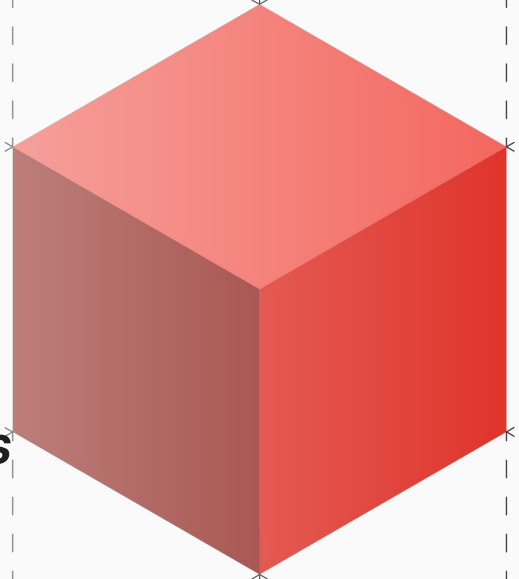
TiDB

TiDB: under the hood

An introduction to distributed databases



NLUUG vj 2025, Utrecht, Daniël van Eeden

AGENDA

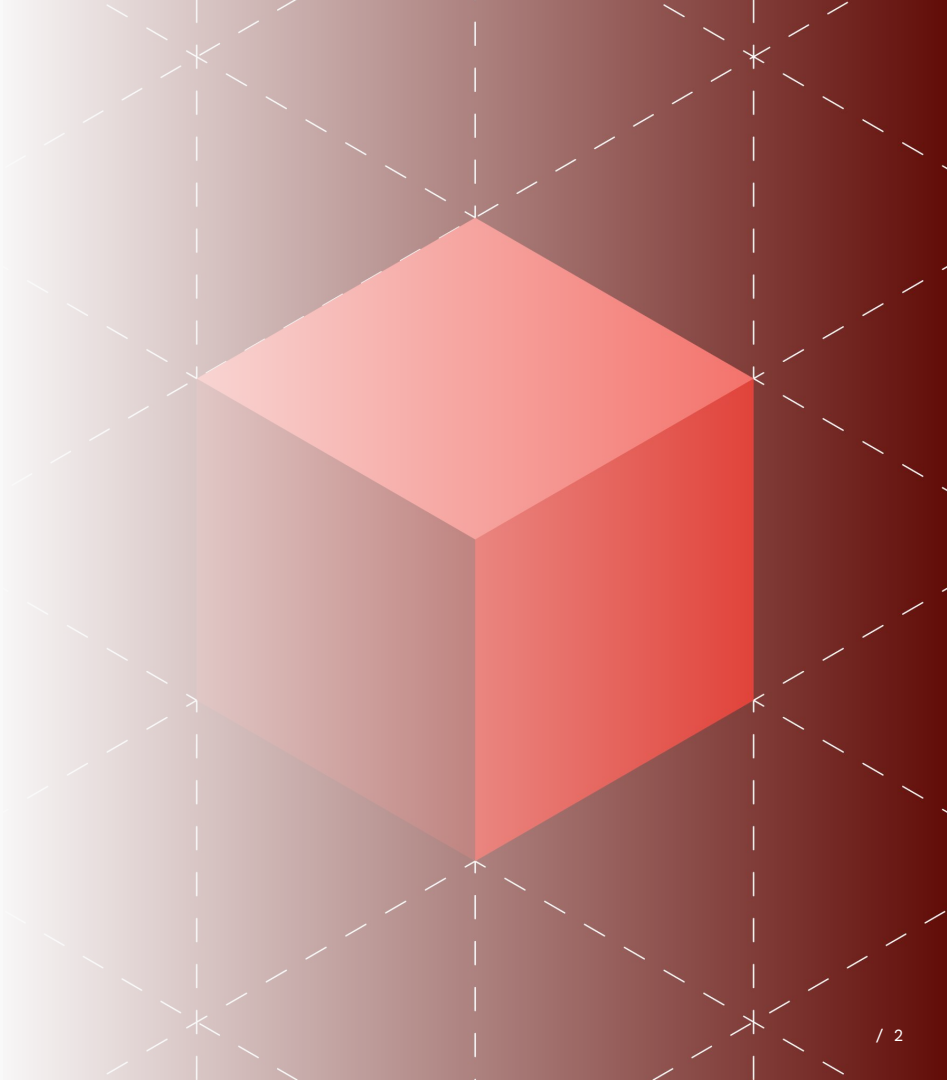
01 Introduction

02 History

03 Distributed databases

04 Building blocks

05 Conclusion



Daniël van Eeden

Working for **PingCAP**, the company behind **TiDB**.

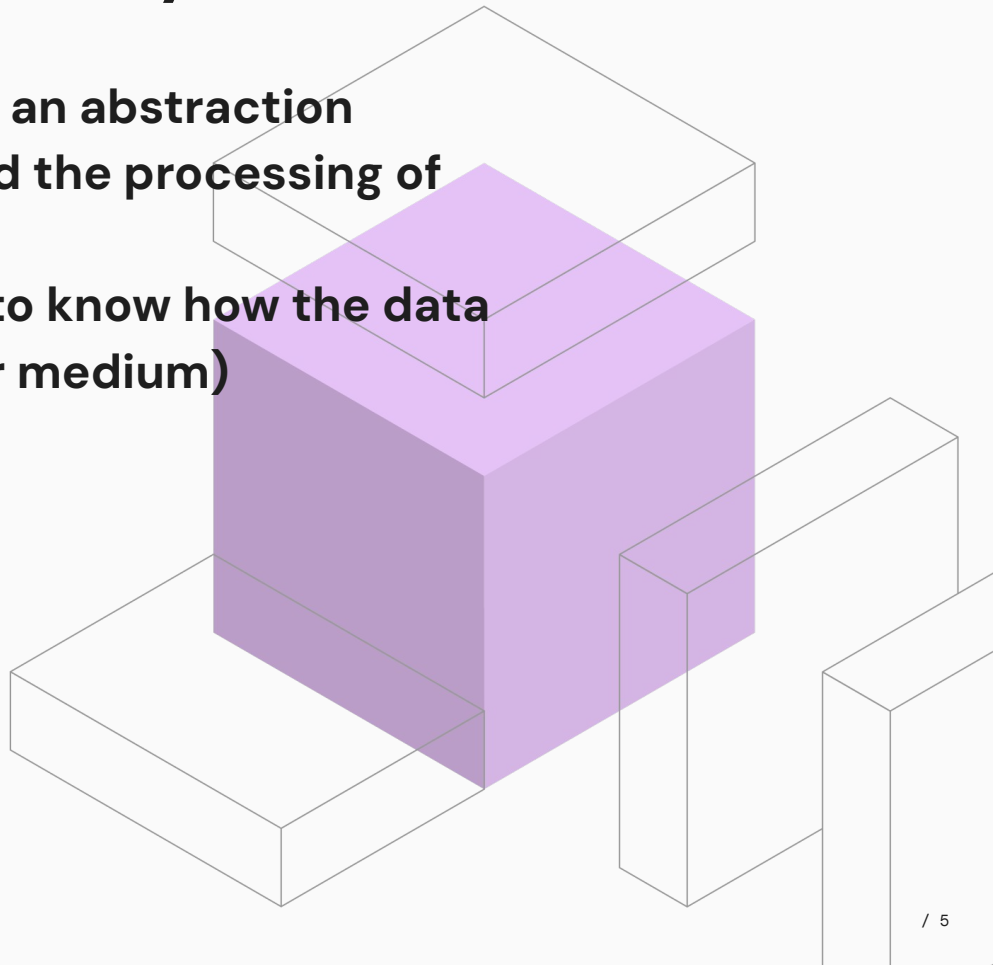
Previously working as a MySQL DBA for Booking.com

Active in the open source community as contributor to: DBD::mysql, go-mysql, Wireshark, TiDB, MySQL, and more.

02 History

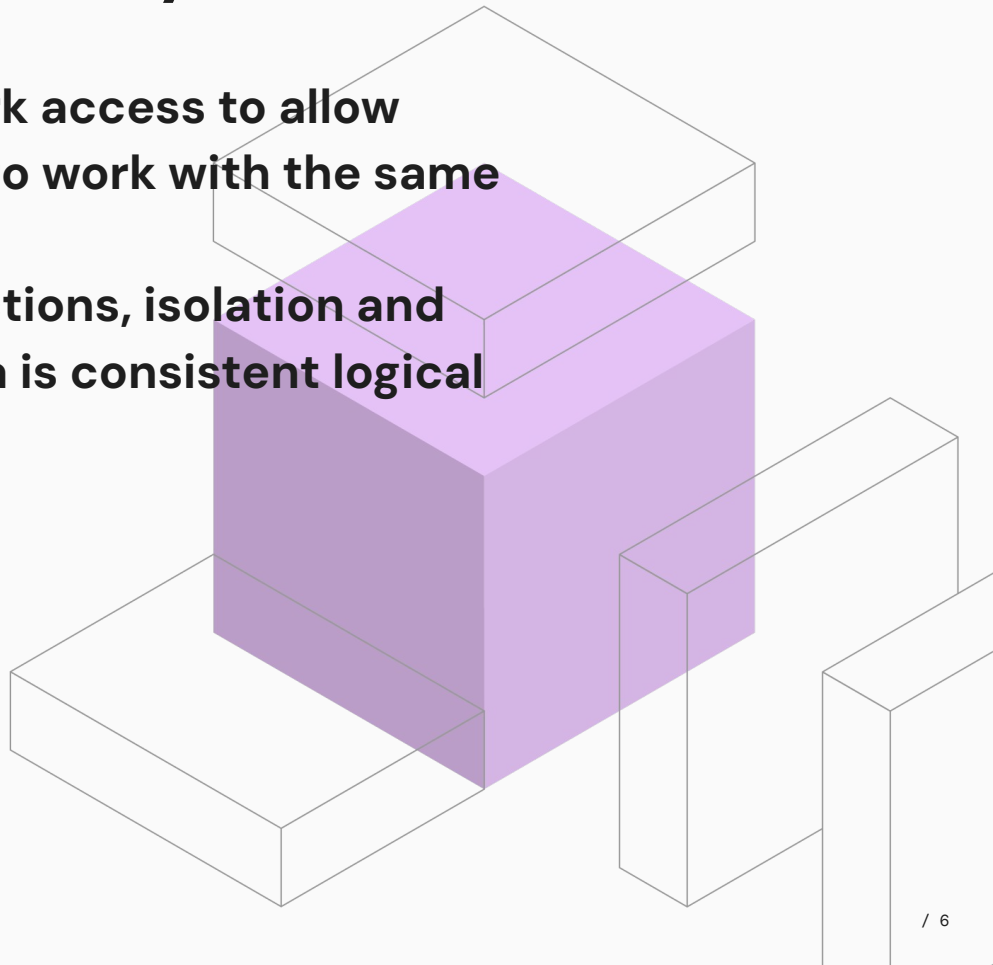
Let's start with some history

- Databases were created to give an abstraction between the storage of data and the processing of data.
- Applications no longer needed to know how the data was stored on disk (or any other medium)



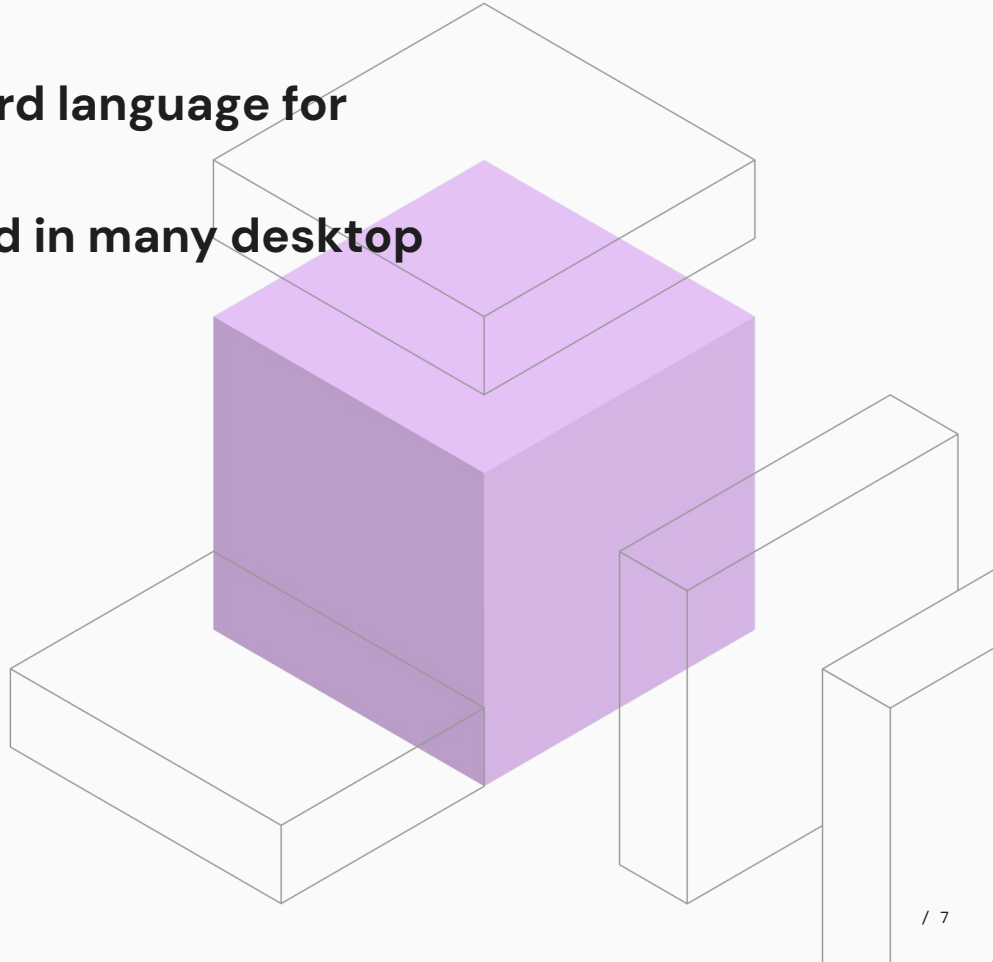
Let's start with some history

- Most databases provide network access to allow multiple application instances to work with the same dataset.
- This brings the need for transactions, isolation and atomicity to make sure the data is consistent logical and physical state.



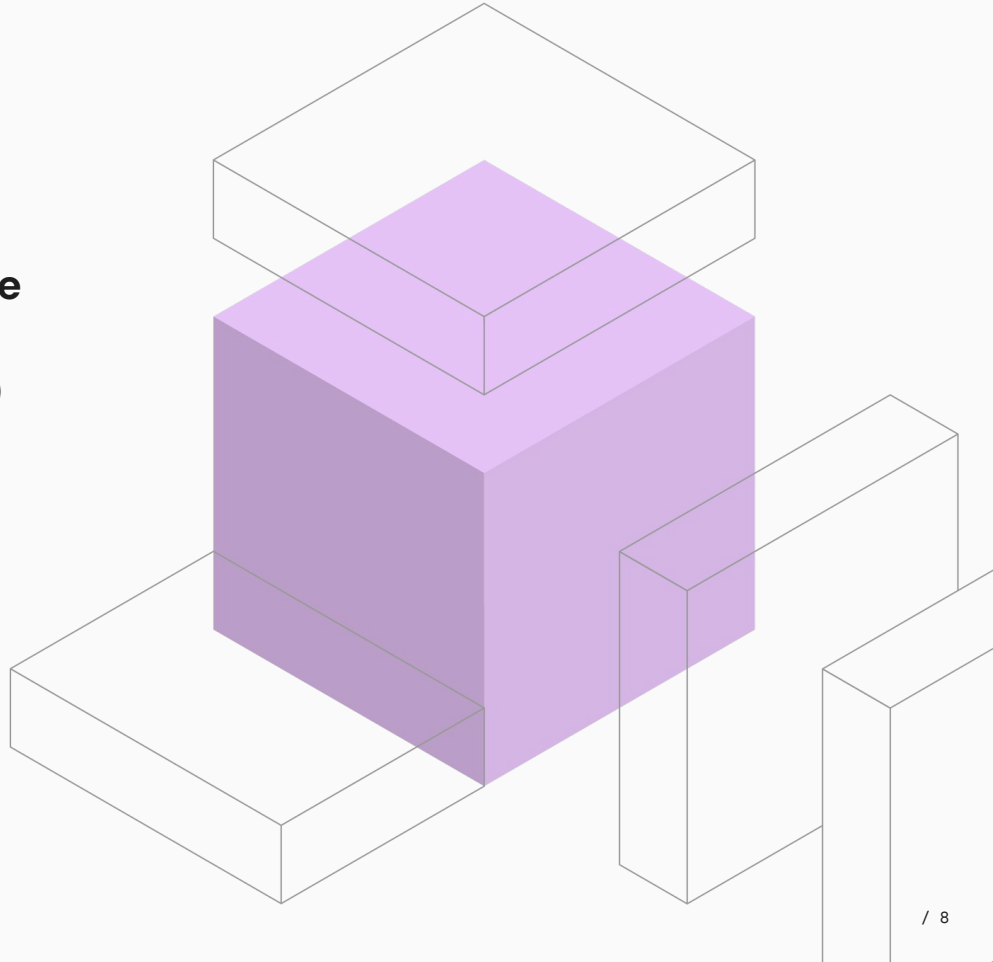
Let's start with some history

- SQL has emerged as the standard language for querying databases.
- Especially with sqlite being used in many desktop and mobile applications



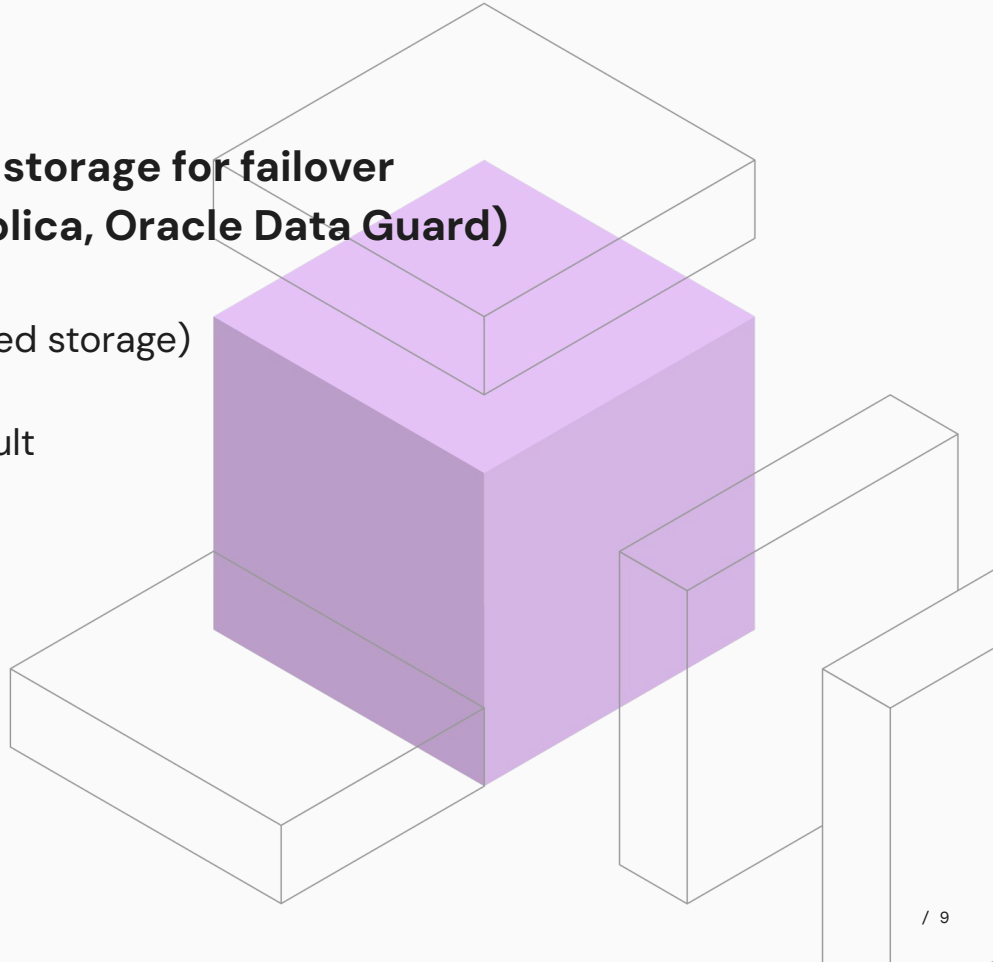
Let's start with some history

- **Databases were often**
  - **Monolithic**
  - **Single server**
  - **Expensive and special hardware**
    - Multi CPU, lots of memory
    - Redundancy (Disk, network, etc)



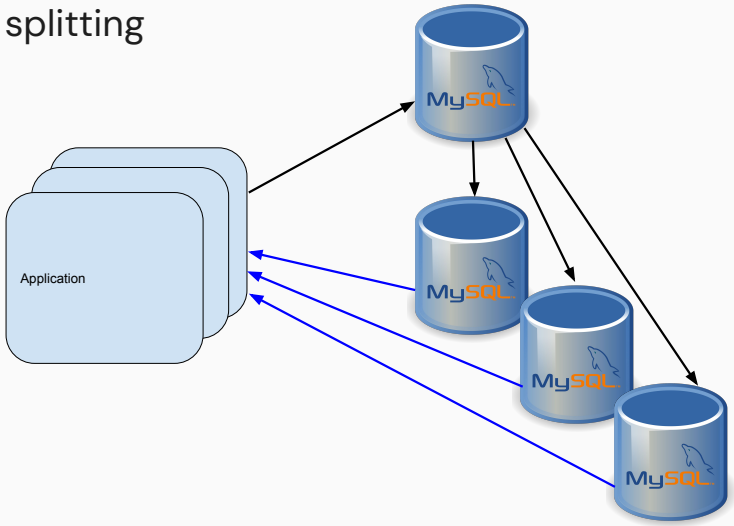
Let's start with some history

- **Next steps**
  - **Multiple machines with shared storage for failover**
  - **A standby instance (MySQL replica, Oracle Data Guard)**
  - **Drawbacks:**
    - need for special hardware (shared storage)
    - failover is manual
    - makes maintenance more difficult



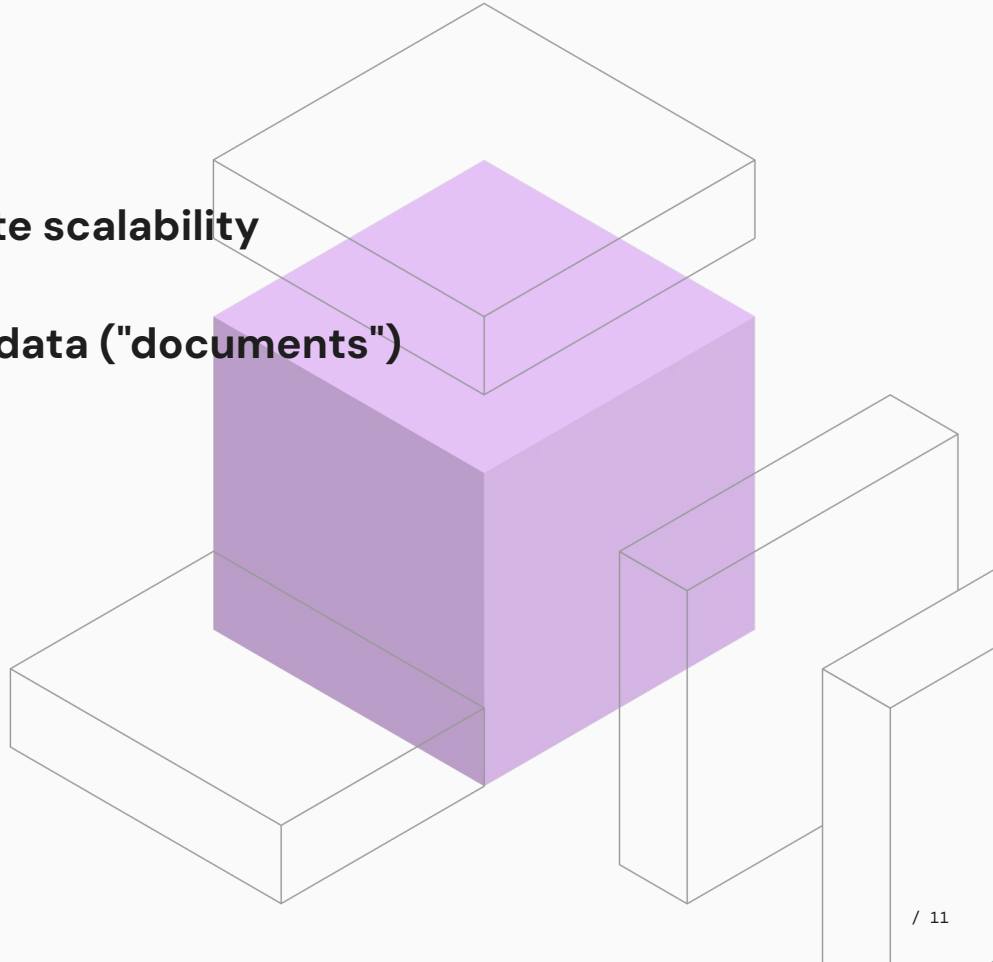
Let's start with some history

- **Scaling for web 2.0**
  - **MySQL with multiple read-only replicas**
  - **Drawbacks**
    - Replication delay
    - Applications need to do read/write splitting
    - Failover is still manual



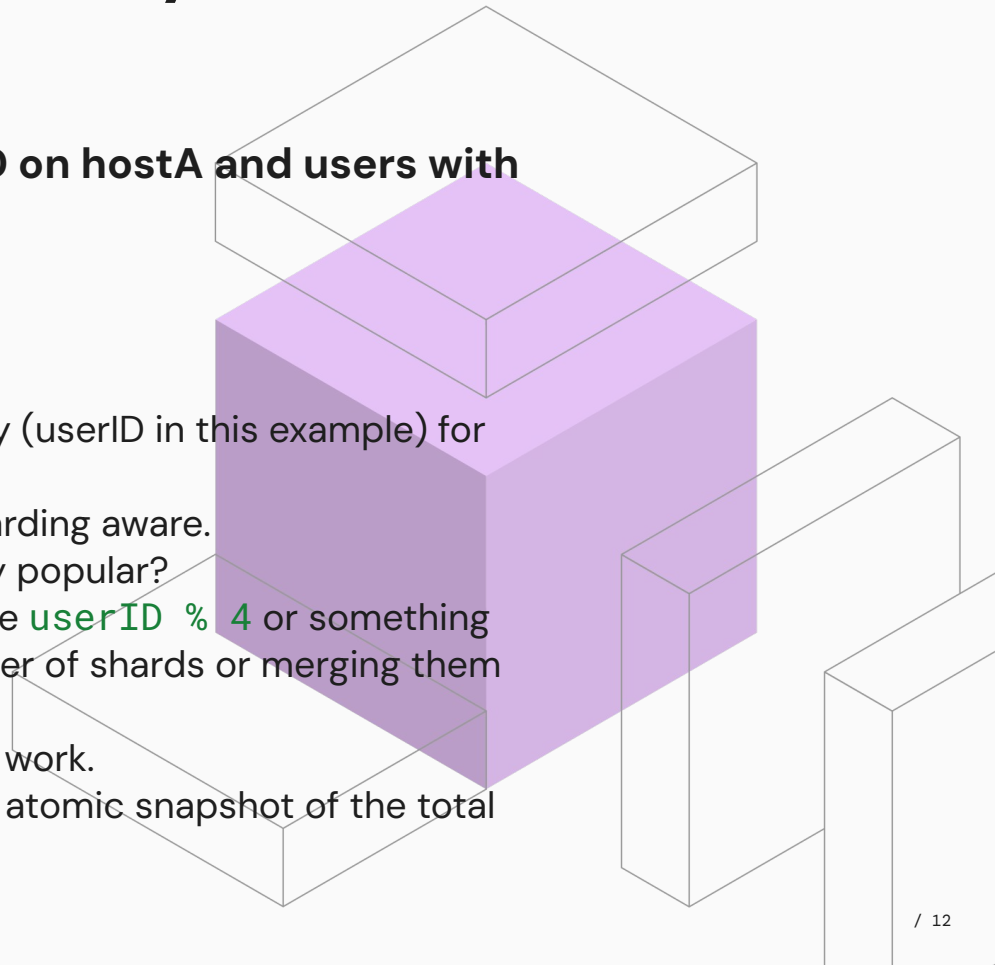
Let's start with some history

- **NoSQL**
  - **MongoDB with sharding**
  - **Easier to get both read and write scalability**
  - **Developers already know JSON**
  - **Easier to work with structured data ("documents")**
  - **Drawbacks:**
    - Duplicating data
    - Analytics can be challenging



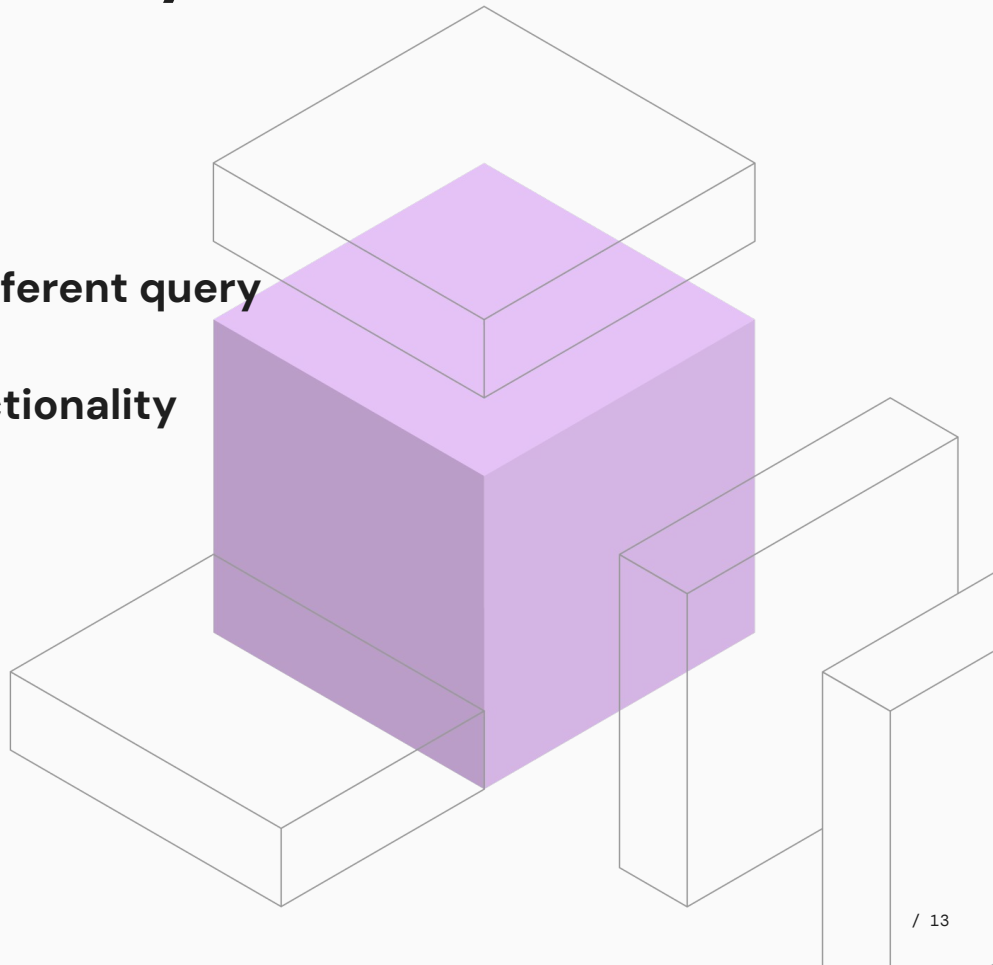
Let's start with some history

- **Sharding**
  - **Store users with an even userID on hostA and users with an odd userID on hostB.**
  - **Now there are two shards.**
  - **Double the write capacity.**
  - **Drawbacks:**
    - Need to decide the sharding key (userID in this example) for each table.
    - The application needs to be sharding aware.
    - What if one user becomes really popular?
    - Instead of even/odd, you can use `userID % 4` or something similar. But increasing the number of shards or merging them becomes hard.
    - Analytics needs a lot of custom work.
    - A backup no longer provides an atomic snapshot of the total database.



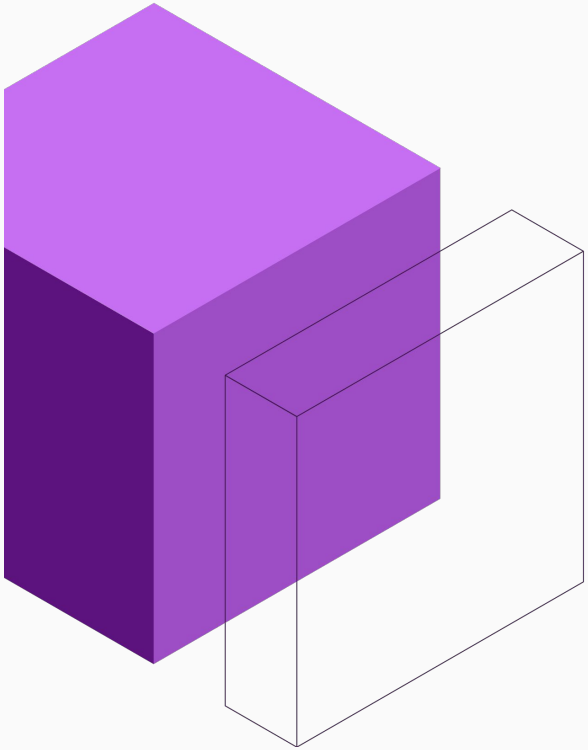
Let's start with some history

- NewSQL
  - Response to NoSQL
  - Good JSON support
  - Sometimes with support for different query languages/protocols
  - Might have some sharding functionality



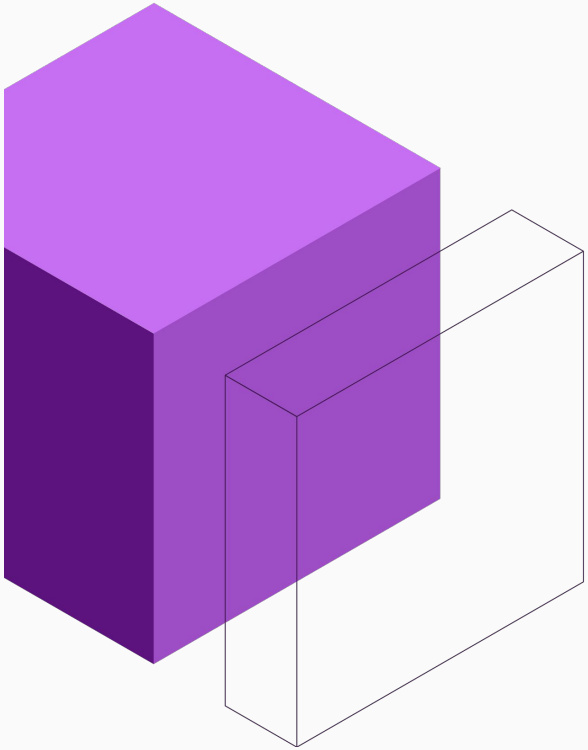
03 Distributed databases

Distributed databases



- Store a database on a set of machines instead of on a single machine.
- Present the database to applications as a single logical database.

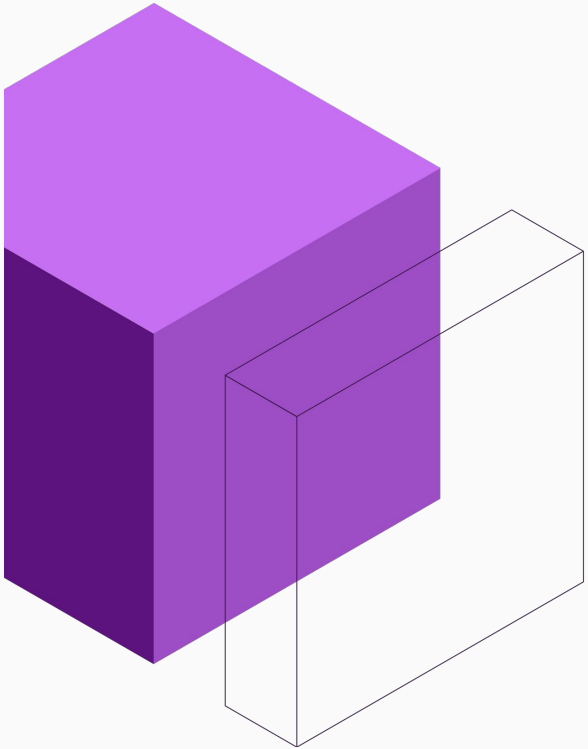
Distributed databases



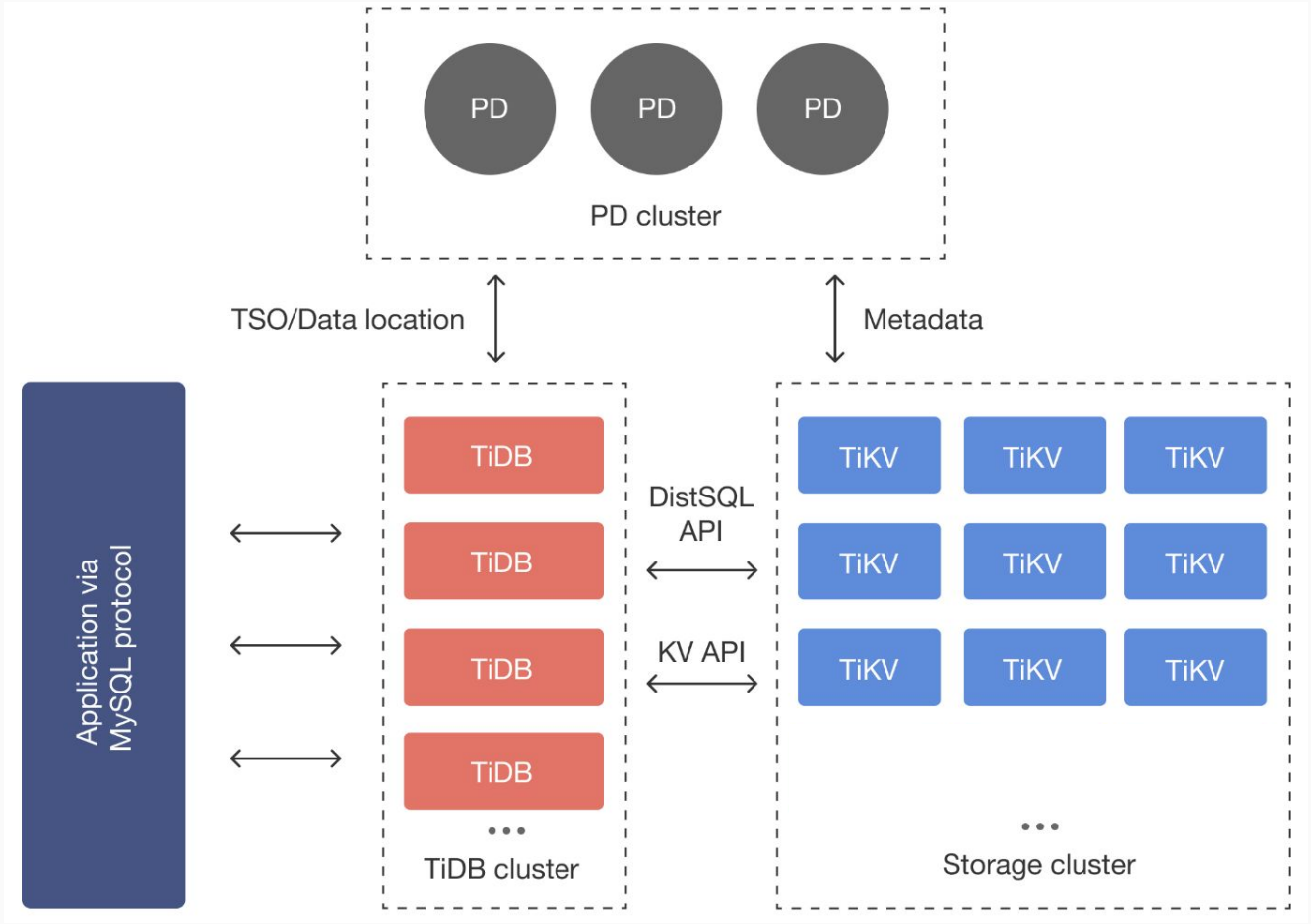
- Monolithic database
  - Every machine stores a full copy of the database
  - More machines means more copies.

- Distributed database
  - Every machine stores a part of the database
  - More machines means more capacity.

Distributed databases



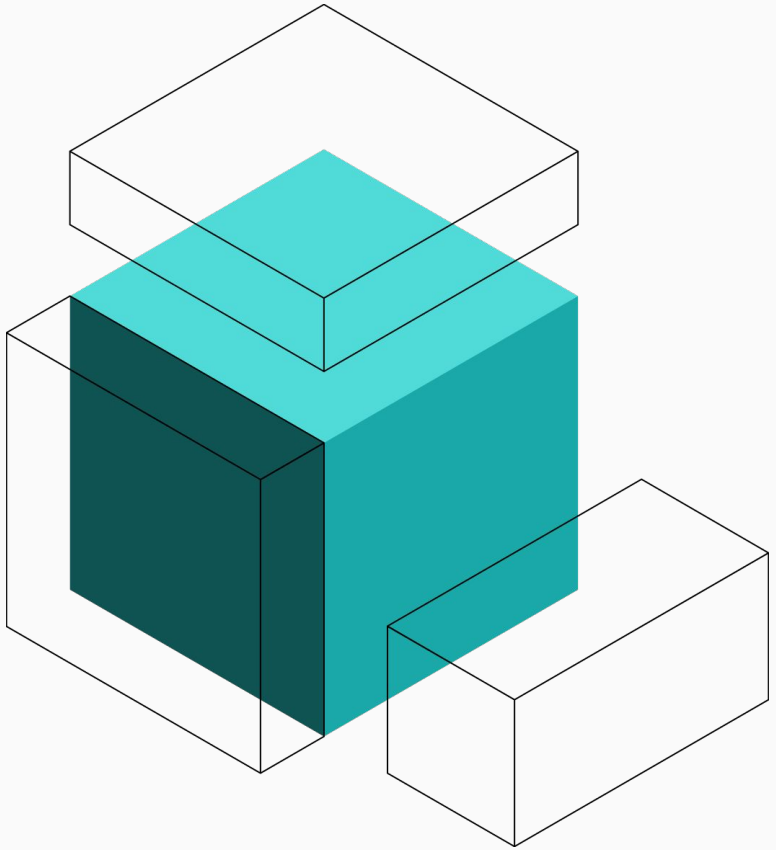
- Redundancy
  - Every part of the database is stored multiple times in the cluster.
  - Every host is labeled with the availability zone.
  - The cluster makes sure that the 3 copies are stored on machines in separate availability zones.


PD
PD
PD
PD cluster
TSO/Data location
Metadata
Application via MySQL protocol
TiDB
TiDB
TiDB
TiDB
TiDB cluster
DistSQL API
KV API
TiKV
TiKV
TiKV
TiKV
TiKV
TiKV
TiKV
TiKV
TiKV
Storage cluster

04 Building blocks

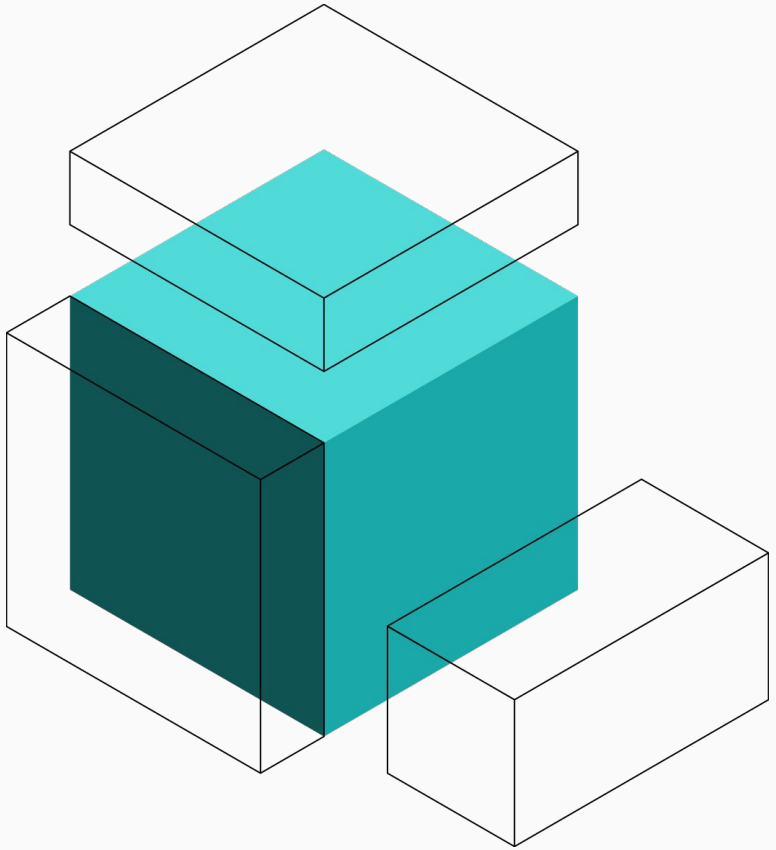
Building blocks

- TiKV
  - RocksDB
  - Raft
- PD
  - TSO (Time Stamp Oracle)
- TiDB
  - SQL processing
  - Query pushdown
- TiFlash



Building blocks: TiKV

- Key-value store
- Can be used by applications directly.
- This is a CNCF (Cloud Native Computing Foundation) project.
- Written in Rust



Data organization within TiDB

- For a row in a Table, row data is encoded in key-value pairs with the format below:

t<tableID>_r<rowID> => [col1, col2, col3, col4]

- If there is secondary index with a column, the index data of a row is encoded in this way:

t<tableID>_i<indexID>_indexedColumnsValue => rowID

Or

t<tableID>_i<indexID>_indexedColumnsValue_rowID => nil

Data organization within TiDB

```
CREATE TABLE user_table (
        id INT,
        name VARCHAR(64),
        email VARCHAR(1024),
        PRIMARY KEY(id)
);
```

user_table		
1	daniel	daniel.van.eeden@pingcap.com
2	foo	bar@pingcap.com
...	...	...

Within TiKV:

t101_r1 => [1,daniel,daniel.van.eeden@pingcap.com]
t101_r2 => [2, foo, bar@pingcap.com]
t101_r… => ...

Data organization within TiDB

```
CREATE TABLE user_table (
        id INT,
        name VARCHAR(64),
        email VARCHAR(1024),
        KEY (name),
        PRIMARY KEY(id)
);
```

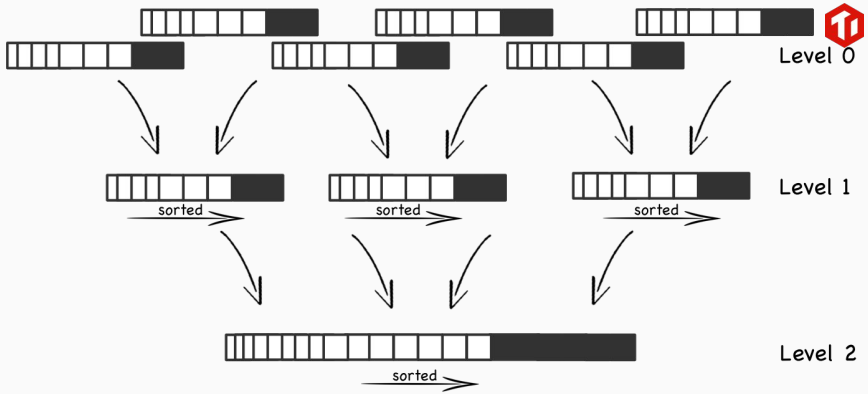
user_table		
1	daniel	daniel.van.eeden@pingcap.com
2	foo	bar@pingcap.com
...	...	...

Within TiKV:

t101_r1 => [1,daniel,daniel.van.eeden@pingcap.com]
t101_r2 => [2, foo, bar@pingcap.com]
t101_… => ...
...
t101_i1_daniel_1 => nil
t101_i1_foo_2 => nil
t101_i1_… => ...

Building blocks: RocksDB

- Used by TiKV
- LSM-Tree
  - **Log Structured Merge Tree**
  - **Sequential writes**
  - **Sorted during compaction**
  - **Good compression**
  - **Deletes via tombstones**
- Benefits to TiKV/TiDB
  - **FLASHBACK TABLE <table>**
  - Allows you to read older versions of the table data
  - Good INSERT performance

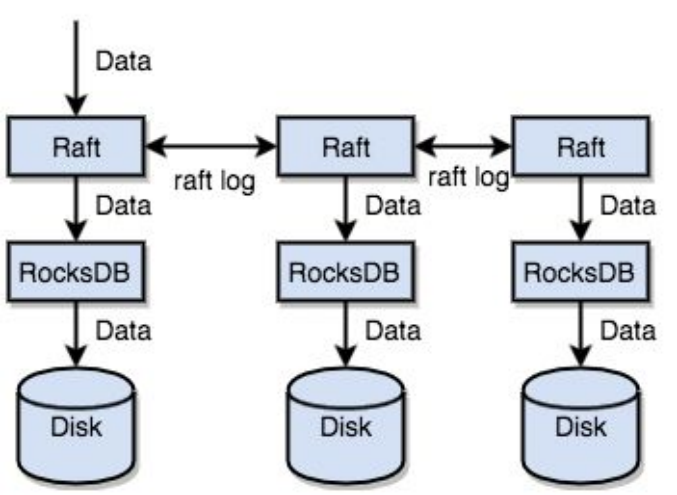


Compaction continues creating fewer, larger and larger files

Source: Wikipedia

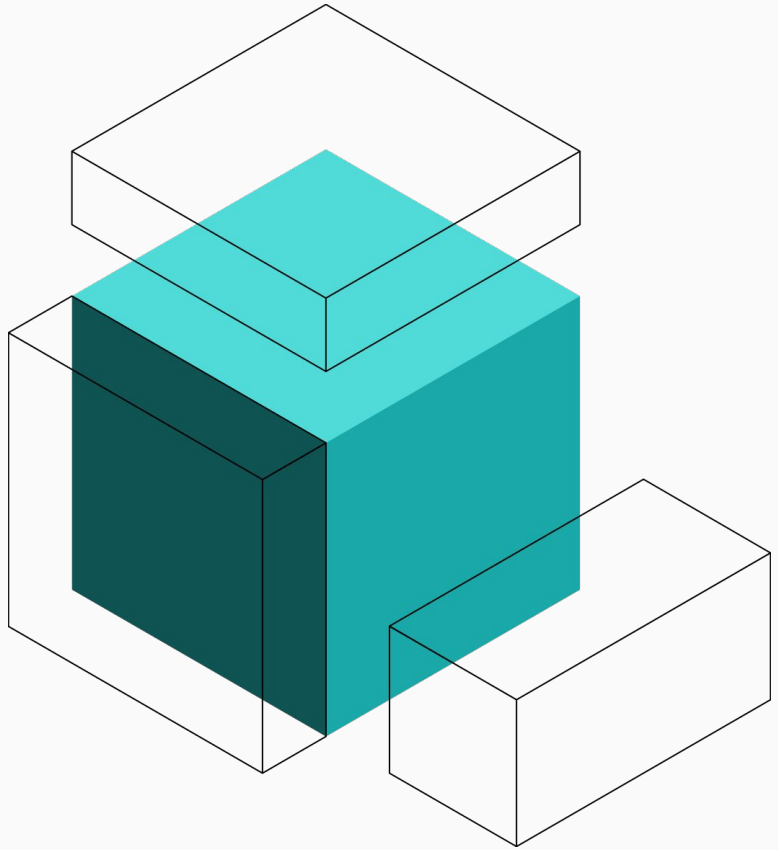
Building blocks: Raft

- Each data region (part of a table) is a raft group
- Raft provides group consensus (e.g. leader election)
- Raft log is used to replicate data



Building blocks: PD

- Placement Driver
- Usually a set of 3 machines
- This is orchestrating placement of data
- Distribute data regions (raft groups) in such a way that
  - Capacity use is balanced
  - Read I/O is balanced
  - Write I/O is balanced
  - Raft leaders are balanced



TiDB Architecture

SELECT id FROM orders WHERE id=1000001

Stateless SQL Layer

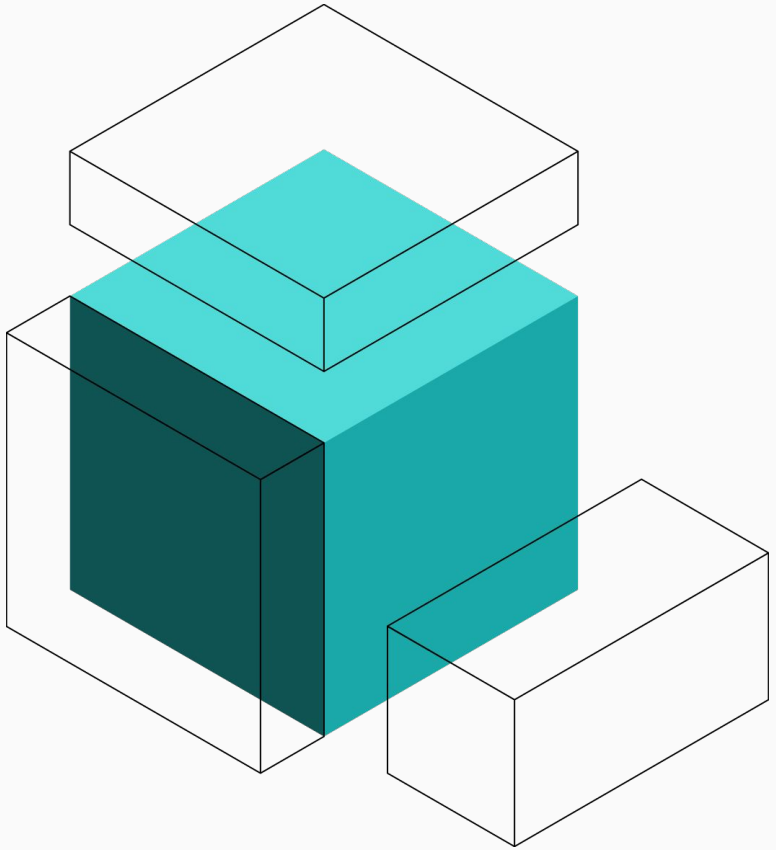
- TiDB node 1
- TiDB node 2
- TiDB node 3
- TiDB node 4

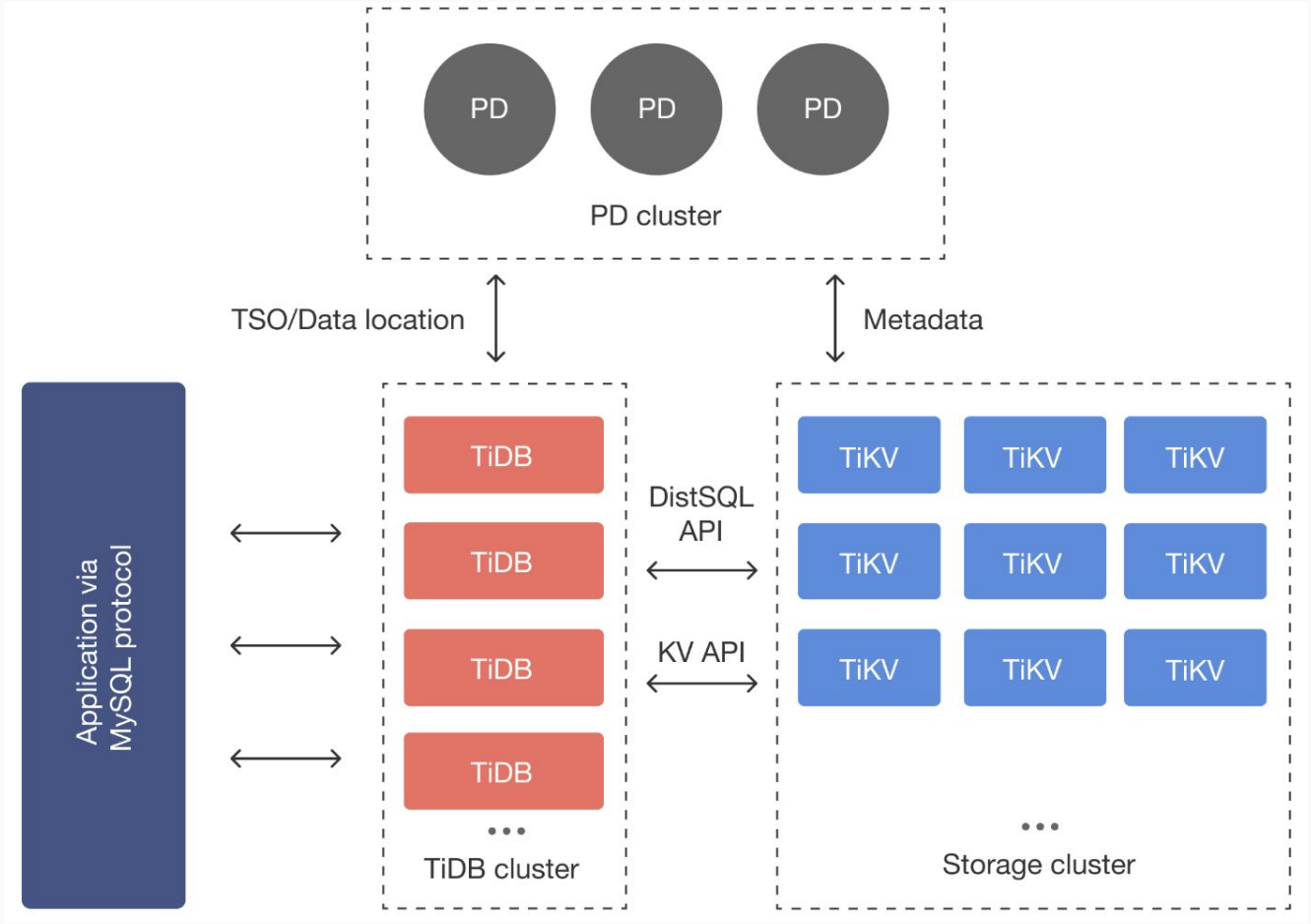
orders	
1	data
2	data
...	...
1000001	...
...	...
999999000	...
...	...

AZ 1	AZ 2	AZ 3
TiKV node 1: Region 5, Region 3	TiKV node 2: Region 1, Region 3	TiKV node 3: Region 2, Region 4
TiKV node 4: Region 6, Region 2	TiKV node 5: Region 5, Region 4	TiKV node 6: Region 6, Region 1
TiKV node 7: Region 1, Region 4	TiKV node 8: Region 6, Region 2	TiKV node 9: Region 3, Region 5

Building blocks: TiDB

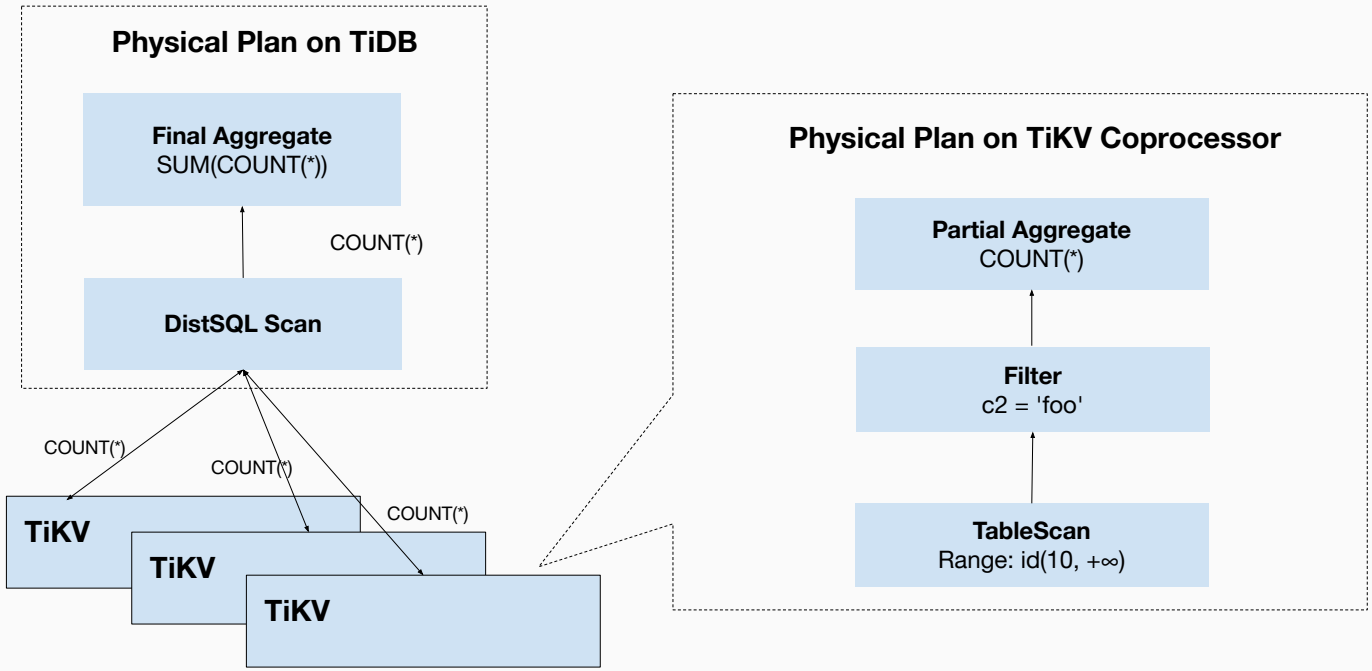
- Provides MySQL protocol support
- Provides MySQL syntax support
- Doesn't store data. Data is stored on TiKV.
- Communicates with TiKV in order to fulfil client requests
- This component is written in Go




PD
PD
PD
PD cluster
TSO/Data location
Metadata
Application via MySQL protocol
TiDB
TiDB
TiDB
TiDB
TiDB cluster
DistSQL API
KV API
TiKV
TiKV
TiKV
TiKV
TiKV
TiKV
TiKV
TiKV
TiKV
Storage cluster

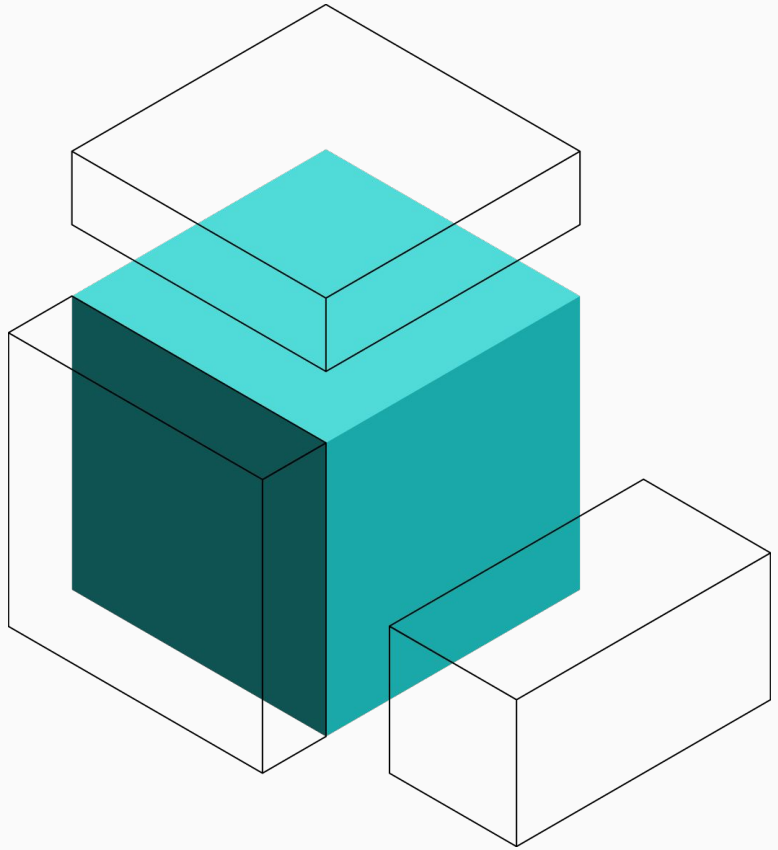
Query Execution/Distributed Computing

```
SELECT COUNT(*) FROM t WHERE id > 10 AND c2 = 'foo';
```



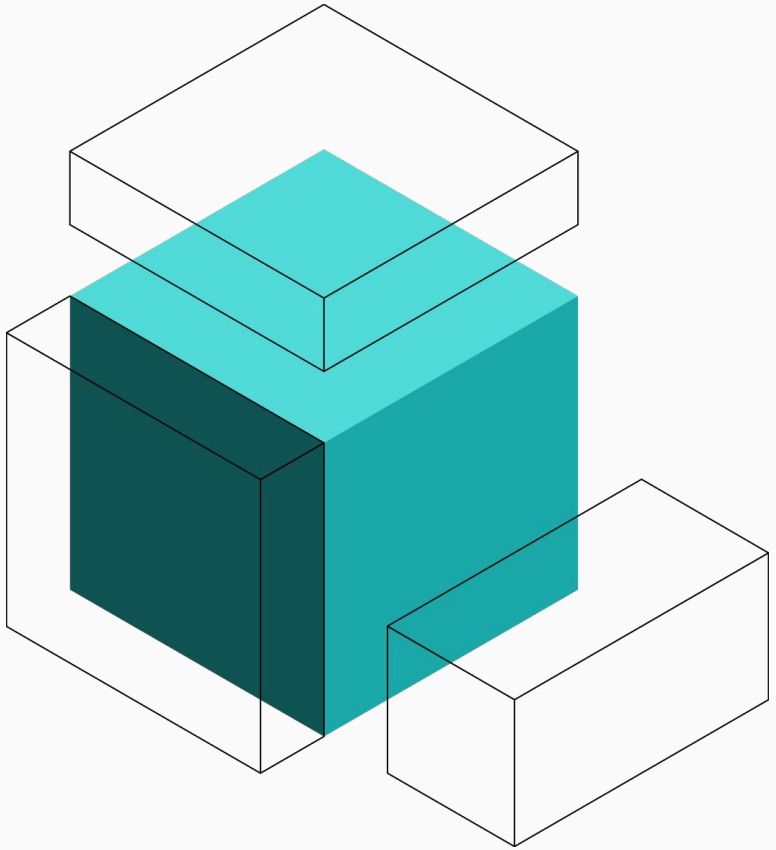
Building blocks: TSO

- **Time Stamp Oracle**
- **TSO is part of PD**
- **Gives out timestamps**
  - **1st part**
    - UNIX timestamp in milliseconds
    - 46 bits
  - **2nd part**
    - Logical timestamp
    - 18 bits
- **logical part is needed for**
  - **multiple requests in the same millisecond**
  - **clock adjustments etc**



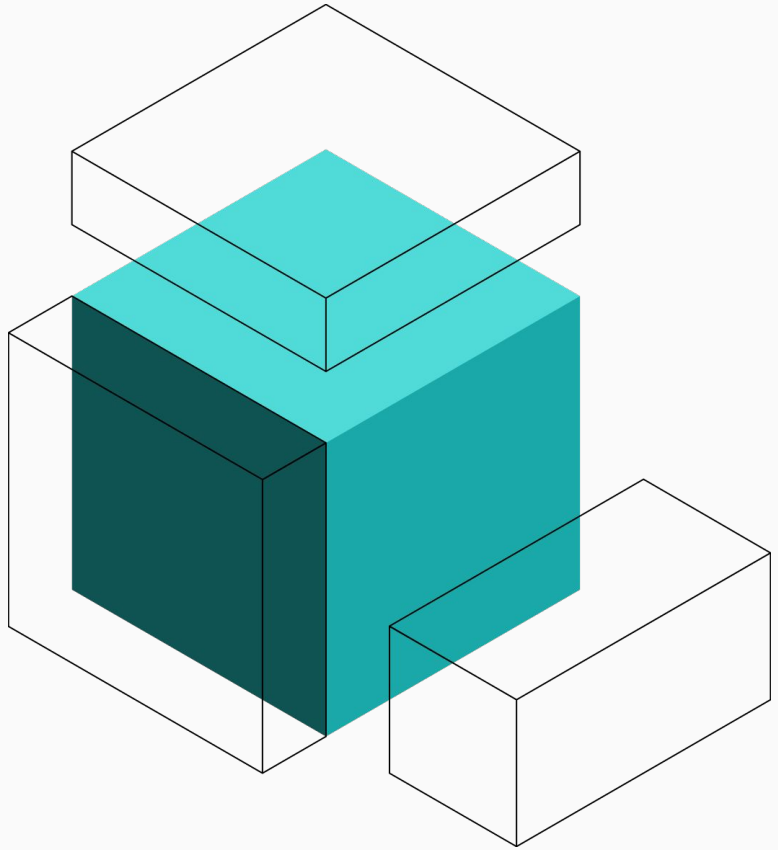
Building blocks: TSO

- timestamps are used for
  - Transaction start time
  - Transaction commit time
  - Locking
  - Transaction isolation
  - KV records



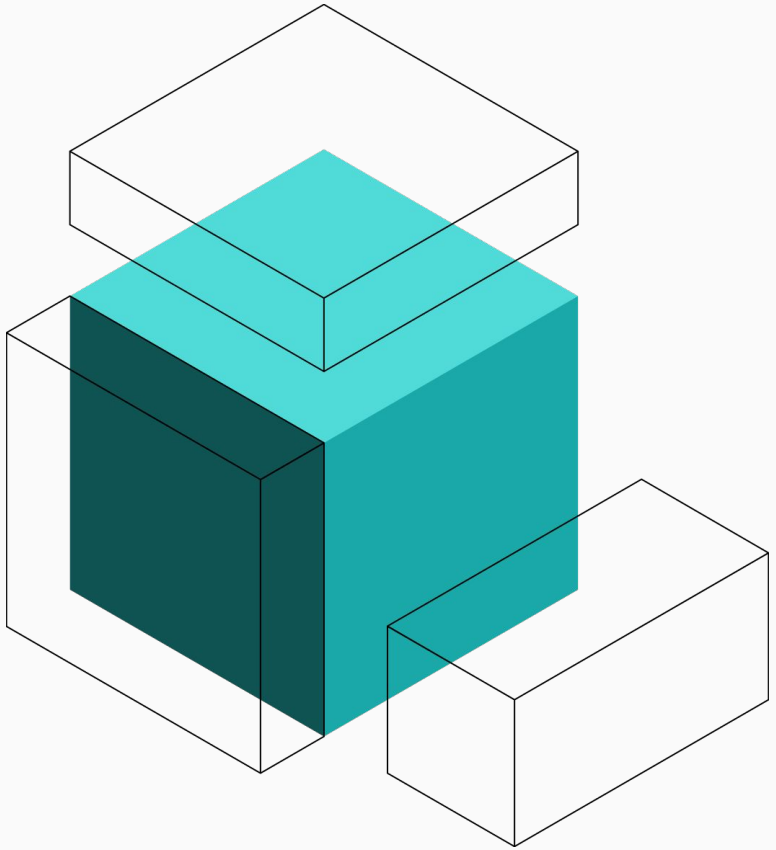
Building blocks: TiFlash

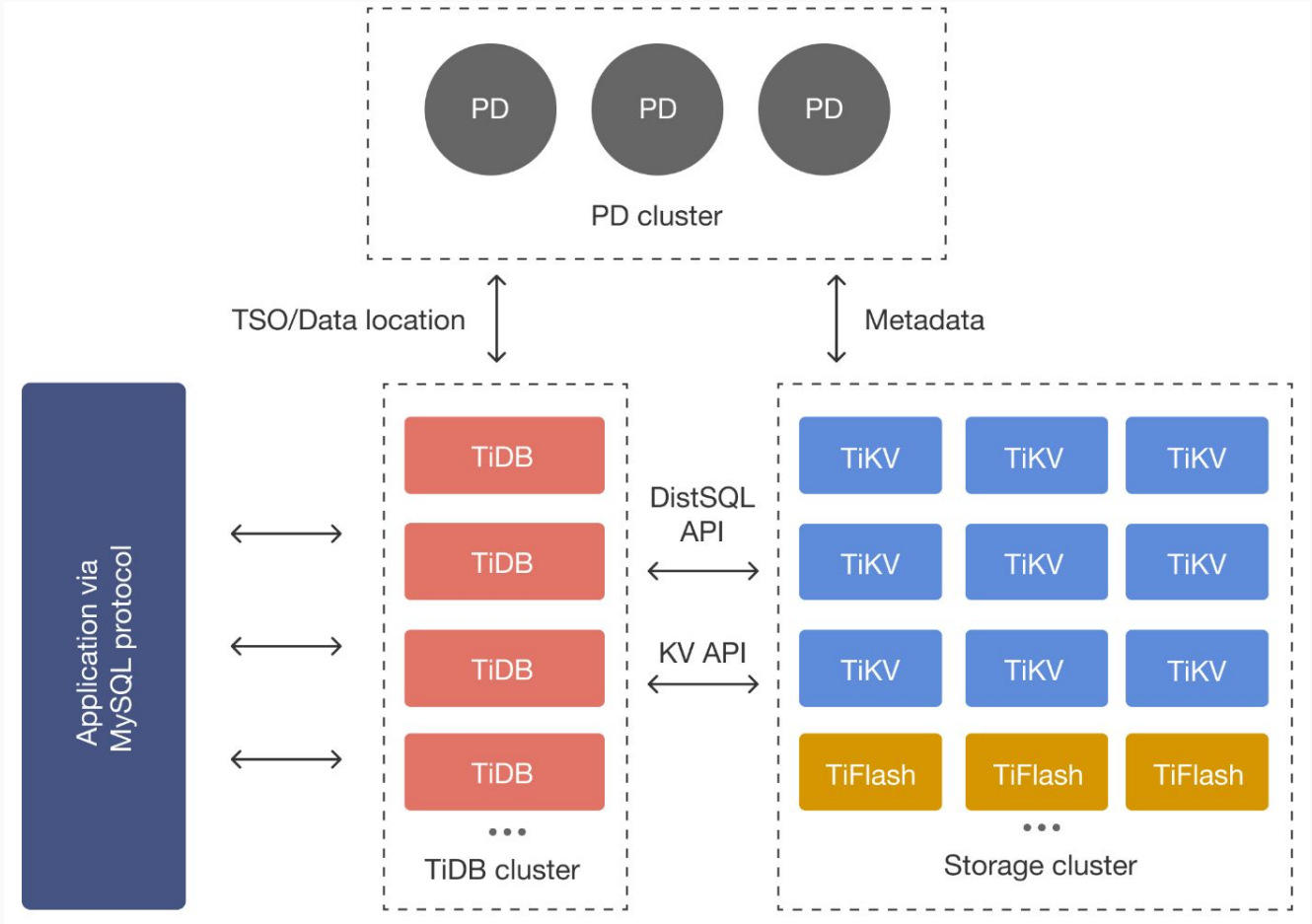
- Stores a **copy** of a table in a columnar format.
- Not all tables have to have a copy in TiFlash
- Written in C++
- Receives data asynchronously as a raft learner.



Building blocks: TiFlash

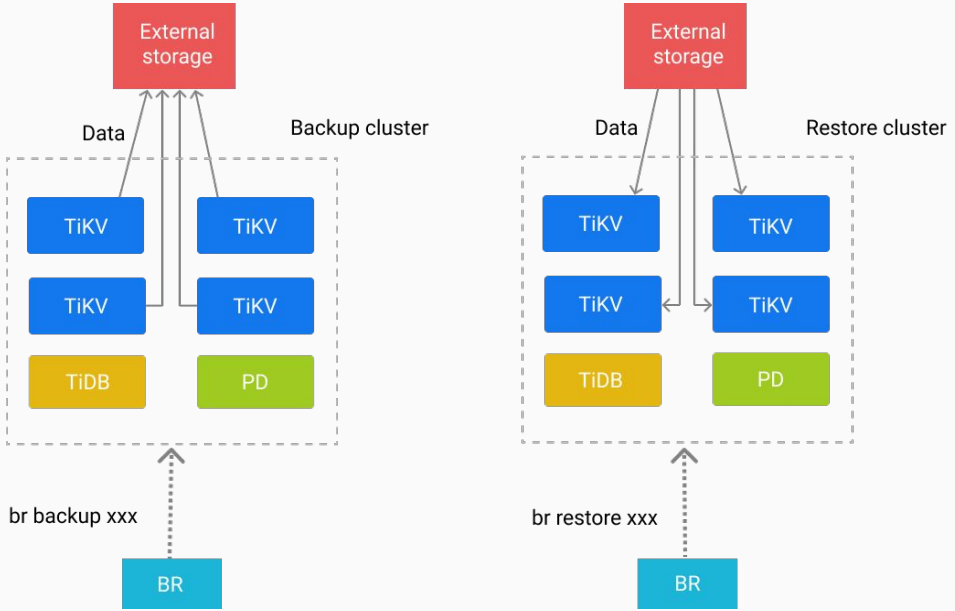
- The optimizer in TiDB is allowed to use TiFlash for queries or parts of queries.
- Optimizer hints are available to restrict or force the use of TiFlash
- Data from TiFlash is guaranteed to be transactionally consistent.




PD
PD
PD
PD cluster
TSO/Data location
Metadata
Application via MySQL protocol
TiDB
TiDB
TiDB
TiDB
TiDB cluster
DistSQL API
KV API
TiKV
TiKV
TiKV
TiKV
TiKV
TiKV
TiKV
TiKV
TiKV
TiFlash
TiFlash
TiFlash
Storage cluster

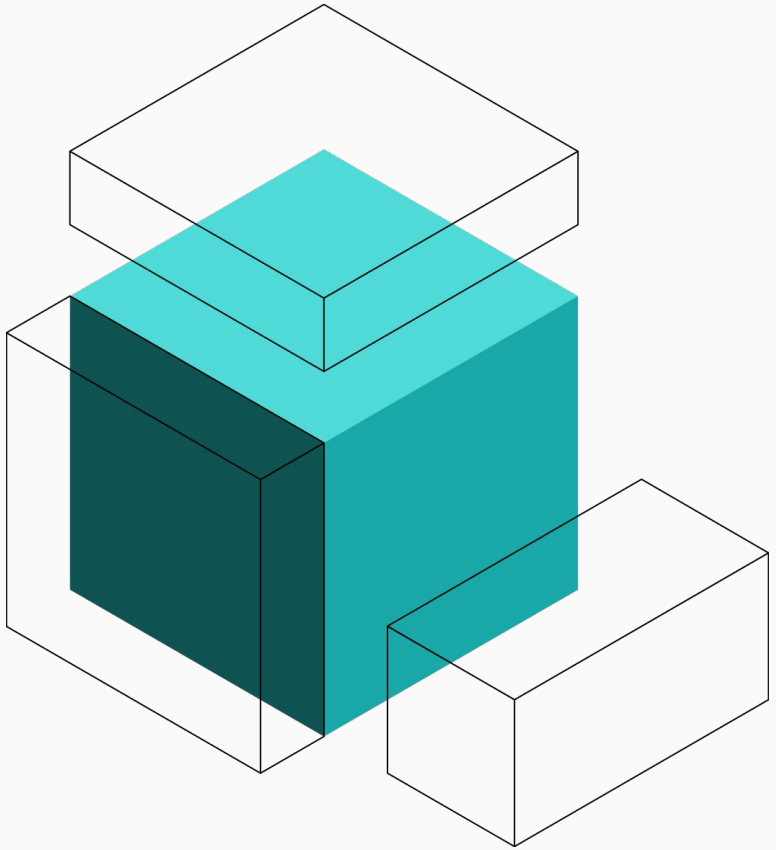
Miscellaneous: BR

- Backup & Restore
  - Using S3 (or compatible). Shared filesystems like NFS also work.
  - Centrally controlled
  - Every TiKV writes its own part of the backup
  - Relies on LSM Tree properties to make a consistent backup



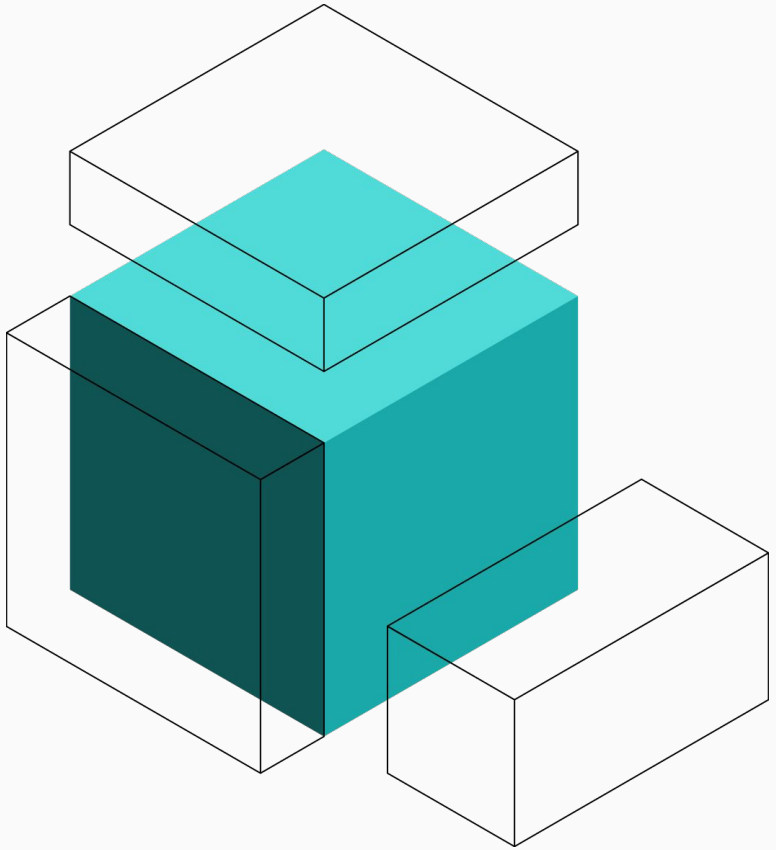
Miscellaneous: TiCDC

- Change Data Capture
- Target
  - Kafka
  - MySQL
  - TiDB
- Connects to TiKV to read changes.
- HA



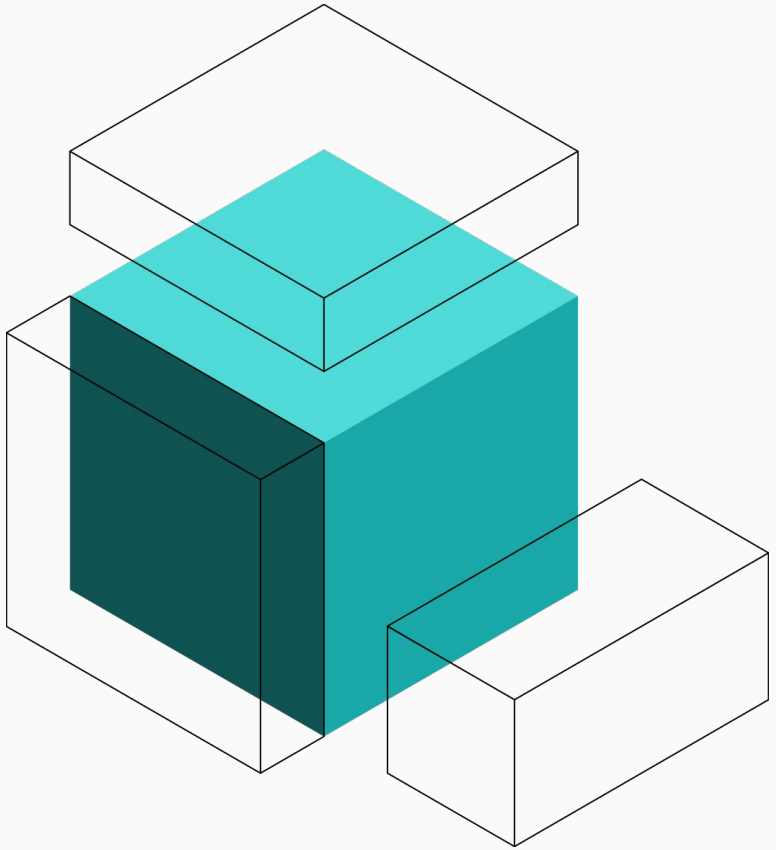
Miscellaneous: DM

- Data Migration
- Read data from MySQL
  - Initial copy (in parallel)
  - Binlogs
- Write to TiDB



Future

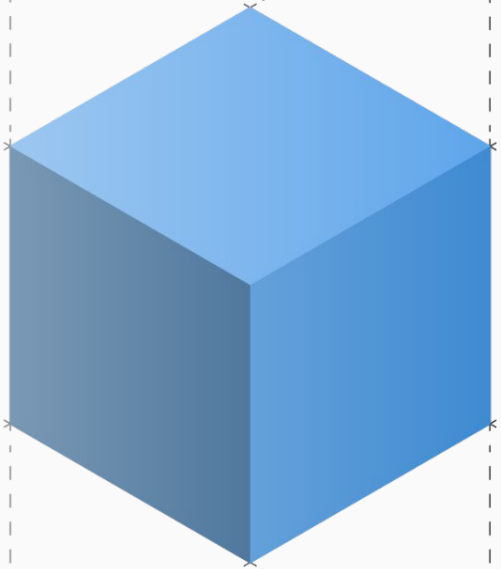
- Cloud Storage Engine
- Put data on S3
- The TiKV machines cache data
- Adding or removing nodes no longer requires the copying of data.
- Scaling up or down is nearly instantaneous



05 Conclusion

Conclusion

- Distributed databases provide you with
  - Scalability
  - High availability
- Distributed are developer friendly
  - No replication delay
  - No read/write splitting needed
  - No sharding needed



Questions?

- Daniel.van.Eeden@pingcap.com
- https://docs.pingcap.com/
- Try for yourself
  - TiUP Playground: https://tiup.io
  - Or TiDB Cloud
- https://labs.tidb.io/