TASKS AND THREADS

CONCURRENCY IN ADA AND JAVA

Jürgen F H Winkler

Institute of Computer Science

Friedrich Schiller University, Jena

Munich, 2001.Mar.28

OVERVIEW

Terminology

Actor in Ada

Actor in Java

Process in Ada

Process in Java

Access Coordination in Ada

Access Coordination in Java

Timeout in Ada

Timeout in Java

Summary

TERMINOLOGY

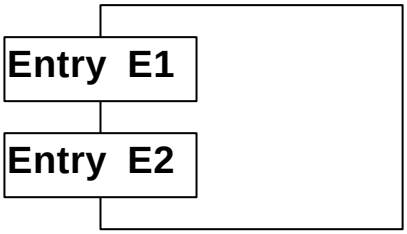
Clearly distinguish between

- **static entities in the program (text) and**
- **dynamic entities during program execution**

	static	dynamic
Ada	**task (object)**	**process**
Java	**Thread*-object**	**process**
neutral	**actor**	**process**

ACTOR IN ADA: TASK-OBJECT

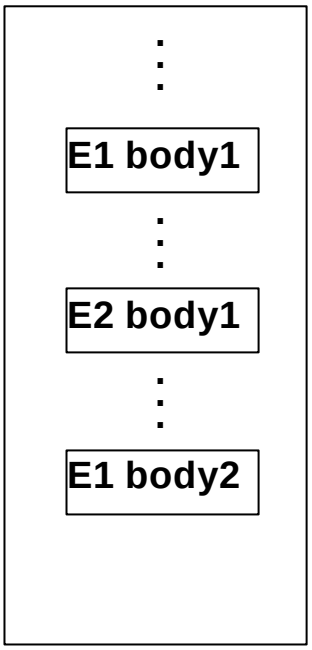
TASK TYPE MyTaskType Spec



statically

entry similar to procedure

TASK BODY MyTaskType

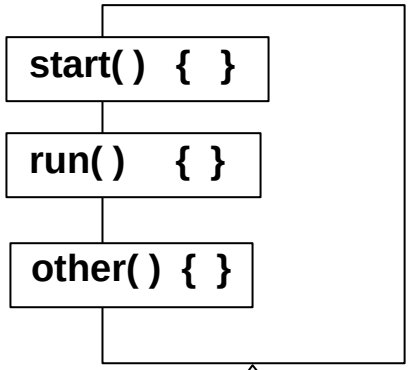


Entry Bodies are embedded in the statement sequence of the task body

=> wide variety of behavior possible

ACTOR IN JAVA: THREAD*-OBJECT

class Thread



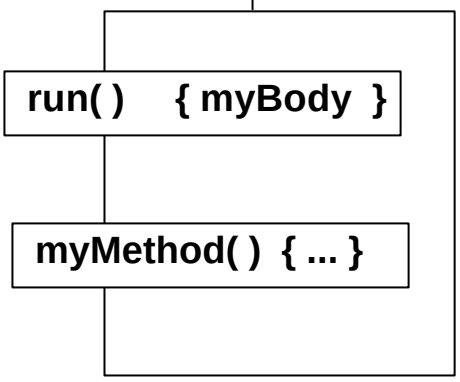
two special methods

public void start ( )

public void run ( )

and other methods

class MyThreadType Spec+Body



run( ) is the essential method

is usually reimplemented in a derived class

run( ) corresponds to the task body in Ada

PROCESS IN ADA : a single execution of some task type object

statically declared

```
   T1:  MyTaskType;
   T2:  YourTaskType;
BEGIN  -- start T1, T2
   . . .
```

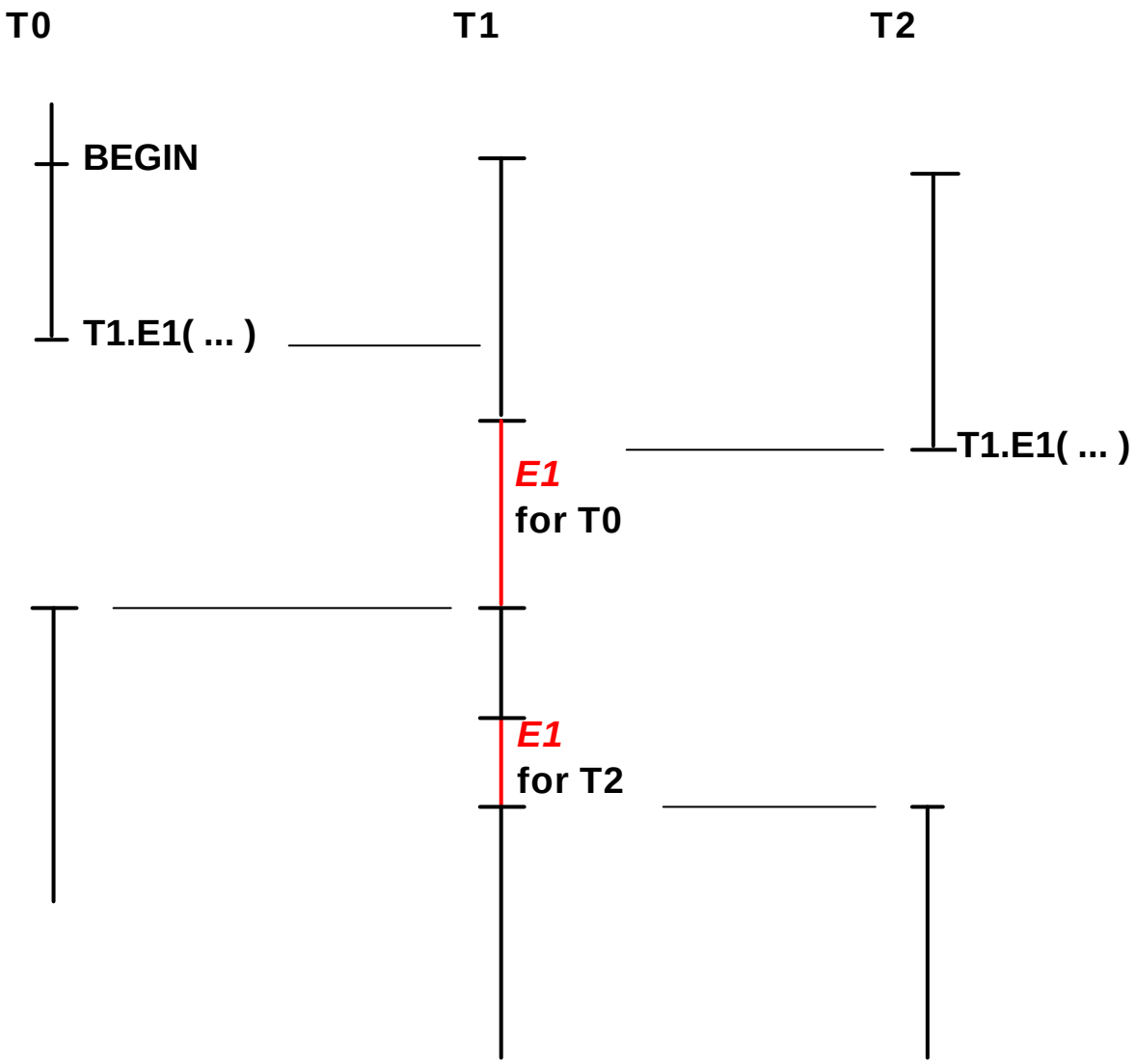
dynamic object

```
TRef1 := new  MyTaskType;
        -- create and start
. . .
```

direct interaction

between processes via

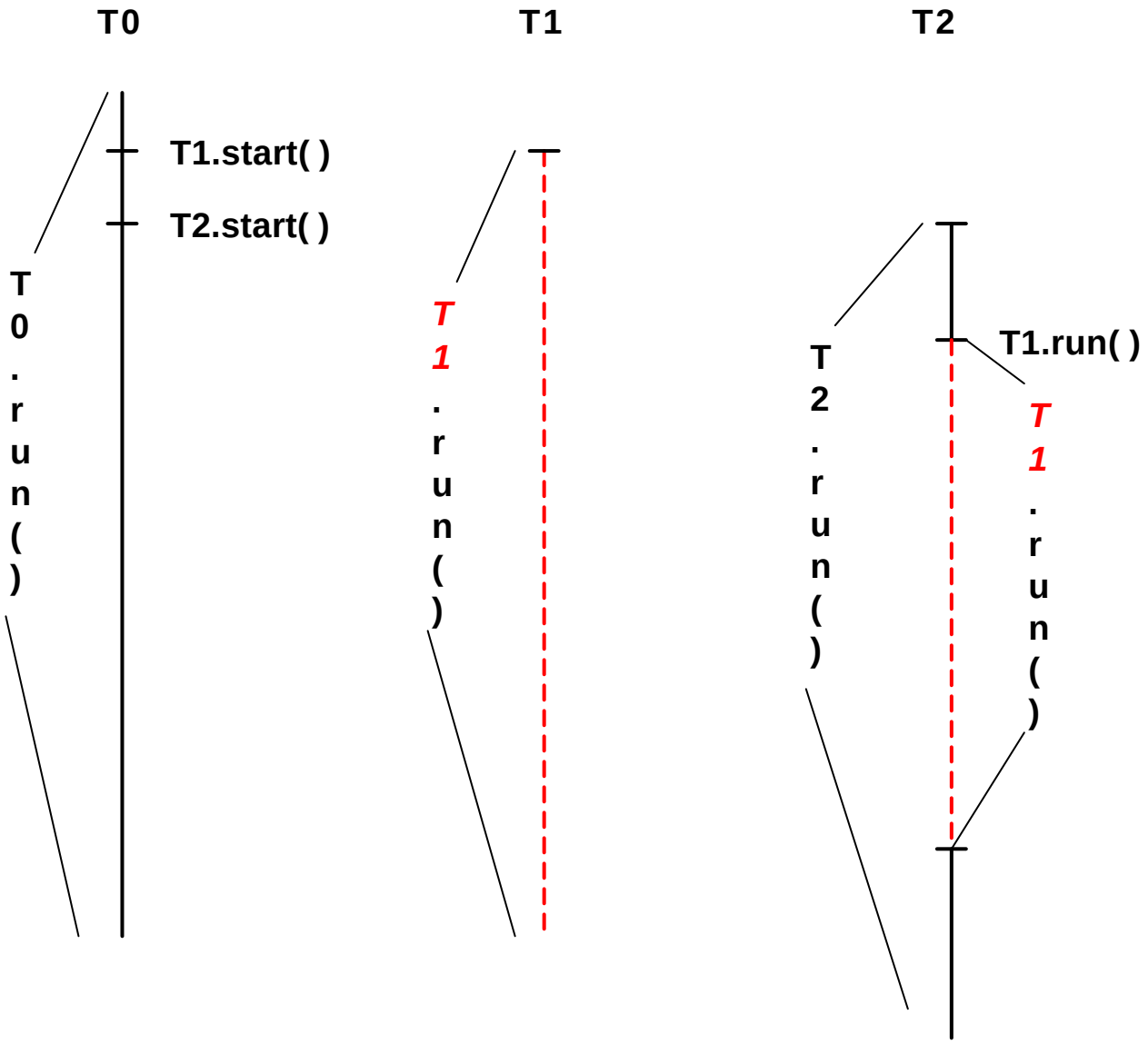
entry call / rendezvous


T0
T1
T2
BEGIN
T1.E1(...)
T1.E1(...)
E1
for T0
E1
for T2

PROCESS IN JAVA: EXECUTION OF A THREAD*-OBJECT

```
MyThreadType T1 = new MyThreadType( );      // creation

YourThreadType T2 = new YourThreadType( );

. . .

T1.start( );        // start and execute T1.run( ) concurrently

T2.start( );
```

NO direct interaction
between processes
i.e. between their
run( ) methods

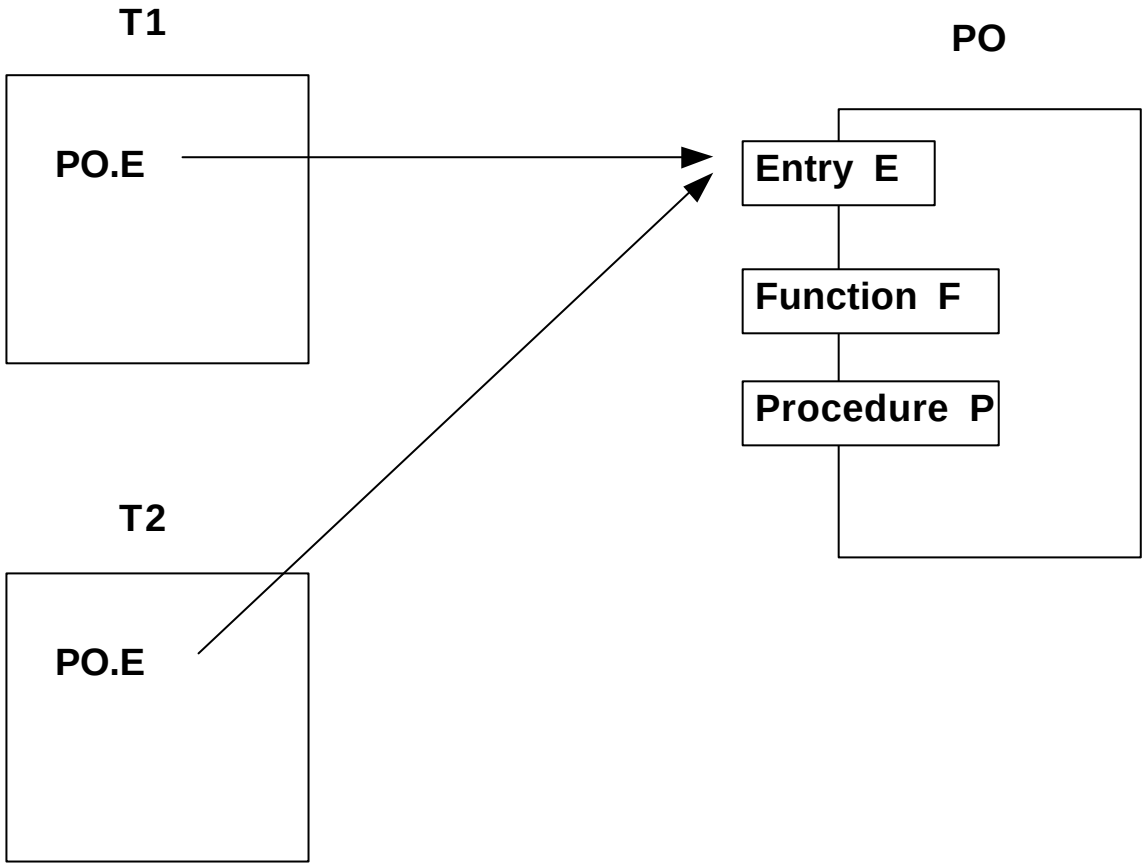


If T2 executes

T1.run( );

the method T1.run( ) is just executed by T2 i.e. concurrently to the execution initiated by T1.start

ACCESS COORDINATION IN ADA : PROTECTED TYPE / OBJECT



exclusive RW-access
+ barrier (coord via state)

concurrent R-access

exclusive RW-access

ACCESS COORDINATION IN ADA : STACK

```
PROTECTED TYPE StackTy IS
    ENTRY Push(Val: IN Integer);
    ENTRY Pop(Val: OUT Integer);
PRIVATE
    MaxNumValues : constant Integer := 10;
    StackData : ARRAY(1..MaxNumValues);
    TOS: Integer Range 0..MaxNumValues := 0;
END StackTy;

PROTECTED BODY StackTy IS
    FUNCTION IsEmpty Return Boolean IS Return TOS=0; END;
    FUNCTION IsFull Return Boolean IS Return TOS = MaxNumValues;
    ENTRY Push(Val: IN Integer) WHEN Not IsFull IS
        TOS := TOS+1;
        StackData(TOS) := Val;
    END Push;
    ENTRY Pop(Val: OUT Integer) WHEN Not IsEmpty IS
        Val := StackData(TOS);
        TOS := TOS-1;
    END Pop;
END;
```

ACCESS COORDINATION IN ADA : NOCREDITACCOUNT

```
PROTECTED  TYPE   NoCreditAccountTy  IS
      PROCEDURE  Deposit(Amount: IN Positive);
      ENTRY  WithDraw(Amount: IN Positive);
PRIVATE
      Balance : Natural := 0;
END  NoCreditAccountTy;

PROTECTED BODY  NoCreditAccountTy  IS
      PROCEDURE  Deposit(Amount: IN Positive) IS
      BEGIN    Balance := Balance + Amount;
      END Deposit;
      ENTRY  WithDraw(Amount: IN Positive)
                  WHEN  Amount <= Balance   IS
      BEGIN     Balance := Balance – Amount;
      END WithDraw;
END NoCreditAccountTy;
```

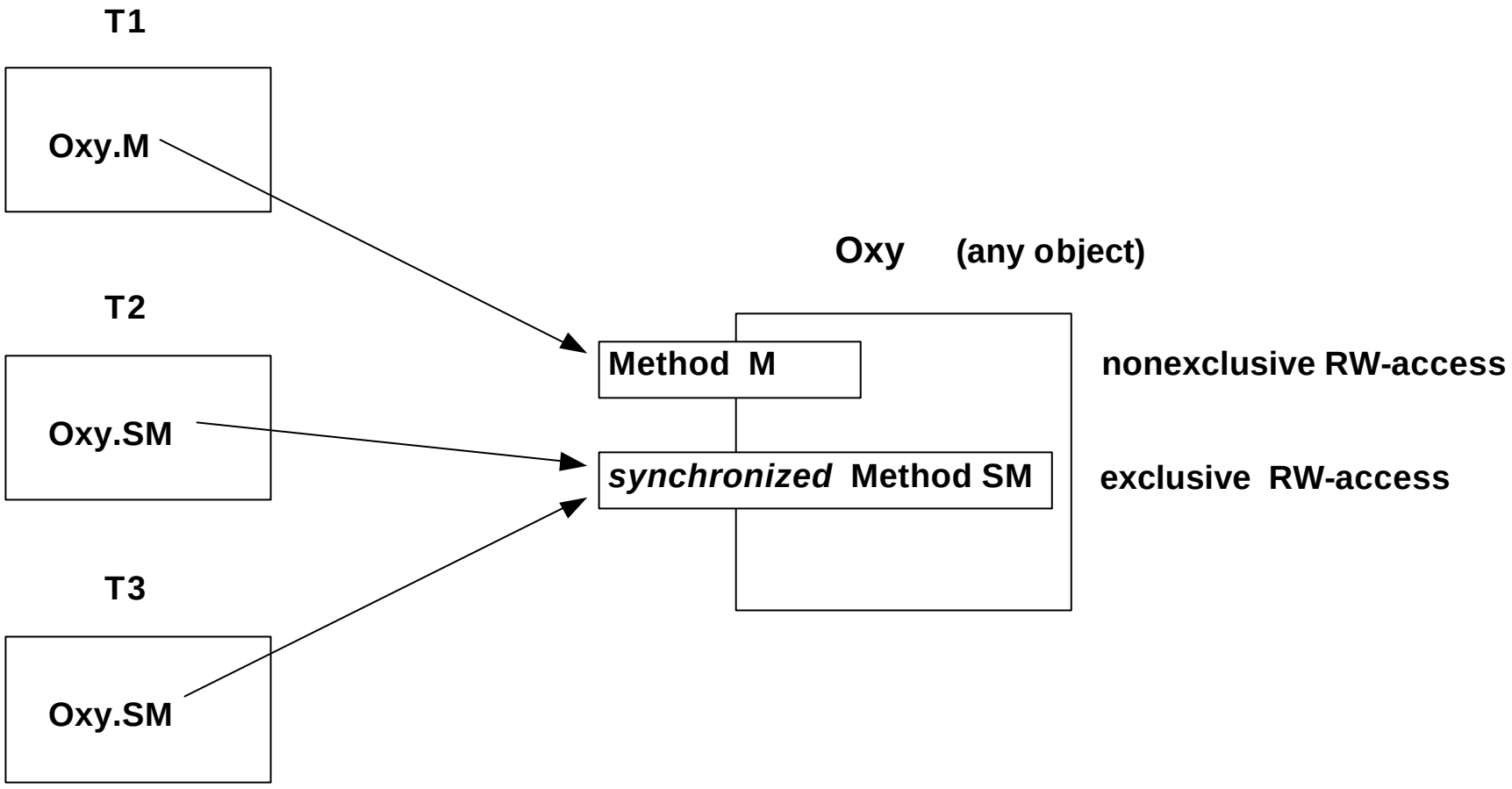
Program is ILLEGAL !
Visibility in barrier: all globals outside of entry espec. parameters NOT visible

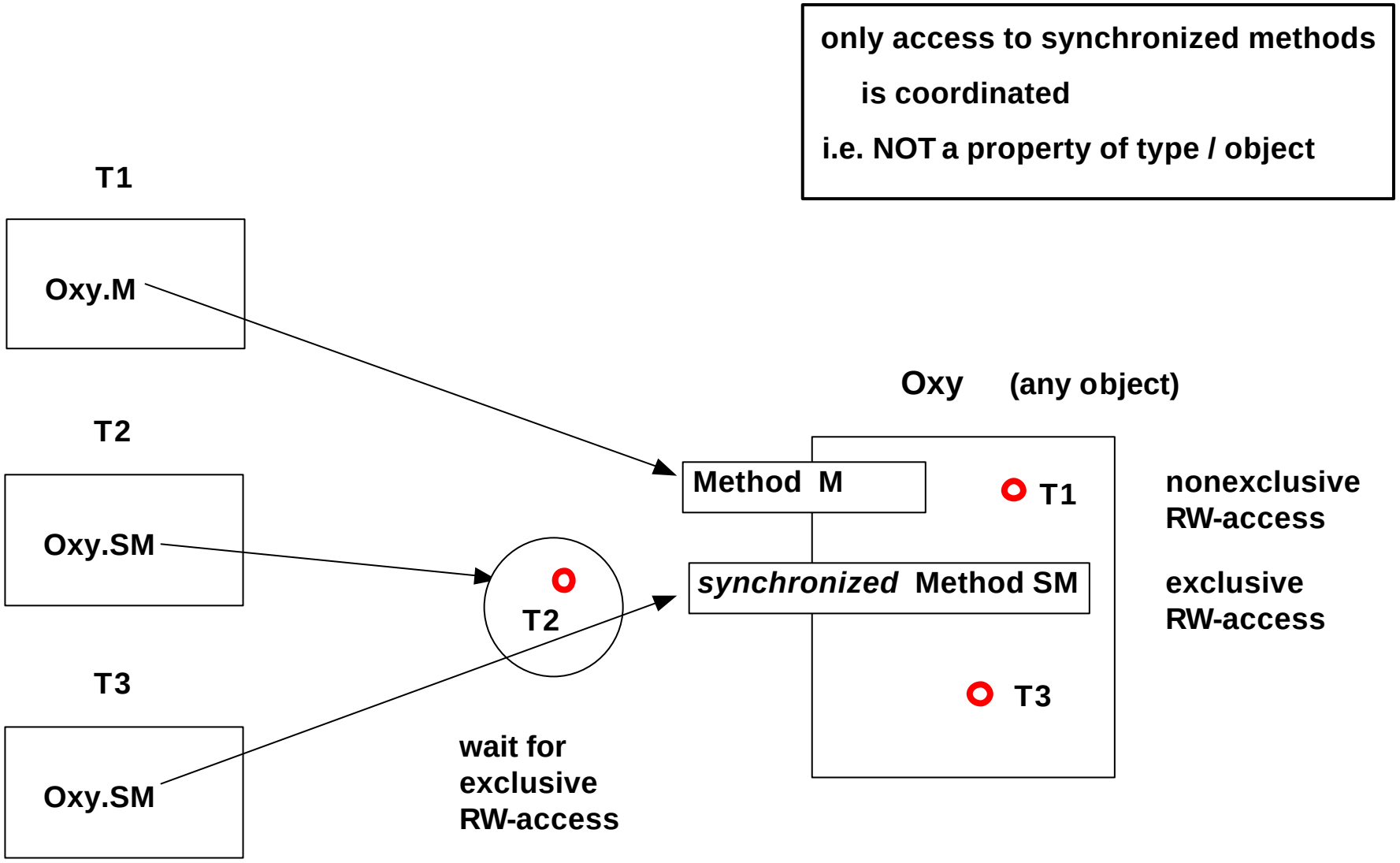
Solution: use REQUEUE and additional internal entry

rather cumbersome

```
PROTECTED TYPE NoCreditAccountTy IS
    PROCEDURE Deposit(Amount: IN Positive);
    ENTRY WithDraw(Amount: IN Positive);
PRIVATE
    Balance : Natural := 0;
    WithDrawAmount : Positive;
    WithDrawOpen: Boolean := True;
    ENTRY InternalWithDraw(Amount: IN Positive);
END NoCreditAccountTy;

PROTECTED BODY NoCreditAccountTy IS

    PROCEDURE Deposit(Amount: IN Positive) IS
    BEGIN   Balance := Balance + Amount;
    END Deposit;

    ENTRY WithDraw(Amount: IN Positive) WHEN WithDrawOpen IS
    BEGIN   IF Amount <= Balance
            THEN Balance := Balance – Amount;
            ELSE   WithDrawAmount := Amount;
                   WithDrawOpen := False;
                   Requeue InternalWithDraw;
                       -- parameter transmission implicit
            END;
    END WithDraw;

    ENTRY InternalWithDraw(Amount: IN Positive) IS
              WHEN WithDrawAmount <= Balance
    BEGIN   Balance := Balance – Amount;
            WithDrawOpen := True;
    END InternalWithDraw;

END NoCreditAccountTy;
```

ACCESS COORDINATION IN JAVA : ANY OBJECT CAN BE *LOCKED*

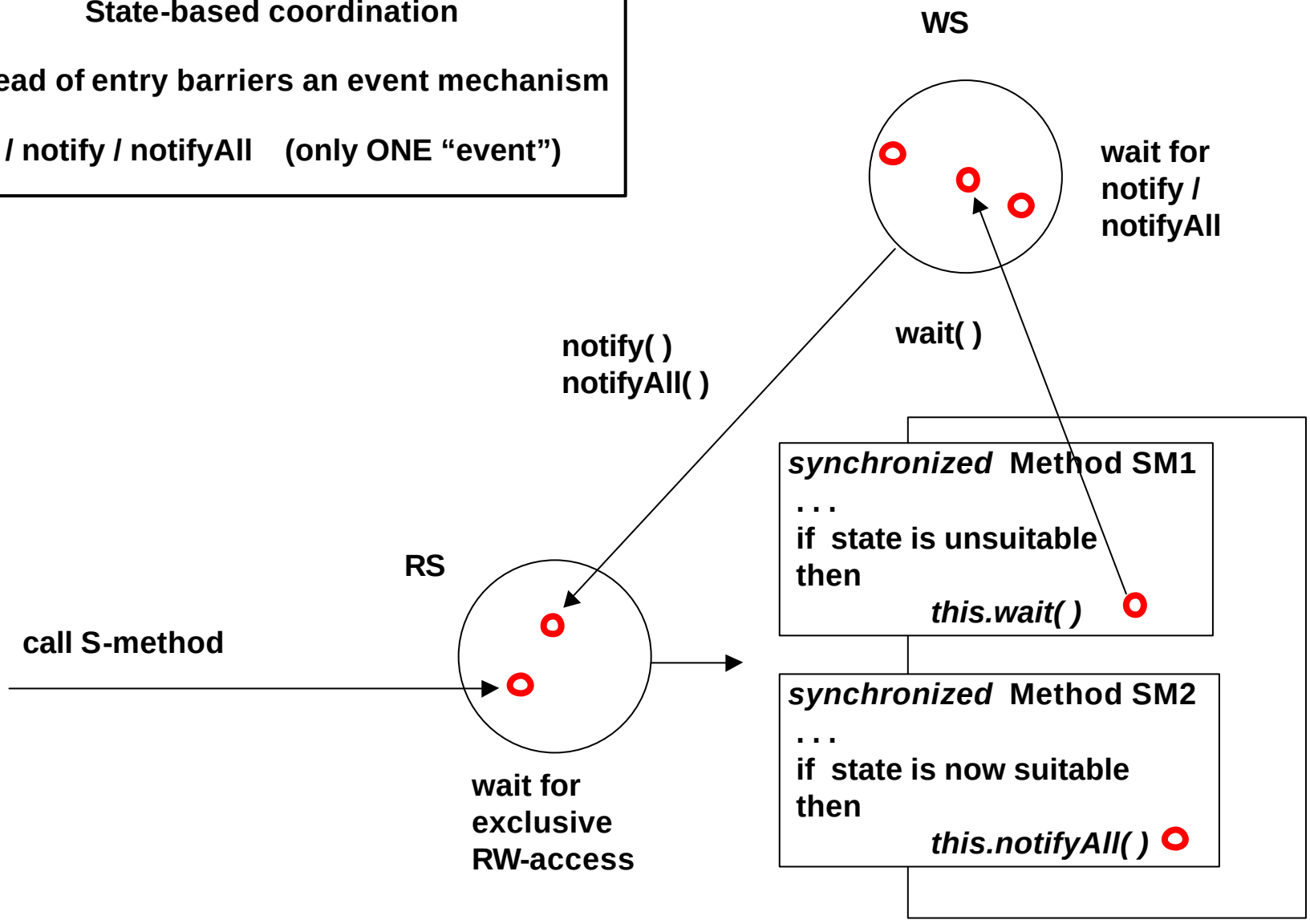



only access to synchronized methods
is coordinated
i.e. NOT a property of type / object
T1
Oxy.M
T2
Oxy.SM
T3
Oxy.SM
Oxy (any object)
Method M
T1
synchronized Method SM
T2
T3
nonexclusive RW-access
exclusive RW-access
wait for exclusive RW-access

State-based coordination

instead of entry barriers an event mechanism

wait / notify / notifyAll (only ONE "event")



Characteristic properties

- for each object there is one RS and one WS
- acquire exclusive access to object: *one* process from RS is *arbitrarily* chosen
- notify( ): *one* process from WS is *arbitrarily* chosen and transferred from WS to RS
- notifyAll( ): transfer all processes from WS to RS

=> since there is only *one WS* and since *notify( ) chooses arbitrarily*

notifyAll( ) must be used quite often

=> processes are deblocked which cannot really continue

they just make another RS-execute-WS round-trip

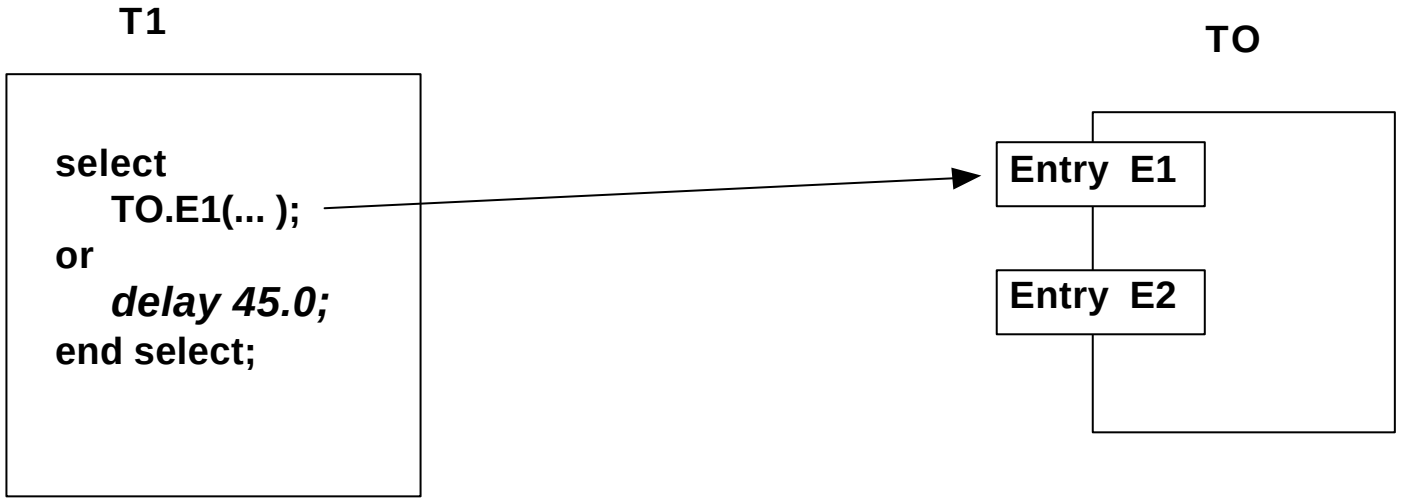
ACCESS COORDINATION IN JAVA : STACK

```
// A stack that has a 3 item limit

class   Stack  {

      static final int STACK_SIZE = 3;
      private int[ ] stack_store = new int[STACK_SIZE];
      private int stack_ptr = 0;

      // push item onto stack
      // If stack is full, wait until it has room
      synchronized public void push(int item)  {
            while (stack_ptr >= STACK_SIZE)  {
                  try  {   wait( );
                        }  catch { InterruptedException e)  {   /* ignore */     }
            }
            if (stack_ptr == 0)
                  notify( );          // pop was awaiting stack to fill
            stack_store[stack_ptr++] = item;
      }

      //  pop item off top of stack
      //  If stack is empty, wait until it has item
      synchronized public int pop( )  {
            while  (stack_ptr == 0)  {
                  try  {   wait( );
                        }  catch { InterruptedException e)  {   /* ignore */     }
            }
            if  (stack_ptr >= STACK_SIZE)
                  notify( );          //  push was awaiting stack to drain
            return (stack_store[--stack_ptr];
      }
}
```

From: Chan/Lee: The Java™ Class Libraries 1996, 1998

(has some flaws)

TIME-OUT IN ADA : CALLER SIDE



Characteristic properties

wait is expressed on the caller side
can be applied to any entry (call)
(time-out wrt BEGIN of rendezvous)

wait also possible on the callee side

TIME-OUT IN JAVA : CALLER SIDE

Characteristic properties

seems to be very simple, but it isn't

method M1 has to be modified in a rather complicated way

Assume: method M1 depends on suitable state

Solution

use wait (long timeout) in method M1

Problem: same WS as for other wait( )-calls is used

```
public synchronized void  M1(..., long timeout)  {
      if  state is not suitable
      then    wait(timeout);
            if  state is not suitable
            then return / throw
            end if
      end if
      perform action
      if  state is suitable for waiting processes
      then  notify( ) / notifyAll( )
      end if
      }
```

Problem

process may continue due to some unrelated notify / notifyAll

=> could give up TOO EARLY

Solution

use wait (long timeout) in method M1 and
CHECK REPEATEDLY WHETHER timeout HAS ELAPSED

```
public synchronized void M1(..., long timeout) {
    if state is not suitable
    then  EndTime = CurrentTime+timeout;
          while true
                wait(EndTime-CurrentTime);
                if state is suitable
                then  break
                else
                      if CurrentTime >= EndTime
                      then  return / throw
                      end if
                end if
          end while
    end if
    perform action

    if state is suitable for waiting processes
    then notify( ) / notifyAll( )
    end if
}
```

- Adapted from Doug Lea: “Concurrent Progr. in Java”
- still not an adequate solution
- each relevant operation has to be modified like that
- main problem: no connection between wait and state condition

SUMMARY

	Ada	Java
structure of actor	**very flexible**	**body = run( )**
direct interaction	**rendevous**	**-.-**
access coordination	**protected object very flexible problem: visib. in barrier**	**synchronized method problem: only excl. RW-access problem: primitive event mech.**
time-out	**easy**	**quite cumbersome**
active server	**task with entries**	**very difficult**
remote server	**-.-**	**RMI**

REFERENCES AND FURTHER READING

BE 98 Brömel, Peter; Ecke, Frank: Description and Comparison of the Concurrency Concepts of Ada, Java, and CHILL. Project Alfa Core. Friedrich Schiller University, Institute of Computer Science, 1998. http://www1.informatik.uni-jena.de/Themen/pap-talk/studarb-pb-fe.htm

BGJ 2000a Gosling, James; Joy, Bill; Steele, Guy L.; Bracha, Gilad: The Java Language Specification – Second Edition -. Addison-Wesley, Boston, etc. 2000. 0-201-31008-2

Brö 99 Brömel, Peter: Vergleich der Nebenläufigkeitskonzepte von Ada, CHILL, Erlang und Java – Fallstudien 1 -. Diplomarbeit, Friedrich-Schiller-Universität, Institut für Informatik, 1999. http://psc.informatik.uni-jena.de/Themen/pap-talk/da-pb.pdf

Bro 2000 Brosgol, Ben: A Comparison of the Concurrency and Real-Time Features of Ada 95 and Java. Ada UK conference 1998, Bristol. Update April 2000. http://wuarchive.wustl.edu/languages/ada/compiler/gnat/distrib/jgnat/papers/ada-java-concurrency-comparison.pdf 2001.Apr.18

CL 98 Chan,Patrick; Lee, Rosanna: The Java Class Libraries. Addison-Wesley, Reading etc., 1996. 0-201-63498-9

Eck 99 Ecke, Frank: Comparison of the Concurrency Concepts of Ada, CHILL, Erlang, and Java – Case Studies 2. Diploma Thesis, Friedrich Schiller University, Institute of Computer Science, 1999. http://psc.informatik.uni-jena.de/pap-talk/da-fe.pdf

ISO 8652 International Standard ISO/IEC 8652:1995(E): Information Technology - Programming Languages - Ada. Second ed. 1995-02-15. ISO/IEC, Geneva 1995

Lea 97 Lea, Doug: Concurrent Programming in Java. Addison-Wesley, Reading etc., 1997. 0-201-69581-2