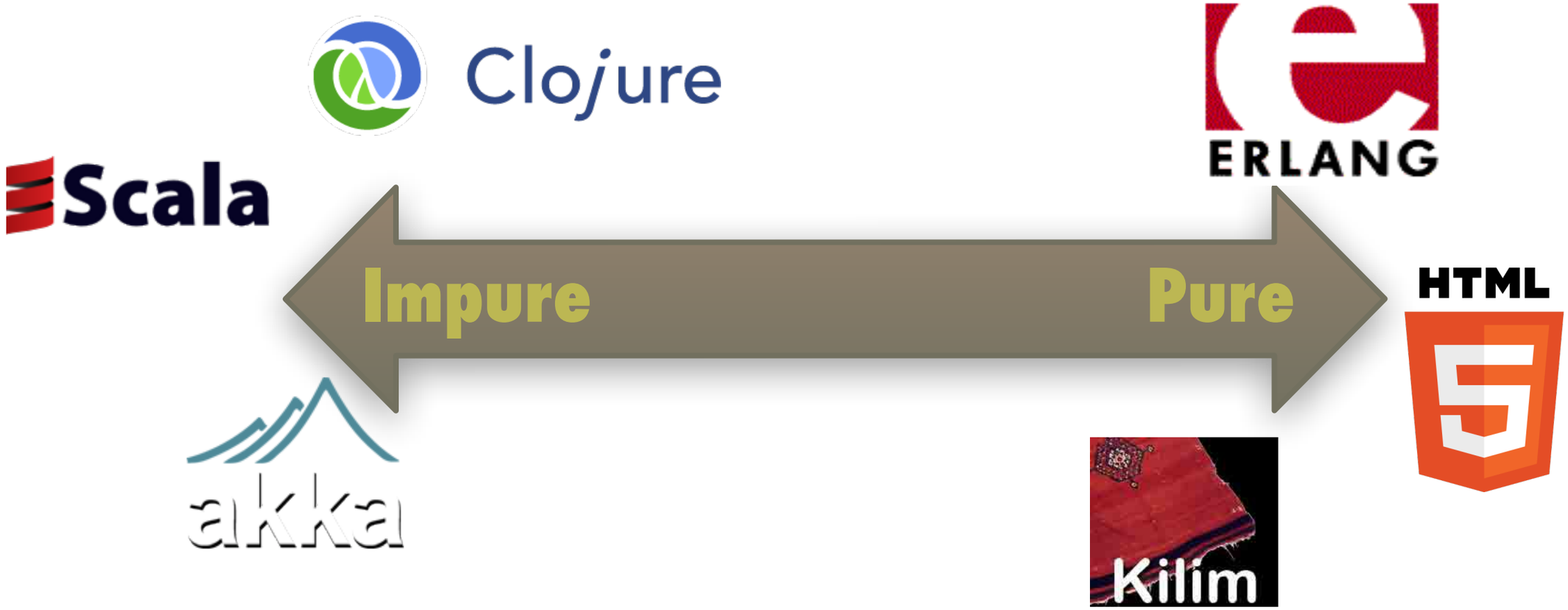
Domains: Safe sharing among actors

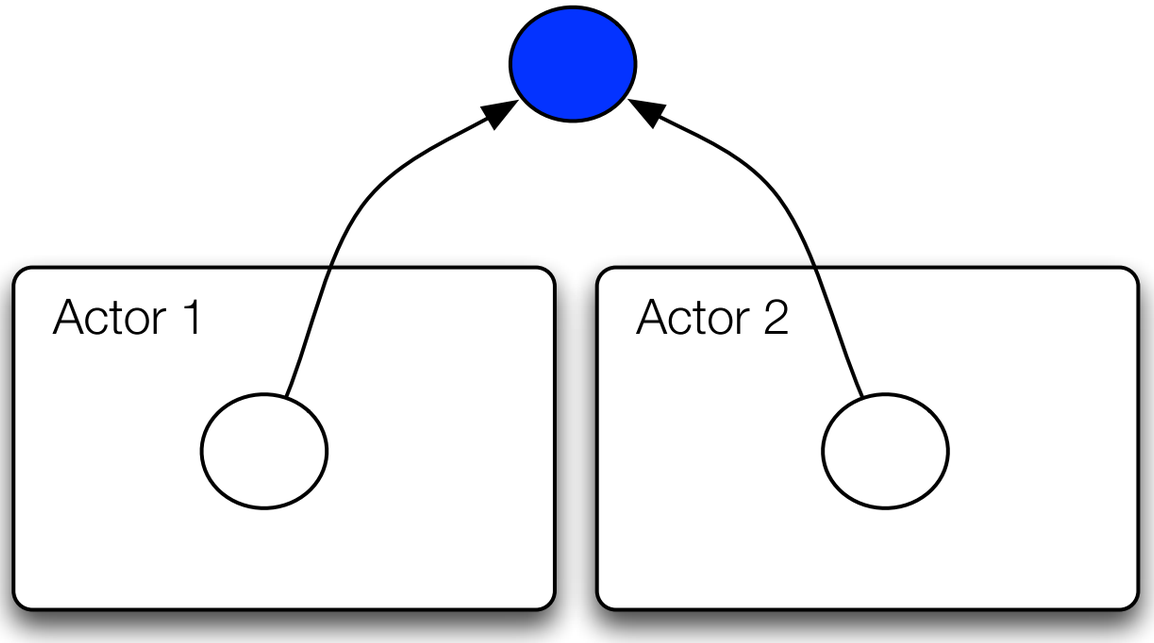
Joeri De Koster, Tom van Cutsem, Theo D'Hondt

Context



Impure

Synchronization?



Clojure

Scala

ERLANG

Impure

Pure

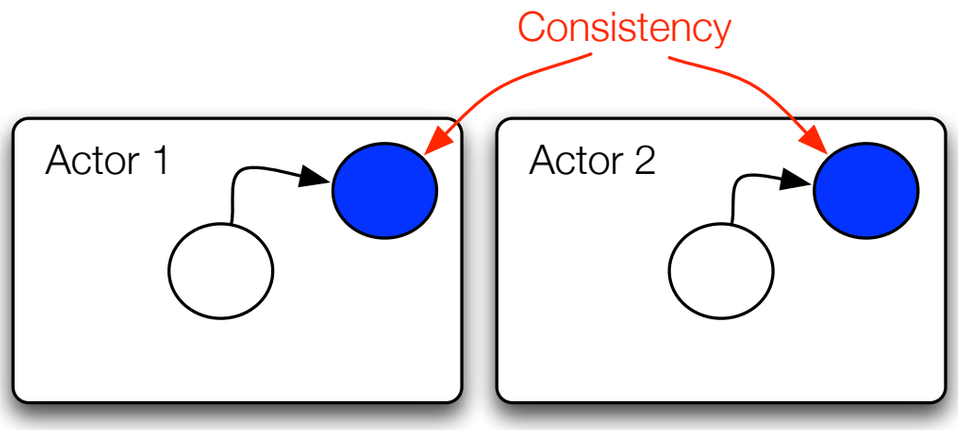
HTML

akka

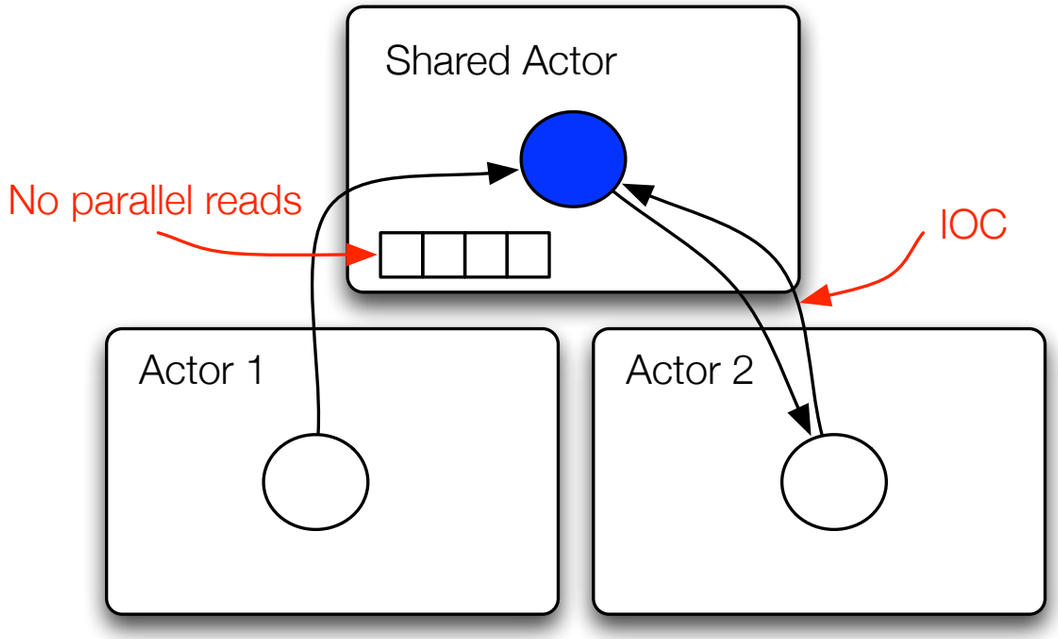
Kilim

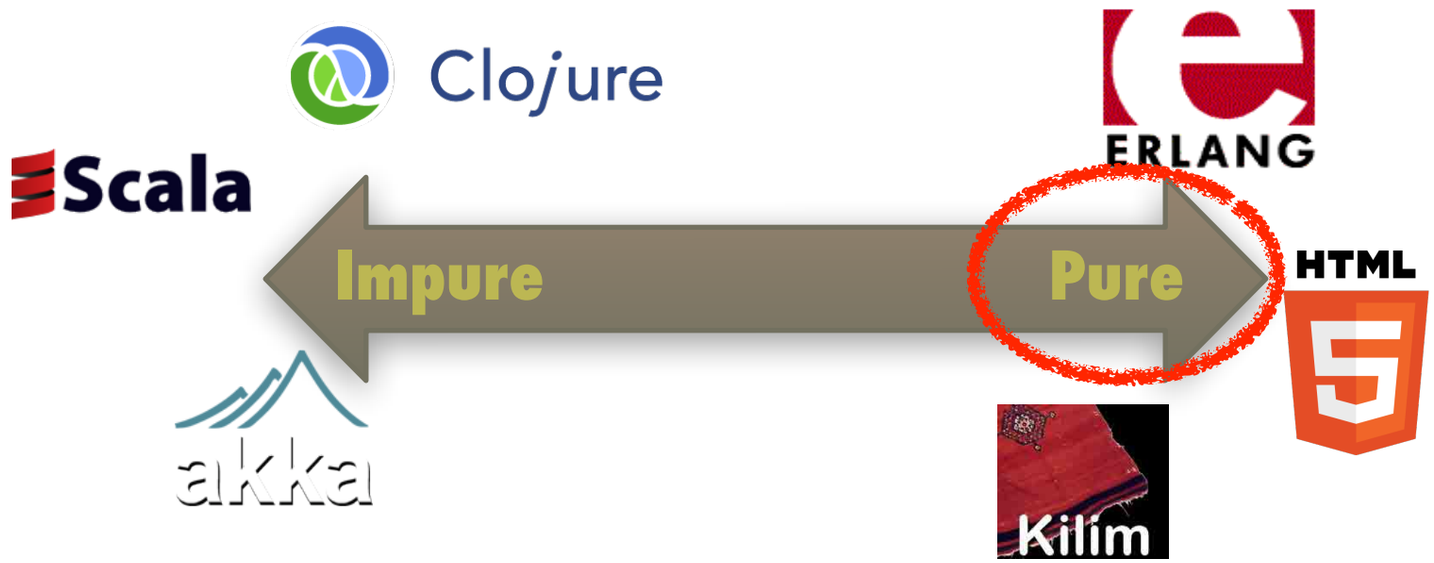
Pure

Replication

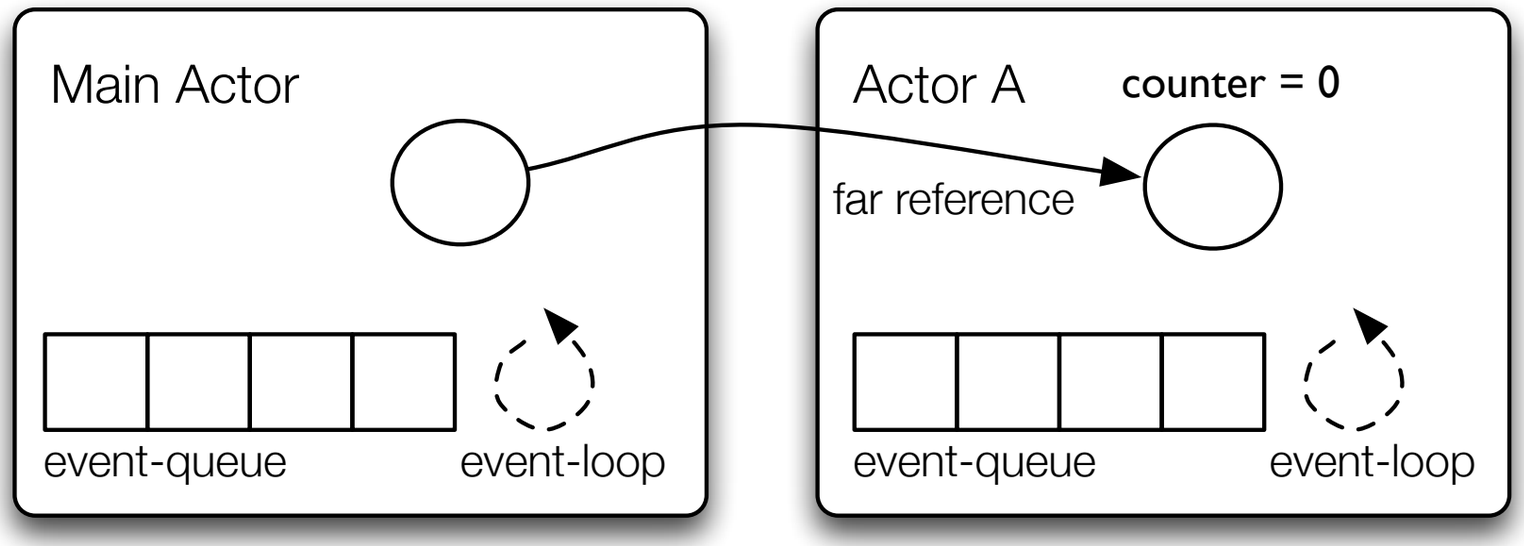


Isolation



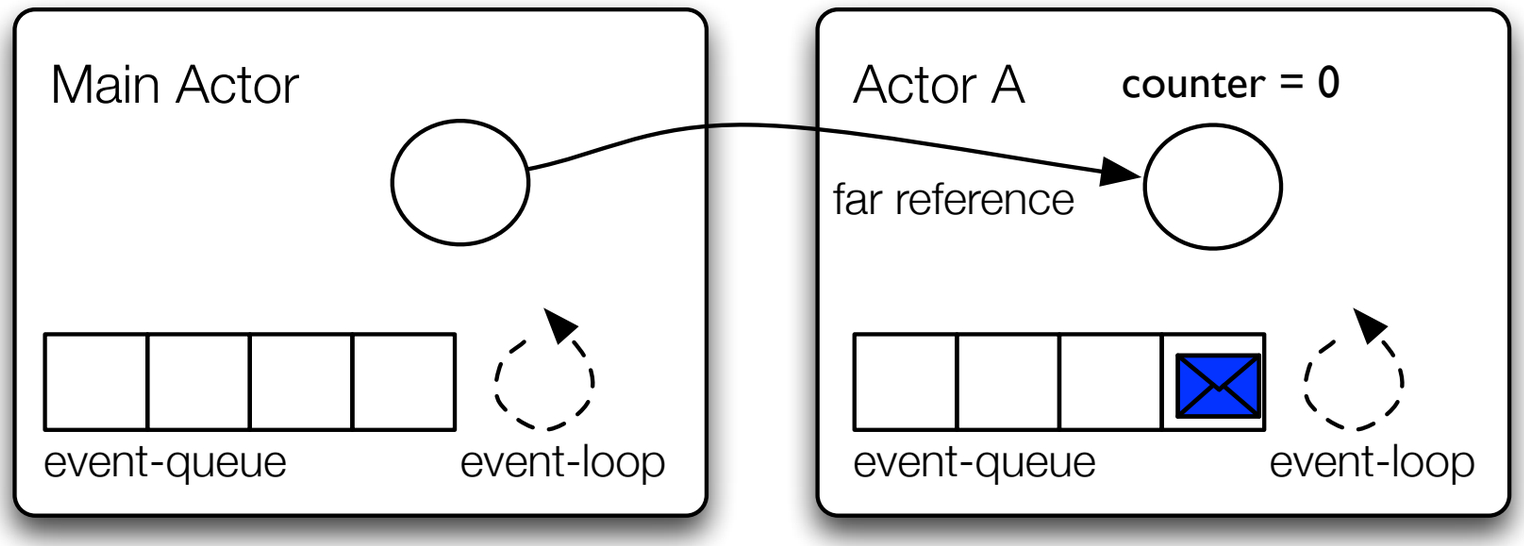


Shacl: Event-loop actors



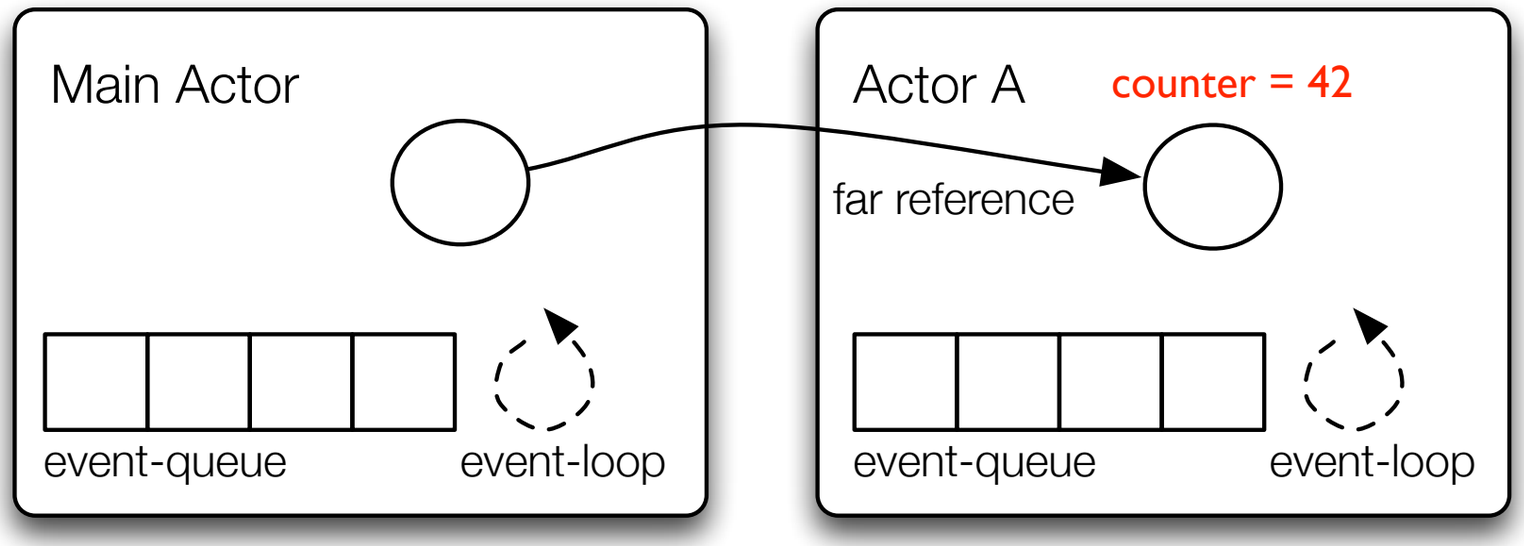

```
far_reference := actor {
  count := 0
  set(n) {
    count = n
  }
  get() {
    count
  }
}

far_reference<-set(42)
```

Shacl: Event-loop actors



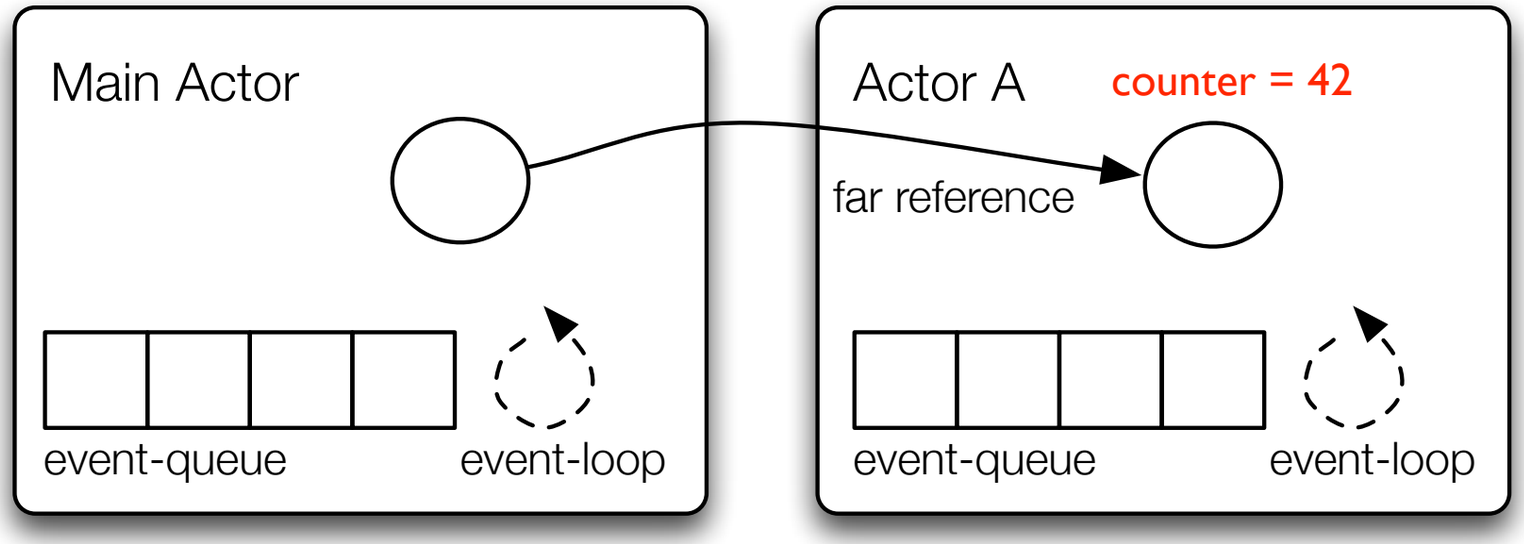

```
far_reference := actor {
  count := 0
  set(n) {
    count = n
  }
  get() {
    count
  }
}

far_reference<-set(42)
```

Shacl: Event-loop actors




```
far_reference := actor {
  count := 0
  set(n) {
    count = n
  }
  get() {
    count
  }
}

far_reference<-set(42)
```

Shacl: Event-loop actors

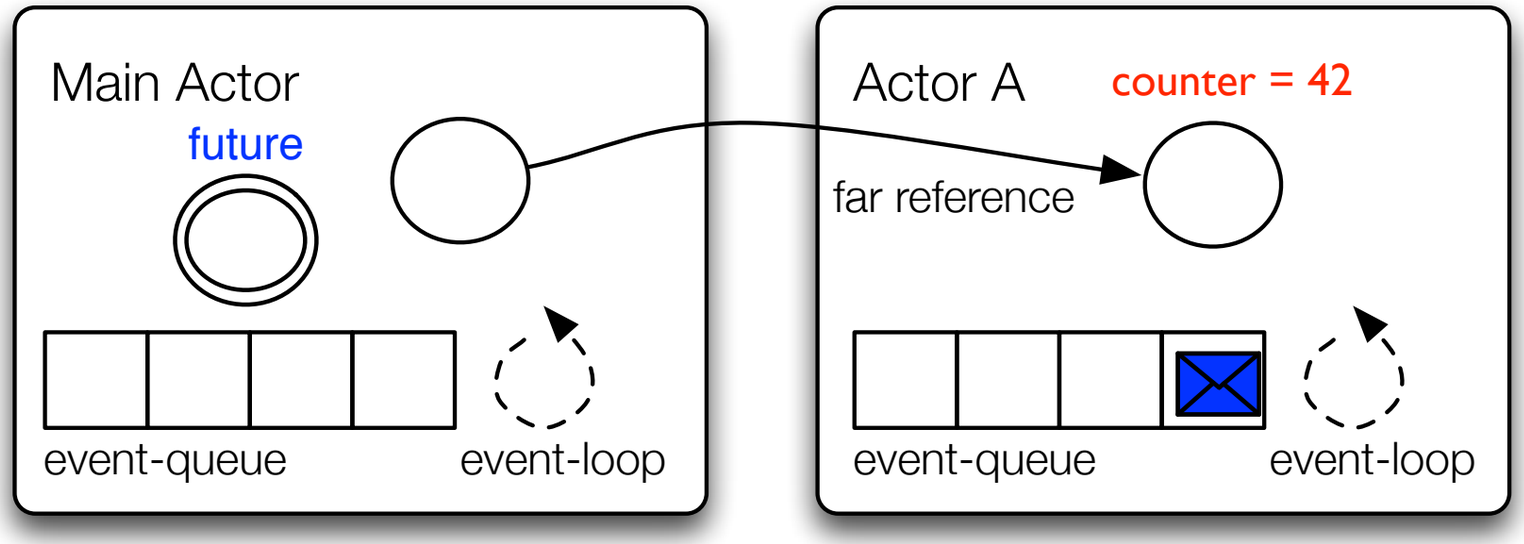



```
far_reference := actor {
  count := 0
  set(n) {
    count = n
  }
  get() {
    count
  }
}

far_reference<-set(42)
```

```
future := far_reference<-get()
```

```
whenResolved(future) { |value|
  display("Counter: ", value)
}
```

Shacl: Event-loop actors

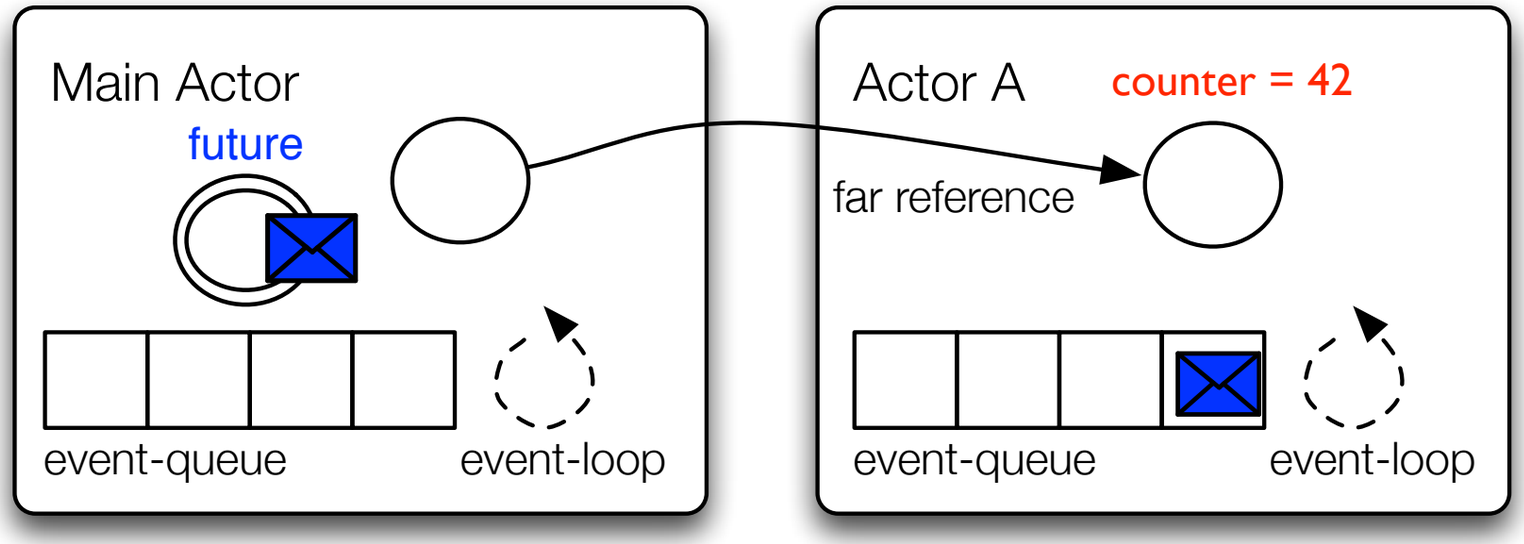



```
far_reference := actor {
  count := 0
  set(n) {
    count = n
  }
  get() {
    count
  }
}

far_reference<-set(42)
```

```
future := far_reference<-get()
```

```
whenResolved(future) { |value|
  display("Counter: ", value)
}
```

Shacl: Event-loop actors

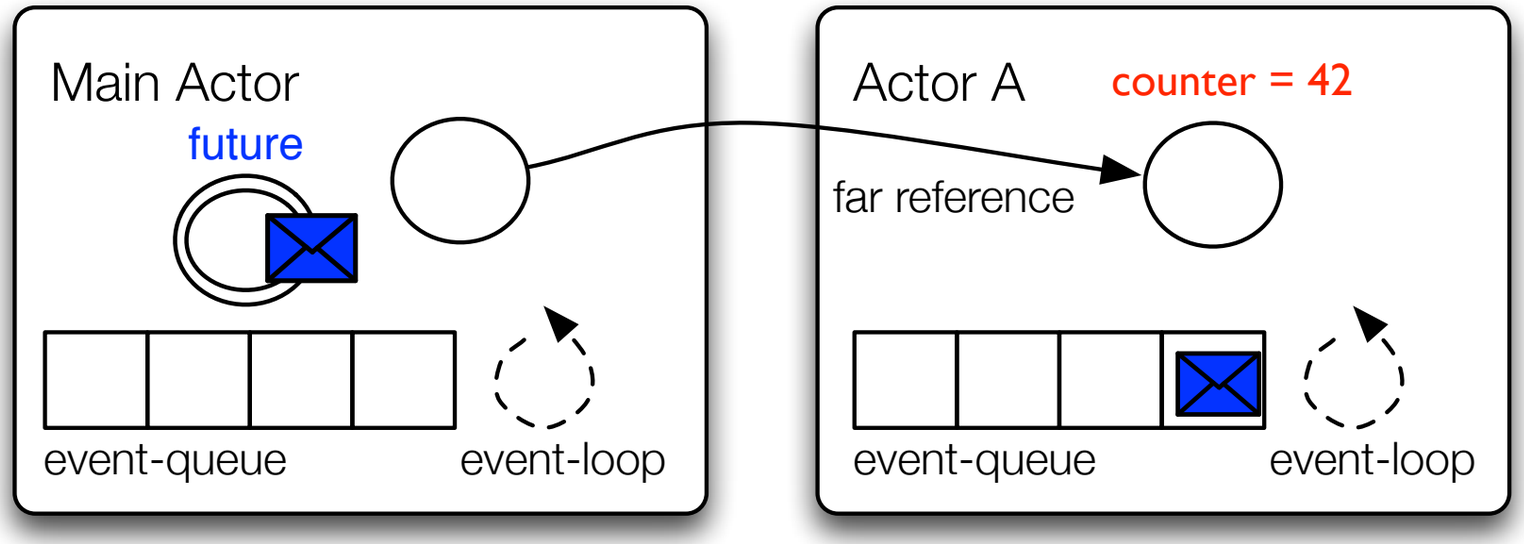



```
far_reference := actor {
  count := 0
  set(n) {
    count = n
  }
  get() {
    count
  }
}

far_reference<-set(42)
```

```
future := far_reference<-get()
```

```
whenResolved(future) { |value|
  display("Counter: ", value)
}
```

Shacl: Event-loop actors

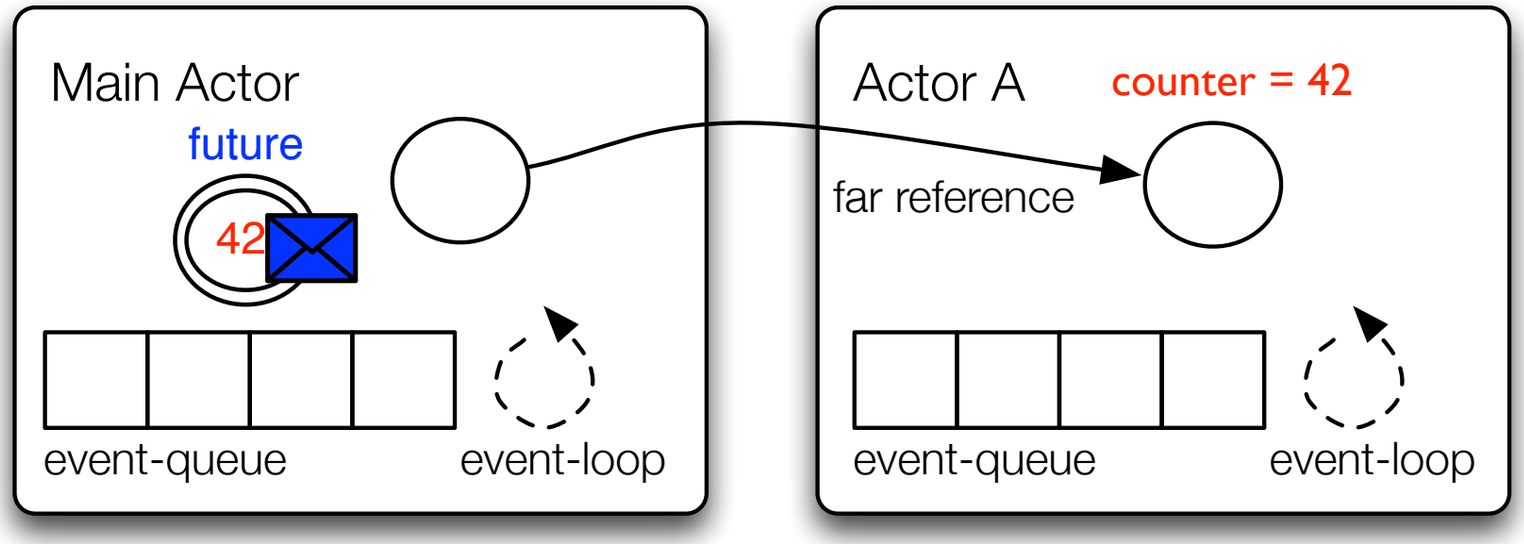



```
far_reference := actor {
  count := 0
  set(n) {
    count = n
  }
  get() {
    count
  }
}

far_reference<-set(42)
```

```
future := far_reference<-get()
```

```
whenResolved(future) { |value|
  display("Counter: ", value)
}
```

Shacl: Event-loop actors

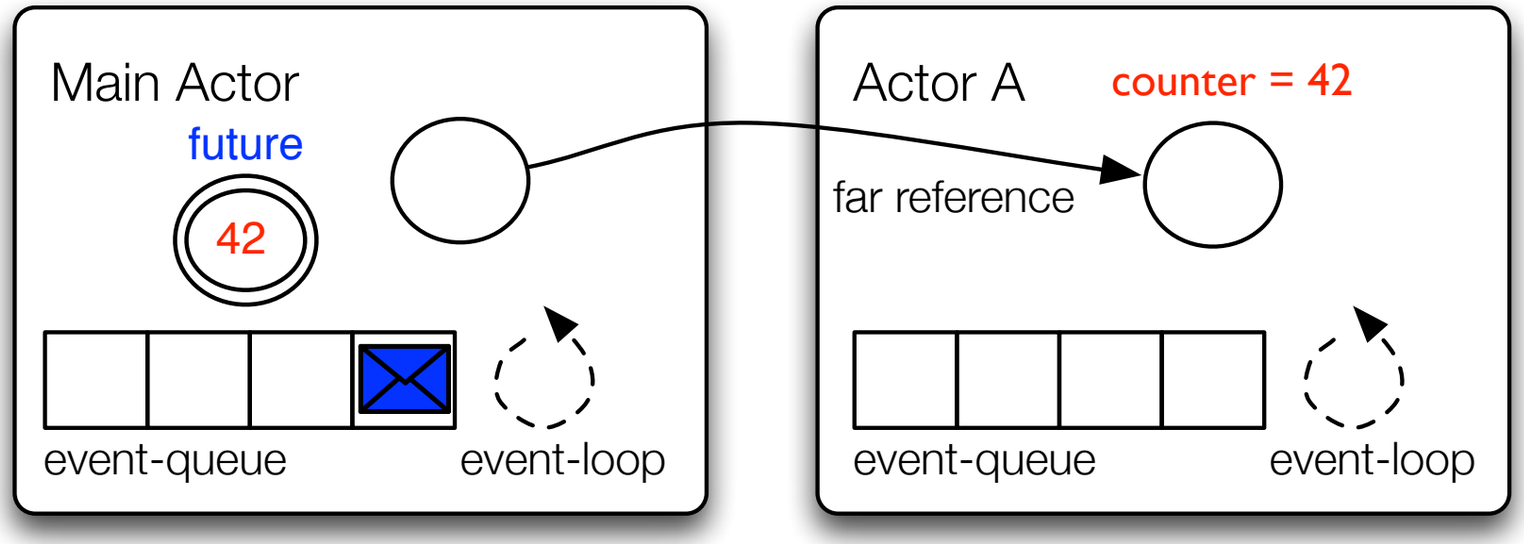



```
far_reference := actor {
  count := 0
  set(n) {
    count = n
  }
  get() {
    count
  }
}

far_reference<-set(42)
```

```
future := far_reference<-get()
```

```
whenResolved(future) { |value|
  display("Counter: ", value)
}
```

Shacl: Event-loop actors

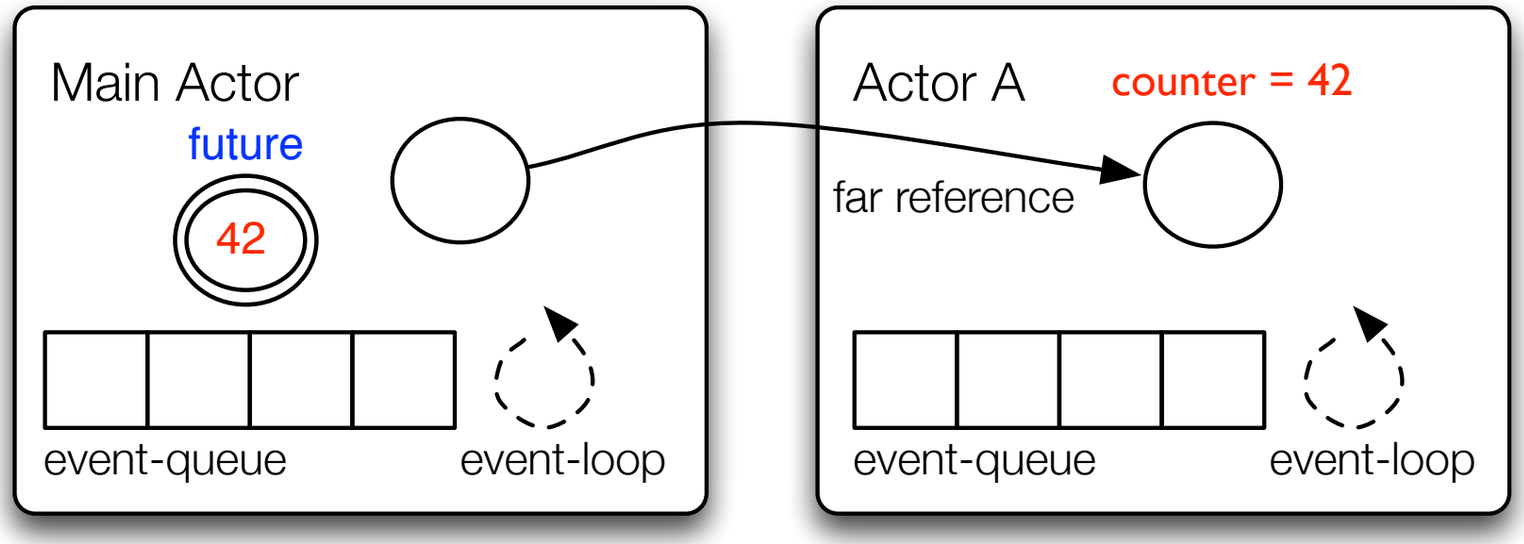



```
far_reference := actor {
  count := 0
  set(n) {
    count = n
  }
  get() {
    count
  }
}

far_reference<-set(42)
```

```
future := far_reference<-get()
```

```
whenResolved(future) { |value|
  display("Counter: ", value)
}
```

Shacl: Event-loop actors

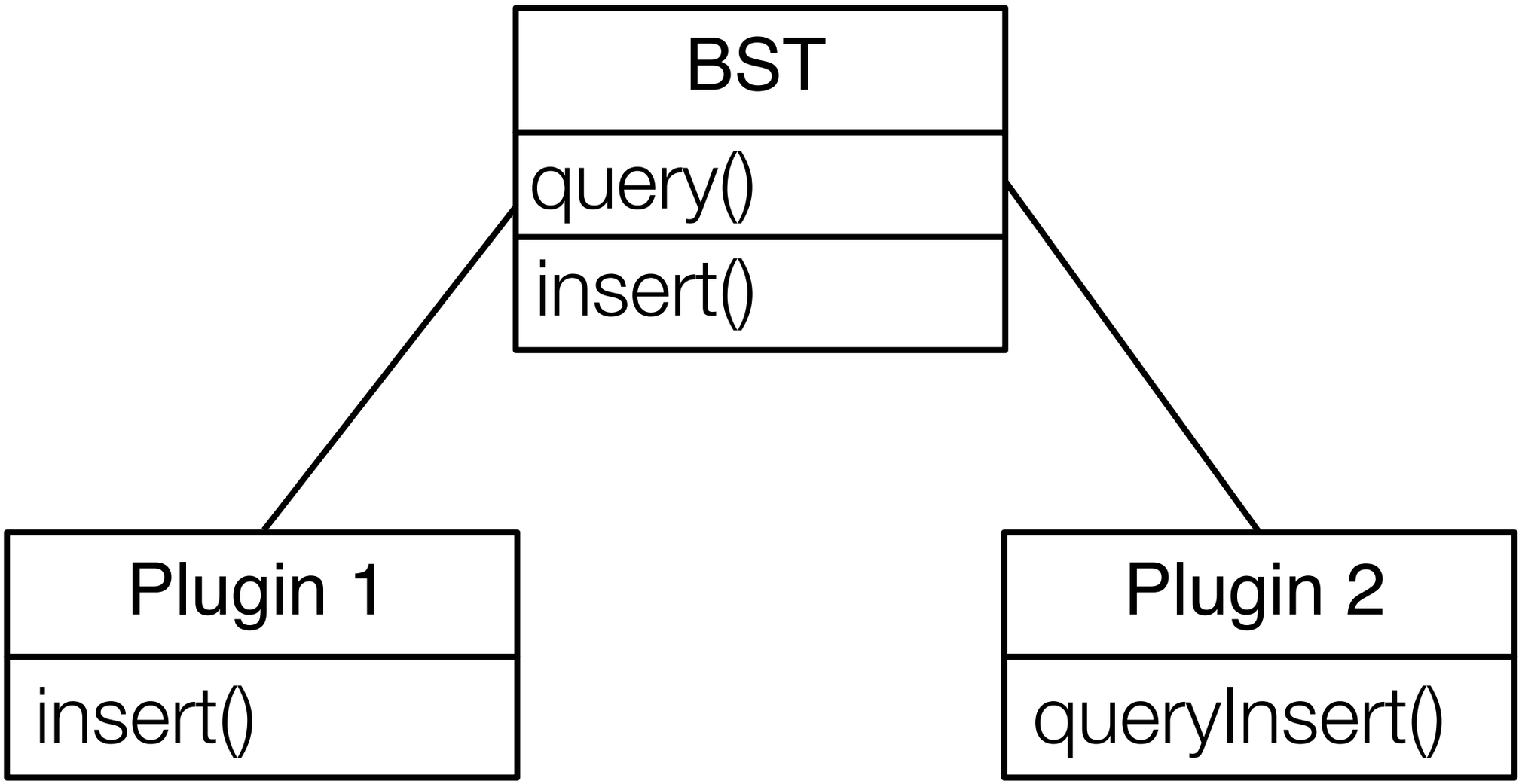



```
far_reference := actor {
  count := 0
  set(n) {
    count = n
  }
  get() {
    count
  }
}

far_reference<-set(42)
```

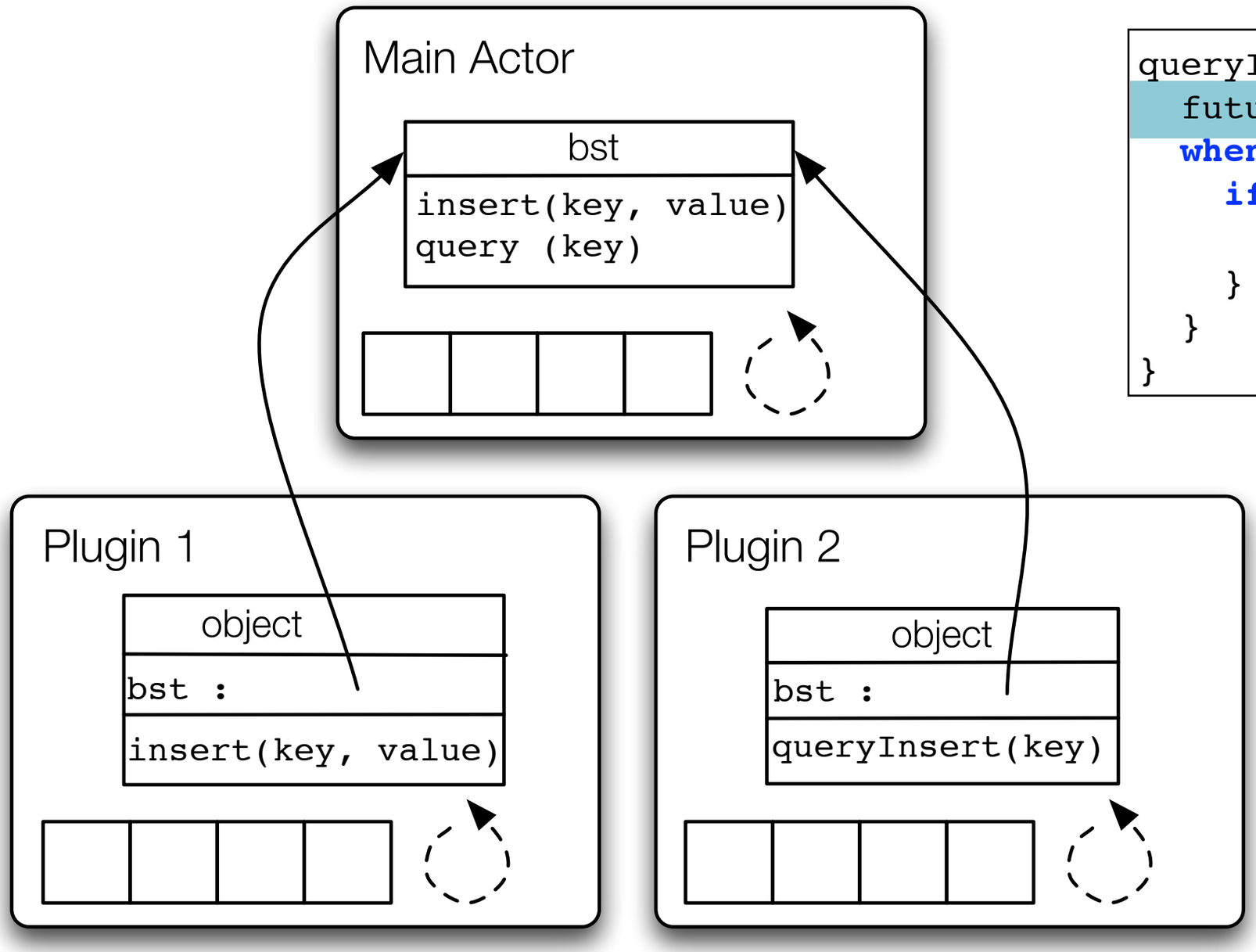
```
future := far_reference<-get()
```

```
whenResolved(future) { |value|
  display("Counter: ", value)
}
```

Motivating Example

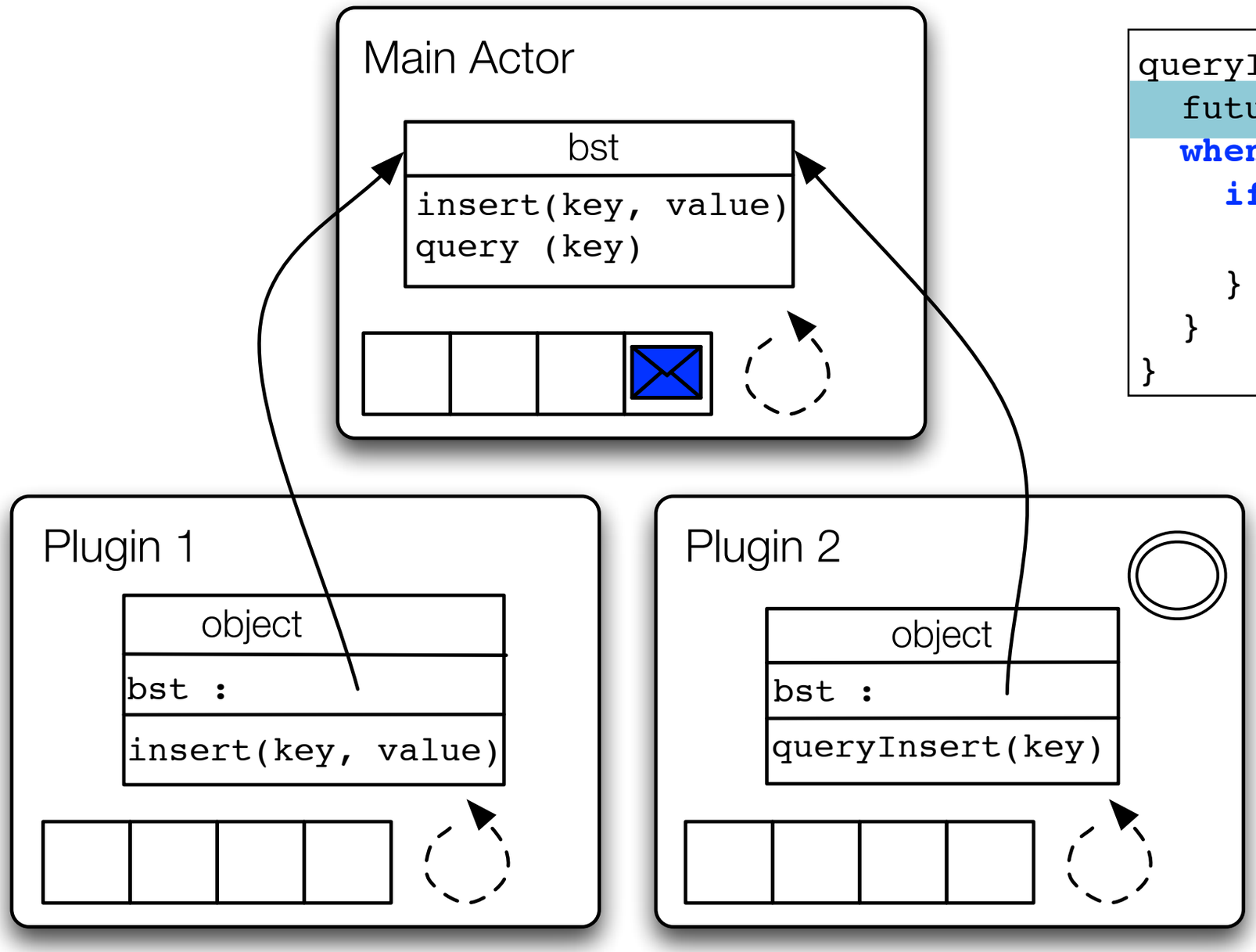


Motivating example



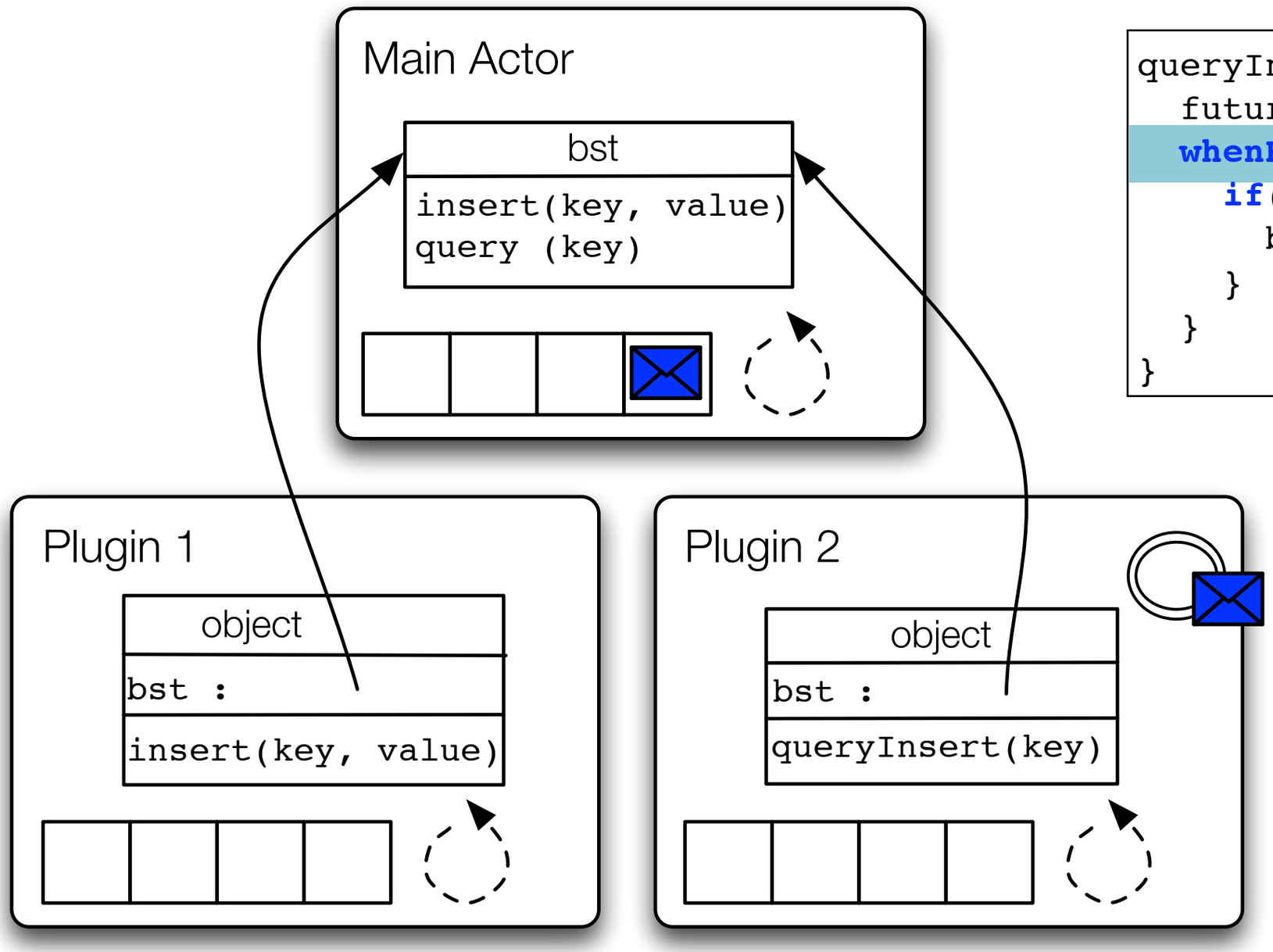

```
queryInsert(key) {
  future := bst<-query(key)
  whenResolved(future) { |value|
    if(value > 0) {
      bst<-insert(key, value - 1)
    }
  }
}
```

Motivating example



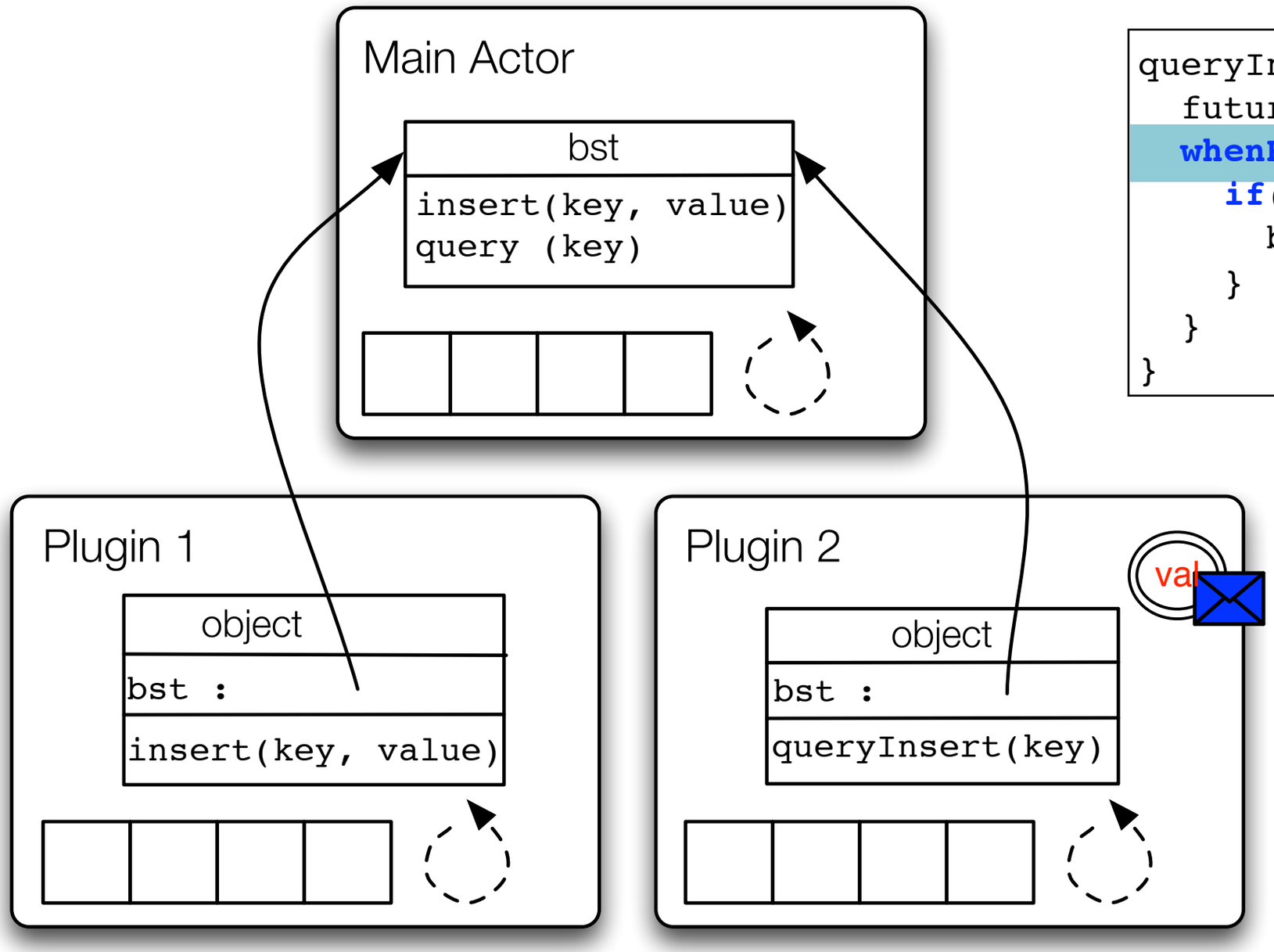

```
queryInsert(key) {
  future := bst<-query(key)
  whenResolved(future) { |value|
    if(value > 0) {
      bst<-insert(key, value - 1)
    }
  }
}
```

Motivating example



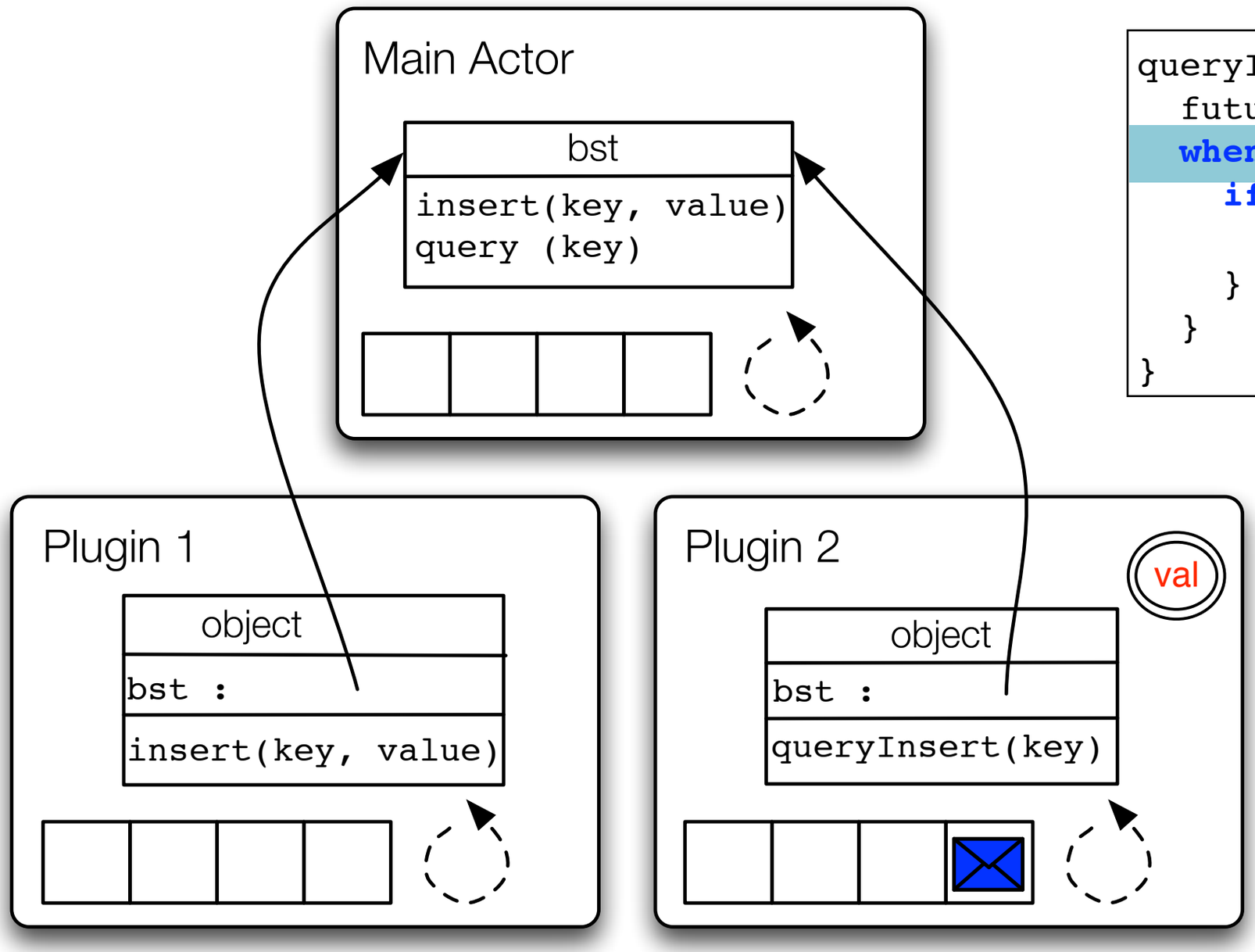

```
queryInsert(key) {
  future := bst<-query(key)
  whenResolved(future) { |value|
    if(value > 0) {
      bst<-insert(key, value - 1)
    }
  }
}
```

Motivating example



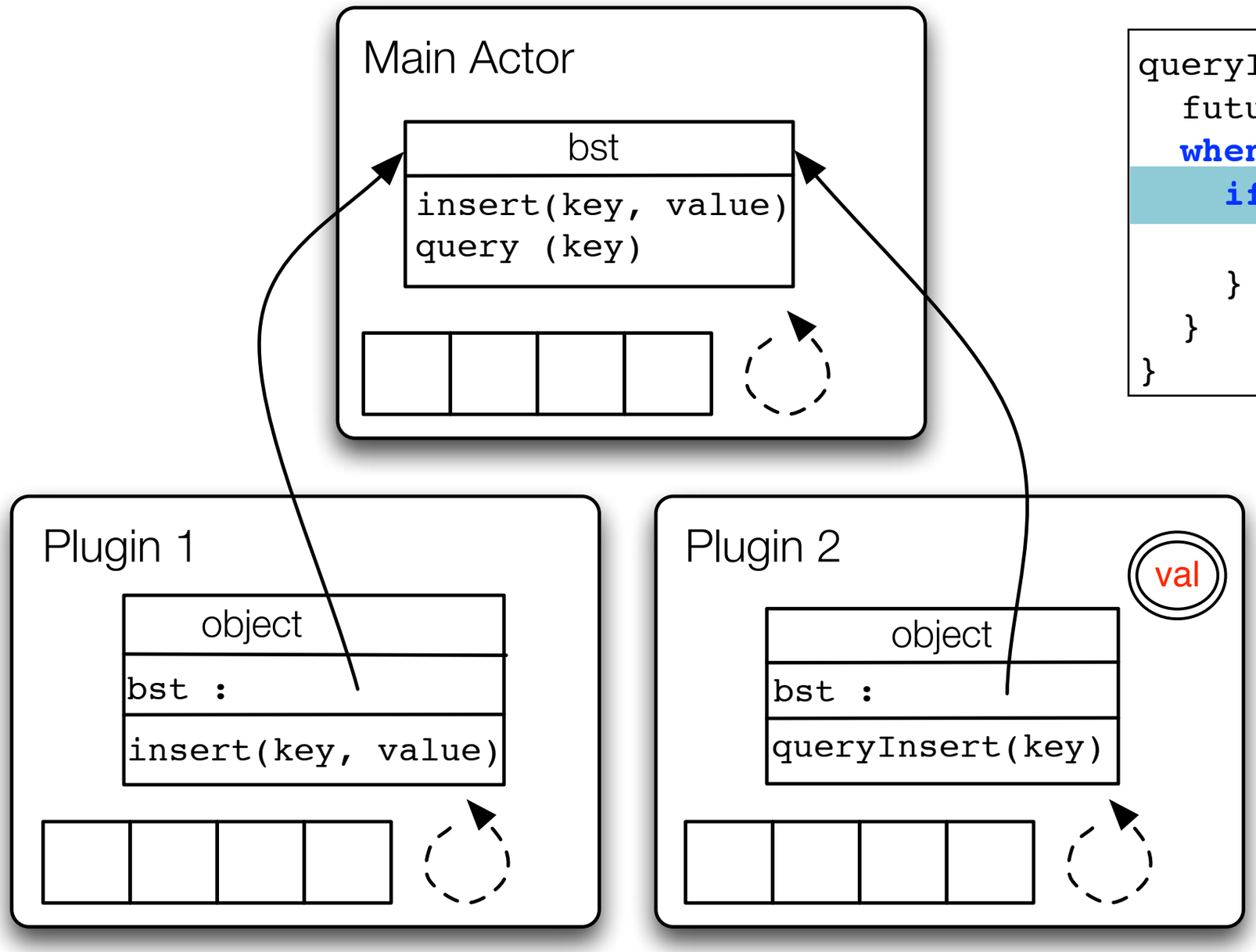

```
queryInsert(key) {
  future := bst<-query(key)
  whenResolved(future) { |value|
    if(value > 0) {
      bst<-insert(key, value - 1)
    }
  }
}
```

Motivating example



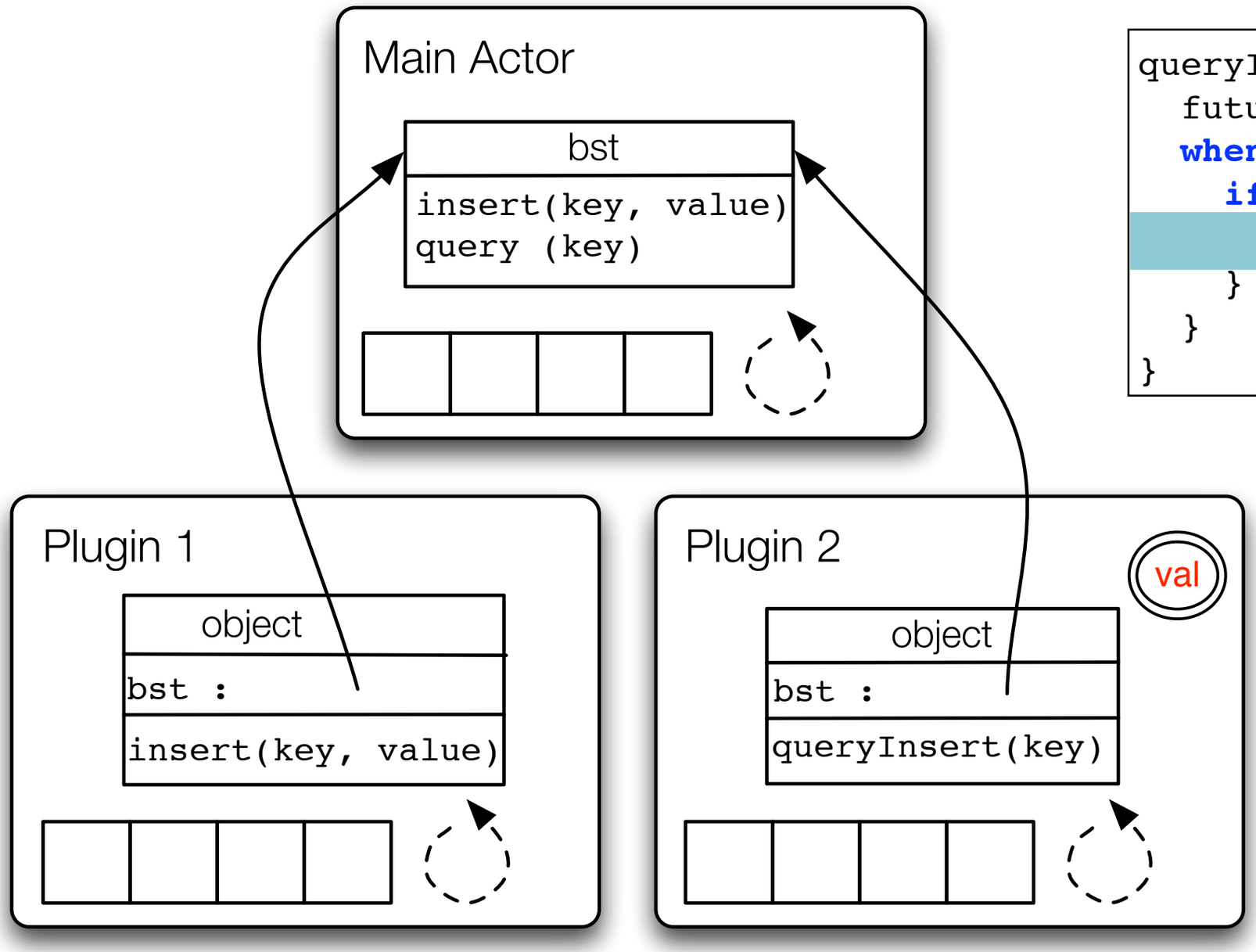

```
queryInsert(key) {
  future := bst<-query(key)
  whenResolved(future) { |value|
    if(value > 0) {
      bst<-insert(key, value - 1)
    }
  }
}
```

Motivating example



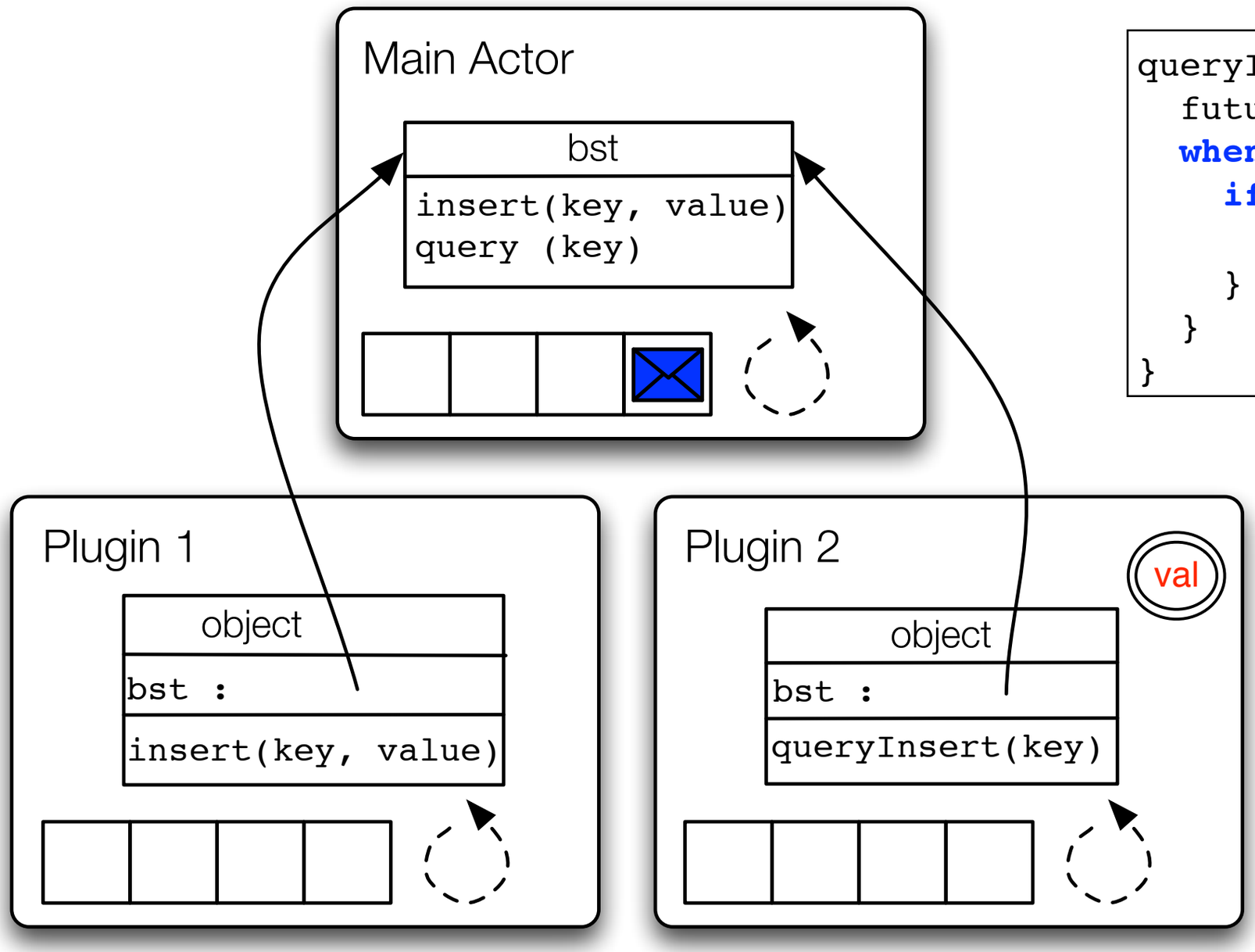

```
queryInsert(key) {
  future := bst<-query(key)
  whenResolved(future) { |value|
    if(value > 0) {
      bst<-insert(key, value - 1)
    }
  }
}
```

Motivating example



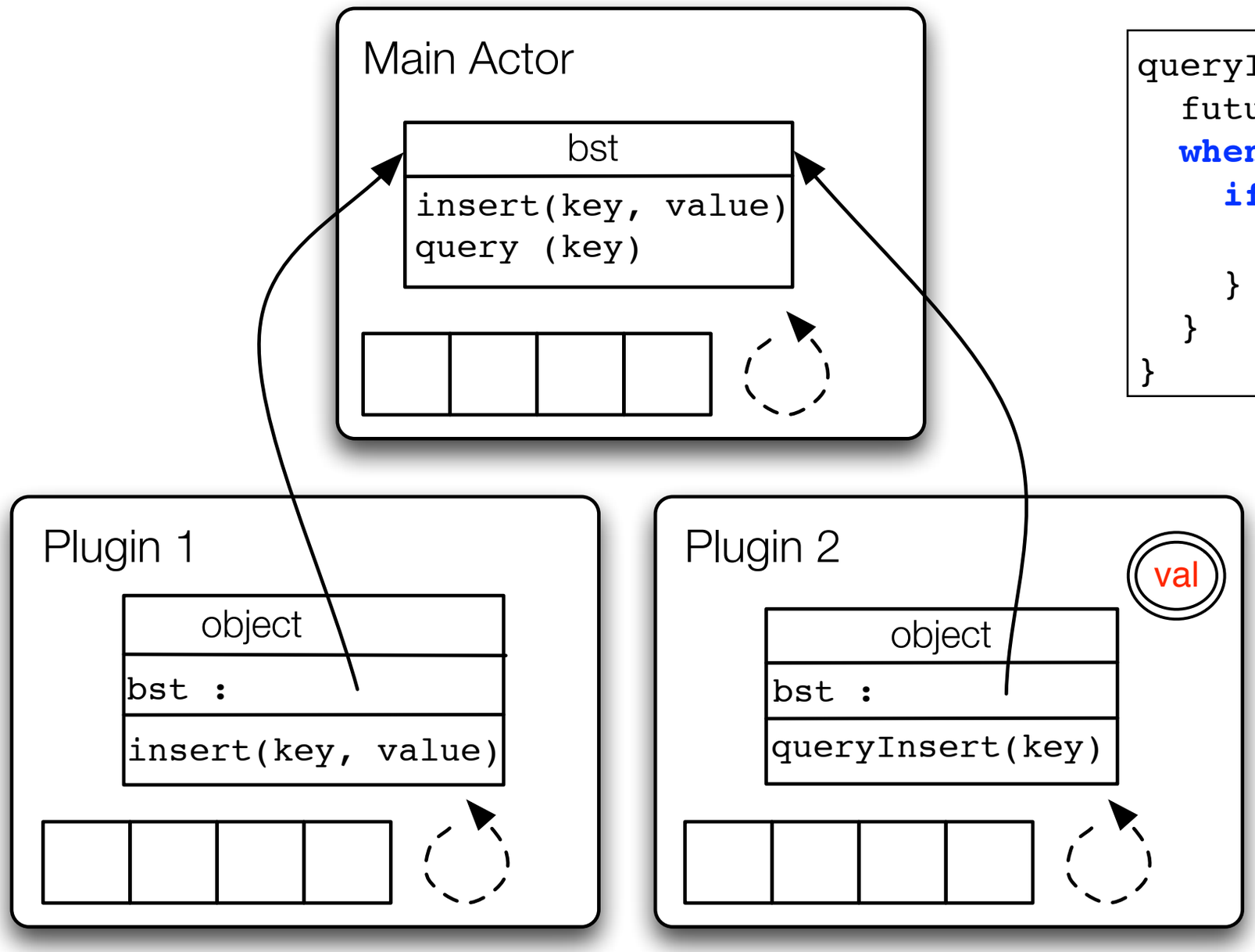

```
queryInsert(key) {
  future := bst<-query(key)
  whenResolved(future) { |value|
    if(value > 0) {
      bst<-insert(key, value - 1)
    }
  }
}
```

Motivating example



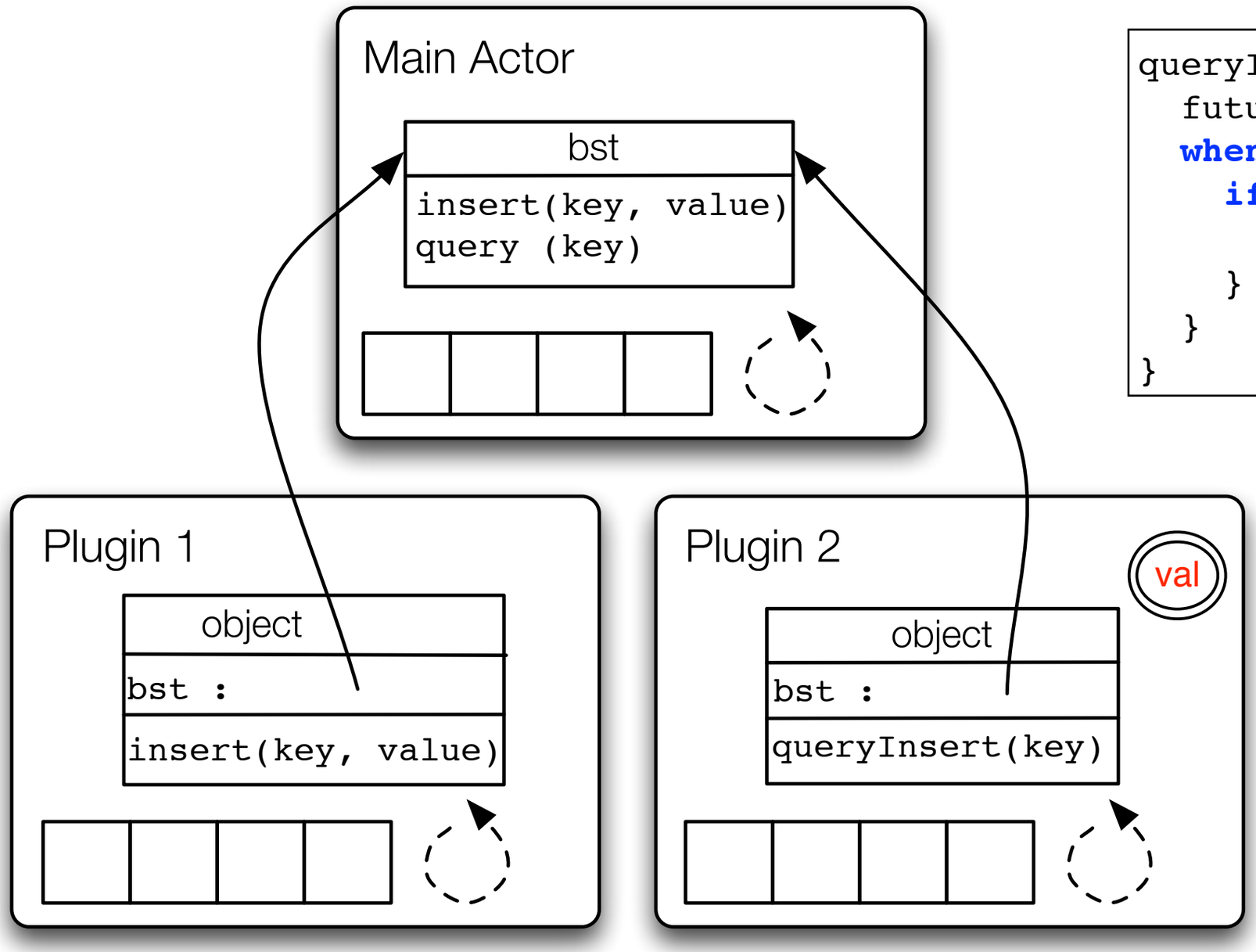

```
queryInsert(key) {
  future := bst<-query(key)
  whenResolved(future) { |value|
    if(value > 0) {
      bst<-insert(key, value - 1)
    }
  }
}
```

Motivating example




```
queryInsert(key) {
  future := bst<-query(key)
  whenResolved(future) { |value|
    if(value > 0) {
      bst<-insert(key, value - 1)
    }
  }
}
```

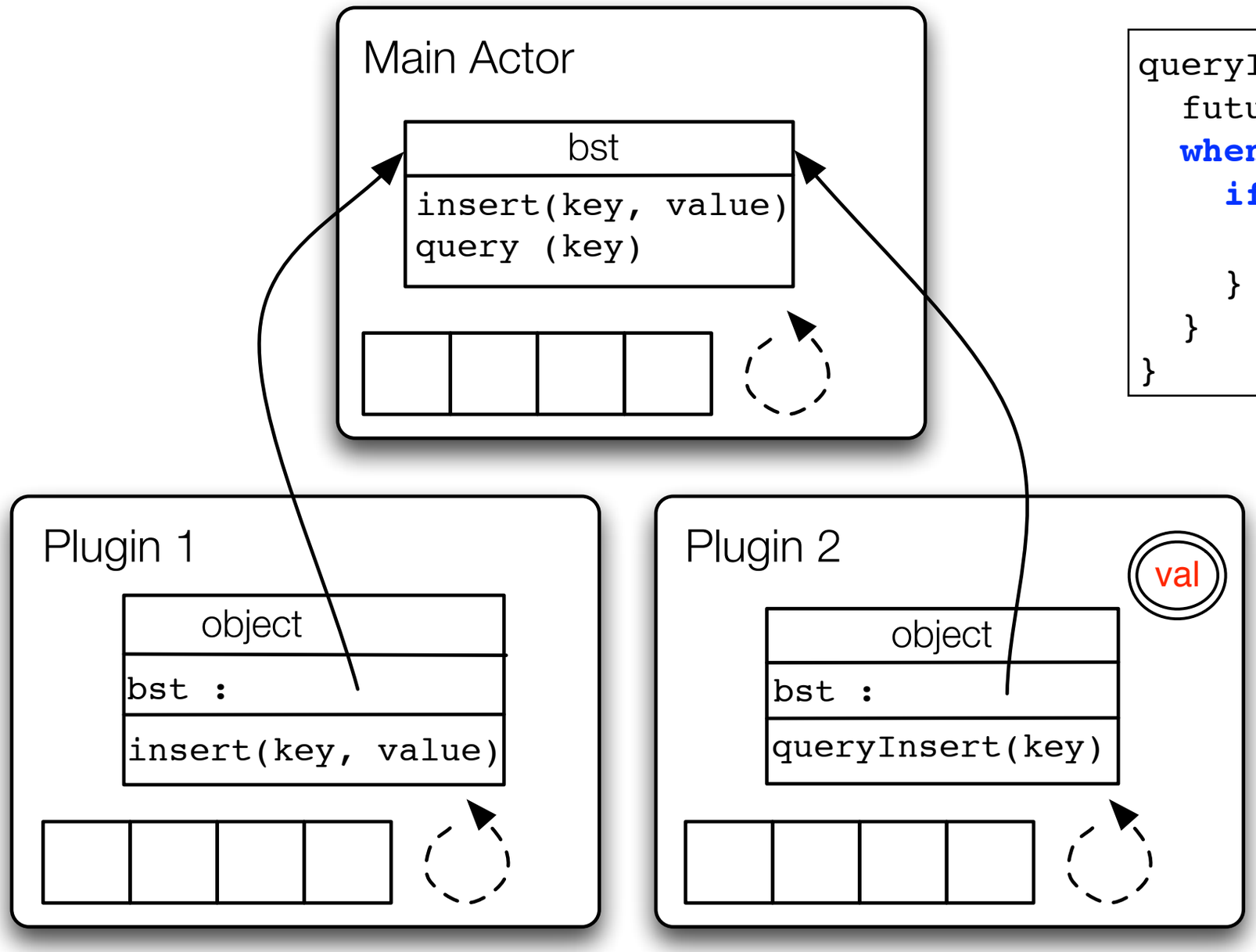
Motivating example




```
queryInsert(key) {
  future := bst<-query(key)
  whenResolved(future) { |value|
    if(value > 0) {
      bst<-insert(key, value - 1)
    }
  }
}
```

1. CPS code

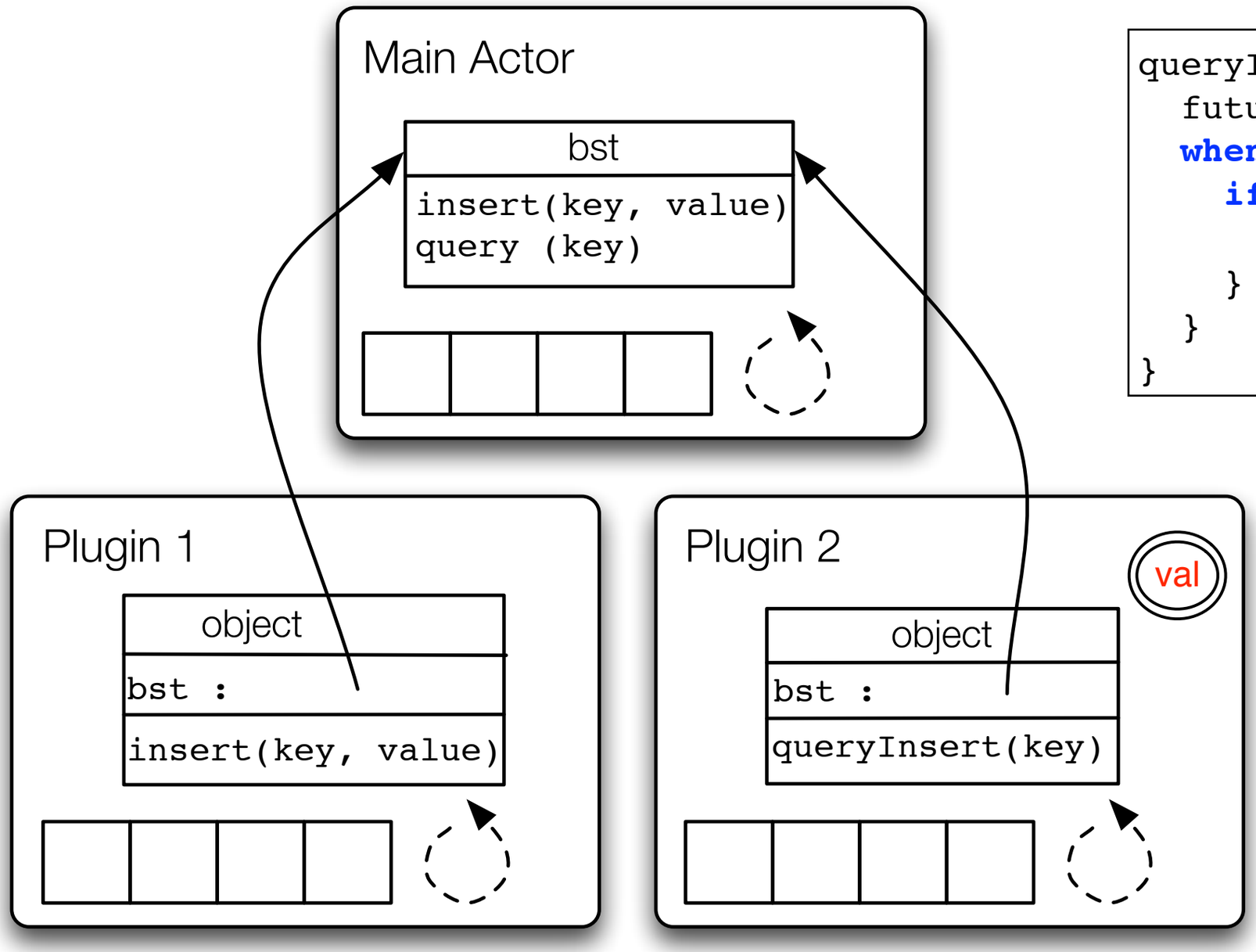
Motivating example




```
queryInsert(key) {
  future := bst<-query(key)
  whenResolved(future) { |value|
    if(value > 0) {
      bst<-insert(key, value - 1)
    }
  }
}
```

1. CPS code
2. No synchronization

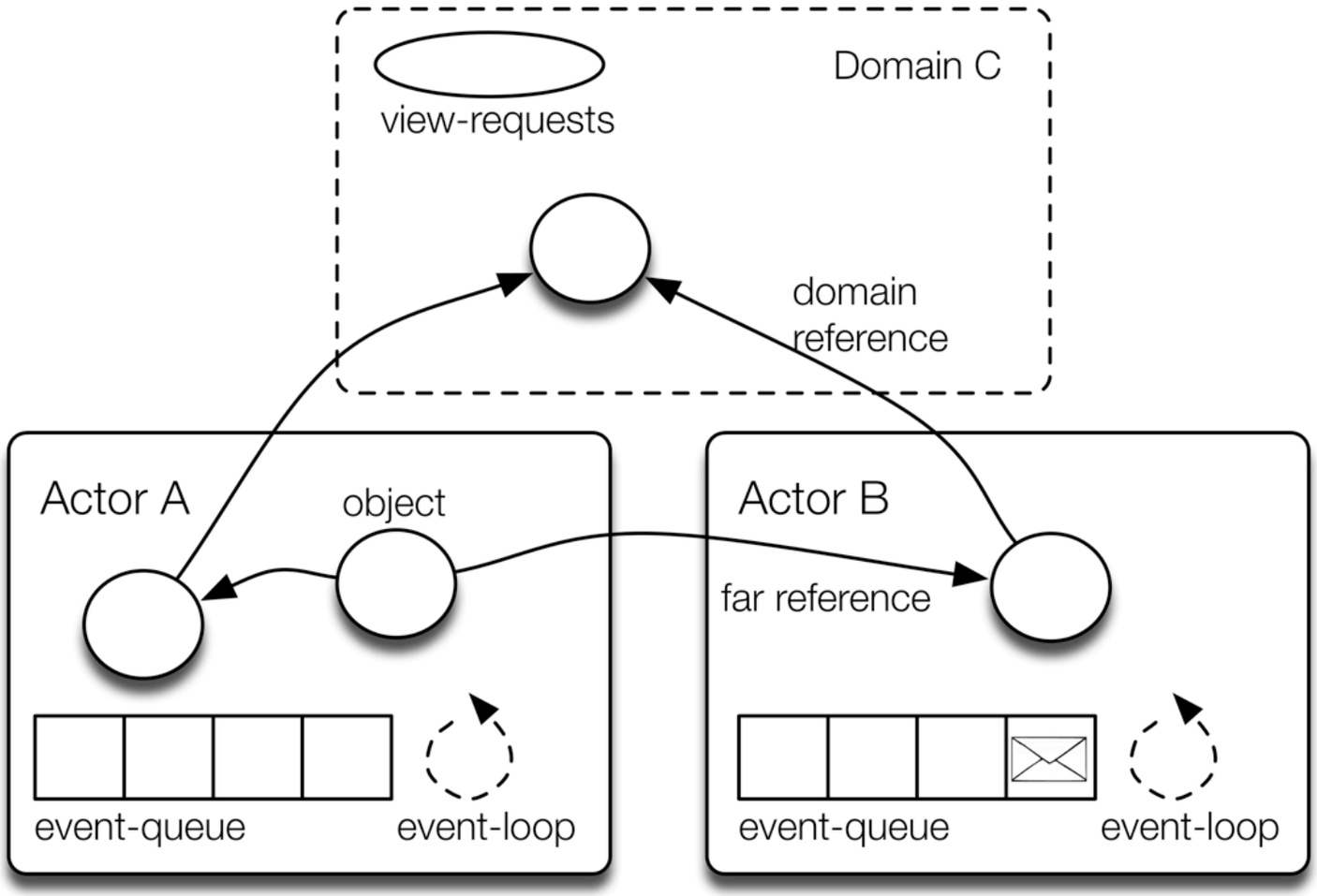
Motivating example




```
queryInsert(key) {
  future := bst<-query(key)
  whenResolved(future) { |value|
    if(value > 0) {
      bst<-insert(key, value - 1)
    }
  }
}
```

1. CPS code
2. No synchronization
3. No parallel reads

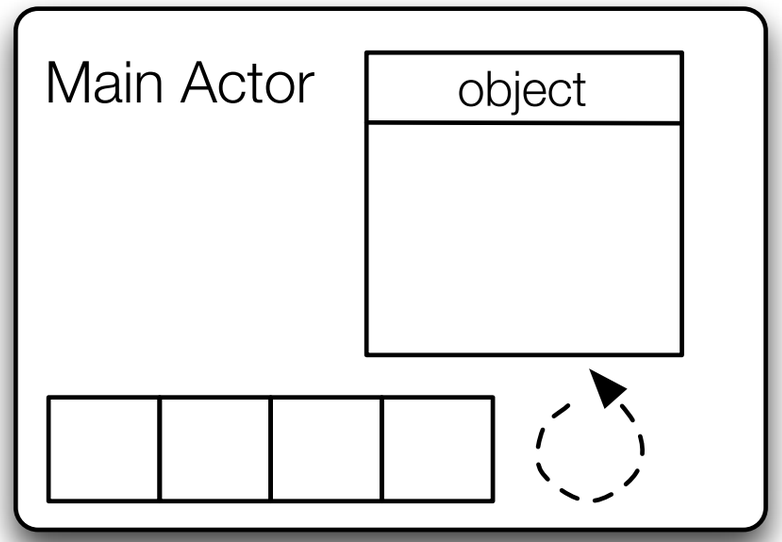
Domains



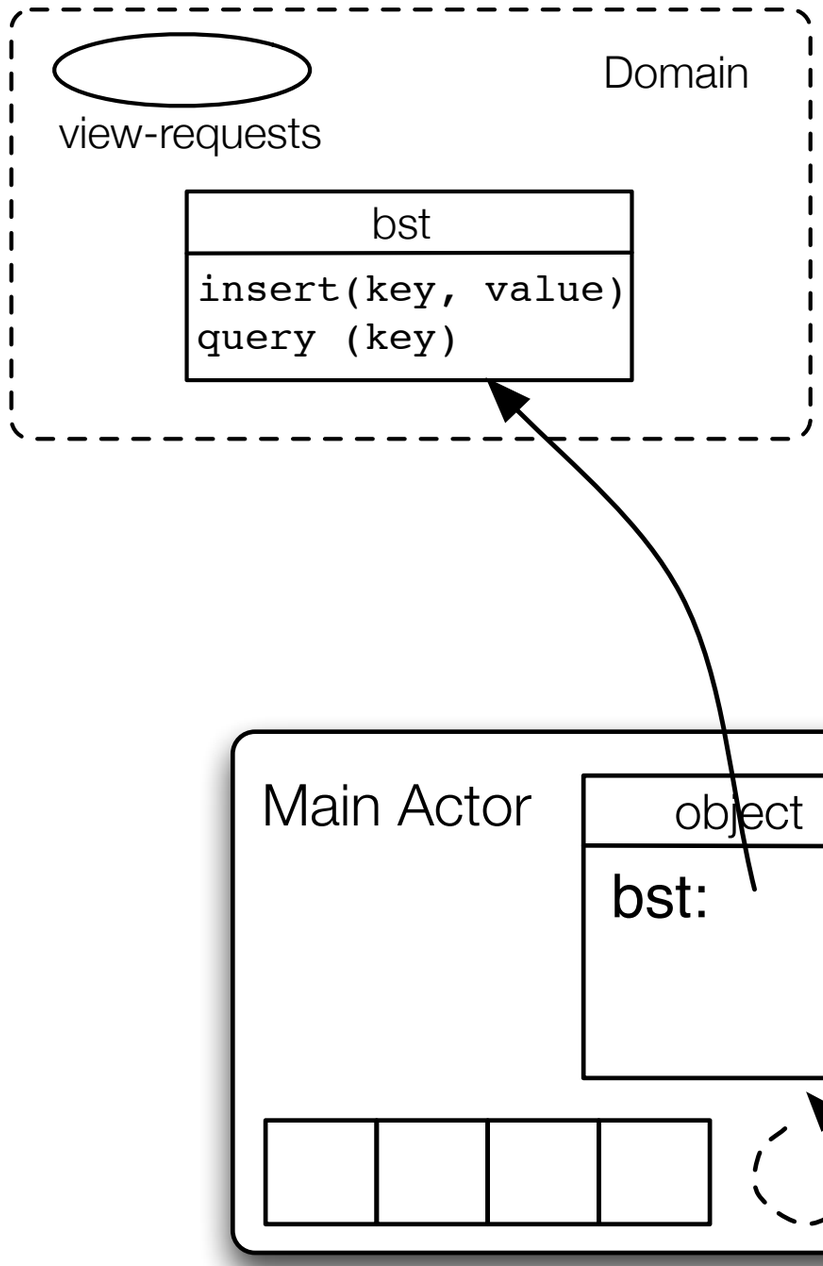

```
domain {
  <expression>+
}

whenShared(<domain>) {
  <expression>+
}

whenExclusive(<domain>) {
  <expression>+
}

when(<domain>+, <domain>+) {
  <expression>+
}
```

Motivating example

```
let bst = domain { ... }

let plugin1 = actor { ... }

let plugin2 = actor {
  queryInsert(bst, key) {
    whenExclusive(bst) {
      value := bst.query(key);
      if(value > 0) {
        bst.insert(key, value - 1)
      }
    }
  }
}

plugin2<-queryInsert(bst, 42)
```



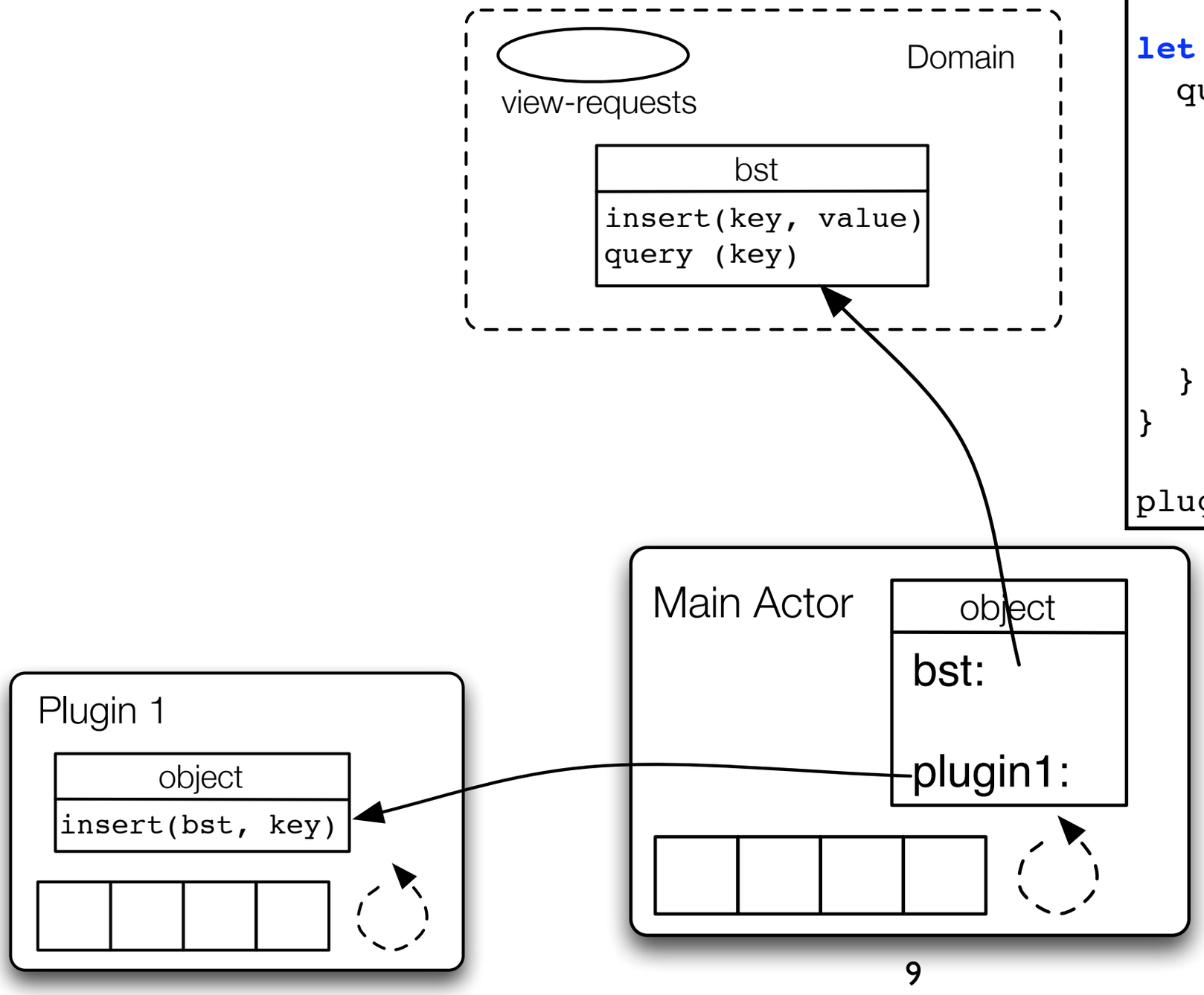
Motivating example



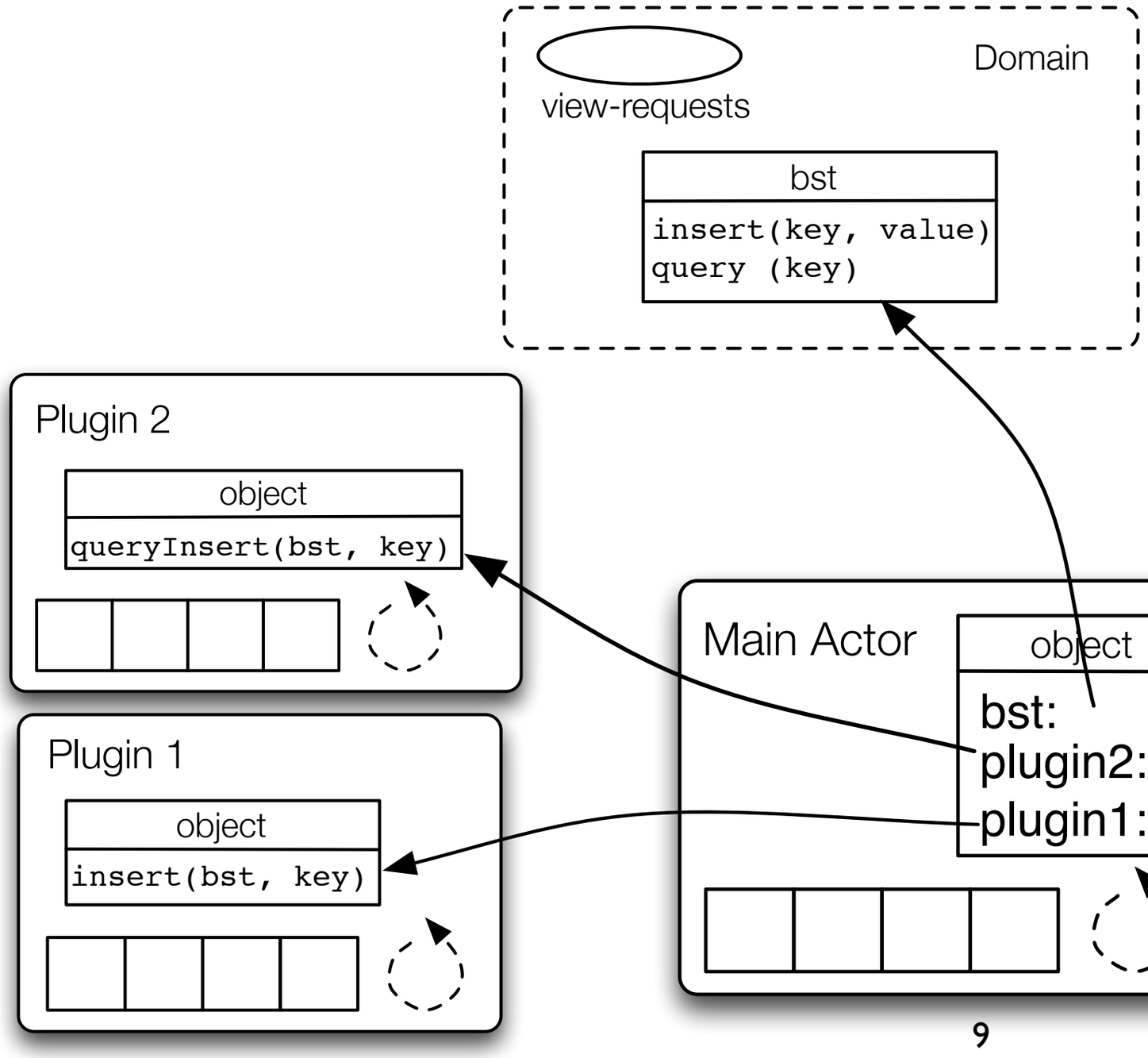

```
let bst = domain { ... }

let plugin1 = actor { ... }

let plugin2 = actor {
  queryInsert(bst, key) {
    whenExclusive(bst) {
      value := bst.query(key);
      if(value > 0) {
        bst.insert(key, value - 1)
      }
    }
  }
}

plugin2<-queryInsert(bst, 42)
```

Motivating example

```
let bst = domain { ... }

let plugin1 = actor { ... }

let plugin2 = actor {
  queryInsert(bst, key) {
    whenExclusive(bst) {
      value := bst.query(key);
      if(value > 0) {
        bst.insert(key, value - 1)
      }
    }
  }
}

plugin2<-queryInsert(bst, 42)
```

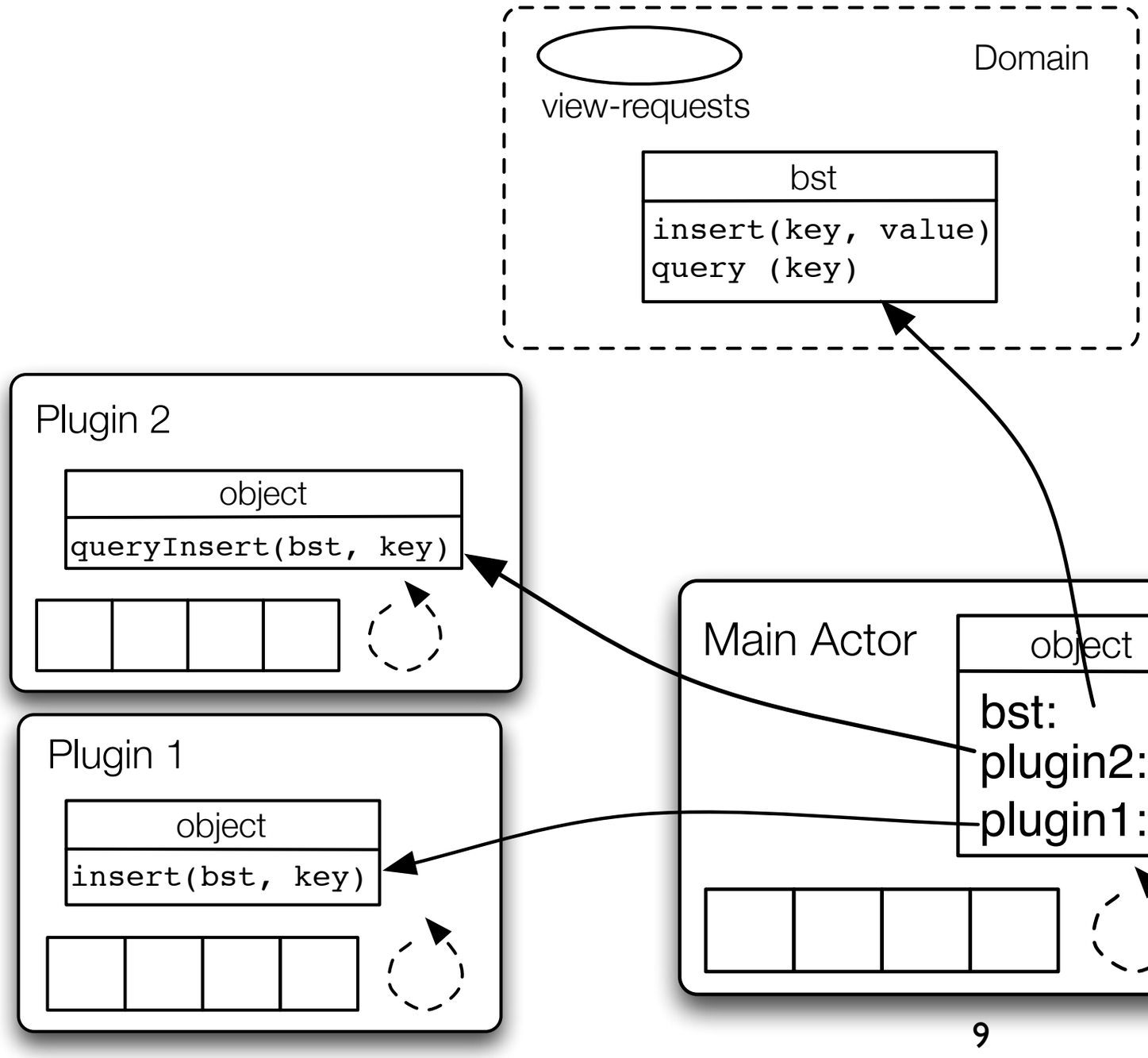


Motivating example



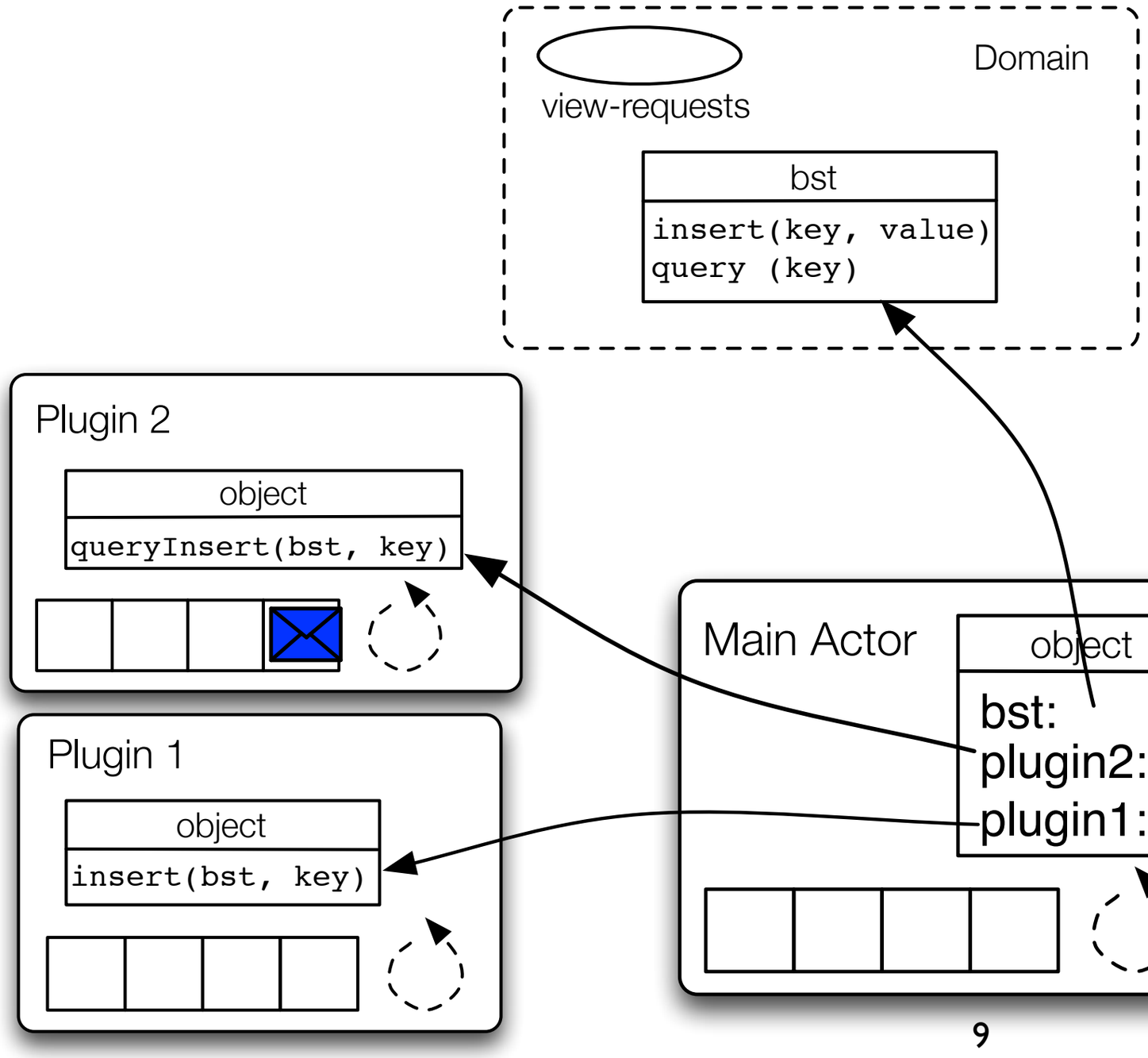

```
let bst = domain { ... }

let plugin1 = actor { ... }

let plugin2 = actor {
  queryInsert(bst, key) {
    whenExclusive(bst) {
      value := bst.query(key);
      if(value > 0) {
        bst.insert(key, value - 1)
      }
    }
  }
}

plugin2<-queryInsert(bst, 42)
```

Motivating example



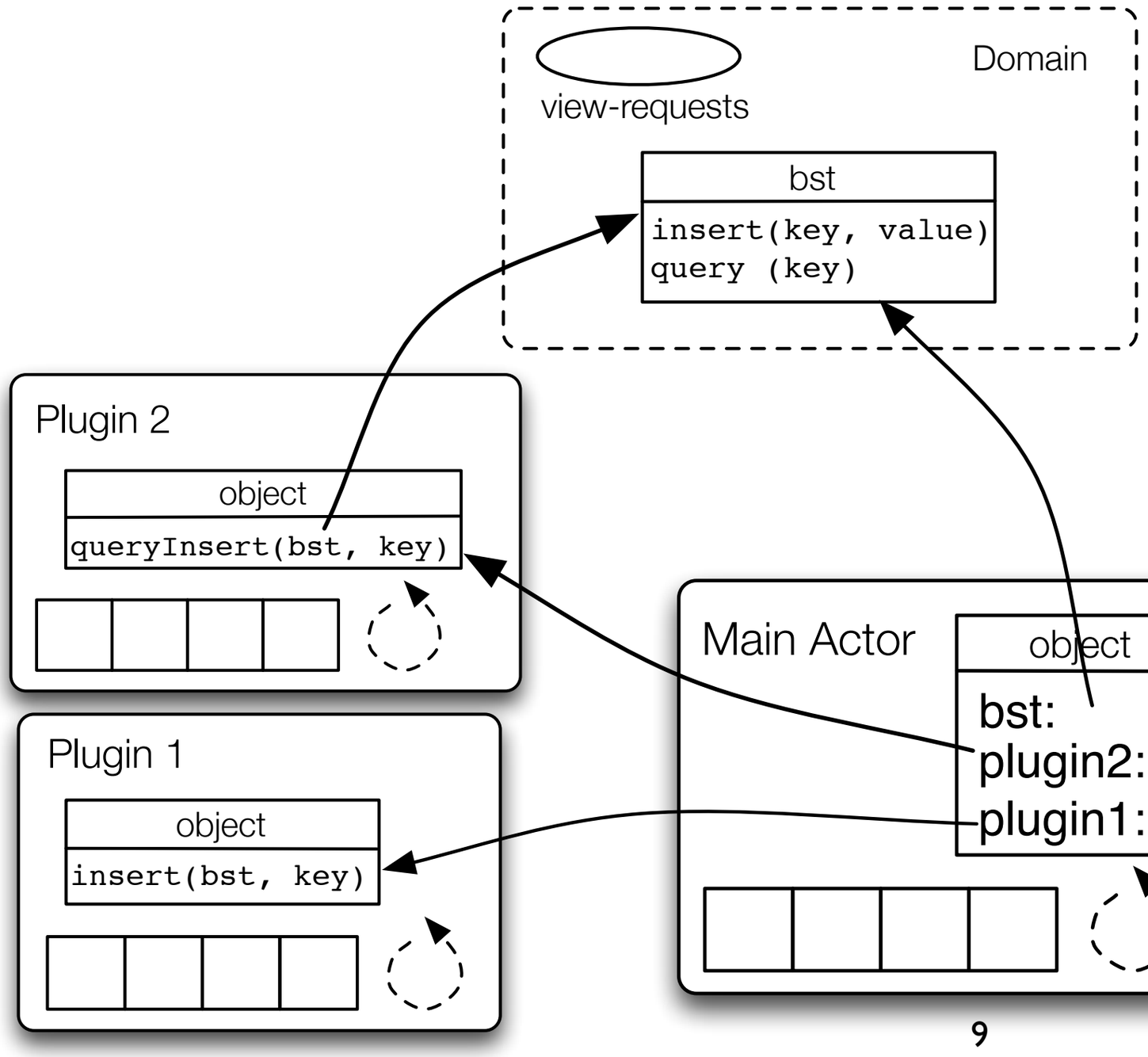

```
let bst = domain { ... }

let plugin1 = actor { ... }

let plugin2 = actor {
  queryInsert(bst, key) {
    whenExclusive(bst) {
      value := bst.query(key);
      if(value > 0) {
        bst.insert(key, value - 1)
      }
    }
  }
}

plugin2<-queryInsert(bst, 42)
```

Motivating example



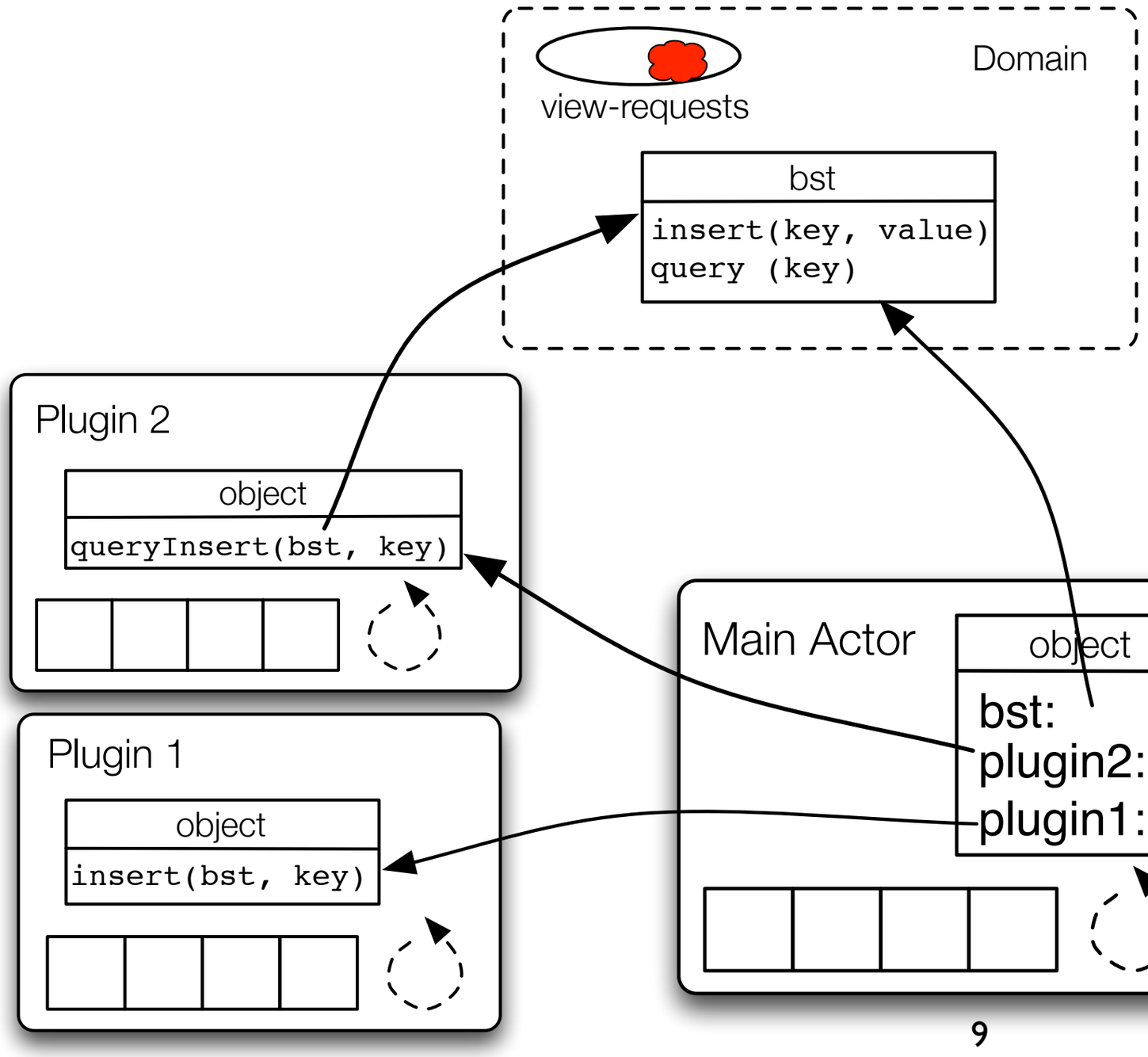

```
let bst = domain { ... }

let plugin1 = actor { ... }

let plugin2 = actor {
  queryInsert(bst, key) {
    whenExclusive(bst) {
      value := bst.query(key);
      if(value > 0) {
        bst.insert(key, value - 1)
      }
    }
  }
}

plugin2<-queryInsert(bst, 42)
```

Motivating example



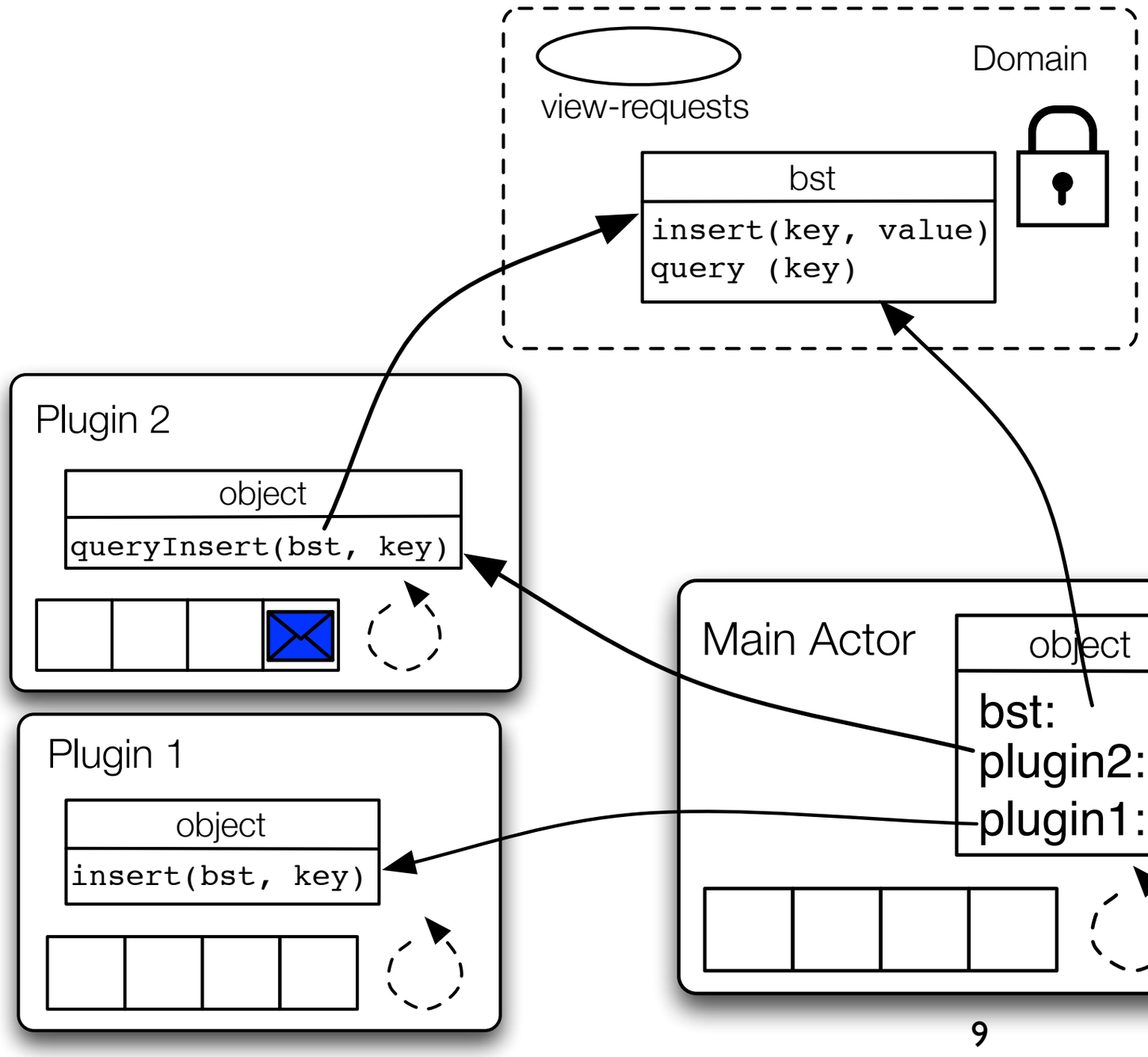

```
let bst = domain { ... }

let plugin1 = actor { ... }

let plugin2 = actor {
  queryInsert(bst, key) {
    whenExclusive(bst) {
      value := bst.query(key);
      if(value > 0) {
        bst.insert(key, value - 1)
      }
    }
  }
}

plugin2<-queryInsert(bst, 42)
```

Motivating example



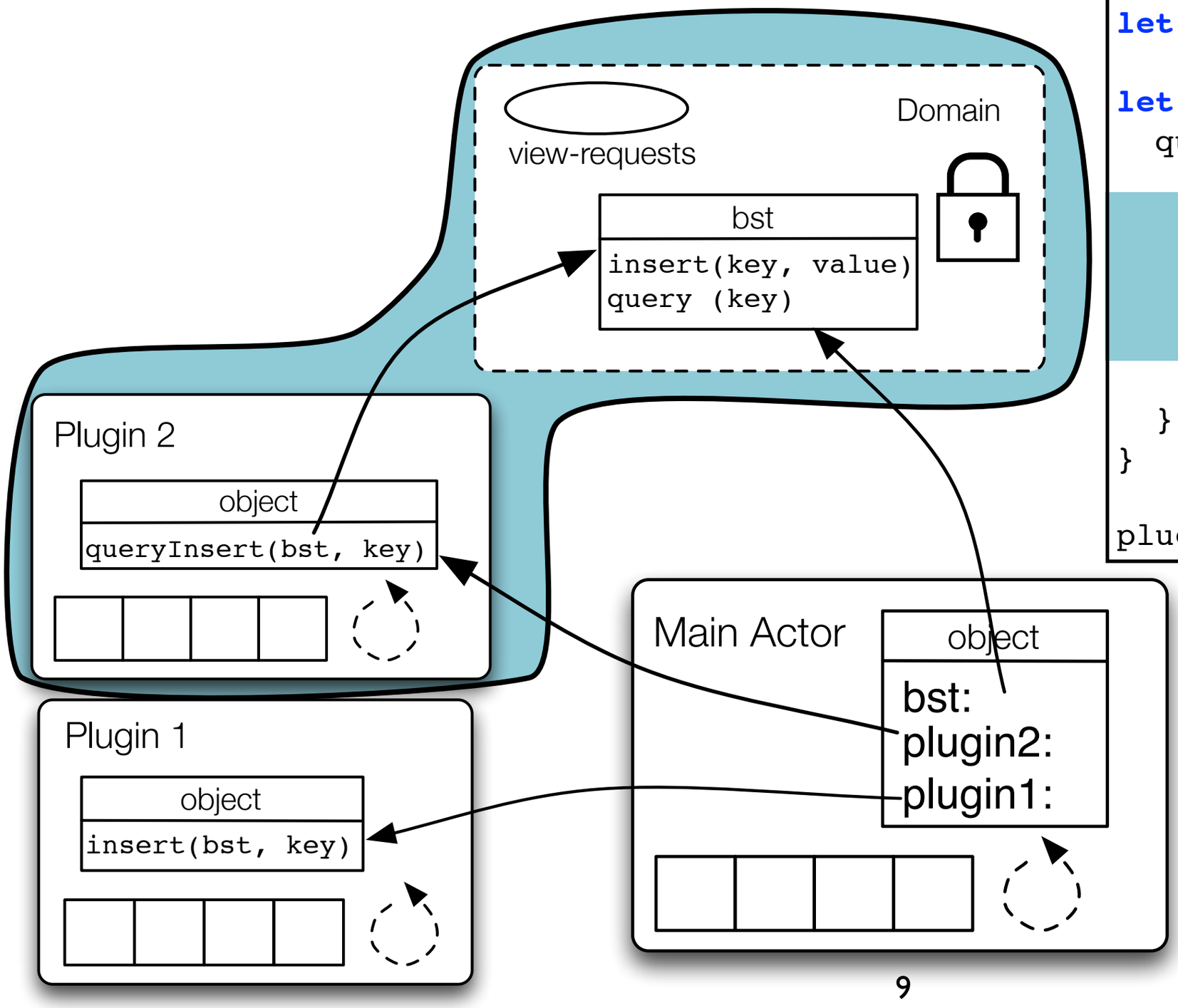

```
let bst = domain { ... }

let plugin1 = actor { ... }

let plugin2 = actor {
  queryInsert(bst, key) {
    whenExclusive(bst) {
      value := bst.query(key);
      if(value > 0) {
        bst.insert(key, value - 1)
      }
    }
  }
}

plugin2<-queryInsert(bst, 42)
```

Motivating example



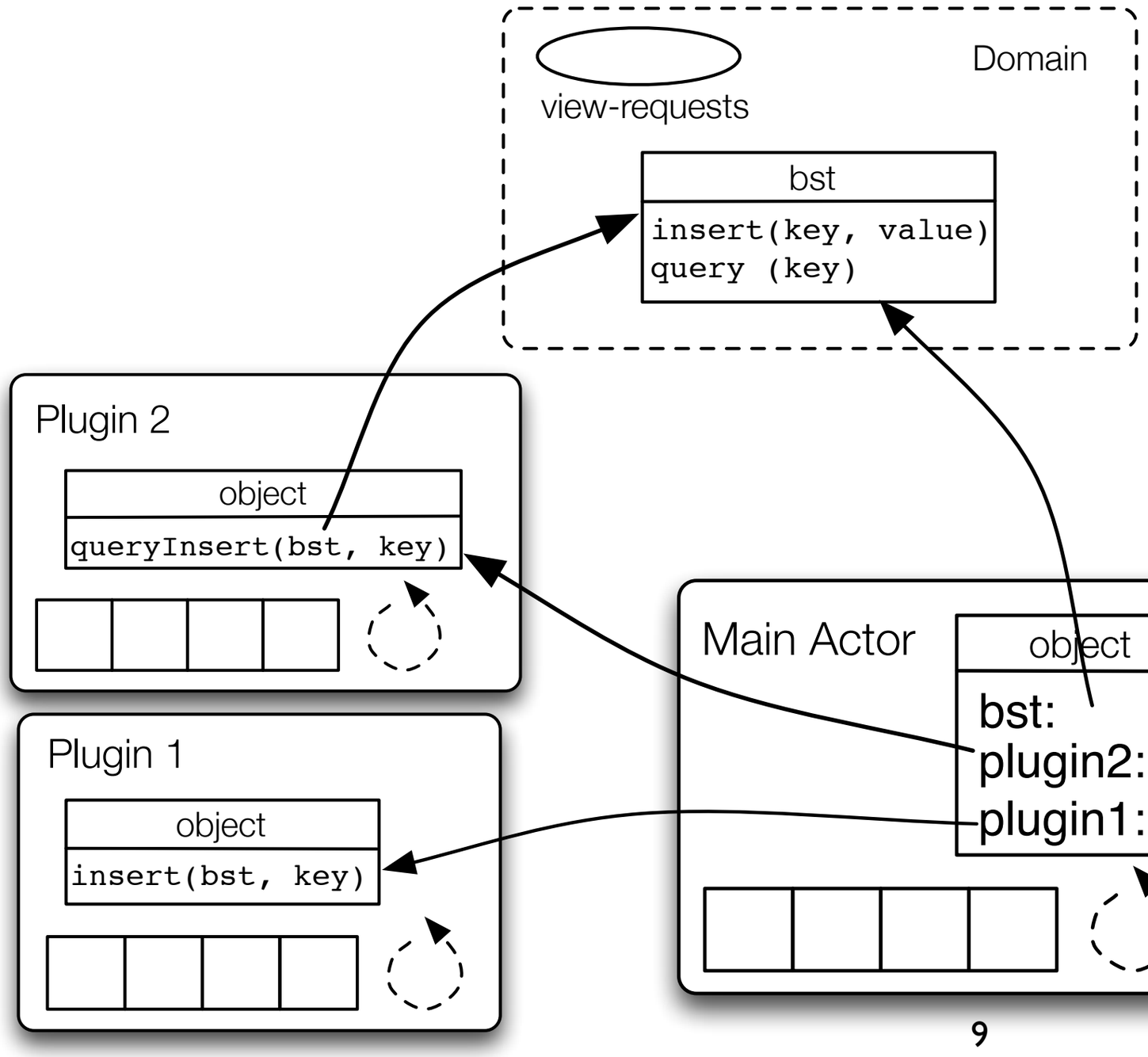

```
let bst = domain { ... }

let plugin1 = actor { ... }

let plugin2 = actor {
  queryInsert(bst, key) {
    whenExclusive(bst) {
      value := bst.query(key);
      if(value > 0) {
        bst.insert(key, value - 1)
      }
    }
  }
}

plugin2<-queryInsert(bst, 42)
```

Motivating example



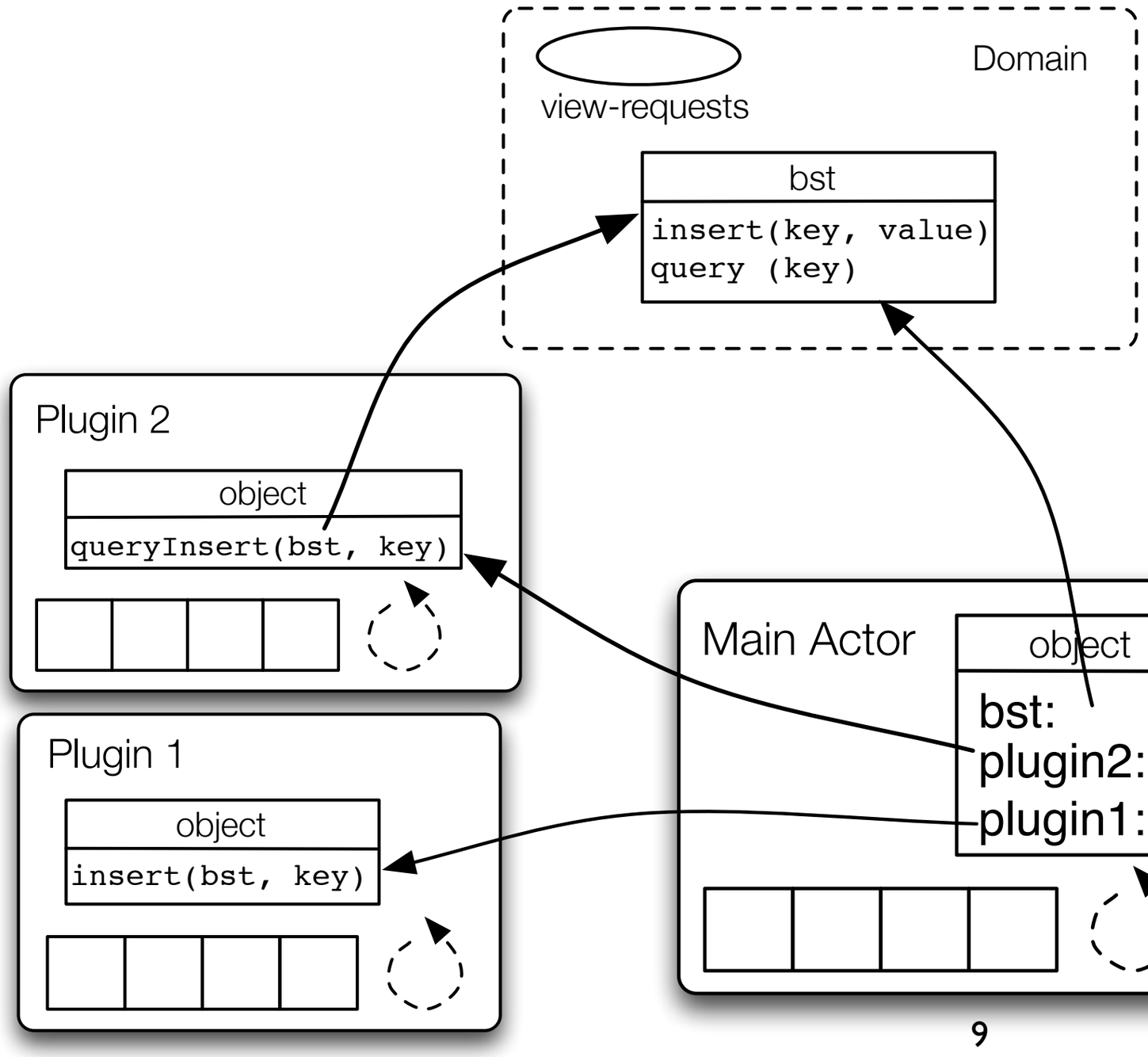

```
let bst = domain { ... }

let plugin1 = actor { ... }

let plugin2 = actor {
  queryInsert(bst, key) {
    whenExclusive(bst) {
      value := bst.query(key);
      if(value > 0) {
        bst.insert(key, value - 1)
      }
    }
  }
}

plugin2<-queryInsert(bst, 42)
```

Motivating example




```
let bst = domain { ... }

let plugin1 = actor { ... }

let plugin2 = actor {
  queryInsert(bst, key) {
    whenExclusive(bst) {
      value := bst.query(key);
      if(value > 0) {
        bst.insert(key, value - 1)
      }
    }
  }
}

plugin2<-queryInsert(bst, 42)
```

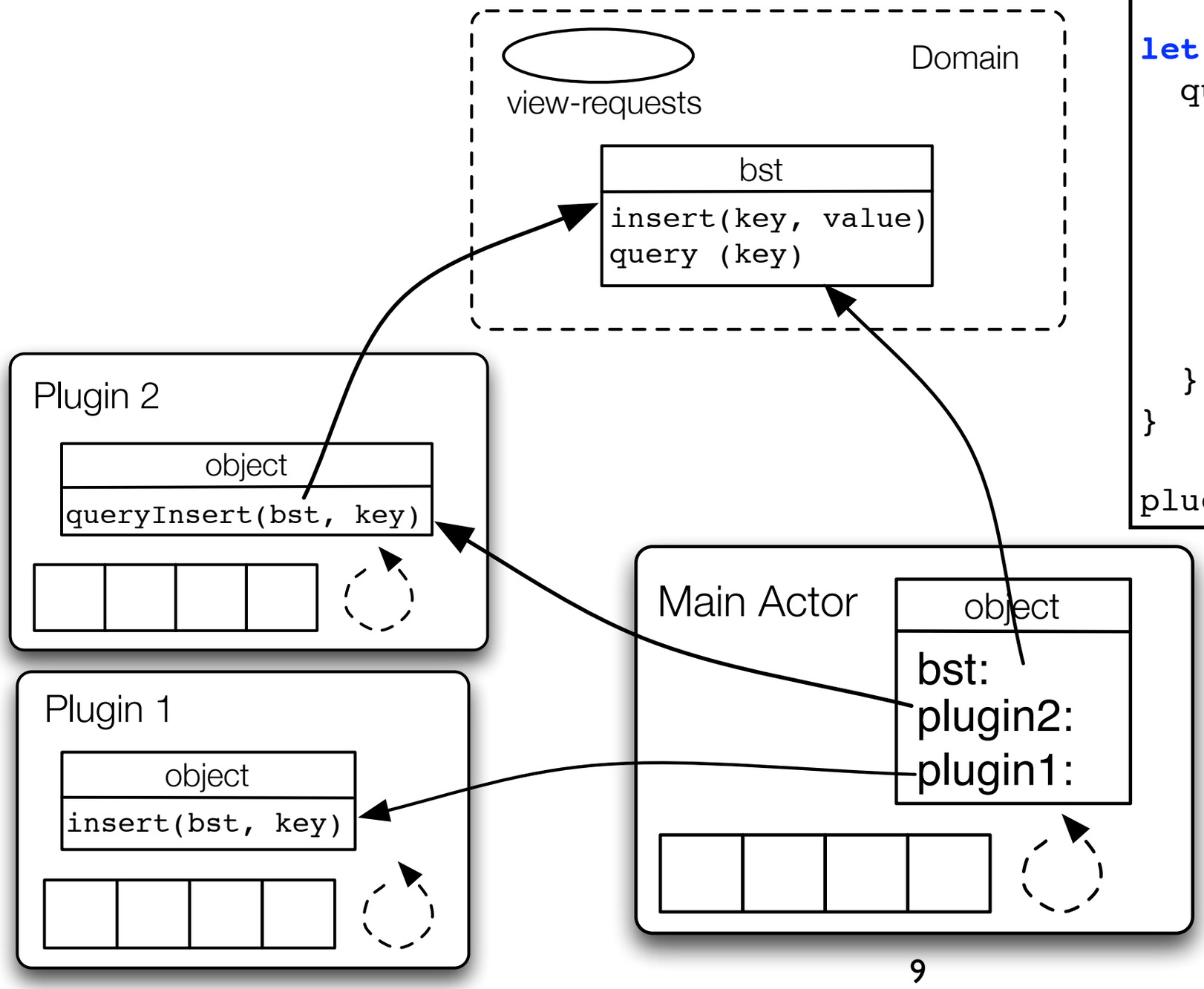
Motivating example




```
let bst = domain { ... }

let plugin1 = actor { ... }

let plugin2 = actor {
  queryInsert(bst, key) {
    whenExclusive(bst) {
      value := bst.query(key);
      if(value > 0) {
        bst.insert(key, value - 1)
      }
    }
  }
}

plugin2<-queryInsert(bst, 42)
```

1. No CPS code

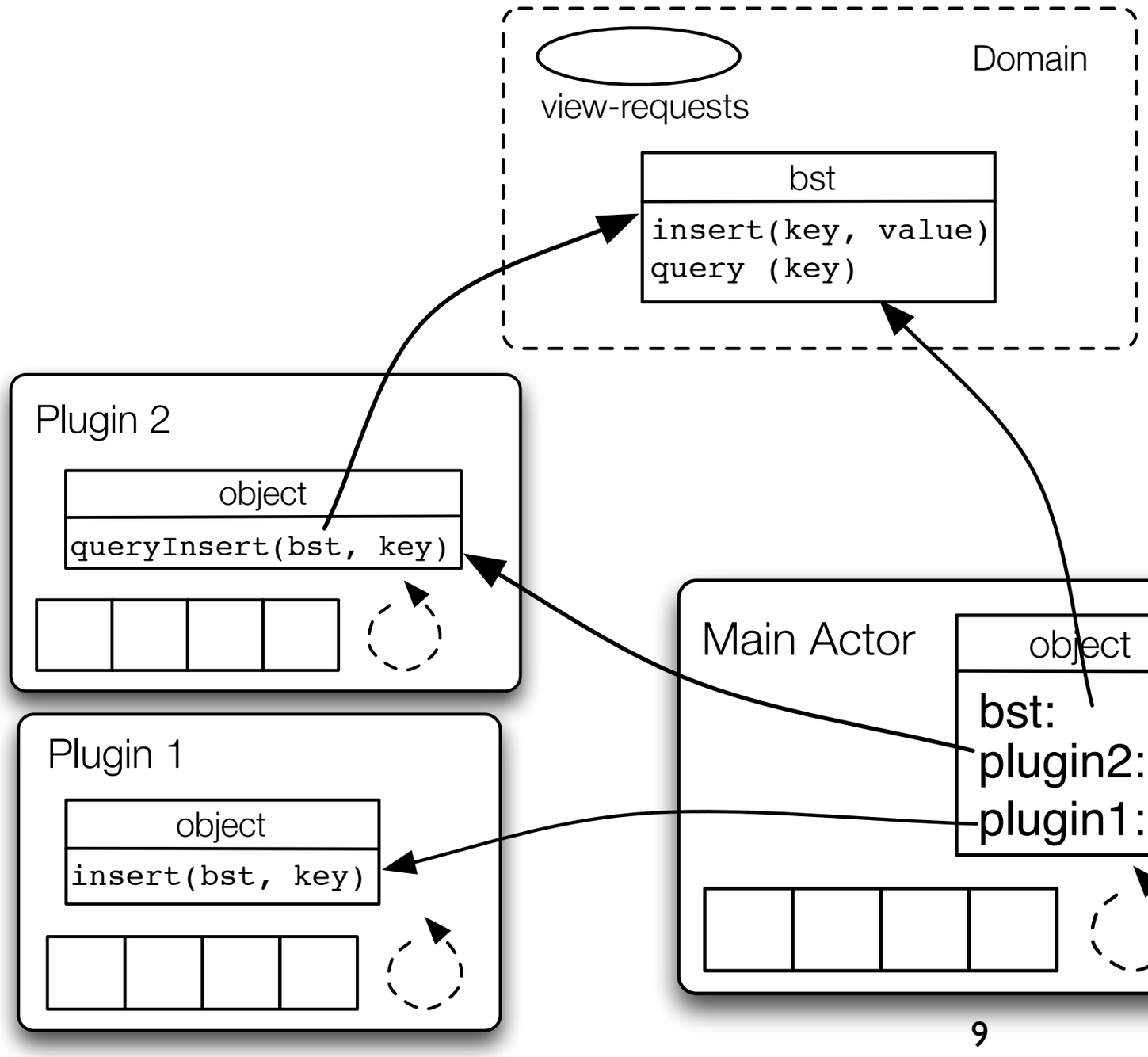
Motivating example




```
let bst = domain { ... }

let plugin1 = actor { ... }

let plugin2 = actor {
  queryInsert(bst, key) {
    whenExclusive(bst) {
      value := bst.query(key);
      if(value > 0) {
        bst.insert(key, value - 1)
      }
    }
  }
}

plugin2<-queryInsert(bst, 42)
```

1. No CPS code
2. Synchronization

Motivating example




```
let bst = domain { ... }

let plugin1 = actor { ... }

let plugin2 = actor {
  queryInsert(bst, key) {
    whenExclusive(bst) {
      value := bst.query(key);
      if(value > 0) {
        bst.insert(key, value - 1)
      }
    }
  }
}

plugin2<-queryInsert(bst, 42)
```

1. No CPS code
2. Synchronization
3. Parallel reads

Conclusion

- **Safe**
  - Isolation properties still valid
  - Deadlock and race-condition free
- **Expressive**
  - Able to express the concept “shared state”
  - No inversion of control
- **Efficient**
  - Synchronous access to shared state
  - Parallel reads through the use of shared views