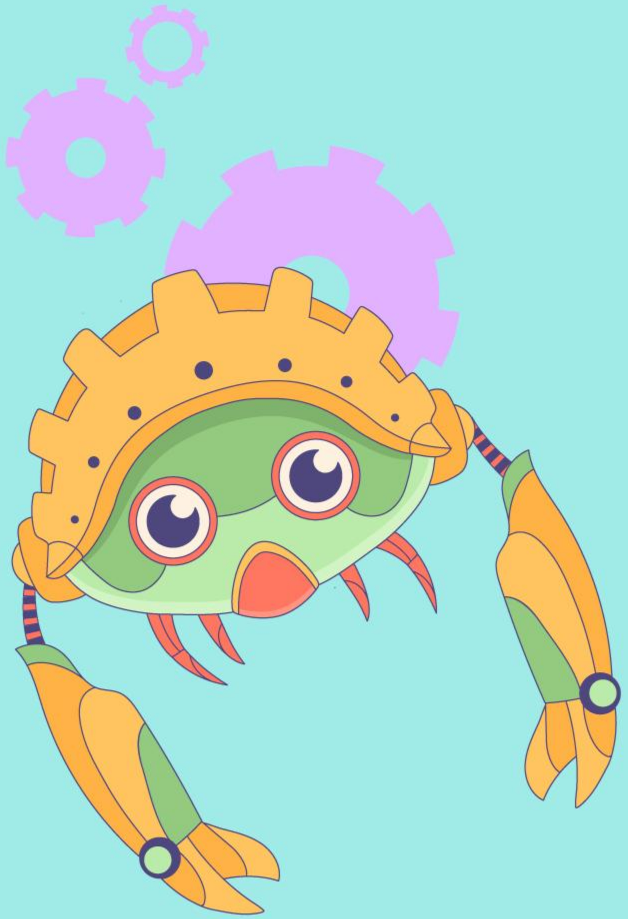
NIKITA LAPKOV

Senior Software Engineer

RHINO: LOW-LATENCY KEY-VALUE DATABASE IN RUST



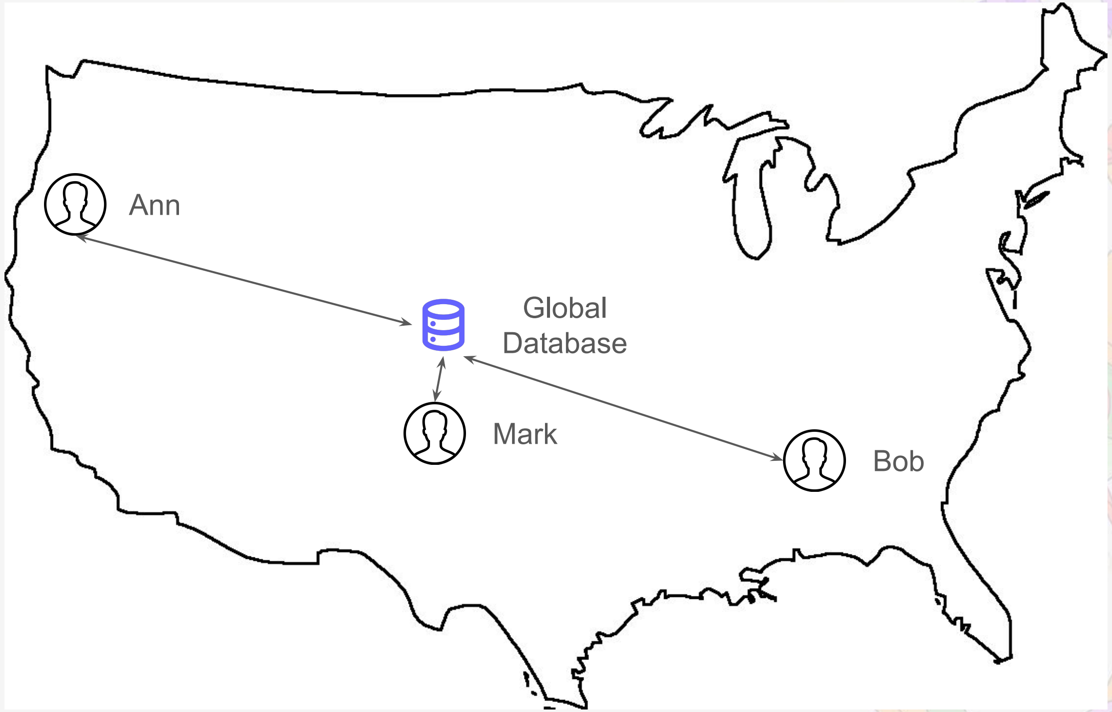
HELLO!

Worked on MongoDB, ClickHouse, YDB

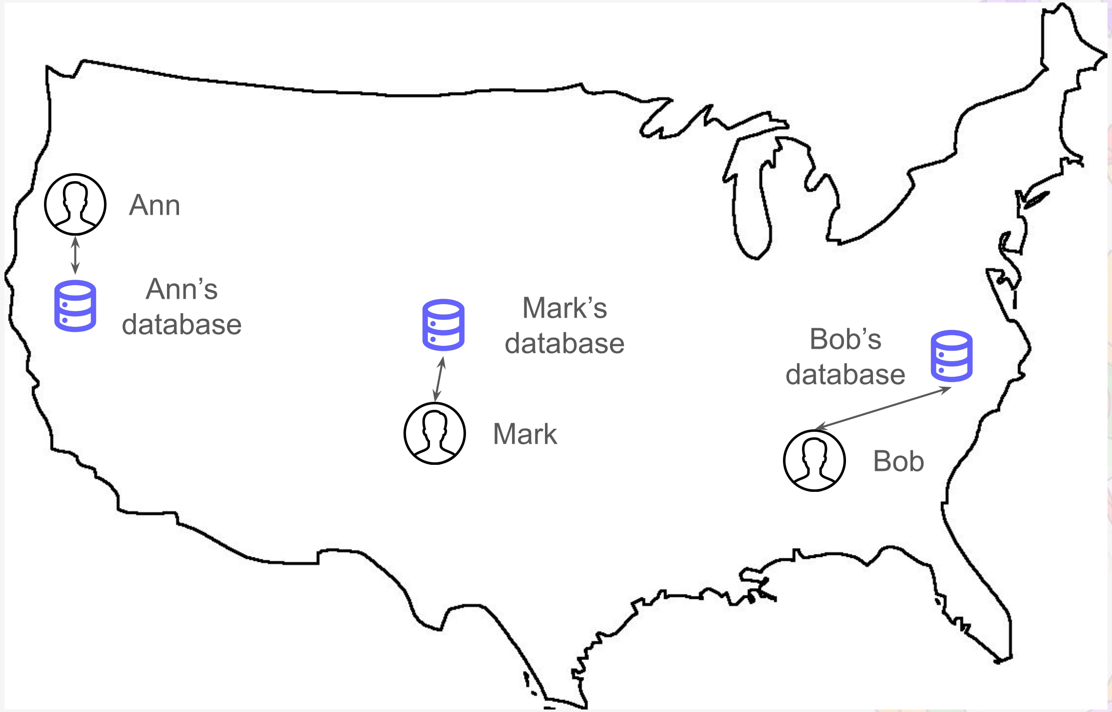
System Programming, Query Engines

laplab.me
hi@laplab.me

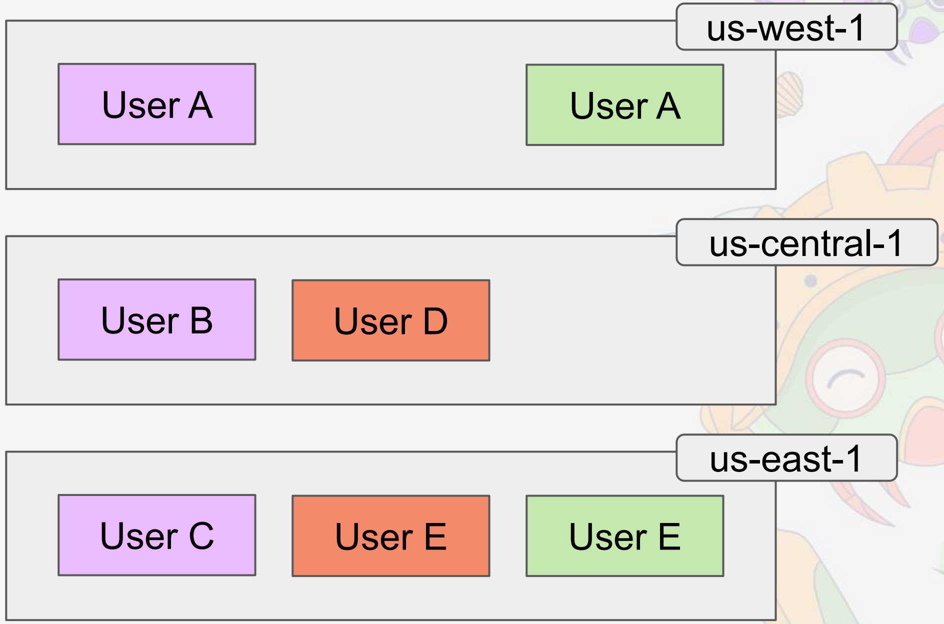
DATA ACCESS LATENCY



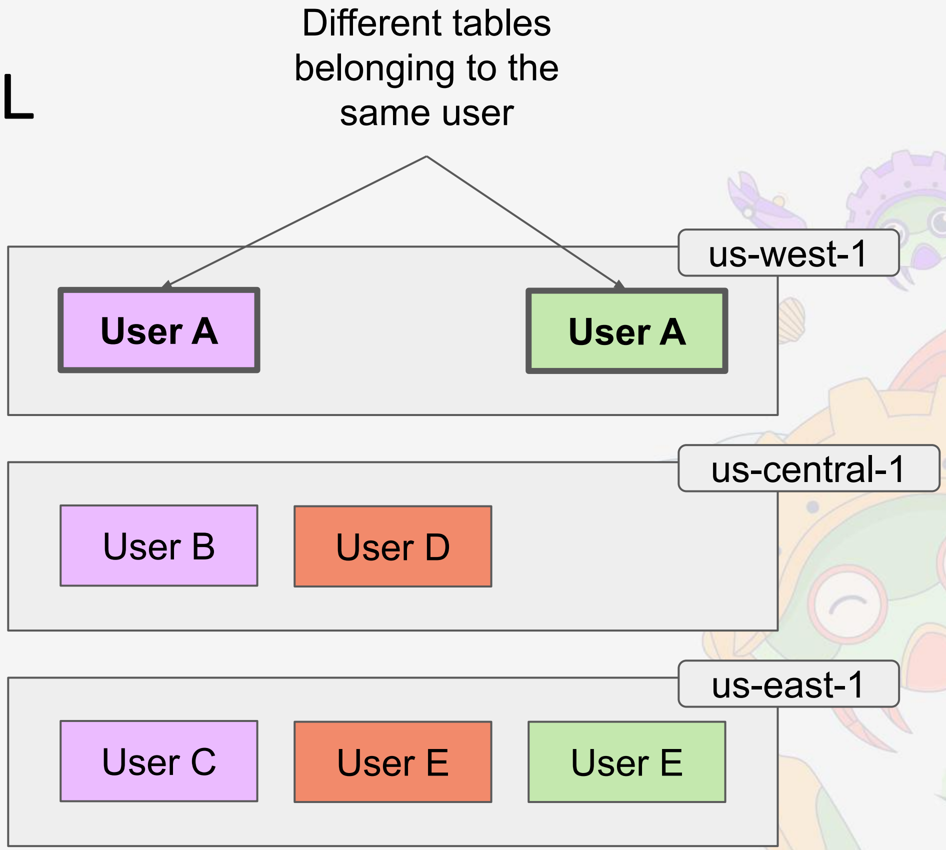
RHINO AS A SOLUTION



DATA MODEL

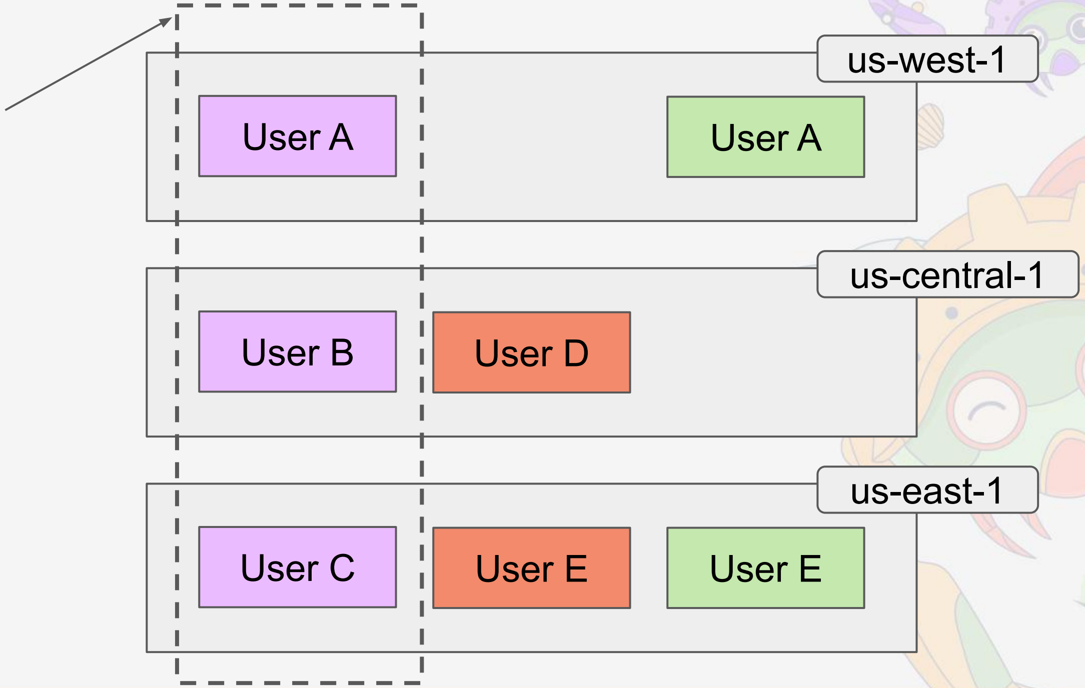

us-west-1
User A
User A
us-central-1
User B
User D
us-east-1
User C
User E
User E

DATA MODEL

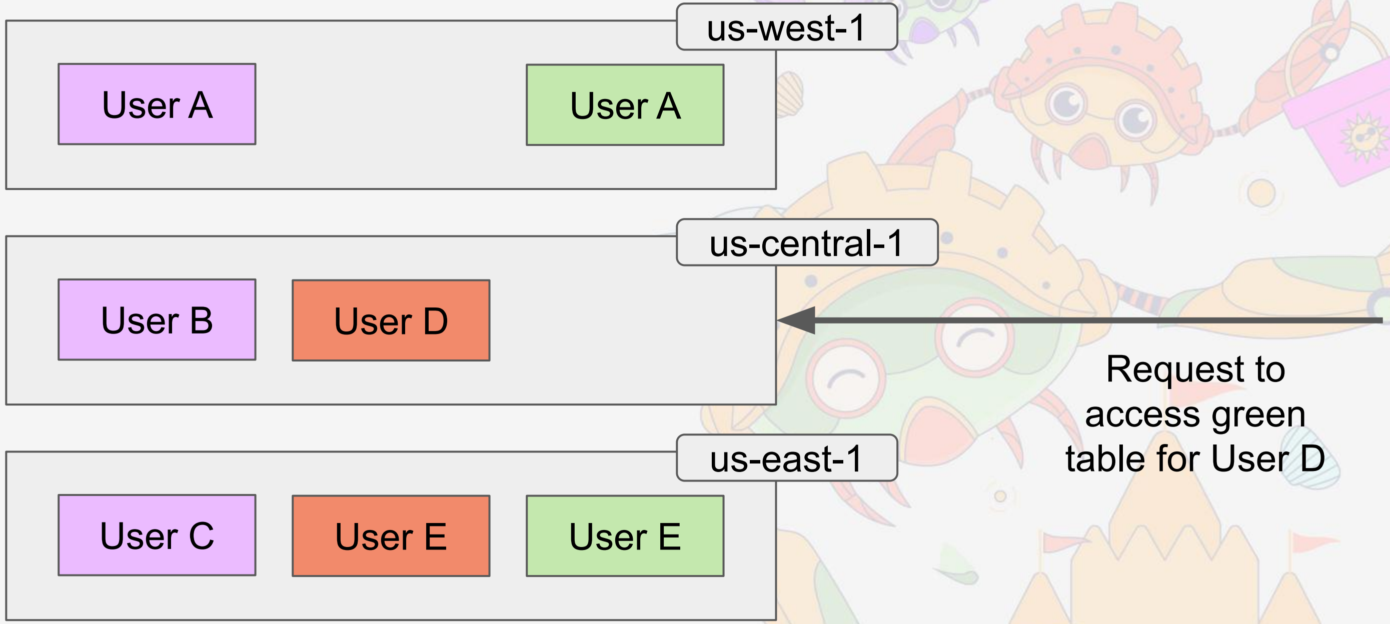

Different tables belonging to the same user
us-west-1
User A
User A
us-central-1
User B
User D
us-east-1
User C
User E
User E

DATA MODEL

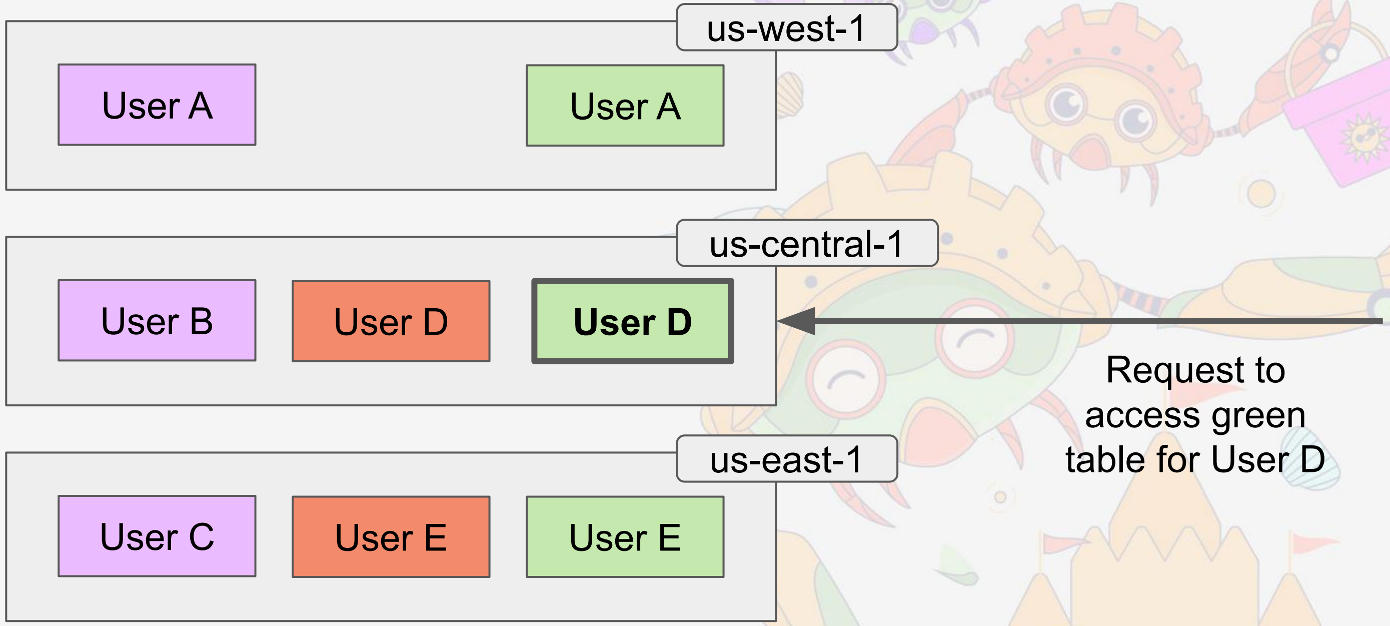
Same global schema between regions



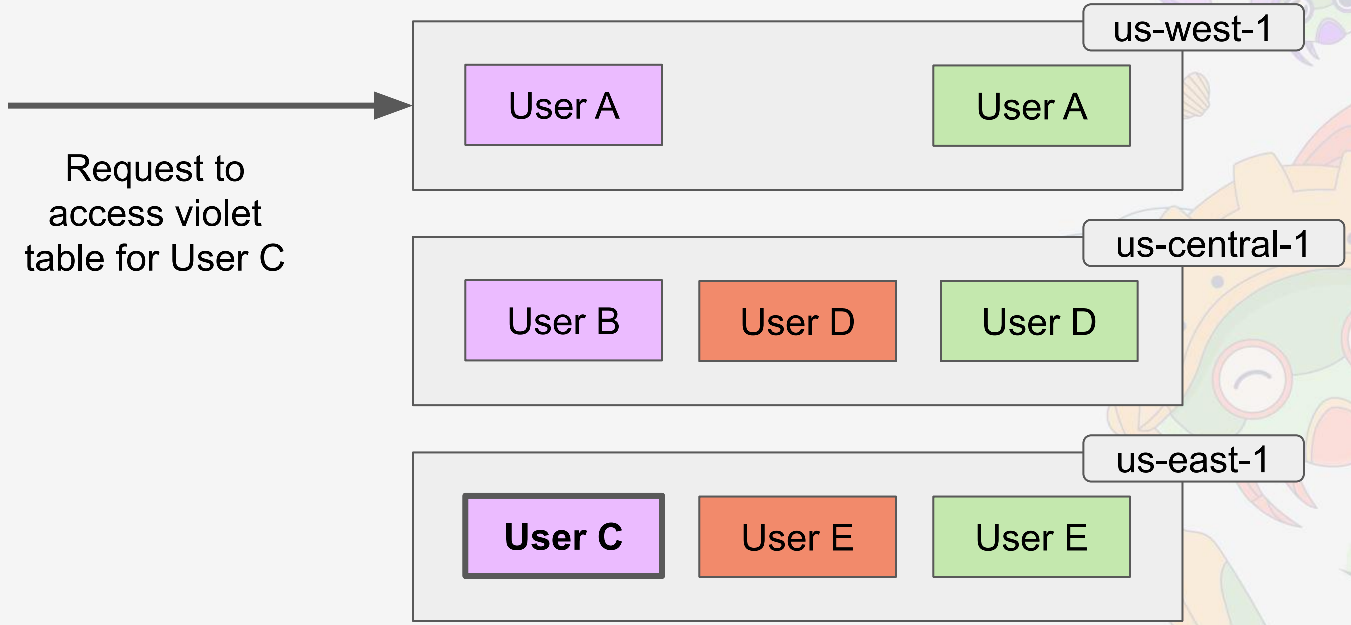
DATA MODEL


us-west-1
User A
User A
us-central-1
User B
User D
us-east-1
User C
User E
User E
Request to access green table for User D

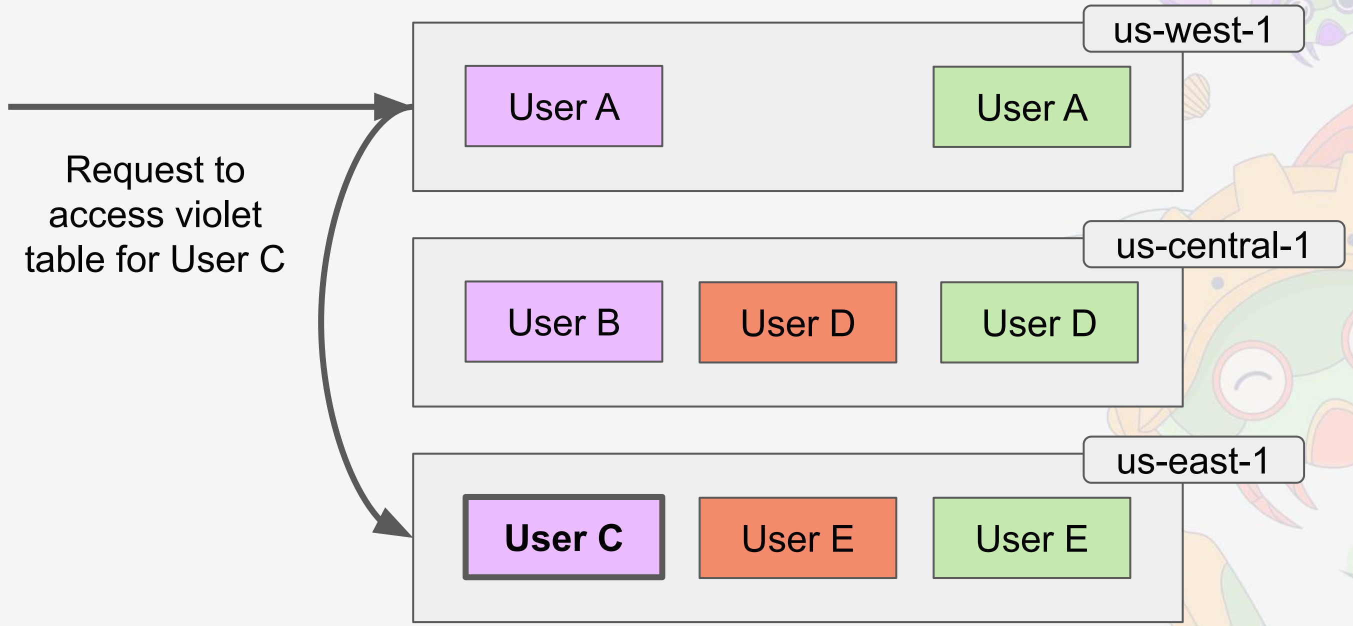
DATA MODEL


us-west-1
User A
User A
us-central-1
User B
User D
User D
us-east-1
User C
User E
User E
Request to access green table for User D

DATA MODEL


Request to access violet table for User C
us-west-1
User A
User A
us-central-1
User B
User D
User D
us-east-1
User C
User E
User E

DATA MODEL

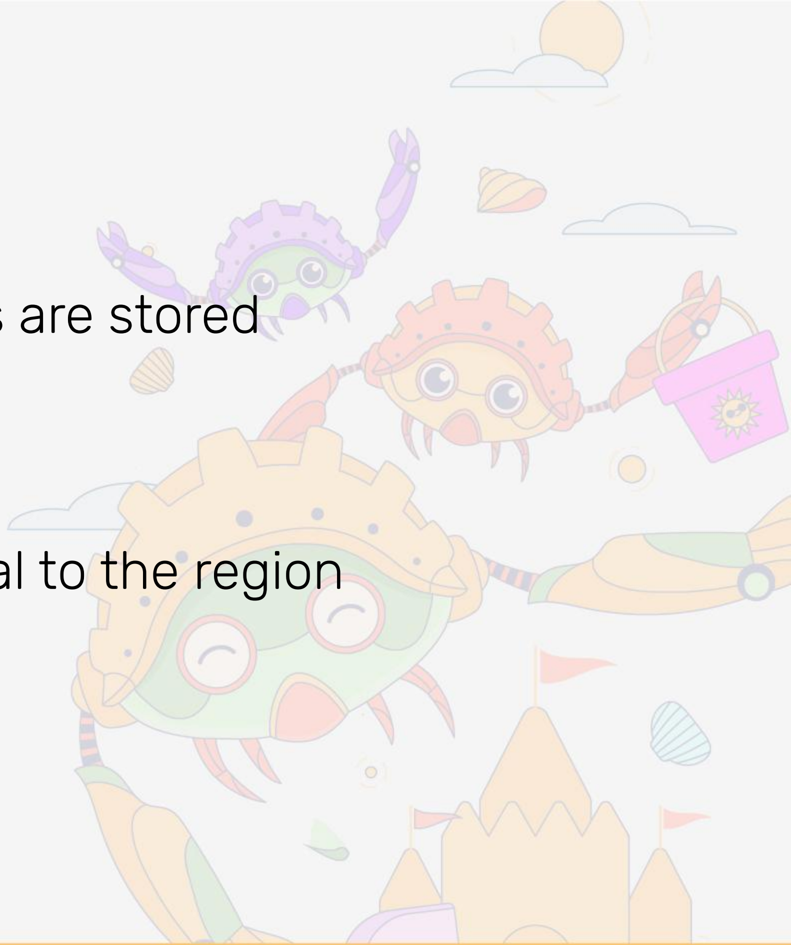

us-west-1
User A
User A
us-central-1
User B
User D
User D
us-east-1
User C
User E
User E
Request to access violet table for User C

ARCHITECTURE REQUIREMENTS

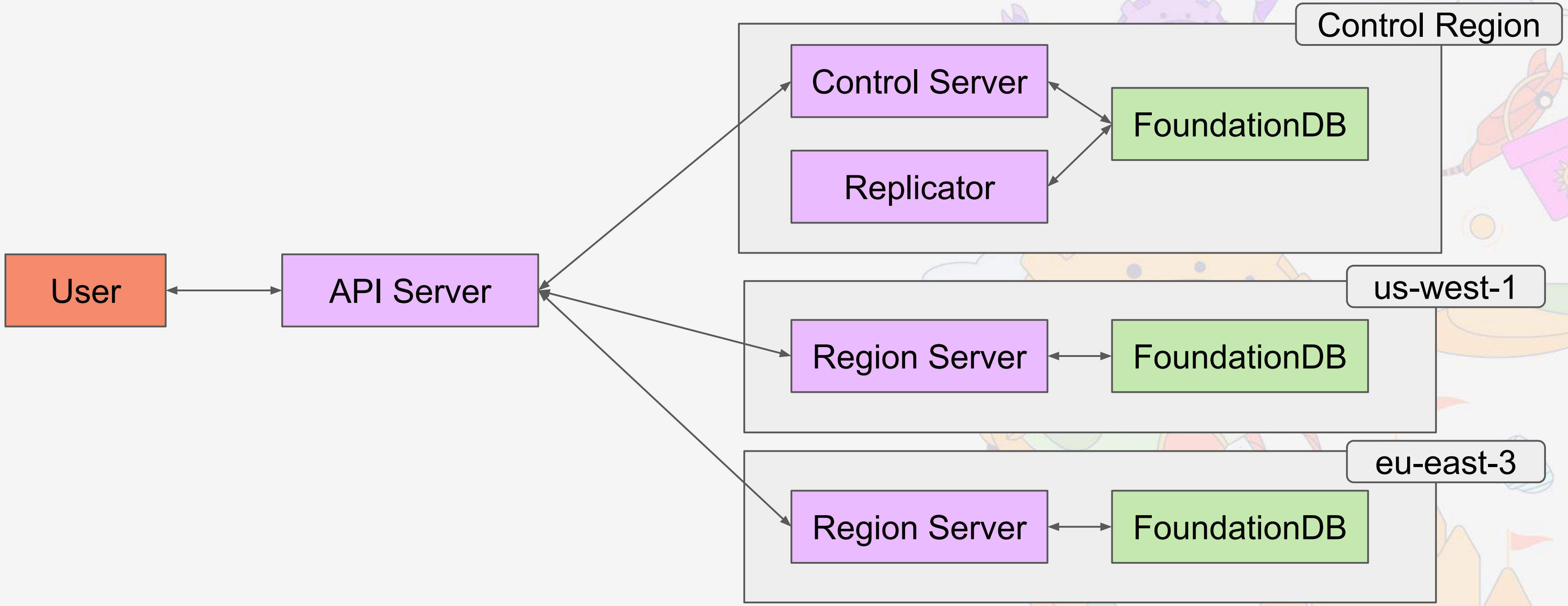
Global routing to find where tables are stored

Global schema management

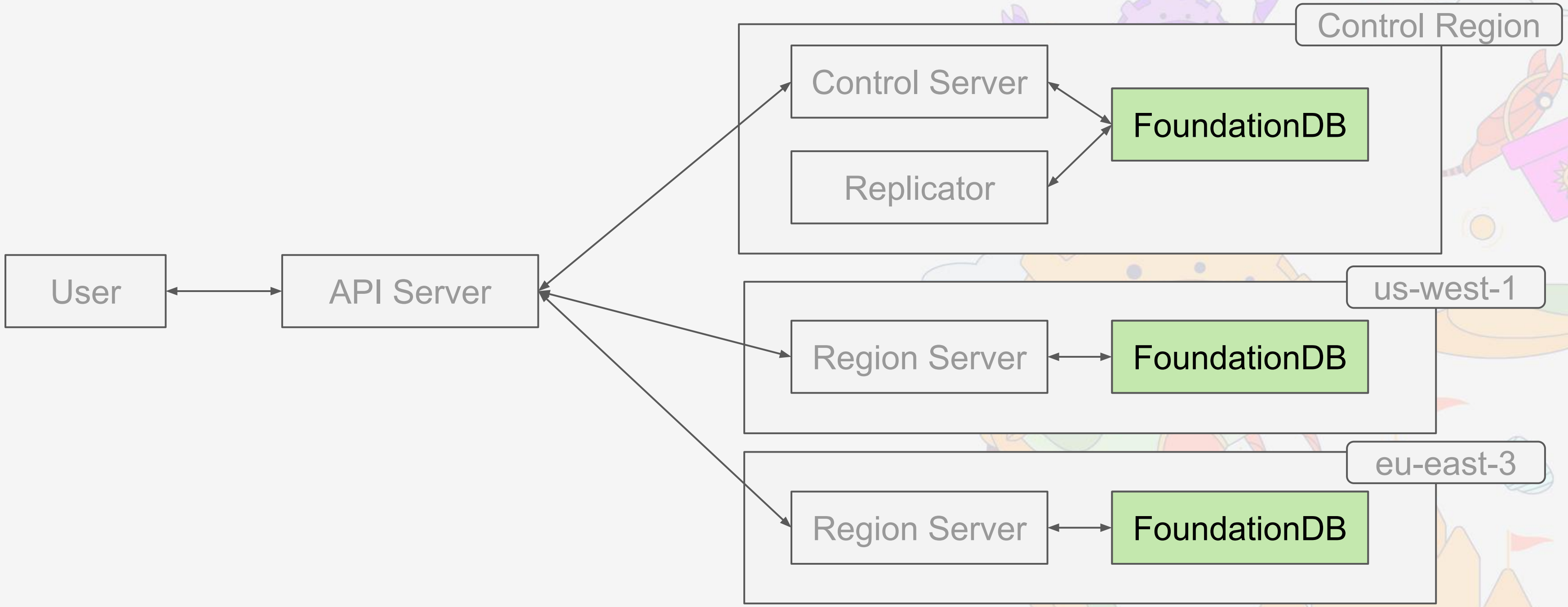
Almost all operations must be local to the region



RHINO ARCHITECTURE



RHINO ARCHITECTURE



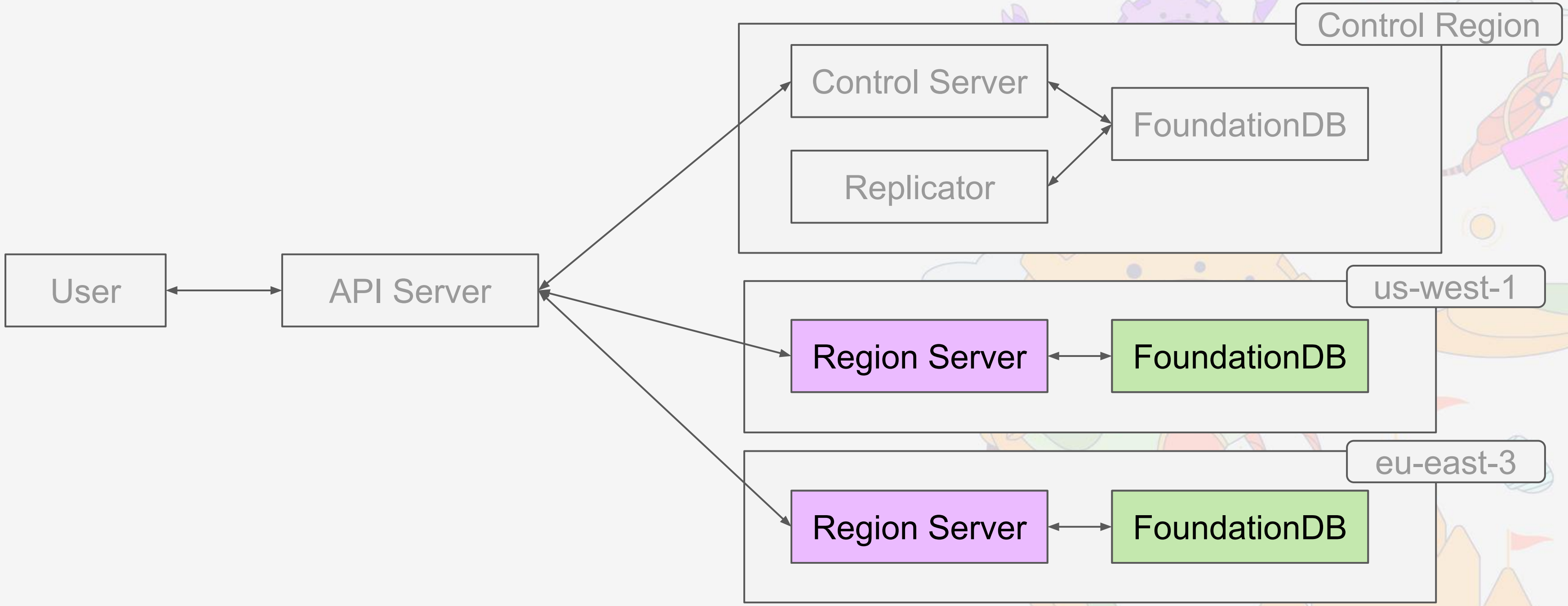
FOUNDATIONDB

Transactional key-value database

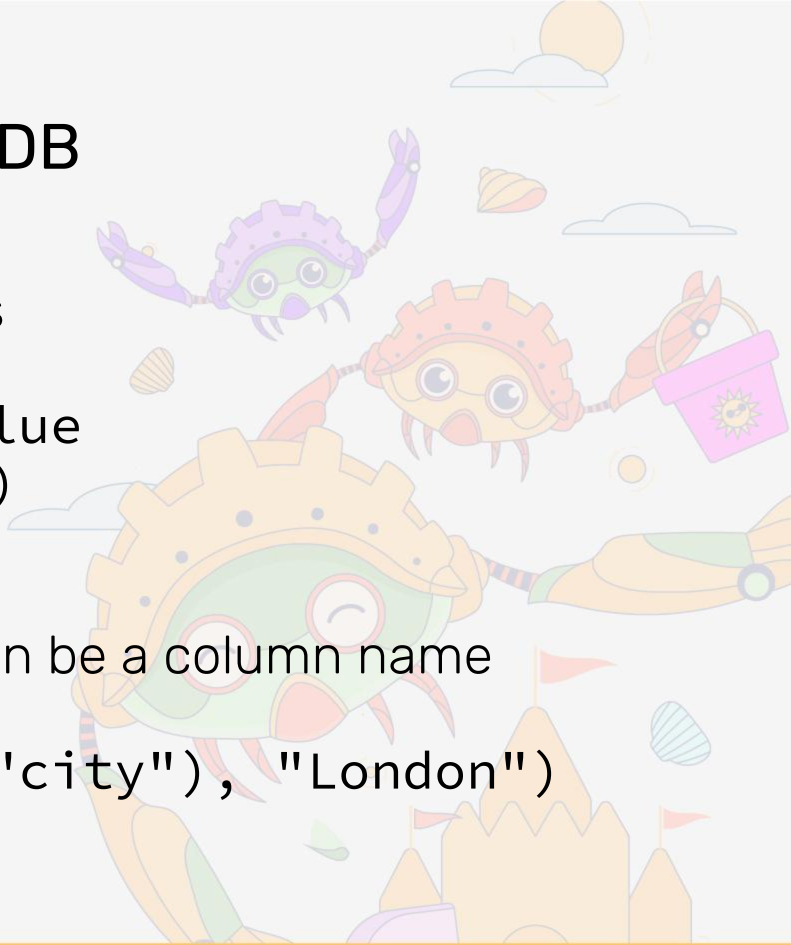
Focus on correctness

Building block for distributed systems

RHINO ARCHITECTURE



TABLES ON TOP OF FOUNDATIONDB



DynamoDB-style key-value tables

```
fn get(primary key) -> value
fn set(primary key, value)
```

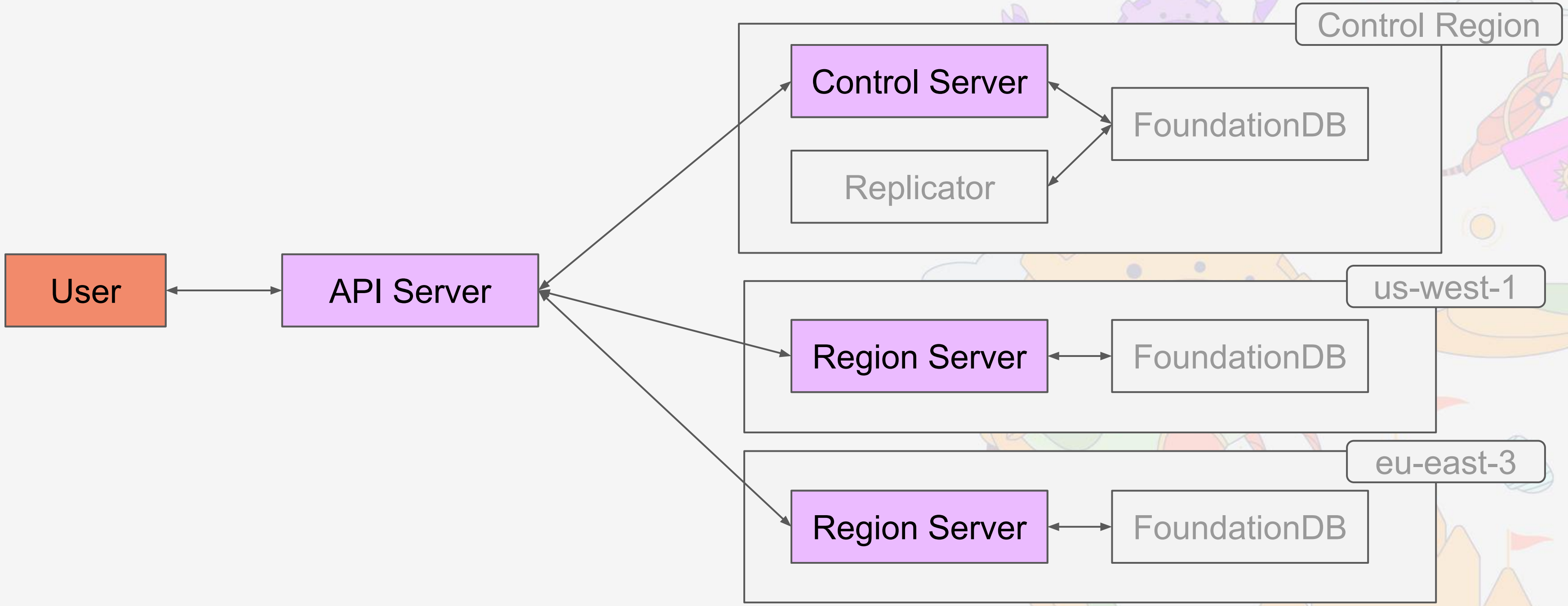
Primary key is a tuple of values
Last component of primary key can be a column name

```
set(("Nikita", "Lapkov", "city"), "London")
```

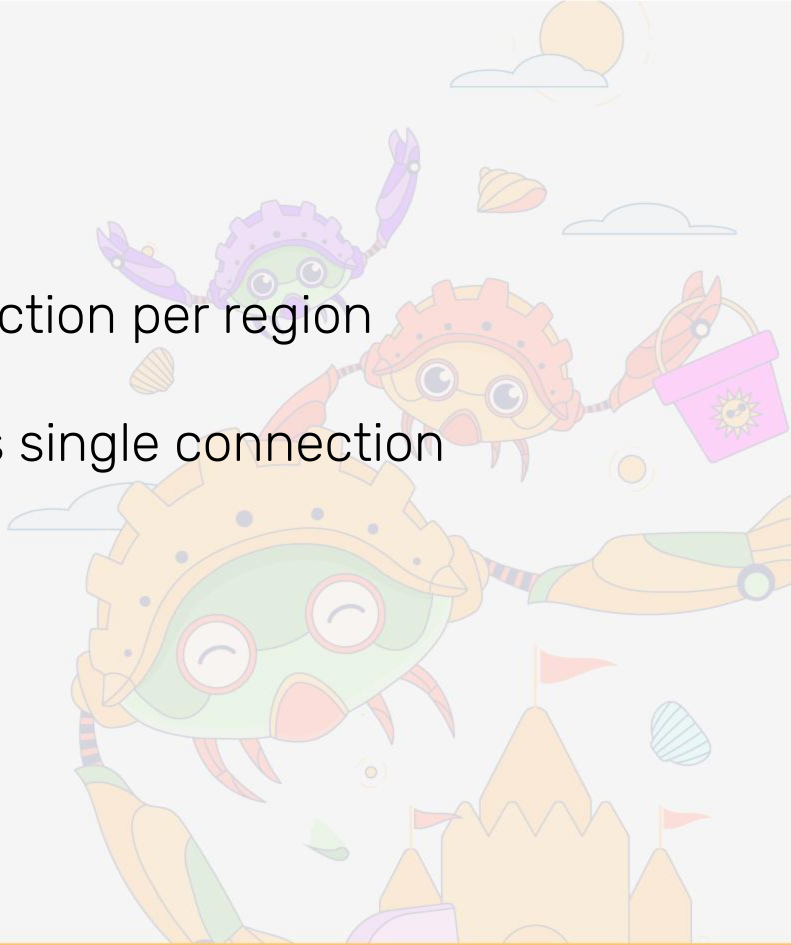
TABLES ON TOP OF FOUNDATIONDB

/tenants/TENANT_ID/table/USER_ID/MyTableName	
Key	**Value**
\x01**Nikita**\x01**Lapkov**\x01**city**\xff	"London"
\x01**Nikita**\x01**Lapkov**\x01**dog_preference**\xff	"big floof"
…	

RHINO ARCHITECTURE



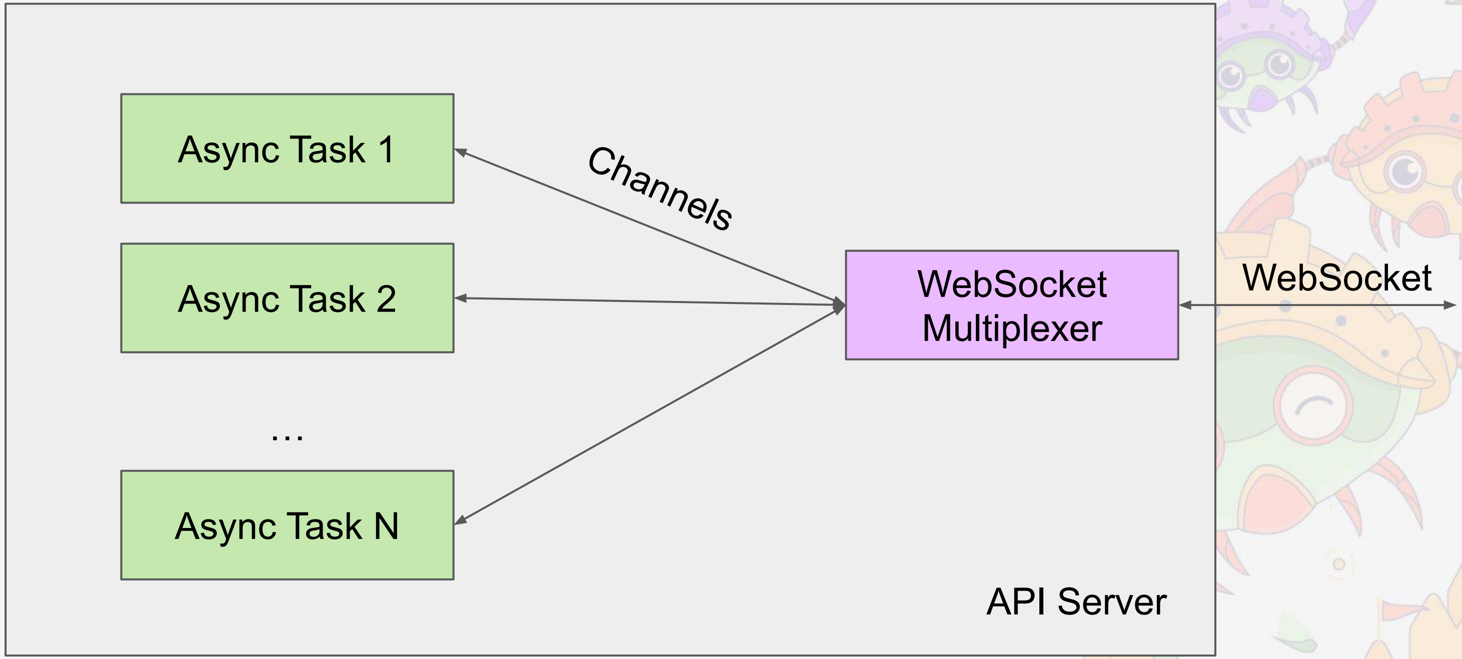
WEBSOCKET MULTIPLEXER



API Server establishes ***one*** connection per region

Requests are ***multiplexed*** on this single connection

WEBSOCKET MULTIPLEXER



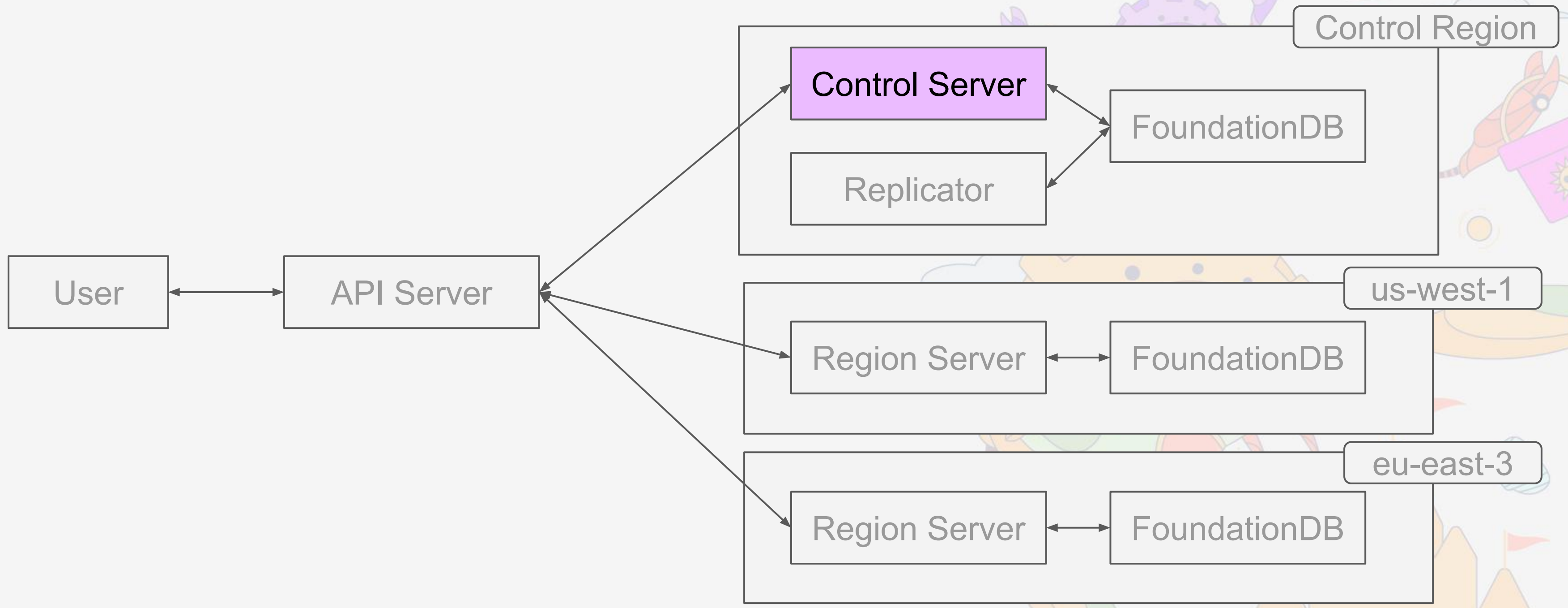
WEBSOCKET MULTIPLEXER

```
let mut handler = create_stream().await?;

handler.send(
    GetTenantIdByToken { token }
        .correlate(new_correlation_id()),
)?;

let response = handler.recv().await
```

RHINO ARCHITECTURE



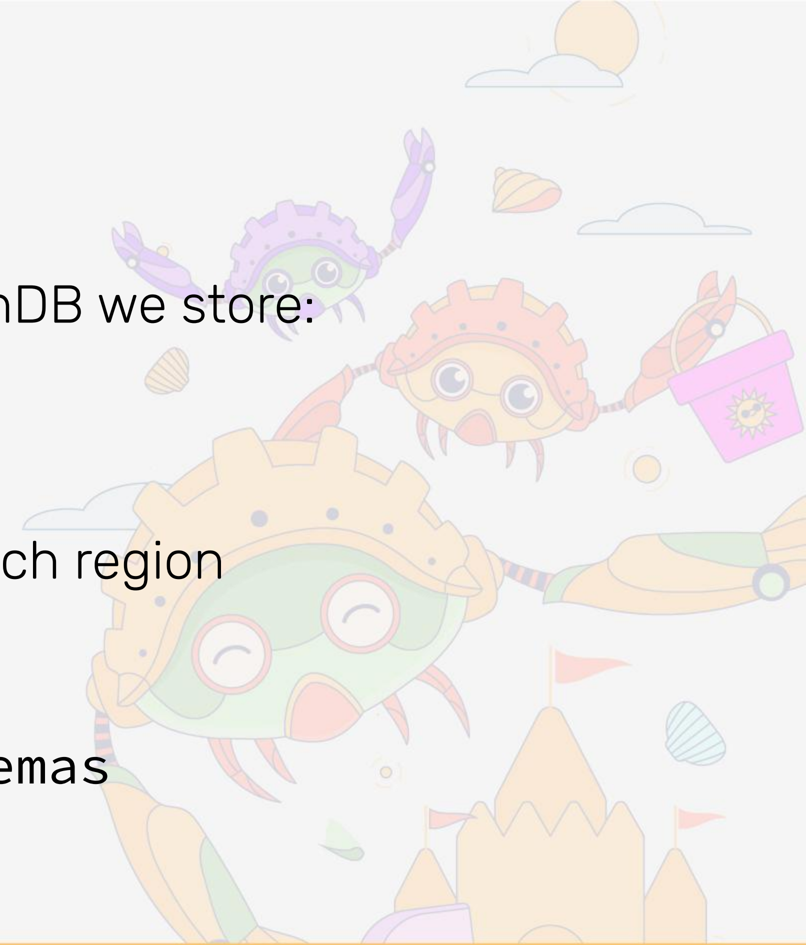
METADATA MANAGEMENT

Using tables on top of FoundationDB we store:

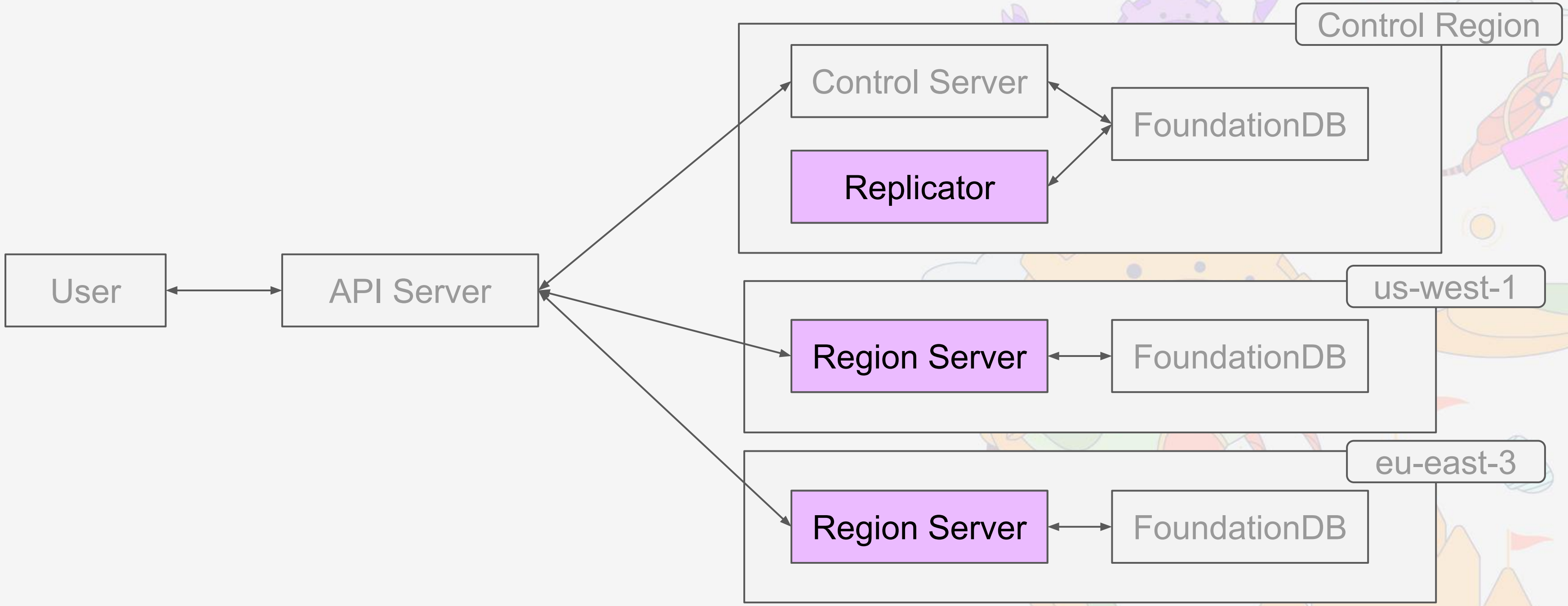
Valid auth tokens in the system
`auth/table/Tokens`

Which tables are available in each region
`routing/table/Shards`

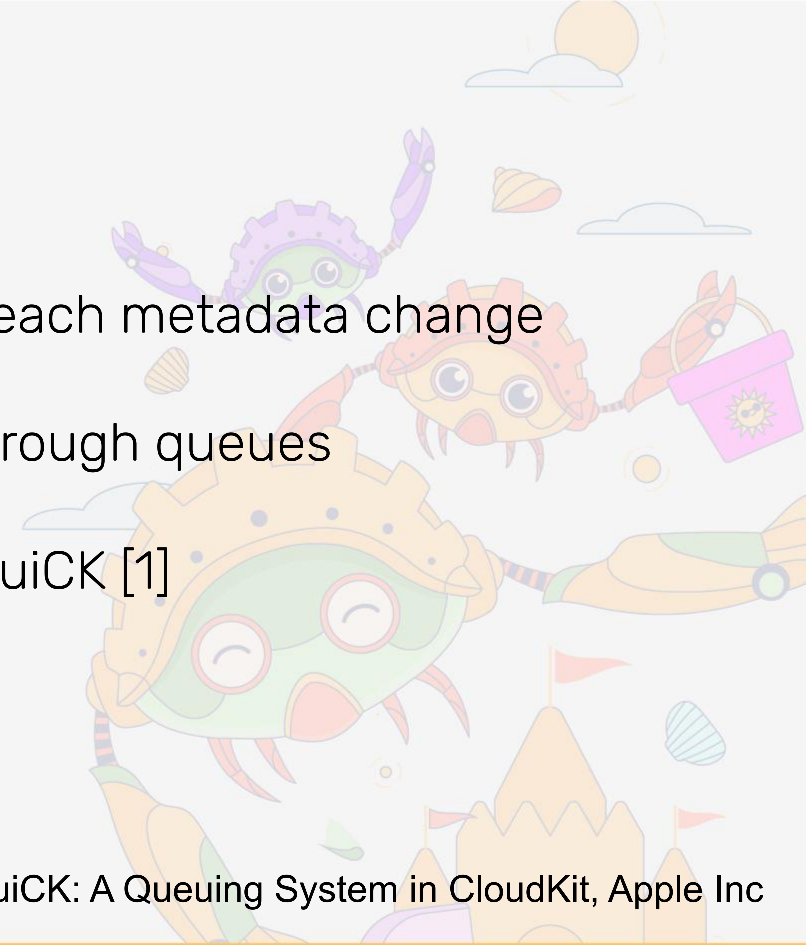
Table's schema
`schema/table/TableSchemas`



RHINO ARCHITECTURE



METADATA REPLICATION



New replication task is added on each metadata change

Replication tasks are managed through queues

Queues are implemented using QuiCK [1]

All replication is idempotent

[1] QuiCK: A Queuing System in CloudKit, Apple Inc

QUEUES USING QuiCK

```
fn enqueue(tuple of values)

fn dequeue(lease duration)
    -> (lease id, tuple of values)

fn complete(lease id)
```

[1] QuiCK: A Queuing System in CloudKit, Apple Inc

QUEUES USING QuiCK

/some/queue/path	
Key	Value

QUEUES USING QuiCK

```
enqueue("my queue item")
```

/some/queue/path	
Key	**Value**
(timestamp, random ULID)	"my queue item"

QUEUES USING QuiCK

`dequeue(lease duration)`

/some/queue/path	
Key	**Value**
(timestamp **+ lease duration**, random ULID)	"my queue item"

QUEUES USING QuiCK

```
dequeue(lease duration)
```

/some/queue/path	
Key	**Value**
(timestamp + lease duration, random ULID)	"my queue item"

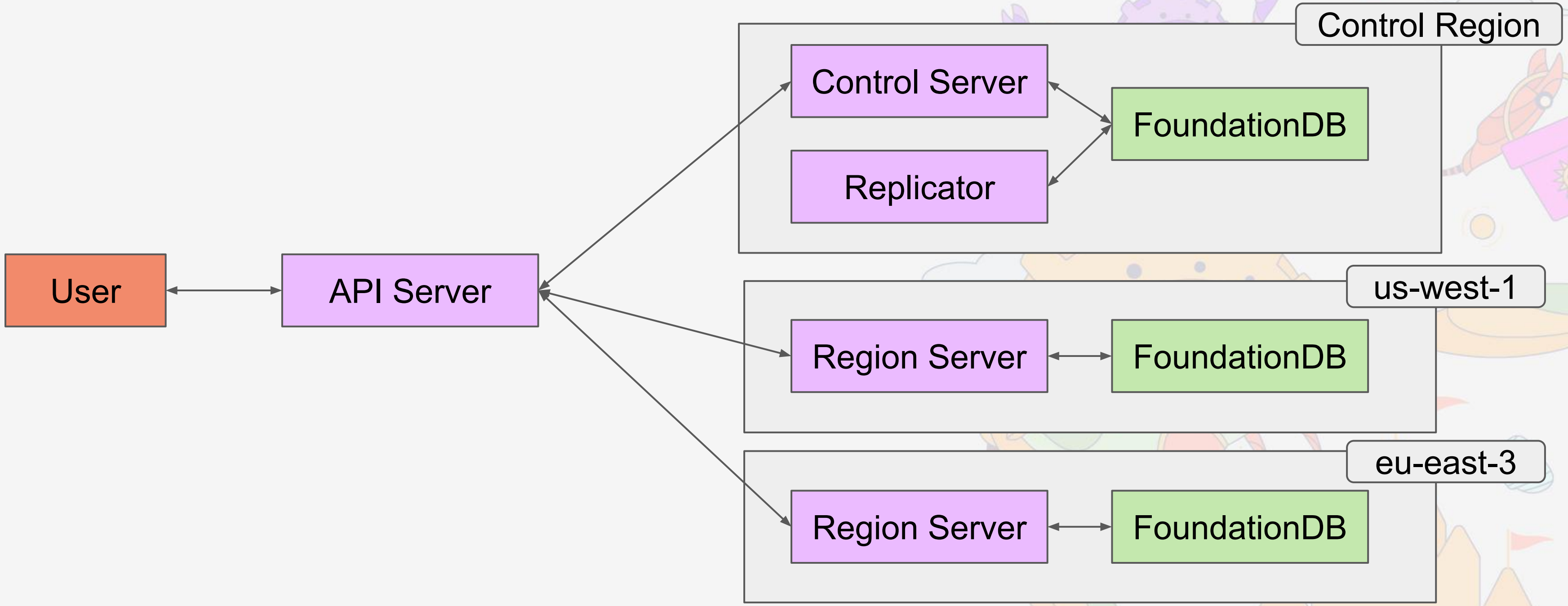
`-> lease id = (timestamp + lease duration, random ULID)`

QUEUES USING QuiCK

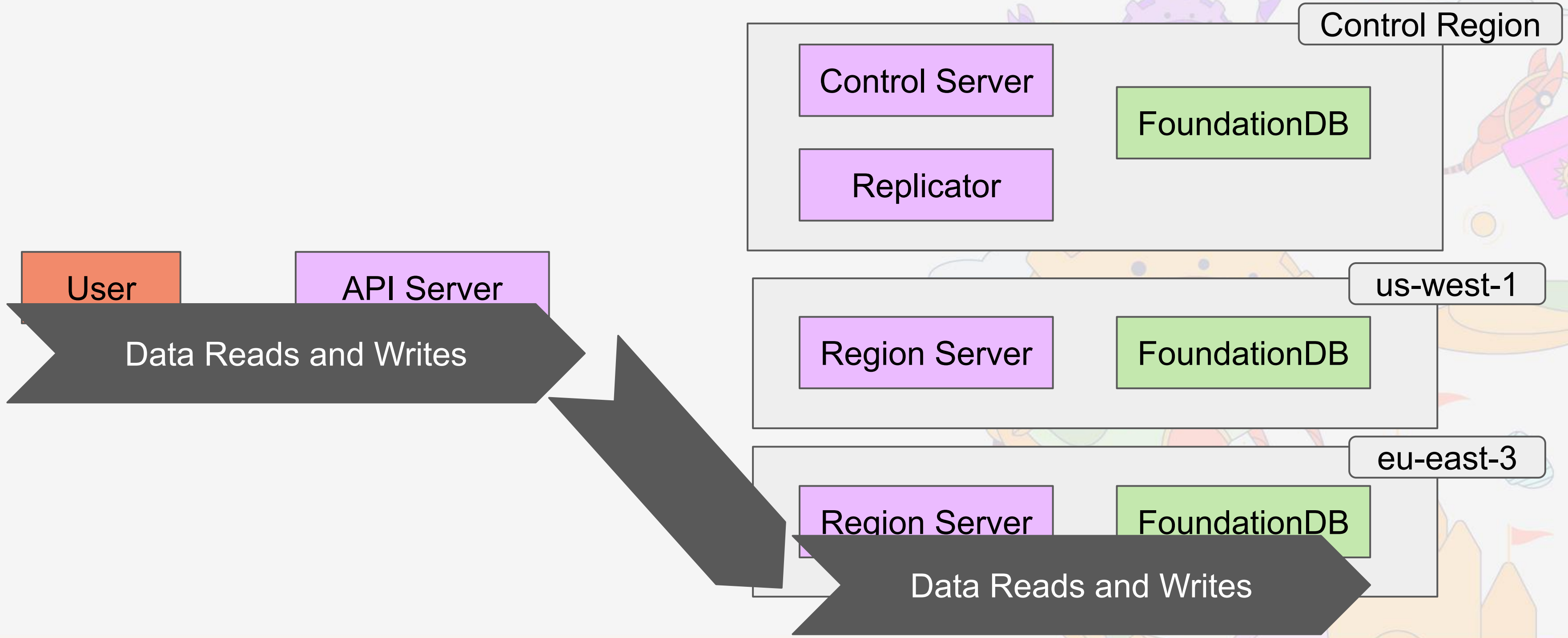
```
complete(lease id)
```

/some/queue/path	
Key	**Value**
(timestamp + lease duration, random ULID)	"my queue item"

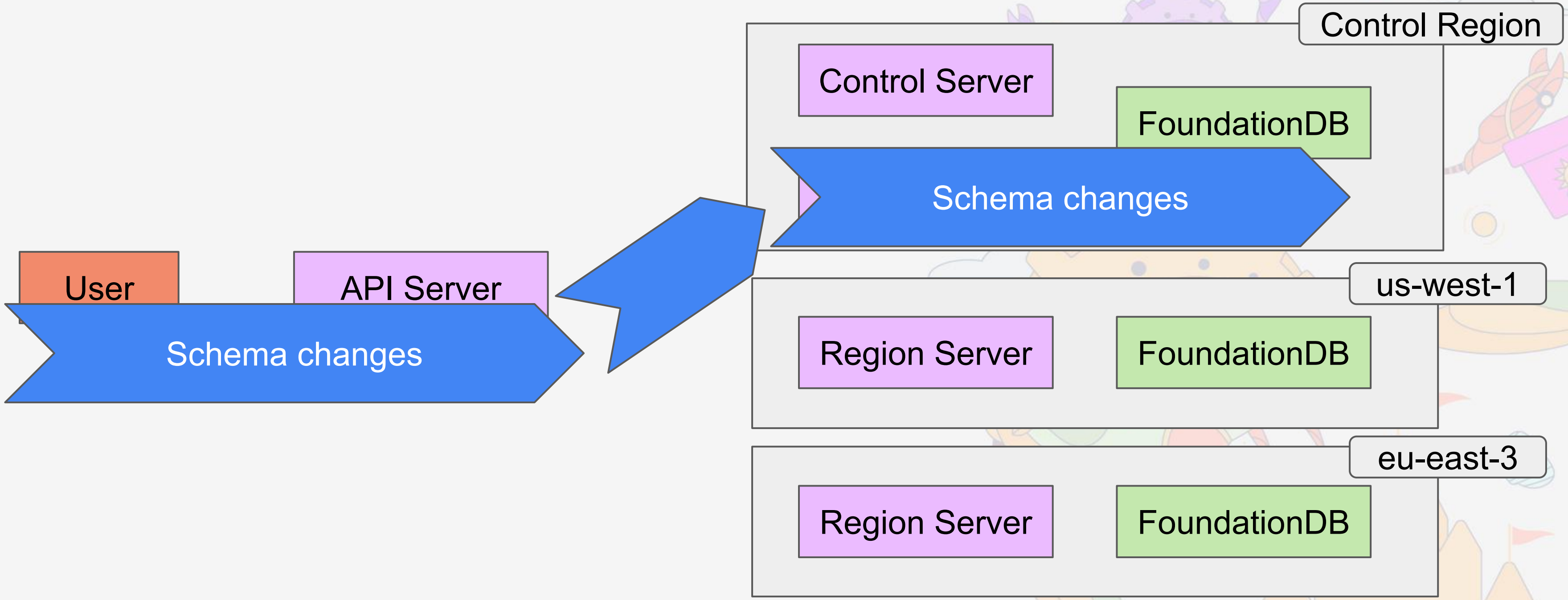
FINAL OVERVIEW



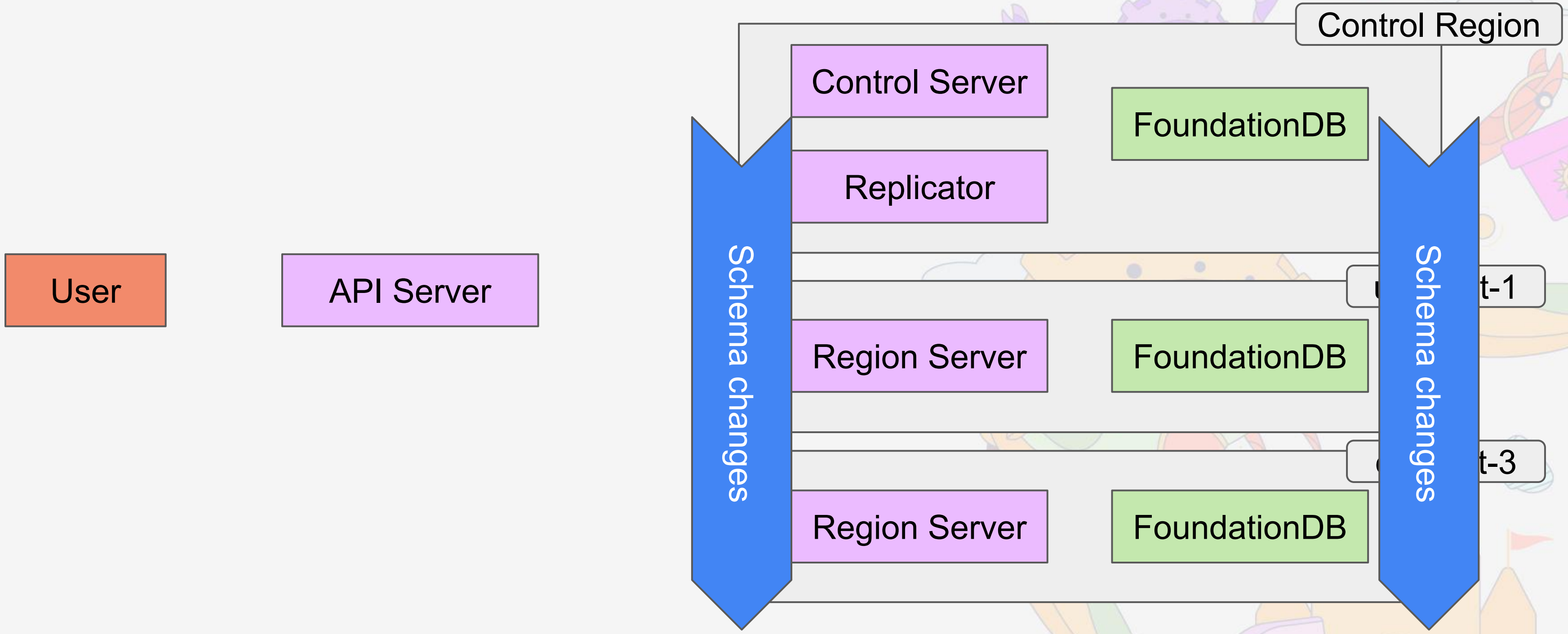
FINAL OVERVIEW



FINAL OVERVIEW



FINAL OVERVIEW



CONCLUSION

Automatic global routing

Global schema management

Low-latency data access

THANK YOU!

hi@laplab.me

github.com/laplab/rhino