

### Aufgabe 1 (10 Punkte)

Die zweite Version des `LinearBuffer` aus 3.1.2 ist so konzipiert, dass Sender und Empfänger überlappend tätig werden können (wenngleich nicht immer fehlerfrei).

a) (2 P.) Die folgende Variante von `send` birgt eine weitere Fehlerquelle. Zeigen Sie dies durch Angabe eines Beispiels mit entsprechenden Zeitschnitten!

```
public void send(M m) throws Overflow {
    if(count()==size) throw new Overflow();
    synchronized(this) { // sender exclusion
        cell[rear] = m;
        ...
    }
}
```

b) (3 P.) Die folgende Variante von `recv` ist auch fehlerhaft; warum?

```
public M recv() throws Underflow {
    synchronized(cell) { // receiver exclusion
        if(count()==0) throw new Underflow();
        M m = cell[front];
        front = (front+1)%(size+1); }
    return m; }
}
```

Kann der Fehler dadurch behoben werden, dass `m` vor dem `synchronized` vereinbart wird?

c) (5 P.) Erweitern Sie die Klasse um eine zusätzliche Methode `urgent`, die eine dringliche Nachricht *vorn im Puffer* ablegt! (Achtung: die richtige Ausschluss-Synchronisation ist hier etwas kniffliger.)

### Aufgabe 2 (16 Punkte)

a) (12 P.) Erweitern Sie die Lösung von Aufgabe 2.4 (siehe Webseite), indem Sie aus der einfachen Uhr einen *Wecker* machen. Nach wie vor soll ein Faden das Uhrwerk realisieren, während ein anderer auf Eingaben des Benutzers reagiert. Ein dritter Faden soll das Lätewerk realisieren.

b) (4 P.) Verfassen Sie eine gut lesbare *Gebrauchsanweisung* für Ihren Wecker!

