Kurs OMSI
im WiSe 2012/13

Objektorientierte Simulation mit ODEMx

Prof. Dr. Joachim Fischer
Dr. Klaus Ahrens
Dipl.-Inf. Ingmar Eveslage

fischer|ahrens|eveslage@informatik.hu-berlin.de

3. Prozess-Scheduling

1. Aufgaben von Klasse Simulation (Wdh.)
2. Process-Listen eines Simulationskontextes
3. Allgemeines Process-Scheduling
4. Weitere Process-Funktionaliät
5. Prozesswarteschlangen: ProcessQueue
6. Spezielles Process-Scheduling (Memory)

PortTail-, PortHead- Konstruktoren (Wdh.)

```
PortHead (Simulation * sim,
          Label portName,
          PortMode m = WAITING_MODE,
          unsigned int max = 10000 )

PortTail (Simulation * sim,
          Label portName,
          PortMode m = WAITING_MODE,
          unsigned int max = 10000 )
```

PortMode-Aufzählungstyp

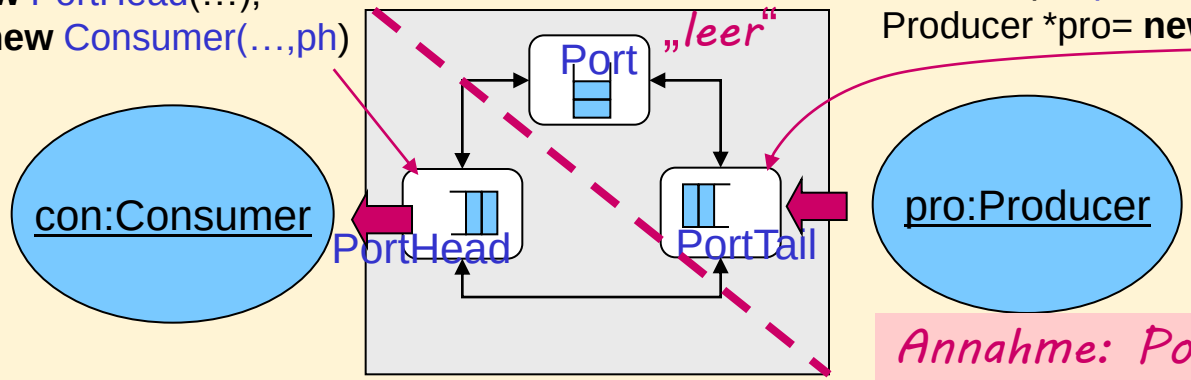
ERROR_MODE	Fehler, falls voll/leer
WAITING_MODE	Prozesswechsel, falls voll/leer
ZERO_MODE	Leeranweisung, falls voll/leer (0 als Return-Wert)

Weitere Port-Funktionalität: cut(), splice()

„vor cut()-Anwendung“

```
PortHead *ph= new PortHead(…);
Consumer *con= new Consumer(…,ph)
```

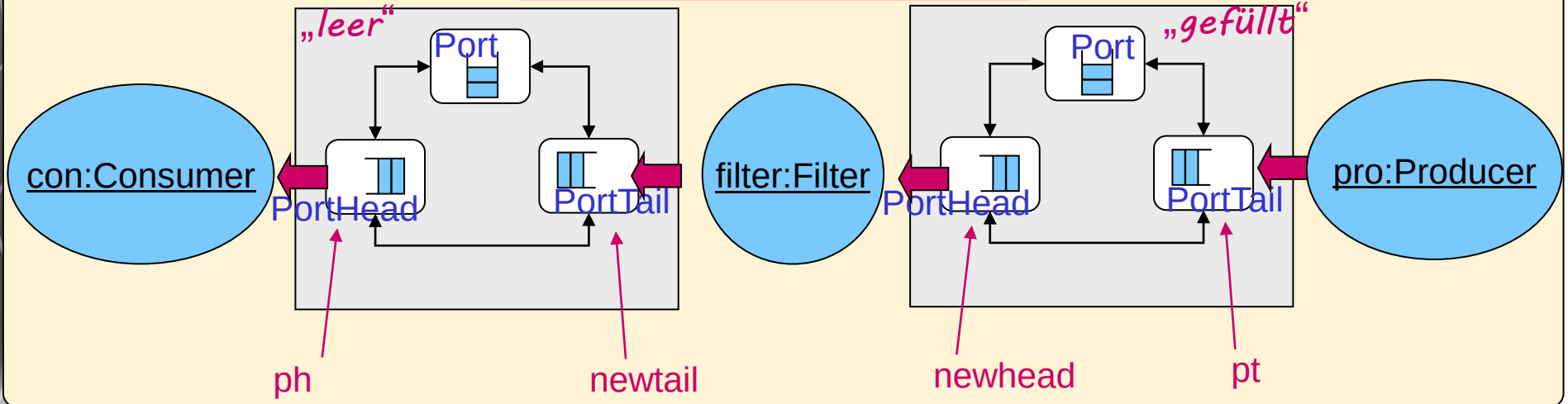
```
PortTail *pt= ph.tail()
Producer *pro= new Producer(…, pt)
```



Annahme: Port sei gefüllt

```
PortHead *newHead= ph->cut();  //bei Ergänzung eines neuen PortHead-Zugangs
PortTail *newTail= ph->tail(); // Ergänzung eines neuen PortTail-Zugangs mit leerem Port

Filter *filter= new Filter(…, newHead, newTail)
```

„nach cut()-Anwendung“



Klasse Memory (Wdh.)

Basisklasse für
- PortHeadT, PortHead,
- PortTailT, PortTail
- Timer,
- WaitCondition

```
class Memory {
public:
        Memory( base::Simulation& sim, const data::Label& label, Type type, MemoryObserver* obs = 0 );
        virtual ~Memory();

        virtual bool remember( base::Sched* newObject );
        virtual bool forget( base::Sched* rememberedObject );

        bool processIsWaiting (base::Process& process) const;
        virtual void eraseMemory();

        virtual bool isAvailable();
        bool waiting() const;

        SizeType countWaiting() const;

        virtual Type getMemoryType() const;

        virtual void alert();


privat:
        std::list< Sched * > memoryList //Liste vermerkter Sched-Objekte (Zeiger)
```

```
enum Type {
    TIMER,
    PORTHEAD,
    PORTTAIL,
    CONDITION,
    USERDEFINED
};
```

Spezialisierungen legen Semantik von isAvailable fest

Wait-Anwendungsbeispiel (Wdh.)

- Process-Member-Funktion wait

```
Memo* wait (Memo* m0,
            Memo* m1= 0, Memo* m2= 0, Memo* m3= 0, Memo* m4= 0, Memo* m5= 0 )
```

liefert eines der Memo-Objekte zurück, sobald dieses „Verfügbarkeit“ liefert;
bis dahin bleibt der Aufrufer blockiert

- Beispiel

```
PortHead *p1, *p2, *p3. *p;
```

```
...
p= wait (p1, p2, p3);
...
```

Aufrufer-Prozess wartet(blockiert) bis in einem der Buffer p1, p2, p3
eine Nachricht hinterlegt worden ist

```
Memory *m;
PortHead *ph;
PortTail *pt;
Timer t;
```

```
...
m= wait (ph,pt, t);
switch (m->getMemoryType()) {
   case TIMER: ...
   case PORTHEAD: ...
   default: ...
}
```

Aufrufer-Prozess wartet(blockiert) bis in einem der Puffer ph eine Nachricht abgelegt wird **oder** in einem Puffer pt Platz geworden ist **oder** ein Timeout anliegt

Rückgabewert von wait

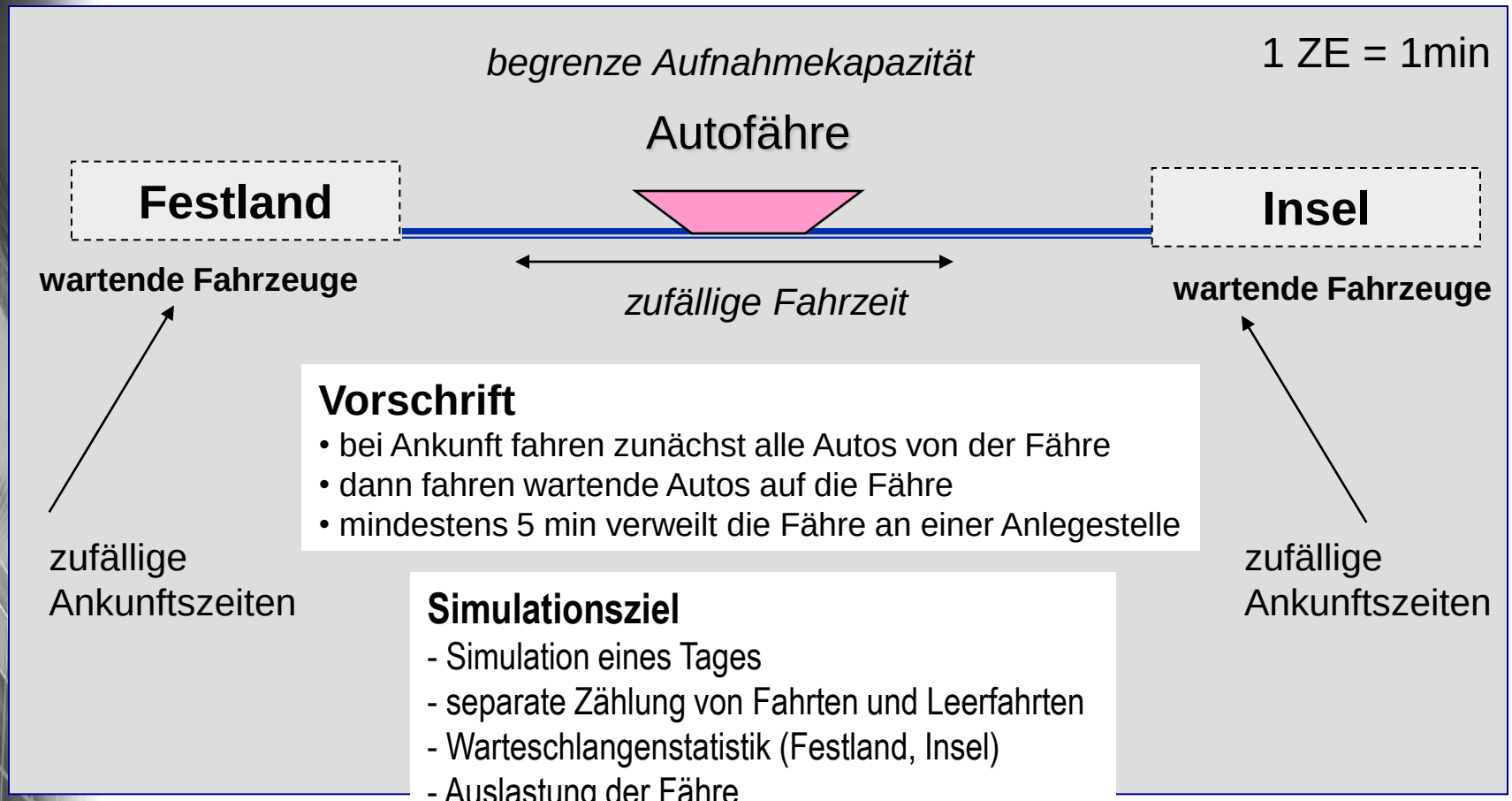
polymorphe Memory-Zeiger

```
…
*Memory  m= wait (m1, m2, m3);
…
```

- Aufrufer-Prozess vermerkt sich in lokale Process*-Liste von m_1, m_2 und m_3 (falls deren „Verfügbarkeit“ nicht gegeben ist und blockiert
- wird auf den blockierten Prozess durch einen nebenläufigen Prozess ein **interrupt**() angewendet, verlässt dieser die m_1-, m_2- und m_3-Liste
- und beendet die **wait()**-Anweisung mit der Rückgabe des NULL-Zeigers (externe Unterbrechung der Blockierung)
- Sobald die „Verfügbarkeit“ von mindestens einem m_i gegeben ist, wird **wait** mit Rückgabe von m_i verlassen (der Prozess hat zuvor die lokalen Listen von m_1, m_2 und m_3 verlassen)
- Sollten mehr als zwei m_i‘s „Verfügbarkeit“ anzeigen, liefert **wait** den ersten von ihnen in der Parameterliste

Beispiel: Autofähre

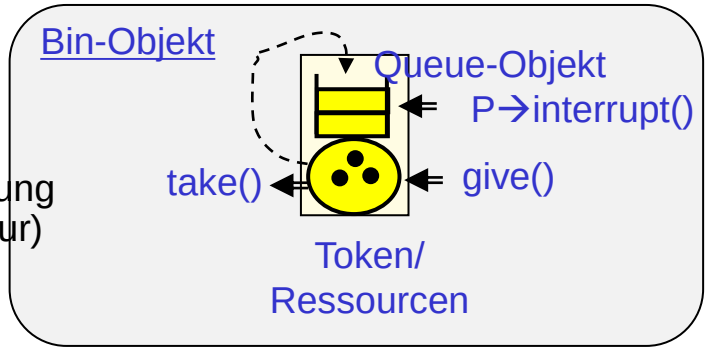
- Wortmodell und informale Darstellung



4. ODEMx-Modul Synchronisation: Bin, Res

- Verwaltung geteilt genutzter Ressourcen mittels **Bin** und **Res**
- Die Klasse **Bin**
- Behandlung von Unterbrechungen
- Die Klasse **Res**
- Ein Beispiel: Autofährbetrieb (Einsatz von **Bin** und **Res**)

Klassen Bin und Res



Gemeinsamkeiten

- verwalten **Ressourcen** und deren geteilte Nutzung
  Ressourcen sind abstrakt (Token ohne Struktur)

- bieten Funktionen zur
  - **Anforderung** und
  - **Freigabe** von Ressourcen

- ein Prozess wird in seiner Anforderung **blockiert**
  und im Queue-Objekt **vermerkt**,
  falls die Ressourcen in geforderter Anzahl z.Z. **nicht** zur Verfügung stehen

- ein blockierter Prozess wird **fortgesetzt**,
  sobald die von ihm benötigte Anzahl von Token zur Verfügung steht
  bei Entnahme der Token

Unterschiede

- Bin kann **beliebig viele** Token (unabhängig von der initialen Ausstattung) aufnehmen, erlaubt dynamische Vernichtung und Generierung von Token
- Res **beschränkt** die maximal verfügbare Token-Menge,
  geht i.d.R. von einer unveränderlichen Token-Menge je Res-Objekt aus

4. ODEMx-Modul Synchronisation:
Bin, Res

- Verwaltung geteilt genutzter Ressourcen mittels Bin und Res
- Die Klasse Bin
- Behandlung von Unterbrechungen
- Die Klasse Res
- Allgemeine Prozesslokalisierung (nicht Bin/Res-spezifisch)
- Ein Beispiel: Autofährbetrieb (Einsatz von Bin und Res)

Klasse Bin: Konzept

- besitzt zur Verwaltung blockierter Prozesse (privates) Queue-Objekt:

  Queue* takeWait

- per Konstruktorparameter wird initiale Tokenzahl (>=0) vermittelt

  (Defaultwert 0)

- Member-Funktion int take(n), n>0
  - Rufer (**Nehmer-Rolle**) wartet evtl. mit Null-Zeit als „Durchläufer“
  - i.d.R. wartet der Rufer solange (d.h. ohne Unterbrechung),

    bis n Token verfügbar sind
  - aber: Warteaktion ist prinzipiell durch nebenläufigen Prozess unterbrechbar
    - Rückgabewert zeigt erfolgte Unterbrechung an: 0 (sonst n)

- Member-Funktion give(n), n>0
  - Rufer (in **Geber-Rolle**) veranlasst

    (zeitlose) Eingabe oder Rückgabe von Token,

    verbunden mit Aktivierung evtl. wartender Nehmer-Prozesse
  - Token müssen nicht zwingend demselben Bin-Objekt zurück gegeben werden

Bin- Semantische Präzisierung

- Prozesse, die auf verfügbare Token warten,
  werden in einer Warteschlange (lokales Objekt von Bin) erfasst
  FCFS-Strategie

- bedeutet für folg. angenommenen Fall:
  - Bin-Objekt o habe 2 Token
  - P1 wartet am längsten (benötigt 3 Token) in o
  - P2 wartet ebenfalls in o (benötigt 1 Token)

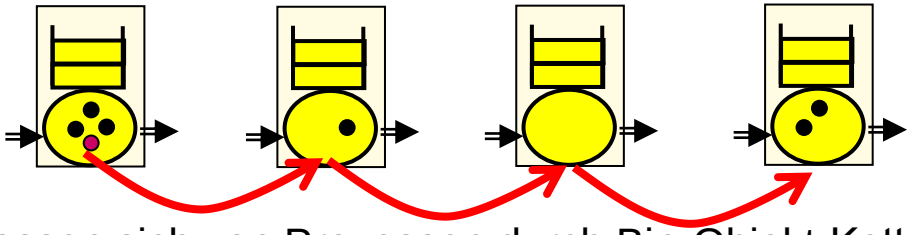
  P2 wäre mit der freien Tokenzahl von o zufrieden,
  darf P1 dennoch nicht überholen

Bin-Anwendungen

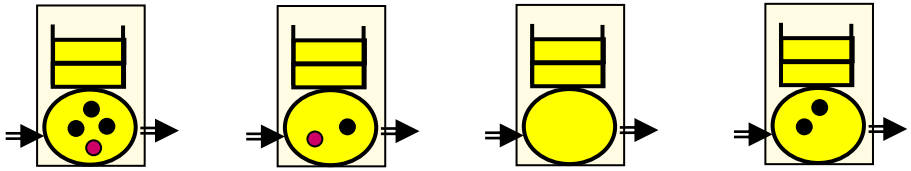
Token haben jedoch
bei einfachen Bins **keine** Identität

Später:
Bin-Templates

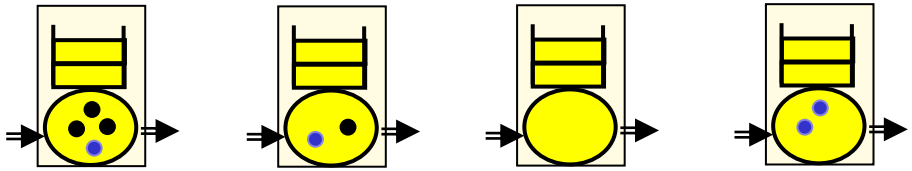
Basistyp
der Token
kann
als Parameter
vermittelt
werden



en lassen sich von Prozessen durch Bin-Objekt-Ketten ‚bewegen'

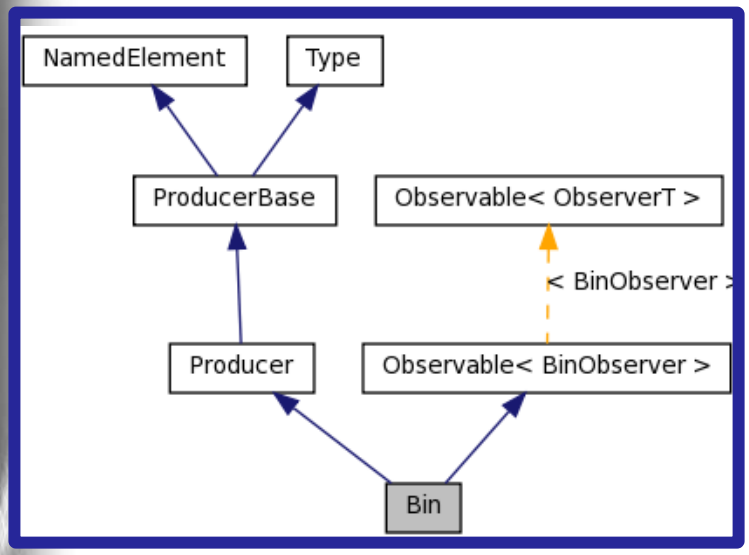


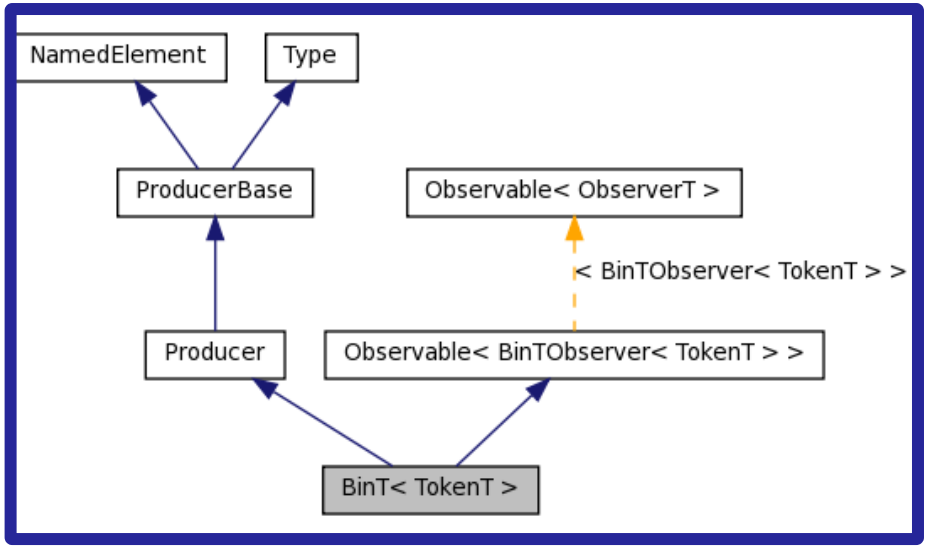
Token lassen sich von Prozessen in Bin-Objekt-Ketten erzeugen
(ohne dass sie zuvor irgendwo entnommen wurden)

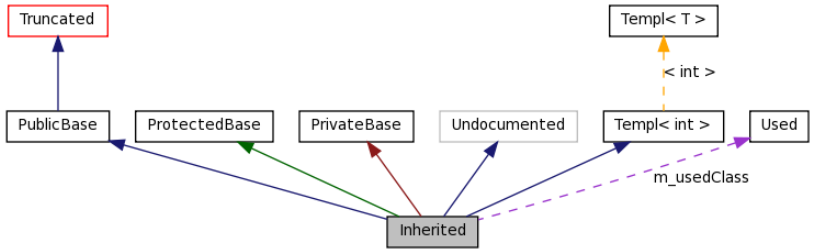


Token lassen sich von Prozessen in Bin-Objekt-Ketten vernichten
(ohne dass sie danach irgendwo abgelegt werden)

Bin- Klassenhierarchie







Klasse Bin (Public-Member-Funktionen)

Public Member Functions

	Bin (base::Simulation &sim, **const** data::Label &label, std::size_t initialTokens, BinObserver *obs=0)
	~Bin ()
const base::ProcessList &	getWaitingProcesses () **const**
	//Get list of blocked processes.

Token management

std::size_t	take (std::size_t n)
	Take n token.
void	give (std::size_t n)
	// Give n token.
std::size_t	getTokenNumber() **const**
	// Number of tokens available.

Klasse Bin (Private-Attribute)

```
private:
        Simulation* env;
        unsigned int tokenNumber;
        unsigned int initToken;
        Accum tokenStatistics;
        unsigned int users;
        unsigned int providers;
        double sumWaitTime;

      // process management
        Queue takeWait;
```

später

Aufnahme blockierter Prozesse

Implementierung von Bin::take

```
std::size_t Bin::take( std::size_t  n ) {
    // resource handling only implemented for processes
        if( ! getCurrentSched() || getCurrentSched()->getSchedType() != base::Sched::PROCESS ) error ...
        base::Process* currentProcess = getCurrentProcess();
    // compute order of service, insert the process into the queue
        takeQueue_.inSort( currentProcess );
    // if not enough tokens or not the first process to serve
        if( n > tokens_ || currentProcess != takeQueue_.getTop() ) {
            ...
    // statistics
            base::SimTime waitStart = getTime();
    // block execution
            while( n > tokens_ || currentProcess != takeQueue_.getTop() ) {
                currentProcess->sleep();
                if( currentProcess->isInterrupted() ) {
                    takeQueue_.remove( currentProcess );
                    return 0;
                }
            }
    // block released here
    // statistics, log the waiting
        ...
        }
    // remove from list
        takeQueue_.remove( currentProcess );
    // awake next process
        awakeFirst( &takeQueue_ );
    return n;
}
```

aktiviert seinen Nachfolger,
dieser wird Verfügbarkeit für sich prüfen

Implementierung von Bin::give

```
void Bin::give( std::size_t  n ) {
      // trace
      ODEMX_TRACE << log … ;
      // observer
      …;
      // release tokens
      tokens_ += n;
      // trace
      ODEMX_TRACE << log … ;
      // statistics
      …;
      // observer
      …;
      // awake next process
      awakeFirst( &takeQueue_ );
}
```

aktiviert nur den unmittelbar nächsten Nachfolger

sollten mehrere Prozesse hintereinander in der Wartschlange durch die Token-Rückgabe ihre jeweiligen Token-Forderungen fortgesetzt werden können,
ist deren Aktivierung durch sukzessiven Vollzug der take-Operation beginnend mit diesem einen Nachfolger gesichert
(siehe take-Operation)

4. ODEMx-Modul Synchronisation:
Bin, Res

- Verwaltung geteilt genutzter Ressourcen mittels Bin und Res
- Die Klasse Bin
- Behandlung von Unterbrechungen
- Die Klasse Res
- Beispiel: Autofährbetrieb (Einsatz von Bin und Res)

Unterbrechung

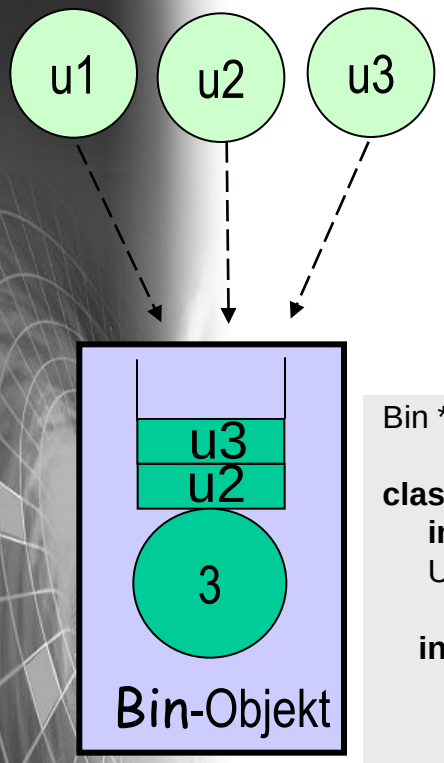
…

- des Wartevorgangs auf verfügbare Ressourcen (Token)
- der zeitlich begrenzten Benutzung der entnommenen Ressourcen (Token)

in beiden Fällen mittels interrupt !

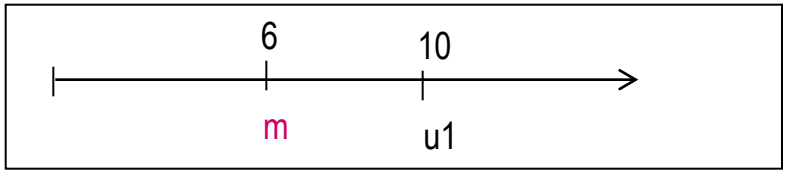
```
int res= service-> take(…); // liefert 0 Token, falls
                            // Blockierung unterbrochen worden ist
                            // sonst Anzahl der entnommenen Token

holdFor (…);                // Verzögerung um Nutzungszeit der entnommenen
                            // Token
if (interupted() ) { …};
service-> give(…);          // Rückgabe von Token
```

Variante B: Unterbrechung des Wartevorgangs



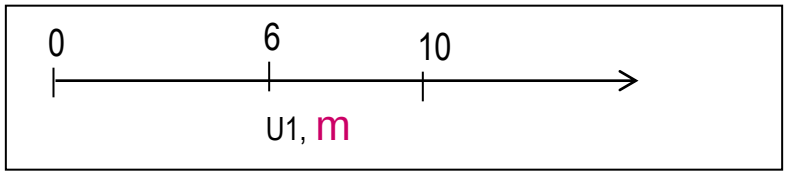
User-Objekte,
die gleichzeitig
gestartet werden

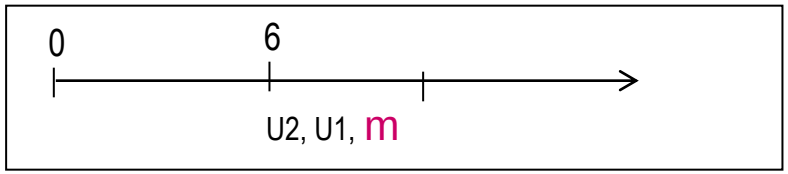
ein weiterer Prozess **m**
zum Zeitpunkt 6.0




```
class Manager:
        public Process {
   ...

   int Manager::main() {
     ...
       u1->interrupt();
     ...
     }
}
```

```
Bin *service= new Bin ("service", 3);

class User: public Process {
    int no;
    User (int n), no(n), Process (…) { ...};

   int User::main () {
      int ret;

      ...
      ret= service-> take (2);
      if (ret) {
         holdFor (10.0);
         service-> give (2);
      } else  ….
                  evtl. Nutzung eines anderen
                  Bin-Objektes nach Warteabbruch
    }
}
```





4. ODEMx-Modul Synchronisation:
Bin, Res

- Verwaltung geteilt genutzter Ressourcen mittels Bin und Res
- Die Klasse Bin
- Behandlung von Unterbrechungen
- Die Klasse Res
- Beispiel: Autofährbetrieb (Einsatz von Bin und Res)

Besonderheiten von Res

- Konstruktor
  sowohl die **initiale**(>=0) als auch **maximale** Tokenzahl (>0) ist einzustellen

- **unsigned int** acquire(n), n>0
  - unterbrechbare Warteaktion (evtl. mit Null-Zeit als „Durchläufer“)
  - ohne Unterbrechung wartet der Rufer solange, bis n Token verfügbar sind
  - Rückgabewert zeigt Unterbrechung an: 0, sonst n

- **unsigned int** release(n), n>0
  - Eingabe von Token
  - Token müssen nicht unbedingt diesem Res–Objekt vorab entnommen sein
  - wird maximale Tokenanzahl überschritten:
    Fehlermitteilung und Fortsetzung bei Ignorierung der überschüssigen Token

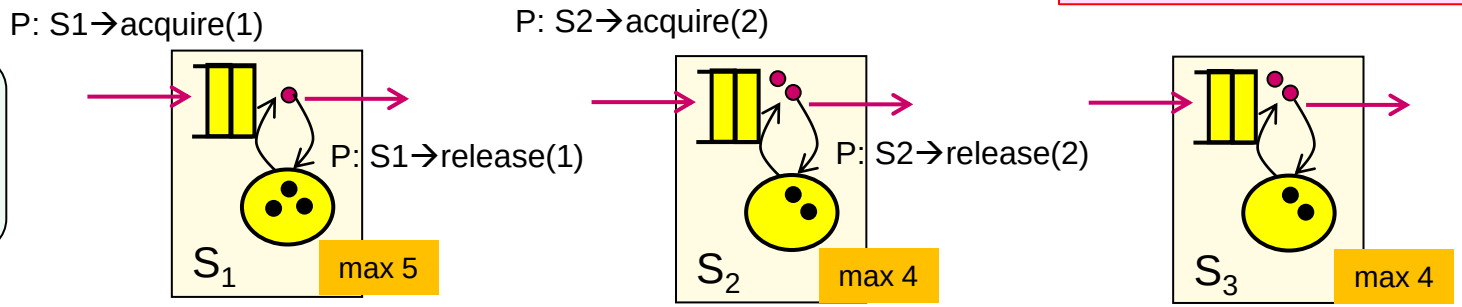
- **Sonderfunktionen zur dynamischen Korrektur der maximalen Token-Zahl**
  - **void** control(n), n>0 Erhöhung der maximalen Token-Anzahl
    bei Aktivierung des nächsten wartenden Prozesses

  - **void** uncontrol(n), n>0 Reduktion
    falls n Token überhaupt noch verfügbar sind, sonst Anpassung von n

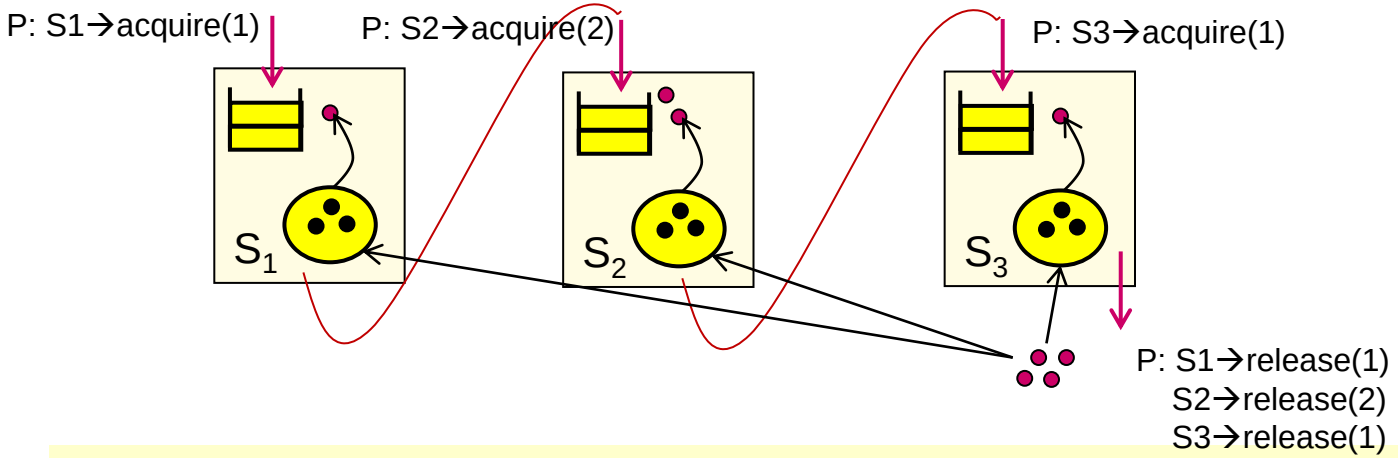
Res-Anwendungen

Token haben jedoch weder Identität noch Typ

- freie Ressource
- belegte -“-



Fall a) Prozesse bewegen sich durch Stationsketten (Res-Objekte), benötigen/benutzen Ressourcen einer Ressource und geben diese nach der Nutzung an die aktuelle Station komplett zurück



Fall b) Prozesse benötigen Ressourcen unterschiedlicher Sorten (Res-Objekte) für einen Arbeitsgang

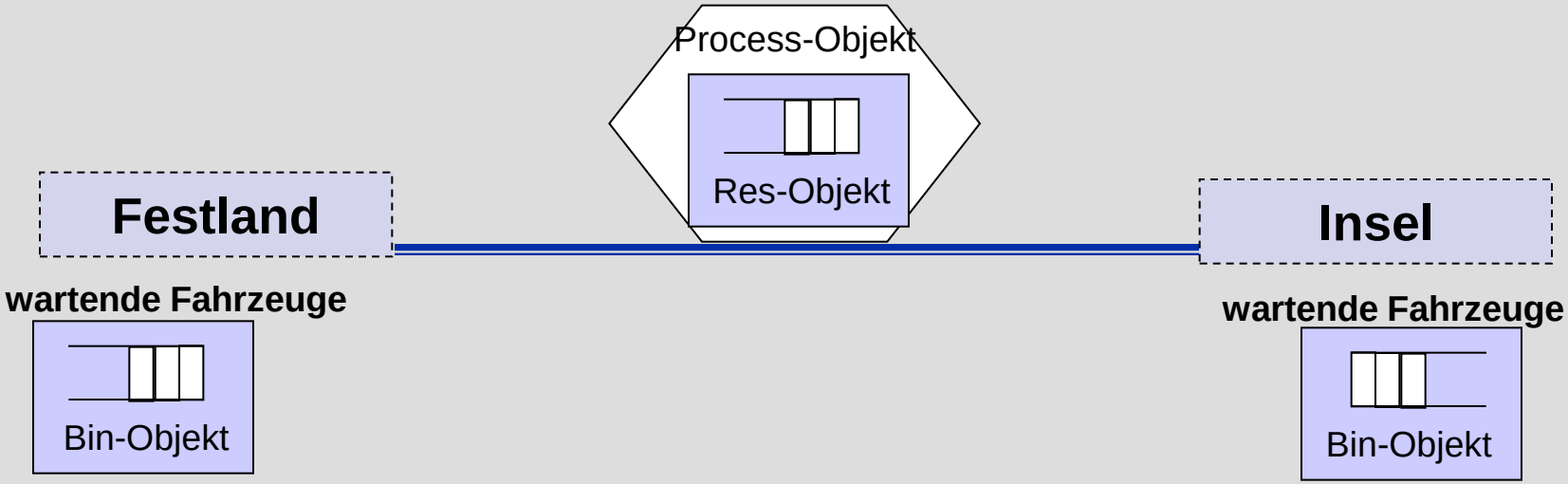
4. ODEMx-Modul Synchronisation:
Bin, Res

- Verwaltung geteilt genutzter Ressourcen mittels Bin und Res
- Die Klasse Bin
- Behandlung von Unterbrechungen
- Die Klasse Res
- Beispiel: Autofährbetrieb (Einsatz von Bin und Res)

Beispiel: Autofähre

- Erfassung von Bewertungsgrößen

1. Auslastung der Fähre (Res-Objekt)
2. Beladungsprofil (Tally-Objekt)



3. separate Zählung der Leer- und Frachtfahrten (Count-Objekte)
4. Benutzungsprofile der Bin-Objekte

Globale Größen

Hauptprogramm wird Zeiger mit Objekten verbinden

```
/* example Ferry.cpp Ferry simulates ... */
#include <odemx/odemx.h>

odemx::base::Simulation& sim = odemx::getDefaultSimulation();

odemx::synchronization::Bin* q[2];
odemx::synchronization::Res* fload;

odemx::random::ContinuousDist* next [2];
odemx::random::ContinuousDist* crossing;

odemx::statistics::Tally *av_load;
odemx::statistics::Count *trips, *empties;

double minStayTime;
```

zwei Warte-Listen von Fahzeugen(Token): Anlegestellen Festland und Insel

eine Warte-Liste von Fahzeugen(Token): Ladebereich der Fähre

zwei Generatoren von Pseudo-Zufallszahlen (polymorph für stetige Vfktionen): hier später: Exponential-Verteilung

ein Generator von Pseudo-Zufallszahlen (polymorph für stetige Vfktionen): hier später: Normal-Verteilung

Statistisches Beladungsprofil (min, max, EW, SA...)

zwei Zähler

Hauptprogramm

```
main( int argc, const char* argv[]) {
        q[0] = new odemx::synchronization::Bin(sim, "mainland", 3);
        q[1] = new odemx::synchronization::Bin(sim, "island", 1);

        fload = new odemx::synchronization::Res(sim, "ferry load", 4, 4);

        next[0] = new odemx::random::NegativeExponential(sim, "mainland", 0.1);
        next[1] = new odemx::random::NegativeExponential(sim, "island", 0.1);
        crossing = new odemx::random::Normal(sim, "crossing", 7.5, 0.5);

        trips = new odemx::statistics::Count(sim, "trips");
        empties = new odemx::statistics::Count(sim, "empty trips");
        av_load = new odemx::statistics::Tally(sim, "av.load");
        minStayTime = 5.0;

        Arrival *a1 = new Arrival(0);
        a1->activateAt(380);
        Arrival *a2 = new Arrival(1);
        a2->activateAt(380);
        Ferry *f = new Ferry(fload);
        f->activateAt(7 * 60); //Start um 7.00 Uhr
        sim.run(); //Abbruch durch Ferry
}
```

Festland: 3 Fahrzeuge zur Zeit 0·0

Insel: 1 Fahrzeug zur Zeit 0·0

Maximale und aktuelle Kapazität der Fähre: 4

Pseudo-Zufallszahlengeneratoren (Exponential-, Normalverteilung)

Zähler für Fähre-Reisen (Hin und zurück)

Zähler für Leerfahrten

Beladungsprofil (min, max, EW, SA)

Ankunftsprozess-Klasse

```
class Arrival : public odemx::base::Process {
        public: int side;
        Arrival (int s): odemx::base::Process(sim, "Arrival"), side(s) {}
        int main ();
};

int Arrival::main() {
        for (;;) {
                holdFor(next[side]->sample());
                ::q[side]->give(1);
        }
        return 0;
}
```

Pseudo-Zufallszahl (exponential-verteilt) wird ermittelt

Fahrzeug-Instanz, repräsentiert durch ein Token, wird „erzeugt“ und in Bin-Objekt abgelegt

Hauptprogramm wird zwei Objekte generieren,
diese sorgen nach Aktivierung für zyklische (zeitversetzte) Erzeugung von Fahrzeugen auf der Festland- bzw. Inselseite

Hauptprogramm

```
main( int argc, const char* argv[]) {
        q[0] = new odemx::synchronization::Bin(sim, "mainland", 3);
        q[1] = new odemx::synchronization::Bin(sim, "island", 1);

        fload = new odemx::synchronization::Res(sim, "ferry load", 4, 4);

        next[0] = new odemx::random::NegativeExponential(sim, "mainland", 0.1);
        next[1] = new odemx::random::NegativeExponential(sim, "island", 0.1);
        crossing = new odemx::random::Normal(sim, "crossing", 7.5, 0.5);

        trips = new odemx::statistics::Count(sim, "trips");
        empties = new odemx::statistics::Count(sim, "empty trips");
        av_load = new odemx::statistics::Tally(sim, "av.load");
        minStayTime = 5.0;

        Arrival *a1 = new Arrival(0);
        a1->activateAt(6*60+20);
        Arrival *a2 = new Arrival(1);
        a2->activateAt(6*60+20);
        Ferry *f = new Ferry(fload);
        f->activateAt(7 * 60); //Start um 7.00 Uhr
        sim.run(); //Abbruch durch Ferry
}
```

zu benutzender Index von q: 0

Beginn der Fahrzeuggenerierung: 6:20 Uhr

Ferry-Klasse

```
class Ferry : public odemx::base::Process {
            odemx::synchronization::Res *myload;
    public: int main ();
            Ferry(odemx::synchronization::Res *r) : odemx::base::Process(sim, "Ferry"),
                                                    myload(r) {}
};
```

Hauptprogramm wird ein Objekt generieren, dieses sorgt für zyklische Aktionsfolge - Fahrzeugentladung, Beladung, Überfahrt

Hauptprogramm

```
main( int argc, const char* argv[]) {
        q[0] = new odemx::synchronization::Bin(sim, "mainland", 3);
        q[1] = new odemx::synchronization::Bin(sim, "island", 1);

        fload = new odemx::synchronization::Res(sim, "ferry load", 4, 4);

        next[0] = new odemx::random::NegativeExponential(sim, "mainland", 0.1);
        next[1] = new odemx::random::NegativeExponential(sim, "island", 0.1);
        crossing = new odemx::random::Normal(sim, "crossing", 7.5, 0.5);

        trips = new odemx::statistics::Count(sim, "trips");
        empties = new odemx::statistics::Count(sim, "empty trips");
        av_load = new odemx::statistics::Tally(sim, "av.load");
        minStayTime = 5.0;

        Arrival *a1 = new Arrival(0);
        a1->activateAt(6*60+20);
        Arrival *a2 = new Arrival(1);
        a2->activateAt(6*60+20);
        Ferry *f = new Ferry(fload);
        f->activateAt(7 * 60); //Start um 7.00 Uhr
        sim.run(); //Abbruch durch Ferry
}
```

Ferry-main

```
int Ferry::main() {
    MyTimer *timeout = new MyTimer (this);
    do {
        for (int side=0; side<=1; side++) {
            double entryTime = sim.getTime();
            // Entladung
                ...
            // Beladung
                ...
            // Mindestaufenthalt
                ...
            // Beladungsstatistiken
                ...
             // Fahrt
                ...
        }
        trips->update(1);  // Reise= Hin- und Rueckfahrt
     }
     while (sim.getTime() < 21*60 + 45); // nur tagsueber
    // zum Schluss noch einmal entladen
    int currentLoad = myload->getTokenLimit() - myload->getTokenNumber();
    holdFor(currentLoad*0.5);
    myload->release(currentLoad);
    sim.exitSimulation();
return 0; }
```

Ferry-main

```
int Ferry::main() {
    MyTimer *timeout =
    do {
        for (int side=0; s
            double er
            // Entladu
                ...
            // Beladu
                ...
            // Mindes
                ...
            // Beladu
                ...
             // Fahrt
                ...
        }
        trips->update(1
     }
     while (sim.getTime() < 21*60 + 45); // nur tagsueber
    // zum Schluss noch einmal entladen
    int currentLoad = myload->getTokenLimit() - myload->getTokenNumber();
    holdFor(currentLoad*0.5);
    myload->release(currentLoad);
    sim.exitSimulation();
return 0; }
```

```
for (int side=0; side<=1; side++) {
                double entryTime = sim.getTime();
                // Entladung
                int currentLoad = myload->getTokenLimit() - myload->getTokenNumber();
                holdFor(currentLoad*0.5);
                myload->release(currentLoad);
                // Beladung
                while ( myload->getTokenNumber()>0 && ::q[side]->getTokenNumber()>0 ) {

                        ::q[side]->take(1);
                        holdFor(0.5);
                        myload->acquire(1);
                }
                // Mindestaufenthalt
                ...
```

Ferry-main

```
int Ferry::main() {
    MyTimer *timeout = new MyTimer (this);
    do {
        for (int side=0; side<=1; side++) {
            double entryTime = sim.getTime();
            // Entladung
                ...
            // Beladung
                ...
            // Mindestau
                ...
            // Beladungss
                ...
             // Fahrt
                ...
        }
        trips->update(1);  //
     }
     while (sim.getTime() < 2
    // zum Schluss noch einr
    int currentLoad = myloac
    holdFor(currentLoad*0.5
    myload->release(currentLoad);
    sim.exitSimulation();
return 0; }
```

```
// Mindestaufenthalt
            double loadTime = sim.getTime() - entryTime;
            while (loadTime < minStayTime && myload->getTokenNumber()>0) {
                timeout->setTimeout (minStayTime - loadTime);
                timeout->activate();
                if (::q[side]->take(1) != 0) { // Auto in Wartezeit angekommen
                    timeout->interrupt(); // timer reset
                    holdFor(0.5);
                    myload->acquire(1);
                }
                loadTime = sim.getTime() - entryTime;
            }
```

Ferry-main

```
int Ferry::main() {
    MyTimer *timeout = new MyTimer (this);
    do {
        for (int side=0; side<=1; side++) {
            double entryTime = sim.getTime();
            // Entladung
                ...
            // Beladung
                ...
            // Mindestaufenthalt
                ...
            // Beladungsstatistiken
                ...
             // Fahrt
                ...
        }
        trips->update(1);  // Reise= Hin- und Rueckfahrt
     }
     while (sim.getTime() < 21*60 + 45); // nur tagsueber
    // zum Schluss noch einmal entladen
    int currentLoad = myload->getTokenLimit() - myload->getTokenNumber();
    holdFor(currentLoad*0.5);
    myload->release(currentLoad);
    sim.exitSimulation();
return 0; }
```

```
// Beladungsstatistiken
   currentLoad = myload->getTokenLimit() - myload->getTokenNumber();
   av_load->update(currentLoad);
   if (currentLoad == 0) empties->update(1);
// Fahrt
   holdFor(crossing->sample());
```

MyTimer-Klasse

```
class MyTimer : public odemx::base::Process {
        odemx::base::Process *interruptible;
        double howLong;
      public:
        MyTimer(odemx::base::Process *i):
            odemx::base::Process(sim, "Timer Interrupt"), interruptible(i), howLong(0) { }
        void setTimeout(double h) { howLong = h; }

        int main() {
            for(;;) {
                holdFor(howLong);
                if (!isInterrupted()) // timer reset
                interruptible->interrupt();
                sleep();
            }
            return 0;
        }
};
```