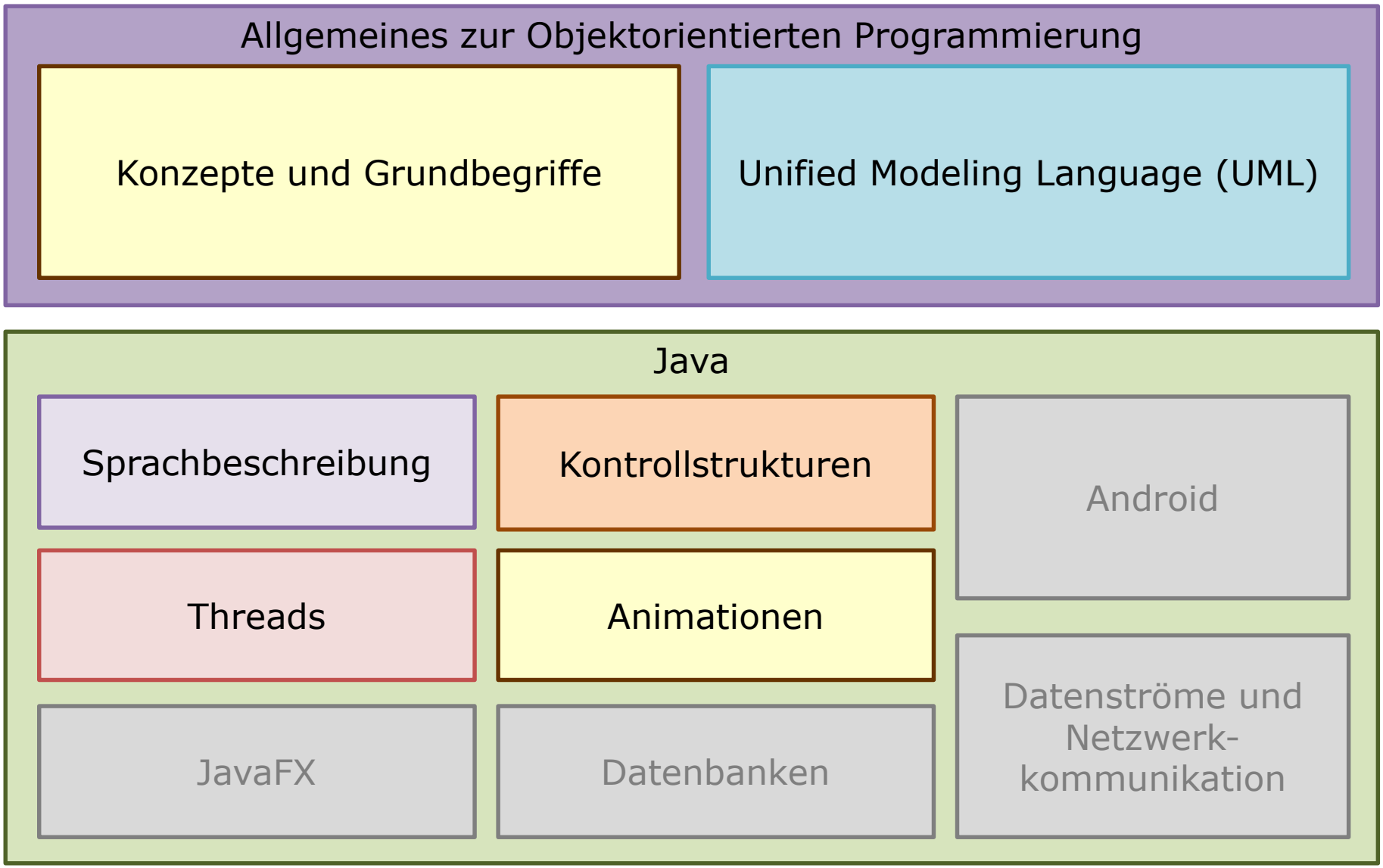
BERUFSAKADEMIE SACHSEN
STAATLICHE STUDIENAKADEMIE
DRESDEN
UNIVERSITY OF COOPERATIVE EDUCATION

Objektorientierte Programmierung

Threads und Animationen

Prof. Dr.-Ing. Tenshi Hara
tenshi.hara@ba-sachsen.de

Aufbau der Lehrveranstaltung



Entwicklung des Multi-Threading

- bei den ersten Rechnern: ausschließliches Nutzungsrecht eines Nutzers mit einem Programm (zeitaufwändige Ergebnisanalyse und Fehlersuche)
- ab 1955: Stapelbetrieb (Jobs; Programm und Ergebnisse auf Lochstreifen/Lochkarten gestanzt)
- Einführung von Vorrechnern: vor Verarbeitung Lochbänder/Lochkarten auf Magnetbänder übertragen
- ab 1965: Mehrprogrammbetrieb (Multi Programming), dadurch bessere Ausnutzung des Prozessors
  - Einsatz der Rechner für Echtzeitbetrieb
  - intelligente periphere Geräte ermöglichten Dialog- und Mehrnutzerbetrieb
- in 1980er-Jahren: Entstehung des Thread-Konzepts

Grade der Nebenläufigkeit

- Multi Tasking: Parallelität durch Zusammenspiel von Hardware und Betriebssystem
  - Hardware-Parallelität wird durch Einsatz mehrerer Prozessoren realisiert (Betriebssystem braucht geeignete Prozesskontrolle)
  - Pseudo-Parallelität bei Rechnern mit nur einem Prozessor wird realisiert, indem mit hoher Frequenz zwischen Programmen umgeschaltet wird (Betriebssystem braucht geeigneten Scheduler)
- kooperatives Multi Tasking: Programme verhalten sich kooperativ
  - Programme geben freiwillig Steuerung an das Betriebssystem zurück
  - Betriebssystem wählt aus Warteschlange nächste Aktivierung
- Single Tasking: Betriebssystem führt jederzeit nur eine Anwendung aus

Multi-User und Multi-Tasking

- wenn mehrere Nutzer einrichtbar: Mehrbenutzersystem
- Multi-Tasking-Betriebssysteme sind Betriebssysteme mit „echter" Parallelität (Preemptive Multi Tasking)
- Multi-User-Betriebssysteme sind Mehrbenutzersysteme, bei denen mehrere Nutzer *gleichzeitig* arbeiten können
  - Voraussetzung für Multi-User-Fähigkeit ist die Multi-Tasking-Fähigkeit
  - Unix ist von Anfang an als Multi-User-Betriebssystem konzipiert

Prozesse und Threads

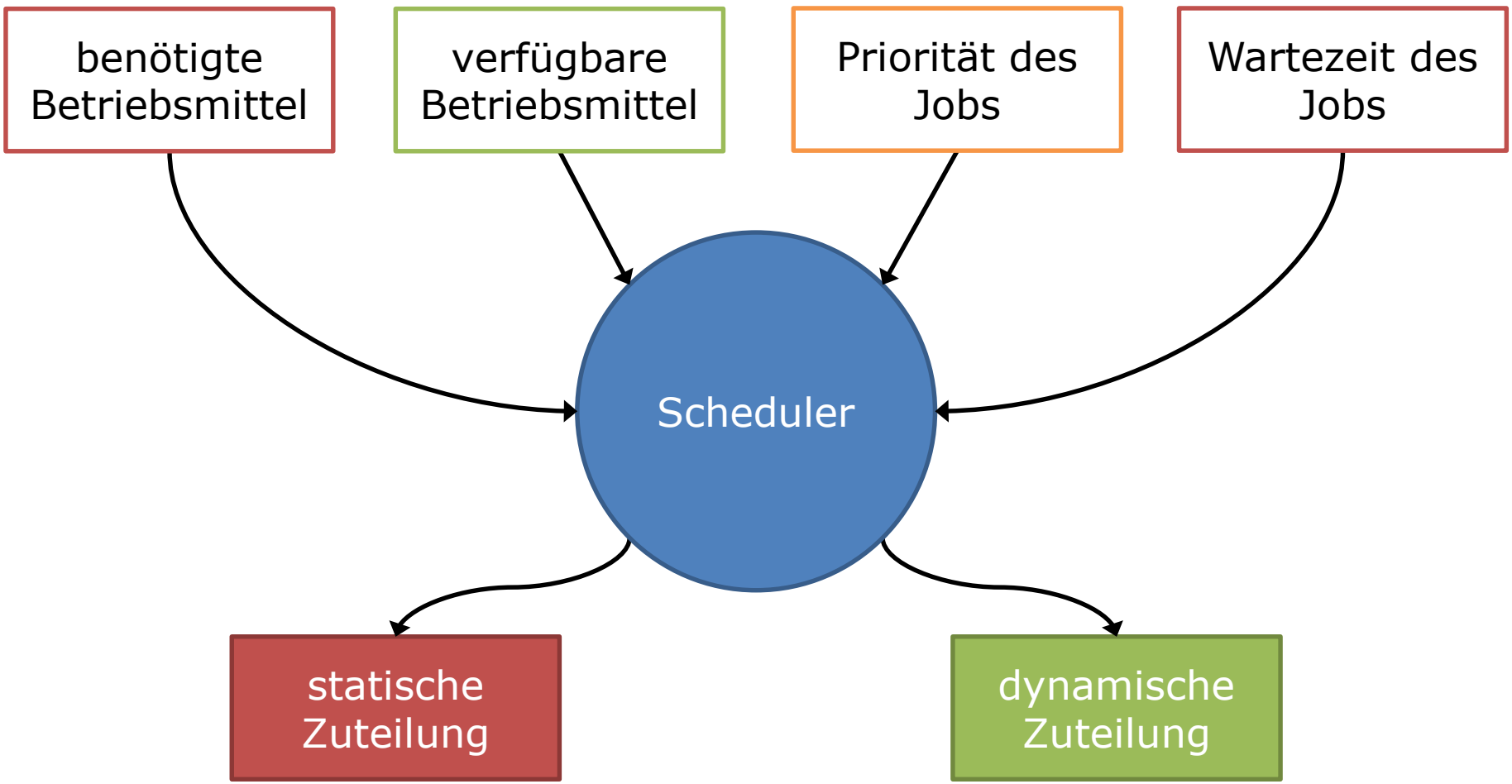
- die Begriffe Task und Prozess sind äquivalent
- die Begriffe Thread und Task sind nicht äquivalent
- Programme sind eingebettet in einem Task mit eigenem Prozesskontrollblock (Prozessvariablen, Tabelle offener Dateien mit Bearbeitungszuständen, …) und Prozessumgebung (Adressraum)
- Tasks kommunizieren untereinander über Inter Process Communication (Pipes, Semaphoren, Shared-Memory, Signale, Queues)
- Threads sind Programmstücke ohne eigenen Kontrollblock und ohne eigenen Adressraum (daher manchmal auch „Leichtgewichtsprozess“)
  - Umschalten zwischen Threads ist sehr einfach (es müssen nur die Register gerettet werden)
  - alle Threads eines Prozesses arbeiten auf den selben Daten

Threads

- Betriebssysteme ohne Thread-Konzept realisieren Threads über Prozesse
- ein Prozess enthält mindestens einen Thread
  (Threads sind immer Teil eines Prozesses)
- durch Threads lassen sich verschiedene Teilaufgaben flexibel lösen
  - reagieren auf Ereignisse
  - Zeitgeber (bspw. für Uhr)
  - Berechnungen im Hintergrund ausführen
  - …
- für die Synchronisation der Threads ist der Programmierer verantwortlich
  - Verhindern von wechselseitigem Ausschluss (Deadlock, MutEx)
  - Zugriff auf Ressourcen

Scheduler

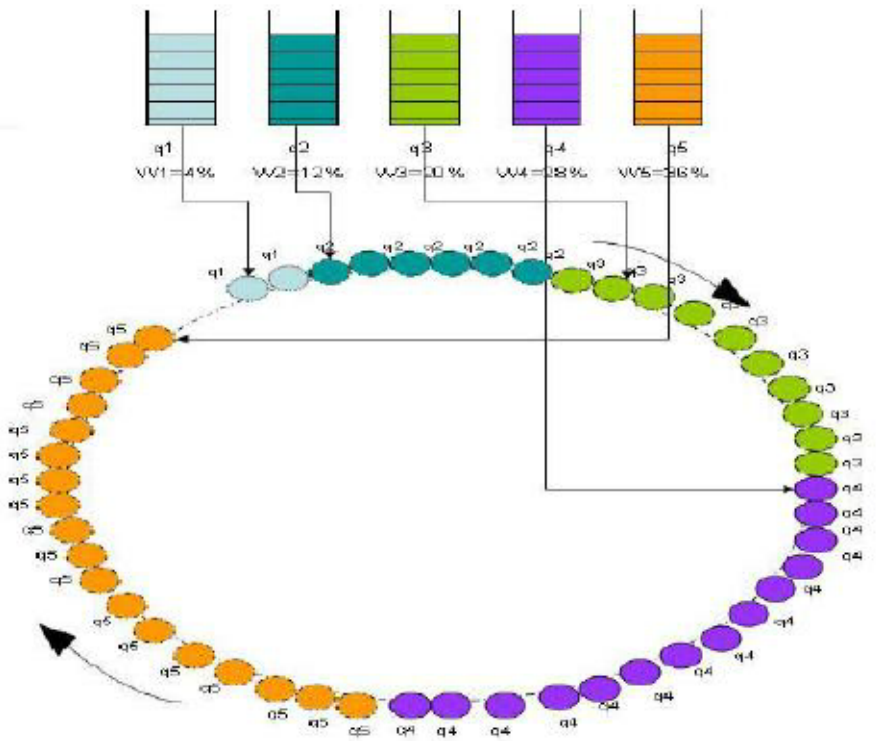
Teil des Betriebssystems und zuständig für die Betriebsmittelzuteilung



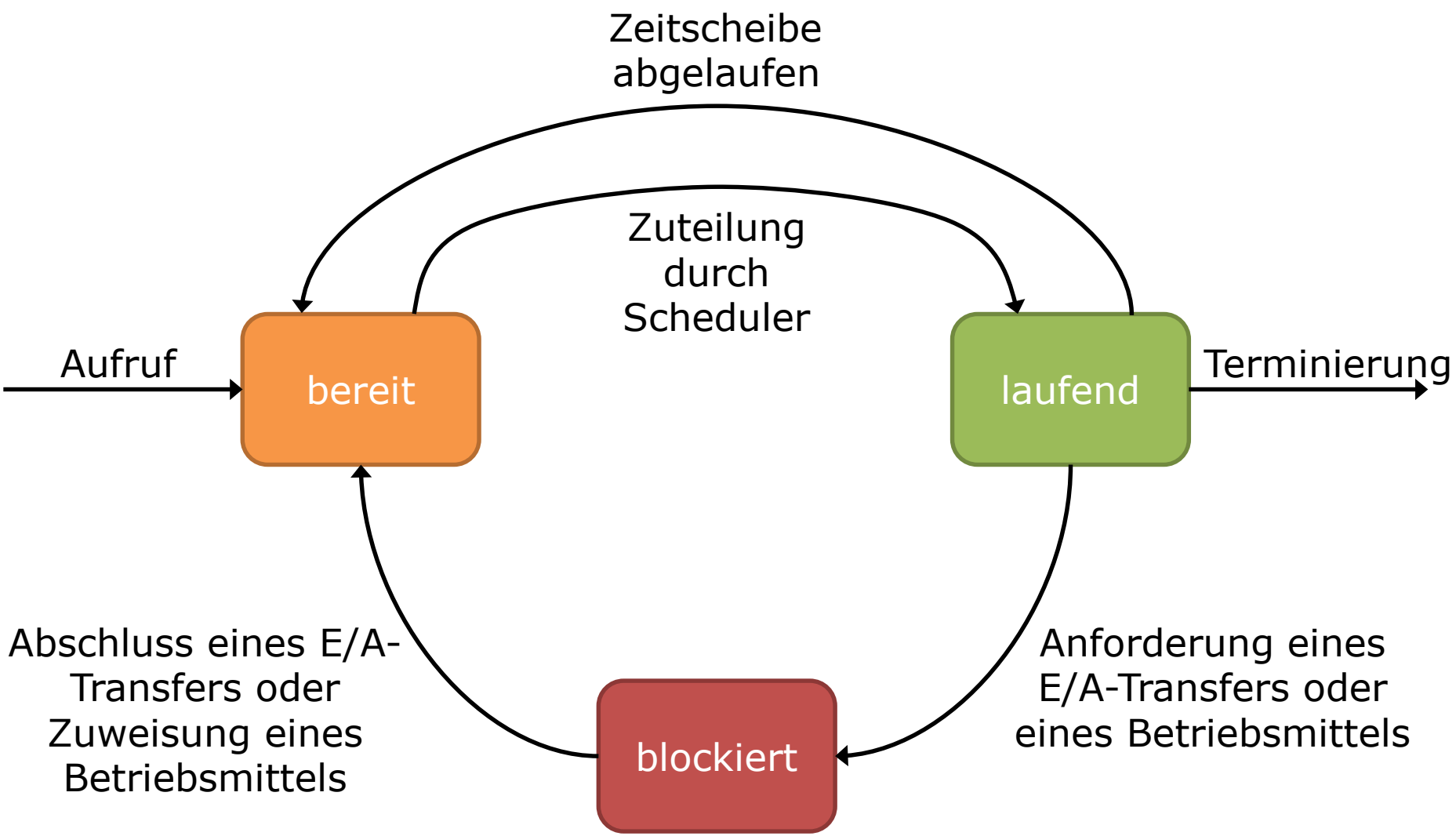
Robin Round

Zuteilungsstrategie

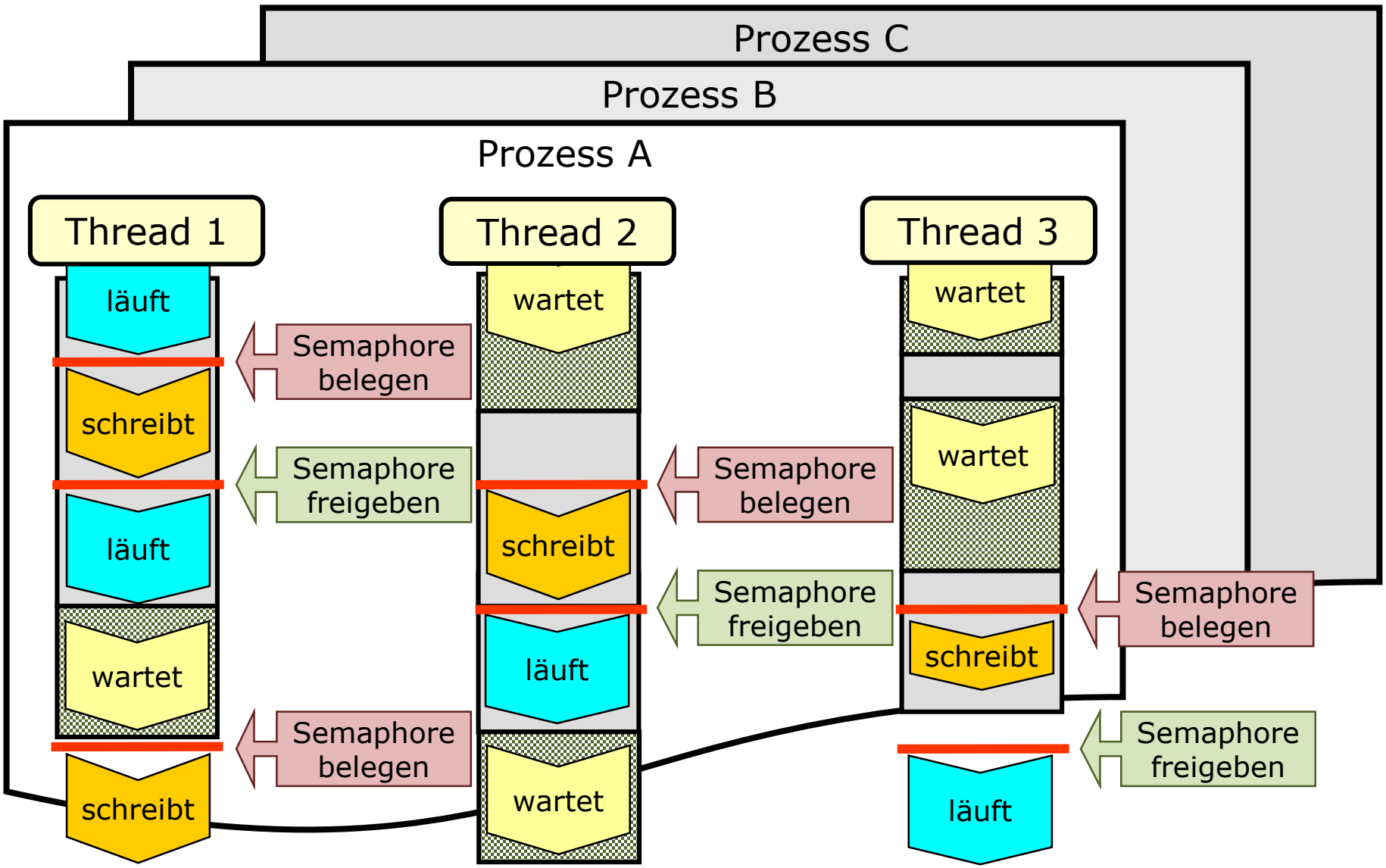
- Betriebsmittelzuteilung ist äquidistant getaktet
- Prozessen werden in vorgegebener Reihenfolge (und dem Takt folgend) Betriebsmittel zugweisen/entzogen
- oft gekoppelt mit Prioritäten
  - High: Prozesse können häufiger eingetaktet werden
  - …
  - Low: Prozesse können seltener eingetaktet werden



Prozesszustände

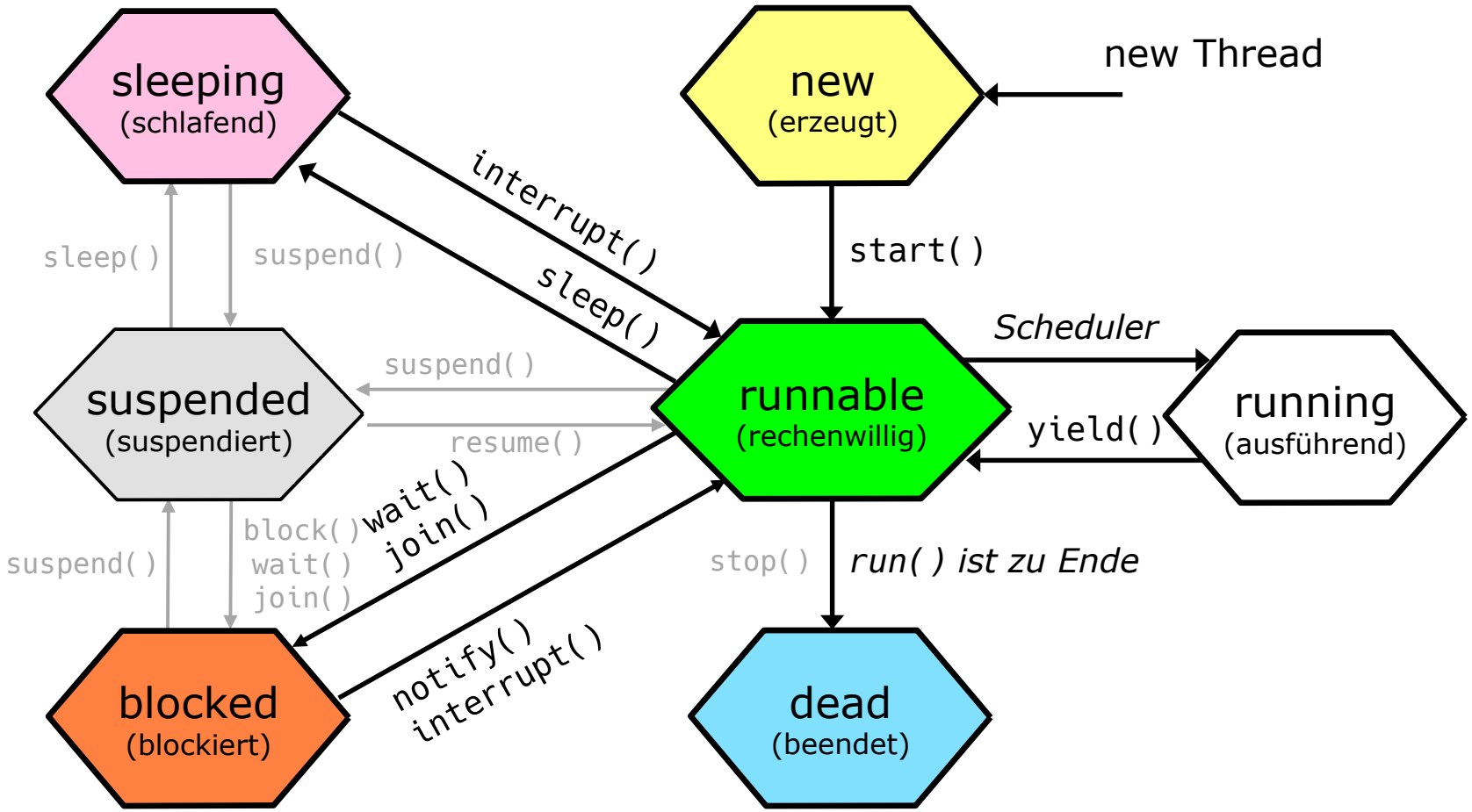


SERIALISIERUNG

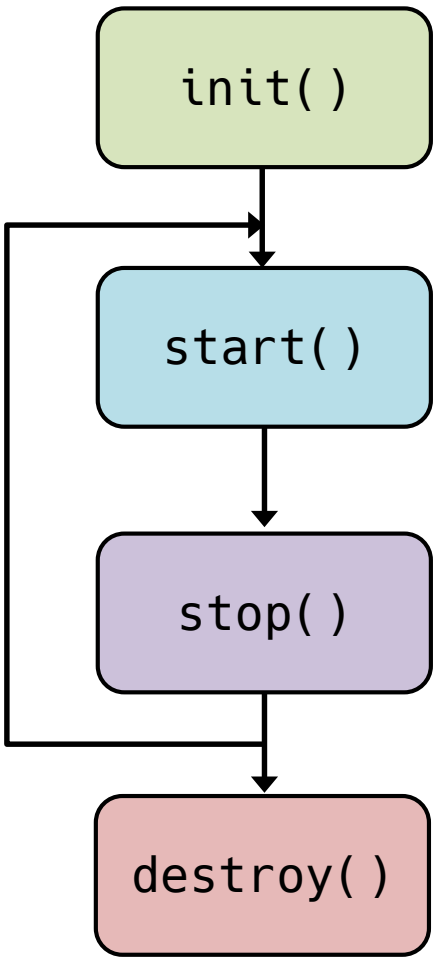


Threads

JAVA-THREADS



Applet-Lebenszyklus des Applet



wird vom Browser aufgerufen wenn Applet zum ersten Mal geladen wird
(ähnlich Konstruktor bei Applikationen)

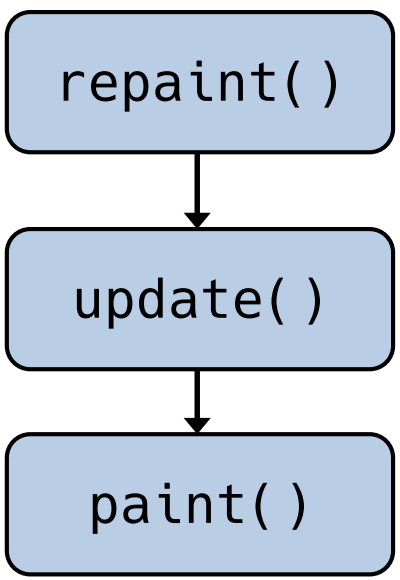
wird automatisch ausgeführt und damit das Applet gestartet; wird auch aufgerufen, wenn Applet wieder sichtbar wird

hält Applet an, wenn HTML-Seite verlassen wird oder anderes Programm vordergründig wird
(Applet wird vorübergehend unsichtbar)

Aufruf, wenn neue Seite geladen wird oder wenn Browser geschlossen wird

Aufrufreihenfolge bei repaint()

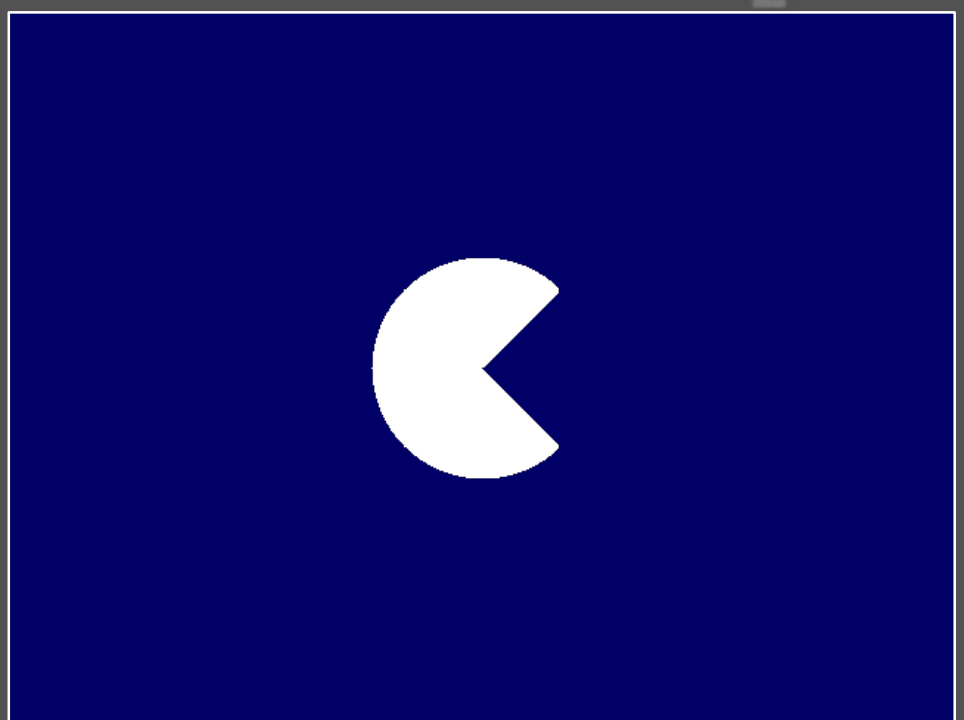
- `paint()` ist bei allen Flächen, auf denen gezeichnet werden kann, vorhanden (Frame, Panel, Canvas, Applet, …).
- wird automatisch aufgerufen, wenn sich irgend etwas an der Zeichenfläche ändert (Größenänderung, Minimieren, Wiederherstellen, Überdecken, Vorholen…)
- `paint()` wird indirekt über `repaint()` aufgerufen (vom Programmierer) (dazwischen liegt der Aufruf von `update()`)



Erzwingt Neuzeichnen (oft bei Animationen)

löscht u.a. den Bildschirmhintergrund (was manchmal nicht gewünscht wird); wenn etwas anderes getan werden soll, muss die Methode überschrieben werden

zeichnet, was in der Methode implementiert ist



PACMAN-ANIMATION

Grundgerüst:	Fenster
Stufe 1:	Attribute einfügen
Stufe 2:	Methode `init()` des Applet
Stufe 3:	Methoden `start()` und `stop()` des Applet
Stufe 4:	Methode `run()` des Thread
Stufe 5:	Methoden `paint()` und `update()` des Applet, um Flackern zu verhindern

Grundgerüst der Pacman-Animation

```
import java.awt.*;
import java.applet.*;

public class Pacman extends Applet 4a {

    1                   // Attribute

  public void init() { } 2
  public void start() { }   3a
  public void stop() { }    3b

    4b                  // Methode „run“ des Thread


  public void paint(Graphics g) { 5a }

    5b                  // Methode „update“

}
```

Stufe 1: Attribute einfügen

```
// Attribute
boolean runFlag;                        // steuert Thread

int x, y, Hoehe, Breite;                // des Applet-Fensters

Image Bild;                             // Bildspeicher
Graphics g;                             // Grafik-Kontext

Color NachtFarbe = new Color(0, 0, 102);
Color MondFarbe  = new Color(216, 202, 207);

Thread t = null;                        // Thread-Deklaration
```

Stufe 2: Methode init()

```
public void init() {
   Dimension d = getSize();                  // aus HTML-Datei

   Breite = d.width;                         // Breite und Höhe
   Hoehe = d.height;                         // des Applet-Fensters
   Bild = createImage(Breite, Hoehe);
   g = Bild.getGraphics();

   x = Breite/2;                             // Mitte des
   y = Hoehe/2;                              // Applet-Fensters
}
```

Stufe 3: Methoden start() und stop()

```
public void start() {
   if (t == null) {
      t = new Thread(this);   // Thread wird gestartet;
      t.start();              // Methode run() wird ausgeführt
   }
}

public void stop() {
   if (t != null) {
      runFlag = false;
      t = null;
   }
}
```

Stufe 4: Methode run() des Thread

```
/* 4a */ implements Runnable

// …

// 4b
public void run () {                    // run() des Thread
   runFlag = true;                      // zur Threadsteuerung
   while (runFlag) {
      g.setColor(NachtFarbe);
      g.fillRect(0, 0, Breite, Hoehe);
      g.setColor(MondFarbe);
      g.fillArc(x, y-25, 150, 150, 270, 180);
      repaint();                        // Bild neu zeichnen
      x += 1;                           // ein Pixel pro Schritt
      if (x > (Breite + 50)) x = -50;
      try { t.sleep(100); }             // Versuche, 100 ms zu schlafen
      catch (InterruptedException e) {} // Fehler abfangen
   }
}
```

Stufe 5: Methoden „paint“ und „update“

```
public void paint(Graphics g) { // 5a
   if (Bild != null)
      g.drawImage(Bild, 0, 0, null);
}

public void update(Graphics g) { //5b
   paint(g);
   // verhindert, dass der Bildhintergrund gelöscht wird
}
```

LITERATURANGABEN

Daniel Basler: Anwendungsentwicklung mit Java, Computer & Literatur Verlag GmbH, ISBN 3-932311-68-x

Rainer Oechsle: Parallele Programmierung mit Java Threads, Fachbuchverlag Leipzig im Carl Hanser Verlag, ISBN 3-446-21780-0

Java 1.1 Fortgeschrittene Programmierung, HERDT-Verlag für Bildungsmedien GmbH, Nackenheim