Kurs OMSI
im WiSe 2014/15

Objektorientierte Simulation mit ODEMx

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

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

6. ODEMx-Modul Synchronisation:

WaitQ, CondQ

- Konzept WaitQ

  Beispiele: Tankerflotte / Hafen / Raffinerie

- Konzept CondQ

  Beispiel: Hafen / Schlepper / Gezeiten

- Weitere Anwendungsbeispiele für WaitQ u. CondQ

- Zusammenfassung/einheitliche Betrachtung

ODEMx-Synchronisationselemente

- Memory → PortHead, PortTail, Timer, Cond
  Memory* m= wait (…);
- PortHead, PortTail, Port
  put(), Msg* m= get()
- Bin
  n= take(k), n= give (k)
- Res
  n= acquire(k), n= release(k)
- WaitQ
  Process* coopt(Selection sel=0),
  bool wait (Weight w=0)
  Process* avail(Selection)
  signal()
- CondQ
  wait(Condition), signal()

~globale Funktion

4 wait-Operationen
mit jeweils
unterschiedlicher Semantik

6. ODEMx-Modul Synchronisation:
WaitQ, CondQ

- Konzept WaitQ

  Beispiel: Tankerflotte / Hafen / Raffinerie

- Konzept CondQ

  Beispiel: Hafen / Schlepper / Gezeiten

- Weitere Anwendungsbeispiele für WaitQ u. CondQ
- Zusammenfassung/einheitliche Betrachtung

Begriffe (Wdh.)

- **Ereignis**

  allgemein:

  … ist ein Geschehen, das in einem gegebenen Kontext zu einem Zeitpunkt eine Bedeutung hat
  (es lässt sich räumlich und zeitlich lokalisieren).

  (Zeitpunkt , Objektkonfiguration, Aktionen)

  Ereignissimulation → Aktivitätssimulation

  - Auslösen neuer Ereignisse (Instanziierung von Ereignissen)
  - zeitverbrauchende zustandsverändernde **Aktivitäten**
    minimal beschrieben durch ein Start- und Ende-Ereignis

- **Ereignisklassen**

  … sind das Ergebnis einer Klassifikationsabstraktion von Ereignissen bei Parametrisierung der Ereigniszeit und der beteiligten Objekte (Objektreferenzen).

  Ereignisse sind Instanzen ihrer jeweils beschreibenden Ereignisklasse.

Begriffe

zwei Spielarten von Ereignissen

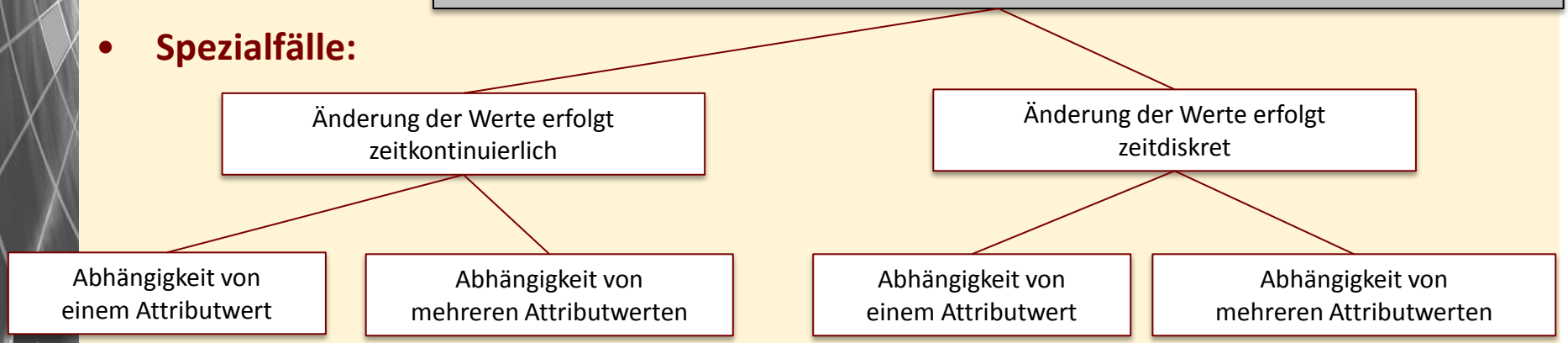
- **Zeitereignis:**

  bei der Instanziierung ist der Ereigniszeitpunkt bekannt

- **Zustandsereignis:**

  bei der Instanziierung ist der Ereigniszeitpunkt nicht bekannt, dieser ergibt sich vielmehr durch die Erfüllung einer Bedingung, die an Attributwerte der beteiligten Objekte geknüpft ist, die sich in Abhängigkeit der Modellzeit ändern.
  Der erste Zeitpunkt, zu dem diese Bedingung erfüllt ist oder der erste Zeitpunkt der Feststellung der Bedingungserfüllung ist dann der Ereigniszeitpunkt

- **Spezialfälle:**



Zustandsereignisse

Aktion: *beliebige Komplexität*
(1) Start, Stop oder Aktivierung, Unterbrechung eines oder mehrerer Prozesse
(2) elementare Aktion

- Ereignis mit Zustandsbedingung
  (erst mit Erfüllung der Bedingung soll Ereignis eintreten)

Annahme
- System besteht aus Prozessen
- Zustand ändert sich mit Zeitfortschritt durch Realisierung der Prozesse

Fragen
- wo werden Zustandsbedingungen vermerkt?
- wer überprüft wann die Zustandsbedingungen ?
- wer löst die Ereignisse aus?

ODEMx unterscheidet zwischen der Behandlung
- zeitkontinuierlicher und
- zeitdiskreter

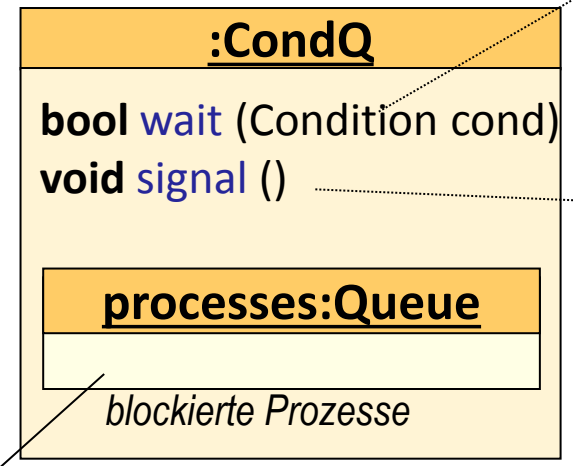
Zustandsgrößen

```
int Contiuous:: integrate
                (SimTime timeEvent, Condition stateEvent=0);
```

```
bool CondQ:: wait (Condition cond);
```

```
typedef bool (Process::*Condition)();
//definiert in Process
```

CondQ-Konzept



Aufrufer-Prozess wird blockiert,
- wartet auf Erfüllung der Bedingung cond

Aufrufer-Prozess
- reaktiviert **alle** blockierten Prozesse (Fortsetzung mit wait bei erneuter Auswertung der Nutzerbedingung)

ACHTUNG
sinnvollerweise sollten signal()-Rufer zumindest partiellen Einfluss auf die Zustands-bedingung der blockierten Prozesse haben

jeder hier vermerkte, blockierte Prozess kennt seine individuelle Fortsetzungsbedingung

Weitere Anforderungen an CondQ

1. über ein CondQ-Objekt sollen sich

   gleichzeitig / nacheinander

   beliebig viele Prozesse (einer oder unterschiedlicher Klassen) registrieren können, die auf individuelle Zustandsbedingungen zu warten haben.

   über die dynamische Vereinigung der Prozesse in einem CondQ-Objekt wird ein einheitlicher Zugang für ihre Reaktivierung (bei erneuter Berechnung der Fortsetzungsbedingung) geschaffen

2. die Warte-Aktivität soll für jeden blockierten Prozess, der in einem CondQ-Objektes erfasst ist, vorzeitig **unterbrechbar** sein,
   der Rückkehrcode informiert den Rufer von wait über die jeweilige Warte-Beendigung

Member-Funktionen von CondQ

```
CondQ (base::Simulation &sim, const data::Label &label, CondQObserver *obs=0)
                    // Construction for user-defined Simulation.
~CondQ ()
                    // Destruction.

getWaitingProcesses () const
                    // Get a list of blocked process objects.


bool wait (base::Condition cond)
                    // Wait for cond.
signal ()
                    // Trigger condition check.
```

```
bool CondQ::wait (Condition cond) {

      processes. inSort (getCurrentProcess());

      // statistics
      SimTime t=env->getTime();

      while (!(getCurrentProcess()->*cond)() ) {
            getCurrentProcess()->sleep();

            if (getCurrentProcess()->isInterrupted()) {
                  processes. remove (getCurrentProcess());

                  return false;
            }
      }

      processes. remove(getCurrentProcess());

      // statistics
      t = env->getTime() - t;
      sumWaitTime += t;
      if (t==0) zeros++;

      users++;
            …
      return true;
}
```

```
void CondQ::signal() {
    // trace
    getTrace()->mark( this, markSignal, getCurrentProcess());

    // observer
    …

    // statistics
    signals++;

    if (processes.isEmpty())
        return;

    // test conditions
    awakeAll(&processes);
}
```

6. ODEMx-Modul Synchronisation:

WaitQ, CondQ

- Konzept WaitQ

  Beispiel: Tankerflotte / Hafen / Raffinerie

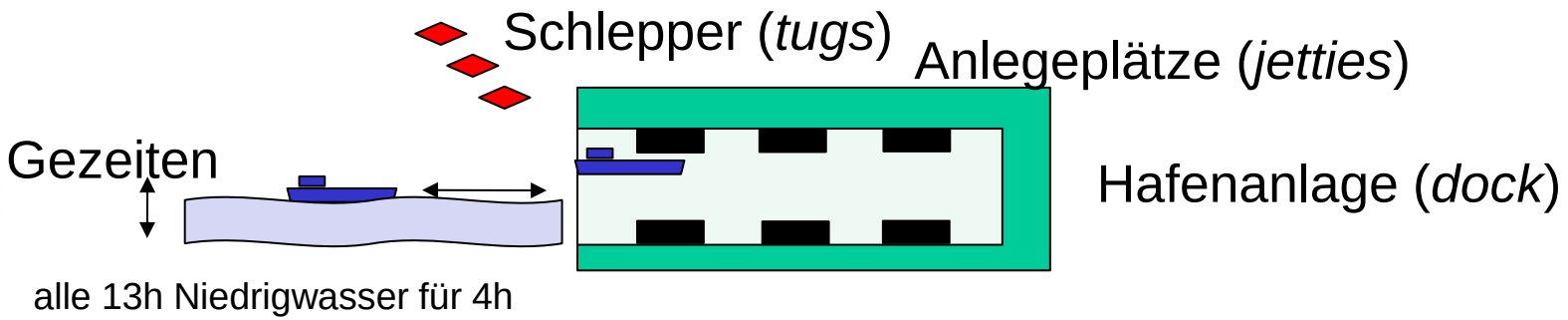
- Konzept CondQ

  Beispiel: Hafen / Schlepper / Gezeiten

- Weitere Anwendungsbeispiele für WaitQ u. CondQ

- Zusammenfassung/einheitliche Betrachtung

Wortmodell: Hafenabfertigung mit Gezeiten



Einlauf von Schiffen
Bedingungen/Aktionen:
- freier Anlegeplatz
- zwei freie Schlepper
- Wasserstand: hoch
- Anlegemanöver= 2h
- Schlepperfreigabe

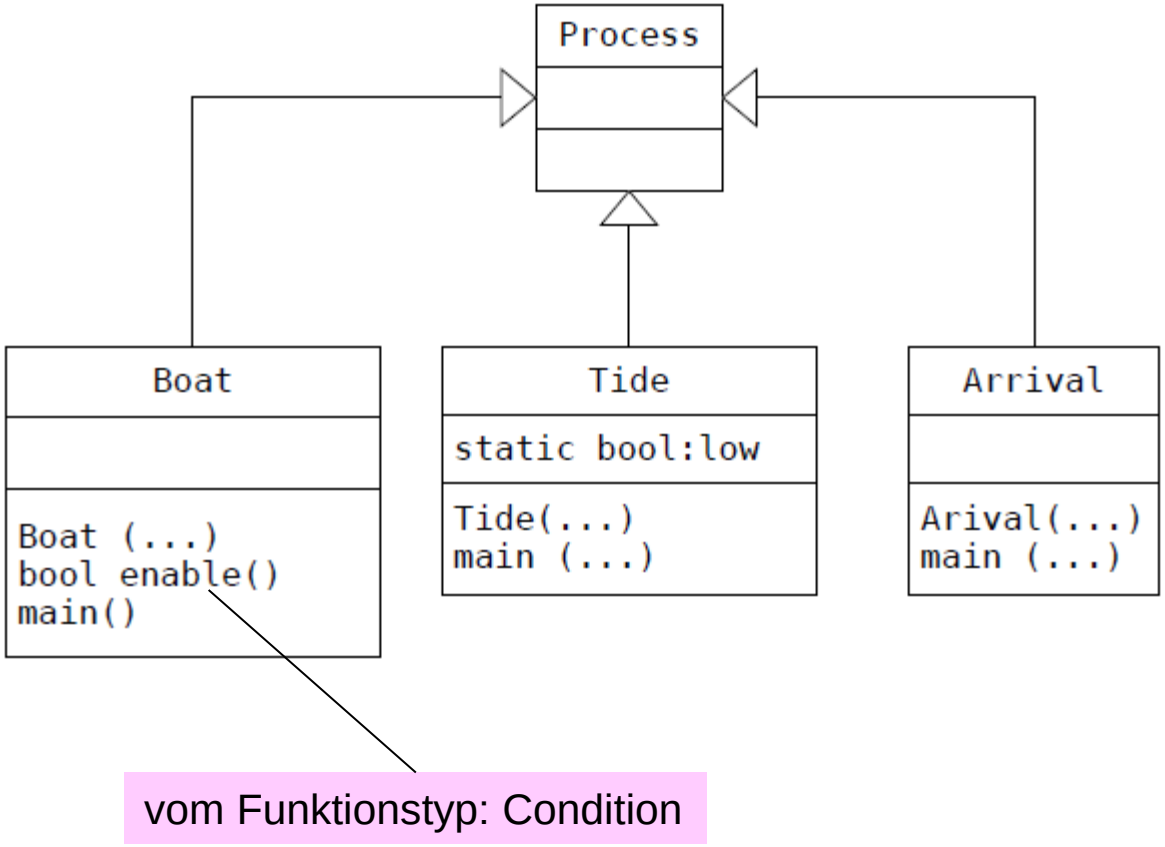
Entladung von Schiffen
Bedingungen/Aktionen:
- stochastische Entladungszeit

Auslauf von Schiffen
Bedingungen/Aktionen:
- ein freier Schlepper
- Auslaufmanöver: 2h
- Schlepperfreigabe
- Platzfreigabe

Ziel: simulative Nachbildung des Betriebsaublauf bei Bestimmung der Auslastung eingesetzter Ressourcen

Systemstruktur (semiformal)

Strukturkomponenten (Typbeschreibung, Objektkonfiguration)



Verhalten von Boat

Wunsch:
explizite Angabe des Ausdrucks

```
Simulation* sim = getDefaultSimulation();
Res   *tugs;                                    // Schlepper
Res   *jetties;                                 // Anlegeplaetze
CondQ *dockq;
ContinuousDist *next, *discharge;

class Boat : public Process {
public:
        int main ();
        Boat() : Process(sim, "boat"){}
        bool enable();
};
```

```
int Boat::main() {
// im Dock anlegen
        jetties->acquire(1);
        dockq->wait((Condition)&Boat::enable);
        tugs->acquire(2);
        holdFor(2.0);
        tugs->release(2);
        dockq->signal();
// loeschen der Ladung
        holdFor(discharge->sample());
// ablegen
        tugs->acquire(1);
        holdFor(2.0);
        tugs->release(1);
        jetties->release(1);
        dockq->signal();

        return 0;
}
```

```
bool Boat::enable() {
   return (tugs->getTokenNumber() >=2) && !Tide::low;
}
```

Lösung in OMSI-2:
Lamda-Kalkül in C++

Verhalten von Tide

```
class Tide : public Process {
public:
        static bool low;
        Tide(): Process(sim, "tide") {};
        int main ();
};

bool Tide::low = false;
```

```
int Tide::main() {
        for (;;) {
                // low
                low = true;
                holdFor(4.0);
                // high
                low = false;
                dockq->signal();
                holdFor(9.0);
        }
        return 0;
}
```

```
int main( int argc, const char* argv[]) {
        next = new Negexp(sim, "next boat", 0.1);
        discharge = new Normal(sim, "discharge", 14.0, 3.0);
        tugs = new Res(sim, "tugs", 3, 3);
        jetties = new Res(sim, "jetties", 2, 2);
        dockq = new CondQ(sim, "dockq");

        report.addProducer(next);
        report.addProducer(discharge);
        report.addProducer(tugs);
        report.addProducer(jetties);
        report.addProducer(dockq);
        sim->startTrace();
        a->activate();
        t->activateIn(1.0);

        sim->runUntil(50.0);
        sim->stopTrace();
        sim->runUntil (28.0*24.0);
        report.generateReport();

        delete a;
        delete t;
        delete next;
        delete discharge;
        delete tugs;
        delete jetties;
        delete dockq;

        return 0;
}
```

a= **new** Arrival();
t= **new** Tide();

Zeit

Simulation: DefaultSimulation **SimTime: 0**

HtmlTrace **ODEMx Version: 2.2**

Filter: "FilterMode: PASS ALL | MarkTypeNames: changeExTime, changeState, changeTokenNumber, create, current process, execute, execute process, init, run until, sleep, time"

Time	Sender	Event	Details		Comment
0	arrival	activate	Partner	DefaultSimulation	
		change state	old Value	CREATED	
			new Value	RUNNABLE	
		change execution time	old Value	0	
			new Value	0	
	tide	activate in	current	DefaultSimulation	
			relative time	1	
		change state	old Value	CREATED	
			new Value	RUNNABLE	
		change execution time	old Value	0	
			new Value	1	
	arrival	change state	old Value	RUNNABLE	
			new Value	CURRENT	
	boat	hold	Partner	arrival	
		change state	old Value	CREATED	
			new Value	RUNNABLE	
		change execution time	old Value	0	
			new Value	0	
	arrival	change state	old Value	CURRENT	
			new Value	CURRENT	
		hold for	current	arrival	
			relative time	26.5738	
		change state	old Value	CURRENT	
			new Value	RUNNABLE	
		change execution time	old Value	0	
			new Value	26.5738	
	boat	change state	old Value	RUNNABLE	
			new Value	CURRENT	
	jetties	change token number	old Value	2	
			new Value	1	
		acquire succeed	current	boat	
			token number	1	
	dockq	wait	Partner	boat	
		continue	Partner	boat	
	tugs	change token number	old Value	3	
			new Value	1	
		acquire succeed	current	boat	
			token number	2	
	boat	hold for	current	boat	
			relative time	2	
		change state	old Value	CURRENT	
			new Value	RUNNABLE	
		change execution time	old Value	0	
			new Value	2	
1	tide	change state	old Value	RUNNABLE	
			new Value	CURRENT	
		hold for	current	tide	
			relative time	4	
		change state	old Value	CURRENT	
			new Value	RUNNABLE	
		change execution time	old Value	1	
			new Value	5	
2	boat	change state	old Value	RUNNABLE	
			new Value	CURRENT	
	tugs	change token number	old Value	1	
			new Value	3	
		release succeed	current	boat	
			token number	2	
	dockq	signal	Partner	boat	
	boat	hold for	current	boat	
			relative time	16.3855	
		change state	old Value	CURRENT	
			new Value	RUNNABLE	
		change execution time	old Value	2	
			new Value	18.3855	
5	tide	change state	old Value	RUNNABLE	
			new Value	CURRENT	
	dockq	signal	Partner	tide	
	tide	hold for	current	tide	
			relative time	9	
		change state	old Value	CURRENT	
			new Value	RUNNABLE	
		change execution time	old Value	5	
			new Value	14	
Time	Sender	Event	Details		Comment

Spezielle Filtereinstellung:
Welche Infos sollen erfasst werden

Simulation: DefaultSimulation SimTime: 670

HtmlReport ODEMx Version: 2.2

Random Number Generators							
Name	Reset at	Type	Uses	Seed	Parameter 1	Parameter 2	Parameter 3
next boat		Negexp	64	33427485	0.1	0	0
discharge		Normal	58	22276755	14	3	0

Queue Statistics					
Name	Reset at	Min queue length	Max queue length	Now queue length	Avg queue length
tugs queue		0	1	0	0
jetties queue		0	7	6	0.634897
dockq queue		0	2	0	0.0770625

Res Statistics											
Name	Reset at	Queue	Acquires	Releases	Init token number	Limit token number	Min token number	Max token number	Now token number	Avg token number	Avg waiting time
tugs		tugs queue	114	114	3	3	0	3	3	2.48657	0
jetties		jetties queue	58	56	2	2	0	2	0	0.373191	6.34256

CondQ Statistics						
Name	Reset at	Queue	Users	Signals	Zero wait users	Avg waiting time
dockq		dockq queue	58	166	36	0.890205

6. ODEMx-Modul Synchronisation:

WaitQ, CondQ

- Konzept WaitQ

  Beispiel: Tankerflotte / Hafen / Raffinerie

- Konzept CondQ

  Beispiel: Hafen / Schlepper / Gezeiten

- Weitere Anwendungsbeispiele für WaitQ u. CondQ

  Beispiel: Bagger-Zuteilung

- Zusammenfassung/einheitliche Betrachtung

Typisches Problem

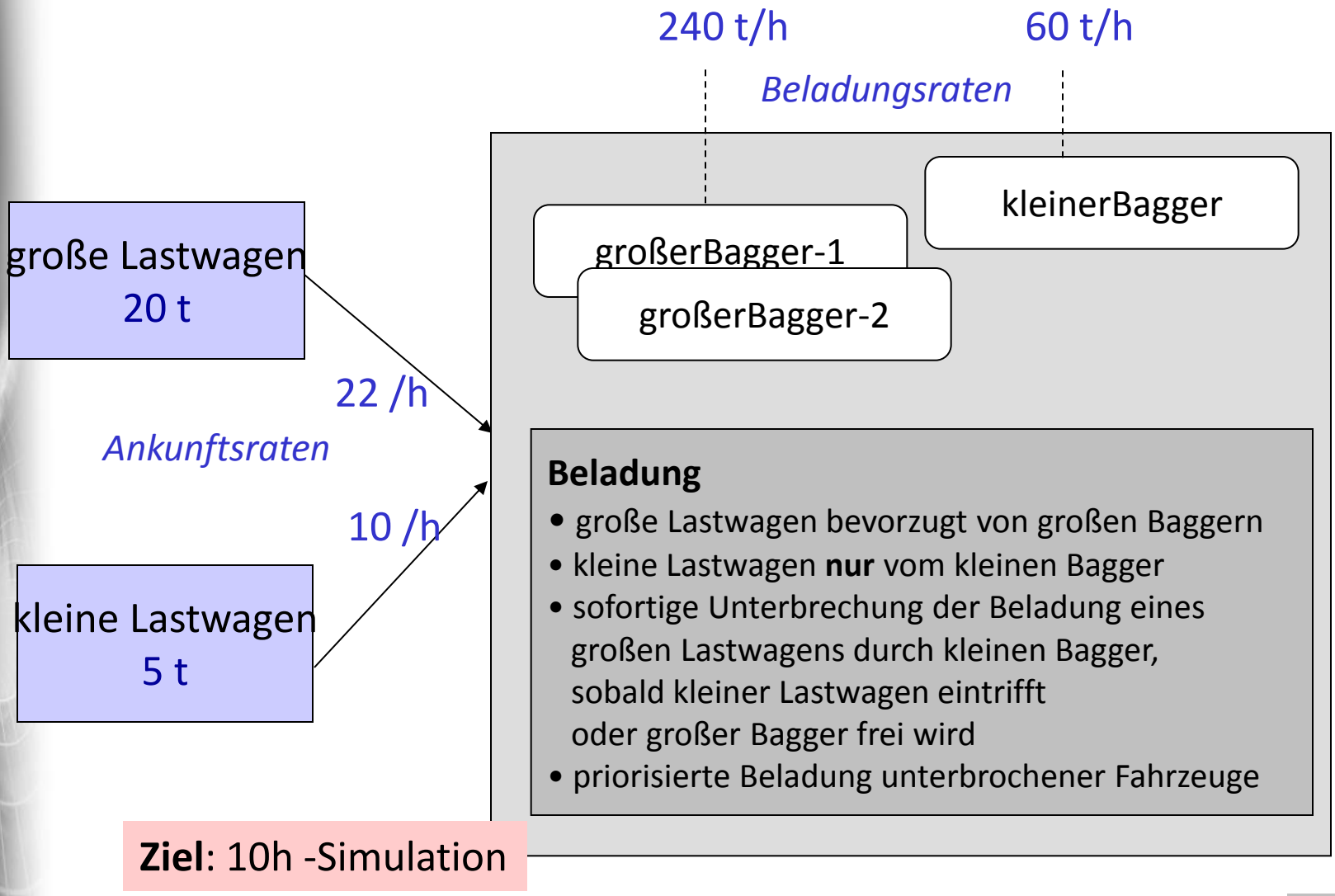
- Zuweisung einer Ressource aus einem Spektrum unterschiedlicher **Ressourcenklassen**
  für die Durchführung spezifischer Arbeitsgänge

- dynamischer Austausch der eingesetzten Ressourcen
  (bei gegebener Verfügbarkeit)
  um Effizienz des Arbeitsganges zu erhöhen

- Unterbrechung von Ressourcennutzungen

- Unterbrechung des Wartevorgangs auf
  eine Ressource (Klasse A)
  bei Fortsetzung mit Reservierung
  einer alternativen Ressource (Klasse B)

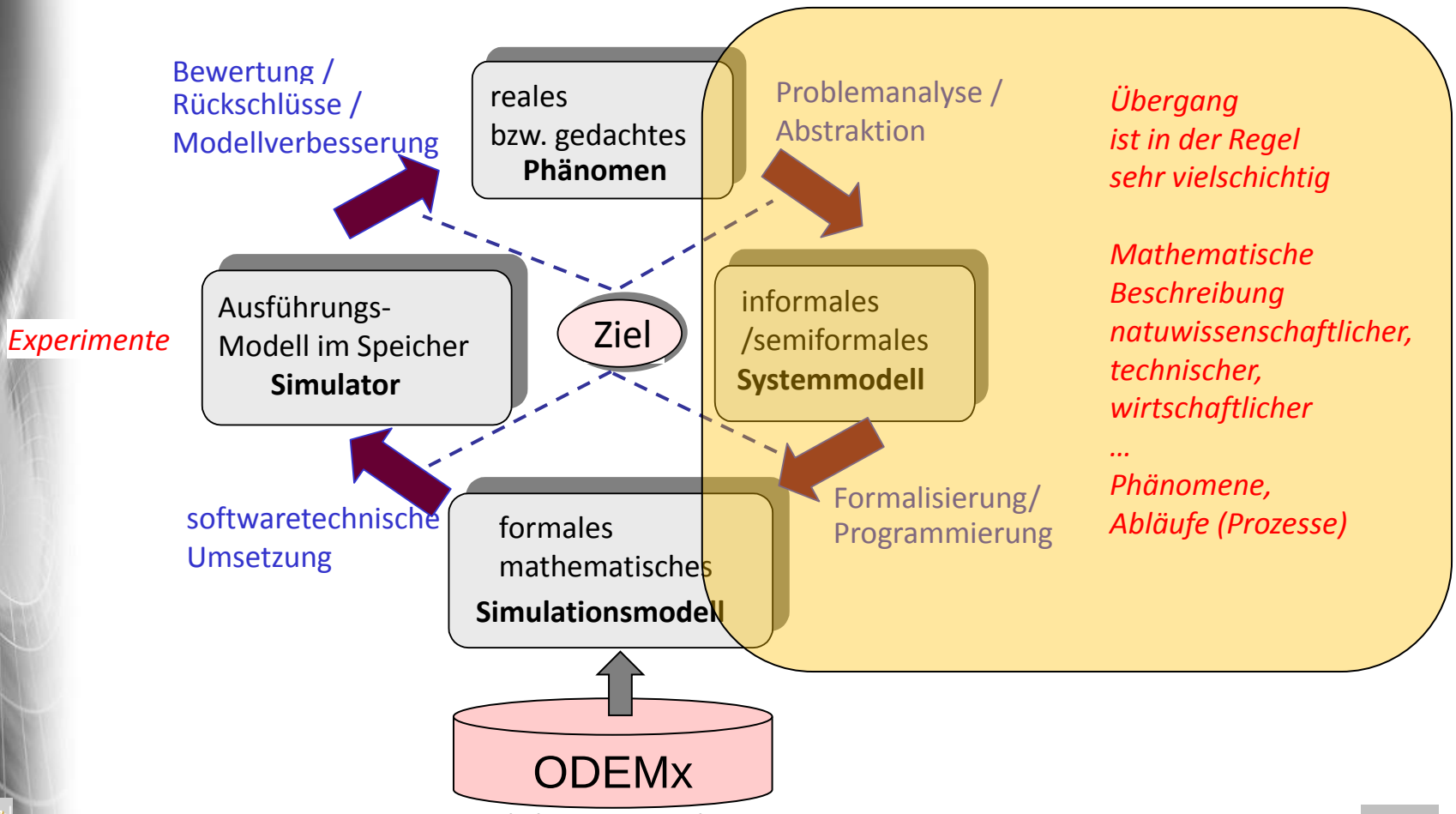
Beispiel: Transportfahrzeuge – Bagger – Beladung



Ziel: 10h -Simulation

Vorgehensweise bei der Systemsimulation

Experimentieren mit ausführbaren Modellen auf dem Computer
- anstatt mit Originalen -



Lösungsvarianten

a) ausschließlich mit Waitq

Unterbrechung der Wartevorgänge des kleinen Baggers an einem WaitQ-Objekt als Master

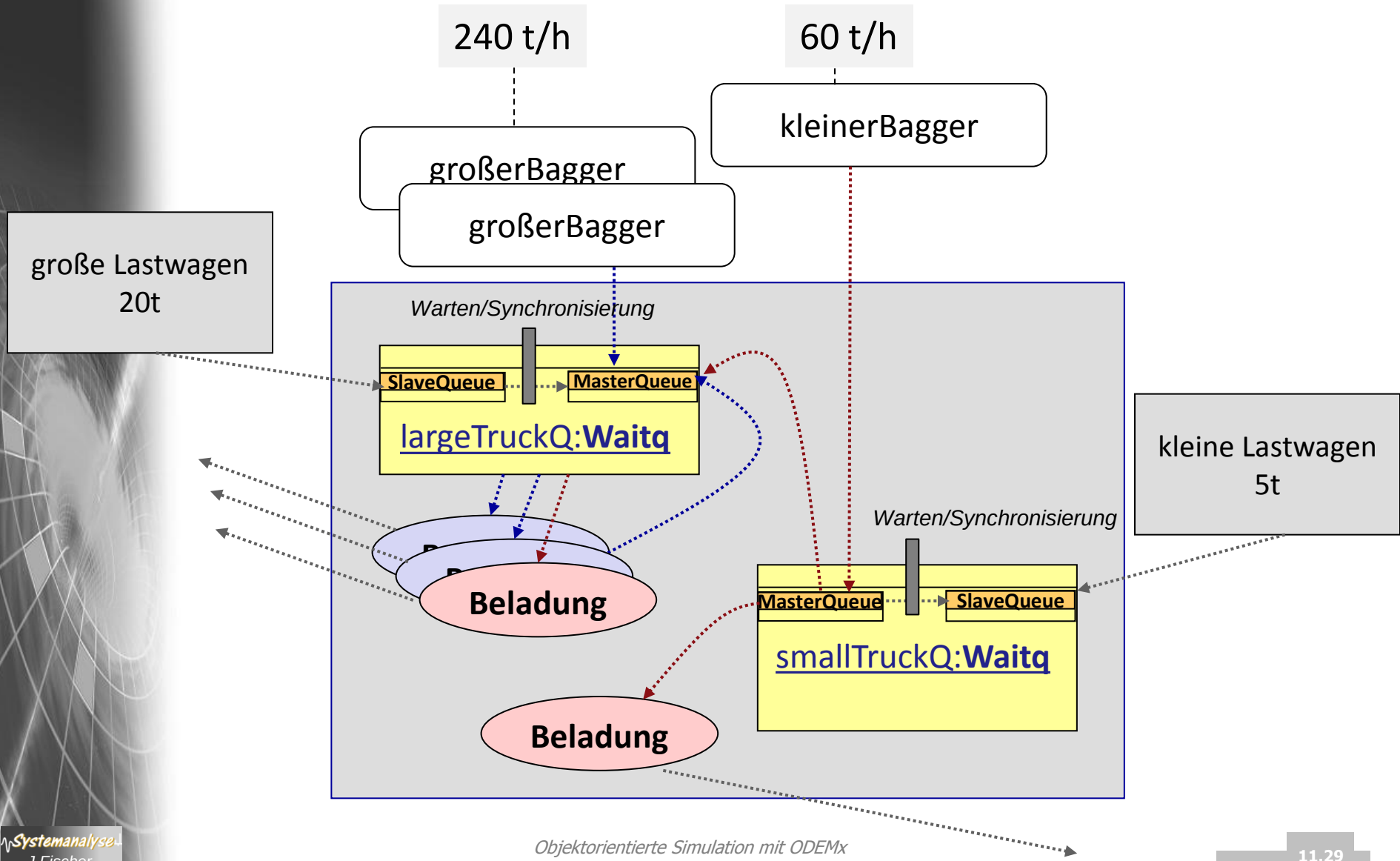
Fortsetzung als Master in einem anderen WaitQ-Objekt

b) mit WaitQ- und CondQ-Einsatz

Warten des kleinen Baggers im CondQ-Objekt bis kleines oder großes Fahrzeug eintrifft

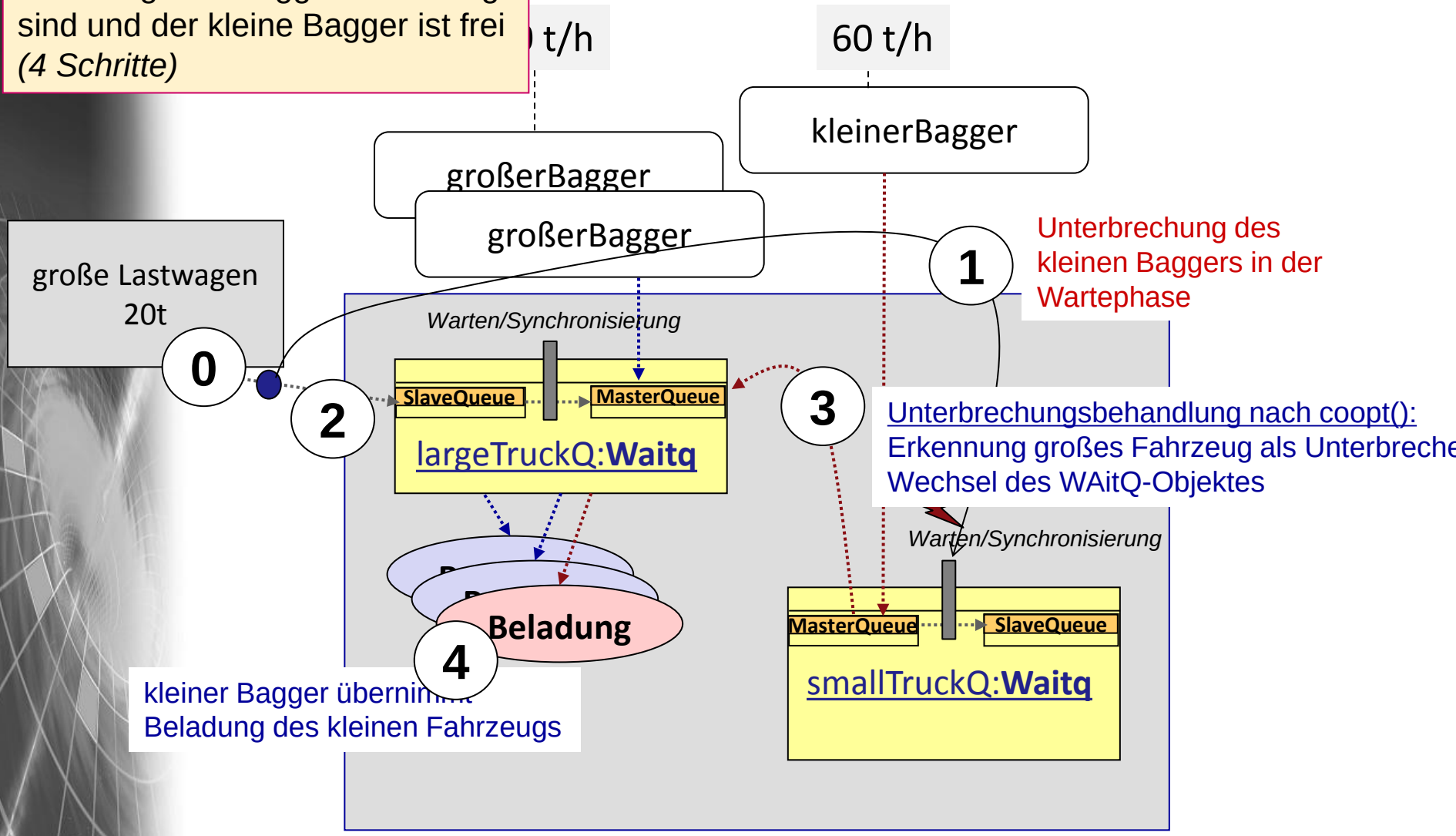
Fortsetzung als Master in einem der WaitQ-Objekte

Variante a): Master-Slave-Rollen bei der Beladung



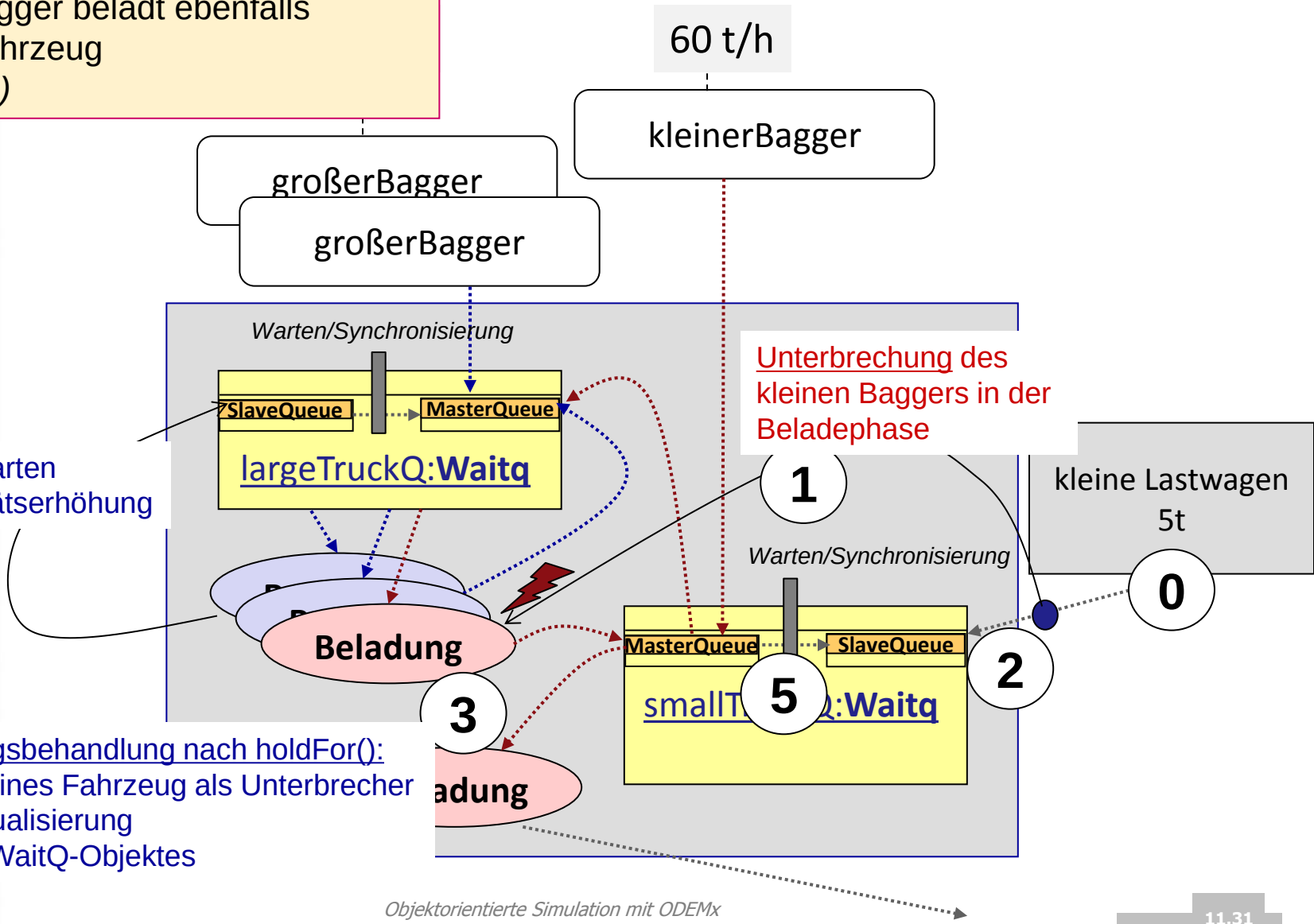
1. Fall:
ein großes Fahrzeug trifft ein, wobei 2 große Bagger beschäftigt sind und der kleine Bagger ist frei *(4 Schritte)*

Master-Slave-Rollen bei der Beladung



2. Fall:
ein kleines Fahrzeug trifft ein,
beide großen Bagger sind beschäftigt,
kleiner Bagger belädt ebenfalls
großes Fahrzeug
(5 Schritte)

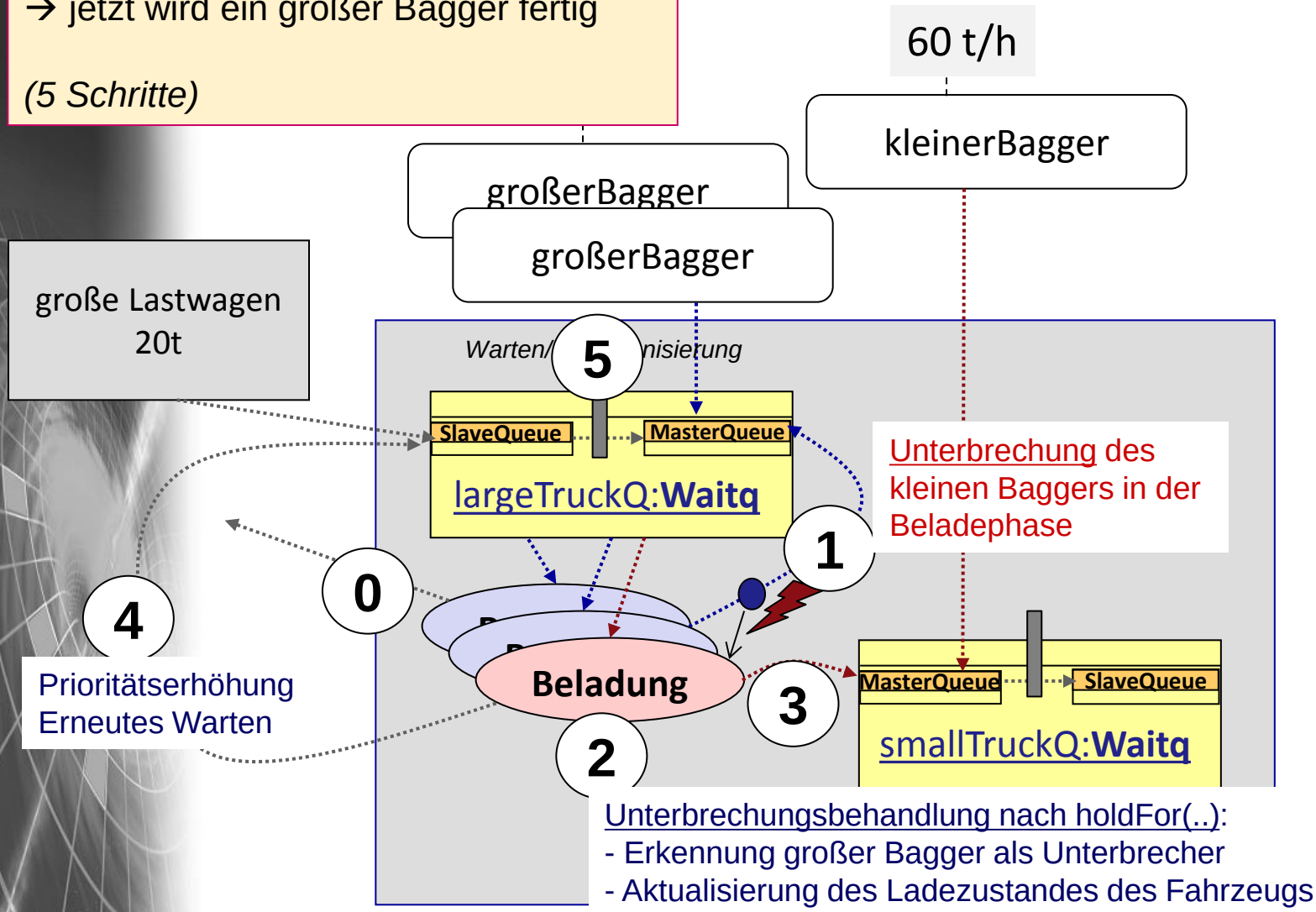
…er-Slave-Rollen bei der Beladung



3. Fall:
beide großen Bagger sind beschäftigt,
kleiner Bagger entlädt großes Fahrzeug
→ jetzt wird ein großer Bagger fertig

(5 Schritte)

...er-Slave-Rollen bei der Beladung



Unterbrechungsbehandlung nach holdFor(..):
- Erkennung großer Bagger als Unterbrecher
- Aktualisierung des Ladezustandes des Fahrzeugs
- Freigabe des Fahrzeugs

Grobgranulare Zustände des kleinen Baggers

- FREE
  wartet als Master in smallTruckQ auf kleine Fahrzeuge

- SMALL
  belädt kleine Fahrzeuge

- LARGE
  belädt große Fahrzeuge

➔Abfrage mit workStatus()

Synchronisation: LargeDigger – LargeTruck

LargeDigger-Objekte als Master

```
int LargeDigger::main() {
      Truck *myTruck;
      for (;;) {
        if (smallDigger->workStatus() == LARGE) {
            // 3.Fall
            // Unterbrechung des kleinen Baggers,
            //der ein grosses Fahrzeug bedient
            smallDigger->interrupt();
        }
      //Warten auf grosses Fahrzeug
        myTruck=
          dynamic_cast<Truck*>(largeTruckQ->coopt());

       //Beladung
        holdFor ((myTruck->maxload-
                  myTruck->load) / loadRate);
        myTruck->load = myTruck->maxload;
        // Freigabe des Fahrzeugs
        myTruck->holdFor();
    }
   return 0;
}
```

LargeTruck-Objekte als Slave

```
int LargeTruck::main() {
      while (load <maxload) {
        // 1.Fall
        if (smallDigger->workStatus() == FREE &&
           (largeTruckQ->getWaitingMasters()).empty()) {
                //frei u. alle grossen Bagger belegt
                smallDigger->interrupt();
        }
         largeTruckQ->wait();

        // Warten auf beliebigen freien Bagger
       }
       return 0;
}
```

Synchronisation: SmallDigger – LargeTruck

SmallDigger-Objekte als Master

```
int SmallDigger::main() {
        double loadStart= 0.0; // Beladungsbeginn
        for (;;) {
          workSt= FREE;
          if (smallTruckQ->avail()) {
                    // unterbrechungsfreie Beladung
                    // eines kleinen Fahrzeugs
                    … workSt= SMALL;
                    … = smallTruckQ->coopt();
           }
           else
           if (largeTruckQ->avail() &&
               largeTruckQ->getWaitingMasters().empty() ) {
                    // unterbrechbare Beladung eines
                    // grossen Fahrzeugs
                    … workSt= LARGE; //2.Fall, 3.Fall
                     …
           }
           else {
                    // unterbrechbares Warten auf
                    // kleines Fahrzeug
                    … //1.Fall
                    … = smallTruckQ->coopt();

                    …
           }
        }
        return 0;
}
```

LargeTruck-Objekte als Slave

```
int LargeTruck::main() {
    while (load < maxload) {
          // 1.Fall
          if (smallDigger->workStatus() == FREE &&
             largeTruckQ->getWaitingMasters().empty() {
                              //frei u. alle grossen Bagger belegt
                    smallDigger->interrupt();
           }
           largeTruckQ->wait();
           // Warten auf beliebigen freien Bagger
     }
     return 0;
}
```

Synchronisation: SmallDigger – LargeTruck

SmallDigger-Objekte als Master

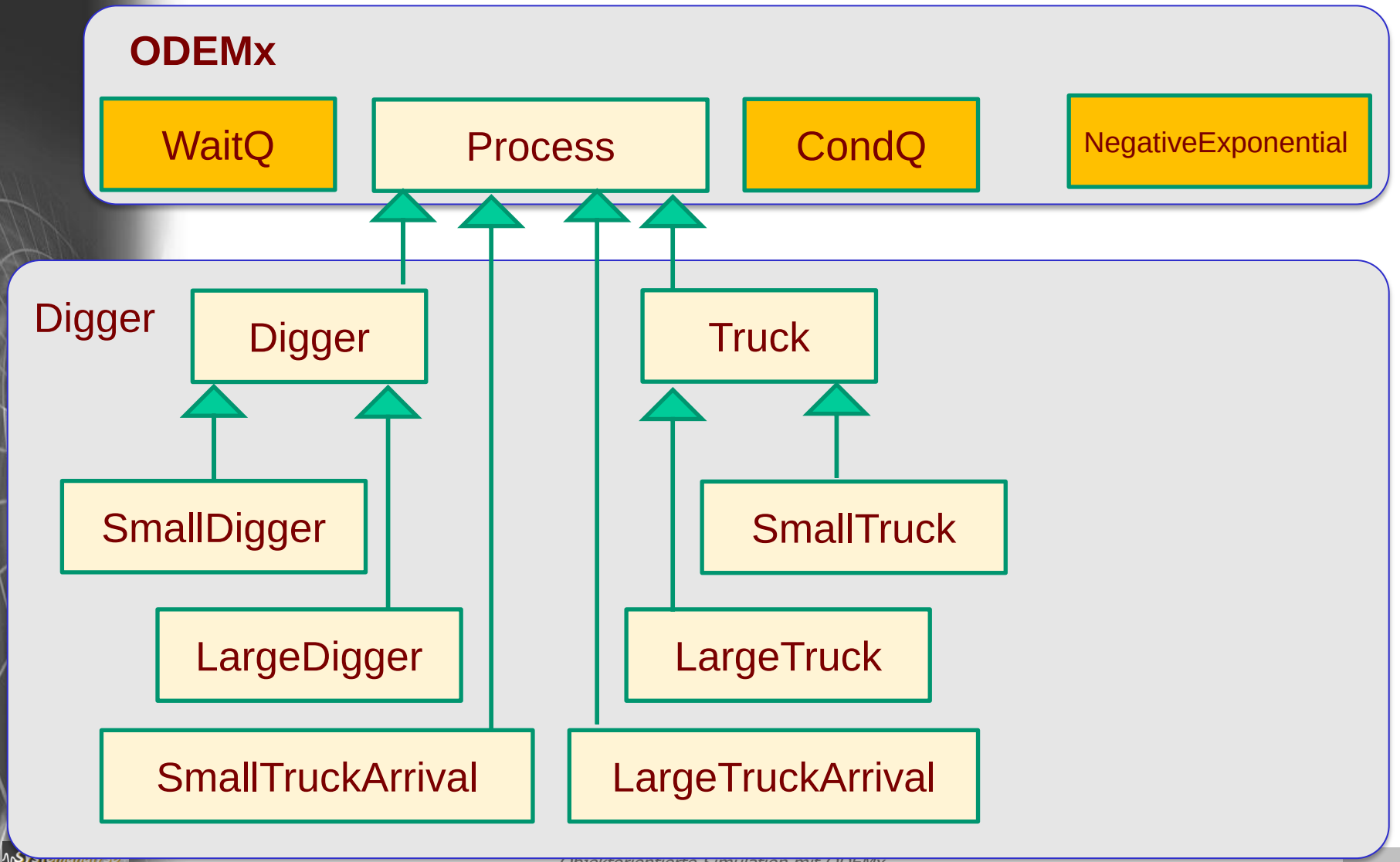
```
int SmallDigger::main() {
        double loadStart= 0; // Beladungsbeginn
        for (;;) {
          workSt= FREE;
          if (smallTruckQ->avail()) {
                    // unterbrechungsfreie Beladung
                    // eines kleinen Fahrzeugs
                    … workSt= SMALL;
          }
          else
          if (largeTruckQ->avail()) {
                    // unterbrechbare Entladung eines
                    // grossen Fahrzeugs
                    … workSt= LARGE; //2.Fall, 3.Fall
           }
           else {
                    // unterbrechbares Warten auf
                    // kleines Fahrzeug
                    … //1.Fall
                     … = smallTruckQ->coopt();

                    …
          }
        }
        return 0;
}
```

```
workSt= LARGE;

loadStart= time();
myTruck= dynamic_cast<Truck*> (largeTruckQ->coopt());
holdFor (t->load/rate);

// mögliche Unterbrechung der Beladung
// d.h. Aktivierierung erfolgt vor Beladungsende
// durch LargeDigger oder SmallTruck

if (! interrupted() )
        myTruck->load= myTruck->maxload;
else {
        //Unterbrechungsbehandlung
        myTruck->load= (time() - loadStart) * rate;
        //myTruck->maxload - myTruck->load
        //ist die verbleibende Ladekapazität
        myTruck->setWaitPriority(1);
        resetInterrupt();
}
```

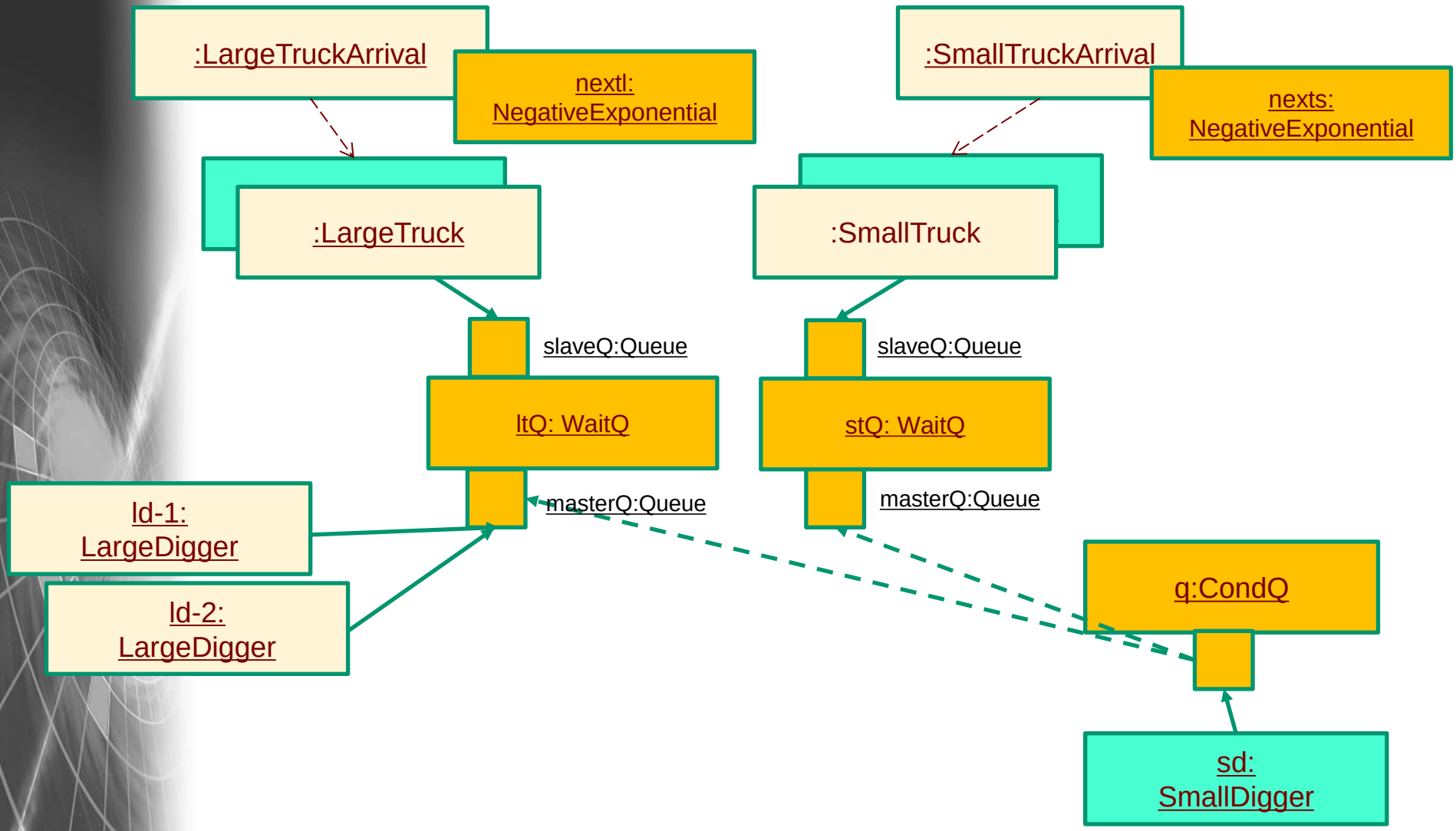
Process: Scheduling-Operationen

void activate(); **void** activateIn (SimTime t); **void** activateAt (SimTime t);	***LIFO***
void activateBefore (Process* p); **void** activateAfter (Process* p);	***davor/ danach***
void hold(); **void** holdFor (SimTime t); **void** holdUntil (SimTime t);	***FIFO***
void sleep(); **virtual void** interrupt(); **void** cancel();	***Prozessunterbrechungen, Abbruch***
bool isInterrupted() Sched* getInterrupter() **void** resetInterrupt()	***Unterbrechungszustand***
virtual int main() = 0;	***Process-Verhaltensfunktion (Lebenslauf)***
bool hasReturned() **const**; **int** getReturnValue() **const**;	***… bei Terminierung mittels return***

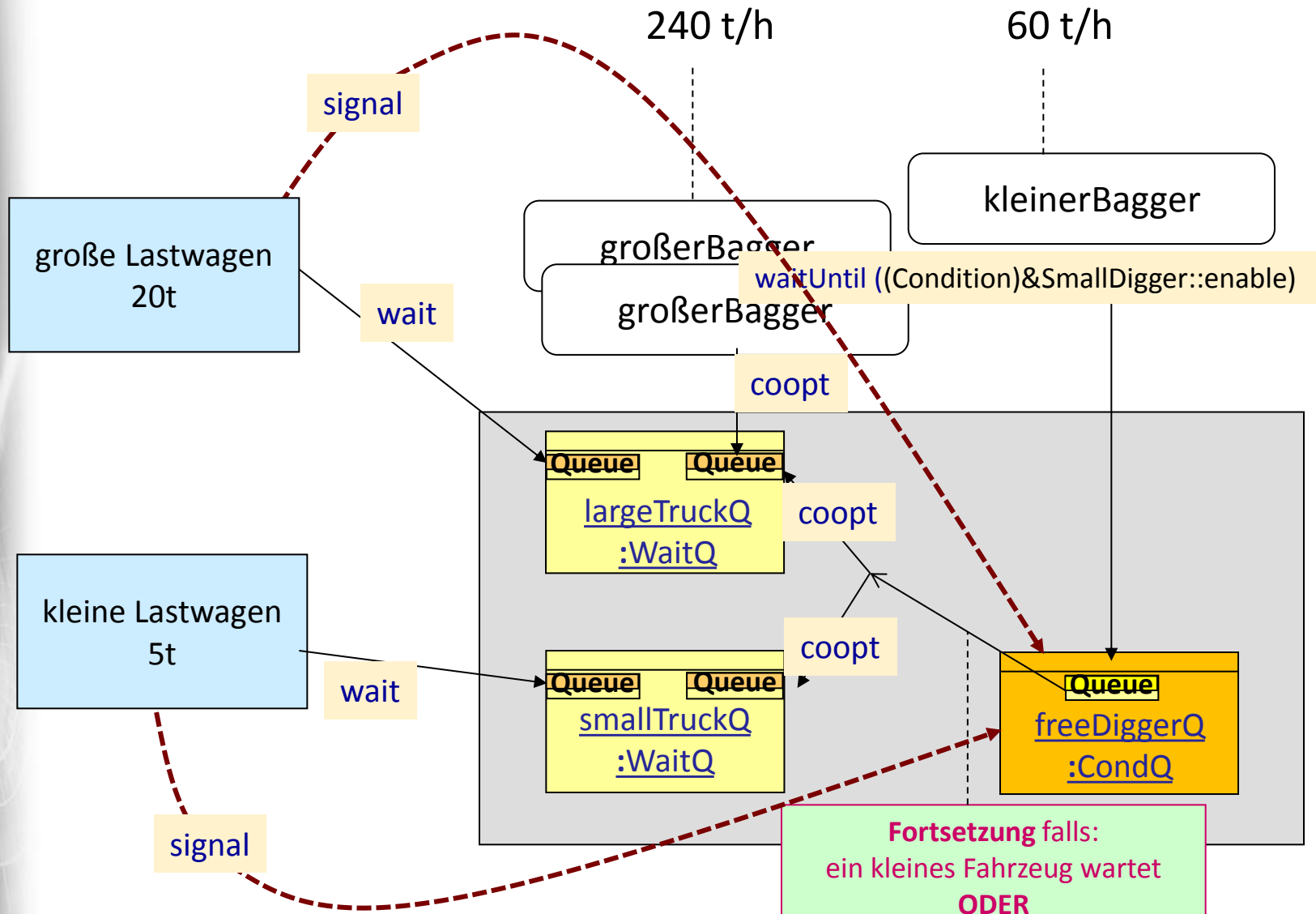
Variante b)


ODEMx
WaitQ
Process
CondQ
NegativeExponential
Digger
Digger
Truck
SmallDigger
SmallTruck
LargeDigger
LargeTruck
SmallTruckArrival
LargeTruckArrival

Systemkonfiguration



Ablauf


240 t/h
60 t/h
signal
große Lastwagen
20t
wait
großerBagger
großerBagger
kleinerBagger
waitUntil ((Condition)&SmallDigger::enable)
coopt
Queue
Queue
largeTruckQ
:WaitQ
coopt
kleine Lastwagen
5t
wait
coopt
Queue
Queue
smallTruckQ
:WaitQ
Queue
freeDiggerQ
:CondQ
signal
Fortsetzung falls:
ein kleines Fahrzeug wartet
ODER
ein großes Fahrzeug wartet und
kein großer Bagger ist frei

Globale Größen und main

```
#include <odemx/odemx.h>
using namespace odemx;
using namespace odemx::data;
using namespace odemx::base;
using namespace odemx::random;
using namespace odemx::synchronization;

#include <iostream>
using namespace std;

Condq   *q;
Waitq    *ltq, *stq;
ContinuousDist  *nextl, *nexts;

class Sdigger *s;  //global für Zustandsabfrage
```

```
int main () {
    Simulation& sim = getDefaultSimulation ();
    LtruckGenerator   *lt;
    StruckGenerator   *st;

 // Statistikpuffer und Loging- Trace- Preparation
 ...
 // Initialisierung
   nextl = new NegativeExponential(sim, "next large", 1/22.0/60.0);
   nexts = new NegativeExponential(sim, "next small", 1/10.0/60.0);
   q       = new Condq (sim, "sq");
   stq     = new Waitq (sim, "s_Truck_q");
   ltq     = new Waitq (sim, "l_Truck_q");

   s       = new Sdigger (sim);
   s->activate ();
   new Ldigger (sim)->activate ();
   new Ldigger (sim)->activate ();
   new LtruckGenerator (sim)->activate ();
   new StruckGenerator (sim)->activate ();

   sim.runUntil (8*60.0);  // 8 h
 // Report- Trace-Handling
 ...
   return 0;
}
```

Digger, LargeDigger

```
class Digger : public Process {
public:
        Truck    *t;    // zugeordneter Lastwagen
        double rate; // Beladerate in min
        Digger(Simulation& sim, const Label &name) :
                Process(sim, name),  t(0) {}
};


class Ldigger : public Digger {
public:
      Ldigger(Simulation& sim) :
              Digger(sim, "Ldigger"), rate(4.0) {}
      virtual int main();
};
```

```
int Ldigger::main() {
     for (;;) {
          // Warten auf grosses Fahrzeug
          t = dynamic_cast<Truck*>(ltq->coopt());
          // Beladung
          holdFor  (t->capacity/rate);
          t->load = t->capacity;
          // Freigabe des Fahrzeugs
          t->activate ();
          t = 0;
          if (s->t) {
               if (ltq->getWaitingSlaves ().size () == 0 &&   // kein großes Fahrz. wartet
                   dynamic_cast <Ltruck*> (s->t)) {           // kleiner Bagger belädt
                                                              // großes Fahrzeug
                    // Unterbrechung des kleinen Baggers
                    s->interrupt ();
               }
          } // if
     } // for
     return 0;
}
```

SmallDigger

```
class Digger : public Process {
public:
        Truck    *t;    // zugeordneter Lastwagen
        double rate; // Beladerate
        Digger(Simulation& sim, const Label &name) :
                Process(sim, name),  t(0) {}
};

class Sdigger : public Digger {
public:
    SimTime started;      // Beladungsbeginn
    Sdigger(Simulation& sim) : Digger(sim, "Sdigger"){}

    virtual int main();

    bool cond () {
       // Aktivierungsbedingung fuer kleinen Bagger:
       // ein kleines Fahrzeug wartet und alle grossen
       // Bagger sind besetzt
       return
       ( (stq->getWaitingSlaves ().size () > 0) ||
         (ltq->getWaitingMasters ().size () == 0) );
    }
};
```

```
int Sdigger::main() {
      rate = 1.0; // t pro min
      for (;;) {
            // Warten auf Aktivierungsbedingung: Sdigger::cond()==TRUE
            q->wait((Condition) &Sdigger::cond);
            if (! stq->getWaitingSlaves ().empty ()) {
                  // kleines Fahrzeug zur Beladung bereit
                  t = dynamic_cast<Truck*>(stq->coopt());
                  holdFor (t->capacity/rate);
            }
            else {
                  // grosses Fahrzeug zur Beladung bereit
                  started = getSimulation ().getTime ();
                  t = dynamic_cast<Truck*>(ltq->coopt());
                  holdFor  (t->capacity/rate);
                  if (!isInterrupted ()) t->load = t->capacity;
                  else {
                        // Unterbrechungsbehandlung
                        // beide Ursachen werden gleich behandelt
                        t->load += (getSimulation ().getTime () - started) * rate;
                        t->setPriority(1);
                        resetInterrupt ();
                  }
            }
            // Freigabe des (teilweise) beladenen Fahrzeugs
            t->activate ();
            t = 0;
      } // for
      return 0;
}
```

Truck, SmallTruck

```
class Truck : public Process {
public:
    double load;       // Ladung
    double capacity;  // Fassungsvermoegen

    Truck (Simulation &sim, const Label &name) :
            Process(sim, name){}
};

class Struck : public Truck {
public:
      Struck (Simulation& sim) : Truck(sim, "S"){}
      virtual int main ();
};
```

```
int Struck::main() {
        capacity = 5.0;
        load= 0.0;
        if (s->getProcessState () == RUNNABLE && s->t) {
                if (dynamic_cast <Ltruck*>(s->t)) {
                        // Unterbrechung des kleinen Baggers, der
                        // gerade ein grosses Fahrzeug bedient
                        s->interrupt ();
                }
        }
        else q->signal(); // Wecken des kleinen Baggers
        stq->wait(); // Warten auf kleinen Bagger
        return 0;
}
```

LargeTruck

```
class Truck : public Process {
public:
    double load;       // Ladung
    double capacity;  // Fassungsvermoegen

    Truck( Simulation &sim, const Label &name) :
            Process(sim, name){}
};

class Ltruck : public Truck {
public:
     Ltruck (Simulation& sim) : Truck(sim, "L") {}
     virtual int main();
};
```

```
int Ltruck::main() {
        capacity = 20.0;
        load= 0.0;
        while (load < capacity) { // noch nicht komplett beladen
                q->signal();          // Aktivierung des kleinen Baggers
                ltq->wait();          // Warten auf beliebigen Bagger
        }
        return 0;
}
```

Logging und Report

```
// Statistikpuffer
using odemx::data::buffer::StatisticsBuffer;
static std::tr1::shared_ptr < StatisticsBuffer > stats =
                                        StatisticsBuffer::create (sim, "Statistik", false);
sim.addConsumer (data::channel_id::statistics, stats);

//enable logging
odemx::data::output::XmlWriterPtr xmlWriter =
                                odemx::data::output::XmlWriter::create("xmltrace.xml");
sim.addConsumer(odemx::data::channel_id::trace, xmlWriter);
```

```
//remove trace comsumer
sim.removeConsumer(xmlWriter);

// Report
data::output::XmlReport xmlReport("digger.xml");
xmlReport.addReportProducer (*stats);
xmlReport.generateReport ();
```

```
int main () {
    Simulation& sim = getDefaultSimulation ();
     LtruckGenerator   *lt;
     StruckGenerator   *st;

// Statistikpuffer und Loging- Trace- Preparation
...
// Initialisierung
   nextl = new NegativeExponential(sim, "next large", 1/22.0);
   nexts = new NegativeExponential(sim, "next small", 1/10.0);
   q     = new Condq (sim, "sq");
   stq   = new Waitq (sim, "s_Truck_q");
   ltq   = new Waitq (sim, "l_Truck_q");

   s     = new Sdigger (sim);
   s->activate ();
   new Ldigger (sim)->activate ();
   new Ldigger (sim)->activate ();
   new LtruckGenerator (sim)->activate ();
   new StruckGenerator (sim)->activate ();

   sim.runUntil (8*60.0);  // 8 h
// Report- Trace-Handling
...
   return 0;
}
```

Report (ODEMx 3.0) nicht gut interpretierbar

ODEMx XML Report

odemx::synchronization::WaitQ

Name	*Reset at*	*master queue*	*slave queue*	*synchronizations*
l_Truck_q	0	l_Truck_q.master-queue	l_Truck_q.slave-queue	223
s_Truck_q	0	s_Truck_q.master-queue	s_Truck_q.slave-queue	96

odemx::data::buffer::StatisticsBuffer Updates

Producer	*Property*	*Updates*	*Min*	*Max*	*Mean*	*Weighted Mean*	*Std Deviation*	*Weighted StDev*
l_Truck_q				0.61039	0.0839453	0.0665617	0.165067	0.145425
l_Truck_q				0.449572	0.143207	0.119505	0.13614	0.131988
l_Truck_q.master-queue	length	103	0	2	0.902913	0.525674	0.782262	0.825623
l_Truck_q.slave-queue	length	456	0	14		72608		3.83903
s_Truck_q	master wait time	96	0	0			0	0
s_Truck_q	slave wait time	96	0	0.740528	0.244685	0.219993	0.195846	0.18316
s_Truck_q.master-queue	length	1	0	0	0	0	0	0
s_Truck_q.slave-queue	length	197	0	9	2.58883	2.36805	2.16771	2.12359
sq	wait time	115	0	0.0174378	0.000188922	0.00117587	0.00166384	0.00435027
sq.queue	length	231	0	1	0.497835	0.00218234	0.499995	0.0466645

maximal wartende Fahrzeuge

maximal wartende Bagger

odemx::random::NegativeExponential

Name	*Reset at*	*inverse mean*	*seed*	*uses*
next large	0	22		215
next small	0	10		100

angekommene Fahrzeuge

odemx::synchronization::CondQ

Name	*Reset at*	*queue*	*signals*	*users*
sq	0	sq.queue	320	115

wünschenswert: Aktuell wartende Fahrzeuge und Bagger, Anzahl der “Durchläufer” wie in früheren ODEMx - Versionen

6. ODEMx-Modul Synchronisation:

WaitQ, CondQ

- Konzept WaitQ

  Beispiel: Tankerflotte / Hafen / Raffinerie

- Konzept CondQ

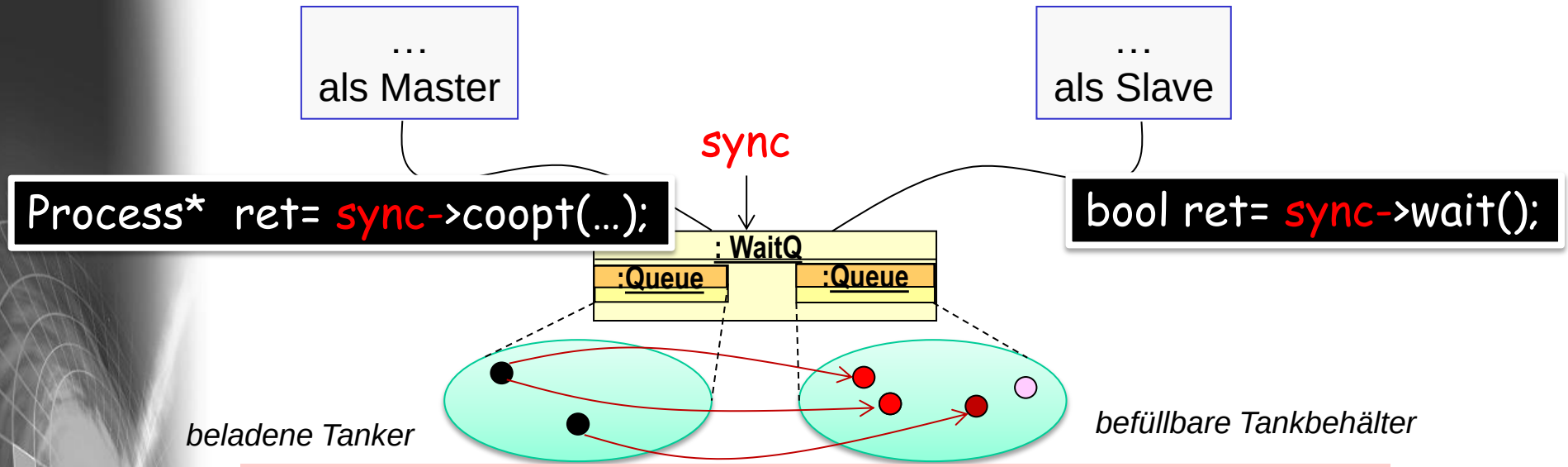
  Beispiel: Hafen / Schlepper / Gezeiten

- Weitere Anwendungsbeispiele für WaitQ u. CondQ

  Beispiel: Bagger-Zuteilung

- Zusammenfassung/einheitliche Betrachtung

Zwischenfazit: Master-Slave-Synchronisation

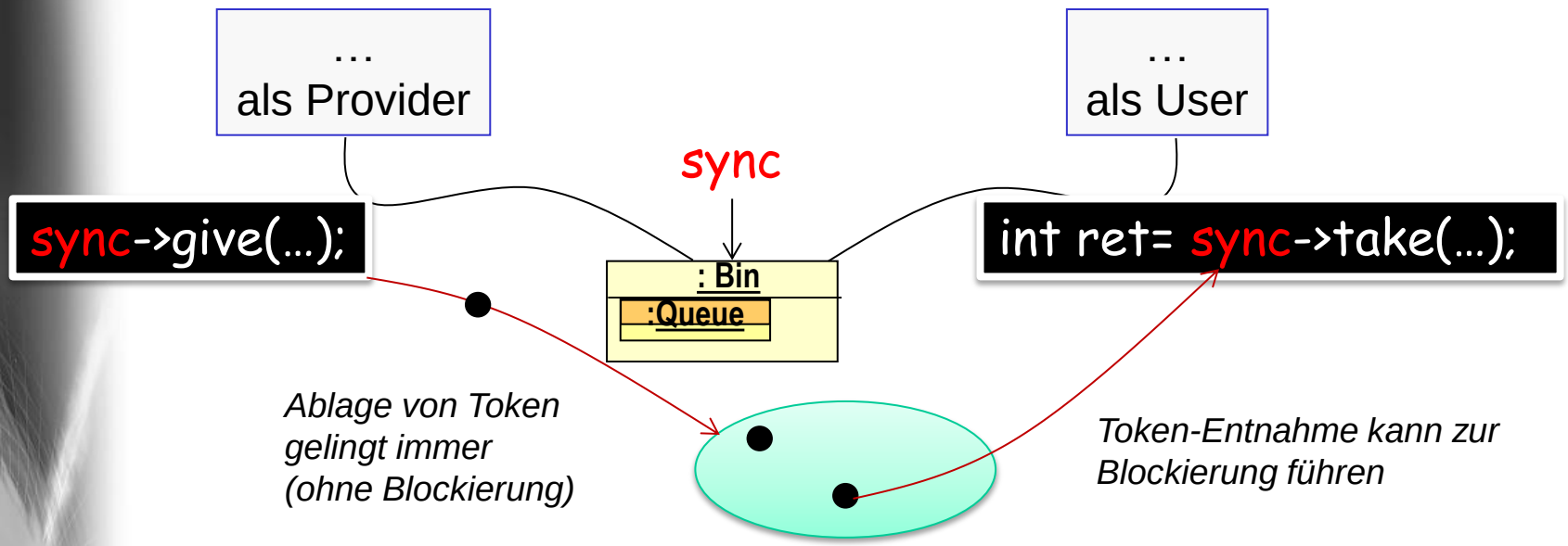


1. Selection-Funktion kann je Master(typ) verschieden sein,
   sie ist in der jeweiligen Prozessableitung zu definieren
2. Eine Prozessableitung kann mit mehreren Funktionen des Selection-Typs ausgestattet sein,
   bei jedem Coopt-Ruf kann eine andere Selektion vorgenommen werden

- Der Wartevorgang eines Masters auf geeignete Slaves kann unterbrochen werden (coopt gibt Null-Zeiger zurück, sonst Slave-Zeiger)
- Der Wartevorgang eines Slaves kann unterbrochen werden (wait gibt False zurück, sonst True)

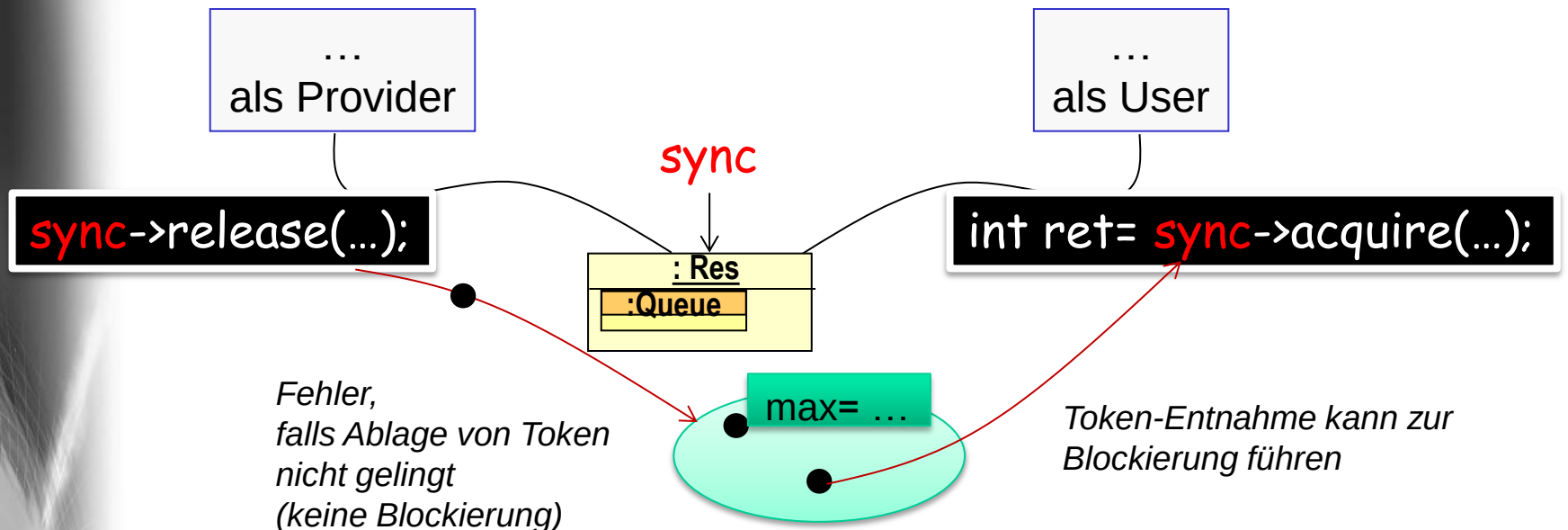
- Typ der Funktion xxx: SELECTION bool xxx (Process *p)

Zwischenfazit: Bin-Synchronisation



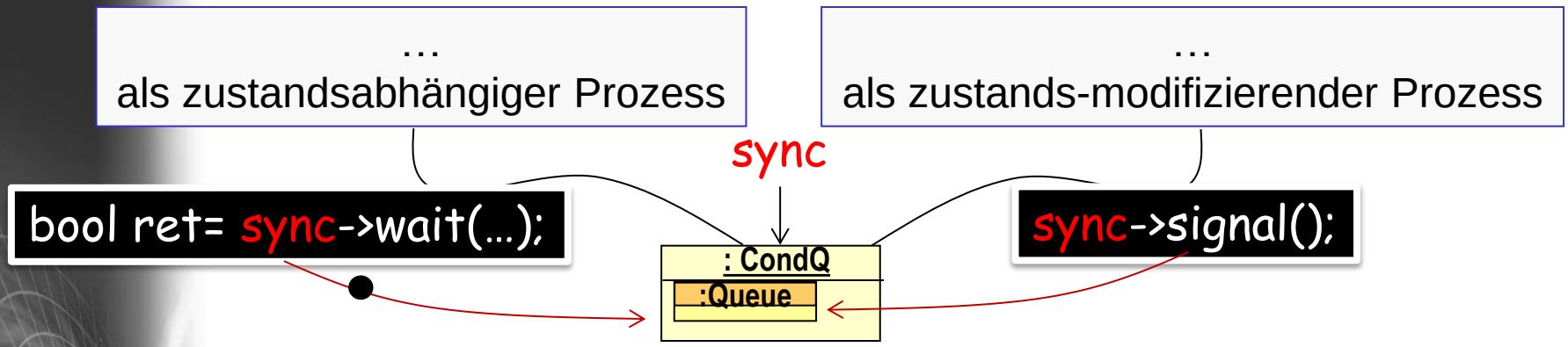
- Der Wartevorgang eines User-Process auf die Verfügbarkeit einer geforderten Anzahl von Token kann unterbrochen werden
  werden (take gibt int-Null zurück, sonst Anzahl der entnommenen Token)

Zwischenfazit: Res-Synchronisation



- Der Wartevorgang eines User-Process auf die Verfügbarkeit einer geforderten Anzahl von Token kann unterbrochen werden
  werden (acquire gibt int-Null zurück, sonst Anzahl der entnommenen Token)

Zwischenfazit: CondQ-Synchronisation



Blockierung bei Nichterfüllung der Zustandsbedingung
Fortsetzung bei Erfüllung

Aktivierung des ersten blockierten Prozesses

- Der Wartevorgang eines zustandsabhängigen Prozesses kann abgebrochen werden, ohne dass die Zustandsbedingung erfüllt ist
  (waitUntil gibt False zurück, sonst True)

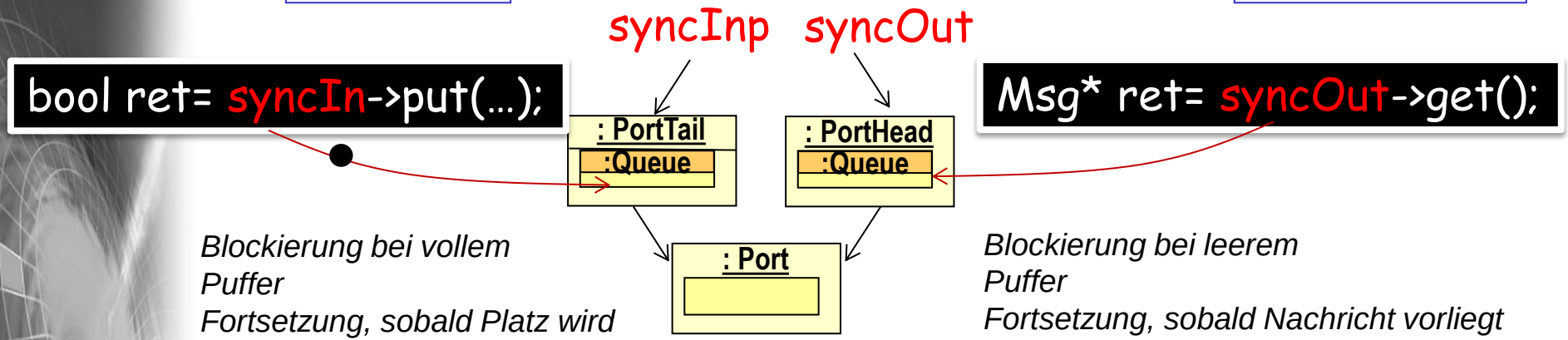
- Typ der Funktion: CONDITION bool xxx ()

Zwischenfazit: Port-Synchronisation

- Modus: Blockierung -

…
als Sender

…
als Empfänger



Blockierung bei vollem Puffer
Fortsetzung, sobald Platz wird

Blockierung bei leerem Puffer
Fortsetzung, sobald Nachricht vorliegt

- Der Wartevorgang eines Sender-Prozesses kann abgebrochen werden, ohne dass Puffer freigeworden ist
  (put gibt false zurück, sonst True)

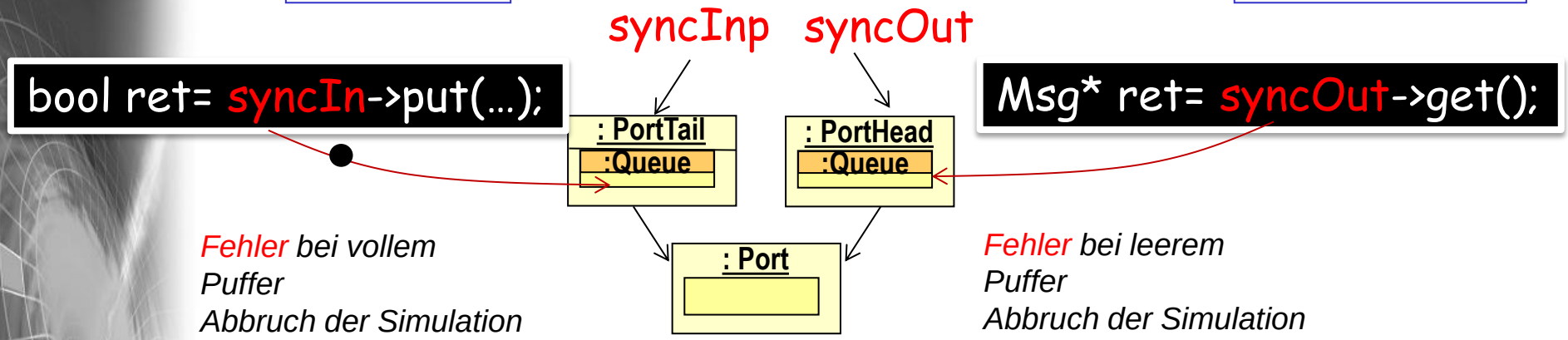
- Der Wartevorgang eines Empfänger-Prozesses kann abgebrochen werden, ohne dass Puffer gefüllt wird
  (get gibt Null-Zeiger zurück, sonst Nachrichten-Zeiger)

Zwischenfazit: Port-Synchronisation

- Modus: Fehler -

…
als Sender

…
als Empfänger



Fehler bei vollem Puffer
Abbruch der Simulation

Fehler bei leerem Puffer
Abbruch der Simulation

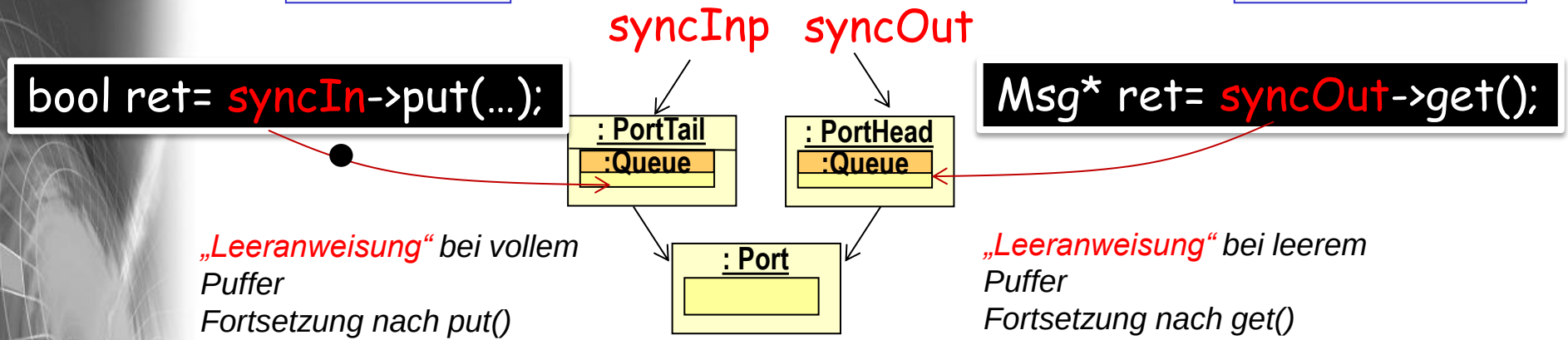
- Es treten weder beim Sender noch beim Empfänger Blockierungen/Verzögerungen ein

Zwischenfazit: Port-Synchronisation

- Modus: Leeranweisung-

…
als Sender

…
als Empfänger



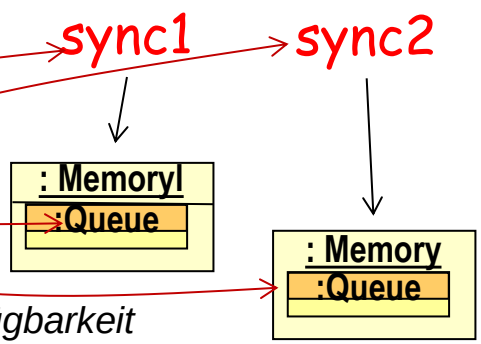
- Es treten weder beim Sender noch beim Empfänger Blockierungen/Verzögerungen ein

Zwischenfazit: Wait-For-Memo-Synchronisation

```
m= wait (ph,pt, t);
switch (m->getMemoType()) {
   case PORTTAIL: …
   case PORTHEAD: …
   case TIMER:…
   case COND:…
   default: …
```

… als Wartender auf Verfügbarkeit verschiedener Objekte

```
Memory *ret= wait(…);
```



Blockierung bei Nichtverfügbarkeit aller Memo-Objekte
Fortsetzung, sobald ein Objekt verfügbar wird

… als Memory-Objekt-Manipulator

```
sync1->get();
```

```
sync2->put();
```

```
sync3
```
Timer

```
sync4->signal()
```

- Der Wartevorgang eines wartenden Prozesses kann abgebrochen werden, ohne dass ein Memo-Objekt verfügbar geworden ist (wait gibt Null-Zeiger zurück, sonst Zeiger zum ersten verfügbaren Memo-Objekt)

ODEMx- Funktionstypen

typedef bool (Process::* Condition)()

typedef bool (Process::* Selection)(Process *partner)

typedef double (Process::* Weight)(Process *partner)

a) sind vom Anwender (als Memberfunktion
benötigter Process-Ableitung zu implementieren)

b) sind im Bedarfsfall bei Aufruf von
- wait
- coopt
- avail

zu übergeben

c) werden dann von wait, coopt bzw. avail zur Laufzeit auf Process-Instanzen angewendet, um aus einer Processinstanzmenge genau eine Instanz auszuwählen