Kurs OMSI
im WiSe 2013/14

Objektorientierte Simulation mit ODEMx

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

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

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
- Anwendungsbeispiel (Bin, Res, WaitQ)

Typische Probleme

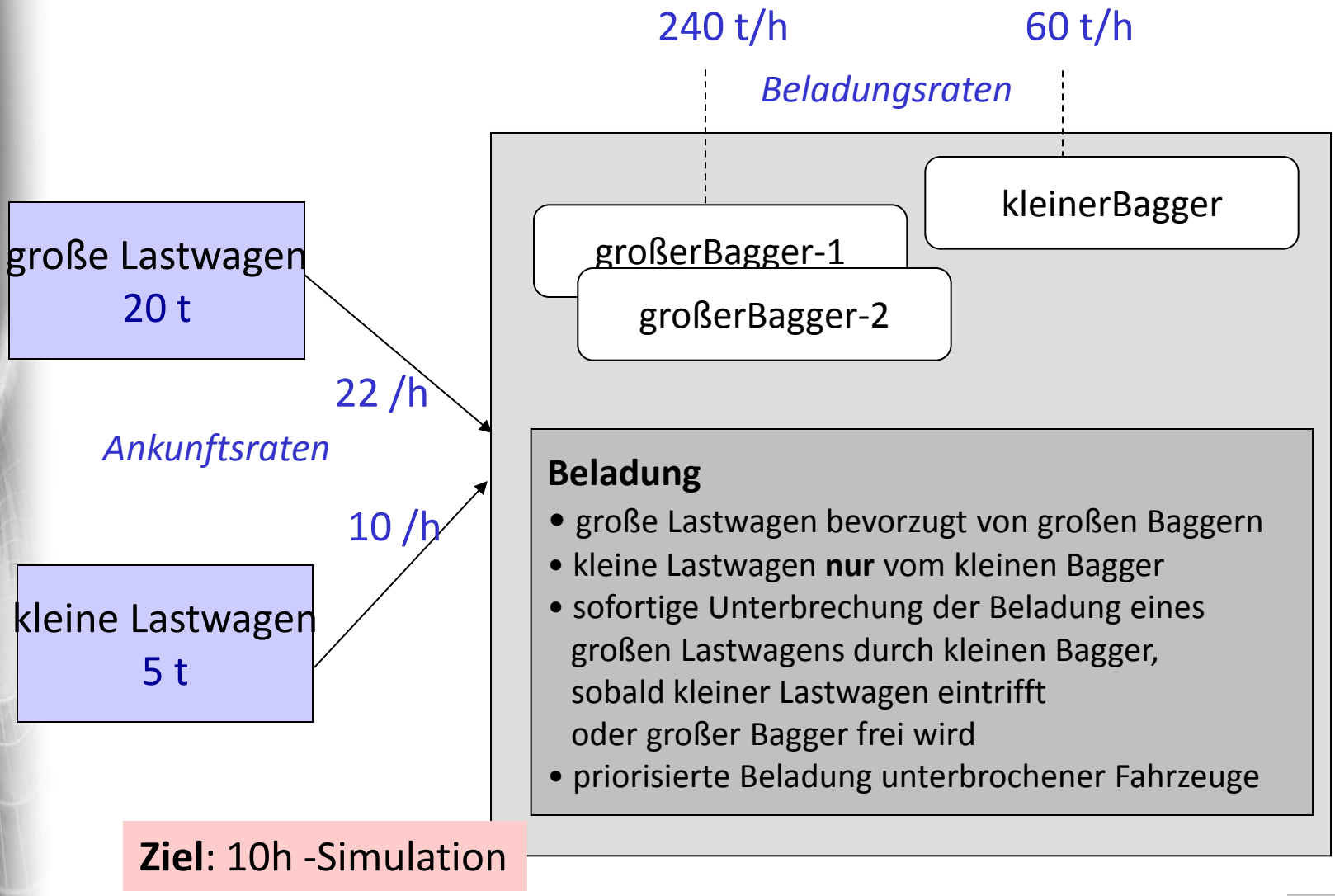
- Zuweisung einer Ressource aus einem Spektrum unterschiedlicher **Ressourcenklassen**
  für die Durchführung spezifischer Arbeitsgänge

- 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)

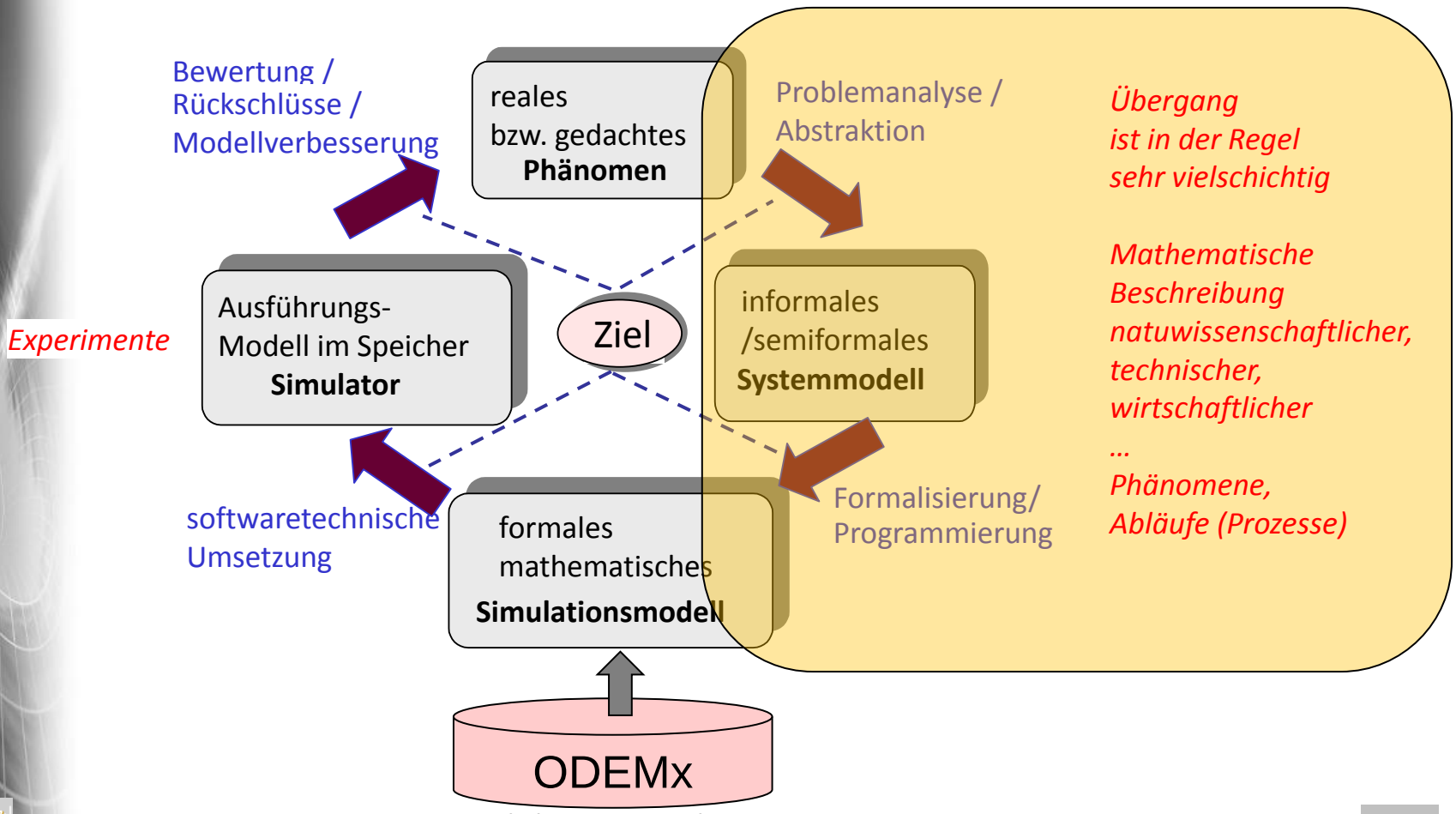
Beispiel: Transportfahrzeuge – Bagger – Beladung



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

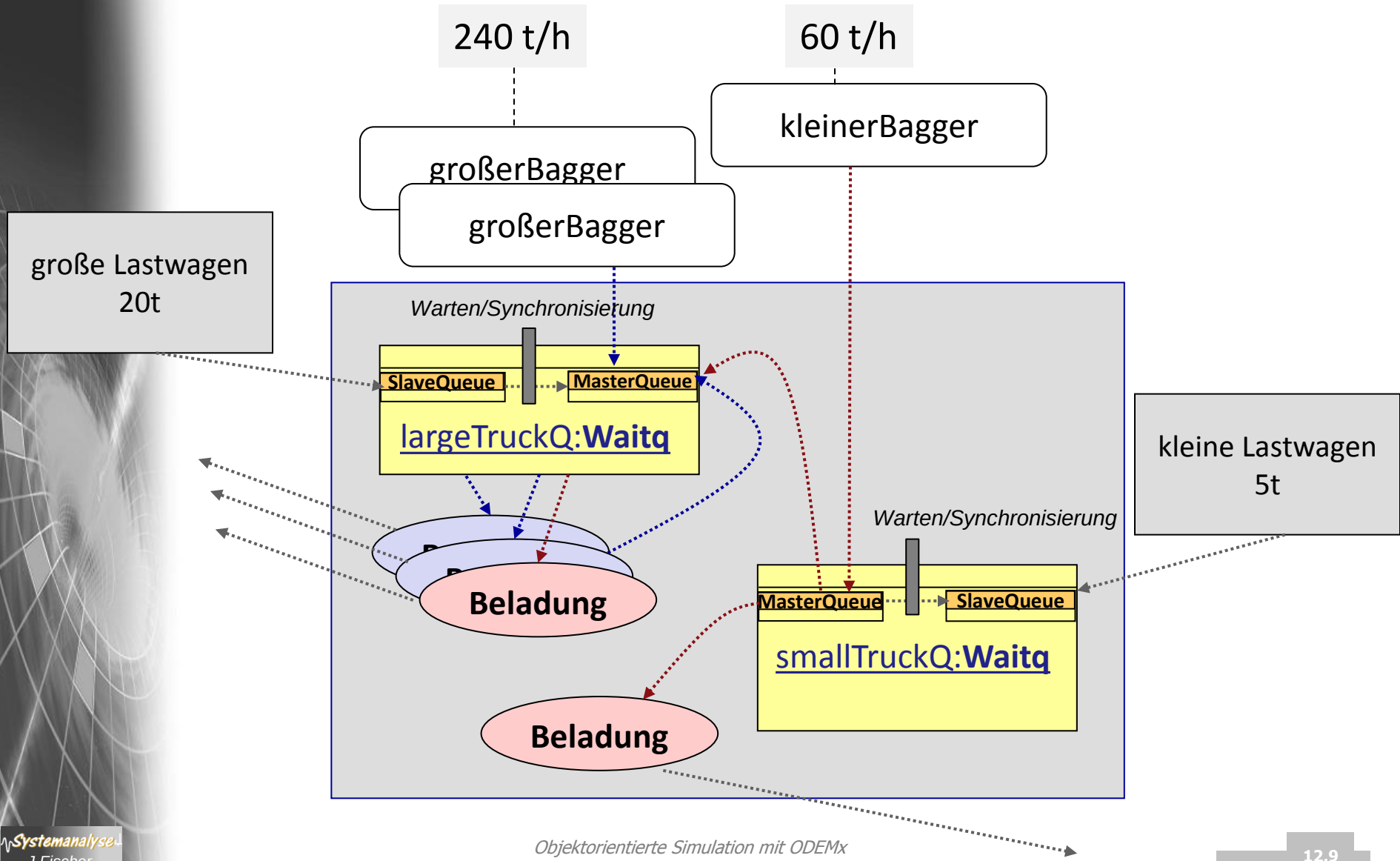
Fortsetzung als Master in einem anderen WaitQ-Objekt

b) mit WaitQ- und CondQ-Einsatz

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

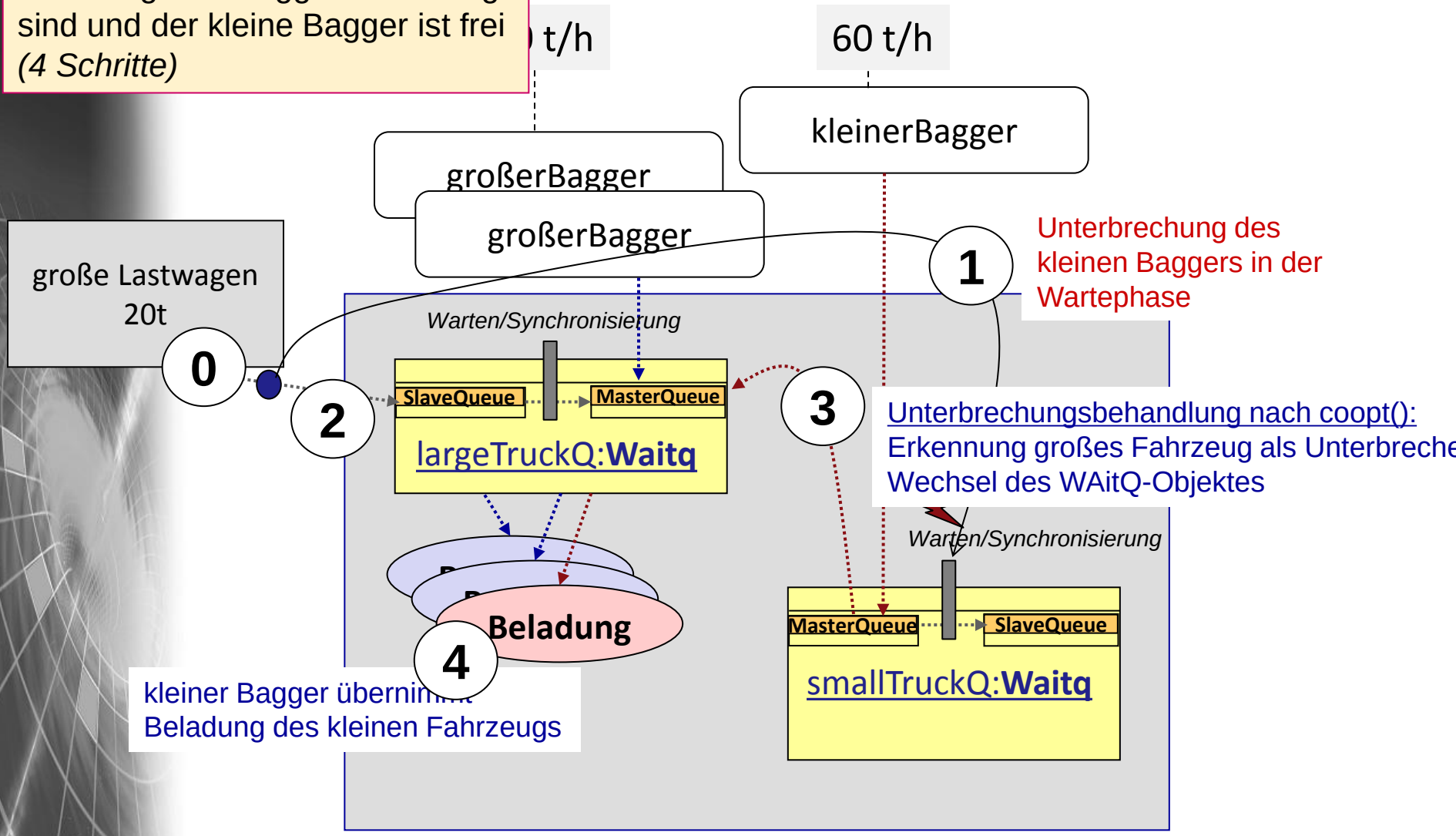
Fortsetzung als Master in einem der WaitQ-Objekte

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



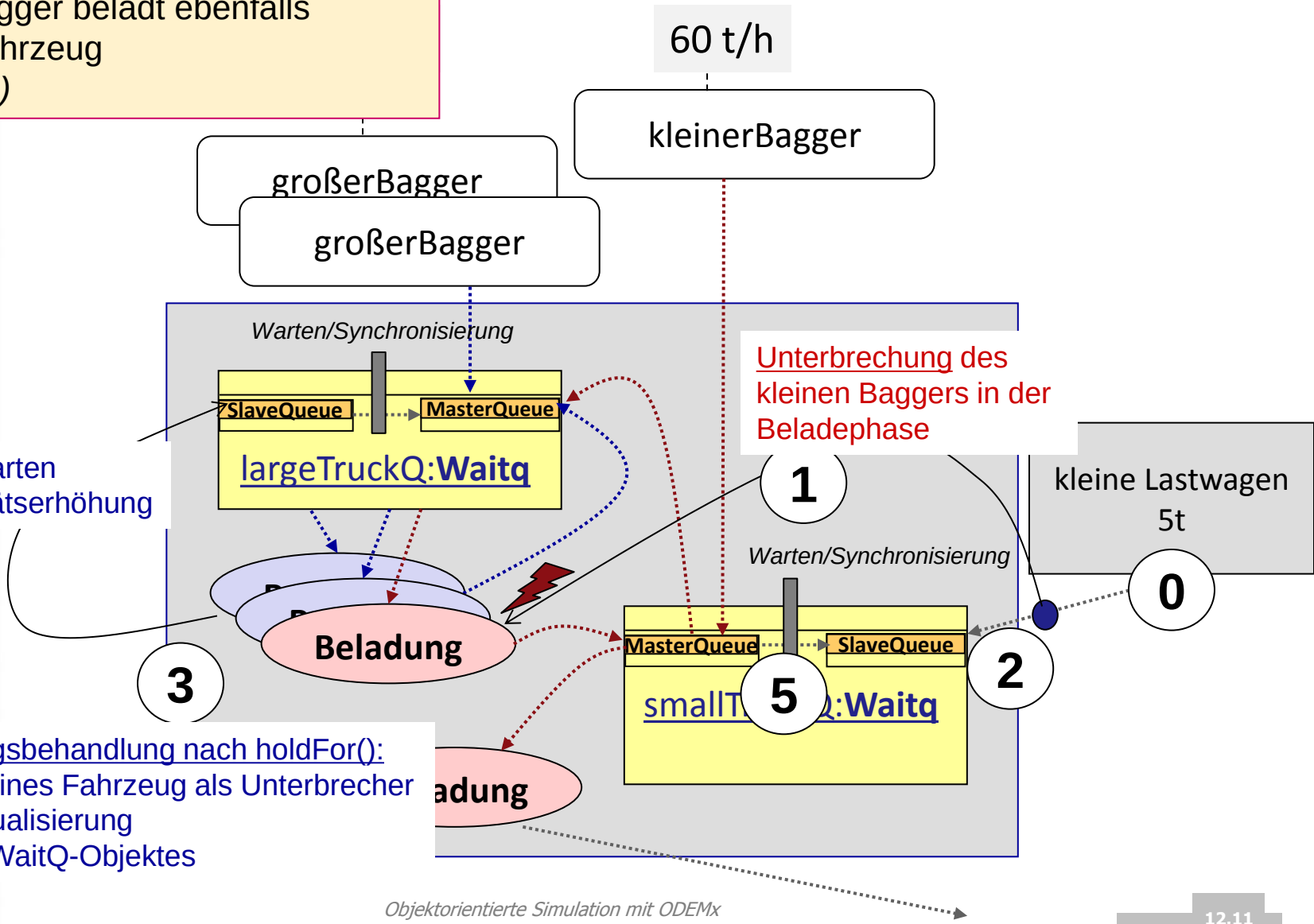
1. Fall:
ein großes Fahrzeug trifft ein, wobei 2 große Bagger beschäftigt sind und der kleine Bagger ist frei *(4 Schritte)*

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

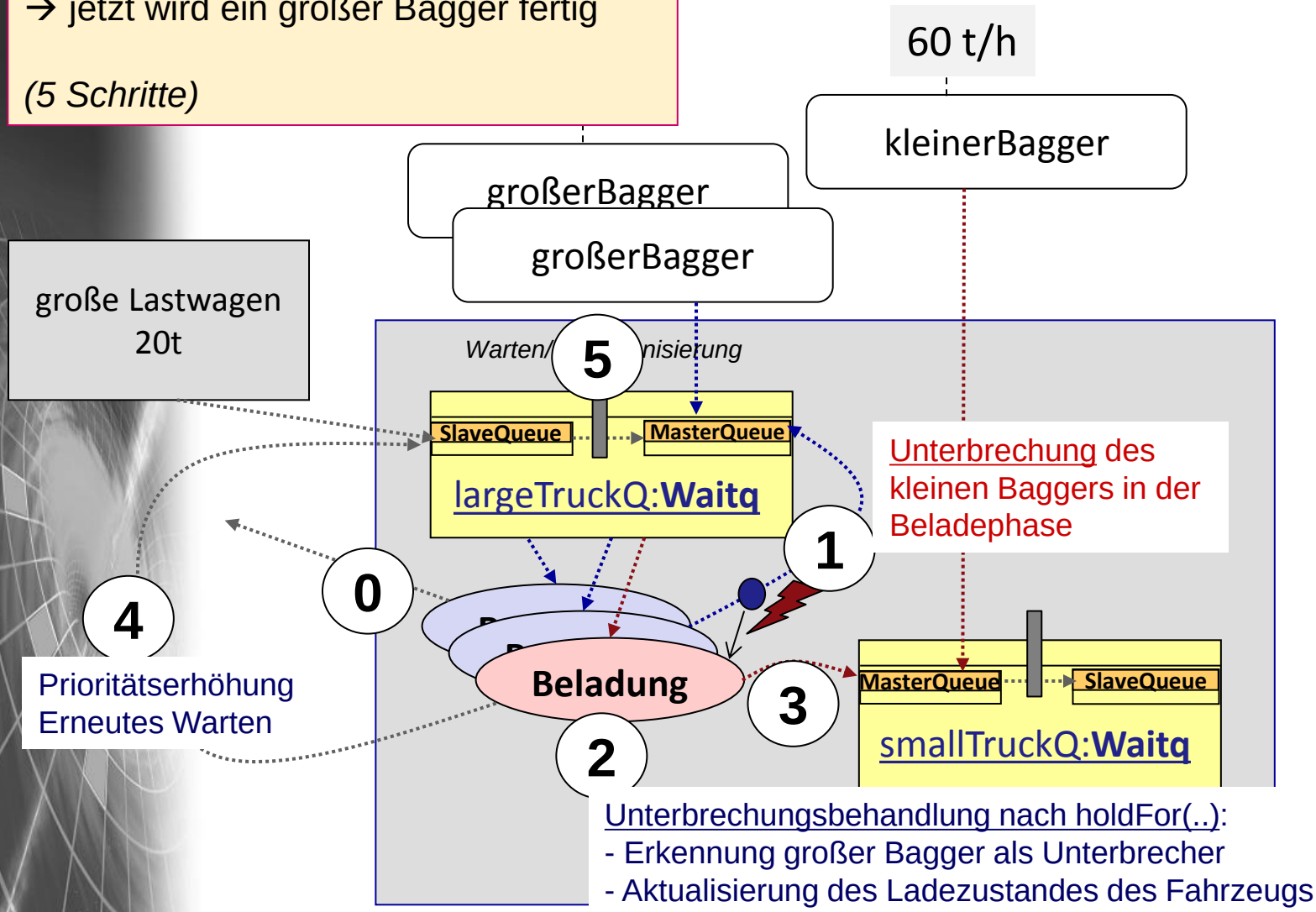


Unterbrechungsbehandlung nach holdFor():
Erkennung kleines Fahrzeug als Unterbrecher
Baladungsaktualisierung
Wechsel des WaitQ-Objektes

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

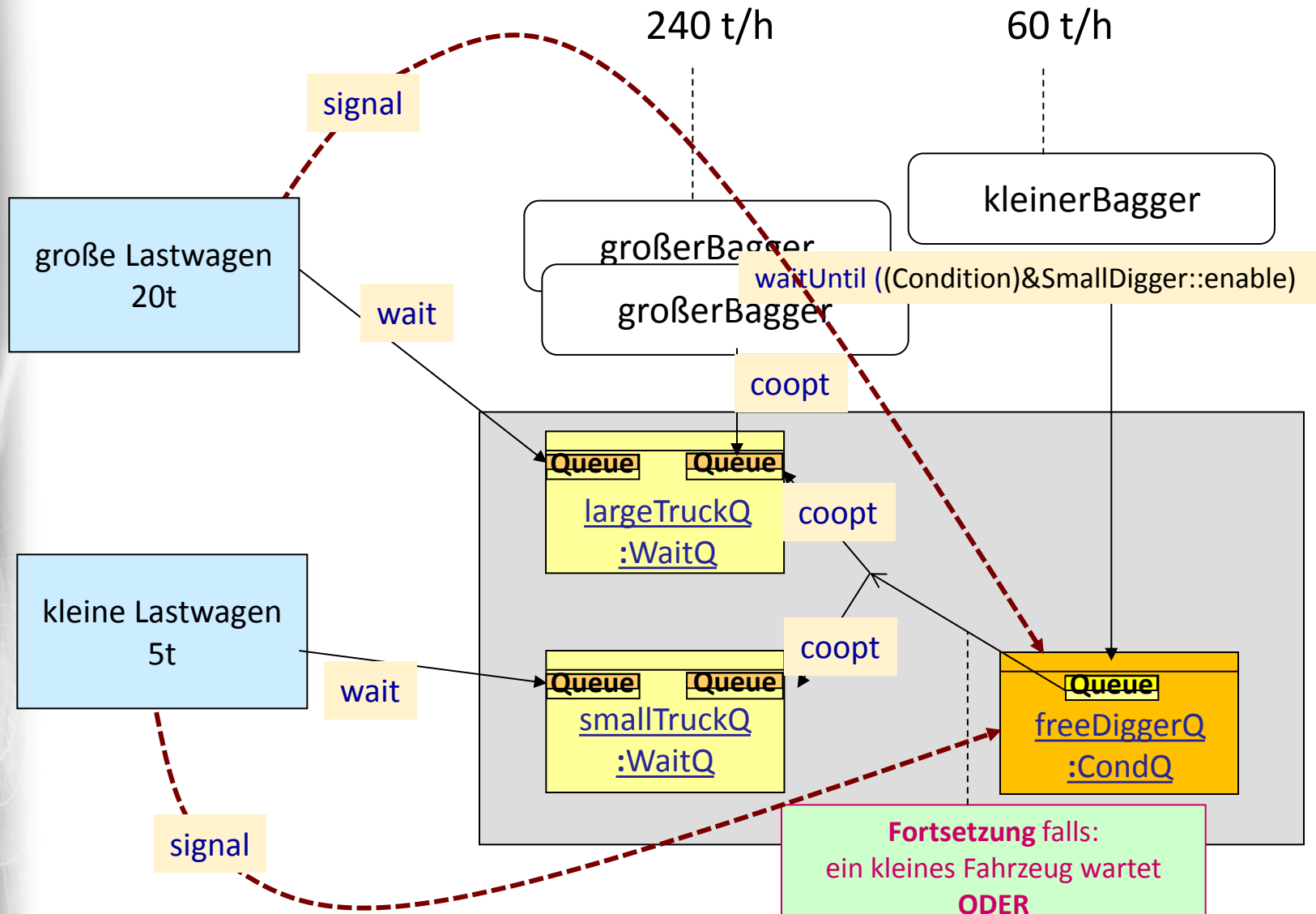
```
int SmallDigger::main() {
        double loadStart= 0; // Beladungsbeginn
        for (;;) {
          workSt= FREE;
          if (smallTruckQ->avail()) {
                    // unterbrechungsfreie Beladung
                    // eines kleinen Fahrzeugs
                    … workSt= SMALL;
          }
          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();
}
```

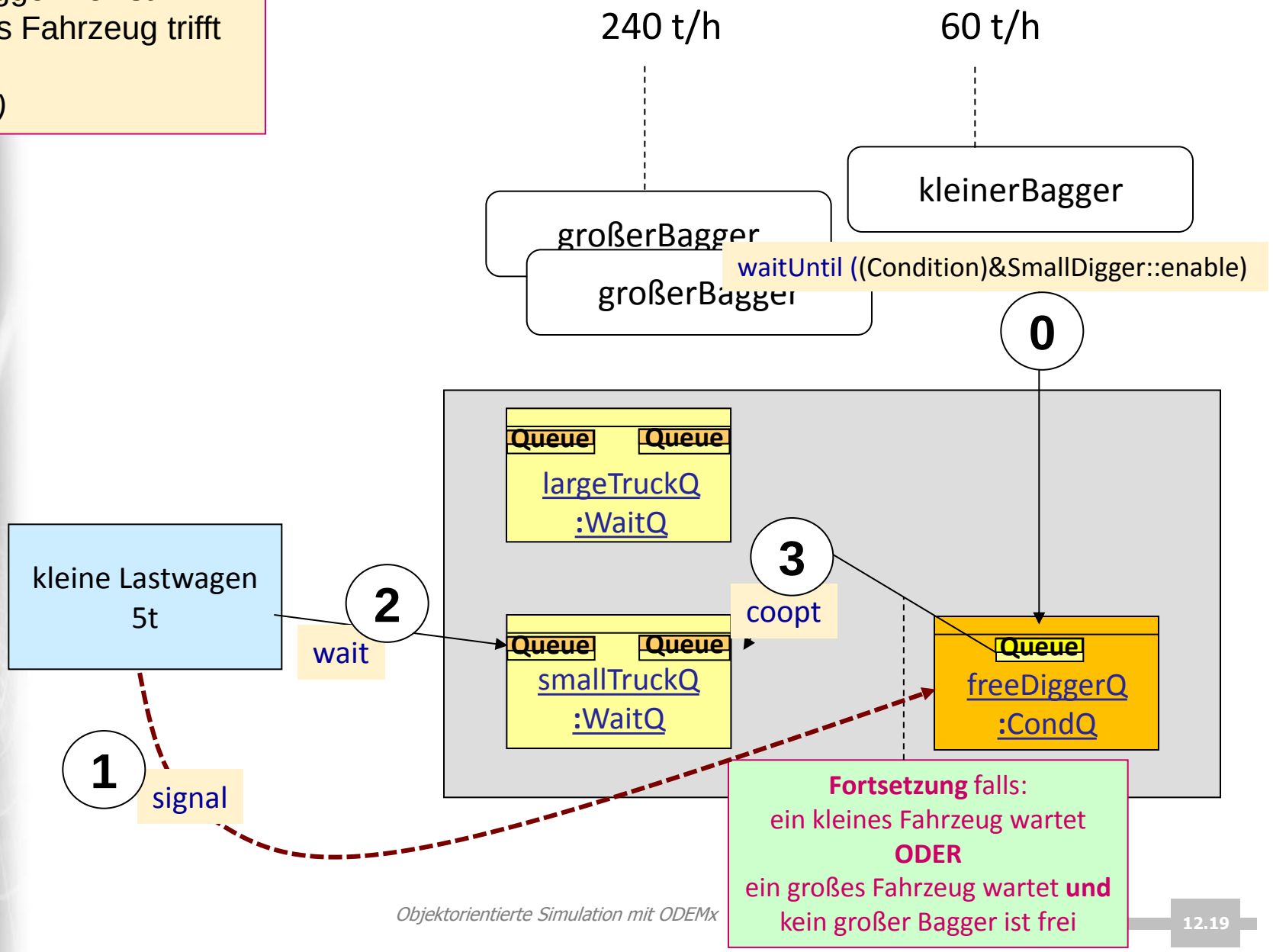
Process: Scheduling-Operationen

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***

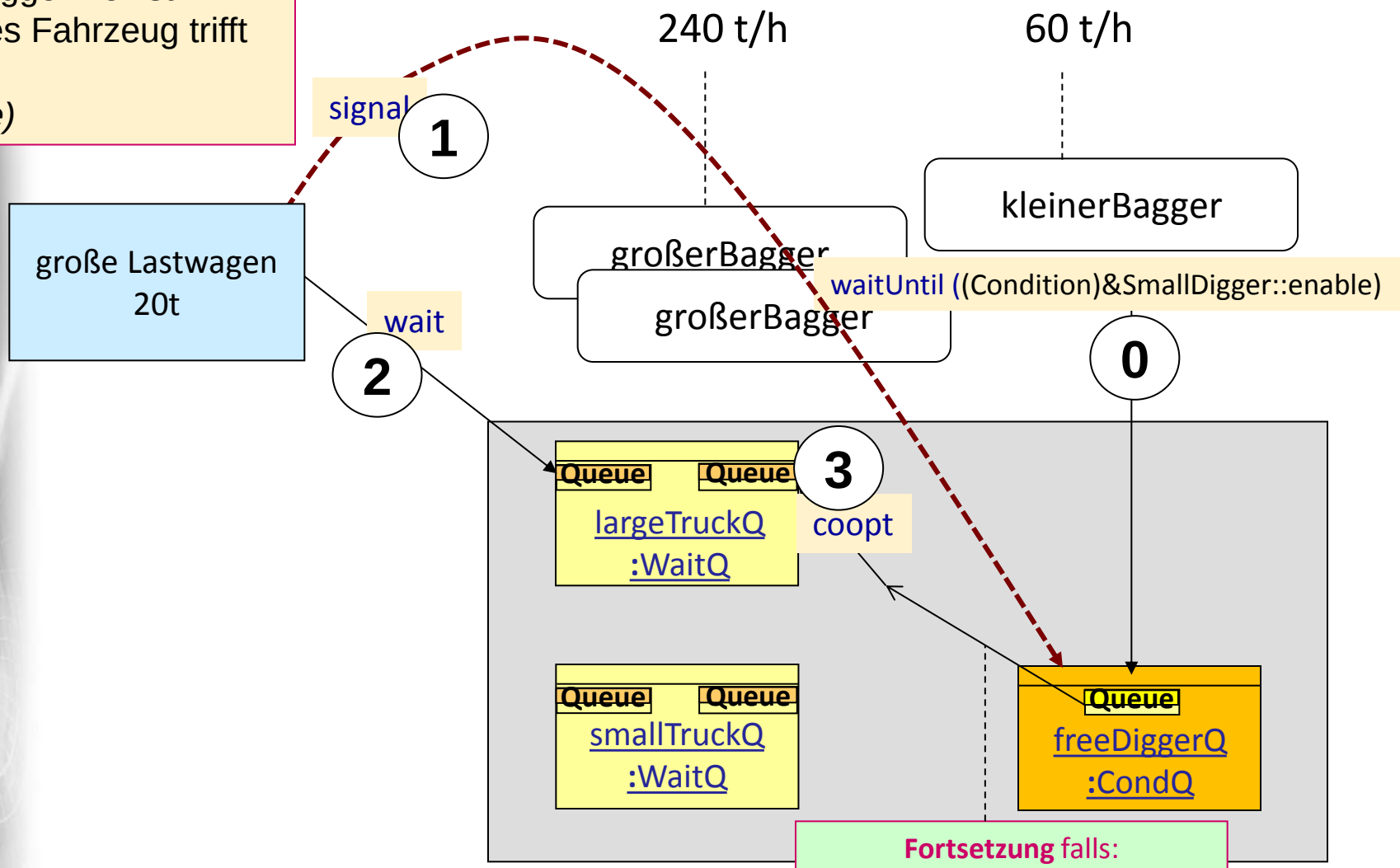
Variante b)


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

1. Fall:
kleiner Bagger frei ist
und kleines Fahrzeug trifft ein
(3 Schritte)

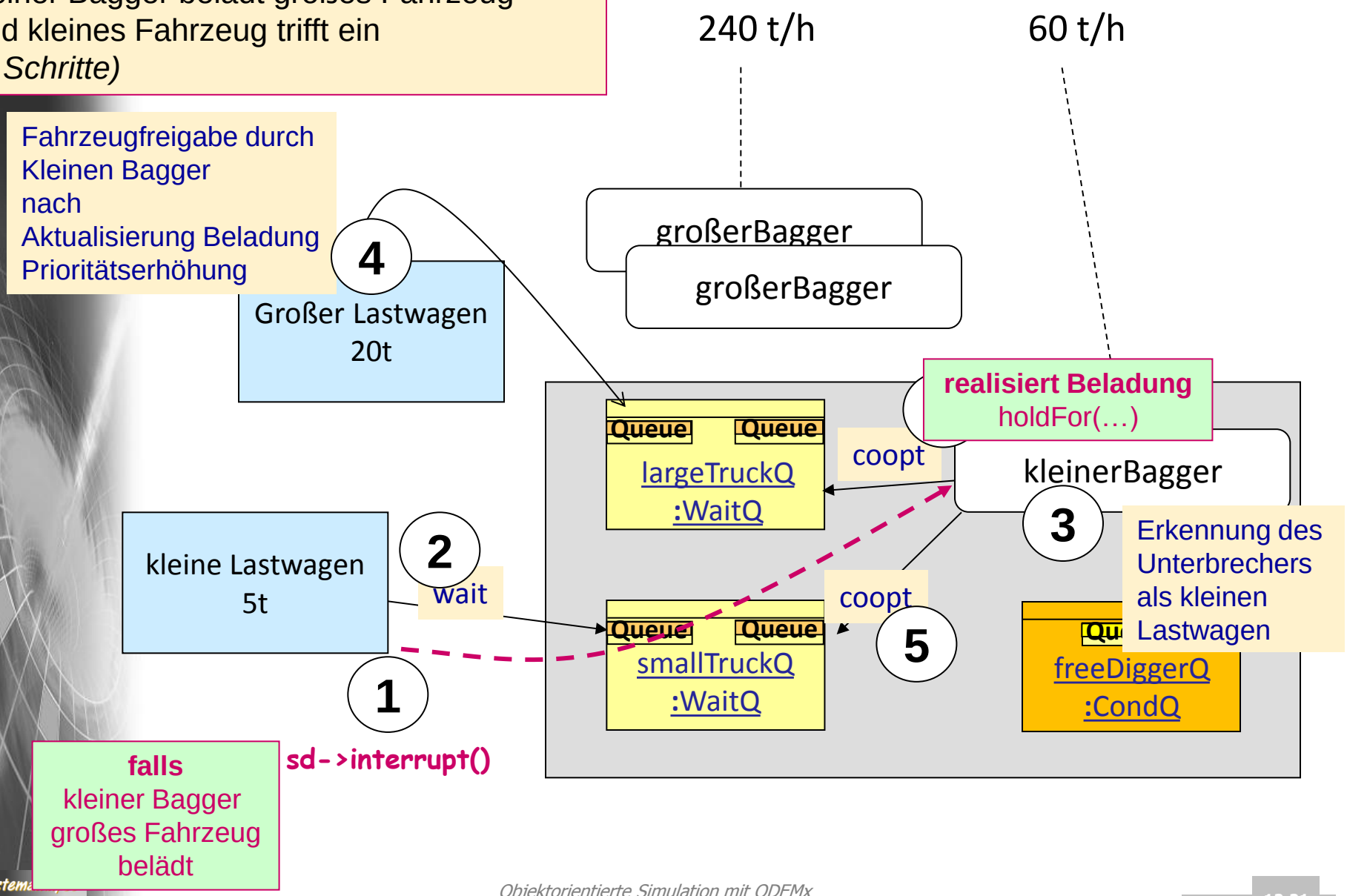


2. Fall:
kleiner Bagger frei ist
und großes Fahrzeug trifft ein
(3 Schritte)

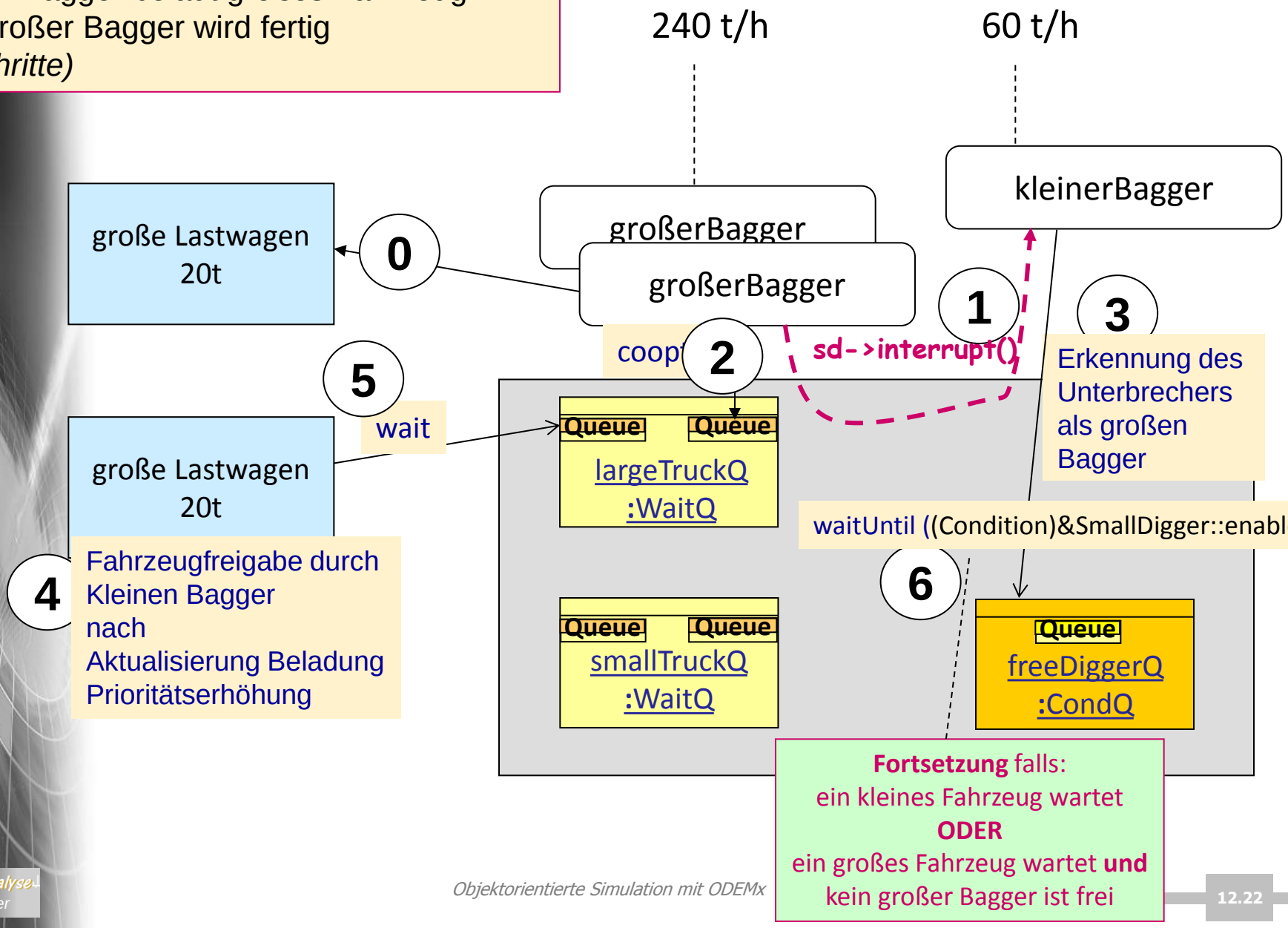


Fortsetzung falls:
ein kleines Fahrzeug wartet
ODER
ein großes Fahrzeug wartet **und**
kein großer Bagger ist frei

3. Fall:
kleiner Bagger belädt großes Fahrzeug
und kleines Fahrzeug trifft ein
(5 Schritte)



4. Fall:
kleiner Bagger belädt großes Fahrzeug
und großer Bagger wird fertig
(3 Schritte)



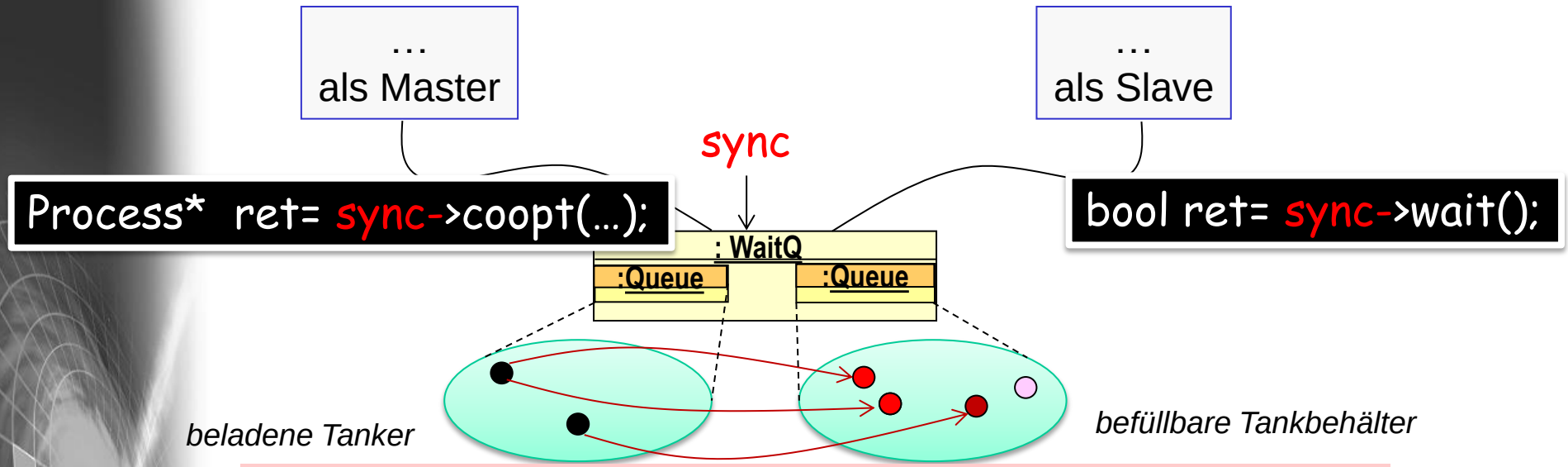
6. ODEMx-Modul Synchronisation:
WaitQ, CondQ

- Konzept WaitQ
  - Beispiel: Tankerflotte, Hafen, Raffinerie
- Konzept CondQ
  - Beispiel: Hafen, Schlepper, Gezeiten
- Weitere Anwendungsbeispiele für WaitQ u. CondQ
- Zusammenfassung/einheitliche Betrachtung
- Anwendungsbeispiel (Bin, Res, WaitQ)

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
- Anwendungsbeispiel (Bin, Res, WaitQ)

Zwischenfazit: Master-Slave-Synchronisation

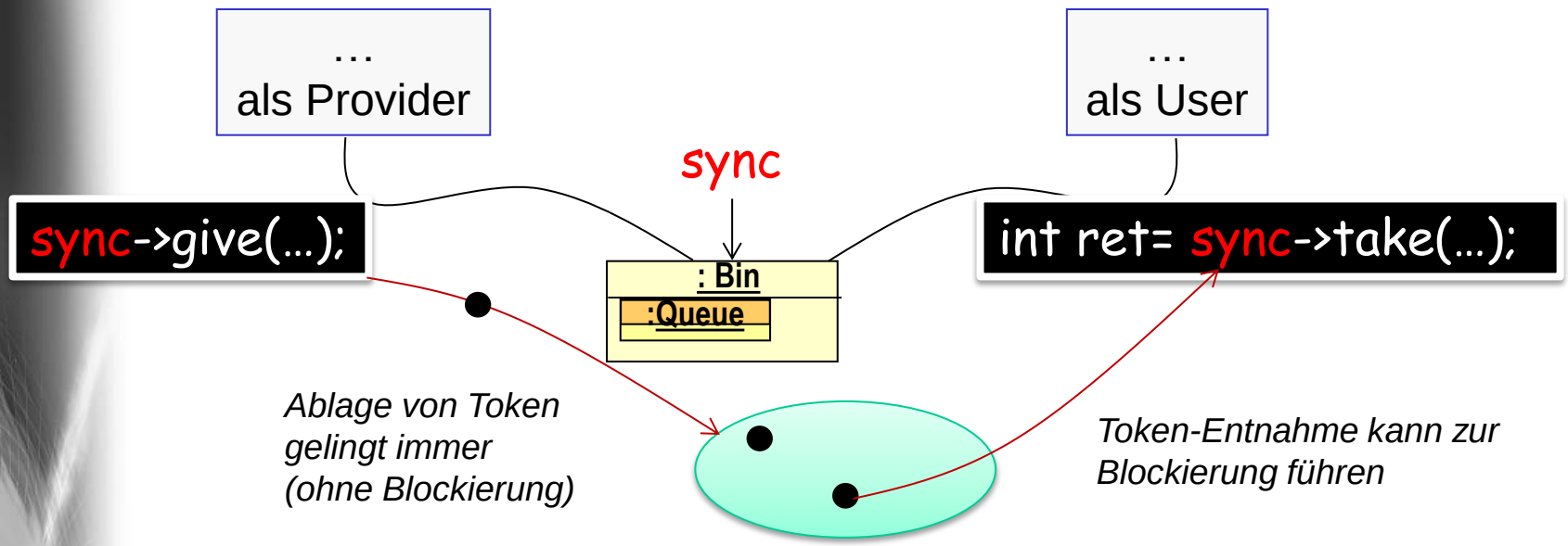


1. Selection-Funktion kann je Master(typ) verschieden sein,
   sie ist in der jeweiligen Prozessableitung zu definieren
2. Eine Prozessableitung kann mit mehreren Funktionen des Selection-Typs ausgestattet sein,
   bei jedem Coopt-Ruf kann eine andere Selektion vorgenommen werden

- 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)

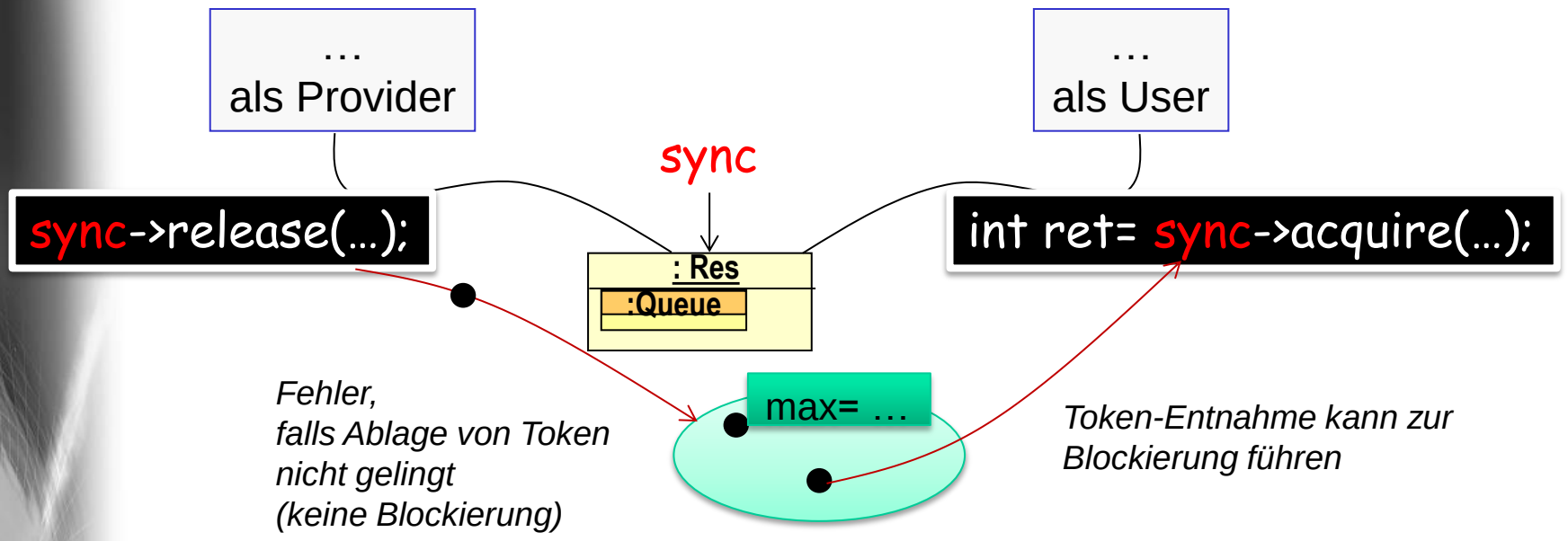
- Typ der Funktion xxx: SELECTION bool xxx (Process *p)

Zwischenfazit: Bin-Synchronisation



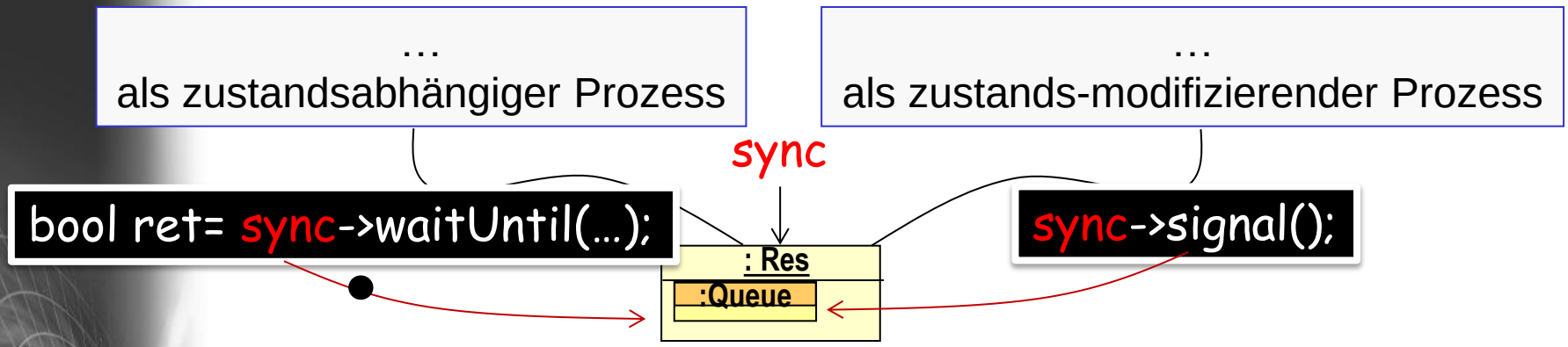
- Der Wartevorgang eines User-Process auf die Verfügbarkeit einer geforderten Anzahl von Token kann unterbrochen werden
  werden (take gibt int-Null zurück, sonst Anzahl der entnommenen Token)

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)

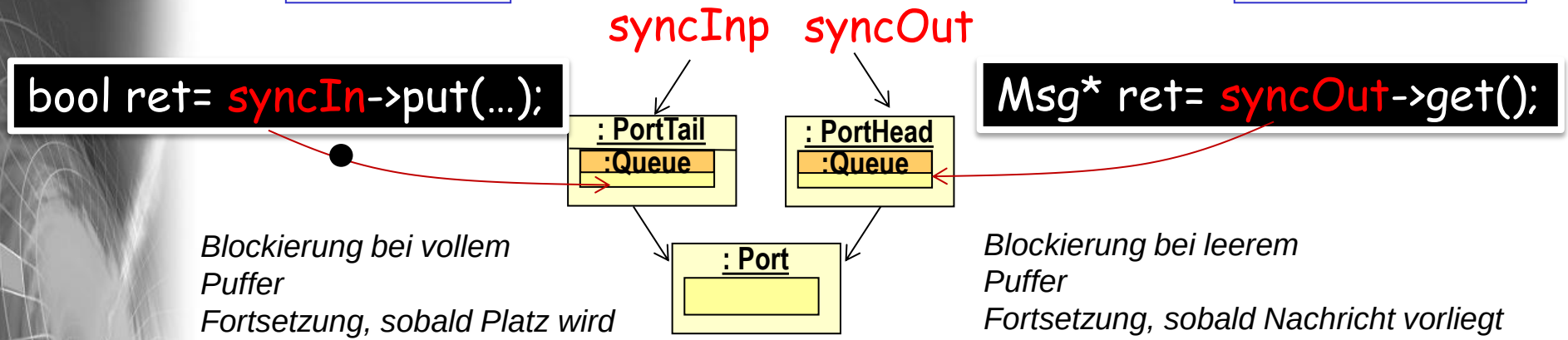
- Typ der Funktion: CONDITION bool xxx ()

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)

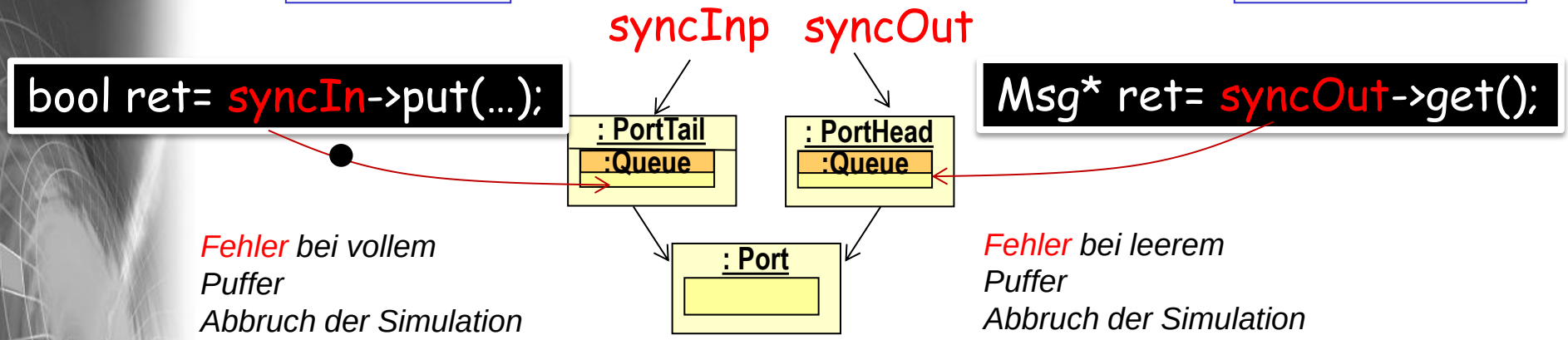
- Der Wartevorgang eines Empfänger-Prozesses kann abgebrochen werden, ohne dass Puffer gefüllt wird
  (get gibt Null-Zeiger zurück, sonst Nachrichten-Zeiger)

Zwischenfazit: Port-Synchronisation

- Modus: Fehler -

…
als Sender

…
als Empfänger



Fehler bei vollem Puffer
Abbruch der Simulation

Fehler bei leerem Puffer
Abbruch der Simulation

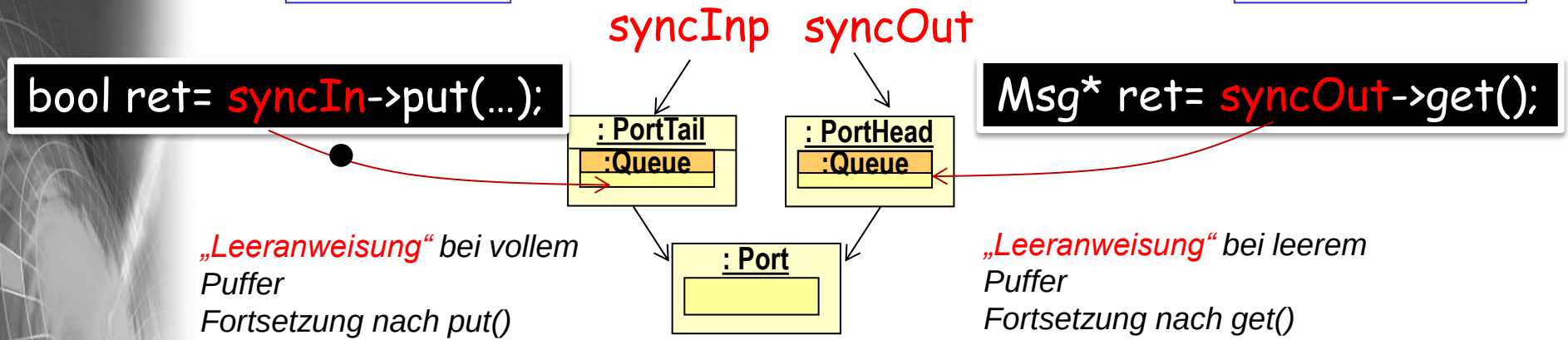
- Es treten weder beim Sender noch beim Empfänger Blockierungen/Verzögerungen ein

Zwischenfazit: Port-Synchronisation

- Modus: Leeranweisung-

…
als Sender

…
als Empfänger



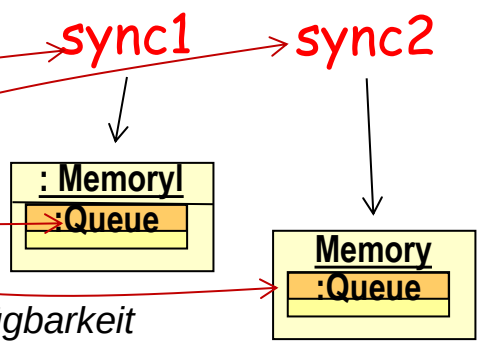
- Es treten weder beim Sender noch beim Empfänger Blockierungen/Verzögerungen ein

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 Process-Instanzmenge genau eine Instanz auszuwählen