#lang minimart

Actors for Racket

Tony Garnock-Jones
tonyg@ccs.neu.edu

Racketcon 2014, Sep. 20th, St. Louis, MO

Actors

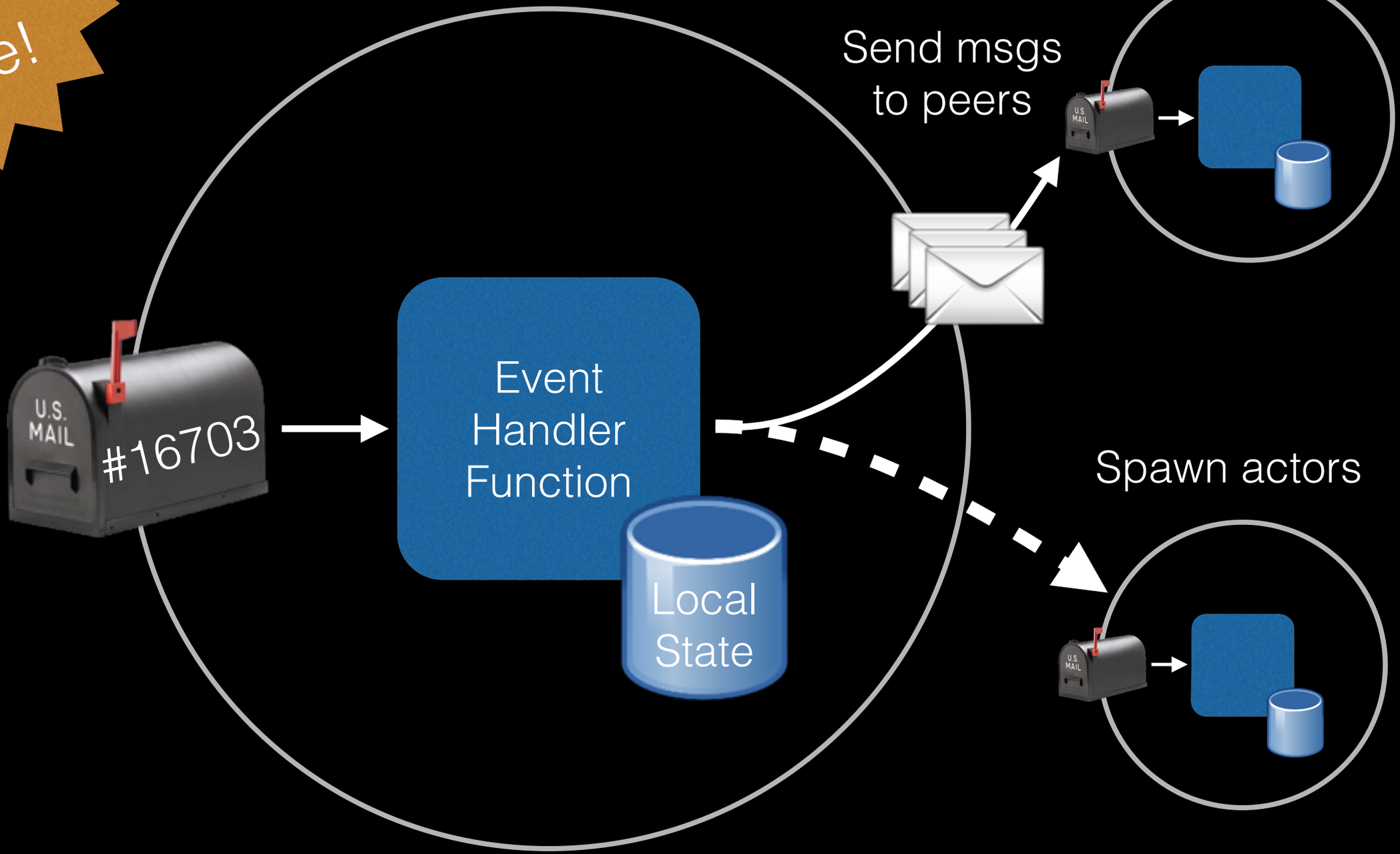
Languages

Influenced

Libraries

Networks

Basic Actor Model



Too Simple

Ubiquitous Problems…

Event broadcasting
Crash/exit signaling
Readiness notification

Composition
Distribution

… many more

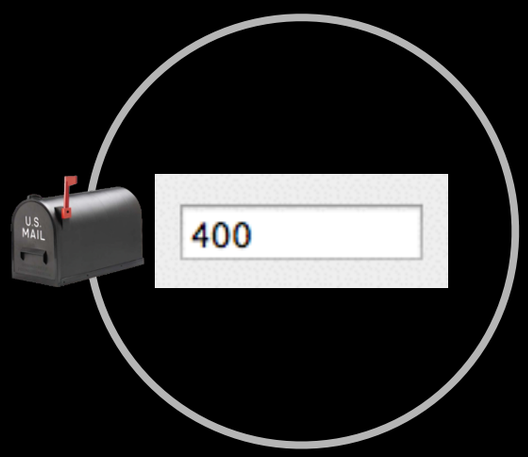
…Uniform Solution

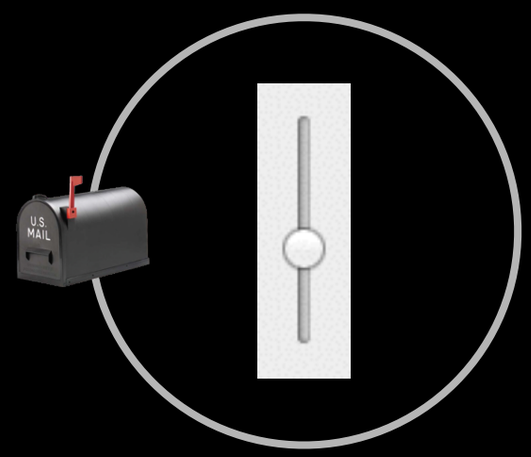
Event broadcasting → Publish / Subscribe (Multicast)

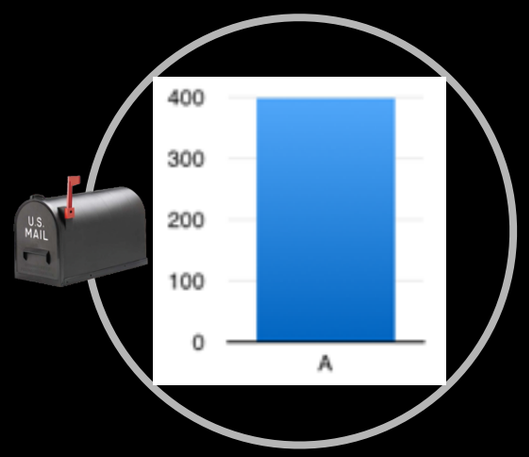
Crash/exit signaling
Readiness notification → Routing Events

Composition
Distribution → Layered groups of *functional* actors

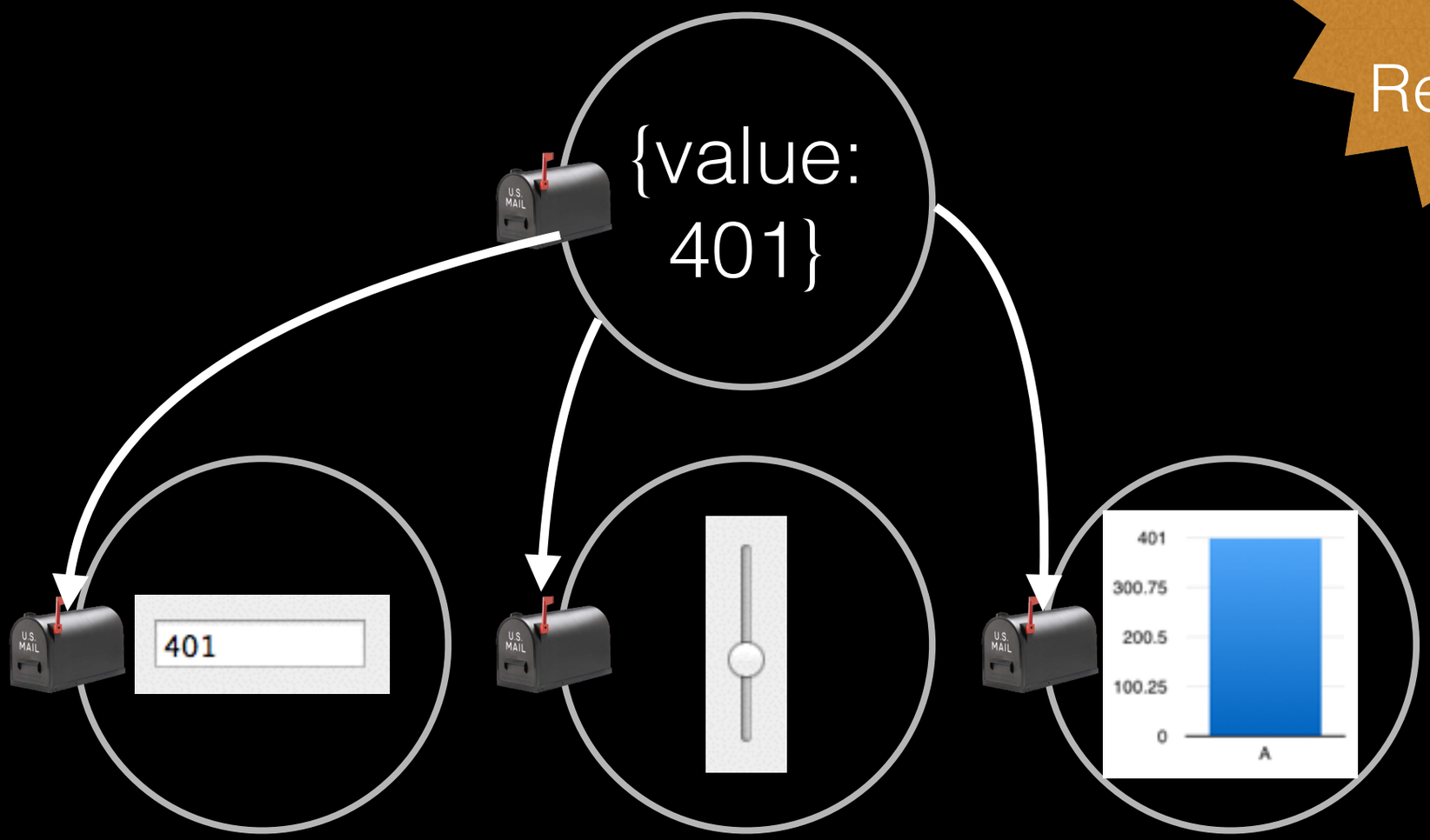
Event Broadcast





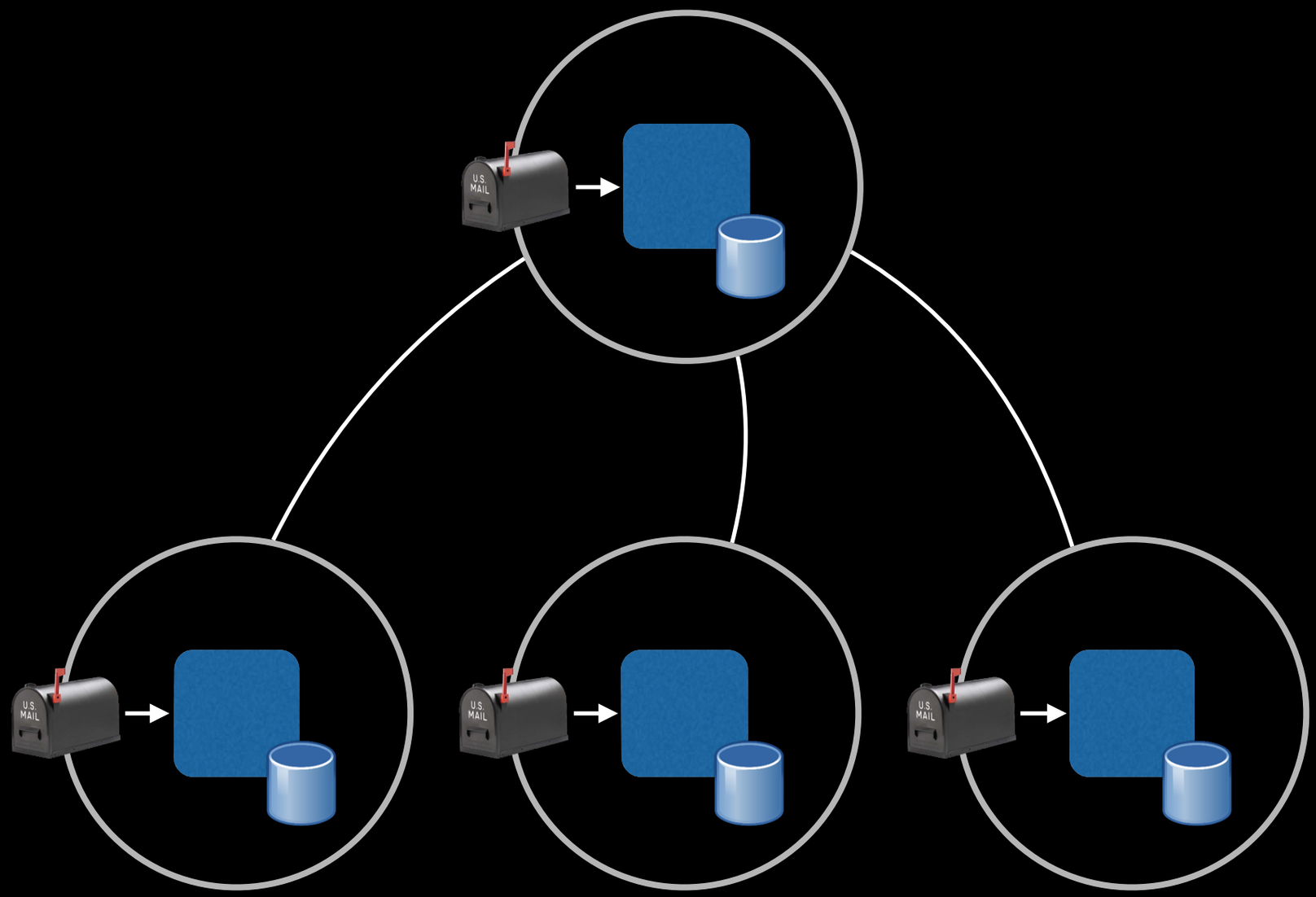


Event Broadcast

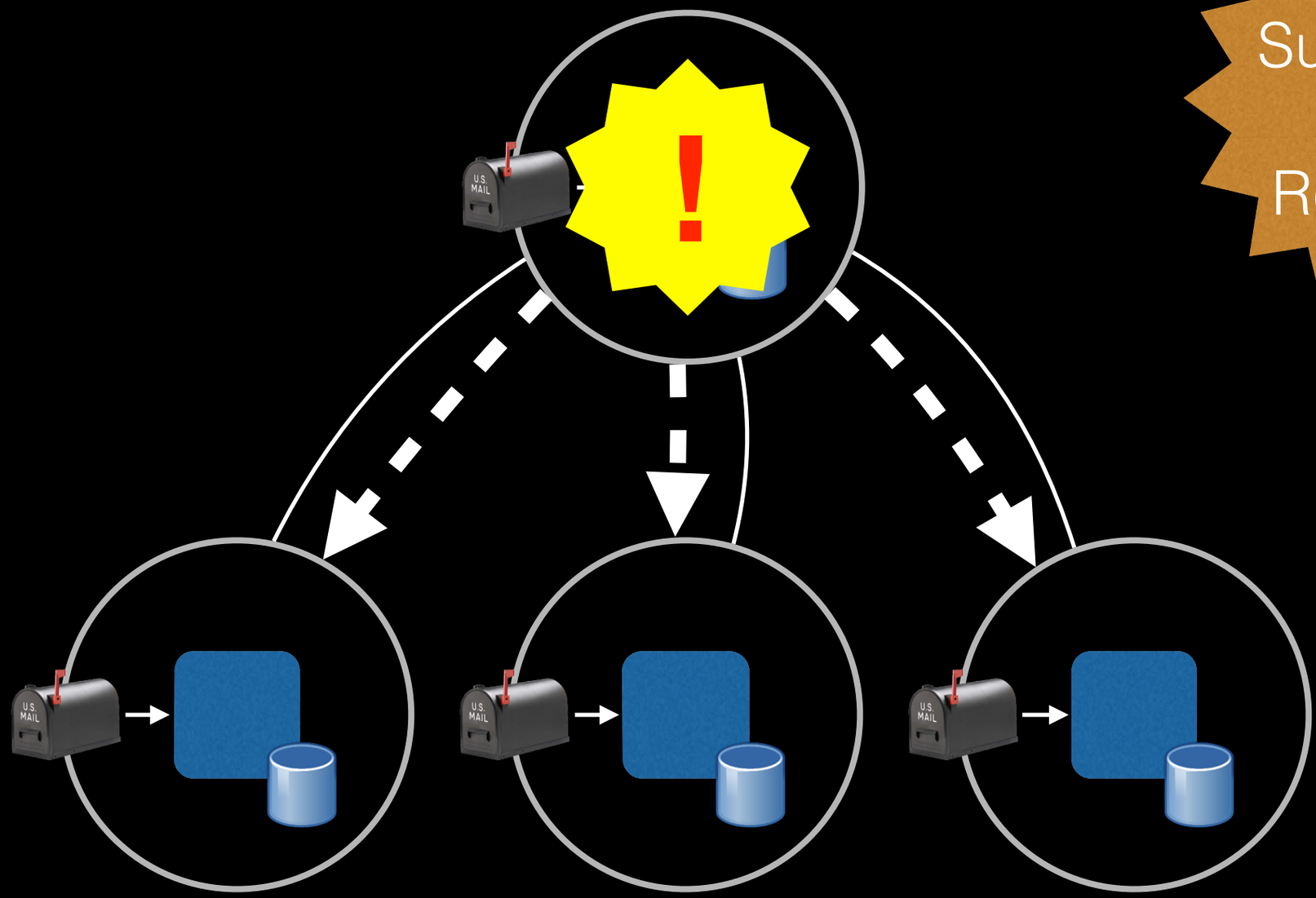


Subscriber List Required!

Crash Signalling

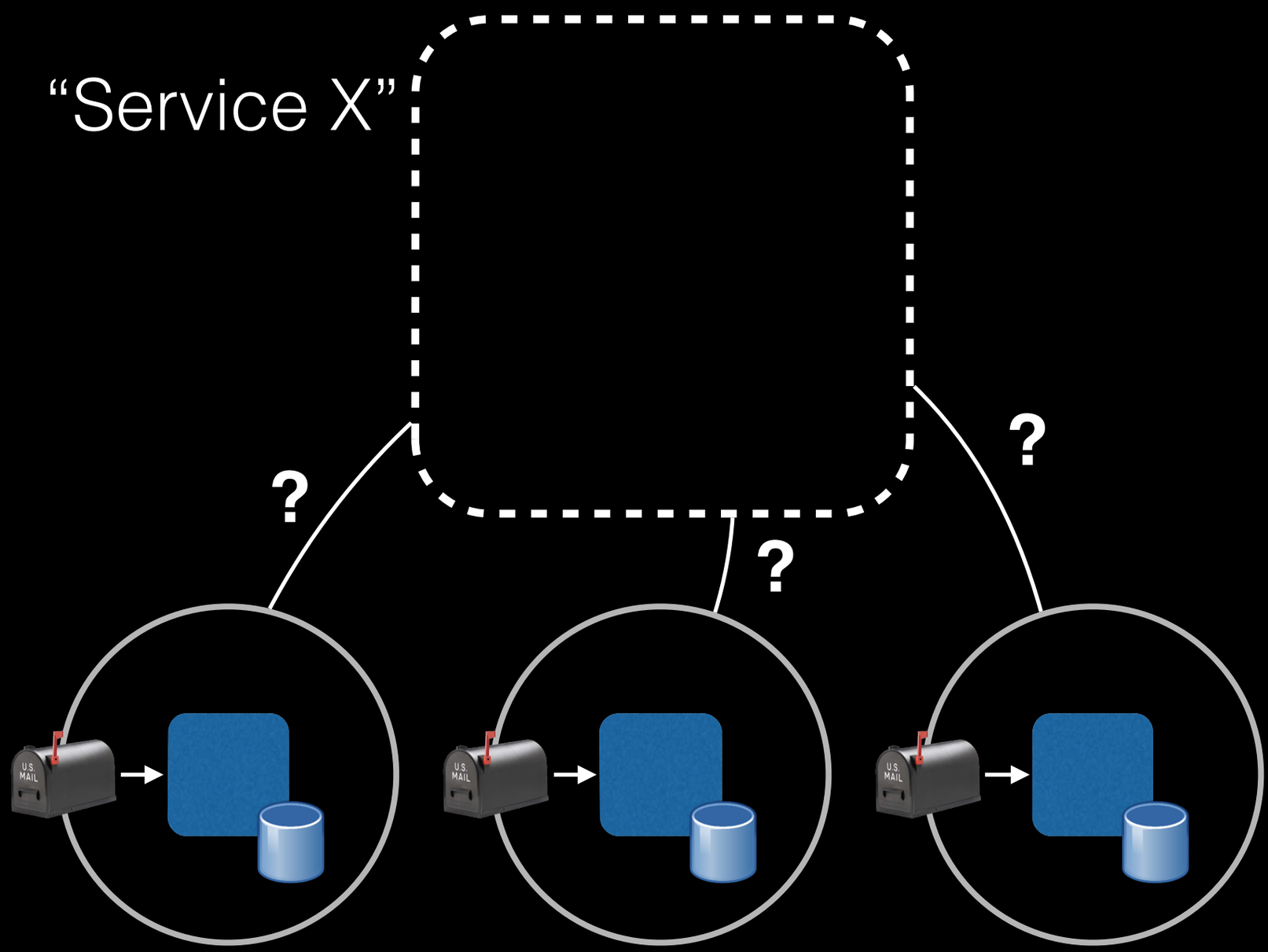


Crash Signalling

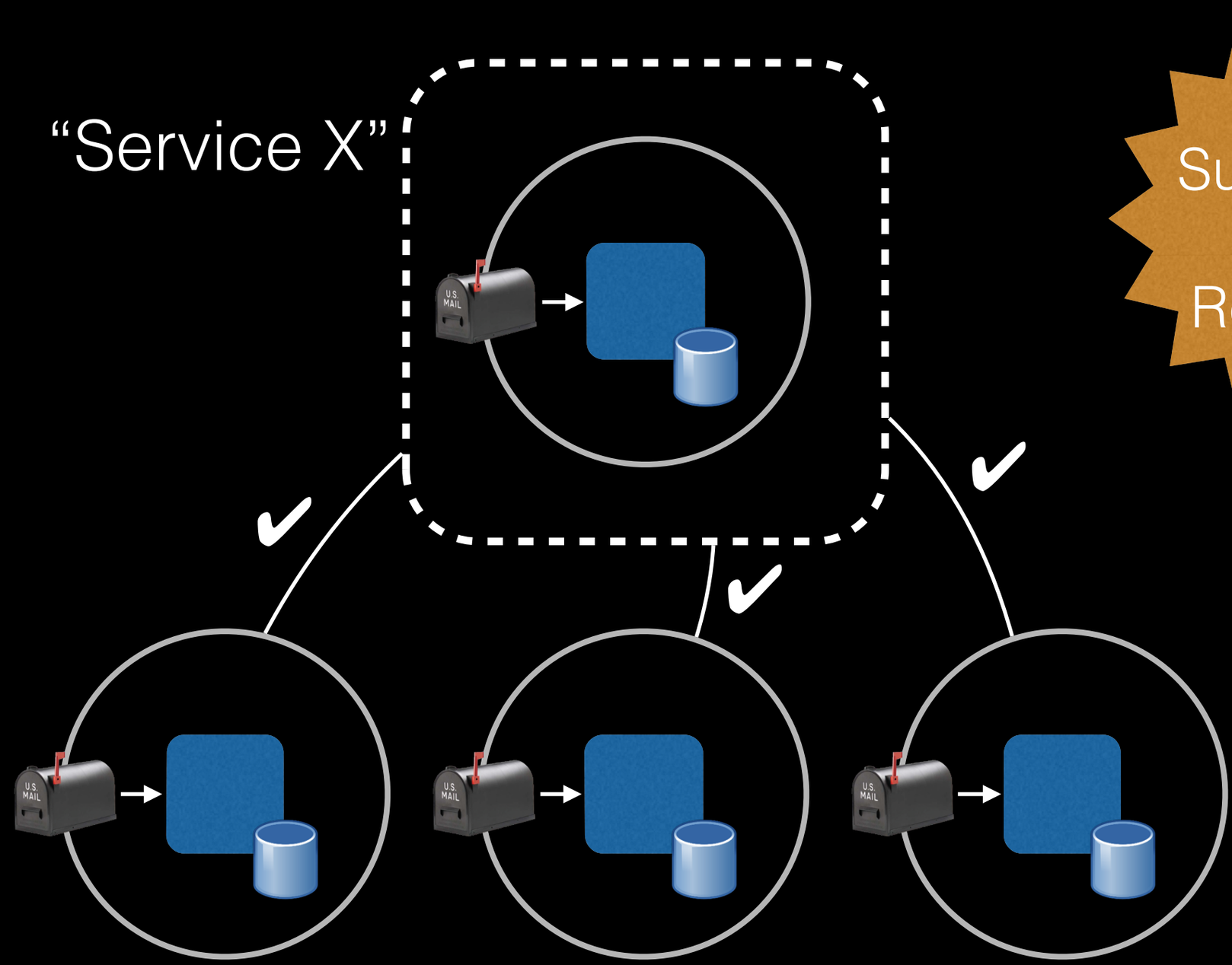


Subscriber List Required!

Readiness Notification

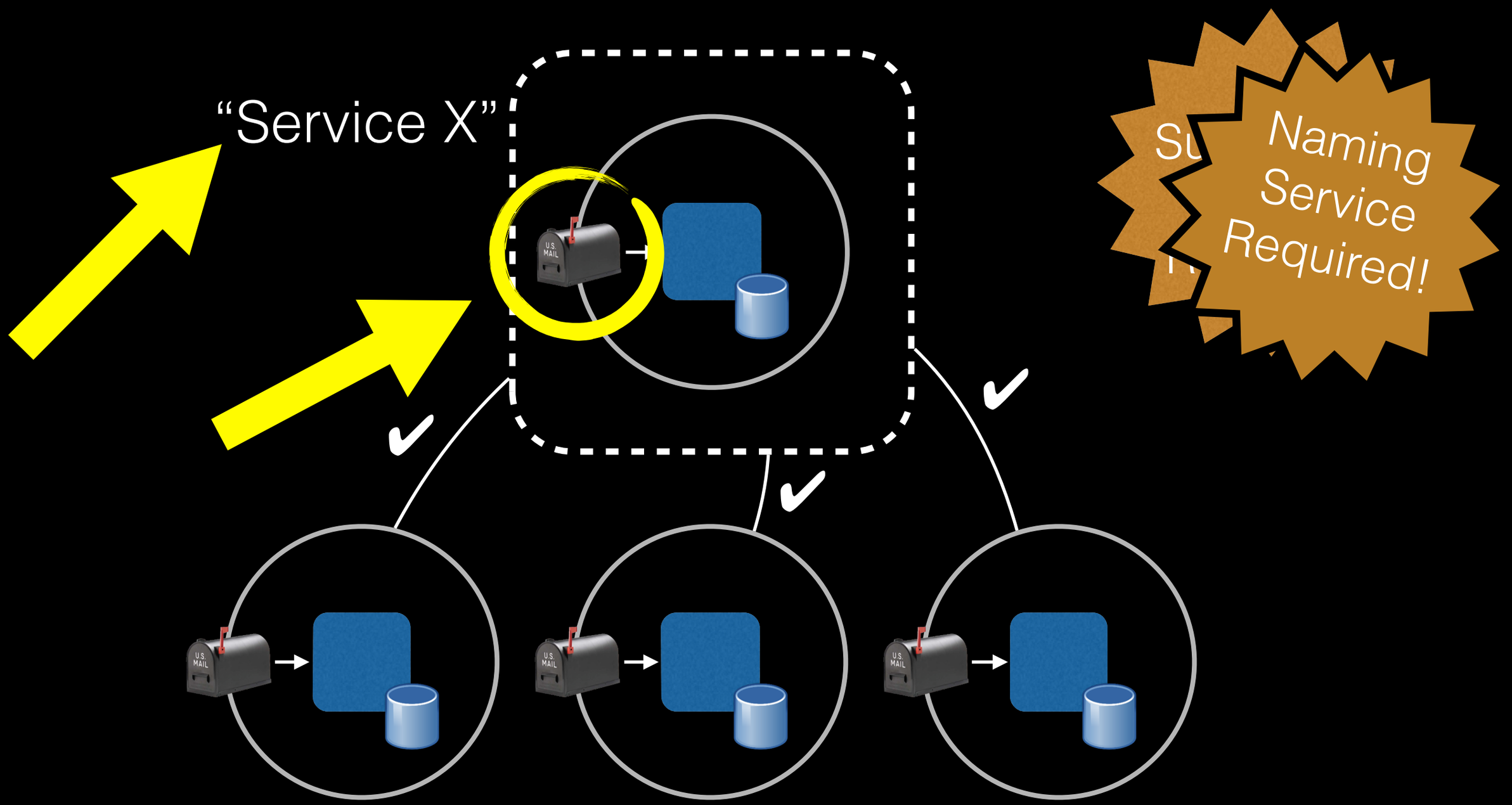


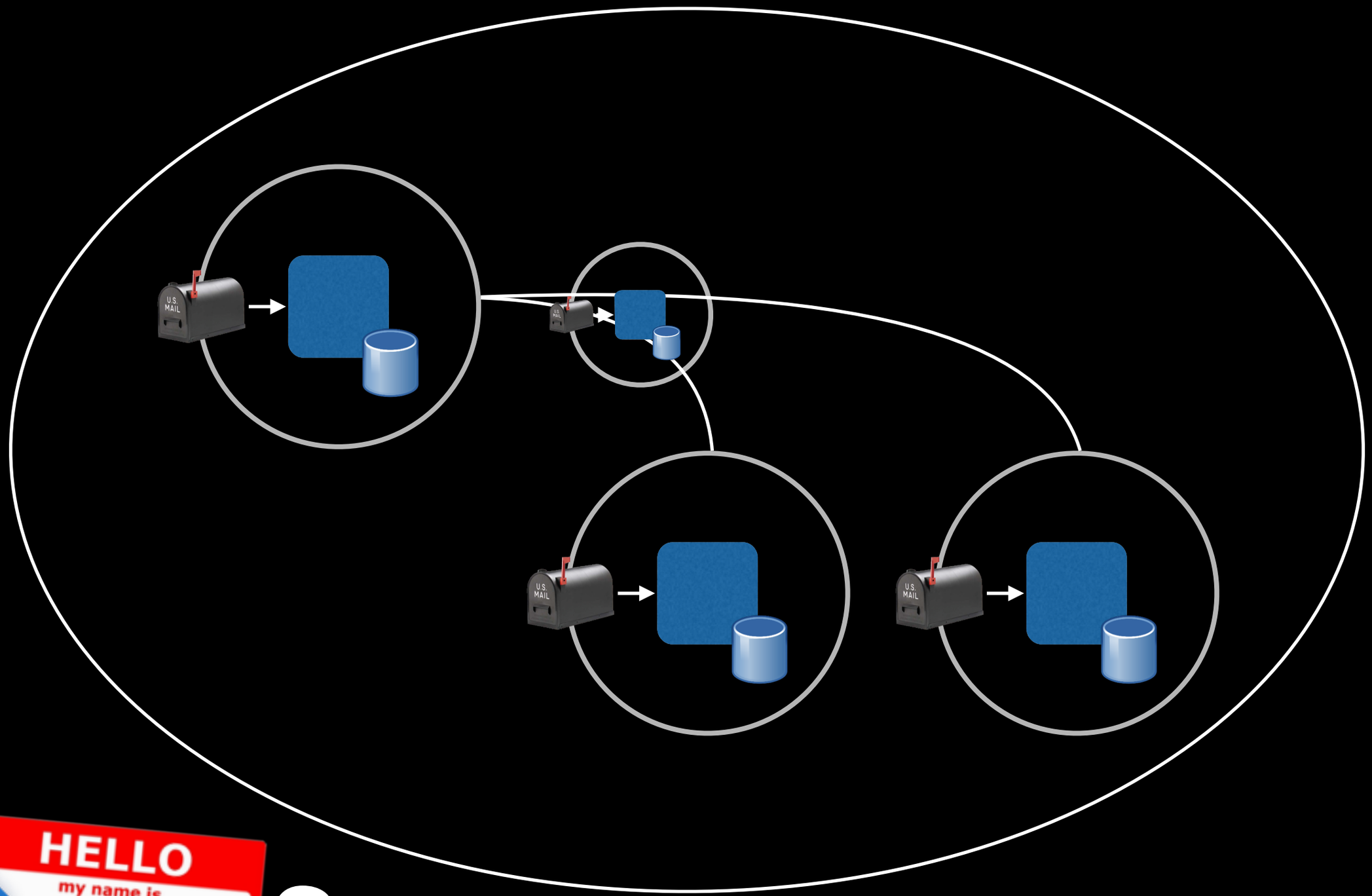
Readiness Notification



Subscriber List Required!

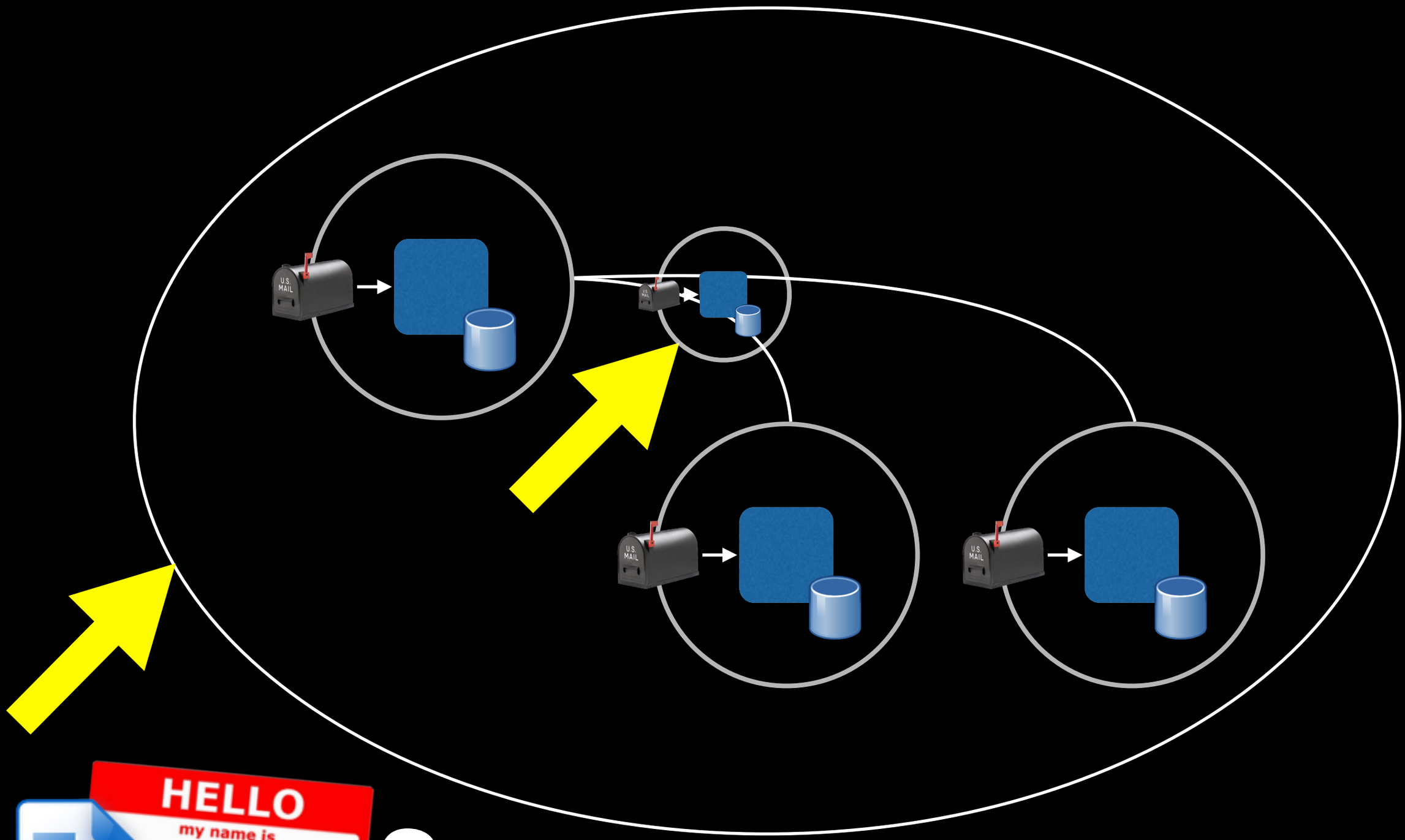
Readiness Notification




U.S. MAIL
U.S. MAIL
U.S. MAIL
U.S. MAIL

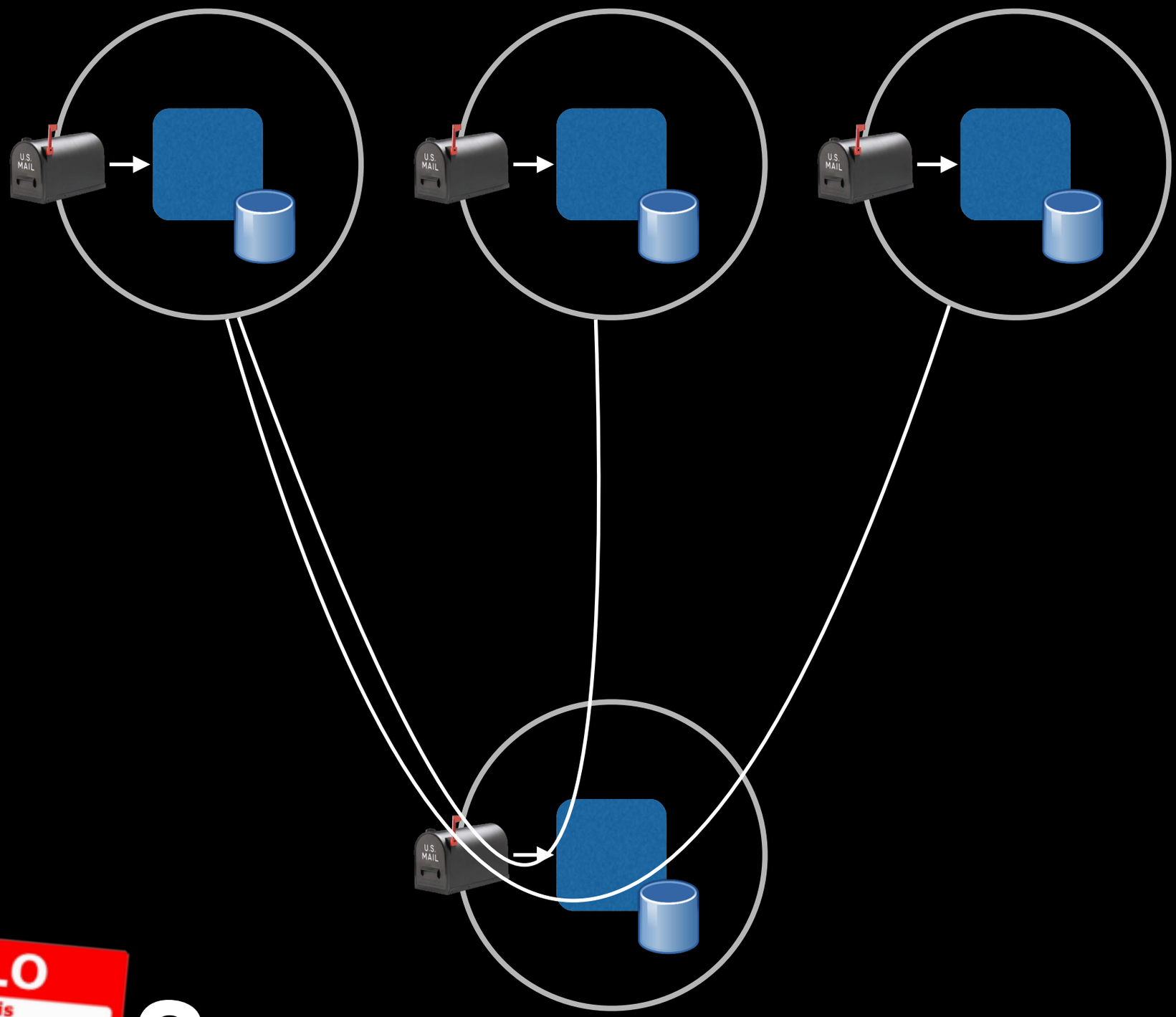

HELLO
my name is

?


U.S. MAIL
U.S. MAIL
U.S. MAIL
U.S. MAIL

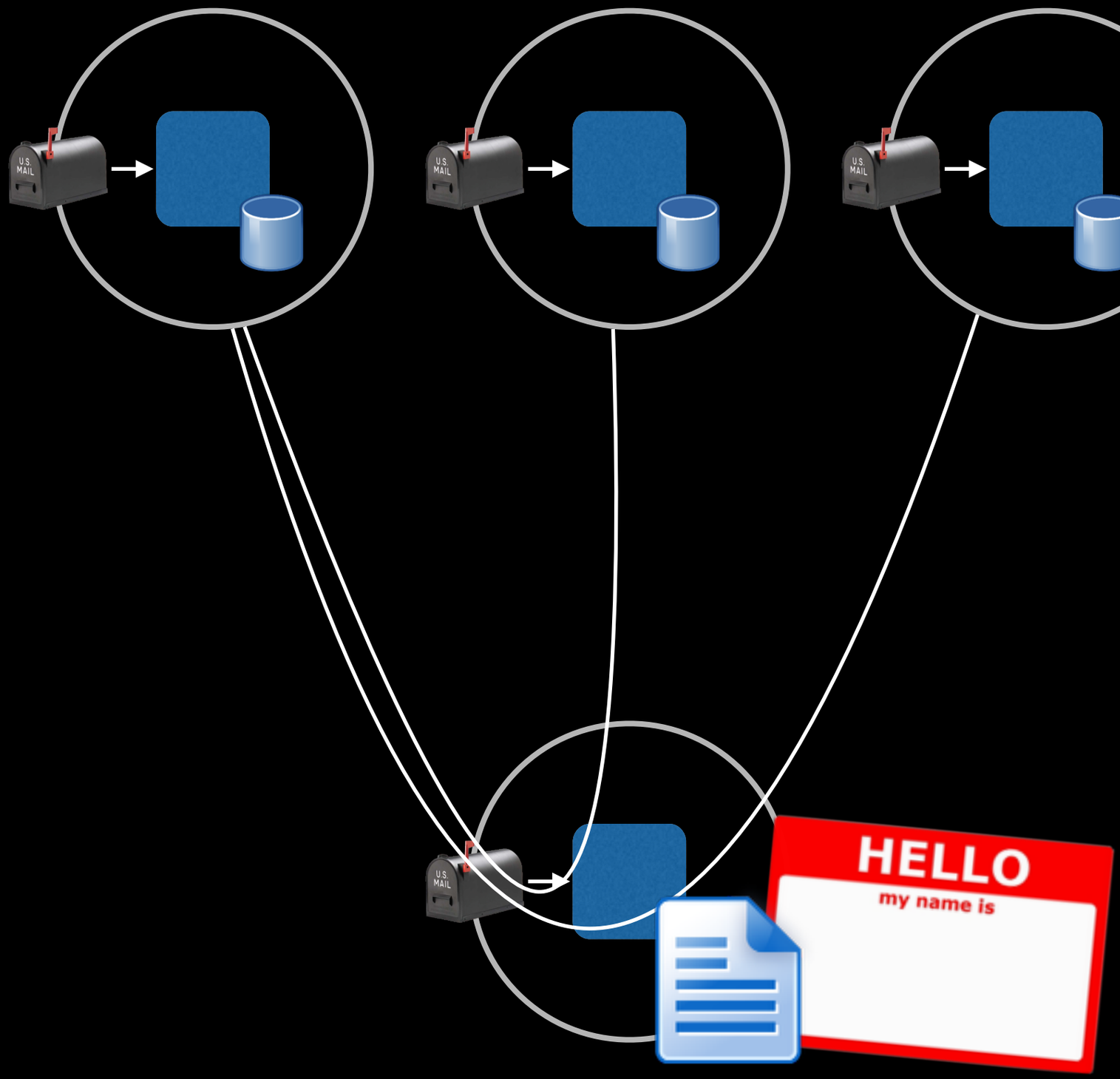

HELLO
my name is

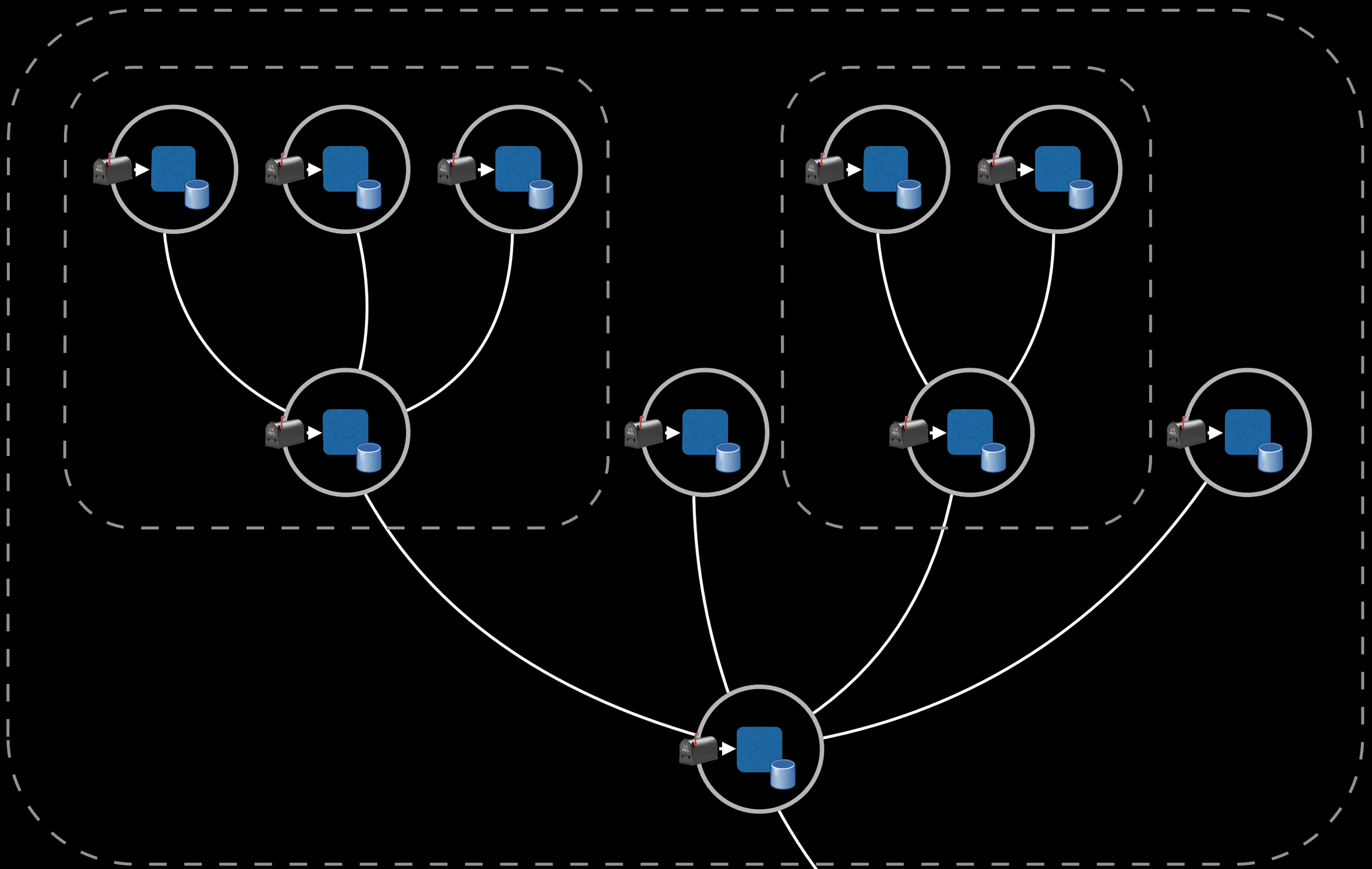
?


U.S. MAIL
U.S. MAIL
U.S. MAIL
U.S. MAIL


HELLO
my name is

?


U.S. MAIL
U.S. MAIL
U.S. MAIL
U.S. MAIL
HELLO
my name is



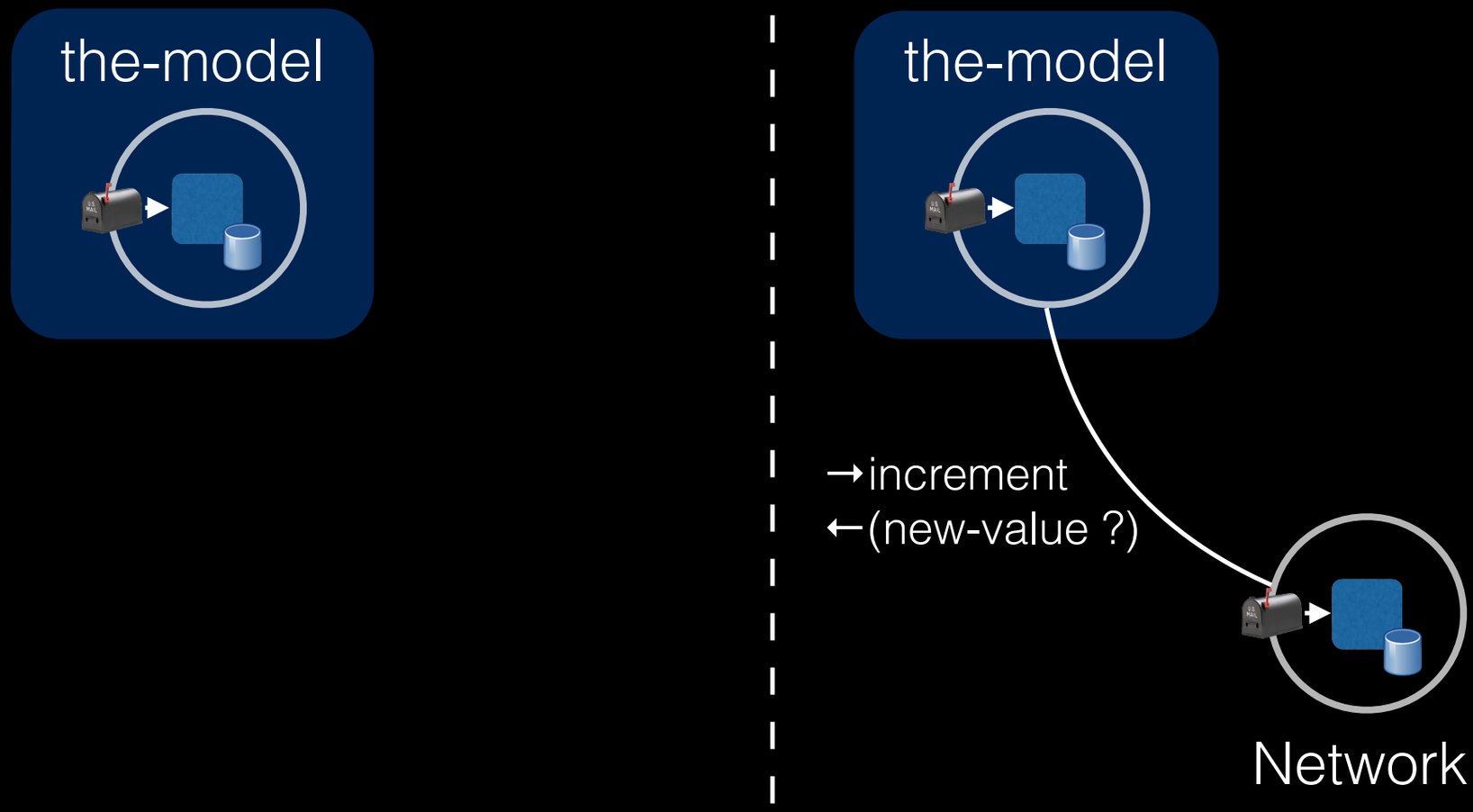
Code!

```
#lang minimart

(struct my-message (topic body) #:prefab)

(actor #:name symbolic-name

       #:state [current-topic "apples"]
       #:state [foo (+ 1 (* 2 3))]
       #:state [bar? #f]

       ;; Might produce messages on the current topic
       (advertise (my-message current-topic ?))

       ;; Want to hear messages on the current topic
       (subscribe (my-message current-topic ($ msg-body))
         (printf "Got message ~a\n" msg-body)
         #:update [foo (+ foo 1)]
         ...)

       (observe-advertisers (my-message ($ topic) ?)
         #:name all-topics
         #:set topic
         (printf "All available topics: ~a\n" all-topics))

       (observe-subscribers ...))
```

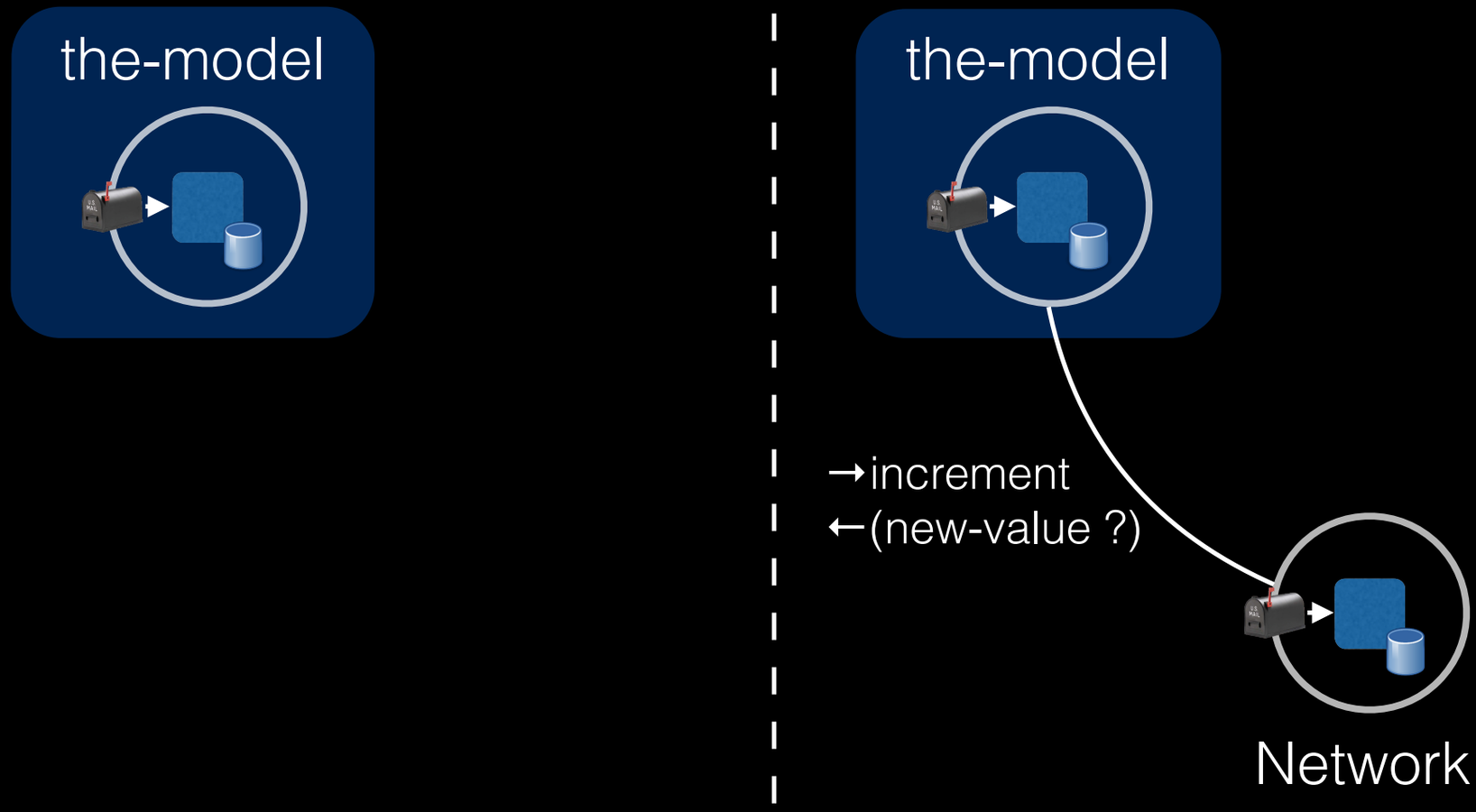
Model/View

Event Broadcasting,
Observable/Observer

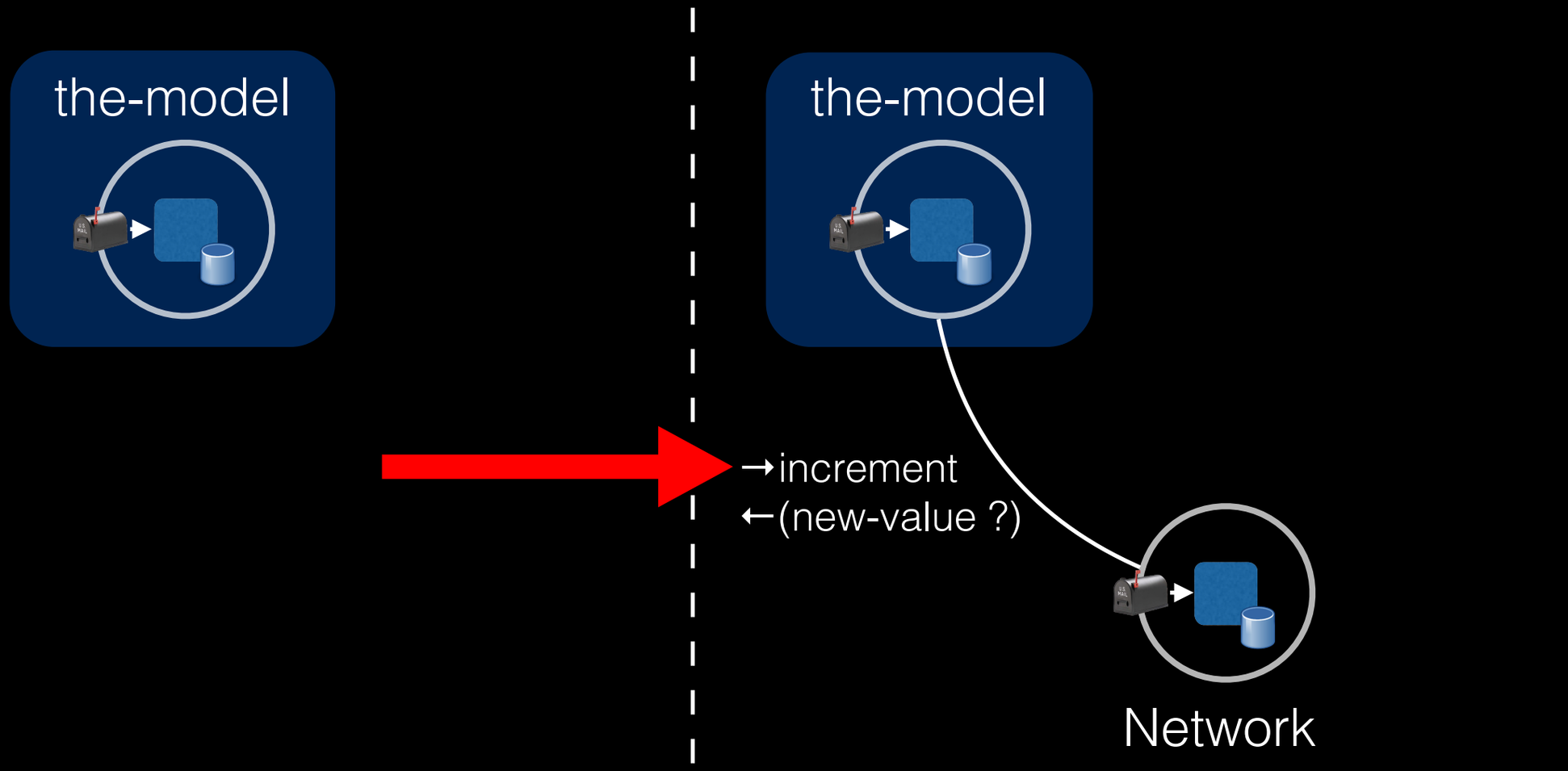
```
(actor #:name the-model
       #:state [value 400]

       (advertise (list 'new-value ?))

       (subscribe 'increment
         #:update [value (+ value 1)]
         (send (list 'new-value value))))
```



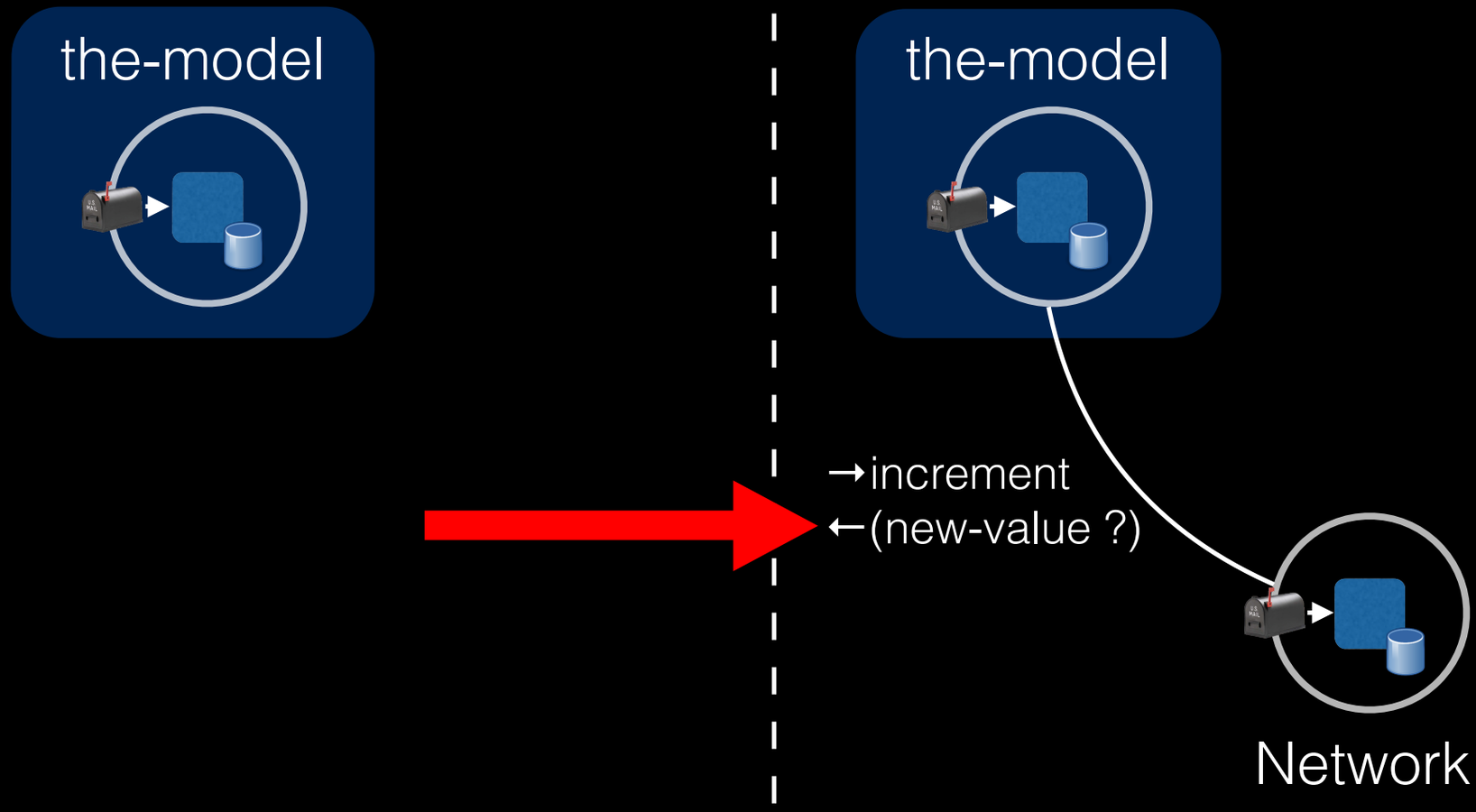
```
(actor #:name the-model
       #:state [value 400]

       (advertise (list 'new-value ?))

       (subscribe 'increment
         #:update [value (+ value 1)]
         (send (list 'new-value value))))
```



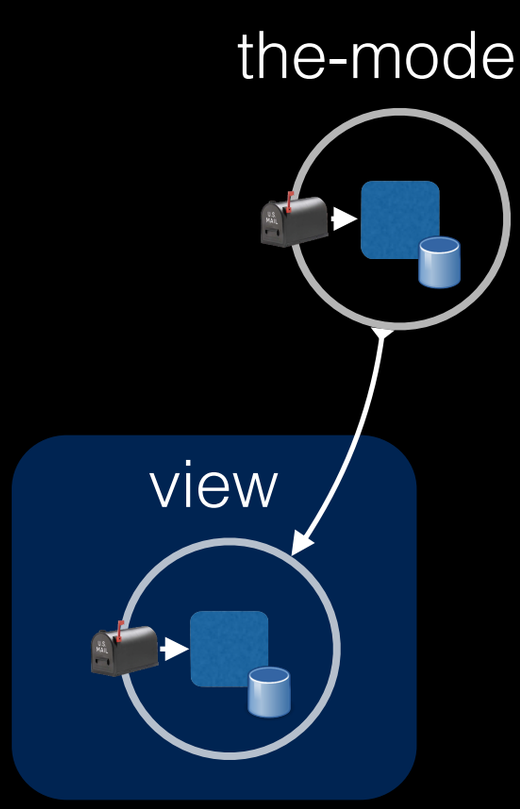
```
(actor #:name the-model
       #:state [value 400]

       (advertise (list 'new-value ?))

       (subscribe 'increment
         #:update [value (+ value 1)]
         (send (list 'new-value value))))
```

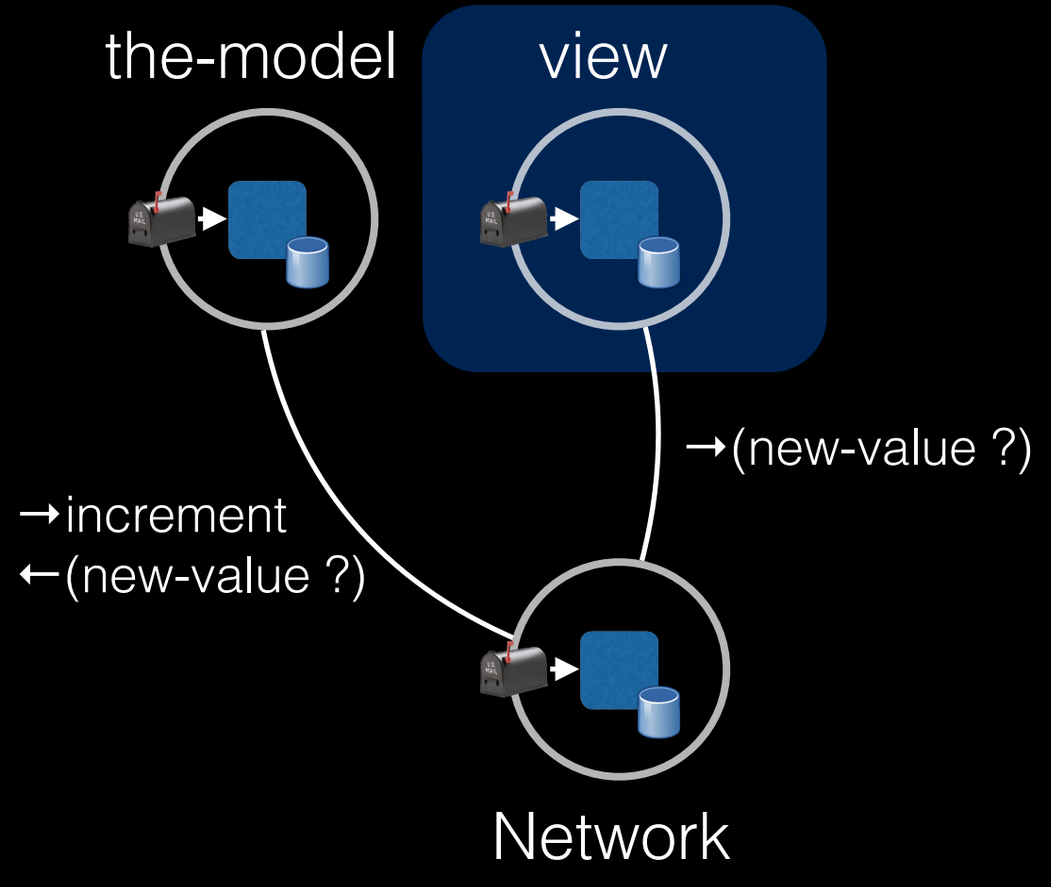


```
(actor #:name the-model
       #:state [value 400]

       (advertise (list 'new-value ?))

       (subscribe 'increment
         #:update [value (+ value 1)]
         (send (list 'new-value value))))
```

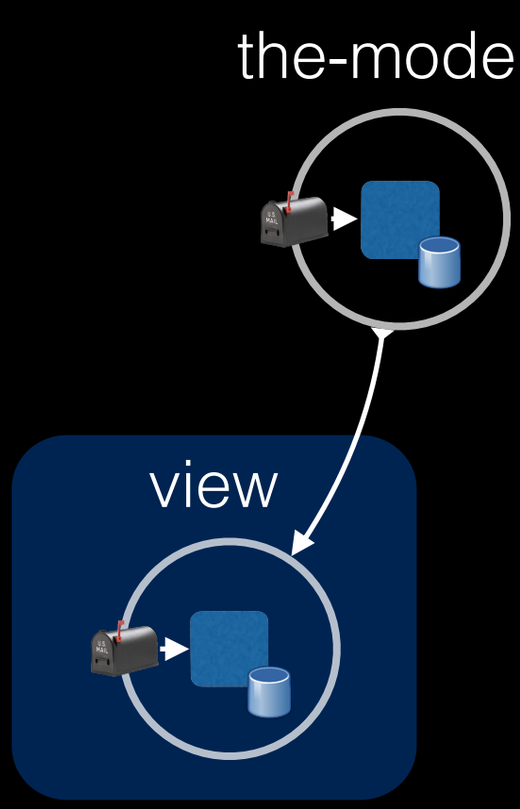


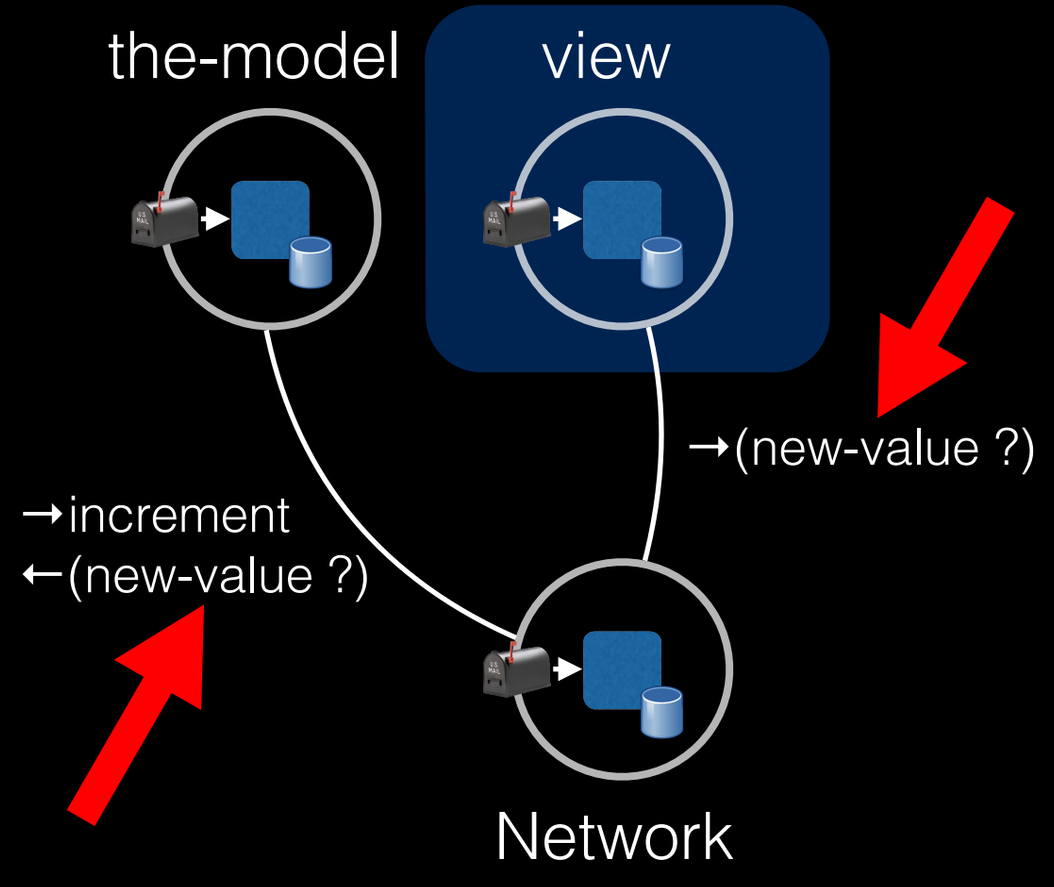
```
(actor #:name view

       (subscribe (list 'new-value ($ the-value))
         (printf "View saw a new value: ~a\n" the-value)))
```



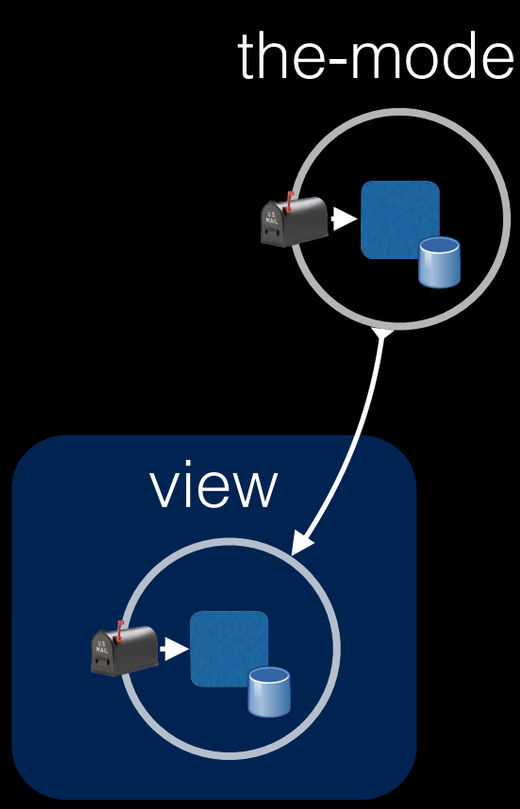


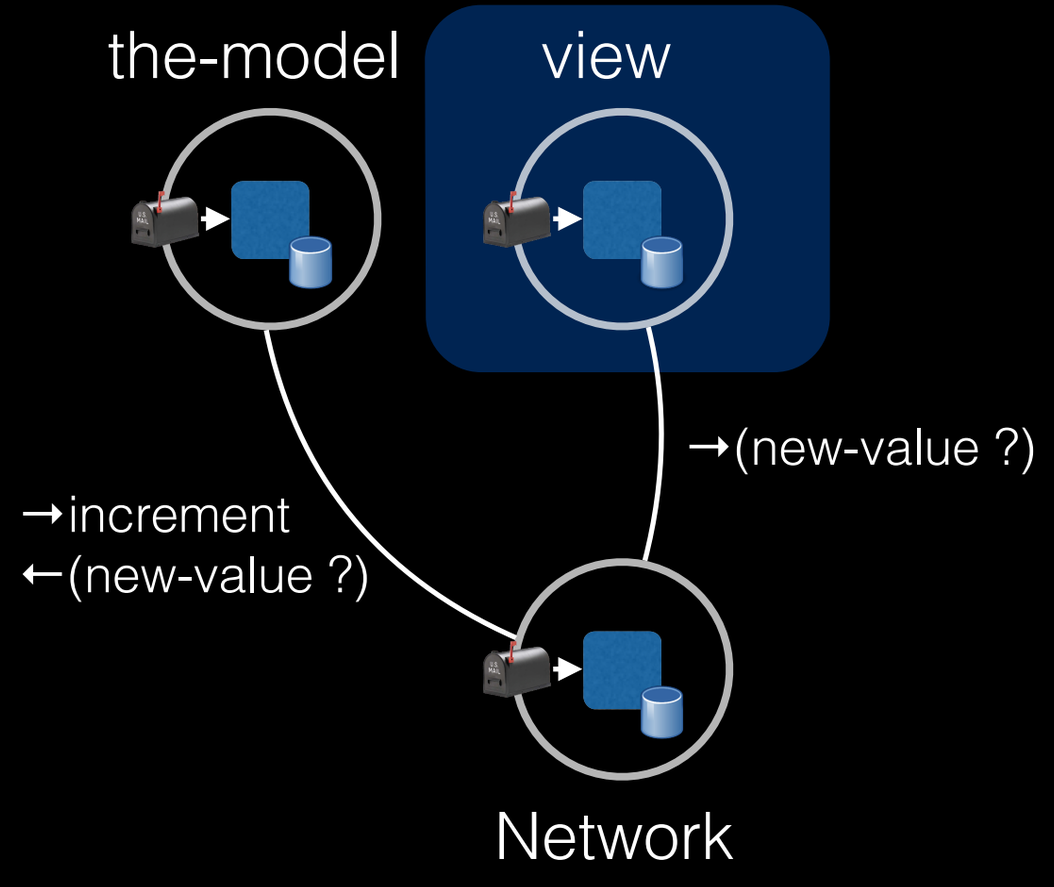
```
(actor #:name view

       (subscribe (list 'new-value ($ the-value))
         (printf "View saw a new value: ~a\n" the-value)))
```



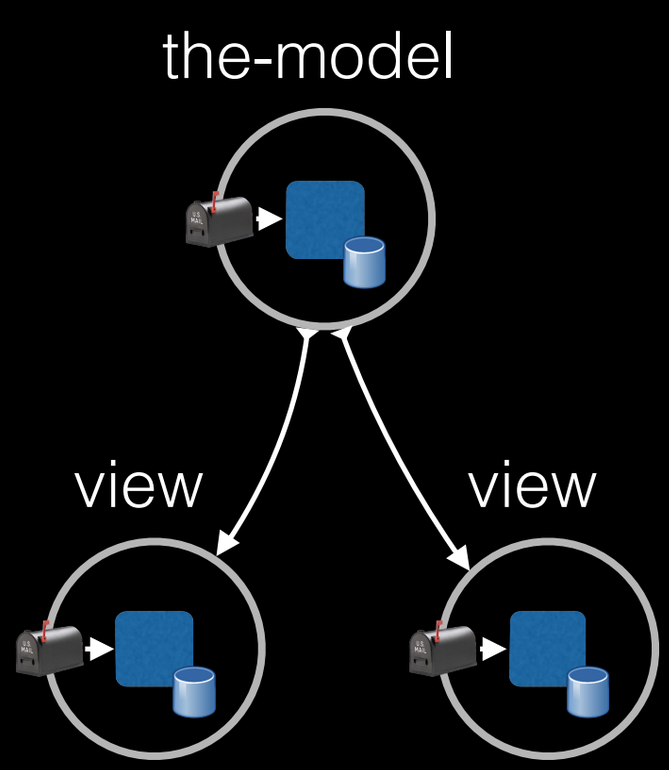


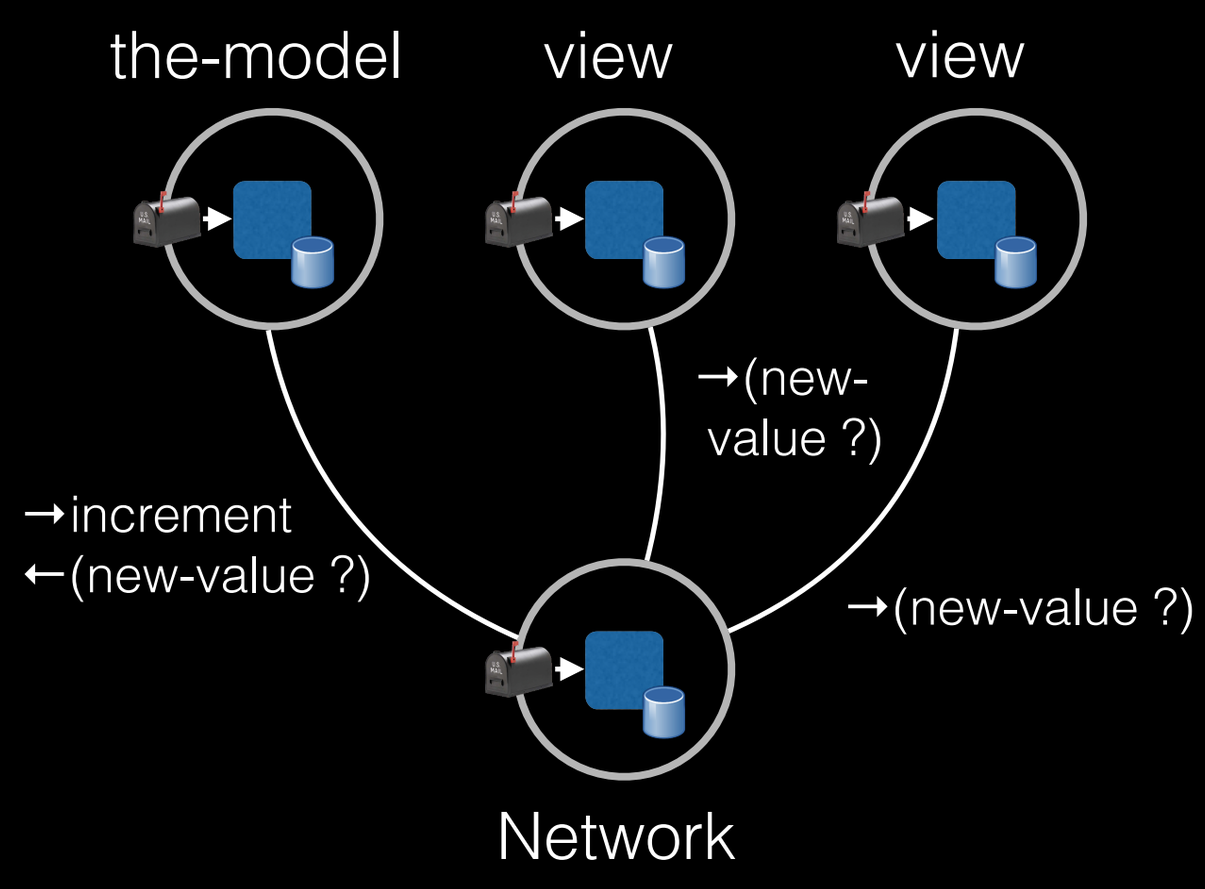
```
(actor #:name view

       (subscribe (list 'new-value ($ the-value))
         (printf "View saw a new value: ~a\n" the-value)))
```





```
(actor #:name view

       (subscribe (list 'new-value ($ the-value))
         (printf "View saw a new value: ~a\n" the-value)))
```

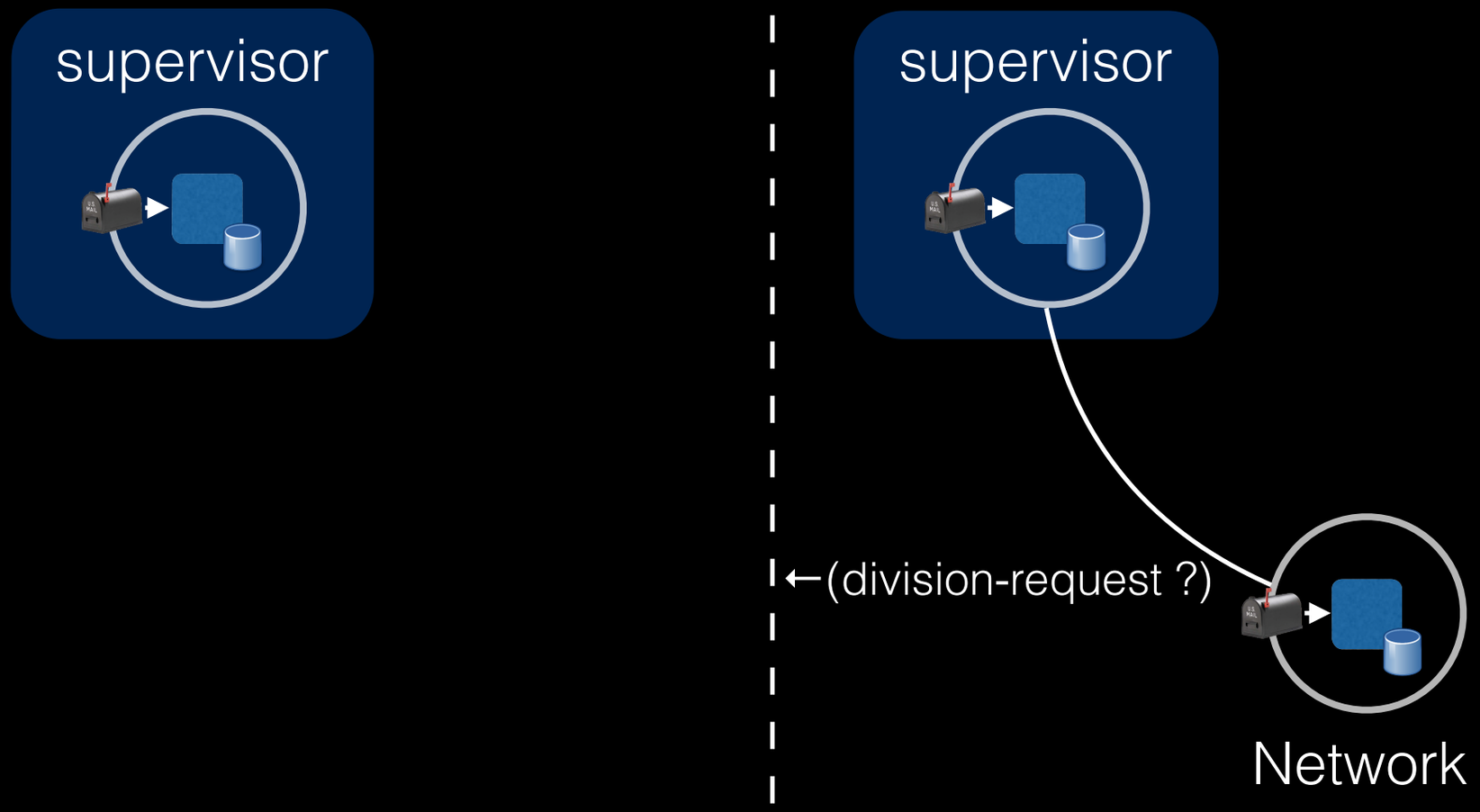




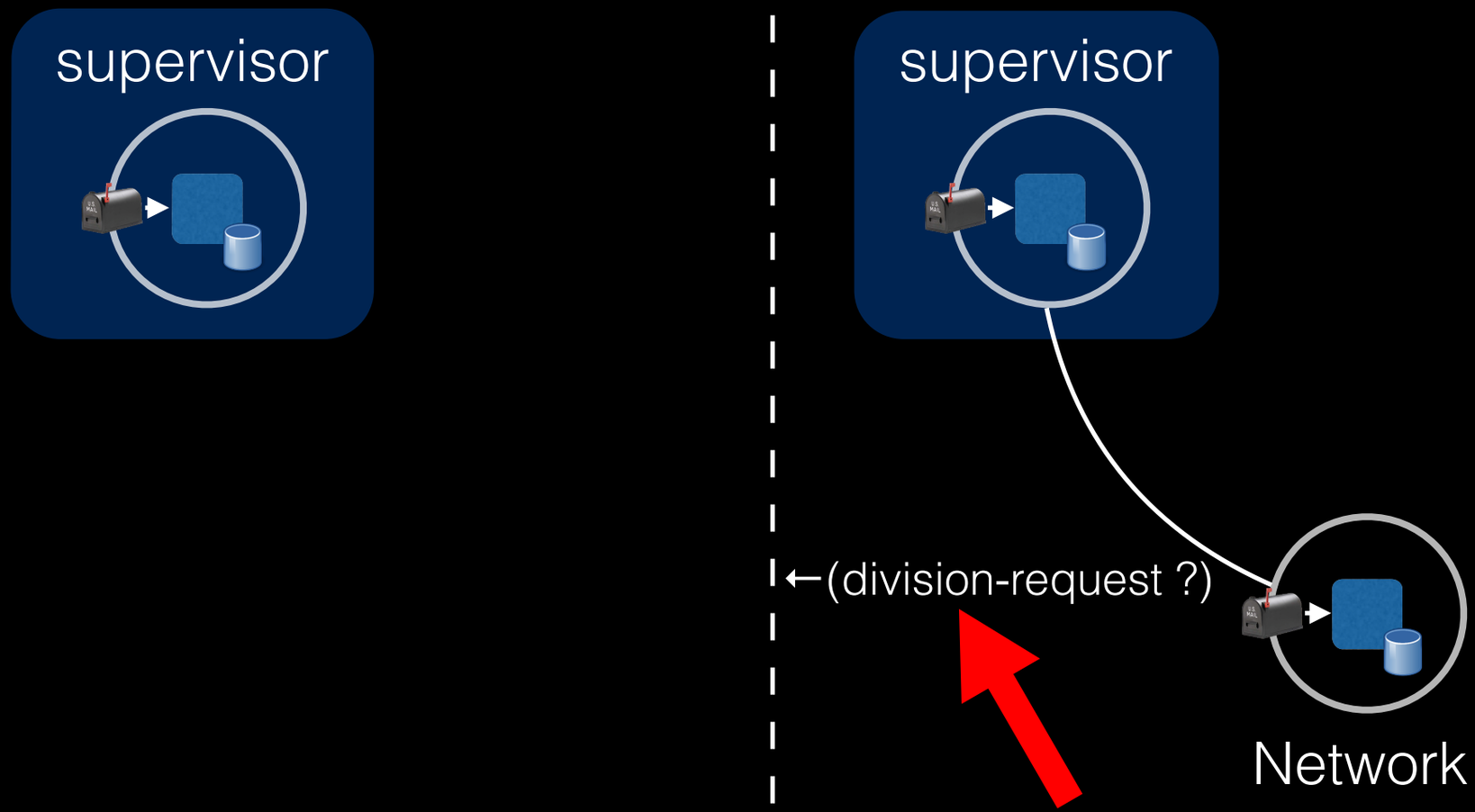
Supervisors

Crash/exit signaling
Readiness notification

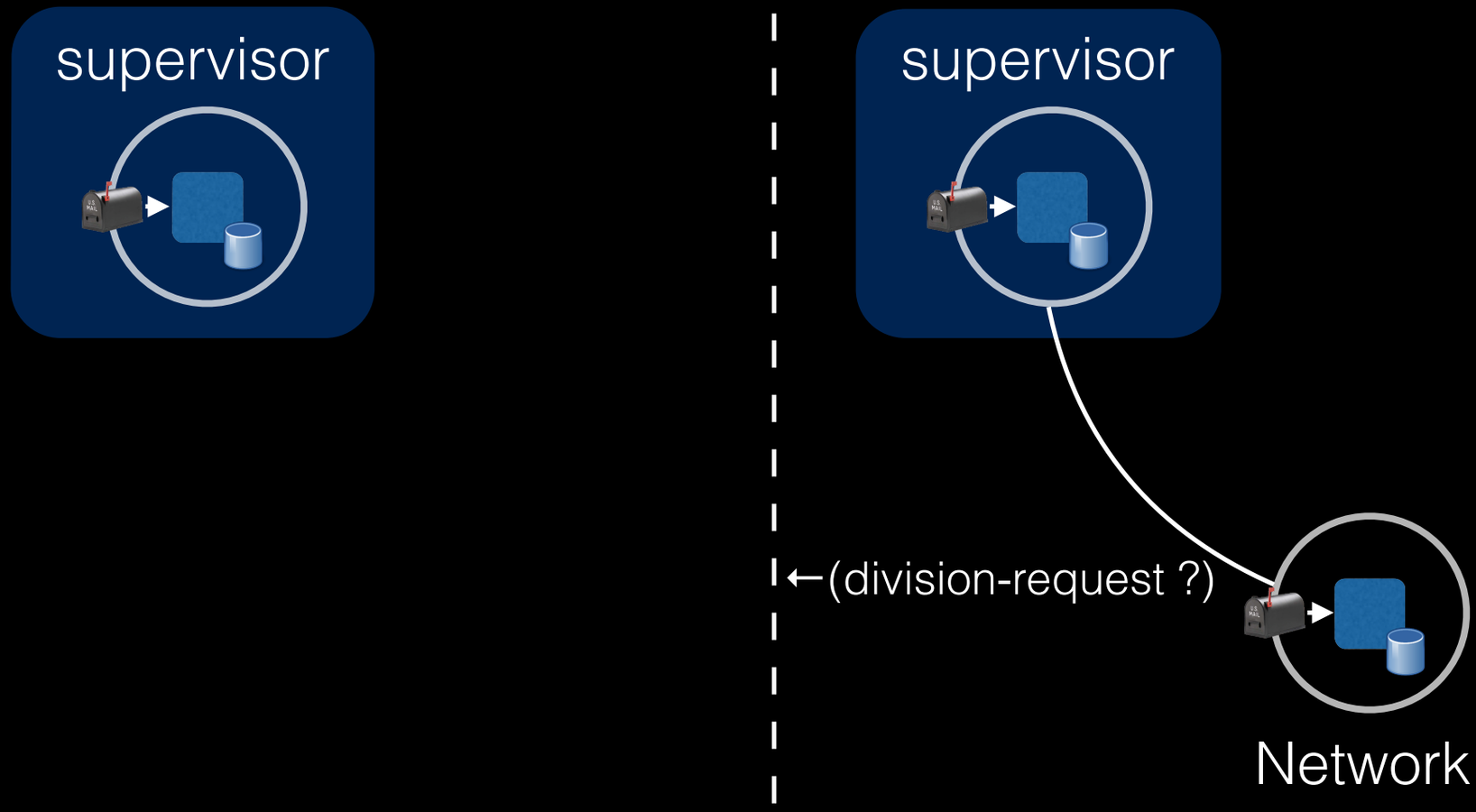
```
(actor #:name supervisor
       (observe-subscribers (list 'division-request ?)
         #:presence server-running?
         (when (not server-running?)
           (printf "SUPERVISOR: Starting server!\n")
           (spawn-server))))
```



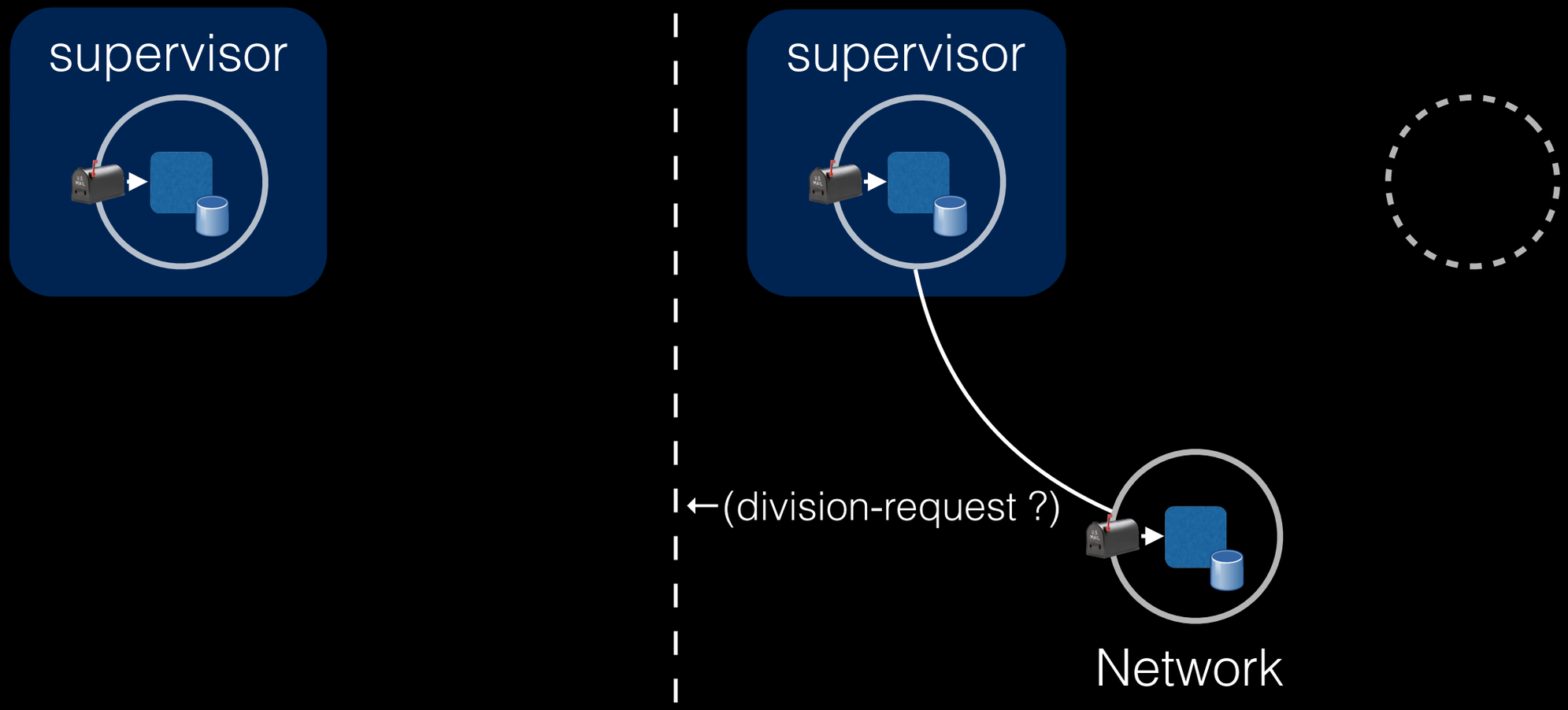
```
(actor #:name supervisor
       (observe-subscribers (list 'division-request ?)
         #:presence server-running?
         (when (not server-running?)
           (printf "SUPERVISOR: Starting server!\n")
           (spawn-server))))
```



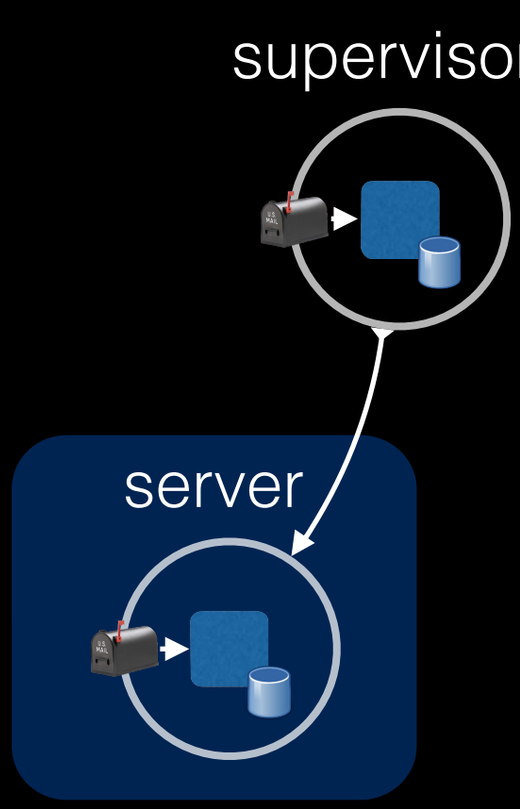
```
(actor #:name supervisor
       (observe-subscribers (list 'division-request ?)
         #:presence server-running?
         (when (not server-running?)
           (printf "SUPERVISOR: Starting server!\n")
           (spawn-server))))
```

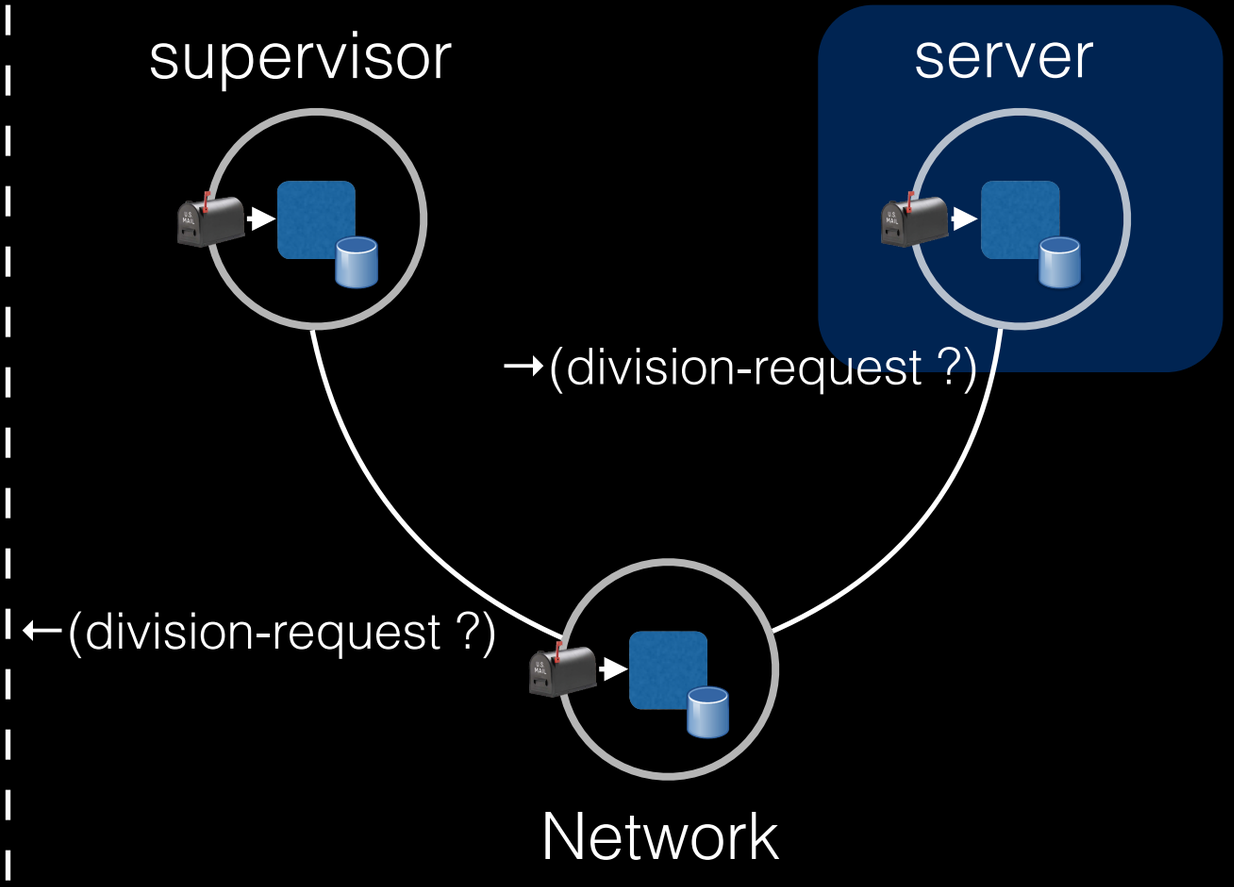


```
(actor #:name supervisor
       (observe-subscribers (list 'division-request ?)
         #:presence server-running?
         (when (not server-running?)
           (printf "SUPERVISOR: Starting server!\n")
           (spawn-server))))
```

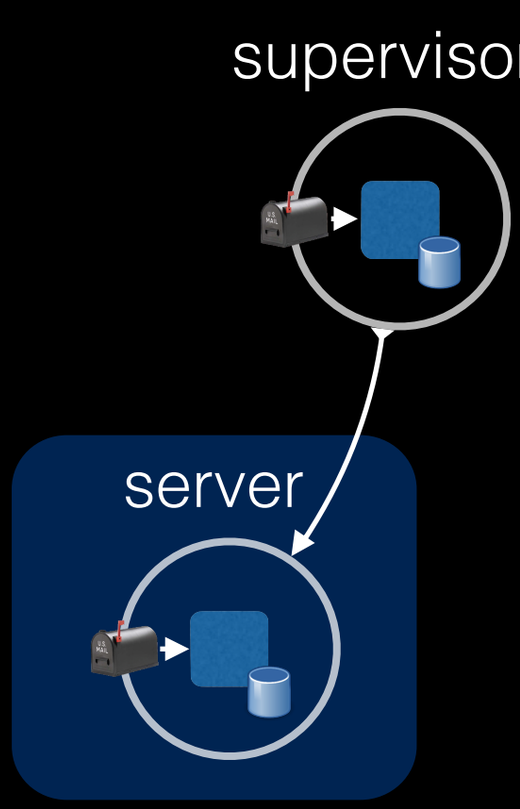


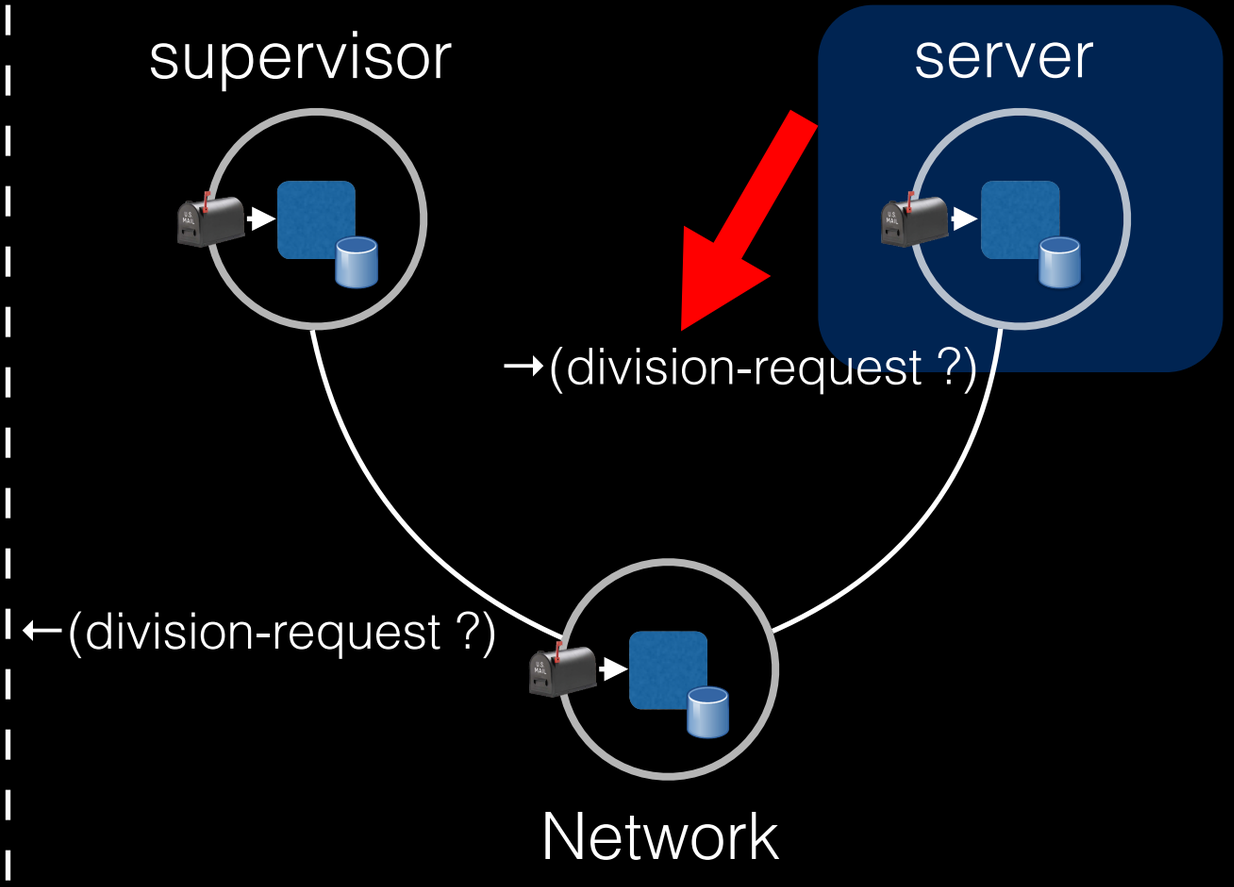
```
(define (spawn-server)
  (actor #:name server
         (subscribe (list 'division-request ($ denominator))
           (printf "SERVER: got request\n")
           ...)))
```



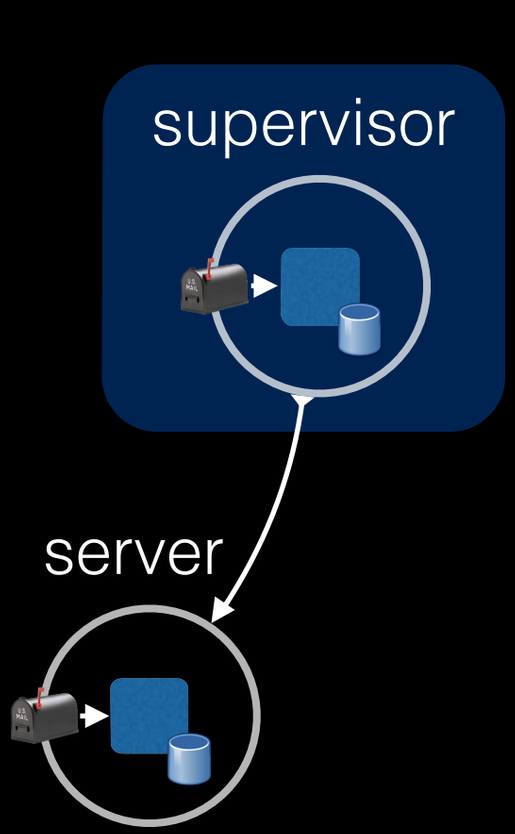


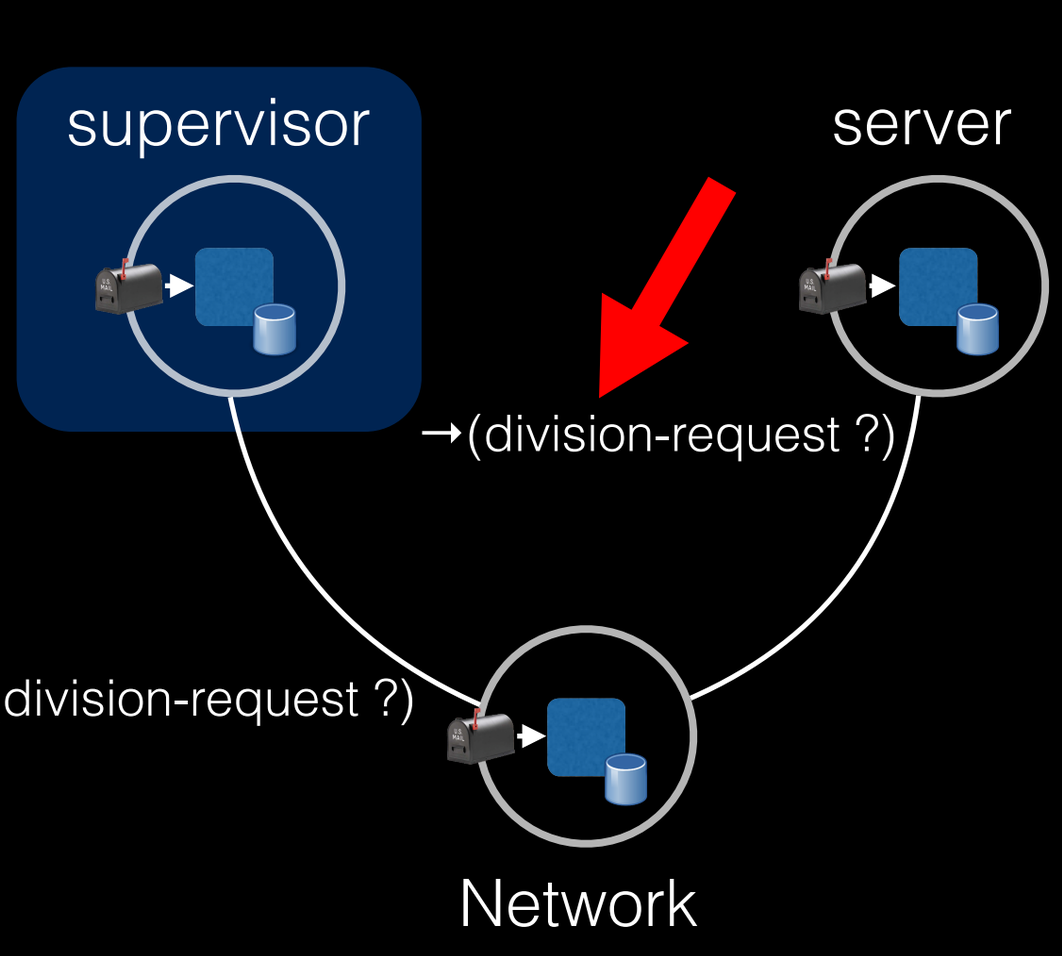
```
(define (spawn-server)
  (actor #:name server
         (subscribe (list 'division-request ($ denominator))
           (printf "SERVER: got request\n")
           ...)))
```



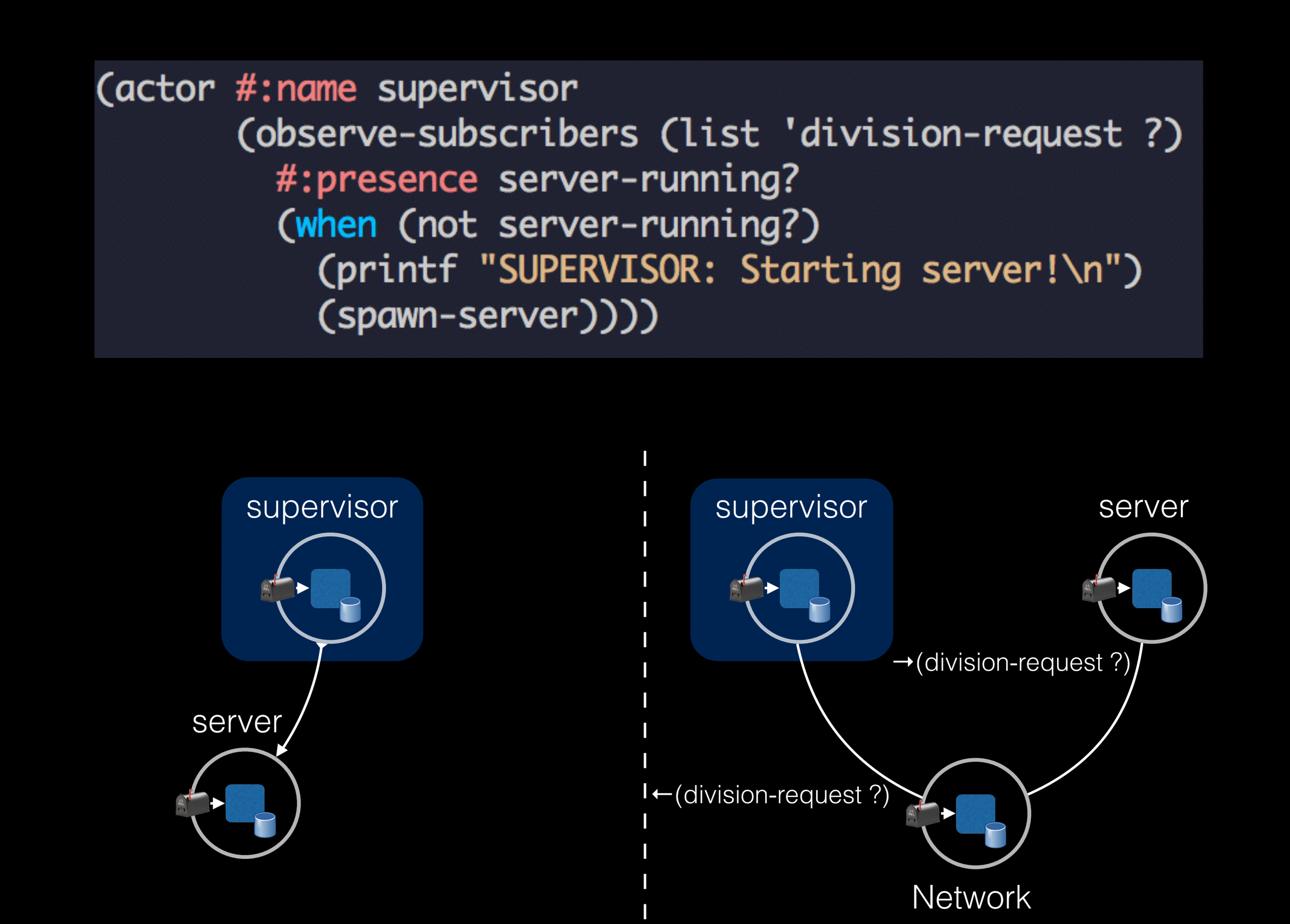


```
(actor #:name supervisor
       (observe-subscribers (list 'division-request ?)
         #:presence server-running?
         (when (not server-running?)
           (printf "SUPERVISOR: Starting server!\n")
           (spawn-server))))
```

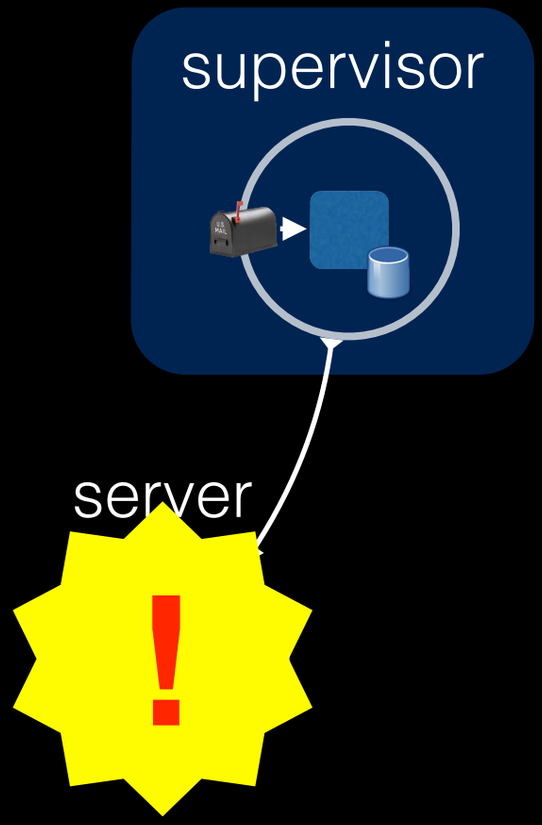


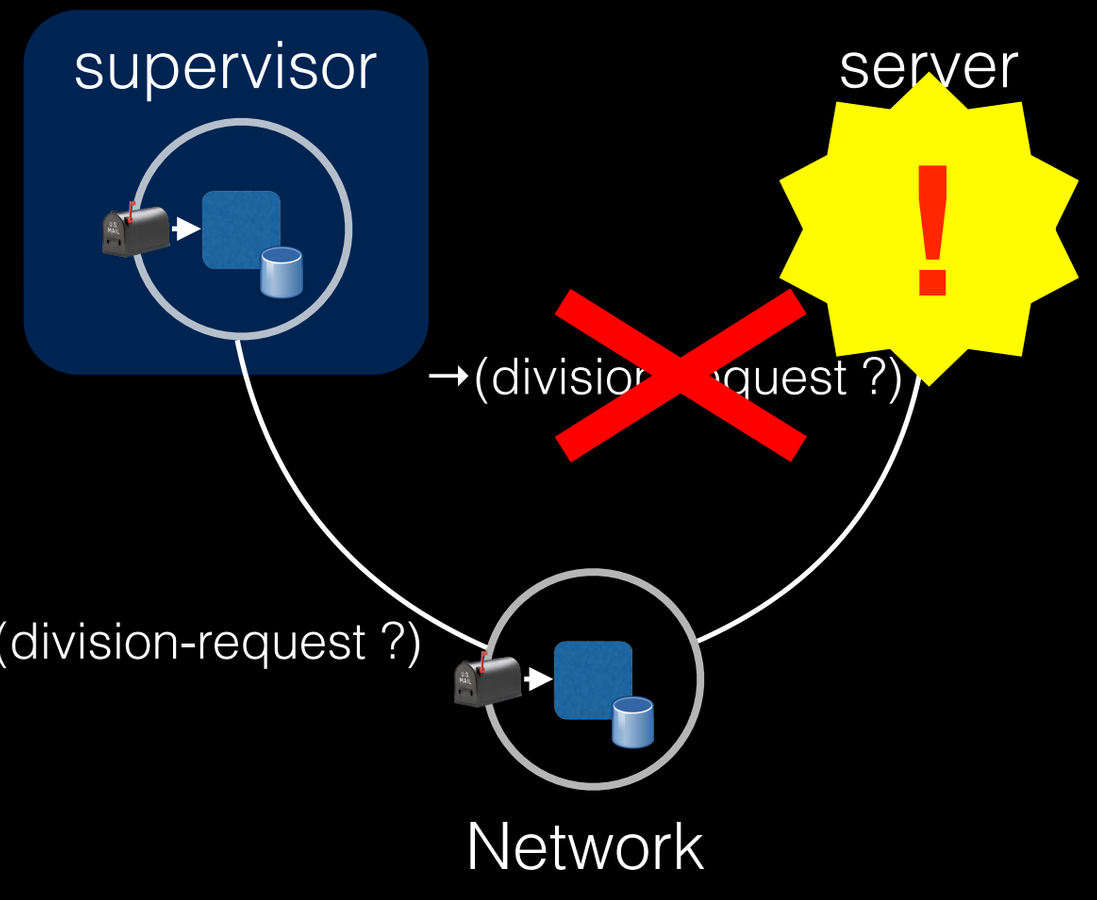


```
(actor #:name supervisor
       (observe-subscribers (list 'division-request ?)
         #:presence server-running?
         (when (not server-running?)
           (printf "SUPERVISOR: Starting server!\n")
           (spawn-server))))
```



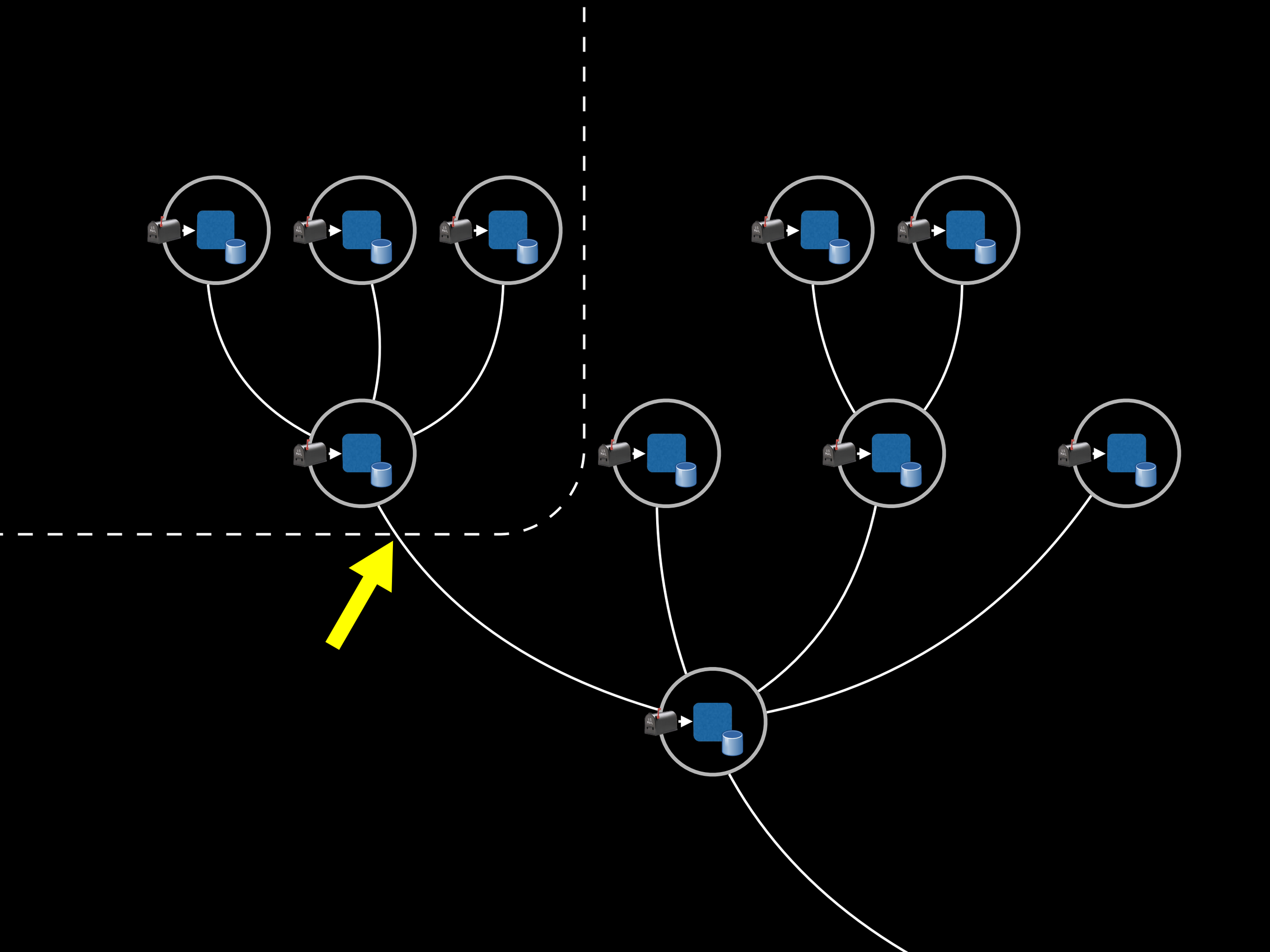
```
(actor #:name supervisor
       (observe-subscribers (list 'division-request ?)
         #:presence server-running?
         (when (not server-running?)
           (printf "SUPERVISOR: Starting server!\n")
           (spawn-server))))
```





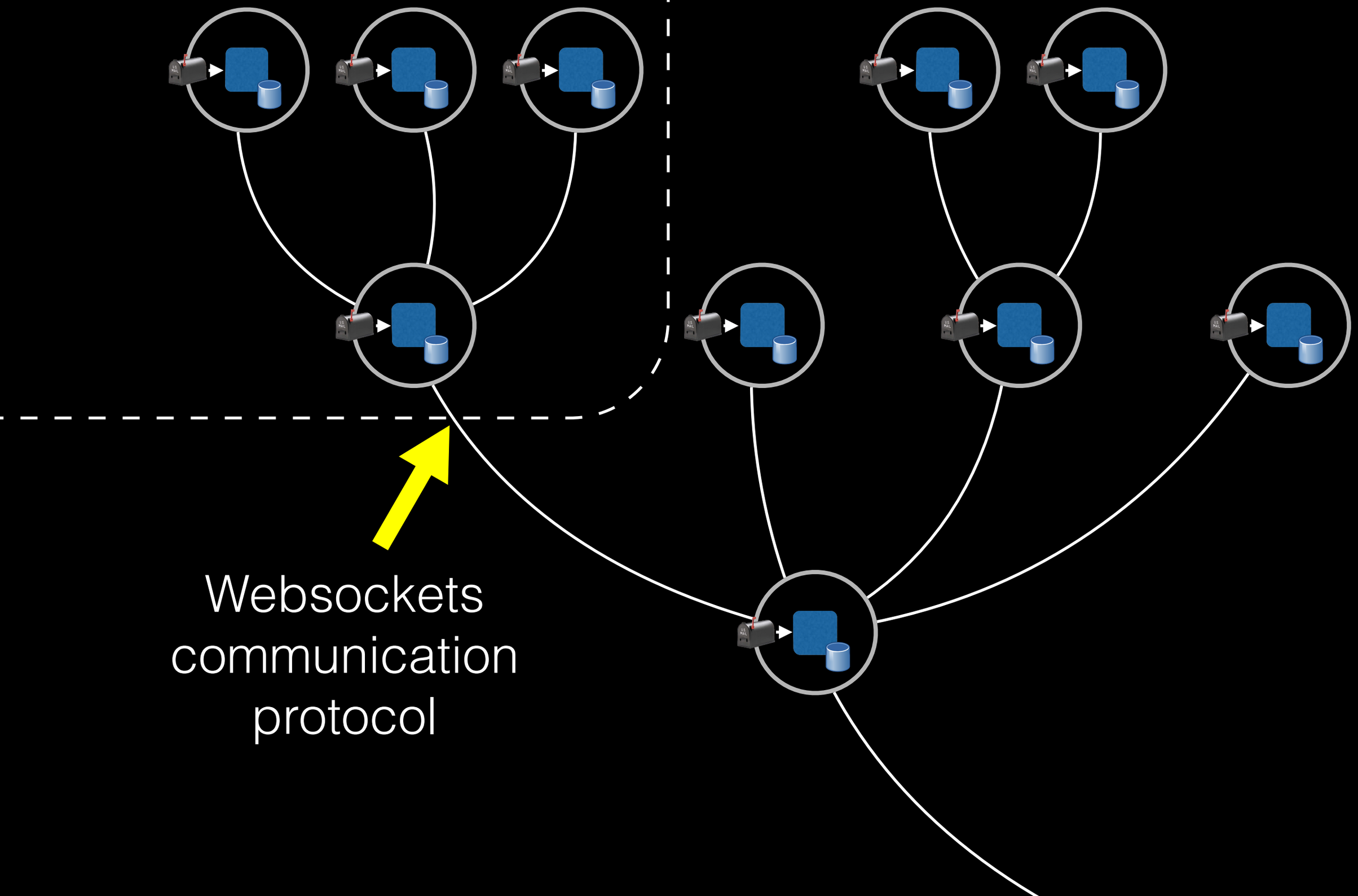
Distribution

Minimart in the Browser



JS-marketplace

Minimart



Websockets
communication
protocol

#lang minimart

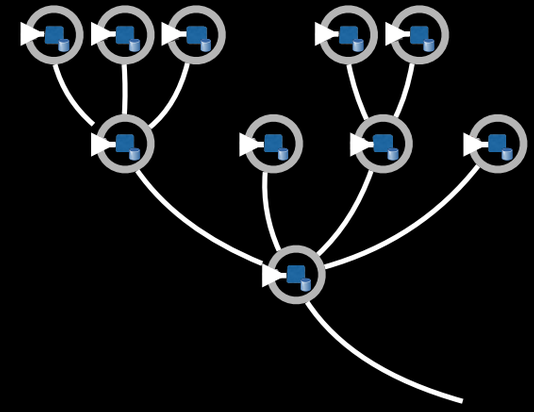
Publish / Subscribe conversations

```
(actor #:name A
  (subscribe (pat ...)
      ... handler ...))
```

Observe others' subscriptions

```
(actor #:name B
  (observe-subscribers
      (pat ...)
    #:presence v
     ... handler ...))
```

Grouping & layering



raco pkg install minimart

https://github.com/tonyg/minimart

https://github.com/tonyg/js-marketplace