Kurs OMSI
im WiSe 2011/12

Objektorientierte Simulation mit ODEMx

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

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

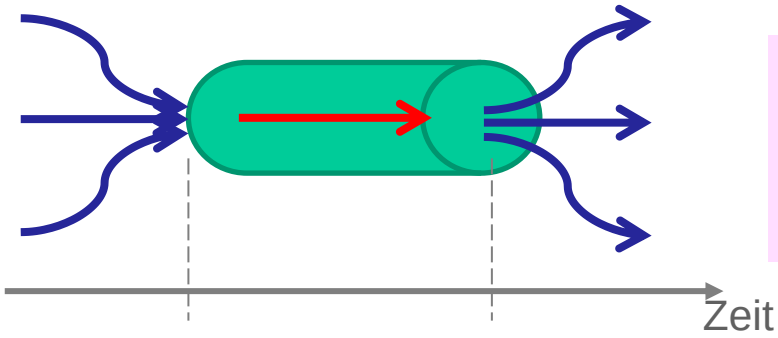
6. ODEMx-Modul Synchronisation:
WaitQ, CondQ

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

Nützliches Modellierungsmuster

- n+1 (>=2) Prozesse kooperieren ab einem Zeitpunkt für eine gewisse Dauer

  Bed.: (1) Zum Startzeitpunkt der Kooperation sind alle n+1 Prozesse verfügbar/für die Kooperation bereit
  (2) Zustandsänderungen der Prozesse sind voneinander abhängig)



Entschärfung der Parallelität
von synchronen Wechselwirkungen
bei Zustandsänderungen
im Simulator

- Effiziente simulative Umsetzung auf einer Ein-Prozessor-Maschine
  - einer der n+1 Prozesse übernimmt als **Master** (aktiv) die Ausführung der Zustandsänderungen sämtlicher Prozesse in Abhängigkeit der Modellzeit
  - alle anderen n Prozesse warten als **Slave** (passiv) auf die Beendigung der Kooperation durch den **Master**

ACHTUNG: Master und Slave sind Rollen, die Prozesse zeitweilig spielen

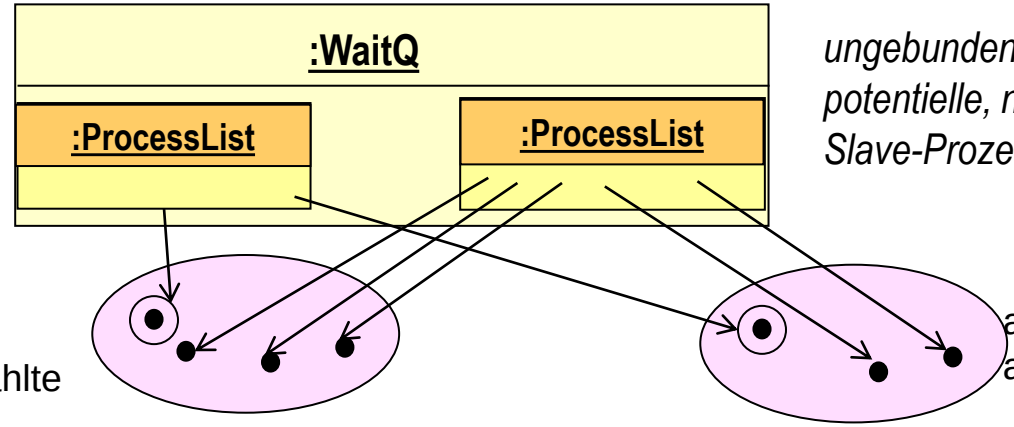
WaitQ-Konzept

Synchronisationsklasse

zur Erfassung von Prozessen und Bildung zeitweiliger Kooperationsgemeinschaften mit unterbrechbarem Warten auf das Zustandekommen der Kooperation, falls Kooperationspartner momentan nicht zur Verfügung stehen

- jeweils **einem** Master lassen sich beliebig **viele** Slave-Prozesse zuordnen
- **Master** bestimmt **allein** die Dauer der Kooperationsleistung (und gibt danach die Slaves, i. allg. gleichzeitig, wieder frei)
- **Master** realisiert **allein** die entsprechenden Zustandsänderungen, die mit der Kooperation aller Partner verbunden sind (benötigt entsprechende Zugriffsrechte auf seine Slaves)

ungebundene potentielle, noch blockierte Master-Prozesse



ungebundene potentielle, noch blockierte Slave-Prozesse

aktiver Master verwaltet temporär ausgewählte Slaves

anderer aktiver Master, andere Slaves

Weitere Anforderungen an WaitQ

1. über ein WaitQ-Objekt sollen sich

   gleichzeitig / nacheinander

   beliebig viele temporäre Master-Slave-Ensemble bilden können

2. folgende Teilaktivitäten bei Nutzung eines WaitQ-Objektes sollen extern (z.B. Timer) vorzeitig **unterbrechbar** sein:
   - Warten eines Prozesses als Master auf die Verfügbarkeit eines Slaves
   - Warten eines Prozesses als Slave auf die Verfügbarkeit eines Masters
   - Erbringung der laufenden Kooperationsleistung (Zustandsänderungen) des Masters

3. ein Master sollte über ein waitQ-Objekt die Verfügbarkeit eines Slaves mit bestimmten Eigenschaften fordern können
   - bestimmter Prozesstyp
   - bestimmte Attribut-Belegungen (Zustand)

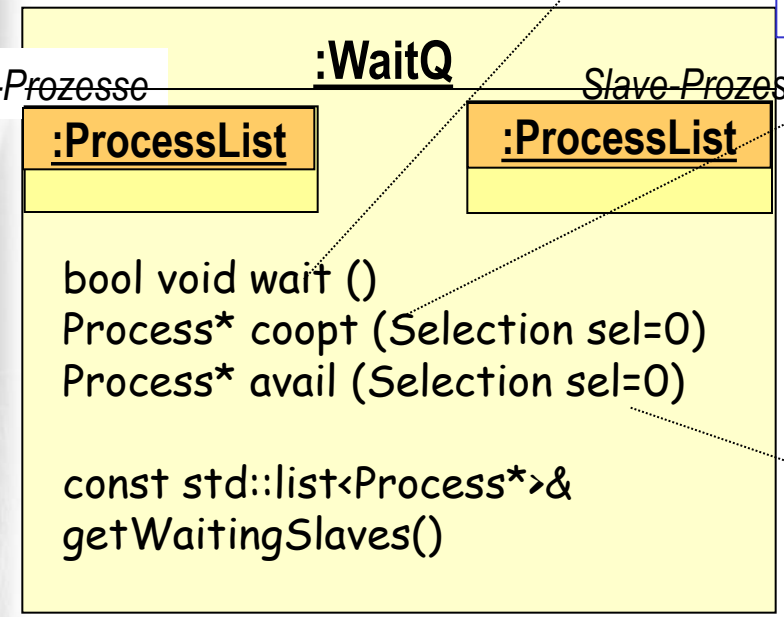
WaitQ- Member-Funktionen

```
                    WaitQ (base::Simulation &sim, const data::Label &label, WaitQObserver *obs=0)
                                        // Construction for user-defined Simulation.
                    ~WaitQ ()
                                        // Destruction.
const base::ProcessList &   getWaitingSlaves () const
                                        // List of blocked slaves.
const base::ProcessList &   getWaitingMasters () const
                                        // List of blocked masters.
// Master-slave synchronisation
                    bool    wait ()
                                        // Wait for activation by a 'master' process.
                    bool    wait (base::Weight weightFct)
                                        // Wait for activation by a 'master' process.
          base::Process *   coopt (base::Selection sel=0)
                                        // Get a 'slave' process.
          base::Process *   coopt (base::Weight weightFct)
                                        // Get a 'slave' process by evaluating a weight function.
          base::Process *   avail (base::Selection sel=0)
                                        // Get available slaves without blocking (optional: select slave)
```

WaitQ-Synchronisation

Achtung !
keine spezielle Funktion zur Slave-Reaktivierung
→ Verwendung von: activate(), …

Aufrufer-Prozess wird zum Slave,
- wartet auf Master-Prozess, falls keiner momentan verfügbar
- aktiviert den ersten wartenden Master-Prozess
- Rückgabewert liefert Info, ob Aktivierung vom Master (true) oder per Unterbrechung der Wartephase (false)



Aufrufer-Prozess wird zum Master,
- wartet auf Slave-Prozess, blockiert falls keiner momentan verfügbar
- liefert den ersten wartenden Slave-Prozess per Rückgabewert, wenn verfügbar

Aufrufer-Prozess wird weder Master noch Slave
- liefert ersten wartenden Slave, für den die Bedingung gilt (sonst Null-Pointer)

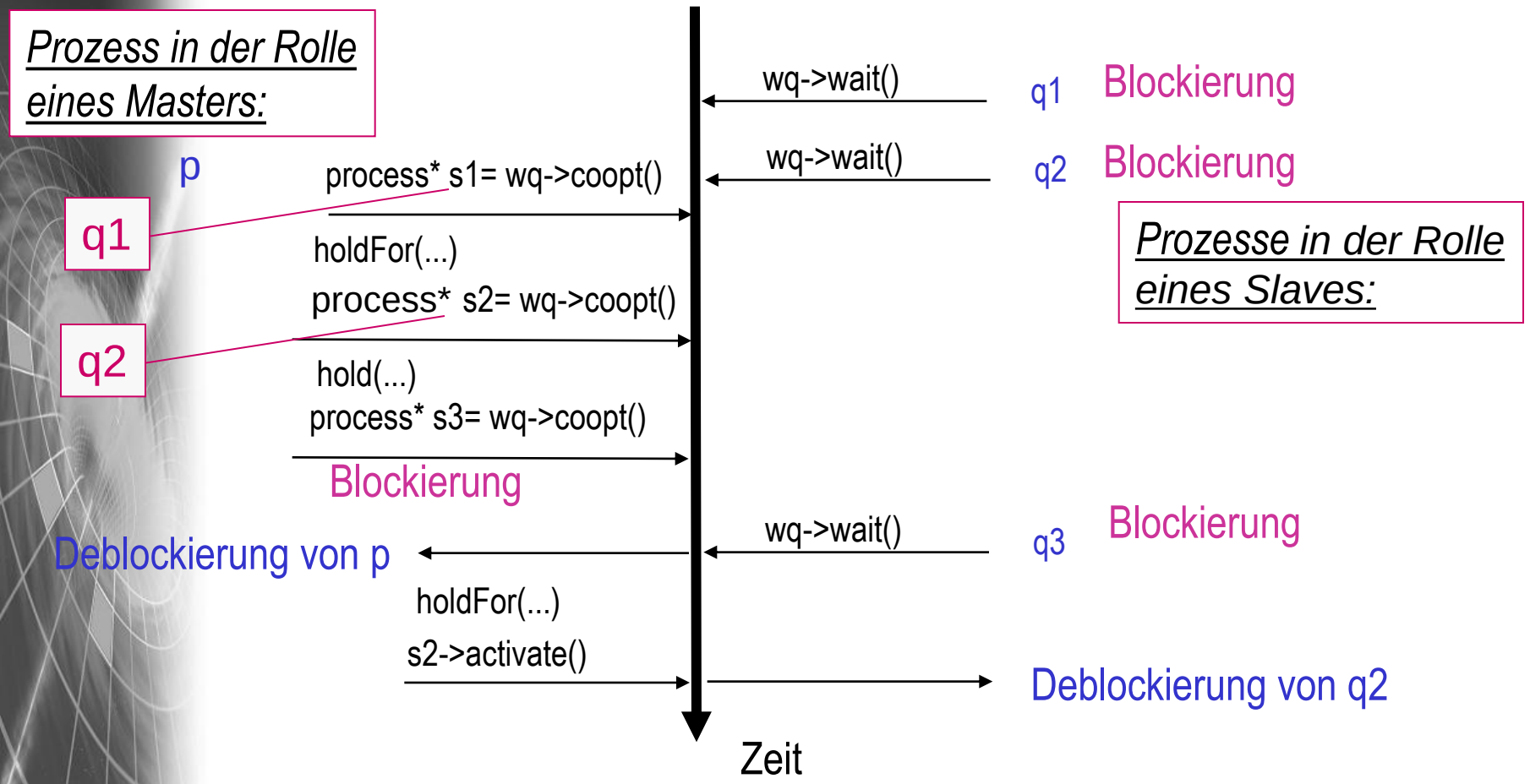
typedef bool (*Selection)(Process*);

bereitzustellen als Member-Funktion einer Process-Ableitung, von der Master-Objekte gebildet werden

Achtung !
Anwendung von: activate() auf einen wartenden Master oder Slave ist FEHLER

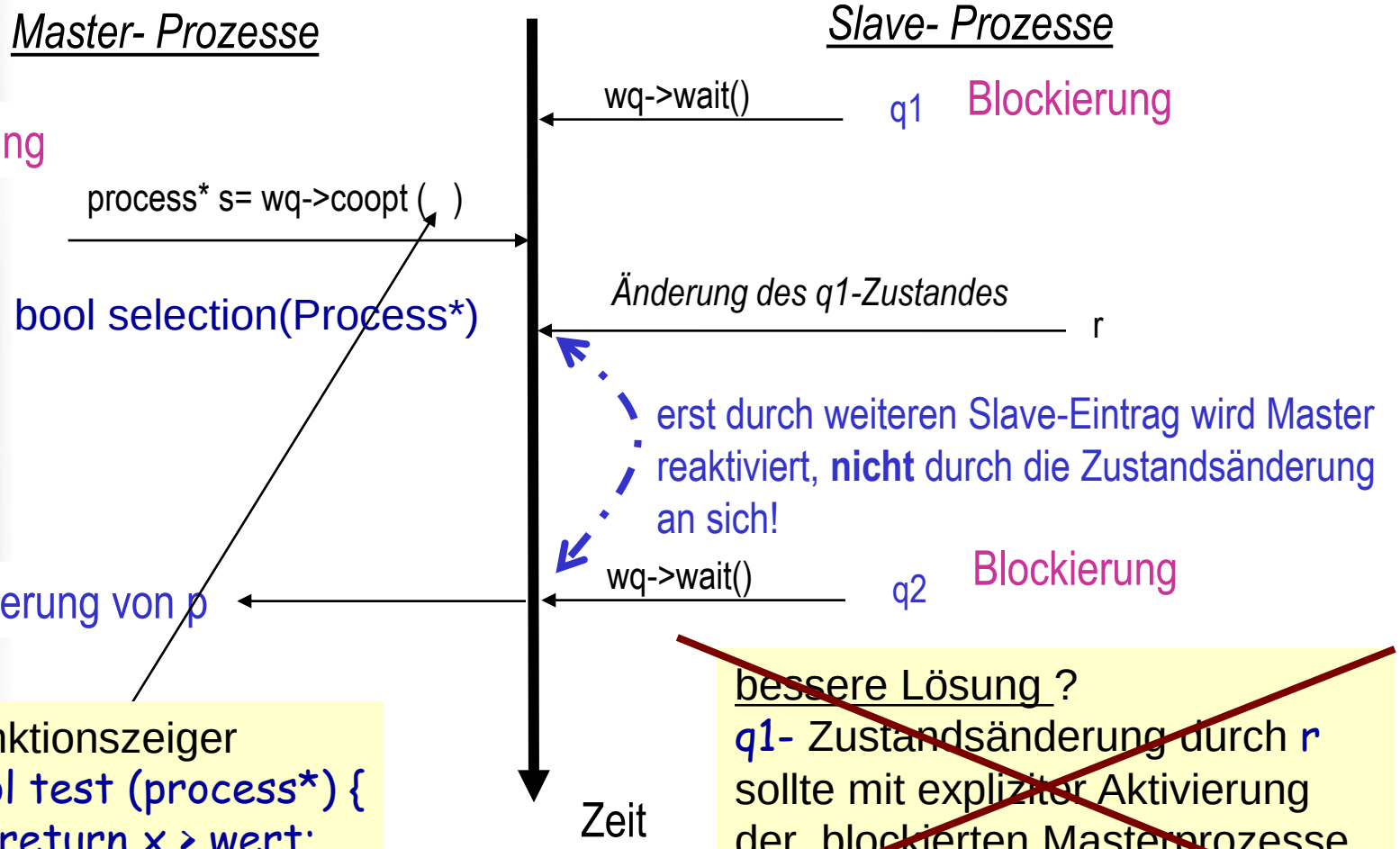
WaitQ-Synchronisation

```
Process *p, *q1, *q2, *q3;      // Zeiger auf Prozessobjekte
WaitQ *wq;
```



WaitQ-Synchronisation

```
Process *p, *q1, *q2, *r;      // Zeiger auf Prozessobjekte
WaitQ *wq;
```



Funktionszeiger

```
bool test (process*) {
    return x > wert;
}
```

bessere Lösung ?

q1- Zustandsänderung durch r sollte mit expliziter Aktivierung der blockierten Masterprozesse in wq verbunden sein

Unterbrechung wartender Master- u. Slave-Prozesse

- das evtl. Warten auf den Partner-Prozess
  kann sowohl beim Master- als auch beim Slave-Prozess mittels
  interrupt abgebrochen werden:

  in diesem Fall liefert

  - coopt() einen NULL-Zeiger

  - wait() den Wert false

  ACHTUNG:
  ein Master sollte jedoch seinen erwählten Slave nicht per interrupt()
  aktivieren !!! (sondern per activate())

  nur dann liefert wait() den Wert true

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

Tanker – Tank – Raffinerie: Wortmodell

1. Durchführung einer 1000h-Simulation logistischer Abläufe eines **Ölhafen**

2. **Tanker** unterschiedlichen Fassungsvermögens (gleichverteilt 15 tb, 20 tb, 25 tb)
   – treffen zufällig im Öl-Hafen ein und
   – können parallel entladen werden

   1 b ≈ 159 l

3. die **Zwischenankunftszeit** der Tanker ist neg.exponential verteilt
   – im Mittel soll alle 8h ein weiterer Tanker eintreffen

4. zur Entladung stehen **5 Tankbehälter** bereit,
   von denen pro Tanker jeweils immer nur einer zugeordnet wird, und zwar der
   – der die längste Zeit leer war
   – dessen freie Kapazität die Schiffsladung komplett übernehmen kann

   steht kein solcher Tank zur Verfügung, muss der Tanker **warten**

5. die **Befüllung** des Tankbehälters (Entladung des Tankers) erfolgt
   mit einer konstanten Pumprate
   – von 1 tb/h;

   zum Anschluss eines Tankers an einen Tank werden
   – 0,5 h **Vorbereitungszeit**

   benötigt.

Wortmodell (Forts.)

6. das maximale **Fassungsvermögen** eines Tanks beträgt 70 tb

7. während der **Befüllung des Tankbehälters** erfolgt keine Entnahme durch die angeschlossene Raffinerie

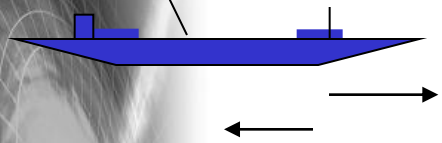
8. die Entnahme von Öl durch die **Raffinerie** erfolgt
   - (nach Abschluss der Befüllung),
     sobald der Tank nur noch 20 tb oder weniger aufnehmen kann
   - mit einer konstanten Pumprate von 4 tb/h;

9. es liegt eine besondere **Ausgangskonfiguration** vor
   - zwei Tanks sind leer
   - einer ist an der Raffinerie angeschlossen und wird in 8h frei
   - zwei werden gefüllt, wobei
     - einer in 3,5h fertig wird mit verbleibender Aufnahmekapazität von 25tb
     - der andere in 12h mit verbleibender Aufnahmekapazität von 45tb
   - der nächste Tanker wird zur Zeit 0 erwartet

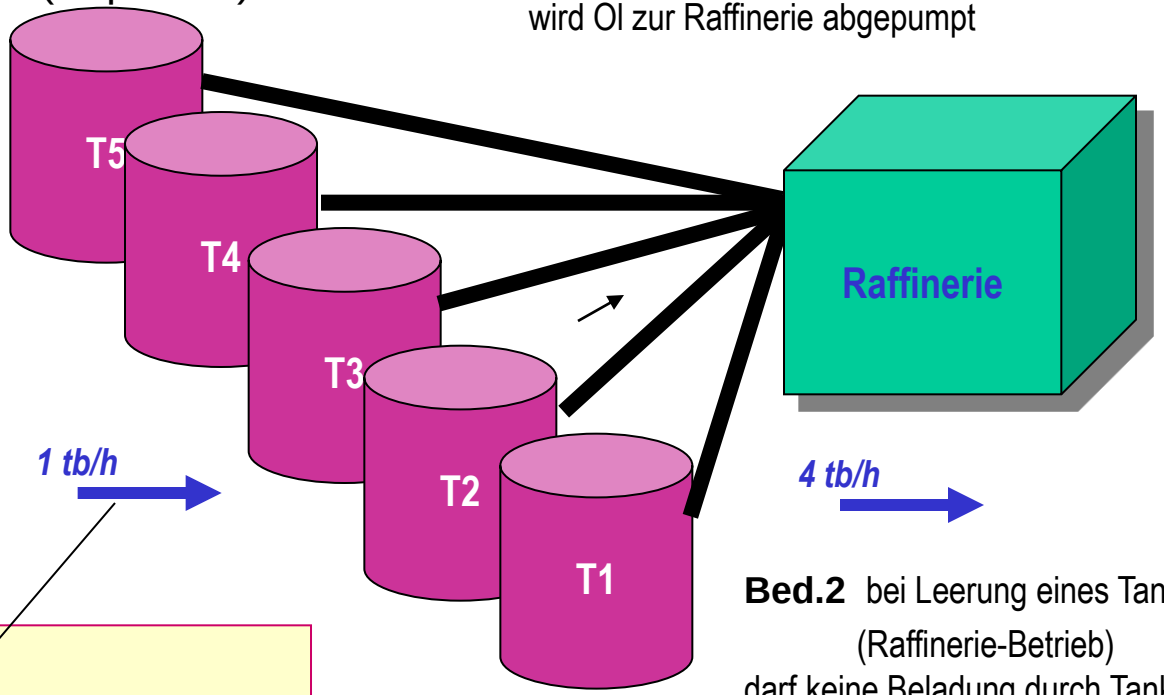
Beispiel: Tanker – Tank – Raffinerie

Zwischenankunftszeit:
n.-exponential verteilt

Ladung: Gleichverteilung
15tb, 20tb, 25tb



(Kap 70 tb)



Bed.1.: ist Tank nahezu voll (≈70%),
wird Öl zur Raffinerie abgepumpt

Bed.2 bei Leerung eines Tanks
(Raffinerie-Betrieb)
darf keine Beladung durch Tanker erfolgen

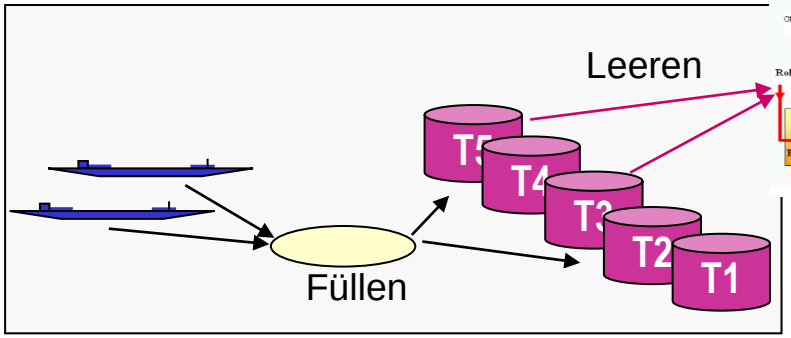
Tankauswahl:
- (1) Tank,
  der die längste Zeit leer war
  und
- (2) dessen frei gebliebene
  Kapazität die Schiffsladung
  komplett übernehmen kann

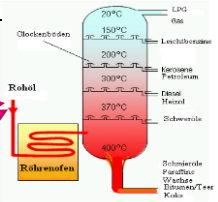
konstante Vorbereitungszeit: 0,5h

Ziel:
1000 h Simulation, bei besonderer Ausgangssituation:
(1) Füllstand der Tankbehälter
- zwei sind leer
- einer wird in 8h leer
- einer wird in 12 h voll (45tb)
- einer wird in 3.5h voll (25tb)

(2) erste Tankerankunft: 0.0

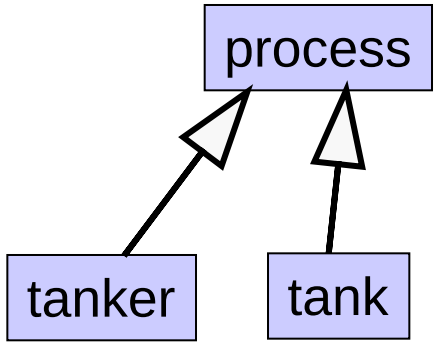
Informales Modell → Simulationsmodell



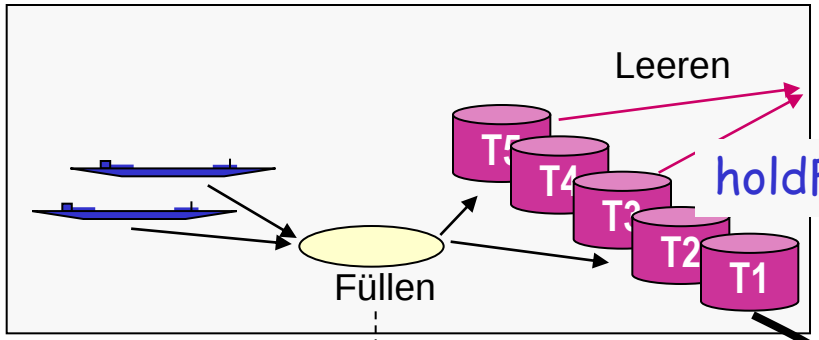


- Kooperationsaktivität: Füllen
  - Tanker und leerer Tank
- Kooperationsaktivität: Leeren
  - „voller“ Tank (und Raffinerie, die aber außerhalb des Systems liegt)
- Master/Slave-Prinzip beim Füllen
  - warum sollten Tanker die Master-Rolle übernehmen ?
  - haben Tank-Objekte nach Kriterien auszuwählen!

    ➔ Anwendung von coopt mit selection-Funtion



Informales Modell → Simulationsmodell



Prozess-terminierung nach Tank-aktivierung

Tank* t= tankQ-> coopt(?)

tankQ: WaitQ
:ProcessList
:ProcessList

tankQ->wait()

beladene Tanker

befüllbare Tankbehälter

Einsortierung
FIFO: Reihenfolge
nach der Wartedauer

Tankauswahl:
- (1) Tank,
  der die längste Zeit leer war
- (2) dessen frei gebliebene Kapazität die Schiffsladung komplett übernehmen kann

```
//Selektionsfunktion als Member-Fkt von Tanker
bool suitableTank (Process* partner) {
      Tank*  selected= dynamic_cast<Tank*> (partner);
      return selected->freeCap >= load;
};
```

freie Tankkapazität

Tankerladung

Umsetzung: Master-Slave-Synchronisation (1)

Tanker-Objekte als Master

```
class Tanker : public Process {
public:
      Tank *myTank; //Tank zur Entladung
      double load; //Fassungsvermoegen[tb]

      Tanker() : Process(sim, "Tanker"),
                        load (5.0*size->sample()){}
protected:
      int main();

//Selektionsfunktion
      bool suitableTank (Process* partner) {
                  ...
      };
};
```

Tank-Objekte als Slave

```
class Tank : public Process {
      double maxCap;    //max. Fassungsverm.
public:
      double freeCap; //akt. Freiraum[tb]

      Tank (double f) : Process(sim, "Tank"),
                  maxCap(70), freeCap(f) {}
protected:
      int main();
};
```

coopt- Implementierung iteriert über die Slave-Prozesse
und wendet auf jedes Element suitableTank() an

Umsetzung: Master-Slave-Synchronisation (2)

Tanker-Objekte als Master

```
int Tanker::main() {
    //Ankunft im Hafen

    //Auswahl des Tanks und Synchronisation
    myTank = dynamic_cast <Tank*>
             (tankq->coopt(
              (Selection) &Tanker::suitableTank));
    //Entladung
    holdFor(setuptime +
                  load*tankerPumpRate);
    //Zeitverbrauch zur Entladung
    myTank->freeCap =
                  myTank->freeCap - load;
    //Beendigung der Kopplung zum Tank
    myTank->holdFor();

    return 0;
}
```

Tank-Objekte als Slave

```
int Tank::main() {
  for (;;) {
    // Warten auf Synchronisation mit Tanker
    // bei anschl. Befuellung des Tanks
    // durch Tanker
    // bis weniger als 20 tb frei sind
    while (freeCap > 20.0) tankq->wait();

    // Entleerung des Tanks
    holdFor( (maxCap - freeCap) *
                              raffPumpRate);
    freeCap = maxCap;
  }
  return 0;
}
```

Umsetzung: Startsituation

Ziel:
1000 h Simulation, bei **besonderer Startsituation**:

(1) Füllstand der Tankbehälter
- zwei sind leer
- einer wird in 8h leer
- einer wird in 12 h voll (45tb)
- einer wird in 3.5h voll (25tb)

(2) erste Tankerankunft: 0.0

```
Tank *t;
    t = new Tank(70.0); t->hold();
    t = new Tank(70.0); t->hold();
    t = new Tank(45.0); t->holdUntil(12.0);
    t = new Tank(25.0); t->holdUntil(3.5);
    t = new Tank(70.0); t->holdUntil(8.0);
```

Aktionen können entweder
- im Hauptprogramm vor Start des Simulationskontextes
  oder
- von einem Konfigurationsprozess übernommen werden,
  der wiederum vom Hauptprogramm zur Zeit 0 in den Ereigniskalender aufzunehmen ist

Report-File

Random Number Generators

Name	Reset at	Type	Uses	Seed	Parameter 1	Parameter 2	Parameter 3
nextTanker	0	Negexp	128	33427485	0.125	0	0
size	0	Randint	129	22276755	3	5	0

Queue Statistics

Name	Reset at	Min queue length	Max queue length	Now queue length	Avg queue length
shoreTanks_master_queue	0	0	5	0	0.184572
shoreTanks_slave_queue	0	0	5	2	1.82777

Waitq Statistics

Name	Reset at	Master Queue	Slave Queue	Number of Synch.	Zero wait masters	Avg masters wait	Zero wait slaves	Avg slaves wait
shoreTanks	0	shoreTanks_master_queue	shoreTanks_slave_queue	128	102	1.46019	27	14.2281