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

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

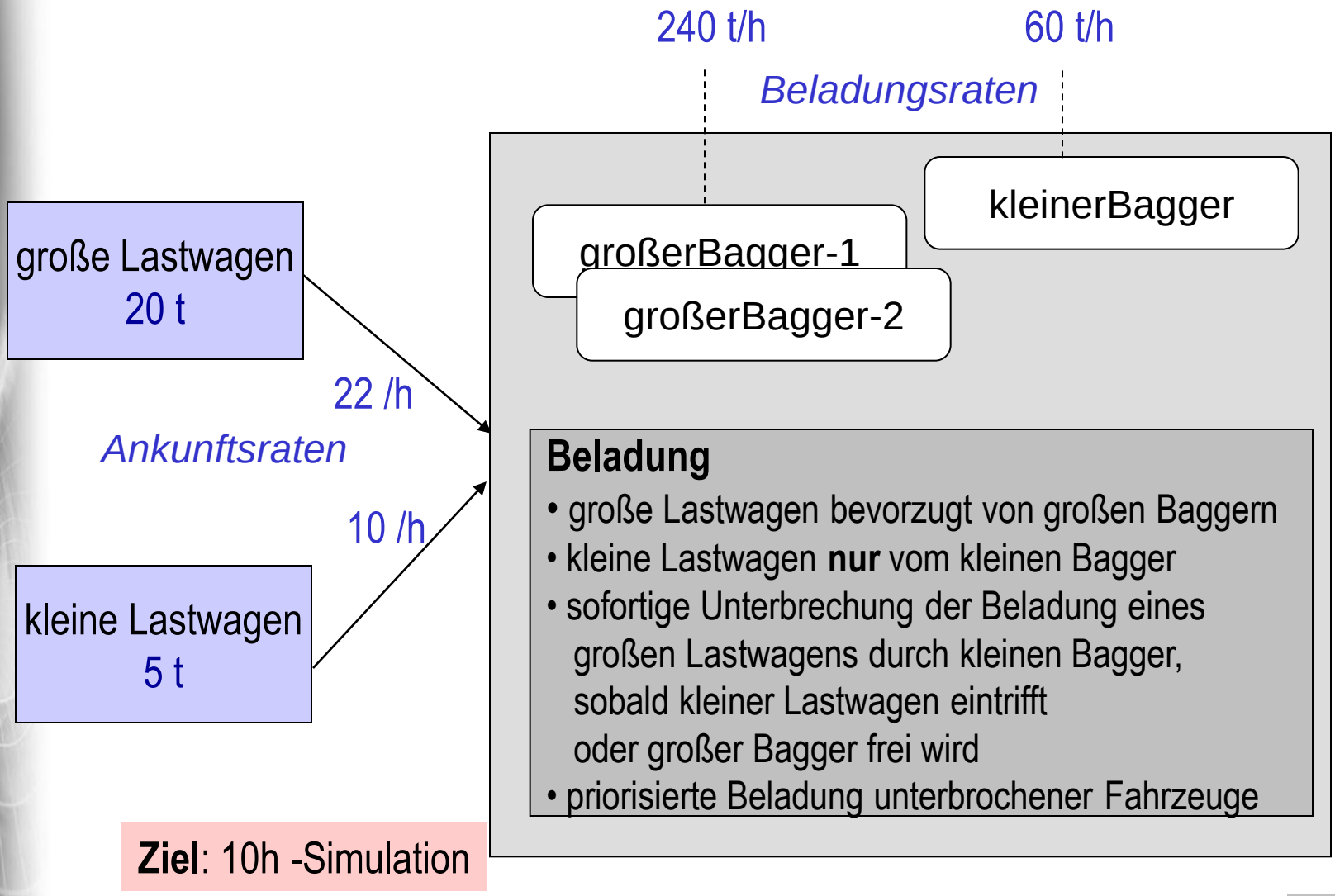
Typische Probleme

- 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



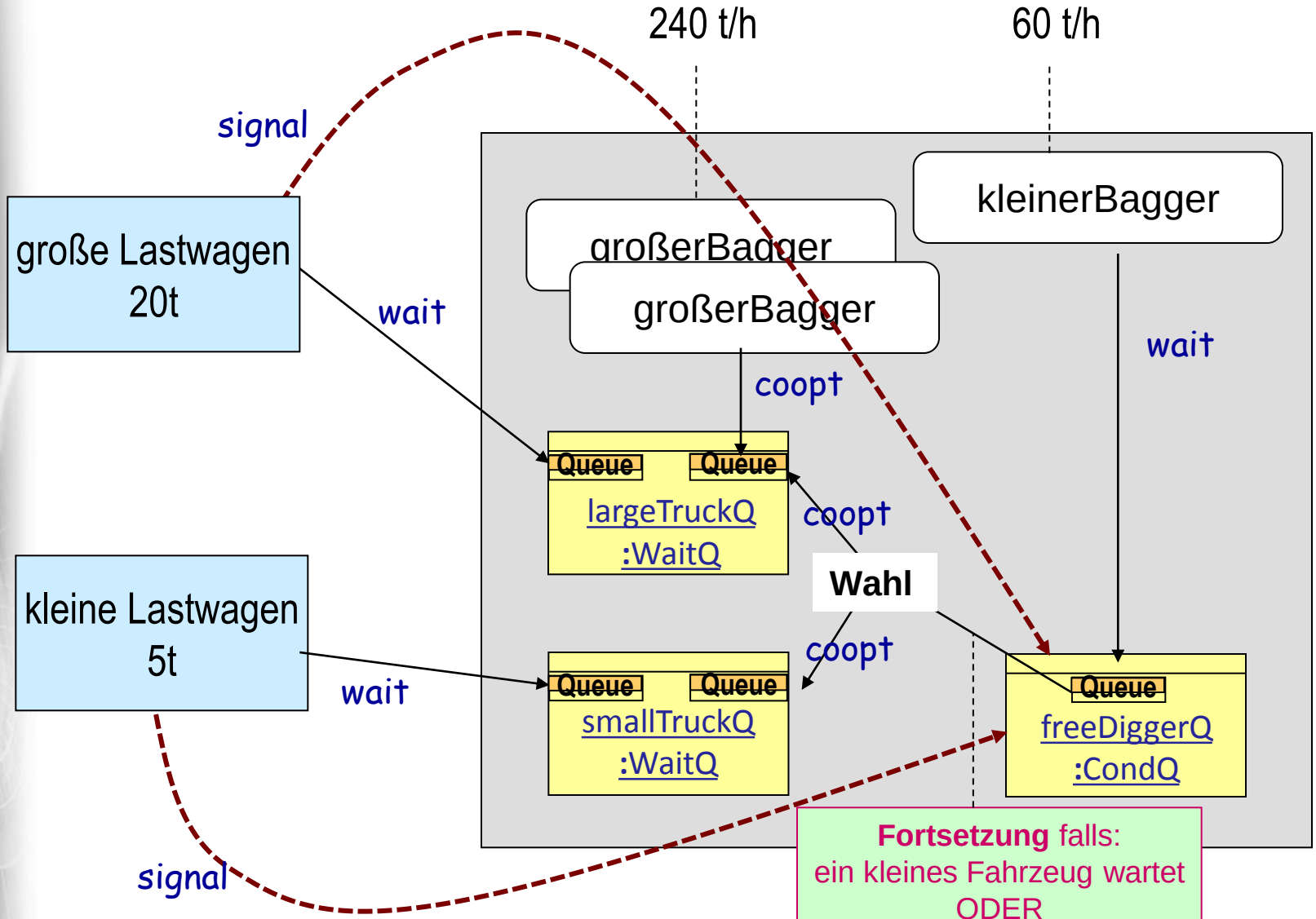
Typische Probleme

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

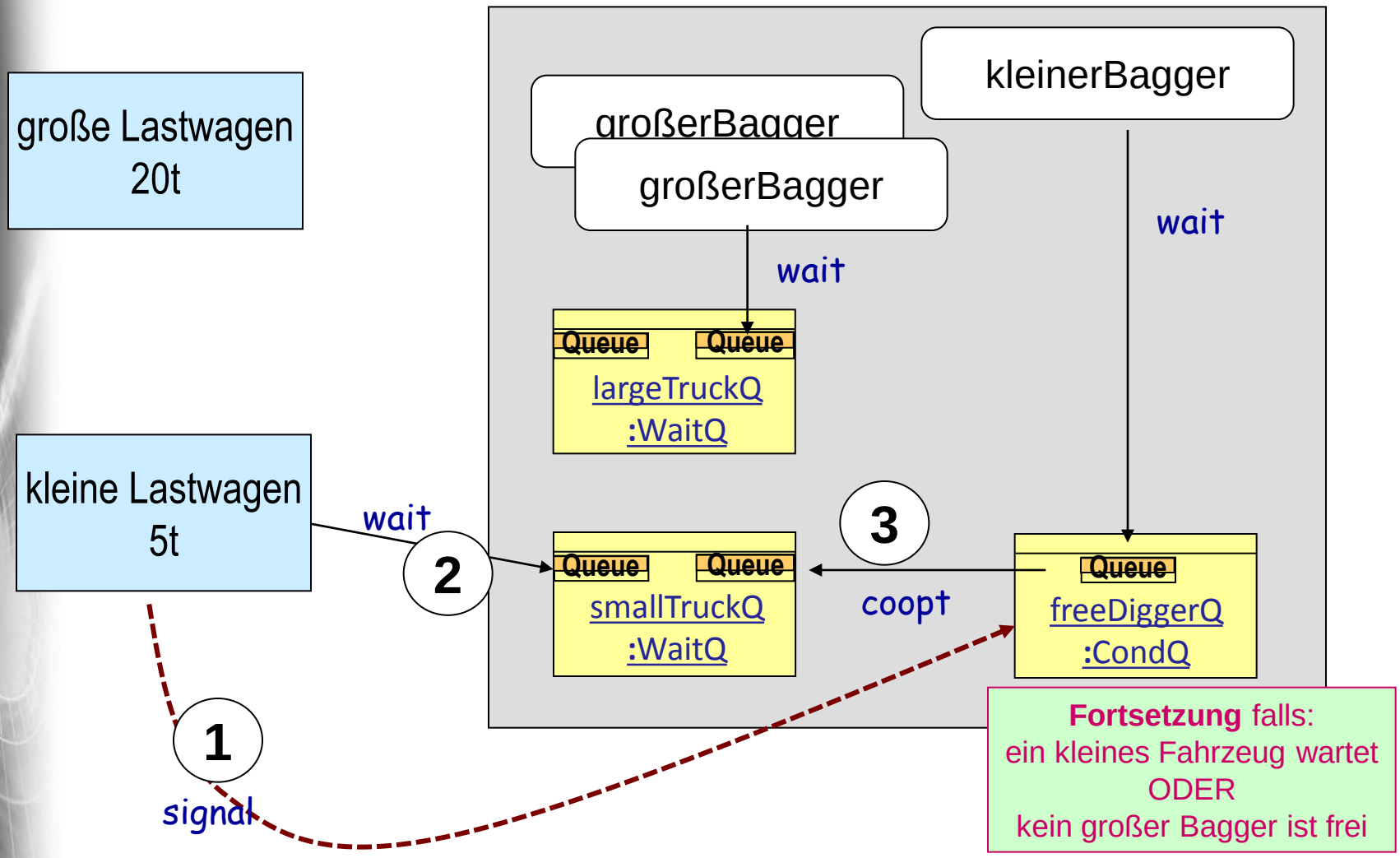
- dynamischer Austausch der eingesetzten Ressourcen (bei gegebener Verfügbarkeit) um Effizienz des Arbeitsganges zu erhöhen

- Unterbrechung von Ressourcennutzungen

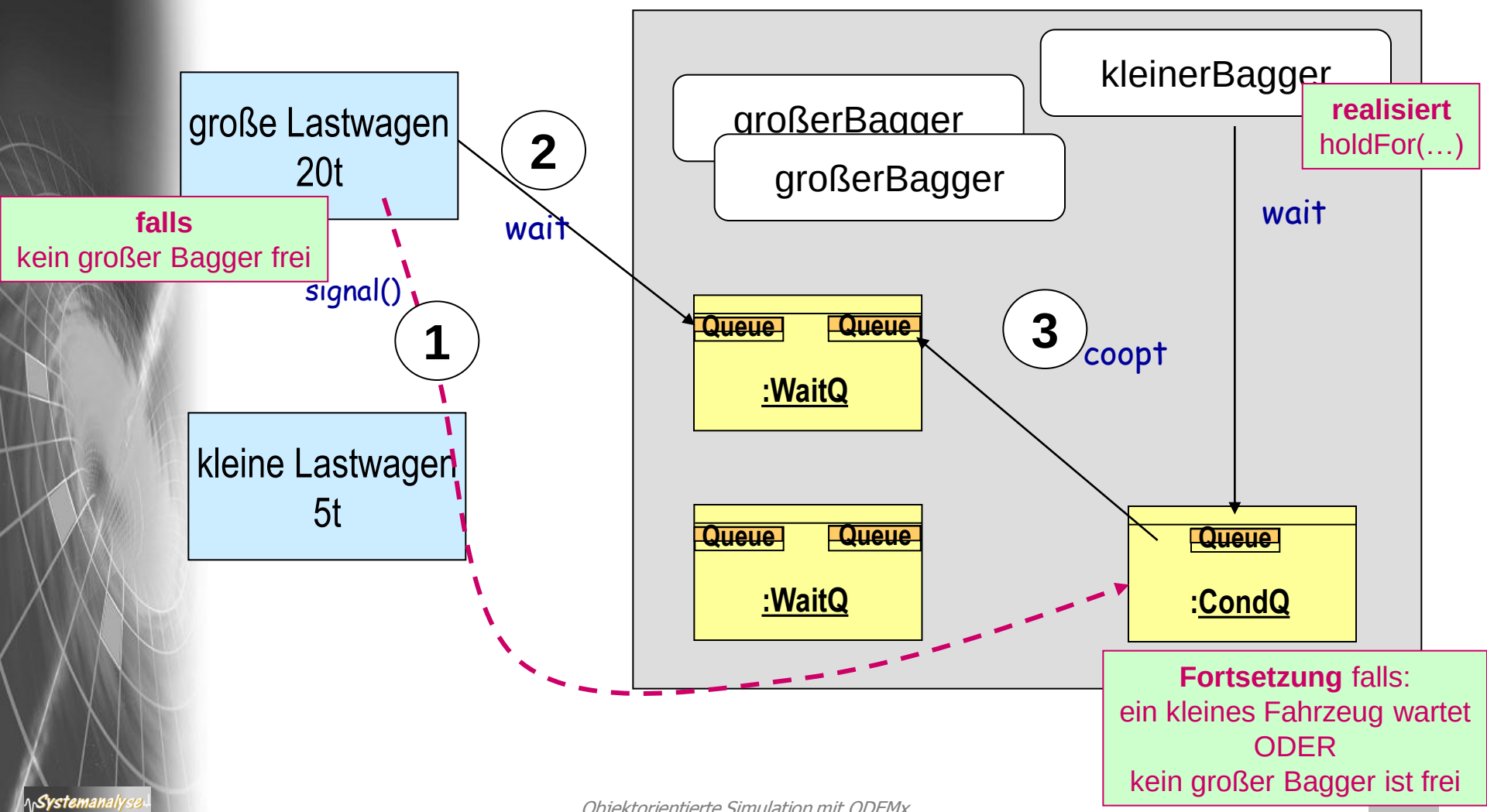
Lösungsansatz



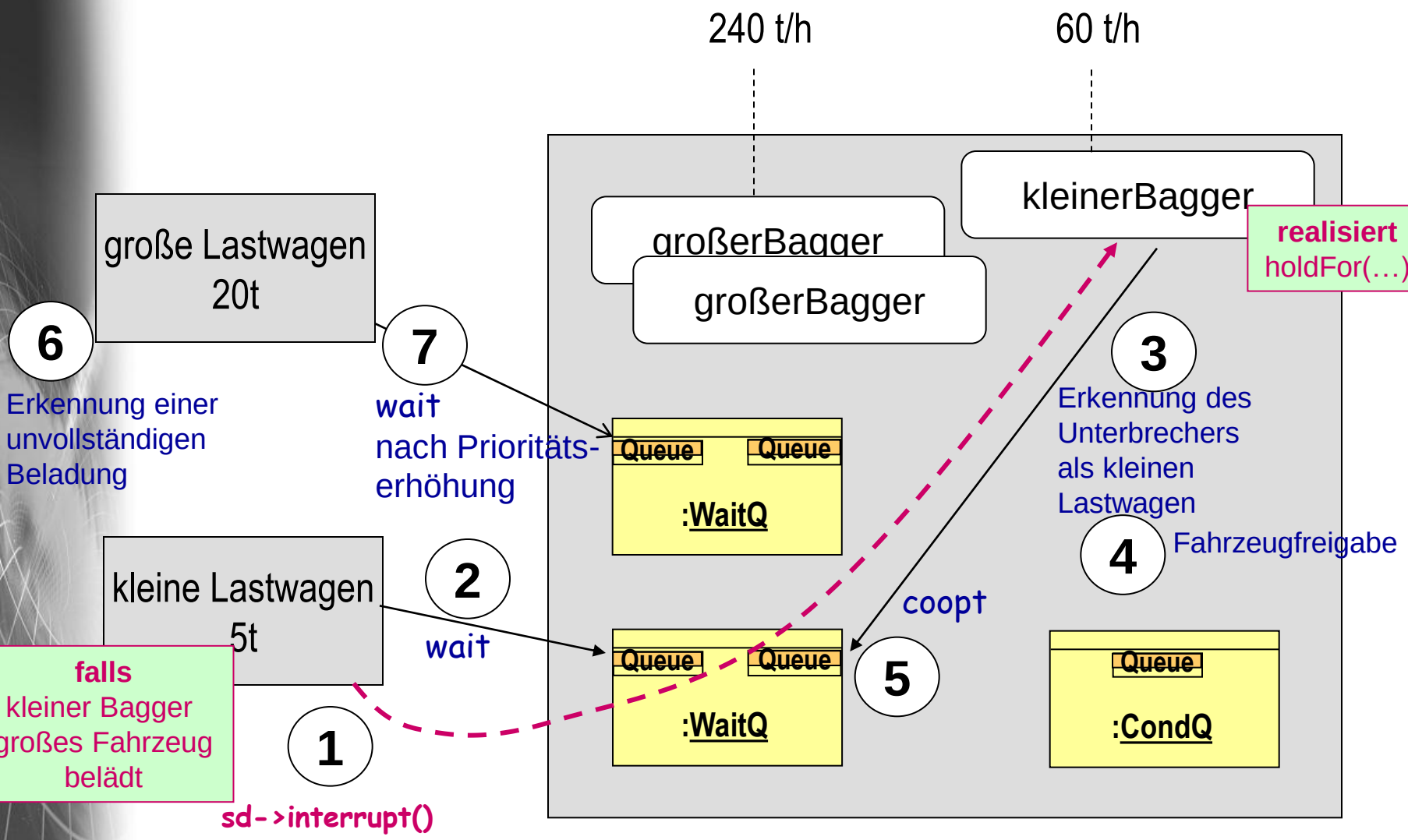
Eintreffen eines 5-Tonners



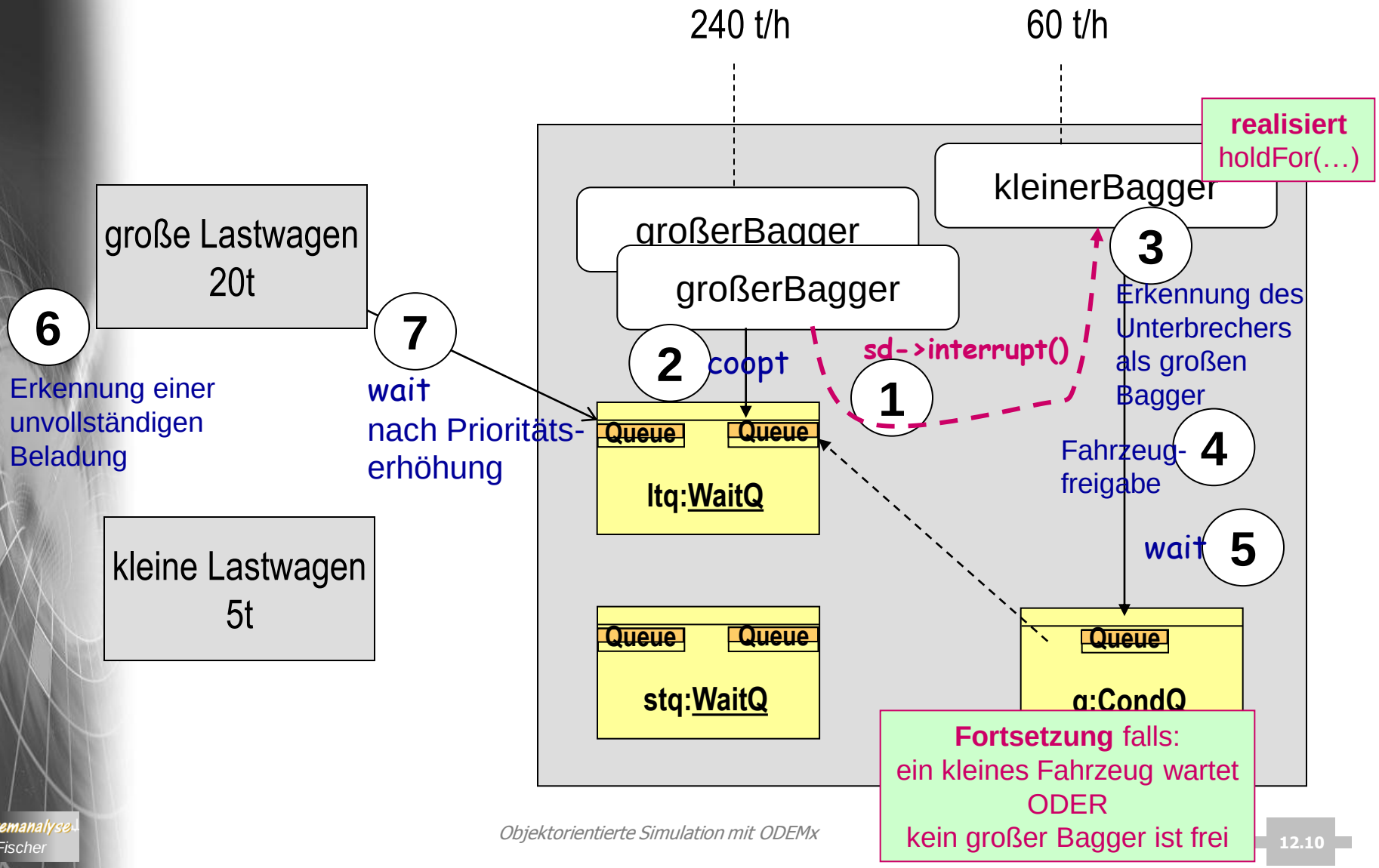
Eintreffen eines 20-Tonners



Kleiner Bagger belädt großes Fahrzeug + Eintreffen eines kleinen Lastwagens



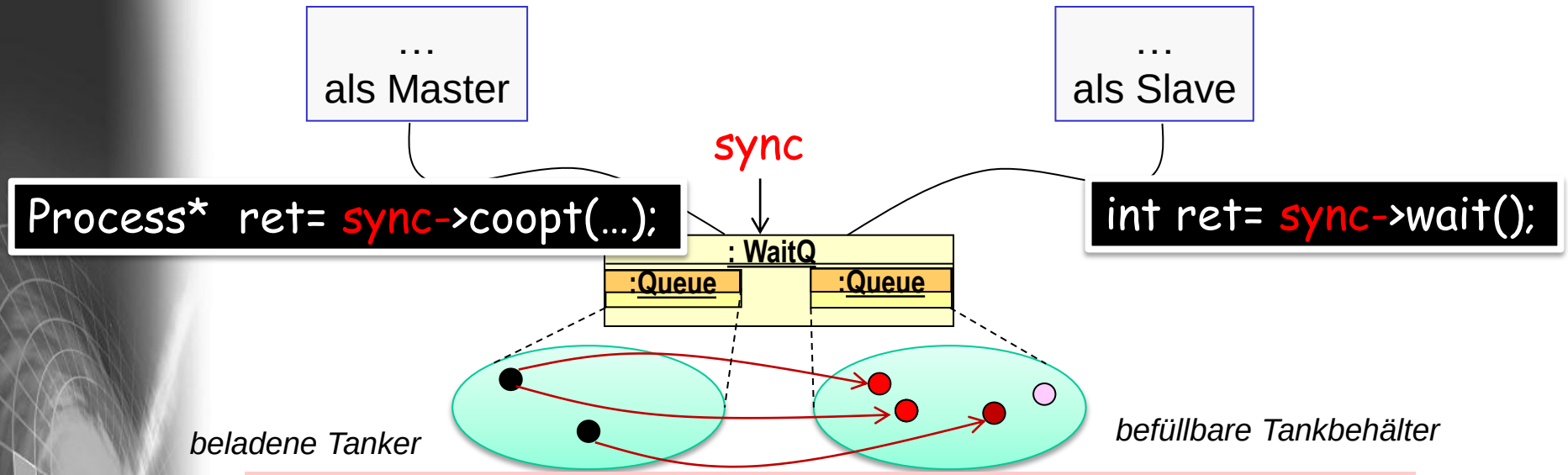
Kleiner Bagger belädt großes Fahrzeug + großer Bagger wird fertig



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

Zwischenfazit: Master-Slave-Synchronisation

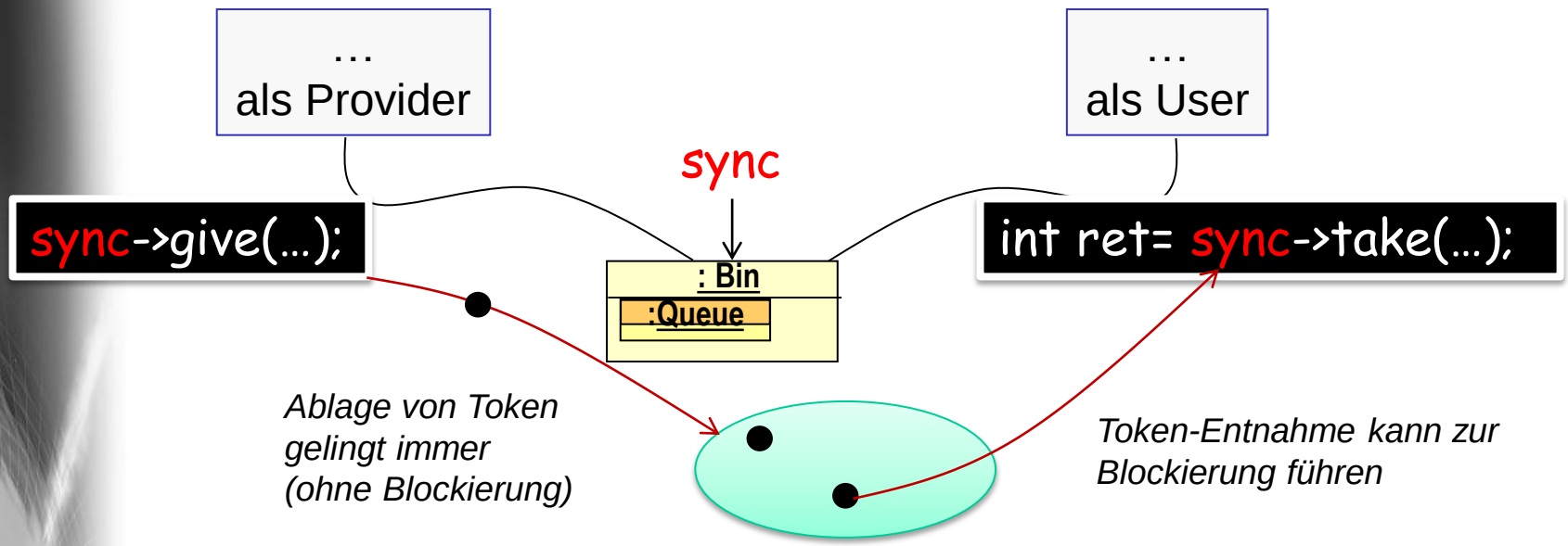


1. Selektion-Funktion kann je Master(typ) verschieden sein,
   sie ist in der jeweiligen Prozessableitung zu definieren
2. Eine Prozessableitung kann mit mehreren Funktionen des Selektion-Typs ausgestattet sein,
   bei jedem Coopt-Ruf kann eine andere Selektion vorgenommen werden

- 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)

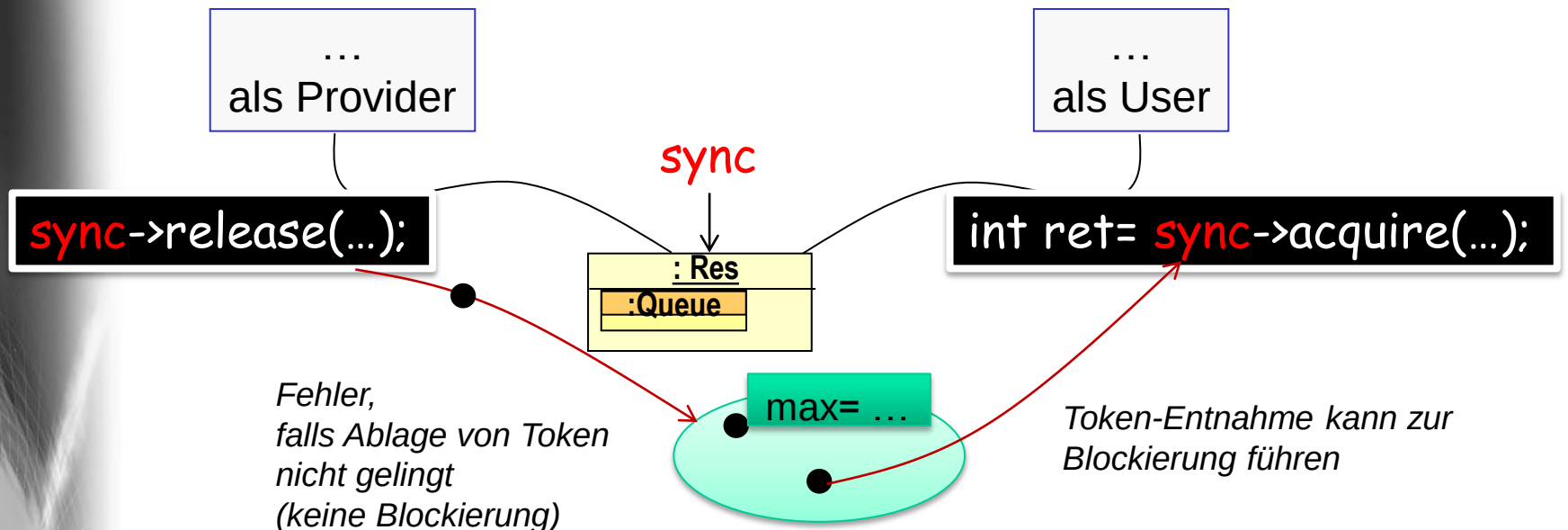
- Typ der Funktion xxx: SELECTION bool xxx (Process *p)

Zwischenfazit: Bin-Synchronisation



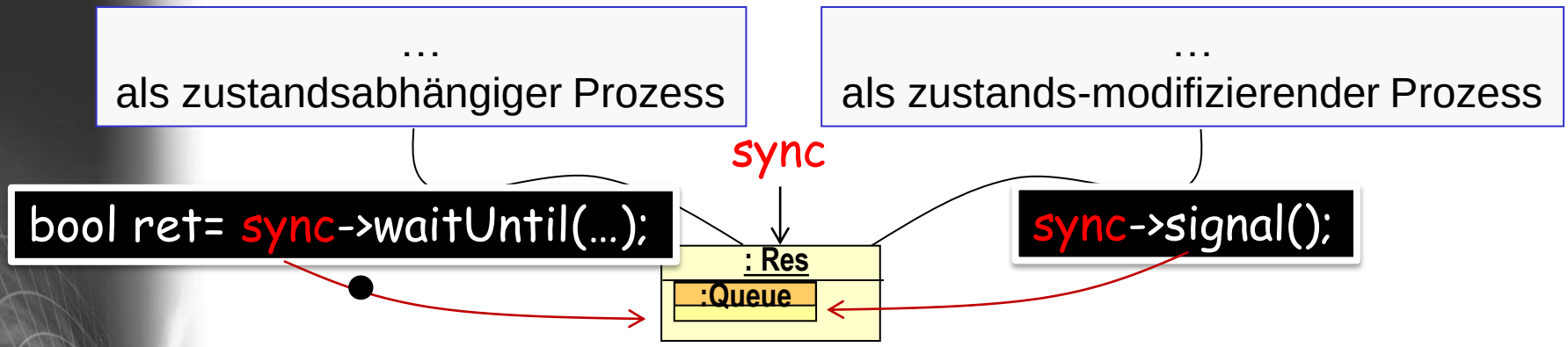
- Der Wartevorgang eines User-Process auf die Verfügbarkeit einer geforderten Anzahl von Token kann unterbrochen werden
  werden (take gibt int-Null zurück)

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

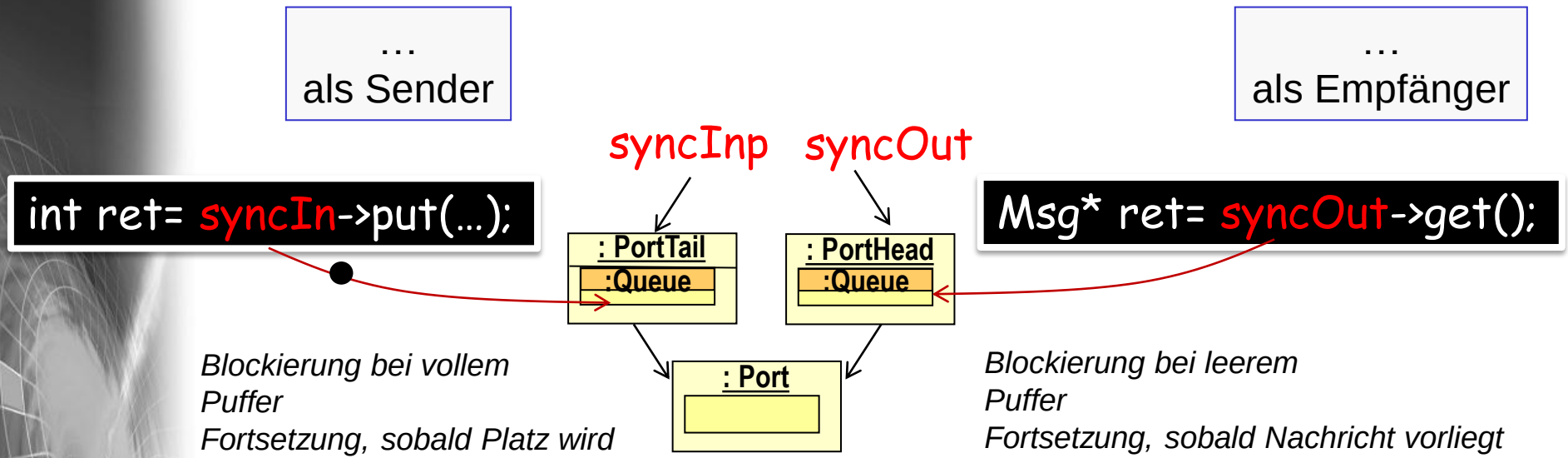
Aktivierung des ersten blockierten Prozesses

- 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)

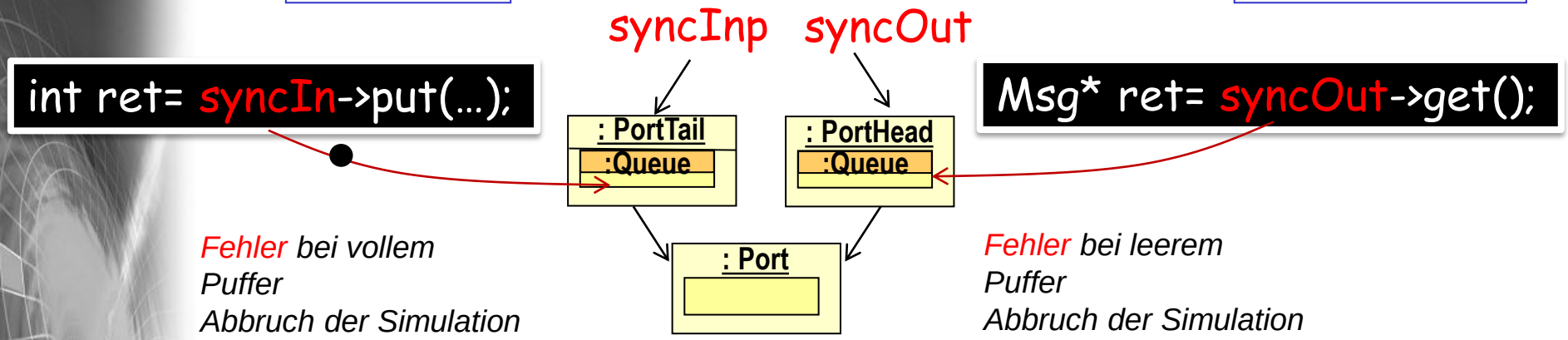
- Der Wartevorgang eines Empfänger-Prozesses kann abgebrochen werden, ohne dass Puffer gefüllt wird (put gibt Null-Zeiger zurück)

Zwischenfazit: Port-Synchronisation

- Modus: Fehler -

…
als Sender

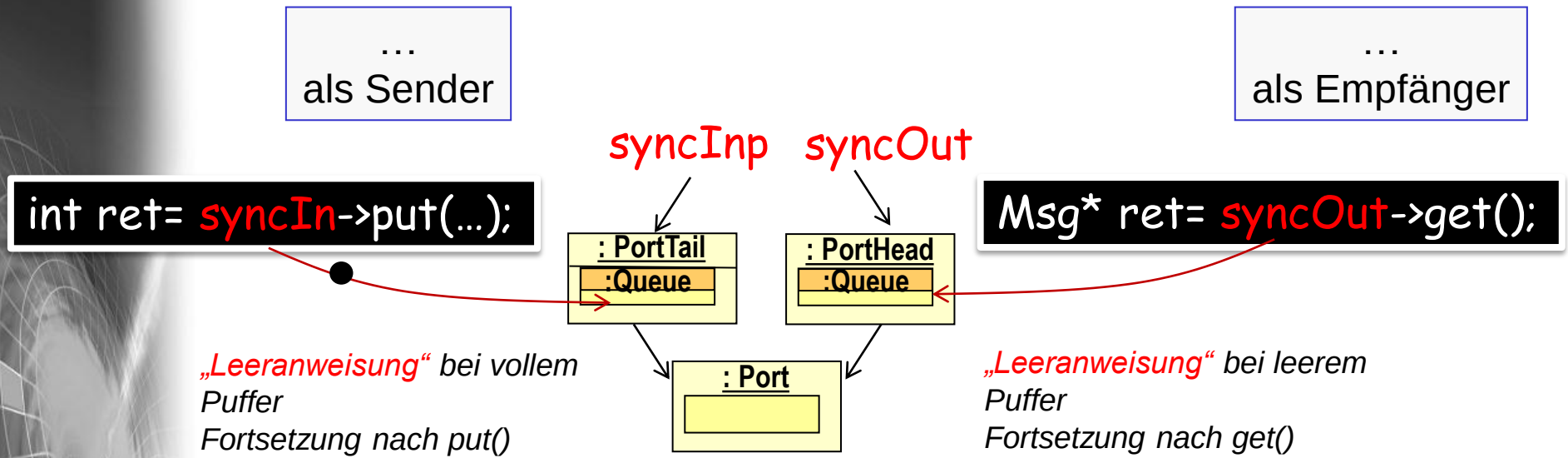
…
als Empfänger



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

Zwischenfazit: Port-Synchronisation

- Modus: Leeranweisung-



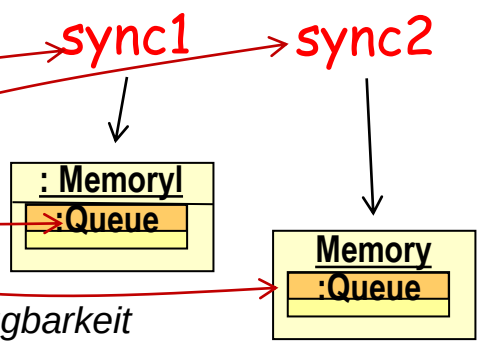
- Es treten weder beim Sender noch beim Empfänger Blockierungen/Verzögerungen ein

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