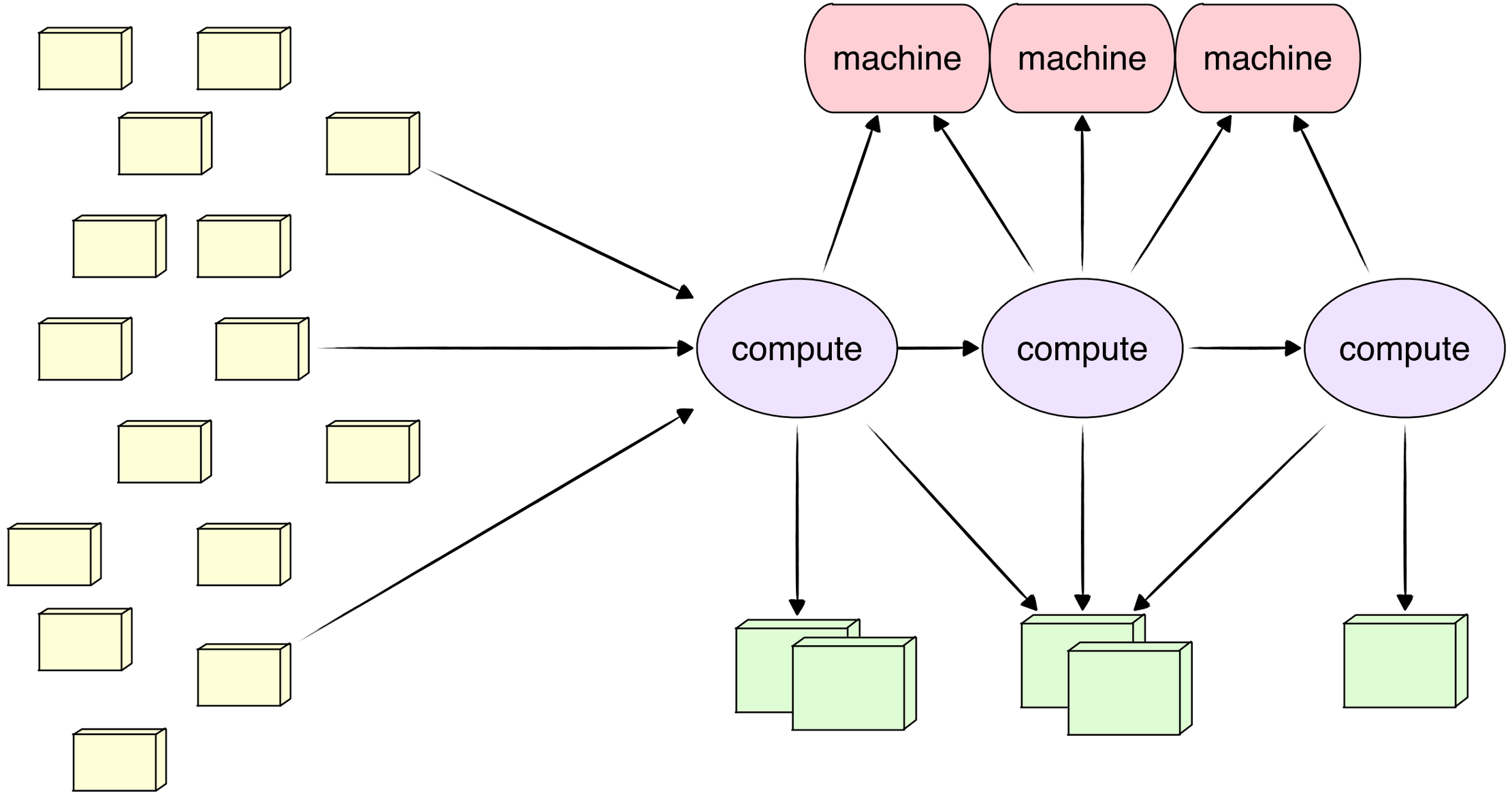
Designing with Data

@MichaelDrogalis

warning!

engineer!

(/ ViaSat Onyx)


machine
machine
machine
compute
compute
compute

data driven designs

“using data”?

- **all** languages have data structures
- typically as a **type** or **typed** collections
- data’s in the database, right?

“using code”

```
(+ 1 2)

(- 4 3)

(* 8 7 6)
```

“using data”

```
{:f :+
 :args [1 2]}

{:f :-
 :args [4 3]}

{:f :*
 :args [8 7 6]}
```

```clojure
(defn calculate [{:keys [f args]}]
  (let [fns {:+ + :- - :* *}]
    (apply (get fns f) args)))
```

Through the OO Lens

	OO	FP (Clj)
multiplicity	high	low
db relationship	mapped	direct
language relationship	indirect	direct
context	out of band	in band

repercussions

anecdotes

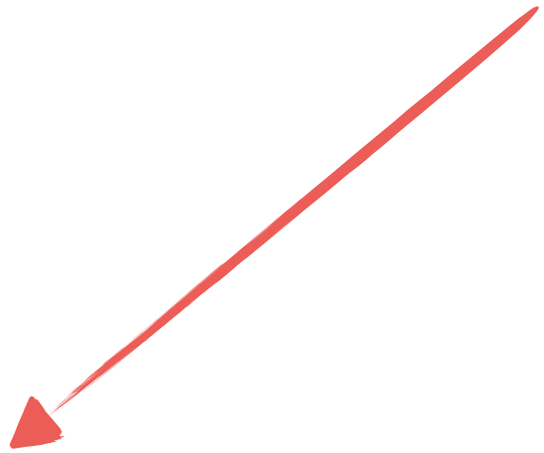
1. generative testing
2. api multiplicity
3. contrasting SQL libs
4. log driven designs
5. 10,000 mistakes
6. first class information models

generative api sql log-driven mistakes info-model

stage 1: thought experiment

Why aren't there more property-based testing frameworks in OO langs?

generative api sql log-driven mistakes info-model



```
(def sort-idempotent-prop
  (prop/for-all [v (gen/vector gen/int)]
                (= (sort v) (sort (sort v)))))

(tc/quick-check 100 sort-idempotent-prop)
```

simples

int

long

double

string

keyword

symbol

...

composites

list

vector

set

map

...

in a type-heavy world...

simples

int

long

double

string

Customer → new generator

DatabaseResult → new generator

proliferation of types: linear effort

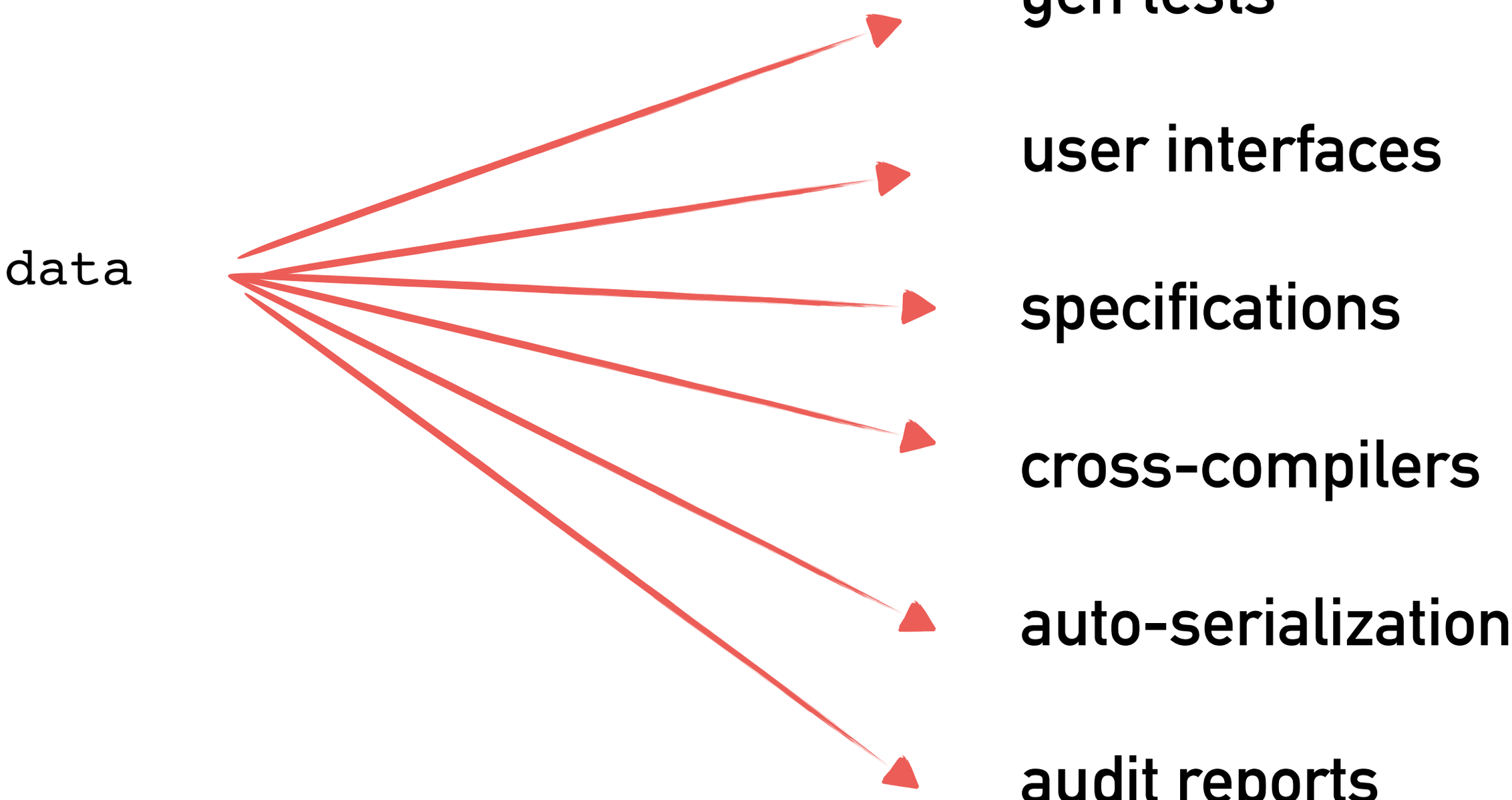
constant number of types: constant effort

*for some classes of problems

generative api sql log-driven mistakes info-model

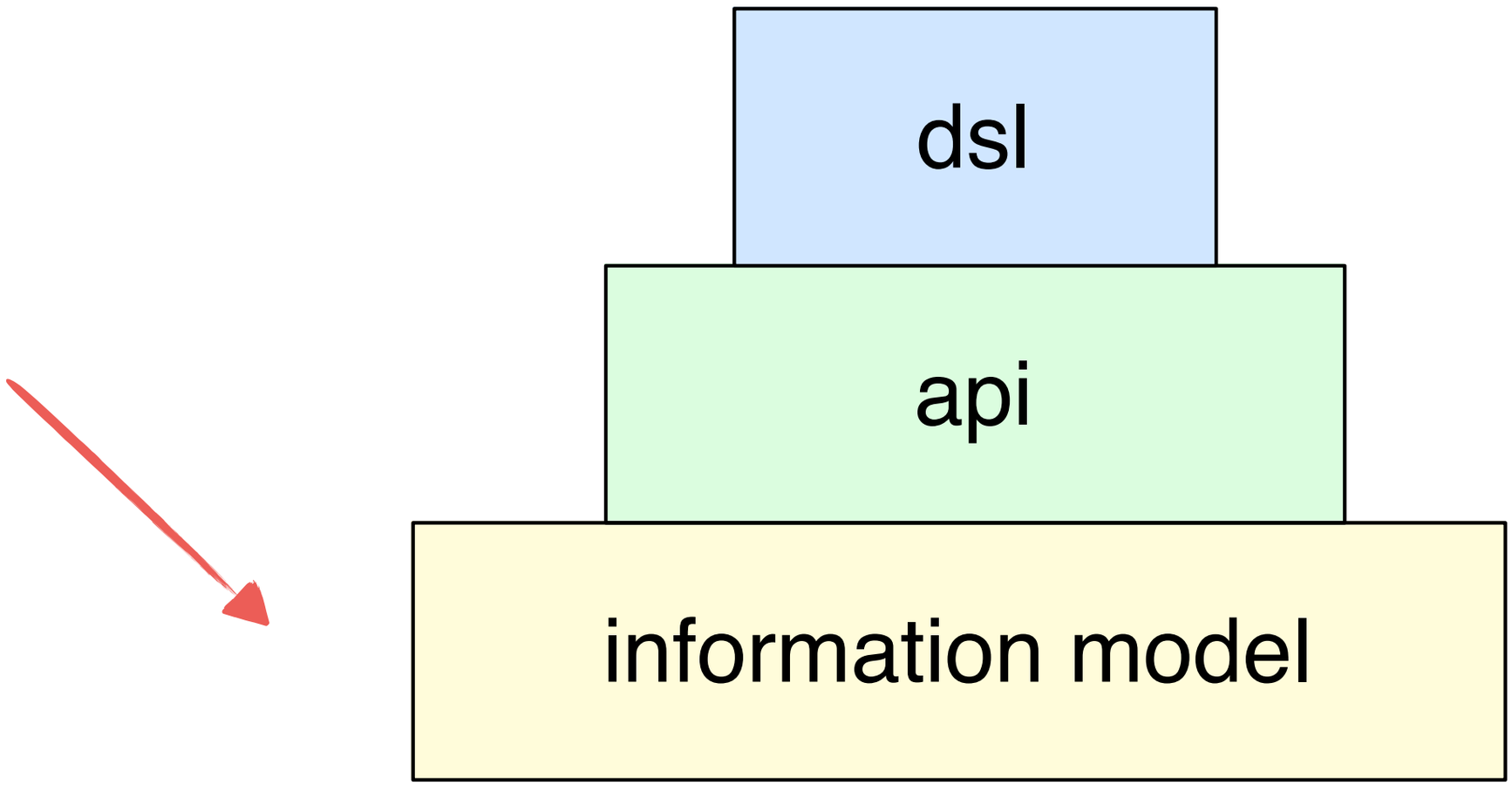
basis for uniformity

generative api sql log-driven mistakes info-model


data
gen tests
user interfaces
specifications
cross-compilers
auto-serialization
audit reports

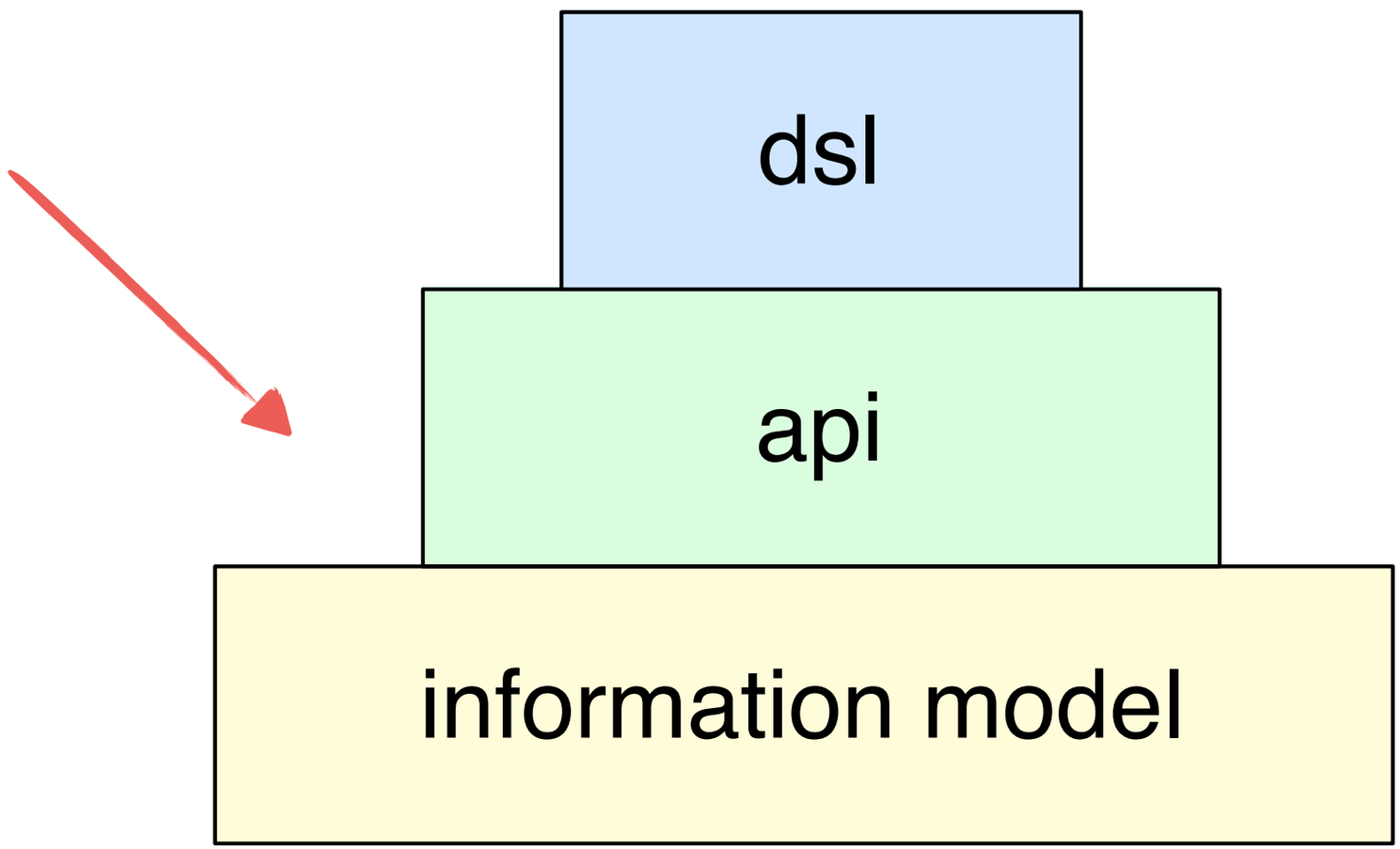
Stage 2: API multiplicity

generative **api** sql log-driven mistakes info-model


dsl
api
information model

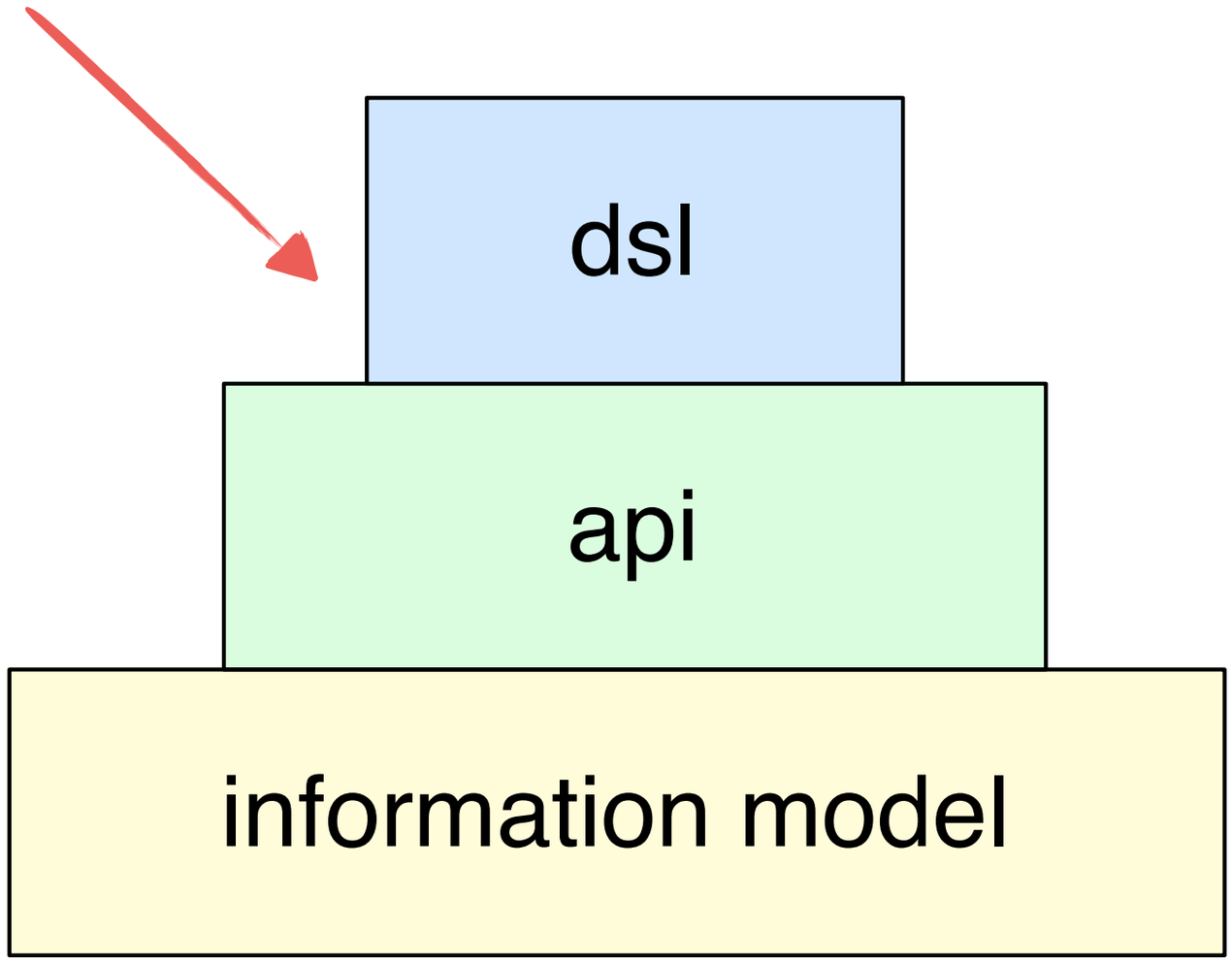
information model

```
[{:flow/from :inc
  :flow/to [:out]
  :flow/short-circuit? true
  :flow/thrown-exception? true
  :flow/predicate [::even-exception?]
  :flow/post-transform ::transform-even}

 {:flow/from :inc
  :flow/to [:out]
  :flow/short-circuit? true
  :flow/thrown-exception? true
  :flow/predicate [::five-exception?]
  :flow/post-transform ::transform-five}]
```


dsl
api
information model

api - zookeeper-clj

```
(defn to-string
  ([^bytes bytes](String. bytes ^String *charset*)))

(defn to-int
  ([bytes]
     (.getInt (ByteBuffer/wrap bytes))))

(defn to-long
  ([bytes]
     (.getLong (ByteBuffer/wrap bytes))))

(defn to-double
  ([bytes]
     (.getDouble (ByteBuffer/wrap bytes))))

(defn to-float
  ([bytes]
     (.getFloat (ByteBuffer/wrap bytes))))

(defn to-short
  ([bytes]
     (.getShort (ByteBuffer/wrap bytes))))

(defn to-char
  ([bytes]
     (.getChar (ByteBuffer/wrap bytes))))
```

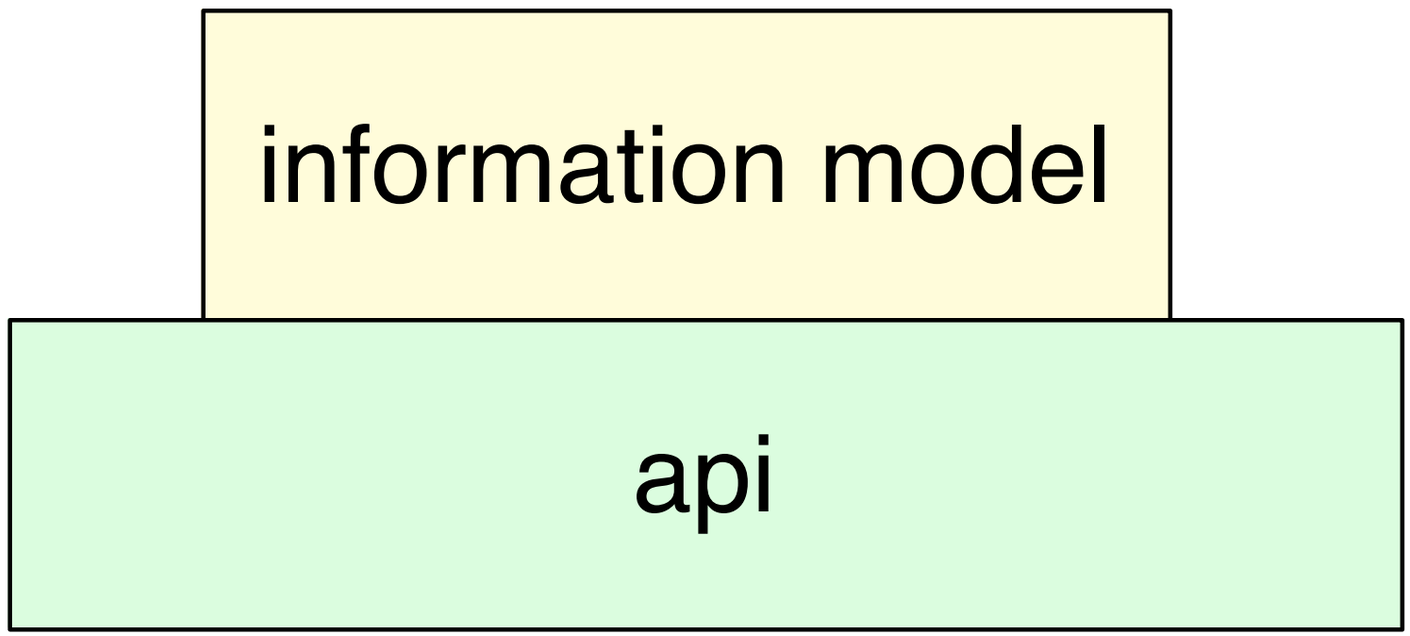

dsl
api
information model

dsl

```
CREATE TABLE recentMeetups
  (id int PRIMARY KEY AUTO_INCREMENT,
   groupId VARCHAR(32),
   groupCity VARCHAR(32),
   category VARCHAR(64));
```

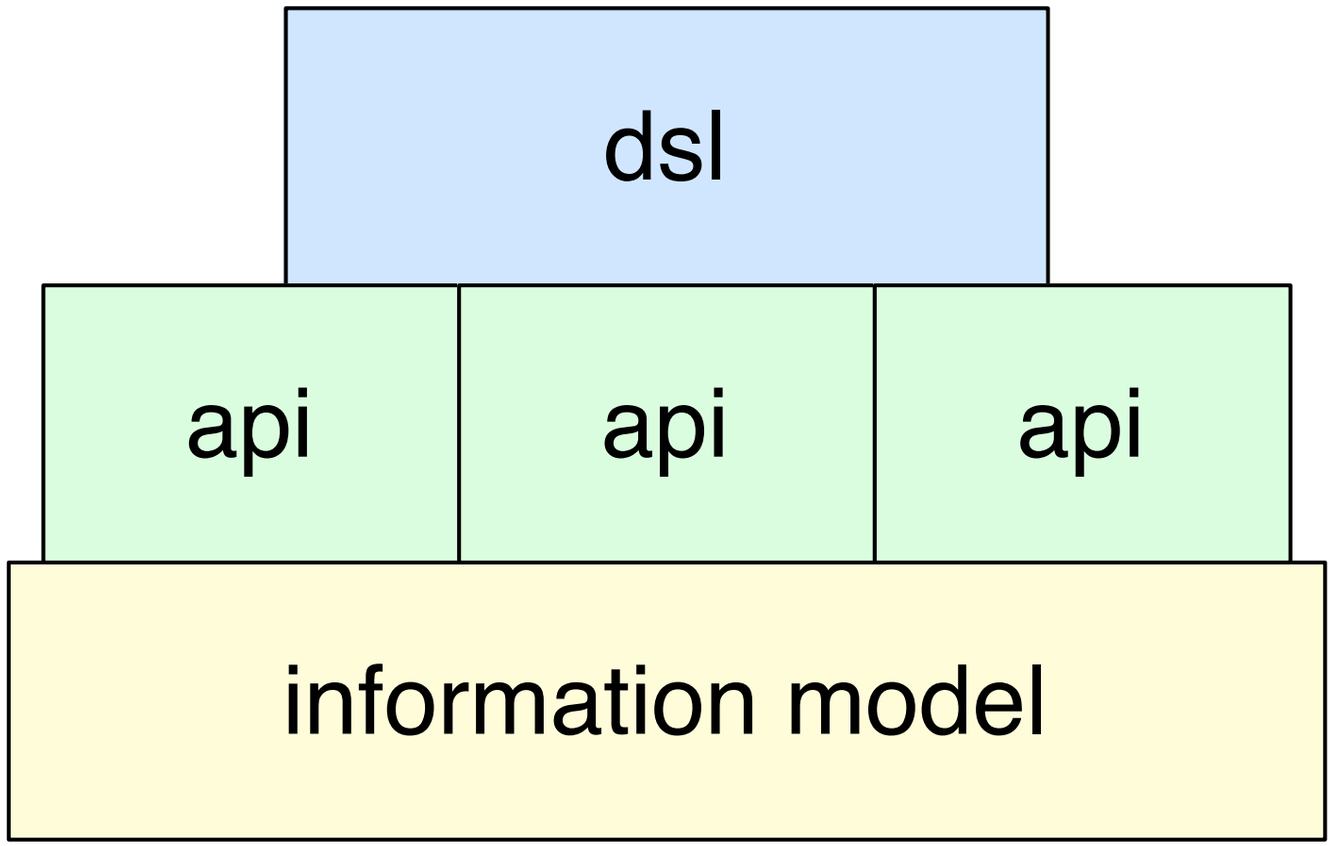
SQL!

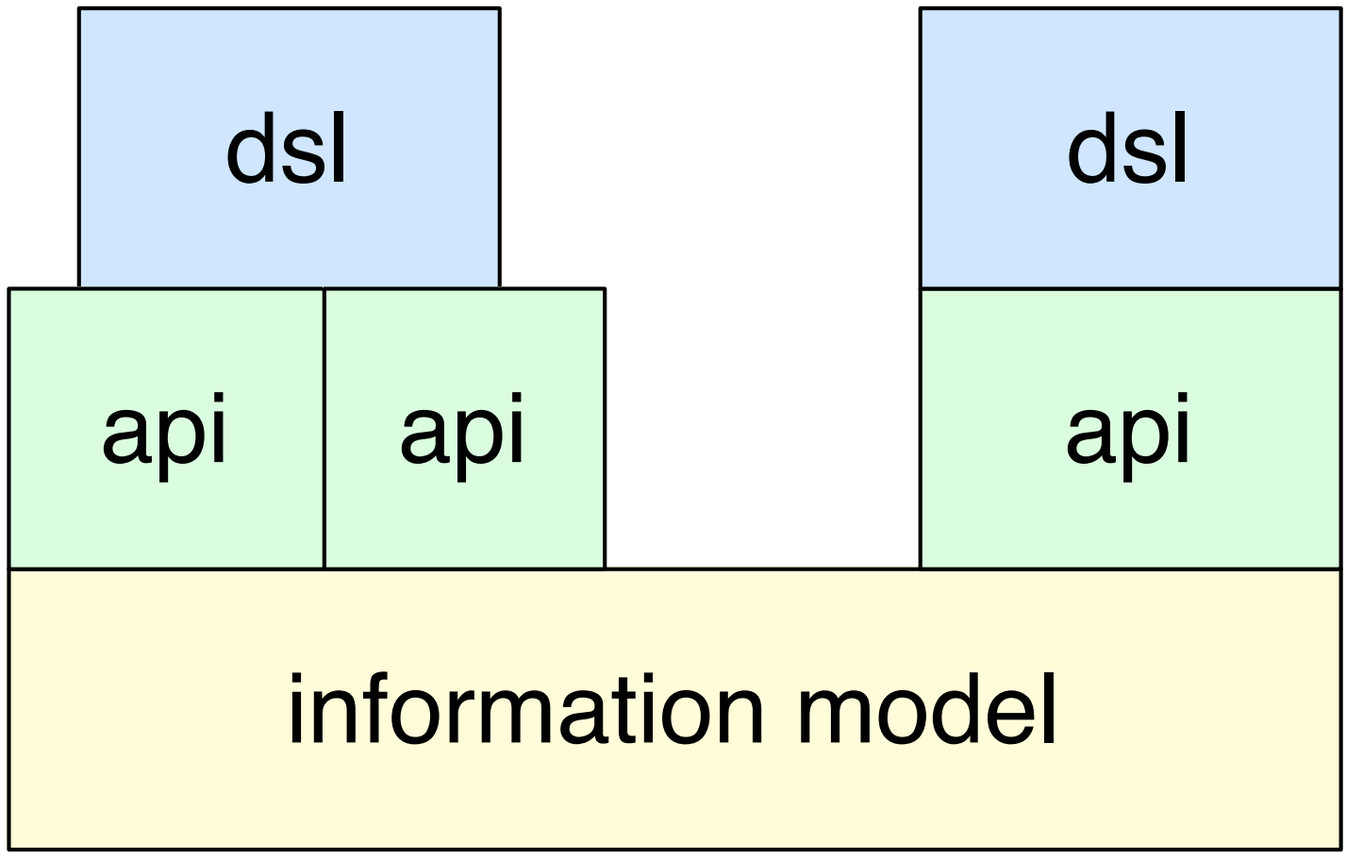
many Cljs -> JVM apps?



which is worse?

generative api sql log-driven mistakes info-model


dsl
api
api
api
information model


dsl
api
api
dsl
api
information model

api 1

```
(defn high-level-api-1 [x y z]
  { ... })

(defn high-level-api-2 [a b]
  { ... })

(defn high-level-api-3 [c]
  { ... })
```

api 2

```
(defn low-level-api-1 [& args]
  { ... })

(defn low-level-api-2 [x]
  { ... })

(defn low-level-api-3 [m n]
  { ... })

(defn low-level-api-4 [a & more]
  { ... })

(defn low-level-api-5 [w x y]
  { ... })
```

everyone trying to do the right thing can lead to conflict

generative api sql log-driven mistakes info-model

information models support views *over* the system

traditional design layers “under”

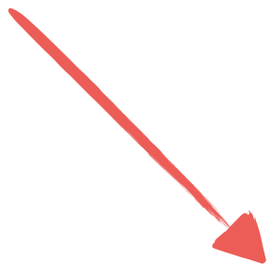
generative **api** sql log-driven mistakes info-model

Stage 3:

HoneySQL vs Korma

generative api **sql** log-driven mistakes info-model

Korma



```
(defentity users)

(select users
        (where {:active true})
        (order :created)
        (limit 5)
        (offset 3))
```

HoneySQL

```
(sql/format
 {:select [:a :b :c]
  :from [:users]
  :where [:= :active true]
  :order-by :created
  :limit 5
  :offset 3})
```

“Then I could write ‘`def-...`”

“Then I could write ‘`def-...`”

generative api **sql** log-driven mistakes info-model

“… I could write …”

generative api **sql** log-driven mistakes info-model

me

generative api sql log-driven mistakes info-model

complex systems are machine-to-machine interactions

emphasis is not on “me”

generative api **sql** log-driven mistakes info-model

```clojure
(defn foo [a b]
  ..)

(defn bar [m n o]
  ...)

(defn baz [x y z]
  ...)

(defn qux [f]
  ...)
```

```clojure
(defn endpoint [{:keys [f args]}]
  ...)

(endpoint {:f :foo
           :args [a b]})

(endpoint {:f :bar
           :args [m n o]})

(endpoint {:f :bar
           :args [x y z]})

(endpoint {:f :qux
           :args [f]})
```

- extensibility isn't positional
  - more functions
  - more args
- discretized load balancing
- cross-language description of endpoints

prefer fewer functional points of contact

generative api sql log-driven mistakes info-model

Stage 4:
log-driven architecture

generative api sql **log-driven** mistakes info-model

```
(.setValue obj 42)
(.databaseCall conn some-data)
  (.updateElement component val-1 val-2 val-3)
```

call-1	call-2	call-3	call-4

```
[{:fn :setValue
  :args [42]}

 {:fn :databaseCall
  :args [some-data]}

{:fn :updateElement
 :args [val-1 val-2 val-3]}]
```

```
[{:fn :setValue
  :args [42]}
```

```
:value.namespace/setValue
```

```
 {:fn :databaseCall
  :args [some-data]}
```

```
:other.ns/databaseCall
```

```
{:fn :updateElement
 :args [val-1 val-2 val-3]}]
```

```
:some.ns/updateElement
```

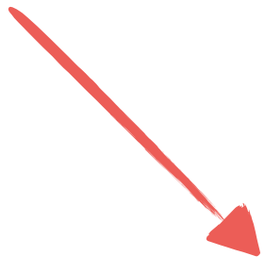
{}

replica

#<The World>

real world

generative api sql **log-driven** mistakes info-model

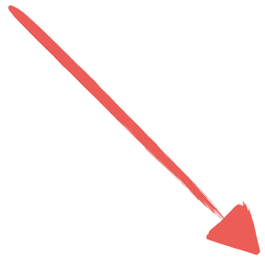


```
[{:fn :setValue
  :args [42]}

 {:fn :databaseCall
  :args [some-data]}

 {:fn :updateElement
  :args [val-1 val-2 val-3]}]
```

generative api sql **log-driven** mistakes info-model

```clojure
(defmulti update-replica
  (fn [world command & args]
    command))

(defmulti do-side-effects!
  (fn [state command before after & args]
    command))

(defmulti reactions
  (fn [command before after & args]
    command))
```

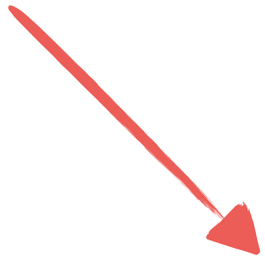
```
(defmethod update-replica:set-value
  [world command & args]
  ...
  new-world)

(defmethod update-replica :database-call
  [world command & args]
  ...
  new-world)

(defmethod update-replica :update-element
  [world command & args]
  ...
  new-world)
```



```
[{:fn :setValue
  :args [42]}

 {:fn :databaseCall
  :args [some-data]}

 {:fn :updateElement
  :args [val-1 val-2 val-3]}]
```

```
:value.namespace/setValue
```

generative api sql **log-driven** mistakes info-model

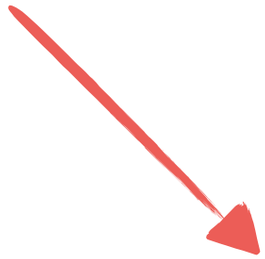


```
[{:fn :setValue
  :args [42]}

 {:fn :databaseCall
  :args [some-data]}

 {:fn :updateElement
  :args [val-1 val-2 val-3]}]
```

```
:value.namespace/setValue
```

```
(update-replica {} :set-value 42)
```

generative api sql **log-driven** mistakes info-model

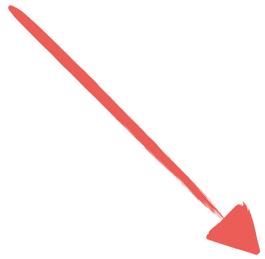


```
[{:fn :setValue
  :args [42]}

 {:fn :databaseCall
  :args [some-data]}

 {:fn :updateElement
  :args [val-1 val-2 val-3]}]
```

```
:value.namespace/setValue
```

```
(update-replica {} :set-value 42)      ⟶ { ... }
```

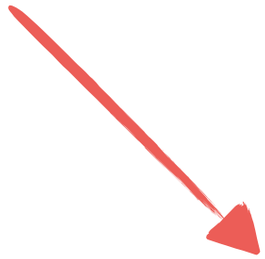
generative api sql **log-driven** mistakes info-model



```
[{:fn :setValue
  :args [42]}

 {:fn :databaseCall
  :args [some-data]}

 {:fn :updateElement
  :args [val-1 val-2 val-3]}]
```

```
:value.namespace/setValue

(update-replica {} :set-value 42)      ⟶ { ... }

(do-side-effects! state :set-value before after 42)
```

generative api sql **log-driven** mistakes info-model

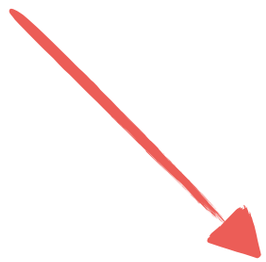


```
[{:fn :setValue
  :args [42]}

 {:fn :databaseCall
  :args [some-data]}

 {:fn :updateElement
  :args [val-1 val-2 val-3]}]
```

```
:value.namespace/setValue
```

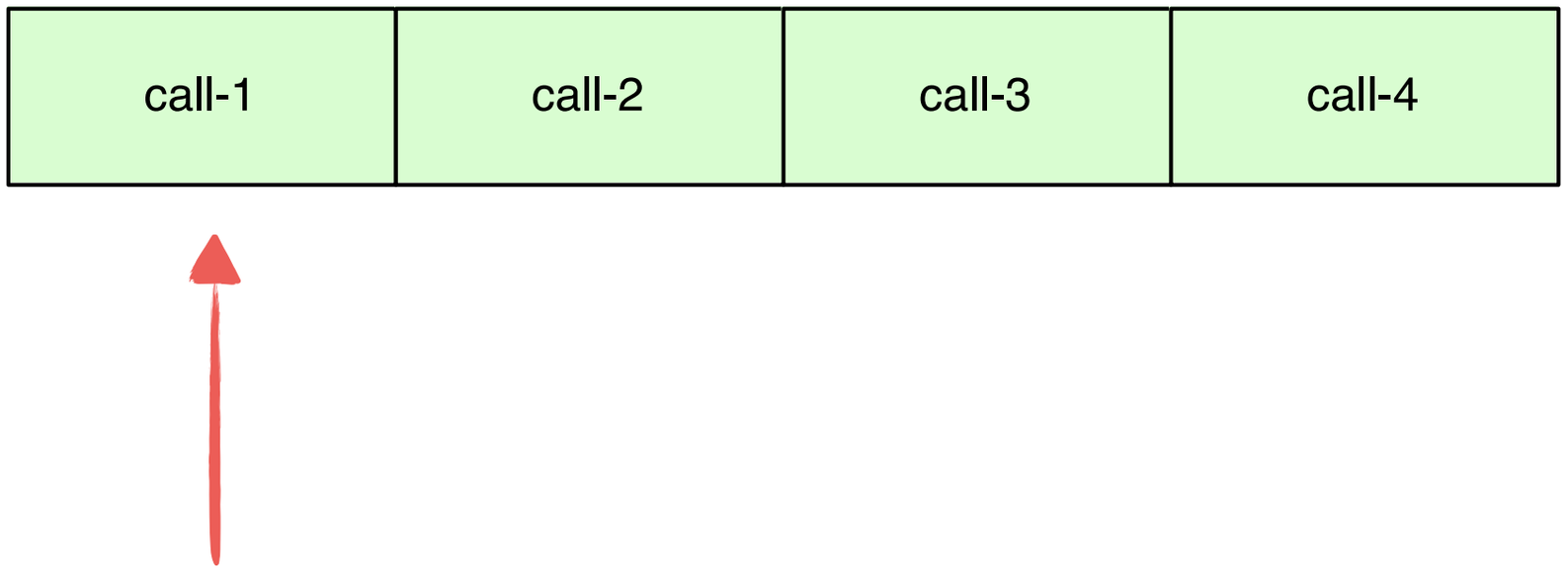
```
(update-replica{} :set-value 42)        ⟶ { ... }

(do-side-effects! state :set-value before after 42)
(reactions :set-value before after 42)
```

generative api sql **log-driven** mistakes info-model



```
[{:fn :setValue
  :args [42]}

 {:fn :databaseCall
  :args [some-data]}

 {:fn :updateElement
  :args [val-1 val-2 val-3]}]
```

generative api sql **log-driven** mistakes info-model


call-1
call-2
call-3
call-4
{}

call-1	call-2	call-3	call-4

{ … }

call-1	call-2	call-3	call-4

{ … }

call-1	call-2	call-3	call-4

{ … }

tooling helps

log entry

diff

```
56 - Sun Jan 24 11:55:37 PST 2016
{:fn :accept-join-cluster,
 :args {:observer #uuid "a027ebff-c35f-42ef-bdbd-4d35e17e08ea",
        :subject #uuid "0715a3fb-92a9-4962-a0d9-76e31b55d096",
        :accepted-observer #uuid "f41d39b3-0e29-4c2f-b947-97f5c6ab5c63",
        :accepted-joiner #uuid "a027ebff-c35f-42ef-bdbd-4d35e17e08ea"},
 :peer-parent #uuid "a027ebff-c35f-42ef-bdbd-4d35e17e08ea",
 :entry-parent 48,
 :message-id 56,
 :created-at 1453665337460}
{:pairs {#uuid "f41d39b3-0e29-4c2f-b947-97f5c6ab5c63" #uuid "0715a3fb-92a9-4962-a0d9-76e31b55d096"},
 :accepted {#uuid "f41d39b3-0e29-4c2f-b947-97f5c6ab5c63" #uuid "a027ebff-c35f-42ef-bdbd-4d35e17e08ea"}}
{:peer-state {#uuid "a027ebff-c35f-42ef-bdbd-4d35e17e08ea" :idle},
 :pairs {#uuid "f41d39b3-0e29-4c2f-b947-97f5c6ab5c63" #uuid "a027ebff-c35f-42ef-bdbd-4d35e17e08ea",
         #uuid "a027ebff-c35f-42ef-bdbd-4d35e17e08ea" #uuid "0715a3fb-92a9-4962-a0d9-76e31b55d096"},
 :accepted nil,
 :peers [nil nil nil nil nil nil nil nil nil nil nil nil #uuid "a027ebff-c35f-42ef-bdbd-4d35e17e08ea"]}
{:exempt-tasks {},
 :peer-sites {#uuid "1bc40864-ff27-41bd-81bb-6c2bf1e20bf4" {:aeron/external-addr "10.0.0.4", :aeron/port 40200, :aeron/acker-id -3172},
              #uuid "17dced13-6c06-4e2e-90ec-8abfefb3d86b" {:aeron/external-addr "10.0.0.6", :aeron/port 40200, :aeron/acker-id -12250},
              #uuid "f41d39b3-0e29-4c2f-b947-97f5c6ab5c63" {:aeron/external-addr "10.0.0.6", :aeron/port 40200, :aeron/acker-id -10015},
              #uuid "0715a3fb-92a9-4962-a0d9-76e31b55d096" {:aeron/external-addr "10.0.0.3", :aeron/port 40200, :aeron/acker-id -7186},
              #uuid "4786738b-5624-49b9-9f5e-85ae145262f1" {:aeron/external-addr "10.0.0.4", :aeron/port 40200, :aeron/acker-id -16247},
              #uuid "a027ebff-c35f-42ef-bdbd-4d35e17e08ea" {:aeron/external-addr "10.0.0.3", :aeron/port 40200, :aeron/acker-id 5905},
              #uuid "dd4b0a36-25c3-4d05-a61b-de028b6fc9e1" {:aeron/external-addr "10.0.0.2", :aeron/port 40200, :aeron/acker-id 29898},
              #uuid "19e8e8d3-5bb5-421c-a845-57f99144fde7" {:aeron/external-addr "10.0.0.3", :aeron/port 40200, :aeron/acker-id -15104},
              #uuid "9ff6b878-c0c8-4d1f-9e95-2b997ec3990a" {:aeron/external-addr "10.0.0.4", :aeron/port 40200, :aeron/acker-id -16579},
              #uuid "b4d5f168-5d9c-47c8-8ff3-36b6fb2b0e0c" {:aeron/external-addr "10.0.0.5", :aeron/port 40200, :aeron/acker-id 16783},
              #uuid "d6bd9770-45d1-41e5-95cc-5672532167fd" {:aeron/external-addr "10.0.0.5", :aeron/port 40200, :aeron/acker-id 3784},
              #uuid "b5fc2dca-0701-4de6-b2e6-bc42e013c855" {:aeron/external-addr "10.0.0.5", :aeron/port 40200, :aeron/acker-id 24414},
              #uuid "f31a9762-ad79-4176-b7f0-2eab040e49a0" {:aeron/external-addr "10.0.0.6", :aeron/port 40200, :aeron/acker-id -3595}},
 :output-tasks {},
 :state-logs-marked #{},
 :job-scheduler :onyx.job-scheduler/balanced,
 :ackers {},
 :saturation {},
 :task-percentages {},
```

replica

split apart the functionally pure from the impure

especially when actions are linearized

generative api sql **log-driven** mistakes info-model

Stage 5:

10,000 mistakes

generative api sql log-driven **mistakes** info-model

the revenge of XML?

generative api sql log-driven **mistakes** info-model

mini-language

generative api sql log-driven mistakes info-model

```
<util:map id="emails" map-class="java.util.TreeMap">
    <entry key="pechorin" value="pechorin@hero.org"/>
    <entry key="raskolnikov" value="raskolnikov@slums.org"/>
    <entry key="stavrogin" value="stavrogin@gov.org"/>
    <entry key="porfiry" value="porfiry@gov.org"/>
</util:map>
```

generative api sql log-driven **mistakes** info-model

```
<util:map id="emails" map-class="java.util.TreeMap">
    <entry key="pechorin" value="pechorin@hero.org"/>
    <entry key="raskolnikov" value="raskolnikov@slums.org"/>
    <entry key="stavrogin" value="stavrogin@gov.org"/>
    <entry key="porfiry" value="porfiry@gov.org"/>
</util:map>
```

generative api sql log-driven **mistakes** info-model

XML-specific editor / IDE?

generative api sql log-driven mistakes info-model

XML != map/list/set . . .

generative api sql log-driven mistakes info-model

beware of implementation details bleeding into data

generative api sql log-driven mistakes info-model

good idea executed poorly

generative api sql log-driven mistakes info-model

```
(+ 1 2)

(- 4 3)

(* 8 7 6)
```

generative api sql log-driven **mistakes** info-model

“data all the things”?

generative api sql log-driven **mistakes** info-model

it's all bytes

generative api sql log-driven **mistakes** info-model

“data” from who’s perspective?

generative api sql log-driven **mistakes** info-model

Stage 6:
first-class information model

generative api sql log-driven mistakes **info-model**

```clojure
(def model
  {:catalog-entry
   {:summary "..."
    :model {:onyx/name
            {:doc "..."
             :type :keyword
             :choices :any
             :tags [:task]
             :restrictions ["Value cannot be `:none`."]
             :optional? false
             :added "0.8.0"}

            :onyx/type
            {:doc "..."
             :type :keyword
             :tags [:task]
             :choices [:input :function :output]
             :optional? false
             :added “0.8.0"}
```

generate schemas

```
(def FlowCondition
  {:flow/from s/Keyword
   :flow/to (s/either s/Keyword [s/Keyword])
   :flow/predicate (s/either s/Keyword [s/Any])
   (s/optional-key :flow/post-transform) NamespacedKeyword
   (s/optional-key :flow/thrown-exception?) s/Bool
   (s/optional-key :flow/action) FlowAction
   (s/optional-key :flow/short-circuit?) s/Bool
   (s/optional-key :flow/exclude-keys) [s/Keyword]
   (s/optional-key :flow/doc) s/Str
   UnsupportedFlowKey s/Any})
```

smarter error messages

```
"Validation of task map failed - :onyx/name must be a keyword,
was a string."
{:section :task-map
 :key :onyx/name
 :cause :type-mismatch
 :doc "..."
 :constraints ["Must be unique across all task maps."]}
```

generate amazing docs

Sections

Summary

All inputs, outputs, and functions in a workflow must be described via a catalog. A catalog is a vector of maps, strikingly similar to Datomic's schema. Configuration and docstrings are described in the catalog.

`:onyx/name` required

The name of the task that represents this catalog entry. Must correspond to a keyword in the workflow associated with this catalog.

Restrictions

- Must be unique across all catalog entries.
- Value cannot be `:none`.
- Value cannot be `:all`.

allowed types :keyword

choices :any

added 0.8.0

Keys

forced documentation

generative api sql log-driven mistakes info-model

cross-compile

generative api sql log-driven mistakes info-model

explicit information models

are leverage for generation

generative api sql log-driven mistakes **info-model**

in conclusion

there is no conclusion

this is a rant about extensibility

Clojure is not an island

don't make your system one

questions?

@MichaelDrogalis