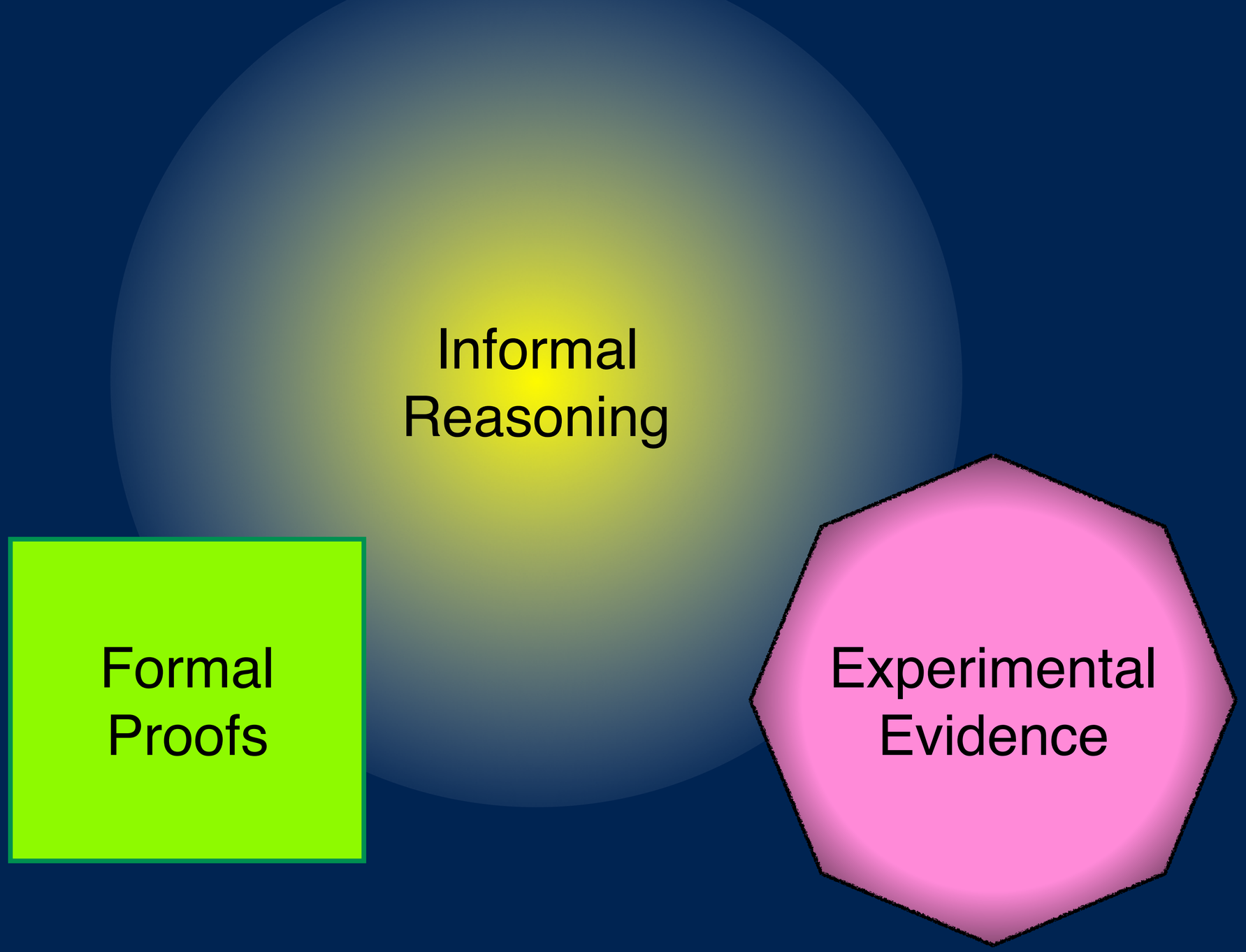
Contracts in Clojure:
Settling Types vs Tests

@jessitron

What do we know?

How do we know it?


Informal
Reasoning
Formal
Proofs
Experimental
Evidence

```
// Scala

def formatReport(data: ReportData): ExcelSheet
```


T.GERRA
SUSSEX

```
(defn format-report [report-data]
  ...)
```

(function-name arg1 arg2)

function application

function definition

```
(defn function-name [param1 param2]
  (println "Hello QCon")
  (do-something-with (and param1 param2)))
```

last expression is the result

```
(function-name arg1 arg2)
```

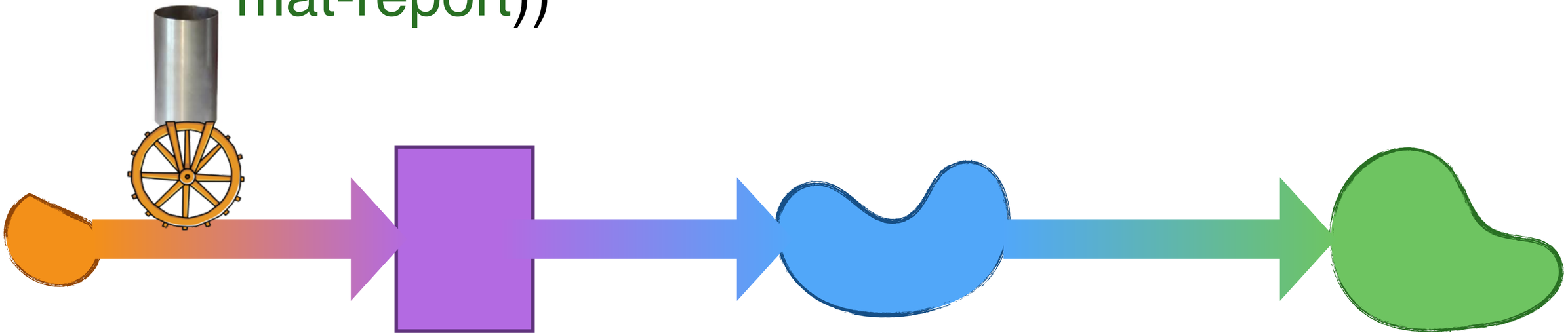
square braces
make a vector

```
(defn function-name [param1 param2]
  (println "Hello QCon")
  (do-something-with (and param1 param2)))
```

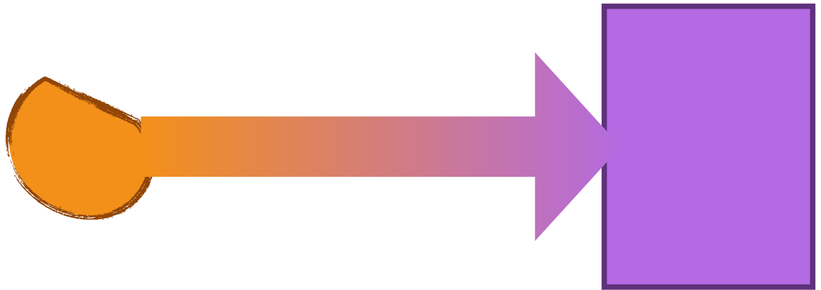
```
(defn format-report [report-data]
  ...)
```

```
(defn ad-performance-report [params]
  (-> (fetch-events params)
      (analyze-ad-performance params)
      format-report))
```

```
(defn ad-performance-report [params]
  (-> (fetch-events params)
      (analyze-ad-performance params)
      format-report))
```



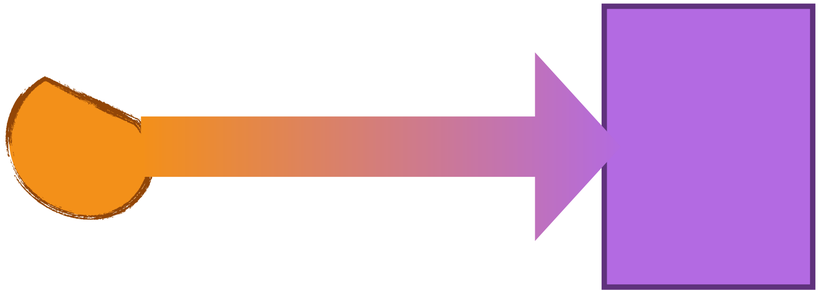
```
(defn fetch-events [params]
  ...)
```



curly braces make a map

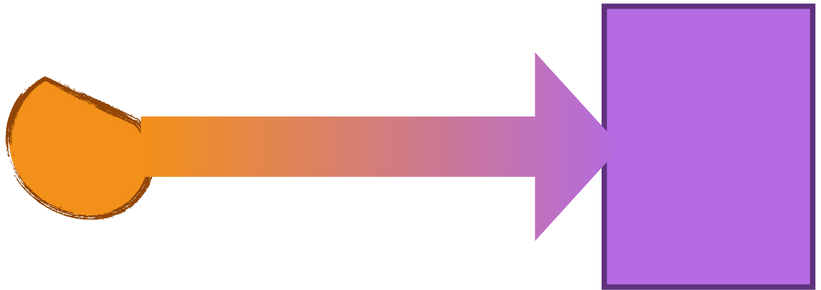
```
{:when 12:34:56 7/8/90
 :what "show"
 :who  "abc123"}
```

keyword



give a thing a name

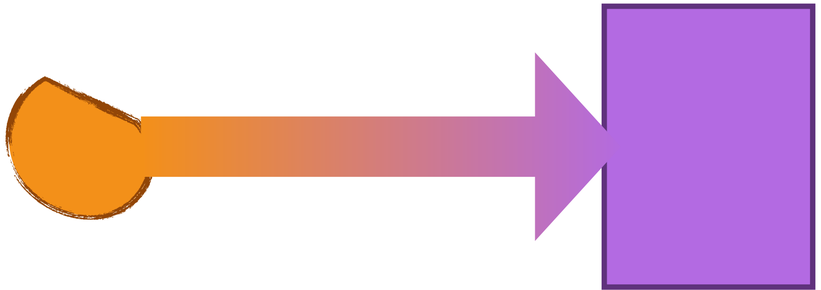
```
(def Event      {:when org.joda.time.DateTime
                 :what java.lang.String
                 :who  java.lang.String})
```



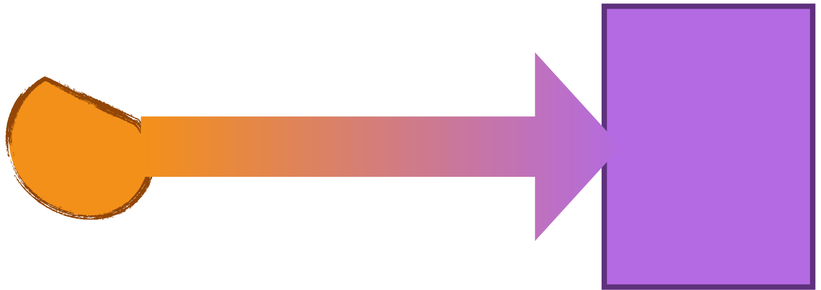
```
(:require [schema.core :as s])
```

dependency

```
(def Event      {:when DateTime
                 :what s/Str
                 :who  s/Str})
```

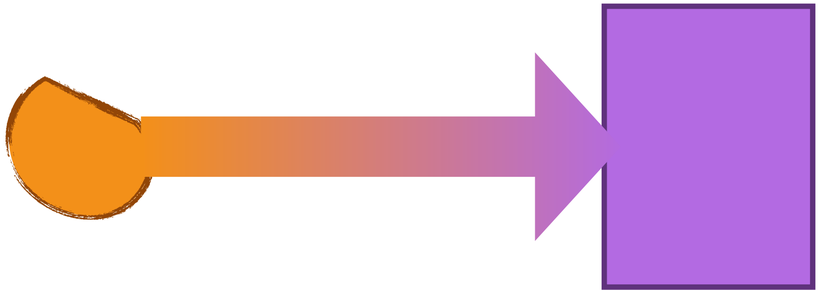


```
(def Incident s/Str)

(def Customer s/Str)

(def Event   {:when DateTime
              :what Incident
              :who  Customer})
```



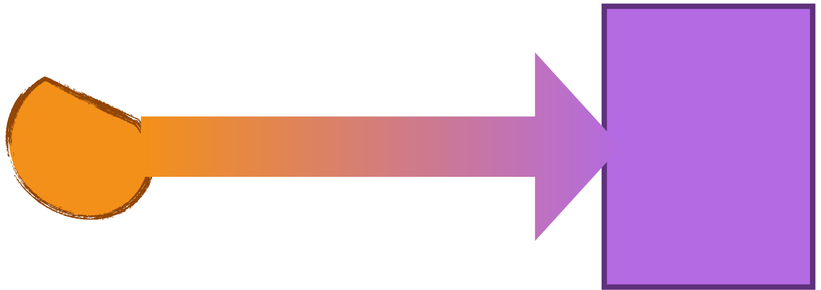
```
(def Event      {:when DateTime
                 :what Incident
                 :who  Customer})
```

[Event]



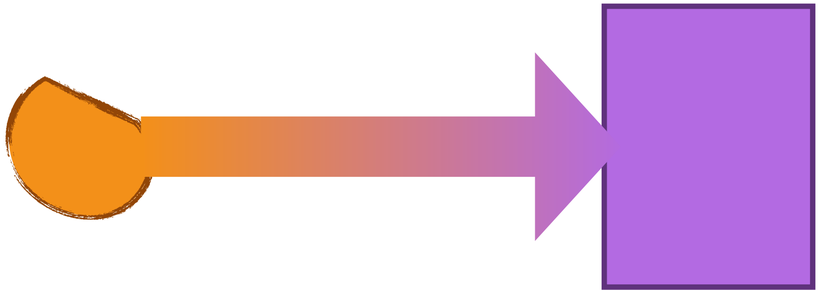
```
(defn fetch-events          [params]
  …)
```

[Event]



```
(:require [schema.core :as s])

(s/defn fetch-events          :-   [Event]
        [params]
        ...)
```

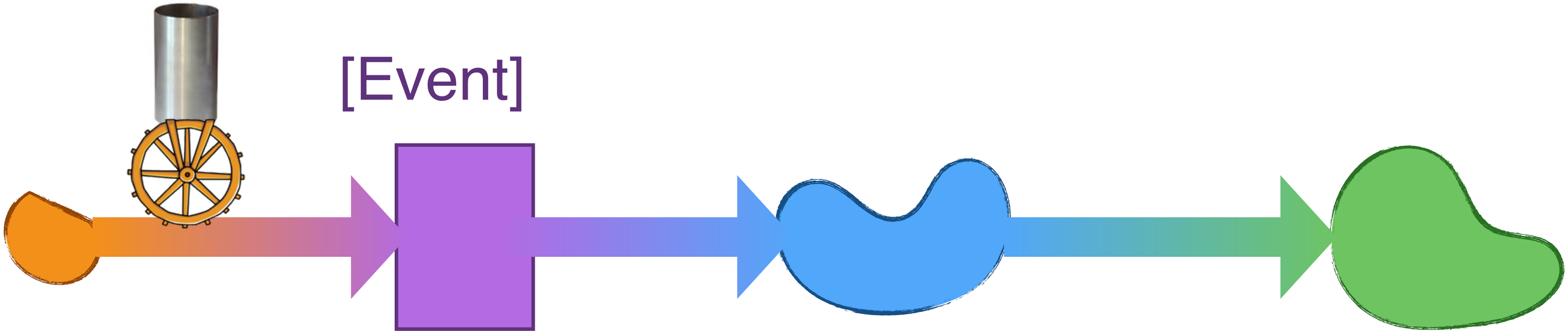
[Event]



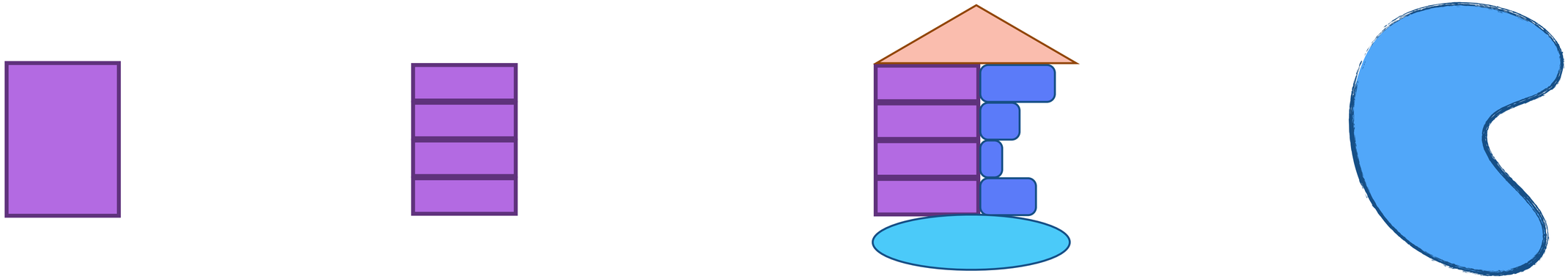
```
(deftest fetch-events-test
  ...
  (= (expected (fetch-events input))))
```

```
(use-fixtures schema.test/validate-schemas)

(deftest fetch-events-test
  ...
  (= (expected (fetch-events input))))
```

```
(defn ad-performance-report [params]
  (-> (fetch-events params)
      (analyze-ad-performance params)
      format-report))
```

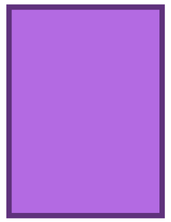


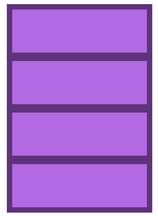
```
(defn analyze-ad-performance [events params]
  (-> events
      (group-up params)
      summarize
      add-total-row
      (add-headers params)))
```

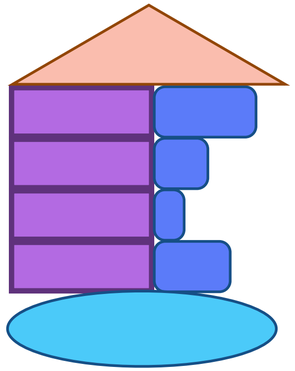


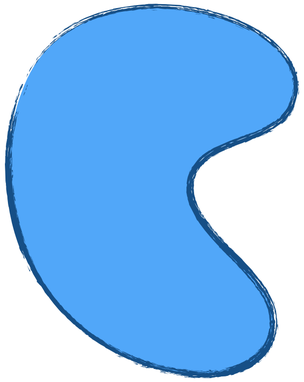
```
(defn analyze-ad-performance [events params]
  (-> events
      (group-up params)
      summarize
      add-total-row
      (add-headers params)))
```

[Event]



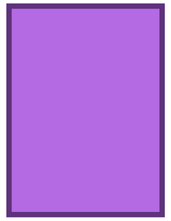




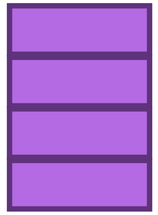


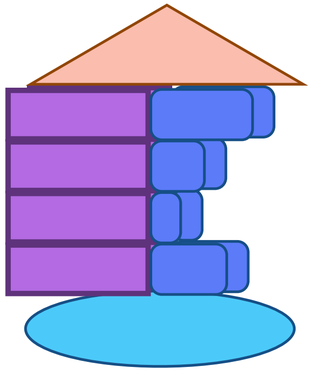
```
(defn analyze-ad-performance [events params]
  (-> events
      (group-up params)
      summarize
      add-total-row
      (add-headers params)))
```

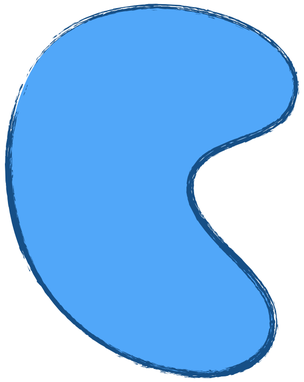
[Event]



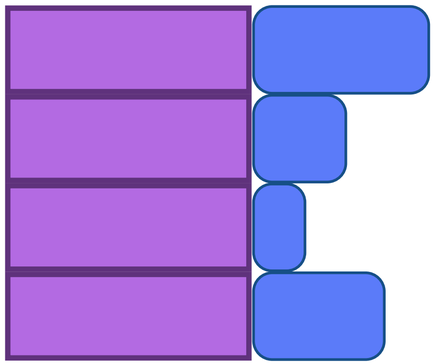
[[Event]]



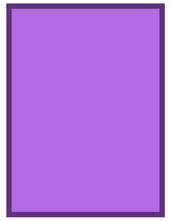




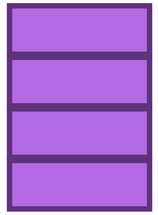
```
(defn analyze-ad-performance [events params]
  (-> events
      (group-up params)
      summarize
      add-total-row
      (add-headers params)))
```

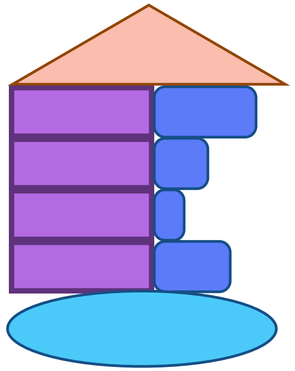


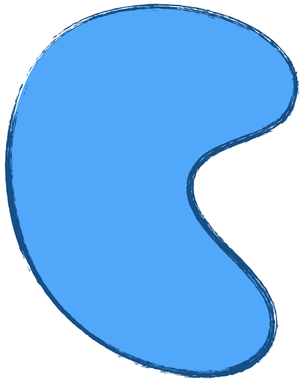
[Event]



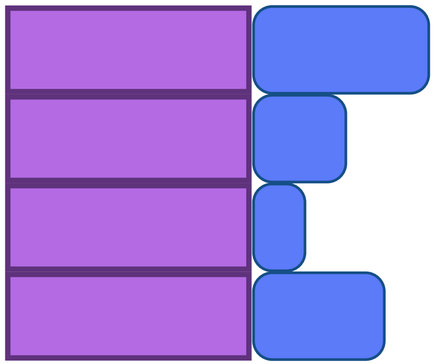
[[Event]]



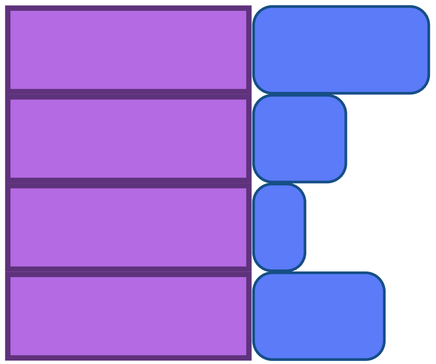




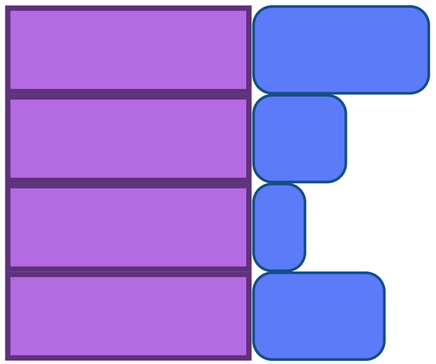
[[Event]]



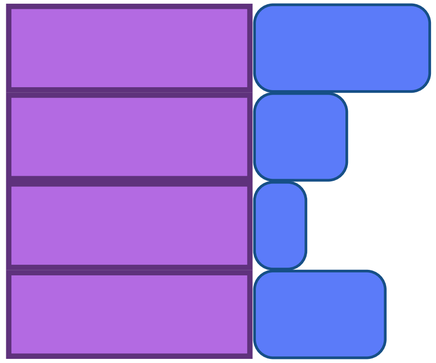
[[Event] Summation]



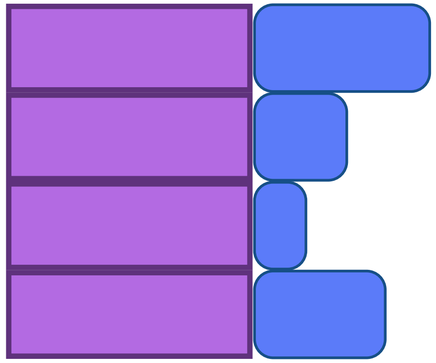
[(one [Event])
(one Summation)]



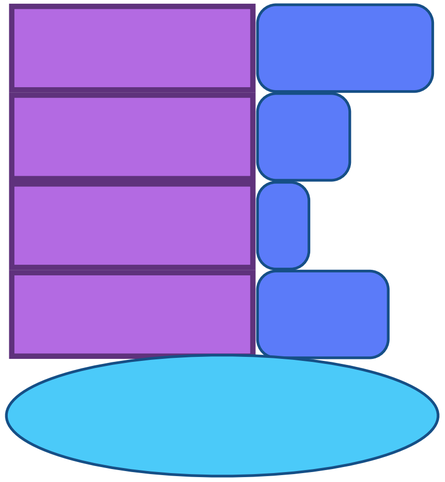
```
[(s/one [Event]   "event list")
 (s/one Summation "group sum")]
```



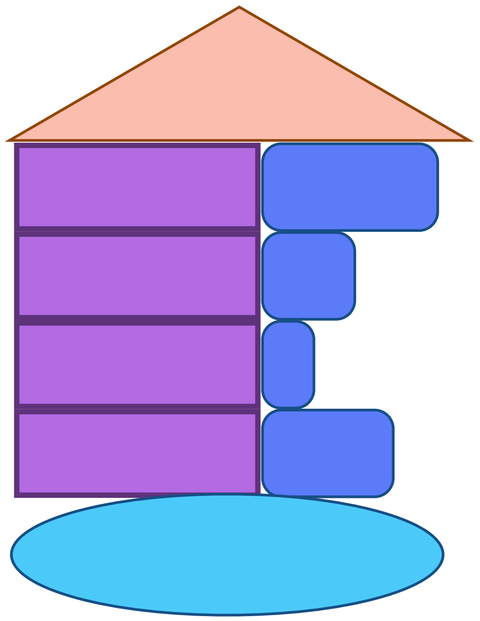
```
{:groups [[(s/one [Event]   "event list")
           (s/one Summation "group sum")]]}
```



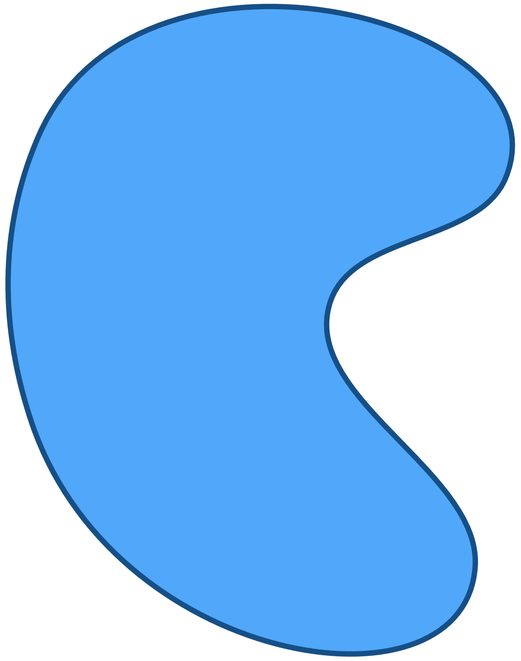
```
{:groups [[(s/one [Event]   "event list")
           (s/one Summation "group sum")]]
 :total Totals}
```



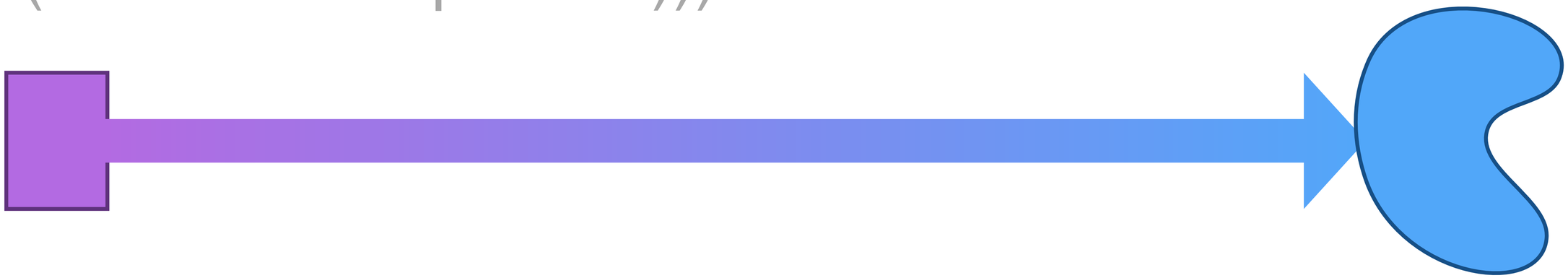
```
{:header Headers
 :groups [[(s/one [Event]   "event list")
           (s/one Summation "group sum")]]
 :total Totals})
```



```
(def ReportData
 {:header Headers
  :groups [[(s/one [Event]   "event list")
            (s/one Summation "group sum")]]
  :total Totals})
```



```
(s/defn analyze-ad-performance :- ReportData
  [events :- [Event]
   params :- Params]
  (-> events
      (group-up params)
      summarize
      add-total-row
      (add-headers params)))
```



```
(s/defn analyze-ad-performance :- ReportData
  ...)
```

```
(s/defn analyze-ad-performance
        :- (at-least ReportData)
  …)
```

```
(defn at-least [map-schema]
  (merge map-schema
         {s/Any s/Any}))
```

```
(s/defn analyze-ad-performance
         :- (at-least ReportData)
  …)
```

What do we know?

What do we know?

data shape

What do we know?

data shape

value boundaries

Headers

:title

- string
- not empty
- capitalized

```
(def Headers
 {:title
   (s/constrained
     s/Str
     (s/pred (complement empty?) "nonempty")
     (s/pred capitalized? "Title Caps"))
  …})
```

What do we know?

data shape

data value boundaries

relationships within values

What do we know?

data shape
data value boundaries
relationships within values

What could we know?

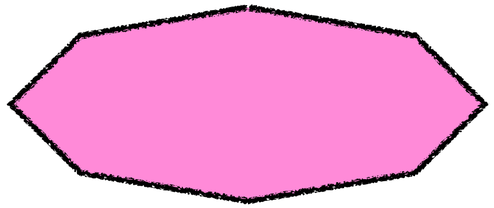
What do we know?

data shape
data value boundaries
relationships within values

What could we know?

produced types

Seq[Event]



```
(s/defn fetch-events :- [Event]
        [params]
  …)
```

```
(s/defn fetch-events :- [Event]
      [params]
  …)
```

what if?

```
(st/defn fetch-events :+ (st/LazySeq Event)
        [params]
  …)
```

check this *later*

```
(:require [schematron.core :as st])


(st/defn fetch-events :+ (st/LazySeq Event)
        [params]
  …)
```

```
(s/defn a-higher-order-function
      [predicate :- (s/=> Bool Event)
       …]
  …)
```

what if?

```
(st/defn a-higher-order-function
        [predicate :+ (st/=> Bool Event)
         ...]
  ...)
```

check this *later*

```
(s/defn a-higher-order-function :- Event
      [predicate :- (st/=> Bool Event)
       …]
  …)
```


```
(st/defn [A] a-higher-order-function :- A
         [predicate :+ (st/=> Bool A)
          …]
  …)
```

```
(a-higher-order-function Event is-happy …)
```

What do we know?

data shape
data value boundaries
relationships within values

What could we know?

produced types

relationships between types

What do we know?

data shape
data value boundaries
relationships within values

What could we know?

produced types

relationships between types

relationships between values

```
(defn group-up [events params]
 {:post [(as-lazy-as events %)]
 ...))
```

postcondition

data shape

data value boundaries

relationships within values

produced types

relationships between types

relationships between values

How do we know it?

```
(deftest analyze-ad-performance-test
  (testing "grouping of rows"
    (let [...
          result (analyze-ad-performance
                  events
                  {})
      (is (= expected (:groups result))))))
```

```
(use-fixtures schema.test/validate-schemas)

(deftest analyze-ad-performance-test
  (testing "grouping of rows"
    (let [...
          result (analyze-ad-performance
                  events
                  {})
      (is (= expected (:groups result))))))
```

```
    result (analyze-ad-performance
            events
            {})
  (is (= expected (:groups result)))))
```

Input does not match schema Params

Missing required key :title

Missing required key :start

Missing required key :end

```
    result (analyze-ad-performance
            events
            (sc/complete {} Params)
  (is (= expected (:groups result)))))
```

"Fill in everything else with something random until it meets this schema"

any string

```
{:title "YhKEzII",
 :start "-217863493-11-21T00:54:39.872Z"],
 :end   "-256417656-09-30T01:08:11.904Z"]}
```

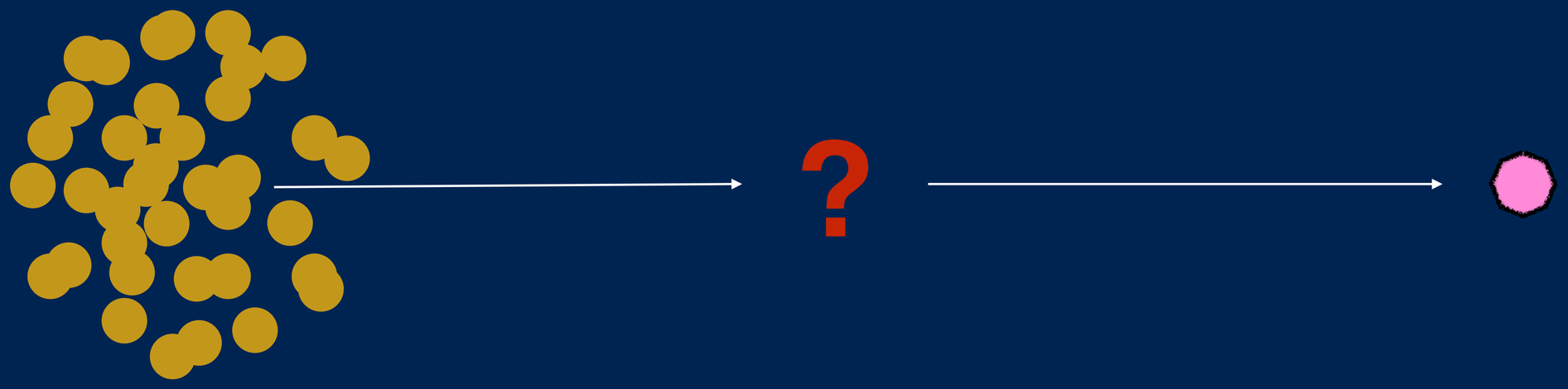
any date
before now

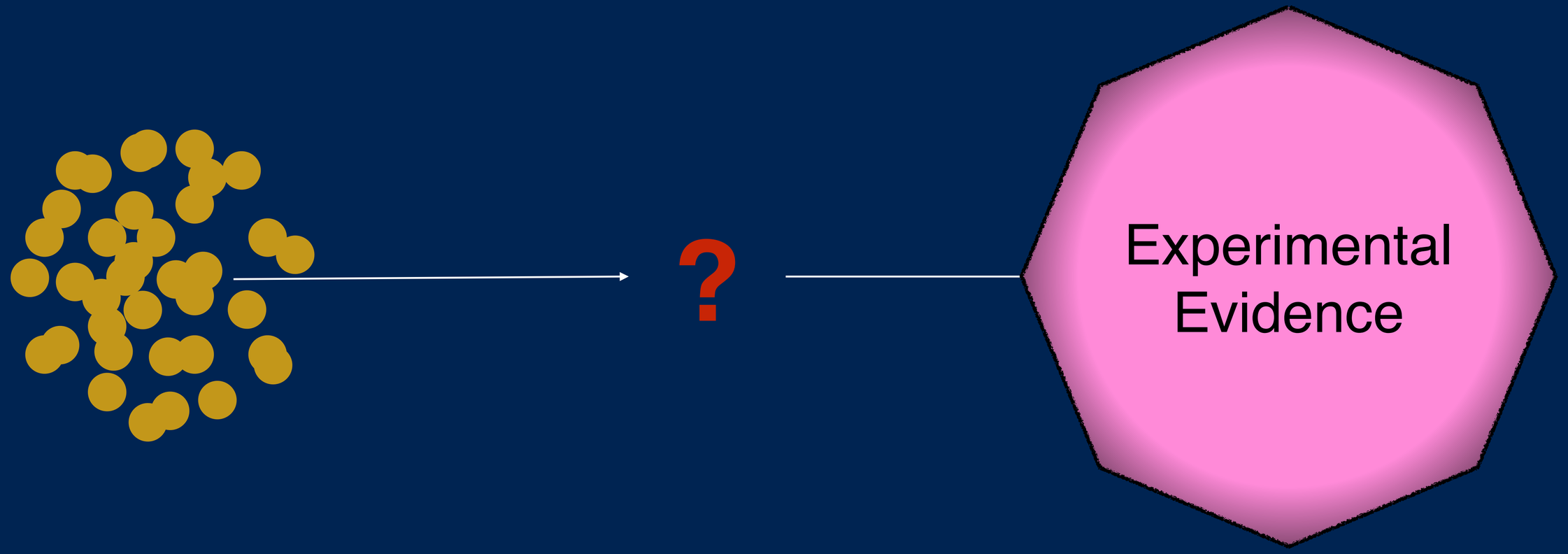
any date before
the start

```
(use-fixtures schema.test/validate-schemas)

(deftest analyze-ad-performance-test
  (testing "grouping of rows"
    (let [...
          result (analyze-ad-performance
                  [events]
                  (sample-one param-gen))
      (is (= expected (:groups result))))))
```

1 test is an anecdote

generative tests are evidence




?
Experimental
Evidence

```
(use-fixtures schema.test/validate-schemas)


(defspec analyze-ad-performance-spec 100
  (for-all [events events-gen
            params param-gen]
    (analyze-ad-performance events params))))
```

```
(s/defn analyze-ad-performance :- ReportData
  ...)
```

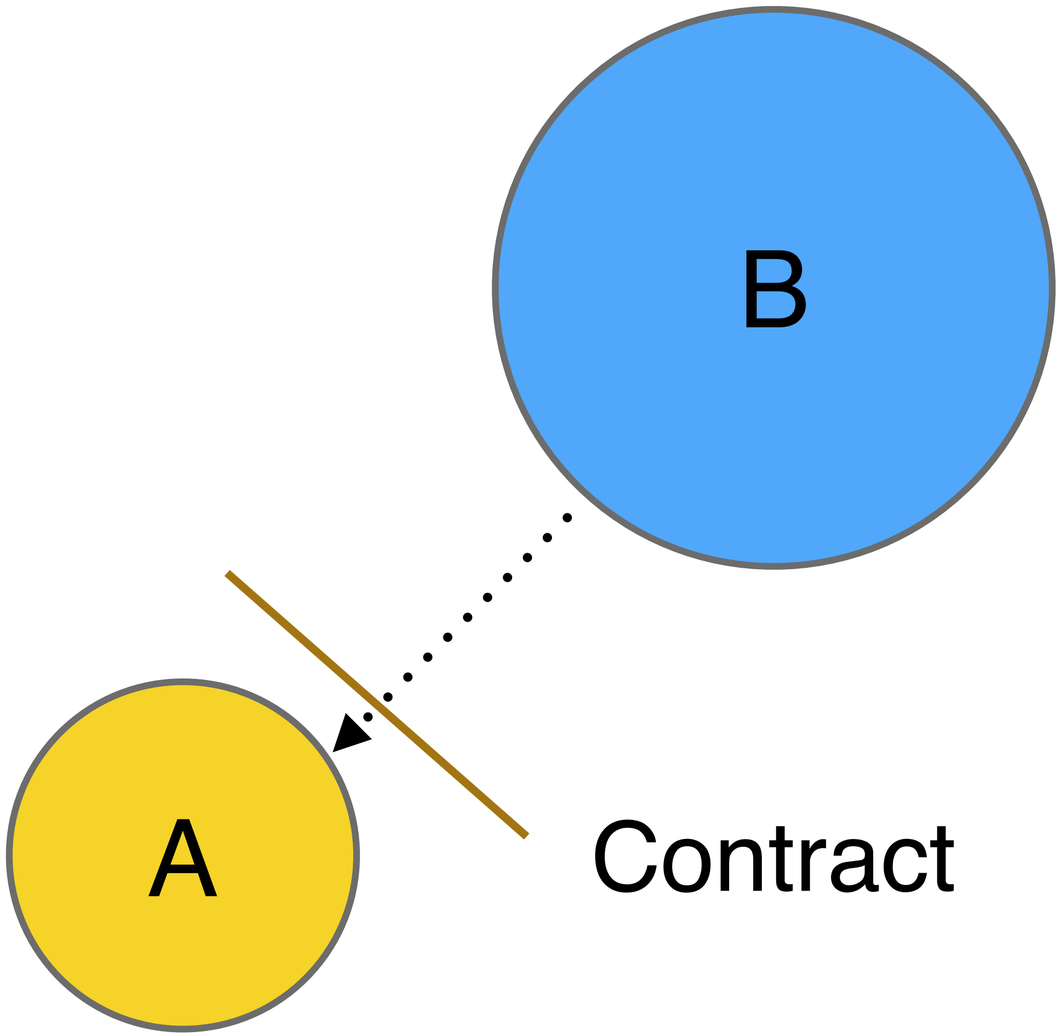
What do we know?

Schemas

How do we know it?

Generative Tests


T.GERRARD & CO
SURREY


B
A
Contract

INTRODUCTION
Introduction

TOPICS
Authentication
Errors
Expanding Objects
Idempotent Requests
Metadata
Pagination
Request IDs
Versioning

CORE RESOURCES
Balance
Charges
Customers
The customer object
Create a customer
Retrieve a customer
Update a customer
Delete a customer

The customer object

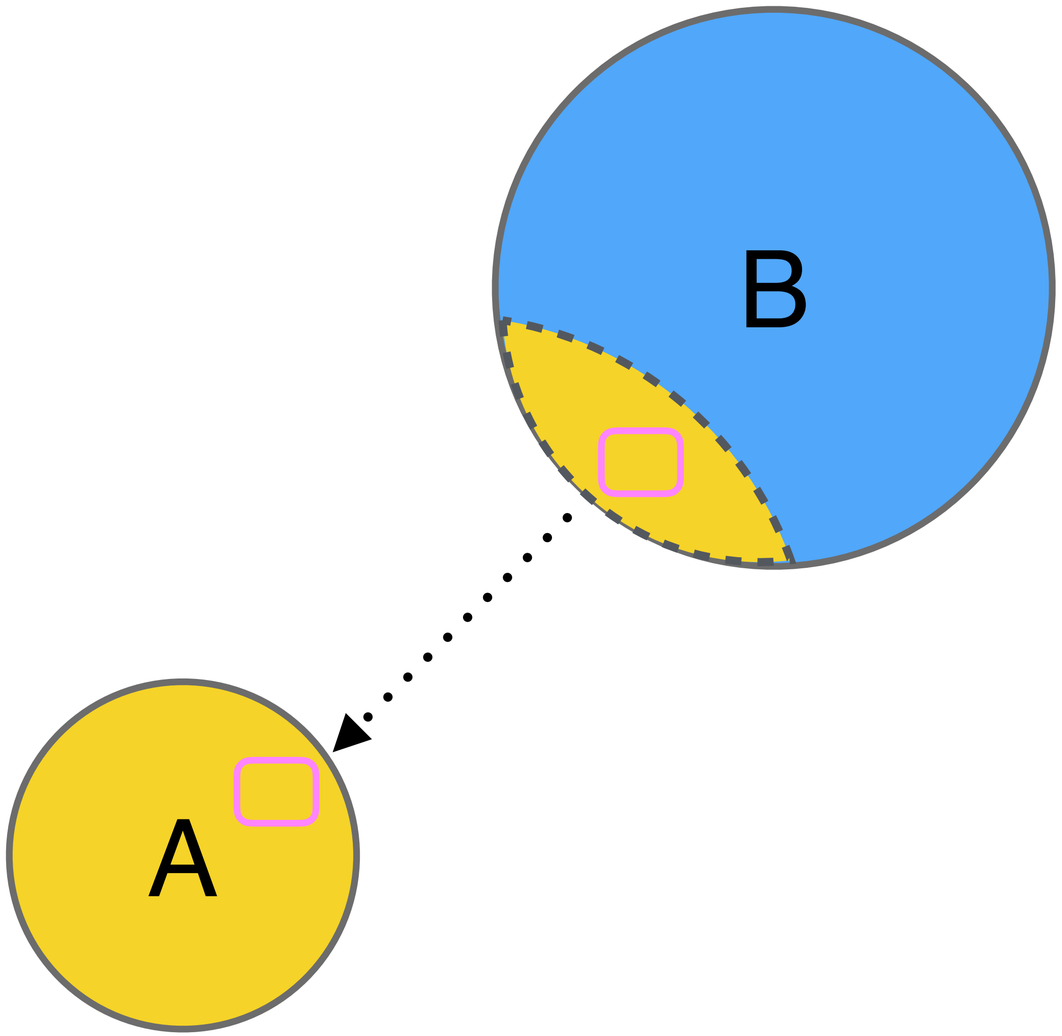
ATTRIBUTES

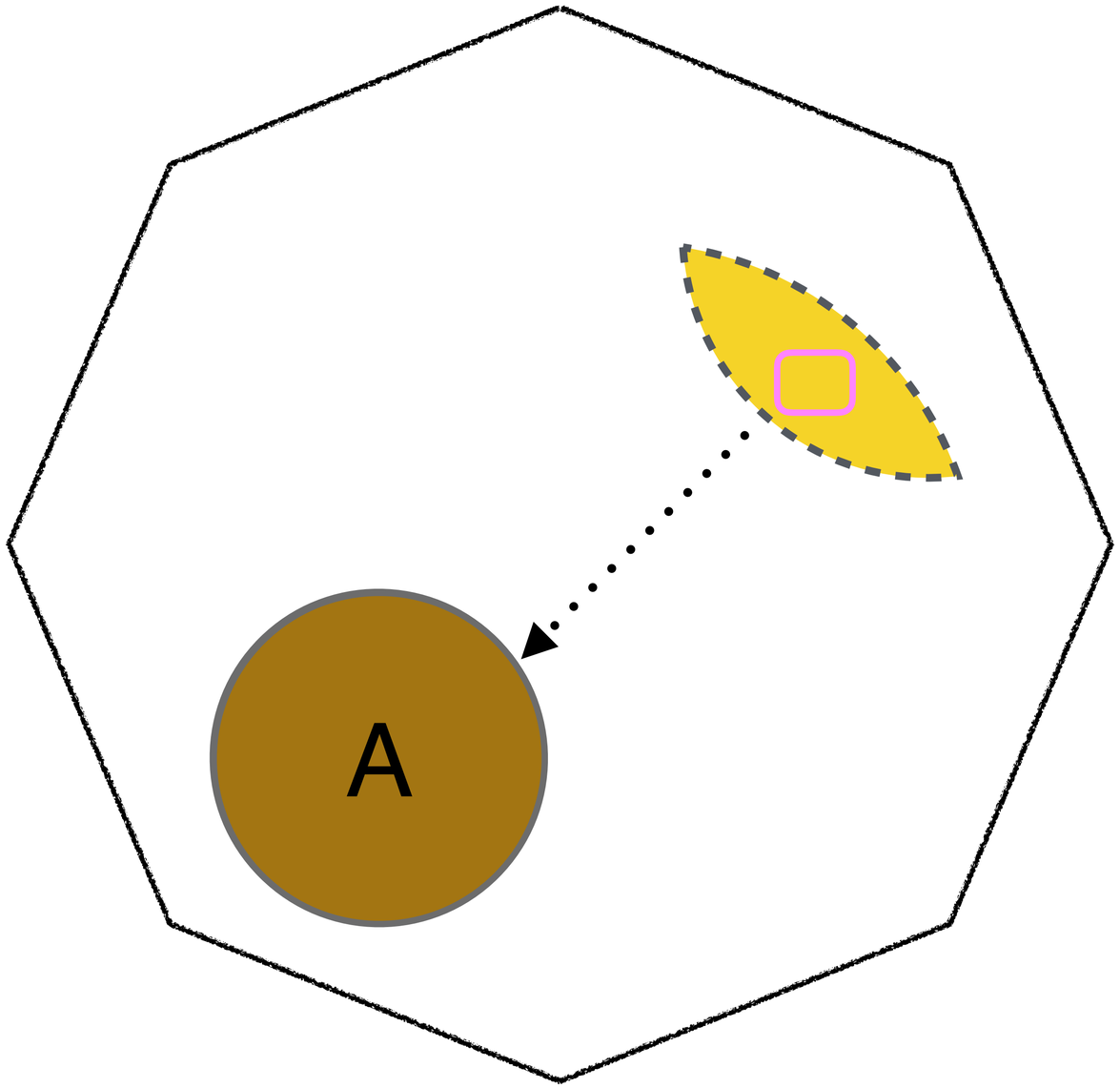
id string	—
object string , value is "customer"	—
account_balance integer	Current balance, if any, being stored on the customer's account. If negative, the customer has credit to apply to the next invoice. If positive, the customer has an amount owed that will be added to the next invoice. The balance does not refer to any unpaid invoices; it solely takes into account amounts that have yet to be successfully applied to any invoice. This balance is only taken into account for recurring charges.
created timestamp	—
currency string	The currency the customer can be charged in for recurring billing purposes (subscriptions, invoices, invoice items).

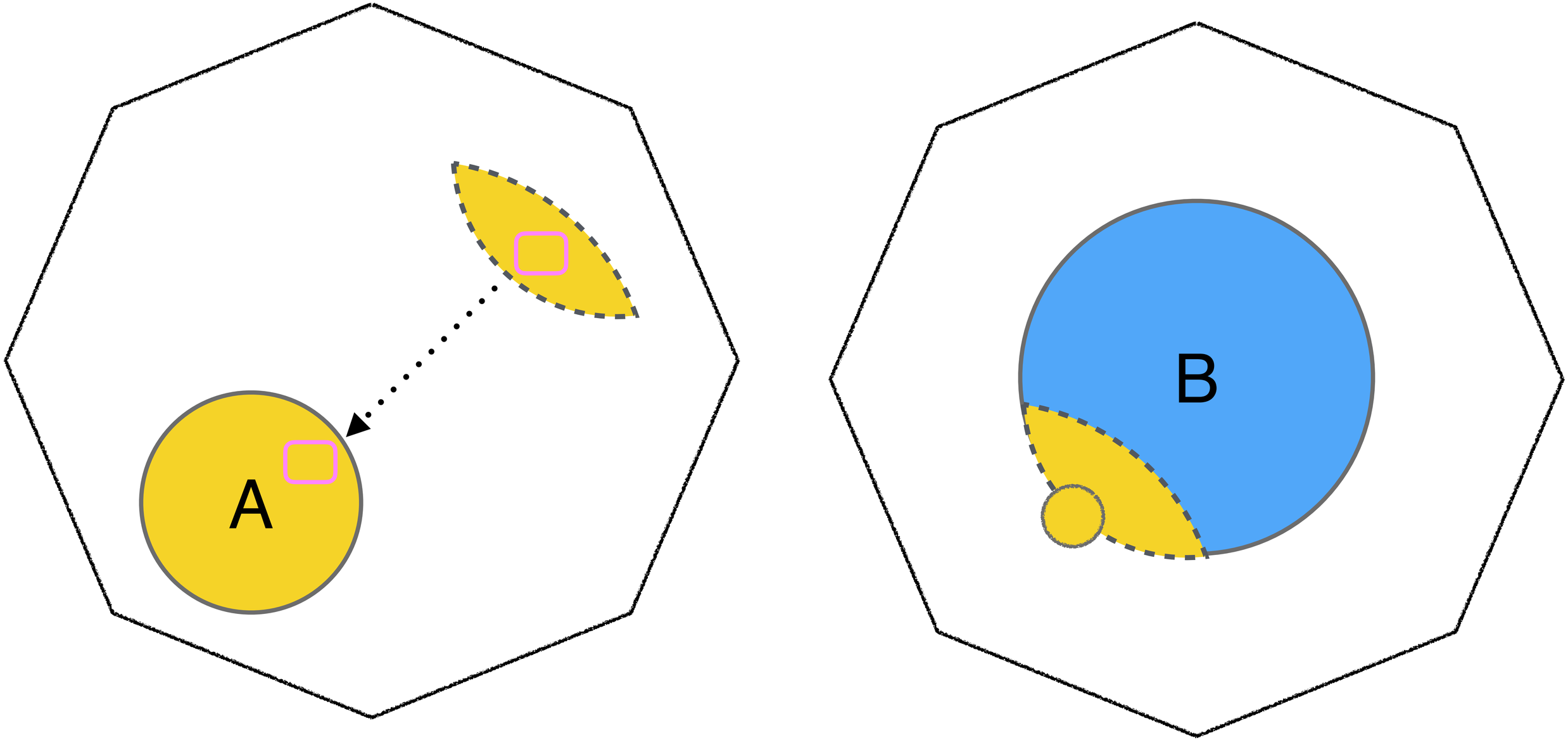
curl Ruby Python PHP Java Node Go

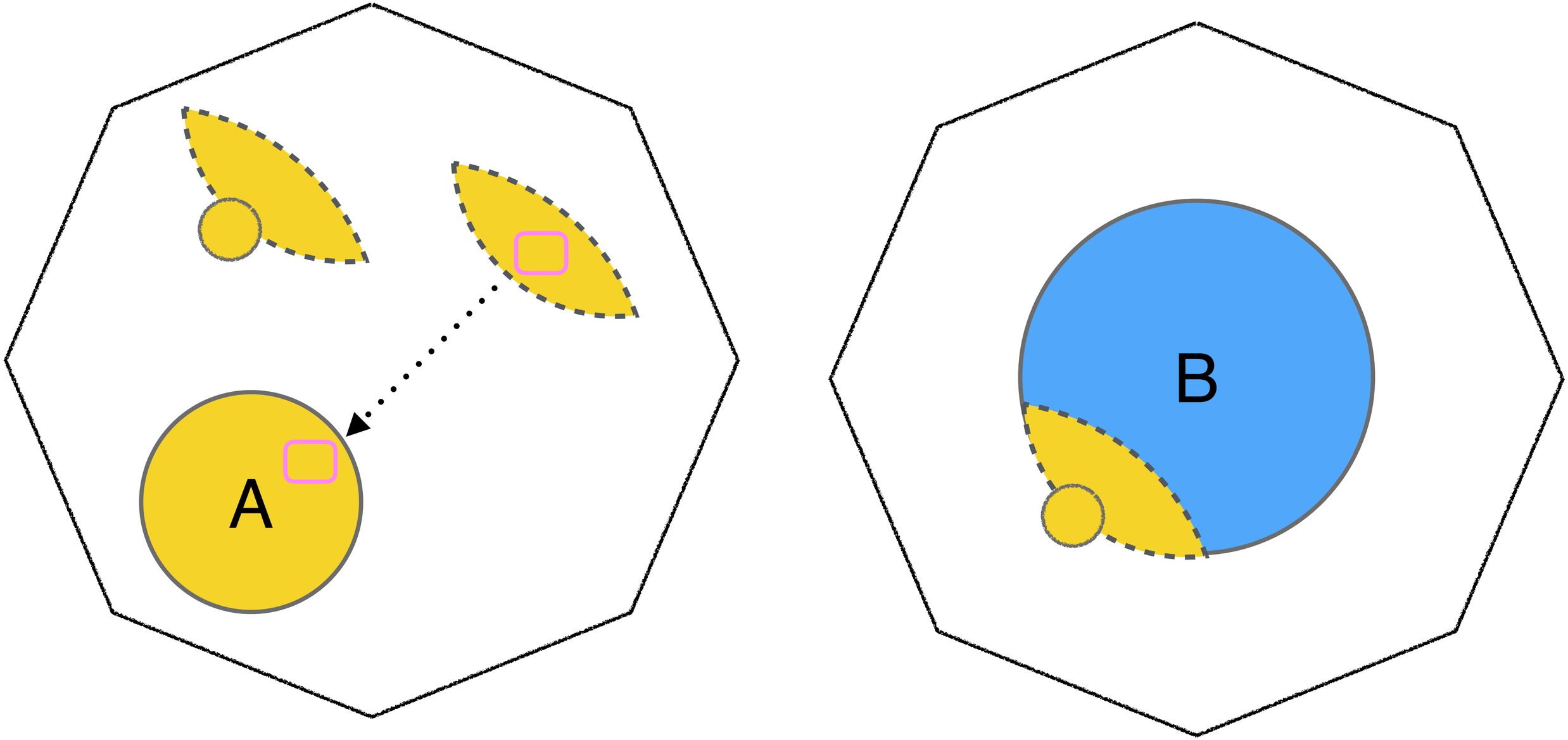
Example Response

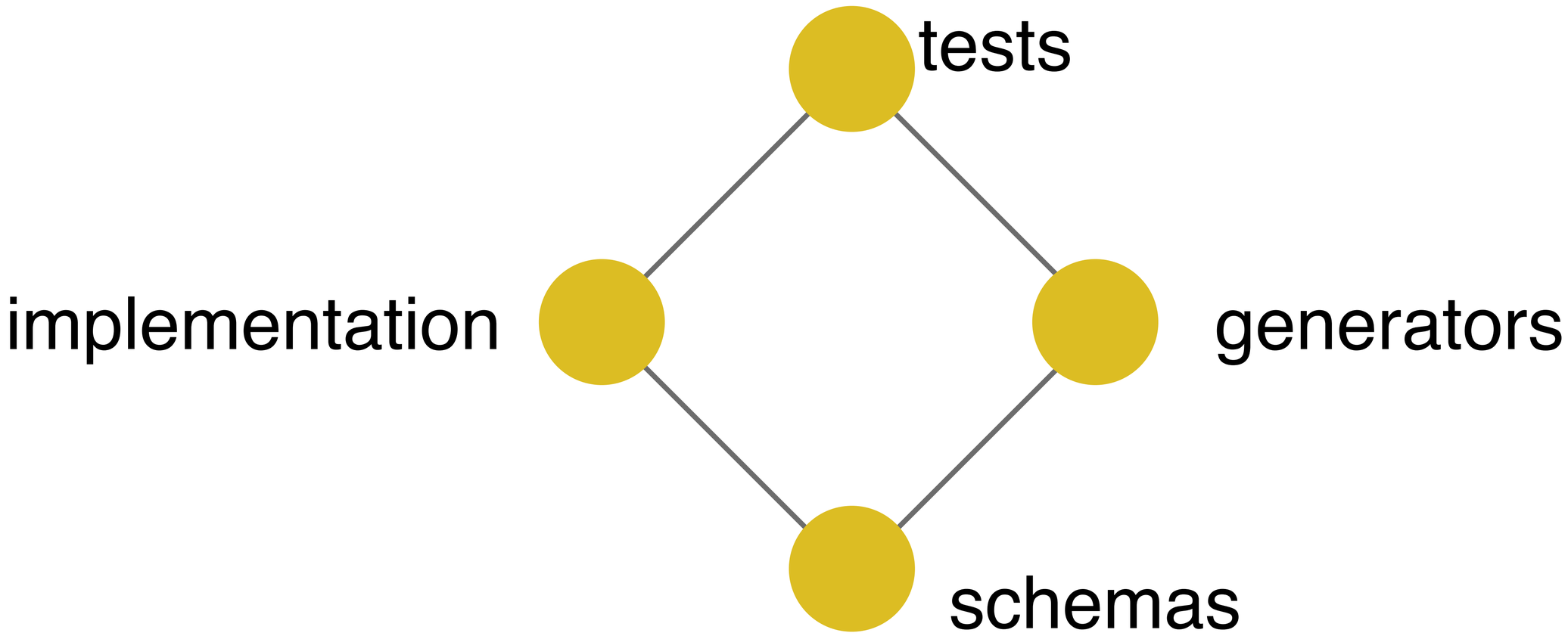

```
com.stripe.model.Customer JSON: {
  "id": "cus_7MxHmoJ3VOQhqO",
  "object": "customer",
  "account_balance": 0,
  "created": 1447781746,
  "currency": "usd",
  "default_source": "card_178DMr2eZvKYlo2CEolHjbJX",
  "delinquent": false,
  "description": null,
  "discount": null,
  "email": "virtumedix+llljd@gmail.com",
  "livemode": false,
  "metadata": {
  },
  "shipping": null,
  "sources": {
    "object": "list",
    "data": [
      {
        "id": "card_178DMr2eZvKYlo2CEolHjbJX",
        "object": "card",
        "address_city": null,
        "address_country": null,
        "address_line1": null,
        "address_line1_check": null,
        "address_line2": null,
        "address_state": null,
        "address_zip": null,
        "address_zip_check": null,
        "brand": "Visa",
```

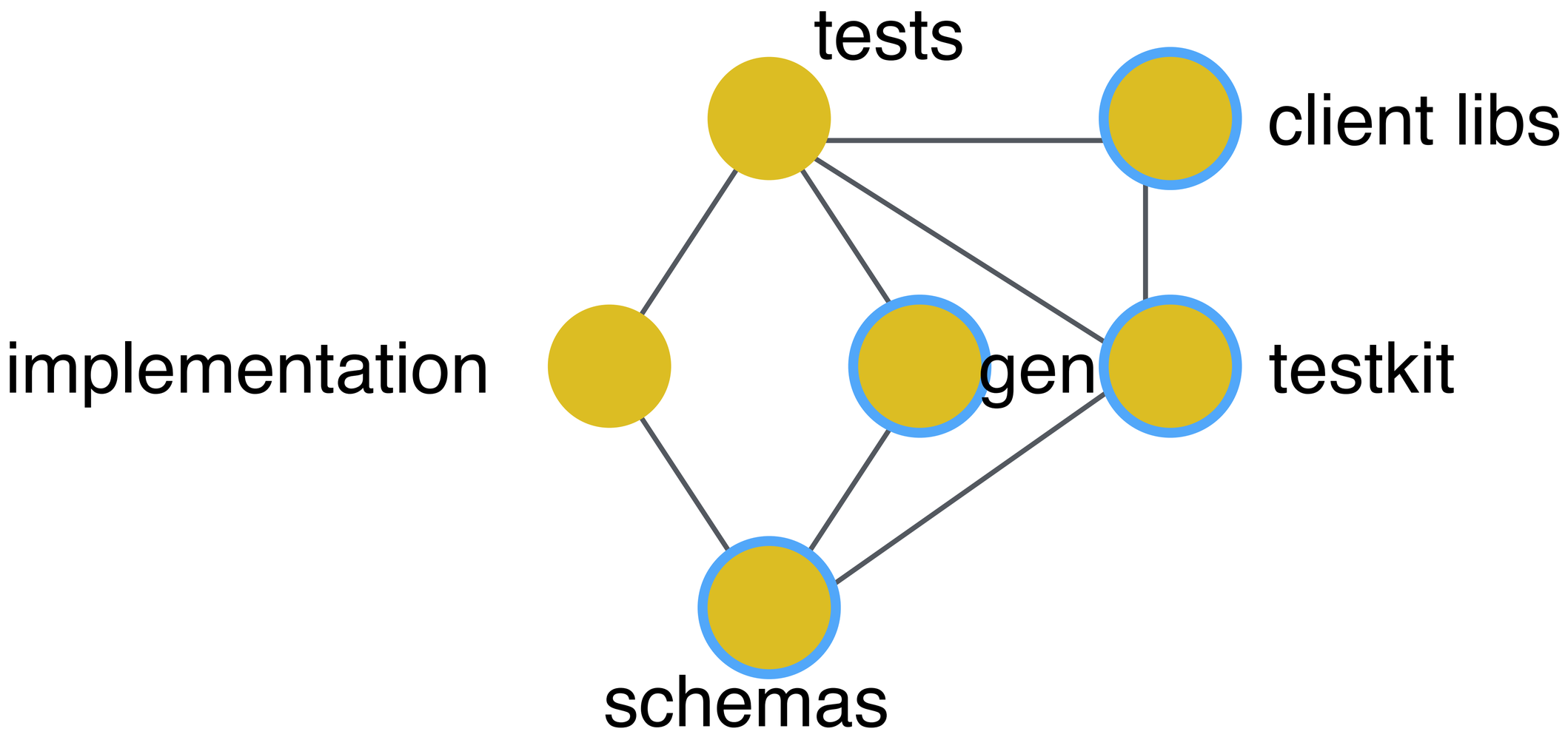

B
A

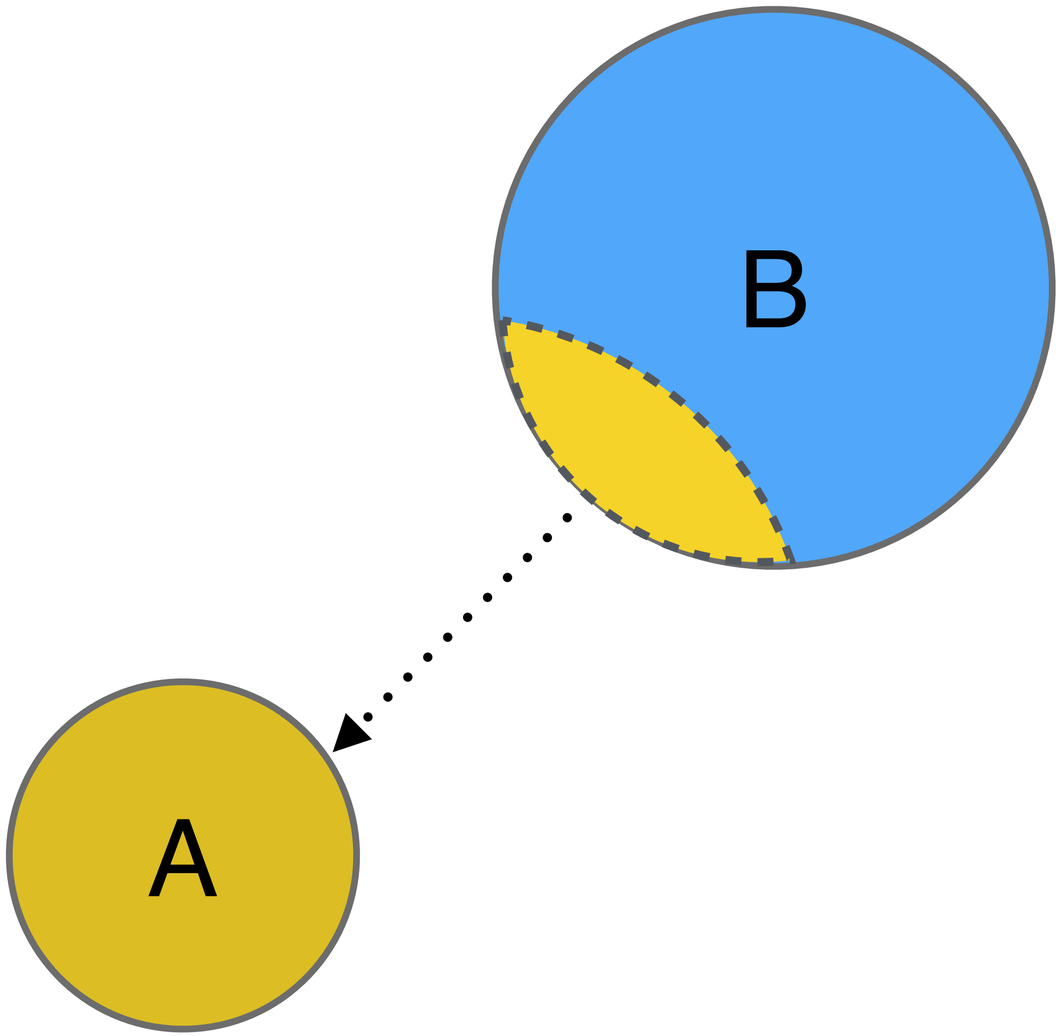

A


A
B


A
B


tests
implementation
generators
schemas


tests
client libs
implementation
gen
testkit
schemas


B
A

Clojure

prismatic/schema

test.check

Science!

… your language …

types and contracts

generative tests

Science!

… your services …

client libraries

testkits

Science!

Informal
Reasoning

Formal
Proofs

Experimental
Evidence

examples

https://github.com/jessitron/contracts-as-types-examples

https://github.com/jessitron/schematron

resources

https://github.com/Prismatic/schema

http://hintjens.com/blog:85 The End of Software Versions

http://david-mcneil.com/post/114783282473/extending-prismatic-schema-to-higher-order

Static typing and productivity: Stefik & Hanenberg 2014

http://dl.acm.org/citation.cfm?id=2661156

@jessitron

blog.jessitron.com

stripe