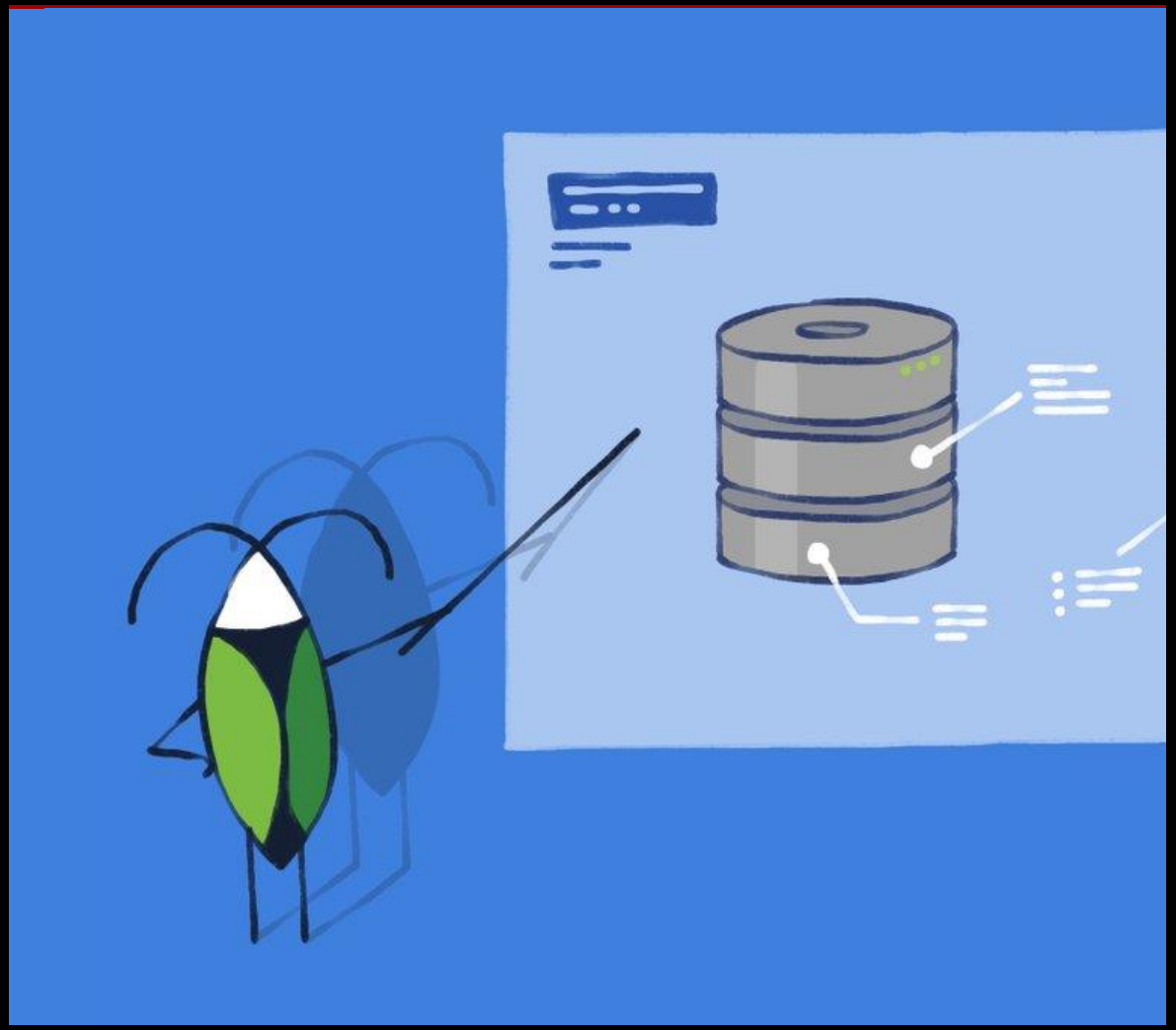
CockroachDB
Einführung, Aufbau und Anwendung von NewSQL

Marvin Bermel, PSE, Böblingen, 21-11-2020
- Modul 3 / Datenbanken -

Mercedes-Benz

Agenda



- Wie wurden Datenbanken entwickelt?
  - Historischer Überblick
- Wer ist CockroachDB?
  - Idee und externe Komponenten
- Was ist CockroachDB?
  - Aufbau und Eigenschaften
- Was bietet CockroachDB noch?
  - Sonstige Eigenschaften
- Anhang

Welche Probleme adressieren moderne Datenbanken?
- Historie und Motivation -

- Historie -

Prä - Internet - Konzept

- Definierte Daten
- Geringes Datenaufkommen
- Lokale Anwendungen
- Geringes Nutzeraufkommen

Relationale DB

Erprobte Anwendung /SQL
Weite Verbreitung
Einsatz auf Solo-Systemen

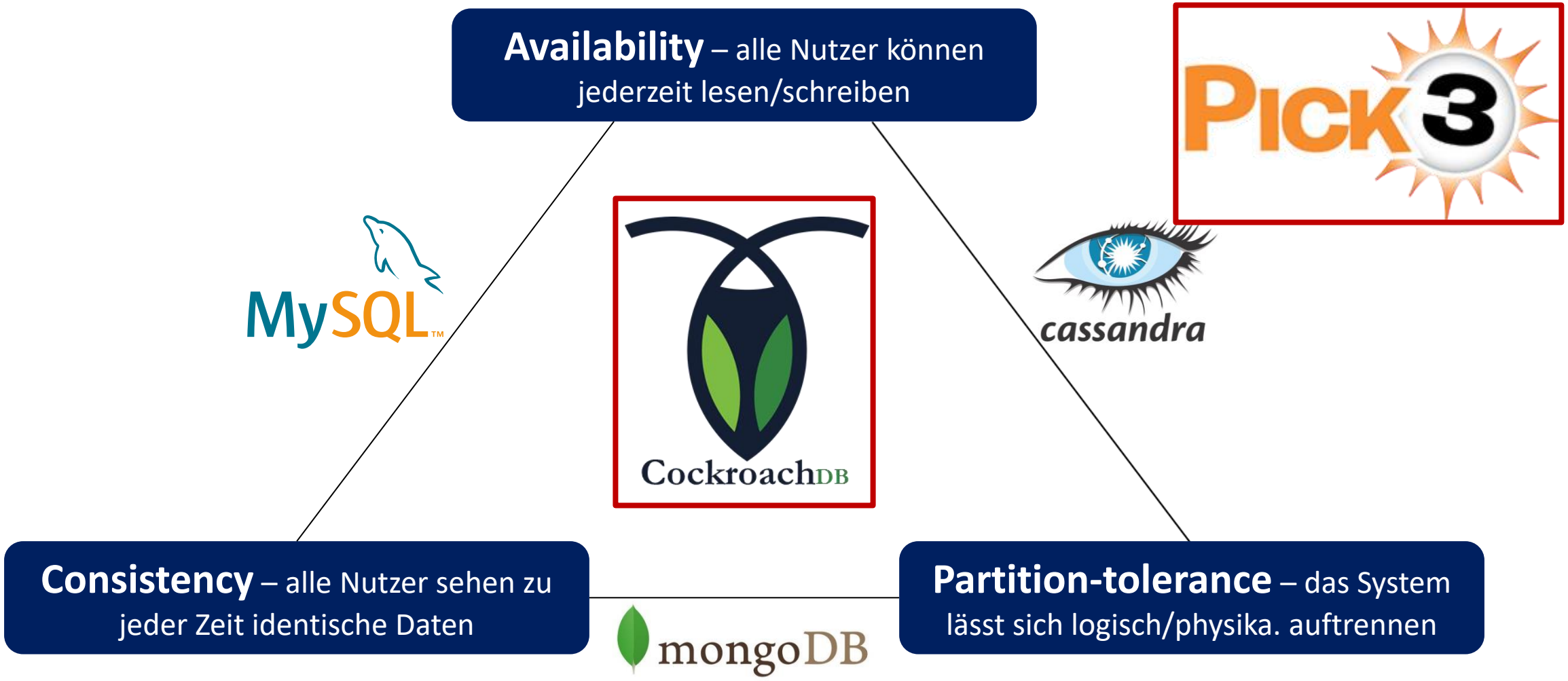
Neue Anwendung / NoSQL
Hohe Diversifikation
Einsatz auf Multi-Systemen

Nicht relationale DB

Post - Internet - Konzept

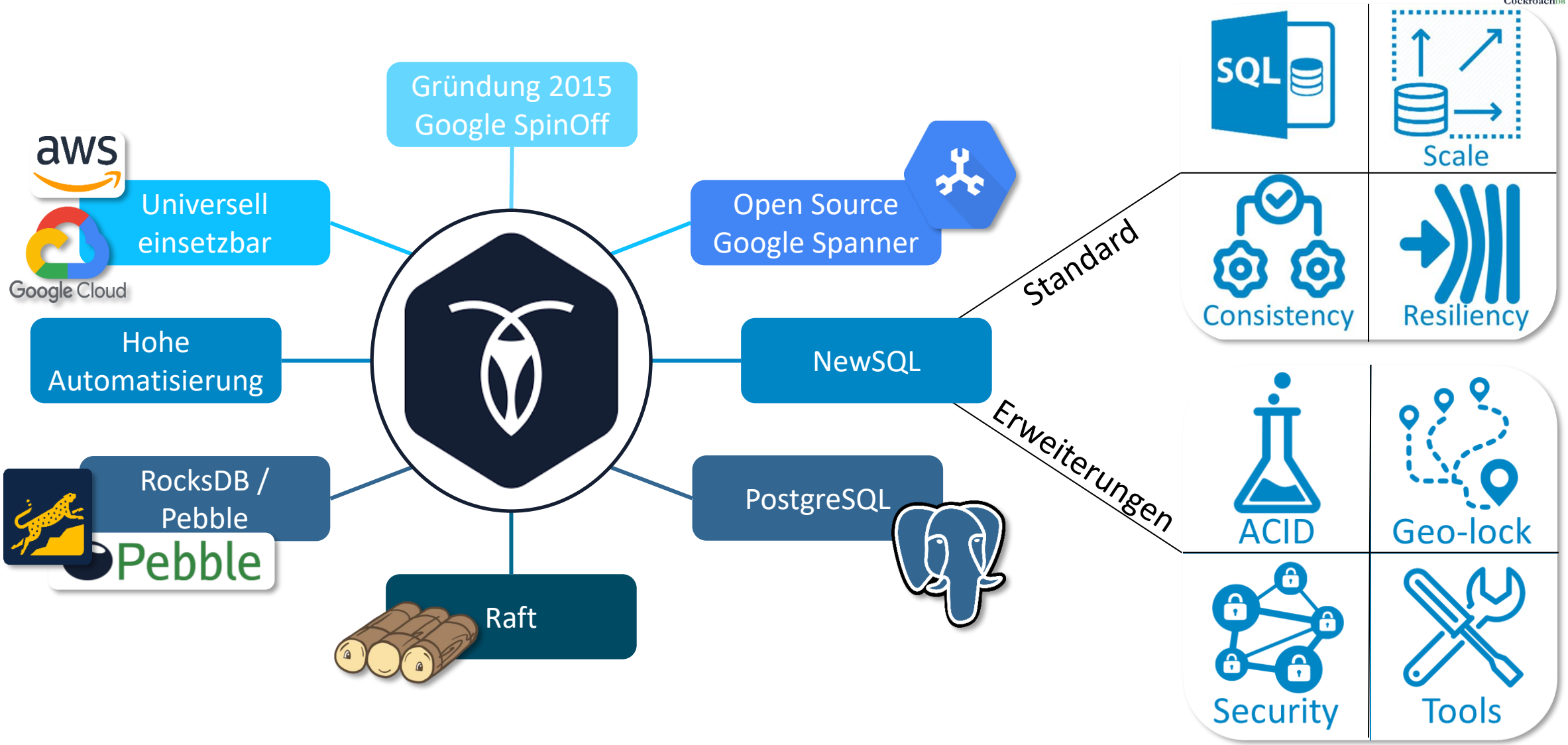
- Unterschiedliche Dateneingabe
- Hohes Datenaufkommen
- Dezentrale Anwendungen
- Hohes Nutzeraufkommen

- Motivation -


Availability – alle Nutzer können jederzeit lesen/schreiben
Pick 3
MySQL
CockroachDB
cassandra
Consistency – alle Nutzer sehen zu jeder Zeit identische Daten
mongoDB
Partition-tolerance – das System lässt sich logisch/physika. auftrennen

Wer ist CockroachDB?
- Konzept und Idee -

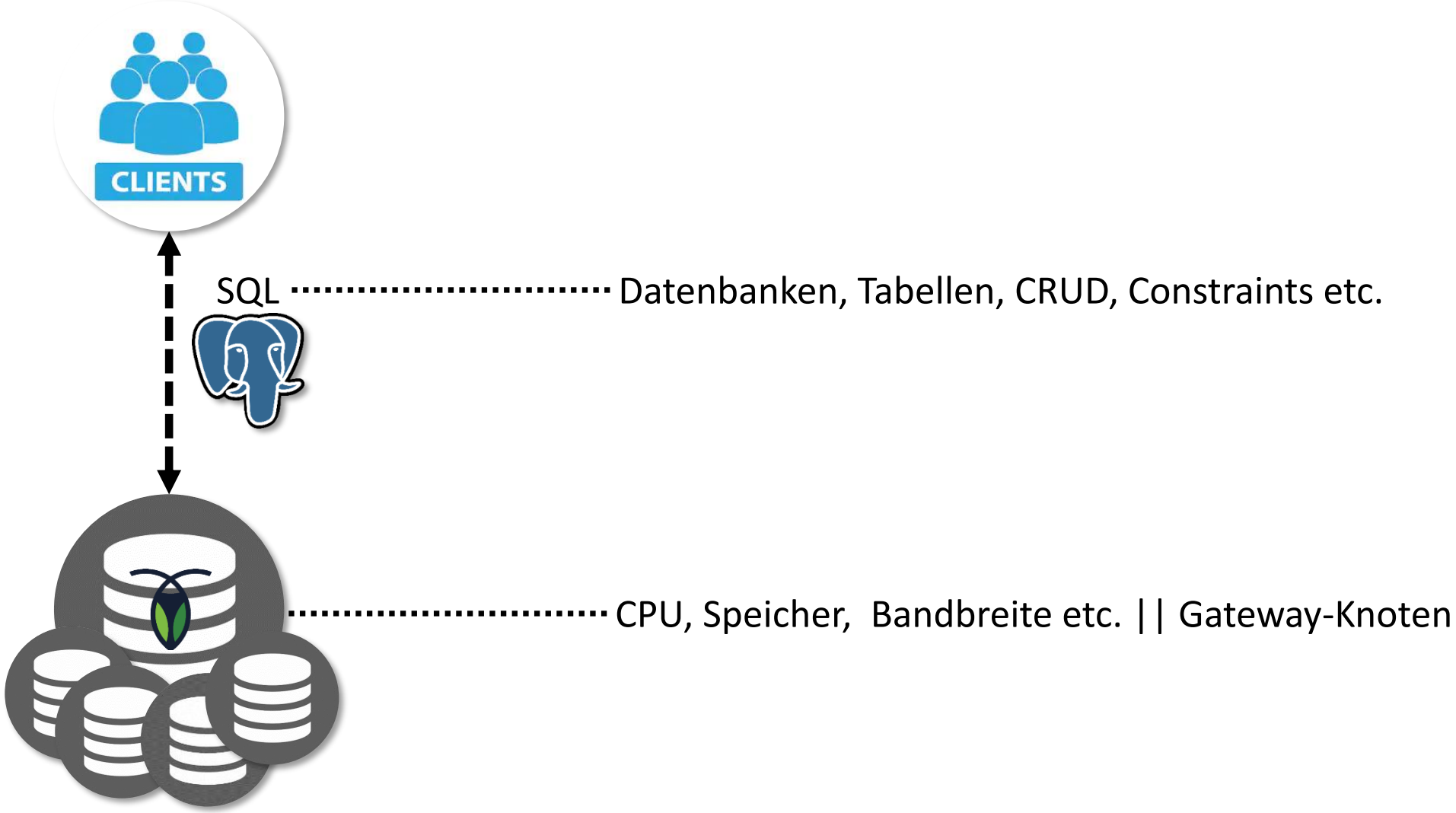
- Konzept und Idee -



Was ist CockroachDB?
- Begriffe, Eigenschaften, Aufbau -

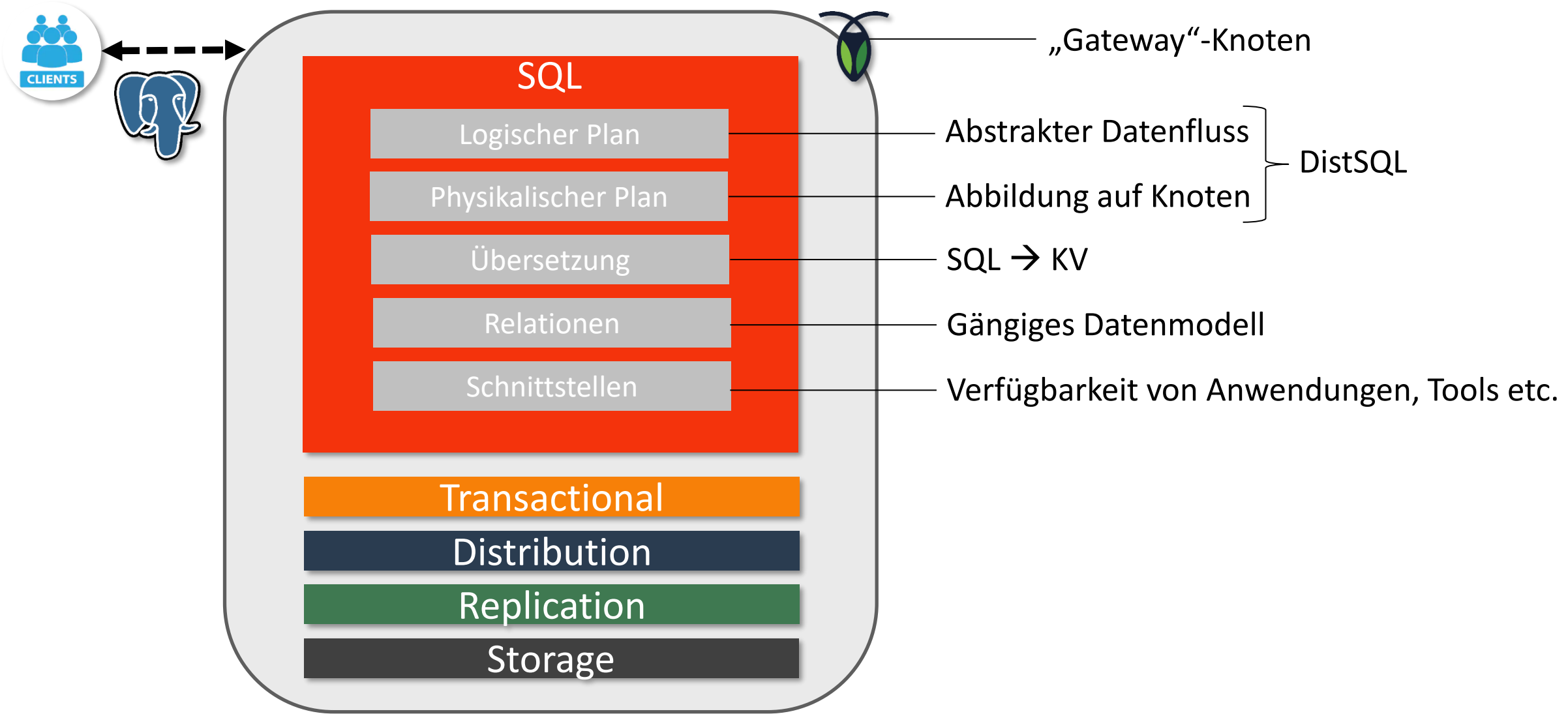
- Aufbau und Eigenschaften –

- Black Box -



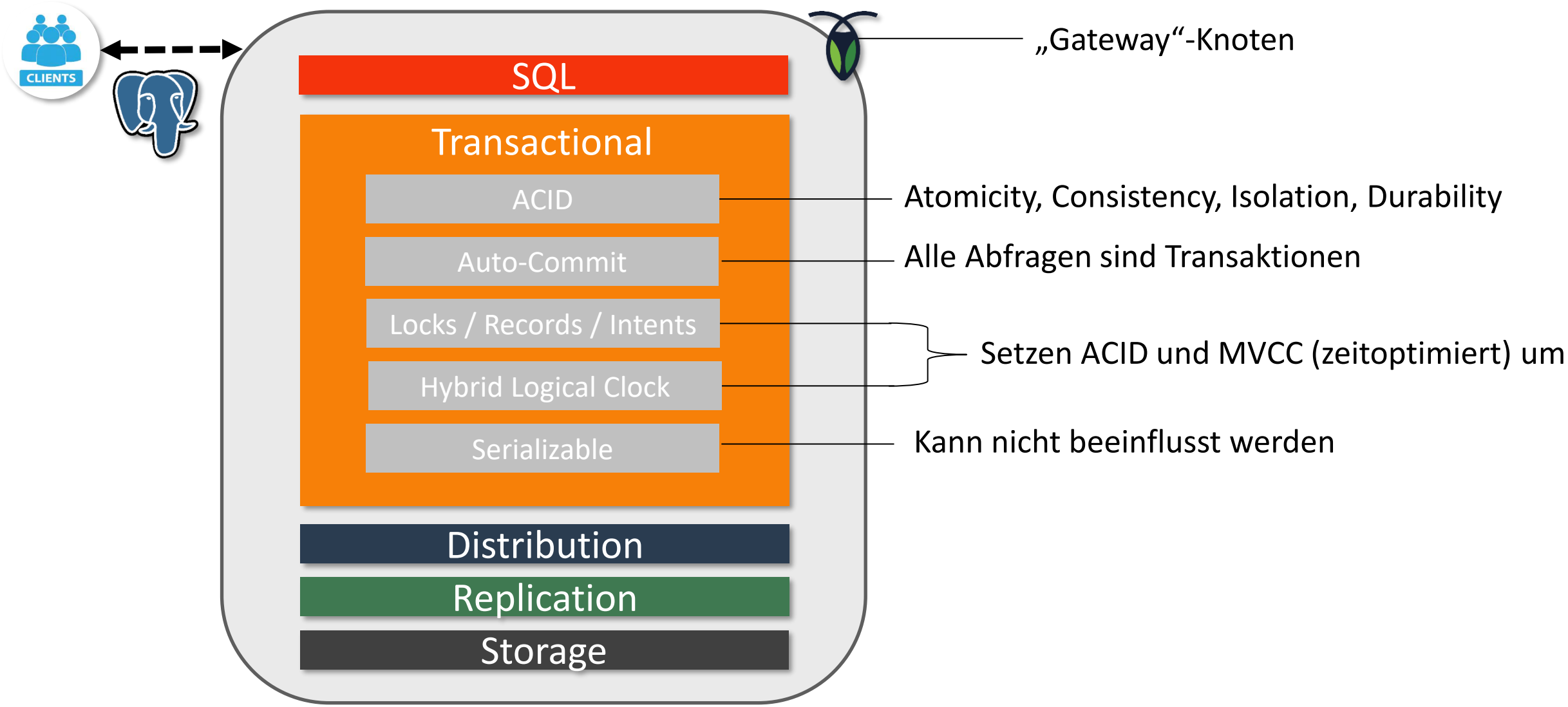
- Aufbau und Eigenschaften –

- White Box -



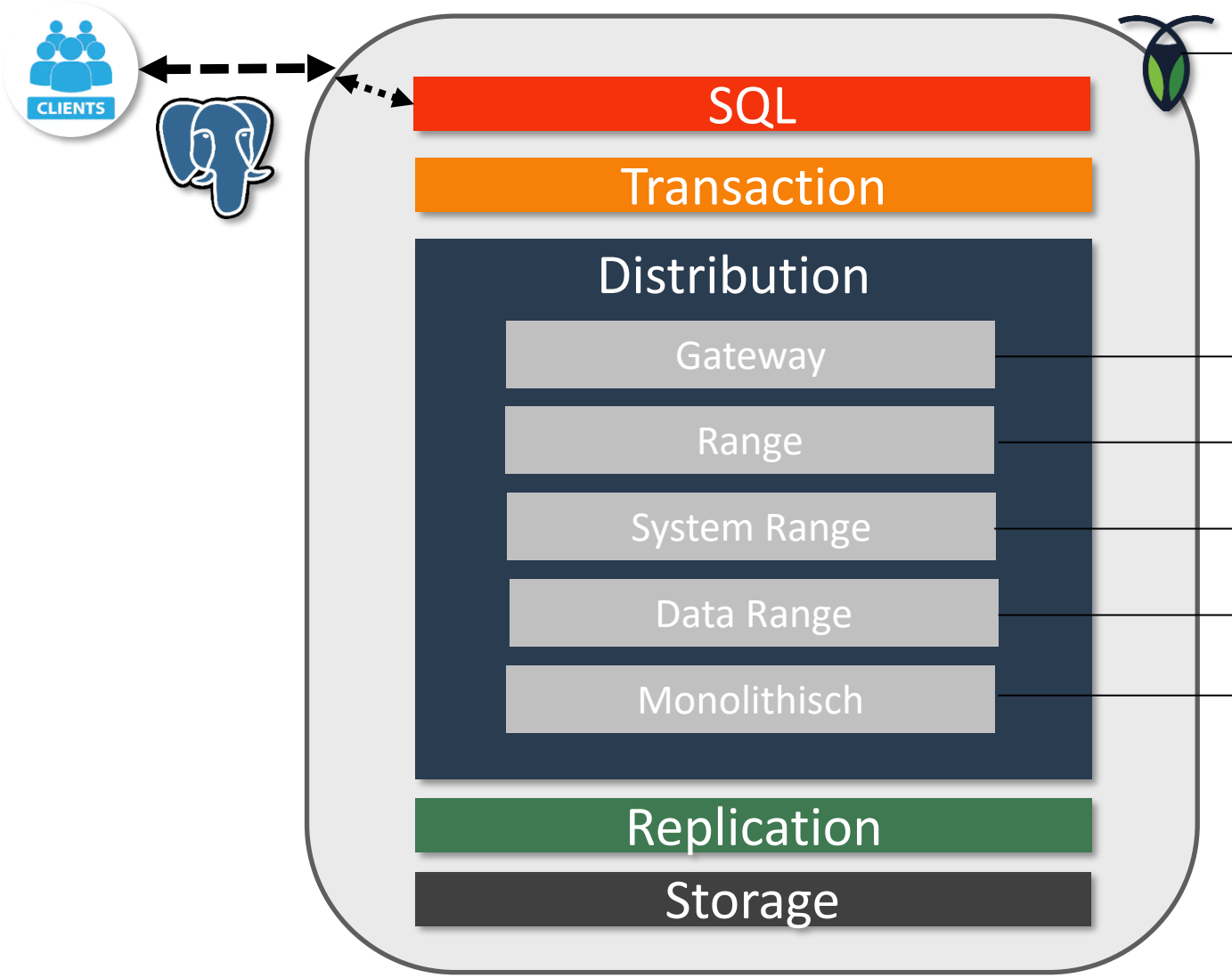
- Aufbau und Eigenschaften –

- White Box -



- Aufbau und Eigenschaften –

- White Box -



„Gateway“-Knoten

Verbindung zum Cluster

Einheit der Datenpakete. Default: 512MB

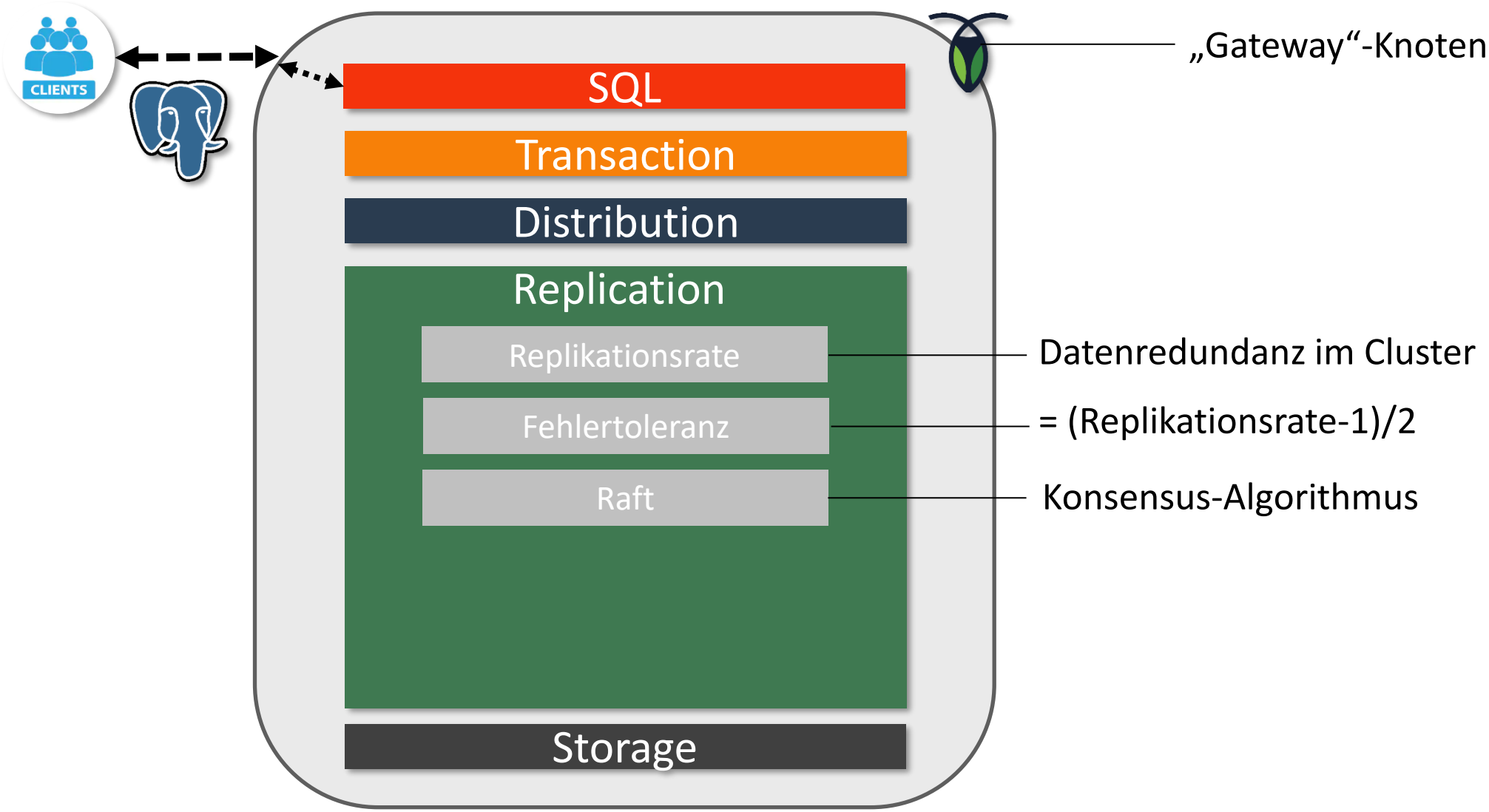
Enthalten Daten wie Port-Informationen etc.

Enthalten Nutzdaten-Verwalter

Jede Relation besitzt einen eigenen Key-Space

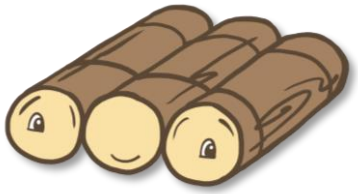
- Aufbau und Eigenschaften –

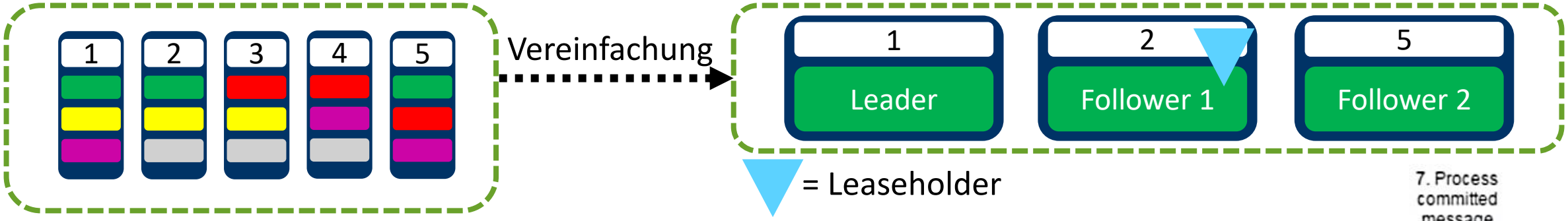
- White Box -



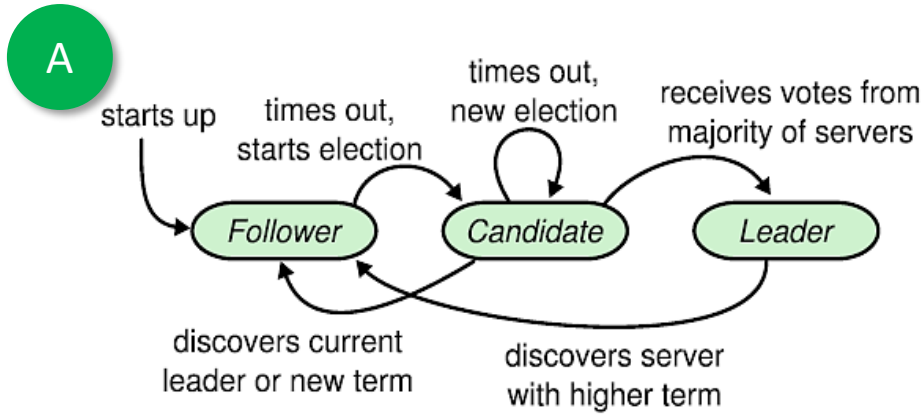
- Raft -

- Konsensus Algorithmus-

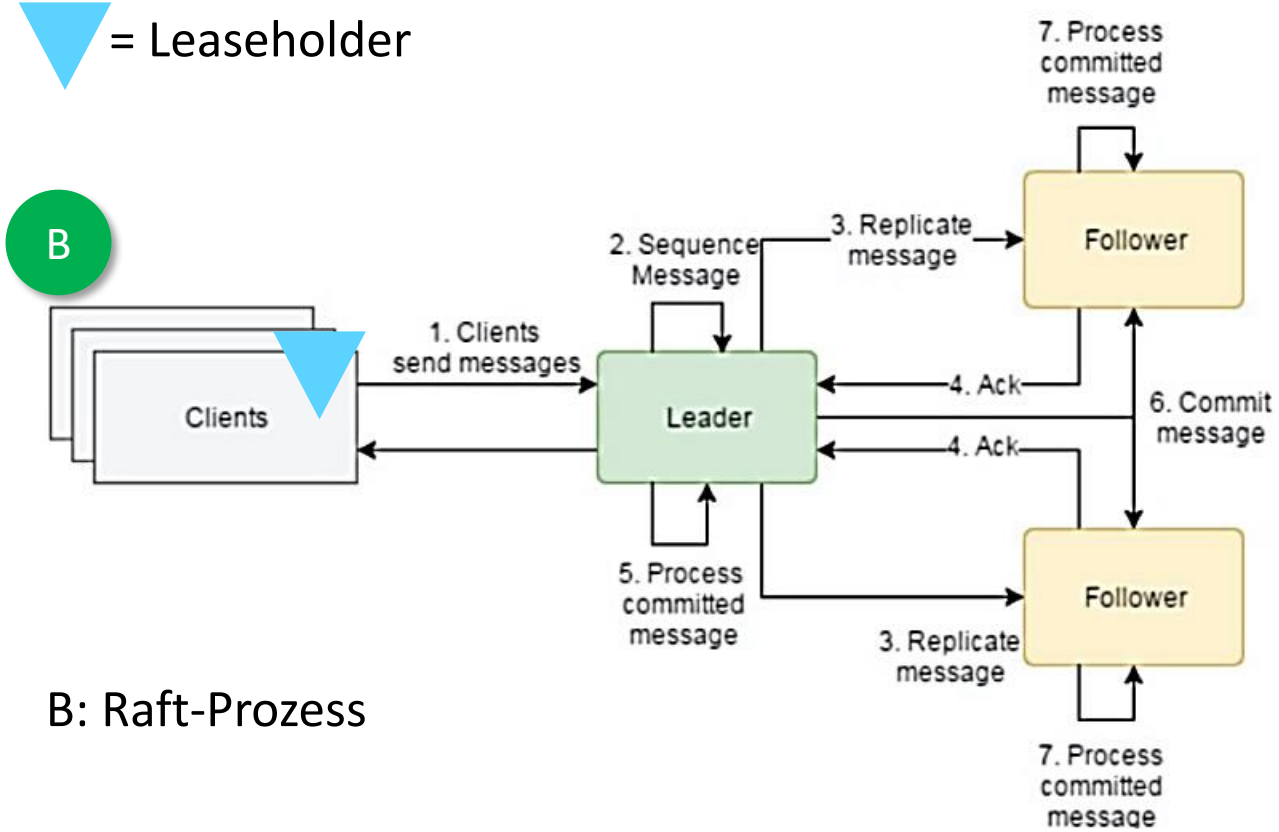




5 Knoten Cluster, 5 Ranges 3 Replikationen



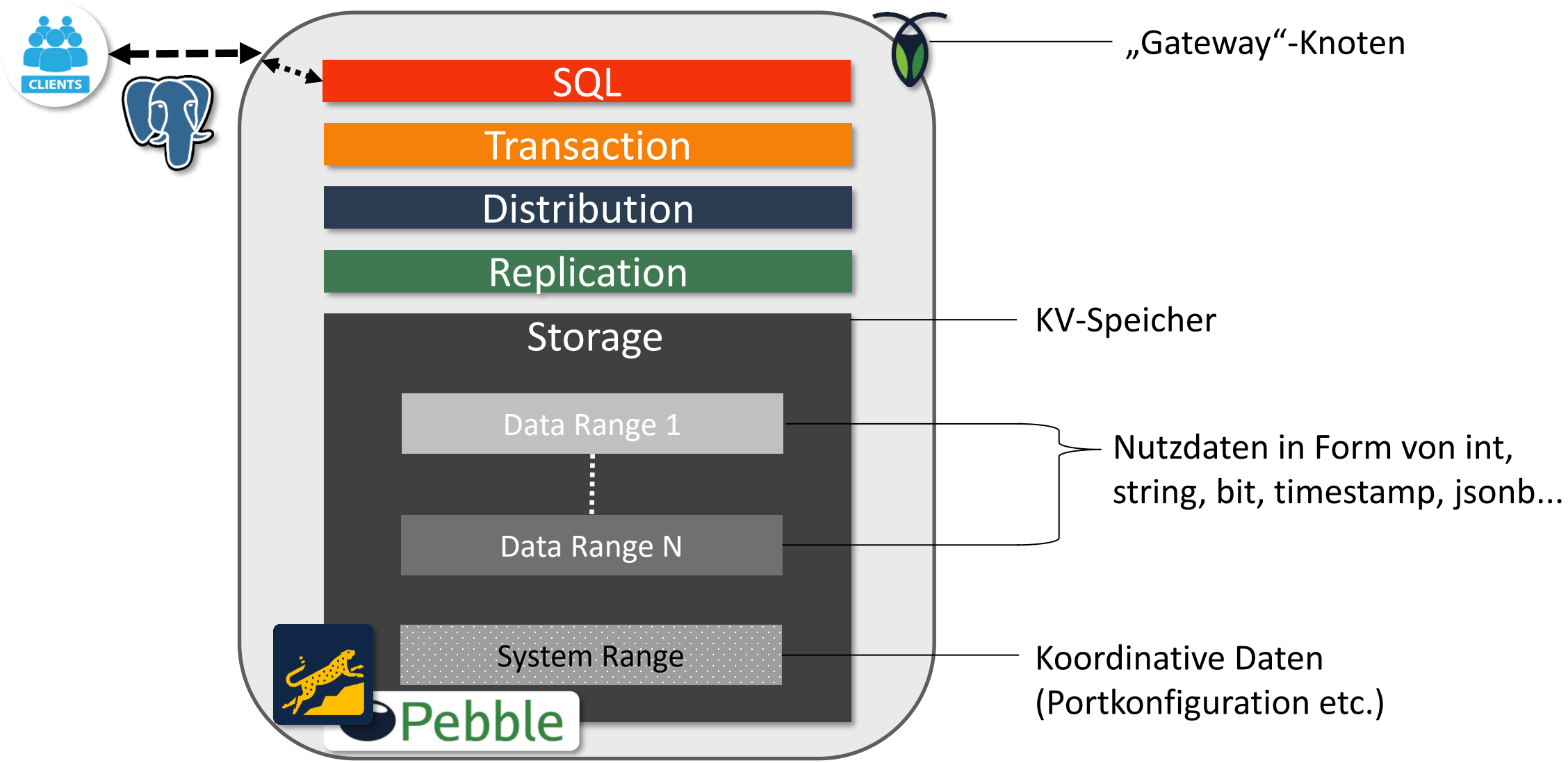
A: Leader-Wahl



B: Raft-Prozess

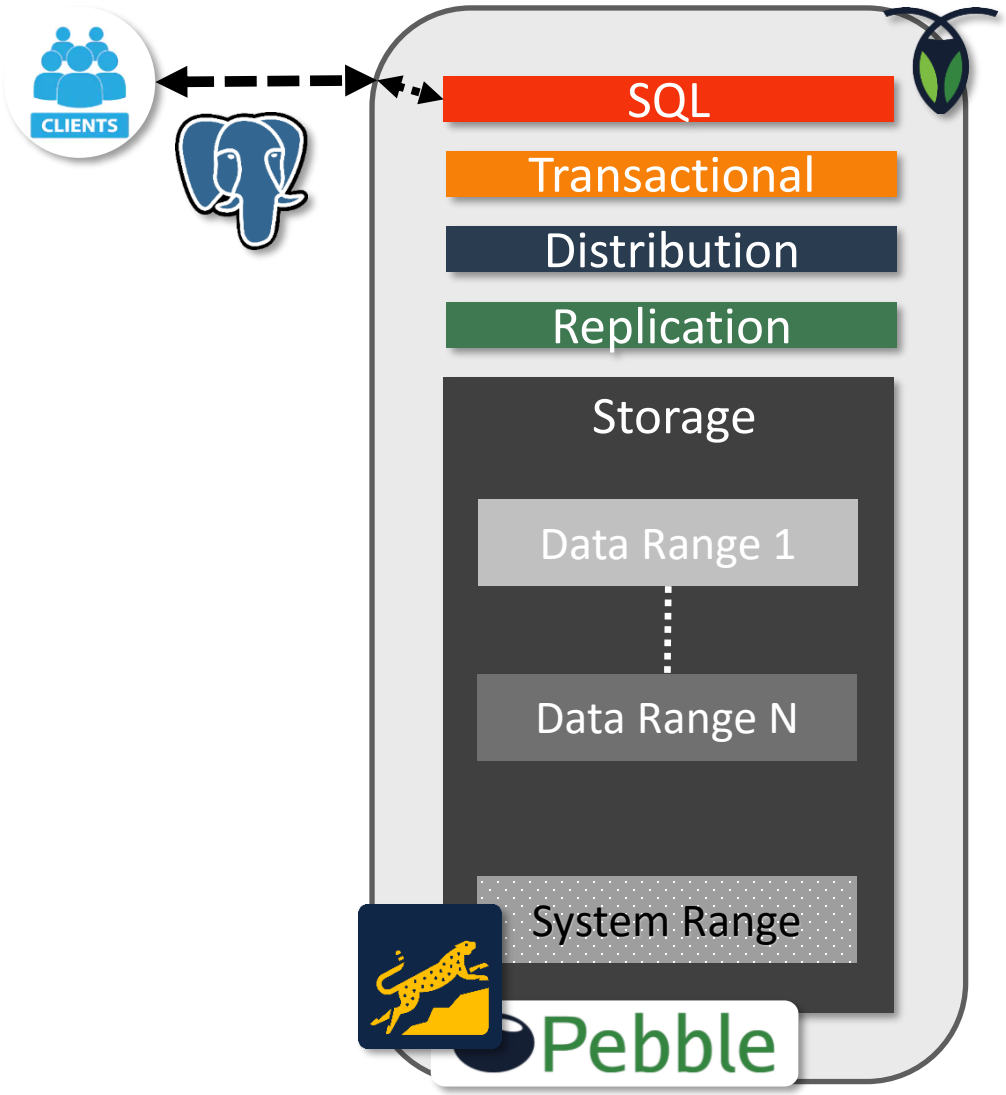
- Aufbau und Eigenschaften –

- White Box -



- Aufbau und Eigenschaften –

- White Box -



Datenmodellierung

```
CREATE TABLE inventory (
      id INT PRIMARY KEY,
      name STRING,
      price FLOAT
)
```

ID	Name	Price
1	Bat	1.11
2	Ball	2.22
3	Glove	3.33

Key	Value
/inventory/primary/1	“Bat”,1.11
/inventory/primary/2	“Ball”,2.22
/inventory/primary/3	“Glove”,3.33

Relational

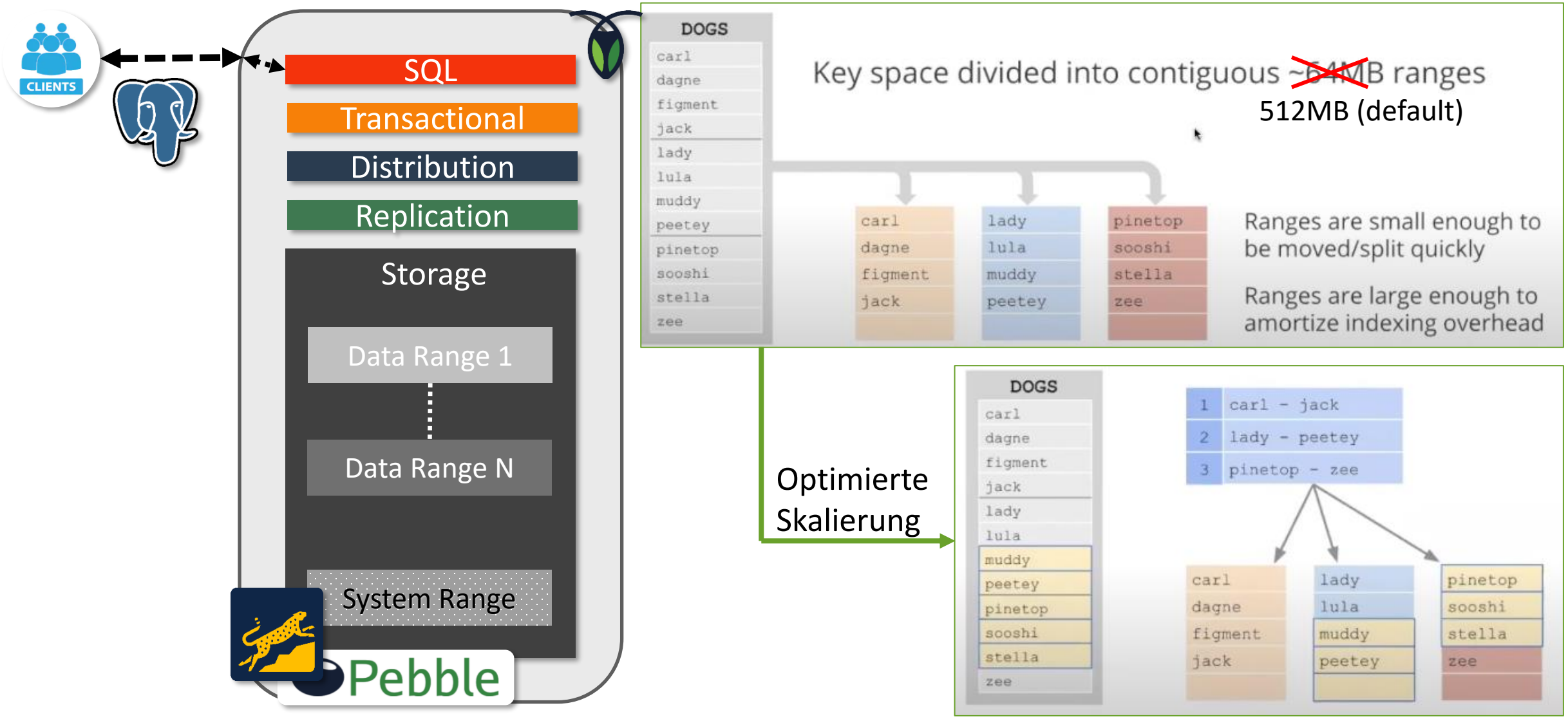
Key-Value

ID	Name	Price
1	Bat	1.11
2	Ball	2.22
3	Glove	3.33
4	Bat	4.44

Key	Value
/inventory/name_idx/”Bat”/1	∅
/inventory/name_idx/”Ball”/2	∅
/inventory/name_idx/”Glove”/3	∅
/inventory/name_idx/”Bat”/4	∅

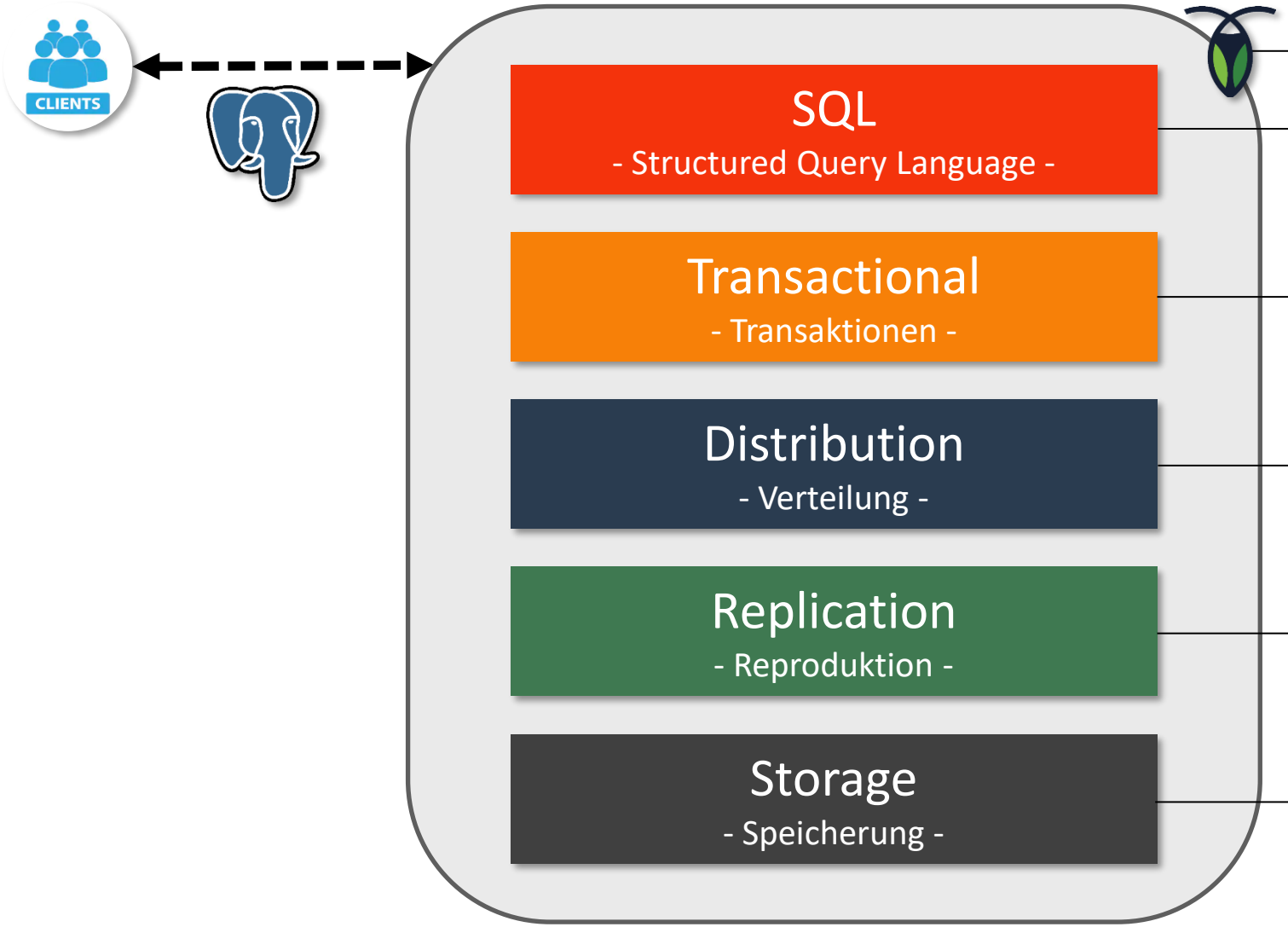
- Aufbau und Eigenschaften –

- White Box -



- Aufbau und Eigenschaften –

- White Box -



„Gateway“-Knoten

Erstellt logische und physikalische Pläne

Erstellt und überwacht Transaktionen

Ordnet Transaktionen KV-Ranges zu, Gateway-Funktionalität

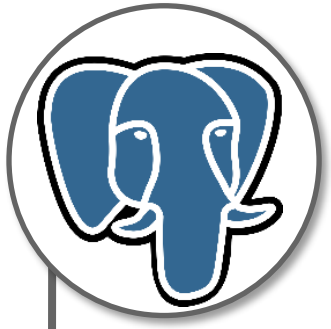
Führt die Transaktion aus, Enthält Replikationskonfiguration

KV-Speicher

Was bietet CockroachDB noch?
- CRUD, Skalierbarkeit, Tools-

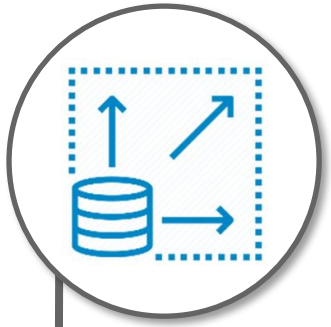
- Eigenschaften -

CRUD



- C -> „Create…“
- R -> „Select…“
- U -> „Update… Set…“
- U -> „Alter Table… X…“
- D -> „Delete… From…“
- D -> „Drop…“

Skalierbarkeit



- Automatisierung
- Geologisch fixiert
- Variable Replikation
- Optimierung

Tools

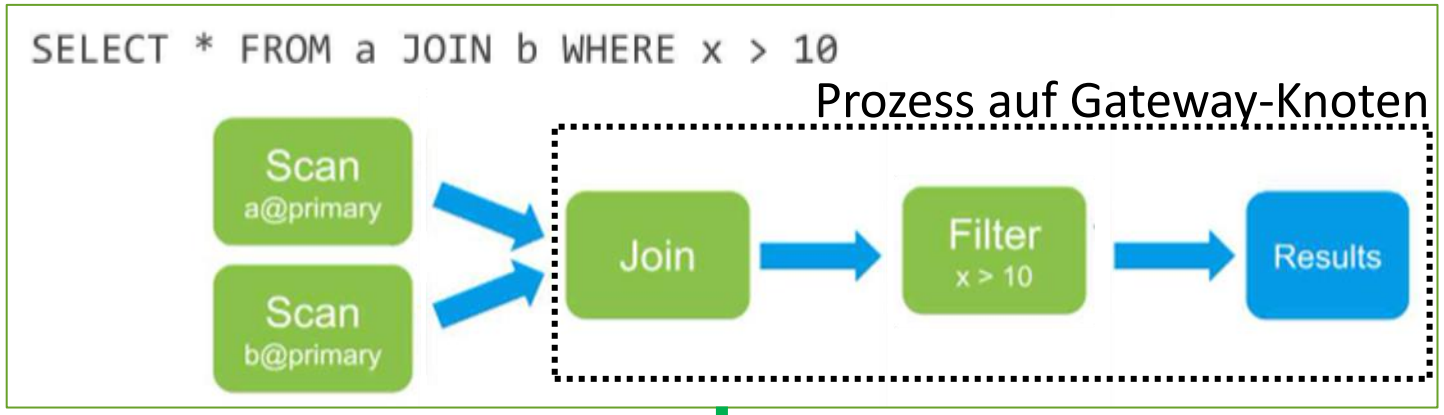
- Admin UI
- Optimierung
- GUIs (DBeaver)

- Optimierte DistSQL -

Tools

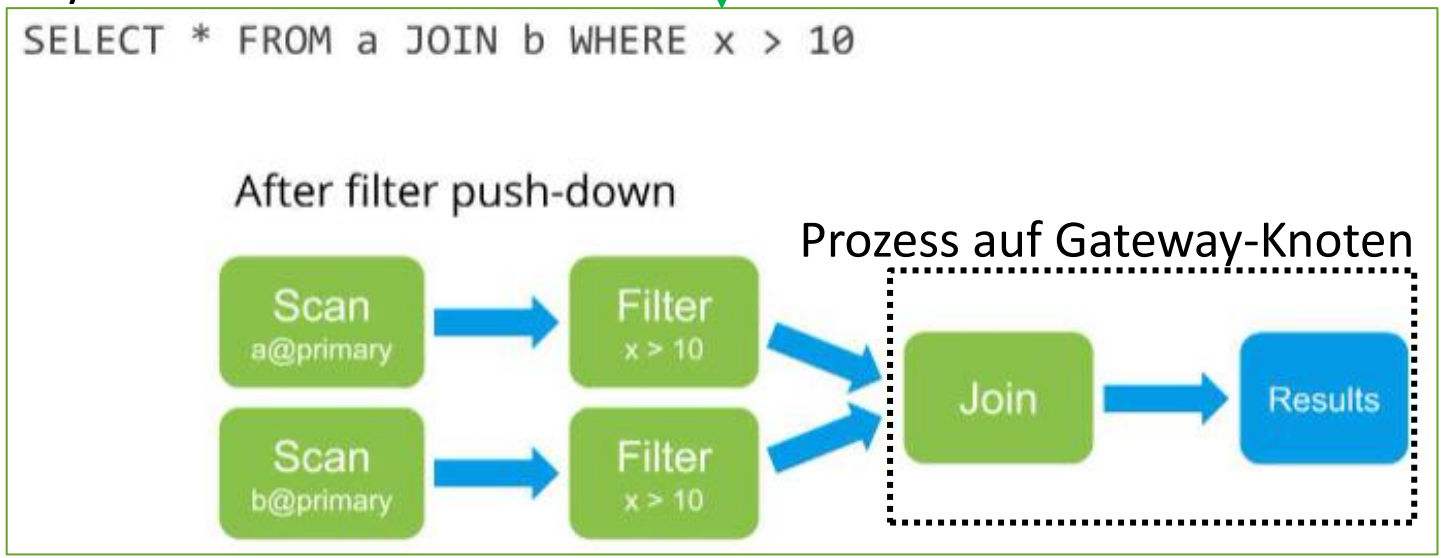
- Admin UI
- Optimierung
- GUIs (DBeaver)

Physikalischer Plan A

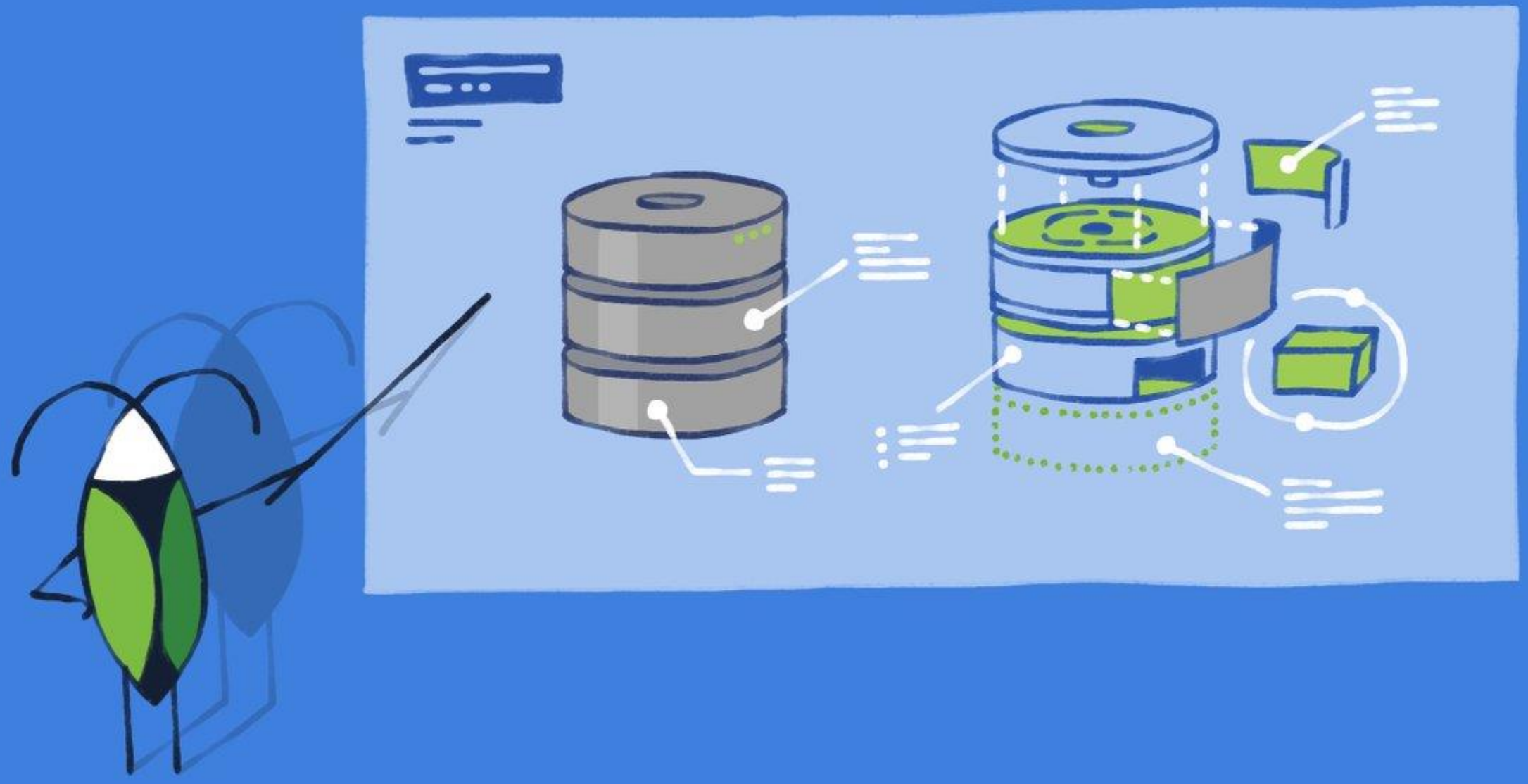


Datenübertragung optimiert

Physikalischer Plan B



– Danke für Ihre und eure Aufmerksamkeit –



https://pbs.twimg.com/media/Dq15tE6UwAEjGsB.jpg

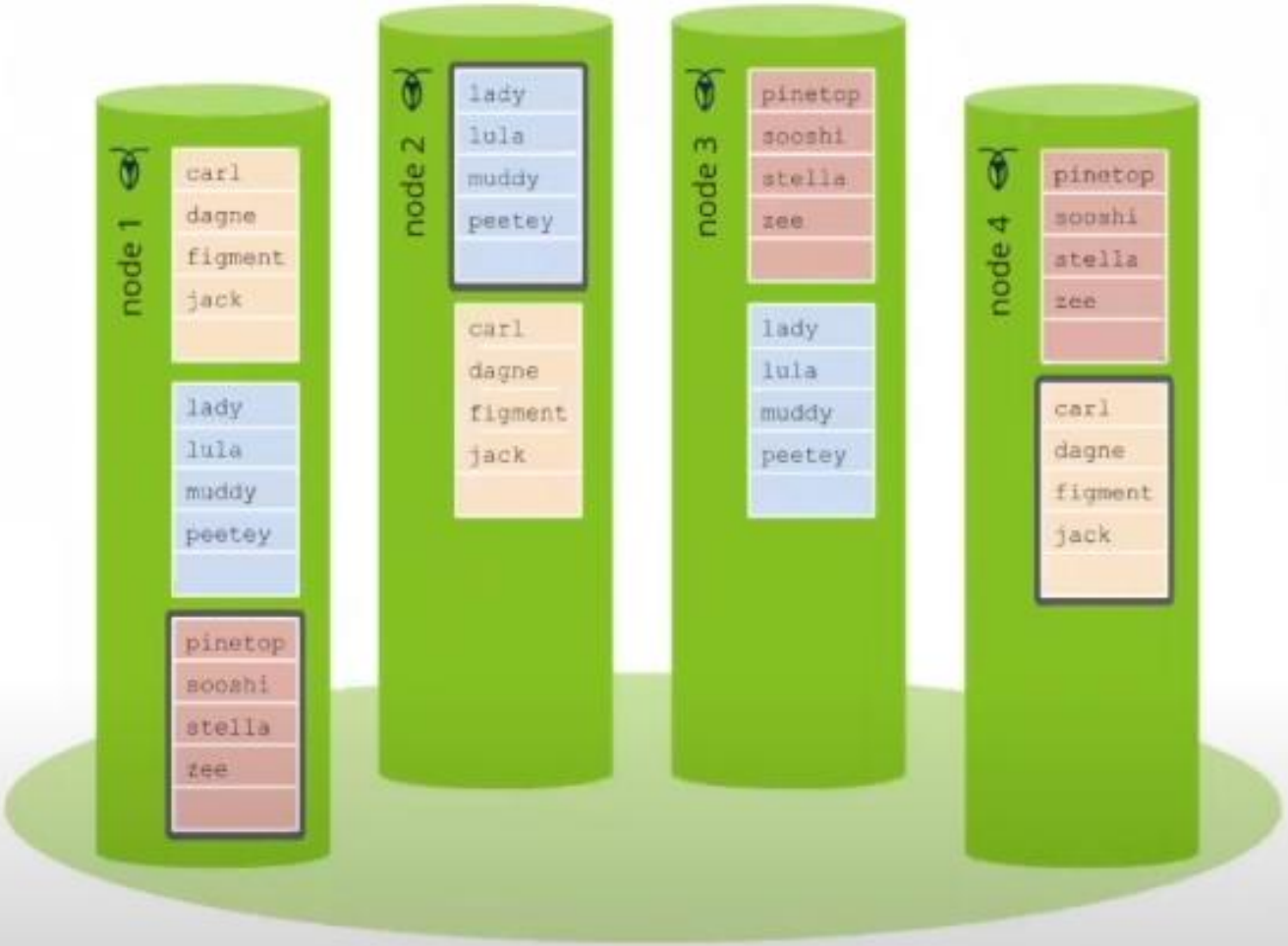
- Sonstiges -

- Vergleich Datenbanken -

	RELATIONAL Single instance, transactions on legacy infrastructure	NOSQL Global, optimized for read access to data	DISTRIBUTEDSQL Architected for transactional cloud applications
Scale	◔ Difficult manual shard or asynchronous replication	◐ Automated for read only access of data	● Simple, global scale for reads and writes
Resilience	◕ Active passive failover creates RPO lag	● Distributed data allows for quick global reads	● All active redundancy eliminates RPO
Transactions	● Ensures consistent transactions	◔ Limited transactional capability	● Serializable isolation ensures consistency
Cloud	○ Architected for legacy infrastructure	◕ Architected for web, read only infrastructure	● Architected for cloud-native apps

- Transaktionen -

- Schritt 1 -

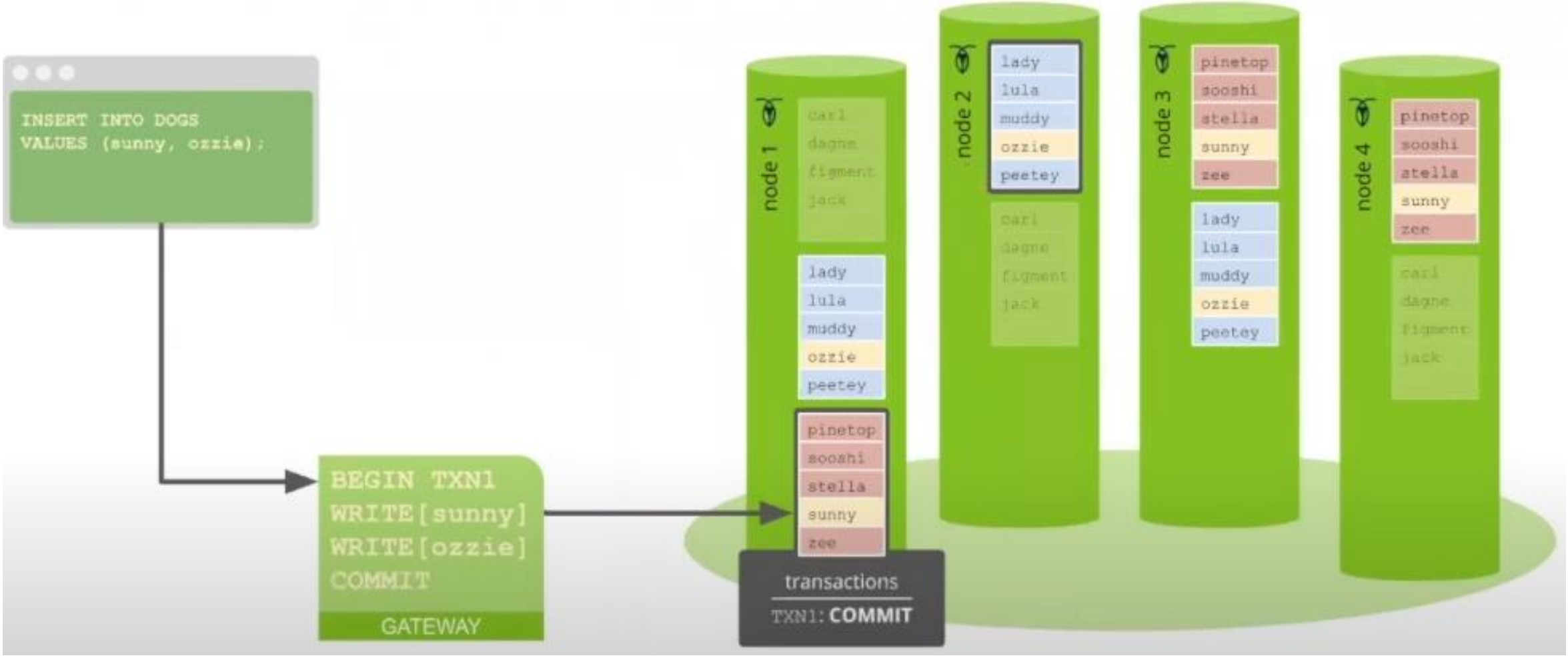


- Transaktionen -

- Schritt 2 -

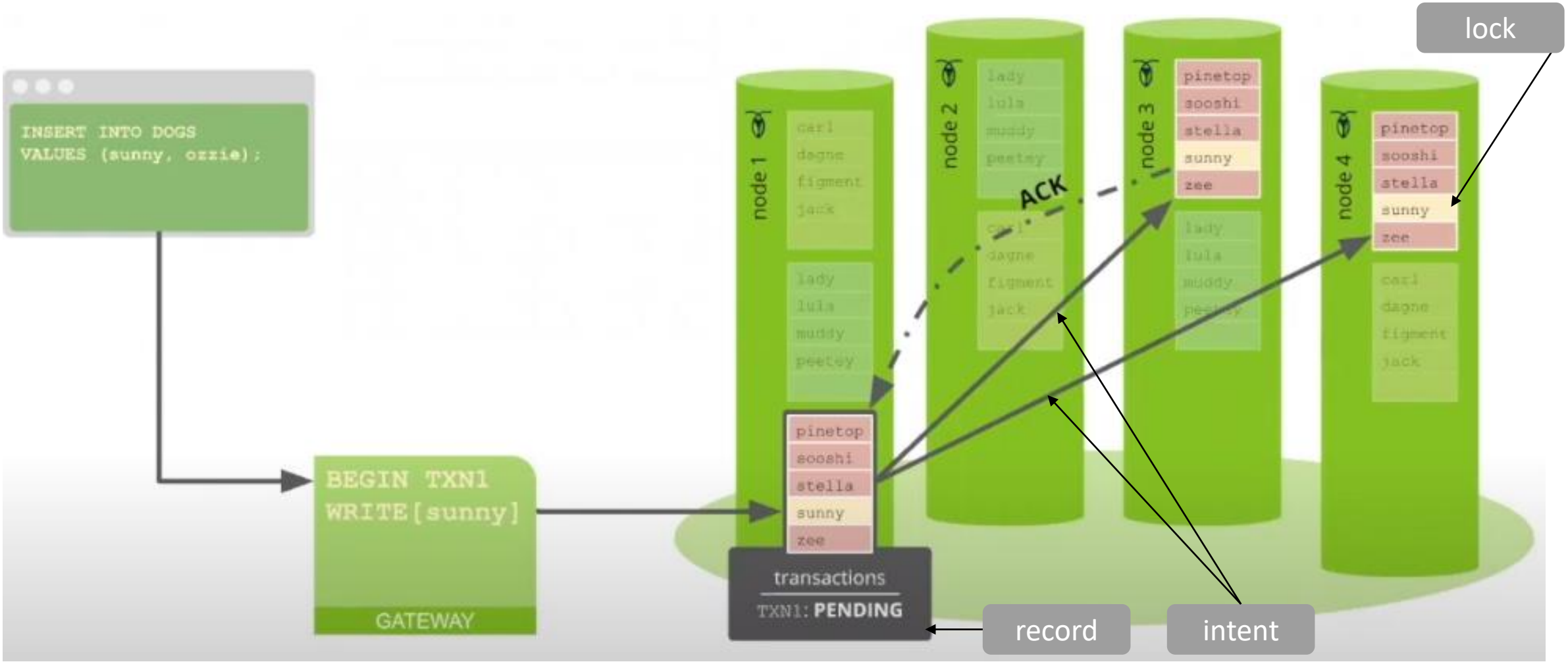
- Transaktionen -

- Schritt 3 -



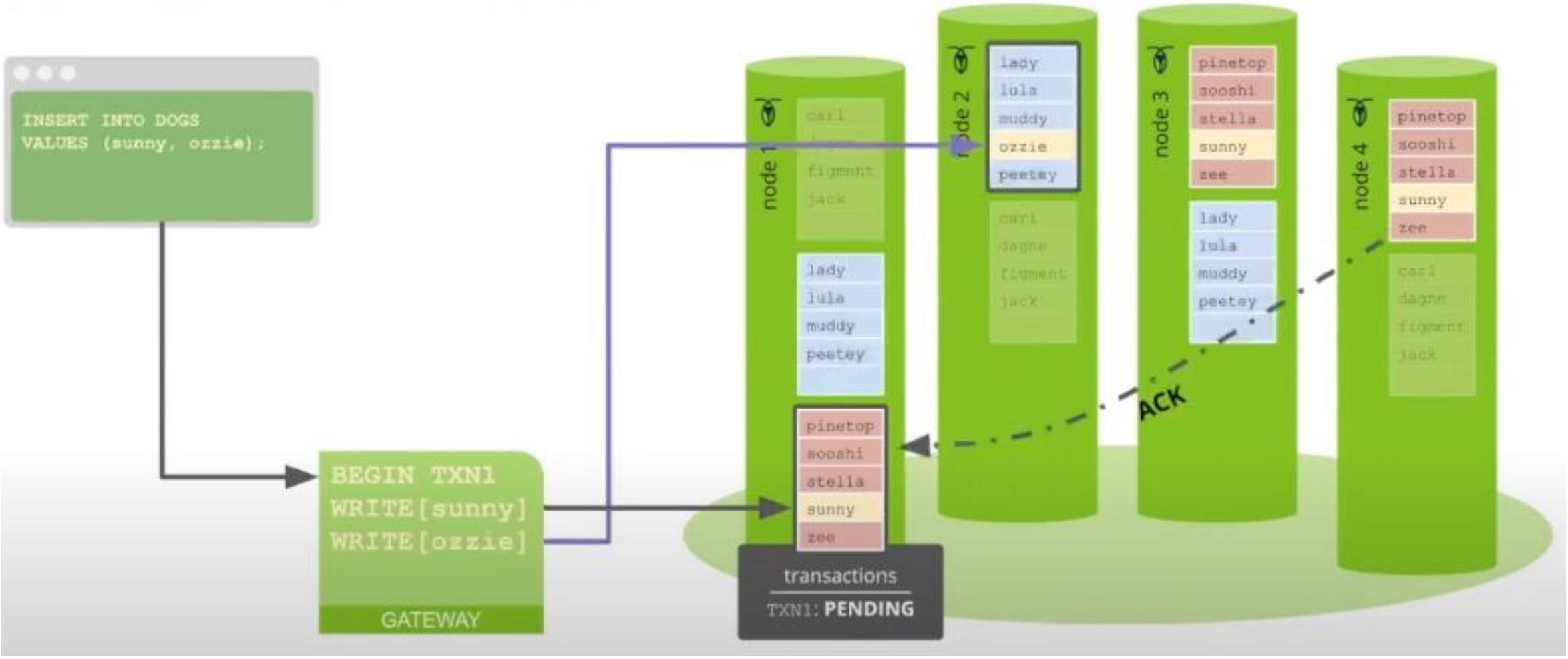
- Transaktionen -

- Schritt 4 -



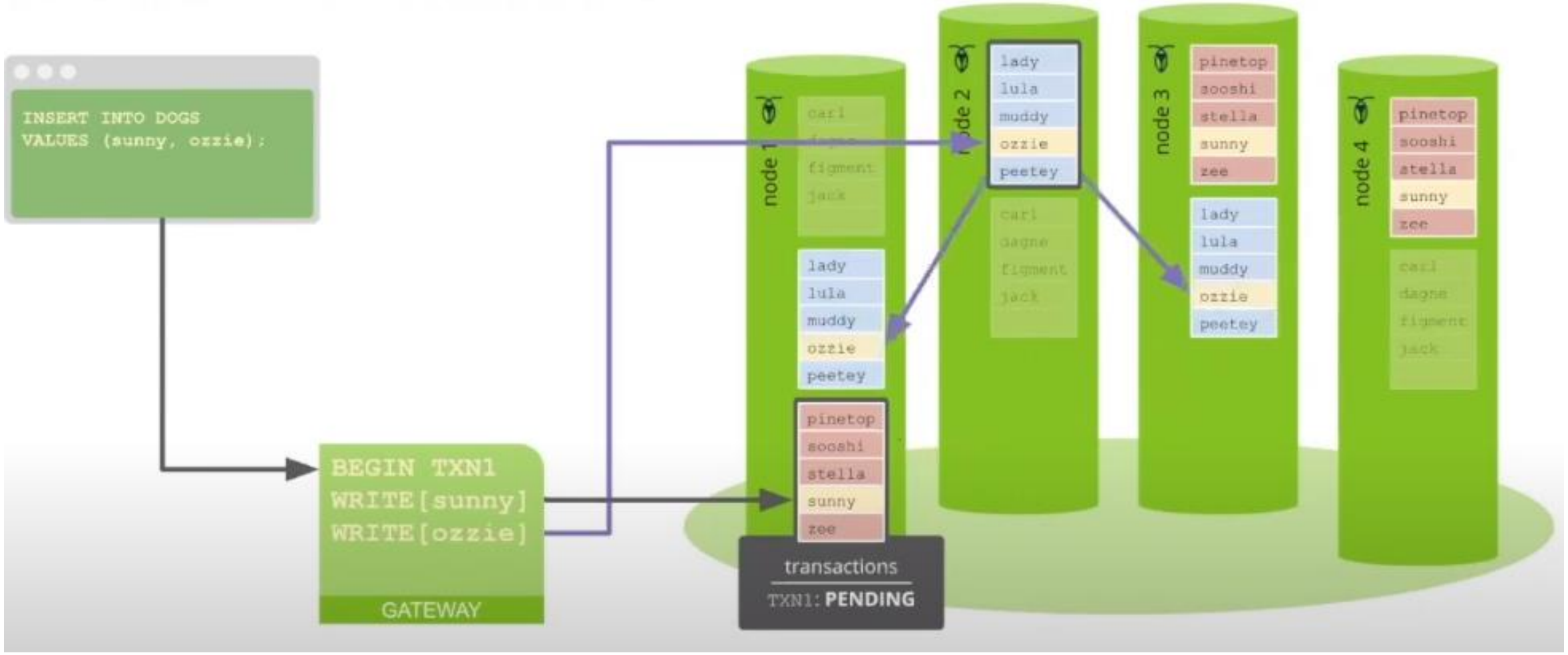
- Transaktionen -

- Schritt 5 -



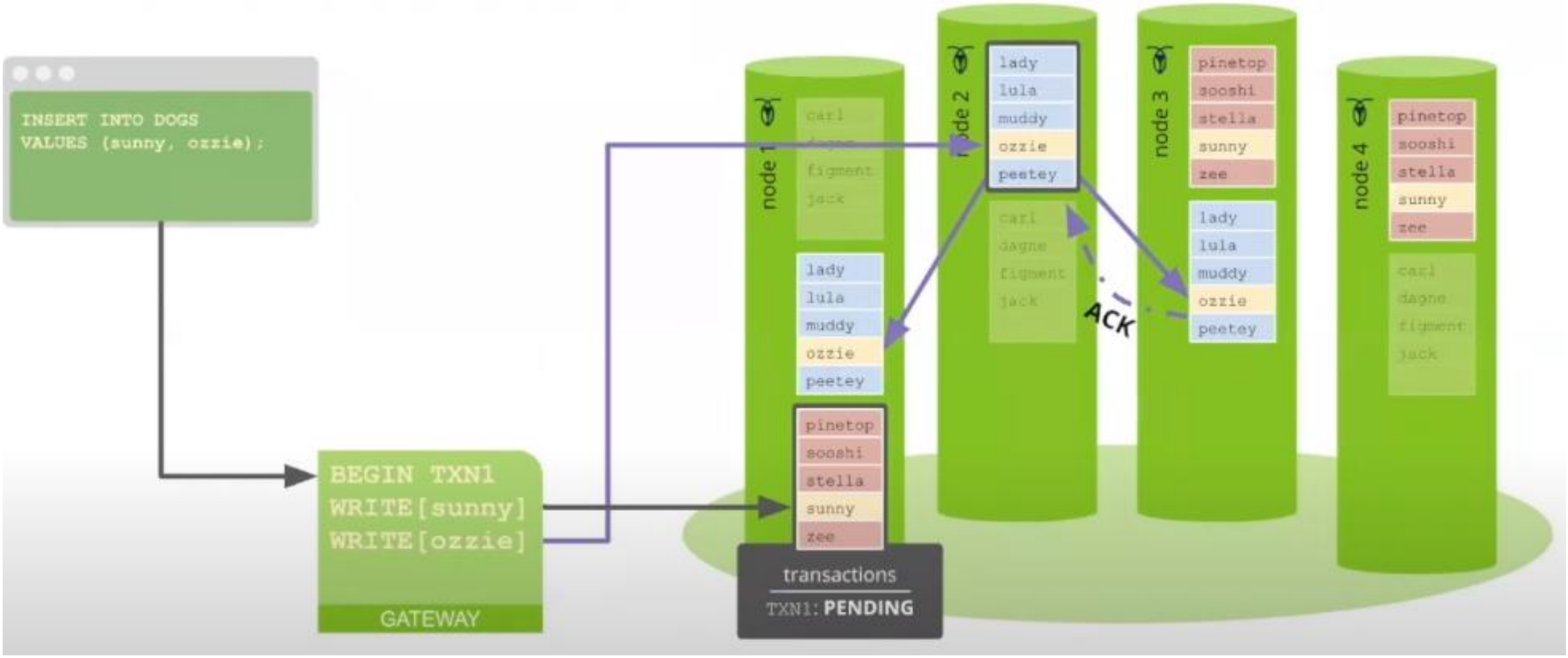
- Transaktionen -

- Schritt 6 -



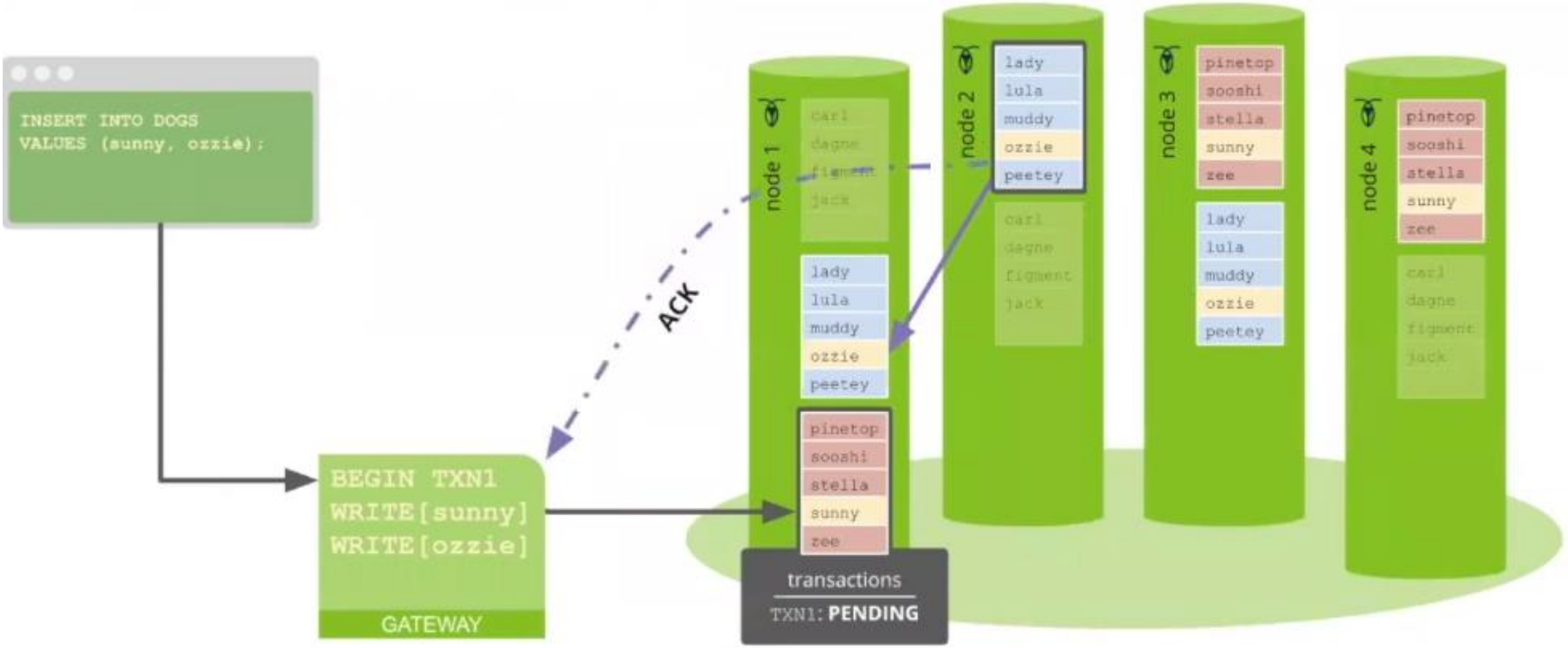
- Transaktionen -

- Schritt 7 -



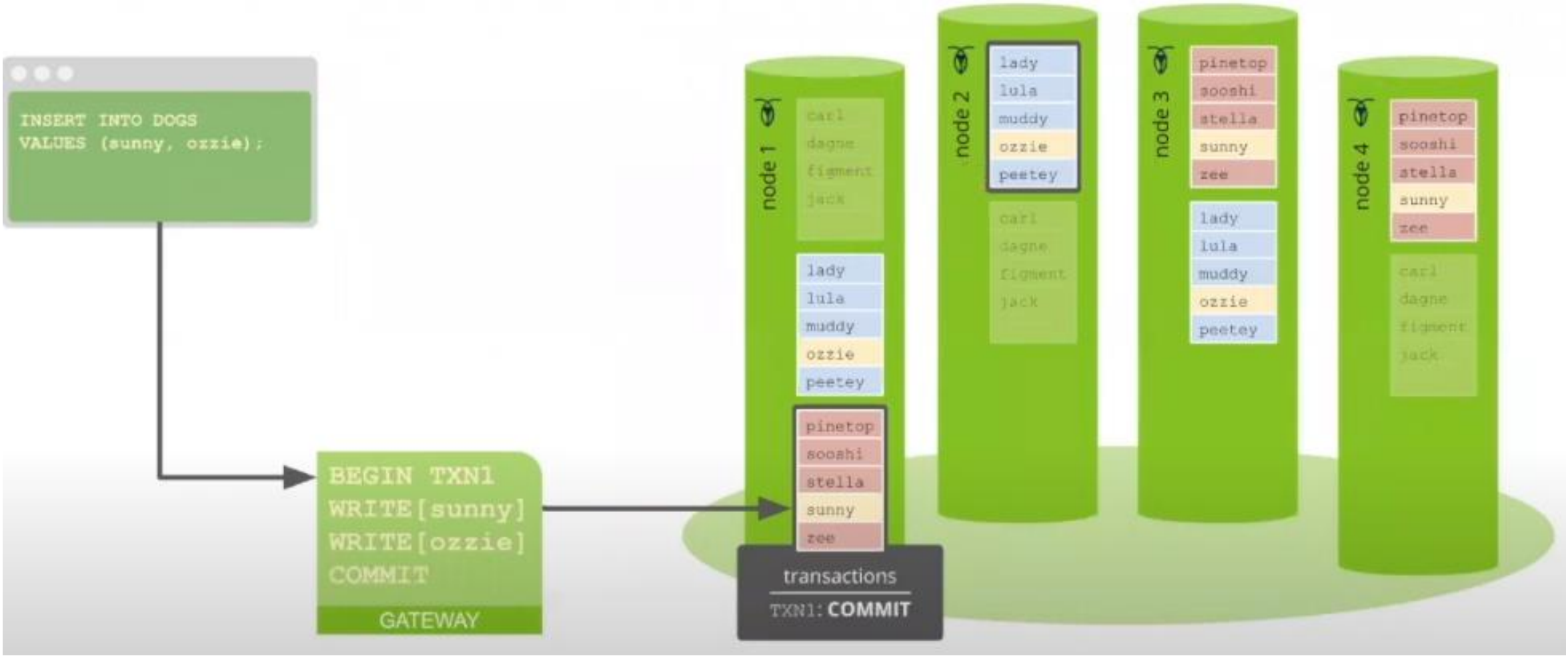
- Transaktionen -

- Schritt 8 -



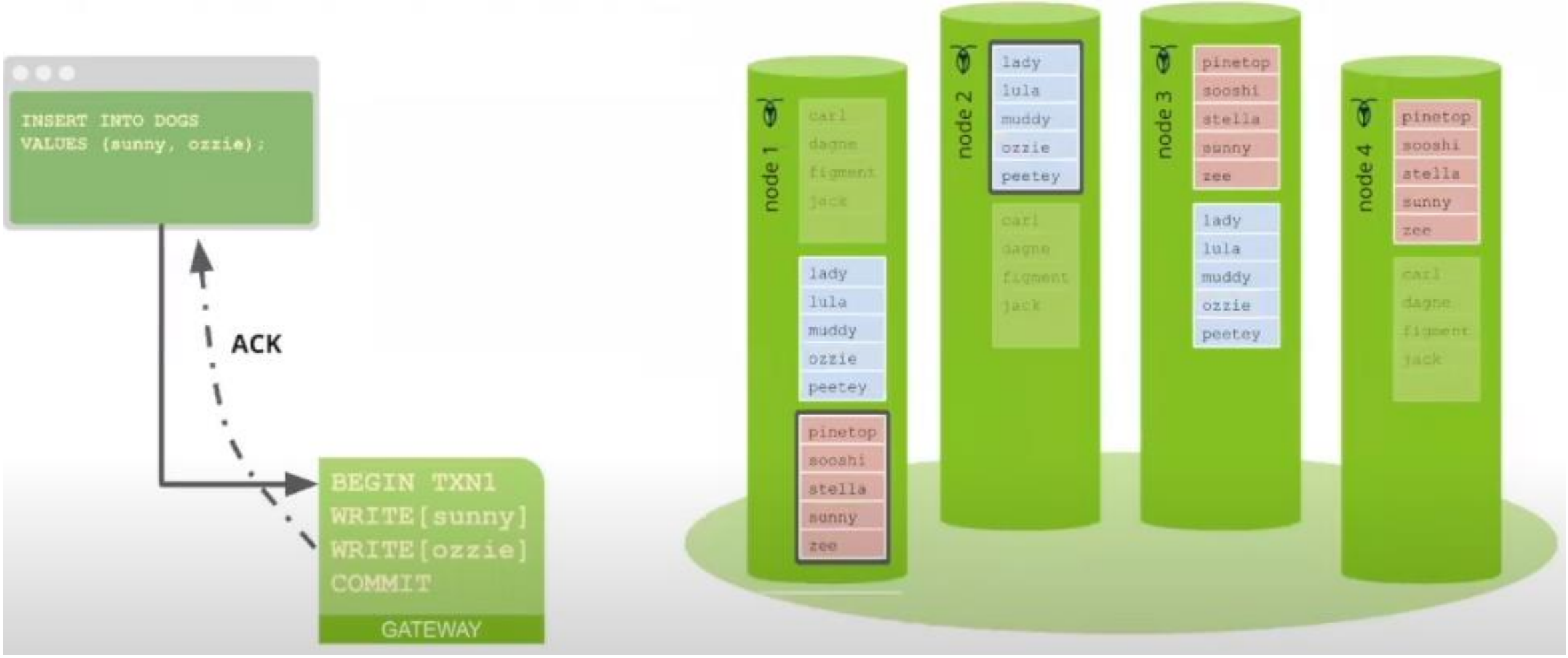
- Transaktionen -

- Schritt 9 -



- Transaktionen -

- Schritt 10 -



Beispiel logischer Plan

. DistSQL -

SQL Input

```
TABLE Orders (OId INT PRIMARY KEY, CId INT, Value DECIMAL, Date DATE)

SELECT CID, SUM(VALUE) FROM Orders
  WHERE DATE > 2015
  GROUP BY CID
  ORDER BY 1 - SUM(Value)
```

Logischer Plan Stufe 1 (ohne „ORDER BY“)

```
TABLE-READER src
  Table: Orders
  Table schema: Oid:INT, Cid:INT, Value:DECIMAL, Date:DATE
  Output filter: (Date > 2015)
  Output schema: Cid:INT, Value:DECIMAL
  Ordering guarantee: Oid

AGGREGATOR summer
  Input schema: Cid:INT, Value:DECIMAL
  Output schema: Cid:INT, ValueSum:DECIMAL
  Group Key: Cid
  Ordering characterization: if input ordered by Cid, output ordered by Cid

EVALUATOR sortval
  Input schema: Cid:INT, ValueSum:DECIMAL
  Output schema: SortVal:DECIMAL, Cid:INT, ValueSum:DECIMAL
  Ordering characterization:
    ValueSum -> ValueSum and -SortVal
    Cid,ValueSum -> Cid,ValueSum and Cid,-SortVal
    ValueSum,Cid -> ValueSum,Cid and -SortVal,Cid
  SQL Expressions: E(x:INT) INT = (1 - x)
  Code {
    EMIT E(ValueSum), CId, ValueSum
  }

AGGREGATOR final:
  Input schema: SortVal:DECIMAL, Cid:INT, ValueSum:DECIMAL
  Input ordering requirement: SortVal
  Group Key: []

Composition: src -> summer -> sortval -> final
```

Logischer Plan Stufe 2 (mit „ORDER BY“)

```
src -> summer -> sortval -> sort(OrderSum) -> final
```

Beispiel physikalischer Plan

. DistSQL -

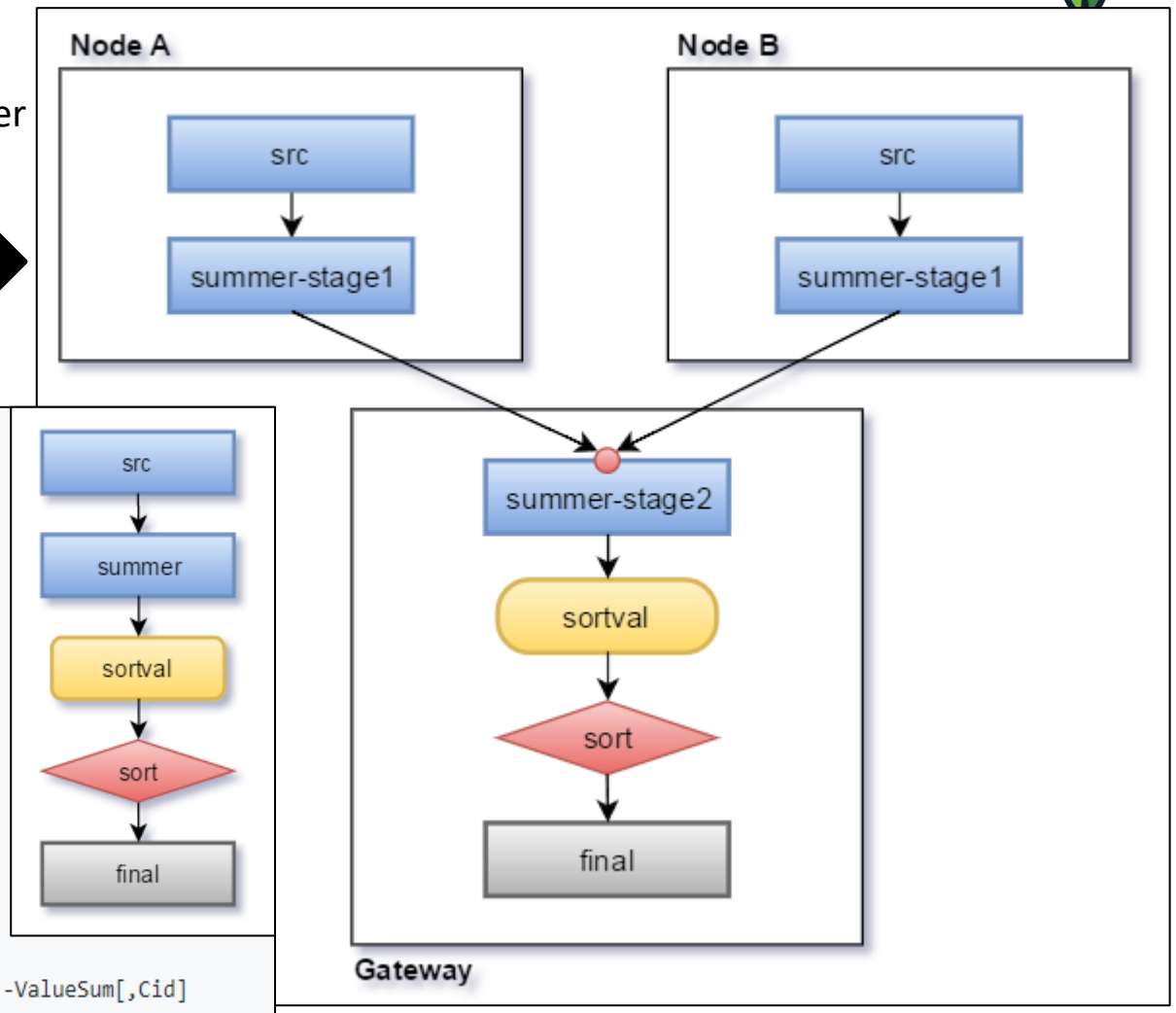
Umsetzung unter Berücksichtig der internen Einschränkungen
- Leaseholder
- Latenz
- Geo-Location
- Auslastung etc.

Logischer Plan

```
TABLE-READER src
  Table: Orders
  Table schema: Oid:INT, Cid:INT, Value:DECIMAL, Date:DATE
  Output filter: (Date > 2015)
  Output schema: Cid:INT, Value:DECIMAL
  Ordering guarantee: Oid

AGGREGATOR summer
  Input schema: Cid:INT, Value:DECIMAL
  Output schema: Cid:INT, ValueSum:DECIMAL
  Group Key: Cid
  Ordering characterization: if input ordered by Cid, output ordered by Cid

EVALUATOR sortval
  Input schema: Cid:INT, ValueSum:DECIMAL
  Output schema: SortVal:DECIMAL, Cid:INT, ValueSum:DECIMAL
  Ordering characterization: if input ordered by [Cid,]ValueSum[,Cid], output ordered by [Cid,]-ValueSum[,Cid]
  SQL Expressions: E(x:INT) INT = (1 - x)
  Code {
    EMIT E(ValueSum), CId, ValueSum
  }
```



Physikalischer Plan

Inhalte

- Die Inhalte stammen aus der offiziellen CockroachDB Dokumentation, begleitenden Vorträgen oder der „Get Started“ der Cockroach Universitiy
  - Link CockroachDB Dokumentation CockroachDB_Docs
  - Link CockroachDB Vorträge CockroachDB Webinar,
  - Link CockroachDB University CockroachDB_University
- Die genutzten Grafiken stammen aus den gleichen Quellen und werden daher nicht extra aufgeführt
- Die genutzten Icons Stammen jeweils vom Anbieter oder dritten Quellen
  - PostgreSQL PostgreSQL_Icon
  - Cassandra Cassandra_Icon
  - MySQL MySQL_Icon
  - Raft Raft_Icon
  - Pick2 Pick2_Icon
  - Pick3 Pick3_Icon