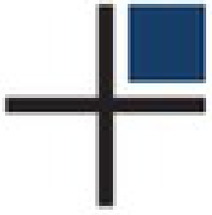
PGDay UK

High Availability with repmgr 3.1

Gianni Ciolli

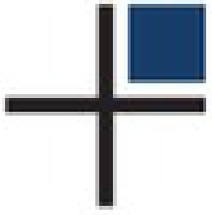
2ndQuadrant
Professional PostgreSQL

London
5 July 2016



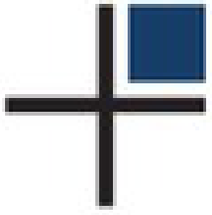
Topics

- High Availability PostgreSQL cluster
- Integration
  - `repmgr` with PgBouncer
  - `repmgr` with Barman



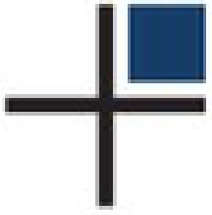
Programme

- Software
- Architecture
- Automation



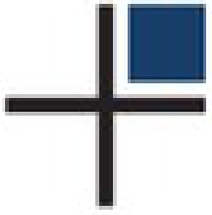
`repmgr` Overview

- Clusterware for PostgreSQL replication
- Open source (GPL)
- Latest release: 3.1.3
  - Released on 27 May 2016
- `http://www.repmgr.org/`



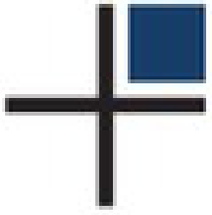
`repmgr` Features

- Monitoring
- Automatic Failover
- Base Backup with rsync or `pg_basebackup`
- "Follow" without restart
- Cascaded Replication support
- Replication Slots support
- Event Notification Log and Commands



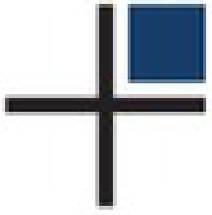
PgBouncer Overview

- Connection Pooler
- Open Source (BSD)
- Latest release: 1.7.2
  - Released on 26 February 2016
- `http://pgbouncer.github.io/`



PgBouncer Features

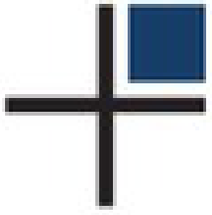
- Connection Pooling
- Connection Concentration
- Lightweight
- Simple
- Flexible
- `PAUSE`, `RESUME`
  - Restart ("bounce") the server smoothly!



Surprise!

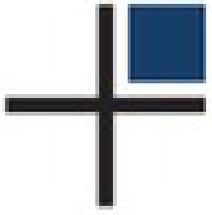
- We also mention **Barman**
  - **Ba**ckup and **R**ecovery **Man**ager
- Why?
- No Production Without Backup!

No Production Without Backup!



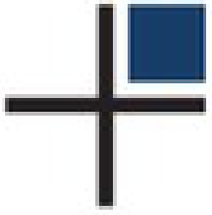
More generally…

- The primary node will change
  - Failover, Switchover, maintenance…
- The primary node will require maintenance
  - Automatic maintenance (cronjob)
  - A fixed alias is useful
- We mention "Barman" thinking about all these processes



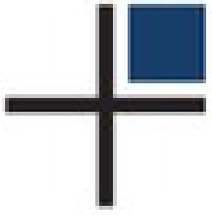
Barman

- Disaster Recovery software
- Open source (GPL)
- Latest version: 1.6.1
  - Released on 23 maggio 2016
- `http://www.pgbarman.org/`



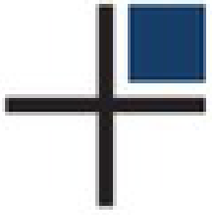
Barman Features

- `.ini` file configuration
- Retention Policies
- Monitoring
- Incremental Backup
- Backup from Standby



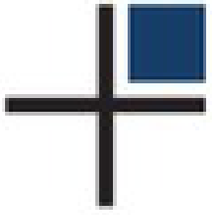
Barman Futures

- Backup methods: `tar`, `pg_basebackup`
- Persistence strategies: `tar`, S3
- Backup compression
- Backup encryption
- Geo-redundancy
- Import/Export
- …

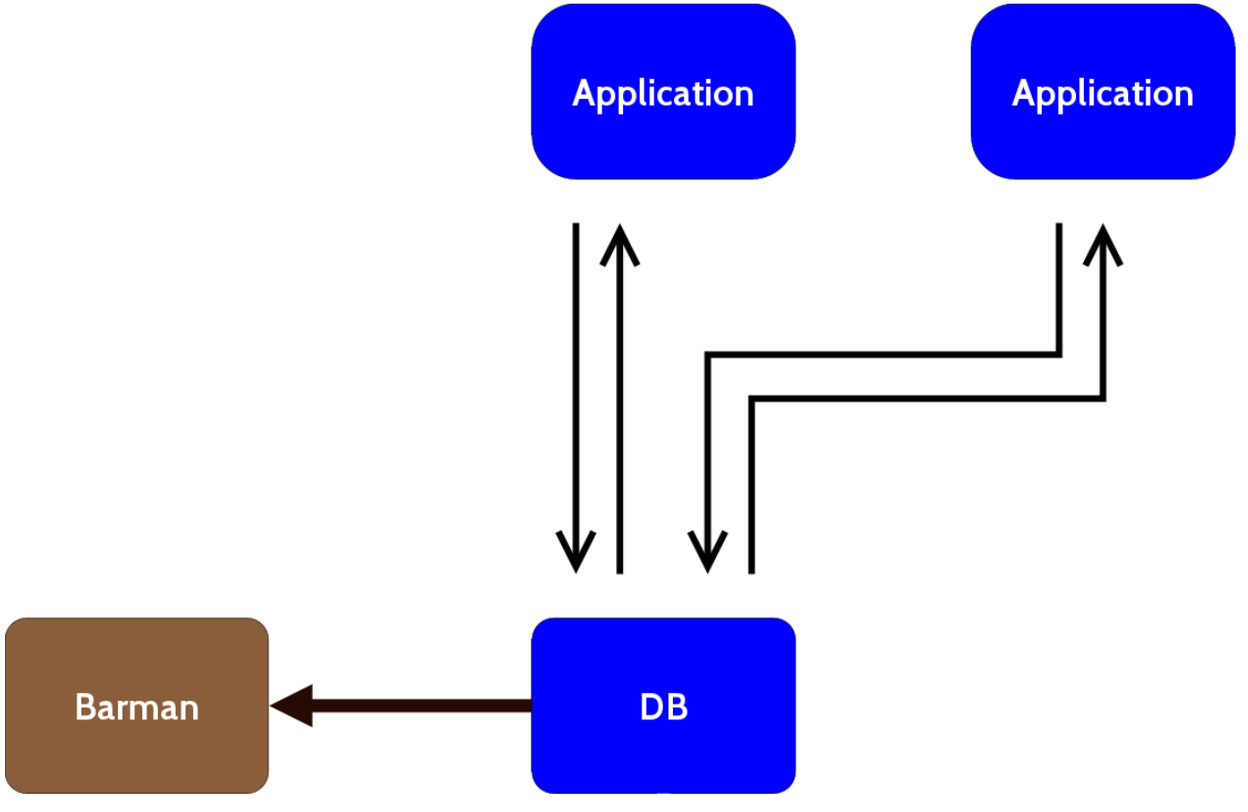


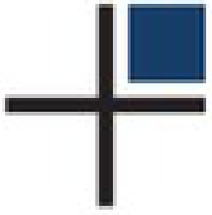
Initial Architecture

- One database server (PostgreSQL)
- One backup server (Barman)



Initial Architecture

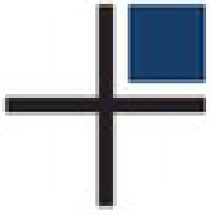




Configuration 1/3

`barman.conf` includes:

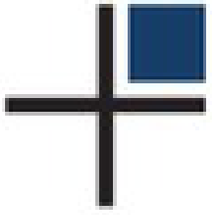
```
[vm95]
ssh_command = ssh vmp
conninfo = service=vmp
description = 9.5 cluster on VMs
```



Configuration 2/3

`~barman/.pg_service.conf` includes:

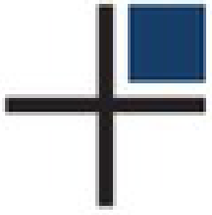
```
[vmp]
host=vm1
user=postgres
dbname=postgres
```



Configuration 3/3

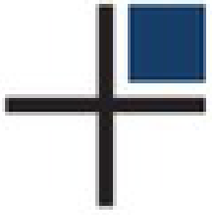
`~barman/.ssh/config` includes:

```
Host vmp
     HostName vm1
     User postgres
     Port 22
```



Configuration

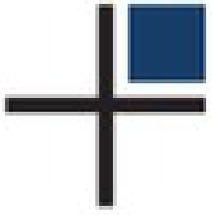
- What depends on state is placed in userspace
  - "system" userspace
- Choice / Best practice?



Introducing repmgr

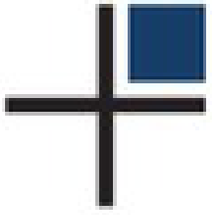
repmgr.conf includes:

```
cluster=vm95
node=1
node_name=vm1
conninfo=host=vm1
```



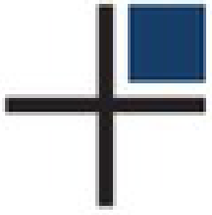
How to use repmgr

```
repmgr primary register

repmgr standby clone (...)
repmgr standby register
repmgr standby unregister
repmgr standby promote
repmgr standby follow
repmgr standby switchover

repmgr witness create

repmgr cluster show
```



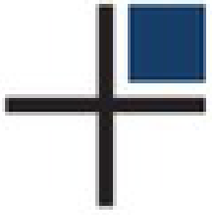
The First Node

```
postgres@vm1:~$ repmgr primary register

postgres@vm1:~$ repmgr cluster show
 Role      | Connection String
 * master  | host=vm1
```

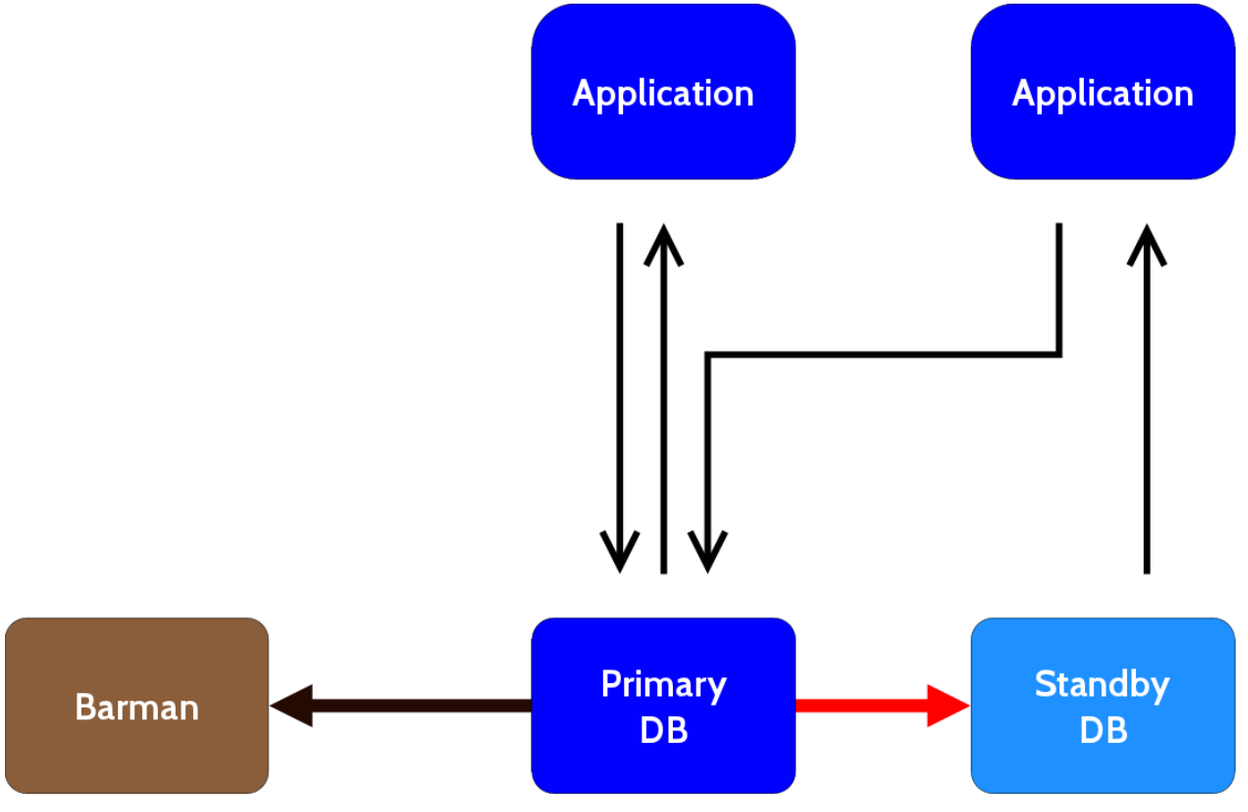


Another Node

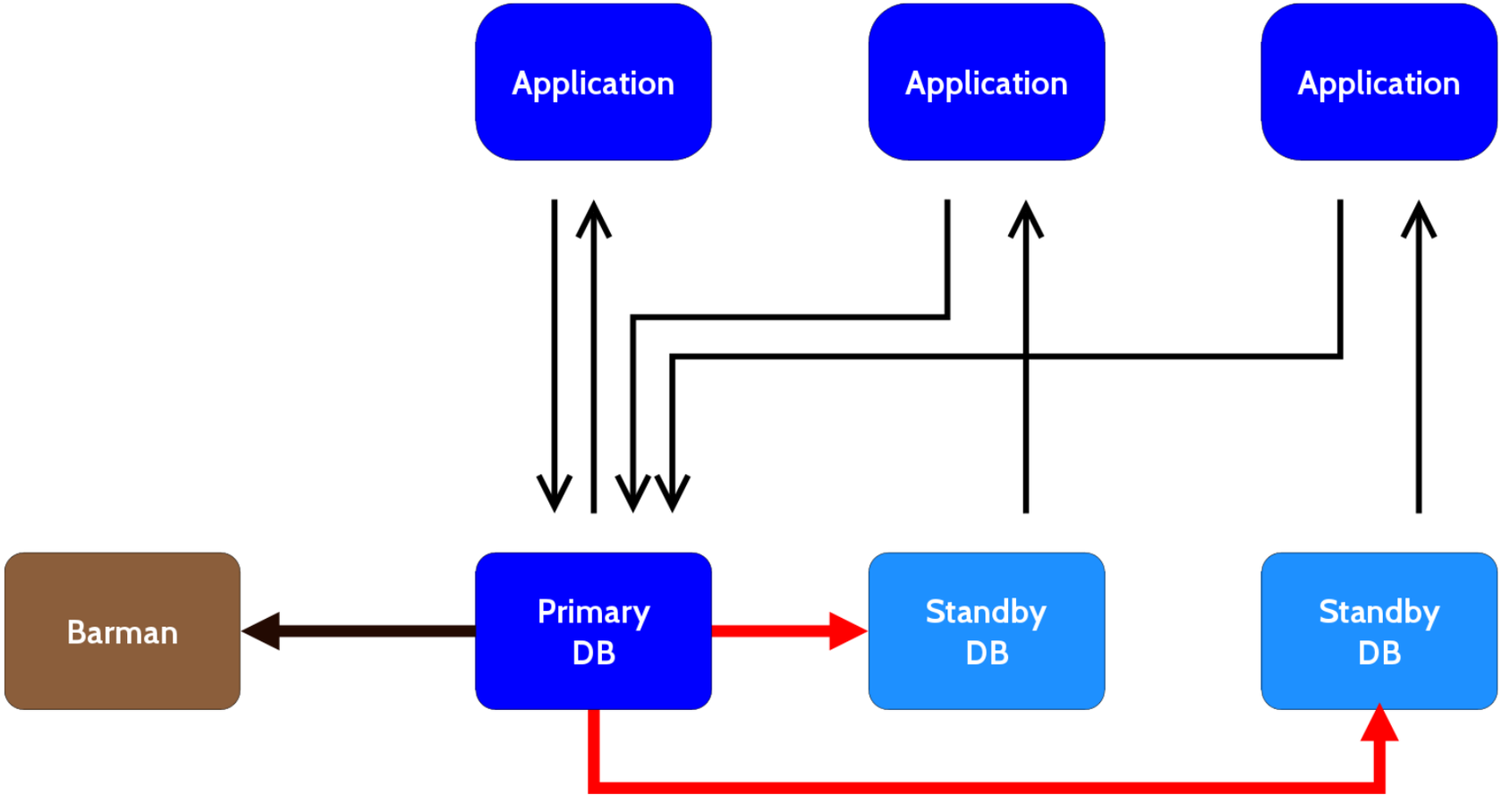
```
postgres@vm2:~$ repmgr standby clone -h vm1

postgres@vm2:~$ repmgr standby register

postgres@vm2:~$ repmgr cluster show
 Role      | Connection String
 * master  | host=vm1
   standby | host=vm2
```

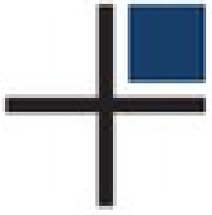


Another Node



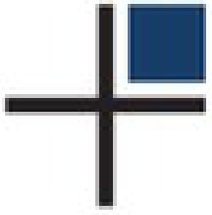
Another Node (again)





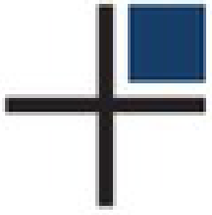
Introducing PgBouncer

- PgBouncer defines its own *databases*
- In PgBouncer:
  - A database is a *connection string*
  - Either local or remote
- Client connects to PgBouncer
  - client connection
- PgBouncer forwards connection to server
  - server connection
- Connections: client >> server



Configuration: db PgBouncer

- Choice: separate reads from writes
  - Best Practice



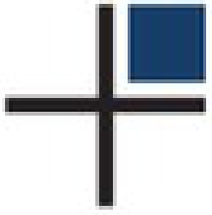
Configuration: db PgBouncer

pgbouncer.ini on vm1:

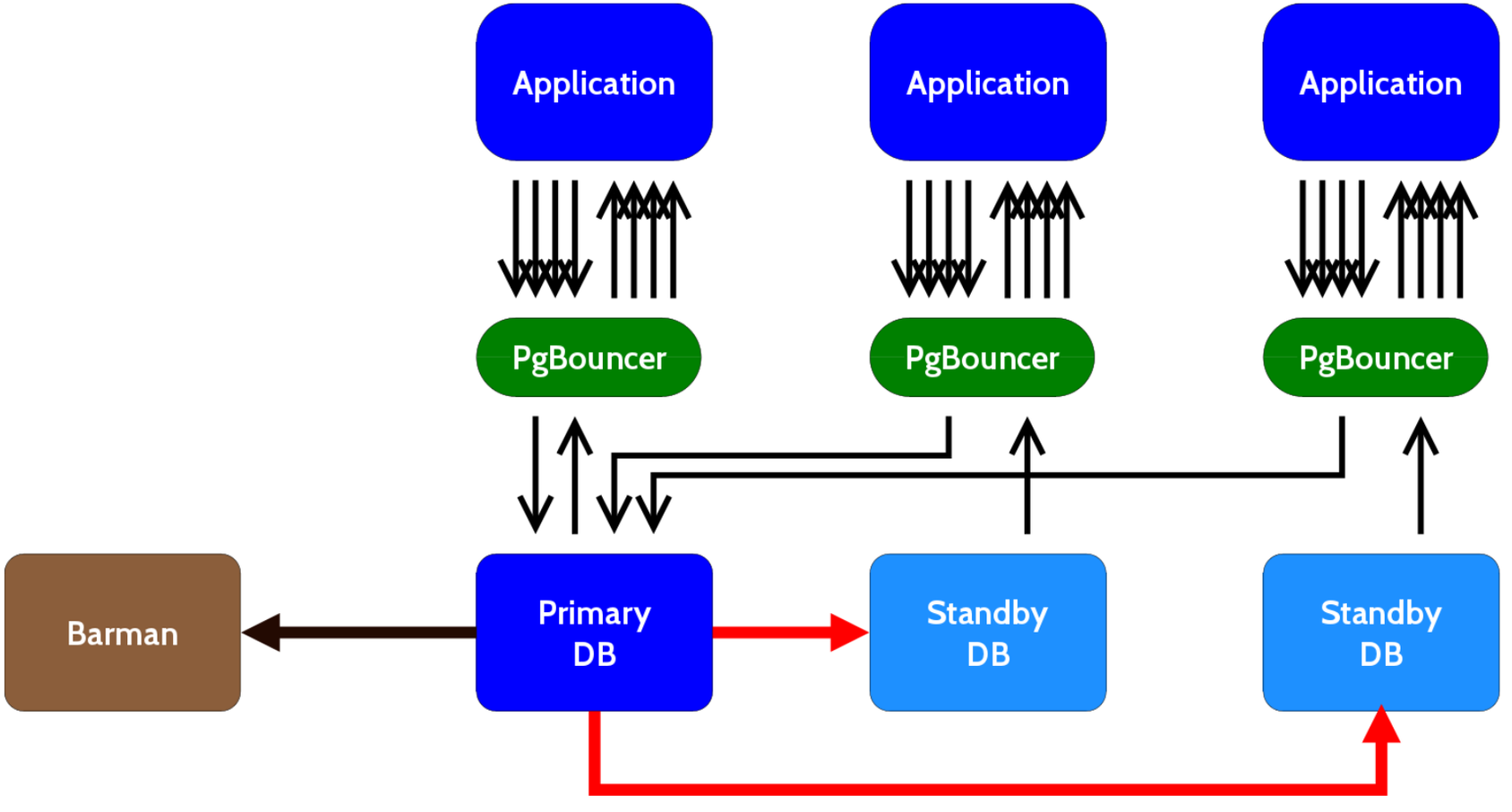
```
[databases]
postgres_rw = host=vm1 dbname=postgres
postgres_ro = host=vm1 dbname=postgres
```

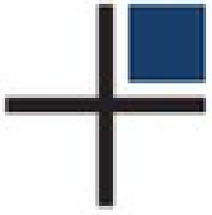
pgbouncer.ini on vm2:

```
[databases]
postgres_rw = host=vm1 dbname=postgres
postgres_ro = host=vm2 dbname=postgres
```



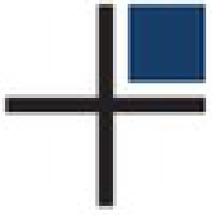
Introducing PgBouncer





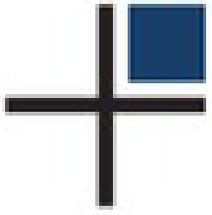
What about Barman?

- Standby nodes are clones of the primary node
- Several copies of one database server
- Barman needs one node
- Barman can take backups from standby. . .
  - (with PostgreSQL 9.6, or using PgEspresso). . .
  - but we'll use the primary node
    - Simpler!
    - Symmetry is useful



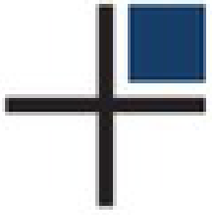
Automating `repmgr`

- Daemon `repmgrd`
  - Automatic Failover
  - Monitoring
- Further automation:
  - When the state changes: reconfigure as appropriate



Cluster state?

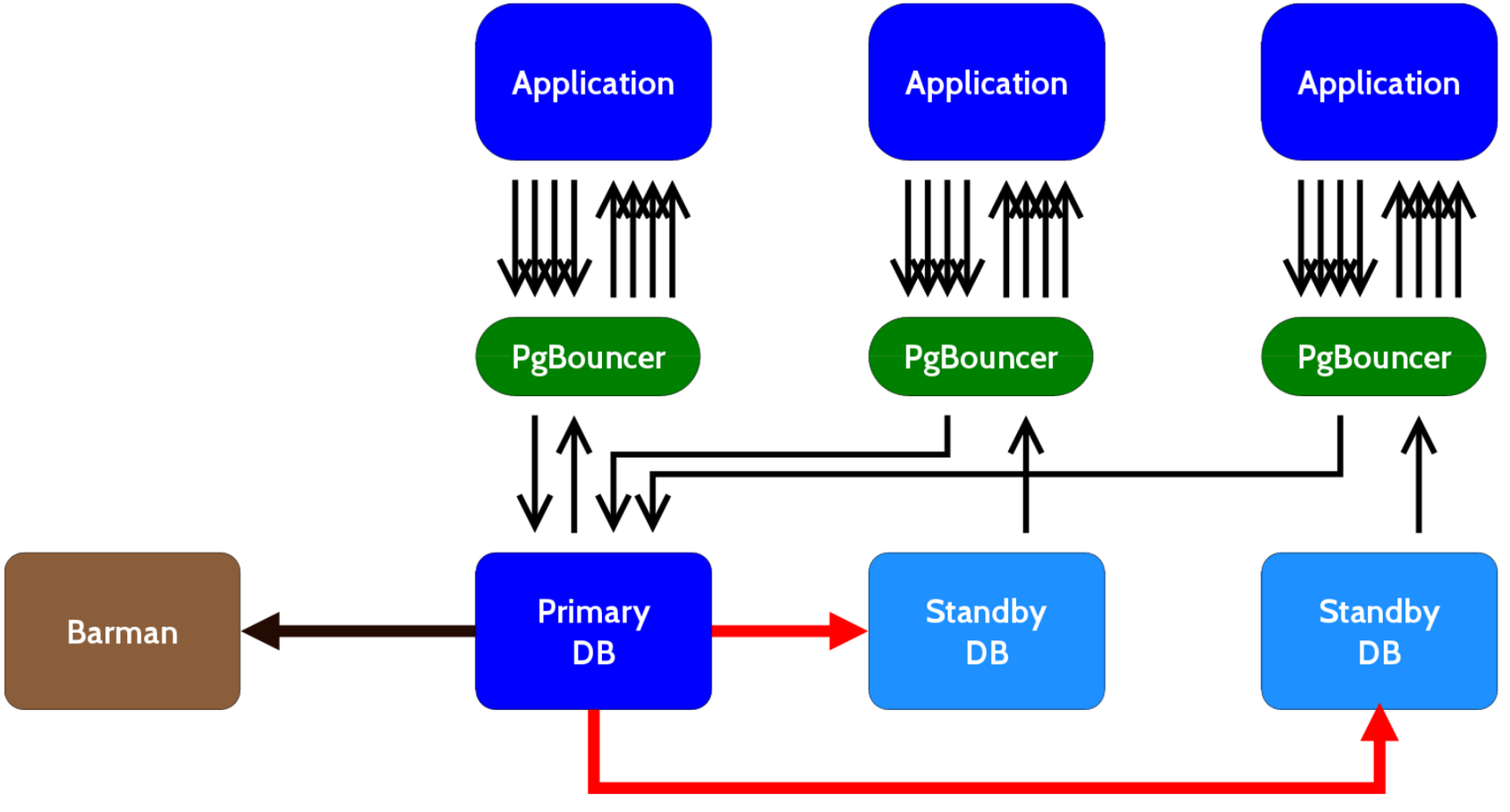
- The *standby* can replace the *primary*
- Two different words:
  - Switchover: planned
  - Failover: unplanned
- Crucial difference!
- The cluster state:
  - List of nodes
  - Which node is primary



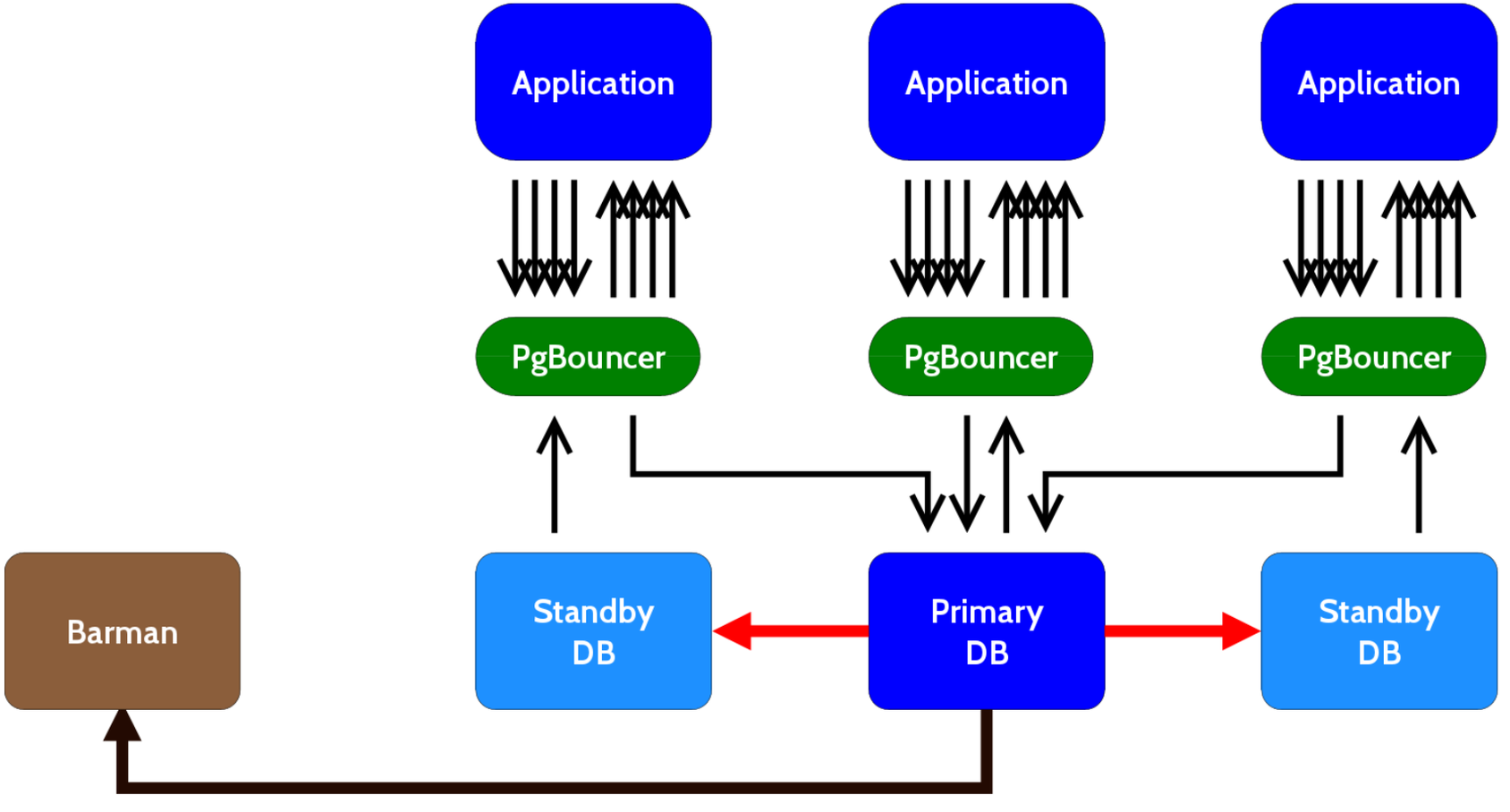
New primary (switchover)

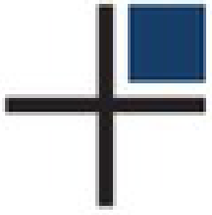
```
postgres@vm1:~$ pg_ctl shutdown

postgres@vm2:~$ repmgr standby promote

postgres@vm3:~$ repmgr standby follow
postgres@vm4:~$ repmgr standby follow
...
postgres@vm100:~$ repmgr standby follow
```

Before switchover



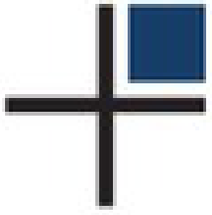
After switchover





New syntax

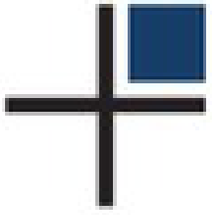
- `repmgr standby switchover`
- Recently introduced
- Complex process
- Performs actions on other nodes
  - Requires SSH access
- Old primary recycled by `pg_rewind`
- Some restrictions
  - To be lifted in the future



Automatic Failover

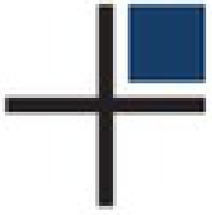
Just add:

```
failover=automatic
master_response_timeout=20
reconnect_attempts=3
reconnect_interval=5
promote_command=repmgr standby promote
follow_command=repmgr standby follow -W
```



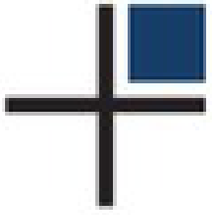
Pick the new primary

- Use node priority
  - Integer
- Criteria
  - The most advanced node
  - The one with higher priority
- Only nodes with positive priority
  - Not all standbys are good masters



State change

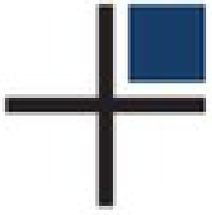
- When state changes
  - We update configuration
- All in (system) *userspace*:
  - `~barman/.ssh/config`
  - `~barman/.pg_service.conf`
- Almost...
- Not in userspace:
  - `/etc/pgbouncer/pgbouncer.ini`



Event Notification Command

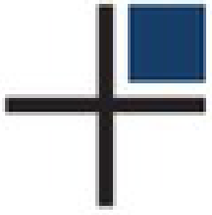
Add to `repmgr.conf` (two lines only):

```
event_notification_command =
  repmgr-agent.sh repmgr.conf
  barman-server %n %e %s

event_notifications =
  master_register, standby_register,
  standby_promote, standby_switchover
```



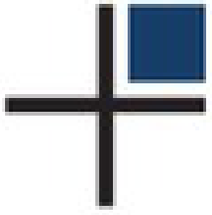
Event Notification Command

- Run a script on cluster events
  - Like AFTER triggers
- Only events that *change the state*



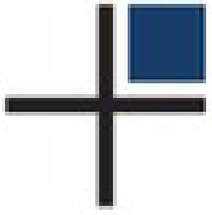
repmgr-agent.sh

- Script that updates configuration
- Idempotent
- Prototype, contributed to `repmgr`
- Reads cluster state
  - From any node
- Rewrites:
  - `~barman/.ssh/config`
  - `~barman/.pg_service.conf`
  - `/etc/pgbouncer/pgbouncer.ini`



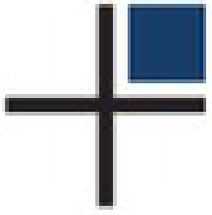
repmgr Future

- Integration with Barman
- Integration with PgBouncer
- More robust failover
  - Share information across nodes



And now. . .

Questions?



And then...

Thank you!

gianni@2ndquadrant.com

@GianniCiolli

Licence