PostgreSQL in großen Installationen

Cybertec Schönig & Schönig GmbH
Hans-Jürgen Schönig

Wieso PostgreSQL?

- Die fortschrittlichste Open Source Database

- Lizenzpolitik: wirkliche Freiheit

- Stabilität, Zuverlässigkeit, Skalierbarkeit

Oder wollen sie ...

Meldung vom 29.11.2008 20:30

MySQL-Gründer warnt vor aktueller Version

vorlesen / MP3-Download

Michael "Monty" Widenius kritisiert in einem Blog-Eintrag das gegenwärtige Entwicklungsmodell der freien Datenbank MySQL und rät davon ab, die gerade veröffentlichte Version 5.1 produktiv einzusetzen. In einer Liste führt er zahlreiche Fehler auf, die seiner Meinung nach vor der Freigabe hätten korrigiert werden müssen. Darunter ist ein fünf Jahre alter Bug, der es bis in den Wikipedia-Eintrag über MySQL geschafft hat. Angreifer können ihn dazu benutzen, sämtliche Slaves in einer replizierten MySQL-Installation lahmzulegen. Insgesamt gebe es noch 120 schwere Fehler; außerdem seien viele neue Funktionen fehlerhaft. So verlangsame das gerade eingeführte Logging in Tabellen den Server um 30 Prozent und mehr, und partitionierte Tabellen seien fehleranfällig bis hin zum totalen Datenverlust.

Insgesamt gebe es 20 kritische Fehler in der aktuellen MySQL-Version, die zum Absturz des Servers oder zu falschen Ergebnissen führen können. Dazu kämen weitere 35 Bugs in Version 5.0, die "vermutlich" in der aktuellen noch nicht beseitigt seien. Als Gründe für die vorzeitige Freigabe der 5.1er-Version nennt Widenius unter anderem eine zu starke Zersplitterung des Entwicklerteams. Außerdem sei ein zu früher Entwicklungsstand zum Release-Kandidaten erklärt worden. Und schließlich lasse sich nur als fertig erklärte Software verkaufen, keine noch in der Entwicklung befindliche.

CYBERTEC
The PostgreSQL Database Company

Oder wollen sie ...

... dass Ihre Spenderniere auch
einer von 200 Fehlern ist?

CYBERTEC
The PostgreSQL Database Company

Anders formuliert ...

PostgreSQL im Einsatz

- für große und kleine Setups

- viele professionelle Features

- “Stabilität” als Feature

Komplexes SQL

```
-- a way of doing "select 1"
WITH x AS ( SELECT row_number()
    OVER (PARTITION BY (SELECT *
        FROM ( SELECT 1) AS e) ORDER BY 1)
            FROM ( SELECT 1 d) AS y)
    SELECT ( SELECT * FROM ( SELECT * FROM x) AS x) AS output
        FROM ( SELECT 1 AS z) AS x
        WHERE z >= ( SELECT 1)
            OR z IN ( (SELECT 1), (SELECT 1))
        ORDER BY ( SELECT 1)
        LIMIT ( SELECT 1 FROM ( SELECT * FROM x) AS a);
```

Wieso ist das wichtig?

- Datenbanken sind mehr als nur Tabellen

- **Performance:** viel Operationen sind in der DB einfach schneller

- **Entwicklungszeit:** SQL ist kürzer, leichter zu entwickeln und in Summe billiger

- **Skalierbarkeit:** 500mio Zeilen in der App zu sortieren ist “uncool”

Wieso ist das wichtig?

- Datenbanken sind mehr als nur Tabellen

- **Performance:** viel Operationen sind in der DB einfach schneller

- **Entwicklungszeit:** SQL ist kürzer, leichter zu entwickeln und in Summe billiger

- **Skalierbarkeit:** 500mio Zeilen in der App zu sortieren ist “uncool”

Transactions / Locking (1)

- Hoch entwickelte Transactions (ACID)

- Business Applikationen kommen nicht ohne Transactions aus

- Feines Locking ermöglicht Parallelität

- Gute Eignung für “größe Eisen”

Transactions / Locking (2)

- Ein Beispiel:

```
SELECT ...
    FROM    a, b, c, d, e
    WHERE ...
    FOR UPDATE OF a, b
    FOR SHARE OF c
```

=> Verbesserte SMP Fähigkeit

CYBERTEC
The PostgreSQL Database Company

Replication (1)

- Viele Funktionalitäten:

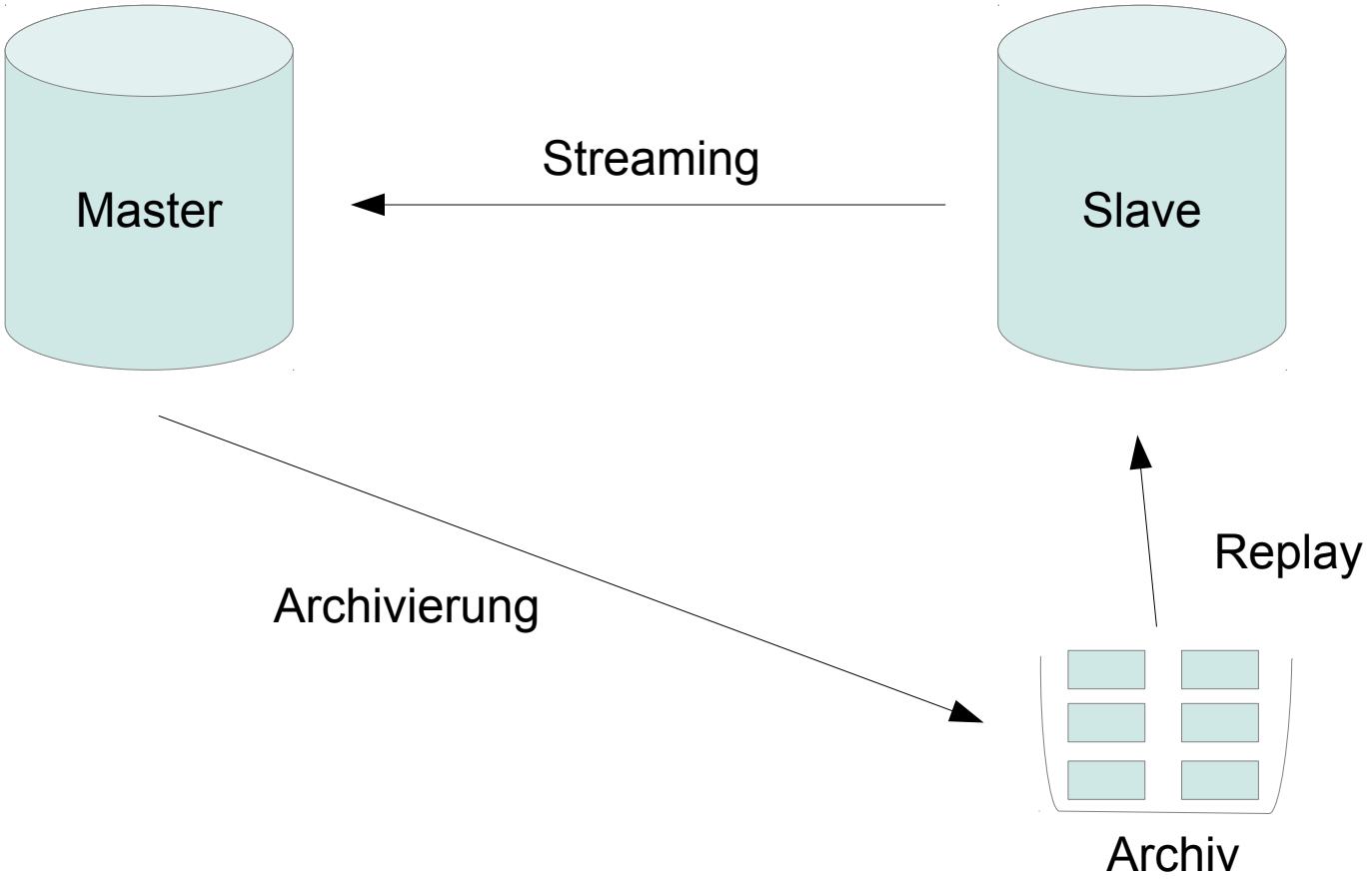
  => Asynchrone Replikation

  => Synchrone Replikation

  => “Queue basierte” Lösungen

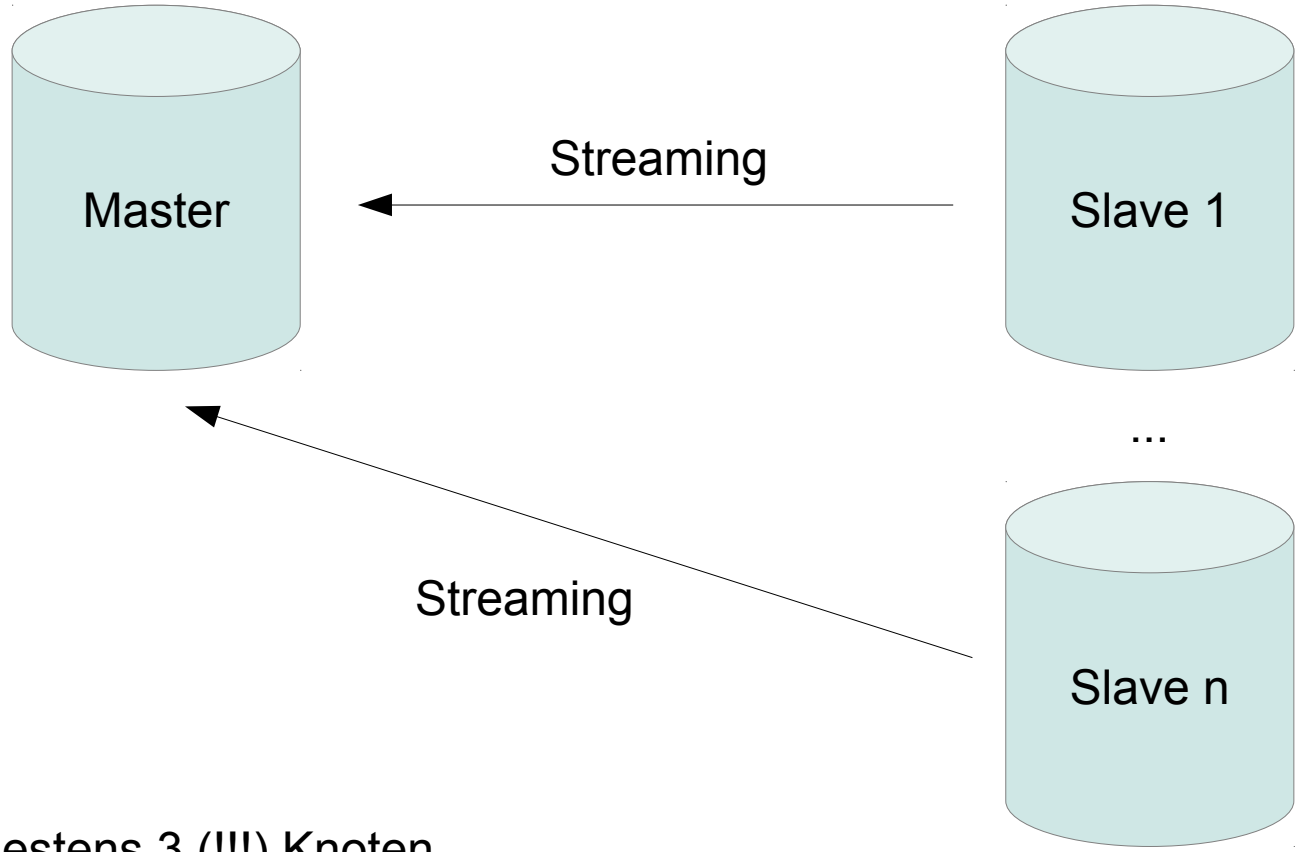
Replication (2)

- Asynchrone Replikation / PITR:



Replication (3)

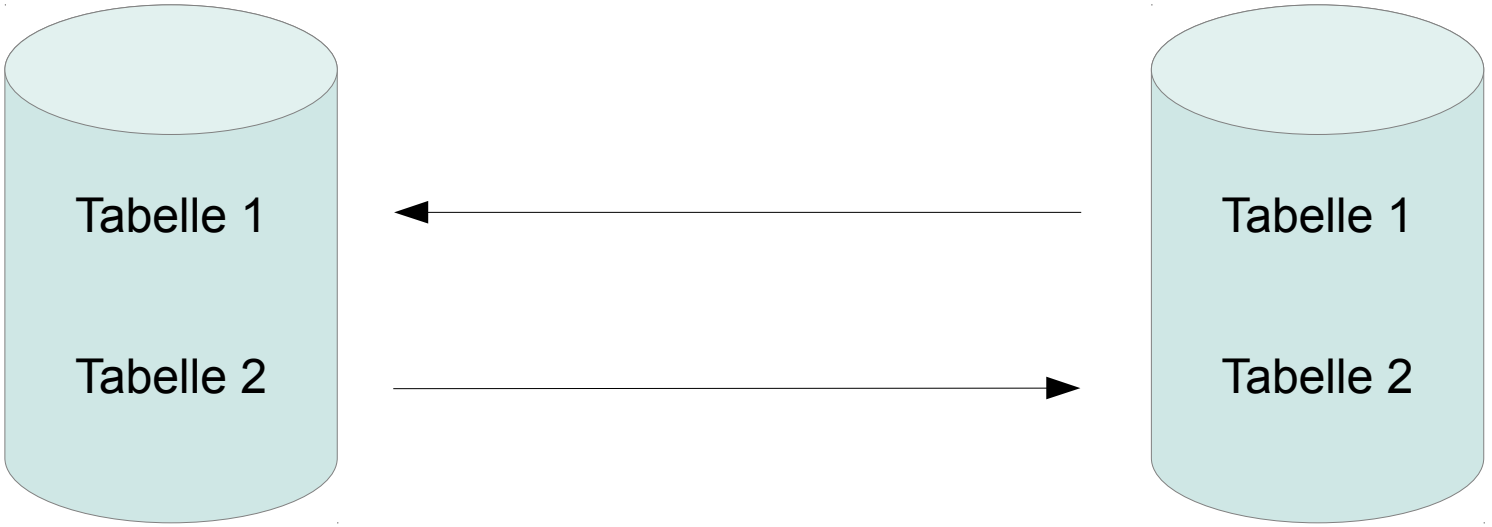
- Synchrone Replikation



Hinweis: Mindestens 3 (!!!) Knoten

Replication (4)

- Queue basierte Lösungen



- Für jede Tabelle gibt es jeweils einen Master und “n” Slaves

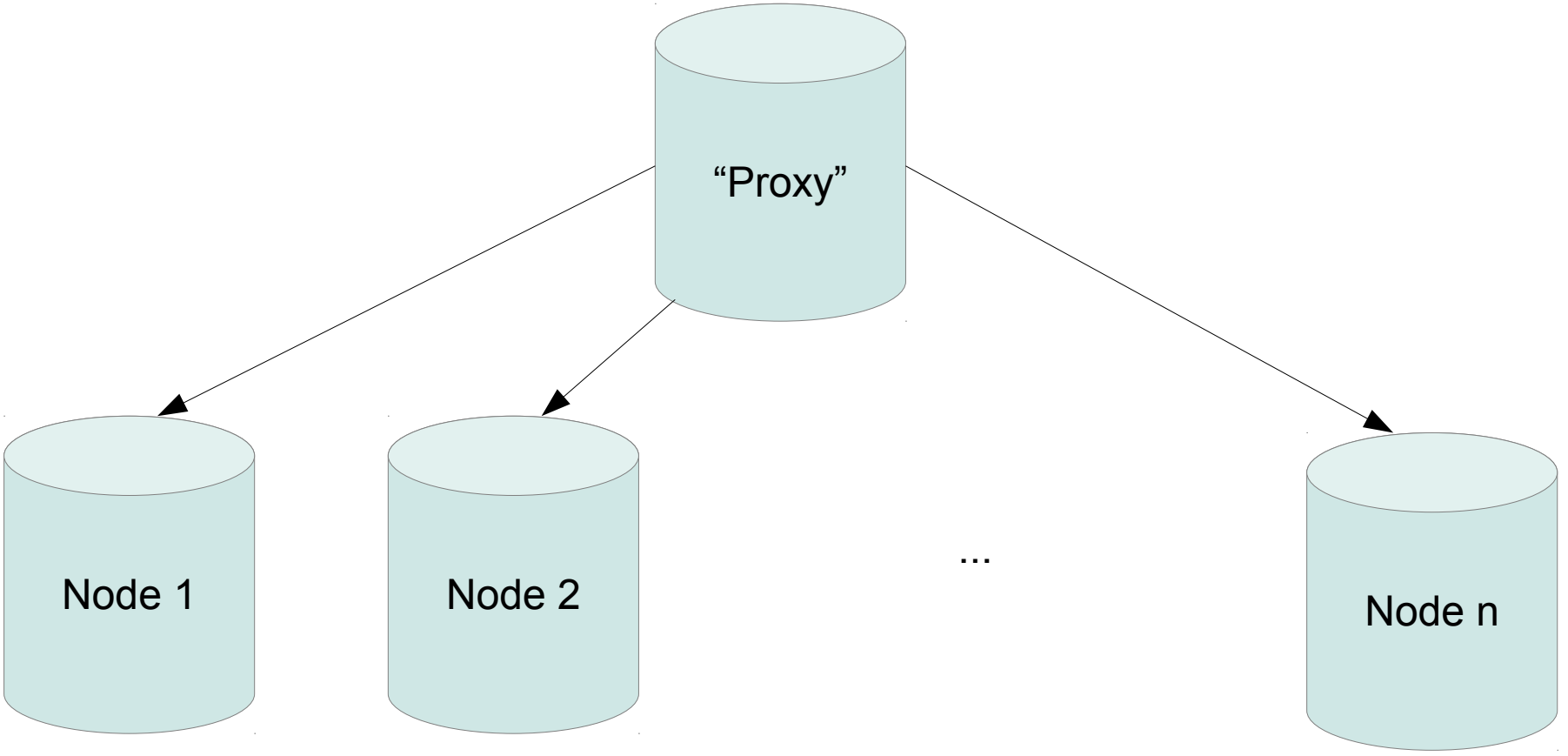
Horizontale Skalierung

- Was tun, wenn wenige Maschinen nicht mehr reichen?

- Wie kann man auf eine ganze Server Farm skalieren?

SQL/MED / PL/Proxy (1)

- Skalieren mittels “Proxy Table”



SQL/MED / PL/Proxy (2)

- Beispiel:

```
SELECT * FROM freunde('bill');
```

SQL/MED / PL/Proxy (3)

- Funktionsweise:

```
test=# SELECT hashtext('bill');
 hashtext
-----------
 168392334
(1 row)

test=# SELECT hashtext('bill')::bit(3);
 hashtext
----------
 110
(1 row)

test=# SELECT hashtext('bill')::bit(3)::int4;
 hashtext
----------
        6                                  <-- Wir routen auf Knoten Nummer 6
(1 row)
```

CYBERTEC
The PostgreSQL Database Company

SQL/MED

Foreign data wrapper:

```
CREATE FOREIGN TABLE
Command:     CREATE FOREIGN TABLE
Description: define a new foreign table
Syntax:
CREATE FOREIGN TABLE [ IF NOT EXISTS ] table_name ( [
  { column_name data_type [ NULL | NOT NULL ] }
    [, ... ]
] )
  SERVER server_name
[ OPTIONS ( option 'value' [, ... ] ) ]
```

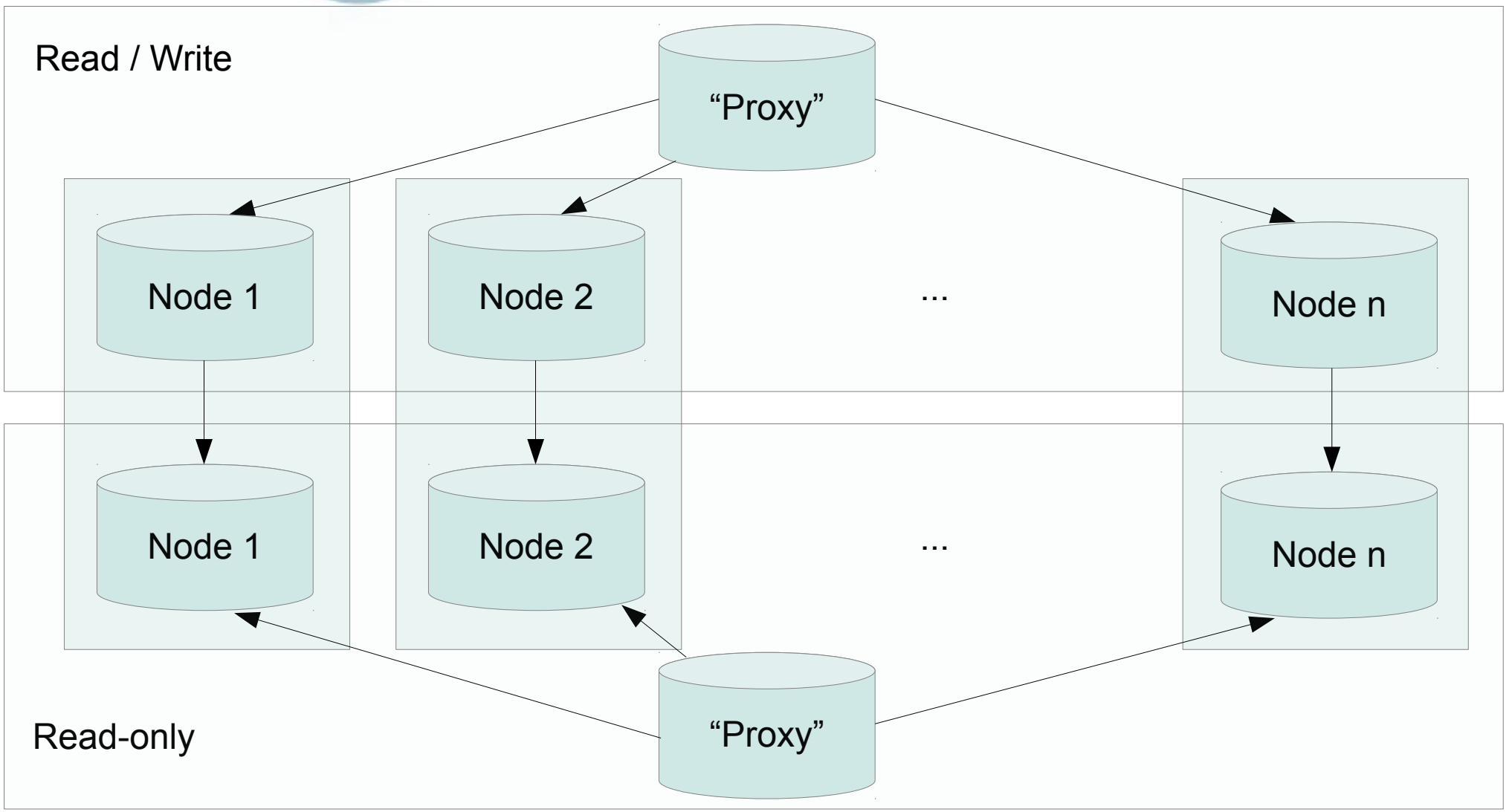
Maximale Skalierbarkeit

- Skalierung mittels Verteilung

- Parallelisierung durch Shared-Nothing Ansatz

- Maximierung der Verfügbarkeit

Maximale Verfügbarkeit



Fazit

Herzlichen Dank für Ihre Aufmerksamkeit

www.postgresql-support.de
hs@cybertec.at

Finally ...

Wir suchen Mitarbeiter !

Bewerbungen bitte an
hs@cybertec.at