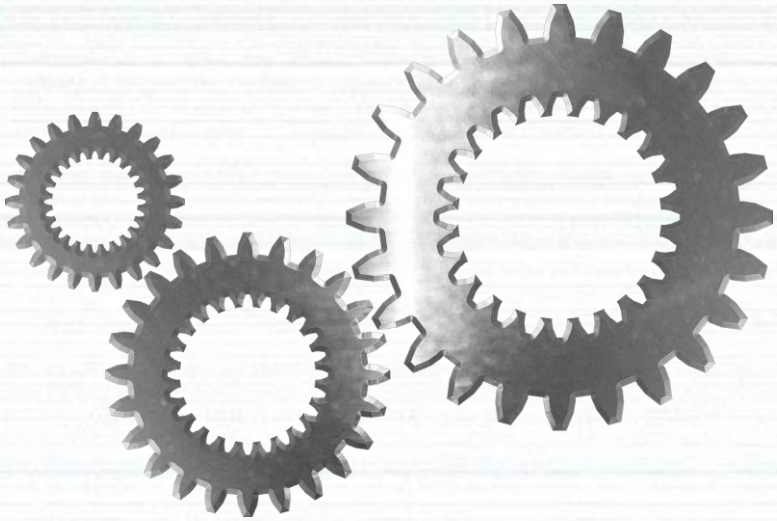
Prozesse und Threads



Was sind die Aufgaben eines Betriebssystems?

Wer hat schon mal was von Prozessen gehört?

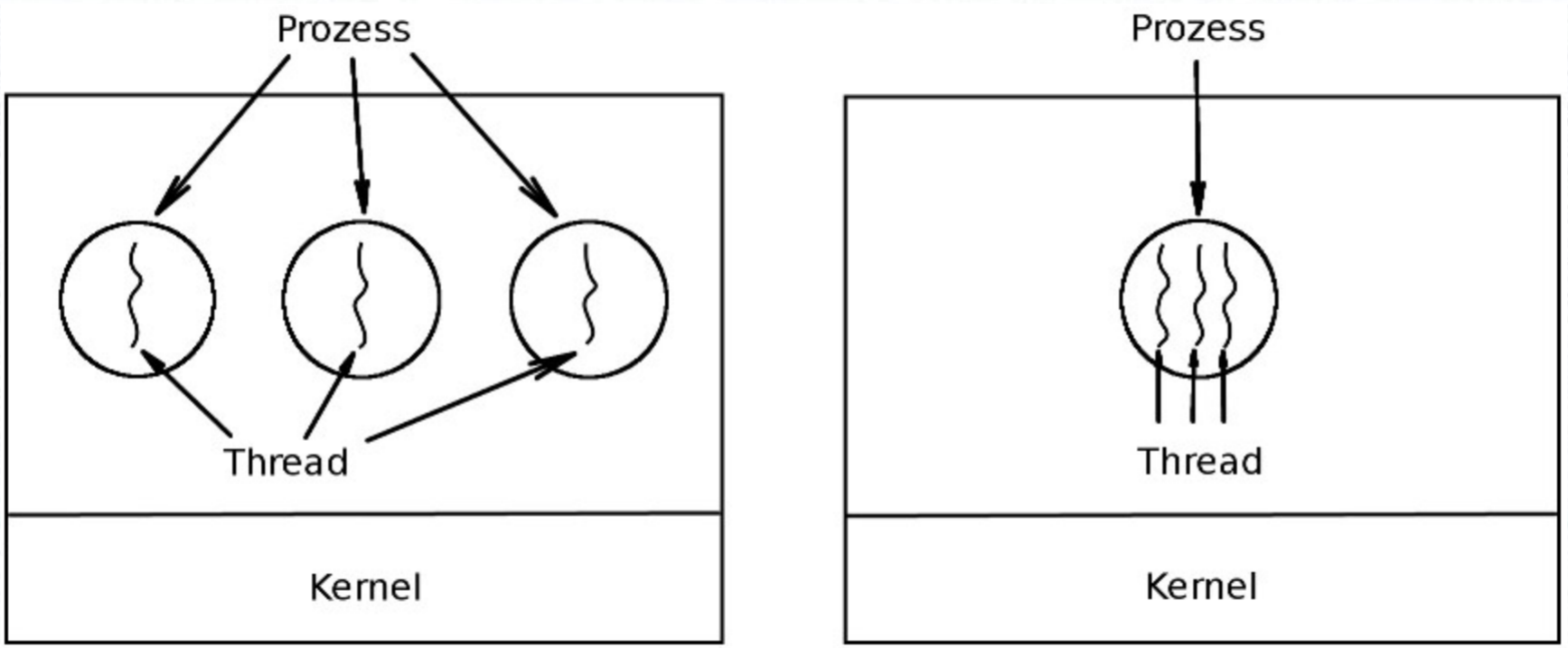
Wer hat schon mal was von Threads gehört?

Betriebssystem und Prozesse

- Ein Programm läuft als Prozess im Betriebssystem (BS) ab
- Das BS wählt nach gewissen Regeln (z.B. Priorität) den Prozess aus, der die CPU bekommt (Scheduling)
- Prozess bekommt einen Zeitslot vom BS

Threads

- Ein Prozess besteht aus mindestens einem Thread (main)



Threads für schnellere Programme

- Threads sind nebenläufige Handlungsstränge, die echt parallel laufen und somit kann ein Programm auf mehreren Prozessoren laufen

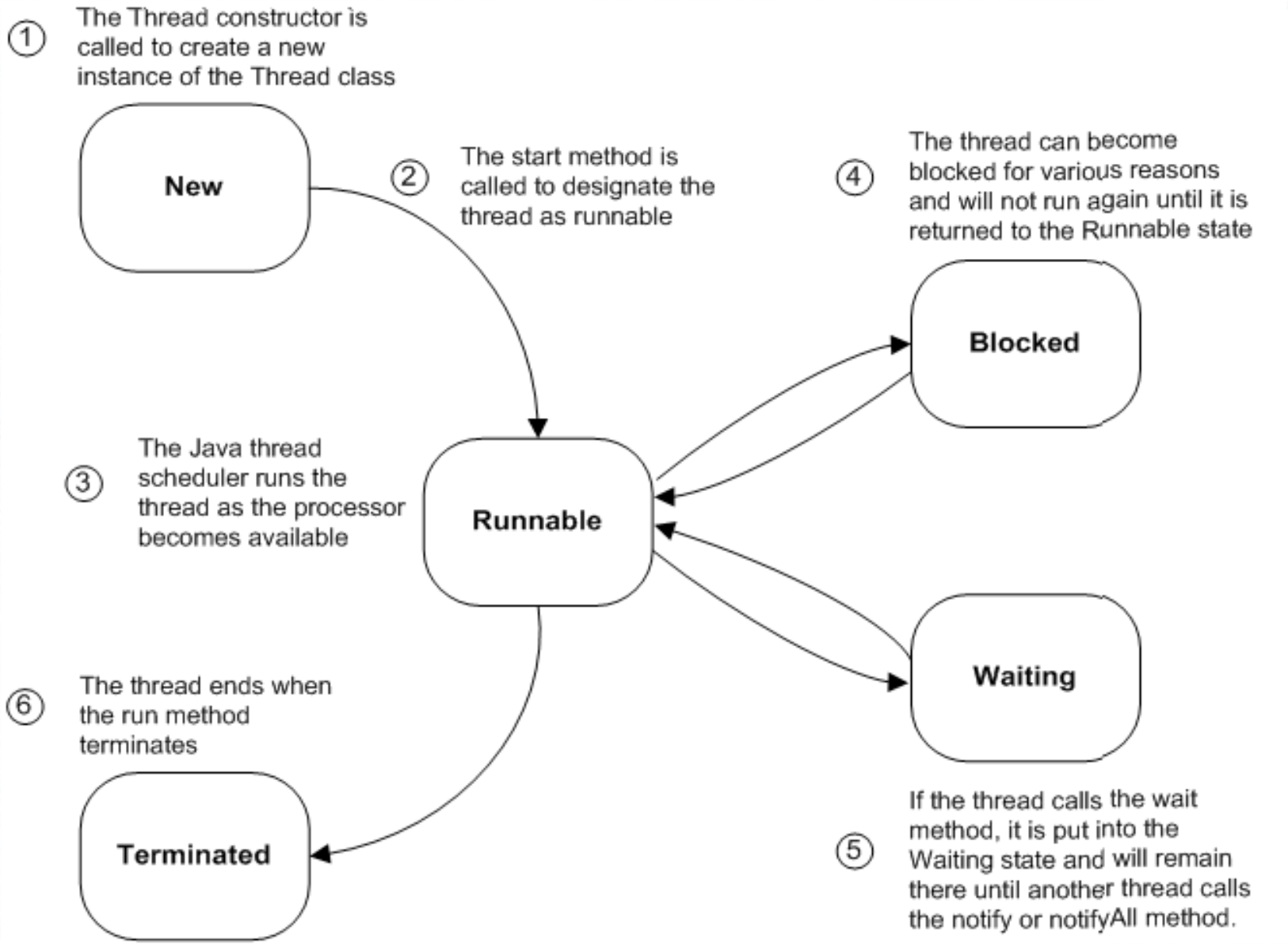
- Threads eines Prozesses teilen sich den Zeitslot auf der CPU

- Das BS entscheidet welcher Thread wie viel Zeit bekommt

Prioritäten

- Jeder Thread hat eine Priorität, die dem BS hilft festzulegen in welcher Reihenfolge Threads die CPU bekommen.
- Eine Priorität kann zwischen MIN_PRIORITY (1) und MAX_PRIORITY (10) sein, defaultmäßig bekommt jeder Thread NORM_PRIORITY (5).
- Threads mit höherer Priorität sind wichtiger und werden beim Scheduling höher priorisiert. Aber keine Garantie!

Thread States



Thread Methoden

```
//Startet den Thread und ruft die run()-Methode auf
public void start()

//setzt die Priorität des Threads
public final void setPriority(int priority)

//lebt der Thread schon/noch
public final boolean isAlive()

//der Thread, der die Methode aufruft blockiert solange, bis der
Thread bei dem join() aufgerufen wird, stirbt oder die Zeit um
ist
public final void join(int millisec)
```

Kleine Threadbeispiele

Eigene Threads können durch Erben von der Klasse Thread oder durch das Implementieren des Interfaces Runnable erstellt werden.

Warum gibt es zwei Möglichkeiten?

Kleine Threadbeispiele

Eigene Threads können durch Erben von der Klasse Thread oder durch das Implementieren des Interfaces Runnable erstellt werden.

Warum gibt es zwei Möglichkeiten?

- Für Klassen die schon erben das Interface Runnable
- Eine vorgefertigte Klasse Thread wird gebraucht, um nicht alle Methoden selber zu schreiben (start, join, …)

Beispiel mit der Thread-Klasse

```
public class MyThreadExample extends Thread{
    private String printText; //auszugebender Text

    public MyThreadExample(String text) {
        printText = text;
    }

    @Override
    public void run() {
        System.out.println(printText);
    }
}
```

Beispiel mit dem Runnable-Interface

```
public class MyRunnableExample implements Runnable{
    private String printText; //auszugebender Text

    public MyRunnableExample(String text) {
        printText = text;
    }

    @Override
    public void run() {
        System.out.println(printText);
    }
}
```

Thread starten

```
public static void main(String[] args) {
  //man ruft mit start () die run () - Methode auf
  MyThreadExample t1 = new MyThreadExample("Tip");
  thread1.start();

  //man muss das Runnable-Object in eine Threadhülle tun
  MyRunnableExample r1 = new MyRunnableExample("Top");
  Thread t2 = new Thread(r1);
  thread2.start();
}
```

Threads schlafen legen

und

aufwecken

Sleep und Interrupt

- Manchmal möchte man einen Thread für eine bestimmte Zeit schlafen legen, das macht man mit:

```
Try {
  Thread.sleep( 2000 );
} catch ( InterruptedException e ) { }
```

- Man kann den Schlaf von außen mit einem Interrupt unterbrechen

```
//sendet ein Interrupt, damit der Thread weiterlaufen kann
public void interrupt()
```

Aufgabe Timer

Baut mit Hilfe eines Threads einen Timer und verwendet dabei die sleep-Methode.

Dabei soll der Thread jede ms eine time Variable aktualisieren.

Aufgabe StoppUhr

Ladet euch die StoppUhr Aufgabe von GitHub herunter und erweitert die Aufgabe durch einen Thread, der jede Sekunde die Zeit aktualisiert.

Fallen im Umgang mit Threads

Fallen im Umgang mit Threads

Befehle sind nicht atomar, also in einer CPU-Anweisung aufzuführen

Beispiel: i++

Was passiert denn bei i++ ?

Fallen im Umgang mit Threads

Befehle sind nicht atomar, also in einer CPU-Anweisung aufzuführen

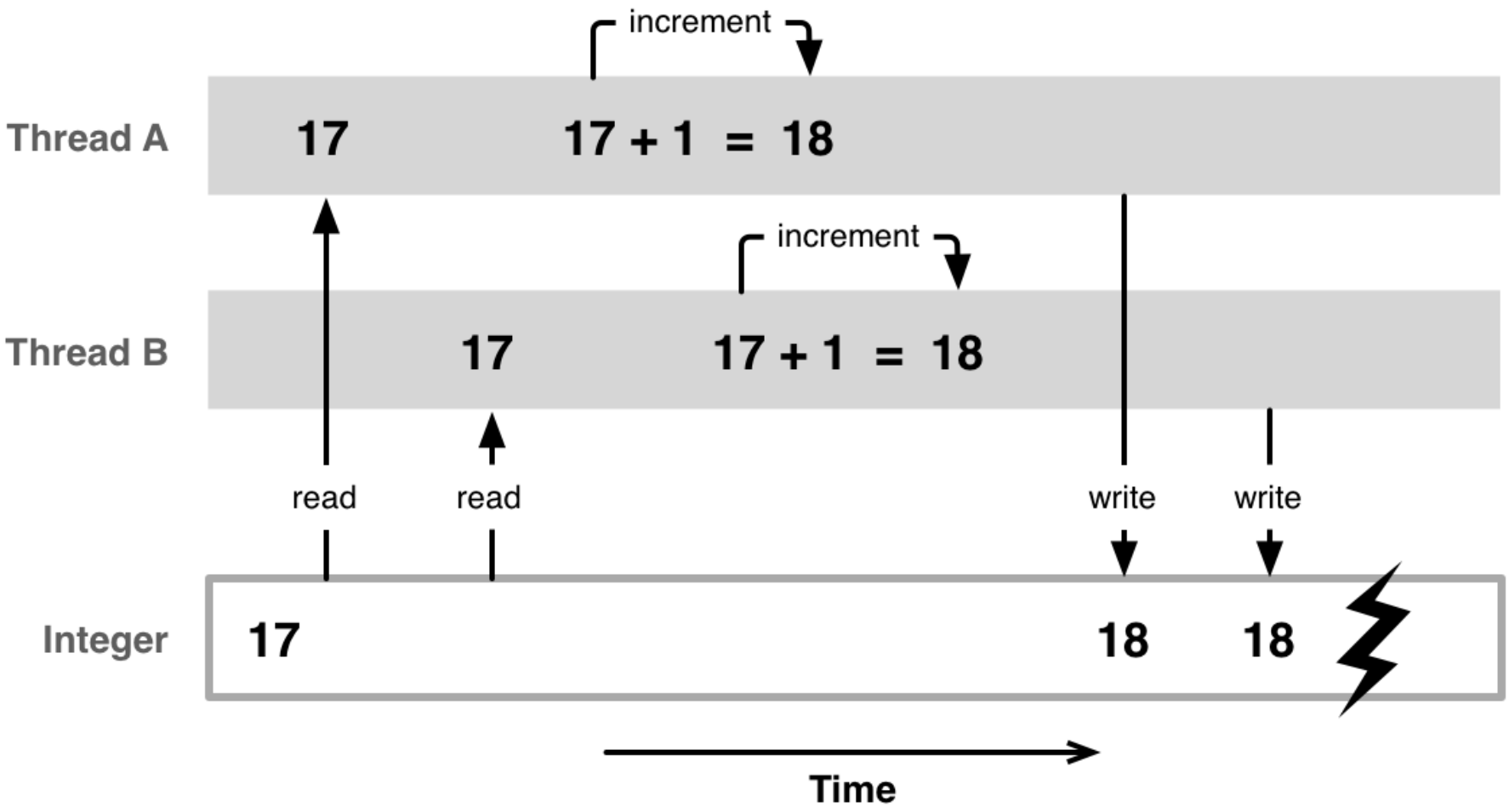
Beispiel: i++

1) Wert von i lesen

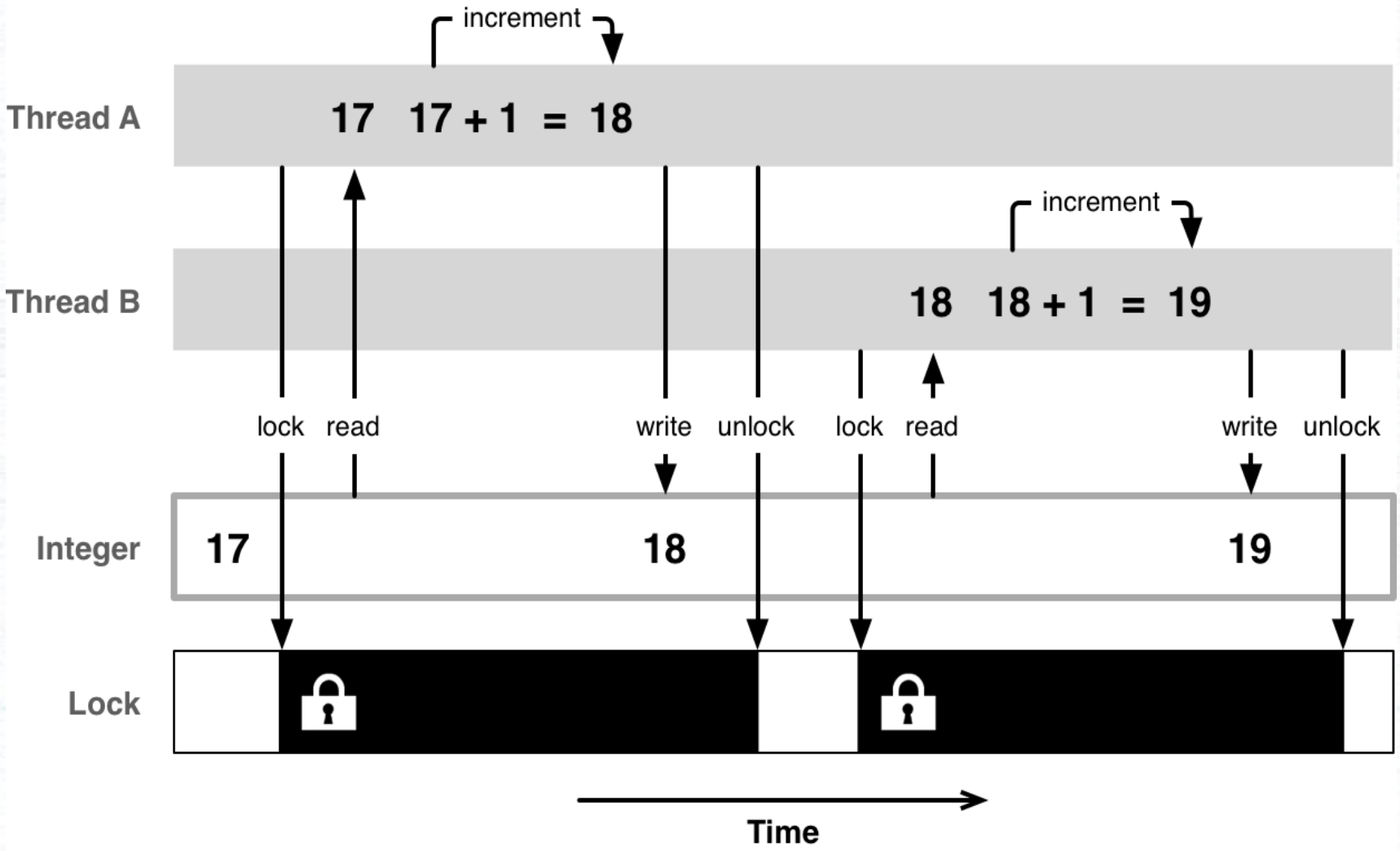
2) Wert + 1 rechnen

3) Den neuen Wert von i speichern

Fallen im Umgang mit Threads
Data Race



Fallen im Umgang mit Threads
Data Race - Lösung



Fallen im Umgang mit Threads
Data Race – Lösung

Lösung:

```
synchronized void foo() { i++; }


//synchronized kann in Kombination mit Methoden,
aber auch mit dem Ressourcen-Objekt verwendet
werden


synchronized(Object obj) {

   //ist das obj gelockt?       true
   System.out.println( Thread.holdsLock(obj) );

}
```

Consumer-Producer-Problem

- Ein Thread produziert eine Ressource
- Ein anderer konsumiert diese
- Wenn der Consumer alle Ressourcen verbraucht hat, legt er sich solange schlafen bis neue produziert wurden
- Wenn ein Producer nichts mehr produzieren kann, legt sich auch dieser schlafen, bis wieder Platz für neue Ressourcen ist

```
class Consumer implements Runnable {

   private final BlockingQueue<Integer> queue;

   public Consumer(BlockingQueue<Integer> queue) {
     this.queue = queue;
   }

   @Override
   public void run() {

      while (true) {

        try {

           System.out.println(queue.take());

        } catch (InterruptedException ex) {}
}}}
```

```
class Producer implements Runnable {
    private final BlockingQueue<Integer> queue;
    public Producer(BlockingQueue<Integer> queue) {
        this.queue = queue;
    }
    @Override
    public void run() {
        for (int i = 0; i < 400; i++) {
            try {
                System.out.println("Produced: " + i);
                queue.put(i);
            } catch (InterruptedException ex) {}
}}}
```

```
public class ProducerConsumerPattern {

  public static void main(String args[]){

   //Creating shared object
   BlockingQueue<Integer> queue = new
                                  LinkedBlockingQueue<>();

   //Creating Producer and Consumer Thread
   Thread prodThread = new Thread(new Producer(queue));
   Thread consThread = new Thread(new Consumer(queue));

   //Starting producer and Consumer thread
   prodThread.start();
   consThread.start();
  }
}
```

Wait und Notify

Wait

Wenn es keine Ressourcen zu konsumieren gibt, legt sich der Konsumer schlafen.

Notify

Hat der Producer etwas erstellt, benachrichtigt er einen/alle Konsumenten.

Wait und Notify

```
//Der Thread wartet/blockiert so lange bis ein anderer Thread ihn mit notify aufweckt

public void wait()


//weckt einen anderen Thread auf, der auf eine Ressource wartet

public void notify()


//weckt alle Threads auf die auf die Ressource mit wait() warten

public void notifyAll()
```

Vorsicht GUI und Thread

Fallen euch Probleme warum es kritisch sein kann, wenn mehrere Threads auf die GUI zugreifen?

Vorsicht GUI und Thread

Fallen euch Probleme warum es kritisch sein kann, wenn mehrere Threads auf die GUI zugreifen?

- Mehrere Thread wollen gleichzeitig eine Komponente verändern
- Der User interagiert mit der GUI, dadurch wird eine Komponente verändert und gleichzeitig versucht ein Thread diese Komponente zu ändern

Lösung: runOnUIThread(Runnable runnable)

Gibt es

Fragen ? ? ?