Groovy

Using closures

Closures

- Short anonymous methods that remove the verbosity of Java's anonymous inner classes
- Derived from the lambda expressions from functional programming

Motivating example

Example 00

Closure example

```
def pickEven(n, block) {
  for(int i = 2; i <= n; i += 2) {
    block(i)
  }
}

pickEven(10, { println it } )
```

- The *pickEven()* method is a higher-order function - a function that takes functions as arguments or returns a function as a result.
- The variable *block* holds a reference to a closure.

Closure as the last argument

- In Groovy, we can pass as many closures as we want.
- If a closure is the last argument, there is an elegant syntax:

```
def pickEven(n, block) {
  for(int i = 2; i <= n; i += 2) {
    block(i)
  }
}

pickEven(10) { println it }
```

Naming closure parameters

- We can give an alternate name to the closure argument, if we like:

```
def pickEven(n, block) {
  for(int i = 2; i <= n; i += 2) {
    block(i)
  }
}

pickEven(10) { evenNumber -> println evenNumber }
```

Binding variables

- A closure is a function with variables bound to a context or environment in which it executes.

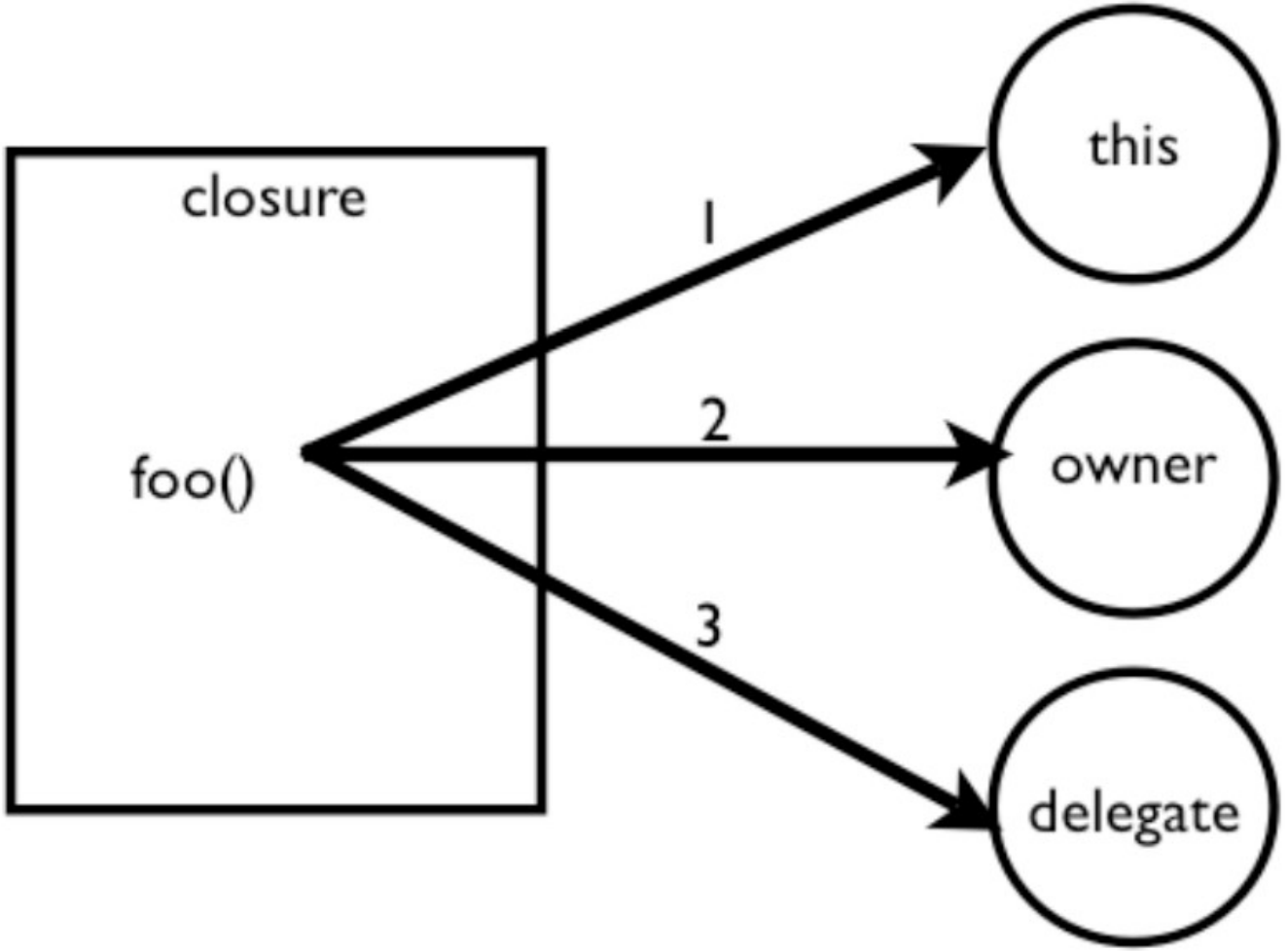
```
def pickEven(n, block) {
  for(int i = 2; i <= n; i += 2) {
    block(i)
  }
}

total = 0
pickEven(10) { total += it }
println "Sum of even numbers from 1 to 10 is ${total}"
```

Curried closures

- From the name Haskell B. Curry, famed mathematician who contributed to lambda calculus
- *curry()* - curries first parameter
- *rcurry()* - curries last parameter
- *ncurry()* - curries n-th parameter

Closures and delegation



Tail recursion and trampolines

- *trampoline()* builds a trampolined variant of the current closure.
- Under trampoline, the function is supposed to perform one step of the calculation and, instead of a recursive call to itself or another function, it return back a new closure, which will be executed by the trampoline as the next step.
- Once a non-closure value is returned, the trampoline stops and returns the value as the final result.

Memoization

- Implementation of dynamic programming built into Groovy
- Trades space for speed
- Results are cached
- *memoize()* has unlimited cache
- Using *memoizeAtMost()* we can limit the cache size