REXX / Object REXX Development

# Concurrency with Object REXX

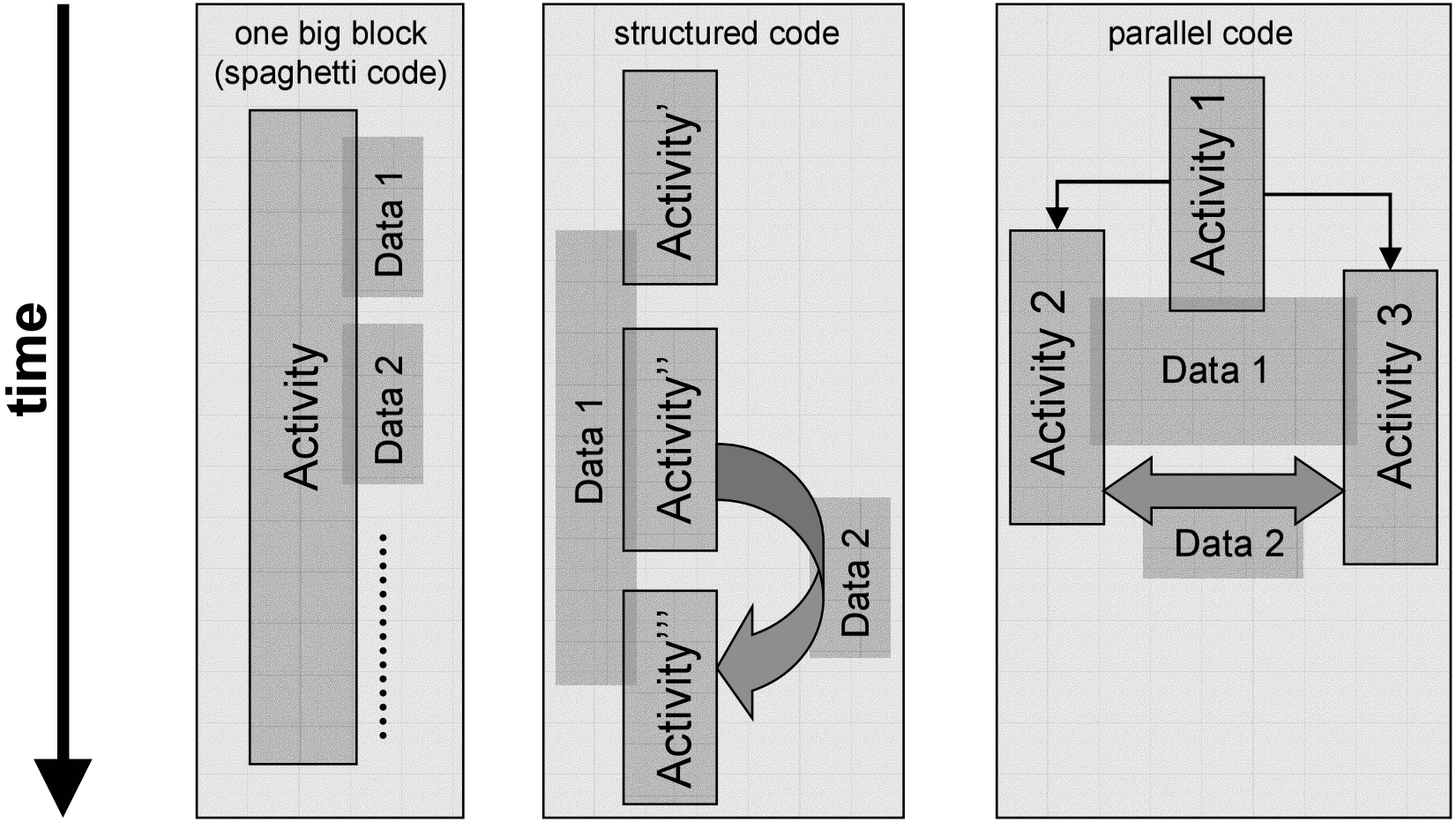
Lavrentios Servissoglou (servissoglou@de.ibm.com)
REXX Symposium 2004, Böblingen/Sindelfingen (Germany)

# Contents



# What is concurrency? (1/3)

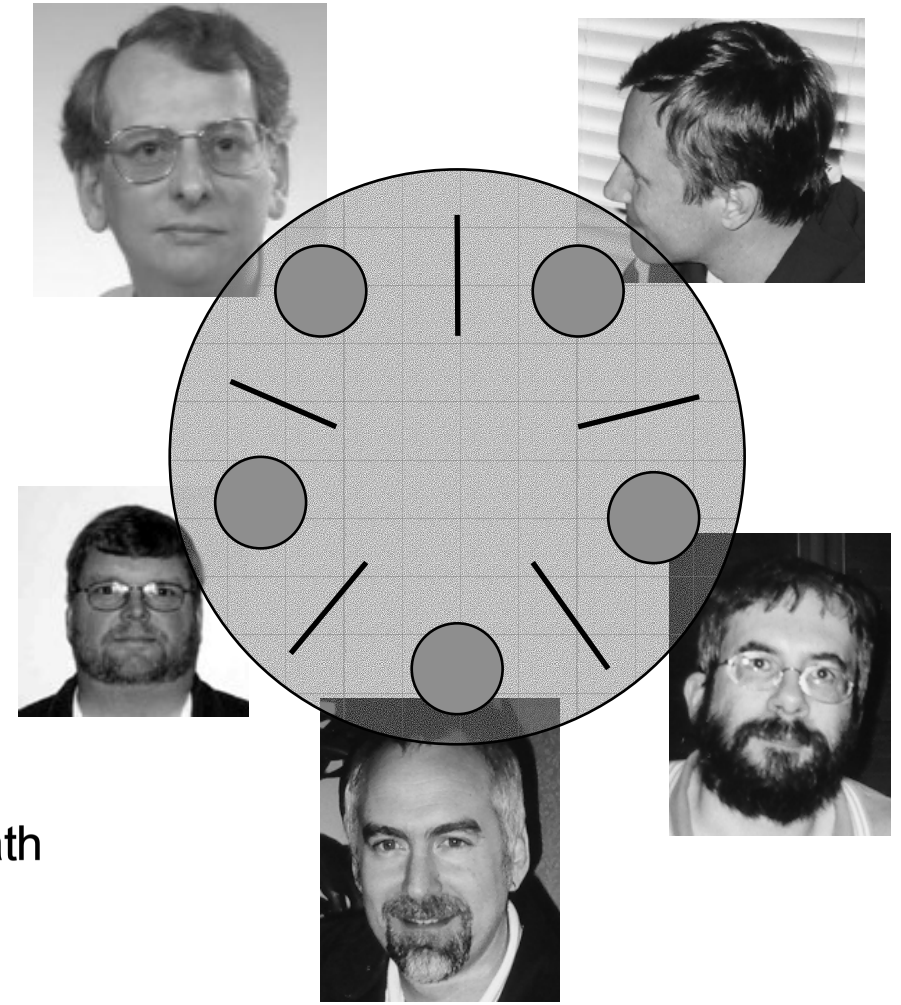


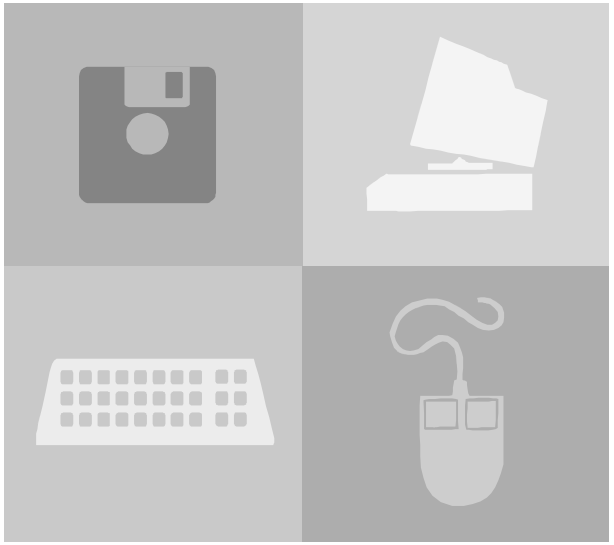
# What is concurrency? (2/3)

**The Dining-Philosophers Problem:**

- 5 philosophers thinking and eating
- Resources: 5 single chopsticks and plates with rice
- You need 2 chopsticks for eating and no chopsticks for thinking
- Arising problems:
- **Deadlock**: Every philosopher has one chopstick and is waiting for the second one
- **Starvation**: A philosopher starves to death



# What is concurrency? (3/3)



- **Pros:**
  - Increase of responsiveness
    (e.g. text mining while user types in search criteria)
  - To shorten the idle times
    between various tasks/threads
    (e.g. printing parallel to word processing)
  - Access to same data pool
    (e.g.)
- **Cons**
  - Increase of resources
    (e.g. maintenance of additional IDs)
  - Increase of complexity
    (e.g. integrity of data)
  - Execution no predictable
    (e.g. trouble with debugging)

# Object REXX (1/1)

- **Supported platforms**
  - AIX 4.1.5,  4.2.X, 4.3.X, 5.1L (32Bit)
  - Linux/386, Linux/PPC, Linux/390
  - OS/2
  - Solaris 5, 6, 7, 8 (Sun)
  - Windows 95/98(SE)/ME
  - Windows NT 3.51, 4.0 and Windows 2000/XP/2003

# Concurrency in Object REXX (0/10)

**“Conceptually, each REXX object is like a small computer with its own processor to run its methods, its memory for object and method variables, and its communication links to other objects for sending and receiving messages.**

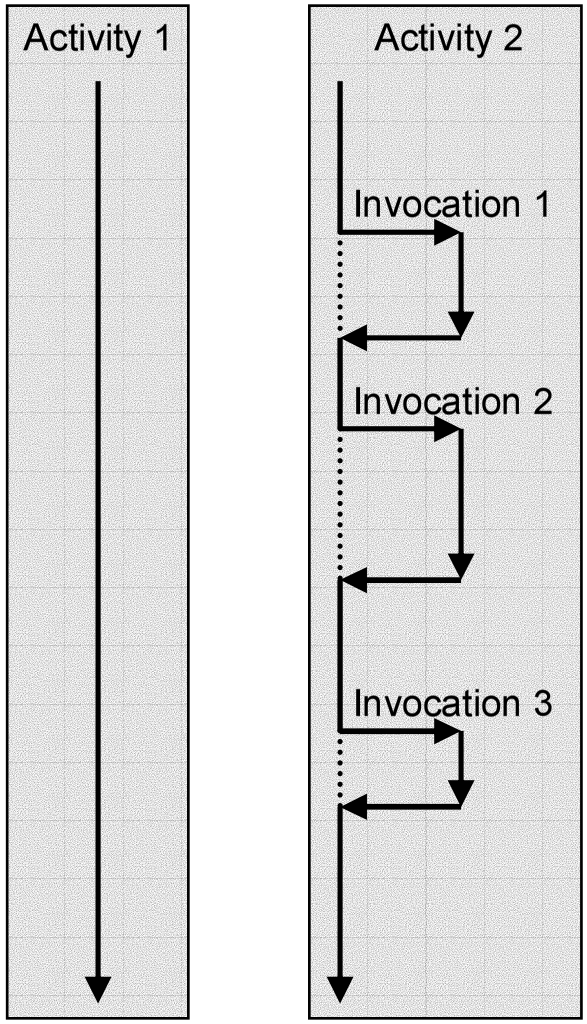
**This is object-based concurrency.**

**It lets more than one method run at the same time. Any number of objects can be active (running) at the same time, exchanging messages to communicate with, and synchronize, each other.”**

**Object REXX Reference**

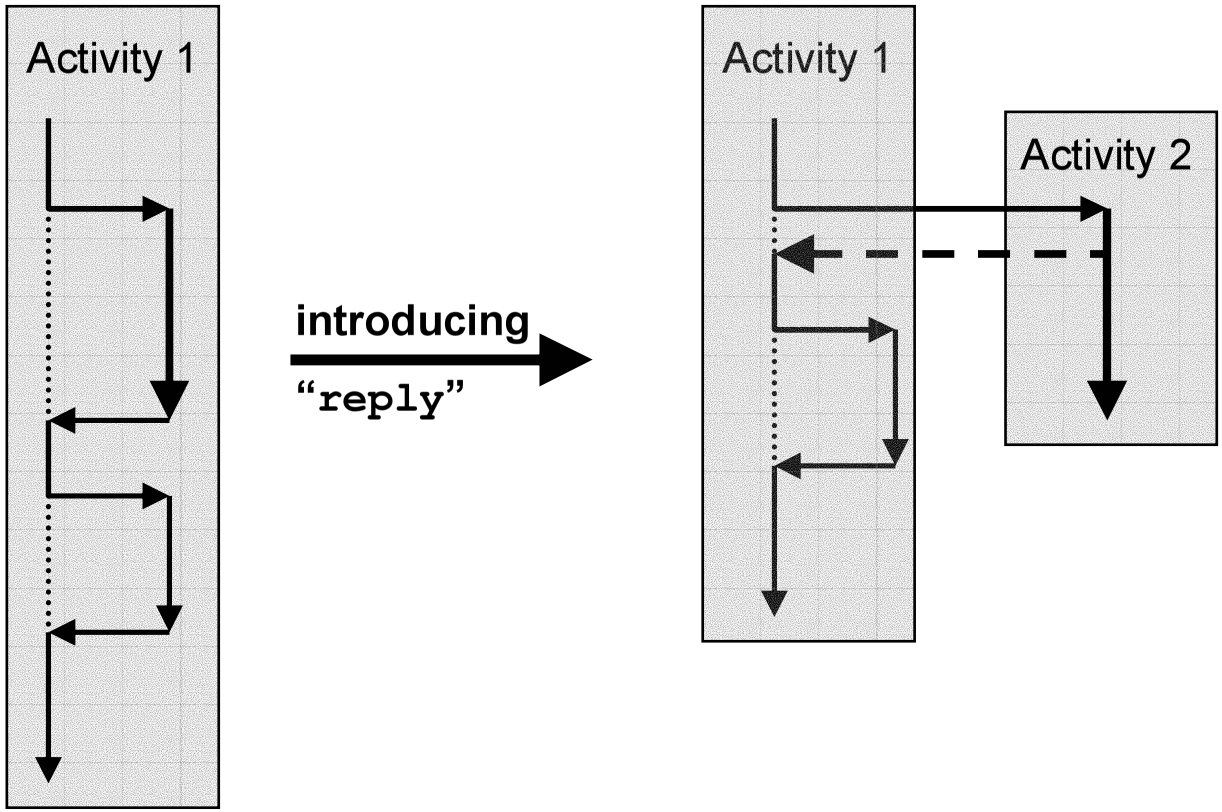
# Concurrency in Object REXX - activities (1/10)

- **Object REXX consists of activities**
- **Each activity has a stack of invocations**
- **Invocation can be an internal function or subroutine call, an external function or subroutine call, an `INTERPRET` instruction, or a message invocation.**



# Concurrency in Object REXX (2/10)

- **Create new activities by introducing "`reply`" with early results**



# Concurrency in Object REXX (3/10)

```
object = .sample~new
say object~output( 10, "Tango" )
do 10
   say "Paso doble"
end
exit

::CLASS sample
::METHOD output
   use arg loop, message
   reply "Object is working"
   do loop
      say message
   end
```

```
Tango
Object is working
Tango
Paso doble
Tango
Paso doble
Tango
Paso doble
Tango
Paso doble
Tango
Paso doble
Tango
Paso doble
Tango
Paso doble
Tango
Paso doble
Tango
Paso doble
Tango
Paso doble
Paso doble
```

# Concurrency in Object REXX (4/10)

- **"`reply`" enables Inter-Object concurrency**

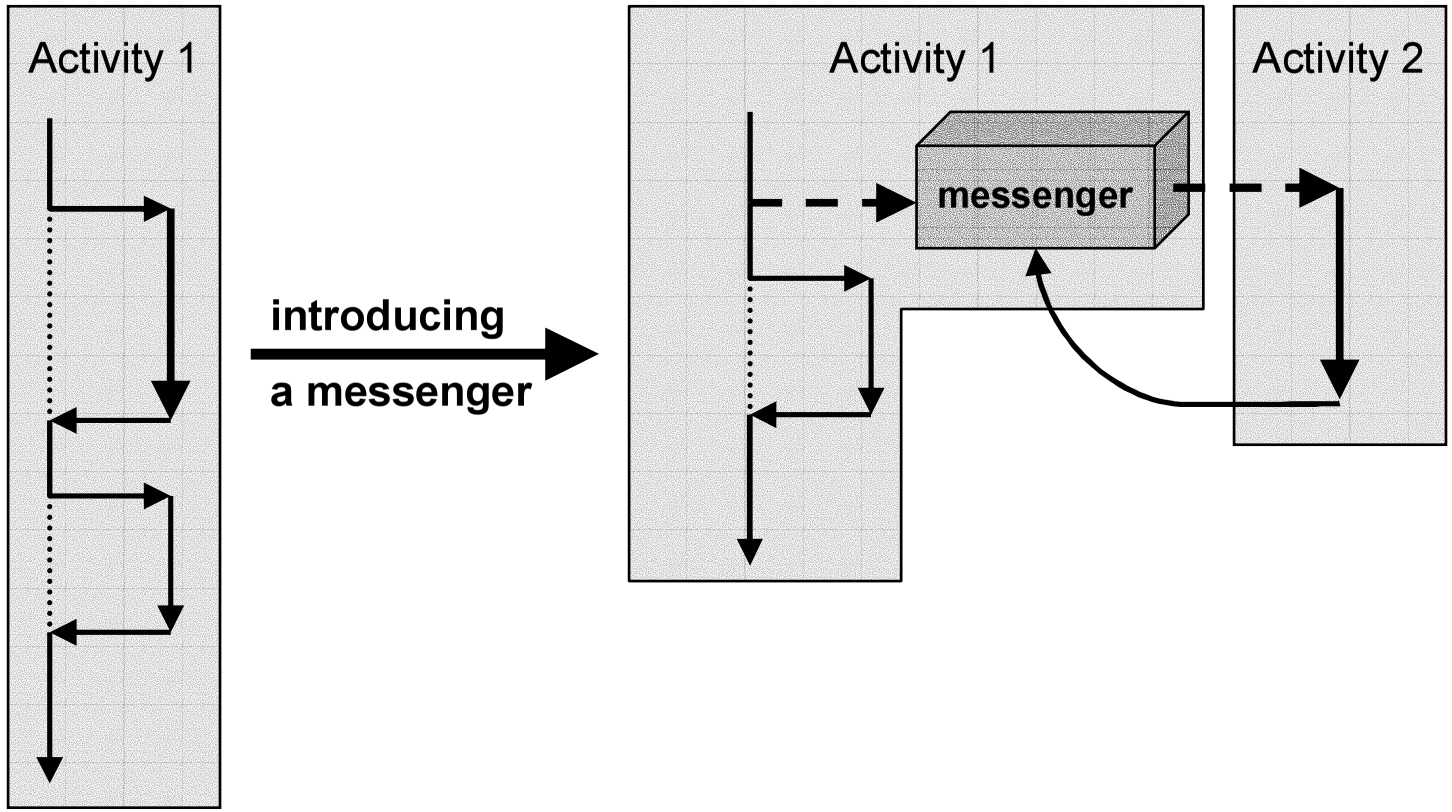
```
object1 = .sample~new
object2 = .sample~new
say object1~output( 10, "object1")
say object2~output( 10, "object2")
say "That's all"
exit

::CLASS sample
::METHOD output
   use arg loop, message
   reply "Loop" message loop "times"
   do loop
      say message
   end
```

```
Loop object1 10 times
object 1
object 2
Loop object2 10 times
object 1
object 2
That's all
object 1
object 2
object 1
object 2
object 1
object 2
object 1
object 2
object 1
object 2
object 1
object 2
object 1
object 2
object 1
object 2
```

# Concurrency in Object REXX (5/10)

- **Create new activities by delegating a new (intermediate) object as messenger**



# Concurrency in Object REXX (6/10)

```
object = .sample~new
messageObject = object~start( "OUTPUT", 10, "Tango" )
do 10
   say "Paso doble"
end
say messageObject~result
exit

::CLASS sample
::METHOD output
   use arg loop, message
   do loop
      say message
   end
   return "Output has finished"
```

non blocking

blocking

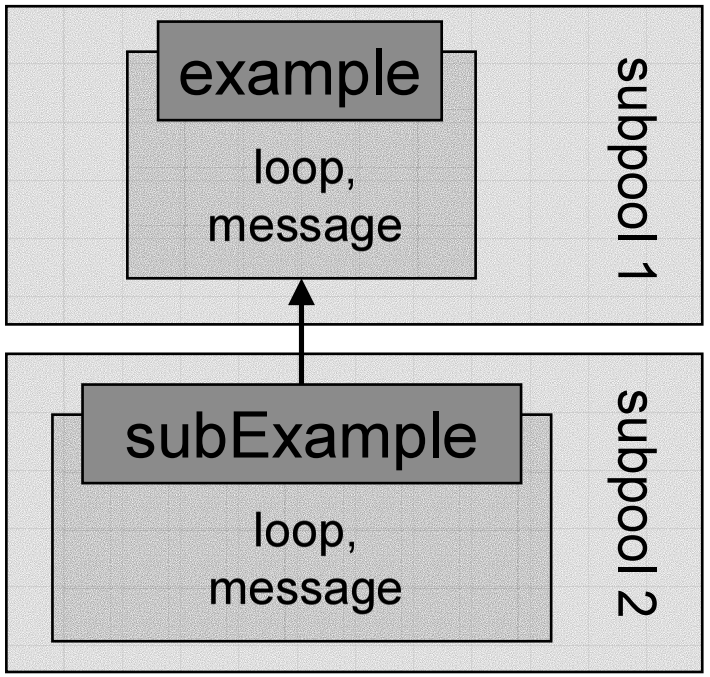
```
Paso doble
Tango
Paso doble
Tango
Paso doble
Tango
Paso doble
Tango
Paso doble
Tango
Paso doble
Tango
Paso doble
Tango
Paso doble
Tango
Paso doble
Tango
Paso doble
Tango
Paso doble
Tango
Output has finished
```

# Concurrency in Object REXX – exclusive access (7/10)

- **Object REXX provides safe data pool by default**
- **Scope of object exists as set of subpools**
- **Activity in subpool locks this subpool against other activities – leads to sequential executing**

```
::class example
::method repeat
use arg loop, message

::class subExample subclass example
::method repeater
use arg loop, message
```



# Concurrency in Object REXX (8/10)

- **There exists various options/methods to “break” the exclusive access to the object variable pool**
- **Unconditional concurrency:**
  - **`SETUNGUARDED`** method
  - **`UNGUARDED`** option
- **Conditional concurrency:**
  - **`GUARD ON (WHEN `*`expression`*`)`**
    and
    **`GUARD OFF (WHEN) `*`expression`*`)`**

# Concurrency in Object REXX (9/10)

```
object = .sample~new
say object~output1( 10, "1st call" )
say object~output2( 10, "2nd call" )
say "That's all"
exit

::CLASS sample
::METHOD output1 UNGUARDED
   use arg loop, message
   reply "output1: Loop" message loop "times"
   do loop
      say message
   end

::METHOD output2 UNGUARDED
   use arg loop, message
   reply "output2: Loop" message loop "times"
   do loop
      say message
   end
```

```
1st call
output1: Loop 1st call 10 times
1st call
output2: Loop 2nd call 10 times
1st call
That's all
2nd call
1st call
2nd call
1st call
2nd call
1st call
2nd call
1st call
2nd call
1st call
2nd call
1st call
2nd call
1st call
2nd call
2nd call
2nd call
```

# Concurrency in Object REXX (10/10)

```
object = .sample~new
say object~output1( 500000, "1st call" )
object~output2( 100000, "2nd call" )
say "That's all"
exit

::CLASS sample
::METHOD output1 UNGUARDED        /* This method may run parallel with other methods */
   expose loopValue
   use arg loop, message
   loopValue = 0
   reply "output1: Loop" message loop "times"  /* early reply: creating new activity */
   do loop
      loopValue = loopValue + 1
      if loopValue//1000=0 then say loopValue message  /* print every 1000th message */
   end

::METHOD output2
   expose loopValue
   use arg loop, message
   output2loop = 0
   guard on when loopValue>loop   /* start loop when loopValue is greater than loop */
   do loop
     output2loop = output2loop + 1
     if output2loop//1000=0 then say output2loop message
   end
```

# Summary

- **Object REXX provides a simple and mighty interface for multithreading**
- **Object REXX provides the tools to safe critical data and to synchronize data access**
- **Object REXX may not prevent programmers to get a deadlock or starvation**