16. Parallelität: Threads

Java-Beispiele:

ThreadBasicTest.java
ThreadSleep.java
ThreadJoin.java
SpotTest.java

Schwerpunkte

- Leichtgewichtige und schwergewichtige Prozesse
- Threads: nutzerprogrammierte Parallelität
- Threads: Lebenszyklus
- Steuerung von Threads:
  Erzeugen, Starten u. a. Operationen
- Synchronisation und Kommunikation

Keine Rechnernutzung ohne Parallelität - auch ohne nutzerdefinierte Parallelprogrammierung

- Ausnutzung der Ressourcen eines Rechners:
  Vieles läuft (scheinbar) parallel
  → Drucken, Tastatureingabe, Netzwerkdienste, ...

- Zwischen zwei Tastaturanschlägen:
  Millionen von Maschinenoperationen

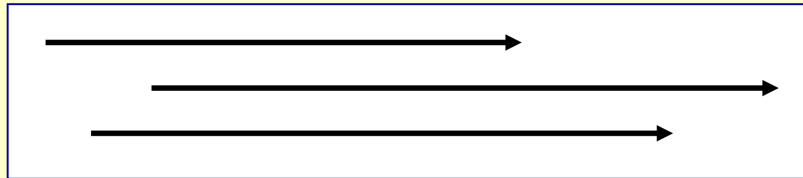
- Windows 7: mehr als 30 Dienste im Hintergrund:
  → Automatisches Laden von Updates von Microsoft-Servern,
  Drucker-Verwaltung,
  Netzwerkdienste (z. B. Verbindungsaufbau mit
  lokalen Servern),
  Backups von Systemdateien anlegen,
  Kommunikation von Programmen und Betriebssytem,
  Virenscanner,
  Fehlerprotokolle anlegen (Computer-Probleme aufzeichnen)
  ...

Grundlagen zur Parallelität:

- Arten von Parallelität
- Zustandsmodell für Thread-Lebenszyklus
- API-Klasse Thread

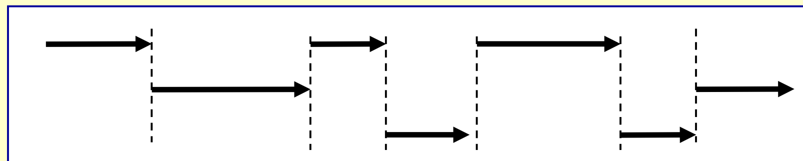
Parallelität

- Mehrere (sequentielle) Programme laufen gleichzeitig



Prozess:
sequentielles Programm

- Computer oft nur mit einem Prozessor:
  Pseudo-Parallelität
  → Rechenzeit scheibchenweise auf Prozesse verteilt

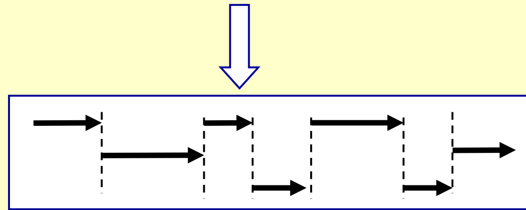


Vorteil: Wartezeiten von anderen Prozessen genutzt

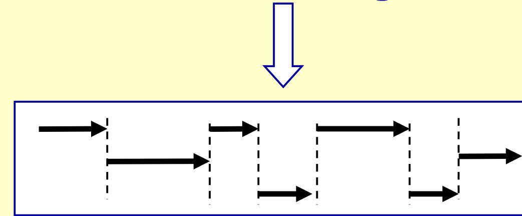
Echte Parallelität: Computer mit mehreren Prozessoren (Co-Prozessoren)

Kontrolle der Parallelität: zwei Formen

Betriebssystem



Anwender-Programm



„Leichtgewichtige“ Prozesse (Threads):

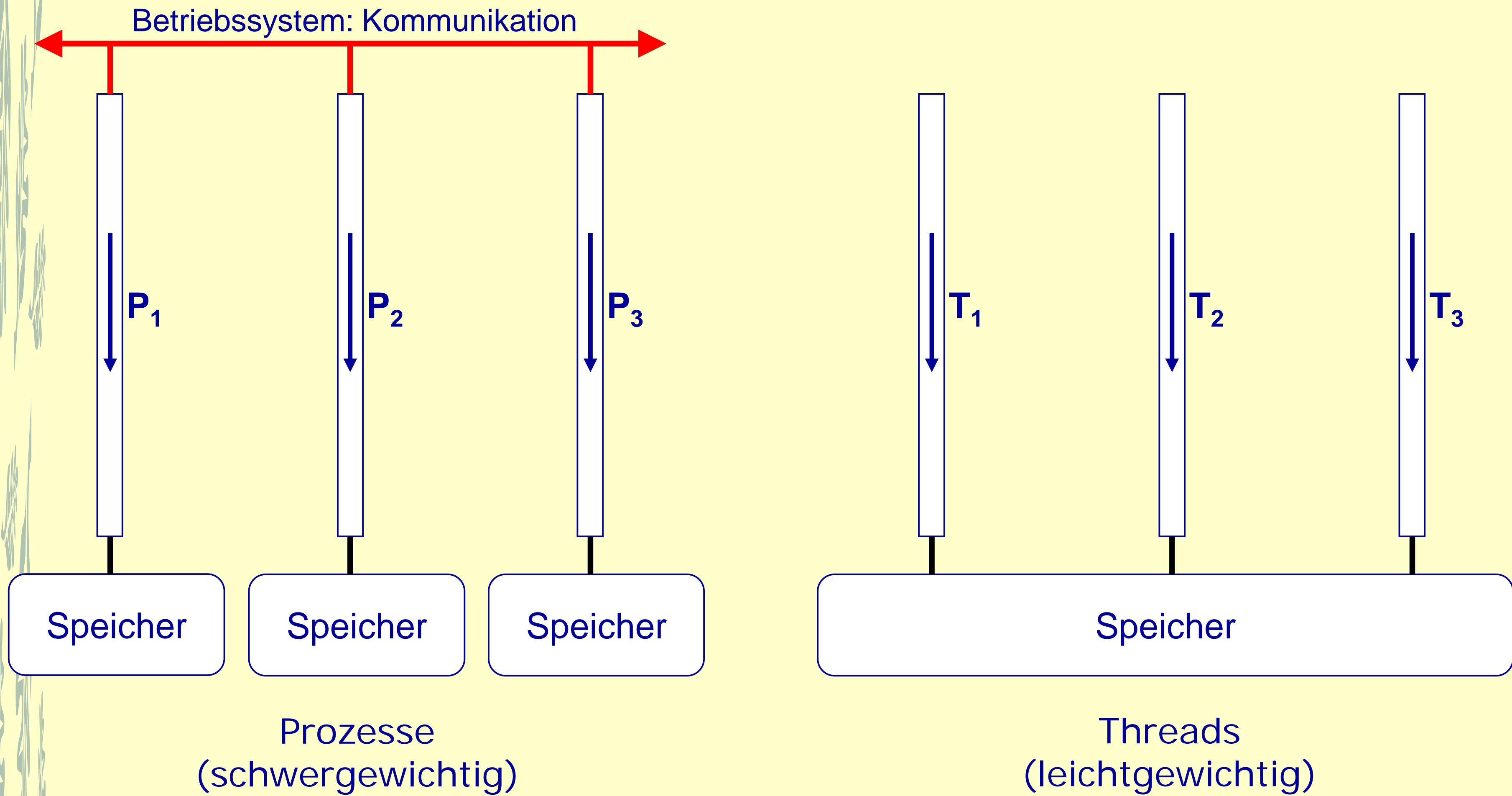
- Kommunikation über gemeinsamen Speicher
- unsicherer
- effizienter Nachrichtenaustausch

Thread = Faden

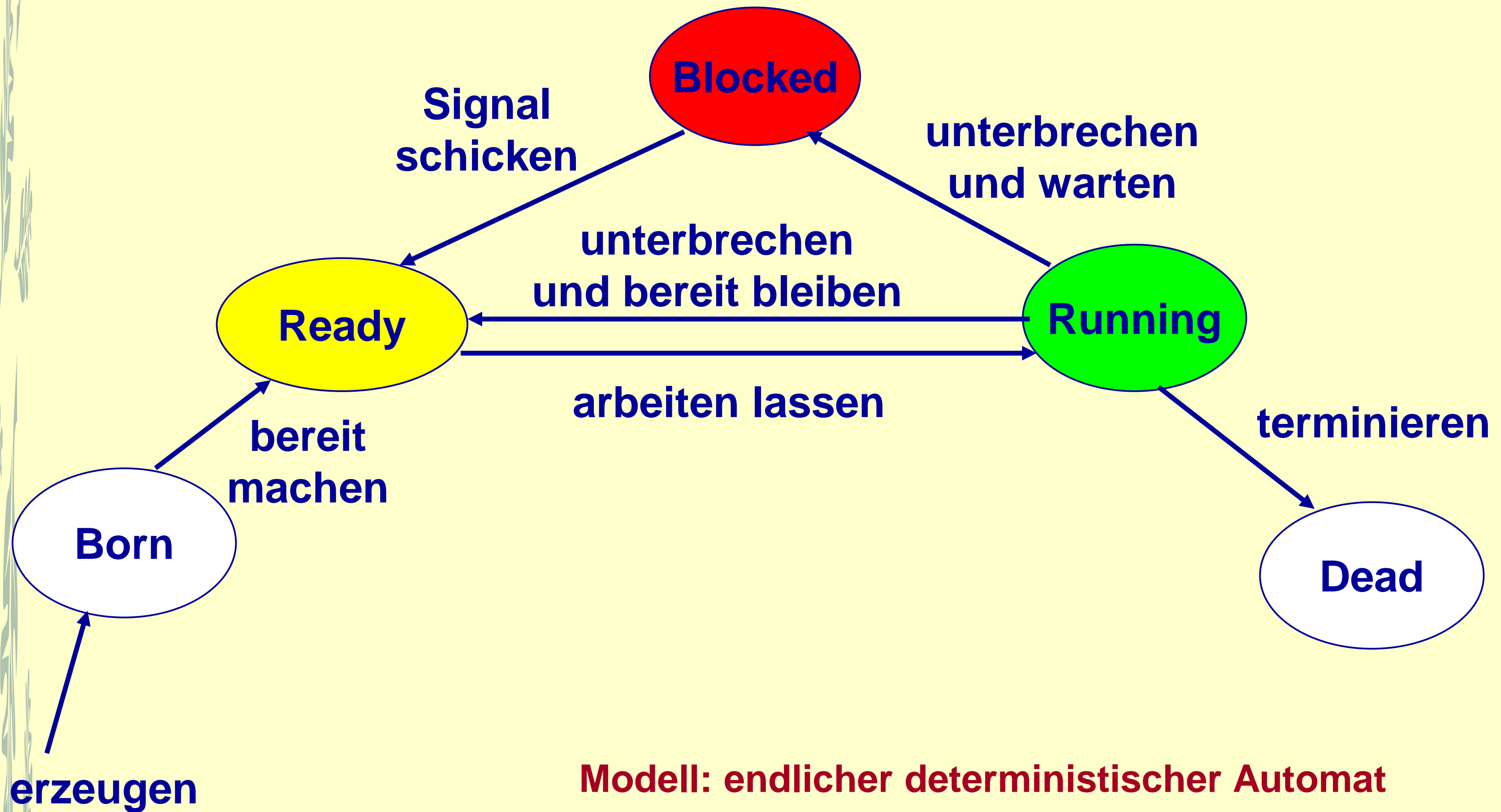
„Schwergewichtige“ Prozesse: Betriebssystem sorgt für Steuerung und Sicherheit

- jeder Prozess mit eigenem Speicher (Speicherbereich)
- Speicher vor Zugriffen anderer Prozesse geschützt
- Kommunikation aufwendig: Nachrichtenaustausch über das Betriebssystem

Prozesse & Threads

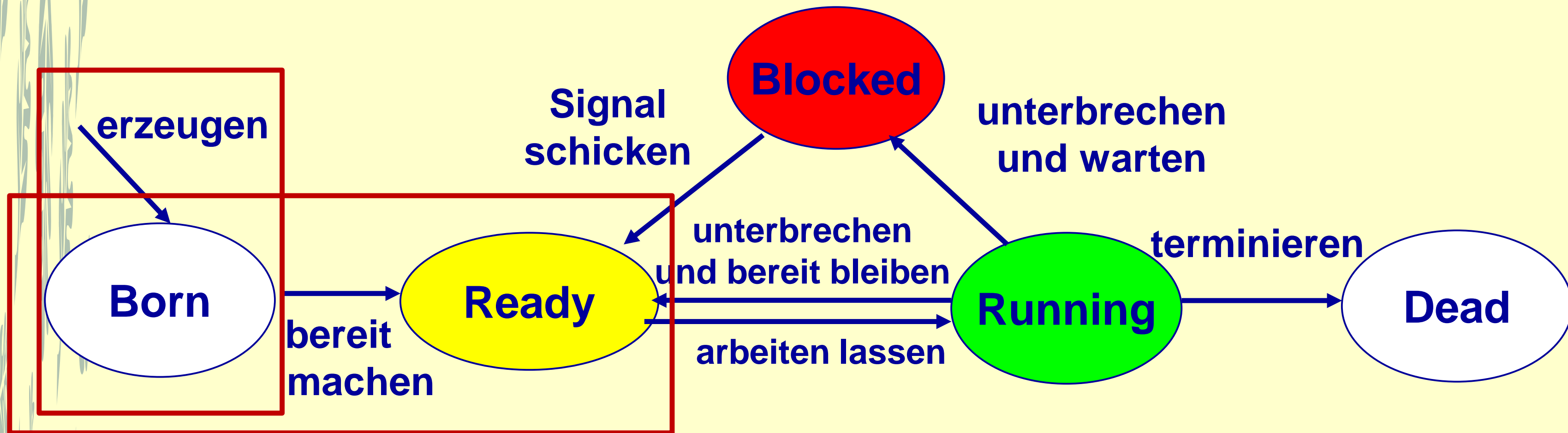


Lebenszyklus von Threads: Zustandsmodell



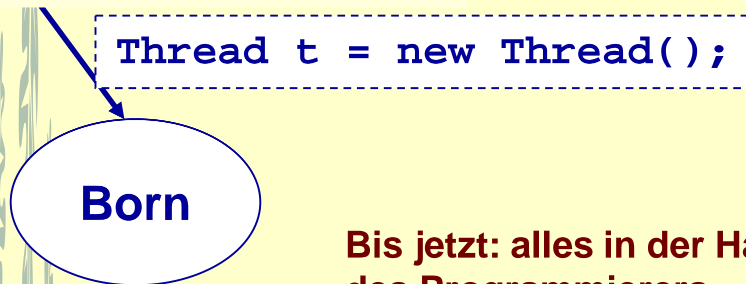
Modell: endlicher deterministischer Automat

Zustandsübergänge im Detail: erzeugen und bereit machen



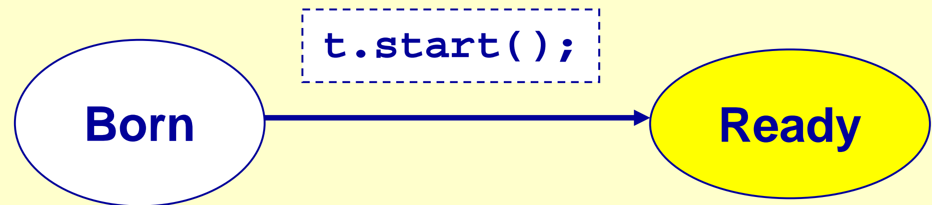
Thread erzeugen:
anderer aktiver Thread erzeugt neuen Thread t

```
Thread t = new Thread();
```



Thread *bereit machen* (starten?):
anderer aktiver Thread „startet“ neuen Thread t
(main ist zu Beginn da)

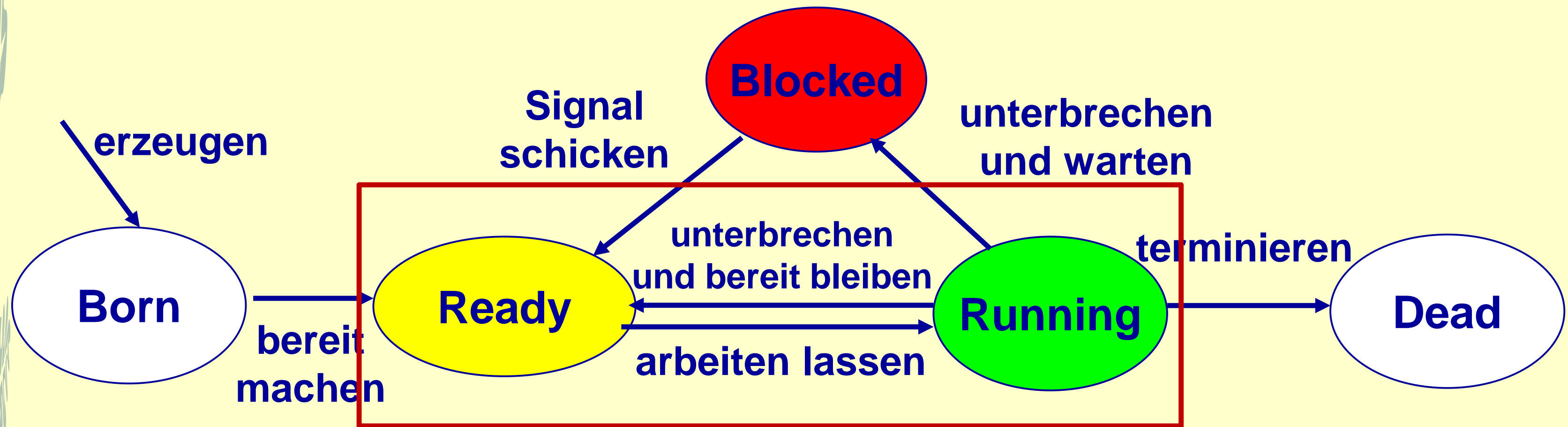
```
t.start();
```



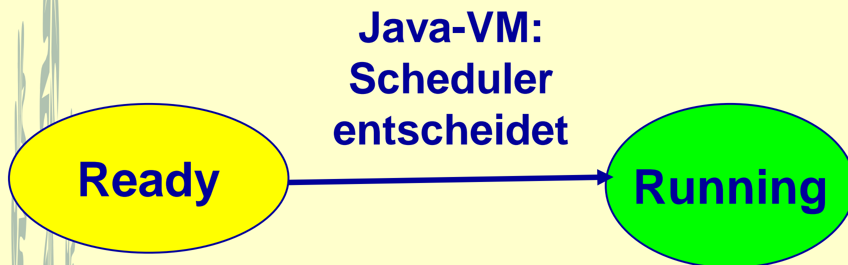
Bis jetzt: alles in der Hand des Programmierers

Anm.: start – bedeutet nicht mit der Arbeit beginnen

Zustandsübergänge im Detail: arbeiten lassen und unterbrechen

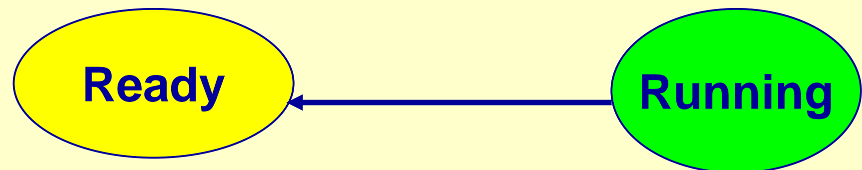


Thread arbeiten lassen:



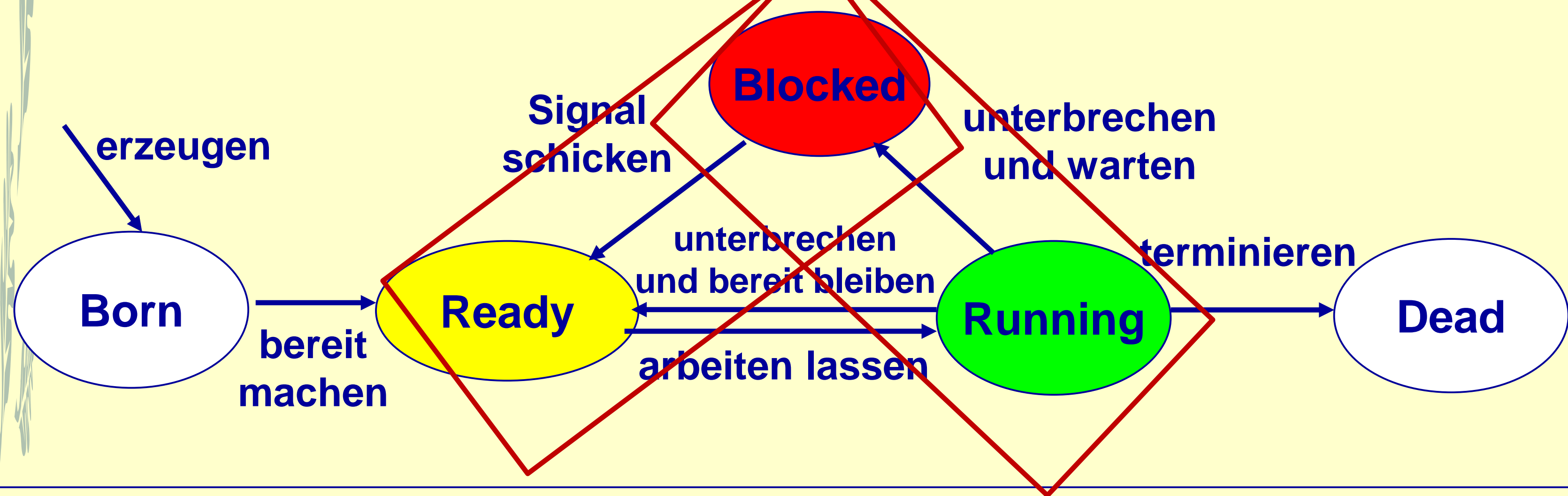
Thread unterbrechen und bereit bleiben lassen:

- Zeitabschnitt verbraucht
- Prozess mit höherer Priorität aufgetaucht
- Prozess gibt Prozessor freiwillig ab: yield()



Jetzt: alles in der Hand der JVM (Scheduler = „Zeitplaner“)

Zustandsübergänge im Detail: unterbrechen und warten



Unterbrechen und warten:

- Sich selbst schlafen legen: t.sleep(m)
- Auf das Ende eines anderen Threads u warten: u.join()
- Auf das Entsperren eines Objektes o warten: o.wait()

Erhalte Signal und sei wieder bereit:

- Schlafzeit beendet
- Anderer Thread weckt t: t.interrupt()
- Anderer Thread, auf dessen Ende gewartet wurde, „gestorben"
- Objekt o ist verfügbar

API-Klasse Thread (Auszug)

java.lang
Class Thread

java.lang.Object
└java.lang.Thread
All Implemented Interfaces:
Runnable

```
public class Thread
extends Object
implements Runnable
```

A *thread* is a thread of execution in a

Every thread has a priority. Threads
new Thread object, the new thread h

Since:
JDK1.0
See Also:
Runnable, Runtime.exit(int),

Thread = Faden

A *thread* is a thread of execution in a program.
The Java Virtual Machine allows an application
to have multiple threads of execution running concurrently.
Every thread has a priority …

Constructor Summary

Thread() Allocates a new Thread object.
Thread(Runnable target) Allocates a new Thread object.

Method Summary

static Th	
void	**interrupt**() Interrupts this thr
boolean	**isAlive**() Tests if this thread is alive
boolean	**isInterrupted**() Tests whether this thread
void	**join**() Waits for this thread to die
void	**join**(long millis) Waits at most millis mill
void	**run**() If this thread was constru and returns.
void	**setPriority**(int newPriority Changes the priority of thi
static void	**sleep**(long millis) Causes the currently executin
static void	**yield**() Causes the currently executing thread object to temporarily pause and allow other threads to execute.

```
static Thread currentThread()
void interrupt()
boolean isAlive()
boolean isInterrupted()
void join()
void join(long millis)
void run()
void setPriority(int newPriority)
static void sleep(long millis)
static void yield()
```

Weitere Methoden: geerbt

API-Klasse Object: Methoden für Threads

java.lang
Class Object

java.lang.Object

`public class` **`Object`**

Class `Object` is the root of the class hierarchy. Every class has `Object` as a superclass. All objects, including arrays, implement the methods of this class.

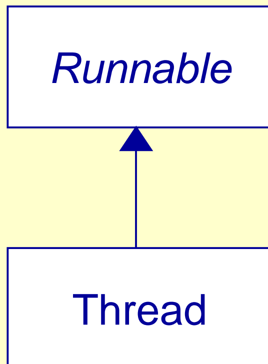
Since:
JDK1.0

See Also:
`Class`

Constructor Summary
`Object``()`

Method Summary	
`void`	**`notify`**`()` Wakes up a single thread that is waiting on this object's monitor.
`void`	**`notifyAll`**`()` Wakes up all threads that are waiting on this object's monitor.
`void`	**`wait`**`()` Causes current thread to wait until another thread invokes the `notify()` method or the `notifyAll()` method for this object.
`void`	**`wait`**`(long timeout)` Causes current thread to wait until either another thread invokes the `notify()` method or the `notifyAll()` method for this object, or a specified amount of time has elapsed.
`void`	**`wait`**`(long timeout, int nanos)` Causes current thread to wait until another thread invokes the `notify()` method or the `notifyAll()` method for this object, or some other thread interrupts the current thread, or a certain amount of real time has elapsed.

API-Klasse Runnable (Auszug)



java.lang

Interface Runnable

All Known Implementing Classes:
AsyncBoxView.ChildState, FutureTask, RenderableImageProducer, Thread, TimerTask

---

```
public interface Runnable
```

The `Runnable` interface should be implemented by any class whose instances are intended to be executed by a thread. The class must define a method of no arguments called `run`.

This interface is designed to provide a common protocol for objects that wish to execute code while they are active. For example, `Runnable` is implemented by class `Thread`. Being active simply means that a thread has been started and has not yet been stopped.

In addition, `Runnable` provides the means for a class to be active while not subclassing `Thread`. A class that implements `Runnable` can run without subclassing `Thread` by instantiating a `Thread` instance and passing itself in as the target. In most cases, the `Runnable` interface should be used if you are only planning to override the `run()` method and no other `Thread` methods. This is important because classes should not be subclassed unless the programmer intends on modifying or enhancing the fundamental behavior of the class.

Since:
JDK1.0

See Also:
`Thread`

Method Summary	
`void`	**`run`**`()` When an object implementing interface `Runnable` is used to create a thread, starting the thread causes the object's `run` method to be called in that separately executing thread.

Jeder Thread beginnt seine Arbeit mit der run-Methode
→ die main-Methode von Threads

Thread: wichtige eigene und geerbte Methoden (Auszug)

```
class Thread extends Object implements Runnable {

void start()                   Thread starten
void run()                     Programm des Threads (wie „main“)
void interrupt()               wecken
void join ()                   warten auf Ende dieses Threads
void join (long millisec)      warten ... maximal millisec

boolean isAlive()              Thread ist aktiv: gestarted,
                                  aber noch nicht tot?

int getPriority()              Priorität erfragen
void setPriority()             Priorität setzen

static Thread currentThread()            aktuelles Thread-Objekt

static void sleep (long milliseconds)    pausieren
static void yield()                      Kontrolle abgeben
}
```

!

Warum als Klassenmethoden? Es gibt immer nur einen Thread im Zustand "Running".

Prozesse als Objekte?

Prozesse (Threads):
- Entstammen dem Gebiet der dynamischen Abläufe
- Sind Algorithmen → imperative Programmierung

Java: Bezug zur Objektorientierung

Modell:
Zu jedem Thread gehört ein Objekt, das ihn kontrolliert
(Objekt = Einheit aus Daten und Operationen …)

→ Beide werden miteinander identifiziert:
„Thread t1“ – eigentlich das Objekt gemeint, das den Thread kontrolliert (eigene Daten: Zustand des Threads)

Grundprinzip: JVM startet einen „Ur-Thread“ (Basisprozess), der main() ausführt (main ist auch ein Thread)

Stack–Trace für Ausnahmebehandlung: main() ist ein Thread

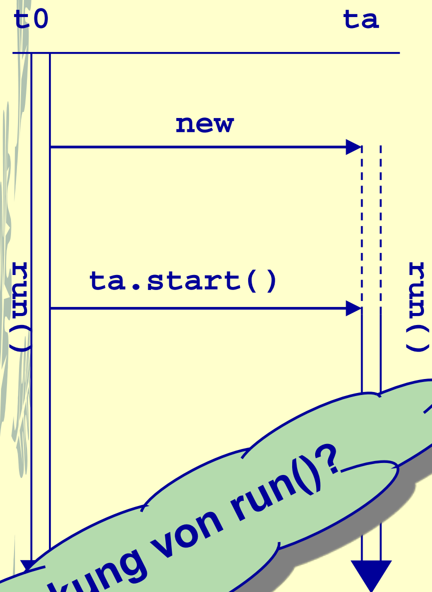
```
% java Ausnahme a1
Exception in thread "main" java.lang.NumberFormatException:a1
at java.lang.NumberFormatException(Unknown Source)
at java.lang.Integer.parseInt(Unknown Source)
at Ausnahme.makeIntFromString(Ausnahme.java:6)
at Ausnahme.main(Ausnahme.java:10)
```

Java™ Platform
Standard Ed. 6

Arbeit mit Threads:

- Erzeugen, starten, arbeiten, sterben
- Schlafen legen
- Warten
- Prioriät setzen

Erzeugen – Starten – Arbeiten – Sterben (1)



Erzeugen: der laufender Thread t0 erzeugt einen neuen
`ta = new ThreadA1();` `(Zustand: born)`

Starten: der laufende Thread startet den neuen
`ta.start();` `(Zustand: Ready)`

Arbeiten: JVM-Scheduler *(„Zeitplaner")* startet `run()`
`(Zustand: Running)`

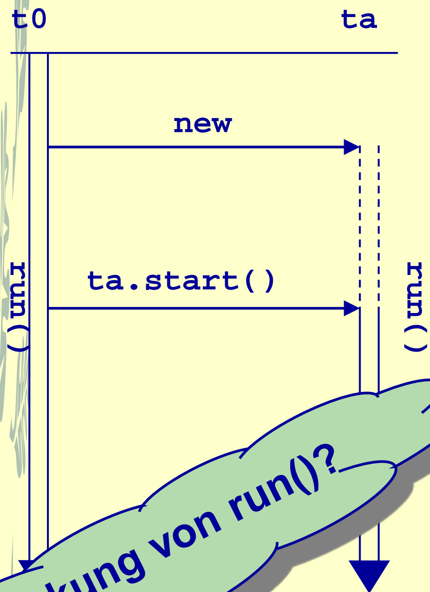
Sterben: run() beendet die Arbeit `(Zustand: Dead)`

ThreadBasicTest.java

```
class ThreadA1 extends Thread   {

  public void run()   {
    for (int i = 1; i < ThreadBasicTest.LIMIT; i++)   {
      System.out.println("A: " + i);
    }
    System.out.println("A done");
  }
}
```

21

Erzeugen – Starten – Arbeiten – Sterben (1)



Erzeugen: der laufender Thread t0 erzeugt ein

```
ta = new ThreadA1();    (Zustand: bo
```

Starten: der laufende Thread startet den neue

```
ta.start();      (Zustand: Ready)
```

Arbeiten: JVM-Scheduler *(„Zeitplaner“)* starte

```
                   (Zustand: Running)
```

Sterben: run() beendet die Arbeit `(Zustand:`

```
lass ThreadA1 extends Thread   {

  public void run()   {
    for (int i = 1; i < ThreadBasicTest.LIMIT; i++)
      System.out.println("A: " + i);
    }
    System.out.println("A done");
  }
}
```

21

ThreadBasicTe

```
A:  1
A:  2
A:  3
A:  4
A:  5
A:  6
A:  7
A:  8
A:  9
A:  10
A:  11
A:  12
A:  13
A:  14
A:  15
A:  16
A:  17
A:  18
A:  19
A:  20
A  done
```

Erzeugen – Starten – Arbeiten – Sterben (2)

```
public class ThreadBasicTest   {
   static final int LIMIT = 21;
   public static Thread ta;
   public static Thread tb;

   public static void main(String[] args) {
      ta = new ThreadA1();
      tb = new ThreadB1();
      ta.start();
      tb.start();
      System.out.println("   done...");
   }
}
```

Erzeugen: ein laufender Thread erzeugt einen neuen
`ta = new ThreadA1();`

Starten: der laufende Thread startet den neuen
`ta.start();`

Arbeiten: JVM-Scheduler startet run()

Sterben: run() beendet die Arbeit

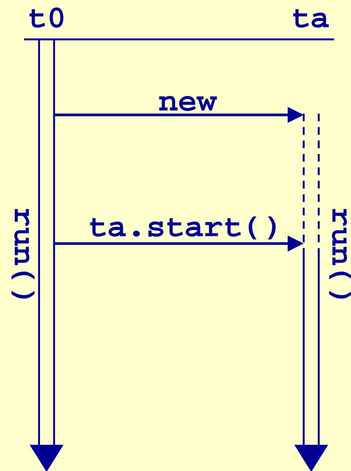
Wie viele Prozesse?
Welche Ausgabe ist zu erwarten?
Welche Reihenfolge für die drei "done"?

```
class ThreadA1 extends Thread   {

   public void run()   {
      for (int i = 1; i < ThreadBaiscTest.LIMIT; i++) {
         System.out.println("A: " + i);
      }
      System.out.println("A done");
   }
}
```

```
class ThreadB1 extends Thread   {

   public void run()   {
      for (int i = -1; i > -ThreadBasicTest.LIMIT; i--){
         System.out.println("\t\tB: " + i);
      }
      System.out.println("\t\tB done");
   }
}
```

Zwei triviale Threads: Zahlen ausgeben (1)

ThreadBasicTest.java




```
public class ThreadBasicTest   {
   static final int LIMIT = 21;
   public static Thread ta;
   public static Thread tb;

   public static void main(String[] args) {
      ta = new ThreadA1();
      tb = new ThreadB1();
      ta.start();
      tb.start();
      System.out.println("   done...");
   }
}
```

```
class ThreadA1 extends Thread   {

    public void run()   {
        for (int i = 1; i < ThreadBasicTest.LIMIT; i++) {
            System.out.println("A: " + i);
        }
        System.out.println("A done");
    }
}
```

```
class ThreadB1 extends Thread   {

    public void run()   {
        for (int i = -1; i > -ThreadBasicTest.LIMIT; i--) {
            System.out.println("\t\tB: " + i);
        }
        System.out.println("\t\tB done");
    }
}
```

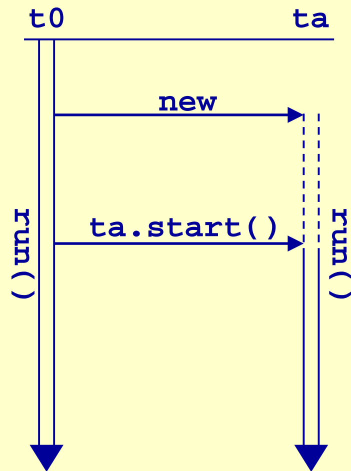
Ermittelte Ausgabe

```
A:  1
A:  2
A:  3
A:  4
A:  5
A:  6
A:  7
A:  8
                B:  -1
   done...
                B:  -2
                B:  -3
                B:  -4
                B:  -5
                B:  -6
                B:  -7
A:  9
A:  10
A:  11
A:  12
A:  13
A:  14
A:  15
A:  16
A:  17
A:  18
A:  19
A:  20
A  done
                B:  -8
                B:  -9
                B:  -10
                B:  -11
                B:  -12
                B:  -13
                B:  -14
                B:  -15
                B:  -16
                B:  -17
                B:  -18
                B:  -19
                B:  -20
                B  done
```

Ausgabe: Windows XP Pentium 4 2 GHz

Zwei triviale Threads: Zahlen ausgeben (2)

ThreadBasicTest.java




```
public class ThreadBasicTest   {
   static final int LIMIT = 21;
   public static Thread ta;
   public static Thread tb;

   public static void main(String[] args) {
      ta = new ThreadA1();
      tb = new ThreadB1();
      ta.start();
      tb.start();
      System.out.println("   done...");
   }
}
```

Zweite ermittelte Ausgabe

```
class ThreadA1 extends Thread   {

   public void run()   {
      for (int i = 1; i < ThreadBasicTest.LIMIT; i++) {
         System.out.println("A: " + i);
      }
      System.out.println("A done");
   }
}
```

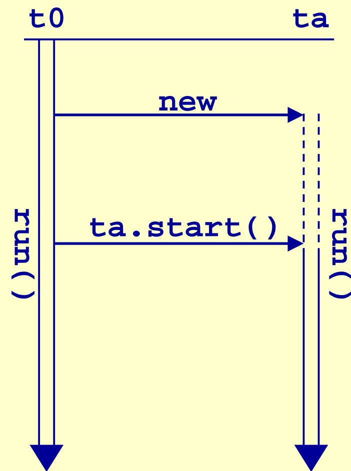
```
class ThreadB1 extends Thread   {

   public void run()   {
      for (int i = -1; i > -ThreadBasicTest.LIMIT; i--) {
         System.out.println("\t\tB: " + i);
      }
      System.out.println("\t\tB done");
   }
}
```

Ausgabe: Windows XP Pentium 4 2 GHz

```
   done...
A:  1
A:  2
A:  3
A:  4
A:  5
A:  6
A:  7
A:  8
                B:  -1
                B:  -2
                B:  -3
                B:  -4
                B:  -5
                B:  -6
                B:  -7
                B:  -8
                B:  -9
                B:  -10
                B:  -11
                B:  -12
                B:  -13
                B:  -14
                B:  -15
                B:  -16
                B:  -17
A:  9
A:  10
A:  11
A:  12
A:  13
A:  14
A:  15
A:  16
A:  17
                B:  -18
                B:  -19
                B:  -20
                B  done
A:  18
A:  19
A:  20
A  done
```

Zwei triviale Threads: Zahlen ausgeben (3)

ThreadBasicTest.java




```
public class ThreadBasicTest   {
   static final int LIMIT = 21;
   public static Thread ta;
   public static Thread tb;

   public static void main(String[] args) {
      ta = new ThreadA1();
      tb = new ThreadB1();
      ta.start();
      tb.start();
      System.out.println("   done...");
   }
}
```

Dritte ermittelte Ausgabe

```
class ThreadA1 extends Thread   {

   public void run()   {
      for (int i = 1; i < ThreadBasicTest.LIMIT; i++) {
         System.out.println("A: " + i);
      }
      System.out.println("A done");
   }
}
```

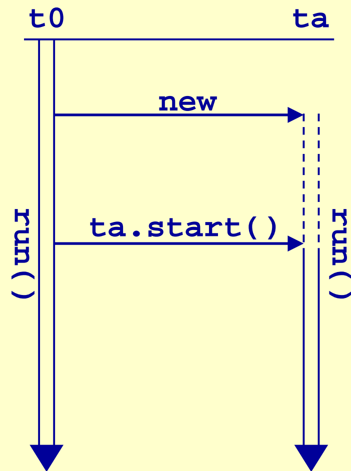
```
class ThreadB1 extends Thread   {

   public void run()   {
      for (int i = -1; i > -ThreadBasicTest.LIMIT; i--) {
         System.out.println("\t\tB: " + i);
      }
      System.out.println("\t\tB done");
   }
}
```

Ausgabe: Windows XP Pentium 4 2 GHz

```
   done...
A: 1
A: 2
A: 3
A: 4
A: 5
A: 6
A: 7
A: 8
A: 9
A: 10
A: 11
A: 12
A: 13
A: 14
A: 15
A: 16
                B: -1
                B: -2
                B: -3
                B: -4
                B: -5
                B: -6
                B: -7
                B: -8
                B: -9
                B: -10
                B: -11
                B: -12
                B: -13
                B: -14
                B: -15
                B: -16
                B: -17
A: 17
A: 18
A: 19
A: 20
A done
                B: -18
                B: -19
                B: -20
                B done
```

Zwei triviale Threads: Zahlen ausgeben (4)

ThreadBasicTest.java




```
public class ThreadBasicTest   {
   static final int LIMIT = 21;
   public static Thread ta;
   public static Thread tb;

   public static void main(String[] args) {
      ta = new ThreadA1();
      tb = new ThreadB1();
      ta.start();
      tb.start();
      System.out.println("   done...");
   }
}
```

Vierte ermittelte Ausgabe

```
class ThreadA1 extends Thread   {

   public void run()   {
      for (int i = 1; i < ThreadBasicTest.LIMIT; i++) {
         System.out.println("A: " + i);
      }
      System.out.println("A done");
   }
}
```

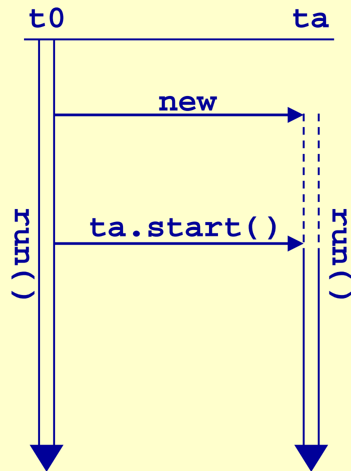
```
class ThreadB1 extends Thread   {

   public void run()   {
      for (int i = -1; i > -ThreadBasicTest.LIMIT; i--) {
         System.out.println("\t\tB: " + i);
      }
      System.out.println("\t\tB done");
   }
}
```

Ausgabe: Windows XP Pentium 4 2 GHz

```
A:  1
A:  2
A:  3
A:  4
A:  5
A:  6
A:  7
A:  8
A:  9
A:  10
A:  11
A:  12
A:  13
A:  14
A:  15
A:  16
A:  17
    done ...
A:  18
A:  19
A:  20
A  done
                    B:  -1
                    B:  -2
                    B:  -3
                    B:  -4
                    B:  -5
                    B:  -6
                    B:  -7
                    B:  -8
                    B:  -9
                    B:  -10
                    B:  -11
                    B:  -12
                    B:  -13
                    B:  -14
                    B:  -15
                    B:  -16
                    B:  -17
                    B:  -18
                    B:  -19
                    B:  -20
                    B  done
```

Zwei triviale Threads: Zahlen ausgeben (5)

ThreadBasicTest.java




```
public class ThreadBasicTest   {
   static final int LIMIT = 21;
   public static Thread ta;
   public static Thread tb;

   public static void main(String[]
      ta = new ThreadA1();
      tb = new ThreadB1();
      ta.start();
      tb.start();
      System.out.println("   done...");
   }
}
```

Fünfte ermittelte Ausgabe: 10. 2. 2012

```
class ThreadA1 extends Thread   {

   public void run()   {
      for (int i = 1; i < ThreadBasicTest.LIMIT; i++) {
         System.out.println("A: " + i);
      }
      System.out.println("A done");
   }
}
```

```
class ThreadB1 extends Thread   {

   public void run()   {
      for (int i = -1; i > -ThreadBasicTest.LIMIT; i--) {
         System.out.println("\t\tB: " + i);
      }
      System.out.println("\t\tB done");
   }
}
```

Ausgabe: Windows 7 Intel Core 2 Duo SP9400 (2-Prozessor-Maschine) 2,40GHz

```
done ...
                B:  - 1
                B:  - 2
                B:  - 3
                B:  - 4
                B:  - 5
                B:  - 6
                B:  - 7
                B:  - 8
                B:  - 9
                B:  - 10
                B:  - 11
                B:  - 12
                B:  - 13
A:  1
A:  2
A:  3
A:  4
A:  5
A:  6
A:  7
A:  8
A:  9
A:  10
A:  11
A:  12
A:  13
A:  14
A:  15
A:  16
A:  17
A:  18
A:  19
A:  20
A  done
                B:  - 14
                B:  - 15
                B:  - 16
                B:  - 17
                B:  - 18
                B:  - 19
                B:  - 20
                B  done
```

Zwei triviale Threads: Vergleich aller Resultate

ThreadBasicTest.java

```
public class ThreadBasicTest {
```

```
A:  1
A:  2
A:  3
A:  4
A:  5
A:  6
A:  7
A:  8
                B:  -1
   done. . .
                B:  -2
                B:  -3
                B:  -4
                B:  -5
                B:  -6
                B:  -7
A:  9
A:  10
A:  11
A:  12
A:  13
A:  14
A:  15
A:  16
A:  17
A:  18
A:  19
A:  20
A  done
                B:  -8
                B:  -9
                B:  -10
                B:  -11
                B:  -12
                B:  -13
                B:  -14
                B:  -15
                B:  -16
                B:  -17
                B:  -18
                B:  -19
                B:  -20
                B  done
```

```
   done. . .
A:  1
A:  2
A:  3
A:  4
A:  5
A:  6
A:  7
A:  8
                B:  -1
                B:  -2
                B:  -3
                B:  -4
                B:  -5
                B:  -6
                B:  -7
                B:  -8
                B:  -9
                B:  -10
                B:  -11
                B:  -12
                B:  -13
                B:  -14
                B:  -15
                B:  -16
                B:  -17
A:  9
A:  10
A:  11
A:  12
A:  13
A:  14
A:  15
A:  16
A:  17
                B:  -18
                B:  -19
                B:  -20
                B  done
A:  18
A:  19
A:  20
A  done
```

```
   done. . .
A:  1
A:  2
A:  3
A:  4
A:  5
A:  6
A:  7
A:  8
A:  9
A:  10
A:  11
A:  12
A:  13
A:  14
A:  15
A:  16
                B:  -1
                B:  -2
                B:  -3
                B:  -4
                B:  -5
                B:  -6
                B:  -7
                B:  -8
                B:  -9
                B:  -10
                B:  -11
                B:  -12
                B:  -13
                B:  -14
                B:  -15
                B:  -16
                B:  -17
A:  17
A:  18
A:  19
A:  20
A  done
                B:  -18
                B:  -19
                B:  -20
                B  done
```

```
A:  1
A:  2
A:  3
A:  4
A:  5
A:  6
A:  7
A:  8
A:  9
A:  10
A:  11
A:  12
A:  13
A:  14
A:  15
A:  16
A:  17
   done . . .
A:  18
A:  19
A:  20
A  done
                B:  -1
                B:  -2
                B:  -3
                B:  -4
                B:  -5
                B:  -6
                B:  -7
                B:  -8
                B:  -9
                B:  -10
                B:  -11
                B:  -12
                B:  -13
                B:  -14
                B:  -15
                B:  -16
                B:  -17
                B:  -18
                B:  -19
                B:  -20
                B  done
```

```
done ...
                B:  -1
                B:  -2
                B:  -3
                B:  -4
                B:  -5
                B:  -6
                B:  -7
                B:  -8
                B:  -9
                B:  -10
                B:  -11
                B:  -12
                B:  -13
A:  1
A:  2
A:  3
A:  4
A:  5
A:  6
A:  7
A:  8
A:  9
A:  10
A:  11
A:  12
A:  13
A:  14
A:  15
A:  16
A:  17
A:  18
A:  19
A:  20
A  done
                B:  -14
                B:  -15
                B:  -16
                B:  -17
                B:  -18
                B:  -19
                B:  -20
                B  done
```

Zwei triviale Threads

Erste Ausgabe

```
A:  1
A:  2
A:  3
A:  4
A:  5
A:  6
A:  7
A:  8
                  B:  - 1
   done. . .
                  B:  - 2
                  B:  - 3
                  B:  - 4
                  B:  - 5
                  B:  - 6
                  B:  - 7
A:  9
A:  10
A:  11
A:  12
A:  13
A:  14
A:  15
A:  16
A:  17
A:  18
A:  19
A:  20
A  done
                  B:  - 8
                  B:  - 9
                  B:  - 10
                  B:  - 11
                  B:  - 12
                  B:  - 13
                  B:  - 14
                  B:  - 15
                  B:  - 16
                  B:  - 17
                  B:  - 18
                  B:  - 19
                  B:  - 20
                  B  done
```

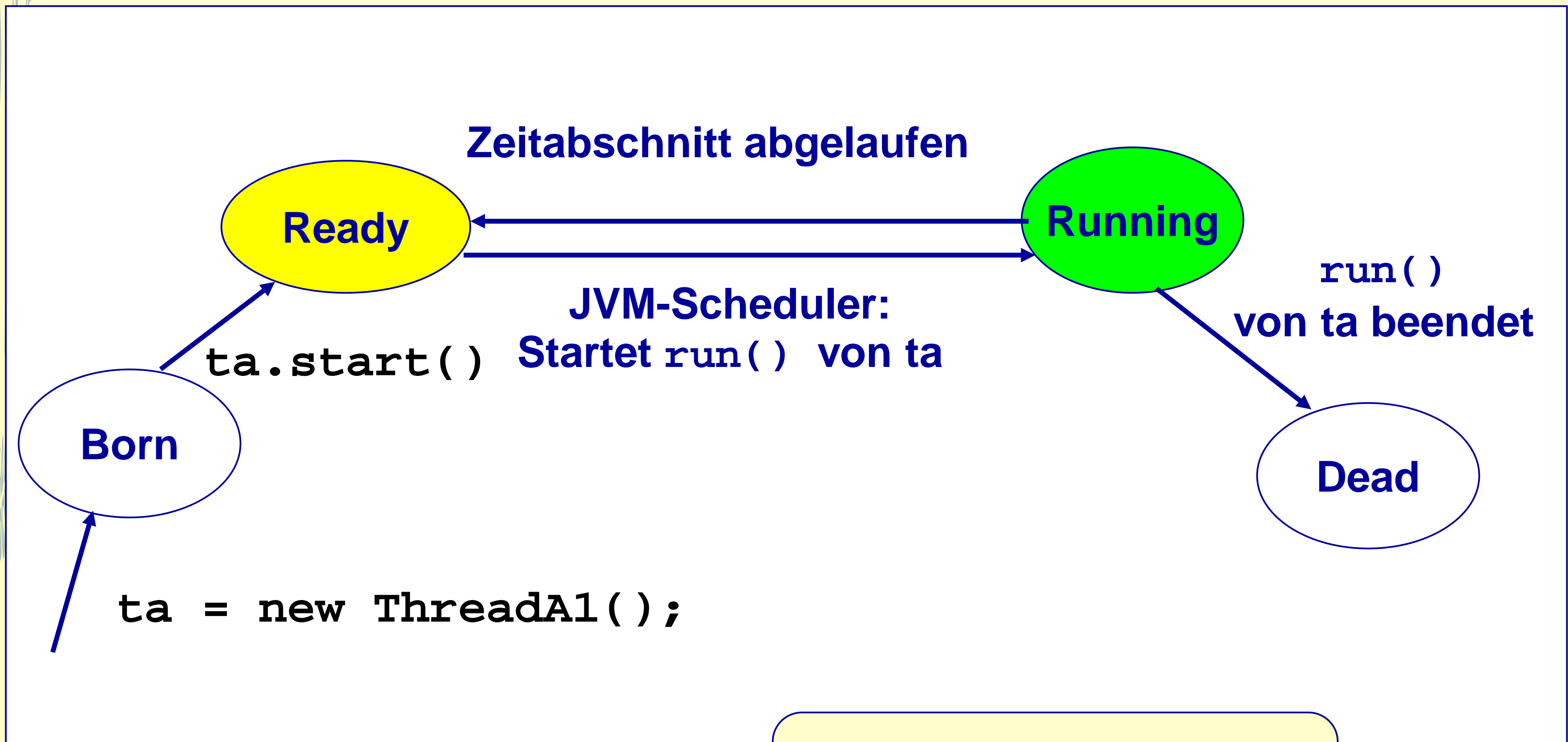
Sechste ermittelte Ausgabe: 9. 1. 2016

```
done...
            B: -1
            B: -2
            B: -3
            B: -4
            B: -5
A: 1
A: 2
            B: -6
A: 3
            B: -7
A: 4
            B: -8
A: 5
            B: -9
A: 6
            B: -10
A: 7
            B: -11
A: 8
A: 9
            B: -12
A: 10
            B: -13
A: 11
            B: -14
A: 12
            B: -15
A: 13
A: 14
A: 15
            B: -16
A: 16
A: 17
A: 18
            B: -17
            B: -18
            B: -19
A: 19
A: 20
            B: -20
            B done
A done
```

Siebente ermittelte Ausgabe: 2. 2. 2016

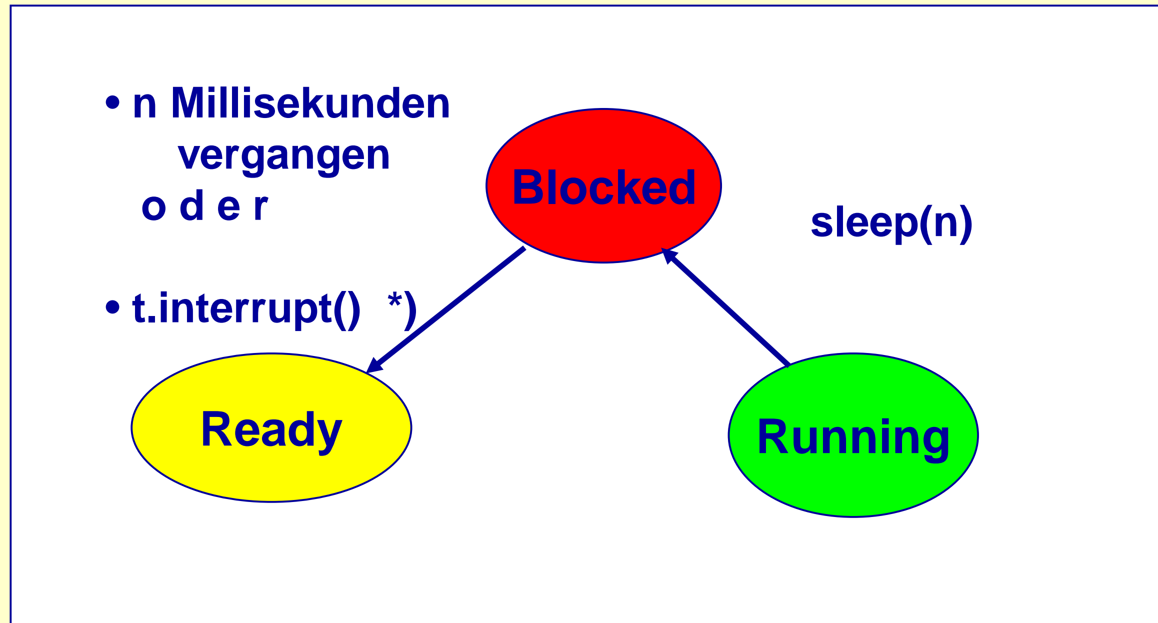
```
done...
            B: -1
            B: -2
            B: -3
            B: -4
            B: -5
            B: -6
            B: -7
A: 1
            B: -8
            B: -9
            B: -10
            B: -11
            B: -12
            B: -13
            B: -14
            B: -15
            B: -16
            B: -17
            B: -18
A: 2
            B: -19
            B: -20
            B done
A: 3
A: 4
A: 5
A: 6
A: 7
A: 8
A: 9
A: 10
A: 11
A: 12
A: 13
A: 14
A: 15
A: 16
A: 17
A: 18
A: 19
A: 20
A done
```

Erzeugen – Starten – Arbeiten – Sterben: Zustandsübergänge im Beispiel für ta (Zusammenfassung)



Im Beispiel: 3 Prozesse;
Im Bild: Zustände für ta

Sleep:
Threads unterbrechen ihre Arbeit selbst



→ Ausnahme InterruptedException ausgelöst (mit sleep)
→ sleep muss in try-catch eingebettet sein
(sonst: Compilationsfehler)

*) anderer Thread kann t vorzeitig wecken:
t.interrupt()

Thread-Beispiel mit sleep()

ThreadSleep.java

```
class ThreadA2 extends Thread   {

   public void run()   {
      for (int i = 1; i < ThreadSleep.LIMIT; i++)   {
         try   {
          sleep(60);
         } catch(InterruptedException e) {}
         System.out.println("A: " + i);
      }
      System.out.println("A done");
   }
}
```

```
class ThreadB2 extends Thread   {

   public void run()   {
      for (int i = -1; i > -ThreadSleep.LIMIT; i--)  {
         try   {
          sleep(40);
         } catch(InterruptedException e) {}
         System.out.println("\t\tB: " + i);
      }
      System.out.println("\t\tB done");
   }
}
```

```
done...
              B: -1
A: 1
              B: -2
A: 2
              B: -3
              B: -4
A: 3
              B: -5
A: 4
              B: -6
              B: -7
A: 5
              B: -8
A: 6
              B: -9
A: 7
              B: -10
              B: -11
A: 8
              B: -12
A: 9
              B: -13
              B: -14
A: 10
              B: -15
A: 11
              B: -16
              B: -17
A: 12
              B: -18
A: 13
              B: -19
              B: -20
              B done
A: 14
A: 15
A: 16
A: 17
A: 18
A: 19
A: 20
A done
```

Ausgabe?

Reihenfolge der drei "done"?

SpotTest: ein Applet

```
public class SpotTest extends Applet {

  /* SpotTest      J M Bishop Aug 2000
   * ========
   *
   * Draws spots of different colours
   *
   * Illustrates simple threads
   */

    int mx, my; //upper left point and ...
    int radius = 10; //... radius of spot
    int boardSize; //size of applet window
    int change;

    public void init() {
      boardSize = getSize().width - 1;
      change = boardSize-radius;

      // creates and starts three threads
      new Spots(Color.red).start();
      new Spots(Color.blue).start();
      new Spots(Color.green).start();
    }
```

```
 class Spots extends Thread {
    Color colour;

   Spots(Color c) {
        colour = c;
   }

   public void run () {
        while (true) {
           draw();
           try {
            sleep (500); // millisecs
           }
           catch (InterruptedException e) {}
        }
   }

   public void draw() {
        Graphics g = getGraphics();
        g.setColor(colour);
        // calculate a new place for a spot
        // and draw it.
        mx = (int)(Math.random()*1000) % change;
        my = (int)(Math.random()*1000) % change;
        g.fillOval(mx, my, radius, radius);
   }
}
```

SpotTest.java

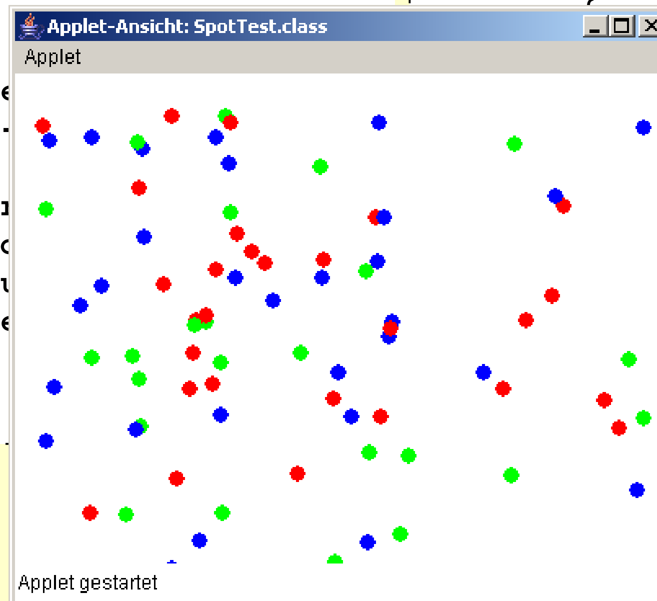
Ausgabe?

SpotTest: Applet mit Ausgabebeispiel

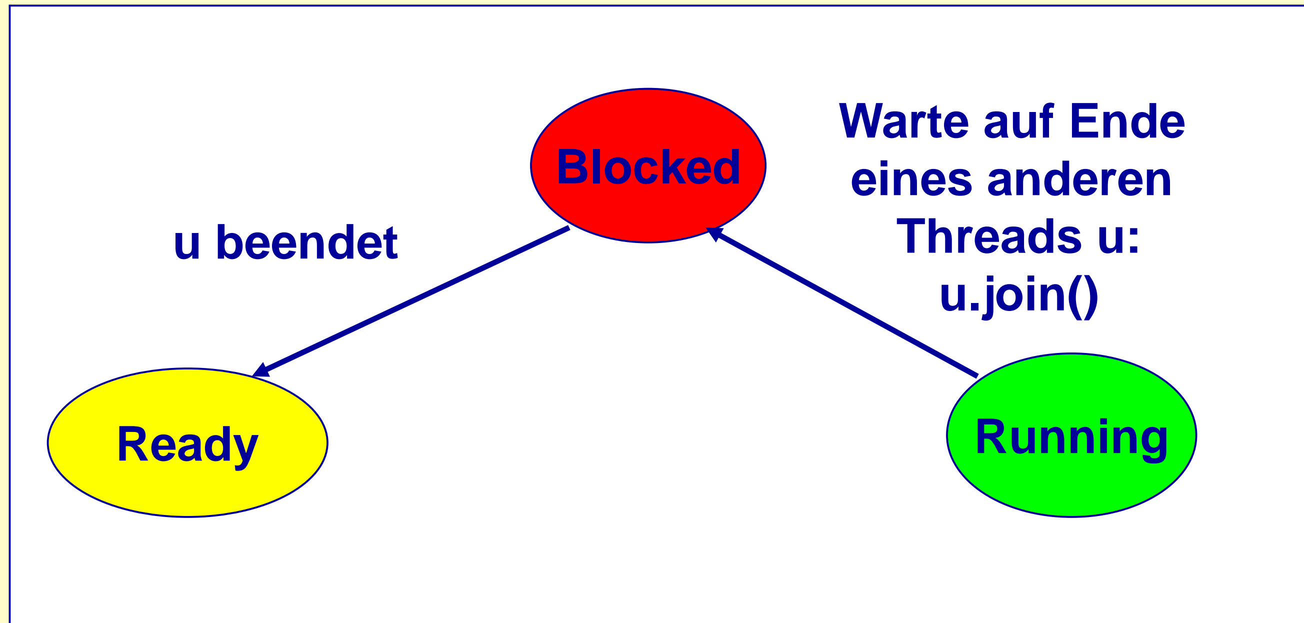
```
public class SpotTest extends Applet {

  /* SpotTest      J M Bishop Aug 2000
   * ========
   *
   * Draws spots of different colours
   *
   * Illustrates simple threads
   */

    int mx, my;
    int radius = 10;
    int boardSize = 200;
    int change;

    public void init() {
      boardSize = getSize
      change = boardSize-

      // creates and star
      new Spots(Color.red
      new Spots(Color.blu
      new Spots(Color.gre
    }
```

```
class Spots extends Thread {
   Color colour;

   Spots(Color c) {
        colour = c;
   }

   public void run () {
        while (true) {
           draw();
           try {
            sleep (500); // millisecs
           }
           catch (InterruptedException e) {}
        }

   ...d draw() {
   ...ics g = getGraphics();
   ...Color(colour);
   ...lculate a new place for a spot
   ...d draw it.
   ...(int)(Math.random()*1000) % change;
   ...(int)(Math.random()*1000) % change;
   ...lOval(mx, my, radius, radius);
```

SpotTest.java



Join:
Warten auf das Ende eines anderen Threads



Sinnvolle Weiterarbeit erst möglich, falls Arbeit von u beendet ist.

Beispiel: sinnvolle Weiterarbeit erst nach Ende eines anderen Threads

Warte auf Ende eines Prozesses ‚Sortiere'

```
Sortiere sort = new Sortiere();
. . .
sort.start();  //sortiere grosses Array
// weitere Aktivitaeten:
. . .
sort.join(); //jetzt: warte auf Ende der Sortierung
// nun: Zugriff auf sortiertes Array
. . .
```

Prozess ‚Sortiere'

```
class Sortiere extends Thread {
   public void run() {
     quicksort(…);
   }
}
```

Thread-Beispiel mit join()

ThreadJoin.java

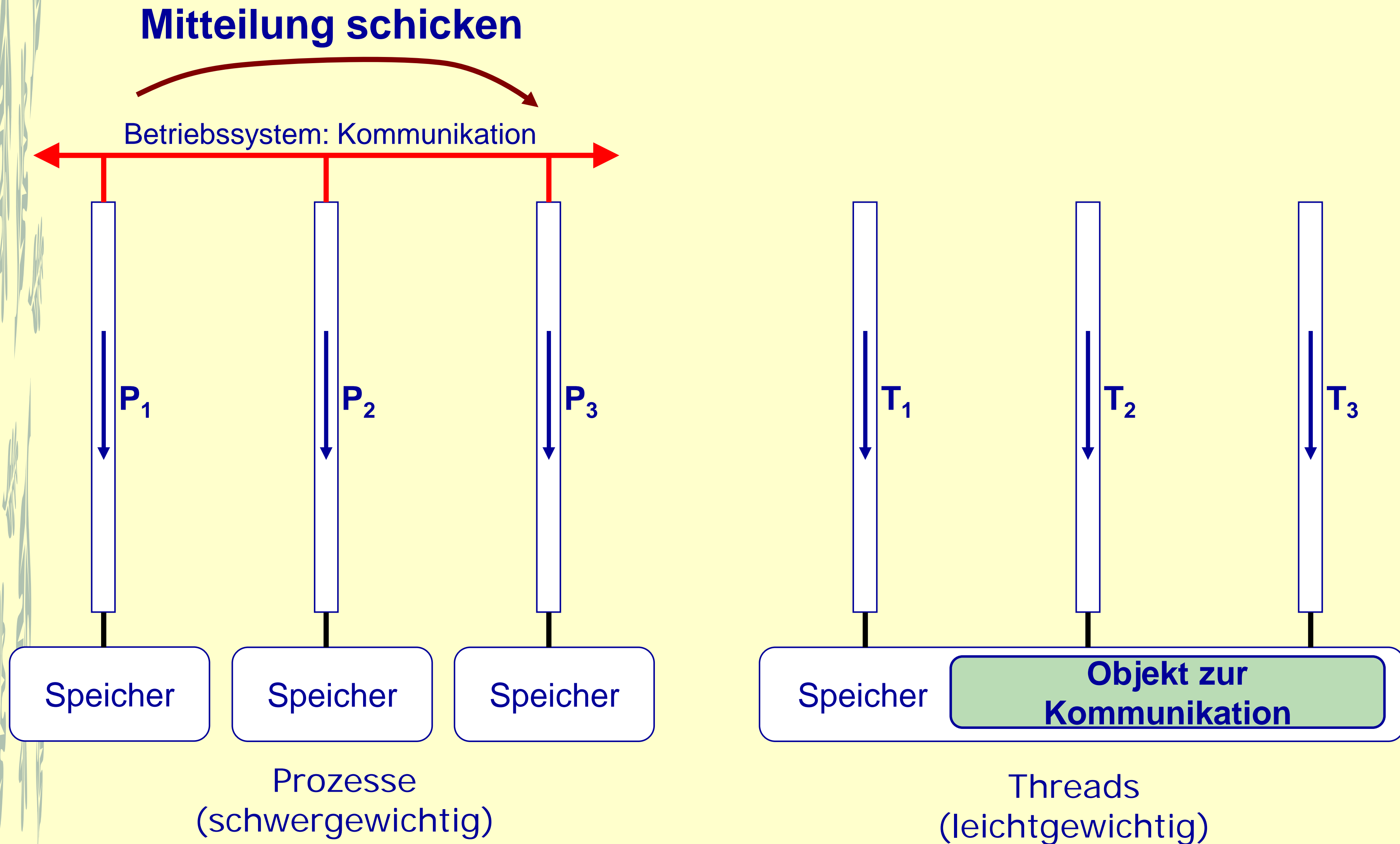
```
class ThreadB3 extends Thread   {

  public void run()   {
    for (int i = -1; i > -ThreadJoin.LIMIT/2; i--)
      System.out.println("\t\tB: " + i);

    try {
      ThreadJoin.ta.join();
    } catch (InterruptedException e) {}
    System.out.println("\t\tB done");
  }
}
```

```
done. . .
A:  1
A:  2
A:  3
A:  4
A:  5
A:  6
A:  7
A:  8
            B:  -1
            B:  -2
            B:  -3
            B:  -4
            B:  -5
            B:  -6
            B:  -7
            B:  -8
            B:  -9

A:  9
A:  10
A:  11
A:  12
A:  13
A:  14
A:  15
A:  16
A:  17
A:  18
A:  19
A:  20
A  done
            B  done
```

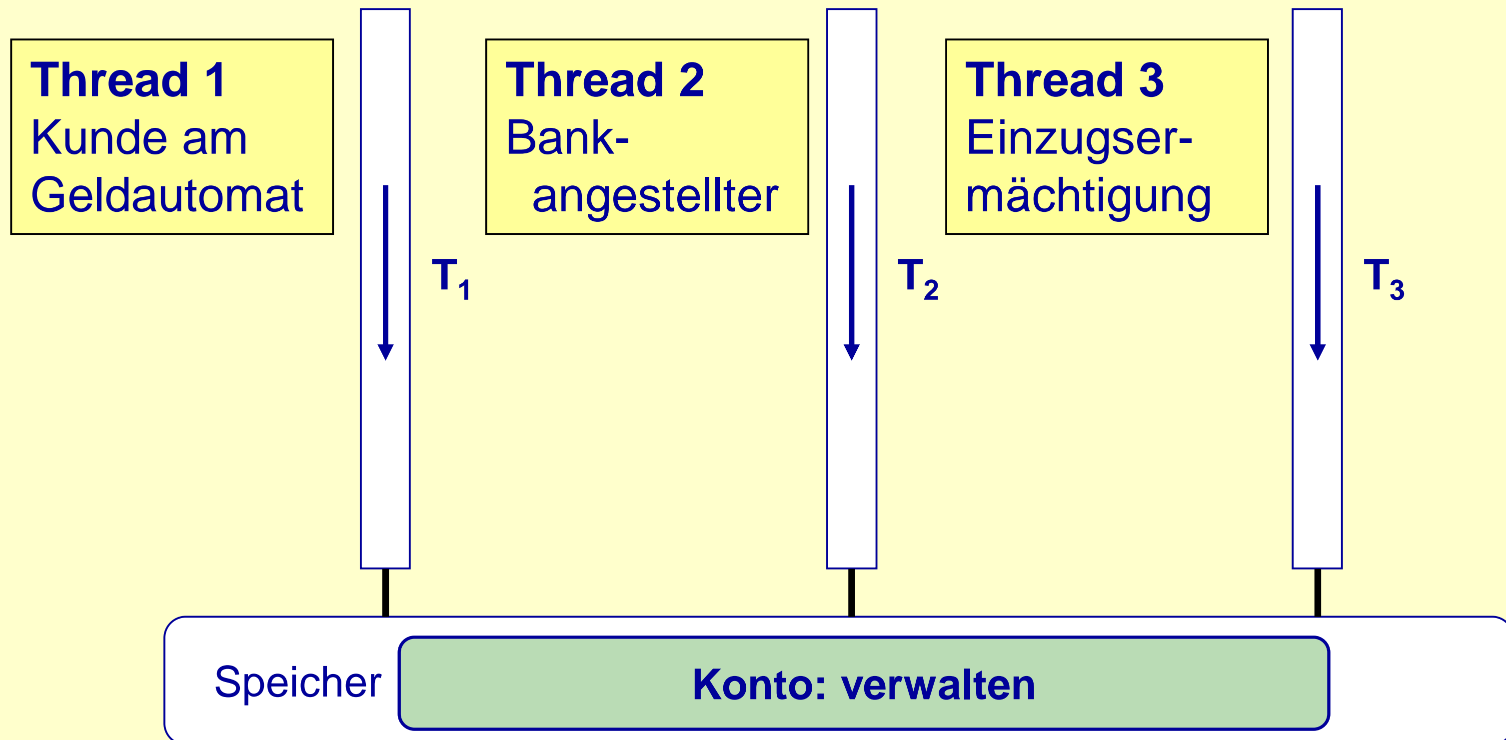
Ausgabe?

Kommunikation und Synchronisation

Kommunikation zwischen Prozessen



Kommunikation zwischen Prozessen: Konto verwalten



Kommunikation zwischen Threads: gemeinsame Speicher = gemeinsame Objekte

Konto:

```
class Account  {
  private long balance;

  void deposit (long amount)  {
    long aux = this.balance;
    aux = aux + amount;
    this.balance = aux;
  }

  void withdraw (long amount)  {
    long aux = this.balance;
    if (aux >= amount)  {
      aux = aux - amount;
      this.balance = aux;
    }
  }
}

Account acc = new . . .;
```

Threads:
Verschiedene Nutzer des Kontos
- Kunde am Geldautomat
- Bankangestellter
- Einzugsermächtigung

Kunde:
`acc.deposit(200)`

Einzugsermächtigung:
`acc.withdraw(200)`

Kommunikation zwischen Threads: Synchronisationsproblem

Konto:

```
class Account  {
   private long balance;

   void deposit (long amount)  {
      long aux = this.balance;
      aux = aux + amount;
      this.balance = aux;
   }

   void withdraw (long amount)  {
      long aux = this.balance;
      if (aux >= amount)  {
         aux = aux - amount;
         this.balance = aux;
      }
   }
}

Account acc = new . .
```

Problem:
- Methoden teilbar:
  Zeitabschnitt des Thread kann mitten in Methode enden

Kunde:
`acc.deposit(200)`

Einzugsermächtigung:
`acc.withdraw(200)`

Kunde: Tread t1	Einzugsermächtigung: t2	Konto
acc.deposit(200) *(aux)*	acc.withdraw(200) *(aux)*	1000
long aux = balance; *(1000)*		
aux = aux + amount; *(1200)*		
	long aux = balance; *(1000)*	
	aux = aux- amount; *(800)*	
balance = aux;		1200
	balance = aux;	800

Synchronisierte Methoden

```
class Account  {
  private long balance;

  synchronized void deposit (long amount)  {
    long aux = this.balance;
    aux = aux + amount;
    this.balance = aux;
  }

  synchronized void withdraw (long amount)  {
    long aux = this.balance;
    if (aux >= amount)  {
      aux = aux - amount;
      this.balance = aux;
    }
  }

}
```

Synchronisierte Methoden:
Wenn ein Thread eine synchronisierte Methode ausführt, so erhält er einen „Lock“ auf das Objekt:
- Kein anderer Thread hat Zugriff auf das Objekt
- Synchronisierte Methode wird vollständig beendet