Object-oriented Coordination in Mobile Ad Hoc Networks

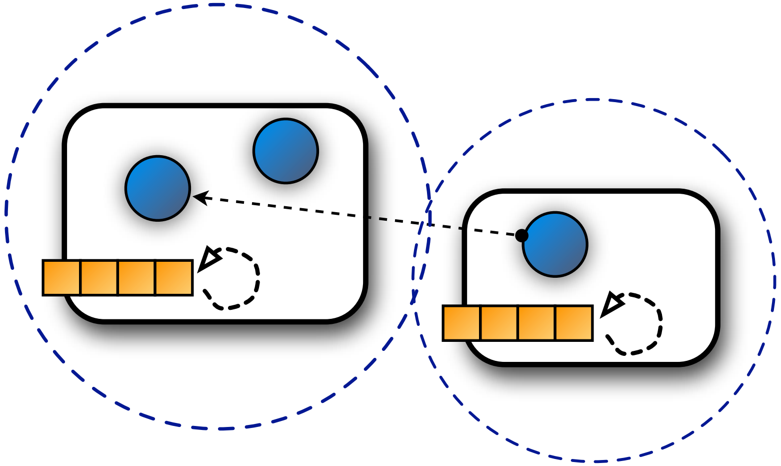
Tom Van Cutsem Jessie Dedecker Wolfgang De Meuter

Programming Technology Lab
Vrije Universiteit Brussel
Brussels, Belgium

June 8th 2007 - International Conference on Coordination Models and Languages

Context

Object-oriented programming languages



Software

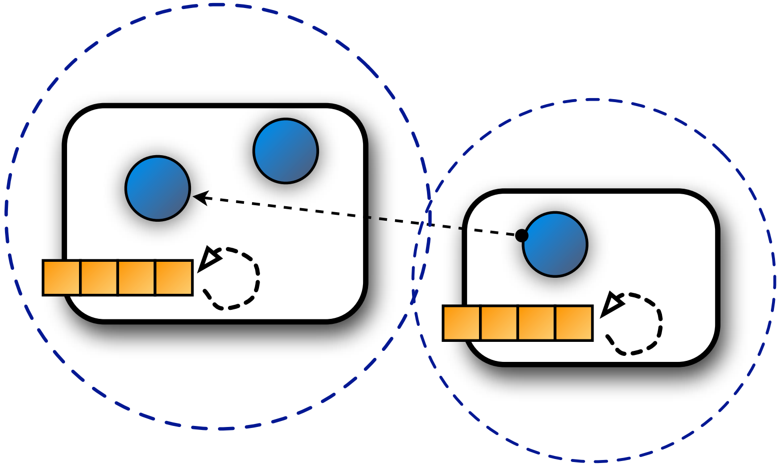
Hardware

Pervasive Computing
(Mobile Networks)

Context

Object-oriented programming languages

AMBIENTTALK

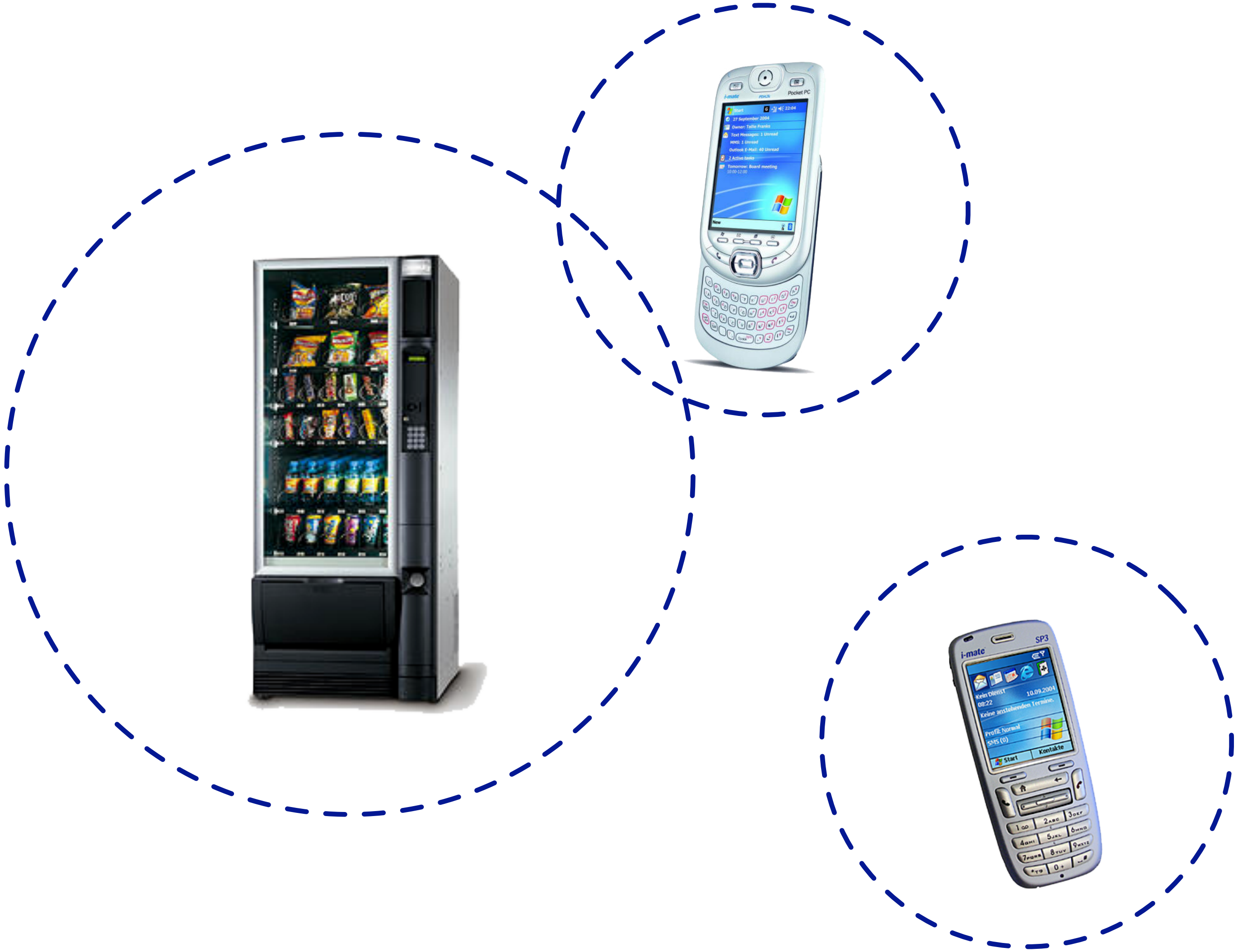


Software

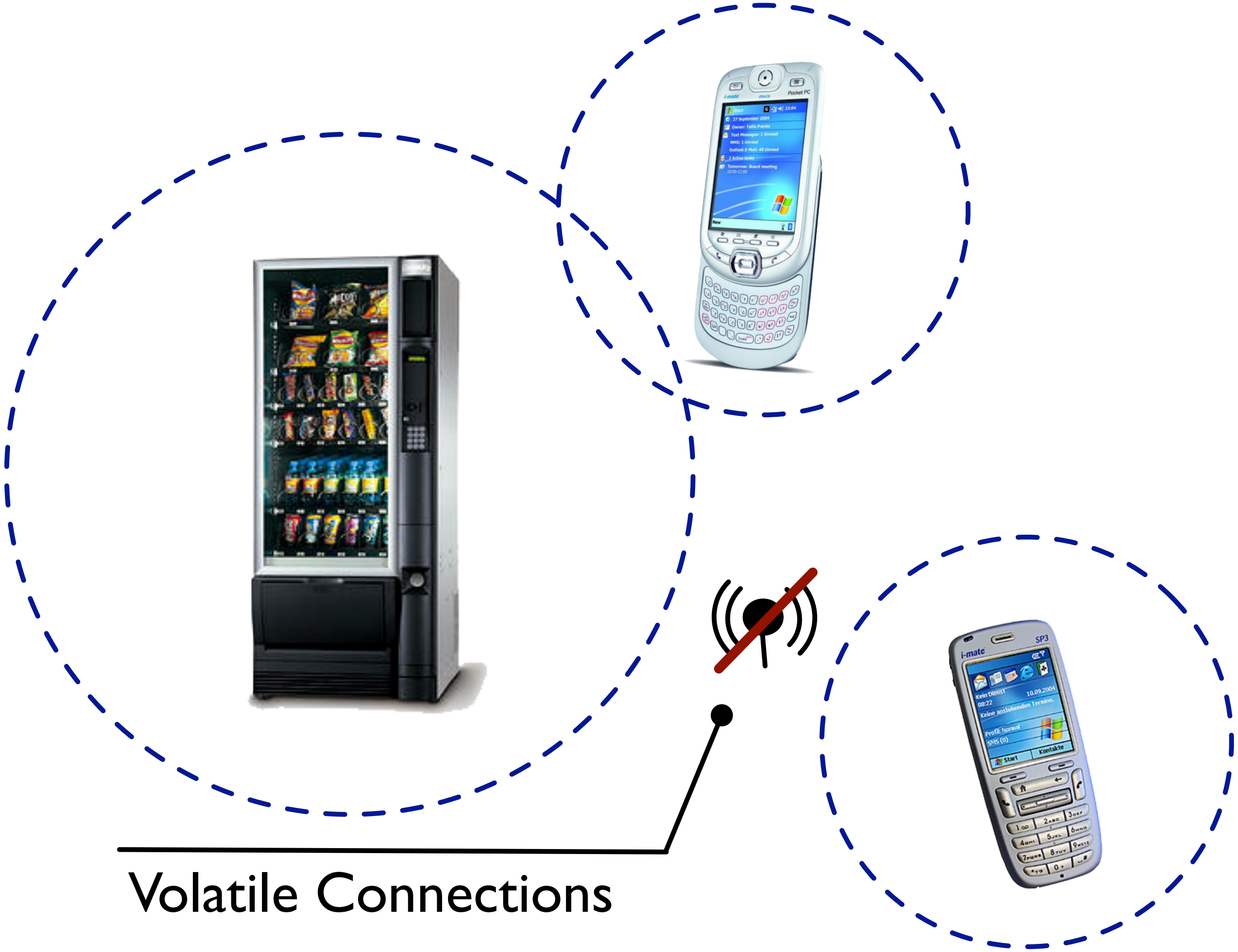
Hardware

Pervasive Computing
(Mobile Networks)

Mobile Ad hoc Networks

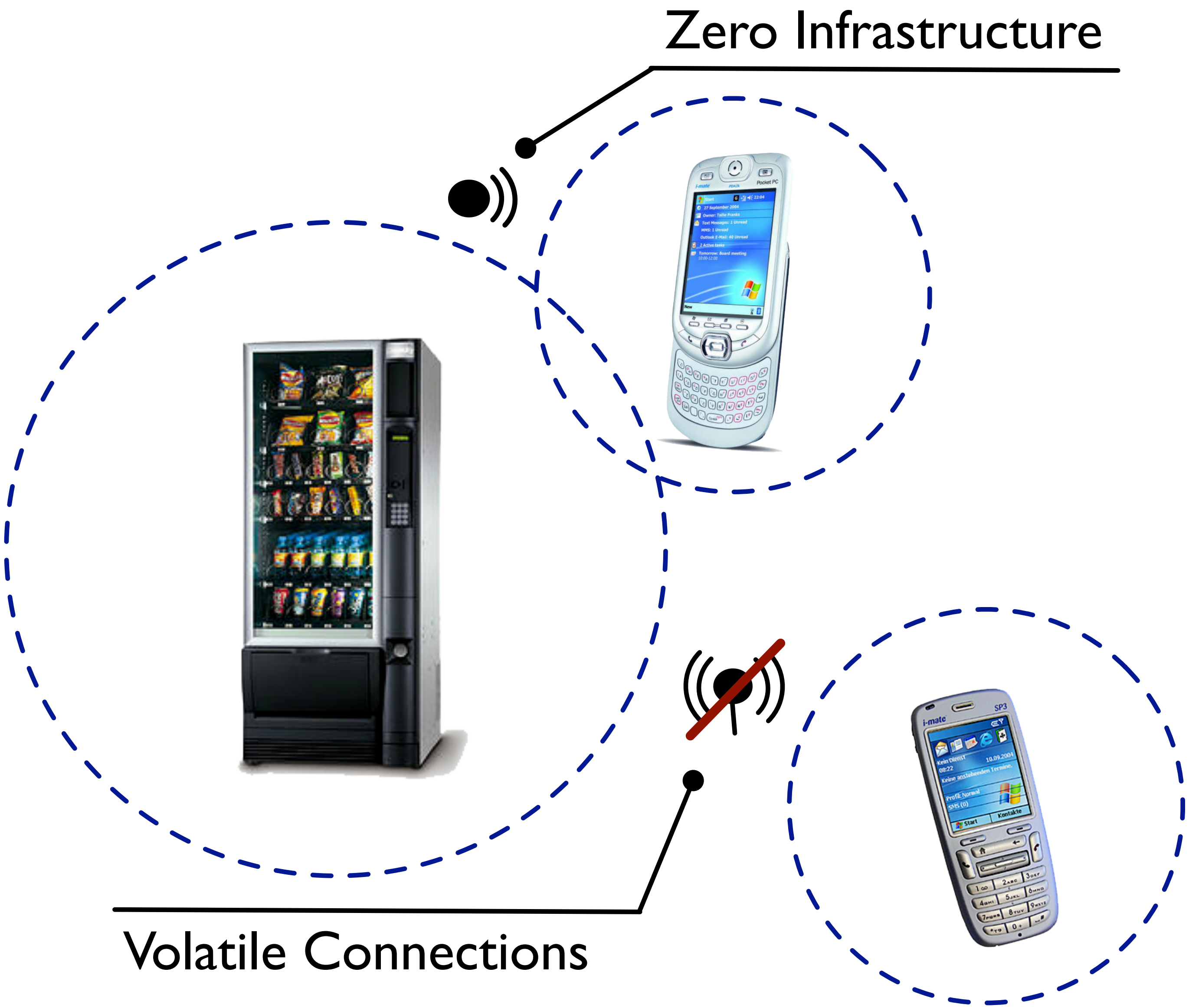


Mobile Ad hoc Networks



Volatile Connections

Mobile Ad hoc Networks



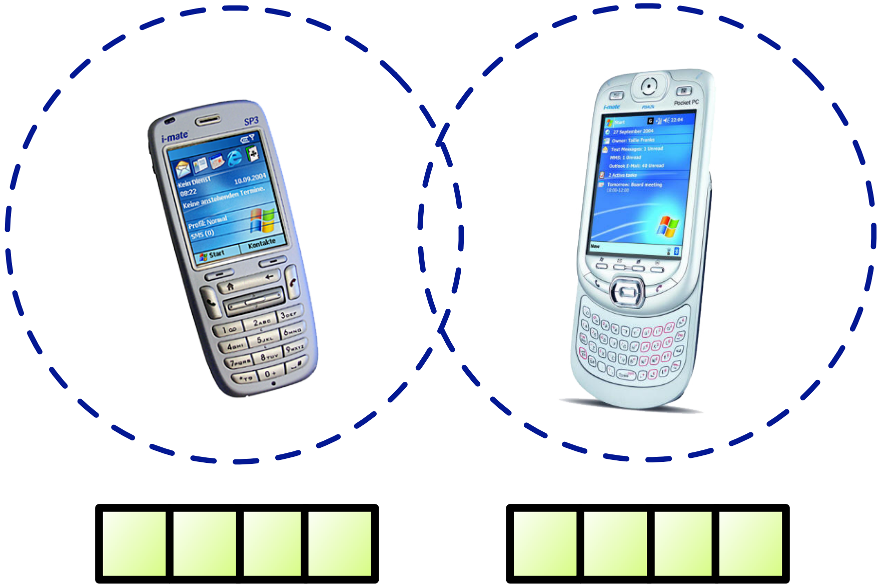
Zero Infrastructure

Volatile Connections

Loose Coupling

Decoupling communication in *Time & Synchronisation*

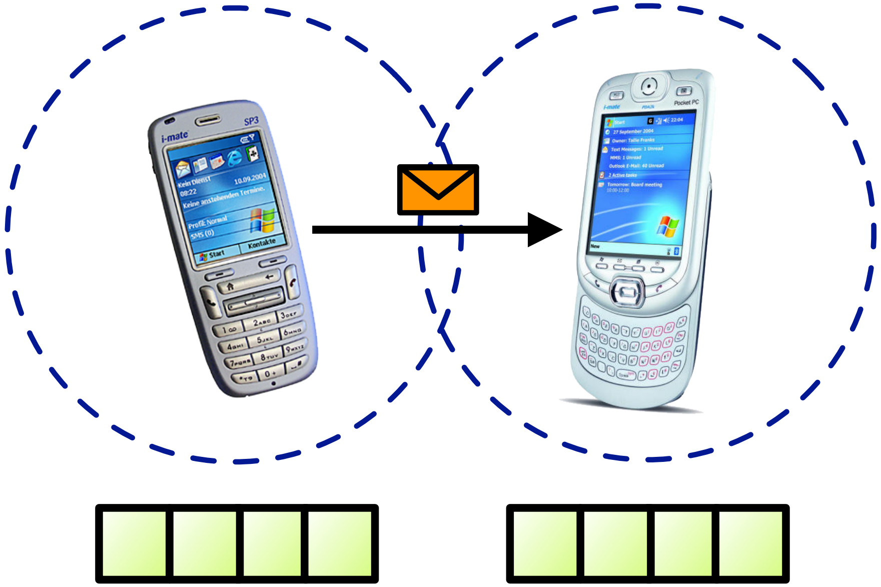
reduces impact of volatile connections



Loose Coupling

Decoupling communication in *Time & Synchronisation*

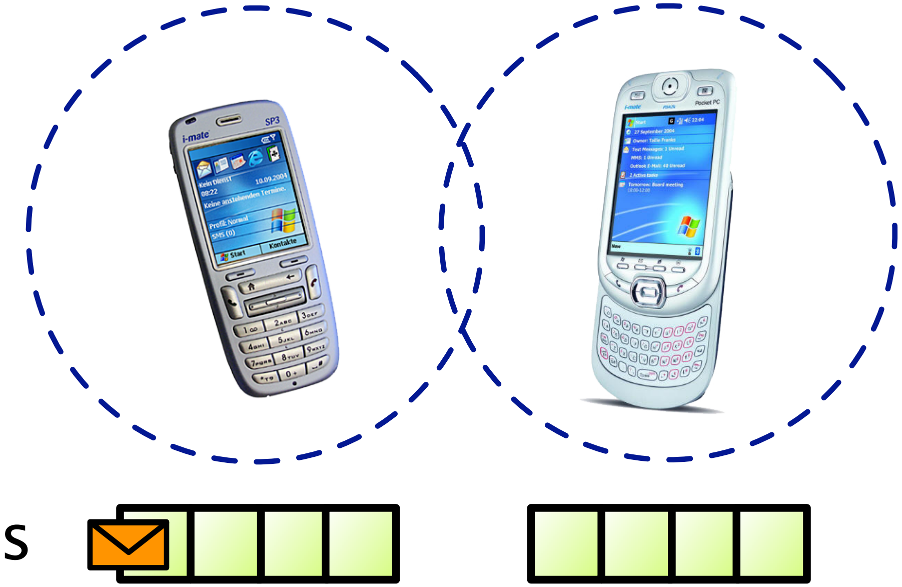
reduces impact of volatile connections



Loose Coupling

Decoupling communication in *Time & Synchronisation*

reduces impact of volatile connections

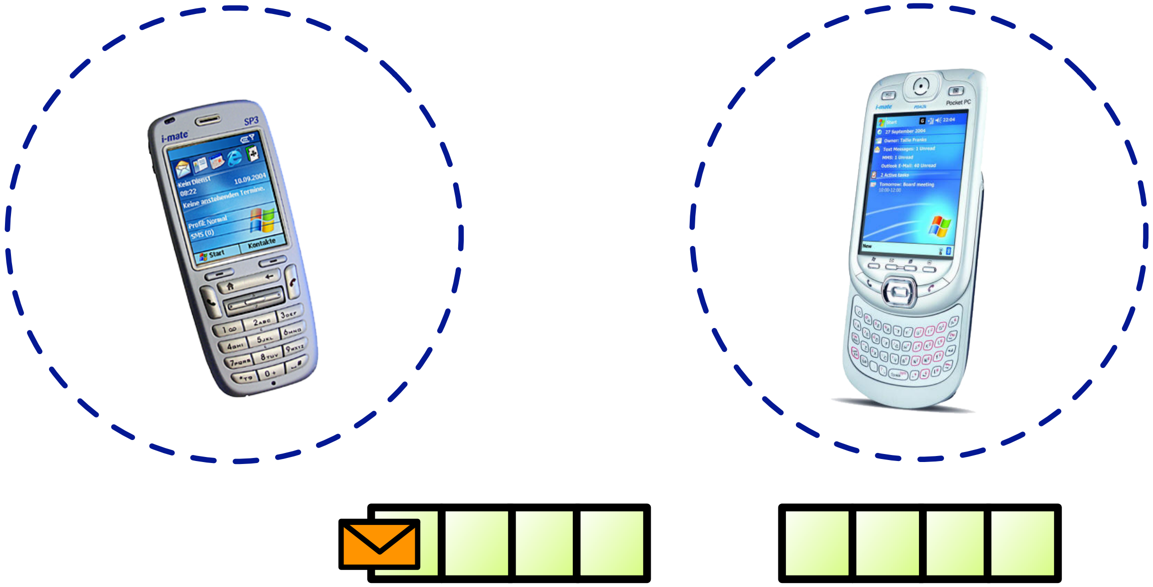


asynchronous
send

Loose Coupling

Decoupling communication in *Time & Synchronisation*

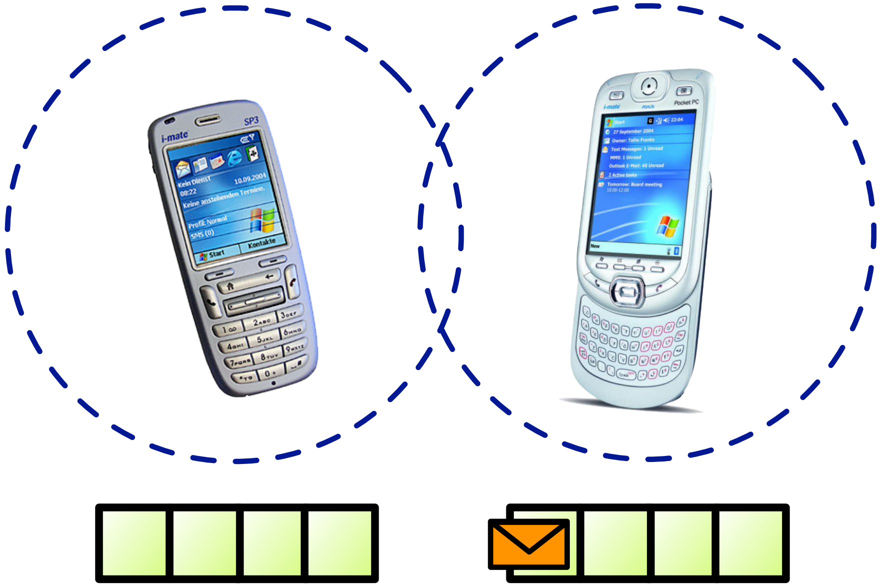
reduces impact of volatile connections



Loose Coupling

Decoupling communication in *Time & Synchronisation*

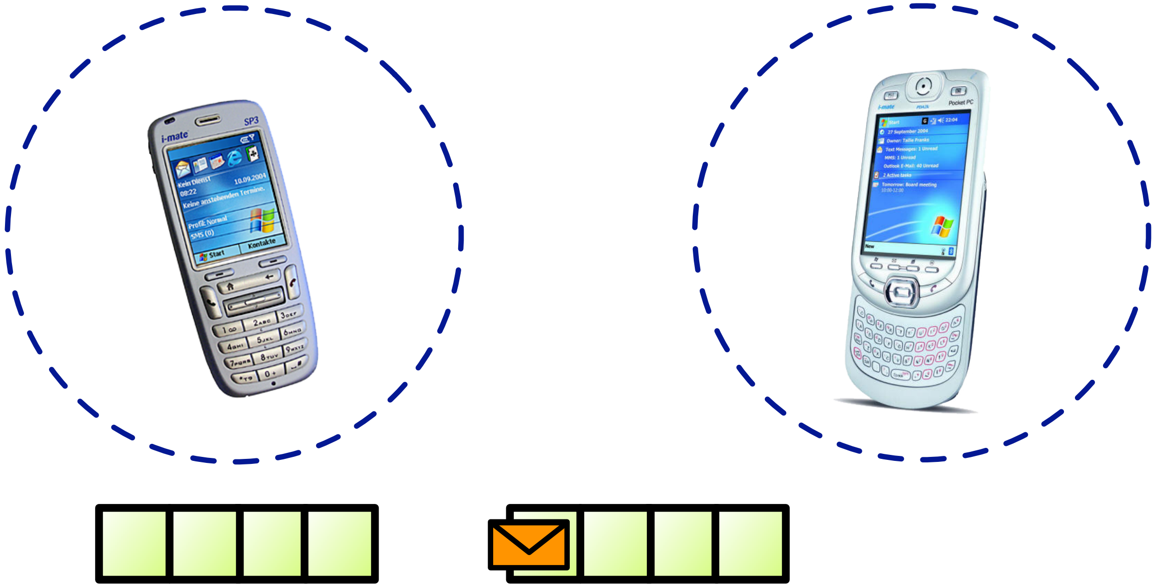
reduces impact of volatile connections



Loose Coupling

Decoupling communication in *Time & Synchronisation*

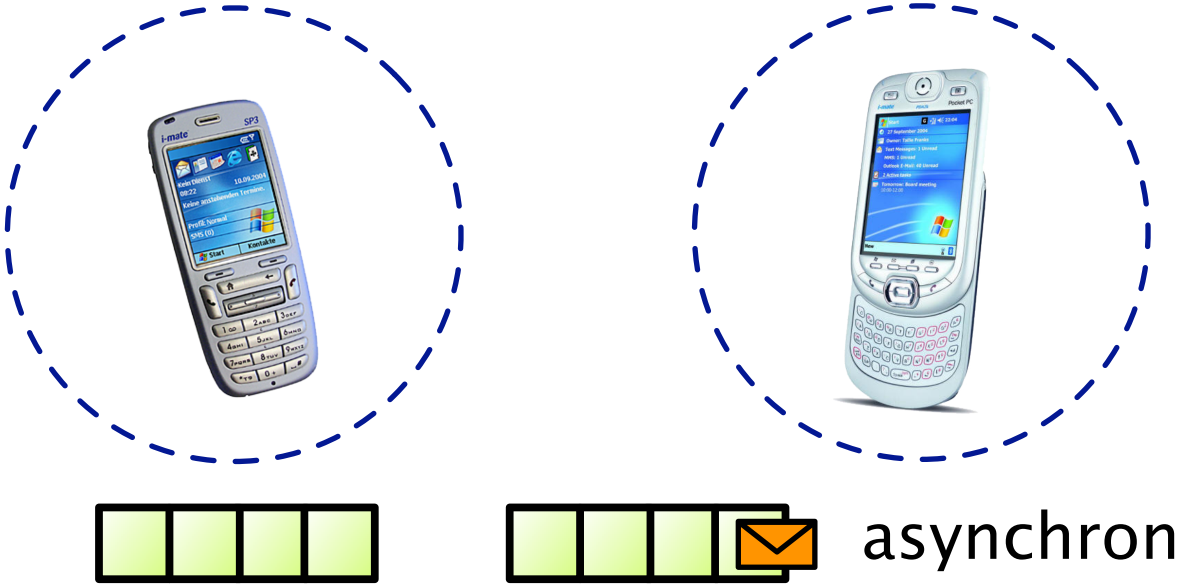
reduces impact of volatile connections



Loose Coupling

Decoupling communication in *Time & Synchronisation*

reduces impact of volatile connections



asynchronous
receive

Loose Coupling

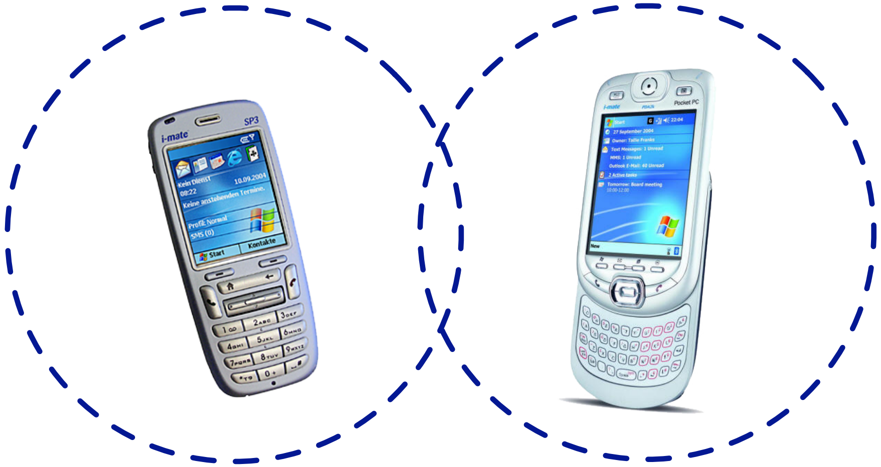
Decoupling communication in *Space*

enables ad hoc anonymous collaborations

Loose Coupling

Decoupling communication in *Space*

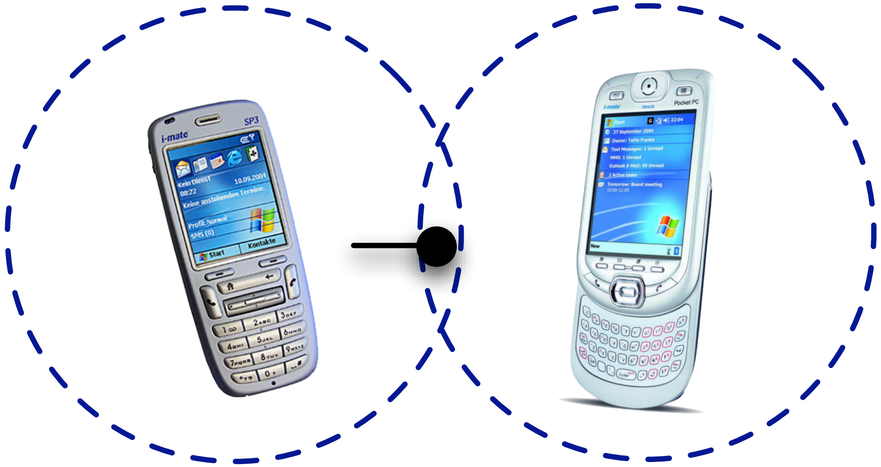
enables ad hoc anonymous collaborations



Loose Coupling

Decoupling communication in *Space*

enables ad hoc anonymous collaborations

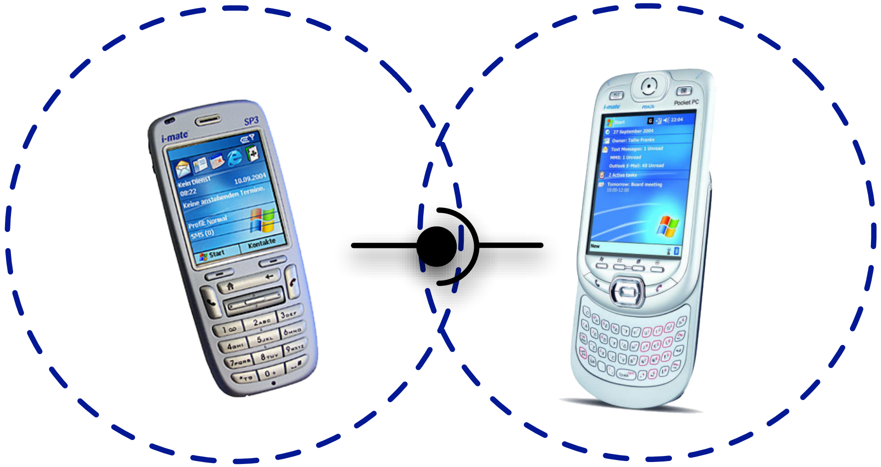


provide service

Loose Coupling

Decoupling communication in *Space*

enables ad hoc anonymous collaborations

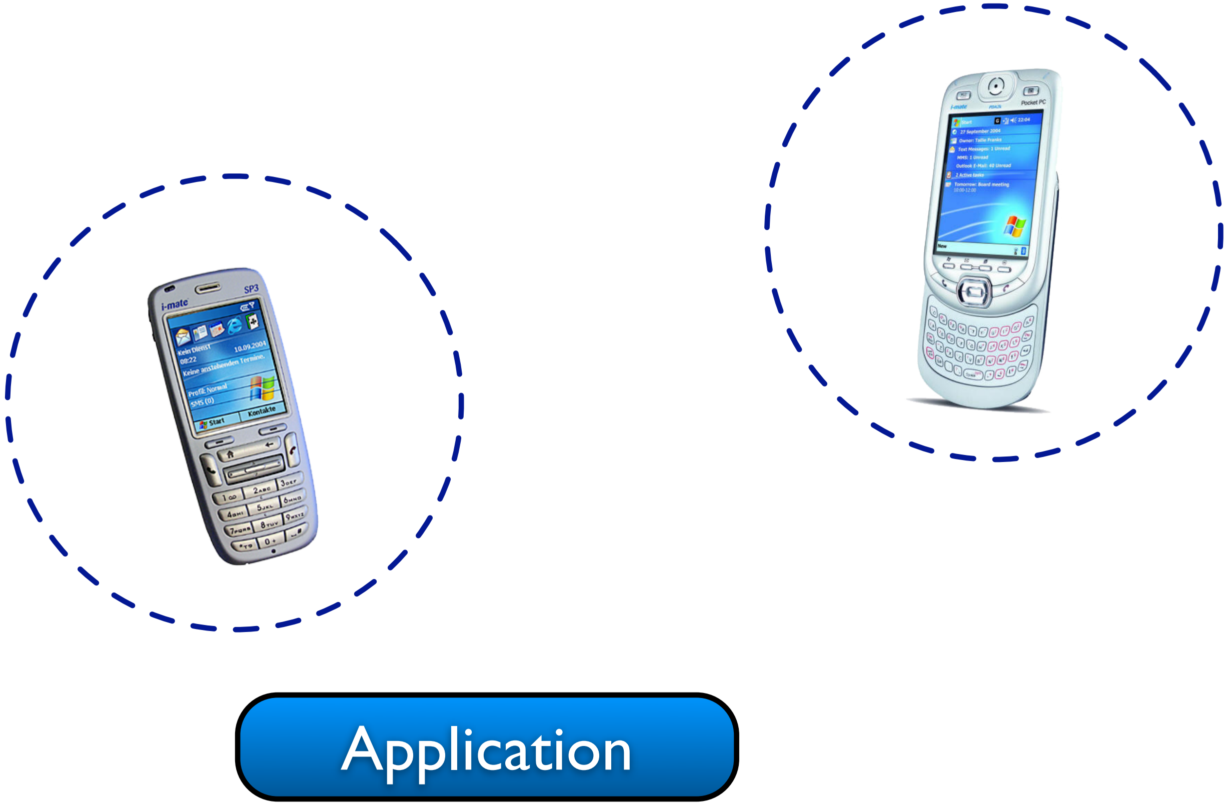


provide service

require service

Connection Awareness

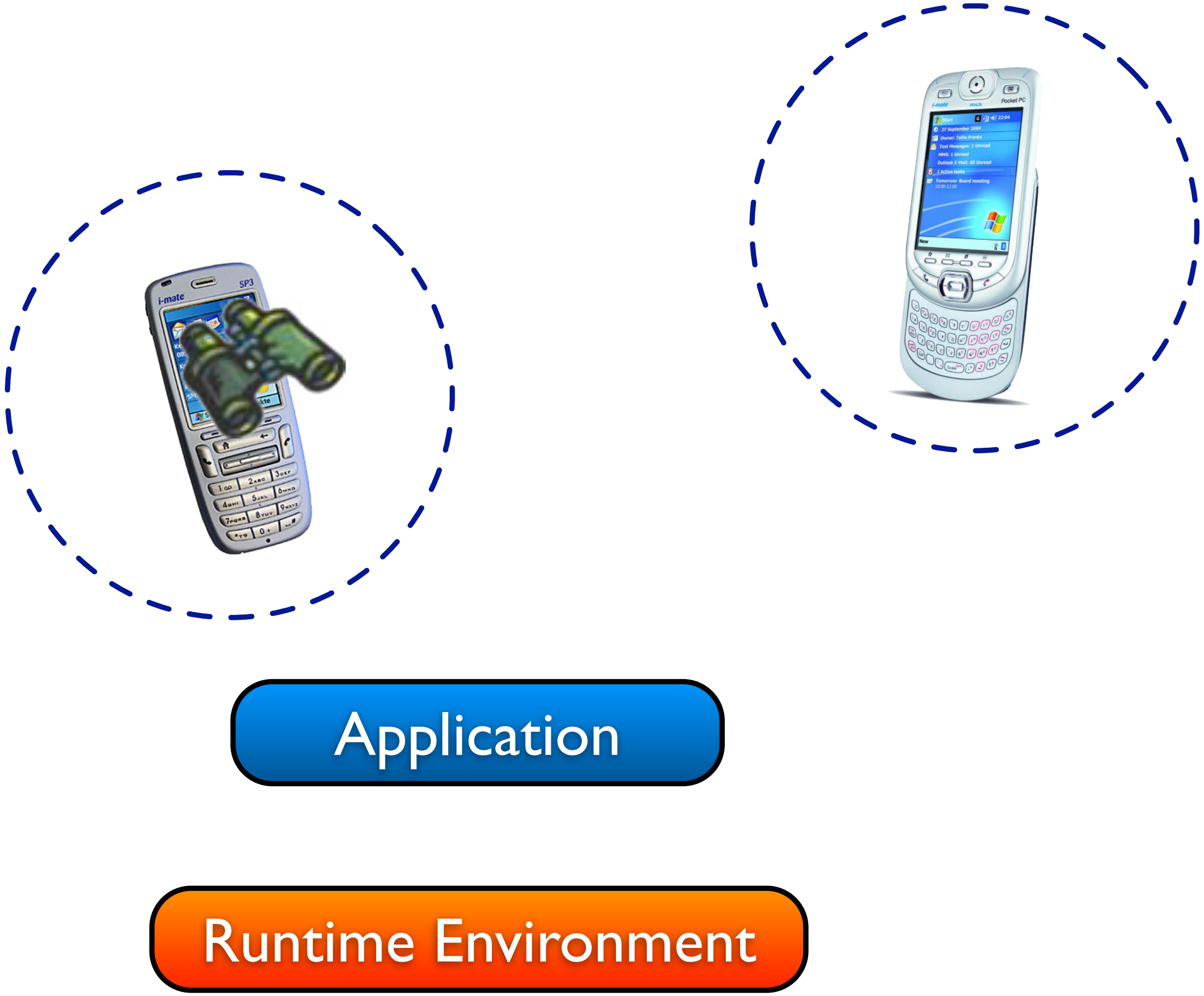
Keeping an up-to-date view on the environment



Application

Connection Awareness

Keeping an up-to-date view on the environment

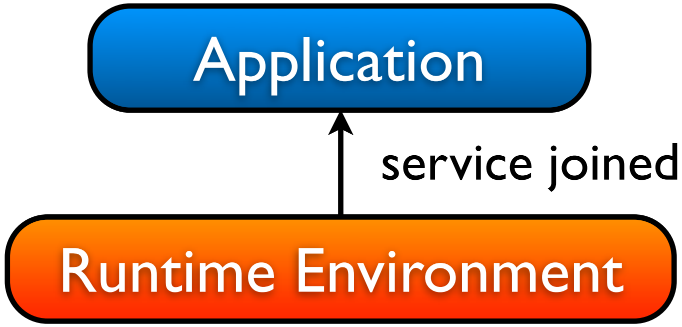


Application

Runtime Environment

Connection Awareness

Keeping an up-to-date view on the environment



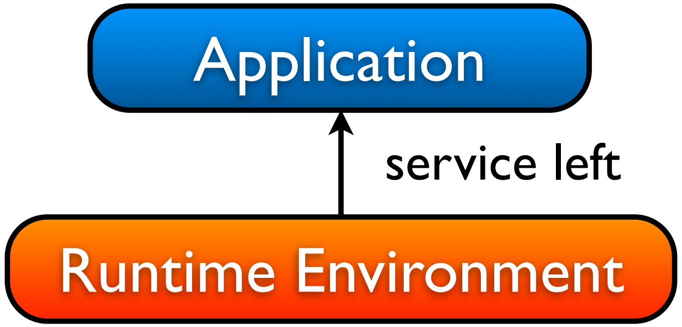
Application

service joined

Runtime Environment

Connection Awareness

Keeping an up-to-date view on the environment



Application

service left

Runtime Environment

Paradigm Mismatch

Object-oriented programming model

Remote object references

- One-to-one communication
- Easy request/response correlation

loosely-coupled communication model

Publish/subscribe
Tuple Spaces

- Decoupled in time
- Decoupled in space
- Synchronization decoupling

Paradigm Mismatch

Object-oriented programming model

Remote object references

- One-to-one communication
- Easy request/response correlation

Ambient References

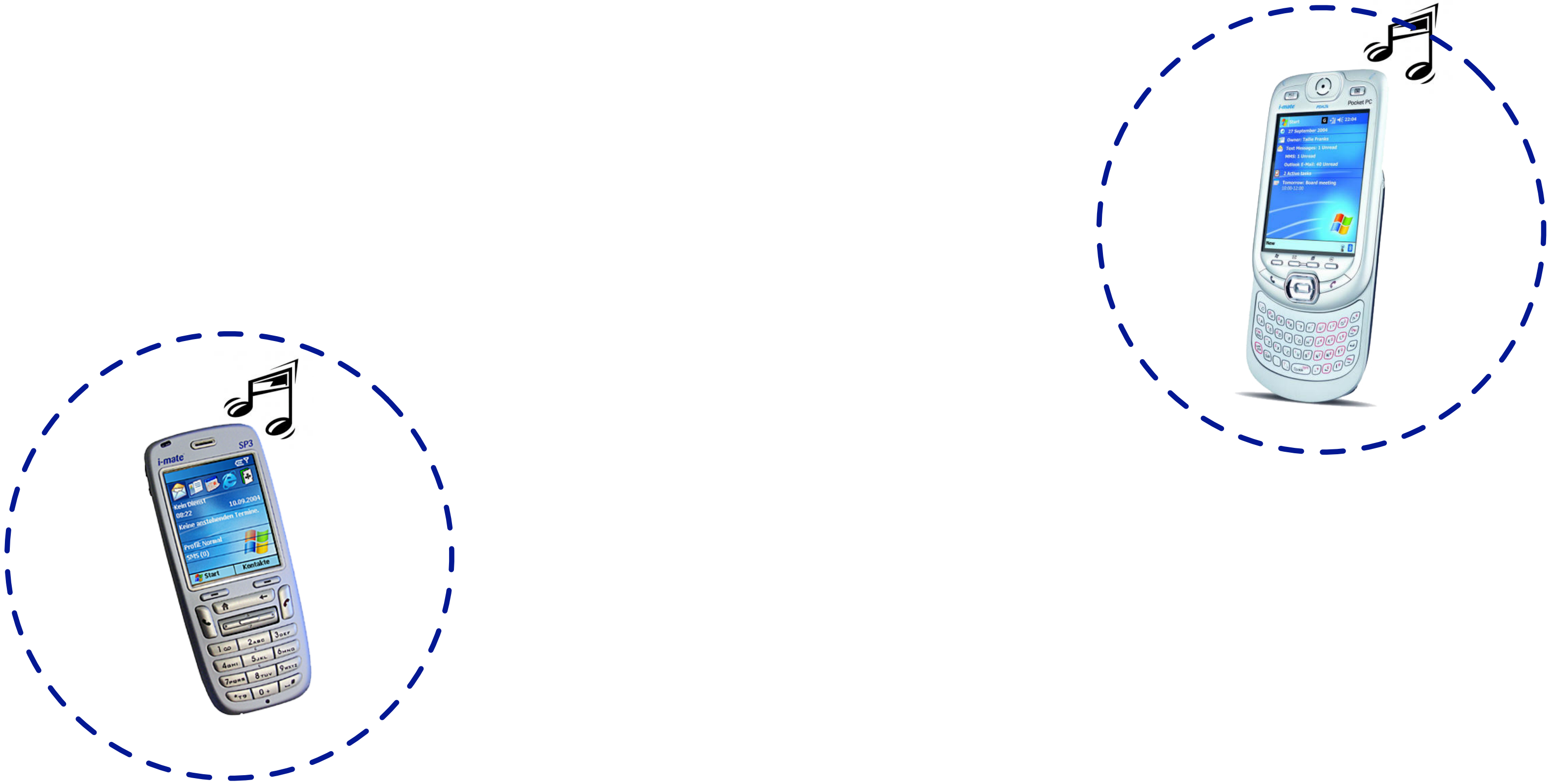
loosely-coupled communication model

Publish/subscribe

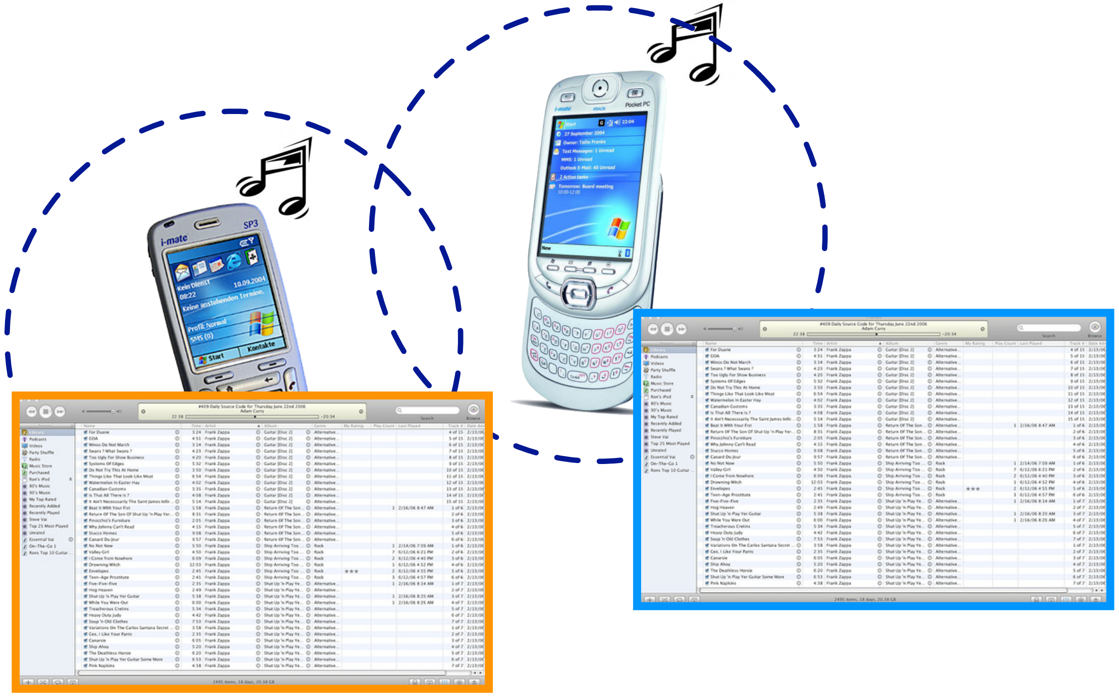
Tuple Spaces

- Decoupled in time
- Decoupled in space
- Synchronization decoupling

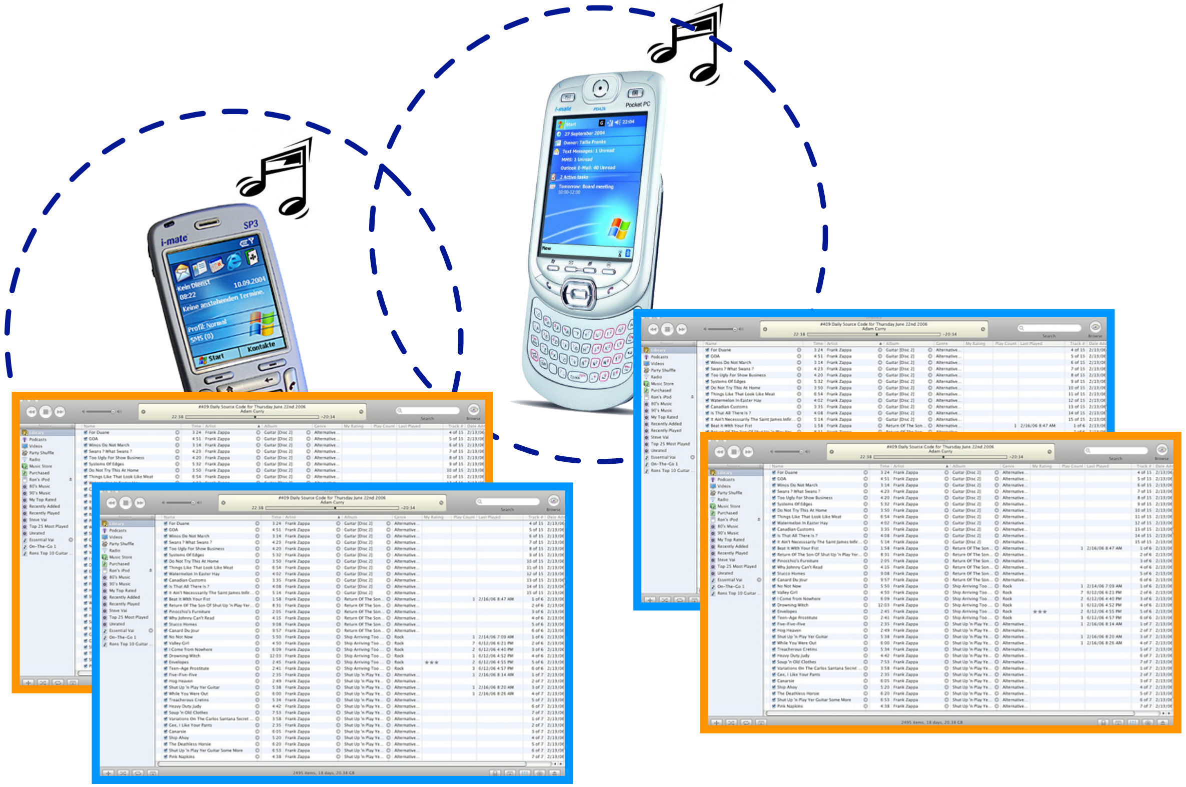
Example: music player



Example: music player



Example: music player



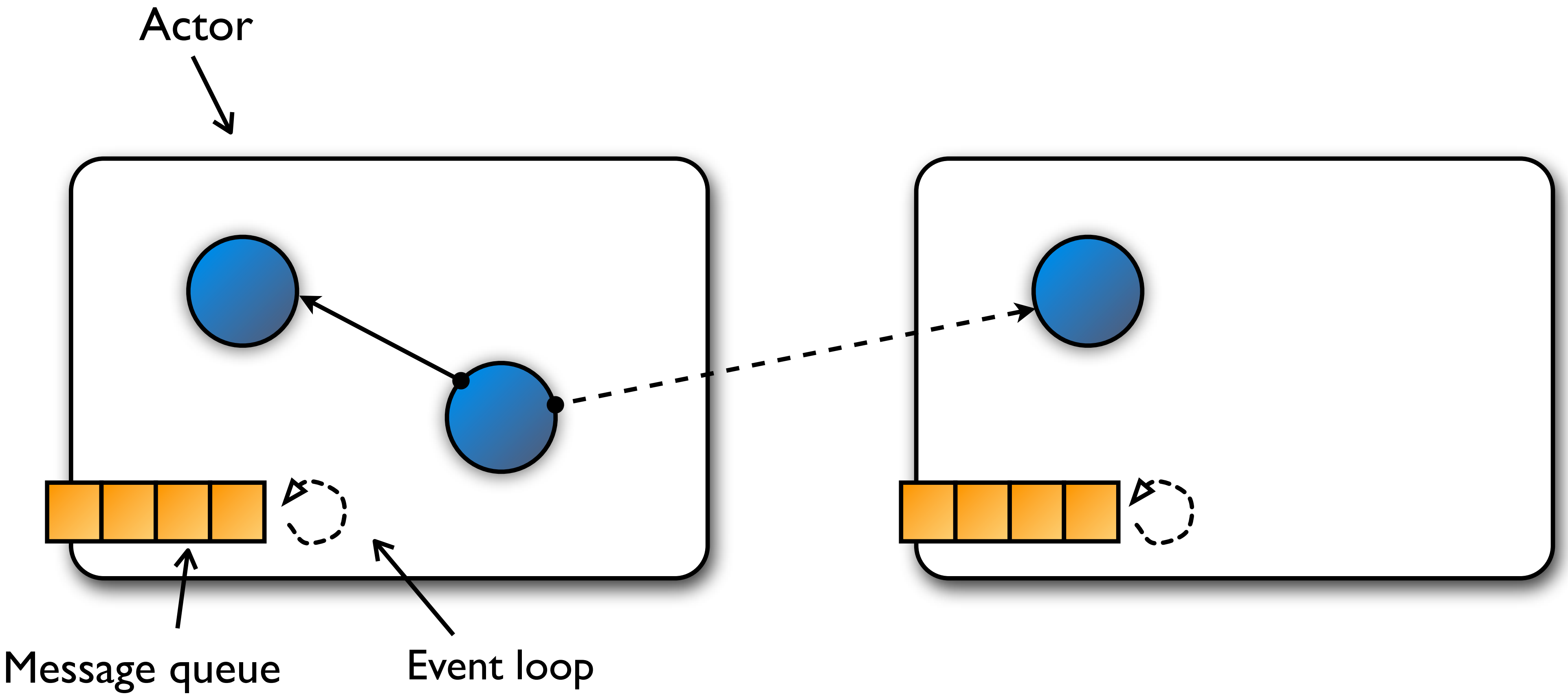
Example: music player

AmbientTalk

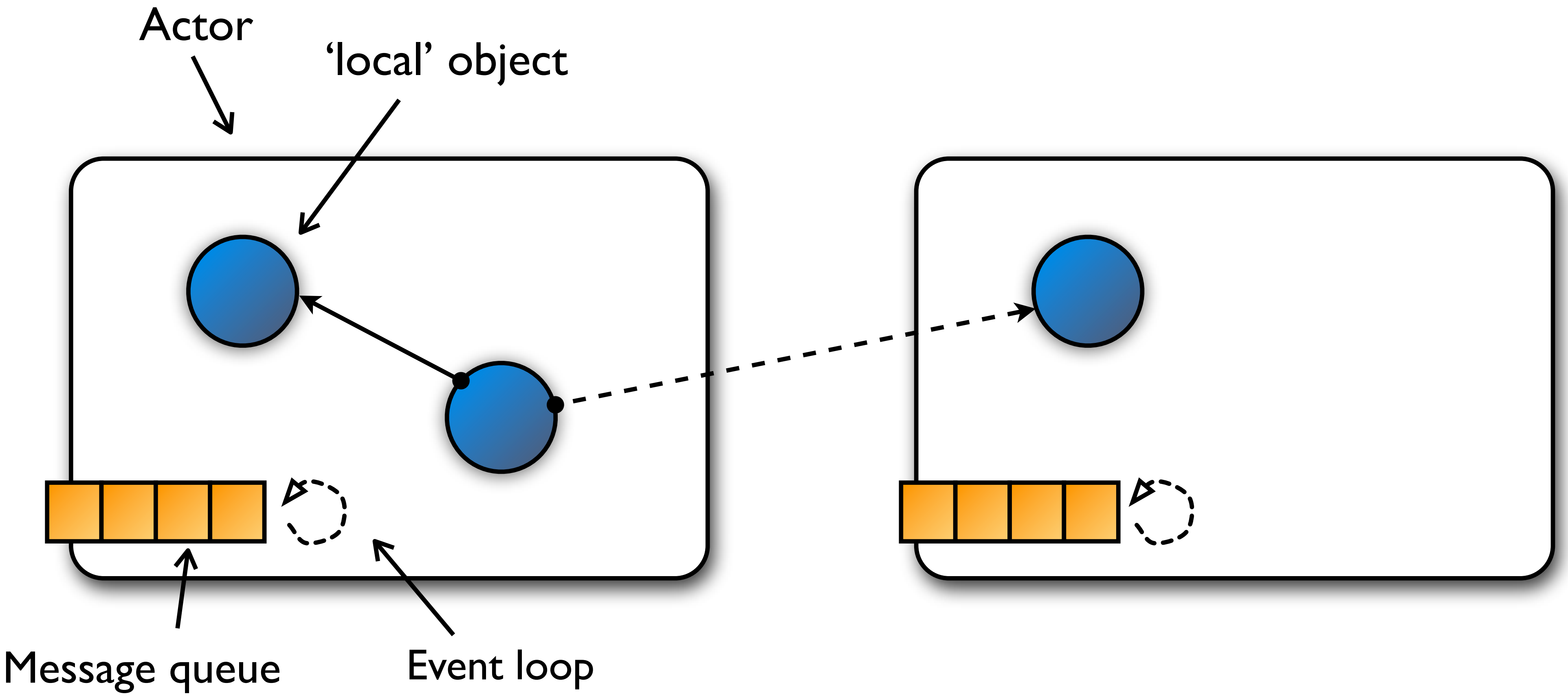
- Distributed object-oriented language
- Event-driven concurrency based on actors
- Future-type asynchronous message sends
- Built-in publish/subscribe engine for service discovery of remote objects

AMBIENTTALK

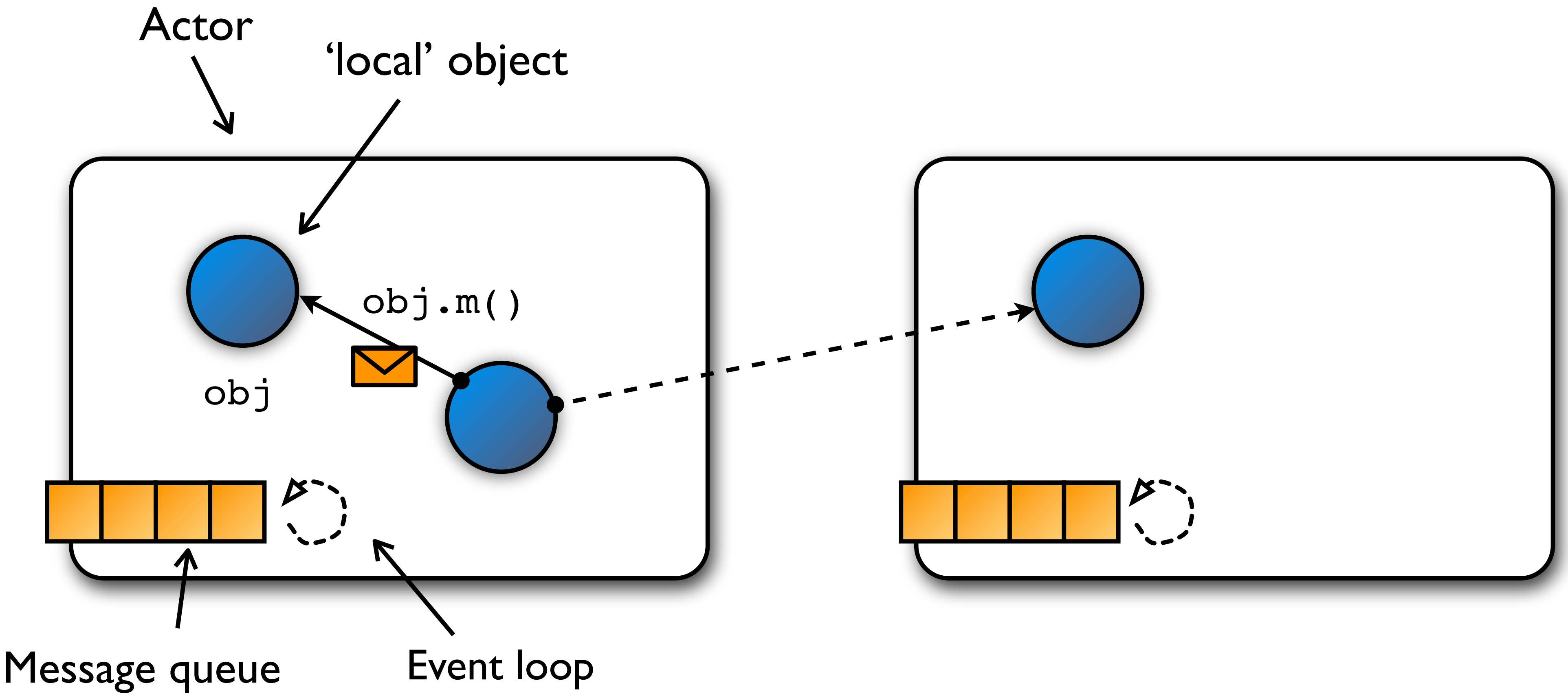
Event loop concurrency



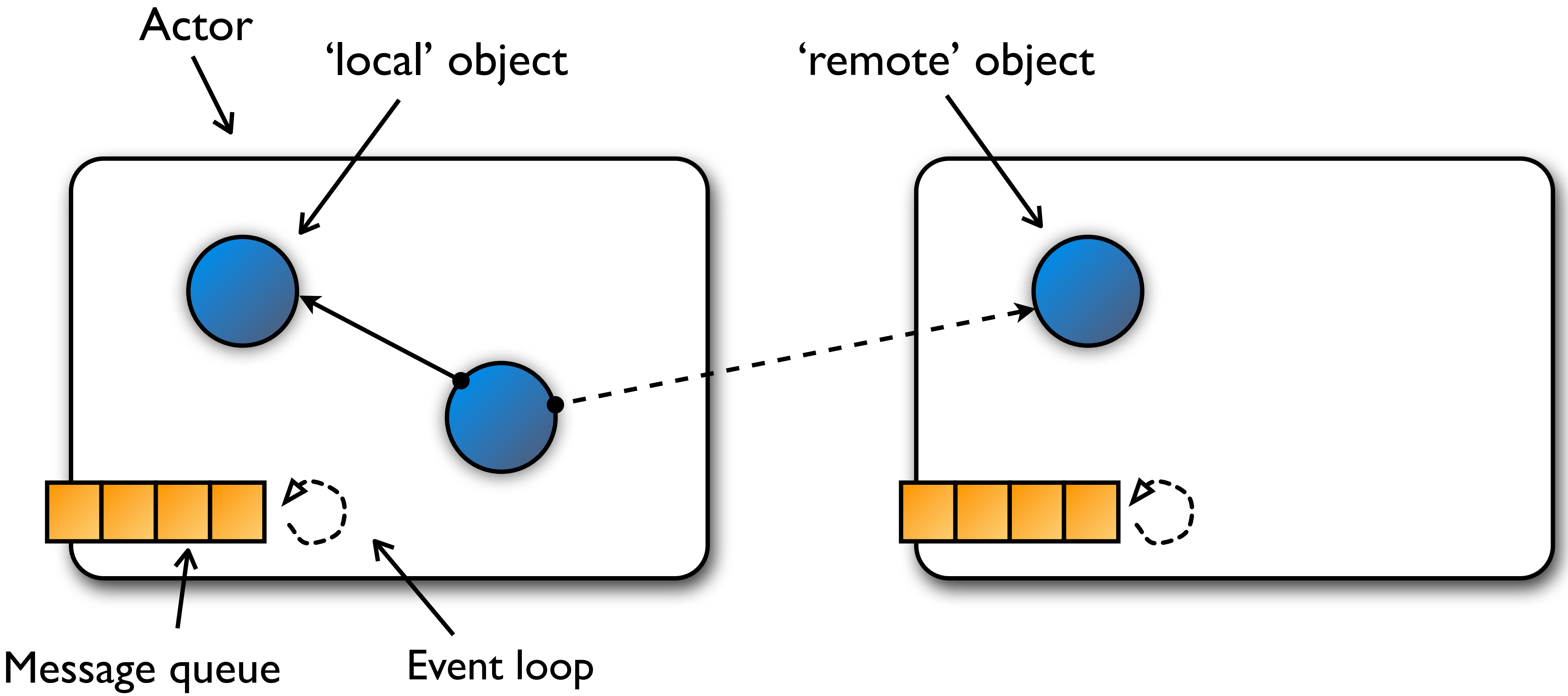
Event loop concurrency



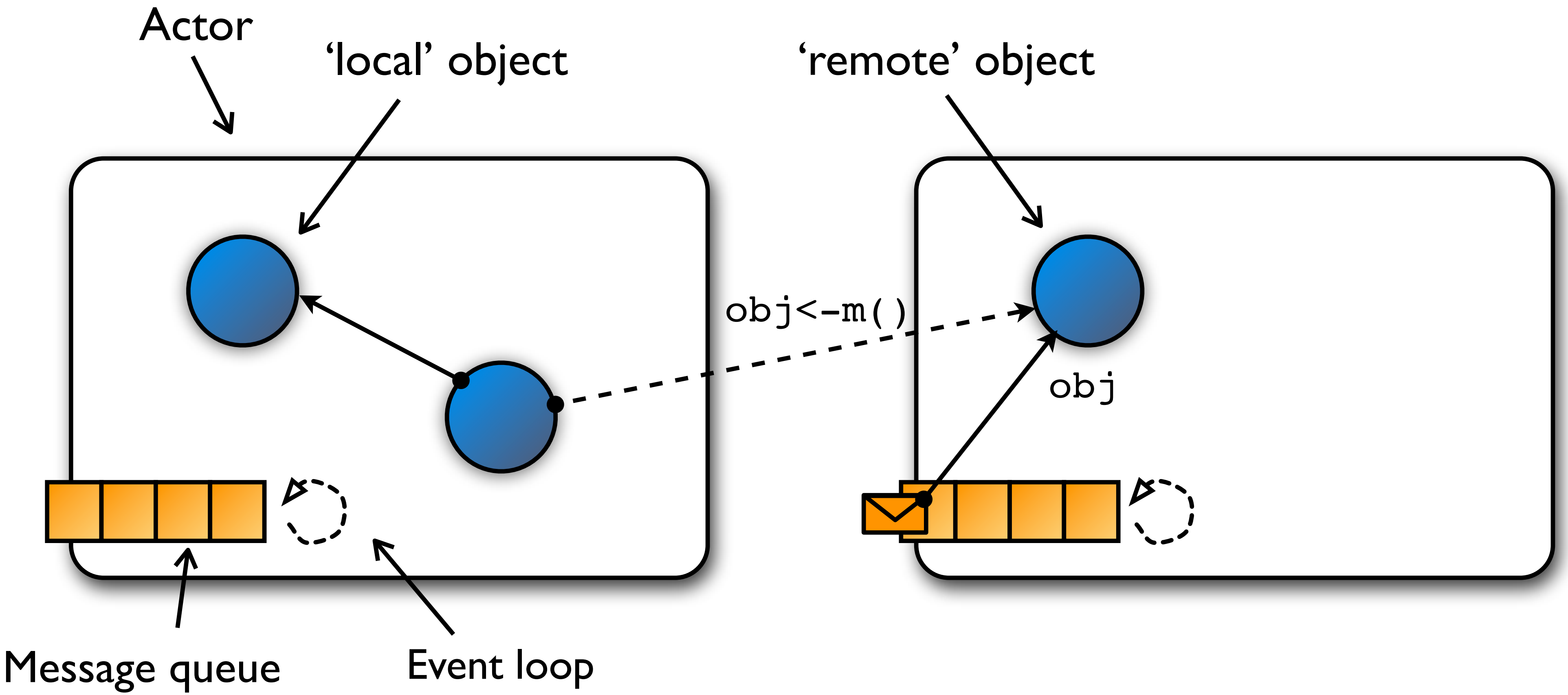
Event loop concurrency



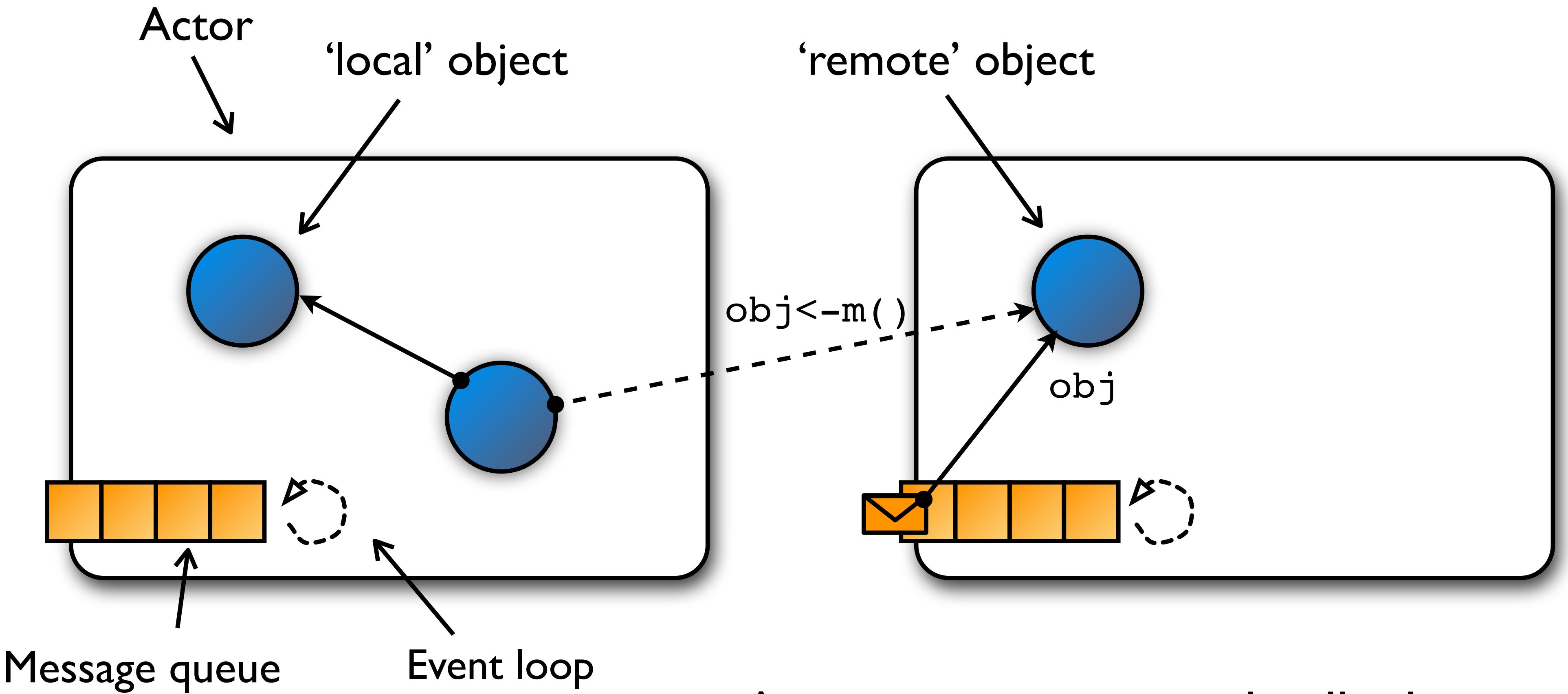
Event loop concurrency



Event loop concurrency



Event loop concurrency

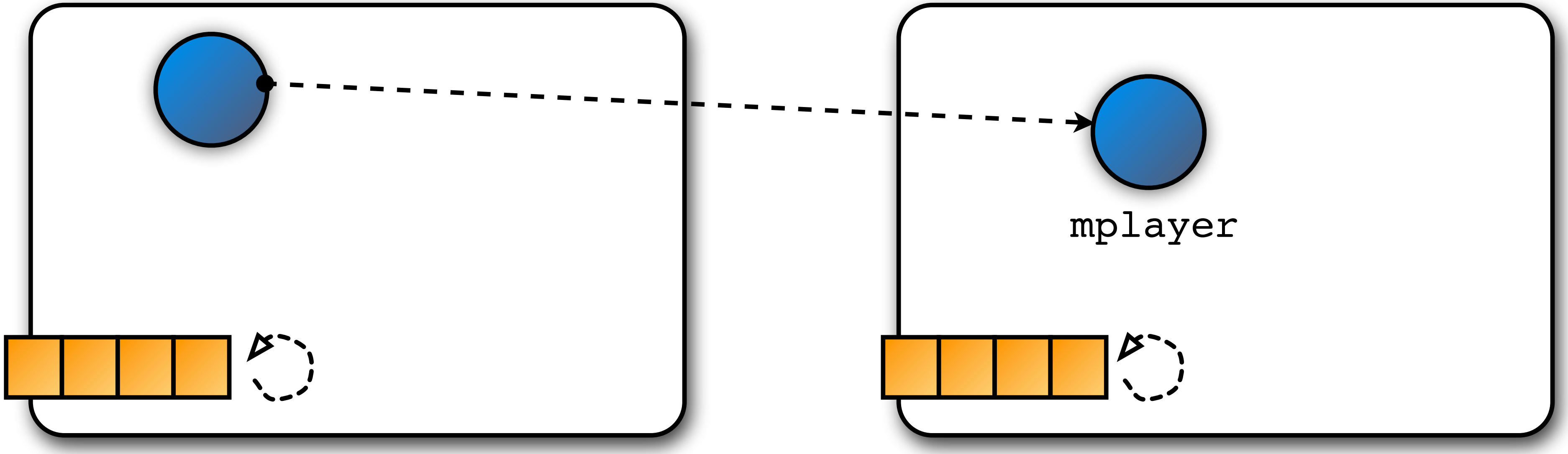


Actors cannot cause deadlock

No race conditions on objects

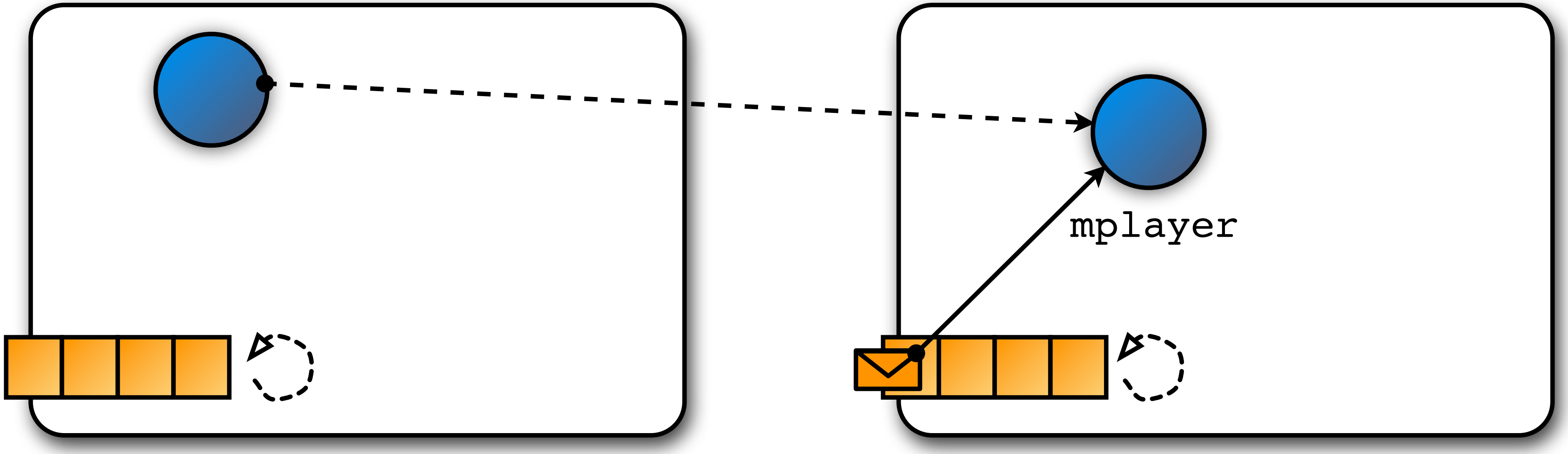
Futures

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



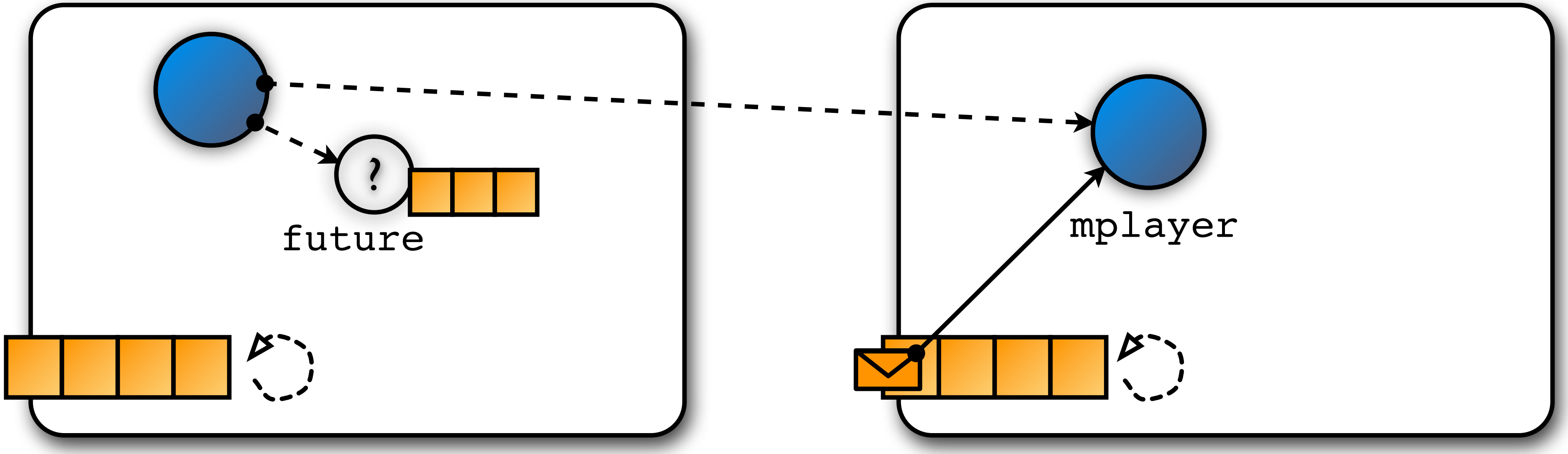
Futures

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



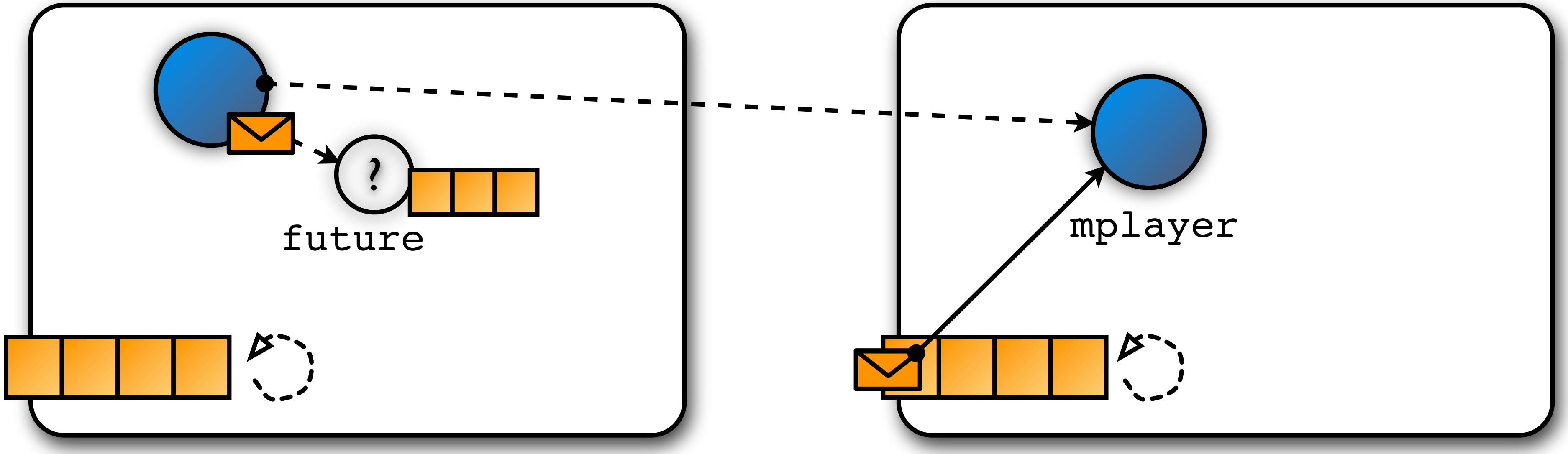
Futures

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



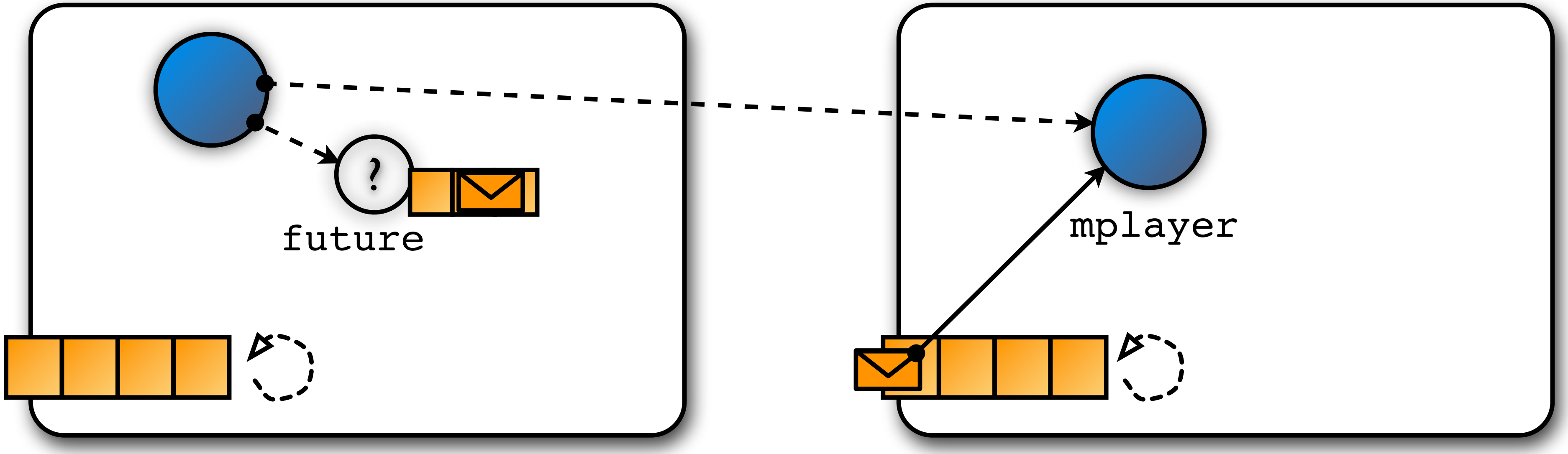
Futures

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



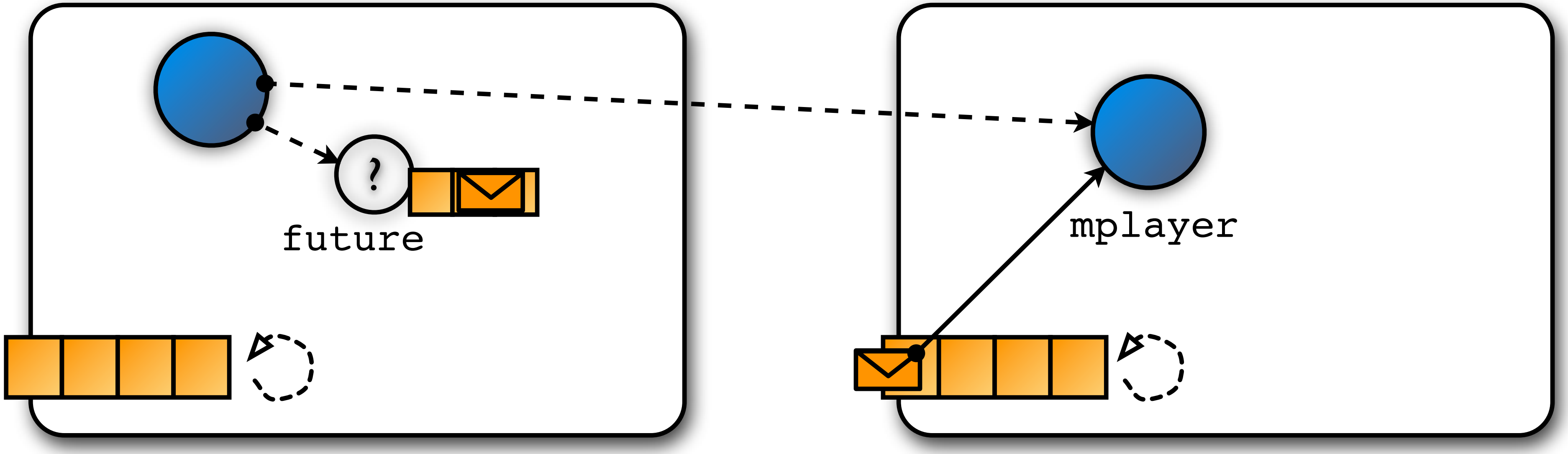
Futures

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



Futures

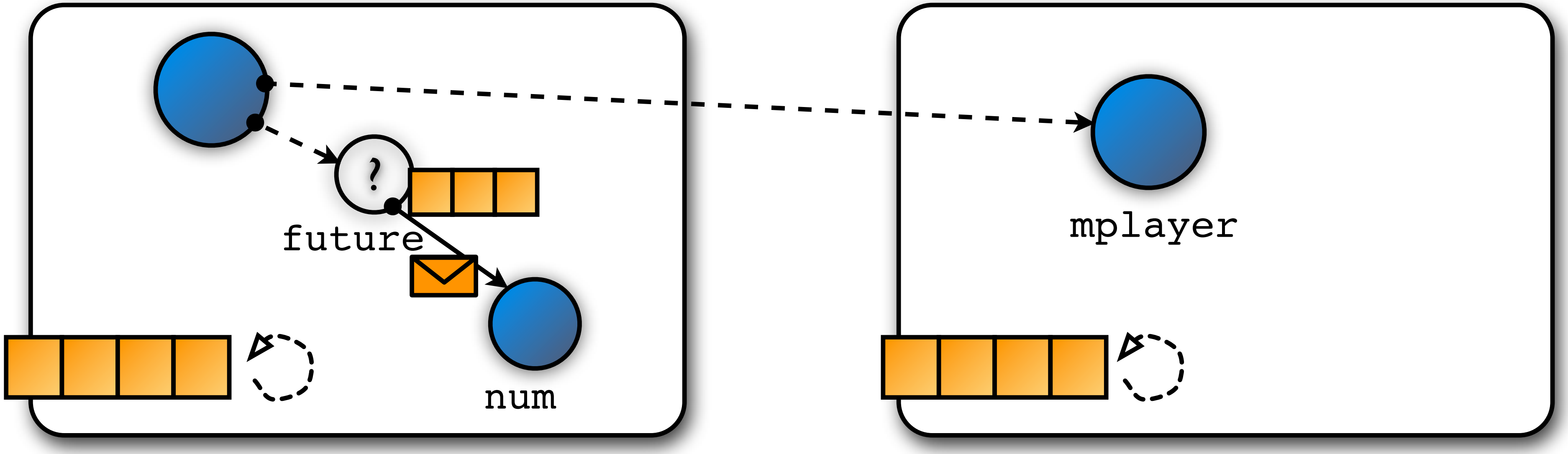
```
def future := mplayer<-numSongsInLibrary()
```



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

Futures

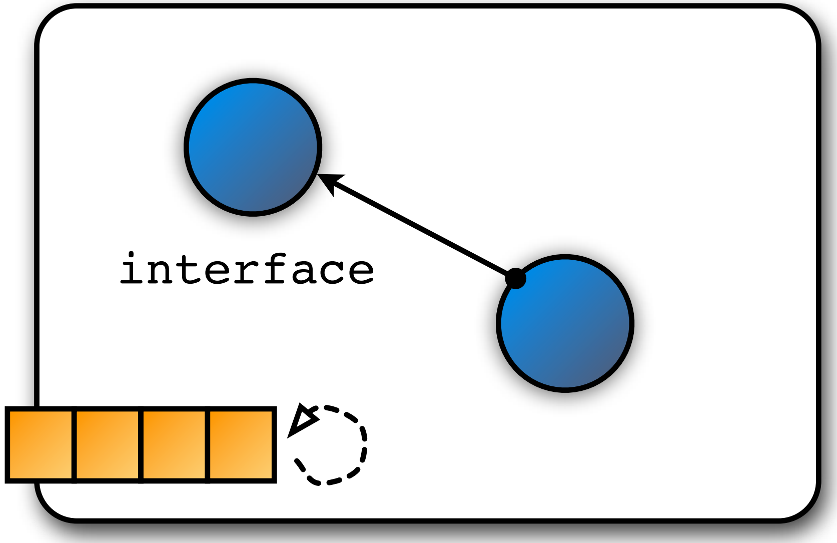
```
def future := mplayer<-numSongsInLibrary()
```



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

Exporting objects

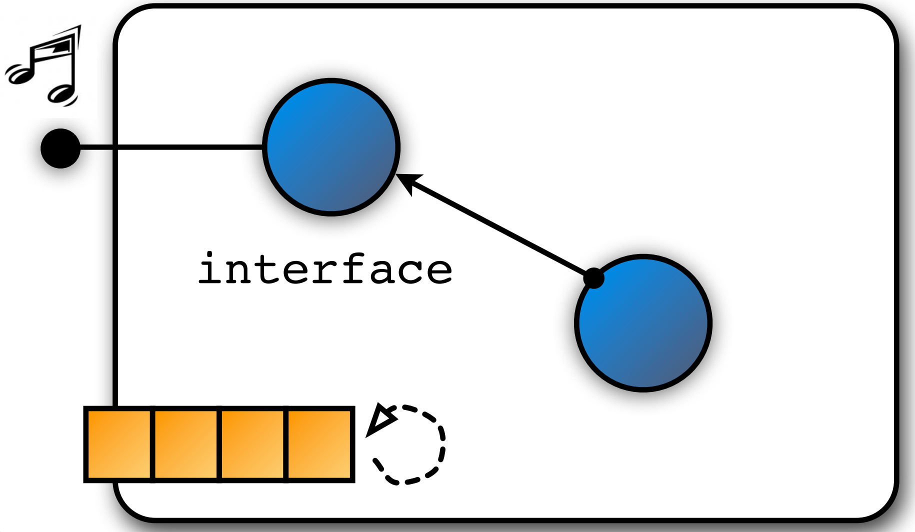
```
deftype MusicPlayer;

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

export: interface as: MusicPlayer;
```



Exporting objects

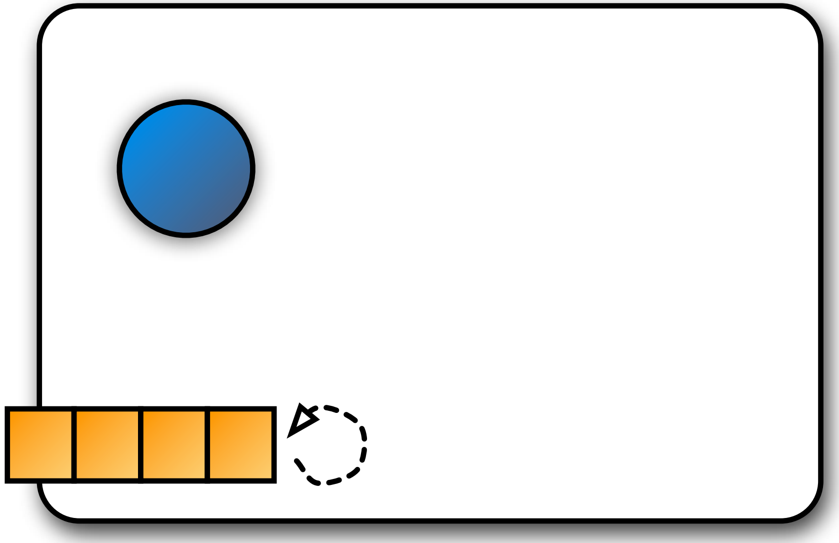
```
deftype MusicPlayer;

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

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



Ambient References

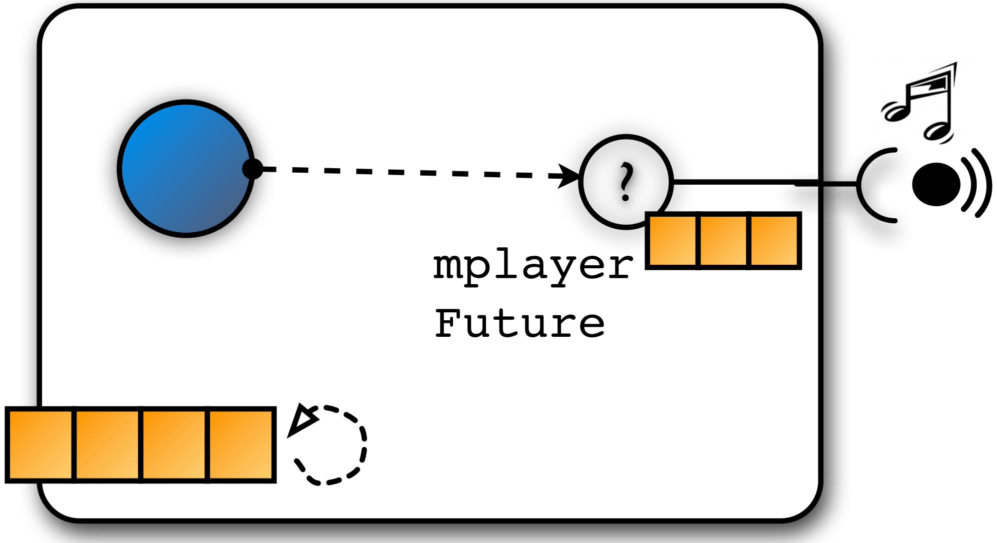
```
def mplayerFuture := ambient: MusicPlayer;
```



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

Ambient References

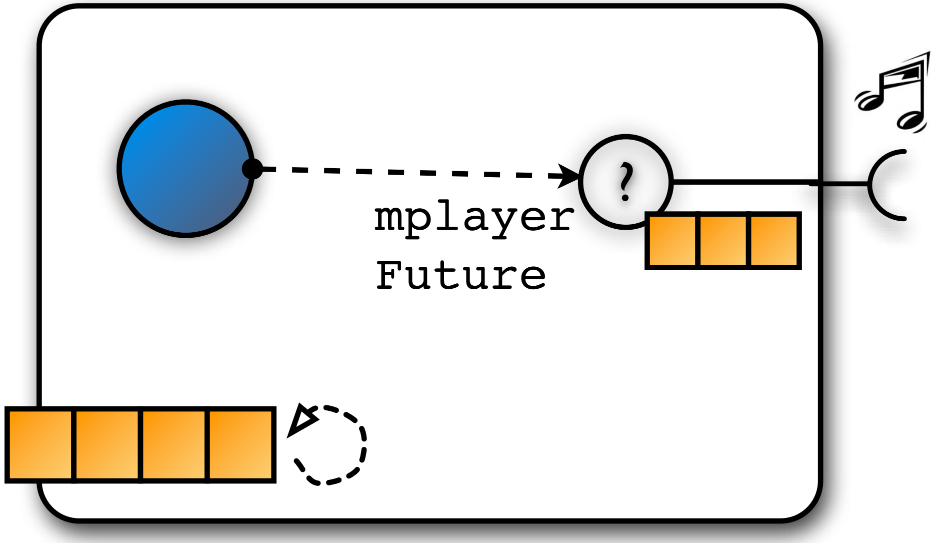
```
def mplayerFuture := ambient: MusicPlayer;
```



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

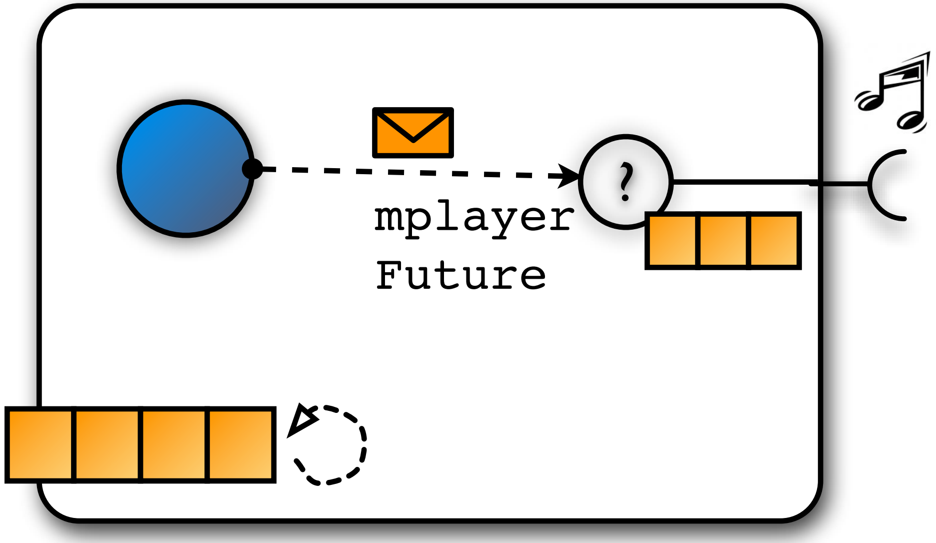
Ambient References

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



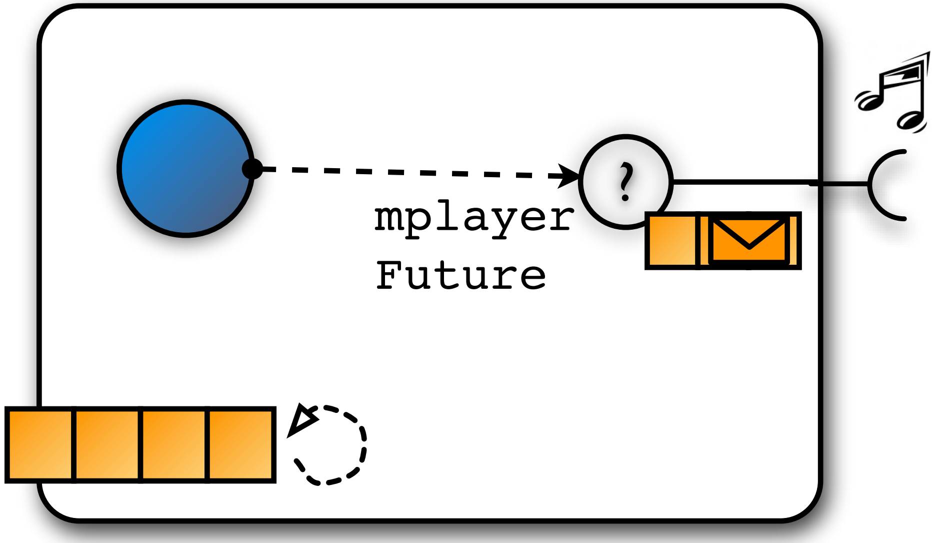
Ambient References

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



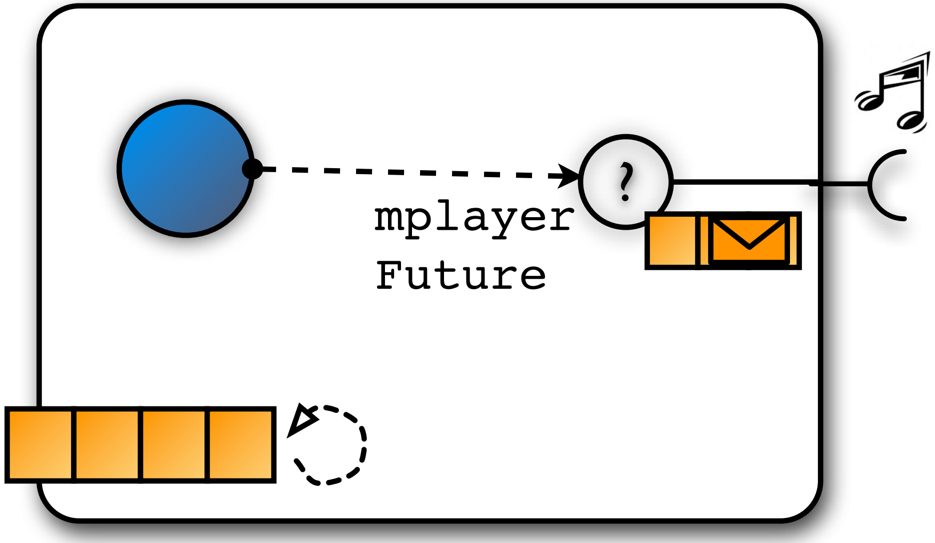
Ambient References

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



Ambient References

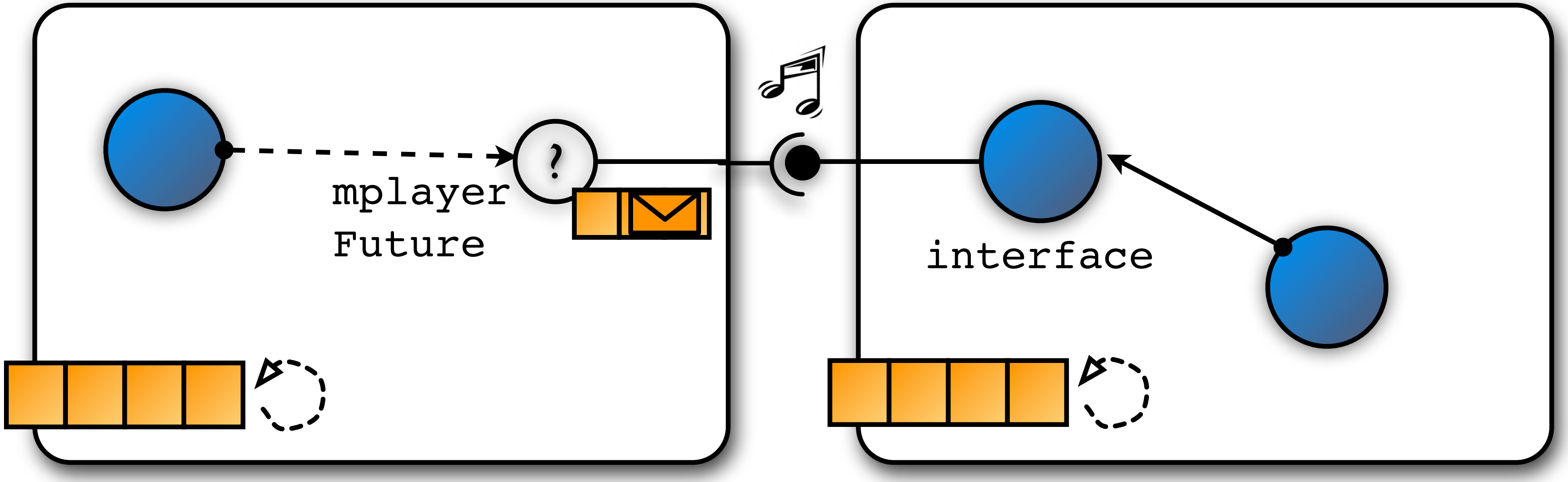
```
def mplayerFuture := ambient: MusicPlayer;
def sessionFuture := mplayerFuture<-openSession();
```



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

Ambient References

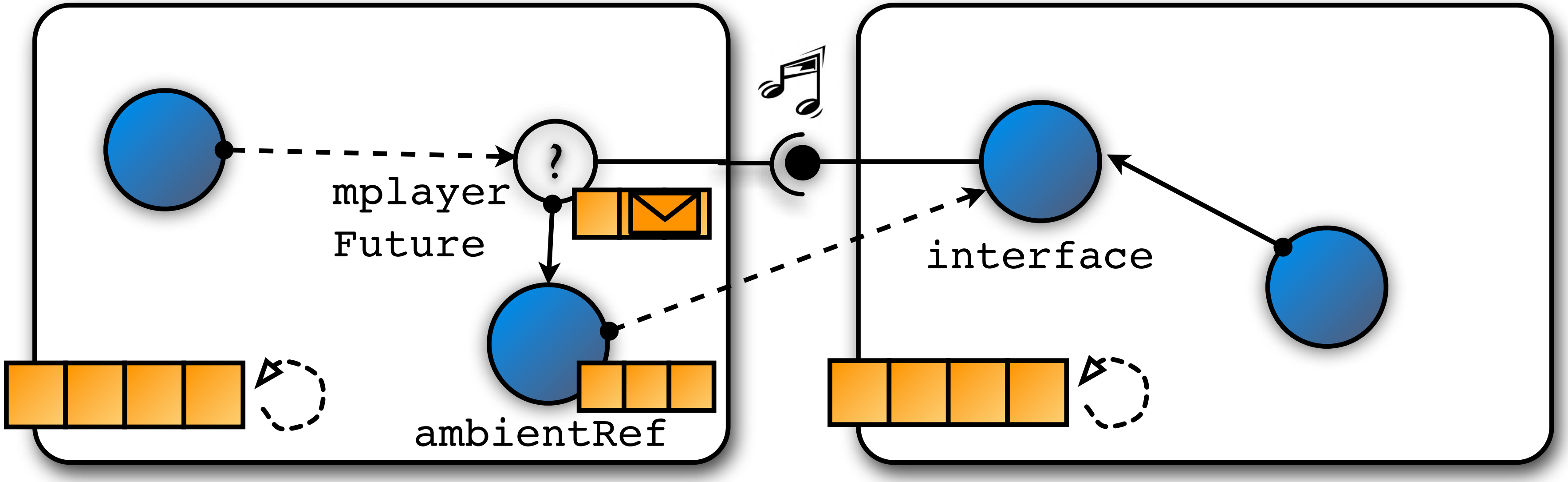
```
def mplayerFuture := ambient: MusicPlayer;
def sessionFuture := mplayerFuture<-openSession();
```



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

Ambient References

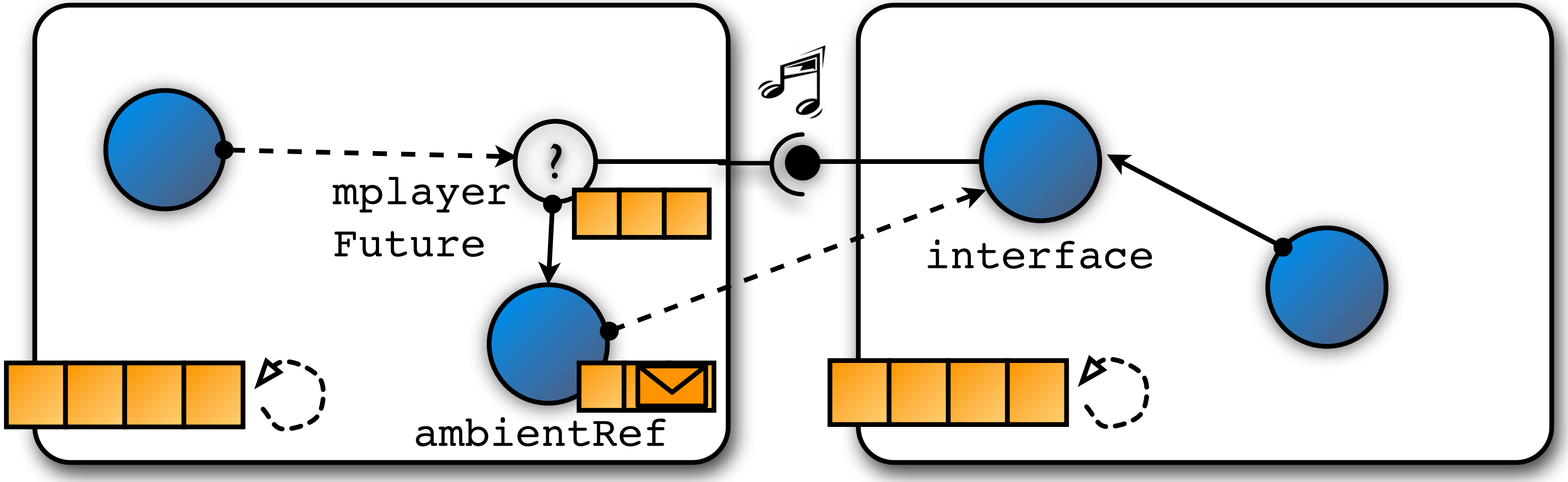
```
def mplayerFuture := ambient: MusicPlayer;
def sessionFuture := mplayerFuture<-openSession();
```



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

Ambient References

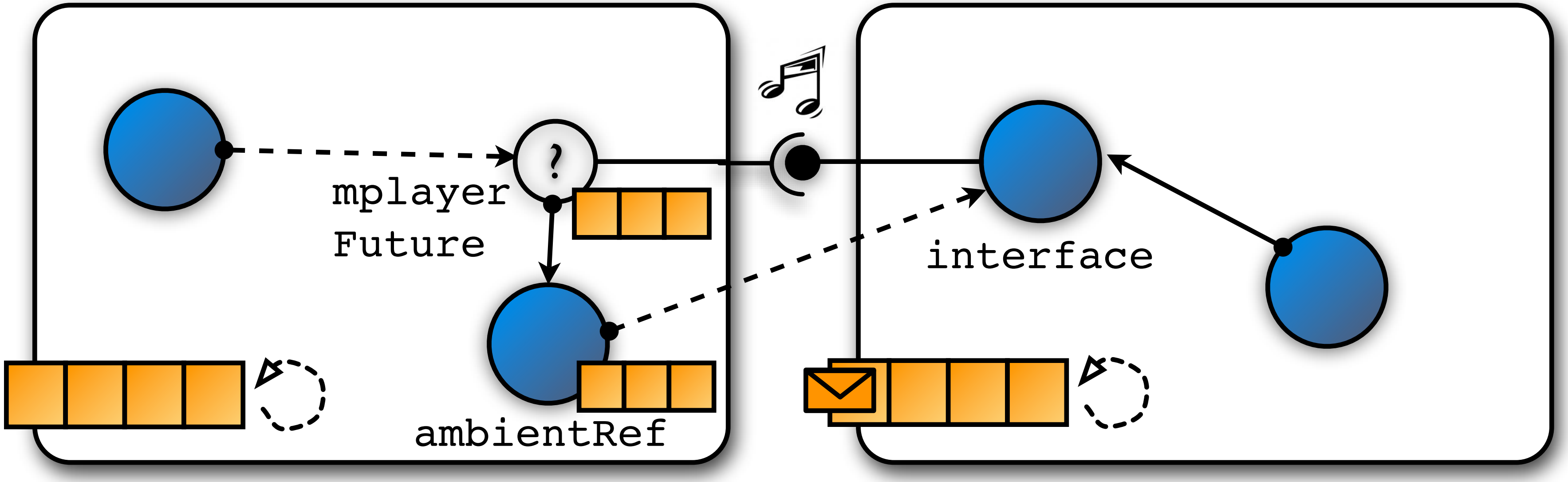
```
def mplayerFuture := ambient: MusicPlayer;
def sessionFuture := mplayerFuture<-openSession();
```



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

Ambient References

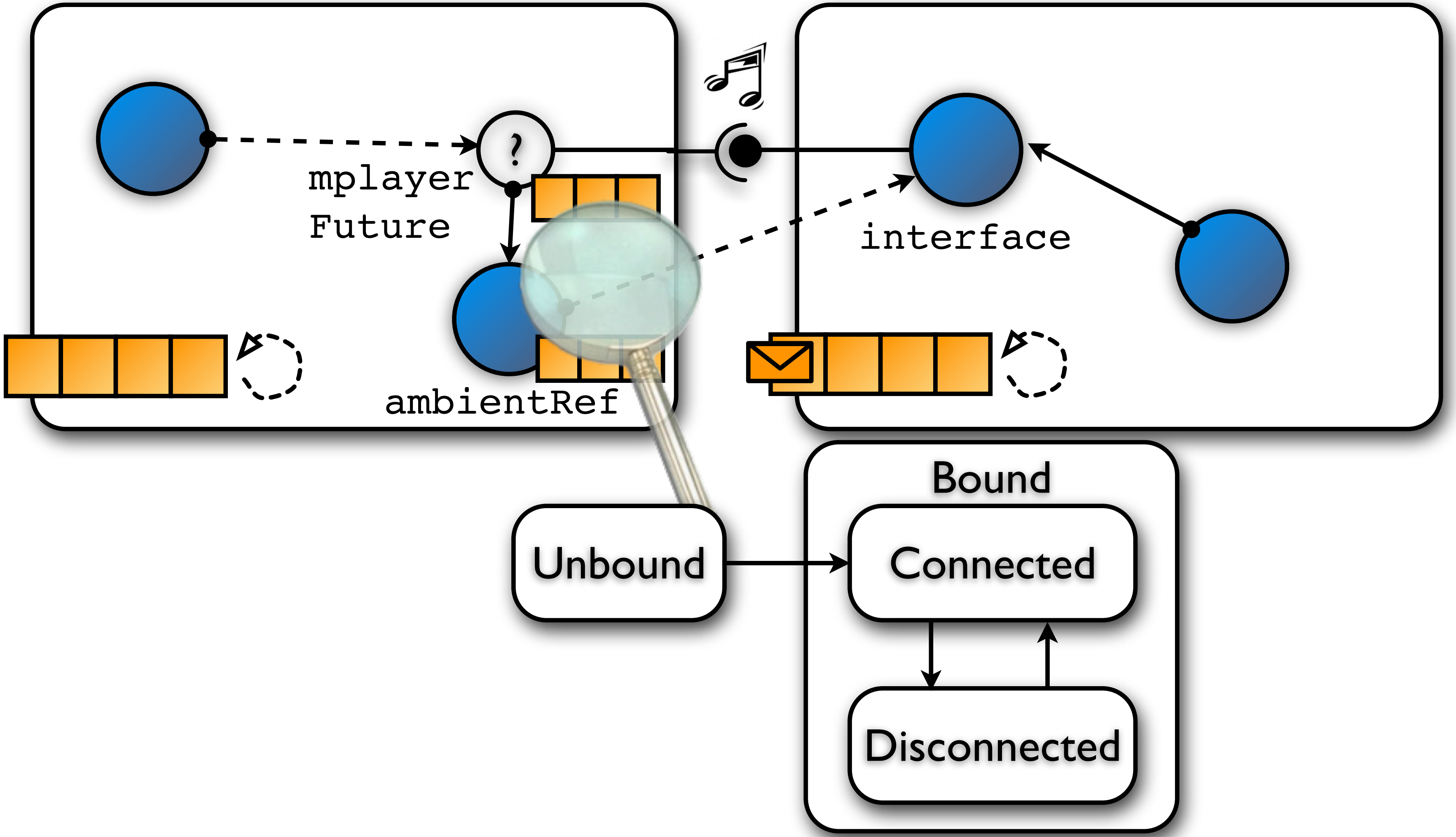
```
def mplayerFuture := ambient: MusicPlayer;
def sessionFuture := mplayerFuture<-openSession();
```



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

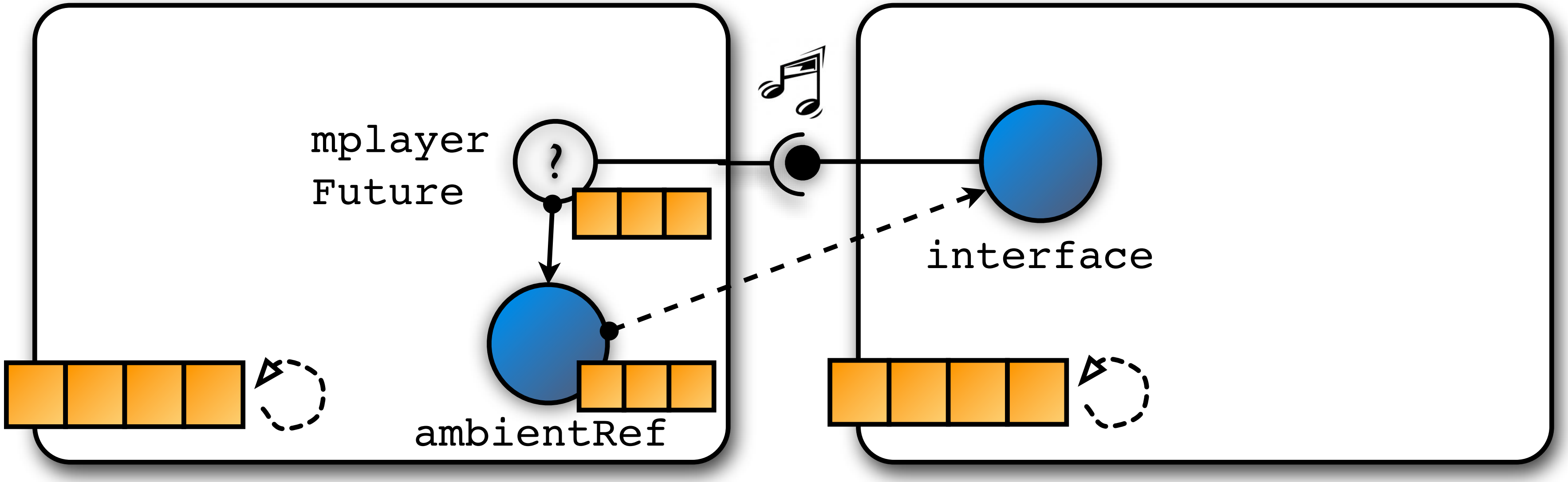
Ambient References

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



Failure handling

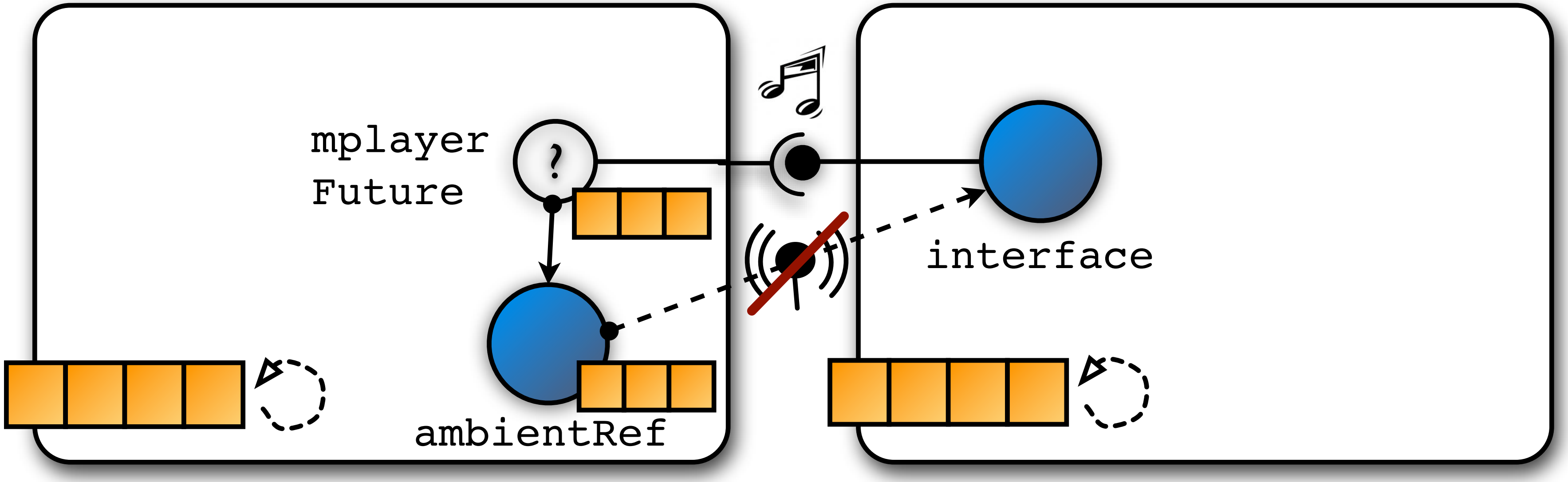
- Observers on ambient references:



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

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

Failure handling

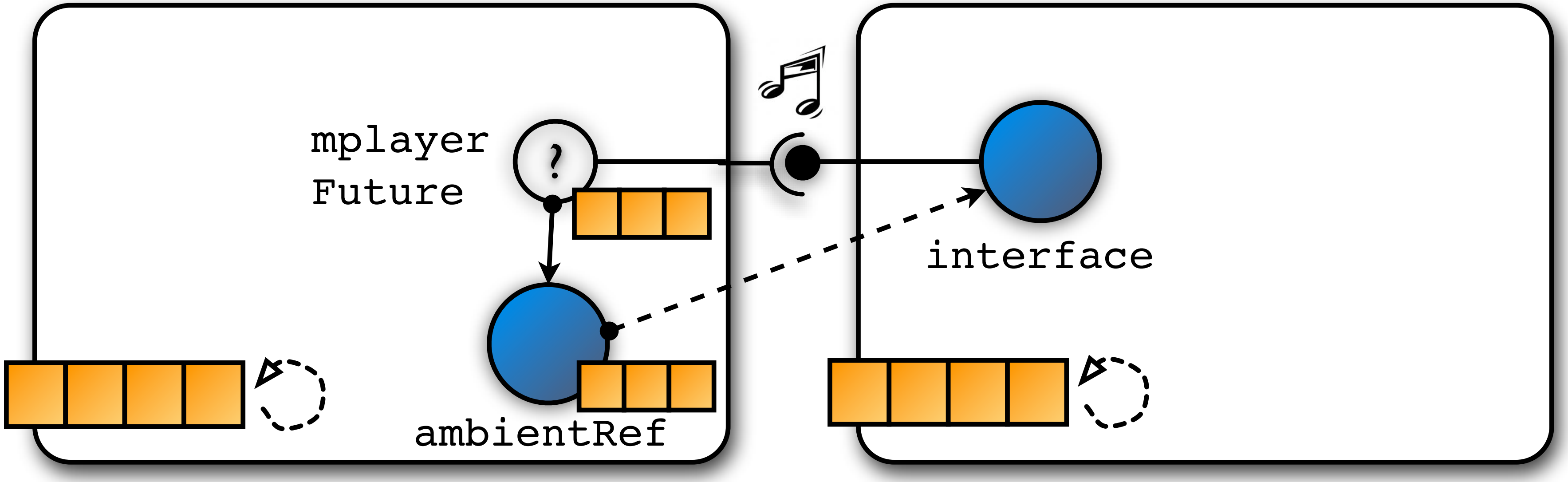
- Observers on ambient references:



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

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

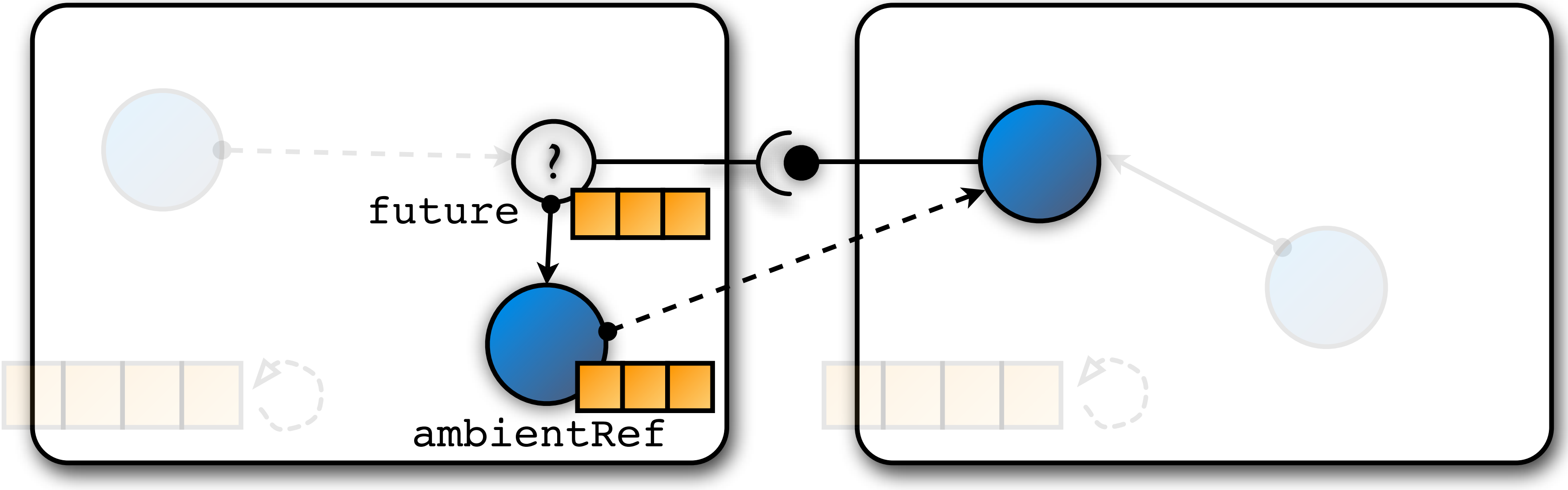
Failure handling

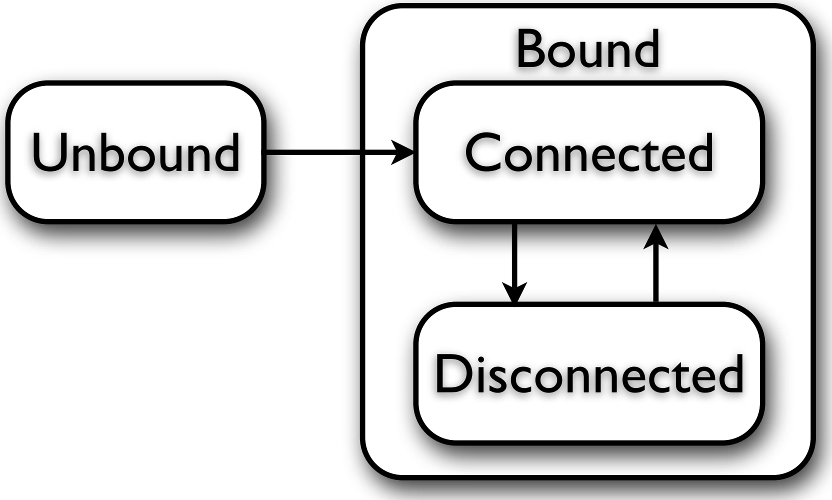
- Observers on ambient references:



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

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

Evaluation





Bound

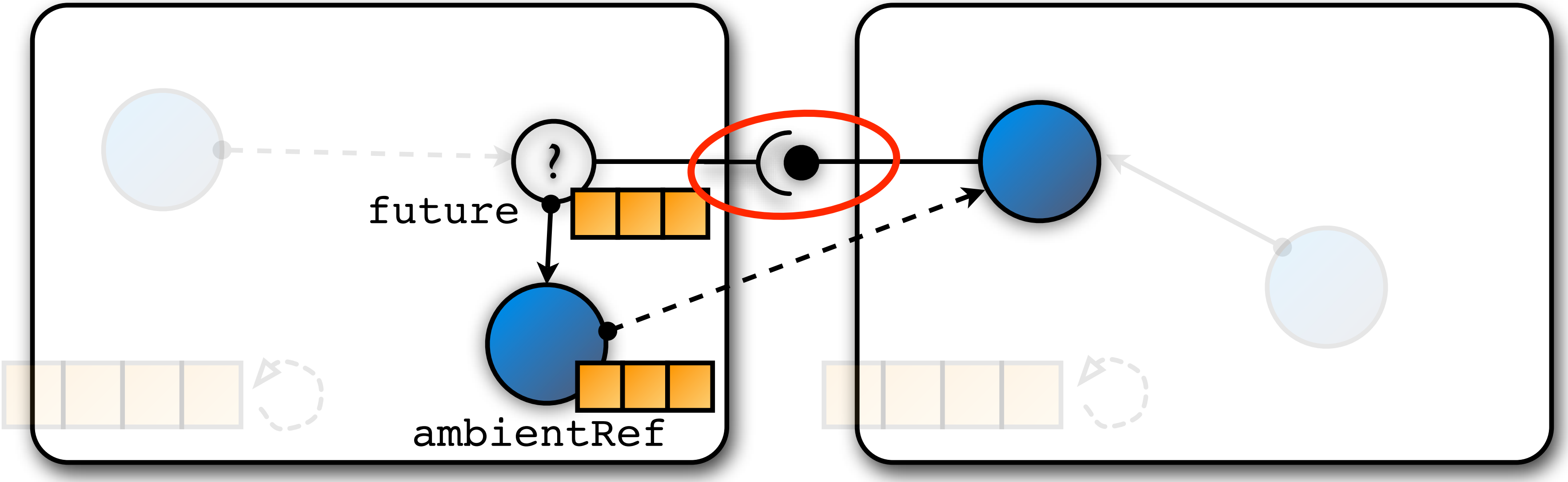
Unbound

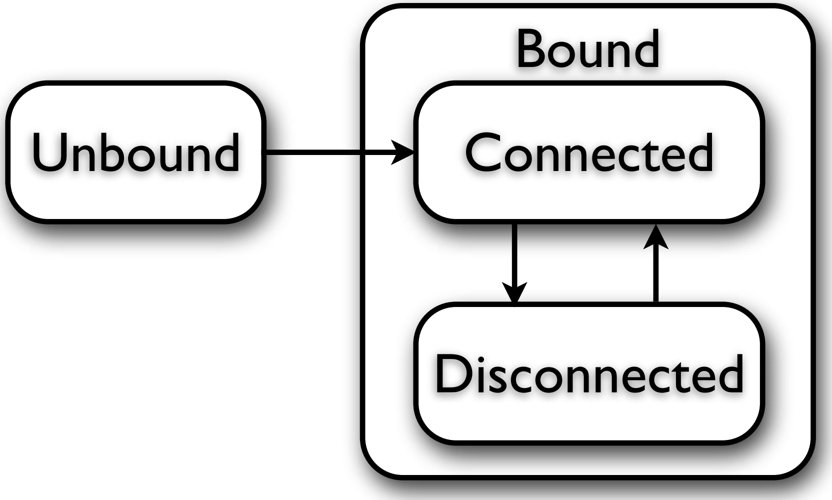
Connected

Disconnected

Evaluation

Decoupling in space



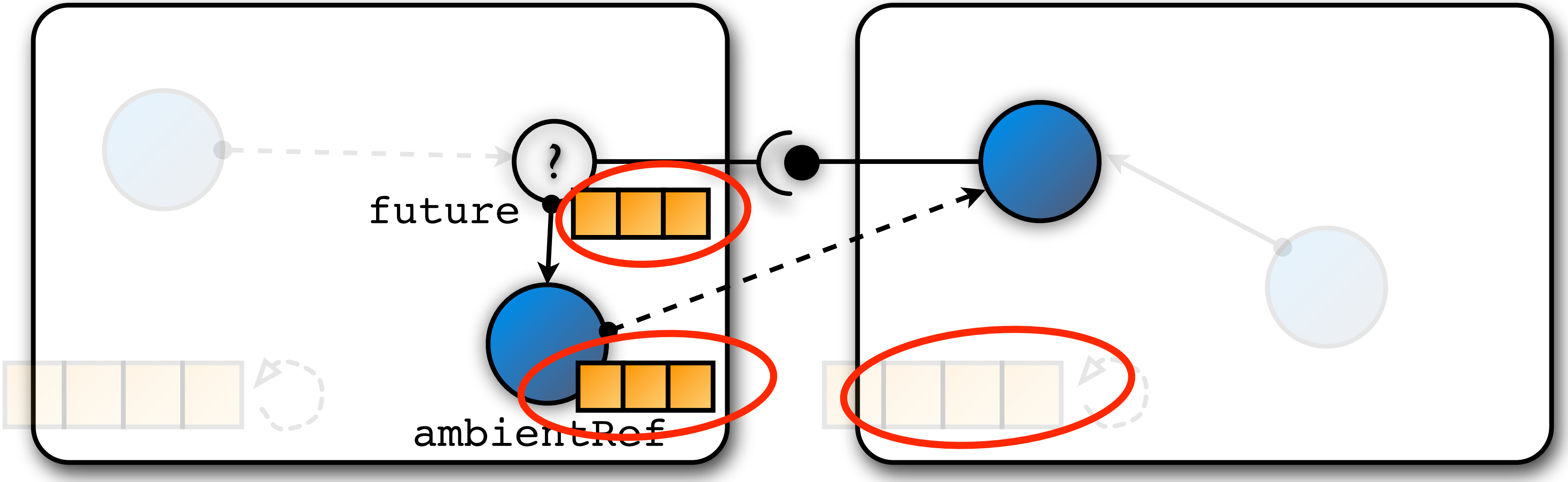

Bound

Unbound

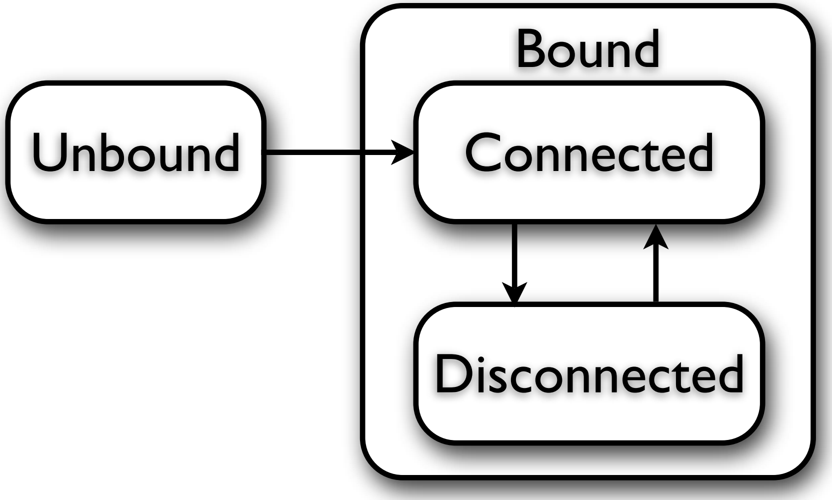
Connected

Disconnected

Evaluation



Synchronisation decoupling

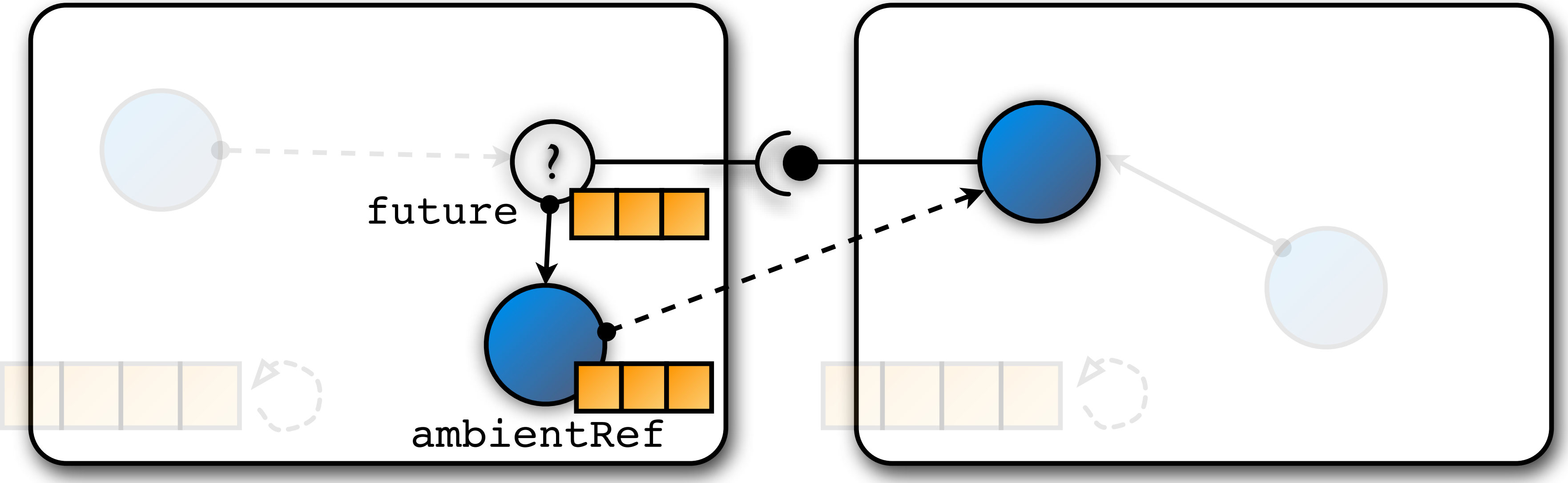

Bound

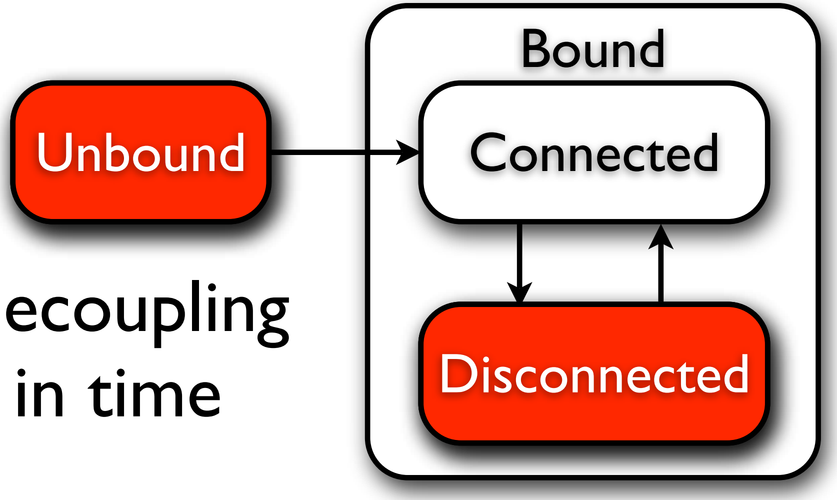
Unbound

Connected

Disconnected

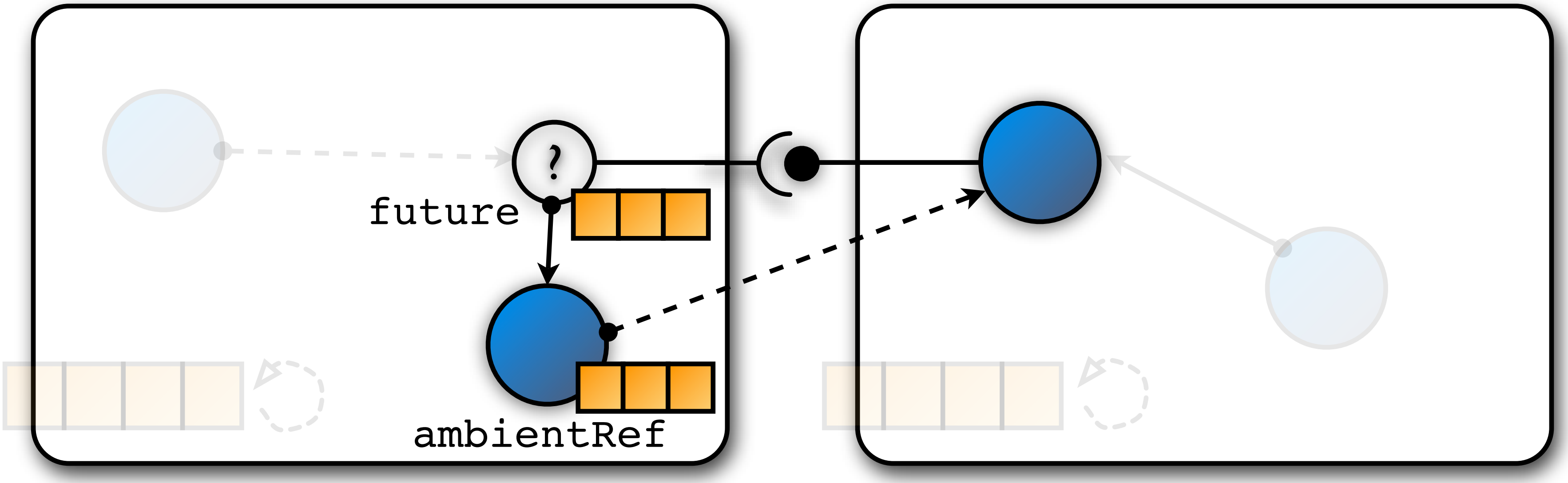
Evaluation

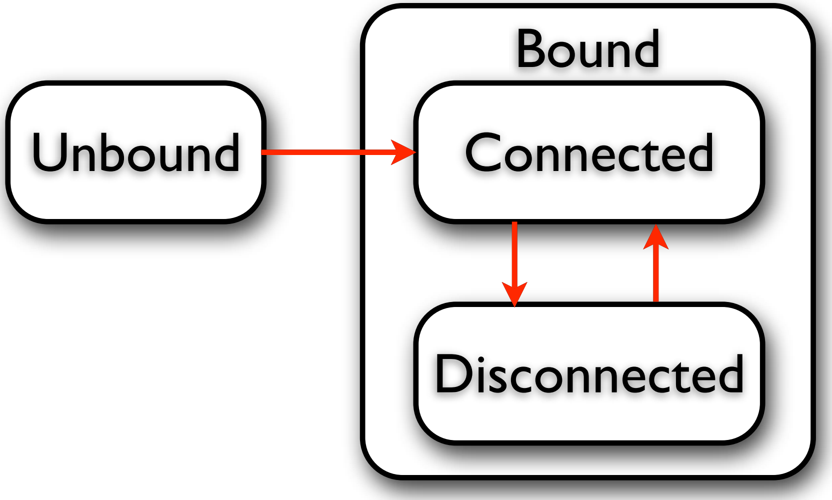




Decoupling
in time

Evaluation





Bound

Unbound

Connected

Disconnected

Connection awareness

Future work

- Integrate leasing with ambient references

```
when: ambientRef expired: {
   ...
}
```

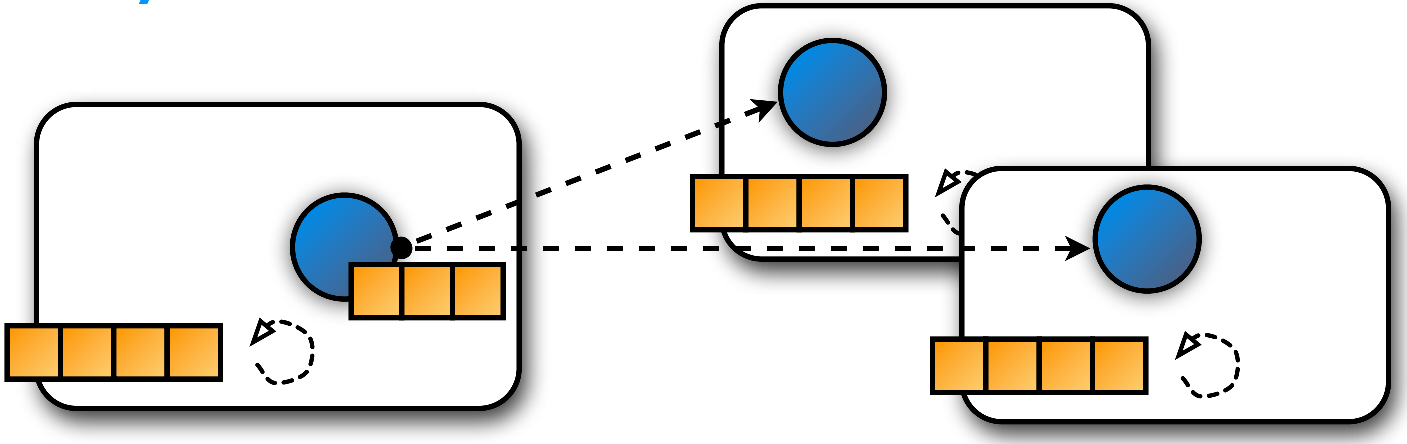
- Ambient references that support many-to-many collaborations

Future work

- Integrate leasing with ambient references

```
when: ambientRef expired: {
   ...
}
```

- Ambient references that support many-to-many collaborations



Conclusion

- Mobile networks necessitate loose coupling
- Ambient reference = loosely coupled remote object reference
  - Anonymous discovery of referent
  - Tolerates network failures by default
- Concrete implementation in AmbientTalk

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