Onyx

Distributed Workflows for Dynamic Systems

Michael Drogalis

Me!

- Independent consultant
- Functional programming, Clojure, distributed systems
- @MichaelDrogalis

What is Onyx?

- A new kind of distributed computation system
- Provides an information model for the developer
- Inverts calling control of traditional frameworks
- Competes in stream, batch, and ingestion space
- Enabled by hardware advances in the last decade

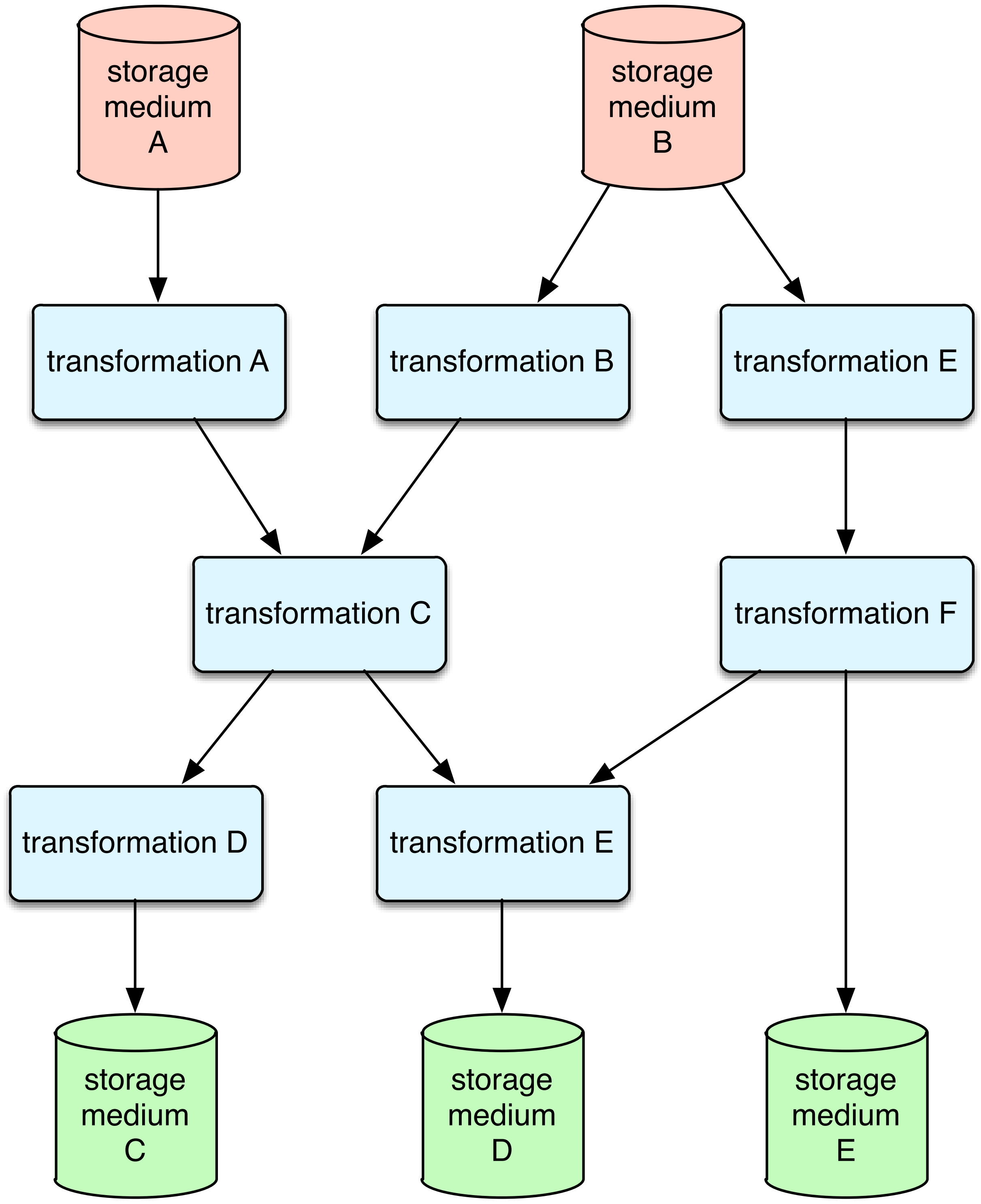
Onyx Goals

- Take apart the monolith - data and fns
- Target Clojure & the JVM
- Batch and stream hybrid, transparent code reuse
- Transactional execution semantics
- Powerful extensibility API
- An API dedicated to side-effects & state

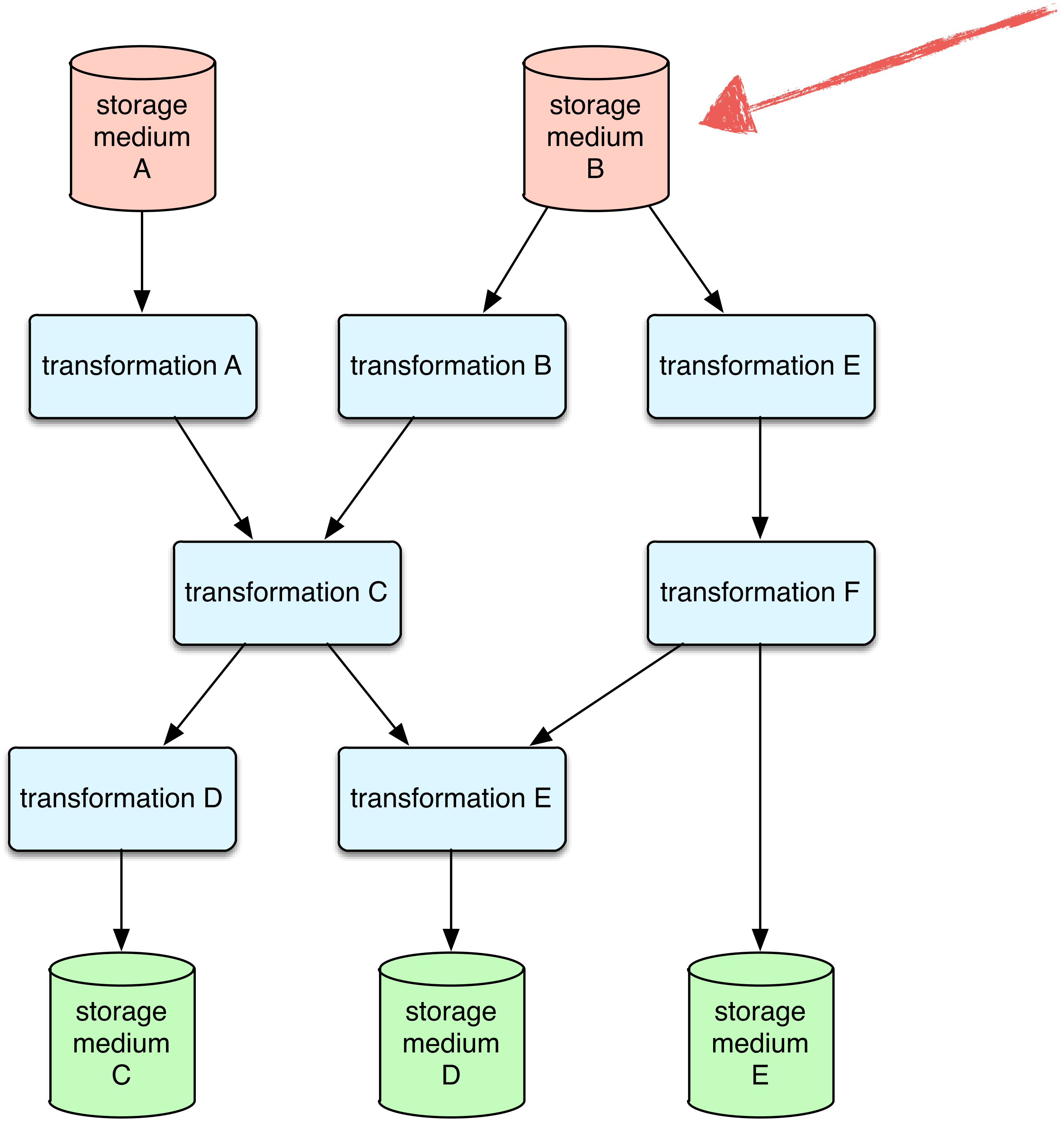
Overview

- The problem space
- Information model and APIs
- Architecture
- Development experience

The Problem

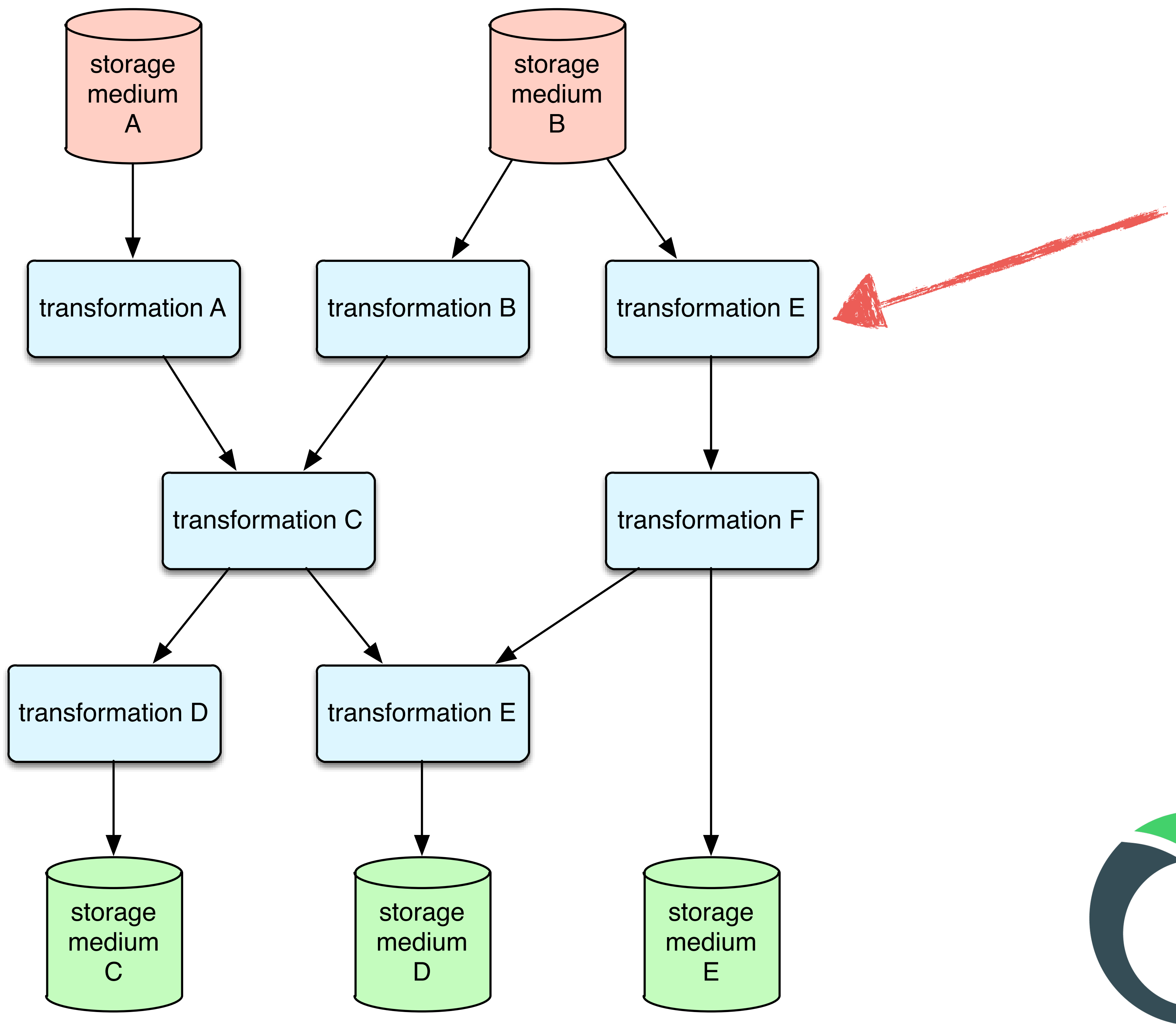

storage medium A
storage medium B
transformation A
transformation B
transformation E
transformation C
transformation F
transformation D
transformation E
storage medium C
storage medium D
storage medium E

The Problem

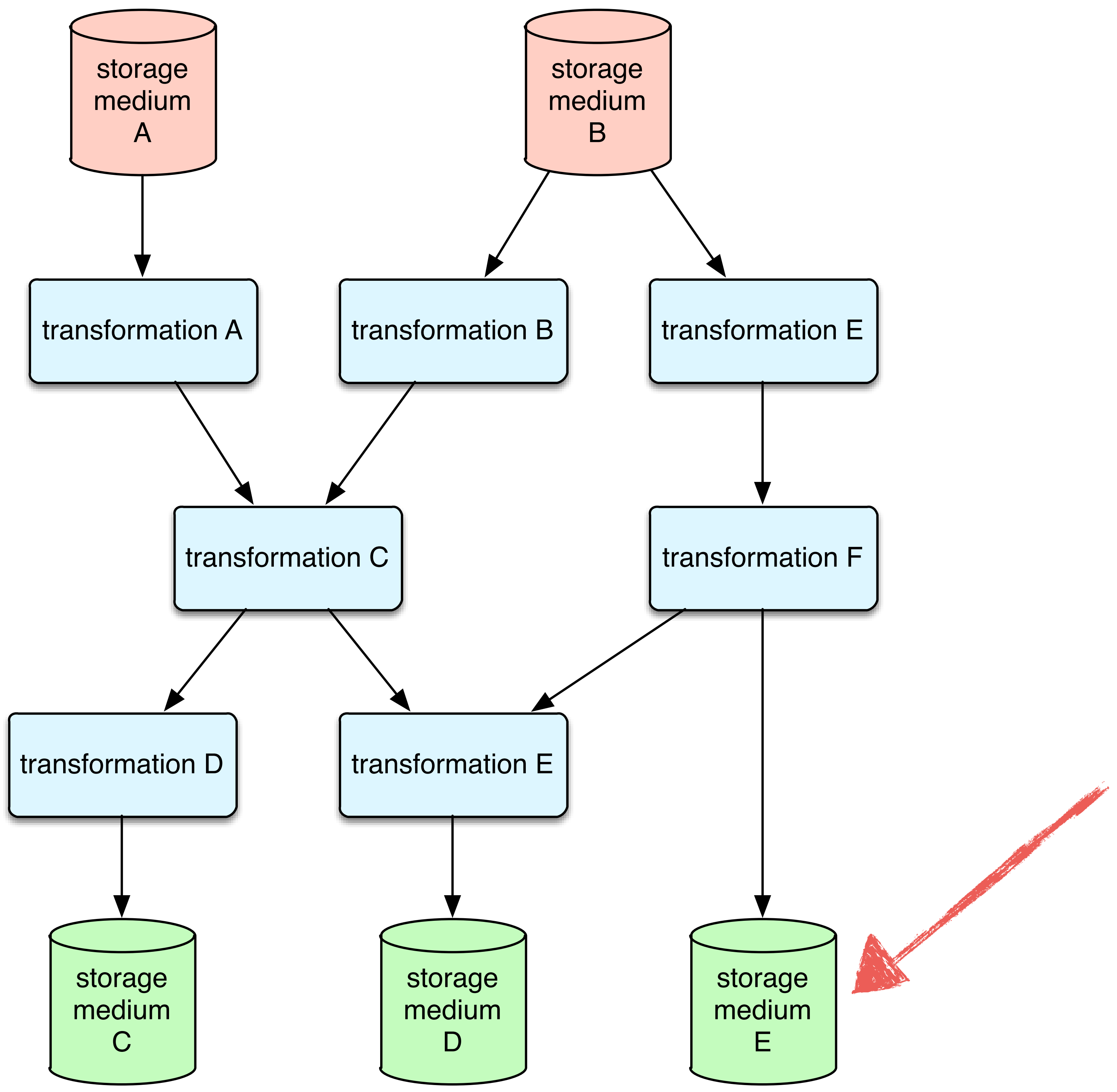


storage medium A
storage medium B
transformation A
transformation B
transformation E
transformation C
transformation F
transformation D
transformation E
storage medium C
storage medium D
storage medium E

The Problem


storage medium A
storage medium B
transformation A
transformation B
transformation E
transformation C
transformation F
transformation D
transformation E
storage medium C
storage medium D
storage medium E

The Problem


storage medium A
storage medium B
transformation A
transformation B
transformation E
transformation C
transformation F
transformation D
transformation E
storage medium C
storage medium D
storage medium E

The Problem

- Inputs, transformations, and outputs not static
- Determined at runtime - usually in another ecosystem!
- Specification of distributed execution needs to be…

The Problem

- Inputs, transformations, and outputs not static
- Determined at runtime - usually in another ecosystem!
- Specification of distributed execution needs to be…
  - language agnostic
  - location agnostic
  - temporally agnostic
  - tolerant to machine generation

Cascading: A Solution?

```
Pipe stopPipe = new Pipe("stop");
Pipe tokenPipe = new HashJoin(docPipe, token, stopPipe, stop, new LeftJoin());

tokenPipe = new Each(tokenPipe, stop, new RegexFilter("^$"));
tokenPipe = new Retain(tokenPipe, fieldSelector);
```

Cascading: A Solution?

```
Pipe stopPipe = new Pipe("stop");
Pipe tokenPipe = new HashJoin(docPipe, token, stopPipe, stop, new LeftJoin());

tokenPipe = new Each(tokenPipe, stop, new RegexFilter("^$"));
tokenPipe = new Retain(tokenPipe, fieldSelector);
```

~~language agnostic~~

~~location agnostic~~

~~temporally agnostic~~

machine tolerant

Cascading: A Solution?

mechanism

```
Pipe stopPipe = new Pipe("stop");
Pipe tokenPipe = new HashJoin(docPipe, token, stopPipe, stop, new LeftJoin());

tokenPipe = new Each(tokenPipe, stop, new RegexFilter("^$"));
tokenPipe = new Retain(tokenPipe, fieldSelector);
```

~~language agnostic~~

~~location agnostic~~

~~temporally agnostic~~

machine tolerant

Cascading: A Solution?

```
Pipe stopPipe = new Pipe("stop");
Pipe tokenPipe = new HashJoin(docPipe, token, stopPipe, stop, new LeftJoin());

tokenPipe = new Each(tokenPipe, stop, new RegexFilter("^$"));
tokenPipe = new Retain(tokenPipe, fieldSelector);
```

structure

~~language agnostic~~

~~location agnostic~~

~~temporally agnostic~~

machine tolerant

Cascading: A Solution?

```
Pipe stopPipe = new Pipe("stop");
Pipe tokenPipe = new HashJoin(docPipe, token, stopPipe, stop, new LeftJoin());

tokenPipe = new Each(tokenPipe, stop, new RegexFilter("^$"));
tokenPipe = new Retain(tokenPipe, fieldSelector);
```

configuration

~~language agnostic~~

~~location agnostic~~

~~temporally agnostic~~

machine tolerant

Cascading: A Solution?

```
Pipe stopPipe = new Pipe("stop");
Pipe tokenPipe = new HashJoin(docPipe, token, stopPipe, stop, new LeftJoin());

tokenPipe = new Each(tokenPipe, stop, new RegexFilter("^$"));
tokenPipe = new Retain(tokenPipe, fieldSelector);
```

forced concretion

~~language agnostic~~

~~location agnostic~~

~~temporally agnostic~~

machine tolerant

Cascading: A Solution?

```
Pipe stopPipe = new Pipe("stop");
Pipe tokenPipe = new HashJoin(docPipe, token, stopPipe, stop, new LeftJoin());

tokenPipe = new Each(tokenPipe, stop, new RegexFilter("^$"));
tokenPipe = new Retain(tokenPipe, fieldSelector);
```

black box

~~language agnostic~~

~~location agnostic~~

~~temporally agnostic~~

machine tolerant

Storm: A Solution?

```
(topology
 {"1" (spout-spec sentence-spout)
  "2" (spout-spec other-spout :p 2)}
 {"3" (bolt-spec {"1" :shuffle "2" :shuffle}
                 split-sentence :p 5)
  "4" (bolt-spec {"3" ["word"]}
                 word-count :p 6)})
```

Storm: A Solution?

```
(topology
 {"1" (spout-spec sentence-spout)
  "2" (spout-spec other-spout :p 2)}
 {"3" (bolt-spec {"1" :shuffle "2" :shuffle}
                 split-sentence :p 5)
  "4" (bolt-spec {"3" ["word"]}
                 word-count :p 6)})
```

~~language agnostic~~

location agnostic

temporally agnostic

~~machine tolerant~~

Cascalog: A Solution?

```
(?- (stdout)
    (<- [?word ?count]
  (sentence :> ?line)
  (tokenize :< ?line :> ?word)
  (count :> ?count)))
```

Cascalog: A Solution?

```
(?- (stdout)
    (<- [?word ?count]
  (sentence :> ?line)
  (tokenize :< ?line :> ?word)
  (count :> ?count)))
```

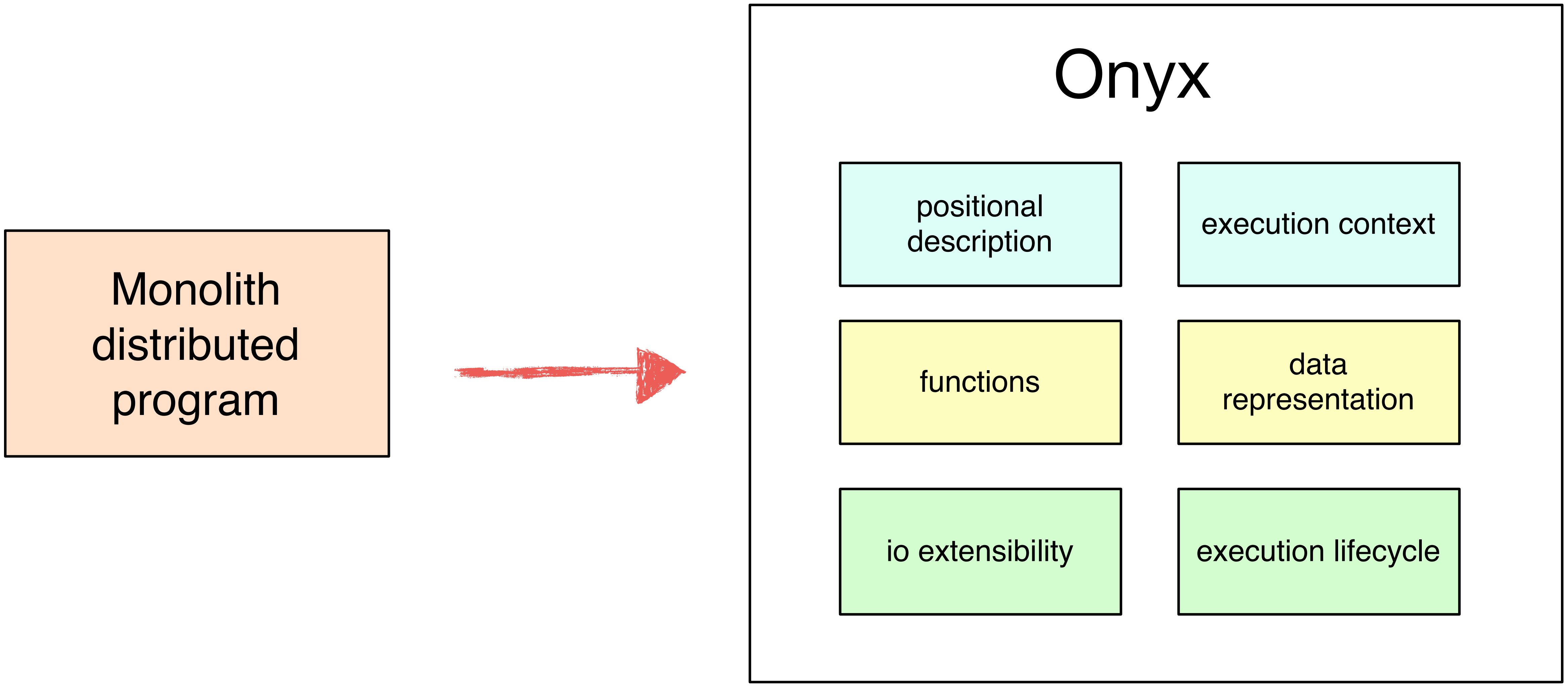
~~language agnostic~~

~~location agnostic~~

~~temporally agnostic~~

~~machine tolerant~~

Decomposing the Monolith



Guiding Example

name	**age**
Mike	23
John	19
Kristen	24

Guiding Example

name	age
Mike	23
John	19
Kristen	24

name	age
MIKE	24
JOHN	20
KRISTEN	25

Data Representation

- Declare names of values from fn -> fn
- Typical representation is tuple and fields
- Ordered, explicit sequence of values

output fields

```
(defbolt transform-person ["name" "age"] [tuple collector]
  (emit-bolt! collector ["MIKE" 24])
  (ack! collector tuple))
```

output tuple

Data Representation

- **Segment** - just a Clojure map
- No notion of ordering, no explicit declaration
- Intermediaries don't have to care
- Forces values to be named

```
{:name "Mike"
 :age 24}
```

Functions

- Plain Clojure functions
- Input: a single segment (additional params allowed)
- Output: one segment, or a seq of segments
- No macros, calling context, stateful parameters

```
(defn transform-person [segment]
  (-> segment
      (update-in [:name] clojure.string/upper-case)
      (update-in [:age] inc)))
```

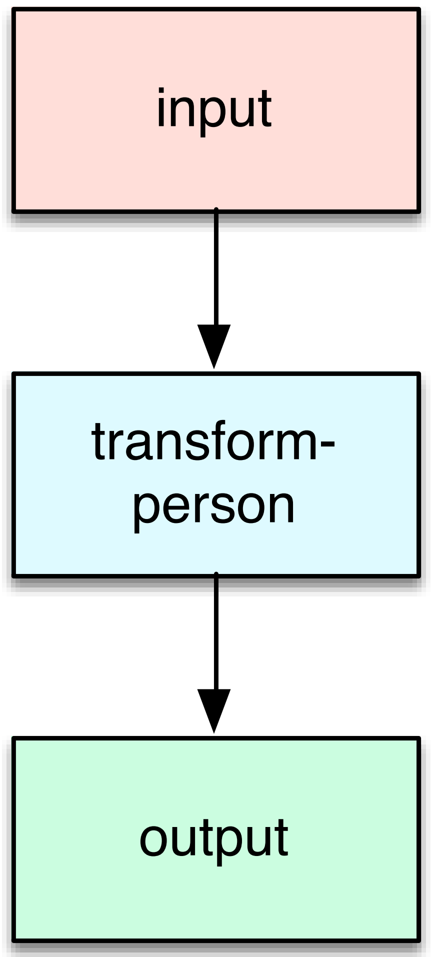
Did Somebody Say Transducers?!

```clojure
(defn transform-person [segment]
  (-> segment
      (update-in [:name] clojure.string/upper-case)
      (update-in [:age] inc)))

(defn derek? [segment]
  (= (:name segment) "Derek"))

(defn x-form []
  (comp (remove derek?))
        (map transform-person)))
```

- Available in edge build (0.4.0-SNAPSHOT)

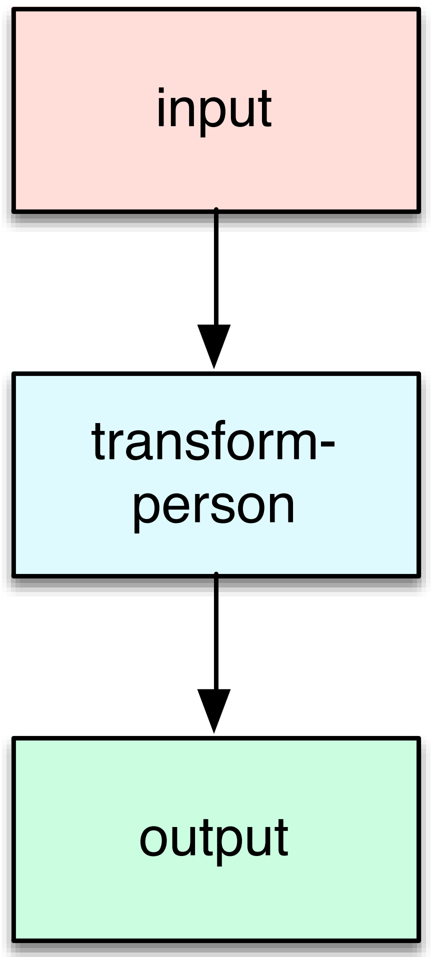
Positional Representation

- **Workflow**: describe the order of data flow through the program



Positional Representation

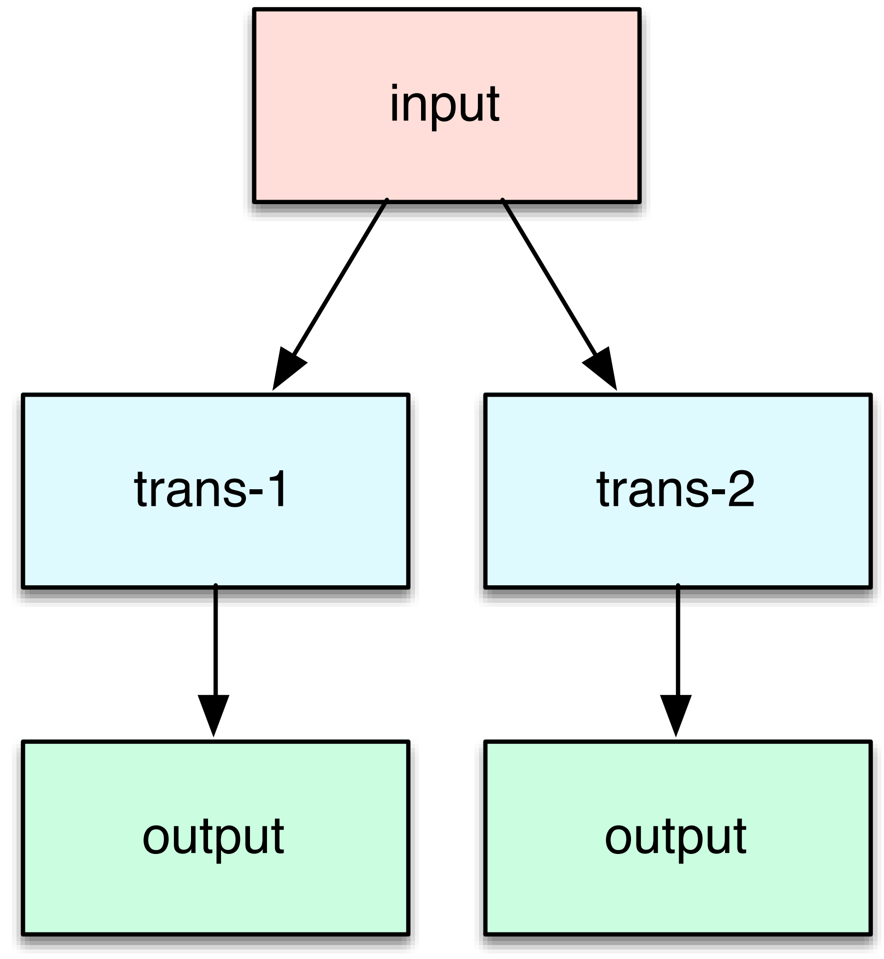
- **Workflow**: describe the order of data flow through the program



```
{:input {:transform-person :output}}
```

Positional Representation

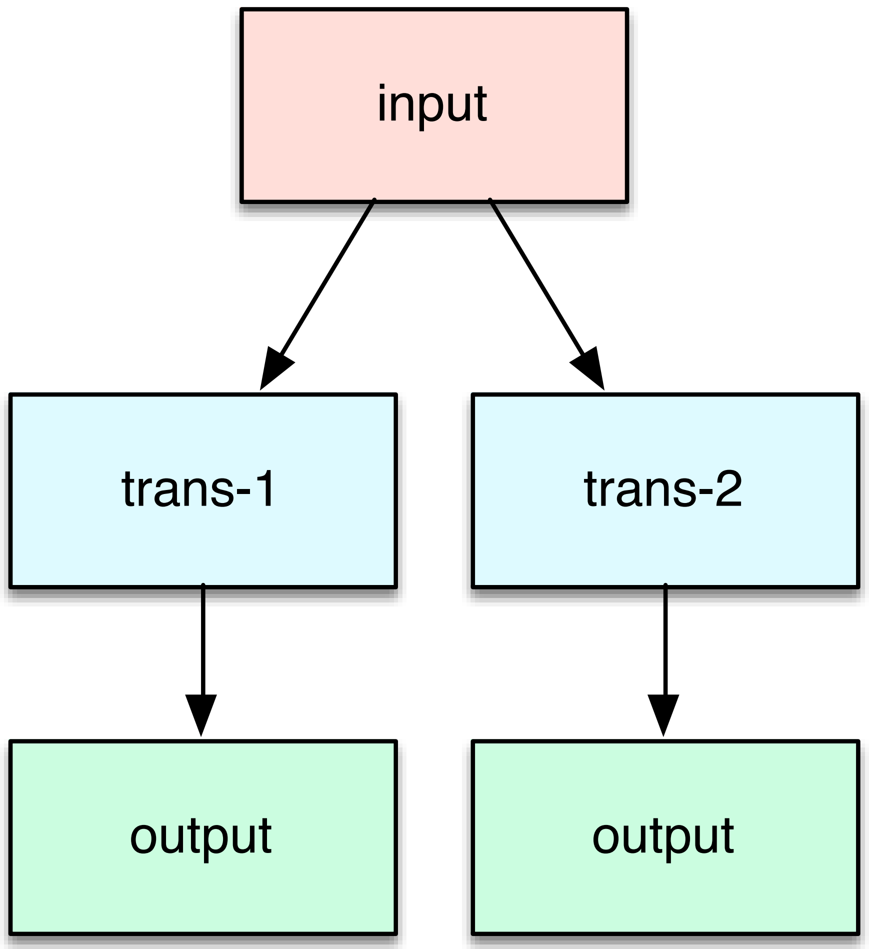
- **Workflow**: describe the order of data flow through the program



```
{:input {:trans-1 :output
         :trans-2 :output}}
```

Positional Representation

- **Workflow**: describe the order of data flow through the program



```
{:input {:trans-1 :output
         :trans-2 :output}}
```

language agnostic

location agnostic

temporally agnostic

machine tolerant

Execution Context

- **Catalog**: configure workflow tasks to implementation workflow

↓

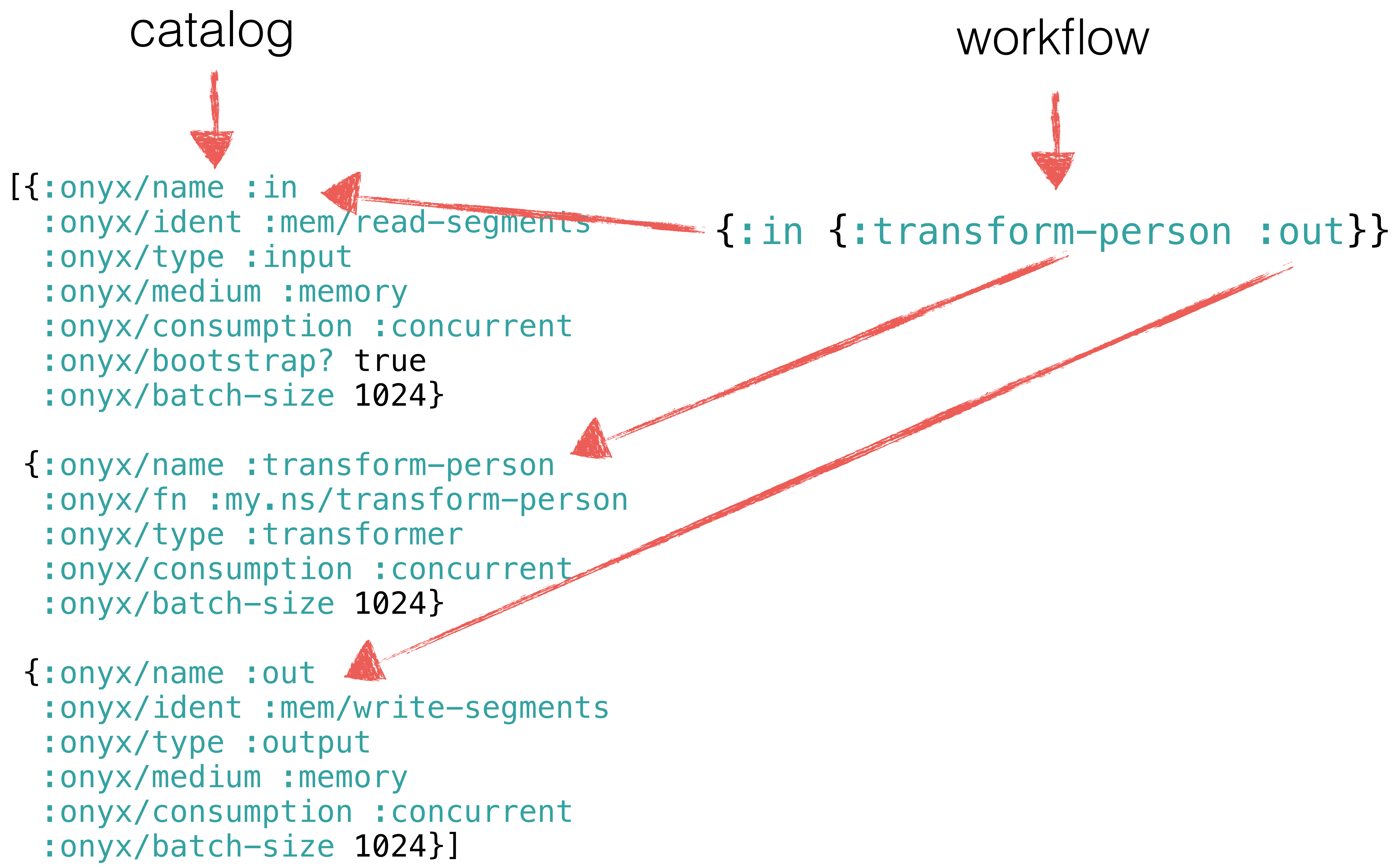
```
{:in {:transform-person :out}}
```

catalog

```
[{:onyx/name :in
  :onyx/ident :mem/read-segments
  :onyx/type :input
  :onyx/medium :memory
  :onyx/consumption :concurrent
  :onyx/bootstrap? true
  :onyx/batch-size 1024}

 {:onyx/name :transform-person
  :onyx/fn :my.ns/transform-person
  :onyx/type :transformer
  :onyx/consumption :concurrent
  :onyx/batch-size 1024}

 {:onyx/name :out
  :onyx/ident :mem/write-segments
  :onyx/type :output
  :onyx/medium :memory
  :onyx/consumption :concurrent
  :onyx/batch-size 1024}]
```

workflow

```
{:in {:transform-person :out}}
```



```
[{:onyx/name :in
  :onyx/ident :mem/read-segments
  :onyx/type :input
  :onyx/medium :memory
  :onyx/consumption :concurrent
  :onyx/bootstrap? true
  :onyx/batch-size 1024}

 {:onyx/name :transform-person
  :onyx/fn :my.ns/transform-person
  :onyx/type :transformer
  :onyx/consumption :concurrent
  :onyx/batch-size 1024}

 {:onyx/name :out
  :onyx/ident :mem/write-segments
  :onyx/type :output
  :onyx/medium :memory
  :onyx/consumption :concurrent
  :onyx/batch-size 1024}]
```

```
{:in {:transform-person :out}}
```

catalog

```
[{:onyx/name :in
  :onyx/ident :mem/read-segments
  :onyx/type :input
  :onyx/medium :memory
  :onyx/consumption :concurrent
  :onyx/bootstrap? true
  :onyx/batch-size 1024}

 {:onyx/name :transform-person
  :onyx/fn :my.ns/transform-person
  :onyx/type :transformer
  :onyx/consumption :concurrent
  :onyx/batch-size 1024}

 {:onyx/name :out
  :onyx/ident :mem/write-segments
  :onyx/type :output
  :onyx/medium :memory
  :onyx/consumption :concurrent
  :onyx/batch-size 1024}]
```

workflow

```
{:in {:transform-person :out}}
```

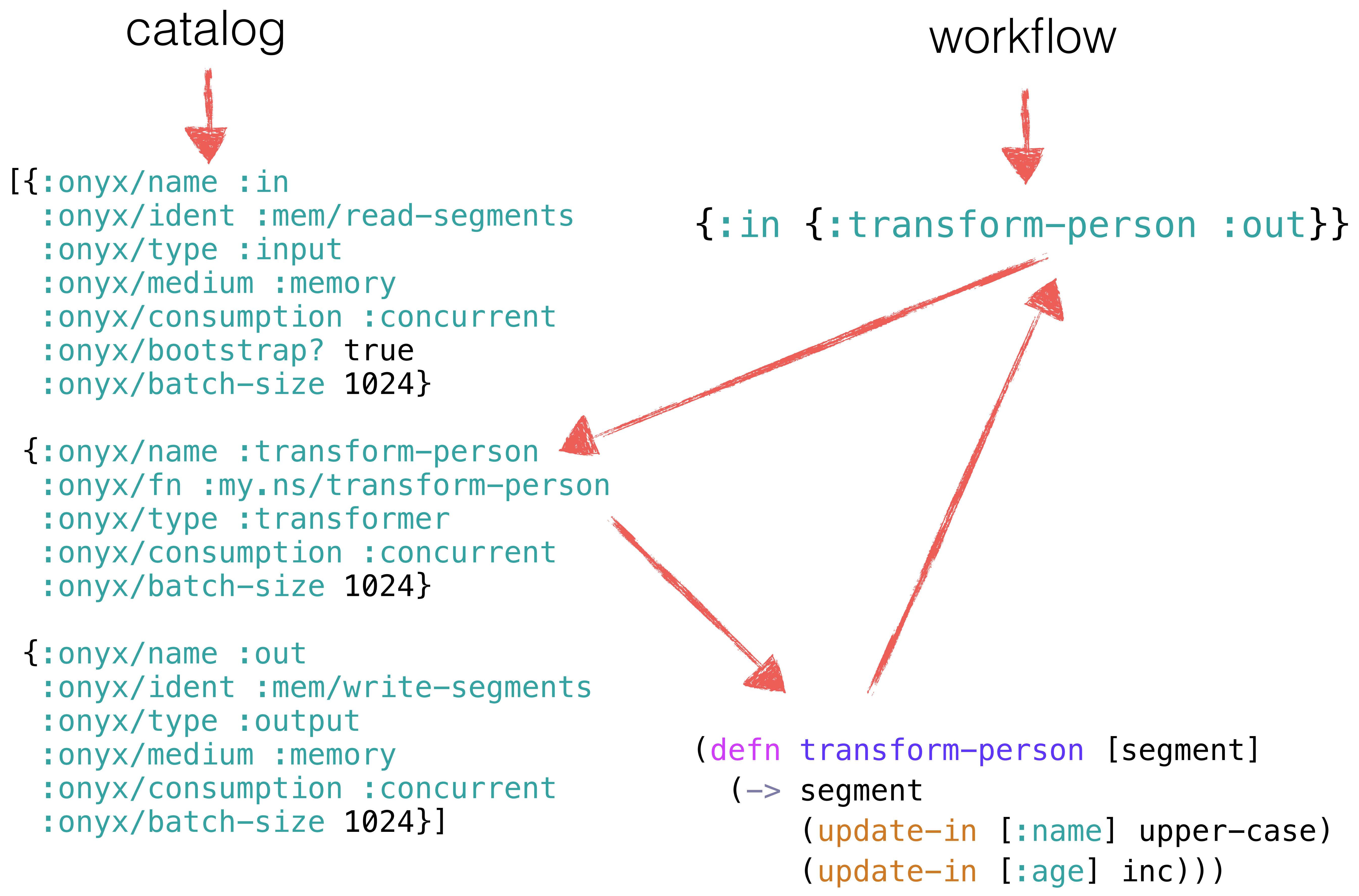
catalog

```
[{:onyx/name :in
  :onyx/ident :mem/read-segments
  :onyx/type :input
  :onyx/medium :memory
  :onyx/consumption :concurrent
  :onyx/bootstrap? true
  :onyx/batch-size 1024}

 {:onyx/name :transform-person
  :onyx/fn :my.ns/transform-person
  :onyx/type :transformer
  :onyx/consumption :concurrent
  :onyx/batch-size 1024}

 {:onyx/name :out
  :onyx/ident :mem/write-segments
  :onyx/type :output
  :onyx/medium :memory
  :onyx/consumption :concurrent
  :onyx/batch-size 1024}]
```

workflow

```
{:in {:transform-person :out}}
```

```
(defn transform-person [segment]
  (-> segment
      (update-in [:name] upper-case)
      (update-in [:age] inc)))
```



catalog

```
[{:onyx/name :in
  :onyx/ident :mem/read-segments
  :onyx/type :input
  :onyx/medium :memory
  :onyx/consumption :concurrent
  :onyx/bootstrap? true
  :onyx/batch-size 1024}

 {:onyx/name :transform-person
  :onyx/fn :my.ns/transform-person
  :onyx/type :transformer
  :onyx/consumption :concurrent
  :onyx/batch-size 1024}

 {:onyx/name :out
  :onyx/ident :mem/write-segments
  :onyx/type :output
  :onyx/medium :memory
  :onyx/consumption :concurrent
  :onyx/batch-size 1024}]
```

workflow

```
{:in {:transform-person :out}}
```

```
(defn transform-person [segment]
  (-> segment
      (update-in [:name] upper-case)
      (update-in [:age] inc)))
```

Extensibility Model

- Extensible to
  - Override things I don't like
  - Augment things I do like
- Reach to new storage mediums
  - With little to no code change in the application
- **Plugins**: adapt through the catalog

```
[{:onyx/name :in
  :onyx/ident :mem/read-segments
  :onyx/type :input
  :onyx/medium :memory
  :onyx/consumption :concurrent
  :onyx/bootstrap? true
  :onyx/batch-size 1024}

 {:onyx/name :transform-person
  :onyx/fn :my.ns/transform-person
  :onyx/type :transformer
  :onyx/consumption :concurrent
  :onyx/batch-size 1024}

 {:onyx/name :out
  :onyx/ident :mem/write-segments
  :onyx/type :output
  :onyx/medium :memory
  :onyx/consumption :concurrent
  :onyx/batch-size 1024}]
```

```clojure
[{:onyx/name :in
  :onyx/ident :mem/read-segments
  :onyx/type :input
  :onyx/medium :memory
  :onyx/consumption :concurrent
  :onyx/bootstrap? true
  :onyx/batch-size 1024}

 {:onyx/name :transform-person
  :onyx/fn :my.ns/transform-person
  :onyx/type :transformer
  :onyx/consumption :concurrent
  :onyx/batch-size 1024}

 {:onyx/name :out
  :onyx/ident :mem/write-segments
  :onyx/type :output
  :onyx/medium :memory
  :onyx/consumption :concurrent
  :onyx/batch-size 1024}]
```

```clojure
{:in {:transform-person :out}}
```

```
[{:onyx/name :in
  :onyx/ident :mem/read-segments
  :onyx/type :input
  :onyx/medium :memory
  :onyx/consumption :concurrent
  :onyx/bootstrap? true
  :onyx/batch-size 1024}

 {:onyx/name :transform-person
  :onyx/fn :my.ns/transform-person
  :onyx/type :transformer
  :onyx/consumption :concurrent
  :onyx/batch-size 1024}

 {:onyx/name :out
  :onyx/ident :mem/write-segments
  :onyx/type :output
  :onyx/medium :memory
  :onyx/consumption :concurrent
  :onyx/batch-size 1024}]
```

```
{:in {:transform-person :out}}
```

```clojure
[{:onyx/name :in
  :onyx/ident :mem/read-segments
  :onyx/type :input
  :onyx/medium :memory
  :onyx/consumption :concurrent
  :onyx/bootstrap? true
  :onyx/batch-size 1024}

 {:onyx/name :transform-person
  :onyx/fn :my.ns/transform-person
  :onyx/type :transformer
  :onyx/consumption :concurrent
  :onyx/batch-size 1024}

 {:onyx/name :out
  :onyx/ident :mem/write-segments
  :onyx/type :output
  :onyx/medium :memory
  :onyx/consumption :concurrent
  :onyx/batch-size 1024}]
```

```clojure
{:onyx/name :in
 :onyx/ident :hornetq/read-segments
 :onyx/type :input
 :onyx/medium :hornetq
 :onyx/consumption :concurrent
 :kafka/topic "my-topic"
 :kafka/zookeeper "127.0.0.1:2181"
 :kafka/group-id "onyx-consumer"
 :kafka/offset-reset "smallest"
 :onyx/batch-size 1024}
```

```clojure
{:in {:transform-person :out}}
```

```
[{:onyx/name :in
  :onyx/ident :mem/read-segments
  :onyx/type :input
  :onyx/medium :memory
  :onyx/consumption :concurrent
  :onyx/bootstrap? true
  :onyx/batch-size 1024}

 {:onyx/name :transform-person
  :onyx/fn :my.ns/transform-person
  :onyx/type :transformer
  :onyx/consumption :concurrent
  :onyx/batch-size 1024}

 {:onyx/name :out
  :onyx/ident :mem/write-segments
  :onyx/type :output
  :onyx/medium :memory
  :onyx/consumption :concurrent
  :onyx/batch-size 1024}]
```

```
{:onyx/name :in
 :onyx/ident :hornetq/read-segments
 :onyx/type :input
 :onyx/medium :hornetq
 :onyx/consumption :concurrent
 :kafka/topic "my-topic"
 :kafka/zookeeper "127.0.0.1:2181"
 :kafka/group-id "onyx-consumer"
 :kafka/offset-reset "smallest"
 :onyx/batch-size 1024}
```

```
{:in {:transform-person :out}}
```

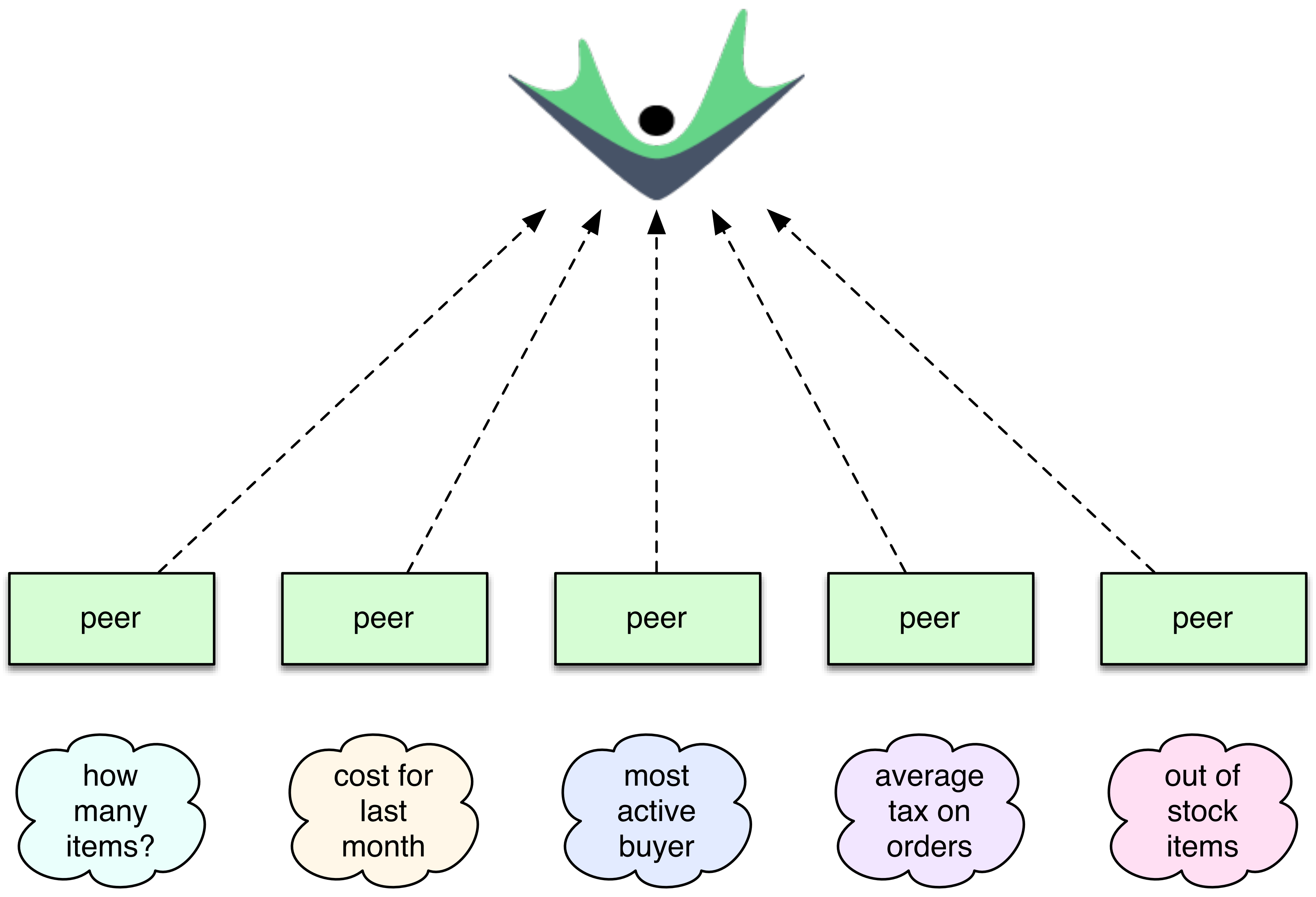
```clojure
[{:onyx/name :in
  :onyx/ident :mem/read-segments
  :onyx/type :input
  :onyx/medium :memory
  :onyx/consumption :concurrent
  :onyx/bootstrap? true
  :onyx/batch-size 1024}

 {:onyx/name :transform-person
  :onyx/fn :my.ns/transform-person
  :onyx/type :transformer
  :onyx/consumption :concurrent
  :onyx/batch-size 1024}

 {:onyx/name :out
  :onyx/ident :mem/write-segments
  :onyx/type :output
  :onyx/medium :memory
  :onyx/consumption :concurrent
  :onyx/batch-size 1024}]
```

```clojure
{:onyx/name :in
 :onyx/ident :hornetq/read-segments
 :onyx/type :input
 :onyx/medium :hornetq
 :onyx/consumption :concurrent
 :kafka/topic "my-topic"
 :kafka/zookeeper "127.0.0.1:2181"
 :kafka/group-id "onyx-consumer"
 :kafka/offset-reset "smallest"
 :onyx/batch-size 1024}
```

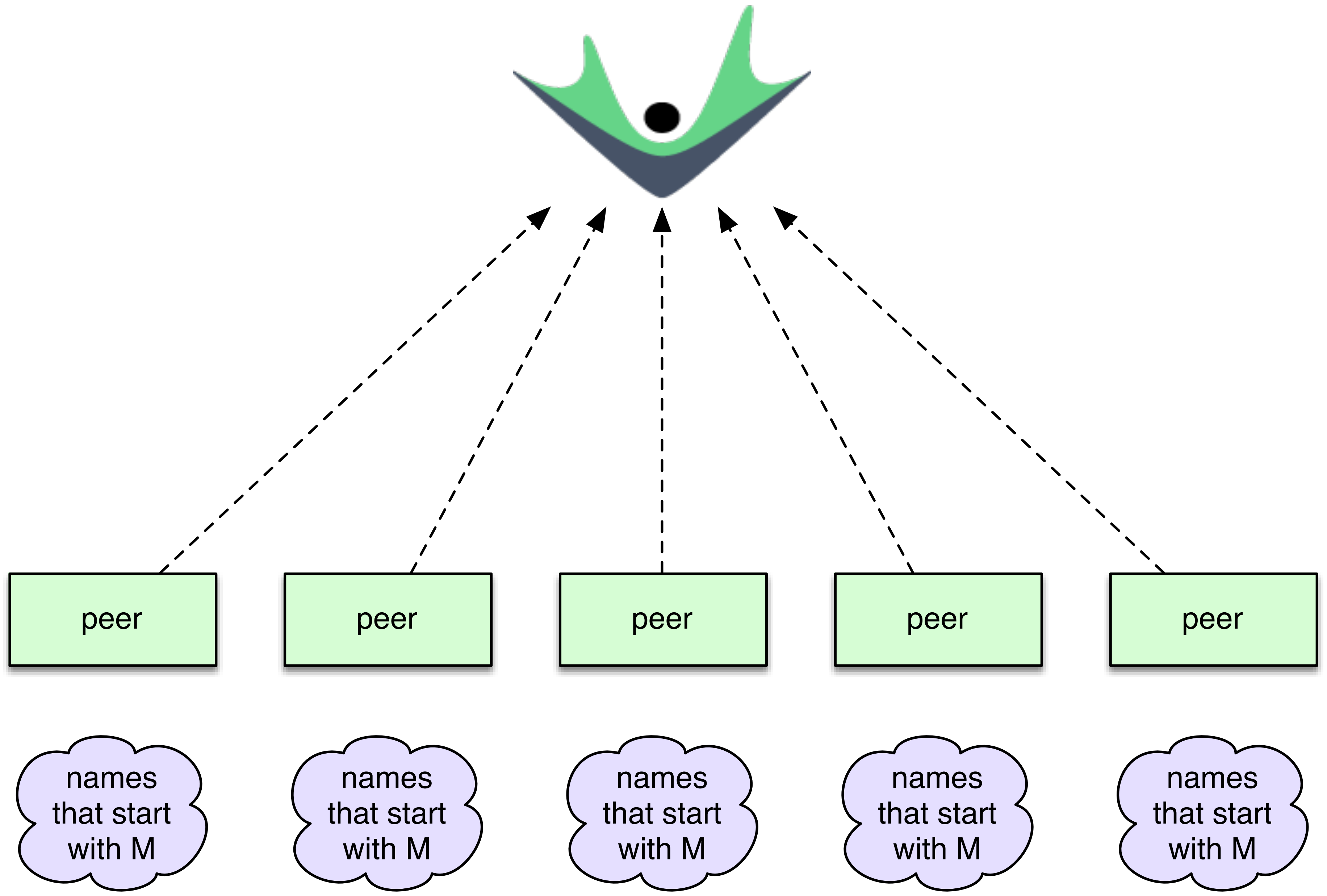
```clojure
{:in {:transform-person :out}}
```

Runtime Discovery

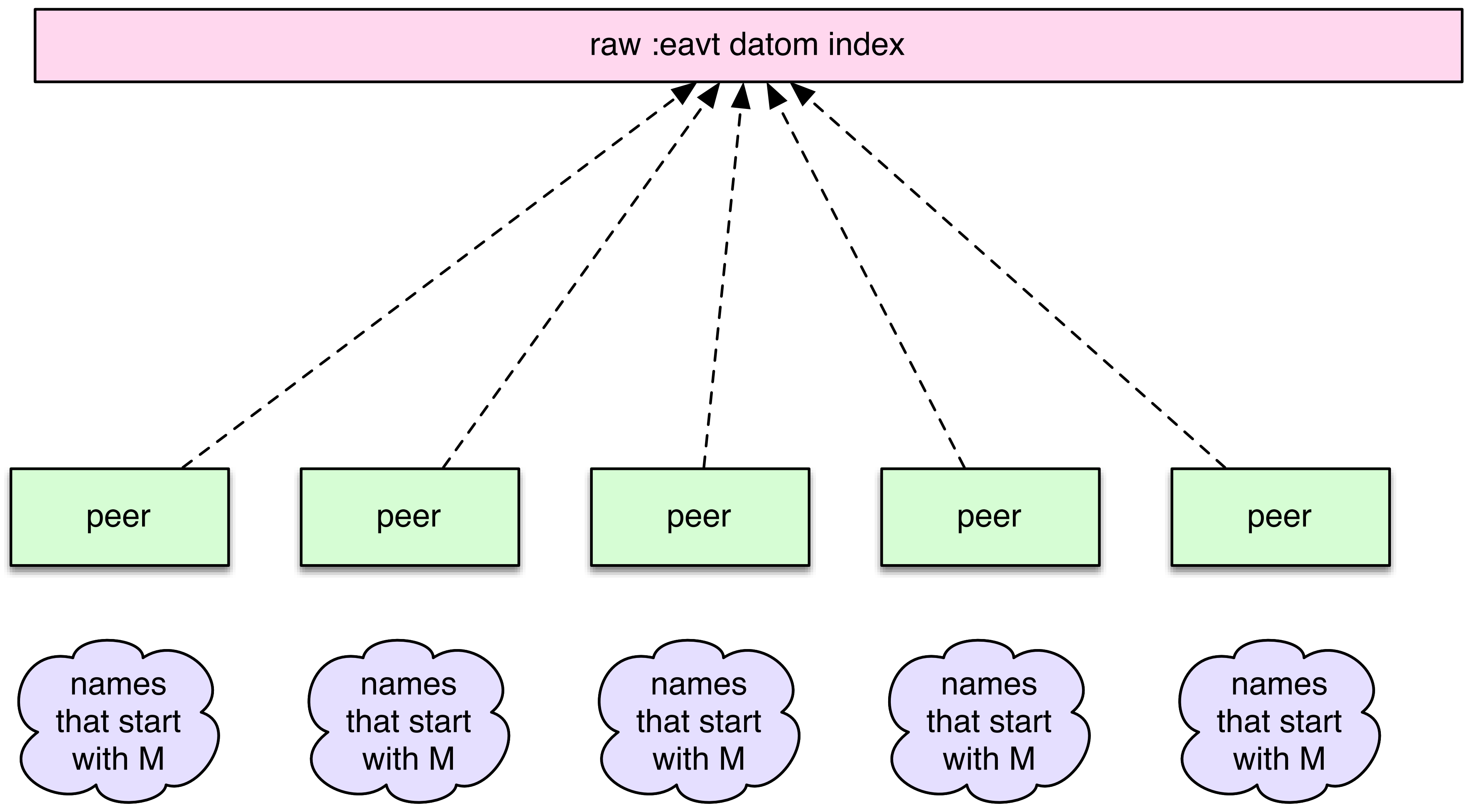
Runtime Discovery



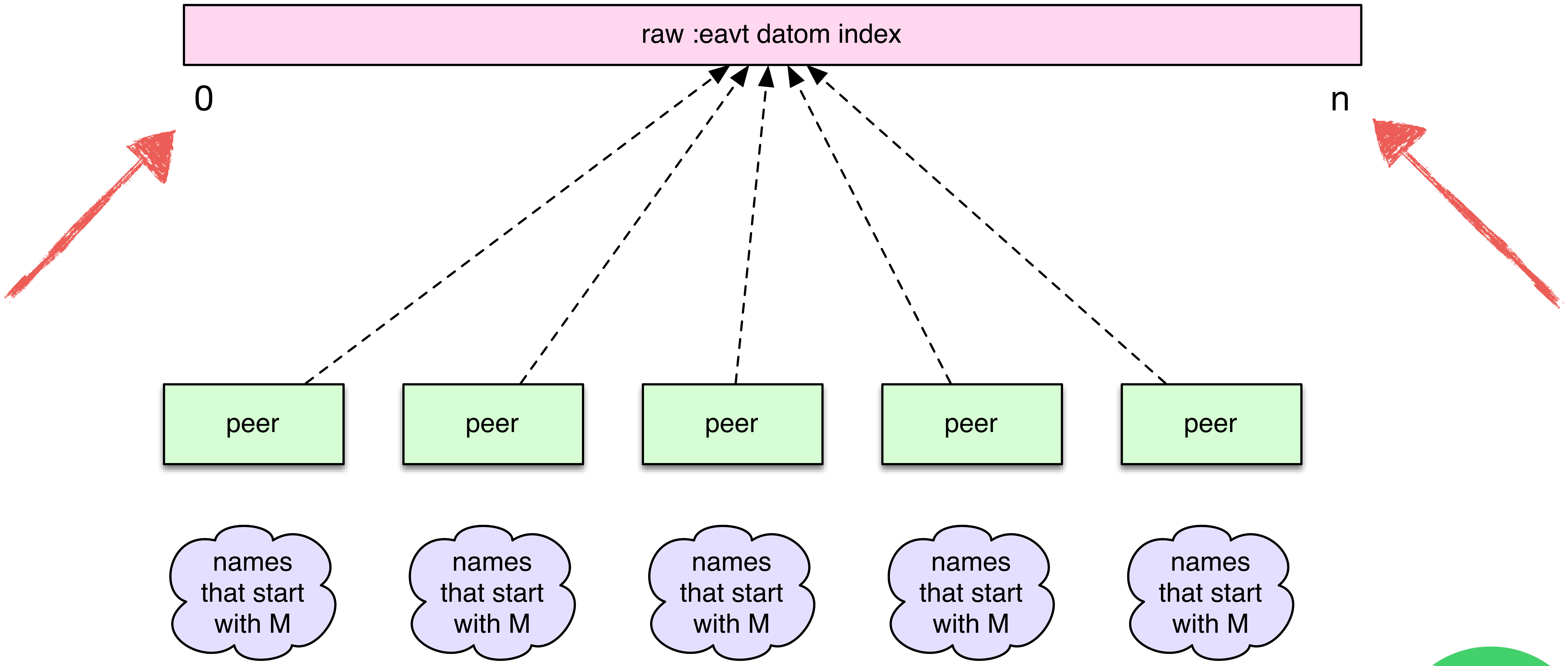
Runtime Discovery



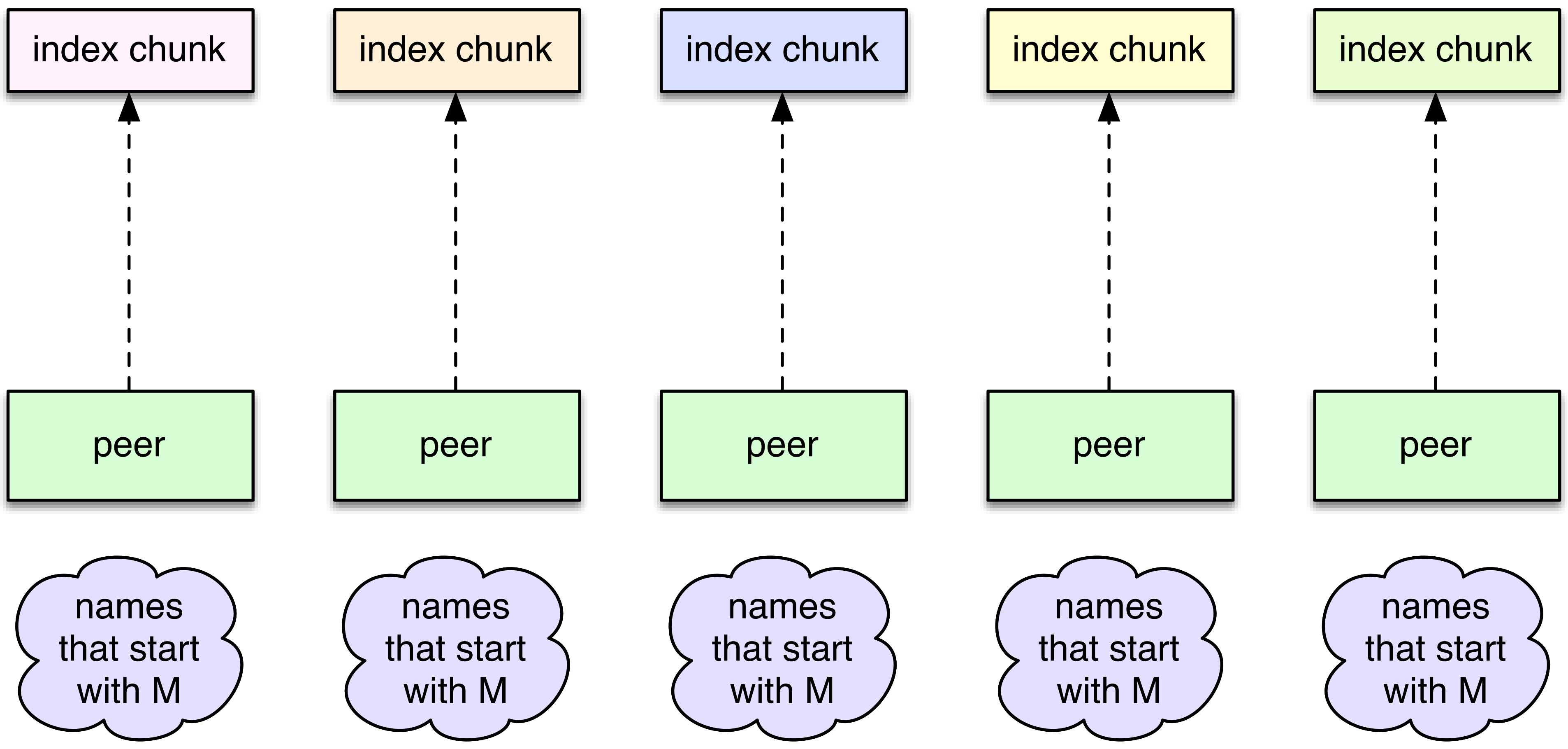
Runtime Discovery



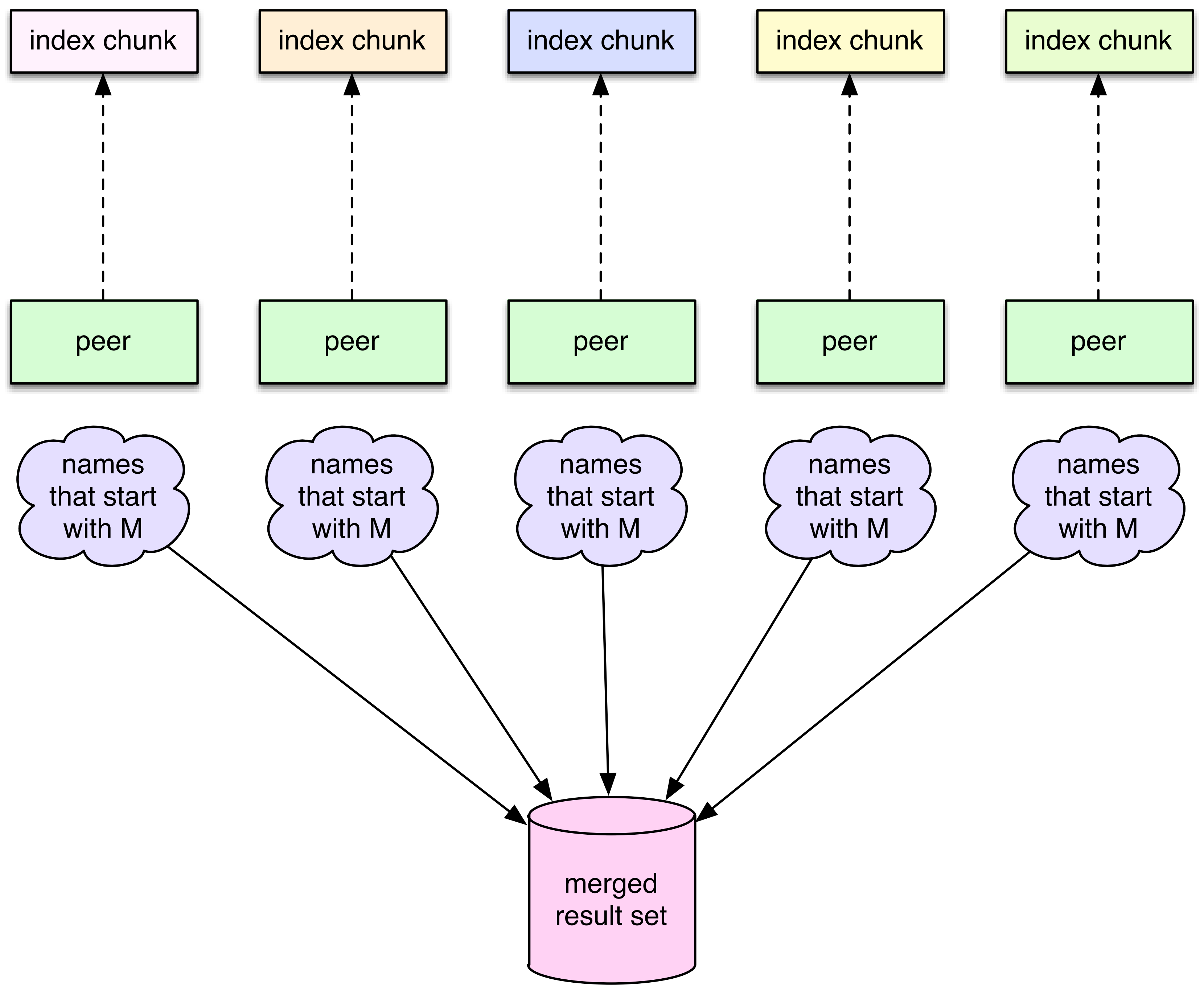
Runtime Discovery



Runtime Discovery



Runtime Discovery



Runtime Discovery

```
{:partition-datoms {:read-datoms {:query :persist}}}
```

Runtime Discovery

```
{:partition-datoms {:read-datoms {:query :persist}}}

 [{:onyx/name :partition-datoms
   :onyx/ident :datomic/partition-datoms
   :onyx/type :input
   :onyx/medium :datomic
   :onyx/consumption :sequential
   :onyx/bootstrap? true
   :datomic/uri db-uri
   :datomic/t t
   :datomic/datoms-per-segment datom-size
   :datomic/partition :com.mdrogalis/people
   :onyx/batch-size batch-size}

  {:onyx/name :read-datoms
   :onyx/ident :datomic/read-datoms
   :onyx/fn :onyx.plugin.datomic/read-datoms
   :onyx/type :transformer
   :onyx/consumption :concurrent
   :onyx/batch-size batch-size
   :datomic/uri db-uri
   :datomic/t t}
```

Runtime Discovery

```
{:partition-datoms {:read-datoms {:query :persist}}}

 [{:onyx/name :partition-datoms
   :onyx/ident :datomic/partition-datoms
   :onyx/type :input
   :onyx/medium :datomic
   :onyx/consumption :sequential
   :onyx/bootstrap? true
   :datomic/uri db-uri
   :datomic/t t
   :datomic/datoms-per-segment datom-size
   :datomic/partition :com.mdrogalis/people
   :onyx/batch-size batch-size}

  {:onyx/name :read-datoms
   :onyx/ident :datomic/read-datoms
   :onyx/fn :onyx.plugin.datomic/read-datoms
   :onyx/type :transformer
   :onyx/consumption :concurrent
   :onyx/batch-size batch-size
   :datomic/uri db-uri
   :datomic/t t}
```

Runtime Discovery

```
{:partition-datoms {:read-datoms {:query :persist}}}

[{:onyx/name :partition-datoms
  :onyx/ident :datomic/partition-datoms
  :onyx/type :input
  :onyx/medium :datomic
  :onyx/consumption :sequential
  :onyx/bootstrap? true
  :datomic/uri db-uri
  :datomic/t t
  :datomic/datoms-per-segment datom-size
  :datomic/partition :com.mdrogalis/people
  :onyx/batch-size batch-size}

 {:onyx/name :read-datoms
  :onyx/ident :datomic/read-datoms
  :onyx/fn :onyx.plugin.datomic/read-datoms
  :onyx/type :transformer
  :onyx/consumption :concurrent
  :onyx/batch-size batch-size
  :datomic/uri db-uri
  :datomic/t t}
```

Runtime Discovery

```
{:partition-datoms {:read-datoms {:query :persist}}}

 [{:onyx/name :partition-datoms
   :onyx/ident :datomic/partition-datoms
   :onyx/type :input
   :onyx/medium :datomic
   :onyx/consumption :sequential
   :onyx/bootstrap? true
   :datomic/uri db-uri
   :datomic/t t
   :datomic/datoms-per-segment datom-size
   :datomic/partition :com.mdrogalis/people
   :onyx/batch-size batch-size}

  {:onyx/name :read-datoms
   :onyx/ident :datomic/read-datoms
   :onyx/fn :onyx.plugin.datomic/read-datoms
   :onyx/type :transformer
   :onyx/consumption :concurrent
   :onyx/batch-size batch-size
   :datomic/uri db-uri
   :datomic/t t}
```

Runtime Discovery

```
{:partition-datoms {:read-datoms {:query :persist}}}
```

```
[{:onyx/name :partition-datoms
  :onyx/ident :datomic/partition-datoms
  :onyx/type :input
  :onyx/medium :datomic
  :onyx/consumption :sequential
  :onyx/bootstrap? true
  :datomic/uri db-uri
  :datomic/t t
  :datomic/datoms-per-segment datom-size
  :datomic/partition :com.mdrogalis/people
  :onyx/batch-size batch-size}

 {:onyx/name :read-datoms
  :onyx/ident :datomic/read-datoms
  :onyx/fn :onyx.plugin.datomic/read-datoms
  :onyx/type :transformer
  :onyx/consumption :concurrent
  :onyx/batch-size batch-size
  :datomic/uri db-uri
  :datomic/t t}
```

```
(defn my-query [{:keys [datoms}]
  {:result (d/q query datoms)})
```

Task State API

- Your program *needs* side-effects
    - database connection, file handles, …
- def/defonce - uh oh
- **Lifecycles**: managed set-up/teardown of state

Task State API

```
(defmulti start-lifecycle?)

(defmulti inject-lifecycle-resources)

(defmulti inject-temporal-resources)

(defmulti close-temporal-resources)

(defmulti close-lifecycle-resources)
```

Task State API

```
(defmulti start-lifecycle?)

(defmulti inject-lifecycle-resources)

(defmulti inject-temporal-resources)

(defmulti close-temporal-resources)

(defmulti close-lifecycle-resources)
```

Task State API

```
(defmulti start-lifecycle?)

(defmulti inject-lifecycle-resources)

(defmulti inject-temporal-resources)

(defmulti close-temporal-resources)

(defmulti close-lifecycle-resources)
```

Task State API

```
{:onyx/name :my-task
 :onyx/ident :strangeloop/task
 :onyx/fn :ns.strangeloop/task
 :onyx/type :transformer
 :onyx/consumption :concurrent
 :my/param "42"
 :onyx/batch-size 1024}
```

Task State API

```
{:onyx/name :my-task
 :onyx/ident :strangeloop/task
 :onyx/fn :ns.strangeloop/task
 :onyx/type :transformer
 :onyx/consumption :concurrent
 :my/param "42"
 :onyx/batch-size 1024}
```

Task State API

```clojure
{:onyx/name :my-task
 :onyx/ident :strangeloop/task
 :onyx/fn :ns.strangeloop/task
 :onyx/type :transformer
 :onyx/consumption :concurrent
 :my/param "42"
 :onyx/batch-size 1024}

(defmethod l-ext/inject-lifecycle-resources
  :strangeloop/task
  [_ {:keys [onyx.core/task-map]}]
  {:my-thing (create-stateful-thing (:my/param task-map))})
```

Task State API

```
{:onyx/name :my-task
 :onyx/ident :strangeloop/task
 :onyx/fn :ns.strangeloop/task
 :onyx/type :transformer
 :onyx/consumption :concurrent
 :my/param "42"
 :onyx/batch-size 1024}

(defmethod l-ext/inject-lifecycle-resources
  :strangeloop/task
  [_ {:keys [onyx.core/task-map]}]
  {:my-thing (create-stateful-thing (:my/param task-map))})
```

Task State API

```
{:onyx/name :my-task
 :onyx/ident :strangeloop/task
 :onyx/fn :ns.strangeloop/task
 :onyx/type :transformer
 :onyx/consumption :concurrent
 :my/param "42"
 :onyx/batch-size 1024}

(defmethod l-ext/inject-lifecycle-resources
  :strangeloop/task
  [_ {:keys [onyx.core/task-map]}]
  {:my-thing (create-stateful-thing (:my/param task-map))})
```

Task State API

```clojure
{:onyx/name :my-task
 :onyx/ident :strangeloop/task
 :onyx/fn :ns.strangeloop/task
 :onyx/type :transformer
 :onyx/consumption :concurrent
 :my/param "42"
 :onyx/batch-size 1024}

(defmethod l-ext/inject-lifecycle-resources
  :strangeloop/task
  [_ {:keys [onyx.core/task-map]}]
  {:my-thing (create-stateful-thing (:my/param task-map))})
```

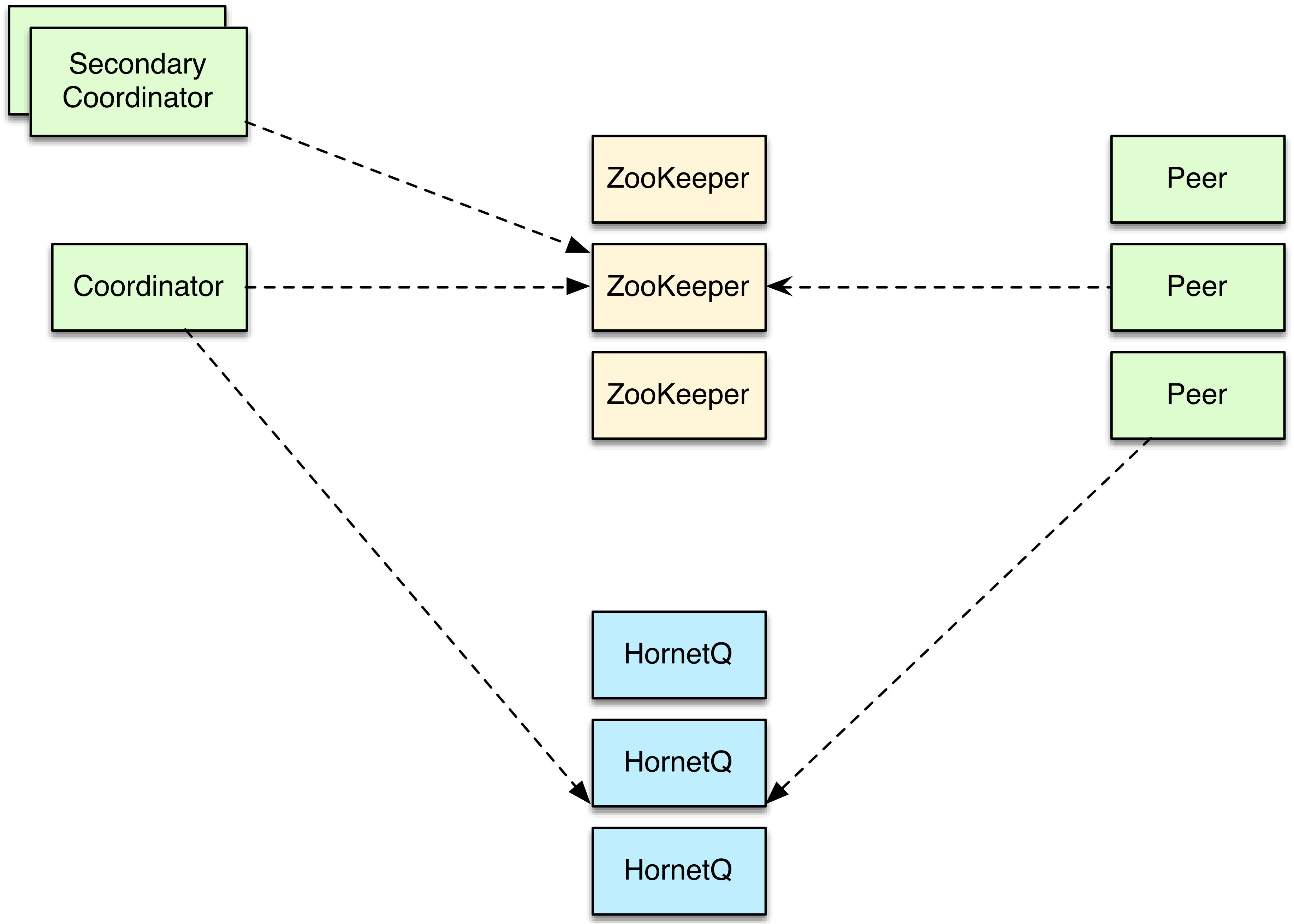
Task State API

```
{:onyx/name :my-task
 :onyx/ident :strangeloop/task
 :onyx/fn :ns.strangeloop/task
 :onyx/type :transformer
 :onyx/consumption :concurrent
 :my/param "42"
 :onyx/batch-size 1024}

(defmethod l-ext/inject-lifecycle-resources
  :strangeloop/task
  [_ {:keys [onyx.core/task-map]}]
  {:my-thing (create-stateful-thing (:my/param task-map))})


(defmethod l-ext/close-temporal-resources
  :strangeloop/task
  [_ event]
  (do-side-effects (:my-thing event))
  {})
```

Task State API

```
{:onyx/name :my-task
 :onyx/ident :strangeloop/task
 :onyx/fn :ns.strangeloop/task
 :onyx/type :transformer
 :onyx/consumption :concurrent
 :my/param "42"
 :onyx/batch-size 1024}

(defmethod l-ext/inject-lifecycle-resources
  :strangeloop/task
  [_ {:keys [onyx.core/task-map]}]
  {:my-thing (create-stateful-thing (:my/param task-map))})


(defmethod l-ext/close-temporal-resources
  :strangeloop/task
  [_ event]
  (do-side-effects (:my-thing event))
  {})
```

Architecture Overview



Wait! That's Slow!

- Queue-based architectures criticized for speed
- Disk locality within data center irrelevant (Berkeley)
- 10+ gig switches now available in data centers

Demo!

Thanks

- CircleCI
- Infinite Cloud

Questions?

https://github.com/MichaelDrogalis/onyx

@MichaelDrogalis