Datahike Life Lessons

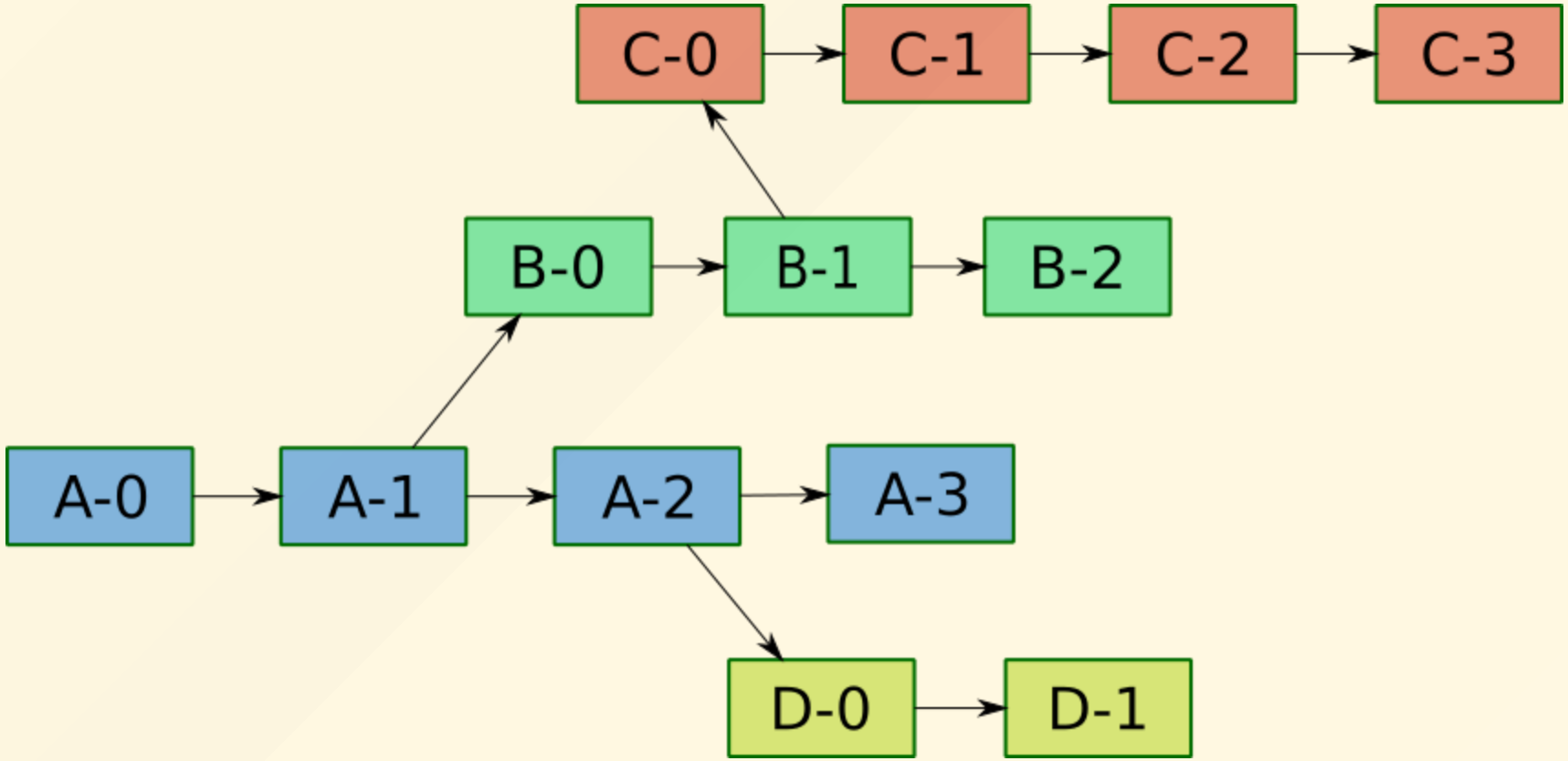
Exploring Datahike through Conway's Game of Life

Presentation

- Introduction
- Wins
- Puzzlers (the hard parts)
- Development and Tooling

Learning Datahike

Conway Game of Life with time travel and branching



What is Datahike?

Open Source Version of Datomic

Immutable Accrete Only Database (Buzzword Bingo)

Basic interaction

```
(transact db [{:storage/location "closet" :storage/volume 300}])
(q '[:find ?eid ?volume
     :where
     [?eid :storage/location "closet"]
     [?eid :storage/volume ?volume]])
; => #{[1 300]}
```

Game of Life

Board is a set of points (x, y) coordinates

```
(def board #{[0 1] [1 1] [2 1] [2 2]})
(board->db board "user/life" db)
(db->board "user/life" db)
; => #{[0 1] [1 1] [2 1] [2 2]}
```

Datahike Wins

Unique Fields

Automatic upsert capability to entities and easy human readable entity reference

```
{:game/name "user/life"
 :game/board <board pieces>}

(entity [:game/name "user/life"])
```

Ala Carte Schema

Each Entity (record) is composed of chosen fields. No extra or missing fields

```
{:board/x 1 :board/y 1} ; Old Piece (cell) fields
{:board/x 1 :board/y 1 :piece/hash 670845861} ; New Piece (cell) fields
```

Index Access

```
(datoms @db {:index :avet :components [:game/name "user/life"]})
```

Combined with unique fields meant I got a list of all transactions for an entity for free

Query Access

```
(q
    '[:find ?x ?y
      :in $ ?game
      :where
      [?e :game/name ?game]
      [?e :game/pieces ?pieces]
      [?pieces :board/x ?x]
      [?pieces :board/y ?y]]
    @db "user/life")
```

Data Structure Database

```
(q
 '[:find ?name ?time
   :in $ ?eid
   :where
   [?eid :alarm/name ?name]
   [?eid :alarm/time ?time]]
 [[1 :alarm/name "morning"]
  [1 :alarm/time "09:15"]] 1)
```

The Hard Part

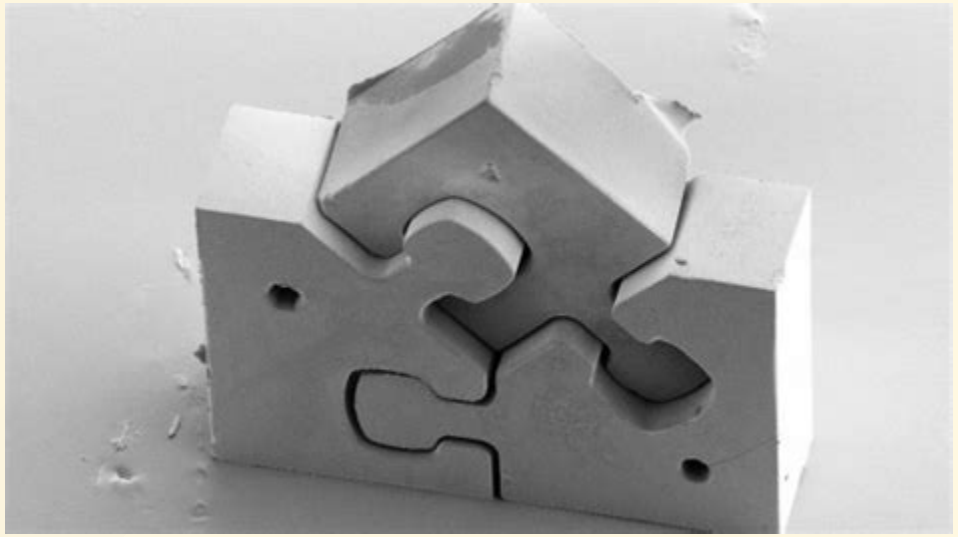
```
{:db/ident         :game/pieces
 :db/valueType     :db.type/ref
 :db/cardinality :db.cardinality/many}
```

Local to Global Thinking

```
(assoc db :game/pieces #{[1 1] [2 2] [3 3]})
```

Works for Clojure Not Datahike

Datahike Puzzlers



Knotty situation

```
(q
  '[:find ?id ?fruit
    :in $ [?fruits ...]
    :where
    [?id :fruit ?fruit]
    [?id :fruit ?fruits]]
  [[1 :fruit :apple]
   [2 :fruit :orange]
   [3 :fruit :pear]]
  #{:apple :pear})
```

```
#{[1 :apple] [3 :pear]}
```

```
(q
  '[:find ?id ?fruit
    :in $ [?fruits ...]
    :where
    [?id :fruit ?fruit]
    (not [?id :fruit ?fruits]])
  [[1 :fruit :apple]
   [2 :fruit :orange]
   [3 :fruit :pear]]
  #{:apple :pear})
```

A) `#{[1 :apple] [3 :pear]}`

B) `#{[2 :orange]}`

C) `#{}`

D) `#{[1 :apple] [2 :orange] [3 :pear]}`

Answer D

```
#{[1 :apple] [2 :orange] [3 :pear]}
```

Interesting Fact

```
(q
  '[:find ?id ?fruit
    :in $ [?fruits ...]
    :where
    [?id :fruit ?fruit]
    (not [?id :fruit ?fruits]])
  [[1 :fruit :apple]
   [2 :fruit :orange]
   [3 :fruit :pear]]
  #{:apple})
```

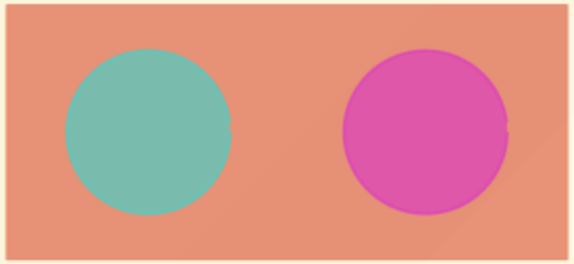
```
#{[2 :orange] [3 :pear]}
```

Datahike Collection Parameters

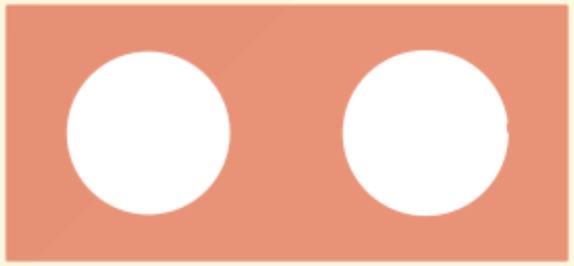
```
--SQL Mental Model
SELECT * FROM fruit WHERE type NOT IN ('apple', 'pear')

--Datomic Collection Parameter Model
SELECT * FROM fruit WHERE type != 'apple'
UNION
SELECT * FROM fruit WHERE type != 'pear'
```

Visual Guide

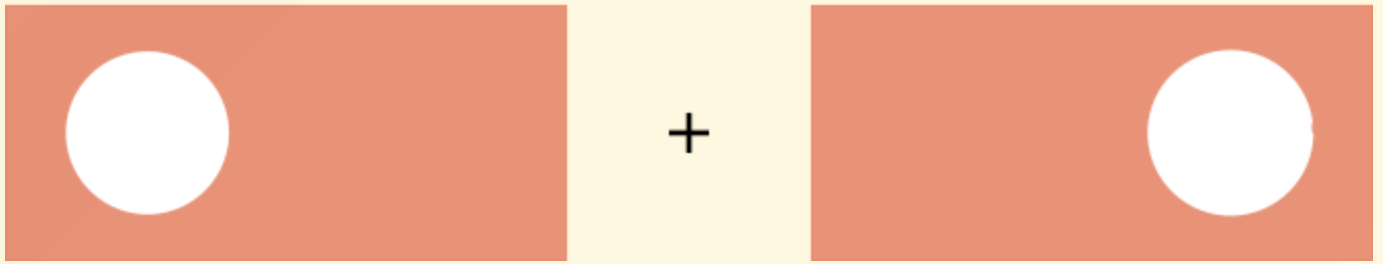
Data Sets



SQL NOT IN



Datomic Collection Not



Use Clojure Sets

```
(q
  '[:find ?id ?fruit
    :in $ ?fruits
    :where
    [?id :fruit ?fruit]
    (not [(?fruits ?fruit)]])
  [[1 :fruit :apple]
   [2 :fruit :orange]
   [3 :fruit :pear]]
  #{:apple :pear})
```

```
#{[2 :orange]}
```

Multitudes

```
(transact
  db
  [{:game/pieces [{:x 1 :y 1} {:x 2 :y 2}]}])

(transact
  db
  [{:game/pieces [{:x 1 :y 1} {:x 3 :y 3}]}])
```

A) `[{:x 1 :y 1} {:x 3 :y 3}]`

B) `[{:x 1 :y 1} {:x 2 :y 2} {:x 3 :y 3}]`

C) `[{:x 2 :y 2} {:x 3 :y 3}]`

D) `[{:x 1 :y 1} {:x 2 :y 2}]`

Answer B

```
[{:x 1 :y 1} {:x 2 :y 2} {:x 3 :y 3}]
```

Only **additions** were done no retractions. Cardinality many does not overwrite!

Database not Memory

```
(transact
  db
  [{:game/pieces [{:x 1 :y 1} {:x 3 :y 3}]}
   [:db/retract <entity id> <{:x 2 :y 2} id>]])
```

Datoms are added or retracted building a concrete representation

board->db

Code Sample time

Time Travel Blues

```
(board->db #{[1 1] [2 2]}) ; 0
(board->db #{[1 1] [3 3]}) ; 1
(board->db #{[1 1]}) ; 2
(db-at-time>board 1) ; using as-of
```

A) `#{[1 1]}`
B) `#{}`
C) `#{[1 1] [2 2] [3 3]}`
D) `#{[2 2] [3 3]}`

Answer C

```
#{[1 1] [2 2] [3 3]}
```

`as-of` only capture **additions** any retractions, like `[2 2]` at time `1`, are not captured

Different histories

`as-of` captures changes in a fields value (e.g. `:shoe/stock` changing from `6`, to `4`, to `8`). `history` allows you to track additions and retractions

db-at-time->board

Code Sample Time

Development Strategy

Planning (Hammock Time)

High Level API functions first. Documentation on function, defaults, and invariants.

Writing things down

Makes things concrete and you're not storing things in your head

REPL Development

New Scratch file every day, don't delete any experiments keep them around.

Reveal

Highly helpful REPL IDE lets you manipulate values not string. Many different formats for data

Reveal

Demo Time

Lessons learned

- Datahike gives you Database super powers
- Datahike Global Database not local memory
- High level writing things down
- REPL with scratch files and Reveal

Questions?

Links

- Game of Life: https://github.com/spinningtopsofdoom/immutable_life
- DataHike: https://github.com/replikativ/datahike
- Datomic Documentation: https://docs.datomic.com/on-prem/
- Reveal: https://vlaaad.github.io/reveal/