AmbientTalk: Object-oriented Event-driven programming in Mobile Ad hoc Networks

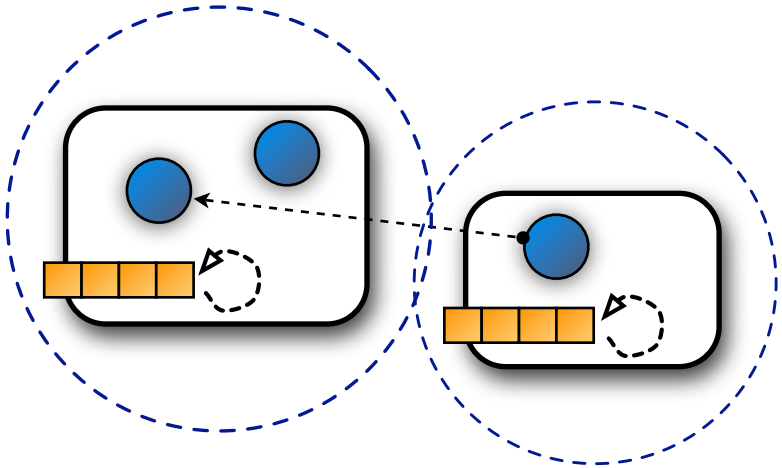
Tom Van Cutsem

Programming Technology Lab
Vrije Universiteit Brussel
Brussels, Belgium

LAMP - EPFL, July 25th 2007, Lausanne

Context

Object-oriented programming languages



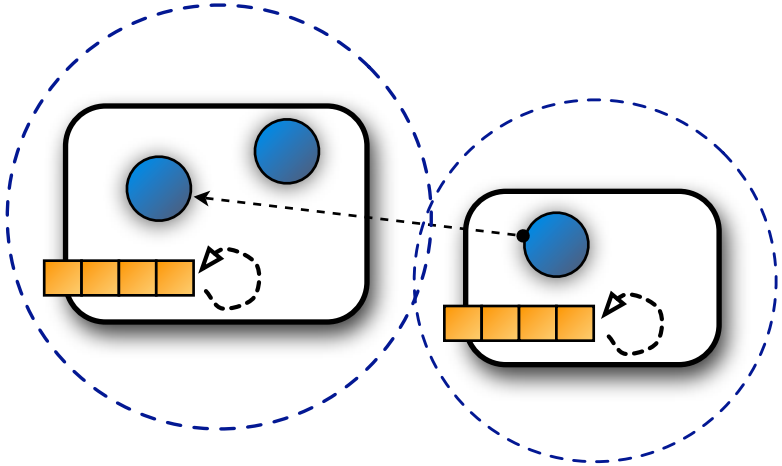
Software

Hardware

Pervasive Computing
(Mobile Networks)

Context

Object-oriented programming languages



Software

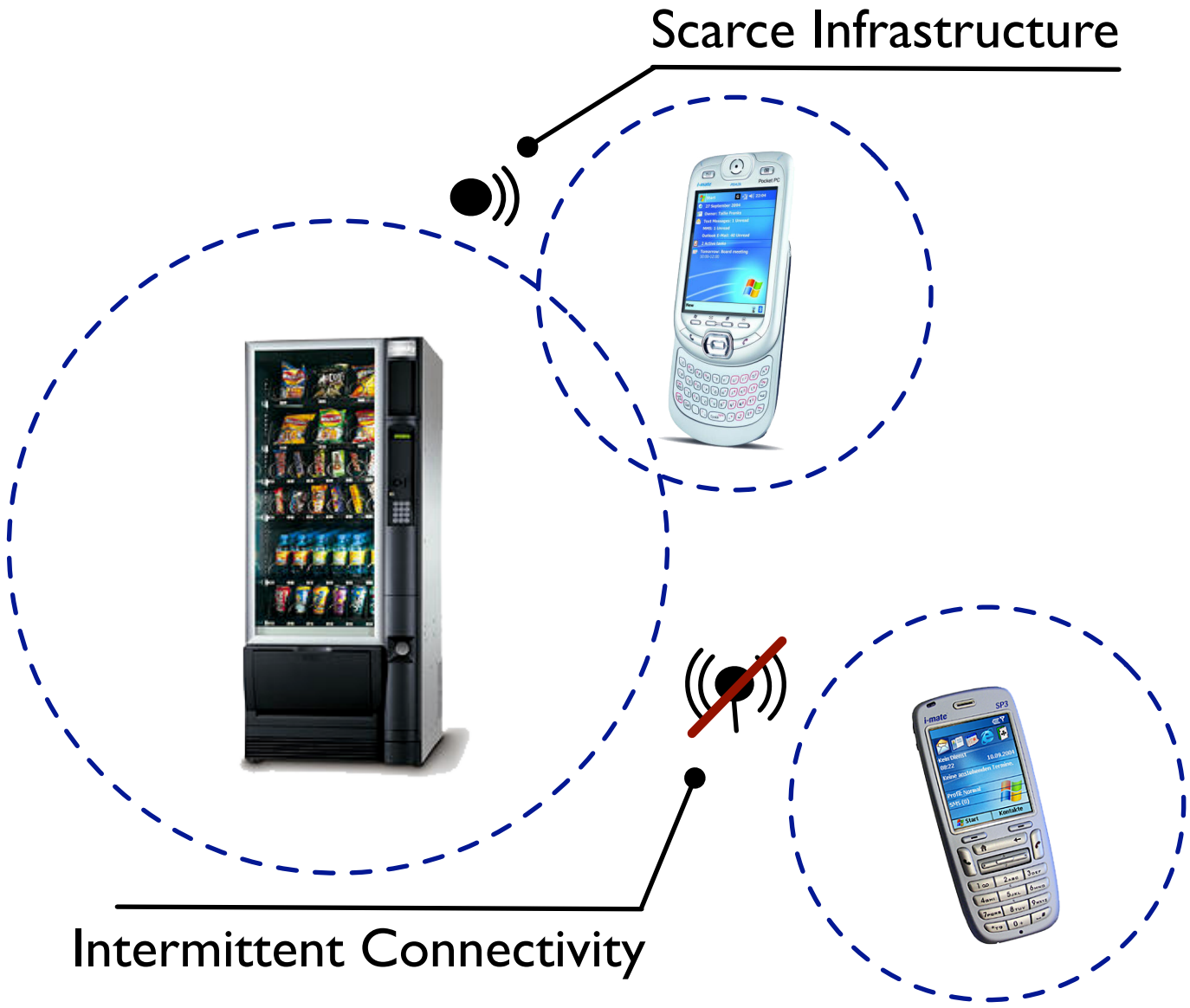
Hardware

Pervasive Computing
(Mobile Networks)

Mobile Ad hoc Networks

Mobile Ad hoc Networks

Mobile Ad hoc Networks



Loose Coupling

[Eugster et al. 03]

Decoupling communication in *Time* & *Synchronisation*

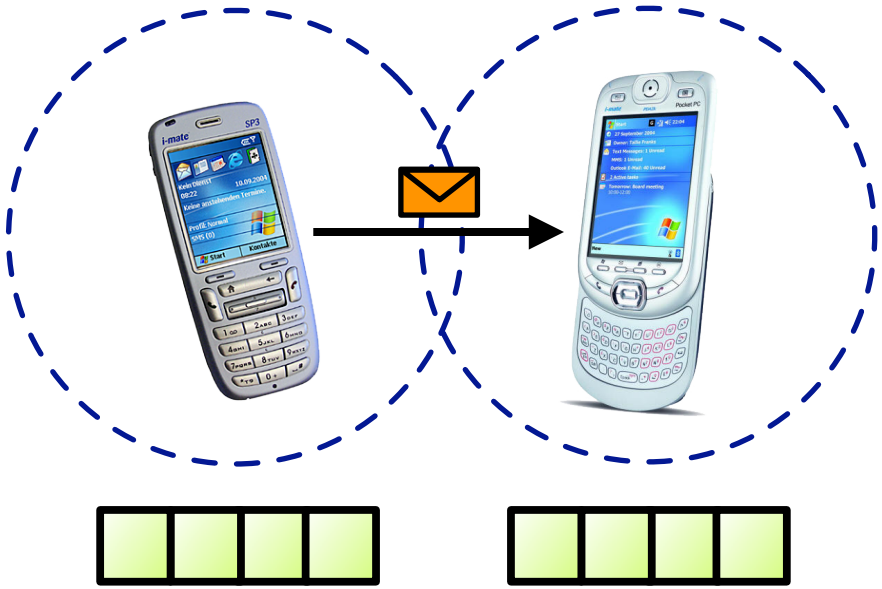
reduces impact of volatile connections

Loose Coupling

[Eugster et al. 03]

Decoupling communication in *Time* & *Synchronisation*

reduces impact of volatile connections



Loose Coupling

[Eugster et al. 03]

Decoupling communication in *Time* & *Synchronisation*

reduces impact of volatile connections

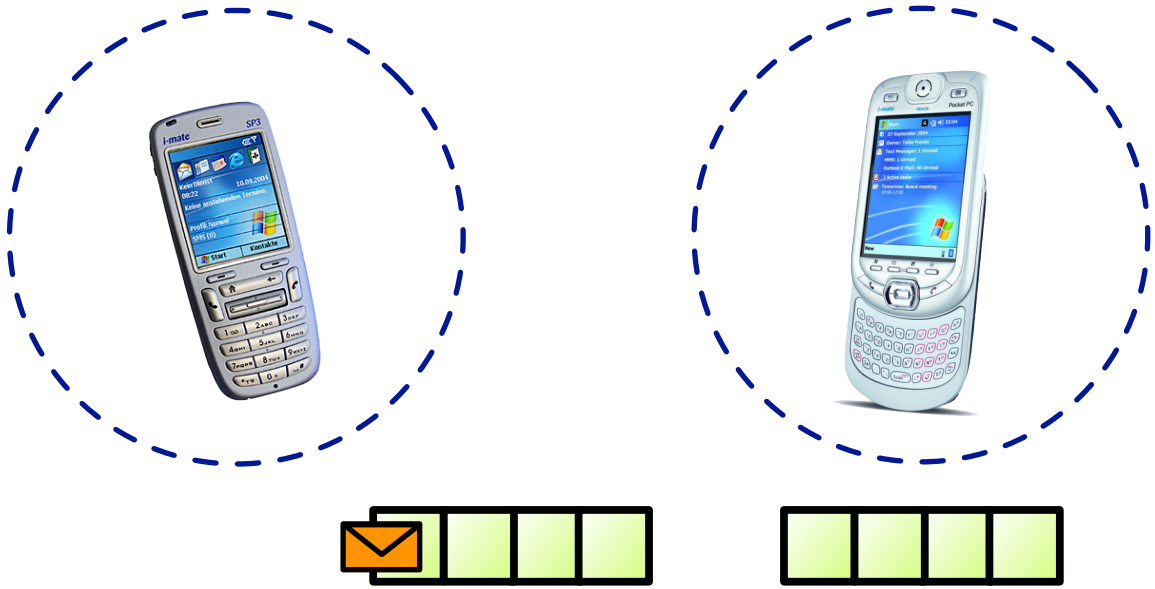
asynchronous send

Loose Coupling

[Eugster et al. 03]

Decoupling communication in *Time* & *Synchronisation*

reduces impact of volatile connections



Loose Coupling

[Eugster et al. 03]

Decoupling communication in *Time* & *Synchronisation*

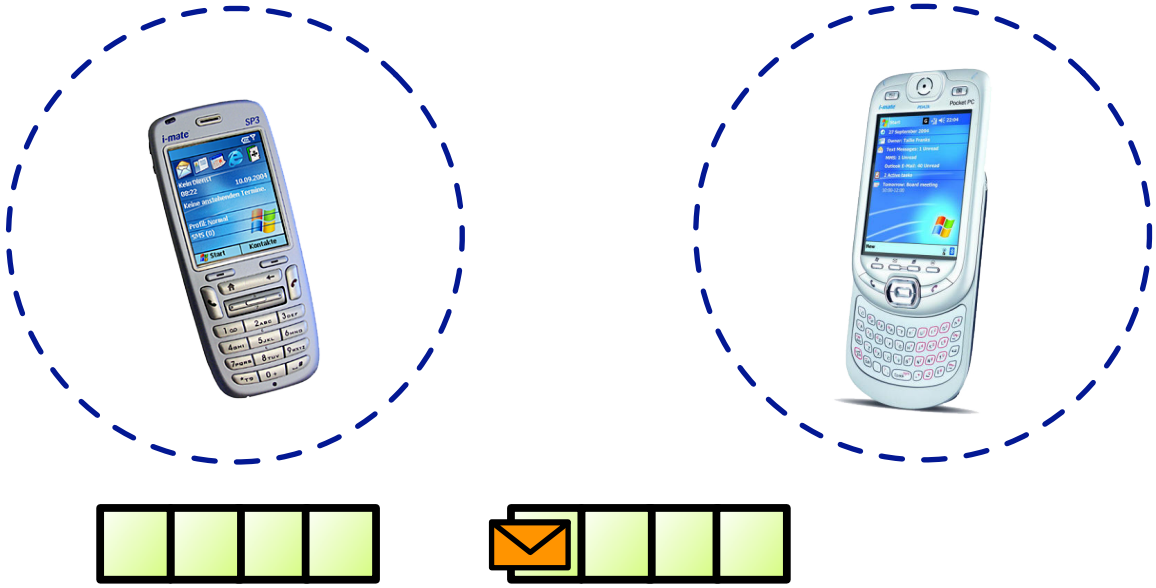
reduces impact of volatile connections

Loose Coupling

[Eugster et al. 03]

Decoupling communication in *Time* & *Synchronisation*

reduces impact of volatile connections

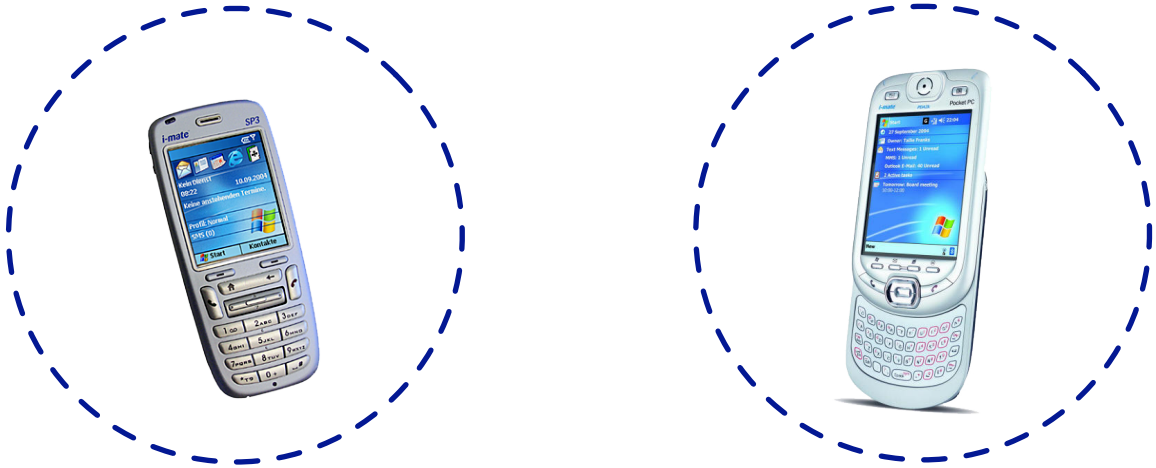


Loose Coupling

[Eugster et al. 03]

Decoupling communication in *Time* & *Synchronisation*

reduces impact of volatile connections



asynchronous receive

Loose Coupling

Decoupling communication in *Space*

enables ad hoc anonymous collaborations

Loose Coupling

Decoupling communication in *Space*

enables ad hoc anonymous collaborations

Loose Coupling

Decoupling communication in *Space*

enables ad hoc anonymous collaborations

provide service

Loose Coupling

Decoupling communication in *Space*

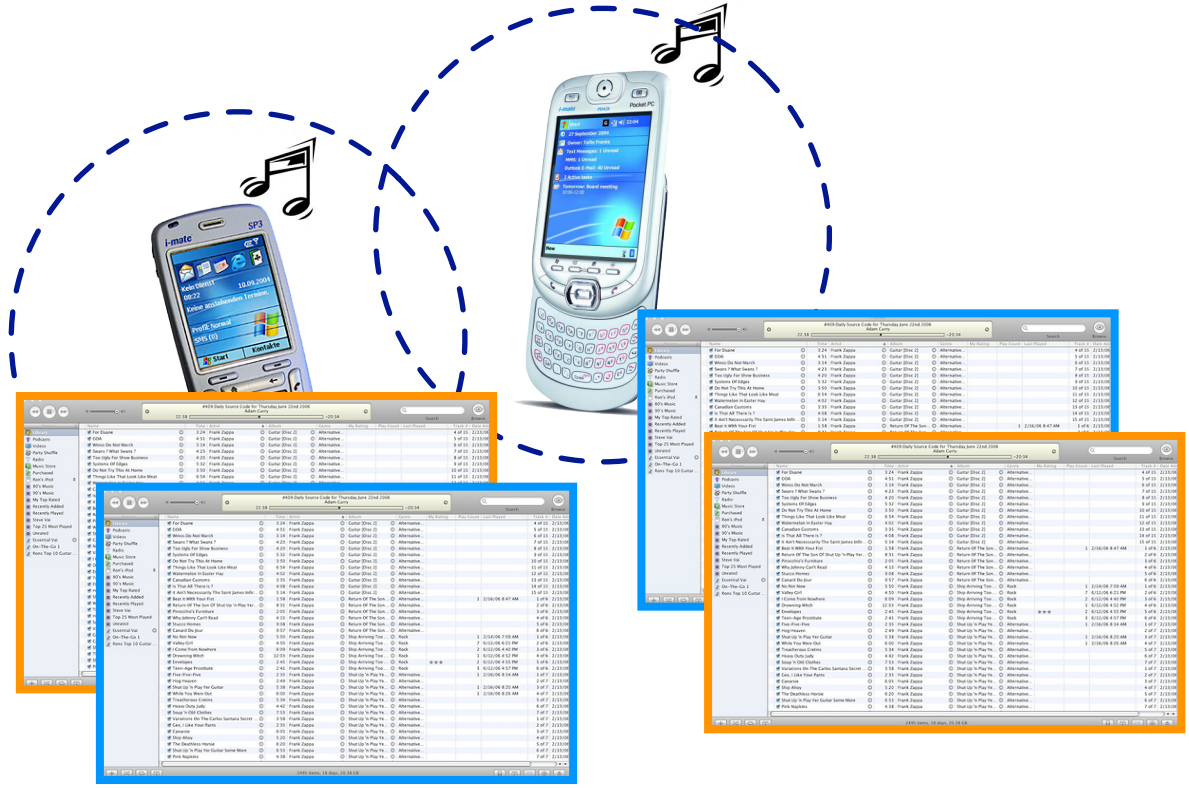
enables ad hoc anonymous collaborations

provide service require service

Example: music player

Example: music player

Example: music player



Example: music player

AmbientTalk: the language

- Distributed prototype-based object-oriented language
- Event-driven concurrency based on actors [Agha86]
- Future-type asynchronous message sends
- Built-in publish/subscribe engine for service discovery of remote objects

AmbientTalk: the project

- Started in 2005
- Small team: 3-6 people
- Interpreter (not optimised)
- Pure Java implementation
- Runs on J2ME/CDC phones

Objects

```
def Window := object: {
  def title := "Untitled";
  def init(t) {
    title := t;
    super := ClosedWindow;
  }
  def show() {
    super := OpenWindow;
  }

  def OpenWindow := object: {
    def draw() { ... }
  }
  def ClosedWindow := object: {
    def draw() { ... }
  }
}
```

- Prototypes
- Delegation
- Trait composition
- First-class delegation

```
def w := Window.new("Test");
w.draw();
```

Extensible language

```
def fac(n) {
  if: (n = 0) then: {
    1
  } else: {
    n * fac(n-1)
  }
}
```

Extensible language

```
def fac(n) {
  if: (n = 0) then: {
    1
  } else: {
    n * fac(n-1)
  }
}
```

```
def Button := jlobby.java.awt.Button;
def b := Button.new("test");
b.addActionListener(object: {
  def actionPerformed(ae) {
    system.println("button pressed");
  }
});
```

Extensible language

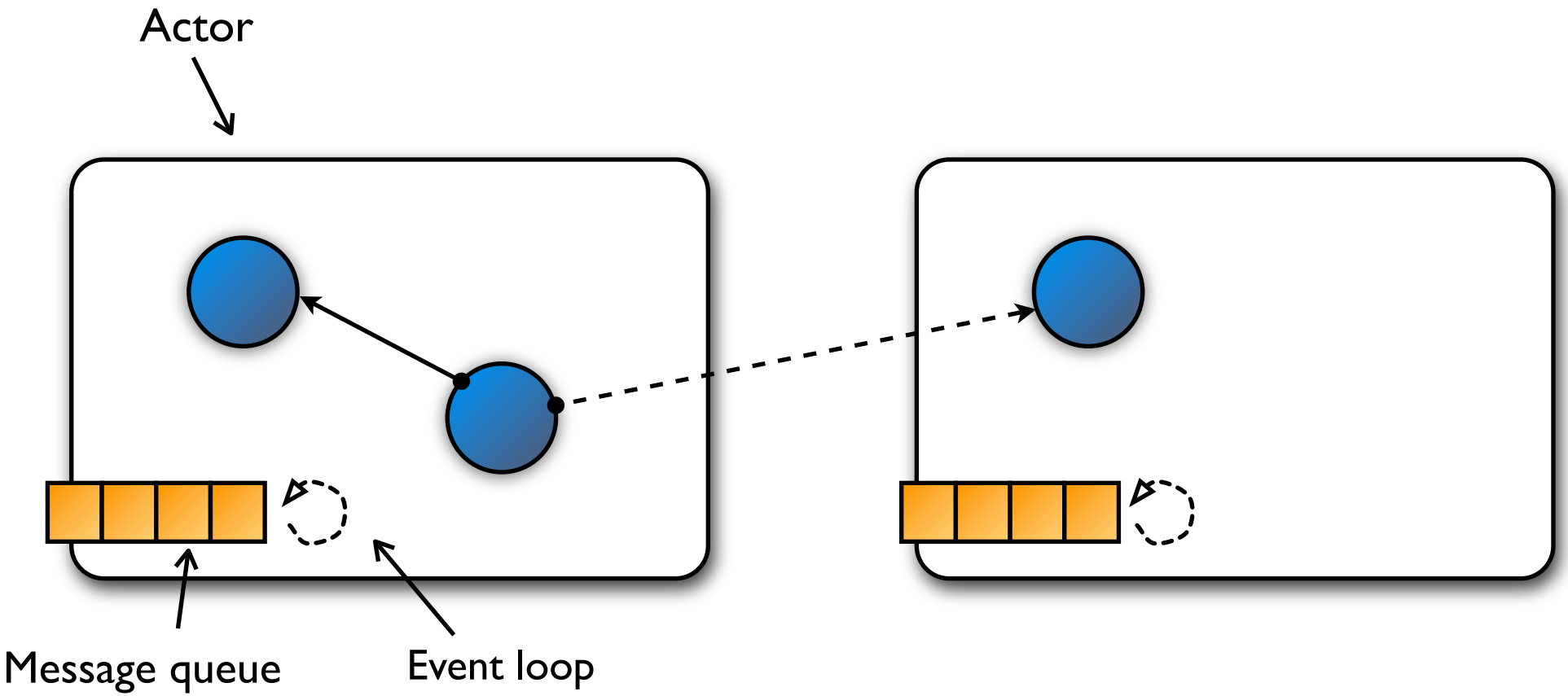
```
def fac(n) {
  if: (n = 0) then: {
    1
  } else: {
    n * fac(n-1)
  }
}
```

- Block closures
- Keyworded messages
- Interfacing with JVM
- Reflection

```
def Button := jlobby.java.awt.Button;
def b := Button.new("test");
b.addActionListener(object: {
  def actionPerformed(ae) {
    system.println("button pressed");
  }
});
```

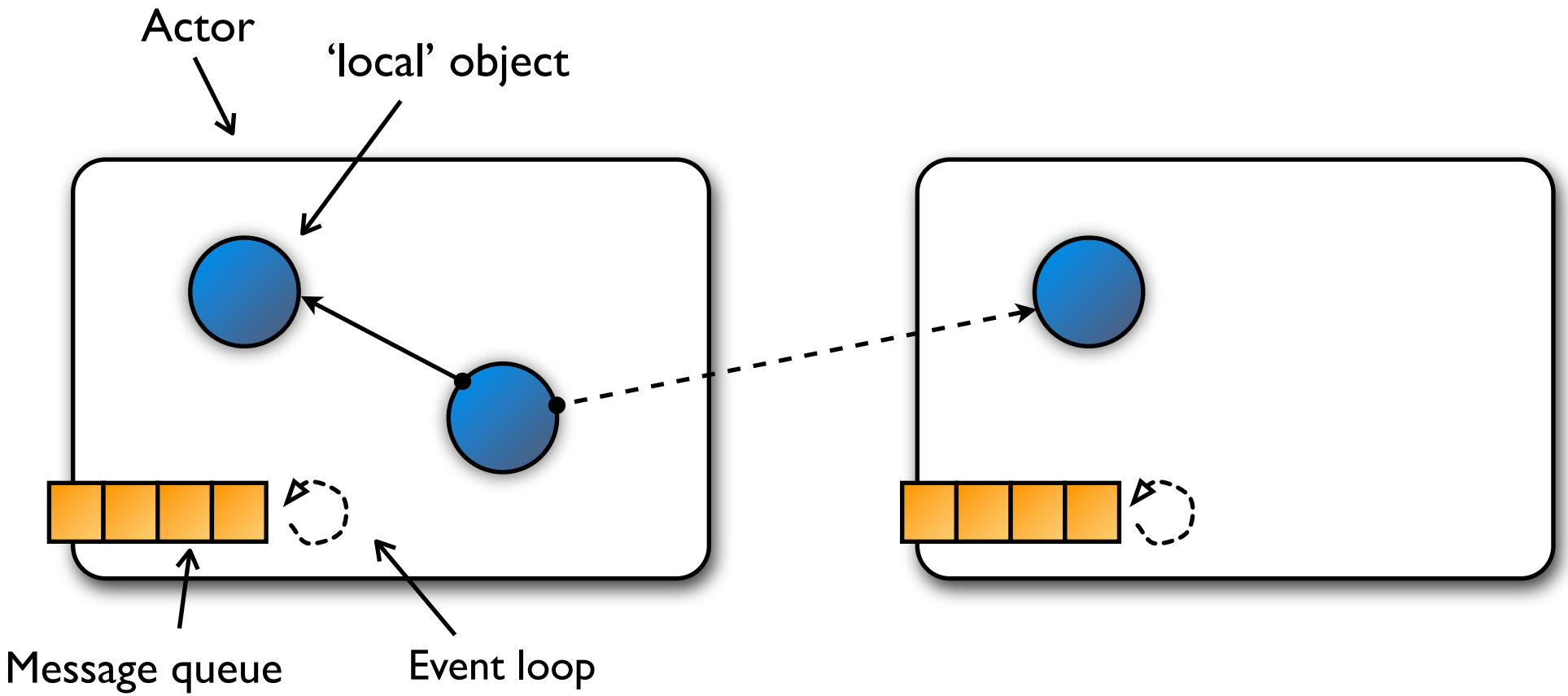
Event loop concurrency

Based on E programming language [Miller05]



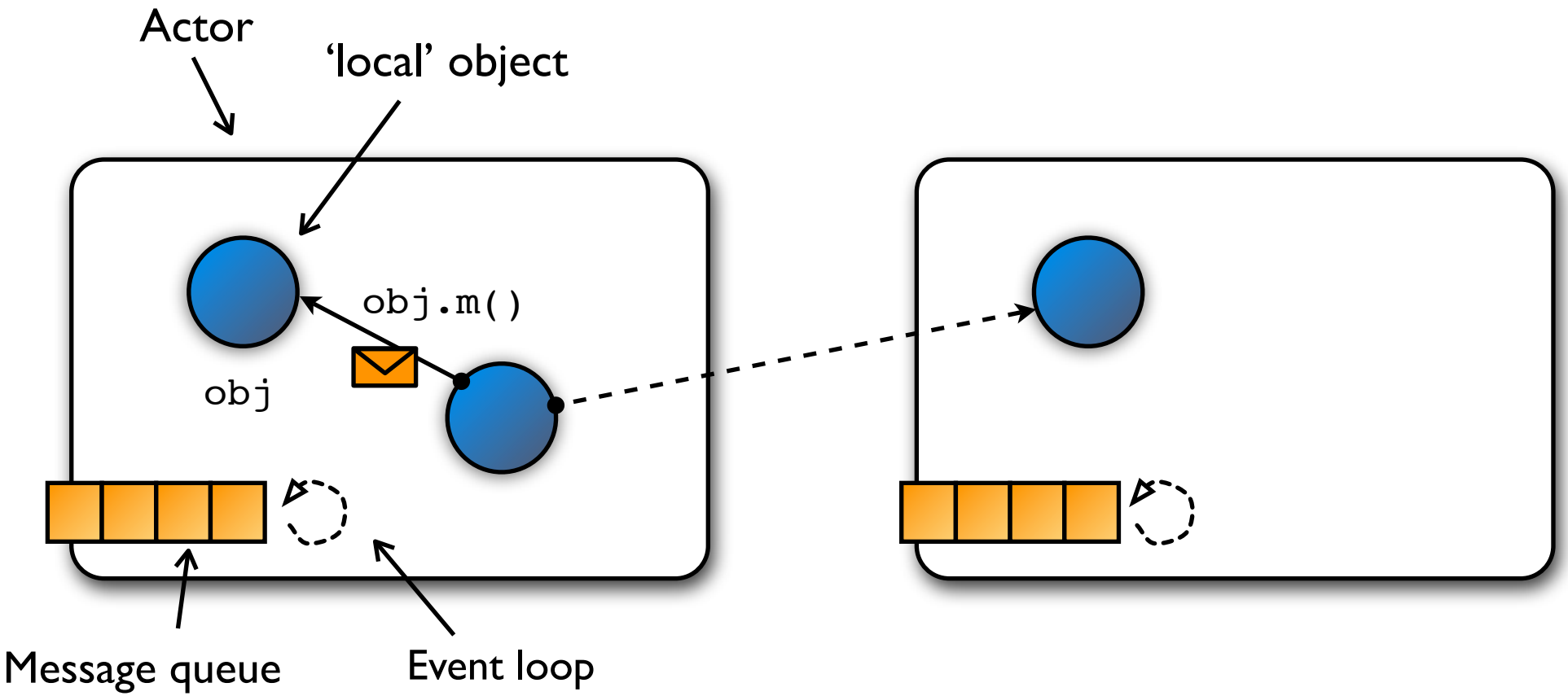
Event loop concurrency

Based on E programming language [Miller05]



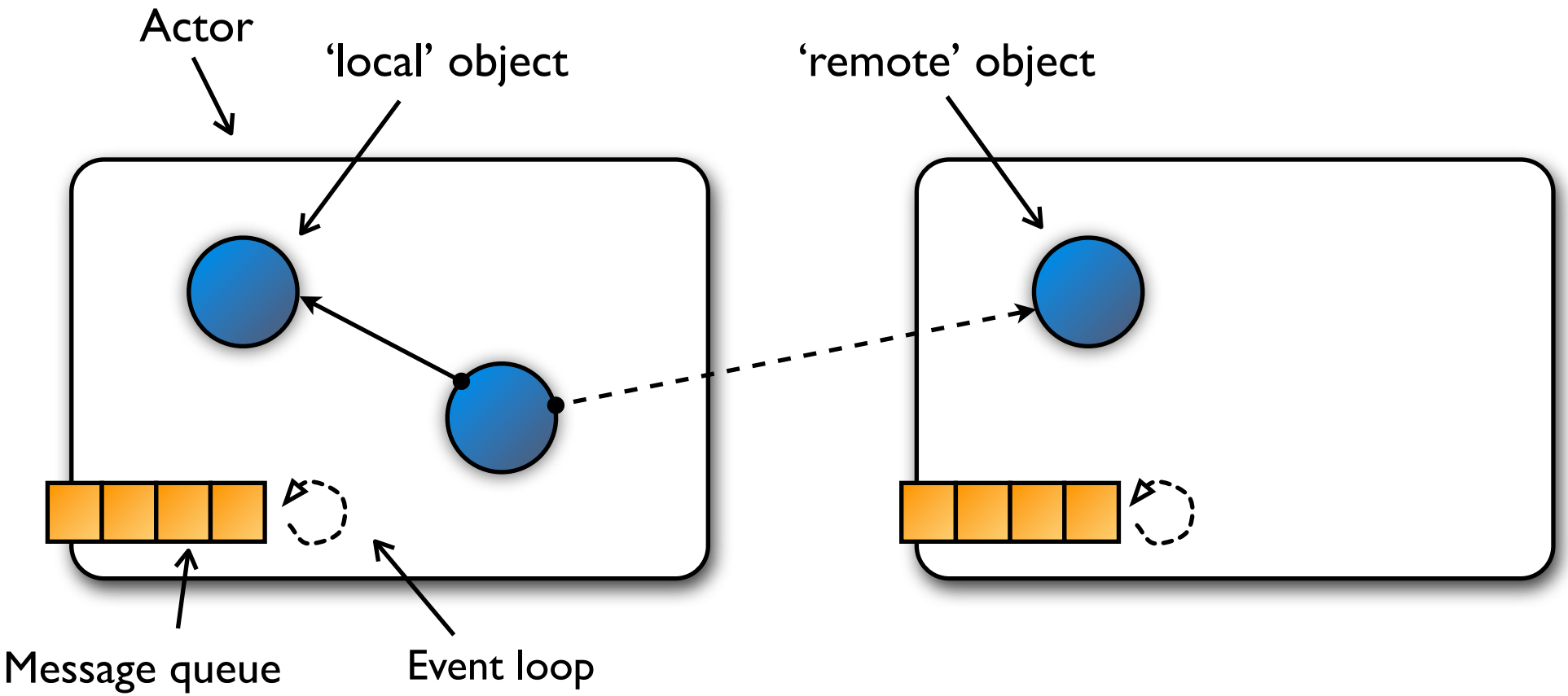
Event loop concurrency

Based on E programming language [Miller05]



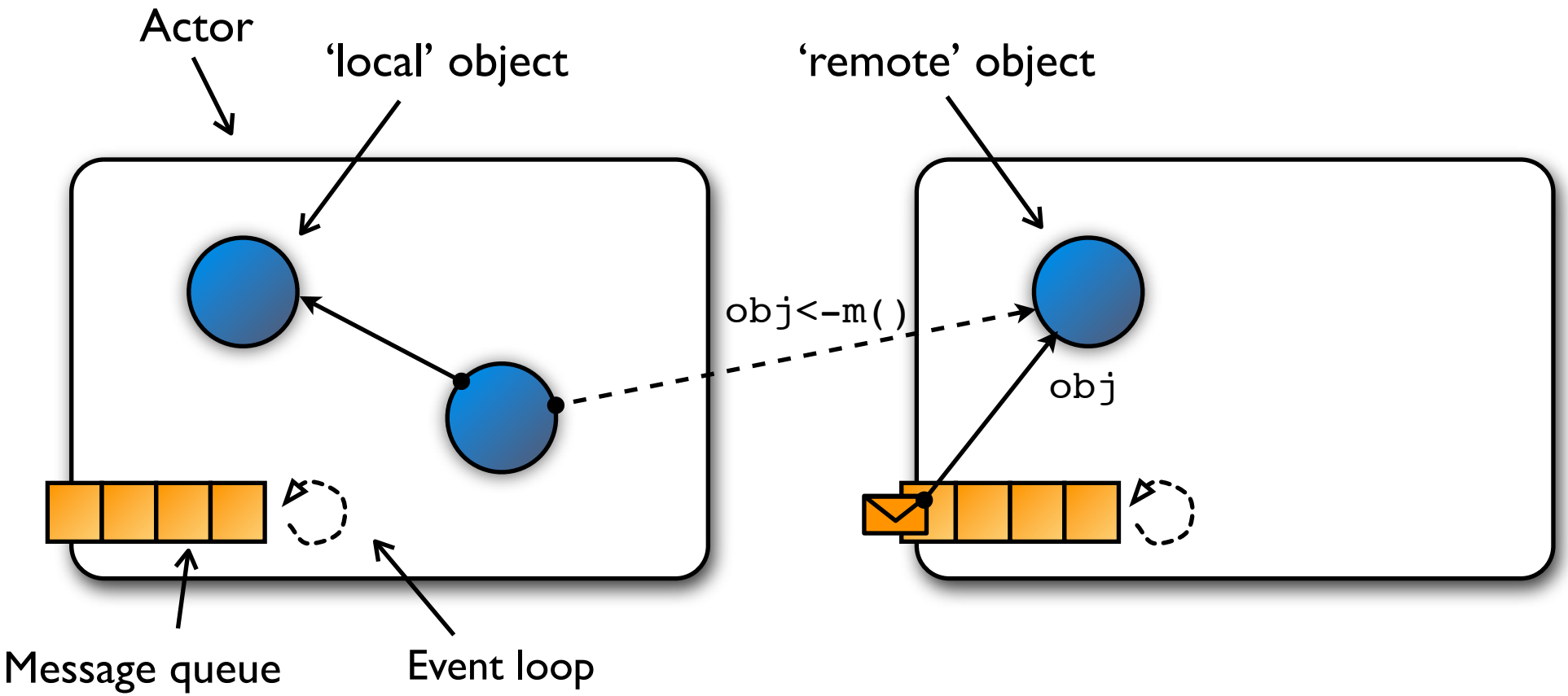
Event loop concurrency

Based on E programming language [Miller05]



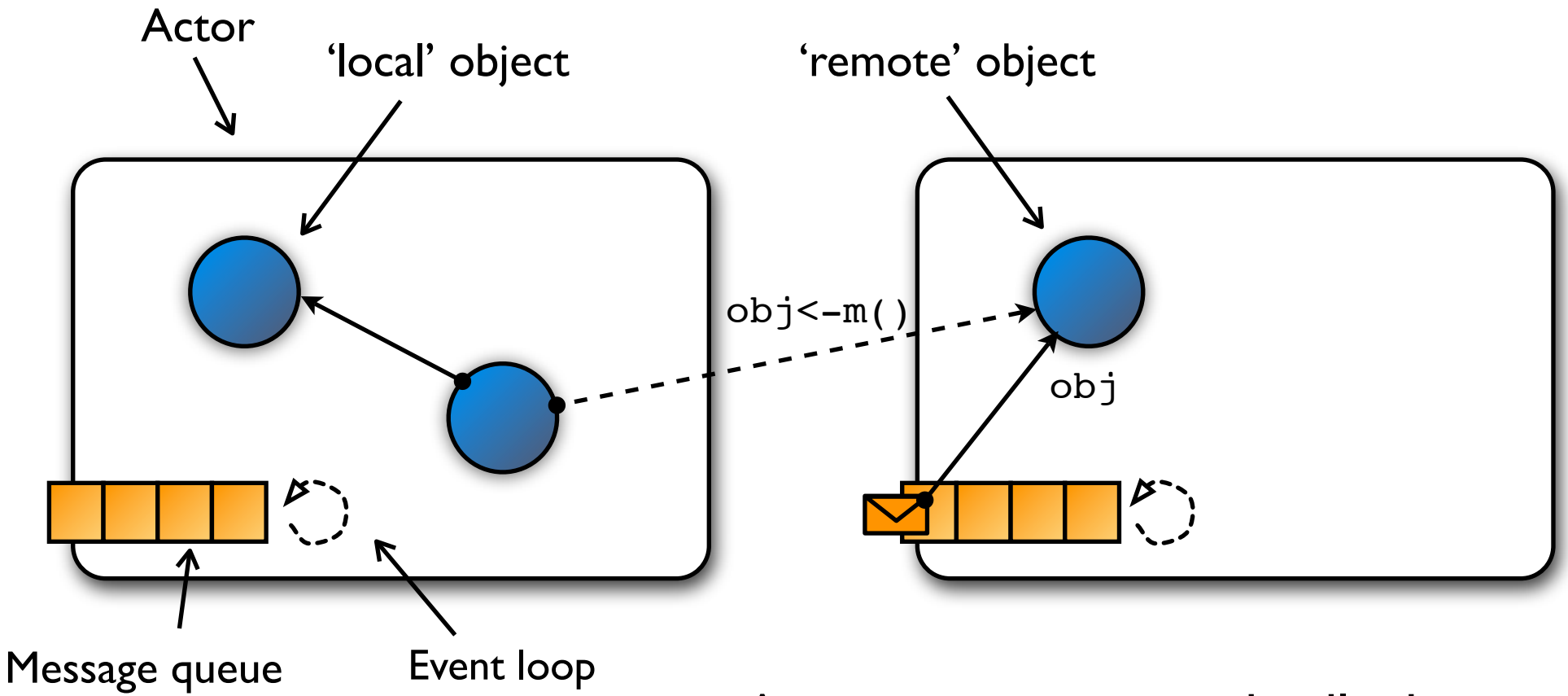
Event loop concurrency

Based on E programming language [Miller05]



Event loop concurrency

Based on E programming language [Miller05]

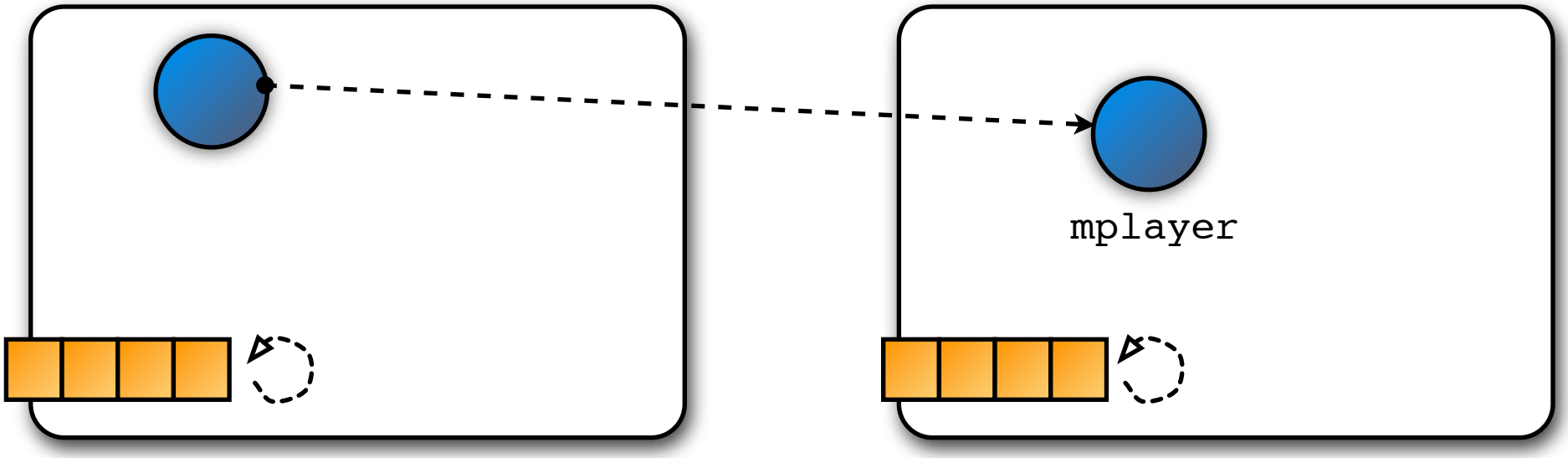


Actors cannot cause deadlock

No race conditions on objects

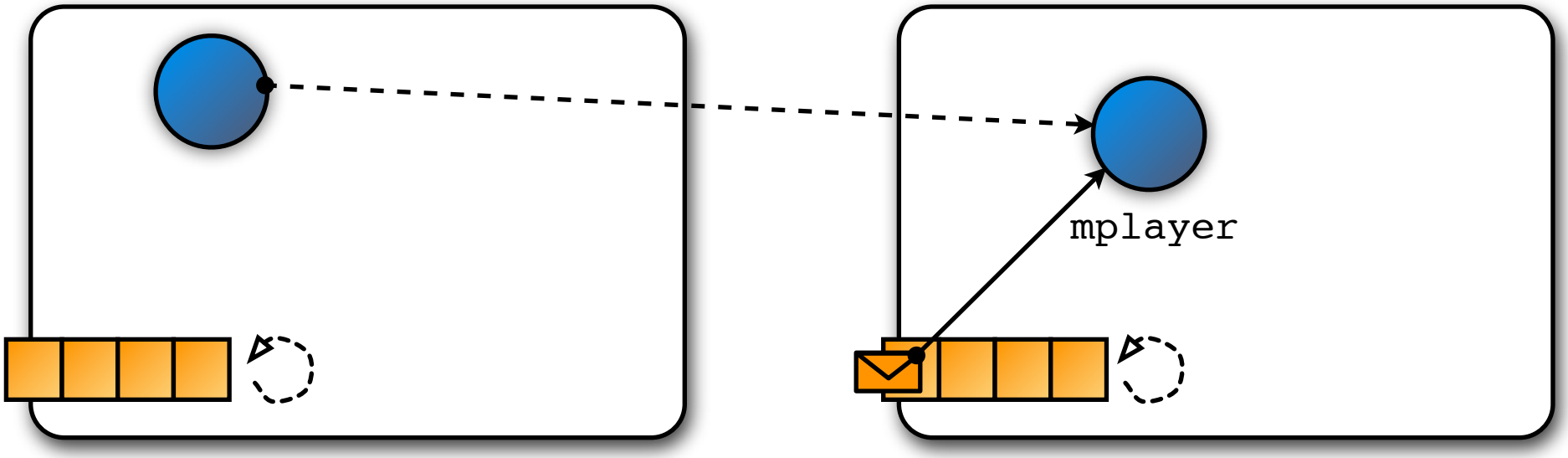
Futures

```
def future := mplayer<-numSongsInLibrary()
```



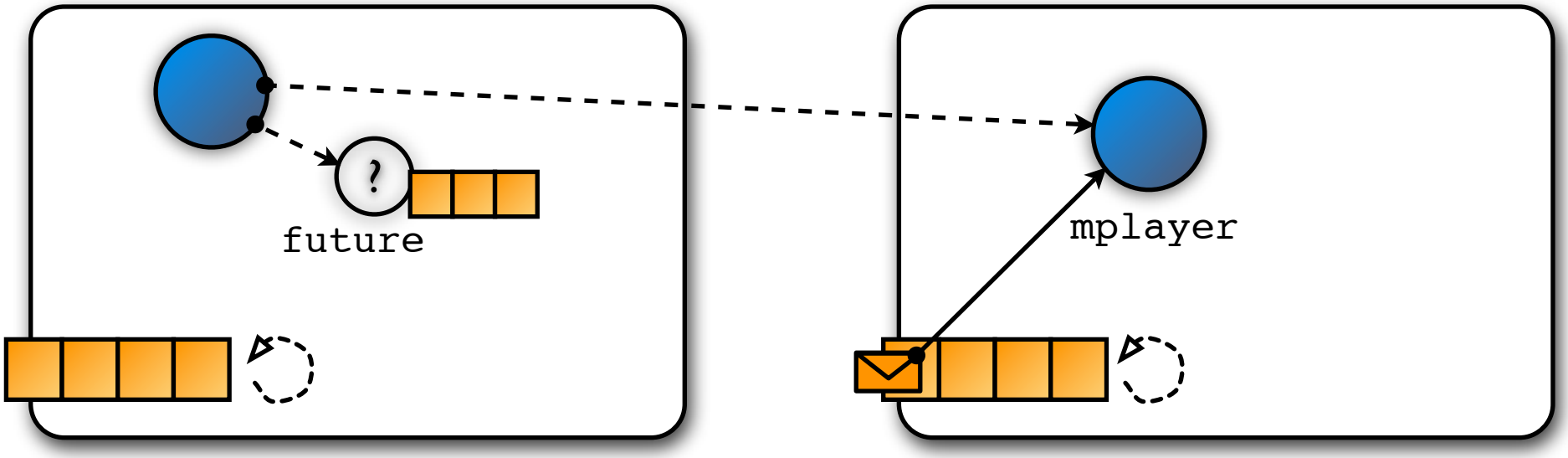
Futures

```
def future := mplayer<-numSongsInLibrary()
```



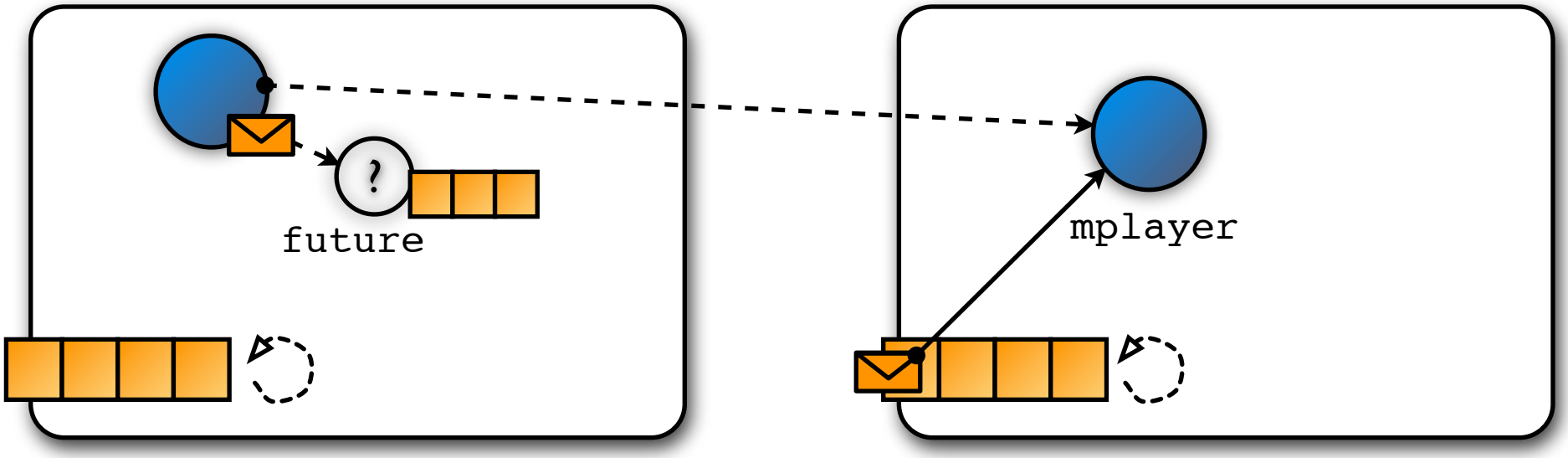
Futures

```
def future := mplayer<-numSongsInLibrary()
```



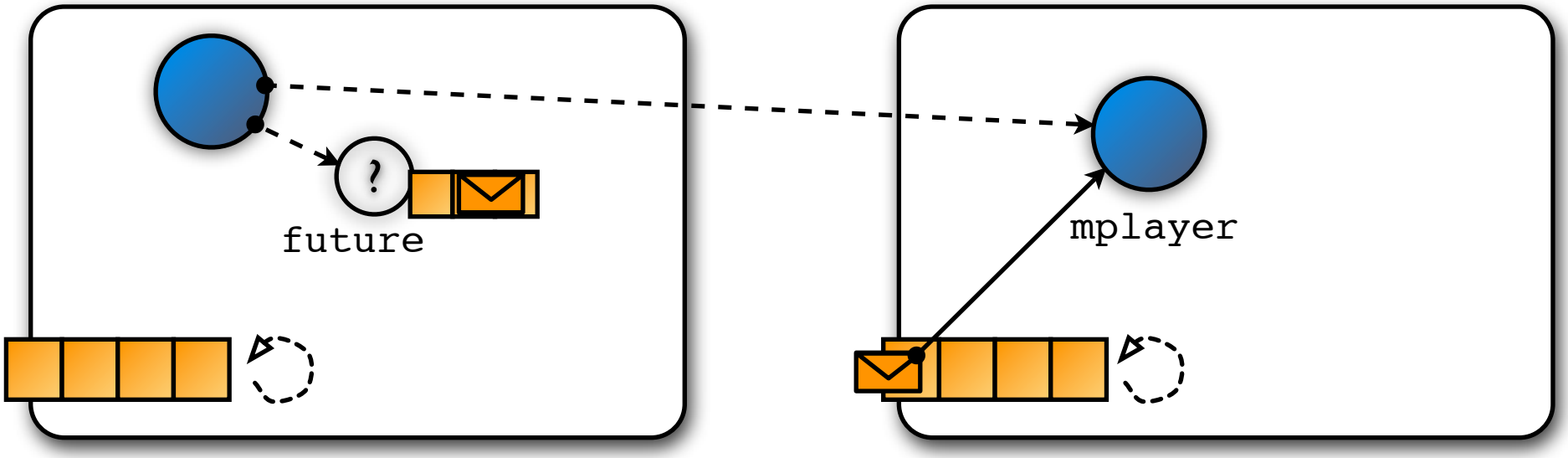
Futures

```
def future := mplayer<-numSongsInLibrary()
```



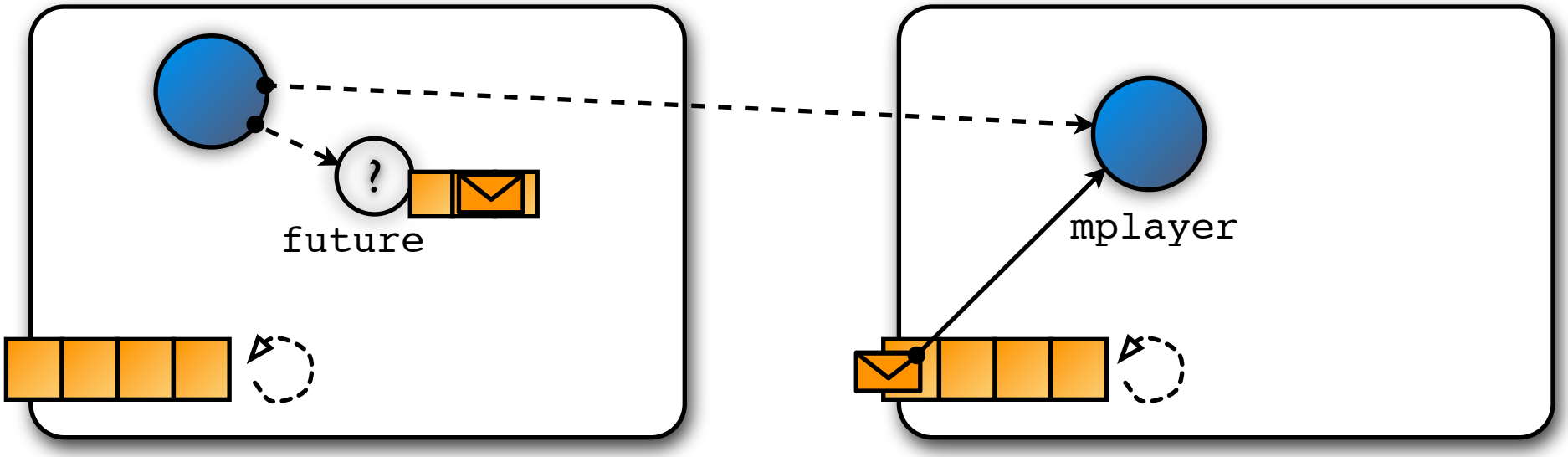
Futures

```
def future := mplayer<-numSongsInLibrary()
```



Futures

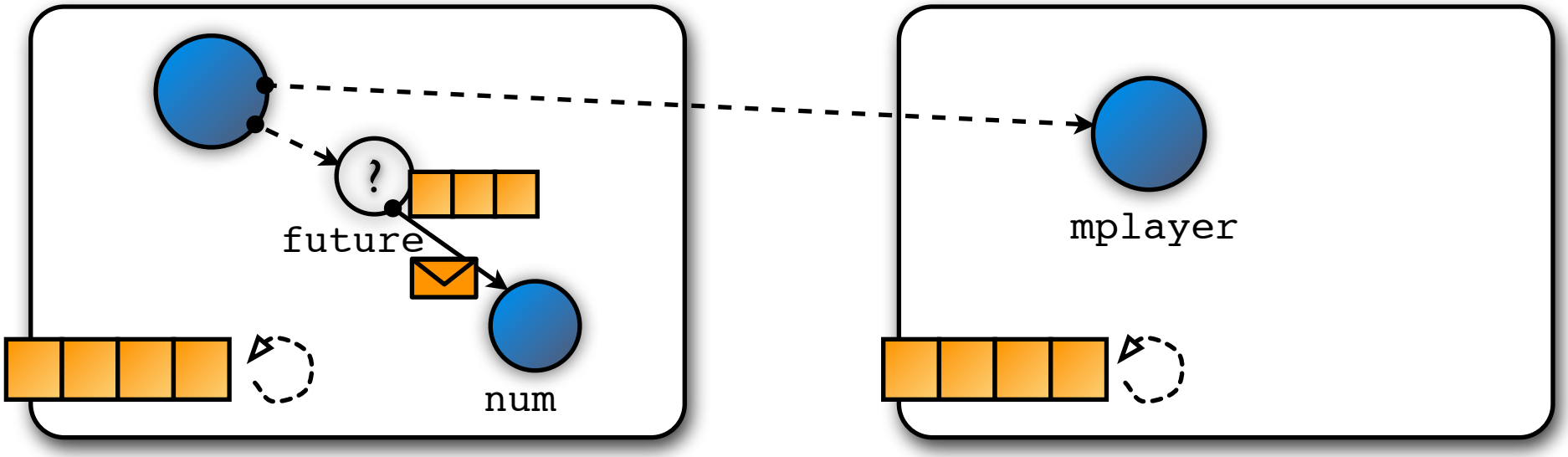
```
def future := mplayer<-numSongsInLibrary()
```




```
when: future becomes: { |num|
  system.println("user shares "+ num + " songs.")
}
```

Futures

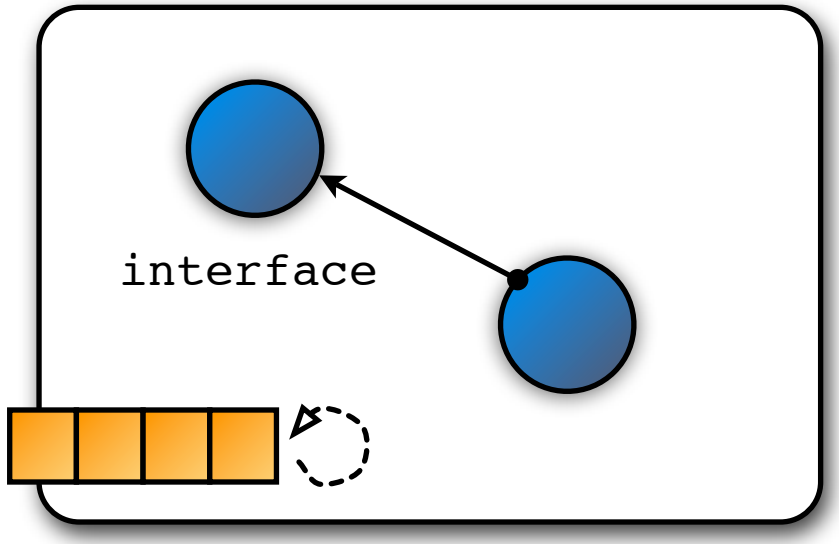
```
def future := mplayer<-numSongsInLibrary()
```




```
when: future becomes: { |num|
  system.println("user shares "+ num + " songs.")
}
```

Exporting objects

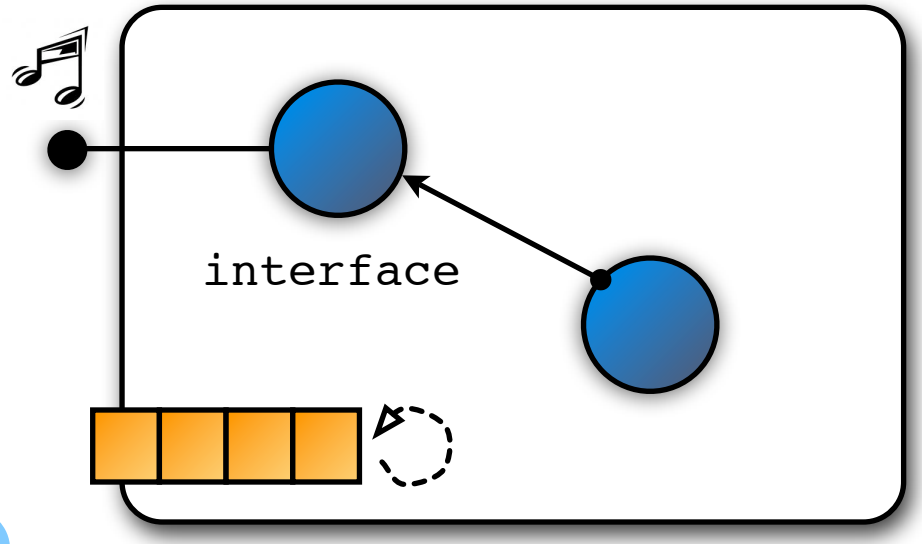
```
deftype MusicPlayer;

def interface := object: {
  def openSession() {
    ...
  }
}

export: interface as: MusicPlayer;
```



Exporting objects

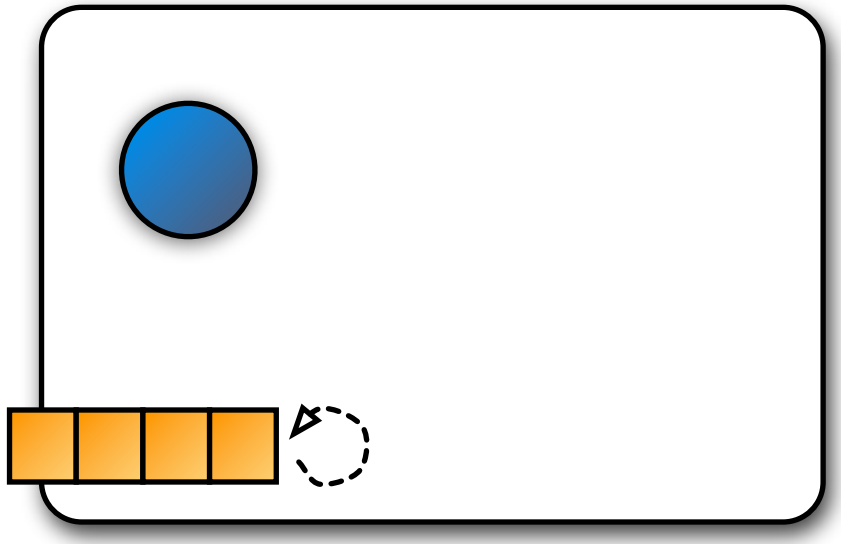
```
deftype MusicPlayer;

def interface := object: {
  def openSession() {
    ...
  }
}
```

```
export: interface as: MusicPlayer;
```



Ambient References

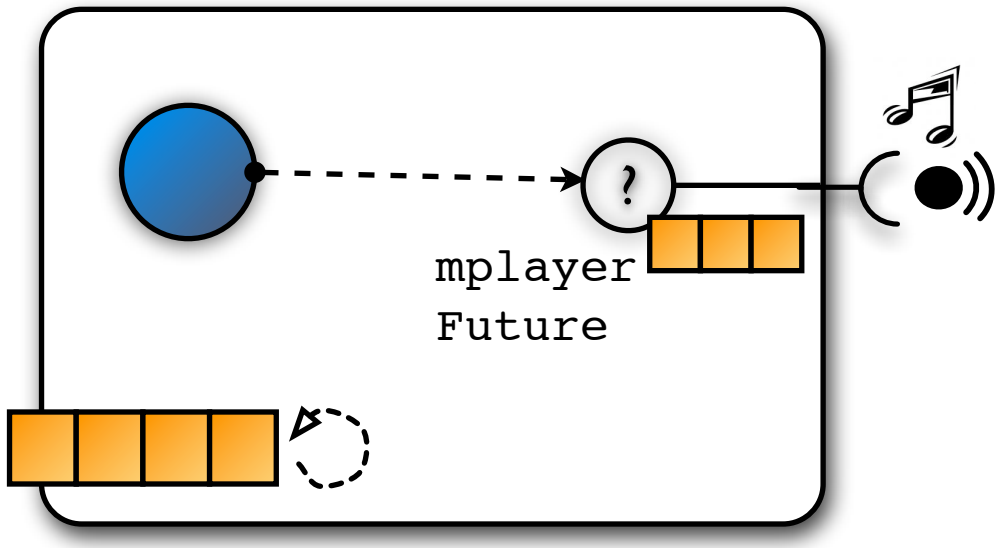
```
def mplayerFuture := ambient: MusicPlayer;
```



- Initiates service discovery
- Immediately returns future for object to be discovered

Ambient References

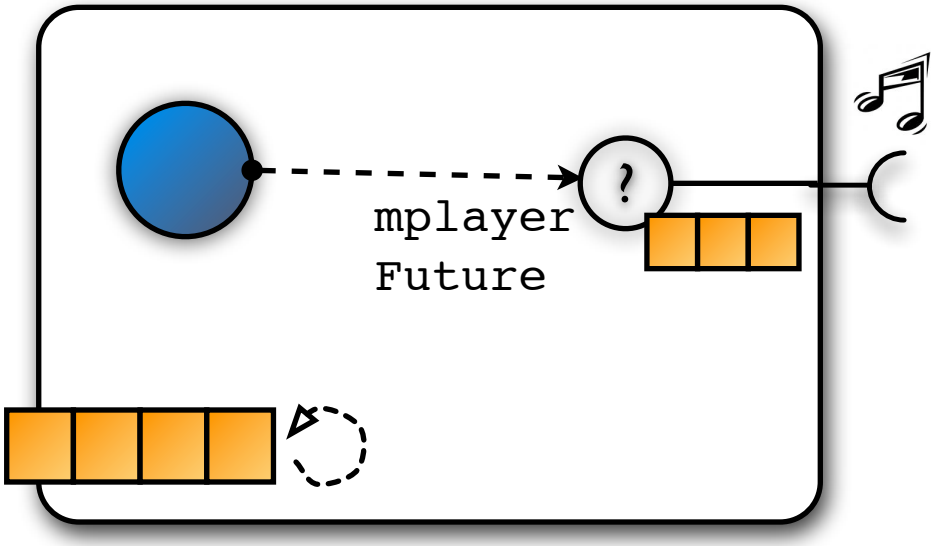
```
def mplayerFuture := ambient: MusicPlayer;
```



- Initiates service discovery
- Immediately returns future for object to be discovered

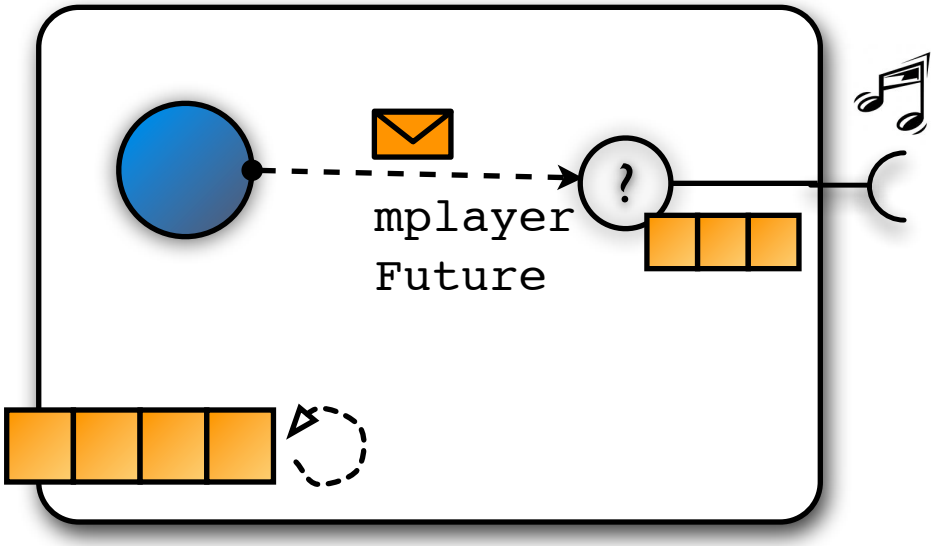
Ambient References

```
def mplayerFuture := ambient: MusicPlayer;
```



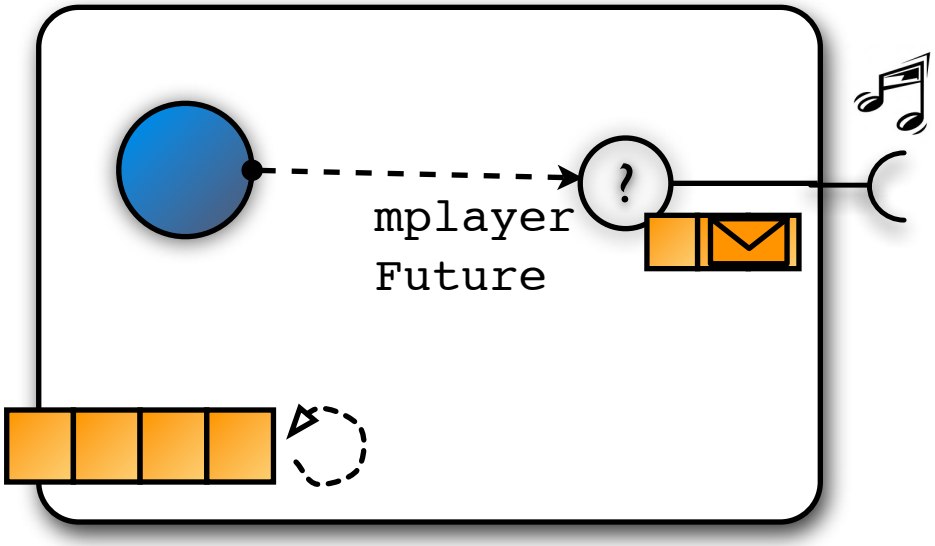
Ambient References

```
def mplayerFuture := ambient: MusicPlayer;
def sessionFuture := mplayerFuture<-openSession();
```



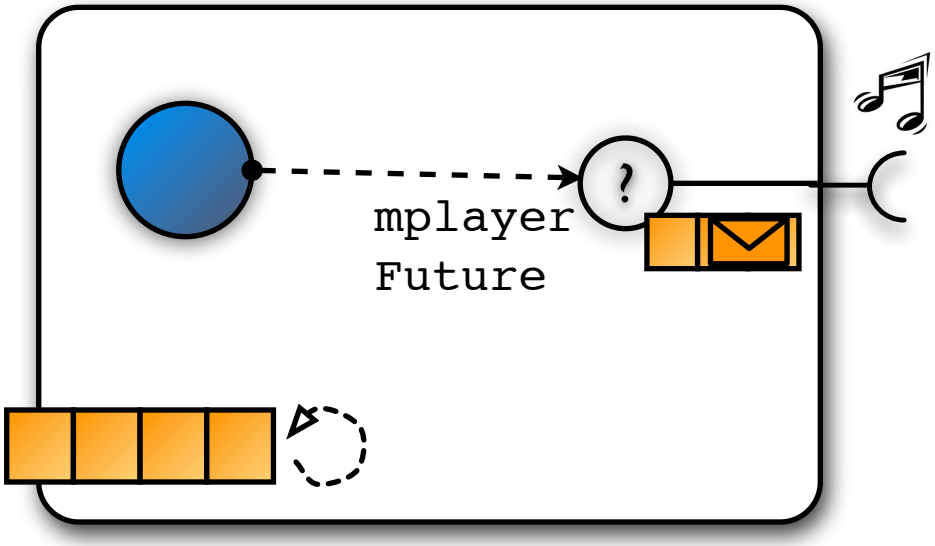
Ambient References

```
def mplayerFuture := ambient: MusicPlayer;
def sessionFuture := mplayerFuture<-openSession();
```



Ambient References

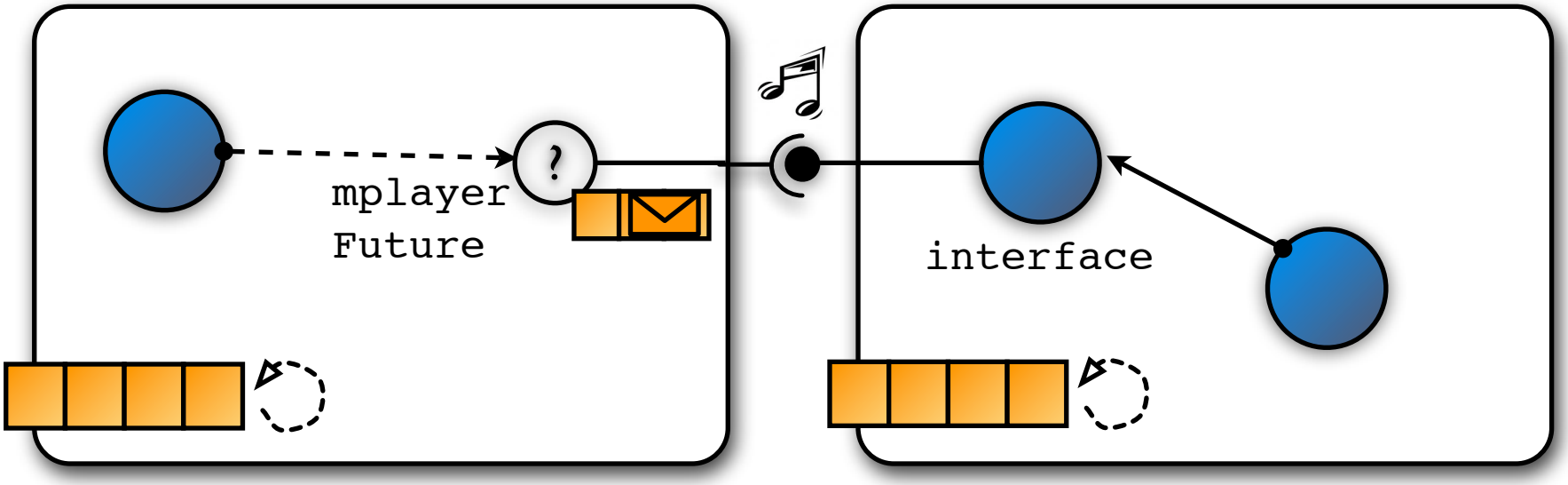
```
def mplayerFuture := ambient: MusicPlayer;
def sessionFuture := mplayerFuture<-openSession();
```




```
when: mplayerFuture becomes: { |ambientRef|
  println("music player found")
}
```

Ambient References

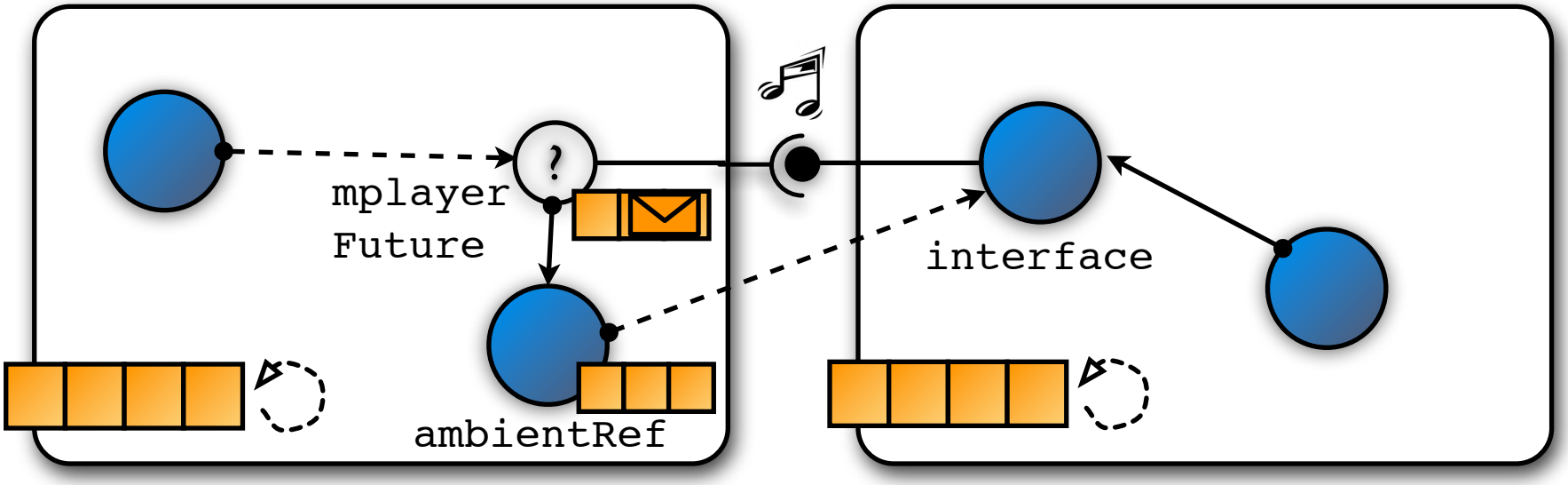
```
def mplayerFuture := ambient: MusicPlayer;
def sessionFuture := mplayerFuture<-openSession();
```




```
when: mplayerFuture becomes: { |ambientRef|
  println("music player found")
}
```

Ambient References

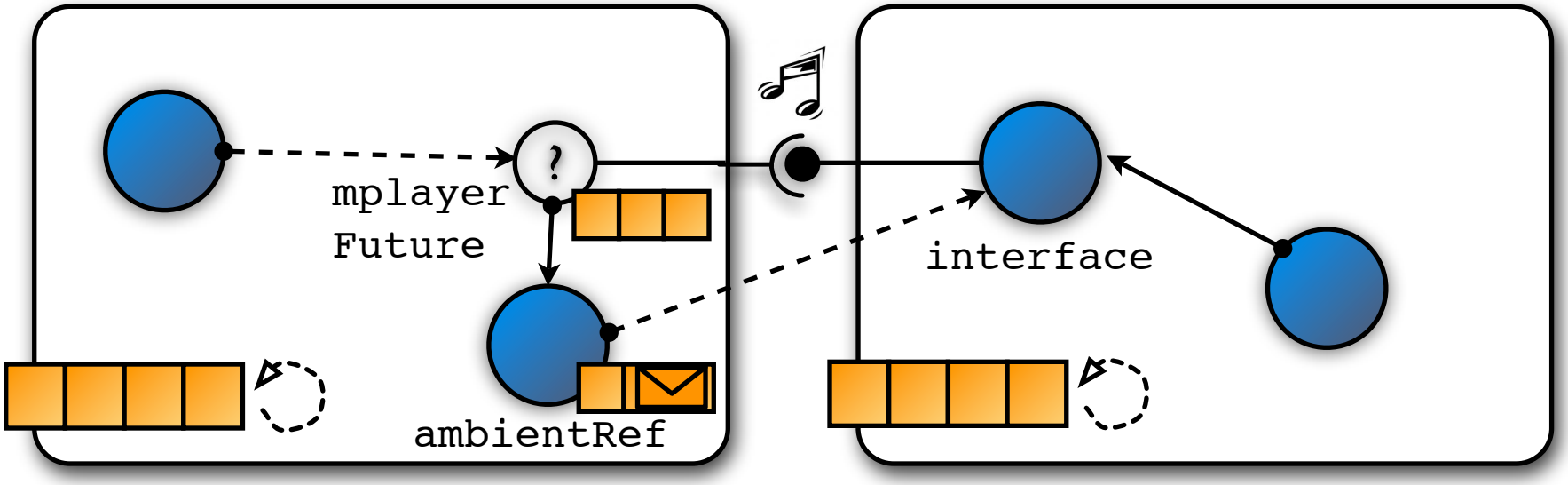
```
def mplayerFuture := ambient: MusicPlayer;
def sessionFuture := mplayerFuture<-openSession();
```




```
when: mplayerFuture becomes: { |ambientRef|
  println("music player found")
}
```

Ambient References

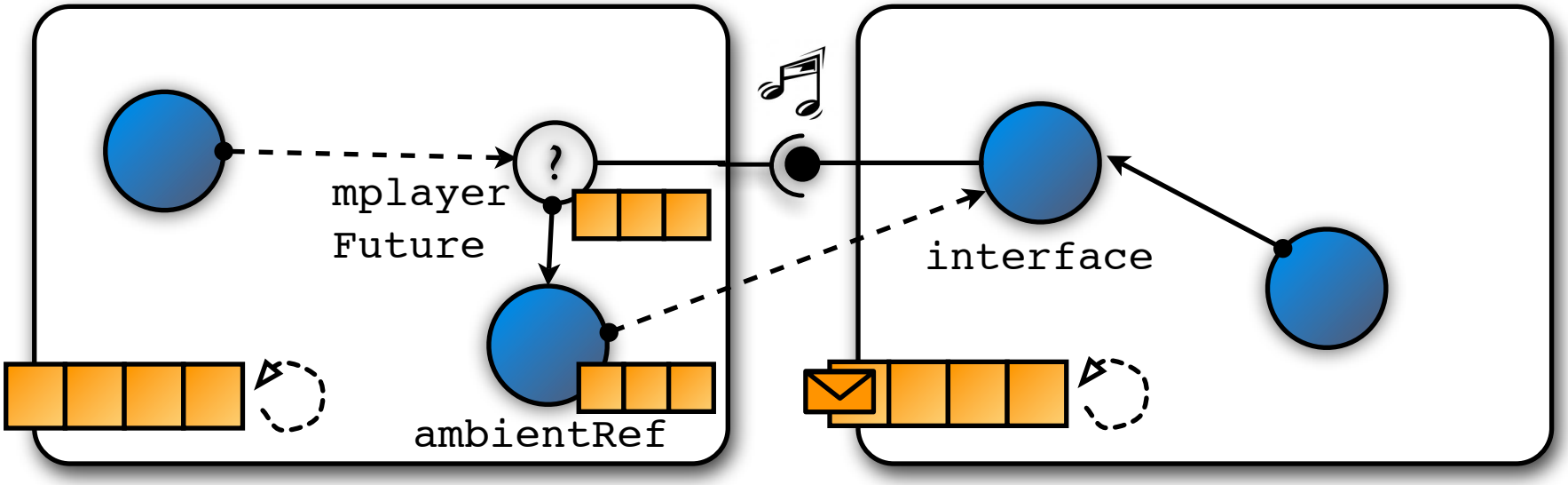
```
def mplayerFuture := ambient: MusicPlayer;
def sessionFuture := mplayerFuture<-openSession();
```




```
when: mplayerFuture becomes: { |ambientRef|
  println("music player found")
}
```

Ambient References

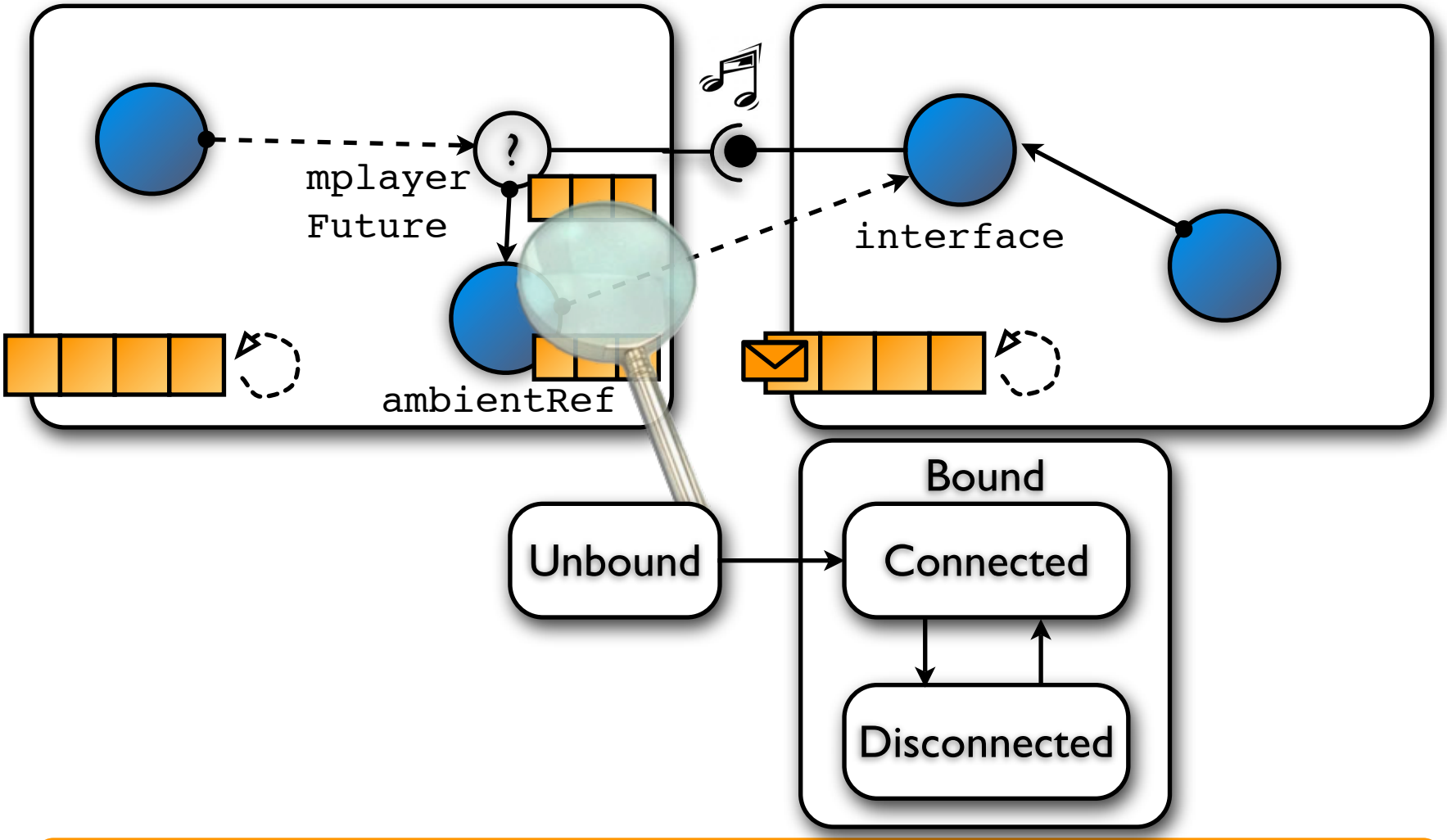
```
def mplayerFuture := ambient: MusicPlayer;
def sessionFuture := mplayerFuture<-openSession();
```




```
when: mplayerFuture becomes: { |ambientRef|
  println("music player found")
}
```

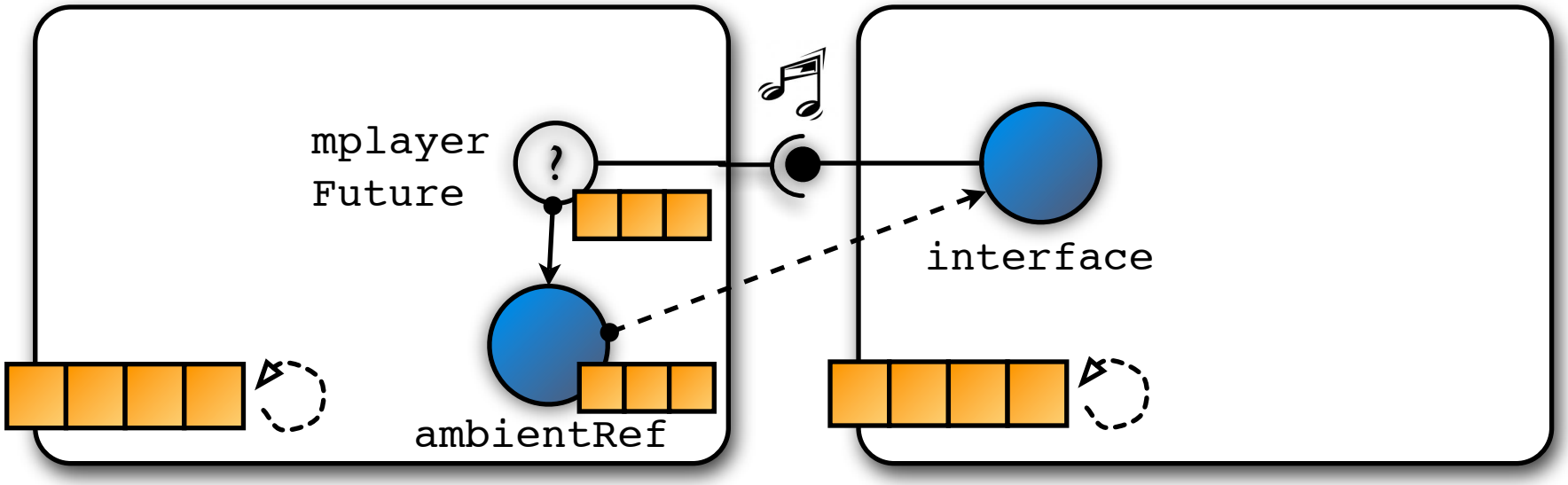
Ambient References

```
def mplayerFuture := ambient: MusicPlayer;
def sessionFuture := mplayerFuture<-openSession();
```



Failure handling

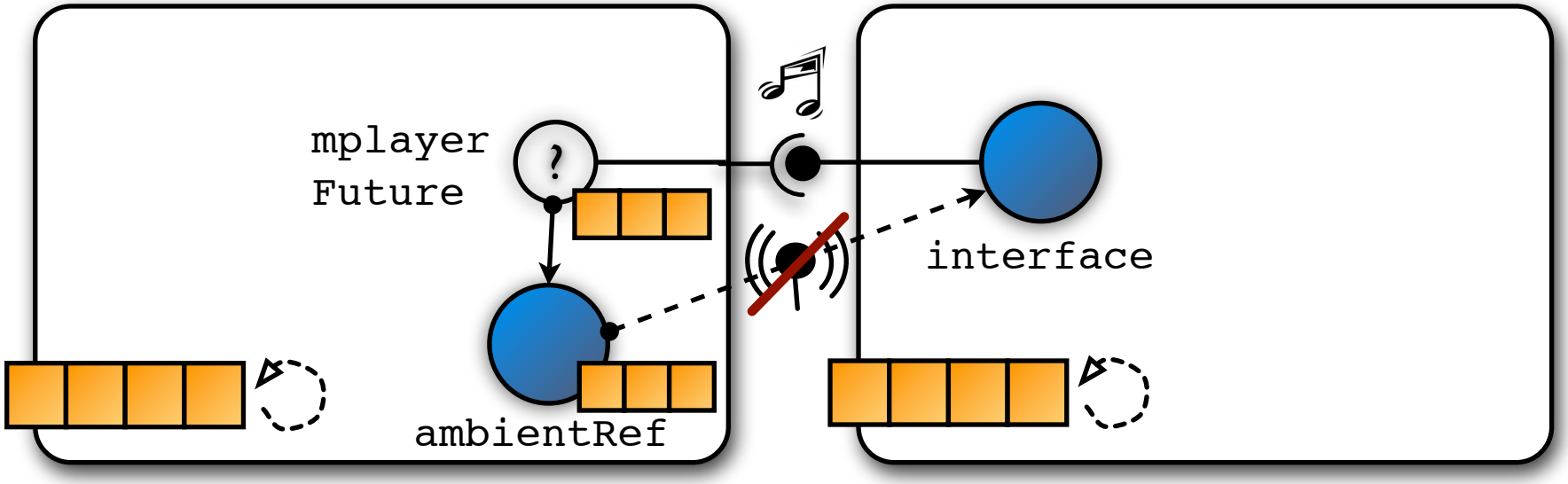
- Event handlers on ambient references:




```
when: ambientRef disconnects: {
  println("music player disconnected")
}

when: ambientRef reconnects: {
  println("music player reconnected")
}
```

Failure handling

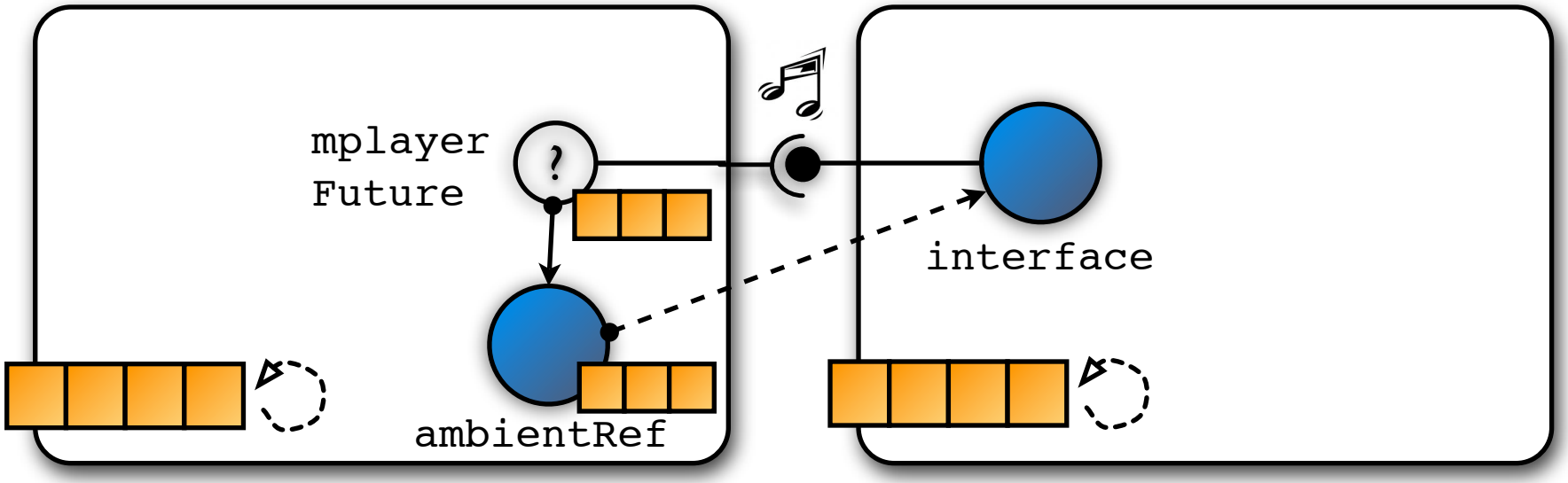
- Event handlers on ambient references:




```
when: ambientRef disconnects: {
  println("music player disconnected")
}

when: ambientRef reconnects: {
  println("music player reconnected")
}
```

Failure handling

- Event handlers on ambient references:




```
when: ambientRef disconnects: {
  println("music player disconnected")
}

when: ambientRef reconnects: {
  println("music player reconnected")
}
```

Failure Handling

- Leased references which eventually expire
- Futures + timeouts

```
when: o<-m()@Timeout(minutes(10)) becomes: { |v|
  // process return value
} catch: TimeoutException using: { |e|
  // deal with timeout
}
```

Variations

- Past experiments:
  - ambient 'omni'-references: broadcasting
  - content-based discovery
- Future experiments:
  - Customisable message delivery guarantees
  - First-class proximity: restrict spatial scope of ambient references

Lessons Learned

- AmbientTalk = OO + Events:
  - Block closures as nested event handlers
  - Futures: non-blocking synchronisation
  - Buffered asynchronous messaging abstracts over intermittent connectivity
  - Ambient references: space-decoupled remote object references

Conclusion

- MANETs: loosely coupled collaboration
- AmbientTalk: actor-based OO language
- Deal with universal MANET characteristics at the language level:


Intermittent Connectivity: time & sync-decoupled references


Scarce Infrastructure: space-decoupled references

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