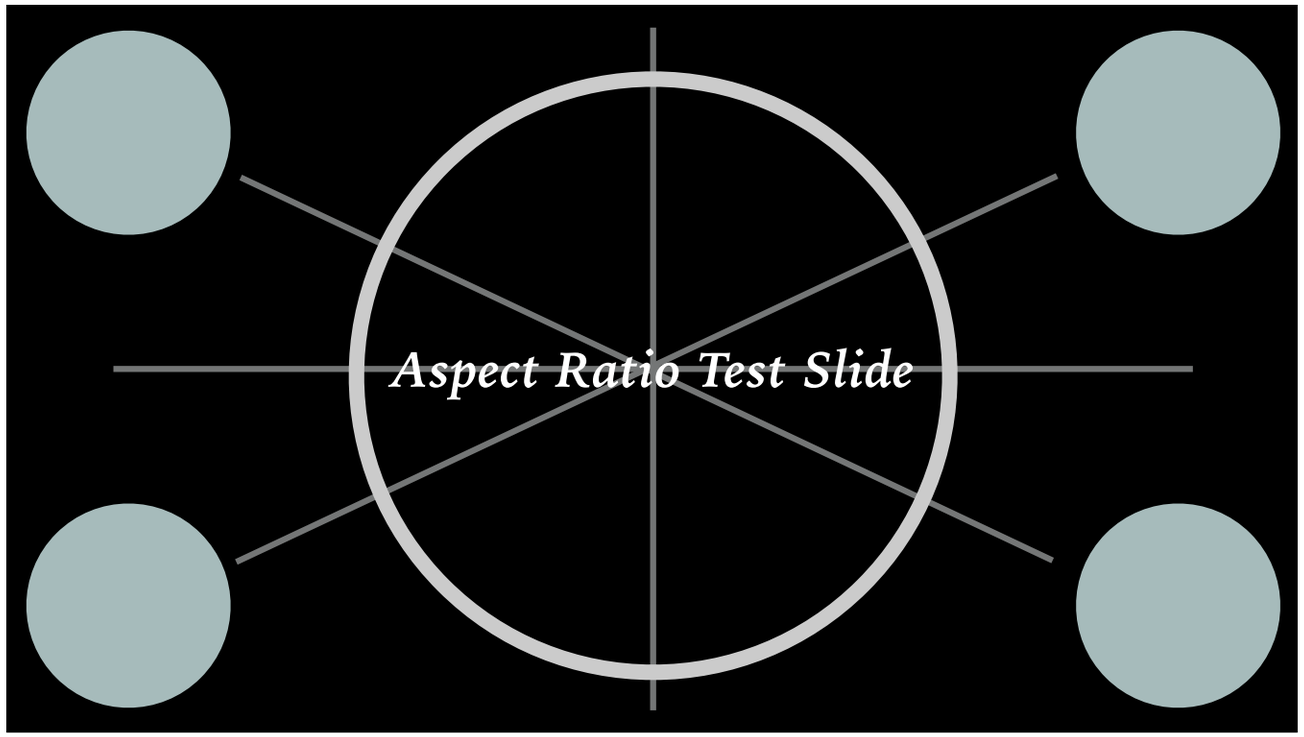

Aspect Ratio Test Slide

# INTRODUCING APACHE COUCHDB 2.0

*by Jan Lehnardt at ApacheCon EU 2016 in Sevilla*

## JAN LEHNARDT

- CouchDB since 2006
- Apache CouchDB since 2008
- PMC Chair & VP of CouchDB since 2011
- Longest active contributor
- CEO at Neighbourhoodie Software in Berlin

Joined CouchDB in 2006, longest standing contributor

Have done everything from evangelising, community work, core engineering.

Still do all of the above

* * *

We shipped 2.0 on Sept. 20th, fulfilling the 10+ year development history of CouchDB

**Seamless** multi-master sync, that scales from **Big Data** to **Mobile,** with an **Intuitive** HTTP/JSON API and designed for **Reliability.**

# THE THREE USE-CASES OF COUCHDB

# 1. GENERAL PURPOSE DATABASE

1. GENERAL PURPOSE DATABASE

2. HIGHLY AVAILABLE & SCALABLE BIG DATA CLUSTER

1. GENERAL PURPOSE DATABASE
2. HIGHLY AVAILABLE & SCALABLE BIG DATA CLUSTER
3. SEAMLESS MOBILE TO CLOUD DATA-SYNCHRONISATION

# MIX AND MATCH ANY ONE

MIX AND MATCH ANY ONE

# DATA SYNCHRONISATION BETWEEN ALL USE-CASES

# MOBILE TO CLOUD DATA SYNC

# MOST MOBILE DATA IS OFFLINE

*for battery power reasons*

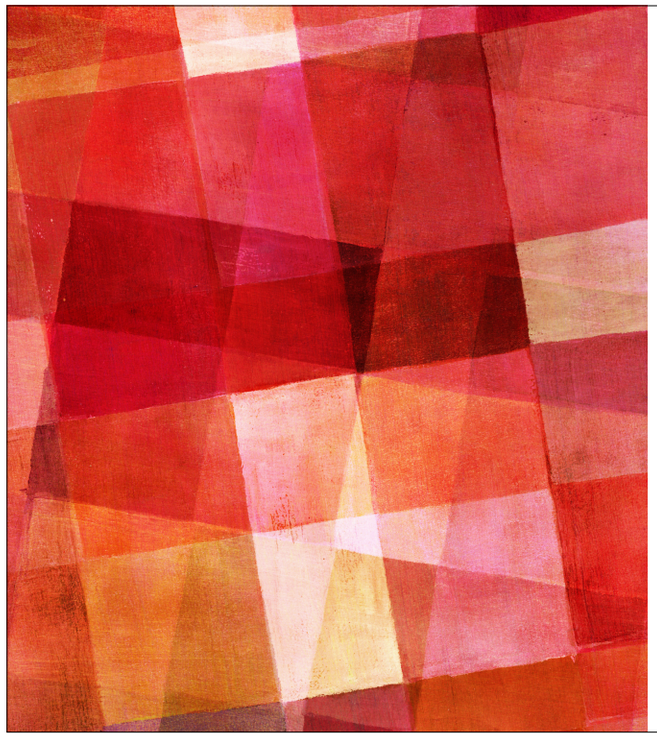
# ALMOST 60% OF MOBILE IS ON 2G

*Google Chrome Dev Summit last week*

CouchDB helps you to build compelling applications in the face of spotty networks.

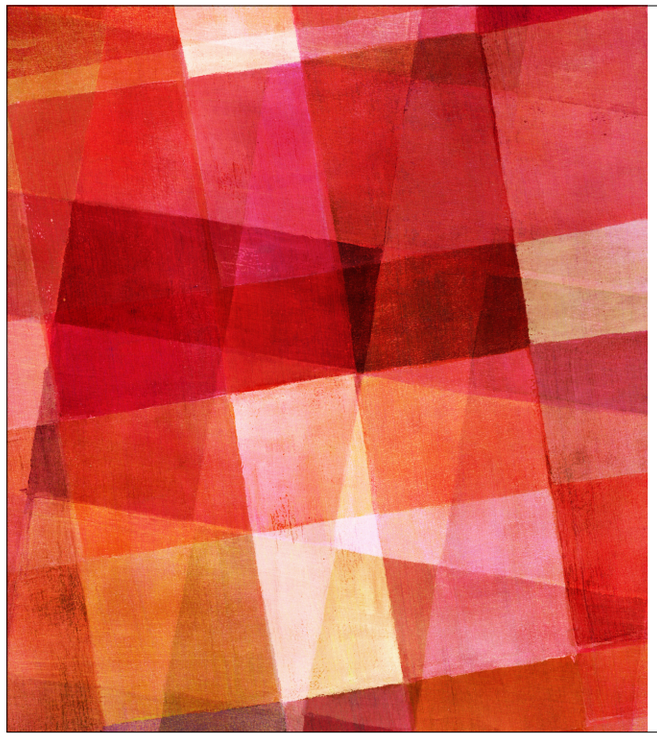
CouchDB helps you to bring mobile data into the Cloud for Big Data analysis.

# GENERAL PURPOSE DATABASE



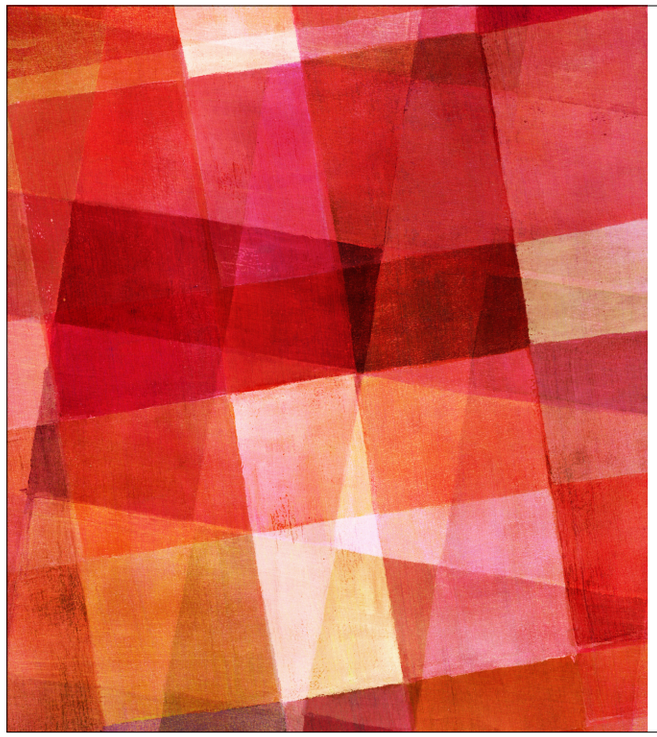
# BASICS

JSON cuts ORM



## BASICS

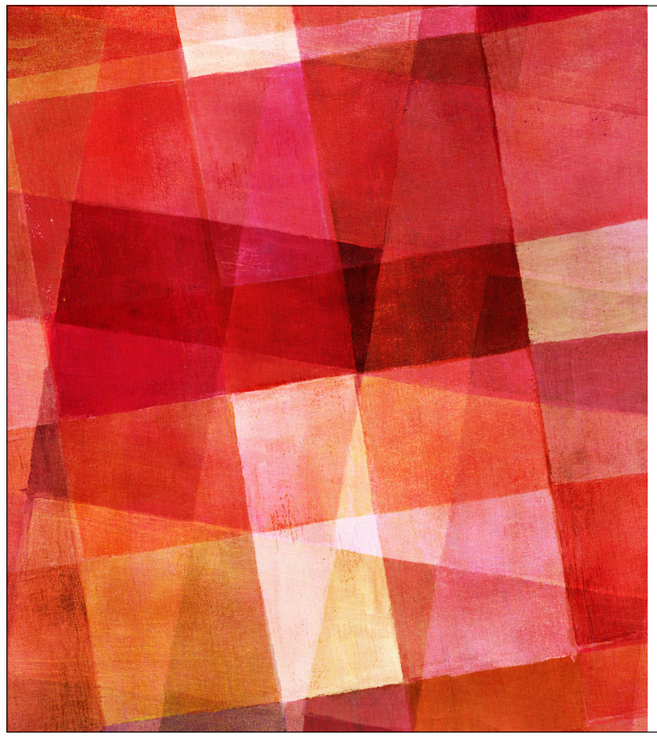
- HTTP



## BASICS

- HTTP
- JSON

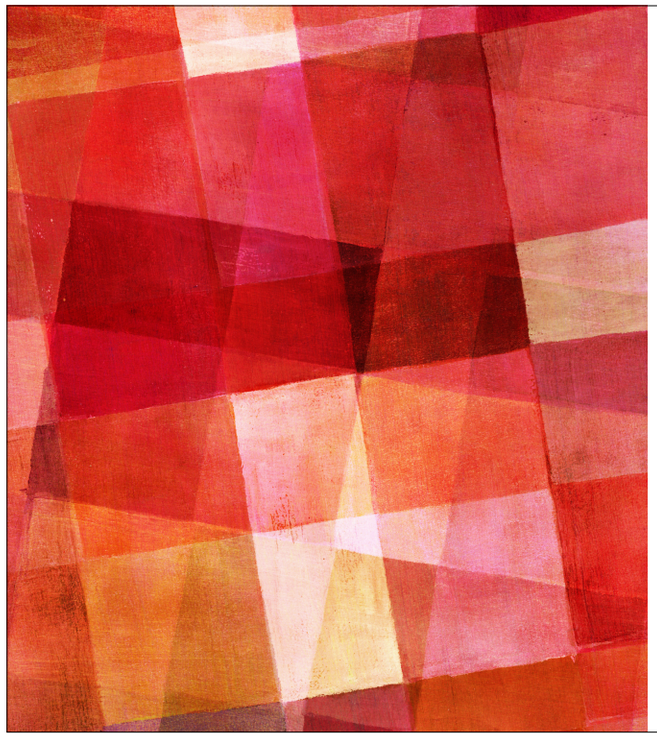
JSON cuts ORM



## BASICS

- HTTP
- JSON
  - Documents

JSON cuts ORM



## BASICS

- HTTP
- JSON
  - Documents
  - Unique IDs, content addressable revisions

JSON cuts ORM

## BASICS

MR: unique

API compatible
- design from 10 years ago
- other databases have features that start failing unpredictably at scale
- CouchDB doesn’t have those features in the first place

## BASICS

➤ Incremental, Persistent Map / Reduce for queries

MR: unique

API compatible
- design from 10 years ago
- other databases have features that start failing unpredictably at scale
- CouchDB doesn’t have those features in the first place

## BASICS

- Incremental, Persistent Map / Reduce for queries
- Changes, “what happened since?”, think `git log` but a **real-time stream** for your database

MR: unique

API compatible
- design from 10 years ago
- other databases have features that start failing unpredictably at scale
- CouchDB doesn’t have those features in the first place

## BASICS

- Incremental, Persistent Map / Reduce for queries
- Changes, “what happened since?”, think `git log` but a **real-time stream** for your database
- API Compatible between single node and cluster, apps can grow without rewrite

MR: unique

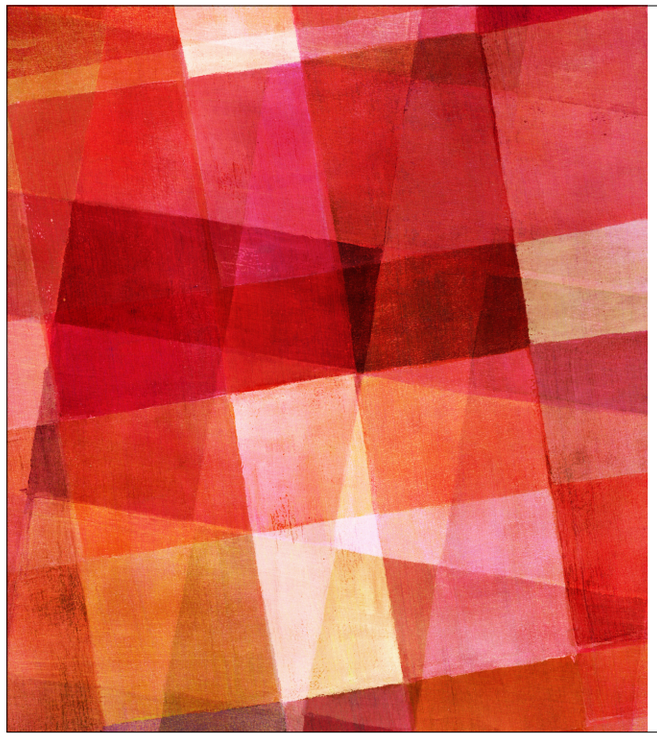
API compatible
- design from 10 years ago
- other databases have features that start failing unpredictably at scale
- CouchDB doesn’t have those features in the first place

## BASICS

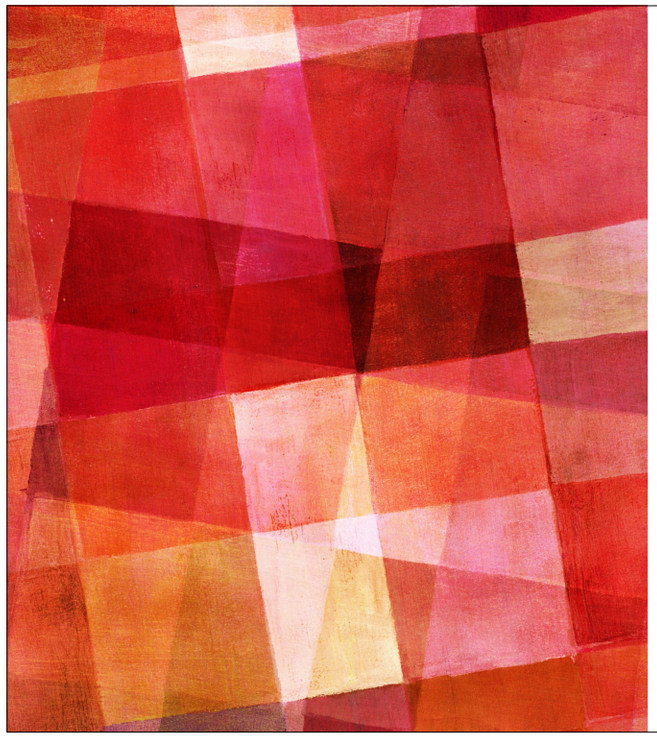
- Incremental, Persistent Map / Reduce for queries
- Changes, “what happened since?”, think `git log` but a **real-time stream** for your database
- API Compatible between single node and cluster, apps can grow without rewrite
  - trade-off: no features that wouldn’t scale in single node version

MR: unique

API compatible
- design from 10 years ago
- other databases have features that start failing unpredictably at scale
- CouchDB doesn’t have those features in the first place

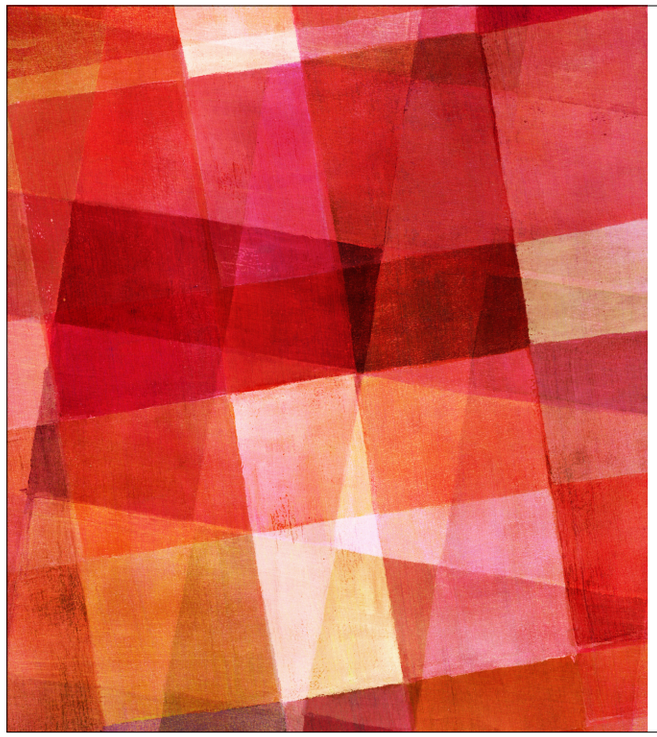


# DESIGN DECISIONS



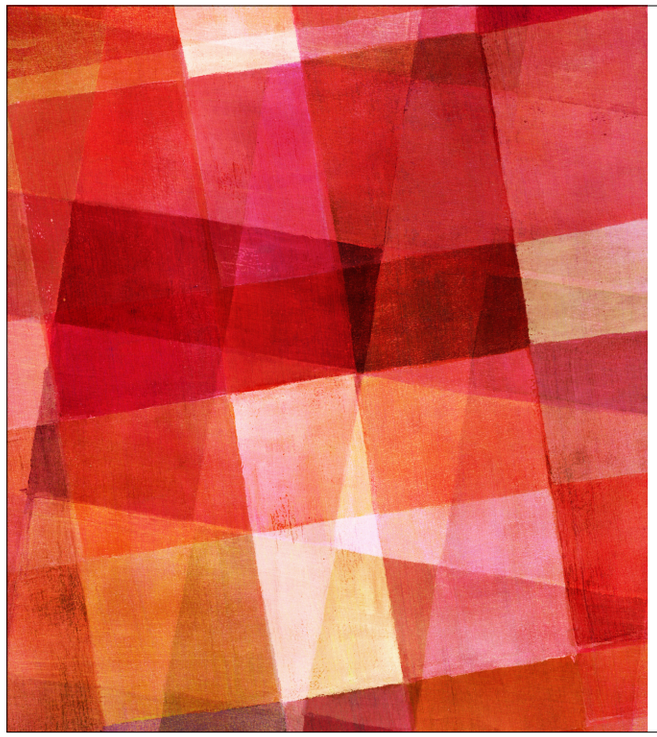
## DESIGN DECISIONS

- Data safety > *



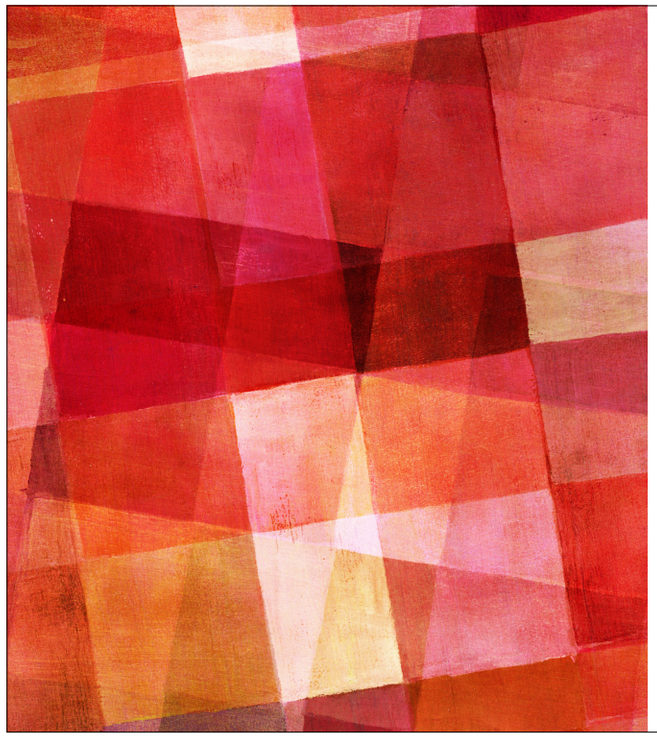
## DESIGN DECISIONS

- Data safety > *
- Fault tolerance



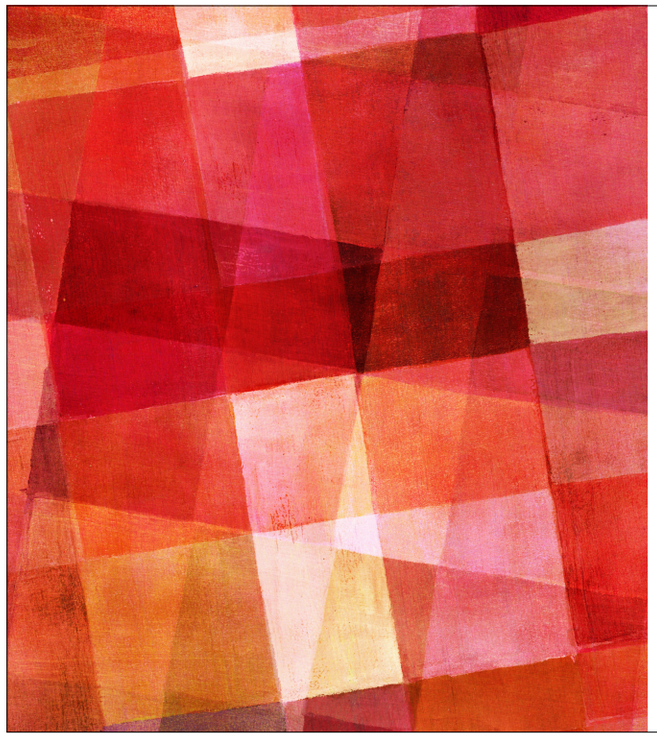
## DESIGN DECISIONS

- Data safety > *
- Fault tolerance
  - Erlang: only one request can fail, not the whole server



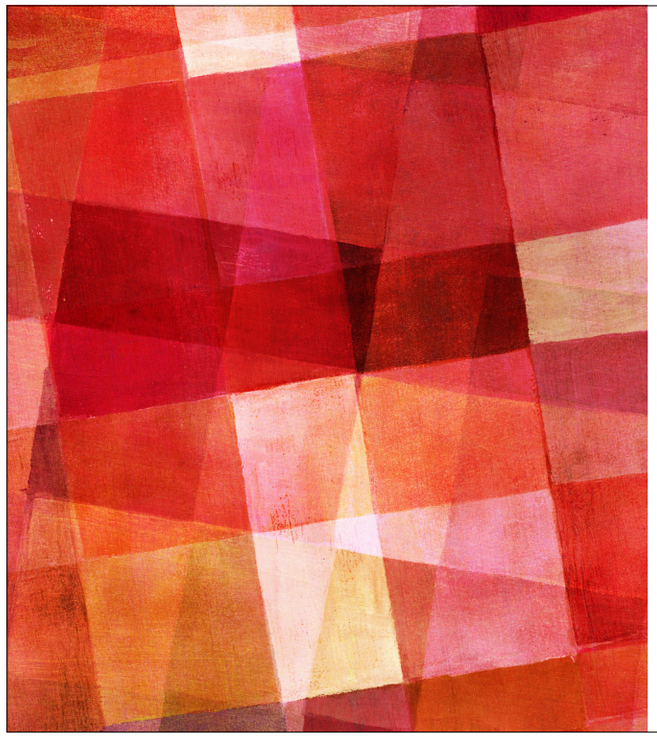
## DESIGN DECISIONS

- Data safety > *
- Fault tolerance
  - Erlang: only one request can fail, not the whole server
  - Crash-only design



## DESIGN DECISIONS

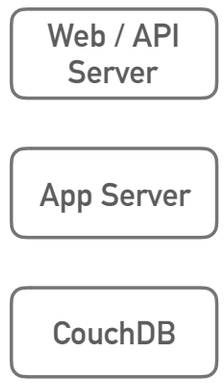
- Data safety > *
- Fault tolerance
  - Erlang: only one request can fail, not the whole server
  - Crash-only design
  - Everything is resumable



## DESIGN DECISIONS

- Data safety > *
- Fault tolerance
  - Erlang: only one request can fail, not the whole server
  - Crash-only design
  - Everything is resumable
  - Everything is idempotent

# GENERAL PURPOSE DATABASE



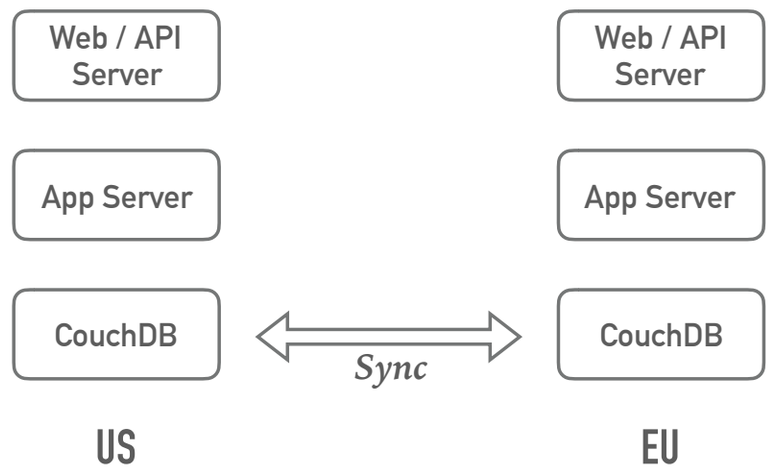
# GENERAL PURPOSE DATABASE

Web / API Server

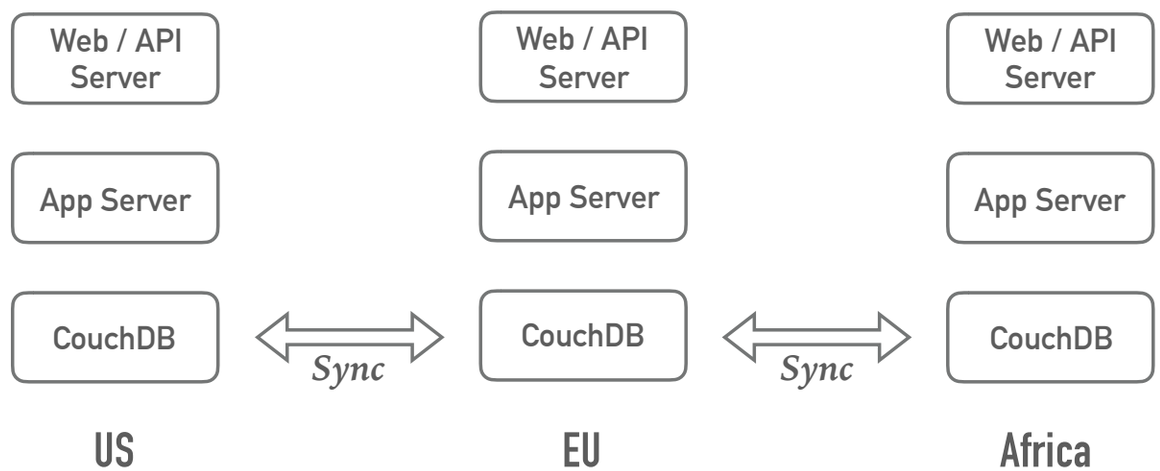
App Server

CouchDB

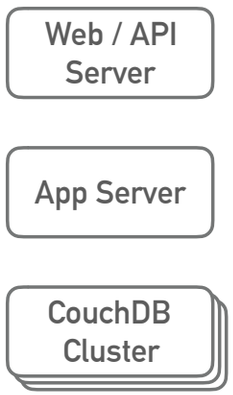
# BONUS 1: GEO DISTRIBUTION



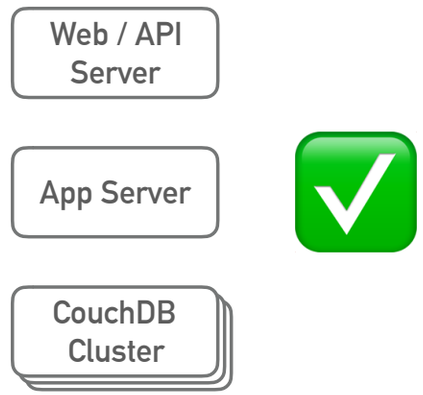
# BONUS: EXTRA


Web / API Server
App Server
CouchDB
Sync
US
Web / API Server
App Server
CouchDB
Sync
EU
Web / API Server
App Server
CouchDB
Africa

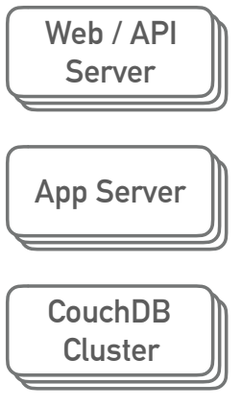
# SCALABLE CLUSTER



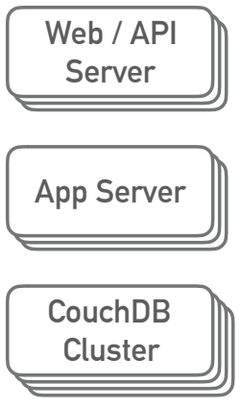
# SCALABLE CLUSTER



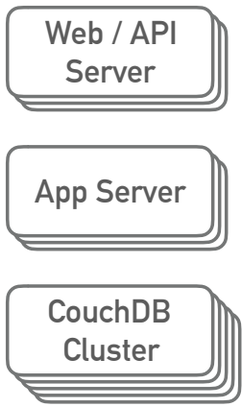
# SCALABLE CLUSTER



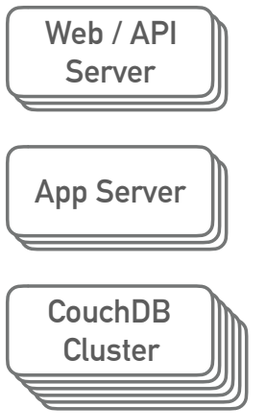
# SCALABLE CLUSTER



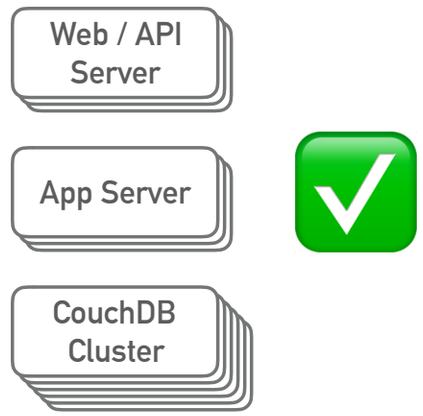
# SCALABLE CLUSTER



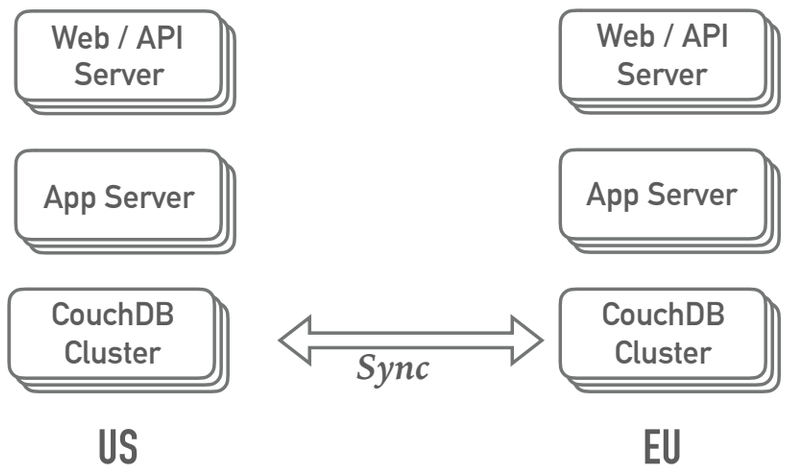
# SCALABLE CLUSTER



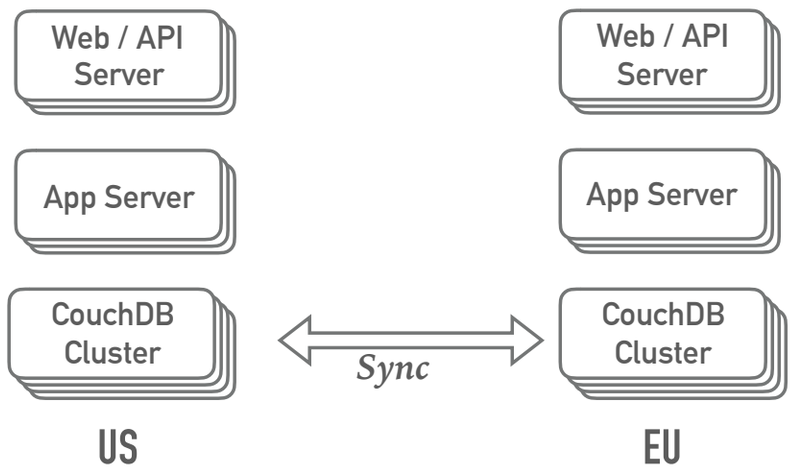
# SCALABLE CLUSTER



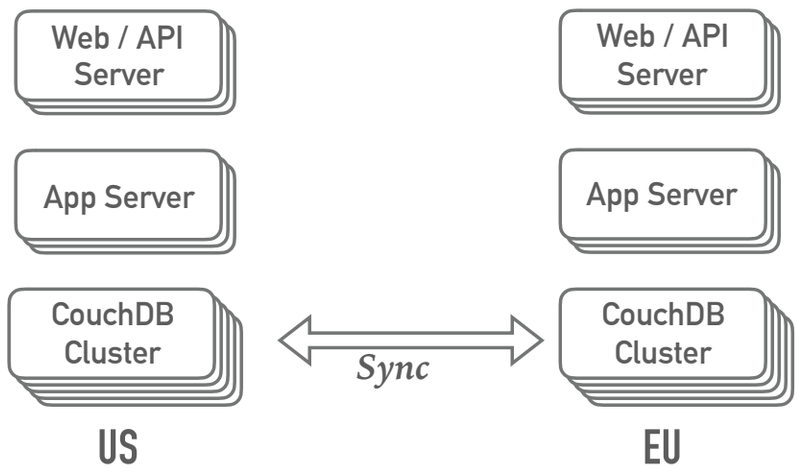
# BONUS STILL THERE



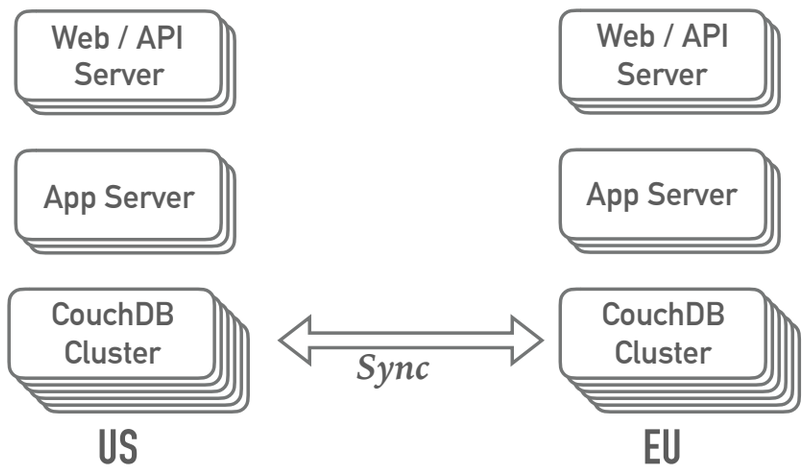
# BONUS STILL THERE



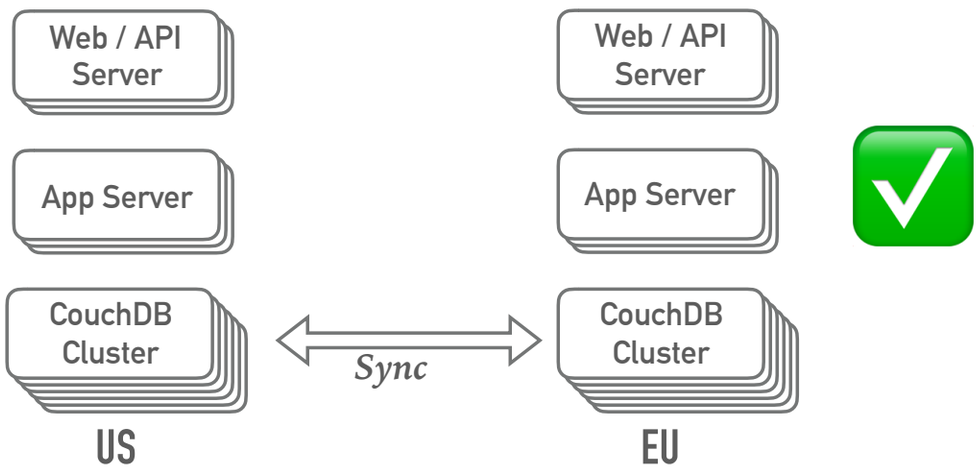
# BONUS STILL THERE

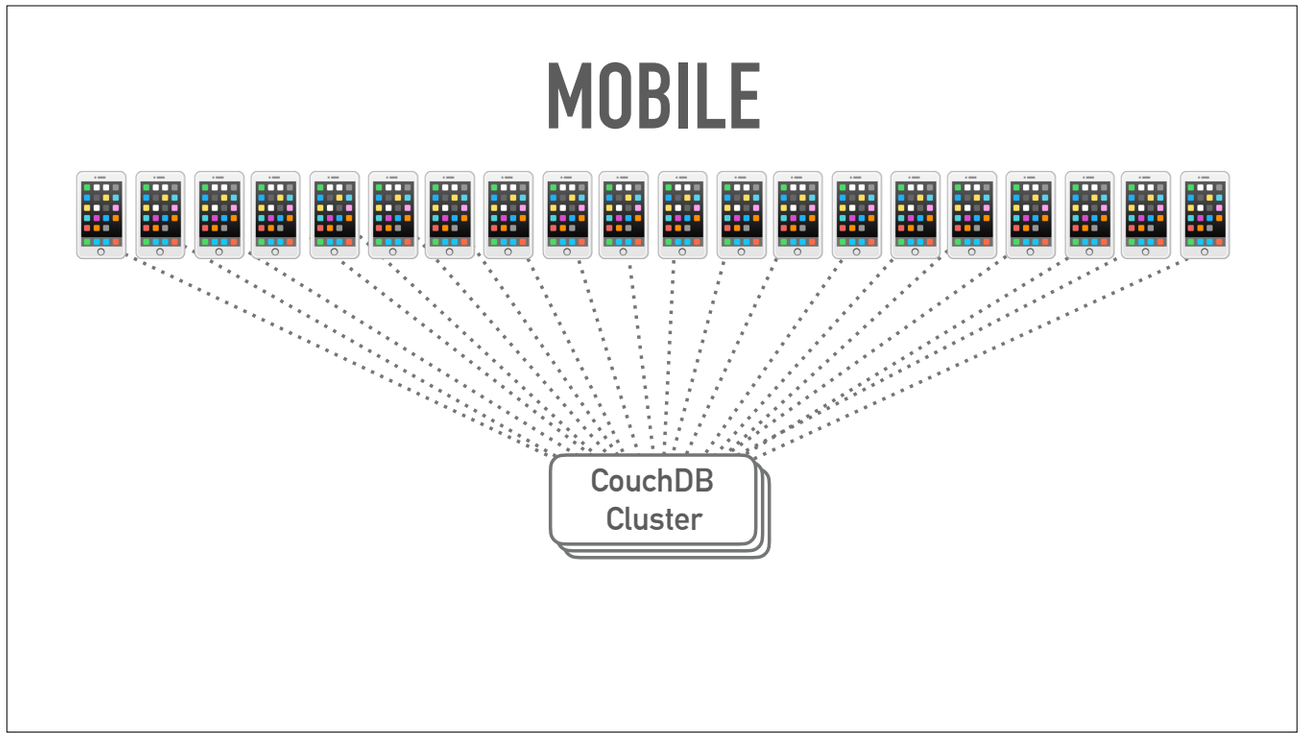


# BONUS STILL THERE

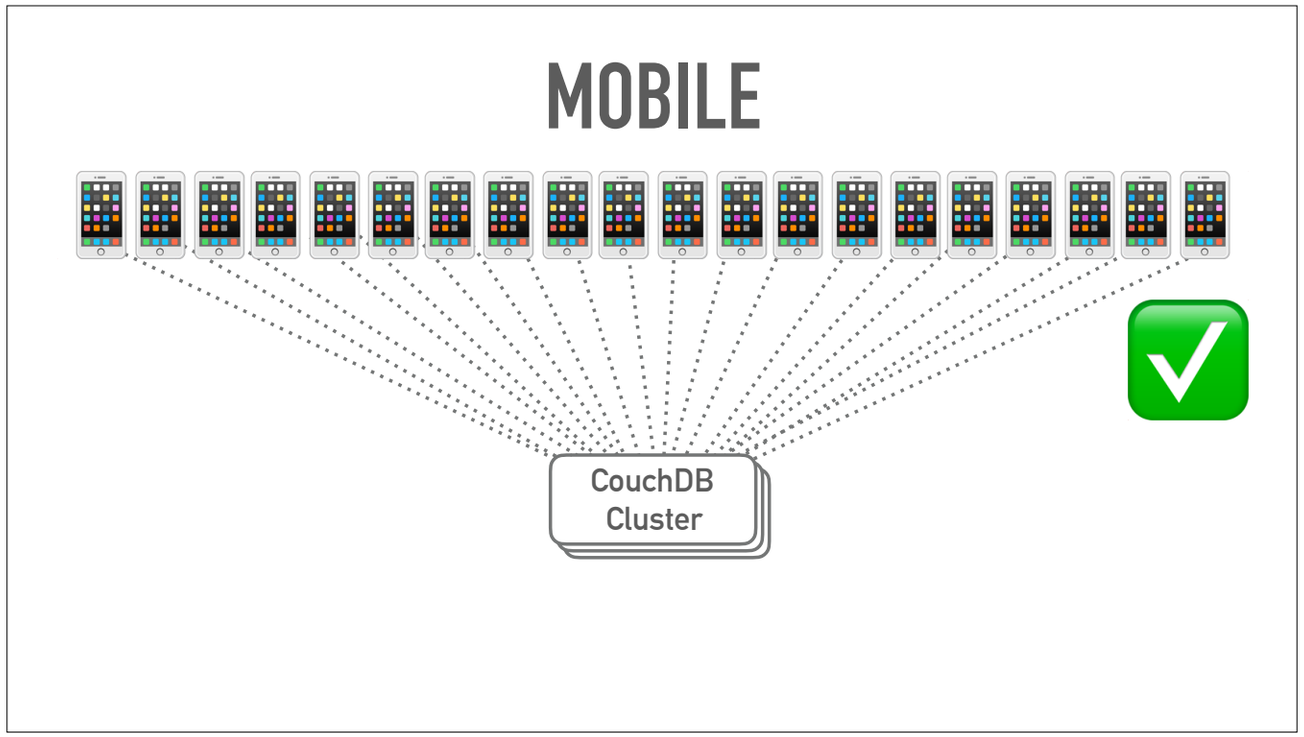


# BONUS STILL THERE



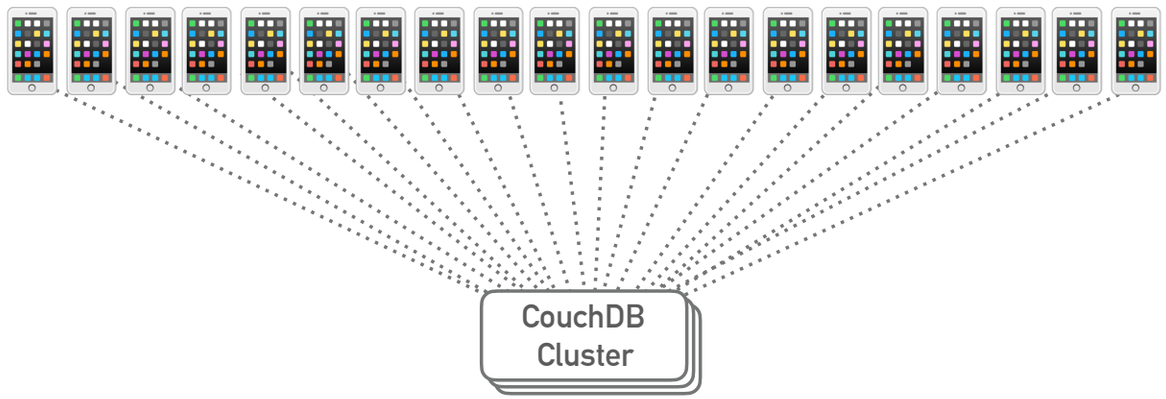



we started offline first



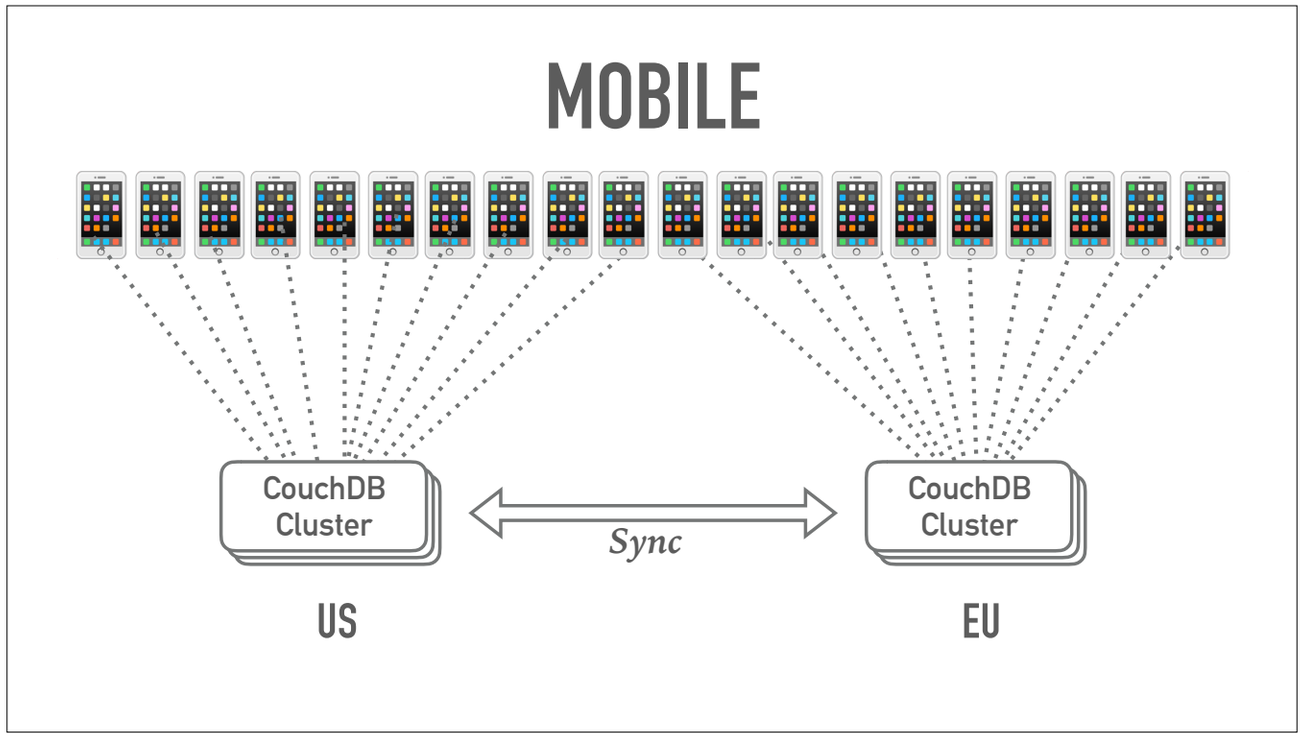

we started offline first

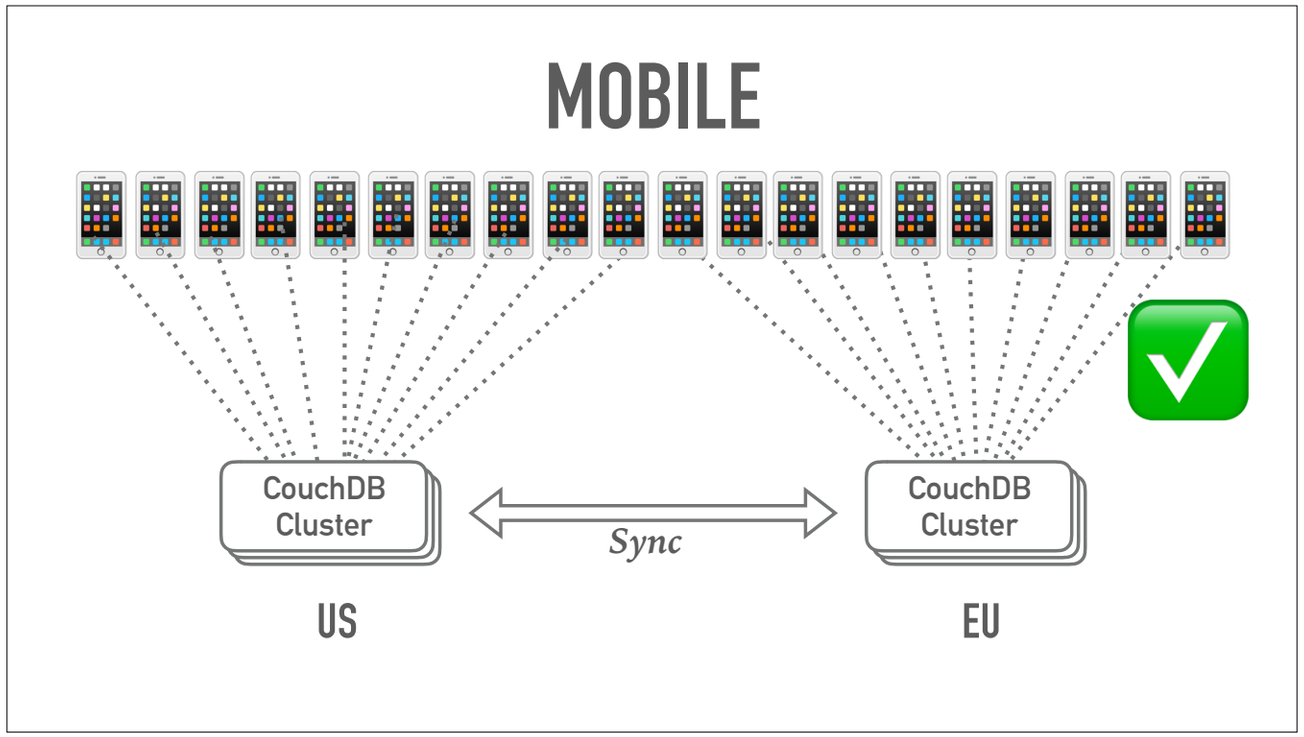
# BONUS

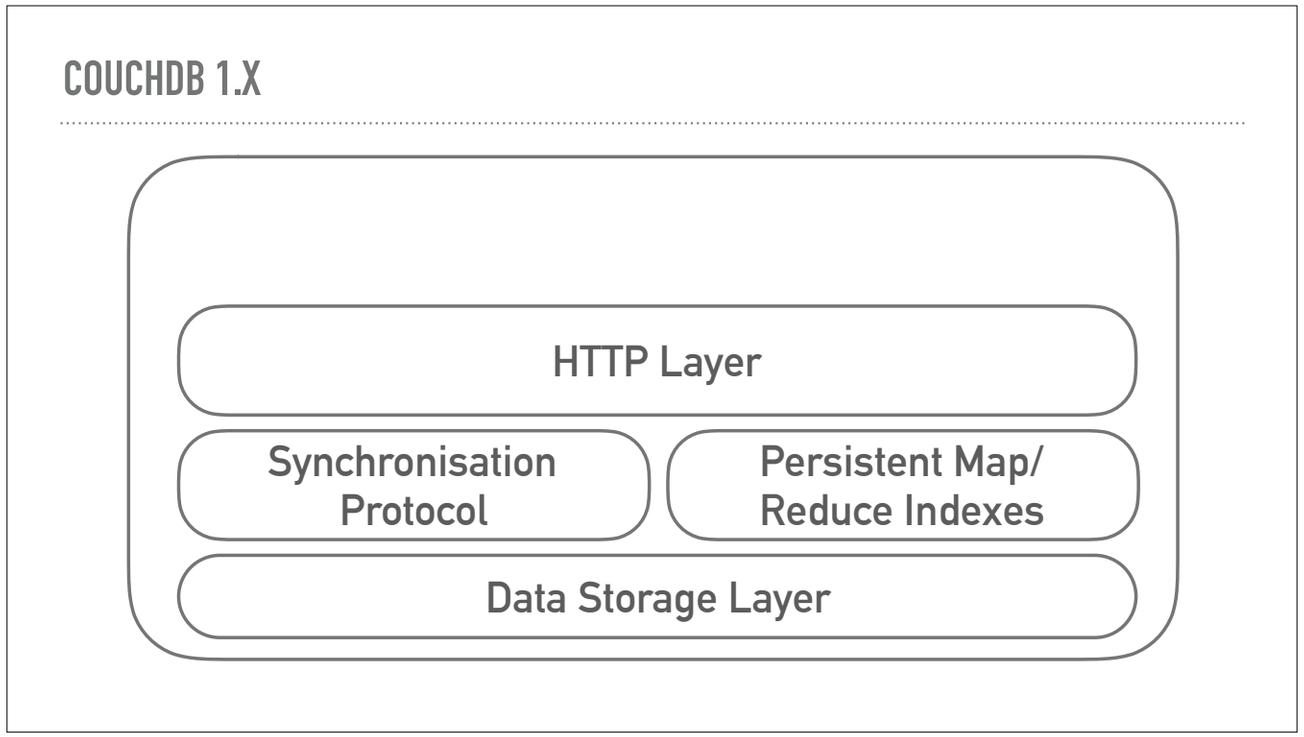


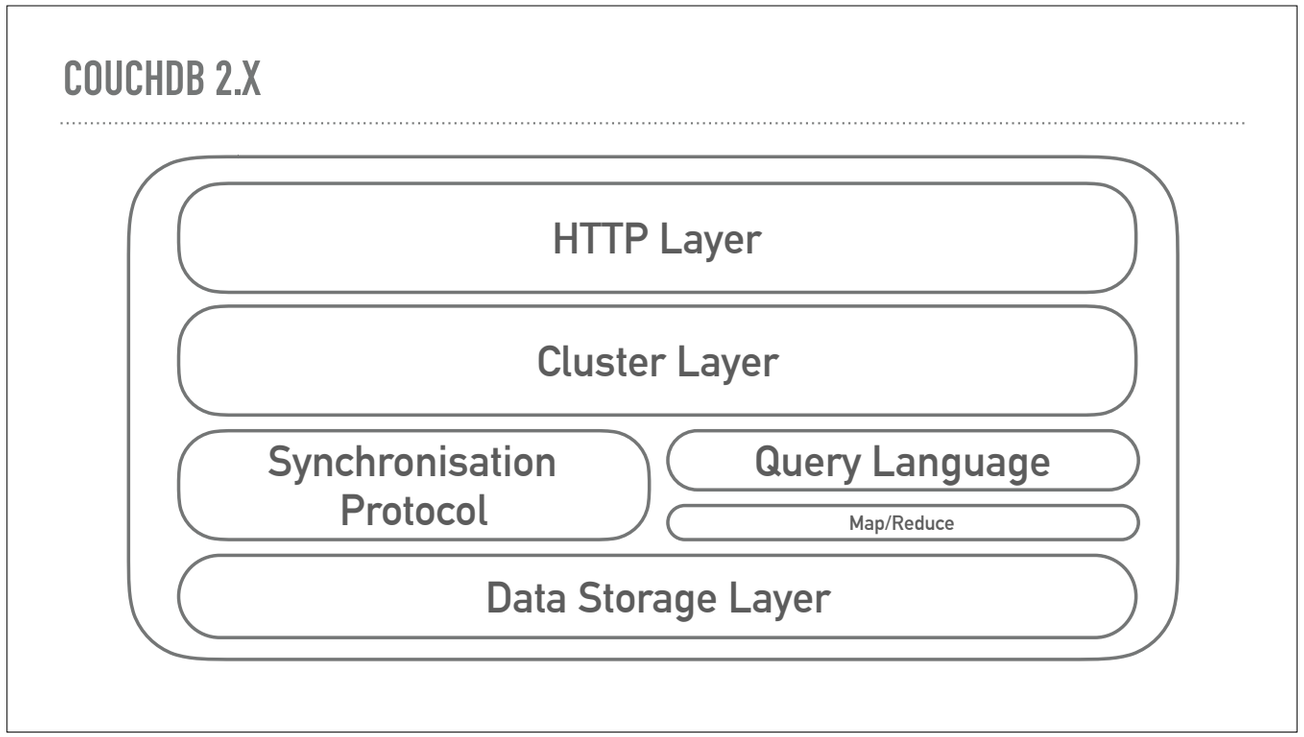
# BONUS

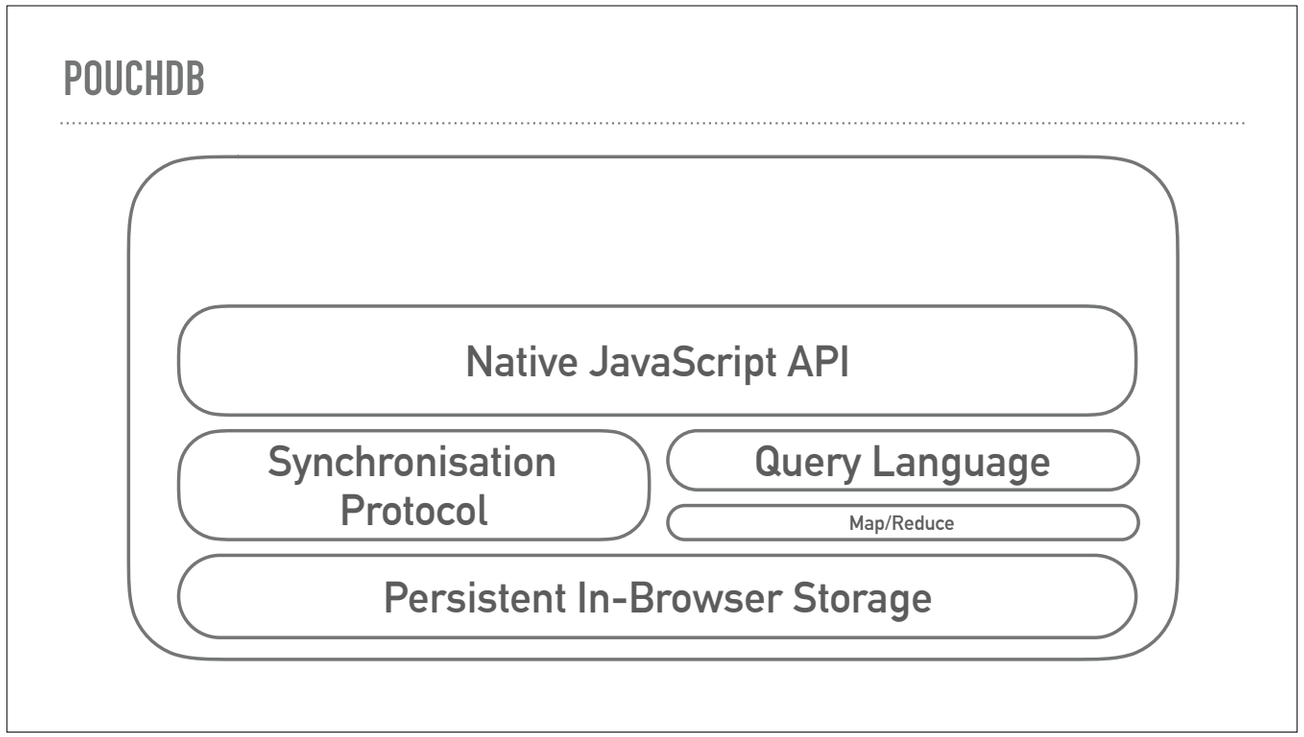
CouchDB
Cluster

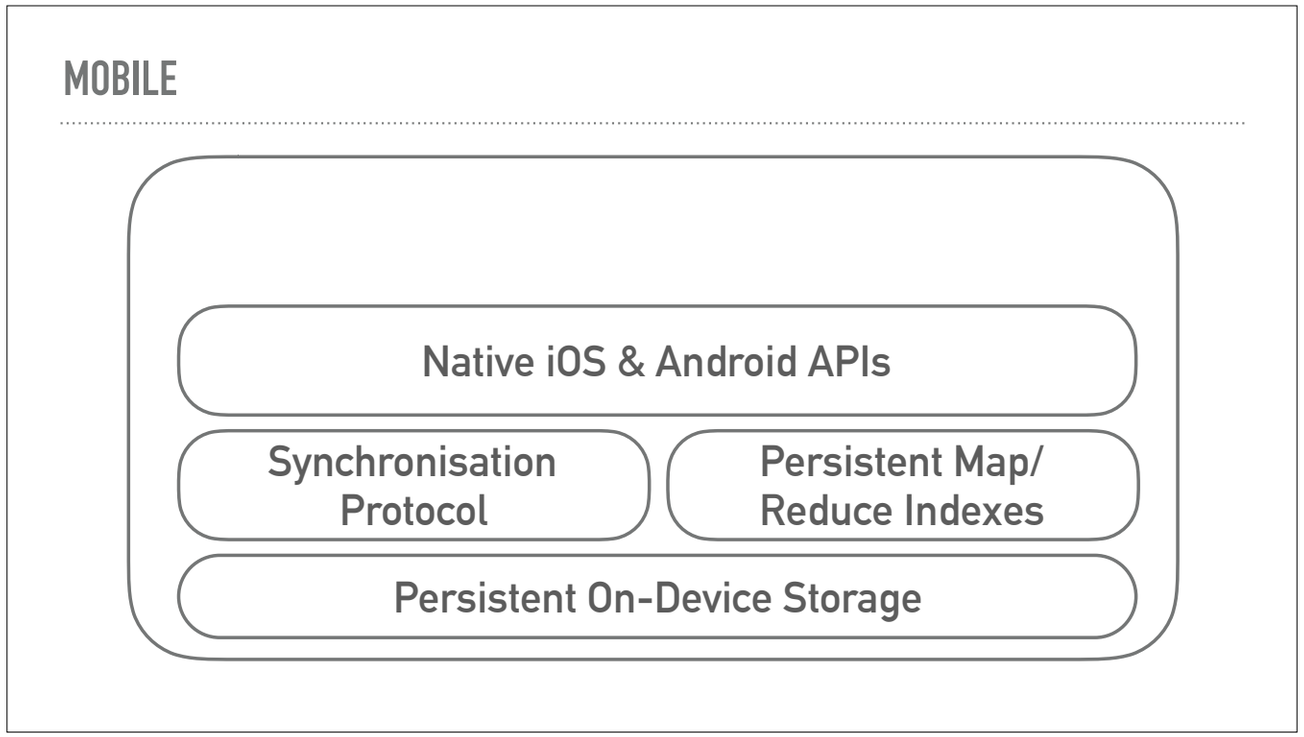

MOBILE
CouchDB Cluster
Sync
CouchDB Cluster
US
EU


MOBILE
CouchDB Cluster
Sync
CouchDB Cluster
US
EU

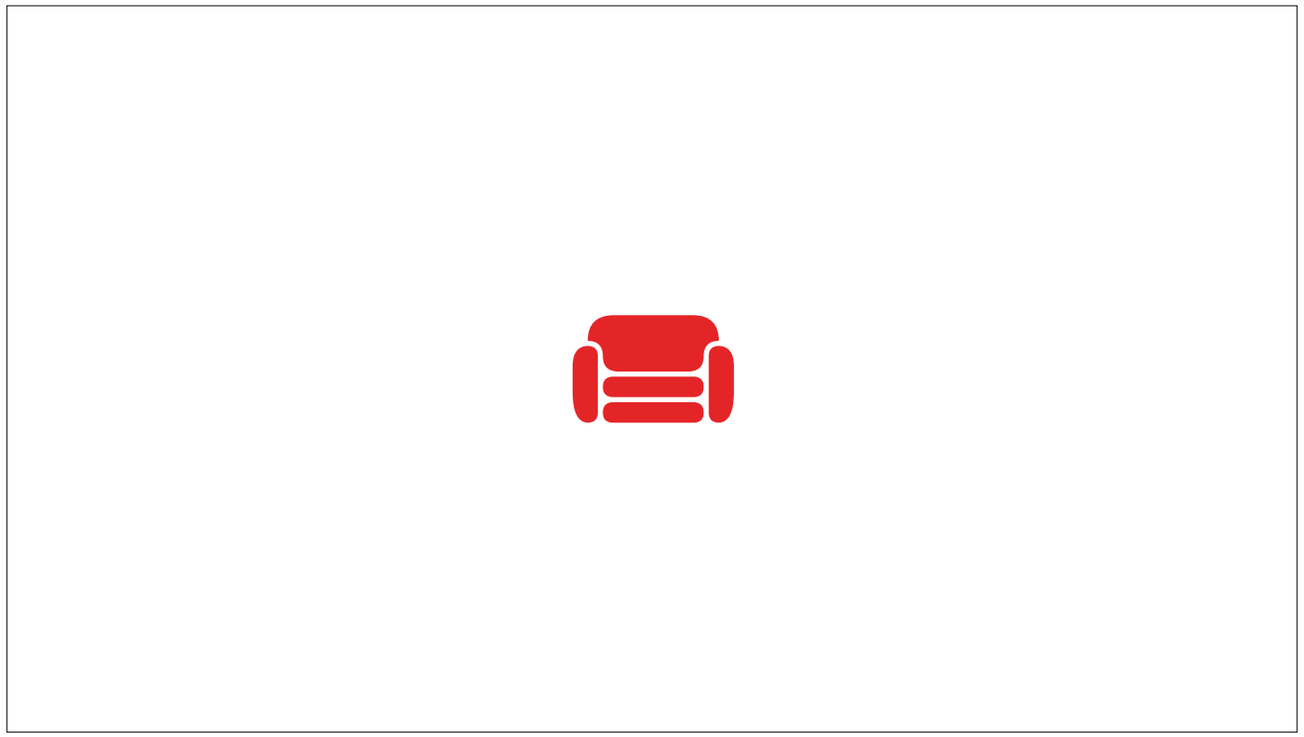

COUCHDB 1.X
HTTP Layer
Synchronisation Protocol
Persistent Map/ Reduce Indexes
Data Storage Layer


COUCHDB 2.X
HTTP Layer
Cluster Layer
Synchronisation Protocol
Query Language
Map/Reduce
Data Storage Layer


POUCHDB
Native JavaScript API
Synchronisation Protocol
Query Language
Map/Reduce
Persistent In-Browser Storage

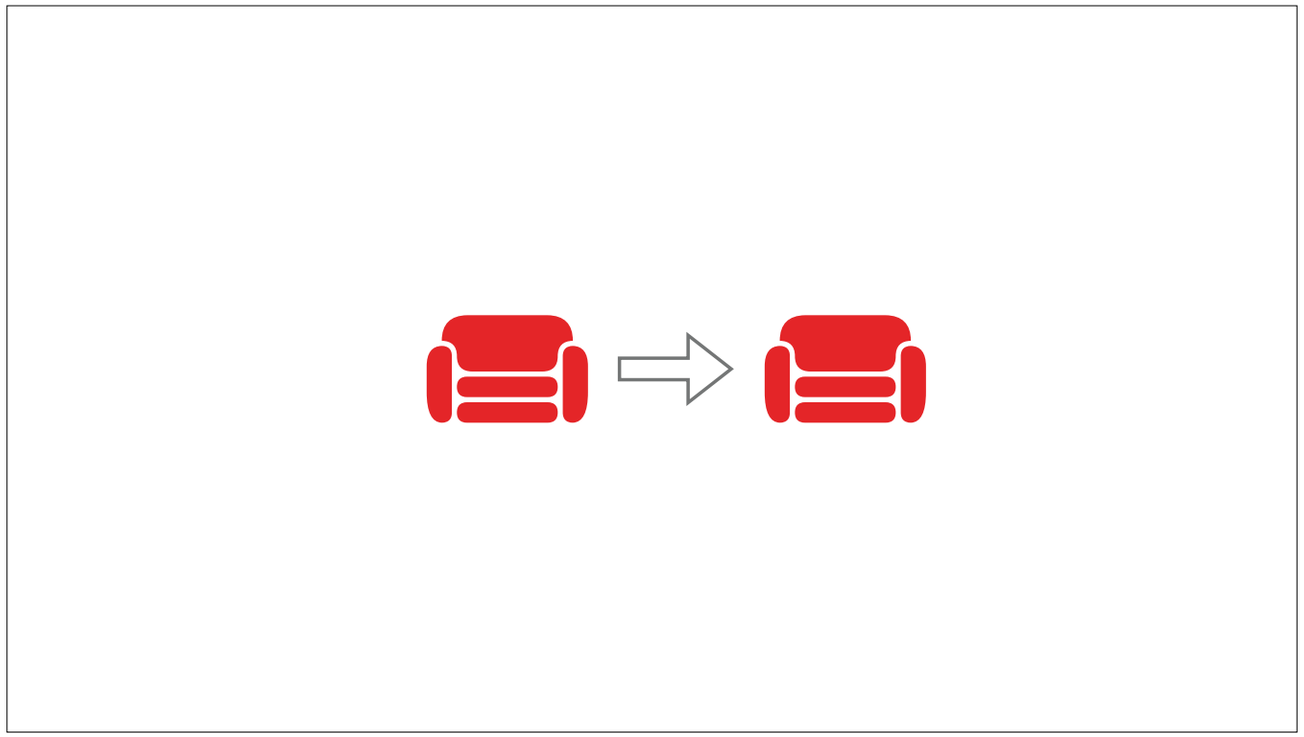

MOBILE
Native iOS & Android APIs
Synchronisation Protocol
Persistent Map/ Reduce Indexes
Persistent On-Device Storage

# TOPOLOGIES



solo

could be single node instance or cluster installation



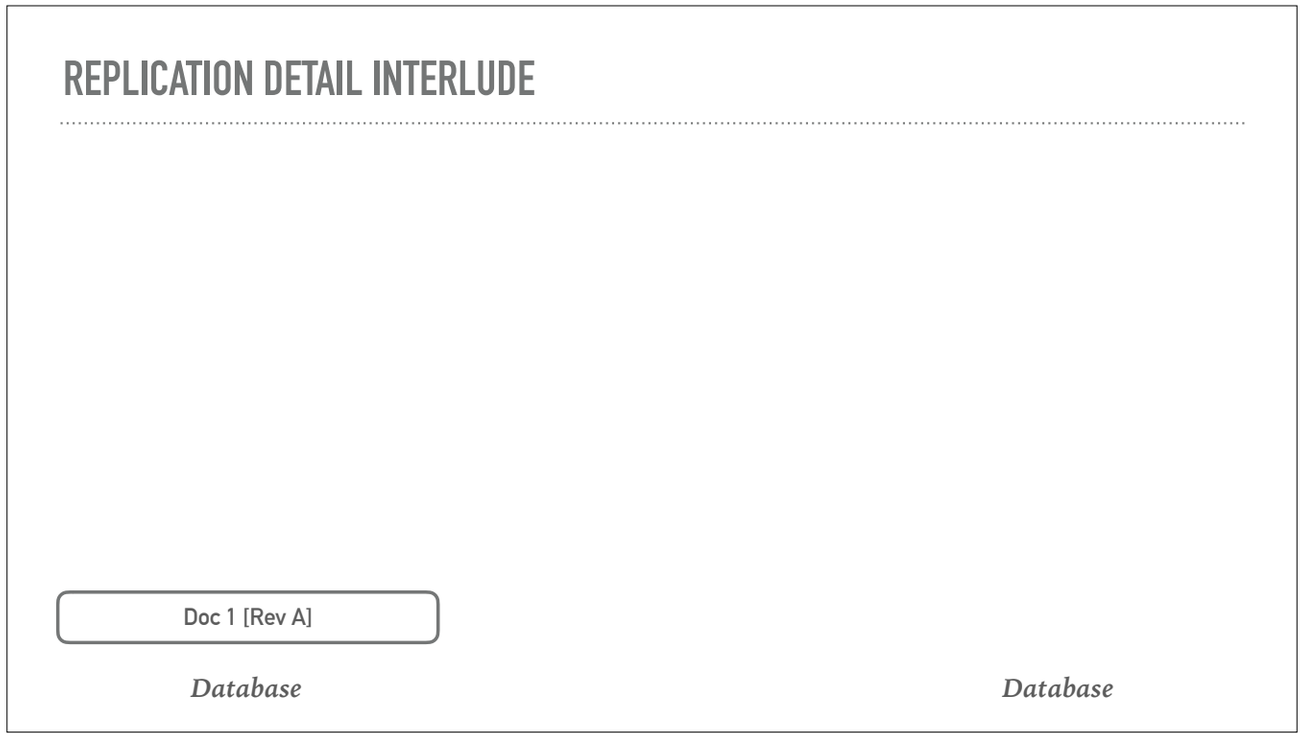
hot spare

explain replication a bit
one way, resume, delta, conflicts

## REPLICATION DETAIL INTERLUDE

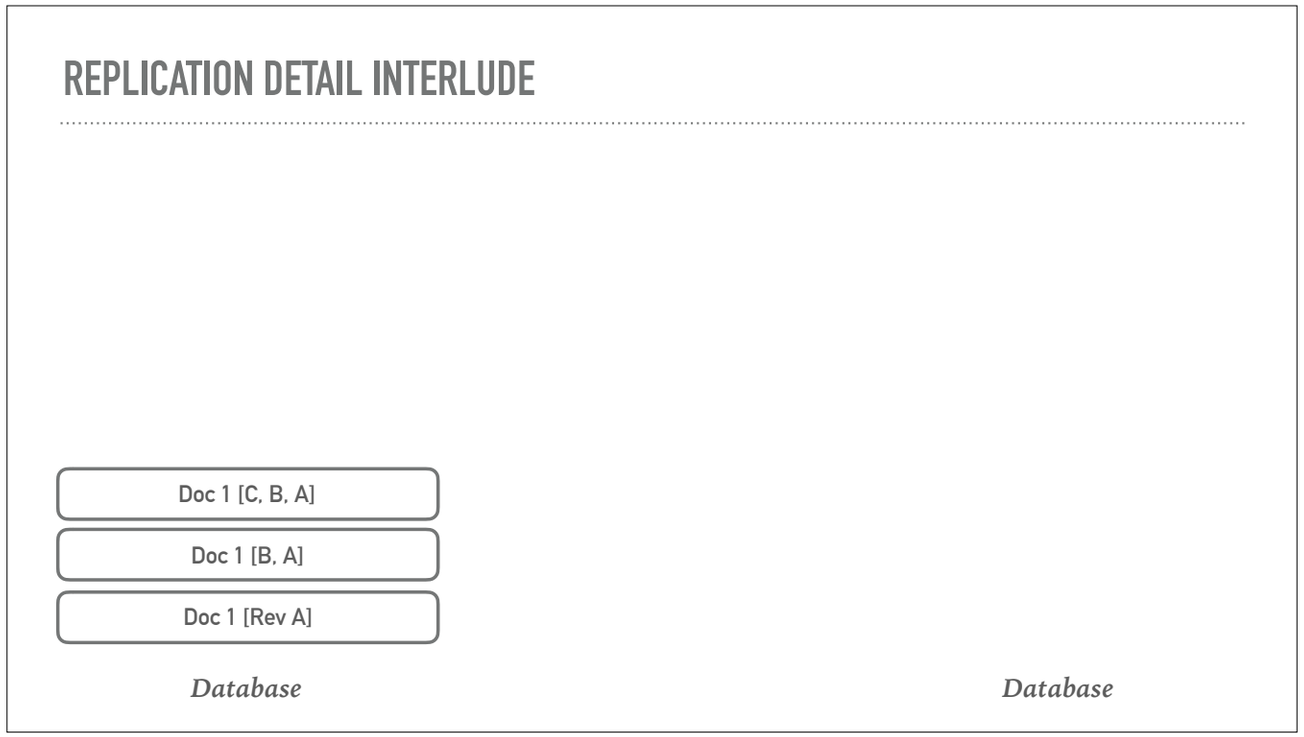
*Database* *Database*

replication details

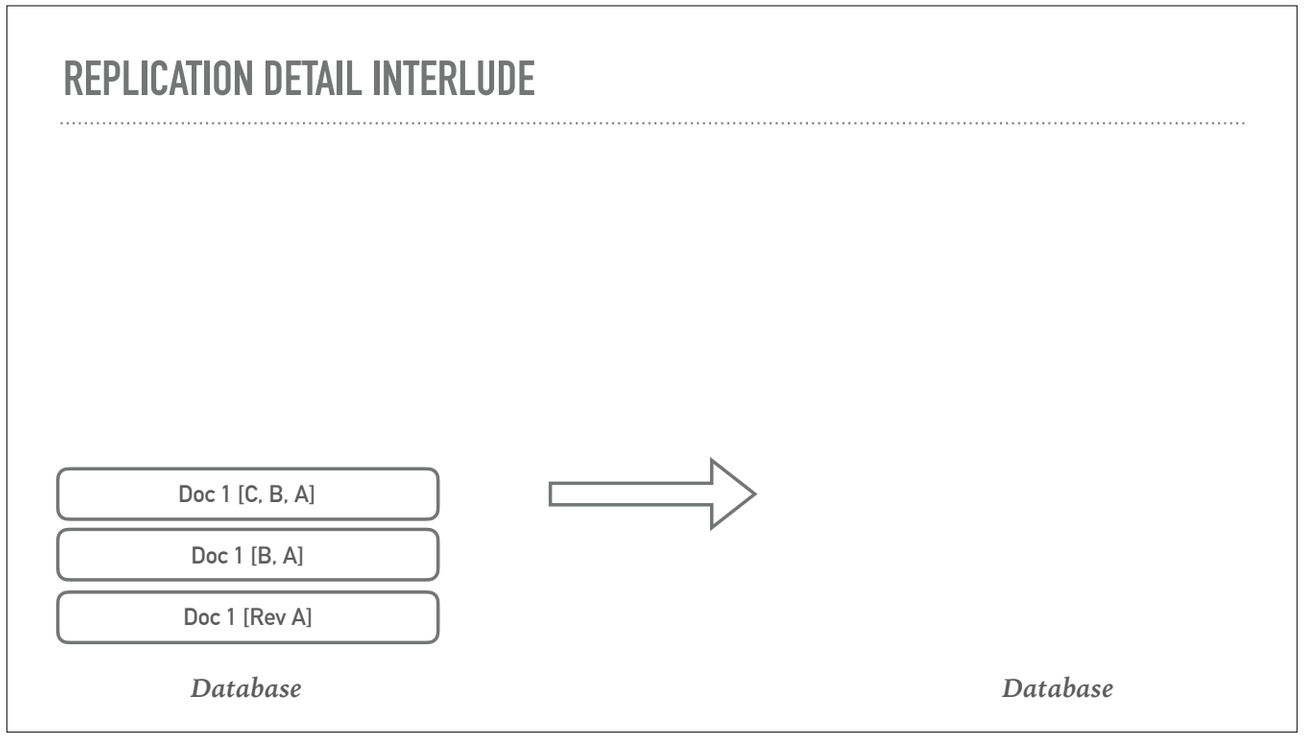



replication details

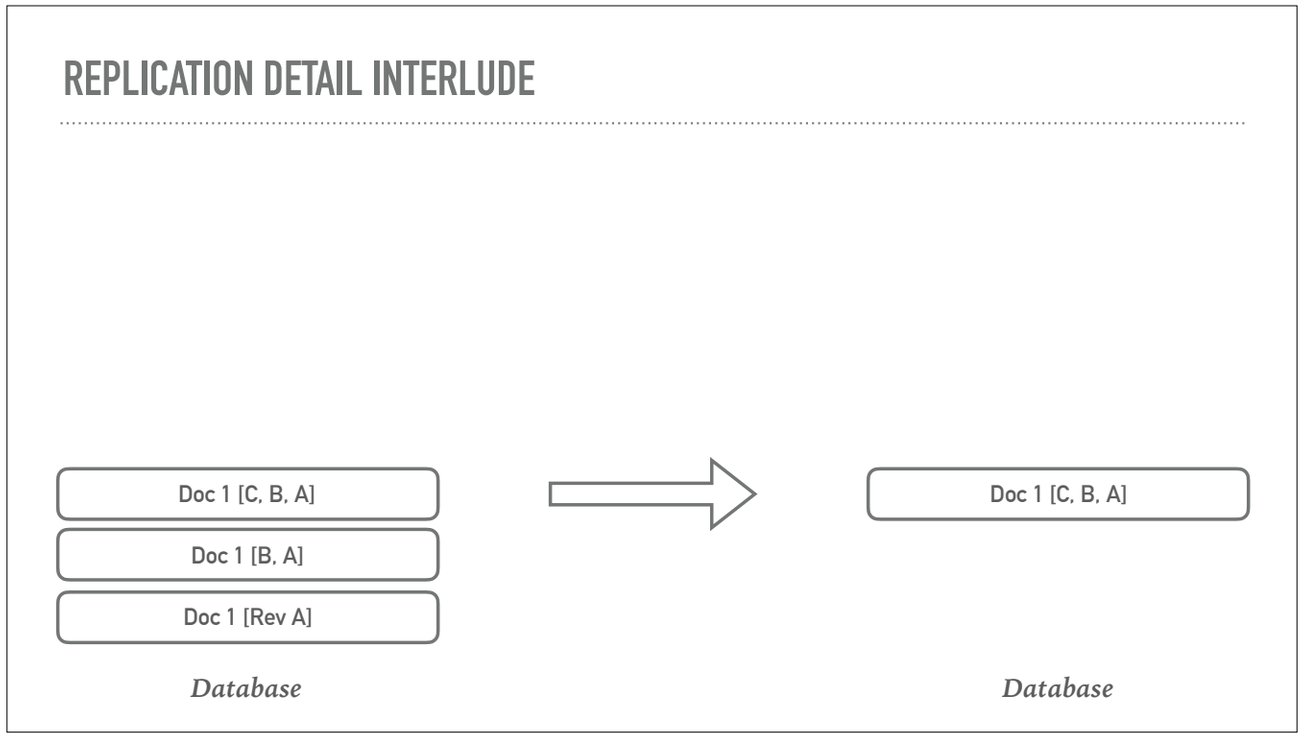

replication details



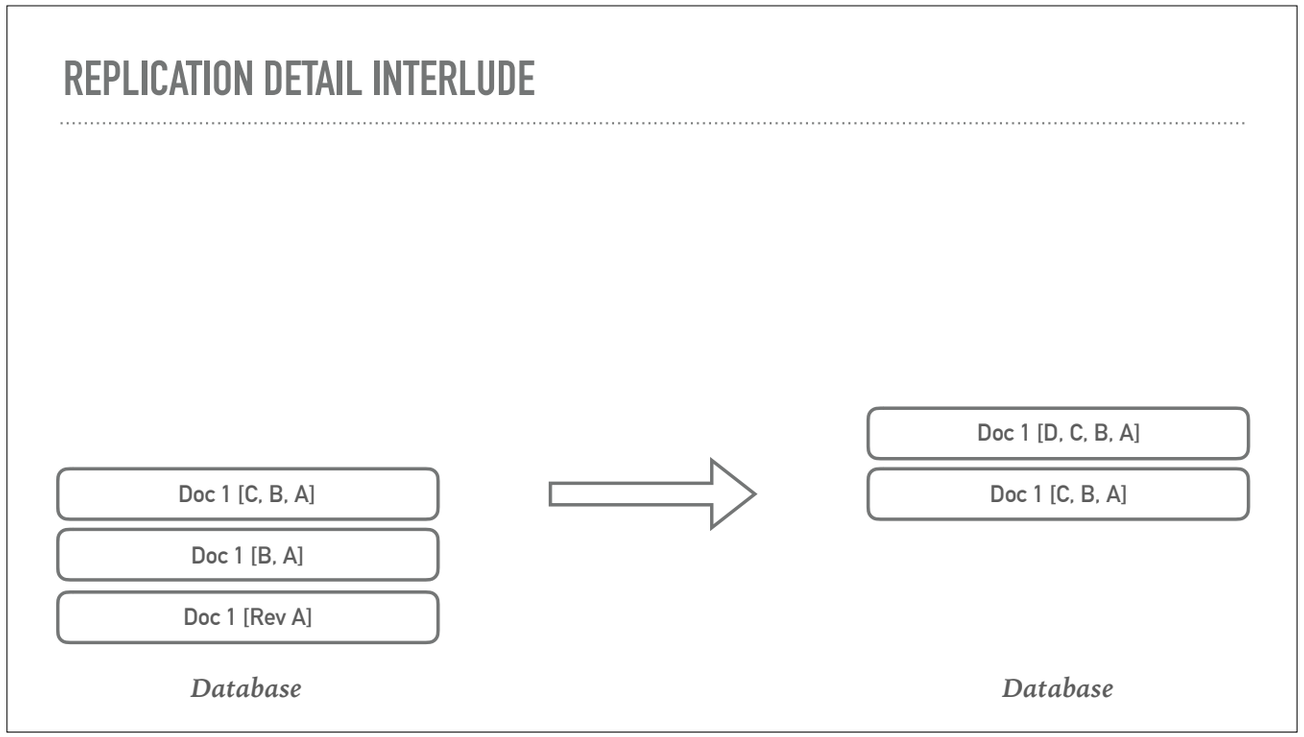

replication details



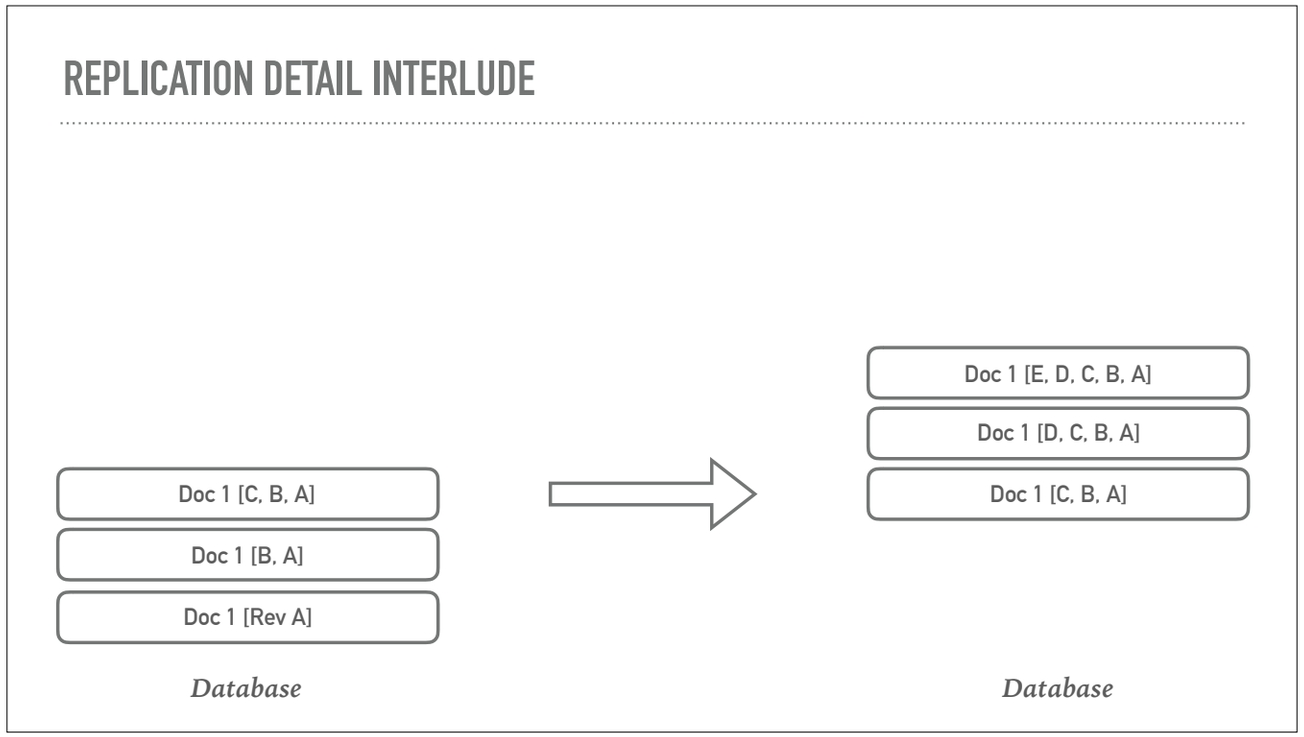

replication details



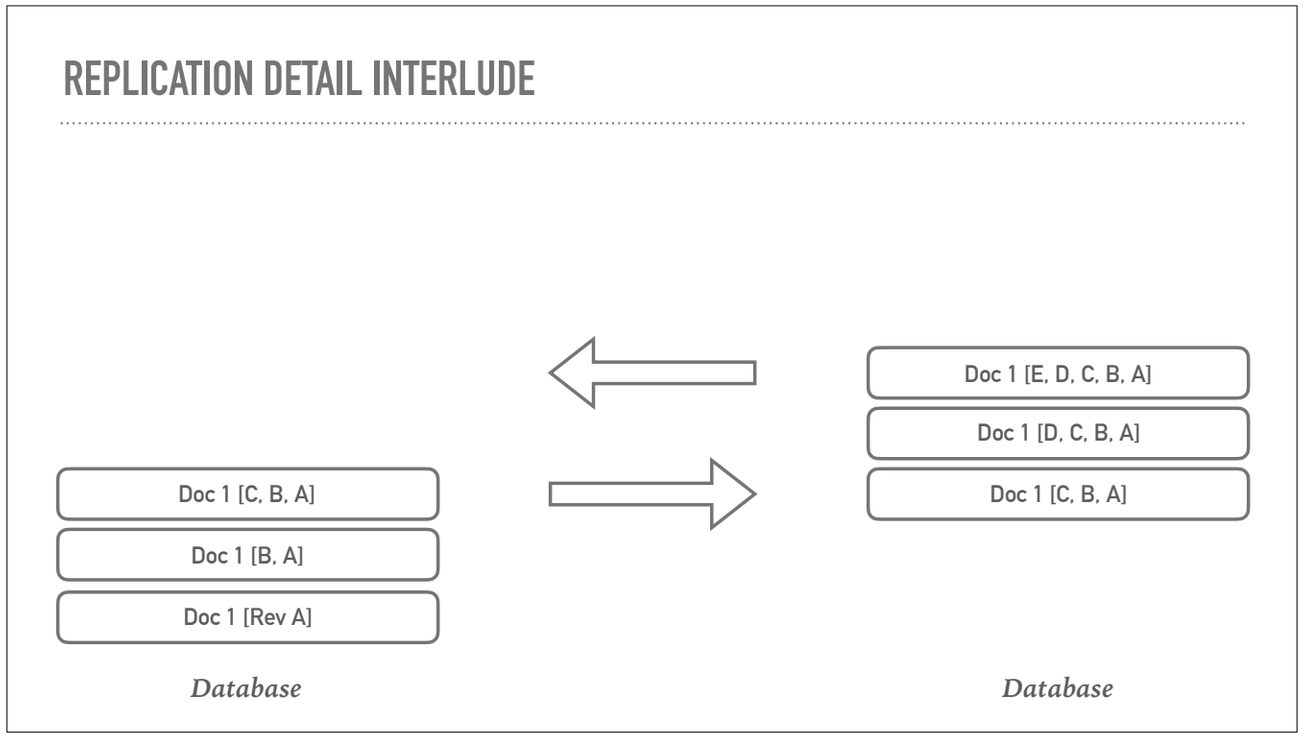

replication details



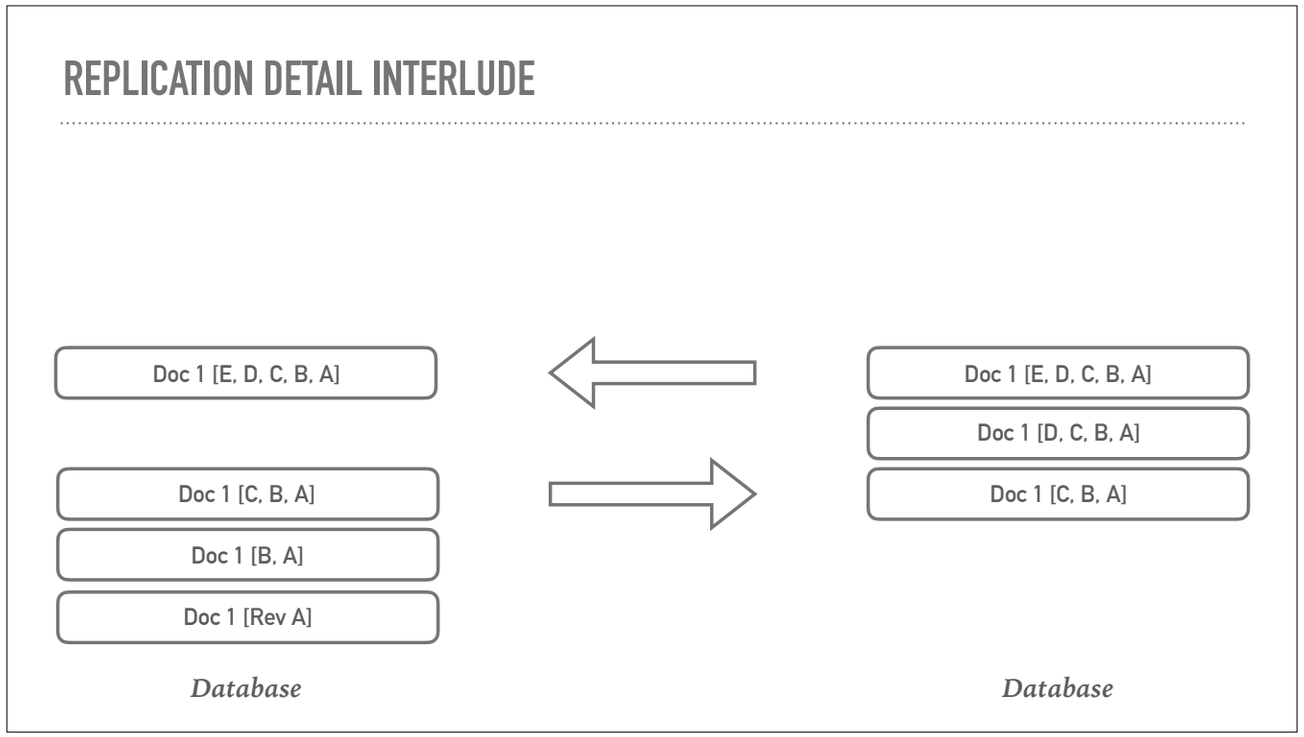

replication details



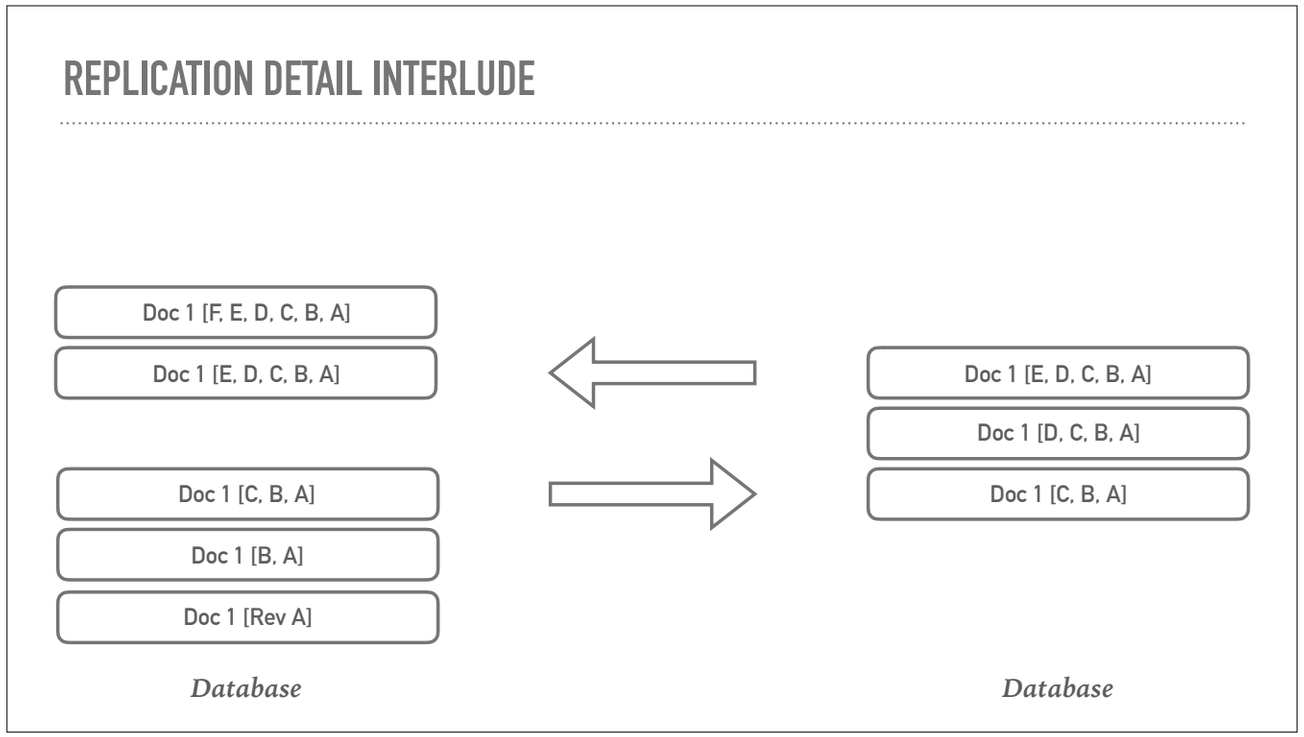

replication details



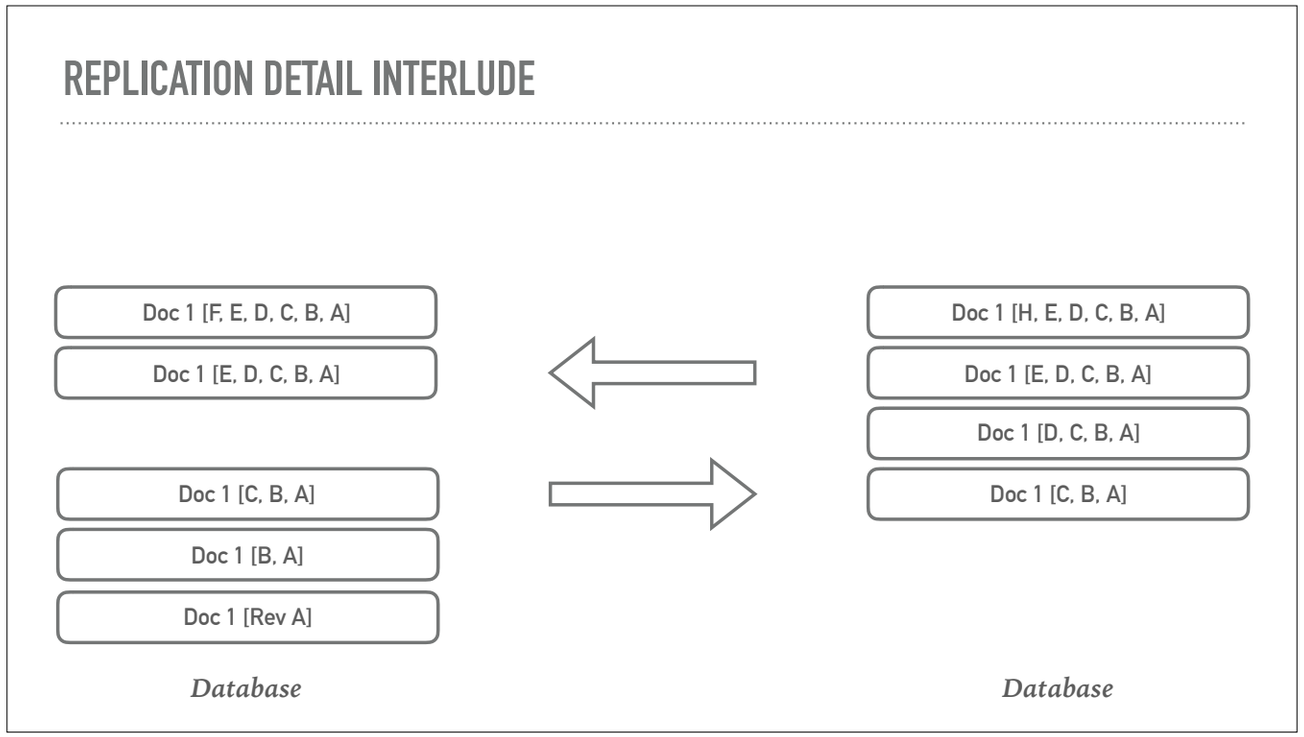

replication details



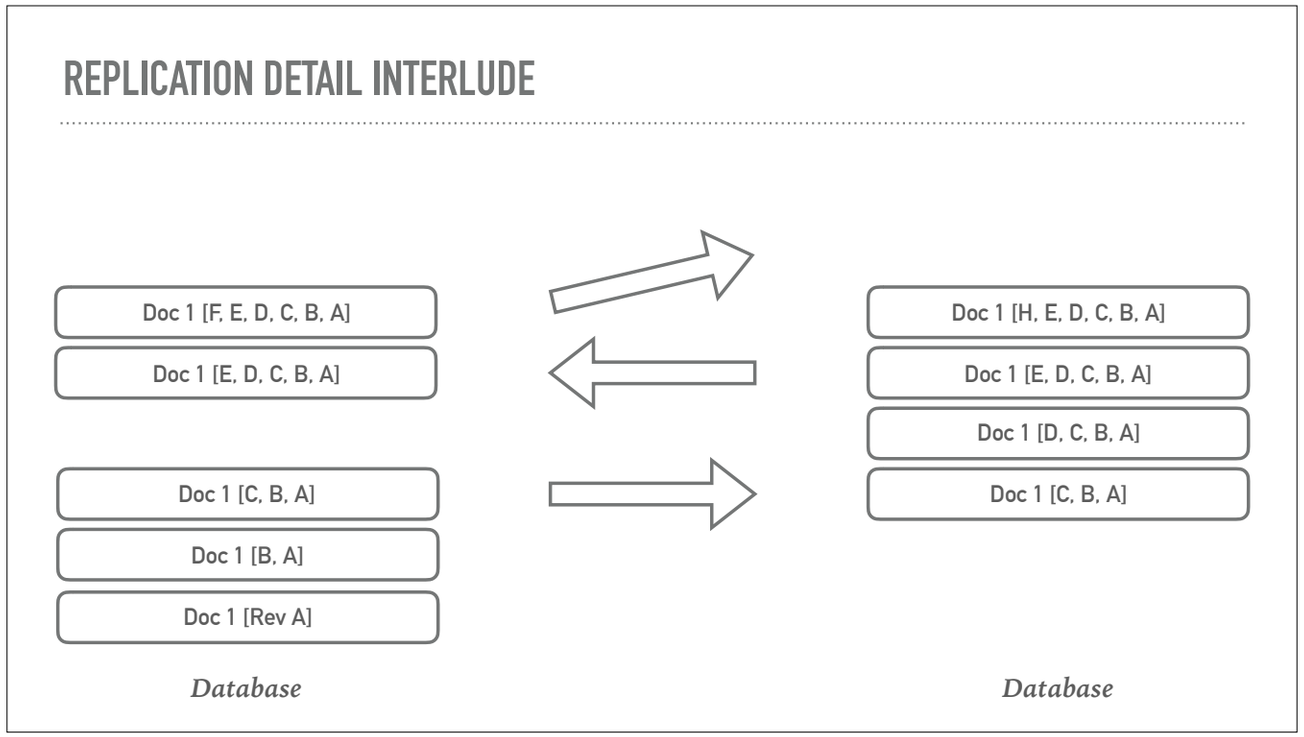

replication details



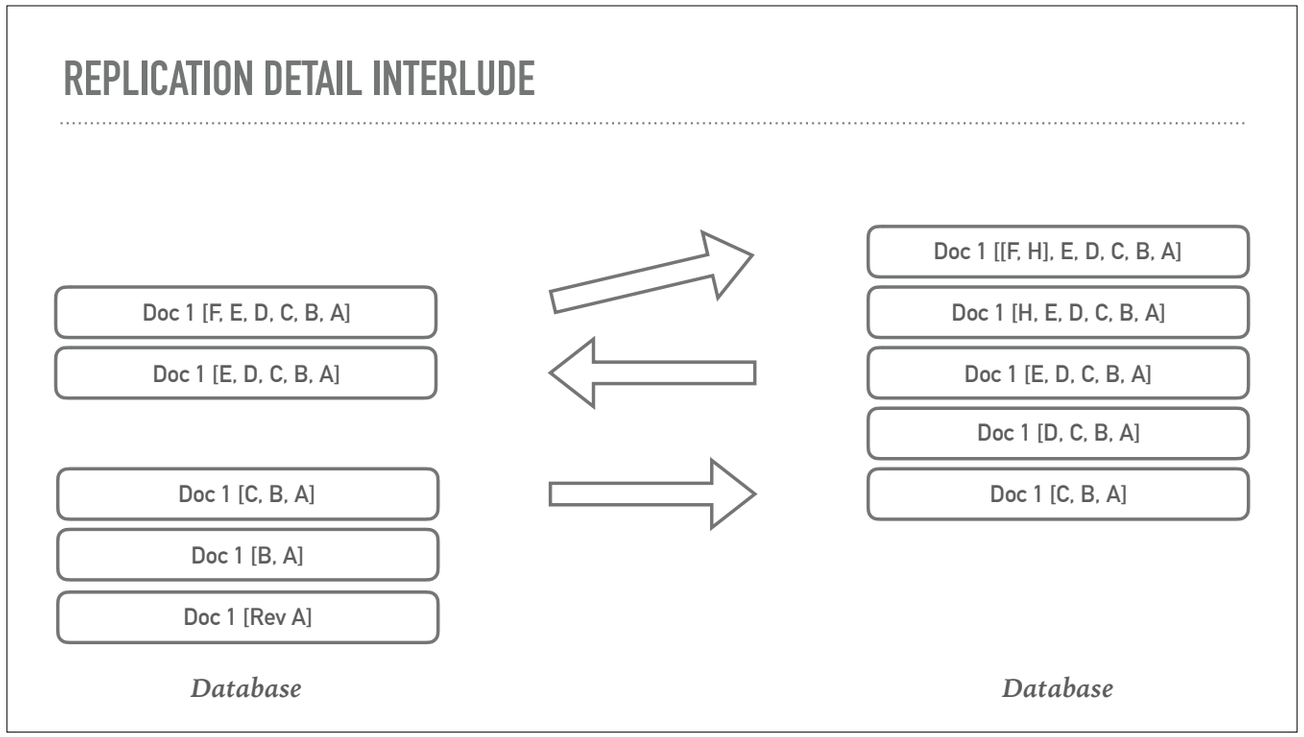

replication details



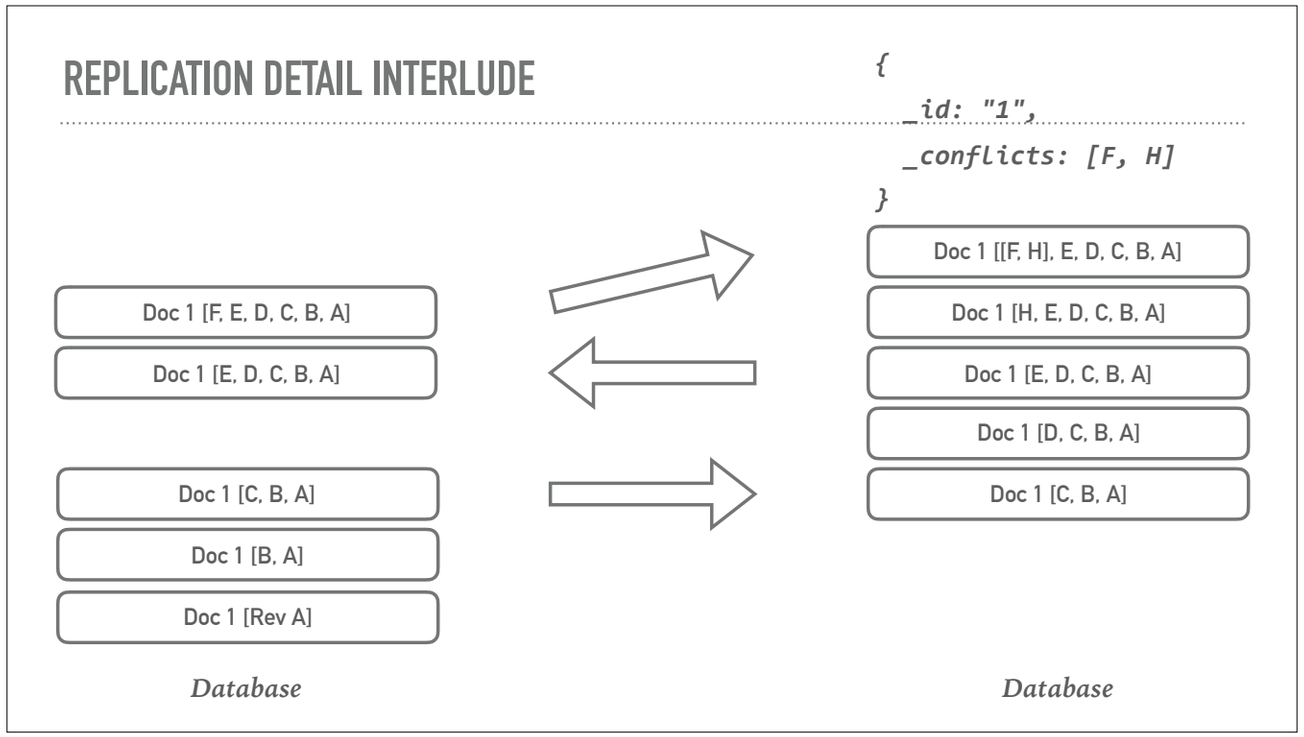

replication details




replication details




replication details



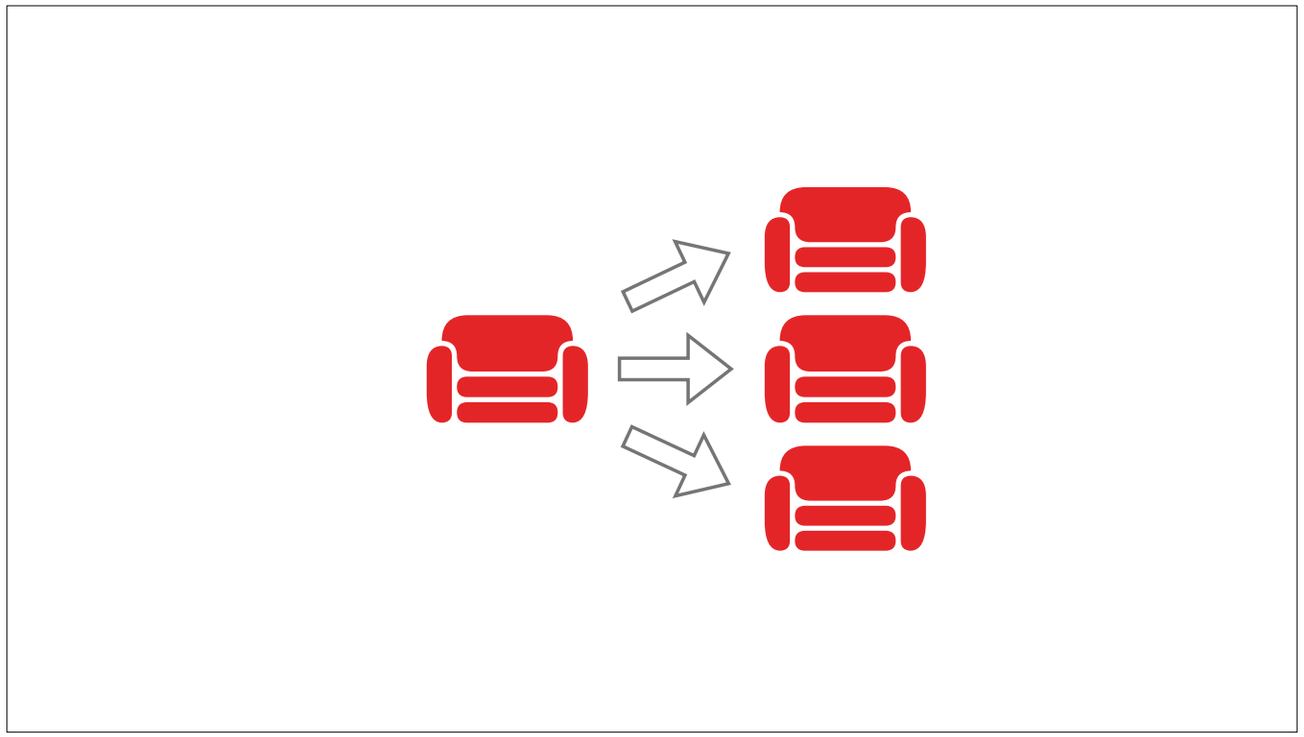

replication details

# SYNCHRONISATION PROTOCOL

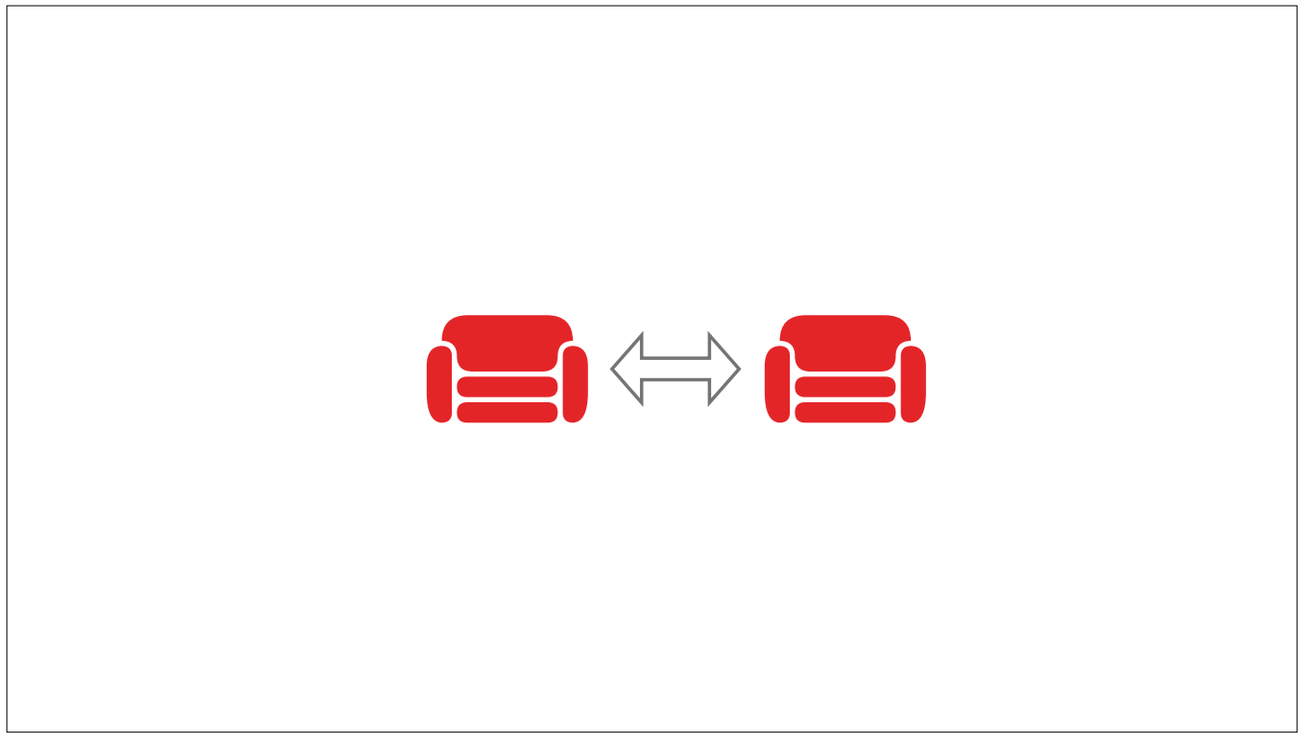
*Come see my talk tomorrow 12:00:*
*"Apache CouchDB Sync Deep Dive"*

*Or Thursday 10:30*
*if you are still here for ApacheCon EU*

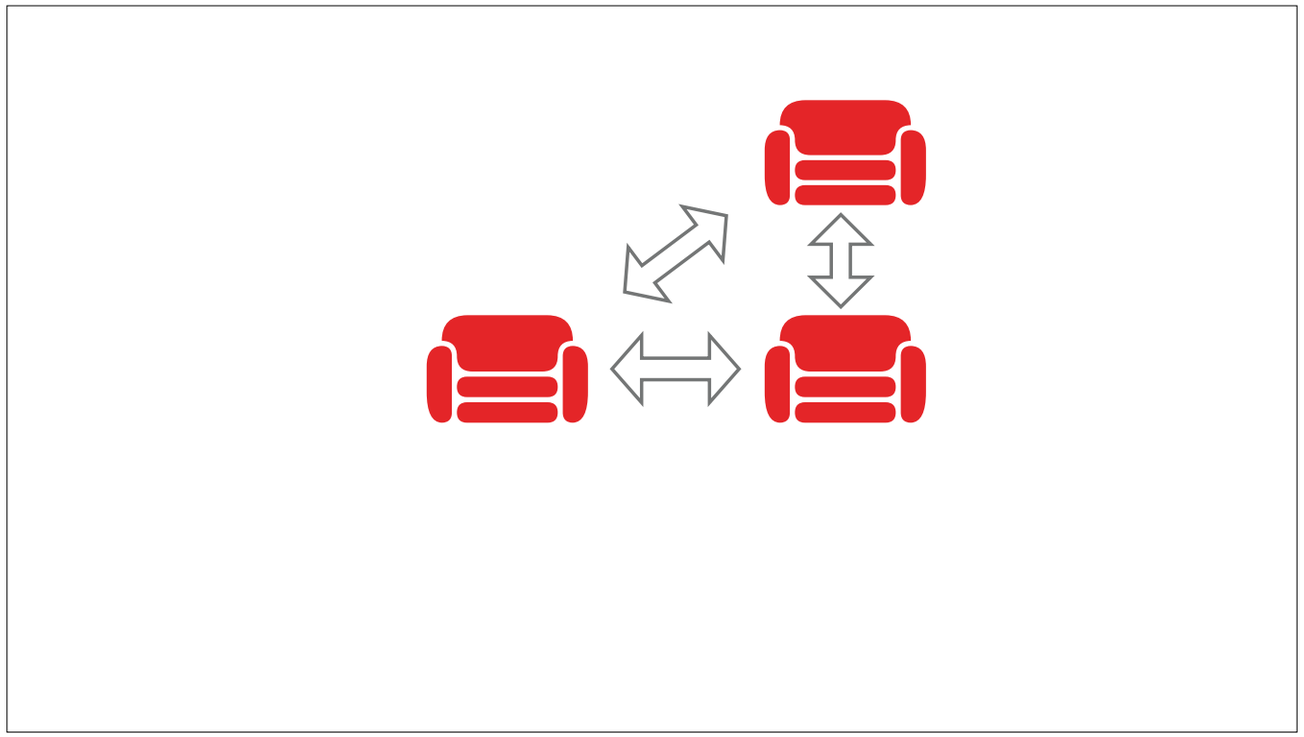
We will learn about identity, versioning schemes, revision trees, conflict detection and resolution and the by sequence index



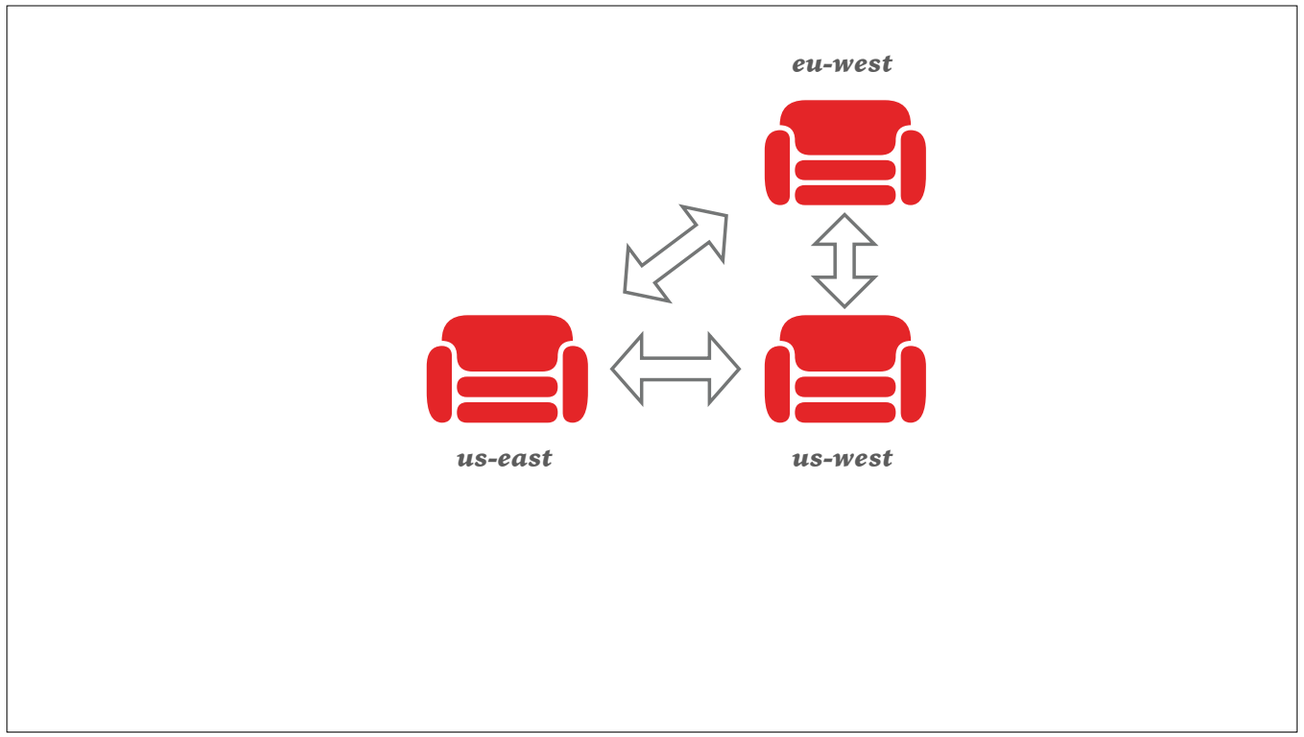
read-only secondaries



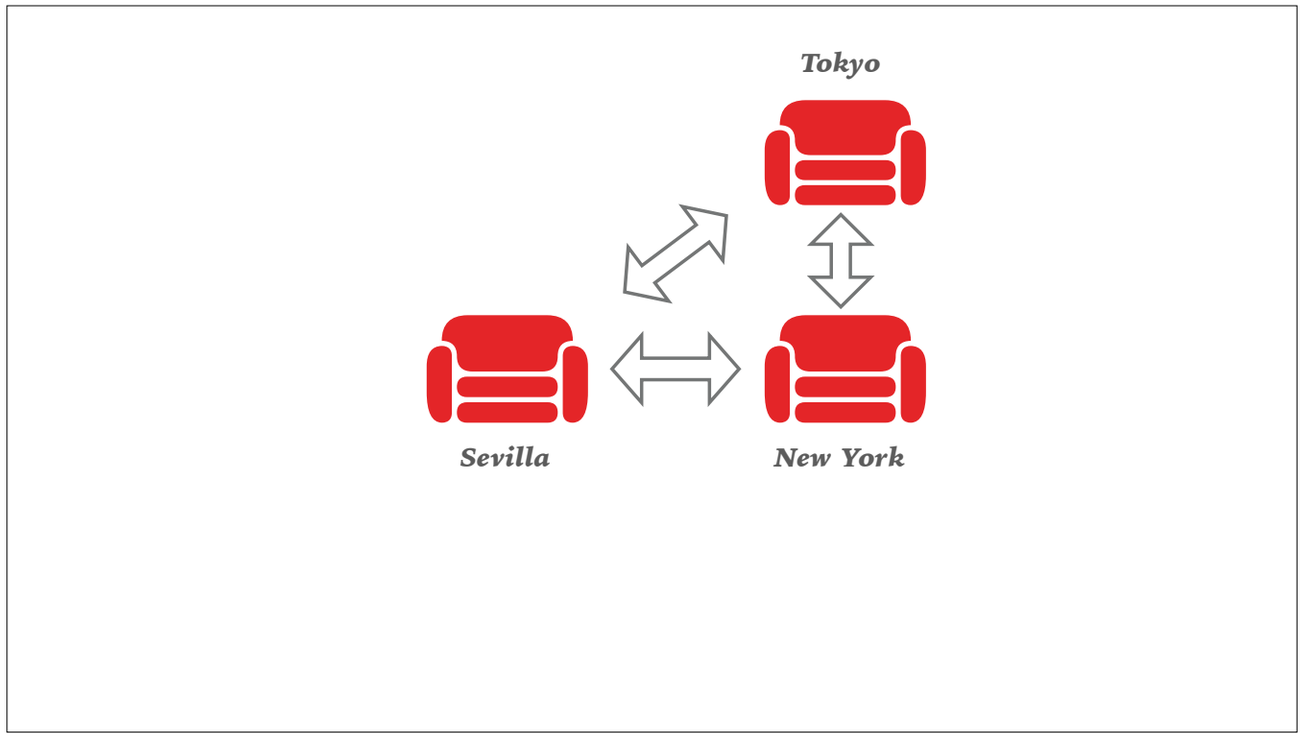
multi-primary



multi-primary

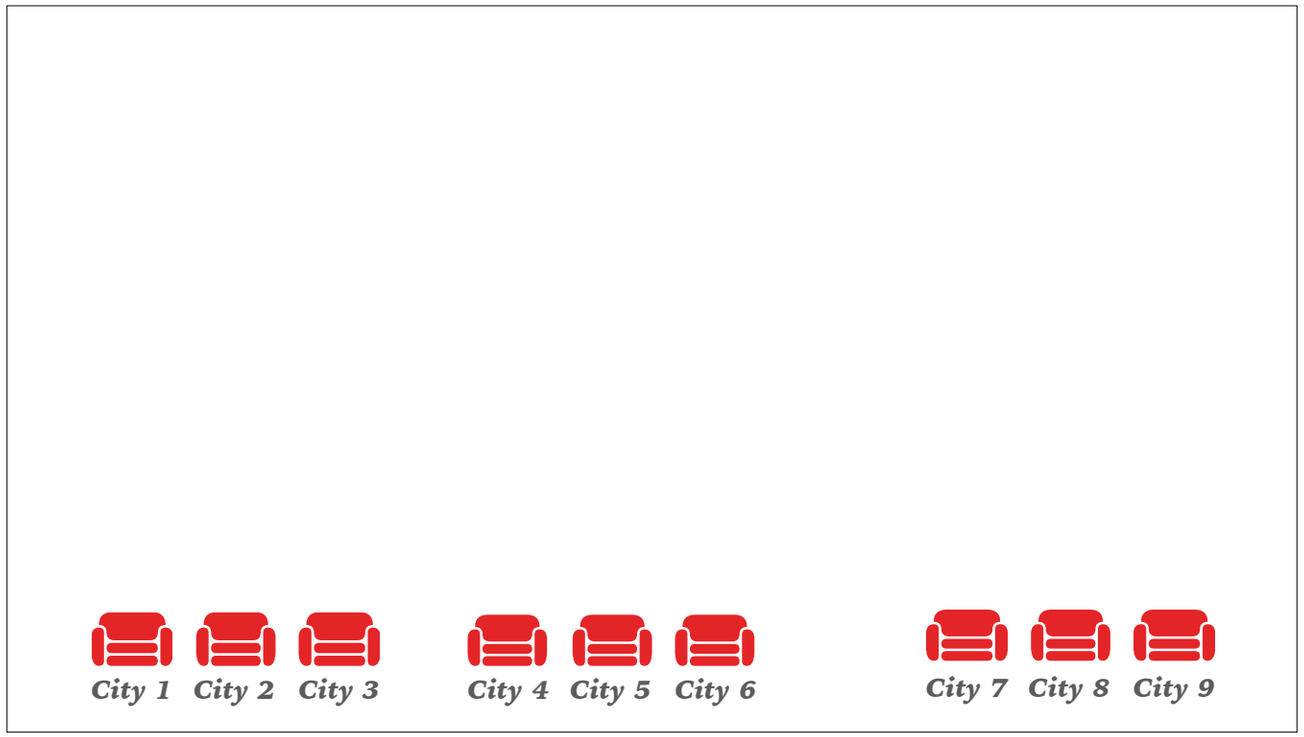



multi-primary

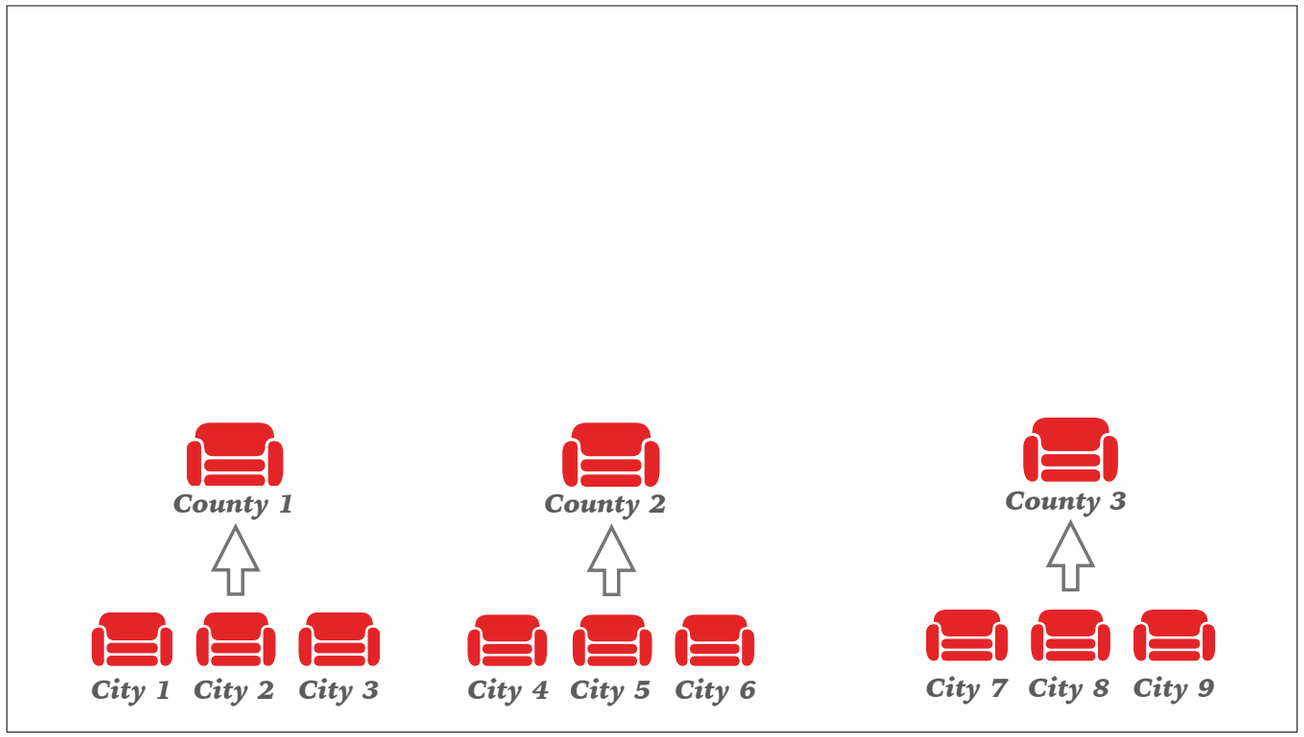



multi-primary

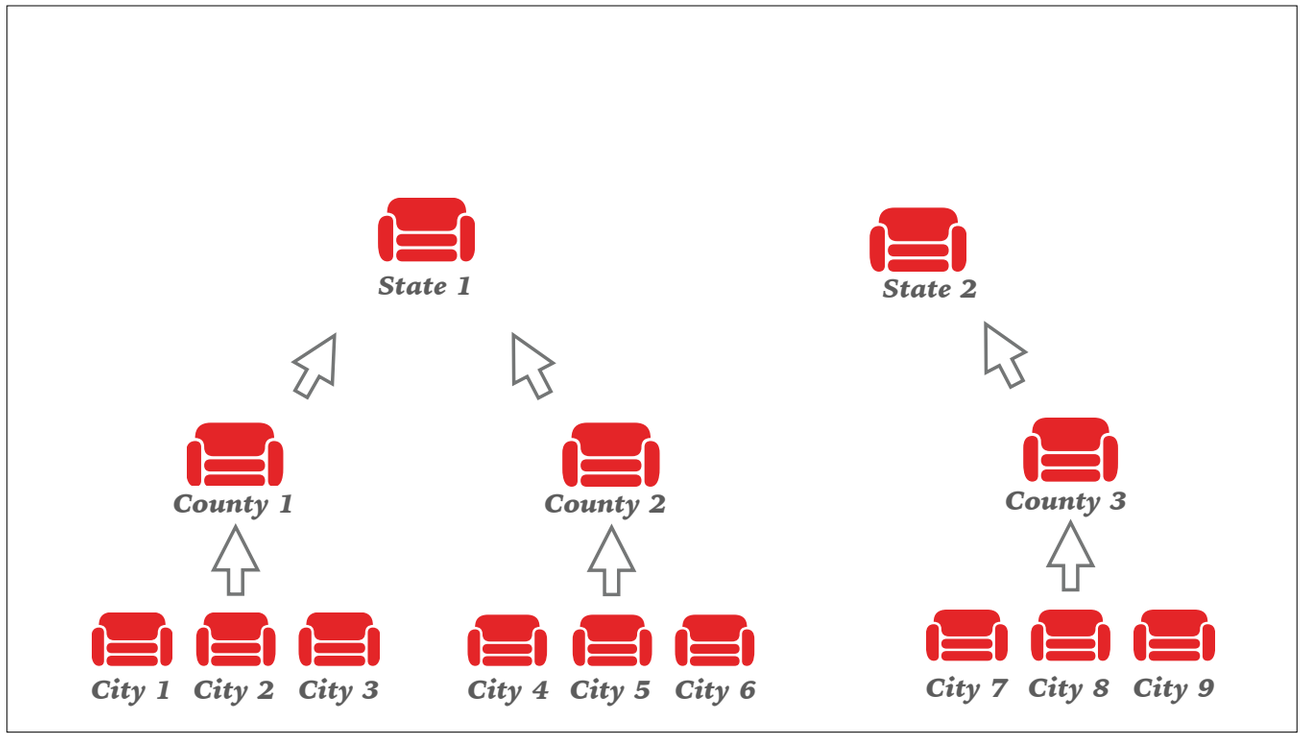
Tree



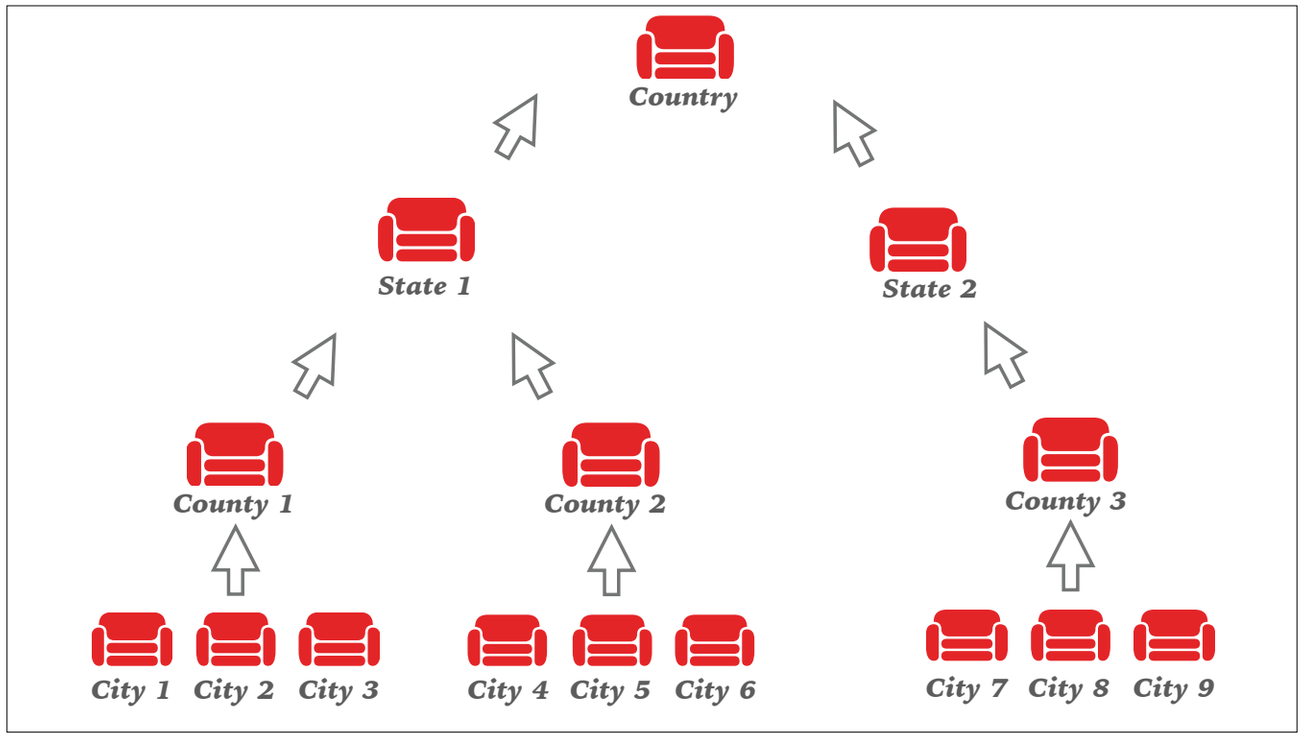

Tree


County 1
County 2
County 3
City 1
City 2
City 3
City 4
City 5
City 6
City 7
City 8
City 9

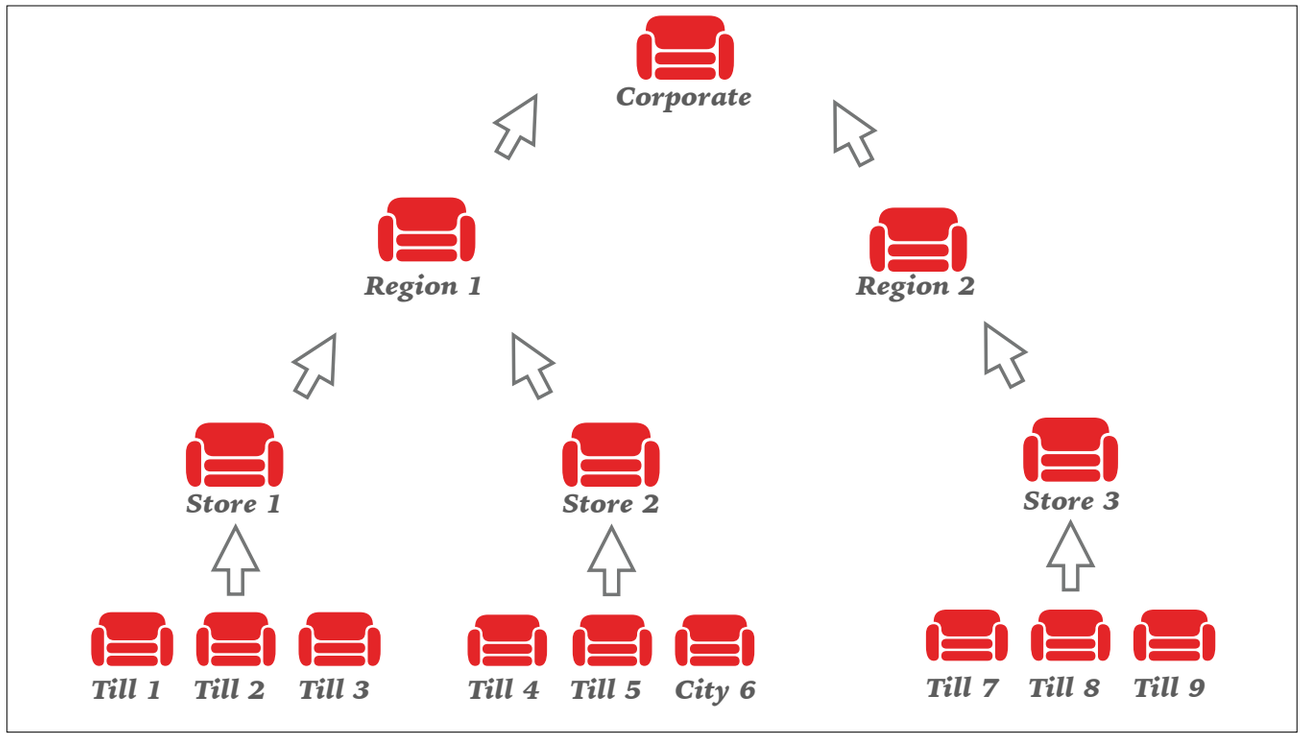
Tree


State 1
State 2
County 1
County 2
County 3
City 1
City 2
City 3
City 4
City 5
City 6
City 7
City 8
City 9

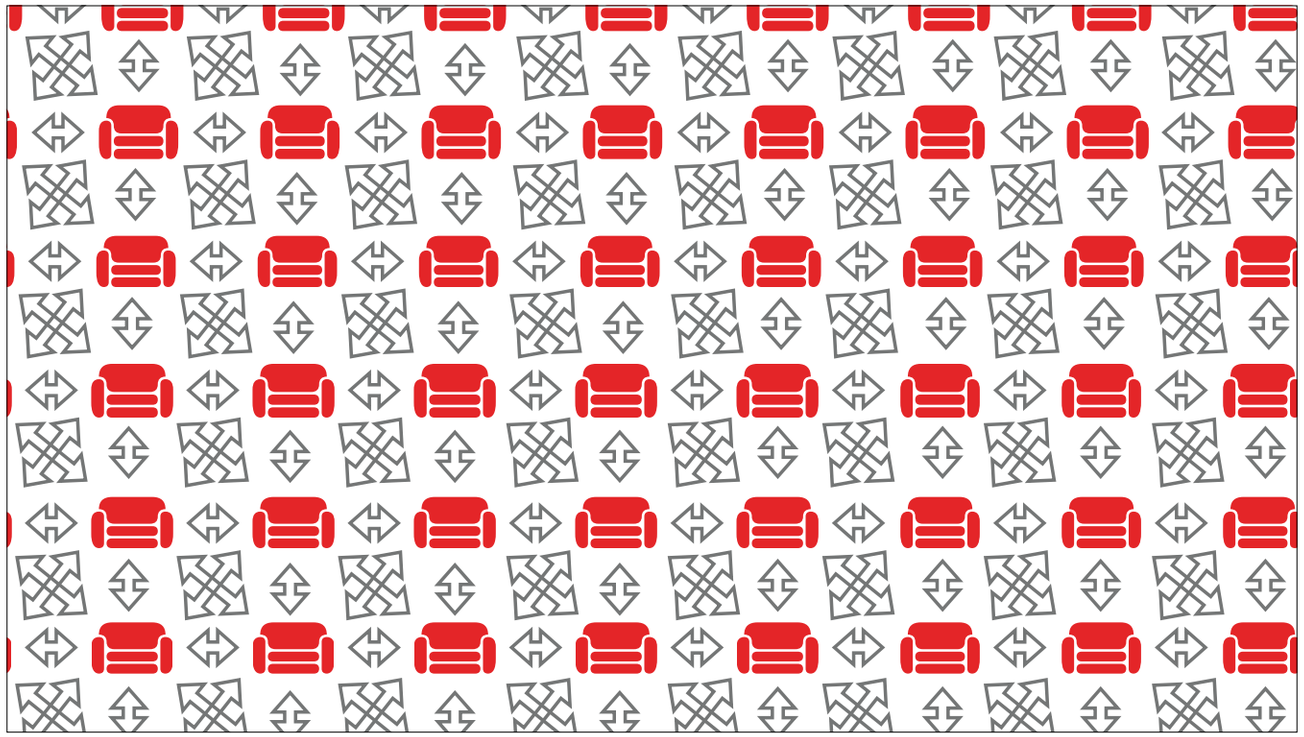
Tree


Country
State 1
State 2
County 1
County 2
County 3
City 1
City 2
City 3
City 4
City 5
City 6
City 7
City 8
City 9

Tree


Corporate
Region 1
Region 2
Store 1
Store 2
Store 3
Till 1
Till 2
Till 3
Till 4
Till 5
City 6
Till 7
Till 8
Till 9

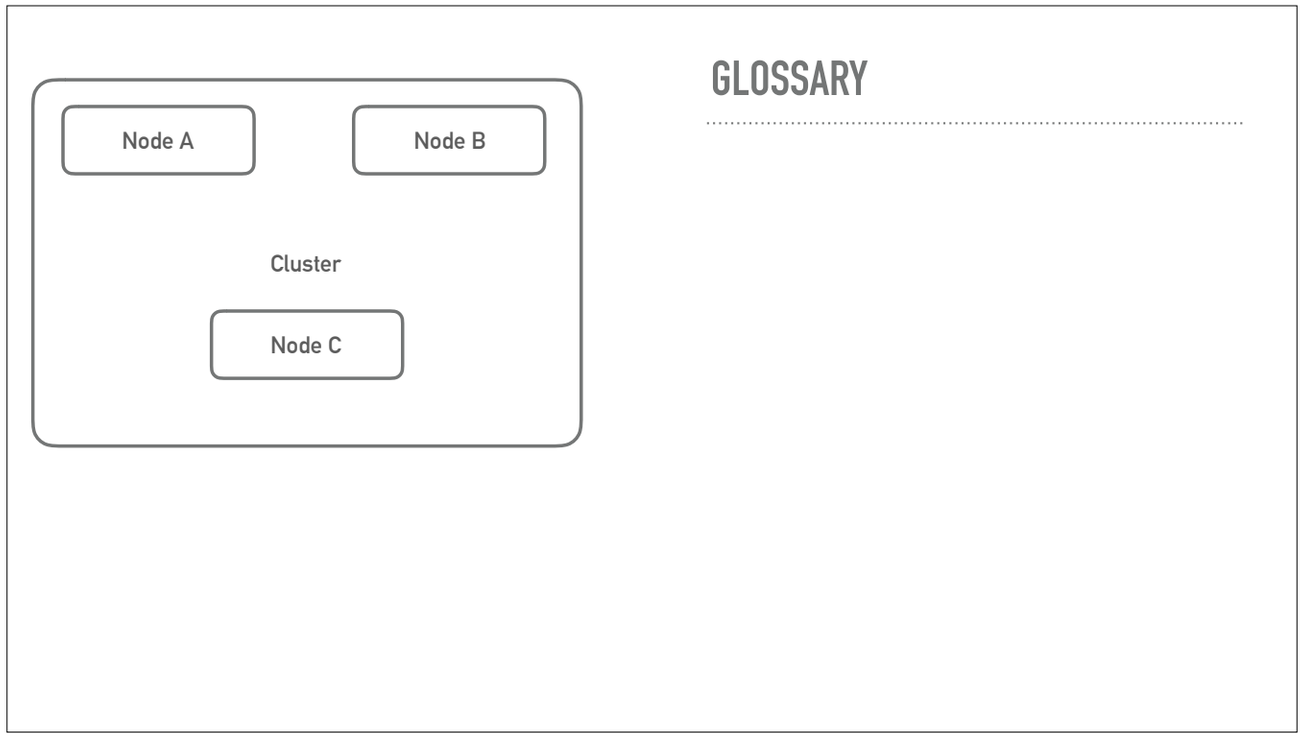
Tree



Mesh
c.f. Internet of Things / Industry of Things

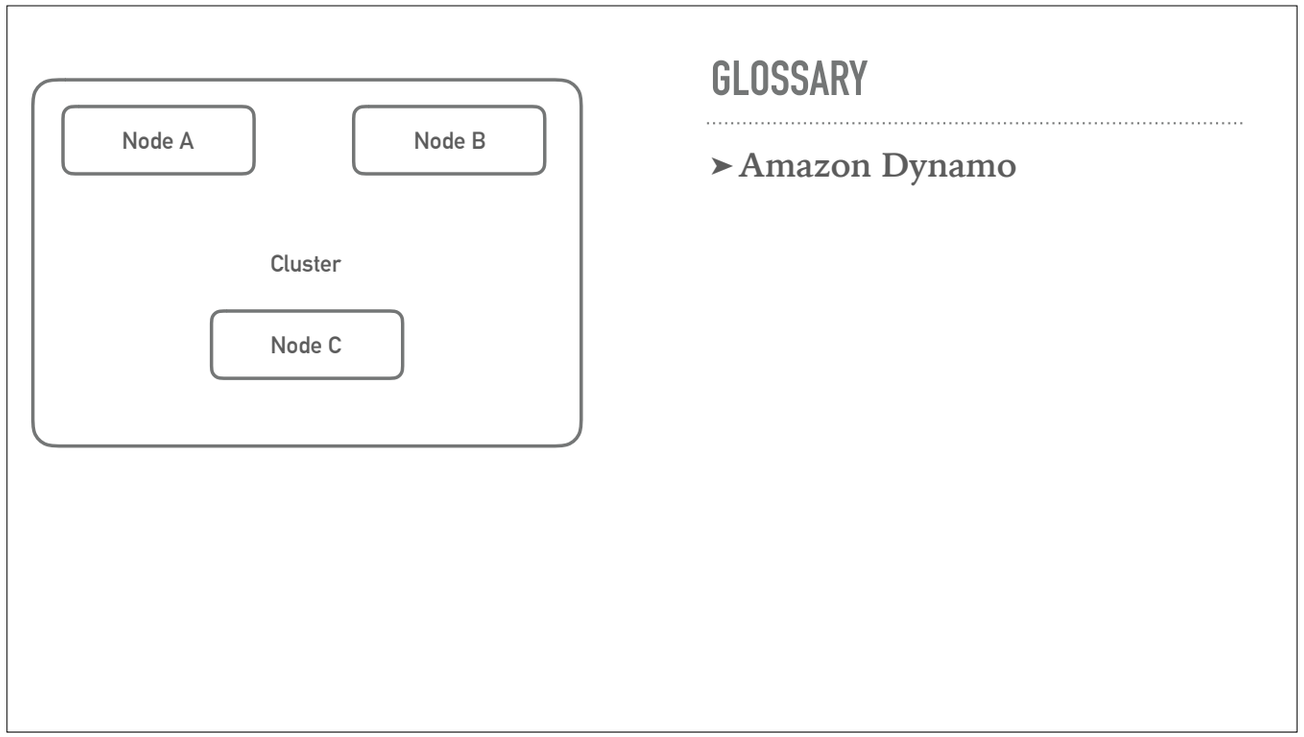
# CLUSTER INTERNALS

# Cluster
- Amazon Dynamo
- Cluster -> nodes
- Database -> shards
- No Primary Node
    - any node can answer any request
        - worst case proxies from other nodes
        - adds a hop, possible latency optimisation with “cluster aware” client libraries
- Consistency: R/W = 1,2,3,N
    - query n=1 asks only one node
    - n=2 asks two nodes
    - n=3 three nodes and so on
    - >n == mode latency vs. more consistency
    - optimisation opportunity: balance of probabilities:
        - do we have to fsync write to two nodes, or is it enough to commit to two memories?
- self healing
- read repair
- full replication support
- 99% API compatible



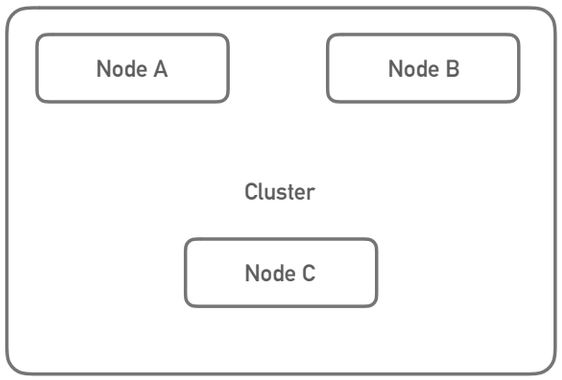

Physical: Cluster & Nodes
Logical: Databases & Shards
Shard map dynamic: you can put shards on different nodes
to scale, first overshard, then move shards to new hardware
- has limit
- re-sharding in future version

no limit to number of nodes or shards or data stored




Physical: Cluster & Nodes
Logical: Databases & Shards
Shard map dynamic: you can put shards on different nodes
to scale, first overshard, then move shards to new hardware
- has limit
- re-sharding in future version

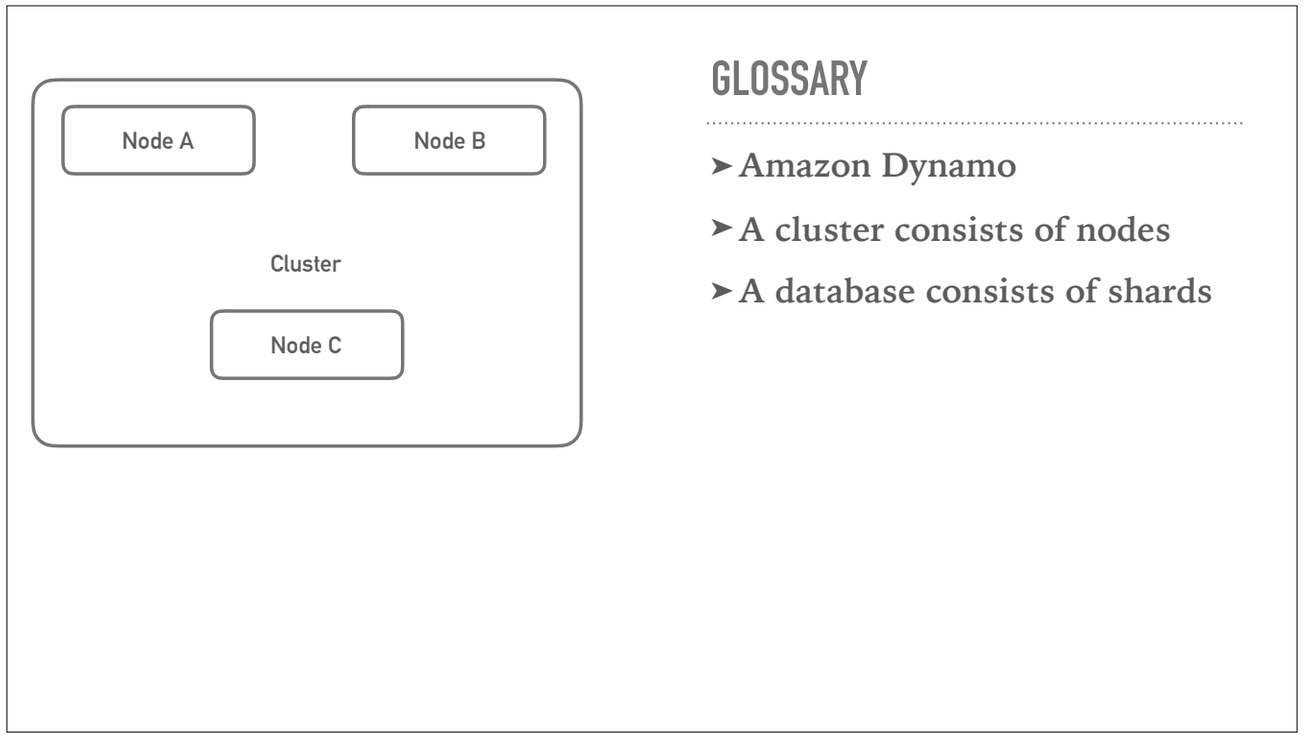
no limit to number of nodes or shards or data stored




## GLOSSARY

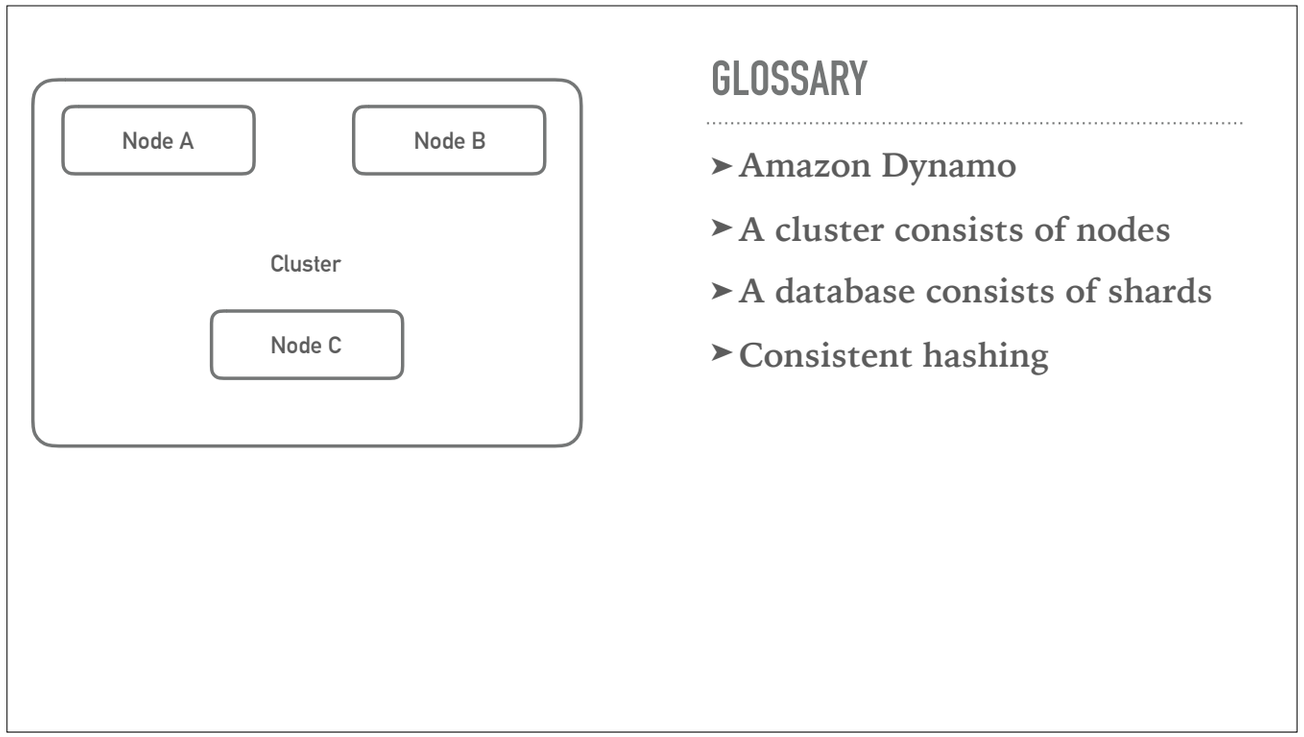
- Amazon Dynamo
- A cluster consists of nodes

Physical: Cluster & Nodes
Logical: Databases & Shards
Shard map dynamic: you can put shards on different nodes
to scale, first overshard, then move shards to new hardware
    - has limit
    - re-sharding in future version
no limit to number of nodes or shards or data stored



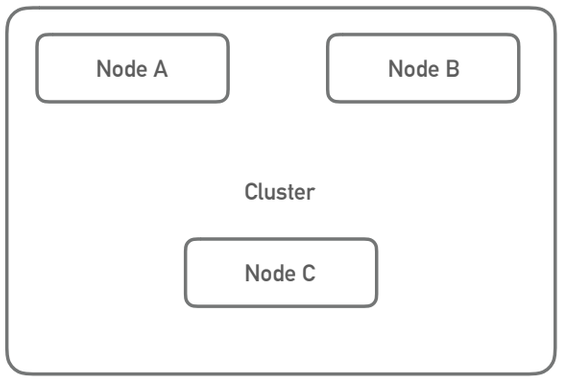

Physical: Cluster & Nodes
Logical: Databases & Shards
Shard map dynamic: you can put shards on different nodes
to scale, first overshard, then move shards to new hardware
- has limit
- re-sharding in future version

no limit to number of nodes or shards or data stored




Physical: Cluster & Nodes
Logical: Databases & Shards
Shard map dynamic: you can put shards on different nodes
to scale, first overshard, then move shards to new hardware
- has limit
- re-sharding in future version

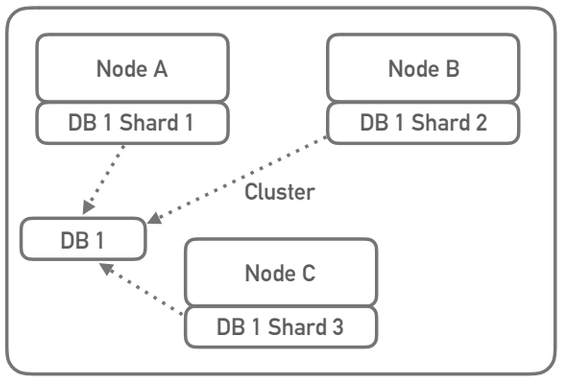
no limit to number of nodes or shards or data stored




## GLOSSARY

- Amazon Dynamo
- A cluster consists of nodes
- A database consists of shards
- Consistent hashing
  - Document ID -> hash fun -> shard ID

Physical: Cluster & Nodes
Logical: Databases & Shards
Shard map dynamic: you can put shards on different nodes
to scale, first overshard, then move shards to new hardware
    - has limit
    - re-sharding in future version
no limit to number of nodes or shards or data stored

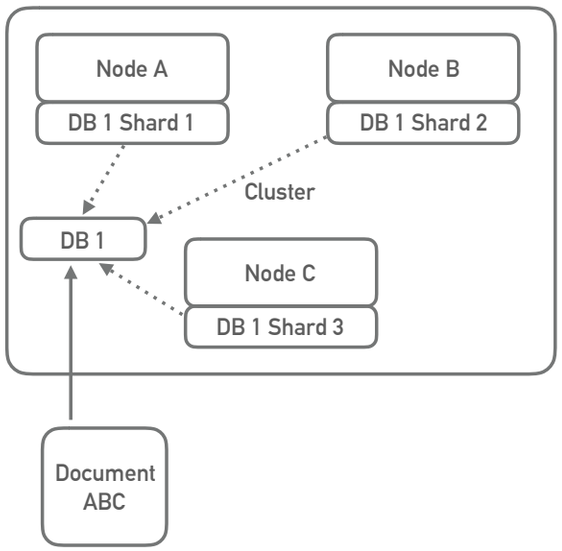



## GLOSSARY

- Amazon Dynamo
- A cluster consists of nodes
- A database consists of shards
- Consistent hashing
  - Document ID -> hash fun -> shard ID

Physical: Cluster & Nodes
Logical: Databases & Shards
Shard map dynamic: you can put shards on different nodes
to scale, first overshard, then move shards to new hardware
- has limit
- re-sharding in future version

no limit to number of nodes or shards or data stored

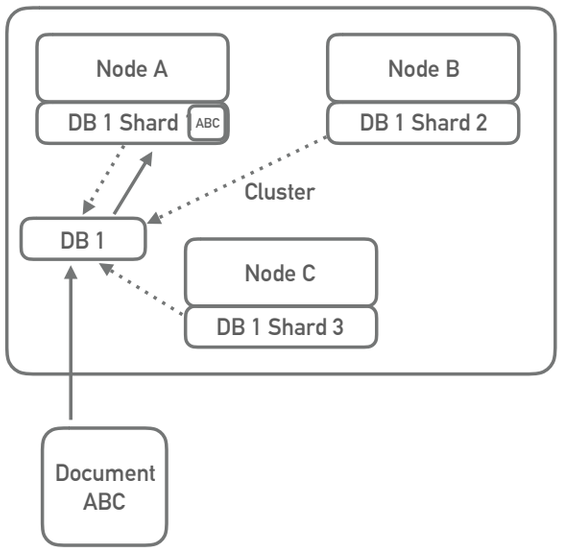



## GLOSSARY

- Amazon Dynamo
- A cluster consists of nodes
- A database consists of shards
- Consistent hashing
  - Document ID -> hash fun -> shard ID

Physical: Cluster & Nodes
Logical: Databases & Shards
Shard map dynamic: you can put shards on different nodes
to scale, first overshard, then move shards to new hardware
- has limit
- re-sharding in future version

no limit to number of nodes or shards or data stored

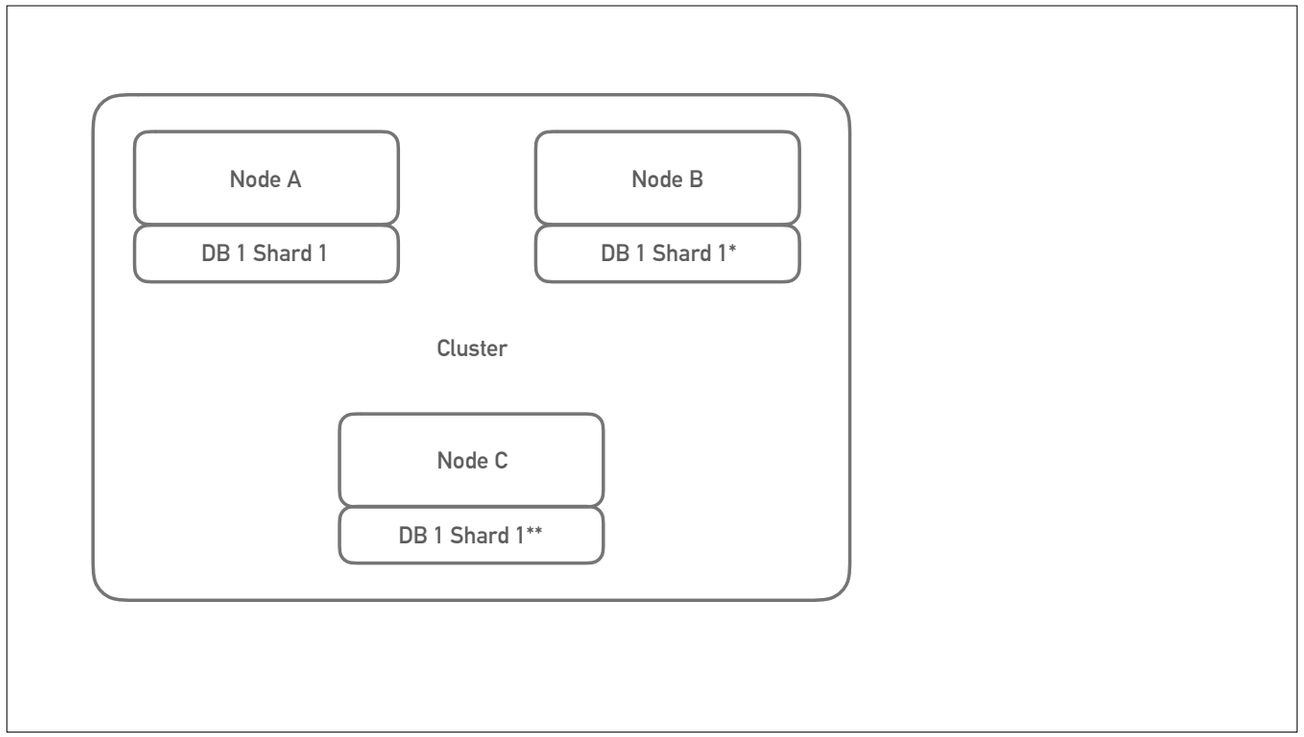



## GLOSSARY

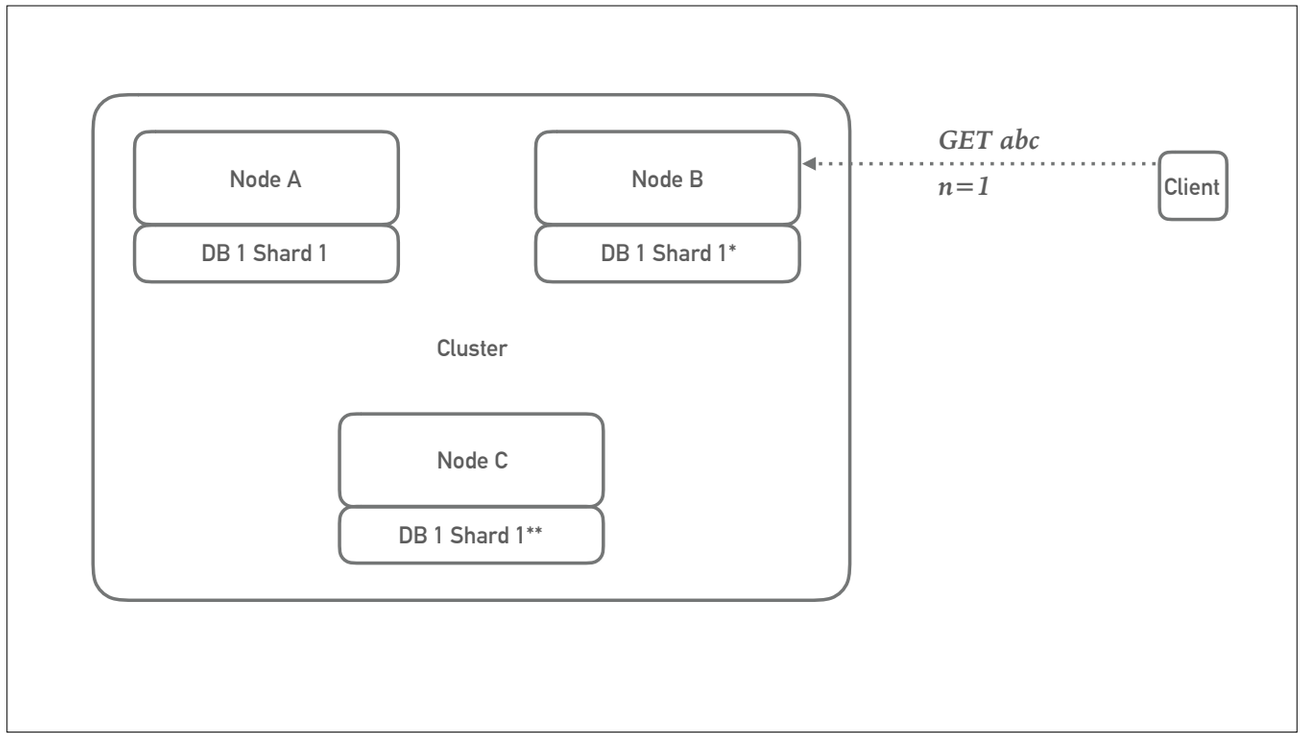
- Amazon Dynamo
- A cluster consists of nodes
- A database consists of shards
- Consistent hashing
  - Document ID -> hash fun -> shard ID

Physical: Cluster & Nodes
Logical: Databases & Shards
Shard map dynamic: you can put shards on different nodes
to scale, first overshard, then move shards to new hardware
- has limit
- re-sharding in future version

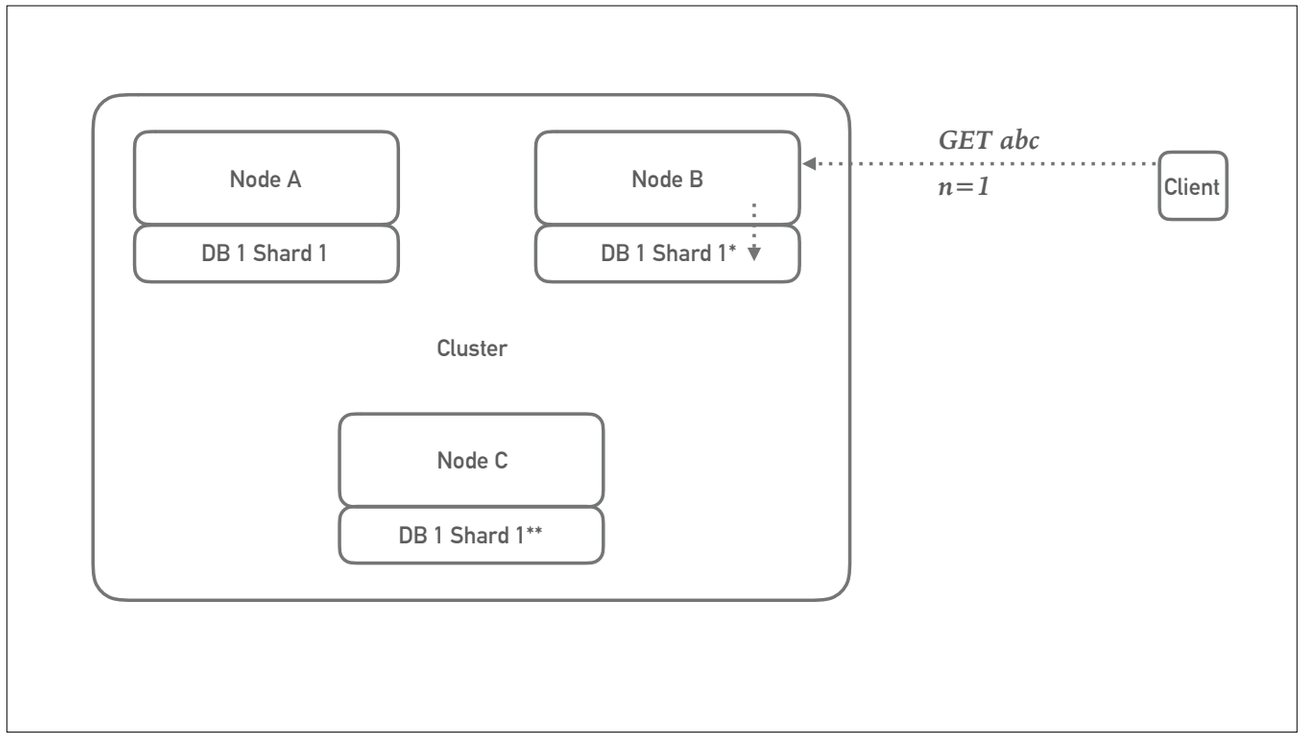
no limit to number of nodes or shards or data stored



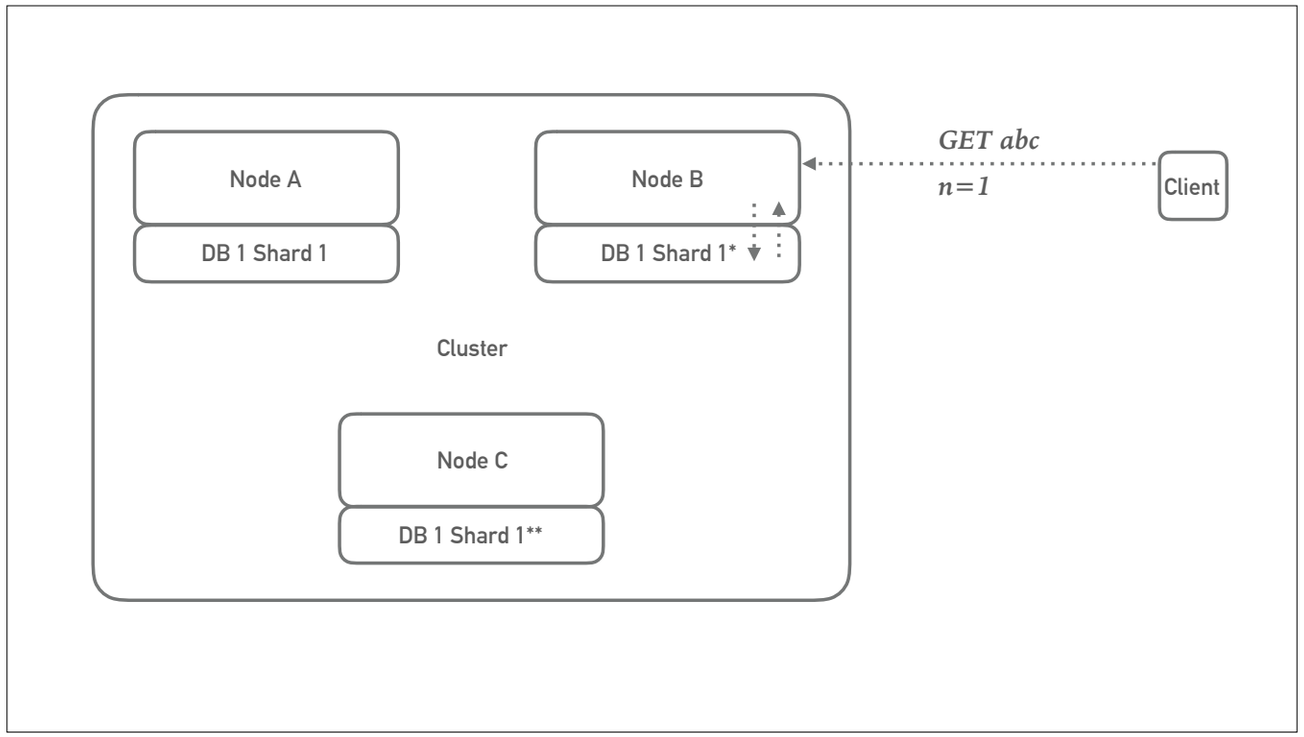

backup shards



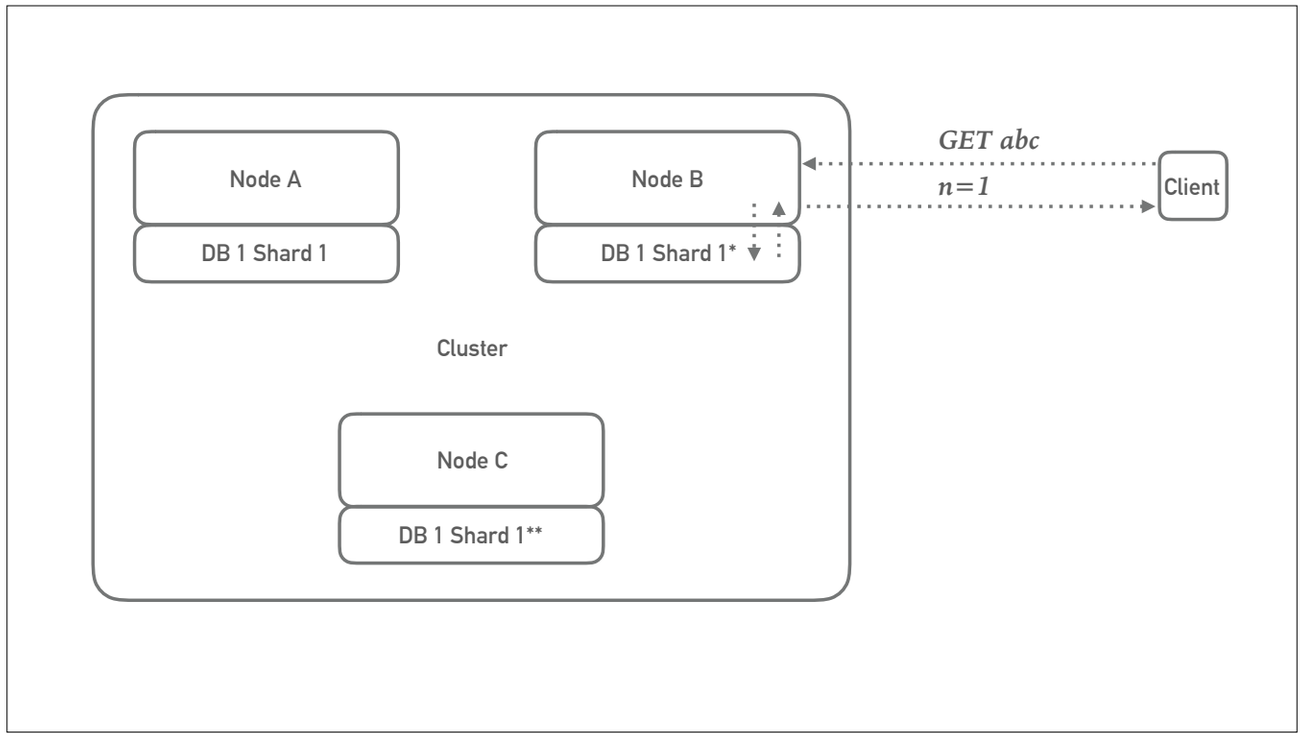

backup shards



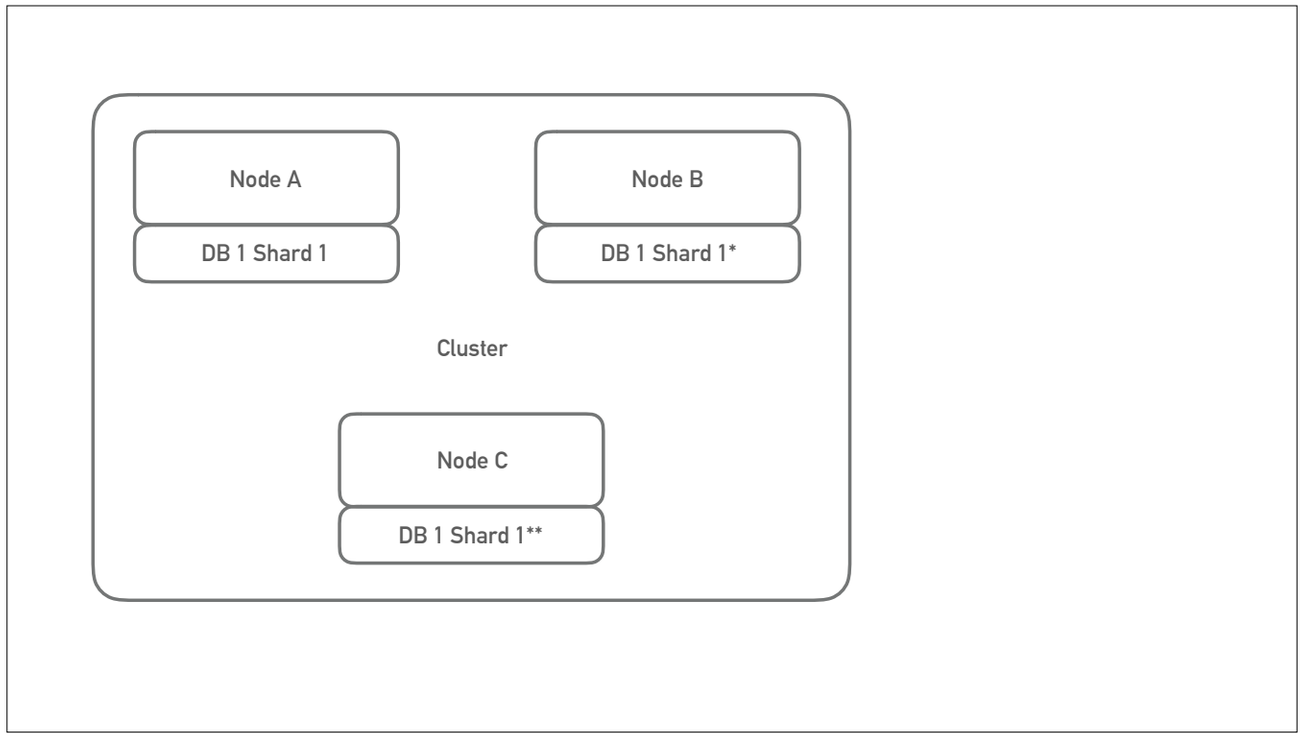

backup shards

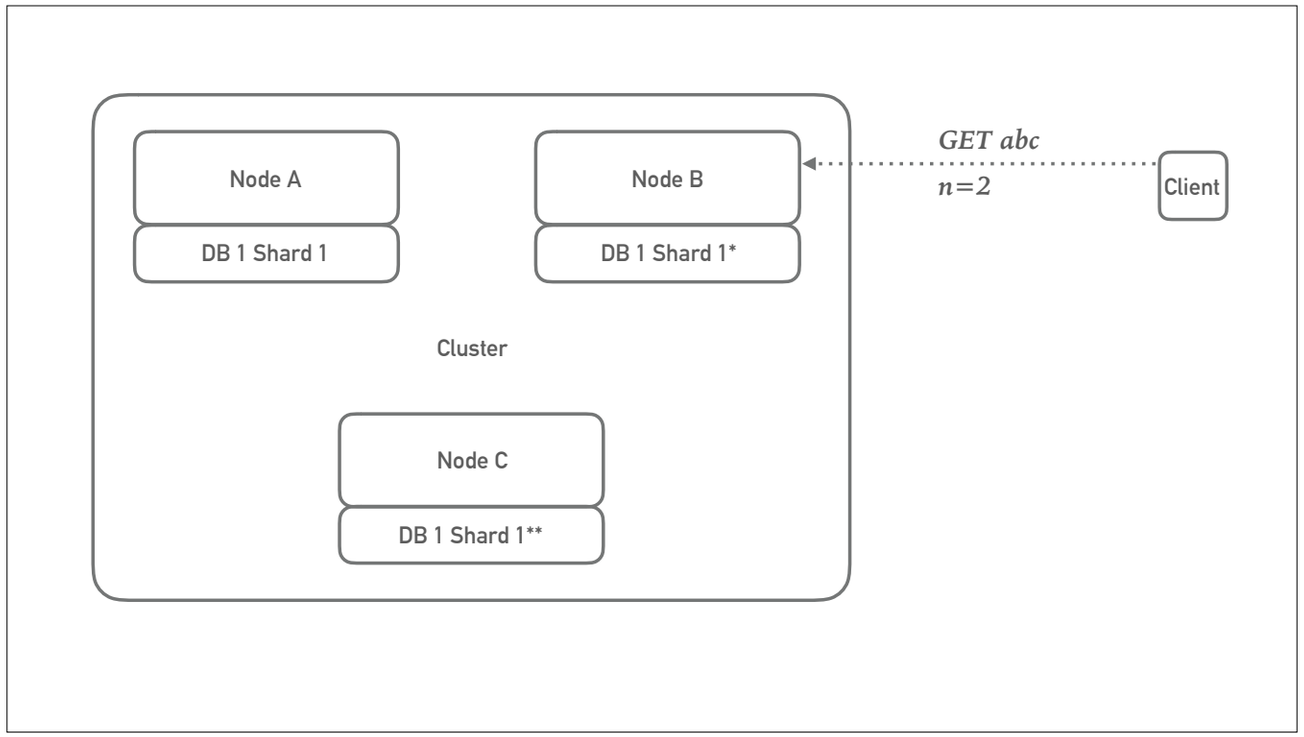


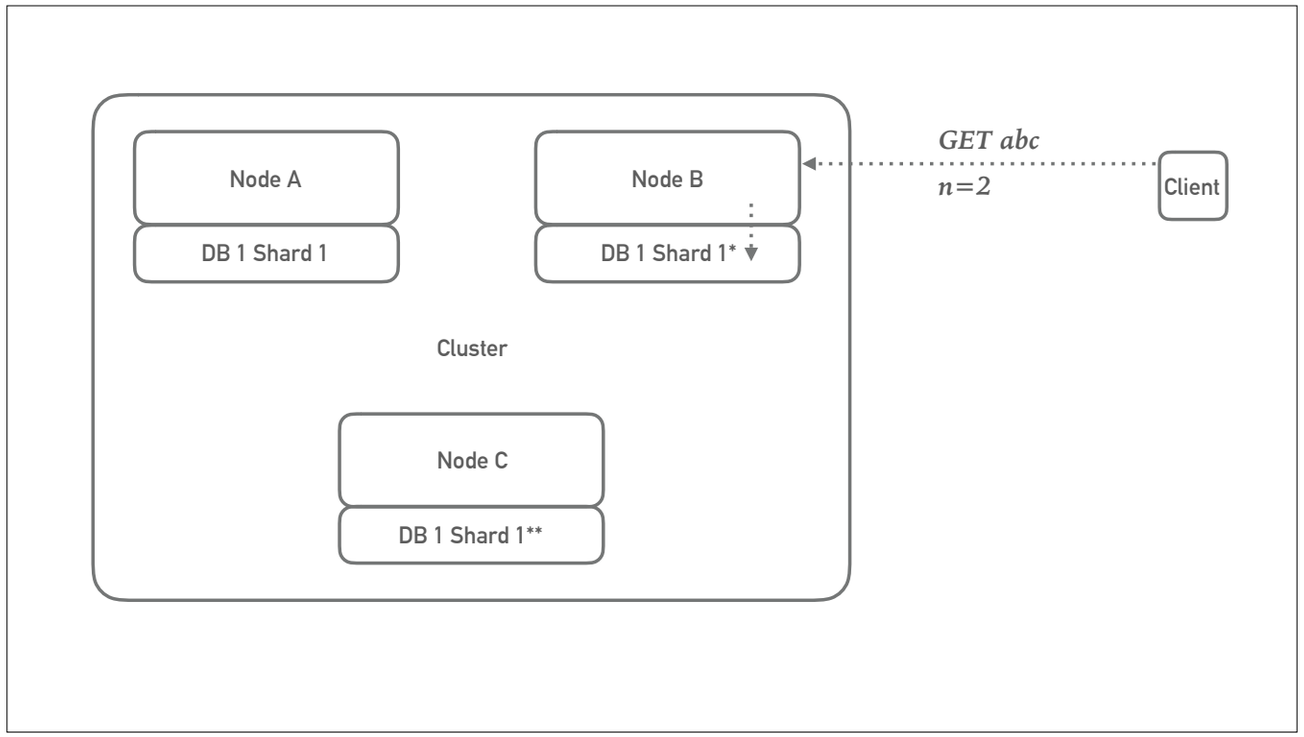

backup shards

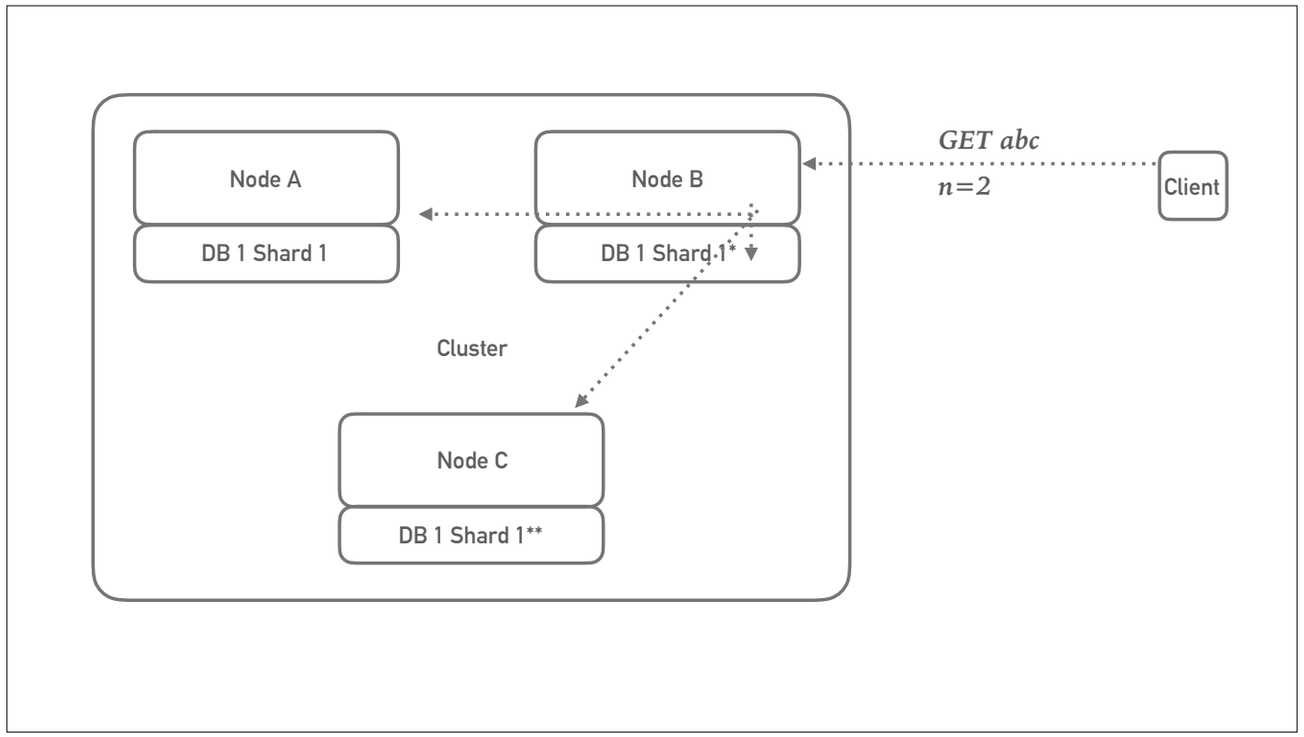


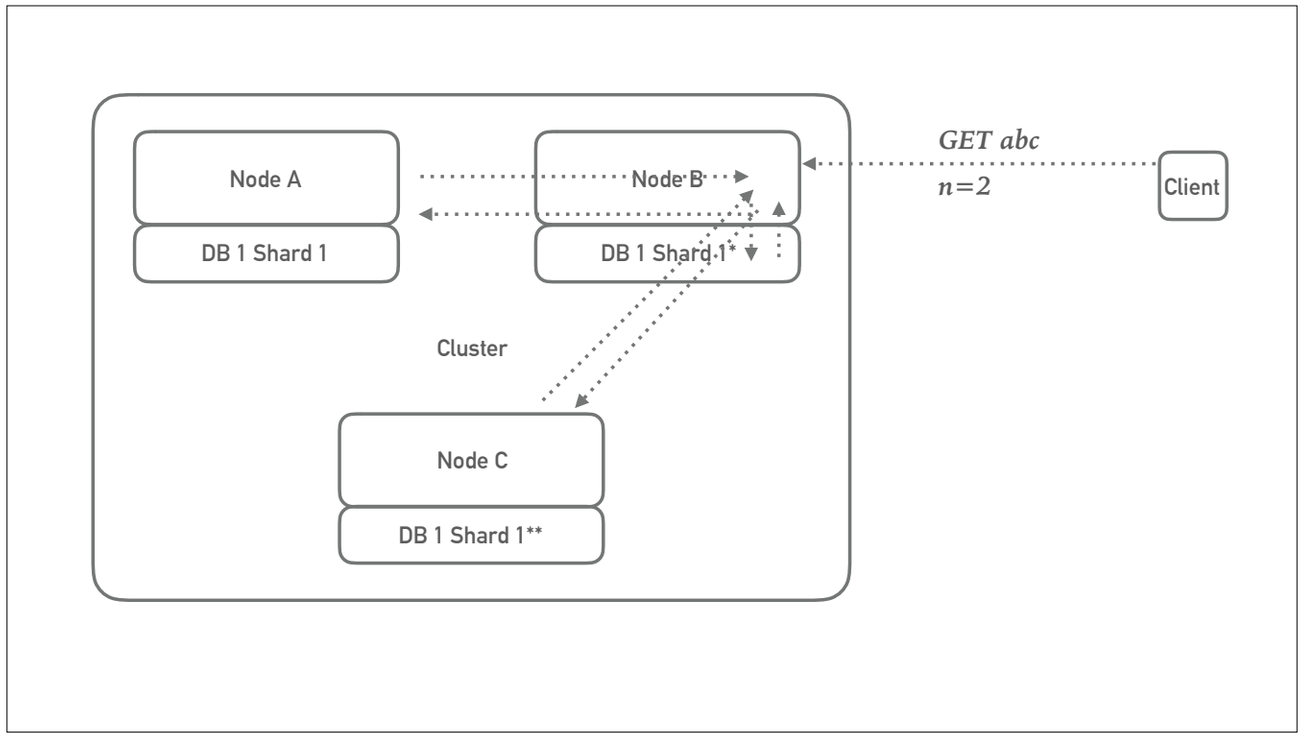

backup shards

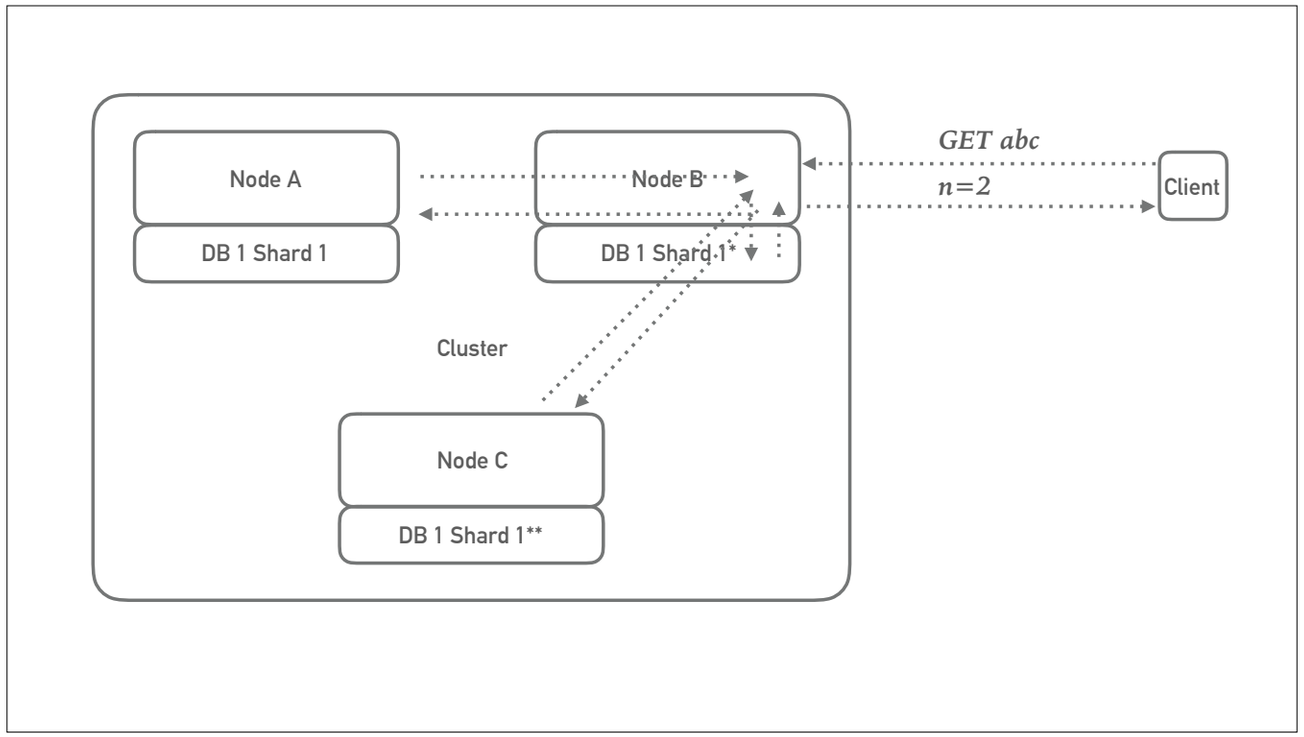

Node A
DB 1 Shard 1
Node B
DB 1 Shard 1*
Cluster
Node C
DB 1 Shard 1**

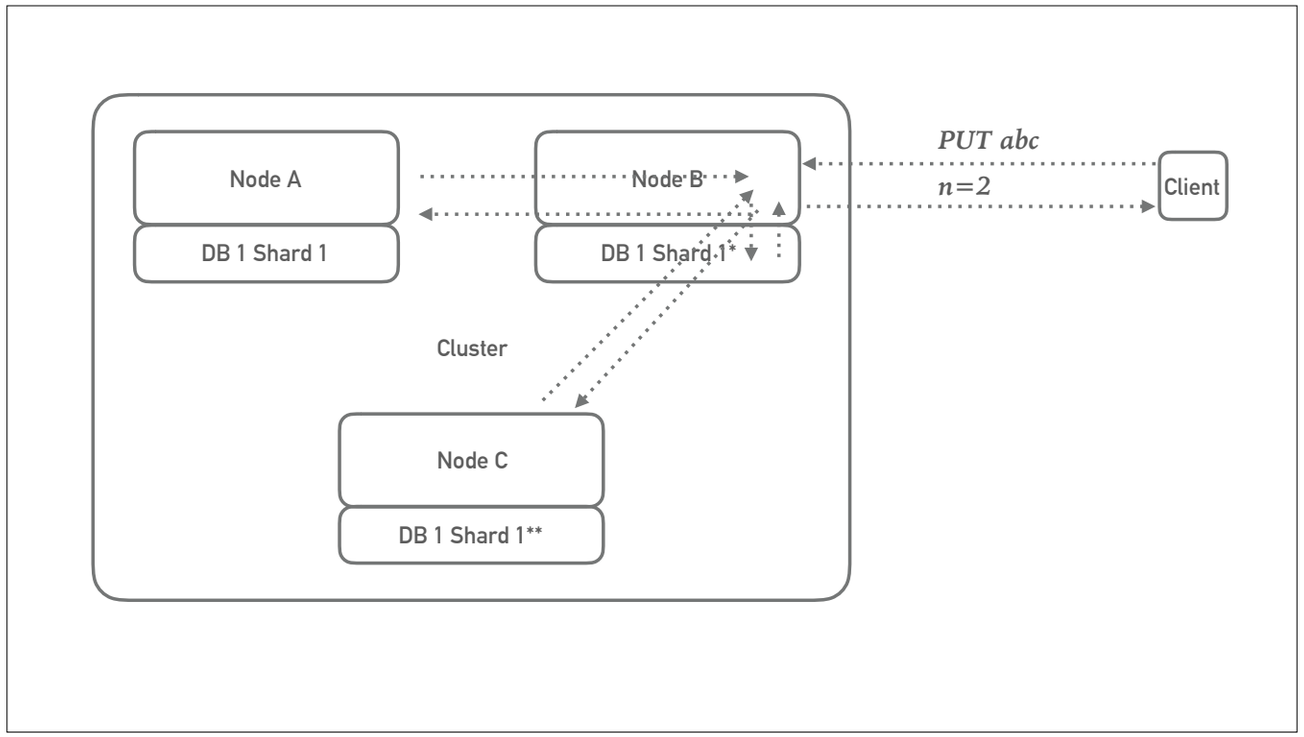

Node A
DB 1 Shard 1
Node B
DB 1 Shard 1*
Cluster
Node C
DB 1 Shard 1**
GET abc
n=2
Client

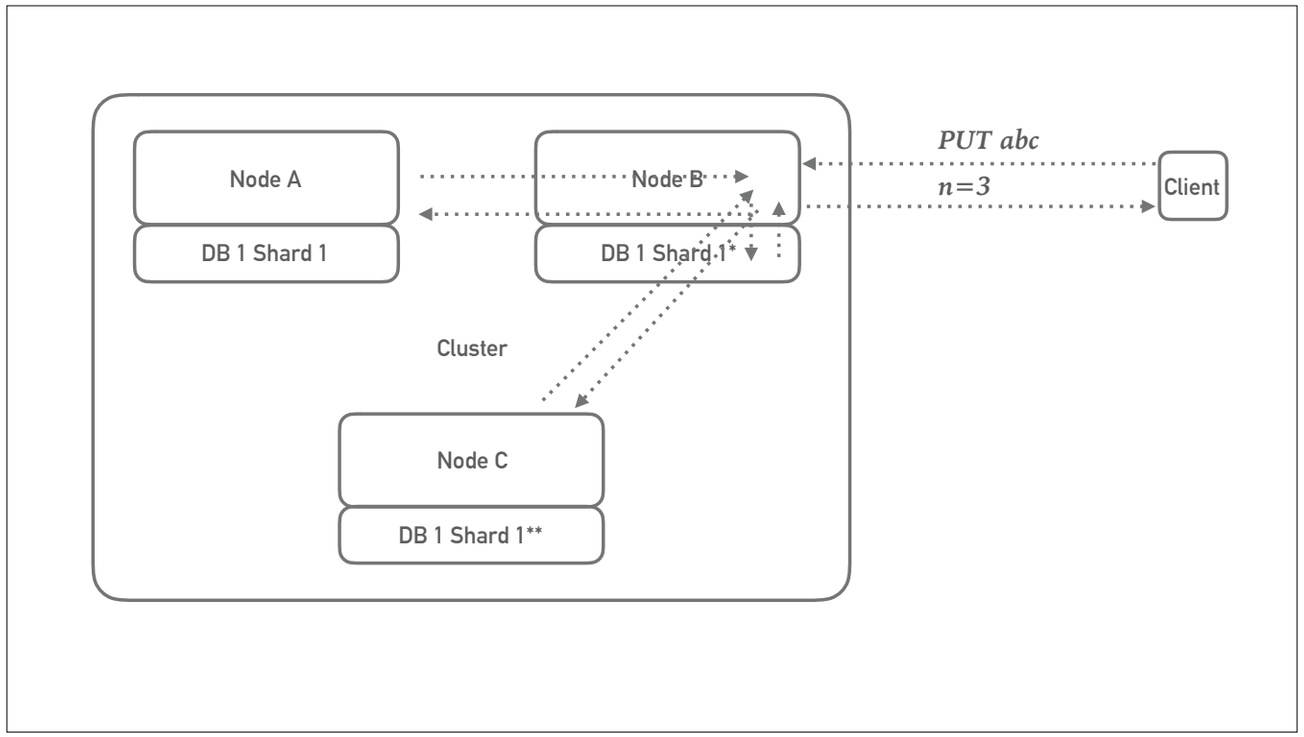

Node A
DB 1 Shard 1
Node B
DB 1 Shard 1*
Cluster
Node C
DB 1 Shard 1**
GET abc
n=2
Client


Node A
DB 1 Shard 1
Node B
DB 1 Shard 1*
Cluster
Node C
DB 1 Shard 1**
GET abc
n=2
Client


Node A
DB 1 Shard 1
Node B
DB 1 Shard 1*
Cluster
Node C
DB 1 Shard 1**
GET abc
n=2
Client

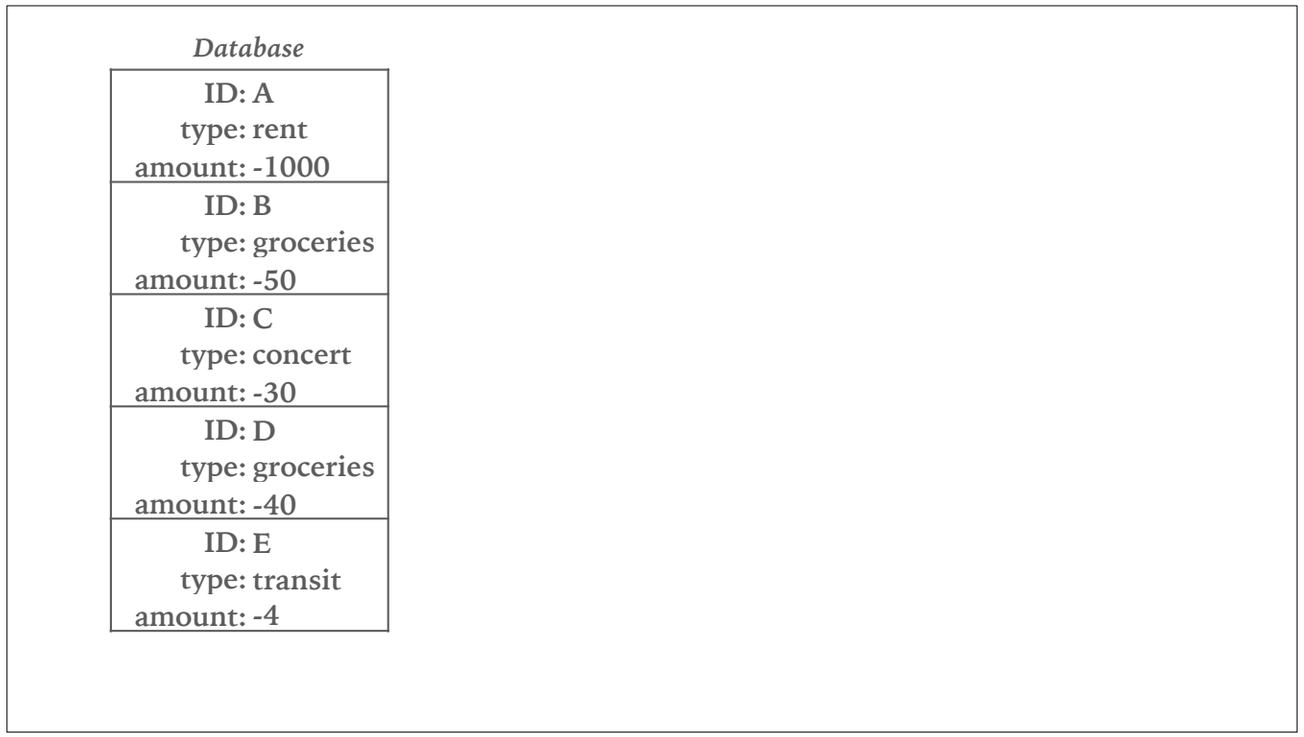

Node A
DB 1 Shard 1
Node B
DB 1 Shard 1*
GET abc
n=2
Client
Cluster
Node C
DB 1 Shard 1**


Node A
DB 1 Shard 1
Node B
DB 1 Shard 1*
PUT abc
n=2
Client
Cluster
Node C
DB 1 Shard 1**


Node A
DB 1 Shard 1
Node B
DB 1 Shard 1*
PUT abc
n=3
Client
Cluster
Node C
DB 1 Shard 1**

# INCREMENTAL,PERSISTENT MAP/REDUCE

- incremental, persistent M/R queries
    - single left-join possible
- Works in single node as well as cluster
- Mango query lang compiles to M/R




Map index is persisted to disk
Reduce is also persisted, and available grouped by key, and total, from the same index

| *Database* | *Map emit(type, amount)* |
| --- | --- |
| ID: A<br>type: rent<br>amount: -1000 | key: concert<br>value: -30 |
| ID: B<br>type: groceries<br>amount: -50 | key: groceries<br>value: -50 |
| ID: C<br>type: concert<br>amount: -30 | key: groceries<br>value: -40 |
| ID: D<br>type: groceries<br>amount: -40 | key: rent<br>value: -1000 |
| ID: E<br>type: transit<br>amount: -4 | key: transit<br>value: -4 |

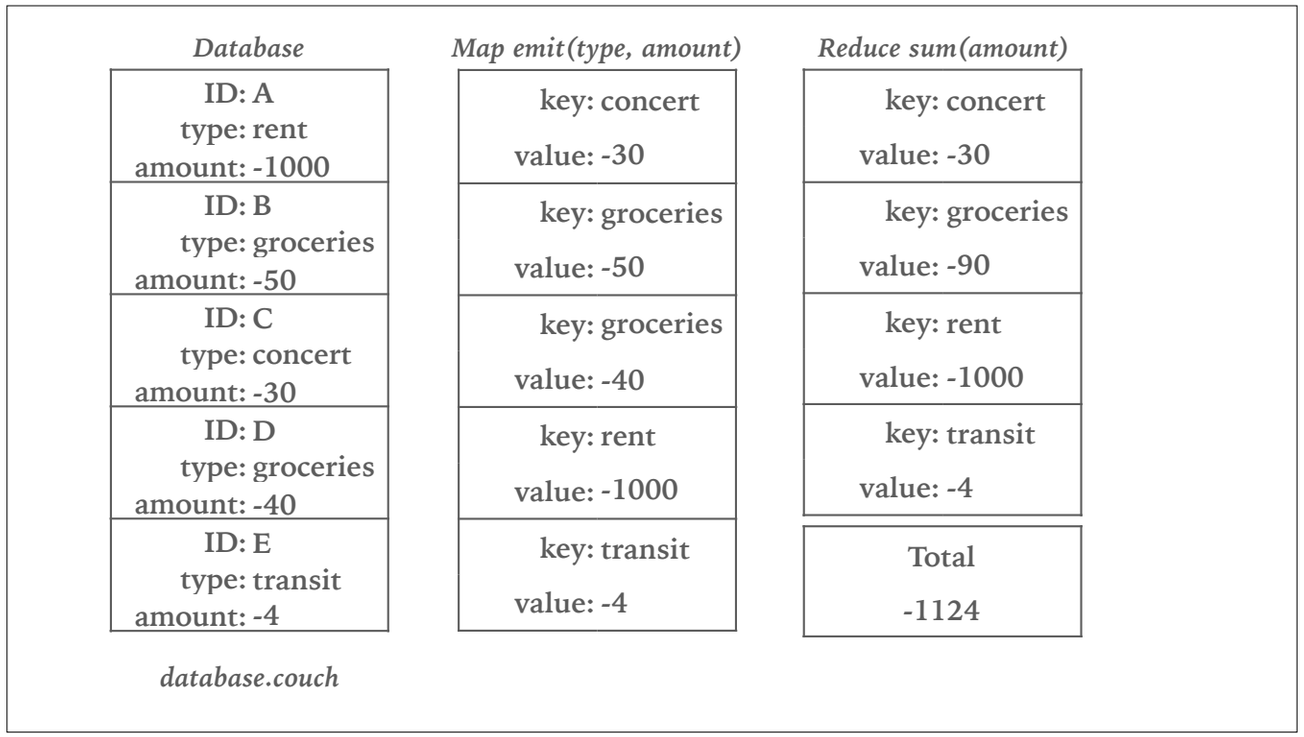
Map index is persisted to disk
Reduce is also persisted, and available grouped by key, and total, from the same index

| *Database* | *Map emit(type, amount)* | *Reduce sum(amount)* |
|---|---|---|
| ID: A<br>type: rent<br>amount: -1000 | key: concert<br>value: -30 | key: concert<br>value: -30 |
| ID: B<br>type: groceries<br>amount: -50 | key: groceries<br>value: -50 | key: groceries<br>value: -90 |
| ID: C<br>type: concert<br>amount: -30 | key: groceries<br>value: -40 | key: rent<br>value: -1000 |
| ID: D<br>type: groceries<br>amount: -40 | key: rent<br>value: -1000 | key: transit<br>value: -4 |
| ID: E<br>type: transit<br>amount: -4 | key: transit<br>value: -4 | |

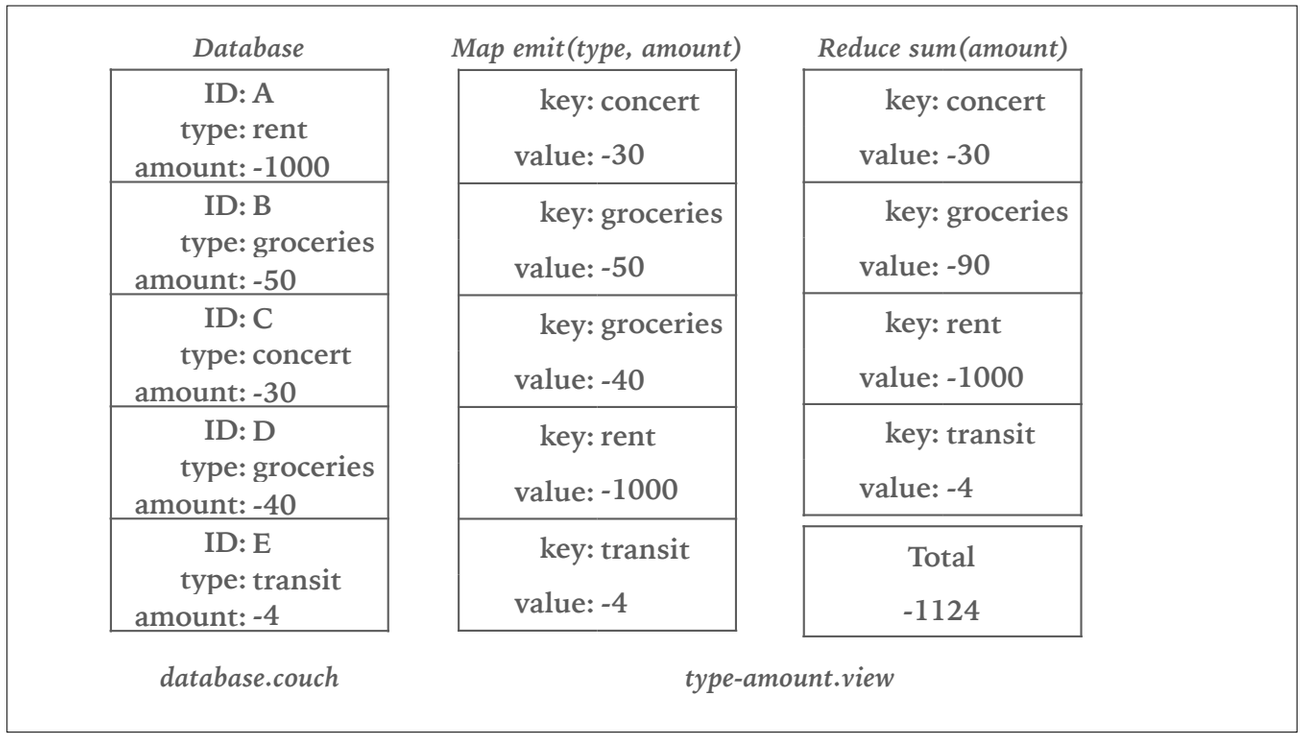
Map index is persisted to disk
Reduce is also persisted, and available grouped by key, and total, from the same index

| *Database* | *Map emit(type, amount)* | *Reduce sum(amount)* |
| --- | --- | --- |
| ID: A<br>type: rent<br>amount: -1000 | key: concert<br>value: -30 | key: concert<br>value: -30 |
| ID: B<br>type: groceries<br>amount: -50 | key: groceries<br>value: -50 | key: groceries<br>value: -90 |
| ID: C<br>type: concert<br>amount: -30 | key: groceries<br>value: -40 | key: rent<br>value: -1000 |
| ID: D<br>type: groceries<br>amount: -40 | key: rent<br>value: -1000 | key: transit<br>value: -4 |
| ID: E<br>type: transit<br>amount: -4 | key: transit<br>value: -4 | Total<br>-1124 |

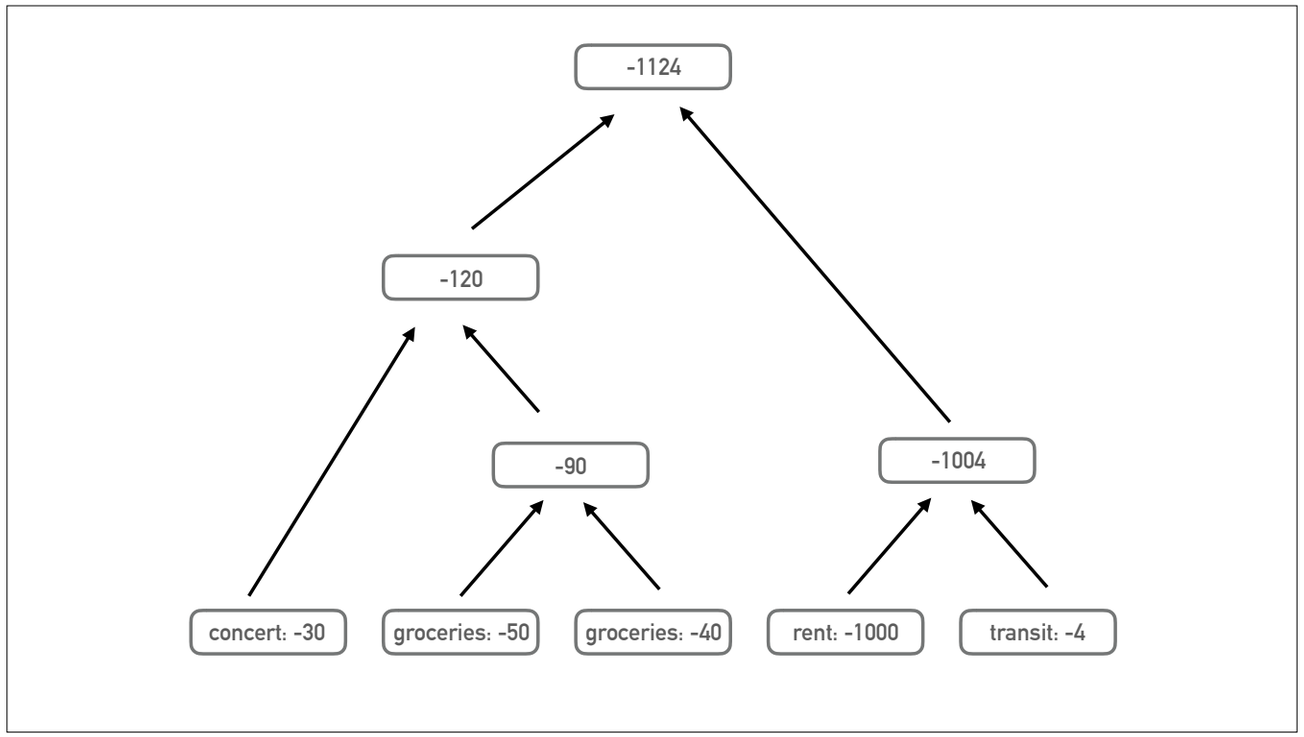
Map index is persisted to disk
Reduce is also persisted, and available grouped by key, and total, from the same index

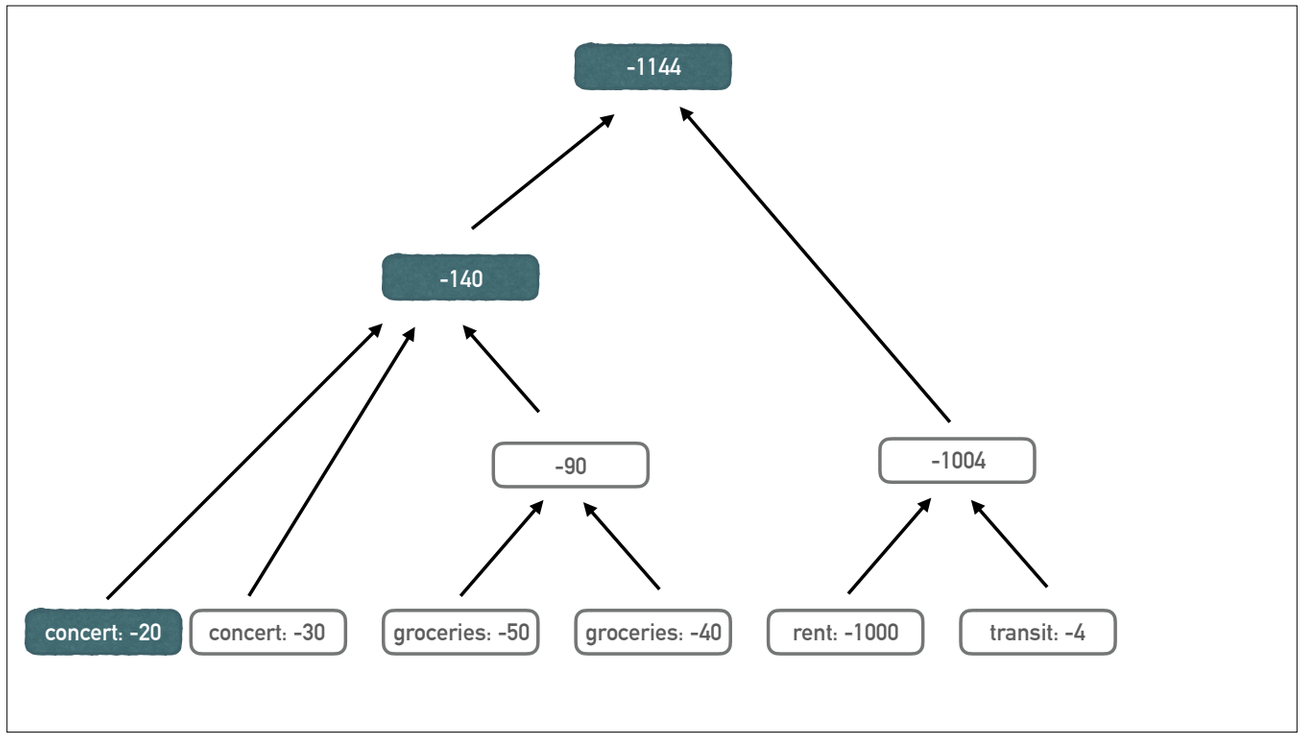



Map index is persisted to disk
Reduce is also persisted, and available grouped by key, and total, from the same index




Map index is persisted to disk
Reduce is also persisted, and available grouped by key, and total, from the same index


-1124
-120
-90
-1004
concert: -30
groceries: -50
groceries: -40
rent: -1000
transit: -4




B+tree, shallow: updates very efficient, only very few nodes need touching

# MANGO QUERY LANGUAGE

## MANGO QUERY LANGUAGE

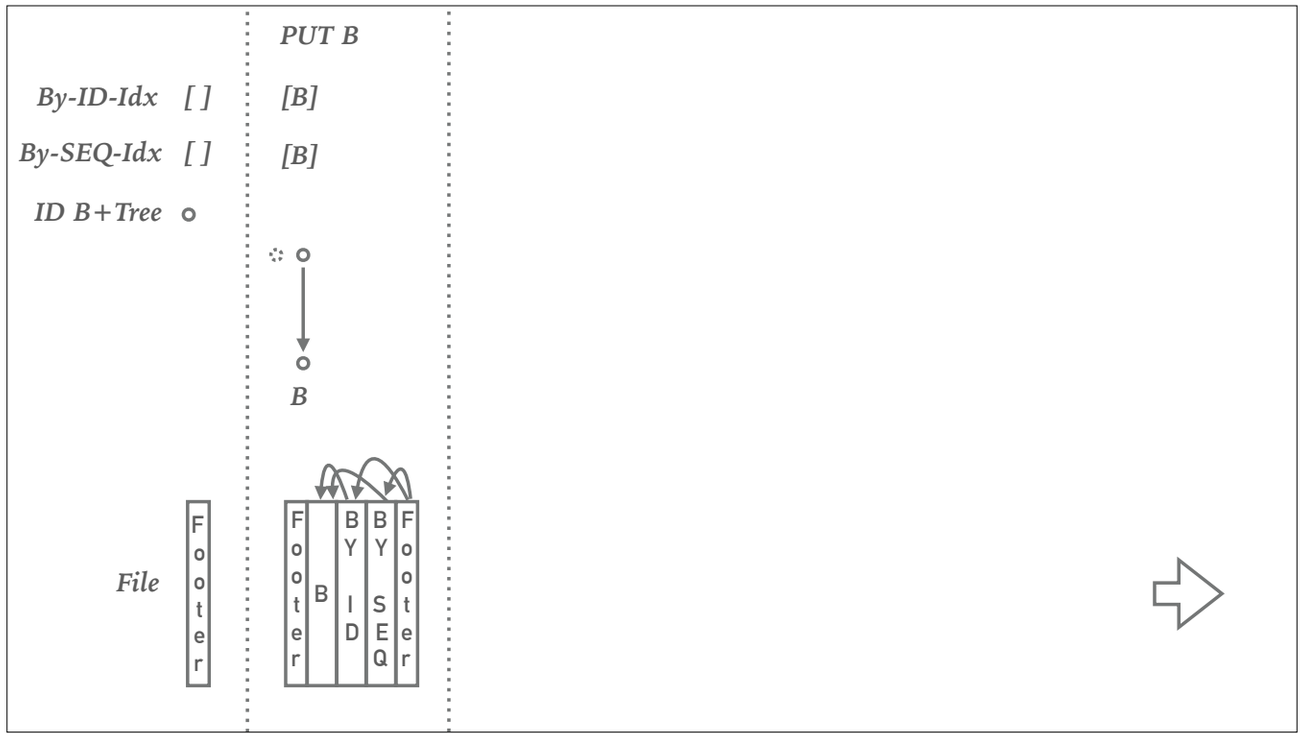
- Compiles to Map / Reduce

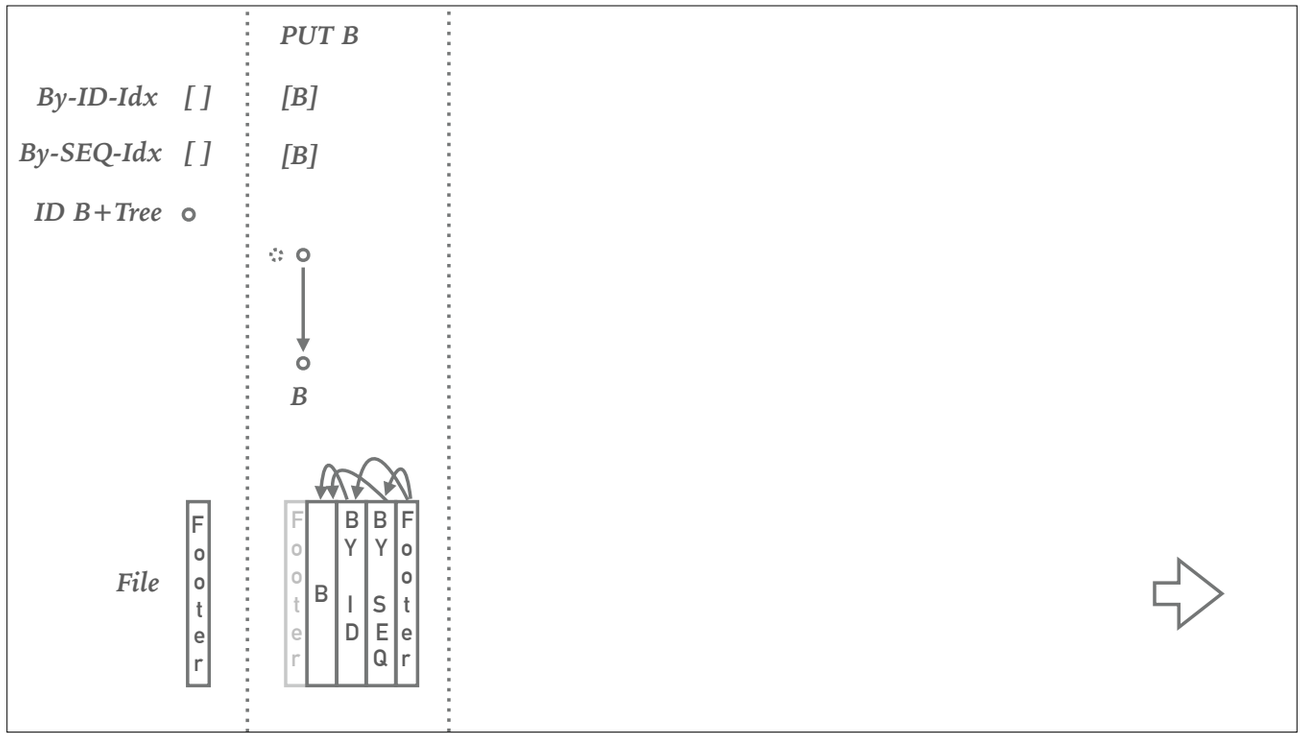
```
{
    "selector": {
        "year": {"$gt": 2010}
    },
    "fields": ["_id", "_rev", "year", "title"],
    "sort": [{"year": "asc"}],
    "limit": 2,
    "skip": 0
}
```

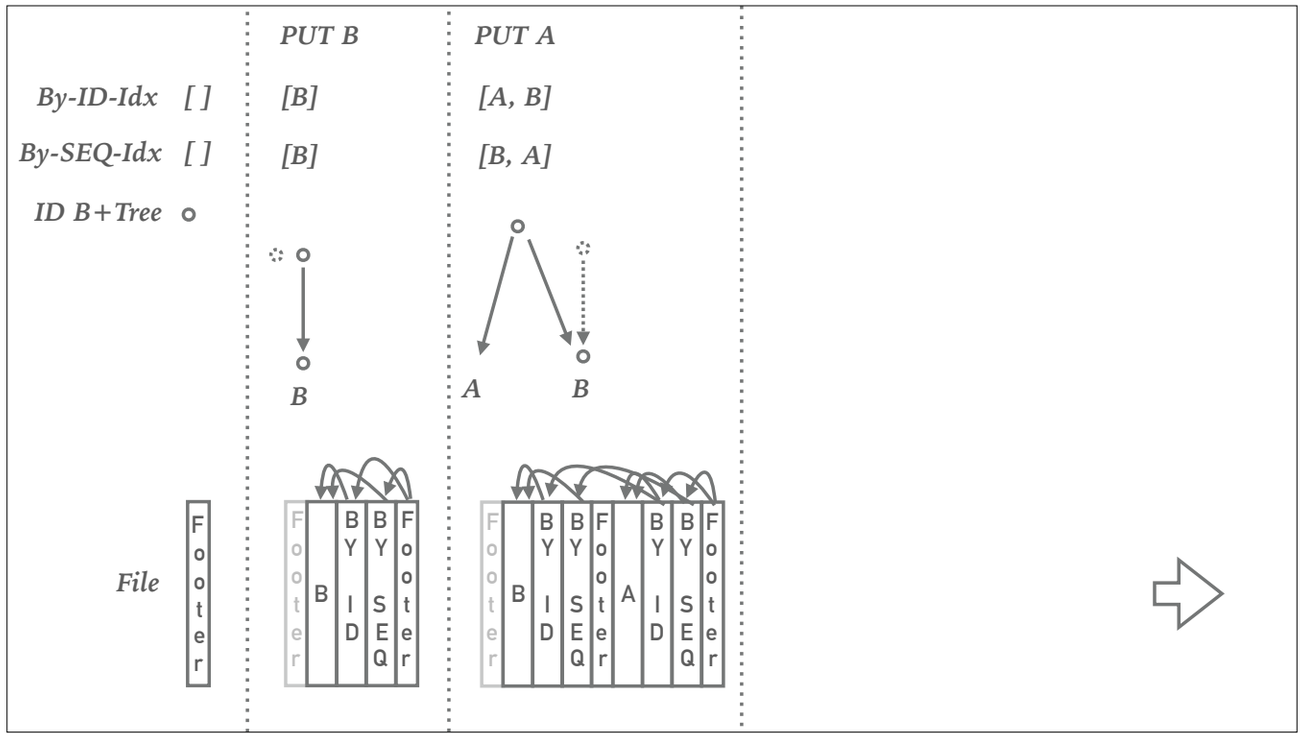
# RELIABLE DATA STORAGE

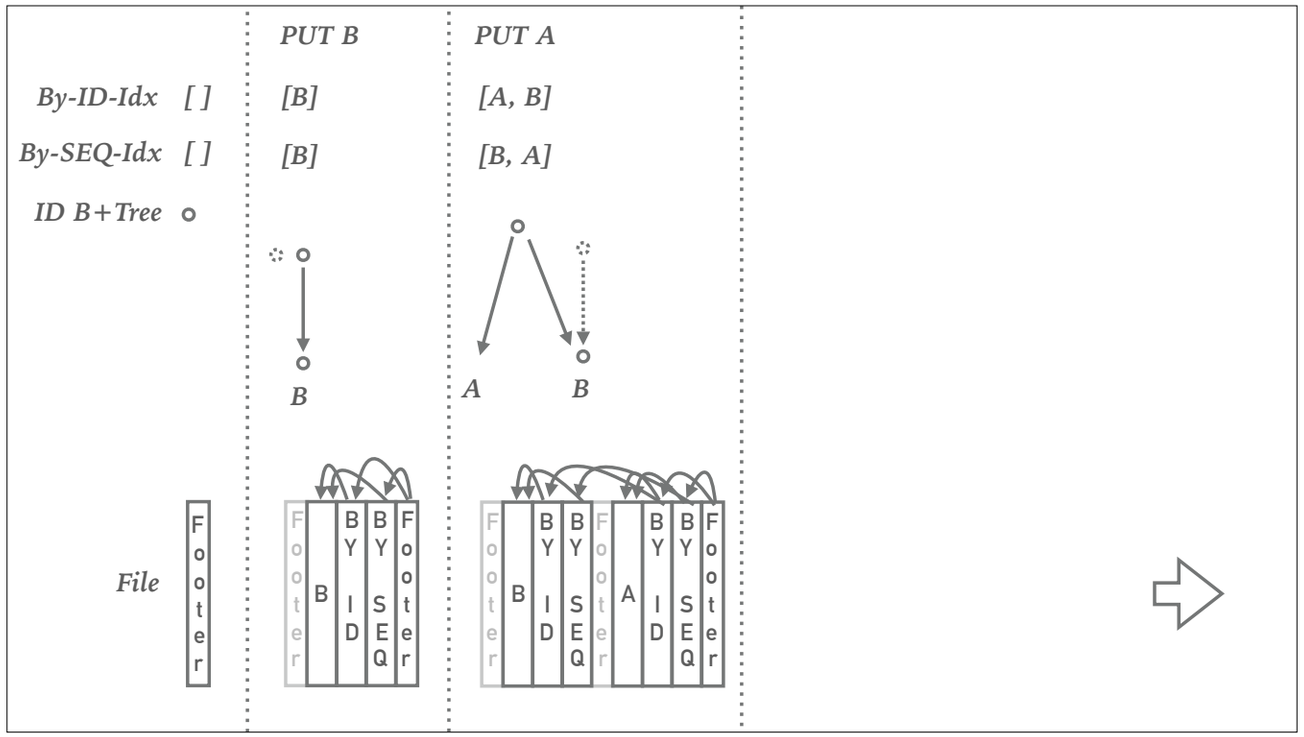
Reliable Data Storage
- Append only files for storage and index
- Data committed to disk is never touched again
    - no partial updates, that cause inconsistencies during catastrophic events
    - no need for repairs
    - instant startup
- downside: compaction / garbage collection / vacuum
    - can run online
    - in v1: simplest possible, copy live data, swap files
        - takes iops away from live traffic
        - hogs FS block cache
    - in v2
        - runs in io “background”
            - takes longer, but doesn’t take live ops away
        - compaction by shard, still hogs FS block cache, but only per shard
        - more compact, by clustering indexes inside file

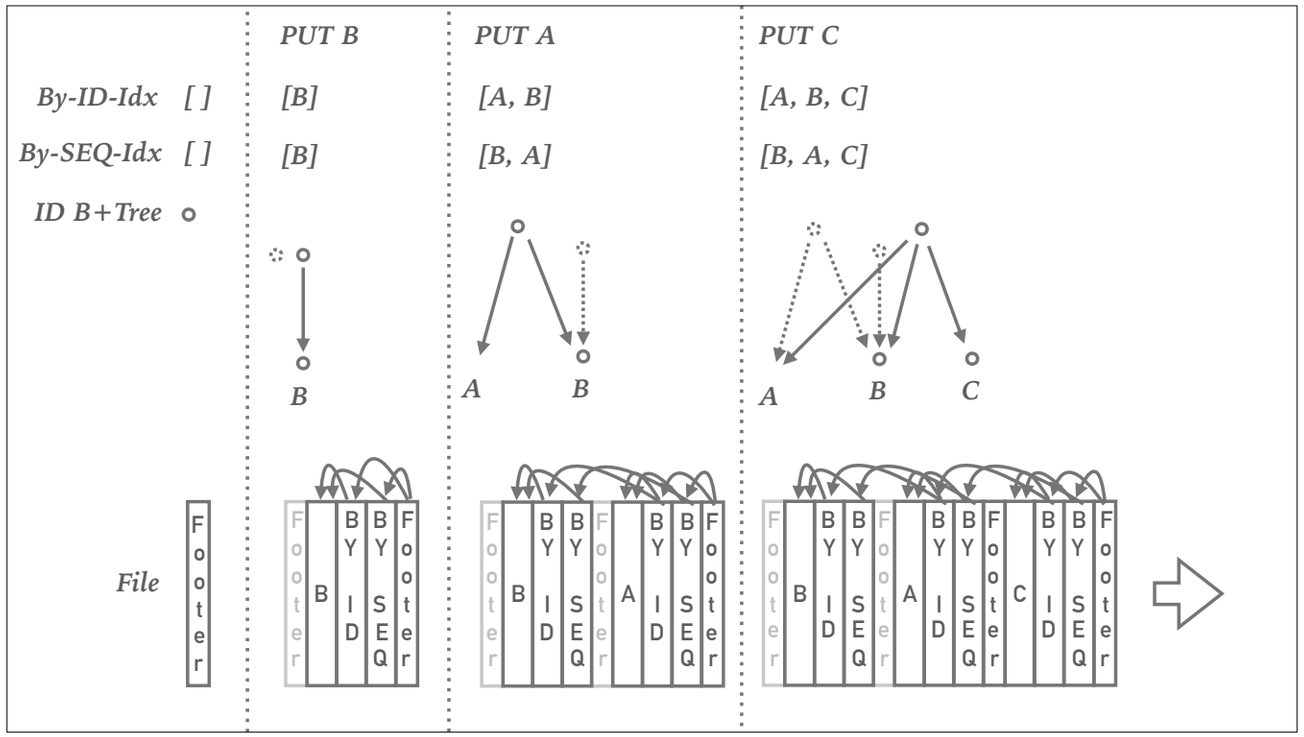

By-ID-Idx [ ]
By-SEQ-Idx [ ]
ID B+Tree
File
Footer

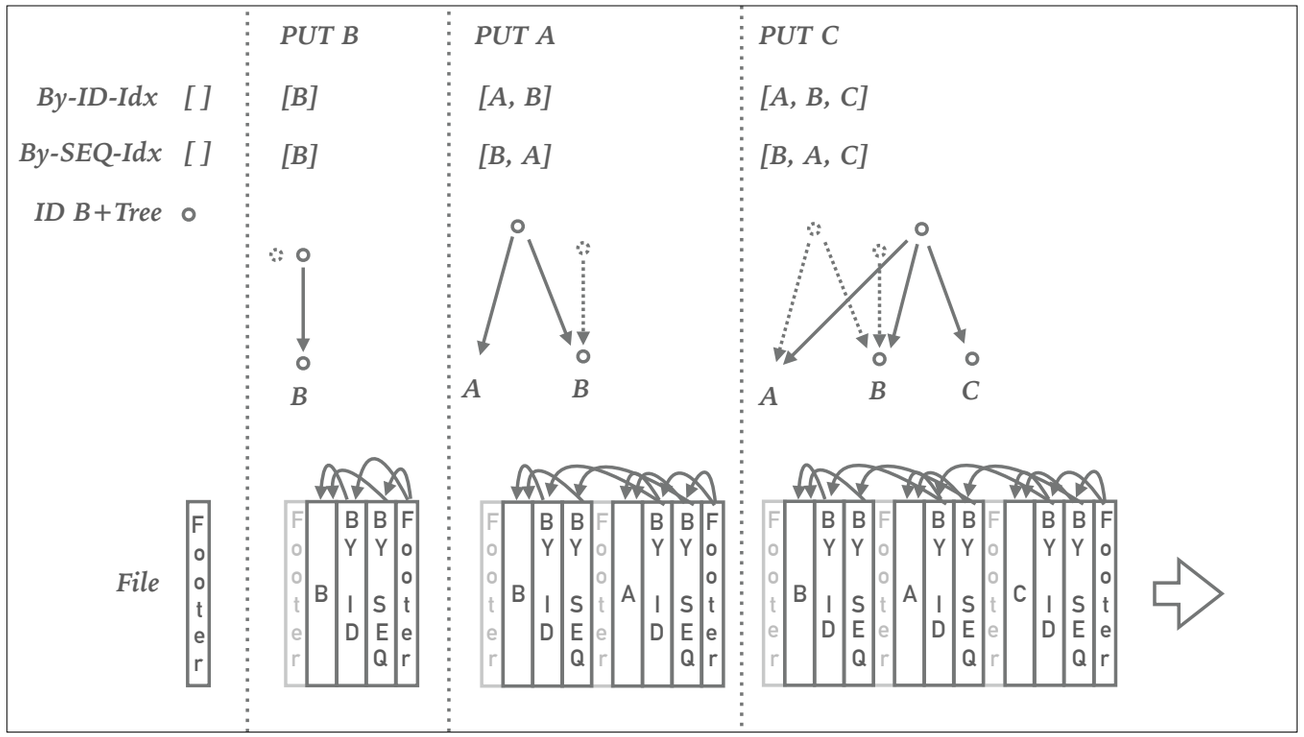

PUT B
By-ID-Idx [ ]
[B]
By-SEQ-Idx [ ]
[B]
ID B+Tree
B
File
Footer
Footer
B
BY ID
BY SEQ
Footer

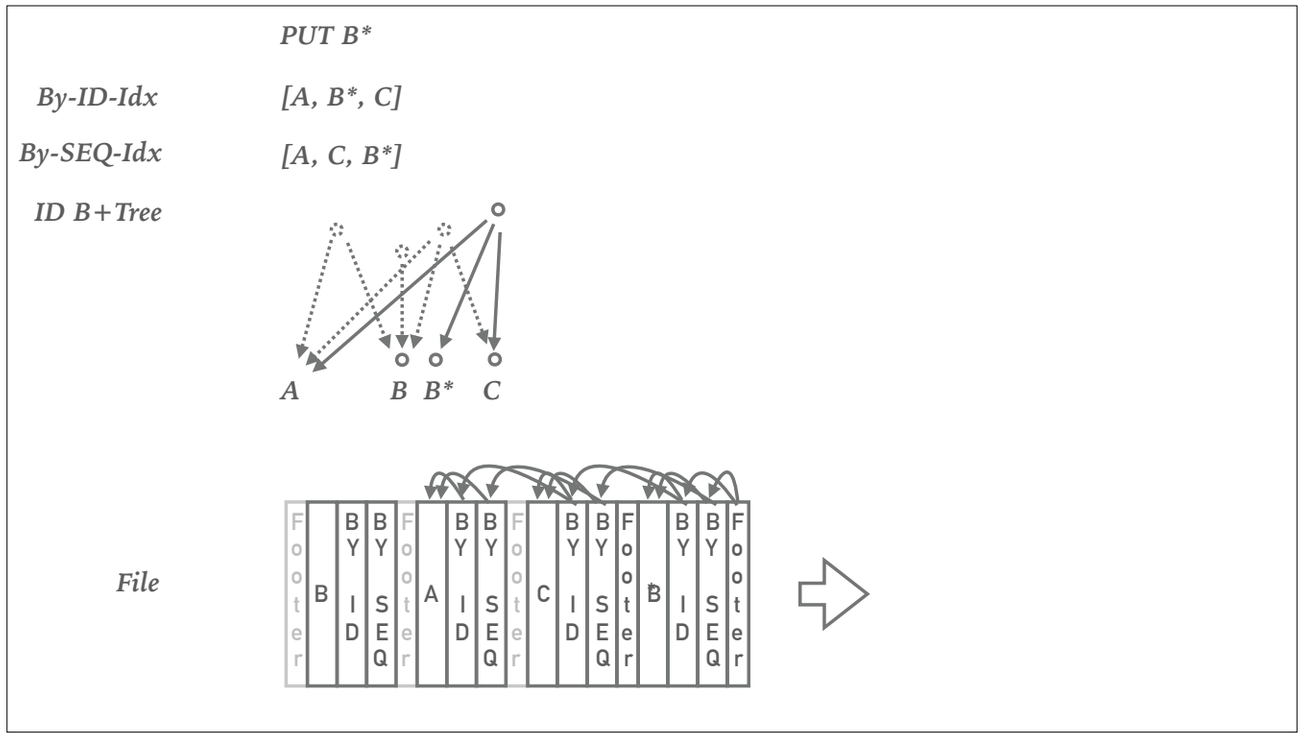

PUT B
By-ID-Idx [ ]
[B]
By-SEQ-Idx [ ]
[B]
ID B+Tree
B
File
Footer
Footer
B
BY ID
BY SEQ
Footer

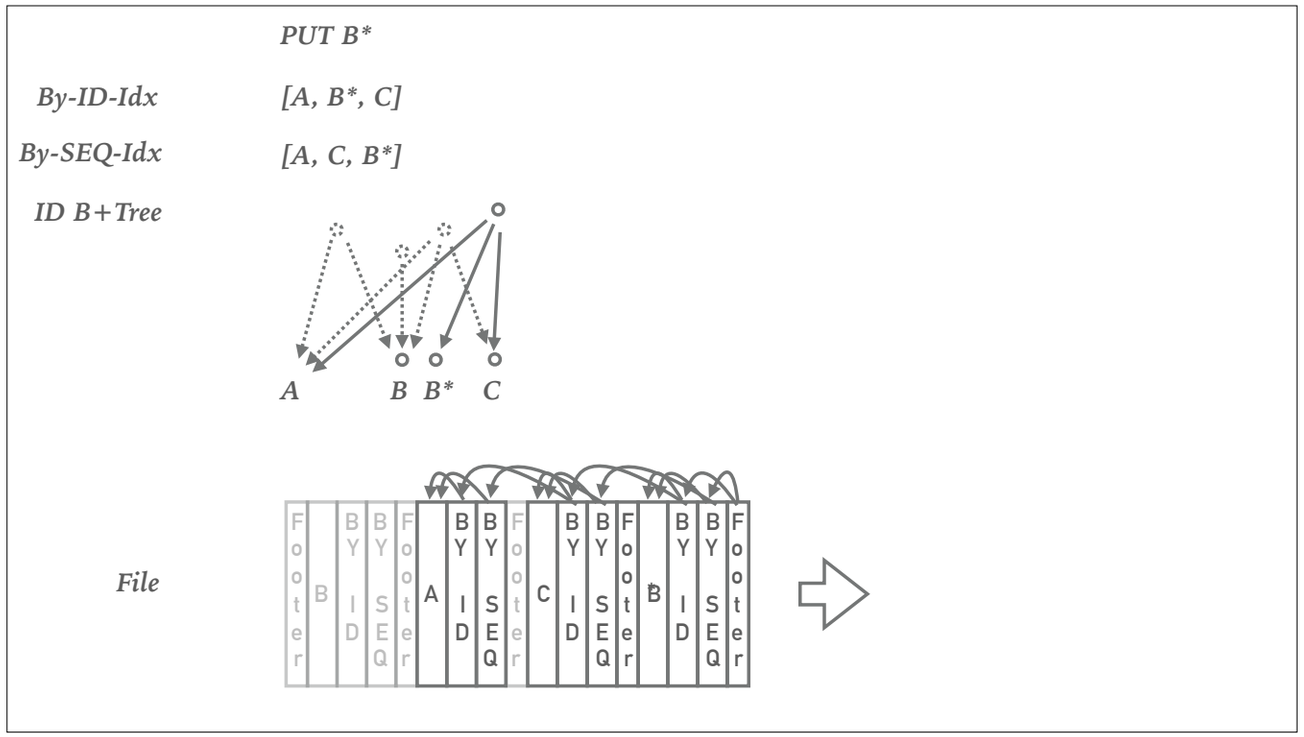

PUT B
PUT A
By-ID-Idx [ ]
[B]
[A, B]
By-SEQ-Idx [ ]
[B]
[B, A]
ID B+Tree
B
A
B
File
Footer
Footer
B
BY ID
BY SEQ
Footer
Footer
B
BY ID
BY SEQ
Footer
A
BY ID
BY SEQ
Footer


PUT B
PUT A
By-ID-Idx [ ]
[B]
[A, B]
By-SEQ-Idx [ ]
[B]
[B, A]
ID B+Tree
B
A
B
File
Footer
Footer
B
BY ID
BY SEQ
Footer
Footer
B
BY ID
BY SEQ
Footer
A
BY ID
BY SEQ
Footer

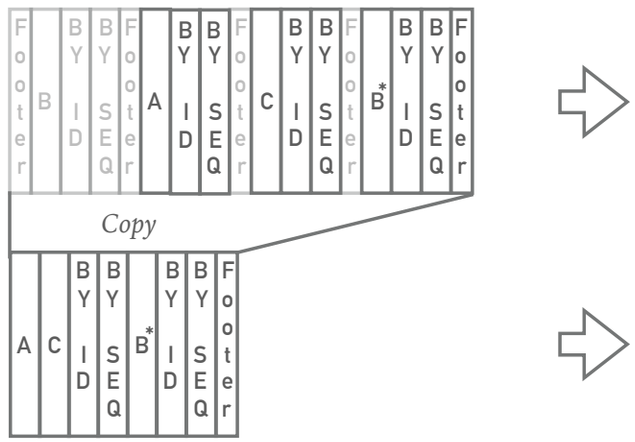

PUT B
PUT A
PUT C
By-ID-Idx [ ]
[B]
[A, B]
[A, B, C]
By-SEQ-Idx [ ]
[B]
[B, A]
[B, A, C]
ID B+Tree
B
A
B
A
B
C
File
Footer
Footer
B
BY ID
BY SEQ
Footer
Footer
B
BY ID
BY SEQ
Footer
A
BY ID
BY SEQ
Footer
Footer
B
BY ID
BY SEQ
Footer
A
BY ID
BY SEQ
Footer
C
BY ID
BY SEQ
Footer


PUT B
PUT A
PUT C
By-ID-Idx [ ]
[B]
[A, B]
[A, B, C]
By-SEQ-Idx [ ]
[B]
[B, A]
[B, A, C]
ID B+Tree
B
A
B
A
B
C
File
Footer
Footer
B
BY ID
BY SEQ
Footer
Footer
B
BY ID
BY SEQ
Footer
A
BY ID
BY SEQ
Footer
Footer
B
BY ID
BY SEQ
Footer
A
BY ID
BY SEQ
Footer
C
BY ID
BY SEQ
Footer

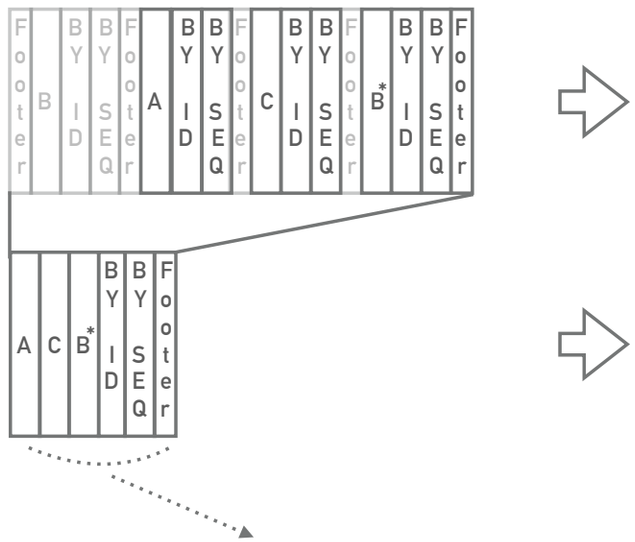

PUT B*
By-ID-Idx
[A, B*, C]
By-SEQ-Idx
[A, C, B*]
ID B+Tree
A
B
B*
C
File
Footer
B
BY ID
BY SEQ
Footer
A
BY ID
BY SEQ
Footer
C
BY ID
BY SEQ
Footer
B*
BY ID
BY SEQ
Footer


PUT B*
By-ID-Idx
[A, B*, C]
By-SEQ-Idx
[A, C, B*]
ID B+Tree
A
B
B*
C
File
Footer
B
BY ID
BY SEQ
Footer
A
BY ID
BY SEQ
Footer
C
BY ID
BY SEQ
Footer
B*
BY ID
BY SEQ
Footer

# COMPACTION V1

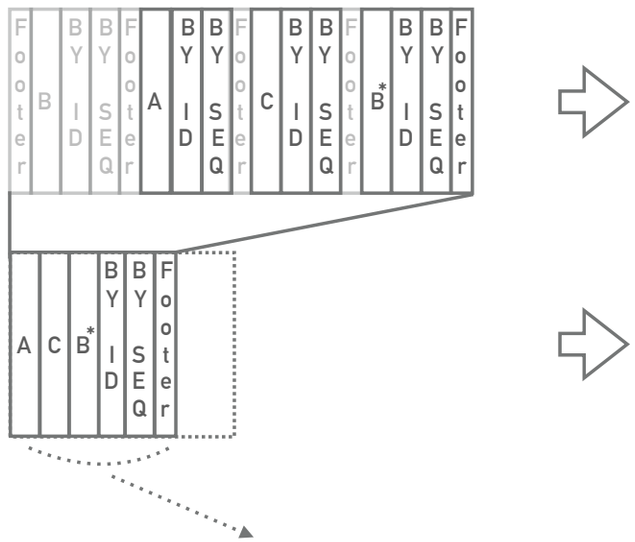


# COMPACTION V2




*Smaller, Indexes clustered, background i/o*

# COMPACTION V2




*Smaller, Indexes clustered, background i/o*

# CASE-STUDIES

# Case Studies (maybe splice into use-cases)
IBM/Cloudant: Big Data as a Service
eEhealth Ebola / Hospital Run / RapidFTR (Family Tracing and Reunification UNICEF / Primero)
if-control: avalanche protection inspection
Industry of things

## CASE STUDIES

# Case Studies (maybe splice into use-cases)
IBM/Cloudant: Big Data as a Service
eEhealth Ebola / Hospital Run / RapidFTR (Family Tracing and Reunification UNICEF / Primero)
if-control: avalanche protection inspection
Industry of things: 14% of all industrial facilities (think oil refinery, plant, etc) are networked, and < 20% **of those** are connected to the public internet.

## CASE STUDIES

- IBM/Cloudant: Big Data as a Service

# Case Studies (maybe splice into use-cases)
IBM/Cloudant: Big Data as a Service
eEhealth Ebola / Hospital Run / RapidFTR (Family Tracing and Reunification UNICEF / Primero)
if-control: avalanche protection inspection
Industry of things: 14% of all industrial facilities (think oil refinery, plant, etc) are networked, and < 20% **of those** are connected to the public internet.

## CASE STUDIES

- IBM/Cloudant: Big Data as a Service
- eHealth Africa: Fighting Ebola with CouchDB and PouchDB

# Case Studies (maybe splice into use-cases)
IBM/Cloudant: Big Data as a Service
eEhealth Ebola / Hospital Run / RapidFTR (Family Tracing and Reunification UNICEF / Primero)
if-control: avalanche protection inspection
Industry of things: 14% of all industrial facilities (think oil refinery, plant, etc) are networked, and < 20% **of those** are connected to the public internet.

## CASE STUDIES

- IBM/Cloudant: Big Data as a Service
- eHealth Africa: Fighting Ebola with CouchDB and PouchDB
- Hospital Run: Hospital management software for regions with limited infrastructure

# Case Studies (maybe splice into use-cases)
IBM/Cloudant: Big Data as a Service
eEhealth Ebola / Hospital Run / RapidFTR (Family Tracing and Reunification UNICEF / Primero)
if-control: avalanche protection inspection
Industry of things: 14% of all industrial facilities (think oil refinery, plant, etc) are networked, and < 20% **of those** are connected to the public internet.

## CASE STUDIES

- IBM/Cloudant: Big Data as a Service
- eHealth Africa: Fighting Ebola with CouchDB and PouchDB
- Hospital Run: Hospital management software for regions with limited infrastructure
- RapidFTR: Family Tracing & Reunification for regions in crisis (e.g. Haiti) / UNICEF

# Case Studies (maybe splice into use-cases)
IBM/Cloudant: Big Data as a Service
eEhealth Ebola / Hospital Run / RapidFTR (Family Tracing and Reunification UNICEF / Primero)
if-control: avalanche protection inspection
Industry of things: 14% of all industrial facilities (think oil refinery, plant, etc) are networked, and < 20% **of those** are connected to the public internet.

## CASE STUDIES

- IBM/Cloudant: Big Data as a Service
- eHealth Africa: Fighting Ebola with CouchDB and PouchDB
- Hospital Run: Hospital management software for regions with limited infrastructure
- RapidFTR: Family Tracing & Reunification for regions in crisis (e.g. Haiti) / UNICEF
- Decisions for Heroes: Coast guard mobile operations center

# Case Studies (maybe splice into use-cases)
IBM/Cloudant: Big Data as a Service
eEhealth Ebola / Hospital Run / RapidFTR (Family Tracing and Reunification UNICEF / Primero)
if-control: avalanche protection inspection
Industry of things: 14% of all industrial facilities (think oil refinery, plant, etc) are networked, and < 20% **of those** are connected to the public internet.

## CASE STUDIES

- IBM/Cloudant: Big Data as a Service
- eHealth Africa: Fighting Ebola with CouchDB and PouchDB
- Hospital Run: Hospital management software for regions with limited infrastructure
- RapidFTR: Family Tracing & Reunification for regions in crisis (e.g. Haiti) / UNICEF
- Decisions for Heroes: Coast guard mobile operations center
- Avalanche protection inspection in the Swiss Alps

# Case Studies (maybe splice into use-cases)
IBM/Cloudant: Big Data as a Service
eEhealth Ebola / Hospital Run / RapidFTR (Family Tracing and Reunification UNICEF / Primero)
if-control: avalanche protection inspection
Industry of things: 14% of all industrial facilities (think oil refinery, plant, etc) are networked, and < 20% **of those** are connected to the public internet.

## CASE STUDIES

- IBM/Cloudant: Big Data as a Service
- eHealth Africa: Fighting Ebola with CouchDB and PouchDB
- Hospital Run: Hospital management software for regions with limited infrastructure
- RapidFTR: Family Tracing & Reunification for regions in crisis (e.g. Haiti) / UNICEF
- Decisions for Heroes: Coast guard mobile operations center
- Avalanche protection inspection in the Swiss Alps
- Industry of things

# Case Studies (maybe splice into use-cases)
IBM/Cloudant: Big Data as a Service
eEhealth Ebola / Hospital Run / RapidFTR (Family Tracing and Reunification UNICEF / Primero)
if-control: avalanche protection inspection
Industry of things: 14% of all industrial facilities (think oil refinery, plant, etc) are networked, and < 20% **of those** are connected to the public internet.

# THANK YOU!

*Introducing Apache CouchDB 2.0*
*Jan Lehnardt @janl jan@apache.org*
*Professional Support for Apache CouchDB: https://neighbourhood.ie*