How *immutability*, *functional programming*, *databases* and *reactivity* change *front-end*

by Nikita Prokopov
@nikitonsky

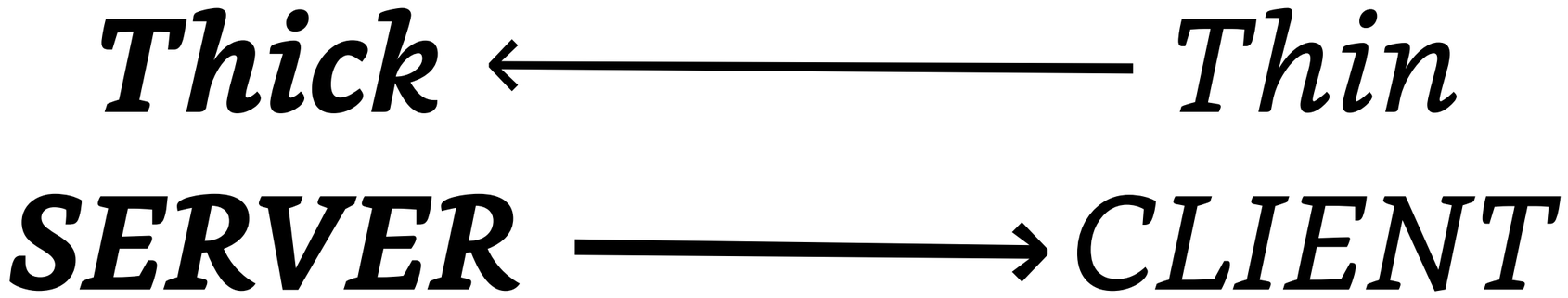

echo


MACHINEZONE


cognician


FIRA CODE
≠ ⟶ ++ :=

State of the Web


Thick
Thin
SERVER
CLIENT

Universal computing device  *Universal computing device*

We don't write
client-server anymore.

We write self-sufficient
applications

Server

Storage
Network
Queues
Cluster

...

Client

UI
Network
?..

When do you want a storage?

Why storage?

Big state

Complex state

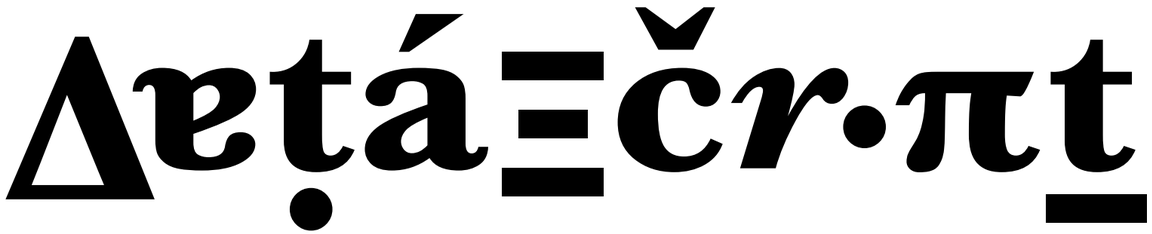
Complex access patterns

Why storage?

Transactions

Persistence

Distributed state



DataScript

Lightweight in-memory data structure that has properties of *the database*

Data Model

<*Entity, Attribute, Value*>

<1, name, Ivan>
<1, age, 20>
<2, name, Oleg>
<2, friends, 3>
<2, friends, 1>
<3, name, Petr>

Data Model

Sparse, irregular data

Multi-valued attributes

Reverse references

Why Indexes?

Fast *lookups* and *scans*

Efficient *row*, *column*, *graph* access

Why Queries?

Declarative, compact, optimizable

What's Datalog?

~ *SQL* + *Recursion*

hierarchies, graph traversals

```
(d/q '[:find    ?dep (sum ?sal) (avg ?sal)
       :in      $ ?me
       :where [?me        :works  ?dep]
              [?person  :works  ?dep]
              [?person  :earns   ?sal]]
     db
     [:email "prokopov@gmail.com])
```

```
[  [ (friend ?a ?b)
     [?a :friend ?b] ]

   [ (friend ?a ?b)
     [?a :friend ?x]
     (friend ?x ?b) ]  ]
```

Superpowers

Queries over collections

Cross-DB joins

User fns/predicates in queries

What's immutability?

$$db_2 = transact(db_1, tx)$$

Database *is* a value

Why immutability?

Testing, mocking

What-if speculations

History tracking

— Efficient immutability ———

$$db_1 \rightarrow db_2 \rightarrow db_3 \rightarrow db_4 \rightarrow db_5 \rightarrow \ldots$$

Transactions

Just data

Custom user fns

Transaction log

```
[ [:db/add       1  :earns   100  ]
  [:db/retract  2  :name  "Ivan"]
  [:db/add      2  :name  "Oleg"] ]
```

Reactive DB

Listen for *transactions queue*

Run queries over *transaction data*

Persistence *(Work in Progress)*

B-Tree indexes

Load segments on demand

Pluggable storages

— Data sync *is hard* ——————————

Syncing two databases *is hard*

Reality is *N-to-N*

Everything will change all the time, in no particular order

Data sync *(Work in Progress)*

Linear logs

Serializable transactions

Optimistic local updates

DataScript

Small, focused, decomplected

Plays well with others

A good foundation

They use DataScript

Precursor

bitfountain

LightMesh

PartsBox

Cognician

I am Fy

— Thanks! ——————————————————

github.com/tonsky/datascript

@nikitonsky