Deconstructing the Database

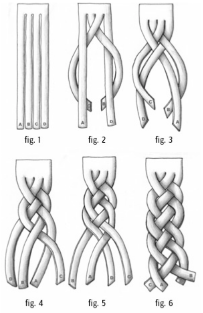
Rich Hickey

What is Datomic?

- A new database
- A sound model of information, with time
- Provides database as a value to applications
- Bring declarative programming to applications
- Focus on reducing complexity

DB Complexity

- Stateful
- Same query, different results
  - no basis
- Over there
- 'Update' poorly defined
  - Places



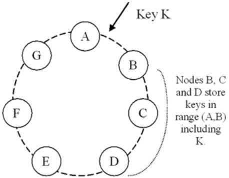
Update

- What does update mean?
- Does the new replace the old?
- Granularity? new ___ replace the old ___
- Visibility?

Manifestations

- Wrong programs
- Scaling problems
- Round-trip fears
- Fear of overloading server
- Coupling, e.g. questions with reporting

Consistency and Scale



- What's possible?
- Distributed redundancy and consistency?
- Elasticity
- Inconsistency huge source of complexity

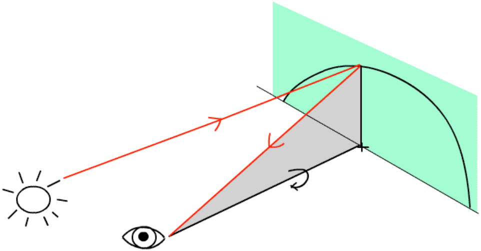
Information and Time

- Old-school memory and records
- The kind you remember

  ... and keep
- Auditing and more

Perception and Reaction

- No polling
- Consistent



Coming to Terms

Value

- An *immutable* magnitude, quantity, number... or immutable composite thereof

Identity

- A putative entity we associate with a series of causally related values (states) over time

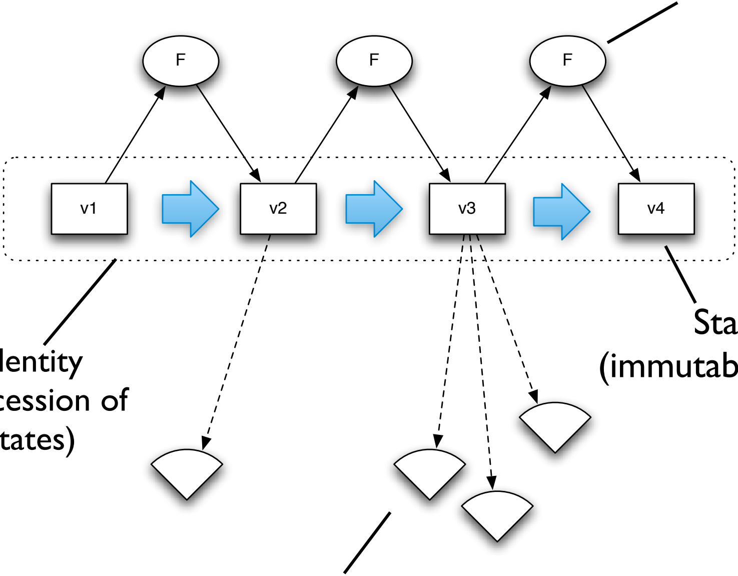
State

- Value of an identity at a moment in time

Time

- Relative before/after ordering of causal values

Epochal Time Model



Process events
(pure functions)

Identity
(succession of
states)

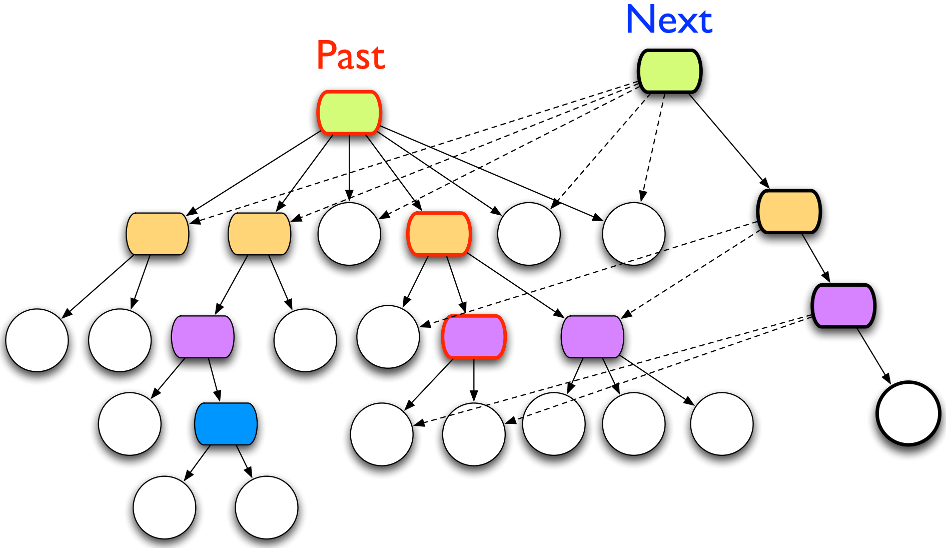
States
(immutable values)

Observers/perception/memory

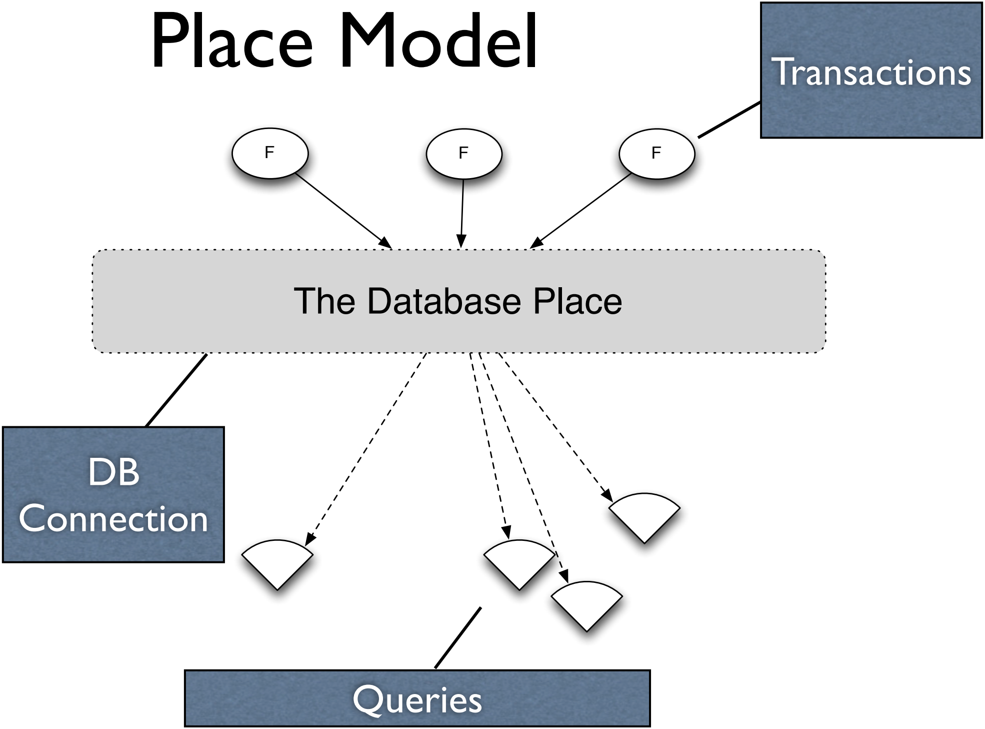
Implementing Values

- Persistent data structures
- Trees
- Structural sharing

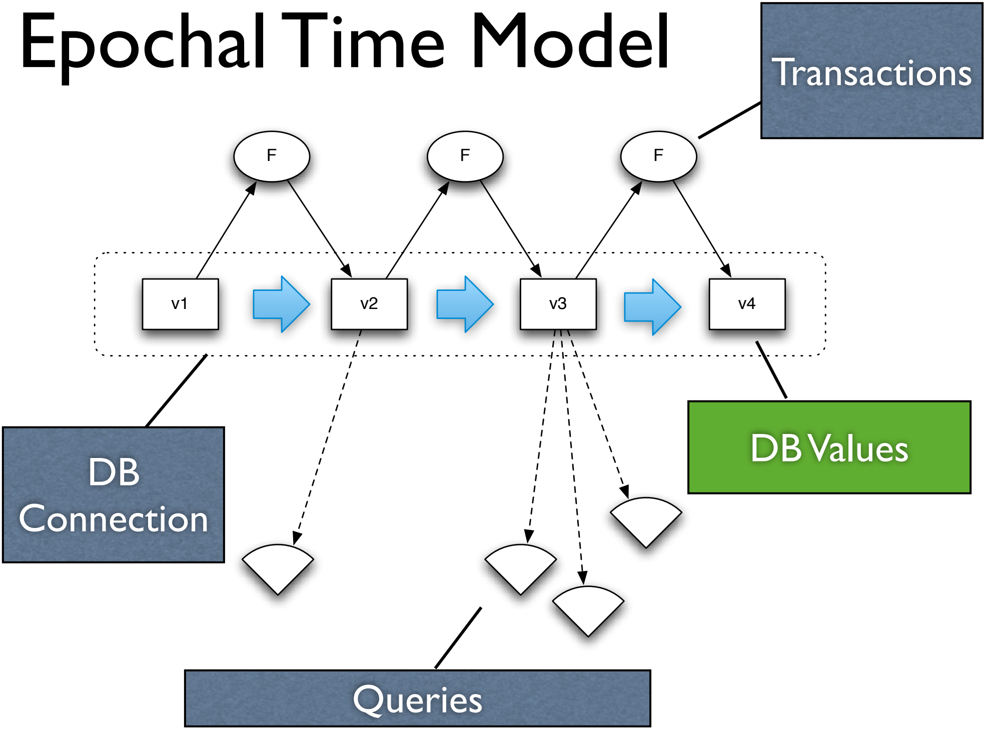
Structural Sharing



Place Model


Transactions
F
F
F
The Database Place
DB Connection
Queries

Epochal Time Model



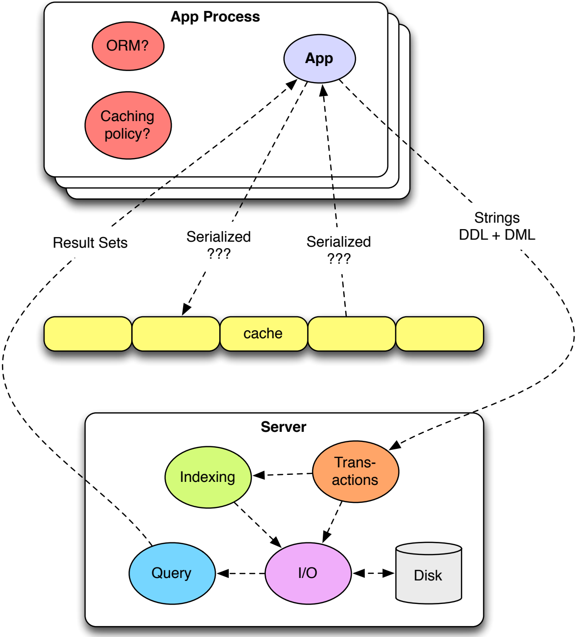
Transactions

DB Connection

DB Values

Queries

Traditional Database



The Choices

- Coordination
  - how much, and where?
  - process requires it
  - perception shouldn’t
- Immutability
  - sine qua non

Approach

- Move to information model
- Split process and perception
- Immutable basis in storage
- Novelty in memory

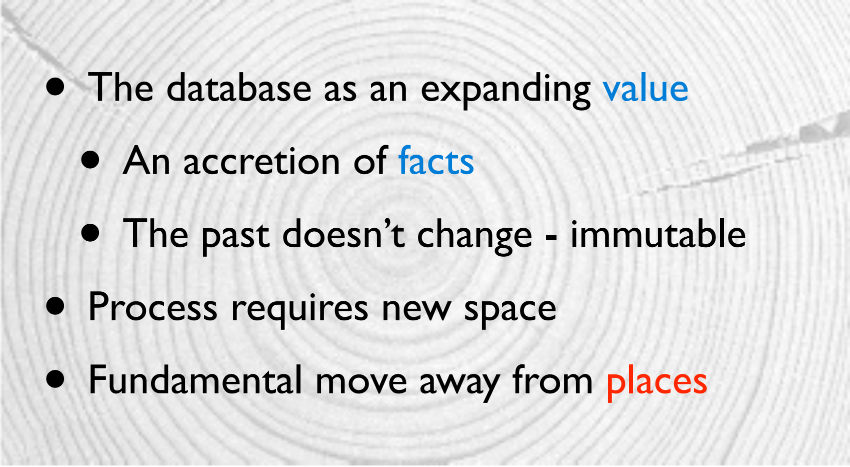
Information

- Inform
  - ‘to convey knowledge via facts’
  - ‘give shape to (the mind)’
- Information
  - the facts

Facts

- Fact - 'an event or thing known to have happened or existed'
  - From: factum - 'something done'
  - Must include time
- Remove structure (a la RDF)
- Atomic Datom
  - Entity/Attribute/Value/Transaction(time)

Database State



- The database as an expanding value
  - An accretion of facts
  - The past doesn't change - immutable
- Process requires new space
- Fundamental move away from places

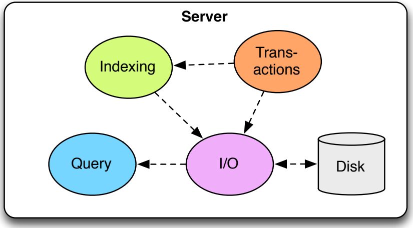
Accretion

- Root per transaction doesn’t work
- Latest values include past as well
  - The past is sub-range
- Important for information model

Process

- Reified
- Primitive representation of novelty
  - Assertions and retractions of facts
  - Minimal
- Other transformations expand into those

Deconstruction



- Process
  - Transactions
  - Indexing
  - O

- Perception/Reaction
  - Query
  - Indexes
  - I

State

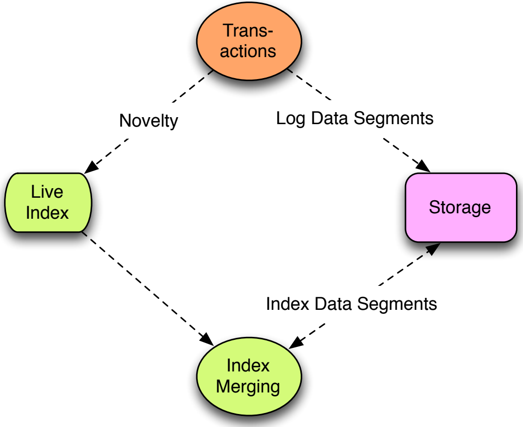
- Must be organized to support query
- Sorted set of facts
- Maintaining sort live in storage - bad
    - BigTable - mem + storage merge
    - occasional merge into storage
    - persistent trees

Indexing

- Maintaining sort live in storage - bad
- BigTable et al:
  - Accumulate novelty in memory
  - Current view: mem + storage merge
  - Occasional integrate mem into storage

    Releases memory

Transactions and Indexing



Trans-actions

Novelty

Log Data Segments

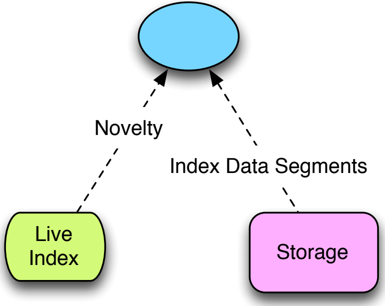
Live Index

Storage

Index Data Segments

Index Merging

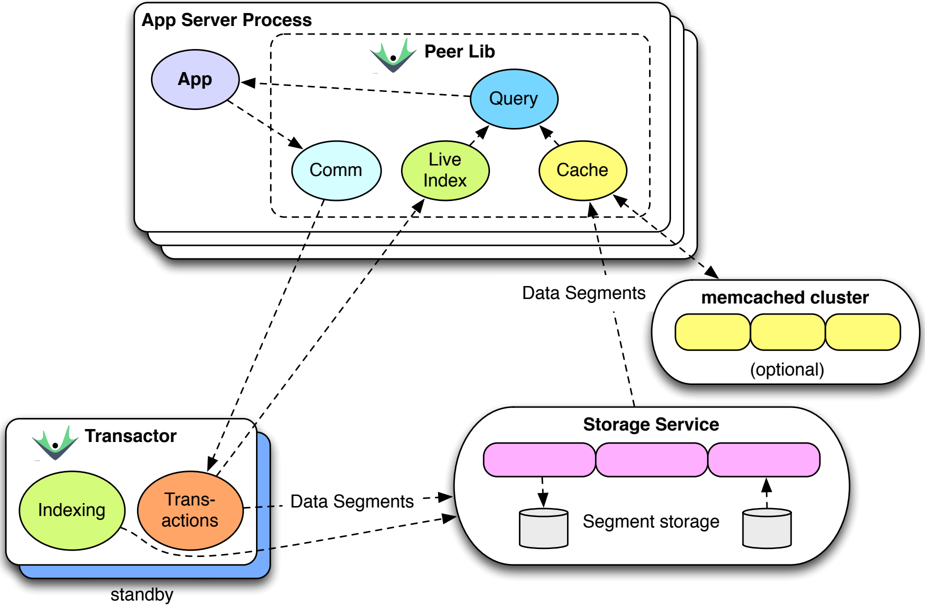
Perception



Components

- Transactor
- Peers
  - Your app servers, analytics machines etc
- Redundant storage service

Datomic Architecture



Transactor

- Accepts transactions
  - Expands, applies, logs, broadcasts
- Periodic indexing, in background
- Indexing creates garbage
  - Storage GC

Peer Servers

- Peers directly access storage service
- Have own query engine
- Have live mem index and merging
- Two-tier cache
    - Datoms w/object values (on heap)
    - Segments (memcached)

Consistency and Scale

- Process/writes go through transactor
  - traditional server scaling/availability
- Immutability supports consistent reads
  - without transactions
- Query scales with peers
  - Elastic/dynamic e.g. auto-scaling

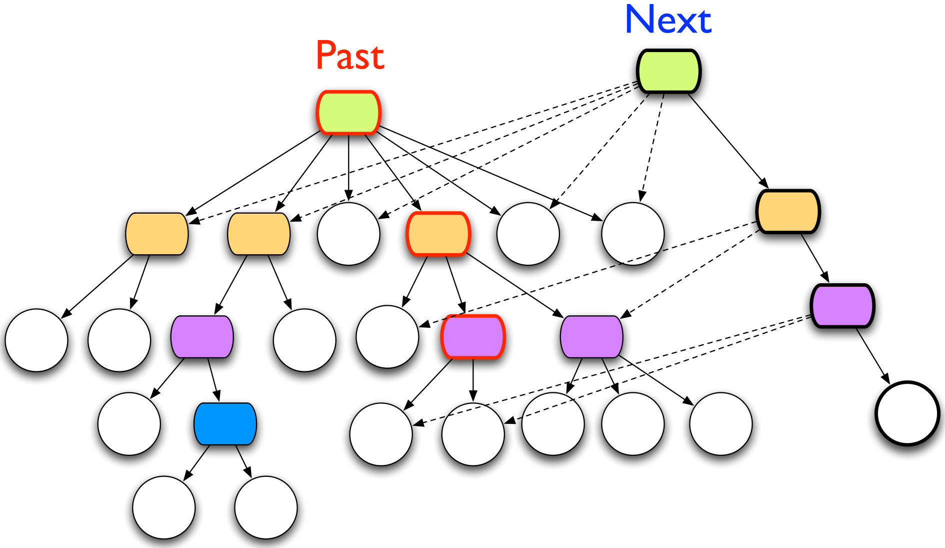
Memory Index

- Persistent sorted set
- Large internal nodes
- Pluggable comparators
- 2 sorts always maintained
  - EAVT, AEVT
- plus AVET, VAET

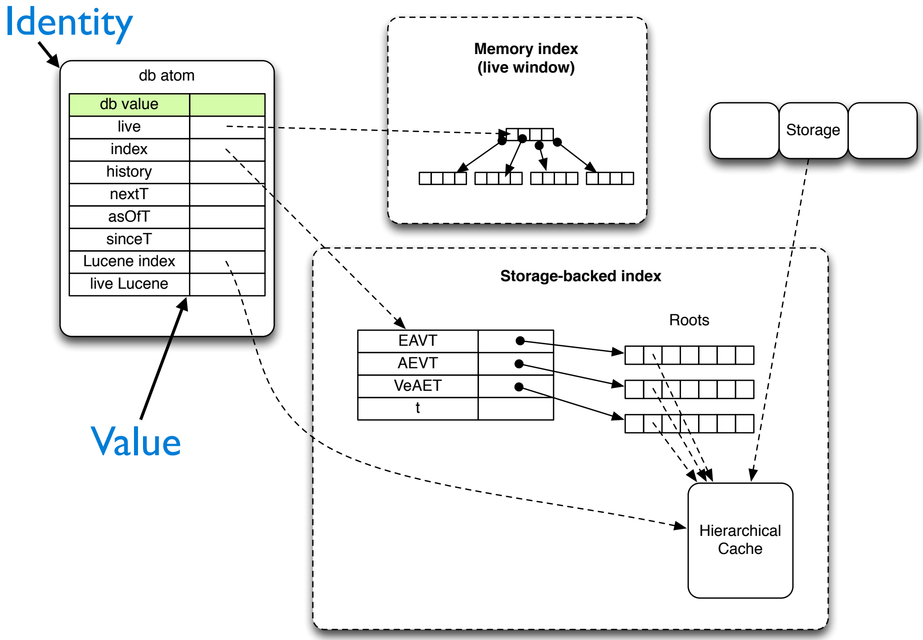
Storage

- Log of tx asserts/retracts (in tree)
- Various covering indexes (trees)
- Storage service/server requirements
  - Data segment values (K->V)
  - atoms (consistent read)
  - pods (conditional put)

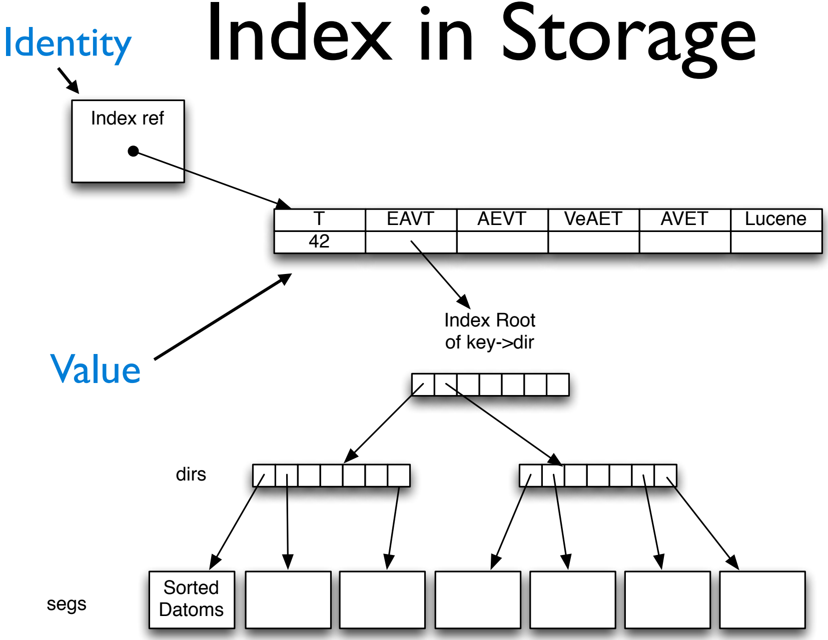
Structural Sharing



What's in a DB Value?



Index in Storage



T	EAVT	AEVT	VeAET	AVET	Lucene
42					

Datomic on Riak

+ ZooKeeper

- Riak

  redundant, distributed, highly available

  durable

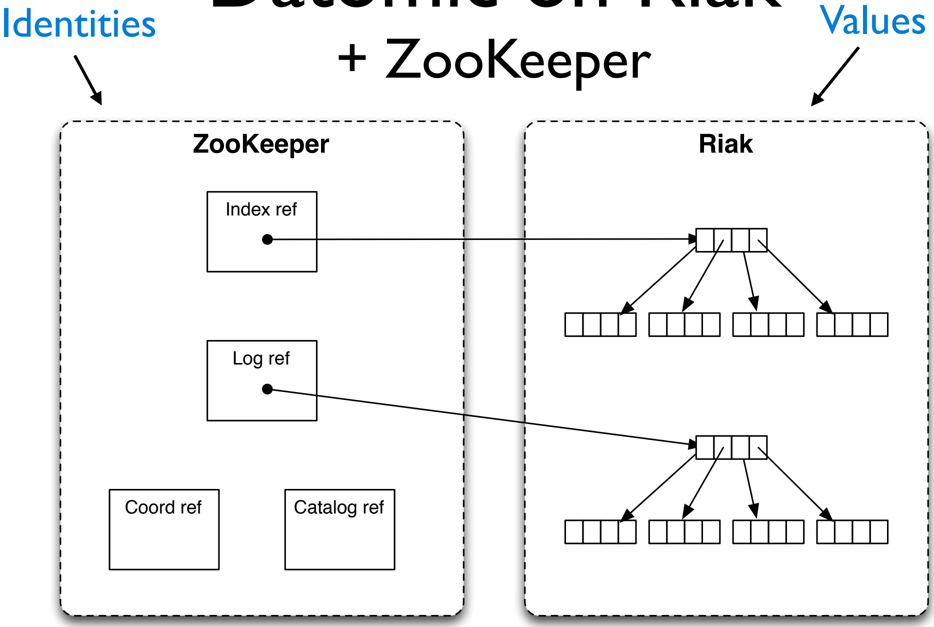
  eventually consistent

- ZooKeeper

  redundant, durable,

  consistent (ordered ops + CAS)

Datomic on Riak
+ ZooKeeper



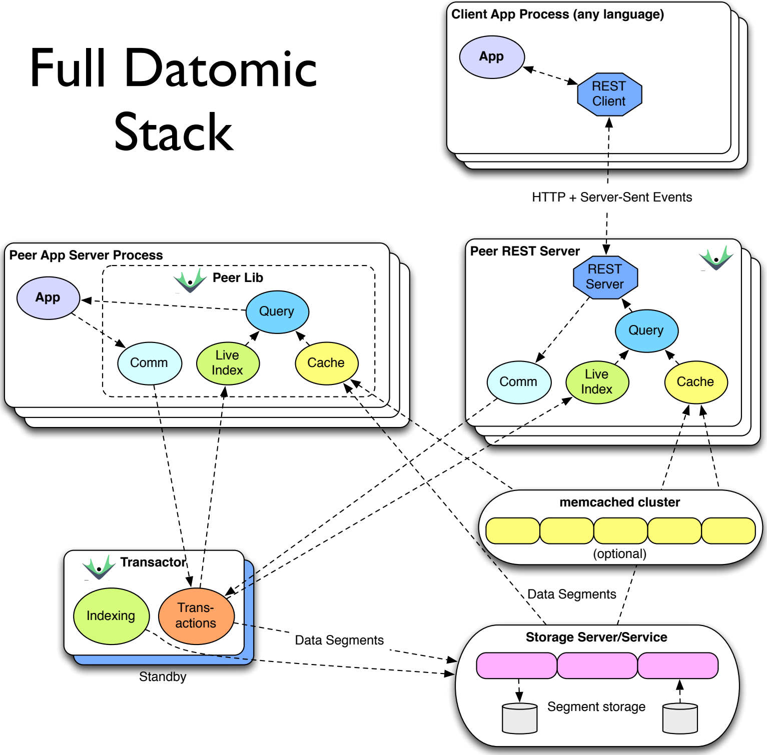
Riak Usage

- Everything put into Riak is immutable
- N=3, W=2, DW=2
- R=1, not-found-ok = false

  'first found' semantics
- There or not

  no vector clocks, siblings etc
- No speculative lookup

Full Datomic Stack



Stable Bases

```
//Peer
Database db = connection.db().asOf(1000);
Peer.q(aQuery, db);

//Client
GET /data/mem/test/1000/datoms?index=aevt
```

basis

- Same query, same results
- db permalinks!
  - communicable, recoverable
- Multiple conversations about same value

DB Values

- Time travel
  - `db.asOf` - past
  - `db.since` - windowed
  - `db.with(tx)` - speculative
- dbs are arguments to query, not implicit
  - mock with datom-shaped data:

```
[[:fred :likes "Pizza"]
 [:sally :likes "Ice cream"]]
```

DB Simplicity Benefits

- Epochal state
  - Coordination only for process
- Transactions well defined
  - Functional accretion
- Freedom to relocate/scale storage, query
- Extensive caching
- Process events

The Database as a Value

- Dramatically less complex
- More powerful
- More scalable
- Better information model

Thanks for Listening!