Prozeß-Interaktion u. -Kommunikation

Definitionen:

- **Prozeß:** (Vorgang der) Programmausführung
- **Interaktion:** aufeinander bezogenes Handeln
- **Kommunikation:** Bidirektionaler (allg.: multidirektionaler) Austausch von Information
- **Information:** Daten, die in Entscheidungen einfließen
- **Daten:** Angaben, die etwas kennzeichnen.

- **'Angaben':** Nennungen, Behauptungen, Schilderungen, Aussagen, Äußerungen

  über gemessene, erhobene, geschätzte, angenommene

  Zahlenwerte, Zeichen, Funktionen, ...

  ('ABC'; oder: 'quadratisch mit der Zeit')
- **'etwas':** Objekt, Gegenstand, Ereignis, Prozeß, Ablauf, ...

  (z.B.: 'elektrisches Potential' oder 'Gefährdungspotential')
- **'kennzeichnen':** Interpretation macht aus Angaben Daten ("SCHWARZ" als Personenname oder Haarfarbe)

Karl Valentin

Prozeß-Interaktion u. -Kommunikation

- Mit der Möglichkeit zur Interaktion und Kommunikation stellt sich die Frage nach **Nebenläufigkeit** (*concurrency*), **Parallelisierung** u. **Synchronisation** betrachteter Prozesse.
  - **Nebenläufigkeit**: Eignung von Vorgängen zur voneinander unabhängigen Ausführung
  - **Parallelisierung:** Vorbereitung zur quasi-gleichzeitigen oder (im Mehrprozessor-Betrieb) gleichzeitigen Ausführung
  - **Synchronisation:** Erzwingung des Zeitpunkts und/oder der Reihenfolge der Ausführung von Anweisungen unterschiedlicher Prozesse (meist zur Ermöglichung einer kollisionsfreien parallelen Ausführung)
- Wichtige Grundmuster der Prozeßsynchronisation:
  - Nachrichtenübertragung (*message passing*) –vgl.Callback-Übg
  - Schranke (Hürde, Barriere; *barrier*) – vgl. Double Buffering
  - Wechselseitiger Ausschluß (*mutual exclusion*) – s.u.

Anspruch korrekter Prozeßsynchronisation:

Bei sequentiellen Programmen hauptsächlich zwei Arten logischer Fehler:

- Undefinierte Operationen

```
FILE *memo;
char  ch;
/*...*/
fprintf(memo,"A"); fflush(memo);
fseek (memo, -1L, SEEK_CUR);
fscanf(memo,"%1c", &ch);
```

und

- Nichtterminierung.

```
clock_t tNow,tEnd;
tEnd=clock()+10*CLOCKS_PER_SEC;
while(tEnd!=tNow)  tNow=clock();
                   >
```

Durch Parallelität verändern sich Ressourcen-Verfügbarkeit und Daten laufend; dadurch kann neben Daten-Inkonsistenz ein Fehler besonderer Art entstehen:
Blockierung: Warten auf Ereignisse, die nicht eintreten.

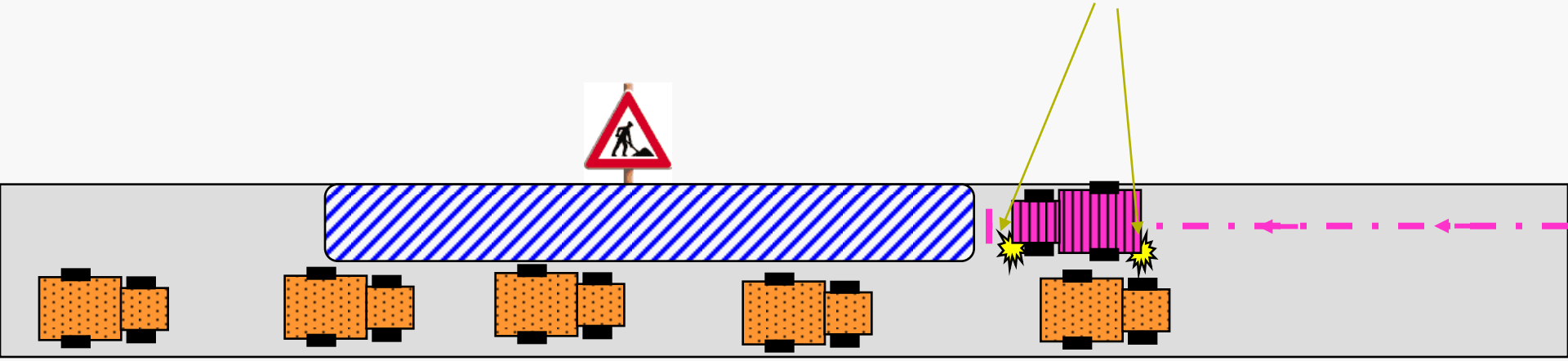
...wollen / ...können

Anmerkungen:

- Ungesteuerte Nutzung gemeinsamer Ressourcen kann bewirken, daß Prozeß-Ergebnisse von Zeitpunkt und Dauer der Ressourcen-Zuweisung abhängen; solche Situationen nennt man **Wettbewerbsbedingungen** (*Race Conditions, RC*).
- Die Verhinderung bzw. Behebung von *RC* ist eine wichtige Aufgabe der Prozeßsynchronisation; dazu werden sog. **kritische Abschnitte** (auch: krit. Bereiche – *Critical Sections / Regions, CS*) eines Prozesses identifiziert, d.h. Programmteile (=Anweisungsfolgen), die in Konkurrenz mit anderen Prozessen dieselben exklusiv nutzbaren Ressourcen beanspruchen.
- Die Prozeßsynchronisation konzentriert sich dann auf die Regelungen zum Eintritt / Verbleib von Prozessen in ihren kritischen Abschnitten.
- Die Realisierung einer kontrollierten Ressourcen-Zuweisung kann u.U. die Entstehung der 3 Blockierungsarten begünstigen:

Prozeß-Interaktion u. -Kommunikation

- Erhält ein Prozeß keine Zuteilung wichtiger System-Ressourcen, so sagt man, daß er **verhungert** (*to starve, starvation*).
  auch: „Hase-und-Igel-Spiel“:
  meist: „durch Prozesse höherer Priorität ausgehungert.“
  Charakteristika: oft von endlicher Dauer;
  Aktivität beschränkt sich auf Akquisitionsversuche



Prozeß-Interaktion u. -Kommunikation

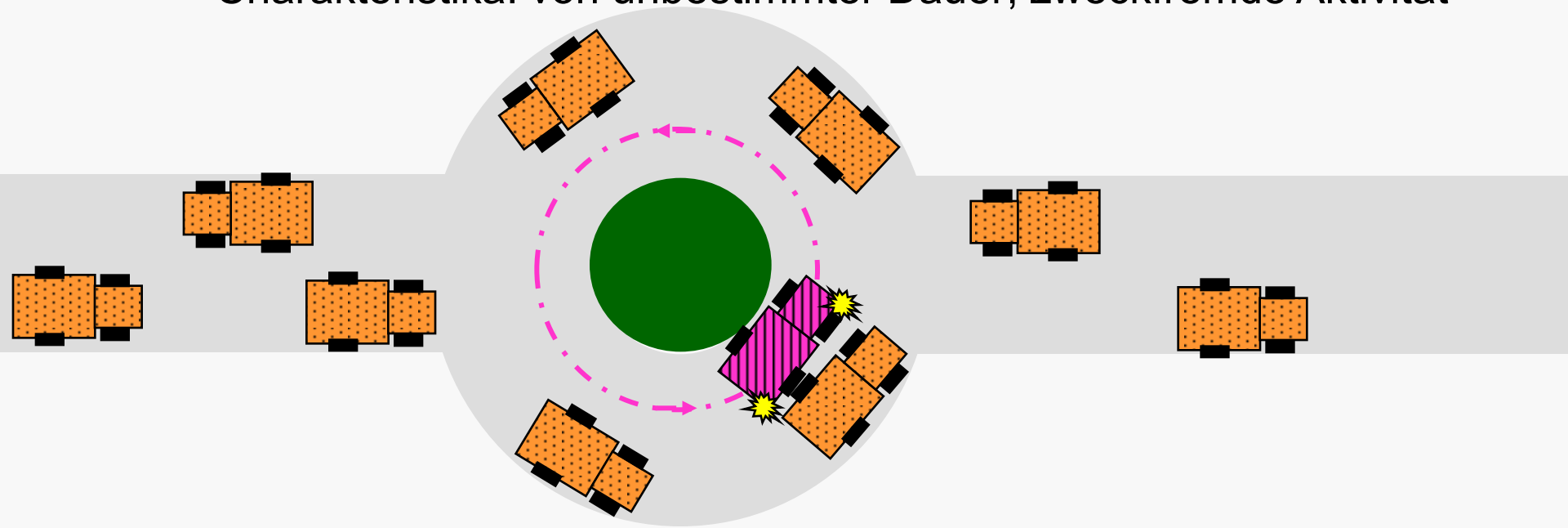
- Eine Situation, in der ein Prozeß beständig um die Nutzung von Systemressourcen gebracht wird, nennt man eine *Blockade* oder einen **Livelock**.

  auch: „nicht abbrechende Ausführung“:

  meist: Interprozeß-Kommunikation, lineare Wartebeziehungen

  „Prozeß verwaltet nur noch sich selbst.“

  Charakteristika: von unbestimmter Dauer; zweckfremde Aktivität



- Eine beständige zyklische Wartebeziehung bezeichnet man als *Verklemmung* oder (einen) **Deadlock**.

  auch: „nicht abbrechende Blockierung“

  meist: Einzel-Reservierung gemeinsam benötigter Ressourcen

  „Prozeß wartet auf Ereignisse, die nicht eintreten können.“

  Charakteristika:

  1. **Stillstand**: Beteiligte Prozesse können ihr Vorhaben nicht fortsetzen (blockiert).
  2. **Permanenz**: Keine spontane Auflösung; zusätzliche Maßnahmen notwendig.
  3. **Zufälligkeit**: Auftreten bei manchen Ausführungen, bei anderen dagegen nicht.

Prozeß-Interaktion u. -Kommunikation

Gleichzeitiges Auftreten von vier Bedingungen ist notwendig und hinreichend für einen Deadlock:

statisch

1. **Exklusiv nutzbare Ressourcen** (*serially reusable*)
   Betriebsmittel, deren Nutzung zu einem konkreten Zeitpunkt höchstens einem Prozeß vorbehalten ist und nicht unterbrochen werden darf (Wechselseitiger Ausschluß – engl. *mutual exclusion*).
2. **Nichtentziehbarkeit** (*non-preemption*)
   Einmal zugewiesene Betriebsmittel werden von Prozessen selbständig o. bei Terminierung zurückgegeben u. können nicht entzogen werden.
3. **Inkrementelle Akquisition**
   Prozesse können Betriebsmittel halten, während sie auf die Freigabe weiterer Betriebsmittel durch andere Prozesse warten.

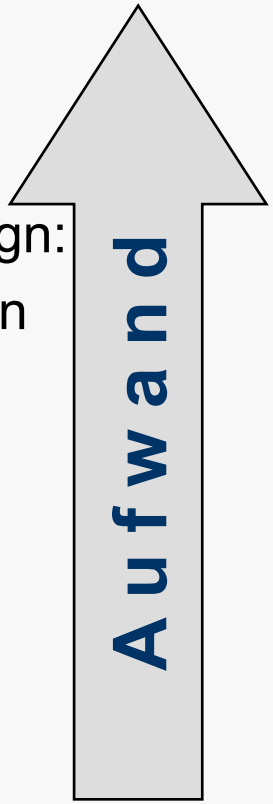
dyna-misch

4. **Zyklische Wartebeziehung** (*circular wait*)
   Es gibt eine geschlossene Kette von Prozessen, in der jeder Prozeß auf die Freigabe eines Betriebsmittels durch seinen Nachfolger wartet.

Deadlock-Freiheit ist schwer zu gewährleisten und kaum zu beweisen (keine allgemein akzeptierte Prüfmethode)!

Anti-Deadlock-Strategien:

- **Vorbeugung** (*Prevention*)
  Prinzipielle Verhinderung der Erfüllung aller 4 Voraussetzgn:
  präemptive Vergabe, gruppenweise Akquisition, Prioritäten
- **Vermeidung** (*Avoidance*)
  Registrierung bereits erfüllter Voraussetzungen
  ggf. Nichtgewährung benötigter Ressourcen
- **Erkennung & Auflösung** (*Detection & Recovery*)
  „Gewaltsamer“ Eingriff von „autorisierter“ Stelle:
  Ressourcen-Entzug, Prozeß-Beendigung



Anti-Deadlock-Maßnahmen:

Verhinderung zyklischer Wartebeziehung:

- Zentrale Warteschlange für Ressourcen-Zuweisung
- Zufallsabhängige Behandlung beteiligter Prozesse
- Einteilung von Prozeß-Gruppen zur Symmetrie-Brechung
- Zuteilung ungleicher Prioritäten

Verhinderung inkrementeller Akquisition:

- Gruppenweise Ressourcen-Reservierung
- Schutz kritischer Abschnitte

Prozeß-Interaktion u. -Kommunikation

Drei Forderungen an die Behandlung kritischer Abschnitte:

- Wechselseitiger Ausschluß (*mutual exclusion*):

  Zu keinem Zeitpunkt darf sich mehr als ein Prozeß (Thread) in seinem kritischen Abschnitt aufhalten.

- Fortschreiten (*progress*):

  Der Prozeß, der als nächster in seinen krit. Abschnitt eintreten darf, soll unter den bereits wartenden ausgewählt werden.

- Begrenztes Warten (*bounded waiting*):

  Einem wartenden Prozeß darf nur begrenzt viele Male der Eintritt in seinen kritischen Abschnitt verwehrt werden.

Lösungsansätze:

- Software-Lösungen (algorithmisch, z.B. innerhalb d. Sw-Umgebung)
- Hardware-Lösungen (Anwendung spezieller Hardware-Funktionen)
- Betriebssystem-unterstützte Lösungen (spezielle BS-Funktionen)

s.a. BS-Vorlesung

Anmerkungen:

- **Software-Lösungen…**
  - … implizieren aktives Warten und daher Effizienz-Probleme
  - ⇨ ... sind bei Ermangelung effizienterer Lösungen einzusetzen
- **Hardware-Lösungen…**
  - … als Interrupt-Sperrung bei Ein-Prozessor-Systemen
    - … vereiteln auch den Wechsel zu anderen Tasks (Multitasking)
    - … sind fatal bei Programmabstürzen u. Mißbrauch durch Programmierer
  - … als Interrupt-Sperrung bei Mehrprozessor-Systemen
    - … sind praktisch wirkungslos, wenn auf nur einen Prozessor beschränkt
  - … mit speziellen Hardware-Funktionen für Ein-Prozessor-Systeme
    - … ermöglichen wechselstg. Ausschl. mit atomaren read-modify-write-Fkt.
    - … verursachen bei anderen Prozessen aktives Warten (Effizienz-Probl.)
    - … lassen die Maßnahmen für Fortschreiten & begrenztes Warten offen
  - … mit speziellen Hardware-Funktionen für Mehrprozessor-Systeme
    - … können zudem wg. Alleinverfügung über d. Bus d. System blockieren
  - ⇨ ... sind nützlich vor allem innerhalb des Betriebssystems

Prozeß-Interaktion u. -Kommunikation

Wichtigste BS-unterstützte Lösung: das Semaphor (E.W.Dijkstra, 1965)

Wirkungsweise – hier: als Sperrvariable (spin lock) mit aktivem Warten:

Das allgemeine Semaphor:

```
void P(int *R)
{(*R)--; /*Res.-Anforderg!*/
  while (*R < 0);
}

void V(int *R)
{  (*R)++;
}
```

(Initialisierung mit **R=N;** für **N** Ressourcen)

Das Binäre Semaphor (der Mutex):

```
void P(int *bS)
{ while (*bS==0);
   *bS=0;
}

void V(int *bS)
{ *bS=1;
}
```

(Initialisierung mit **bS=1;**)

Namen historisch gewachsen; bS äquivalent zu allgem. Semaphor für **N=1**

Anwendung:

```
P(&bS);   /* "Passieren"   */
/*...*/   /*krit.Abschnitt*/
V(&bS);   /* "Verlassen"   */
```