AARHUS UNIVERSITET

Software Engineering and Architecture

Concurrency

A Classic Java Take on it

AARHUS UNIVERSITET

Disclaimer…

- I have *relatively little experience* in large scale, realistic, development of parallel and concurrent programs ☹
- The ‘handling concurrency’ scene is a vast topic, and has transformed considerably over the last decade!
  - Multicore processors
  - And OS/Libraries to take advantage of them !
- We will only treat *classic and basic issues and solutions!*
  - So, read up on the material once you are ‘out there’…
  - The problems are the same, but solutions become better…

AARHUS UNIVERSITET

Motivation

- Concurrency = many ‘objects’ executing at the *same* time
- Why?

- Modelling: This is how the world is!
  - Many people working in parallel, collaborating, sharing…

- Quality Attributes of our architecture
  - Performance
  - Responsiveness / Availability

AARHUS UNIVERSITET

Responsiveness

- *Sometimes* our computations take quite a while to complete
- Example:
  - User 1 searches for all flights to Bali
    - Server is busy requesting a lot of external booking systems
  - *Meanwhile*
  - User 2 wants to search for flights to Tokyo
- *But what happens here?*

```
while(! isStopped()){
    Socket clientSocket = null;
    try {
        clientSocket = this.serverSocket.accept();
    } catch (IOException e) {
        if(isStopped()) {
            System.out.println("Server Stopped.") ;
            return;
        }
        throw new RuntimeException(
            "Error accepting client connection", e);
    }
    try {
        processClientRequest(clientSocket);
    } catch (Exception e) {
        //log exception and go on to next request.
    }
}
```

Solution

- Just like one cashier in Føtex can only handle a limited number of customers at the same time; so can a single thread
- Solution: *Employ more cashiers / threads!*
- *However… Poses its own set of challenges!*

AARHUS UNIVERSITET

Analyzing Code…

- … is based upon a *sequential execution of statements*

```
interface Account {
    public boolean deposit(long amount);
    public boolean withdraw(long amount);
    public long getBalance();
}
```

```
class SingleThread {
  public static void main(String[] args) {
    Account a = new AccountImpl();

    a.deposit(500);
    a.withdraw(100);
    a.withdraw(100);
}
```

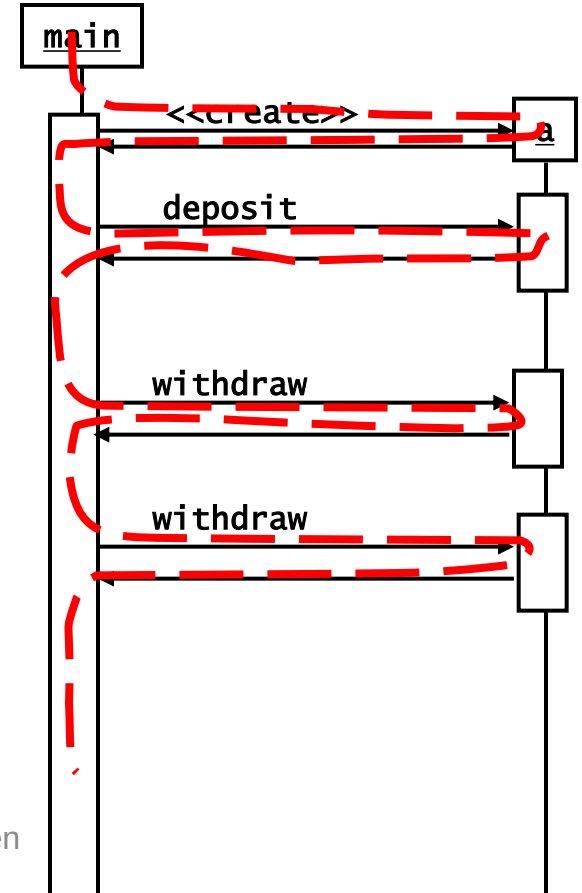
- Exercise: What is ‘a.balance()’ after the last withdraw()?
  - Assuming the balance is 0.00 at the start…

AARHUS UNIVERSITET

Program Thread…

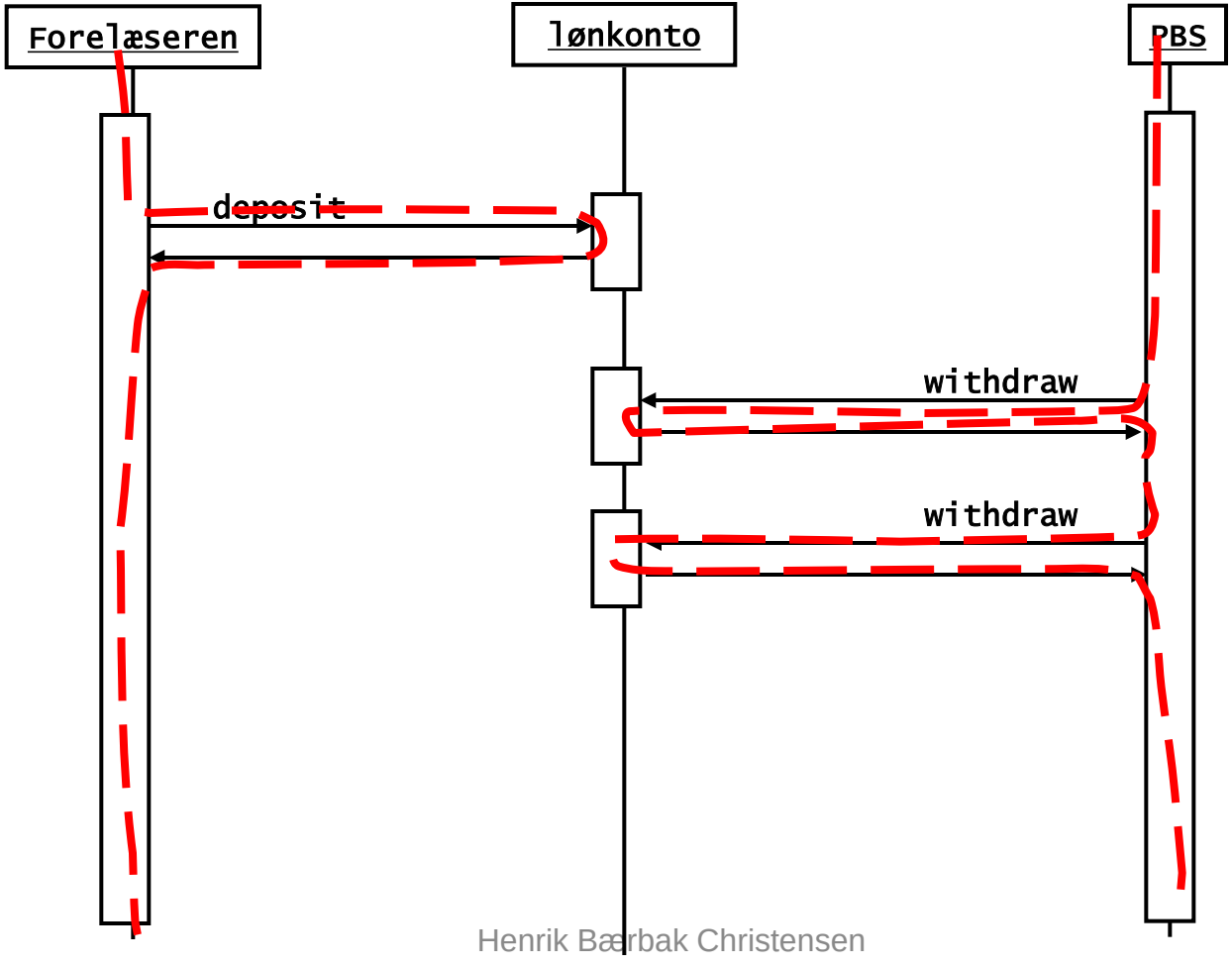
- The *program thread* weaves through methods and statements…

- In machine code
  - Register PC
    - Program counter
  - Increments for every instruction
  - Some instructions change PC
    - JMP 47 =
      - Change PC to address 47
    - I.e. a method call…



AARHUS UNIVERSITET

Two Threads!



AARHUS UNIVERSITET

Three Types of Concurrency

- When more than one thread executes in a program, we say that it is *concurrently executed*, it is a *concurrent* program.
- Three categories of concurrent programs
  - Independent threads
    - Like running your music player program while coding in IntelliJ
  - Shared resources
    - Like two threads reading/writing to the *same* account object
  - Collaborating threads
    - Like one thread inserting into a buffer and assuming some other thread will remove those items from the buffer

Java

- Java was one of the first mainstream languages to have threads as part of the language!
  - Before that,
    it was the job of the OS
    - *Processes*
  - *Coded using OS libraries*

```
public class ThreadDemo1 {
  public static void main(String[] args) {
    Thread a = new OutputThread('a');
    Thread b = new OutputThread('b');
    a.start(); b.start();
  }
}

class OutputThread extends Thread {
  private char c;

  OutputThread(char outputChar) {
    c = outputChar;
  }

  public void run() {
    for (int i=0; i<100; i++) {
      System.out.print(c); System.out.flush();
    }
  }
}
```

- Core class:
  - Thread
    - *Part of a framework!*

Thread

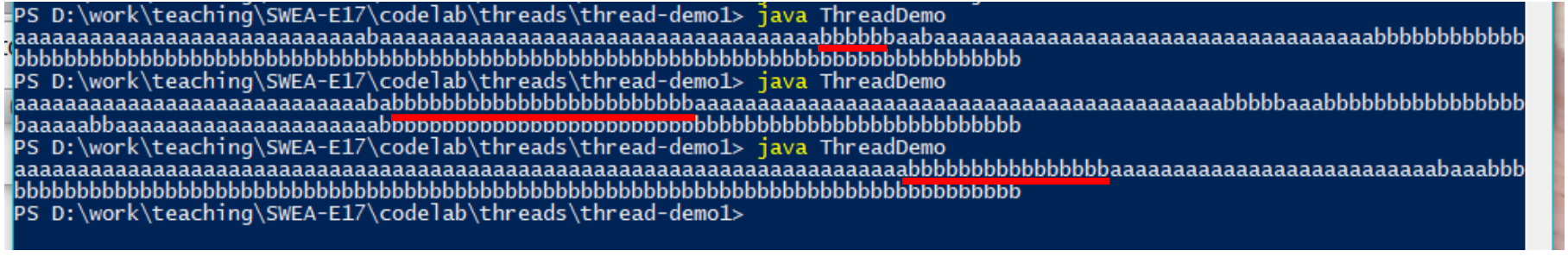
- Anatomy
  - Create a thread
  - Call start();
    - Will execute ‘run()’
- Exercise
  - How many threads?
  - What does it do?
  - *And what is the output?*

```
public class ThreadDemo1 {
  public static void main(String[] args) {
    Thread a = new OutputThread('a');
    Thread b = new OutputThread('b');
    a.start(); b.start();
  }
}

class OutputThread extends Thread {
  private char c;

  OutputThread(char outputChar) {
    c = outputChar;
  }

  public void run() {
    for (int i=0; i<100; i++) {
      System.out.print(c); System.out.flush();
    }
  }
}
```

AARHUS UNIVERSITET

Output

- The hallmark of *concurrent programs:* ***non-determinism***



- *Welcome to debugging hell!!!*
- *Welcome to testing hell!!!*
  - Testing is almost impossible, as there is a lot of randomness involved

AARHUS UNIVERSITET

Scheduling

- Threads execute concurrently
  - Abstractly speaking, even if they do not always in practice !
- In my youth we had *one CPU*
  - Today you 4, 8, 12, …, and several thousands in your GFX card
- Concurrency is (partly) *simulated* by
- *Thread scheduling*
  - *Preemptive:*
    - *Thread runs for n milliseconds, is interrupted and the scheduler then picks the next thread for execution*
  - *(Non-preemptive): Thread 'yields()' to signal thread change…*

AARHUS UNIVERSITET

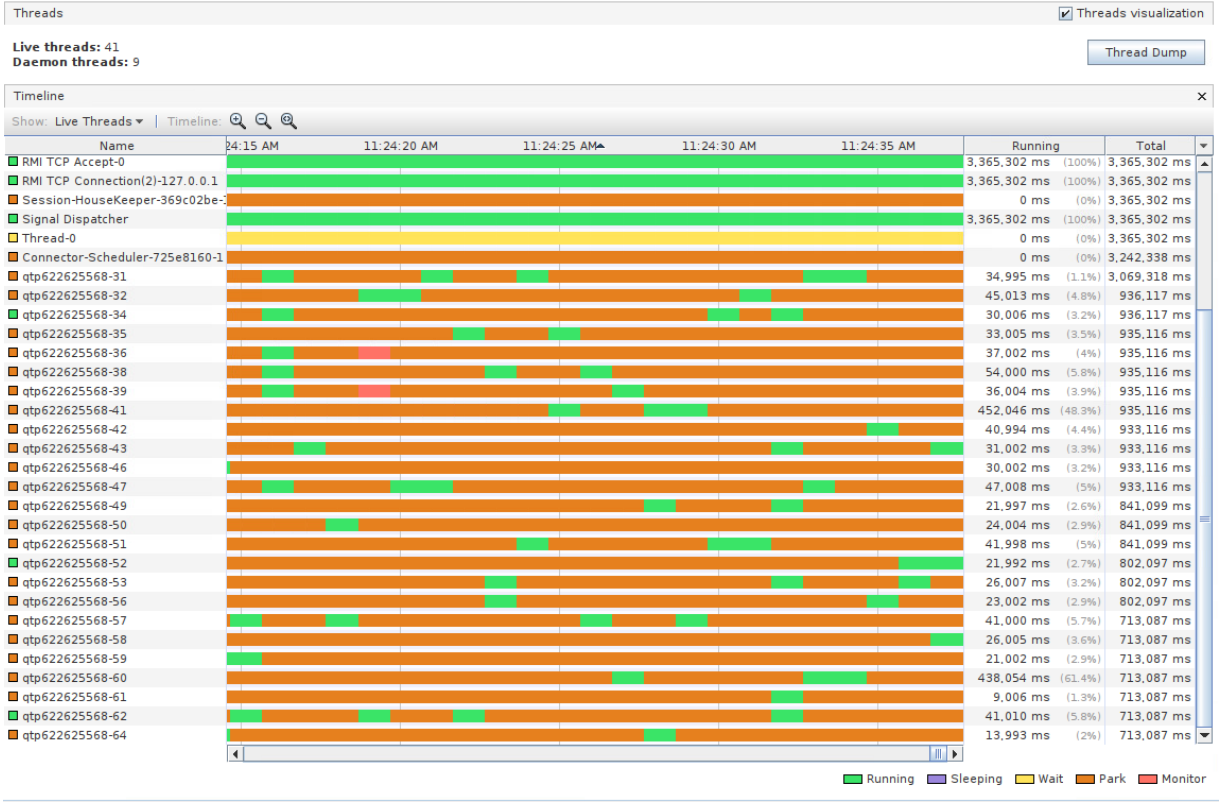
Thread States

- Any thread in a program are in one of several *states*
- *An incomplete list for Java includes*
  - *RUNNABLE:* *running or able to run (‘running’/’ready’)*
    - 100 threads may be runnable but only 1 [2, 4, 8] are *actually* executing code, the others are waiting for the scheduler to switch to them (ready/parked)
  - *BLOCKED:* *not executing, but waiting for a lock*
    - Used to handle ‘shared resources’, see later…
  - *WAITING:* *not executing, but in a wait-set, waiting*
    - Used to handle ‘producer-consumer’ / collaborating threads

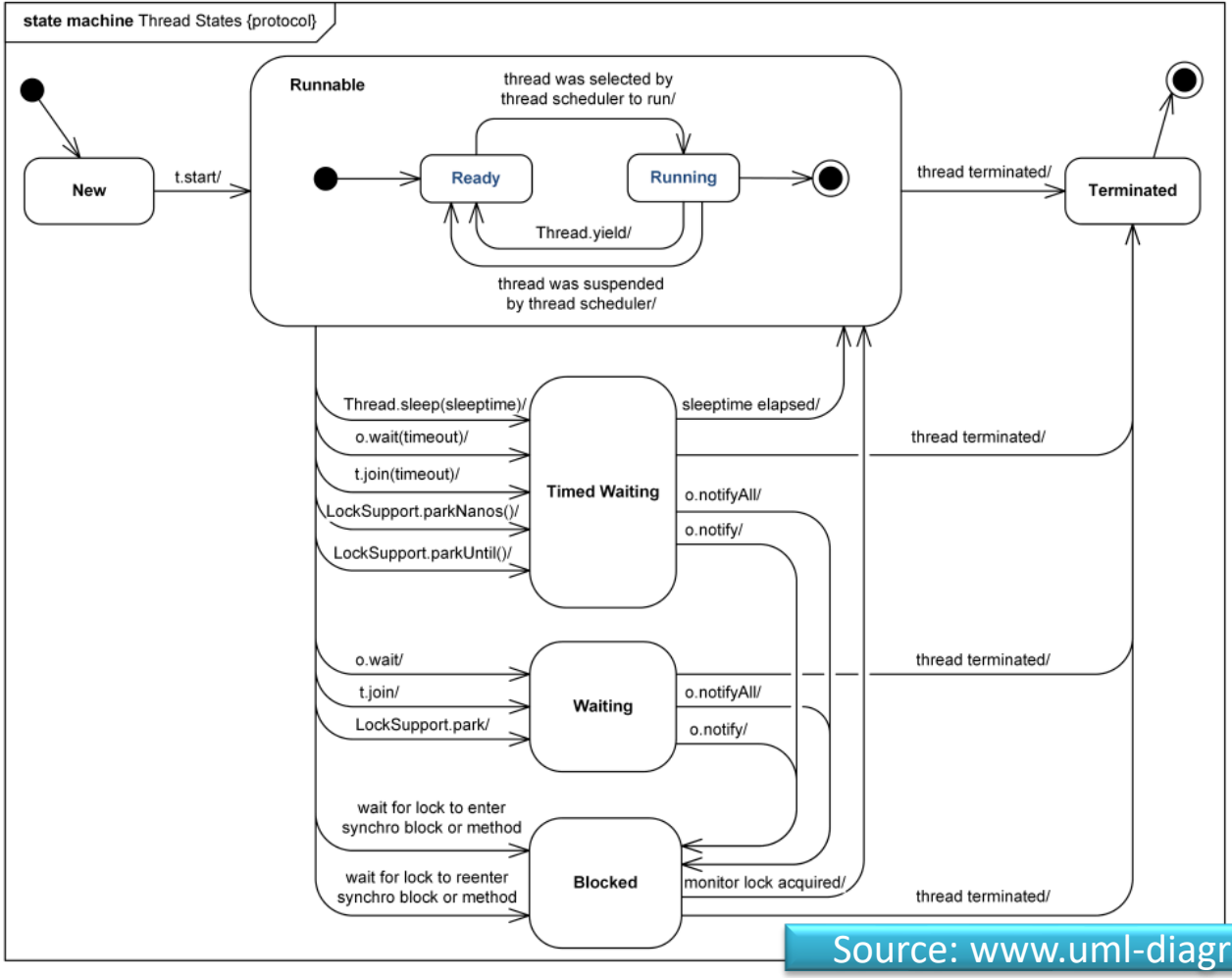
VisualVM can show thread states

AARHUS UNIVERSITET

- *Runnable* threads are either running or parked…



Thread States



Source: www.uml-diagrams.org

AARHUS UNIVERSITET

Subclassing? No no no ☺

- *Program to an interface!*
- Process
  - Provide Thread object with the Runnable instance
- Exercise:
  - What design pattern?

Runnable interface

```
public class ThreadDemo2 {
  public static void main(String[] args) {
    Thread a = new Thread(new OutputRunnable('a') );
    Thread b = new Thread(new OutputRunnable('b') );
    a.start(); b.start();
  }
}
class OutputRunnable implements Runnable {
  private char c;
  OutputRunnable(char outputchar) {
    c = outputchar;
  }
  public void run() {
    for (int i=0; i<100; ++i) {
      System.out.print(c); System.out.flush();
    }
  }
}
```

AARHUS UNIVERSITET

Overviewing Threads

- You may install ‘visualvm’, and overview a lot of the inner workings of your application’s threads (and heap and…)