

# Concurrent Programming

## CS511

# Teachers

Instructor: Eduardo Bonelli

E-mail: [ebonelli@stevens.edu](mailto:ebonelli@stevens.edu)

Office hours: Mon and Wed, 5-6:30PM.

CAs: Dennis Bollettino, Alfredo Mendez, Matthew Turner

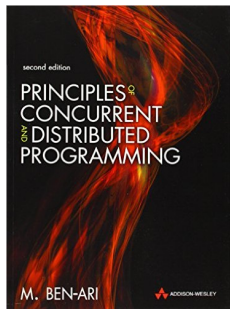
Office: Gateway South 350

# Ask questions!

- ▶ Feel free to interrupt and ask questions at any time
  - ▶ Your questions also help me better understand the topics
  - ▶ It also helps classmates who might have similar doubts
- ▶ Contact me by email
- ▶ Come see me during office hours

# Bibliography

- ▶ Slides, above all
- ▶ The book we use



# Credits

This course has benefitted from material from the following sources:

- ▶ <https://sites.google.com/site/pconctpiunq/> (Daniel Ciolek and Hernán Melgratti)
- ▶ Slides from the course on Concurrency at Chalmers (TDA382/DIT390)

# General Structure of the Course

- ▶ Lectures
- ▶ Assignments:
  - ▶ Compulsory
- ▶ Exercise booklets
  - ▶ Crucial
- ▶ Quizzes
- ▶ Exams:
  - ▶ Midterm and Endterm

Read syllabus for full details

# Weight of Grading Categories

Homework	(30%)
Quizzes	(20%)
Midterm	(25%)
Endterm	(25%)

Additional considerations:

- ▶ There is no curving for this course

# Contents

- ▶ First half: Process synchronization in shared memory model
  - ▶ Java/Groovy
- ▶ Third Quarter: Process synchronization in message passing model
  - ▶ Erlang
- ▶ Fourth Quarter: Model checking
  - ▶ Spin (Promela)
  - ▶ Concurrency is hard! We need tool support based on formal methods



# Course Objectives

- ▶ Understand classic problems in Concurrent Programming (CP) such as [synchronization](#)
- ▶ Understand the primary primitives used in CP
- ▶ Develop skills to be able to use these primitives in solving synchronization problems
- ▶ Understand the fundamentals of [model-checking](#) for checking properties of concurrent systems
- ▶ Apply [model-checking](#) to simple synchronization problems

About this Course

What is Concurrency?

Process Scheduling and New Types of Program Errors

Shared Memory Model

# The paper that began the field of concurrency<sup>1</sup>

## Solution of a Problem in Concurrent Programming Control

E. W. DIJKSTRA

*Technological University, Eindhoven, The Netherlands*

A number of mainly independent sequential-cyclic processes with restricted means of communication with each other can be made in such a way that at any moment one and only one of them is engaged in the "critical section" of its cycle.

### Introduction

Given in this paper is a solution to a problem for which, to the knowledge of the author, has been an open question since at least 1962, irrespective of the solvability. The paper consists of three parts: the problem, the solution, and the proof. Although the setting of the problem might seem somewhat academic at first, the author trusts that anyone familiar with the logical problems that arise in computer coupling will appreciate the significance of the fact that this problem indeed can be solved.

computer can only request one one-way message at a time. And only this will make the reader realize to what extent this problem is far from trivial.

### The Solution

The common store consists of:

"Boolean array  $b, c[1:N]$ ; integer  $k$ "

The integer  $k$  will satisfy  $1 \leq k \leq N$ ,  $b[i]$  and  $c[i]$  will only be set by the  $i$ th computer; they will be inspected by the others. It is assumed that all computers are started well outside their critical sections with all Boolean arrays mentioned set to **true**; the starting value of  $k$  is immaterial.

The program for the  $i$ th computer ( $1 \leq i \leq N$ ) is:

```
"integer j;  
Li0: b[i] := false;  
Li1: if k ≠ i then  
Li2: begin c[i] := true;  
Li3: if b[k] then k := i;  
go to Li1  
end  
else  
Li4: begin c[i] := false;  
for j := 1 step 1 until N do
```

CACM (September, 1965)

---

<sup>1</sup>**The Early Days of Concurrency. A Memoir**, Leslie Lamport, Microsoft Research. Talk at 1st ACM SIGOPS Summer School on Advanced Topics in Systems (SATIS '18), Aug 14-17, 2018, Tromsø, Norway.

[drive.google.com/file/d/1cISMnW-HsczLYoIEzNNOLv4c9r-5d4Dw/view](https://drive.google.com/file/d/1cISMnW-HsczLYoIEzNNOLv4c9r-5d4Dw/view)

# Concurrency

- ▶ The study of systems of **interacting computer programs** which **share resources** and run **concurrently**, i.e. at the same time
- ▶ **Parallelism**
  - ▶ Occurring **physically** at the same time
- ▶ **Concurrency**
  - ▶ Occurring **logically** at the same time, but could be implemented without real parallelism
- ▶ We focus on **high-level synchronization**
  - ▶ Distinction concurrency/parallelism is irrelevant for us

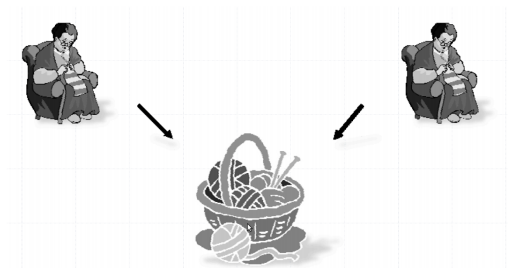
# Interaction Models

- ▶ What is process synchronization?
  - ▶ Ensure that instructions are executed in certain order
- ▶ Synchronization is irrelevant if processes do not interact with each other
- ▶ They just “do their own thing”

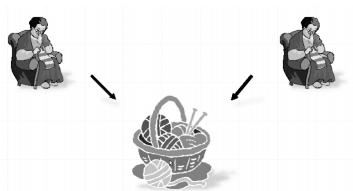


# Process Interaction

- ▶ Concurrency, and hence process synchronization, is useful **only** when processes interact with each other
- ▶ What does it mean for processes to interact?
  - ▶ They **share resources**
- ▶ For example, two grannies and only one set of knitting needles



# Interaction Models



- ▶ How may the needles be shared?
- ▶ Two fundamental models of interaction
  - ▶ Shared Memory
    - ▶ Read/assign to a variable or memory location
  - ▶ Message passing
    - ▶ Send/receive

About this Course

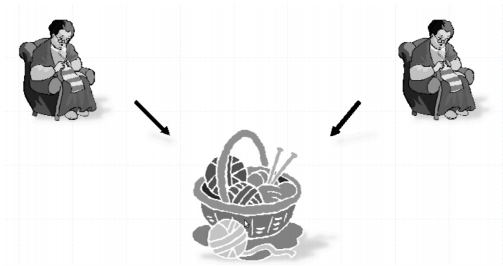
What is Concurrency?

Process Scheduling and New Types of Program Errors

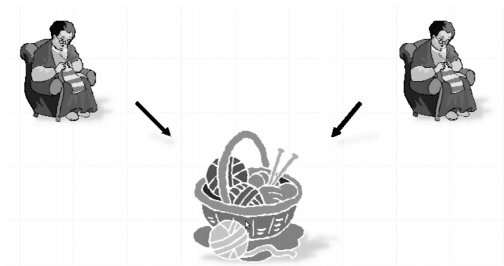
Shared Memory Model



# Competitive Processes

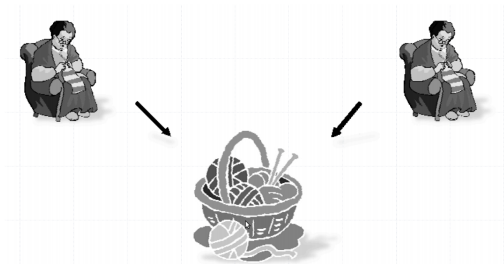


# Competitive Processes



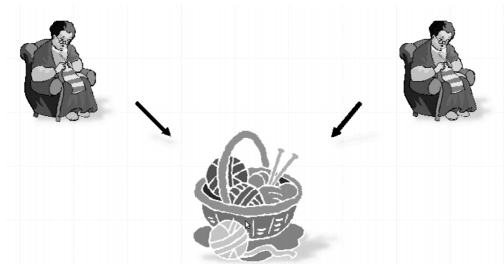
- **Deadlock**: each granny takes a needle and waits indefinitely until the other one has freed the one she has.

# Competitive Processes



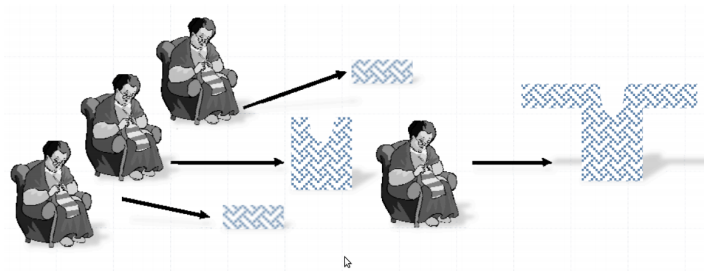
- ▶ **Deadlock:** each granny takes a needle and waits indefinitely until the other one has freed the one she has.
- ▶ **Livelock:** each granny takes a needle, sees that the other granny has the other needle and returns it (this repeats indefinitely).

# Competitive Processes



- ▶ **Deadlock:** each granny takes a needle and waits indefinitely until the other one has freed the one she has.
- ▶ **Livelock:** each granny takes a needle, sees that the other granny has the other needle and returns it (this repeats indefinitely).
- ▶ **Starvation:** one of the grannies always takes the needles before the other one.

## Cooperative<sup>2</sup> Processes



- Communication mechanisms are necessary for cooperation to be possible

---

<sup>2</sup>Unrelated to cooperative or non-preemptive scheduling

About this Course

What is Concurrency?

Process Scheduling and New Types of Program Errors

Shared Memory Model

# Modelling Program Execution

What is the value of  $x$  after executing this program?

```
1  int x=0;
2
3  // Thread P
4  Thread.start{
5      x=1
6  }
7
8  // Thread Q
9  Thread.start{
10     x=2
11 }
```

► Value of  $x$  after execution of just P?

# Modelling Program Execution

What is the value of  $x$  after executing this program?

```
1  int x=0;
2
3  // Thread P
4  Thread.start{
5      x=1
6  }
7
8  // Thread Q
9  Thread.start{
10     x=2
11 }
```

- ▶ Value of  $x$  after execution of just P?
- ▶ Value of  $x$  after execution of just Q?



# Modelling Program Execution

What is the value of  $x$  after executing this program?

```
1  int x=0;
2
3  // Thread P
4  Thread.start{
5      x=1
6  }
7
8  // Thread Q
9  Thread.start{
10     x=2
11 }
```

- ▶ Value of  $x$  after execution of just P?
- ▶ Value of  $x$  after execution of just Q?
- ▶ Value of  $x$  after execution of P || Q?

# Modelling Program Execution

What is the value of  $x$  after executing this program?

```
1  int x=0;
2
3  // Thread P
4  Thread.start{
5      x=1
6  }
7
8  // Thread Q
9  Thread.start{
10     x=2
11 }
```

- ▶ Value of  $x$  after execution of just P?
- ▶ Value of  $x$  after execution of just Q?
- ▶ Value of  $x$  after execution of  $P \parallel Q$ ?  $\{x = 0, x = 1\}$  More than one result is possible!

# Modelling Program Execution

What is the value of  $x$  after executing this program?

```
1 // Thread P      1 // Thread Q
2 Thread.start{    2 Thread.start{
3     println "A"; 3     println "C";
4     println "B"; 4     println "D"
5 }                5 }
```

- ▶ Number of interleavings is exponential in the number of instructions
- ▶ If P has  $m$  instructions and Q has  $n$  instructions, then there are

$$\binom{m+n}{m} = \frac{(m+n)!}{m!n!}$$

interleavings

# Modelling Program Execution

A **Transition System**  $\mathcal{A}$  is a tuple

$$(S, \rightarrow, I)$$

where

- ▶  $S$  is a set of **states**
- ▶  $\rightarrow \subseteq S \times S$  is a **transition relation**
- ▶  $I \subseteq S$  set of **initial states**

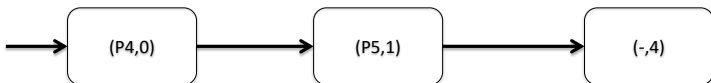
Note:

- ▶  $\mathcal{A}$  is said to be **finite** if  $S$  is finite
- ▶ We write  $s \rightarrow s'$  for  $(s, s') \in \rightarrow$ .

## Example 1 – Sequential Program

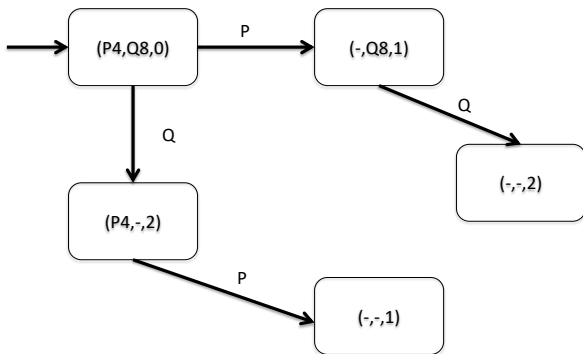
```
1 int x=0;
2
3 Thread.start { //P
4     x = 1;
5     x = x + 3;
6 }
```

- ▶  $S$ : tuples that include
  1. the program pointer of each thread at a given point in time
  2. the value of the variables and
- ▶  $s \rightarrow t$  if executing a statement in  $s$  results in the state  $t$

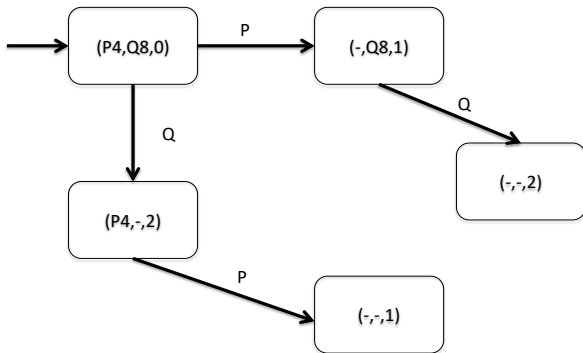


## Example 3 – Concurrent Processes

```
1  int x=0;
2
3  Thread.start{ // P
4      x=1
5  }
6
7  Thread.start{ // Q
8      x=2
9  }
```



## Example 3 – Concurrent Processes



Examples of paths in textual notation:

- ▶  $(P4, Q8, 0) \rightarrow (-, Q8, 1) \rightarrow (-, -, 2)$
- ▶  $(P4, Q8, 0) \rightarrow (P4, -, 2) \rightarrow (-, -, 1)$
- ▶  $(P4, Q8, 0) \rightarrow (P4, -, 2)$

# Execution Speed as a Synchronization Mechanism?

► No

► Eg. The following still has two possible results

```
1 int x=0;
2
3 Thread.start{ // P
4     sleep(500);
5     x=1
6 }
7
8 Thread.start{ // Q
9     x=2
10 }
```



# Summary

- ▶ We need concurrency to exploit the processor
- ▶ Concurrent programs are non-deterministic
- ▶ In this course we will study different synchronization mechanisms that will allow us to control the behavior of concurrent programs
- ▶ In particular, we will use synchronization mechanisms to ensure that our programs satisfy desirable properties to be introduced later