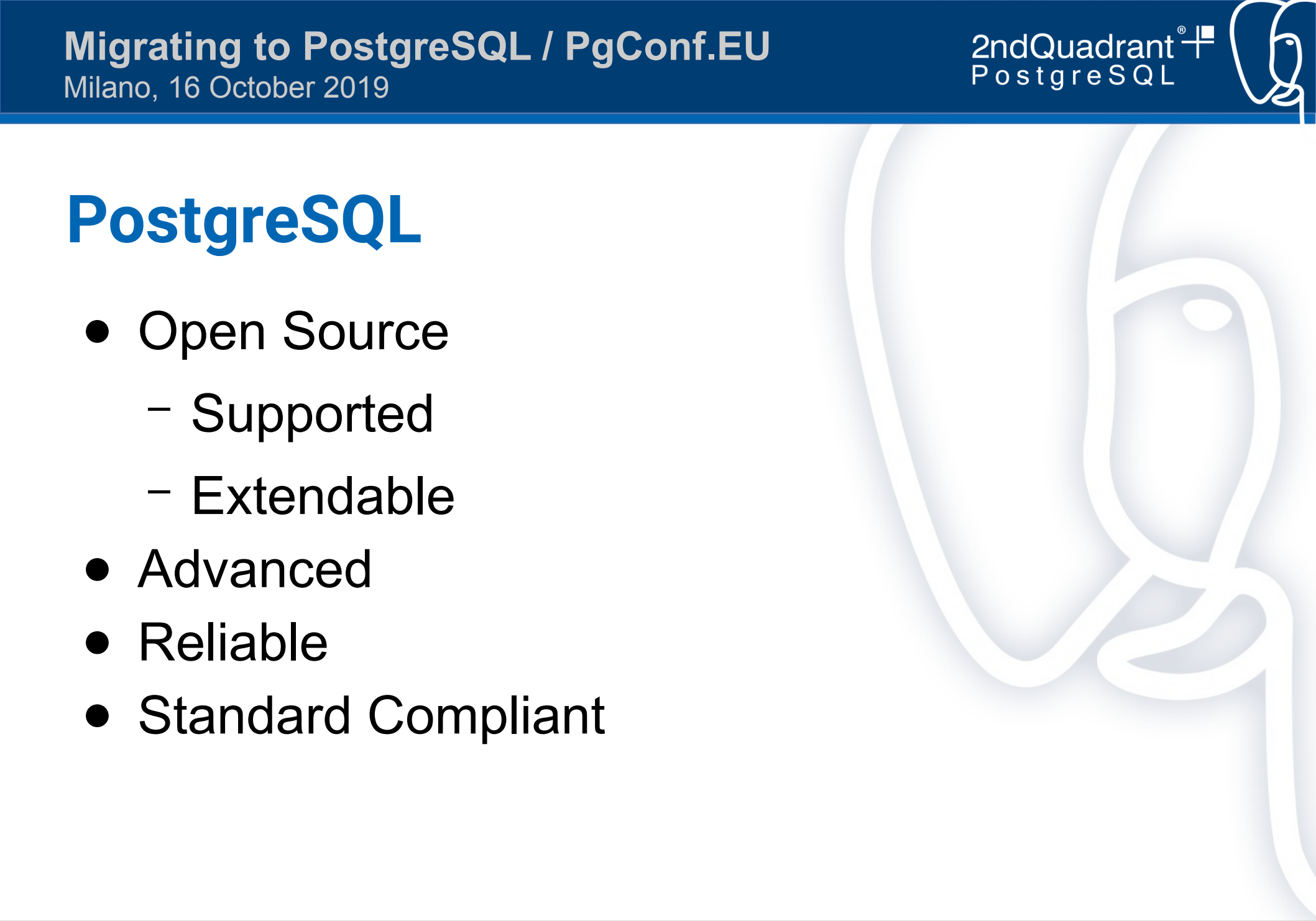
2ndQuadrant®
PostgreSQL

Migrating to PostgreSQL

Boriss Mejías
Consultant - 2ndQuadrant
Air Guitar Player

Why Migrate to PostgreSQL?



PostgreSQL

- Open Source
  - Supported
  - Extendable
- Advanced
- Reliable
- Standard Compliant

PostgreSQL

- It's an **All-Rounder**
  - Low Latency
  - Big Data
  - High Availability
  - "Document" Database
- Sometimes better than dedicated solutions
  - Scale to petabytes (from Elasticsearch)

PostgreSQL

- **Awesome Community**

Why Migrate to Open Source?

Why Migrate to Open Source?

Reason #1 is “Cost”

Why Migrate to Open Source?

Reason #1 is “Cost”
(or it used to be)

Top Reasons to Stay in Open Source

1. Competitive features, innovation
2. Freedom from vendor lock-in
3. Quality of solutions
4. Ability to customize and fix
5. Cost

https://www.slideshare.net/blackducksoftware/2016-future-of-open-source-survey-results

Migration Timeline

- Effort Assessment
- Decision (is it worth?)
- Preparation
- Testing
- Migration
- Cleanup

Effort Assessment

- Schema
- Data
- Code
  - What language? (SQL / Other)
  - Where? (Client / Server)
- Architecture

Schema

- Usually the easiest part
  - Available via common tools
- Map data types as appropriate
  - Look for simplifications
- Consider custom datatypes
  - Simpler is better than complex
  - Complex is better than complicated

Zen of Python

Data Type

- PostgreSQL has several data types
- Classical: text, numbers, boolean, time/date
- Modern: Arrays, JSON
- User-defined:
  - Composite
  - Enumerative
  - Your data type in C

Data Type Gotcha

- Oracle NUMBER to NUMERIC
- MySQL BOOLEAN to BOOLEAN
- Oracle NULL

```
SELECT first_name
  || second_name
  || last_name;
```

Data Type Gotcha

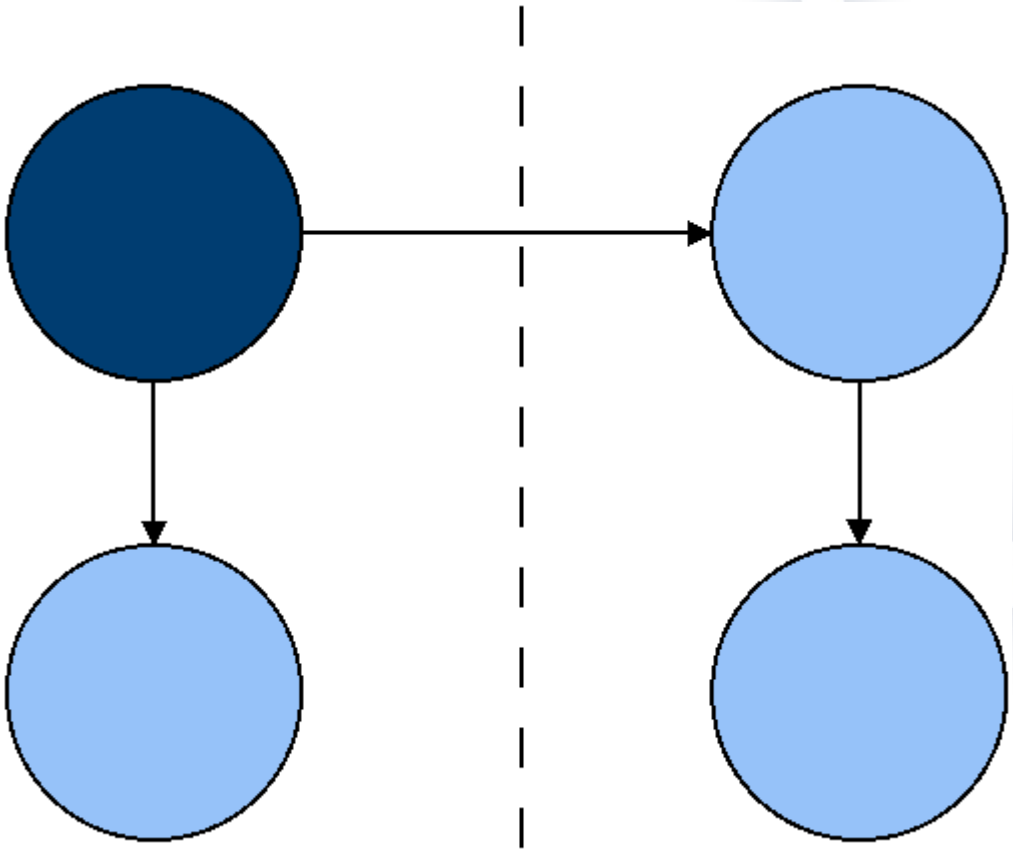
- Oracle NUMBER to NUMERIC
- MySQL BOOLEAN to BOOLEAN
- Oracle NULL

```
SELECT first_name
  || COALESCE(second_name, '')
  || last_name;
```

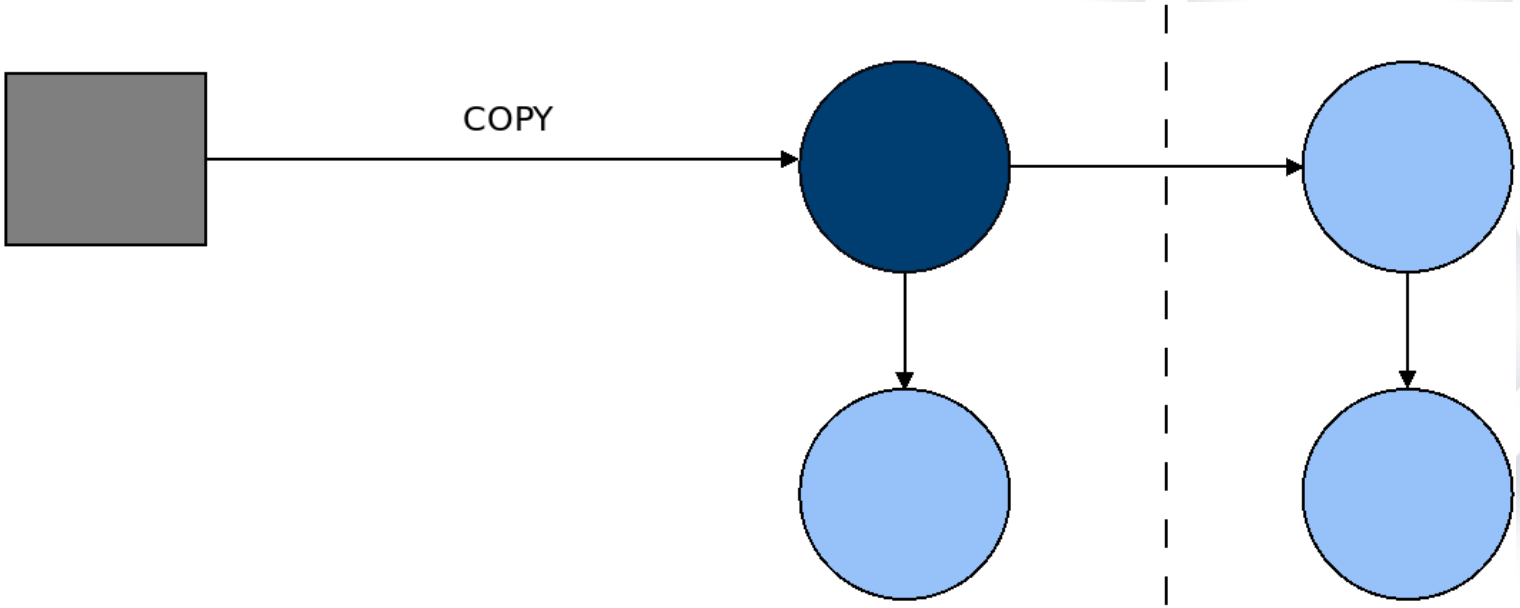
Architecture Assessment

- High Availability
- Disaster Recovery
- Multi-Master
- Selective Replication

Target Architecture



Target Architecture



Solutions Mapping

- Rich PostgreSQL ecosystem
  - Core
  - Contrib / Extensions
  - Third Party (both FLOSS and proprietary)
- Sometimes difficult to find exact match
  - Might not be needed
  - You must match the **purpose**, not the tool

Application Code

- Many programming languages and frameworks have PostgreSQL drivers
  - Not an issue (usually)
- Real issue: SQL variants with different feature sets:
  - Emulate missing features
  - Remove useless emulations

Application Code Gotcha

- SELECT 1 FROM DUAL;
- Upper case default in Oracle
  - CREATE TABLE DUAL ();
  - DUAL → dual → “DUAL”
- Exceptions in store procedures

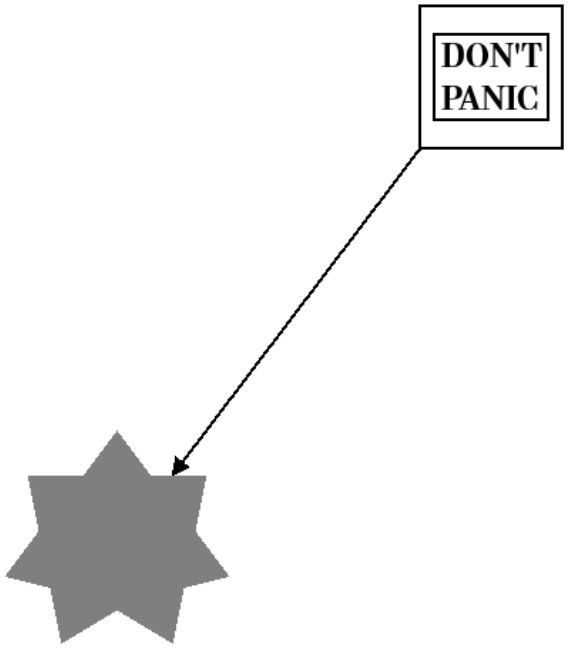
Planning the Migration

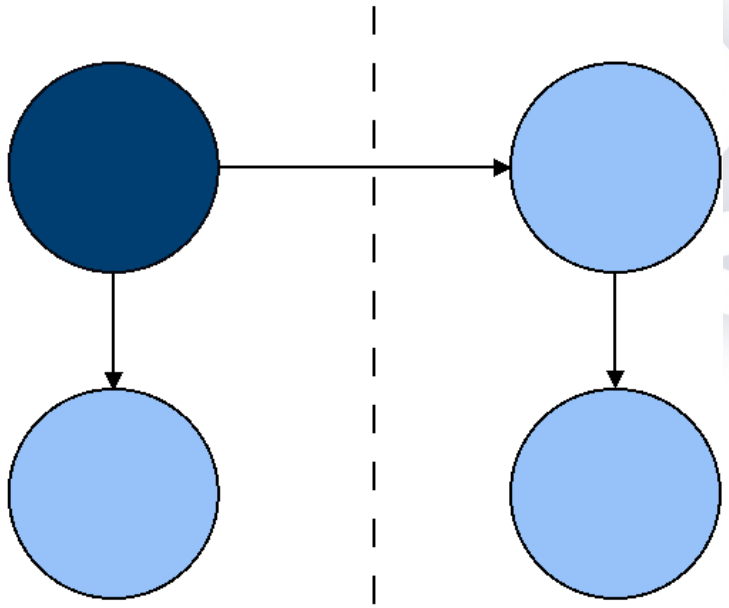
- The Assessment includes (at least) one Plan
  - Time
  - Cost
  - Contingency / Rollback

Take Advantage of the Application

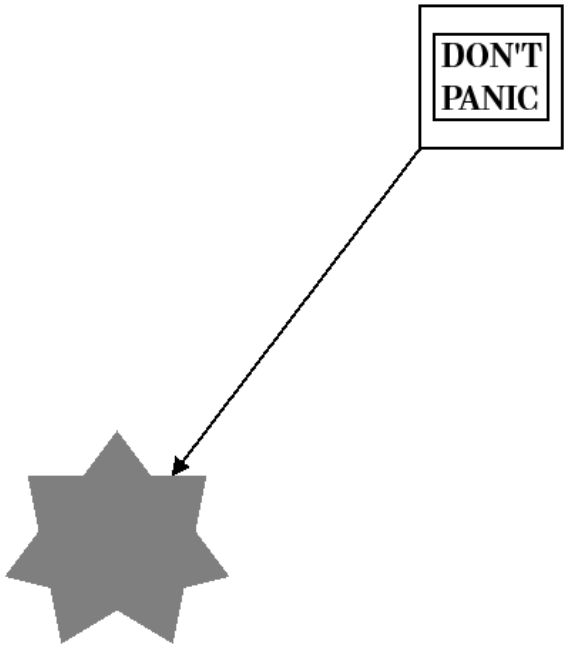
- Some applications support multiple databases
- They have done all the major part of the work
- Functions, procedures, data types. It all works already with PostgreSQL
- Just need to migrate the data

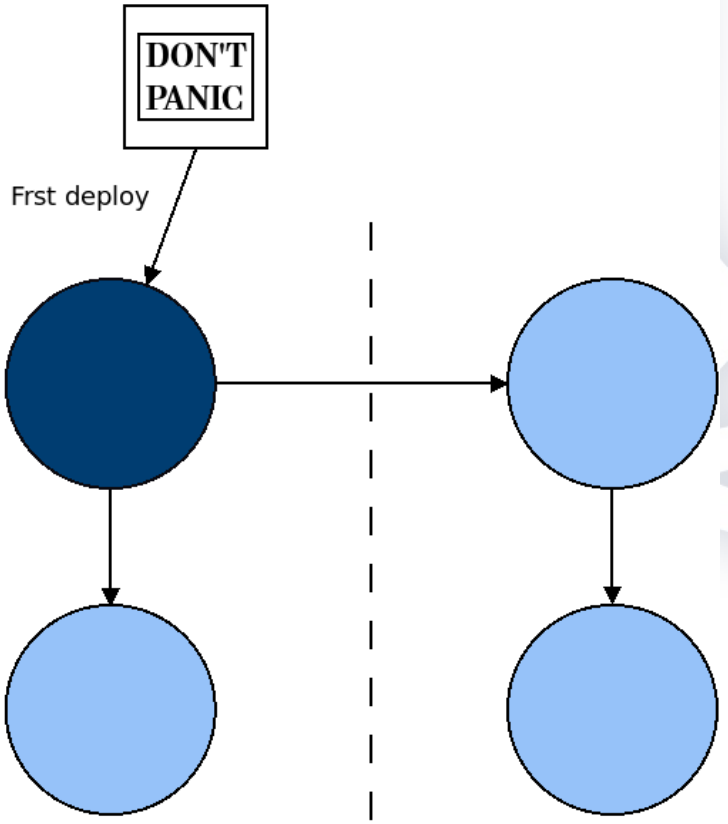
Don’t Panic



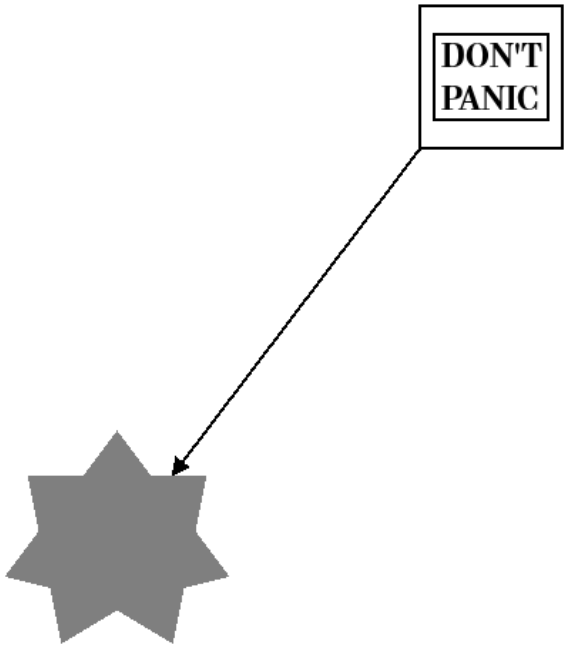


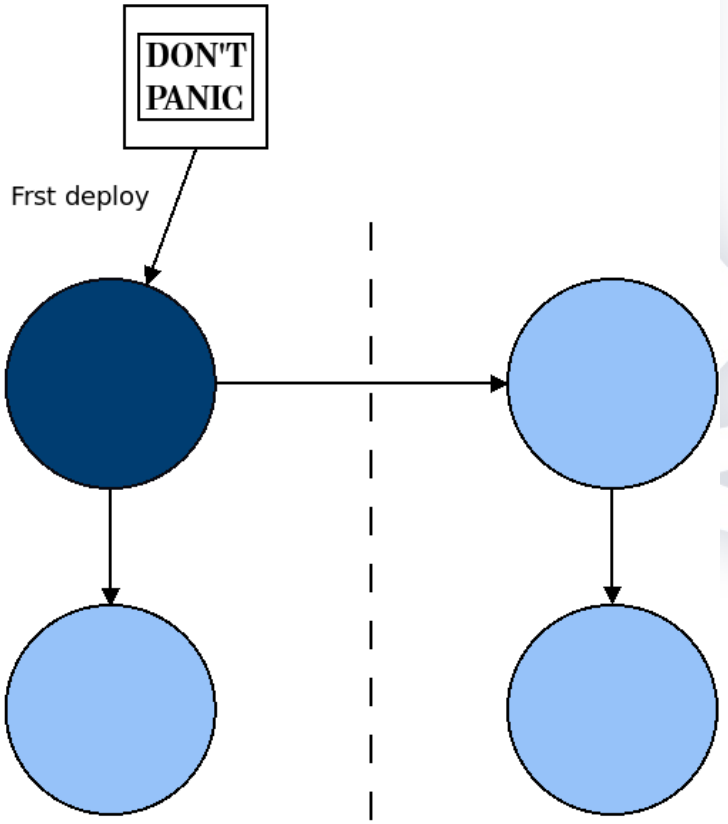
Vanilla Deployment



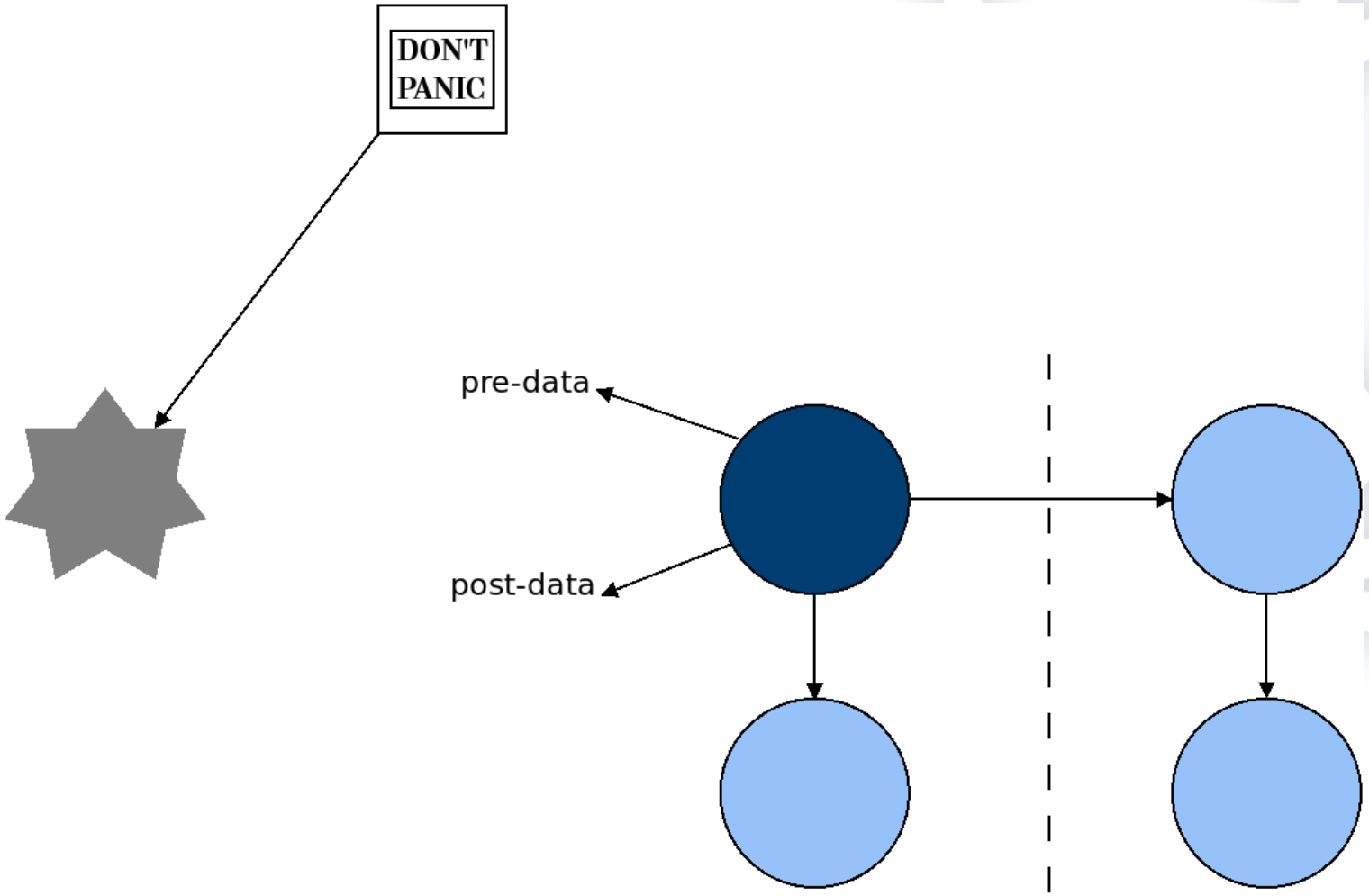


Bilberry Deployment

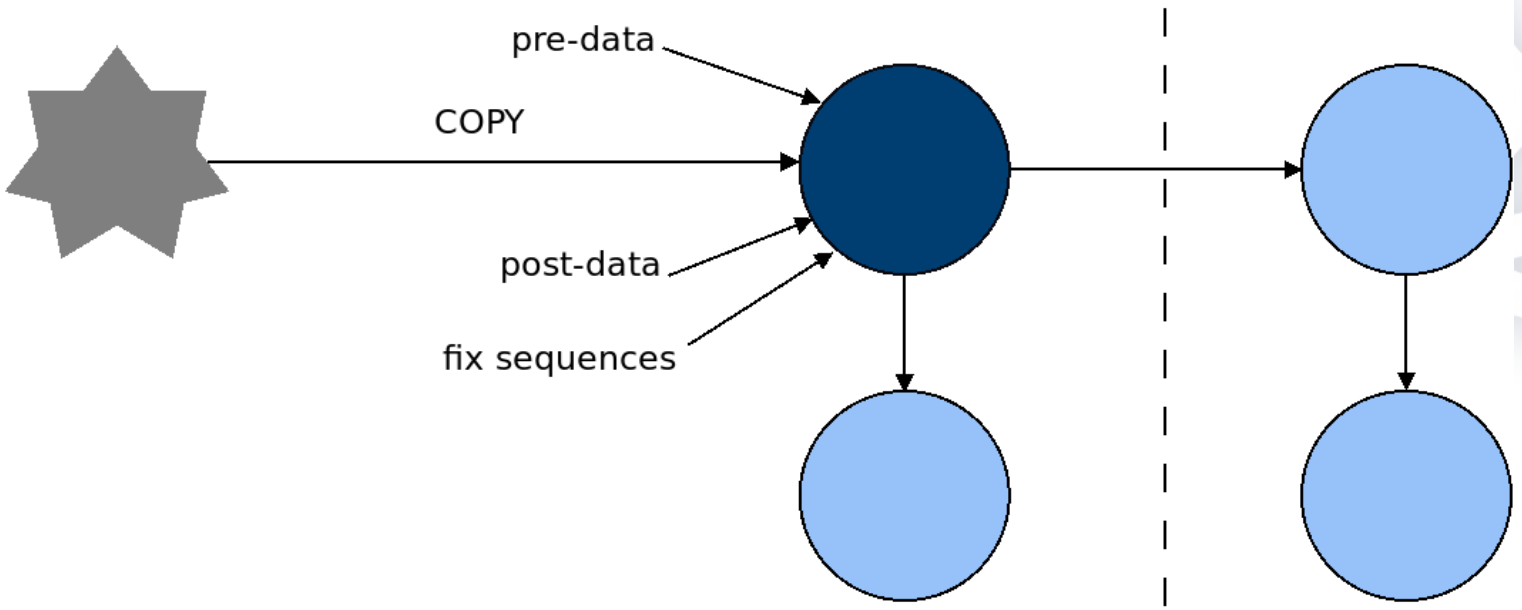




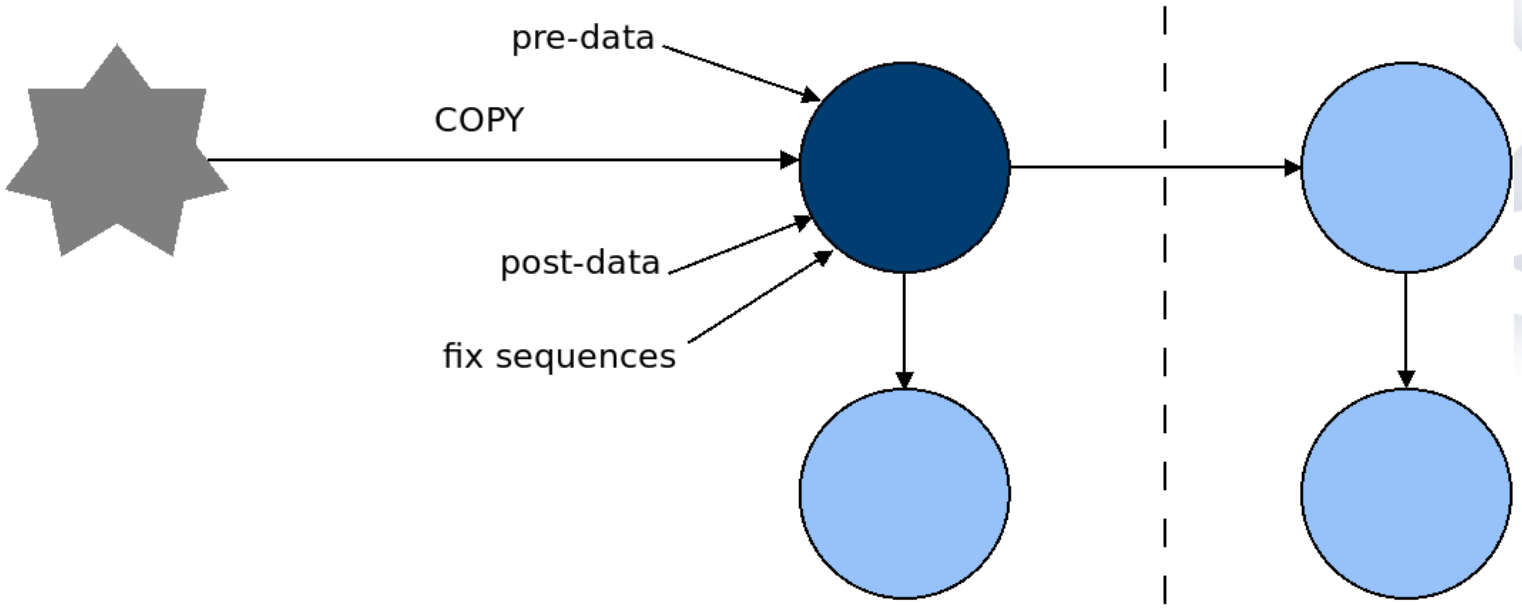
Get the Schema Definition



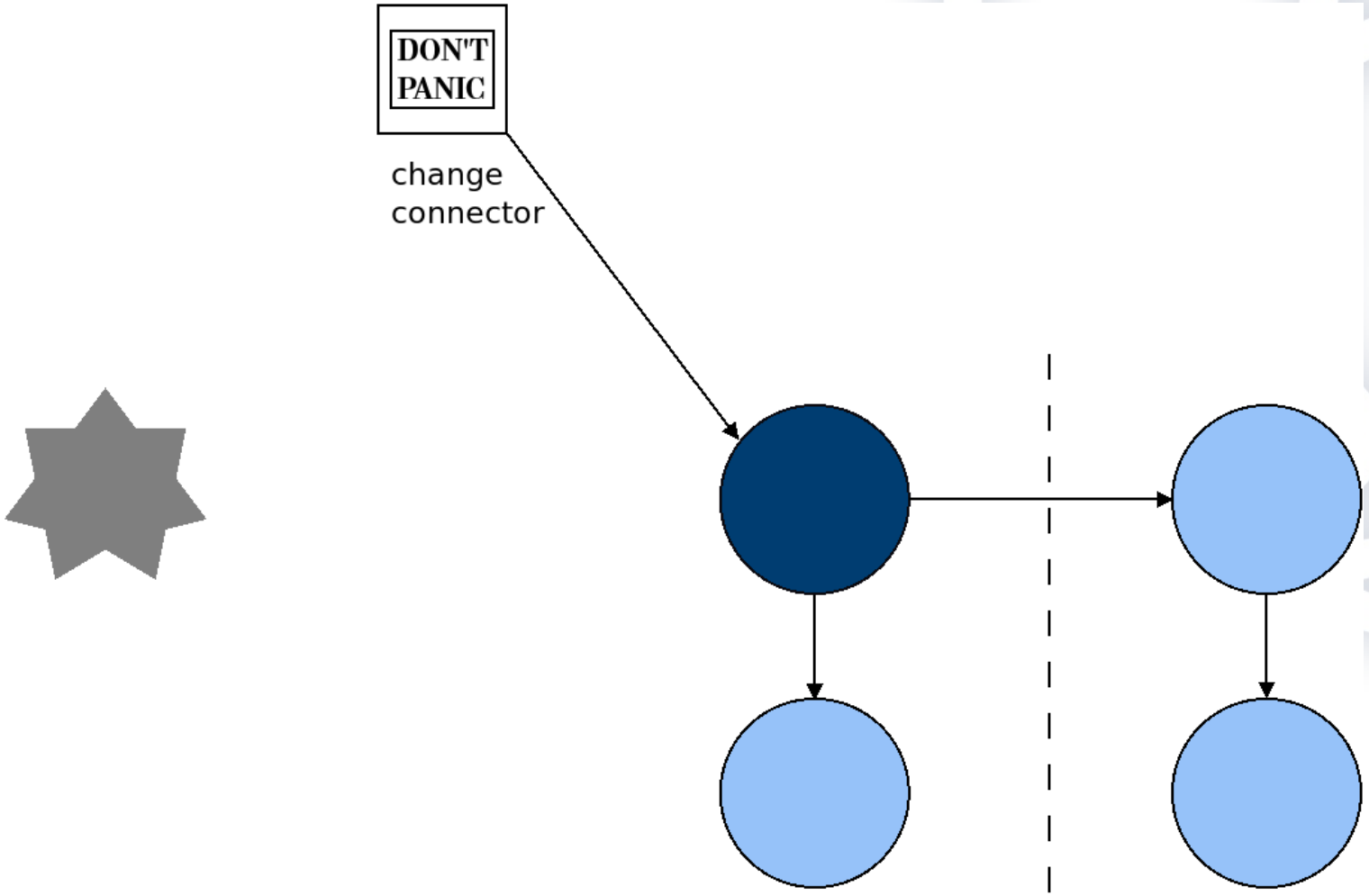
Migrate Data



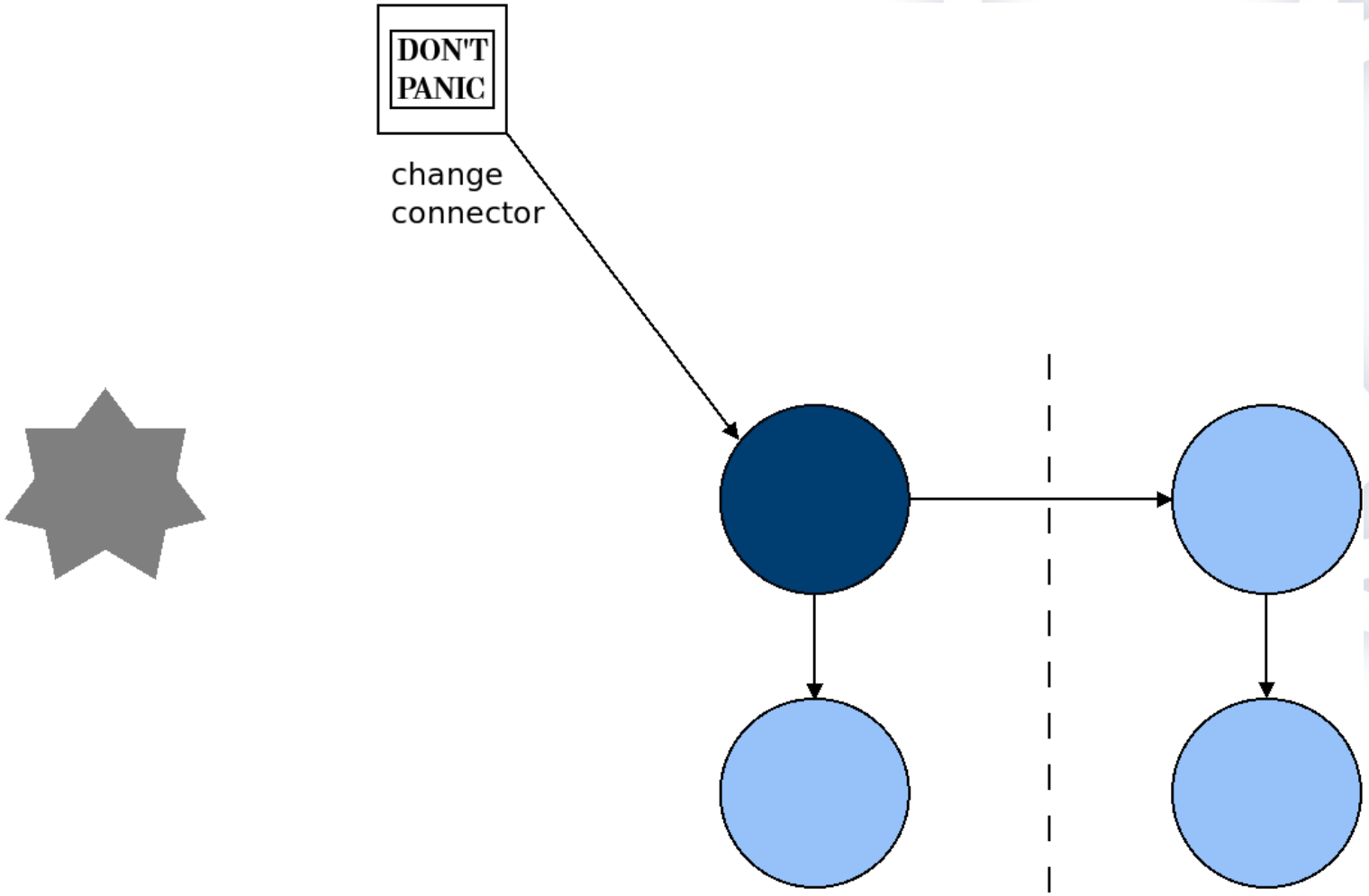
Migrate Data → Downtime



Redirect the Application



Redirect the Application → Showtime!



Alternative Strategy: Phasing Out

- Use PostgreSQL for new services
  - Keep old services as they are
- Useful when standard plans are too complicated / expensive
- No need to migrate old data and code
  - Easier: “just” plan new system

Phasing Out and Integration

- Integrate new PostgreSQL with existing DBs
- Preserve continuity of services
- Foreign Data Wrappers
  - Pluggable adaptors for other systems
  - SQL/MED standard
  - Some of them are read/write

Alternative Strategy: Preparation

- Modify the existing system before migrating
- Make it nearer to PostgreSQL
  - Stop using incompatible features
  - Rewrite/simplify queries
- Enables application compatibility
- Makes migration easier / cheaper / faster

Testing

- Compatibility
- Performance

- The migration process **includes writing tests**

Performance Testing

- Test must include difficult / critical queries
- Ensure that newer optimisations don't cause regressions on other queries
- Use pgbench (custom scripts)
- Analyse the current workload
- Reproduce it
- Properly dimension hardware

Scripted Migration

- The migration procedure should be scripted as much as possible
- A script can be:
  - Repeated
  - Versioned
  - Benchmarked
  - Tested

Scripted Migration

- The migration procedure should be scripted as much as possible
- A script can be:
  - Repeated
  - Versioned
  - Benchmarked
  - Tested (in staging environment)

Scripted Migration

- The migration procedure should be scripted as much as possible
- A script can be:
  - Repeated
  - Versioned
  - Benchmarked
  - Tested (in staging environment, please!)

Thoughts

- Focus on the purpose not on emulating
- Make a plan
- Test, test, test
- Learn PostgreSQL

Thoughts

- Focus on the purpose not on emulating
- Make a plan
- Test, test, test, test, test, test, test
- Learn PostgreSQL

Thoughts

- Focus on the purpose not on emulating
- Make a plan
- Test, test, test, test, test, test, test
- Learn PostgreSQL and get help

Thoughts

- Focus on the purpose not on emulating
- Make a plan
- Test, test, test, test, test, test, test
- Learn PostgreSQL and get help
- Don’t Panic

2ndQuadrant PostgreSQL

Thanks and Remember
Benjamin Zander's Rule #6

Boriss Mejias
boriss.mejias@2ndquadrant.com
@tchorix