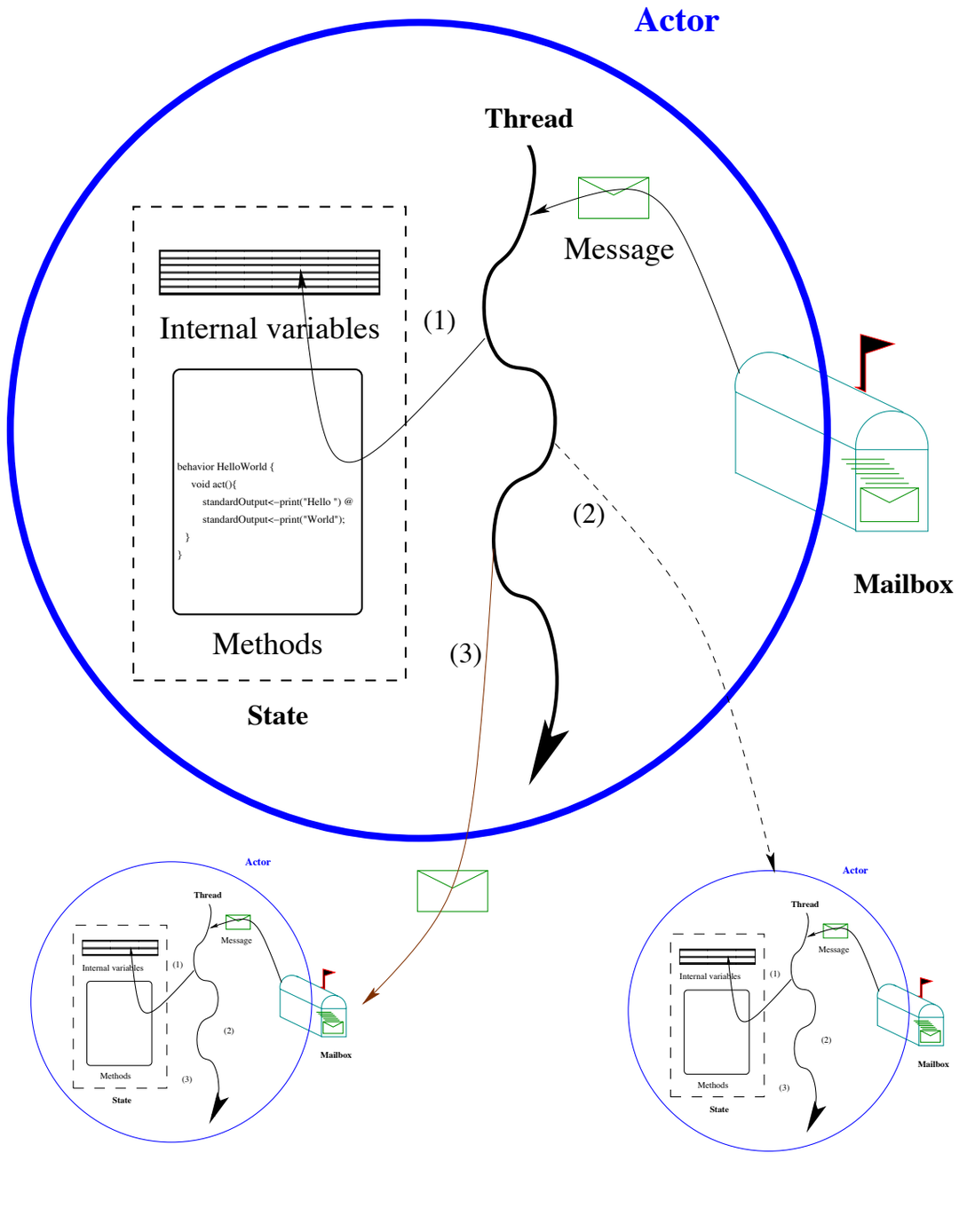
Actors in Java

Actors 2


Actor
Thread
Message
Internal variables
(1)
behavior HelloWorld {
void act(){
standardOutput<–print("Hello ") @
standardOutput<–print("World");
}
}
Methods
(2)
Mailbox
(3)
State
Actor
Thread
Message
Internal variables
(1)
(2)
Methods
(3)
State
Mailbox
Actor
Thread
Message
Internal variables
(1)
(2)
Methods
(3)
State
Mailbox

Actors 3

- Combine state and behavior (a thread of control)
- Encapsulate state (all fields/methods are private)
- Only act in response to asynchronous messages (when you send a message it doesn't block for a response)
- In response to a message an actor can send other messages, change its state and/or create new actors
- Process messages from its mailbox in first-in-first-out order

- No shared memory means no memory inconsistency
- No blocking methods/message sends means no deadlocks
- Only issue is you have to do all communication carefully via message passing (need to make sure you don't send an object reference that two actors could modify concurrently)

Actors in Java

```
public class MyActor {
      private LinkedList<Message> mailbox = new LinkedList<Message>();

      public synchronized void putMessage(Message message) {
         mailbox.addLast(message);
         notify();
      }
      private synchronized Message getMessage() {
         if (mailbox.isEmpty()) {
            try {
               wait();
            } catch (InterruptedException e) { ... }
         }
         return mailbox.removeFirst();
      }
      private doSomething(Message message) {
         //process messages here
      }

      public void run() {
         while (true) {
            Message message = getMessage();
            doSomething(message);
         }
      }
}
```

The general idea is that your threads will place messages in the mailbox on the assumption that the actor will process them later.

The actor can let other actors know it has finished processing the message by putting other messages in their mailbox.

You only have one synchronization point to ever worry about (placing the messages in the mailbox).

Actors in Java

```
public class MyActor {
      …
      private handleSearchQuery(SearchQueryMessage sqm) {
         //process message here
         …
      }

      private handleSearchResponse(SearchQueryMessage sqm) {
         //process message here
         …
      }

      private handleFileDownload(SearchQueryMessage sqm) {
         //process message here
         …
      }

      public void run() {
         while (true) {
            Message message = getMessage();
            if (message instanceof SearchQueryMessage) {
               handleSearchQuery((SearchQueryMessage)message);
            } else if (message instanceof SearchResponseMessage) {
                handleSearchResponse((SearchResponseMessage)message);
            } else if (message instanceof FileDownloadMessage) {
                handleFileDownload((FileDownloadMessage)message);
            }
         }
      }
}
```

You can expand on this easily with the instanceof operator and multiple functions.

So what you can do is have your mailbox accept messages of multiple types and process them each individually.

The great part about this is that you know you'll only be processing one message at a type so you don't need to worry about synchronization issues.

Actors in Java

```
public class MyActor {
      …
      private handleSearchQuery(SearchQueryMessage sqm) {
         //process message here
         …
      }

      private handleSearchResponse(SearchQueryMessage sqm) {
         //process message here
         …
      }

      private handleFileDownload(SearchQueryMessage sqm) {
         //process message here
         …
      }

      public void run() {
         while (true) {
            Message message = getMessage();
            if (message instanceof SearchQueryMessage) {
               handleSearchQuery((SearchQueryMessage)message);
            } else if (message instanceof SearchResponseMessage) {
                handleSearchResponse((SearchResponseMessage)message);
            } else if (message instanceof FileDownloadMessage) {
                handleFileDownload((FileDownloadMessage)message);
            }
         }
      }
}
```

The one thing you do need to worry about is that depending on how threads access the mailbox, the messages in your mailbox might not be in the exact order you think.

For example, if two threads try to put a message in the mailbox at the same time; sometimes when you run the program one message will be processed first, while other times the other will be.

Actors

- Threads, synchronization, etc can be extremely complex (leading to many headaches and sleepless nights)
- Just as Java made memory management significantly easier than c (and c++), actors make concurrency **MUCH** easier

Actor Languages

- **Erlang -** Functional: http://ftp.sunet.se/pub/lang/erlang/index.html
- **SALSA -** Java based: http://wcl.cs.rpi.edu/salsa/ (implemented in Java)
- **Scala -** Pattern based: http://www.scala-lang.org/ (implemented in Java)