Linguistic Symbiosis between Actors and Threads

Tom Van Cutsem   Stijn Mostinckx   Wolfgang De Meuter

Programming Technology Lab
Vrije Universiteit Brussel
Brussels, Belgium

International Conference on Dynamic Languages, August 27th 2007, Lugano

Overview

- AmbientTalk: OO DSL for mobile ad hoc networks
- Pure event-driven concurrency model (actors [Agha86])
- How to do a *safe* linguistic symbiosis between actors and threads?

AMBIENTTALK

Actors vs. Threads

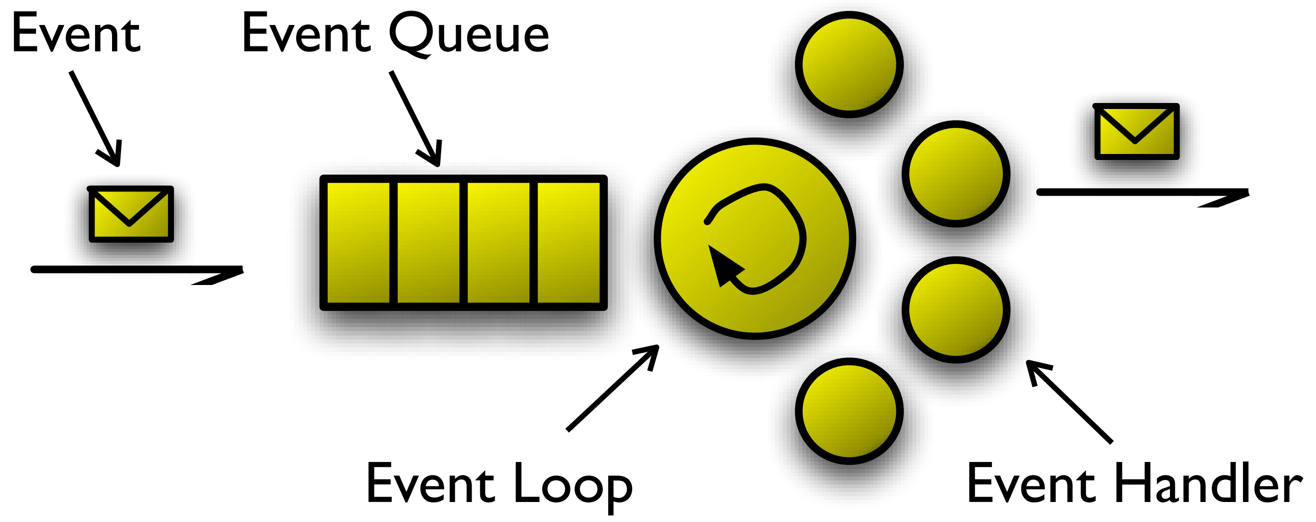
```
actor: {
  def obj := object: {
    def m() { ... }
  }

  def button := Button.new(“Click Me”);
  button.addActionListener(object: {
    def actionPerformed(actionEvent) {
      obj.m();
    }
  })

  obj.m();
}
```

Actors vs. Threads

```
actor: {
  def obj := object: {
    def m() { ... }
  }

  def button := Button.new("Click Me");
  button.addActionListener(object: {
    def actionPerformed(actionEvent) {
      obj.m();
    }
  })

  obj.m();
}
```

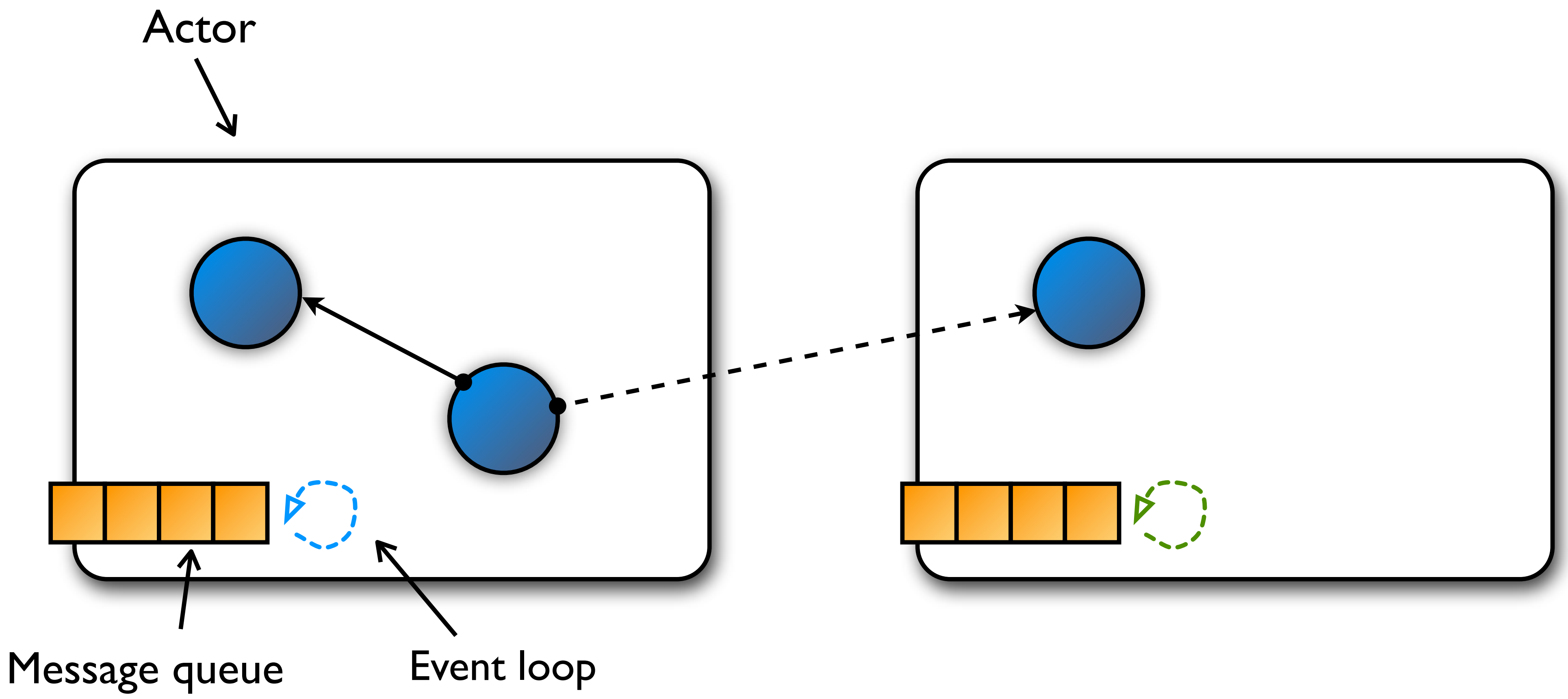
Event Loop Concurrency

- Events are executed serially
- Event notification is strictly asynchronous
- Event loops should have no shared state



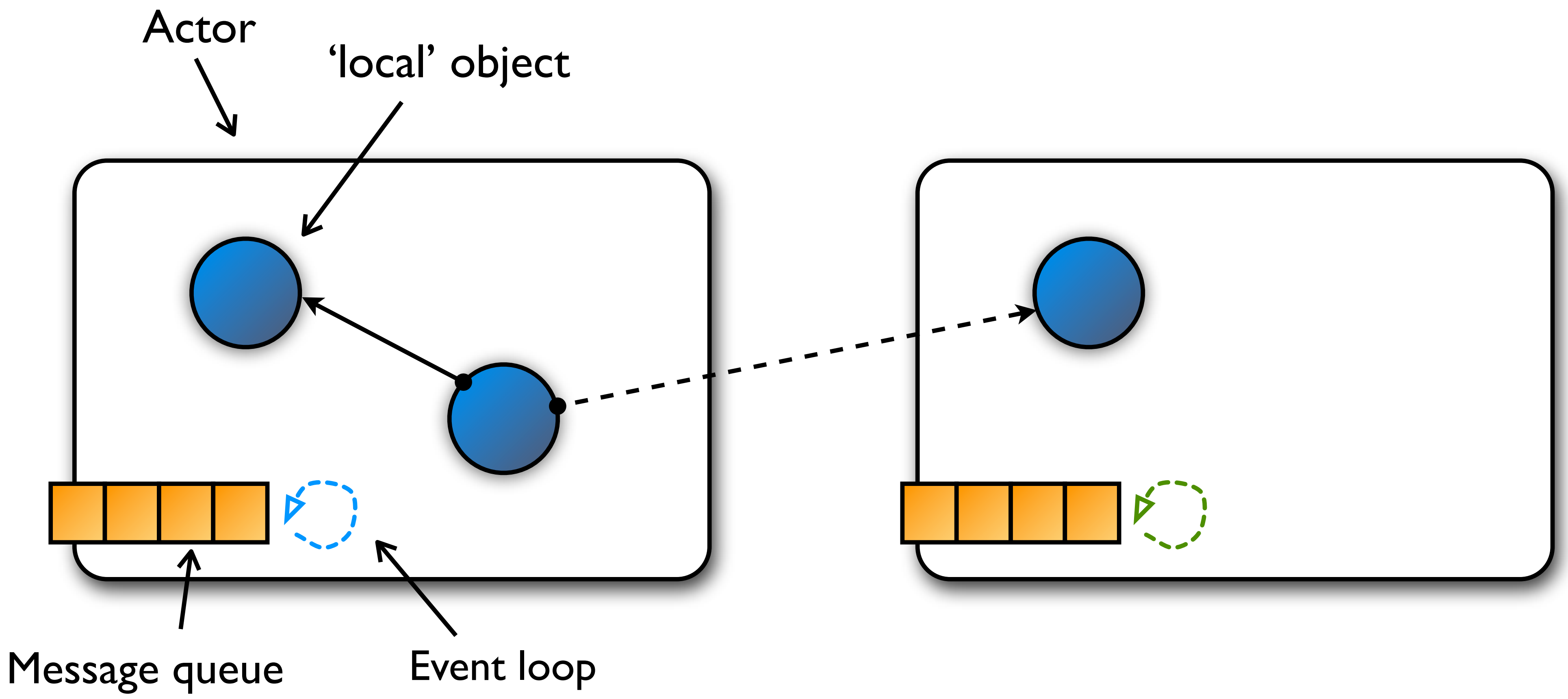
Event loop concurrency

Based on E programming language [Miller05]



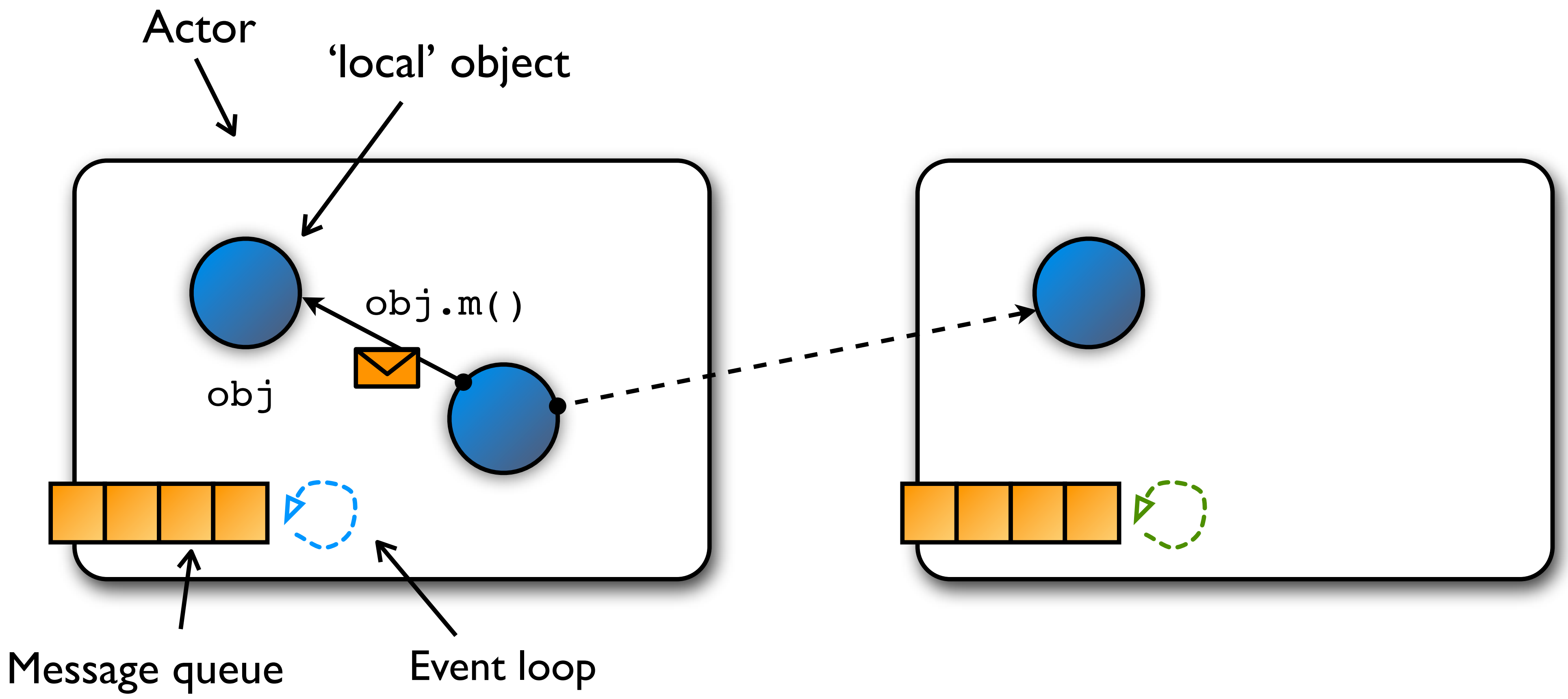
Event loop concurrency

Based on E programming language [Miller05]



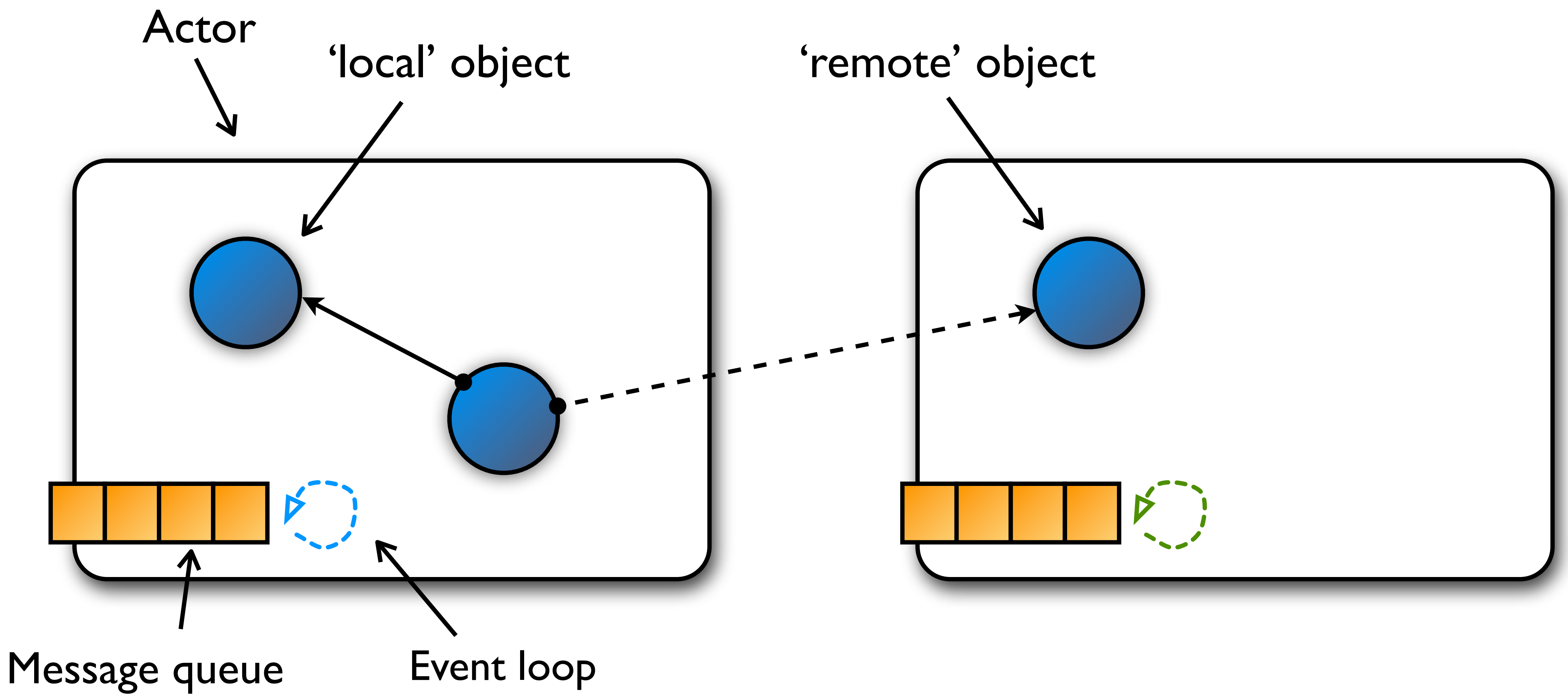
Event loop concurrency

Based on E programming language [Miller05]



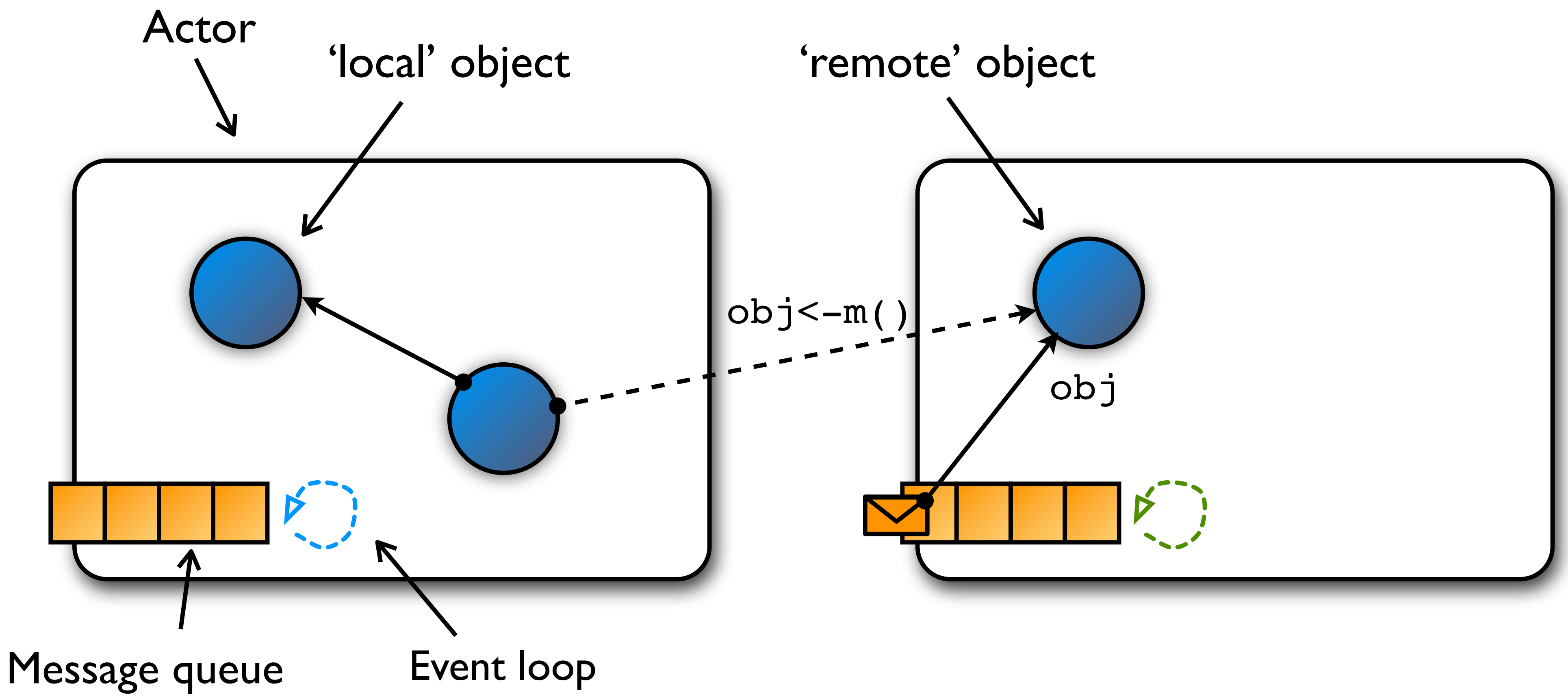
Event loop concurrency

Based on E programming language [Miller05]



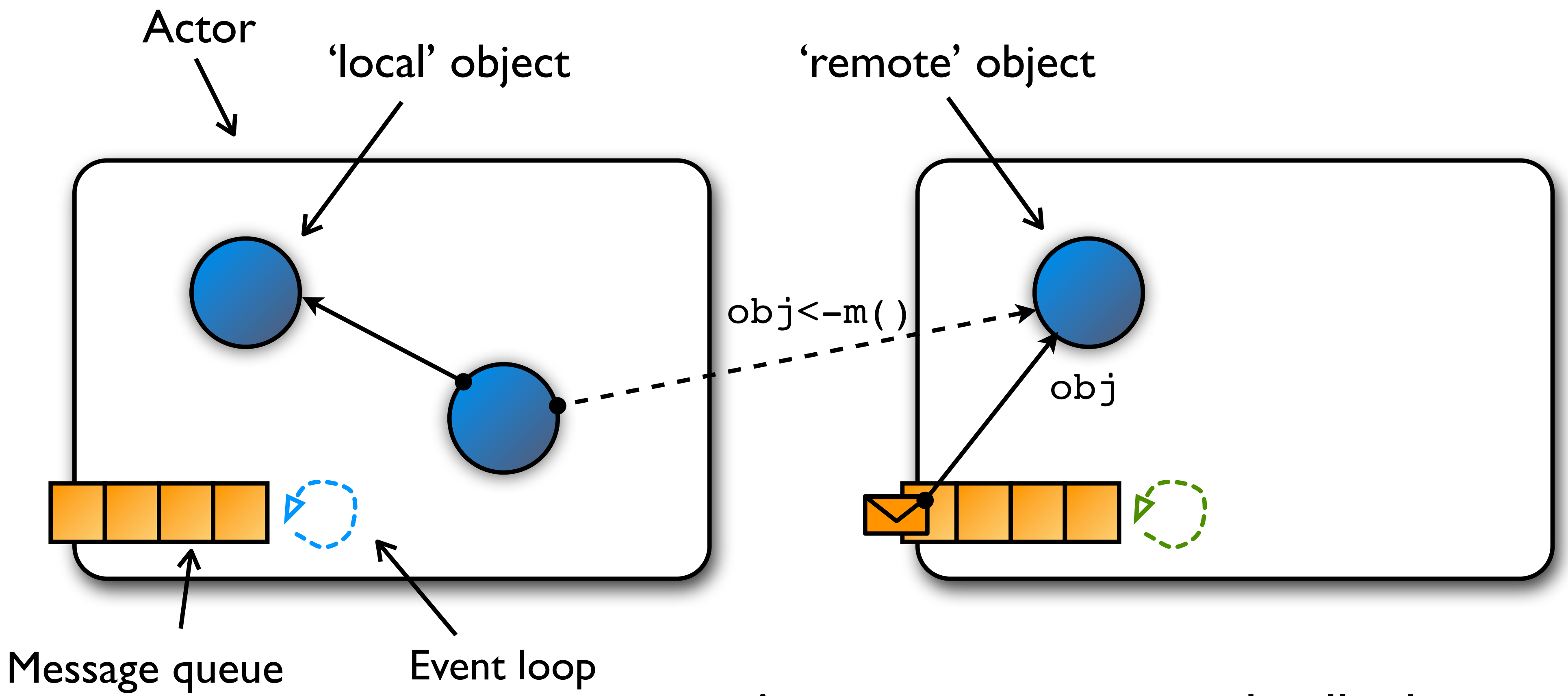
Event loop concurrency

Based on E programming language [Miller05]



Event loop concurrency

Based on E programming language [Miller05]



Actors cannot cause deadlock

No race conditions on objects

AmbientTalk/Java

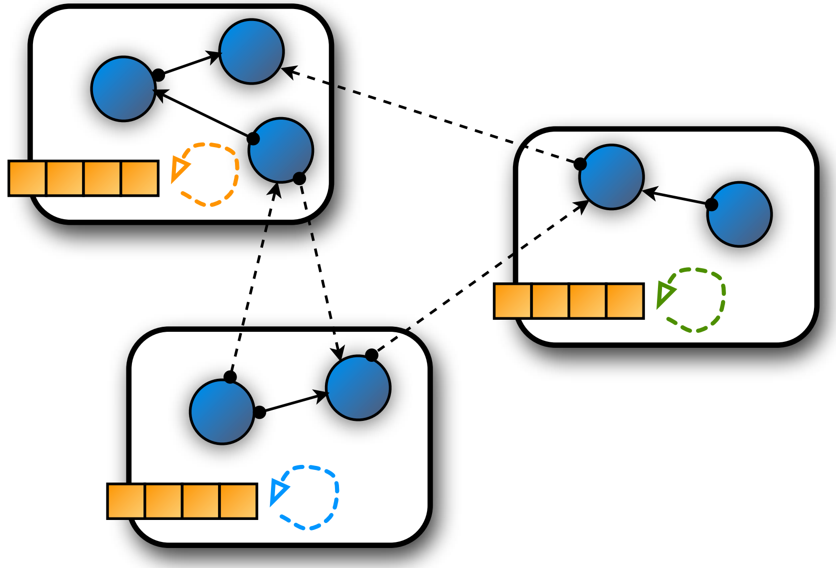
Based on Inter-language Reflection [Gybels et al 05]

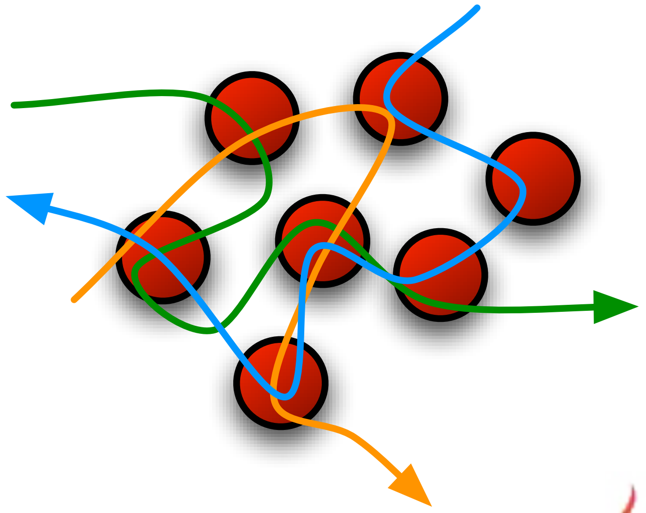
- AmbientTalk is implemented in Java
- Data mapping: cfr. JRuby, Jython, JScheme, LuaJava, JPiccola, ...
- Tight integration at the syntactic level

```
def Button := jlobby.java.awt.Button;
def button := Button.new("Click Me");
button.addActionListener(object: {
  def actionPerformed(actionEvent) { ... }
});
button.setVisible(true);
```

Actor/Thread Mapping

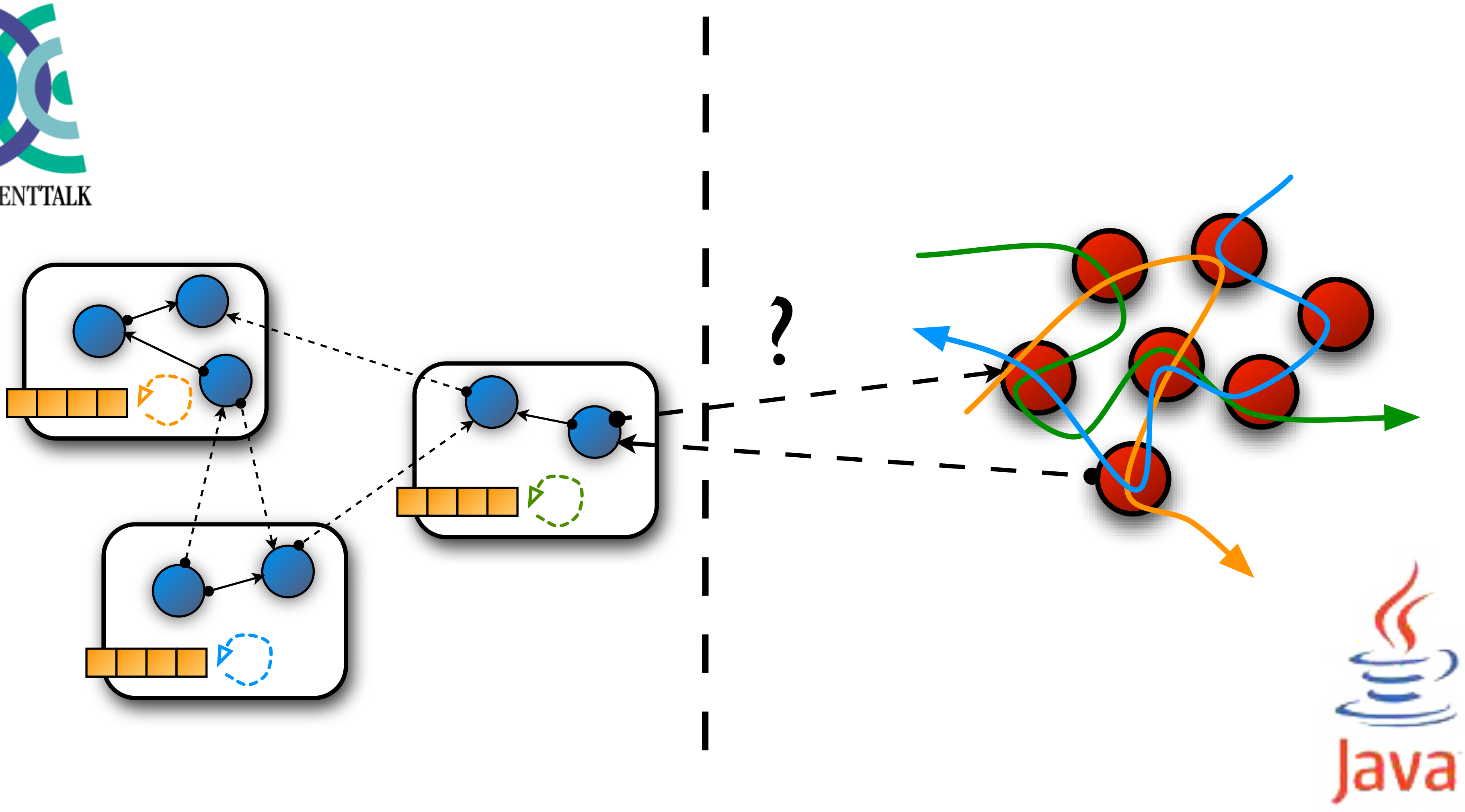
AMBIENTTALK





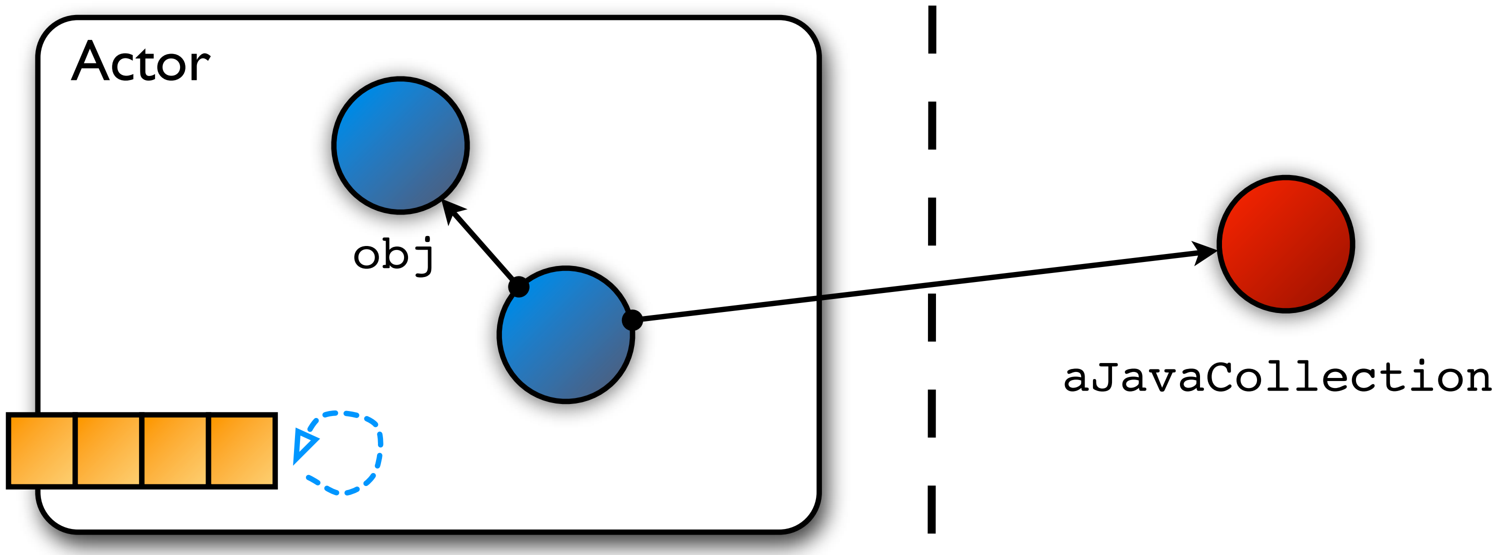
Java

Actor/Thread Mapping



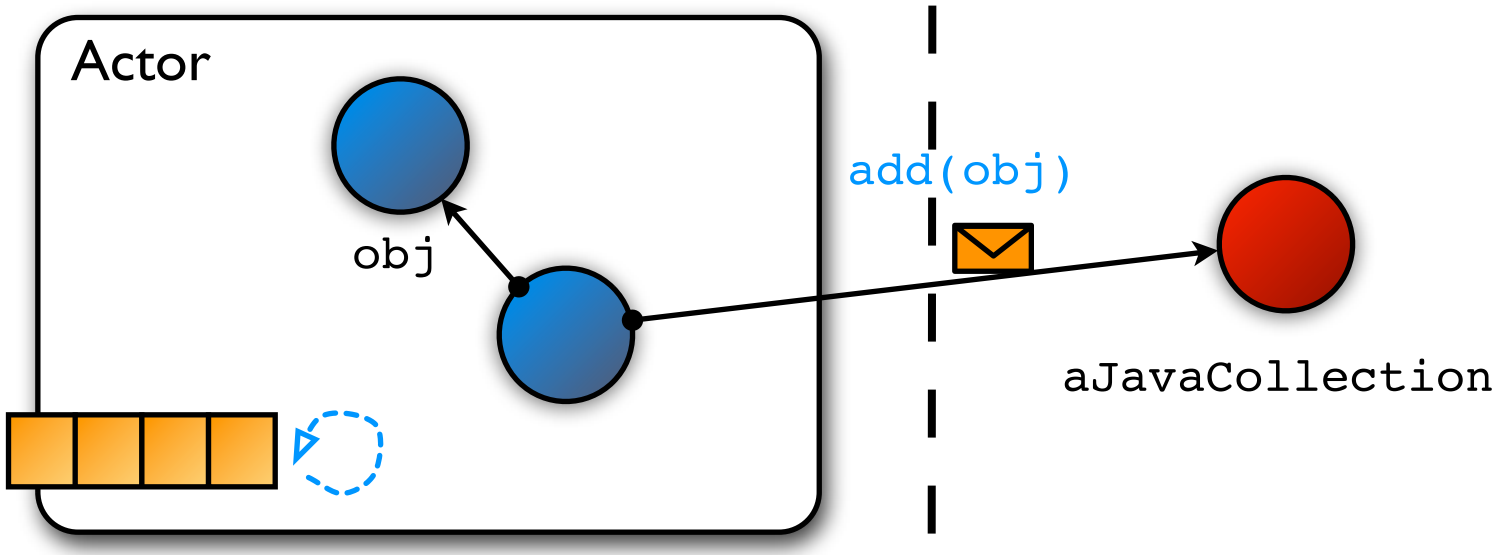
Actors as Threads

```
def obj := object: { ... };
aJavaCollection.add(obj);
```



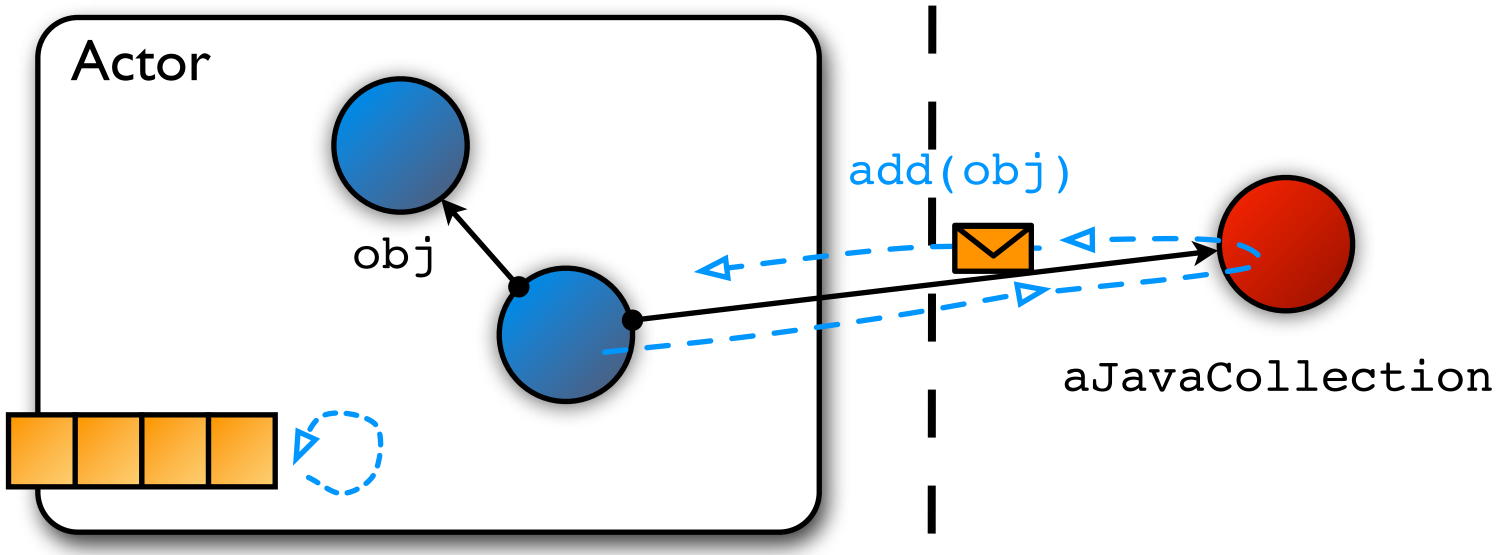
Actors as Threads

```
def obj := object: { ... };
aJavaCollection.add(obj);
```



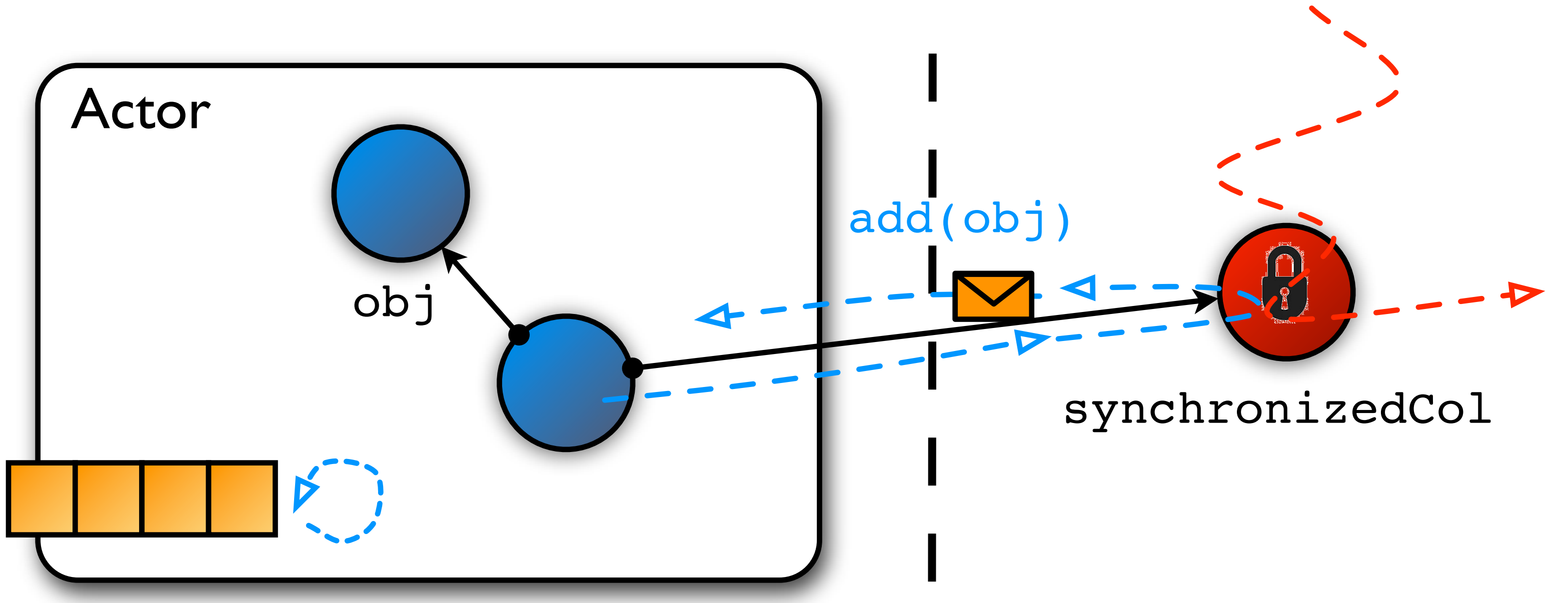
Actors as Threads

```
def obj := object: { ... };
aJavaCollection.add(obj);
```



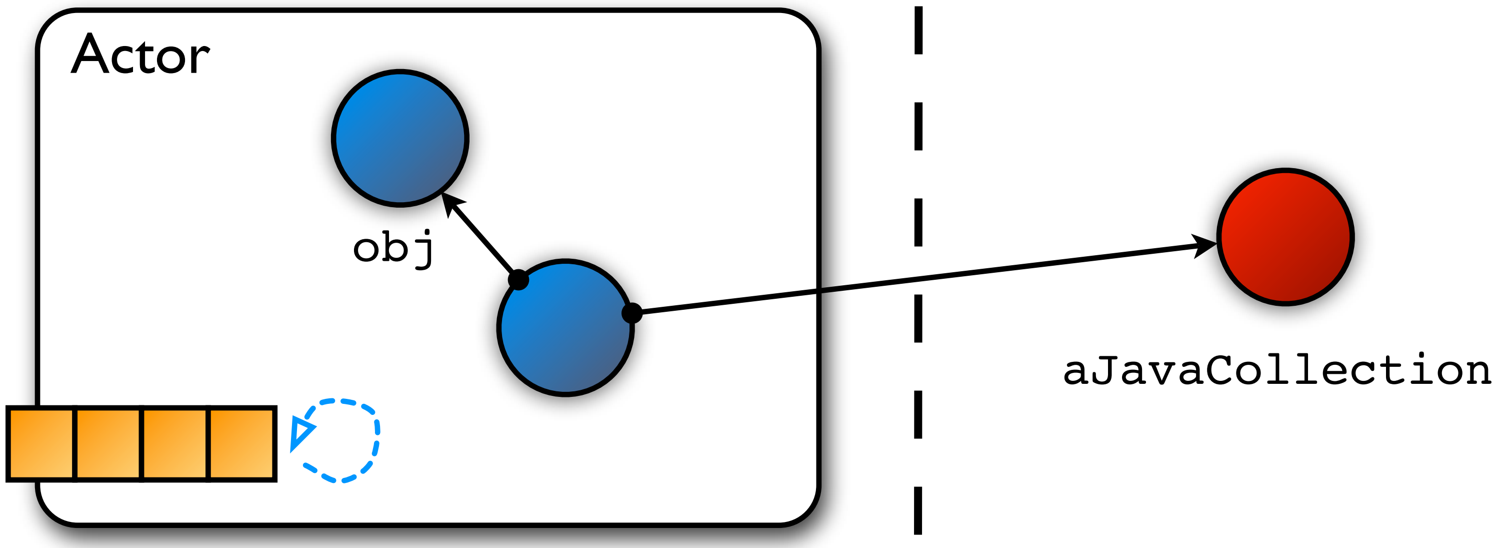
Actors as Threads

```
def obj := object: { ... };
aJavaCollection.add(obj);
```



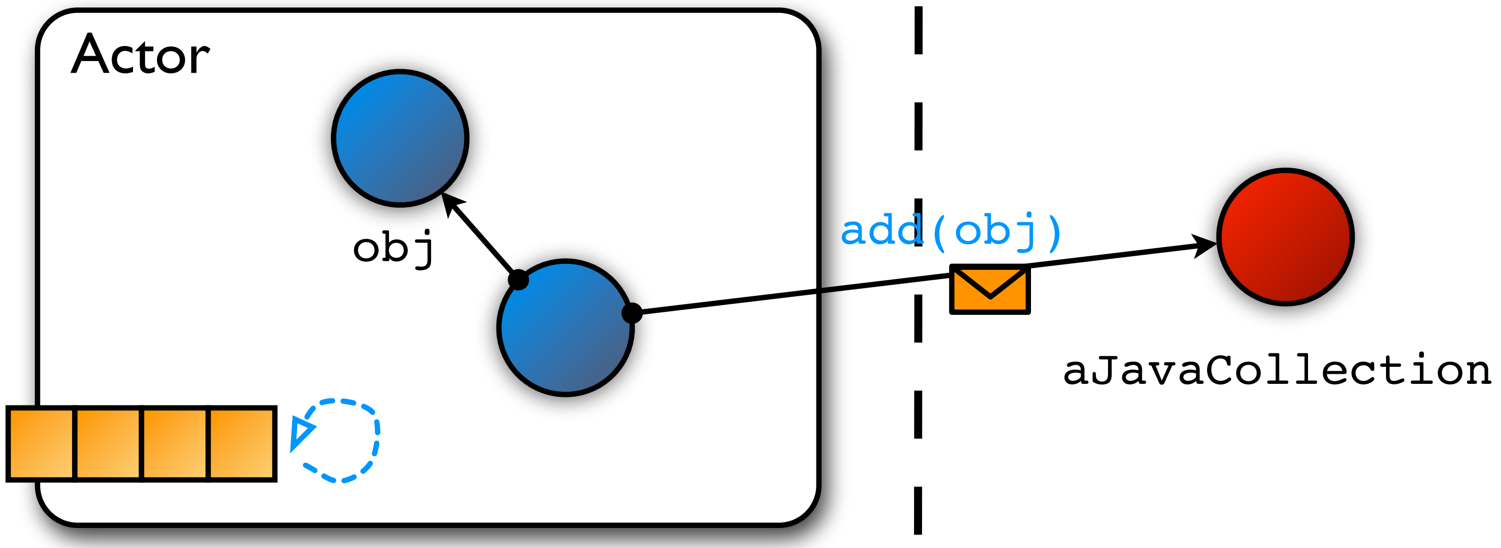
Actors as Threads

```
def obj := object: {
  def compareTo(other) { ... }
}

aJavaCollection.add(obj);
```



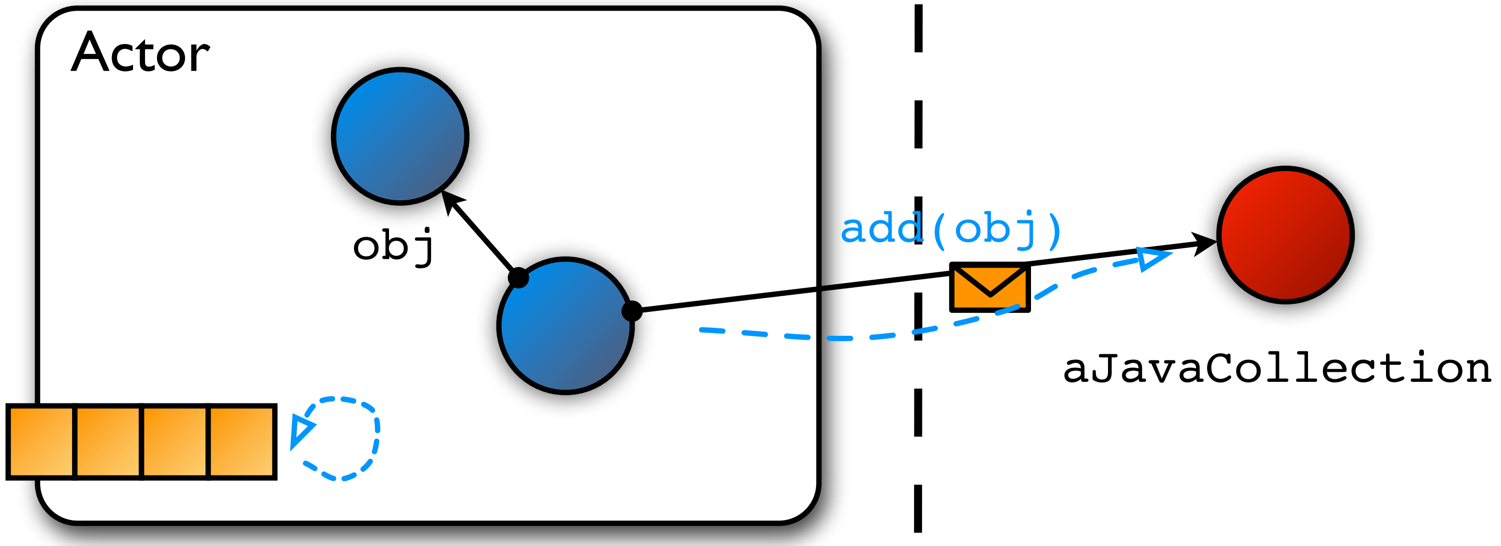
Actors as Threads

```
def obj := object: {
  def compareTo(other) { ... }
}

aJavaCollection.add(obj);
```



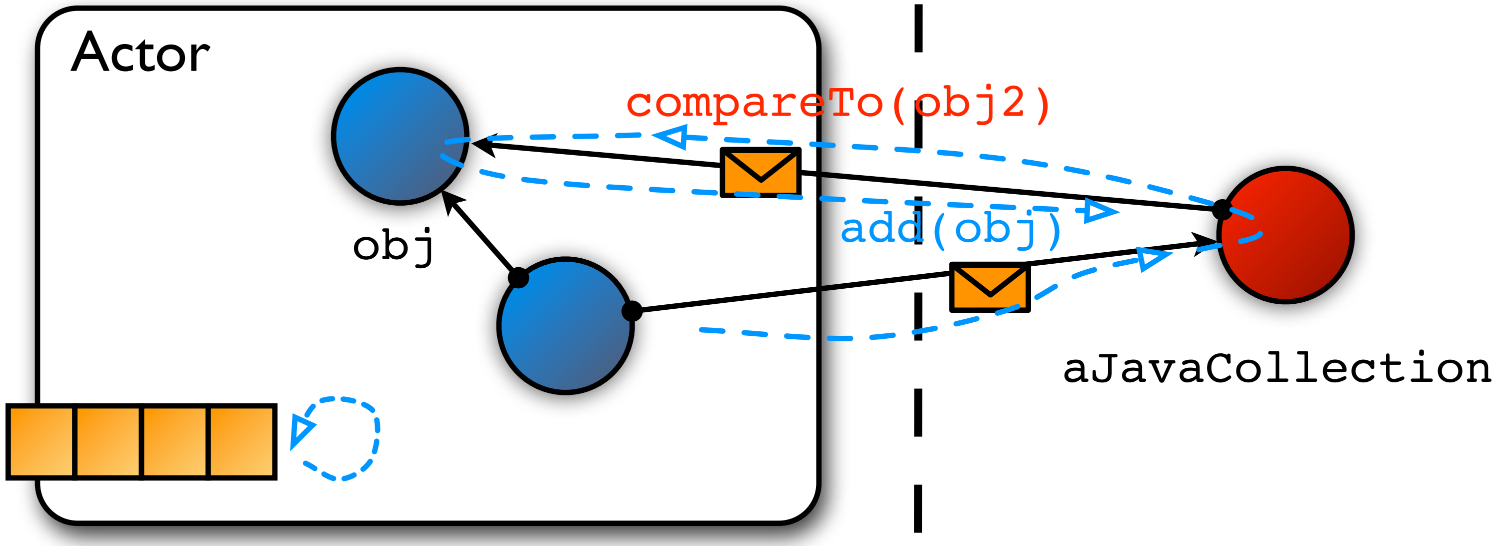
Actors as Threads

```
def obj := object: {
  def compareTo(other) { ... }
}

aJavaCollection.add(obj);
```



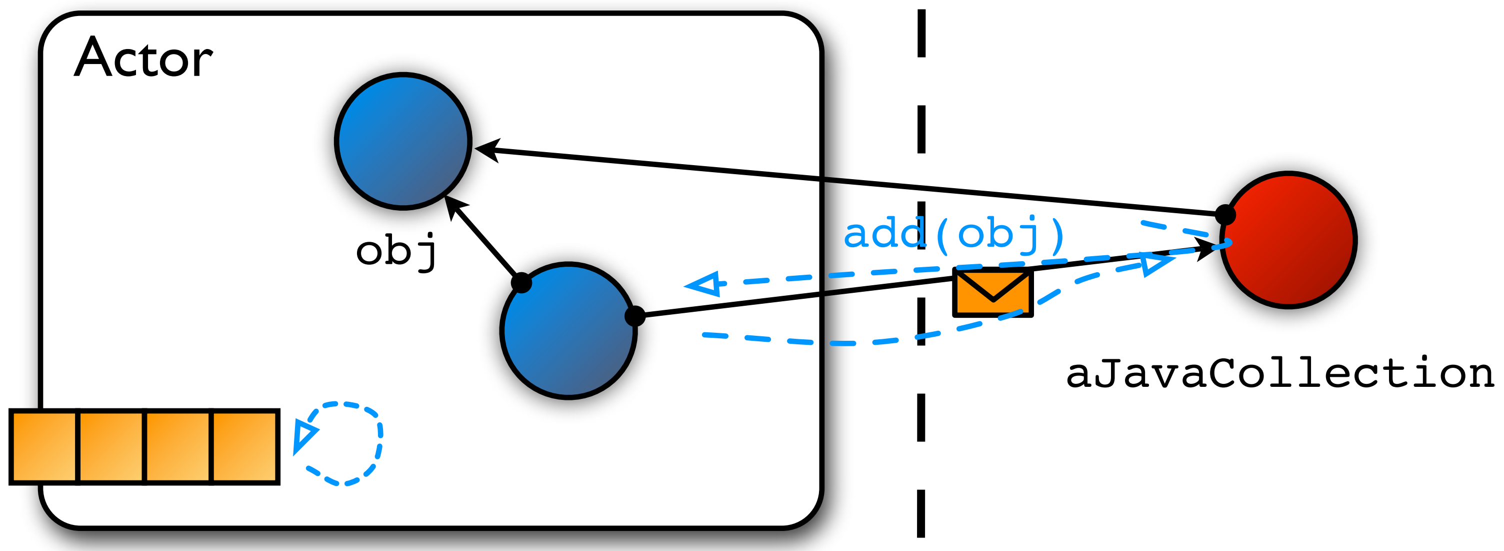
Actors as Threads

```
def obj := object: {
  def compareTo(other) { ... }
}

aJavaCollection.add(obj);
```



Actors as Threads

```
def obj := object: {
  def compareTo(other) { ... }
}

aJavaCollection.add(obj);
```



Threads as Actors

```
interface junit.framework.Test {
  public int countTestCases();
  public void run(TestResult r);
}
```

```
def ambientTalkTest := object: {
  def countTestCases() { ... }
  def run(result) { ... }
}
```

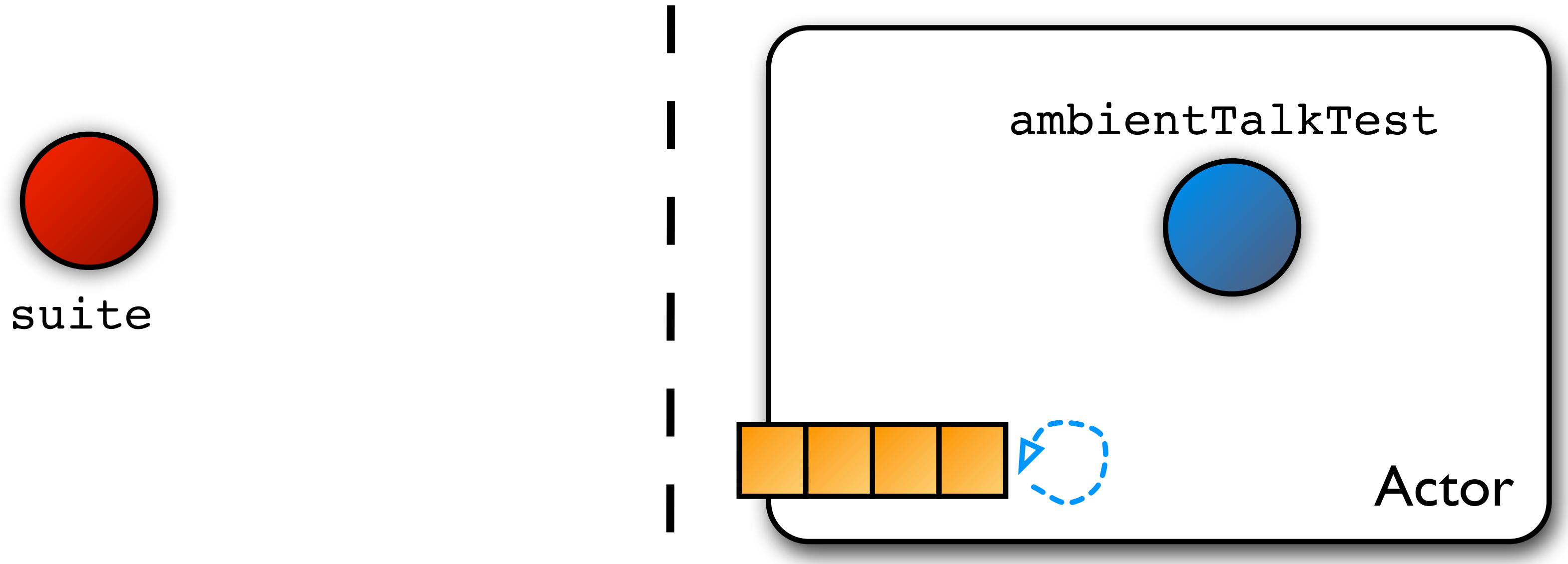
Threads as Actors

```
interface junit.framework.Test {
  public int countTestCases();
  public void run(TestResult r);
}
```

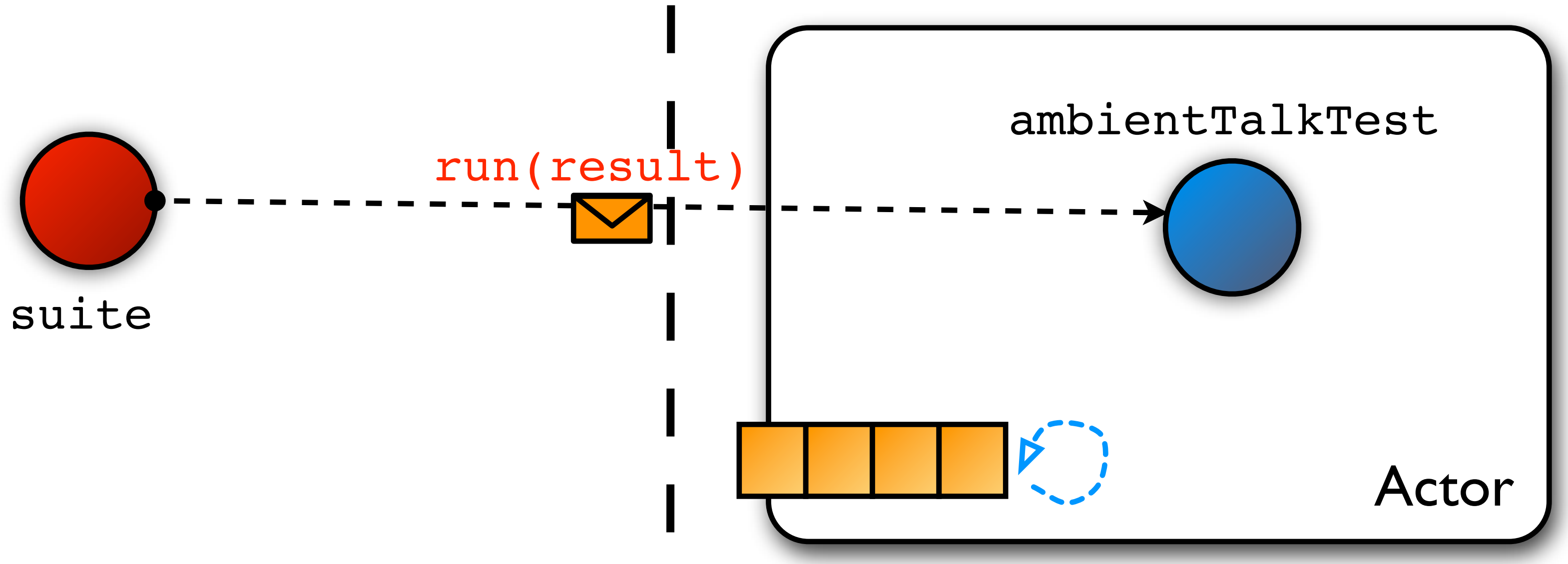
```
def ambientTalkTest := object: {
  def countTestCases() { ... }
  def run(result) { ... }
}
```

```
TestSuite suite = new TestSuite();
ATObject atUnitTest = /* load ambienttalk test */;
suite.addTest((Test) wrap(atUnitTest, Test.class));
suite.addTest(aJavaUnitTest);
junit.textuit.TestRunner.run(suite);
```

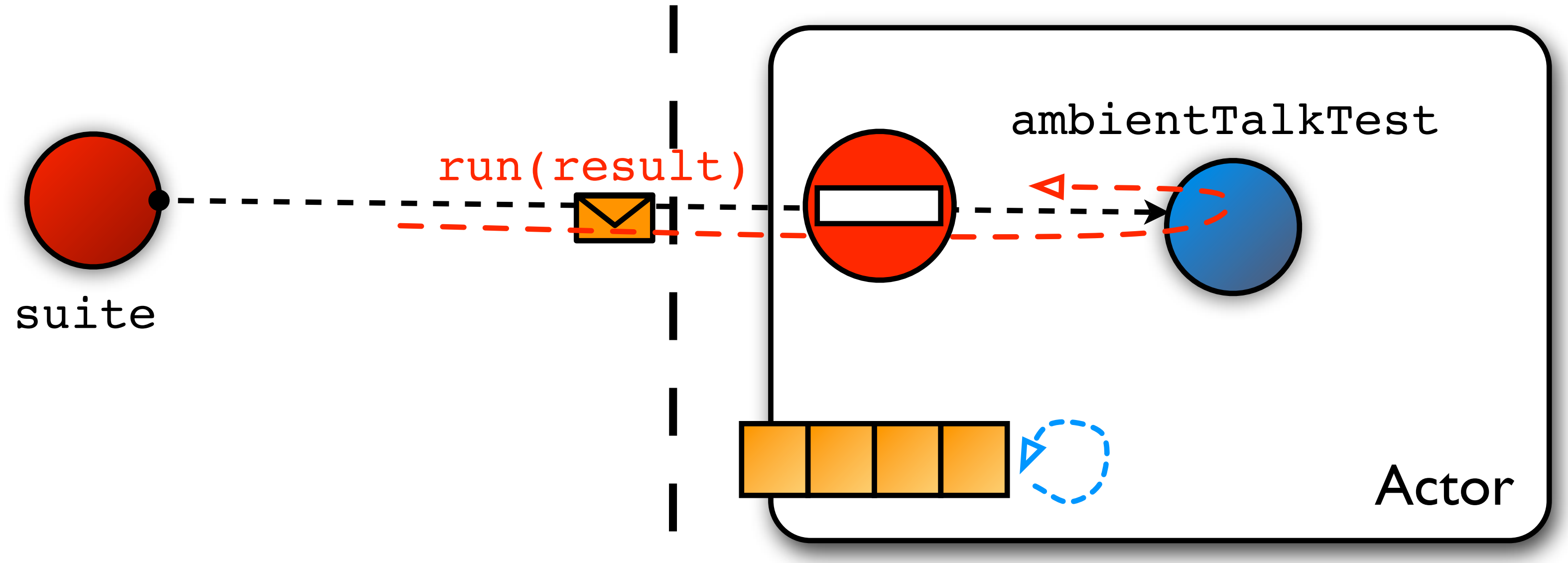
Threads as Actors



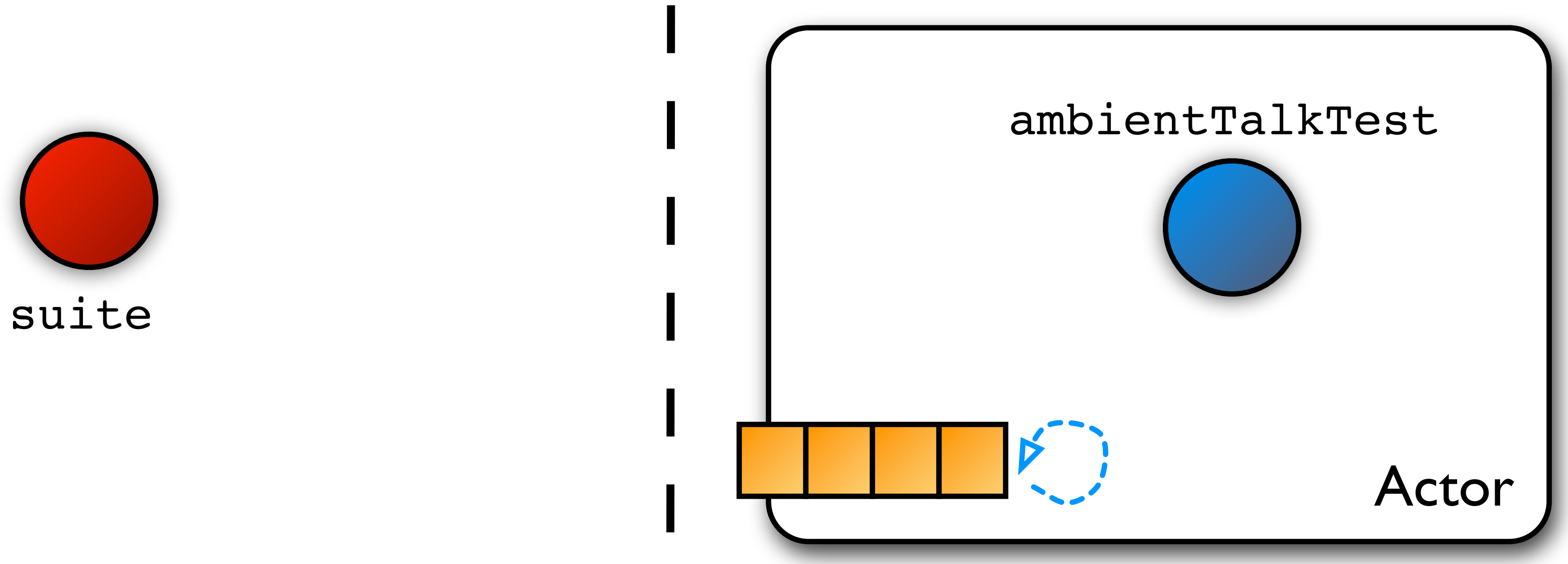
Threads as Actors



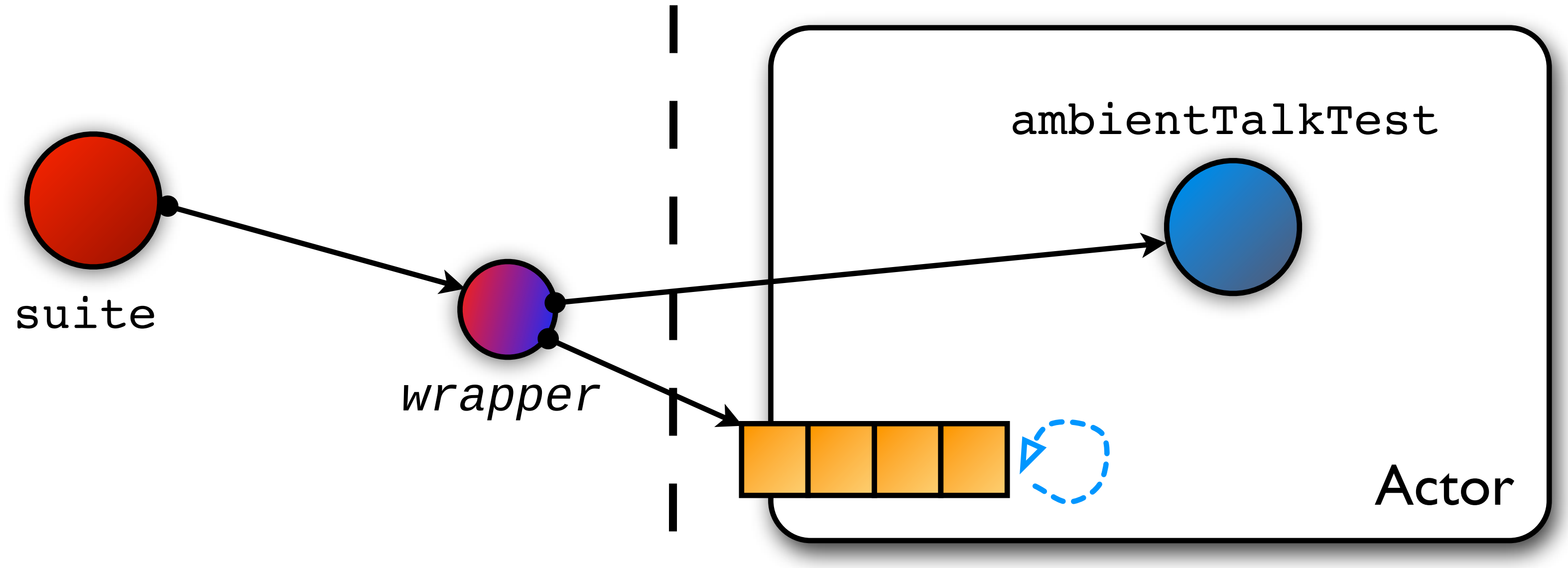
Threads as Actors



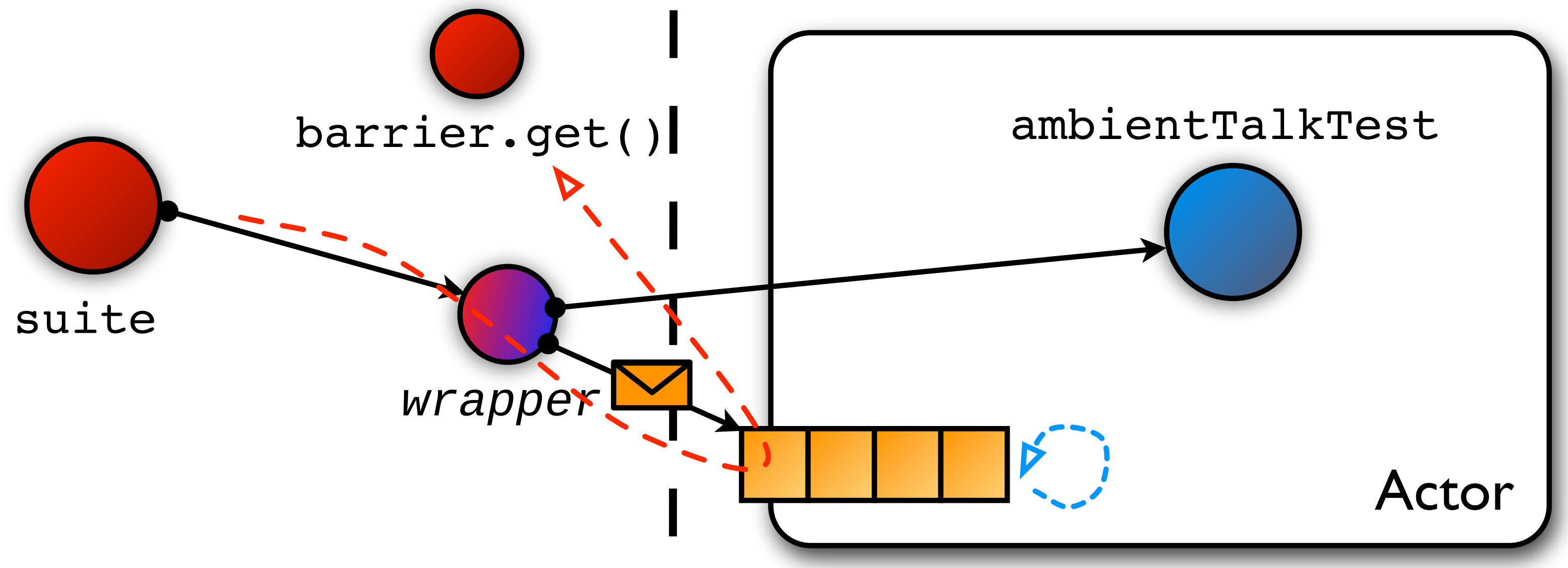
Threads as Actors



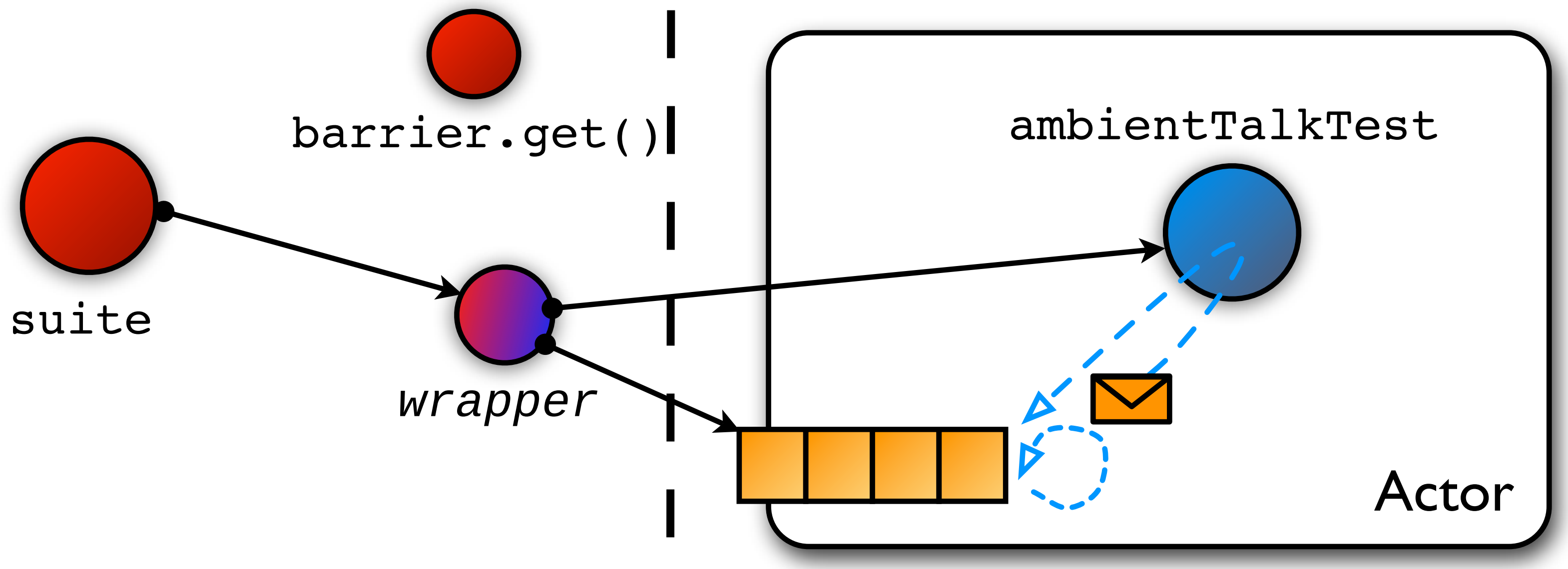
Threads as Actors



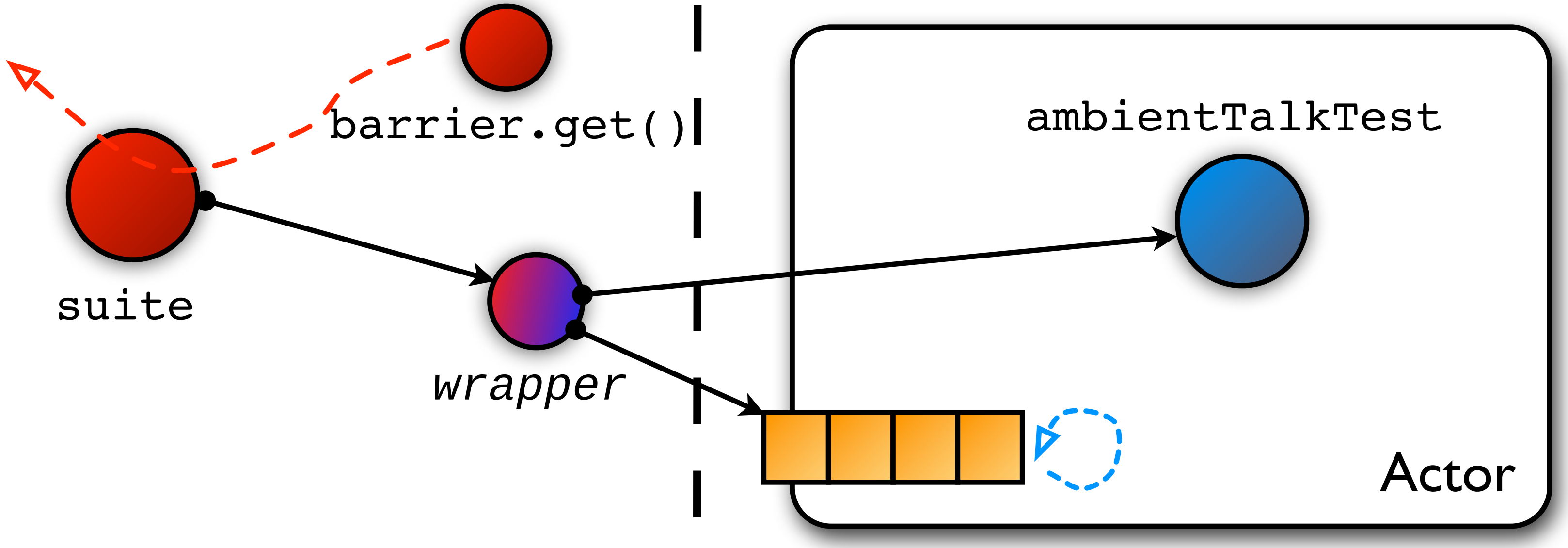
Threads as Actors



Threads as Actors



Threads as Actors



Threads as Actors

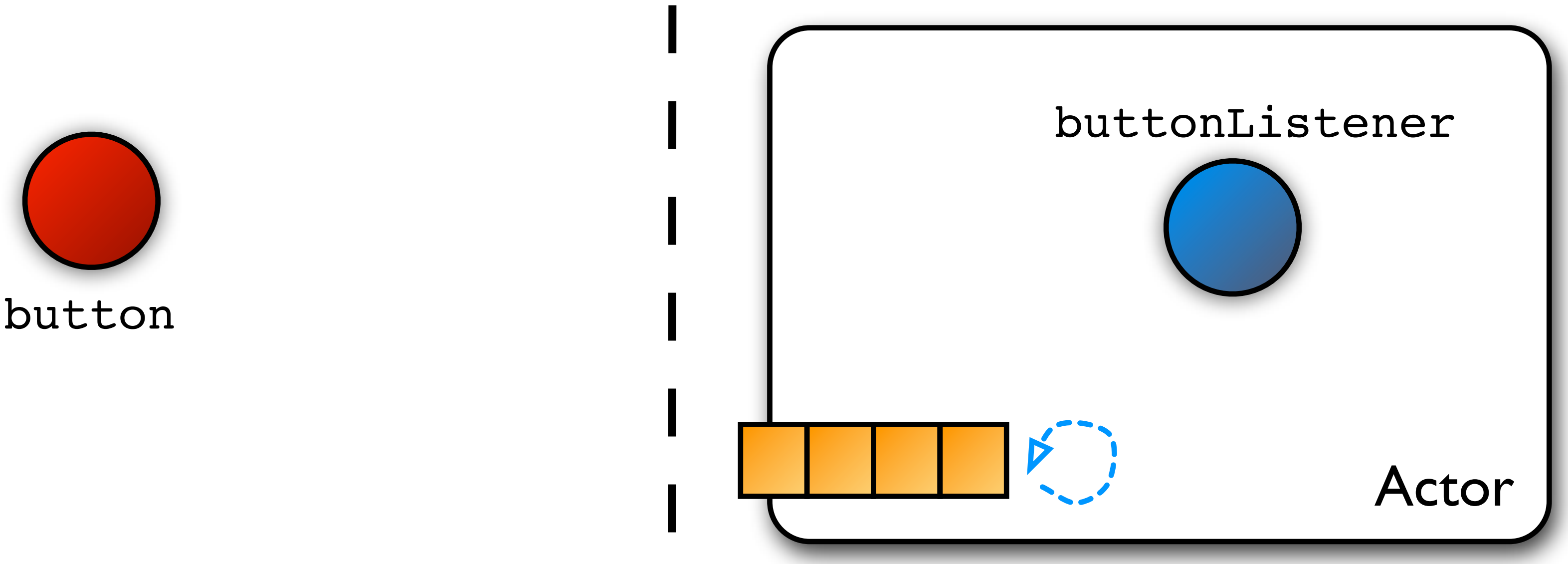
```
ActionListener l = ...;
l.actionPerformed(actionEvent);
```

```
def button := Button.new("Click Me");
button.addActionListener(object: {
  def actionPerformed(actionEvent) {
    ...
  }
});
```

Threads as Actors

```
ActionListener l = ...;
l.actionPerformed(actionEvent);
```

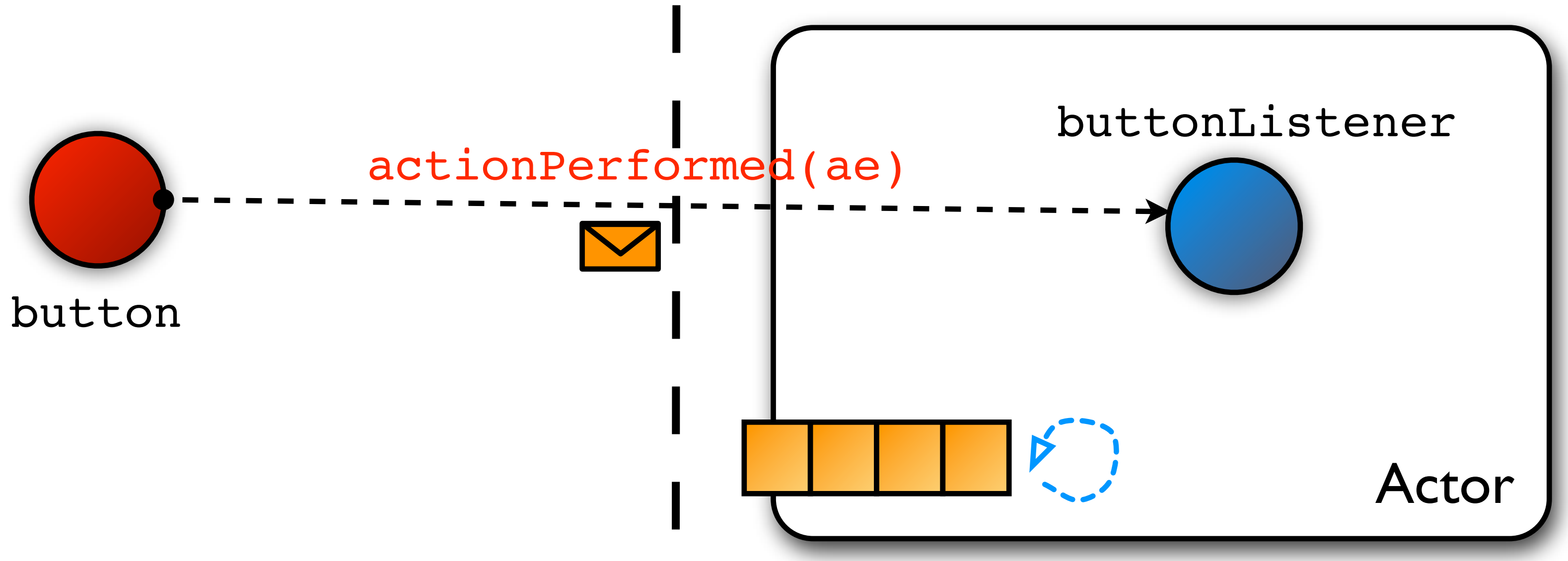
```
def button := Button.new("Click Me");
button.addActionListener(object: {
  def actionPerformed(actionEvent) {
    ...
  }
});
```



Threads as Actors

```
ActionListener l = ...;
l.actionPerformed(actionEvent);
```

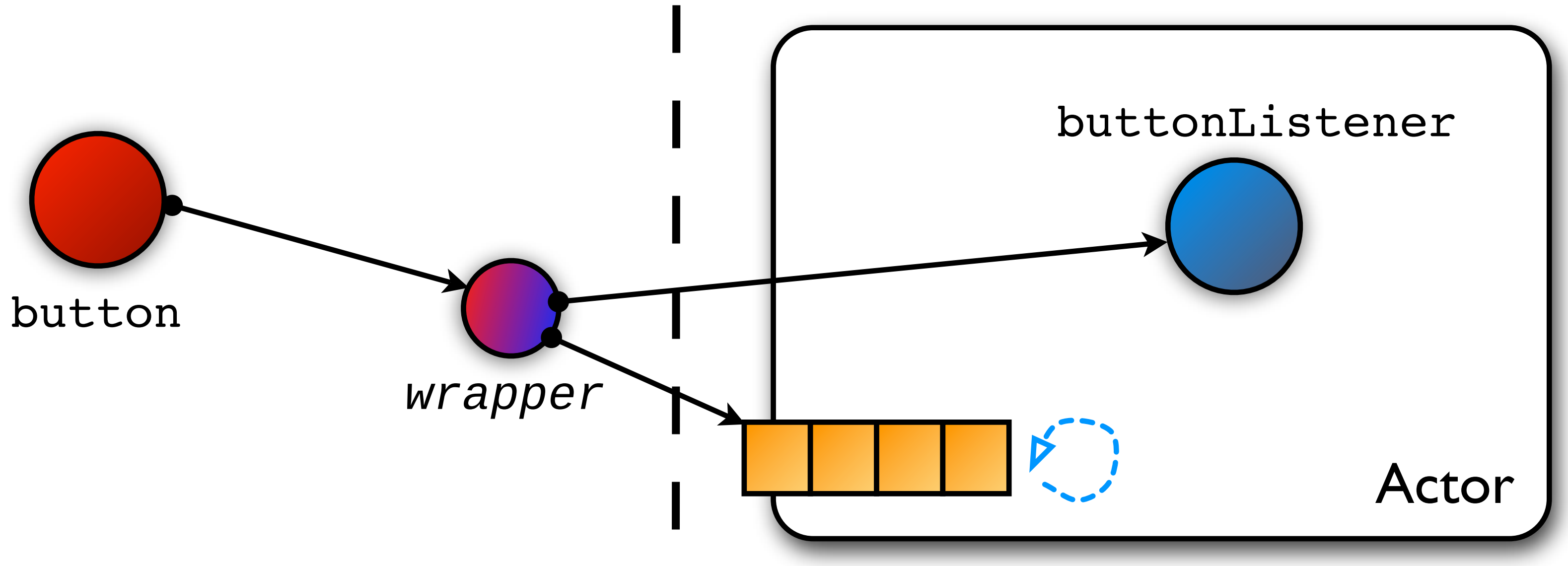
```
def button := Button.new("Click Me");
button.addActionListener(object: {
  def actionPerformed(actionEvent) {
    ...
  }
});
```



Threads as Actors

```
ActionListener l = ...;
l.actionPerformed(actionEvent);
```

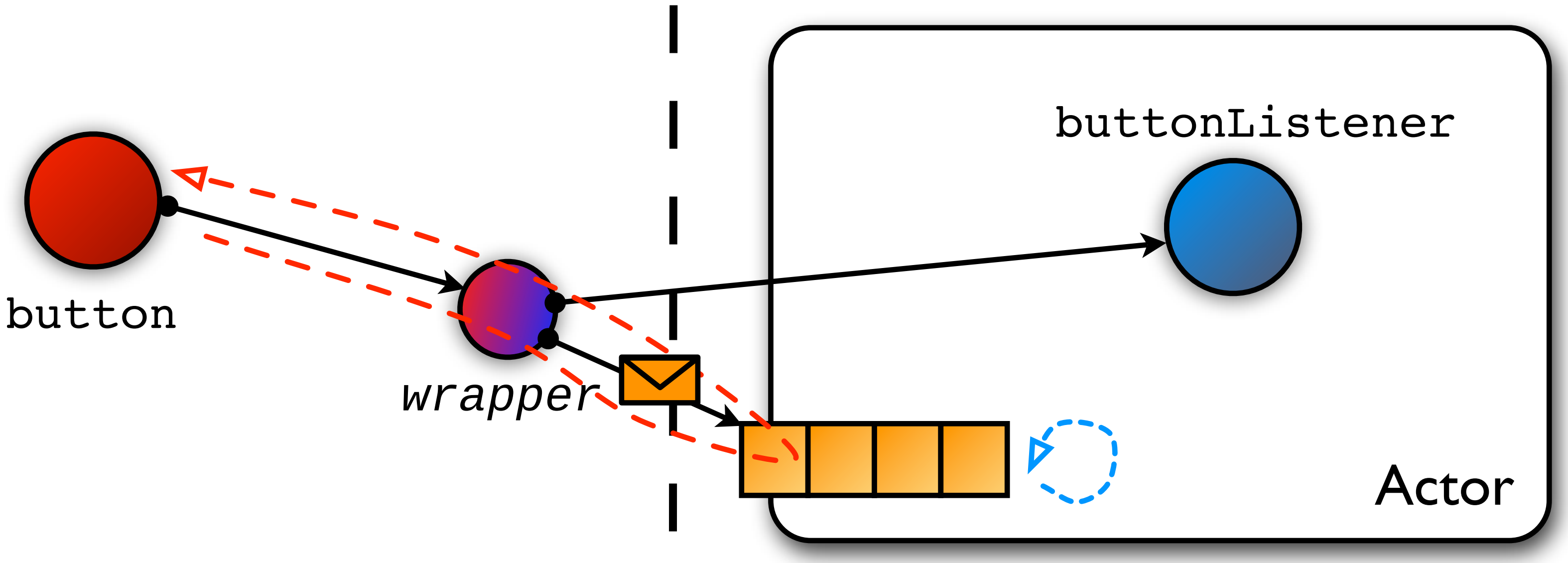
```
def button := Button.new(“Click Me”);
button.addActionListener(object: {
  def actionPerformed(actionEvent) {
    ...
  }
});
```



Threads as Actors

```
ActionListener l = ...;
l.actionPerformed(actionEvent);
```

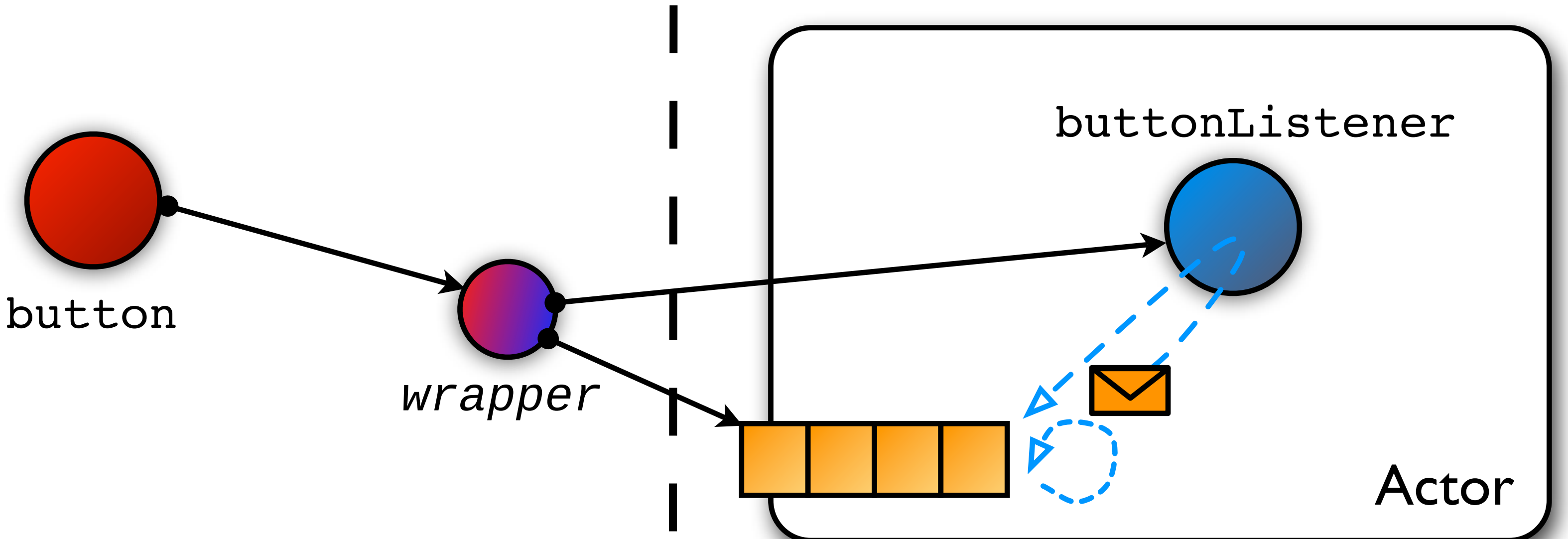
```
def button := Button.new("Click Me");
button.addActionListener(object: {
  def actionPerformed(actionEvent) {
    ...
  }
});
```



Threads as Actors

```
ActionListener l = ...;
l.actionPerformed(actionEvent);
```

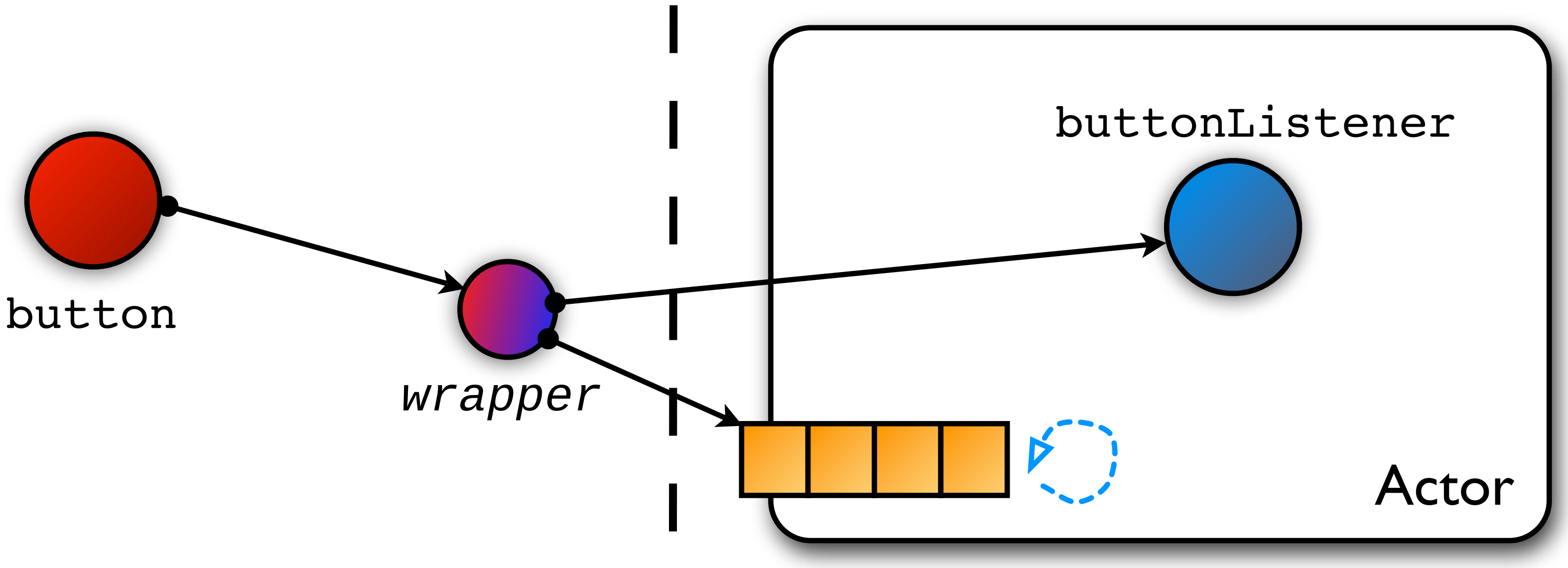
```
def button := Button.new("Click Me");
button.addActionListener(object: {
  def actionPerformed(actionEvent) {
    ...
  }
});
```



Threads as Actors

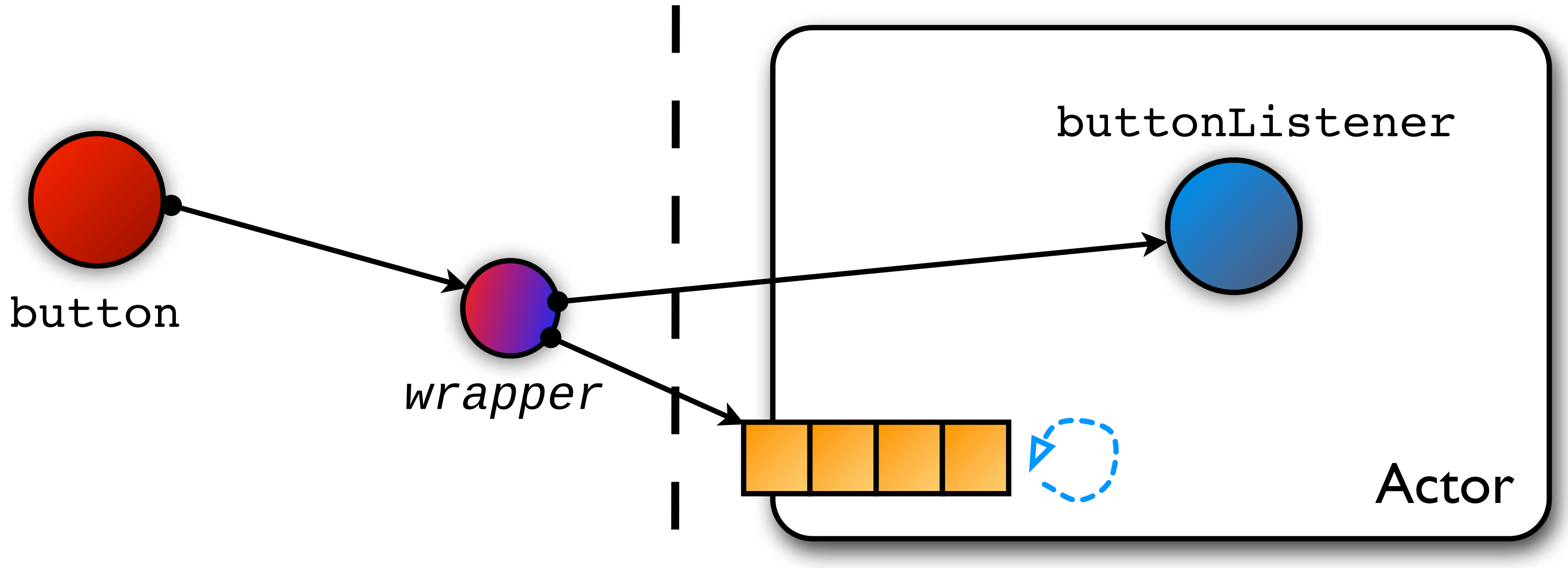
```
ActionListener l = ...;
l.actionPerformed(actionEvent);
```

```
def button := Button.new("Click Me");
button.addActionListener(object: {
  def actionPerformed(actionEvent) {
    ...
  }
});
```

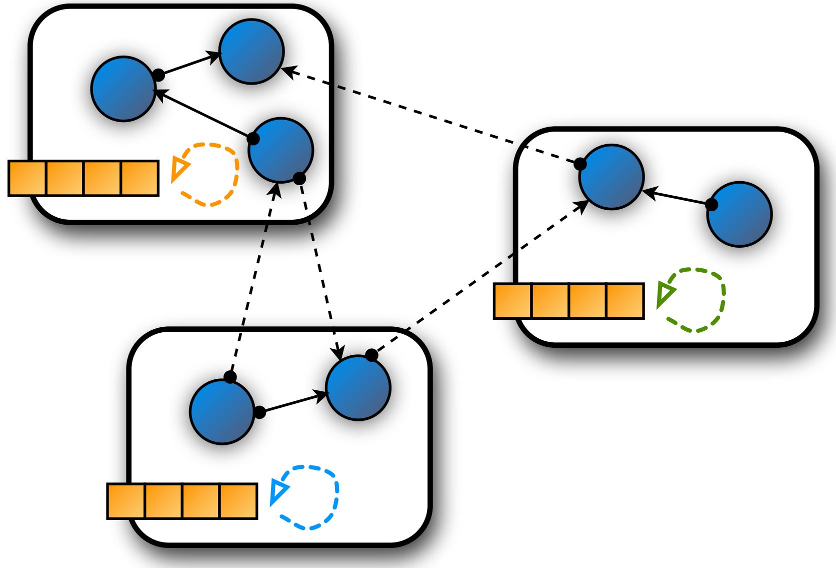


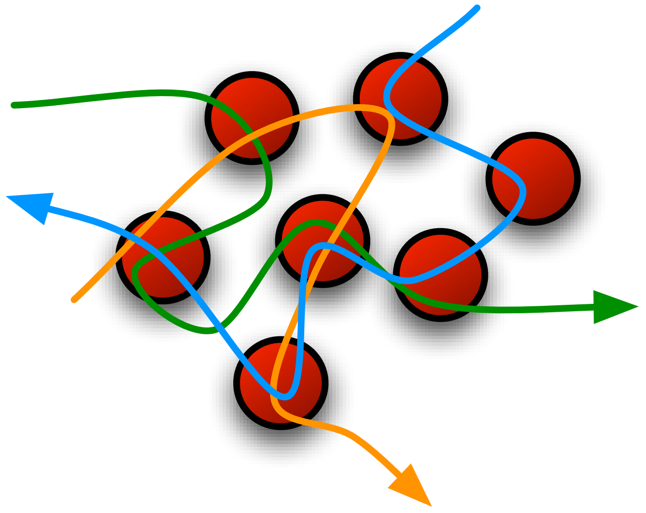
Threads as Actors

```
interface I extends java.util.EventListener {
  public void event(...);
}
```

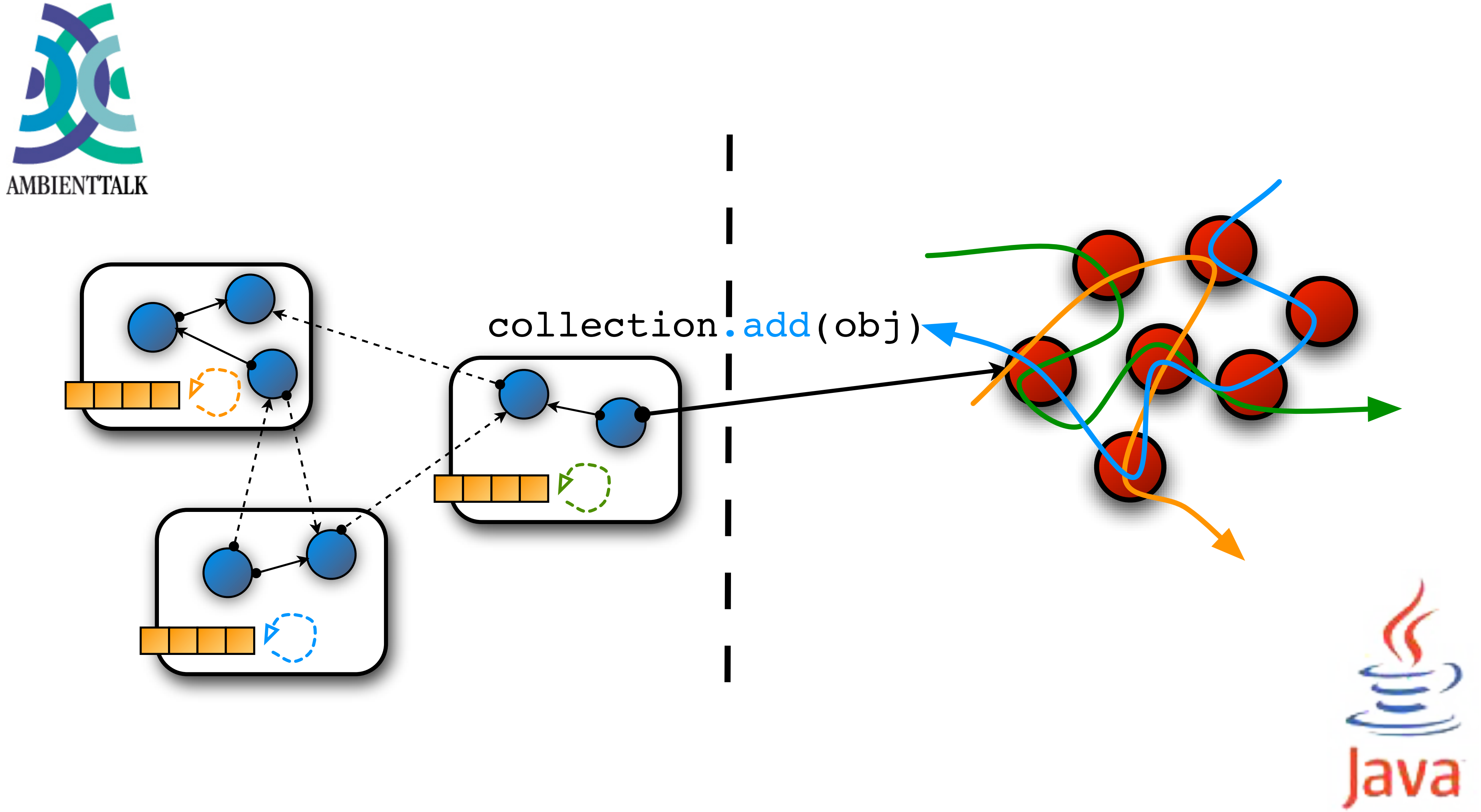


Summary

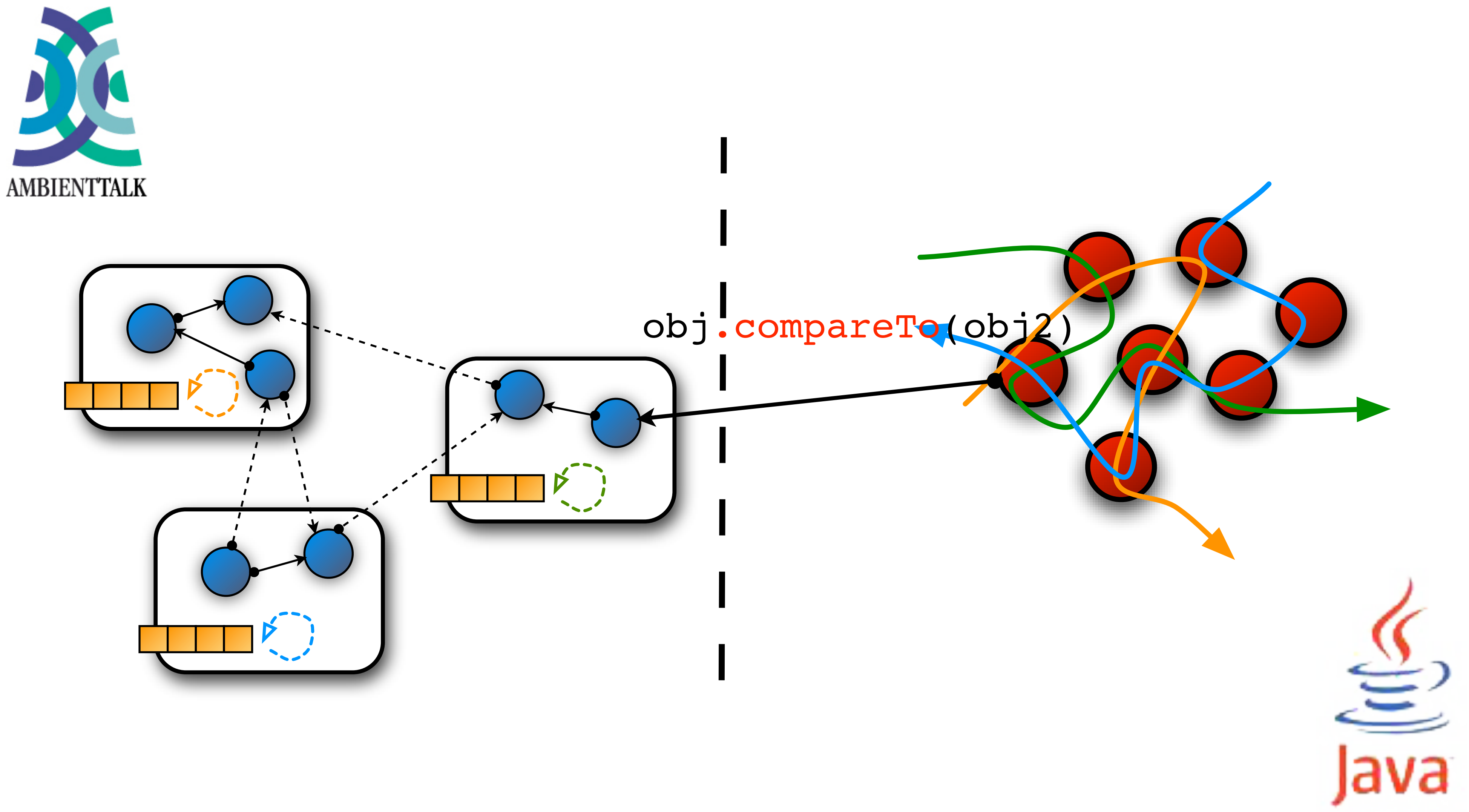




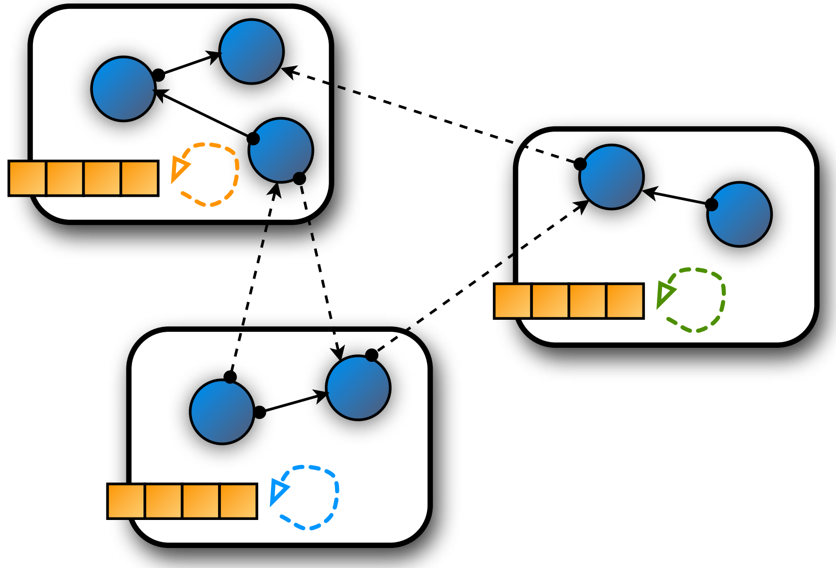
Summary

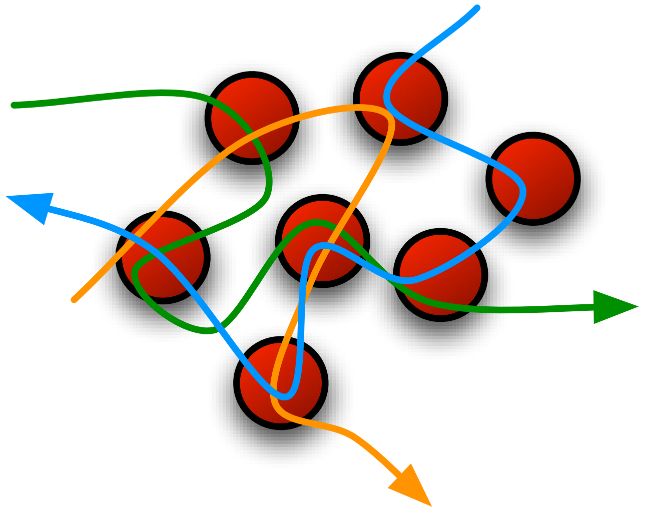


Summary

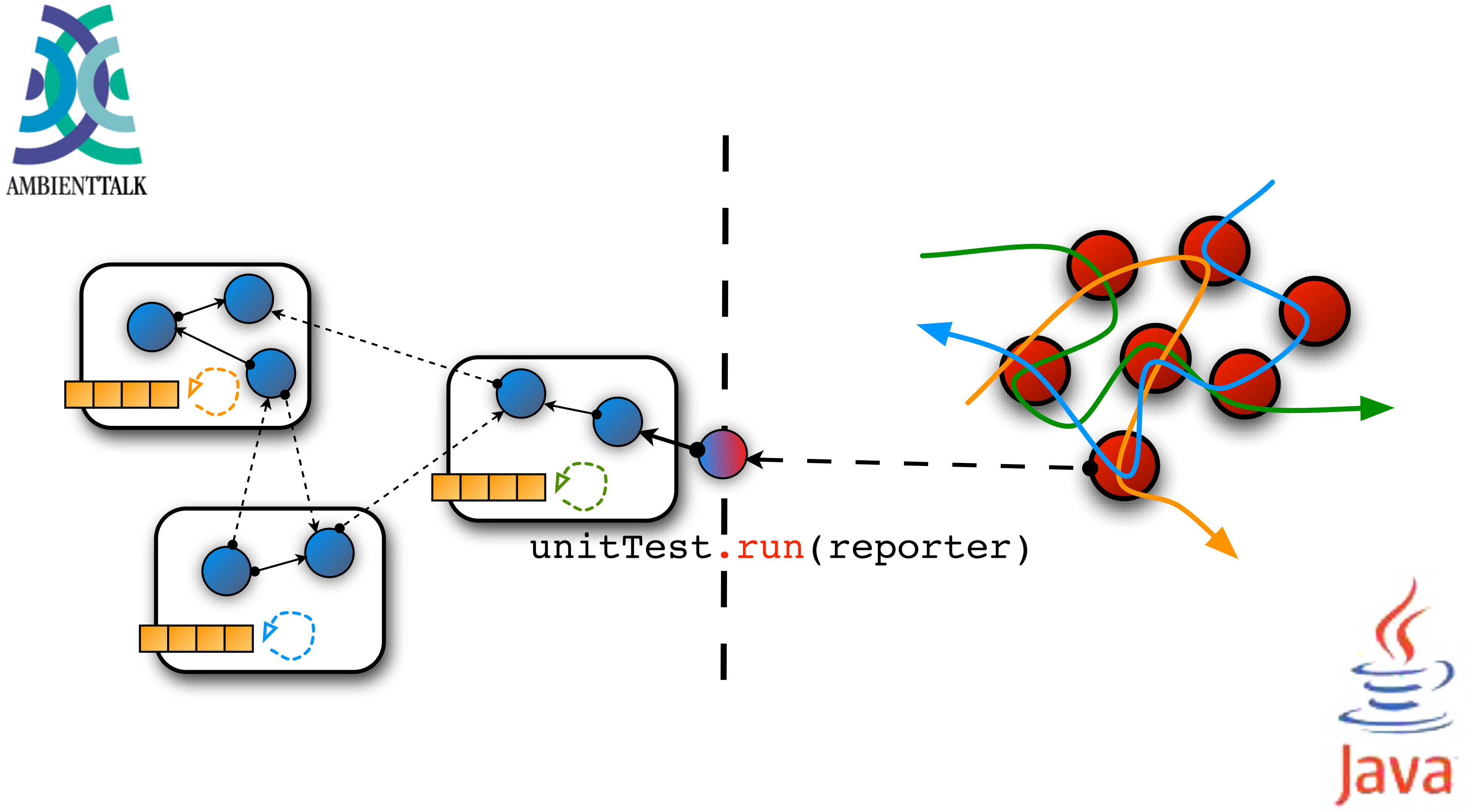


Summary



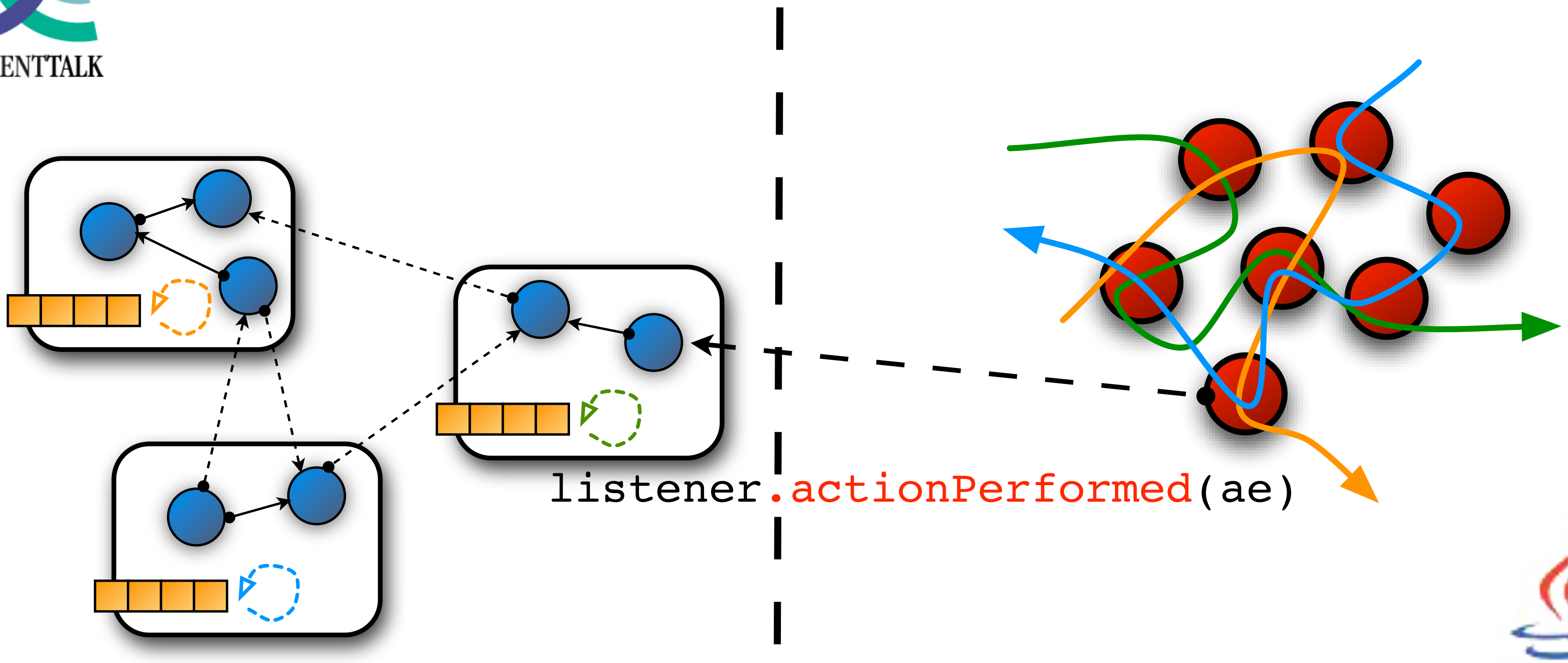


Summary



Summary

AMBIENTTALK



Java

Experience

- AmbientTalk using Java: AWT and Swing for GUI construction
- Java using AmbientTalk: JEdit plugin for collaborative text editing
- Self/Squeak's Morphic UI framework in AmbientTalk

Conclusions

- AmbientTalk: object-oriented (distributed) event-driven programming
- Symbiotic Thread/Actor mapping:
  - AmbientTalk invocations proceed immediately
  - Automatic synchronization of Java invocations
  - Support for Java "event notifications" (listeners)

AMBIENTTALK

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