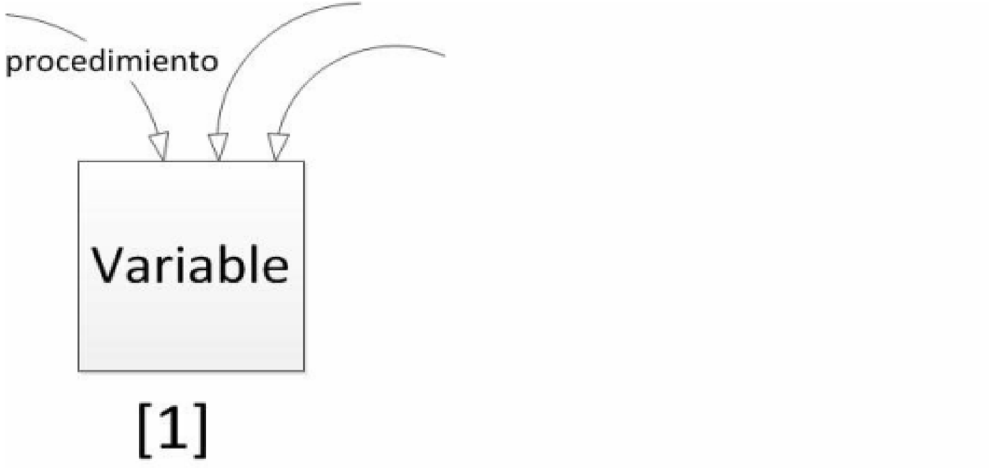
Concurrencia sin dolor

- Ignacio Blasco López
- Sebastián Ortega Torres

512

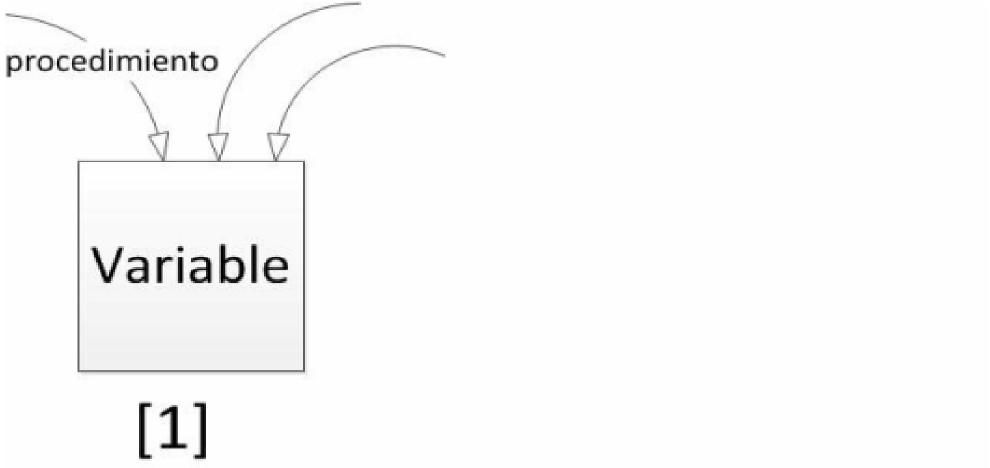
Imperativo vs Funcional

Imperativo vs Funcional



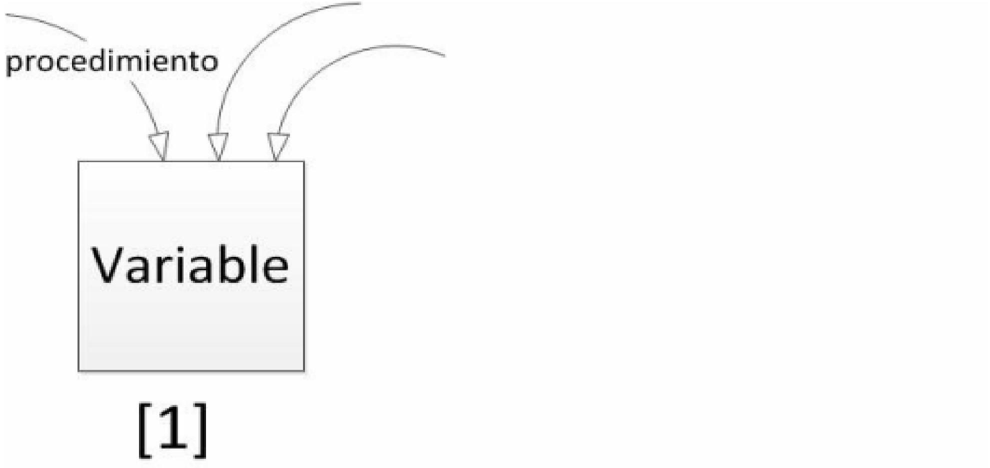
Imperativo vs Funcional

- Modelo Imperativo



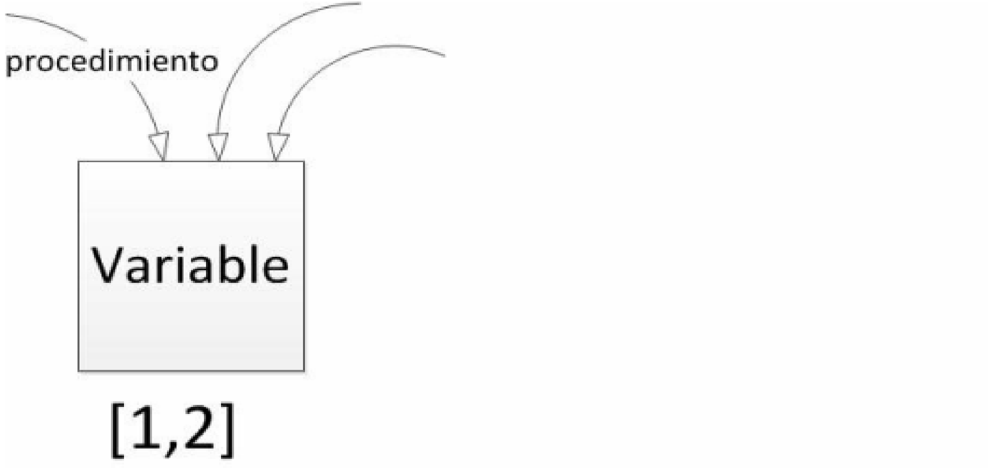
Imperativo vs Funcional

- Modelo Imperativo
  - Modificar variables



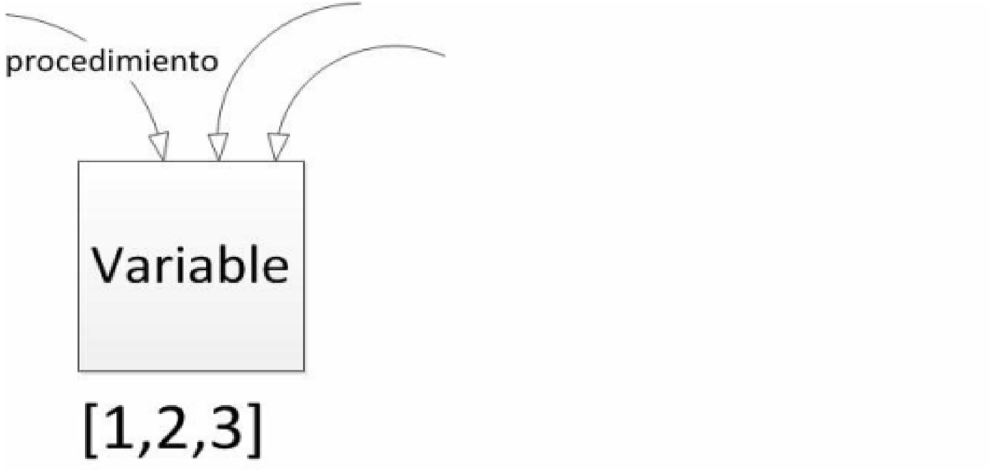
Imperativo vs Funcional

- Modelo Imperativo
  - Modificar variables



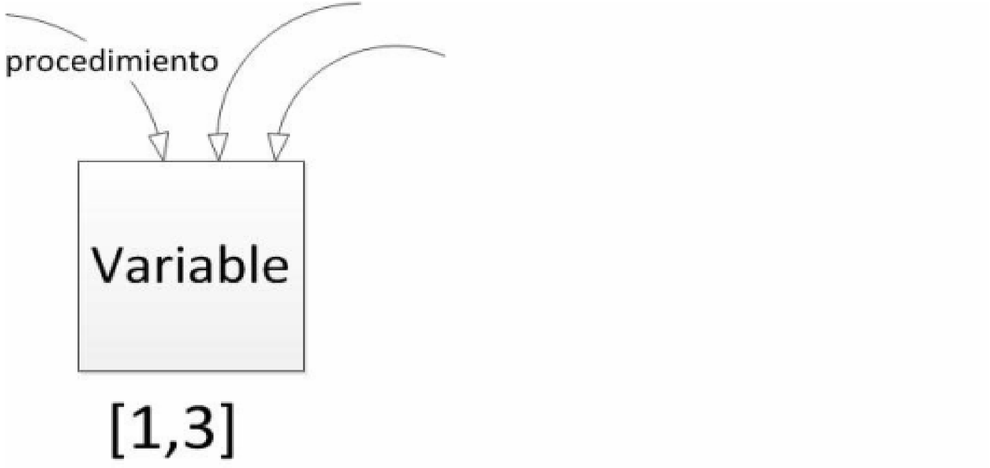
Imperativo vs Funcional

- Modelo Imperativo
  - Modificar variables



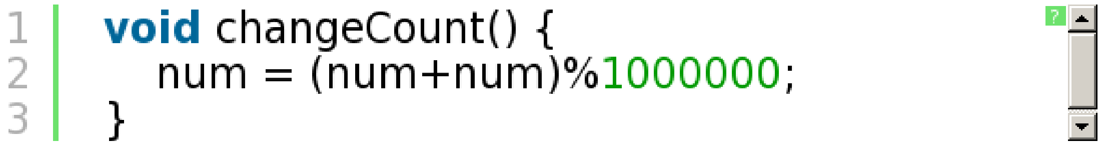
Imperativo vs Funcional

- Modelo Imperativo
  - Modificar variables

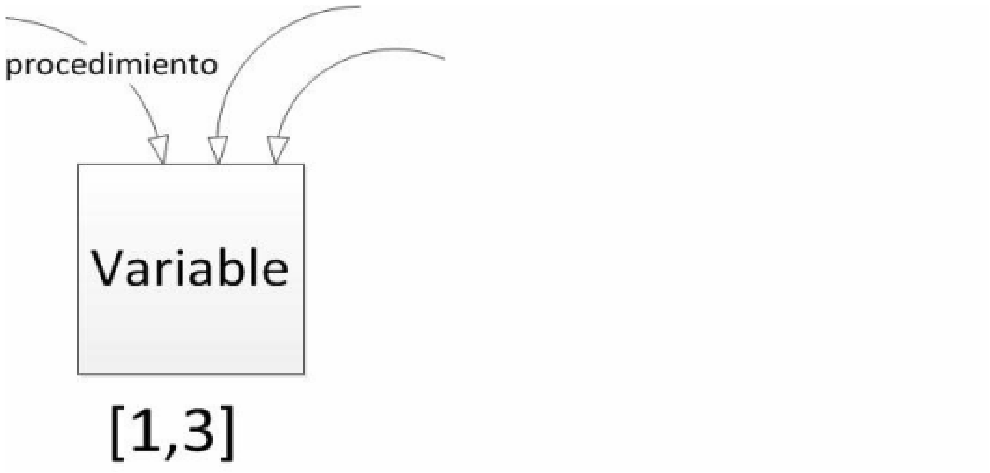


Imperativo vs Funcional

- Modelo Imperativo
  - Modificar variables

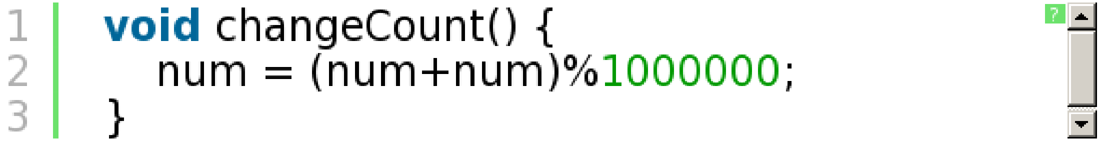


```
void changeCount() {
    num = (num+num)%1000000;
}
```

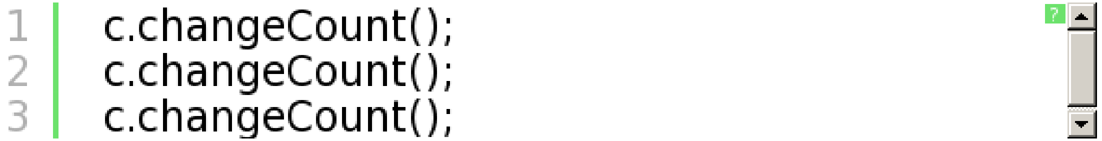


Imperativo vs Funcional

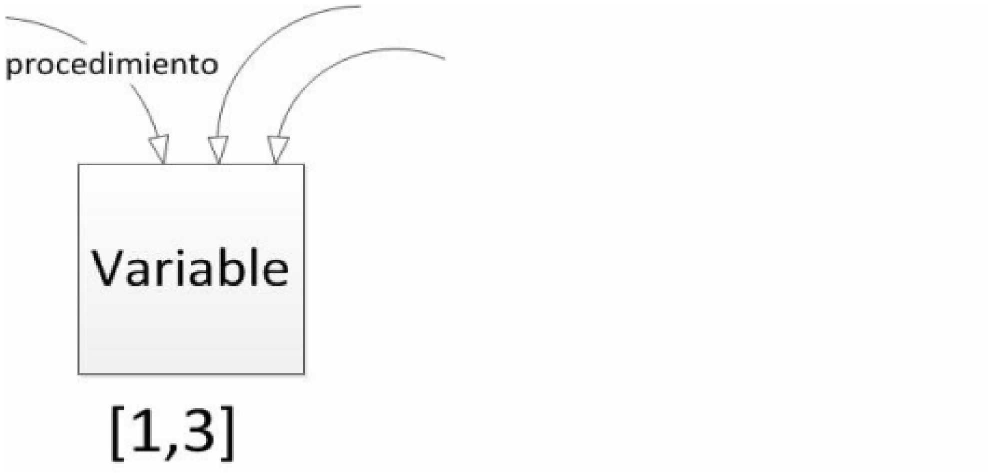
- Modelo Imperativo
  - Modificar variables



```
void changeCount() {
    num = (num+num)%1000000;
}
```

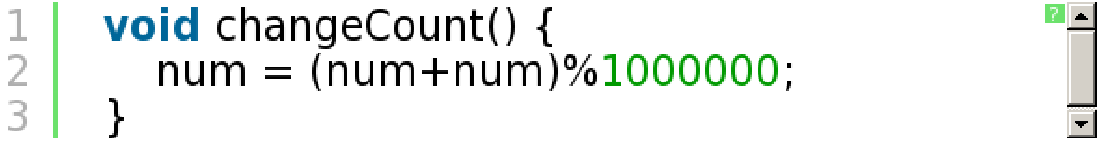


```
c.changeCount();
c.changeCount();
c.changeCount();
```

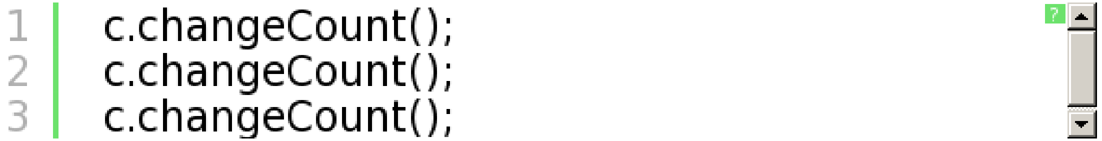


Imperativo vs Funcional

- Modelo Imperativo
  - Modificar variables

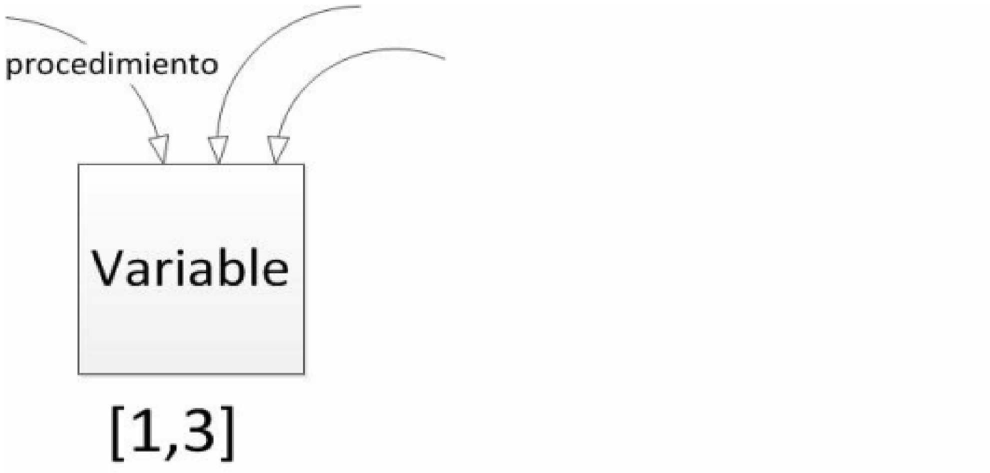


```
void changeCount() {
    num = (num+num)%1000000;
}
```



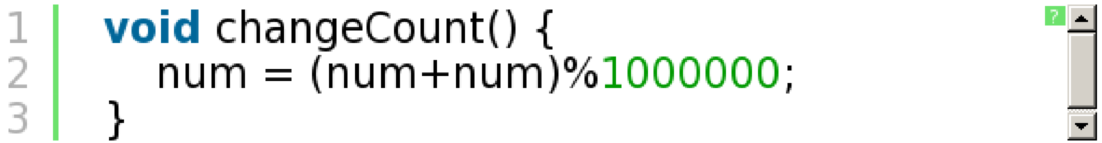
```
c.changeCount();
c.changeCount();
c.changeCount();
```

- Dependencia temporal

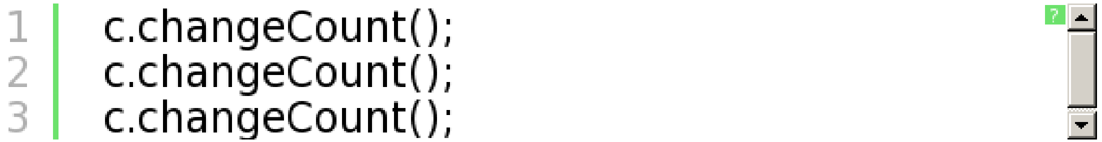


Imperativo vs Funcional

- Modelo Imperativo
    - Modificar variables

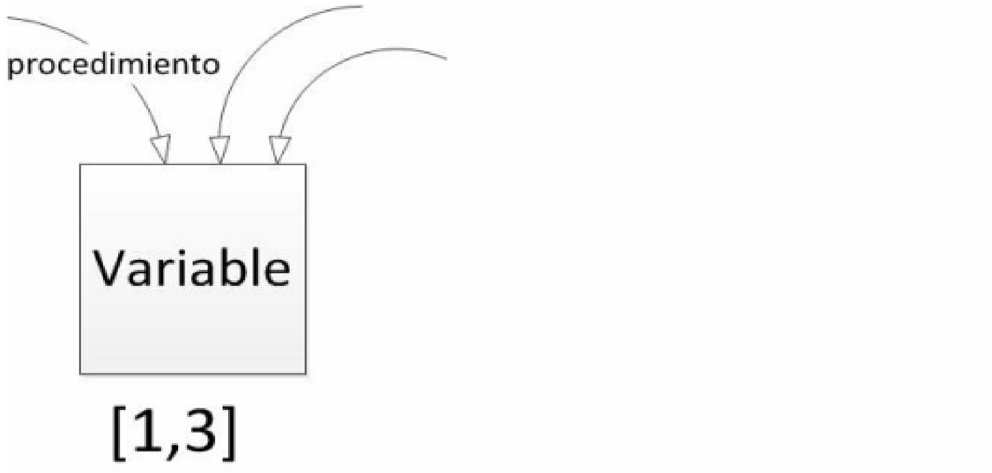


```
void changeCount() {
    num = (num+num)%1000000;
}
```



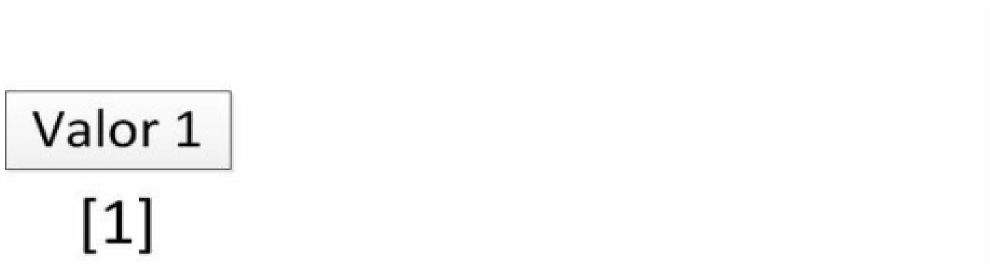
```
c.changeCount();
c.changeCount();
c.changeCount();
```

- Dependencia temporal
- Empeora con concurrencia



Imperativo vs Funcional

Imperativo vs Funcional



```
[1]
```

Imperativo vs Funcional

- Modelo Funcional

[1]

Imperativo vs Funcional

- Modelo Funcional
  - Combinar funciones

[1]

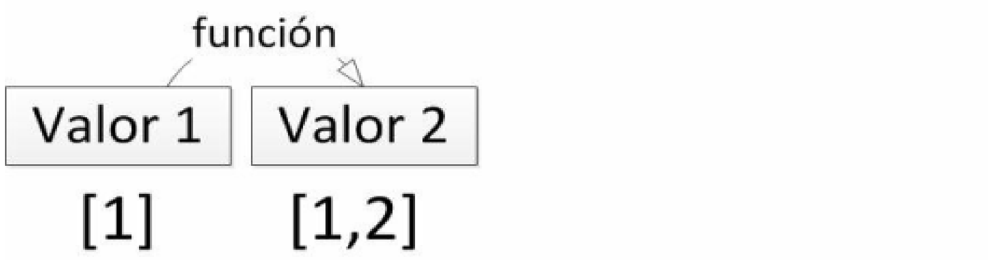
Imperativo vs Funcional

- Modelo Funcional
  - Combinar funciones
  - Generar un valor a partir del siguiente

[1]

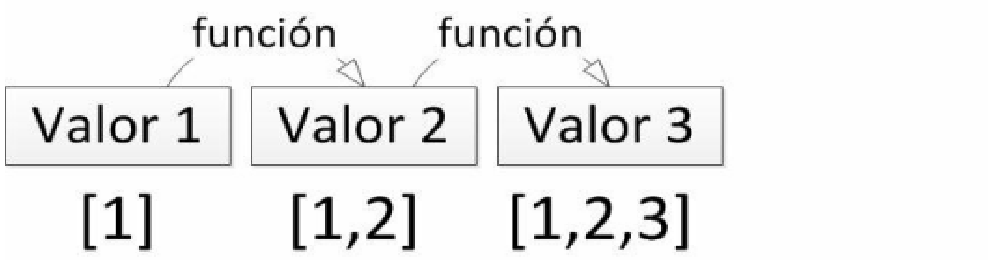
Imperativo vs Funcional

- Modelo Funcional
    - Combinar funciones
    - Generar un valor a partir del siguiente



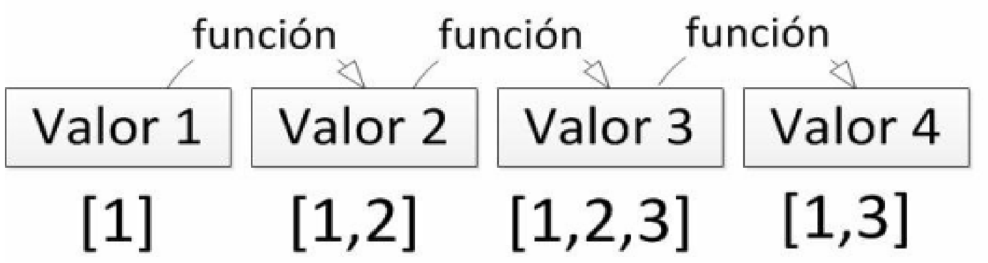
Imperativo vs Funcional

- Modelo Funcional
  - Combinar funciones
  - Generar un valor a partir del siguiente



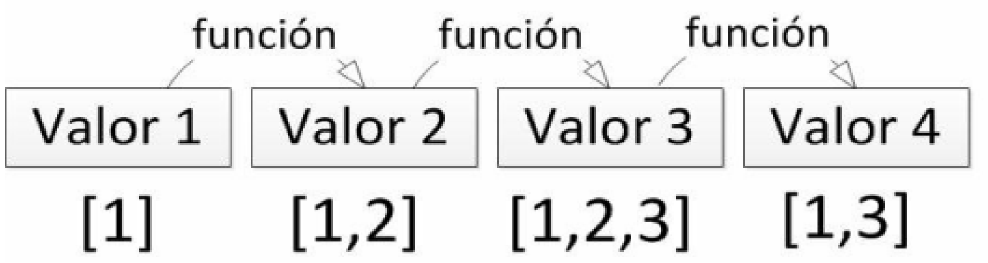
Imperativo vs Funcional

- Modelo Funcional
  - Combinar funciones
  - Generar un valor a partir del siguiente



Imperativo vs Funcional

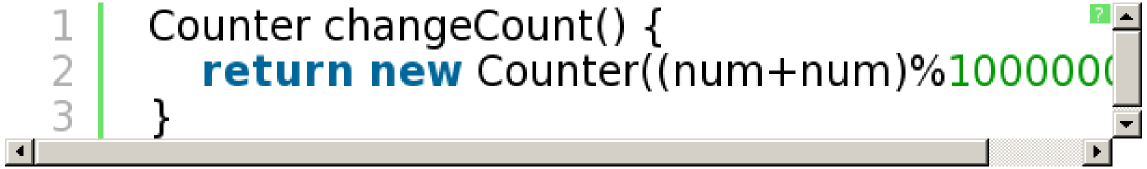
- Modelo Funcional
  - Combinar funciones
  - Generar un valor a partir del siguiente



función función función

Valor 1 Valor 2 Valor 3 Valor 4

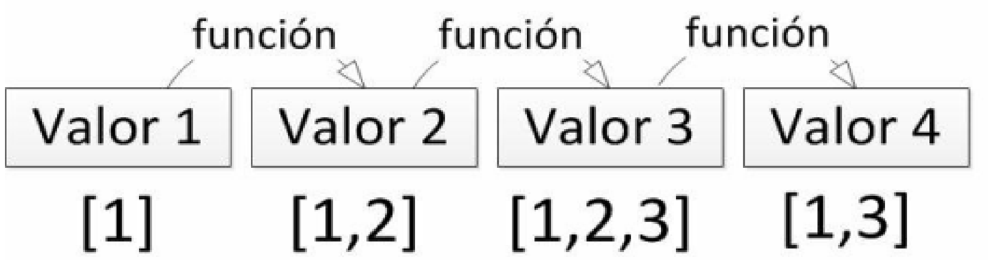
[1] [1,2] [1,2,3] [1,3]

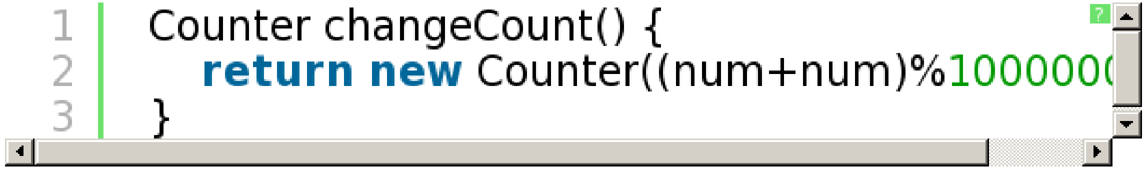


```
Counter changeCount() {
   return new Counter((num+num)%1000000
}
```

Imperativo vs Funcional

- Modelo Funcional
    - Combinar funciones
    - Generar un valor a partir del siguiente



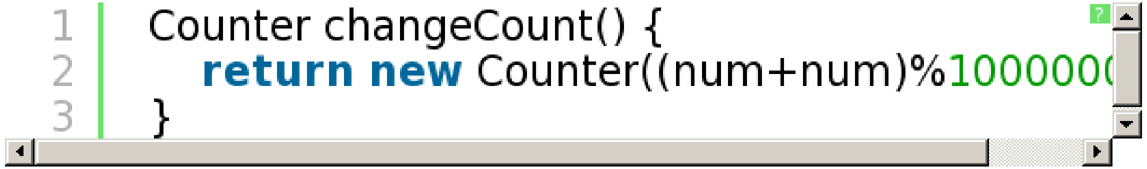


```
Counter changeCount() {
    return new Counter((num+num)%1000000
}
```

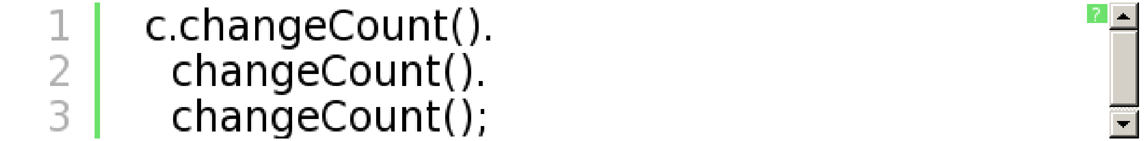
```
c.changeCount().
  changeCount().
  changeCount();
```

Imperativo vs Funcional

- Modelo Funcional
  - Combinar funciones
  - Generar un valor a partir del siguiente

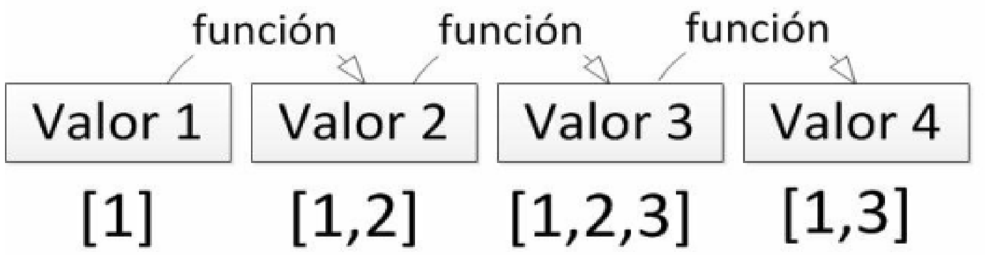


```
Counter changeCount() {
    return new Counter((num+num)%1000000
}
```



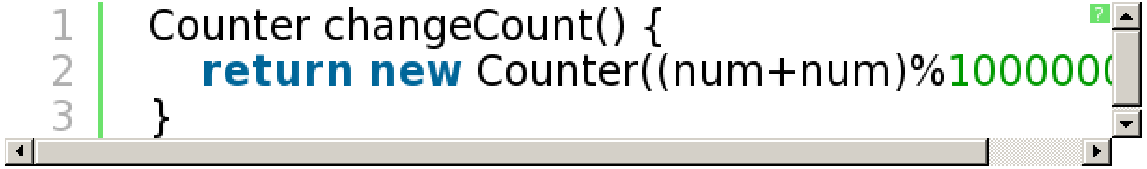
```
c.changeCount().
  changeCount().
  changeCount();
```

- Transparencia referencial

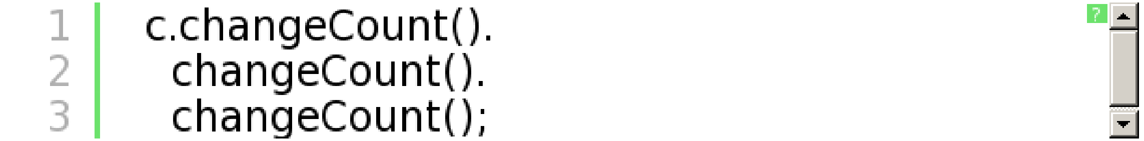


Imperativo vs Funcional

- Modelo Funcional
    - Combinar funciones
    - Generar un valor a partir del siguiente

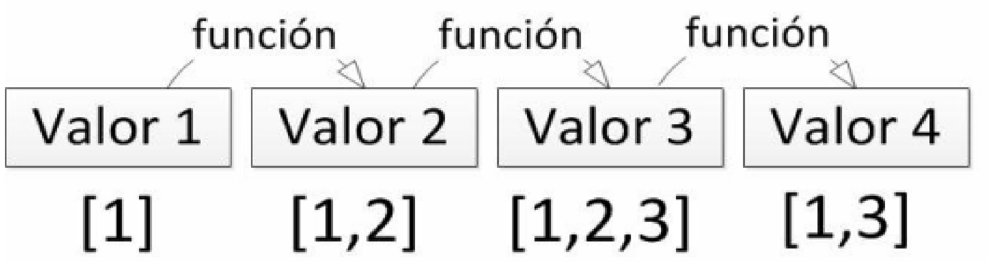


```
Counter changeCount() {
   return new Counter((num+num)%1000000
}
```



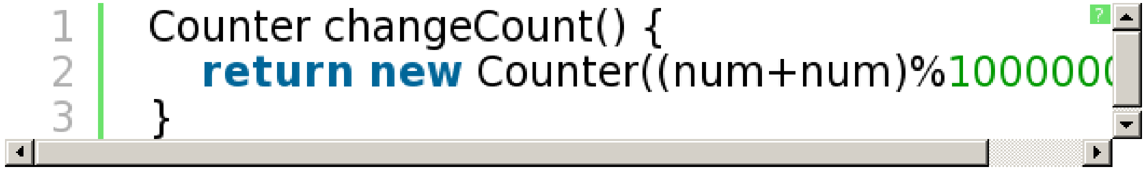
```
c.changeCount().
  changeCount().
  changeCount();
```

- Transparencia referencial
- Comparticion sin peligro

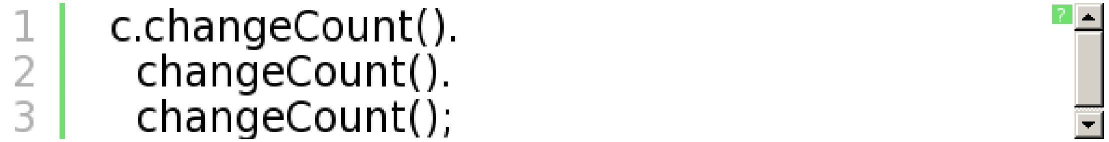


Imperativo vs Funcional

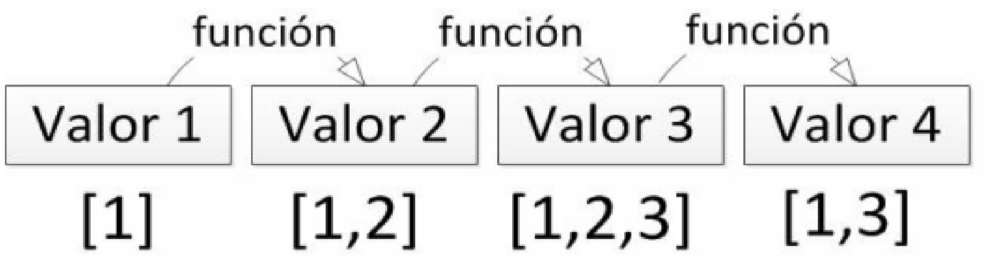
- Modelo Funcional
    - Combinar funciones
    - Generar un valor a partir del siguiente



```
Counter changeCount() {
   return new Counter((num+num)%1000000
}
```



```
c.changeCount().
  changeCount().
  changeCount();
```



función función función

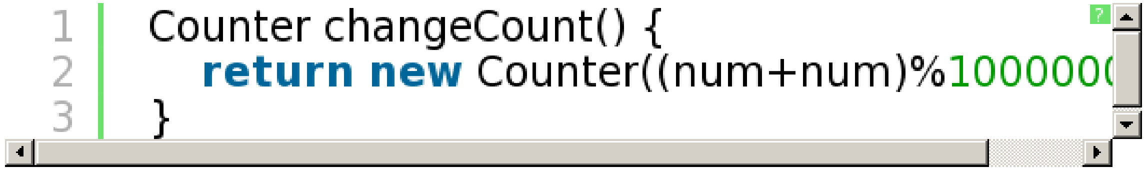
Valor 1 | Valor 2 | Valor 3 | Valor 4

[1] [1,2] [1,2,3] [1,3]

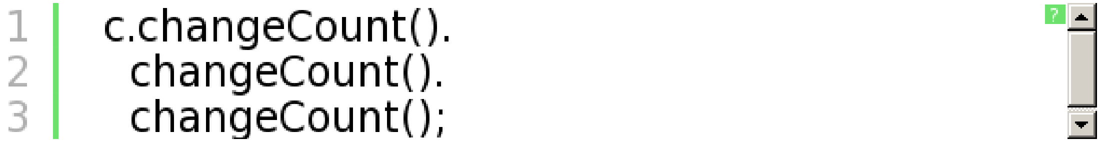
- Transparencia referencial
- Comparticion sin peligro
- Genera mas basura

Imperativo vs Funcional

- Modelo Funcional
  - Combinar funciones
  - Generar un valor a partir del siguiente

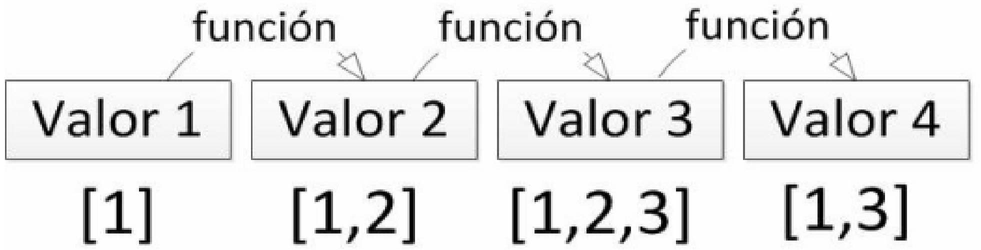


```
Counter changeCount() {
    return new Counter((num+num)%1000000
}
```



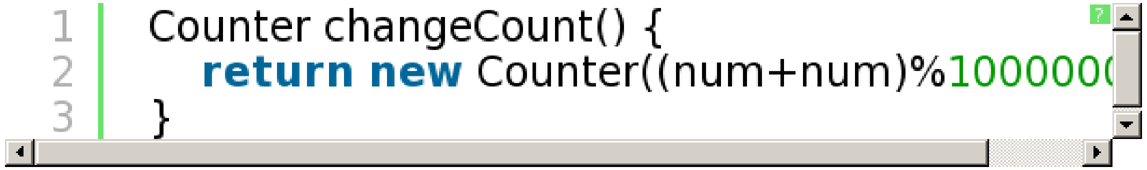
```
c.changeCount().
  changeCount().
  changeCount();
```

- Transparencia referencial
- Comparticion sin peligro
- Genera mas basura
- Necesita estructuras especificas

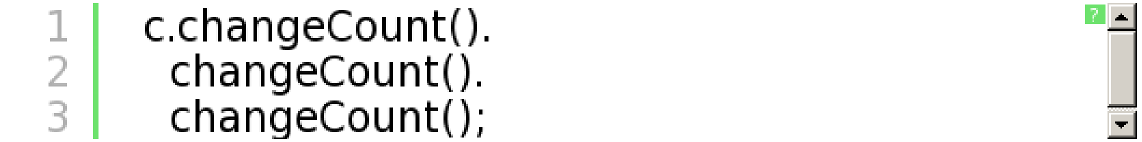


Imperativo vs Funcional

- Modelo Funcional
  - Combinar funciones
  - Generar un valor a partir del siguiente

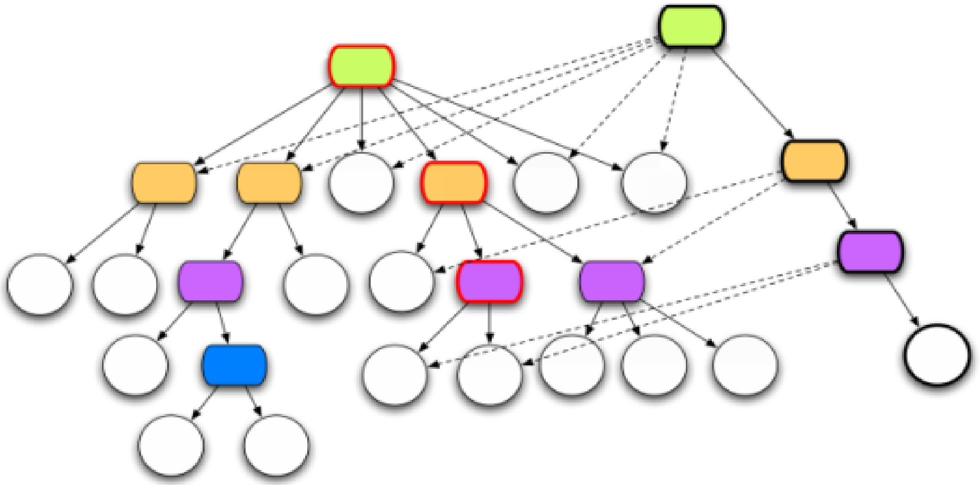


```
Counter changeCount() {
    return new Counter((num+num)%1000000
}
```



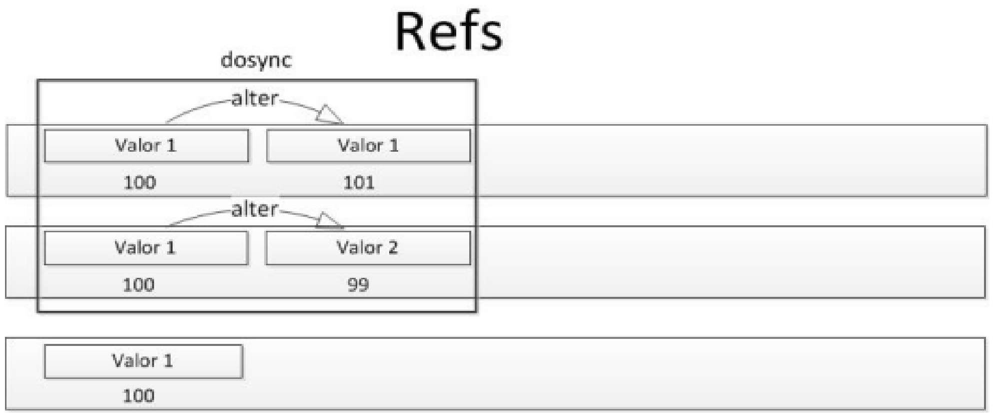
```
c.changeCount().
  changeCount().
  changeCount();
```

- Transparencia referencial
- Comparticion sin peligro
- Genera mas basura
- Necesita estructuras especificas



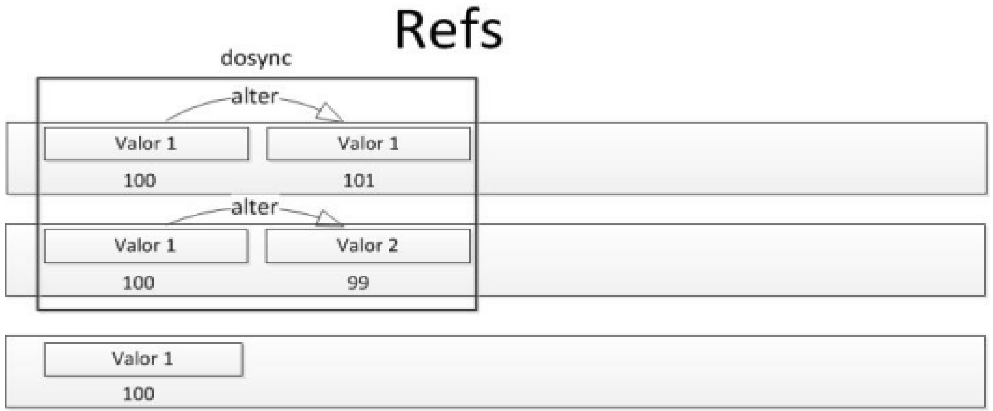
STM

STM


Refs
dosync
alter
Valor 1
100
Valor 1
101
alter
Valor 1
100
Valor 2
99
Valor 1
100

STM

- Software Transactional Memory

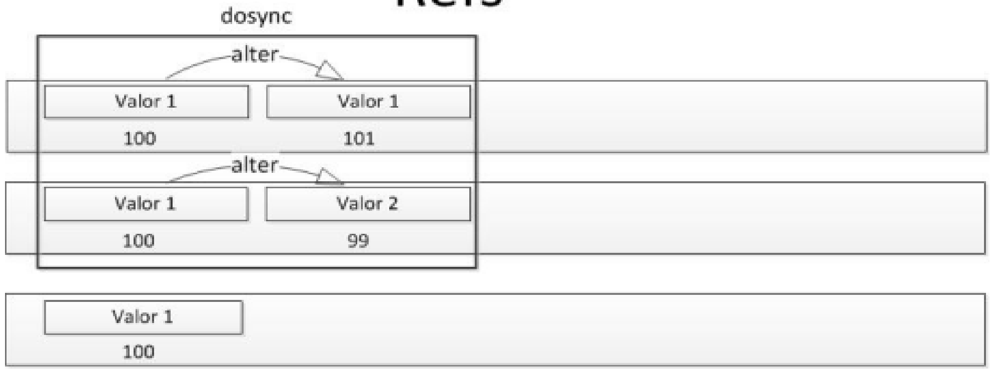


STM

- Software Transactional Memory

```clojure
(def mi-ref (ref 1))
```

Refs

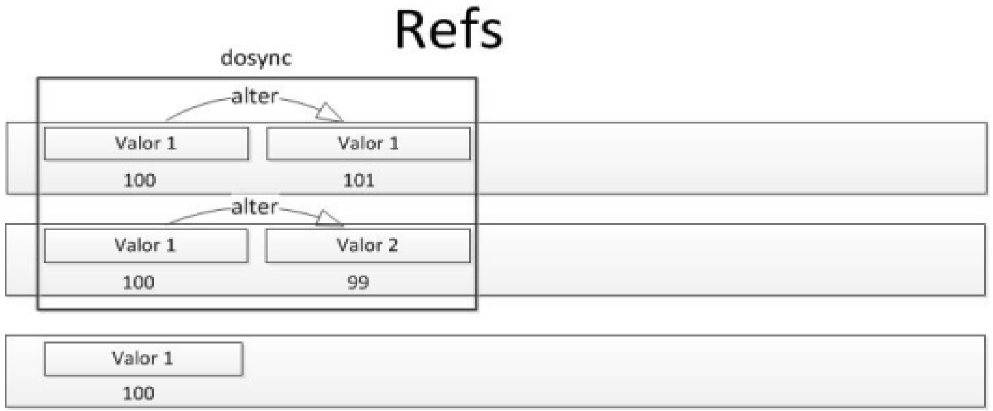


STM

- Software Transactional Memory

```
(def mi-ref (ref 1))
```

- Transacción



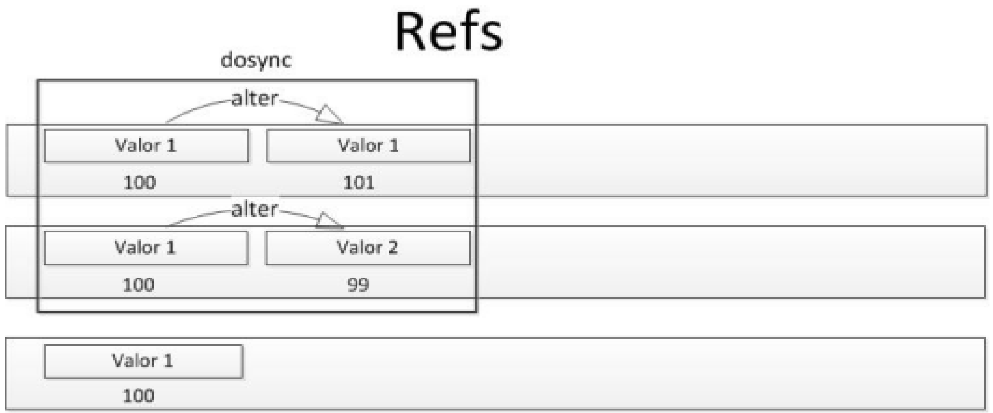
STM

- Software Transactional Memory

```clojure
(def mi-ref (ref 1))
```

- Transacción

```clojure
(dosync ...)
```



STM

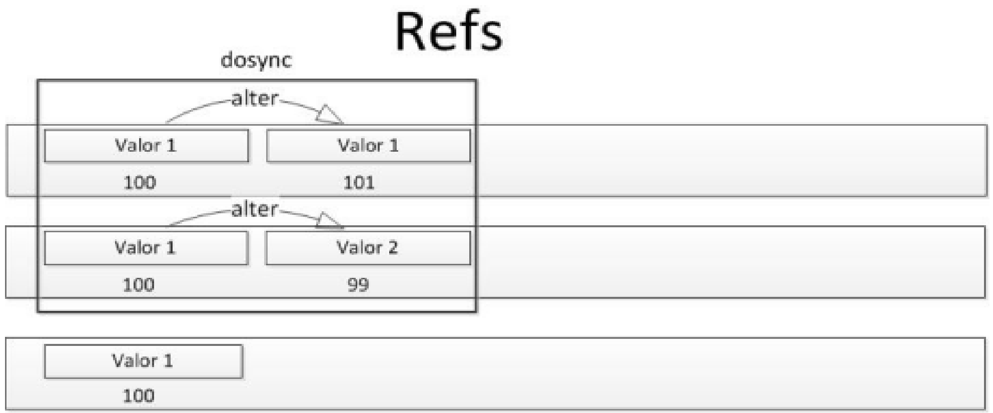
- Software Transactional Memory

```
(def mi-ref (ref 1))
```

- Transacción

```
(dosync ...)
```

```
(alter mi-ref inc)
```



Refs

STM

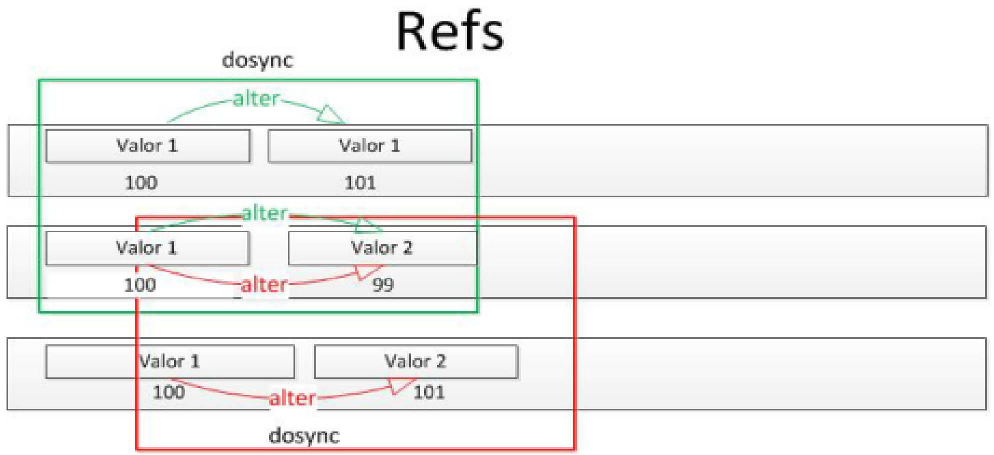
- Software Transactional Memory

```
(def mi-ref (ref 1))
```

- Transacción

```
(dosync ...)
```

```
(alter mi-ref inc)
```


Refs

STM

- Software Transactional Memory

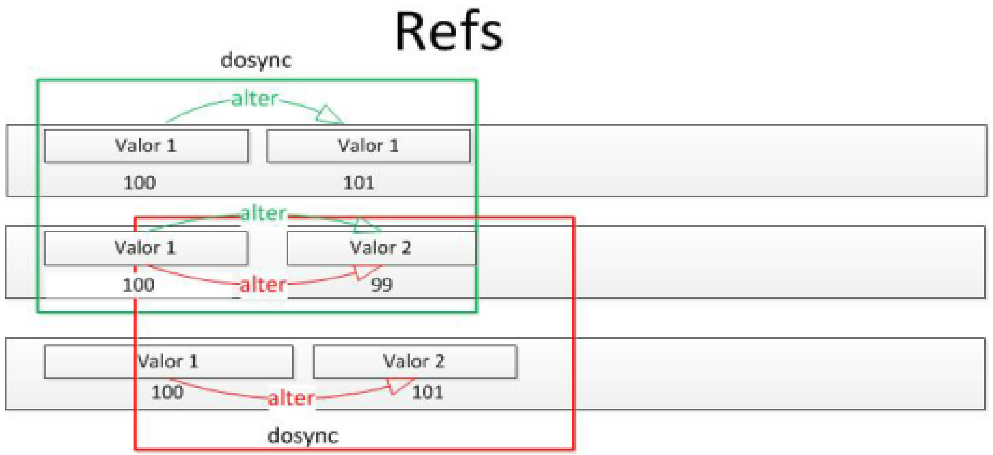
```
(def mi-ref (ref 1))
```

- Transacción

```
(dosync ...)
```

```
(alter mi-ref inc)
```

- Multiversion Concurrency Control (MVCC)



STM

- Software Transactional Memory

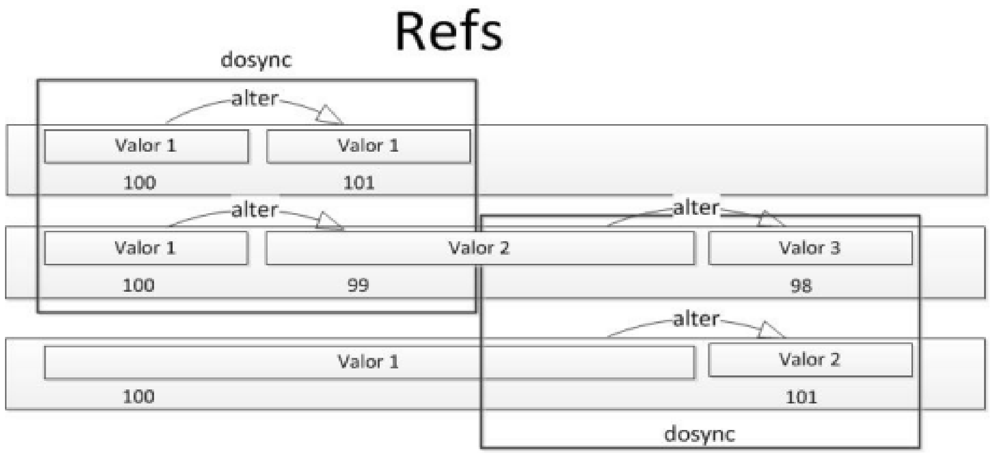
```clojure
(def mi-ref (ref 1))
```

- Transacción

```clojure
(dosync ...)
```

```clojure
(alter mi-ref inc)
```

- Multiversion Concurrency Control (MVCC)



STM

- Software Transactional Memory

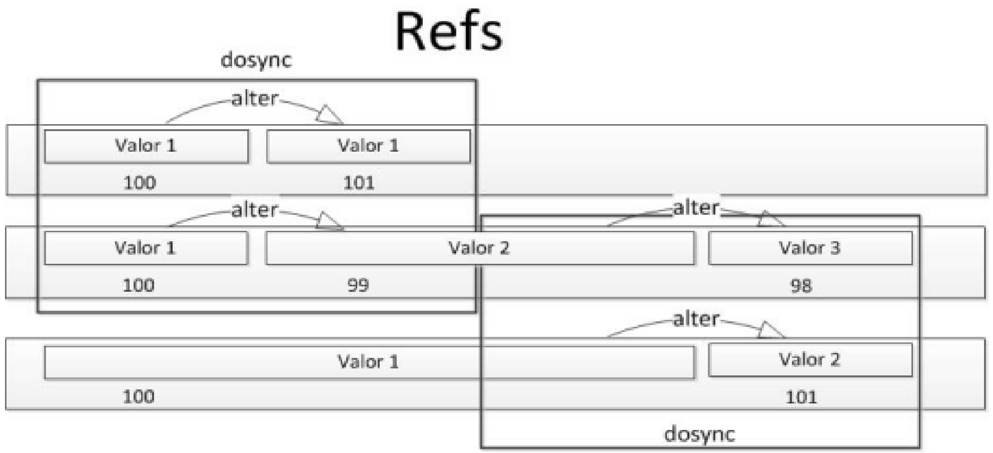
```
(def mi-ref (ref 1))
```

- Transacción

```
(dosync ...)
```

```
(alter mi-ref inc)
```

- Multiversion Concurrency Control (MVCC)
- ... and Retry



STM

- Software Transactional Memory

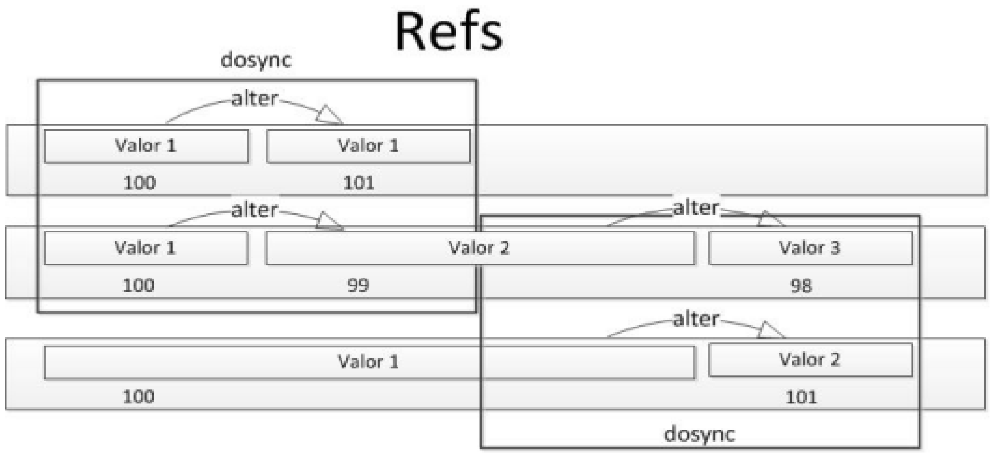
```
(def mi-ref (ref 1))
```

- Transacción

```
(dosync ...)
```

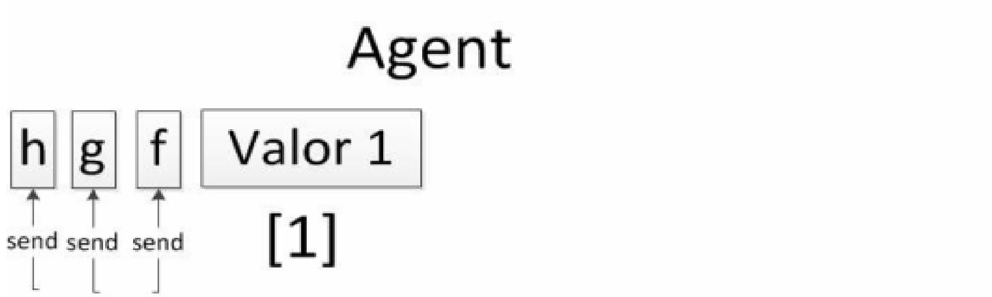
```
(alter mi-ref inc)
```

- Multiversion Concurrency Control (MVCC)
- ... and Retry
- Prohibidos efectos secundarios!!



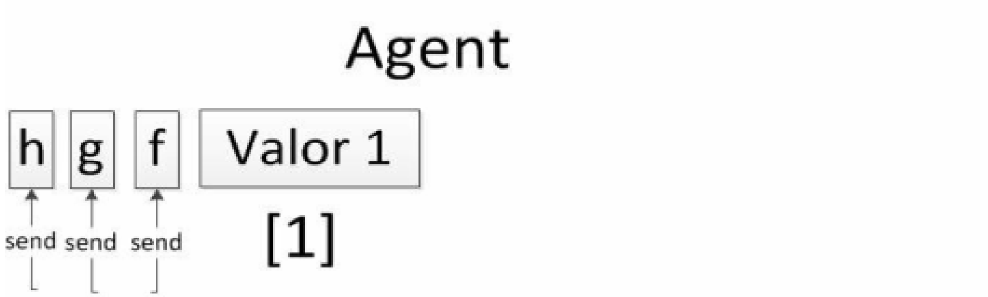
Agent

Agent



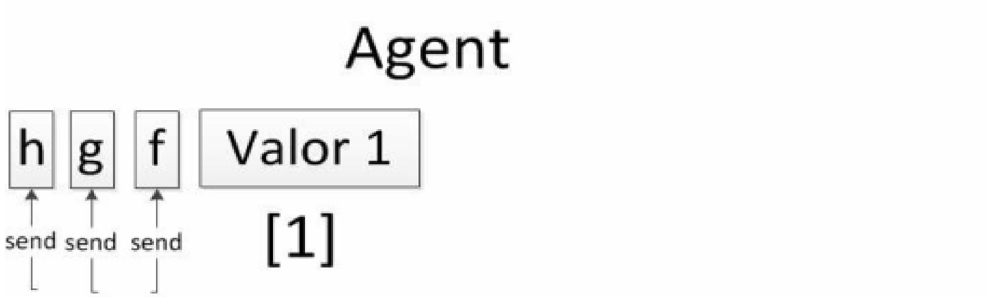
Agent

- Comportamiento asíncrono



Agent

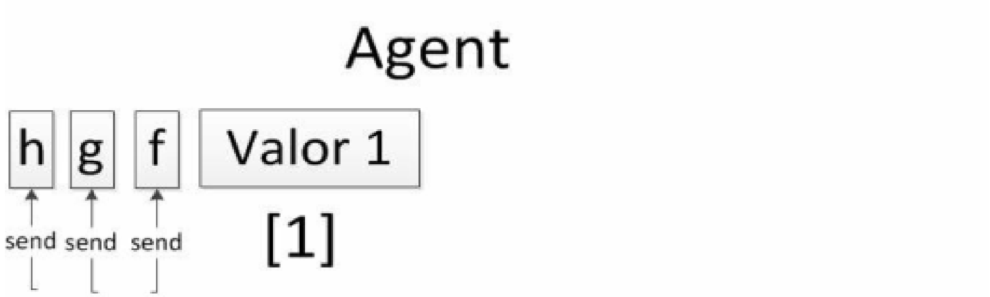
- Comportamiento asíncrono
- Ejecucion secuencial



Agent

- Comportamiento asíncrono
- Ejecucion secuencial

```
(def james (agent "bond"))
```

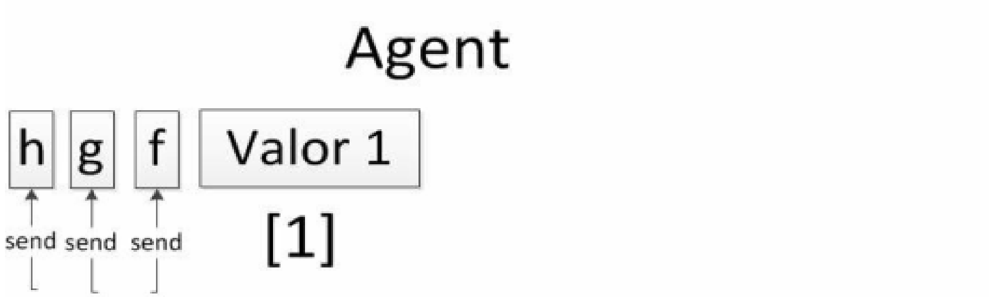


Agent

- Comportamiento asíncrono
- Ejecucion secuencial

```
(def james (agent "bond"))
```

```
(send james str " 007")
```

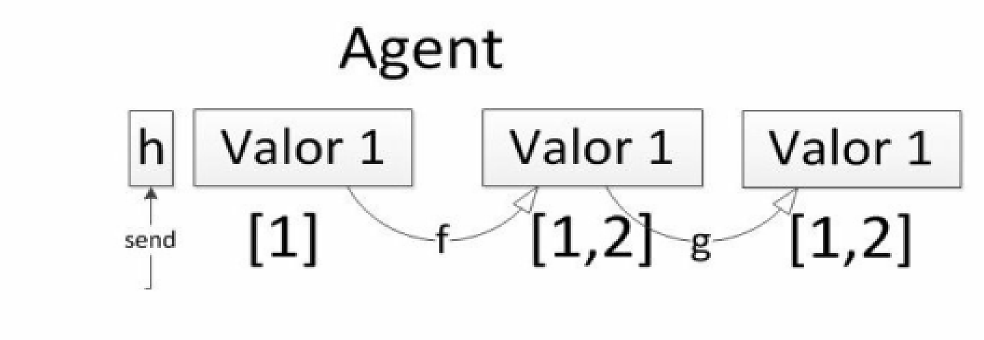


Agent

- Comportamiento asíncrono
- Ejecucion secuencial

```
(def james (agent "bond"))
```

```
(send james str " 007")
```



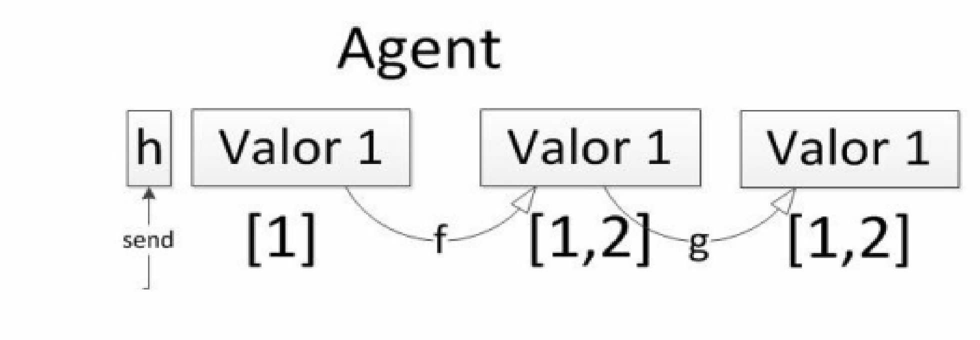
Agent

- Comportamiento asíncrono
- Ejecucion secuencial

```
(def james (agent "bond"))
```

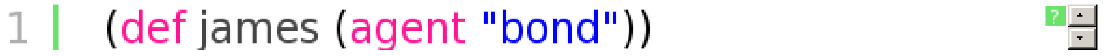
```
(send james str " 007")
```

- Ejecucion de entrada/salida

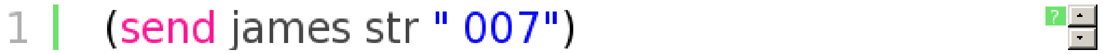


Agent

- Comportamiento asíncrono
- Ejecucion secuencial

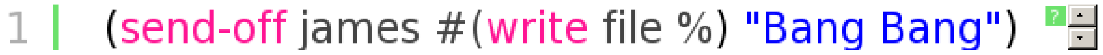


```
(def james (agent "bond"))
```

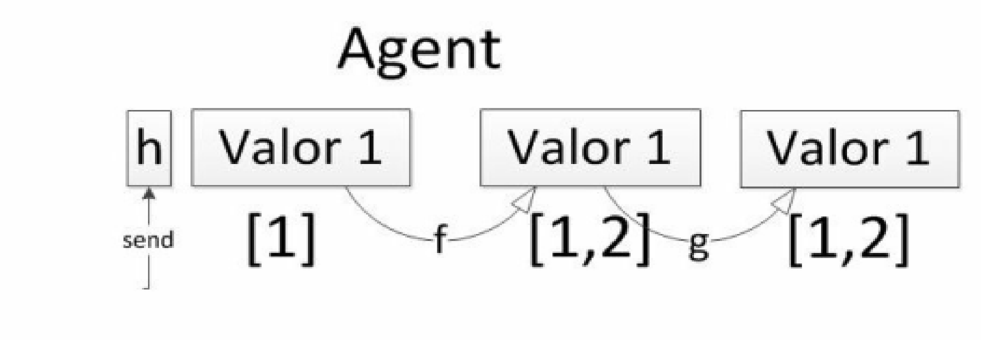


```
(send james str " 007")
```

- Ejecucion de entrada/salida



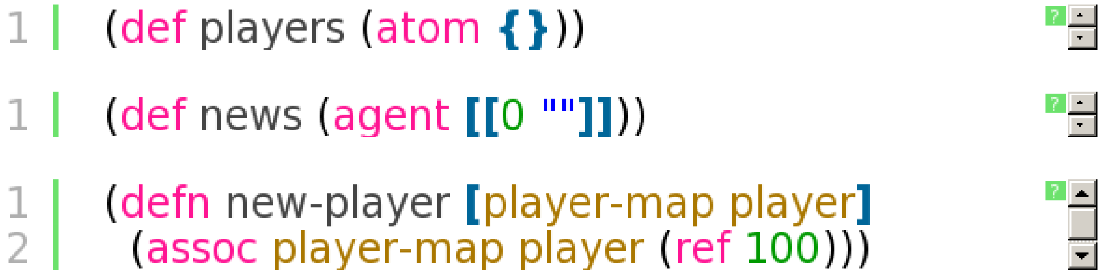
```
(send-off james #(write file %) "Bang Bang")
```



Implementacion del juego

Implementacion del juego

- Ejemplo en el juego



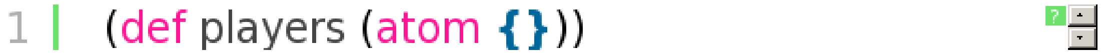
```clojure
(def players (atom {}))
```

```clojure
(def news (agent [[0 ""]]))
```

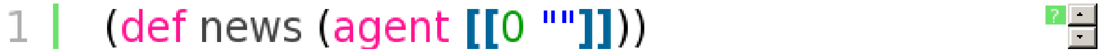
```clojure
(defn new-player [player-map player]
  (assoc player-map player (ref 100)))
```

Implementacion del juego

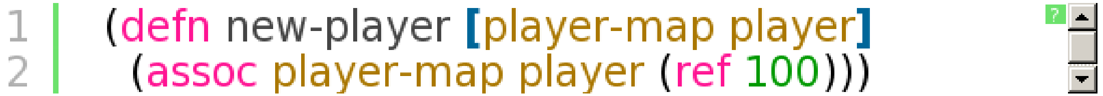
- Ejemplo en el juego



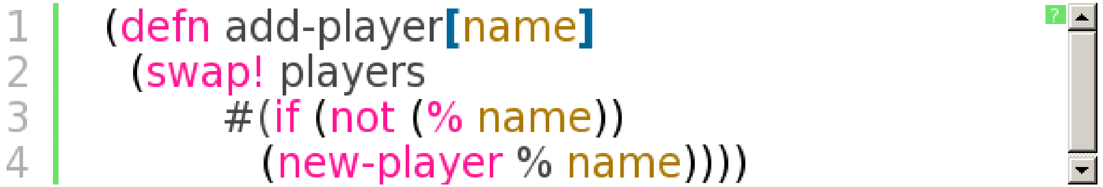
```
(def players (atom {}))
```



```
(def news (agent [[0 ""]]))
```



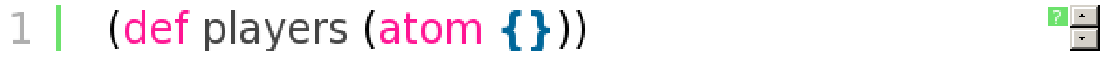
```
(defn new-player [player-map player]
  (assoc player-map player (ref 100)))
```



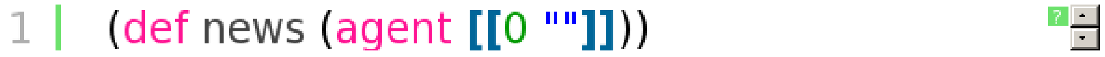
```
(defn add-player[name]
  (swap! players
     #(if (not (% name))
       (new-player % name))))
```

Implementacion del juego

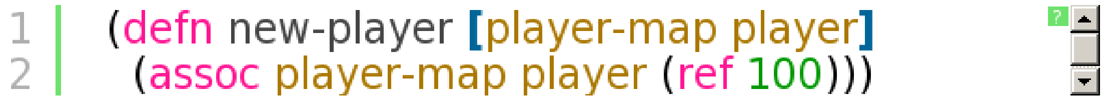
- Ejemplo en el juego



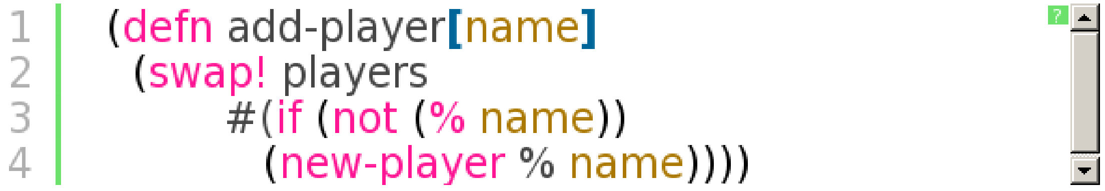
```
(def players (atom {}))
```



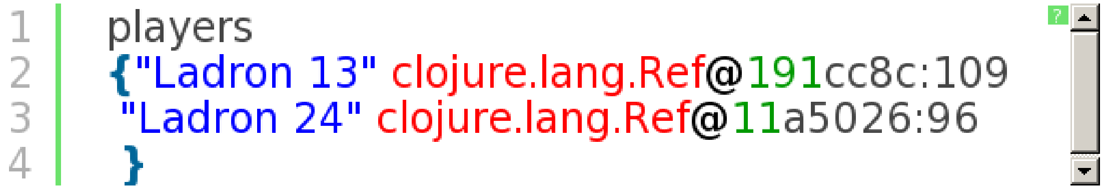
```
(def news (agent [[0 ""]]))
```



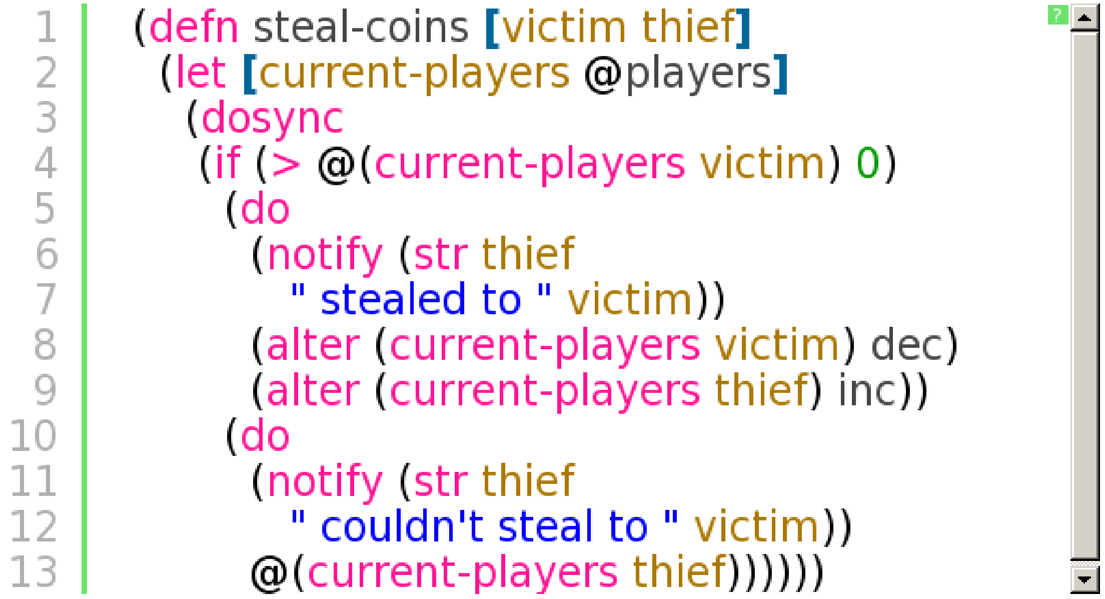
```
(defn new-player [player-map player]
  (assoc player-map player (ref 100)))
```



```
(defn add-player[name]
  (swap! players
         #(if (not (% name))
            (new-player % name))))
```



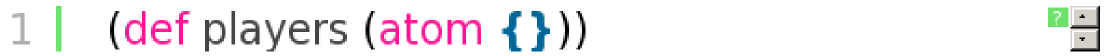
```
players
{"Ladron 13" clojure.lang.Ref@191cc8c:109
 "Ladron 24" clojure.lang.Ref@11a5026:96
 }
```



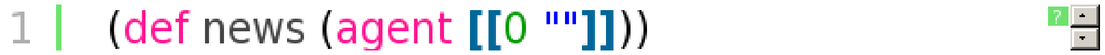
```
(defn steal-coins [victim thief]
  (let [current-players @players]
    (dosync
     (if (> @(current-players victim) 0)
       (do
         (notify (str thief
            " stealed to " victim))
         (alter (current-players victim) dec)
         (alter (current-players thief) inc))
       (do
         (notify (str thief
            " couldn't steal to " victim))
         @(current-players thief))))))
```

Implementacion del juego

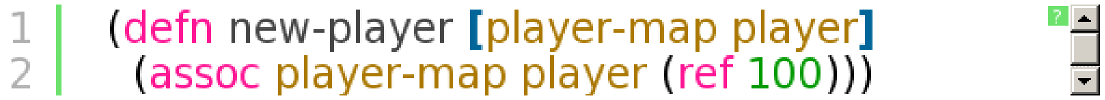
- Ejemplo en el juego



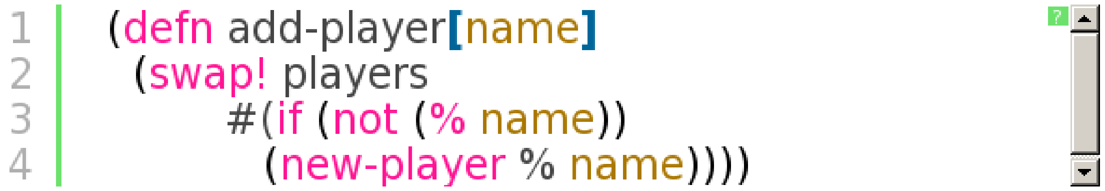
```clojure
(def players (atom {}))
```



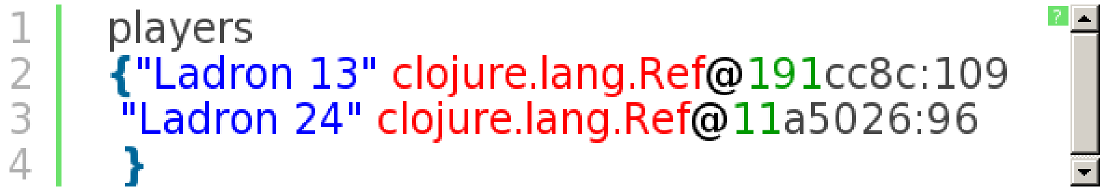
```clojure
(def news (agent [[0 ""]]))
```



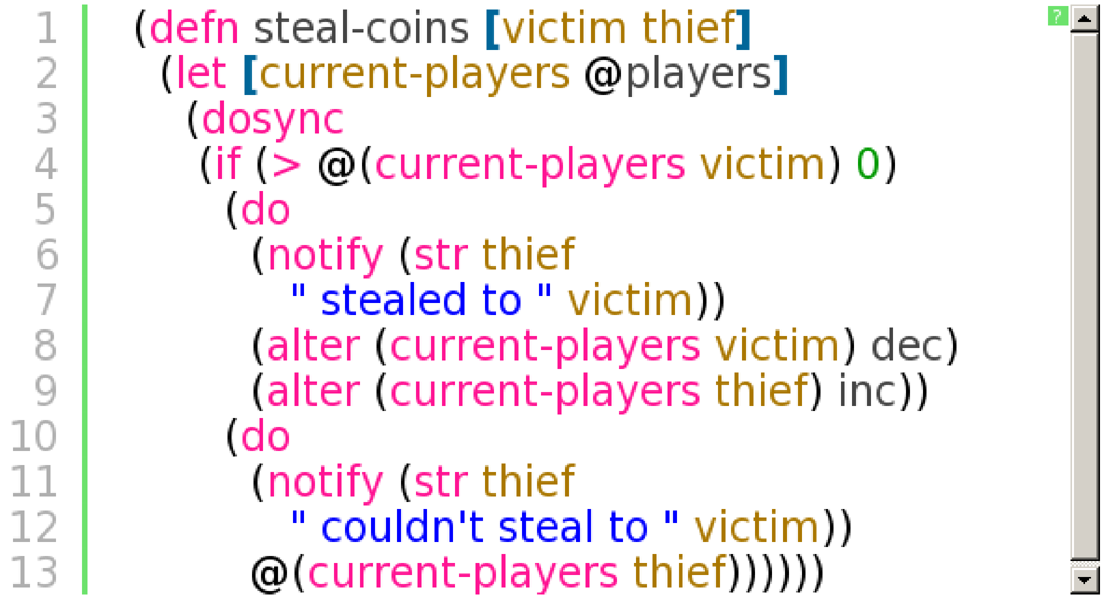
```clojure
(defn new-player [player-map player]
  (assoc player-map player (ref 100)))
```



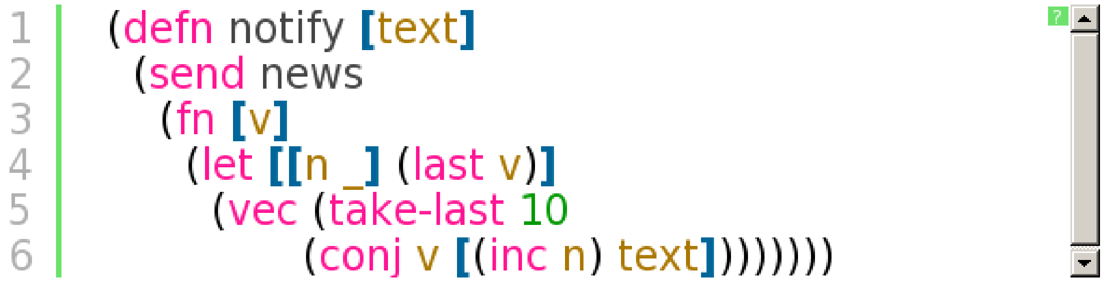
```clojure
(defn add-player[name]
  (swap! players
         #(if (not (% name))
            (new-player % name))))
```



```clojure
players
{"Ladron 13" clojure.lang.Ref@191cc8c:109
 "Ladron 24" clojure.lang.Ref@11a5026:96
 }
```



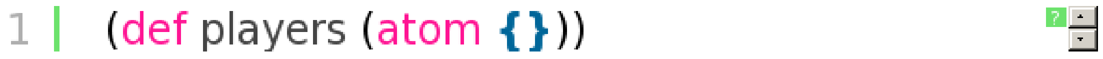
```clojure
(defn steal-coins [victim thief]
  (let [current-players @players]
    (dosync
     (if (> @(current-players victim) 0)
       (do
         (notify (str thief
            " stealed to " victim))
         (alter (current-players victim) dec)
         (alter (current-players thief) inc))
       (do
         (notify (str thief
            " couldn't steal to " victim))
         @(current-players thief))))))
```



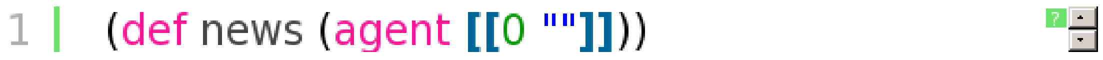
```clojure
(defn notify [text]
  (send news
    (fn [v]
      (let [[n _] (last v)]
        (vec (take-last 10
               (conj v [(inc n) text]))))))))
```

Implementacion del juego

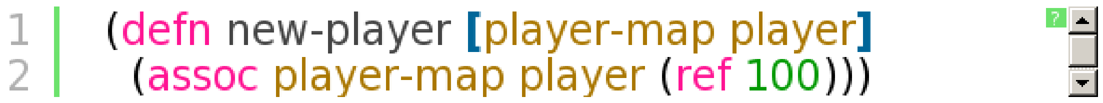
- Ejemplo en el juego



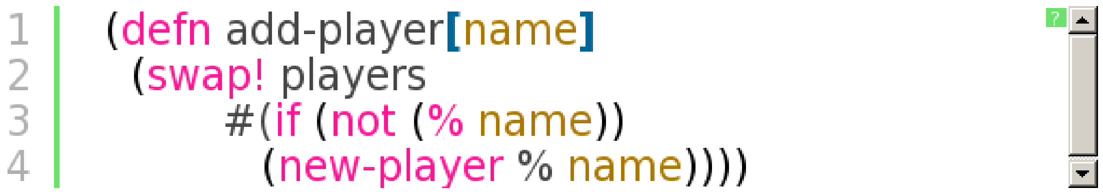
```clojure
(def players (atom {}))
```



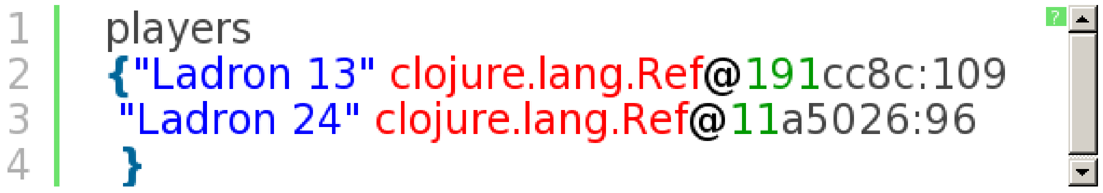
```clojure
(def news (agent [[0 ""]]))
```



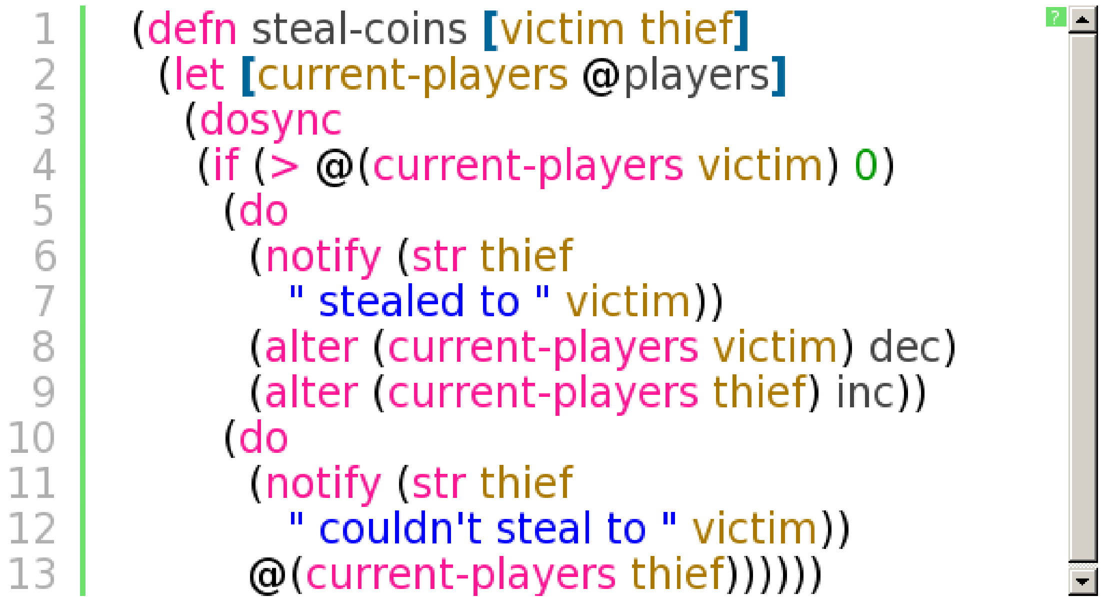
```clojure
(defn new-player [player-map player]
  (assoc player-map player (ref 100)))
```



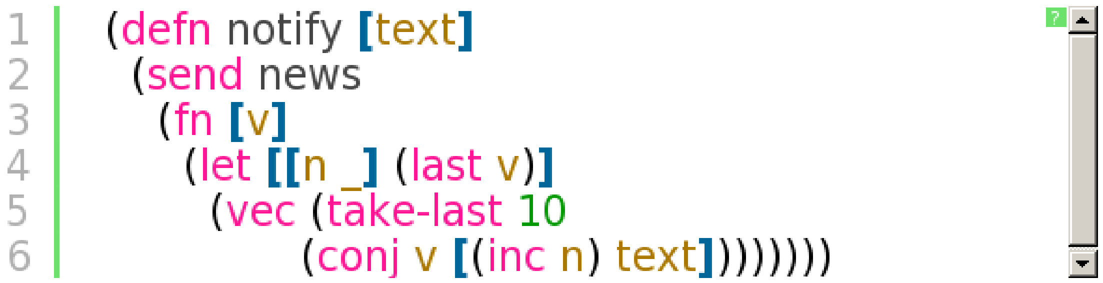
```clojure
(defn add-player[name]
  (swap! players
         #(if (not (% name))
            (new-player % name))))
```



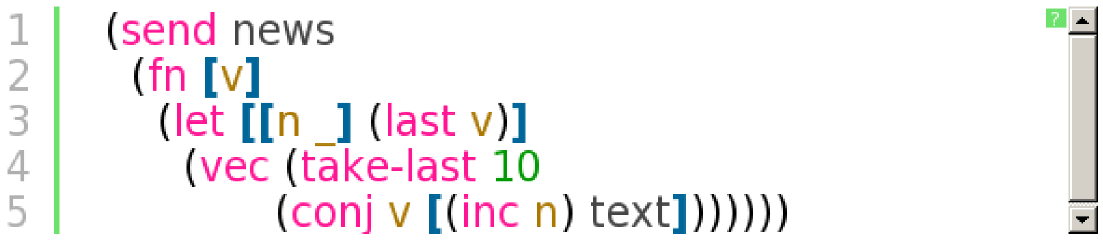
```clojure
players
{"Ladron 13" clojure.lang.Ref@191cc8c:109
 "Ladron 24" clojure.lang.Ref@11a5026:96
 }
```



```clojure
(defn steal-coins [victim thief]
  (let [current-players @players]
    (dosync
     (if (> @(current-players victim) 0)
       (do
         (notify (str thief
            " stealed to " victim))
         (alter (current-players victim) dec)
         (alter (current-players thief) inc))
       (do
         (notify (str thief
            " couldn't steal to " victim))
         @(current-players thief))))))
```



```clojure
(defn notify [text]
  (send news
    (fn [v]
      (let [[n _] (last v)]
        (vec (take-last 10
               (conj v [(inc n) text])))))))
```



```clojure
(send news
  (fn [v]
    (let [[n _] (last v)]
      (vec (take-last 10
             (conj v [(inc n) text]))))))
```