Common Lisp Native Coroutines

(sort of. . . ahem. . . actually, no)

Didier Verna

April 4 2017

Coroutines

- Very old idea
- *yield* values without losing its state
- *resume* its computation later (yielding more values)
- transfer control elsewhere
- ...

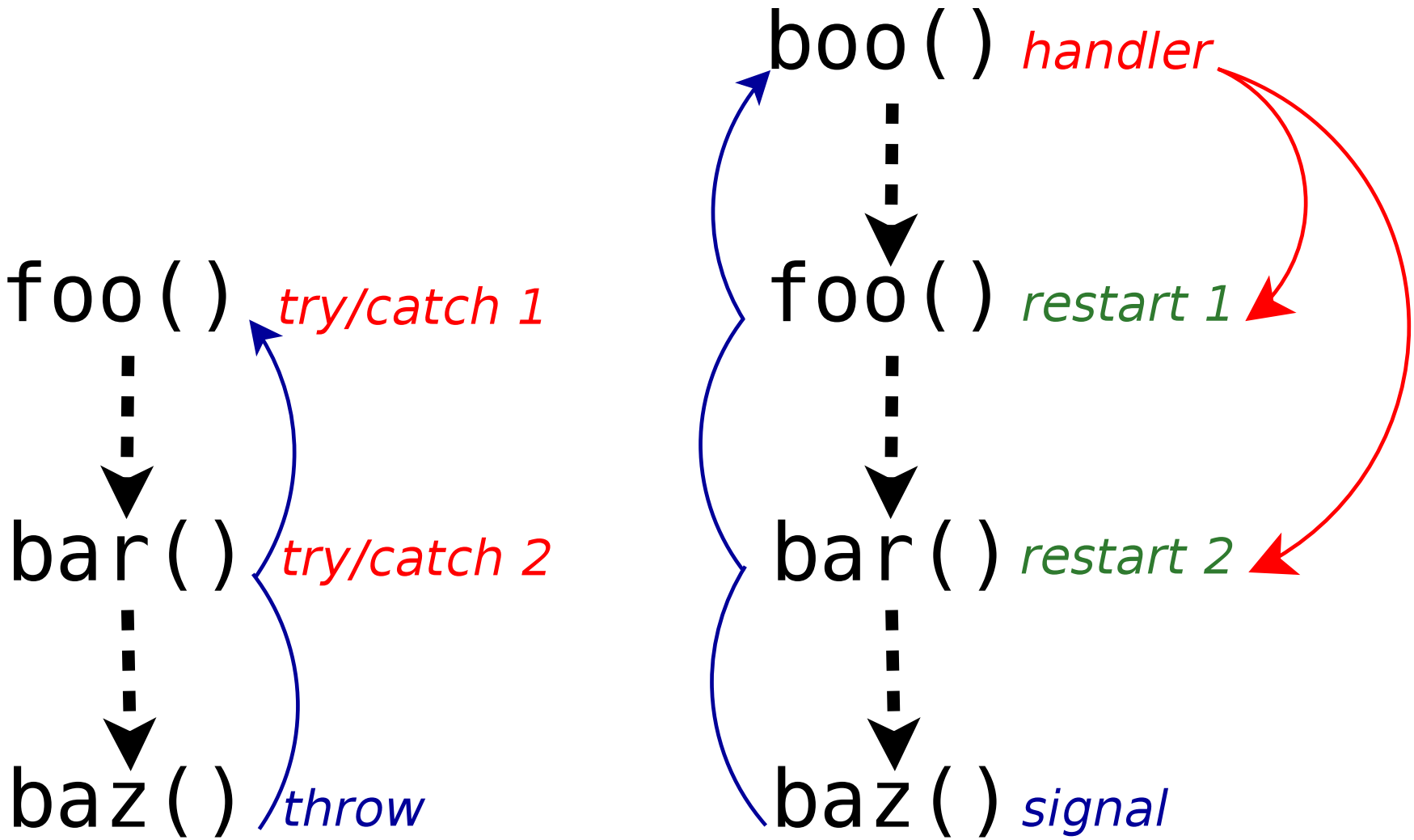
Examples

```
(defun squares ()
  (loop :for i :from 0
        :do (yield (* i i))))


(defun preorder (tree)
  (if (atom tree)
      (yield tree)
      (progn (preorder (car tree))
             (preorder (cdr tree)))))
```

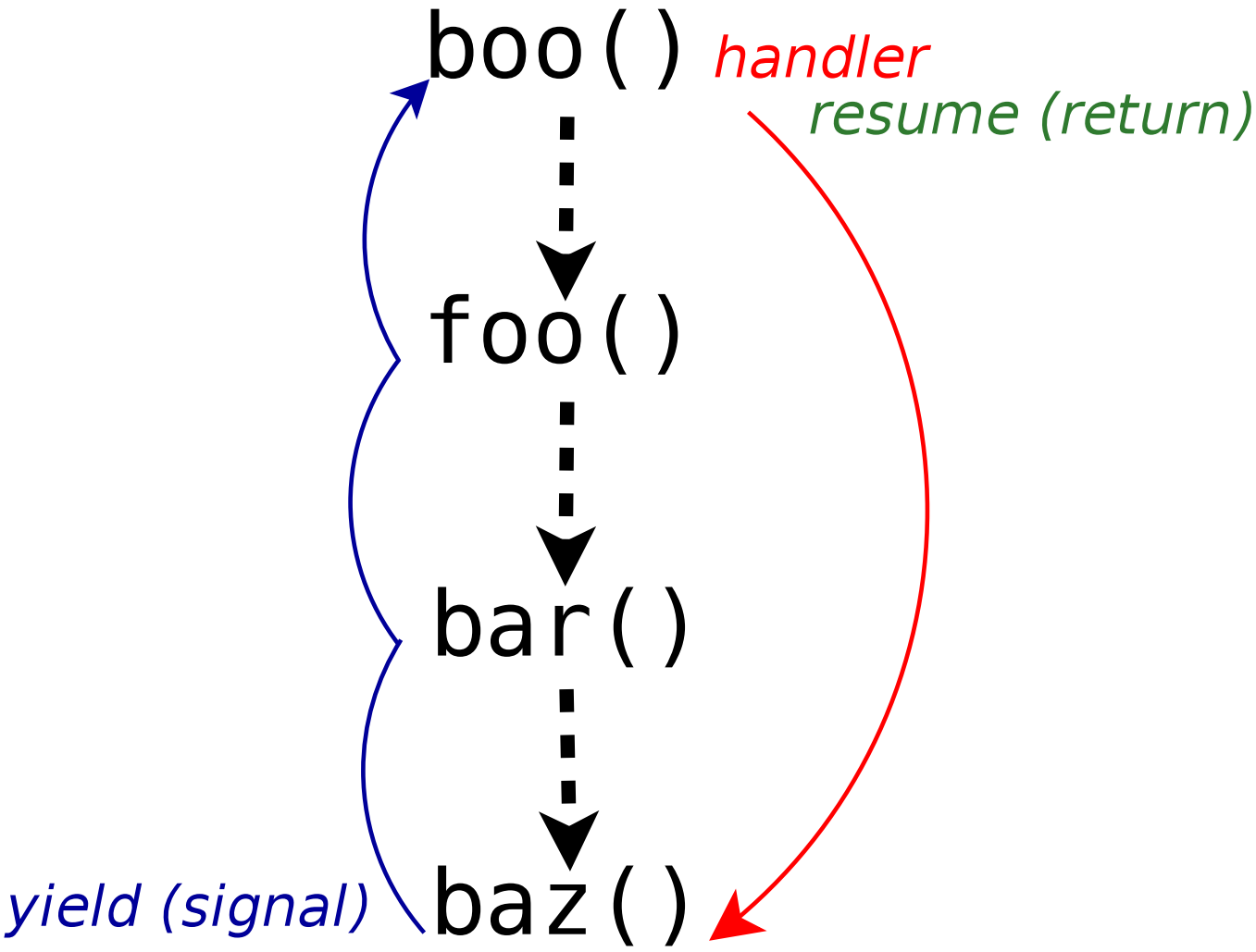
The Condition System

3D Separation of Concerns, no mandatory stack unwinding



Tricking the Condition System into Coroutin'ing

A handler *declining*, but still side-effecting!



Back to the Examples

```
(define-condition yield ()
  ((value :accessor value :initarg :value)))

(defun yield (value)
  (signal 'yield :value value))


(defun squares ()
  (loop :for i :from 0
        :do (yield (* i i))))

(defun preorder (tree)
  (if (atom tree)
      (yield tree)
      (progn (preorder (car tree))
             (preorder (cdr tree)))))
```

Handling yielded values

```
?
?
?
?
?
          ,coroutine)
?
```

Handling yielded values

```
?
?
        (handler-bind ((yield (lambda (condition)
                                (let ((,value (value condition)))
                                  ,@body))))
          ,coroutine)
?
```

Handling yielded values

```
?
  `(restart-case
       (handler-bind ((yield (lambda (condition)
                               (let ((,value (value condition)))
                                 ,@body))))
         ,coroutine)
     (abort ())))
```

Handling yielded values

```
(defmacro with-coroutine (coroutine value &body body)
  `(restart-case
       (handler-bind ((yield (lambda (condition)
                               (let ((,value (value condition)))
                                 ,@body))))
         ,coroutine)
     (abort ())))
```

Handling yielded values

```
(defmacro with-coroutine (coroutine value &body body)
  `(restart-case
       (handler-bind ((yield (lambda (condition)
                               (let ((,value (value condition)))
                                 ,@body))))
         ,coroutine)
     (abort ())))
```

```
(defun ssq (n)
  (let ((step 0)
        (sum 0))
    (with-coroutine (squares) sq
      (incf sum sq)
      (incf step)
      (when (> step n)
        (abort)))
    sum))
```

```
(defun leaves (tree)
  (let (leaves)
    (with-coroutine (preorder tree) leaf
      (push leaf leaves))
    (nreverse leaves)))
```