad list of checks
  - mine it for queries and techniques

# many useful checks

- disk space
- caching RAM
- response time
- connections
- idle transacts
- table growth
- waiting queries
- long queries
- database size
- table bloat
- system load
- replication lag
- XID wraparound
- execution time

# OS checks

- disk space (per volume!)
- system load
- memory usage
- IO activity
- network activity



# database checks

- connections: active, idle, idle xtns
- blocked queries (number, time)
- query times (pg\_stat\_statements)
- table size & growth
- table & index bloat
- XID wraparound
- replication lag

# activity log

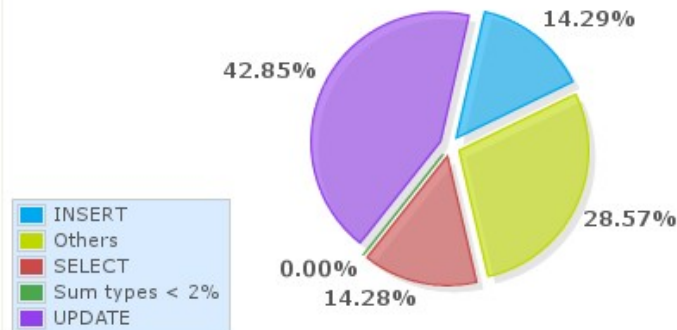
- connections & disconnections
- slow queries
- DB swap usage
- schema changes
- lock waits & deadlocks

# pgfouine, pgbadger

## Queries by type ^

Type	Count	Percentage
SELECT	48,568	14.28%
INSERT	48,578	14.29%
UPDATE	145,701	42.85%
DELETE	0	0.00%
OTHERS	97,154	28.57%

Type of queries



To Image

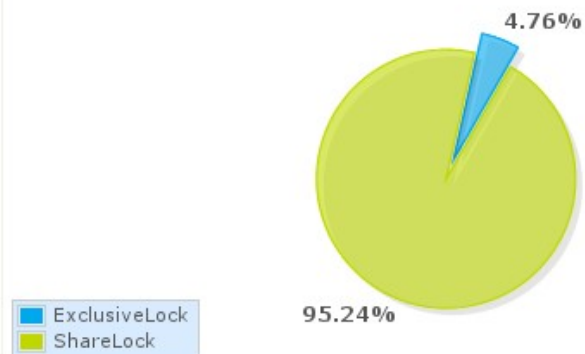
Download

Reset

## Locks by type ^

Type	Object	Count	Total Duration	Av. duration (s)
ExclusiveLock		1	1.14s	1.14s
	tuple	1	1.14s	1.14s
ShareLock		20	25.46s	1.27s
	transaction	20	25.46s	1.27s
<b>Total</b>		21	26.60s	1.27s

Type of locks



To Image

Download

Reset

# DB activity

```
<ctrl>b,n
```

```
$EDITOR /etc/postgresql/9.2/main  
/postgresql.conf
```

```
service postgresql reload
```

```
<ctrl>b,n
```

```
cd /setup/pgbench
```

```
runbench_log.sh
```

# DB activity

```
<ctrl>b,n
```

```
su - postgres
```

```
psql
```

```
select * from pg_stat_activity;
```

```
\x
```

```
select * from pg_stat_activity;
```

# DB activity

```
\c bench  
  
select * from pg_stat_user_tables;  
  
select pg_size_pretty(  
    pg_total_relation_size(  
        'pgbench_history'));  
  
select * from pg_stat{TAB}
```

# DB logs

```
exit
```

```
cd /var/log/postgresql
```

```
less activitylog-Mon.csv
```

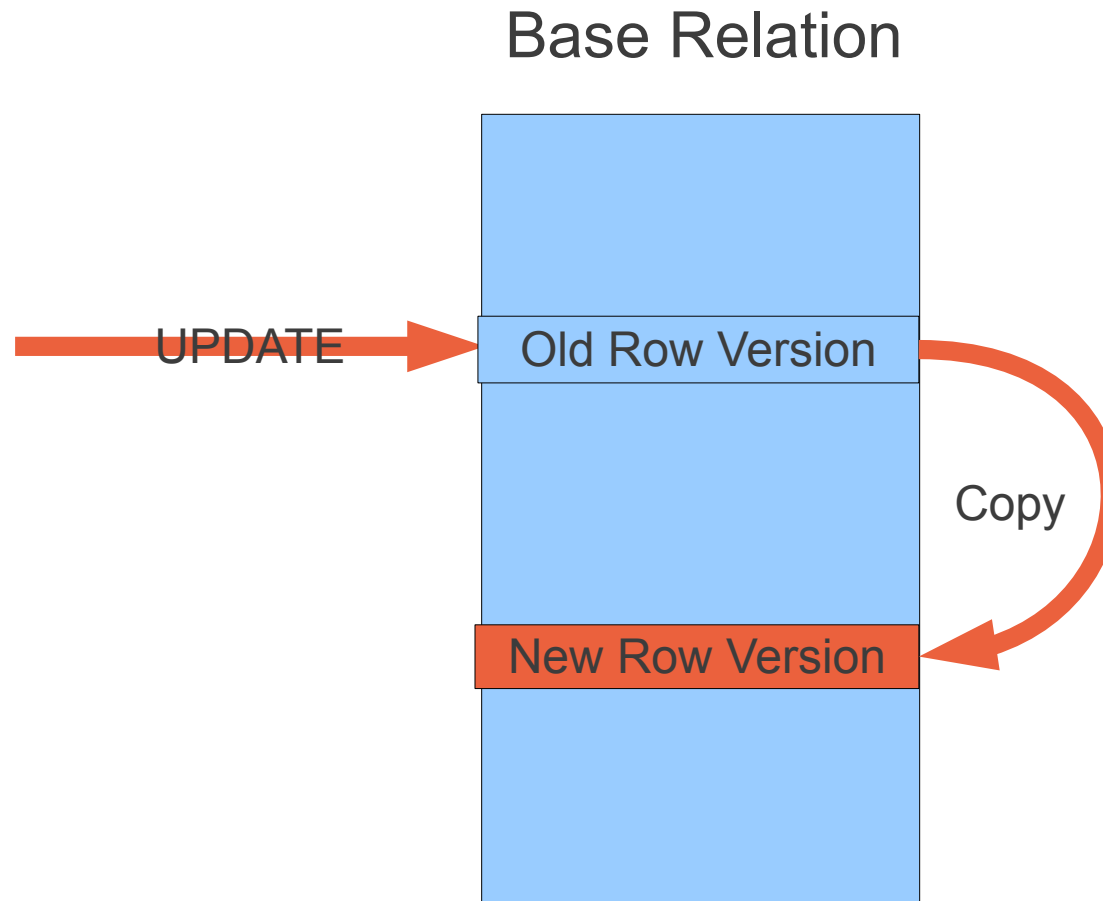
```
pgbadger --format csv --out  
    /setup/postgres/badger.html  
    activitylog-Tue.csv
```



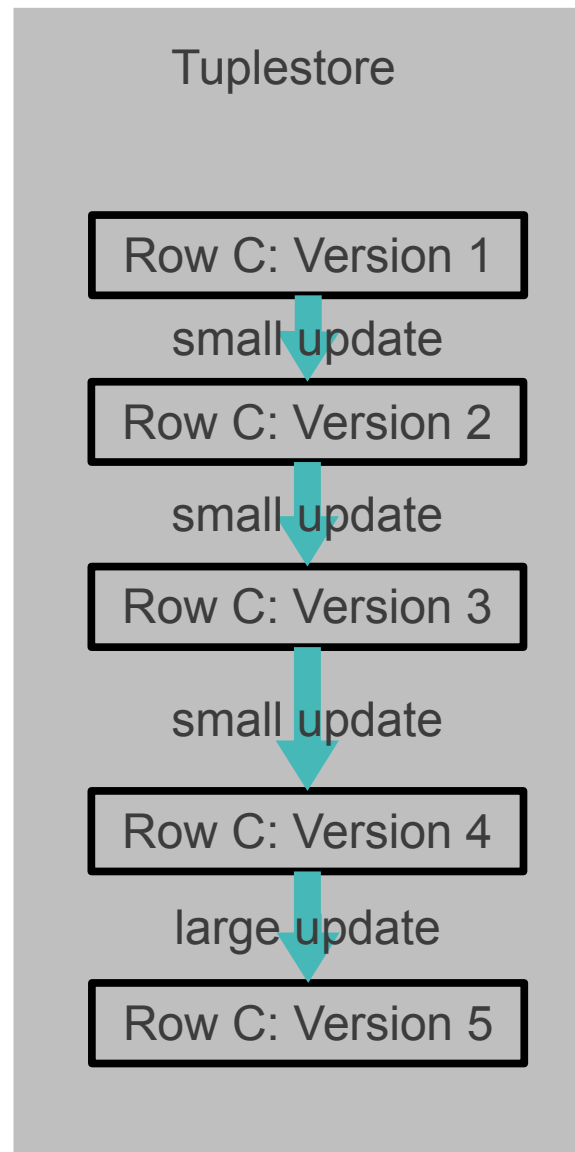
**VACUUM  
& ANALYZE**



# non-overwriting



# garbage collection



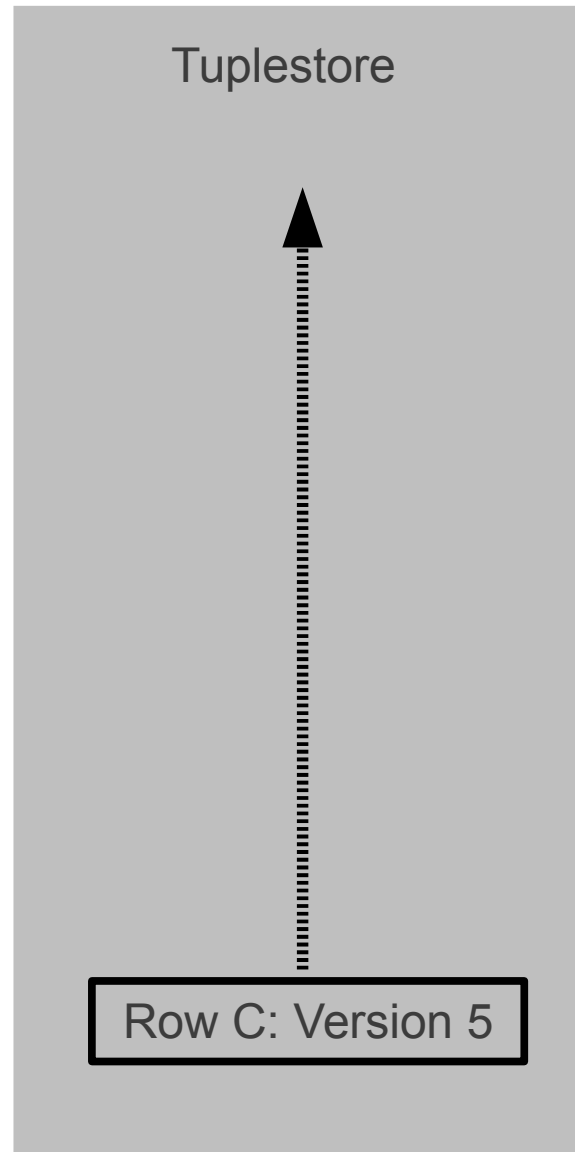
# garbage collection

Tuplestore

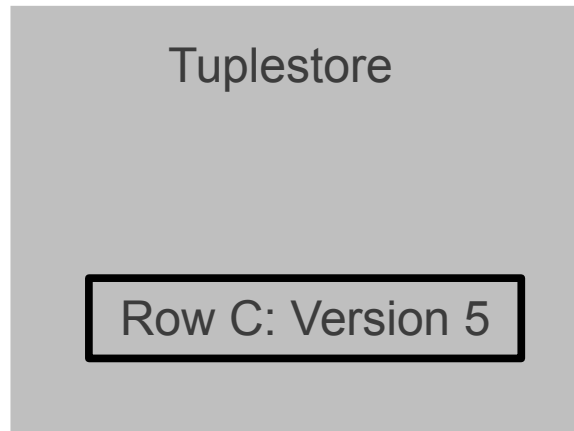
Row C: Version 5



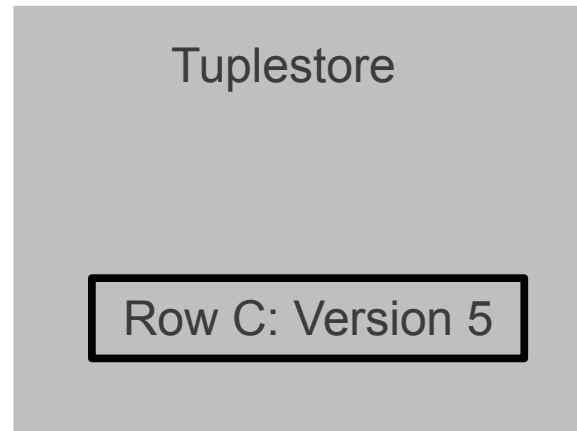
# garbage collection



# garbage collection



# garbage collection



# Autovacuum



# Multiple Autovacuum





# falling behind



# bloat check

- pg\_stat\_user\_tables:  
dead\_rows/live\_rows
- nagios bloat query
- trend table/database size
- scans very slow to return the first row

# fixing bloat

- manual VACUUMs
  - even VACUUM FULL
- pg\_reorg (bad cases)
- tune autovacuum
  - increase frequency for busy tables
  - increase workload size (vacuum\_cost\_limit)
  - more autovacuum workers

# vacuum

```
<ctrl>b,n
```

```
psql bench
```

```
\x
```

```
select * from pg_stat_user_tables;
```

```
select pg_total_relation_size(  
    'pgbench_tellers');
```

# **vacuum**

```
vacuum analyze pgbench_tellers;  
select pg_total_relation_size(  
    'pgbench_tellers');  
vacuum full pgbench_tellers;  
select pg_total_relation_size(  
    'pgbench_tellers');
```

# XID wraparound



# preempt wraparound

- track age(datfrozenxid) for each table
  - nagios has probe for this
- VACUUM FREEZE tables preemptively
- lower vacuum\_freeze\_min\_age
  - maybe to 250,000

# xid freezing

```
select relname,  
       age(relfrozenxid) as xid_age  
from pg_class JOIN  
pg_stat_user_tables USING (relname)  
order by xid_age desc;  
  
VACUUM FREEZE;  
  
select relname ...
```



# ANALYZE

- keeps DB statistics up to date
- non-intrusive
  - uses sampling
  - low IO
- should be run frequently (hourly?)
  - autovacuum takes care of this
- when not run, query plans get really bad

# ANALYZE

- manual ANALYZE when:
  - you create & populate a brand new table
  - you do a massive DELETE or TRUNCATE
  - you're seeing bad query plans due to bad stats
- ANALYZE individual tables

# stats

```
\a
```

```
select * from pg_stats where  
schemaname = 'public';
```



**managing queries**

# pg\_stat\_activity

```
-[ RECORD 2 ]-----+-----  
datid          | 16422  
datname        | libdata  
procpid        | 46295  
usesysid       | 10  
username       | dataentry  
application_name | psql  
client_addr    | 192.168.101.114  
client_port    | 5432  
backend_start  | 2012-08-26 15:09:05.233-07  
xact_start     | 2012-08-26 15:09:06.113-07  
query_start    | 2012-08-26 15:11:53.521-07  
waiting        | f  
current_query   | <IDLE> in transaction
```

# locks

- write queries can block on other write queries
  - as can table schema changes
  - queries can wait forever on locks
- look old “<IDLE> in transaction”
  - that's a zombie database connection
  - kill it kill it kill it

# killing zombies

- `pg_cancel_backend(pid)`
  - kills running queries with `sigINT`
  - like `CTRL-C`
- `pg_terminate_backend(pid)`
  - kills bad connections, idle transactions
  - can cause DB to reload in some cases

# zombie killing

```
cd /setup/pgbench  
runbench_locks.sh  
<ctrl>b,n  
su - postgres  
psql bench  
select * from pg_stat_activity;
```



# zombie killing

```
select pg_cancel_backend(pid)
from pg_stat_activity
where state = 'idle in transaction'
and state_change <
( now() - interval '10 seconds');
```

```
select * from pg_stat_activity;
```

# EXPLAIN

```
Nested Loop (cost=792.00..828.08 rows=1422317 width=99)
  -> HashAggregate (cost=792.00..792.00 rows=1 width=4)
    -> Index Scan using
index_player_summaries_on_player_id on player_summaries
ps (cost=0.00..791.80 rows=403 width=4)
    Index Cond: (player_id = 21432312)
  -> Index Scan using index_player_summaries_on_match_id
on player_summaries (cost=0.00..33.98 rows=600 width=99)
    Index Cond: (match_id = ps.match_id)
```

# EXPLAIN ANALYZE

```
Nested Loop  (cost=792.00..828.08 rows=1422317
width=99)  (actual time=9928.869..20753.723
rows=13470 loops=1)
```

```
    -> HashAggregate  (cost=792.00..792.00 rows=1 width=4)
    (actual time=9895.105..9897.096 rows=1347 loops=1)
```

```
        -> Index Scan using
```

```
index_player_summaries_on_player_id on player_summaries
ps  (cost=0.00..791.80 rows=403 width=4)  (actual
time=27.413..9890.887 rows=1347 loops=1)
```

```
            Index Cond: (player_id = 21432312)
```

```
    -> Index Scan using index_player_summaries_on_match_id
on player_summaries  (cost=0.00..33.98 rows=600 width=99)
    (actual time=7.375..8.037 rows=10 loops=1347)
```

```
            Index Cond: (match_id = ps.match_id)
```

```
Total runtime: 20764.371 ms"
```

# explain.depesz.com

options

HTML TEXT STATS

exclusive	inclusive	rows x	rows	loops	node
30.788	20753.723	↑ 105.6	13470	1	→ Nested Loop (cost=792.00..828.08 rows=1422317 width=99) (actual time=9928.869..20753.723 rows=13470 loops=1)
6.209	9897.096	↓ 1347.0	1347	1	→ HashAggregate (cost=792.00..792.00 rows=1 width=4) (actual time=9895.105..9897.096 rows=1347 loops=1)
9890.887	9890.887	↓ 3.3	1347	1	→ Index Scan using index_player_summaries_on_player_id on player_summaries ps (cost=0.00..791.80 rows=403 width=4) (actual time=27.413..9890.887 rows=1347 loops=1) Index Cond: (player_id = 21432312)
10825.839	10825.839	↑ 60.0	10	1347	→ Index Scan using index_player_summaries_on_match_id on player_summaries (cost=0.00..33.98 rows=600 width=99) (actual time=7.375..8.037 rows=10 loops=1347) Index Cond: (match_id = ps.match_id)

# explain.depesz.com

options

HTML TEXT STATS

exclusive	inclusive	rows x	rows	loops	node
30.788	20753.723	↑ 105.6	13470	1	→ Nested Loop (cost=792.00..828.08 rows=1422317 width=99) (actual time=9928.869..20753.723 rows=13470 loops=1)
6.209	9897.096	↓ 1347.0	1347	1	→ HashAggregate (cost=792.00..792.00 rows=1 width=4) (actual time=9895.105..9897.096 rows=1347 loops=1)
9890.887	9890.887	↓ 3.3	1347	1	→ Index Scan using index_player_summaries_on_player_id on player_summaries ps (cost=0.00..791.80 rows=403 width=4) (actual time=27.413..9890.887 rows=1347 loops=1) Index Cond: (player_id = 21432312)
10825.839	10825.839	↑ 60.0	10	1347	→ Index Scan using index_player_summaries_on_match_id on player_summaries (cost=0.00..33.98 rows=600 width=99) (actual time=7.375..8.037 rows=10 loops=1347) Index Cond: (match_id = ps.match_id)

# what to look for

- “seq scan” on large table
  - maybe index needed
- cartesian joins
- really bad row estimates
  - ANALYZE needed?

# explain analyze

```
explain select count(*) from loans  
where checkout_date between  
'2011-01-01' and '2011-03-31';
```

```
explain analyze ...
```

```
explain ( analyze on, buffers  
on ) ...
```

```
explain ( analyze on, format  
yaml ) ...
```

# explain analyze

```
\i /setup/postgres/  
explain_quarterly_report.sql
```



# questions?

- [www.pgexperts.com/tutorials.html](http://www.pgexperts.com/tutorials.html)
- Josh Berkus: [josh@pgexperts.com](mailto:josh@pgexperts.com)
  - PGX: [www.pgexperts.com](http://www.pgexperts.com)
  - Blog: [www.databasesoup.com](http://www.databasesoup.com)
- Upcoming Events
  - Postgres Open: Sept. 17-19, Chicago
  - pgConf.EU: Oct. 29 – Nov. 1, Dublin



Copyright 2013 PostgreSQL Experts Inc. Released under the Creative Commons Attribution License. All images are the property of their respective owners. The Don't Panic slogan and logo is property of the BBC, and the Dilbert image belongs to Scott Adams and is used here as parody.

