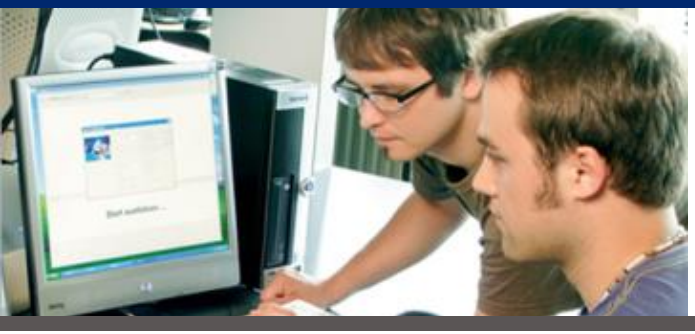


Threads in Java

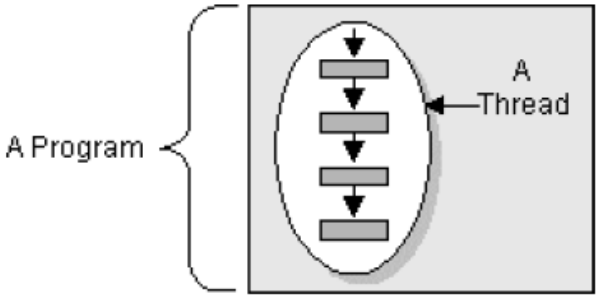
Betriebssysteme

Tobias Lauer

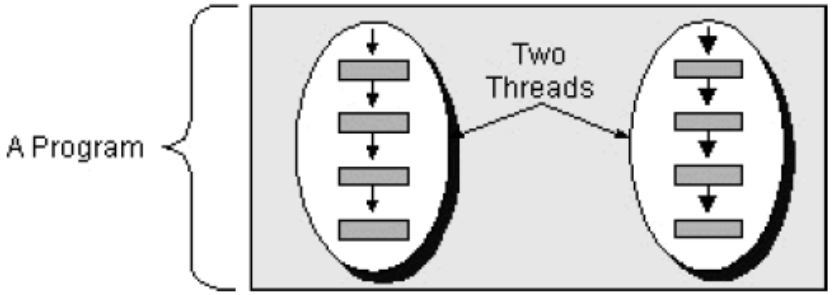
Threads

- Parallelisierung von Aufgaben
- Vorteile
  - Nutzung mehrerer Prozessoren/Cores für Teilaufgaben
    - Anwendung läuft schneller
  - Unterstützung von GUI-Applikationen
    - Anwendung reagiert schneller auf User-Input
  - Nebenläufigkeit von Operationen verschiedener Nutzer
    - Vermeidung von Wartezeiten für User bei Server-Anfragen

single-threaded



multi-threaded



Threads
In Java

- Java verwendet Threads zur Ausführung von Aufgaben
- Ein Thread ist ein Objekt, welches in der Lage ist, eine einzelne Aufgabe zu einem Zeitpunkt auszuführen
- Das JRE erstellt auch schon von sich aus mehrere Threads (ohne dass wir es mitbekommen)
  - Programmeinstieg in der main()-Methode („main-Thread“)
  - Garbage Collection Thread (räumt den Speicher auf)
  - GUI Thread, falls GUI vorhanden (Interaktion mit Komponenten)
- Eigene weitere Threads können erzeugt und gestartet werden
- Speziell wenn gemeinsame Objekte (gemeinsamer Speicher) verwendet werden, müssen Threads oft auch zusammenarbeiten

Threads
In Java

- Deklaration, zwei Möglichkeiten:
  - Erzeugen einer Klasse, die das **`Runnable`** Interface implementiert
  - Erweitern der Klasse **`Thread`**
- Erzeugung
  - Erzeugen eines **`Runnable`** Objektes und Übergabe dieses Objektes an den Konstruktor der Klasse **`Thread`**
  - Erzeugen einer Instanz der eigenen Subklasse von **`Thread`**
- Start
  - Die **`start`**-Methode des eigenen **`Thread`** Objektes aufrufen
- Stop
  - Verwendung eines Flags zum beenden der **`run`**-Methode
  - Ein einmal beendeter Thread kann nicht wieder gestartet werden

Threads
In Java

```
public class MyThread extends Thread
{
  public void run()
  {
    for(int i = 0; i < 10000; i++)
      System.out.println("A:" + i);
  }

  public static void main(String[] args)
  {
    Thread t = new MyThread();
    t.start();
    for(int j = 0; j < 10000; j++)
      System.out.println("B:" + j);
  }
}
```

Erweiterung von Thread

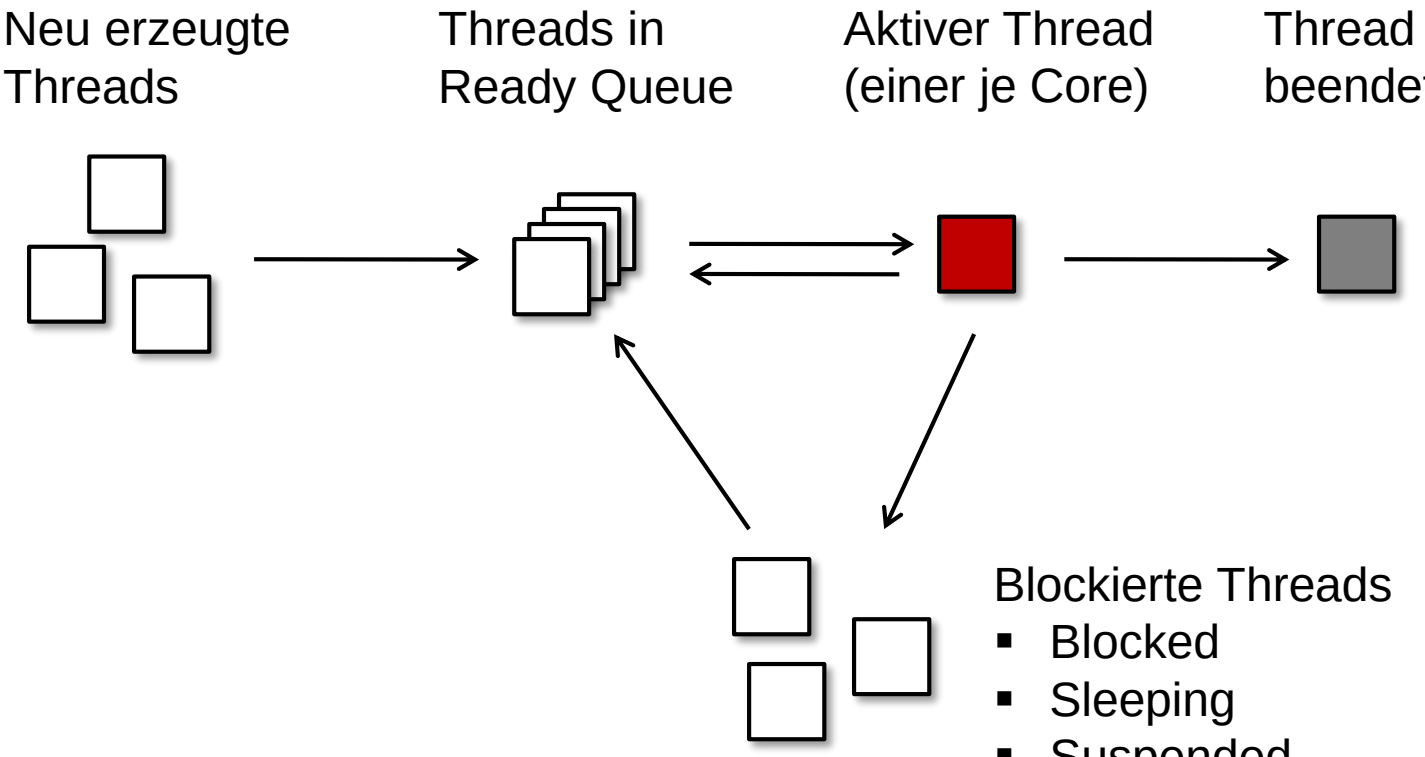
Implementierung des *Runnable* Interface: Die Methode *run()* enthält den Code, der beim Start des Threads abgearbeitet werden soll.

Erzeugung des Threads

Starten des Threads

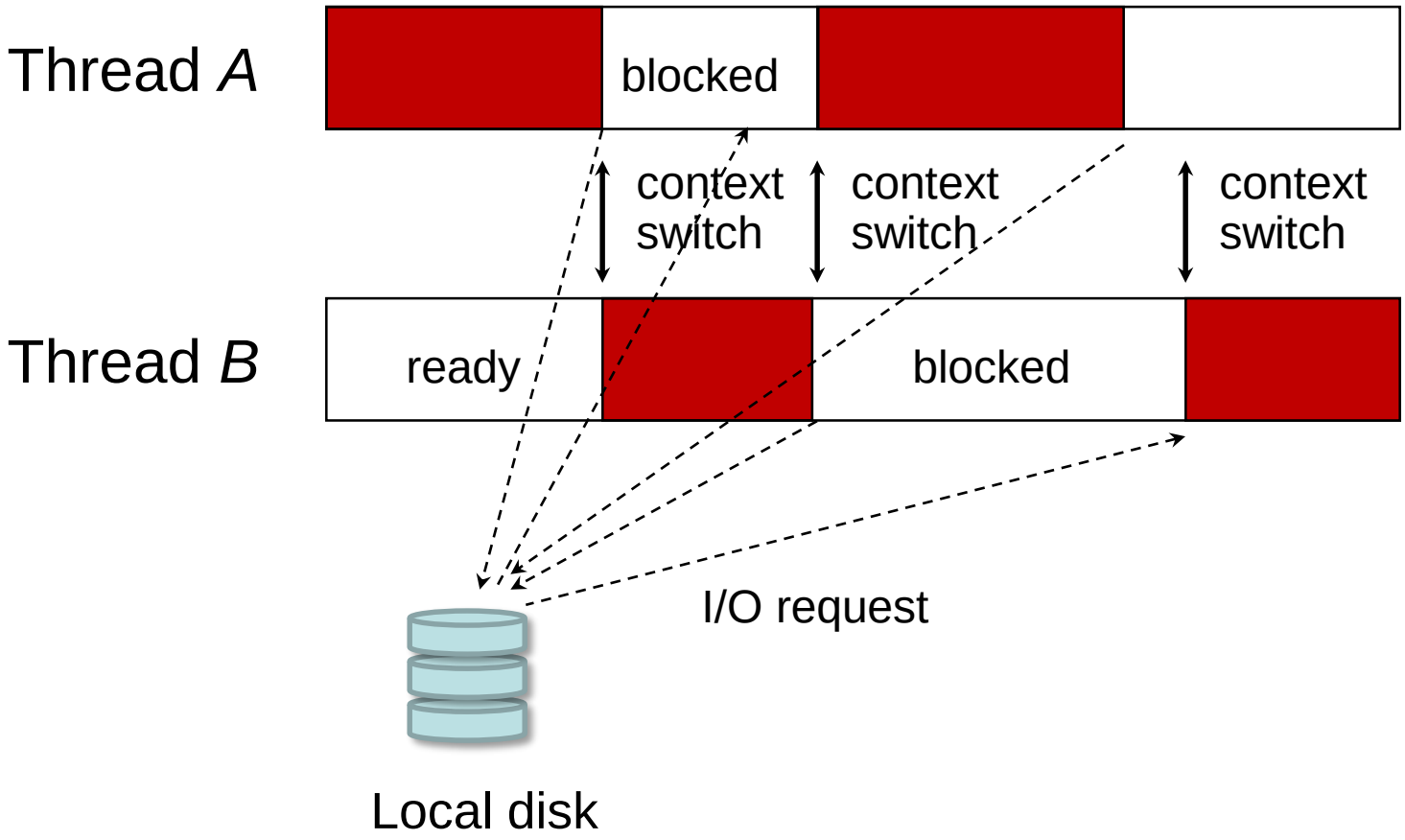
Ab hier laufen der *main*-Thread und Thread *t* gemeinsam

Thread Zustände
Lebenszyklus



Thread Zustände

Scheduling und Verdrängung



Thread- Zustände: Ready vs. Blocked

- Ein Thread kann in den Zustand blocked wechseln, wenn:
  - Thread wartet auf I/O
  - Seine `suspend()` Methode wurde aufgerufen
  - Ruft eine `join()` Methode eines anderen Thread auf
  - Schläft durch die Verwendung von `Thread.sleep()`
  - Ist “nett” durch die Verwendung von `Thread.yield()`
  - Ruft eine `wait()` Methode auf einem Objekt auf
  - Versucht einen kritischen Bereich aufzurufen, dessen Sperre (Lock) bereits vergeben ist

- Er wechselt wieder in den Zustand ready, wenn
  - I/O ist beendet
  - Seine `resume()` Methode aufgerufen wird
  - Der andere Thread, für den `join()` aufgerufen wurde, beendet ist oder der Timeout für die Zusammenführung erreicht wurde
  - Wartezeit (`sleep`-Dauer) ist abgelaufen
  - Der Thread wurde für die Ausführung neu geplant (rescheduled)
  - Eine `notify()` Methode wurde auf dem Objekt, auf das er wartet, aufgerufen
  - Die Sperre (Lock) wurde aufgehoben

Thread Zustände
Prioritäten

- Java Threads werden von 1 (niedrig) bis 10 priorisiert
- Die Standardpriorität (**`Thread.NORM_PRIORITY`**) ist 5
- Die Priorität eines Threads kann mit **`getPriority`** ermittelt und mit **`setPriority`** manipuliert werden.
- Jeder Thread wird ausgeführt, wenn kein höher priorisierter Thread im Ready-Zustand ist
- Keine Aussage über Threads mit gleicher Priorität
- Java Threads werden präemptiv gescheduled
  - Verdrängung eines Threads durch den Scheduler
  - Vermeidung von Livelocks (s. spätere Folie)