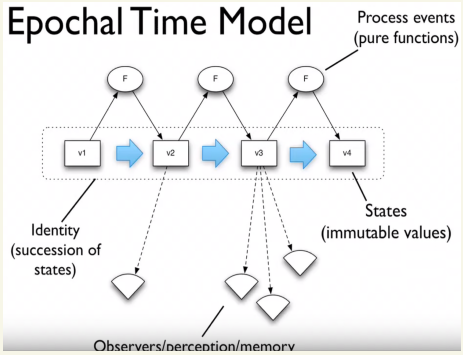
Datomic with Rich Hickey

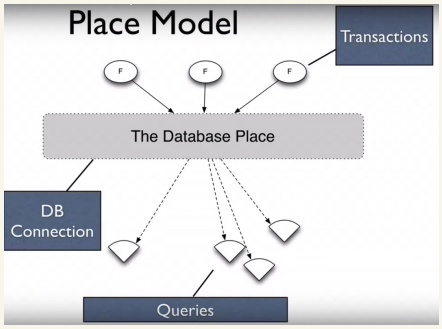
Definitions

Value: an immutable magnitude ... 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.

In datomic, an entire database can be a value to the application.



Epochal time can be achieved "in-memory" using: persistent data structures, trees and structural sharing.



The traditional way of programming against a DB is to program against a DB connection

Transactions
- are a property of the connection.
- in relational DBs have atleast a basis of an unknown but consistent state.
- in noSQL stores have no basis at all.

Databases conflate identity and value; and collapses time.

How should we think about Database State? as an expanding value; an accretion of immutable facts.

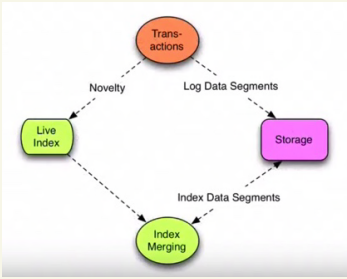
Accretion:
- Root per transaction (as in in-memory structural sharing) does not work for DBs.
  - crossing processes and time
  - a DB will have to maintain every root
  - cannot do global GC.
- Instead, the latest value of the database is the whole database; latest value contains the past values as well.

Facts:
- factum: something which happened.
- always has a time dimension.
- atomic
- in Datomic, represented by Datom. E/A/V/Transaction

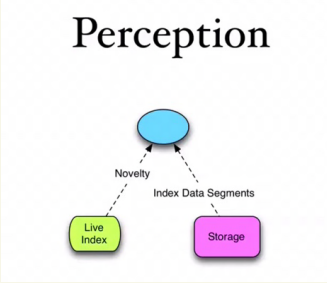
Process:
- assertions and retractions of facts
- similar to CQRS event-logs
- it is a primitive representation of novelty

State Implementation
- SSTables, like BigTable
- Sorted set of facts
- Compaction of SSTables into disk creates persistent tree structures.

Transactions and Indexing

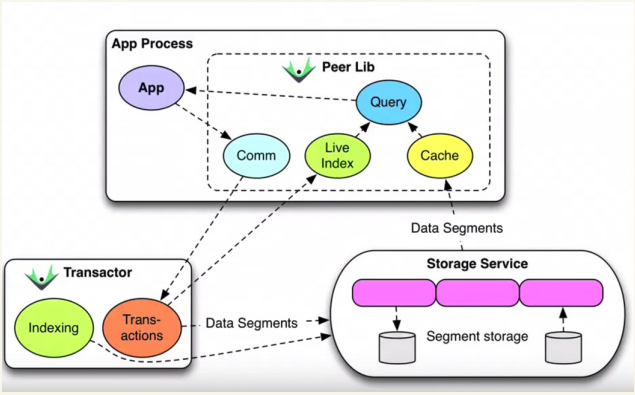


Process



Perception

Architecture



- the core of the architecture lives in the application server.
- storage is commoditized – can be a relational database, DynamoDB, etc. – which makes storage independently scalable.
- reads are horizontally scalable – just add more application servers.
- Transaction co-ordination is the sole concern of the Datomic server. Hence, called Transactors.
- Transactors do not have query load; only write loads.
- Indexing, although shown as part of the Transactor box, can live on a separate machine.
- Transactors reflect back transactions to Peers, so all Peers are maintaining the same live index.
- Transactors performs periodic indexing in the background.
- Indexing creates garbage. Transactor is responsible for Storage GC.

Memory Index

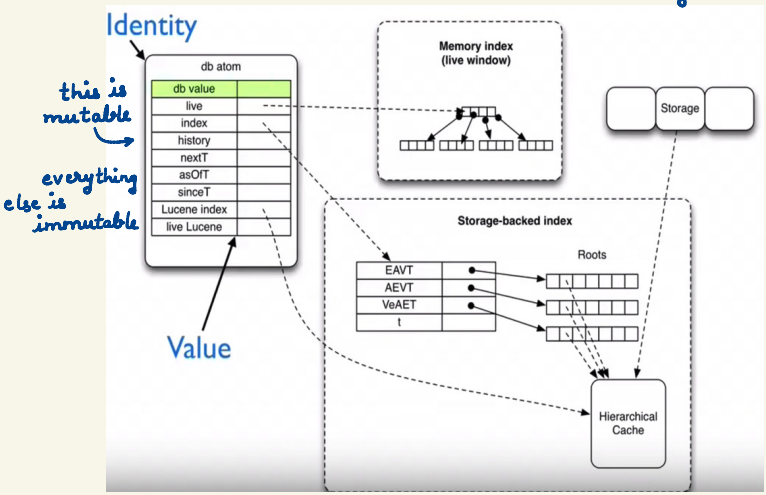
- Persistent sorted set.
- Pluggable comparators.
- EAVT and AEVT sort always maintained.
- AVET and VAET are the other sorts.

Storage

- Two things stored in storage:
  - Log of tx asserts/retracts (in tree).
  - Covering indices (trees)
- Storage System Requirements
  - Consistent Read
  - Conditional Put

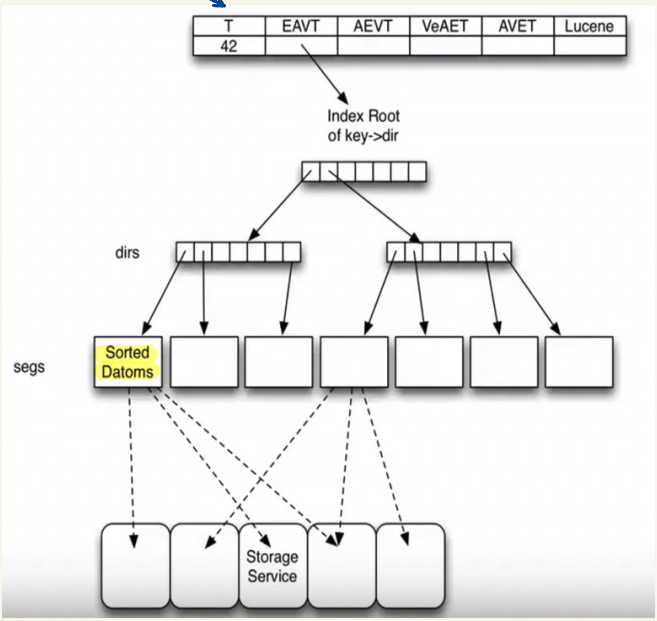
DB Value

This is how the database looks in memory.



Index Storage

set of trees, one for each sort.



Transaction function

$f(db, args\ldots) \longrightarrow tx\text{-}data$

↳ database as a value.

$tx\text{-}data$: assert | retract | tx-fn (args…)

expand and splice until everything is just assert/retracts.

Peers

- directly access storage service.
- own query engine on each peer.
- Two tier cache:
  - Segments (off-heap / memcached)
  - Datoms w/object values (on heap)

Datalog

- subset of Prolog.
- set-oriented, guaranteed termination.
- db + rules + queries
- database is reified; not implicit as in SQL.
- you can call your own code with expression clauses.

example query:

```
connection.q(query, db1, db2, otherInputs ...);

{:find [?customer ?product]
 :where [[?customer :shipAddress ?addr]
         [?addr :zip ?zip]
         [?product :product/weight ?weight]
         [?product :product/price ?price]
         [(Shipping/estimate ?zip ?weight) ?shipCost]
         [(<= ?price ?shipCost)]]}
```

everything is either a list or a map.

all joins are implicit.