因循

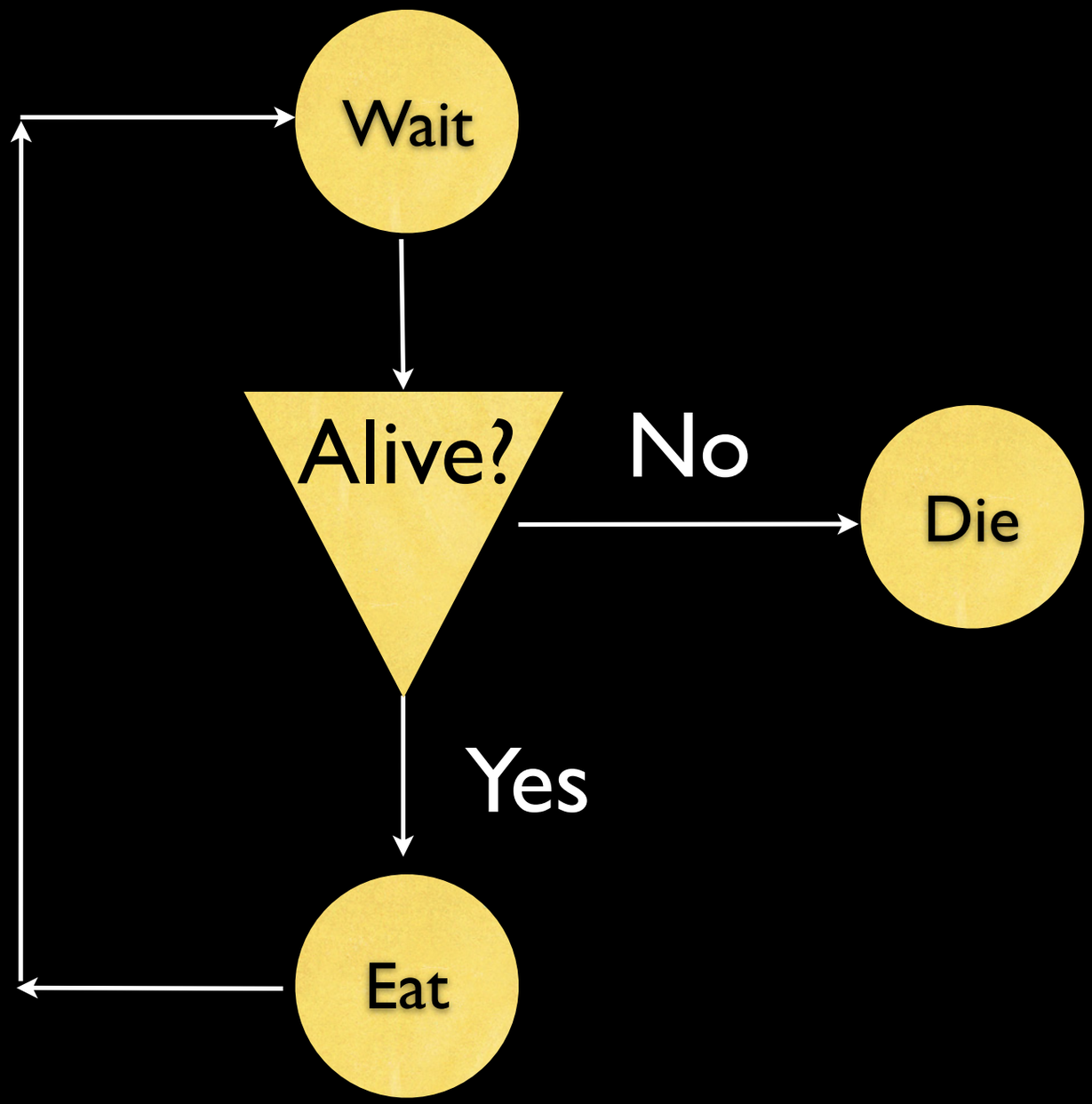
Agent Based Modeling with Coroutines

因循 *[yīn xún]*

- to continue the same old routine
- to carry on just as before
- to procrastinate

What do they allow you to do?

- Interrupt a task, and have it pick up right back where you left it
- Cooperatively swap between concurrent tasks


Wait
Alive?
No
Die
Yes
Eat

```
coroutine guard_1 apply {who {
   while {[is_alive $who]} {
      yield eating
   }
   return die
}} Melvin
```

```
coroutine guard_2 apply {who {
while 1 {
  set o [nearby $who]
  if { $o ne {} } {
    yield “follow $o”
  } else {
    yield [look_dopey]
  }
}
}} Harry
```

```
coroutine lancelot {who {
 while 1 {
   set o [nearby $who]
   if {[alive $o]} {
     yield “stabstabstab $who $o”
   } else {
     yield run
   }
}
}} Lancelot
```

```
proc scene_121 {} {
 set actors {guard_1 guard_2 lancelot}
 for
  fo
   s
   d
   i
     }
   }
  }
 }
}
```

```
while [is_alive $who] {
 yield eating
}
return die
```

```
proc scene_121 {} {
 set actors {guard_1 guard_2 lancelot}
 for
  fo
   s
   d
   i
    }
   }
  }
 }
}
```

```
while 1 {
    set o [nearby $who]
    if { $o ne {} } {
      yield “follow $o”
    } else {
      yield look_dopey
    }
  }
```

```
proc scene_121 {} {
 set actors {guard 1 guard 2 lancelot}
 for while 1 {
  fo   set o [nearby $who]
    s  if {[alive $o]} {
    d    yield “stabstabstab $who $o”
    i  } else {
      }    yield run
    }   }
  }  }
 }
}
```

```
proc scene_121 {} {
 set actors {guard_1 guard_2 lancelot}
 for
  fo
   s
   d
   i
    }
   }
  }
 }
}
```

```
while [is_alive $who] {
 yield eating
}
return die
```

```
proc scene_121 {} {
  set actors {guard 1 guard 2 lancelot}
  for while 1 {
    fo   set o [nearby $who]
      s  if {[alive $o]} {
      d    yield “stabstabstab $who $o”
      i  } else {
        }    yield run
      }   }
    }   }
  }
}
```

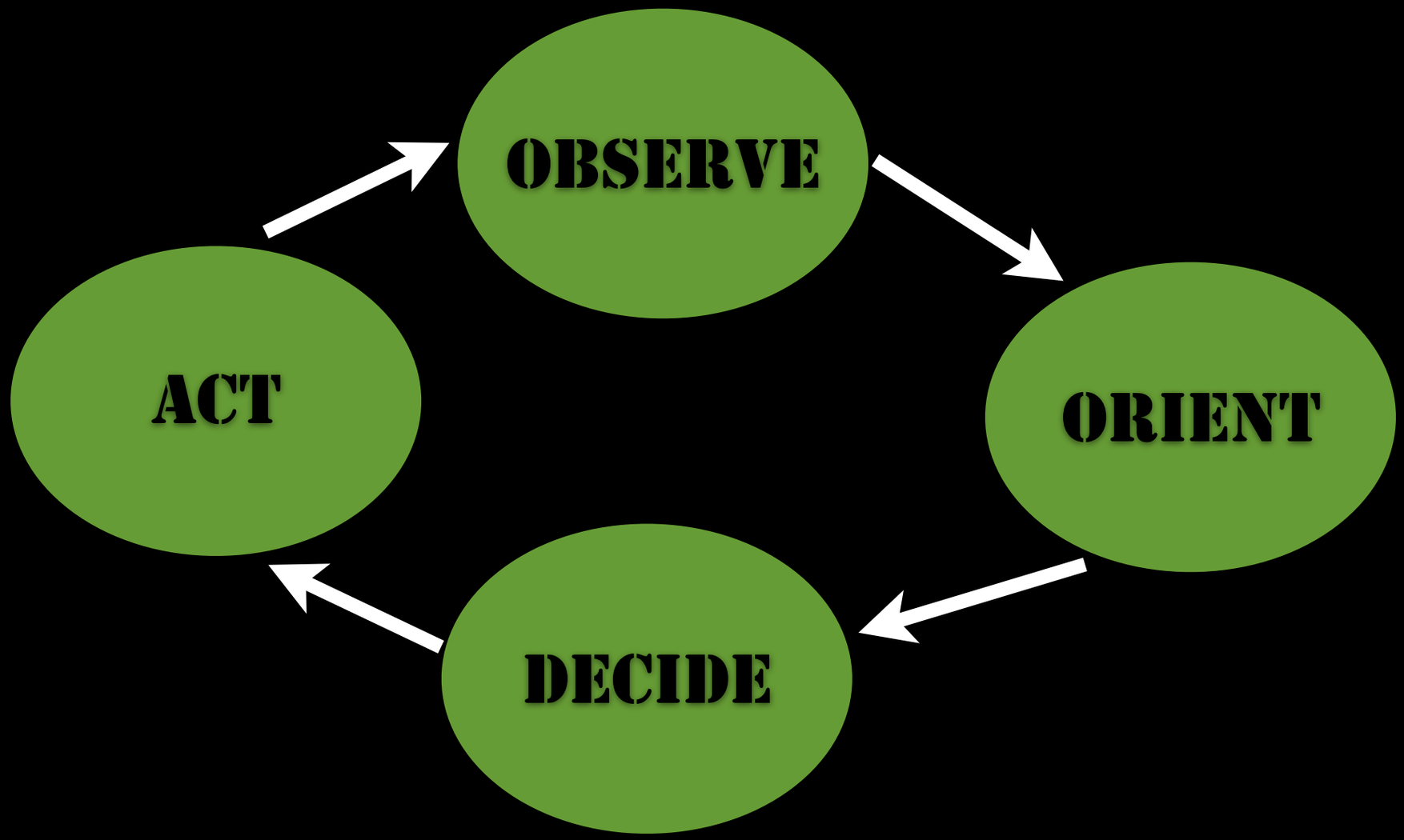
```
proc scene_121 {} {
  set actors {guard_1 guard_2 lancelot}
  for
    fo
      s
      d
      i
        }
      }
    }
  }
}
```

```
while 1 {
    set o [nearby $who]
    if { $o ne {} } {
      yield “follow $o”
    } else {
      yield look_dopey
    }
  }
```

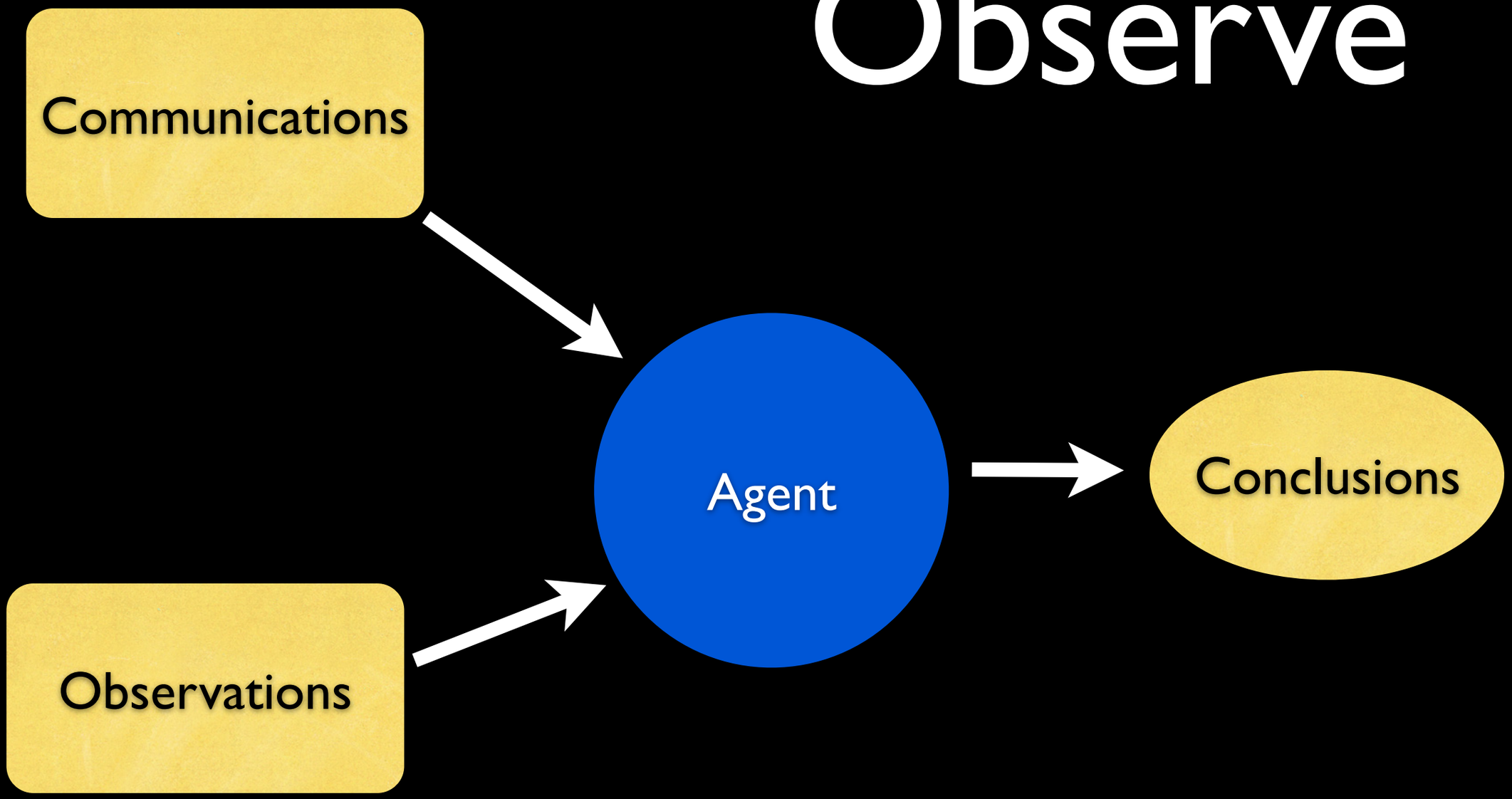
Effective Coroutines...

- Spend most of their existence waiting
- React to the state of the world
- Send signals rather than perform a direct action

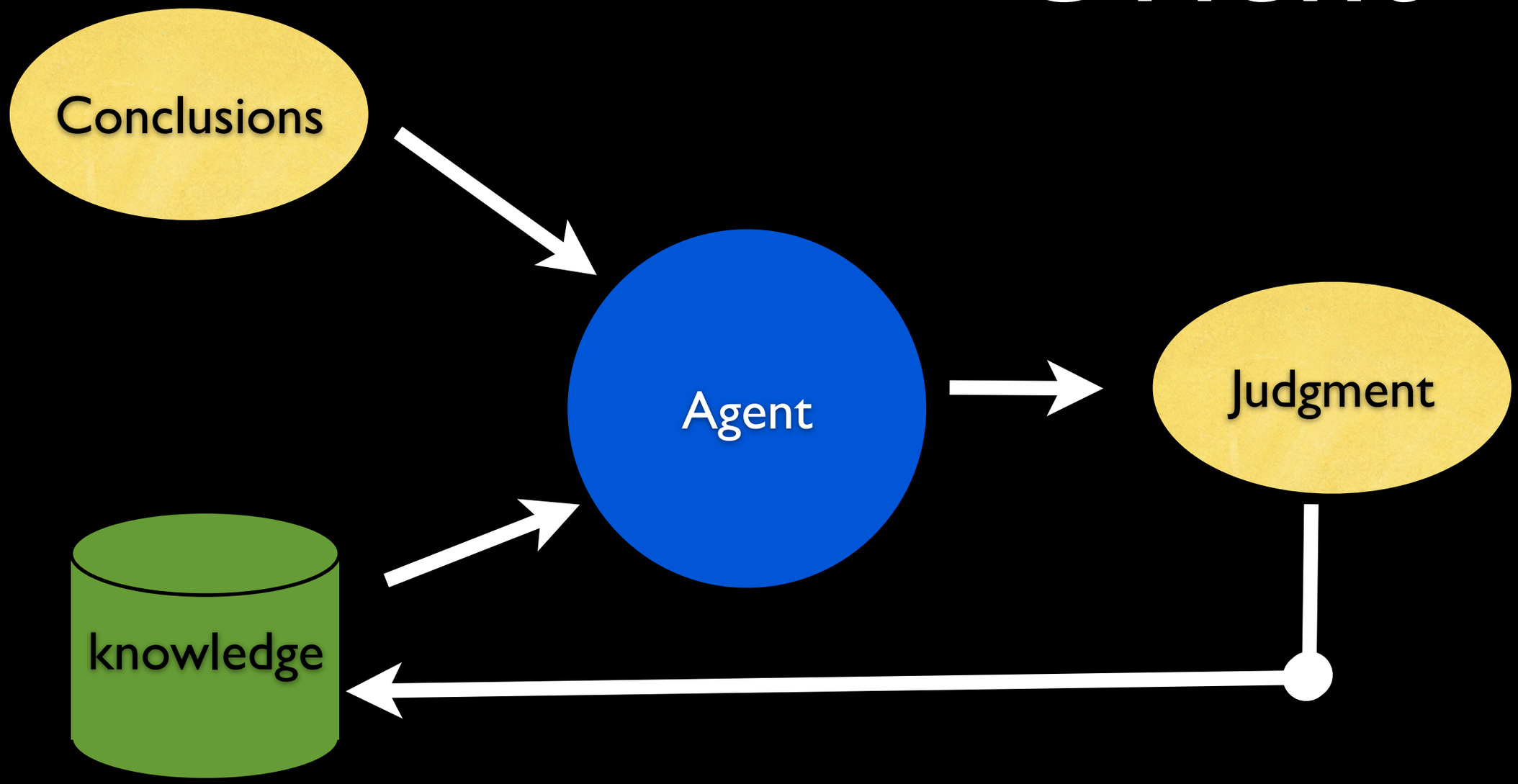
OODA Loop



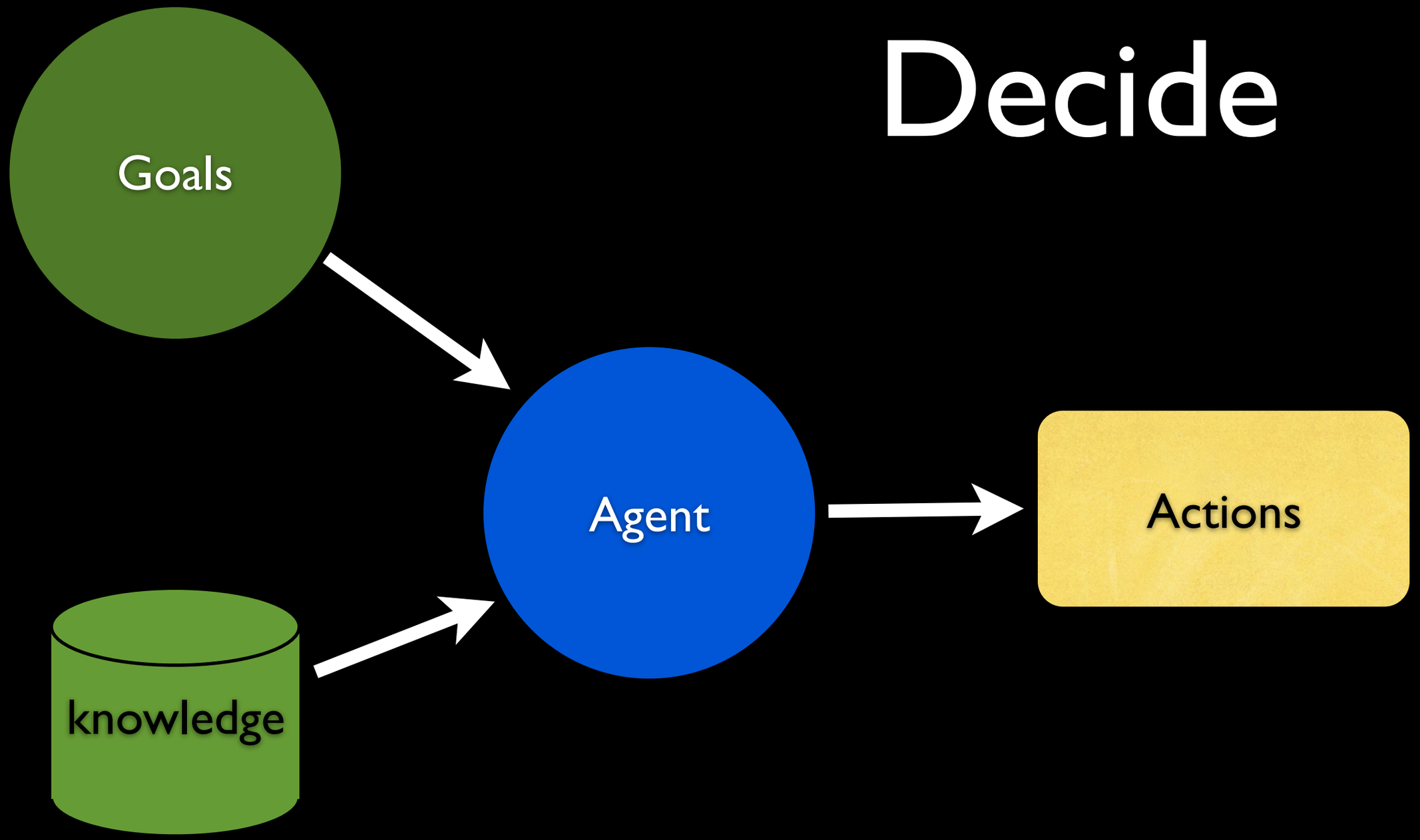
Observe



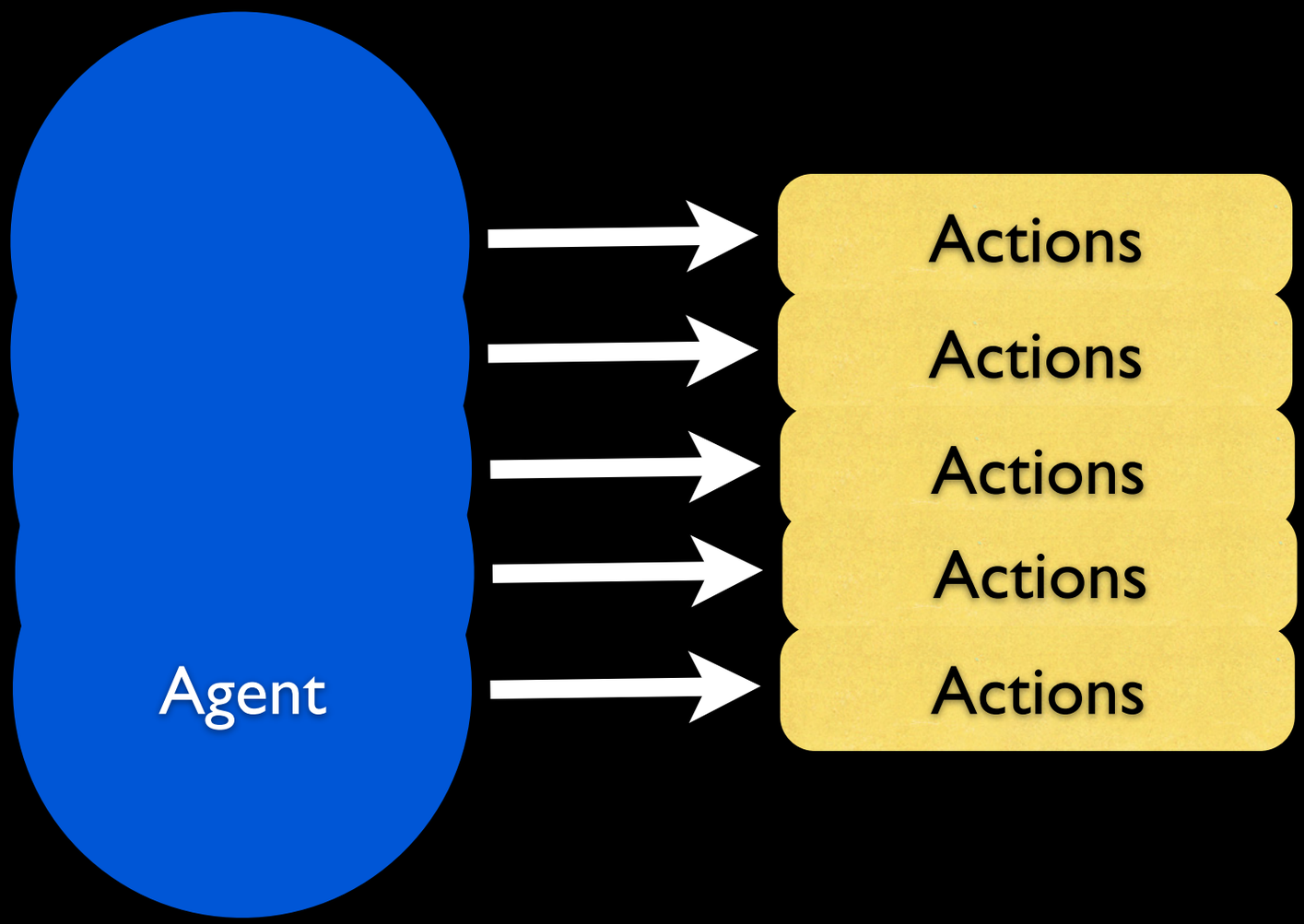
Orient



Decide



Act



Why Coroutines

- Observe? Batch Process
- Orient? Functional
- Act? Procedural
- > Decide <

Decision

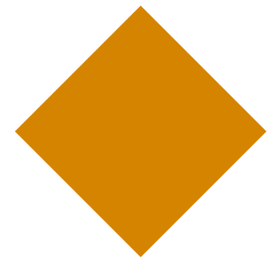
- Many actions span multiple timesteps
- Implementing them as state machines led to ... curious... results

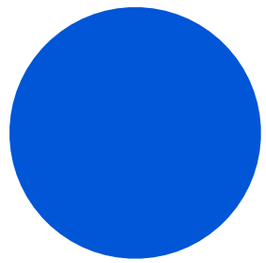
Attend Objective

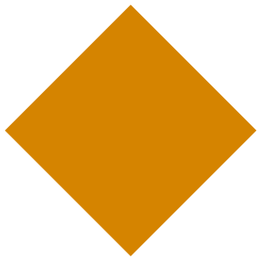
```
if { $objective eq {} } {
  set objective [my select_task]
}
if {$objective == {} } {
 return 0
}
if {[objective_done $objective]} {
 set objective {}
 return 0
}
if {[my isnear $objective]} {
 return 1
}
if {[my is_moving]} {
 return 1
}
if {[catch {my route_to $objective}]} {
  return -1
}
return 0
```

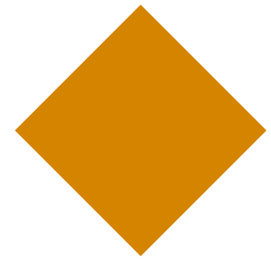
Go Home

```
if {$objective != {} } {
 return 0
}
if {[my isnear $home]} {
 return 1
}
if {[my is_moving]} {
  return 1
}
if {[catch {my route_to $home}]} {
  return -1
}
return 0
```









Attend Objective

```
if { $objective eq {} } {
  set objective [my select_task]
}
if {$objective == {} } {
 return 0
}
if {[objective_done $objective]} {
 set objective {}
 return 0
}
if {[my isnear $objective]} {
 return 1
}
if {[my is_moving]} {
 return 1
}
if {[catch {my route_to $objective}]} {
  return -1
}
return 0
```

Go Home

```
if {$objective != {} } {
 return 0
}
if {[my isnear $home]} {
 return 1
}
if {[my is_moving]} {
  return 1
}
if {[catch {my route_to $home}]} {
  return -1
}
return 0
```

Attend Objective

```
set objective [my select_task]
if {$objective == {} } {
 return 0
}
if {[catch {my route_to $objective}]} {
  return -1
}
while {[my is_moving]} {
  yield 1
}
while {![objective_done $objective]} {
 yield 1
}
set objective [my select_task]
if {$objective == {}} {
 return 0
}
return 1
```

Go Home

```
if {$objective != {} } {
 return 0
}
if {[my isnear $home]} {
 return 0
}
if {[catch {my route_to $home}]} {
  return -1
}
while {[my is_moving]} {
  yield 1
}
return 0
```

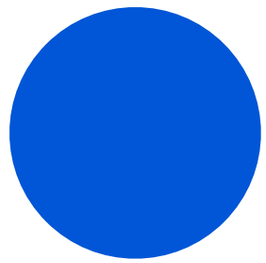
```
method move_to dest {
 if {[my isnear $dest]} {
  return 0
 }
 if {[catch {my route_to $dest}]} {
   return -1
 }
 while {[my is_moving]} {
   yield 1
 }
 return 0
}
```

Attend Objective

```
set objective [my select_task]
if {$objective == {} } {
 return 0
}
if {[my move_to $objective] < 0} {
  set objective {}
  return -1
}
while {![objective_done $objective]} {
 yield 1
}
set objective [my select_task]
if {$objective == {}} {
 return 0
}
return 1
```

Go Home

```
if {$objective != {} } {
 return 0
}
if {[my isnear $home]} {
 return 0
}
my move_to $home
return 0
```



```
method move_to dest {
 if {[my isnear $dest]} {
  return 0
 }
 set start [my location]
 if {[catch {my route_to $dest}]} {
   return -1
 }
 while {[my is_moving]} {
   if {![my comptsafe $compartment]} {
    my retreat $start
    set objective {}
    return 1
   }
   yield 1
 }
 return 0
}
```

```
method retreat start {
 if {[catch {my route_to $start}]} {
   return -1
 }
 while {[my is_moving]} {
   if {[my comptsafe $compartment]} {
    my route_to {}
    return 1
   }
   yield 1
 }
 return 0
}
```


I SEE WHAT YOU DID THERE

Coroutines work:

- In subroutines
- In TclOO Methods
- In Apply scripts

Questions?