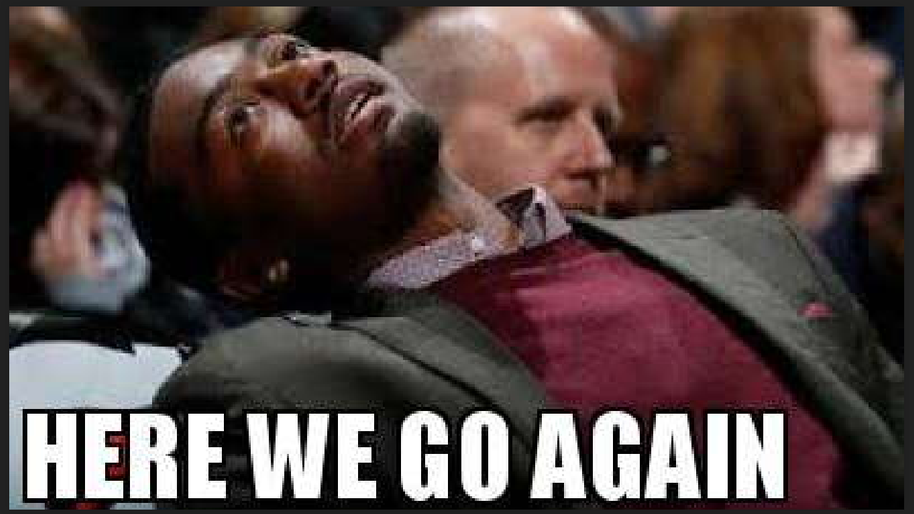
PAF

...Another brick in the HA wall

WHO AM I ?

- Jehan-Guillaume de Rorthais
- aka. ioguix
- using PostgreSQL since 2007
- involved in PostgreSQL community since 2008
- @dalibo since 2009

High Availability



MENU

- Quick intro about HA
- Quick intro about Pacemaker
- Why PAF ?
- PAF abilities

HA IN SHORT

- Part of the Business Continuity Planning
- In short: double everything
- should it include automatic failover ?

AUTO FAILOVER: TECH

Hard to achieve:

- how to detect a real failure?
- why the master doesn't answer?
- is it under high load? switched off?
- is it a network issue? hick up?
- how to avoid split brain?

BUILDING AUTO FAILOVER

- many issues to understand
- solutions: quorum, fencing, watchdog, ...
- complex setup
- complex maintenances
- document, document, document
- test, test, test

If you don't have time, don't do auto failover (almost).

QUORUM

- resources run in the cluster partition hosting the greater number of nodes
- useful on network split
- ...or when you require a minimal number of node alive
- based on vote

FENCING

- ability to poweroff or reboot any node of your cluster
- the definitive solution to know the real state of an unresponsive node
- hardware fencing (smart PDU, UPS, IPMI)
- IO fencing (SAN, network)
- virtual fencing (libvirt, xen, vbox, ...)
- software: do not rely on it (eg. ssh)
- meatware

Really, do it. Do not think you are safe without it.

WATCHDOG

- feed your local dog or it will kill your node
- either hardware or software (cf. softdog)
- self-fencing (suicide) on purpose
- auto-self-fencing when node is unresponsive

Pacemaker

Will assimilate your resource...

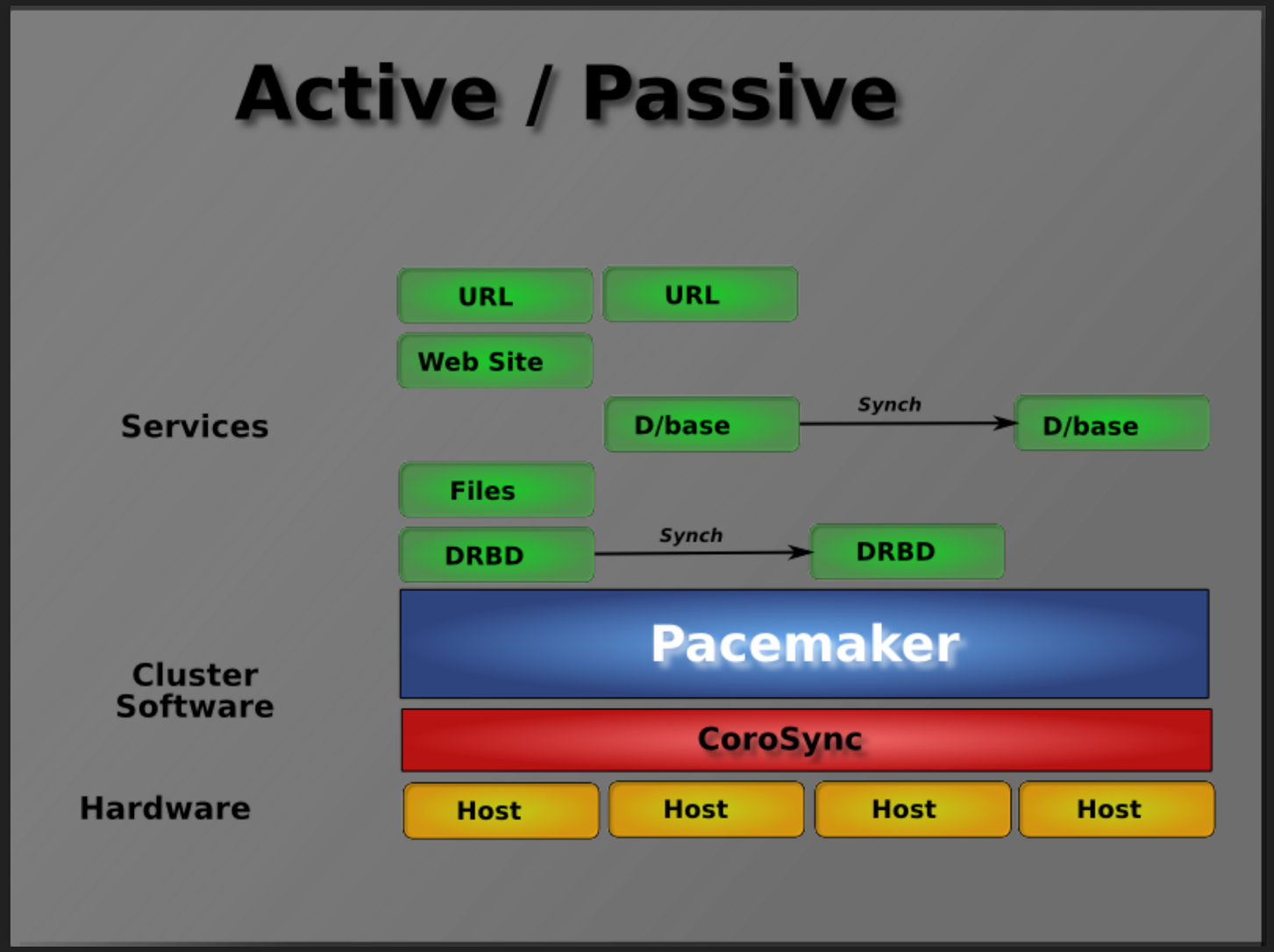
Resistance is futile.

(T.N.: 'service' eq 'resource')

PACEMAKER IN SHORT

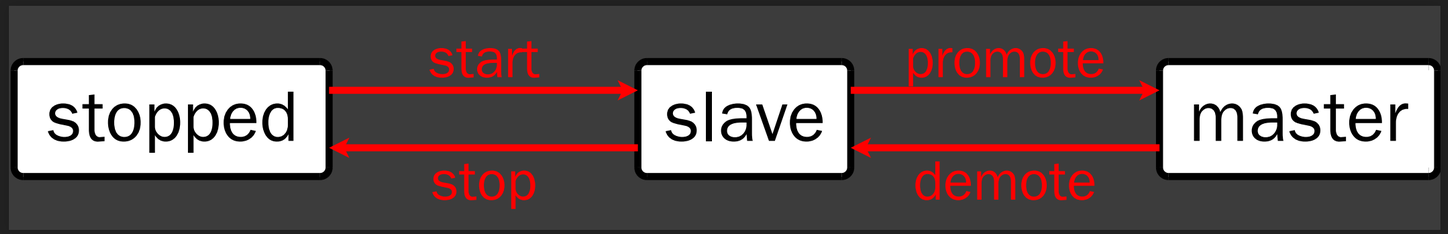
- is a "Cluster Resource Manager"
- support fencing, quorum and watchdog
- multi-resource, dependencies, resources order, constraints, rules, etc
- **Resource Agents** are the glue between the CRM and the services
- RA can be stateless or multi-state
- RA API: script OCF, upstart, systemd, LSB

PACEMAKER ARCHITECTURE



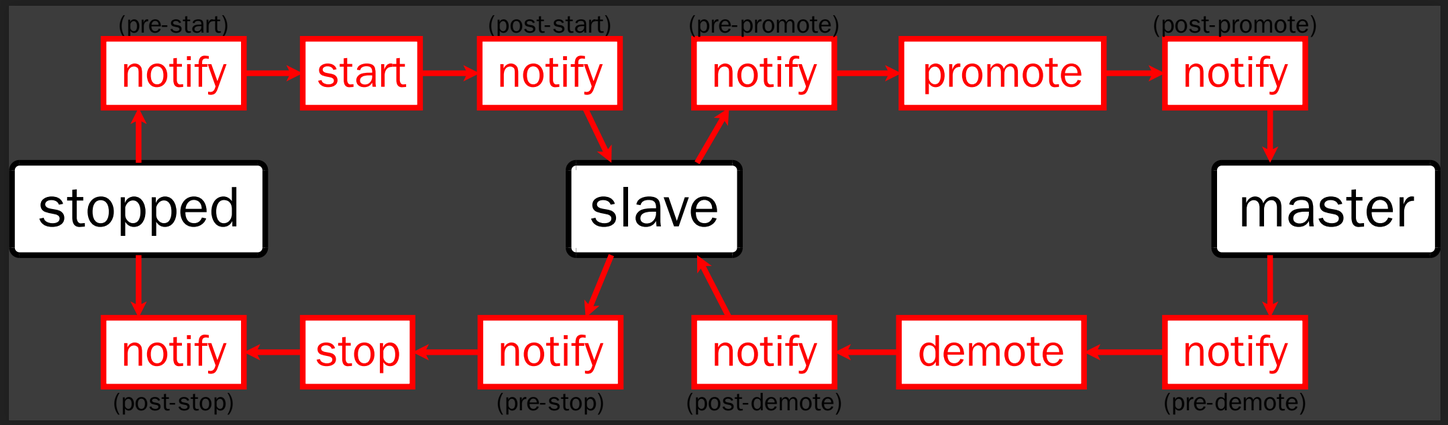
CRM MECHANISM

- kind of automate
- 4 states: stopped, started, slave or master
- the CRM compute transitions between two states
- only **ONE** CRMd is handling the whole cluster: the DC
- minimal actions API (eg. systemd): start, stop, monitor(status)
- extended actions API (OCF): start, stop, promote, demote, monitor, **notify**
- for a multi-state resource:



NOTIFY ACTION

- only available with OCF resource agents
- triggered before and after actions
- triggered on **ALL** node
- action wait for all pre-notify feedback to run
- next actions wait for all post-notify feedback to run
- allows the resource agent to run specific service code



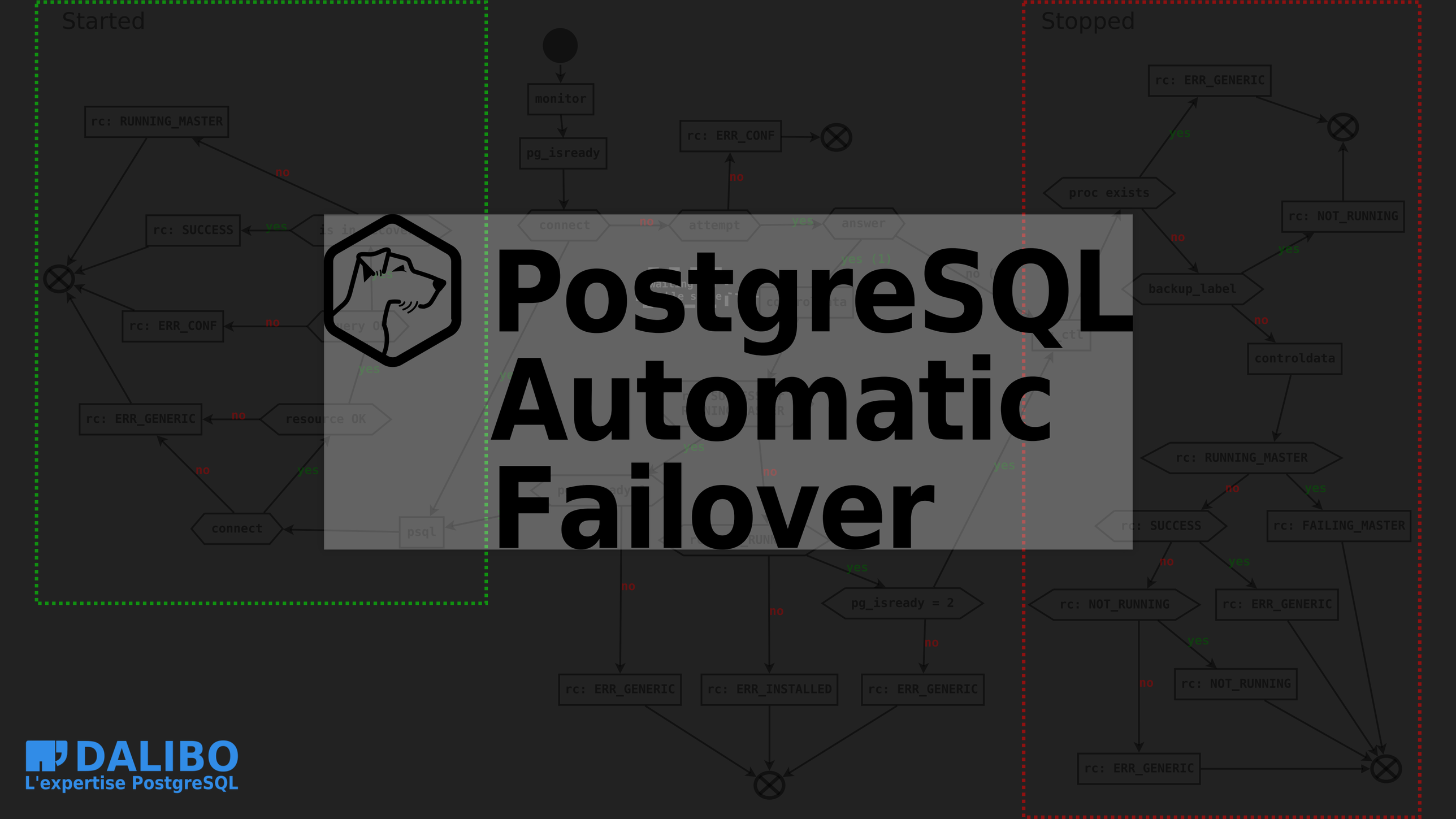
NOTIFY DATAS

Datas available to the RA during the notify actions:

```
active    => [ ],
inactive  => [
               { rsc => 'pgsqld:2', uname => 'srv1' },
               { rsc => 'pgsqld:0', uname => 'srv2' },
               { rsc => 'pgsqld:1', uname => 'srv3' }
             ],
master    => [ ],
slave     => [ ],
promote   => [ { rsc => 'pgsqld:0', uname => 'srv1' }],
demote    => [ ],
start     => [
               { rsc => 'pgsqld:0', uname => 'srv1' },
               { rsc => 'pgsqld:1', uname => 'srv3' },
               { rsc => 'pgsqld:2', uname => 'srv2' }
             ],
stop      => [ ],
type      => 'pre'
```

MASTER SCORE

- set preference on slave to promote
- highest score is promoted to master
- a slave must have a positive score to be promoted
- no promotion if no master score anywhere
- set by the resource agent and/or the admin



PostgreSQL Automatic Failover

DALIBO
L'expertise PostgreSQL

HISTORY

- pgconf.eu 2012 talk on Pacemaker/pgsql
- had a hard time to build a PoC and document
- discussion with Magnus about demote
- (other small projects around this before)
- PAF started in 2015
- lots of questions to Pacemaker's devs
- authors: Maël Rimbault, Me
- some contributors and feedbacks (Thanks!)

WHY ?

The existing RA:

- achieve multiple architectures (stateless and multistate)
- implementation details to understand (lock file)
- only failover (no role swapping or recovery)
- hard and heavy to manage (start/stop order, etc)
- hard to setup
- fake Pacemaker state because of demote, mess in the code
- old code...

GOALS

- keep Pacemaker: it does most of the job for us
- focus on our expertise: PostgreSQL
- stick to the OCF API and Pacemaker behavior, embrace them
- keep a **SIMPLE** RA setup
- support **ONLY** multi-state
- support **ONLY** Streaming Replication
- **REQUIRE** Streaming Replication and Hot Standby
- ease of administration
- keep the code clean and documented
- support PostgreSQL 9.3 and after

VERSIONS

Two versions to catch them (almost) ALL!

- **1.x**: up to EL6 and Debian 7
- ...or Pacemaker 1.12/corosync 1.x
- **2.x**: from EL7 and Debian 8
- ... or Pacemaker 1.13/Corosync 2.x

GUTS

- written in perl
- demote = stop + start (= slave)
- slave election during failover
- detect various kind of transitions thanks to notify (recover and move)

PAF CONFIGURATION

- system_user
- bindir
- datadir (oops, 1.1 only)
- pgdata
- pghost
- pgport
- recovery_template
- start_opts

OLD CONFIGURATION

Compare with historical `pgsql` RA:

- pgctl
- start_opt
- ctl_opt
- psql
- pgdata
- pgdba
- pghost
- pgport
- pglibs
- monitor_user

OLD CONFIGURATION (2)

Encore?

- monitor_password
- monitor_sql
- config
- pgdb
- logfile
- socketdir
- **stop_escalate**
- **rep_mode**
- **node_list**
- restore_command

OLD CONFIGURATION (3)

Not done yet...

- archive_cleanup_command
- recovery_end_command
- master_ip
- repuser
- primary_conninfo_opt
- **restart_on_promote**
- **replication_slot_name**
- tmpdir
- **xlog_check_count**
- **crm_attr_timeout**

OLD CONFIGURATION (4)

Promise, the last ones:

- stop_escalate_in_slave
- check_wal_receiver

Features

The following demos considers:

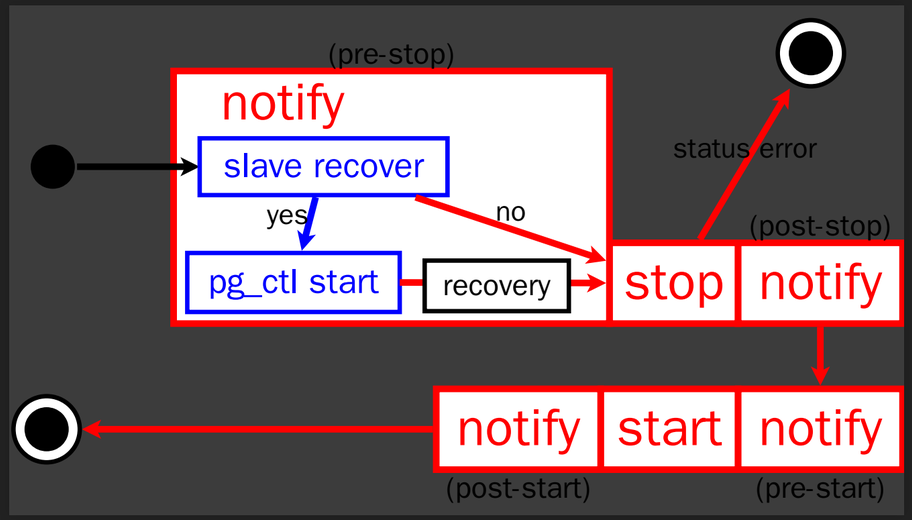
- one master & two slaves
- a secondary IP address following the master role: 192.168.122.50
- a really simple recovery.conf template file:

```
standby_mode = on
primary_conninfo = 'host=192.168.122.50 application_name=$(hostname -s)'
recovery_target_timeline = 'latest'
```

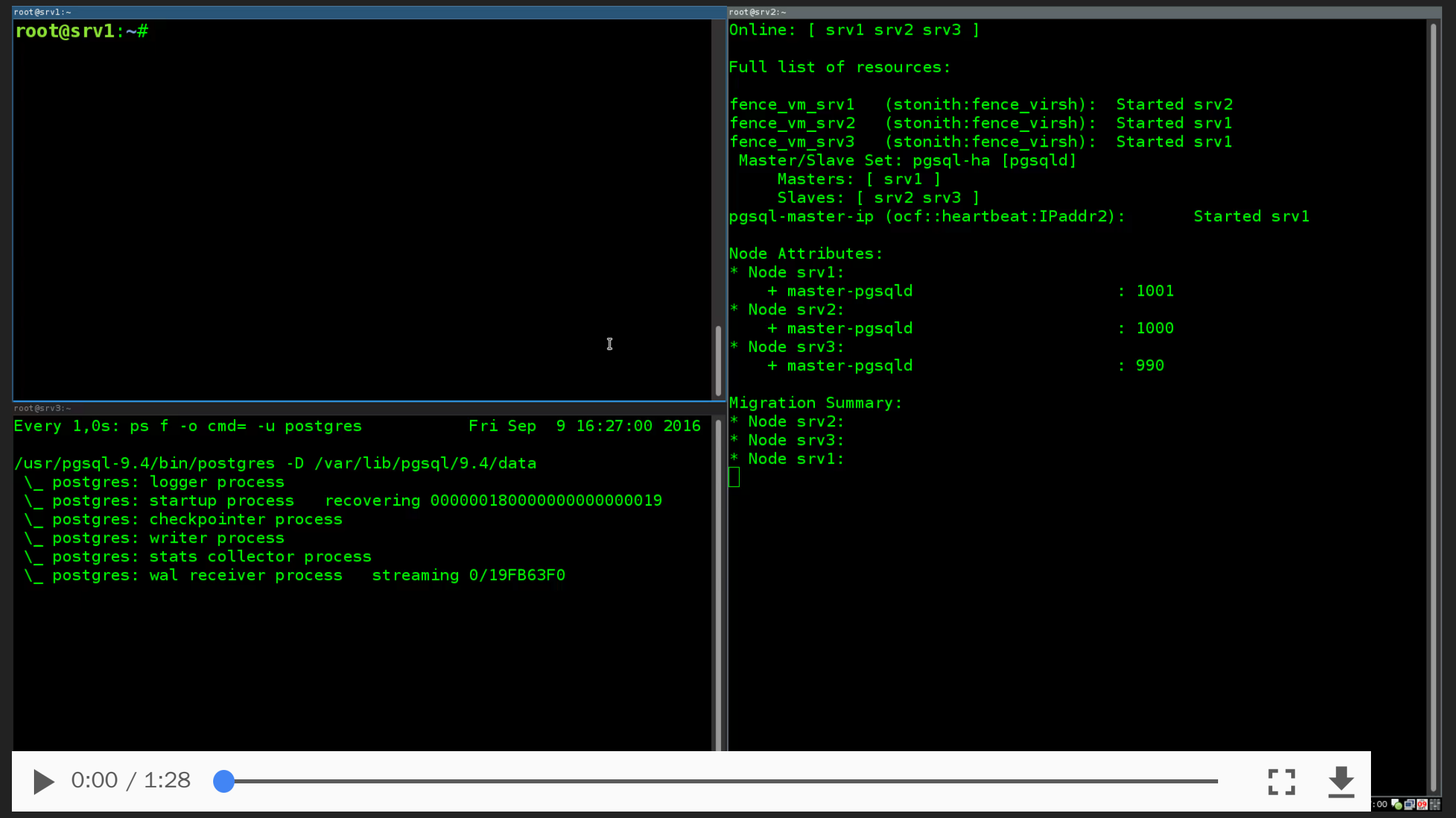
- a monitor action every 15s

STANDBY RECOVER

Transition: `stop` -> `start`



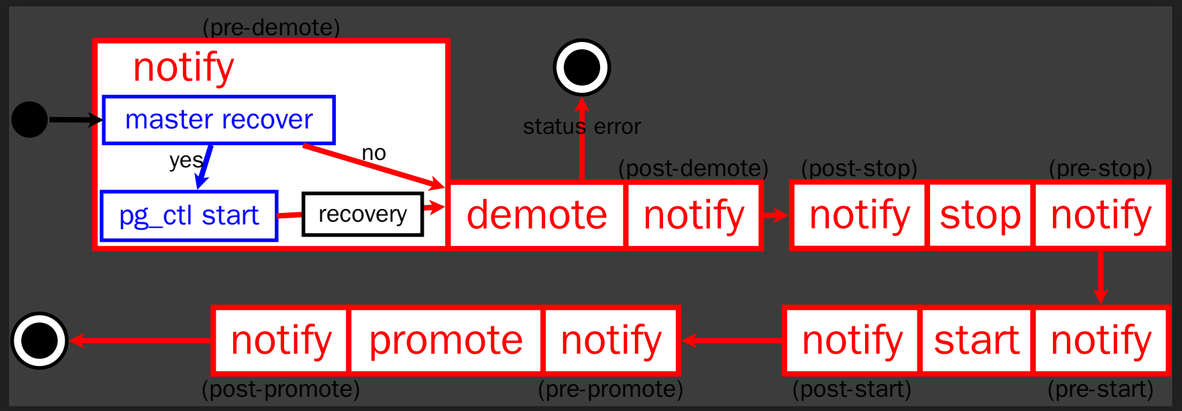
Slave recover demo:



DALIBO
L'expertise PostgreSQL

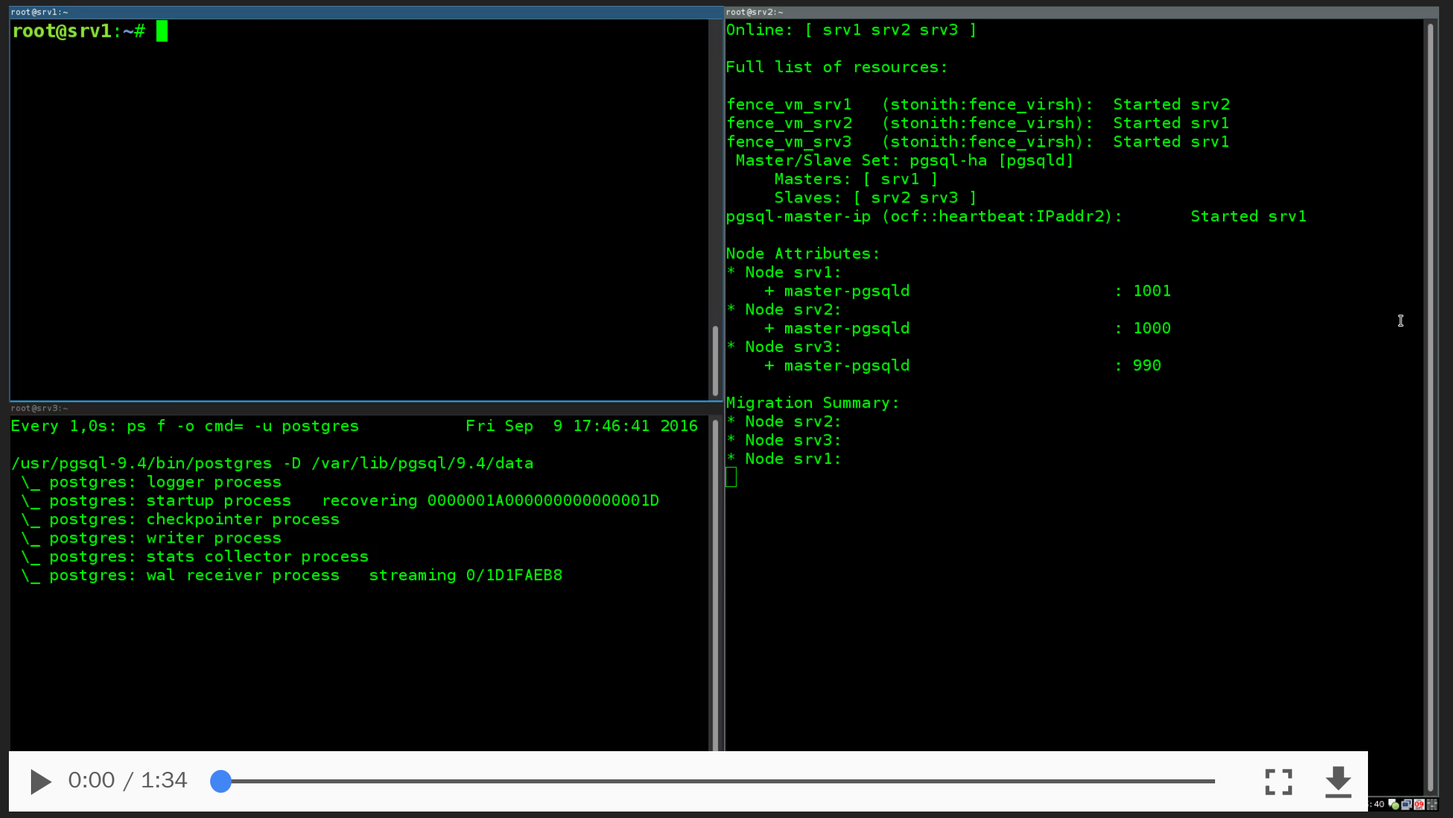
MASTER RECOVER

Transition: `demote` -> `stop` -> `start` -> `promote`



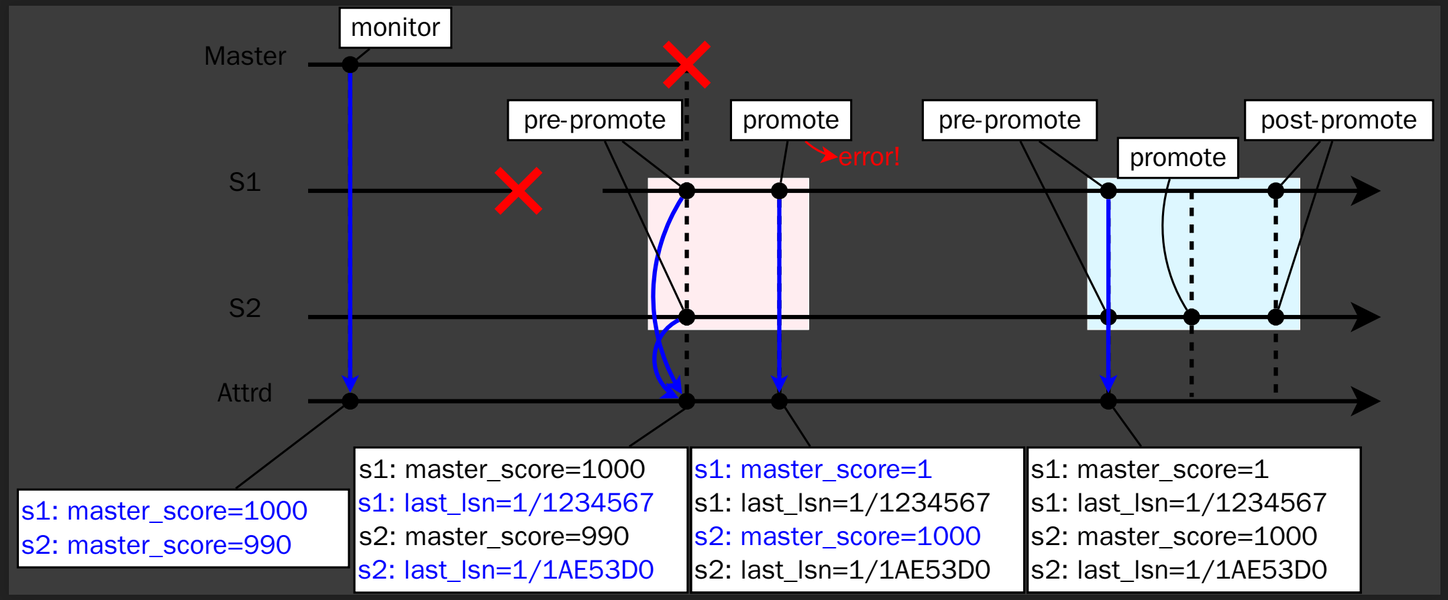
DALIBO
L'expertise PostgreSQL

Master recover demo:

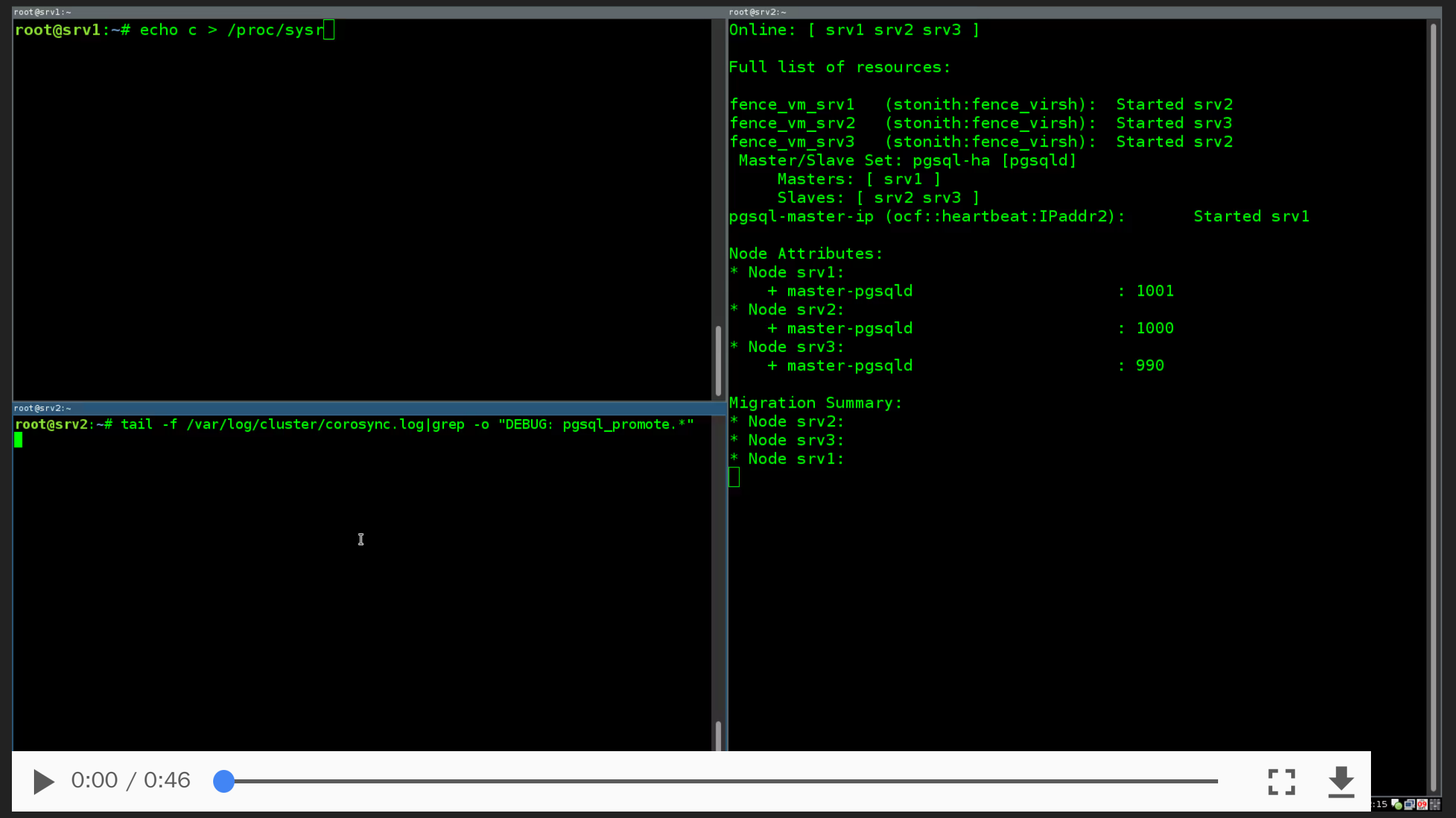


DALIBO
L'expertise PostgreSQL

FAILOVER & ELECTION



Failover demo:



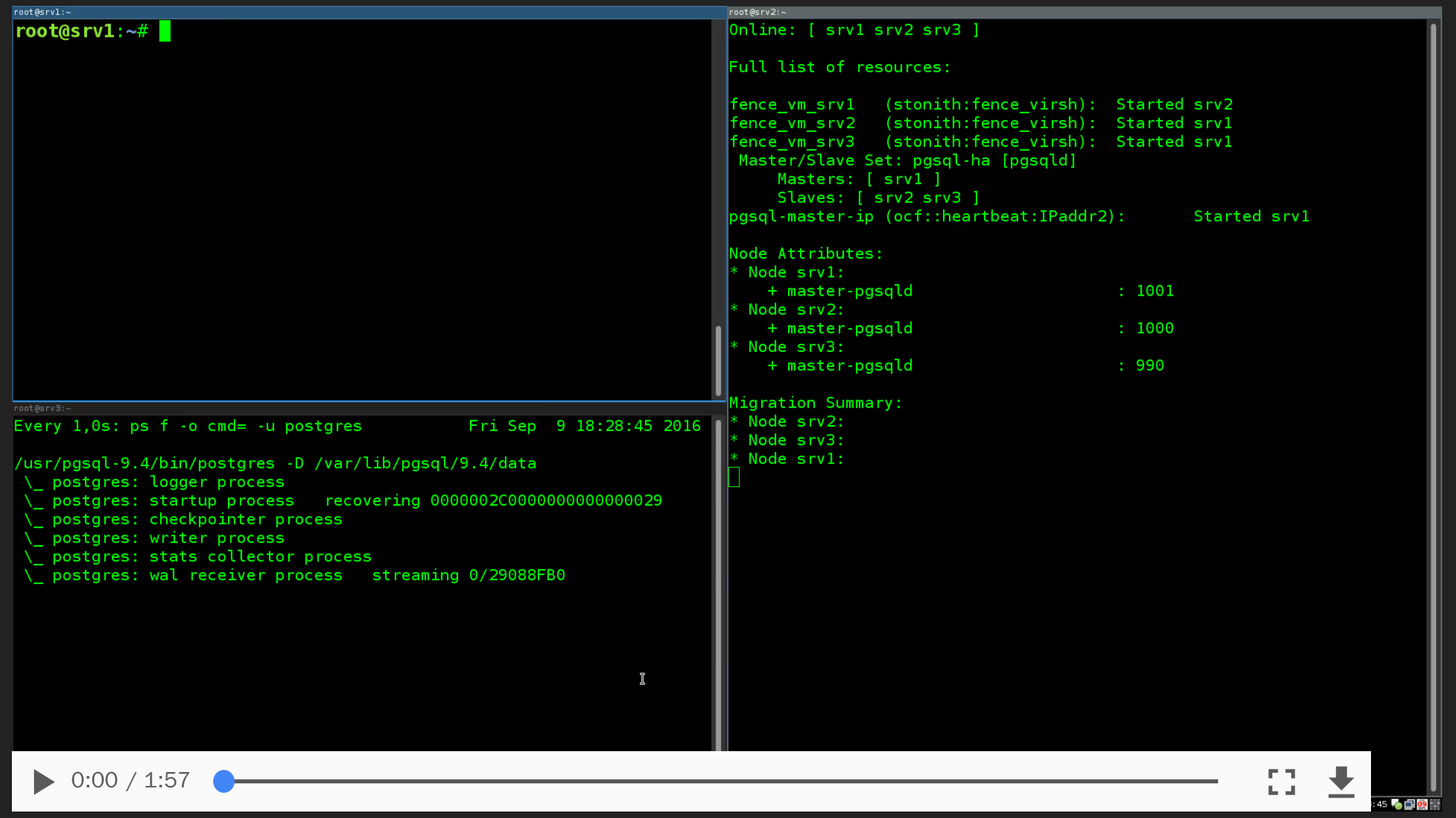
DALIBO
L'expertise PostgreSQL

YOU THINK IT'S OVER ?

CONTROLLED SWITCHOVER

- only with 2.0
- the designated standby checks itself
- it cancel the promotion if the previous master will not be able to catch up with it.

Controlled switchover demo:



DALIBO
L'expertise PostgreSQL

Whishlist

- recovery.conf as GUC
- live demote
- pgbench handling of errors

Where?

- site officiel: http://dalibo.github.io/PAF/
- code: https://github.com/dalibo/PAF
- packages: https://github.com/dalibo/PAF/releases
- support: https://github.com/dalibo/PAF/issues
- mailing list: pgsql-general