copenhagen goto; conference

Writing Datomic in Clojure

Rich Hickey

Datomic, Clojure

Overview

- What is Datomic?
- Architecture
- Implementation - Clojure Applied
- Summary

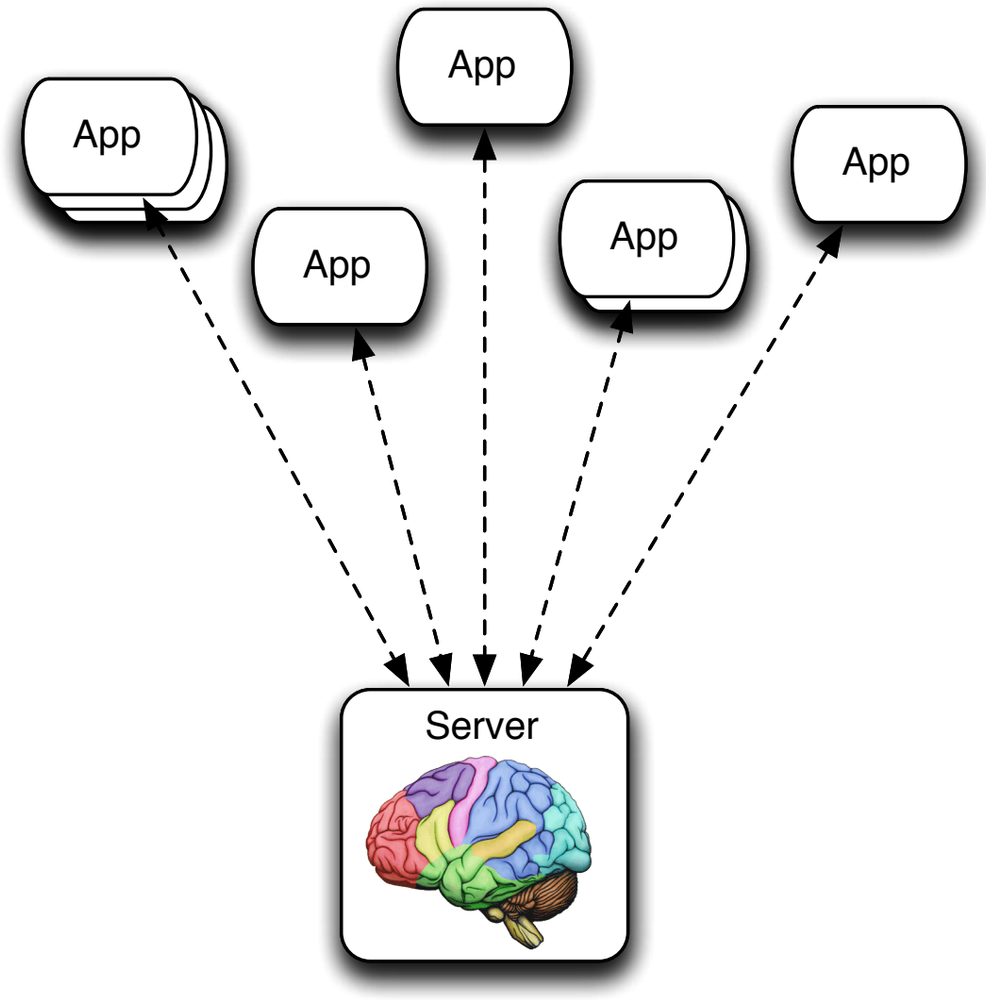
What is Datomic?

- A new kind of database
- Bringing data power *into* the application
- A sound model of information, with time
- Enabled by architectural advances

Why Datomic?

- Architecture
- Data Model

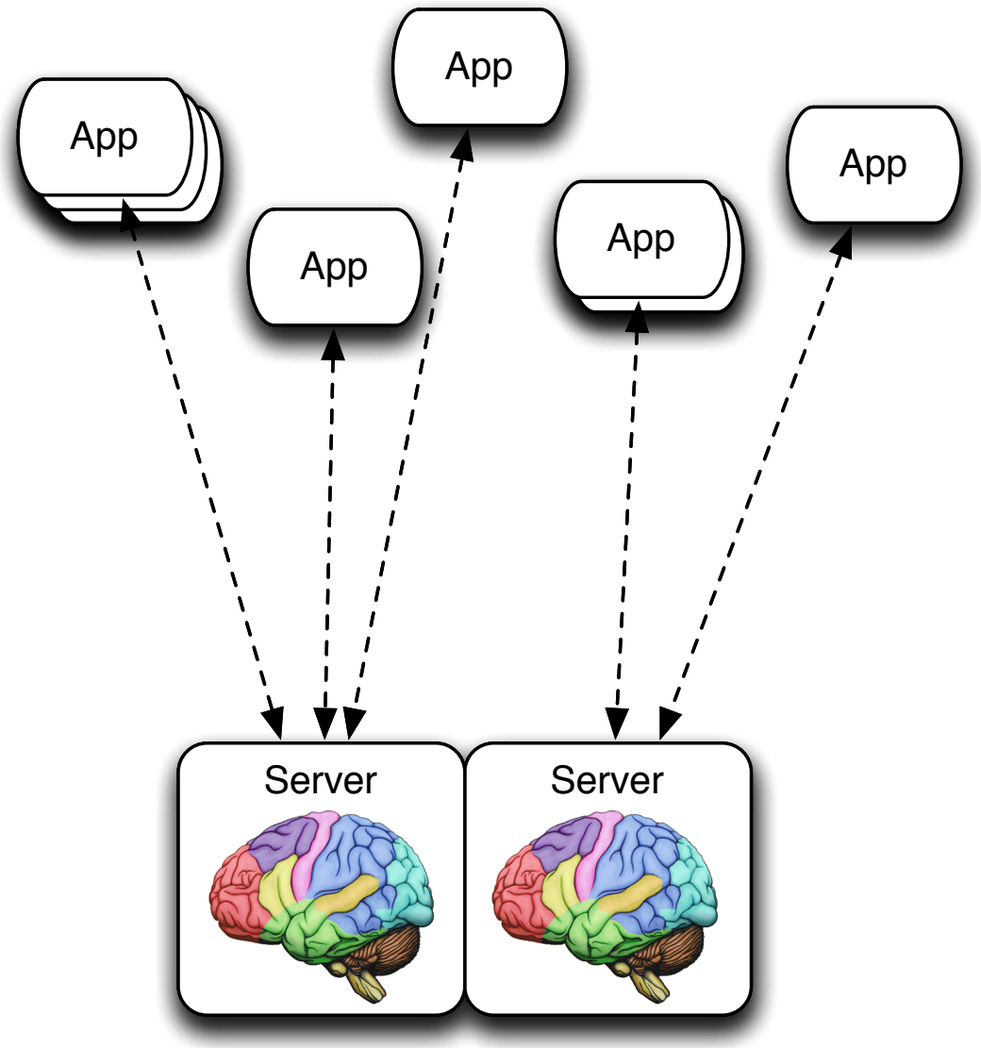
Architectures



Client-Server

Queries
Transactions
Consistency
Storage

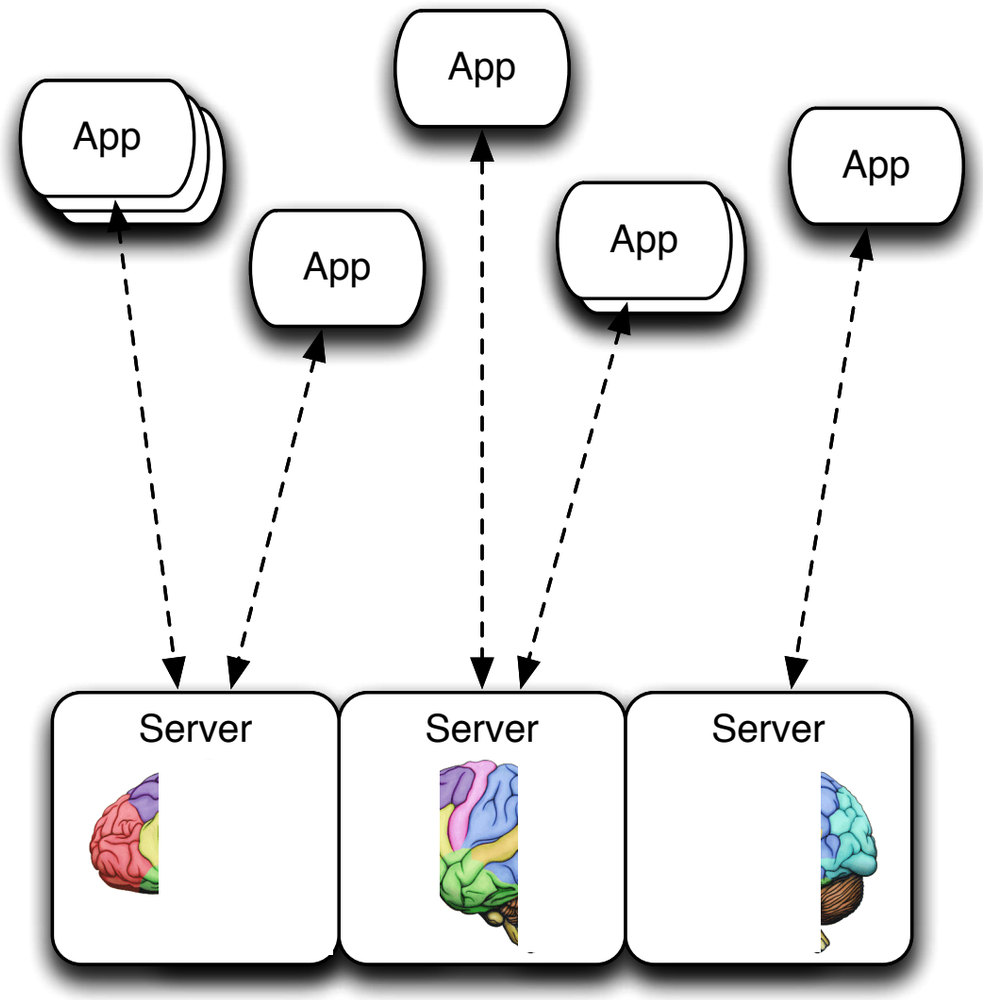
Architectures



Clustered Client-Server

Queries
Transactions
Consistency
Storage

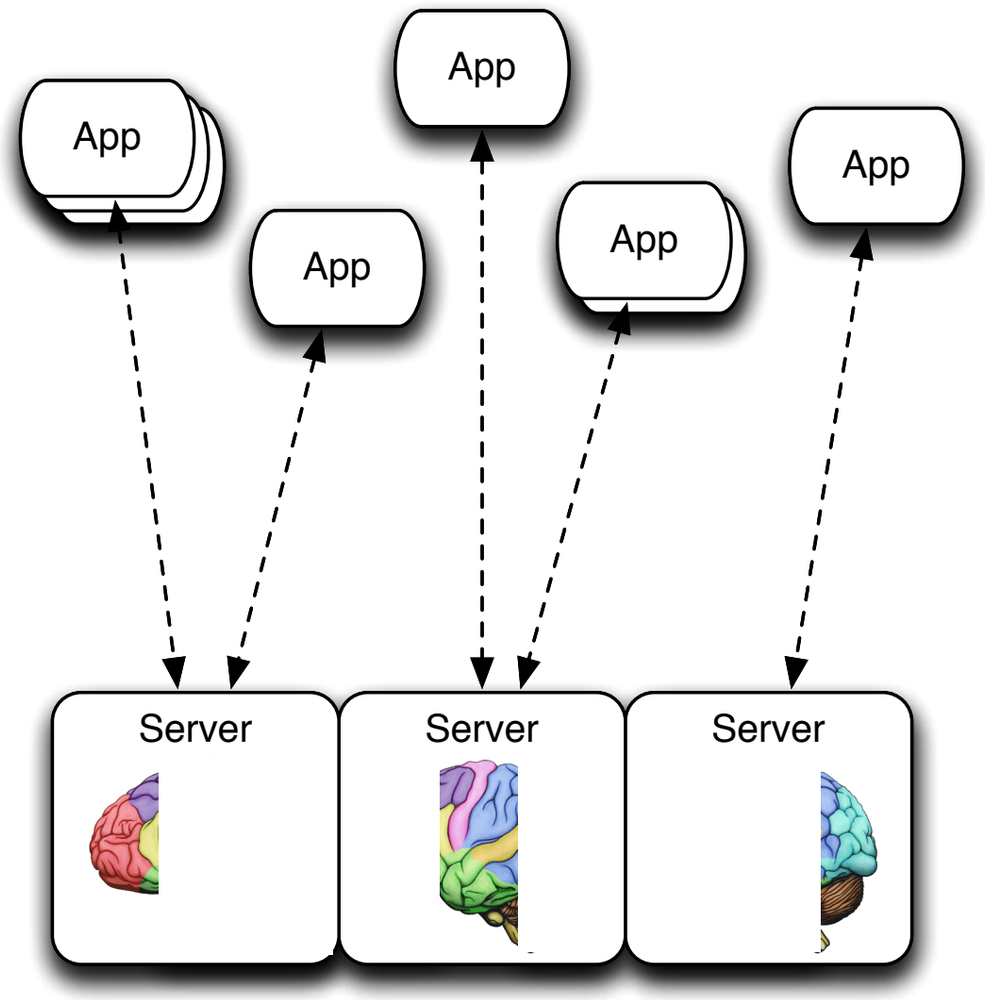
Architectures



Sharded Client-Server

Queries
Transactions
Consistency
Storage

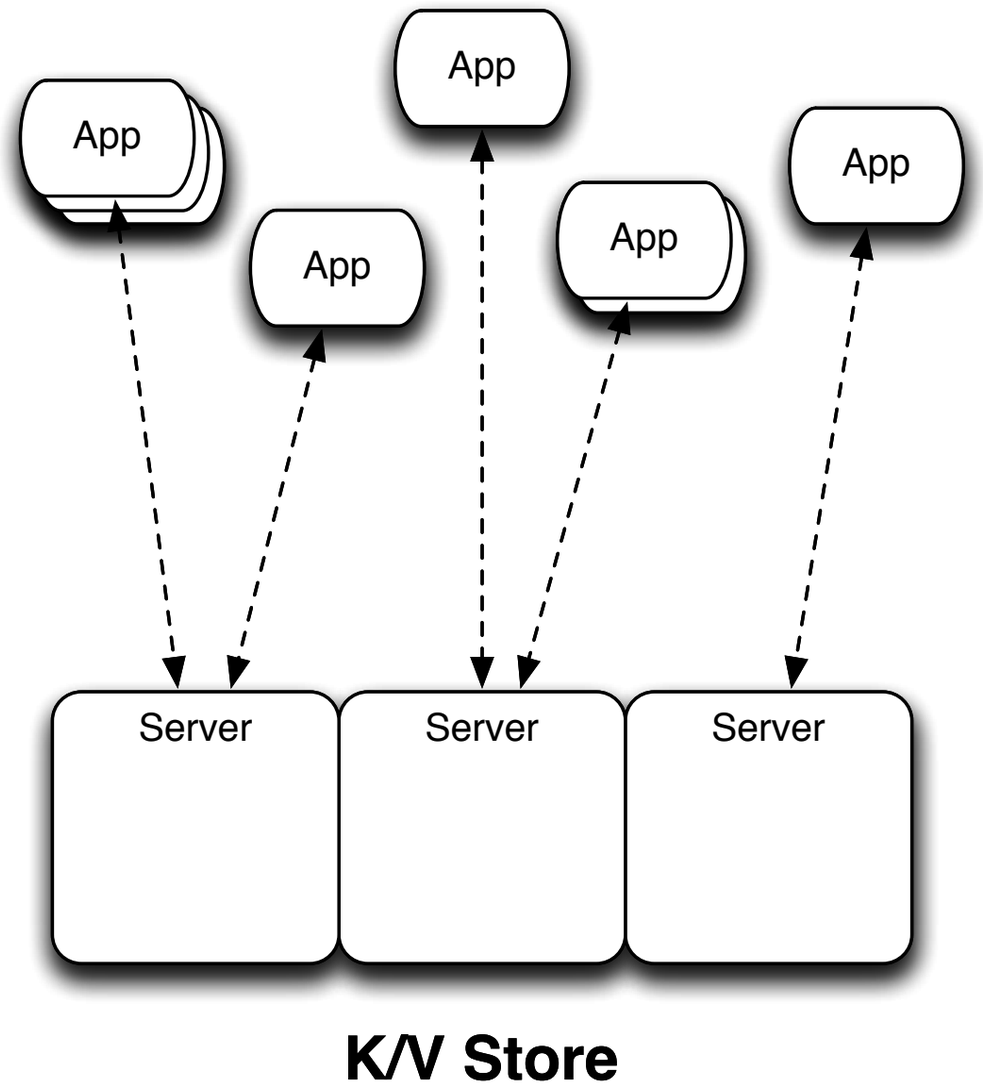
Architectures



Sharded Client-Server

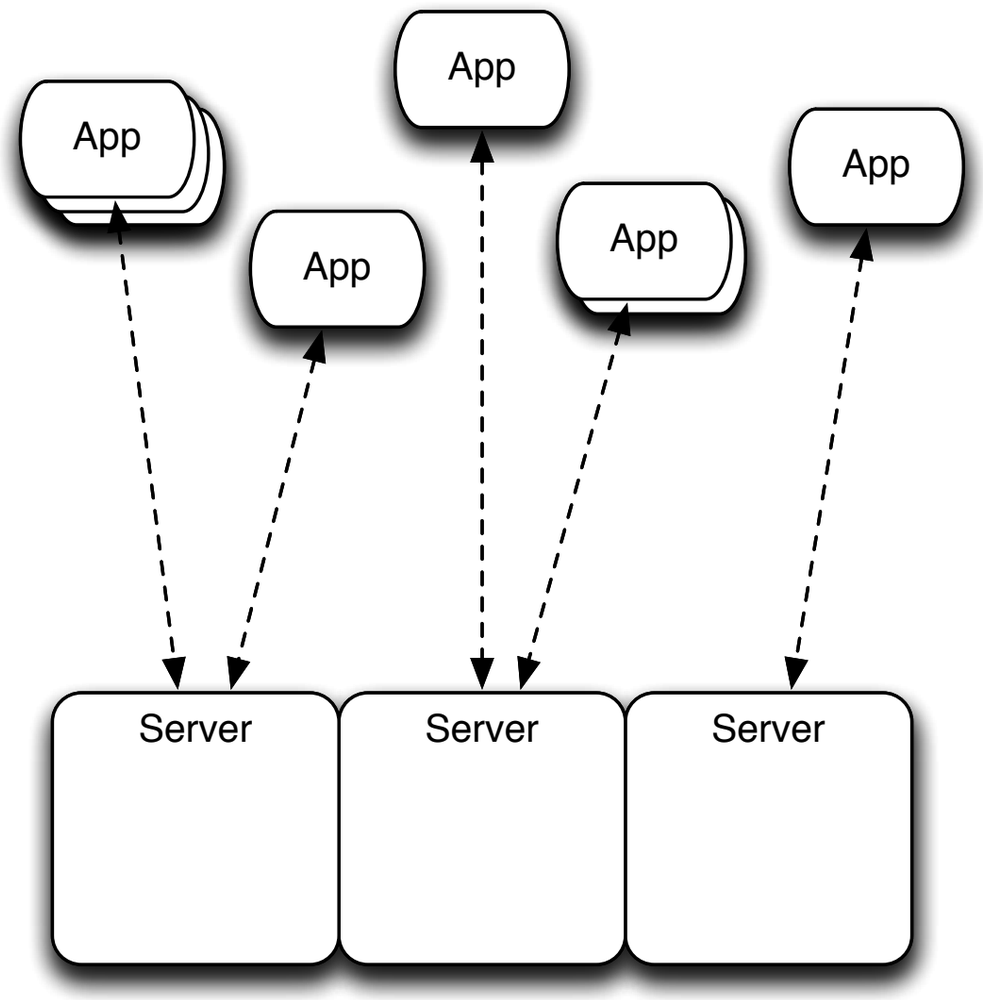
Queries
Transactions
Consistency
Storage

Architectures



Queries
Transactions
Consistency
Storage

Architectures



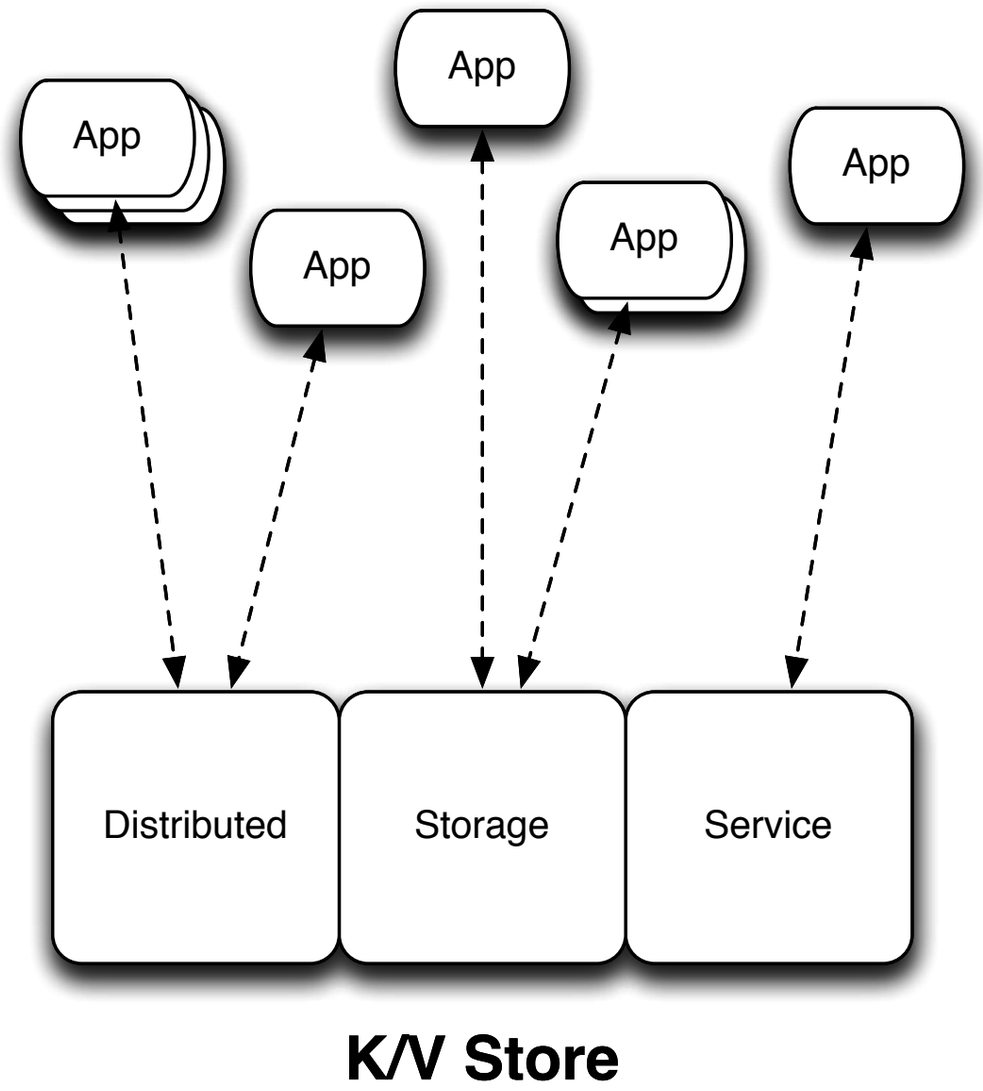
~~Queries~~

~~Transactions~~

~~Consistency~~

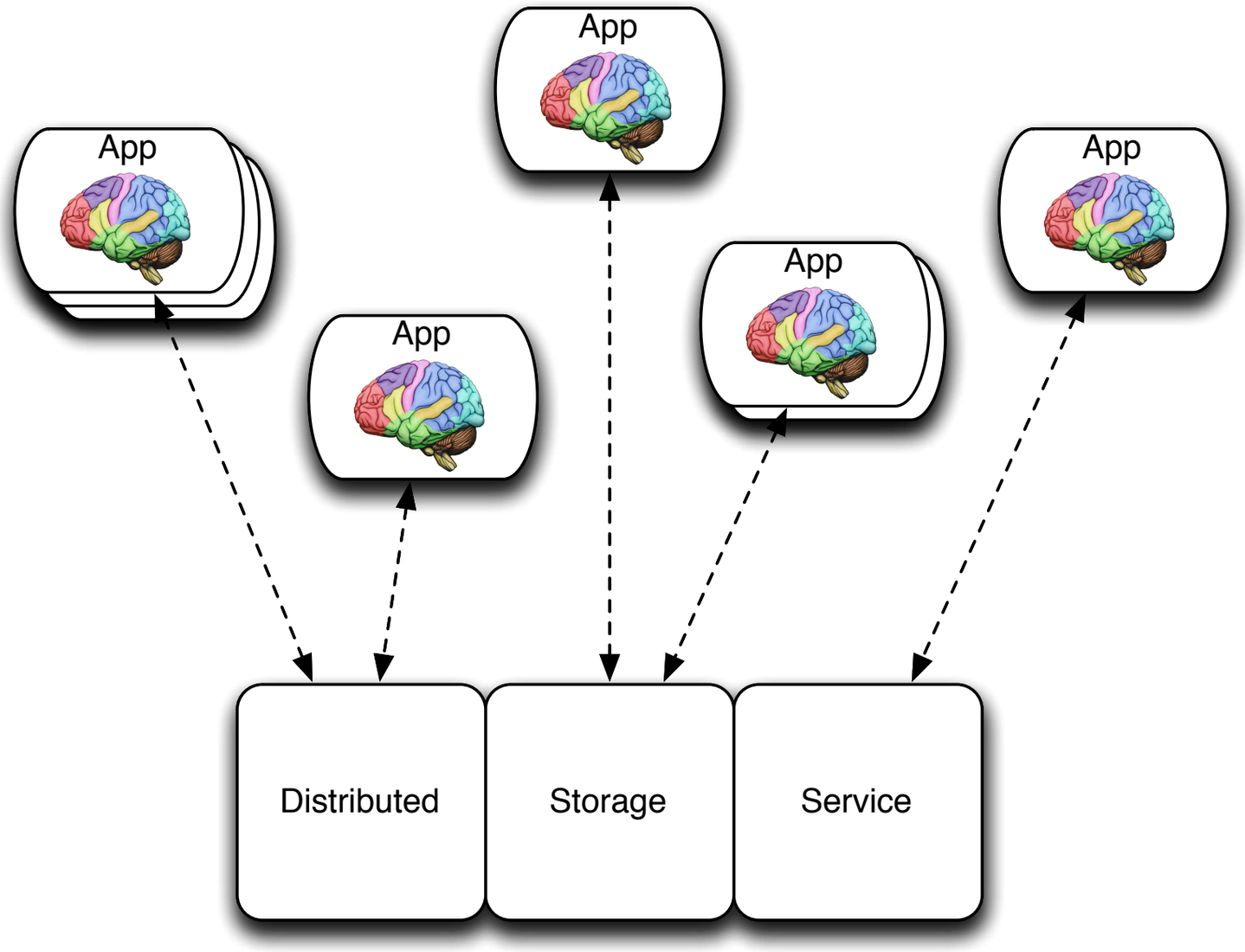
Storage

Architectures

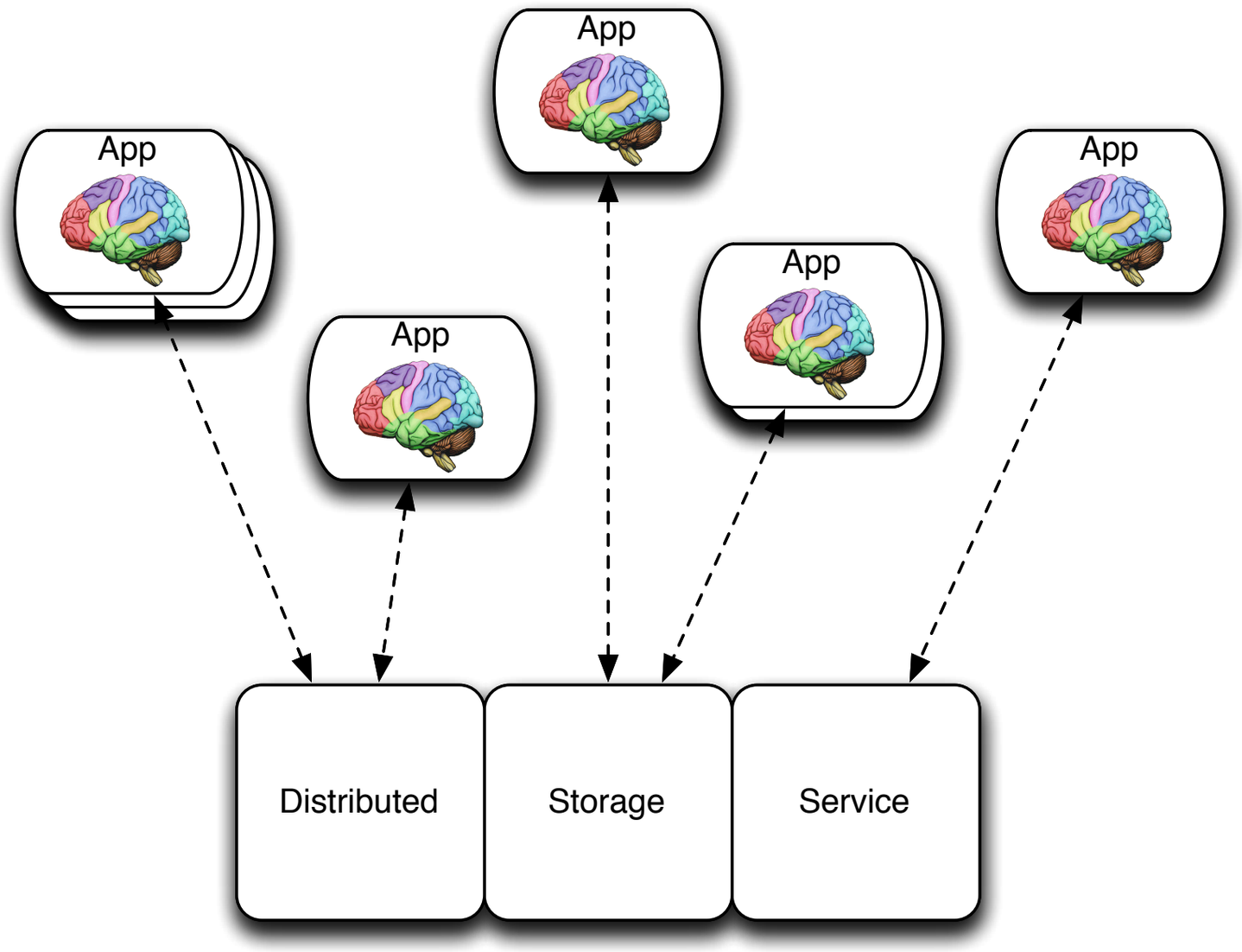


~~Queries~~
~~Transactions~~
~~Consistency~~
Storage

Datomic Architecture



Datomic Architecture



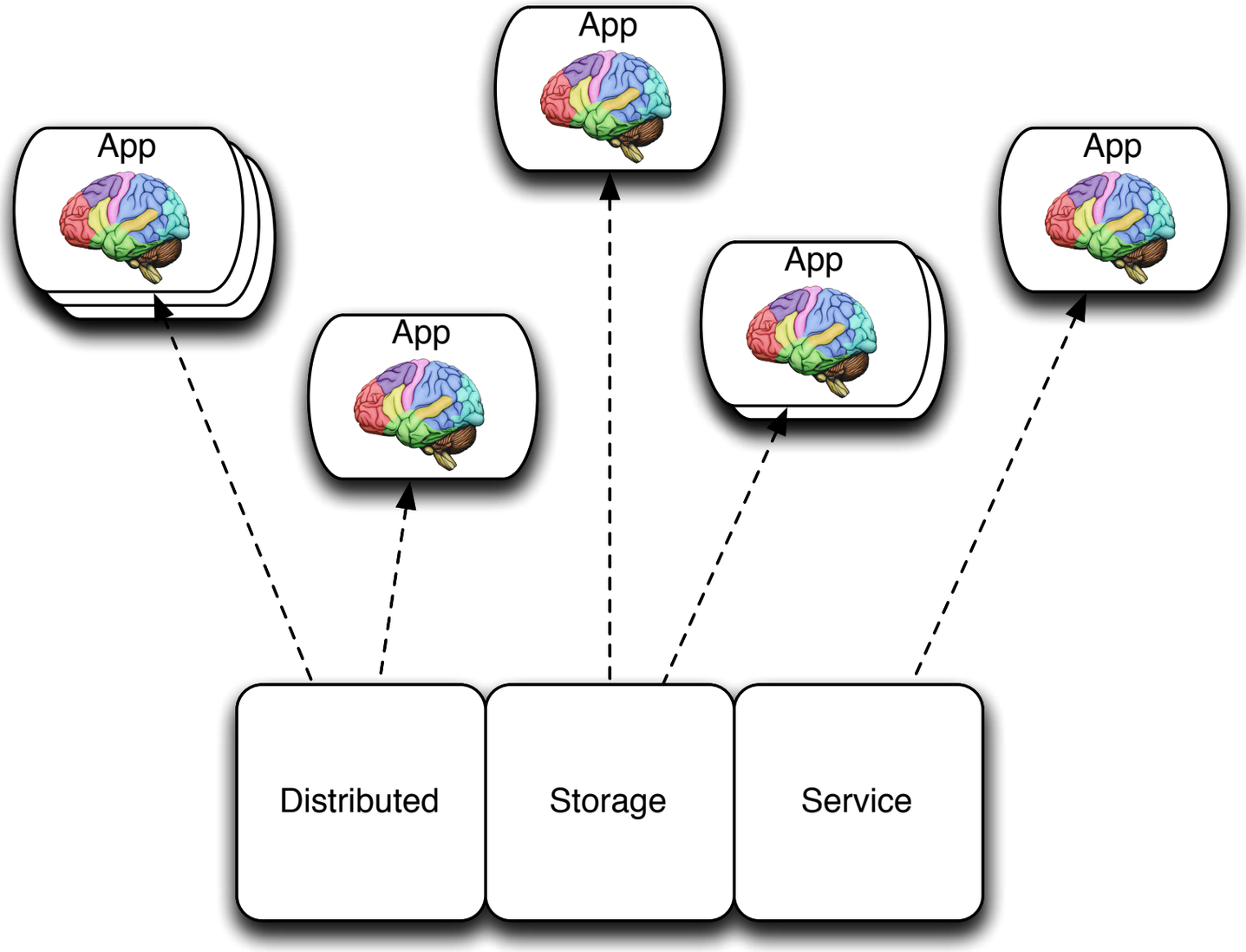
Queries

~~Transactions~~

~~Consistency~~

Storage

Datomic Architecture



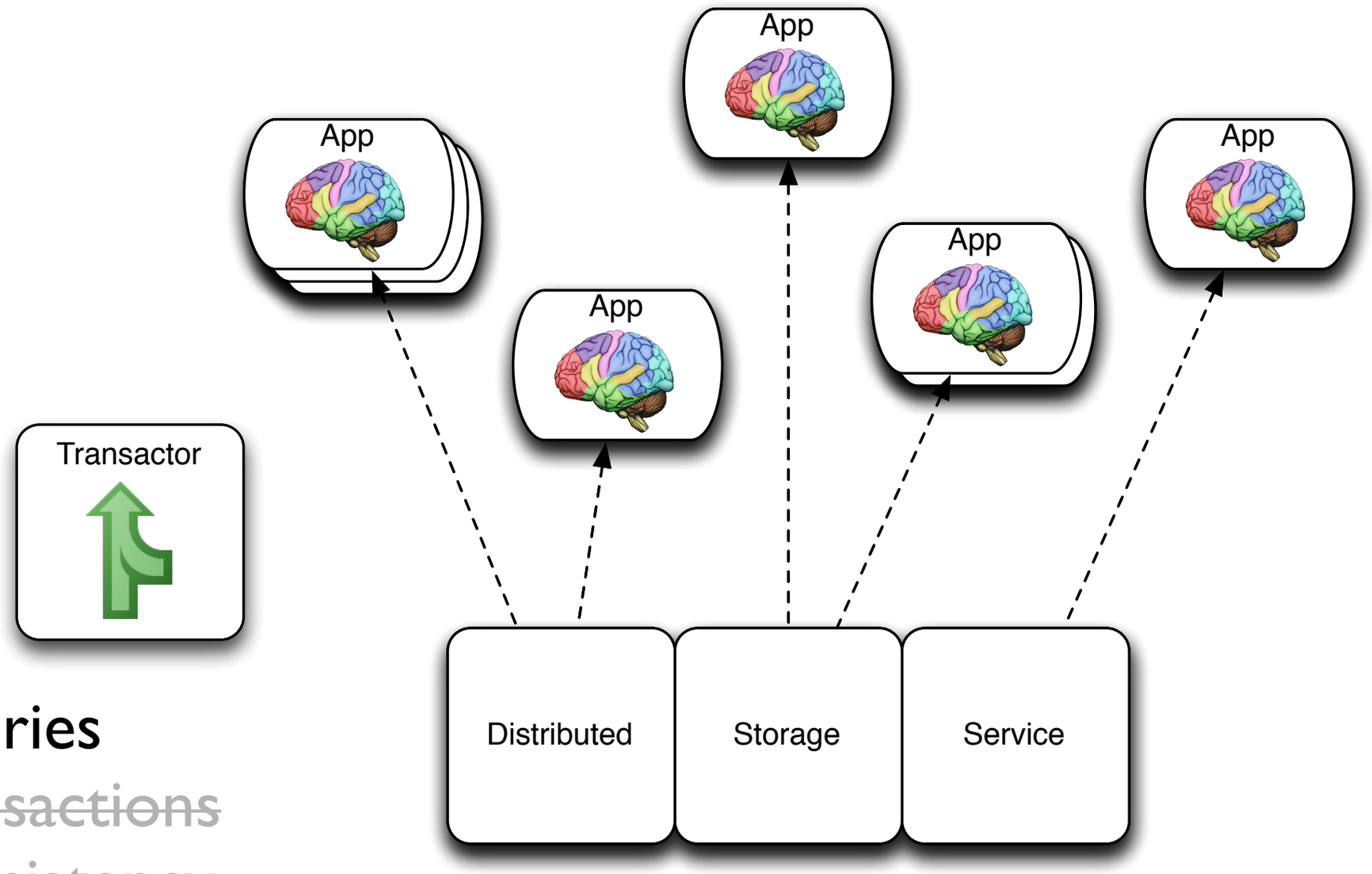
Queries

~~Transactions~~

~~Consistency~~

Storage

Datomic Architecture



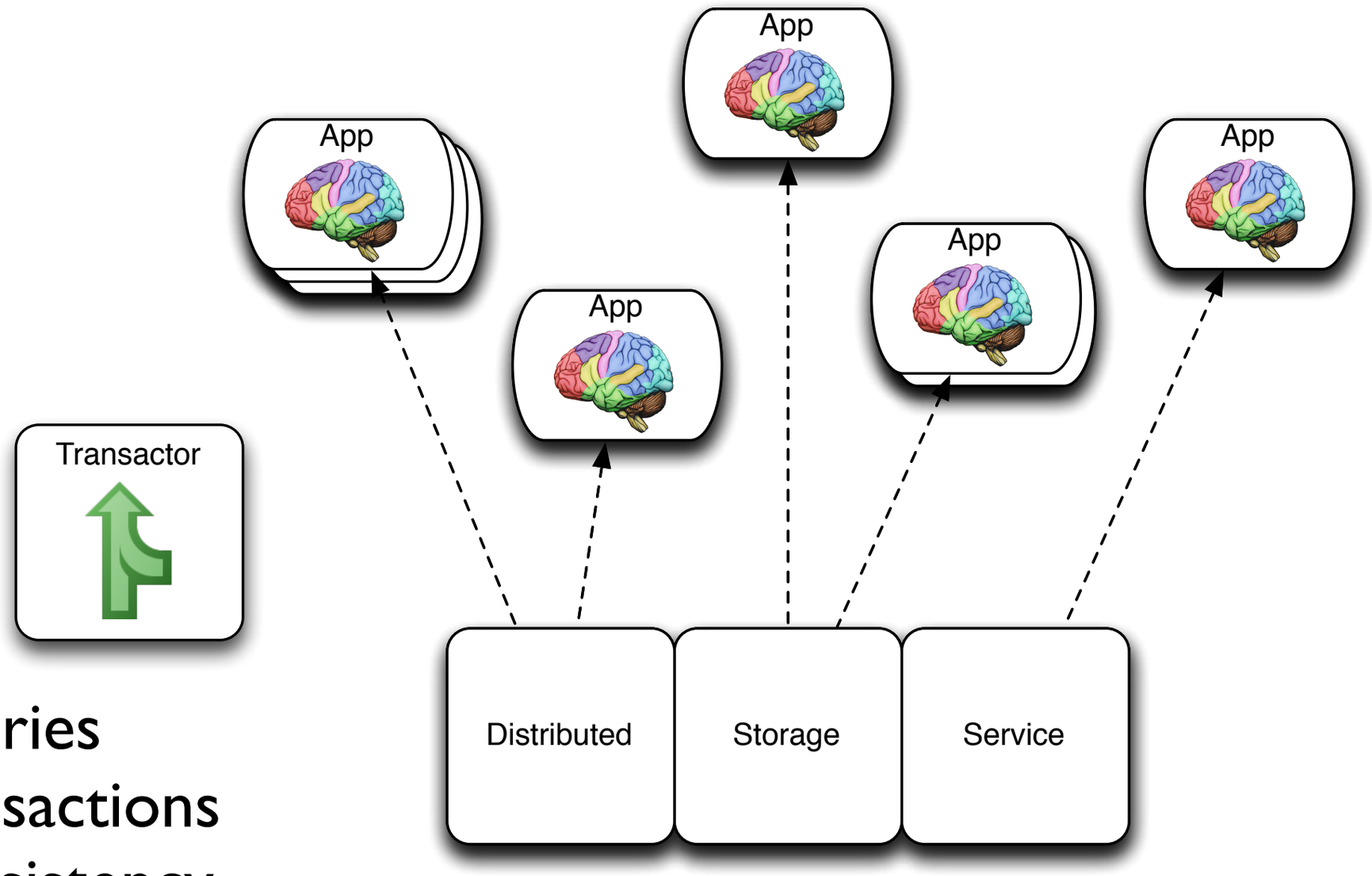
Queries

~~Transactions~~

~~Consistency~~

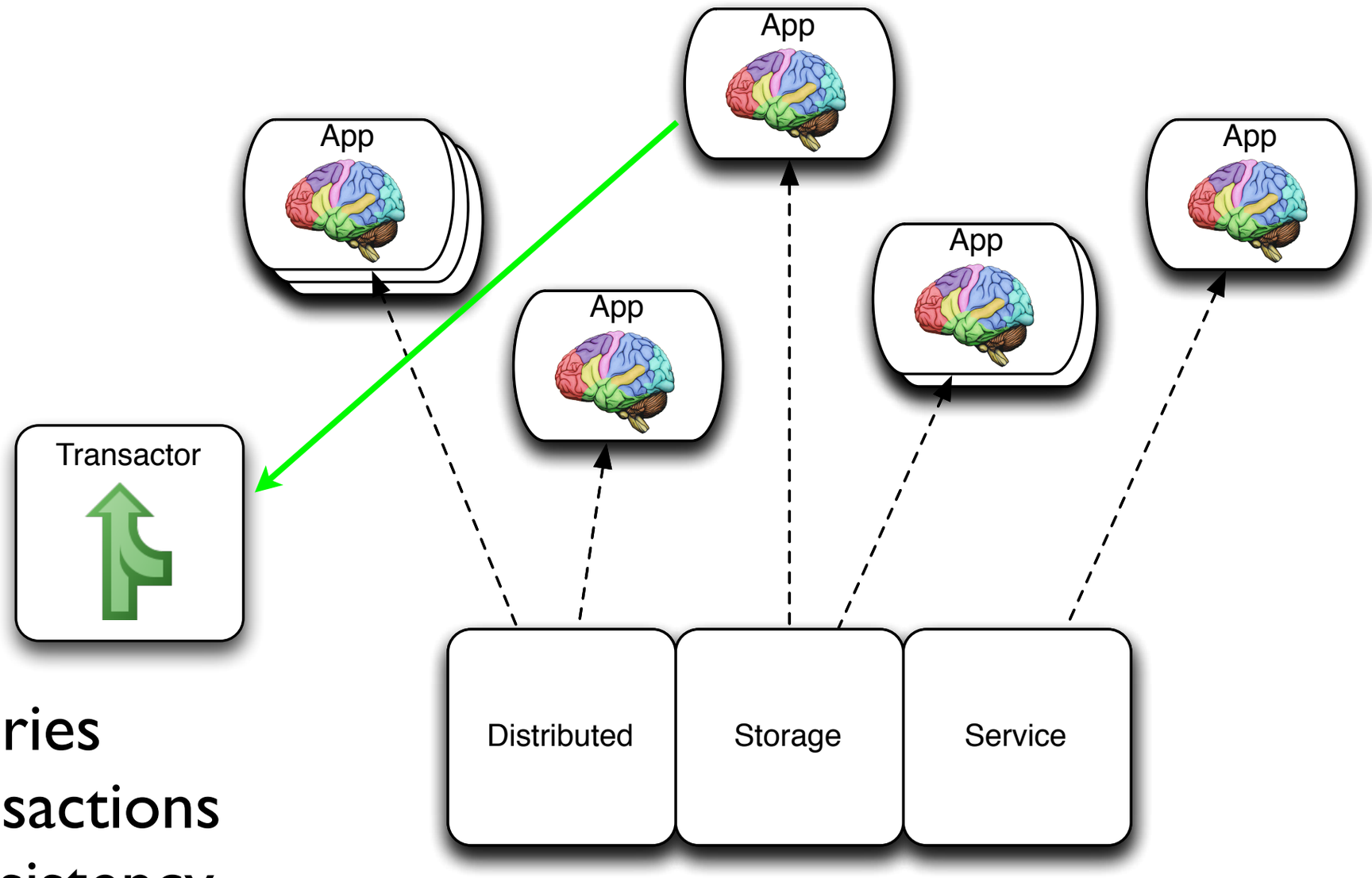
Storage

Datomic Architecture



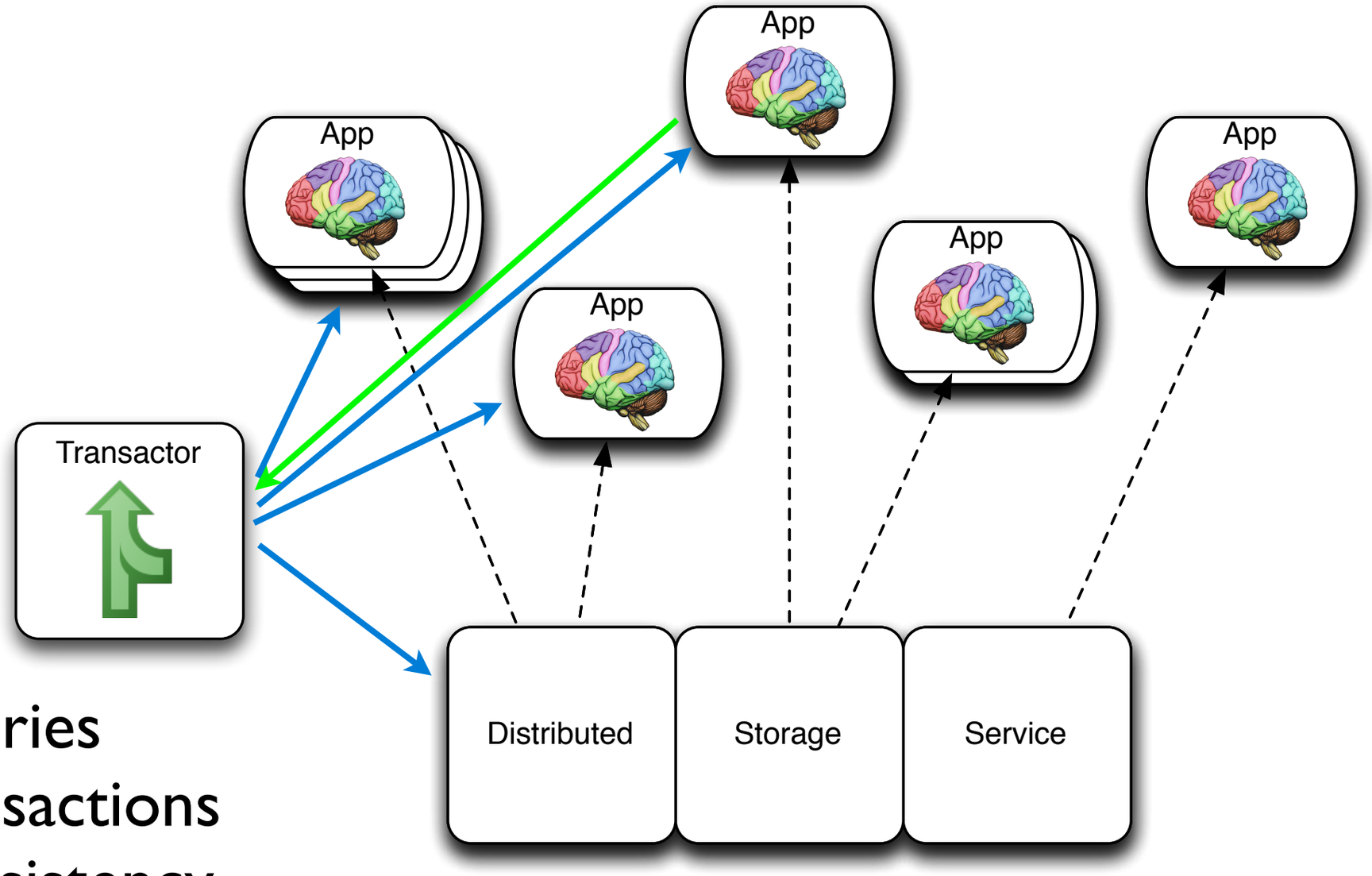
Queries
Transactions
Consistency
Storage

Datomic Architecture



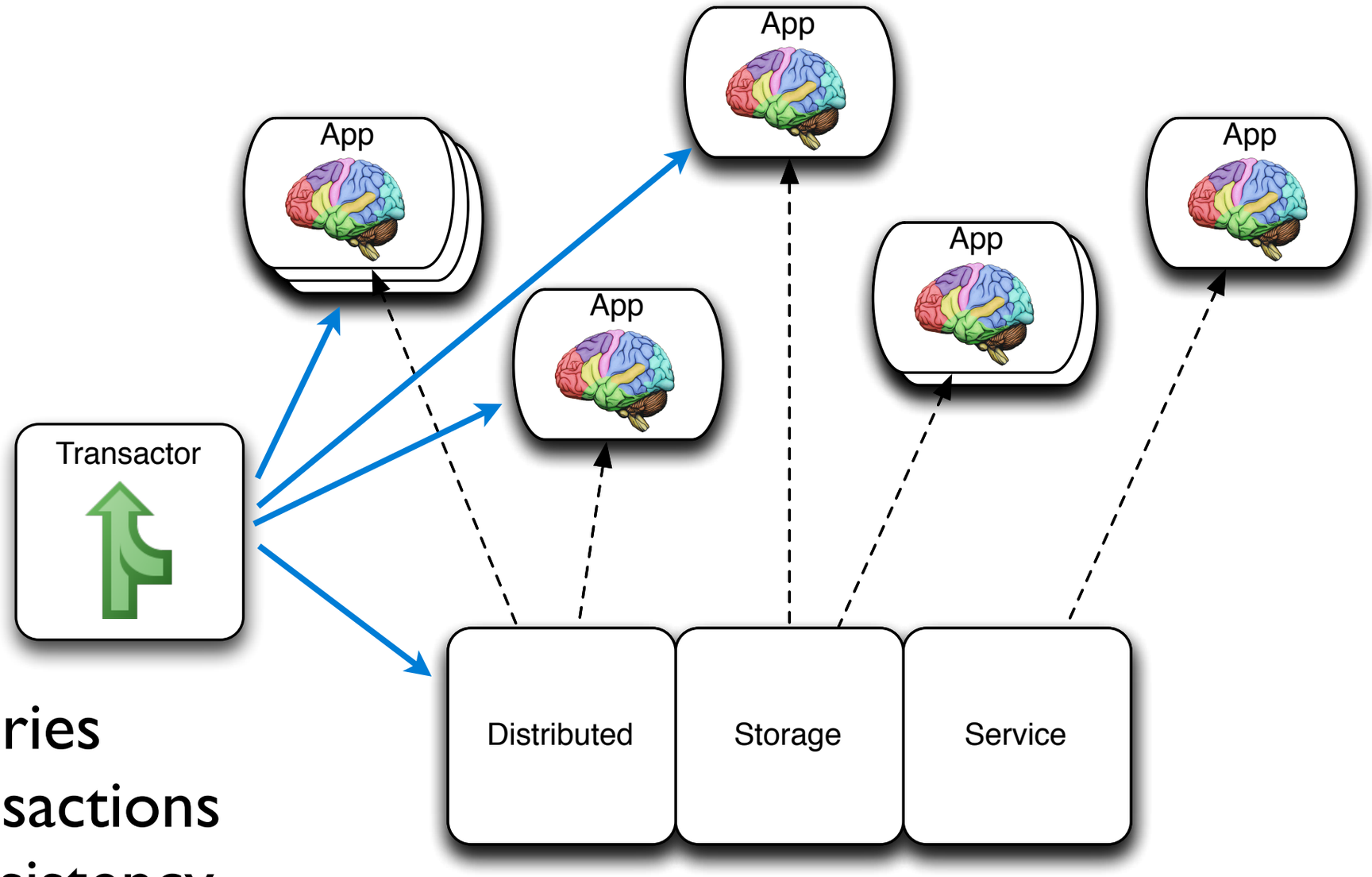
Queries
Transactions
Consistency
Storage

Datomic Architecture



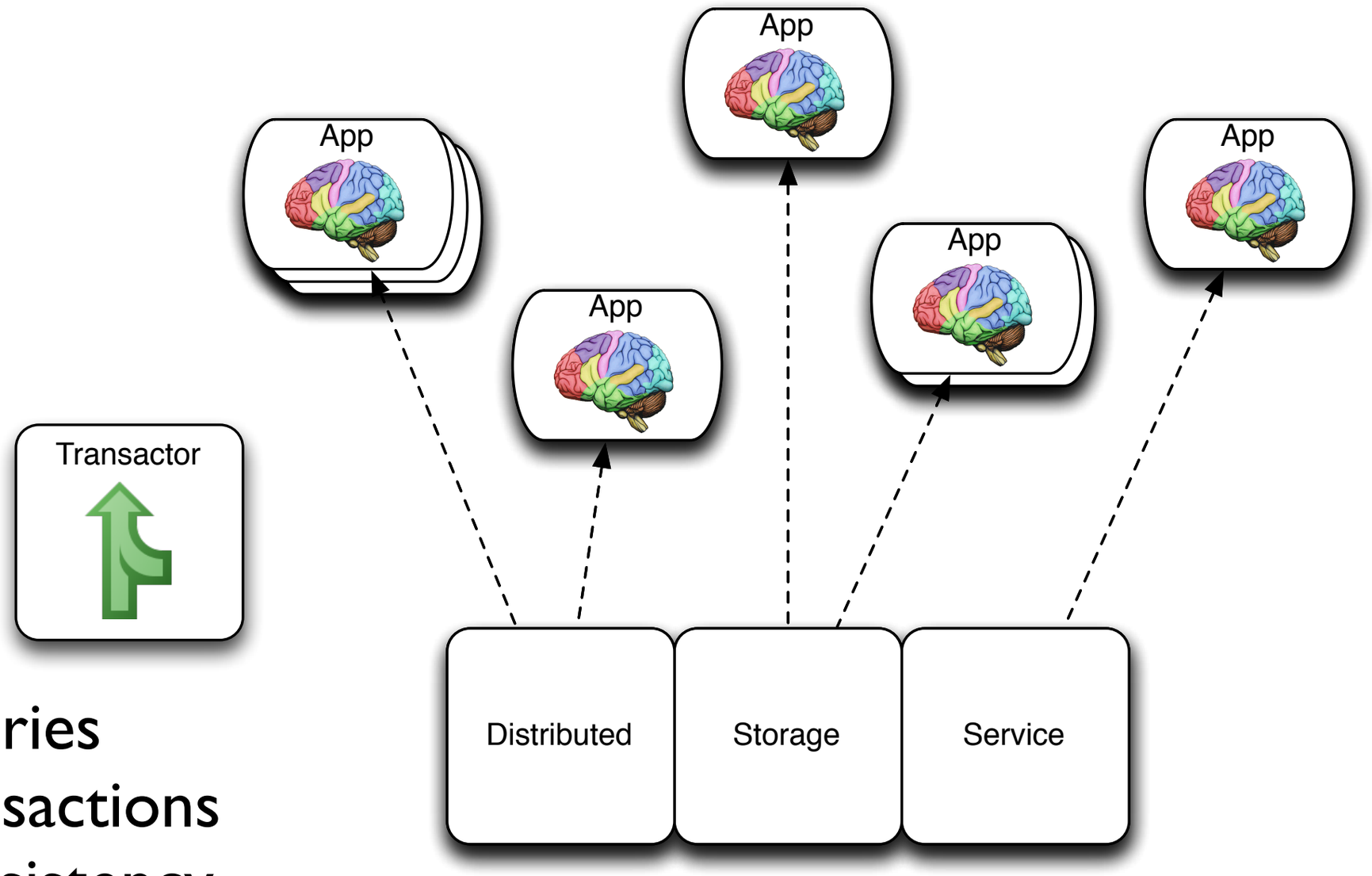
Queries
Transactions
Consistency
Storage

Datomic Architecture



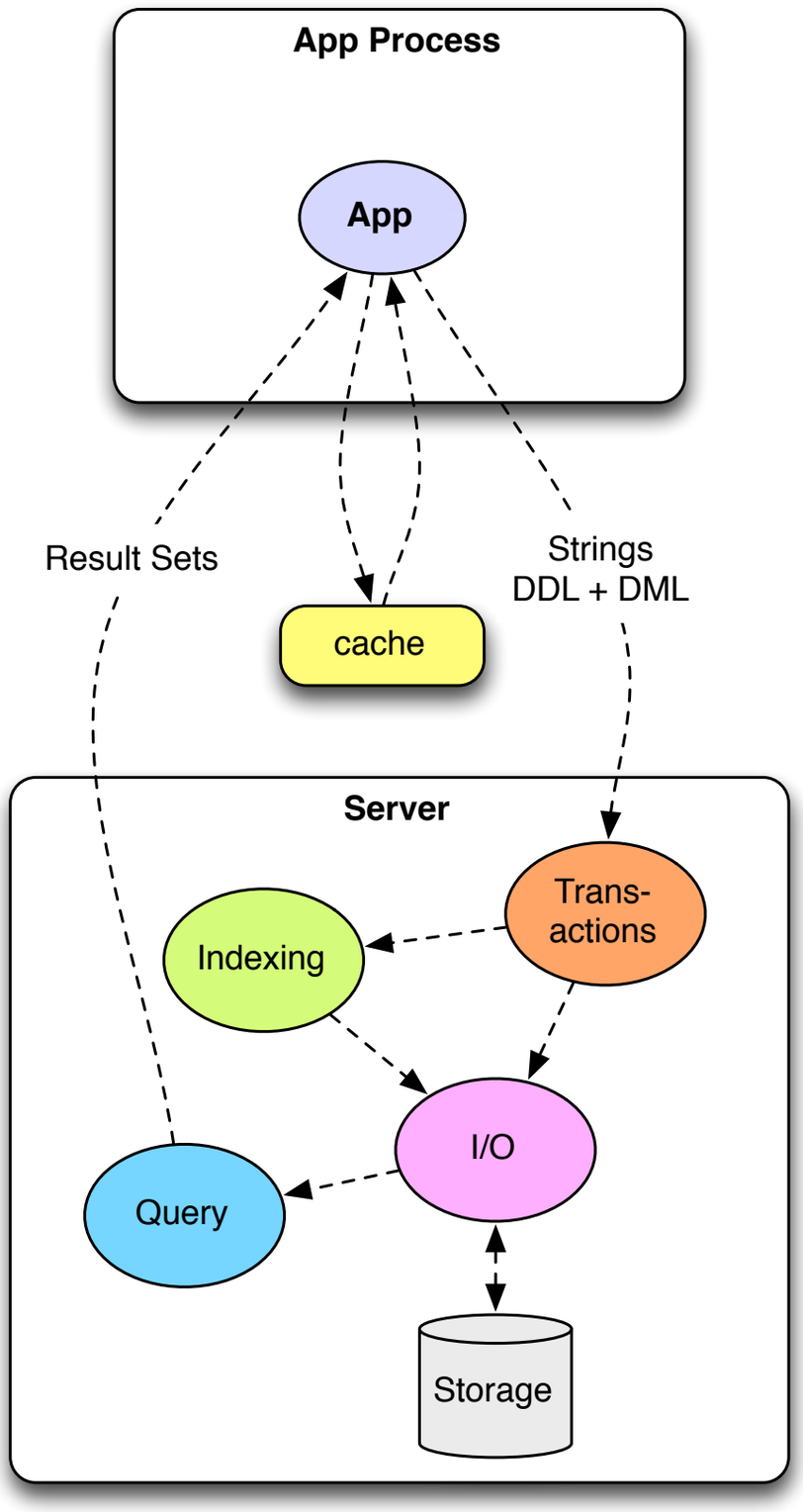
Queries
Transactions
Consistency
Storage

Datomic Architecture

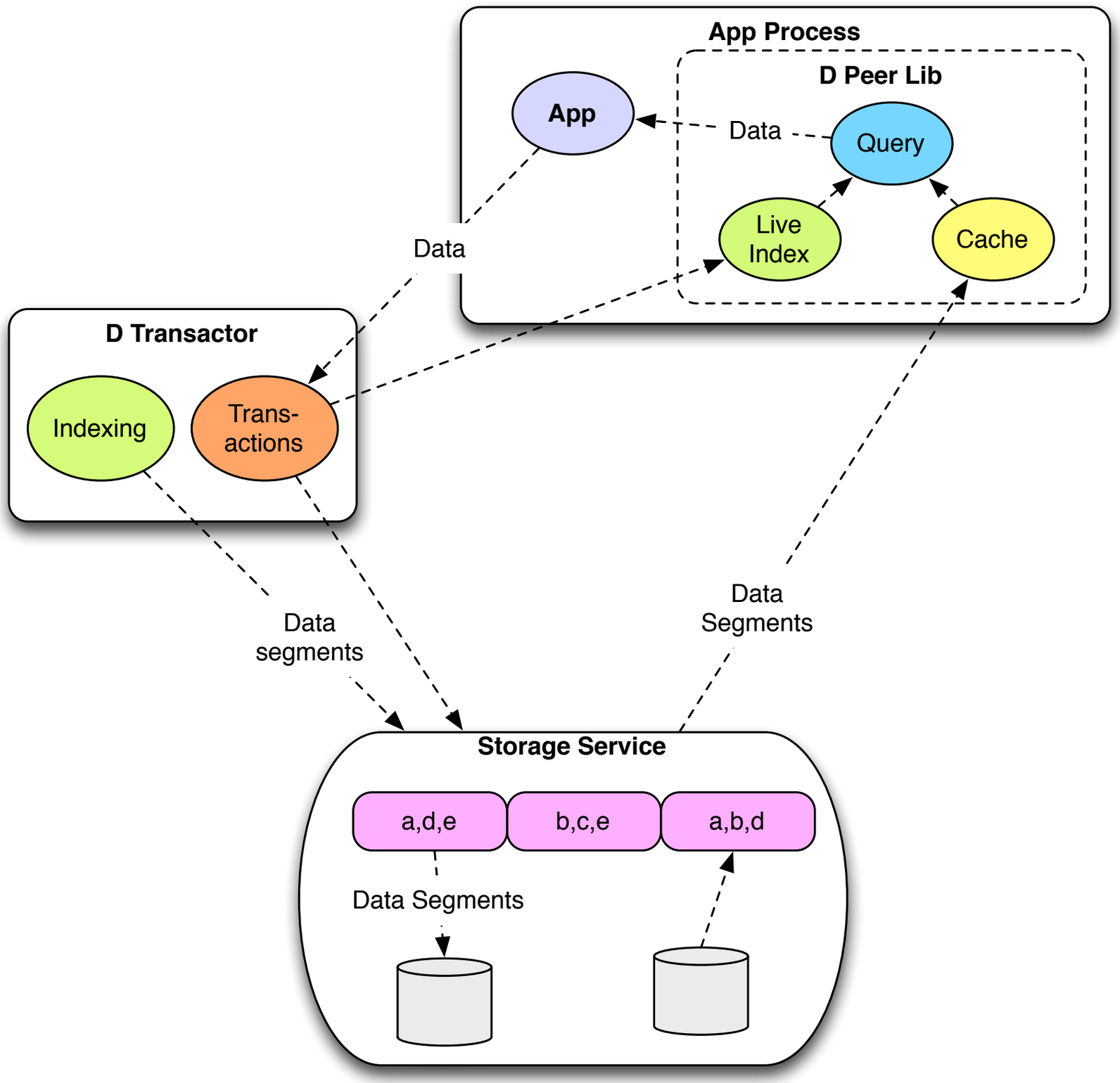


Queries
Transactions
Consistency
Storage

The Database, Deconstructed



Traditional DB



Datomic

Designed for the Cloud

- Ephemeral instances, unreliable disks
- Redundancy in storage service
- Leverages reliable storage services
  - e.g. DynamoDB

Elastic Scaling

- More peers, more power
- Fewer peers, less power, lower cost
- Demand-driven
  - No configuration

Get Your Own Brain

- Query, communication and memory engine
- Goes into your app, making it a peer
- The db is effectively local
- Ad hoc, long running queries - ok

Logic

- Declarative search and business logic
- The query language is Datalog
  - Simple rules and data patterns
- Joins are implicit, meaning is evident
- db and non-db sources

Perception

- Obtain a queue of transactions
  - not just your own
- Query transactions for filtering/triggering

Consistency

- ACID transactions add new facts
- Database presented to app as a value
- Data in storage service is immutable

Programmability

- Transactions/Rules/Queries/Results are data
- Extensible types, predicates, etc
- Queries can invoke your code

A Database of Facts

- A single storage construct, the datom
  - Entity/Attribute/Value/Transaction
- Attribute definition is the only 'schema'

Adaptability

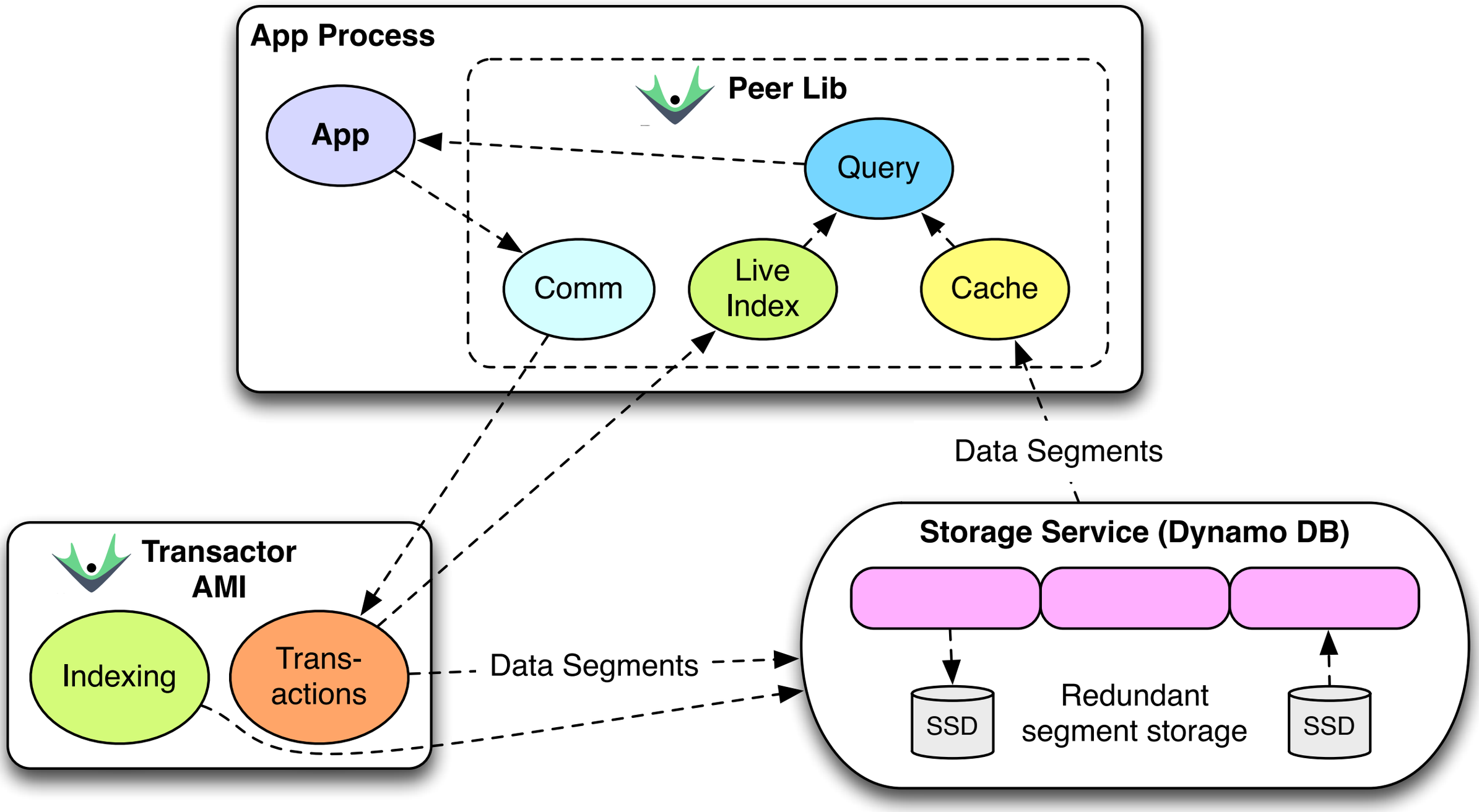
- Sparse, irregular, hierarchical data
- Single and multi-valued attributes
- No structural rigidity

Time Built-in

- Every datom retains its transaction
- Transactions are totally ordered
- Transactions are first-class entities
- Get the db as-of, or since, a point in time

Implementation

Architecture



State

- Immutable, expanding value
- 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

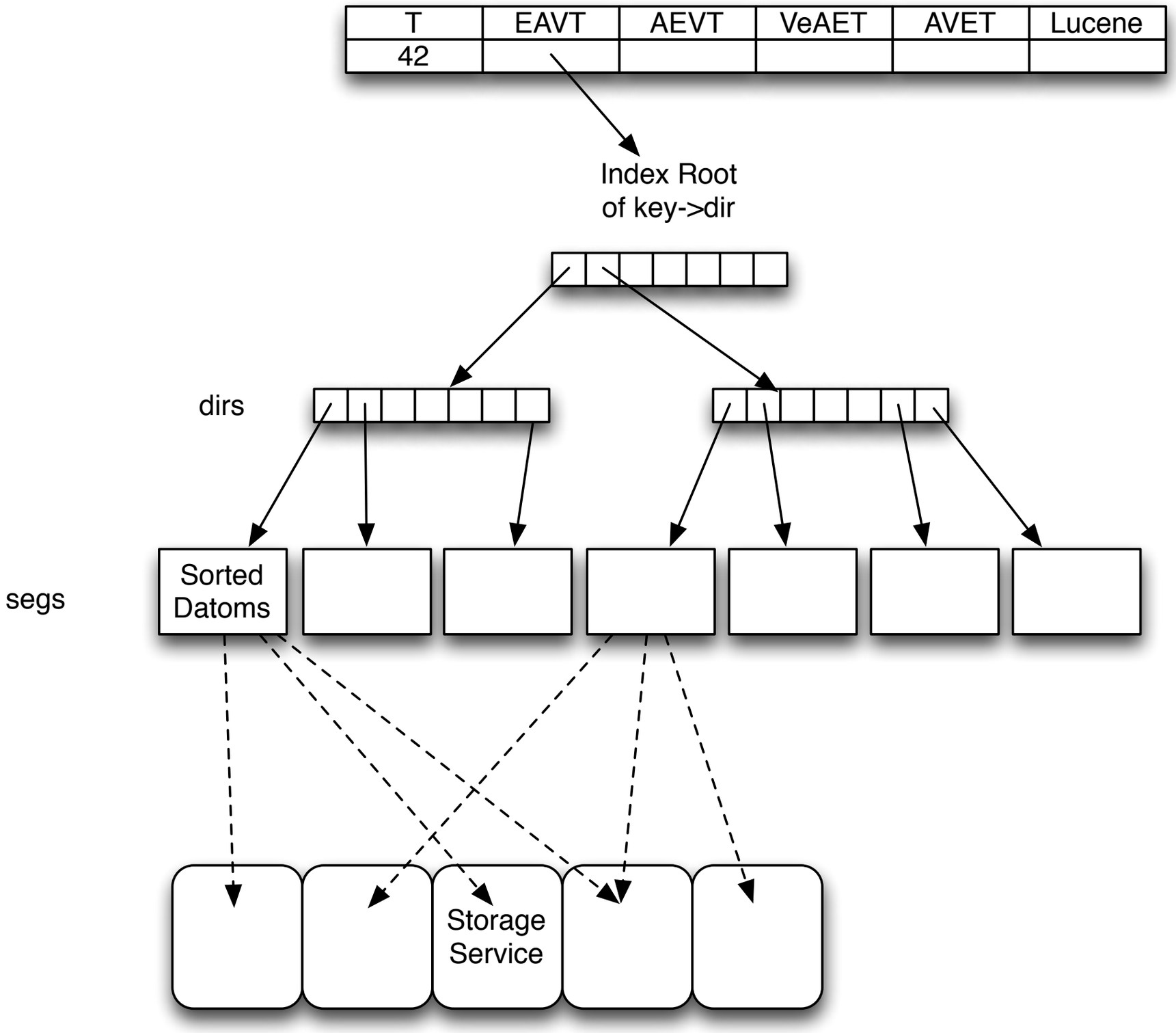
Memory Index

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

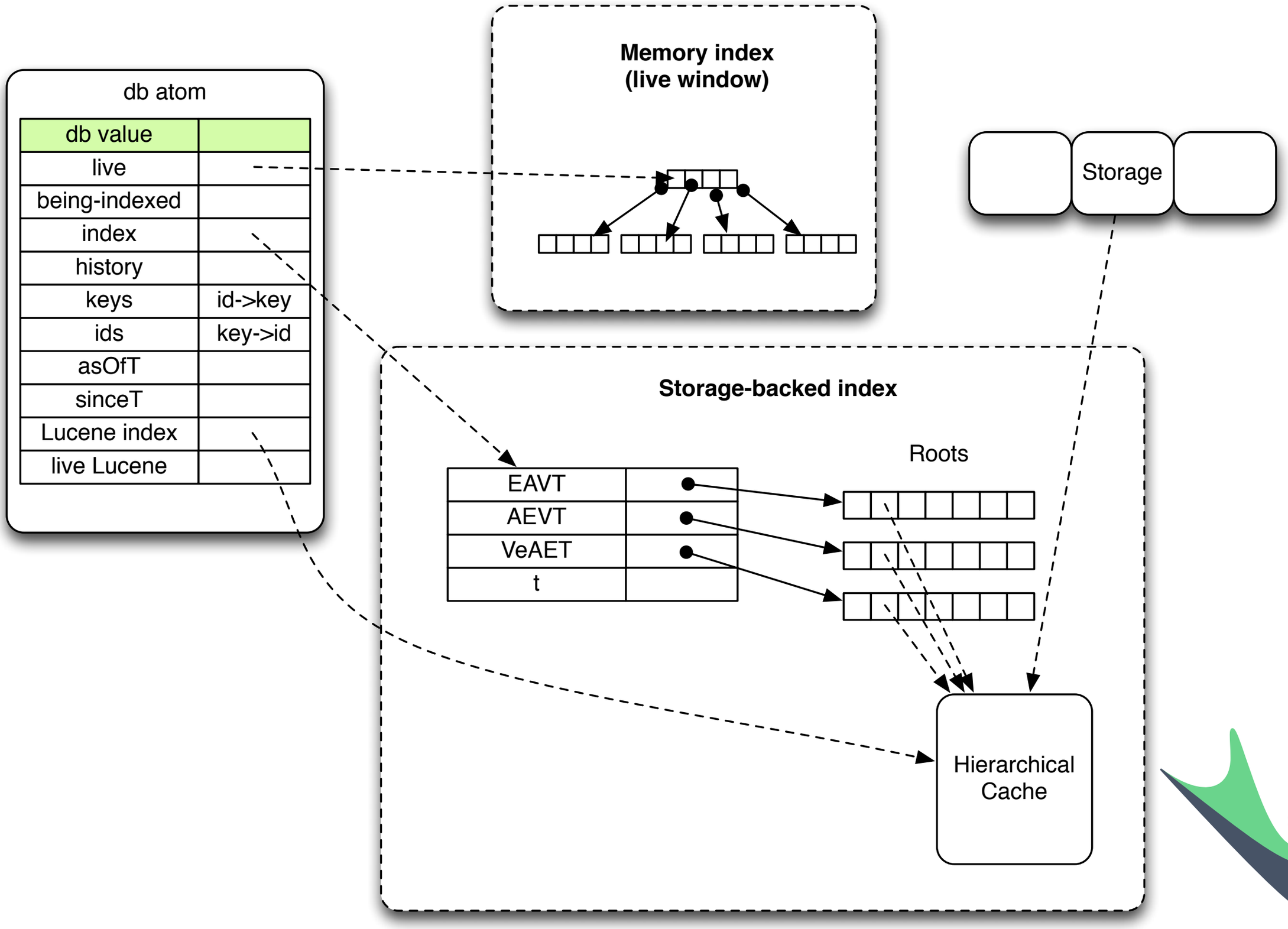
Storage

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

Index Storage



What's in a DB Value?

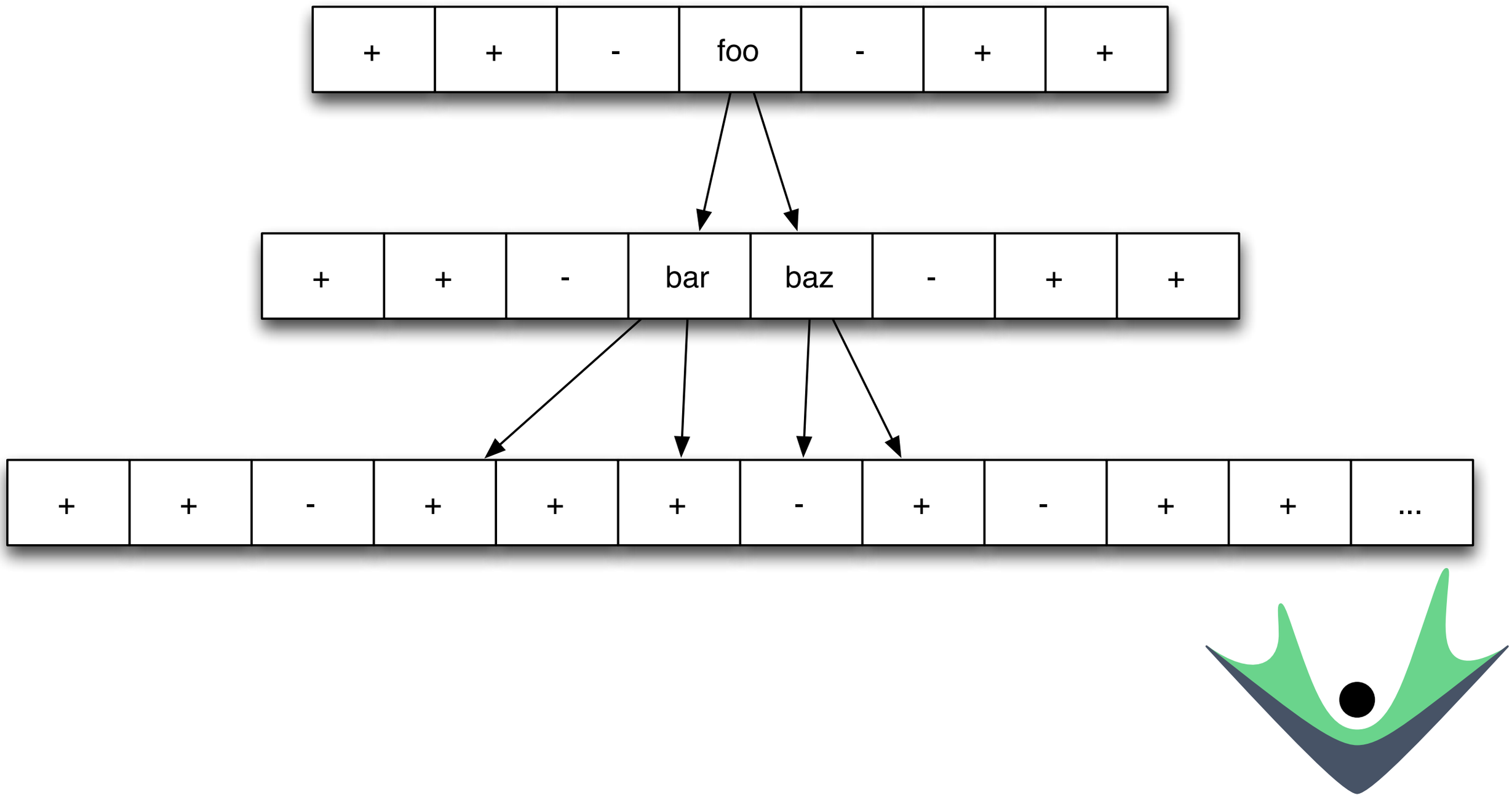


Process

- Assert/retract can't express transformation
- Transaction function:

  (f db & args) -> tx-data
- tx-data: assert|retract|(tx-fn args...)
- Expand/splice until all assert/retracts

Process Expansion



Transactor

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

Transactor Implementation

- HornetQ for transaction communication
- Extensive internal pipelining - j.u.c. queues
- Async message decompression
  - transaction expansion/application
  - encoding for, communication with storage
- Java interop to storage APIs

Indexing

- Extensive use of laziness
- Parallel processing
- Parallel I/O
- Async, rejoins via queue

Declarative Programming

- Embedded Datalog
- Takes data sources and rule sets as args
- Extended to work with scalars/collections
- Expression clauses call your code

Datalog Implementation

- Data driven, in and out
- Query/Subquery Recursive (QSQR)
  - Dynamic, set oriented
- DB joins leverage indexes
- Expressions use Clojure compiler
  - caching of transforms at all stages

Over Here

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

Peer Implementation

- HornetQ for transaction communication
- Google Guava caches
- Java APIs for storage
- Entities are like multimaps
  - key -> value(s)
  - reverse attrs

Consistency and Scale

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

Testing

- test.generative was born here
- Functional tests
- Simulation-based testing

Simplicity is Agility

- Key protocols extremely small (< 7 fns)
- Memory, embedded SQL, remote SQL, Infinispan, DynamoDB
- Move from our own dynamo cluster to DynamoDB:
  - 2 weeks
- Support PostgreSQL, Infinispan
  - 1 day each

Leverage

✓ Read/print data

✓ Embedded language

✓ Runtime compilation

✓ Extend standard interfaces/protocols

✓ Interop

✓ State model - extended

Summary

- Clojure was made for this kind of app
- Fast enough at all levels
- Most key subsystems < 1000 lines
- A ton of concurrency, no sweat
- Leverage interop - Hornetq, Guava etc
- Startup time could be better
- Datomic is Simple