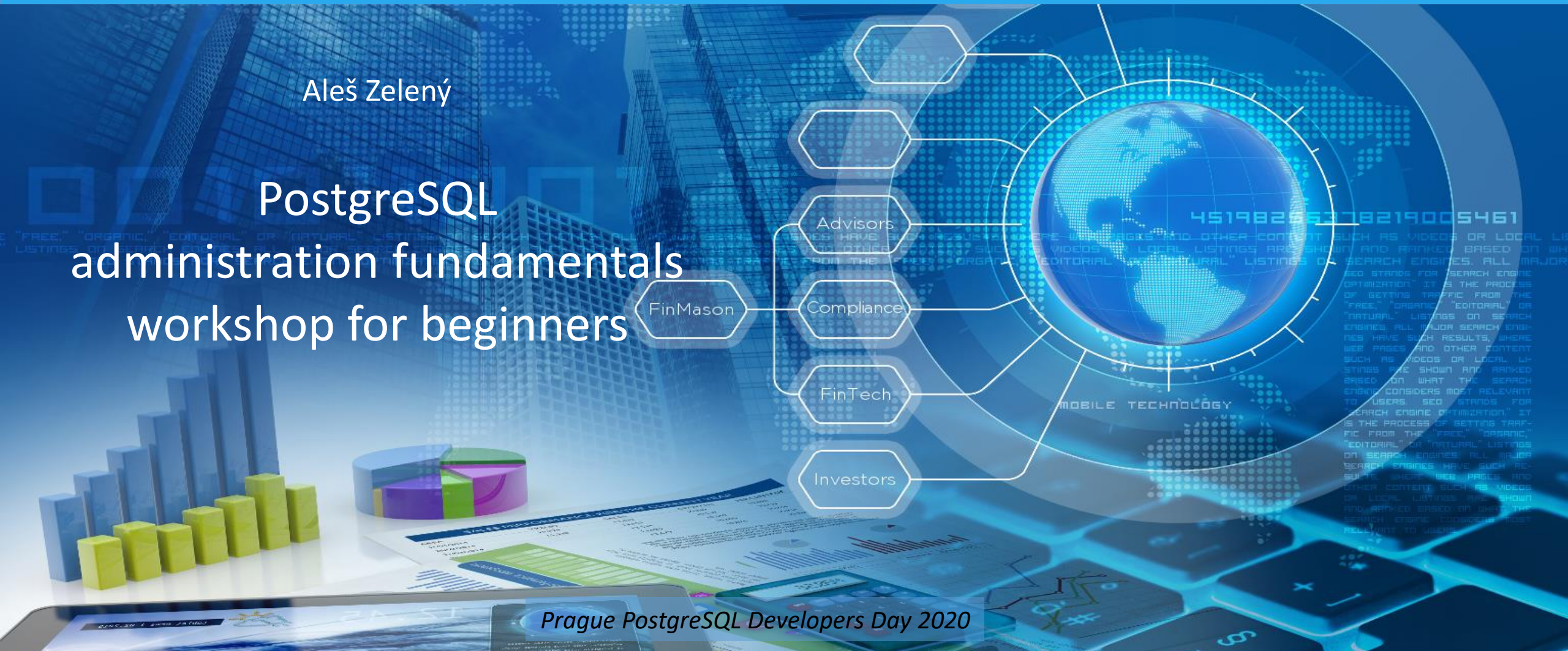
finmason



Aleš Zelený

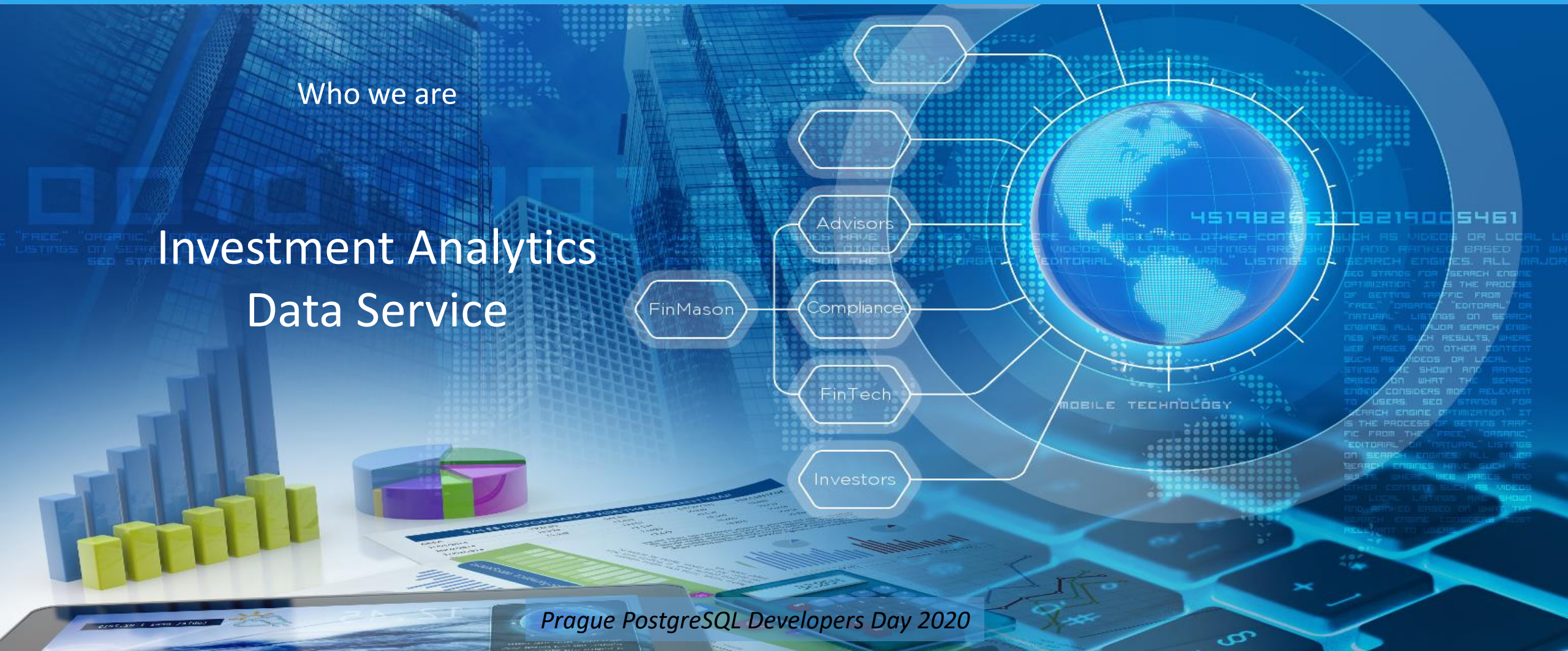
PostgreSQL administration fundamentals workshop for beginners

Prague PostgreSQL Developers Day 2020

finmason

Who we are

Investment Analytics Data Service



Prague PostgreSQL Developers Day 2020

finmason

Agenda

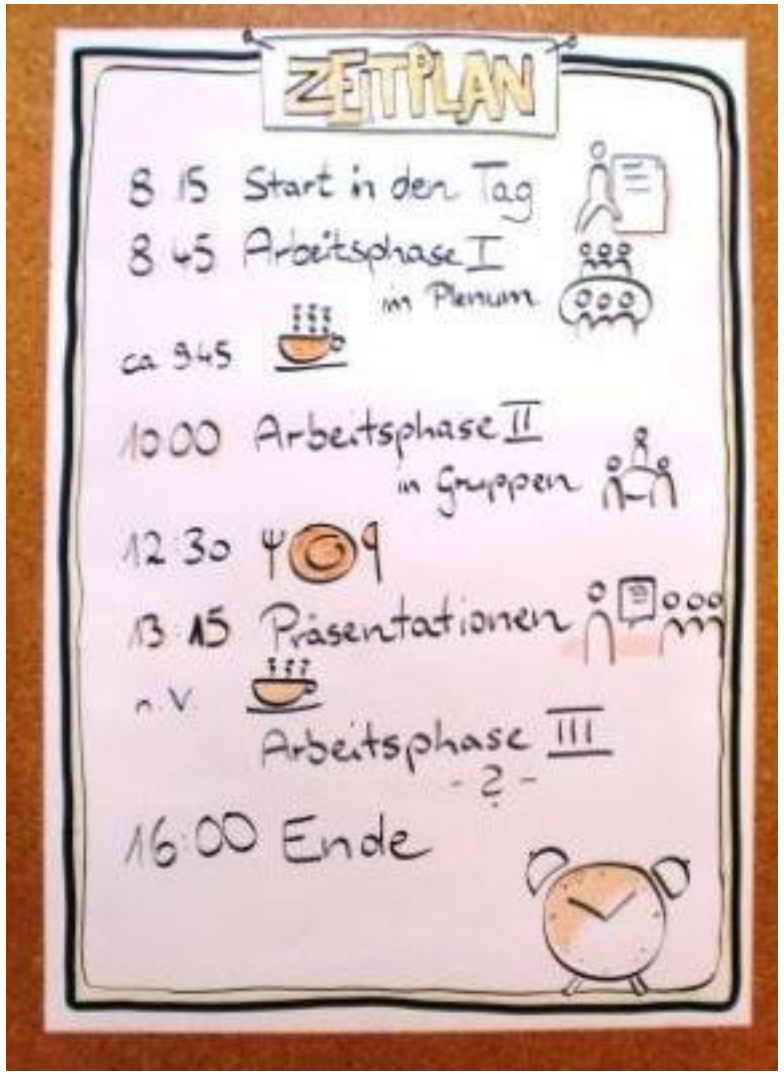
Architecture minimum

PostgreSQL installation

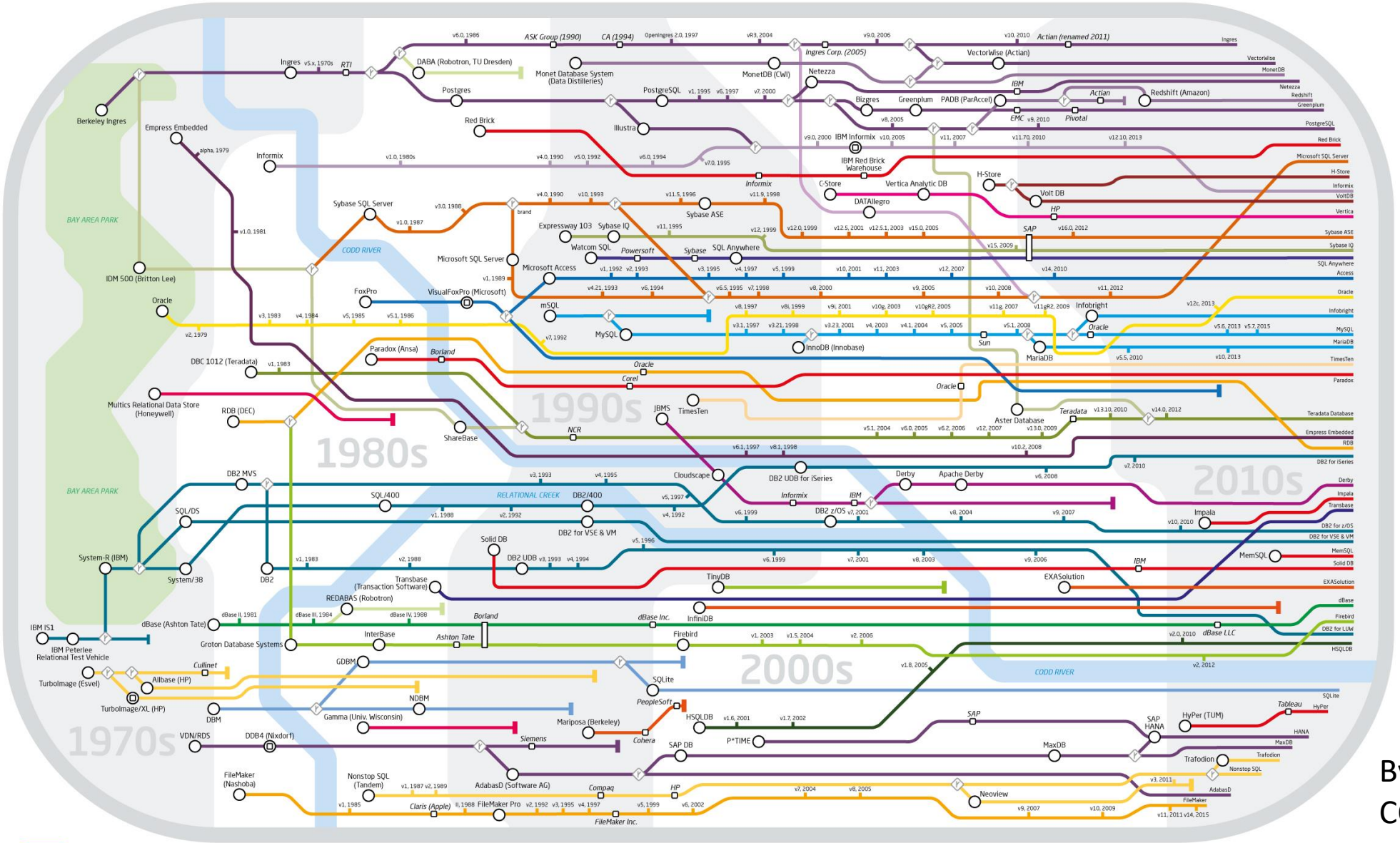
First steps after installation

PostgreSQL instance configuration

Basic DBA tasks



Genealogy of Relational Database Management Systems

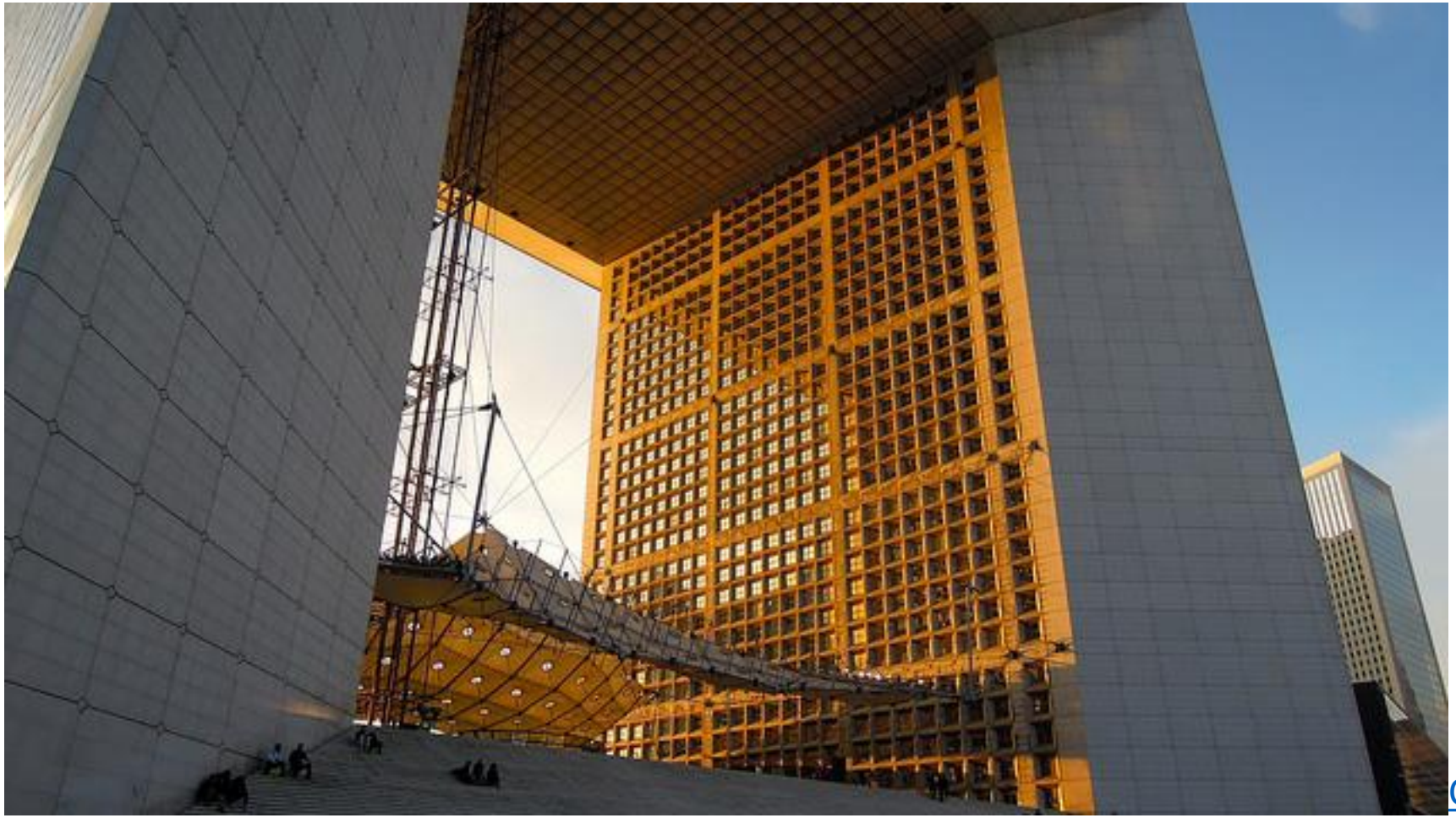


By HPI
CC BY-SA 3.0

Felix Naumann, Jana Bauckmann, Claudia Exeler, Jan-Peer Rudolph, Fabian Tschirschnitz
Contact - Hasso Plattner Institut, University of Potsdam, felix.naumann@hpi.de
Design - Alexander Sandt Grafik-Design, Hamburg
Version 6.0 - October 2018
https://hpi.de/naumann/projects/rdbms-genealogy.html

finmason

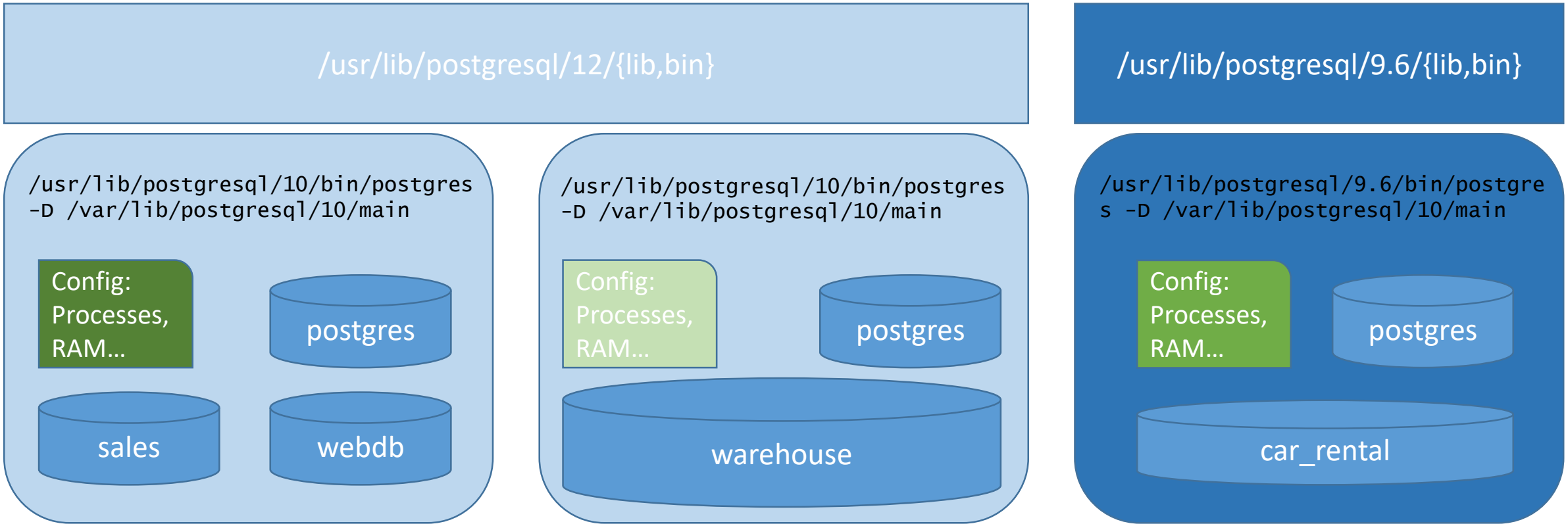
Architecture…



finmason

PostgreSQL in operating system

- RDBMS SW installed -> Postgres instances -> databases managed by instance.



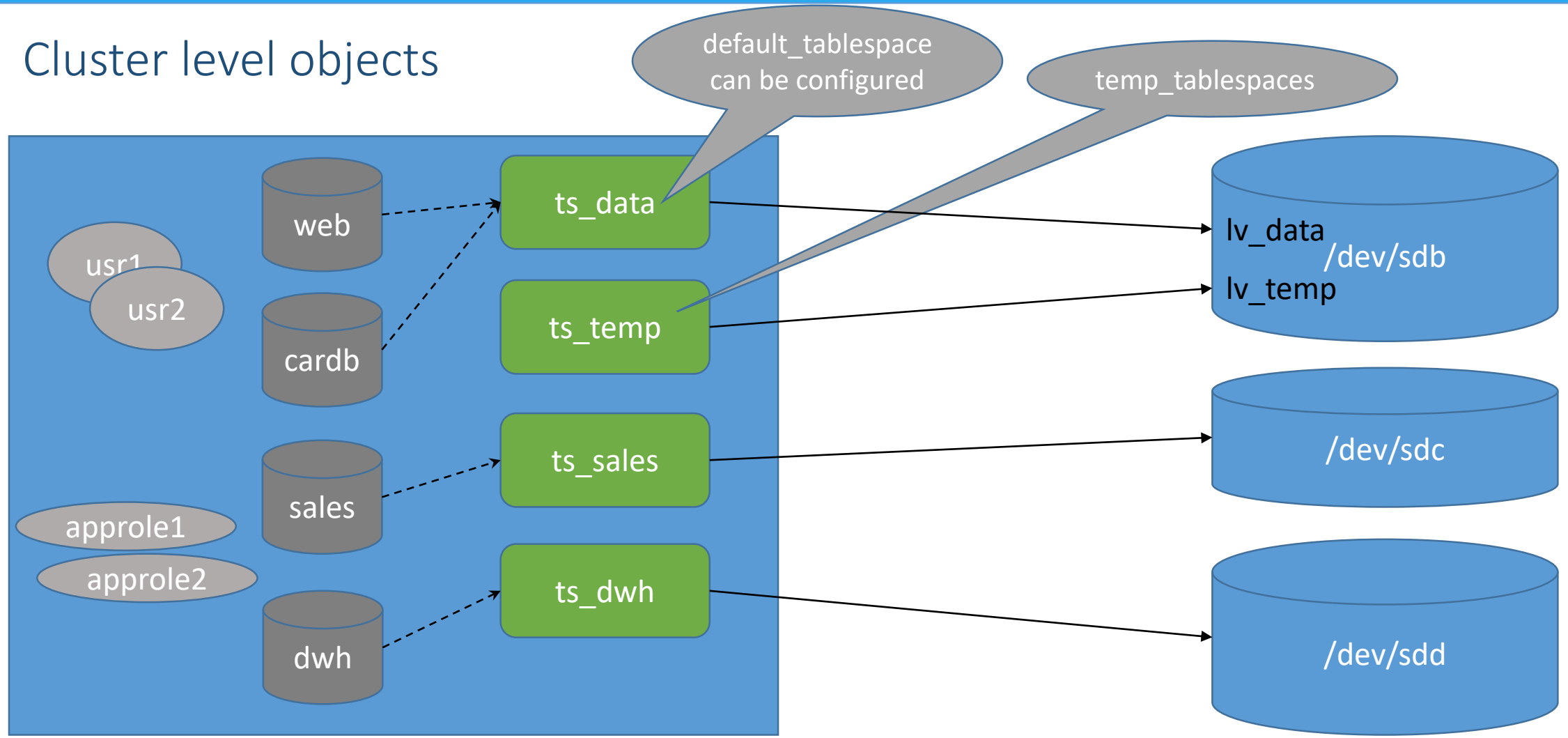
PostgreSQL cluster on filesystem

- Filesystem layout advise on wiki
  - Consider using tablespaces per storage tier or per usecase
    - Storage tier
    - Database
    - Customer (kind of multitenant separation)
  - WAL – transaction logs on separate file system
    - Keep enough free space, as WAL logs (formerly called xlog-s) might growth under workload peaks or in case DB replication target is offline
  - PostgreSQL log files – place them on filesystem distinct from postgres cluster data directory, if they growth unexpectedly fast (some repeated errors or flood by failed login attempts), they will not runt your database cluster filesystem out of space. Rotate them and keep for long time, in case of an investigation, they are usefull.
- Do not share filesystem for different clusters
  - Unless working on a LAB demo…

PostgreSQL cluster level objects

- Configuration (postgresql.conf – can be queried from `pg_settings` view)
- HBA rules (from postgres 10 there is also `pg_hba_file_rules` view)
- Databases (templates and postgres DBs are created by initdb utility during cluster creation)
  - template0
  - Template1 (can be modified, so all new databases will contain your modifications, like pre-installed extensions)
  - postgres (can be used for user data, I'd recommend create your own databases to store application data)
  - Your custom databases – this is probably why you are running aRDBMS.
- Roles (and users whose are roles with login privilege)
- Tablespaces (can point to different filesystems, so you can utilize multiple disks)
- Write Ahead Logs (transaction logs)
- Physical backups
- … and many others

Cluster level objects



Single database can have tables in different tablespaces

Database level objects

- Schemas
  - Tables
    - Tables within a schema might be placed in different tablespaces for performance tiering or space management purposes
  - Functions
  - Triggers and trigger functions
  - Materialized views
  - Operators
  - …
- Object privileges
- search_path
  - default is configured at cluster level
  - Can be configured per user in each database

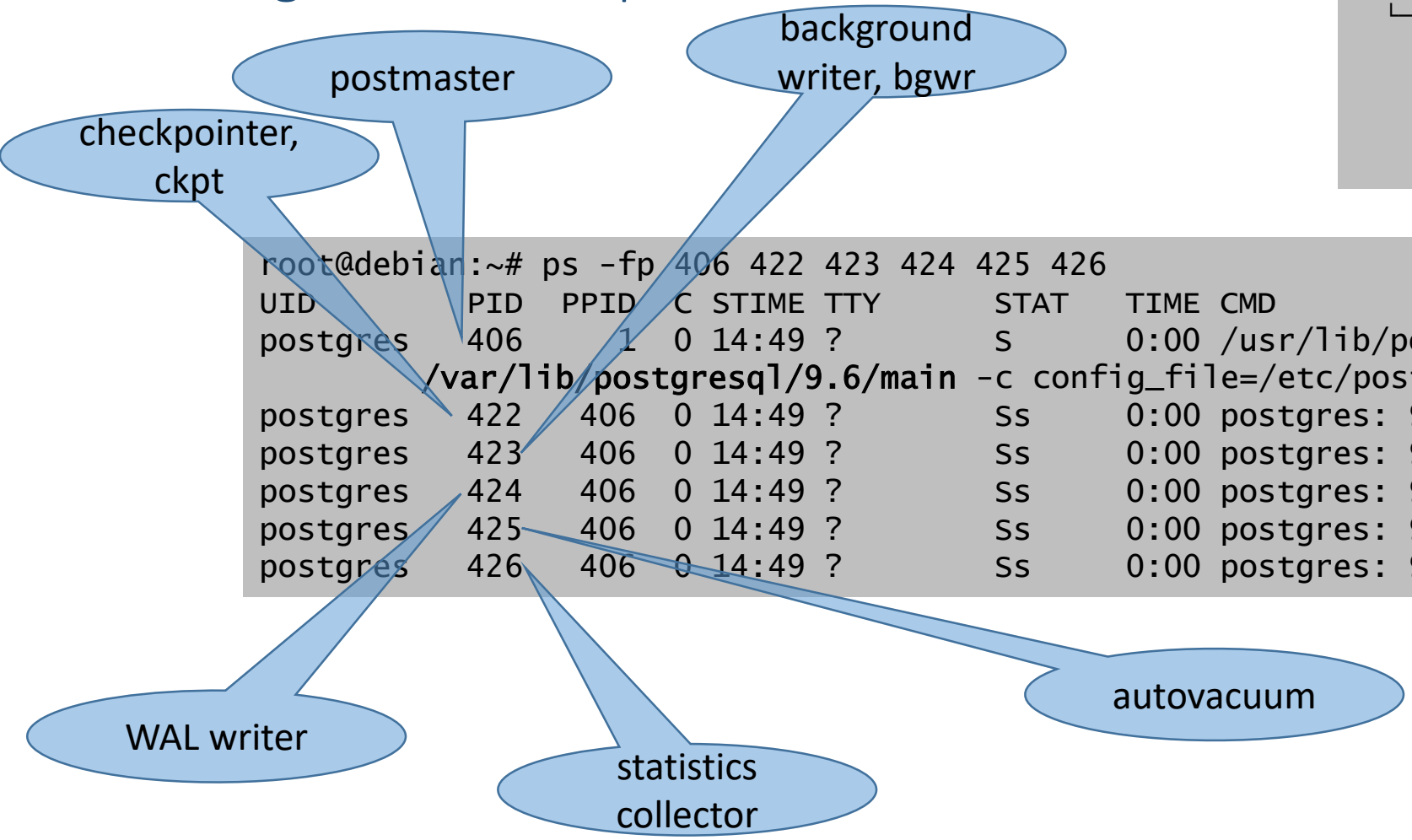
Postgres instance processes

- postmaster is a deprecated alias of postgres ☺
  - responsible for instance start, more on this later
  - responsible to handle new client connections
- checkpointer - responsible for checkpoint – consistent state of all files as checkpoint (recovery starts from last ckpt)
- background writer - writes dirty blocks from shared buffers to data files
- wal writer
  - writes data to transaction logs, necessary to ensure durability and recoverability
- autovaccum - database housekeeping process
- statistic collector
- background worker processes

Postgres instance processes

```
root@debian:~# pstree -laps 406
systemd,1
  └─postgres,406 -D /var/lib/postgresql/9.6/main …
      ├─postgres,422
      ├─postgres,423
      ├─postgres,424
      ├─postgres,425
      └─postgres,426
```

```
root@debian:~# ps -fp 406 422 423 424 425 426
UID        PID  PPID  C STIME TTY          STAT   TIME CMD
postgres   406     1  0 14:49 ?            S      0:00 /usr/lib/postgresql/9.6/bin/postgres -D
           /var/lib/postgresql/9.6/main -c config_file=/etc/postgresql/9.6/main/postgresql.conf
postgres   422   406  0 14:49 ?            Ss     0:00 postgres: 9.6/main: checkpointer process
postgres   423   406  0 14:49 ?            Ss     0:00 postgres: 9.6/main: writer process
postgres   424   406  0 14:49 ?            Ss     0:00 postgres: 9.6/main: wal writer process
postgres   425   406  0 14:49 ?            Ss     0:00 postgres: 9.6/main: autovacuum launcher process
postgres   426   406  0 14:49 ?            Ss     0:00 postgres: 9.6/main: stats collector process
```



finmason

Postgres instance processes (release 10)

bgworker – background worker process, starting from postgres release 10 used to provide logical replication feature, some parallel operations (sequential scan…) are supported, see max_worker_processes configuration parameter. Extension modules might create bgworker processes even in previous releases.

```
root@debian:~# pstree -laps 405
systemd,1
  └─postgres,405 -D /var/lib/postgresql/10/warehouse
      ├─postgres,410
      ├─postgres,411
      ├─postgres,412
      ├─postgres,413
      ├─postgres,414
      └─postgres,415
```

```
root@debian:~# ps -fp 405 410 411 412 413 414 415
UID        PID  PPID  C STIME TTY      STAT   TIME CMD
postgres   405     1  0 14:49 ?        S      0:00 /usr/lib/postgresql/10/bin/postgres -D
        /var/lib/postgresql/10/warehouse -c
config_file=/etc/postgresql/10/warehouse/postgresql.con
postgres   410   405  0 14:49 ?        Ss     0:00 postgres: 10/warehouse: checkpointer process
postgres   411   405  0 14:49 ?        Ss     0:00 postgres: 10/warehouse: writer process
postgres   412   405  0 14:49 ?        Ss     0:00 postgres: 10/warehouse: wal writer process
postgres   413   405  0 14:49 ?        Ss     0:00 postgres: 10/warehouse: autovacuum launcher
process
postgres   414   405  0 14:49 ?        Ss     0:00 postgres: 10/warehouse: stats collector process
postgres   415   405  0 14:49 ?        Ss     0:00 postgres: 10/warehouse: bgworker: logical
        replication launcher
```

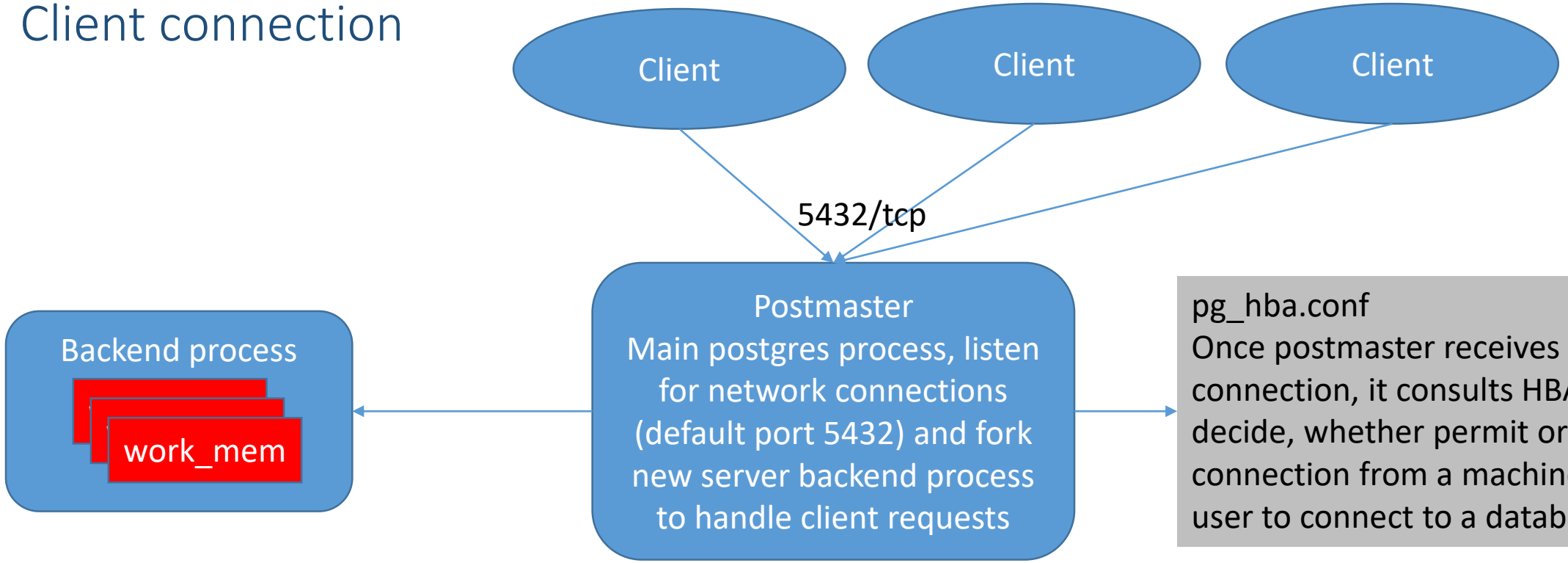
finmason

Client connection

- Client need to know connection string or Environment Variables – recipe how contact server
  - Database server host (or IP address)
  - Username (and password) used for connection
  - Database where client wants to connect, there is no connection to postgres instance only
- Keyword=value format
  - host=localhost port=5432 dbname=mydb connect_timeout=10
- URI format
  - postgresql://[user[:password]@][netloc][:port][,...][/dbname][?param1=value1&...]
    - postgresql://user:secret@localhost
    - postgresql://other@localhost/otherdb?connect_timeout=10&application_name=myapp
    - postgresql://host1:123,host2:456/somedb?target_session_attrs=any&application_name=myapp

from release 10, first successful is used

Client connection



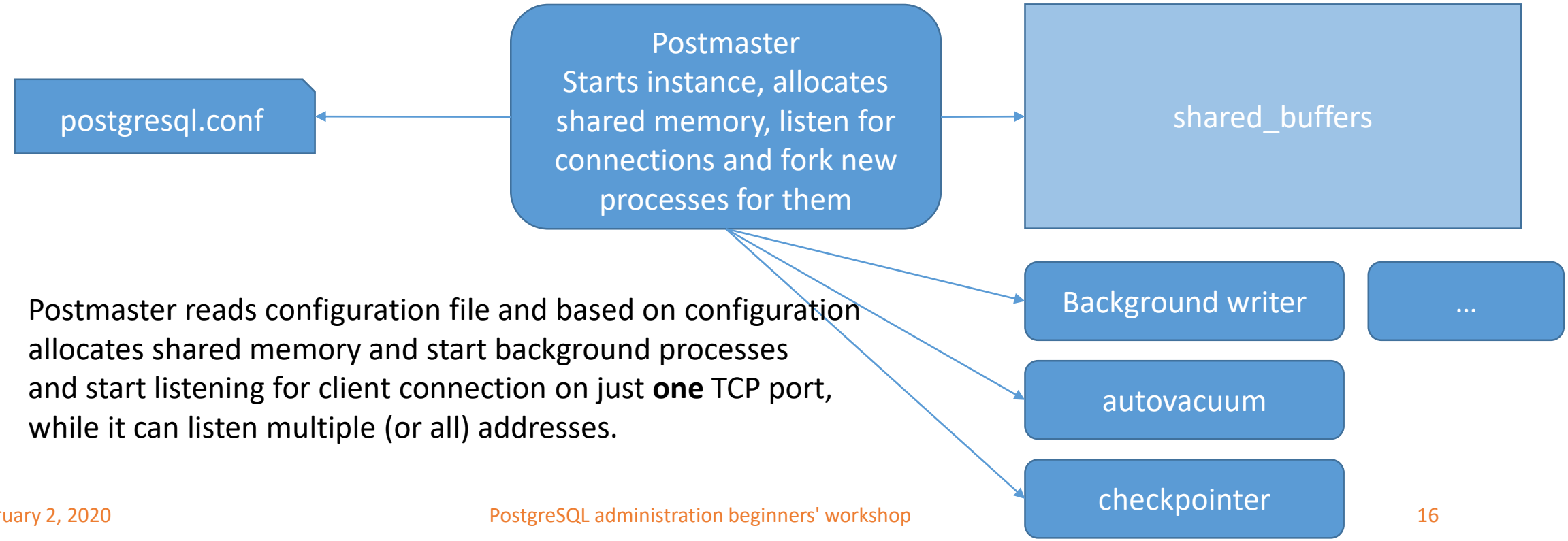

```
local   all             all                                     peer
host    all             all             127.0.0.1/32            ident
# Subnet range 192.168.151.[0 - 63]
host    all             all             192.168.151.0/26        md5
host    all             all             192.168.151.64/32       md5
```

Postmaster starts postgresql instance

1

systemd/init/user start postgres process at least with with one parameter or environment set…

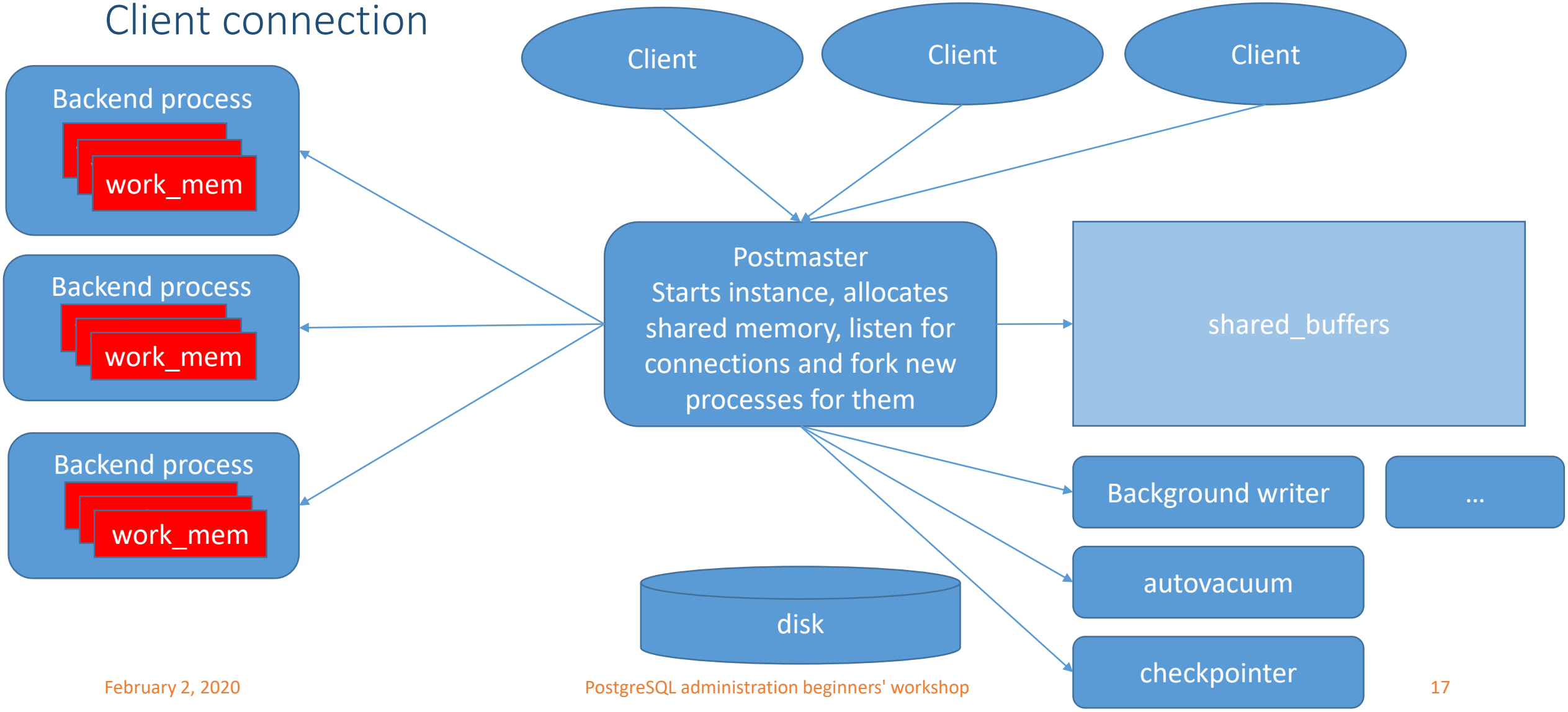
```
postgres@debian:~$ /usr/lib/postgresql/10/bin/postgres
postgres does not know where to find the server configuration file.
You must specify the --config-file or -D invocation option or set the PGDATA environment variable.
```



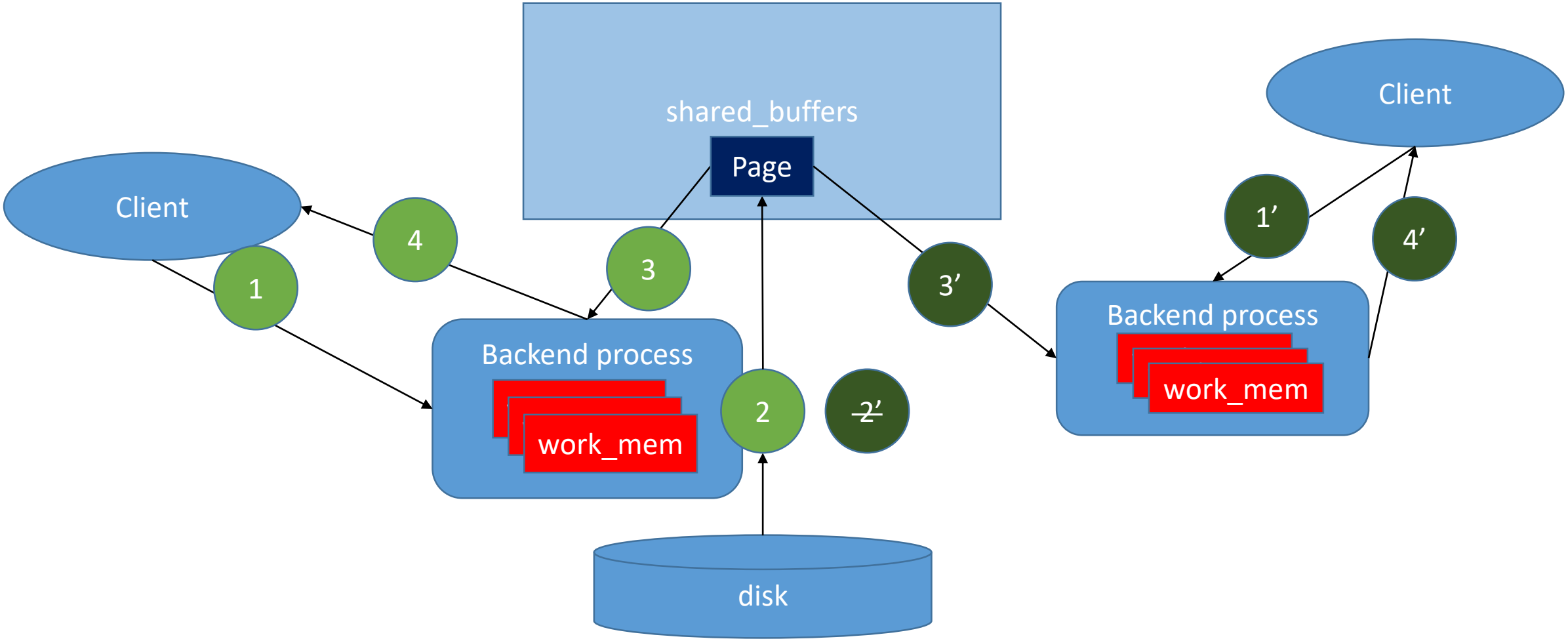
2

Postmaster reads configuration file and based on configuration allocates shared memory and start background processes and start listening for client connection on just **one** TCP port, while it can listen multiple (or all) addresses.

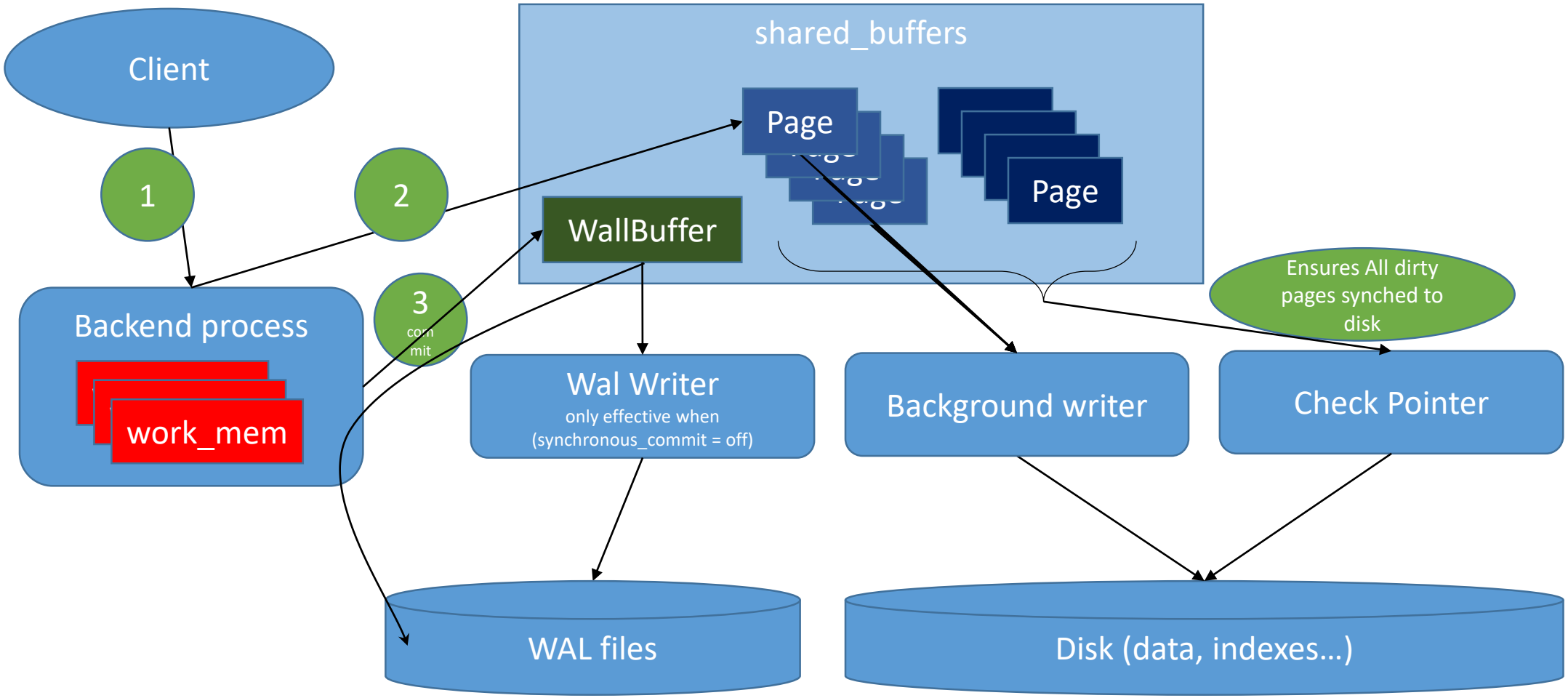
Client connection



Shared buffers usage for data reads

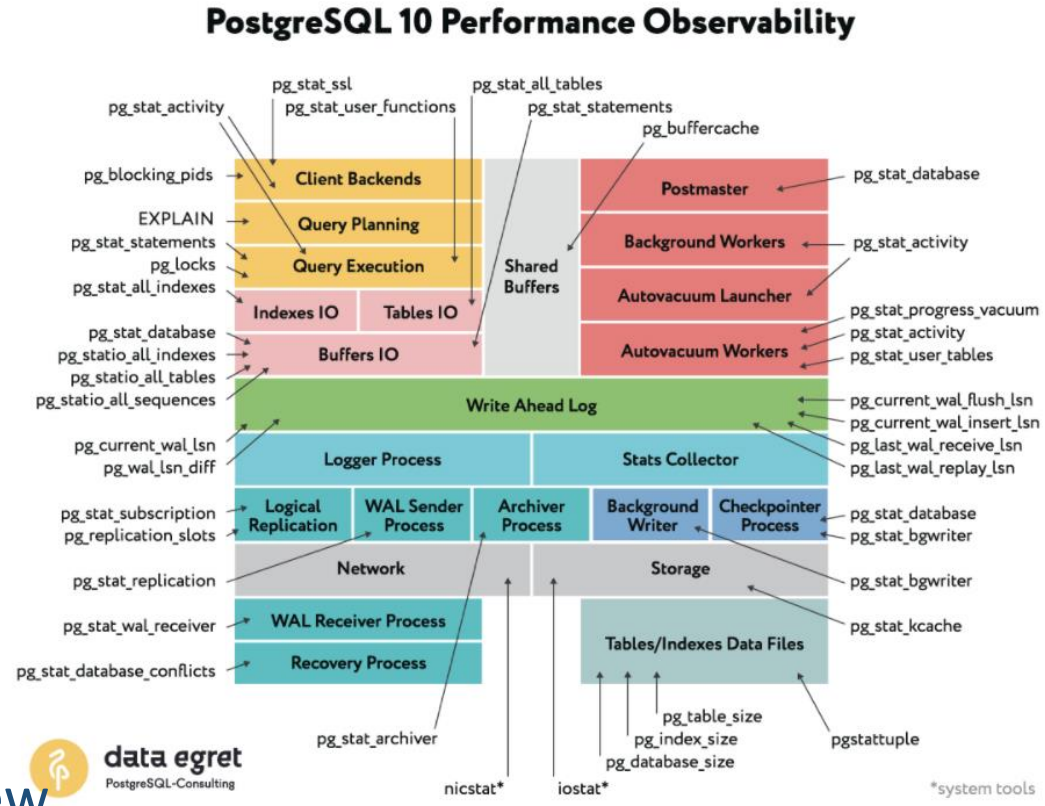


Shared buffers usage for data changes

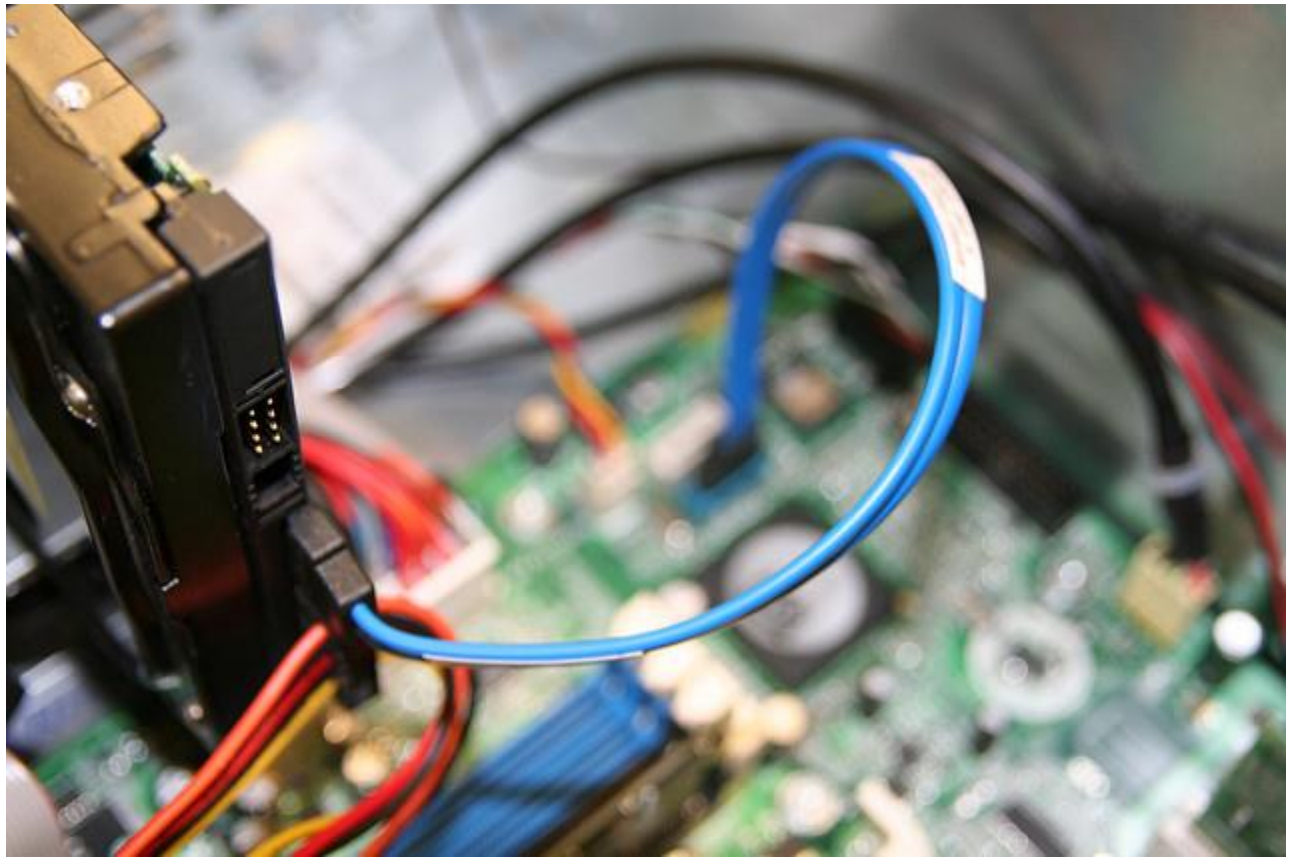


Shared memory, processes, lots of information

- Understand internal views and processes
  - PostgreSQL workings in one picture by Alexey Lesovsky, © DataEgret.com | All rights reserved
- Browse through documentation
- pg_stat_statements
  - “must have” extension from contributed module
- Do not forget using OS tools
- Postgres log file is also very valuable source of information
- All the information together provides good view what is going on with your postgres instance



PostgreSQL installation



PostgreSQL installation

- Linux setup from packaging system
  - PGDG – Postgres Global Development Group (latest version 12 at time of slides writing)
  - RHEL/FEDORA
    - Install PGDG repository and RDBMS packages using yum or dnf utilities
    - Step by step guide on Linux downloads (Red Hat family)
  - Debian/Ubuntu
    - Install PGDG repository and RDBMS packages using apt/apt-get utility
    - Step by step guide for Linux downloads (Debian)
    - Step by step guide for Linux downloads (Ubuntu)
  - SuSE
    - Step by step guide for Linux downloads (SuSE)
  - Interactive installer by EnterpriseDB
    - Linux, Mac OS X, Windows

PostgreSQL installation – packages to install on server

- Fedora (RHEL, CentOS)
  - postgresql12-server
    - RDBMS server
  - postgresql12-contrib
    - “must have” additional supplied modules
  - postgresql12
    - Client programs and libraries
  - postgresql12-server-debuginfo-12.1
    - Optional debug symbols (see perf utility), repository is disabled in default installation (`/etc/yum.repos.d/pgdg-12-fedora.repo`)
  - utility
    - pspg, pgadmin
- host names used in examples
  - fedora
- Debian, Ubuntu (9.6 example)
  - postgresql-9.6
    - RDBMS server
  - postgresql-contrib-9.6
    - additional supplied modules
  - postgresql-client-9.6
    - client libraries and client binaries
  - postgresql-9.6-dbg
    - Optional debug symbols (see perf utility)
  - utility
    - pspg, pgadmin
- host names used in examples
  - debian, ubuntu

Checks before installation (TCP port 5432)

- Is default PostgreSQL TCP port used (can we use default port)?

```
root@debian:~# lsof -Pni :5432 || echo "There is no process listening on port 5432"
There is no process listening on port 5432
```

- Server connectivity to port 5432 (or any other we want use for Postgres)
  - On DB server run netcat utility on desired port (without IP nc will listen on all interfaces)

```
root@debian:~# nc -vl -p 5432
listening on [any] 5432 ...
root@debian:~# nc -vln -s 192.168.151.16 -p 5432
listening on [192.168.151.16] 5432 ...
```

  - Check connection from remote client to DB server

```
root@ubuntu:~# nc -vzn 192.168.151.16 5432
Connection to 192.168.151.16 5432 port [tcp/*] succeeded!
```

finmason

Based on OS defaults, test might prove an action is needed…

- Server connectivity to port 5432 (or any other we want use for Postgres)
  - On DB server run netcat on desired port (example fedora server)

```
root@fedora:~# nc -vln 192.168.151.48 5432
```

  - Chek connection from remote client

```
root@ubuntu:~# nc -vzn 192.168.151.48 5432
Trying 192.168.151.48...
nc: connect to 192.168.151.48 port 5432 (tcp) failed: No route to host
root@ubuntu:~# ping -c 1 fedora
PING fedora (192.168.151.48) 56(84) bytes of data.
64 bytes from fedora (192.168.151.48): icmp_seq=1 ttl=64 time=0.305 ms

--- fedora ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.305/0.305/0.305/0.000 ms
```

finmason

Tweak firewall rules if needed…

- Fedora database server example

```
[root@fedora ~]# firewall-cmd --add-port=5432/tcp --permanent
Success
```

or

```
[root@fedora ~]# firewall-cmd --add-service=postgresql --permanent
Success
```

And next because of permanent option, reload is needed

```
[root@fedora ~]# firewall-cmd --reload
Success
root@fedora:~# nc -vln 192.168.151.48 5432
Ncat: Version 7.60 ( https://nmap.org/ncat )
Ncat: Generating a temporary 1024-bit RSA key. Use --ssl-key and --ssl-cert to use a permanent one.
Ncat: SHA-1 fingerprint: 262A 5B2C CE9E BE18 3CB3 2608 34A0 5D5F F92C E096
Ncat: Listening on 192.168.151.64:5432
```

- Chek connection from remote host after firewall reload

```
root@ubuntu:~# nc -vzn 192.168.151.64 5432
Connection to 192.168.151.64 5432 port [tcp/*] succeeded!
```

finmason

Fedora install…

- Install PGDG yum repository

```
[root@localhost ~]# cat /etc/fedora-release
Fedora release 31
```

```
dnf install –y dnf install https://download.postgresql.org/pub/repos/yum/reporpms/F-31-x86_64/pgdg-fedora-repo-latest.noarch.rpm
```

- Install Client, Server and contributed package (libs are in dependencies)

```
dnf install -y dnf install postgresql12 dnf install postgresql12-server postgresql12-contrib
```

- Initialize Postgres cluster and start the postgres instance

```
[root@localhost ~]# PGSETUP_INITDB_OPTIONS='-k' /usr/pgsql-12/bin/postgresql-12-setup initdb
Initializing database ... OK
[root@localhost ~]# systemctl enable postgresql-12
```

```
Created symlink /etc/systemd/system/multi-user.target.wants/postgresql-12.service → /usr/lib/systemd/system/postgresql-12.service.
[root@localhost ~]# systemctl start postgresql-12
```

- Example: `PGSETUP_INITDB_OPTIONS="--pgdata=/u01/pgsql/12/data/ -E 'UTF-8'"`
  - If using sudo: `PGSETUP_INITDB_OPTIONS='-k' sudo -E /usr/pgsql-11/bin/postgresql-11-setup initdb`

Fedora – postgres instance is up and running!

- Powerful interactive console… is psql, let's try it

```
[root@localhost ~]# psql
psql: FATAL:  role "root" does not exist
[root@localhost ~]# su - postgres
[postgres@localhost ~]$ psql
psql (10.1)
Type "help" for help.
postgres=# select version();
                                                 version
-----------------------------------------------------------------------------------------------------
 PostgreSQL 10.1 on x86_64-pc-linux-gnu, compiled by gcc (GCC) 7.2.1 20170915 (Red Hat 7.2.1-2), 64-bit
```

Debian / Ubuntu install is similar

```
root@debian:~# lsb_release -rc
Release:        9.3
Codename:       stretch
```

- Install PGDG apt repository

```
echo "deb http://apt.postgresql.org/pub/repos/apt/ stretch-pgdg main" >
/etc/apt/sources.list.d/pgdg.list
```

```
wget --quiet -O - https://www.postgresql.org/media/keys/ACCC4CF8.asc | apt-key add -
```

```
sudo apt-get update
```

- Install Client, Server (libs and contrib are already included)

```
apt install postgresql-client-10 postgresql-10 postgresql-client-common postgresql-common
```

- “common” packages are automatically installed as dependency to client/server package

apt will create a postgres cluster during installation

```
root@debian:~# apt install postgresql-client-10 postgresql-10
Reading package lists... Done
Building dependency tree
Reading state information... Done
…
Setting up postgresql-client-common (189.pgdg90+1) ...
Setting up postgresql-common (189.pgdg90+1) ...
…
Creating config file /etc/postgresql-common/createcluster.conf with new version
Setting up postgresql-10 (10.1-1.pgdg90+1) ...
Creating new PostgreSQL cluster 10/main ...
/usr/lib/postgresql/10/bin/initdb -D /var/lib/postgresql/10/main --auth-local peer --auth-host
md5
…
The database cluster will be initialized with locale "en_US.UTF-8".
…
Data page checksums are disabled.
```

apt created postgres cluster has disabled page checksums

```
Data page checksums are disabled.
…
fixing permissions on existing directory /var/lib/postgresql/10/main ... ok
…
Success. You can now start the database server using:

    /usr/lib/postgresql/10/bin/pg_ctl -D /var/lib/postgresql/10/main -l logfile start

Ver Cluster Port Status Owner    Data directory              Log file
10  main    5432 down   postgres /var/lib/postgresql/10/main /var/log/postgresql/postgresql-10-
main.log
…
root@debian:~#
```

- Disabled page checksum are default, can't be changed by a configuration parameter, page_checksums are part of cluster initialization, --data-checksums option of initdb utility, but on Debian/Ubuntu there are few additional utilities to consider, like pg_createcluster which is integrated to OS and accepts data checksums as well.

finmason

Debian/Ubuntu specific utilities (some of them)

- pg_createcluster

```
root@debian:~# pg_createcluster 10 warehouse -- --data-checksums
Creating new PostgreSQL cluster 10/warehouse ...
/usr/lib/postgresql/10/bin/initdb -D /var/lib/postgresql/10/warehouse --auth-local peer --auth-host md5 --data-checksums
The files belonging to this database system will be owned by user "postgres".
This user must also own the server process.

The database cluster will be initialized with locale "en_US.UTF-8".
The default database encoding has accordingly been set to "UTF8".
The default text search configuration will be set to "english".

Data page checksums are enabled.
…

Ver Cluster   Port Status Owner    Data directory                  Log file
10  warehouse 5433 down   postgres /var/lib/postgresql/10/warehouse /var/log/postgresql/postgresql-10-warehouse.log
```

Debian/Ubuntu specific utilities (some of them)

- pg_lsclusters
  - Provides handy list of created clusters (data directories with configuration)

```
root@debian:~# pg_lsclusters
Ver Cluster   Port Status Owner    Data directory                  Log file
10  main      5432 online postgres /var/lib/postgresql/10/main      /var/log/postgresql/postgresql-10-main.log
10  warehouse 5433 down   postgres /var/lib/postgresql/10/warehouse /var/log/postgresql/postgresql-10-warehouse.log
```

- pg_conftool
  - Show or set configuration value without editing configuration file by hand
- pg_ctlcluster
  - `pg_ctlcluster 10 warehouse start` … instance control, what else one might expect ☺
- pg_renamecluster, pg_dropcluster
  - self explanatory utility names, most of them are suitable for automation

Where are configuration and log files?

- RH/Fedora/CentOS
  - as already mentioned, under data directory
  - log rotation has to be configured in postgres configuration file or by creating logrotate rules
  - Default data directory
  - `/var/lib/pgsql/12/data/`
  - Log files directory
  - `/var/lib/pgsql/12/data/log`
    - `pg_log` for ver. <= 9.6
- Debian/Ubuntu
  - `pg_lsclusters` show us that information for log file
  - /var/log/postgresql/postgresql-<version>-<cluster_name>.log
    - /var/log/postgresql/postgresql-10-main.log
    - /var/log/postgresql/postgresql-10-warehouse.log
- Configuration is in `/etc` same structure as for logs
  - /etc/postgresql/<version>/<cluster_name>/postgresql.conf
  - /etc/postgresql/10/main/postgresql.conf
  - /etc/postgresql/10/warehouse/postgresql.conf
- postgresql-common also **configure log rotation**
  - /etc/logrotate.d/postgresql-common

finmason

psql console

- \h – list available help on SQL commands

```
=# \h drop index
Command:     DROP INDEX
Description: remove an index
Syntax:
DROP INDEX [ CONCURRENTLY ] [ IF EXISTS ] name [, ...] [ CASCADE | RESTRICT ]

URL: https://www.postgresql.org/docs/12/sql-dropindex.html
```

- \? – help on psql console commands

```
=# \?
General
  \copyright             show PostgreSQL usage and distribution terms
…
Informational
  (options: S = show system objects, + = additional detail)
  \d[S+]                 list tables, views, and sequences
  \d[S+]  NAME           describe table, view, sequence, or index
```

`\q` is the quit command in psql, from version 11 `quit` and `exit` commands

- Powerful interactive console…

```
postgres=# show data_directory;
     data_directory
-----------------------
 /var/lib/pgsql/10/data
(1 row)
postgres=# \q
[postgres@localhost ~]$
```

- We have issued our first select statement and learned how to display a parameter value and how to quit psql console.
  - Let’s explore content of the data directory

finmason

Things to do after installation

- Tweak configuration
  - Some parameters are usually subject of change because defaults are very conservative
  - Configure remote server access
  - Create roles (users) – no one really wants all DB traffic to run under postgres user
  - Create databases
  - … ***setup BACKUP & RECOVERY procedures***
  - Setup monitoring and alerting
- By default there is **no password** for default **postgres superuser account** and default **database postgres** is created

Image by Harald Traue from Pixabay

finmason

Things usually tweaked

- listen_address
- host based authentication
- archive_mode
- shared_buffers
- Logging configuration
- Backup & recovery procedure
  - Always setup ***REGULAR RECOVERY TEST*** procedure
  - pg_basebackup
  - pgBackRest
  - Barman
  - WAL-E …. and many other backup tools are available
- Replication to a disaster recovery site

Image by Harald Traue from Pixabay

Postgres cluster data directory is well documented

- **base** is the directory, where by default databases resides
- **log** is directory where … yes text server log files are
  - Releases up to 9.x it was named **pg_log**
- **pg_wal** is directory for **binary** transaction log (Write Ahead Log) used during recovery
- **pg_hba.conf** is text configuration file to control user access to database by client origin – HBA: host-based authentication – this file is supposed to be maintained by DBA
- **postgresql.conf** – main text configuration file, DBA responsibility
- **postgresql.auto.conf** – do not touch (it is "alter system" command managed)
- **postgresql cluster** - directory (might be referred by environment variable PGDATA) traditionally called database cluster to store all necessary configuration and data managed by Postgres instance

base	pg_snapshots
current_logfiles	pg_stat
global	pg_stat_tmp
log	pg_subtrans
pg_commit_ts	pg_tblspc
pg_dynshmem	pg_twophase
pg_hba.conf	PG_VERSION
pg_ident.conf	pg_wal
pg_logical	pg_xact
pg_multixact	***postgresql.auto.conf***
pg_notify	**postgresql.conf**
pg_replslot	postmaster.opts
pg_serial	postmaster.pid

Database instance is up and running, so connect to it!

- Client – (host named ubuntu), Server – (host named fedora)
- Server side

```
[root@fedora log]# systemctl status postgresql-10.service
● postgresql-10.service - PostgreSQL 10 database server
   Loaded: loaded (/usr/lib/systemd/system/postgresql-10.service; enabled; vendor preset: disabled)
   Active: active (running) since Sat 2018-02-03 12:47:55 CET; 3h 12min ago
     Docs: https://www.postgresql.org/docs/10/static/
  Process: 763 ExecStartPre=/usr/pgsql-10/bin/postgresql-10-check-db-dir ${PGDATA} (code=exited,
status=0/SUCCESS)
 Main PID: 777 (postmaster)
...
```

  - Server is up and running

Local connection and server “uptime”

- Client – (host named ubuntu), Server – (host named fedora)
- Server side check using psql
  - As postgres user (as shown previously) test connection

```
[postgres@fedora ~]$ psql
psql (10.1)
Type "help" for help.

postgres=# select pg_postmaster_start_time();
   pg_postmaster_start_time
-------------------------------
 2018-02-03 12:47:55.808918+01
(1 row)

postgres=# \q
[postgres@fedora ~]$
```

Remote connection test

- Client side check using psql

```
root@debian:~# psql -h 192.168.151.48 -U postgres -d postgres
psql: could not connect to server: Connection refused
        Is the server running on host "192.168.151.48" and accepting
        TCP/IP connections on port 5432?
```

- Options
  - -h server_host
  - -U database username
  - -d database name, default database is postgres created by initdb utility
- The test obviously failed…
  - Why, we have already checked network connectivity and added FW rule for 5432/TCP…?

Connectivity issue diagnostic

- Check server log file

```
2018-02-03 12:47:55.819 CET [827] LOG:  database system was shut down at 2018-02-02 00:20:39 CET
2018-02-03 12:47:55.895 CET [777] LOG:  database system is ready to accept connections
```

  - There is no record for our unsuccessful connection test

- Is postgres listening on network port 5432

```
[postgres@fedora log]$ lsof -Pni :5432
COMMAND   PID     USER   FD   TYPE DEVICE SIZE/OFF NODE NAME
postmaste 777 postgres    3u  IPv6  23983      0t0  TCP [::1]:5432 (LISTEN)
postmaste 777 postgres    4u  IPv4  24002      0t0  TCP 127.0.0.1:5432 (LISTEN)
```

  - On localhost… that is not where we are trying to connect

```
postgres=# show listen_addresses;
 listen_addresses
------------------
 localhost
(1 row)
[postgres@fedora data]$ grep listen_addresses postgresql.conf
#listen_addresses = 'localhost'         # what IP address(es) to listen on;
```

listen_addresses – where postgres server listens for connections

- Change listen_address in postgresql.conf

```
[postgres@fedora data]$ grep listen_addresses postgresql.conf
listen_addresses = '*'          # what IP address(es) to listen on;
```

  - Parameter is static, so server restart is necessary

```
[root@fedora log]# systemctl restart postgresql-10.service
```

- Repeat test from client

```
root@ubuntu:~# psql -h fedora -U postgres
psql: FATAL:  no pg_hba.conf entry for host "192.168.151.32", user "postgres", database "postgres", SSL off
root@ubuntu:~#
```

- Failure, again…? What about a server log?

```
2018-02-03 16:39:40.217 CET [831] LOG:  shutting down
2018-02-03 16:39:40.234 CET [777] LOG:  database system is shut down
2018-02-03 16:39:40.331 CET [1907] LOG:  database system was shut down at 2018-02-03 16:39:40 CET
2018-02-03 16:39:40.338 CET [1905] LOG:  database system is ready to accept connections
2018-02-03 16:42:09.066 CET [1916] FATAL:  no pg_hba.conf entry for host "192.168.151.32", user "postgres",
database "postgres", SSL off
```

finmason

Configure host-based authentication

- Remember the pg_hba.conf ?
  - It might seem as an additional obstacle, but it is fair security feature and yes, default is restrictive:

```
[root@fedora data]# grep -v -e "^[[:space:]]*$" -e "^[[:space:]]*#" pg_hba.conf
local   all             all                                     peer
host    all             all             127.0.0.1/32            ident
host    all             all             ::1/128                 ident
local   replication     all                                     peer
host    replication     all             127.0.0.1/32            ident
host    replication     all             ::1/128                 ident
```

  - Configure HBA to accept connection from our testing subnet and an additional host

```
local   all             all                                     peer
host    all             all             127.0.0.1/32            ident
# Subnet range 192.168.151.[0 - 63]
host    all             all             192.168.151.0/26        md5
host    all             all             192.168.151.64/32       md5
host    all             all             ::1/128                 ident
```

  - Activate new settings

```
[root@fedora data]# systemctl reload postgresql-10.service
```

  - Check database server log file

```
2018-02-03 17:34:53.298 CET [1905] LOG:  received SIGHUP, reloading configuration files
```

Connection should work now…

- Repeat the test

```
root@ubuntu:~# psql -h fedora -U postgres
Password for user postgres:
psql: FATAL:  password authentication failed for user "postgres"
```

- Server log file

```
2018-02-03 17:39:29.088 CET [2209] FATAL:  password authentication failed for user "postgres"
2018-02-03 17:39:29.088 CET [2209] DETAIL:  User "postgres" has no password assigned.
        Connection matched pg_hba.conf line 84: "host    all     all     192.168.151.0/26    md5"
```

  - Activate new settings

```
[root@fedora data]# systemctl reload postgresql-10.service
```

  - Check database server log file

```
2018-02-03 17:34:53.298 CET [1905] LOG:  received SIGHUP, reloading configuration files
```

finmason

Create a user or set password for existing one

- On database server cerate an user

```
postgres@fedora:~# psql
postgres=# create role testuser login password 'heslo';
CREATE ROLE
postgres=# \password
Enter new password:
Enter it again:
```

- Or set password for existing one (for example default postgres user)

```
postgres@fedora:~# psql
postgres=# select current_user, current_database();
 current_user | current_database
--------------+------------------
 postgres     | postgres
postgres=# \password postgres
Enter new password:
Enter it again:
```

Test connection from remote host

- Yes, it works!

```
root@ubuntu:~# psql -h fedora -U postgres -c "select current_user,
current_database();"
Password for user postgres:
 current_user | current_database
--------------+------------------
 postgres     | postgres
(1 row)
```

- Really?

```
root@ubuntu:~# psql -h fedora -U testuser -c "select current_user,
current_database();"
Password for user testuser:
psql: FATAL:  database "testuser" does not exist
root@ubuntu:~#
```

Yes, it worked as expected

- If target database is not specified, postgres tries connect to database with same name as username, that is why previous failure was expected (by the way, all already mentioned failures were expected correct behavior)

```
root@ubuntu:~# psql -h fedora -U testuser -d postgres -c "select
current_user, current_database(), pg_backend_pid(),
inet_server_addr();"
Password for user testuser:
 current_user | current_database | pg_backend_pid | inet_server_addr
--------------+------------------+----------------+-----------------
 testuser     | postgres         |           1268 | 192.168.151.48
(1 row)
```

Installation summary

- Thanks to packaging systems and Windows installer by EDB, installation is very easy
  - Enable repository
  - Install desired packages, for DB servers install contrib package if not already part of server
- Check firewall rules on DB server
- Configure listen_addresses properly
- Configure pg_hba.conf
  - Do not allow whole internet to access your database! Be specific as much as you can.
  - First matching row is used for a connection, subsequent rules are ignored.
  - If there is no matching HBA entry, access is denied
    - Use IP ranges/subnets instead of *
- Create users or set password for existing ones

PostgreSQL cluster and operating system

- RDBMS SW installed -> Postgres instance -> databases managed by instance.

SW installation

/usr/lib/postgresql/10/{lib,bin}
/usr/lib/postgresql/10/bin/postgres # server binary file

Postgres instance – process running in OS, main process is postmaster:

```
root@debian:~# ps -fp 399
UID          PID  PPID  C STIME TTY          TIME CMD
postgres     399     1  0 07:21 ?        00:00:00 /usr/lib/postgresql/10/bin/postgres -D /var/lib/postgresql/10/main
```

PostgreSQL architecture, simplified.

- Prepare a bit more complex environment….
  - We'll add an older major version 9.6 due to a legacy application requirement

```
root@debian:~# apt install postgresql-9.6 postgresql-client-9.6 postgresql-contrib-9.6
root@debian:~# pg_ctlcluster 9.6 main start
root@debian:~# pg_lsclusters
Ver Cluster   Port Status Owner    Data directory                  Log file
9.6 main      5434 online postgres /var/lib/postgresql/9.6/main    /var/log/…esql-9.6-main.log
10  main      5432 online postgres /var/lib/postgresql/10/main     /var/log/…esql-10-main.log
10  warehouse 5433 online postgres /var/lib/postgresql/10/warehouse /var/log/…house.log
```

Create more complex example

- Main cluster running on default port 5432

```
postgres@debian:~$ psql
psql (10.1)
Type "help" for help.

postgres=# \l
                                 List of databases
   Name    |  Owner   | Encoding |   Collate   |    Ctype    |   Access privileges
-----------+----------+----------+-------------+-------------+-----------------------
 postgres  | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
 template0 | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 | =c/postgres          +
           |          |          |             |             | postgres=CTc/postgres
 template1 | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 | =c/postgres          +
           |          |          |             |             | postgres=CTc/postgres
(3 rows)
postgres=# select current_setting('server_version');
 current_setting
-----------------
 10.1
(1 row)
```

Create more complex example

- Main cluster running on default port 5432 wants new databases

```
postgres=# create database sales;
CREATE DATABASE
postgres=# create database webdb;
CREATE DATABASE
postgres=# \l
                                  List of databases
   Name    |  Owner   | Encoding |   Collate   |    Ctype    |   Access privileges
-----------+----------+----------+-------------+-------------+-----------------------
 postgres  | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
 sales     | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
 template0 | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 | =c/postgres          +
           |          |          |             |             | postgres=CTc/postgres
 template1 | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 | =c/postgres          +
           |          |          |             |             | postgres=CTc/postgres
 webdb     | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
(5 rows)

postgres=# \q
```

Create more complex example

- Warehouse cluster on port 5433 feels unused without its own database

```
postgres@debian:~$ psql -p 5433
psql (10.1)
Type "help" for help.

postgres=# create database warehouse;
CREATE DATABASE
postgres=# \l
                                  List of databases
   Name    |  Owner   | Encoding |   Collate   |    Ctype    |   Access privileges
-----------+----------+----------+-------------+-------------+-----------------------
 postgres  | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
 template0 | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 | =c/postgres          +
           |          |          |             |             | postgres=CTc/postgres
 template1 | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 | =c/postgres          +
           |          |          |             |             | postgres=CTc/postgres
 warehouse | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
(4 rows)
```

Create more complex example

- main cluster in version 9.6 on port 5434 can’t be abandoned, DB is needed

```
postgres@debian:~$ psql -p 5434
psql (10.1, server 9.6.6)
Type "help" for help.

postgres=# select current_setting('server_version_num');
 current_setting
-----------------
 90606
(1 row)

postgres=# create database car_rental;
CREATE DATABASE
```

There are more things to do after installation

Let’s play for a while, before the hard-serious work begins

Things to do after installation

Bookmark documentation,

go through **Part III. Server Administration**.

Logging configuration

- Log rotation (save logs as long as possible, same for OS statistics, sar…)
  - log_destination = 'stderr'
  - logging_collector = on
  - log_directory = 'pg_log'
  - log_filename = 'postgresql.log' # 'postgresql%d.log' if NOT using logrotate…
  - log_truncate_on_rotation = off # on if NOT using logrotate…
  - log_rotation_age = 1d # daily if NOT using logrotate
  - log_rotation_size = 0 # can be used instead if time rotation, if NOT using…
- log_checkpoints = on
- log_connections = on
- log_disconnections = on
- log_line_prefix = '%t %c %l %r %d %u %x %v %i:'

finmason

Logging configuration

- log_lock_waits = on
- log_statement = 'ddl’
- log_temp_files = 2048 # log temporary files equal or larger then [Bytes]
- log_min_duration_statement = 5000 # log statemetns lastimg for … ms – slow queries
- log_timezone = 'Europe/Prague’ # UTC for international companies!
- track_counts = on # default
- track_io_timing = on # use pg_test_timing in advance to test your CPU speed
- autovacuum = on # default
- log_autovacuum_min_duration = 0 # if there are too many log entries 250 [ms]…
- default_tablespace = <some_ts> # **create** some_ts where public is not usage granted
- temp_tablespace = ‘temp’ # **create** temp tablespace on separate filesystem

Instance configuration

- shared_buffers = <NN>GB # usually you want to change default value
- work_mem = 64MB # not a per session, but per SQL exec plan node
- maintenance_work_mem = 1GB # index creation, vacuum operations…
- shared_preload_libraries = 'pg_stat_statements' # must have extension
- wal_level = replica # or higher
- min_wal_size = <NN>GB
- max_wal_size = <NN>GB # This is a soft limit; WAL size can exceed max_wal_size under special circumstances, like under heavy load, a failing archive_command, or a high wal_keep_segments setting.
- checkpoint_completion_target = 0.75

Instance configuration

- cluster_name = <your instance name> # handy for monitoring and identifying instances
- archive_mode = on # ALLWAYS set to **on**. Change requires instance restart.
- ## non production environments
  - archive_command = '/bin/true’
- ## production
  - archive_command = 'rsync -a -q -e \"ssh -q\" %p ...%f’
  - or whatever provided by a trusted backup and recovery solution
- use “quiet” option for your rsync, scp, aws s3 cp … commands, otherwise your logs will be flooded by output from copy “progress bar” (nice in interactive session, ugly in logfiles)
- effective_cache_size = …GB # 4GB is default and it is usually fine, should be set to an estimate of how much memory is available for disk caching by the operating system

Instance configuration

- Autovacuum settings
  - tiding up is a tedious work, someone has to cleanup all the mess produced by usual database workload
  - vacuum eliminates table bloat and protects you against possible wraparound
- vacuum_cost_limit = 5000
  - once started, do more work
- autovacuum_vacuum_cost_delay = 4ms
  - mess is ugly (and might impact consumed space and performance), do not relax too long while vacuumimg
- vacuum prevent **bloat** in your database
  - MVCC implementation in Postgres enables minimum locking - high throupt at cost of table bloat and necessary vacuum maintenance
- check_postgres is great tool to help you monitor table bloat

Instance configuration / first step after installation

- DESIGN and IMPLEMENT a BACKUP & RECOVERY PROCESS
- DEGIGN and IMPLEMENT REGULAR AUTOMATED (optimally for each backup) RESTORE TEST
  - untested backup is just a wish…

DBA tasks

- BACKUP & RECOVERY
- Incident / disaster event WILL happen, the only question is…
  - ~~WHEN ?~~
  - ARE YOU READY TO SOLVE a disaster?

User and privilege management

- DBA is usually responsible for
  - user management
  - user access control
- DBA should come with proposal for access control
  - do not mix physical users (user/login role) with group privileges (roles)
  - development teams might in their DDL deployment workflow grant permissions to group not to a physical users
- There are multiple different Authentication Methods available, some of them are
  - Password Authentication
    - scram-sha-256 (available from release 10)
    - md5
    - ~~password~~ – never use this clear text method
  - GSSAPI

Safety control rod axe man is an emergency shutdown of a nuclear reactor.

Do not swap use cases ☺

picture: CC BY 2.0

Roles and users

- PostgreSQL use roles to manage user access control
  - NOLOGIN - might be treated as group in OS
  - LOGIN – might be treated as user on OS (we have already done it when dealing with connection)
- both types of roles can be granted (looks like a sudo for login roles)
  - roles might be used in pg_hba.conf what is very handy
    - user entry in pg_hba.conf prefixed with **+** mark really means “match any of the roles that are directly or indirectly members of this role”
- roles grants might be inherited
- use different roles for application schema and access to application data
  - weather – role owning database objects (tables…)
  - weather_online – role to grant access for an online module of your app
  - weather_reports – reporting might work with limited access

Databases

- Databases are cluster level object
  - it is not possible to select data from multiple databases in one session (always connected to exactly one database [dblink and FDW provides capability to access another database/datasource])
- use UTF8 database encoding
  - if it is not possible, then use UTF8… otherwise you will be in troubles, responsible for collation handling at application level, some functions count characters and they will not know how to handle and unknown charset
- usually databases are owned by the application owner role
  - this does not mean, that an application role need create database privilege, DBA might create database and set DB ownership, this way, application role permissions are limited and your design is more safe

Disaster resiliency solution

- DBA have to provide appropriate DR solution, postgres offer several options
  - pg_dump (and pg_dumpall for globals) to dump database (or selected tables)
  - offline backup – if you can afford downtime, it is OK to use offline backups
  - online backup – pg_basebackup can do all the work for you, or you can combine pg_start_backup(), pg_stop_backup() functions and copy database cluster online, together with transaction logs it will create for you a recoverable backup
  - there are ready to use mature solutions available (open, major contributors are mentioned)
    - pgbarman by 2nd Quadrant (AWS S3 integration from 2.10 - 5 Nov 2019)
    - pgBackRest by Crunchy Data (AWS S3 integration)
    - wal-e , wal-g by Citus Data (AWS S3 integration)
    - BART by EDB and more, some commercial solutions exists as well
    - pg_probackup by Postgres Professional I its free, form vanilla releases requires a patch, “Pro” releases by Postgres Professional are supported directly.

Disaster resiliency solution

- Always backup to a different machine, or preferably have an off-site backup
- Postgres supports various replication methods, so you can have replicas in DR site, hot standby is supported, so your read-only workload might run on one ore more replicas
  - asynchronous
  - synchronous
  - logical (this replication can replicate only subset of source database tables, replica is writable)
- It is possible to switch from a primary instance to a replica, to promote new primary, for example before planned downtime, or if primary instance is lost by a failure
- OS clusterware solution is also perfectly valid solution

Monitoring (and alerting)

- Any professional solution, including PostgreSQL requires to have proper monitoring and alerting (sometimes comes as integrated package) in place.
  - MUNIN
  - NAGIOS
  - ZABBIX
  - OKMETER
  - pgwatch2 (influx, Grafana based)
  - check_postgres (Nagios integration)
  - pgbadger for log parsing (report might contain sensitive data – statements and bind variables)
  - POWA
  - …

finmason

There is nice cheat sheet

- written by Pavel Stěhule in Czech language – tahák.
- psql can produce nice tables

```
postgres=# select * from pg_user;
```

usename	usesysid	usecreatedb	usesuper	userepl	usebypassrls	passwd	valuntil	useconfig
postgres	10	t	t	t	t	********	«¤»	«¤»
weather_reports	16390	f	f	f	f	********	«¤»	«¤»
weather_online	16389	f	f	f	f	********	«¤»	«¤»
joethedba	16438	f	f	f	f	********	«¤»	«¤»
sample_user	16442	f	f	f	f	********	«¤»	«¤»

```
(5 rows)
[local]:5432 postgres@postgres(10.1)=(1)# select session_user, current_user;
```

session_user	current_user
postgres	postgres

```
(1 row)
```

- psql can use a pspg pager for interactive browsing through result set, by Pavel Stěhule ☺ Already in PGDG repositories, ready to install.

Summary

- After installation, configure listen_address to allow remote connection to your DB
- Create roles and configure HBA to accept user connection attempts
  - +role_name is your friend in hba configuration file
- Create NOLOGIN roles to own databases, schemas and objects within a schema
  - Use nologin roles to control access to your database resources
  - Grant the access control roles to real end users (people)
  - roles by default inherit privileges from other granted roles
- Backup is a must have, restores has to be regularly tested
- Postgres instance consist of several processes and backend processes which are created per user
- Monitoring is an obligation

Summary … (cont.)

- Instance memory settings are by default conservative, revise them and tweak
- Instance logging has lots of options, configure them, default is so silent
- PostgreSQL supports different replication options to have DR implemented, including synchronous and asynchronous replication, hot standby mode might be used to balance your read workload and logical replication allow replica in read-write mode
- PostgreSQL cluster is represented by a directory structure
  - for some parts of it, it might be wise to use separate file systems
- PostgreSQL is very flexible thanks to the extensibility by supplied modules – pg_stat_statements is one of “must have” extensions

Too many mistakes in slides

- Why we show so many errors?
  - In memory to my beginner time with postgres, I've tried to show some traps where is easy to fall. None of them is postgres bug, as most often, there is bug between keyboard and chair ☺
  - And solution for these difficulties before one came familiar how this RDBMS works.
- Postgres is powerful engine supported by famous community
- There is no problem to order commercial support if your company policy requires that
  - and at least for the "adoption" period, I'd advice to order it
    - it helps with smooth implementation
    - avoids judgement on postgres caused by lack of admin experience
    - helps to finance community, most of the companies offering commercial support finance full time developers of the open source code

Feedback is welcome

finmason

Bonus slides (?) - Examples and usual mistakes, grant advice…

- Before we start, make sure HBA rules will not block us…
  - btw lines with null address are excluded…, so local and IPv6 (long lines) are not listed

```
postgres=# select line_number, type, database, user_name, address, netmask, auth_method, options from
pg_hba_file_rules where address !~ '::1';
 line_number | type |   database    | user_name |    address    |     netmask     | auth_method | options
-------------+------+---------------+-----------+---------------+-----------------+-------------+---------
          92 | host | {all}         | {all}     | 127.0.0.1     | 255.255.255.255 | md5         |
          93 | host | {all}         | {all}     | 192.168.151.0 | 255.255.255.0   | md5         |
          99 | host | {replication} | {all}     | 127.0.0.1     | 255.255.255.255 | md5         |
(3 rows)
```

Example, from create database to… lots of mistakes

- Create a role first, so it can own a database we'll create later

```
postgres=# create role weather;
CREATE ROLE
postgres=# \du weather
            List of roles
 Role name |  Attributes  | Member of
-----------+--------------+-----------
 weather   | Cannot login | {}
postgres=#
postgres=# create role weather_online with login password 'secret';
CREATE ROLE
postgres=# create role weather_reports with login password 'secret';
CREATE ROLE
```

Example, from create database to… lots of mistakes

- Review roles, we have created

```
postgres=# \du
                                   List of roles
    Role name    |                         Attributes                         | Member of
-----------------+------------------------------------------------------------+-----------
 postgres        | Superuser, Create role, Create DB, Replication, Bypass RLS | {}
 weather         | Cannot login                                               | {}
 weather_online  |                                                            | {}
 weather_reports |                                                            | {}

postgres=#
```

Create a database

- Create database and list what databases exists in our cluster

```
postgres=# create database weather owner weather;
CREATE DATABASE
postgres=# \l
                                  List of databases
   Name    |  Owner   | Encoding |   Collate   |    Ctype    |   Access privileges
-----------+----------+----------+-------------+-------------+-----------------------
 postgres  | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
 sales     | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
 template0 | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 | =c/postgres          +
           |          |          |             |             | postgres=CTc/postgres
 template1 | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 | =c/postgres          +
           |          |          |             |             | postgres=CTc/postgres
 weather   | weather  | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
 webdb     | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
(6 rows)
```

Create a database and …

- connect to the new database
  - can’t work, since role weather has no password and is NOLOGIN (default,)

```
postgres@debian:~$ psql -h 192.168.151.16 -d weather -U weather;
Password for user weather:
psql: FATAL:  password authentication failed for user "weather"
FATAL:  password authentication failed for user "weather"
```

Create a database and …

- connect to the new database
  - can’t work, since role weather has no password and is NOLOGIN (default,)

```
postgres@debian:~$ psql -h 192.168.151.16 -d weather -U weather;
Password for user weather:
psql: FATAL:  password authentication failed for user "weather"
FATAL:  password authentication failed for user "weather“
postgres@debian:~$ psql -h 192.168.151.16 -d weather -U weather_online
Password for user weather_online:
psql (10.1)
SSL connection (protocol: TLSv1.2, cipher: ECDHE-RSA-AES256-GCM-SHA384, bits: 256, compression: off)
Type "help" for help.
weather=>\q

postgres@debian:~$ psql -h 192.168.151.16 -d weather -U weather_reports
Password for user weather_reports:
psql (10.1)
SSL connection (protocol: TLSv1.2, cipher: ECDHE-RSA-AES256-GCM-SHA384, bits: 256, compression: off)
Type "help" for help.
weather=> \q
```

Create a database and … start securing our brand new DB

- First securing step
  - avoid access to public schema, or drop it.
- Behind the scenes, we have learned that in PostgreSQL DDL statements are also transactional

```
weather=# revoke all on schema public from public;
REVOKE
```

```
weather=# begin;
BEGIN
weather=# drop schema public;
DROP SCHEMA
weather=# \dn
List of schemas
 Name | Owner
------+-------
(0 rows)

weather=# rollback;
ROLLBACK
weather=# \dn
  List of schemas
  Name  |  Owner
--------+----------
 public | postgres
(1 row)
```

Create a database and … think about schemas

- create schema for an application
  - authorization means, that the schema is owed by weather role
  - schema ownership can also be altered later

```
weather=# create schema weather authorization weather;
CREATE SCHEMA
weather=#
```

```
weather=# create schema weather authorization weather;
CREATE SCHEMA
weather=# ALTER SCHEMA weather OWNER TO weather;
ALTER SCHEMA
```

Create a table in the new schema

- connect as one of our application users and list schemas within weather database

```
postgres@debian:~$ psql -h 192.168.151.16 -d weather -U weather_online
Password for user weather_online:
psql (10.1)
SSL connection (protocol: TLSv1.2, cipher: ECDHE-RSA-AES256-GCM-SHA384, bits: 256,
compression: off)
Type "help" for help.

weather=> \dn
  List of schemas
  Name   |  Owner
---------+----------
 public  | postgres
 weather | weather
(2 rows)
```

Create a table in the new schema

- failure is expected
  - we have revoke all on public schema from all users (public)

```
weather=> create table foo(bar int);
ERROR:  no schema has been selected to create in
LINE 1: create table foo(bar int);
                     ^
weather=>
```

Try a better design

- Have a role weather_app which will be used as group
  - users access will be managed through the role membership
  - … yes we have to connect to right database, schema is database level object

```
postgres=# create role weather_app;
CREATE ROLE
postgres=# \du weather_app;
             List of roles
  Role name  |  Attributes  | Member of
-------------+--------------+-----------
 weather_app | Cannot login | {}

postgres=# grant usage on schema weather to weather_app;
ERROR:  schema "weather" does not exist
postgres=#
```

Try a better design

- let user weather_onlie use our application thanks to role grant

```
postgres=# \c weather
You are now connected to database "weather" as user "postgres".
weather=# grant usage on schema weather to weather_app;
GRANT

weather=# grant weather_app to weather_online;
GRANT ROLE
```

Try a better design

- Test the new setup.

```
postgres@debian:~$ psql -h 192.168.151.16 -d weather -U weather_online
Password for user weather_online:
psql (10.1)
SSL connection (protocol: TLSv1.2, cipher: ECDHE-RSA-AES256-GCM-SHA384, bits: 256,
compression: off)
Type "help" for help.

weather=> create table foo(bar int);
ERROR:  no schema has been selected to create in
LINE 1: create table foo(bar int);
                     ^
```

finmason

Try a better design

- … yes the new role, much better we have reached another privilege barrier

```
weather=> set role weather_app;
SET
weather=> select current_user, session_user;
 current_user |  session_user
--------------+----------------
 weather_app  | weather_online
(1 row)

weather=> create table foo(bar int);
ERROR:  no schema has been selected to create in
LINE 1: create table foo(bar int);
                     ^
weather=>
```

Try a better design

- use psql command to discover what is wrong
  - \dn lists schemas (can be memorized as “list namespaces”)
  - weater_app has no **C**reate privilege

```
weather=> \dn+
                              List of schemas
  Name   |  Owner   |   Access privileges   |      Description
---------+----------+-----------------------+-------------------------
 public  | postgres | postgres=UC/postgres  | standard public schema
 weather | weather  | weather=UC/weather   +|
         |          | weather_app=U/weather |
(2 rows)
```

Try a better design

- use psql command to discover what is wrong
  - \dn lists schemas (can be memorized as “list namespaces”)
  - weater_app has no **C**reate privilege, so fix it for now

```
weather=> \dn+
                             List of schemas
  Name   |  Owner   |   Access privileges   |      Description
---------+----------+-----------------------+-------------------------
 public  | postgres | postgres=UC/postgres  | standard public schema
 weather | weather  | weather=UC/weather   +|
         |          | weather_app=U/weather |
(2 rows)
postgres@debian:~$ psql weather
psql (10.1)
Type "help" for help.

weather=# grant create on schema weather to weather_app;
GRANT
weather=# \q
```

Try a better design

- no privilege on public, specify schema you want to use

```
postgres@debian:~$ psql -h 192.168.151.16 -d weather -U weather_online
Password for user weather_online:
psql (10.1)
SSL connection (protocol: TLSv1.2, cipher: ECDHE-RSA-AES256-GCM-SHA384, bits: 256, compression: off)
Type "help" for help.

weather=> select current_user, session_user;
  current_user  |  session_user
----------------+----------------
 weather_online | weather_online
(1 row)

weather=> create table foo(bar int);
ERROR:  no schema has been selected to create in
LINE 1: create table foo(bar int);
                     ^
weather=> create table weather.foo(bar int);
CREATE TABLE
```

Try a better design

- Here is, why it worked now
  - the **C**reate privilege

```
weather=> \dn+
                                 List of schemas
  Name   |  Owner   |   Access privileges    |      Description
---------+----------+------------------------+------------------------
 public  | postgres | postgres=UC/postgres   | standard public schema
 weather | weather  | weather=UC/weather    +|
         |          | weather_app=UC/weather |
(2 rows)
```

How role inheritance works

- by default roles are created with inherit option
  - so the role receives all privileges inherited from whole hierarchy of inherited roles
  - we'll disable it for now and drop our testing table

```
postgres@debian:~$ psql
psql (10.1)
Type "help" for help.

postgres=# alter role weather_online noinherit;
ALTER ROLE

postgres=# \c weather
You are now connected to database "weather" as user "postgres".

weather=# drop table weather.foo;
DROP TABLE
weather=# \q
```

How role inheritance works

- privileges are unchanged, only inheritance is disabled for this test
  - we can get our identity by a simple select

```
postgres@debian:~$ psql -h 192.168.151.16 -d weather -U weather_online
Password for user weather_online:
psql (10.1)
SSL connection (protocol: TLSv1.2, cipher: ECDHE-RSA-AES256-GCM-SHA384, bits: 256, compression: off)
Type "help" for help.

weather=> create table weather.foo(bar int);
ERROR:  permission denied for schema weather
LINE 1: create table weather.foo(bar int);
                     ^
weather=> select current_user, session_user;
  current_user   |  session_user
-----------------+----------------
 weather_online | weather_online
(1 row)
```

How role inheritance works

- setting right role (thanks to the fact, the role is granted to our test user)
  - testing table is created this time

```
weather=> set role weather_app;
SET
weather=>  select current_user, session_user;
 current_user |  session_user
--------------+---------------
 weather_app  | weather_online
(1 row)

weather=> create table weather.foo(bar int);
CREATE TABLE
weather=> \d weather.foo
                Table "weather.foo"
 Column |  Type   | Collation | Nullable | Default
--------+---------+-----------+----------+---------
 bar    | integer |           |          |
```

How role inheritance works

- reset role command will put our session to initial state

```
weather=> reset role;
RESET
weather=>  select current_user, session_user;
  current_user  |  session_user
----------------+----------------
 weather_online | weather_online
(1 row)
```

How role inheritance works

- reset role command will put our session to initial state
  - and for sure, we do not have the privileges from the moment

```
weather=> reset role;
RESET
weather=>  select current_user, session_user;
  current_user  |  session_user
----------------+---------------
 weather_online | weather_online
(1 row)
weather=> insert info weather.foo (bar) values (1);
ERROR:  syntax error at or near "info"
LINE 1: insert info weather.foo (bar) values (1);
               ^
weather=>
```

How role inheritance works

- inheritance is convenient, why to specify schema all the time?
  - search_path (works similar to OS PATH variable) can be changed at several levels
    - session level by user: `set search_path=my_schema,your_schema;`
    - specific user in a database
    - database level
    - postgres cluster level (in postgresql.conf)

```
postgres@debian:~$ psql
psql (10.1)
Type "help" for help.

postgres=# alter role weather_online inherit;
ALTER ROLE
postgres=# alter database weather set search_path=weather;
ALTER DATABASE
postgres=# \q
```

search_path test

- since search_path is set at database level, we don't need to specify it again at session level, using schema anyway did not cause any harm

```
postgres@debian:~$ psql -h 192.168.151.16 -d weather -U weather_online
Password for user weather_online:
psql (10.1)
SSL connection (protocol: TLSv1.2, cipher: ECDHE-RSA-AES256-GCM-SHA384, bits: 256, compression: off)
Type "help" for help.

weather=> select * from weather.foo ;
 bar
-----
(0 rows)

weather=> insert into foo (bar) values (1);
INSERT 0 1
weather=> select * from foo;
 bar
-----
   1
(1 row)

weather=> \q
```

Table ownership is important

- Privileges can be retrieved from information_schema views
  - this is, because we have created the table while wheather_app role was set, so it owned the table

```
postgres@debian:~$ psql weather
psql (10.1)
Type "help" for help.

weather=# SELECT grantee, privilege_type
weather-# FROM information_schema.role_table_grants
weather-# WHERE table_name='foo';
   grantee    | privilege_type
--------------+----------------
 weather_app | INSERT
 weather_app | SELECT
 weather_app | UPDATE
 weather_app | DELETE
 weather_app | TRUNCATE
 weather_app | REFERENCES
 weather_app | TRIGGER
(7 rows)

weather=#
```

Try a better design

- Table was created, but it is owned by the user used to create the table
  - we might want to have application tables to be owned by the application (nologin) role only

```
postgres@debian:~$ psql
psql (10.1)
Type "help" for help.

postgres=# \c weather
You are now connected to database "weather" as user "postgres".
weather=# \dt weather.foo ;
          List of relations
 Schema  | Name | Type  |    Owner
---------+------+-------+-------------
 weather | foo  | table | weather_app
(1 row)
weather=# drop table weather.foo ;
DROP TABLE
```

Try a better design

- Table was created, but it is owned by the user used to create the table
  - we might want to have application tables to be owned by the application (nologin) role only

```
weather=# set role weather;
SET
weather=> create table foo(foo_id bigserial, bar int);
CREATE TABLE
weather=> \q
```

- or we can crate table under any user with enough privileges and afetrwards

```
ALTER TABLE foo OWNER TO weather;
```

Try a better design

- verify the new setup
  - this looks good, table is properly owned and user has no permission on it

```
postgres@debian:~$ psql -h 192.168.151.16 -d weather -U weather_online
Password for user weather_online:
psql (10.1)
SSL connection (protocol: TLSv1.2, cipher: ECDHE-RSA-AES256-GCM-SHA384, bits: 256, compression: off)
Type "help" for help.

weather=> \dt weather.foo
        List of relations
 Schema  | Name | Type  |  Owner
---------+------+-------+---------
 weather | foo  | table | weather
(1 row)

weather=> select * from weather.foo;
ERROR:  permission denied for relation foo
weather=>
```

Try a better design

- grant the permissions to let our weather_online user work
  - we can use handy macro ALL to grant some privileges on all existing objects of same type to avoid listing objects and issuing grant one by one

```
postgres@debian:~$ psql
psql (10.1)
Type "help" for help.

postgres=# \c weather
weather=# grant select, insert, update, delete, references on ALL tables in schema
weather to weather_app;
GRANT
weather=# \q
```

Try a better design

- check failed
  - we have granted table permissions, but the sequence (created for us by postgres automatically to take default value for foo_id column) grant was omitted

```
postgres@debian:~$ psql -h 192.168.151.16 -d weather -U weather_online
Password for user weather_online:
psql (10.1)
SSL connection (protocol: TLSv1.2, cipher: ECDHE-RSA-AES256-GCM-SHA384, bits: 256, compression:
off)
Type "help" for help.

weather=> insert into foo(bar) values (1);
ERROR:  permission denied for sequence foo_foo_id_seq
```

Default privileges

- ALL is nice macro, but postgres offer also default privilege grant
  - default is applied on all newly created objects for which the default was configured

```
postgres@debian:~$ psql
psql (10.1)
Type "help" for help.

postgres=# \c weather
You are now connected to database "weather" as user "postgres".
weather=# alter default privileges in schema weather grant select, insert, update, delete
on tables to weather_app;
ALTER DEFAULT PRIVILEGES
weather=# alter default privileges in schema weather grant select, usage on sequences to
weather_app;
ALTER DEFAULT PRIVILEGES
weather=# create table weather.foo2(foo2_id bigserial, bar int);
CREATE TABLE
weather=# alter table weather.foo2 owner to weather;
ALTER TABLE
weather=# \q
```

Default privileges

- verify the new table access taken by default privileges

```
postgres@debian:~$ psql -h 192.168.151.16 -d weather -U weather_online
Password for user weather_online:
psql (10.1)
SSL connection (protocol: TLSv1.2, cipher: ECDHE-RSA-AES256-GCM-SHA384, bits: 256, compression: off)
Type "help" for help.

weather=> insert into weather.foo2(bar) values(1);
INSERT 0 1
weather=> select * from foo2;
 foo2_id | bar
---------+-----
       1 |   1
(1 row)

weather=> select * from foo;
 foo_id | bar
--------+-----
(0 rows)
```

Default privileges

- looks good, what about table foo?
  - insert failed, because default privileges were set after table foo and it's sequence already exists

```
postgres@debian:~$ psql -h 192.168.151.16 -d weather -U weather_online
Password for user weather_online:
psql (10.1)
SSL connection (protocol: TLSv1.2, cipher: ECDHE-RSA-AES256-GCM-SHA384, bits: 256, compression: off)
Type "help" for help.

weather=> insert into weather.foo(bar) values(1);
ERROR:  permission denied for sequence foo_foo_id_seq
weather=>
```

Default privileges

- let weather_online use the sequence (through inherited grant, for sure)

```
postgres@debian:~$ psql weather
psql (10.1)
Type "help" for help.

weather=# grant usage on ALL sequences in schema weather to weather_app;
GRANT
weather=# \q
postgres@debian:~$ psql -h 192.168.151.16 -d weather -U weather_online
Password for user weather_online:
psql (10.1)
SSL connection (protocol: TLSv1.2, cipher: ECDHE-RSA-AES256-GCM-SHA384, bits: 256, compression: off)
Type "help" for help.

weather=> insert into weather.foo(bar) values(1);
INSERT 0 1
weather=> select * from weather.foo;
 foo_id | bar
--------+-----
      1 |   1
(1 row)
```

Default privileges

- this was to demonstrate, that usage is enough to use the sequence
  - whereas select has to be granted if we want have the ability select from sequences
  - now it is up to us to decide, whether we want to have select in default grants

```
weather=> select * from weather.foo2_foo2_id_seq;
 last_value | log_cnt | is_called
------------+---------+-----------
          1 |      32 | t
(1 row)

weather=> select * from weather.foo_foo_id_seq;
ERROR:  permission denied for relation foo_foo_id_seq
weather=> \q
```

Permissions setup recommendation summary

- Create a NOLOGIN role which will act as OWNER for your application
  - database
  - schema
  - tables, functions… in schema DB objects
- Create NOLOGIN roles to control access to your database objects
  - grant explicit privileges, do not use ALL (unless you have to)
  - do not grant the “owner” role to access control, through inheritance all user will be almost “owners”
- users (people, or application server connection pool account) will be granted by access control nologin roles
  - user management is separated from access management to individual database objects
- drop public schema or revoke privileges on it

Never grant superuser

- to “owners”, access control roles, personal people accounts
- unless the people are DBAs
  - Do not hesitate to have DBA NOLOGIN role granted with superuser and personal accounts for DB admins will be granted with your DBA role.
  - Role attributes LOGIN, SUPERUSER, CREATEDB, and CREATEROLE can be thought of as special system privileges, whose are never inherited as ordinary privileges on database objects are.
  - If your DBA want to use superuser privilege, `set role DBA;` command is necessary and your DBAs will be aware of the fact, they are working using superuser privilege.

Default privileges might make such setup easy to manage

Database

users
(login roles)

nologin roles
(groups)