Concurrent Programming in AmbientTalk

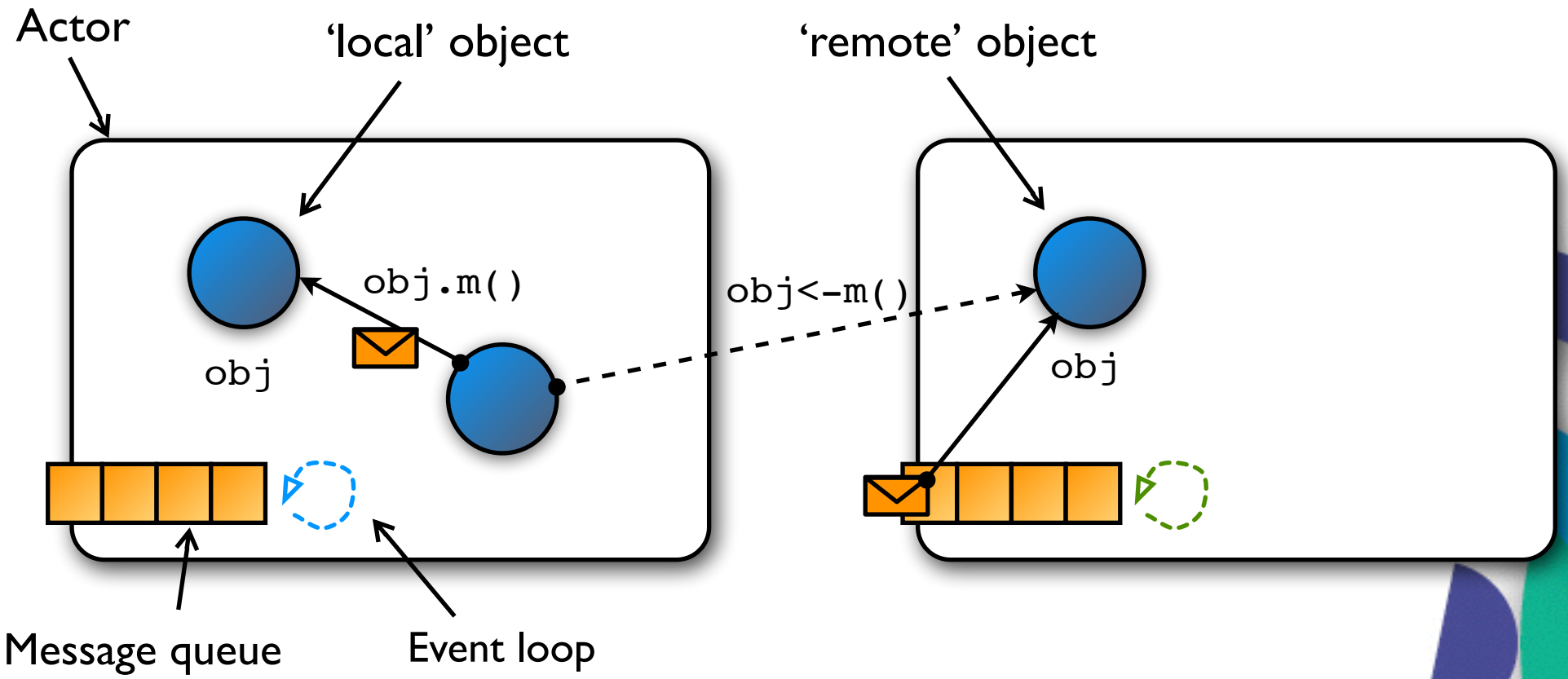
Guards and Unit Testing

Elisa Gonzalez Boix
egonzale@vub.ac.be

Vrije Universiteit Brussel

Actors Recap

Based on E programming language [Miller05]

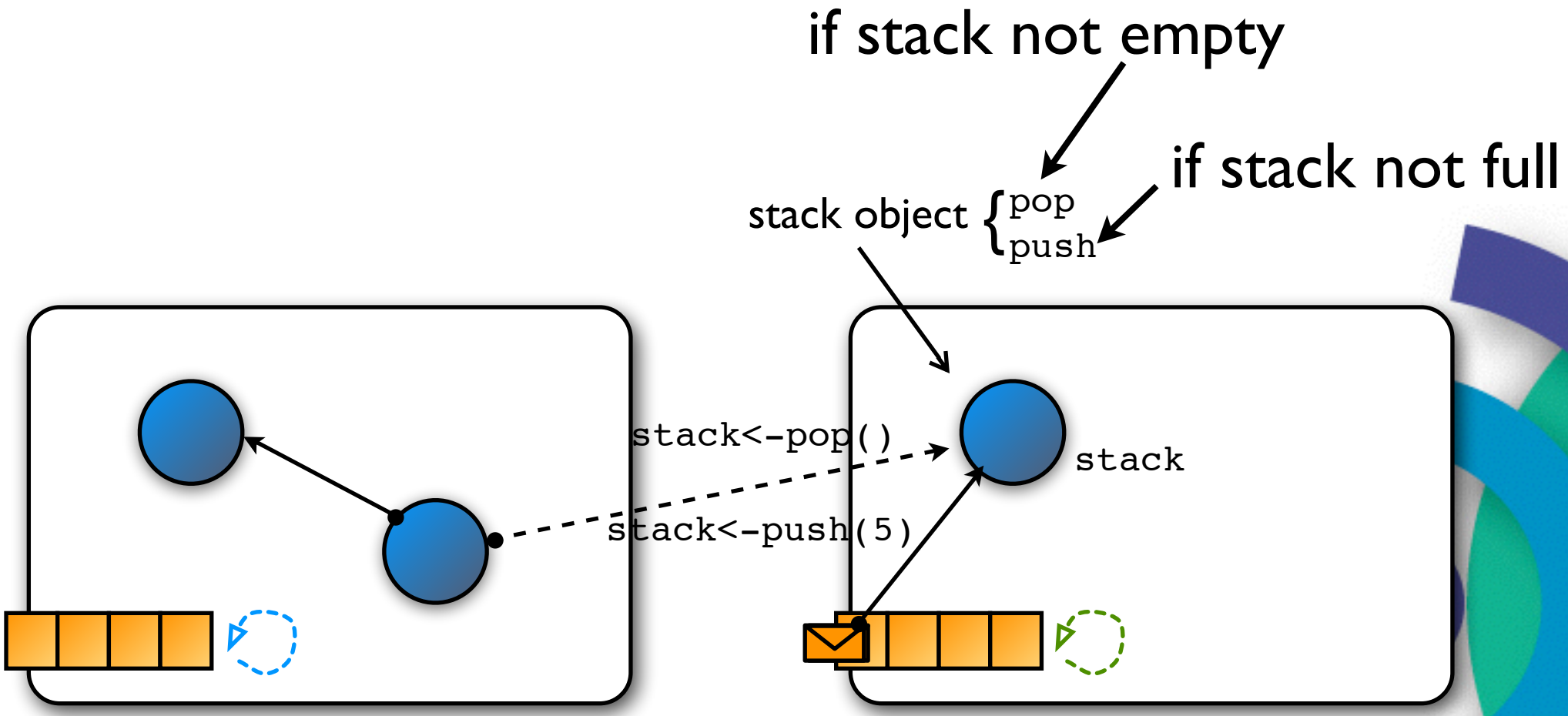


Guards

Guards

[Dijkstra, 1975]

- Conditional synchronization mechanism



Guards in AT

language extension

guards per object basis

access to lexical scope!

```
import /.at.lang.guards;
def stack := guardedObject: {
   def vector := Vector.new();
   def pop()@Guard: {!vector.isEmpty} {
           vector.pop();
   };
   def push(element)@Guard: {!((vector.length >= MAX).or:
{element % 2 == 0} )} {
      vector.add(element);
   };
};
```

boolean conditions needs to evaluate to true to execute the method

Guards in AT

Guard predicate applied only to asynchronous message passing!

```
stack<-push(1);
...

when: stack<-pop()@FutureMessage becomes: { |num|
        self.assertEquals("1", print: num);
};

stack<-pop();
```

It won't be executed until somebody does a push!

Unit Testing

Unitest Framework

- ~ JUnit, SUnit

```
def UnitTest := /.at.unit.test;
```

```
def myUnitTest := extend: UnitTest.new("my UnitTest") with: {
   def testSomething() {
      self.assertEquals(3,1+2);
   }
}
```

Method defined in UnitTest object

```
myUnitTest.runTest()
```

Asynchronous UnitTest

by default, parallel!

```
.new("my UnitTest", true)
```

```
def myUnitTest := extend: UnitTest.new("my UnitTest") with: {
  def testAsyncOneCustomer(){
    def f := when: customer<-haveComputer() becomes:{ | id |
      self.assertEquals(id, 1);
    };
    f;
  };
};
```

Return value of an asynchronous unit test should be a future!

```
myUnitTest.runTest()
```

Unitest Framework

```
assertTrue(boolean)
assertFalse(boolean)
assertEquals(o1,o2)
assertNotEquals(o1,o2)
assertLessThan(o1,o2)
assertGreaterThan(o1,o2)
assertLessThanOrEquals(o1,o2)
assertGreaterThanOrEquals(o1,o2)
assertMatches(str, pattern)
assertNotNil(val)
```

```
assert: exceptionType raisedIn: closure
fail(reason)
```

http://soft.vub.ac.be/amop