Reflective Programming in AmbientTalk

Mirrors and Mirages

Elisa Gonzalez Boix
egonzale@vub.ac.be

AMBIENTTALK

Software Languages.Lab

Vrije
Universiteit
Brussel

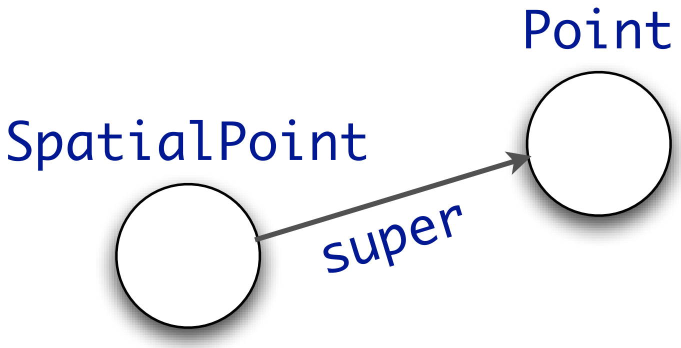
Mirror-based Reflection

Based on Self's mirrors [Bracha and Ungar04]

- **Stratification:** base- and meta-level behaviour are separated.
- **Encapsulation:** mirrors encapsulate meta-level behaviour.
- **Ontological Correspondence:** meta-level is expressed using base-level concepts.

AmbientTalk Objects

```
def Point := object: {
  def x := 0;
  def y := 0;
  def init(anX, aY) {
    x := anX;
    y := aY;
  };
  def +(other) {
    self.new(x + other.x, y + other.y) }
}

def SpatialPoint := extend: Point with:{
  def z := 0;
}
```



AmbientTalk Objects

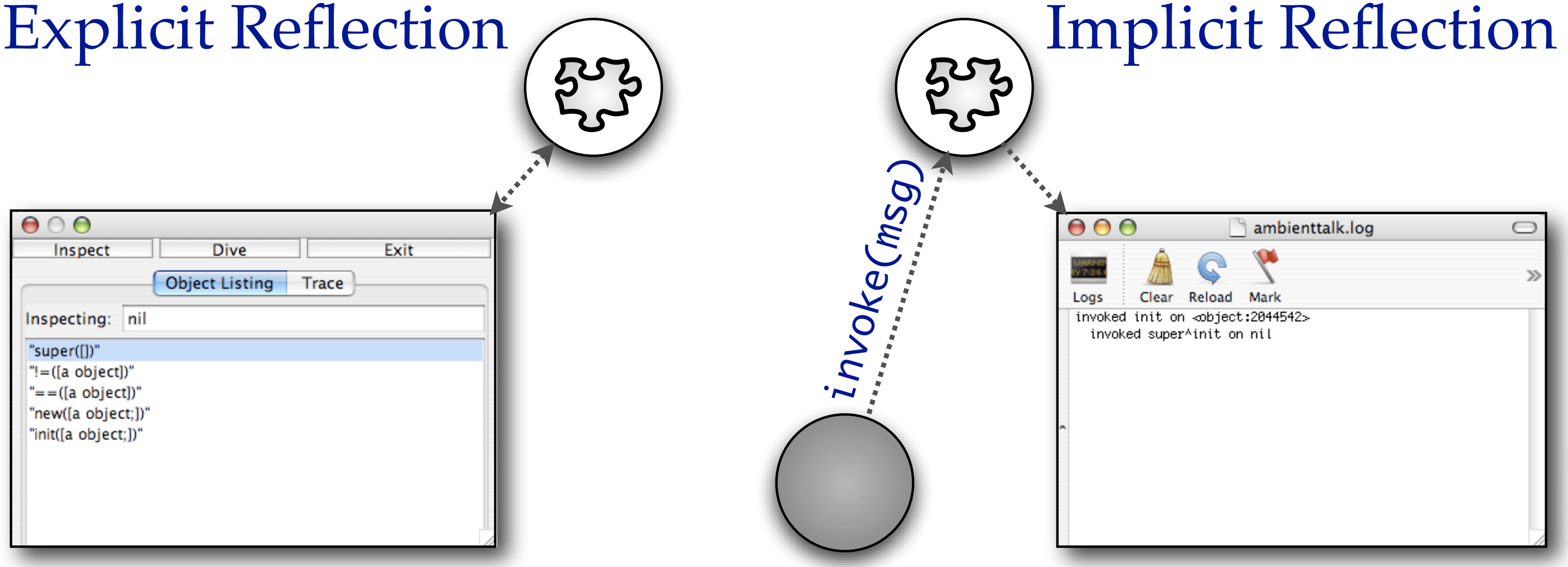
- Every AmbientTalk object understands:

```
==(obj)

new(@initargs)

init(@initargs)

super
```

Reflection Types

Explicit Reflection

Implicit Reflection



Inspecting: nil

```
"super([])"
"!=([a object])"
"==([a object])"
"new([a object;])"
"init([a object;])"
```

```
invoked init on <object:2044542>
  invoked super^init on nil
```

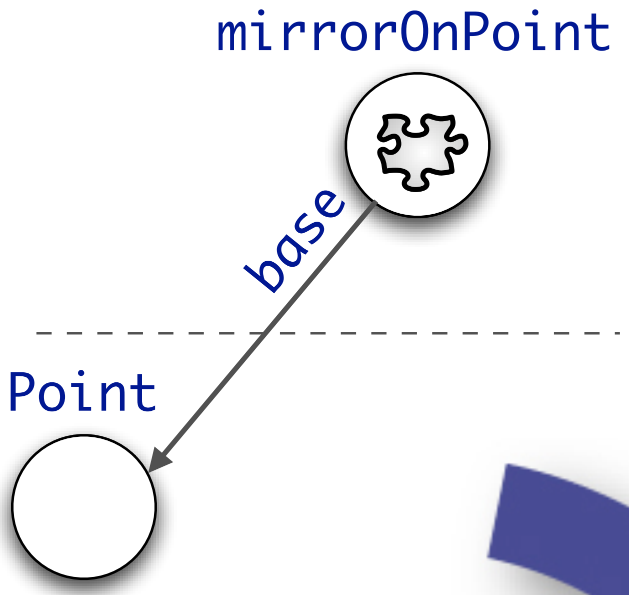
`invoke(msg)`

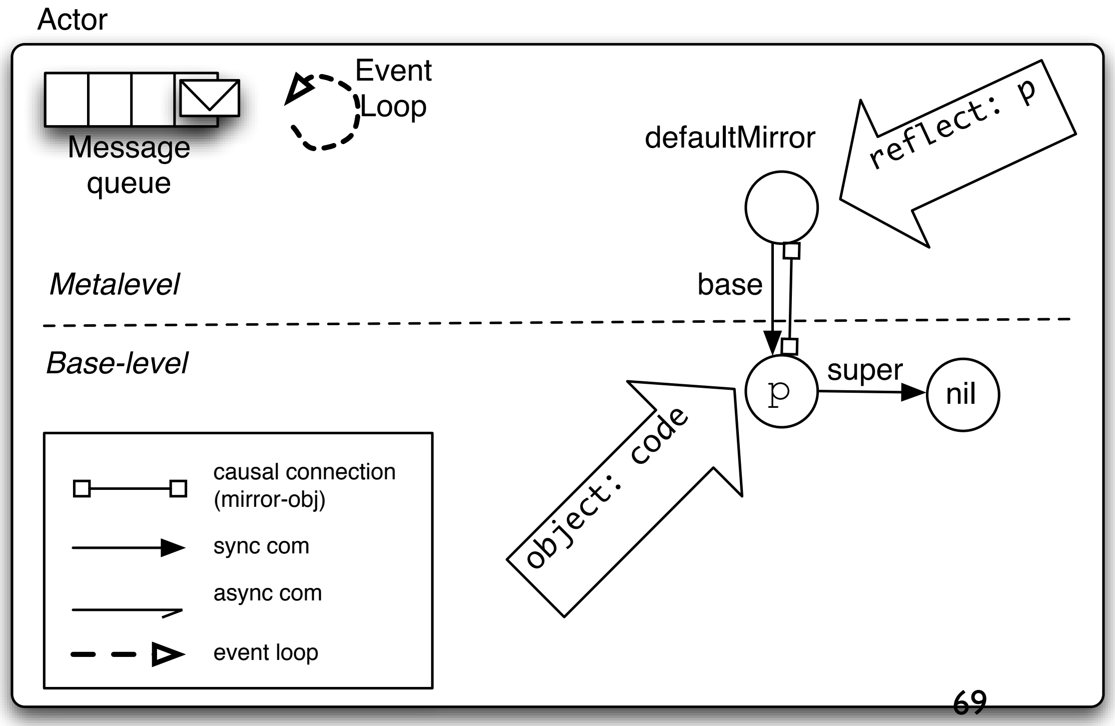
`interpreter`

Explicit Reflection

Mirrors on Objects

```
def mirrorOnP := (reflect: Point);
```



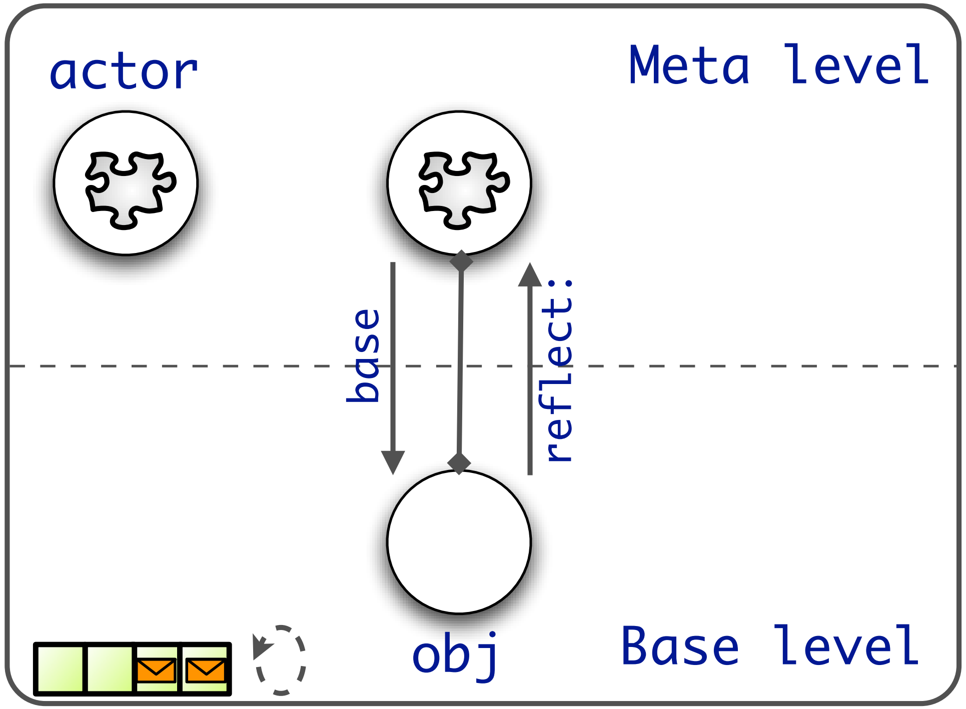


Mirrors on Objects

- Mirrors support introspection, invocation and self-modification:

```
// introspection: list all slots of an object
mirrorOnP.listSlots().map: { |slot| slot.name };

// invocation: reflectively access the contents of a slot
mirrorOnP.grabSlot(‘x);
mirrorOnP.grabSlot(‘x:=);

// invocation: reflectively invoke a method
mirrorOnP.invoke(p, createInvocation(‘distanceToOrigin, []));

// self-modification: add a slot to an object
def [accessor, mutator] := createFieldSlot(‘z, 0);
mirrorOnP.addSlot(accessor);
```

Mirrors on Actors



- Reifies the event loop.
- Reifies inter-object operations (e.g. creation and sending of asynchronous messages)

Mirrors on Actors

- Mirrors on Actors support introspection and modifying an actor’s mailbox

```
def retractMessagesMatching: selector {

  def mailbox := reflectOnActor().listIncomingLetters();
  mailbox := from: mailbox retain: { |letter|
    letter.message.selector == selector
  };
  mailbox.each: { |letter|
   letter.cancel()
  };
  mailbox;
};
```

Implicit Reflection

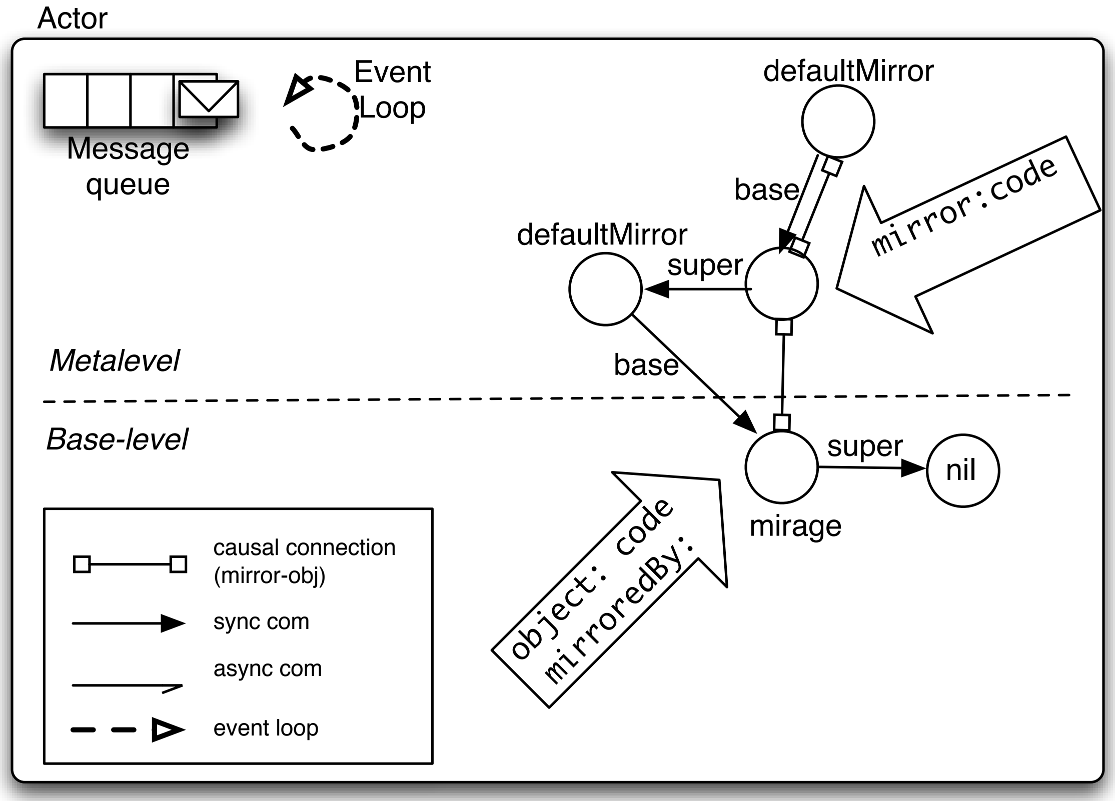
Mirages

- Only mirages support full intercession:

```
def Point := object: {
 ...
} mirroredBy: ( extend: defaultMirror with:{
  def invoke(rcv,sel,arg) {
    log(“invoked ” + sel +
        “ on ” + self.base);
    super.invoke(rcv,sel,arg);
  }
});
```

Mirages

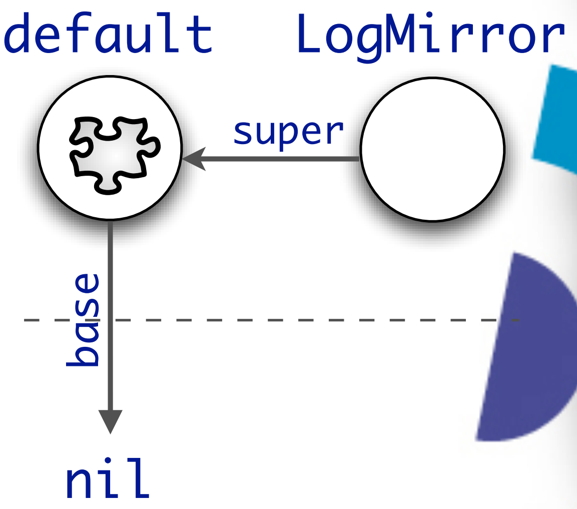
- Only mirages support full intercession:



Mirror Prototypes

- Complete MOP implementation (*).
- Not causally connected

```
def LogMirror := extend: actor.defaultMirror with: {
 def invoke(rcv,sel,arg) {
    log(“invoked ” + sel +
        “ on ” + self.base);
    super.invoke(rcv,sel,arg);
  };
};
```

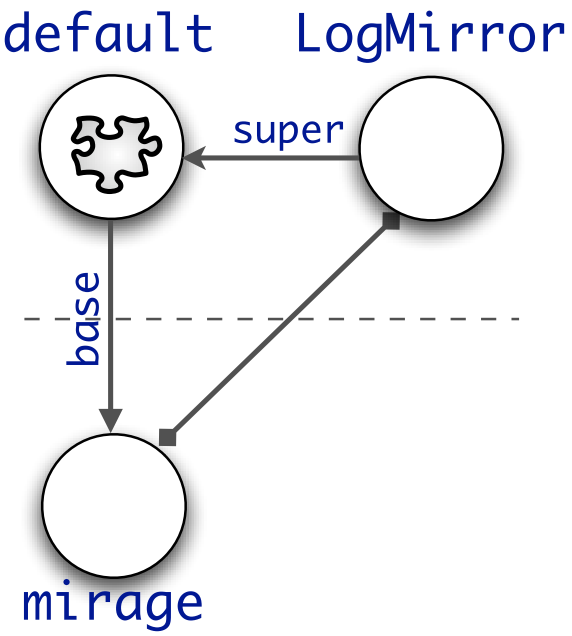


(*)default interface available at language reference in MirrorRoot.

Mirror Prototypes

- Absorbing the mirror:

```
def MiragePoint := object: { ... } mirroredBy: LogMirror;
```



Mirror initialization

```
def LogMirror :=  mirror: {
   def var;
   def init(base, val){
      //it should always first initialize defaultMirror!
      super^init(base);
      var := val;
   };
   def invoke (rcv,sel,arg) {
      ...
   };
};

def MiragePoint := object: {
...
} mirroredBy: { |base|  LogMirror.new(base, aVar) };
```

Implicit Reflection on Actors

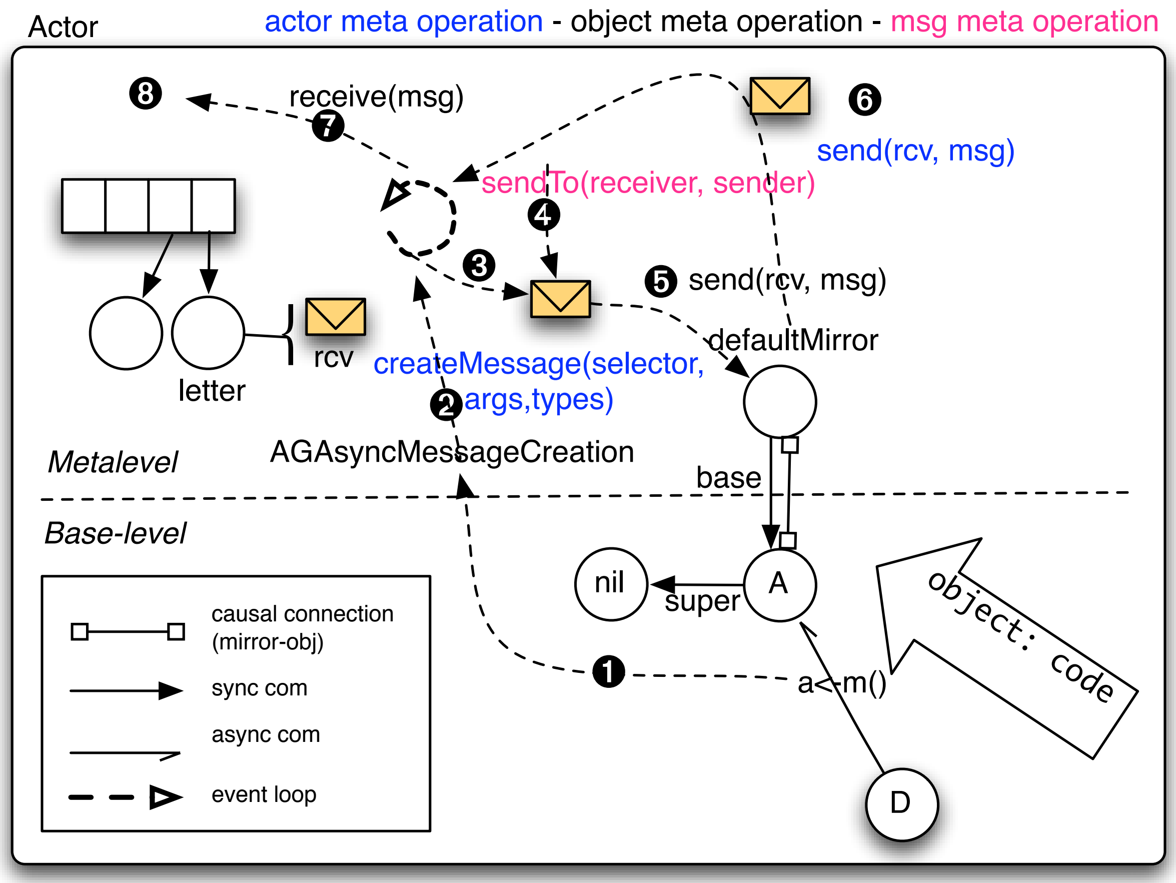
- New actor mirrors can be installed dynamically:

```
actor.install: (extend: actor with: {
  def createMirror(onObj) {
    extend: super.createMirror(onObj) with: {
      def invoke(rcv,sel,args){
        system.println(“invoked “ + sel);
        super.invoke(rcv,sel,args);
      }
    }
  }
})
```

AmbientTalk Meta Level

- MOP divided into series of independent protocols:
  - **message invocation protocol**
  - object marshaling protocol
  - slot access and modification protocol
  - structural access protocol
  - object instantiation protocol
  - relational testing protocol
  - type tag protocol
  - evaluation protocol

Message Invocation Protocol



Actor

actor meta operation - object meta operation - msg meta operation

❽ receive(msg)

❼

❻ send(rcv, msg)

sendTo(receiver, sender)

❹

❸

❺ send(rcv, msg)

rcv

letter

createMessage(selector, args,types)

❷

defaultMirror

AGAsyncMessageCreation

Metalevel

base

Base-level

nil

super

A

❶ a<-m()

D

object: code

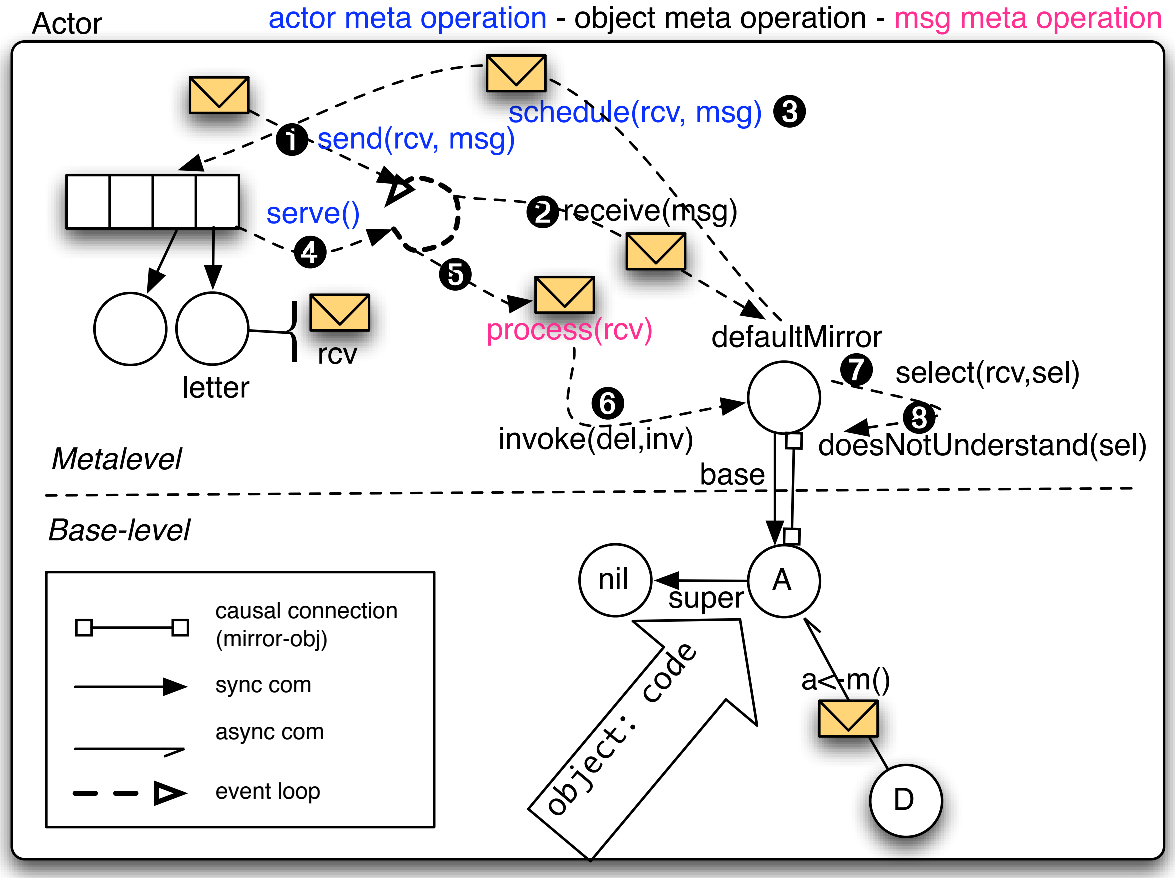
causal connection (mirror-obj)

sync com

async com

event loop

Message Invocation Protocol



Mirages Applied: Futures

```
def FutureMirror := extend: actor.defaultMirror with: {
  def state := UNRESOLVED;
  def resolvedValue := nil;
  def inbox := [];
  def invoke(rcv,sel,args) {
    raise: IllegalOperation.new(
        "Cannot synchronously invoke methods on a future");
  };
  def receive(msg) {
    if: (state == RESOLVED) then: {
      resolvedValue<+msg;
    } else: {
      inbox := inbox + [msg];
    };
  };
```

Future Mirror

```
def FutureMirror := extend: actor.defaultMirror with: {
  def subscribers := [];
  def subscribe(closure) {
    if: (state == UNRESOLVED) then: {
      subscribers := subscribers + [closure];
    } else: {
      closure<-apply([resolvedValue])
    };
  };
  def resolve(value) {
    if: (state == UNRESOLVED) then: {
      state := RESOLVED;
      resolvedValue := value;
      inbox.each: { |msg| value<+ msg };
      subscribers.each: { |clo| clo<-apply([value]) };
}}};
```

Integration with message sending

```
actor.install: (extend: actor with: {
  def createMessage(sel,args) {
    def msg := super.createMessage(sel,args);
    extend: msg with: {
      def future := makeFuture();
      def process(receiver) {
        def result := super.process(receiver);
        (reflect: future)<-resolve(result);
        result;
  }}};

  def send(message) {
    super.send(message);
    message.future;
}}
```