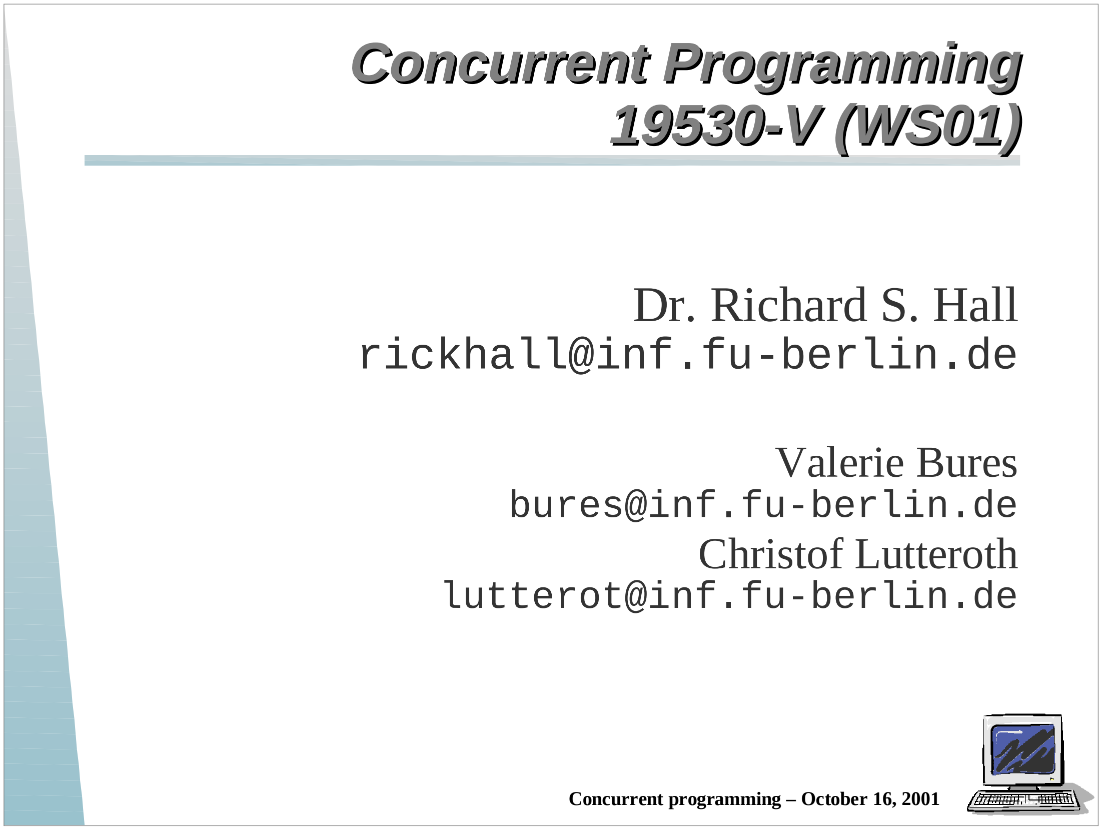
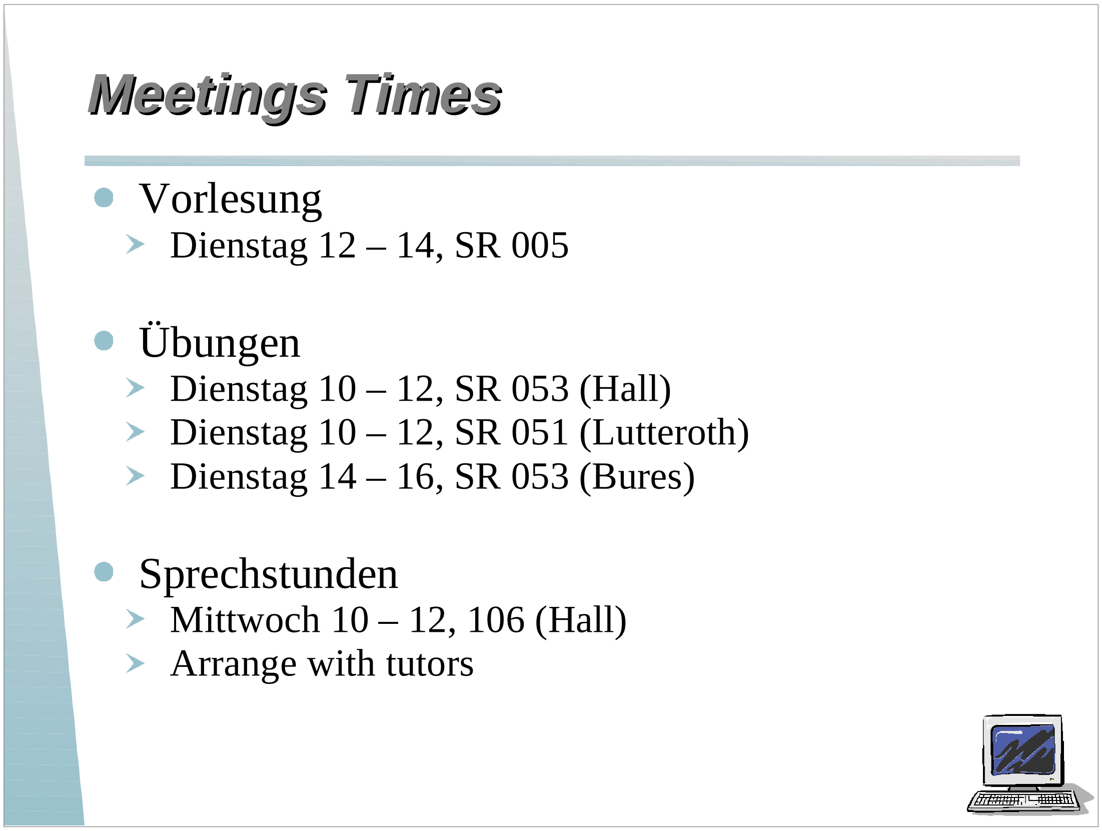


Concurrent Programming 19530-V (WS01)

Dr. Richard S. Hall
rickhall@inf.fu-berlin.de

Valerie Bures
bures@inf.fu-berlin.de
Christof Lutteroth
lutterot@inf.fu-berlin.de



Meetings Times

- Vorlesung
  - Dienstag 12 – 14, SR 005
- Übungen
  - Dienstag 10 – 12, SR 053 (Hall)
  - Dienstag 10 – 12, SR 051 (Lutteroth)
  - Dienstag 14 – 16, SR 053 (Bures)
- Sprechstunden
  - Mittwoch 10 – 12, 106 (Hall)
  - Arrange with tutors

Purpose of this Class

- Discuss the unique characteristics of designing and implementing concurrent software systems
  - Practical software engineering perspective
    - General approach and methodology
  - Technological perspective
    - Java programming language
- Provide students with sufficient background on current programming so that they can write reasonably complex concurrent programs

Expectations

- Benoteter Schein based on
  - Exercises (Übungen)
    - 40% of grade
    - Approximately one assignment per week
    - Collected and graded
    - Students will present solutions in their Übungen
  - Project
    - 20% of grade
    - Broken into parts over the last few weeks
  - Klausur
    - 40% of grade
    - ***Important!!!***

      The Klausur will be the last day of class, Februar 12
  - Scores on all three will be averaged, 60% needed to pass the class

Übungen

- There will be no exercises for this lecture
- The first exercises will be handed out next week
- This means that the Übungen do not meet this week or the next week
- Übungen start next week on 23.10.2001
  - Organizational issues

Reading List

- Concurrency: State Models & Java Programs
  - Jeff Magee and Jeff Kramer, Wiley, 1999.
    (This is the main textbook for the class.)
- Foundations of Multithreaded, Parallel and Distributed Computing
  - Gregory A. Andrews, Addison-Wesley, 2000.
- Concurrent Programming in Java, Second Edition
  - Doug Lea, Addison-Wesley Publishing, 2000.

What is Sequential Programming?

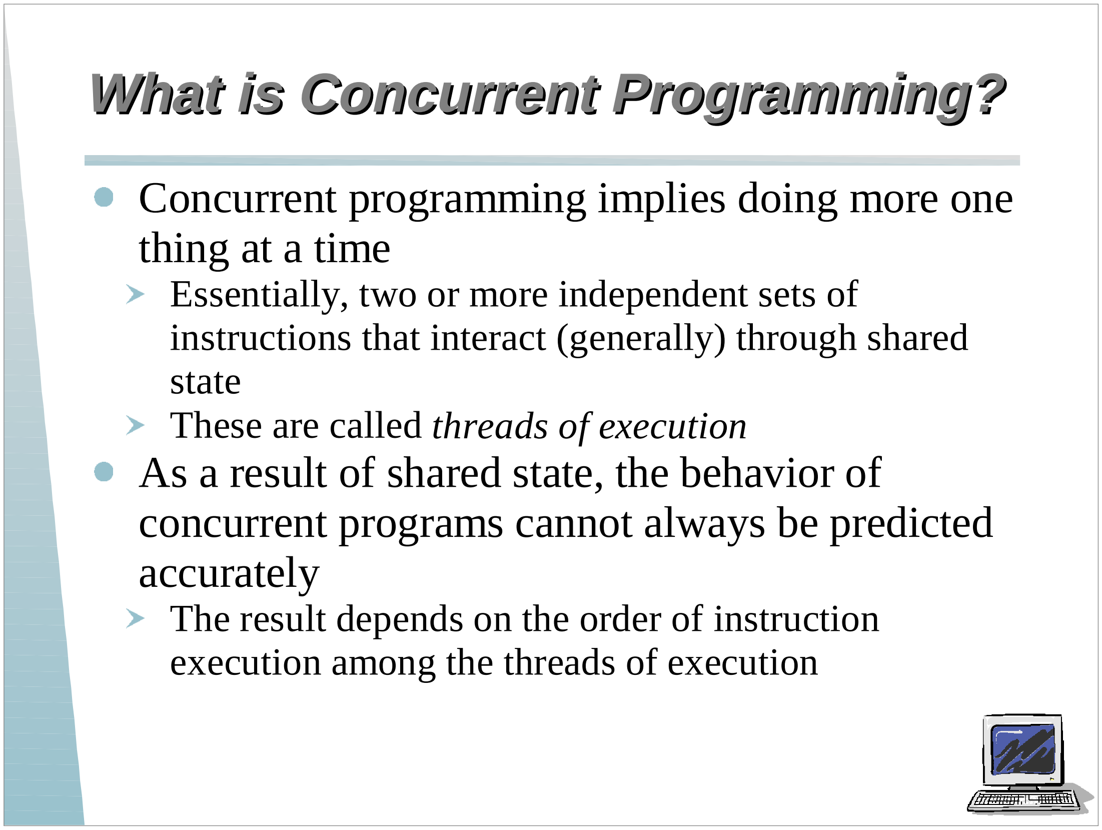
- Instructions occur in a predictable "sequence" every time they are executed
- For example, we can always predict what the following output will be, knowing the input

```
Int x = 0, y = 0;
x = read();  // input 5
if (x > 0)
   y = 100;
else
   y = -100
System.out.println(y);
```

- Why do we know the output?
  - Command execution is repeatable and sequential

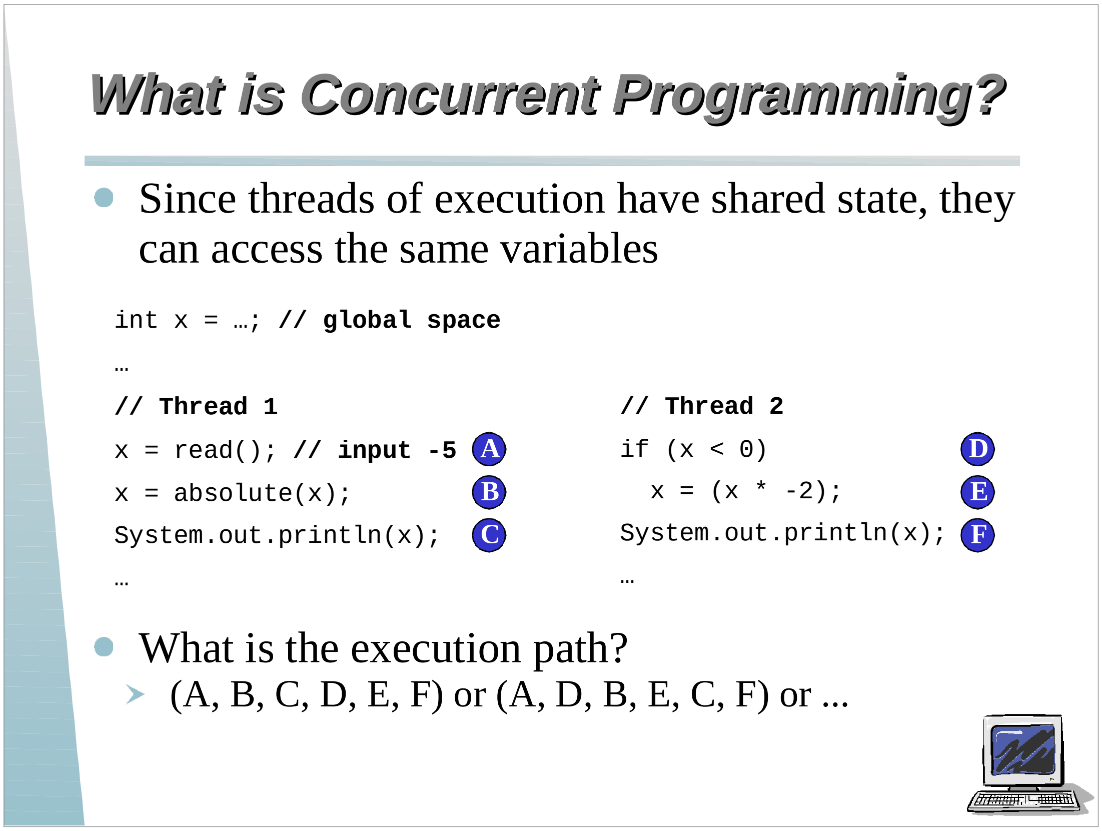
Confusing Terminology

- In the English language, *concurrent* means "*happening at the same time as something else*"
- In computer science, *concurrent* means "*happening in non-sequential order*"
  - This does not generally mean that things are happening at the same time, although they could be
- In English and computer science, *parallel* means "*happening at the same time as something else*"
  - Everything in concurrent programming applies to parallel programming as well



What is Concurrent Programming?

- Concurrent programming implies doing more one thing at a time
  - Essentially, two or more independent sets of instructions that interact (generally) through shared state
  - These are called *threads of execution*
- As a result of shared state, the behavior of concurrent programs cannot always be predicted accurately
  - The result depends on the order of instruction execution among the threads of execution



What is Concurrent Programming?

- Since threads of execution have shared state, they can access the same variables

```
int x = …; // global space
…
// Thread 1                            // Thread 2
x = read(); // input -5  Ⓐ             if (x < 0)              Ⓓ
x = absolute(x);         Ⓑ               x = (x * -2);         Ⓔ
System.out.println(x);   Ⓒ             System.out.println(x);  Ⓕ
…                                      …
```

- What is the execution path?
  - (A, B, C, D, E, F) or (A, D, B, E, C, F) or ...

How do we achieve concurrency?

- Multiple computers
- Multiple processes
- Multiple threads
  - We will learn more about this later
- Any combination of the above

Why do we want concurrency?

- *Performance* - multi-processor machines
- *Responsiveness* - user interfaces
- *Efficiency* - blocking calls
- *Naturalness* - related, but separate activity streams

Class Discussion Overview

- Over the course of this class we will discuss many topics, including but not limited to
  - Finite State Processes / Label Transition Systems
  - Atomicity
  - Mutual exclusion
  - Semaphores
  - Monitors
  - Synchronization
  - Condition variables
  - Deadlock
  - Safety and liveness
  - Concurrent programming in Java