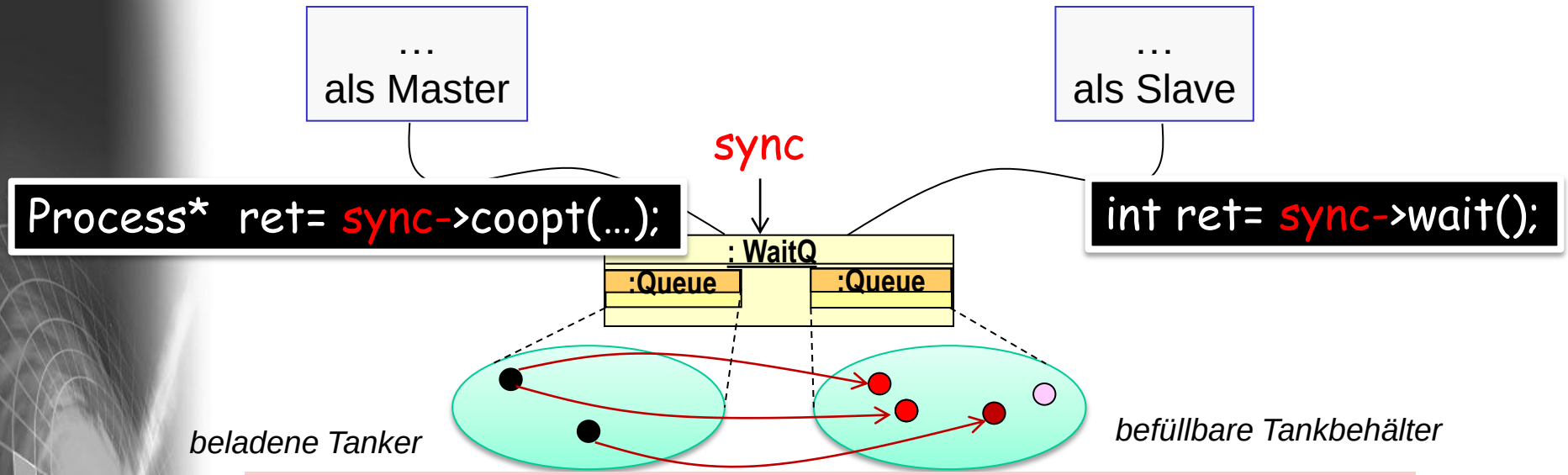
# *Kurs OMSI*
## *im WiSe 2010/11*

## *Objektorientierte Simulation mit ODEMx*

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

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

# Zwischenfazit: Master-Slave-Synchronisation

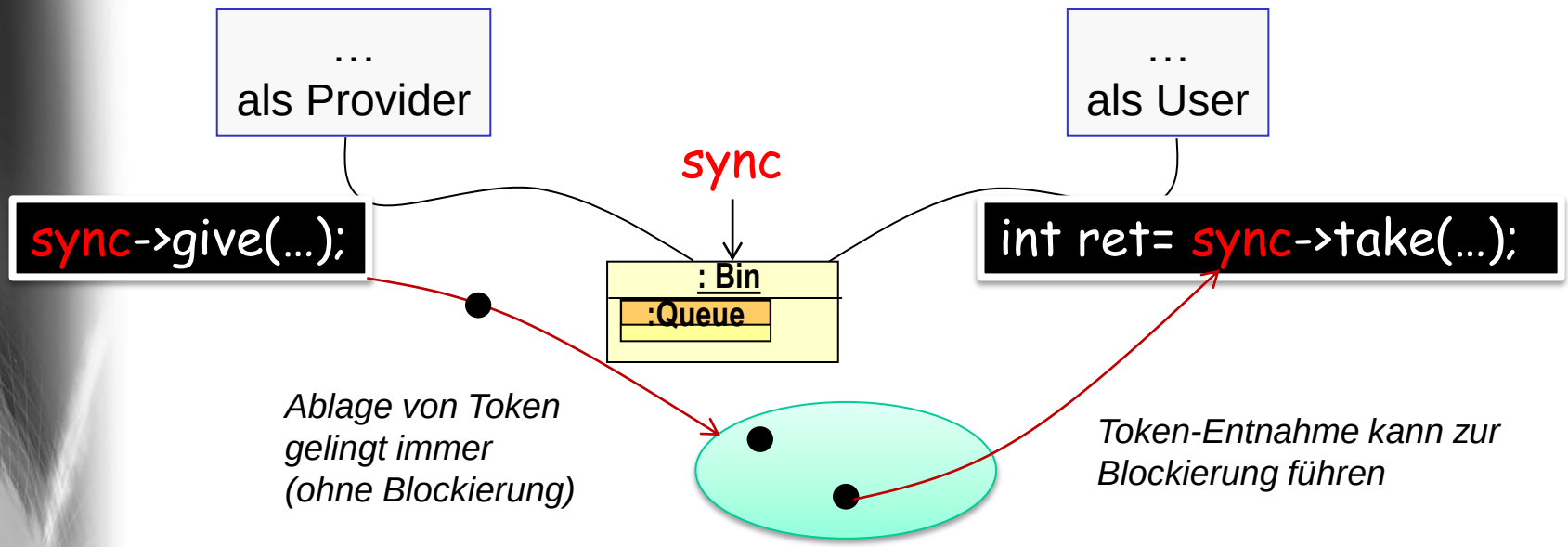



1. Selektion-Funktion kann je Master(typ) verschieden sein,
   sie ist in der jeweiligen Prozessableitung zu definieren
2. Eine Prozessableitung kann mit mehreren Funktionen des Selektion-Typs ausgestattet sein,
   bei jedem Coopt-Ruf kann eine andere Selektion vorgenommen werden

- Der Wartevorgang eines Masters auf geeignete Slaves kann unterbrochen werden (coopt gibt Null-Zeiger zurück)
- Der Wartevorgang eines Slaves kann unterbrochen werden (wait gibt int-Null zurück)

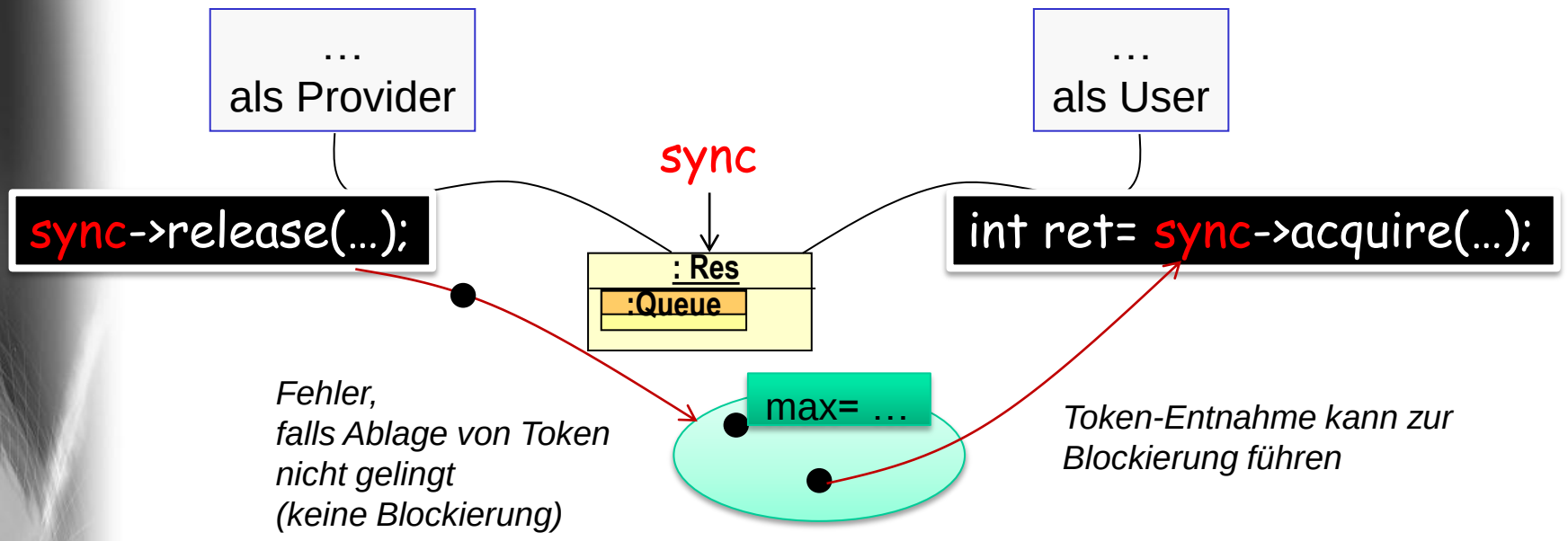
- Typ der Funktion xxx: SELECTION  bool xxx (Process *p)

# Zwischenfazit: Bin-Synchronisation



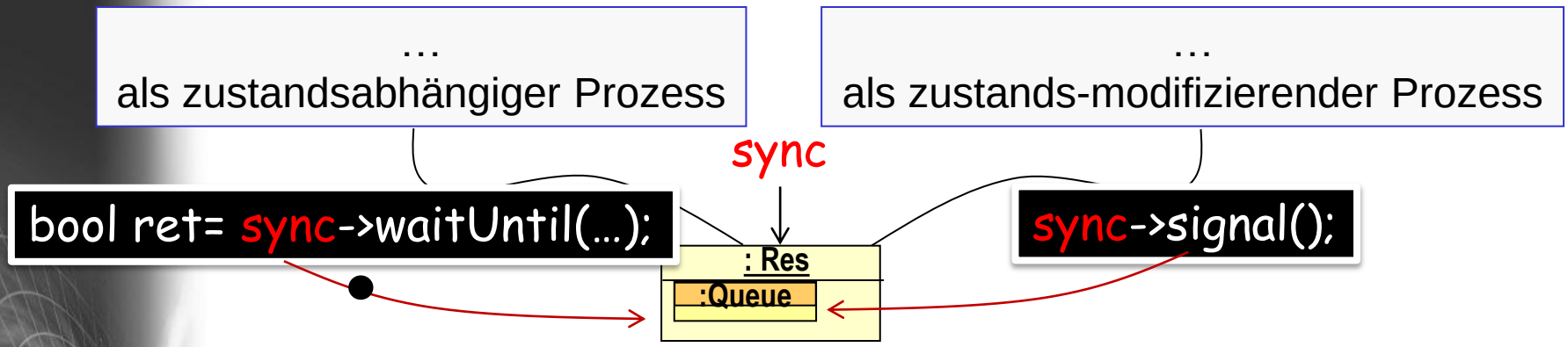

- Der Wartevorgang eines User-Process auf die Verfügbarkeit einer geforderten Anzahl von Token kann unterbrochen werden
  werden (take gibt int-Null zurück)

# 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)

# Zwischenfazit: CondQ-Synchronisation




*Blockierung bei Nichterfüllung der Zustandsbedingung*
*Fortsetzung bei Erfüllung*

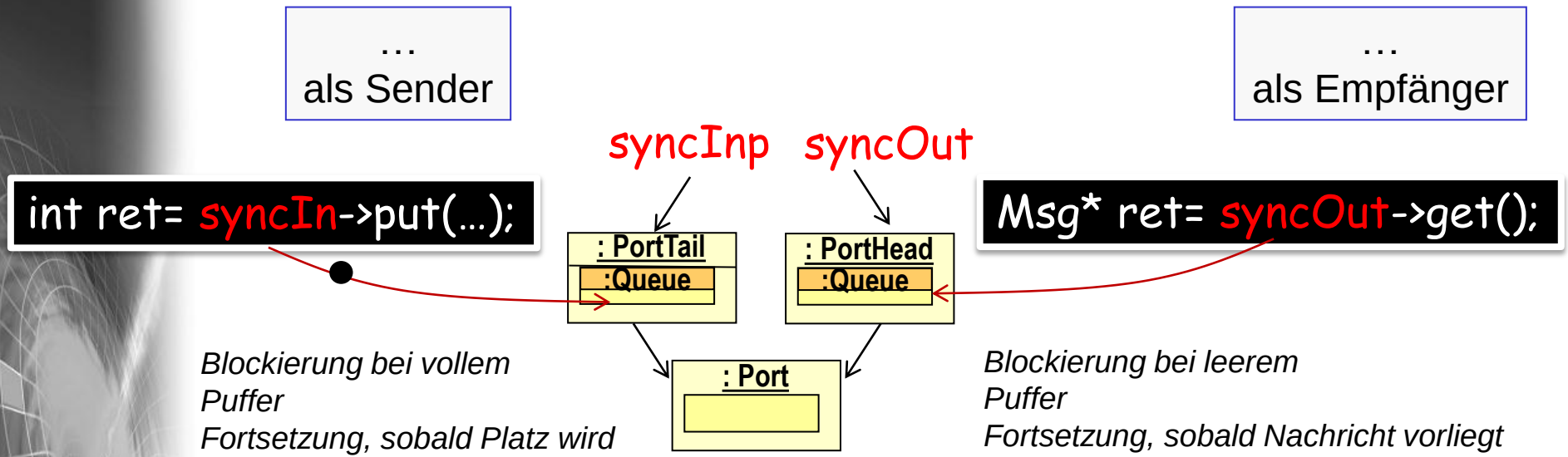
*Aktivierung des ersten blockierten Prozesses*

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

- Typ der Funktion: CONDITION bool xxx ()

# Zwischenfazit: Port-Synchronisation

*- Modus: Blockierung -*




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

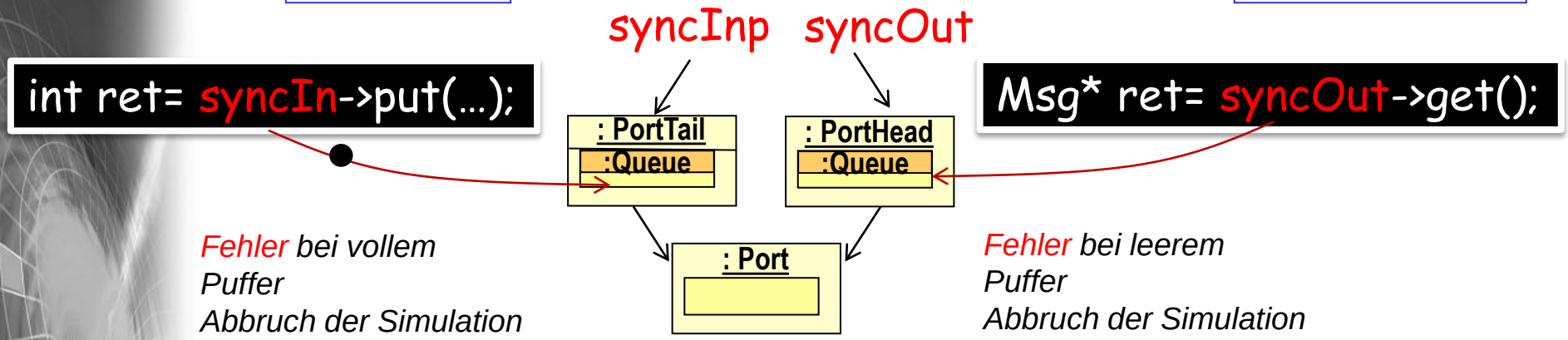
- Der Wartevorgang eines Empfänger-Prozesses kann abgebrochen werden, ohne dass Puffer gefüllt wird (put gibt Null-Zeiger zurück)

# Zwischenfazit: Port-Synchronisation

*- Modus: Fehler -*

…
als Sender

…
als Empfänger



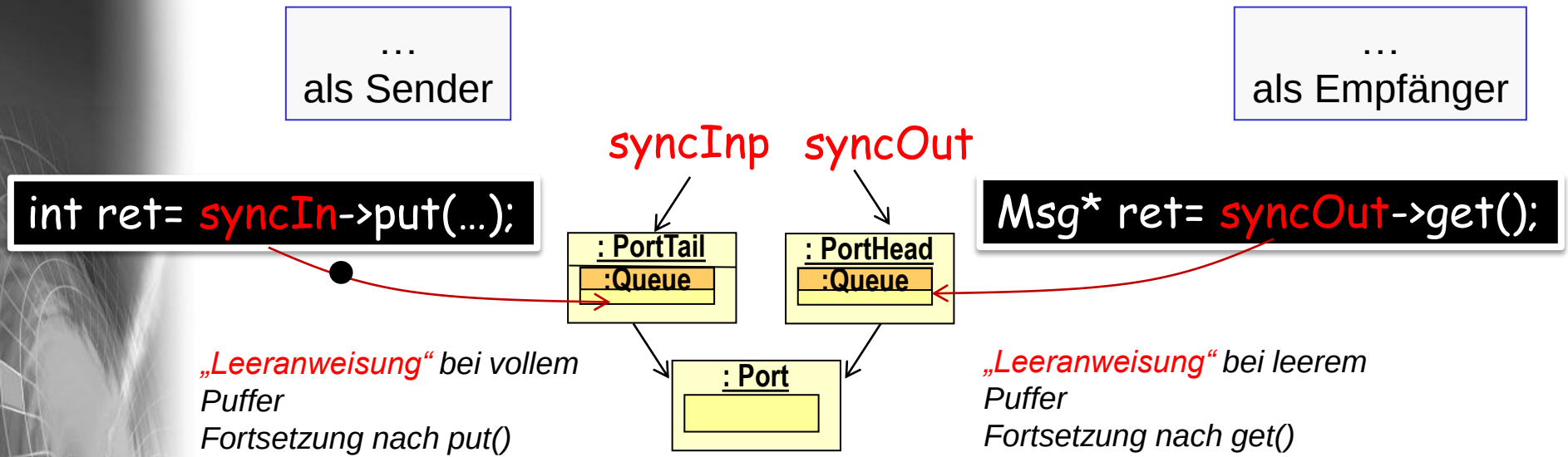

*Fehler bei vollem Puffer*
*Abbruch der Simulation*

*Fehler bei leerem Puffer*
*Abbruch der Simulation*

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

# Zwischenfazit: Port-Synchronisation

*- Modus: Leeranweisung-*



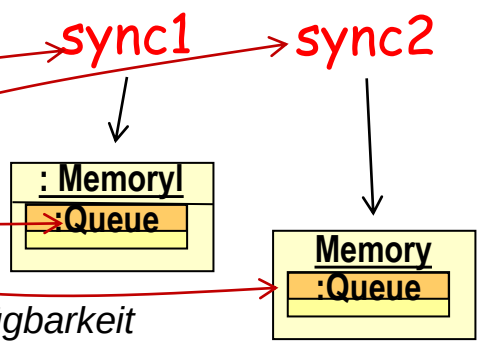

- Es treten weder beim Sender noch beim Empfänger Blockierungen/Verzögerungen ein

# 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)

# 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

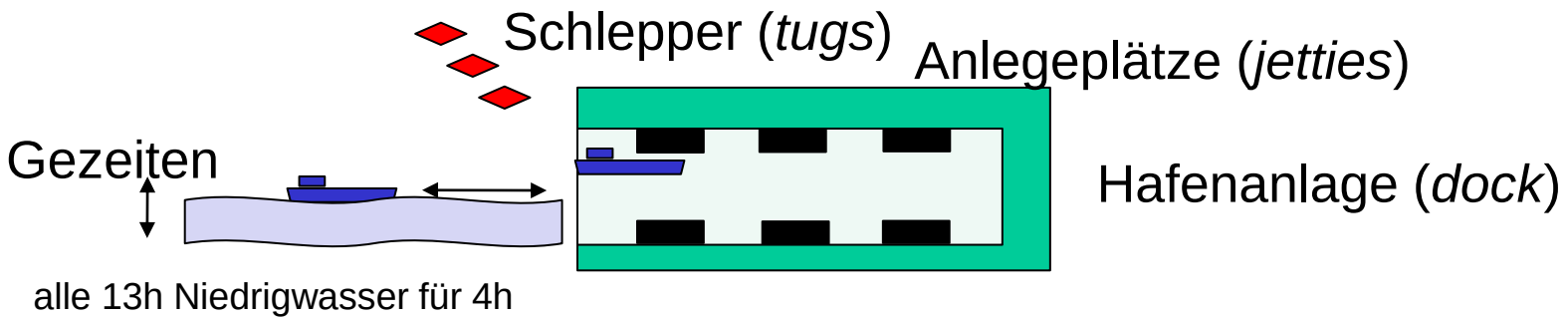
zu übergeben

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

# 7. ODEMx-Modul Synchronisation:
## WaitQ, CondQ

- Konzept WaitQ
  - Beispiel: Tankerflotte, Hafen, Raffinerie
- Konzept CondQ
  - Beispiel: Hafen, Schlepper, Gezeiten
- Weitere Anwendungsbeispiele für WaitQ u. CondQ

# Wortmodell: Hafenabfertigung mit Gezeiten




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

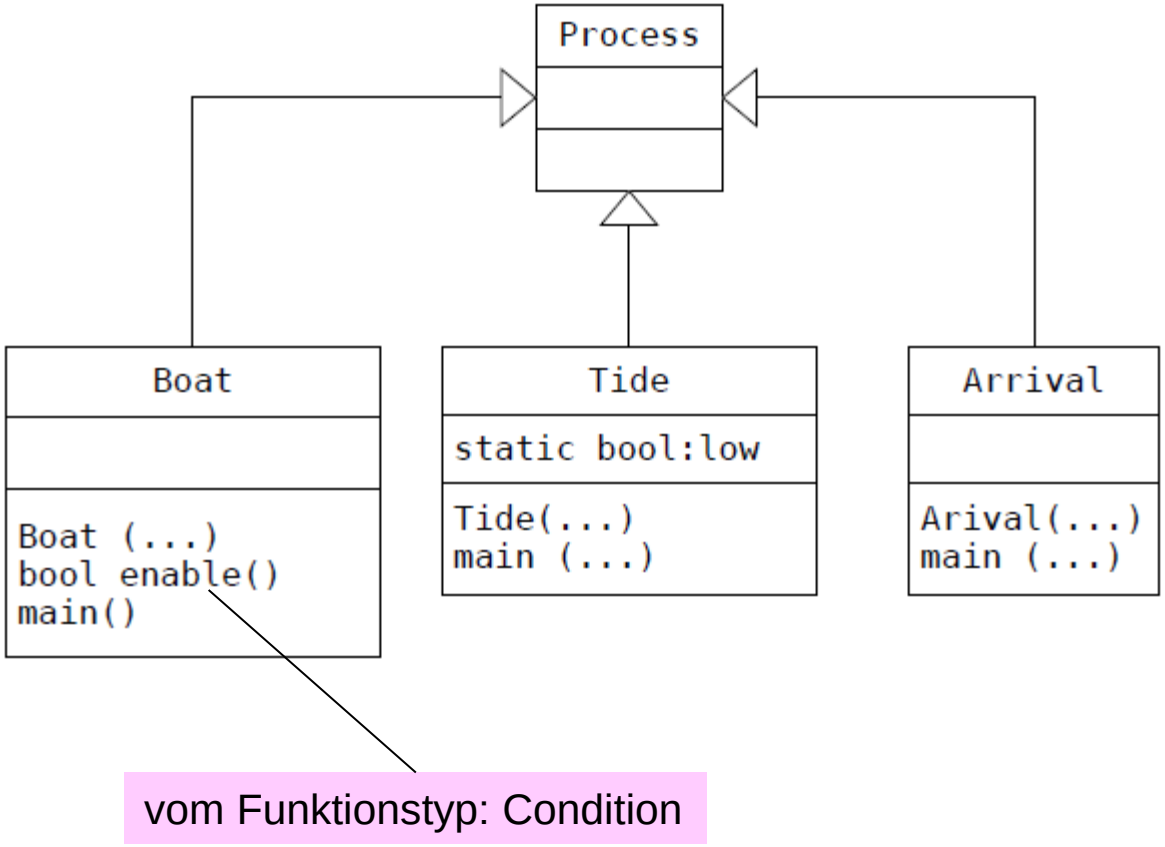
Entladung von Schiffen
Bedingungen/Aktionen:
- stochasitische Entladungszeit

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

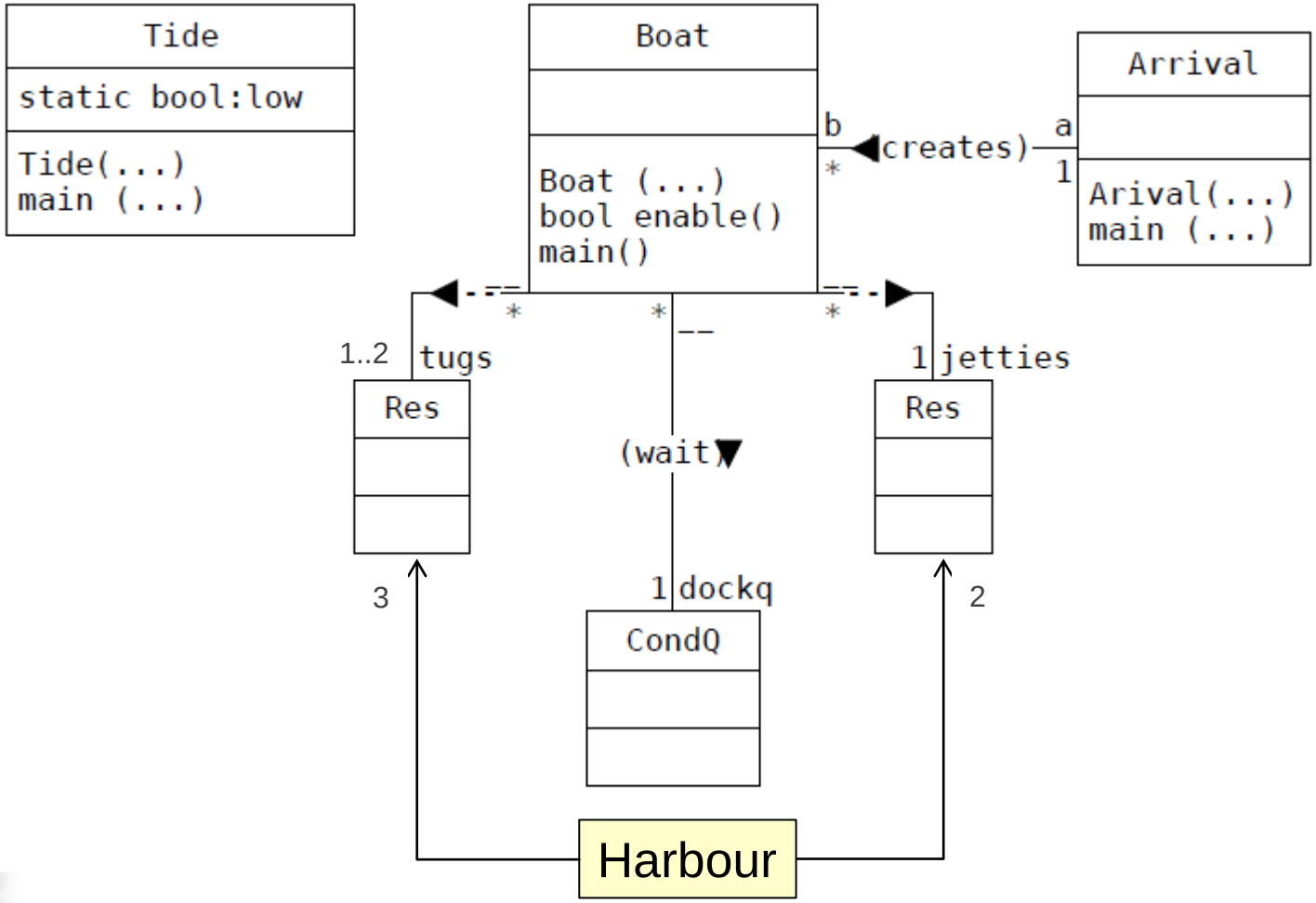
Ziel: simulative Workflow-Nachbildung bei Bestimmung der Auslastung eingesetzter Ressourcen

# Systemstruktur (semiformal)

Strukturkomponenten (Typbeschreibung, Objektkonfiguration)



# Objektkonfiguration



# Verhalten von Boat

```
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();
};
```

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

```
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;
}
```

# 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);
        a->activate();
        t->activateIn(1.0);

        sim->runUntil(50.0);
        sim->pauseTrace();
        sim->runUntil (28.0*24.0);

        report.generateReport();
        sim->stopTrace();
        sim->exitSimulation();
        delete a;
        delete t;
        delete next;
        delete discharge;
        delete tugs;
        delete jetties;
        delete dockq;

        return 0;
}
```

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

**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 |

# 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 |

# 7. ODEMx-Modul Synchronisation:

## WaitQ, CondQ

- Konzept WaitQ
  - Beispiel: Tankerflotte, Hafen, Raffinerie
- Konzept CondQ
  - Beispiel: Hafen, Schlepper, Gezeiten
- Weitere Anwendungsbeispiele für WaitQ u. CondQ

# 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

# Beispiel: Transportfahrzeuge – Bagger – Beladung

große Lastwagen
20 t

22 /h

*Ankunftsraten*

10 /h

kleine Lastwagen
5 t

240 t/h

60 t/h

*Beladungsraten*

kleinerBagger

großerBagger-1

großerBagger-2

**Beladung**

- große Lastwagen bevorzugt von großen Baggern
- kleine Lastwagen **nur** vom kleinen Bagger
- sofortige Unterbrechung der Beladung eines großen Lastwagens durch kleinen Bagger, sobald kleiner Lastwagen eintrifft oder großer Bagger frei wird
- priorisierte Beladung unterbrochener Fahrzeuge

**Ziel**: 10h -Simulation

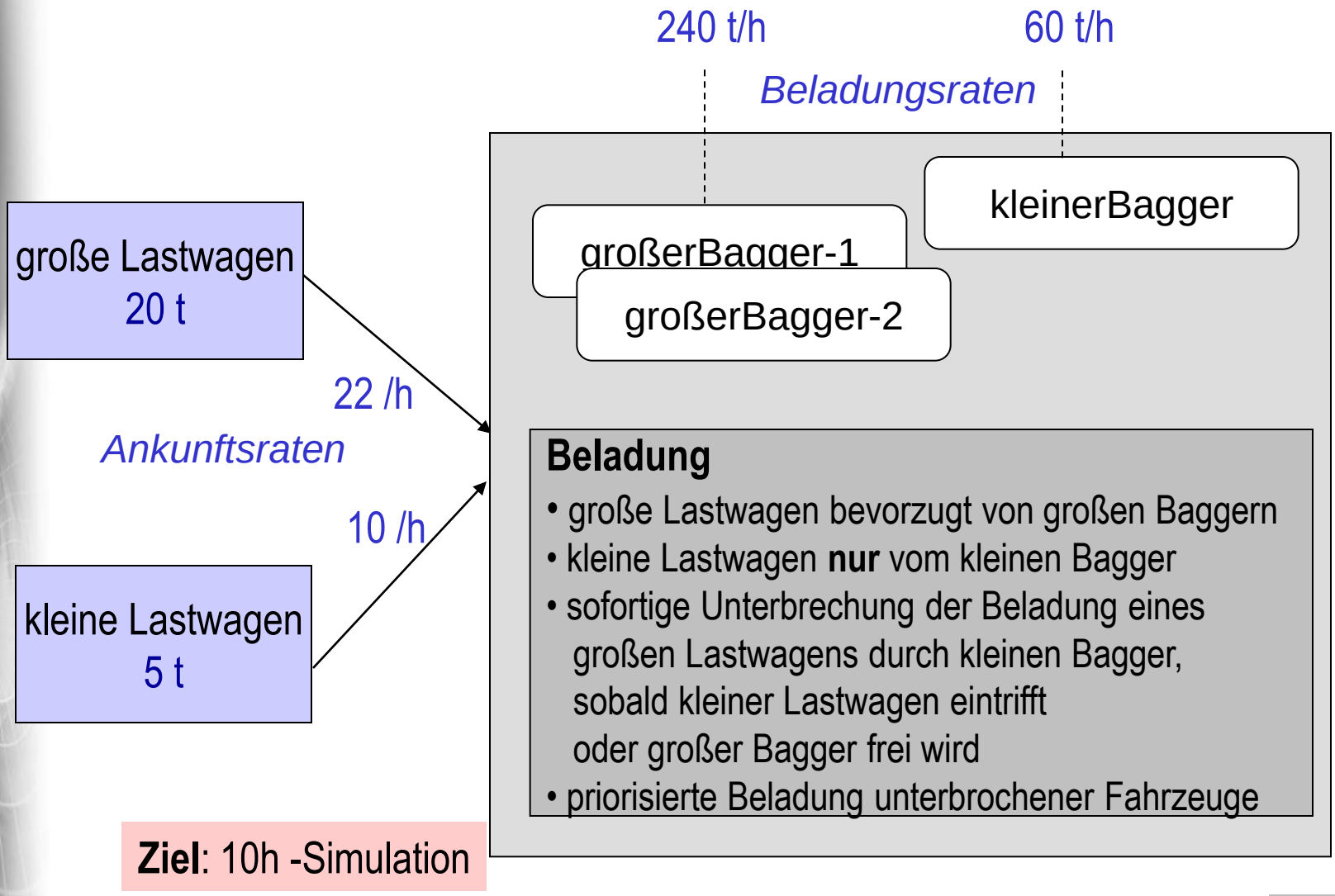
# *Typische Probleme*

- 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

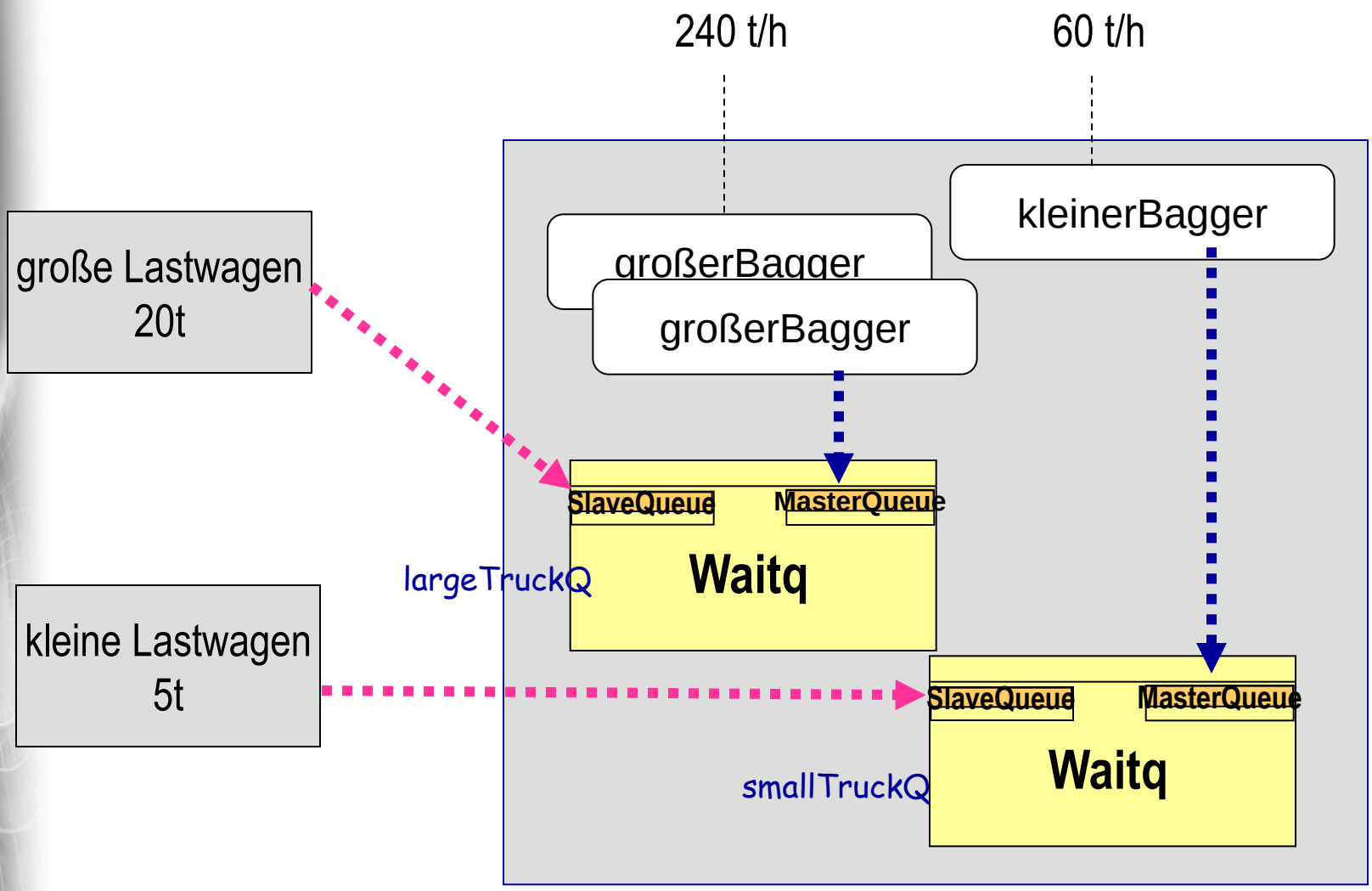
# Beispiel: Transportfahrzeuge – Bagger – Beladung



# Lösungsvariante ausschließlich mit Waitq

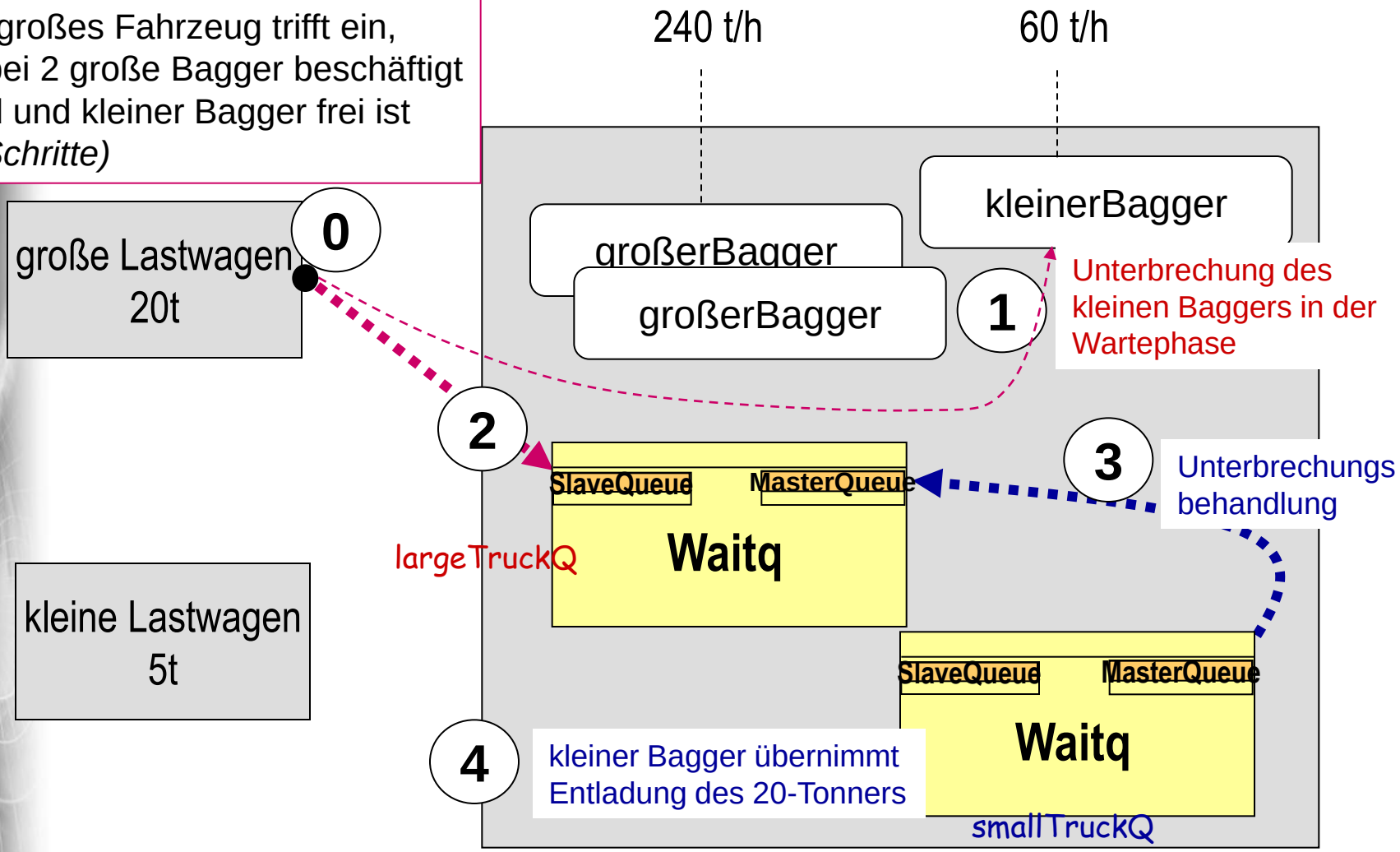
→33

# Master-Slave-Rollen bei der Beladung



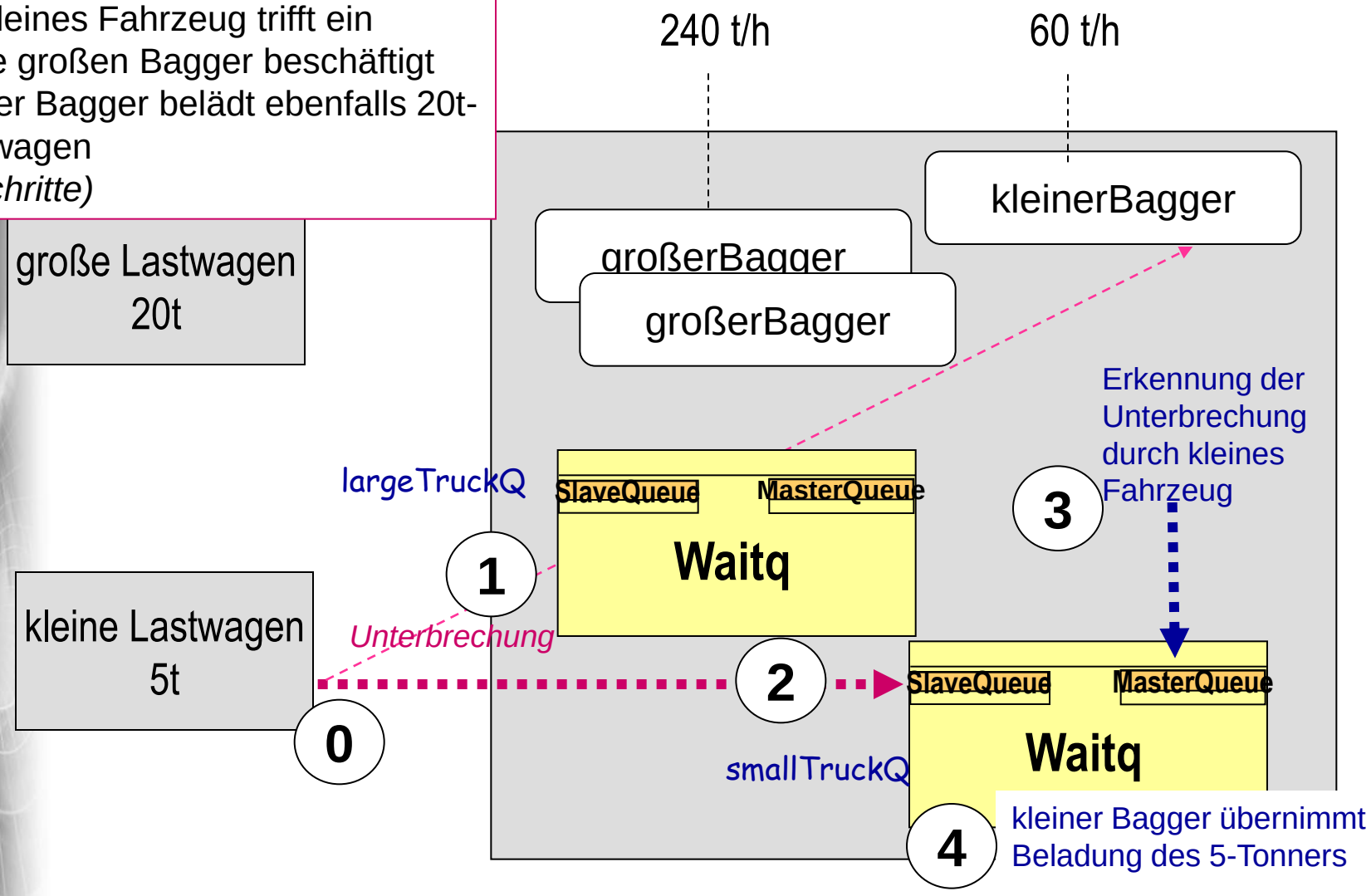
# Einsatz des kleinen Baggers für große Fahrzeuge

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



# Einsatz des kleinen Baggers für kleine Fahrzeuge

**2. Fall:**
ein kleines Fahrzeug trifft ein
beide großen Bagger beschäftigt
kleiner Bagger belädt ebenfalls 20t-Lastwagen
*(4 Schritte)*



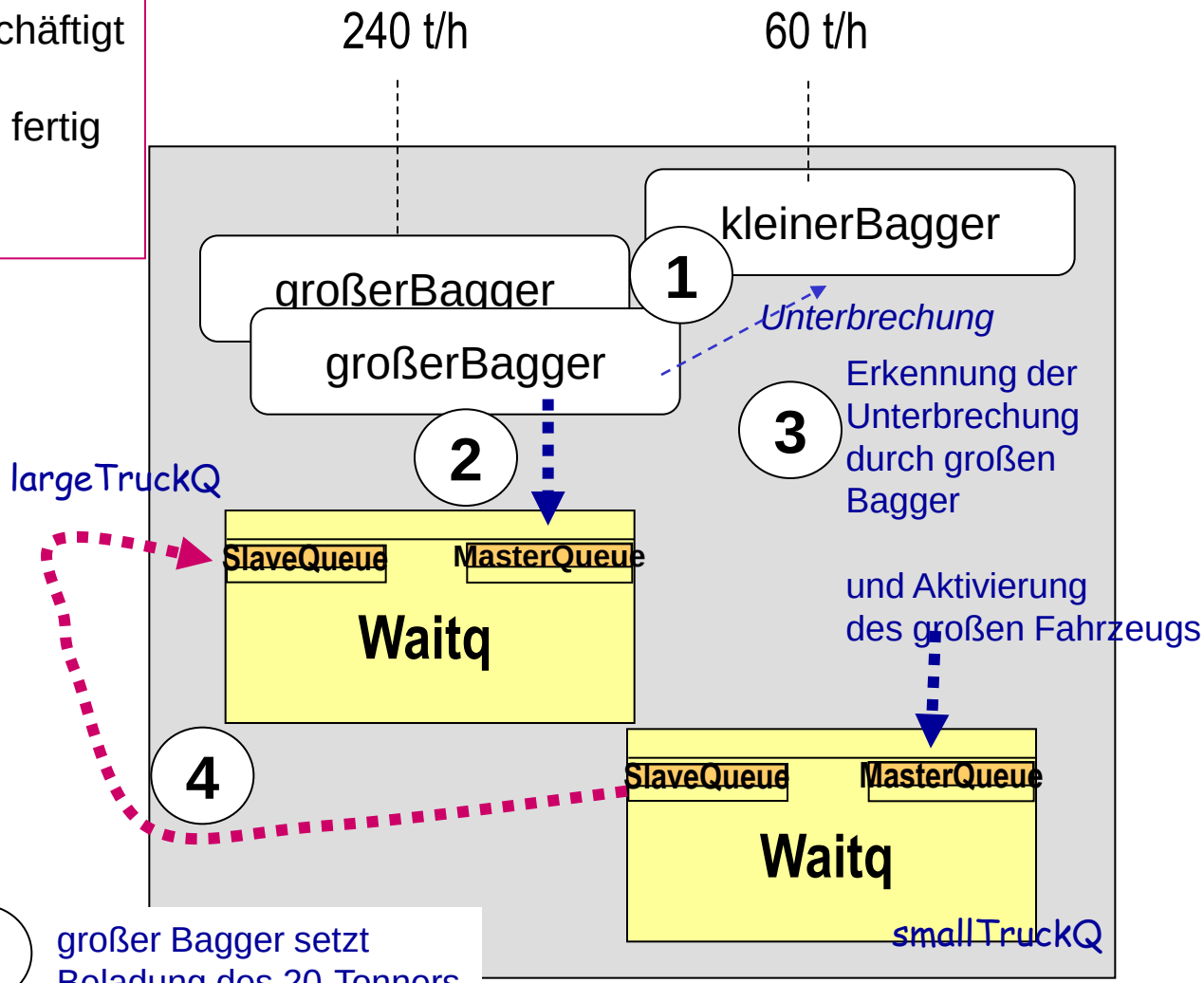
# Einsatz des großen Baggers für große Fahrzeuge

**3. Fall:**
beide großen Bagger sind beschäftigt
kleiner Bagger entlädt 20t
→ jetzt wird ein großer Bagger fertig

*(5 Schritte)*

große Lastwagen
20t

kleine Lastwagen
5t



# Grobgranulare Zustände des kleinen Baggers

- FREE
  wartet als Master in smallTruckQ auf kleine Fahrzeuge

- SMALL
  entlädt kleine Fahrzeuge

- LARGE
  entlädt 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());
      //Entladung
       holdFor (myTruck->maxload /loadRate);
       myTruck->load = maxload;

       // Freigabe des Fahrzeugs
       myTruck->hold();
    }
   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 – SmallTruck

SmallDigger-Objekte als Master

```
int SmallDigger::main() {
      double unloadStart= 0; // Entladungsbeginn
      for (;;) {
        workSt= FREE;
        if (smallTruckQ->avail()) {
              // unterbrechungsfreie Entladung
            // 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;
}
```

SmallTruck-Objekte als Slave

```
int SmallTruck::main() {
     // 2.Fall
     if (smallDigger->workStatus() == LARGE)  {
        //Unterbrechung des kleinen Baggers
        smallDigger->interrupt();
     }
     // zeitloses Warten auf kleinen Bagger
     smallTruckQ->wait();
     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;
}
```

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;
}
```