An overview of PostgreSQL's backup, archiving and replication

What to do, what not to do, where the pitfalls are

Gunnar „Nick“ Bluth
Currently PostgreSQL DBA at ELSTER
(if you're german, your tax declaration is in my DB)
@nickbluth
nick@pro-open.de

Agenda

- Logical (aka. SQL-) backup
- Binary backup
- PITR & „replication“ with archiving
- Binary streaming replication

- Classic misconceptions and pitfalls
- What you most probably want to do

Some assertions

- You value your data
  - Otherwise you'd not be here ;-)
- You have (at least) 2 decent servers available
  - ~ same amount of CPU and RAM
  - ECC memory
  - BBU HDD controller / SAN
  - a working UPS
- These should by all means seperated as far as possible (and feasible) from each other
- You know your RTO and RPO requirements

Omnipotent natural laws

- Gravity
- Speed of light
- Murphy's law *
  - Disaster does strike
  - Unlike lightning, disaster tends to strike more than once in a row
  - Ask the Gitlab guys!

And keep Einstein in mind

- „Only two things are infinite, the universe and human stupidity, and I'm not sure about the former.“ **

Evolution

- The options and tools evolved over the years
- Nastily, the docs have mostly been amended
- Usually, you want to do it the way that comes last in the docs...
- Which means you're hopefully ready to go after reading ~ 50 pages of A4 in 2 chapters
- You should still read all of it!

The options you had with 7.0

Logical backup

Sidenote: I especially like this one

Prev

Chapter 33. Database Recovery

This section needs to be written. Volunteers?

Prev
Debugging Messages

Logical aka. „SQL-“ backup

- `pg_dump[all]` connects to your DB just as any other client and provides you with a snapshot of your data
  - You can restore the state of the DB at the moment you *initiated* the backup
- Can dump whole clusters (`pg_dumpall`), databases, single tables
- Can provide textual (SQL) representation or custom („proprietary“) format

Textual format of pg_dump

- Plain SQL
- Uses `COPY` for performance
- Can be used to port DBs….

- Can be read by humans

Custom format of pg_dump

- `pg_dump -Fc`
- Restored using `pg_restore` (into `psql` or straight into a DB)
- Can restore single tables
- Compressed by default

Directory format of pg_dump

- `pg_dump -Fd`
- Can backup (and restore) in parallel (`-j X`)
- Restored using `pg_restore` (into `psql` or straight into a DB)
- Can restore single tables
- Compressed by default

Never forget pg_dumpall!

- `pg_dump` reads from **databases**
- So, global objects are not saved!
  - Roles
  - Tablespaces
- So, whenever you `pg_dump`, do a `pg_dumpall --globals-only` along with it!

RTO & RPO of logical backup

- RTO
  - between minutes and days
  - basically depending on size of DB
- RPO
  - your last backup run
  - in the worst case, the one before*!

Pros and cons

+ backup is readable by humans (or can be made so), schema & roles can go to your VCS
+ can be read by newer versions of PG
+ can backup & restore single entities if need be
+ will reveal issues with the „dark corners“ of your DB (when initialised with data checksums)*

- can only backup and thus restore a single point in time
- rather slow
- RPO & RTO... uhm, well

The way beyond pg_dump

- 7.1 added the WAL
- 8.0 added the ability to do
  - On-line backup
  - PITR (no, that's not Pain In The Rear!)
- 9.1 added `pg_basebackup`
  - „gift-wrapping“ existing backup methods
- 9.2 allowed `pg_basebackup` to also fetch WAL data

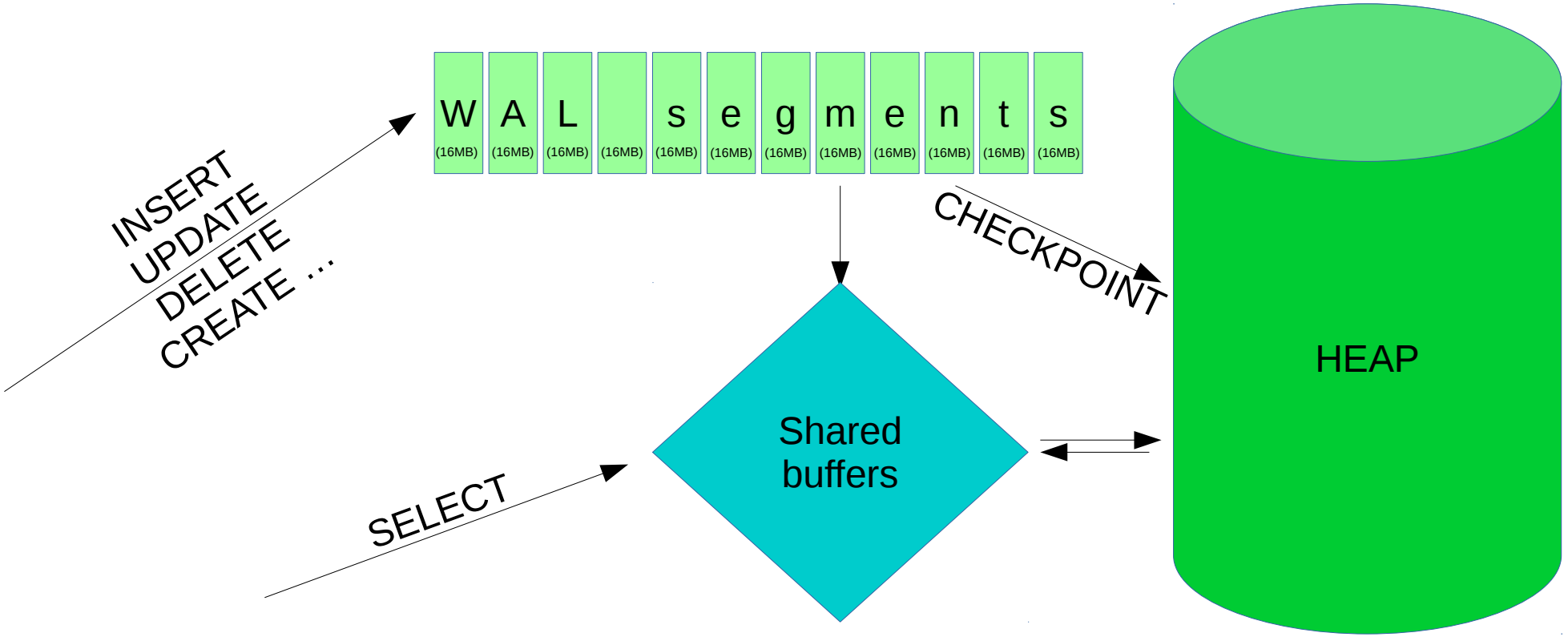
On-line, binary backup

- Erm, we're not there yet ;-)
- We have to discuss some of Postgres' peculiarities first
- Everything binary is centered around the WAL...

What the WAL is

- The Write Ahead Log (WAL) is basically the logbook of the DB
- Other DBMS call the equivalent „redolog“
  - some also have an „undolog“, PG doesn‘t need that
- Every change is first written to the WAL
- At a `CHECKPOINT` (which can be spread!), the content is written to the HEAP, usually creating new row versions

WAL (vastly simplified)



WAL organisation

- The WAL consists of a chain of files, 16MB each („segments“)
- Or more like a ring, as WAL segments get renamed and overwritten when feasible
- It resides in `$PGDATA/pg_xlog` (10.0 ff: „`pg_wal`“**)
- The size is determined by `wal_min_size` and `wal_max_size` (default: 1GB/2GB)
- These are SOFT limits!

The archiver

- WAL segments are written by the „wal writer“ process
- WAL segments are read and applied to the HEAP by the „checkpointer“ process
- In between, they are handed to the archiver process
  - when `archive_mode != ‚off‘`
  - which is almost certainly what you want!

Binary snapshot

- Prepare your database:
  - `pg_start_backup()`
- Get a snapshot
  - We'll discuss the options later!
- „Release“ the HEAP again
  - `pg_stop_backup()`

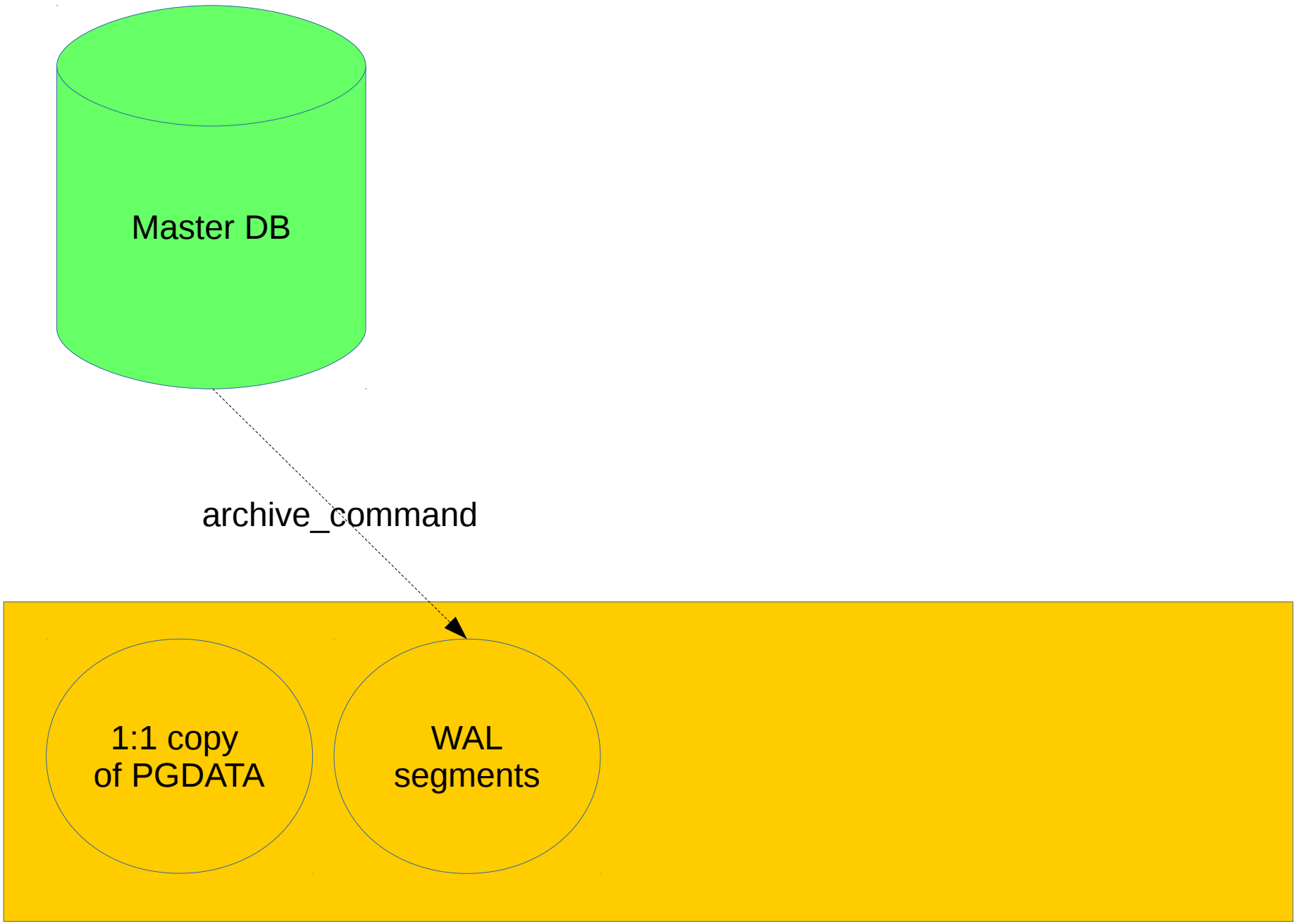
Master DB

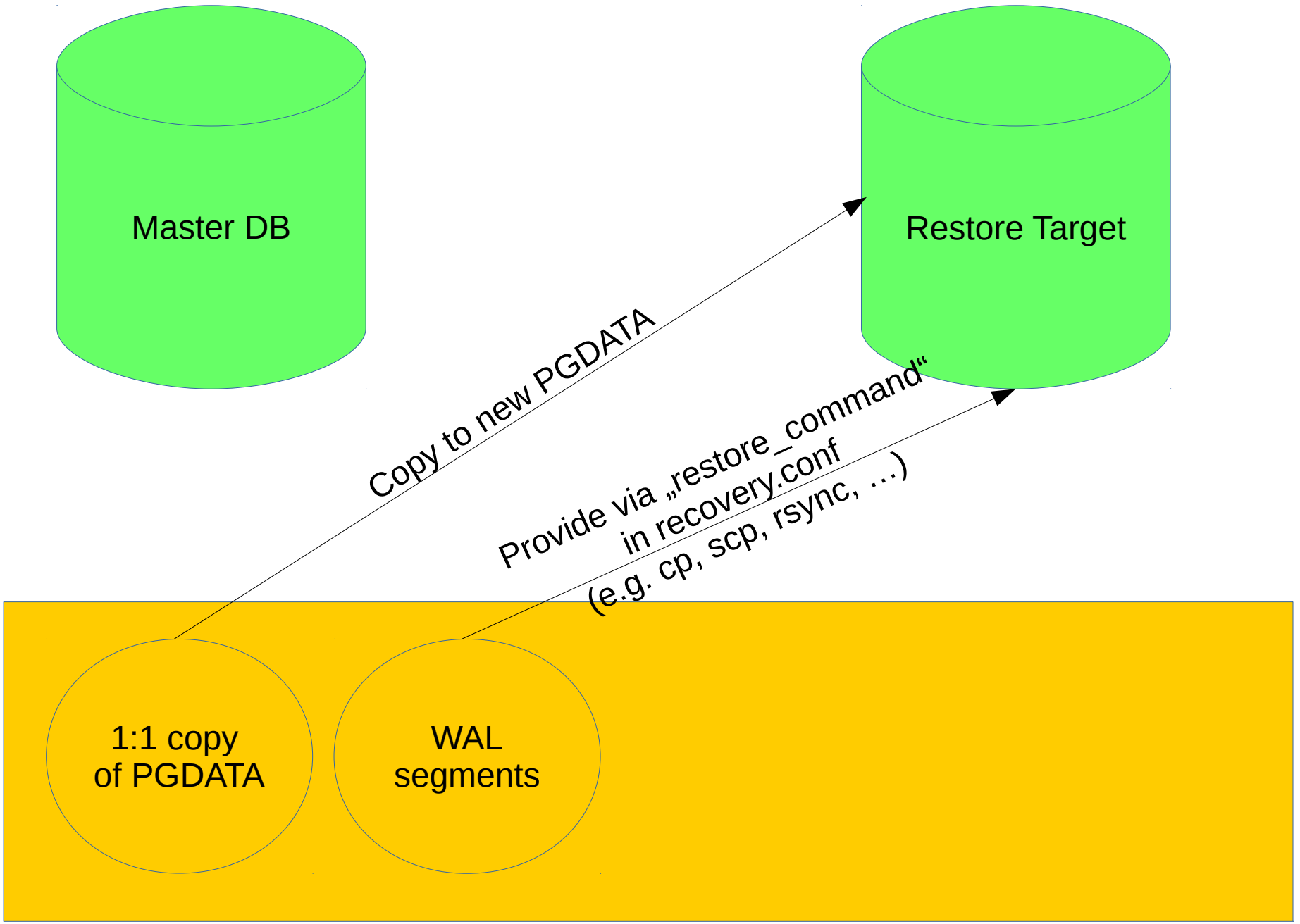
snapshot

1:1 copy
of PGDATA

Ok, anything more that I need?

- Oh yes!
- All the WAL segments since the pg_start_backup()!
- Hopefully, they are still there, eh?
  - If you wrote a lot of data into your DB after pg_start_backup(), they might have been recycled already!*


Master DB
archive_command
1:1 copy
of PGDATA
WAL
segments


Master DB
Restore Target
Copy to new PGDATA
Provide via „restore_command“
in recovery.conf
(e.g. cp, scp, rsync, …)
1:1 copy
of PGDATA
WAL
segments

RTO & RPO

- RTO
  - between minutes & days
  - depending on size & activity during backup
- RPO
  - the end of your backup
  - or the end of the one before!*

Pros and cons

+ 1:1 copy of your DB
+ rather failsafe
+ rather fast
+ RTO fine

- can only back up and thus restore a single point in time
- can only back up and thus restore whole DB clusters
- RPO... still, uhm, well

Options to get that snapshot

- LVM / filesystem snapshot
- rsync
- pg_basebackup

Options to get that snapshot

- ~~LVM / filesystem snapshot~~
- ~~rsync~~
- pg_basebackup

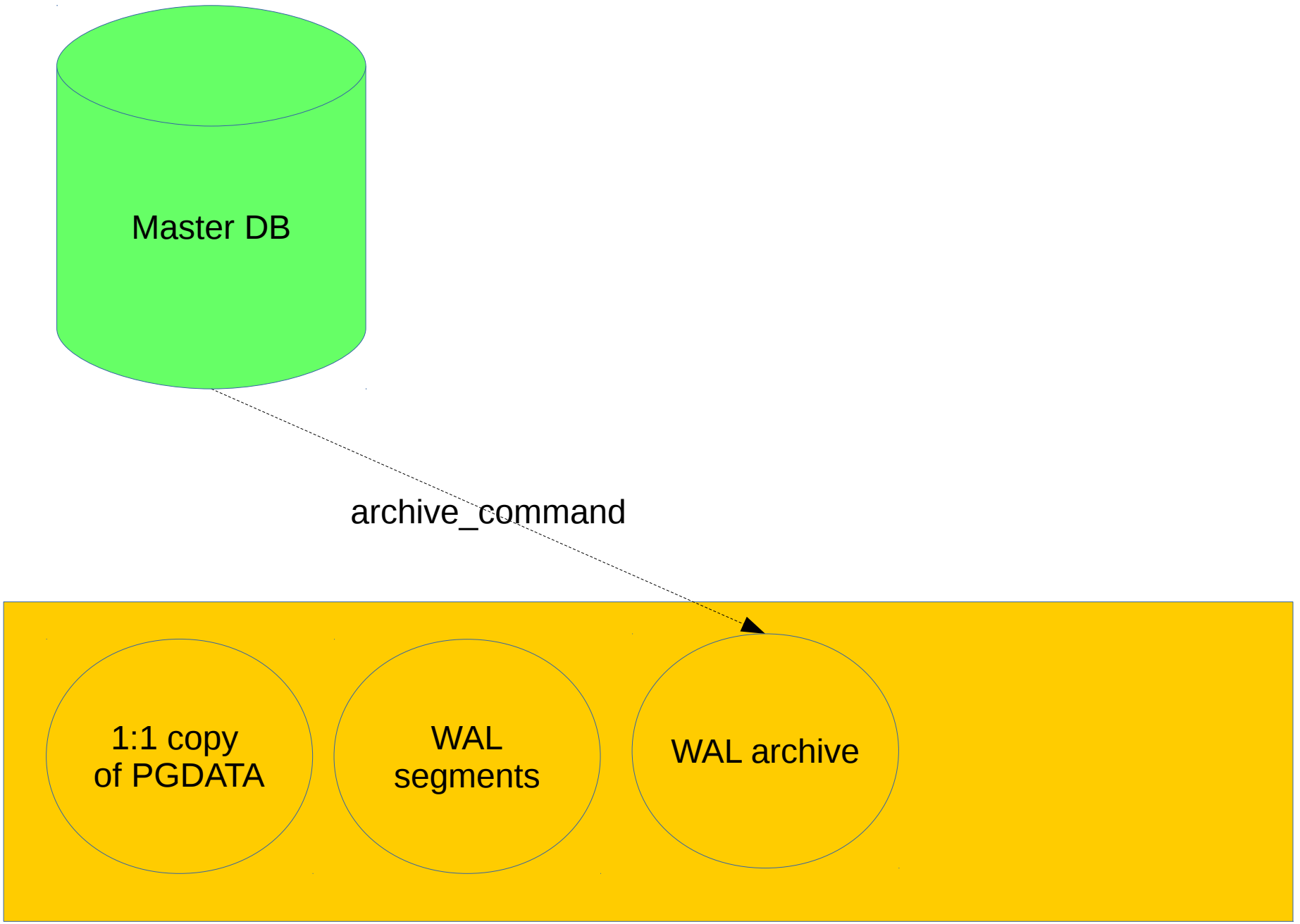
Options to get the WAL segments

- archive_command (postgresql.conf)
- pg_basebackup
  - With --xlog-method=[fetch|stream]
  - -X [s|f]

USE BOTH!

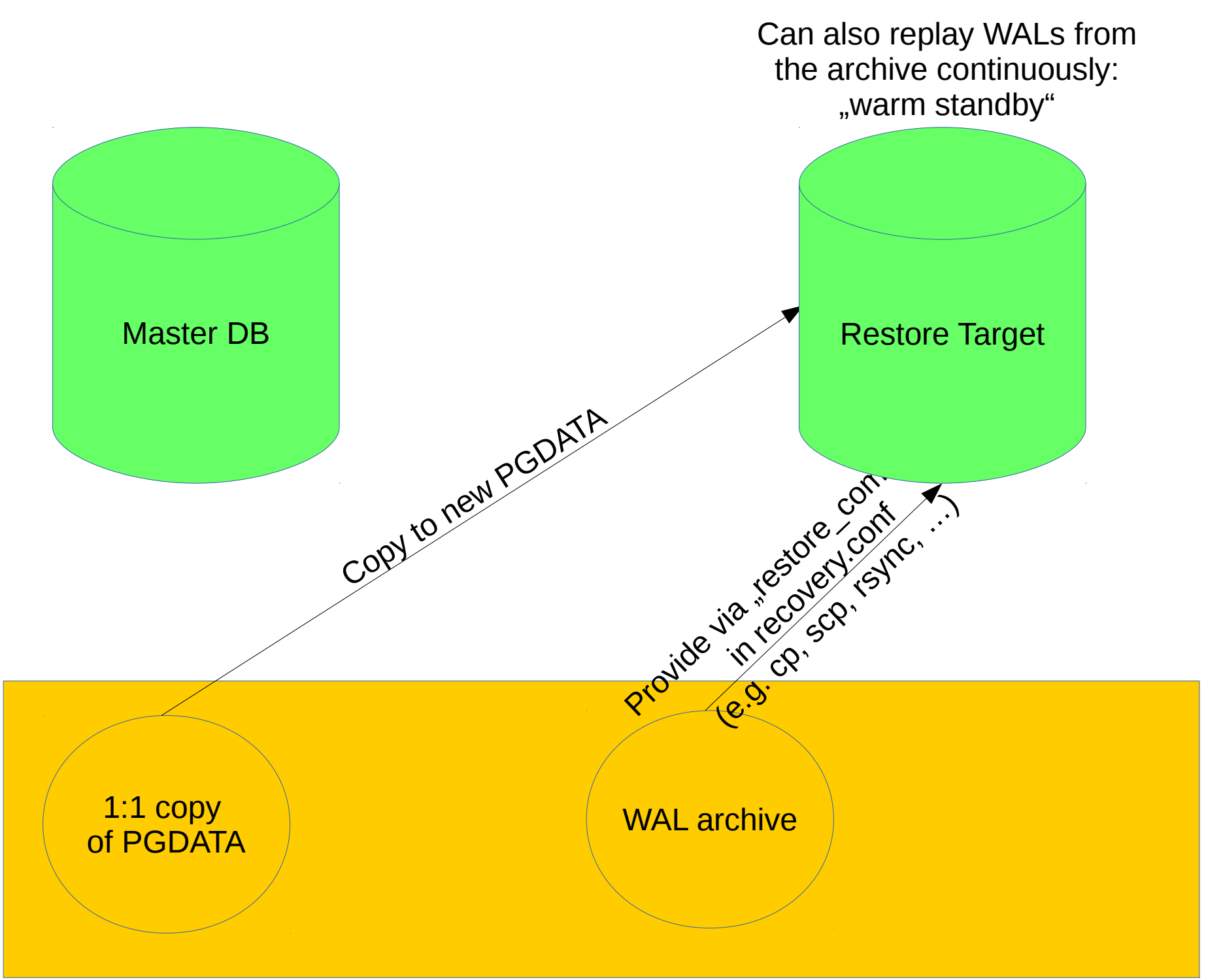
Why use both?

- Actually, get used to both
- When you have a WAL archive anyway, you can (probably) rely on that
- But pg_basebackup with -X is also handy to clone new slaves (we'll get there)


Master DB
archive_command
1:1 copy
of PGDATA
WAL
segments
WAL archive

Why do I want to have a WAL archive?

- The WAL segments, together with the snapshot of your HEAP, allow you to restore your database to any point in time
  - e.g., the moment right before you forgot the WHERE in your „DELETE FROM customers“ statement ;-) **
- That‘s Point In Time Recovery („PITR“)
- Obviously, you need two things for that:
  - a binary snapshot of your HEAP
  - all WAL segments between your snapshot and your mistake


Can also replay WALs from
the archive continuously:
„warm standby“
Master DB
Restore Target
Copy to new PGDATA
Provide via „restore_com
in recovery.conf
(e.g. cp, scp, rsync, …)
1:1 copy
of PGDATA
WAL archive

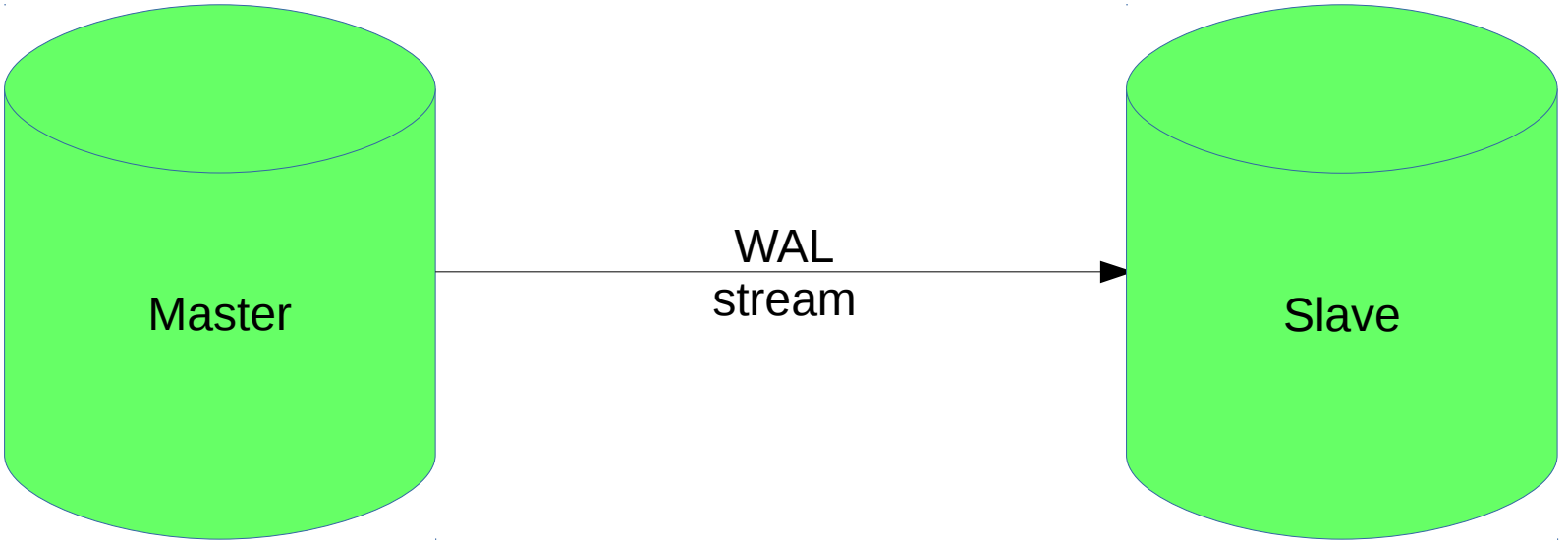
RTO & RPO

- RTO
  - minutes to hours (cold standby)
  - seconds (warm standby)
- RPO
  - your last archived WAL segment

- warm standby = „poor man's replication“

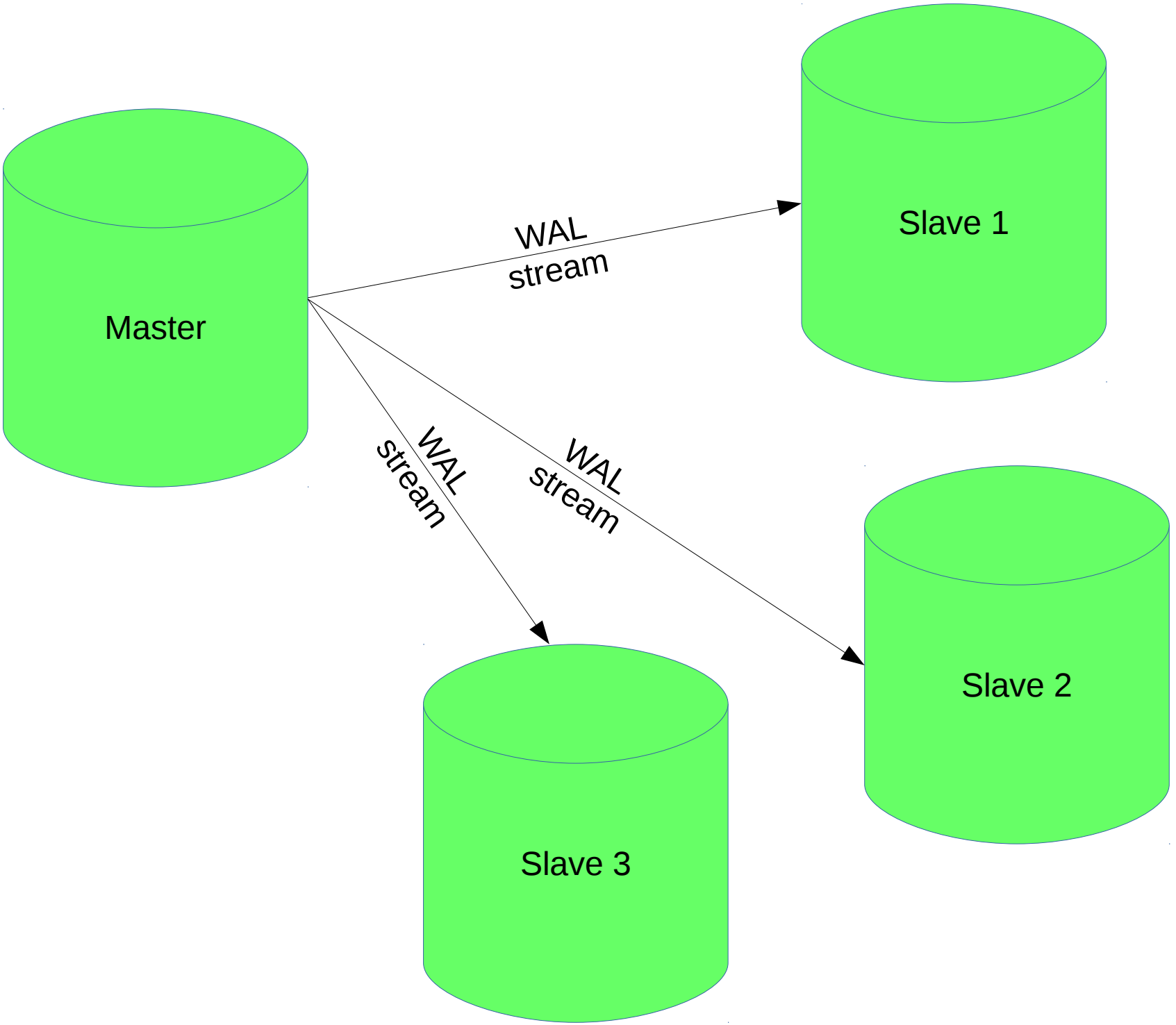
Binary streaming replication

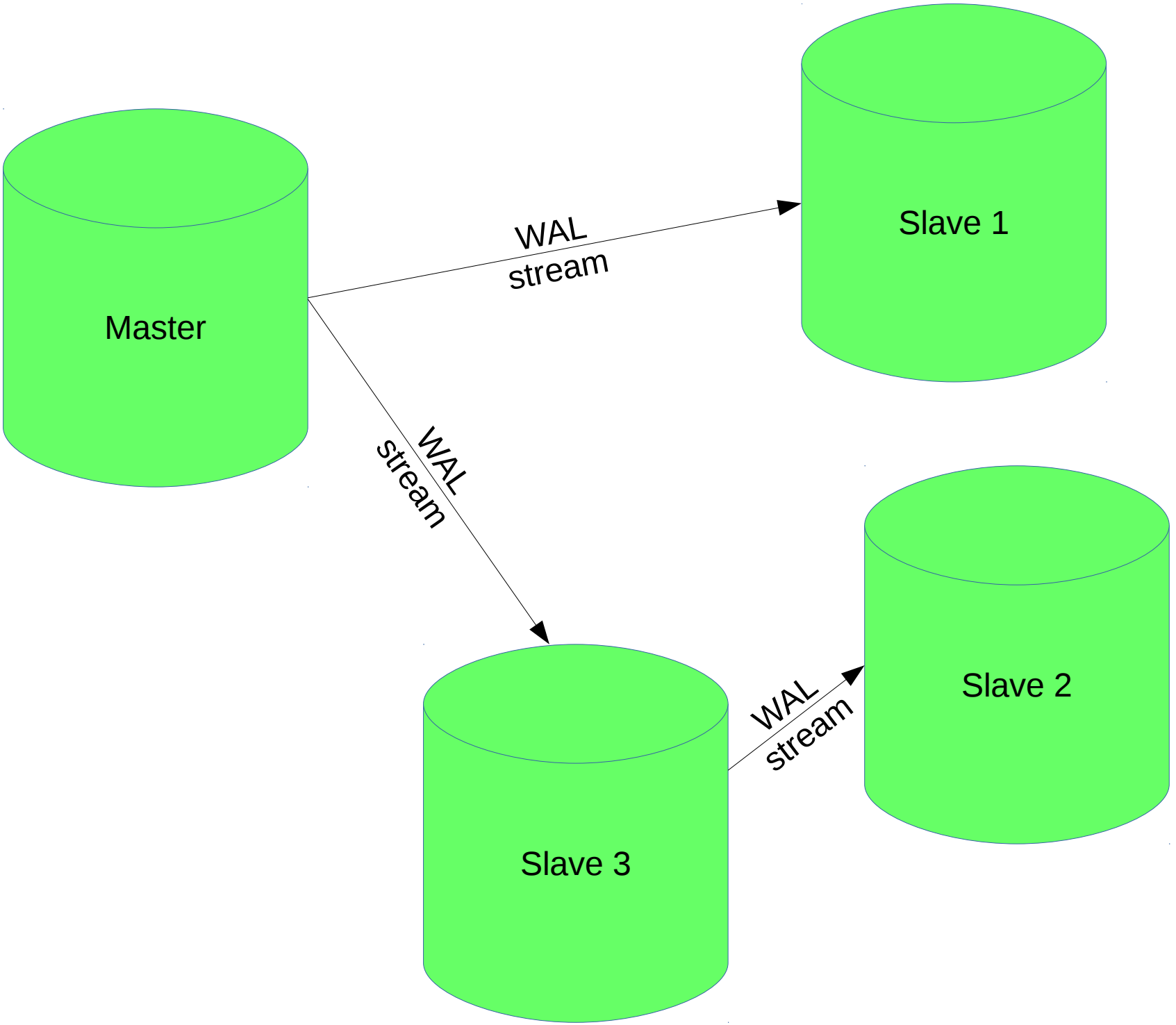
- Binary streaming is like a warm standby server as seen before
- But the WAL segments get sent over the network directly
- Transactions are replayed immediately
  - i.e., „ASAP“


Master
WAL
stream
Slave

Let's talk about options!

- Streaming replication can be synchronous or asynchronous
  - choose per transaction!
  - choose between remote_write & remote_apply
- *can* use replication slots
- *can* be cascaded
- slaves *can* serve RO queries
  - you *can* take your backup from a slave (
- Streaming slave can be delayed (so you can still press the big red button) **


Master
Slave 1
Slave 2
Slave 3
WAL
stream
WAL
stream
WAL
stream


Master
Slave 1
WAL
stream
WAL
stream
Slave 3
WAL
stream
Slave 2

Sync replication pitfalls

- You can now have N sync slaves
- Make sure you always have N+1 slaves in total
  - If you go to N-1, your DB will still work
  - but not finish any transactions before you get back to N! *
- Network latency / roundtrip time becomes an issue!
  - so choose wisely (you can!) which transactions should by sync
  - and where to put your sync slave

Pros and cons

+ 1:1 copy of your DB, online
+ Reliable & battle proven
+ RTO & RPO very good
+ very flexible

- works on whole DB clusters only
- implications on network connection loss

So, with replication,...

- I don't need the WAL archive anymore, right?

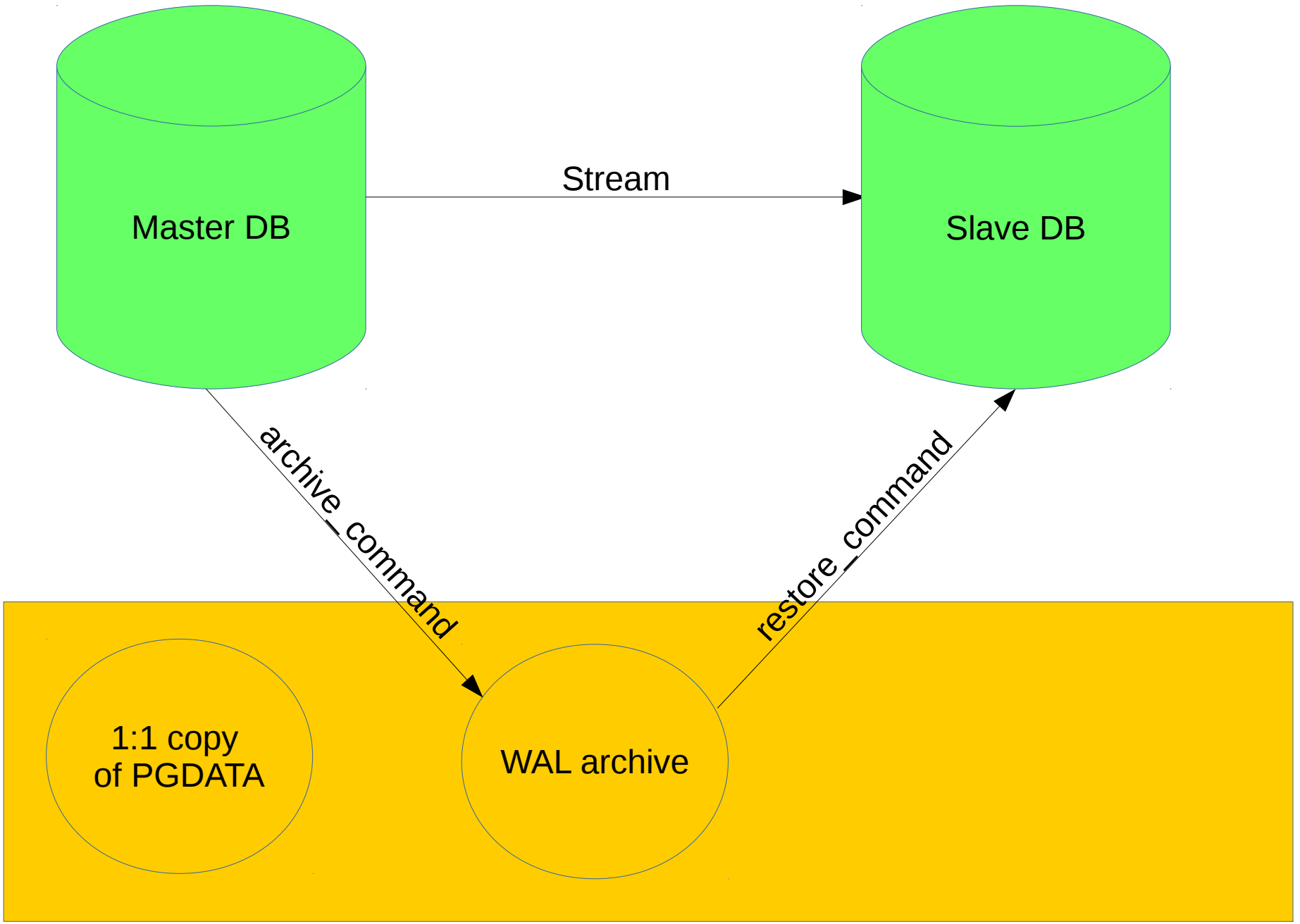
R U effing kidding me?!?

We need to talk...

- Replication does not replace backup
- And, while we're on it: **
- RAID does not replace backups
- SAN does not replace backups
- „The Cloud“ does not replace backups **

Putting it all together

- You want to have a WAL archive
- You want to have (a) replication slaves
  - maybe more than one
  - maybe a sync one
  - maybe a delayed one
  - maybe cascaded
- RTO: minimal
- RPO:
  - closest possible (sync slave)
  - closest feasible (async slave)
- Protection against human errors (RTO obviously rises...)
- Allow read only queries on slave(s)


Master DB
Stream
Slave DB
archive_command
restore_command
1:1 copy
of PGDATA
WAL archive

Pros and cons

+ all of replication
+ all of WAL archive

- major version still has to be the same

Configure postgresql.conf

- `wal_level = replica # or logical already`
- `archive_mode = on   # always to cascade`
- `archive_command = /your/archive_script.sh %p %f`
- `max_wal_senders = 10 # or more`
- `max_replication_slots = 10 # or more`
- `synchronous_commit = local # for now`
- `synchronous_standby_names = '' | <set>`
- `hot_standby = on`
- `log_collector = on`

Set up your WAL archive

- Don't roll your own! **
  - Use pgbarman, pgbackrest, WAL-E, …
  - Follow their instructions
- Invest the saved time in thinking about redundancy, persistance and data safety
- Your DB server is not a good place to keep your archive **
- Even the same datacenter is a bad choice (unless you mirror)

/your/archive_script.sh

- Only slightly complex functionality will not fit in archive_command
- A script can be changed w/out HUPing the DB
- Purpose of the script: somehow get %p ($1) to your WAL archive as %f ($2)
- rsync is not a bad choice, however:
  - make sure %f does not exist in the archive yet before you start sending
  - call sync remotely (or mount your archive sync) after sending
  - rsync tends to give RCs > 127, filter these
- Make sure it never, ever returns RC=0 w/out having done the job
  - Unless you're still setting everything up
  - „set -e“ etc.
  - Errors will end up in PG's log (as we turned log_collector on)

Let me repeat that

- You are most probably writing into some OS pagecache, and potentially async on top (NFS)!
- Your backup is not safe until it has been flushed to persistent storage in a safe location *
- Your archived WAL segments are not safe until they have been flushed to persistent storage in a safe location *
- You'll probably make some compromises, but keep the implications on the RCO in mind **

Now, activate archiving

- And watch it
- PG will not throw away WAL segments it could not archive
  - your PGDATA can run out of disk space!
- Replication slots have the same implication, so keep that in mind

Now, try a full backup

- Since you're using a tool anyway**, you're hopefully ready to go already (rights, replication permission, preparation, ...)
- E.g., do
  - `barman backup all`

Doing your first slave

- Add a „replication“ line to your master's `pg_hba.conf`
- Prepare the new PGDATA
  - e.g. on Debian/Ubuntu do a `pg_createcluster` and `rm -rf` the result (no, really)
  - Make sure the `postgresql.conf` etc. match your master's
- Run
- `pg_basebackup -X stream -h <master> -U <user> -R -D <new_pgdata>`
- Add a restore_command to the resulting `recovery.conf`
  - Which gets the segment from your archive
- Start the slave, enjoy, rinse, repeat

Now, start looking for software

- E.g.
  - repmgr
  - PAF
  - pglookout
  - ...

Logical replication

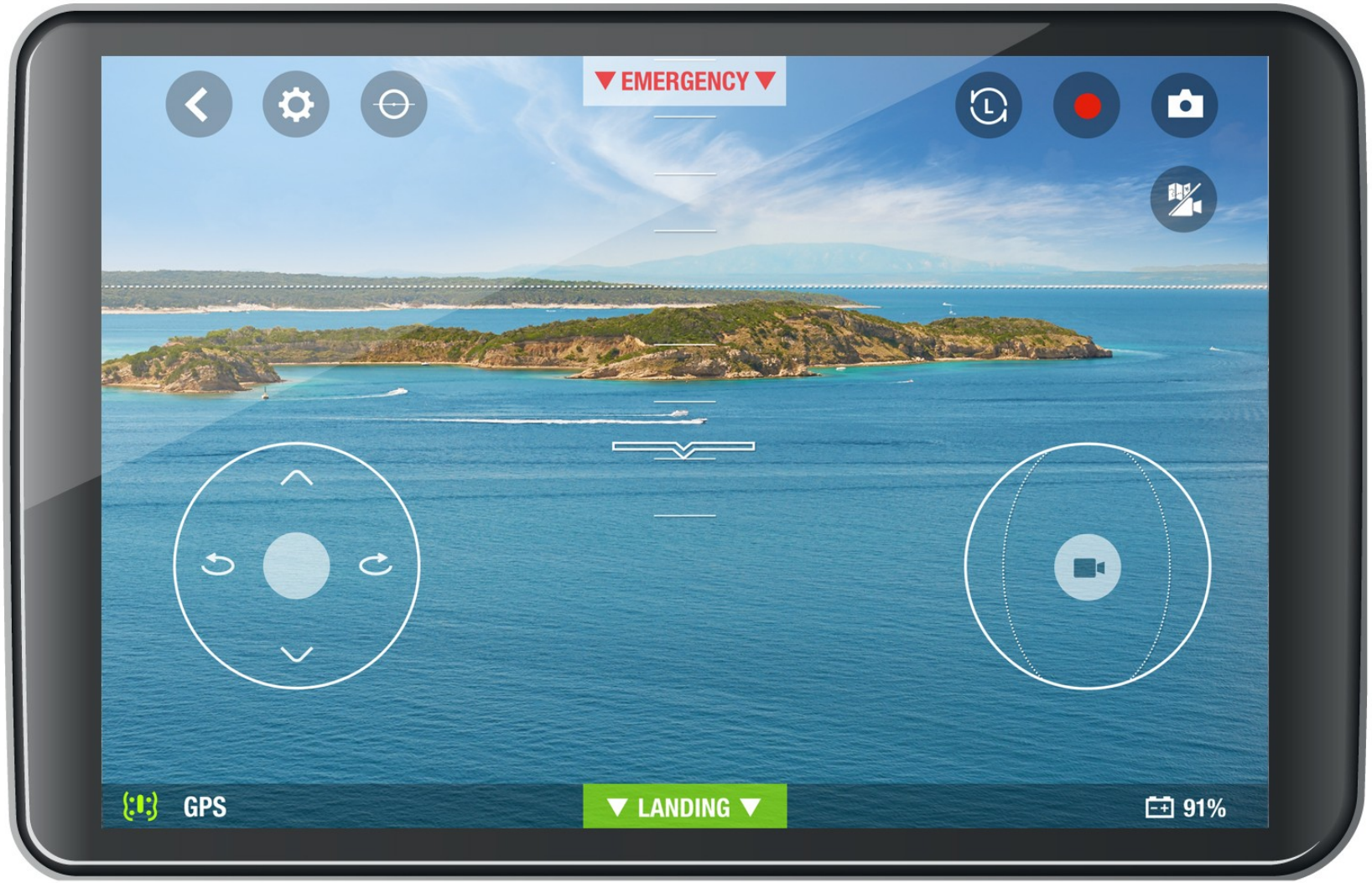
- In the not so recent past...
-

Logical replication

- Coming into core with 10.0
- Already available with e.g. pglogical
- If you can afford a few MB extra backup volume, already set
  - `wal_level = logical`
- Allows for e.g.
  - painless, low-downtime version upgrades
  - sharding
  - collecting data from different DBs in a DWH
  - multi-master
  - ...

When in core

- Somewhat moving target yet, but will be more like



Famous last words

- Don't reinvent the wheel!
- Test your backup procedure!
- Test your restore procedure!!! **
- Monitor your logs and your lags!
- Make sure your configs are in sync!
- Make sure everybody in your team understands your backup and restore procedures! **
- In case of disaster *
  - keep calm and follow your procedures **

Thank you for your attention!