Konzepte von Betriebssystem-Komponenten

Semaphore

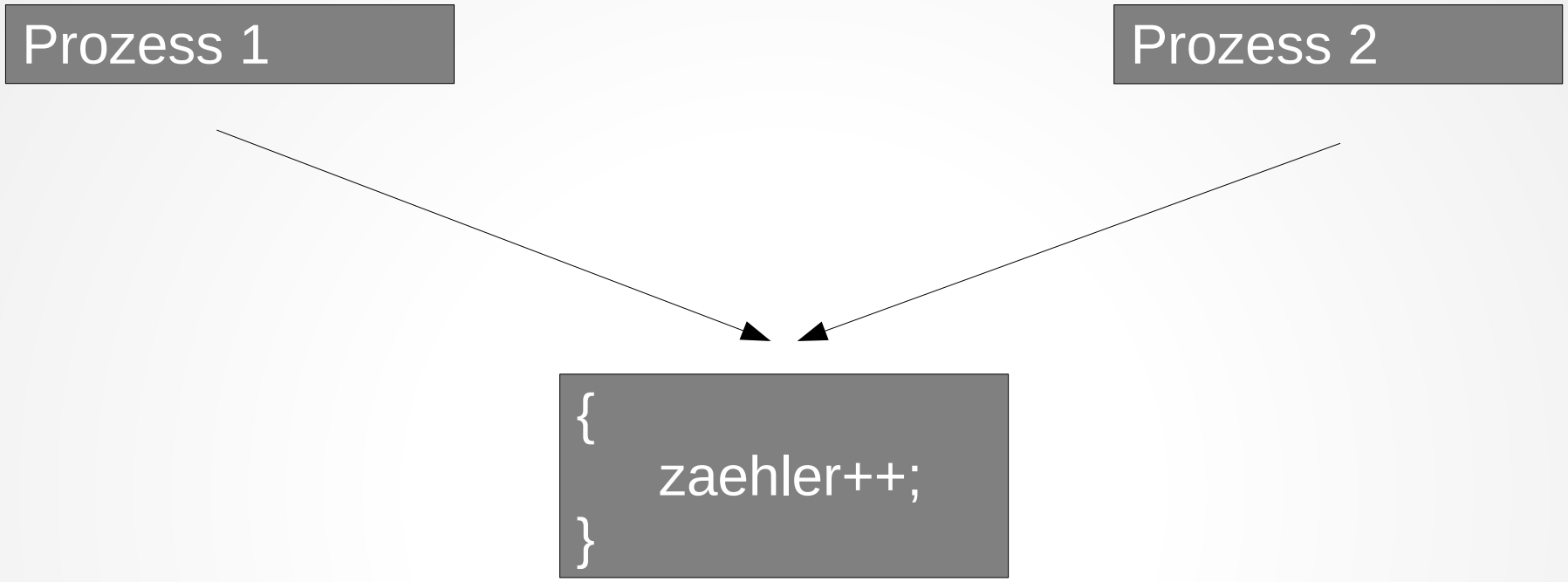
04. Juli 2013

Michael Gunselmann

Gliederung

- Problemfeld Parallelität
- Synchronisierung unabhängiger Prozesse
- Synchronisierungsmöglichkeit Semaphor
- Verklemmung und deren Vermeidung

Problemfeld Parallelität



Welchen Wert hat „zaehler“ nach n-maligem inkrementieren?

Problemfeld Parallelität

- Zwischen Lesen und Reagieren auf Variablenwert kann sich dieser ändern
- Kooperation zwischen Prozessen notwendig
  - bestimmte Abschnitte vor gleichzeitigem Betreten schützen

    → kritischer Abschnitt

Synchronisierung von Prozessen

- Vorgaben für folgende Betrachtungen:
  - Zwei unabhängige Prozesse, keine Aussage über zeitliche Verschränkung möglich
  - Kritischen Abschnitt vor gleichzeitigem Betreten schützen
  - Atomarer Schreibzugriff auf Variablen

Synchronisierung von Prozessen

```
integer turn = 1;
parbeginn
  process 1: begin
    L1: if turn = 2 then goto L1;
        critical section 1;
        turn = 2;
        // further loop statements
        goto L1;
  end;
```

```
process 2: begin
    L2: if turn = 1 then goto L2;
        critical section 2;
        turn = 1;
        // further loop statements
        goto L2;
 end;
```

Synchronisierung von Prozessen

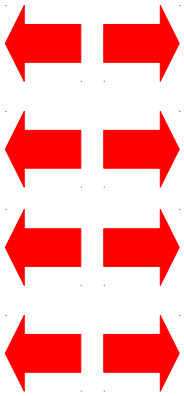
- Sehr restriktiv, kritischer Abschnitt wird nur abwechselnd betreten

  → keine zeitliche Unabhängigkeit

  - Prozess 2 hält, sobald Prozess 1 hält
- Verbesserung: Je eine Sperrvariable pro kritischem Abschnitt

Synchronisierung von Prozessen

```
integer c1 = 1, c2 = 1;
parbeginn
  process 1: begin                          process 2: begin
    L1: if c2 = 0 then goto L1;                 L2: if c1 = 0 then goto L2;
        c1 = 0;                                     c2 = 0;
        critical section 1;                         critical section 2;
        c1 = 1;                                     c2 = 1;
        // further loop statements                  // further loop statements
        goto L1;                                    goto L2;
  end;                                      end;
```



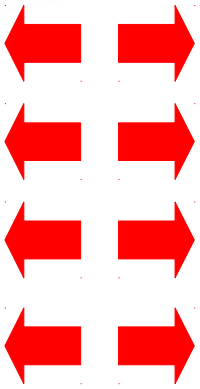
Synchronisierung von Prozessen

- Prüfen und reservieren läuft nicht atomar

  → zwei Prozesse im kritischen Abschnitt möglich
- Verbesserung: Erst eigenen kritischen Abschnitt sperren, dann prüfen

Synchronisierung von Prozessen

```
integer c1 = 1, c2 = 1;
parbeginn
  process 1: begin                          process 2: begin
    L1: c1 = 0;                                 L2: c2 = 0;
        if c2 = 0 then c1 = 1;                      if c1 = 0 then c2 = 1;
            goto L1;                                    goto L2;
        critical section 1;                         critical section 2;
        c1 = 1;                                     c2 = 1;
        // further loop statements                  // further loop statements
        goto L1;                                    goto L2;
  end;                                      end;
```

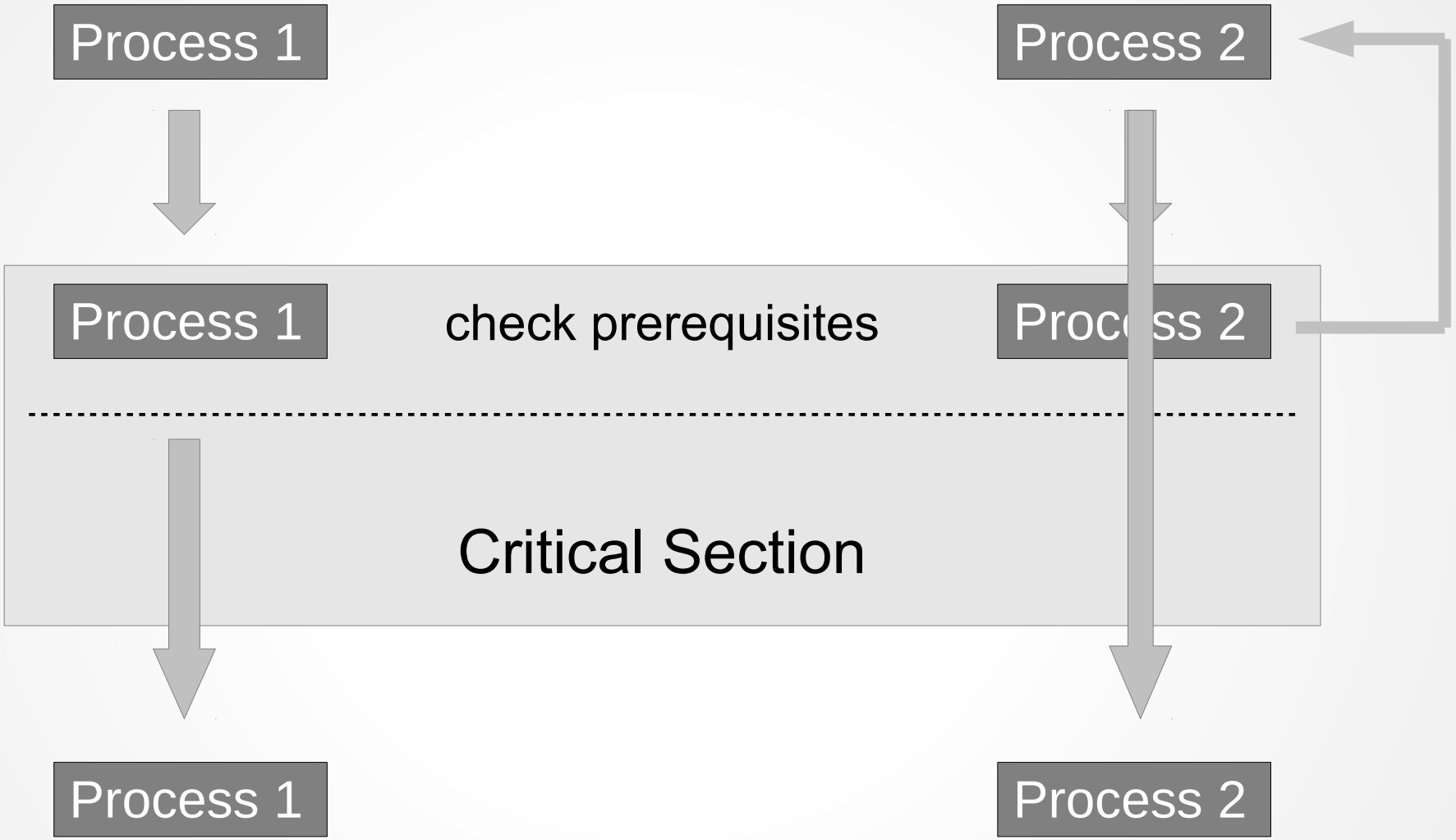


Synchronisierung von Prozessen

- Beide Prozesse könnten exakt gleich schnell laufen

  → Live-Lock
- alle Beispiele bieten keine Lösung
- Lösung liegt im Kombinieren der Beispiele 1 und 3
  - Jeweils ein Prozess erhält Vorrang
  - Variable gibt präferierten Prozess an

Synchronisierung von Prozessen



Synchronisierung von Prozessen

```
process 1: begin
    A1: c1 = 0;
    L1: if c2 = 0 then
                if turn = 1 then goto L1;
                c1 = 1;
           B1: if turn = 2 then goto B1;
                goto A1;
        critical section 1;
        c1 = 1;
        // further loop statements
        goto L1;
  end;
```

Synchronisierung von Prozessen

- Lösung ist auf beliebig viele Prozesse erweiterbar
  - c1, c2 → c[n]
  - turn gibt weiterhin privilegierten Prozess an

```
B1:  if turn = 2 then goto B1;
```

  - teures, aktives Warten
  - gewünscht: wartende Prozesse schlafen und werden geweckt

Semaphore

- nicht-negativer Integer
- Spezialfall: binärer Semaphor
  - Nimmt nur Werte 0 und 1 an
- Operationen zum atomaren in- und dekrementieren
- Synchronisierung zwischen n Prozessen möglich → Semaphor ist implementierbar

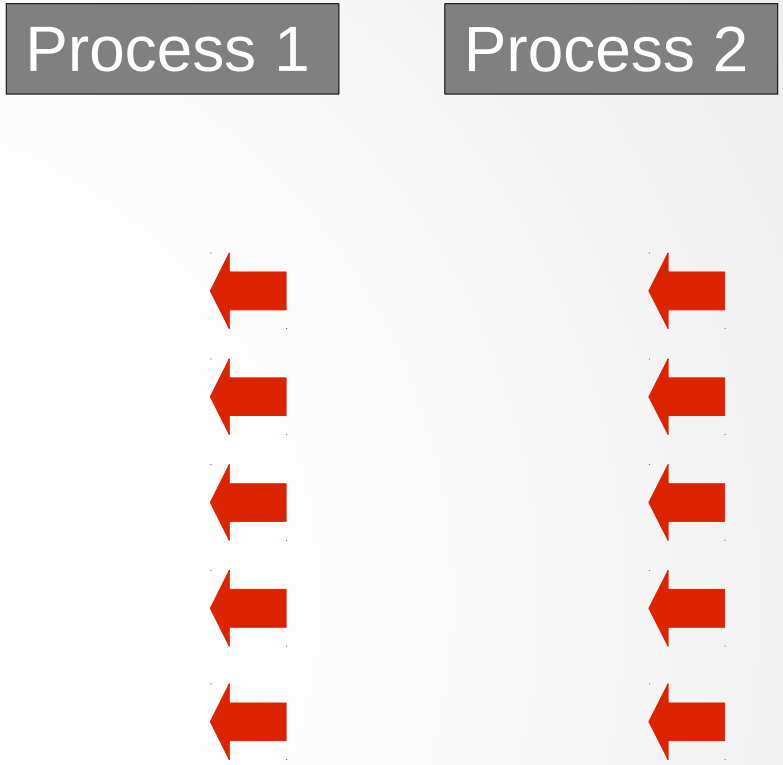
Semaphore

- V(Integer sem)
  - Argument identifiziert Semaphor
  - erhöht sem unteilbar um 1
  - Entspricht freigeben im kritischen Abschnitt

Semaphore

- P(Integer sem)
  - Argument identifiziert Semaphor
  - dekrementiert sem unteilbar um 1
  - sperrt wenn sem = 0
  - Entspricht sperren / probieren im kritischen Abschnitt

Semaphore



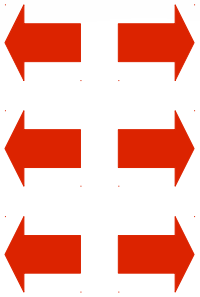

```
Integer free = 1;
process i: begin
   L1:  P(free);
        critical section i;
        V(free);
        //further loop statements
        goto L1;
end;
```

Semaphore

- Alles vorhanden, um parallelisieren zu können
  - Parallele Prozesse
  - Gemeinsame Variablen
  - Synchronisierungsmechanismus, um Parallelität gezielt einzuschränken

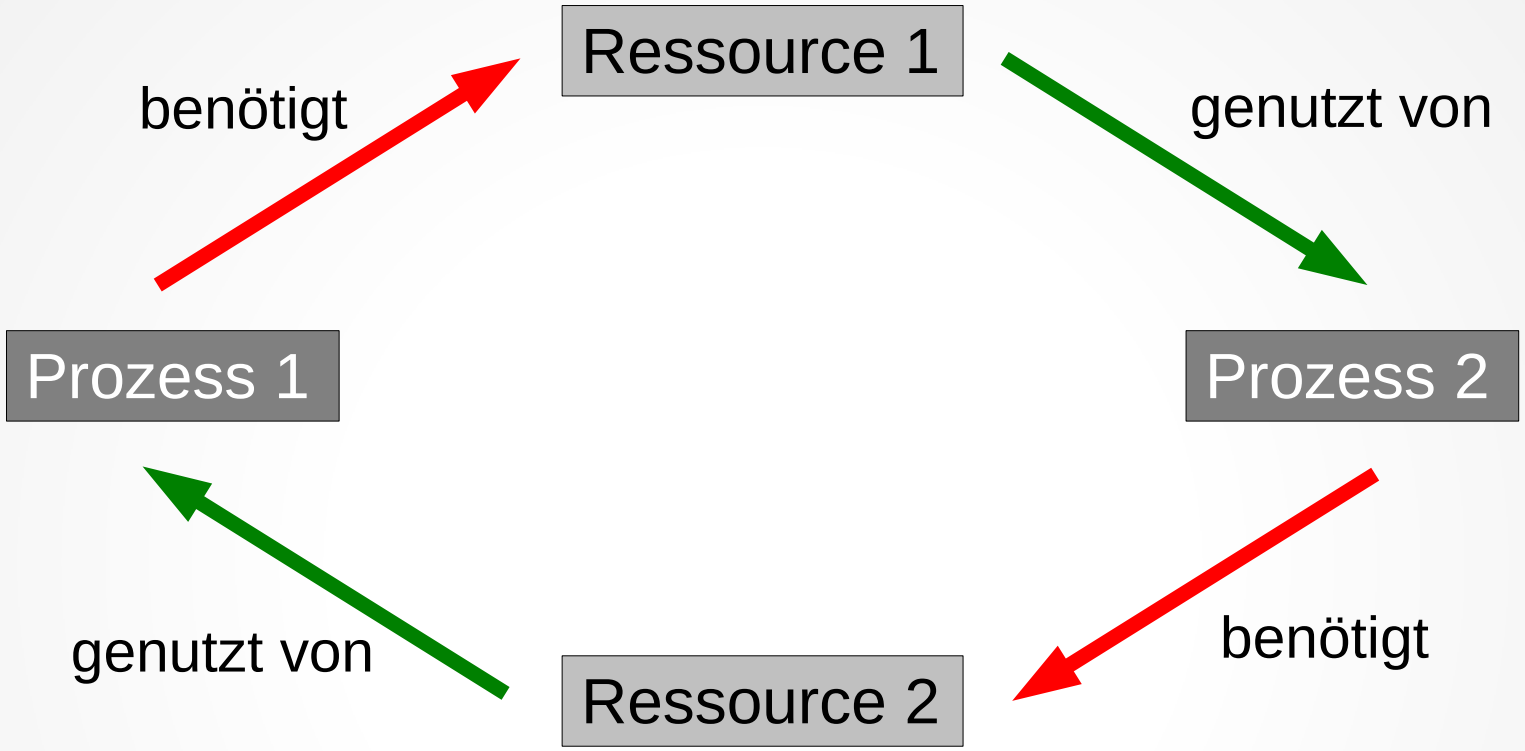
Verklemmung

```
Integer r1 = 1, r2 = 1;
process 1: begin
    P(r1);
    P(r2);
    // do something
    V(r1);
    V(r2);
end;
```



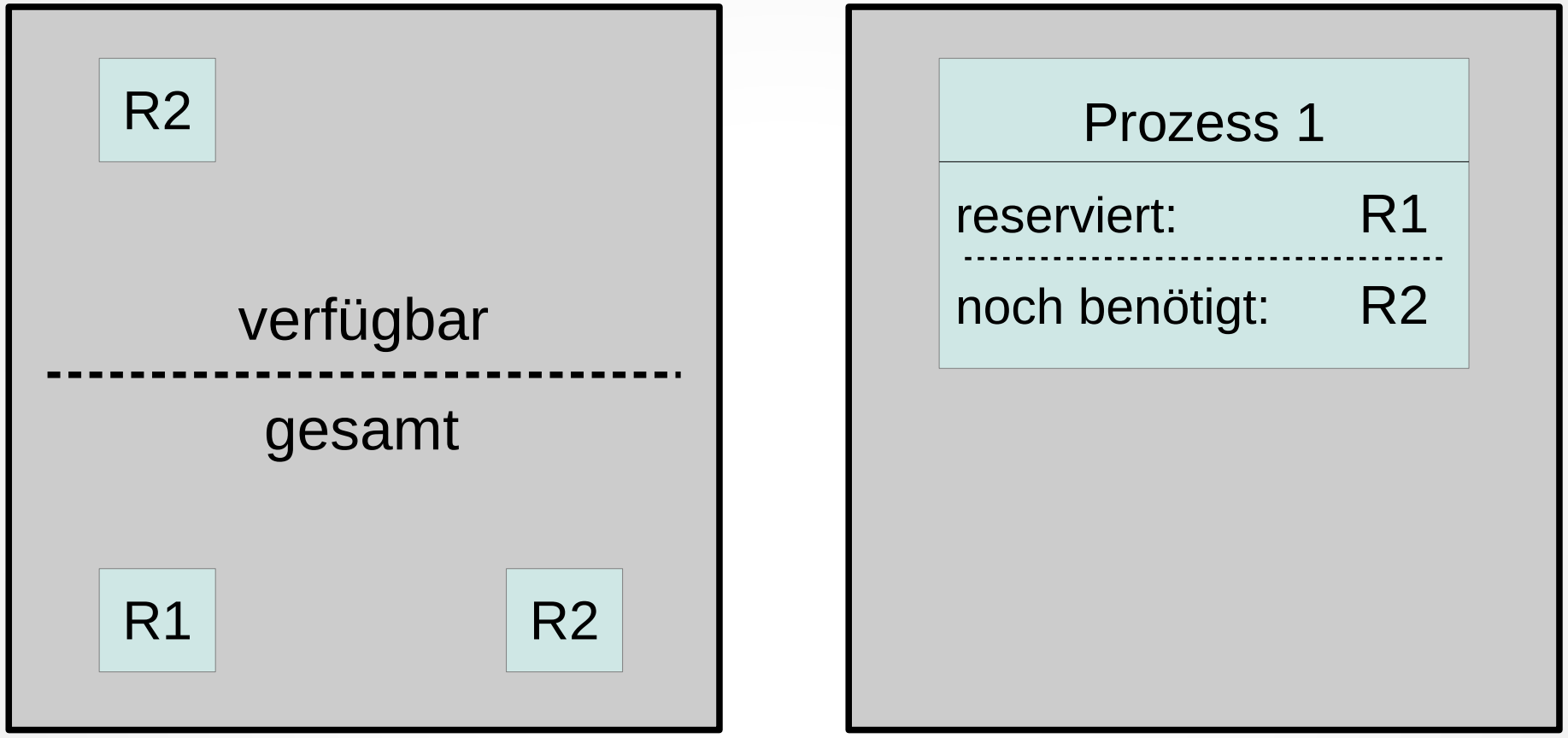
```
process 2: begin
    P(r2);
    P(r1);
    // do something
    V(r2);
    V(r1);
end;
```

Verklemmung



- Prozesse warten auf Betriebsmittel, die jeweils anderen Prozessen zugeteilt sind

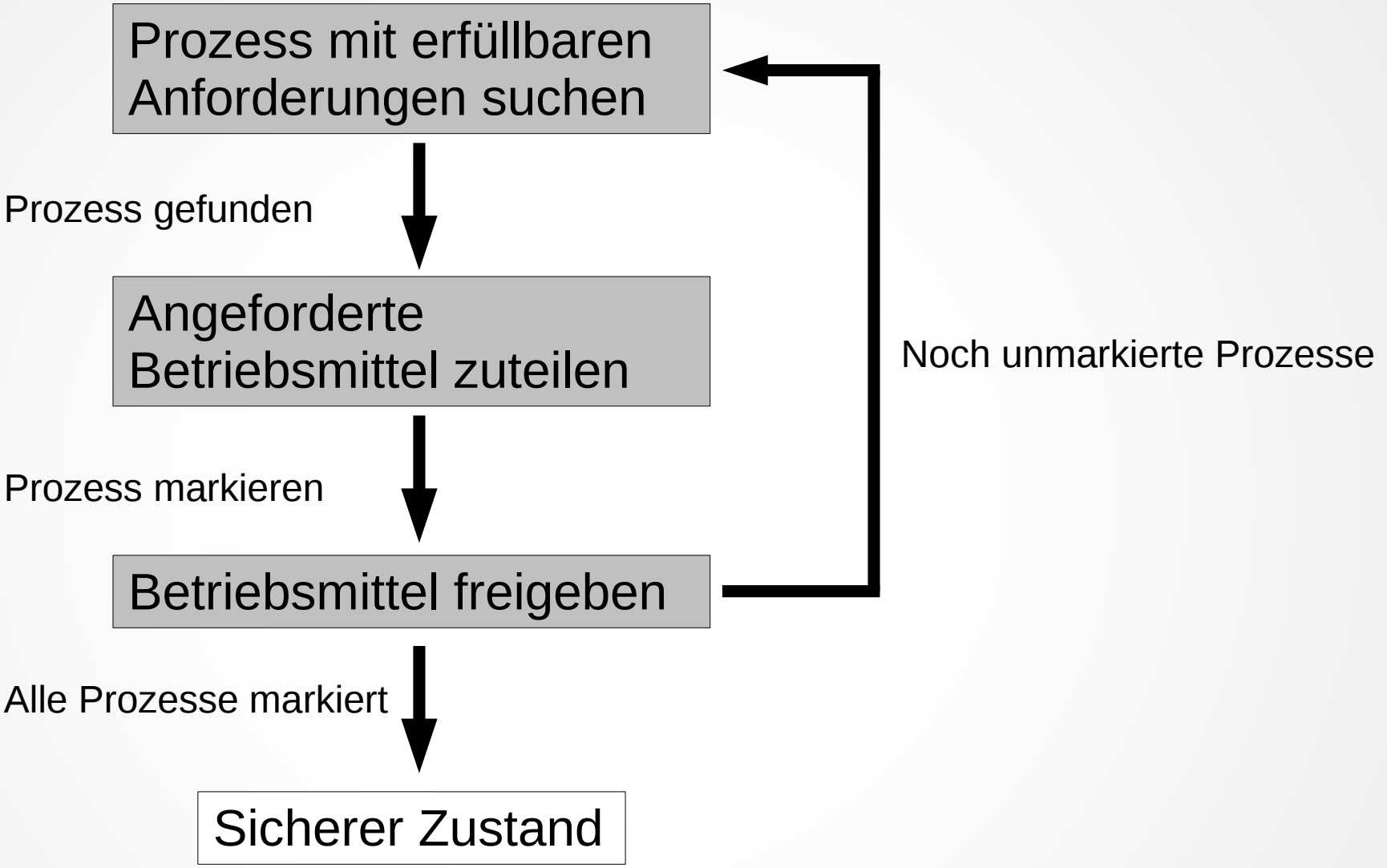
Verklemmung



Verklemmung

- Bankieralgorithmus, um Verklemmung zu vermeiden
- Abarbeitung aller Prozesse wird bei jeder Ressourcenanforderung simuliert
  - Ressourcen werden reserviert und freigegeben
- Nach Ausführung ist garantiert, dass das System verklemmungsfrei bleibt

Verklemmung



Resümee

- Koordination zur fehlerfreien Parallelisierung notwendig
- Eine Möglichkeit: Semaphore
- Einfache Anwendung möglich
- Aufwändige Handhabung auftretender Probleme (z.B. Verklemmung)