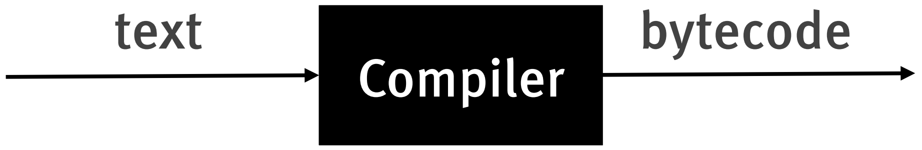
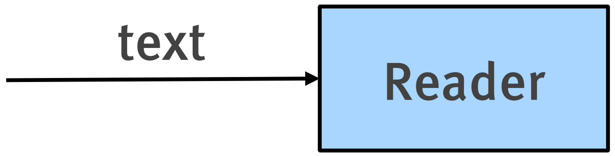
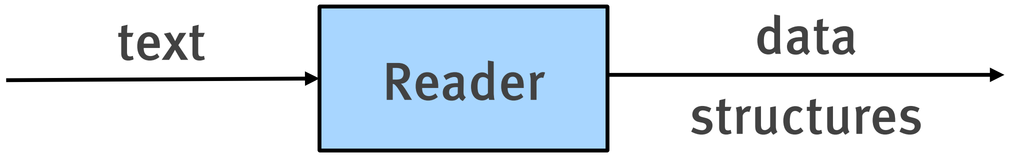
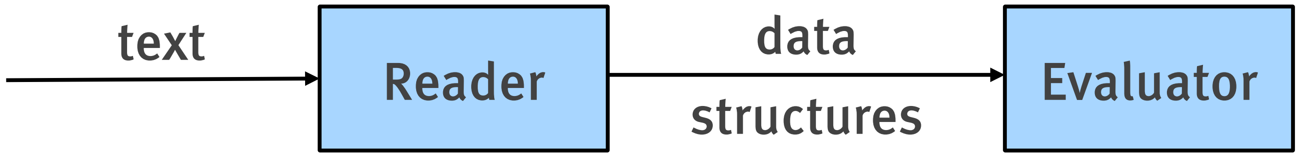
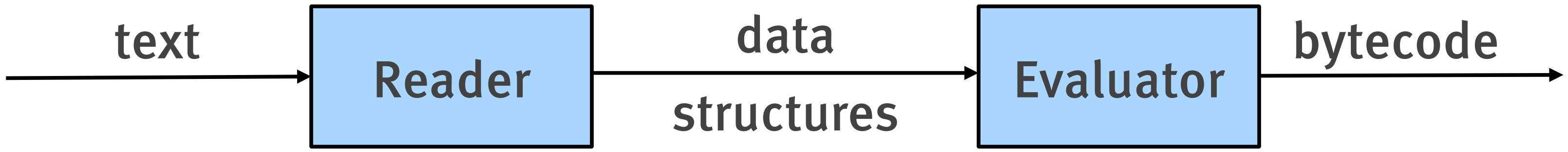

text
Compiler
bytecode

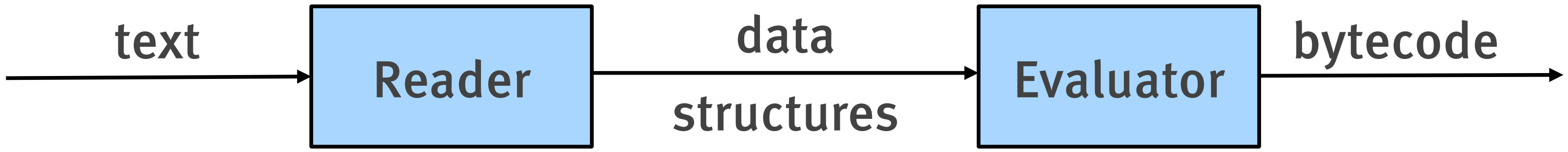
text


text
Reader

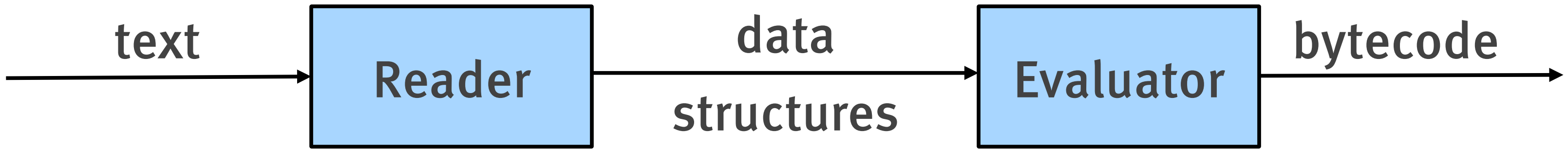

text
Reader
data
structures


text
Reader
data
structures
Evaluator


text
Reader
data
structures
Evaluator
bytecode

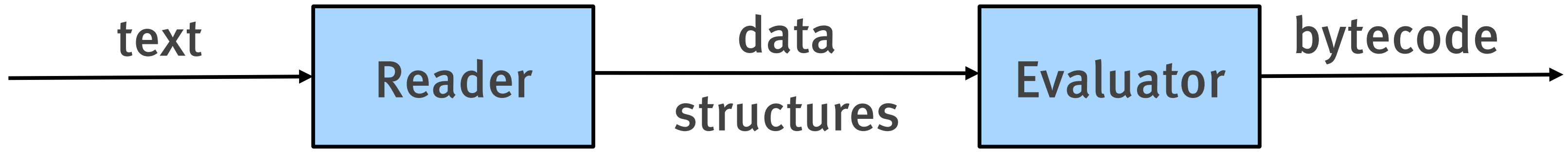


```
“(if true
   (print "true"
   (print "false"))”
```

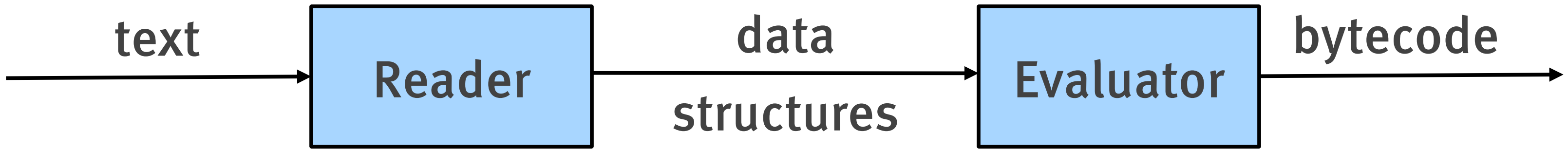


```
“(if true
   (print "true"
   (print "false"))”
```

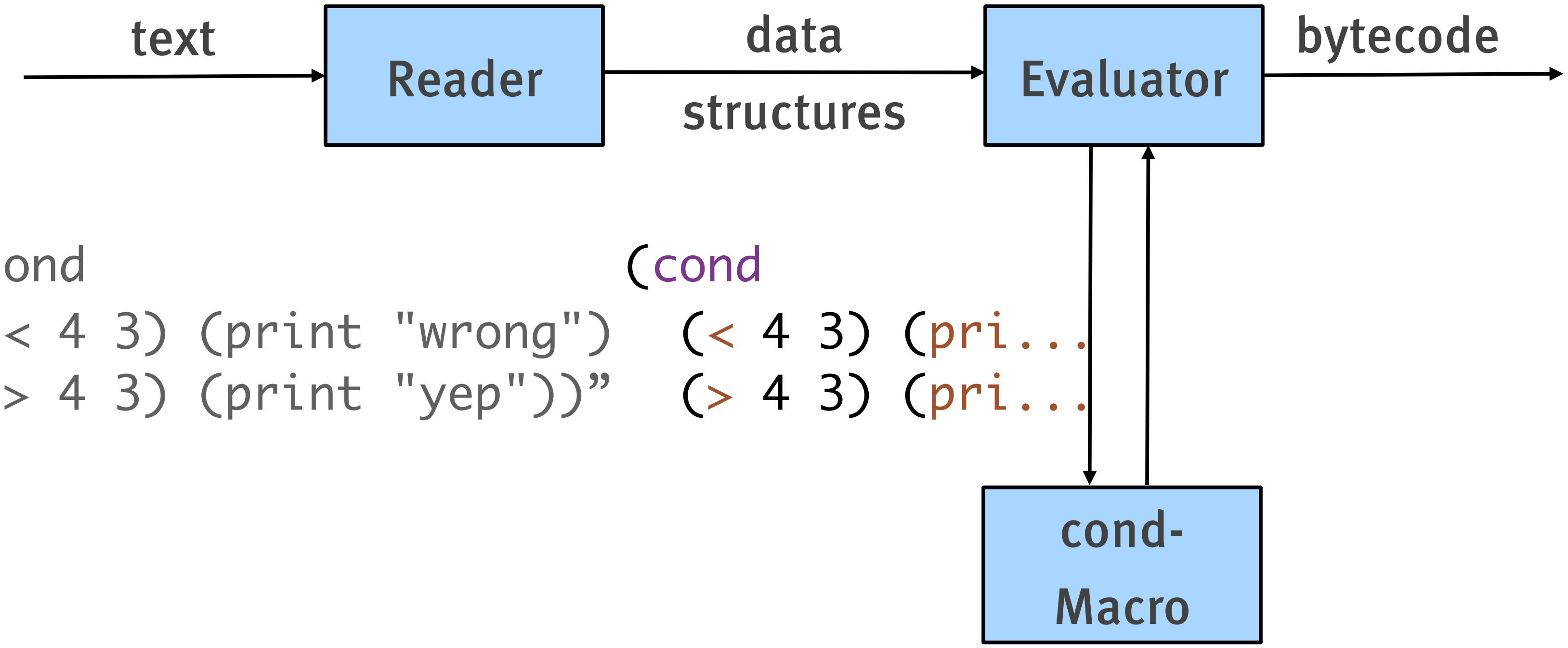
```
(if true
  (print "true"
  (print "false"))
```



```
“(cond
  (< 4 3) (print "wrong")
  (> 4 3) (print "yep"))”
```

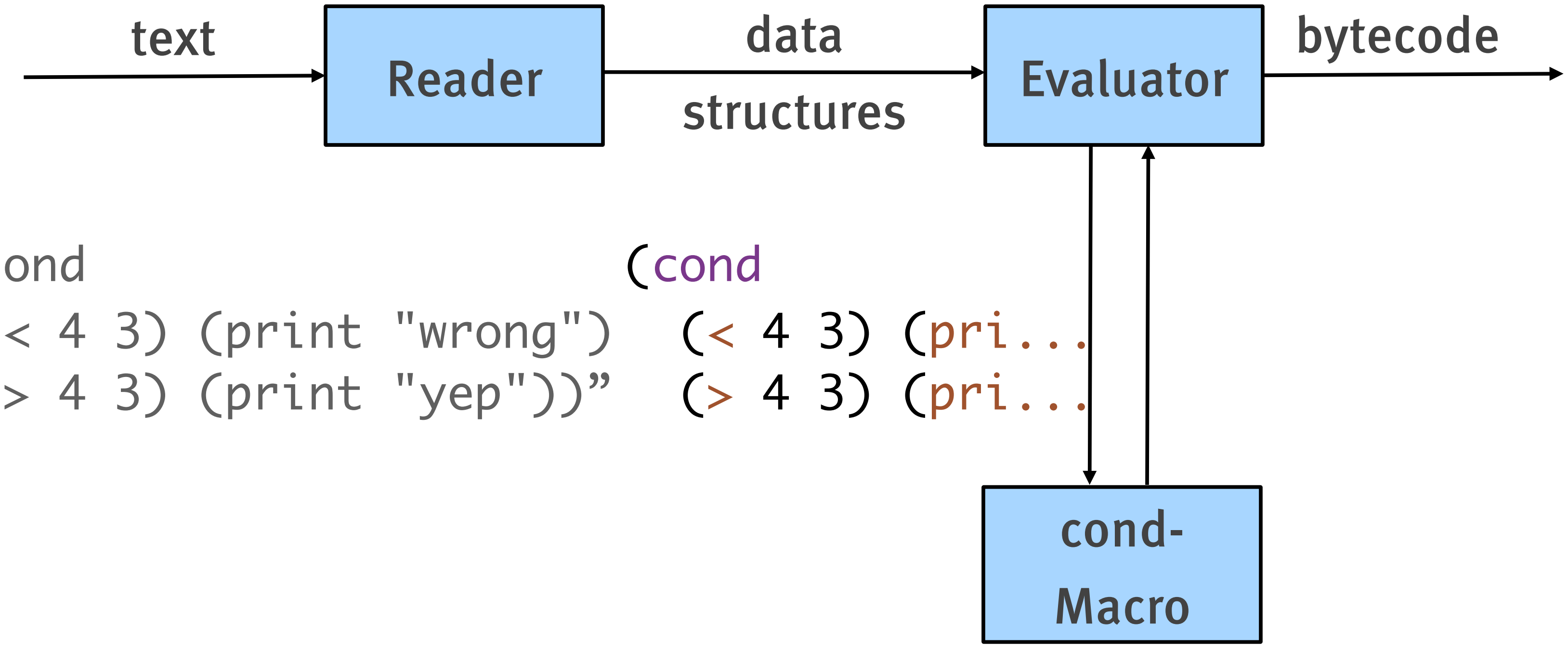


```
"(cond                        (cond
   (< 4 3) (print "wrong")     (< 4 3) (print "wrong")
   (> 4 3) (print "yep"))"     (> 4 3) (print "yep"))
```



```
“(cond
   (< 4 3) (print "wrong")
   (> 4 3) (print "yep"))”
```

```
(cond
  (< 4 3) (pri...
  (> 4 3) (pri...
```



```
"(cond
   (< 4 3) (print "wrong")
   (> 4 3) (print "yep"))"
```

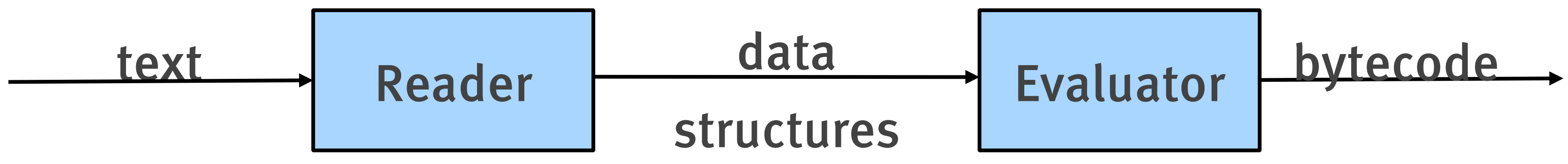
```
(cond
  (< 4 3) (pri...
  (> 4 3) (pri...
```

```
(if (< 4 3)
  (print "wrong")
  (if (> 4 3)
    (print "yep")
    nil))
```

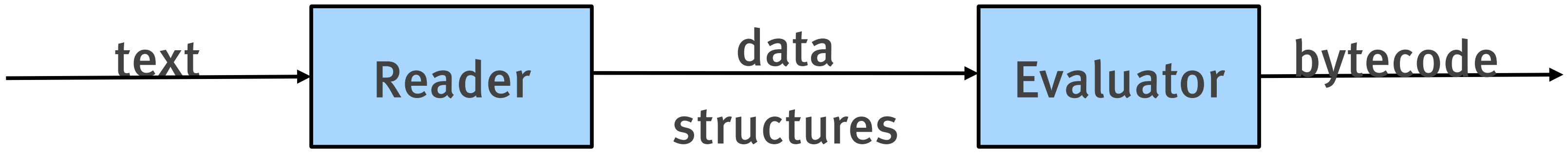
```
(defmacro my-cond [c1 e1 c2 e2]
  (list 'if c1
        e1
        (list 'if c2
              e2
              nil)))
```

```
(defmacro my-cond [c1 e1 c2 e2]
  (list 'if c1
        e1
        (list 'if c2
              e2
              nil)))


(my-cond
  false (println "won't see this")
  true (println "it works!"))
it works!
```

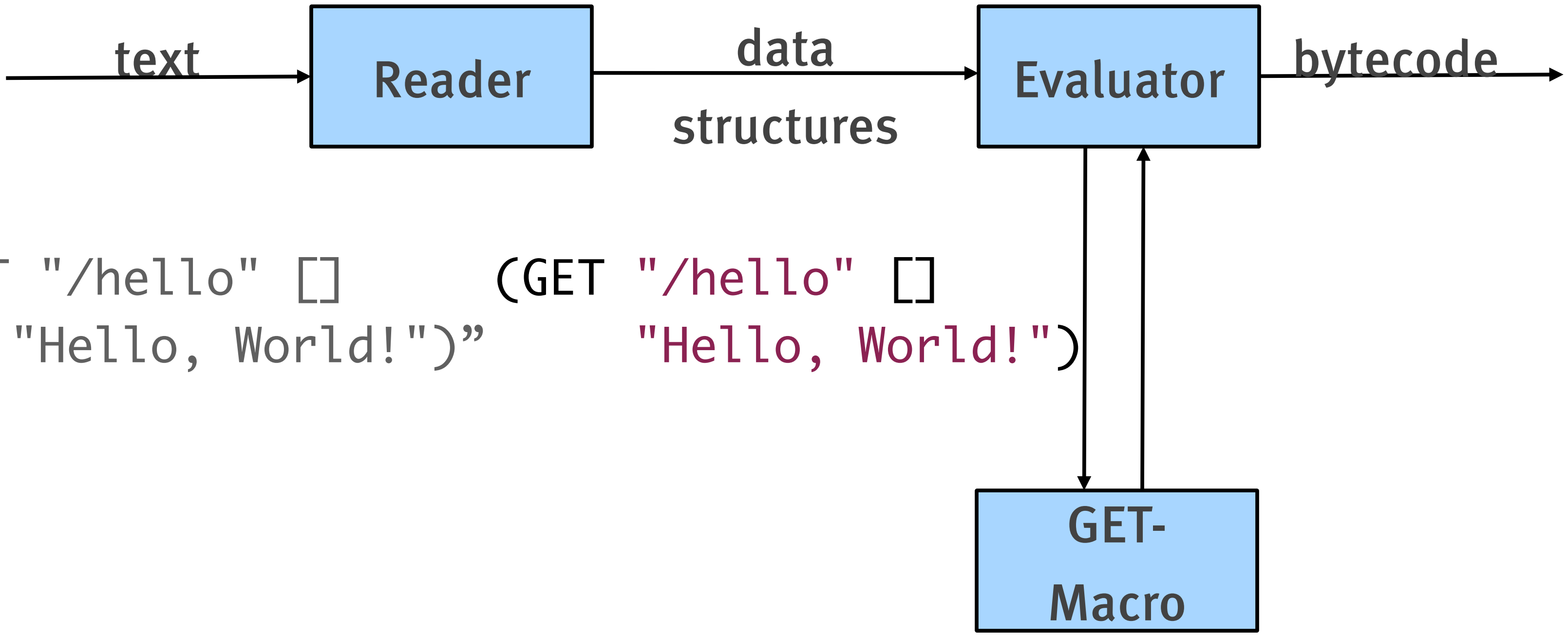
```
(defmacro my-cond [c1 e1 c2 e2]
  `(if ~c1
     ~e1
     (if ~c2
       ~e2
       nil)))


(my-cond
  false (println "won't see this")
  true (println "it works!"))
it works!
```



```
“(GET "/hello" []
     "Hello, World!")”
```



```
“(GET "/hello" []          (GET "/hello" []
     "Hello, World!")”          "Hello, World!")
```



“(GET "/hello" []
    "Hello, World!")”

```
(GET "/hello" []
     "Hello, World!")
```

```
(fn [req]
  (if (and (match (:uri req) "/hello")
           (= (:request-method req) :get))
   {:body "Hello, World!" …}
   nil))
```

innoQ