Serverless Distributed Ledger

Chris Anderson

Director of Product, **Fauna**

Co-founder of Couchbase,
Architect of Couchbase Mobile

@jchris

FAUNA

Blockchain

Blockchain

Each new block carries a signature of the previous block. If you know the current block, you can read the entire history securely. Useful for data provenance, history tracking, etc.

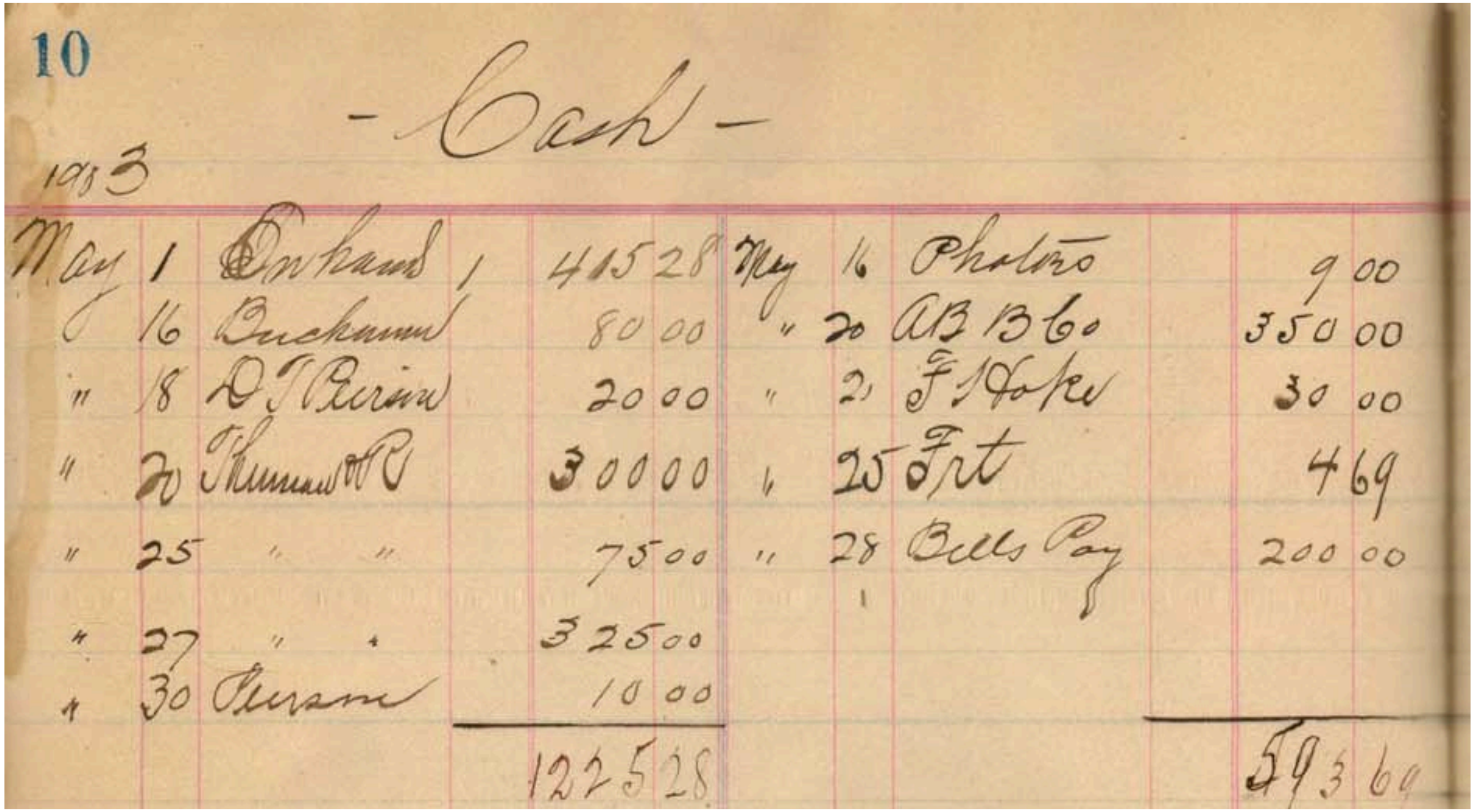
Combined with proof-of-work makes for an immutable log.

Blockchain

Blockchain

#noBlockchain

Ledger



10

- Cash -

1903

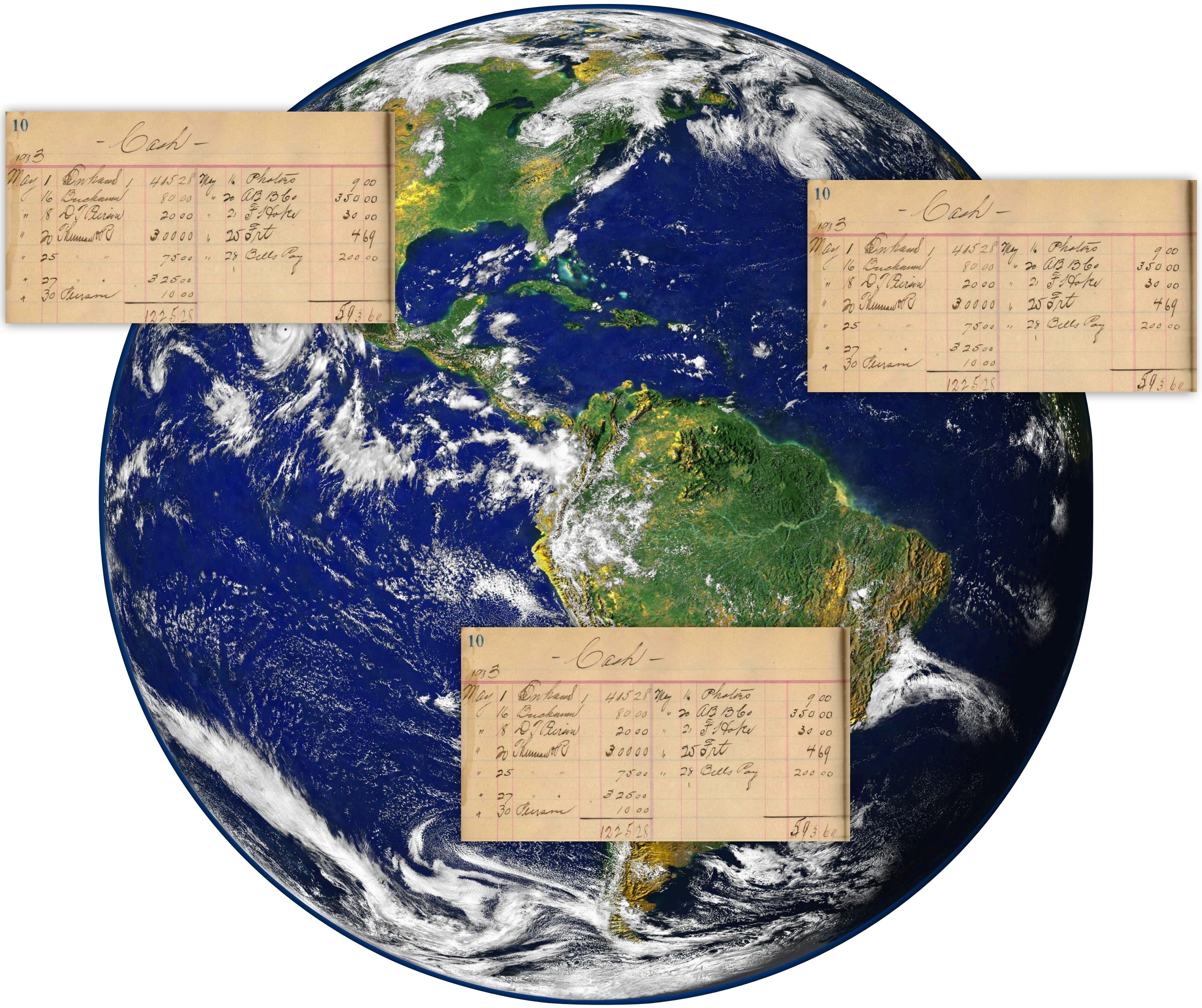
May	1	On hand	415 28	May	16	Photos	9 00
"	16	Buckman	80 00	"	20	A B B Co	350 00
"	18	D T Pierce	20 00	"	21	F Noke	30 00
"	20	Thurman & C	300 00	"	25	Frt	4 69
"	25	" "	75 00	"	28	Bills Pay	200 00
"	27	" "	325 00				
"	30	Pierson	10 00				
			1225 28				593 69

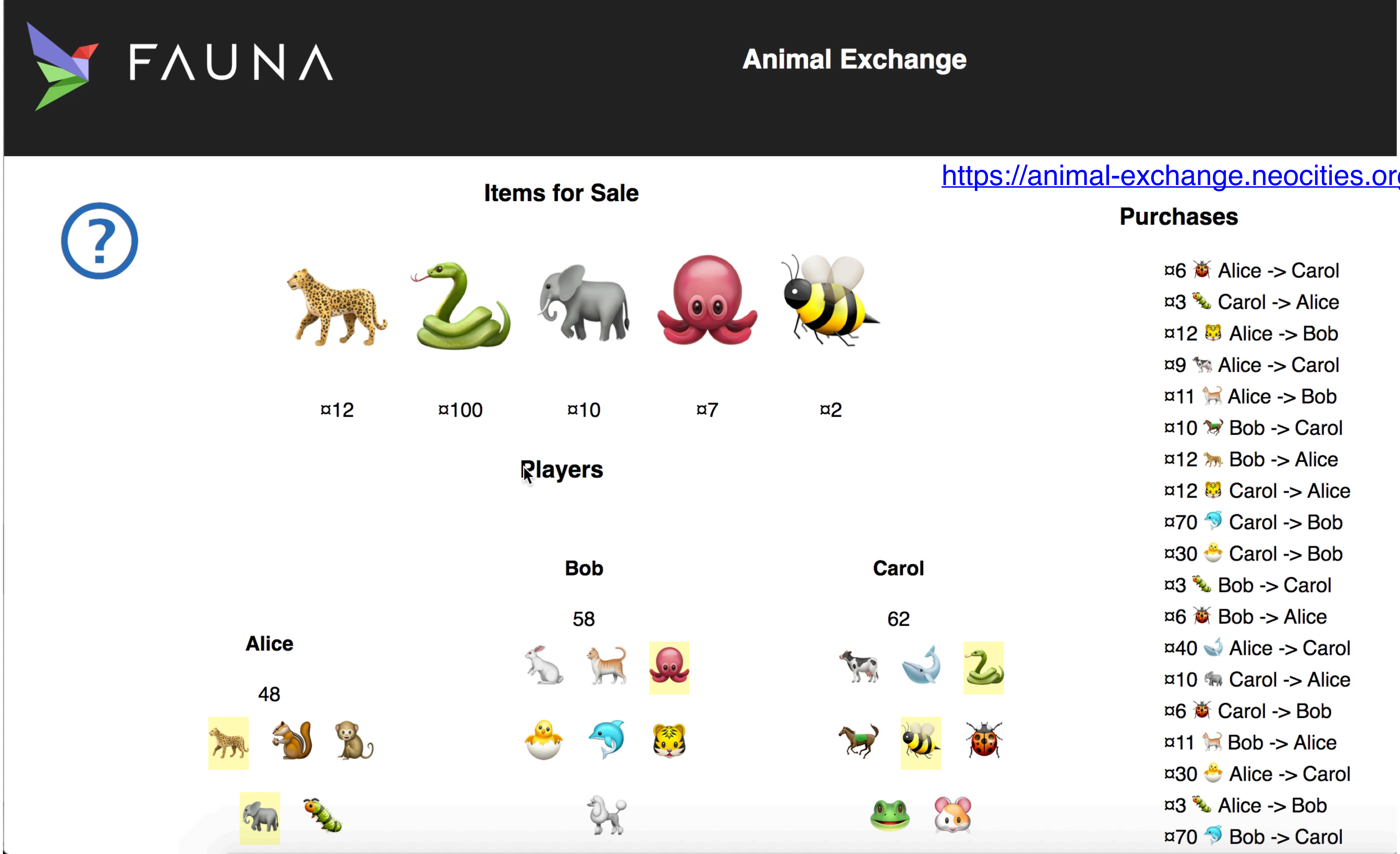
Ledger

A **distributed ledger** is a consensus of replicated, shared, and synchronized digital data geographically spread across multiple sites, countries, or institutions.

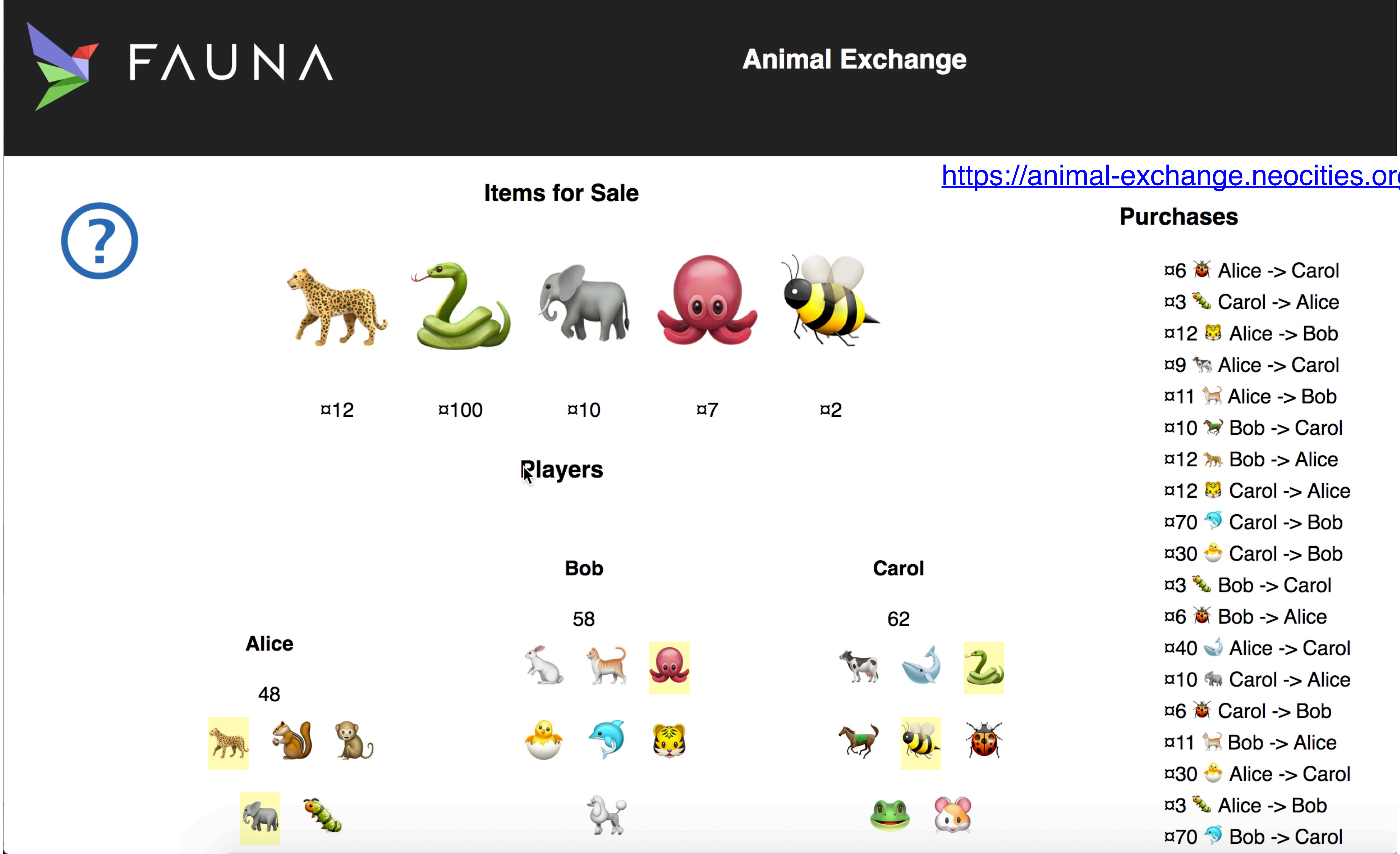
Distributed Ledger

Distributed Ledger

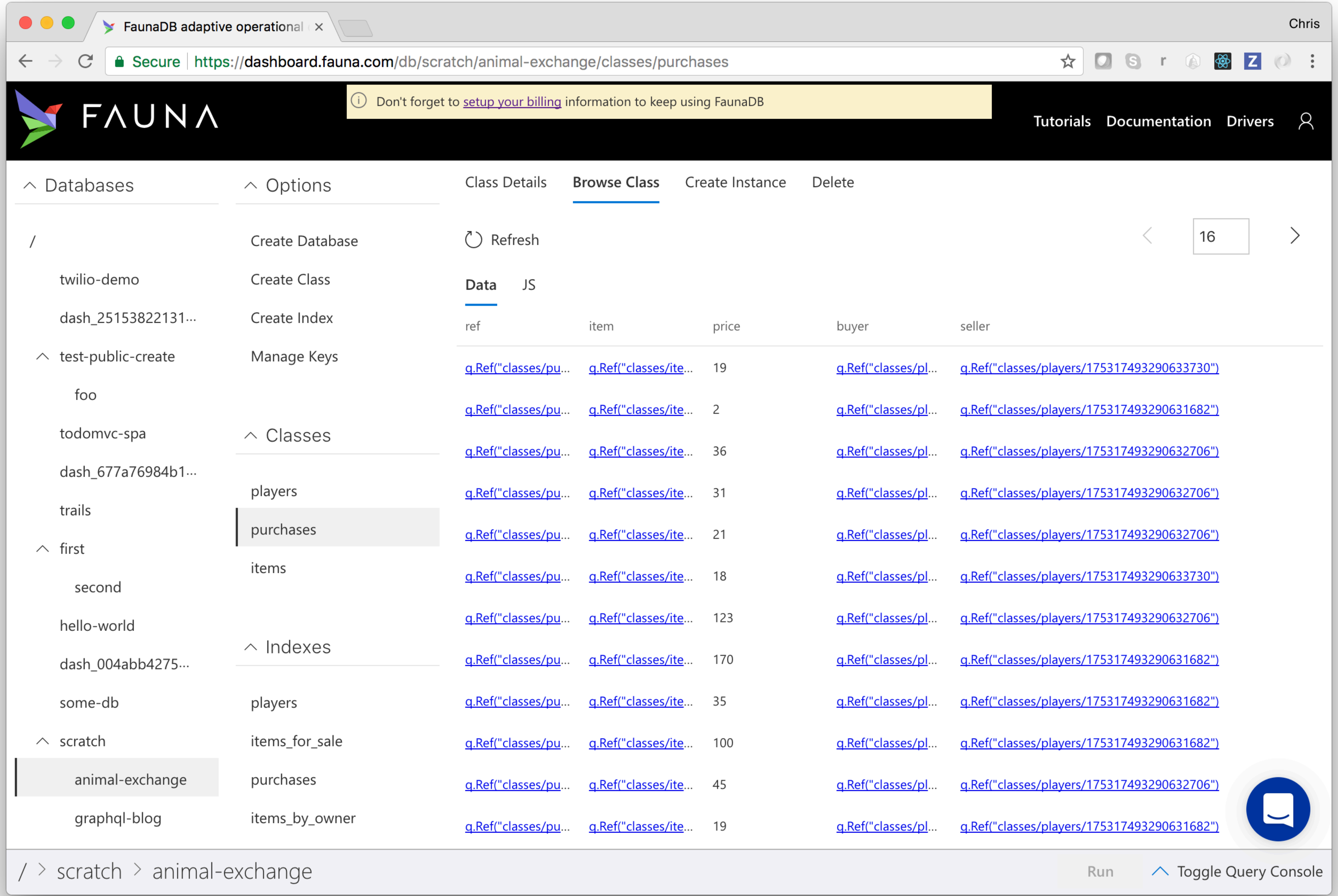




https://animal-exchange.neocities.org/



https://animal-exchange.neocities.org/


FaunaDB adaptive operational
Chris
Secure
https://dashboard.fauna.com/db/scratch/animal-exchange/classes/purchases
FAUNA
Don't forget to setup your billing information to keep using FaunaDB
Tutorials
Documentation
Drivers
Databases
/
twilio-demo
dash_25153822131...
test-public-create
foo
todomvc-spa
dash_677a76984b1...
trails
first
second
hello-world
dash_004abb4275...
some-db
scratch
animal-exchange
graphql-blog
Options
Create Database
Create Class
Create Index
Manage Keys
Classes
players
purchases
items
Indexes
players
items_for_sale
purchases
items_by_owner
Class Details
Browse Class
Create Instance
Delete
Refresh
16
Data
JS
ref
item
price
buyer
seller
q.Ref("classes/pu... q.Ref("classes/ite... 19 q.Ref("classes/pl... q.Ref("classes/players/175317493290633730")
q.Ref("classes/pu... q.Ref("classes/ite... 2 q.Ref("classes/pl... q.Ref("classes/players/175317493290631682")
q.Ref("classes/pu... q.Ref("classes/ite... 36 q.Ref("classes/pl... q.Ref("classes/players/175317493290632706")
q.Ref("classes/pu... q.Ref("classes/ite... 31 q.Ref("classes/pl... q.Ref("classes/players/175317493290632706")
q.Ref("classes/pu... q.Ref("classes/ite... 21 q.Ref("classes/pl... q.Ref("classes/players/175317493290632706")
q.Ref("classes/pu... q.Ref("classes/ite... 18 q.Ref("classes/pl... q.Ref("classes/players/175317493290633730")
q.Ref("classes/pu... q.Ref("classes/ite... 123 q.Ref("classes/pl... q.Ref("classes/players/175317493290632706")
q.Ref("classes/pu... q.Ref("classes/ite... 170 q.Ref("classes/pl... q.Ref("classes/players/175317493290631682")
q.Ref("classes/pu... q.Ref("classes/ite... 35 q.Ref("classes/pl... q.Ref("classes/players/175317493290631682")
q.Ref("classes/pu... q.Ref("classes/ite... 100 q.Ref("classes/pl... q.Ref("classes/players/175317493290631682")
q.Ref("classes/pu... q.Ref("classes/ite... 45 q.Ref("classes/pl... q.Ref("classes/players/175317493290632706")
q.Ref("classes/pu... q.Ref("classes/ite... 19 q.Ref("classes/pl... q.Ref("classes/players/175317493290631682")
/ scratch animal-exchange
Run
Toggle Query Console

Getting Serious

Getting Serious

Global Consensus

Getting Serious

Global Consensus

Distributed Ledger Transaction

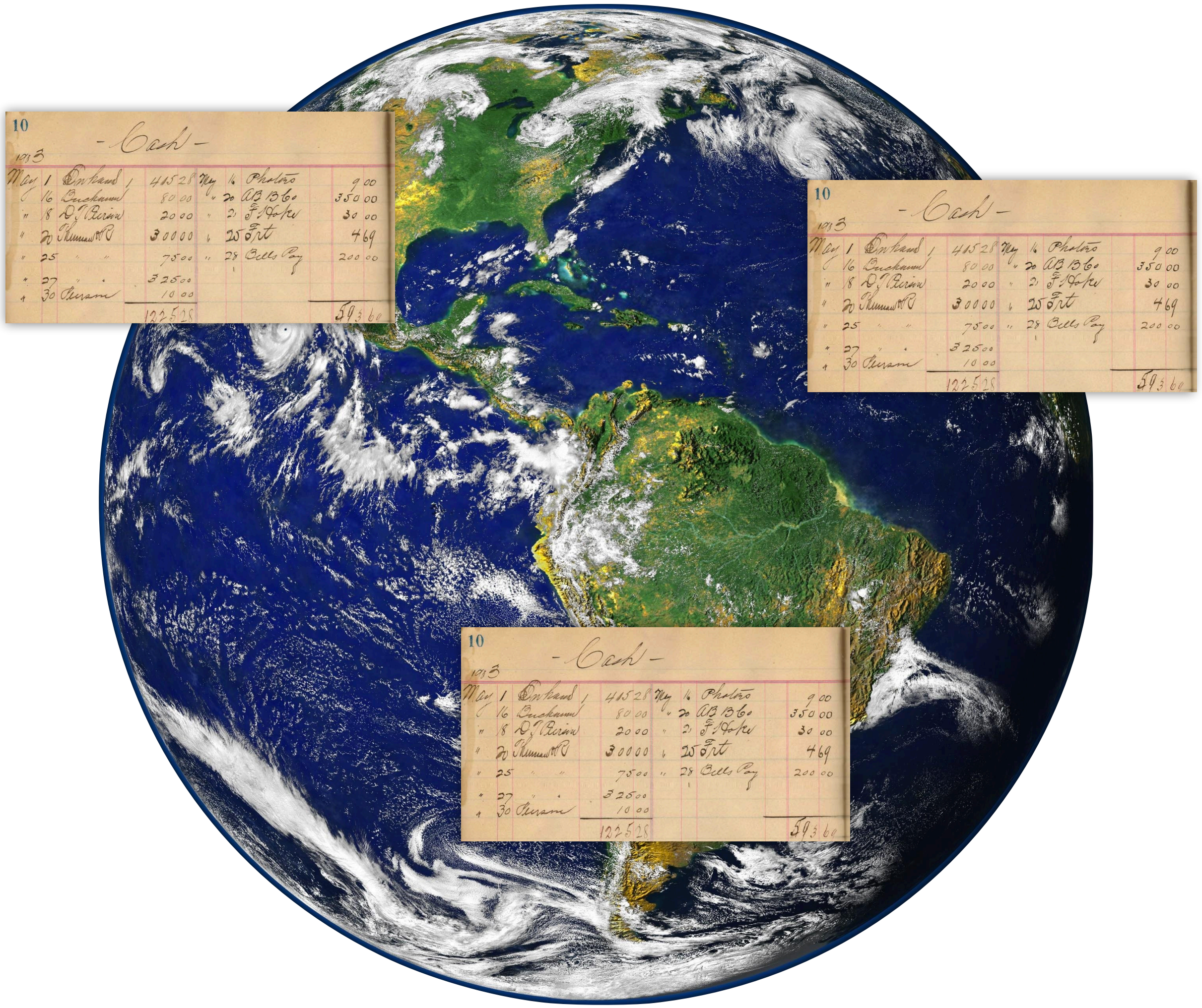
Getting Serious

Global Consensus

Distributed Ledger Transaction

Serverless Security Model

Global Consensus



Global Consensus

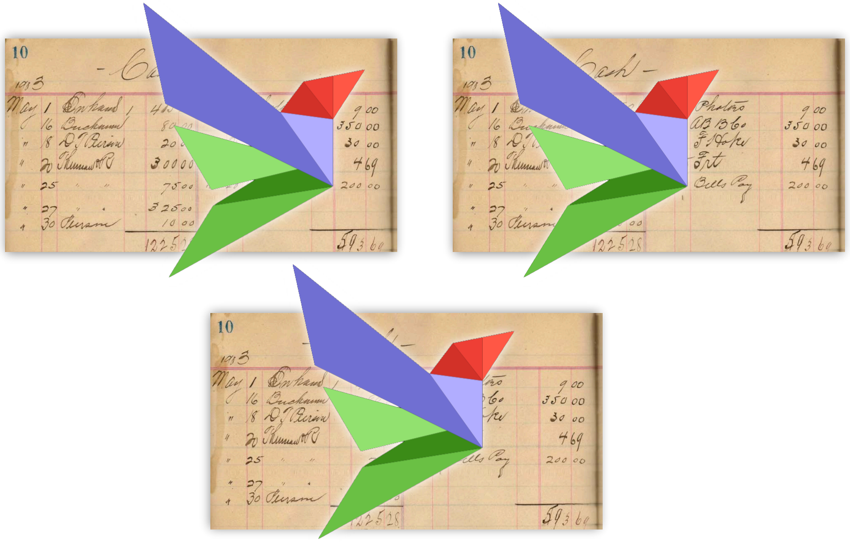
Global Consensus

Each ledger member runs a high-availability FaunaDB cluster.

Architecture

Each high-availability FaunaDB cluster contains a full copy of the dataset.

This can be partitioned for horizontal scaling.



Architecture

Each node runs FaunaDB Enterprise.

Implemented in Scala, delivered as a .jar packaged for your environment.

The cluster can be dynamically resized while serving traffic.

Calvin Protocol

https://fauna.com/blog/distributed-consistency-at-scale-spanner-vs-calvin

FaunaDB uses a distributed write-ahead-log to provide ACID transactions. In the presence of write conflicts transactions may be retried internally.

Transactions commit across all datacenters.

Throughput oriented, each Calvin log segment may contain multiple transactions.

Calvin Protocol

https://fauna.com/blog/distributed-consistency-at-scale-spanner-vs-calvin

Fauna, Inc. [US] https://fauna.com/blog/distributed-consistency-at-scale-spanner-vs-calvin

Spanner vs. Calvin: distributed consistency at scale

Daniel J. Abadi
April 06, 2017

Daniel J. Abadi is an Associate Professor at Yale University. He does research primarily in database system architecture and implementation. He received a Ph.D. from MIT and a M.Phil from Cambridge.

Introduction

In 2012, two research papers were published that described the design of geographically replicated, consistent, ACID compliant, transactional

https://github.com/fauna/animal-exchange/blob/master/src/model.js

Ledger Transaction

```
client.query(
    q.Let({
      buyer : q.Get(player.ref),
      item : q.Get(item.ref)
    }, q.Let({
      isForSale : q.Select(["data", "for_sale"], q.Var("item")),
      itemPrice : q.Select(["data", "price"], q.Var("item")),
      buyerBalance : q.Select(["data", "credits"], q.Var("buyer")),
      seller : q.Get(q.Select(["data", "owner"], q.Var("item")))
    }, q.If(q.Not(q.Var("isForSale")),
        "purchase failed: item not for sale",
        q.If(q.Equals(q.Select("ref", q.Var("buyer")), q.Select("ref", q.Var("seller"))),
          // buyer = seller, remove item from sale
          q.Do(
            q.Update(q.Select("ref", q.Var("item")), {
              data : {
                for_sale : false
              }
            }),
            "item removed from sale"
          ),
          // check balance
          q.If(q.LT(q.Var("buyerBalance"), q.Var("itemPrice")),
            "purchase failed: insufficient funds",
            // all clear! record the purchase, update the buyer, seller and item.

// all clear! record the purchase, update the buyer, seller and item.
q.Do(
  q.Create(q.Class("purchases"), {
    data : {
      item : q.Select("ref", q.Var("item")),
      price : q.Var("itemPrice"),
      buyer : q.Select("ref", q.Var("buyer")),
      seller : q.Select("ref", q.Var("seller"))
    }
  }),
  q.Update(q.Select("ref", q.Var("buyer")), {
    data : {
      credits : q.Subtract(q.Var("buyerBalance"), q.Var("itemPrice"))
    }
  }),
  q.Update(q.Select("ref", q.Var("seller")), {
    data : {
      credits : q.Add(q.Select(["data", "credits"], q.Var("seller")), q.Var("itemPrice"))
    }
  }),
  q.Update(q.Select("ref", q.Var("item")), {
    data : {
      owner : q.Select("ref", q.Var("buyer")),
      for_sale : false
    }
  }),
  "purchase success" ) ) ) ))) )
```

https://github.com/fauna/animal-exchange/blob/master/src/model.js

Ledger Transaction

```
client.query(
     q.Let({
       buyer : q.Get(player.ref),
       item : q.Get(item.ref)
     }, q.Let({
       isForSale : q.Select(["data", "for_sale"], q.Var("item")),
       itemPrice : q.Select(["data", "price"], q.Var("item")),
       buyerBalance : q.Select(["data", "credits"], q.Var("buyer")),
       seller : q.Get(q.Select(["data", "owner"], q.Var("item")))
     }, q.If(q.Not(q.Var("isForSale")),
         "purchase failed: item not for sale",
         q.If(q.Equals(q.Select("ref", q.Var("buyer")), q.Select("ref", q.Var("seller"))),
           // buyer = seller, remove item from sale
           q.Do(
             q.Update(q.Select("ref", q.Var("item")), {
               data : {
                 for_sale : false
               }
             }),
             "item removed from sale"
           ),
           // check balance
           q.If(q.LT(q.Var("buyerBalance"), q.Var("itemPrice")),
             "purchase failed: insufficient funds",
             // all clear! record the purchase, update the buyer, seller and item.
```

```
// all clear! record the purchase, update the buyer, seller and item.
 q.Do(
   q.Create(q.Class("purchases"), {
     data : {
       item : q.Select("ref", q.Var("item")),
       price : q.Var("itemPrice"),
       buyer : q.Select("ref", q.Var("buyer")),
       seller : q.Select("ref", q.Var("seller"))
     }
   }),
   q.Update(q.Select("ref", q.Var("buyer")), {
     data : {
       credits : q.Subtract(q.Var("buyerBalance"), q.Var("itemPrice"))
     }
   }),
   q.Update(q.Select("ref", q.Var("seller")), {
     data : {
       credits : q.Add(q.Select(["data", "credits"], q.Var("seller")), q.Var("itemPrice
     }
   }),
   q.Update(q.Select("ref", q.Var("item")), {
     data : {
       owner : q.Select("ref", q.Var("buyer")),
       for_sale : false
     }
   }),
   "purchase success" ) ) ) ))) )
```

https://github.com/fauna/animal-exchange/blob/master/src/model.js

Ledger Transaction

Ensure item is for sale

Buyer != seller

Check buyer balance

```
client.query(
    q.Let({
      buyer : q.Get(player.ref),
      item : q.Get(item.ref)
    }, q.Let({
      isForSale : q.Select(["data", "for_sale"], q.Var("item")),
      itemPrice : q.Select(["data", "price"], q.Var("item")),
      buyerBalance : q.Select(["data", "credits"], q.Var("buyer")),
      seller : q.Get(q.Select(["data", "owner"], q.Var("item")))
    }, q.If(q.Not(q.Var("isForSale")),
        "purchase failed: item not for sale",
        q.If(q.Equals(q.Select("ref", q.Var("buyer")), q.Select("ref", q.Var("seller"))),
          // buyer == seller, remove item from sale
          q.Do(
            q.Update(q.Select("ref", q.Var("item")), {
              data : {
                for_sale : false
              }
            }),
            "item removed from sale"
          ),
          // check balance
          q.If(q.LT(q.Var("buyerBalance"), q.Var("itemPrice")),
            "purchase failed: insufficient funds",
            // all clear! record the purchase, update the buyer, seller and item.
```

Write a purchase record

Deduct from buyer balance

Add to seller balance

Update item owner

```
// all clear! record the purchase, update the buyer, seller and item.
q.Do(
  q.Create(q.Class("purchases"), {
    data : {
      item : q.Select("ref", q.Var("item")),
      price : q.Var("itemPrice"),
      buyer : q.Select("ref", q.Var("buyer")),
      seller : q.Select("ref", q.Var("seller"))
    }
  }),
  q.Update(q.Select("ref", q.Var("buyer")), {
    data : {
      credits : q.Subtract(q.Var("buyerBalance"), q.Var("itemPrice"))
    }
  }),
  q.Update(q.Select("ref", q.Var("seller")), {
    data : {
      credits : q.Add(q.Select(["data", "credits"], q.Var("seller")), q.Var("itemPrice
    }
  }),
  q.Update(q.Select("ref", q.Var("item")), {
    data : {
      owner : q.Select("ref", q.Var("buyer")),
      for_sale : false
    }
  }),
  "purchase success" ) ) ) ))) )
```

Update Buyer Balance

```
q.Update(q.Select("ref", q.Var("buyer")), {
  data : {
    credits : q.Subtract(q.Var("buyerBalance"), q.Var("itemPrice"))
  }
})
```

Queries are composed on the client, and sent to the server as an abstract syntax tree encoded as JSON.

https://github.com/fauna/animal-exchange/blob/master/src/model.js

Client Library in Your Language

```
update( select('ref', var('buyer')),
 data: {
  credits: subtract(var('buyerBalance'), var('itemPrice'))
 })
```

Java Javascript Scala Ruby C# Python Go Swift

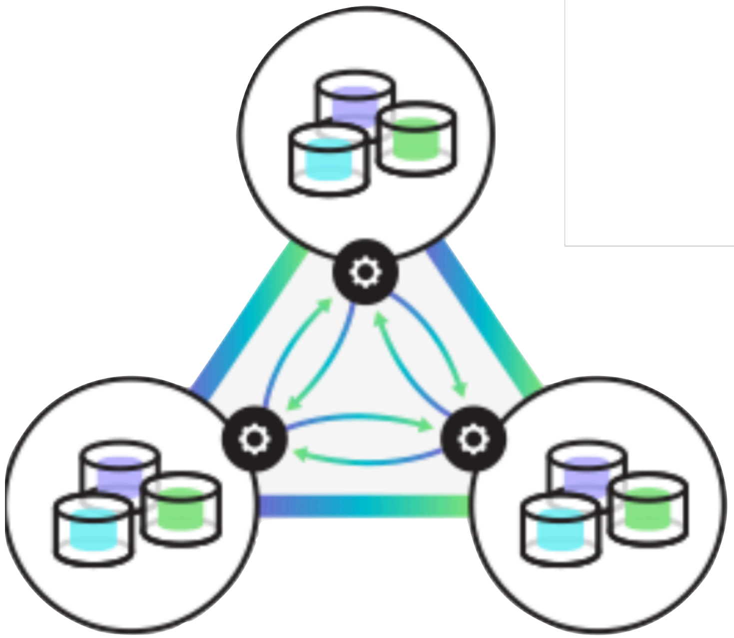
ACID Transactions

Not just for distributed ledgers

Enhance developer productivity

Simplify applications

Address mission-critical use cases at scale

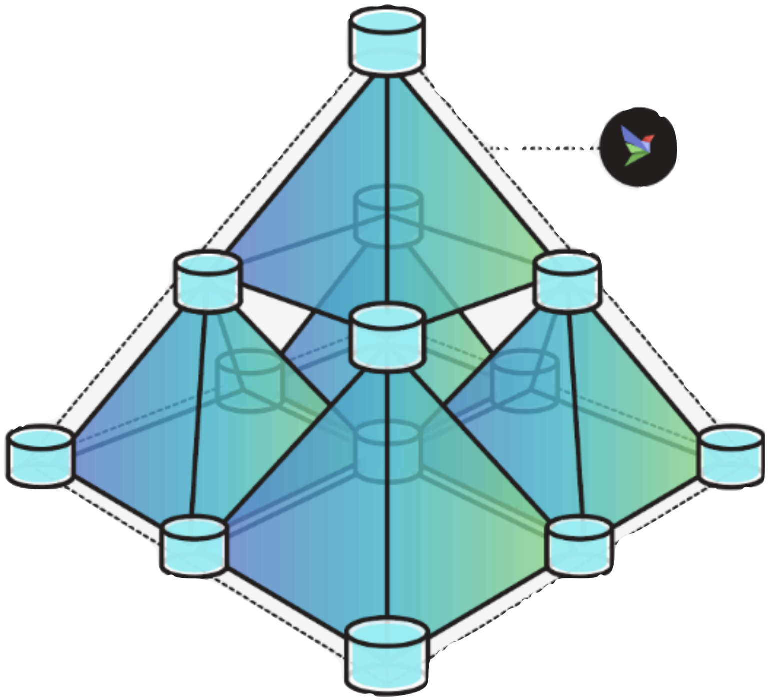


Serverless Security

Layered access approach.

Lambdas use keys that only have privileges to run predefined functions.

User defined functions use keys that cannot modify schema rules or old temporal snapshots.



AWS Lambda

Function as a Service

JavaScript runs in response to events.

Authenticate users, process resources, etc.

For distributed ledger, this is the code that reacts to user events by submitting queries that call predefined functions.

Code can run on premise.

Predefined Function

FaunaDB user defined functions API where query fragments can be stored and executed by other queries.

Only objects with the call permission on a function can call it, so in the distributed ledger use case the Lambdas are granted keys that authenticate into the access-control graph in a place where they only have permissions to call the UDF.

```
{
  "call": { "function": "create_entry" },
  "arguments": [
    "First Post Title",
    “This is my first blog post!" ]
}
```

Temporal Data

FaunaDB stores data in temporal snapshots, and has APIs for updating old snapshots, for instance to fix data-entry mistakes. Old snapshots are cleaned up after a configurable TTL.

https://fauna.com/tutorials/timeline
https://fauna.com/blog/time-traveling-databases

For distributed ledger, the UDFs run in a role limited to the current snapshot, so any snapshot editing can only be done from an administrative interface.

FaunaDB: Serverless Database Table Stakes

Runs in the cloud(s)

- “Not my server, not my problem, that’s what I say.” / “Around the world.”

Friendly to JSON / NoSQL

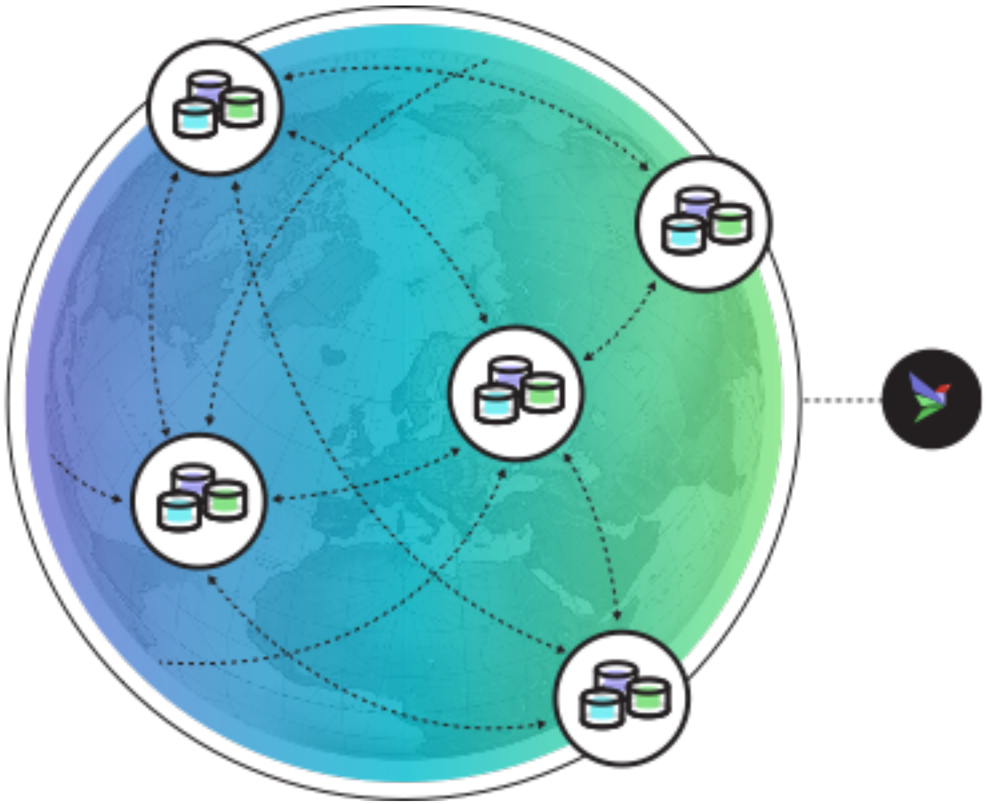
- Schema enforcement is optional, we ❤ rich nested data structures

Relational queries and constraints

- Proper database features are BACK and they SCALE

Event feeds and temporal features

- So you can build streams and triggers



FAUNA

FaunaDB: The First Serverless Database

What makes Fauna different!

Object level security

- Model your business rules in the database.

Escape the provisioning trap

- No need to fear traffic spikes, or pay in advance for speculatively high throughput.

Hierarchal multi-tenancy

- Makes creating new logical datasets cheap and easy. Serverless processes can scale your business without operator intervention.

Stateless client

- Your Lambdas aren't paying setup and teardown costs for nothing.

FAUNA

FaunaDB: The First Serverless Database

What makes Fauna different!

Object level security

- Model your business rules in the database.

Escape the provisioning trap

- No need to fear traffic spikes, or pay in advance for speculatively high throughput.

Hierarchal multi-tenancy

- Makes creating new logical datasets cheap and easy. Serverless processes can scale your business without operator intervention.

Stateless client

- Your Lambdas aren't paying setup and teardown costs for nothing.

FAUNA

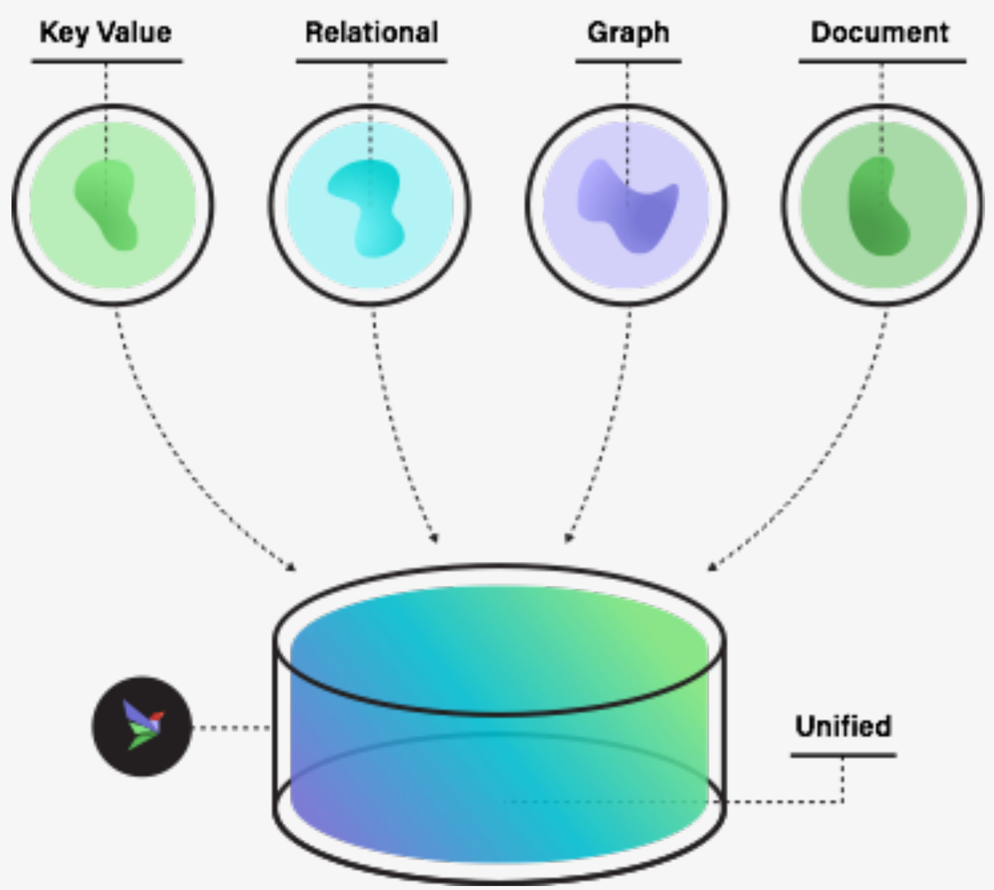
FaunaDB - General Purpose Database

Key Features

- Transactions with global consistency
- Rich query support
- Serverless ease-of-use or on-premise
- Hierarchal Multi-tenancy

Use Cases

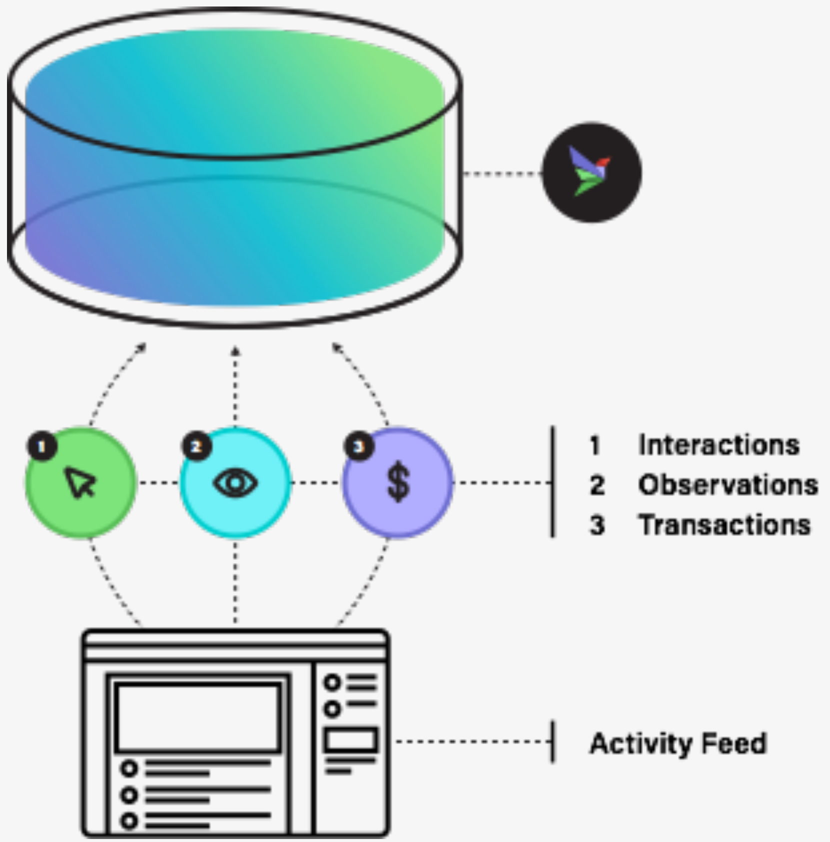
- Distributed Ledger
- Social Graph Content
- Single Page Applications



FAUNA

A simplified developer experience

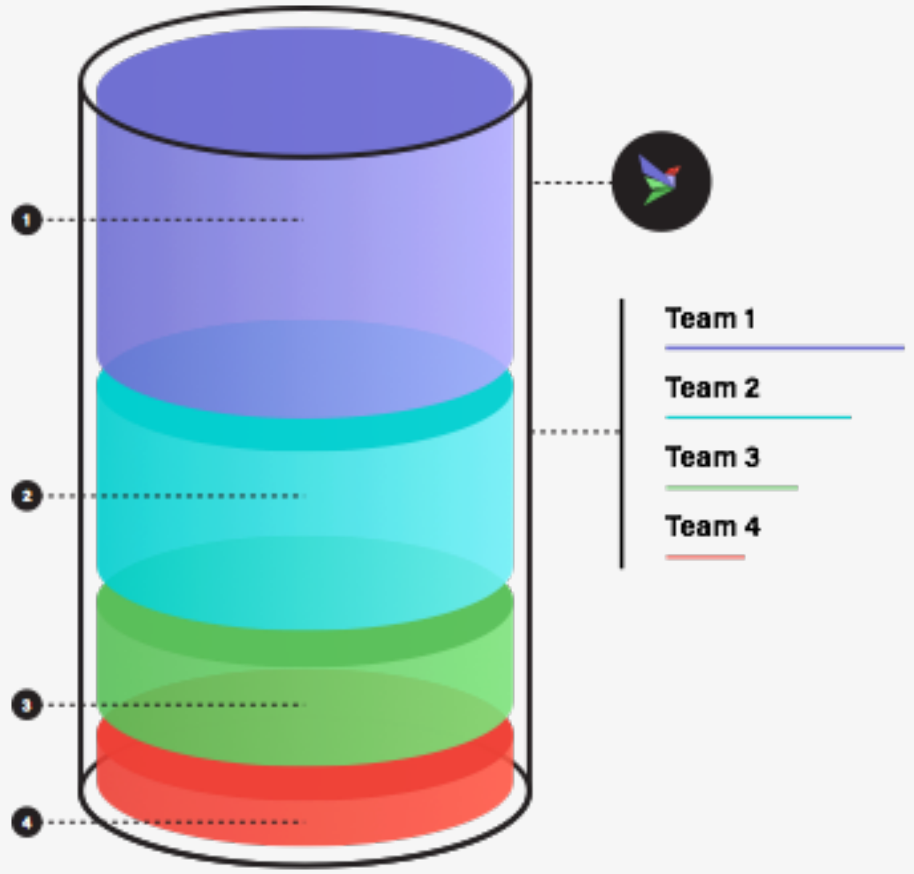
- Expression-oriented, flexible, safe
- Simplify multi-part queries into simple questions:
  - In the rental car fleet, which make and model built between 2013 and 2015 has parts from manufacturers X and Y?
- Increase developer productivity
  - Isolate from complexity of different data models
  - Prevent context switches when moving among query languages and data sets
- Extensible: support data domains such as geographic indexing, full-text search, iterative machine learning.



Communal resources allotted across teams

- Safe sharing across multiple teams, projects or companies in a single FaunaDB cluster
- Dynamically tune resource allocation across tenants
- Align resource utilization with business priorities
    - Prioritize customer data over batch analytics
- Amortize infrastructure costs across multiple services

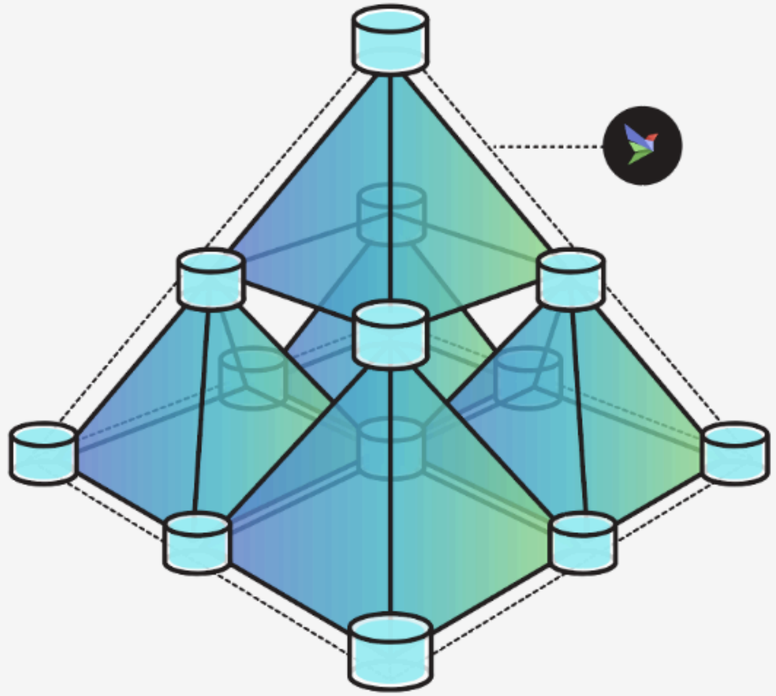
The more diverse applications, datasets, and workloads are hosted in a single FaunaDB cluster, the better the price/performance becomes compared to a traditional, statically provisioned siloed data architecture.



FAUNA

Shared, hierarchical database infrastructure

- Databases within databases
- Shared resources across teams, projects, applications
- Delegated administration
- Security through isolation
- Can reflect organizational structure, physical structure, etc.



A globally shared resource pool

- Native geo-replication
- Physical cluster spans all data centers
- Logical databases assigned by business priority
- No impact on operational overhead
- Increases compute elasticity
- Enables:
    - Low-latency real-time data
    - Geographical data compliance (safe harbor)